百度智能云

FACE 文档

目录

目录 2
文档导览 7
购买指南 9
免费测试资源 9
产品价格 12
如何购买 27
API文档 34
概述 34
通用说明 35
人脸识别基础接口 36
人脸检测与属性分析 36
人脸1：1对比 47
人脸1：N搜索 54
人脸M：N搜索 58
人脸库管理系列接口 63
在线图片活体V3 85
视频活体检测 93
场景化搜索（公测） 106
REST-API-SDK 128
实名认证相关接口 393
人脸实名认证V3 393
身份证与名字比对 398
人脸实名认证（含有效期核验） 400
身份证与名字比对（含有效期核验） 404
人证核验（含证件状态） 406
人像特效相关文档 408
人脸融合 408
人脸属性编辑 413
人脸关键点检测 417
错误码说明 454
常见问题及排查 466
人脸实名认证方案 468
方案介绍 468
价格说明 470
人脸实名认证方案-APP端 473
方案简介 473
方案集成前准备 476
Android-方案集成指南 483
IOS-方案集成指南 506
HarmonyOS-方案集成指南 539

SDK合规说明文档 560
人脸实名认证方案-H5端（含小程序） 564
方案简介 564
方案接入指南 568
H5人脸实名认证方案配套接口 577
APP内嵌H5配置说明 599
服务端接入 604
服务端接入-方案简介 604
方案简介 604
Android-服务端接入指南 606
iOS-服务端接入指南 611
HarmonyOS-服务端接入指南 625
人脸实名认证V4 629
人脸对比V4 635
在线图片活体V4 642
常见问题 653
错误码 656
人脸意愿核身方案 663
方案简介 664
价格说明 664
方案接入指南 666
意愿核身方案配套接口 674
私有化部署 694
产品介绍 694
整体介绍 694
使用前须知 698
公有云转私有化 699
快速开始 701
更新记录 703
接口文档 704
3.错误码对照表 743
错误码 747
常见问题及排查 757
总体说明 757
性能及环境问题 758
鉴权失败问题 761
安装部署问题 765
接口调用问题 771
其他常见问题 774
购买指南 777
价格说明 777

硬件配置及推荐 777
部署运维 779
部署说明 779
运维支持 794
人脸离线识别SDK 808
概览 808
功能介绍 820
激活授权 823
硬件选型 825
应用策略 827
产品定价 831
Android-SDK 832
WIN-C++-SDK 950
Linux-ARM-SDK 990
人脸识别特征值同步接口 1023
Android-SDK-常见问题 1031
Windows-SDK-常见问题 1061
历史版本 1071
API文档-v2 1072
人脸检测 1072
人脸对比 1078
人脸查找 1081
人脸库管理 1089
身份验证 1102
在线活体检测 1106
H5活体检测 1113
错误码 1117
REST API SDK-v2 1120
JAVA语言 1120
PHP语言 1122
Python语言 1150
C#语言 1179
C++语言 1209
Nodejs语言 1237
离线采集SDK 1268
Android SDK4.0 1268
IOS SDK4.0 1287
V3.3.0.0版本 1303
4、接口设计说明# 1339
1.3 开发包说明## 1360

windows版本 1376
Linux版本 1389
安全加固采集SDK升级文档 1398
安全加固采集SDK接入文档 1405
增强级SDK 1443
4、接口设计说明# 1459
金融级SDK 1481
离线识别SDK 1518
Android-SDK-v2.0说明 1518
Android-人脸通行工程 1537
Android-人证核验工程 1563
Android-广告屏分析工程 1590
Win-SDK-C++（历史版本） 1603
Win-SDK-C#（历史版本） 1640
Linux-X86-SDK（历史版本） 1675
人脸注册工具平台 1691
人脸注册工具平台 1691
人脸产品套件-壁虎 1693
产品和硬件介绍 1693
接口定义说明 1700
实名认证 1707
增强级人脸实名认证 1707
金融级人脸实名认证 1713
增强级人脸比对 1719
金融级人脸比对 1723
活体检测 1729
公有云接口 1747
标准级APP端实名认证方案 1758
增强级APP端实名认证方案 1782
金融级APP实名认证方案 1807
标准级H5端实名认证方案 1845
增强级H5端实名认证方案 1862
金融级H5实名认证方案 1879
人脸应用套件 1897
产品概述 1897
软件说明文档 1899
硬件说明文档 1906
资料下载 1915
公有云采集SDK 4.1.5 1916
概览 1916
常见问题 1936

Android-SDK4.1.5 1941

4.接口设计说明 1958

IOS-SDK4.1.5 1963

SDK合规使用指南 1977

文档导览

欢迎使用百度人脸识别服务。

本文档主要针对初次接触百度人脸识别的开发者，为了让您能够更方便快速的找到产品说明，我们准备了一份导览文档，请您查看

产品定价与购买

- 产品价格：文档
- 付费购买：文档

在线API接入指南

- 接入说明：文档

在线API接口文档

- 人脸检测，获得眼、口、鼻轮廓，识别多种人脸属性，如性别，年龄，表情等信息：API文档
- 人脸对比，对比两张人脸的相似度并返回评分：API文档
- 人脸搜索，在一个指定人脸库中查找相似的人脸：API文档
- 人脸库管理，是人脸搜索的前提：API文档
- 活体检测：抵御照片、视频、模具等作弊攻击，保障业务安全

  (1)在线活体检测（单张图片的活体检测）：API文档

  (2)H5视频活体检测（针对视频的活体检测）：API文档
- 身份验证：验证用户的身份证号码和姓名以及现场拍摄的图片与公安数据源是否匹配，用于判断用户信息是否真实。 API文档
- 人脸融合：对两张人脸进行融合处理，生成的人脸同时具备两张人脸的外貌特征，API文档
- 错误码：文档

在线接口的Http SDK

- 在线接口的Http SDK说明文档（包含java、PHP、Python、C#、C++、Nodejs六种语言），是帮助您快速调用在线API接口的工具包 SDK文档

  SDK下载

离线采集SDK

离线调用人脸检测、人脸追踪、人脸采集等能力，**从视频流中快速获取人脸图片并确保获取的人脸图片质量**，配合API接口，高效构建各场景人脸识别应用。我们提供了Android、IOS、Windows、Linux四个版本的SDK，提供如下核心能力：

- 人脸检测、人脸关键点采集
- 人脸追踪、动态定位人脸
- 动作配合式活体检测

说明文档

SDK下载

离线识别SDK

人脸离线识别SDK，包含人脸采集、活体检测、人脸对比/识别、人脸库管理等能力，并全部离线化、本地化。

可以集成到单台硬件设备中，实现人脸检测、人脸追踪、人脸采集、人脸比对、小型人脸库（3万人以内）的快速查找。

说明文档

SDK下载

> 此产品适用于小人脸库、低并发场景。

私有化部署方案

私有化方案是能够在本地部署、离线使用的模型服务，离线部署的模型可以实现在线API的全部功能。

部署至客户本地服务器，实现图片中的人脸检测、关键点和属性识别、人脸比对、大型人脸库的快速查找，支持分布式高并发业务处理，适用于安防、监控等场景

立即使用

产品整体介绍

硬件推荐及配置

部署说明

接口说明

> 此产品适用于大人脸库、高并发场景。

场景解决方案

人脸实名认证

应用活体检测、证件识别、人脸对比等多种技术能力，捕获当前用户照片并与公民身份照片进行比对，实现身份验证。此方案能够帮助您识别业务场景中的用户，是否为「真人」且为「本人」，从而更加快捷地完成身份核实工作，提高业务处理效率，减少人工成本。

- 公安验证：API文档
- **活体检测：百度提供了6种形式的活体检测服务，为您的产品保驾护航**

（1）动作配合式活体：SDK。申请使用；SDK文档

（2）在线图片活体：API。立即使用；API文档

（3）在线H5视频活体：API。立即使用；API文档

（4）离线RGB活体：SDK。申请使用；SDK文档

（5）离线近红外活体：SDK。申请使用；SDK文档

（6）离线3D结构光活体：SDK。申请使用；SDK文档

人脸考勤

面向移动考勤、办公室前台考勤、摄像机无感知考勤的应用场景，提供有针对性的人脸考勤解决方案、完善的集成说明，帮助开发者便捷地搭建一个高效率，高防作弊能力的智能考勤系统。

方案介绍文档

人脸登录

使用人脸比对、人脸搜索功能，实现在互联网APP、线下酒店的刷脸登录。

方案介绍文档

人脸闸机

针对人员出入的闸机及门禁场景，面向不同行业领域，提供行业定制的全流程解决方案，包括景区、楼宇的软硬件+管理系统的组合打包方案，及适配多种硬件环境的API+SDK基础开发组件方案。

方案介绍文档

摄像头选型建议

使用人脸识别服务的前提是获取用户的人脸图片，在线下的零售会员识别、安防监控等场景应该选择什么样的摄像头来采集人脸图片、以及用什么样的角度部署，可查看如下文档：

说明文档

人脸采集注意事项

人脸识别或对比的最终效果，取决于采集到的人脸**是否符合质量要求**。从业务使用角度，主要影响两个核心业务步骤：

1. **人脸注册环节**：如果注册的人脸质量不佳，则会影响注册环节的特征抽取，导致原始注册的人脸信息较差，后面的识别/对比都会受到直接的影响，往往得到的相似度分值，将不会特别准确。
2. **人脸识别/对比环节**：因为注册人脸质量不佳，每次的识别/对比都会存在一定的分值误差，往往造成明明是本人却过不去的情况。

采集阶段应该有哪些注意事项呢，请查看如下文档：

人脸采集完整说明:说明文档

人脸采集阶段**人脸图片质量控制**说明：说明文档

开发工具

百度提供了一些开发工具，减少开发者使用人脸识别服务的开发成本，让人脸识别服务更简单：

- 人脸注册工具平台

  通过可视化操作，快速创建采集用户人脸图片的H5、小程序页面—产品说明文档

购买指南

免费测试资源

免费测试资源为**自动发放**，在您完成实名认证后，首次进入控制台即自动领取。领取到的资源会在10分钟内发放至账户。人脸方向免费资源领取需完成个人或企业实名认证，企业认证可领取更多测试资源，了解个人/企业认证。

1、公有云 API

1.1 基础人脸服务接口

基础服务说明：

> 人脸库管理系列接口，包含人脸注册、人脸更新、人脸删除等子接口，每个接口并发独立。
> 业务用量超过免费测试量，需前往控制台采购，具体价格查看 产品价格文档。

API接口	个人认证	企业认证	有效期
在线图片活体v3	调用量：500次/月 QPS： 2QPS	调用量：1千次/月 QPS： 10QPS	-
人脸检测	调用量：1千次/月 QPS： 1QPS	调用量：1万次/月 QPS： 2QPS	-
人脸对比	调用量：1千次/月 QPS： 1QPS	调用量：1万次/月 QPS： 2QPS	-
人脸搜索	调用量：1千次/月 QPS： 1QPS	调用量：1万次/月 QPS： 2QPS	-
人脸搜索M：N识别	调用量：1千次/月 QPS： 1QPS	调用量：1万次/月 QPS： 2QPS	-
人脸库管理-人脸注册	调用量：1千次/月 QPS： 1QPS	调用量：1万次/月 QPS： 2QPS	-
人脸库管理-人脸更新	调用量：1千次/月 QPS： 1QPS	调用量：1万次/月 QPS： 2QPS	-
人脸库管理-删除用户 人脸库管理-用户信息查询 人脸库管理-获取组列表 人脸库管理-获取用户列表 人脸库管理-复制用户 人脸库管理-获取用户人脸列表 人脸库管理-创建用户组 人脸库管理-删除用户组 人脸库管理-删除人脸	调用量：不限量 QPS： 2QPS	调用量：不限量 QPS： 10QPS	-

邀测接口说明

> 当前场景化搜索服务属于公测状态，如需更多免费资源，可以通过 提交工单 申请

接口	个人认证免费测试量	企业认证免费测试量	有效期
人脸搜索（视频监控）	调用量：500次，QPS：2QPS	调用量：1000次，QPS：10QPS	1年有效期
人脸库管理（场景化）-人脸注册	调用量：500次，QPS：2QPS	调用量：1000次，QPS：10QPS	1年有效期
人脸库管理（场景化）-人脸更新	调用量：500次，QPS：2QPS	调用量：1000次，QPS：10QPS	1年有效期
人脸库管理（场景化）-创建用户组	调用量：不限量，QPS：10QPS	调用量：不限量，QPS：10QPS	-
人脸库管理（场景化）-用户信息查询	调用量：不限量，QPS：10QPS	调用量：不限量，QPS：10QPS	-
人脸库管理（场景化）-获取用户列表	调用量：不限量，QPS：10QPS	调用量：不限量，QPS：10QPS	-
人脸库管理（场景化）-复制用户	调用量：不限量，QPS：10QPS	调用量：不限量，QPS：10QPS	-
人脸库管理（场景化）-删除用户	调用量：不限量，QPS：10QPS	调用量：不限量，QPS：10QPS	-
人脸库管理（场景化）-删除用户组	调用量：不限量，QPS：10QPS	调用量：不限量，QPS：10QPS	-
人脸库管理（场景化）-获取用户组列表	调用量：不限量，QPS：10QPS	调用量：不限量，QPS：10QPS	-

1.2 实名认证

接口	个人认证免费测试量	企业认证免费测试量	有效期
视频活体检测	调用量：总量500次 QPS： 2QPS	调用量：总量1000次 QPS： 10QPS（扩容购买）	1年有效期
随机校验码(支持读数字/动作)	调用量：总量500次 QPS： 2QPS	调用量：总量1000次 QPS： 10QPS（扩容购买）	1年有效期
在线图片活体V4	暂无，需企业认证后领取免费测试量	调用量：总量500次 QPS： 10QPS（扩容购买）	1年有效期
身份证与名字对比	企业认证	调用量：总量500次 QPS： 2QPS 完成企业认证后，请进入百度智能云-人脸识别后台，约在1个小时左右发放	-
人脸实名认证V4	企业认证	调用量：总量500次 QPS： 2QPS 完成企业认证后，请进入百度智能云-人脸识别后台，约在1个小时左右发放，**如您之前已领取人脸实名认证V3系列免费额度，则不再重复发放，如需测试V4系列**，请提交工单	1年有效期
人脸对比V4	企业认证	调用量：总量500次 QPS： 2QPS（扩容购买） 完成企业认证后，请进入百度智能云-人脸识别后台，约在1个小时左右发放	1年有效期

> 1、完成百度云企业认证后,请进入百度智能云-人脸识别后台，约在1个小时左右自动开通实名认证接口，请耐心等待。
> 2、人脸实名认证V3系列API的免费额度查询及购买，请移步服务列表-历史版本。

1.3 人像特效

接口	个人认证免费测试量	企业认证免费测试量	有效期
人脸融合	调用量：总量500次 QPS： 2QPS	调用量：总量500次 QPS： 2QPS	1年有效期
人脸属性编辑	调用量：总量500次 QPS： 2QPS	调用量：总量500次 QPS： 2QPS	1年有效期
人脸关键点	调用量：总量500次 QPS： 2QPS	调用量：总量500次 QPS： 2QPS	1年有效期

> 人像特效API QPS支持认证后赠送2QPS，开通付费后，除虚拟换妆API外，其余API并发支持将扩充至 10QPS。如需更多QPS，可以通过提交工单与我们联系

2、SDK

SDK	免费测试量
人脸离线采集 SDK	手机端SDK需与API搭配使用，您可以创建实名认证方案申请
人脸离线识别 SDK	每个账号下默认分配 **5个** 试用版序列号，有效期为**三个月**，供前期测试。如需正式使用，支持 购买授权

3、私有化部署包

申请私有化部署包，即可获得1个月免费测试有效期

产品价格

1、公有云 API

1.1 基础服务

基础服务 API ： 人脸检测、人脸对比 、人脸搜索与库管理、人脸识别特征值同步接口 ，其中**人脸搜索与库管理**包含多个子接口。

账户完成个人认证，可领取 **1QPS**，1千次/月的免费测试资源 ；完成企业认证，可领取 **2QPS**，1万次/月的免费测试资源。领取入口为 控制台-免费资源领取页。

如果免费测试资源无法满足您的测试需求，您可以随时购买扩充QPS ，QPS 可**包月购买**，也可**按天购买**，灵活多样，适应多场景需求。具体价格如下：

购买 QPS 数量	按月购买	按天购买
0<QPS<=10	270元/月/QPS	16元/天/QPS
10<QPS<=50	235元/月/QPS	14元/天/QPS
50<QPS<=100	170元/月/QPS	12元/天/QPS
100<QPS	135元/月/QPS	10元/天/QPS

> 说明：
>
> 1. 同一个百度云账号下**多个应用共享各接口的并发支持**。
> 2. 阶梯价格梯度按照单次购买的数量计算，不同订单不累计。
> 3. 每个接口单独计算QPS，若您需要人脸注册和人脸比对接口各10个QPS，则您需要分别购买8个QPS的人脸注册和人脸比对接口（企业认证用户免费赠送2QPS）。
> 4. 每个接口最多只可购买200个QPS，如需更多QPS，请 提交工单 与我们联系。

调用计费示例

- 如果您需要50个QPS，除免费赠送的2QPS外（需要企业认证），您还需购买48个QPS。则计费方式为：`270*10+235*38=11630` 元/月
- 如果您需要购买1个月+5天的1个QPS，可以按照包月+按天的搭配模式购买，即：`270*1+16*5=350` 元

****人脸识别V3**** 百度智能云将于2022年11月24日起，对部分人脸识别公有云接口新增“按量后付费”与“资源包”购买方式，并于12月1日起下调“按QPS付费”与“QPS叠加包”价格，包括：人脸检测、人脸对比(V3) 、人脸搜索、人脸搜索 M：N识别、人脸注册、人脸更新，详情可参考官网公告。

> - **“按量后付费”与“资源包”购买方式将于2022年11月24日起正式上线**，您可登录控制台进行购买，如有相关问题，可随

时**联系您的商务经理**，或提交工单进行咨询。

- 完成个人认证后，可领取1千次/月的免费测试额度，测试并发数为1QPS；完成企业认证后，可领取1万次/月的免费测试额度，测试并发数为2QPS。免费测试额度耗尽之后，如需继续使用，您可按需选择如下几种购买方式中的一种进行购买。
- 以”按量后付费“、”资源包“二种购买方式中的任一方式开通付费后，接口并发都会默认免费提升至10QPS。
- 以“按QPS付费”的购买方式开通付费后，接口调用量无限制。接口并发总量=默认赠送的QPS+购买的QPS，例如：企业认证的账号默认享有2QPS的免费并发，购买10QPS后，此账号享有的并发量为12QPS。

按量后付费

说明：
开通按量后付费后，系统将自动调整至10QPS，并按以下价格按调用量进行计费。

月调用量N（次）	0 < N <= 100万	100万 < N ≤ 500万	500万 < N ≤ 1000万	1000万 < N ≤ 5000万	5000万 < N≤ 1亿	1亿 < N
人脸检测	0.0004元	0.00038元	0.00035元	0.0003元	0.00028元	0.00025元
人脸对比	0.003元	0.00285元	0.0027元	0.00255元	0.0024元	0.0021元
人脸搜索	0.003元	0.00285元	0.0027元	0.00255元	0.0024元	0.0021元
人脸搜索M：N识别	0.003元	0.00285元	0.0027元	0.00255元	0.0024元	0.0021元
人脸注册	0.0004元	0.00038元	0.00035元	0.0003元	0.00028元	0.00025元
人脸更新	0.0004元	0.00038元	0.00035元	0.0003元	0.00028元	0.00025元

预付费资源包

说明
1. 预付费资源包有效期为1年，支持叠加购买，按照购买时间顺序依次抵扣，全部使用完毕后自动切换为按量后付费模式。次数包购买时会自动开通**按量后付费，请勿在消耗次数包时终止后付费，避免影响正常使用**。次数包超出有效期未抵扣额度自动失效，无法继续使用，请根据实际业务需求酌情购买。
2.预付费资源包购买后**7天内未产生调用**可提交工单申请退款，**超过7天**或**已产生调用的次数包**无法退款。

资源包规格	10万次	100万次	500万次	1000万次	5000万次	1亿次
人脸检测	38元	360元	1700元	3200元	15000元	24000元
人脸对比	48元	470元	2300元	4400元	21500元	42000元
人脸搜索	48元	470元	2300元	4400元	21500元	42000元
人脸搜索M：N识别	48元	470元	2300元	4400元	21500元	42000元
人脸注册	38元	360元	1700元	3200元	15000元	24000元
人脸更新	38元	360元	1700元	3200元	15000元	24000元

按QPS付费

> **说明：**
> 1.完成个人认证后，默认并发数为1QPS；完成企业认证后，默认并发数为2QPS。
> 2.购买**按QPS付费**后，享有无限调用量。并发总数=默认赠送的QPS+购买的QPS。
> 例如：您需要的并发总量为50QPS，那么除免费赠送的2QPS外（需完成企业认证），您还需购买48个QPS。

购买 QPS 数量	按月购买	按天购买
0<QPS<=10	270元/月/QPS	16元/天/QPS
10<QPS<=50	235元/月/QPS	14元/天/QPS
50<QPS<=100	170元/月/QPS	12元/天/QPS
100<QPS	135元/月/QPS	10元/天/QPS

QPS叠加包

> **说明：**
> 1.完成个人认证后，默认并发数为1QPS；完成企业认证后，默认并发数为2QPS。
> 2.QPS叠加包必须与**“按量后付费”或“预付费次数包”**两种付费方式之一配合使用。当产品支持QPS无法满足业务需求时，可购买QPS叠加包进行并发扩容，适用于业务用量增加需临时扩容的情况。

QPS叠加包	按月购买	按天购买
0<QPS<=10	270元/月/QPS	16元/天/QPS
10<QPS<=50	235元/月/QPS	14元/天/QPS
50<QPS<=100	170元/月/QPS	12元/天/QPS
100<QPS	135元/月/QPS	10元/天/QPS

1.2 活体检测

活体验证 API ： 在线图片活体（V3）、视频活体检测 与 随机校验码 、在线图片活体（V4），具体价格如下：

1.2.1 在线图片活体(V3)****

个人认证免费测试量可领取500次，企业认证免费测试量可领取1000次。免费测试量耗尽之后，如需继续使用，您可按需选择如下两种购买方式中的一种进行购买。
如需从按量后付费切换至按QPS预付费，您可在控制台自助点击停止按量后付费；如需从按QPS预付费切换只按量后付费，您需等待所购QPS失效或完成退款后操作。

（1）按量后付费

开通按量后付费期间，QPS为10，按如下价格计费。

接口	产品价格	计费错误码
在线图片活体(V3)	0.002元/次	无

（2）QPS叠加包

当开通“按量后付费”后，如默认支持的10QPS不满足业务需求时，可购买QPS叠加包提升并发量。

QPS 数量	按月购买	按天购买
0<QPS<=10	270元/月/QPS	16元/天/QPS
10<QPS<=50	235元/月/QPS	14元/天/QPS
50<QPS<=100	170元/月/QPS	12元/天/QPS
100<QPS	135元/月/QPS	10元/天/QPS

（3）按QPS预付费

按QPS资源购买，QPS有效期内，调用量无限。

购买 QPS 数量	按月购买	按天购买
0<QPS<=10	270元/月/QPS	16元/天/QPS
10<QPS<=50	235元/月/QPS	14元/天/QPS
50<QPS<=100	170元/月/QPS	12元/天/QPS
100<QPS	135元/月/QPS	10元/天/QPS

1.2.2 视频活体检测与随机校验码****

视频活体检测

（1）按量后付费

> 开通按量后付费期间，QPS为10，按如下价格计费。 计费调用量包含**成功调用**及**计费错误码 223120、216500** ,可参考错误码文档

月调用量（次）	价格（元/次）
0 < n <= 10,000	0.2
10,000 < n <= 100,000	0.18
100,000 < n <=500,000	0.16
500,000 < n <=1000,000	0.15
1000,000 < n	0.14

（2）QPS叠加包

> 个人认证赠送2QPS ，完成百度云企业认证，并开通付费，并发将扩充至 10QPS。如需更多QPS，可购买QPS叠加包提升并发量。

购买 QPS叠加包数量	按月购买	按天购买
0<QPS<=10	270元/月/QPS	16元/天/QPS
10<QPS<=50	235元/月/QPS	14元/天/QPS
50<QPS<=100	170元/月/QPS	12元/天/QPS
100<QPS	135元/月/QPS	10元/天/QPS

随机校验码

（1）按量后付费

> **视频活体检测**计费调用量包含**成功调用**及**需计费的失败调用223120、216500**。错误码说明请参考错误码文档

月调用量（次）	价格（元/次）
0 < n <= 10,000	0.010
10,000 < n <= 100,000	0.009
100,000 < n <=500,000	0.008
500,000 < n <=1，000,000	0.007
1,000,000 < n	0.006

（2）QPS叠加包

> 个人认证赠送2QPS ，完成百度云企业认证，并开通付费，并发将扩充至 10QPS。如需更多QPS，可购买QPS叠加包提升并发量。

购买 QPS叠加包数量	按月购买	按天购买
0<QPS<=10	270元/月/QPS	16元/天/QPS
10<QPS<=50	235元/月/QPS	14元/天/QPS
50<QPS<=100	170元/月/QPS	12元/天/QPS
100<QPS	135元/月/QPS	10元/天/QPS

1.2.3 在线图片活体（V4)**** 企业认证账号可领取500次免费测试量，耗尽之后开始计费。

（1）按量后付费

开通按量后付费期间，QPS为10，按如下价格计费。

月调用量（次）	价格（元/次）
0 < n <= 10,000	0.100
10,000 < n <= 100,000	0.090
100,000 < n <=500,000	0.075
500,000 < n	0.060

（2）预付费资源包

> 1. 预付费包年次数包有效期为1年，支持叠加购买，按照购买时间顺序依次抵扣，全部使用完毕后自动切换为按量后付费模式。次数包购买时会自动开通**按量后付费**，**请勿在消耗次数包时终止后付费，避免影响正常使用**。次数包超出有效期未抵扣额度自动失效，无法继续使用，请根据实际业务需求酌情购买。
> 2. 预付费包年次数包购买后**7天内未产生调用**，如有需要，您可对购买的QPS进行自助退款，您可前往百度智能云‘控制台--财 务--退订管理’页面进行自助退订，详细退订规则见退订说明， **超过7天**或**已产生调用的次数包**无法退款。

次数包	价格	有效期
10,000 次	980 元	一年
100,000 次	9,500 元	一年
500,000 次	45,000 元	一年
1,000,000 次	80,000 元	一年
3,000,000 次	210,000 元	一年
5,000,000 次	300,000 元	一年

（3）QPS叠加包

> 完成企业认证后接口并发支持默认赠送2QPS，开通付费后，并发支持将扩充到10QPS，如需更多QPS，可购买QPS叠加包。

购买 QPS 数量	按月购买	按天购买
0<QPS<=10	270元/月/QPS	16元/天/QPS
10<QPS<=50	235元/月/QPS	14元/天/QPS
50<QPS<=100	170元/月/QPS	12元/天/QPS
100<QPS	135元/月/QPS	10元/天/QPS

1.3 身份验证

身份验证 API ：身份证与名字比对 、 人脸实名认证（V3）、人脸比对（V4）、人脸实名认证（V4）

****1.3.1 身份证与名字比对****

> **身份证与名字比对**接口需要开通企业认证后方可使用。完成企业认证后，企业认证用户可领取**500次免费测试量，免费测试量用尽后开始计费。**开通付费/购买次数包
>
> 用户完成企业认证后接口并发支持默认赠送**2QPS**，开通付费后，并发支持将扩充到**10QPS**，如需更多QPS，可以通过提交工单与我们联系。

（1）按量后付费

接口	产品价格
身份证与名字比对	**0.3 元/次**

（2）预付费资源包

次数包	价格	有效期
1000 次	290 元	一年
10,000 次	2,800 元	一年
50,000 次	13,000 元	一年
100,000 次	24,000 元	一年
500,000 次	105,000 元	一年
1,000,000 次	200,000 元	一年
2,000,000 次	360,000 元	一年

> **说明**
> 1、计费调用量包含**成功调用及计费错误码 222351** ,错误码详情说明可参考错误码文档
> 2、预付费包年次数包有效期为1年，支持叠加购买，按照购买时间顺序依次抵扣，全部使用完毕后自动切换为按量后付费模式。次数包购买时会自动开通**按量后付费，请勿在消耗次数包时终止后付费，避免影响正常使用**。次数包超出有效期未抵扣额度自动失效，无法继续使用，请根据实际业务需求酌情购买。
> 3、预付费包年次数包购买后**7天内未产生调用**，如有需要，您可对购买的QPS进行自助退款，您可前往百度智能云'控制台--财务--退订管理'页面进行自助退订，详细退订规则见退订说明， **超过7天或已产生调用的次数包**无法退款。

1.3.2 身份证与名字比对（含有效期核验）

> **身份证与名字比对（含有效期核验）**接口需要开通企业认证后方可使用。完成企业认证后，如需进行接口测试，**请联系商务同学申请测试额度**。开通付费/购买次数包
> 接口开通付费后，并发支持将扩充到**10QPS**，如需更多QPS，可以通过提交工单与我们联系。

（1）按量后付费

分段阶梯计费：到达相应阶梯的计费调用量按照所在阶梯单价进行计费。如1月份内接口调用量为15000次，则费用为 ：10000 x 1.5 + 5000 x 1.425 = 22125 元.

月调用量（次）	价格（元/次）
0 < n <= 10,000	1.5
10,000 < n <= 100,000	1.425
100,000 < n	1.35

（2）预付费资源包

次数包	价格	有效期
10,000 次	14,250 元	一年
50,000 次	67,500 元	一年
100,000 次	127,500 元	一年

> **说明**
> 1、预付费包年次数包有效期为1年，支持叠加购买，按照购买时间顺序依次抵扣，全部使用完毕后自动切换为按量后付费模式。次数包购买时会自动开通**按量后付费，请勿在消耗次数包时终止后付费，避免影响正常使用**。次数包超出有效期未抵扣额度自动失效，无法继续使用，请根据实际业务需求酌情购买。

> 2、预付费包年次数包购买后**7天内未产生调用**，如有需要，您可对购买的QPS进行自助退款，您可前往百度智能云'控制台--财务--退订管理'页面进行自助退订，详细退订规则见退订说明， **超过7天**或**已产生调用的次数包**无法退款。

1.3.3 人脸比对（V4)

> **温馨提示：**
> 人脸比对（V4) 接口**需要开通企业认证**才能使用。
> （1）开通企业认证后累计500次免费调用，免费额度用尽后开始计费。为避免影响您的业务使用，请及时在概览页开通后付费。
> （2）开通企业认证后接口赠送2QPS，开通付费后，并发支持将扩充到10QPS，如需更多QPS，可直接在概览页付费开通提升更多QPS。

（1）按量后付费

分段阶梯计费：到达相应阶梯的计费调用量按照所在阶梯单价进行计费。如1月份内接口调用量为15000次，则费用为 ：10000 x 0.2 + 5000 x 0.18 = 2900 元.

月调用量（次）	价格（元/次）
0 < n <= 10,000	0.20
10,000 < n <= 100,000	0.18
100,000 < n <=500,000	0.15
500,000 < n	0.12

（2）预付费资源包

有效期为1年，次数包支持叠加购买，按照购买时间顺序依次抵扣，全部使用完毕后自动切换为按量后付费模式。超出有效期未抵扣额度自动失效，无法继续使用，请根据实际业务需求酌情购买。

次数包	价格	有效期
10,000 次	1,950 元	一年
100,000次	19,000元	一年
500,000 次	90,000 元	一年
1,000,000 次	160,000 元	一年
3,000,000 次	420,000 元	一年
5,000,000 次	600,000元	一年

（3）QPS叠加包

购买 QPS 数量	按月购买	按天购买
0<QPS<=10	270元/月/QPS	16元/天/QPS
10<QPS<=50	235元/月/QPS	14元/天/QPS
50<QPS<=100	170元/月/QPS	12元/天/QPS
100<QPS	135元/月/QPS	10元/天/QPS

> **说明**
> 1、本接口**只计费成功调用**。失败调用会返回对应的错误码，详情说明可参考错误码文档

2、预付费包年次数包**有效期为1年**，支持叠加购买，按照购买时间顺序依次抵扣，全部使用完毕后自动切换为按量后付费模式。次数包购买时会自动开通**按量后付费**，**请勿在消耗次数包时终止后付费，避免影响正常使用**。次数包超出有效期未抵扣额度自动失效，无法继续使用，请根据实际业务需求酌情购买。

3、预付费包年次数包购买后**7天内未产生调用**，如有需要，您可对购买的QPS进行自助退款，您可前往百度智能云'控制台--财务--退订管理'页面进行自助退订，详细退订规则见退订说明， **超过7天**或**已产生调用的次数包**无法退款。

1.3.4 人脸实名认证（V4）

温馨提示：

人脸实名认证（V4）接口需要完成企业认证后方可使用。新企业认证用户可获得**500次免费测试量**，免费测试量用尽后开始计费。如您之前已领取/使用过人脸实名认证（V3）系列API，则已享受过500次免费测试量，**不再重复提供**。开通付费/购买次数包

完成企业认证后接口并发支持默认赠送**2QPS**，开通付费后，并发支持将扩充到**10QPS**，如需更多QPS，可以**联系您的商务经理**，或通过合作咨询与我们联系。

（1）按量后付费：

月调用量（次）	价格（元/次）
0 < n <= 10,000	0.8
10,000 < n <= 100,000	0.72
100,000 < n <=200,000	0.6
200,000 < n	0.48

（2）预付费资源包

次数包	价格	有效期
1,000 次	780 元	一年
5,000次	3,700元	一年
10,000 次	7,000 元	一年
50,000 次	33,000 元	一年
100,000 次	61,000 元	一年
500,000 次	275,000 元	一年
1000,000 次	500,000 元	一年

说明

1、本接口**只计费成功调用**。失败调用会返回对应的错误码，详情说明可参考错误码文档

2、预付费包年次数包**有效期为1年**，支持叠加购买，按照购买时间顺序依次抵扣，全部使用完毕后自动切换为按量后付费模式。次数包购买时会自动开通**按量后付费**，**请勿在消耗次数包时终止后付费，避免影响正常使用**。次数包超出有效期未抵扣额度自动失效，无法继续使用，请根据实际业务需求酌情购买。

3、预付费包年次数包购买后**7天内未产生调用**，如有需要，您可对购买的QPS进行自助退款，您可前往百度智能云'控制台--财务--退订管理'页面进行自助退订，详细退订规则见退订说明， **超过7天**或**已产生调用的次数包**无法退款。

1.3.5 人脸实名认证（V4）含有效期核验

> **温馨提示：**
> **人脸实名认证（V4）**接口需要完成企业认证后方可使用。完成企业认证后，如需进行接口测试，**请联系商务同学申请测试额度**。开通付费/购买次数包
>
> 接口开通付费后，并发支持将扩充到**10QPS**，如需更多QPS，可以**联系您的商务经理**，或通过合作咨询与我们联系。

（1）按量后付费：

月调用量（次）	价格（元/次）
0 < n <= 10,000	2.5
10,000 < n <= 100,000	2.375
100,000 < n	2.25

（2）预付费资源包

次数包	价格	有效期
10,000 次	23,750 元	一年
50,000 次	11,2500 元	一年
100,000 次	21,2500 元	一年

> **说明**
> 1、本接口**只计费成功调用**。失败调用会返回对应的错误码，详情说明可参考错误码文档
>
> 2、预付费包年次数包**有效期为1年**，支持叠加购买，按照购买时间顺序依次抵扣，全部使用完毕后自动切换为按量后付费模式。次数包购买时会自动开通**按量后付费，请勿在消耗次数包时终止后付费，避免影响正常使用**。次数包超出有效期未抵扣额度自动失效，无法继续使用，请根据实际业务需求酌情购买。
> 3、预付费包年次数包购买后**7天内未产生调用**，如有需要，您可对购买的QPS进行自助退款，您可前往百度智能云'控制台--财务--退订管理'页面进行自助退订，详细退订规则见退订说明， **超过7天或已产生调用的次数包**无法退款。

1.3.6 人脸实名认证（V3）

> **温馨提示：**
> **人脸实名认证（V3）**接口为历史版本接口，如您在百度智能云控制台-服务列表-历史版本位置未看到此接口，则无权限，请移步查看人脸实名认证（V4）文档。
>
> 完成企业认证后接口并发支持默认赠送**2QPS**，开通付费后，并发支持将扩充到**10QPS**，如需更多QPS，可以**联系您的商务经理**，或通过合作咨询与我们联系。

（1）按量后付费：

月调用量（次）	价格（元/次）
0 < n <= 10,000	0.6
10,000 < n <= 50,000	0.55
50,000 < n <=100,000	0.5
100,000 < n	0.45

（2）预付费资源包

次数包	价格	有效期
1,000 次	590 元	一年
5,000次	2,900元	一年
10,000 次	5,600 元	一年
50,000 次	27,000 元	一年
100,000 次	50,000 元	一年
500,000 次	240,000 元	一年
1000,000 次	450,000 元	一年

> **说明：**
> 1、**人脸实名认证（V3）**接口计费包含**成功调用及部分失败错误码（222350、222351、222354、222355）**，详情说明可参考错误码文档
>
> 2、预付费包年次数包**有效期为1年**，支持叠加购买，按照购买时间顺序依次抵扣，全部使用完毕后自动切换为按量后付费模式。次数包购买时会自动开通**按量后付费，请勿在消耗次数包时终止后付费，避免影响正常使用**。次数包超出有效期未抵扣额度自动失效，无法继续使用，请根据实际业务需求酌情购买。
>
> 3、预付费包年次数包购买后**7天内未产生调用**，如有需要，您可对购买的QPS进行自助退款，您可前往百度智能云'控制台--财务--退订管理'页面进行自助退订，详细退订规则见退订说明， **超过7天或已产生调用的次数包**无法退款。

1.3.7 人证核验（含证件状态）

> **人证核验（含证件状态）**接口需要开通企业认证后方可使用。完成企业认证后，企业认证用户可领取**100次免费测试量，免费测试量用尽后开始计费。**开通付费
>
> 用户完成企业认证后接口并发支持默认赠送**2QPS**，开通付费后，并发支持将扩充到**10QPS**，如需更多QPS，可以通过提交工单与我们联系。

（1）按量后付费

分段阶梯计费：到达相应阶梯的计费调用量按照所在阶梯单价进行计费。如1月份内接口调用量为15000次，则费用为 ：10000 x 1.5 + 5000 x 1.425 = 22125 元.

月调用量（次）	价格（元/次）
0 < n <= 10,000	1.5
10,000 < n <= 100,000	1.425
100,000 < n	1.35

1.3.8 意愿核身（权威库）

> **温馨提示：**
> **意愿核身（权威库）**接口需要完成企业认证后方可使用。新企业认证用户可获得**200次免费测试量**，免费测试量用尽后开始计费。
>
> 完成企业认证后接口并发支持默认赠送**2QPS**，开通付费后，并发支持将扩充到**10QPS**，如需更多QPS，可以**联系您的商务经理**，或通过合作咨询与我们联系。

（1）按量后付费：

月调用量（次）	价格（元/次）
0 < n <= 10,000	1
10,000 < n <= 100,000	0.9
100,000 < n <=200,000	0.75
200,000 < n	0.6

（2）预付费资源包

次数包	价格	有效期
1,000 次	970 元	一年
5,000次	4,600元	一年
10,000 次	8,700 元	一年
50,000 次	41,000 元	一年
100,000 次	76,000 元	一年
500,000 次	340,000 元	一年
1000,000 次	620,000 元	一年

> **说明**
> 1、本接口**只计费成功调用**。失败调用会返回对应的错误码，详情说明可参考错误码文档
> 2、预付费包年次数包**有效期为1年**，支持叠加购买，按照购买时间顺序依次抵扣，全部使用完毕后自动切换为按量后付费模式。次数包购买时会自动开通**按量后付费，请勿在消耗次数包时终止后付费，避免影响正常使用**。次数包超出有效期未抵扣额度自动失效，无法继续使用，请根据实际业务需求酌情购买。
> 3、预付费包年次数包购买后**7天内未产生调用**，如有需要，您可对购买的QPS进行自助退款，您可前往百度智能云‘控制台--财务--退订管理’页面进行自助退订，详细退订规则见退订说明， **超过7天或已产生调用的次数包**无法退款。

1.3.9 意愿核身（自传照片）

> **温馨提示：**
> **意愿核身（自传照片）**接口需要完成企业认证后方可使用。新企业认证用户可获得**200次免费测试量**，免费测试量用尽后开始计费。
> 完成企业认证后接口并发支持默认赠送**2QPS**，开通付费后，并发支持将扩充到**10QPS**，如需更多QPS，可以**联系您的商务经理**，或通过合作咨询与我们联系。

（1）按量后付费：

月调用量（次）	价格（元/次）
0 < n <= 10,000	0.3
10,000 < n <= 100,000	0.27
100,000 < n <=200,000	0.21
200,000 < n	0.18

（2）预付费资源包

次数包	价格	有效期
10,000 次	2,800 元	一年
50,000次	13,000元	一年
100,000 次	25,000 元	一年
200,000 次	48,000 元	一年
500,000 次	100,000元	一年
1,000,000 次	180,000 元	一年

> **说明**
> 1、本接口**只计费成功调用**。失败调用会返回对应的错误码，详情说明可参考错误码文档
> 2、预付费包年次数包**有效期为1年**，支持叠加购买，按照购买时间顺序依次抵扣，全部使用完毕后自动切换为按量后付费模式。次数包购买时会自动开通**按量后付费，请勿在消耗次数包时终止后付费，避免影响正常使用**。次数包超出有效期未抵扣额度自动失效，无法继续使用，请根据实际业务需求酌情购买。
> 3、预付费包年次数包购买后**7天内未产生调用**，如有需要，您可对购买的QPS进行自助退款，您可前往百度智能云'控制台--财务--退订管理'页面进行自助退订，详细退订规则见退订说明， **超过7天**或**已产生调用的次数包**无法退款。

1.4 人像特效

人像特效 API：人脸融合、人脸属性编辑、人脸关键点

****1.4.1 人脸融合**** 个人或企业认证可领取**500次免费测试量**，用尽后开始计费，采用分段阶梯计费，具体信息如下：

（1）按量后付费：

月调用量（万次）	价格（元/次）
0 < n <= 30	0.04
30 < n <= 100	0.02
100 < n <= 500	0.01
500 < n <= 1000	0.005
1000 < n <= 3000	0.002
3000 < n	0.0018

（2）预付费资源包： 购买包年次数包，QPS 会免费提升至 10。

次数包	价格	有效期
1,000,000 次	20,000 元	一年
5,000,000 次	50,000 元	一年
10,000,000 次	80,000 元	一年
30,000,000 次	110,000 元	一年

（3）QPS叠加包：

QPS个数	价格
10 个	100元/天

> **说明**

1. 接口并发支持默认 **2QPS** ，开通付费后，并发支持将扩充至 **10QPS**。如需更多QPS，可以联系您的商务经理，或通过合作咨询 与我们联系。
2. 次数包支持叠加购买，按照购买时间顺序依次抵扣，全部使用完毕后自动切换为按量后付费模式。次数包购买时会自动开通**按量后付费**，**请勿在消耗次数包时终止后付费，避免影响正常使用**。次数包超出有效期未抵扣额度自动失效，无法继续使用，请根据实际业务需求酌情购买。
3. 预付费包年次数包购买后**7天内未产生调用**，如有需要，您可对购买的QPS进行自助退款，您可前往百度智能云'控制台--财务--退订管理'页面进行自助退订，详细退订规则见退订说明， **超过7天或已产生调用的次数包**无法退款。
   4.**人脸融合**接口计费包含**成功调用及部分失败错误码（222211、222307、222308、223129）**，详情说明可参考错误码文档

1.4.2 人脸属性编辑

个人或企业认证可领取**500次免费测试量**，用尽后开始计费，采用分段阶梯计费，具体信息如下：
（1）按量后付费：

月调用量（万次）	价格（元/次）
0 < n <= 5	0.05
5 < n <= 20	0.04
20 < n <= 40	0.02
40 < n <= 100	0.01
100 < n <= 300	0.005
300 < n	0.003

（2）预付费资源包： 购买包年次数包，QPS 会免费提升至 10。

次数包	价格	有效期
10,000 次	450 元	一年
50,000 次	2,000 元	一年
100,000 次	2,500 元	一年
300,000 次	3,500 元	一年
1,000,000 次	5,000 元	一年
10,000,000 次	30,000 元	一年

说明

1. 接口并发支持默认 **2QPS** ，开通付费后，并发支持将扩充至 **10QPS**。如需更多QPS，可以通过 商务咨询 与我们联系
2. 次数包支持叠加购买，按照购买时间顺序依次抵扣，全部使用完毕后自动切换为按量后付费模式。次数包购买时会自动开通**按量后付费**，**请勿在消耗次数包时终止后付费，避免影响正常使用**。次数包超出有效期未抵扣额度自动失效，无法继续使用，请根据实际业务需求酌情购买。
3. 预付费包年次数包购买后**7天内未产生调用**，如有需要，您可对购买的QPS进行自助退款，您可前往百度智能云'控制台--财务--退订管理'页面进行自助退订，详细退订规则见退订说明， ， **超过7天或已产生调用的次数包**无法退款。

1.4.3 人脸关键点

个人或企业认证可领取**500次免费测试量**，用尽后开始计费，采用分段阶梯计费，具体信息如下：

（1）按量后付费：

月调用量（万次）	价格（元/次）
0 < n <= 1	0.01
1 < n <= 5	0.008
5 < n <= 10	0.006
10 < n	0.005

（2）预付费资源包： 购买包年次数包，QPS 会免费提升至 10。

次数包	价格	有效期
5,000 次	48 元	一年
10,000 次	90 元	一年
100,000 次	600 元	一年
300,000 次	1,500 元	一年

> **说明**
>
> 1. 接口并发支持默认 **2QPS** ，开通付费后，并发支持将扩充至 **10QPS**。如需更多QPS，可以通过 商务咨询 与我们联系
> 2. 次数包支持叠加购买，按照购买时间顺序依次抵扣，全部使用完毕后自动切换为按量后付费模式。次数包购买时会自动开通**按量后付费**，**请勿在消耗次数包时终止后付费，避免影响正常使用**。次数包超出有效期未抵扣额度自动失效，无法继续使用，请根据实际业务需求酌情购买。
> 3. 预付费包年次数包购买后**7天内未产生调用**，如有需要，您可对购买的QPS进行自助退款，您可前往百度智能云‘控制台--财务--退订管理’页面进行自助退订，详细退订规则见退订说明， **超过7天**或**已产生调用的次数包**无法退款。

2、SDK

2.1 人脸离线采集 SDK

申请人脸实名认证手机端APP方案，获得用于Android、iOS APP的SDK，配合API调用。
如您需要单独使用H5端SDK，实现**微信内实时活体检测**，请提交商务咨询或申请人脸实名认证H5方案。

2.2 人脸离线识别 SDK

人脸离线识别 SDK根据设备授权数量收取费用，按独立阶梯计费，随着购买数量的增加，授权单价将会降低。购买授权并激活后，即可下载授权文件，永久有效。每个账号下默认分配 **5个试用版序列号**，有效期为**三个月**，供您前期测试。正式版产品价格如下表所示：

累计购买授权数	每个授权单价
第100-999个	119元/个
第1000-4999个	99元/个
第5000个及以上	79元/个

> **说明**：
>
> 1. 序列号按照设备维度授权（依据于硬件指纹信息），每个序列号仅可激活一台设备,激活后授权永久有效，但如果硬件信息发生变更（如拆卸网卡或后续增加网卡），或者硬件损坏，则授权不可恢复。

2. 序列号不限制开发平台，仅绑定于指定的设备。例如RK3399开发板，无论是使用安卓还是 Ubuntu，序列号都可以激活。
3. 序列号购买后会明文展示在控制台，购买后**不支持退款**。

   示例：序列号的购买是「独立阶梯计费」，例如您需购买900个授权，采购区间对应100-1000的之间，所以需付费119元/个 x 900个 = 107,100元； 当您购买2000个授权，采购区间对应1000-4999之间，则付费 99元/个 x 2000=198,000元；若您的购买量达到10000个及以上，请您提交商务咨询，我们将安排销售人员与您线下对接。

3、私有化部署包

人脸识别私有化部署包提供**人脸检测与属性分析、人脸比对、人脸搜索、活体检测**等功能，支持百万级超大型人脸库，可实现毫秒级响应。

私有化部署包按显卡授权数量计费，您有多少显卡购买多少授权数量即可，提交商务咨询获得报价.

说明：

1. 完成企业认证，即可获取人脸识别私有化部署包免费测试申请（具备有效期）
2. 准备部署，请了解环境要求
3. 百度在通用场景的基础上针对用摄像头大角度俯拍等场景进行了优化，适用于安防监控等场景，您可以在申请页面根据业务需求灵活选择采用使用**通用版**还是**安防版本**。

如何购买

本文主要讲述购买百度人脸识别服务的方法与注意事项。

账户充值

1. 直接点击充值，或者通过以下方式进入充值页面：
   - 在控制台首页，将鼠标移到右上角导航栏的“财务”按钮上方，弹出小窗口，点击“充值”按钮。

   - 在控制台首页，点击右上角导航栏的“财务”按钮，进入“财务中心”，在“财务总览”页面，点击"充值"按钮。

2. 进入充值页面，在“充值方式”处选择“在线支付”或“线下汇款”。
   - 在线支付：选择支付平台“银联个人”、“银联企业”、“快捷支付”、“支付宝”或“微信支付”。
   - 线下汇款：您可获取您的专属账号，并线下汇款至专属账户，系统会将汇款直接匹配到您的百度开放云账户，实现自动加款，快速到账。适用于所有通过百度智能云完成个人认证及企业认证的用户（暂不支持从百度钱包同步实名状态的用户）。具体操作如下：- 点击汇款信息栏对应的“获取您的专属汇款账号”，获取银行电汇自动加款专属账号。

> - 击“发送验证码”，输入收到的验证码后，点击“确定”按钮，获取您的专属汇款账号。
> - 到任意银行网点或网银，通过银行电汇完成百度智能云账户加款。

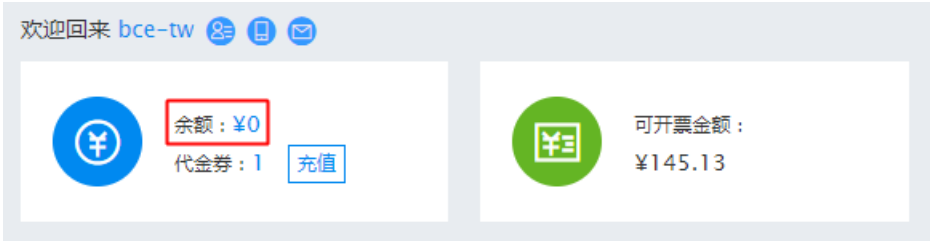
> 1.为保证汇款顺利进行，请务必保证汇款方名称与实名认证名称一致。
> 2.汇款方名称将是您后期开增值税专用发票的名称。

3. 选择“充值金额”或手动输入金额，点击“确认充值”。
4. 若充值未成功，点击“提交工单”，则进入提交工单页面，请您提交工单给我们。
5. 充值成功后，可点击导航栏的“收支明细”查看充值记录。

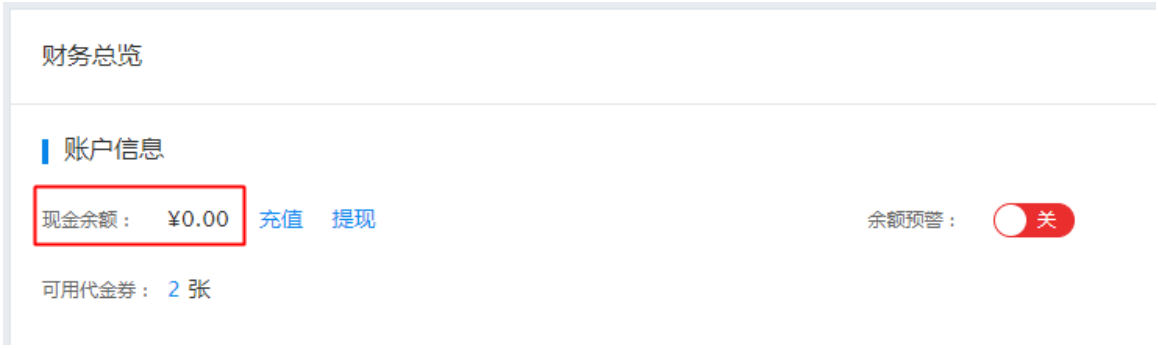
查看余额

有如下两种方式可查看当前账户余额：

1. 在控制台首页会显示当前账户余额。



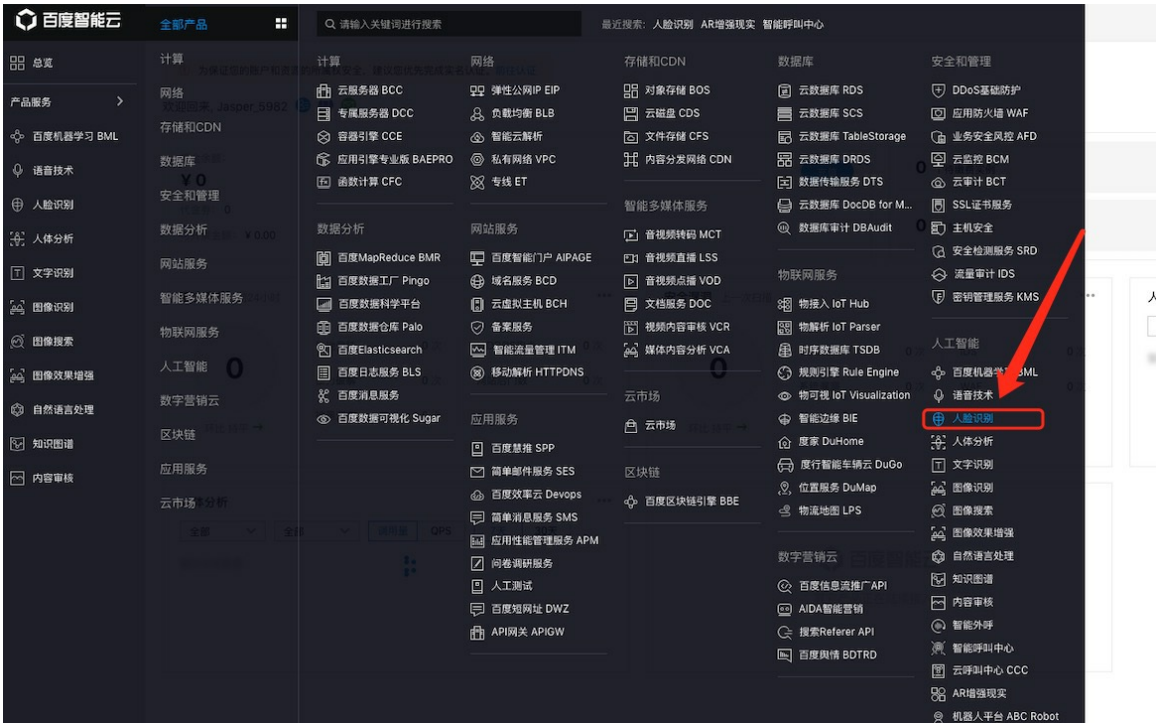
2. 由导航栏进入“财务中心”，在“账务总览”页面，“账户信息”区域会显示账户余额。



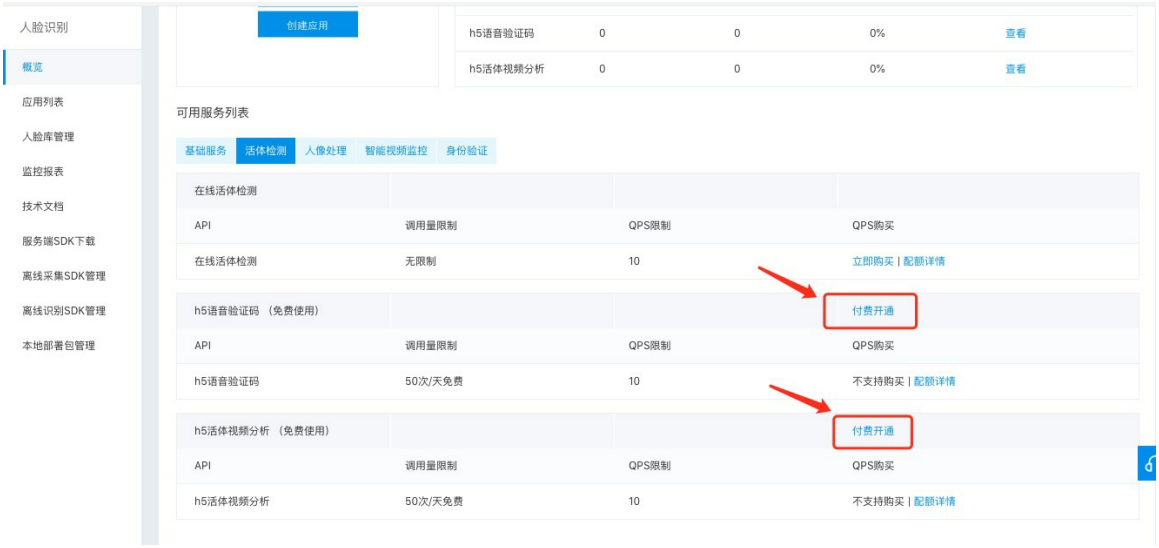
开通付费

按调用量计费的服务当免费额度使用完成后，如需继续使用，需要开通付费。

1. 登录百度云，进入人脸识别控制台。

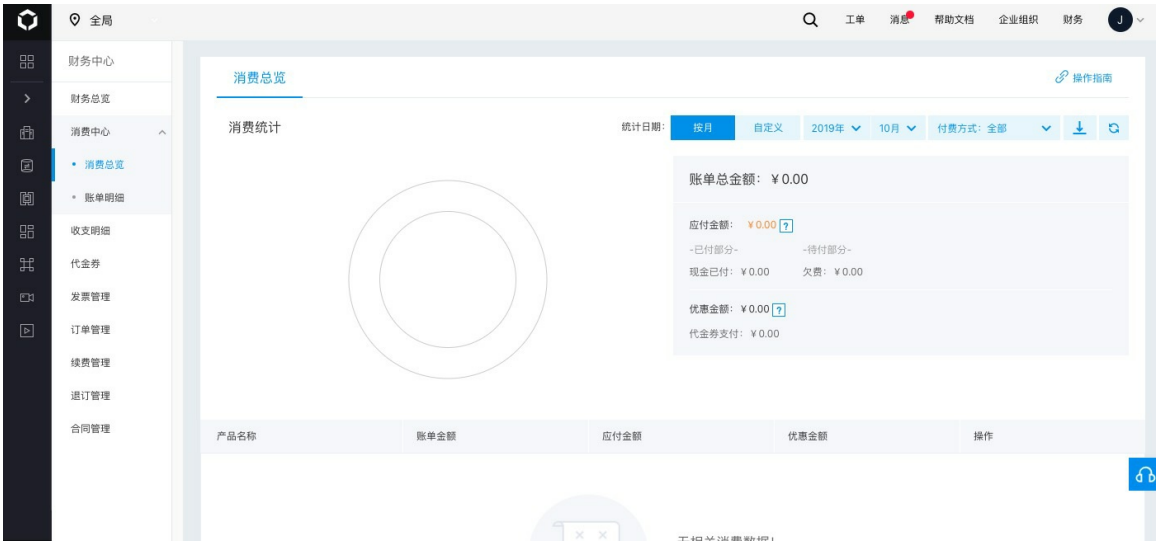


2. 选择服务，如图位置点击 "开通付费"，进入购买页面。



3. 选择需要开通付费的服务，开通付费。

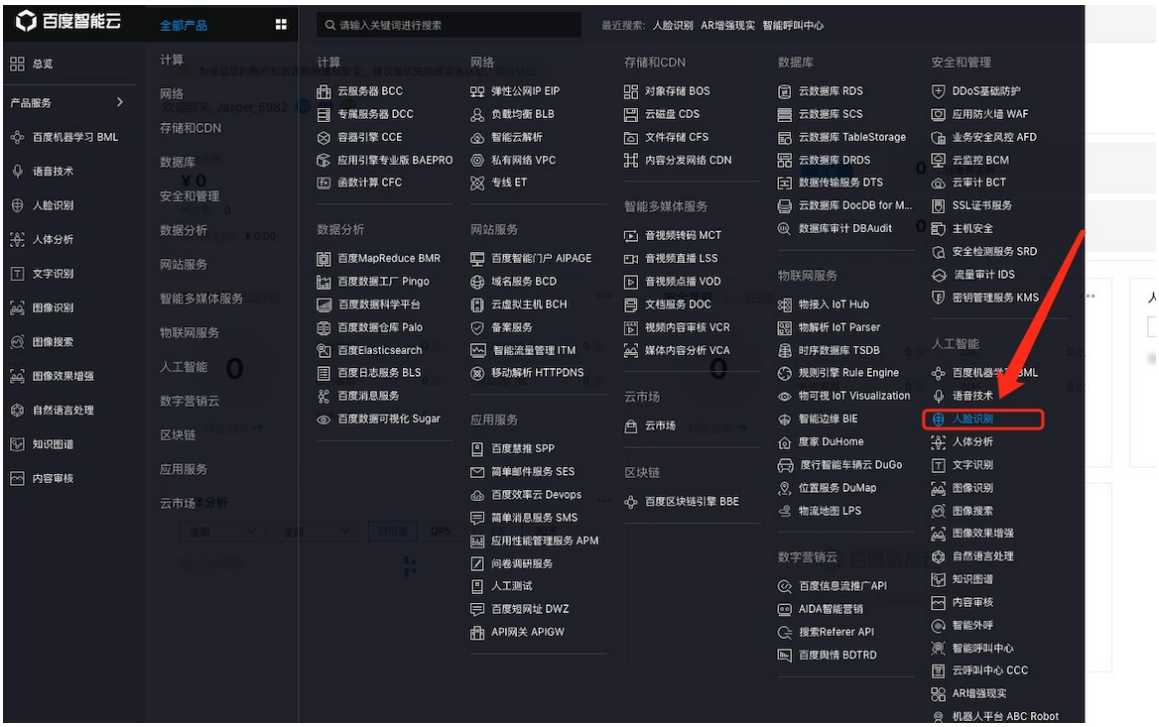
4. 开通付费后，服务即可正常使用，您可以在财务中心查看您的消费情况



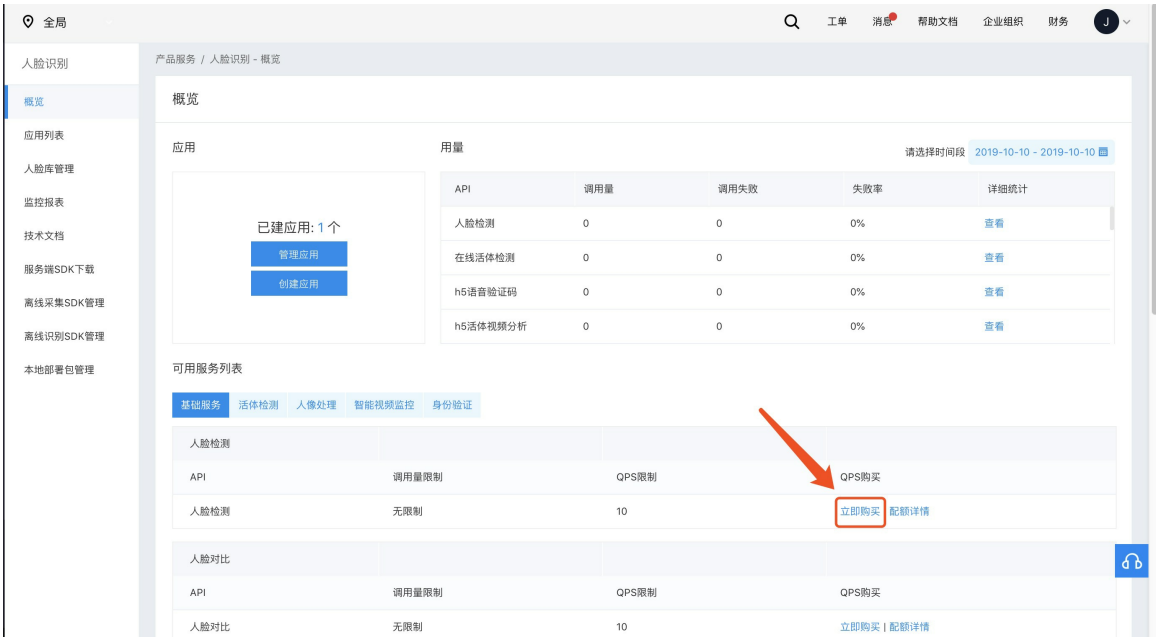
购买QPS

当赠送的 QPS 不足以满足您的业务需求时，您可以付费扩充 QPS。

1. 登录百度云，进入人脸识别控制台。

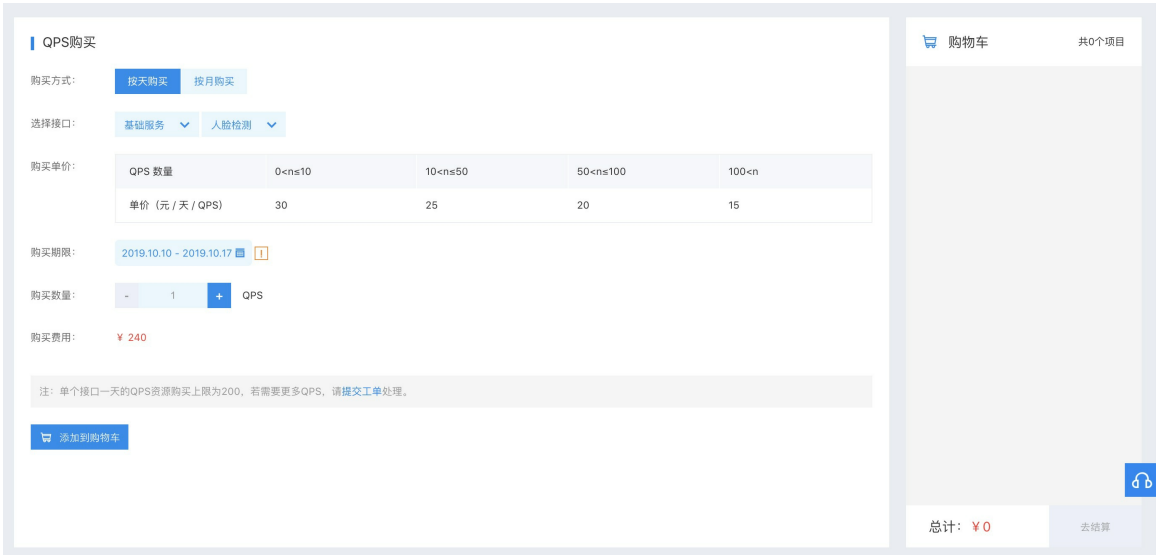


2. 选择服务，如图位置点击 "立即购买"，进入购买页面。

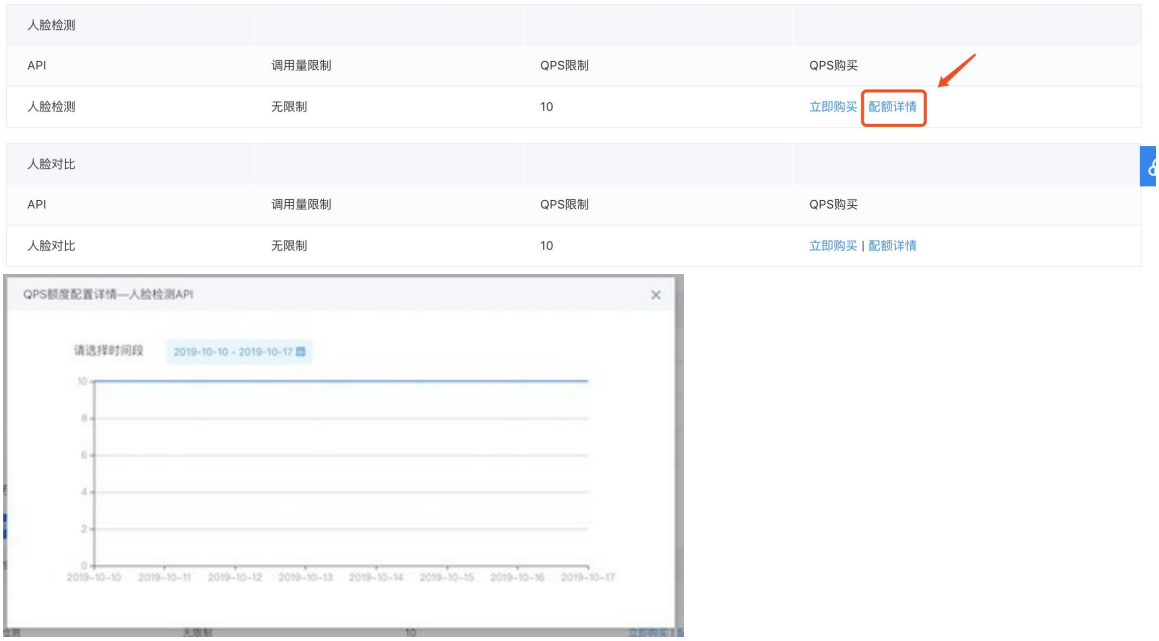


3. 选择购买方式、接口、购买期限、购买数量，添加至购物车并进行结算。

注意：由于购买 QPS 需要预留机器资源 购买后不支持退款。购买时请确认接口、期限、数量等信息无误



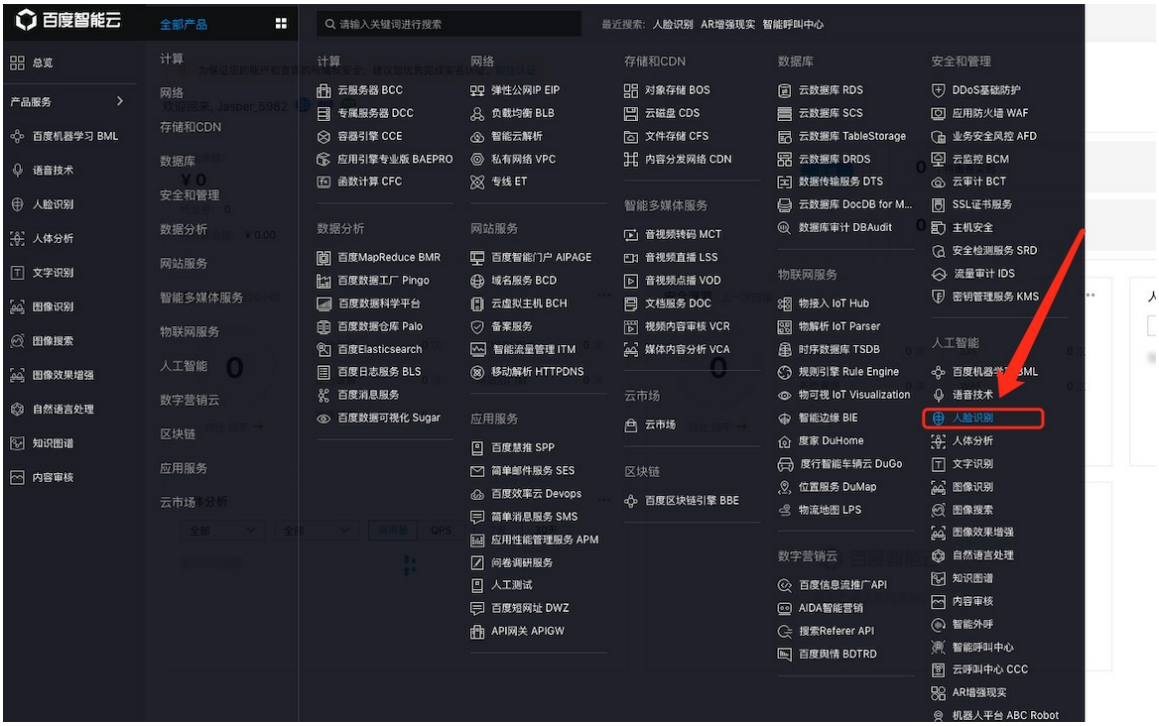
4. 购买 QPS 后，您可以在配额详情中查看对应服务的 QPS 配额



购买次数包

部分服务提供预付费次数包，价格更优惠。

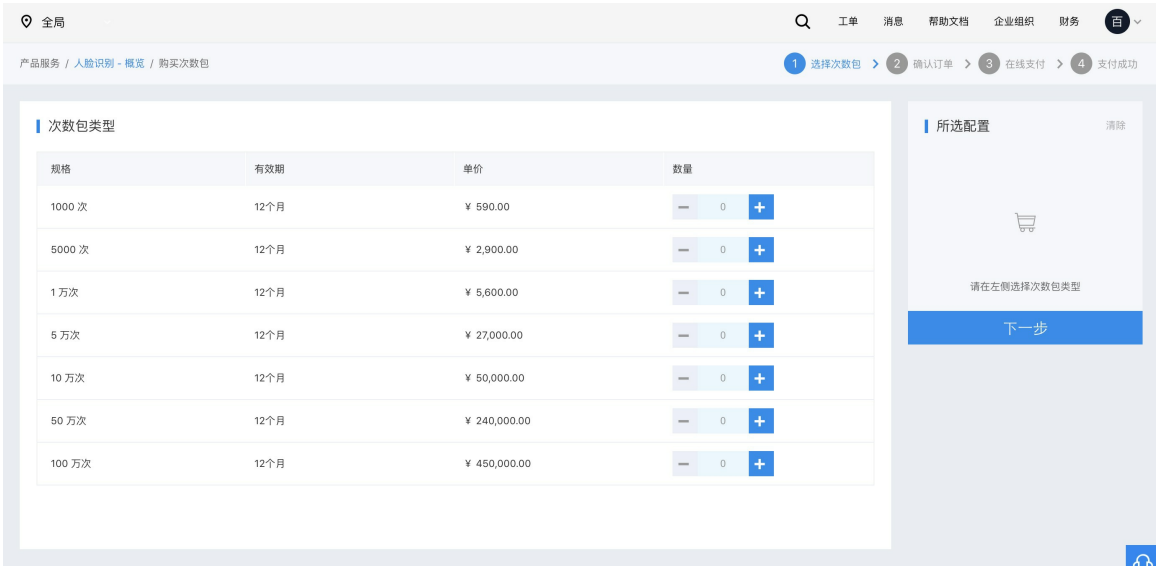
1. 登录百度云，进入人脸识别控制台。



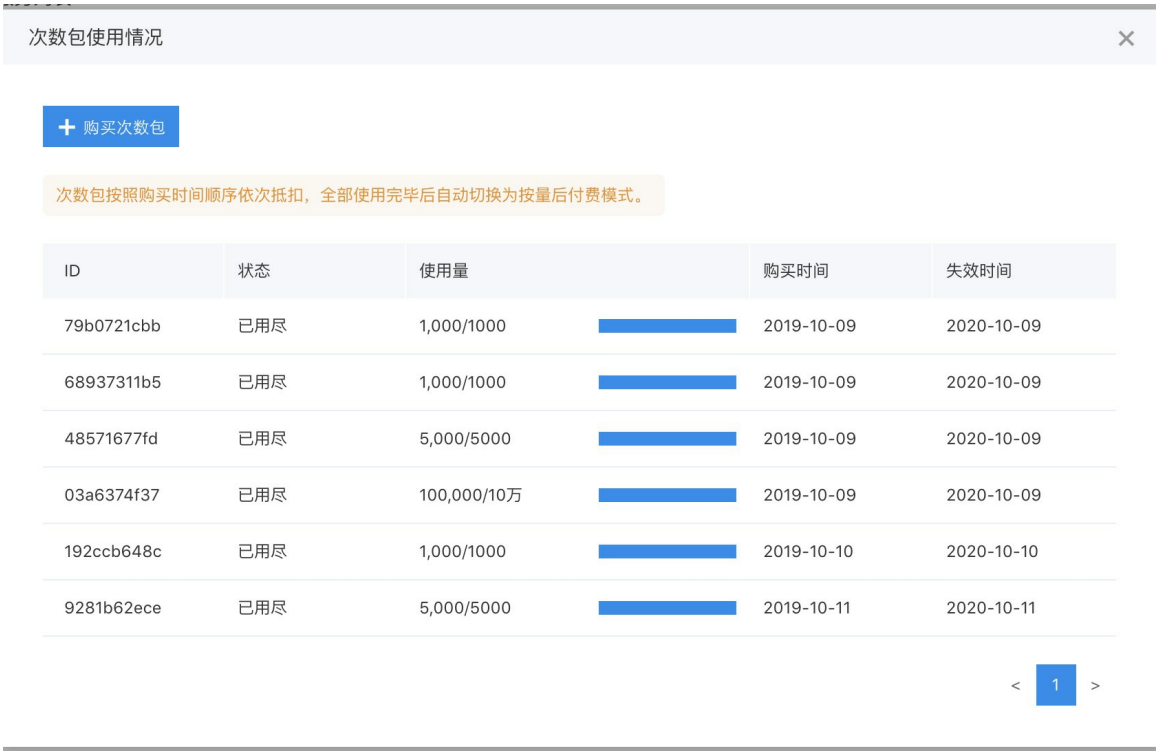
2. 选择服务，如图位置点击 "购买"，进入购买页面。

注意：V2、V3是不同的接口，购买时请确认选择的接口无误

3. 选择购买的次数包与购买数量，添加至购物车并进行结算。



4. 购买次数包后，可以在人脸识别控制台查看次数包使用详情。



> 次数包按照购买时间顺序依次抵扣，全部使用完毕后自动切换为按量后付费模式。

退订说明

关于退订，您可前往百度智能云“控制台--财务--退订管理”页面进行自助退订，详细退订规则见退订说明。

发票说明

关于发票申请说明，可参考文档：申请百度智能云发票

API文档

概述

您好，欢迎使用人脸识别（FACE）服务。

人脸识别基于深度学习技术，准确识别图片和视频流中的人脸信息，包含人脸检测与属性分析、人脸对比、人脸搜索、活体检测、人脸特效等API接口调用能力。

此外，本产品还面向人脸实名认证场景，推出了人脸实名认证解决方案：具备身份证识别、活体检测、风控、安全加密、权威库人脸核验、人脸比对等多项组合能力，支持在H5/APP业务场景快速完成用户身份核验，减少企业人工审核成本的同时，提升用户体验。

本文档主要针对API开发者，描述人脸识别接口服务的相关技术内容。如果您对文档内容有任何疑问，可以在百度智能云控制台内提交工单。

实名认证接口能力

接口名称	接口能力简要描述
身份证与名字比对	验证用户输入的身份证号码和姓名是否匹配，判断用户信息是否真实。
身份证与名字比对（含有效期核验）	验证用户输入的身份证号码、姓名、身份证有效期是否匹配，判断用户信息是否真实
人脸实名认证V4	基于姓名、身份证号、当前获取的人脸图片，与权威库进行三要素一致性对比，得出比对分数，并基于此进行业务判断是否为同一人。 可选功能包括质量检测、活体检测、图片加密及风控等。
人脸实名认证V4（含有效期核验）	基于姓名、身份证号、身份证有效期、当前获取的人脸图片，与权威库进行一致性对比，得出比对分数，并基于此进行业务判断是否为同一人。 可选功能包括质量检测、活体检测、图片加密及风控等。
人脸对比V4	对比两张图片中人脸的相似度，并返回相似度分值；支持生活照、证件照、身份证芯片照、带网纹照四种类型的人脸照片。 可选功能包括质量检测、活体检测、图片加密及风控等。
在线图片活体V4	判断图片中的人脸是否为二次翻拍，或是否为合成图攻击。 支持质量检测、返回人脸基础信息、图片加密及风控等。
人证核验（含证件状态）	验证用户输入的身份证号码、姓名、身份证有效期是否匹配，判断用户信息是否真实。并返回当前用户证件、户籍状态。

人脸识别接口能力

接口名称	接口能力简要描述
人脸检测	标记图片里人脸的位置，展示人脸关键点信息， 展示人脸属性（如年龄、性别等），展示人脸图片质量（如遮挡范围、光照、模糊、完整度、置信度等）
人脸对比	比对两张图片中人脸的相似度，并返回相似度分值。 支持生活照、证件照、身份证芯片照、带网纹照等类型的照片。具备活体检测能力，可判断人脸是否为二次翻拍。
人脸搜索与库管理	可在人脸库中，找到最相似的人脸。 提供完整的搜索与库管理能力，包括人脸搜索、人脸信息的注册/更新/删除/信息查询等。
场景化搜索	与『人脸搜索与库管理』类似，区别在于场景化搜索更适用于抓拍机摄像头等大角度俯拍的视频监控场景。
在线图片活体检测	判断图片中的人脸是否为二次翻拍以及是否为合成图。同时也会返回人脸的基础信息（人脸框位置、人脸旋转角度）和人脸质量检测信息（遮挡、光照、模糊度、完整度等）。
视频活体检测	用户新录制并上传一个视频，接口可返回活体置信度。 可实现动作视频活体、数字视频活体两种活体检测方式

人像特效接口能力

接口名称	接口能力简要描述
人脸融合	对两张人脸进行融合处理，生成的人脸同时具备两张人脸的外貌特征。
人脸属性编辑	对人脸属性特征进行编辑，实现性别互换、变老人、变小孩等图片特效。
人脸关键点	支持人脸 72关键点、150关键点、201 关键点检测，关键点包括人脸、眼睛、眉毛、嘴唇以及鼻子轮廓等。

通用说明

API认证机制

AI开放能力采用access_token认证机制。access_token是用户的访问令牌，承载了用户的身份、权限等信息。 要获取access_token，您需要在控制台上先创建应用，并获取到应用的API Key和Secret Key（以下分别简称为AK和SK）。

获取到AK和SK后，您有两种方式获取access_token：

- 代码形式→适用于有计算机基础的用户，或已开始开始将接口能力集成到业务中的用户
- 在线调试工具（推荐）→可快速调试接口效果

获取到access_token后，您需要将其拼接到调用接口服务的请求URL里。服务端接收到正确的access_token，则通过鉴权认证，并为您返回接口的业务信息。

整体流程图如下图。进一步的细节请参考鉴权认证机制。

请求参数说明

- **请求体格式化**：Content-Type为`application/json`，通过json格式化请求体。
- **Base64编码**：请求的图片需经过`Base64编码`，图片的base64编码指将图片数据编码成一串字符串，使用该字符串代替图像地址。您可以首先得到图片的二进制，然后用Base64格式编码即可。需要注意的是，图片的base64编码是不包含图片头的，如``
- **图片格式**：现支持PNG、JPG、JPEG、BMP，**不支持GIF图片**

请求参数包括如下3种：

参数类型	说明
URL参数	主要包括鉴权信息access_token，放在URL的query里
Header	Content-Type:application/json
Body	用于放置每个接口的请求，请参考具体接口的API文档

响应参数说明

响应主要关注响应body。

如调用成功，body里会包含调用所需的响应数据；如调用失败，也会在body里返回错误码和错误描述信息。

通信协议

为保证安全性，建议通过HTTPS调用。

人脸识别基础接口

人脸检测与属性分析

能力介绍

接口能力

本文档为人脸检测与属性分析接口使用文档，人脸检测与属性分析接口能识别出人脸位置、人脸关键点、人脸属性值、人脸质量信息。

- **人脸检测**：检测图片中的人脸并标记出位置信息。
- **人脸关键点**：展示人脸的核心关键点信息，及150个关键点信息。
- **人脸属性值**：展示人脸属性信息，如年龄、性别等。
- **人脸质量信息**：返回人脸各部分的遮挡、光照、模糊、完整度、置信度等信息。

业务应用

典型应用场景：如**人脸属性分析**，**基于人脸关键点的加工分析**，**人脸营销活动**等。

> 说明：检测响应速度，与图片中人脸数量相关，人脸数量较多时响应时间会有些许延长。

质量检测

如果需要判断一张图片中的人脸，是否符合后续识别或者对比的条件，可以使用此接口，在请求时在face_field参数中请求quality。基于返回结果quality中，以下字段及对应阈值，进行质量检测的判断，以保证人脸质量符合后续业务操作要求。

指标	字段与解释	推荐数值界限
遮挡范围	**occlusion**，取值范围[0~1]，0为无遮挡，1是完全遮挡 含有多个具体子字段，表示脸部多个部位 通常用作判断头发、墨镜、口罩等遮挡	left_eye：0.6, #左眼被遮挡的阈值 right_eye：0.6, #右眼被遮挡的阈值 nose：0.7, #鼻子被遮挡的阈值 mouth：0.7, #嘴巴被遮挡的阈值 left_cheek：0.8, #左脸颊被遮挡的阈值 right_cheek：0.8, #右脸颊被遮挡的阈值 chin_contour：0.6, #下巴被遮挡阈值
模糊度范围	**blur**，取值范围[0~1]，0是最清晰，1是最模糊	小于0.7
光照范围	**illumination**，取值范围[0~255] 脸部光照的灰度值，0表示光照不好 以及对应客户端SDK中，YUV的Y分量	大于40
姿态角度	**Pitch**：三维旋转之俯仰角度[-90(上), 90(下)] **Roll**：平面内旋转角[-180(逆时针), 180(顺时针)] **Yaw**：三维旋转之左右旋转角[-90(左), 90(右)]	分别小于20度
人脸完整度	**completeness**（0或1），0为人脸溢出图像边界， 1为人脸都在图像边界内	视业务逻辑判断
人脸大小	人脸部分的大小 建议长宽像素值范围：80*80~200*200	人脸部分不小于**100*100**像素

在线调试

您可以在 示例代码中心 中调试该接口，可进行签名验证、查看在线调用的请求内容和返回结果、示例代码的自动生成。

调用方式

请求URL数据格式

向API服务地址使用POST发送请求，必须在URL中带上参数access_token，可通过后台的API Key和Secret Key生成，具体方式请参考“Access Token获取”。

示例代码

Bash

PHP

JAVA

Python

Cpp

C#

Node

```
##### !/bin/bash
curl -i -k 'https://aip.baidubce.com/oauth/2.0/token?grant_type=client_credentials&client_id=【应用的AK】&client_secret=【应用的SK】'
```

> **注意**：access_token的有效期为30天，**切记需要每30天进行定期更换，或者每次请求都拉取新token**；

例如此接口，使用HTTPS POST发送：

```
https://aip.baidubce.com/rest/2.0/face/v3/detect?access_token=24.f9ba9c5341b67688ab4added8bc91dec.2592000.1485570332.282335-8574074
```

POST中Body的参数，按照下方请求参数说明选择即可。

> **提示**：如果您为百度智能云老用户，正在使用其他非AI的服务，可以参考百度智能云AKSK鉴权方式发送请求，虽然**请求方式**和**鉴权方法**和本文所介绍的不同，但请求参数和返回结果一致。

请求说明

注意事项：

- **请求体格式化**：Content-Type为application/json，通过json格式化请求体。
- **Base64编码**：请求的图片需经过Base64编码，图片的base64编码指将图片数据编码成一串字符串，使用该字符串代替图像地址。您可以首先得到图片的二进制，然后用Base64格式编码即可。需要注意的是，图片的base64编码是不包含图片头的，如
- **图片格式**：现支持PNG、JPG、JPEG、BMP，**不支持GIF图片**

- **如图片中存在多个人脸（大于1），您想要使用此接口获取的多个face_token进行对比、入库、检索等操作，需要在调用接口时将face_field参数传入feature值。**

请求示例

HTTP方法：`POST`

请求URL： `https://aip.baidubce.com/rest/2.0/face/v3/detect`

URL参数：

参数	值
access_token	通过API Key和Secret Key获取的access_token,参考“Access Token获取”

Header：

参数	值
Content-Type	application/json

Body中放置请求参数，参数详情如下：

请求参数

参数	必选	类型	说明
image	是	string	图片信息(**总数据大小应小于10M，分辨率应小于1920*1080**)，图片上传方式根据image_type来判断
image_type	是	string	图片类型 **BASE64**:（**推荐**）图片的base64值，base64编码后的图片数据，编码后的图片大小不超过2M； **FACE_TOKEN**: 人脸图片的唯一标识，调用人脸检测接口时，会为每个人脸图片赋予一个唯一的FACE_TOKEN，同一张图片多次检测得到的FACE_TOKEN是同一个。
face_field	否	string	包括**age,expression,face_shape,gender,glasses,landmark,landmark150,quality,eye_status,emotion,face_type,mask,spoofing信息** 逗号分隔. 默认只返回face_token、人脸框、概率和旋转角度
max_face_num	否	uint32	最多处理人脸的数目，默认值为1，根据人脸检测排序类型检测图片中排序第一的人脸（默认为人脸面积最大的人脸），**最大值20**
face_type	否	string	人脸的类型 **LIVE**表示生活照：通常为手机、相机拍摄的人像图片、或从网络获取的人像图片等 **IDCARD**表示身份证芯片照：二代身份证内置芯片中的人像照片 **WATERMARK**表示带水印证件照：一般为带水印的小图，如公安网小图 **CERT**表示证件照片：如拍摄的身份证、工卡、护照、学生证等证件图片 默认**LIVE**
liveness_control	否	string	**活体控制** 检测结果中不符合要求的人脸会被过滤 **NONE**: 不进行控制 **LOW**:较低的活体要求(高通过率 低攻击拒绝率) **NORMAL**: 一般的活体要求(平衡的攻击拒绝率, 通过率) **HIGH**: 较高的活体要求(高攻击拒绝率 低通过率) 默认**NONE**
face_sort_type	否	int	人脸检测排序类型 **0**:代表检测出的人脸按照人脸面积从大到小排列 **1**:代表检测出的人脸按照距离图片中心从近到远排列 **默认为0**
display_corp_image	否	int	是否显示检测人脸的裁剪图base64值 0：不显示（默认） 1：显示 **当取值为1时，max_face_num字段的取值上限按5计算，即最多可返回5张人脸的裁剪图**

> 说明：face_field参数，默认只返回人脸框、概率和旋转角度，age等更多属性，请在此参数中添加。

示例代码

提示一：使用示例代码前，请记得替换其中的示例Token、图片地址或Base64信息。

提示二：部分语言依赖的类或库，请在代码注释中查看下载地址。

Bash

PHP

JAVA

Python

Cpp

C#

人脸检测与属性分析

```
curl -i -k 'https://aip.baidubce.com/rest/2.0/face/v3/detect?access_token=【调用鉴权接口获取的token】' --data '{"image":"027d8308a2ec665acb1bdf63e513bcb9","image_type":"FACE_TOKEN","face_field":"faceshape,facetype"}' -H 'Content-Type:application/json; charset=UTF-8'
```

返回说明

返回参数

- 返回结果

字段	必选	类型	说明
face_num	是	int	检测到的图片中的人脸数量
face_list	是	array	人脸信息列表，具体包含的参数参考下面的列表。
+face_token	是	string	人脸图片的唯一标识 （人脸检测face_token有效期为60min）。**如果希望使用此token做人脸1:1比对，请求接口时，需传入face_field=feature**
+location	是	array	人脸在图片中的位置
++left	是	double	人脸区域离左边界的距离
++top	是	double	人脸区域离上边界的距离
++width	是	double	人脸区域的宽度
++height	是	double	人脸区域的高度
++rotation	是	int64	人脸框相对于竖直方向的顺时针旋转角，[-180,180]
+face_probability	是	double	人脸置信度，范围【0~1】，代表这是一张人脸的概率，0最小、1最大。其中返回0或1时，数据类型为Integer
+angle	是	array	人脸旋转角度参数
++yaw	是	double	三维旋转之左右旋转角[-90(左), 90(右)]
++pitch	是	double	三维旋转之俯仰角度[-90(上), 90(下)]
++roll	是	double	平面内旋转角[-180(逆时针), 180(顺时针)]
+age	否	double	年龄 ，当**face_field包含age时返回**
+expression	否	array	表情，当 **face_field包含expression时返回**
++type	否	string	**none**:不笑；**smile**:微笑；**laugh**:大笑
++probability	否	double	表情置信度，范围【0~1】，0最小、1最大。

+face_shape	否	array	脸型，当**face_field包含face_shape时返回**
++type	否	string	**square**: 正方形 **triangle**:三角形 **oval**: 椭圆 **heart**: 心形 **round**: 圆形
++probability	否	double	置信度，范围【0~1】，代表这是人脸形状判断正确的概率，0最小、1最大。
+gender	否	array	性别，**face_field包含gender时返回**
++type	否	string	male:**男性** female:**女性**
++probability	否	double	性别置信度，范围【0~1】，0代表概率最小、1代表最大。
+glasses	否	array	是否带眼镜，**face_field包含glasses时返回**
++type	否	string	**none**:无眼镜，**common**:普通眼镜，**sun**:墨镜
++probability	否	double	眼镜置信度，范围【0~1】，0代表概率最小、1代表最大。
+eye_status	否	array	双眼状态（睁开/闭合） **face_field包含eye_status时返回**
++left_eye	否	double	左眼状态 [0,1]取值，越接近0闭合的可能性越大
++right_eye	否	double	右眼状态 [0,1]取值，越接近0闭合的可能性越大
+emotion	否	array	情绪 **face_field包含emotion时返回**
++type	否	string	**angry**:愤怒 **disgust**:厌恶 **fear**:恐惧 **happy**:高兴 **sad**:伤心 **surprise**:惊讶 **neutral**:无表情 **pouty**:撅嘴 **grimace**:鬼脸
++probability	否	double	情绪置信度，范围0~1
+face_type	否	array	真实人脸/卡通人脸 **face_field包含face_type时返回**
++type	否	string	**human**: 真实人脸 **cartoon**: 卡通人脸
++probability	否	double	人脸类型判断正确的置信度，范围【0~1】，0代表概率最小、1代表最大。
+mask	否	array	口罩识别 **face_field包含mask时返回**
++type	否	int	没戴口罩/戴口罩 取值0或1 **0代表没戴口罩 1 代表戴口罩**
++probability	否	double	置信度，范围0~1
+landmark	否	array	4个关键点位置，左眼中心、右眼中心、鼻尖、嘴中心。**face_field包含landmark时返回**
+landmark72	否	array	72个特征点位置 **face_field包含landmark时返回**
+landmark150	否	object	150个特征点位置 **face_field包含landmark150时返回**
+quality	否	array	人脸质量信息。**face_field包含quality时返回**
++occlusion	否	array	人脸各部分遮挡的概率，范围[0~1]，0表示完整，1表示不完整
+++left_eye	否	double	左眼遮挡比例，[0-1] ，1表示完全遮挡
+++right_eye	否	double	右眼遮挡比例，[0-1] ， 1表示完全遮挡
+++nose	否	double	鼻子遮挡比例，[0-1] ， 1表示完全遮挡
+++mouth	否	double	嘴巴遮挡比例，[0-1] ， 1表示完全遮挡
+++left_cheek	否	double	左脸颊遮挡比例，[0-1] ， 1表示完全遮挡
+++right_cheek	否	double	右脸颊遮挡比例，[0-1] ， 1表示完全遮挡
+++chin_contour	否	double	下巴遮挡比例，，[0-1] ， 1表示完全遮挡
++blur	否	double	人脸模糊程度，范围[0~1]，0表示清晰，1表示模糊
++illumination	否	double	取值范围在[0~255], 表示脸部区域的光照程度 越大表示光照越好

++completeness	否	int64	人脸完整度，0或1, 0为人脸溢出图像边界，1为人脸都在图像边界内
+liveness	否	array	单张图片的活体信息。**liveness_control包含LOW\NORMAL\HIGH时返回**
++livemapscore	否	double	单张图片的活体得分范围[0~1]。
+spoofing	否	double	判断图片是合成图的概率
+not_spoofing	否	double	判断图片不是合成图的概率
corp_image_base64	否	string	检测人脸框的人脸图片base64值

关于合成图检测spoofing的判断阈值选择，可参考以下数值信息：

阈值	误拒率（FRR）	通过率（TAR）	攻击拒绝率（TRR））
0.00023	5%	95%	94.93%
0.00048（**推荐**）	1%	99%	89.71%
0.00066	0.5%	99.5%	88.02%
0.00109	0.1%	99.9%	84.57%
0.00171	0.05%	99.95%	81.52%
0.00547	0.01%	99.99%	65.52%

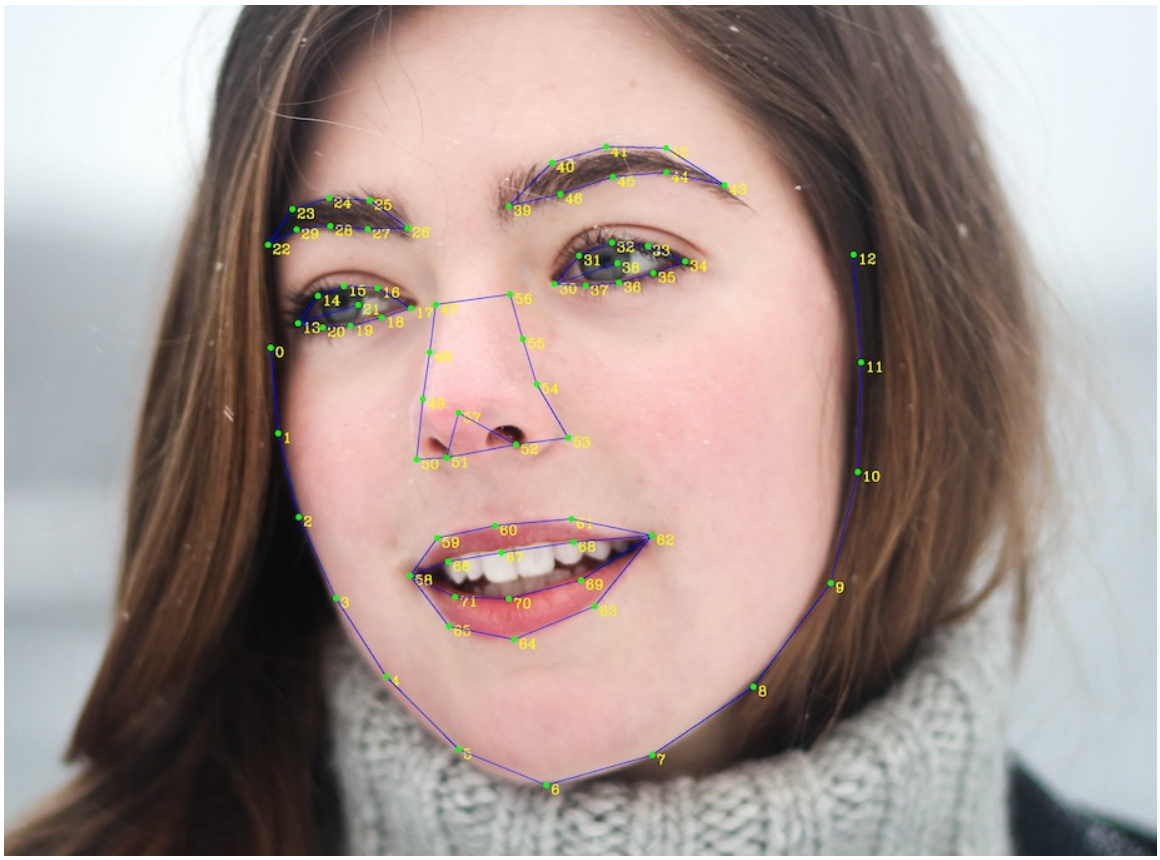
关于以上数值的概念介绍：

- **拒绝率（TRR）**：如99%，代表100次作弊假体攻击，会有99次被拒绝。
- **误拒率（FRR）**：如0.5%，指1000次真人请求，会有5次因为活体分数低于阈值被错误拒绝。
- **通过率（TAR）**：如99%，指100次真人请求，会有99次因为活体分数高于阈值而通过。
- **阈值（Threshold）**：高于此数值，则可判断为是合成图攻击。
- **返回示例**

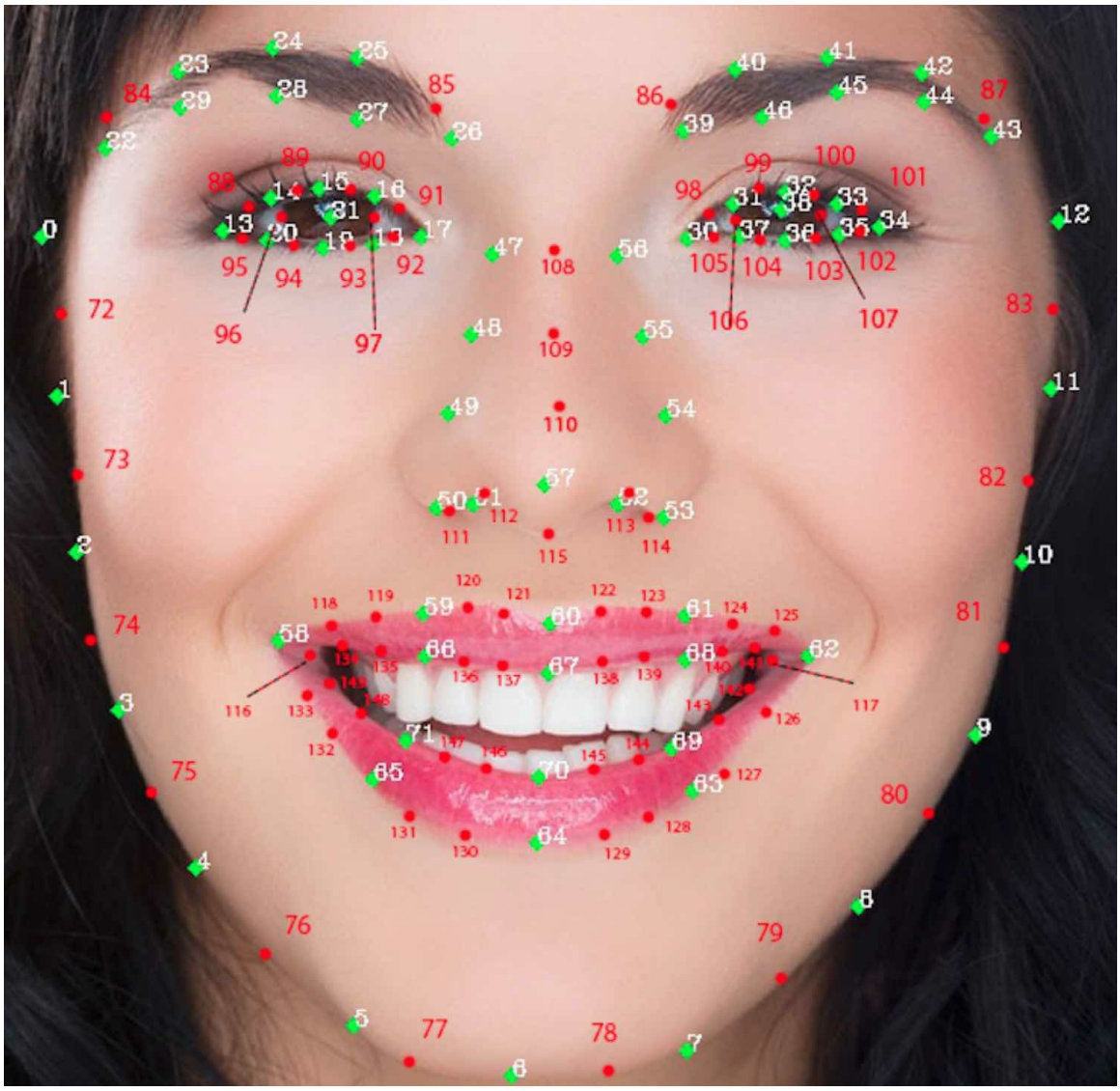

```
{
 "face_num": 1,
 "face_list": [
     {
        "face_token": "35235asfas21421fakghktyfdgh68bio",
        "location": {
           "left": 117,
           "top": 131,
           "width": 172,
           "height": 170,
           "rotation": 4
        },
        "face_probability": 1,
        "angle" :{
            "yaw" : -0.34859421849251
            "pitch": 1.9135693311691
            "roll" :2.3033397197723
        }
        "landmark": [
           {
              "x": 161.74819946289,
              "y": 163.30244445801
```

```
            },
            ...
        ],
        "landmark72": [
            {
                "x": 115.86531066895,
                "y": 170.0546875
            } ,
            ...
        ],
        "age": 29.298097610474,
        "expression": {
            "type": "smile",
            "probability" : 0.5543018579483
        },
        "gender": {
            "type": "male",
            "probability": 0.99979132413864
        },
        "glasses": {
    "type": "sun",
            "probability": 0.99999964237213
        },

        "face_shape": {
            "type": "triangle",
            "probability": 0.5543018579483
        }
        "quality": {
            "occlusion": {
                "left_eye": 0,
                "right_eye": 0,
                "nose": 0,
                "mouth": 0,
                "left_cheek": 0.0064102564938366,
                "right_cheek": 0.0057411273010075,
                "chin": 0
            },
            "blur": 1.1886881756684e-10,
            "illumination": 141,
            "completeness": 1
        }
    }
  ]
}
```


72个关键点分布图（对应landmark72个点的顺序，序号从0-71）：



150个关键点分布图，红色关键点为在72关键点基础上增加的关键点（对应landmark150个点的顺序，序号从0-149，且每个关键点有对应的英文命名作为参数名，请参考图片下方表格）：



关键点名称如下图，

序号	命名	序号	命名	序号	命名	序号	命名	序号	命名
0	cheek_right_1	31	eye_left_eyelid_upper_2	61	mouth_lip_upper_outer_9	91	eye_right_eyelid_upper_7	121	mouth_lip_upper_outer_5
1	cheek_right_3	32	eye_left_eyelid_upper_4	62	mouth_corner_left_outer	92	eye_right_eyelid_lower_7	122	mouth_lip_upper_outer_7
2	cheek_right_5	33	eye_left_eyelid_upper_6	63	mouth_lip_lower_outer_9	93	eye_right_eyelid_lower_5	123	mouth_lip_upper_outer_8
3	cheek_right_7	34	eye_left_corner_left	64	mouth_lip_lower_outer_6	94	eye_right_eyelid_lower_3	124	mouth_lip_upper_outer_10
4	cheek_right_9	35	eye_left_eyelid_lower_6	65	mouth_lip_lower_outer_3	95	eye_right_eyelid_lower_1	125	mouth_lip_upper_outer_11
5	cheek_right_11	36	eye_left_eyelid_lower_4	66	mouth_lip_upper_inner_3	96	eye_right_eyeball_right	126	mouth_lip_lower_outer_11
6	chin_2	37	eye_left_eyelid_lower_2	67	mouth_lip_upper_inner_6	97	eye_right_eyeball_left	127	mouth_lip_lower_outer_1
7	cheek_left_11	38	eye_left_eyeball_center	68	mouth_lip_upper_inner_9	98	eye_left_eyelid_upper_1	128	mouth_lip_lower_outer_8
8	cheek_left_9	39	eyebrow_left_corner_right	69	mouth_lip_lower_inner_9	99	eye_left_eyelid_upper_3	129	mouth_lip_lower_outer_7
9	cheek_left_7	40	eyebrow_left_upper_2	70	mouth_lip_lower_inner_6	100	eye_left_eyelid_upper_5	130	mouth_lip_lower_outer_5
10	cheek_left_5	41	eyebrow_left_upper_3	71	mouth_lip_lower_inner_3	101	eye_left_eyelid_upper_7	131	mouth_lip_lower_outer_4
11	cheek_left_3	42	eyebrow_left_upper_4	72	cheek_right_2	102	eye_left_eyelid_lower_7	132	mouth_lip_lower_outer_2
12	cheek_left_1	43	eyebrow_left_corner_left	73	cheek_right_4	103	eye_left_eyelid_lower_5	133	mouth_lip_lower_outer_1
13	eye_right_corner_right	44	eyebrow_left_lower_3	74	cheek_right_6	104	eye_left_eyelid_lower_3	134	mouth_lip_upper_inner_1
14	eye_right_eyelid_upper_2	45	eyebrow_left_lower_2	75	cheek_right_8	105	eye_left_eyelid_lower_1	135	mouth_lip_upper_inner_2
15	eye_right_eyelid_upper_4	46	eyebrow_left_lower_1	76	cheek_right_10	106	eye_left_eyeball_right	136	mouth_lip_upper_inner_4
16	eye_right_eyelid_upper_6	47	nose_right_contour_1	77	chin_1	107	eye_left_eyeball_left	137	mouth_lip_upper_inner_5
17	eye_right_corner_left	48	nose_right_contour_2	78	chin_3	108	nose_bridge_1	138	mouth_lip_upper_inner_7
18	eye_right_eyelid_lower_6	49	nose_right_contour_3	79	cheek_left_10	109	nose_bridge_2	139	mouth_lip_upper_inner_8
19	eye_right_eyelid_lower_4	50	nose_right_contour_4	80	cheek_left_8	110	nose_bridge_3	140	mouth_lip_upper_inner_10
20	eye_right_eyelid_lower_2	51	nose_right_contour_6	81	cheek_left_6	111	nose_right_contour_5	141	mouth_lip_upper_inner_11
21	eye_right_eyeball_center	52	nose_left_contour_6	82	cheek_left_4	112	nose_right_contour_7	142	mouth_lip_lower_inner_11
22	eyebrow_right_corner_right	53	nose_left_contour_4	83	cheek_left_2	113	nose_left_contour_7	143	mouth_lip_lower_inner_10
23	eyebrow_right_upper_2	54	nose_left_contour_3	84	eyebrow_right_upper_1	114	nose_left_contour_5	144	mouth_lip_lower_inner_8
24	eyebrow_right_upper_3	55	nose_left_contour_2	85	eyebrow_right_upper_5	115	nose_middle_contour	145	mouth_lip_lower_inner_7
25	eyebrow_right_upper_4	56	nose_left_contour_1	86	eyebrow_left_upper_1	116	mouth_corner_right_inner	146	mouth_lip_lower_inner_5
26	eyebrow_right_corner_left	57	nose_tip	87	eyebrow_left_upper_5	117	mouth_corner_left_inner	147	mouth_lip_lower_inner_4
27	eyebrow_right_lower_3	58	mouth_corner_right_outer	88	eye_right_eyelid_upper_1	118	mouth_lip_upper_outer_1	148	mouth_lip_lower_inner_2
28	eyebrow_right_lower_2	59	mouth_lip_upper_outer_3	89	eye_right_eyelid_upper_3	119	mouth_lip_upper_outer_2	149	mouth_lip_lower_inner_1
29	eyebrow_right_lower_1	60	mouth_lip_upper_outer_6	90	eye_right_eyelid_upper_5	120	mouth_lip_upper_outer_4		
30	eye_left_corner_right								

错误码

- **接口流控及鉴权错误码**

错误码	错误信息	描述	处理建议
2	Service temporarily unavailable	服务暂不可用	服务暂不可用，请再次请求，如果持续出现此类错误，请在控制台提交工单联系技术支持团队
4	Open api request limit reached	集群超限额	集群超限额，请再次请求，如果持续出现此类错误，请在控制台提交工单联系技术支持团队
6	no permission to access data	没有接口权限	请确认您调用的接口已经被赋权 常见问题是有V3版本权限，调用的是v2版本接口； 需要企业认证的，开通企业认证后一个小时左右即可使用。
17	Open api daily request limit reached	每天流量超限额	免费测试资源使用完毕，每天请求量超限额，已支持计费的接口，您可以在控制台人脸识别选择购买相关接口的次数包或开通按量后付费；邀测和未支持计费的接口，您可以在控制台提交工单申请提升限额
18	Open api qps request limit reached	QPS超限额	可直接自助购买更多QPS、联系商务接口人、或者提交工单
19	Open api total request limit reached	请求总量超限额	免费测试资源使用完毕，每天请求量超限额，已支持计费的接口，您可以在控制台人脸识别选择购买相关接口的次数包或开通按量后付费；邀测和未支持计费的接口，您可以在控制台提交工单申请提升限额
100	Invalid parameter	无效的access_token参数	token拉取失败，可以参考“Access Token获取”重新获取
110	Access token invalid or no longer valid	Access Token失效	**token有效期为30天，注意需要定期更换**，也可以每次请求都拉取新token
111	Access token expired	Access token过期	**token有效期为30天，注意需要定期更换**，也可以每次请求都拉取新token

- **人脸检测与属性分析错误码**

错误码	错误信息	描述	处理建议
222200	request body should be json format	1：请求体不是有效的JSON对象 2：请求体中字段参数值数据类型与接口文档要求不符	参考API文档说明，修改请求参数
222013	param[image] format error	参数格式错误	参考API文档说明，修改请求参数
222015	param[image_type] format error	参数格式错误	参考API文档说明，修改请求参数
222016	param[max_face_num] format erro	参数格式错误	参考API文档说明，修改请求参数
222017	param[face_field] format error	参数格式错误	参考API文档说明，修改请求参数
222020	param[liveness_control] format error	参数格式错误	参考API文档说明，修改请求参数
222024	param[face_type] format error	参数格式错误	参考API文档说明，修改请求参数
222039	param[face_sort_type] format error]	参数格式错误	参考API文档说明，修改请求参数
222043	param[display_corp_image] format error	参数格式错误	参考API文档说明，修改请求参数
222202	pic not has face	图片中没有人脸	检查图片质量，确保存在人脸
222203	image check fail	图片无法解析人脸	检查图片质量
222204	image download fail	从图片的url下载图片失败	确认图片 URL 公网可访问（没有白名单限制）
222209	face token not exist	face token不存在	请确认您操作的人脸已创建成功。 若face_token未注册到人脸库则有效期只有1小时 若注册到人脸库的face_token永久有效
222301	get face fail	获取人脸失败 备注：image_type为FACE_TOKEN模式时才会出现	请重试请求，如果持续出现此类错误，请提交工单
222304	image size is too large	图片尺寸太大	请确保图片尺寸不超过 6000 x 6000
222915	system busy	后端服务连接失败	请重试请求，若尝试多次无效，请提交工单咨询

人脸1：1对比

人脸1：1对比接口分为V2、V3和V4三个版本，本文档为V3版本接口的说明文档。

> 辨别接口版本的方法是：进入【应用列表】，点击【应用名称】，在【API列表】中可以看到【请求地址】，若请求地址中带有【v3】标识，则您具有的是v3权限，本文档适用于该接口；若请求地址中带有【v2】标识，则您具有的是v2权限，应该去阅读v2文档。

如您希望使用最新版接口，把图片加密及风控功能应用于您的业务，请移步人脸对比V4。

能力介绍

- **两张人脸图片相似度对比**：比对两张图片中人脸的相似度，并返回相似度分值。

- **多种图片类型**：支持**生活照**、**证件照**、**身份证芯片照**、**带网纹照**、**红外黑白照**五种类型的人脸对比。
- **活体检测控制**：基于图片中的破绽分析，判断其中的人脸是否为**二次翻拍**（举例：如用户A用手机拍摄了一张包含人脸的图片一，用户B翻拍了图片一得到了图片二，并用图片二伪造成用户A去进行识别操作，这种情况普遍发生在金融开户、实名认证等环节。）。
- **质量检测控制**：分析图片的中人脸的模糊度、角度、光照强度等特征，判断图片质量。
- **典型应用场景**：如**人证合一验证**，**用户认证**等，可与您现有的人脸库进行比对验证。

您可以在 示例代码中心 中调试该接口，可进行签名验证、查看在线调用的请求内容和返回结果、示例代码的自动生成。

请求结构

```
 POST /rest/2.0/face/v3/match?access_token={access_token}
HOST:aip.baidubce.com
Content-Type:application/json
{Body的参数，请参考下方请求参数说明。}
```

如果您正在使用其他百度智能云的服务，希望使用与其他产品（如云服务器、对象存储等）一致的鉴权机制，可以参考百度云AKSK鉴权方式发送请求，虽然**请求方式**和**鉴权方法**和本文所介绍的不同，但请求参数和返回结果一致。

请求参数

HTTP方法：`POST`

请求URL： `https://aip.baidubce.com/rest/2.0/face/v3/match`

URL参数：

参数	值
access_token	通过API Key和Secret Key获取的access_token,用于校验您的身份。获取方式详见“鉴权认证机制” **注意**：access_token的有效期为30天，**切记需要每30天进行定期更换，或者每次请求都拉取新token**。

Header：

参数	值
Content-Type	application/json

Body中放置请求参数，参数详情如下：

请求参数

- **Base64编码**：请求的图片需经过`Base64编码`，图片的base64编码指将图片数据编码成一串字符串，使用该字符串代替图像地址。您可以首先得到图片的二进制，然后用Base64格式编码即可。需要注意的是，图片的base64编码是不包含图片头的，如``。
- **图片格式**：现支持PNG、JPG、JPEG、BMP，**不支持GIF图片**。

参数	必选	类型	说明
image	是	string	图片信息(**总数据大小应小于10M，图片尺寸在1920x1080以下**)，图片上传方式根据image_type来判断。 **两张图片通过json格式上传，格式参考本表格下方的示例**
image_type	是	string	图片类型 **BASE64**:（**推荐**）图片的base64值，base64编码后的图片数据，编码后的图片大小不超过2M； **FACE_TOKEN**: 人脸图片的唯一标识，调用人脸检测接口时，会为每个人脸图片赋予一个唯一的FACE_TOKEN，同一张图片多次检测得到的FACE_TOKEN是同一个。
face_type	否	string	人脸的类型 **LIVE**：表示生活照：通常为手机、相机拍摄的人像图片、或从网络获取的人像图片等， **IDCARD**：表示身份证芯片照：二代身份证内置芯片中的人像照片， **WATERMARK**：表示带水印证件照：一般为带水印的小图，如公安网小图 **CERT**：表示证件照片：如拍摄的身份证、工卡、护照、学生证等证件图片 **INFRARED** 表示红外照片：使用红外相机拍摄的照片 **HYBRID**：表示混合类型，如果传递此值时会先对图片进行检测判断所属类型(生活照 or 证件照) （仅针对请求参数 image_type 为 BASE64 或 URL 时有效） **默认LIVE**
quality_control	否	string	图片质量控制 **NONE**: 不进行控制 **LOW**:较低的质量要求 **NORMAL**: 一般的质量要求 **HIGH**: 较高的质量要求 **默认 NONE** 若图片质量不满足要求，则返回结果中会提示质量检测失败
liveness_control	否	string	活体检测控制 **NONE**: 不进行控制 **LOW**:较低的活体要求(高通过率 低攻击拒绝率) **NORMAL**: 一般的活体要求(平衡的攻击拒绝率, 通过率) **HIGH**: 较高的活体要求(高攻击拒绝率 低通过率) **默认 NONE** 若活体检测结果不满足要求，则返回结果中会提示活体检测失败
face_sort_type	否	int	人脸检测排序类型 **0**:代表检测出的人脸按照人脸面积从大到小排列 **1**:代表检测出的人脸按照距离图片中心从近到远排列 **默认为0**
spoofing_control	否	string	合成图控制参数 **NONE**: 不进行控制 **LOW**:较低的合成图阈值数值，由于合成图判定逻辑为大于阈值视为合成图攻击，该项代表**低通过率、高攻击拒绝率** **NORMAL**: 一般的合成图阈值数值，由于合成图判定逻辑为大于阈值视为合成图攻击，该项代表**平衡的攻击拒绝率, 通过率** **HIGH**: 较高的合成图阈值数值，由于合成图判定逻辑为大于阈值视为合成图攻击，该项代表**高通过率、低攻击拒绝率** **默认为NONE**

说明：两张图片的上传使用json格式，如下：

```
[
  {
     "image": "sfasq35sadvsvqwr5q...",
     "image_type": "BASE64",
     "face_type": "LIVE",
     "quality_control": "LOW",
     "liveness_control": "HIGH"
   },
   {
     "image": "sfasq35sadvsvqwr5q...",
     "image_type": "BASE64",
     "face_type": "IDCARD",
     "quality_control": "LOW",
     "liveness_control": "HIGH"
   }
]
```

请求示例

提示一：使用示例代码前，请记得替换其中的示例Token、图片地址或Base64信息。

提示二：部分语言依赖的类或库，请在代码注释中查看下载地址。

Bash

PHP

JAVA

Python

Cpp

C#

人脸对比

```
curl -i -k 'https://aip.baidubce.com/rest/2.0/face/v3/match?access_token=【调用鉴权接口获取的token】' --data '[{"image": "sfasq35sadvsvqwr5q...", "image_type": "BASE64", "face_type": "LIVE", "quality_control": "LOW"}, {"image": "sfasq35sadvsvqwr5q...", "image_type": "BASE64", "face_type": "IDCARD", "quality_control": "LOW"}]' -H 'Content-Type:application/json; charset=UTF-8'
```

返回参数

参数名	必选	类型	说明
score	是	float	人脸相似度得分，推荐阈值80分
face_list	是	array	人脸信息列表
+face_token	是	string	人脸的唯一标志

返回示例

```
{
  "score": 44.3,
  "face_list": [  //返回的顺序与传入的顺序保持一致
     {
        "face_token": "fid1"
     },
     {
        "face_token": "fid2"
     }
  ]
}
```

- **质量控制参数说明**

> 不同的控制度下所对应的质量控制阈值，如果检测出来的质量信息某一项不符合控制阈值的要求，则会返回错误信息。

遮挡情况的阈值

控制度	left_eye	right_eye	nose	mouth	left_cheek	right_cheek	chin_contour
LOW	0.8	0.8	0.8	0.8	0.8	0.8	0.8
NORMAL	0.6	0.6	0.6	0.6	0.6	0.6	0.6
HIGH	0.2	0.2	0.2	0.2	0.2	0.2	0.2

光照、模糊度、完整度的阈值

控制度	illumination	blurdegree	completeness
LOW	20	0.8	0
NORMAL	40	0.6	0
HIGH	100	0.2	1

活体控制参数说明

> 不同的控制度下所对应的活体控制阈值，如果检测出来的活体分数小于控制阈值，则会返回错误信息。

控制度	阈值	说明
LOW	0.05	活体误拒率：万分之一；拒绝率：97.75%
NORMAL	0.3	活体误拒率：千分之一；拒绝率：98.82%
HIGH	0.9	活体误拒率：百分之一；拒绝率：99.77%

> 误拒率: 把**真人识别为假人**的概率. 阈值越高，安全性越高, 要求也就越高, 对应的误识率就越高。通过率=1-误拒率。

关于以上数值的概念介绍：

拒绝率（TRR）：如99%，代表100次作弊假体攻击，会有99次被拒绝。 误拒率（FRR）：如0.5%，指1000次真人请求，会有5次因为活体分数低于阈值被错误拒绝。 通过率（TAR）：如99%，指100次真人请求，会有99次因为活体分数高于阈值而通过。 阈值（Threshold）：高于此数值，则可判断为活体。

错误码

- **接口流控及鉴权错误码**

错误码	错误信息	描述	处理建议
2	Service temporarily unavailable	服务暂不可用	服务暂不可用，请再次请求，如果持续出现此类错误，请在控制台提交工单联系技术支持团队
4	Open api request limit reached	集群超限额	集群超限额，请再次请求，如果持续出现此类错误，请在控制台提交工单联系技术支持团队
6	no permission to access data	没有接口权限	请确认您调用的接口已经被赋权 常见问题是有V3版本权限，调用的是v2版本接口； 需要企业认证的，开通企业认证后一个小时左右即可使用。
17	Open api daily request limit reached	每天流量超限额	免费测试资源使用完毕，每天请求量超限额，已支持计费的接口，您可以在控制台人脸识别选择购买相关接口的次数包或开通按量后付费；邀测和未支持计费的接口，您可以在控制台提交工单申请提升限额
18	Open api qps request limit reached	QPS超限额	可直接自助购买更多QPS、联系商务接口人、或者提交工单
19	Open api total request limit reached	请求总量超限额	免费测试资源使用完毕，每天请求量超限额，已支持计费的接口，您可以在控制台人脸识别选择购买相关接口的次数包或开通按量后付费；邀测和未支持计费的接口，您可以在控制台提交工单申请提升限额
100	Invalid parameter	无效的access_token参数	token拉取失败，可以参考“Access Token获取”重新获取
110	Access token invalid or no longer valid	Access Token失效	**token有效期为30天，注意需要定期更换**，也可以每次请求都拉取新token
111	Access token expired	Access token过期	**token有效期为30天，注意需要定期更换**，也可以每次请求都拉取新token

- **人脸1：1对比错误码**

错误码	错误信息	描述	处理建议
222200	request body should be json format	1：请求体不是有效的 JSON 对象 2：请求体中字段参数值数据类型与接口文档要求不符	参考API文档说明，修改请求参数
222013	param[image] format error	参数格式错误	参考API文档说明，修改请求参数
222015	param[image_type] format error	参数格式错误	参考API文档说明，修改请求参数
222019	param[quality_control] format error	参数格式错误	参考API文档说明，修改请求参数
222020	param[liveness_control] format	参数格式错误	参考API文档说明，修改请求参数

222020	error	参数格式错误	参考API文档说明，修改请求参数
222024	param[face_type] format error	参数格式错误	参考API文档说明，修改请求参数
222039	param[face_sort_type] format error]	参数格式错误	参考API文档说明，修改请求参数
222043	param[display_corp_image] format error	参数格式错误	参考API文档说明，修改请求参数
223130	param[spoofing_control] format error	参数格式错误	参考API文档说明，修改请求参数
222202	pic not has face	图片中没有人脸	检查图片质量，确保存在人脸
222203	image check fail	图片无法解析人脸	检查图片质量
222204	image download fail	从图片的url下载图片失败	确认图片 URL 公网可访问（没有白名单限制）
222208	the number of image is incorrect	请求体中图片数量不对	多张图片请使用json格式传输
222209	face token not exist	face token不存在	请确认您操作的人脸已创建成功。 若face_token未注册到人脸库则有效期只有1小时 若注册到人脸库的face_token永久有效
222301	get face fail	获取人脸失败 备注：image_type为FACE_TOKEN模式时才会出现	请重试请求，如果持续出现此类错误，请提交工单
222304	image size is too large	图片尺寸太大	请确保图片尺寸不超过 6000 x 6000
223113	face is covered	人脸有被遮挡	提示用户请勿遮挡面部
223114	face is fuzzy	人脸模糊	提示用户请勿在拍摄人脸照时晃动手机
223115	face light is not good	人脸光照不好	提示用户到光线适宜的地方拍摄
223116	incomplete face	人脸不完整	提示用户请勿遮挡面部
223121	left eye is occlusion	质量检测未通过，左眼遮挡程度过高	提示用户请勿遮挡左眼
223122	right eye is occlusion	质量检测未通过，右眼遮挡程度过高	提示用户请勿遮挡右眼
223123	left cheek is occlusion	质量检测未通过，左脸遮挡程度过高	提示用户请勿遮挡左脸颊
223124	right cheek is occlusion	质量检测未通过，右脸遮挡程度过高	提示用户请勿遮挡右脸颊
223125	chin contour is occlusion	质量检测未通过，下巴遮挡程度过高	提示用户请勿遮挡下巴
223126	nose is occlusion	质量检测未通过，鼻子遮挡程度过高	提示用户请勿遮挡鼻子
223127	mouth is occlusion	质量检测未通过，嘴巴遮挡程度过高	提示用户请勿遮挡嘴巴
223129	face not forward	人脸未面向正前方（人脸的角度信息超出限制）	请使用面向正前方的人脸图片
223120	the first image liveness check fail	第一张图片人脸活体检测未通过	此图人脸活体检测结果为非活体
223120	the second image liveness check fail	第二张图片人脸活体检测未通过	此图人脸活体检测结果为非活体
223131	the first image spoofing check fail	第一张图片人脸合成图检测未通过	此图人脸合成图检测结果为合成图
223131	the second image spoofing check fail	第二张图片人脸合成图检测未通过	此图人脸合成图检测结果为合成图

222915	system busy	后端服务连接失败	请重试请求，若尝试多次无效，请提交工单咨询

人脸1：N搜索

能力介绍

业务能力

本文档为人脸1：N搜索API产品的使用说明文档。如果您业务上的图片主要由普通摄像头/抓拍机设备采集的大角度俯拍照片为主，建议您使用场景化搜索服务，查看文档详情

- **人脸1：N搜索**：也称为1：N识别，在指定人脸库集合中，找到最相似的人脸。
- **注意**：需要完成人脸1：N搜索，需配合**人脸库管理系列API**一同使用，首先构建一个人脸库，用于存放所有待比对的人脸特征，具体文档可参考**人脸库管理相关接口**文档说明；

> M：N识别的原理，相当于在多个人脸的图片中，先分别找出所有人脸，然后分别在待查找的人脸集合中，分别做1：N识别，最后将识别结果汇总在一起进行返回。
>
> 若人脸库超过1年未使用（无入库、搜索等操作），平台将对相关人脸库资源进行释放，以确保用户信息安全。

人脸1：N搜索

在线调试 您可以在 示例代码中心 中调试该接口，可进行签名验证、查看在线调用的请求内容和返回结果、示例代码的自动生成。

请求说明

注意事项：

- **请求体格式化**：Content-Type为`application/json`，通过`json`格式化请求体。
- **Base64编码**：请求的图片需经过`Base64编码`，图片的base64编码指将图片数据编码成一串字符串，使用该字符串代替图像地址。您可以首先得到图片的二进制，然后用Base64格式编码即可。需要注意的是，图片的base64编码是不包含图片头的，如``。
- **图片格式**：现支持PNG、JPG、JPEG、BMP，**不支持GIF图片**。

请求示例

HTTP方法：`POST`

请求URL： `https://aip.baidubce.com/rest/2.0/face/v3/search`

URL参数：

参数	值
access_token	通过API Key和Secret Key获取的access_token,参考“Access Token获取”

Header如下：

参数	值
Content-Type	application/json

Body中放置请求参数，参数详情如下：

请求参数

参数	必选	类型	说明
image	是	string	图片信息，图片上传方式根据image_type来判断，**为base64时，编码后图片大小不超过2M，分辨率应小于1920*1080**
image_type	是	string	图片类型 **BASE64**:（**推荐**）图片的base64值，base64编码后的图片数据，**编码后的图片大小不超过2M**； **FACE_TOKEN**: 人脸图片的唯一标识，调用人脸检测接口时，会为每个人脸图片赋予一个唯一的FACE_TOKEN，同一张图片多次检测得到的FACE_TOKEN是同一个。
group_id_list	是	string	从指定的group中进行查找 用逗号分隔，**上限10个**
quality_control	否	string	图片质量控制 **NONE**: 不进行控制 **LOW**:较低的质量要求 **NORMAL**: 一般的质量要求 **HIGH**: 较高的质量要求 **默认 NONE** 若图片质量不满足要求，则返回结果中会提示质量检测失败
liveness_control	否	string	活体检测控制 **NONE**: 不进行控制 **LOW**:较低的活体要求(高通过率 低攻击拒绝率) **NORMAL**: 一般的活体要求(平衡的攻击拒绝率, 通过率) **HIGH**: 较高的活体要求(高攻击拒绝率 低通过率) **默认NONE** 若活体检测结果不满足要求，则返回结果中会提示活体检测失败
spoofing_control	否	string	合成图控制 **NONE**: 不进行控制 **LOW**:较低的合成图阈值数值，由于合成图判定逻辑为大于阈值视为合成图攻击，该项代表**低通过率、高攻击拒绝率** **NORMAL**: 一般的合成图阈值数值，由于合成图判定逻辑为大于阈值视为合成图攻击，该项代表**平衡的攻击拒绝率, 通过率** **HIGH**: 较高的合成图阈值数值，由于合成图判定逻辑为大于阈值视为合成图攻击，该项代表**高通过率、低攻击拒绝率**) **默认为NONE**
user_id	否	string	当需要对特定用户进行比对时，指定user_id进行比对。即人脸认证功能。
max_user_num	否	unit32	查找后返回的用户数量。返回相似度最高的几个用户，默认为1，最多返回50个。
face_sort_type	否	int	人脸检测排序类型 **0**:代表检测出的人脸按照人脸面积从大到小排列 **1**:代表检测出的人脸按照距离图片中心从近到远排列 **默认为0**
match_threshold	否	int	匹配阈值（设置阈值后，score低于此阈值的用户信息将不会返回） 最大100 最小0 默认0 **此阈值设置得越高，检索速度将会越快，推荐使用阈值80**

> 说明：如果使用base 64格式的图片，两张请求的图片请分别进行base64编码。

示例代码

提示一：使用示例代码前，请记得替换其中的示例Token、图片地址或Base64信息。

提示二：部分语言依赖的类或库，请在代码注释中查看下载地址。

Bash

PHP

JAVA

Python

Cpp

C#

人脸搜索

```
curl -i -k 'https://aip.baidubce.com/rest/2.0/face/v3/search?access_token=【调用鉴权接口获取的token】' --data '{"image":"027d8308a2ec665acb1bdf63e513bcb9","image_type":"FACE_TOKEN","group_id_list":"group_repeat,group_233' -H 'Content-Type:application/json; charset=UTF-8'
```

返回说明

返回参数

- 返回结果

字段	必选	类型	说明
face_token	是	string	人脸图片的唯一标识（有效期60min）。**此token由用于搜索的人脸图片生成，非搜索到的人脸face_token**
user_list	是	array	匹配的用户信息列表
+group_id	是	string	用户所属的group_id
+user_id	是	string	用户的user_id
+user_info	是	string	注册用户时携带的user_info
+score	是	float	用户的匹配得分，推荐阈值80分

- 返回示例

```
{
 "face_token": "fid",
 "user_list": [
   {
     "group_id" : "test1",
     "user_id": "u333333",
     "user_info": "Test User",
     "score": 99.3
   }
 ]
}
```

- **质量控制参数说明**

> 不同的控制度下所对应的质量控制阈值，如果检测出来的质量信息某一项不符合控制阈值的要求，则会返回错误信息。

遮挡情况的阈值

控制度	left_eye	right_eye	nose	mouth	left_cheek	right_cheek	chin_contour
LOW	0.8	0.8	0.8	0.8	0.8	0.8	0.8
NORMAL	0.6	0.6	0.6	0.6	0.6	0.6	0.6
HIGH	0.2	0.2	0.2	0.2	0.2	0.2	0.2

模糊度、完整度的阈值

控制度	illumination	blurdegree	completeness
LOW	20	0.8	0
NORMAL	40	0.6	0
HIGH	100	0.2	1

活体控制参数说明

> 不同的控制度下所对应的活体控制阈值，如果检测出来的活体分数小于控制阈值，则会返回错误信息。

控制度	阈值	说明
LOW	0.05	活体误拒率：万分之一；拒绝率：63.9%
NORMAL	0.3	活体误拒率：千分之一；拒绝率：90.3%
HIGH	0.9	活体误拒率：百分之一；拒绝率：97.6%

> 1、误拒率: 把**真人识别为假人**的概率. 阈值越高，安全性越高, 要求也就越高, 对应的误识率就越高 2、通过率=1-误拒率

关于以上数值的概念介绍：

拒绝率（TRR）：如99%，代表100次作弊假体攻击，会有99次被拒绝。 误拒率（FRR）：如0.5%，指1000次真人请求，会有5次因为活体分数低于阈值被错误拒绝。 通过率（TAR）：如99%，指100次真人请求，会有99次因为活体分数高于阈值而通过。 阈值（Threshold）：高于此数值，则可判断为活体。

合成图控制参数说明

> 不同的控制度下所对应的合成图检测（PS、人脸融合等）阈值，如果检测出来的分数大于控制阈值，则会返回错误信息。

控制度	阈值	误拒率（FRR）	通过率	攻击拒绝率（TRR））
LOW	0.00023	5%	95%	94.93%
NORMAL（推荐）	0.00048	1%	99%	89.71%
HIGH	0.00109	0.1%	99.9%	84.57%

> 1、误拒率：把**正常图片识别为合成图片**的概率。阈值越低，安全性越高，要求也就越高，对应的误识率就越高。 2、通过率=1-误拒率

关于以上数值的概念介绍：

阈值（Threshold）：高于此数值，则可判断为是合成图攻击。

人脸M：N搜索

业务能力

本文档为**人脸M:N搜索API**的使用说明文档。如果您业务上的图片主要由普通摄像头/抓拍机设备采集的大角度俯拍照片为主，建议您使用场景化搜索服务，查看文档详情

- **人脸M：N搜索**：也称为M：N识别，待识别图片中含有多个人脸时，在指定人脸库集合中，找到与待识别图片中的多个人脸分别最相似的人脸。
- **注意**：需要完成人脸M：N搜索，需配合**人脸库管理系列API**一同使用，首先构建一个人脸库，用于存放所有待比对的人脸特征，具体文档可参考**人脸库管理相关接口**文档说明.

> M：N识别的原理，相当于从待比对的多人脸图片中，先分别找出所有人脸，然后将这些人脸信息分别在人脸库中进行1：N搜索，最后将所有搜索结果汇总在一起进行返回。
>
> 若人脸库超过1年未使用（无入库、搜索等操作），平台将对相关人脸库资源进行释放，以确保用户信息安全。

在线调试

您可以在 示例代码中心 中调试该接口，可进行签名验证、查看在线调用的请求内容和返回结果、示例代码的自动生成。

请求说明

注意事项：

- **请求体格式化**：Content-Type为`application/json`，通过`json`格式化请求体。
- **Base64编码**：请求的图片需经过`Base64编码`，图片的base64编码指将图片数据编码成一串字符串，使用该字符串代替图像地址。您可以首先得到图片的二进制，然后用Base64格式编码即可。需要注意的是，图片的base64编码是不包含图片头的，如``。
- **图片格式**：现支持PNG、JPG、JPEG、BMP，**不支持GIF图片**。

请求示例

HTTP方法：POST

请求URL： `https://aip.baidubce.com/rest/2.0/face/v3/multi-search`

URL参数：

参数	值
access_token	通过API Key和Secret Key获取的access_token,参考“Access Token获取”

Header如下：

参数	值
Content-Type	application/json

Body中放置请求参数，参数详情如下：

请求参数

参数	必选	类型	说明
image	是	string	图片信息(数据大小应小于10M 分辨率应小于1920*1080)
image_type	是	string	图片类型 **BASE64**:（**推荐**）图片的base64值; **FACE_TOKEN**: face_token 人脸标识
group_id_list	是	string	从指定的group中进行查找 用逗号分隔，**上限10个**
max_face_num	否	int	最多处理人脸的数目 **默认值为1(仅检测图片中面积最大的那个人脸) 最大值10**
match_threshold	否	int	匹配阈值（设置阈值后，score低于此阈值的用户信息将不会返回） 最大100 最小0 默认80 **此阈值设置得越高，检索速度将会越快，推荐使用默认阈值80**
quality_control	否	string	质量控制(质量不符合要求的人脸不会出现在返回结果中) **NONE**: 不进行控制 **LOW**:较低的质量要求 **NORMAL**: 一般的质量要求 **HIGH**: 较高的质量要求 **默认NONE**
liveness_control	否	string	活体控制(活体分数不符合要求的人脸不会出现在返回结果中) **NONE**: 不进行控制 **LOW**:较低的活体要求(高通过率 低攻击拒绝率) **NORMAL**: 一般的活体要求(平衡的攻击拒绝率, 通过率) **HIGH**: 较高的活体要求(高攻击拒绝率 低通过率) **默认NONE**
spoofing_control	否	string	合成图控制 **NONE**: 不进行控制 **LOW**:较低的合成图阈值数值，由于合成图判定逻辑为大于阈值视为合成图攻击，该项代表**低通过率、高攻击拒绝率** **NORMAL**: 一般的合成图阈值数值，由于合成图判定逻辑为大于阈值视为合成图攻击，该项代表**平衡的攻击拒绝率, 通过率** **HIGH**: 较高的合成图阈值数值，由于合成图判定逻辑为大于阈值视为合成图攻击，该项代表**高通过率、低攻击拒绝率**) **默认为NONE**
max_user_num	否	unit32	识别返回的最大用户数，默认为1，最大20个

请求示例

```
{
    "image": "/9j/4AAQSkZJRgABAQAAAQABAAD/2wBDAAgGBgcGBQgHBwcJCQgKD...",
    "image_type": "BASE64",
    "group_id_list": "group1",
    "max_face_num" : 5,
    "quality_control": "LOW",
    "liveness_control": "NORMAL"
}
```

返回说明

返回参数

字段	必选	类型	说明
log_id	是	uint64	请求标识码，随机数，唯一
face_num	是	int	图片中的人脸数量
face_list	是	array	人脸信息列表
+face_token	是	string	人脸图片的唯一标识（有效期60min）。**此token由用于搜索的人脸图片生成，非搜索到的人脸face_token**
+location	是	array	人脸在图片中的位置
++left	是	double	人脸区域离左边界的距离
++top	是	double	人脸区域离上边界的距离
++width	是	double	人脸区域的宽度
++height	是	double	人脸区域的高度
++rotation	是	int64	人脸框相对于竖直方向的顺时针旋转角，[-180,180]
+user_list	是	array	匹配的用户信息列表
++group_id	是	string	用户所属的group_id
++user_id	是	string	用户的user_id
++user_info	是	string	注册用户时携带的user_info
++score	是	float	用户的匹配得分 **80分以上可以判断为同一人，此分值对应万分之一误识率**

返回示例

```
{
  "error_code": 0,
  "error_msg": "SUCCESS",
  "log_id": 240483475,
  "timestamp": 1535533440,
  "cached": 0,
  "result": {
    "face_num": 2,
    "face_list": [
      {
        "face_token": "6fe19a6ee0c4233db9b5bba4dc2b9233",
        "location": {
          "left": 31.95568085,
          "top": 120.3764267,
          "width": 87,
          "height": 85,
          "rotation": -5
        },
        "user_list": [
          {
            "group_id": "group1",
            "user_id": "5abd24fd062e49bfa906b257ec40d284",
            "user_info": "userinfo1",
            "score": 69.85684967041
          },
          {
            "group_id": "group1",
            "user_id": "2abf89cffb31473a9948268fde9e1c3f",
            "user_info": "userinfo2",
            "score": 66.586112976074
          }
        ]
      },
      {
        "face_token": "fde61e9c074f48cf2bbb319e42634f41",
        "location": {
          "left": 219.4467773,
          "top": 104.7486954,
          "width": 81,
          "height": 77,
          "rotation": 3
        },
        "user_list": [
          {
            "group_id": "group1",
            "user_id": "088717532b094c3990755e91250adf7d",
            "user_info": "userinfo",
            "score": 65.154159545898
          }
        ]
      }
    ]
  }
}
```


- **质量控制参数说明**

> 不同的控制度下所对应的质量控制阈值，如果检测出来的质量信息某一项不符合控制阈值的要求，则会返回错误信息。

遮挡情况的阈值

控制度	left_eye	right_eye	nose	mouth	left_cheek	right_cheek	chin_contour
LOW	0.8	0.8	0.8	0.8	0.8	0.8	0.8
NORMAL	0.6	0.6	0.6	0.6	0.6	0.6	0.6
HIGH	0.2	0.2	0.2	0.2	0.2	0.2	0.2

模糊度、完整度的阈值

控制度	illumination	blurdegree	completeness
LOW	20	0.8	0
NORMAL	40	0.6	0
HIGH	100	0.2	1

活体控制参数说明

> 不同的控制度下所对应的活体控制阈值，如果检测出来的活体分数小于控制阈值，则会返回错误信息。

控制度	阈值	说明
LOW	0.05	活体误拒率：万分之一；拒绝率：97.75%
NORMAL	0.3	活体误拒率：千分之一；拒绝率：98.82%
HIGH	0.9	活体误拒率：百分之一；拒绝率：99.77%

> 1、误拒率: 把**真人识别为假人**的概率. 阈值越高，安全性越高, 要求也就越高, 对应的误识率就越高
> 2、通过率=1-误拒率

关于以上数值的概念介绍：

拒绝率（TRR）：如99%，代表100次作弊假体攻击，会有99次被拒绝。 误拒率（FRR）：如0.5%，指1000次真人请求，会有5次因为活体分数低于阈值被错误拒绝。 通过率（TAR）：如99%，指100次真人请求，会有99次因为活体分数高于阈值而通过。 阈值（Threshold）：高于此数值，则可判断为活体。

合成图控制参数说明

> 不同的控制度下所对应的合成图检测（PS、人脸融合等）阈值，如果检测出来的分数大于控制阈值，则会返回错误信息。

控制度	阈值	误拒率（FRR）	通过率	攻击拒绝率（TRR））
LOW	0.00023	5%	95%	94.93%
NORMAL（推荐）	0.00048	1%	99%	89.71%
HIGH	0.00109	0.1%	99.9%	84.57%

> 1、误拒率：把**正常图片识别为合成图片**的概率。阈值越低，安全性越高，要求也就越高，对应的误识率就越高。 2、通过率=1-误拒率

关于以上数值的概念介绍：

阈值（Threshold）：高于此数值，则可判断为是合成图攻击。

人脸库管理系列接口

业务能力

本文档为**人脸库管理系列接口**的使用说明文档。

人脸库管理接口，主要用于配合人脸1：N搜搜及人脸M：N检索使用，要完成1：N搜索M：N搜索，首先需要构建一个人脸库，用于存放所有人脸特征，相关接口如下：

- **人脸注册**：向人脸库中添加人脸
- **人脸更新**：更新人脸库中指定用户下的人脸信息
- **人脸删除**：删除指定用户的某张人脸
- **用户信息查询**：查询人脸库中某个用户的详细信息
- **获取用户人脸列表**：获取某个用户组中的全部人脸列表
- **获取用户列表**：查询指定用户组中的用户列表
- **复制用户**：将指定用户复制到另外的人脸组
- **删除用户**：删除指定用户
- **创建用户组**：创建一个新的用户组
- **删除用户组**：删除指定用户组
- **组列表查询**：查询人脸库中用户组的列表

人脸库结构

人脸库、用户组、用户、用户下的人脸**层级关系**如下所示：

```
|- 人脸库(appid)
 |- 用户组一（group_id）
    |- 用户01（uid）
       |- 人脸（faceid）
    |- 用户02（uid）
       |- 人脸（faceid）
       |- 人脸（faceid）
       ....
     ....
 |- 用户组二（group_id）
 |- 用户组三（group_id）
 ....
```

关于人脸库的设置限制

- **每个appid对应一个人脸库，且不同appid之间，人脸库互不相通**。
- 每个人脸库下，可以创建多个用户组，用户组（group）数量**没有限制**。
- 每个用户组（group）下，可添加**无限**个user_id，**无限**张人脸（注：为了保证查询速度，单个group中的人脸容量上限建议为**80万**）。
- 每个用户（user_id）所能注册的最大人脸数量**20**。

> 提醒：每个人脸库对应一个appid，一定确保不要轻易删除后台应用列表中的appid，删除后则此人脸库将失效，无法进行任何查找！

质量判断

为了保证识别效果，请控制注册人脸的质量，在调用人脸注册接口时需要使用质量控制和活体控制参数，从而保证图片的质量以及注册进入人脸库的人脸是活体。

人脸注册

在线调试 您可以在 示例代码中心 中调试该接口，可进行签名验证、查看在线调用的请求内容和返回结果、示例代码的自动生成。

请求说明

注意事项：

- **请求体格式化**：Content-Type为`application/json`，通过`json`格式化请求体。
- **Base64编码**：请求的图片需经过`Base64编码`，图片的base64编码指将图片数据编码成一串字符串，使用该字符串代替图像地址。您可以首先得到图片的二进制，然后用Base64格式编码即可。需要注意的是，图片的base64编码是不包含图片头的，如``
- **图片格式**：现支持PNG、JPG、JPEG、BMP，**不支持GIF图片**

请求示例

HTTP方法：`POST`

请求URL：`https://aip.baidubce.com/rest/2.0/face/v3/faceset/user/add`

URL参数：

参数	值
access_token	通过API Key和Secret Key获取的access_token,参考“Access Token获取”

Header如下：

参数	值
Content-Type	application/json

Body中放置请求参数，参数详情如下：

请求参数

参数	必选	类型	说明
image	是	string	图片信息(**总数据大小应小于10M，分辨率应小于1920*1080**)，图片上传方式根据image_type来判断。 注：组内每个uid下的人脸图片数目上限为20张
image_type	是	string	图片类型 **BASE64**:（**推荐**）图片的base64值，base64编码后的图片数据，编码后的图片大小不超过2M； **FACE_TOKEN**：人脸图片的唯一标识，调用人脸检测接口时，会为每个人脸图片赋予一个唯一的FACE_TOKEN，同一张图片多次检测得到的FACE_TOKEN是同一个。
group_id	是	string	用户组id，标识一组用户（由数字、字母、下划线组成），长度限制48B。**产品建议**：根据您的业务需求，可以将需要注册的用户，按照业务划分，分配到不同的group下，例如按照会员手机尾号作为groupid，用于刷脸支付、会员计费消费等，这样可以尽可能控制每个group下的用户数与人脸数，提升检索的准确率
user_id	是	string	用户id（由数字、字母、下划线组成），长度限制48B
user_info	否	string	用户资料，长度限制256B 默认空
quality_control	否	string	图片质量控制 **NONE**: 不进行控制 **LOW**:较低的质量要求 **NORMAL**: 一般的质量要求 **HIGH**: 较高的质量要求 **默认 NONE** 若图片质量不满足要求，则返回结果中会提示质量检测失败
liveness_control	否	string	活体检测控制 **NONE**: 不进行控制 **LOW**:较低的活体要求(高通过率 低攻击拒绝率) **NORMAL**: 一般的活体要求(平衡的攻击拒绝率, 通过率) **HIGH**: 较高的活体要求(高攻击拒绝率 低通过率) **默认NONE** 若活体检测结果不满足要求，则返回结果中会提示活体检测失败
spoofing_control	否	string	合成图控制 **NONE**: 不进行控制 **LOW**:较低的合成图阈值数值，由于合成图判定逻辑为大于阈值视为合成图攻击，该项代表**低通过率、高攻击拒绝率** **NORMAL**: 一般的合成图阈值数值，由于合成图判定逻辑为大于阈值视为合成图攻击，该项代表**平衡的攻击拒绝率, 通过率** **HIGH**: 较高的合成图阈值数值，由于合成图判定逻辑为大于阈值视为合成图攻击，该项代表**高通过率、低攻击拒绝率**) **默认为NONE**
action_type	否	string	**操作方式** APPEND: 当user_id在库中已经存在时，对此user_id重复注册时，新注册的图片默认会追加到该user_id下 REPLACE：当对此user_id重复注册时,则会用新图替换库中该user_id下所有图片 默认使用APPEND
face_sort_type	否	int	人脸检测排序类型 **0**:代表检测出的人脸按照人脸面积从大到小排列 **1**:代表检测出的人脸按照距离图片中心从近到远排列 **默认为0**

> **说明**：人脸注册完毕后，生效时间**一般为5s以内**，之后便可以进行人脸搜索或认证操作。

示例代码

提示一：使用示例代码前，请记得替换其中的示例Token、图片地址或Base64信息。

提示二：部分语言依赖的类或库，请在代码注释中查看下载地址。

Bash

PHP

JAVA

Python

Cpp

C#

```
 人脸注册
curl -i -k 'https://aip.baidubce.com/rest/2.0/face/v3/faceset/user/add?access_token=【调用鉴权接口获取的token】' --data '{"image":"027d8308a2ec665acb1bdf63e513bcb9","image_type":"FACE_TOKEN","group_id":"group_repeat","user_id":"us' -H 'Content-Type:application/json; charset=UTF-8'
```

返回说明

返回参数

字段	必选	类型	说明
log_id	是	uint64	请求标识码，随机数，唯一
face_token	是	string	人脸图片的唯一标识（有效期永久）
location	是	array	人脸在图片中的位置
+left	是	double	人脸区域离左边界的距离
+top	是	double	人脸区域离上边界的距离
+width	是	double	人脸区域的宽度
+height	是	double	人脸区域的高度
+rotation	是	int64	人脸框相对于竖直方向的顺时针旋转角，[-180,180]

返回示例

```
{
 "face_token": "2fa64a88a9d5118916f9a303782a97d3",
 "location": {
    "left": 117,
    "top": 131,
    "width": 172,
    "height": 170,
    "rotation": 4
 }
}
```


- **质量控制参数说明**

> 不同的控制度下所对应的质量控制阈值，如果检测出来的质量信息某一项不符合控制阈值的要求，则会返回错误信息。

遮挡情况的阈值

控制度	left_eye	right_eye	nose	mouth	left_cheek	right_cheek	chin_contour
LOW	0.8	0.8	0.8	0.8	0.8	0.8	0.8
NORMAL	0.6	0.6	0.6	0.6	0.6	0.6	0.6
HIGH	0.2	0.2	0.2	0.2	0.2	0.2	0.2

模糊度、完整度的阈值

控制度	illumination	blurdegree	completeness
LOW	20	0.8	0
NORMAL	40	0.6	0
HIGH	100	0.2	1

活体控制参数说明

> 不同的控制度下所对应的活体控制阈值不同，如果检测出来的活体分数小于控制阈值，则会返回错误信息。

控制度	阈值	说明
LOW	0.05	活体误拒率：万分之一；拒绝率：97.75%
NORMAL	0.3	活体误拒率：千分之一；拒绝率：98.82%
HIGH	0.9	活体误拒率：百分之一；拒绝率：99.77%

> 1、误拒率: 把**真人识别为假人**的概率. 阈值越高，安全性越高, 要求也就越高, 对应的误识率就越高 2、通过率=1-误拒率

关于以上数值的概念介绍：

拒绝率（TRR）：如99%，代表100次作弊假体攻击，会有99次被拒绝。 误拒率（FRR）：如0.5%，指1000次真人请求，会有5次因为活体分数低于阈值被错误拒绝。 通过率（TAR）：如99%，指100次真人请求，会有99次因为活体分数高于阈值而通过。 阈值（Threshold）：高于此数值，则可判断为活体。

合成图控制参数说明

> 不同的控制度下所对应的合成图检测（PS、人脸融合等）阈值，如果检测出来的分数大于控制阈值，则会返回错误信息。

控制度	阈值	误拒率（FRR）	通过率	攻击拒绝率（TRR））
LOW	0.00023	5%	95%	94.93%
NORMAL（推荐）	0.00048	1%	99%	89.71%
HIGH	0.00109	0.1%	99.9%	84.57%

> 1、误拒率：把**正常图片识别为合成图片**的概率。阈值越低，安全性越高，要求也就越高，对应的误识率就越高。 2、通过率=1-误拒率

关于以上数值的概念介绍：

阈值（Threshold）：高于此数值，则可判断为是合成图攻击。

人脸更新

接口描述

用于对人脸库中指定用户，更新其下的人脸图像。

> **说明**：针对一个user_id执行更新操作，新上传的人脸图像将覆盖此user_id原有所有图像。

在线调试 您可以在示例代码中心中调试该接口，可进行签名验证、查看在线调用的请求内容和返回结果、示例代码的自动生成。

请求说明

注意事项：

- **请求体格式化**：Content-Type为`application/json`，通过`json`格式化请求体。
- **Base64编码**：请求的图片需经过`Base64编码`，图片的base64编码指将图片数据编码成一串字符串，使用该字符串代替图像地址。您可以首先得到图片的二进制，然后用Base64格式编码即可。需要注意的是，图片的base64编码是不包含图片头的，如``
- **图片格式**：现支持PNG、JPG、JPEG、BMP，**不支持GIF图片**

请求示例

HTTP方法：`POST`

请求URL：`https://aip.baidubce.com/rest/2.0/face/v3/faceset/user/update`

URL参数：

参数	值
access_token	通过API Key和Secret Key获取的access_token,参考“Access Token获取”

Header如下：

参数	值
Content-Type	application/json

Body中放置请求参数，参数详情如下：

请求参数

参数	必选	类型	说明
image	是	string	图片信息(**总数据大小应小于10M，分辨率应小于1920*1080**)，图片上传方式根据image_type来判断
image_type	是	string	图片类型 **BASE64**:（**推荐**）图片的base64值，base64编码后的图片数据，编码后的图片大小不超过2M； **FACE_TOKEN**: 人脸图片的唯一标识
group_id	是	string	用户组id，标识一组用户（由数字、字母、下划线组成），长度限制48B
user_id	是	string	用户id（由数字、字母、下划线组成），长度限制48B
user_info	否	string	用户资料，长度限制48B 默认空
quality_control	否	string	图片质量控制 **NONE**: 不进行控制 **LOW**:较低的质量要求 **NORMAL**: 一般的质量要求 **HIGH**: 较高的质量要求 **默认 NONE** 若图片质量不满足要求，则返回结果中会提示质量检测失败
liveness_control	否	string	活体检测控制 **NONE**: 不进行控制 **LOW**:较低的活体要求(高通过率 低攻击拒绝率) **NORMAL**: 一般的活体要求(平衡的攻击拒绝率, 通过率) **HIGH**: 较高的活体要求(高攻击拒绝率 低通过率) **默认NONE** 若活体检测结果不满足要求，则返回结果中会提示活体检测失败
spoofing_control	否	string	合成图控制 **NONE**: 不进行控制 **LOW**:较低的合成图阈值数值，由于合成图判定逻辑为大于阈值视为合成图攻击，该项代表**低通过率、高攻击拒绝率** **NORMAL**: 一般的合成图阈值数值，由于合成图判定逻辑为大于阈值视为合成图攻击，该项代表**平衡的攻击拒绝率, 通过率** **HIGH**: 较高的合成图阈值数值，由于合成图判定逻辑为大于阈值视为合成图攻击，该项代表**高通过率、低攻击拒绝率**) **默认为NONE**
action_type	否	string	**操作方式** UPDATE: 会使用新图替换库中该user_id下所有图片, 若user_id不存在则会报错 REPLACE：当user_id不存在时, 则会注册这个user_id的用户 默认使用UPDATE

示例代码

提示一：使用示例代码前，请记得替换其中的示例Token、图片地址或Base64信息。

提示二：部分语言依赖的类或库，请在代码注释中查看下载地址。

```
Bash
```

PHP

JAVA

Python

Cpp

C#

人脸更新

```
curl -i -k 'https://aip.baidubce.com/rest/2.0/face/v3/faceset/user/update?access_token=【调用鉴权接口获取的token】' --data '{"image":"027d8308a2ec665acb1bdf63e513bcb9","image_type":"FACE_TOKEN","group_id":"group_repeat","user_id":"us' -H 'Content-Type:application/json; charset=UTF-8'
```

返回说明

返回参数

字段	必选	类型	说明
log_id	是	uint64	请求标识码，随机数，唯一
face_token	是	string	人脸图片的唯一标识（有效期永久）
location	是	array	人脸在图片中的位置
+left	是	double	人脸区域离左边界的距离
+top	是	double	人脸区域离上边界的距离
+width	是	double	人脸区域的宽度
+height	是	double	人脸区域的高度
+rotation	是	int64	人脸框相对于竖直方向的顺时针旋转角，[-180,180]

返回示例

```
{
 "face_token": "2fa64a88a9d5118916f9a303782a97d3",
 "location": {
    "left": 117,
    "top": 131,
    "width": 172,
    "height": 170,
    "rotation": 4
 }
}
```

- **质量控制参数说明**

> 不同的控制度下所对应的质量控制阈值，如果检测出来的质量信息某一项不符合控制阈值的要求，则会返回错误信息。

遮挡情况的阈值

控制度	left_eye	right_eye	nose	mouth	left_cheek	right_cheek	chin_contour
LOW	0.8	0.8	0.8	0.8	0.8	0.8	0.8
NORMAL	0.6	0.6	0.6	0.6	0.6	0.6	0.6
HIGH	0.2	0.2	0.2	0.2	0.2	0.2	0.2

模糊度、完整度的阈值

控制度	illumination	blurdegree	completeness
LOW	20	0.8	0
NORMAL	40	0.6	0
HIGH	100	0.2	1

活体控制参数说明

> 不同的控制度下所对应的活体控制阈值不同，如果检测出来的活体分数小于控制阈值，则会返回错误信息。

控制度	阈值	说明
LOW	0.05	活体误拒率：万分之一；拒绝率：97.75%
NORMAL	0.3	活体误拒率：千分之一；拒绝率：98.82%
HIGH	0.9	活体误拒率：百分之一；拒绝率：99.77%

> 1、误拒率: 把**真人识别为假人**的概率. 阈值越高，安全性越高, 要求也就越高, 对应的误识率就越高 2、通过率=1-误拒率

关于以上数值的概念介绍：

拒绝率（TRR）：如99%，代表100次作弊假体攻击，会有99次被拒绝。 误拒率（FRR）：如0.5%，指1000次真人请求，会有5次因为活体分数低于阈值被错误拒绝。 通过率（TAR）：如99%，指100次真人请求，会有99次因为活体分数高于阈值而通过。 阈值（Threshold）：高于此数值，则可判断为活体。

合成图控制参数说明

> 不同的控制度下所对应的合成图检测（PS、人脸融合等）阈值，如果检测出来的分数大于控制阈值，则会返回错误信息。

控制度	阈值	误拒率（FRR）	通过率	攻击拒绝率（TRR））
LOW	0.00023	5%	95%	94.93%
NORMAL（推荐）	0.00048	1%	99%	89.71%
HIGH	0.00109	0.1%	99.9%	84.57%

> 1、误拒率：把**正常图片识别为合成图片**的概率。阈值越低，安全性越高，要求也就越高，对应的误识率就越高。 2、通过

> 率=1-误拒率

关于以上数值的概念介绍：

阈值（Threshold）：高于此数值，则可判断为是合成图攻击。

人脸删除

接口描述

删除用户的某一张人脸，如果该用户只有一张人脸图片，则同时删除用户。

在线调试 您可以在示例代码中心中调试该接口，可进行签名验证、查看在线调用的请求内容和返回结果、示例代码的自动生成。

请求说明

请求示例

HTTP方法：`POST`

请求URL：`https://aip.baidubce.com/rest/2.0/face/v3/faceset/face/delete`

URL参数：

参数	值
access_token	通过API Key和Secret Key获取的access_token,参考“Access Token获取”

Header如下：

参数	值
Content-Type	application/json

Body中放置请求参数，参数详情如下：

请求参数

参数	必选	类型	说明
log_id	是	uint64	请求标识码，随机数，唯一
user_id	是	string	用户id（由数字、字母、下划线组成），长度限制48B
group_id	是	string	用户组id（由数字、字母、下划线组成） 长度限制48B，删除指定group_id中的user_id信息
face_token	是	string	需要删除的人脸图片token，（由数字、字母、下划线组成）长度限制64B

返回说明

通过返回的error_code判断是否成功 如失败则查看error_msg获得具体错误信息

返回示例

```
 // 删除成功
{
  "error_code": 0,
  "log_id": 73473737,
}
// 删除发生错误
{
  "error_code": 223106,
  "log_id": 1382953199,
  "error_msg": "face is not exist"
}
```

用户信息查询

接口描述

获取人脸库中某个用户的信息(user_info信息和用户所属的组)。

在线调试 您可以在示例代码中心中调试该接口，可进行签名验证、查看在线调用的请求内容和返回结果、示例代码的自动生成。

请求说明

请求示例

HTTP方法：`POST`

请求URL： `https://aip.baidubce.com/rest/2.0/face/v3/faceset/user/get`

URL参数：

参数	值
access_token	通过API Key和Secret Key获取的access_token,参考“Access Token获取”

Header如下：

参数	值
Content-Type	application/json

Body中放置请求参数，参数详情如下：

请求参数

参数	必选	类型	说明
user_id	是	string	用户id（由数字、字母、下划线组成），长度限制48B
group_id	是	string	用户组id(由数字、字母、下划线组成，长度限制48B)，如传入“**@ALL**”则从所有组中查询用户信息。注：处于不同组，但uid相同的用户，我们认为是同一个用户。

示例代码

提示一：使用示例代码前，请记得替换其中的示例Token、图片地址或Base64信息。

提示二：部分语言依赖的类或库，请在代码注释中查看下载地址。

Bash

PHP

JAVA

Python

Cpp

C#

用户信息查询

```
curl -i -k 'https://aip.baidubce.com/rest/2.0/face/v3/faceset/user/get?access_token=【调用鉴权接口获取的token】' --data '{"user_id":"user1","group_id":"group1"}' -H 'Content-Type:application/json; charset=UTF-8'
```

返回说明

返回参数

字段	必选	类型	说明
log_id	是	uint64	请求标识码，随机数，唯一
user_list	是	array	查询到的用户列表
+user_info	是	string	用户资料，被查询用户的资料
+group_id	是	string	用户组id，被查询用户的所在组

返回示例

```
{
 "user_list": [
    {
        "user_info": "user info ...",
        "group_id": "gid1"
    },
    {
        "user_info": "user info2 ...",
        "group_id": "gid2"
    }
 ]
}
```

获取用户人脸列表

接口描述

用于获取一个用户的全部人脸列表。

在线调试 您可以在示例代码中心中调试该接口，可进行签名验证、查看在线调用的请求内容和返回结果、示例代码的自动生成。

请求说明

请求示例

HTTP方法：`POST`

请求URL： `https://aip.baidubce.com/rest/2.0/face/v3/faceset/face/getlist`

URL参数：

参数	值
access_token	通过API Key和Secret Key获取的access_token,参考“Access Token获取”

Header如下：

参数	值
Content-Type	application/json

Body中放置请求参数，参数详情如下：

请求参数

参数	必选	类型	说明
user_id	是	string	用户id（由数字、字母、下划线组成），长度限制48B
group_id	是	string	用户组id (由数字、字母、下划线组成），长度限制48B

示例代码

提示一：使用示例代码前，请记得替换其中的示例Token、图片地址或Base64信息。

提示二：部分语言依赖的类或库，请在代码注释中查看下载地址。

Bash

PHP

JAVA

Python

Cpp

C#

获取用户人脸列表

```
curl -i -k 'https://aip.baidubce.com/rest/2.0/face/v3/faceset/face/getlist?access_token=【调用鉴权接口获取的token】' --data '{"user_id":"user1","group_id":"group1"}' -H 'Content-Type:application/json; charset=UTF-8'
```

返回说明

返回参数

字段	必选	类型	说明
log_id	是	uint64	请求标识码，随机数，唯一
face_list	是	array	人脸列表
+ face_token	是	string	人脸图片的唯一标识（有效期永久）
+ctime	是	string	人脸创建时间

返回示例

```
{
 "face_list": [
    {
        "face_token": "fid1",
        "ctime": "2018-01-01 00:00:00"
    },
    {
        "face_token": "fid2",
        "ctime": "2018-01-01 10:00:00"
    }
 ]
}
```

获取用户列表

接口描述

用于查询指定用户组中的用户列表。

在线调试 您可以在示例代码中心中调试该接口，可进行签名验证、查看在线调用的请求内容和返回结果、示例代码的自动生成。

请求说明

请求示例

HTTP方法：`POST`

请求URL： `https://aip.baidubce.com/rest/2.0/face/v3/faceset/group/getusers`

URL参数：

参数	值
access_token	通过API Key和Secret Key获取的access_token,参考“Access Token获取”

Header如下：

参数	值
Content-Type	application/json

Body中放置请求参数，参数详情如下：

请求参数

参数	必选	类型	说明
group_id	是	string	用户组id，长度限制48B
start	否	uint32	默认值0，起始序号
length	否	uint32	返回数量，默认值100，最大值1000

示例代码

提示一：使用示例代码前，请记得替换其中的示例Token、图片地址或Base64信息。

提示二：部分语言依赖的类或库，请在代码注释中查看下载地址。

Bash

PHP

JAVA

Python

Cpp

C#

获取用户列表

```
curl -i -k 'https://aip.baidubce.com/rest/2.0/face/v3/faceset/group/getusers?access_token=【调用鉴权接口获取的token】' --data '{"group_id":"group1"}' -H 'Content-Type:application/json; charset=UTF-8'
```

返回说明

- 返回结果

字段	必选	类型	说明
user_id_list	是	array	用户ID列表

- 返回示例

```
{
  "user_id_list": [
    "uid1",
    "uid2"
  ]
}
```

复制用户

接口描述

用于将已经存在于人脸库中的用户**复制到一个新的组**。

> **说明**：并不是向一个指定组内添加用户，而是直接从其它组复制用户信息 如果需要注册用户，请直接使用**人脸注册接口**
>
> **在线调试 您可以在示例代码中心中调试该接口**，可进行签名验证、查看在线调用的请求内容和返回结果、示例代码的自动生成。

请求说明

请求示例

HTTP方法：`POST`

请求URL： `https://aip.baidubce.com/rest/2.0/face/v3/faceset/user/copy`

URL参数：

参数	值
access_token	通过API Key和Secret Key获取的access_token,参考“Access Token获取

Header如下：

参数	值
Content-Type	application/json

Body中放置请求参数，参数详情如下：

请求参数

参数	必选	类型	说明
user_id	是	string	用户id，长度限制48B
src_group_id	是	string	从指定组里复制信息
dst_group_id	是	string	需要添加用户的组id

返回说明

通过返回的error_code判断是否成功 如失败则查看error_msg获得具体错误信息

返回示例

```
 // 正确返回值
{
  "error_code": 0,
 "log_id": 3314921889,
}
// 发生错误时返回值
{
 "error_code": 223111,
 "log_id": 3111284097,
 "error_msg": "dst group is not exist"
}
```

删除用户

接口描述

用于将用户从某个组中删除。

在线调试 您可以在示例代码中心中调试该接口，可进行签名验证、查看在线调用的请求内容和返回结果、示例代码的自动生成。

请求说明

请求示例

HTTP方法：`POST`

请求URL： `https://aip.baidubce.com/rest/2.0/face/v3/faceset/user/delete`

URL参数：

参数	值
access_token	通过API Key和Secret Key获取的access_token,参考“Access Token获取”

Header如下：

参数	值
Content-Type	application/json

Body中放置请求参数，参数详情如下：

请求参数

参数	必选	类型	说明
group_id	是	string	用户组id(由数字、字母、下划线组成，长度限制48B），如传入“**@ALL**”则从所有组中删除用户
user_id	是	string	用户id（由数字、字母、下划线组成），长度限制48B

返回说明

通过返回的error_code判断是否成功 如失败则查看error_msg获得具体错误信息：

返回参数

字段	必选	类型	说明
log_id	是	uint64	请求标识码，随机数，唯一

返回示例

```
 // 正确返回值
{

 "error_code": 0,
 "log_id": 3314921889,
}
// 发生错误时返回值
{
 "error_code": 223103,
 "log_id": 815967402,
 "error_msg": "user is not exist"
}
```

创建用户组

接口描述

用于创建一个空的用户组，如果用户组已存在 则返回错误。

在线调试 您可以在示例代码中心中调试该接口，可进行签名验证、查看在线调用的请求内容和返回结果、示例代码的自动生成。

请求说明

请求示例

HTTP方法：`POST`

请求URL： `https://aip.baidubce.com/rest/2.0/face/v3/faceset/group/add`

URL参数：

参数	值
access_token	通过API Key和Secret Key获取的access_token,参考“Access Token获取”

Header如下：

参数	值
Content-Type	application/json

Body中放置请求参数，参数详情如下：

请求参数

参数	必选	类型	说明
group_id	是	string	用户组id，标识一组用户（由数字、字母、下划线组成），长度限制48B。

示例代码

提示一：使用示例代码前，请记得替换其中的示例Token、图片地址或Base64信息。

提示二：部分语言依赖的类或库，请在代码注释中查看下载地址。

Bash

PHP

JAVA

Python

Cpp

C#

创建用户组

```
curl -i -k 'https://aip.baidubce.com/rest/2.0/face/v3/faceset/group/add?access_token=【调用鉴权接口获取的token】' --data '{"group_id":"group1"}' -H 'Content-Type:application/json; charset=UTF-8'
```

返回说明

通过返回的error_code判断是否成功 如失败则查看error_msg获得具体错误信息：

返回参数

字段	必选	类型	说明
log_id	是	uint64	请求标识码，随机数，唯一

返回示例

```
 // 正确返回值
{

  "error_code": 0,
  "log_id": 3314921889,
}
// 发生错误时返回值
{
  "error_code":  223101,
  "log_id": 815967402,
  "error_msg": " group is already exist"
}
```

删除用户组

接口描述

删除用户组下所有的用户及人脸，如果组不存在 则返回错误。 注：组内的人脸数量如果大于500条，会在后台异步进行删除。

在删除期间，无法向该组中添加人脸。1秒钟可以删除20条记录，相当于一小时可以将7万人的人脸组删除干净。用户组删除可能存在一定延迟，期间指定该用户组搜索，仍可搜索到未删除完成的用户信息。如果您的用户组下数据量较大并且希望保证删除的时效性，建议您先循环删除组内的用户，最后删除用户组。

在线调试 您可以在示例代码中心中调试该接口，可进行签名验证、查看在线调用的请求内容和返回结果、示例代码的自动生成。

请求说明

请求示例

HTTP方法：`POST`

请求URL： `https://aip.baidubce.com/rest/2.0/face/v3/faceset/group/delete`

URL参数：

参数	值
access_token	通过API Key和Secret Key获取的access_token,参考“Access Token获取”

Header如下：

参数	值
Content-Type	application/json

Body中放置请求参数，参数详情如下：

请求参数

参数	必选	类型	说明
group_id	是	string	用户组id，标识一组用户（由数字、字母、下划线组成），长度限48B。

示例代码

提示一：使用示例代码前，请记得替换其中的示例Token、图片地址或Base64信息。

提示二：部分语言依赖的类或库，请在代码注释中查看下载地址。

Bash

PHP

JAVA

Python

Cpp

C#

```
 删除用户组
curl -i -k 'https://aip.baidubce.com/rest/2.0/face/v3/faceset/group/delete?access_token=【调用鉴权接口获取的token】' --data '{"group_id":"group1"}' -H 'Content-Type:application/json; charset=UTF-8'
```

返回说明

通过返回的error_code判断是否成功 如失败则查看error_msg获得具体错误信息：

返回参数

字段	必选	类型	说明
log_id	是	uint64	请求标识码，随机数，唯一

返回示例

```
 // 正确返回值
{

 "error_code":0,
 "log_id": 3314921889,
}
// 发生错误时返回值
{
 "error_code": 223100,
 "log_id": 815967402,
 "error_msg": " group is not exist"
}
```

组列表查询

接口描述

用于查询用户组的列表。

在线调试 您可以在示例代码中心中调试该接口，可进行签名验证、查看在线调用的请求内容和返回结果、示例代码的自动生成。

请求说明

请求示例

HTTP方法：`POST`

请求URL： `https://aip.baidubce.com/rest/2.0/face/v3/faceset/group/getlist`

URL参数：

参数	值
access_token	通过API Key和Secret Key获取的access_token,参考“Access Token获取”

Header如下：

参数	值
Content-Type	application/json

Body中放置请求参数，参数详情如下：

请求参数

参数	必选	类型	说明
start	否	uint32	默认值0，起始序号
length	否	uint32	返回数量，默认值100，最大值1000

示例代码

提示一：使用示例代码前，请记得替换其中的示例Token、图片地址或Base64信息。

提示二：部分语言依赖的类或库，请在代码注释中查看下载地址。

Bash

PHP

JAVA

Python

Cpp

C#

```
 组列表查询
curl -i -k 'https://aip.baidubce.com/rest/2.0/face/v3/faceset/group/getlist?access_token=【调用鉴权接口获取的token】' --data '{"start":0,"length":100}' -H 'Content-Type:application/json; charset=UTF-8'
```

返回说明

返回参数

- 返回结果

字段	必选	类型	说明
group_id_list	是	array	group

- 返回示例

```
{
  "group_id_list": [
    "gid1",
    "gid2"
  ]
}
```

在线图片活体V3

能力介绍

接口能力

- **人脸基础信息**：包括人脸框位置，人脸空间旋转角度，人脸置信度等信息。
- **人脸质量检测**：判断人脸的遮挡、光照、模糊度、完整度等质量信息。可用于判断上传的人脸是否符合标准。
- **基于图片的活体检测**：基于单张图片，判断图片中的人脸是否为二次翻拍（举例：如用户A用手机拍摄了一张包含人脸的图片一，用户B翻拍了图片一得到了图片二，并用图片二伪造成用户A去进行识别操作，这种情况普遍发生在金融开户、实名认证等环节）以及是否为合成图攻击。此能力可用于H5/App场景下的一些人脸采集场景中，增加人脸注册的安全性和真实性。

在线调试

您可以在 示例代码中心 中调试该接口，可进行签名验证、查看在线调用的请求内容和返回结果、示例代码的自动生成。

调用方式

请求URL数据格式

向API服务地址使用POST发送请求，必须在URL中带上参数`access_token`，可通过后台的API Key和Secret Key生成，具体方式请参考“Access Token获取”。

示例代码

Bash

PHP

JAVA

Python

Cpp

C#

Node

```
##### !/bin/bash
curl -i -k 'https://aip.baidubce.com/oauth/2.0/token?grant_type=client_credentials&client_id=【百度云应用的AK】&client_secret=【百度云应用的SK】'
```

> **注意**：access_token的有效期为30天，**切记需要每30天进行定期更换，或者每次请求都拉取新token**；

例如此接口，使用HTTPS POST发送：

```
https://aip.baidubce.com/rest/2.0/face/v3/faceverify?access_token=24.f9ba9c5341b67688ab4added8bc91dec.2592000.1485570332.282335-8574074
```

POST中Body的参数，按照下方请求参数说明选择即可。

> **提示**：如果您为百度云老用户，正在使用其他非AI的服务，可以参考百度云AKSK鉴权方式发送请求，虽然**请求方式**和**鉴权方法**和本文所介绍的不同，但请求参数和返回结果一致。

请求说明

注意事项：

- **请求体格式化**：Content-Type为application/json，通过json格式化请求体。
- **Base64编码**：请求的图片需经过Base64编码，图片的base64编码指将图片数据编码成一串字符串，使用该字符串代替图像地址。您可以首先得到图片的二进制，然后用Base64格式编码即可。需要注意的是，图片的base64编码是不包含图片头的，如
- **图片格式**：现支持PNG、JPG、JPEG、BMP，**不支持GIF图片**

请求示例

HTTP方法：POST

请求URL： https://aip.baidubce.com/rest/2.0/face/v3/faceverify

URL参数：

参数	值
access_token	通过API Key和Secret Key获取的access_token,参考“Access Token获取”

Header：

参数	值
Content-Type	application/json

Body中放置请求参数，参数详情如下：

请求参数

参数	是否必选	类型	说明
image	是	string	图片信息(总数据大小应小于10M，分辨率应小于1920*1080)，图片上传方式根据image_type来判断； **可以上传同一个用户的1张、3张或8张图片来进行活体判断** 注： (1)后端会选择每组照片中的最高分数作为整体分数。 图片通过json格式上传，格式参考表格下方示例 (2)支持1、3、8张图片输入进行计算，请求格式为数组格式
image_type	是	string	图片类型 **BASE64**:（**推荐**）图片的base64值，base64编码后的图片数据，编码后的图片大小不超过2M； **FACE_TOKEN**: 人脸图片的唯一标识，调用人脸检测接口时，会为每个人脸图片赋予一个唯一的FACE_TOKEN，同一张图片多次检测得到的FACE_TOKEN是同一个。
face_field	否	string	包括**age,beauty,expression,face_shape,gender,glasses,landmark,quality,face_type,spoofing信息**，逗号分隔，默认只返回face_token、活体数、人脸框、概率和旋转角度
option	否	string	场景信息，程序会视不同的场景选用相对应的模型。当前支持的场景有COMMON(通用场景)，GATE(闸机场景)，**默认使用COMMON**

说明：图片的上传使用json格式，发送内容为数组，（即 [{XXXXX}，{XXXXX}] 这种格式）具体如下：

```
[
  {
     "image": "sfasq35sadvsvqwr5q...",
     "image_type": "BASE64",
      "face_field": "age,beauty,spoofing",
      "option": "COMMON"
  },
  {
     "image": "http://xxx.baidu.com/image1.png",
     "image_type": "URL",
     "face_field": "age,beauty,spoofing",
     "option": "COMMON"
  },
  {
    "image": "9f30d19f86f89f2f07ce88b69557061a",
    "image_type": "FACE_TOKEN",
    "face_field": "age,beauty,spoofing",
    "option": "COMMON"
  }
]
```

示例代码

提示一：使用示例代码前，请记得替换其中的示例Token、图片地址或Base64信息。

提示二：部分语言依赖的类或库，请在代码注释中查看下载地址。

Bash

PHP

JAVA

Python

Cpp

C#

在线图片活体v3

```
curl -i -k 'https://aip.baidubce.com/rest/2.0/face/v3/faceverify?access_token=【调用鉴权接口获取的token】' --data '[{"image":"sfasq35sadvsvqwr5q...","image_type":"BASE64","face_field":"age,beauty,expression"},{"image":"http://xxx.baidu.com/image1.png","image_type":"URL","face_field":"age,beauty"}]' -H 'Content-Type:application/json; charset=UTF-8'
```

返回参数

参数	是否必须	类型	说明
face_liveness	是	float	所有图片的总体活体最高得分，范围【0~1】
thresholds	是	array	由服务端返回最新的阈值数据（随着模型的优化，阈值可能会变化），将此参数与返回的face_liveness进行比较，可以作为活体判断的依据。 frr_1e-4：万分之一误识率的阈值；frr_1e-3：千分之一误识率的阈值；frr_1e-2：百分之一误识率的阈值。误识率越低，准确率越高，相应的拒绝率也越高
face_list	是	array	每张图片的详细信息描述，如果只上传一张图片，则只返回一个结果。
+face_token	是	string	人脸图片的唯一标识
+location	是	array	人脸在图片中的位置
++left	是	double	人脸区域离左边界的距离
++top	是	double	人脸区域离上边界的距离
++width	是	double	人脸区域的宽度
++height	是	double	人脸区域的高度
++rotati	是	int64	人脸框相对于竖直方向的顺时针旋转角，[-180,180]

on			
+face_probability	是	double	人脸置信度，范围【0~1】，代表这是一张人脸的概率，0最小、1最大。
+angle	是	array	人脸旋转角度参数
++yaw	是	double	三维旋转之左右旋转角[-90(左), 90(右)]
++pitch	是	double	三维旋转之俯仰角度[-90(上), 90(下)]
++roll	是	double	平面内旋转角[-180(逆时针), 180(顺时针)]
+age	否	double	年龄 ，当**face_field包含age时返回**
+expression	否	array	表情，当 **face_field包含expression时返回**
++type	否	string	**none**:不笑；**smile**:微笑；**laugh**:大笑
++probability	否	double	表情置信度，范围【0~1】，0最小、1最大。
+face_shape	否	array	脸型，当**face_field包含face_shape时返回**
++type	否	double	**square**: 正方形 **triangle**:三角形 **oval**: 椭圆 **heart**: 心形 **round**: 圆形
++probability	否	double	置信度，范围【0~1】，代表这是人脸形状判断正确的概率，0最小、1最大。
+gender	否	array	性别，**face_field包含gender时返回**
++type	否	string	male:**男性** female:**女性**
++probability	否	double	性别置信度，范围【0~1】，0代表概率最小、1代表最大。
+glasses	否	array	是否带眼镜，**face_field包含glasses时返回**
++type	否	string	**none**:无眼镜，**common**:普通眼镜，**sun**:墨镜
++probability	否	double	眼镜置信度，范围【0~1】，0代表概率最小、1代表最大。
+face_type	否	array	真实人脸/卡通人脸 **face_field包含face_type时返回**
++type	否	string	**human**: 真实人脸 **cartoon**: 卡通人脸
++probability	否	double	人脸类型判断正确的置信度，范围【0~1】，0代表概率最小、1代表最大。
+landmark	否	array	4个关键点位置，左眼中心、右眼中心、鼻尖、嘴中心。**face_field包含landmark时返回**
+landmark72	否	array	72个特征点位置 **face_field包含landmark时返回**
+quality	否	array	人脸质量信息。**face_field包含quality时返回**
++occlusion	否	array	人脸各部分遮挡的概率，范围[0~1]，0表示完整，1表示不完整
+++left_eye	否	double	左眼遮挡比例，[0-1] ， 1表示完全遮挡

+++right_eye	否	double	右眼遮挡比例，[0-1] ， 1表示完全遮挡
+++nose	否	double	鼻子遮挡比例，[0-1] ， 1表示完全遮挡
+++mouth	否	double	嘴巴遮挡比例，[0-1] ， 1表示完全遮挡
+++left_cheek	否	double	左脸颊遮挡比例，[0-1] ， 1表示完全遮挡
+++right_cheek	否	double	右脸颊遮挡比例，[0-1] ， 1表示完全遮挡
+++chin	否	double	下巴遮挡比例，[0-1] ， 1表示完全遮挡
++blur	否	double	人脸模糊程度，范围[0~1]，0表示清晰，1表示模糊
++illumination	否	double	取值范围在[0~255], 表示脸部区域的光照程度 越大表示光照越好
++completeness	否	int64	人脸完整度，0或1, 0为人脸溢出图像边界，1为人脸都在图像边界内
liveness	否	double	单张图片活体检测结果
+livemapscore	否	double	单张图片的活体得分，范围[0~1]
+spoofing	否	double	合成图打分 判断图片是否为合成图 face_field包含时返回spoofing

返回示例

```
{
  "error_code": 0,
  "error_msg": "SUCCESS",
  "log_id": 9999201750012,
  "timestamp": 1587021157,
  "cached": 0,
  "result": {
    "thresholds": {
      "frr_1e-4": 0.05,
      "frr_1e-3": 0.3,
      "frr_1e-2": 0.9
    },
    "face_liveness": 1,
    "face_list": [
      {
        "face_token": "6e6880584e9ad6d22e309da37ed72b78",
        "location": {
          "left": 60.21,
          "top": 134.93,
          "width": 162,
          "height": 164,
          "rotation": -4
        },
        "face_probability": 1,
        "angle": {
          "yaw": -0.9,
          "pitch": 12.87,
          "roll": -5.36
        },
        "liveness": {
          "livemapscore": 1
        },
        "spoofing": 0.0002664364583
      }
    ]
  }
}
```


活体阈值参考

请务必在产品侧做好以下条件限制

- 检测的图片为二次采集，即通过相机当场拍摄，确保时间及操作条件的约束；
- SDK输出的多帧情况，只要这些帧中，**任何一张通过了阈值，即可判断为活体，建议可用三帧情况**

关于活体检测faceliveness的判断阈值选择，可参考以下数值信息：

阈值（Threshold）	误拒率（FRR）	通过率（TAR）	攻击拒绝率（TRR））
0.05	0.01%	99.99%	97.75%
0.1	0.05%	99.95%	98.33%
0.3 （**推荐**）	0.1%	99.9%	98.82%
0.5	0.5%	99.5%	99.67%
0.9	1%	99%	99.77%

关于以上数值的概念介绍：

- **拒绝率（TRR）**：如99%，代表100次作弊假体攻击，会有99次被拒绝。

- **误拒率（FRR）**：如0.5%，指1000次真人请求，会有5次因为活体分数低于阈值被错误拒绝。
- **通过率（TAR）**：如99%，指100次真人请求，会有99次因为活体分数高于阈值而通过。
- **阈值（Threshold）**：高于此数值，则可判断为活体。

合成图阈值参考

> 新推出合成图检测能力，在face_field字段中增加spoofing参数，进行判断，若spoofing分值高于合成图推荐阈值，则可判断为合成图攻击；此参数在face_liveness基础上进行合成图判断。

关于合成图检测spoofing的判断阈值选择，可参考以下数值信息：

阈值	误拒率（FRR）	通过率（TAR）	攻击拒绝率（TRR））
0.00023	5%	95%	94.93%
0.00048（**推荐**）	1%	99%	89.71%
0.00066	0.5%	99.5%	88.02%
0.00109	0.1%	99.9%	84.57%
0.00171	0.05%	99.95%	81.52%
0.00547	0.01%	99.99%	65.52%

- **阈值（Threshold）**：高于此数值，则可判断为是合成图攻击。

质量检测参考

指标	字段与解释	推荐数值界限
遮挡范围	**occlusion**，取值范围[0~1]，0为无遮挡，1是完全遮挡 含有多个具体子字段，表示脸部多个部位 通常用作判断头发、墨镜、口罩等遮挡	left_eye：0.6, #左眼被遮挡的阈值 right_eye：0.6, #右眼被遮挡的阈值 nose：0.7, #鼻子被遮挡的阈值 mouth：0.7, #嘴巴被遮挡的阈值 left_check：0.8, #左脸颊被遮挡的阈值 right_check：0.8, #右脸颊被遮挡的阈值 chin_contour：0.6, #下巴被遮挡阈值
模糊度范围	**blur**，取值范围[0~1]，0是最清晰，1是最模糊	小于0.7
光照范围	**illumination**，取值范围[0~255] 脸部光照的灰度值，0表示光照不好 以及对应客户端SDK中，YUV的Y分量	大于40
姿态角度	**Pitch**：三维旋转之俯仰角度[-90(上), 90(下)] **Roll**：平面内旋转角[-180(逆时针), 180(顺时针)] **Yaw**：三维旋转之左右旋转角[-90(左), 90(右)]	分别小于20度
人脸完整度	**completeness**（0或1），0为人脸溢出图像边界，1为人脸都在图像边界内	视业务逻辑判断
人脸大小	人脸部分的大小 建议长宽像素值范围：80*80~200*200	人脸部分不小于**100*100**像素

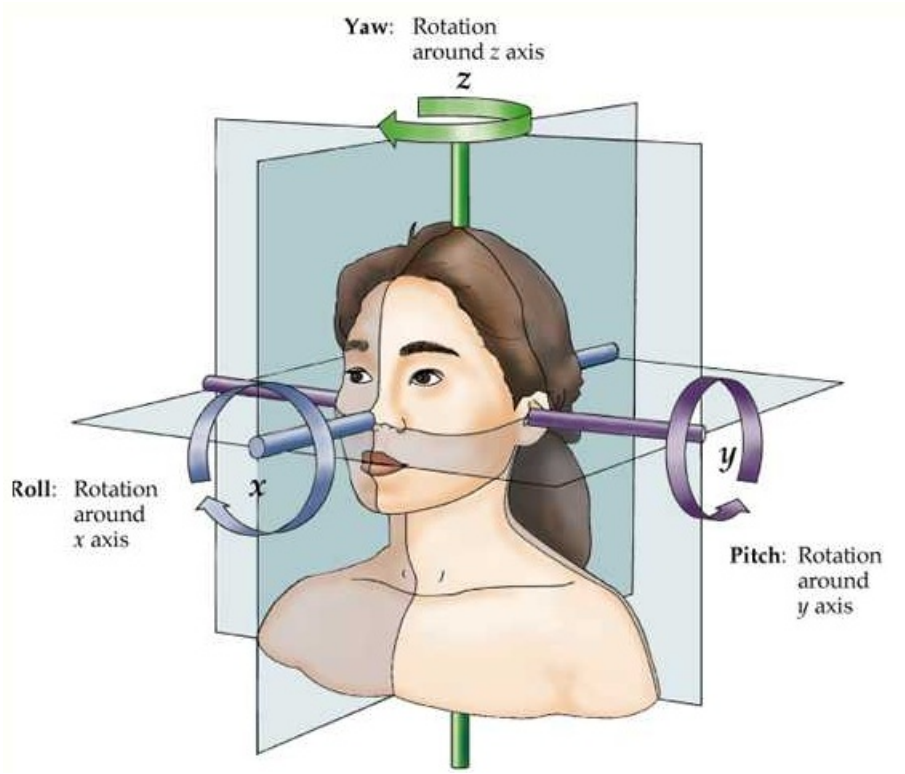
人脸空间姿态角参考

姿态角分为`Pitch`、`Roll`、`Yaw`，用于表示人脸在空间三维坐标系内的角度，常用于判断识别角度的界限值。

各角度阈值如下：

> Pitch：三维旋转之俯仰角度，范围：[-90（上），90（下）]，推荐俯仰角绝对值不大于20度；
> Roll：平面内旋转角，范围：[-180（逆时针），180（顺时针）]，推荐旋转角绝对值不大于20度；
> Yaw：三维旋转之左右旋转角，范围：[-90（左），90（右）]，推荐旋转角绝对值不大于20度；

各角度范围示意图如下：



从姿态角度来看，**这三个值的绝对值越小越好**，这样代表人脸足够正视前方，最利于实际注册/识别使用。

错误码

请参考人脸识别错误码

视频活体检测

如果您对文档内容有任何疑问，可以通过以下几种方式联系我们：

- 在百度云控制台内 提交工单，咨询问题类型请选择**人工智能服务**；
- 如有需要讨论的疑问，欢迎进入 AI社区 与其他开发者们一同交流。

一、视频活体检测

能力介绍

业务能力

视频活体检测产品，是由两个接口（视频活体检测+随机校验码）组合而成，可实现**动作视频活体**、**数字视频活体**两种活体检测方式。主要应用在H5场景下，通过用户新录制并上传一个视频，来进行活体检测的判断，同时比单张图片活体检测方式更加安全。其主要功能如下所示：

- **质量检测（可选）**：判断图片中是否包含人脸，以及人脸在姿态、遮挡、模糊、光照等方面是否符合识别条件。
- **视频多帧活体检测**：录制并上传的视频，会在云端进行随机抽帧分析，并得出最终的活体检测分数。
- **随机校验码**：（用于在语音/动作活体检测中生成随机数字/动作）

- 为防止用户提交非当前操作的视频，选择随机验证码后，即可在录制视频时，随机生成数字/动作，用户需要读出数字/做出相应动作，在后续识别时校验，以判断视频是否为现场录制。
- 随机校验码作为辅助性质的验证条件，是一个可选项，可根据业务具体应用场景来选择是否使用，以及**根据业务场景决定选择语音/动作活体检测方式**。如业务场景比较嘈杂或方言口音比较重，可不使用语音活体检测方式，选择动作活体检测方式进行校验。

- **唇语识别Beta版**：
  - 对用户上传的视频进行唇语识别，返回唇语识别是否通过，返回结果为单独字段，与视频活体与语音/动作校验不冲突。
  - 若需要使用唇语识别，需要先使用随机校验码和视频活体检测接口，且活体检测方式应配置为语音活体检测。
  - **唇语识别能力当前为Beta版本，识别准确率较低，仅用于辅助 语音活体检测方式 进行验证，您可以通过接口的入参来设置是否使用该能力**
- **合成图识别Beta版**：
  - 对用户上传的视频抽帧进行合成图像识别，能识别出AI变脸、AI换脸等合成图，让业务更加安全。
  - **合成图识别能力当前为Beta版本，仅用于辅助验证，您可以通过接口的入参来设置是否使用该能力**

> 以上四项能力，分为两个接口，使用顺序为`随机校验码接口`->`视频活体检测接口`，具体调用逻辑可以参考文档 接口文档

主要适用场景

- 微信服务号：用于对操作用户真实性要求严格的场景，用于依托于微信服务号的金融开户、实名认证、账户信息变更二次验证等服务。
- APP内Webview：对于如Cordova架构开发的APP，或者APP内变更频繁的身份信息页等情况，可以采用此方案完成活体检测。
- 浏览器：如果仅是一个H5宣传页，或者Wap版本网页等，可以通过此方法快速集成更加安全的活体检测功能。

此方案的优劣势

- 优势：相对于APP有动作校验、单张图片静默判断，此方法在没有APP情况下，可以更快速、轻量级地实现活体检测，同时保障一定安全性。
- 劣势：由于视频较大，上传时间可能较长，另由于不同手机的浏览器内核差异较大，容易出现兼容性问题。

调用方式

请求URL数据格式

向API服务地址使用POST发送请求，必须在URL中带上参数`access_token`，可通过后台的API Key和Secret Key生成，具体方式请参考“Access Token获取”。

示例代码

Bash
PHP
JAVA
Python
Cpp

C#

Node

```
##### !/bin/bash
curl -i -k 'https://aip.baidubce.com/oauth/2.0/token?grant_type=client_credentials&client_id=【百度云应用的AK】&client_secret=【百度云应用的SK】'
```

> **注意**：access_token的有效期为30天，**切记需要每30天进行定期更换，或者每次请求都拉取新token**；

例如此接口，使用HTTPS POST发送：

```
https://aip.baidubce.com/rest/2.0/face/v1/merge?access_token=24.f9ba9c5341b67688ab4added8bc91dec.2592000.1485570332.282335-8574074
```

POST中Body的参数，按照下方请求参数说明选择即可。

> **提示**：如果您为百度云老用户，正在使用其他非AI的服务，可以参考百度云AKSK鉴权方式发送请求，虽然**请求方式**和**鉴权方法**和本文所介绍的不同，但请求参数和返回结果一致。

1.1 随机校验码接口（原语音验证码接口）

接口描述

此接口主要用于生成随机码，用于视频的语音/动作识别校验使用，以判断视频的即时性，而非事先录制的，提升作弊的难度。

在线调试 您可以在 示例代码中心 中调试该接口，可进行签名验证、查看在线调用的请求内容和返回结果、示例代码的自动生成。

请求说明

请求示例

HTTP方法：POST

请求URL： https://aip.baidubce.com/rest/2.0/face/v1/faceliveness/sessioncode

URL参数：

参数	值
access_token	通过API Key和Secret Key获取的access_token,参考“Access Token获取”

Header：

参数	值
Content-Type	application/x-www-form-urlencoded

Body中放置请求参数，参数详情如下：

- 请求参数

参数名	必选	类型	说明
type	否	int	0：下发语音验证码和唇语验证码，**默认类型** 1：下发视频动作活体验证码
min_code_length	否	int	当type=0时，生成语音和唇语验证码的最小长度：最小3，最大6，**默认3** 当type=1时，视生成视频动作活体验证码的最小长度，最小1，最大3，**默认3**
max_code_length	否	int	当type=0时，生成语音和唇语验证码的最大长度，最小3，最大6，**默认3** 当type=1时，生成视频动作活体验证码的最大长度：最小1，最大3，**默认3**

> **说明**：
>
> - 当传参为1-1-3时，代表随机生成1-3个动作进行核验；
> - 当传参为1-1-1时，代表随机生成1个动作进行核验；
> - 当传参为1-2-2时，代表随机生成2个动作进行核验；
> - 当传参为1-3-3时，代表随机生成3个动作进行核验。

返回说明

返回参数

字段	必选	类型	说明
session_id	是	string	随机校验码会话id，有效期5分钟，请提示用户在五分钟内完成全部操作 验证码使用过即失效，每次使用视频活体前请重新拉取验证码
code	是	string	随机验证码，数字形式，1~6位数字； 若为动作活体时，返回数字表示的动作对应关系为：0:眨眼 4:抬头 5:低头 7:左右转头（不区分先后顺序，分别向左和向右转头），注：『7：左右转头』为2022年4月底上线的新动作，此后接入的新客户默认开放该动作，老客户默认不触发该动作，可提交工单要求配置触发

返回示例

```
{
  "err_no": 0,
  "err_msg": "SUCCESS",
  "result": {
     "session_id": "S59faeeebb9111890355690", //会话ID
     "code": "045" //当为视频动作活体时，返回值的代表所需动作和动作顺序。 0:眨眼 4:抬头  5:低头
  },
  "timestamp": 1509617387,
  "cached": 0,
  "serverlogid": "0587756642",
  "error_code": 0,
   "error_message": "SUCCESS"
}
```

> error_code为0即代表成功, 其他情况则为失败

1.2 视频活体检测接口

接口描述

此接口一方面通过随机验证码接口获取语音/动作校验码，通过session code来判断视频是否作弊。另一方面进行视频抽帧，判断是否为活体。

在线调试 您可以在 示例代码中心 中调试该接口，可进行签名验证、查看在线调用的请求内容和返回结果、示例代码的自动生成。

请求说明

请求示例

HTTP方法：`POST`

请求URL： `https://aip.baidubce.com/rest/2.0/face/v1/faceliveness/verify`

URL参数：

参数	值
access_token	通过API Key和Secret Key获取的access_token,参考“Access Token获取”

Header：

参数	值
Content-Type	application/x-www-form-urlencoded

Body中放置请求参数，参数详情如下：

请求参数

参数名	必选	类型	说明
type_identify	否	string	voice为语音验证，action为视频动作活体验证，默认为voice （若您需要静默视频活体验证，此参数无需传入）
session_id	否	string	会话ID （当此字段为空时，为静默视频活体检测） 当使用语音验证及视频动作活体验证时，此字段必须传入，且获取验证码时要填入对应的验证类型。session_id获取方式参考随机校验码文档。 **注**：session_id与type_identify参数的核验方式需保持一致，如：当type_identify参数传入action时，session_id也需为动作校验码
video_base64	是	string	base64 编码的视频数据（编码前建议先将视频进行转码，h.264编码，mp4封装）需要注意的是，视频的base64编码是不包含视频头的，如 data:video/mp4;base64,； 建议视频长度控制在01s-10s之间，视频大小建议在2M以内（视频大小强制要求在20M以内，推荐使用等分辨率压缩，压缩分辨率建议不小于640*480） 视频大小分辨率建议限制在16~2032之间
lip_identify	否	string	辅助语音验证进行，用于判断验证码是否为当事人读出 取值COMMON/STRICT/OFF, COMMON代表使用唇语识别，STRICT代表使用唇语识别并使用更加严格的策略判断是否通过 OFF代表关闭唇语识别，默认OFF
face_field	否	string	需要使用合成图功能时, 此项传入**spoofing** 需要使用图片质量信息时，则传入**quality** 字段之间使用**,**号分隔，eg：**spoofing,quality**

> 唇语识别中，使用STRICT策略会比使用COMMON策略通过率降低，但攻击拒绝率会提升 建议视频大小控制在10M/1min以内

请求示例

```
{
  "session_id": "S62b193a44a46c363371046",
  "video_base64": "video_base64_value",
  "type_identify": "voice",
  "lip_identify": "STRICT",
  "face_field": "spoofing,quality"
}
```

返回说明

返回参数

参数名	类型	说明
score	float	活体检测的总体打分 范围[0,1]，分数越高则活体的概率越大
maxspoofing	float	返回的1-8张图片中合成图检测得分的最大值 范围[0,1]，分数越高则概率越大
spoofing_score	float	返回的1-8张图片中合成图检测得分的中位数 范围[0,1]，分数越高则概率越大
thresholds	array	阈值 按活体检测分数>阈值来判定活体检测是否通过(阈值视产品需求选择其中一个)
code	array	验证码信息
+create	string	生成的验证码
+identify	string	验证码的语音识别结果
+similarity	float	验证码相似度 取值0~1 1代表完全一致 0代表完全不一致 推荐阈值**0.75**
lip_language	string	唇语识别结果 pass代表唇语验证通过，fail代表唇语验证未通过，当存在请求字段lip_identify字段值为COMMON 或 STRICT时返回
action_verify	string	动作识别结果 pass代表动作验证通过，fail代表动作验证未通过，当存在请求字段type_identify字段值为action时返回
best_image	array	质量最佳图片
+face_token	string	人脸图片的唯一标识
+pic	string	base64编码后的图片信息
+liveness_score	float	此图片的活体分数，范围[0,1]
pic_list	array	返回1-8张抽取出来的图片信息
+face_token	string	人脸图片的唯一标识
+spoofing	float	此图片的合成图分数，范围[0,1]

返回示例

```
 {
  "err_no": 0,
  "err_msg": "SUCCESS",
  "result": {
      "score": 0.18,
      "maxspoofing": 0.0002082588035,
  "spoofing_score": 0.00018671568975,
      "code": {
          "create": "853",
          "identify": "23456789",
          "similarity": 0.13
      },
      "lip_language": "fail",     //lip_identify 为 COMMON 或 STRICT 时返回该字段值，值为pass 或 false
      "action_verify": "fail",    //type_identify 为 action 时返回该字段值，值为pass 或 false【注意：action_verify与
lip_language是互斥的】
      "thresholds": {
            "frr_1e-4": 0.05,     //万分之一误拒率的阈值
            "frr_1e-3": 0.3,      //千分之一误拒率的阈值
            "frr_1e-2": 0.9,      //百分之一误拒率的阈值
      },
   "best_image": {
          "pic": "图片base64值",
          "face_token": "0839b921224816fb558b0a74ee6284fb",
          "face_id": "0839b921224816fb558b0a74ee6284fb",
          "liveness_score": 0.9634260269,
          "spoofing": 0.0001962436945,
```

```
        "quality": {
            "occlusion": {
                "left_eye": 0,
                "right_eye": 0,
                "nose": 0,
                "mouth": 0,
                "left_cheek": 0,
                "right_cheek": 0,
                "chin_contour": 0
            },
            "blur": 0,
            "illumination": 114,
            "completeness": 1
        },
        "angle": {
            "yaw": 1.59,
            "pitch": 0.26,
            "roll": -5.46
        }
    },
    "pic_list": [ //默认返回8张图片
      {
        "pic" : "gAQTGF2YzU4LjkyLjEwMAD", //图片base64编码后的值
        "face_token": "f043b6c7d202cb25e8dfc24fccf37553",
        "face_id": "f043b6c7d202cb25e8dfc24fccf37553",
        "liveness_score": 0.18,
        "spoofing": 0.000179775554,
        "quality": {
          "occlusion": {
              "left_eye": 0,
              "right_eye": 0,
              "nose": 0,
              "mouth": 0,
              "left_cheek": 0,
              "right_cheek": 0,
              "chin_contour": 0
          },
          "blur": 0,
          "illumination": 114,
          "completeness": 1
        },
        "angle": {
          "yaw": 1.59,
          "pitch": 0.26,
          "roll": -5.46
        }
      },
      {
        "pic" : "gAQTGF2YzU4LjkyLjEwMAD", //图片base64编码后的值
        "face_token": "d12b7e32069d337dc5d57cd3d15e2935",
        "face_id": "d12b7e32069d337dc5d57cd3d15e2935",
        "liveness_score": 0.06,
        "spoofing": 0.0002082588035,
        "quality": {
          "occlusion": {
              "left_eye": 0,
              "right_eye": 0,
              "nose": 0,
              "mouth": 0,
              "left_cheek": 0,
              "right_cheek": 0,
              "chin_contour": 0
```

```
            },
            "blur": 0,
            "illumination": 114,
            "completeness": 1
        },
        "angle": {
            "yaw": 1.59,
            "pitch": 0.26,
            "roll": -5.46
        }
    },
    {
        "pic" : "gAQTGF2YzU4LjkyLjEwMAD", //图片base64编码后的值
        "face_token": "6411f95f491fb665c389de03a33f12a4",
        "face_id": "6411f95f491fb665c389de03a33f12a4",
        "liveness_score": 0.06,
        "spoofing": 0.0001938246714,
        "quality": {
            "occlusion": {
                "left_eye": 0,
                "right_eye": 0,
                "nose": 0,
                "mouth": 0,
                "left_cheek": 0,
                "right_cheek": 0,
                "chin_contour": 0
            },
            "blur": 0,
            "illumination": 114,
            "completeness": 1
        },
        "angle": {
            "yaw": 1.59,
            "pitch": 0.26,
            "roll": -5.46
        }
    },
    {
        "pic" : "gAQTGF2YzU4LjkyLjEwMAD", //图片base64编码后的值
        "face_token": "d7fb09b7942555bf1e4b8d47a63c2e4b",
        "face_id": "d7fb09b7942555bf1e4b8d47a63c2e4b",
        "liveness_score": 0.05,
        "spoofing": 0.0001579090458
    },
    {
        "pic" : "gAQTGF2YzU4LjkyLjEwMAD", //图片base64编码后的值
        "face_token": "b14f94951a1d1b0fb8770bb1ef2bf2e7",
        "face_id": "b14f94951a1d1b0fb8770bb1ef2bf2e7",
        "liveness_score": 0.05,
        "spoofing": 0.0001889262057,
        "quality": {
            "occlusion": {
                "left_eye": 0,
                "right_eye": 0,
                "nose": 0,
                "mouth": 0,
                "left_cheek": 0,
                "right_cheek": 0,
                "chin_contour": 0
            },
            "blur": 0,
            "illumination": 114,
```

```
            "completeness": 1
        },
        "angle": {
            "yaw": 1.59,
            "pitch": 0.26,
            "roll": -5.46
        }
    },
    {
        "pic" : "gAQTGF2YzU4LjkyLjEwMAD", //图片base64编码后的值
        "face_token": "f7f363d0e71e91906dbdcfec0573de70",
        "face_id": "f7f363d0e71e91906dbdcfec0573de70",
        "liveness_score": 0.05,
        "spoofing": 0.0001999060332,
        "quality": {
            "occlusion": {
                "left_eye": 0,
                "right_eye": 0,
                "nose": 0,
                "mouth": 0,
                "left_cheek": 0,
                "right_cheek": 0,
                "chin_contour": 0
            },
            "blur": 0,
            "illumination": 114,
            "completeness": 1
        },
        "angle": {
            "yaw": 1.59,
            "pitch": 0.26,
            "roll": -5.46
        }
    },
    {
        "pic" : "gAQTGF2YzU4LjkyLjEwMAD", //图片base64编码后的值
        "face_token": "388bdccbd70200f62c5f46a84b012691",
        "face_id": "388bdccbd70200f62c5f46a84b012691",
        "liveness_score": 0.04,
        "spoofing": 0.0001798788871,
        "quality": {
            "occlusion": {
                "left_eye": 0,
                "right_eye": 0,
                "nose": 0,
                "mouth": 0,
                "left_cheek": 0,
                "right_cheek": 0,
                "chin_contour": 0
            },
            "blur": 0,
            "illumination": 114,
            "completeness": 1
        },
        "angle": {
            "yaw": 1.59,
            "pitch": 0.26,
            "roll": -5.46
        }
    },
    {
        "pic" : "gAQTGF2YzU4LjkyLjEwMAD", //图片base64编码后的值
        "face_token": "5fa4e1116f00912f66dba9b42a6a739e",
```

```
            "face_token": "[illegible]",
            "face_id": "5fa4e1116f00912f66dba9b42a6a739e",
            "liveness_score": 0.04,
            "spoofing": 0.0001976373605,
            "quality": {
              "occlusion": {
                  "left_eye": 0,
                  "right_eye": 0,
                  "nose": 0,
                  "mouth": 0,
                  "left_cheek": 0,
                  "right_cheek": 0,
                  "chin_contour": 0
              },
              "blur": 0,
              "illumination": 114,
              "completeness": 1
            },
            "angle": {
              "yaw": 1.59,
              "pitch": 0.26,
              "roll": -5.46
            }
          }
      ]
  },
  "timestamp": 1597148814,
  "cached": 0,
  "serverlogid": 1614796453,
  "error_code": 0,
  "error_msg": "SUCCESS"
}
```


活体阈值参考

请务必在产品侧做好以下条件限制

- 检测的图片为二次采集，即通过相机当场拍摄，确保时间及操作条件的约束

关于活体检测faceliveness的判断阈值选择，可参考以下数值信息：

阈值（Threshold）	误拒率（FRR）	通过率（TAR）	攻击拒绝率（TRR））
0.05	0.01%	99.99%	97.75%
0.1	0.05%	99.95%	98.33%
0.3 （推荐）	0.1%	99.9%	98.82%
0.5	0.5%	99.5%	99.67%
0.9	1%	99%	99.77%

关于以上数值的概念介绍：

- **拒绝率（TRR）**：如99%，代表100次作弊假体攻击，会有99次被拒绝。
- **误拒率（FRR）**：如0.5%，指1000次真人请求，会有5次因为活体分数低于阈值被错误拒绝。
- **通过率（TAR）**：如99%，指100次真人请求，会有99次因为活体分数高于阈值而通过。
- **阈值（Threshold）**：高于此数值，则可判断为活体。

合成图阈值参考

> 新推出合成图检测能力，在入参字段中增加spoofing参数，进行判断，若spoofing分值高于合成图推荐阈值，则可判断为合成图攻击

关于合成图检测spoofing的判断阈值选择，可参考以下数值信息：

阈值	误拒率（FRR）	通过率（TAR）	攻击拒绝率（TRR））
0.00023	5%	95%	94.93%
0.00048（**推荐**）	1%	99%	89.71%
0.00066	0.5%	99.5%	88.02%
0.00109	0.1%	99.9%	84.57%
0.00171	0.05%	99.95%	81.52%
0.00547	0.01%	99.99%	65.52%

- **阈值（Threshold）**：高于此数值，则可判断为是合成图攻击。

错误码列表

错误码	error_msg	错误信息	描述
216430	rtse/face service error	rtse/face 服务异常	请重新尝试
216431	voice service error	语音识别服务异常	请重新尝试
216432	video service call fail	视频解析服务调用失败	请重新尝试
216433	video service error	视频解析服务发生错误	请重新尝试
216434	liveness check fail	活体检测失败	请重新尝试
216500	code digit error	验证码位数错误	验证码错误， 请增加一层验证环节
216501	not found face	没有找到人脸	请查看上传视频是否包含人脸，可能因为人脸过大导致，建议用户调整距离重新录制视频
216502	session lapse	当前会话已失效	请重新获取语音验证码
216505	redis connect error	redis连接失败	请重新尝试
216506	redis operation error	redis操作失败	请重新尝试
216507	found many faces	视频中有多张人脸	请重新录制视频
216508	not found video info	没有找到视频信息	请参考文档修改视频格式
216509	voice can not identify	视频中的声音无法识别 （声音过低或者有杂 音导致无法识别）	请重新录制视频
216908	视频中人脸质量较差 (返回信息中包含具体原因)	视频中人脸质量过低 (返回的错误信息会包含 具体的错误信息包含 **illumiantion**(光照不足) **angle**(角度不好) **blur** (人脸模糊) **occlusion**(有遮挡) **too large**(人脸过大, **占屏幕2/3以上**) 等原因	请重新录制视频
222027	code length param error	验证码长度错误 (最小值大于最大值)	参考API说明文档，修改参数
222028	param[min_code_length] format error	参数格式错误	参考API说明文档，修改参数
222029	param[max_code_length] format error	参数格式错误	参考API说明文档，修改参数
222030	param[match_threshold] format error	参数格式错误	参考API说明文档，修改参数

二、实时活体检测方案

如您需要使用**实时**的活体检测方式，即无需用户录制并上传视频，直接在前端实时完成活体检测流程，以提升整体核验流程的流畅度及用户体验。可通过**H5实名认证方案**进行接入。您可根据您的业务环境选择**APP方案接入**或**H5方案接入**。

> **注：** 此实时检测能力暂不支持纯服务端接入的方式。如您需要使用此能力，需通过**H5实名认证方案**进行接入。

如果您对文档内容有任何疑问，可以通过以下几种方式联系我们：

- 在百度云控制台内 提交工单，咨询问题类型请选择**人工智能服务**；

- 如有需要讨论的疑问，欢迎进入 AI社区 与其他开发者们一同交流。

场景化搜索（公测）

能力介绍

场景化搜索当前暂只公测视频监控场景下的人脸搜索，后续将陆续开放戴口罩人脸搜索等场景。公测期间，各个接口均有免费资源用于测试，如需增加免费测试额度，请提交工单申请。

人脸搜索（视频监控）在抓拍机摄像头等设备大角度俯拍的视频监控场景予以专项优化，使用人脸搜索（视频监控）需配套使用人脸库管理（场景化）构建人脸库，具体接口如下：

> 注意：人脸搜索场景需要与人脸库管理场景匹配，人脸库中组场景与人脸场景也需保持一致，混用通用生活照场景（通用生活照文档）会导致调用报错或漏识别。

业务能力

- **人脸搜索（视频监控）**：也称为1：N识别，在指定人脸集合中，找到最相似的人脸；
- **人脸库管理（场景化）-人脸注册**：向人脸库中添加人脸
- **人脸库管理（场景化）-人脸更新**：更新人脸库中指定用户下的人脸信息
- **人脸库管理（场景化）-用户信息查询**：查询人脸库中某个用户的详细信息
- **人脸库管理（场景化）-获取用户列表**：查询指定用户组中的用户列表
- **人脸库管理（场景化）-复制用户**：将指定用户复制到另外的人脸组
- **人脸库管理（场景化）-删除用户**：删除指定用户
- **人脸库管理（场景化）-创建用户组**：创建一个新的用户组
- **人脸库管理（场景化）-删除用户组**：删除指定用户组
- **人脸库管理（场景化）-组列表查询**：查询人脸库中用户组的列表

人脸库结构

人脸库、用户组、用户、用户下的人脸**层级关系**如下所示：

```
|- 人脸库(appid)
 |- 用户组一（group_id）
    |- 用户01（uid）
       |- 人脸（faceid）
    |- 用户02（uid）
       |- 人脸（faceid）
       |- 人脸（faceid）
       ....
     ....
 |- 用户组二（group_id）
 |- 用户组三（group_id）
 ....
```

关于人脸库的设置限制

- **每个appid对应一个人脸库，且不同appid之间，人脸库互不相通**；
- 每个人脸库下，可以创建多个用户组，用户组（group）数量**没有限制**；

- 每个用户组（group）下，可添加**无限**个user_id，**无限**张人脸（注：为了保证查询速度，单个group中的人脸容量上限建议为**80万**）；
- 每个用户（user_id）所能注册的最大人脸数量**20**；

> 提醒：每个人脸库对应一个appid，一定确保不要轻易删除后台应用列表中的appid，删除后则此人脸库将失效，无法进行任何查找！

质量判断

为了保证识别效果，请控制注册人脸的质量，在调用人脸注册接口时使用质量控制和活体控制参数保证图片的质量以及注册进入人脸库的人脸是活体

调用方式

请求URL数据格式

向API服务地址使用POST发送请求，必须在URL中带上参数`access_token`，可通过后台的API Key和Secret Key生成，具体方式请参考“Access Token获取”。

示例代码

Bash

PHP

JAVA

Python

Cpp

C#

Node

```
##### !/bin/bash
curl -i -k 'https://aip.baidubce.com/oauth/2.0/token?grant_type=client_credentials&client_id=【百度云应用的AK】&client_secret=【百度云应用的SK】'
```

> **注意**：`access_token`的有效期为30天，**切记需要每30天进行定期更换，或者每次请求都拉取新token**；

例如此接口，使用HTTPS POST发送：

```
https://aip.baidubce.com/rest/2.0/face/v1/merge?
access_token=24.f9ba9c5341b67688ab4added8bc91dec.2592000.1485570332.282335-8574074
```

POST中Body的参数，按照下方请求参数说明选择即可。

> **提示**：如果您为百度云老用户，正在使用其他非AI的服务，可以参考百度云AKSK鉴权方式发送请求，虽然**请求方式**和**鉴权方法**和本文所介绍的不同，但请求参数和返回结果一致。

在线调试

您可以在 示例代码中心 中调试该接口，可进行签名验证、查看在线调用的请求内容和返回结果、示例代码的自动生成。

人脸搜索（视频监控）

请求说明

注意事项：

- **与人脸库场景匹配**：人脸搜索（视频监控）需要与人脸库管理（场景化）接口配合使用，且人脸库管理（场景化）-人脸注册/人脸更新/创建用户组接口，scene_type传入sec。
- **请求体格式化**：Content-Type为`application/json`，通过json格式化请求体。
- **Base64编码**：请求的图片需经过`Base64编码`，图片的base64编码指将图片数据编码成一串字符串，使用该字符串代替图像地址。您可以首先得到图片的二进制，然后用Base64格式编码即可。需要注意的是，图片的base64编码是不包含图片头的，如``
- **图片格式**：现支持PNG、JPG、JPEG、BMP，**不支持GIF图片**

请求示例

HTTP方法：`POST`

请求URL：`https://aip.baidubce.com/rest/2.0/face/capture/search`

URL参数：

参数	值
access_token	通过API Key和Secret Key获取的access_token,参考“Access Token获取”

Header如下：

参数	值
Content-Type	application/json

Body中放置请求参数，参数详情如下：

请求参数

参数	必选	类型	说明
image	是	string	图片信息(**总数据大小应小于10M**)，图片上传方式根据image_type来判断
image_type	是	string	图片类型 **BASE64**:图片的base64值，base64编码后的图片数据，编码后的图片大小不超过2M
group_id_list	是	string	从指定的group中进行查找 用逗号分隔，**上限10个**
quality_control	否	string	图片质量控制 **NONE**: 不进行控制 **LOW**:较低的质量要求 **NORMAL**: 一般的质量要求 **HIGH**: 较高的质量要求 **默认 NONE** 若图片质量不满足要求，则返回结果中会提示质量检测失败
liveness_control	否	string	活体检测控制 **NONE**: 不进行控制 **LOW**:较低的活体要求(高通过率 低攻击拒绝率) **NORMAL**: 一般的活体要求(平衡的攻击拒绝率, 通过率) **HIGH**: 较高的活体要求(高攻击拒绝率 低通过率) **默认NONE** 若活体检测结果不满足要求，则返回结果中会提示活体检测失败
user_id	否	string	当需要对特定用户进行比对时，指定user_id进行比对。即人脸认证功能。
max_user_num	否	string	查找后返回的用户数量。返回相似度最高的几个用户，默认为1，最多返回50个。
face_sort_type	否	string	人脸检测排序类型 **0**:代表检测出的人脸按照人脸面积从大到小排列 **1**:代表检测出的人脸按照距离图片中心从近到远排列 **默认为0**

示例代码

提示一：使用示例代码前，请记得替换其中的示例Token、图片地址或图片Base64信息。

提示二：部分语言依赖的类或库，请在代码注释中查看下载地址。

Bash

PHP

JAVA

Python

C#

```
人脸搜索
curl -i -k 'https://aip.baidubce.com/rest/2.0/face/capture/search?access_token=【调用鉴权接口获取的token】' --data '{"image":"[图片Base64编码]","image_type":"BASE64","group_id_list":"group_repeat,group_233","quality_control":"LOW","liveness_control":"NORMA' -H 'Content-Type:application/json; charset=UTF-8'
```

返回说明

返回参数

- 返回结果

字段	必选	类型	说明
log_id	是	uint64	请求标识码，随机数，唯一
face_token	是	string	人脸标志
user_list	是	array	匹配的用户信息列表
+group_id	是	string	用户所属的group_id
+user_id	是	string	用户的user_id
+user_info	是	string	注册用户时携带的user_info
+score	是	float	用户的匹配得分，推荐阈值80分

- 返回示例

```
{
 "face_token": "fid",
 "user_list": [
   {
      "group_id" : "test1",
      "user_id": "u333333",
      "user_info": "Test User",
      "score": 99.3
   }
 ]
}
```

- **质量控制参数说明**

> 不同的控制度下所对应的质量控制阈值，如果检测出来的质量信息某一项不符合控制阈值的要求，则会返回错误信息。

遮挡情况的阈值

控制度	left_eye	right_eye	nose	mouth	left_cheek	right_cheek	chin_contour
LOW	0.8	0.8	0.8	0.8	0.8	0.8	0.8
NORMAL	0.6	0.6	0.6	0.6	0.6	0.6	0.6
HIGH	0.2	0.2	0.2	0.2	0.2	0.2	0.2

模糊度、完整度的阈值

控制度	illumination	blurdegree	completeness
LOW	20	0.8	0
NORMAL	40	0.6	0
HIGH	100	0.2	1

活体控制参数说明

> 不同的控制度下所对应的活体控制阈值，如果检测出来的活体分数小于控制阈值，则会返回错误信息。

控制度	阈值	说明
LOW	0.05	活体误拒率：万分之一；拒绝率：63.9%
NORMAL	0.3	活体误拒率：千分之一；拒绝率：90.3%
HIGH	0.9	活体误拒率：百分之一；拒绝率：97.6%

> 1、误拒率: 把**真人识别为假人**的概率. 阈值越高，安全性越高, 要求也就越高, 对应的误识率就越高 2、通过率=1-误拒率

关于以上数值的概念介绍：

拒绝率（TRR）：如99%，代表100次作弊假体攻击，会有99次被拒绝。 误拒率（FRR）：如0.5%，指1000次真人请求，会有5次因为活体分数低于阈值被错误拒绝。 通过率（TAR）：如99%，指100次真人请求，会有99次因为活体分数高于阈值而通过。 阈值（Threshold）：高于此数值，则可判断为活体。

人脸库管理（场景化）-人脸注册

请求说明

注意事项：

- **与用户组场景匹配**：人脸注册接口传递的scene_type值需与用户组创建时保持一致，如混用人脸注册V3接口，会导致无法准确搜索。
- **请求体格式化**：Content-Type为`application/json`，通过json格式化请求体。
- **Base64编码**：请求的图片需经过`Base64编码`，图片的base64编码指将图片数据编码成一串字符串，使用该字符串代替图像地址。您可以首先得到图片的二进制，然后用Base64格式编码即可。需要注意的是，图片的base64编码是不包含图片头的，如``
- **图片格式**：现支持PNG、JPG、JPEG、BMP，**不支持GIF图片**

请求示例

HTTP方法：`POST`

请求URL： `https://aip.baidubce.com/rest/2.0/face/scene/faceset/user/add`

URL参数：

参数	值
access_token	通过API Key和Secret Key获取的access_token,参考“Access Token获取”

Header如下：

参数	值
Content-Type	application/json

Body中放置请求参数，参数详情如下：

请求参数

参数	必选	类型	说明
image	是	string	图片信息(**总数据大小应小于10M**)，图片上传方式根据image_type来判断。 注：组内每个uid下的人脸图片数目上限为20张
image_type	是	string	图片类型 **BASE64**:图片的base64值，base64编码后的图片数据，编码后的图片大小不超过2M；
group_id	是	string	用户组id，标识一组用户（由数字、字母、下划线组成），长度限制48B。**产品建议**：根据您的业务需求，可以将需要注册的用户，按照业务划分，分配到不同的group下，例如按照会员手机尾号作为groupid，用于刷脸支付、会员计费消费等，这样可以尽可能控制每个group下的用户数与人脸数，提升检索的准确率
user_id	是	string	用户id（由数字、字母、下划线组成），长度限制128B
scene_type	是	string	场景类型选择，**SEC**: 视频监控场景
user_info	否	string	用户资料，长度限制256B 默认空
quality_control	否	string	图片质量控制 **NONE**: 不进行控制 **LOW**:较低的质量要求 **NORMAL**: 一般的质量要求 **HIGH**: 较高的质量要求 **默认 NONE** 若图片质量不满足要求，则返回结果中会提示质量检测失败
liveness_control	否	string	活体检测控制 **NONE**: 不进行控制 **LOW**:较低的活体要求(高通过率 低攻击拒绝率) **NORMAL**: 一般的活体要求(平衡的攻击拒绝率, 通过率) **HIGH**: 较高的活体要求(高攻击拒绝率 低通过率) **默认NONE** 若活体检测结果不满足要求，则返回结果中会提示活体检测失败
action_type	否	string	**操作方式** APPEND: 当user_id在库中已经存在时，对此user_id重复注册时，新注册的图片默认会追加到该user_id下 REPLACE：当对此user_id重复注册时,则会用新图替换库中该user_id下所有图片 默认使用APPEND
face_sort_type	否	string	人脸检测排序类型 **0**:代表检测出的人脸按照人脸面积从大到小排列 **1**:代表检测出的人脸按照距离图片中心从近到远排列 **默认为0**

> **说明**：人脸注册完毕后，生效时间**一般为5s以内**，之后便可以进行人脸搜索或认证操作。

示例代码

提示一：使用示例代码前，请记得替换其中的示例Token、图片地址或Base64信息。

提示二：部分语言依赖的类或库，请在代码注释中查看下载地址。

```
Bash
```

PHP

JAVA

Python

Cpp

C#

人脸注册

```
curl -i -k 'https://aip.baidubce.com/rest/2.0/face/scene/faceset/user/add?access_token=【调用鉴权接口获取的token】' --data '{"image":"[图片Base64编码]","image_type":"BASE64","group_id":"group_repeat","user_id":"user1","scene_type":"SEC","user_info":"abc","quality_c
' -H 'Content-Type:application/json; charset=UTF-8'
```

返回说明

返回参数

字段	必选	类型	说明
log_id	是	uint64	请求标识码，随机数，唯一
face_token	是	string	人脸图片的唯一标识
location	是	array	人脸在图片中的位置
+left	是	double	人脸区域离左边界的距离
+top	是	double	人脸区域离上边界的距离
+width	是	double	人脸区域的宽度
+height	是	double	人脸区域的高度
+rotation	是	int64	人脸框相对于竖直方向的顺时针旋转角，[-180,180]

返回示例

```
{
 "face_token": "2fa64a88a9d5118916f9a303782a97d3",
 "location": {
     "left": 117,
     "top": 131,
     "width": 172,
     "height": 170,
     "rotation": 4
 }
}
```

- **质量控制参数说明**

> 不同的控制度下所对应的质量控制阈值，如果检测出来的质量信息某一项不符合控制阈值的要求，则会返回错误信息。

遮挡情况的阈值

控制度	left_eye	right_eye	nose	mouth	left_cheek	right_cheek	chin_contour
LOW	0.8	0.8	0.8	0.8	0.8	0.8	0.8
NORMAL	0.6	0.6	0.6	0.6	0.6	0.6	0.6
HIGH	0.2	0.2	0.2	0.2	0.2	0.2	0.2

模糊度、完整度的阈值

控制度	illumination	blurdegree	completeness
LOW	20	0.8	0
NORMAL	40	0.6	0
HIGH	100	0.2	1

活体控制参数说明

> 不同的控制度下所对应的活体控制阈值不同，如果检测出来的活体分数小于控制阈值，则会返回错误信息。

控制度	阈值	说明
LOW	0.05	活体误拒率：万分之一；拒绝率：97.75%
NORMAL	0.3	活体误拒率：千分之一；拒绝率：98.82%
HIGH	0.9	活体误拒率：百分之一；拒绝率：99.77%

> 1、误拒率: 把**真人识别为假人**的概率. 阈值越高，安全性越高, 要求也就越高, 对应的误识率就越高 2、通过率=1-误拒率

关于以上数值的概念介绍：

拒绝率（TRR）：如99%，代表100次作弊假体攻击，会有99次被拒绝。 误拒率（FRR）：如0.5%，指1000次真人请求，会有5次因为活体分数低于阈值被错误拒绝。 通过率（TAR）：如99%，指100次真人请求，会有99次因为活体分数高于阈值而通过。 阈值（Threshold）：高于此数值，则可判断为活体。

人脸库管理（场景化）-人脸更新

接口描述

用于对人脸库中指定用户，更新其下的人脸图像。

> **说明**：针对一个user_id执行更新操作，新上传的人脸图像将覆盖此user_id原有所有图像。

请求说明

注意事项：

- **与用户组场景匹配**：人脸更新接口传递的scene_type值需与用户组创建时保持一致，如混用人脸更新V3接口，会导致无法准

确搜索。

- **请求体格式化**：Content-Type为`application/json`，通过`json`格式化请求体。
- **Base64编码**：请求的图片需经过`Base64编码`，图片的base64编码指将图片数据编码成一串字符串，使用该字符串代替图像地址。您可以首先得到图片的二进制，然后用Base64格式编码即可。需要注意的是，图片的base64编码是不包含图片头的，如``
- **图片格式**：现支持PNG、JPG、JPEG、BMP，**不支持GIF图片**

请求示例

HTTP方法：`POST`

请求URL： `https://aip.baidubce.com/rest/2.0/face/scene/faceset/user/update`

URL参数：

参数	值
access_token	通过API Key和Secret Key获取的access_token,参考“Access Token获取”

Header如下：

参数	值
Content-Type	application/json

Body中放置请求参数，参数详情如下：

请求参数

参数	必选	类型	说明
image	是	string	图片信息(**总数据大小应小于10M**)，图片上传方式根据image_type来判断
image_type	是	string	图片类型 **BASE64**:图片的base64值，base64编码后的图片数据，编码后的图片大小不超过2M；
group_id	是	string	用户组id，标识一组用户（由数字、字母、下划线组成），长度限制128B
user_id	是	string	用户id（由数字、字母、下划线组成），长度限制48B
scene_type	是	string	场景类型选择，**SEC**: 视频监控场景
user_info	否	string	用户资料，长度限制48B 默认空
quality_control	否	string	图片质量控制 **NONE**: 不进行控制 **LOW**:较低的质量要求 **NORMAL**: 一般的质量要求 **HIGH**: 较高的质量要求 **默认 NONE** 若图片质量不满足要求，则返回结果中会提示质量检测失败
liveness_control	否	string	活体检测控制 **NONE**: 不进行控制 **LOW**:较低的活体要求(高通过率 低攻击拒绝率) **NORMAL**: 一般的活体要求(平衡的攻击拒绝率, 通过率) **HIGH**: 较高的活体要求(高攻击拒绝率 低通过率) **默认NONE** 若活体检测结果不满足要求，则返回结果中会提示活体检测失败
action_type	否	string	**操作方式** UPDATE: 会使用新图替换库中该user_id下所有图片, 若user_id不存在则会报错 REPLACE：当user_id不存在时, 则会注册这个user_id的用户 默认使用UPDATE

示例代码

提示一：使用示例代码前，请记得替换其中的示例Token、图片地址或Base64信息。

提示二：部分语言依赖的类或库，请在代码注释中查看下载地址。

Bash

PHP

JAVA

Python

Cpp

C#

人脸更新

```
curl -i -k 'https://aip.baidubce.com/rest/2.0/face/scene/faceset/user/update?access_token=【调用鉴权接口获取的token】' --data '{"image":"[图片Base64编码]","image_type":"BASE64","group_id":"group_repeat","user_id":"user1","scene_type":"SEC","user_info":"cba","quality_c
' -H 'Content-Type:application/json; charset=UTF-8'
```

返回说明

返回参数

字段	必选	类型	说明
log_id	是	uint64	请求标识码，随机数，唯一
face_token	是	string	人脸图片的唯一标识
location	是	array	人脸在图片中的位置
+left	是	double	人脸区域离左边界的距离
+top	是	double	人脸区域离上边界的距离
+width	是	double	人脸区域的宽度
+height	是	double	人脸区域的高度
+rotation	是	int64	人脸框相对于竖直方向的顺时针旋转角，[-180,180]

返回示例

```
{
 "face_token": "2fa64a88a9d5118916f9a303782a97d3",
 "location": {
     "left": 117,
     "top": 131,
     "width": 172,
     "height": 170,
     "rotation": 4
 }
}
```

人脸库管理（场景化）-用户信息查询

接口描述

获取人脸库中某个用户的信息(user_info信息和用户所属的组)。

请求说明

请求示例

HTTP方法：`POST`

请求URL： `https://aip.baidubce.com/rest/2.0/face/v3/faceset/user/get`

URL参数：

参数	值
access_token	通过API Key和Secret Key获取的access_token,参考“Access Token获取”

Header如下：

参数	值
Content-Type	application/json

Body中放置请求参数，参数详情如下：

请求参数

参数	必选	类型	说明
user_id	是	string	用户id（由数字、字母、下划线组成），长度限制48B
group_id	是	string	用户组id(由数字、字母、下划线组成，长度限制48B)，如传入“**@ALL**”则从所有组中查询用户信息。注：处于不同组，但uid相同的用户，我们认为是同一个用户。

示例代码

提示一：使用示例代码前，请记得替换其中的示例Token、图片地址或Base64信息。

提示二：部分语言依赖的类或库，请在代码注释中查看下载地址。

Bash
PHP
JAVA
Python
Cpp
C#

用户信息查询

```
curl -i -k 'https://aip.baidubce.com/rest/2.0/face/v3/faceset/user/get?access_token=【调用鉴权接口获取的token】' --data '{"user_id":"user1","group_id":"group1"}' -H 'Content-Type:application/json; charset=UTF-8'
```

返回说明

返回参数

字段	必选	类型	说明
log_id	是	uint64	请求标识码，随机数，唯一
user_list	是	array	查询到的用户列表
+user_info	是	string	用户资料，被查询用户的资料
+group_id	是	string	用户组id，被查询用户的所在组

返回示例

```
{
 "user_list": [
    {
        "user_info": "user info ...",
        "group_id": "gid1"
    },
    {
        "user_info": "user info2 ...",
        "group_id": "gid2"
    }
 ]
}
```

人脸库管理（场景化）-获取用户列表

接口描述

用于查询指定用户组中的用户列表。

请求说明

请求示例

HTTP方法：`POST`

请求URL： `https://aip.baidubce.com/rest/2.0/face/v3/faceset/group/getusers`

URL参数：

参数	值
access_token	通过API Key和Secret Key获取的access_token,参考“Access Token获取”

Header如下：

参数	值
Content-Type	application/json

Body中放置请求参数，参数详情如下：

请求参数

参数	必选	类型	说明
group_id	是	string	用户组id，长度限制48B
start	否	uint32	默认值0，起始序号
length	否	uint32	返回数量，默认值100，最大值1000

示例代码

提示一：使用示例代码前，请记得替换其中的示例Token、图片地址或Base64信息。

提示二：部分语言依赖的类或库，请在代码注释中查看下载地址。

Bash

PHP

JAVA

Python

Cpp

C#

```
 获取用户列表
curl -i -k 'https://aip.baidubce.com/rest/2.0/face/v3/faceset/group/getusers?access_token=【调用鉴权接口获取的token】' --data '{"group_id":"group1"}' -H 'Content-Type:application/json; charset=UTF-8'
```

返回说明

- 返回结果

字段	必选	类型	说明
user_id_list	是	array	用户ID列表

- 返回示例

```
{
  "user_id_list": [
    "uid1",
    "uid2"
  ]
}
```

人脸库管理（场景化）-复制用户

接口描述

用于将已经存在于人脸库中的用户**复制到一个新的组**。

> **说明**：并不是向一个指定组内添加用户，而是直接从其它组复制用户信息 如果需要注册用户，请直接使用**人脸注册接口**

请求说明

请求示例

HTTP方法：`POST`

请求URL： `https://aip.baidubce.com/rest/2.0/face/v3/faceset/user/copy`

URL参数：

参数	值
access_token	通过API Key和Secret Key获取的access_token,参考“Access Token获取

Header如下：

参数	值
Content-Type	application/json

Body中放置请求参数，参数详情如下：

请求参数

参数	必选	类型	说明
user_id	是	string	用户id，长度限制48B
src_group_id	是	string	从指定组里复制信息
dst_group_id	是	string	需要添加用户的组id

返回说明

通过返回的error_code判断是否成功 如失败则查看error_msg获得具体错误信息

返回示例

```
 // 正确返回值
{
  "error_code": 0,
 "log_id": 3314921889,
}
// 发生错误时返回值
{
 "error_code": 223111,
 "log_id": 3111284097,
 "error_msg": "dst group is not exist"
}
```

人脸库管理（场景化）-删除用户

接口描述

用于将用户从某个组中删除。

请求说明

请求示例

HTTP方法：`POST`

请求URL： `https://aip.baidubce.com/rest/2.0/face/v3/faceset/user/delete`

URL参数：

参数	值
access_token	通过API Key和Secret Key获取的access_token,参考“Access Token获取”

Header如下：

参数	值
Content-Type	application/json

Body中放置请求参数，参数详情如下：

请求参数

参数	必选	类型	说明
group_id	是	string	用户组id(由数字、字母、下划线组成，长度限制48B），如传入“**@ALL**”则从所有组中删除用户
user_id	是	string	用户id（由数字、字母、下划线组成），长度限制48B

返回说明

通过返回的error_code判断是否成功 如失败则查看error_msg获得具体错误信息：

返回参数

字段	必选	类型	说明
log_id	是	uint64	请求标识码，随机数，唯一

返回示例

```
 // 正确返回值
{

 "error_code": 0,
 "log_id": 3314921889,
}
// 发生错误时返回值
{
 "error_code": 223103,
 "log_id": 815967402,
 "error_msg": "user is not exist"
}
```

人脸库管理（场景化）-创建用户组

接口描述

用于创建一个空的用户组，如果用户组已存在 则返回错误。创建用户组时传递的场景化值scene_type，一经创建无法更改，在该组下注册/更新人脸以及人脸搜索时，场景均需保持一致。

请求说明

请求示例

HTTP方法：`POST`

请求URL： `https://aip.baidubce.com/rest/2.0/face/scene/faceset/group/add`

URL参数：

参数	值
access_token	通过API Key和Secret Key获取的access_token,参考“Access Token获取”

Header如下：

参数	值
Content-Type	application/json

Body中放置请求参数，参数详情如下：

请求参数

参数	必选	类型	说明
group_id	是	string	用户组id，标识一组用户（由数字、字母、下划线组成），长度限制48B。
scene_type	是	string	场景类型选择，**SEC**: 视频监控场景

示例代码

提示一：使用示例代码前，请记得替换其中的示例Token、图片地址或Base64信息。

提示二：部分语言依赖的类或库，请在代码注释中查看下载地址。

Bash

PHP

JAVA

Python

Cpp

C#

创建用户组

```
curl -i -k 'https://aip.baidubce.com/rest/2.0/face/scene/faceset/group/add?access_token=【调用鉴权接口获取的token】' --data '{"group_id":"group1","scene_type":"SEC"}' -H 'Content-Type:application/json; charset=UTF-8'
```

返回说明

通过返回的error_code判断是否成功 如失败则查看error_msg获得具体错误信息：

返回参数

字段	必选	类型	说明
log_id	是	uint64	请求标识码，随机数，唯一

返回示例

```
// 正确返回值
{

  "error_code": 0,
  "log_id": 3314921889,
}
// 发生错误时返回值
{
  "error_code":  223101,
  "log_id": 815967402,
  "error_msg": " group is already exist"
}
```

人脸库管理（场景化）-删除用户组

接口描述

删除用户组下所有的用户及人脸，如果组不存在 则返回错误。 注：组内的人脸数量如果大于500条，会在后台异步进行删除。在删除期间，无法向该组中添加人脸。1秒钟可以删除20条记录，相当于一小时可以将7万人的人脸组删除干净。

请求说明

请求示例

HTTP方法：POST

请求URL： https://aip.baidubce.com/rest/2.0/face/v3/faceset/group/delete

URL参数：

参数	值
access_token	通过API Key和Secret Key获取的access_token,参考“Access Token获取”

Header如下：

参数	值
Content-Type	application/json

Body中放置请求参数，参数详情如下：

请求参数

参数	必选	类型	说明
group_id	是	string	用户组id，标识一组用户（由数字、字母、下划线组成），长度限48B。

示例代码

提示一：使用示例代码前，请记得替换其中的示例Token、图片地址或Base64信息。

提示二：部分语言依赖的类或库，请在代码注释中查看下载地址。

Bash

PHP

JAVA

Python

Cpp

C#

删除用户组

```
curl -i -k 'https://aip.baidubce.com/rest/2.0/face/v3/faceset/group/delete?access_token=【调用鉴权接口获取的token】' --data '{"group_id":"group1"}' -H 'Content-Type:application/json; charset=UTF-8'
```

返回说明

通过返回的error_code判断是否成功 如失败则查看error_msg获得具体错误信息：

返回参数

字段	必选	类型	说明
log_id	是	uint64	请求标识码，随机数，唯一

返回示例

```
 // 正确返回值
{

 "error_code":0,
 "log_id": 3314921889,
}
// 发生错误时返回值
{
 "error_code": 223100,
 "log_id": 815967402,
 "error_msg": " group is not exist"
}
```

人脸库管理（场景化）-组列表查询

接口描述

用于查询用户组的列表。

请求说明

请求示例

HTTP方法：`POST`

请求URL： `https://aip.baidubce.com/rest/2.0/face/v3/faceset/group/getlist`

URL参数：

参数	值
access_token	通过API Key和Secret Key获取的access_token,参考“Access Token获取”

Header如下：

参数	值
Content-Type	application/json

Body中放置请求参数，参数详情如下：

请求参数

参数	必选	类型	说明
start	否	uint32	默认值0，起始序号
length	否	uint32	返回数量，默认值100，最大值1000

示例代码

提示一：使用示例代码前，请记得替换其中的示例Token、图片地址或Base64信息。

提示二：部分语言依赖的类或库，请在代码注释中查看下载地址。

Bash

PHP

JAVA

Python

Cpp

C#

```
 组列表查询
curl -i -k 'https://aip.baidubce.com/rest/2.0/face/v3/faceset/group/getlist?access_token=【调用鉴权接口获取的token】' --data '{"start":0,"length":100}' -H 'Content-Type:application/json; charset=UTF-8'
```

返回说明

返回参数

- 返回结果

字段	必选	类型	说明
group_id_list	是	array	group

- 返回示例

```
{
  "group_id_list": [
    "gid1",
    "gid2"
  ]
}
```

错误码

请参考人脸识别错误码

REST-API-SDK

Python-SDK

简介

Hi，您好，欢迎使用百度人脸识别服务。

本文档主要针对Python开发者，描述百度人脸识别接口服务的相关技术内容。如果您对文档内容有任何疑问，可以通过以下几种方式联系我们：

- 在百度云控制台内提交工单，咨询问题类型请选择**人工智能服务**；
- 如有疑问，进入AI社区交流：http://ai.baidu.com/forum/topic/list/165

接口能力

接口名称	接口能力简要描述
人脸检测	检测人脸并定位，返回五官关键点，及人脸各属性值
人脸比对	返回两两比对的人脸相似值
人脸查找	在一个人脸集合中找到找到相似的人脸，由一系列接口组成，包括人脸识别、人脸认证、人脸库管理相关接口（人脸注册、人脸更新、人脸删除、用户信息查询、组列表查询、组内用户列表查询、组间复制用户、组内删除用户）

版本更新记录

上线日期	版本号	更新内容
2019.4.17	2.2.15	人脸v3文档更新，新增N:M接口
2018.5.10	2.2.4	修复人脸V3问题
2018.4.28	2.2.3	全面切换为人脸V3接口
2018.4.9	2.2.2	新增身份验证，在线活体检测接口
2018.01.12	2.1.0	新增M：N多人脸识别
2017.12.22	2.0.0	SDK代码重构
2017.5.11	1.0.0	人脸识别服务上线

快速入门

安装人脸识别 Python SDK

人脸识别 Python SDK目录结构

```
 ├─── README.md
├─── aip                  //SDK目录
│   ├─── __init__.py      //导出类
│   ├─── base.py          //aip基类
│   ├─── http.py          //http请求
│   └─── face.py //人脸识别
└─── setup.py             //setuptools安装
```

支持Python版本：2.7.+ ,3.+

安装使用Python SDK有如下方式：

- 如果已安装pip，执行`pip install baidu-aip`即可。
- 如果已安装setuptools，执行`python setup.py install`即可。

新建AipFace

AipFace是人脸识别的Python SDK客户端，为使用人脸识别的开发人员提供了一系列的交互方法。

参考如下代码新建一个AipFace：

```
 from aip import AipFace

""" 你的 APPID AK SK """
APP_ID = '你的 App ID'
API_KEY = '你的 Api Key'
SECRET_KEY = '你的 Secret Key'

client = AipFace(APP_ID, API_KEY, SECRET_KEY)
```

在上面代码中，常量APP_ID在百度云控制台中创建，常量API_KEY与SECRET_KEY是在创建完毕应用后，系统分配给用户的，均为字符串，用于标识用户，为访问做签名验证，可在AI服务控制台中的**应用列表**中查看。

注意：如您以前是百度云的老用户，其中API_KEY对应百度云的“Access Key ID”，SECRET_KEY对应百度云的“Access Key Secret”。

配置AipFace

如果用户需要配置AipFace的网络请求参数(一般不需要配置)，可以在构造AipFace之后调用接口设置参数，目前只支持以下参数：

接口	说明
setConnectionTimeoutInMillis	建立连接的超时时间（单位：毫秒
setSocketTimeoutInMillis	通过打开的连接传输数据的超时时间（单位：毫秒）

接口说明

人脸检测

人脸检测：检测图片中的人脸并标记出位置信息;

```
 image = "取决于image_type参数，传入BASE64字符串或URL字符串或FACE_TOKEN字符串"

imageType = "BASE64"

""" 调用人脸检测 """
client.detect(image, imageType);

""" 如果有可选参数 """
options = {}
options["face_field"] = "age"
options["max_face_num"] = 2
options["face_type"] = "LIVE"
options["liveness_control"] = "LOW"

""" 带参数调用人脸检测 """
client.detect(image, imageType, options)
```

python人脸检测 请求参数详情**

参数名称	是否必选	类型	默认值	说明
image	是	string		图片信息(**总数据大小应小于10M**)，图片上传方式根据image_type来判断
image_type	是	string		图片类型 **BASE64**:图片的base64值，base64编码后的图片数据，编码后的图片大小不超过2M；**URL**:图片的 URL地址(可能由于网络等原因导致下载图片时间过长)； **FACE_TOKEN**: 人脸图片的唯一标识，调用人脸检测接口时，会为每个人脸图片赋予一个唯一的FACE_TOKEN，同一张图片多次检测得到的FACE_TOKEN是同一个
face_field	否	string		包括**age,expression,face_shape,gender,glasses,landmark，landmark150,quality,eye_status,emotion,face_type信息** 逗号分隔. 默认只返回face_token、人脸框、概率和旋转角度
max_face_num	否	string	1	最多处理人脸的数目，默认值为1，仅检测图片中面积最大的那个人脸；**最大值10**，检测图片中面积最大的几张人脸。
face_type	否	string		人脸的类型 **LIVE**表示生活照：通常为手机、相机拍摄的人像图片、或从网络获取的人像图片等**IDCARD**表示身份证芯片照：二代身份证内置芯片中的人像照片 **WATERMARK**表示带水印证件照：一般为带水印的小图，如公安网小图 **CERT**表示证件照片：如拍摄的身份证、工卡、护照、学生证等证件图片 默认**LIVE**
liveness_control	否	string	NONE	活体检测控制 **NONE**: 不进行控制 **LOW**:较低的活体要求(高通过率 低攻击拒绝率) **NORMAL**: 一般的活体要求(平衡的攻击拒绝率, 通过率) **HIGH**: 较高的活体要求(高攻击拒绝率 低通过率) **默认NONE**

人脸检测 返回数据参数详情

字段	必选	类型	说明
face_num	是	int	检测到的图片中的人脸数量
face_list	是	array	人脸信息列表，具体包含的参数参考下面的列表。
+face_token	是	string	人脸图片的唯一标识
+location	是	array	人脸在图片中的位置
++left	是	double	人脸区域离左边界的距离
++top	是	double	人脸区域离上边界的距离
++width	是	double	人脸区域的宽度
++height	是	double	人脸区域的高度
++rotation	是	int64	人脸框相对于竖直方向的顺时针旋转角，[-180,180]
+face_probability	是	double	人脸置信度，范围【0~1】，代表这是一张人脸的概率，0最小、1最大。
+angle	是	array	人脸旋转角度参数
++yaw	是	double	三维旋转之左右旋转角[-90(左), 90(右)]
++pitch	是	double	三维旋转之俯仰角度[-90(上), 90(下)]
++roll	是	double	平面内旋转角[-180(逆时针), 180(顺时针)]
+age	否	double	年龄 ，当**face_field包含age时返回**
+expression	否	array	表情，当 **face_field包含expression时返回**
++type	否	string	**none**:不笑；**smile**:微笑；**laugh**:大笑
++probability	否	double	表情置信度，范围【0~1】，0最小、1最大。
+face_shape	否	array	脸型，当**face_field包含face_shape时返回**
++type	否	double	**square**: 正方形 **triangle**:三角形 **oval**: 椭圆 **heart**: 心形 **round**: 圆形

++probability	否	double	置信度，范围【0~1】，代表这是人脸形状判断正确的概率，0最小、1最大。
+gender	否	array	性别，**face_field包含gender时返回**
++type	否	string	male:**男性** female:**女性**
++probability	否	double	性别置信度，范围【0~1】，0代表概率最小、1代表最大。
+glasses	否	array	是否带眼镜，**face_field包含glasses时返回**
++type	否	string	**none**:无眼镜，**common**:普通眼镜，**sun**:墨镜
++probability	否	double	眼镜置信度，范围【0~1】，0代表概率最小、1代表最大。
+eye_status	否	array	双眼状态（睁开/闭合） **face_field包含eye_status时返回**
++left_eye	否	double	左眼状态 [0,1]取值，越接近0闭合的可能性越大
++right_eye	否	double	右眼状态 [0,1]取值，越接近0闭合的可能性越大
+emotion	否	array	情绪 **face_field包含emotion时返回**
++type	否	string	**angry**:愤怒 **disgust**:厌恶 **fear**:恐惧 **happy**:高兴 **sad**:伤心 **surprise**:惊讶 **neutral**:无情绪
++probability	否	double	情绪置信度，范围0~1
++probability	否	double	人种置信度，范围【0~1】，0代表概率最小、1代表最大。
+face_type	否	array	真实人脸/卡通人脸 **face_field包含face_type时返回**
++type	否	string	**human**: 真实人脸 **cartoon**: 卡通人脸
++probability	否	double	人脸类型判断正确的置信度，范围【0~1】，0代表概率最小、1代表最大。
+landmark	否	array	4个关键点位置，左眼中心、右眼中心、鼻尖、嘴中心。**face_field包含landmark时返回**
+landmark72	否	array	72个特征点位置 **face_field包含landmark时返回**
+landmark150	否	array	150个特征点位置 **face_field包含landmark150时返回**
+quality	否	array	人脸质量信息。**face_field包含quality时返回**
++occlusion	否	array	人脸各部分遮挡的概率，范围[0~1]，0表示完整，1表示不完整
+++left_eye	否	double	左眼遮挡比例，[0-1] ，1表示完全遮挡
+++right_eye	否	double	右眼遮挡比例，[0-1] ， 1表示完全遮挡
+++nose	否	double	鼻子遮挡比例，[0-1] ， 1表示完全遮挡
+++mouth	否	double	嘴巴遮挡比例，[0-1] ， 1表示完全遮挡
+++left_cheek	否	double	左脸颊遮挡比例，[0-1] ， 1表示完全遮挡
+++right_cheek	否	double	右脸颊遮挡比例，[0-1] ， 1表示完全遮挡
+++chin_contour	否	double	下巴遮挡比例，，[0-1] ， 1表示完全遮挡
++blur	否	double	人脸模糊程度，范围[0~1]，0表示清晰，1表示模糊
++illumination	否	double	取值范围在[0~255], 表示脸部区域的光照程度 越大表示光照越好
++completeness	否	int64	人脸完整度，0或1, 0为人脸溢出图像边界，1为人脸都在图像边界内

人脸检测 返回示例

```
{
"face_num": 1,
"face_list": [
    {
        "face_token": "35235asfas21421fakghktyfdgh68bio",
        "location": {
            "left": 117,
```

```
            "top": 131,
            "width": 172,
            "height": 170,
            "rotation": 4
        },
        "face_probability": 1,
        "angle" :{
            "yaw" : -0.34859421849251
            "pitch" 1.9135693311691
            "roll" :2.3033397197723
        }
        "landmark": [
            {
                "x": 161.74819946289,
                "y": 163.30244445801
            },
            ...
        ],
        "landmark72": [
            {
                "x": 115.86531066895,
                "y": 170.0546875
            } ,
            ...
        ],
        "age": 29.298097610474,
        "expression": {
            "type": "smile",
            "probability" : 0.5543018579483
        },
        "gender": {
            "type": "male",
            "probability": 0.99979132413864
        },
        "glasses": {
       "type": "sun",
            "probability": 0.99999964237213
        },
        "face_shape": {
            "type": "triangle",
            "probability": 0.5543018579483
        }
        "quality": {
            "occlusion": {
                "left_eye": 0,
                "right_eye": 0,
                "nose": 0,
                "mouth": 0,
                "left_cheek": 0.0064102564938366,
                "right_cheek": 0.0057411273010075,
                "chin": 0
            },
            "blur": 1.1886881756684e-10,
            "illumination": 141,
            "completeness": 1
        }
    }
  ]
}
```


72个关键点分布图（对应landmark72个点的顺序，序号从0-

71）：https://ai.bdstatic.com/file/52BC00FFD4754A6298D977EDAD033DA0

人脸搜索

- **1：N人脸搜索**：也称为1：N识别，在指定人脸集合中，找到最相似的人脸；
- **1：N人脸认证**：基于uid维度的1：N识别，由于uid已经锁定固定数量的人脸，所以检索范围更聚焦；

> **1：N人脸识别**与**1：N人脸认证**的差别在于：人脸搜索是在指定人脸集合中进行直接地人脸检索操作，而人脸认证是基于uid，先调取这个uid对应的人脸，再在这个uid对应的人脸集合中进行检索（因为每个uid通常对应的只有一张人脸，所以通常也就变为了1：1对比）；实际应用中，人脸认证需要用户或系统先输入id，这增加了验证安全度，但也增加了复杂度，具体使用哪个接口需要视您的业务场景判断。

```
image = "取决于image_type参数，传入BASE64字符串或URL字符串或FACE_TOKEN字符串"

imageType = "BASE64"

groupIdList = "3,2"

""" 调用人脸搜索 """
client.search(image, imageType, groupIdList);

""" 如果有可选参数 """
options = {}
options["match_threshold"] = 70
options["quality_control"] = "NORMAL"
options["liveness_control"] = "LOW"
options["user_id"] = "233451"
options["max_user_num"] = 3

""" 带参数调用人脸搜索 """
client.search(image, imageType, groupIdList, options)
```

人脸搜索 请求参数详情

参数名称	是否必选	类型	默认值	说明
image	是	string		图片信息(**总数据大小应小于10M**)，图片上传方式根据image_type来判断
image_type	是	string		图片类型 **BASE64**:图片的base64值，base64编码后的图片数据，编码后的图片大小不超过2M；**URL**:图片的 URL地址(可能由于网络等原因导致下载图片时间过长)； **FACE_TOKEN**: 人脸图片的唯一标识，调用人脸检测接口时，会为每个人脸图片赋予一个唯一的FACE_TOKEN，同一张图片多次检测得到的FACE_TOKEN是同一个
group_id_list	是	string		从指定的group中进行查找 用逗号分隔，**上限10个**
match_threshold	否	string		匹配阈值（设置阈值后，score低于此阈值的用户信息将不会返回） 最大100 最小0 默认80 **此阈值设置得越高，检索速度将会越快，推荐使用默认阈值80**
quality_control	否	string	NONE	图片质量控制 **NONE**: 不进行控制 **LOW**:较低的质量要求 **NORMAL**: 一般的质量要求 **HIGH**: 较高的质量要求 **默认 NONE**
liveness_control	否	string	NONE	活体检测控制 **NONE**: 不进行控制 **LOW**:较低的活体要求(高通过率 低攻击拒绝率) **NORMAL**: 一般的活体要求(平衡的攻击拒绝率, 通过率) **HIGH**: 较高的活体要求(高攻击拒绝率 低通过率) **默认NONE**
user_id	否	string		当需要对特定用户进行比对时，指定user_id进行比对。即人脸认证功能。
max_user_num	否	string		查找后返回的用户数量。返回相似度最高的几个用户，默认为1，最多返回50个。

人脸搜索 返回数据参数详情

字段	必选	类型	说明
face_token	是	string	人脸标志
user_list	是	array	匹配的用户信息列表
+group_id	是	string	用户所属的group_id
+user_id	是	string	用户的user_id
+user_info	是	string	注册用户时携带的user_info
+score	是	float	用户的匹配得分

人脸搜索 返回示例

```
{
 "face_token": "fid",
 "user_list": [
   {
     "group_id" : "test1",
     "user_id": "u333333",
     "user_info": "Test User",
     "score": 99.3
   }
 ]
}
```

人脸搜索 M:N 识别

待识别的图片中，存在多张人脸的情况下，支持在一个人脸库中，一次请求，同时返回图片中所有人脸的识别结果。

```
image = "取决于image_type参数，传入BASE64字符串或URL字符串或FACE_TOKEN字符串"

imageType = "BASE64"

groupIdList = "3,2"

""" 调用人脸搜索 M:N 识别 """
client.multiSearch(image, imageType, groupIdList);

""" 如果有可选参数 """
options = {}
options["max_face_num"] = 3
options["match_threshold"] = 70
options["quality_control"] = "NORMAL"
options["liveness_control"] = "LOW"
options["max_user_num"] = 3

""" 带参数调用人脸搜索 M:N 识别 """
client.multiSearch(image, imageType, groupIdList, options)
```

人脸搜索 M:N 识别 请求参数详情

参数名称	是否必选	类型	默认值	说明
image	是	string		图片信息(**总数据大小应小于10M**)，图片上传方式根据image_type来判断
image_type	是	string		图片类型 **BASE64**:图片的base64值，base64编码后的图片数据，编码后的图片大小不超过2M；**URL**:图片的 URL地址(可能由于网络等原因导致下载图片时间过长)； **FACE_TOKEN**: 人脸图片的唯一标识，调用人脸检测接口时，会为每个人脸图片赋予一个唯一的FACE_TOKEN，同一张图片多次检测得到的FACE_TOKEN是同一个
group_id_list	是	string		从指定的group中进行查找 用逗号分隔，**上限10个**
max_face_num	否	string		最多处理人脸的数目 **默认值为1(仅检测图片中面积最大的那个人脸) 最大值10**
match_threshold	否	string		匹配阈值（设置阈值后，score低于此阈值的用户信息将不会返回） 最大100 最小0 默认80 **此阈值设置得越高，检索速度将会越快，推荐使用默认阈值80**
quality_control	否	string	NONE	图片质量控制 **NONE**: 不进行控制 **LOW**:较低的质量要求 **NORMAL**: 一般的质量要求 **HIGH**: 较高的质量要求 **默认 NONE**
liveness_control	否	string	NONE	活体检测控制 **NONE**: 不进行控制 **LOW**:较低的活体要求(高通过率 低攻击拒绝率) **NORMAL**: 一般的活体要求(平衡的攻击拒绝率, 通过率) **HIGH**: 较高的活体要求(高攻击拒绝率 低通过率) **默认NONE**
max_user_num	否	string		查找后返回的用户数量。返回相似度最高的几个用户，默认为1，最多返回20个。

人脸搜索 M:N 识别 返回数据参数详情

字段	必选	类型	说明
face_num	是	int	图片中的人脸数量
face_list	是	array	人脸信息列表
+face_token	是	string	人脸标志
+location	是	array	人脸在图片中的位置
++left	是	double	人脸区域离左边界的距离
++top	是	double	人脸区域离上边界的距离
++width	是	double	人脸区域的宽度
++height	是	double	人脸区域的高度
++rotation	是	int64	人脸框相对于竖直方向的顺时针旋转角，[-180,180]
+user_list	是	array	匹配的用户信息列表
++group_id	是	string	用户所属的group_id
++user_id	是	string	用户的user_id
++user_info	是	string	注册用户时携带的user_info
++score	是	float	用户的匹配得分 **80分以上可以判断为同一人，此分值对应万分之一误识率**

人脸搜索 M:N 识别 返回示例

```
{
  "error_code": 0,
  "error_msg": "SUCCESS",
  "log_id": 240483475,
  "timestamp": 1535533440,
  "cached": 0,
  "result": {
    "face_num": 2,
    "face_list": [
      {
        "face_token": "6fe19a6ee0c4233db9b5bba4dc2b9233",
        "location": {
          "left": 31.95568085,
          "top": 120.3764267,
          "width": 87,
          "height": 85,
          "rotation": -5
        },
        "user_list": [
          {
            "group_id": "group1",
            "user_id": "5abd24fd062e49bfa906b257ec40d284",
            "user_info": "userinfo1",
            "score": 69.85684967041
          },
          {
            "group_id": "group1",
            "user_id": "2abf89cffb31473a9948268fde9e1c3f",
            "user_info": "userinfo2",
            "score": 66.586112976074
          }
        ]
      },
      {
        "face_token": "fde61e9c074f48cf2bbb319e42634f41",
        "location": {
          "left": 219.4467773,
          "top": 104.7486954,
          "width": 81,
          "height": 77,
          "rotation": 3
        },
        "user_list": [
          {
            "group_id": "group1",
            "user_id": "088717532b094c3990755e91250adf7d",
            "user_info": "userinfo",
            "score": 65.154159545898
          }
        ]
      }
    ]
  }
}
```


人脸注册

用于从人脸库中新增用户，可以设定多个用户所在组，及组内用户的人脸图片，

典型应用场景：构建您的人脸库，如**会员人脸注册**，**已有用户补全人脸信息**等。

人脸库、用户组、用户、用户下的人脸**层级关系**如下所示：

```
|- 人脸库
 |- 用户组一
   |- 用户01
     |- 人脸
   |- 用户02
     |- 人脸
     |- 人脸
     ....
    ....
 |- 用户组二
 |- 用户组三
 |- 用户组四
 ....
```

关于人脸库的设置限制

每个appid对应一个人脸库，且不同appid之间，人脸库互不相通；

- 每个人脸库下，可以创建多个用户组，用户组（group）数量**没有限制**；
- 每个用户组（group）下，可添加最多**无限**张人脸，**无限**个uid；
- 每个用户（uid）所能注册的最大人脸数量**20个**；

为了保证识别效果，请控制注册人脸的质量（通过`/detect`人脸检测接口判断），具体参数可详见下表所示：

质量判断

可通过人脸检测接口，基于以下字段和对应阈值，进行质量检测的判断，以保证人脸质量符合后续业务操作要求。

指标	字段与解释	推荐数值界限
遮挡范围	**occlusion**（0~1），0为无遮挡，1是完全遮挡 含有多个具体子字段，表示脸部多个部位 通常用作判断头发、墨镜、口罩等遮挡	left_eye：0.6, #左眼被遮挡的阈值 right_eye：0.6, #右眼被遮挡的阈值 nose：0.7, #鼻子被遮挡的阈值 mouth：0.7, #嘴巴被遮挡的阈值 left_check：0.8, #左脸颊被遮挡的阈值 right_check：0.8, #右脸颊被遮挡的阈值 chin_contour：0.6, #下巴被遮挡阈值
模糊度范围	**Blur**（0~1），0是最清晰，1是最模糊	小于0.7
光照范围	**illumination**（0~255） 脸部光照的灰度值，0表示光照不好 以及对应客户端SDK中，YUV的Y分量	大于40
姿态角度	**Pitch**：三维旋转之俯仰角度[-90(上), 90(下)] **Roll**：平面内旋转角[-180(逆时针), 180(顺时针)] **Yaw**：三维旋转之左右旋转角[-90(左), 90(右)]	分别小于20度
人脸完整度	**completeness**（0或1），0为人脸溢出图像边界，1为人脸都在图像边界内	视业务逻辑判断
人脸大小	人脸部分的大小 建议长宽像素值范围：80*80~200*200	人脸部分不小于**100*100**像素

```
 image = "取决于image_type参数，传入BASE64字符串或URL字符串或FACE_TOKEN字符串"

imageType = "BASE64"

groupId = "group1"

userId = "user1"

""" 调用人脸注册 """
client.addUser(image, imageType, groupId, userId);

""" 如果有可选参数 """
options = {}
options["user_info"] = "user's info"
options["quality_control"] = "NORMAL"
options["liveness_control"] = "LOW"
options["action_type"] = "REPLACE"

""" 带参数调用人脸注册 """
client.addUser(image, imageType, groupId, userId, options)
```

人脸注册 请求参数详情

参数名称	是否必选	类型	默认值	说明
image	是	string		图片信息(总数据大小应小于10M)，图片上传方式根据image_type来判断。注：组内每个uid下的人脸图片数目上限为20张
image_type	是	string		图片类型 **BASE64**:图片的base64值，base64编码后的图片数据，编码后的图片大小不超过2M；**URL**:图片的 URL地址(可能由于网络等原因导致下载图片时间过长)；**FACE_TOKEN**: 人脸图片的唯一标识，调用人脸检测接口时，会为每个人脸图片赋予一个唯一的FACE_TOKEN，同一张图片多次检测得到的FACE_TOKEN是同一个
group_id	是	string		用户组id（由数字、字母、下划线组成），长度限制128B
user_id	是	string		用户id（由数字、字母、下划线组成），长度限制128B
user_info	否	string		用户资料，长度限制256B
quality_control	否	string	NONE	图片质量控制 **NONE**: 不进行控制 **LOW**:较低的质量要求 **NORMAL**: 一般的质量要求 **HIGH**: 较高的质量要求 **默认 NONE**
liveness_control	否	string	NONE	活体检测控制 **NONE**: 不进行控制 **LOW**:较低的活体要求(高通过率 低攻击拒绝率) **NORMAL**: 一般的活体要求(平衡的攻击拒绝率, 通过率) **HIGH**: 较高的活体要求(高攻击拒绝率 低通过率) **默认NONE**
action_type	否	string	APPEND	操作方式 APPEND: 当user_id在库中已经存在时，对此user_id重复注册时，新注册的图片默认会追加到该user_id下,REPLACE：当对此user_id重复注册时,则会用新图替换库中该user_id下所有图片,默认使用APPEND

人脸注册 返回数据参数详情

字段	必选	类型	说明
log_id	是	uint64	请求标识码，随机数，唯一
face_token	是	string	人脸图片的唯一标识
location	是	array	人脸在图片中的位置
+left	是	double	人脸区域离左边界的距离
+top	是	double	人脸区域离上边界的距离
+width	是	double	人脸区域的宽度
+height	是	double	人脸区域的高度
+rotation	是	int64	人脸框相对于竖直方向的顺时针旋转角，[-180,180]

人脸注册 返回示例

```
{
 "face_token": "2fa64a88a9d5118916f9a303782a97d3",
 "location": {
     "left": 117,
     "top": 131,
     "width": 172,
     "height": 170,
     "rotation": 4
 }
}
```

人脸更新

用于对人脸库中指定用户，更新其下的人脸图像。

> **说明：**针对一个uid执行更新操作，新上传的人脸图像将覆盖此uid原有所有图像。

```
 image = "取决于image_type参数，传入BASE64字符串或URL字符串或FACE_TOKEN字符串"

imageType = "BASE64"

groupId = "group1"

userId = "user1"

""" 调用人脸更新 """
client.updateUser(image, imageType, groupId, userId);

""" 如果有可选参数 """
options = {}
options["user_info"] = "user's info"
options["quality_control"] = "NORMAL"
options["liveness_control"] = "LOW"
options["action_type"] = "REPLACE"

""" 带参数调用人脸更新 """
client.updateUser(image, imageType, groupId, userId, options)
```

人脸更新 请求参数详情

参数名称	是否必选	类型	默认值	说明
image	是	string		图片信息(**总数据大小应小于10M**)，图片上传方式根据image_type来判断
image_type	是	string		图片类型 **BASE64**:图片的base64值，base64编码后的图片数据，编码后的图片大小不超过2M；**URL**:图片的 URL地址(可能由于网络等原因导致下载图片时间过长)；**FACE_TOKEN**: 人脸图片的唯一标识，调用人脸检测接口时，会为每个人脸图片赋予一个唯一的FACE_TOKEN，同一张图片多次检测得到的FACE_TOKEN是同一个
group_id	是	string		更新指定groupid下uid对应的信息
user_id	是	string		用户id（由数字、字母、下划线组成），长度限制128B
user_info	否	string		用户资料，长度限制256B
quality_control	否	string	NONE	图片质量控制 **NONE**: 不进行控制 **LOW**:较低的质量要求 **NORMAL**: 一般的质量要求 **HIGH**: 较高的质量要求 **默认 NONE**
liveness_control	否	string	NONE	活体检测控制 **NONE**: 不进行控制 **LOW**:较低的活体要求(高通过率 低攻击拒绝率) **NORMAL**: 一般的活体要求(平衡的攻击拒绝率, 通过率) **HIGH**: 较高的活体要求(高攻击拒绝率 低通过率) **默认NONE**
action_type	否	string	UPDATE	**操作方式 UPDATE**：会使用新图替换该user_id下所有图片，若user_id不存在则会报错。**REPLACE**：当user_id不存在时，则会注册这个user_id的用户 。**默认使用UPDATE**

人脸更新 返回数据参数详情

字段	必选	类型	说明
log_id	是	uint64	请求标识码，随机数，唯一
face_token	是	string	人脸图片的唯一标识
location	是	array	人脸在图片中的位置
+left	是	double	人脸区域离左边界的距离
+top	是	double	人脸区域离上边界的距离
+width	是	double	人脸区域的宽度
+height	是	double	人脸区域的高度
+rotation	是	int64	人脸框相对于竖直方向的顺时针旋转角，[-180,180]

人脸更新 返回示例

```
{
 "face_token": "2fa64a88a9d5118916f9a303782a97d3",
 "location": {
    "left": 117,
    "top": 131,
    "width": 172,
    "height": 170,
    "rotation": 4
 }
}
```

人脸删除

用于从人脸库中删除一个用户。

人脸删除注意事项：

- 删除的内容，包括用户所有图像和身份信息；
- 如果一个uid存在于多个用户组内，将会同时将从各个组中把用户删除
- 如果指定了group_id，则只删除此group下的uid相关信息

```
 userId = "user1"

groupId = "group1"

faceToken = "face_token_23123"

""" 调用人脸删除 """
client.faceDelete(userId, groupId, faceToken);
```

人脸删除 请求参数详情

参数名称	是否必选	类型	说明
log_id	是	uint64	请求标识码，随机数，唯一
user_id	是	string	用户id（由数字、字母、下划线组成），长度限制128B
group_id	是	string	用户组id（由数字、字母、下划线组成），长度限制128B
face_token	是	string	需要删除的人脸图片token，（由数字、字母、下划线组成）长度限制64B

人脸删除 返回数据参数详情

字段	是否必选	类型	说明
log_id	是	uint64	请求唯一标识码，随机数

人脸删除 返回示例

```
 // 删除成功
{
 "error_code": 0,
 "log_id": 73473737,
}
// 删除发生错误
{
 "error_code": 223106,
 "log_id": 1382953199,
 "error_msg": "face is not exist"
}
```

用户信息查询

获取人脸库中某个用户的信息(user_info信息和用户所属的组)。

```
 userId = "user1"

groupId = "group1"

""" 调用用户信息查询 """
client.getUser(userId, groupId);
```

用户信息查询 请求参数详情

参数名称	是否必选	类型	说明
user_id	是	string	用户id（由数字、字母、下划线组成），长度限制128B
group_id	是	string	用户组id（由数字、字母、下划线组成），长度限制128B

用户信息查询 返回数据参数详情

字段	必选	类型	说明
log_id	是	uint64	请求标识码，随机数，唯一
user_list	是	array	查询到的用户列表
+user_info	是	string	用户资料，被查询用户的资料
+group_id	是	string	用户组id，被查询用户的所在组

用户信息查询 返回示例

```
{
 "user_list": [
    {
        "user_info": "user info ...",
        "group_id": "gid1"
    },
    {
        "user_info": "user info2 ...",
        "group_id": "gid2"
    }
 ]
}
```

获取用户人脸列表

用于获取一个用户的全部人脸列表。

```
  userId = "user1"

groupId = "group1"

""" 调用获取用户人脸列表 """
client.faceGetlist(userId, groupId);
```

获取用户人脸列表 请求参数详情

参数名称	是否必选	类型	说明
user_id	是	string	用户id（由数字、字母、下划线组成），长度限制128B
group_id	是	string	用户组id（由数字、字母、下划线组成），长度限制128B

获取用户人脸列表 返回数据参数详情

字段	必选	类型	说明
log_id	是	uint64	请求标识码，随机数，唯一
face_list	是	array	人脸列表
+face_token	是	string	人脸图片的唯一标识
+ctime	是	string	人脸创建时间

获取用户人脸列表 返回示例

```
{
 "face_list": [
    {
        "face_token": "fid1",
        "ctime": "2018-01-01 00:00:00"
    },
    {
        "face_token": "fid2",
        "ctime": "2018-01-01 10:00:00"
    }
 ]
}
```

获取用户列表

用于查询指定用户组中的用户列表。

```
 groupId = "group1"

""" 调用获取用户列表 """
client.getGroupUsers(groupId);

""" 如果有可选参数 """
options = {}
options["start"] = 0
options["length"] = 50

""" 带参数调用获取用户列表 """
client.getGroupUsers(groupId, options)
```

获取用户列表 请求参数详情

参数名称	是否必选	类型	默认值	说明
group_id	是	string		用户组id（由数字、字母、下划线组成），长度限制128B
start	否	string	0	默认值0，起始序号
length	否	string	100	返回数量，默认值100，最大值1000

获取用户列表 返回数据参数详情

字段	必选	类型	说明
user_id_list	是	array	用户ID列表

获取用户列表 返回示例

```
{
 "user_id_list": [
    "uid1",
    "uid2"
 ]
}
```

复制用户

用于将已经存在于人脸库中的用户**复制到一个新的组**。

```
 userId = "user1"

""" 调用复制用户 """
client.userCopy(userId);

""" 如果有可选参数 """
options = {}
options["src_group_id"] = "11111"
options["dst_group_id"] = "222222"

""" 带参数调用复制用户 """
client.userCopy(userId, options)
```

复制用户 请求参数详情

参数名称	是否必选	类型	说明
user_id	是	string	用户id（由数字、字母、下划线组成），长度限制128B
src_group_id	否	string	从指定组里复制信息
dst_group_id	否	string	需要添加用户的组id

复制用户 返回数据参数详情

字段	必选	类型	说明
log_id	是	id	log_id

复制用户 返回示例

```
 // 正确返回值
{
  "error_code": 0,
 "log_id": 3314921889,
}
// 发生错误时返回值
{
 "error_code": 216100,
 "log_id": 3111284097,
 "error_msg": "already add"
}
```

删除用户

用于将用户从某个组中删除。

```
 groupId = "group1"

userId = "user1"

""" 调用删除用户 """
client.deleteUser(groupId, userId);
```

删除用户 请求参数详情

参数名称	是否必选	类型	说明
group_id	是	string	用户组id（由数字、字母、下划线组成），长度限制128B
user_id	是	string	用户id（由数字、字母、下划线组成），长度限制128B

删除用户 返回数据参数详情

字段	是否必选	类型	说明
log_id	是	uint64	请求唯一标识码，随机数

删除用户 返回示例

```
 // 正确返回值
{

 "error_code": 0,
 "log_id": 3314921889,
}
// 发生错误时返回值
{
 "error_code":  223103,
 "log_id": 815967402,
 "error_msg": "user is not exist"
}
```

创建用户组

用于创建一个空的用户组，如果用户组已存在 则返回错误。

```
 groupId = "group1"

""" 调用创建用户组 """
client.groupAdd(groupId);
```

创建用户组 请求参数详情

参数名称	是否必选	类型	说明
group_id	是	string	用户组id（由数字、字母、下划线组成），长度限制128B

创建用户组 返回数据参数详情

字段	必选	类型	说明
log_id	是	uint64	请求标识码，随机数，唯一

创建用户组 返回示例

```
 {

 "error_code": 0,
 "log_id": 3314921889,
}
// 发生错误时返回值
{
 "error_code":  223101,
 "log_id": 815967402,
 "error_msg": " group is already exist"
}
```

删除用户组

删除用户组下所有的用户及人脸，如果组不存在 则返回错误。

```
groupId = "group1"

""" 调用删除用户组 """
client.groupDelete(groupId)
```

删除用户组 请求参数详情

参数名称	是否必选	类型	说明
group_id	是	string	用户组id（由数字、字母、下划线组成），长度限制128B

删除用户组 返回数据参数详情

字段	必选	类型	说明
log_id	是	uint64	请求标识码，随机数，唯一

删除用户组 返回示例

```
// 正确返回值
{
  "error_code":0,
  "log_id": 3314921889,
}
// 发生错误时返回值
{
  "error_code": 223100,
  "log_id": 815967402,
  "error_msg": " group is not exist"
}
```

组列表查询

用于查询用户组的列表。

```
""" 调用组列表查询 """
client.getGroupList();

""" 如果有可选参数 """
options = {}
options["start"] = 0
options["length"] = 50

""" 带参数调用组列表查询 """
client.getGroupList(options)
```

组列表查询 请求参数详情

参数名称	是否必选	类型	默认值	说明
start	否	string	0	默认值0，起始序号
length	否	string	100	返回数量，默认值100，最大值1000

组列表查询 返回数据参数详情

字段	必选	类型	说明
group_id_list	是	array	group

组列表查询 返回示例

```
{
  "group_id_list": [
    "gid1",
    "gid2"
  ]
}
```

身份验证

质量检测（可选）活体检测（可选）公安验证（必选）

```
image = "取决于image_type参数，传入BASE64字符串或URL字符串或FACE_TOKEN字符串"

imageType = "BASE64"

idCardNumber = "110233112299822211"

name = "张三"

""" 调用身份验证 """
client.personVerify(image, imageType, idCardNumber, name);

""" 如果有可选参数 """
options = {}
options["quality_control"] = "NORMAL"
options["liveness_control"] = "LOW"

""" 带参数调用身份验证 """
client.personVerify(image, imageType, idCardNumber, name, options)
```

身份验证 请求参数详情

参数名称	是否必选	类型	默认值	说明
image	是	string		图片信息(**总数据大小应小于10M**)，图片上传方式根据image_type来判断
image_type	是	string		图片类型 **BASE64**:图片的base64值，base64编码后的图片大小不超过2M；**URL**:图片的 URL地址(可能由于网络等原因导致下载图片时间过长)；**FACE_TOKEN**: 人脸图片的唯一标识，调用人脸检测接口时，会为每个人脸图片赋予一个唯一的FACE_TOKEN，同一张图片多次检测得到的FACE_TOKEN是同一个
id_card_number	是	string		身份证号（真实身份证号号码）
name	是	string		utf8，姓名（真实姓名，和身份证号匹配）
quality_control	否	string	NONE	图片质量控制 **NONE**: 不进行控制 **LOW**:较低的质量要求 **NORMAL**: 一般的质量要求 **HIGH**: 较高的质量要求 **默认 NONE**
liveness_control	否	string	NONE	活体检测控制 **NONE**: 不进行控制 **LOW**:较低的活体要求(高通过率 低攻击拒绝率) **NORMAL**: 一般的活体要求(平衡的攻击拒绝率, 通过率) **HIGH**: 较高的活体要求(高攻击拒绝率 低通过率) **默认NONE**

身份验证 返回数据参数详情

参数	必须	类型	说明
log_id	是	uint64	日志id
score	是	float	与公安小图相似度可能性，用于验证生活照与公安小图是否为同一人，有正常分数时为[0~1]，推荐阈值0.8，超过即判断为同一人

身份验证 返回示例

```
{
 "score": 44.3,
}
```

语音校验码接口

此接口主要用于生成随机码，用于视频的语音识别校验使用，以判断视频的即时性，而非事先录制的，提升作弊的难度。

```
""" 调用语音校验码接口 """
client.videoSessioncode();

""" 如果有可选参数 """
options = {}
options["appid"] = "223245"

""" 带参数调用语音校验码接口 """
client.videoSessioncode(, options)
```

语音校验码接口 请求参数详情

参数名称	是否必选	类型	说明
appid	否	string	百度云创建应用时的唯一标识ID

语音校验码接口 返回数据参数详情

字段	必选	类型	说明
session_id	是	string	语音校验码会话id
code	是	string	语音验证码，数字形式，3~6位数字

语音校验码接口 返回示例

```
{
 "err_no": 0,
 "err_msg": "SUCCESS",
 "result": {
   "session_id": "S59faeeebb9111890355690",
   "code": "9940"
 },
 "timestamp": 1509617387,
 "cached": 0,
 "serverlogid": "0587756642"
}
```

在线活体检测

接口能力

- **人脸基础信息**：包括人脸框位置，人脸空间旋转角度，人脸置信度等信息。

- **人脸质量检测**：判断人脸的遮挡、光照、模糊度、完整度等质量信息。可用于判断上传的人脸是否符合标准。
- **基于图片的活体检测**：基于单张图片，判断图片中的人脸是否为二次翻拍（举例：如用户A用手机拍摄了一张包含人脸的图片一，用户B翻拍了图片一得到了图片二，并用图片二伪造成用户A去进行识别操作，这种情况普遍发生在金融开户、实名认证等环节）。此能力可用于H5场景下的一些人脸采集场景中，增加人脸注册的安全性和真实性。

```
result = client.faceverify([
  {
    'image': base64.b64encode(open('1.jpg', 'rb').read()),
    'image_type': 'BASE64',
  },
  {
    'image': base64.b64encode(open('2.jpg', 'rb').read()),
    'image_type': 'BASE64',
  }
])
```

请求参数

参数	是否必选	类型	说明
image	是	string	图片信息(总数据大小应小于10M)，图片上传方式根据image_type来判断； **可以上传同一个用户的1张、3张或8张图片来进行活体判断，注：后端会选择每组照片中的最高分数作为整体分数。**
image_type	是	string	图片类型 **BASE64**:图片的base64值，base64编码后的图片数据，编码后的图片大小不超过2M； **URL**:图片的 URL地址(可能由于网络等原因导致下载图片时间过长)； **FACE_TOKEN**: 人脸图片的唯一标识，调用人脸检测接口时，会为每个人脸图片赋予一个唯一的FACE_TOKEN，同一张图片多次检测得到的FACE_TOKEN是同一个。
face_field	否	string	包括**age,expression,faceshape,gender,glasses,landmark,quality,facetype信息**，逗号分隔，默认只返回face_token、活体数、人脸框、概率和旋转角度

返回参数

参数	类型	是否必须	说明
face_liveness	是	float	活体分数值
thresholds	是	array	由服务端返回最新的阈值数据（随着模型的优化，阈值可能会变化），将此参数与返回的face_liveness进行比较，可以作为活体判断的依据。 frr_1e-4：万分之一误识率的阈值；frr_1e-3：千分之一误识率的阈值；frr_1e-2：百分之一误识率的阈值。误识率越低，准确率越高，相应的拒绝率也越高
face_list	是	array	每张图片的详细信息描述，如果只上传一张图片，则只返回一个结果。
+face_token	是	string	人脸图片的唯一标识
+location	是	array	人脸在图片中的位置
++left	是	double	人脸区域离左边界的距离
++top	是	double	人脸区域离上边界的距离
++width	是	double	人脸区域的宽度

++width	是	double	人脸区域的宽度
++height	是	double	人脸区域的高度
++rotation	是	int64	人脸框相对于竖直方向的顺时针旋转角，[-180,180]
+face_probability	是	double	人脸置信度，范围【0~1】，代表这是一张人脸的概率，0最小、1最大。
+angle	是	array	人脸旋转角度参数
++yaw	是	double	三维旋转之左右旋转角[-90(左), 90(右)]
++pitch	是	double	三维旋转之俯仰角度[-90(上), 90(下)]
++roll	是	double	平面内旋转角[-180(逆时针), 180(顺时针)]
+age	否	double	年龄 ，当**face_field包含age时返回**
+expression	否	array	表情，当 **face_field包含expression时返回**
++type	否	string	**none**:不笑；**smile**:微笑；**laugh**:大笑
++probability	否	double	表情置信度，范围【0~1】，0最小、1最大。
+face_shape	否	array	脸型，当**face_field包含faceshape时返回**
++type	否	double	**square**: 正方形 **triangle**:三角形 **oval**: 椭圆 **heart**: 心形 **round**: 圆形
++probability	否	double	置信度，范围【0~1】，代表这是人脸形状判断正确的概率，0最小、1最大。
+gender	否	array	性别，**face_field包含gender时返回**
++type	否	string	male:**男性** female:**女性**
++probability	否	double	性别置信度，范围【0~1】，0代表概率最小、1代表最大。
+glasses	否	array	是否带眼镜，**face_field包含glasses时返回**
++type	否	string	**none**:无眼镜，**common**:普通眼镜，**sun**:墨镜
++probability	否	double	眼镜置信度，范围【0~1】，0代表概率最小、1代表最大。
+landmark	否	array	4个关键点位置，左眼中心、右眼中心、鼻尖、嘴中心。**face_field包含landmark时返回**
+landmark72	否	array	72个特征点位置 **face_field包含landmark时返回**
+quality	否	array	人脸质量信息。**face_field包含quality时返回**
++occlusion	否	array	人脸各部分遮挡的概率，范围[0~1]，0表示完整，1表示不完整
+++left_eye	否	double	左眼遮挡比例，[0-1] ， 1表示完全遮挡
+++right_eye	否	double	右眼遮挡比例，[0-1] ， 1表示完全遮挡

+++nose	否	double	鼻子遮挡比例，[0-1] ， 1表示完全遮挡
+++mouth	否	double	嘴巴遮挡比例，[0-1] ， 1表示完全遮挡
+++left_cheek	否	double	左脸颊遮挡比例，[0-1] ， 1表示完全遮挡
+++right_cheek	否	double	右脸颊遮挡比例，[0-1] ， 1表示完全遮挡
+++chin	否	double	下巴遮挡比例， ，[0-1] ， 1表示完全遮挡
++blur	否	double	人脸模糊程度，范围[0~1]，0表示清晰，1表示模糊
++illumination	否	double	取值范围在[0~255], 表示脸部区域的光照程度 越大表示光照越好
++completeness	否	int64	人脸完整度，0或1, 0为人脸溢出图像边界，1为人脸都在图像边界内
+parsing_info	否	string	人脸分层结果 结果数据是使用gzip压缩后再base64编码 使用前需base64解码后再解压缩 **原数据格式为string 形如0,0,0,0,0,1,1,1,1,1,1,2,2,2,2,2,2,2,2,2,…**

返回示例


```
{
"thresholds": {
    "frr_1e-4": 0.05,  //万分之一误拒率的阈值
    "frr_1e-3": 0.3,   //千分之一误拒率的阈值
    "frr_1e-2": 0.9    //百分之一误拒率的阈值
},
"face_liveness": 0.05532243927,
"face_list": [
    {
        "face_token": "df46f7c7db4aa09a093c26fb8d1a8d44",
        "location": {
            "left": 328.9026489,
            "top": 97.16340637,
            "width": 162,
            "height": 154,
            "rotation": 32
        },
        "face_probability": 1,
        "angle": {
            "yaw": 10.16196251,
            "pitch": 2.244354248,
            "roll": 33.82199097
        },
        "liveness": {
            "faceliveness": 0.004187555984,
            "livemapscore": 0.04492170034
        },
        "age": 23
    },
    {
        "face_token": "901d2c64274fccd687d311a6e6110a01",
        "location": {
            "left": 411.4876404,
```

```
            "top": 166.3593445,
            "width": 329,
            "height": 308,
            "rotation": 45
        },
        "face_probability": 0.9194830656,
        "angle": {
            "yaw": -1.716423035,
            "pitch": 7.344647408,
            "roll": 45.79914856
        },
        "liveness": {
            "faceliveness": 0.0001665892196,
            "livemapscore": 0.001787073661
        },
        "age": 23
    },
    {
        "face_token": "7d57e36981c48b4946eb97c8d838b02a",
        "location": {
            "left": 161.4559937,
            "top": 199.8726501,
            "width": 218,
            "height": 201,
            "rotation": -1
        },
        "face_probability": 1,
        "angle": {
            "yaw": -8.187754631,
            "pitch": 6.973727226,
            "roll": -1.25429821
        },
        "liveness": {
            "faceliveness": 0.02942637168,
            "livemapscore": 0.05532243927
        },
        "age": 23
    }
]
}
```


人脸对比

接口能力

- **两张人脸图片相似度对比**：比对两张图片中人脸的相似度，并返回相似度分值；
- **多种图片类型**：支持**生活照**、**证件照**、**身份证芯片照**、**带网纹照**四种类型的人脸对比；
- **活体检测**：基于图片中的破绽分析，判断其中的人脸是否为**二次翻拍**（举例：如用户A用手机拍摄了一张包含人脸的图片一，用户B翻拍了图片一得到了图片二，并用图片二伪造成用户A去进行识别操作，这种情况普遍发生在金融开户、实名认证等环节。）；
- **质量检测**：返回模糊、光照等质量检测信息，用于辅助判断图片是否符合识别要求；

业务应用

用于比对多张图片中的人脸相似度并返回两两比对的得分，可用于判断两张脸是否是同一人的可能性大小。

典型应用场景：如**人证合一验证**，**用户认证**等，可与您现有的人脸库进行比对验证。

```
result = client.match([
    {
        'image': base64.b64encode(open('1.jpg', 'rb').read()).decode(),
        'image_type': 'BASE64',
    },
    {
        'image': base64.b64encode(open('2.jpg', 'rb').read()).decode(),
        'image_type': 'BASE64',
    }
])
```

请求参数

参数	必选	类型	说明
image	是	string	图片信息(**总数据大小应小于10M**)，图片上传方式根据image_type来判断。 **两张图片通过json格式上传，格式参考表格下方示例**
image_type	是	string	图片类型 **BASE64**:图片的base64值，base64编码后的图片数据，编码后的图片大小不超过2M；**URL**:图片的 URL地址(可能由于网络等原因导致下载图片时间过长)；**FACE_TOKEN**: 人脸图片的唯一标识，调用人脸检测接口时，会为每个人脸图片赋予一个唯一的FACE_TOKEN，同一张图片多次检测得到的FACE_TOKEN是同一个。
face_type	否	string	人脸的类型 **LIVE**表示生活照：通常为手机、相机拍摄的人像图片、或从网络获取的人像图片等，**IDCARD**表示身份证芯片照：二代身份证内置芯片中的人像照片， **WATERMARK**表示带水印证件照：一般为带水印的小图，如公安网小图 **CERT**表示证件照片：如拍摄的身份证、工卡、护照、学生证等证件图片 默认LIVE
quality_control	否	string	图片质量控制 **NONE**: 不进行控制 **LOW**:较低的质量要求 **NORMAL**: 一般的质量要求 **HIGH**: 较高的质量要求 **默认 NONE**
liveness_control	否	string	活体检测控制 **NONE**: 不进行控制 **LOW**:较低的活体要求(高通过率 低攻击拒绝率) **NORMAL**: 一般的活体要求(平衡的攻击拒绝率, 通过率) **HIGH**: 较高的活体要求(高攻击拒绝率 低通过率) **默认NONE**

返回参数

参数名	必选	类型	说明
score	是	float	人脸相似度得分
face_list	是	array	人脸信息列表
+face_token	是	string	人脸的唯一标志

返回示例

```
{
  "score": 44.3,
  "face_list": [  //返回的顺序与传入的顺序保持一致
      {
         "face_token": "fid1"
      },
      {
         "face_token": "fid2"
      }
  ]
}
```

人脸实名认证V4

能力介绍

1. 业务能力

- **质量检测（可选）**：判断图片中是否包含人脸，以及人脸在姿态、遮挡、模糊、光照等方面是否符合识别条件。
- **活体检测（可选）**：基于图片中的破绽分析，判断其中的人脸是否为**二次翻拍**（举例：如用户A用手机拍摄了一张包含人脸的图片一，用户B翻拍了图片一得到了图片二，并用图片二伪造成用户A去进行识别操作，这种情况普遍发生在金融开户、实名认证等环节）。
- **图片加密及风控（可选）**：当配合增强级采集SDK、增强级安全加固采集SDK、金融级采集SDK、金融级安全加固采集SDK版本使用，对采集SDK输出的加密图片进行解密（加密传输可以有效避免第三方非法黑产绕过APP模拟请求攻击云端接口的行为，Eg：脚本攻击等）；以及结合百度安全实验室大数据风控能力，对采集SDK的发起端设备进行风控识别，辨别是否为风险设备，Eg：ROM注入、视频劫持等；
- **人脸实名认证（必选）**：基于姓名和身份证号，调取公安权威数据源人脸图，将当前获取的人脸图片，与公安数据源人脸图进行对比，得出比对分数，并基于此进行业务判断是否为同一人。由于公安数据源人脸图具有最权威的身份证明作用，故对用户本人的验证结果可信度也最为合理。

2. 业务逻辑

- 上述能力，人脸实名认证能力为必选能力，质量检测、活体检测、图片加密及风控为可选能力，验证顺序为`人脸质量检测->活体检测->人脸实名认证`。
- **如选择了质量检测和活体检测能力，则有任意一个条件不通过，整个请求流程终止，并返回错误码，描述具体的不符合信息**。
- 基于此顺序串行验证逻辑，可以避免大量不符合条件的请求流转到人脸实名认证，节约您的成本。

3. 推荐阈值

- 此接口使用的对比算法，针对带水纹证件照采用了专项的模型处理，可保证水纹信息的影响降到尽可能低。
- 如比对成功，最终返回的有效数据为一个**对比分值**，在0~100之间，您可以设定具体的阈值来判断是否验证通过。
- **人证相似度的推荐阈值为80，对应的误识率为万分之一**。

```
 ###### 必填参数
id_card_number = ''
name = ''
image = ''

###### 选填参数
options = {}
options["xxx"] = "xxx"

###### 调用接口
client.faceMingJingVerify(id_card_number, name, image, options)
```

请求参数

参数	必选	类型	说明
app	否	string	APP端类型，配合采集SDK使用时须传入 **ios**：iOS端采集SDK **android**：安卓端采集SDK
sec_level	否	string	SDK安全级别，配合采集SDK使用时须传入，默认common **common**：配合4.1/4.1.5版本采集SDK，人脸图片未进行加密处理 **lite**: 配合5.2版本SDK
skey	否	string	使用5.2版本SDK请求时必填 **skey**：从SDK获取的密钥信息skey
x_device_id	否	string	使用5.2版本SDK请求时必填 **deviceId**：从SDK 获取的密钥信息deviceId
data	否	string	使用5.2版本SDK请求时必填， SDK输出的加密数据
id_card_number	是	string	身份证件号
name	是	string	姓名(需要是 utf8 编码)
liveness_control	否	string	活体控制参数 **NONE**: 不进行控制 **LOW**:较低的活体要求(高通过率 低攻击拒绝率) **NORMAL**: 一般的活体要求(平衡的攻击拒绝率, 通过率) **HIGH**: 较高的活体要求(高攻击拒绝率 低通过率) **默认为NONE**
spoofing_control	否	string	合成图控制参数 **NONE**: 不进行控制 **LOW**:较低的合成图阈值数值，由于合成图判定逻辑为大于阈值视为合成图攻击，该项代表**低通过率、高攻击拒绝率** **NORMAL**: 一般的合成图阈值数值，由于合成图判定逻辑为大于阈值视为合成图攻击，该项代表**平衡的攻击拒绝率, 通过率** **HIGH**: 较高的合成图阈值数值，由于合成图判定逻辑为大于阈值视为合成图攻击，该项代表**高通过率、低攻击拒绝率**) **默认为NONE**
quality_control	否	string	质量控制参数 **NONE**: 不进行控制 **LOW**:较低的质量要求 **NORMAL**: 一般的质量要求 **HIGH**: 较高的质量要求 **默认为NONE**
image	是	string	图片信息(数据大小应小于10M 分辨率应小于1920*1080)，**5.2版本SDK请求时已包含在加密数据data中，无需额外传入**
image_type	否	string	图片类型 **BASE64**：图片的base64值 **URL**：图片的 URL **FACE_TOKEN**：人脸标识 **默认 BASE64**

返回参数

参数	类型	说明
log_id	number	调用的日志id
result	jsonObject	认证返回的结果
+score	float	与公安数据源人脸图相似度可能性，用于验证生活照与公安数据源人脸图是否为同一人，有正常分数时为[0~100]，推荐阈值80，超过即判断为同一人
+verify_status	number	认证状态，取值如下： 0 ： 正常 1 ： 身份证号与姓名不匹配或该身份证号不存在 2 ： 公安网图片不存在或质量过低
dec_image	string	对SDK传入的加密图片进行解密。 仅APP场景且进行了图片加密时，此参数返回解密后的人脸图片信息
risk_level	string	判断设备是否发生过风险行为来判断风险级别，取值（数值由高到低）： 1 – 高危 2 – 嫌疑 3 – 普通 4 – 正常
risk_tag	string	风险标签，若判断为有风险，则会有风险标签json 数组告知风险类型 例如：general_inject

返回示例

```
{
    "log_id": 1370579072568000512,
    "result": {
        "score": 40.884,
        "verify_status": 0
    },
    "dec_image": "/9j/4AAQSkZJRgABAgAAAQABAAD",
    "risk_level": "3",
    "risk_tag": [
        "若判断为有风险，则会有风险标签json 数组告知风险类型，如：general_inject"
    ]
}
```


人脸比对V4

能力介绍

1. 接口能力

- **两张人脸图片相似度对比**：对比两张图片中人脸的相似度，并返回相似度分值；
- **多种图片类型**：支持**生活照**、**证件照**、**身份证芯片照**、**带网纹照**四种类型的人脸对比；
- **活体检测控制**：基于图片中的破绽分析，判断其中的人脸是否为**二次翻拍**（举例：如用户A用手机拍摄了一张包含人脸的图片一，用户B翻拍了图片一得到了图片二，并用图片二伪造成用户A去进行识别操作，这种情况普遍发生在金融开户、实名认证等环节。）；
- **质量检测控制**：分析图片的中人脸的模糊度、角度、光照强度等特征，判断图片质量；
- **图片加密及风控：**配合增强级、金融级采集SDK使用
    - 对采集SDK输出的加密图片进行解密（加密传输可以有效避免第三方非法黑产**绕过APP模拟请求攻击云端接口的行为**，Eg：脚本攻击等）；
    - 以及结合百度安全实验室大数据风控能力，对采集SDK的发起端设备进行风控识别，辨别是否为**风险设备**，Eg：ROM注入、视频劫持等；

2. 业务应用

- 用于对比多张图片中的人脸相似度并返回两两对比的得分，可用于判断两张脸是否是同一人的可能性大小。
- 典型应用场景：如**人证合一验证**，**用户认证**等，可与您现有的人脸库进行对比验证。

```
###### 必填参数
image = ''
imageType = ''
registerImage = ''
registerImageType = ''

###### 选填参数
options = {}
options["xxx"] = "xxx"

###### 调用接口
client.faceMingJingMatch(image, imageType, registerImage, registerImageType, options)
```

请求参数

参数	必选	类型	说明
app	否	string	APP类型，配合采集SDK使用时须传入 **ios**：iOS端采集SDK **android**：安卓端采集SDK
sec_level	否	string	SDK安全级别，配合采集SDK使用时须传入，默认common **common**：配合4.1/4.1.5版本SDK使用，人脸图片未进行加密处理 **lite**：配合5.2版本SDK使用
skey	否	string	使用5.2版本SDK请求时必填 从SDK获取的密钥信息
x_device_id	否	string	使用5.2版本SDK请求时必填 从SDK获取的密钥信息
data	否	string	使用5.2版本SDK请求时必填 SDK输出的加密数据
image	是	string	图片信息(数据大小应小于10M 分辨率应小于1920*1080)，***5.2版本SDK请求时已包含在加密数据data中，无需额外传入**
image_type	是	string	图片类型 **BASE64**：图片的base64值 **URL**：图片的URL **FACE_TOKEN**：人脸标识 默认 BASE64
face_type	否	string	人脸的类型 **LIVE**：表示生活照：通常为手机、相机拍摄的人像图片、或从网络获取的人像图片等 **IDCARD**：表示身份证芯片照：二代身份证内置芯片中的人像照片 **WATERMARK**：表示带水印证件照：一般为带水印的小图，如公安网小图 **CERT**：表示证件照片：如拍摄的身份证、工卡、护照、学生证等证件图片 **INFRARED** 表示红外照片：使用红外相机拍摄的照片 默认LIVE
quality_control	否	string	质量控制 **NONE**: 不进行控制 **LOW**:较低的质量要求 **NORMAL**: 一般的质量要求 **HIGH**: 较高的质量要求 **默认NONE**
liveness_control	否	string	活体控制 **NONE**: 不进行控制 **LOW**:较低的活体要求(高通过率 低攻击拒绝率) **NORMAL**: 一般的活体要求(平衡的攻击拒绝率, 通过率) **HIGH**: 较高的活体要求(高攻击拒绝率 低通过率) **默认NONE**
registerimage	是	string	图片信息(总数据大小应小于10M)，图片上传方式根据imagetype来判断。本图片特指客户服务器上传图片，非加密图片Base64值
registerimagetype	是	string	图片类型 **BASE64**：图片的base64值 **URL**：图片的 URL **FACE_TOKEN**：人脸标识 默认 BASE64
registerfacetype	否	string	人脸的类型
registerqualitycontrol	否	string	图片质量控制 **NONE**: 不进行控制 **LOW**:较低的质量要求 **NORMAL**: 一般的质量要求 **HIGH**: 较高的质量要求 **默认 NONE** 若图片质量不满足要求，则返回结果中会提示质量检测失败
registerlivenesscontrol	否	string	活体检测控制 **NONE**: 不进行控制 **LOW**:较低的活体要求(高通过率 低攻击拒绝率) **NORMAL**: 一般的活体要求(平衡的攻击拒绝率, 通过率) **HIGH**: 较高的活体要求(高攻击拒绝率 低通过率) **默认 NONE** 若活体检测结果不满足要求，则返回结果中会提示活体检测失败
facesorttype	否	int	人脸检测排序类型 0:代表检测出的人脸按照人脸面积从大到小排列 1:代表检测出的人脸按照距离图片中心从近到远排列 默认为0

返回参数

参数名	类型	说明
log_id	number	调用的日志id
result	jsonObject	认证返回的结果
+ score	number	人脸相似度得分，推荐阈值80分
+ face_list	jsonArray	人脸信息列表
++ face_token	string	人脸标志
dec_image	string	APP场景传入加密图片时，该项返回解密后的图片
risk_level	string	风控返回参数，只有在 risk_identify 为 true 时才返回，判断设备是否发生过风险行为来判断风险级别，取值（数值有高到低）： 1 – 高危 2 – 嫌疑 3 – 普通 4 – 正常
risk_tag	jsonArray	风控返回参数，只有在 risk_identify 为 true 时才返回，风险标签，若判断为有风险，则会有风险标签 json 数组告知风险类型
risk_warn_code	number	风控返回参数，只有在 risk_identify 为 true 时才返回，只有在风控服务异常的时候才返回这个字段

返回示例


```
{
  "log_id": 1370585066551377920,
  "result": {
    "score": 99.06919861,
    "face_list": [
      {
        "face_token": "549f9f1d1c7ec8c86931540b1939e8ed"
      },
      {
        "face_token": "1a319460ef89e8d27fb59062a28dbad7"
      }
    ]
  },
  "dec_image": "/9j/4AAQSkZJRgABAQAAAQABAAD",
  "risk_level": "3",
  "risk_tag": [
    "空"
  ]
}
```


在线图片活体V4

能力介绍

1. 接口能力

- **人脸基础信息**：包括人脸框位置，人脸空间旋转角度，人脸置信度等信息。
- **人脸质量检测**：判断人脸的遮挡、光照、模糊度、完整度等质量信息。可用于判断上传的人脸是否符合标准。
- **基于图片的活体检测**：基于单张图片，判断图片中的人脸是否为二次翻拍（举例：如用户A用手机拍摄了一张包含人脸的图片一，用户B翻拍了图片一得到了图片二，并用图片二伪造成用户A去进行识别操作，这种情况普遍发生在金融开户、实名

认证等环节）以及是否为合成图攻击。此能力可用于H5场景下的一些人脸采集场景中，增加人脸注册的安全性和真实性。

- **图片加密及风控**：配合采集SDK5.0版本使用，对采集SDK输出的加密图片进行解密（加密传输可以有效避免第三方非法黑产绕过APP模拟请求攻击云端接口的行为，Eg：脚本攻击等）；以及结合百度安全实验室大数据风控能力，对采集SDK的发起端设备进行风控识别，辨别是否为风险设备，Eg：ROM注入、视频劫持等；

```
###### 必填参数
sdkVersion = ""

###### 选填参数
options = {}
options["xxx"] = "xxx"

###### 调用接口
client.onlinePictureLiveV4(sdkVersion, options)
```

请求参数

参数	是否必选	类型	说明
sdk_version	是	string	1 ：非加密图片，适用于4.1/4.1.5版本SDK、H5场景或纯服务端场景 4：适用于5.2版本SDK
face_field	否	string	包括**age,expression,face_shape,gender,glasses,landmark,quality,face_type,spoofing**信息，逗号分隔，默认只返回face_token、活体数、人脸框、概率和旋转角度
option	否	string	场景信息，程序会视不同的场景选用相对应的模型。当前支持的场景有COMMON(通用场景)，GATE(闸机场景)，FINANCE(金融场景)，LOGISTICS(物流场景)，INTERNET(泛互联网场景), **默认使用COMMON** 。注意：如果请求参数中存在多个option值时，则取第一个option的值，同时除通用场景外的其他场景需要联系工作人员对您所使用的appid进行配置
app	否	string	端类型 ios / android **5.2版本SDK必传该项**
s_key	否	string	端上提供的用于解密图片的skey **5.2版本SDK必传该项**
device_id	否	string	端上提供的用于解密图片的deviceId **5.2版本SDK必传该项**
data	否	string	端上提供的加密后的图片数组 **5.2版本SDK必传该项**
image_list	否	array	图片BASE64数组 4.1/4.1.5版本SDK、H5场景或纯服务端场景必填

返回参数

参数	是否必须	类型	说明
log_id	是	string	日志id
error_code	是	int	错误码状态，若为0则表示认证成功
error_msg	是	string	错误码说明，若为 success 则表示认证成功
result	是	object	活体结果
+face_liveness	是	float	活体分数值
+thresh			由服务端返回最新的阈值数据（随着模型的优化，阈值可能会变化），将此参数与返回的face_liveness进行比较，可以作为活体判断的依据。frr_1e-4：万分之一误识率的阈值； frr_1e-

+thresholds	是	array	face_liveness进行比较，可以作为活体判断的依据。 frr_1e-4：万分之一误识率的阈值； frr_1e-3：千分之一误识率的阈值； frr_1e-2：百分之一误识率的阈值。 误识率越低，准确率越高，相应的拒绝率也越高
+face_list	是	array	每张图片的详细信息描述，如果只上传一张图片，则只返回一个结果。
++face_token	是	string	人脸图片的唯一标识
++location	是	array	人脸在图片中的位置
+++left	是	double	人脸区域离左边界的距离
+++top	是	double	人脸区域离上边界的距离
+++width	是	double	人脸区域的宽度
+++height	是	double	人脸区域的高度
+++rotation	是	int64	人脸框相对于竖直方向的顺时针旋转角，[-180,180]
++face_probability	是	double	人脸置信度，范围【0~1】，代表这是一张人脸的概率，0最小、1最大。
++angel	是	array	人脸旋转角度参数
+++yaw	是	double	三维旋转之左右旋转角[-90(左), 90(右)]
+++pitch	是	double	三维旋转之俯仰角度[-90(上), 90(下)]
+++roll	是	double	平面内旋转角[-180(逆时针), 180(顺时针)]
++age	否	double	年龄 ，当**face_field包含age时返回**
++expression	否	array	表情，当 **face_field包含expression时返回**
+++type	否	string	**none**:不笑；**smile**:微笑；**laugh**:大笑
+++probability	否	double	表情置信度，范围【0~1】，0最小、1最大。
++face_shape	否	array	脸型，当**face_field包含face_shape时返回**
+++type	否	double	**square**: 正方形 **triangle**:三角形 **oval**: 椭圆 **heart**: 心形 **round**: 圆形
+++probability	否	double	置信度，范围【0~1】，代表这是人脸形状判断正确的概率，0最小、1最大。
++gender	否	array	性别，**face_field包含gender时返回**
+++type	否	string	male:**男性** female:**女性**
+++probability	否	double	性别置信度，范围【0~1】，0代表概率最小、1代表最大。

++glasses	否	array	是否带眼镜，**face_field包含glasses时返回**
++type	否	string	**none**:无眼镜，**common**:普通眼镜，**sun**:墨镜
+++probability	否	double	眼镜置信度，范围【0~1】，0代表概率最小、1代表最大。
++face_type	否	array	真实人脸/卡通人脸 **face_field包含face_type时返回**
+++type	否	string	**human**: 真实人脸 **cartoon**: 卡通人脸
+++probability	否	double	人脸类型判断正确的置信度，范围【0~1】，0代表概率最小、1代表最大。
++landmark	否	array	4个关键点位置，左眼中心、右眼中心、鼻尖、嘴中心。**face_field包含landmark时返回**
++landmark72	否	array	72个特征点位置 **face_field包含landmark时返回**
++quality	否	array	人脸质量信息。**face_field包含quality时返回**
+++occlusion	否	array	人脸各部分遮挡的概率，范围[0~1]，0表示完整，1表示不完整
++++left_eye	否	double	左眼遮挡比例，[0-1] ， 1表示完全遮挡
++++right_eye	否	double	右眼遮挡比例，[0-1] ， 1表示完全遮挡
++++nose	否	double	鼻子遮挡比例，[0-1] ， 1表示完全遮挡
++++mouth	否	double	嘴巴遮挡比例，[0-1] ， 1表示完全遮挡
++++left_cheek	否	double	左脸颊遮挡比例，[0-1] ， 1表示完全遮挡
++++right_cheek	否	double	右脸颊遮挡比例，[0-1] ， 1表示完全遮挡
++++chin	否	double	下巴遮挡比例，，[0-1] ， 1表示完全遮挡
+++blur	否	double	人脸模糊程度，范围[0~1]，0表示清晰，1表示模糊
+++illumination	否	double	取值范围在[0~255], 表示脸部区域的光照程度 越大表示光照越好
+++completeness	否	int64	人脸完整度，0或1, 0为人脸溢出图像边界，1为人脸都在图像边界内
++spoofing	否	double	合成图打分 判断图片是否为合成图 face_field包含时返回spoofing
risk_lev	否	string	判断设备是否发生过风险行为来判断风险级别，取值（数值有高到低）：1 – 高危 2 – 嫌疑 3 – 普

el	否	string	通 4 – 正常
risk_tag	否	string	风险标签，若判断为有风险，则会有风险标签 json 数组告知风险类型
dec_image	否	array	非安全加固增强级采集sdk、非安全加固金融级采集sdk、安全加固增强级采集sdk、安全加固金融级采集sdk会返回解密后的原图

返回示例


```
{
  "result": {
    "thresholds": {
      "frr_1e-3": 0.3,
      "frr_1e-2": 0.9,
      "frr_1e-4": 0.05
    },
    "face_liveness": 0.1976952702,
    "face_list": [
      {
        "liveness": {
          "livemapscore": 0.1976952702
        },
        "angle": {
          "roll": 0.31,
          "pitch": -2.07,
          "yaw": 0.85
        },
        "face_token": "abd1b4ce743099336cfed40193ff4944",
        "location": {
          "top": 161.74,
          "left": 79.04,
          "rotation": 1,
          "width": 318,
          "height": 333
        },
        "face_probability": 1
      },
      {
        "liveness": {
          "livemapscore": 0.1976952702
        },
        "angle": {
          "roll": 0.31,
          "pitch": -2.07,
          "yaw": 0.85
        },
        "face_token": "abd1b4ce743099336cfed40193ff4944",
        "location": {
          "top": 161.74,
          "left": 79.04,
          "rotation": 1,
          "width": 318,
          "height": 333
        },
        "face_probability": 1
      },
      {
        "liveness": {
          "livemapscore": 0.1976952702
        },
        "angle": {
          "roll": 0.31,
```

```
            "pitch": -2.07,
            "yaw": 0.85
        },
        "face_token": "abd1b4ce743099336cfed40193ff4944",
        "location": {
            "top": 161.74,
            "left": 79.04,
            "rotation": 1,
            "width": 318,
            "height": 333
        },
        "face_probability": 1
    },
    {
        "liveness": {
            "livemapscore": 0.1976952702
        },
        "angle": {
            "roll": 0.31,
            "pitch": -2.07,
            "yaw": 0.85
        },
        "face_token": "abd1b4ce743099336cfed40193ff4944",
        "location": {
            "top": 161.74,
            "left": 79.04,
            "rotation": 1,
            "width": 318,
            "height": 333
        },
        "face_probability": 1
    },
    {
        "liveness": {
            "livemapscore": 0.1976952702
        },
        "angle": {
            "roll": 0.31,
            "pitch": -2.07,
            "yaw": 0.85
        },
        "face_token": "abd1b4ce743099336cfed40193ff4944",
        "location": {
            "top": 161.74,
            "left": 79.04,
            "rotation": 1,
            "width": 318,
            "height": 333
        },
        "face_probability": 1
    },
    {
        "liveness": {
            "livemapscore": 0.1976952702
        },
        "angle": {
            "roll": 0.31,
            "pitch": -2.07,
            "yaw": 0.85
        },
        "face_token": "abd1b4ce743099336cfed40193ff4944",
        "location": {
            "top": 161.74,
```

```
                "top": 161.74,
                "left": 79.04,
                "rotation": 1,
                "width": 318,
                "height": 333
            },
            "face_probability": 1
        },
        {
            "liveness": {
                "livemapscore": 0.1976952702
            },
            "angle": {
                "roll": 0.31,
                "pitch": -2.07,
                "yaw": 0.85
            },
            "face_token": "abd1b4ce743099336cfed40193ff4944",
            "location": {
                "top": 161.74,
                "left": 79.04,
                "rotation": 1,
                "width": 318,
                "height": 333
            },
            "face_probability": 1
        },
        {
            "liveness": {
                "livemapscore": 0.1976952702
            },
            "angle": {
                "roll": 0.31,
                "pitch": -2.07,
                "yaw": 0.85
            },
            "face_token": "abd1b4ce743099336cfed40193ff4944",
            "location": {
                "top": 161.74,
                "left": 79.04,
                "rotation": 1,
                "width": 318,
                "height": 333
            },
            "face_probability": 1
        }
    ]
},
"risk_level": "3",
"log_id": 1423545654699779516,
"risk_tag": [
    "空"
],
"dec_image": [
    "/9j/4AAQ..."
]
}
```


错误信息

错误返回格式

若请求错误，服务器将返回的JSON文本包含以下参数：

- **error_code**：错误码。
- **error_msg**：错误描述信息，帮助理解和解决发生的错误。

错误码

服务端返回的错误码

错误码	错误信息	描述
1	Unknown error	服务器内部错误，请再次请求， 如果持续出现此类错误，请通过QQ群（860337848）或提交工单联系技术支持团队。
2	Service temporarily unavailable	服务暂不可用，请再次请求， 如果持续出现此类错误，请通过QQ群（860337848）或提交工单联系技术支持团队。
3	Unsupported openapi method	调用的API不存在，请检查请求URL后重新尝试，一般为URL中有非英文字符，如“-”，可手动输入重试
4	Open api request limit reached	集群超限额，请再次请求， 如果持续出现此类错误，请通过QQ群（860337848）或提交工单联系技术支持团队。
6	No permission to access data	无权限访问该用户数据，创建应用时未勾选相关接口
13	Get service token failed	获取token失败
14	IAM Certification failed	IAM 鉴权失败
15	app not exsits or create failed	应用不存在或者创建失败
17	Open api daily request limit reached	每天请求量超限额，可通过QQ群（860337848）联系群管、提交工单提升限额
18	Open api qps request limit reached	QPS超限额，可通过QQ群（860337848）联系群管、提交工单提升限额
19	Open api total request limit reached	请求总量超限额，可通过QQ群（860337848）联系群管、提交工单提升限额
100	Invalid parameter	无效的access_token参数，请检查后重新尝试
110	Access token invalid or no longer valid	access_token无效
111	Access token expired	access token过期
222001	param[] is null	必要参数未传入
222002	param[start] format error	参数格式错误
222003	param[length] format error	参数格式错误
222004	param[op_app_id_list] format error	参数格式错误
222005	param[group_id_list] format error	参数格式错误
222006	group_id format error	参数格式错误
222007	uid format error	参数格式错误
222008	face_id format error	参数格式错误

222008	face_id format error	参数格式错误
222009	quality_conf format error	参数格式错误
222010	user_info format error	参数格式错误
222011	param[uid_list] format error	参数格式错误
222012	param[op_app_id] format error	参数格式错误
222013	param[image] format error	参数格式错误
222014	param[app_id] format error	参数格式错误
222015	param[image_type] format error	参数格式错误
222016	param[max_face_num] format error	参数格式错误
222017	param[face_field] format error	参数格式错误
222018	param[user_id] format error	参数格式错误
222019	param[quality_control] format error	参数格式错误
222020	param[liveness_control] format error	参数格式错误
222021	param[max_user_num] format error	参数格式错误
222022	param[id_card_number] format error	参数格式错误
222023	param[name] format error	参数格式错误
222024	param[face_type] format error	参数格式错误
222025	param[face_token] format error	参数格式错误
222026	param[max_star_num] format error	参数格式错误
222201	network not available	服务端请求失败
222202	pic not has face	图片中没有人脸
222203	image check fail	无法解析人脸
222204	image_url_download_fail	从图片的url下载图片失败
222205	network not availablel	服务端请求失败
222206	rtse service return fail	服务端请求失败
222207	match user is not found	未找到匹配的用户
222208	the number of image	图片的数量错误

222208	is incorrect	图片的数量错误
222209	face token not exist	face token不存在
222300	add face fail	人脸图片添加失败
222301	get face fail	获取人脸图片失败
222302	system error	服务端请求失败
222303	get face fail	获取人脸图片失败
223100	group is not exist	操作的用户组不存在
223101	group is already exist	该用户组已存在
223102	user is already exist	该用户已存在
223103	user is not exist	找不到该用户
223104	group_list is too large	group_list包含组 数量过多
223105	face is already exist	该人脸已存在
223106	face is not exist	该人脸不存在
223110	uid_list is too large	uid_list包含数量过多
223111	dst group is not exist	目标用户组不存在
223112	quality_conf format error	quality_conf格式不正确
223113	face is covered	人脸有被遮挡
223114	face is fuzzy	人脸模糊
223115	face light is not good	人脸光照不好
223116	incomplete face	人脸不完整
223117	app_list is too large	app_list包含app数量 过多
223118	quality control error	质量控制项错误
223119	liveness control item error	活体控制项错误
223120	liveness check fail	活体检测未通过
223121	left eye is occlusion	质量检测未通过 左眼 遮挡程度过高
223122	right eye is occlusion	质量检测未通过 右眼 遮挡程度过高
223123	left cheek is occlusion	质量检测未通过 左脸 遮挡程度过高
223124	right cheek is occlusion	质量检测未通过 右脸 遮挡程度过高
223125	chin contour is occlusion	质量检测未通过 下巴遮挡程度过高
223126	nose is occlusion	质量检测未通过 鼻子遮挡程度过高
223127	mouth is occlusion	质量检测未通过 嘴巴 遮挡程度过高
	police picture is none or	公安网图片不存在或

222350	low quality	质量过低
222351	id number and name not match or id number not exist	身份证号与姓名不匹配或该身份证号不存在
222352	name format error	身份证名字格式错误
222353	id number format error	身份证号码格式错误
222354	id number not exist	公安库里不存在此身份证号
222355	police picture not exist	身份证号码正确，公安库里没有对应的照片
222360	invalid name or id number	身份证号码或名字非法（公安网校验不通过）
222901	system busy	系统繁忙
222902	system busy	系统繁忙
222903	system busy	系统繁忙
222904	system busy	系统繁忙
222905	system busy	系统繁忙
222906	system busy	系统繁忙
222907	system busy	系统繁忙
222908	system busy	系统繁忙
222909	system busy	系统繁忙
222910	system busy	系统繁忙
222911	system busy	系统繁忙
222912	system busy	系统繁忙
222913	system busy	系统繁忙
222914	system busy	系统繁忙
222915	system busy	系统繁忙
222916	system busy	系统繁忙
222361	system busy	系统繁忙

Java-SDK

简介

Hi，您好，欢迎使用百度人脸识别服务。

本文档主要针对Java开发者，描述百度人脸识别接口服务的相关技术内容。如果您对文档内容有任何疑问，可以通过以下几种方式联系我们：

- 在百度云控制台内提交工单，咨询问题类型请选择**人工智能服务**；
- 如有疑问，进入AI社区交流：http://ai.baidu.com/forum/topic/list/165

接口能力

接口名称	接口能力简要描述
人脸检测	检测人脸并定位，返回五官关键点，及人脸各属性值
人脸比对	返回两两比对的人脸相似值
人脸查找	在一个人脸集合中找到找到相似的人脸，由一系列接口组成，包括人脸识别、人脸认证、人脸库管理相关接口（人脸注册、人脸更新、人脸删除、用户信息查询、组列表查询、组内用户列表查询、组间复制用户、组内删除用户）

版本更新记录

上线日期	版本号	更新内容
2019.4.17	4.11.1	人脸v3文档更新，新增N:M接口
2018.6.1	4.4.0	更新视频活体接口参数名
2018.5.10	4.3.2	修复h5视频活体检测接口问题
2018.5.10	4.3.1	修复人脸活体检测接口问题
2018.4.28	4.3.0	人脸接口更新为v3版本
2018.4.3	4.2.0	新增人脸在线活体检测、身份验证接口
2018.1.11	4.1.0	新增人脸比对M:N接口
2017.12.22	4.0.0	接口统一升级
2017.11.14	3.3.2	人脸检测接口升级v2版本
2017.10.18	3.2.1	使用proxy问题修复
2017.8.25	3.0.0	更新sdk打包方式：所有AI服务集成一个SDK
2017.7.14	1.3.6	更新SDK打包方式
2017.4.27	1.3.4	人脸比对、识别、认证和人脸库设置接口升级为v2版本
2017.4.20	1.3.3	AI SDK同步版本更新
2017.4.13	1.3.2	AI SDK同步版本更新
2017.3.23	1.3	兼容Android环境
2017.3.2	1.2	上线人脸查找接口，增加对图片参数要求限制的检查，增加设置超时接口
2017.1.20	1.1	上线人脸比对接口，同时修复部分云用户调用不成功的错误
2017.1.6	1.0	初始版本，上线人脸属性识别接口

快速入门

安装Face Java SDK

Face Java SDK目录结构

```
com.baidu.aip
    ├─── auth                    //签名相关类
    ├─── http                    //Http通信相关类
    ├─── client                  //公用类
    ├─── exception                 //exception类
    ├─── face
    │      └─── AipFace       //AipFace类
    └─── util                    //工具类
```

支持 JAVA版本：1.7+

使用maven依赖：

添加以下依赖即可。其中版本号可在maven官网查询

```
 <dependency>
   <groupId>com.baidu.aip</groupId>
   <artifactId>java-sdk</artifactId>
   <version>${version}</version>
</dependency>
```

直接使用JAR包步骤如下：

1.在官方网站下载Java SDK压缩工具包。

2.将下载的`aip-java-sdk-version.zip`解压后，复制到工程文件夹中。

3.在Eclipse右键“工程 -> Properties -> Java Build Path -> Add JARs”。

4.添加SDK工具包`aip-java-sdk-version.jar`和第三方依赖工具包`json-20160810.jar` `slf4j-api-1.7.25.jar` `slf4j-simple-1.7.25.jar（可选）`。

其中，`version`为版本号，添加完成后，用户就可以在工程中使用Face Java SDK。

新建AipFace

AipFace是人脸识别的Java客户端，为使用人脸识别的开发人员提供了一系列的交互方法。

用户可以参考如下代码新建一个AipFace,初始化完成后建议**单例使用**,避免重复获取access_token：

```
public class Sample {
  //设置APPID/AK/SK
  public static final String APP_ID = "你的 App ID";
  public static final String API_KEY = "你的 Api Key";
  public static final String SECRET_KEY = "你的 Secret Key";

  public static void main(String[] args) {
     // 初始化一个AipFace
     AipFace client = new AipFace(APP_ID, API_KEY, SECRET_KEY);

     // 可选：设置网络连接参数
     client.setConnectionTimeoutInMillis(2000);
     client.setSocketTimeoutInMillis(60000);

     // 可选：设置代理服务器地址, http和socket二选一，或者均不设置
     client.setHttpProxy("proxy_host", proxy_port);  // 设置http代理
     client.setSocketProxy("proxy_host", proxy_port);  // 设置socket代理

     // 调用接口
     String image = "取决于image_type参数，传入BASE64字符串或URL字符串或FACE_TOKEN字符串";
     String imageType = "BASE64";

     // 人脸检测
     JSONObject res = client.detect(image, imageType, options);
     System.out.println(res.toString(2));

  }
}
```

在上面代码中，常量`APP_ID`在百度云控制台中创建，常量`API_KEY`与`SECRET_KEY`是在创建完毕应用后，系统分配给用户的，

均为字符串，用于标识用户，为访问做签名验证，可在AI服务控制台中的**应用列表**中查看。

注意：如您以前是百度云的老用户，其中`API_KEY`对应百度云的“Access Key ID”，`SECRET_KEY`对应百度云的“Access Key Secret”。

配置AipFace

如果用户需要配置AipFace的一些细节参数，可以在构造AipFace之后调用接口设置参数，目前只支持以下参数：

接口	说明
setConnectionTimeoutInMillis	建立连接的超时时间（单位：毫秒）
setSocketTimeoutInMillis	通过打开的连接传输数据的超时时间（单位：毫秒）
setHttpProxy	设置http代理服务器
setSocketProxy	设置socket代理服务器 （http和socket类型代理服务器只能二选一）

SDK默认使用slf4j-simple包进行日志输出，若用户需要使用自定义日志实现，可去除slf4j-simple依赖包，再额外添加相应的日志实现包即可。 maven去除slf4j-simple依赖包示例：

```
 <dependency>
   <groupId>com.baidu.aip</groupId>
   <artifactId>java-sdk</artifactId>
   <version>${version}</version>
   <exclusions>
      <exclusion>
         <groupId>org.slf4j</groupId>
         <artifactId>slf4j-simple</artifactId>
      </exclusion>
   </exclusions>
</dependency>
```

接口说明

人脸检测

人脸检测：检测图片中的人脸并标记出位置信息;

```
 public void sample(AipFace client) {
   // 传入可选参数调用接口
   HashMap<String, Object> options = new HashMap<String, Object>();
   options.put("face_field", "age");
   options.put("max_face_num", "2");
   options.put("face_type", "LIVE");
   options.put("liveness_control", "LOW");

   String image = "取决于image_type参数，传入BASE64字符串或URL字符串或FACE_TOKEN字符串";
   String imageType = "BASE64";

   // 人脸检测
   JSONObject res = client.detect(image, imageType, options);
   System.out.println(res.toString(2));

 }
```

人脸检测 请求参数详情

参数	必选	类型	说明
image	是	string	图片信息(**总数据大小应小于10M**)，图片上传方式根据image_type来判断
image_type	是	string	图片类型 **BASE64**:图片的base64值，base64编码后的图片数据，编码后的图片大小不超过2M； **URL**:图片的 URL地址(可能由于网络等原因导致下载图片时间过长)； **FACE_TOKEN**: 人脸图片的唯一标识，调用人脸检测接口时，会为每个人脸图片赋予一个唯一的FACE_TOKEN，同一张图片多次检测得到的FACE_TOKEN是同一个。
face_field	否	string	包括**age,expression,face_shape,gender,glasses,landmark,landmark150,quality,eye_status,emotion,face_type,mask,spoofing信息** 逗号分隔. 默认只返回face_token、人脸框、概率和旋转角度
max_face_num	否	uint32	最多处理人脸的数目，默认值为1，根据人脸检测排序类型检测图片中排序第一的人脸（默认为人脸面积最大的人脸），**最大值120**
face_type	否	string	人脸的类型 **LIVE**表示生活照：通常为手机、相机拍摄的人像图片、或从网络获取的人像图片等 **IDCARD**表示身份证芯片照：二代身份证内置芯片中的人像照片 **WATERMARK**表示带水印证件照：一般为带水印的小图，如公安网小图 **CERT**表示证件照片：如拍摄的身份证、工卡、护照、学生证等证件图片 默认**LIVE**
liveness_control	否	string	**活体控制** 检测结果中不符合要求的人脸会被过滤 **NONE**: 不进行控制 **LOW**:较低的活体要求(高通过率 低攻击拒绝率) **NORMAL**: 一般的活体要求(平衡的攻击拒绝率, 通过率) **HIGH**: 较高的活体要求(高攻击拒绝率 低通过率) 默认**NONE**
face_sort_type	否	int	人脸检测排序类型 **0**:代表检测出的人脸按照人脸面积从大到小排列 **1**:代表检测出的人脸按照距离图片中心从近到远排列 **默认为0**
display_corp_image	否	int	是否显示检测人脸的裁剪图base64值 0：不显示（默认） 1：显示 **当取值为1时，max_face_num字段的取值上限按5计算，即最多可返回5张人脸的裁剪图**

人脸检测 返回数据参数详情

字段	必选	类型	说明
face_num	是	int	检测到的图片中的人脸数量
face_list	是	array	人脸信息列表，具体包含的参数参考下面的列表。
+face_token	是	string	人脸图片的唯一标识
+location	是	array	人脸在图片中的位置
++left	是	double	人脸区域离左边界的距离
++top	是	double	人脸区域离上边界的距离
++width	是	double	人脸区域的宽度
++height	是	double	人脸区域的高度
++rotation	是	int64	人脸框相对于竖直方向的顺时针旋转角，[-180,180]
+face_probability	是	double	人脸置信度，范围【0~1】，代表这是一张人脸的概率，0最小、1最大。

+angel	是	array	人脸旋转角度参数
++yaw	是	double	三维旋转之左右旋转角[-90(左), 90(右)]
++pitch	是	double	三维旋转之俯仰角度[-90(上), 90(下)]
++roll	是	double	平面内旋转角[-180(逆时针), 180(顺时针)]
+age	否	double	年龄 ，当**face_field包含age时返回**
+expression	否	array	表情，当 **face_field包含expression时返回**
++type	否	string	**none**:不笑；**smile**:微笑；**laugh**:大笑
++probability	否	double	表情置信度，范围【0~1】，0最小、1最大。
+face_shape	否	array	脸型，当**face_field包含face_shape时返回**
++type	否	double	**square**: 正方形 **triangle**:三角形 **oval**: 椭圆 **heart**: 心形 **round**: 圆形
++probability	否	double	置信度，范围【0~1】，代表这是人脸形状判断正确的概率，0最小、1最大。
+gender	否	array	性别，**face_field包含gender时返回**
++type	否	string	male:**男性** female:**女性**
++probability	否	double	性别置信度，范围【0~1】，0代表概率最小、1代表最大。
+glasses	否	array	是否带眼镜，**face_field包含glasses时返回**
++type	否	string	**none**:无眼镜，**common**:普通眼镜，**sun**:墨镜
++probability	否	double	眼镜置信度，范围【0~1】，0代表概率最小、1代表最大。
+eye_status	否	array	双眼状态（睁开/闭合） **face_field包含eye_status时返回**
++left_eye	否	double	左眼状态 [0,1]取值，越接近0闭合的可能性越大
++right_eye	否	double	右眼状态 [0,1]取值，越接近0闭合的可能性越大
+emotion	否	array	情绪 **face_field包含emotion时返回**
++type	否	string	**angry**:愤怒 **disgust**:厌恶 **fear**:恐惧 **happy**:高兴 **sad**:伤心 **surprise**:惊讶 **neutral**:无情绪
++probability	否	double	情绪置信度，范围0~1
++probability	否	double	人种置信度，范围【0~1】，0代表概率最小、1代表最大。
+face_type	否	array	真实人脸/卡通人脸 **face_field包含face_type时返回**
++type	否	string	**human**: 真实人脸 **cartoon**: 卡通人脸
++probability	否	double	人脸类型判断正确的置信度，范围【0~1】，0代表概率最小、1代表最大。
+landmark	否	array	4个关键点位置，左眼中心、右眼中心、鼻尖、嘴中心。**face_field包含landmark时返回**
+landmark72	否	array	72个特征点位置 **face_field包含landmark时返回**
+landmark150	否	array	150个特征点位置 **face_field包含landmark150时返回**
+quality	否	array	人脸质量信息。**face_field包含quality时返回**
++occlusion	否	array	人脸各部分遮挡的概率，范围[0~1]，0表示完整，1表示不完整
+++left_eye	否	double	左眼遮挡比例，[0-1] ，1表示完全遮挡
+++right_eye	否	double	右眼遮挡比例，[0-1] ， 1表示完全遮挡
+++nose	否	double	鼻子遮挡比例，[0-1] ， 1表示完全遮挡
+++mouth	否	double	嘴巴遮挡比例，[0-1] ， 1表示完全遮挡
+++left_cheek	否	double	左脸颊遮挡比例，[0-1] ， 1表示完全遮挡
+++right_cheek	否	double	右脸颊遮挡比例，[0-1] ， 1表示完全遮挡
+++chin_contour	否	double	下巴遮挡比例，，[0-1] ， 1表示完全遮挡

++blur	否	double	人脸模糊程度，范围[0~1]，0表示清晰，1表示模糊
++illumination	否	double	取值范围在[0~255], 表示脸部区域的光照程度 越大表示光照越好
++completeness	否	int64	人脸完整度，0或1, 0为人脸溢出图像边界，1为人脸都在图像边界内

人脸检测 返回示例

```
{
 "face_num": 1,
 "face_list": [
     {
        "face_token": "35235asfas21421fakghktyfdgh68bio",
        "location": {
           "left": 117,
           "top": 131,
           "width": 172,
           "height": 170,
           "rotation": 4
        },
        "face_probability": 1,
        "angle" :{
            "yaw" : -0.34859421849251
            "pitch" 1.9135693311691
            "roll" :2.3033397197723
        }
        "landmark": [
           {
              "x": 161.74819946289,
              "y": 163.30244445801
           },
           ...
        ],
        "landmark72": [
           {
              "x": 115.86531066895,
              "y": 170.0546875
           } ,
           ...
        ],
        "age": 29.298097610474,
        "expression": {
           "type": "smile",
           "probability" : 0.5543018579483
        },
        "gender": {
           "type": "male",
           "probability": 0.99979132413864
        },
        "glasses": {
       "type": "sun",
           "probability": 0.99999964237213
        },
        "face_shape": {
           "type": "triangle",
           "probability": 0.5543018579483
        }
        "quality": {
           "occlusion": {
              "left_eye": 0,
              "right_eye": 0,
              "nose": 0,
```

```
                "mouth": 0,
                "left_cheek": 0.0064102564938366,
                "right_cheek": 0.0057411273010075,
                "chin": 0
            },
            "blur": 1.1886881756684e-10,
            "illumination": 141,
            "completeness": 1
        }
    }
]
}
```

**72个关键点分布图（对应landmark72个点的顺序，序号从0-71）：https://ai.bdstatic.com/file/52BC00FFD4754A6298D977EDAD033DA0

人脸搜索

- **1：N人脸搜索**：也称为1：N识别，在指定人脸集合中，找到最相似的人脸；
- **1：N人脸认证**：基于uid维度的1：N识别，由于uid已经锁定固定数量的人脸，所以检索范围更聚焦；

> **1：N人脸识别**与**1：N人脸认证**的差别在于：人脸搜索是在指定人脸集合中进行直接地人脸检索操作，而人脸认证是基于uid，先调取这个uid对应的人脸，再在这个uid对应的人脸集合中进行检索（因为每个uid通常对应的只有一张人脸，所以通常也就变为了1：1对比）；实际应用中，人脸认证需要用户或系统先输入id，这增加了验证安全度，但也增加了复杂度，具体使用哪个接口需要视您的业务场景判断。

```
public void sample(AipFace client) {
  // 传入可选参数调用接口
  HashMap<String, String> options = new HashMap<String, Object>();
  options.put("match_threshold", "70");
  options.put("quality_control", "NORMAL");
  options.put("liveness_control", "LOW");
  options.put("user_id", "233451");
  options.put("max_user_num", "3");

  String image = "取决于image_type参数，传入BASE64字符串或URL字符串或FACE_TOKEN字符串";
  String imageType = "BASE64";
  String groupIdList = "3,2";

  // 人脸搜索
  JSONObject res = client.search(image, imageType, groupIdList, options);
  System.out.println(res.toString(2));

}
```

人脸搜索 请求参数详情

参数名称	是否必选	类型	默认值	说明
image	是	String		图片信息(**总数据大小应小于10M**)，图片上传方式根据image_type来判断
image_type	是	String		图片类型 **BASE64**:图片的base64值，base64编码后的图片数据，编码后的图片大小不超过2M； **URL**:图片的 URL地址(可能由于网络等原因导致下载图片时间过长)； **FACE_TOKEN**: 人脸图片的唯一标识，调用人脸检测接口时，会为每个人脸图片赋予一个唯一的FACE_TOKEN，同一张图片多次检测得到的FACE_TOKEN是同一个。
group_id_list	是	String		从指定的group中进行查找 用逗号分隔，**上限10个**
match_threshold	否	String		匹配阈值（设置阈值后，score低于此阈值的用户信息将不会返回） 最大100 最小0 默认80 **此阈值设置得越高，检索速度将会越快，推荐使用默认阈值80**
quality_control	否	String	NONE	图片质量控制 **NONE**: 不进行控制 **LOW**:较低的质量要求 **NORMAL**: 一般的质量要求 **HIGH**: 较高的质量要求 **默认 NONE**
liveness_control	否	String	NONE	活体检测控制 **NONE**: 不进行控制 **LOW**:较低的活体要求(高通过率 低攻击拒绝率) **NORMAL**: 一般的活体要求(平衡的攻击拒绝率, 通过率) **HIGH**: 较高的活体要求(高攻击拒绝率 低通过率) **默认NONE**
user_id	否	String		当需要对特定用户进行比对时，指定user_id进行比对。即人脸认证功能。
max_user_num	否	String		查找后返回的用户数量。返回相似度最高的几个用户，默认为1，最多返回50个。

人脸搜索 返回数据参数详情

字段	必选	类型	说明
face_token	是	string	人脸标志
user_list	是	array	匹配的用户信息列表
+group_id	是	string	用户所属的group_id
+user_id	是	string	用户的user_id
+user_info	是	string	注册用户时携带的user_info
+score	是	float	用户的匹配得分

人脸搜索 返回示例

```
{
 "face_token": "fid",
 "user_list": [
   {
     "group_id" : "test1",
     "user_id": "u333333",
     "user_info": "Test User",
     "score": 99.3
   }
 ]
}
```

人脸搜索 M:N 识别

待识别的图片中，存在多张人脸的情况下，支持在一个人脸库中，一次请求，同时返回图片中所有人脸的识别结果。

```
public void sample(AipFace client) {
  // 传入可选参数调用接口
  HashMap<String, String> options = new HashMap<String, Object>();
  options.put("max_face_num", "3");
  options.put("match_threshold", "70");
  options.put("quality_control", "NORMAL");
  options.put("liveness_control", "LOW");
  options.put("max_user_num", "3");

  String image = "取决于image_type参数，传入BASE64字符串或URL字符串或FACE_TOKEN字符串";
  String imageType = "BASE64";
  String groupIdList = "3,2";

  // 人脸搜索 M:N 识别
  JSONObject res = client.multiSearch(image, imageType, groupIdList, options);
  System.out.println(res.toString(2));

}
```

人脸搜索 M:N 识别 请求参数详情

参数名称	是否必选	类型	默认值	说明
image	是	String		图片信息(**总数据大小应小于10M**)，图片上传方式根据image_type来判断
image_type	是	String		图片类型 **BASE64**:图片的base64值，base64编码后的图片数据，编码后的图片大小不超过2M； **URL**:图片的 URL地址(可能由于网络等原因导致下载图片时间过长)； **FACE_TOKEN**: 人脸图片的唯一标识，调用人脸检测接口时，会为每个人脸图片赋予一个唯一的FACE_TOKEN，同一张图片多次检测得到的FACE_TOKEN是同一个。
group_id_list	是	String		从指定的group中进行查找 用逗号分隔，**上限10个**
max_face_num	否	String		最多处理人脸的数目 **默认值为1(仅检测图片中面积最大的那个人脸) 最大值10**
match_threshold	否	String		匹配阈值（设置阈值后，score低于此阈值的用户信息将不会返回） 最大100 最小0 默认80 **此阈值设置得越高，检索速度将会越快，推荐使用默认阈值80**
quality_control	否	String	NONE	图片质量控制 **NONE**: 不进行控制 **LOW**:较低的质量要求 **NORMAL**: 一般的质量要求 **HIGH**: 较高的质量要求 **默认 NONE**
liveness_control	否	String	NONE	活体检测控制 **NONE**: 不进行控制 **LOW**:较低的活体要求(高通过率 低攻击拒绝率) **NORMAL**: 一般的活体要求(平衡的攻击拒绝率, 通过率) **HIGH**: 较高的活体要求(高攻击拒绝率 低通过率) **默认NONE**
max_user_num	否	String		查找后返回的用户数量。返回相似度最高的几个用户，默认为1，最多返回20个。

人脸搜索 M:N 识别 返回数据参数详情

字段	必选	类型	说明
face_num	是	int	图片中的人脸数量
face_list	是	array	人脸信息列表
+face_token	是	string	人脸标志
+location	是	array	人脸在图片中的位置
++left	是	double	人脸区域离左边界的距离
++top	是	double	人脸区域离上边界的距离
++width	是	double	人脸区域的宽度
++height	是	double	人脸区域的高度
++rotation	是	int64	人脸框相对于竖直方向的顺时针旋转角，[-180,180]
+user_list	是	array	匹配的用户信息列表
++group_id	是	string	用户所属的group_id
++user_id	是	string	用户的user_id
++user_info	是	string	注册用户时携带的user_info
++score	是	float	用户的匹配得分 **80分以上可以判断为同一人，此分值对应万分之一误识率**

人脸搜索 M:N 识别 返回示例

```
{
  "error_code": 0,
  "error_msg": "SUCCESS",
  "log_id": 240483475,
  "timestamp": 1535533440,
  "cached": 0,
  "result": {
    "face_num": 2,
    "face_list": [
      {
        "face_token": "6fe19a6ee0c4233db9b5bba4dc2b9233",
        "location": {
          "left": 31.95568085,
          "top": 120.3764267,
          "width": 87,
          "height": 85,
          "rotation": -5
        },
        "user_list": [
          {
            "group_id": "group1",
            "user_id": "5abd24fd062e49bfa906b257ec40d284",
            "user_info": "userinfo1",
            "score": 69.85684967041
          },
          {
            "group_id": "group1",
            "user_id": "2abf89cffb31473a9948268fde9e1c3f",
            "user_info": "userinfo2",
            "score": 66.586112976074
          }
        ]
      },
      {
        "face_token": "fde61e9c074f48cf2bbb319e42634f41",
        "location": {
          "left": 219.4467773,
          "top": 104.7486954,
          "width": 81,
          "height": 77,
          "rotation": 3
        },
        "user_list": [
          {
            "group_id": "group1",
            "user_id": "088717532b094c3990755e91250adf7d",
            "user_info": "userinfo",
            "score": 65.154159545898
          }
        ]
      }
    ]
  }
}
```


人脸注册

用于从人脸库中新增用户，可以设定多个用户所在组，及组内用户的人脸图片，

典型应用场景：构建您的人脸库，如**会员人脸注册**，**已有用户补全人脸信息**等。

人脸库、用户组、用户、用户下的人脸**层级关系**如下所示：

```
|- 人脸库
 |- 用户组一
   |- 用户01
     |- 人脸
   |- 用户02
     |- 人脸
     |- 人脸
     ....
    ....
 |- 用户组二
 |- 用户组三
 |- 用户组四
 ....
```

关于人脸库的设置限制

- **每个appid对应一个人脸库，且不同appid之间，人脸库互不相通**；
- 每个人脸库下，可以创建多个用户组，用户组（group）数量**没有限制**；
- 每个用户组（group）下，可添加最多**无限**张人脸，**无限**个uid；
- 每个用户（uid）所能注册的最大人脸数量**20个**；

为了保证识别效果，请控制注册人脸的质量（通过`/detect`人脸检测接口判断），具体参数可详见下表所示：

质量判断

可通过人脸检测接口，基于以下字段和对应阈值，进行质量检测的判断，以保证人脸质量符合后续业务操作要求。

指标	字段与解释	推荐数值界限
遮挡范围	**occlusion**（0~1），0为无遮挡，1是完全遮挡 含有多个具体子字段，表示脸部多个部位 通常用作判断头发、墨镜、口罩等遮挡	left_eye：0.6, #左眼被遮挡的阈值 right_eye：0.6, #右眼被遮挡的阈值 nose：0.7, #鼻子被遮挡的阈值 mouth：0.7, #嘴巴被遮挡的阈值 left_check：0.8, #左脸颊被遮挡的阈值 right_check：0.8, #右脸颊被遮挡的阈值 chin_contour：0.6, #下巴被遮挡阈值
模糊度范围	**Blur**（0~1），0是最清晰，1是最模糊	小于0.7
光照范围	**illumination**（0~255） 脸部光照的灰度值，0表示光照不好 以及对应客户端SDK中，YUV的Y分量	大于40
姿态角度	**Pitch**：三维旋转之俯仰角度[-90(上), 90(下)] **Roll**：平面内旋转角[-180(逆时针), 180(顺时针)] **Yaw**：三维旋转之左右旋转角[-90(左), 90(右)]	分别小于20度
人脸完整度	**completeness**（0或1），0为人脸溢出图像边界，1为人脸都在图像边界内	视业务逻辑判断
人脸大小	人脸部分的大小 建议长宽像素值范围：80*80~200*200	人脸部分不小于**100*100**像素

```
public void sample(AipFace client) {
  // 传入可选参数调用接口
  HashMap<String, String> options = new HashMap<String, Object>();
  options.put("user_info", "user's info");
  options.put("quality_control", "NORMAL");
  options.put("liveness_control", "LOW");
  options.put("action_type", "REPLACE");

  String image = "取决于image_type参数，传入BASE64字符串或URL字符串或FACE_TOKEN字符串";
  String imageType = "BASE64";
  String groupId = "group1";
  String userId = "user1";

  // 人脸注册
  JSONObject res = client.addUser(image, imageType, groupId, userId, options);
  System.out.println(res.toString(2));

}
```

人脸注册 请求参数详情

参数名称	是否必选	类型	默认值	说明
image	是	String		图片信息(总数据大小应小于10M)，图片上传方式根据image_type来判断。注：组内每个uid下的人脸图片数目上限为20张
image_type	是	String		图片类型 **BASE64**:图片的base64值，base64编码后的图片数据，编码后的图片大小不超过2M； **URL**:图片的 URL地址(可能由于网络等原因导致下载图片时间过长)； **FACE_TOKEN**: 人脸图片的唯一标识，调用人脸检测接口时，会为每个人脸图片赋予一个唯一的FACE_TOKEN，同一张图片多次检测得到的FACE_TOKEN是同一个。
group_id	是	String		用户组id（由数字、字母、下划线组成），长度限制48B
user_id	是	String		用户id（由数字、字母、下划线组成），长度限制48B
user_info	否	String		用户资料，长度限制256B
quality_control	否	String	NONE	图片质量控制 **NONE**: 不进行控制 **LOW**:较低的质量要求 **NORMAL**: 一般的质量要求 **HIGH**: 较高的质量要求 **默认 NONE**
liveness_control	否	String	NONE	活体检测控制 **NONE**: 不进行控制 **LOW**:较低的活体要求(高通过率 低攻击拒绝率) **NORMAL**: 一般的活体要求(平衡的攻击拒绝率, 通过率) **HIGH**: 较高的活体要求(高攻击拒绝率 低通过率) **默认NONE**
action_type	否	String	APPEND	操作方式 APPEND: 当user_id在库中已经存在时，对此user_id重复注册时，新注册的图片默认会追加到该user_id下,REPLACE：当对此user_id重复注册时,则会用新图替换库中该user_id下所有图片,默认使用APPEND

人脸注册 返回数据参数详情

字段	必选	类型	说明
log_id	是	uint64	请求标识码，随机数，唯一
face_token	是	string	人脸图片的唯一标识
location	是	array	人脸在图片中的位置
+left	是	double	人脸区域离左边界的距离
+top	是	double	人脸区域离上边界的距离
+width	是	double	人脸区域的宽度
+height	是	double	人脸区域的高度
+rotation	是	int64	人脸框相对于竖直方向的顺时针旋转角，[-180,180]

人脸注册 返回示例

```
{
 "face_token": "2fa64a88a9d5118916f9a303782a97d3",
 "location": {
     "left": 117,
     "top": 131,
     "width": 172,
     "height": 170,
     "rotation": 4
 }
}
```

人脸更新

用于对人脸库中指定用户，更新其下的人脸图像。

> **说明：**针对一个uid执行更新操作，新上传的人脸图像将覆盖此uid原有所有图像。

```
public void sample(AipFace client) {
  // 传入可选参数调用接口
  HashMap<String, String> options = new HashMap<String, Object>();
  options.put("user_info", "user's info");
  options.put("quality_control", "NORMAL");
  options.put("liveness_control", "LOW");
  options.put("action_type", "REPLACE");

  String image = "取决于image_type参数，传入BASE64字符串或URL字符串或FACE_TOKEN字符串";
  String imageType = "BASE64";
  String groupId = "group1";
  String userId = "user1";

  // 人脸更新
  JSONObject res = client.updateUser(image, imageType, groupId, userId, options);
  System.out.println(res.toString(2));

}
```

人脸更新 请求参数详情

参数名称	是否必选	类型	默认值	说明
image	是	String		图片信息(**总数据大小应小于10M**)，图片上传方式根据image_type来判断
image_type	是	String		图片类型 **BASE64**:图片的base64值，base64编码后的图片数据，编码后的图片大小不超过2M； **URL**:图片的 URL地址(可能由于网络等原因导致下载图片时间过长)； **FACE_TOKEN**: 人脸图片的唯一标识，调用人脸检测接口时，会为每个人脸图片赋予一个唯一的FACE_TOKEN，同一张图片多次检测得到的FACE_TOKEN是同一个。
group_id	是	String		更新指定groupid下uid对应的信息
user_id	是	String		用户id（由数字、字母、下划线组成），长度限制48B
user_info	否	String		用户资料，长度限制256B
quality_control	否	String	NONE	图片质量控制 **NONE**: 不进行控制 **LOW**:较低的质量要求 **NORMAL**: 一般的质量要求 **HIGH**: 较高的质量要求 **默认 NONE**
liveness_control	否	String	NONE	活体检测控制 **NONE**: 不进行控制 **LOW**:较低的活体要求(高通过率 低攻击拒绝率) **NORMAL**: 一般的活体要求(平衡的攻击拒绝率, 通过率) **HIGH**: 较高的活体要求(高攻击拒绝率 低通过率) **默认NONE**
action_type	否	String	UPDATE	操作方式 **UPDATE**: 当user_id在库中已经存在时，对此user_id重复注册时，新注册的图片默认会追加到该user_id下,**REPLACE**：当对此user_id重复注册时,则会用新图替换库中该user_id下所有图片,默认使用**UPDATE**

人脸更新 返回数据参数详情

字段	必选	类型	说明
log_id	是	uint64	请求标识码，随机数，唯一
face_token	是	string	人脸图片的唯一标识
location	是	array	人脸在图片中的位置
+left	是	double	人脸区域离左边界的距离
+top	是	double	人脸区域离上边界的距离
+width	是	double	人脸区域的宽度
+height	是	double	人脸区域的高度
+rotation	是	int64	人脸框相对于竖直方向的顺时针旋转角，[-180,180]

人脸更新 返回示例

```
{
 "face_token": "2fa64a88a9d5118916f9a303782a97d3",
 "location": {
    "left": 117,
    "top": 131,
    "width": 172,
    "height": 170,
    "rotation": 4
 }
}
```

人脸删除

用于从人脸库中删除一个用户。

人脸删除注意事项：

- 删除的内容，包括用户所有图像和身份信息；
- 如果一个uid存在于多个用户组内，将会同时将从各个组中把用户删除
- 如果指定了group_id，则只删除此group下的uid相关信息

```
public void sample(AipFace client) {
  // 传入可选参数调用接口
  HashMap<String, String> options = new HashMap<String, Object>();

  String userId = "user1";
  String groupId = "group1";
  String faceToken = "face_token_23123";

  // 人脸删除
  JSONObject res = client.faceDelete(userId, groupId, faceToken, options);
  System.out.println(res.toString(2));

}
```

人脸删除 请求参数详情

参数名称	是否必选	类型	说明
log_id	是	uint64	请求标识码，随机数，唯一
user_id	是	String	用户id（由数字、字母、下划线组成），长度限制48B
group_id	是	String	用户组id（由数字、字母、下划线组成），长度限制48B
face_token	是	String	需要删除的人脸图片token，（由数字、字母、下划线组成）长度限制64B

人脸删除 返回数据参数详情

字段	是否必选	类型	说明
log_id	是	uint64	请求唯一标识码，随机数

人脸删除 返回示例

```
 // 删除成功
{
  "error_code": 0,
  "log_id": 73473737,
}
// 删除发生错误
{
  "error_code": 223106,
  "log_id": 1382953199,
  "error_msg": "face is not exist"
}
```

用户信息查询

获取人脸库中某个用户的信息(user_info信息和用户所属的组)。

```
public void sample(AipFace client) {
  // 传入可选参数调用接口
  HashMap<String, String> options = new HashMap<String, String>();

  String userId = "user1";
  String groupId = "group1";

  // 用户信息查询
  JSONObject res = client.getUser(userId, groupId, options);
  System.out.println(res.toString(2));

}
```

用户信息查询 请求参数详情

参数名称	是否必选	类型	说明
user_id	是	String	用户id（由数字、字母、下划线组成），长度限制48B
group_id	是	String	用户组id（由数字、字母、下划线组成），长度限制48B

用户信息查询 返回数据参数详情

字段	必选	类型	说明
log_id	是	uint64	请求标识码，随机数，唯一
user_list	是	array	查询到的用户列表
+user_info	是	string	用户资料，被查询用户的资料
+group_id	是	string	用户组id，被查询用户的所在组

用户信息查询 返回示例

```
{
 "user_list": [
    {
        "user_info": "user info ...",
        "group_id": "gid1"
    },
    {
        "user_info": "user info2 ...",
        "group_id": "gid2"
    }
 ]
}
```

获取用户人脸列表

用于获取一个用户的全部人脸列表。

```
public void sample(AipFace client) {
  // 传入可选参数调用接口
  HashMap<String, String> options = new HashMap<String, Object>();

  String userId = "user1";
  String groupId = "group1";

  // 获取用户人脸列表
  JSONObject res = client.faceGetlist(userId, groupId, options);
  System.out.println(res.toString(2));

}
```

获取用户人脸列表 请求参数详情

参数名称	是否必选	类型	说明
user_id	是	String	用户id（由数字、字母、下划线组成），长度限制48B
group_id	是	String	用户组id（由数字、字母、下划线组成），长度限制48B

获取用户人脸列表 返回数据参数详情

字段	必选	类型	说明
log_id	是	uint64	请求标识码，随机数，唯一
face_list	是	array	人脸列表
+face_token	是	string	人脸图片的唯一标识
+ctime	是	string	人脸创建时间

获取用户人脸列表 返回示例

```
{
 "face_list": [
    {
        "face_token": "fid1",
        "ctime": "2018-01-01 00:00:00"
    },
    {
        "face_token": "fid2",
        "ctime": "2018-01-01 10:00:00"
    }
 ]
}
```

获取用户列表

用于查询指定用户组中的用户列表。

```
public void sample(AipFace client) {
  // 传入可选参数调用接口
  HashMap<String, String> options = new HashMap<String, Object>();
  options.put("start", "0");
  options.put("length", "50");

  String groupId = "group1";

  // 获取用户列表
  JSONObject res = client.getGroupUsers(groupId, options);
  System.out.println(res.toString(2));

}
```

获取用户列表 请求参数详情

参数名称	是否必选	类型	默认值	说明
group_id	是	String		用户组id（由数字、字母、下划线组成），长度限制48B
start	否	String	0	默认值0，起始序号
length	否	String	100	返回数量，默认值100，最大值1000

获取用户列表 返回数据参数详情

字段	必选	类型	说明
user_id_list	是	array	用户ID列表

获取用户列表 返回示例

```
{
 "user_id_list": [
    "uid1",
    "uid2"
 ]
}
```

复制用户

用于将已经存在于人脸库中的用户**复制到一个新的组**。

```
public void sample(AipFace client) {
  // 传入可选参数调用接口
  HashMap<String, String> options = new HashMap<String, Object>();
  options.put("src_group_id", "11111");
  options.put("dst_group_id", "222222");

  String userId = "user1";

  // 复制用户
  JSONObject res = client.userCopy(userId, options);
  System.out.println(res.toString(2));

}
```

复制用户 请求参数详情

参数名称	是否必选	类型	说明
user_id	是	String	用户id（由数字、字母、下划线组成），长度限制48B
src_group_id	否	String	从指定组里复制信息
dst_group_id	否	String	需要添加用户的组id

复制用户 返回数据参数详情

字段	必选	类型	说明
log_id	是	id	log_id

复制用户 返回示例

```
 // 正确返回值
{
  "error_code": 0,
 "log_id": 3314921889,
}
// 发生错误时返回值
{
 "error_code": 216100,
 "log_id": 3111284097,
 "error_msg": "already add"
}
```

删除用户

用于将用户从某个组中删除。

```
public void sample(AipFace client) {
  // 传入可选参数调用接口
  HashMap<String, String> options = new HashMap<String, Object>();

  String groupId = "group1";
  String userId = "user1";

  // 删除用户
  JSONObject res = client.deleteUser(groupId, userId, options);
  System.out.println(res.toString(2));

}
```

删除用户 请求参数详情

参数名称	是否必选	类型	说明
group_id	是	String	用户组id（由数字、字母、下划线组成），长度限制48B
user_id	是	String	用户id（由数字、字母、下划线组成），长度限制48B

删除用户 返回数据参数详情

字段	是否必选	类型	说明
log_id	是	uint64	请求唯一标识码，随机数

删除用户 返回示例

```
 // 正确返回值
{

 "error_code": 0,
 "log_id": 3314921889,
}
// 发生错误时返回值
{
 "error_code": 223103,
 "log_id": 815967402,
 "error_msg": "user is not exist"
}
```

创建用户组

用于创建一个空的用户组，如果用户组已存在 则返回错误。

```
public void sample(AipFace client) {
  // 传入可选参数调用接口
  HashMap<String, String> options = new HashMap<String, Object>();

  String groupId = "group1";

  // 创建用户组
  JSONObject res = client.groupAdd(groupId, options);
  System.out.println(res.toString(2));

}
```

创建用户组 请求参数详情

参数名称	是否必选	类型	说明
group_id	是	String	用户组id（由数字、字母、下划线组成），长度限制48B

创建用户组 返回数据参数详情

字段	必选	类型	说明
log_id	是	uint64	请求标识码，随机数，唯一

创建用户组 返回示例

```
 {

 "error_code": 0,
 "log_id": 3314921889,
}
// 发生错误时返回值
{
 "error_code": 223101,
 "log_id": 815967402,
 "error_msg": " group is already exist"
}
```

删除用户组

删除用户组下所有的用户及人脸，如果组不存在 则返回错误。

```
public void sample(AipFace client) {
  // 传入可选参数调用接口
  HashMap<String, String> options = new HashMap<String, Object>();

  String groupId = "group1";

  // 删除用户组
  JSONObject res = client.groupDelete(groupId, options);
  System.out.println(res.toString(2));

}
```

删除用户组 请求参数详情

参数名称	是否必选	类型	说明
group_id	是	String	用户组id（由数字、字母、下划线组成），长度限制48B

删除用户组 返回数据参数详情

字段	必选	类型	说明
log_id	是	uint64	请求标识码，随机数，唯一

删除用户组 返回示例

```
 // 正确返回值
{

 "error_code":0,
 "log_id": 3314921889,
}
// 发生错误时返回值
{
 "error_code": 223100,
 "log_id": 815967402,
 "error_msg": " group is not exist"
}
```

组列表查询

用于查询用户组的列表。

```
public void sample(AipFace client) {
  // 传入可选参数调用接口
  HashMap<String, String> options = new HashMap<String, Object>();
  options.put("start", "0");
  options.put("length", "50");


  // 组列表查询
  JSONObject res = client.getGroupList(options);
  System.out.println(res.toString(2));

}
```

组列表查询 请求参数详情

参数名称	是否必选	类型	默认值	说明
start	否	String	0	默认值0，起始序号
length	否	String	100	返回数量，默认值100，最大值1000

组列表查询 返回数据参数详情

字段	必选	类型	说明
group_id_list	是	array	group

组列表查询 返回示例

```
{
  "group_id_list": [
    "gid1",
    "gid2"
  ]
}
```

身份验证

质量检测（可选）活体检测（可选）公安验证（必选）

```
public void sample(AipFace client) {
  // 传入可选参数调用接口
  HashMap<String, String> options = new HashMap<String, Object>();
  options.put("quality_control", "NORMAL");
  options.put("liveness_control", "LOW");

  String image = "取决于image_type参数，传入BASE64字符串或URL字符串或FACE_TOKEN字符串";
  String imageType = "BASE64";
  String idCardNumber = "110233112299822211";
  String name = "张三";

  // 身份验证
  JSONObject res = client.personVerify(image, imageType, idCardNumber, name, options);
  System.out.println(res.toString(2));

}
```

身份验证 请求参数详情

参数名称	是否必选	类型	默认值	说明
image	是	String		图片信息(**总数据大小应小于10M**)，图片上传方式根据image_type来判断
image_type	是	String		图片类型 **BASE64**:图片的base64值，base64编码后的图片数据，编码后的图片大小不超过2M； **URL**:图片的 URL地址(可能由于网络等原因导致下载图片时间过长)； **FACE_TOKEN**: 人脸图片的唯一标识，调用人脸检测接口时，会为每个人脸图片赋予一个唯一的FACE_TOKEN，同一张图片多次检测得到的FACE_TOKEN是同一个。
id_card_number	是	String		身份证号（真实身份证号号码）
name	是	String		utf8，姓名（真实姓名，和身份证号匹配）
quality_control	否	String	NONE	图片质量控制 **NONE**: 不进行控制 **LOW**:较低的质量要求 **NORMAL**: 一般的质量要求 **HIGH**: 较高的质量要求 **默认 NONE**
liveness_control	否	String	NONE	活体检测控制 **NONE**: 不进行控制 **LOW**:较低的活体要求(高通过率 低攻击拒绝率) **NORMAL**: 一般的活体要求(平衡的攻击拒绝率, 通过率) **HIGH**: 较高的活体要求(高攻击拒绝率 低通过率) **默认NONE**

身份验证 返回数据参数详情

参数	必须	类型	说明
log_id	是	uint64	日志id
score	是	float	与公安小图相似度可能性，用于验证生活照与公安小图是否为同一人，有正常分数时为[0~1]，推荐阈值0.8，超过即判断为同一人

身份验证 返回示例

```
{
 "score": 44.3,
}
```

语音校验码接口

此接口主要用于生成随机码，用于视频的语音识别校验使用，以判断视频的即时性，而非事先录制的，提升作弊的难度。

```
public void sample(AipFace client) {
  // 传入可选参数调用接口
  HashMap<String, String> options = new HashMap<String, Object>();
  options.put("appid", "223245");


  // 语音校验码接口
  JSONObject res = client.videoSessioncode(, options);
  System.out.println(res.toString(2));

}
```

语音校验码接口 请求参数详情

参数名称	是否必选	类型	说明
appid	否	String	百度云创建应用时的唯一标识ID

语音校验码接口 返回数据参数详情

字段	必选	类型	说明
session_id	是	string	语音校验码会话id
code	是	string	语音验证码，数字形式，3~6位数字

语音校验码接口 返回示例

```
{
 "err_no": 0,
 "err_msg": "SUCCESS",
 "result": {
   "session_id": "S59faeeebb9111890355690",
   "code": "9940"
 },
 "timestamp": 1509617387,
 "cached": 0,
 "serverlogid": "0587756642"
}
```

视频活体检测接口

此接口一方面通过语音识别得到校验码，通过session code来判断视频是否作弊。另一方面进行视频抽帧，判断是否为活体。

```
public void sample(AipFace client) {
  // 传入可选参数调用接口
  HashMap<String, String> options = new HashMap<String, Object>();

  String sessionId = "1102331122998222l1";

  // 参数为本地路径
  String video = "video.mp4";
  JSONObject res = client.videoFaceliveness(sessionId, video, options);
  System.out.println(res.toString(2));

  // 参数为二进制数组
  byte[] file = readFile("video.mp4");
  res = client.videoFaceliveness(file, sessionId, options);
  System.out.println(res.toString(2));
}
```

视频活体检测接口 请求参数详情

参数名称	是否必选	类型	说明
session_id	是	String	语音校验码会话id，使用此接口的前提是已经调用了语音校验码接口
video_base64	是	String	base64编码后的视频数据（视频限制：最佳为上传5-15s的mp4文件。视频编码方式：h264编码；音频编码格式：aac，pcm均可。）

视频活体检测接口 返回数据参数详情

字段	必选	类型	说明
score	是	float	活体检测分数
thresholds	是	array	阈值参考，实际业务应用中，请以score>阈值判定通过，可直接选择不同误识别率的阈值，无需对应具体的分值，选择阈值参数即可。
code	是	array	语音校验码信息
create	是	string	生成的校验码，通过create和identify两个字段的对比，可以判断上传的视频是否为目标视频。
identify	是	string	语音识别出来的校验码
pic_list	是	array	抽取图片信息列表
pic_list[i].face_id	是	string	face唯一ID
pic_list[i].pic	是	string/encryption	base64编码后的图片信息

视频活体检测接口 返回示例

```
{
 err_no:0,
 err_msg: 'success',
 result: {
  score: 0.984654366,
  thresholds: {
    "frr_1e-4": 0.05, //万分之一误识别率的阈值
    "frr_1e-3": 0.3,  //千分之一误识别率的阈值
    "frr_1e-2": 0.9   //百分之一误识别率的阈值
    },
    code: {
      "create": "5789",
      "identify": "5789"
    },
    pic_list: [
      {
        "face_id": 5745745747,
        "pic": "gsagaheryzxv..."
      },
      {
        "face_id": 5745745747,
        "pic": "gsagaheryzxv..."
      }
    ]
 },
 "timestamp": 1509611848,
 "cached": 0,
 "serverlogid": "2248375729"
}
```

人脸对比

接口能力

- **两张人脸图片相似度对比**：比对两张图片中人脸的相似度，并返回相似度分值；
- **多种图片类型**：支持**生活照**、**证件照**、**身份证芯片照**、**带网纹照**四种类型的人脸对比；
- **活体检测**：基于图片中的破绽分析，判断其中的人脸是否为**二次翻拍**（举例：如用户A用手机拍摄了一张包含人脸的图片

一，用户B翻拍了图片一得到了图片二，并用图片二伪造成用户A去进行识别操作，这种情况普遍发生在金融开户、实名认证等环节。）；

- **质量检测**：返回模糊、光照等质量检测信息，用于辅助判断图片是否符合识别要求；

业务应用

用于比对多张图片中的人脸相似度并返回两两比对的得分，可用于判断两张脸是否是同一人的可能性大小。

典型应用场景：如**人证合一验证**，**用户认证**等，可与您现有的人脸库进行比对验证。

```
public void sample(AipFace client) {
  String image1 = "base64_1";
  String image2 = "base64_2";

  // image1/image2也可以为url或facetoken, 相应的imageType参数需要与之对应。
  MatchRequest req1 = new MatchRequest(image1, "BASE64");
  MatchRequest req2 = new MatchRequest(image2, "BASE64");
  ArrayList<MatchRequest> requests = new ArrayList<MatchRequest>();
  requests.add(req1);
  requests.add(req2);

  JSONObject res = client.match(requests);
  System.out.println(res.toString(2));

}
```

请求参数

参数	必选	类型	说明
image	是	string	图片信息(**总数据大小应小于10M**)，图片上传方式根据image_type来判断。 **两张图片通过json格式上传，格式参考表格下方示例**
image_type	是	string	图片类型 **BASE64**:图片的base64值，base64编码后的图片数据，需urlencode，编码后的图片大小不超过2M；**URL**:图片的 URL地址(可能由于网络等原因导致下载图片时间过长)；**FACE_TOKEN**: 人脸图片的唯一标识，调用人脸检测接口时，会为每个人脸图片赋予一个唯一的FACE_TOKEN，同一张图片多次检测得到的FACE_TOKEN是同一个。
face_type	否	string	人脸的类型 **LIVE**表示生活照：通常为手机、相机拍摄的人像图片、或从网络获取的人像图片等，**IDCARD**表示身份证芯片照：二代身份证内置芯片中的人像照片， **WATERMARK**表示带水印证件照：一般为带水印的小图，如公安网小图 **CERT**表示证件照片：如拍摄的身份证、工卡、护照、学生证等证件图片 默认LIVE
quality_control	否	string	图片质量控制 **NONE**: 不进行控制 **LOW**:较低的质量要求 **NORMAL**: 一般的质量要求 **HIGH**: 较高的质量要求 **默认 NONE**
liveness_control	否	string	活体检测控制 **NONE**: 不进行控制 **LOW**:较低的活体要求(高通过率 低攻击拒绝率) **NORMAL**: 一般的活体要求(平衡的攻击拒绝率, 通过率) **HIGH**: 较高的活体要求(高攻击拒绝率 低通过率) **默认NONE**

返回参数

参数名	必选	类型	说明
score	是	float	人脸相似度得分
face_list	是	array	人脸信息列表
+face_token	是	string	人脸的唯一标志

- 返回示例

```
{
 "score": 44.3,
 "face_list": [  //返回的顺序与传入的顺序保持一致
    {
       "face_token": "fid1"
    },
    {
       "face_token": "fid2"
    }
 ]
}
```

在线活体检测

人脸基础信息，人脸质量检测，基于图片的活体检测

```
public void sample(AipFace client) {
  String image = "image_base64_content";
  FaceVerifyRequest req = new FaceVerifyRequest(image, "BASE64");
  ArrayList<FaceVerifyRequest> list = new ArrayList<FaceVerifyRequest>();
  list.add(req);
  JSONObject res = client.faceverify(list);
  System.out.println(res.toString(2));
}
```

在线活体检测 请求参数详情

参数名称	是否必选	类型	说明
image	是	String	图片信息(**总数据大小应小于10M**)，图片上传方式根据image_type来判断
image_type	是	String	图片类型 **BASE64**:图片的base64值，base64编码后的图片数据，需urlencode，编码后的图片大小不超过2M；**URL**:图片的 URL地址(可能由于网络等原因导致下载图片时间过长)；**FACE_TOKEN**:人脸图片的唯一标识，调用人脸检测接口时，会为每个人脸图片赋予一个唯一的FACE_TOKEN，同一张图片多次检测得到的FACE_TOKEN是同一个
face_fields	否	String	包括age,expression,faceshape,gender,glasses,landmark,quality,facetype信息，逗号分隔，默认只返回face_token、活体数、人脸框、概率和旋转角度。

在线活体检测 返回数据参数详情

参数	类型	是否必须	说明
log_id	是	uint64	请求唯一标识码，随机数
face_liveness	是	float	活体分数值
thresholds	是	array	由服务端返回最新的阈值数据（随着模型的优化，阈值可能会变化），可以作为活体判断的依据。frr_1e-4：万分之一误识率的阈值；frr_1e-3：千分之一误识率的阈值；frr_1e-2：百分之一误识率的阈值。误识率越底，准确率越高。
face_list	是	array	每张图片的详细信息描述，如果只上传一张图片，则只返回一个结果。

在线活体检测 返回示例

```
{
```

```
    "thresholds": {
        "frr_1e-4": 0.05,
        "frr_1e-3": 0.3,
        "frr_1e-2": 0.9
    },
    "face_liveness": 0.05532243927,
    "face_list": [
        {
            "face_token": "df46f7c7db4aa09a093c26fb8d1a8d44",
            "location": {
                "left": 328.9026489,
                "top": 97.16340637,
                "width": 162,
                "height": 154,
                "rotation": 32
            },
            "face_probability": 1,
            "angle": {
                "yaw": 10.16196251,
                "pitch": 2.244354248,
                "roll": 33.82199097
            },
            "liveness": {
                "faceliveness": 0.004187555984,
                "livemapscore": 0.04492170034
            },
            "age": 23
        },
        {
            "face_token": "901d2c64274fccd687d311a6e6110a01",
            "location": {
                "left": 411.4876404,
                "top": 166.3593445,
                "width": 329,
                "height": 308,
                "rotation": 45
            },
            "face_probability": 0.9194830656,
            "angle": {
                "yaw": -1.716423035,
                "pitch": 7.344647408,
                "roll": 45.79914856
            },
            "liveness": {
                "faceliveness": 0.0001665892196,
                "livemapscore": 0.001787073661
            },
            "age": 23
        },
        {
            "face_token": "7d57e36981c48b4946eb97c8d838b02a",
            "location": {
                "left": 161.4559937,
                "top": 199.8726501,
                "width": 218,
                "height": 201,
                "rotation": -1
            },
            "face_probability": 1,
            "angle": {
                "yaw": -8.187754631,
                "pitch": 6.973727226,
                "roll": -1.25429821
```

```
            "roll": -1.25429821
        },
        "liveness": {
            "faceliveness": 0.02942637168,
            "livemapscore": 0.05532243927
        },
        "age": 23
    }
  ]
}
```


人脸实名认证V4

能力介绍

1. 业务能力

- **质量检测（可选）**：判断图片中是否包含人脸，以及人脸在姿态、遮挡、模糊、光照等方面是否符合识别条件。
- **活体检测（可选）**：基于图片中的破绽分析，判断其中的人脸是否为**二次翻拍**（举例：如用户A用手机拍摄了一张包含人脸的图片一，用户B翻拍了图片一得到了图片二，并用图片二伪造成用户A去进行识别操作，这种情况普遍发生在金融开户、实名认证等环节）。
- **图片加密及风控（可选）**：当配合增强级采集SDK、增强级安全加固采集SDK、金融级采集SDK、金融级安全加固采集SDK版本使用，对采集SDK输出的加密图片进行解密（加密传输可以有效避免第三方非法黑产绕过APP模拟请求攻击云端接口的行为，Eg：脚本攻击等）；以及结合百度安全实验室大数据风控能力，对采集SDK的发起端设备进行风控识别，辨别是否为风险设备，Eg：ROM注入、视频劫持等；
- **人脸实名认证（必选）**：基于姓名和身份证号，调取公安权威数据源人脸图，将当前获取的人脸图片，与公安数据源人脸图进行对比，得出比对分数，并基于此进行业务判断是否为同一人。由于公安数据源人脸图具有最权威的身份证明作用，故对用户本人的验证结果可信度也最为合理。

2. 业务逻辑

- 上述能力，人脸实名认证能力为必选能力，质量检测、活体检测、图片加密及风控为可选能力，验证顺序为`人脸质量检测->活体检测->人脸实名认证`。
- **如选择了质量检测和活体检测能力，则有任意一个条件不通过，整个请求流程终止，并返回错误码，描述具体的不符合信息**。
- 基于此顺序串行验证逻辑，可以避免大量不符合条件的请求流转到人脸实名认证，节约您的成本。

3. 推荐阈值

- 此接口使用的对比算法，针对带水纹证件照采用了专项的模型处理，可保证水纹信息的影响降到尽可能低。
- 如比对成功，最终返回的有效数据为一个**对比分值**，在0~100之间，您可以设定具体的阈值来判断是否验证通过。
- **人证相似度的推荐阈值为80，对应的误识率为万分之一**。

```
public void sample(AipFace client) {

    // 必填参数
    String idCardNumber = "";
    String name = "";
    String image = "";

    // 选填参数
    HashMap<String, Object> options = new HashMap<String, Object>();
    options.put("xxx", "xxx");

    // 调用接口
    JSONObject res = client.faceMingJingVerify(idCardNumber, name, image, options);
    System.out.println(res.toString(2));
}
```

请求参数

参数	必选	类型	说明
app	否	string	APP端类型，配合采集SDK使用时须传入 **ios**：iOS端采集SDK **android**：安卓端采集SDK
sec_level	否	string	SDK安全级别，配合采集SDK使用时须传入，默认common **common**：配合4.1/4.1.5版本采集SDK，人脸图片未进行加密处理 **lite**: 配合5.2版本SDK
skey	否	string	使用5.2版本SDK请求时必填 **skey**：从SDK获取的密钥信息skey
x_device_id	否	string	使用5.2版本SDK请求时必填 **deviceId**：从SDK 获取的密钥信息deviceId
data	否	string	使用5.2版本SDK请求时必填， SDK输出的加密数据
id_card_number	是	string	身份证件号
name	是	string	姓名(需要是 utf8 编码)
liveness_control	否	string	活体控制参数 **NONE**: 不进行控制 **LOW**:较低的活体要求(高通过率 低攻击拒绝率) **NORMAL**: 一般的活体要求(平衡的攻击拒绝率, 通过率) **HIGH**: 较高的活体要求(高攻击拒绝率 低通过率) **默认为NONE**
spoofing_control	否	string	合成图控制参数 **NONE**: 不进行控制 **LOW**:较低的合成图阈值数值，由于合成图判定逻辑为大于阈值视为合成图攻击，该项代表**低通过率、高攻击拒绝率** **NORMAL**: 一般的合成图阈值数值，由于合成图判定逻辑为大于阈值视为合成图攻击，该项代表**平衡的攻击拒绝率, 通过率** **HIGH**: 较高的合成图阈值数值，由于合成图判定逻辑为大于阈值视为合成图攻击，该项代表**高通过率、低攻击拒绝率**) **默认为NONE**
quality_control	否	string	质量控制参数 **NONE**: 不进行控制 **LOW**:较低的质量要求 **NORMAL**: 一般的质量要求 **HIGH**: 较高的质量要求 **默认为NONE**
image	是	string	图片信息(数据大小应小于10M 分辨率应小于1920*1080)，**5.2版本SDK请求时已包含在加密数据data中，无需额外传入**
image_type	否	string	图片类型 **BASE64**：图片的base64值 **URL**：图片的 URL **FACE_TOKEN**：人脸标识 **默认 BASE64**

返回参数

参数	类型	说明
log_id	number	调用的日志id
result	jsonObject	认证返回的结果
+score	float	与公安数据源人脸图相似度可能性，用于验证生活照与公安数据源人脸图是否为同一人，有正常分数时为[0~100]，推荐阈值80，超过即判断为同一人
+verify_status	number	认证状态，取值如下： 0 ： 正常 1 ： 身份证号与姓名不匹配或该身份证号不存在 2 ： 公安网图片不存在或质量过低
dec_image	string	对SDK传入的加密图片进行解密。 仅APP场景且进行了图片加密时，此参数返回解密后的人脸图片信息
risk_level	string	判断设备是否发生过风险行为来判断风险级别，取值（数值由高到低）： 1 – 高危 2 – 嫌疑 3 – 普通 4 – 正常
risk_tag	string	风险标签，若判断为有风险，则会有风险标签json 数组告知风险类型 例如：general_inject

返回示例


```
{
  "log_id": 1370579072568000512,
  "result": {
     "score": 40.884,
     "verify_status": 0
  },
  "dec_image": "/9j/4AAQSkZJRgABAgAAAQABAAD",
  "risk_level": "3",
  "risk_tag": [
     "若判断为有风险，则会有风险标签json 数组告知风险类型，如：general_inject"
  ]
}
```


人脸比对V4

能力介绍

1. 接口能力

- **两张人脸图片相似度对比**：对比两张图片中人脸的相似度，并返回相似度分值；
- **多种图片类型**：支持**生活照**、**证件照**、**身份证芯片照**、**带网纹照**四种类型的人脸对比；
- **活体检测控制**：基于图片中的破绽分析，判断其中的人脸是否为**二次翻拍**（举例：如用户A用手机拍摄了一张包含人脸的图片一，用户B翻拍了图片一得到了图片二，并用图片二伪造成用户A去进行识别操作，这种情况普遍发生在金融开户、实名认证等环节。）；
- **质量检测控制**：分析图片的中人脸的模糊度、角度、光照强度等特征，判断图片质量；
- **图片加密及风控：**配合增强级、金融级采集SDK使用
  - 对采集SDK输出的加密图片进行解密（加密传输可以有效避免第三方非法黑产**绕过APP模拟请求攻击云端接口的行为**，Eg：脚本攻击等）；
  - 以及结合百度安全实验室大数据风控能力，对采集SDK的发起端设备进行风控识别，辨别是否为**风险设备**，Eg：ROM注入、视频劫持等；

2. 业务应用

- 用于对比多张图片中的人脸相似度并返回两两对比的得分，可用于判断两张脸是否是同一人的可能性大小。

- 典型应用场景：如**人证合一验证**，**用户认证**等，可与您现有的人脸库进行对比验证。

```
public void sample(AipFace client) {

  // 必填参数
  String image = "";
  String imageType = "";
  String registerImage = "";
  String registerImageType = "";

  // 选填参数
  HashMap<String, Object> options = new HashMap<String, Object>();
  options.put("xxx", "xxx");

  // 调用接口
  JSONObject res = client.faceMingJingMatch(image, imageType, registerImage, registerImageType, null);
  System.out.println(res.toString(2));
}
```

请求参数

参数	必选	类型	说明
app	否	string	APP类型，配合采集SDK使用时须传入 **ios**：iOS端采集SDK **android**：安卓端采集SDK
sec_level	否	string	SDK安全级别，配合采集SDK使用时须传入，默认common **common**：配合4.1/4.1.5版本SDK使用，人脸图片未进行加密处理 **lite**：配合5.2版本SDK使用
skey	否	string	使用5.2版本SDK请求时必填 从SDK获取的密钥信息
x_device_id	否	string	使用5.2版本SDK请求时必填 从SDK获取的密钥信息
data	否	string	使用5.2版本SDK请求时必填 SDK输出的加密数据
image	是	string	图片信息(数据大小应小于10M 分辨率应小于1920*1080)，***5.2版本SDK请求时已包含在加密数据data中，无需额外传入**
image_type	是	string	图片类型 **BASE64**：图片的base64值 **URL**：图片的URL **FACE_TOKEN**：人脸标识 默认 BASE64
face_type	否	string	人脸的类型 **LIVE**：表示生活照：通常为手机、相机拍摄的人像图片、或从网络获取的人像图片等 **IDCARD**：表示身份证芯片照：二代身份证内置芯片中的人像照片 **WATERMARK**：表示带水印证件照：一般为带水印的小图，如公安网小图 **CERT**：表示证件照片：如拍摄的身份证、工卡、护照、学生证等证件图片 **INFRARED** 表示红外照片：使用红外相机拍摄的照片 默认LIVE
quality_control	否	string	质量控制 **NONE**: 不进行控制 **LOW**:较低的质量要求 **NORMAL**: 一般的质量要求 **HIGH**: 较高的质量要求 **默认NONE**
liveness_control	否	string	活体控制 **NONE**: 不进行控制 **LOW**:较低的活体要求(高通过率 低攻击拒绝率) **NORMAL**: 一般的活体要求(平衡的攻击拒绝率, 通过率) **HIGH**: 较高的活体要求(高攻击拒绝率 低通过率) **默认NONE**
registerimage	是	string	图片信息(总数据大小应小于10M)，图片上传方式根据imagetype来判断。本图片特指客户服务器上传图片，非加密图片Base64值
registerimagetype	是	string	图片类型 **BASE64**：图片的base64值 **URL**：图片的 URL **FACE_TOKEN**：人脸标识 默认 BASE64
registerfacetype	否	string	人脸的类型
registerqualitycontrol	否	string	图片质量控制 **NONE**: 不进行控制 **LOW**:较低的质量要求 **NORMAL**: 一般的质量要求 **HIGH**: 较高的质量要求 **默认 NONE** 若图片质量不满足要求，则返回结果中会提示质量检测失败
registerlivenesscontrol	否	string	活体检测控制 **NONE**: 不进行控制 **LOW**:较低的活体要求(高通过率 低攻击拒绝率) **NORMAL**: 一般的活体要求(平衡的攻击拒绝率, 通过率) **HIGH**: 较高的活体要求(高攻击拒绝率 低通过率) **默认 NONE** 若活体检测结果不满足要求，则返回结果中会提示活体检测失败
facesorttype	否	int	人脸检测排序类型 0:代表检测出的人脸按照人脸面积从大到小排列 1:代表检测出的人脸按照距离图片中心从近到远排列 默认为0

返回参数

参数名	类型	说明
log_id	number	调用的日志id
result	jsonObject	认证返回的结果
+ score	number	人脸相似度得分，推荐阈值80分
+ face_list	jsonArray	人脸信息列表
++ face_token	string	人脸标志
dec_image	string	APP场景传入加密图片时，该项返回解密后的图片
risk_level	string	风控返回参数，只有在 risk_identify 为 true 时才返回，判断设备是否发生过风险行为来判断风险级别，取值（数值有高到低）： 1 – 高危 2 – 嫌疑 3 – 普通 4 – 正常
risk_tag	jsonArray	风控返回参数，只有在 risk_identify 为 true 时才返回，风险标签，若判断为有风险，则会有风险标签 json 数组告知风险类型
risk_warn_code	number	风控返回参数，只有在 risk_identify 为 true 时才返回，只有在风控服务异常的时候才返回这个字段

返回示例


```
{
  "log_id": 1370585066551377920,
  "result": {
    "score": 99.06919861,
    "face_list": [
      {
        "face_token": "549f9f1d1c7ec8c86931540b1939e8ed"
      },
      {
        "face_token": "1a319460ef89e8d27fb59062a28dbad7"
      }
    ]
  },
  "dec_image": "/9j/4AAQSkZJRgABAQAAAQABAAD",
  "risk_level": "3",
  "risk_tag": [
    "空"
  ]
}
```


在线图片活体V4

能力介绍

1. 接口能力

- **人脸基础信息**：包括人脸框位置，人脸空间旋转角度，人脸置信度等信息。
- **人脸质量检测**：判断人脸的遮挡、光照、模糊度、完整度等质量信息。可用于判断上传的人脸是否符合标准。
- **基于图片的活体检测**：基于单张图片，判断图片中的人脸是否为二次翻拍（举例：如用户A用手机拍摄了一张包含人脸的图片一，用户B翻拍了图片一得到了图片二，并用图片二伪造成用户A去进行识别操作，这种情况普遍发生在金融开户、实名

认证等环节）以及是否为合成图攻击。此能力可用于H5场景下的一些人脸采集场景中，增加人脸注册的安全性和真实性。

- **图片加密及风控**：配合采集SDK5.0版本使用，对采集SDK输出的加密图片进行解密（加密传输可以有效避免第三方非法黑产绕过APP模拟请求攻击云端接口的行为，Eg：脚本攻击等）；以及结合百度安全实验室大数据风控能力，对采集SDK的发起端设备进行风控识别，辨别是否为风险设备，Eg：ROM注入、视频劫持等；

```
public void sample(AipFace client) {

 // 必填参数
  String sdkVersion = "";

  // 选填参数
  HashMap<String, Object> options = new HashMap<String, Object>();
  options.put("xxx", "xxx");

  // 调用接口
  JSONObject res = client.onlinePictureLiveV4(sdkVersion, options);
  System.out.println(res.toString(2));
}
```

请求参数

参数	是否必选	类型	说明
sdk_version	是	string	1 ：非加密图片，适用于4.1/4.1.5版本SDK、H5场景或纯服务端场景 4：适用于5.2版本SDK
face_field	否	string	包括**age,expression,face_shape,gender,glasses,landmark,quality,face_type,spoofing**信息，逗号分隔，默认只返回face_token、活体数、人脸框、概率和旋转角度
option	否	string	场景信息，程序会视不同的场景选用相对应的模型。当前支持的场景有COMMON(通用场景)，GATE(闸机场景)，FINANCE(金融场景)，LOGISTICS(物流场景)，INTERNET(泛互联网场景), **默认使用COMMON** 。注意：如果请求参数中存在多个option值时，则取第一个option的值，同时除通用场景外的其他场景需要联系工作人员对您所使用的appid进行配置
app	否	string	端类型 ios / android **5.2版本SDK必传该项**
s_key	否	string	端上提供的用于解密图片的skey **5.2版本SDK必传该项**
device_id	否	string	端上提供的用于解密图片的deviceId **5.2版本SDK必传该项**
data	否	string	端上提供的加密后的图片数组 **5.2版本SDK必传该项**
image_list	否	array	图片BASE64数组 4.1/4.1.5版本SDK、H5场景或纯服务端场景必填

返回参数

参数	是否必须	类型	说明
log_id	是	string	日志id
error_code	是	int	错误码状态，若为0则表示认证成功
error_msg	是	string	错误码说明，若为 success 则表示认证成功
result	是	object	活体结果
+face_li			

veness	是	float	活体分数值
+thresholds	是	array	由服务端返回最新的阈值数据（随着模型的优化，阈值可能会变化），将此参数与返回的face_liveness进行比较，可以作为活体判断的依据。 frr_1e-4：万分之一误识率的阈值； frr_1e-3：千分之一误识率的阈值； frr_1e-2：百分之一误识率的阈值。 误识率越低，准确率越高，相应的拒绝率也越高
+face_list	是	array	每张图片的详细信息描述，如果只上传一张图片，则只返回一个结果。
++face_token	是	string	人脸图片的唯一标识
++location	是	array	人脸在图片中的位置
+++left	是	double	人脸区域离左边界的距离
+++top	是	double	人脸区域离上边界的距离
+++width	是	double	人脸区域的宽度
+++height	是	double	人脸区域的高度
+++rotation	是	int64	人脸框相对于竖直方向的顺时针旋转角，[-180,180]
++face_probability	是	double	人脸置信度，范围【0~1】，代表这是一张人脸的概率，0最小、1最大。
++angel	是	array	人脸旋转角度参数
+++yaw	是	double	三维旋转之左右旋转角[-90(左), 90(右)]
+++pitch	是	double	三维旋转之俯仰角度[-90(上), 90(下)]
+++roll	是	double	平面内旋转角[-180(逆时针), 180(顺时针)]
++age	否	double	年龄 ，当**face_field包含age时返回**
++expression	否	array	表情，当 **face_field包含expression时返回**
+++type	否	string	**none**:不笑；**smile**:微笑；**laugh**:大笑
+++probability	否	double	表情置信度，范围【0~1】，0最小、1最大。
++face_shape	否	array	脸型，当**face_field包含face_shape时返回**
+++type	否	double	**square**: 正方形 **triangle**:三角形 **oval**: 椭圆 **heart**: 心形 **round**: 圆形
+++probability	否	double	置信度，范围【0~1】，代表这是人脸形状判断正确的概率，0最小、1最大。
++gender	否	array	性别，**face_field包含gender时返回**
+++typ			

e	否	string	male:**男性** female:**女性**
+++probability	否	double	性别置信度，范围【0~1】，0代表概率最小、1代表最大。
++glasses	否	array	是否带眼镜，**face_field包含glasses时返回**
++type	否	string	**none**:无眼镜，**common**:普通眼镜，**sun**:墨镜
+++probability	否	double	眼镜置信度，范围【0~1】，0代表概率最小、1代表最大。
++face_type	否	array	真实人脸/卡通人脸 **face_field包含face_type时返回**
+++type	否	string	**human**: 真实人脸 **cartoon**: 卡通人脸
+++probability	否	double	人脸类型判断正确的置信度，范围【0~1】，0代表概率最小、1代表最大。
++landmark	否	array	4个关键点位置，左眼中心、右眼中心、鼻尖、嘴中心。**face_field包含landmark时返回**
++landmark72	否	array	72个特征点位置 **face_field包含landmark时返回**
++quality	否	array	人脸质量信息。**face_field包含quality时返回**
+++occlusion	否	array	人脸各部分遮挡的概率，范围[0~1]，0表示完整，1表示不完整
++++left_eye	否	double	左眼遮挡比例，[0-1] ， 1表示完全遮挡
++++right_eye	否	double	右眼遮挡比例，[0-1] ， 1表示完全遮挡
++++nose	否	double	鼻子遮挡比例，[0-1] ， 1表示完全遮挡
++++mouth	否	double	嘴巴遮挡比例，[0-1] ， 1表示完全遮挡
++++left_cheek	否	double	左脸颊遮挡比例，[0-1] ， 1表示完全遮挡
++++right_cheek	否	double	右脸颊遮挡比例，[0-1] ， 1表示完全遮挡
++++chin	否	double	下巴遮挡比例， ，[0-1] ， 1表示完全遮挡
+++blur	否	double	人脸模糊程度，范围[0~1]，0表示清晰，1表示模糊
+++illumination	否	double	取值范围在[0~255], 表示脸部区域的光照程度 越大表示光照越好
+++completeness	否	int64	人脸完整度，0或1, 0为人脸溢出图像边界，1为人脸都在图像边界内

++spoofing	否	double	合成图打分 判断图片是否为合成图 face_field包含时返回spoofing
risk_level	否	string	判断设备是否发生过风险行为来判断风险级别，取值（数值有高到低）：1 – 高危 2 – 嫌疑 3 – 普通 4 – 正常
risk_tag	否	string	风险标签，若判断为有风险，则会有风险标签 json 数组告知风险类型
dec_image	否	array	非安全加固增强级采集sdk、非安全加固金融级采集sdk、安全加固增强级采集sdk、安全加固金融级采集sdk会返回解密后的原图

返回示例


```
{
  "result": {
    "thresholds": {
      "frr_1e-3": 0.3,
      "frr_1e-2": 0.9,
      "frr_1e-4": 0.05
    },
    "face_liveness": 0.1976952702,
    "face_list": [
      {
        "liveness": {
          "livemapscore": 0.1976952702
        },
        "angle": {
          "roll": 0.31,
          "pitch": -2.07,
          "yaw": 0.85
        },
        "face_token": "abd1b4ce743099336cfed40193ff4944",
        "location": {
          "top": 161.74,
          "left": 79.04,
          "rotation": 1,
          "width": 318,
          "height": 333
        },
        "face_probability": 1
      },
      {
        "liveness": {
          "livemapscore": 0.1976952702
        },
        "angle": {
          "roll": 0.31,
          "pitch": -2.07,
          "yaw": 0.85
        },
        "face_token": "abd1b4ce743099336cfed40193ff4944",
        "location": {
          "top": 161.74,
          "left": 79.04,
          "rotation": 1,
          "width": 318,
          "height": 333
        },
        "face_probability": 1
      },
      {
        "liveness": {
```

```
            "livemapscore": 0.1976952702
        },
        "angle": {
            "roll": 0.31,
            "pitch": -2.07,
            "yaw": 0.85
        },
        "face_token": "abd1b4ce743099336cfed40193ff4944",
        "location": {
            "top": 161.74,
            "left": 79.04,
            "rotation": 1,
            "width": 318,
            "height": 333
        },
        "face_probability": 1
    },
    {
        "liveness": {
            "livemapscore": 0.1976952702
        },
        "angle": {
            "roll": 0.31,
            "pitch": -2.07,
            "yaw": 0.85
        },
        "face_token": "abd1b4ce743099336cfed40193ff4944",
        "location": {
            "top": 161.74,
            "left": 79.04,
            "rotation": 1,
            "width": 318,
            "height": 333
        },
        "face_probability": 1
    },
    {
        "liveness": {
            "livemapscore": 0.1976952702
        },
        "angle": {
            "roll": 0.31,
            "pitch": -2.07,
            "yaw": 0.85
        },
        "face_token": "abd1b4ce743099336cfed40193ff4944",
        "location": {
            "top": 161.74,
            "left": 79.04,
            "rotation": 1,
            "width": 318,
            "height": 333
        },
        "face_probability": 1
    },
    {
        "liveness": {
            "livemapscore": 0.1976952702
        },
        "angle": {
            "roll": 0.31,
            "pitch": -2.07,
            "yaw": 0.85
```

```
                "yaw": 0.85
            },
            "face_token": "abd1b4ce743099336cfed40193ff4944",
            "location": {
                "top": 161.74,
                "left": 79.04,
                "rotation": 1,
                "width": 318,
                "height": 333
            },
            "face_probability": 1
        },
        {
            "liveness": {
                "livemapscore": 0.1976952702
            },
            "angle": {
                "roll": 0.31,
                "pitch": -2.07,
                "yaw": 0.85
            },
            "face_token": "abd1b4ce743099336cfed40193ff4944",
            "location": {
                "top": 161.74,
                "left": 79.04,
                "rotation": 1,
                "width": 318,
                "height": 333
            },
            "face_probability": 1
        },
        {
            "liveness": {
                "livemapscore": 0.1976952702
            },
            "angle": {
                "roll": 0.31,
                "pitch": -2.07,
                "yaw": 0.85
            },
            "face_token": "abd1b4ce743099336cfed40193ff4944",
            "location": {
                "top": 161.74,
                "left": 79.04,
                "rotation": 1,
                "width": 318,
                "height": 333
            },
            "face_probability": 1
        }
    ]
},
"risk_level": "3",
"log_id": 1423545654699779516,
"risk_tag": [
    "空"
],
"dec_image": [
        "/9j/4AAQ..."
]
}
```

错误信息

错误返回格式

若请求错误，服务器将返回的JSON文本包含以下参数：

- **error_code**：错误码。
- **error_msg**：错误描述信息，帮助理解和解决发生的错误。

错误码

SDK本地检测参数返回的错误码：

error_code	error_msg	备注
SDK100	image size error	图片大小超限
SDK101	image length error	图片边长不符合要求
SDK102	read image file error	读取图片文件错误
SDK108	connection or read data time out	连接超时或读取数据超时
SDK109	unsupported image format	不支持的图片格式

服务端返回的错误码

错误码	错误信息	描述
1	Unknown error	服务器内部错误，请再次请求， 如果持续出现此类错误，请通过QQ群（860337848）或提交工单联系技术支持团队。
2	Service temporarily unavailable	服务暂不可用，请再次请求， 如果持续出现此类错误，请通过QQ群（860337848）或提交工单联系技术支持团队。
3	Unsupported openapi method	调用的API不存在，请检查请求URL后重新尝试，一般为URL中有非英文字符，如“-”，可手动输入重试
4	Open api request limit reached	集群超限额，请再次请求， 如果持续出现此类错误，请通过QQ群（860337848）或提交工单联系技术支持团队。
6	No permission to access data	无权限访问该用户数据，创建应用时未勾选相关接口
13	Get service token failed	获取token失败
14	IAM Certification failed	IAM 鉴权失败
15	app not exsits or create failed	应用不存在或者创建失败
17	Open api daily request limit reached	每天请求量超限额，可通过QQ群（860337848）联系群管、提交工单提升限额
18	Open api qps request limit reached	QPS超限额，可通过QQ群（860337848）联系群管、提交工单提升限额
19	Open api total request limit reached	请求总量超限额，可通过QQ群（860337848）联系群管、提交工单提升限额
100	Invalid parameter	无效的access_token参数，请检查后重新尝试
110	Access token invalid or no longer valid	access_token无效
111	Access token expired	access token过期
222001	param[] is null	必要参数未传入

222001	param[] is null	必要参数未传入
222002	param[start] format error	参数格式错误
222003	param[length] format error	参数格式错误
222004	param[op_app_id_list] format error	参数格式错误
222005	param[group_id_list] format error	参数格式错误
222006	group_id format error	参数格式错误
222007	uid format error	参数格式错误
222008	face_id format error	参数格式错误
222009	quality_conf format error	参数格式错误
222010	user_info format error	参数格式错误
222011	param[uid_list] format error	参数格式错误
222012	param[op_app_id] format error	参数格式错误
222013	param[image] format error	参数格式错误
222014	param[app_id] format error	参数格式错误
222015	param[image_type] format error	参数格式错误
222016	param[max_face_num] format error	参数格式错误
222017	param[face_field] format error	参数格式错误
222018	param[user_id] format error	参数格式错误
222019	param[quality_control] format error	参数格式错误
222020	param[liveness_control] format error	参数格式错误
222021	param[max_user_num] format error	参数格式错误
222022	param[id_card_number] format error	参数格式错误
222023	param[name] format error	参数格式错误
222024	param[face_type] format error	参数格式错误
222025	param[face_token] format error	参数格式错误
222026	param[max_star_num]	参数格式错误

	format error	
222201	network not available	服务端请求失败
222202	pic not has face	图片中没有人脸
222203	image check fail	无法解析人脸
222204	image_url_download_fail	从图片的url下载 图片失败
222205	network not availablel	服务端请求失败
222206	rtse service return fail	服务端请求失败
222207	match user is not found	未找到匹配的用户
222208	the number of image is incorrect	图片的数量错误
222209	face token not exist	face token不存在
222300	add face fail	人脸图片添加失败
222301	get face fail	获取人脸图片失败
222302	system error	服务端请求失败
222303	get face fail	获取人脸图片失败
223100	group is not exist	操作的用户组不存在
223101	group is already exist	该用户组已存在
223102	user is already exist	该用户已存在
223103	user is not exist	找不到该用户
223104	group_list is too large	group_list包含组 数量过多
223105	face is already exist	该人脸已存在
223106	face is not exist	该人脸不存在
223110	uid_list is too large	uid_list包含数量过多
223111	dst group is not exist	目标用户组不存在
223112	quality_conf format error	quality_conf格式不正确
223113	face is covered	人脸有被遮挡
223114	face is fuzzy	人脸模糊
223115	face light is not good	人脸光照不好
223116	incomplete face	人脸不完整
223117	app_list is too large	app_list包含app数量 过多
223118	quality control error	质量控制项错误
223119	liveness control item error	活体控制项错误
223120	liveness check fail	活体检测未通过
223121	left eye is occlusion	质量检测未通过 左眼 遮挡程度过高
		质量检测未通过 右眼

223122	right eye is occlusion	遮挡程度过高
223123	left cheek is occlusion	质量检测未通过 左脸 遮挡程度过高
223124	right cheek is occlusion	质量检测未通过 右脸 遮挡程度过高
223125	chin contour is occlusion	质量检测未通过 下巴遮挡程度过高
223126	nose is occlusion	质量检测未通过 鼻子遮挡程度过高
223127	mouth is occlusion	质量检测未通过 嘴巴 遮挡程度过高
222350	police picture is none or low quality	公安网图片不存在或 质量过低
222351	id number and name not match or id number not exist	身份证号与姓名不匹配或该 身份证号不存在
222352	name format error	身份证名字格式错误
222353	id number format error	身份证号码格式错误
222354	id number not exist	公安库里不存在此身份证号
222355	police picture not exist	身份证号码正确，公安库里没有 对应的照片
222360	invalid name or id number	身份证号码或名字非法（公安网校 验不通过）
222901	system busy	系统繁忙
222902	system busy	系统繁忙
222903	system busy	系统繁忙
222904	system busy	系统繁忙
222905	system busy	系统繁忙
222906	system busy	系统繁忙
222907	system busy	系统繁忙
222908	system busy	系统繁忙
222909	system busy	系统繁忙
222910	system busy	系统繁忙
222911	system busy	系统繁忙
222912	system busy	系统繁忙
222913	system busy	系统繁忙
222914	system busy	系统繁忙
222915	system busy	系统繁忙
222916	system busy	系统繁忙
222361	system busy	系统繁忙

PHP-SDK

简介

Hi，您好，欢迎使用百度人脸识别服务。

本文档主要针对PHP开发者，描述百度人脸识别接口服务的相关技术内容。如果您对文档内容有任何疑问，可以通过以下几种方式联系我们：

- 在百度云控制台内提交工单，咨询问题类型请选择**人工智能服务**；
- 如有疑问，进入AI社区交流：http://ai.baidu.com/forum/topic/list/165

接口能力

接口名称	接口能力简要描述
人脸检测	检测人脸并定位，返回五官关键点，及人脸各属性值
人脸比对	返回两两比对的人脸相似值
人脸查找	在一个人脸集合中找到找到相似的人脸，由一系列接口组成，包括人脸识别、人脸认证、人脸库管理相关接口（人脸注册、人脸更新、人脸删除、用户信息查询、组列表查询、组内用户列表查询、组间复制用户、组内删除用户）

版本更新记录

上线日期	版本号	更新内容
2019.4.17	2.2.15	人脸v3文档更新，新增N:M接口
2018.5.10	2.2.4	修复人脸V3问题
2018.4.28	2.2.3	全面切换为人脸V3接口
2018.4.9	2.2.2	新增身份验证，在线活体检测接口
2018.01.12	2.1.0	新增M：N多人脸识别
2017.12.22	2.0.0	SDK代码重构
2017.5.11	1.0.0	人脸识别服务上线

快速入门

安装人脸识别 PHP SDK

人脸识别 PHP SDK目录结构

```
├──── AipFace.php          //人脸识别
└──── lib
   ├──── AipHttpClient.php      //内部http请求类
   ├──── AipBCEUtil.php         //内部工具类
   └──── AipBase                //Aip基类
```

支持PHP版本：5.3+

使用PHP SDK开发骤如下：

1.在官方网站下载php SDK压缩包。

2.将下载的`aip-php-sdk-version.zip`解压后，复制AipFace.php以及lib/*到工程文件夹中。

3.引入AipFace.php

新建AipFace

AipFace是人脸识别的PHP SDK客户端，为使用人脸识别的开发人员提供了一系列的交互方法。

参考如下代码新建一个AipFace：

```
 require_once 'AipFace.php';

// 你的 APPID AK SK
const APP_ID = '你的 App ID';
const API_KEY = '你的 Api Key';
const SECRET_KEY = '你的 Secret Key';

$client = new AipFace(APP_ID, API_KEY, SECRET_KEY);
```

在上面代码中，常量APP_ID在百度云控制台中创建，常量API_KEY与SECRET_KEY是在创建完毕应用后，系统分配给用户的，均为字符串，用于标识用户，为访问做签名验证，可在AI服务控制台中的**应用列表**中查看。

注意：如您以前是百度云的老用户，其中API_KEY对应百度云的“Access Key ID”，SECRET_KEY对应百度云的“Access Key Secret”。

配置AipFace

如果用户需要配置AipFace的网络请求参数(一般不需要配置)，可以在构造AipFace之后调用接口设置参数，目前只支持以下参数：

接口	说明
setConnectionTimeoutInMillis	建立连接的超时时间（单位：毫秒)
setSocketTimeoutInMillis	通过打开的连接传输数据的超时时间（单位：毫秒）

接口说明

人脸检测

人脸检测：检测图片中的人脸并标记出位置信息;

```
 $image = "取决于image_type参数，传入BASE64字符串或URL字符串或FACE_TOKEN字符串";

$imageType = "BASE64";

// 调用人脸检测
$client->detect($image, $imageType);

// 如果有可选参数
$options = array();
$options["face_field"] = "age";
$options["max_face_num"] = 2;
$options["face_type"] = "LIVE";
$options["liveness_control"] = "LOW";

// 带参数调用人脸检测
$client->detect($image, $imageType, $options);
```

人脸检测 请求参数详情

参数名称	是否必选	类型	默认值	说明
image	是	string		图片信息(**总数据大小应小于10M**)，图片上传方式根据image_type来判断
image_type	是	string		图片类型 **BASE64**:图片的base64值，base64编码后的图片数据，编码后的图片大小不超过2M；**URL**:图片的 URL地址(可能由于网络等原因导致下载图片时间过长)； **FACE_TOKEN**: 人脸图片的唯一标识，调用人脸检测接口时，会为每个人脸图片赋予一个唯一的FACE_TOKEN，同一张图片多次检测得到的FACE_TOKEN是同一个
face_field	否	string		包括**age,expression,face_shape,gender,glasses,landmark,landmark150,quality,eye_status,emotion,face_type信息** 逗号分隔. 默认只返回face_token、人脸框、概率和旋转角度
max_face_num	否	string	1	最多处理人脸的数目，默认值为1，仅检测图片中面积最大的那个人脸；**最大值10**，检测图片中面积最大的几张人脸。
face_type	否	string		人脸的类型 **LIVE**表示生活照：通常为手机、相机拍摄的人像图片、或从网络获取的人像图片等**IDCARD**表示身份证芯片照：二代身份证内置芯片中的人像照片 **WATERMARK**表示带水印证件照：一般为带水印的小图，如公安网小图 **CERT**表示证件照片：如拍摄的身份证、工卡、护照、学生证等证件图片 默认**LIVE**
liveness_control	否	string	NONE	活体检测控制 **NONE**: 不进行控制 **LOW**:较低的活体要求(高通过率 低攻击拒绝率) **NORMAL**: 一般的活体要求(平衡的攻击拒绝率, 通过率) **HIGH**: 较高的活体要求(高攻击拒绝率 低通过率) **默认NONE**

人脸检测 返回数据参数详情

字段	必选	类型	说明
face_num	是	int	检测到的图片中的人脸数量
face_list	是	array	人脸信息列表，具体包含的参数参考下面的列表。
+face_token	是	string	人脸图片的唯一标识
+location	是	array	人脸在图片中的位置
++left	是	double	人脸区域离左边界的距离
++top	是	double	人脸区域离上边界的距离
++width	是	double	人脸区域的宽度
++height	是	double	人脸区域的高度
++rotation	是	int64	人脸框相对于竖直方向的顺时针旋转角，[-180,180]
+face_probability	是	double	人脸置信度，范围【0~1】，代表这是一张人脸的概率，0最小、1最大。
+angel	是	array	人脸旋转角度参数
++yaw	是	double	三维旋转之左右旋转角[-90(左), 90(右)]
++pitch	是	double	三维旋转之俯仰角度[-90(上), 90(下)]
++roll	是	double	平面内旋转角[-180(逆时针), 180(顺时针)]
+age	否	double	年龄 ，当**face_field包含age时返回**
+expression	否	array	表情，当 **face_field包含expression时返回**
++type	否	string	**none**:不笑；**smile**:微笑；**laugh**:大笑
++probability	否	double	表情置信度，范围【0~1】，0最小、1最大。
+face_shape	否	array	脸型，当**face_field包含face_shape时返回**

++type	否	double	**square**: 正方形 **triangle**:三角形 **oval**: 椭圆 **heart**: 心形 **round**: 圆形
++probability	否	double	置信度，范围【0~1】，代表这是人脸形状判断正确的概率，0最小、1最大。
+gender	否	array	性别，**face_field包含gender时返回**
++type	否	string	male:**男性** female:**女性**
++probability	否	double	性别置信度，范围【0~1】，0代表概率最小、1代表最大。
+glasses	否	array	是否带眼镜，**face_field包含glasses时返回**
++type	否	string	**none**:无眼镜，**common**:普通眼镜，**sun**:墨镜
++probability	否	double	眼镜置信度，范围【0~1】，0代表概率最小、1代表最大。
+eye_status	否	array	双眼状态（睁开/闭合） **face_field包含eye_status时返回**
++left_eye	否	double	左眼状态 [0,1]取值，越接近0闭合的可能性越大
++right_eye	否	double	右眼状态 [0,1]取值，越接近0闭合的可能性越大
+emotion	否	array	情绪 **face_field包含emotion时返回**
++type	否	string	**angry**:愤怒 **disgust**:厌恶 **fear**:恐惧 **happy**:高兴 **sad**:伤心 **surprise**:惊讶 **neutral**:无表情 **pouty**:撅嘴 **grimace**:鬼脸
++probability	否	double	情绪置信度，范围0~1
++probability	否	double	人种置信度，范围【0~1】，0代表概率最小、1代表最大。
+face_type	否	array	真实人脸/卡通人脸 **face_field包含face_type时返回**
++type	否	string	**human**: 真实人脸 **cartoon**: 卡通人脸
++probability	否	double	人脸类型判断正确的置信度，范围【0~1】，0代表概率最小、1代表最大。
+landmark	否	array	4个关键点位置，左眼中心、右眼中心、鼻尖、嘴中心。**face_field包含landmark时返回**
+landmark72	否	array	72个特征点位置 **face_field包含landmark时返回**
+landmark150	否	array	150个特征点位置 **face_field包含landmark150时返回**
+quality	否	array	人脸质量信息。**face_field包含quality时返回**
++occlusion	否	array	人脸各部分遮挡的概率，范围[0~1]，0表示完整，1表示不完整
+++left_eye	否	double	左眼遮挡比例，[0-1] ，1表示完全遮挡
+++right_eye	否	double	右眼遮挡比例，[0-1] ， 1表示完全遮挡
+++nose	否	double	鼻子遮挡比例，[0-1] ， 1表示完全遮挡
+++mouth	否	double	嘴巴遮挡比例，[0-1] ， 1表示完全遮挡
+++left_cheek	否	double	左脸颊遮挡比例，[0-1] ， 1表示完全遮挡
+++right_cheek	否	double	右脸颊遮挡比例，[0-1] ， 1表示完全遮挡
+++chin_contour	否	double	下巴遮挡比例，，[0-1] ， 1表示完全遮挡
++blur	否	double	人脸模糊程度，范围[0~1]，0表示清晰，1表示模糊
++illumination	否	double	取值范围在[0~255], 表示脸部区域的光照程度 越大表示光照越好
++completeness	否	int64	人脸完整度，0或1, 0为人脸溢出图像边界，1为人脸都在图像边界内

人脸检测 返回示例

```
{
```

```
"face_num": 1,
"face_list": [
    {
        "face_token": "35235asfas21421fakghktyfdgh68bio",
        "location": {
            "left": 117,
            "top": 131,
            "width": 172,
            "height": 170,
            "rotation": 4
        },
        "face_probability": 1,
        "angle" :{
            "yaw" : -0.34859421849251
            "pitch" 1.9135693311691
            "roll" :2.3033397197723
        }
        "landmark": [
            {
                "x": 161.74819946289,
                "y": 163.30244445801
            },
            ...
        ],
        "landmark72": [
            {
                "x": 115.86531066895,
                "y": 170.0546875
            } ,
            ...
        ],
        "age": 29.298097610474,
        "expression": {
            "type": "smile",
            "probability" : 0.5543018579483
        },
        "gender": {
            "type": "male",
            "probability": 0.99979132413864
        },
        "glasses": {
       "type": "sun",
            "probability": 0.99999964237213
        },
        "face_shape": {
            "type": "triangle",
            "probability": 0.5543018579483
        }
        "quality": {
            "occlusion": {
                "left_eye": 0,
                "right_eye": 0,
                "nose": 0,
                "mouth": 0,
                "left_cheek": 0.0064102564938366,
                "right_cheek": 0.0057411273010075,
                "chin": 0
            },
            "blur": 1.1886881756684e-10,
            "illumination": 141,
            "completeness": 1
        }
```

```
        }
    ]
}
```

**72个关键点分布图（对应landmark72个点的顺序，序号从0-71）：https://ai.bdstatic.com/file/52BC00FFD4754A6298D977EDAD033DA0

人脸搜索

- **1：N人脸搜索**：也称为1：N识别，在指定人脸集合中，找到最相似的人脸；
- **1：N人脸认证**：基于uid维度的1：N识别，由于uid已经锁定固定数量的人脸，所以检索范围更聚焦；

> **1：N人脸识别**与**1：N人脸认证**的差别在于：人脸搜索是在指定人脸集合中进行直接地人脸检索操作，而人脸认证是基于uid，先调取这个uid对应的人脸，再在这个uid对应的人脸集合中进行检索（因为每个uid通常对应的只有一张人脸，所以通常也就变为了1：1对比）；实际应用中，人脸认证需要用户或系统先输入id，这增加了验证安全度，但也增加了复杂度，具体使用哪个接口需要视您的业务场景判断。

```php
$image = "取决于image_type参数，传入BASE64字符串或URL字符串或FACE_TOKEN字符串";

$imageType = "BASE64";

$groupIdList = "3,2";

// 调用人脸搜索
$client->search($image, $imageType, $groupIdList);

// 如果有可选参数
$options = array();
$options["match_threshold"] = 70;
$options["quality_control"] = "NORMAL";
$options["liveness_control"] = "LOW";
$options["user_id"] = "233451";
$options["max_user_num"] = 3;

// 带参数调用人脸搜索
$client->search($image, $imageType, $groupIdList, $options);
```

人脸搜索 请求参数详情

参数名称	是否必选	类型	默认值	说明
image	是	string		图片信息(**总数据大小应小于10M**)，图片上传方式根据image_type来判断
image_type	是	string		图片类型 **BASE64**:图片的base64值，base64编码后的图片数据，编码后的图片大小不超过2M；**URL**:图片的 URL地址(可能由于网络等原因导致下载图片时间过长)； **FACE_TOKEN**: 人脸图片的唯一标识，调用人脸检测接口时，会为每个人脸图片赋予一个唯一的FACE_TOKEN，同一张图片多次检测得到的FACE_TOKEN是同一个
group_id_list	是	string		从指定的group中进行查找 用逗号分隔，**上限10个**
match_threshold	否	string		匹配阈值（设置阈值后，score低于此阈值的用户信息将不会返回） 最大100 最小0 默认80 **此阈值设置得越高，检索速度将会越快，推荐使用默认阈值80**
quality_control	否	string	NONE	图片质量控制 **NONE**: 不进行控制 **LOW**:较低的质量要求 **NORMAL**: 一般的质量要求 **HIGH**: 较高的质量要求 **默认 NONE**
liveness_control	否	string	NONE	活体检测控制 **NONE**: 不进行控制 **LOW**:较低的活体要求(高通过率 低攻击拒绝率) **NORMAL**: 一般的活体要求(平衡的攻击拒绝率, 通过率) **HIGH**: 较高的活体要求(高攻击拒绝率 低通过率) **默认NONE**
user_id	否	string		当需要对特定用户进行比对时，指定user_id进行比对。即人脸认证功能。
max_user_num	否	string		查找后返回的用户数量。返回相似度最高的几个用户，默认为1，最多返回50个。

人脸搜索 返回数据参数详情

字段	必选	类型	说明
face_token	是	string	人脸标志
user_list	是	array	匹配的用户信息列表
+group_id	是	string	用户所属的group_id
+user_id	是	string	用户的user_id
+user_info	是	string	注册用户时携带的user_info
+score	是	float	用户的匹配得分

人脸搜索 返回示例

```
{
 "face_token": "fid",
 "user_list": [
   {
     "group_id" : "test1",
     "user_id": "u333333",
     "user_info": "Test User",
     "score": 99.3
   }
 ]
}
```

人脸搜索 M:N 识别

待识别的图片中，存在多张人脸的情况下，支持在一个人脸库中，一次请求，同时返回图片中所有人脸的识别结果。

```php
$image = "取决于image_type参数，传入BASE64字符串或URL字符串或FACE_TOKEN字符串";

$imageType = "BASE64";

$groupIdList = "3,2";

// 调用人脸搜索 M:N 识别
$client->multiSearch($image, $imageType, $groupIdList);

// 如果有可选参数
$options = array();
$options["max_face_num"] = 3;
$options["match_threshold"] = 70;
$options["quality_control"] = "NORMAL";
$options["liveness_control"] = "LOW";
$options["max_user_num"] = 3;

// 带参数调用人脸搜索 M:N 识别
$client->multiSearch($image, $imageType, $groupIdList, $options);
```

人脸搜索 M:N 识别 请求参数详情

参数名称	是否必选	类型	默认值	说明
image	是	string		图片信息(**总数据大小应小于10M**)，图片上传方式根据image_type来判断
image_type	是	string		图片类型 **BASE64**:图片的base64值，base64编码后的图片数据，编码后的图片大小不超过2M；**URL**:图片的 URL地址(可能由于网络等原因导致下载图片时间过长)； **FACE_TOKEN**: 人脸图片的唯一标识，调用人脸检测接口时，会为每个人脸图片赋予一个唯一的FACE_TOKEN，同一张图片多次检测得到的FACE_TOKEN是同一个
group_id_list	是	string		从指定的group中进行查找 用逗号分隔，**上限10个**
max_face_num	否	string		最多处理人脸的数目 **默认值为1(仅检测图片中面积最大的那个人脸) 最大值10**
match_threshold	否	string		匹配阈值（设置阈值后，score低于此阈值的用户信息将不会返回） 最大100 最小0 默认80 **此阈值设置得越高，检索速度将会越快，推荐使用默认阈值80**
quality_control	否	string	NONE	图片质量控制 **NONE**: 不进行控制 **LOW**:较低的质量要求 **NORMAL**: 一般的质量要求 **HIGH**: 较高的质量要求 **默认 NONE**
liveness_control	否	string	NONE	活体检测控制 **NONE**: 不进行控制 **LOW**:较低的活体要求(高通过率 低攻击拒绝率) **NORMAL**: 一般的活体要求(平衡的攻击拒绝率, 通过率) **HIGH**: 较高的活体要求(高攻击拒绝率 低通过率) **默认NONE**
max_user_num	否	string		查找后返回的用户数量。返回相似度最高的几个用户，默认为1，最多返回20个。

人脸搜索 M:N 识别 返回数据参数详情

字段	必选	类型	说明
face_num	是	int	图片中的人脸数量
face_list	是	array	人脸信息列表
+face_token	是	string	人脸标志
+location	是	array	人脸在图片中的位置
++left	是	double	人脸区域离左边界的距离
++top	是	double	人脸区域离上边界的距离
++width	是	double	人脸区域的宽度
++height	是	double	人脸区域的高度
++rotation	是	int64	人脸框相对于竖直方向的顺时针旋转角，[-180,180]
+user_list	是	array	匹配的用户信息列表
++group_id	是	string	用户所属的group_id
++user_id	是	string	用户的user_id
++user_info	是	string	注册用户时携带的user_info
++score	是	float	用户的匹配得分 **80分以上可以判断为同一人，此分值对应万分之一误识率**

人脸搜索 M:N 识别 返回示例

```
{
  "error_code": 0,
  "error_msg": "SUCCESS",
  "log_id": 240483475,
  "timestamp": 1535533440,
  "cached": 0,
  "result": {
    "face_num": 2,
    "face_list": [
      {
        "face_token": "6fe19a6ee0c4233db9b5bba4dc2b9233",
        "location": {
          "left": 31.95568085,
          "top": 120.3764267,
          "width": 87,
          "height": 85,
          "rotation": -5
        },
        "user_list": [
          {
            "group_id": "group1",
            "user_id": "5abd24fd062e49bfa906b257ec40d284",
            "user_info": "userinfo1",
            "score": 69.85684967041
          },
          {
            "group_id": "group1",
            "user_id": "2abf89cffb31473a9948268fde9e1c3f",
            "user_info": "userinfo2",
            "score": 66.586112976074
          }
        ]
      },
      {
        "face_token": "fde61e9c074f48cf2bbb319e42634f41",
        "location": {
          "left": 219.4467773,
          "top": 104.7486954,
          "width": 81,
          "height": 77,
          "rotation": 3
        },
        "user_list": [
          {
            "group_id": "group1",
            "user_id": "088717532b094c3990755e91250adf7d",
            "user_info": "userinfo",
            "score": 65.154159545898
          }
        ]
      }
    ]
  }
}
```


人脸注册

用于从人脸库中新增用户，可以设定多个用户所在组，及组内用户的人脸图片，

典型应用场景：构建您的人脸库，如**会员人脸注册**，**已有用户补全人脸信息**等。

人脸库、用户组、用户、用户下的人脸**层级关系**如下所示：

```
|- 人脸库
 |- 用户组一
   |- 用户01
     |- 人脸
   |- 用户02
     |- 人脸
     |- 人脸
     ....
    ....
 |- 用户组二
 |- 用户组三
 |- 用户组四
 ....
```

关于人脸库的设置限制

- **每个appid对应一个人脸库，且不同appid之间，人脸库互不相通**；
- 每个人脸库下，可以创建多个用户组，用户组（group）数量**没有限制**；
- 每个用户组（group）下，可添加最多**无限**张人脸，**无限**个uid；
- 每个用户（uid）所能注册的最大人脸数量**20个**；

为了保证识别效果，请控制注册人脸的质量（通过`/detect`人脸检测接口判断），具体参数可详见下表所示：

质量判断

可通过人脸检测接口，基于以下字段和对应阈值，进行质量检测的判断，以保证人脸质量符合后续业务操作要求。

指标	字段与解释	推荐数值界限
遮挡范围	**occlusion**（0~1），0为无遮挡，1是完全遮挡 含有多个具体子字段，表示脸部多个部位 通常用作判断头发、墨镜、口罩等遮挡	left_eye：0.6, #左眼被遮挡的阈值 right_eye：0.6, #右眼被遮挡的阈值 nose：0.7, #鼻子被遮挡的阈值 mouth：0.7, #嘴巴被遮挡的阈值 left_check：0.8, #左脸颊被遮挡的阈值 right_check：0.8, #右脸颊被遮挡的阈值 chin_contour：0.6, #下巴被遮挡阈值
模糊度范围	**Blur**（0~1），0是最清晰，1是最模糊	小于0.7
光照范围	**illumination**（0~255） 脸部光照的灰度值，0表示光照不好 以及对应客户端SDK中，YUV的Y分量	大于40
姿态角度	**Pitch**：三维旋转之俯仰角度[-90(上), 90(下)] **Roll**：平面内旋转角[-180(逆时针), 180(顺时针)] **Yaw**：三维旋转之左右旋转角[-90(左), 90(右)]	分别小于20度
人脸完整度	**completeness**（0或1），0为人脸溢出图像边界，1为人脸都在图像边界内	视业务逻辑判断
人脸大小	人脸部分的大小 建议长宽像素值范围：80*80~200*200	人脸部分不小于**100*100**像素

```
 $image = "取决于image_type参数，传入BASE64字符串或URL字符串或FACE_TOKEN字符串";

$imageType = "BASE64";

$groupId = "group1";

$userId = "user1";

// 调用人脸注册
$client->addUser($image, $imageType, $groupId, $userId);

// 如果有可选参数
$options = array();
$options["user_info"] = "user's info";
$options["quality_control"] = "NORMAL";
$options["liveness_control"] = "LOW";
$options["action_type"] = "REPLACE";

// 带参数调用人脸注册
$client->addUser($image, $imageType, $groupId, $userId, $options);
```

人脸注册 请求参数详情

参数名称	是否必选	类型	默认值	说明
image	是	string		图片信息(总数据大小应小于10M)，图片上传方式根据image_type来判断。注：组内每个uid下的人脸图片数目上限为20张
image_type	是	string		图片类型 **BASE64**:图片的base64值，base64编码后的图片数据，编码后的图片大小不超过2M；**URL**:图片的 URL地址(可能由于网络等原因导致下载图片时间过长)；**FACE_TOKEN**: 人脸图片的唯一标识，调用人脸检测接口时，会为每个人脸图片赋予一个唯一的FACE_TOKEN，同一张图片多次检测得到的FACE_TOKEN是同一个
group_id	是	string		用户组id（由数字、字母、下划线组成），长度限制128B
user_id	是	string		用户id（由数字、字母、下划线组成），长度限制128B
user_info	否	string		用户资料，长度限制256B
quality_control	否	string	NONE	图片质量控制 **NONE**: 不进行控制 **LOW**:较低的质量要求 **NORMAL**: 一般的质量要求 **HIGH**: 较高的质量要求 **默认 NONE**
liveness_control	否	string	NONE	活体检测控制 **NONE**: 不进行控制 **LOW**:较低的活体要求(高通过率 低攻击拒绝率) **NORMAL**: 一般的活体要求(平衡的攻击拒绝率, 通过率) **HIGH**: 较高的活体要求(高攻击拒绝率 低通过率) **默认NONE**
action_type	否	string	APPEND	操作方式 APPEND: 当user_id在库中已经存在时，对此user_id重复注册时，新注册的图片默认会追加到该user_id下,REPLACE：当对此user_id重复注册时,则会用新图替换库中该user_id下所有图片,默认使用APPEND

人脸注册 返回数据参数详情

字段	必选	类型	说明
log_id	是	uint64	请求标识码，随机数，唯一
face_token	是	string	人脸图片的唯一标识
location	是	array	人脸在图片中的位置
+left	是	double	人脸区域离左边界的距离
+top	是	double	人脸区域离上边界的距离
+width	是	double	人脸区域的宽度
+height	是	double	人脸区域的高度
+rotation	是	int64	人脸框相对于竖直方向的顺时针旋转角，[-180,180]

人脸注册 返回示例

```
{
 "face_token": "2fa64a88a9d5118916f9a303782a97d3",
 "location": {
     "left": 117,
     "top": 131,
     "width": 172,
     "height": 170,
     "rotation": 4
 }
}
```

人脸更新

用于对人脸库中指定用户，更新其下的人脸图像。

> **说明：**针对一个uid执行更新操作，新上传的人脸图像将覆盖此uid原有所有图像。

```
 $image = "取决于image_type参数，传入BASE64字符串或URL字符串或FACE_TOKEN字符串";

$imageType = "BASE64";

$groupId = "group1";

$userId = "user1";

// 调用人脸更新
$client->updateUser($image, $imageType, $groupId, $userId);

// 如果有可选参数
$options = array();
$options["user_info"] = "user's info";
$options["quality_control"] = "NORMAL";
$options["liveness_control"] = "LOW";
$options["action_type"] = "REPLACE";

// 带参数调用人脸更新
$client->updateUser($image, $imageType, $groupId, $userId, $options);
```

人脸更新 请求参数详情

参数名称	是否必选	类型	默认值	说明
image	是	string		图片信息(**总数据大小应小于10M**)，图片上传方式根据image_type来判断
image_type	是	string		图片类型 **BASE64**:图片的base64值，base64编码后的图片数据，编码后的图片大小不超过2M；**URL**:图片的 URL地址(可能由于网络等原因导致下载图片时间过长)；**FACE_TOKEN**: 人脸图片的唯一标识，调用人脸检测接口时，会为每个人脸图片赋予一个唯一的FACE_TOKEN，同一张图片多次检测得到的FACE_TOKEN是同一个
group_id	是	string		更新指定groupid下uid对应的信息
user_id	是	string		用户id（由数字、字母、下划线组成），长度限制128B
user_info	否	string		用户资料，长度限制256B
quality_control	否	string	NONE	图片质量控制 **NONE**: 不进行控制 **LOW**:较低的质量要求 **NORMAL**: 一般的质量要求 **HIGH**: 较高的质量要求 **默认 NONE**
liveness_control	否	string	NONE	活体检测控制 **NONE**: 不进行控制 **LOW**:较低的活体要求(高通过率 低攻击拒绝率) **NORMAL**: 一般的活体要求(平衡的攻击拒绝率, 通过率) **HIGH**: 较高的活体要求(高攻击拒绝率 低通过率) **默认NONE**
action_type	否	string	UPDATE	操作方式 **UPDATE**：当user_id在库中已经存在时，对此user_id重复注册时，新注册的图片默认会追加到该user_id下,**REPLACE**：当对此user_id重复注册时,则会用新图替换库中该user_id下所有图片,默认使用**UPDATE**

人脸更新 返回数据参数详情

字段	必选	类型	说明
log_id	是	uint64	请求标识码，随机数，唯一
face_token	是	string	人脸图片的唯一标识
location	是	array	人脸在图片中的位置
+left	是	double	人脸区域离左边界的距离
+top	是	double	人脸区域离上边界的距离
+width	是	double	人脸区域的宽度
+height	是	double	人脸区域的高度
+rotation	是	int64	人脸框相对于竖直方向的顺时针旋转角，[-180,180]

人脸更新 返回示例

```
{
 "face_token": "2fa64a88a9d5118916f9a303782a97d3",
 "location": {
    "left": 117,
    "top": 131,
    "width": 172,
    "height": 170,
    "rotation": 4
 }
}
```

人脸删除

用于从人脸库中删除一个用户。

人脸删除注意事项：

- 删除的内容，包括用户所有图像和身份信息；
- 如果一个uid存在于多个用户组内，将会同时将从各个组中把用户删除
- 如果指定了group_id，则只删除此group下的uid相关信息

```
 $userId = "user1";

$groupId = "group1";

$faceToken = "face_token_23123";

// 调用人脸删除
$client->faceDelete($userId, $groupId, $faceToken);
```

人脸删除 请求参数详情

参数名称	是否必选	类型	说明
log_id	是	uint64	请求标识码，随机数，唯一
user_id	是	string	用户id（由数字、字母、下划线组成），长度限制128B
group_id	是	string	用户组id（由数字、字母、下划线组成），长度限制128B
face_token	是	string	需要删除的人脸图片token，（由数字、字母、下划线组成）长度限制64B

人脸删除 返回数据参数详情

字段	是否必选	类型	说明
log_id	是	uint64	请求唯一标识码，随机数

人脸删除 返回示例

```
 // 删除成功
{
 "error_code": 0,
 "log_id": 73473737,
}
// 删除发生错误
{
 "error_code": 223106,
 "log_id": 1382953199,
 "error_msg": "face is not exist"
}
```

用户信息查询

获取人脸库中某个用户的信息(user_info信息和用户所属的组)。

```
 $userId = "user1";

$groupId = "group1";

// 调用用户信息查询
$client->getUser($userId, $groupId);
```

用户信息查询 请求参数详情

参数名称	是否必选	类型	说明
user_id	是	string	用户id（由数字、字母、下划线组成），长度限制128B
group_id	是	string	用户组id（由数字、字母、下划线组成），长度限制128B

用户信息查询 返回数据参数详情

字段	必选	类型	说明
log_id	是	uint64	请求标识码，随机数，唯一
user_list	是	array	查询到的用户列表
+user_info	是	string	用户资料，被查询用户的资料
+group_id	是	string	用户组id，被查询用户的所在组

用户信息查询 返回示例

```
{
 "user_list": [
    {
        "user_info": "user info ...",
        "group_id": "gid1"
    },
    {
        "user_info": "user info2 ...",
        "group_id": "gid2"
    }
 ]
}
```

获取用户人脸列表

用于获取一个用户的全部人脸列表。

```
 $userId = "user1";

$groupId = "group1";

// 调用获取用户人脸列表
$client->faceGetlist($userId, $groupId);
```

获取用户人脸列表 请求参数详情

参数名称	是否必选	类型	说明
user_id	是	string	用户id（由数字、字母、下划线组成），长度限制128B
group_id	是	string	用户组id（由数字、字母、下划线组成），长度限制128B

获取用户人脸列表 返回数据参数详情

字段	必选	类型	说明
log_id	是	uint64	请求标识码，随机数，唯一
face_list	是	array	人脸列表
+face_token	是	string	人脸图片的唯一标识
+ctime	是	string	人脸创建时间

获取用户人脸列表 返回示例

```
{
 "face_list": [
    {
        "face_token": "fid1",
        "ctime": "2018-01-01 00:00:00"
    },
    {
        "face_token": "fid2",
        "ctime": "2018-01-01 10:00:00"
    }
 ]
}
```

获取用户列表

用于查询指定用户组中的用户列表。

```
 $groupId = "group1";

// 调用获取用户列表
$client->getGroupUsers($groupId);

// 如果有可选参数
$options = array();
$options["start"] = 0;
$options["length"] = 50;

// 带参数调用获取用户列表
$client->getGroupUsers($groupId, $options);
```

获取用户列表 请求参数详情

参数名称	是否必选	类型	默认值	说明
group_id	是	string		用户组id（由数字、字母、下划线组成），长度限制128B
start	否	string	0	默认值0，起始序号
length	否	string	100	返回数量，默认值100，最大值1000

获取用户列表 返回数据参数详情

字段	必选	类型	说明
user_id_list	是	array	用户ID列表

获取用户列表 返回示例

```
{
 "user_id_list": [
    "uid1",
    "uid2"
 ]
}
```

复制用户

用于将已经存在于人脸库中的用户**复制到一个新的组**。

```
 $userId = "user1";

// 调用复制用户
$client->userCopy($userId);

// 如果有可选参数
$options = array();
$options["src_group_id"] = "11111";
$options["dst_group_id"] = "222222";

// 带参数调用复制用户
$client->userCopy($userId, $options);
```

复制用户 请求参数详情

参数名称	是否必选	类型	说明
user_id	是	string	用户id（由数字、字母、下划线组成），长度限制128B
src_group_id	否	string	从指定组里复制信息
dst_group_id	否	string	需要添加用户的组id

复制用户 返回数据参数详情

字段	必选	类型	说明
log_id	是	id	log_id

复制用户 返回示例

```
 // 正确返回值
{
  "error_code": 0,
 "log_id": 3314921889,
}
// 发生错误时返回值
{
 "error_code": 216100,
 "log_id": 3111284097,
 "error_msg": "already add"
}
```

删除用户

用于将用户从某个组中删除。

```
 $groupId = "group1";

$userId = "user1";

// 调用删除用户
$client->deleteUser($groupId, $userId);
```

删除用户 请求参数详情

参数名称	是否必选	类型	说明
group_id	是	string	用户组id（由数字、字母、下划线组成），长度限制128B
user_id	是	string	用户id（由数字、字母、下划线组成），长度限制128B

删除用户 返回数据参数详情

字段	是否必选	类型	说明
log_id	是	uint64	请求唯一标识码，随机数

删除用户 返回示例

```
 // 正确返回值
{

 "error_code": 0,
 "log_id": 3314921889,
}
// 发生错误时返回值
{
 "error_code": 223103,
 "log_id": 815967402,
 "error_msg": "user is not exist"
}
```

创建用户组

用于创建一个空的用户组，如果用户组已存在 则返回错误。

```
 $groupId = "group1";

// 调用创建用户组
$client->groupAdd($groupId);
```

创建用户组 请求参数详情

参数名称	是否必选	类型	说明
group_id	是	string	用户组id（由数字、字母、下划线组成），长度限制128B

创建用户组 返回数据参数详情

字段	必选	类型	说明
log_id	是	uint64	请求标识码，随机数，唯一

创建用户组 返回示例

```
 {

 "error_code": 0,
 "log_id": 3314921889,
}
// 发生错误时返回值
{
 "error_code": 223101,
 "log_id": 815967402,
 "error_msg": " group is already exist"
}
```

删除用户组

删除用户组下所有的用户及人脸，如果组不存在 则返回错误。

```
$groupId = "group1";

// 调用删除用户组
$client->groupDelete($groupId);
```

删除用户组 请求参数详情

参数名称	是否必选	类型	说明
group_id	是	string	用户组id（由数字、字母、下划线组成），长度限制128B

删除用户组 返回数据参数详情

字段	必选	类型	说明
log_id	是	uint64	请求标识码，随机数，唯一

删除用户组 返回示例

```
// 正确返回值
{

  "error_code":0,
  "log_id": 3314921889,
}
// 发生错误时返回值
{
  "error_code": 223100,
  "log_id": 815967402,
  "error_msg": " group is not exist"
}
```

组列表查询

用于查询用户组的列表。

```
// 调用组列表查询
$client->getGroupList();

// 如果有可选参数
$options = array();
$options["start"] = 0;
$options["length"] = 50;

// 带参数调用组列表查询
$client->getGroupList(, $options);
```

组列表查询 请求参数详情

参数名称	是否必选	类型	默认值	说明
start	否	string	0	默认值0，起始序号
length	否	string	100	返回数量，默认值100，最大值1000

组列表查询 返回数据参数详情

字段	必选	类型	说明
group_id_list	是	array	group

组列表查询 返回示例

```
{
  "group_id_list": [
    "gid1",
    "gid2"
  ]
}
```

身份验证

质量检测（可选）活体检测（可选）公安验证（必选）

```
$image = "取决于image_type参数，传入BASE64字符串或URL字符串或FACE_TOKEN字符串";

$imageType = "BASE64";

$idCardNumber = "110233112299822211";

$name = "张三";

// 调用身份验证
$client->personVerify($image, $imageType, $idCardNumber, $name);

// 如果有可选参数
$options = array();
$options["quality_control"] = "NORMAL";
$options["liveness_control"] = "LOW";

// 带参数调用身份验证
$client->personVerify($image, $imageType, $idCardNumber, $name, $options);
```

身份验证 请求参数详情

参数名称	是否必选	类型	默认值	说明
image	是	string		图片信息(**总数据大小应小于10M**)，图片上传方式根据image_type来判断
image_type	是	string		图片类型 **BASE64**:图片的base64值，base64编码后的图片数据，编码后的图片大小不超过2M；**URL**:图片的 URL地址(可能由于网络等原因导致下载图片时间过长)；**FACE_TOKEN**: 人脸图片的唯一标识，调用人脸检测接口时，会为每个人脸图片赋予一个唯一的FACE_TOKEN，同一张图片多次检测得到的FACE_TOKEN是同一个
id_card_number	是	string		身份证号（真实身份证号号码）
name	是	string		utf8，姓名（真实姓名，和身份证号匹配）
quality_control	否	string	NONE	图片质量控制 **NONE**: 不进行控制 **LOW**:较低的质量要求 **NORMAL**: 一般的质量要求 **HIGH**: 较高的质量要求 **默认 NONE**
liveness_control	否	string	NONE	活体检测控制 **NONE**: 不进行控制 **LOW**:较低的活体要求(高通过率 低攻击拒绝率) **NORMAL**: 一般的活体要求(平衡的攻击拒绝率, 通过率) **HIGH**: 较高的活体要求(高攻击拒绝率 低通过率) **默认NONE**

身份验证 返回数据参数详情

参数	必须	类型	说明
log_id	是	uint64	日志id
score	是	float	与公安小图相似度可能性，用于验证生活照与公安小图是否为同一人，有正常分数时为[0~1]，推荐阈值80，超过即判断为同一人

身份验证 返回示例

```
{
 "score": 44.3,
}
```

语音校验码接口

此接口主要用于生成随机码，用于视频的语音识别校验使用，以判断视频的即时性，而非事先录制的，提升作弊的难度。

```
// 调用语音校验码接口
$client->videoSessioncode();

// 如果有可选参数
$options = array();
$options["appid"] = "223245";

// 带参数调用语音校验码接口
$client->videoSessioncode($options);
```

语音校验码接口 请求参数详情

参数名称	是否必选	类型	说明
appid	否	string	百度云创建应用时的唯一标识ID

语音校验码接口 返回数据参数详情

字段	必选	类型	说明
session_id	是	string	语音校验码会话id
code	是	string	语音验证码，数字形式，3~6位数字

语音校验码接口 返回示例

```
{
 "err_no": 0,
 "err_msg": "SUCCESS",
 "result": {
   "session_id": "S59faeeebb9111890355690",
   "code": "9940"
 },
 "timestamp": 1509617387,
 "cached": 0,
 "serverlogid": "0587756642"
}
```

在线活体检测

接口能力

- **人脸基础信息**：包括人脸框位置，人脸空间旋转角度，人脸置信度等信息。

- **人脸质量检测**：判断人脸的遮挡、光照、模糊度、完整度等质量信息。可用于判断上传的人脸是否符合标准。
- **基于图片的活体检测**：基于单张图片，判断图片中的人脸是否为二次翻拍（举例：如用户A用手机拍摄了一张包含人脸的图片一，用户B翻拍了图片一得到了图片二，并用图片二伪造成用户A去进行识别操作，这种情况普遍发生在金融开户、实名认证等环节）。此能力可用于H5场景下的一些人脸采集场景中，增加人脸注册的安全性和真实性。

```
$result = $client->faceverify(array(
    array(
        'image' => base64_encode(file_get_contents('1.jpg')),
        'image_type' => 'BASE64',
    ),
    array(
        'image' => base64_encode(file_get_contents('2.jpg')),
        'image_type' => 'BASE64',
    ),
));
```

请求参数

参数	是否必选	类型	说明
image	是	string	图片信息(总数据大小应小于10M)，图片上传方式根据image_type来判断； **可以上传同一个用户的1张、3张或8张图片来进行活体判断，注：后端会选择每组照片中的最高分数作为整体分数。**
image_type	是	string	图片类型 **BASE64**:图片的base64值，base64编码后的图片数据，编码后的图片大小不超过2M； **URL**:图片的 URL地址(可能由于网络等原因导致下载图片时间过长)； **FACE_TOKEN**: 人脸图片的唯一标识，调用人脸检测接口时，会为每个人脸图片赋予一个唯一的FACE_TOKEN，同一张图片多次检测得到的FACE_TOKEN是同一个。
face_field	否	string	包括**age,expression,faceshape,gender,glasses,landmark,quality,facetype信息**，逗号分隔，默认只返回face_token、活体数、人脸框、概率和旋转角度

返回参数

参数	类型	是否必须	说明
face_liveness	是	float	活体分数值
thresholds	是	array	由服务端返回最新的阈值数据（随着模型的优化，阈值可能会变化），将此参数与返回的face_liveness进行比较，可以作为活体判断的依据。 frr_1e-4：万分之一误识率的阈值；frr_1e-3：千分之一误识率的阈值；frr_1e-2：百分之一误识率的阈值。误识率越低，准确率越高，相应的拒绝率也越高
face_list	是	array	每张图片的详细信息描述，如果只上传一张图片，则只返回一个结果。
+face_token	是	string	人脸图片的唯一标识
+location	是	array	人脸在图片中的位置
++left	是	double	人脸区域离左边界的距离
++top	是	double	人脸区域离上边界的距离
++width	是	double	人脸区域的宽度

++width	是	double	人脸区域的宽度
++height	是	double	人脸区域的高度
++rotation	是	int64	人脸框相对于竖直方向的顺时针旋转角，[-180,180]
+face_probability	是	double	人脸置信度，范围【0~1】，代表这是一张人脸的概率，0最小、1最大。
+angel	是	array	人脸旋转角度参数
++yaw	是	double	三维旋转之左右旋转角[-90(左), 90(右)]
++pitch	是	double	三维旋转之俯仰角度[-90(上), 90(下)]
++roll	是	double	平面内旋转角[-180(逆时针), 180(顺时针)]
+age	否	double	年龄 ，当**face_field包含age时返回**
+expression	否	array	表情，当 **face_field包含expression时返回**
++type	否	string	**none**:不笑；**smile**:微笑；**laugh**:大笑
++probability	否	double	表情置信度，范围【0~1】，0最小、1最大。
+face_shape	否	array	脸型，当**face_field包含faceshape时返回**
++type	否	double	**square**: 正方形 **triangle**:三角形 **oval**: 椭圆 **heart**: 心形 **round**: 圆形
++probability	否	double	置信度，范围【0~1】，代表这是人脸形状判断正确的概率，0最小、1最大。
+gender	否	array	性别，**face_field包含gender时返回**
++type	否	string	male:**男性** female:**女性**
++probability	否	double	性别置信度，范围【0~1】，0代表概率最小、1代表最大。
+glasses	否	array	是否带眼镜，**face_field包含glasses时返回**
++type	否	string	**none**:无眼镜，**common**:普通眼镜，**sun**:墨镜
++probability	否	double	眼镜置信度，范围【0~1】，0代表概率最小、1代表最大。
++probability	否	double	人种置信度，范围【0~1】，0代表概率最小、1代表最大。
+face_type	否	array	真实人脸/卡通人脸 **face_field包含facetype时返回**
++type	否	string	**human**: 真实人脸 **cartoon**: 卡通人脸
++probability	否	double	人脸类型判断正确的置信度，范围【0~1】，0代表概率最小、1代表最大。
+landmark	否	array	4个关键点位置，左眼中心、右眼中心、鼻尖、嘴中心。**face_field包含landmark时返回**
+landmark72	否	array	72个特征点位置 **face_field包含landmark时返回**

+quality	否	array	人脸质量信息。**face_field包含quality时返回**
++occlusion	否	array	人脸各部分遮挡的概率，范围[0~1]，0表示完整，1表示不完整
+++left_eye	否	double	左眼遮挡比例，[0-1] ， 1表示完全遮挡
+++right_eye	否	double	右眼遮挡比例，[0-1] ， 1表示完全遮挡
+++nose	否	double	鼻子遮挡比例，[0-1] ， 1表示完全遮挡
+++mouth	否	double	嘴巴遮挡比例，[0-1] ， 1表示完全遮挡
+++left_cheek	否	double	左脸颊遮挡比例，[0-1] ， 1表示完全遮挡
+++right_cheek	否	double	右脸颊遮挡比例，[0-1] ， 1表示完全遮挡
+++chin	否	double	下巴遮挡比例，，[0-1] ， 1表示完全遮挡
++blur	否	double	人脸模糊程度，范围[0~1]，0表示清晰，1表示模糊
++illumination	否	double	取值范围在[0~255], 表示脸部区域的光照程度 越大表示光照越好
++completeness	否	int64	人脸完整度，0或1, 0为人脸溢出图像边界，1为人脸都在图像边界内
+parsing_info	否	string	人脸分层结果 结果数据是使用gzip压缩后再base64编码 使用前需base64解码后再解压缩 **原数据格式为string 形如0,0,0,0,0,1,1,1,1,1,1,2,2,2,2,2,2,2,2,2,…**

- 返回示例


```
{
 "thresholds": {
    "frr_1e-4": 0.05,  //万分之一误拒率的阈值
    "frr_1e-3": 0.3,   //千分之一误拒率的阈值
    "frr_1e-2": 0.9    //百分之一误拒率的阈值
 },
 "face_liveness": 0.05532243927,
 "face_list": [
    {
        "face_token": "df46f7c7db4aa09a093c26fb8d1a8d44",
        "location": {
            "left": 328.9026489,
            "top": 97.16340637,
            "width": 162,
            "height": 154,
            "rotation": 32
        },
        "face_probability": 1,
        "angle": {
            "yaw": 10.16196251,
            "pitch": 2.244354248,
            "roll": 33.82199097
        },
        "liveness": {
```

```
            "faceliveness": 0.004187555984,
            "livemapscore": 0.04492170034
        },
        "age": 23
    },
    {
        "face_token": "901d2c64274fccd687d311a6e6110a01",
        "location": {
            "left": 411.4876404,
            "top": 166.3593445,
            "width": 329,
            "height": 308,
            "rotation": 45
        },
        "face_probability": 0.9194830656,
        "angle": {
            "yaw": -1.716423035,
            "pitch": 7.344647408,
            "roll": 45.79914856
        },
        "liveness": {
            "faceliveness": 0.0001665892196,
            "livemapscore": 0.001787073661
        },
        "age": 23
    },
    {
        "face_token": "7d57e36981c48b4946eb97c8d838b02a",
        "location": {
            "left": 161.4559937,
            "top": 199.8726501,
            "width": 218,
            "height": 201,
            "rotation": -1
        },
        "face_probability": 1,
        "angle": {
            "yaw": -8.187754631,
            "pitch": 6.973727226,
            "roll": -1.25429821
        },
        "liveness": {
            "faceliveness": 0.02942637168,
            "livemapscore": 0.05532243927
        },
        "age": 23
    }
]
}
```


人脸对比

接口能力

- **两张人脸图片相似度对比**：比对两张图片中人脸的相似度，并返回相似度分值；
- **多种图片类型**：支持**生活照**、**证件照**、**身份证芯片照**、**带网纹照**四种类型的人脸对比；
- **活体检测**：基于图片中的破绽分析，判断其中的人脸是否为**二次翻拍**（举例：如用户A用手机拍摄了一张包含人脸的图片一，用户B翻拍了图片一得到了图片二，并用图片二伪造成用户A去进行识别操作，这种情况普遍发生在金融开户、实名认证等环节。）；

- **质量检测**：返回模糊、光照等质量检测信息，用于辅助判断图片是否符合识别要求；

业务应用

用于比对多张图片中的人脸相似度并返回两两比对的得分，可用于判断两张脸是否是同一人的可能性大小。

典型应用场景：如**人证合一验证**，**用户认证**等，可与您现有的人脸库进行比对验证。

```
$result = $client->match(array(
    array(
        'image' => base64_encode(file_get_contents('1.jpg')),
        'image_type' => 'BASE64',
    ),
    array(
        'image' => base64_encode(file_get_contents('2.jpg')),
        'image_type' => 'BASE64',
    ),
));
```

请求参数

参数	必选	类型	说明
image	是	string	图片信息(**总数据大小应小于10M**)，图片上传方式根据image_type来判断。 **两张图片通过json格式上传，格式参考表格下方示例**
image_type	是	string	图片类型 **BASE64**:图片的base64值，base64编码后的图片数据，编码后的图片大小不超过2M；**URL**:图片的 URL地址(可能由于网络等原因导致下载图片时间过长)；**FACE_TOKEN**: 人脸图片的唯一标识，调用人脸检测接口时，会为每个人脸图片赋予一个唯一的FACE_TOKEN，同一张图片多次检测得到的FACE_TOKEN是同一个。
face_type	否	string	人脸的类型 **LIVE**表示生活照：通常为手机、相机拍摄的人像图片、或从网络获取的人像图片等，**IDCARD**表示身份证芯片照：二代身份证内置芯片中的人像照片， **WATERMARK**表示带水印证件照：一般为带水印的小图，如公安网小图 **CERT**表示证件照片：如拍摄的身份证、工卡、护照、学生证等证件图片 默认LIVE
quality_control	否	string	图片质量控制 **NONE**: 不进行控制 **LOW**:较低的质量要求 **NORMAL**: 一般的质量要求 **HIGH**: 较高的质量要求 **默认 NONE**
liveness_control	否	string	活体检测控制 **NONE**: 不进行控制 **LOW**:较低的活体要求(高通过率 低攻击拒绝率) **NORMAL**: 一般的活体要求(平衡的攻击拒绝率, 通过率) **HIGH**: 较高的活体要求(高攻击拒绝率 低通过率) **默认NONE**

返回参数

参数名	必选	类型	说明
score	是	float	人脸相似度得分
face_list	是	array	人脸信息列表
+face_token	是	string	人脸的唯一标志

- 返回示例

```
{
  "score": 44.3,
  "face_list": [  //返回的顺序与传入的顺序保持一致
     {
        "face_token": "fid1"
     },
     {
        "face_token": "fid2"
     }
  ]
}
```

人脸实名认证V4

能力介绍

1. 业务能力

- **质量检测（可选）**：判断图片中是否包含人脸，以及人脸在姿态、遮挡、模糊、光照等方面是否符合识别条件。
- **活体检测（可选）**：基于图片中的破绽分析，判断其中的人脸是否为**二次翻拍**（举例：如用户A用手机拍摄了一张包含人脸的图片一，用户B翻拍了图片一得到了图片二，并用图片二伪造成用户A去进行识别操作，这种情况普遍发生在金融开户、实名认证等环节）。
- **图片加密及风控（可选）**：当配合增强级采集SDK、增强级安全加固采集SDK、金融级采集SDK、金融级安全加固采集SDK版本使用，对采集SDK输出的加密图片进行解密（加密传输可以有效避免第三方非法黑产绕过APP模拟请求攻击云端接口的行为，Eg：脚本攻击等）；以及结合百度安全实验室大数据风控能力，对采集SDK的发起端设备进行风控识别，辨别是否为风险设备，Eg：ROM注入、视频劫持等；
- **人脸实名认证（必选）**：基于姓名和身份证号，调取公安权威数据源人脸图，将当前获取的人脸图片，与公安数据源人脸图进行对比，得出比对分数，并基于此进行业务判断是否为同一人。由于公安数据源人脸图具有最权威的身份证明作用，故对用户本人的验证结果可信度也最为合理。

2. 业务逻辑

- 上述能力，人脸实名认证能力为必选能力，质量检测、活体检测、图片加密及风控为可选能力，验证顺序为`人脸质量检测->活体检测->人脸实名认证`。
- **如选择了质量检测和活体检测能力，则有任意一个条件不通过，整个请求流程终止，并返回错误码，描述具体的不符合信息**。
- 基于此顺序串行验证逻辑，可以避免大量不符合条件的请求流转到人脸实名认证，节约您的成本。

3. 推荐阈值

- 此接口使用的对比算法，针对带水纹证件照采用了专项的模型处理，可保证水纹信息的影响降到尽可能低。
- 如比对成功，最终返回的有效数据为一个**对比分值**，在0~100之间，您可以设定具体的阈值来判断是否验证通过。
- **人证相似度的推荐阈值为80，对应的误识率为万分之一**。

```php
 // 必填参数
$idCardNumber = "";
$name = "";
$image = "";

// 如果有可选参数
$options = array();
$options["xxx"] = "xxx";

// 调用接口
$client->faceMingJingVerify($idCardNumber, $name, $image, $options)
```

请求参数

参数	必选	类型	说明
app	否	string	APP端类型，配合采集SDK使用时须传入 **ios**：iOS端采集SDK **android**：安卓端采集SDK
sec_level	否	string	SDK安全级别，配合采集SDK使用时须传入，默认common **common**：配合4.1/4.1.5版本采集SDK，人脸图片未进行加密处理 **lite**: 配合5.2版本SDK
skey	否	string	使用5.2版本SDK请求时必填 **skey**：从SDK获取的密钥信息skey
x_device_id	否	string	使用5.2版本SDK请求时必填 **deviceId**：从SDK 获取的密钥信息deviceId
data	否	string	使用5.2版本SDK请求时必填， SDK输出的加密数据
id_card_number	是	string	身份证件号
name	是	string	姓名(需要是 utf8 编码)
livenesss_control	否	string	活体控制参数 **NONE**: 不进行控制 **LOW**:较低的活体要求(高通过率 低攻击拒绝率) **NORMAL**: 一般的活体要求(平衡的攻击拒绝率, 通过率) **HIGH**: 较高的活体要求(高攻击拒绝率 低通过率) **默认为NONE**
spoofing_control	否	string	合成图控制参数 **NONE**: 不进行控制 **LOW**:较低的合成图阈值数值，由于合成图判定逻辑为大于阈值视为合成图攻击，该项代表**低通过率、高攻击拒绝率** **NORMAL**: 一般的合成图阈值数值，由于合成图判定逻辑为大于阈值视为合成图攻击，该项代表**平衡的攻击拒绝率, 通过率** **HIGH**: 较高的合成图阈值数值，由于合成图判定逻辑为大于阈值视为合成图攻击，该项代表**高通过率、低攻击拒绝率**) **默认为NONE**
quality_control	否	string	质量控制参数 **NONE**: 不进行控制 **LOW**:较低的质量要求 **NORMAL**: 一般的质量要求 **HIGH**: 较高的质量要求 **默认为NONE**
image	是	string	图片信息(数据大小应小于10M 分辨率应小于1920*1080)，**5.2版本SDK请求时已包含在加密数据data中，无需额外传入**
image_type	否	string	图片类型 **BASE64**：图片的base64值 **URL**：图片的 URL **FACE_TOKEN**：人脸标识 **默认 BASE64**

返回参数

参数	类型	说明
log_id	number	调用的日志id
result	jsonObject	认证返回的结果
+score	float	与公安数据源人脸图相似度可能性，用于验证生活照与公安数据源人脸图是否为同一人，有正常分数时为[0~100]，推荐阈值80，超过即判断为同一人
+verify_status	number	认证状态，取值如下： 0 ： 正常 1 ： 身份证号与姓名不匹配或该身份证号不存在 2 ： 公安网图片不存在或质量过低
dec_image	string	对SDK传入的加密图片进行解密。 仅APP场景且进行了图片加密时，此参数返回解密后的人脸图片信息
risk_level	string	判断设备是否发生过风险行为来判断风险级别，取值（数值由高到低）： 1 – 高危 2 – 嫌疑 3 – 普通 4 – 正常
risk_tag	string	风险标签，若判断为有风险，则会有风险标签json 数组告知风险类型 例如：general_inject

返回示例


```
{
  "log_id": 1370579072568000512,
  "result":{
     "score": 40.884,
     "verify_status": 0
  },
  "dec_image": "/9j/4AAQSkZJRgABAgAAAQABAAD",
  "risk_level": "3",
  "risk_tag": [
     "若判断为有风险，则会有风险标签json 数组告知风险类型，如：general_inject"
  ]
}
```


人脸比对V4

能力介绍

1. 接口能力

- **两张人脸图片相似度对比**：对比两张图片中人脸的相似度，并返回相似度分值；
- **多种图片类型**：支持**生活照**、**证件照**、**身份证芯片照**、**带网纹照**四种类型的人脸对比；
- **活体检测控制**：基于图片中的破绽分析，判断其中的人脸是否为**二次翻拍**（举例：如用户A用手机拍摄了一张包含人脸的图片一，用户B翻拍了图片一得到了图片二，并用图片二伪造成用户A去进行识别操作，这种情况普遍发生在金融开户、实名认证等环节。）；
- **质量检测控制**：分析图片的中人脸的模糊度、角度、光照强度等特征，判断图片质量；
- **图片加密及风控：**配合增强级、金融级采集SDK使用
  - 对采集SDK输出的加密图片进行解密（加密传输可以有效避免第三方非法黑产**绕过APP模拟请求攻击云端接口的行为**，Eg：脚本攻击等）；
  - 以及结合百度安全实验室大数据风控能力，对采集SDK的发起端设备进行风控识别，辨别是否为**风险设备**，Eg：ROM注入、视频劫持等；

2. 业务应用

- 用于对比多张图片中的人脸相似度并返回两两对比的得分，可用于判断两张脸是否是同一人的可能性大小。

- 典型应用场景：如**人证合一验证**，**用户认证**等，可与您现有的人脸库进行对比验证。

```php
 // 必填参数
$image = "";
$imageType = "";
$registerImage = "";
$registerImageType = "";

// 如果有可选参数
$options = array();
$options["xxx"] = "xxx";

// 调用接口
$client->faceMingJingMatch($image, $imageType, $registerImage, $registerImageType, $options)
```

请求参数

参数	必选	类型	说明
app	否	string	APP类型，配合采集SDK使用时须传入 **ios**：iOS端采集SDK **android**：安卓端采集SDK
sec_level	否	string	SDK安全级别，配合采集SDK使用时须传入，默认common **common**：配合4.1/4.1.5版本SDK使用，人脸图片未进行加密处理 **lite**：配合5.2版本SDK使用
skey	否	string	使用5.2版本SDK请求时必填 从SDK获取的密钥信息
x_device_id	否	string	使用5.2版本SDK请求时必填 从SDK获取的密钥信息
data	否	string	使用5.2版本SDK请求时必填 SDK输出的加密数据
image	是	string	图片信息(数据大小应小于10M 分辨率应小于1920*1080)，***5.2版本SDK请求时已包含在加密数据data中，无需额外传入**
image_type	是	string	图片类型 **BASE64**：图片的base64值 **URL**：图片的URL **FACE_TOKEN**：人脸标识 默认 BASE64
face_type	否	string	人脸的类型 **LIVE**：表示生活照：通常为手机、相机拍摄的人像图片、或从网络获取的人像图片等 **IDCARD**：表示身份证芯片照：二代身份证内置芯片中的人像照片 **WATERMARK**：表示带水印证件照：一般为带水印的小图，如公安网小图 **CERT**：表示证件照片：如拍摄的身份证、工卡、护照、学生证等证件图片 **INFRARED** 表示红外照片：使用红外相机拍摄的照片 默认LIVE
quality_control	否	string	质量控制 **NONE**: 不进行控制 **LOW**:较低的质量要求 **NORMAL**: 一般的质量要求 **HIGH**: 较高的质量要求 **默认NONE**
liveness_control	否	string	活体控制 **NONE**: 不进行控制 **LOW**:较低的活体要求(高通过率 低攻击拒绝率) **NORMAL**: 一般的活体要求(平衡的攻击拒绝率, 通过率) **HIGH**: 较高的活体要求(高攻击拒绝率 低通过率) **默认NONE**
registerimage	是	string	图片信息(总数据大小应小于10M)，图片上传方式根据imagetype来判断。本图片特指客户服务器上传图片，非加密图片Base64值
registerimagetype	是	string	图片类型 **BASE64**：图片的base64值 **URL**：图片的 URL **FACE_TOKEN**：人脸标识 默认 BASE64
registerfacetype	否	string	人脸的类型
registerqualitycontrol	否	string	图片质量控制 **NONE**: 不进行控制 **LOW**:较低的质量要求 **NORMAL**: 一般的质量要求 **HIGH**: 较高的质量要求 **默认 NONE** 若图片质量不满足要求，则返回结果中会提示质量检测失败
registerlivenesscontrol	否	string	活体检测控制 **NONE**: 不进行控制 **LOW**:较低的活体要求(高通过率 低攻击拒绝率) **NORMAL**: 一般的活体要求(平衡的攻击拒绝率, 通过率) **HIGH**: 较高的活体要求(高攻击拒绝率 低通过率) **默认 NONE** 若活体检测结果不满足要求，则返回结果中会提示活体检测失败
facesorttype	否	int	人脸检测排序类型 0:代表检测出的人脸按照人脸面积从大到小排列 1:代表检测出的人脸按照距离图片中心从近到远排列 默认为0

返回参数

参数名	类型	说明
log_id	number	调用的日志id
result	jsonObject	认证返回的结果
+ score	number	人脸相似度得分，推荐阈值80分
+ face_list	jsonArray	人脸信息列表
++ face_token	string	人脸标志
dec_image	string	APP场景传入加密图片时，该项返回解密后的图片
risk_level	string	风控返回参数，只有在 risk_identify 为 true 时才返回，判断设备是否发生过风险行为来判断风险级别，取值（数值有高到低）： 1 – 高危 2 – 嫌疑 3 – 普通 4 – 正常
risk_tag	jsonArray	风控返回参数，只有在 risk_identify 为 true 时才返回，风险标签，若判断为有风险，则会有风险标签 json 数组告知风险类型
risk_warn_code	number	风控返回参数，只有在 risk_identify 为 true 时才返回，只有在风控服务异常的时候才返回这个字段

返回示例


```
{
  "log_id": 1370585066551377920,
  "result": {
    "score": 99.06919861,
    "face_list": [
      {
        "face_token": "549f9f1d1c7ec8c86931540b1939e8ed"
      },
      {
        "face_token": "1a319460ef89e8d27fb59062a28dbad7"
      }
    ]
  },
  "dec_image": "/9j/4AAQSkZJRgABAQAAAQABAAD",
  "risk_level": "3",
  "risk_tag": [
    "空"
  ]
}
```


在线图片活体V4

能力介绍

1. 接口能力

- **人脸基础信息**：包括人脸框位置，人脸空间旋转角度，人脸置信度等信息。
- **人脸质量检测**：判断人脸的遮挡、光照、模糊度、完整度等质量信息。可用于判断上传的人脸是否符合标准。
- **基于图片的活体检测**：基于单张图片，判断图片中的人脸是否为二次翻拍（举例：如用户A用手机拍摄了一张包含人脸的图片一，用户B翻拍了图片一得到了图片二，并用图片二伪造成用户A去进行识别操作，这种情况普遍发生在金融开户、实名

认证等环节）以及是否为合成图攻击。此能力可用于H5场景下的一些人脸采集场景中，增加人脸注册的安全性和真实性。

- **图片加密及风控**：配合采集SDK5.0版本使用，对采集SDK输出的加密图片进行解密（加密传输可以有效避免第三方非法黑产绕过APP模拟请求攻击云端接口的行为，Eg：脚本攻击等）；以及结合百度安全实验室大数据风控能力，对采集SDK的发起端设备进行风控识别，辨别是否为风险设备，Eg：ROM注入、视频劫持等；

```
 // 必填参数
$sdkVersion = 1;

// 如果有可选参数
$options = array();
$options["xxx"] = "xxx";

// 调用接口
$client->faceMingJingMatch($sdkVersion, $options)
```

请求参数

参数	是否必选	类型	说明
sdk_version	是	string	1 ：非加密图片，适用于4.1/4.1.5版本SDK、H5场景或纯服务端场景 4：适用于5.2版本SDK
face_field	否	string	包括**age,expression,face_shape,gender,glasses,landmark,quality,face_type,spoofing**信息，逗号分隔，默认只返回face_token、活体数、人脸框、概率和旋转角度
option	否	string	场景信息，程序会视不同的场景选用相对应的模型。当前支持的场景有COMMON(通用场景)，GATE(闸机场景)，FINANCE(金融场景)，LOGISTICS(物流场景)，INTERNET(泛互联网场景), **默认使用COMMON** 。注意：如果请求参数中存在多个option值时，则取第一个option的值，同时除通用场景外的其他场景需要联系工作人员对您所使用的appid进行配置
app	否	string	端类型 ios / android **5.2版本SDK必传该项**
s_key	否	string	端上提供的用于解密图片的skey **5.2版本SDK必传该项**
device_id	否	string	端上提供的用于解密图片的deviceId **5.2版本SDK必传该项**
data	否	string	端上提供的加密后的图片数组 **5.2版本SDK必传该项**
image_list	否	array	图片BASE64数组 4.1/4.1.5版本SDK、H5场景或纯服务端场景必填

返回参数

参数	是否必须	类型	说明
log_id	是	string	日志id
error_code	是	int	错误码状态，若为0则表示认证成功
error_msg	是	string	错误码说明，若为 success 则表示认证成功
result	是	object	活体结果
+face_liveness	是	float	活体分数值
+thresh			由服务端返回最新的阈值数据（随着模型的优化，阈值可能会变化），将此参数与返回的face_liveness进行比较，可以作为活体判断的依据。frr_1e-4：万分之一误识率的阈值； frr_1e-

+thresholds	是	array	face_liveness进行比较，可以作为活体判断的依据。 frr_1e-4：万分之一误识率的阈值； frr_1e-3：千分之一误识率的阈值； frr_1e-2：百分之一误识率的阈值。 误识率越低，准确率越高，相应的拒绝率也越高
+face_list	是	array	每张图片的详细信息描述，如果只上传一张图片，则只返回一个结果。
++face_token	是	string	人脸图片的唯一标识
++location	是	array	人脸在图片中的位置
+++left	是	double	人脸区域离左边界的距离
+++top	是	double	人脸区域离上边界的距离
+++width	是	double	人脸区域的宽度
+++height	是	double	人脸区域的高度
+++rotation	是	int64	人脸框相对于竖直方向的顺时针旋转角，[-180,180]
++face_probability	是	double	人脸置信度，范围【0~1】，代表这是一张人脸的概率，0最小、1最大。
++angel	是	array	人脸旋转角度参数
+++yaw	是	double	三维旋转之左右旋转角[-90(左), 90(右)]
+++pitch	是	double	三维旋转之俯仰角度[-90(上), 90(下)]
+++roll	是	double	平面内旋转角[-180(逆时针), 180(顺时针)]
++age	否	double	年龄 ，当**face_field包含age时返回**
++expression	否	array	表情，当 **face_field包含expression时返回**
+++type	否	string	**none**:不笑；**smile**:微笑；**laugh**:大笑
+++probability	否	double	表情置信度，范围【0~1】，0最小、1最大。
++face_shape	否	array	脸型，当**face_field包含face_shape时返回**
+++type	否	double	**square**: 正方形 **triangle**:三角形 **oval**: 椭圆 **heart**: 心形 **round**: 圆形
+++probability	否	double	置信度，范围【0~1】，代表这是人脸形状判断正确的概率，0最小、1最大。
++gender	否	array	性别，**face_field包含gender时返回**
+++type	否	string	male:**男性** female:**女性**
+++probability	否	double	性别置信度，范围【0~1】，0代表概率最小、1代表最大。

++glasses	否	array	是否带眼镜，**face_field包含glasses时返回**
++type	否	string	**none**:无眼镜，**common**:普通眼镜，**sun**:墨镜
+++probability	否	double	眼镜置信度，范围【0~1】，0代表概率最小、1代表最大。
++face_type	否	array	真实人脸/卡通人脸 **face_field包含face_type时返回**
+++type	否	string	**human**: 真实人脸 **cartoon**: 卡通人脸
+++probability	否	double	人脸类型判断正确的置信度，范围【0~1】，0代表概率最小、1代表最大。
++landmark	否	array	4个关键点位置，左眼中心、右眼中心、鼻尖、嘴中心。**face_field包含landmark时返回**
++landmark72	否	array	72个特征点位置 **face_field包含landmark时返回**
++quality	否	array	人脸质量信息。**face_field包含quality时返回**
+++occlusion	否	array	人脸各部分遮挡的概率，范围[0~1]，0表示完整，1表示不完整
++++left_eye	否	double	左眼遮挡比例，[0-1] ， 1表示完全遮挡
++++right_eye	否	double	右眼遮挡比例，[0-1] ， 1表示完全遮挡
++++nose	否	double	鼻子遮挡比例，[0-1] ， 1表示完全遮挡
++++mouth	否	double	嘴巴遮挡比例，[0-1] ， 1表示完全遮挡
++++left_cheek	否	double	左脸颊遮挡比例，[0-1] ， 1表示完全遮挡
++++right_cheek	否	double	右脸颊遮挡比例，[0-1] ， 1表示完全遮挡
++++chin	否	double	下巴遮挡比例，，[0-1] ， 1表示完全遮挡
+++blur	否	double	人脸模糊程度，范围[0~1]，0表示清晰，1表示模糊
+++illumination	否	double	取值范围在[0~255], 表示脸部区域的光照程度 越大表示光照越好
+++completeness	否	int64	人脸完整度，0或1, 0为人脸溢出图像边界，1为人脸都在图像边界内
++spoofing	否	double	合成图打分 判断图片是否为合成图 face_field包含时返回spoofing
risk_lev	否	string	判断设备是否发生过风险行为来判断风险级别，取值（数值有高到低）：1 – 高危 2 – 嫌疑 3 – 普

el	否	string	通 4 – 正常
risk_tag	否	string	风险标签，若判断为有风险，则会有风险标签 json 数组告知风险类型
dec_image	否	array	非安全加固增强级采集sdk、非安全加固金融级采集sdk、安全加固增强级采集sdk、安全加固金融级采集sdk会返回解密后的原图

返回示例


```
{
  "result": {
    "thresholds": {
      "frr_1e-3": 0.3,
      "frr_1e-2": 0.9,
      "frr_1e-4": 0.05
    },
    "face_liveness": 0.1976952702,
    "face_list": [
      {
        "liveness": {
          "livemapscore": 0.1976952702
        },
        "angle": {
          "roll": 0.31,
          "pitch": -2.07,
          "yaw": 0.85
        },
        "face_token": "abd1b4ce743099336cfed40193ff4944",
        "location": {
          "top": 161.74,
          "left": 79.04,
          "rotation": 1,
          "width": 318,
          "height": 333
        },
        "face_probability": 1
      },
      {
        "liveness": {
          "livemapscore": 0.1976952702
        },
        "angle": {
          "roll": 0.31,
          "pitch": -2.07,
          "yaw": 0.85
        },
        "face_token": "abd1b4ce743099336cfed40193ff4944",
        "location": {
          "top": 161.74,
          "left": 79.04,
          "rotation": 1,
          "width": 318,
          "height": 333
        },
        "face_probability": 1
      },
      {
        "liveness": {
          "livemapscore": 0.1976952702
        },
        "angle": {
          "roll": 0.31,
```

```
            "pitch": -2.07,
            "yaw": 0.85
        },
        "face_token": "abd1b4ce743099336cfed40193ff4944",
        "location": {
            "top": 161.74,
            "left": 79.04,
            "rotation": 1,
            "width": 318,
            "height": 333
        },
        "face_probability": 1
    },
    {
        "liveness": {
            "livemapscore": 0.1976952702
        },
        "angle": {
            "roll": 0.31,
            "pitch": -2.07,
            "yaw": 0.85
        },
        "face_token": "abd1b4ce743099336cfed40193ff4944",
        "location": {
            "top": 161.74,
            "left": 79.04,
            "rotation": 1,
            "width": 318,
            "height": 333
        },
        "face_probability": 1
    },
    {
        "liveness": {
            "livemapscore": 0.1976952702
        },
        "angle": {
            "roll": 0.31,
            "pitch": -2.07,
            "yaw": 0.85
        },
        "face_token": "abd1b4ce743099336cfed40193ff4944",
        "location": {
            "top": 161.74,
            "left": 79.04,
            "rotation": 1,
            "width": 318,
            "height": 333
        },
        "face_probability": 1
    },
    {
        "liveness": {
            "livemapscore": 0.1976952702
        },
        "angle": {
            "roll": 0.31,
            "pitch": -2.07,
            "yaw": 0.85
        },
        "face_token": "abd1b4ce743099336cfed40193ff4944",
        "location": {
            "top": 161.74,
```

```
                "top": 161.74,
                "left": 79.04,
                "rotation": 1,
                "width": 318,
                "height": 333
            },
            "face_probability": 1
        },
        {
            "liveness": {
                "livemapscore": 0.1976952702
            },
            "angle": {
                "roll": 0.31,
                "pitch": -2.07,
                "yaw": 0.85
            },
            "face_token": "abd1b4ce743099336cfed40193ff4944",
            "location": {
                "top": 161.74,
                "left": 79.04,
                "rotation": 1,
                "width": 318,
                "height": 333
            },
            "face_probability": 1
        },
        {
            "liveness": {
                "livemapscore": 0.1976952702
            },
            "angle": {
                "roll": 0.31,
                "pitch": -2.07,
                "yaw": 0.85
            },
            "face_token": "abd1b4ce743099336cfed40193ff4944",
            "location": {
                "top": 161.74,
                "left": 79.04,
                "rotation": 1,
                "width": 318,
                "height": 333
            },
            "face_probability": 1
        }
    ]
},
"risk_level": "3",
"log_id": 1423545654699779516,
"risk_tag": [
    "空"
],
"dec_image": [
    "/9j/4AAQ..."
]
}
```


错误信息

错误返回格式

若请求错误，服务器将返回的JSON文本包含以下参数：

- **error_code**：错误码。
- **error_msg**：错误描述信息，帮助理解和解决发生的错误。

错误码

服务端返回的错误码

错误码	错误信息	描述
1	Unknown error	服务器内部错误，请再次请求， 如果持续出现此类错误，请通过QQ群（860337848）或提交工单联系技术支持团队。
2	Service temporarily unavailable	服务暂不可用，请再次请求， 如果持续出现此类错误，请通过QQ群（860337848）或提交工单联系技术支持团队。
3	Unsupported openapi method	调用的API不存在，请检查请求URL后重新尝试，一般为URL中有非英文字符，如“-”，可手动输入重试
4	Open api request limit reached	集群超限额，请再次请求， 如果持续出现此类错误，请通过QQ群（860337848）或提交工单联系技术支持团队。
6	No permission to access data	无权限访问该用户数据，创建应用时未勾选相关接口
13	Get service token failed	获取token失败
14	IAM Certification failed	IAM 鉴权失败
15	app not exsits or create failed	应用不存在或者创建失败
17	Open api daily request limit reached	每天请求量超限额，可通过QQ群（860337848）联系群管、提交工单提升限额
18	Open api qps request limit reached	QPS超限额，可通过QQ群（860337848）联系群管、提交工单提升限额
19	Open api total request limit reached	请求总量超限额，可通过QQ群（860337848）联系群管、提交工单提升限额
100	Invalid parameter	无效的access_token参数，请检查后重新尝试
110	Access token invalid or no longer valid	access_token无效
111	Access token expired	access token过期
222001	param[] is null	必要参数未传入
222002	param[start] format error	参数格式错误
222003	param[length] format error	参数格式错误
222004	param[op_app_id_list] format error	参数格式错误
222005	param[group_id_list] format error	参数格式错误
222006	group_id format error	参数格式错误
222007	uid format error	参数格式错误
222008	face_id format error	参数格式错误

222008	face_id format error	参数格式错误
222009	quality_conf format error	参数格式错误
222010	user_info format error	参数格式错误
222011	param[uid_list] format error	参数格式错误
222012	param[op_app_id] format error	参数格式错误
222013	param[image] format error	参数格式错误
222014	param[app_id] format error	参数格式错误
222015	param[image_type] format error	参数格式错误
222016	param[max_face_num] format error	参数格式错误
222017	param[face_field] format error	参数格式错误
222018	param[user_id] format error	参数格式错误
222019	param[quality_control] format error	参数格式错误
222020	param[liveness_control] format error	参数格式错误
222021	param[max_user_num] format error	参数格式错误
222022	param[id_card_number] format error	参数格式错误
222023	param[name] format error	参数格式错误
222024	param[face_type] format error	参数格式错误
222025	param[face_token] format error	参数格式错误
222026	param[max_star_num] format error	参数格式错误
222201	network not available	服务端请求失败
222202	pic not has face	图片中没有人脸
222203	image check fail	无法解析人脸
222204	image_url_download_fail	从图片的url下载图片失败
222205	network not availablel	服务端请求失败
222206	rtse service return fail	服务端请求失败
222207	match user is not found	未找到匹配的用户
222208	the number of image	图片的数量错误

222208	is incorrect	图片的数量错误
222209	face token not exist	face token不存在
222300	add face fail	人脸图片添加失败
222301	get face fail	获取人脸图片失败
222302	system error	服务端请求失败
222303	get face fail	获取人脸图片失败
223100	group is not exist	操作的用户组不存在
223101	group is already exist	该用户组已存在
223102	user is already exist	该用户已存在
223103	user is not exist	找不到该用户
223104	group_list is too large	group_list包含组 数量过多
223105	face is already exist	该人脸已存在
223106	face is not exist	该人脸不存在
223110	uid_list is too large	uid_list包含数量过多
223111	dst group is not exist	目标用户组不存在
223112	quality_conf format error	quality_conf格式不正确
223113	face is covered	人脸有被遮挡
223114	face is fuzzy	人脸模糊
223115	face light is not good	人脸光照不好
223116	incomplete face	人脸不完整
223117	app_list is too large	app_list包含app数量 过多
223118	quality control error	质量控制项错误
223119	liveness control item error	活体控制项错误
223120	liveness check fail	活体检测未通过
223121	left eye is occlusion	质量检测未通过 左眼 遮挡程度过高
223122	right eye is occlusion	质量检测未通过 右眼 遮挡程度过高
223123	left cheek is occlusion	质量检测未通过 左脸 遮挡程度过高
223124	right cheek is occlusion	质量检测未通过 右脸 遮挡程度过高
223125	chin contour is occlusion	质量检测未通过 下巴遮挡程度过高
223126	nose is occlusion	质量检测未通过 鼻子遮挡程度过高
223127	mouth is occlusion	质量检测未通过 嘴巴 遮挡程度过高
[illegible]	police picture is none or	公安网图片不存在或

222350	low quality	质量过低
222351	id number and name not match or id number not exist	身份证号与姓名不匹配或该身份证号不存在
222352	name format error	身份证名字格式错误
222353	id number format error	身份证号码格式错误
222354	id number not exist	公安库里不存在此身份证号
222355	police picture not exist	身份证号码正确，公安库里没有对应的照片
222360	invalid name or id number	身份证号码或名字非法（公安网校验不通过）
222901	system busy	系统繁忙
222902	system busy	系统繁忙
222903	system busy	系统繁忙
222904	system busy	系统繁忙
222905	system busy	系统繁忙
222906	system busy	系统繁忙
222907	system busy	系统繁忙
222908	system busy	系统繁忙
222909	system busy	系统繁忙
222910	system busy	系统繁忙
222911	system busy	系统繁忙
222912	system busy	系统繁忙
222913	system busy	系统繁忙
222914	system busy	系统繁忙
222915	system busy	系统繁忙
222916	system busy	系统繁忙
222361	system busy	系统繁忙

C#-SDK

简介

Hi，您好，欢迎使用百度人脸识别服务。

本文档主要针对C#开发者，描述百度人脸识别接口服务的相关技术内容。如果您对文档内容有任何疑问，可以通过以下几种方式联系我们：

- 在百度云控制台内提交工单，咨询问题类型请选择**人工智能服务**；
- 如有疑问，进入AI社区交流：http://ai.baidu.com/forum/topic/list/165

接口能力

接口名称	接口能力简要描述
人脸检测	检测人脸并定位，返回五官关键点，及人脸各属性值
人脸比对	返回两两比对的人脸相似值
人脸查找	在一个人脸集合中找到找到相似的人脸，由一系列接口组成，包括人脸识别、人脸认证、人脸库管理相关接口（人脸注册、人脸更新、人脸删除、用户信息查询、组列表查询、组内用户列表查询、组间复制用户、组内删除用户）

版本更新记录

上线日期	版本号	更新内容
2019.4.17	3.6.11	人脸v3文档更新，新增N:M接口
2018.5.10	3.5.1	细节修复
2018.4.28	3.5.0	全面切换为人脸V3接口
2018.4.9	3.4.0	新增.Net Core支持；新增身份证验证接口，在线活体检测接口
2018.1.11	3.3.1	新增人脸识别N:M接口
2017.12.21	3.3.0	接口升级
2017.11.14	3.2.1	人脸检测接口升级
2017.9.12	3.0.0	更新SDK打包方式：所有AI服务集成一个SDK
2017.4.1	1.0	第一版！

快速入门

安装人脸 C# SDK

支持平台：.Net Framework 3.5 4.0 4.5， .Net Core 2.0

方法一：使用Nuget管理依赖 （推荐） 在NuGet中搜索 `Baidu.AI`，安装最新版即可。

packet地址 https://www.nuget.org/packages/Baidu.AI/

方法二：下载安装

人脸 C# SDK目录结构

```
Baidu.Aip
  ├─── net35
  │    ├─── AipSdk.dll            // 百度AI服务 windows 动态库
  │    ├─── AipSdk.xml            // 注释文件
  │    └─── Newtonsoft.Json.dll   // 第三方依赖
  ├─── net40
  ├─── net45
  └─── netstandard2.0
     ├─── AipSdk.deps.json
     └─── AipSdk.dll
```

> 如果需要在 Unity 平台使用，可引用工程源码自行编译。

安装

1.在官方网站下载C# SDK压缩工具包。

2.解压后，将 `AipSdk.dll` 和 `Newtonsoft.Json.dll` 中添加为引用。

新建交互类

Baidu.Aip.Face.Face是人脸的交互类，为使用人脸的开发人员提供了一系列的交互方法。

用户可以参考如下代码新建一个交互类：

```
 // 设置APPID/AK/SK
var APP_ID = "你的 App ID";
var API_KEY = "你的 Api Key";
var SECRET_KEY = "你的 Secret Key";

var client = new Baidu.Aip.Face.Face(API_KEY, SECRET_KEY);
client.Timeout = 60000;  // 修改超时时间
```

在上面代码中，常量`APP_ID`在百度云控制台中创建，常量`API_KEY`与`SECRET_KEY`是在创建完毕应用后，系统分配给用户的，均为字符串，用于标识用户，为访问做签名验证，可在AI服务控制台中的**应用列表**中查看。

注意：如您以前是百度云的老用户，其中`API_KEY`对应百度云的“Access Key ID”，`SECRET_KEY`对应百度云的“Access Key Secret”。

接口说明

人脸检测

人脸检测：检测图片中的人脸并标记出位置信息;

```
 public void DetectDemo() {
var image = "取决于image_type参数，传入BASE64字符串或URL字符串或FACE_TOKEN字符串";

var imageType = "BASE64";

// 调用人脸检测，可能会抛出网络等异常，请使用try/catch捕获
var result = client.Detect(image, imageType);
Console.WriteLine(result);
// 如果有可选参数
var options = new Dictionary<string, object>{
    {"face_field", "age"},
    {"max_face_num", 2},
    {"face_type", "LIVE"},
    {"liveness_control", "LOW"}
};
// 带参数调用人脸检测
result = client.Detect(image, imageType, options);
Console.WriteLine(result);
}
```

人脸检测 请求参数详情

参数名称	是否必选	类型	默认值	说明
image	是	string		图片信息(**总数据大小应小于10M**)，图片上传方式根据image_type来判断
image_type	是	string		图片类型 **BASE64**:图片的base64值，base64编码后的图片数据，编码后的图片大小不超过2M；**URL**:图片的 URL地址(可能由于网络等原因导致下载图片时间过长)； **FACE_TOKEN**: 人脸图片的唯一标识，调用人脸检测接口时，会为每个人脸图片赋予一个唯一的FACE_TOKEN，同一张图片多次检测得到的FACE_TOKEN是同一个
face_field	否	string		包括**age,expression,face_shape,gender,glasses,landmark,landmark150,quality,eye_status,emotion,face_type信息** 逗号分隔. 默认只返回face_token、人脸框、概率和旋转角度
max_face_num	否	string	1	最多处理人脸的数目，默认值为1，仅检测图片中面积最大的那个人脸；**最大值10**，检测图片中面积最大的几张人脸。
face_type	否	string		人脸的类型 **LIVE**表示生活照：通常为手机、相机拍摄的人像图片、或从网络获取的人像图片等**IDCARD**表示身份证芯片照：二代身份证内置芯片中的人像照片 **WATERMARK**表示带水印证件照：一般为带水印的小图，如公安网小图 **CERT**表示证件照片：如拍摄的身份证、工卡、护照、学生证等证件图片 默认**LIVE**
liveness_control	否	string	NONE	活体检测控制 **NONE**: 不进行控制 **LOW**:较低的活体要求(高通过率 低攻击拒绝率) **NORMAL**: 一般的活体要求(平衡的攻击拒绝率, 通过率) **HIGH**: 较高的活体要求(高攻击拒绝率 低通过率) **默认NONE**

人脸检测 返回数据参数详情

字段	必选	类型	说明
face_num	是	int	检测到的图片中的人脸数量
face_list	是	array	人脸信息列表，具体包含的参数参考下面的列表。
+face_token	是	string	人脸图片的唯一标识
+location	是	array	人脸在图片中的位置
++left	是	double	人脸区域离左边界的距离
++top	是	double	人脸区域离上边界的距离
++width	是	double	人脸区域的宽度
++height	是	double	人脸区域的高度
++rotation	是	int64	人脸框相对于竖直方向的顺时针旋转角，[-180,180]
+face_probability	是	double	人脸置信度，范围【0~1】，代表这是一张人脸的概率，0最小、1最大。
+angel	是	array	人脸旋转角度参数
++yaw	是	double	三维旋转之左右旋转角[-90(左), 90(右)]
++pitch	是	double	三维旋转之俯仰角度[-90(上), 90(下)]
++roll	是	double	平面内旋转角[-180(逆时针), 180(顺时针)]
+age	否	double	年龄 ，当**face_field包含age时返回**
+expression	否	array	表情，当 **face_field包含expression时返回**
++type	否	string	**none**:不笑；**smile**:微笑；**laugh**:大笑
++probability	否	double	表情置信度，范围【0~1】，0最小、1最大。
+face_shape	否	array	脸型，当**face_field包含face_shape时返回**
++type	否	double	**square**: 正方形 **triangle**:三角形 **oval**: 椭圆 **heart**: 心形 **round**: 圆形

++probability	否	double	置信度，范围【0~1】，代表这是人脸形状判断正确的概率，0最小、1最大。
+gender	否	array	性别，**face_field包含gender时返回**
++type	否	string	male:**男性** female:**女性**
++probability	否	double	性别置信度，范围【0~1】，0代表概率最小、1代表最大。
+glasses	否	array	是否带眼镜，**face_field包含glasses时返回**
++type	否	string	**none**:无眼镜，**common**:普通眼镜，**sun**:墨镜
++probability	否	double	眼镜置信度，范围【0~1】，0代表概率最小、1代表最大。
+eye_status	否	array	双眼状态（睁开/闭合） **face_field包含eye_status时返回**
++left_eye	否	double	左眼状态 [0,1]取值，越接近0闭合的可能性越大
++right_eye	否	double	右眼状态 [0,1]取值，越接近0闭合的可能性越大
+emotion	否	array	情绪 **face_field包含emotion时返回**
++type	否	string	**angry**:愤怒 **disgust**:厌恶 **fear**:恐惧 **happy**:高兴 **sad**:伤心 **surprise**:惊讶 **neutral**:无情绪
++probability	否	double	情绪置信度，范围0~1
++probability	否	double	人种置信度，范围【0~1】，0代表概率最小、1代表最大。
+face_type	否	array	真实人脸/卡通人脸 **face_field包含face_type时返回**
++type	否	string	**human**: 真实人脸 **cartoon**: 卡通人脸
++probability	否	double	人脸类型判断正确的置信度，范围【0~1】，0代表概率最小、1代表最大。
+landmark	否	array	4个关键点位置，左眼中心、右眼中心、鼻尖、嘴中心。**face_field包含landmark时返回**
+landmark72	否	array	72个特征点位置 **face_field包含landmark时返回**
+landmark150	否	array	150个特征点位置 **face_field包含landmark150时返回**
+quality	否	array	人脸质量信息。**face_field包含quality时返回**
++occlusion	否	array	人脸各部分遮挡的概率，范围[0~1]，0表示完整，1表示不完整
+++left_eye	否	double	左眼遮挡比例，[0-1] ，1表示完全遮挡
+++right_eye	否	double	右眼遮挡比例，[0-1] ， 1表示完全遮挡
+++nose	否	double	鼻子遮挡比例，[0-1] ， 1表示完全遮挡
+++mouth	否	double	嘴巴遮挡比例，[0-1] ， 1表示完全遮挡
+++left_cheek	否	double	左脸颊遮挡比例，[0-1] ， 1表示完全遮挡
+++right_cheek	否	double	右脸颊遮挡比例，[0-1] ， 1表示完全遮挡
+++chin_contour	否	double	下巴遮挡比例，，[0-1] ， 1表示完全遮挡
++blur	否	double	人脸模糊程度，范围[0~1]，0表示清晰，1表示模糊
++illumination	否	double	取值范围在[0~255], 表示脸部区域的光照程度 越大表示光照越好
++completeness	否	int64	人脸完整度，0或1, 0为人脸溢出图像边界，1为人脸都在图像边界内

人脸检测 返回示例

```
{
"face_num": 1,
"face_list": [
    {
        "face_token": "35235asfas21421fakghktyfdgh68bio",
        "location": {
            "left": 117,
```

```
            "top": 131,
            "width": 172,
            "height": 170,
            "rotation": 4
        },
        "face_probability": 1,
        "angle" :{
            "yaw" : -0.34859421849251
            "pitch" 1.9135693311691
            "roll" :2.3033397197723
        }
        "landmark": [
            {
                "x": 161.74819946289,
                "y": 163.30244445801
            },
            ...
        ],
        "landmark72": [
            {
                "x": 115.86531066895,
                "y": 170.0546875
            } ,
            ...
        ],
        "age": 29.298097610474,
        "expression": {
            "type": "smile",
            "probability" : 0.5543018579483
        },
        "gender": {
            "type": "male",
            "probability": 0.99979132413864
        },
        "glasses": {
       "type": "sun",
            "probability": 0.99999964237213
        },
        "face_shape": {
            "type": "triangle",
            "probability": 0.5543018579483
        }
        "quality": {
            "occlusion": {
                "left_eye": 0,
                "right_eye": 0,
                "nose": 0,
                "mouth": 0,
                "left_cheek": 0.0064102564938366,
                "right_cheek": 0.0057411273010075,
                "chin": 0
            },
            "blur": 1.1886881756684e-10,
            "illumination": 141,
            "completeness": 1
        }
    }
  ]
}
```


**72个关键点分布图（对应landmark72个点的顺序，序号从0-

71）：https://ai.bdstatic.com/file/52BC00FFD4754A6298D977EDAD033DA0

人脸搜索

- **1：N人脸搜索**：也称为1：N识别，在指定人脸集合中，找到最相似的人脸；
- **1：N人脸认证**：基于uid维度的1：N识别，由于uid已经锁定固定数量的人脸，所以检索范围更聚焦；

> **1：N人脸识别**与**1：N人脸认证**的差别在于：人脸搜索是在指定人脸集合中进行直接地人脸检索操作，而人脸认证是基于uid，先调取这个uid对应的人脸，再在这个uid对应的人脸集合中进行检索（因为每个uid通常对应的只有一张人脸，所以通常也就变为了1：1对比）；实际应用中，人脸认证需要用户或系统先输入id，这增加了验证安全度，但也增加了复杂度，具体使用哪个接口需要视您的业务场景判断。

```
 public void SearchDemo() {
var image = "取决于image_type参数，传入BASE64字符串或URL字符串或FACE_TOKEN字符串";

var imageType = "BASE64";

var groupIdList = "3,2";

// 调用人脸搜索，可能会抛出网络等异常，请使用try/catch捕获
var result = client.Search(image, imageType, groupIdList);
Console.WriteLine(result);
// 如果有可选参数
var options = new Dictionary<string, object>{
    {"match_threshold", 70},
    {"quality_control", "NORMAL"},
    {"liveness_control", "LOW"},
    {"user_id", "233451"},
    {"max_user_num", 3}
};
// 带参数调用人脸搜索
result = client.Search(image, imageType, groupIdList, options);
Console.WriteLine(result);
}
```

人脸搜索 请求参数详情

参数名称	是否必选	类型	默认值	说明
image	是	string		图片信息(**总数据大小应小于10M**)，图片上传方式根据image_type来判断
image_type	是	string		图片类型 **BASE64**:图片的base64值，base64编码后的图片数据， 编码后的图片大小不超过2M；**URL**:图片的 URL地址(可能由于网络等原因导致下载图片时间过长)； **FACE_TOKEN**: 人脸图片的唯一标识，调用人脸检测接口时，会为每个人脸图片赋予一个唯一的FACE_TOKEN，同一张图片多次检测得到的FACE_TOKEN是同一个
group_id_list	是	string		从指定的group中进行查找 用逗号分隔，**上限10个**
match_threshold	否	string		匹配阈值（设置阈值后，score低于此阈值的用户信息将不会返回） 最大100 最小0 默认80 **此阈值设置得越高，检索速度将会越快，推荐使用默认阈值80**
quality_control	否	string	NONE	图片质量控制 **NONE**: 不进行控制 **LOW**:较低的质量要求 **NORMAL**: 一般的质量要求 **HIGH**: 较高的质量要求 **默认 NONE**
liveness_control	否	string	NONE	活体检测控制 **NONE**: 不进行控制 **LOW**:较低的活体要求(高通过率 低攻击拒绝率) **NORMAL**: 一般的活体要求(平衡的攻击拒绝率, 通过率) **HIGH**: 较高的活体要求(高攻击拒绝率 低通过率) **默认NONE**
user_id	否	string		当需要对特定用户进行比对时，指定user_id进行比对。即人脸认证功能。
max_user_num	否	string		查找后返回的用户数量。返回相似度最高的几个用户，默认为1，最多返回50个。

人脸搜索 返回数据参数详情

字段	必选	类型	说明
face_token	是	string	人脸标志
user_list	是	array	匹配的用户信息列表
+group_id	是	string	用户所属的group_id
+user_id	是	string	用户的user_id
+user_info	是	string	注册用户时携带的user_info
+score	是	float	用户的匹配得分

人脸搜索 返回示例

```
{
 "face_token": "fid",
 "user_list": [
   {
     "group_id" : "test1",
     "user_id": "u333333",
     "user_info": "Test User",
     "score": 99.3
   }
 ]
}
```

人脸搜索 M:N 识别

待识别的图片中，存在多张人脸的情况下，支持在一个人脸库中，一次请求，同时返回图片中所有人脸的识别结果。

```
 public void MultiSearchDemo() {
var image = "取决于image_type参数，传入BASE64字符串或URL字符串或FACE_TOKEN字符串";

var imageType = "BASE64";

var groupIdList = "3,2";

// 调用人脸搜索 M:N 识别，可能会抛出网络等异常，请使用try/catch捕获
var result = client.MultiSearch(image, imageType, groupIdList);
Console.WriteLine(result);
// 如果有可选参数
var options = new Dictionary<string, object>{
    {"max_face_num", 3},
    {"match_threshold", 70},
    {"quality_control", "NORMAL"},
    {"liveness_control", "LOW"},
    {"max_user_num", 3}
};
// 带参数调用人脸搜索 M:N 识别
result = client.MultiSearch(image, imageType, groupIdList, options);
Console.WriteLine(result);
}
```

人脸搜索 M:N 识别 请求参数详情

参数名称	是否必选	类型	默认值	说明
image	是	string		图片信息(**总数据大小应小于10M**)，图片上传方式根据image_type来判断
image_type	是	string		图片类型 **BASE64**:图片的base64值，base64编码后的图片数据， 编码后的图片大小不超过2M；**URL**:图片的 URL地址(可能由于网络等原因导致下载图片时间过长)； **FACE_TOKEN**: 人脸图片的唯一标识，调用人脸检测接口时，会为每个人脸图片赋予一个唯一的FACE_TOKEN，同一张图片多次检测得到的FACE_TOKEN是同一个
group_id_list	是	string		从指定的group中进行查找 用逗号分隔，**上限10个**
max_face_num	否	string		最多处理人脸的数目 **默认值为1(仅检测图片中面积最大的那个人脸) 最大值10**
match_threshold	否	string		匹配阈值（设置阈值后，score低于此阈值的用户信息将不会返回） 最大100 最小0 默认80 **此阈值设置得越高，检索速度将会越快，推荐使用默认阈值80**
quality_control	否	string	NONE	图片质量控制 **NONE**: 不进行控制 **LOW**:较低的质量要求 **NORMAL**: 一般的质量要求 **HIGH**: 较高的质量要求 **默认 NONE**
liveness_control	否	string	NONE	活体检测控制 **NONE**: 不进行控制 **LOW**:较低的活体要求(高通过率 低攻击拒绝率) **NORMAL**: 一般的活体要求(平衡的攻击拒绝率, 通过率) **HIGH**: 较高的活体要求(高攻击拒绝率 低通过率) **默认NONE**
max_user_num	否	string		查找后返回的用户数量。返回相似度最高的几个用户，默认为1，最多返回20个。

人脸搜索 M:N 识别 返回数据参数详情

字段	必选	类型	说明
face_num	是	int	图片中的人脸数量
face_list	是	array	人脸信息列表
+face_token	是	string	人脸标志
+location	是	array	人脸在图片中的位置
++left	是	double	人脸区域离左边界的距离
++top	是	double	人脸区域离上边界的距离
++width	是	double	人脸区域的宽度
++height	是	double	人脸区域的高度
++rotation	是	int64	人脸框相对于竖直方向的顺时针旋转角，[-180,180]
+user_list	是	array	匹配的用户信息列表
++group_id	是	string	用户所属的group_id
++user_id	是	string	用户的user_id
++user_info	是	string	注册用户时携带的user_info
++score	是	float	用户的匹配得分 **80分以上可以判断为同一人，此分值对应万分之一误识率**

人脸搜索 M:N 识别 返回示例

```
{
  "error_code": 0,
  "error_msg": "SUCCESS",
  "log_id": 240483475,
  "timestamp": 1535533440,
  "cached": 0,
  "result": {
    "face_num": 2,
    "face_list": [
      {
        "face_token": "6fe19a6ee0c4233db9b5bba4dc2b9233",
        "location": {
          "left": 31.95568085,
          "top": 120.3764267,
          "width": 87,
          "height": 85,
          "rotation": -5
        },
        "user_list": [
          {
            "group_id": "group1",
            "user_id": "5abd24fd062e49bfa906b257ec40d284",
            "user_info": "userinfo1",
            "score": 69.85684967041
          },
          {
            "group_id": "group1",
            "user_id": "2abf89cffb31473a9948268fde9e1c3f",
            "user_info": "userinfo2",
            "score": 66.586112976074
          }
        ]
      },
      {
        "face_token": "fde61e9c074f48cf2bbb319e42634f41",
        "location": {
          "left": 219.4467773,
          "top": 104.7486954,
          "width": 81,
          "height": 77,
          "rotation": 3
        },
        "user_list": [
          {
            "group_id": "group1",
            "user_id": "088717532b094c3990755e91250adf7d",
            "user_info": "userinfo",
            "score": 65.154159545898
          }
        ]
      }
    ]
  }
}
```


人脸注册

用于从人脸库中新增用户，可以设定多个用户所在组，及组内用户的人脸图片，

典型应用场景：构建您的人脸库，如**会员人脸注册**，**已有用户补全人脸信息**等。

人脸库、用户组、用户、用户下的人脸**层级关系**如下所示：

```
|- 人脸库
 |- 用户组一
   |- 用户01
     |- 人脸
   |- 用户02
     |- 人脸
     |- 人脸
     ....
    ....
 |- 用户组二
 |- 用户组三
 |- 用户组四
 ....
```

关于人脸库的设置限制

- **每个appid对应一个人脸库，且不同appid之间，人脸库互不相通**；
- 每个人脸库下，可以创建多个用户组，用户组（group）数量**没有限制**；
- 每个用户组（group）下，可添加最多**无限**张人脸，**无限**个uid；
- 每个用户（uid）所能注册的最大人脸数量**20个**；

为了保证识别效果，请控制注册人脸的质量（通过`/detect`人脸检测接口判断），具体参数可详见下表所示：

质量判断

可通过人脸检测接口，基于以下字段和对应阈值，进行质量检测的判断，以保证人脸质量符合后续业务操作要求。

指标	字段与解释	推荐数值界限
遮挡范围	**occlusion**（0~1），0为无遮挡，1是完全遮挡 含有多个具体子字段，表示脸部多个部位 通常用作判断头发、墨镜、口罩等遮挡	left_eye：0.6, #左眼被遮挡的阈值 right_eye：0.6, #右眼被遮挡的阈值 nose：0.7, #鼻子被遮挡的阈值 mouth：0.7, #嘴巴被遮挡的阈值 left_check：0.8, #左脸颊被遮挡的阈值 right_check：0.8, #右脸颊被遮挡的阈值 chin_contour：0.6, #下巴被遮挡阈值
模糊度范围	**Blur**（0~1），0是最清晰，1是最模糊	小于0.7
光照范围	**illumination**（0~255） 脸部光照的灰度值，0表示光照不好 以及对应客户端SDK中，YUV的Y分量	大于40
姿态角度	**Pitch**：三维旋转之俯仰角度[-90(上), 90(下)] **Roll**：平面内旋转角[-180(逆时针), 180(顺时针)] **Yaw**：三维旋转之左右旋转角[-90(左), 90(右)]	分别小于20度
人脸完整度	**completeness**（0或1），0为人脸溢出图像边界，1为人脸都在图像边界内	视业务逻辑判断
人脸大小	人脸部分的大小 建议长宽像素值范围：80*80~200*200	人脸部分不小于**100*100**像素

```
 public void UserAddDemo() {
var image = "取决于image_type参数，传入BASE64字符串或URL字符串或FACE_TOKEN字符串";

var imageType = "BASE64";

var groupId = "group1";

var userId = "user1";

// 调用人脸注册，可能会抛出网络等异常，请使用try/catch捕获
var result = client.UserAdd(image, imageType, groupId, userId);
Console.WriteLine(result);
// 如果有可选参数
var options = new Dictionary<string, object>{
    {"user_info", "user's info"},
    {"quality_control", "NORMAL"},
    {"liveness_control", "LOW"},
    {"action_type", "REPLACE"}
};
// 带参数调用人脸注册
result = client.UserAdd(image, imageType, groupId, userId, options);
Console.WriteLine(result);
}
```

人脸注册 请求参数详情

参数名称	是否必选	类型	默认值	说明
image	是	string		图片信息(总数据大小应小于10M)，图片上传方式根据image_type来判断。注：组内每个uid下的人脸图片数目上限为20张
image_type	是	string		图片类型 **BASE64**:图片的base64值，base64编码后的图片数据， 编码后的图片大小不超过2M；**URL**:图片的 URL地址(可能由于网络等原因导致下载图片时间过长)； **FACE_TOKEN**: 人脸图片的唯一标识，调用人脸检测接口时，会为每个人脸图片赋予一个唯一的FACE_TOKEN，同一张图片多次检测得到的FACE_TOKEN是同一个
group_id	是	string		用户组id（由数字、字母、下划线组成），长度限制128B
user_id	是	string		用户id（由数字、字母、下划线组成），长度限制128B
user_info	否	string		用户资料，长度限制256B
quality_control	否	string	NONE	图片质量控制 **NONE**: 不进行控制 **LOW**:较低的质量要求 **NORMAL**: 一般的质量要求 **HIGH**: 较高的质量要求 **默认 NONE**
liveness_control	否	string	NONE	活体检测控制 **NONE**: 不进行控制 **LOW**:较低的活体要求(高通过率 低攻击拒绝率) **NORMAL**: 一般的活体要求(平衡的攻击拒绝率, 通过率) **HIGH**: 较高的活体要求(高攻击拒绝率 低通过率) **默认NONE**
action_type	否	string	APPEND	操作方式 APPEND: 当user_id在库中已经存在时，对此user_id重复注册时，新注册的图片默认会追加到该user_id下,REPLACE：当对此user_id重复注册时,则会用新图替换库中该user_id下所有图片,默认使用APPEND

人脸注册 返回数据参数详情

字段	必选	类型	说明
log_id	是	uint64	请求标识码，随机数，唯一
face_token	是	string	人脸图片的唯一标识
location	是	array	人脸在图片中的位置
+left	是	double	人脸区域离左边界的距离
+top	是	double	人脸区域离上边界的距离
+width	是	double	人脸区域的宽度
+height	是	double	人脸区域的高度
+rotation	是	int64	人脸框相对于竖直方向的顺时针旋转角，[-180,180]

人脸注册 返回示例

```
{
 "face_token": "2fa64a88a9d5118916f9a303782a97d3",
 "location": {
     "left": 117,
     "top": 131,
     "width": 172,
     "height": 170,
     "rotation": 4
 }
}
```

人脸更新

用于对人脸库中指定用户，更新其下的人脸图像。

> **说明：**针对一个uid执行更新操作，新上传的人脸图像将覆盖此uid原有所有图像。

```
 public void UserUpdateDemo() {
var image = "取决于image_type参数，传入BASE64字符串或URL字符串或FACE_TOKEN字符串";

var imageType = "BASE64";

var groupId = "group1";

var userId = "user1";

// 调用人脸更新，可能会抛出网络等异常，请使用try/catch捕获
var result = client.UserUpdate(image, imageType, groupId, userId);
Console.WriteLine(result);
// 如果有可选参数
var options = new Dictionary<string, object>{
    {"user_info", "user's info"},
    {"quality_control", "NORMAL"},
    {"liveness_control", "LOW"},
    {"action_type", "REPLACE"}
};
// 带参数调用人脸更新
result = client.UserUpdate(image, imageType, groupId, userId, options);
Console.WriteLine(result);
}
```

人脸更新 请求参数详情

参数名称	是否必选	类型	默认值	说明
image	是	string		图片信息(**总数据大小应小于10M**)，图片上传方式根据image_type来判断
image_type	是	string		图片类型 **BASE64**:图片的base64值，base64编码后的图片数据， 编码后的图片大小不超过2M；**URL**:图片的 URL地址(可能由于网络等原因导致下载图片时间过长)；**FACE_TOKEN**: 人脸图片的唯一标识，调用人脸检测接口时，会为每个人脸图片赋予一个唯一的FACE_TOKEN，同一张图片多次检测得到的FACE_TOKEN是同一个
group_id	是	string		更新指定groupid下uid对应的信息
user_id	是	string		用户id（由数字、字母、下划线组成），长度限制128B
user_info	否	string		用户资料，长度限制256B
quality_control	否	string	NONE	图片质量控制 **NONE**: 不进行控制 **LOW**:较低的质量要求 **NORMAL**: 一般的质量要求 **HIGH**: 较高的质量要求 **默认 NONE**
liveness_control	否	string	NONE	活体检测控制 **NONE**: 不进行控制 **LOW**:较低的活体要求(高通过率 低攻击拒绝率) **NORMAL**: 一般的活体要求(平衡的攻击拒绝率, 通过率) **HIGH**: 较高的活体要求(高攻击拒绝率 低通过率) **默认NONE**
action_type	否	string	UPDATE	操作方式 **UPDATE**: 当user_id在库中已经存在时，对此user_id重复注册时，新注册的图片默认会追加到该user_id下,**REPLACE**：当对此user_id重复注册时,则会用新图替换库中该user_id下所有图片,默认使用**UPDATE**

人脸更新 返回数据参数详情

字段	必选	类型	说明
log_id	是	uint64	请求标识码，随机数，唯一
face_token	是	string	人脸图片的唯一标识
location	是	array	人脸在图片中的位置
+left	是	double	人脸区域离左边界的距离
+top	是	double	人脸区域离上边界的距离
+width	是	double	人脸区域的宽度
+height	是	double	人脸区域的高度
+rotation	是	int64	人脸框相对于竖直方向的顺时针旋转角，[-180,180]

人脸更新 返回示例

```
{
 "face_token": "2fa64a88a9d5118916f9a303782a97d3",
 "location": {
    "left": 117,
    "top": 131,
    "width": 172,
    "height": 170,
    "rotation": 4
  }
}
```

人脸删除

用于从人脸库中删除一个用户。

人脸删除注意事项：

- 删除的内容，包括用户所有图像和身份信息；
- 如果一个uid存在于多个用户组内，将会同时将从各个组中把用户删除
- 如果指定了group_id，则只删除此group下的uid相关信息

```
 public void FaceDeleteDemo() {
 var userId = "user1";

 var groupId = "group1";

 var faceToken = "face_token_23123";

 // 调用人脸删除，可能会抛出网络等异常，请使用try/catch捕获
 var result = client.FaceDelete(userId, groupId, faceToken);
 Console.WriteLine(result);
}
```

人脸删除 请求参数详情

参数名称	是否必选	类型	说明
log_id	是	uint64	请求标识码，随机数，唯一
user_id	是	string	用户id（由数字、字母、下划线组成），长度限制128B
group_id	是	string	用户组id（由数字、字母、下划线组成），长度限制128B
face_token	是	string	需要删除的人脸图片token，（由数字、字母、下划线组成）长度限制64B

人脸删除 返回数据参数详情

字段	是否必选	类型	说明
log_id	是	uint64	请求唯一标识码，随机数

人脸删除 返回示例

```
 // 删除成功
{
  "error_code": 0,
  "log_id": 73473737,
}
// 删除发生错误
{
  "error_code": 223106,
  "log_id": 1382953199,
  "error_msg": "face is not exist"
}
```

用户信息查询

获取人脸库中某个用户的信息(user_info信息和用户所属的组)。

```
 public void UserGetDemo() {
 var userId = "user1";

 var groupId = "group1";

 // 调用用户信息查询，可能会抛出网络等异常，请使用try/catch捕获
 var result = client.UserGet(userId, groupId);
 Console.WriteLine(result);
}
```

用户信息查询 请求参数详情

参数名称	是否必选	类型	说明
user_id	是	string	用户id（由数字、字母、下划线组成），长度限制128B
group_id	是	string	用户组id（由数字、字母、下划线组成），长度限制128B

用户信息查询 返回数据参数详情

字段	必选	类型	说明
log_id	是	uint64	请求标识码，随机数，唯一
user_list	是	array	查询到的用户列表
+user_info	是	string	用户资料，被查询用户的资料
+group_id	是	string	用户组id，被查询用户的所在组

用户信息查询 返回示例

```
{
 "user_list": [
    {
        "user_info": "user info ...",
        "group_id": "gid1"
    },
    {
        "user_info": "user info2 ...",
        "group_id": "gid2"
    }
 ]
}
```

获取用户人脸列表

用于获取一个用户的全部人脸列表。

```
 public void FaceGetlistDemo() {
 var userId = "user1";

 var groupId = "group1";

 // 调用获取用户人脸列表，可能会抛出网络等异常，请使用try/catch捕获
 var result = client.FaceGetlist(userId, groupId);
 Console.WriteLine(result);
}
```

获取用户人脸列表 请求参数详情

参数名称	是否必选	类型	说明
user_id	是	string	用户id（由数字、字母、下划线组成），长度限制128B
group_id	是	string	用户组id（由数字、字母、下划线组成），长度限制128B

获取用户人脸列表 返回数据参数详情

字段	必选	类型	说明
log_id	是	uint64	请求标识码，随机数，唯一
face_list	是	array	人脸列表
+face_token	是	string	人脸图片的唯一标识
+ctime	是	string	人脸创建时间

获取用户人脸列表 返回示例

```
{
 "face_list": [
    {
        "face_token": "fid1",
        "ctime": "2018-01-01 00:00:00"
    },
    {
        "face_token": "fid2",
        "ctime": "2018-01-01 10:00:00"
    }
 ]
}
```

获取用户列表

用于查询指定用户组中的用户列表。

```
 public void GroupGetusersDemo() {
var groupId = "group1";

// 调用获取用户列表，可能会抛出网络等异常，请使用try/catch捕获
var result = client.GroupGetusers(groupId);
Console.WriteLine(result);
// 如果有可选参数
var options = new Dictionary<string, object>{
    {"start", 0},
    {"length", 50}
};
// 带参数调用获取用户列表
result = client.GroupGetusers(groupId, options);
Console.WriteLine(result);
}
```

获取用户列表 请求参数详情

参数名称	是否必选	类型	默认值	说明
group_id	是	string		用户组id（由数字、字母、下划线组成），长度限制128B
start	否	string	0	默认值0，起始序号
length	否	string	100	返回数量，默认值100，最大值1000

获取用户列表 返回数据参数详情

字段	必选	类型	说明
user_id_list	是	array	用户ID列表

获取用户列表 返回示例

```
{
 "user_id_list": [
    "uid1",
    "uid2"
 ]
}
```

复制用户

用于将已经存在于人脸库中的用户**复制到一个新的组**。

```
 public void UserCopyDemo() {
 var userId = "user1";

 // 调用复制用户，可能会抛出网络等异常，请使用try/catch捕获
 var result = client.UserCopy(userId);
 Console.WriteLine(result);
 // 如果有可选参数
 var options = new Dictionary<string, object>{
    {"src_group_id", "11111"},
    {"dst_group_id", "222222"}
 };
 // 带参数调用复制用户
 result = client.UserCopy(userId, options);
 Console.WriteLine(result);
}
```

复制用户 请求参数详情

参数名称	是否必选	类型	说明
user_id	是	string	用户id（由数字、字母、下划线组成），长度限制128B
src_group_id	否	string	从指定组里复制信息
dst_group_id	否	string	需要添加用户的组id

复制用户 返回数据参数详情

字段	必选	类型	说明
log_id	是	id	log_id

复制用户 返回示例

```
 // 正确返回值
{
  "error_code": 0,
 "log_id": 3314921889,
}
// 发生错误时返回值
{
 "error_code": 216100,
 "log_id": 3111284097,
 "error_msg": "already add"
}
```

删除用户

用于将用户从某个组中删除。

```
 public void UserDeleteDemo() {
 var groupId = "group1";

 var userId = "user1";

 // 调用删除用户，可能会抛出网络等异常，请使用try/catch捕获
 var result = client.UserDelete(groupId, userId);
 Console.WriteLine(result);
}
```

删除用户 请求参数详情

参数名称	是否必选	类型	说明
group_id	是	string	用户组id（由数字、字母、下划线组成），长度限制128B
user_id	是	string	用户id（由数字、字母、下划线组成），长度限制128B

删除用户 返回数据参数详情

字段	是否必选	类型	说明
log_id	是	uint64	请求唯一标识码，随机数

删除用户 返回示例

```
 // 正确返回值
{

 "error_code": 0,
 "log_id": 3314921889,
}
// 发生错误时返回值
{
 "error_code":  223103,
 "log_id": 815967402,
 "error_msg": "user is not exist"
}
```

创建用户组

用于创建一个空的用户组，如果用户组已存在 则返回错误。

```
 public void GroupAddDemo() {
 var groupId = "group1";

 // 调用创建用户组，可能会抛出网络等异常，请使用try/catch捕获
 var result = client.GroupAdd(groupId);
 Console.WriteLine(result);
}
```

创建用户组 请求参数详情

参数名称	是否必选	类型	说明
group_id	是	string	用户组id（由数字、字母、下划线组成），长度限制128B

创建用户组 返回数据参数详情

字段	必选	类型	说明
log_id	是	uint64	请求标识码，随机数，唯一

创建用户组 返回示例

```
 {

 "error_code": 0,
 "log_id": 3314921889,
}
// 发生错误时返回值
{
 "error_code":  223101,
 "log_id": 815967402,
 "error_msg": " group is already exist"
}
```

删除用户组

删除用户组下所有的用户及人脸，如果组不存在 则返回错误。

```
 public void GroupDeleteDemo() {
 var groupId = "group1";

 // 调用删除用户组，可能会抛出网络等异常，请使用try/catch捕获
 var result = client.GroupDelete(groupId);
 Console.WriteLine(result);
}
```

删除用户组 请求参数详情

参数名称	是否必选	类型	说明
group_id	是	string	用户组id（由数字、字母、下划线组成），长度限制128B

删除用户组 返回数据参数详情

字段	必选	类型	说明
log_id	是	uint64	请求标识码，随机数，唯一

删除用户组 返回示例

```
 // 正确返回值
{

  "error_code":0,
  "log_id": 3314921889,
}
// 发生错误时返回值
{
  "error_code": 223100,
  "log_id": 815967402,
  "error_msg": " group is not exist"
}
```

组列表查询

用于查询用户组的列表。

```
 public void GroupGetlistDemo() {
 // 调用组列表查询，可能会抛出网络等异常，请使用try/catch捕获
 var result = client.GroupGetlist();
 Console.WriteLine(result);
 // 如果有可选参数
 var options = new Dictionary<string, object>{
    {"start", 0},
    {"length", 50}
 };
 // 带参数调用组列表查询
 result = client.GroupGetlist(options);
 Console.WriteLine(result);
}
```

组列表查询 请求参数详情

参数名称	是否必选	类型	默认值	说明
start	否	string	0	默认值0，起始序号
length	否	string	100	返回数量，默认值100，最大值1000

组列表查询 返回数据参数详情

字段	必选	类型	说明
group_id_list	是	array	group

组列表查询 返回示例

```
 {
   "group_id_list": [
      "gid1",
      "gid2"
   ]
}
```

身份验证

质量检测（可选）活体检测（可选）公安验证（必选）

```
 public void PersonVerifyDemo() {
var image = "取决于image_type参数，传入BASE64字符串或URL字符串或FACE_TOKEN字符串";

var imageType = "BASE64";

var idCardNumber = "110233112299822211";

var name = "张三";

// 调用身份验证，可能会抛出网络等异常，请使用try/catch捕获
var result = client.PersonVerify(image, imageType, idCardNumber, name);
Console.WriteLine(result);
// 如果有可选参数
var options = new Dictionary<string, object>{
    {"quality_control", "NORMAL"},
    {"liveness_control", "LOW"}
};
// 带参数调用身份验证
result = client.PersonVerify(image, imageType, idCardNumber, name, options);
Console.WriteLine(result);
}
```

身份验证 请求参数详情

参数名称	是否必选	类型	默认值	说明
image	是	string		图片信息(**总数据大小应小于10M**)，图片上传方式根据image_type来判断
image_type	是	string		图片类型 **BASE64**:图片的base64值，base64编码后的图片数据， 编码后的图片大小不超过2M；**URL**:图片的 URL地址(可能由于网络等原因导致下载图片时间过长)；**FACE_TOKEN**: 人脸图片的唯一标识，调用人脸检测接口时，会为每个人脸图片赋予一个唯一的FACE_TOKEN，同一张图片多次检测得到的FACE_TOKEN是同一个
id_card_number	是	string		身份证号（真实身份证号号码）
name	是	string		utf8，姓名（真实姓名，和身份证号匹配）
quality_control	否	string	NONE	图片质量控制 **NONE**: 不进行控制 **LOW**:较低的质量要求 **NORMAL**: 一般的质量要求 **HIGH**: 较高的质量要求 **默认 NONE**
liveness_control	否	string	NONE	活体检测控制 **NONE**: 不进行控制 **LOW**:较低的活体要求(高通过率 低攻击拒绝率) **NORMAL**: 一般的活体要求(平衡的攻击拒绝率, 通过率) **HIGH**: 较高的活体要求(高攻击拒绝率 低通过率) **默认NONE**

身份验证 返回数据参数详情

参数	必须	类型	说明
log_id	是	uint64	日志id
score	是	float	与公安小图相似度可能性，用于验证生活照与公安小图是否为同一人，有正常分数时为[0~1]，推荐阈值0.8，超过即判断为同一人

身份验证 返回示例

```
{
 "score": 44.3,
}
```

语音校验码接口

此接口主要用于生成随机码，用于视频的语音识别校验使用，以判断视频的即时性，而非事先录制的，提升作弊的难度。

```
 public void VideoSessioncodeDemo() {
// 调用语音校验码接口，可能会抛出网络等异常，请使用try/catch捕获
var result = client.VideoSessioncode();
Console.WriteLine(result);
// 如果有可选参数
var options = new Dictionary<string, object>{
    {"appid", "223245"}
};
// 带参数调用语音校验码接口
result = client.VideoSessioncode(options);
Console.WriteLine(result);
}
```

语音校验码接口 请求参数详情

参数名称	是否必选	类型	说明
appid	否	string	百度云创建应用时的唯一标识ID

语音校验码接口 返回数据参数详情

字段	必选	类型	说明
session_id	是	string	语音校验码会话id
code	是	string	语音验证码，数字形式，3~6位数字

语音校验码接口 返回示例

```
{
 "err_no": 0,
 "err_msg": "SUCCESS",
 "result": {
   "session_id": "S59faeeebb9111890355690",
   "code": "9940"
 },
 "timestamp": 1509617387,
 "cached": 0,
 "serverlogid": "0587756642"
}
```

人脸对比

接口能力

- **两张人脸图片相似度对比**：比对两张图片中人脸的相似度，并返回相似度分值；
- **多种图片类型**：支持**生活照**、**证件照**、**身份证芯片照**、**带网纹照**四种类型的人脸对比；
- **活体检测**：基于图片中的破绽分析，判断其中的人脸是否为**二次翻拍**（举例：如用户A用手机拍摄了一张包含人脸的图片一，用户B翻拍了图片一得到了图片二，并用图片二伪造成用户A去进行识别操作，这种情况普遍发生在金融开户、实名认证等环节。）；

- **质量检测**：返回模糊、光照等质量检测信息，用于辅助判断图片是否符合识别要求；

业务应用

用于比对多张图片中的人脸相似度并返回两两比对的得分，可用于判断两张脸是否是同一人的可能性大小。

典型应用场景：如**人证合一验证**，**用户认证**等，可与您现有的人脸库进行比对验证。

```
  public string ReadImg(string img)
{
    return Convert.ToBase64String(File.ReadAllBytes(img));
}

public void demo()
{
    var faces = new JArray
    {
        new JObject
        {
            {"image", ReadImg("ym1.jpeg")},
            {"image_type", "BASE64"},
            {"face_type", "LIVE"},
            {"quality_control", "LOW"},
            {"liveness_control", "NONE"},
        },
        new JObject
        {
            {"image", ReadImg("ym2.jpeg")},
            {"image_type", "BASE64"},
            {"face_type", "LIVE"},
            {"quality_control", "LOW"},
            {"liveness_control", "NONE"},
        }
    };

    var result = client.Match(faces);
    Console.Write(result);
}
```

请求参数

参数为 JArray，JArray中的元素为JObject

JObject 中的参数：

参数	必选	类型	说明
image	是	string	图片信息(**总数据大小应小于10M**)，图片上传方式根据image_type来判断。 **两张图片通过json格式上传，格式参考表格下方示例**
image_type	是	string	图片类型 **BASE64**:图片的base64值，base64编码后的图片数据， 编码后的图片大小不超过2M；**URL**:图片的 URL地址(可能由于网络等原因导致下载图片时间过长)；**FACE_TOKEN**: 人脸图片的唯一标识，调用人脸检测接口时，会为每个人脸图片赋予一个唯一的FACE_TOKEN，同一张图片多次检测得到的FACE_TOKEN是同一个。
face_type	否	string	人脸的类型 **LIVE**表示生活照：通常为手机、相机拍摄的人像图片、或从网络获取的人像图片等，**IDCARD**表示身份证芯片照：二代身份证内置芯片中的人像照片， **WATERMARK**表示带水印证件照：一般为带水印的小图，如公安网小图 **CERT**表示证件照片：如拍摄的身份证、工卡、护照、学生证等证件图片 默认LIVE
quality_control	否	string	图片质量控制 **NONE**: 不进行控制 **LOW**:较低的质量要求 **NORMAL**: 一般的质量要求 **HIGH**: 较高的质量要求 **默认 NONE**
liveness_control	否	string	活体检测控制 **NONE**: 不进行控制 **LOW**:较低的活体要求(高通过率 低攻击拒绝率) **NORMAL**: 一般的活体要求(平衡的攻击拒绝率, 通过率) **HIGH**: 较高的活体要求(高攻击拒绝率 低通过率) **默认NONE**

最终上传的内容如下：

```
[
{
    "image": "sfasq35sadvsvqwr5q...",
    "image_type": "BASE64",
    "face_type": "LIVE",
    "quality_control": "LOW",
    "liveness_control": "HIGH"
},
{
    "image": "sfasq35sadvsvqwr5q...",
    "image_type": "BASE64",
    "face_type": "IDCARD",
    "quality_control": "LOW",
    "liveness_control": "HIGH"
}
]
```

返回参数

参数名	必选	类型	说明
score	是	float	人脸相似度得分
face_list	是	array	人脸信息列表
+face_token	是	string	人脸的唯一标志

返回示例

```
{
 "score": 44.3,
 "face_list": [  //返回的顺序与传入的顺序保持一致
    {
       "face_token": "fid1"
    },
    {
       "face_token": "fid2"
    }
 ]
}
```

关于活体检测faceliveness的判断阈值选择，可参考以下数值信息：

拒绝率（TRR）	误拒率（FRR）	通过率（TAR）	阈值（Threshold）
0.90325733	0.1%	99.9%	0.022403
0.96254072	0.5%	99.5%	0.393241（**推荐**）
0.97557003	1%	99%	0.649192
0.98990228	2%	98%	0.933801
0.99446254	3%	97%	0.973637
0.99641694	4%	96%	0.988479
0.99739414	5%	95%	0.994058

关于以上数值的概念介绍：

- **拒绝率（TRR）**：如99%，代表100次作弊假体攻击，会有99次被拒绝。
- **误拒率（FRR）**：如0.5%，指1000次真人请求，会有5次因为活体分数低于阈值被错误拒绝。
- **通过率（TAR）**：如99%，指100次真人请求，会有99次因为活体分数高于阈值而通过。
- **阈值（Threshold）**：高于此数值，则可判断为活体。

在线活体检测

人脸基础信息，人脸质量检测，基于图片的活体检测

```
  public string ReadImg(string img)
{
    return Convert.ToBase64String(File.ReadAllBytes(img));
}

public void Demo() {
    var faces = new JArray
    {
        new JObject
        {
            {"image", ReadImg("/ym1.jpeg")},
            {"image_type", "BASE64"},
            {"face_field", "age"},
        },
        new JObject
        {
            {"image", ReadImg("/ym2.jpeg")},
            {"image_type", "BASE64"}
        }
    };
    var result = client.Faceverify(faces);
    Console.WriteLine(result);
}
```

在线活体检测 请求参数详情

参数名称	是否必选	类型	说明
image	是	String	图片信息(**总数据大小应小于10M**)，图片上传方式根据image_type来判断
image_type	是	String	图片类型 **BASE64**:图片的base64值，base64编码后的图片数据， 编码后的图片大小不超过2M；**URL**:图片的 URL地址(可能由于网络等原因导致下载图片时间过长)；**FACE_TOKEN**: 人脸图片的唯一标识，调用人脸检测接口时，会为每个人脸图片赋予一个唯一的FACE_TOKEN，同一张图片多次检测得到的FACE_TOKEN是同一个
face_fields	否	String	包括age,expression,faceshape,gender,glasses,landmark,quality,facetype信息，逗号分隔，默认只返回face_token、活体数、人脸框、概率和旋转角度。

在线活体检测 返回数据参数详情

参数	类型	是否必须	说明
log_id	是	uint64	请求唯一标识码，随机数
face_liveness	是	float	活体分数值
thresholds	是	array	由服务端返回最新的阈值数据（随着模型的优化，阈值可能会变化），可以作为活体判断的依据。frr_1e-4：万分之一误识率的阈值；frr_1e-3：千分之一误识率的阈值；frr_1e-2：百分之一误识率的阈值。误识率越底，准确率越高。
face_list	是	array	每张图片的详细信息描述，如果只上传一张图片，则只返回一个结果。

在线活体检测 返回示例

```
{
  "thresholds": {
    "frr_1e-4": 0.05,
```

```
        "frr_1e-3": 0.3,
        "frr_1e-2": 0.9
    },
    "face_liveness": 0.05532243927,
    "face_list": [
        {
            "face_token": "df46f7c7db4aa09a093c26fb8d1a8d44",
            "location": {
                "left": 328.9026489,
                "top": 97.16340637,
                "width": 162,
                "height": 154,
                "rotation": 32
            },
            "face_probability": 1,
            "angle": {
                "yaw": 10.16196251,
                "pitch": 2.244354248,
                "roll": 33.82199097
            },
            "liveness": {
                "faceliveness": 0.004187555984,
                "livemapscore": 0.04492170034
            },
            "age": 23
        },
        {
            "face_token": "901d2c64274fccd687d311a6e6110a01",
            "location": {
                "left": 411.4876404,
                "top": 166.3593445,
                "width": 329,
                "height": 308,
                "rotation": 45
            },
            "face_probability": 0.9194830656,
            "angle": {
                "yaw": -1.716423035,
                "pitch": 7.344647408,
                "roll": 45.79914856
            },
            "liveness": {
                "faceliveness": 0.0001665892196,
                "livemapscore": 0.001787073661
            },
            "age": 23
        },
        {
            "face_token": "7d57e36981c48b4946eb97c8d838b02a",
            "location": {
                "left": 161.4559937,
                "top": 199.8726501,
                "width": 218,
                "height": 201,
                "rotation": -1
            },
            "face_probability": 1,
            "angle": {
                "yaw": -8.187754631,
                "pitch": 6.973727226,
                "roll": -1.25429821
            },
```

```
            "liveness": {
                "faceliveness": 0.02942637168,
                "livemapscore": 0.05532243927
            },
            "age": 23
        }
    ]
}
```

人脸实名认证V4

能力介绍

1. 业务能力

- **质量检测（可选）**：判断图片中是否包含人脸，以及人脸在姿态、遮挡、模糊、光照等方面是否符合识别条件。
- **活体检测（可选）**：基于图片中的破绽分析，判断其中的人脸是否为**二次翻拍**（举例：如用户A用手机拍摄了一张包含人脸的图片一，用户B翻拍了图片一得到了图片二，并用图片二伪造成用户A去进行识别操作，这种情况普遍发生在金融开户、实名认证等环节）。
- **图片加密及风控（可选）**：当配合增强级采集SDK、增强级安全加固采集SDK、金融级采集SDK、金融级安全加固采集SDK版本使用，对采集SDK输出的加密图片进行解密（加密传输可以有效避免第三方非法黑产绕过APP模拟请求攻击云端接口的行为，Eg：脚本攻击等）；以及结合百度安全实验室大数据风控能力，对采集SDK的发起端设备进行风控识别，辨别是否为风险设备，Eg：ROM注入、视频劫持等；
- **人脸实名认证（必选）**：基于姓名和身份证号，调取公安权威数据源人脸图，将当前获取的人脸图片，与公安数据源人脸图进行对比，得出比对分数，并基于此进行业务判断是否为同一人。由于公安数据源人脸图具有最权威的身份证明作用，故对用户本人的验证结果可信度也最为合理。

2. 业务逻辑

- 上述能力，人脸实名认证能力为必选能力，质量检测、活体检测、图片加密及风控为可选能力，验证顺序为`人脸质量检测->活体检测->人脸实名认证`。
- **如选择了质量检测和活体检测能力，则有任意一个条件不通过，整个请求流程终止，并返回错误码，描述具体的不符合信息**。
- 基于此顺序串行验证逻辑，可以避免大量不符合条件的请求流转到人脸实名认证，节约您的成本。

3. 推荐阈值

- 此接口使用的对比算法，针对带水纹证件照采用了专项的模型处理，可保证水纹信息的影响降到尽可能低。
- 如比对成功，最终返回的有效数据为一个**对比分值**，在0~100之间，您可以设定具体的阈值来判断是否验证通过。
- **人证相似度的推荐阈值为80，对应的误识率为万分之一**。

```
 public void Demo() {

 // 必填参数
var idCardNumber = "";
var name = "";
 var image = ""

 // 选填参数
var options = new Dictionary<string, object>{
    {"xxx", "xxx"},
};

// 调用接口
result = client.faceMingJingVerify(idCardNumber, name, image, options);
Console.WriteLine(result);
}
```

请求参数

参数	必选	类型	说明
app	否	string	APP端类型，配合采集SDK使用时须传入 **ios**：iOS端采集SDK **android**：安卓端采集SDK
sec_level	否	string	SDK安全级别，配合采集SDK使用时须传入，默认common **common**：配合4.1/4.1.5版本采集SDK，人脸图片未进行加密处理 **lite**: 配合5.2版本SDK
skey	否	string	使用5.2版本SDK请求时必填 **skey**：从SDK获取的密钥信息skey
x_device_id	否	string	使用5.2版本SDK请求时必填 **deviceId**：从SDK 获取的密钥信息deviceId
data	否	string	使用5.2版本SDK请求时必填， SDK输出的加密数据
id_card_number	是	string	身份证件号
name	是	string	姓名(需要是 utf8 编码)
liveness_control	否	string	活体控制参数 **NONE**: 不进行控制 **LOW**:较低的活体要求(高通过率 低攻击拒绝率) **NORMAL**: 一般的活体要求(平衡的攻击拒绝率, 通过率) **HIGH**: 较高的活体要求(高攻击拒绝率 低通过率) **默认为NONE**
spoofing_control	否	string	合成图控制参数 **NONE**: 不进行控制 **LOW**:较低的合成图阈值数值，由于合成图判定逻辑为大于阈值视为合成图攻击，该项代表**低通过率、高攻击拒绝率** **NORMAL**: 一般的合成图阈值数值，由于合成图判定逻辑为大于阈值视为合成图攻击，该项代表**平衡的攻击拒绝率, 通过率** **HIGH**: 较高的合成图阈值数值，由于合成图判定逻辑为大于阈值视为合成图攻击，该项代表**高通过率、低攻击拒绝率**) **默认为NONE**
quality_control	否	string	质量控制参数 **NONE**: 不进行控制 **LOW**:较低的质量要求 **NORMAL**: 一般的质量要求 **HIGH**: 较高的质量要求 **默认为NONE**
image	是	string	图片信息(数据大小应小于10M 分辨率应小于1920*1080)，**5.2版本SDK请求时已包含在加密数据data中，无需额外传入**
image_type	否	string	图片类型 **BASE64**：图片的base64值 **URL**：图片的 URL **FACE_TOKEN**：人脸标识 **默认 BASE64**

返回参数

参数	类型	说明
log_id	number	调用的日志id
result	jsonObject	认证返回的结果
+score	float	与公安数据源人脸图相似度可能性，用于验证生活照与公安数据源人脸图是否为同一人，有正常分数时为[0~100]，推荐阈值80，超过即判断为同一人
+verify_status	number	认证状态，取值如下： 0 ： 正常 1 ： 身份证号与姓名不匹配或该身份证号不存在 2 ： 公安网图片不存在或质量过低
dec_image	string	对SDK传入的加密图片进行解密。 仅APP场景且进行了图片加密时，此参数返回解密后的人脸图片信息
risk_level	string	判断设备是否发生过风险行为来判断风险级别，取值（数值由高到低）： 1 – 高危 2 – 嫌疑 3 – 普通 4 – 正常
risk_tag	string	风险标签，若判断为有风险，则会有风险标签json 数组告知风险类型 例如：general_inject

返回示例


```
{
  "log_id": 1370579072568000512,
  "result": {
     "score": 40.884,
     "verify_status": 0
  },
  "dec_image": "/9j/4AAQSkZJRgABAgAAAQABAAD",
  "risk_level": "3",
  "risk_tag": [
     "若判断为有风险，则会有风险标签json 数组告知风险类型，如：general_inject"
  ]
}
```


人脸比对V4

能力介绍

1. 接口能力

- **两张人脸图片相似度对比**：对比两张图片中人脸的相似度，并返回相似度分值；
- **多种图片类型**：支持**生活照**、**证件照**、**身份证芯片照**、**带网纹照**四种类型的人脸对比；
- **活体检测控制**：基于图片中的破绽分析，判断其中的人脸是否为**二次翻拍**（举例：如用户A用手机拍摄了一张包含人脸的图片一，用户B翻拍了图片一得到了图片二，并用图片二伪造成用户A去进行识别操作，这种情况普遍发生在金融开户、实名认证等环节。）；
- **质量检测控制**：分析图片的中人脸的模糊度、角度、光照强度等特征，判断图片质量；
- **图片加密及风控：**配合增强级、金融级采集SDK使用
  - 对采集SDK输出的加密图片进行解密（加密传输可以有效避免第三方非法黑产**绕过APP模拟请求攻击云端接口的行为**，Eg：脚本攻击等）；
  - 以及结合百度安全实验室大数据风控能力，对采集SDK的发起端设备进行风控识别，辨别是否为**风险设备**，Eg：ROM注入、视频劫持等；

2. 业务应用

- 用于对比多张图片中的人脸相似度并返回两两对比的得分，可用于判断两张脸是否是同一人的可能性大小。

- 典型应用场景：如**人证合一验证**，**用户认证**等，可与您现有的人脸库进行对比验证。

```
 public void Demo() {

  // 必填参数
 var image = "";
 var imageType = "";
  var registerImage = "";
  var registerImageType = "";

  // 选填参数
 var options = new Dictionary<string, object>{
     {"xxx", "xxx"},
 };

 // 调用接口
 result = client.faceMingJingMatch(image, imageType, registerImage, registerImageType, options);
 Console.WriteLine(result);
}
```

请求参数

参数	必选	类型	说明
app	否	string	APP类型，配合采集SDK使用时须传入 **ios**：iOS端采集SDK **android**：安卓端采集SDK
sec_level	否	string	SDK安全级别，配合采集SDK使用时须传入，默认common **common**：配合4.1/4.1.5版本SDK使用，人脸图片未进行加密处理 **lite**：配合5.2版本SDK使用
skey	否	string	使用5.2版本SDK请求时必填 从SDK获取的密钥信息
x_device_id	否	string	使用5.2版本SDK请求时必填 从SDK获取的密钥信息
data	否	string	使用5.2版本SDK请求时必填 SDK输出的加密数据
image	是	string	图片信息(数据大小应小于10M 分辨率应小于1920*1080)，***5.2版本SDK请求时已包含在加密数据data中，无需额外传入**
image_type	是	string	图片类型 **BASE64**：图片的base64值 **URL**：图片的URL **FACE_TOKEN**：人脸标识 默认 BASE64
face_type	否	string	人脸的类型 **LIVE**：表示生活照：通常为手机、相机拍摄的人像图片、或从网络获取的人像图片等 **IDCARD**：表示身份证芯片照：二代身份证内置芯片中的人像照片 **WATERMARK**：表示带水印证件照：一般为带水印的小图，如公安网小图 **CERT**：表示证件照片：如拍摄的身份证、工卡、护照、学生证等证件图片 **INFRARED** 表示红外照片：使用红外相机拍摄的照片 默认LIVE
quality_control	否	string	质量控制 **NONE**: 不进行控制 **LOW**:较低的质量要求 **NORMAL**: 一般的质量要求 **HIGH**: 较高的质量要求 **默认NONE**
liveness_control	否	string	活体控制 **NONE**: 不进行控制 **LOW**:较低的活体要求(高通过率 低攻击拒绝率) **NORMAL**: 一般的活体要求(平衡的攻击拒绝率, 通过率) **HIGH**: 较高的活体要求(高攻击拒绝率 低通过率) **默认NONE**
registerimage	是	string	图片信息(总数据大小应小于10M)，图片上传方式根据imagetype来判断。本图片特指客户服务器上传图片，非加密图片Base64值
registerimagetype	是	string	图片类型 **BASE64**：图片的base64值 **URL**：图片的 URL **FACE_TOKEN**：人脸标识 默认 BASE64
registerfacetype	否	string	人脸的类型
registerqualitycontrol	否	string	图片质量控制 **NONE**: 不进行控制 **LOW**:较低的质量要求 **NORMAL**: 一般的质量要求 **HIGH**: 较高的质量要求 **默认 NONE** 若图片质量不满足要求，则返回结果中会提示质量检测失败
registerlivenesscontrol	否	string	活体检测控制 **NONE**: 不进行控制 **LOW**:较低的活体要求(高通过率 低攻击拒绝率) **NORMAL**: 一般的活体要求(平衡的攻击拒绝率, 通过率) **HIGH**: 较高的活体要求(高攻击拒绝率 低通过率) **默认 NONE** 若活体检测结果不满足要求，则返回结果中会提示活体检测失败
facesorttype	否	int	人脸检测排序类型 0:代表检测出的人脸按照人脸面积从大到小排列 1:代表检测出的人脸按照距离图片中心从近到远排列 默认为0

返回参数

参数名	类型	说明
log_id	number	调用的日志id
result	jsonObject	认证返回的结果
+ score	number	人脸相似度得分，推荐阈值80分
+ face_list	jsonArray	人脸信息列表
++ face_token	string	人脸标志
dec_image	string	APP场景传入加密图片时，该项返回解密后的图片
risk_level	string	风控返回参数，只有在 risk_identify 为 true 时才返回，判断设备是否发生过风险行为来判断风险级别，取值（数值有高到低）： 1 – 高危 2 – 嫌疑 3 – 普通 4 – 正常
risk_tag	jsonArray	风控返回参数，只有在 risk_identify 为 true 时才返回，风险标签，若判断为有风险，则会有风险标签 json 数组告知风险类型
risk_warn_code	number	风控返回参数，只有在 risk_identify 为 true 时才返回，只有在风控服务异常的时候才返回这个字段

返回示例


```
{
  "log_id": 1370585066551377920,
  "result": {
    "score": 99.06919861,
    "face_list": [
      {
        "face_token": "549f9f1d1c7ec8c86931540b1939e8ed"
      },
      {
        "face_token": "1a319460ef89e8d27fb59062a28dbad7"
      }
    ]
  },
  "dec_image": "/9j/4AAQSkZJRgABAQAAAQABAAD",
  "risk_level": "3",
  "risk_tag": [
    "空"
  ]
}
```


在线图片活体V4

能力介绍

1. 接口能力

- **人脸基础信息**：包括人脸框位置，人脸空间旋转角度，人脸置信度等信息。
- **人脸质量检测**：判断人脸的遮挡、光照、模糊度、完整度等质量信息。可用于判断上传的人脸是否符合标准。
- **基于图片的活体检测**：基于单张图片，判断图片中的人脸是否为二次翻拍（举例：如用户A用手机拍摄了一张包含人脸的图片一，用户B翻拍了图片一得到了图片二，并用图片二伪造成用户A去进行识别操作，这种情况普遍发生在金融开户、实名

认证等环节）以及是否为合成图攻击。此能力可用于H5场景下的一些人脸采集场景中，增加人脸注册的安全性和真实性。

- **图片加密及风控**：配合采集SDK5.0版本使用，对采集SDK输出的加密图片进行解密（加密传输可以有效避免第三方非法黑产绕过APP模拟请求攻击云端接口的行为，Eg：脚本攻击等）；以及结合百度安全实验室大数据风控能力，对采集SDK的发起端设备进行风控识别，辨别是否为风险设备，Eg：ROM注入、视频劫持等；

```
 public void Demo() {

 // 必填参数
var sdkVersion = "";

 // 选填参数
var options = new Dictionary<string, object>{
    {"xxx", "xxx"},
};

// 调用接口
result = client.onlinePictureLiveV4(sdkVersion, options);
Console.WriteLine(result);
}
```

请求参数

参数	是否必选	类型	说明
sdk_version	是	string	1 ：非加密图片，适用于4.1/4.1.5版本SDK、H5场景或纯服务端场景 4：适用于5.2版本SDK
face_field	否	string	包括**age,expression,face_shape,gender,glasses,landmark,quality,face_type,spoofing**信息，逗号分隔，默认只返回face_token、活体数、人脸框、概率和旋转角度
option	否	string	场景信息，程序会视不同的场景选用相对应的模型。当前支持的场景有COMMON(通用场景)，GATE(闸机场景)，FINANCE(金融场景)，LOGISTICS(物流场景)，INTERNET(泛互联网场景), **默认使用COMMON** 。注意：如果请求参数中存在多个option值时，则取第一个option的值，同时除通用场景外的其他场景需要联系工作人员对您所使用的appid进行配置
app	否	string	端类型 ios / android **5.2版本SDK必传该项**
s_key	否	string	端上提供的用于解密图片的skey **5.2版本SDK必传该项**
device_id	否	string	端上提供的用于解密图片的deviceId **5.2版本SDK必传该项**
data	否	string	端上提供的加密后的图片数组 **5.2版本SDK必传该项**
image_list	否	array	图片BASE64数组 4.1/4.1.5版本SDK、H5场景或纯服务端场景必填

返回参数

参数	是否必须	类型	说明
log_id	是	string	日志id
error_code	是	int	错误码状态，若为0则表示认证成功
error_msg	是	string	错误码说明，若为 success 则表示认证成功
result	是	object	活体结果

+face_liveness	是	float	活体分数值
+thresholds	是	array	由服务端返回最新的阈值数据（随着模型的优化，阈值可能会变化），将此参数与返回的face_liveness进行比较，可以作为活体判断的依据。 frr_1e-4：万分之一误识率的阈值； frr_1e-3：千分之一误识率的阈值； frr_1e-2：百分之一误识率的阈值。 误识率越低，准确率越高，相应的拒绝率也越高
+face_list	是	array	每张图片的详细信息描述，如果只上传一张图片，则只返回一个结果。
++face_token	是	string	人脸图片的唯一标识
++location	是	array	人脸在图片中的位置
+++left	是	double	人脸区域离左边界的距离
+++top	是	double	人脸区域离上边界的距离
+++width	是	double	人脸区域的宽度
+++height	是	double	人脸区域的高度
+++rotation	是	int64	人脸框相对于竖直方向的顺时针旋转角，[-180,180]
++face_probability	是	double	人脸置信度，范围【0~1】，代表这是一张人脸的概率，0最小、1最大。
++angel	是	array	人脸旋转角度参数
+++yaw	是	double	三维旋转之左右旋转角[-90(左), 90(右)]
+++pitch	是	double	三维旋转之俯仰角度[-90(上), 90(下)]
+++roll	是	double	平面内旋转角[-180(逆时针), 180(顺时针)]
++age	否	double	年龄 ，当**face_field包含age时返回**
++expression	否	array	表情，当 **face_field包含expression时返回**
+++type	否	string	**none**:不笑；**smile**:微笑；**laugh**:大笑
+++probability	否	double	表情置信度，范围【0~1】，0最小、1最大。
++face_shape	否	array	脸型，当**face_field包含face_shape时返回**
+++type	否	double	**square**: 正方形 **triangle**:三角形 **oval**: 椭圆 **heart**: 心形 **round**: 圆形
+++probability	否	double	置信度，范围【0~1】，代表这是人脸形状判断正确的概率，0最小、1最大。
++gender	否	array	性别，**face_field包含gender时返回**

+++type	否	string	male:**男性** female:**女性**
+++probability	否	double	性别置信度，范围【0~1】，0代表概率最小、1代表最大。
++glasses	否	array	是否带眼镜，**face_field包含glasses时返回**
++type	否	string	**none**:无眼镜，**common**:普通眼镜，**sun**:墨镜
+++probability	否	double	眼镜置信度，范围【0~1】，0代表概率最小、1代表最大。
++face_type	否	array	真实人脸/卡通人脸 **face_field包含face_type时返回**
+++type	否	string	**human**: 真实人脸 **cartoon**: 卡通人脸
+++probability	否	double	人脸类型判断正确的置信度，范围【0~1】，0代表概率最小、1代表最大。
++landmark	否	array	4个关键点位置，左眼中心、右眼中心、鼻尖、嘴中心。**face_field包含landmark时返回**
++landmark72	否	array	72个特征点位置 **face_field包含landmark时返回**
++quality	否	array	人脸质量信息。**face_field包含quality时返回**
+++occlusion	否	array	人脸各部分遮挡的概率，范围[0~1]，0表示完整，1表示不完整
++++left_eye	否	double	左眼遮挡比例，[0-1] ， 1表示完全遮挡
++++right_eye	否	double	右眼遮挡比例，[0-1] ， 1表示完全遮挡
++++nose	否	double	鼻子遮挡比例，[0-1] ， 1表示完全遮挡
++++mouth	否	double	嘴巴遮挡比例，[0-1] ， 1表示完全遮挡
++++left_cheek	否	double	左脸颊遮挡比例，[0-1] ， 1表示完全遮挡
++++right_cheek	否	double	右脸颊遮挡比例，[0-1] ， 1表示完全遮挡
++++chin	否	double	下巴遮挡比例，，[0-1] ， 1表示完全遮挡
+++blur	否	double	人脸模糊程度，范围[0~1]，0表示清晰，1表示模糊
+++illumination	否	double	取值范围在[0~255], 表示脸部区域的光照程度 越大表示光照越好
+++completeness	否	int64	人脸完整度，0或1, 0为人脸溢出图像边界，1为人脸都在图像边界内

++spoofing	否	double	合成图打分 判断图片是否为合成图 face_field包含时返回spoofing
risk_level	否	string	判断设备是否发生过风险行为来判断风险级别，取值（数值有高到低）：1 – 高危 2 – 嫌疑 3 – 普通 4 – 正常
risk_tag	否	string	风险标签，若判断为有风险，则会有风险标签 json 数组告知风险类型
dec_image	否	array	非安全加固增强级采集sdk、非安全加固金融级采集sdk、安全加固增强级采集sdk、安全加固金融级采集sdk会返回解密后的原图

返回示例


```
{
  "result": {
    "thresholds": {
      "frr_1e-3": 0.3,
      "frr_1e-2": 0.9,
      "frr_1e-4": 0.05
    },
    "face_liveness": 0.1976952702,
    "face_list": [
      {
        "liveness": {
          "livemapscore": 0.1976952702
        },
        "angle": {
          "roll": 0.31,
          "pitch": -2.07,
          "yaw": 0.85
        },
        "face_token": "abd1b4ce743099336cfed40193ff4944",
        "location": {
          "top": 161.74,
          "left": 79.04,
          "rotation": 1,
          "width": 318,
          "height": 333
        },
        "face_probability": 1
      },
      {
        "liveness": {
          "livemapscore": 0.1976952702
        },
        "angle": {
          "roll": 0.31,
          "pitch": -2.07,
          "yaw": 0.85
        },
        "face_token": "abd1b4ce743099336cfed40193ff4944",
        "location": {
          "top": 161.74,
          "left": 79.04,
          "rotation": 1,
          "width": 318,
          "height": 333
        },
        "face_probability": 1
      },
      {
```

```
            "liveness": {
                "livemapscore": 0.1976952702
            },
            "angle": {
                "roll": 0.31,
                "pitch": -2.07,
                "yaw": 0.85
            },
            "face_token": "abd1b4ce743099336cfed40193ff4944",
            "location": {
                "top": 161.74,
                "left": 79.04,
                "rotation": 1,
                "width": 318,
                "height": 333
            },
            "face_probability": 1
        },
        {
            "liveness": {
                "livemapscore": 0.1976952702
            },
            "angle": {
                "roll": 0.31,
                "pitch": -2.07,
                "yaw": 0.85
            },
            "face_token": "abd1b4ce743099336cfed40193ff4944",
            "location": {
                "top": 161.74,
                "left": 79.04,
                "rotation": 1,
                "width": 318,
                "height": 333
            },
            "face_probability": 1
        },
        {
            "liveness": {
                "livemapscore": 0.1976952702
            },
            "angle": {
                "roll": 0.31,
                "pitch": -2.07,
                "yaw": 0.85
            },
            "face_token": "abd1b4ce743099336cfed40193ff4944",
            "location": {
                "top": 161.74,
                "left": 79.04,
                "rotation": 1,
                "width": 318,
                "height": 333
            },
            "face_probability": 1
        },
        {
            "liveness": {
                "livemapscore": 0.1976952702
            },
            "angle": {
                "roll": 0.31,
                "pitch": -2.07,
                "yaw": 0.85
            },
            "face_token": "abd1b4ce743099336cfed40193ff4944",
            "location": {
                "top": 161.74,
                "left": 79.04,
                "rotation": 1,
                "width": 318,
                "height": 333
            },
            "face_probability": 1
        },
        {
            "liveness": {
                "livemapscore": 0.1976952702
            },
            "angle": {
                "roll": 0.31,
                "pitch": -2.07,
```

```
                "pitch": -2.07,
                "yaw": 0.85
            },
            "face_token": "abd1b4ce743099336cfed40193ff4944",
            "location": {
                "top": 161.74,
                "left": 79.04,
                "rotation": 1,
                "width": 318,
                "height": 333
            },
            "face_probability": 1
        },
        {
            "liveness": {
                "livemapscore": 0.1976952702
            },
            "angle": {
                "roll": 0.31,
                "pitch": -2.07,
                "yaw": 0.85
            },
            "face_token": "abd1b4ce743099336cfed40193ff4944",
            "location": {
                "top": 161.74,
                "left": 79.04,
                "rotation": 1,
                "width": 318,
                "height": 333
            },
            "face_probability": 1
        },
        {
            "liveness": {
                "livemapscore": 0.1976952702
            },
            "angle": {
                "roll": 0.31,
                "pitch": -2.07,
                "yaw": 0.85
            },
            "face_token": "abd1b4ce743099336cfed40193ff4944",
            "location": {
                "top": 161.74,
                "left": 79.04,
                "rotation": 1,
                "width": 318,
                "height": 333
            },
            "face_probability": 1
        }
    ]
},
"risk_level": "3",
"log_id": 1423545654699779516,
"risk_tag": [
    "空"
],
"dec_image": [
        "/9j/4AAQ..."
]
}
```

FAQ

1. 调用接口时抛出异常：基础连接已关闭 此问题一般是网络原因，请检查网络相关问题：

- 检查网络连通性
- 检查网络是否有代理

2. SSL报错 "The authentication or decryption has failed" 可能由于网络代理等原因导致证书不正确，属于常见的https问题，可以参考这个答案。 如果报此错，请开发者务必关注HTTPS安全。

3. 调用接口，等待一段时间后出现超时 可以设置Timeout参数。

4. 在.Net Core下运行报错 'GBK' is not a supported encoding name' 添加System.Text.Encoding.CodePages引用，并注册。`Encoding.RegisterProvider(CodePagesEncodingProvider.Instance)`

错误信息

错误返回格式

若请求错误，服务器将返回的JSON文本包含以下参数：

- **error_code**：错误码。
- **error_msg**：错误描述信息，帮助理解和解决发生的错误。

错误码

错误码	错误信息	描述
1	Unknown error	服务器内部错误，请再次请求， 如果持续出现此类错误，请通过QQ群（860337848）或提交工单联系技术支持团队。
2	Service temporarily unavailable	服务暂不可用，请再次请求， 如果持续出现此类错误，请通过QQ群（860337848）或提交工单联系技术支持团队。
3	Unsupported openapi method	调用的API不存在，请检查请求URL后重新尝试，一般为URL中有非英文字符，如“-”，可手动输入重试
4	Open api request limit reached	集群超限额，请再次请求， 如果持续出现此类错误，请通过QQ群（860337848）或提交工单联系技术支持团队。
6	No permission to access data	无权限访问该用户数据，创建应用时未勾选相关接口
13	Get service token failed	获取token失败
14	IAM Certification failed	IAM 鉴权失败
15	app not exsits or create failed	应用不存在或者创建失败
17	Open api daily request limit reached	每天请求量超限额，可通过QQ群（860337848）联系群管、提交工单提升限额
18	Open api qps request limit reached	QPS超限额，可通过QQ群（860337848）联系群管、提交工单提升限额
19	Open api total request limit reached	请求总量超限额，可通过QQ群（860337848）联系群管、提交工单提升限额
100	Invalid parameter	无效的access_token参数，请检查后重新尝试
110	Access token invalid or no longer valid	access_token无效
111	Access token expired	access token过期

111	Access token expired	access token过期
222001	param[] is null	必要参数未传入
222002	param[start] format error	参数格式错误
222003	param[length] format error	参数格式错误
222004	param[op_app_id_list] format error	参数格式错误
222005	param[group_id_list] format error	参数格式错误
222006	group_id format error	参数格式错误
222007	uid format error	参数格式错误
222008	face_id format error	参数格式错误
222009	quality_conf format error	参数格式错误
222010	user_info format error	参数格式错误
222011	param[uid_list] format error	参数格式错误
222012	param[op_app_id] format error	参数格式错误
222013	param[image] format error	参数格式错误
222014	param[app_id] format error	参数格式错误
222015	param[image_type] format error	参数格式错误
222016	param[max_face_num] format error	参数格式错误
222017	param[face_field] format error	参数格式错误
222018	param[user_id] format error	参数格式错误
222019	param[quality_control] format error	参数格式错误
222020	param[liveness_control] format error	参数格式错误
222021	param[max_user_num] format error	参数格式错误
222022	param[id_card_number] format error	参数格式错误
222023	param[name] format error	参数格式错误
222024	param[face_type] format error	参数格式错误
222025	param[face_token] format error	参数格式错误

222026	param[max_star_num] format error	参数格式错误
222201	network not available	服务端请求失败
222202	pic not has face	图片中没有人脸
222203	image check fail	无法解析人脸
222204	image_url_download_fail	从图片的url下载图片失败
222205	network not availablel	服务端请求失败
222206	rtse service return fail	服务端请求失败
222207	match user is not found	未找到匹配的用户
222208	the number of image is incorrect	图片的数量错误
222209	face token not exist	face token不存在
222300	add face fail	人脸图片添加失败
222301	get face fail	获取人脸图片失败
222302	system error	服务端请求失败
222303	get face fail	获取人脸图片失败
223100	group is not exist	操作的用户组不存在
223101	group is already exist	该用户组已存在
223102	user is already exist	该用户已存在
223103	user is not exist	找不到该用户
223104	group_list is too large	group_list包含组数量过多
223105	face is already exist	该人脸已存在
223106	face is not exist	该人脸不存在
223110	uid_list is too large	uid_list包含数量过多
223111	dst group is not exist	目标用户组不存在
223112	quality_conf format error	quality_conf格式不正确
223113	face is covered	人脸有被遮挡
223114	face is fuzzy	人脸模糊
223115	face light is not good	人脸光照不好
223116	incomplete face	人脸不完整
223117	app_list is too large	app_list包含app数量过多
223118	quality control error	质量控制项错误
223119	liveness control item error	活体控制项错误
223120	liveness check fail	活体检测未通过
223121	left eye is occlusion	质量检测未通过 左眼遮挡程度过高

		遮挡程度过高
223122	right eye is occlusion	质量检测未通过 右眼 遮挡程度过高
223123	left cheek is occlusion	质量检测未通过 左脸 遮挡程度过高
223124	right cheek is occlusion	质量检测未通过 右脸 遮挡程度过高
223125	chin contour is occlusion	质量检测未通过 下巴遮挡程度过高
223126	nose is occlusion	质量检测未通过 鼻子遮挡程度过高
223127	mouth is occlusion	质量检测未通过 嘴巴 遮挡程度过高
222350	police picture is none or low quality	公安网图片不存在或 质量过低
222351	id number and name not match or id number not exist	身份证号与姓名不匹配或该 身份证号不存在
222352	name format error	身份证名字格式错误
222353	id number format error	身份证号码格式错误
222354	id number not exist	公安库里不存在此身份证号
222355	police picture not exist	身份证号码正确，公安库里没有 对应的照片
222360	invalid name or id number	身份证号码或名字非法（公安网校 验不通过）
222901	system busy	系统繁忙
222902	system busy	系统繁忙
222903	system busy	系统繁忙
222904	system busy	系统繁忙
222905	system busy	系统繁忙
222906	system busy	系统繁忙
222907	system busy	系统繁忙
222908	system busy	系统繁忙
222909	system busy	系统繁忙
222910	system busy	系统繁忙
222911	system busy	系统繁忙
222912	system busy	系统繁忙
222913	system busy	系统繁忙
222914	system busy	系统繁忙
222915	system busy	系统繁忙
222916	system busy	系统繁忙
222361	system busy	系统繁忙

简介

Hi，您好，欢迎使用百度人脸识别服务。

本文档主要针对Nodejs开发者，描述百度人脸识别接口服务的相关技术内容。如果您对文档内容有任何疑问，可以通过以下几种方式联系我们：

- 在百度云控制台内提交工单，咨询问题类型请选择**人工智能服务**；
- 如有疑问，进入AI社区交流：http://ai.baidu.com/forum/topic/list/165

接口能力

接口名称	接口能力简要描述
人脸检测	检测人脸并定位，返回五官关键点，及人脸各属性值
人脸比对	返回两两比对的人脸相似值
人脸查找	在一个人脸集合中找到找到相似的人脸，由一系列接口组成，包括人脸识别、人脸认证、人脸库管理相关接口（人脸注册、人脸更新、人脸删除、用户信息查询、组列表查询、组内用户列表查询、组间复制用户、组内删除用户）

版本更新记录

上线日期	版本号	更新内容
2019.4.17	2.4.1	人脸v3文档更新，新增N:M接口
2018.5.10	2.3.1	修复人脸V3接口问题
2018.4.28	2.3.0	全面切换为人脸V3接口
2018.4.9	2.2.0	新增身份验证，在线活体检测接口
2018.1.12	2.1.0	新增M：N多人脸识别接口
2017.12.21	2.0.0	实现代码重构，接口返回标准promise对象
2017.11.14	1.5.1	人脸检测接口升级到v2
2017.5.11	1.0.0	初版

快速入门

安装人脸识别 Node SDK

人脸识别 Node SDK目录结构

```
├─── src
|   ├─── auth                    //授权相关类
|   ├─── http                    //Http通信相关类
|   ├─── client                  //公用类
|   ├─── util                    //工具类
|   └─── const                   //常量类
├─── AipFace.js                 //人脸识别交互类
├─── index.js                   //入口文件
└─── package.json                 //npm包描述文件
```

支持 node 版本 4.0+

直接使用node开发包步骤如下：

1.在官方网站下载node SDK压缩包。

2.将下载的aip-node-sdk-version.zip解压后，复制到工程文件夹中。

3.进入目录，运行npm install安装sdk依赖库

4.把目录当做模块依赖

其中，version为版本号，添加完成后，用户就可以在工程中使用人脸识别 Node SDK。

直接使用npm安装依赖：

```
npm install baidu-aip-sdk
```

新建AipFaceClient

AipFaceClient是人脸识别的node客户端，为使用人脸识别的开发人员提供了一系列的交互方法。

用户可以参考如下代码新建一个AipFaceClient：

```
 var AipFaceClient = require("baidu-aip-sdk").face;

// 设置APPID/AK/SK
var APP_ID = "你的 App ID";
var API_KEY = "你的 Api Key";
var SECRET_KEY = "你的 Secret Key";

// 新建一个对象，建议只保存一个对象调用服务接口
var client = new AipFaceClient(APP_ID, API_KEY, SECRET_KEY);
```

为了使开发者更灵活的控制请求，模块提供了设置全局参数和全局请求拦截器的方法；本库发送网络请求依赖的是request模块，因此参数格式与request模块的参数相同

更多参数细节您可以参考request官方参数文档。

```
 var HttpClient = require("baidu-aip-sdk").HttpClient;

// 设置request库的一些参数，例如代理服务地址，超时时间等
// request参数请参考 https://github.com/request/request#requestoptions-callback
HttpClient.setRequestOptions({timeout: 5000});

// 也可以设置拦截每次请求（设置拦截后，调用的setRequestOptions设置的参数将不生效），
// 可以按需修改request参数（无论是否修改，必须返回函数调用参数）
// request参数请参考 https://github.com/request/request#requestoptions-callback
HttpClient.setRequestInterceptor(function(requestOptions) {
    // 查看参数
    console.log(requestOptions)
    // 修改参数
    requestOptions.timeout = 5000;
    // 返回参数
    return requestOptions;
});
```

在上面代码中，常量APP_ID在百度云控制台中创建，常量API_KEY与SECRET_KEY是在创建完毕应用后，系统分配给用户的，均为字符串，用于标识用户，为访问做签名验证，可在AI服务控制台中的**应用列表**中查看。

注意：如您以前是百度云的老用户，其中API_KEY对应百度云的“Access Key ID”，SECRET_KEY对应百度云的“Access Key Secret”。

接口说明

人脸检测

人脸检测：检测图片中的人脸并标记出位置信息;

```
 var image = "取决于image_type参数，传入BASE64字符串或URL字符串或FACE_TOKEN字符串";

var imageType = "BASE64";

// 调用人脸检测
client.detect(image, imageType).then(function(result) {
    console.log(JSON.stringify(result));
}).catch(function(err) {
    // 如果发生网络错误
    console.log(err);
});

// 如果有可选参数
var options = {};
options["face_field"] = "age";
options["max_face_num"] = "2";
options["face_type"] = "LIVE";
options["liveness_control"] = "LOW";

// 带参数调用人脸检测
client.detect(image, imageType, options).then(function(result) {
    console.log(JSON.stringify(result));
}).catch(function(err) {
    // 如果发生网络错误
    console.log(err);
});;
```

人脸检测 请求参数详情

参数名称	是否必选	类型	默认值	说明
image	是	string		图片信息(**总数据大小应小于10M**)，图片上传方式根据image_type来判断
image_type	是	string		图片类型 **BASE64**:图片的base64值，base64编码后的图片数据，编码后的图片大小不超过2M；**URL**:图片的 URL地址(可能由于网络等原因导致下载图片时间过长)；**FACE_TOKEN**: 人脸图片的唯一标识，调用人脸检测接口时，会为每个人脸图片赋予一个唯一的FACE_TOKEN，同一张图片多次检测得到的FACE_TOKEN是同一个
face_field	否	string		包括**age,expression,face_shape,gender,glasses,landmark,landmark150,quality,eye_status,emotion,face_type信息** 逗号分隔. 默认只返回face_token、人脸框、概率和旋转角度
max_face_num	否	string	1	最多处理人脸的数目，默认值为1，仅检测图片中面积最大的那个人脸；**最大值10**，检测图片中面积最大的几张人脸。
face_type	否	string		人脸的类型 **LIVE**表示生活照：通常为手机、相机拍摄的人像图片、或从网络获取的人像图片等**IDCARD**表示身份证芯片照：二代身份证内置芯片中的人像照片 **WATERMARK**表示带水印证件照：一般为带水印的小图，如公安网小图 **CERT**表示证件照片：如拍摄的身份证、工卡、护照、学生证等证件图片 默认**LIVE**
liveness_control	否	string	NONE	活体检测控制 **NONE**: 不进行控制 **LOW**:较低的活体要求(高通过率 低攻击拒绝率) **NORMAL**: 一般的活体要求(平衡的攻击拒绝率, 通过率) **HIGH**: 较高的活体要求(高攻击拒绝率 低通过率) **默认NONE**

人脸检测 返回数据参数详情

字段	必选	类型	说明
face_num	是	int	检测到的图片中的人脸数量
face_list	是	array	人脸信息列表，具体包含的参数参考下面的列表。
+face_token	是	string	人脸图片的唯一标识
+location	是	array	人脸在图片中的位置
++left	是	double	人脸区域离左边界的距离
++top	是	double	人脸区域离上边界的距离
++width	是	double	人脸区域的宽度
++height	是	double	人脸区域的高度
++rotation	是	int64	人脸框相对于竖直方向的顺时针旋转角，[-180,180]
+face_probability	是	double	人脸置信度，范围【0~1】，代表这是一张人脸的概率，0最小、1最大。
+angel	是	array	人脸旋转角度参数
++yaw	是	double	三维旋转之左右旋转角[-90(左), 90(右)]
++pitch	是	double	三维旋转之俯仰角度[-90(上), 90(下)]
++roll	是	double	平面内旋转角[-180(逆时针), 180(顺时针)]
+age	否	double	年龄 ，当**face_field包含age时返回**
+expression	否	array	表情，当 **face_field包含expression时返回**
++type	否	string	**none**:不笑；**smile**:微笑；**laugh**:大笑
++probability	否	double	表情置信度，范围【0~1】，0最小、1最大。
+face_shape	否	array	脸型，当**face_field包含face_shape时返回**
++type	否	double	**square**: 正方形 **triangle**:三角形 **oval**: 椭圆 **heart**: 心形 **round**: 圆形
++probability	否	double	置信度，范围【0~1】，代表这是人脸形状判断正确的概率，0最小、1最大。
+gender	否	array	性别，**face_field包含gender时返回**
++type	否	string	male:**男性** female:**女性**
++probability	否	double	性别置信度，范围【0~1】，0代表概率最小、1代表最大。
+glasses	否	array	是否带眼镜，**face_field包含glasses时返回**
++type	否	string	**none**:无眼镜，**common**:普通眼镜，**sun**:墨镜
++probability	否	double	眼镜置信度，范围【0~1】，0代表概率最小、1代表最大。
+eye_status	否	array	双眼状态（睁开/闭合） **face_field包含eye_status时返回**
++left_eye	否	double	左眼状态 [0,1]取值，越接近0闭合的可能性越大
++right_eye	否	double	右眼状态 [0,1]取值，越接近0闭合的可能性越大
+emotion	否	array	情绪 **face_field包含emotion时返回**
++type	否	string	**angry**:愤怒 **disgust**:厌恶 **fear**:恐惧 **happy**:高兴 **sad**:伤心 **surprise**:惊讶 **neutral**:无情绪
++probability	否	double	情绪置信度，范围0~1
++probability	否	double	人种置信度，范围【0~1】，0代表概率最小、1代表最大。
+face_type	否	array	真实人脸/卡通人脸 **face_field包含face_type时返回**
++type	否	string	**human**: 真实人脸 **cartoon**: 卡通人脸
++probability	否	double	人脸类型判断正确的置信度，范围【0~1】，0代表概率最小、1代表最大。

+landmark	否	array	4个关键点位置，左眼中心、右眼中心、鼻尖、嘴中心。**face_field包含landmark时返回**
+landmark72	否	array	72个特征点位置 **face_field包含landmark时返回**
+landmark150	否	array	150个特征点位置 **face_field包含landmark150时返回**
+quality	否	array	人脸质量信息。**face_field包含quality时返回**
++occlusion	否	array	人脸各部分遮挡的概率，范围[0~1]，0表示完整，1表示不完整
+++left_eye	否	double	左眼遮挡比例，[0-1] ，1表示完全遮挡
+++right_eye	否	double	右眼遮挡比例，[0-1] ， 1表示完全遮挡
+++nose	否	double	鼻子遮挡比例，[0-1] ， 1表示完全遮挡
+++mouth	否	double	嘴巴遮挡比例，[0-1] ， 1表示完全遮挡
+++left_cheek	否	double	左脸颊遮挡比例，[0-1] ， 1表示完全遮挡
+++right_cheek	否	double	右脸颊遮挡比例，[0-1] ， 1表示完全遮挡
+++chin_contour	否	double	下巴遮挡比例，，[0-1] ， 1表示完全遮挡
++blur	否	double	人脸模糊程度，范围[0~1]，0表示清晰，1表示模糊
++illumination	否	double	取值范围在[0~255], 表示脸部区域的光照程度 越大表示光照越好
++completeness	否	int64	人脸完整度，0或1, 0为人脸溢出图像边界，1为人脸都在图像边界内

人脸检测 返回示例


```
{
 "face_num": 1,
 "face_list": [
     {
        "face_token": "35235asfas21421fakghktyfdgh68bio",
        "location": {
           "left": 117,
           "top": 131,
           "width": 172,
           "height": 170,
           "rotation": 4
        },
        "face_probability": 1,
        "angle" :{
            "yaw" : -0.34859421849251
            "pitch" 1.9135693311691
            "roll" :2.3033397197723
        }
        "landmark": [
           {
              "x": 161.74819946289,
              "y": 163.30244445801
           },
           ...
        ],
        "landmark72": [
           {
              "x": 115.86531066895,
              "y": 170.0546875
           } ,
           ...
        ],
        "age": 29.298097610474,
        "expression": {
           "type": "smile",
```

```
            "probability" : 0.5543018579483
        },
        "gender": {
            "type": "male",
            "probability": 0.99979132413864
        },
        "glasses": {
       "type": "sun",
            "probability": 0.99999964237213
        },

        "face_shape": {
            "type": "triangle",
            "probability": 0.5543018579483
        }
        "quality": {
            "occlusion": {
                "left_eye": 0,
                "right_eye": 0,
                "nose": 0,
                "mouth": 0,
                "left_cheek": 0.0064102564938366,
                "right_cheek": 0.0057411273010075,
                "chin": 0
            },
            "blur": 1.1886881756684e-10,
            "illumination": 141,
            "completeness": 1
        }
    }
  ]
}
```


**72个关键点分布图（对应landmark72个点的顺序，序号从0-71）：https://ai.bdstatic.com/file/52BC00FFD4754A6298D977EDAD033DA0

人脸搜索

- **1：N人脸搜索**：也称为1：N识别，在指定人脸集合中，找到最相似的人脸；
- **1：N人脸认证**：基于uid维度的1：N识别，由于uid已经锁定固定数量的人脸，所以检索范围更聚焦；

> **1：N人脸识别**与**1：N人脸认证**的差别在于：人脸搜索是在指定人脸集合中进行直接地人脸检索操作，而人脸认证是基于uid，先调取这个uid对应的人脸，再在这个uid对应的人脸集合中进行检索（因为每个uid通常对应的只有一张人脸，所以通常也就变为了1：1对比）；实际应用中，人脸认证需要用户或系统先输入id，这增加了验证安全度，但也增加了复杂度，具体使用哪个接口需要视您的业务场景判断。

```
 var image = "取决于image_type参数，传入BASE64字符串或URL字符串或FACE_TOKEN字符串";

var imageType = "BASE64";

var groupIdList = "3,2";

// 调用人脸搜索
client.search(image, imageType, groupIdList).then(function(result) {
    console.log(JSON.stringify(result));
}).catch(function(err) {
    // 如果发生网络错误
    console.log(err);
});

// 如果有可选参数
var options = {};
options["match_threshold"] = "70";
options["quality_control"] = "NORMAL";
options["liveness_control"] = "LOW";
options["user_id"] = "233451";
options["max_user_num"] = "3";

// 带参数调用人脸搜索
client.search(image, imageType, groupIdList, options).then(function(result) {
    console.log(JSON.stringify(result));
}).catch(function(err) {
    // 如果发生网络错误
    console.log(err);
});;
```

人脸搜索 请求参数详情

参数名称	是否必选	类型	默认值	说明
image	是	string		图片信息(**总数据大小应小于10M**)，图片上传方式根据image_type来判断
image_type	是	string		图片类型 **BASE64**:图片的base64值，base64编码后的图片数据，编码后的图片大小不超过2M；**URL**:图片的 URL地址(可能由于网络等原因导致下载图片时间过长)； **FACE_TOKEN**: 人脸图片的唯一标识，调用人脸检测接口时，会为每个人脸图片赋予一个唯一的FACE_TOKEN，同一张图片多次检测得到的FACE_TOKEN是同一个
group_id_list	是	string		从指定的group中进行查找 用逗号分隔，**上限10个**
match_threshold	否	string		匹配阈值（设置阈值后，score低于此阈值的用户信息将不会返回） 最大100 最小0 默认80 **此阈值设置得越高，检索速度将会越快，推荐使用默认阈值80**
quality_control	否	string	NONE	图片质量控制 **NONE**: 不进行控制 **LOW**:较低的质量要求 **NORMAL**: 一般的质量要求 **HIGH**: 较高的质量要求 **默认 NONE**
liveness_control	否	string	NONE	活体检测控制 **NONE**: 不进行控制 **LOW**:较低的活体要求(高通过率 低攻击拒绝率) **NORMAL**: 一般的活体要求(平衡的攻击拒绝率, 通过率) **HIGH**: 较高的活体要求(高攻击拒绝率 低通过率) **默认NONE**
user_id	否	string		当需要对特定用户进行比对时，指定user_id进行比对。即人脸认证功能。
max_user_num	否	string		查找后返回的用户数量。返回相似度最高的几个用户，默认为1，最多返回50个。

人脸搜索 返回数据参数详情

字段	必选	类型	说明
face_token	是	string	人脸标志
user_list	是	array	匹配的用户信息列表
+group_id	是	string	用户所属的group_id
+user_id	是	string	用户的user_id
+user_info	是	string	注册用户时携带的user_info
+score	是	float	用户的匹配得分

人脸搜索 返回示例

```
{
 "face_token": "fid",
 "user_list": [
   {
      "group_id" : "test1",
      "user_id": "u333333",
      "user_info": "Test User",
      "score": 99.3
   }
 ]
}
```

人脸搜索 M:N 识别

待识别的图片中，存在多张人脸的情况下，支持在一个人脸库中，一次请求，同时返回图片中所有人脸的识别结果。

```
 var image = "取决于image_type参数，传入BASE64字符串或URL字符串或FACE_TOKEN字符串";

var imageType = "BASE64";

var groupIdList = "3,2";

// 调用人脸搜索 M:N 识别
client.faceMultiSearchV3(image, imageType, groupIdList).then(function(result) {
    console.log(JSON.stringify(result));
}).catch(function(err) {
    // 如果发生网络错误
    console.log(err);
});

// 如果有可选参数
var options = {};
options["max_face_num"] = "3";
options["match_threshold"] = "70";
options["quality_control"] = "NORMAL";
options["liveness_control"] = "LOW";
options["max_user_num"] = "3";

// 带参数调用人脸搜索 M:N 识别
client.faceMultiSearchV3(image, imageType, groupIdList, options).then(function(result) {
    console.log(JSON.stringify(result));
}).catch(function(err) {
    // 如果发生网络错误
    console.log(err);
});;
```

人脸搜索 M:N 识别 请求参数详情

参数名称	是否必选	类型	默认值	说明
image	是	string		图片信息(**总数据大小应小于10M**)，图片上传方式根据image_type来判断
image_type	是	string		图片类型 **BASE64**:图片的base64值，base64编码后的图片数据，编码后的图片大小不超过2M；**URL**:图片的 URL地址(可能由于网络等原因导致下载图片时间过长)； **FACE_TOKEN**: 人脸图片的唯一标识，调用人脸检测接口时，会为每个人脸图片赋予一个唯一的FACE_TOKEN，同一张图片多次检测得到的FACE_TOKEN是同一个
group_id_list	是	string		从指定的group中进行查找 用逗号分隔，**上限10个**
max_face_num	否	string		最多处理人脸的数目 **默认值为1(仅检测图片中面积最大的那个人脸) 最大值10**
match_threshold	否	string		匹配阈值（设置阈值后，score低于此阈值的用户信息将不会返回） 最大100 最小0 默认80 **此阈值设置得越高，检索速度将会越快，推荐使用默认阈值80**
quality_control	否	string	NONE	图片质量控制 **NONE**: 不进行控制 **LOW**:较低的质量要求 **NORMAL**: 一般的质量要求 **HIGH**: 较高的质量要求 **默认 NONE**
liveness_control	否	string	NONE	活体检测控制 **NONE**: 不进行控制 **LOW**:较低的活体要求(高通过率 低攻击拒绝率) **NORMAL**: 一般的活体要求(平衡的攻击拒绝率, 通过率) **HIGH**: 较高的活体要求(高攻击拒绝率 低通过率) **默认NONE**
max_user_num	否	string		查找后返回的用户数量。返回相似度最高的几个用户，默认为1，最多返回20个。

人脸搜索 M:N 识别 返回数据参数详情

字段	必选	类型	说明
face_num	是	int	图片中的人脸数量
face_list	是	array	人脸信息列表
+face_token	是	string	人脸标志
+location	是	array	人脸在图片中的位置
++left	是	double	人脸区域离左边界的距离
++top	是	double	人脸区域离上边界的距离
++width	是	double	人脸区域的宽度
++height	是	double	人脸区域的高度
++rotation	是	int64	人脸框相对于竖直方向的顺时针旋转角，[-180,180]
+user_list	是	array	匹配的用户信息列表
++group_id	是	string	用户所属的group_id
++user_id	是	string	用户的user_id
++user_info	是	string	注册用户时携带的user_info
++score	是	float	用户的匹配得分 **80分以上可以判断为同一人，此分值对应万分之一误识率**

人脸搜索 M:N 识别 返回示例

```
{
  "error_code": 0,
  "error_msg": "SUCCESS",
  "log_id": 240483475,
  "timestamp": 1535533440,
  "cached": 0,
  "result": {
    "face_num": 2,
    "face_list": [
      {
        "face_token": "6fe19a6ee0c4233db9b5bba4dc2b9233",
        "location": {
          "left": 31.95568085,
          "top": 120.3764267,
          "width": 87,
          "height": 85,
          "rotation": -5
        },
        "user_list": [
          {
            "group_id": "group1",
            "user_id": "5abd24fd062e49bfa906b257ec40d284",
            "user_info": "userinfo1",
            "score": 69.85684967041
          },
          {
            "group_id": "group1",
            "user_id": "2abf89cffb31473a9948268fde9e1c3f",
            "user_info": "userinfo2",
            "score": 66.586112976074
          }
        ]
      },
      {
        "face_token": "fde61e9c074f48cf2bbb319e42634f41",
        "location": {
          "left": 219.4467773,
          "top": 104.7486954,
          "width": 81,
          "height": 77,
          "rotation": 3
        },
        "user_list": [
          {
            "group_id": "group1",
            "user_id": "088717532b094c3990755e91250adf7d",
            "user_info": "userinfo",
            "score": 65.154159545898
          }
        ]
      }
    ]
  }
}
```


人脸注册

用于从人脸库中新增用户，可以设定多个用户所在组，及组内用户的人脸图片，

典型应用场景：构建您的人脸库，如**会员人脸注册**，**已有用户补全人脸信息**等。

人脸库、用户组、用户、用户下的人脸**层级关系**如下所示：

```
|- 人脸库
 |- 用户组一
   |- 用户01
     |- 人脸
   |- 用户02
     |- 人脸
     |- 人脸
     ....
    ....
 |- 用户组二
 |- 用户组三
 |- 用户组四
 ....
```

关于人脸库的设置限制

- **每个appid对应一个人脸库，且不同appid之间，人脸库互不相通**；
- 每个人脸库下，可以创建多个用户组，用户组（group）数量**没有限制**；
- 每个用户组（group）下，可添加最多**无限**张人脸，**无限**个uid；
- 每个用户（uid）所能注册的最大人脸数量**20个**；

为了保证识别效果，请控制注册人脸的质量（通过`/detect`人脸检测接口判断），具体参数可详见下表所示：

质量判断

可通过人脸检测接口，基于以下字段和对应阈值，进行质量检测的判断，以保证人脸质量符合后续业务操作要求。

指标	字段与解释	推荐数值界限
遮挡范围	**occlusion**（0~1），0为无遮挡，1是完全遮挡 含有多个具体子字段，表示脸部多个部位 通常用作判断头发、墨镜、口罩等遮挡	left_eye：0.6, #左眼被遮挡的阈值 right_eye：0.6, #右眼被遮挡的阈值 nose：0.7, #鼻子被遮挡的阈值 mouth：0.7, #嘴巴被遮挡的阈值 left_check：0.8, #左脸颊被遮挡的阈值 right_check：0.8, #右脸颊被遮挡的阈值 chin_contour：0.6, #下巴被遮挡阈值
模糊度范围	**Blur**（0~1），0是最清晰，1是最模糊	小于0.7
光照范围	**illumination**（0~255） 脸部光照的灰度值，0表示光照不好 以及对应客户端SDK中，YUV的Y分量	大于40
姿态角度	**Pitch**：三维旋转之俯仰角度[-90(上), 90(下)] **Roll**：平面内旋转角[-180(逆时针), 180(顺时针)] **Yaw**：三维旋转之左右旋转角[-90(左), 90(右)]	分别小于20度
人脸完整度	**completeness**（0或1），0为人脸溢出图像边界，1为人脸都在图像边界内	视业务逻辑判断
人脸大小	人脸部分的大小 建议长宽像素值范围：80*80~200*200	人脸部分不小于**100*100**像素

```
 var image = "取决于image_type参数，传入BASE64字符串或URL字符串或FACE_TOKEN字符串";

var imageType = "BASE64";

var groupId = "group1";

var userId = "user1";

// 调用人脸注册
client.addUser(image, imageType, groupId, userId).then(function(result) {
    console.log(JSON.stringify(result));
}).catch(function(err) {
    // 如果发生网络错误
    console.log(err);
});

// 如果有可选参数
var options = {};
options["user_info"] = "user's info";
options["quality_control"] = "NORMAL";
options["liveness_control"] = "LOW";
options["action_type"] = "REPLACE";

// 带参数调用人脸注册
client.addUser(image, imageType, groupId, userId, options).then(function(result) {
    console.log(JSON.stringify(result));
}).catch(function(err) {
    // 如果发生网络错误
    console.log(err);
});;
```

人脸注册 请求参数详情

参数名称	是否必选	类型	默认值	说明
image	是	string		图片信息(总数据大小应小于10M)，图片上传方式根据image_type来判断。注：组内每个uid下的人脸图片数目上限为20张
image_type	是	string		图片类型 **BASE64**:图片的base64值，base64编码后的图片数据，编码后的图片大小不超过2M；**URL**:图片的 URL地址(可能由于网络等原因导致下载图片时间过长)；**FACE_TOKEN**: 人脸图片的唯一标识，调用人脸检测接口时，会为每个人脸图片赋予一个唯一的FACE_TOKEN，同一张图片多次检测得到的FACE_TOKEN是同一个
group_id	是	string		用户组id（由数字、字母、下划线组成），长度限制128B
user_id	是	string		用户id（由数字、字母、下划线组成），长度限制128B
user_info	否	string		用户资料，长度限制256B
quality_control	否	string	NONE	图片质量控制 **NONE**: 不进行控制 **LOW**:较低的质量要求 **NORMAL**: 一般的质量要求 **HIGH**: 较高的质量要求 **默认 NONE**
liveness_control	否	string	NONE	活体检测控制 **NONE**: 不进行控制 **LOW**:较低的活体要求(高通过率 低攻击拒绝率) **NORMAL**: 一般的活体要求(平衡的攻击拒绝率, 通过率) **HIGH**: 较高的活体要求(高攻击拒绝率 低通过率) **默认NONE**
action_type	否	string	APPEND	操作方式 APPEND: 当user_id在库中已经存在时，对此user_id重复注册时，新注册的图片默认会追加到该user_id下,REPLACE：当对此user_id重复注册时,则会用新图替换库中该user_id下所有图片,默认使用APPEND

人脸注册 返回数据参数详情

字段	必选	类型	说明
log_id	是	uint64	请求标识码，随机数，唯一
face_token	是	string	人脸图片的唯一标识
location	是	array	人脸在图片中的位置
+left	是	double	人脸区域离左边界的距离
+top	是	double	人脸区域离上边界的距离
+width	是	double	人脸区域的宽度
+height	是	double	人脸区域的高度
+rotation	是	int64	人脸框相对于竖直方向的顺时针旋转角，[-180,180]

人脸注册 返回示例

```
{
 "face_token": "2fa64a88a9d5118916f9a303782a97d3",
 "location": {
    "left": 117,
    "top": 131,
    "width": 172,
    "height": 170,
    "rotation": 4
 }
}
```

人脸更新

用于对人脸库中指定用户，更新其下的人脸图像。

> **说明：**针对一个uid执行更新操作，新上传的人脸图像将覆盖此uid原有所有图像。

```
 var image = "取决于image_type参数，传入BASE64字符串或URL字符串或FACE_TOKEN字符串";

var imageType = "BASE64";

var groupId = "group1";

var userId = "user1";

// 调用人脸更新
client.updateUser(image, imageType, groupId, userId).then(function(result) {
    console.log(JSON.stringify(result));
}).catch(function(err) {
    // 如果发生网络错误
    console.log(err);
});

// 如果有可选参数
var options = {};
options["user_info"] = "user's info";
options["quality_control"] = "NORMAL";
options["liveness_control"] = "LOW";
options["action_type"] = "REPLACE";

// 带参数调用人脸更新
client.updateUser(image, imageType, groupId, userId, options).then(function(result) {
    console.log(JSON.stringify(result));
}).catch(function(err) {
    // 如果发生网络错误
    console.log(err);
});;
```

人脸更新 请求参数详情

参数名称	是否必选	类型	默认值	说明
image	是	string		图片信息(**总数据大小应小于10M**)，图片上传方式根据image_type来判断
image_type	是	string		图片类型 **BASE64**:图片的base64值，base64编码后的图片数据，编码后的图片大小不超过2M；**URL**:图片的 URL地址(可能由于网络等原因导致下载图片时间过长)；**FACE_TOKEN**: 人脸图片的唯一标识，调用人脸检测接口时，会为每个人脸图片赋予一个唯一的FACE_TOKEN，同一张图片多次检测得到的FACE_TOKEN是同一个
group_id	是	string		更新指定groupid下uid对应的信息
user_id	是	string		用户id（由数字、字母、下划线组成），长度限制128B
user_info	否	string		用户资料，长度限制256B
quality_control	否	string	NONE	图片质量控制 **NONE**: 不进行控制 **LOW**:较低的质量要求 **NORMAL**: 一般的质量要求 **HIGH**: 较高的质量要求 **默认 NONE**
liveness_control	否	string	NONE	活体检测控制 **NONE**: 不进行控制 **LOW**:较低的活体要求(高通过率 低攻击拒绝率) **NORMAL**: 一般的活体要求(平衡的攻击拒绝率, 通过率) **HIGH**: 较高的活体要求(高攻击拒绝率 低通过率) **默认NONE**
action_type	否	string	UPDATE	操作方式 **UPDATE**: 当user_id在库中已经存在时，对此user_id重复注册时，新注册的图片默认会追加到该user_id下,**REPLACE**：当对此user_id重复注册时,则会用新图替换库中该user_id下所有图片,默认使用**UPDATE**

人脸更新 返回数据参数详情

字段	必选	类型	说明
log_id	是	uint64	请求标识码，随机数，唯一
face_token	是	string	人脸图片的唯一标识
location	是	array	人脸在图片中的位置
+left	是	double	人脸区域离左边界的距离
+top	是	double	人脸区域离上边界的距离
+width	是	double	人脸区域的宽度
+height	是	double	人脸区域的高度
+rotation	是	int64	人脸框相对于竖直方向的顺时针旋转角，[-180,180]

人脸更新 返回示例

```
{
 "face_token": "2fa64a88a9d5118916f9a303782a97d3",
 "location": {
    "left": 117,
    "top": 131,
    "width": 172,
    "height": 170,
    "rotation": 4
 }
}
```

人脸删除

用于从人脸库中删除一个用户。

人脸删除注意事项：

- 删除的内容，包括用户所有图像和身份信息；
- 如果一个uid存在于多个用户组内，将会同时将从各个组中把用户删除
- 如果指定了group_id，则只删除此group下的uid相关信息

```
 var userId = "user1";

var groupId = "group1";

var faceToken = "face_token_23123";

// 调用人脸删除
client.faceDelete(userId, groupId, faceToken).then(function(result) {
    console.log(JSON.stringify(result));
}).catch(function(err) {
    // 如果发生网络错误
    console.log(err);
});
```

人脸删除 请求参数详情

参数名称	是否必选	类型	说明
log_id	是	uint64	请求标识码，随机数，唯一
user_id	是	string	用户id（由数字、字母、下划线组成），长度限制128B
group_id	是	string	用户组id（由数字、字母、下划线组成），长度限制128B
face_token	是	string	需要删除的人脸图片token，（由数字、字母、下划线组成）长度限制64B

人脸删除 返回数据参数详情

字段	是否必选	类型	说明
log_id	是	uint64	请求唯一标识码，随机数

人脸删除 返回示例

```
 // 删除成功
{
  "error_code": 0,
  "log_id": 73473737,
}
// 删除发生错误
{
  "error_code": 223106,
  "log_id": 1382953199,
  "error_msg": "face is not exist"
}
```

用户信息查询

获取人脸库中某个用户的信息(user_info信息和用户所属的组)。

```
 var userId = "user1";

var groupId = "group1";

// 调用用户信息查询
client.getUser(userId, groupId).then(function(result) {
    console.log(JSON.stringify(result));
}).catch(function(err) {
    // 如果发生网络错误
    console.log(err);
});
```

用户信息查询 请求参数详情

参数名称	是否必选	类型	说明
user_id	是	string	用户id（由数字、字母、下划线组成），长度限制128B
group_id	是	string	用户组id（由数字、字母、下划线组成），长度限制128B

用户信息查询 返回数据参数详情

字段	必选	类型	说明
log_id	是	uint64	请求标识码，随机数，唯一
user_list	是	array	查询到的用户列表
+user_info	是	string	用户资料，被查询用户的资料
+group_id	是	string	用户组id，被查询用户的所在组

用户信息查询 返回示例

```
{
 "user_list": [
    {
        "user_info": "user info ...",
        "group_id": "gid1"
    },
    {
        "user_info": "user info2 ...",
        "group_id": "gid2"
    }
 ]
}
```

获取用户人脸列表

用于获取一个用户的全部人脸列表。

```
 var userId = "user1";

var groupId = "group1";

// 调用获取用户人脸列表
client.faceGetlist(userId, groupId).then(function(result) {
    console.log(JSON.stringify(result));
}).catch(function(err) {
    // 如果发生网络错误
    console.log(err);
});
```

获取用户人脸列表 请求参数详情

参数名称	是否必选	类型	说明
user_id	是	string	用户id（由数字、字母、下划线组成），长度限制128B
group_id	是	string	用户组id（由数字、字母、下划线组成），长度限制128B

获取用户人脸列表 返回数据参数详情

字段	必选	类型	说明
log_id	是	uint64	请求标识码，随机数，唯一
face_list	是	array	人脸列表
+face_token	是	string	人脸图片的唯一标识
+ctime	是	string	人脸创建时间

获取用户人脸列表 返回示例

```
{
 "face_list": [
    {
        "face_token": "fid1",
        "ctime": "2018-01-01 00:00:00"
    },
    {
        "face_token": "fid2",
        "ctime": "2018-01-01 10:00:00"
    }
 ]
}
```

获取用户列表

用于查询指定用户组中的用户列表。

```
 var groupId = "group1";

// 调用获取用户列表
client.getGroupUsers(groupId).then(function(result) {
    console.log(JSON.stringify(result));
}).catch(function(err) {
    // 如果发生网络错误
    console.log(err);
});

// 如果有可选参数
var options = {};
options["start"] = "0";
options["length"] = "50";

// 带参数调用获取用户列表
client.getGroupUsers(groupId, options).then(function(result) {
    console.log(JSON.stringify(result));
}).catch(function(err) {
    // 如果发生网络错误
    console.log(err);
});;
```

获取用户列表 请求参数详情

参数名称	是否必选	类型	默认值	说明
group_id	是	string		用户组id（由数字、字母、下划线组成），长度限制128B
start	否	string	0	默认值0，起始序号
length	否	string	100	返回数量，默认值100，最大值1000

获取用户列表 返回数据参数详情

字段	必选	类型	说明
user_id_list	是	array	用户ID列表

获取用户列表 返回示例

```
 {
 "user_id_list": [
    "uid1",
    "uid2"
 ]
}
```

复制用户

用于将已经存在于人脸库中的用户**复制到一个新的组**。

```
 var userId = "user1";

// 调用复制用户
client.userCopy(userId).then(function(result) {
    console.log(JSON.stringify(result));
}).catch(function(err) {
    // 如果发生网络错误
    console.log(err);
});

// 如果有可选参数
var options = {};
options["src_group_id"] = "11111";
options["dst_group_id"] = "222222";

// 带参数调用复制用户
client.userCopy(userId, options).then(function(result) {
    console.log(JSON.stringify(result));
}).catch(function(err) {
    // 如果发生网络错误
    console.log(err);
});;
```

复制用户 请求参数详情

参数名称	是否必选	类型	说明
user_id	是	string	用户id（由数字、字母、下划线组成），长度限制128B
src_group_id	否	string	从指定组里复制信息
dst_group_id	否	string	需要添加用户的组id

复制用户 返回数据参数详情

字段	必选	类型	说明
log_id	是	id	log_id

复制用户 返回示例

```
 // 正确返回值
{
   "error_code": 0,
  "log_id": 3314921889,
}
// 发生错误时返回值
{
  "error_code": 216100,
  "log_id": 3111284097,
  "error_msg": "already add"
}
```

删除用户

用于将用户从某个组中删除。

```
 var groupId = "group1";

var userId = "user1";

// 调用删除用户
client.deleteUser(groupId, userId).then(function(result) {
    console.log(JSON.stringify(result));
}).catch(function(err) {
    // 如果发生网络错误
    console.log(err);
});
```

删除用户 请求参数详情

参数名称	是否必选	类型	说明
group_id	是	string	用户组id（由数字、字母、下划线组成），长度限制128B
user_id	是	string	用户id（由数字、字母、下划线组成），长度限制128B

删除用户 返回数据参数详情

字段	是否必选	类型	说明
log_id	是	uint64	请求唯一标识码，随机数

删除用户 返回示例

```
 // 正确返回值
{

 "error_code": 0,
 "log_id": 3314921889,
}
// 发生错误时返回值
{
 "error_code":  223103,
 "log_id": 815967402,
 "error_msg": "user is not exist"
}
```

创建用户组

用于创建一个空的用户组，如果用户组已存在 则返回错误。

```
 var groupId = "group1";

// 调用创建用户组
client.groupAdd(groupId).then(function(result) {
    console.log(JSON.stringify(result));
}).catch(function(err) {
    // 如果发生网络错误
    console.log(err);
});
```

创建用户组 请求参数详情

参数名称	是否必选	类型	说明
group_id	是	string	用户组id（由数字、字母、下划线组成），长度限制128B

创建用户组 返回数据参数详情

字段	必选	类型	说明
log_id	是	uint64	请求标识码，随机数，唯一

创建用户组 返回示例

```
 {

  "error_code": 0,
  "log_id": 3314921889,
}
// 发生错误时返回值
{
  "error_code":  223101,
  "log_id": 815967402,
  "error_msg": " group is already exist"
}
```

删除用户组

删除用户组下所有的用户及人脸，如果组不存在 则返回错误。

```
 var groupId = "group1";

// 调用删除用户组
client.groupDelete(groupId).then(function(result) {
    console.log(JSON.stringify(result));
}).catch(function(err) {
    // 如果发生网络错误
    console.log(err);
});
```

删除用户组 请求参数详情

参数名称	是否必选	类型	说明
group_id	是	string	用户组id（由数字、字母、下划线组成），长度限制128B

删除用户组 返回数据参数详情

字段	必选	类型	说明
log_id	是	uint64	请求标识码，随机数，唯一

删除用户组 返回示例

```
 // 正确返回值
{

  "error_code":0,
  "log_id": 3314921889,
}
// 发生错误时返回值
{
  "error_code": 223100,
  "log_id": 815967402,
  "error_msg": " group is not exist"
}
```

组列表查询

用于查询用户组的列表。

```
 // 调用组列表查询
client.getGrouplist().then(function(result) {
    console.log(JSON.stringify(result));
}).catch(function(err) {
    // 如果发生网络错误
    console.log(err);
});

// 如果有可选参数
var options = {};
options["start"] = "0";
options["length"] = "50";

// 带参数调用组列表查询
client.getGrouplist(, options).then(function(result) {
    console.log(JSON.stringify(result));
}).catch(function(err) {
    // 如果发生网络错误
    console.log(err);
});;
```

组列表查询 请求参数详情

参数名称	是否必选	类型	默认值	说明
start	否	string	0	默认值0，起始序号
length	否	string	100	返回数量，默认值100，最大值1000

组列表查询 返回数据参数详情

字段	必选	类型	说明
group_id_list	是	array	group

组列表查询 返回示例

```
 {
   "group_id_list": [
      "gid1",
      "gid2"
   ]
}
```

身份验证

质量检测（可选）活体检测（可选）公安验证（必选）

```
 var image = "取决于image_type参数，传入BASE64字符串或URL字符串或FACE_TOKEN字符串";

var imageType = "BASE64";

var idCardNumber = "110233112299822211";

var name = "张三";

// 调用身份验证
client.personVerify(image, imageType, idCardNumber, name).then(function(result) {
    console.log(JSON.stringify(result));
}).catch(function(err) {
    // 如果发生网络错误
    console.log(err);
});

// 如果有可选参数
var options = {};
options["quality_control"] = "NORMAL";
options["liveness_control"] = "LOW";

// 带参数调用身份验证
client.personVerify(image, imageType, idCardNumber, name, options).then(function(result) {
    console.log(JSON.stringify(result));
}).catch(function(err) {
    // 如果发生网络错误
    console.log(err);
});;
```

身份验证 请求参数详情

参数名称	是否必选	类型	默认值	说明
image	是	string		图片信息(**总数据大小应小于10M**)，图片上传方式根据image_type来判断
image_type	是	string		图片类型 **BASE64**:图片的base64值，base64编码后的图片大小不超过2M；**URL**:图片的 URL地址(可能由于网络等原因导致下载图片时间过长)；**FACE_TOKEN**: 人脸图片的唯一标识，调用人脸检测接口时，会为每个人脸图片赋予一个唯一的FACE_TOKEN，同一张图片多次检测得到的FACE_TOKEN是同一个
id_card_number	是	string		身份证号（真实身份证号号码）
name	是	string		utf8，姓名（真实姓名，和身份证号匹配）
quality_control	否	string	NONE	图片质量控制 **NONE**: 不进行控制 **LOW**:较低的质量要求 **NORMAL**: 一般的质量要求 **HIGH**: 较高的质量要求 **默认 NONE**
liveness_control	否	string	NONE	活体检测控制 **NONE**: 不进行控制 **LOW**:较低的活体要求(高通过率 低攻击拒绝率) **NORMAL**: 一般的活体要求(平衡的攻击拒绝率, 通过率) **HIGH**: 较高的活体要求(高攻击拒绝率 低通过率) **默认NONE**

身份验证 返回数据参数详情

参数	必须	类型	说明
log_id	是	uint64	日志id
score	是	float	与公安小图相似度可能性，用于验证生活照与公安小图是否为同一人，有正常分数时为[0~1]，推荐阈值0.8，超过即判断为同一人

身份验证 返回示例

```
 {
  "score": 44.3,
}
```

语音校验码接口

此接口主要用于生成随机码，用于视频的语音识别校验使用，以判断视频的即时性，而非事先录制的，提升作弊的难度。

```
 // 调用语音校验码接口
client.videoSessioncode().then(function(result) {
    console.log(JSON.stringify(result));
}).catch(function(err) {
    // 如果发生网络错误
    console.log(err);
});

// 如果有可选参数
var options = {};
options["appid"] = "223245";

// 带参数调用语音校验码接口
client.videoSessioncode(, options).then(function(result) {
    console.log(JSON.stringify(result));
}).catch(function(err) {
    // 如果发生网络错误
    console.log(err);
});;
```

语音校验码接口 请求参数详情

参数名称	是否必选	类型	说明
appid	否	string	百度云创建应用时的唯一标识ID

语音校验码接口 返回数据参数详情

字段	必选	类型	说明
session_id	是	string	语音校验码会话id
code	是	string	语音验证码，数字形式，3~6位数字

语音校验码接口 返回示例

```
{
 "err_no": 0,
 "err_msg": "SUCCESS",
 "result": {
  "session_id": "S59faeeebb9111890355690",
  "code": "9940"
 },
 "timestamp": 1509617387,
 "cached": 0,
 "serverlogid": "0587756642"
}
```


在线活体检测

接口能力

- **人脸基础信息**：包括人脸框位置，人脸空间旋转角度，人脸置信度等信息。
- **人脸质量检测**：判断人脸的遮挡、光照、模糊度、完整度等质量信息。可用于判断上传的人脸是否符合标准。
- **基于图片的活体检测**：基于单张图片，判断图片中的人脸是否为二次翻拍（举例：如用户A用手机拍摄了一张包含人脸的图片一，用户B翻拍了图片一得到了图片二，并用图片二伪造成用户A去进行识别操作，这种情况普遍发生在金融开户、实名认证等环节）。此能力可用于H5场景下的一些人脸采集场景中，增加人脸注册的安全性和真实性。

```
client.faceverify([{
      image: new Buffer(fs.readFileSync('./assets/face/sample.jpg')).toString('base64'),
      image_type: 'BASE64'
  },{
      image: new Buffer(fs.readFileSync('./assets/face/sample.jpg')).toString('base64'),
      image_type: 'BASE64'
  }]).then(function (result) {
  console.log('<faceverify>: ' + JSON.stringify(result));
});
```

请求参数

参数	是否必选	类型	说明
image	是	string	图片信息(总数据大小应小于10M)，图片上传方式根据image_type来判断； **可以上传同一个用户的1张、3张或8张图片来进行活体判断，注：后端会选择每组照片中的最高分数作为整体分数。**
image_type	是	string	图片类型 **BASE64**:图片的base64值，base64编码后的图片数据，编码后的图片大小不超过2M； **URL**:图片的 URL地址(可能由于网络等原因导致下载图片时间过长)； **FACE_TOKEN**: 人脸图片的唯一标识，调用人脸检测接口时，会为每个人脸图片赋予一个唯一的FACE_TOKEN，同一张图片多次检测得到的FACE_TOKEN是同一个。
face_field	否	string	包括**age,expression,faceshape,gender,glasses,landmark,quality,facetype信息**，逗号分隔，默认只返回face_token、活体数、人脸框、概率和旋转角度

返回参数

参数	类型	是否必须	说明
face_liveness	是	float	活体分数值

thresholds	是	array	由服务端返回最新的阈值数据（随着模型的优化，阈值可能会变化），将此参数与返回的face_liveness进行比较，可以作为活体判断的依据。 frr_1e-4：万分之一误识率的阈值；frr_1e-3：千分之一误识率的阈值；frr_1e-2：百分之一误识率的阈值。误识率越低，准确率越高，相应的拒绝率也越高
face_list	是	array	每张图片的详细信息描述，如果只上传一张图片，则只返回一个结果。
+face_token	是	string	人脸图片的唯一标识
+location	是	array	人脸在图片中的位置
++left	是	double	人脸区域离左边界的距离
++top	是	double	人脸区域离上边界的距离
++width	是	double	人脸区域的宽度
++height	是	double	人脸区域的高度
++rotation	是	int64	人脸框相对于竖直方向的顺时针旋转角，[-180,180]
+face_probability	是	double	人脸置信度，范围【0~1】，代表这是一张人脸的概率，0最小、1最大。
+angel	是	array	人脸旋转角度参数
++yaw	是	double	三维旋转之左右旋转角[-90(左), 90(右)]
++pitch	是	double	三维旋转之俯仰角度[-90(上), 90(下)]
++roll	是	double	平面内旋转角[-180(逆时针), 180(顺时针)]
+age	否	double	年龄 ，当**face_field包含age时返回**
+expression	否	array	表情，当 **face_field包含expression时返回**
++type	否	string	**none**:不笑；**smile**:微笑；**laugh**:大笑
++probability	否	double	表情置信度，范围【0~1】，0最小、1最大。
+face_shape	否	array	脸型，当**face_field包含faceshape时返回**
++type	否	double	**square**: 正方形 **triangle**:三角形 **oval**: 椭圆 **heart**: 心形 **round**: 圆形
++probability	否	double	置信度，范围【0~1】，代表这是人脸形状判断正确的概率，0最小、1最大。
+gender	否	array	性别，**face_field包含gender时返回**
++type	否	string	male:**男性** female:**女性**
++probability	否	double	性别置信度，范围【0~1】，0代表概率最小、1代表最大。
+glasses	否	array	是否带眼镜，**face_field包含glasses时返回**
++type	否	string	**none**:无眼镜，**common**:普通眼镜，**sun**:墨镜

++probability	否	double	眼镜置信度，范围【0~1】，0代表概率最小、1代表最大。
+face_type	否	array	真实人脸/卡通人脸 **face_field包含facetype时返回**
++type	否	string	**human**: 真实人脸 **cartoon**: 卡通人脸
++probability	否	double	人脸类型判断正确的置信度，范围【0~1】，0代表概率最小、1代表最大。
+landmark	否	array	4个关键点位置，左眼中心、右眼中心、鼻尖、嘴中心。**face_field包含landmark时返回**
+landmark72	否	array	72个特征点位置 **face_field包含landmark时返回**
+quality	否	array	人脸质量信息。**face_field包含quality时返回**
++occlusion	否	array	人脸各部分遮挡的概率，范围[0~1]，0表示完整，1表示不完整
+++left_eye	否	double	左眼遮挡比例，[0-1] ， 1表示完全遮挡
+++right_eye	否	double	右眼遮挡比例，[0-1] ， 1表示完全遮挡
+++nose	否	double	鼻子遮挡比例，[0-1] ， 1表示完全遮挡
+++mouth	否	double	嘴巴遮挡比例，[0-1] ， 1表示完全遮挡
+++left_cheek	否	double	左脸颊遮挡比例，[0-1] ， 1表示完全遮挡
+++right_cheek	否	double	右脸颊遮挡比例，[0-1] ， 1表示完全遮挡
+++chin	否	double	下巴遮挡比例，，[0-1] ， 1表示完全遮挡
++blur	否	double	人脸模糊程度，范围[0~1]，0表示清晰，1表示模糊
++illumination	否	double	取值范围在[0~255], 表示脸部区域的光照程度 越大表示光照越好
++completeness	否	int64	人脸完整度，0或1, 0为人脸溢出图像边界，1为人脸都在图像边界内
+parsing_info	否	string	人脸分层结果 结果数据是使用gzip压缩后再base64编码 使用前需base64解码后再解压缩 **原数据格式为string 形如0,0,0,0,0,1,1,1,1,1,1,2,2,2,2,2,2,2,2,2,2,…**

- 返回示例


```
{
"thresholds": {
    "frr_1e-4": 0.05,  //万分之一误拒率的阈值
    "frr_1e-3": 0.3,   //千分之一误拒率的阈值
    "frr_1e-2": 0.9    //百分之一误拒率的阈值
},
"face_liveness": 0.05532243927,
"face_list": [
    {
```

```
        "face_token": "df46f7c7db4aa09a093c26fb8d1a8d44",
        "location": {
            "left": 328.9026489,
            "top": 97.16340637,
            "width": 162,
            "height": 154,
            "rotation": 32
        },
        "face_probability": 1,
        "angle": {
            "yaw": 10.16196251,
            "pitch": 2.244354248,
            "roll": 33.82199097
        },
        "liveness": {
            "faceliveness": 0.004187555984,
            "livemapscore": 0.04492170034
        },
        "age": 23
    },
    {
        "face_token": "901d2c64274fccd687d311a6e6110a01",
        "location": {
            "left": 411.4876404,
            "top": 166.3593445,
            "width": 329,
            "height": 308,
            "rotation": 45
        },
        "face_probability": 0.9194830656,
        "angle": {
            "yaw": -1.716423035,
            "pitch": 7.344647408,
            "roll": 45.79914856
        },
        "liveness": {
            "faceliveness": 0.0001665892196,
            "livemapscore": 0.001787073661
        },
        "age": 23
    },
    {
        "face_token": "7d57e36981c48b4946eb97c8d838b02a",
        "location": {
            "left": 161.4559937,
            "top": 199.8726501,
            "width": 218,
            "height": 201,
            "rotation": -1
        },
        "face_probability": 1,
        "angle": {
            "yaw": -8.187754631,
            "pitch": 6.973727226,
            "roll": -1.25429821
        },
        "liveness": {
            "faceliveness": 0.02942637168,
            "livemapscore": 0.05532243927
        },
        "age": 23
    }
```

```
    ]
}
```

人脸对比

接口能力

- **两张人脸图片相似度对比**：比对两张图片中人脸的相似度，并返回相似度分值；
- **多种图片类型**：支持**生活照**、**证件照**、**身份证芯片照**、**带网纹照**四种类型的人脸对比；
- **活体检测**：基于图片中的破绽分析，判断其中的人脸是否为**二次翻拍**（举例：如用户A用手机拍摄了一张包含人脸的图片一，用户B翻拍了图片一得到了图片二，并用图片二伪造成用户A去进行识别操作，这种情况普遍发生在金融开户、实名认证等环节。）；
- **质量检测**：返回模糊、光照等质量检测信息，用于辅助判断图片是否符合识别要求；

业务应用

用于比对多张图片中的人脸相似度并返回两两比对的得分，可用于判断两张脸是否是同一人的可能性大小。

典型应用场景：如**人证合一验证**，**用户认证**等，可与您现有的人脸库进行比对验证。

```
client.match([{
    image: new Buffer(fs.readFileSync('./assets/face/sample.jpg')).toString('base64'),
    image_type: 'BASE64'
},{
    image: new Buffer(fs.readFileSync('./assets/face/sample.jpg')).toString('base64'),
    image_type: 'BASE64'
}]).then(function (result) {
    console.log('<match>: ' + JSON.stringify(result));
});
```

请求参数

参数	必选	类型	说明
image	是	string	图片信息(**总数据大小应小于10M**)，图片上传方式根据image_type来判断。 **两张图片通过json格式上传，格式参考表格下方示例**
image_type	是	string	图片类型 **BASE64**:图片的base64值，base64编码后的图片数据，编码后的图片大小不超过2M；**URL**:图片的 URL地址(可能由于网络等原因导致下载图片时间过长)；**FACE_TOKEN**: 人脸图片的唯一标识，调用人脸检测接口时，会为每个人脸图片赋予一个唯一的FACE_TOKEN，同一张图片多次检测得到的FACE_TOKEN是同一个。
face_type	否	string	人脸的类型 **LIVE**表示生活照：通常为手机、相机拍摄的人像图片、或从网络获取的人像图片等，**IDCARD**表示身份证芯片照：二代身份证内置芯片中的人像照片， **WATERMARK**表示带水印证件照：一般为带水印的小图，如公安网小图 **CERT**表示证件照片：如拍摄的身份证、工卡、护照、学生证等证件图片 默认LIVE
quality_control	否	string	图片质量控制 **NONE**: 不进行控制 **LOW**:较低的质量要求 **NORMAL**: 一般的质量要求 **HIGH**: 较高的质量要求 **默认 NONE**
liveness_control	否	string	活体检测控制 **NONE**: 不进行控制 **LOW**:较低的活体要求(高通过率 低攻击拒绝率) **NORMAL**: 一般的活体要求(平衡的攻击拒绝率, 通过率) **HIGH**: 较高的活体要求(高攻击拒绝率 低通过率) **默认NONE**

返回参数

参数名	必选	类型	说明
score	是	float	人脸相似度得分
face_list	是	array	人脸信息列表
+face_token	是	string	人脸的唯一标志

返回示例

```
{
  "score": 44.3,
  "face_list": [  //返回的顺序与传入的顺序保持一致
     {
        "face_token": "fid1"
     },
     {
        "face_token": "fid2"
     }
  ]
}
```

人脸实名认证V4

能力介绍

1. 业务能力

- **质量检测（可选）**：判断图片中是否包含人脸，以及人脸在姿态、遮挡、模糊、光照等方面是否符合识别条件。
- **活体检测（可选）**：基于图片中的破绽分析，判断其中的人脸是否为**二次翻拍**（举例：如用户A用手机拍摄了一张包含人脸的图片一，用户B翻拍了图片一得到了图片二，并用图片二伪造成用户A去进行识别操作，这种情况普遍发生在金融开户、实名认证等环节）。
- **图片加密及风控（可选）**：当配合增强级采集SDK、增强级安全加固采集SDK、金融级采集SDK、金融级安全加固采集SDK版本使用，对采集SDK输出的加密图片进行解密（加密传输可以有效避免第三方非法黑产绕过APP模拟请求攻击云端接口的行为，Eg：脚本攻击等）；以及结合百度安全实验室大数据风控能力，对采集SDK的发起端设备进行风控识别，辨别是否为风险设备，Eg：ROM注入、视频劫持等；
- **人脸实名认证（必选）**：基于姓名和身份证号，调取公安权威数据源人脸图，将当前获取的人脸图片，与公安数据源人脸图进行对比，得出比对分数，并基于此进行业务判断是否为同一人。由于公安数据源人脸图具有最权威的身份证明作用，故对用户本人的验证结果可信度也最为合理。

2. 业务逻辑

- 上述能力，人脸实名认证能力为必选能力，质量检测、活体检测、图片加密及风控为可选能力，验证顺序为`人脸质量检测->活体检测->人脸实名认证`。
- **如选择了质量检测和活体检测能力，则有任意一个条件不通过，整个请求流程终止，并返回错误码，描述具体的不符合信息**。
- 基于此顺序串行验证逻辑，可以避免大量不符合条件的请求流转到人脸实名认证，节约您的成本。

3. 推荐阈值

- 此接口使用的对比算法，针对带水纹证件照采用了专项的模型处理，可保证水纹信息的影响降到尽可能低。
- 如比对成功，最终返回的有效数据为一个**对比分值**，在0~100之间，您可以设定具体的阈值来判断是否验证通过。
- **人证相似度的推荐阈值为80，对应的误识率为万分之一**。

```
 // 必填参数
var idCardNumber = '';
var name = '';
var image = '';

// 选填参数
var options = {};
options["xxx"] = "xxx";

// 调用接口
client.faceMingJingVerify(idCardNumber, name, image, options).then(function (result) {
    console.log(JSON.stringify(result));
}).catch(function(err) {
    // 如果发生网络错误
    console.log(err);
});
```

请求参数

参数	必选	类型	说明
app	否	string	APP端类型，配合采集SDK使用时须传入 **ios**：iOS端采集SDK **android**：安卓端采集SDK
sec_level	否	string	SDK安全级别，配合采集SDK使用时须传入，默认common **common**：配合4.1/4.1.5版本采集SDK，人脸图片未进行加密处理 **lite**: 配合5.2版本SDK
skey	否	string	使用5.2版本SDK请求时必填 **skey**：从SDK获取的密钥信息skey
x_device_id	否	string	使用5.2版本SDK请求时必填 **deviceId**：从SDK 获取的密钥信息deviceId
data	否	string	使用5.2版本SDK请求时必填， SDK输出的加密数据
id_card_number	是	string	身份证件号
name	是	string	姓名(需要是 utf8 编码)
livenesss_control	否	string	活体控制参数 **NONE**: 不进行控制 **LOW**:较低的活体要求(高通过率 低攻击拒绝率) **NORMAL**: 一般的活体要求(平衡的攻击拒绝率, 通过率) **HIGH**: 较高的活体要求(高攻击拒绝率 低通过率) **默认为NONE**
spoofing_control	否	string	合成图控制参数 **NONE**: 不进行控制 **LOW**:较低的合成图阈值数值，由于合成图判定逻辑为大于阈值视为合成图攻击，该项代表**低通过率、高攻击拒绝率** **NORMAL**: 一般的合成图阈值数值，由于合成图判定逻辑为大于阈值视为合成图攻击，该项代表**平衡的攻击拒绝率, 通过率** **HIGH**: 较高的合成图阈值数值，由于合成图判定逻辑为大于阈值视为合成图攻击，该项代表**高通过率、低攻击拒绝率**) **默认为NONE**
quality_control	否	string	质量控制参数 **NONE**: 不进行控制 **LOW**:较低的质量要求 **NORMAL**: 一般的质量要求 **HIGH**: 较高的质量要求 **默认为NONE**
image	是	string	图片信息(数据大小应小于10M 分辨率应小于1920*1080)，**5.2版本SDK请求时已包含在加密数据data中，无需额外传入**
image_type	否	string	图片类型 **BASE64**：图片的base64值 **URL**：图片的 URL **FACE_TOKEN**：人脸标识 **默认 BASE64**

返回参数

参数	类型	说明
log_id	number	调用的日志id
result	jsonObject	认证返回的结果
+score	float	与公安数据源人脸图相似度可能性，用于验证生活照与公安数据源人脸图是否为同一人，有正常分数时为[0~100]，推荐阈值80，超过即判断为同一人
+verify_status	number	认证状态，取值如下： 0 ： 正常 1 ： 身份证号与姓名不匹配或该身份证号不存在 2 ： 公安网图片不存在或质量过低
dec_image	string	对SDK传入的加密图片进行解密。 仅APP场景且进行了图片加密时，此参数返回解密后的人脸图片信息
risk_level	string	判断设备是否发生过风险行为来判断风险级别，取值（数值由高到低）： 1 – 高危 2 – 嫌疑 3 – 普通 4 – 正常
risk_tag	string	风险标签，若判断为有风险，则会有风险标签json 数组告知风险类型 例如：general_inject

返回示例


```
{
  "log_id": 1370579072568000512,
  "result": {
     "score": 40.884,
     "verify_status": 0
  },
  "dec_image": "/9j/4AAQSkZJRgABAgAAAQABAAD",
  "risk_level": "3",
  "risk_tag": [
     "若判断为有风险，则会有风险标签json 数组告知风险类型，如：general_inject"
  ]
}
```


人脸比对V4

能力介绍

1. 接口能力

- **两张人脸图片相似度对比**：对比两张图片中人脸的相似度，并返回相似度分值；
- **多种图片类型**：支持**生活照**、**证件照**、**身份证芯片照**、**带网纹照**四种类型的人脸对比；
- **活体检测控制**：基于图片中的破绽分析，判断其中的人脸是否为**二次翻拍**（举例：如用户A用手机拍摄了一张包含人脸的图片一，用户B翻拍了图片一得到了图片二，并用图片二伪造成用户A去进行识别操作，这种情况普遍发生在金融开户、实名认证等环节。）；
- **质量检测控制**：分析图片的中人脸的模糊度、角度、光照强度等特征，判断图片质量；
- **图片加密及风控：**配合增强级、金融级采集SDK使用
  - 对采集SDK输出的加密图片进行解密（加密传输可以有效避免第三方非法黑产**绕过APP模拟请求攻击云端接口的行为**，Eg：脚本攻击等）；
  - 以及结合百度安全实验室大数据风控能力，对采集SDK的发起端设备进行风控识别，辨别是否为**风险设备**，Eg：ROM注入、视频劫持等；

2. 业务应用

- 用于对比多张图片中的人脸相似度并返回两两对比的得分，可用于判断两张脸是否是同一人的可能性大小。

- 典型应用场景：如**人证合一验证**，**用户认证**等，可与您现有的人脸库进行对比验证。

```
 // 必填参数
var image = '';
var imageType = '';
var registerImage = '';
var registerImageType = '';

// 选填参数
var options = {};
options["xxx"] = "xxx";

// 调用接口
client.faceMingJingMatch(image, imageType, registerImage, registerImageType, options).then(function (result) {
    console.log(JSON.stringify(result));
}).catch(function(err) {
    // 如果发生网络错误
    console.log(err);
});
```

请求参数

参数	必选	类型	说明
app	否	string	APP类型，配合采集SDK使用时须传入 **ios**：iOS端采集SDK **android**：安卓端采集SDK
sec_level	否	string	SDK安全级别，配合采集SDK使用时须传入，默认common **common**：配合4.1/4.1.5版本SDK使用，人脸图片未进行加密处理 **lite**：配合5.2版本SDK使用
skey	否	string	使用5.2版本SDK请求时必填 从SDK获取的密钥信息
x_device_id	否	string	使用5.2版本SDK请求时必填 从SDK获取的密钥信息
data	否	string	使用5.2版本SDK请求时必填 SDK输出的加密数据
image	是	string	图片信息(数据大小应小于10M 分辨率应小于1920*1080)，***5.2版本SDK请求时已包含在加密数据data中，无需额外传入**
image_type	是	string	图片类型 **BASE64**：图片的base64值 **URL**：图片的URL **FACE_TOKEN**：人脸标识 默认 BASE64
face_type	否	string	人脸的类型 **LIVE**：表示生活照：通常为手机、相机拍摄的人像图片、或从网络获取的人像图片等 **IDCARD**：表示身份证芯片照：二代身份证内置芯片中的人像照片 **WATERMARK**：表示带水印证件照：一般为带水印的小图，如公安网小图 **CERT**：表示证件照片：如拍摄的身份证、工卡、护照、学生证等证件图片 **INFRARED** 表示红外照片：使用红外相机拍摄的照片 默认LIVE
quality_control	否	string	质量控制 **NONE**: 不进行控制 **LOW**:较低的质量要求 **NORMAL**: 一般的质量要求 **HIGH**: 较高的质量要求 **默认NONE**
liveness_control	否	string	活体控制 **NONE**: 不进行控制 **LOW**:较低的活体要求(高通过率 低攻击拒绝率) **NORMAL**: 一般的活体要求(平衡的攻击拒绝率, 通过率) **HIGH**: 较高的活体要求(高攻击拒绝率 低通过率) **默认NONE**
registerimage	是	string	图片信息(总数据大小应小于10M)，图片上传方式根据imagetype来判断。本图片特指客户服务器上传图片，非加密图片Base64值
registerimagetype	是	string	图片类型 **BASE64**：图片的base64值 **URL**：图片的 URL **FACE_TOKEN**：人脸标识 默认 BASE64
registerfacetype	否	string	人脸的类型
registerqualitycontrol	否	string	图片质量控制 **NONE**: 不进行控制 **LOW**:较低的质量要求 **NORMAL**: 一般的质量要求 **HIGH**: 较高的质量要求 **默认 NONE** 若图片质量不满足要求，则返回结果中会提示质量检测失败
registerlivenesscontrol	否	string	活体检测控制 **NONE**: 不进行控制 **LOW**:较低的活体要求(高通过率 低攻击拒绝率) **NORMAL**: 一般的活体要求(平衡的攻击拒绝率, 通过率) **HIGH**: 较高的活体要求(高攻击拒绝率 低通过率) **默认 NONE** 若活体检测结果不满足要求，则返回结果中会提示活体检测失败
facesorttype	否	int	人脸检测排序类型 0:代表检测出的人脸按照人脸面积从大到小排列 1:代表检测出的人脸按照距离图片中心从近到远排列 默认为0

返回参数

参数名	类型	说明
log_id	number	调用的日志id
result	jsonObject	认证返回的结果
+ score	number	人脸相似度得分，推荐阈值80分
+ face_list	jsonArray	人脸信息列表
++ face_token	string	人脸标志
dec_image	string	APP场景传入加密图片时，该项返回解密后的图片
risk_level	string	风控返回参数，只有在 risk_identify 为 true 时才返回，判断设备是否发生过风险行为来判断风险级别，取值（数值有高到低）： 1 – 高危 2 – 嫌疑 3 – 普通 4 – 正常
risk_tag	jsonArray	风控返回参数，只有在 risk_identify 为 true 时才返回，风险标签，若判断为有风险，则会有风险标签 json 数组告知风险类型
risk_warn_code	number	风控返回参数，只有在 risk_identify 为 true 时才返回，只有在风控服务异常的时候才返回这个字段

返回示例


```
{
  "log_id": 1370585066551377920,
  "result": {
    "score": 99.06919861,
    "face_list": [
      {
        "face_token": "549f9f1d1c7ec8c86931540b1939e8ed"
      },
      {
        "face_token": "1a319460ef89e8d27fb59062a28dbad7"
      }
    ]
  },
  "dec_image": "/9j/4AAQSkZJRgABAQAAAQABAAD",
  "risk_level": "3",
  "risk_tag": [
    "空"
  ]
}
```


在线图片活体V4

能力介绍

1. 接口能力

- **人脸基础信息**：包括人脸框位置，人脸空间旋转角度，人脸置信度等信息。
- **人脸质量检测**：判断人脸的遮挡、光照、模糊度、完整度等质量信息。可用于判断上传的人脸是否符合标准。
- **基于图片的活体检测**：基于单张图片，判断图片中的人脸是否为二次翻拍（举例：如用户A用手机拍摄了一张包含人脸的图片一，用户B翻拍了图片一得到了图片二，并用图片二伪造成用户A去进行识别操作，这种情况普遍发生在金融开户、实名

认证等环节）以及是否为合成图攻击。此能力可用于H5场景下的一些人脸采集场景中，增加人脸注册的安全性和真实性。

- **图片加密及风控**：配合采集SDK5.0版本使用，对采集SDK输出的加密图片进行解密（加密传输可以有效避免第三方非法黑产绕过APP模拟请求攻击云端接口的行为，Eg：脚本攻击等）；以及结合百度安全实验室大数据风控能力，对采集SDK的发起端设备进行风控识别，辨别是否为风险设备，Eg：ROM注入、视频劫持等；

```
 // 必填参数
var sdkVersion = '';

// 选填参数
var options = {};
options["xxx"] = "xxx";

// 调用接口
client.onlinePictureLiveV4(sdkVersion, options).then(function (result) {
    console.log(JSON.stringify(result));
}).catch(function(err) {
    // 如果发生网络错误
    console.log(err);
});
```

请求参数

参数	是否必选	类型	说明
sdk_version	是	string	1 ：非加密图片，适用于4.1/4.1.5版本SDK、H5场景或纯服务端场景 4：适用于5.2版本SDK
face_field	否	string	包括**age,expression,face_shape,gender,glasses,landmark,quality,face_type,spoofing**信息，逗号分隔，默认只返回face_token、活体数、人脸框、概率和旋转角度
option	否	string	场景信息，程序会视不同的场景选用相对应的模型。当前支持的场景有COMMON(通用场景)，GATE(闸机场景)，FINANCE(金融场景)，LOGISTICS(物流场景)，INTERNET(泛互联网场景), **默认使用COMMON** 。注意：如果请求参数中存在多个option值时，则取第一个option的值，同时除通用场景外的其他场景需要联系工作人员对您所使用的appid进行配置
app	否	string	端类型 ios / android **5.2版本SDK必传该项**
s_key	否	string	端上提供的用于解密图片的skey **5.2版本SDK必传该项**
device_id	否	string	端上提供的用于解密图片的deviceId **5.2版本SDK必传该项**
data	否	string	端上提供的加密后的图片数组 **5.2版本SDK必传该项**
image_list	否	array	图片BASE64数组 4.1/4.1.5版本SDK、H5场景或纯服务端场景必填

返回参数

参数	是否必须	类型	说明
log_id	是	string	日志id
error_code	是	int	错误码状态，若为0则表示认证成功
error_msg	是	string	错误码说明，若为 success 则表示认证成功
result	是	object	活体结果

+face_liveness	是	float	活体分数值
+thresholds	是	array	由服务端返回最新的阈值数据（随着模型的优化，阈值可能会变化），将此参数与返回的face_liveness进行比较，可以作为活体判断的依据。 frr_1e-4：万分之一误识率的阈值； frr_1e-3：千分之一误识率的阈值； frr_1e-2：百分之一误识率的阈值。 误识率越低，准确率越高，相应的拒绝率也越高
+face_list	是	array	每张图片的详细信息描述，如果只上传一张图片，则只返回一个结果。
++face_token	是	string	人脸图片的唯一标识
++location	是	array	人脸在图片中的位置
+++left	是	double	人脸区域离左边界的距离
+++top	是	double	人脸区域离上边界的距离
+++width	是	double	人脸区域的宽度
+++height	是	double	人脸区域的高度
+++rotation	是	int64	人脸框相对于竖直方向的顺时针旋转角，[-180,180]
++face_probability	是	double	人脸置信度，范围【0~1】，代表这是一张人脸的概率，0最小、1最大。
++angel	是	array	人脸旋转角度参数
+++yaw	是	double	三维旋转之左右旋转角[-90(左), 90(右)]
+++pitch	是	double	三维旋转之俯仰角度[-90(上), 90(下)]
+++roll	是	double	平面内旋转角[-180(逆时针), 180(顺时针)]
++age	否	double	年龄 ，当**face_field包含age时返回**
++expression	否	array	表情，当 **face_field包含expression时返回**
+++type	否	string	**none**:不笑；**smile**:微笑；**laugh**:大笑
+++probability	否	double	表情置信度，范围【0~1】，0最小、1最大。
++face_shape	否	array	脸型，当**face_field包含face_shape时返回**
+++type	否	double	**square**: 正方形 **triangle**:三角形 **oval**: 椭圆 **heart**: 心形 **round**: 圆形
+++probability	否	double	置信度，范围【0~1】，代表这是人脸形状判断正确的概率，0最小、1最大。
++gender	否	array	性别，**face_field包含gender时返回**

+++type	否	string	male:**男性** female:**女性**
+++probability	否	double	性别置信度，范围【0~1】，0代表概率最小、1代表最大。
++glasses	否	array	是否带眼镜，**face_field包含glasses时返回**
++type	否	string	**none**:无眼镜，**common**:普通眼镜，**sun**:墨镜
+++probability	否	double	眼镜置信度，范围【0~1】，0代表概率最小、1代表最大。
++face_type	否	array	真实人脸/卡通人脸 **face_field包含face_type时返回**
+++type	否	string	**human**: 真实人脸 **cartoon**: 卡通人脸
+++probability	否	double	人脸类型判断正确的置信度，范围【0~1】，0代表概率最小、1代表最大。
++landmark	否	array	4个关键点位置，左眼中心、右眼中心、鼻尖、嘴中心。**face_field包含landmark时返回**
++landmark72	否	array	72个特征点位置 **face_field包含landmark时返回**
++quality	否	array	人脸质量信息。**face_field包含quality时返回**
+++occlusion	否	array	人脸各部分遮挡的概率，范围[0~1]，0表示完整，1表示不完整
++++left_eye	否	double	左眼遮挡比例，[0-1] ， 1表示完全遮挡
++++right_eye	否	double	右眼遮挡比例，[0-1] ， 1表示完全遮挡
++++nose	否	double	鼻子遮挡比例，[0-1] ， 1表示完全遮挡
++++mouth	否	double	嘴巴遮挡比例，[0-1] ， 1表示完全遮挡
++++left_cheek	否	double	左脸颊遮挡比例，[0-1] ， 1表示完全遮挡
++++right_cheek	否	double	右脸颊遮挡比例，[0-1] ， 1表示完全遮挡
++++chin	否	double	下巴遮挡比例，，[0-1] ， 1表示完全遮挡
+++blur	否	double	人脸模糊程度，范围[0~1]，0表示清晰，1表示模糊
+++illumination	否	double	取值范围在[0~255], 表示脸部区域的光照程度 越大表示光照越好
+++completeness	否	int64	人脸完整度，0或1, 0为人脸溢出图像边界，1为人脸都在图像边界内

++spoofing	否	double	合成图打分 判断图片是否为合成图 face_field包含时返回spoofing
risk_level	否	string	判断设备是否发生过风险行为来判断风险级别，取值（数值有高到低）：1 – 高危 2 – 嫌疑 3 – 普通 4 – 正常
risk_tag	否	string	风险标签，若判断为有风险，则会有风险标签 json 数组告知风险类型
dec_image	否	array	非安全加固增强级采集sdk、非安全加固金融级采集sdk、安全加固增强级采集sdk、安全加固金融级采集sdk会返回解密后的原图

返回示例


```
{
  "result": {
    "thresholds": {
      "frr_1e-3": 0.3,
      "frr_1e-2": 0.9,
      "frr_1e-4": 0.05
    },
    "face_liveness": 0.1976952702,
    "face_list": [
      {
        "liveness": {
          "livemapscore": 0.1976952702
        },
        "angle": {
          "roll": 0.31,
          "pitch": -2.07,
          "yaw": 0.85
        },
        "face_token": "abd1b4ce743099336cfed40193ff4944",
        "location": {
          "top": 161.74,
          "left": 79.04,
          "rotation": 1,
          "width": 318,
          "height": 333
        },
        "face_probability": 1
      },
      {
        "liveness": {
          "livemapscore": 0.1976952702
        },
        "angle": {
          "roll": 0.31,
          "pitch": -2.07,
          "yaw": 0.85
        },
        "face_token": "abd1b4ce743099336cfed40193ff4944",
        "location": {
          "top": 161.74,
          "left": 79.04,
          "rotation": 1,
          "width": 318,
          "height": 333
        },
        "face_probability": 1
      },
      {
```

```
            "liveness": {
                "livemapscore": 0.1976952702
            },
            "angle": {
                "roll": 0.31,
                "pitch": -2.07,
                "yaw": 0.85
            },
            "face_token": "abd1b4ce743099336cfed40193ff4944",
            "location": {
                "top": 161.74,
                "left": 79.04,
                "rotation": 1,
                "width": 318,
                "height": 333
            },
            "face_probability": 1
        },
        {
            "liveness": {
                "livemapscore": 0.1976952702
            },
            "angle": {
                "roll": 0.31,
                "pitch": -2.07,
                "yaw": 0.85
            },
            "face_token": "abd1b4ce743099336cfed40193ff4944",
            "location": {
                "top": 161.74,
                "left": 79.04,
                "rotation": 1,
                "width": 318,
                "height": 333
            },
            "face_probability": 1
        },
        {
            "liveness": {
                "livemapscore": 0.1976952702
            },
            "angle": {
                "roll": 0.31,
                "pitch": -2.07,
                "yaw": 0.85
            },
            "face_token": "abd1b4ce743099336cfed40193ff4944",
            "location": {
                "top": 161.74,
                "left": 79.04,
                "rotation": 1,
                "width": 318,
                "height": 333
            },
            "face_probability": 1
        },
        {
            "liveness": {
                "livemapscore": 0.1976952702
            },
            "angle": {
                "roll": 0.31,
                "pitch": -2.07
```

```
                "pitch": -2.07,
                "yaw": 0.85
            },
            "face_token": "abd1b4ce743099336cfed40193ff4944",
            "location": {
                "top": 161.74,
                "left": 79.04,
                "rotation": 1,
                "width": 318,
                "height": 333
            },
            "face_probability": 1
        },
        {
            "liveness": {
                "livemapscore": 0.1976952702
            },
            "angle": {
                "roll": 0.31,
                "pitch": -2.07,
                "yaw": 0.85
            },
            "face_token": "abd1b4ce743099336cfed40193ff4944",
            "location": {
                "top": 161.74,
                "left": 79.04,
                "rotation": 1,
                "width": 318,
                "height": 333
            },
            "face_probability": 1
        },
        {
            "liveness": {
                "livemapscore": 0.1976952702
            },
            "angle": {
                "roll": 0.31,
                "pitch": -2.07,
                "yaw": 0.85
            },
            "face_token": "abd1b4ce743099336cfed40193ff4944",
            "location": {
                "top": 161.74,
                "left": 79.04,
                "rotation": 1,
                "width": 318,
                "height": 333
            },
            "face_probability": 1
        }
    ]
},
"risk_level": "3",
"log_id": 1423545654699779516,
"risk_tag": [
    "空"
],
"dec_image": [
        "/9j/4AAQ..."
]
}
```

错误信息

错误返回格式

若请求错误，服务器将返回的JSON文本包含以下参数：

- **error_code**：错误码。
- **error_msg**：错误描述信息，帮助理解和解决发生的错误。

错误码

错误码	错误信息	描述
1	Unknown error	服务器内部错误，请再次请求， 如果持续出现此类错误，请通过QQ群（860337848）或提交工单联系技术支持团队。
2	Service temporarily unavailable	服务暂不可用，请再次请求， 如果持续出现此类错误，请通过QQ群（860337848）或提交工单联系技术支持团队。
3	Unsupported openapi method	调用的API不存在，请检查请求URL后重新尝试，一般为URL中有非英文字符，如“-”，可手动输入重试
4	Open api request limit reached	集群超限额，请再次请求， 如果持续出现此类错误，请通过QQ群（860337848）或提交工单联系技术支持团队。
6	No permission to access data	无权限访问该用户数据，创建应用时未勾选相关接口
13	Get service token failed	获取token失败
14	IAM Certification failed	IAM 鉴权失败
15	app not exsits or create failed	应用不存在或者创建失败
17	Open api daily request limit reached	每天请求量超限额，可通过QQ群（860337848）联系群管、提交工单提升限额
18	Open api qps request limit reached	QPS超限额，可通过QQ群（860337848）联系群管、提交工单提升限额
19	Open api total request limit reached	请求总量超限额，可通过QQ群（860337848）联系群管、提交工单提升限额
100	Invalid parameter	无效的access_token参数，请检查后重新尝试
110	Access token invalid or no longer valid	access_token无效
111	Access token expired	access token过期
222001	param[] is null	必要参数未传入
222002	param[start] format error	参数格式错误
222003	param[length] format error	参数格式错误
222004	param[op_app_id_list] format error	参数格式错误
222005	param[group_id_list] format error	参数格式错误
222006	group_id format error	参数格式错误

222007	uid format error	参数格式错误
222008	face_id format error	参数格式错误
222009	quality_conf format error	参数格式错误
222010	user_info format error	参数格式错误
222011	param[uid_list] format error	参数格式错误
222012	param[op_app_id] format error	参数格式错误
222013	param[image] format error	参数格式错误
222014	param[app_id] format error	参数格式错误
222015	param[image_type] format error	参数格式错误
222016	param[max_face_num] format error	参数格式错误
222017	param[face_field] format error	参数格式错误
222018	param[user_id] format error	参数格式错误
222019	param[quality_control] format error	参数格式错误
222020	param[liveness_control] format error	参数格式错误
222021	param[max_user_num] format error	参数格式错误
222022	param[id_card_number] format error	参数格式错误
222023	param[name] format error	参数格式错误
222024	param[face_type] format error	参数格式错误
222025	param[face_token] format error	参数格式错误
222026	param[max_star_num] format error	参数格式错误
222201	network not available	服务端请求失败
222202	pic not has face	图片中没有人脸
222203	image check fail	无法解析人脸
222204	image_url_download_fail	从图片的url下载图片失败
222205	network not availablel	服务端请求失败
222206	rtse service return fail	服务端请求失败
222207	match user is not found	未找到匹配的用户

	found	
222208	the number of image is incorrect	图片的数量错误
222209	face token not exist	face token不存在
222300	add face fail	人脸图片添加失败
222301	get face fail	获取人脸图片失败
222302	system error	服务端请求失败
222303	get face fail	获取人脸图片失败
223100	group is not exist	操作的用户组不存在
223101	group is already exist	该用户组已存在
223102	user is already exist	该用户已存在
223103	user is not exist	找不到该用户
223104	group_list is too large	group_list包含组数量过多
223105	face is already exist	该人脸已存在
223106	face is not exist	该人脸不存在
223110	uid_list is too large	uid_list包含数量过多
223111	dst group is not exist	目标用户组不存在
223112	quality_conf format error	quality_conf格式不正确
223113	face is covered	人脸有被遮挡
223114	face is fuzzy	人脸模糊
223115	face light is not good	人脸光照不好
223116	incomplete face	人脸不完整
223117	app_list is too large	app_list包含app数量过多
223118	quality control error	质量控制项错误
223119	liveness control item error	活体控制项错误
223120	liveness check fail	活体检测未通过
223121	left eye is occlusion	质量检测未通过 左眼遮挡程度过高
223122	right eye is occlusion	质量检测未通过 右眼遮挡程度过高
223123	left cheek is occlusion	质量检测未通过 左脸遮挡程度过高
223124	right cheek is occlusion	质量检测未通过 右脸遮挡程度过高
223125	chin contour is occlusion	质量检测未通过 下巴遮挡程度过高
223126	nose is occlusion	质量检测未通过 鼻子遮挡程度过高
223127	mouth is occlusion	质量检测未通过 嘴巴遮挡程度过高

222350	police picture is none or low quality	公安网图片不存在或 质量过低
222351	id number and name not match or id number not exist	身份证号与姓名不匹配或该 身份证号不存在
222352	name format error	身份证名字格式错误
222353	id number format error	身份证号码格式错误
222354	id number not exist	公安库里不存在此身份证号
222355	police picture not exist	身份证号码正确，公安库里没有 对应的照片
222360	invalid name or id number	身份证号码或名字非法（公安网校 验不通过）
222901	system busy	系统繁忙
222902	system busy	系统繁忙
222903	system busy	系统繁忙
222904	system busy	系统繁忙
222905	system busy	系统繁忙
222906	system busy	系统繁忙
222907	system busy	系统繁忙
222908	system busy	系统繁忙
222909	system busy	系统繁忙
222910	system busy	系统繁忙
222911	system busy	系统繁忙
222912	system busy	系统繁忙
222913	system busy	系统繁忙
222914	system busy	系统繁忙
222915	system busy	系统繁忙
222916	system busy	系统繁忙
222361	system busy	系统繁忙

C++-SDK

简介

Hi，您好，欢迎使用百度人脸识别服务。

本文档主要针对C++开发者，描述百度人脸识别接口服务的相关技术内容。如果您对文档内容有任何疑问，可以通过以下几种方式联系我们：

- 在百度云控制台内提交工单，咨询问题类型请选择**人工智能服务**；
- 如有疑问，进入AI社区交流：http://ai.baidu.com/forum/topic/list/165

接口能力

接口名称	接口能力简要描述
人脸检测	检测人脸并定位，返回五官关键点，及人脸各属性值
人脸比对	返回两两比对的人脸相似值
人脸查找	在一个人脸集合中找到找到相似的人脸，由一系列接口组成，包括人脸识别、人脸认证、人脸库管理相关接口（人脸注册、人脸更新、人脸删除、用户信息查询、组列表查询、组内用户列表查询、组间复制用户、组内删除用户）

版本更新记录

上线日期	版本号	更新内容
2019.4.17	0.8.1	人脸v3文档更新，新增N:M接口
2018.5.10	0.7.1	修复人脸V3接口问题
2018.4.28	0.7.0	全面切换为人脸V3接口
2018.4.9	0.6.0	新增身份验证，在线活体检测接口
2018.1.12	0.5.0	新增M：N多人脸识别接口
2017.12.21	0.4.0	更改了人脸认证,更新用户,人脸更新,组件复制用户的接口参数
2017.11.14	0.3.1	人脸检测接口升级到v2
2017.11.9	0.3.0	初始化参数修改
2017.10.31	0.1.0	人脸识别第一版

快速入门

安装人脸识别 C++ SDK

人脸识别 C++ SDK目录结构

```
├──── base
│   ├──── base.h                    // 请求客户端基类
│   ├──── base64.h                  // base64加密相关类
│   ├──── http.h                    // http请求封装类
│   └──── utils.h                   // 工具类
└──── face.h                  // 人脸识别 交互类
```

最低支持 C++ 11+

直接使用开发包步骤如下：

1.在官方网站下载C++ SDK压缩包。

2.将下载的`aip-cpp-sdk-version.zip`解压, 其中文件为包含实现代码的头文件。

3.安装依赖库libcurl（需要支持https） openssl jsoncpp(>1.6.2版本，0.x版本将不被支持)。

4.编译工程时添加 C++11 支持 (gcc/clang 添加编译参数 -std=c++11), 添加第三方库链接参数 lcurl, lcrypto, ljsoncpp。

5.在源码中include face.h ，引入压缩包中的头文件以使用aip命名空间下的类和方法。

新建client

client是人脸识别的C++客户端，为使用人脸识别的开发人员提供了一系列的交互方法。当您引入了相应头文件后就可以新建一个client对象

用户可以参考如下代码新建一个client：

```
 #include "face.h"

// 设置APPID/AK/SK
std::string app_id = "你的 App ID";
std::string api_key = "你的 Api key";
std::string secret_key = "你的 Secret Key";

aip::Face client(app_id, api_key, secret_key);
```

在上面代码中，常量APP_ID在百度云控制台中创建，常量API_KEY与SECRET_KEY是在创建完毕应用后，系统分配给用户的，均为字符串，用于标识用户，为访问做签名验证，可在AI服务控制台中的**应用列表**中查看。

注意：如您以前是百度云的老用户，其中API_KEY对应百度云的“Access Key ID”，SECRET_KEY对应百度云的“Access Key Secret”。

接口说明

人脸检测

人脸检测：检测图片中的人脸并标记出位置信息;

```
 Json::Value result;

std::string image = "取决于image_type参数，传入BASE64字符串或URL字符串或FACE_TOKEN字符串";

std::string image_type = "BASE64";

// 调用人脸检测
result = client.detect(image, image_type, aip::json_null);

// 如果有可选参数
std::map<std::string, std::string> options;
options["face_field"] = "age";
options["max_face_num"] = "2";
options["face_type"] = "LIVE";
options["liveness_control"] = "LOW";

// 带参数调用人脸检测
result = client.detect(image, image_type, options);
```

人脸检测 请求参数详情

参数名称	是否必选	类型	默认值	说明
image	是	std::string		图片信息(**总数据大小应小于10M**)，图片上传方式根据image_type来判断
image_type	是	std::string		图片类型 **BASE64**:图片的base64值，base64编码后的图片数据，编码后的图片大小不超过2M；**URL**:图片的 URL地址(可能由于网络等原因导致下载图片时间过长)； **FACE_TOKEN**: 人脸图片的唯一标识，调用人脸检测接口时，会为每个人脸图片赋予一个唯一的FACE_TOKEN，同一张图片多次检测得到的FACE_TOKEN是同一个
face_field	否	std::string		包括**age,expression,face_shape,gender,glasses,landmark,landmark150,quality,eye_status,emotion,face_type信息** 逗号分隔. 默认只返回face_token、人脸框、概率和旋转角度
max_face_num	否	std::string	1	最多处理人脸的数目，默认值为1，仅检测图片中面积最大的那个人脸；**最大值10**，检测图片中面积最大的几张人脸。
face_type	否	std::string		人脸的类型 **LIVE**表示生活照：通常为手机、相机拍摄的人像图片、或从网络获取的人像图片等**IDCARD**表示身份证芯片照：二代身份证内置芯片中的人像照片 **WATERMARK**表示带水印证件照：一般为带水印的小图，如公安网小图 **CERT**表示证件照片：如拍摄的身份证、工卡、护照、学生证等证件图片 默认**LIVE**
liveness_control	否	std::string	NONE	活体检测控制 **NONE**: 不进行控制 **LOW**:较低的活体要求(高通过率 低攻击拒绝率) **NORMAL**: 一般的活体要求(平衡的攻击拒绝率, 通过率) **HIGH**: 较高的活体要求(高攻击拒绝率 低通过率) **默认NONE**

人脸检测 返回数据参数详情

字段	必选	类型	说明
face_num	是	int	检测到的图片中的人脸数量
face_list	是	array	人脸信息列表，具体包含的参数参考下面的列表。
+face_token	是	string	人脸图片的唯一标识
+location	是	array	人脸在图片中的位置
++left	是	double	人脸区域离左边界的距离
++top	是	double	人脸区域离上边界的距离
++width	是	double	人脸区域的宽度
++height	是	double	人脸区域的高度
++rotation	是	int64	人脸框相对于竖直方向的顺时针旋转角，[-180,180]
+face_probability	是	double	人脸置信度，范围【0~1】，代表这是一张人脸的概率，0最小、1最大。
+angel	是	array	人脸旋转角度参数
++yaw	是	double	三维旋转之左右旋转角[-90(左), 90(右)]
++pitch	是	double	三维旋转之俯仰角度[-90(上), 90(下)]
++roll	是	double	平面内旋转角[-180(逆时针), 180(顺时针)]
+age	否	double	年龄 ，当**face_field包含age时返回**
+expression	否	array	表情，当 **face_field包含expression时返回**
++type	否	string	**none**:不笑；**smile**:微笑；**laugh**:大笑
++probability	否	double	表情置信度，范围【0~1】，0最小、1最大。
+face_shape	否	array	脸型，当**face_field包含face_shape时返回**

++type	否	double	**square**: 正方形 **triangle**:三角形 **oval**: 椭圆 **heart**: 心形 **round**: 圆形
++probability	否	double	置信度，范围【0~1】，代表这是人脸形状判断正确的概率，0最小、1最大。
+gender	否	array	性别，**face_field包含gender时返回**
++type	否	string	male:**男性** female:**女性**
++probability	否	double	性别置信度，范围【0~1】，0代表概率最小、1代表最大。
+glasses	否	array	是否带眼镜，**face_field包含glasses时返回**
++type	否	string	**none**:无眼镜，**common**:普通眼镜，**sun**:墨镜
++probability	否	double	眼镜置信度，范围【0~1】，0代表概率最小、1代表最大。
+eye_status	否	array	双眼状态（睁开/闭合） **face_field包含eye_status时返回**
++left_eye	否	double	左眼状态 [0,1]取值，越接近0闭合的可能性越大
++right_eye	否	double	右眼状态 [0,1]取值，越接近0闭合的可能性越大
+emotion	否	array	情绪 **face_field包含emotion时返回**
++type	否	string	**angry**:愤怒 **disgust**:厌恶 **fear**:恐惧 **happy**:高兴 **sad**:伤心 **surprise**:惊讶 **neutral**:无情绪
++probability	否	double	情绪置信度，范围0~1
++probability	否	double	人种置信度，范围【0~1】，0代表概率最小、1代表最大。
+face_type	否	array	真实人脸/卡通人脸 **face_field包含face_type时返回**
++type	否	string	**human**: 真实人脸 **cartoon**: 卡通人脸
++probability	否	double	人脸类型判断正确的置信度，范围【0~1】，0代表概率最小、1代表最大。
+landmark	否	array	4个关键点位置，左眼中心、右眼中心、鼻尖、嘴中心。**face_field包含landmark时返回**
+landmark72	否	array	72个特征点位置 **face_field包含landmark时返回**
+landmark150	否	array	150个特征点位置 **face_field包含landmark150时返回**
+quality	否	array	人脸质量信息。**face_field包含quality时返回**
++occlusion	否	array	人脸各部分遮挡的概率，范围[0~1]，0表示完整，1表示不完整
+++left_eye	否	double	左眼遮挡比例，[0-1] ，1表示完全遮挡
+++right_eye	否	double	右眼遮挡比例，[0-1] ， 1表示完全遮挡
+++nose	否	double	鼻子遮挡比例，[0-1] ， 1表示完全遮挡
+++mouth	否	double	嘴巴遮挡比例，[0-1] ， 1表示完全遮挡
+++left_cheek	否	double	左脸颊遮挡比例，[0-1] ， 1表示完全遮挡
+++right_cheek	否	double	右脸颊遮挡比例，[0-1] ， 1表示完全遮挡
+++chin_contour	否	double	下巴遮挡比例，，[0-1] ， 1表示完全遮挡
++blur	否	double	人脸模糊程度，范围[0~1]，0表示清晰，1表示模糊
++illumination	否	double	取值范围在[0~255], 表示脸部区域的光照程度 越大表示光照越好
++completeness	否	int64	人脸完整度，0或1, 0为人脸溢出图像边界，1为人脸都在图像边界内

人脸检测 返回示例


```
{
"face_num": 1,
"face_list": [
    {
        "face_token": "35235asfas21421fakghktyfdgh68bio",
        "location": {
```

```
            "left": 117,
            "top": 131,
            "width": 172,
            "height": 170,
            "rotation": 4
        },
        "face_probability": 1,
        "angle" :{
            "yaw" : -0.34859421849251
            "pitch" 1.9135693311691
            "roll" :2.3033397197723
        }
        "landmark": [
            {
                "x": 161.74819946289,
                "y": 163.30244445801
            },
            ...
        ],
        "landmark72": [
            {
                "x": 115.86531066895,
                "y": 170.0546875
            } ,
            ...
        ],
        "age": 29.298097610474,
        "expression": {
            "type": "smile",
            "probability" : 0.5543018579483
        },
        "gender": {
            "type": "male",
            "probability": 0.99979132413864
        },
        "glasses": {
       "type": "sun",
            "probability": 0.99999964237213
        },
        "face_shape": {
            "type": "triangle",
            "probability": 0.5543018579483
        }
        "quality": {
            "occlusion": {
                "left_eye": 0,
                "right_eye": 0,
                "nose": 0,
                "mouth": 0,
                "left_cheek": 0.0064102564938366,
                "right_cheek": 0.0057411273010075,
                "chin": 0
            },
            "blur": 1.1886881756684e-10,
            "illumination": 141,
            "completeness": 1
        }
    }
  ]
}
```

**72个关键点分布图（对应landmark72个点的顺序，序号从0-71）：https://ai.bdstatic.com/file/52BC00FFD4754A6298D977EDAD033DA0

人脸搜索

- **1：N人脸搜索**：也称为1：N识别，在指定人脸集合中，找到最相似的人脸；
- **1：N人脸认证**：基于uid维度的1：N识别，由于uid已经锁定固定数量的人脸，所以检索范围更聚焦；

> **1：N人脸识别**与**1：N人脸认证**的差别在于：人脸搜索是在指定人脸集合中进行直接地人脸检索操作，而人脸认证是基于uid，先调取这个uid对应的人脸，再在这个uid对应的人脸集合中进行检索（因为每个uid通常对应的只有一张人脸，所以通常也就变为了1：1对比）；实际应用中，人脸认证需要用户或系统先输入id，这增加了验证安全度，但也增加了复杂度，具体使用哪个接口需要视您的业务场景判断。

```
 Json::Value result;

std::string image = "取决于image_type参数，传入BASE64字符串或URL字符串或FACE_TOKEN字符串";

std::string image_type = "BASE64";

std::string group_id_list = "3,2";

// 调用人脸搜索
result = client.face_search_v3(image, image_type, group_id_list, aip::json_null);

// 如果有可选参数
std::map<std::string, std::string> options;
options["match_threshold"] = "70";
options["quality_control"] = "NORMAL";
options["liveness_control"] = "LOW";
options["user_id"] = "233451";
options["max_user_num"] = "3";

// 带参数调用人脸搜索
result = client.face_search_v3(image, image_type, group_id_list, options);
```

人脸搜索 请求参数详情

参数名称	是否必选	类型	默认值	说明
image	是	std::string		图片信息(**总数据大小应小于10M**)，图片上传方式根据image_type来判断
image_type	是	std::string		图片类型 **BASE64**:图片的base64值，base64编码后的图片数据，编码后的图片大小不超过2M；**URL**:图片的 URL地址(可能由于网络等原因导致下载图片时间过长)； **FACE_TOKEN**: 人脸图片的唯一标识，调用人脸检测接口时，会为每个人脸图片赋予一个唯一的FACE_TOKEN，同一张图片多次检测得到的FACE_TOKEN是同一个
group_id_list	是	std::string		从指定的group中进行查找 用逗号分隔，**上限10个**
match_threshold	否	std::string		匹配阈值（设置阈值后，score低于此阈值的用户信息将不会返回） 最大100 最小0 默认80 **此阈值设置得越高，检索速度将会越快，推荐使用默认阈值80**
quality_control	否	std::string	NONE	图片质量控制 **NONE**: 不进行控制 **LOW**:较低的质量要求 **NORMAL**: 一般的质量要求 **HIGH**: 较高的质量要求 **默认 NONE**
liveness_control	否	std::string	NONE	活体检测控制 **NONE**: 不进行控制 **LOW**:较低的活体要求(高通过率 低攻击拒绝率) **NORMAL**: 一般的活体要求(平衡的攻击拒绝率, 通过率) **HIGH**: 较高的活体要求(高攻击拒绝率 低通过率) **默认NONE**
user_id	否	std::string		当需要对特定用户进行比对时，指定user_id进行比对。即人脸认证功能。
max_user_num	否	std::string		查找后返回的用户数量。返回相似度最高的几个用户，默认为1，最多返回50个。

人脸搜索 返回数据参数详情

字段	必选	类型	说明
face_token	是	string	人脸标志
user_list	是	array	匹配的用户信息列表
+group_id	是	string	用户所属的group_id
+user_id	是	string	用户的user_id
+user_info	是	string	注册用户时携带的user_info
+score	是	float	用户的匹配得分

人脸搜索 返回示例

```
{
 "face_token": "fid",
 "user_list": [
   {
     "group_id" : "test1",
     "user_id": "u333333",
     "user_info": "Test User",
     "score": 99.3
   }
 ]
}
```

人脸搜索 M:N 识别

待识别的图片中，存在多张人脸的情况下，支持在一个人脸库中，一次请求，同时返回图片中所有人脸的识别结果。

```
 Json::Value result;

std::string image = "取决于image_type参数，传入BASE64字符串或URL字符串或FACE_TOKEN字符串";

std::string image_type = "BASE64";

std::string group_id_list = "3,2";

// 调用人脸搜索 M:N 识别
result = client.multi_search(image, image_type, group_id_list, aip::json_null);

// 如果有可选参数
std::map<std::string, std::string> options;
options["max_face_num"] = "3";
options["match_threshold"] = "70";
options["quality_control"] = "NORMAL";
options["liveness_control"] = "LOW";
options["max_user_num"] = "3";

// 带参数调用人脸搜索 M:N 识别
result = client.multi_search(image, image_type, group_id_list, options);
```

人脸搜索 M:N 识别 请求参数详情

参数名称	是否必选	类型	默认值	说明
image	是	std::string		图片信息(**总数据大小应小于10M**)，图片上传方式根据image_type来判断
image_type	是	std::string		图片类型 **BASE64**:图片的base64值，base64编码后的图片数据，编码后的图片大小不超过2M；**URL**:图片的 URL地址(可能由于网络等原因导致下载图片时间过长)； **FACE_TOKEN**: 人脸图片的唯一标识，调用人脸检测接口时，会为每个人脸图片赋予一个唯一的FACE_TOKEN，同一张图片多次检测得到的FACE_TOKEN是同一个
group_id_list	是	std::string		从指定的group中进行查找 用逗号分隔，**上限10个**
max_face_num	否	std::string		最多处理人脸的数目 **默认值为1(仅检测图片中面积最大的那个人脸) 最大值10**
match_threshold	否	std::string		匹配阈值（设置阈值后，score低于此阈值的用户信息将不会返回） 最大100 最小0 默认80 **此阈值设置得越高，检索速度将会越快，推荐使用默认阈值80**
quality_control	否	std::string	NONE	图片质量控制 **NONE**: 不进行控制 **LOW**:较低的质量要求 **NORMAL**: 一般的质量要求 **HIGH**: 较高的质量要求 **默认 NONE**
liveness_control	否	std::string	NONE	活体检测控制 **NONE**: 不进行控制 **LOW**:较低的活体要求(高通过率 低攻击拒绝率) **NORMAL**: 一般的活体要求(平衡的攻击拒绝率, 通过率) **HIGH**: 较高的活体要求(高攻击拒绝率 低通过率) **默认NONE**
max_user_num	否	std::string		查找后返回的用户数量。返回相似度最高的几个用户，默认为1，最多返回20个。

人脸搜索 M:N 识别 返回数据参数详情

字段	必选	类型	说明
face_num	是	int	图片中的人脸数量
face_list	是	array	人脸信息列表
+face_token	是	string	人脸标志
+location	是	array	人脸在图片中的位置
++left	是	double	人脸区域离左边界的距离
++top	是	double	人脸区域离上边界的距离
++width	是	double	人脸区域的宽度
++height	是	double	人脸区域的高度
++rotation	是	int64	人脸框相对于竖直方向的顺时针旋转角，[-180,180]
+user_list	是	array	匹配的用户信息列表
++group_id	是	string	用户所属的group_id
++user_id	是	string	用户的user_id
++user_info	是	string	注册用户时携带的user_info
++score	是	float	用户的匹配得分 **80分以上可以判断为同一人，此分值对应万分之一误识率**

人脸搜索 M:N 识别 返回示例

```
{
  "error_code": 0,
  "error_msg": "SUCCESS",
  "log_id": 240483475,
  "timestamp": 1535533440,
  "cached": 0,
  "result": {
    "face_num": 2,
    "face_list": [
      {
        "face_token": "6fe19a6ee0c4233db9b5bba4dc2b9233",
        "location": {
          "left": 31.95568085,
          "top": 120.3764267,
          "width": 87,
          "height": 85,
          "rotation": -5
        },
        "user_list": [
          {
            "group_id": "group1",
            "user_id": "5abd24fd062e49bfa906b257ec40d284",
            "user_info": "userinfo1",
            "score": 69.85684967041
          },
          {
            "group_id": "group1",
            "user_id": "2abf89cffb31473a9948268fde9e1c3f",
            "user_info": "userinfo2",
            "score": 66.586112976074
          }
        ]
      },
      {
        "face_token": "fde61e9c074f48cf2bbb319e42634f41",
        "location": {
          "left": 219.4467773,
          "top": 104.7486954,
          "width": 81,
          "height": 77,
          "rotation": 3
        },
        "user_list": [
          {
            "group_id": "group1",
            "user_id": "088717532b094c3990755e91250adf7d",
            "user_info": "userinfo",
            "score": 65.154159545898
          }
        ]
      }
    ]
  }
}
```


人脸注册

用于从人脸库中新增用户，可以设定多个用户所在组，及组内用户的人脸图片，

典型应用场景：构建您的人脸库，如**会员人脸注册**，**已有用户补全人脸信息**等。

人脸库、用户组、用户、用户下的人脸**层级关系**如下所示：

```
|- 人脸库
 |- 用户组一
   |- 用户01
     |- 人脸
   |- 用户02
     |- 人脸
     |- 人脸
     ....
    ....
 |- 用户组二
 |- 用户组三
 |- 用户组四
 ....
```

关于人脸库的设置限制

- **每个appid对应一个人脸库，且不同appid之间，人脸库互不相通**；
- 每个人脸库下，可以创建多个用户组，用户组（group）数量**没有限制**；
- 每个用户组（group）下，可添加最多**无限**张人脸，**无限**个uid；
- 每个用户（uid）所能注册的最大人脸数量**20个**；

为了保证识别效果，请控制注册人脸的质量（通过`/detect`人脸检测接口判断），具体参数可详见下表所示：

质量判断

可通过人脸检测接口，基于以下字段和对应阈值，进行质量检测的判断，以保证人脸质量符合后续业务操作要求。

指标	字段与解释	推荐数值界限
遮挡范围	**occlusion**（0~1），0为无遮挡，1是完全遮挡 含有多个具体子字段，表示脸部多个部位 通常用作判断头发、墨镜、口罩等遮挡	left_eye：0.6, #左眼被遮挡的阈值 right_eye：0.6, #右眼被遮挡的阈值 nose：0.7, #鼻子被遮挡的阈值 mouth：0.7, #嘴巴被遮挡的阈值 left_check：0.8, #左脸颊被遮挡的阈值 right_check：0.8, #右脸颊被遮挡的阈值 chin_contour：0.6, #下巴被遮挡阈值
模糊度范围	**Blur**（0~1），0是最清晰，1是最模糊	小于0.7
光照范围	**illumination**（0~255） 脸部光照的灰度值，0表示光照不好 以及对应客户端SDK中，YUV的Y分量	大于40
姿态角度	**Pitch**：三维旋转之俯仰角度[-90(上), 90(下)] **Roll**：平面内旋转角[-180(逆时针), 180(顺时针)] **Yaw**：三维旋转之左右旋转角[-90(左), 90(右)]	分别小于20度
人脸完整度	**completeness**（0或1），0为人脸溢出图像边界，1为人脸都在图像边界内	视业务逻辑判断
人脸大小	人脸部分的大小 建议长宽像素值范围：80*80~200*200	人脸部分不小于**100*100**像素

```
 Json::Value result;

std::string image = "取决于image_type参数，传入BASE64字符串或URL字符串或FACE_TOKEN字符串";

std::string image_type = "BASE64";

std::string group_id = "group1";

std::string user_id = "user1";

// 调用人脸注册
result = client.user_add(image, image_type, group_id, user_id, aip::json_null);

// 如果有可选参数
std::map<std::string, std::string> options;
options["user_info"] = "user's info";
options["quality_control"] = "NORMAL";
options["liveness_control"] = "LOW";
options["action_type"] = "REPLACE";

// 带参数调用人脸注册
result = client.user_add(image, image_type, group_id, user_id, options);
```

人脸注册 请求参数详情

参数名称	是否必选	类型	默认值	说明
image	是	std::string		图片信息(总数据大小应小于10M)，图片上传方式根据image_type来判断。注：组内每个uid下的人脸图片数目上限为20张
image_type	是	std::string		图片类型 **BASE64**:图片的base64值，base64编码后的图片数据，编码后的图片大小不超过2M；**URL**:图片的 URL地址(可能由于网络等原因导致下载图片时间过长)；**FACE_TOKEN**: 人脸图片的唯一标识，调用人脸检测接口时，会为每个人脸图片赋予一个唯一的FACE_TOKEN，同一张图片多次检测得到的FACE_TOKEN是同一个
group_id	是	std::string		用户组id（由数字、字母、下划线组成），长度限制128B
user_id	是	std::string		用户id（由数字、字母、下划线组成），长度限制128B
user_info	否	std::string		用户资料，长度限制256B
quality_control	否	std::string	NONE	图片质量控制 **NONE**: 不进行控制 **LOW**:较低的质量要求 **NORMAL**: 一般的质量要求 **HIGH**: 较高的质量要求 **默认 NONE**
liveness_control	否	std::string	NONE	活体检测控制 **NONE**: 不进行控制 **LOW**:较低的活体要求(高通过率 低攻击拒绝率) **NORMAL**: 一般的活体要求(平衡的攻击拒绝率, 通过率) **HIGH**: 较高的活体要求(高攻击拒绝率 低通过率) **默认NONE**
action_type	否	std::string	APPEND	操作方式 APPEND: 当user_id在库中已经存在时，对此user_id重复注册时，新注册的图片默认会追加到该user_id下,REPLACE：当对此user_id重复注册时,则会用新图替换库中该user_id下所有图片,默认使用APPEND

人脸注册 返回数据参数详情

字段	必选	类型	说明
log_id	是	uint64	请求标识码，随机数，唯一
face_token	是	string	人脸图片的唯一标识
location	是	array	人脸在图片中的位置
+left	是	double	人脸区域离左边界的距离
+top	是	double	人脸区域离上边界的距离
+width	是	double	人脸区域的宽度
+height	是	double	人脸区域的高度
+rotation	是	int64	人脸框相对于竖直方向的顺时针旋转角，[-180,180]

人脸注册 返回示例

```
{
 "face_token": "2fa64a88a9d5118916f9a303782a97d3",
 "location": {
    "left": 117,
    "top": 131,
    "width": 172,
    "height": 170,
    "rotation": 4
 }
}
```

人脸更新

用于对人脸库中指定用户，更新其下的人脸图像。

> **说明：**针对一个uid执行更新操作，新上传的人脸图像将覆盖此uid原有所有图像。

```
 Json::Value result;

std::string image = "取决于image_type参数，传入BASE64字符串或URL字符串或FACE_TOKEN字符串";

std::string image_type = "BASE64";

std::string group_id = "group1";

std::string user_id = "user1";

// 调用人脸更新
result = client.user_update(image, image_type, group_id, user_id, aip::json_null);

// 如果有可选参数
std::map<std::string, std::string> options;
options["user_info"] = "user's info";
options["quality_control"] = "NORMAL";
options["liveness_control"] = "LOW";
options["action_type"] = "REPLACE";

// 带参数调用人脸更新
result = client.user_update(image, image_type, group_id, user_id, options);
```

人脸更新 请求参数详情

参数名称	是否必选	类型	默认值	说明
image	是	std::string		图片信息(**总数据大小应小于10M**)，图片上传方式根据image_type来判断
image_type	是	std::string		图片类型 **BASE64**:图片的base64值，base64编码后的图片数据，编码后的图片大小不超过2M；**URL**:图片的 URL地址(可能由于网络等原因导致下载图片时间过长)；**FACE_TOKEN**: 人脸图片的唯一标识，调用人脸检测接口时，会为每个人脸图片赋予一个唯一的FACE_TOKEN，同一张图片多次检测得到的FACE_TOKEN是同一个
group_id	是	std::string		更新指定groupid下uid对应的信息
user_id	是	std::string		用户id（由数字、字母、下划线组成），长度限制128B
user_info	否	std::string		用户资料，长度限制256B
quality_control	否	std::string	NONE	图片质量控制 **NONE**: 不进行控制 **LOW**:较低的质量要求 **NORMAL**: 一般的质量要求 **HIGH**: 较高的质量要求 **默认 NONE**
liveness_control	否	std::string	NONE	活体检测控制 **NONE**: 不进行控制 **LOW**:较低的活体要求(高通过率 低攻击拒绝率) **NORMAL**: 一般的活体要求(平衡的攻击拒绝率, 通过率) **HIGH**: 较高的活体要求(高攻击拒绝率 低通过率) **默认NONE**
action_type	否	std::string	UPDATE	操作方式 **UPDATE**: 当user_id在库中已经存在时，对此user_id重复注册时，新注册的图片默认会追加到该user_id下,**REPLACE**：当对此user_id重复注册时,则会用新图替换库中该user_id下所有图片,默认使用**UPDATE**

人脸更新 返回数据参数详情

字段	必选	类型	说明
log_id	是	uint64	请求标识码，随机数，唯一
face_token	是	string	人脸图片的唯一标识
location	是	array	人脸在图片中的位置
+left	是	double	人脸区域离左边界的距离
+top	是	double	人脸区域离上边界的距离
+width	是	double	人脸区域的宽度
+height	是	double	人脸区域的高度
+rotation	是	int64	人脸框相对于竖直方向的顺时针旋转角，[-180,180]

人脸更新 返回示例

```
{
 "face_token": "2fa64a88a9d5118916f9a303782a97d3",
 "location": {
    "left": 117,
    "top": 131,
    "width": 172,
    "height": 170,
    "rotation": 4
 }
}
```

人脸删除

用于从人脸库中删除一个用户。

人脸删除注意事项：

- 删除的内容，包括用户所有图像和身份信息；
- 如果一个uid存在于多个用户组内，将会同时将从各个组中把用户删除
- 如果指定了group_id，则只删除此group下的uid相关信息

```
 Json::Value result;

std::string user_id = "user1";

std::string group_id = "group1";

std::string face_token = "face_token_23123";

// 调用人脸删除
result = client.face_delete(user_id, group_id, face_token, aip::json_null);
```

人脸删除 请求参数详情

参数名称	是否必选	类型	说明
log_id	是	uint64	请求标识码，随机数，唯一
user_id	是	std::string	用户id（由数字、字母、下划线组成），长度限制128B
group_id	是	std::string	用户组id（由数字、字母、下划线组成），长度限制128B
face_token	是	std::string	需要删除的人脸图片token，（由数字、字母、下划线组成）长度限制64B

人脸删除 返回数据参数详情

字段	是否必选	类型	说明
log_id	是	uint64	请求唯一标识码，随机数

人脸删除 返回示例

```
 // 删除成功
{
  "error_code": 0,
  "log_id": 73473737,
}
// 删除发生错误
{
  "error_code": 223106,
  "log_id": 1382953199,
  "error_msg": "face is not exist"
}
```

用户信息查询

获取人脸库中某个用户的信息(user_info信息和用户所属的组)。

```
 Json::Value result;

std::string user_id = "user1";

std::string group_id = "group1";

// 调用用户信息查询
result = client.user_get(user_id, group_id, aip::json_null);
```

用户信息查询 请求参数详情

参数名称	是否必选	类型	说明
user_id	是	std::string	用户id（由数字、字母、下划线组成），长度限制128B
group_id	是	std::string	用户组id（由数字、字母、下划线组成），长度限制128B

用户信息查询 返回数据参数详情

字段	必选	类型	说明
log_id	是	uint64	请求标识码，随机数，唯一
user_list	是	array	查询到的用户列表
+user_info	是	string	用户资料，被查询用户的资料
+group_id	是	string	用户组id，被查询用户的所在组

用户信息查询 返回示例

```
{
 "user_list": [
    {
        "user_info": "user info ...",
        "group_id": "gid1"
    },
    {
        "user_info": "user info2 ...",
        "group_id": "gid2"
    }
 ]
}
```

获取用户人脸列表

用于获取一个用户的全部人脸列表。

```
 Json::Value result;

std::string user_id = "user1";

std::string group_id = "group1";

// 调用获取用户人脸列表
result = client.face_getlist(user_id, group_id, aip::json_null);
```

获取用户人脸列表 请求参数详情

参数名称	是否必选	类型	说明
user_id	是	std::string	用户id（由数字、字母、下划线组成），长度限制128B
group_id	是	std::string	用户组id（由数字、字母、下划线组成），长度限制128B

获取用户人脸列表 返回数据参数详情

字段	必选	类型	说明
log_id	是	uint64	请求标识码，随机数，唯一
face_list	是	array	人脸列表
+face_token	是	string	人脸图片的唯一标识
+ctime	是	string	人脸创建时间

获取用户人脸列表 返回示例

```
{
 "face_list": [
    {
        "face_token": "fid1",
        "ctime": "2018-01-01 00:00:00"
    },
    {
        "face_token": "fid2",
        "ctime": "2018-01-01 10:00:00"
    }
 ]
}
```

获取用户列表

用于查询指定用户组中的用户列表。

```
 Json::Value result;

std::string group_id = "group1";

// 调用获取用户列表
result = client.group_getusers(group_id, aip::json_null);

// 如果有可选参数
std::map<std::string, std::string> options;
options["start"] = "0";
options["length"] = "50";

// 带参数调用获取用户列表
result = client.group_getusers(group_id, options, aip::json_null);
```

获取用户列表 请求参数详情

参数名称	是否必选	类型	默认值	说明
group_id	是	std::string		用户组id（由数字、字母、下划线组成），长度限制128B
start	否	std::string	0	默认值0，起始序号
length	否	std::string	100	返回数量，默认值100，最大值1000

获取用户列表 返回数据参数详情

字段	必选	类型	说明
user_id_list	是	array	用户ID列表

获取用户列表 返回示例

```
{
 "user_id_list": [
    "uid1",
    "uid2"
 ]
}
```

复制用户

用于将已经存在于人脸库中的用户**复制到一个新的组**。

```
 Json::Value result;

std::string user_id = "user1";

// 调用复制用户
result = client.user_copy(user_id, aip::json_null);

// 如果有可选参数
std::map<std::string, std::string> options;
options["src_group_id"] = "11111";
options["dst_group_id"] = "222222";

// 带参数调用复制用户
result = client.user_copy(user_id, options, aip::json_null);
```

复制用户 请求参数详情

参数名称	是否必选	类型	说明
user_id	是	std::string	用户id（由数字、字母、下划线组成），长度限制128B
src_group_id	否	std::string	从指定组里复制信息
dst_group_id	否	std::string	需要添加用户的组id

复制用户 返回数据参数详情

字段	必选	类型	说明
log_id	是	id	log_id

复制用户 返回示例

```
 // 正确返回值
{
  "error_code": 0,
 "log_id": 3314921889,
}
// 发生错误时返回值
{
 "error_code": 216100,
 "log_id": 3111284097,
 "error_msg": "already add"
}
```

删除用户

用于将用户从某个组中删除。

```
 Json::Value result;

std::string group_id = "group1";

std::string user_id = "user1";

// 调用删除用户
result = client.user_delete(group_id, user_id, aip::json_null);
```

删除用户 请求参数详情

参数名称	是否必选	类型	说明
group_id	是	std::string	用户组id（由数字、字母、下划线组成），长度限制128B
user_id	是	std::string	用户id（由数字、字母、下划线组成），长度限制128B

删除用户 返回数据参数详情

字段	是否必选	类型	说明
log_id	是	uint64	请求唯一标识码，随机数

删除用户 返回示例

```
 // 正确返回值
{

 "error_code": 0,
 "log_id": 3314921889,
}
// 发生错误时返回值
{
 "error_code":  223103,
 "log_id": 815967402,
 "error_msg": "user is not exist"
}
```

创建用户组

用于创建一个空的用户组，如果用户组已存在 则返回错误。

```
 Json::Value result;

std::string group_id = "group1";

// 调用创建用户组
result = client.group_add(group_id, aip::json_null);
```

创建用户组 请求参数详情

参数名称	是否必选	类型	说明
group_id	是	std::string	用户组id（由数字、字母、下划线组成），长度限制128B

创建用户组 返回数据参数详情

字段	必选	类型	说明
log_id	是	uint64	请求标识码，随机数，唯一

创建用户组 返回示例

```
 {

  "error_code": 0,
  "log_id": 3314921889,
}
// 发生错误时返回值
{
  "error_code":  223101,
  "log_id": 815967402,
  "error_msg": " group is already exist"
}
```

删除用户组

删除用户组下所有的用户及人脸，如果组不存在 则返回错误。

```
 Json::Value result;

std::string group_id = "group1";

// 调用删除用户组
result = client.group_delete(group_id, aip::json_null);
```

删除用户组 请求参数详情

参数名称	是否必选	类型	说明
group_id	是	std::string	用户组id（由数字、字母、下划线组成），长度限制128B

删除用户组 返回数据参数详情

字段	必选	类型	说明
log_id	是	uint64	请求标识码，随机数，唯一

删除用户组 返回示例

```
 // 正确返回值
{

 "error_code":0,
 "log_id": 3314921889,
}
// 发生错误时返回值
{
 "error_code": 223100,
 "log_id": 815967402,
 "error_msg": " group is not exist"
}
```

组列表查询

用于查询用户组的列表。

```
 Json::Value result;

// 调用组列表查询
result = client.group_getlist(aip::json_null);

// 如果有可选参数
std::map<std::string, std::string> options;
options["start"] = "0";
options["length"] = "50";

// 带参数调用组列表查询
result = client.group_getlist(options, aip::json_null);
```

组列表查询 请求参数详情

参数名称	是否必选	类型	默认值	说明
start	否	std::string	0	默认值0，起始序号
length	否	std::string	100	返回数量，默认值100，最大值1000

组列表查询 返回数据参数详情

字段	必选	类型	说明
group_id_list	是	array	group

组列表查询 返回示例

```
 {
   "group_id_list": [
      "gid1",
      "gid2"
   ]
}
```

身份验证

质量检测（可选）活体检测（可选）公安验证（必选）

```
 Json::Value result;

std::string image = "取决于image_type参数，传入BASE64字符串或URL字符串或FACE_TOKEN字符串";

std::string image_type = "BASE64";

std::string id_card_number = "110233112299822211";

std::string name = "张三";

// 调用身份验证
result = client.person_verify(image, image_type, id_card_number, name, aip::json_null);

// 如果有可选参数
std::map<std::string, std::string> options;
options["quality_control"] = "NORMAL";
options["liveness_control"] = "LOW";

// 带参数调用身份验证
result = client.person_verify(image, image_type, id_card_number, name, options, aip::json_null);
```

身份验证 请求参数详情

参数名称	是否必选	类型	默认值	说明
image	是	std::string		图片信息(**总数据大小应小于10M**)，图片上传方式根据image_type来判断
image_type	是	std::string		图片类型 **BASE64**:图片的base64值，base64编码后的图片数据，编码后的图片大小不超过2M；**URL**:图片的 URL地址(可能由于网络等原因导致下载图片时间过长)；**FACE_TOKEN**: 人脸图片的唯一标识，调用人脸检测接口时，会为每个人脸图片赋予一个唯一的FACE_TOKEN，同一张图片多次检测得到的FACE_TOKEN是同一个
id_card_number	是	std::string		身份证号（真实身份证号号码）
name	是	std::string		utf8，姓名（真实姓名，和身份证号匹配）
quality_control	否	std::string	NONE	图片质量控制 **NONE**: 不进行控制 **LOW**:较低的质量要求 **NORMAL**: 一般的质量要求 **HIGH**: 较高的质量要求 **默认 NONE**
liveness_control	否	std::string	NONE	活体检测控制 **NONE**: 不进行控制 **LOW**:较低的活体要求(高通过率 低攻击拒绝率) **NORMAL**: 一般的活体要求(平衡的攻击拒绝率, 通过率) **HIGH**: 较高的活体要求(高攻击拒绝率 低通过率) **默认NONE**

身份验证 返回数据参数详情

参数	必须	类型	说明
log_id	是	uint64	日志id
score	是	float	与公安小图相似度可能性，用于验证生活照与公安小图是否为同一人，有正常分数时为[0~1]，推荐阈值0.8，超过即判断为同一人

身份验证 返回示例

```
{
 "score": 44.3,
}
```

语音校验码接口

此接口主要用于生成随机码，用于视频的语音识别校验使用，以判断视频的即时性，而非事先录制的，提升作弊的难度。

```
 Json::Value result;

// 调用语音校验码接口
result = client.video_sessioncode(aip::json_null);

// 如果有可选参数
std::map<std::string, std::string> options;
options["appid"] = "223245";

// 带参数调用语音校验码接口
result = client.video_sessioncode(options, aip::json_null);
```

语音校验码接口 请求参数详情

参数名称	是否必选	类型	说明
appid	否	std::string	百度云创建应用时的唯一标识ID

语音校验码接口 返回数据参数详情

字段	必选	类型	说明
session_id	是	string	语音校验码会话id
code	是	string	语音验证码，数字形式，3~6位数字

语音校验码接口 返回示例

```
{
 "err_no": 0,
 "err_msg": "SUCCESS",
 "result": {
   "session_id": "S59faeeebb9111890355690",
   "code": "9940"
 },
 "timestamp": 1509617387,
 "cached": 0,
 "serverlogid": "0587756642"
}
```

在线活体检测

接口能力

- **人脸基础信息**：包括人脸框位置，人脸空间旋转角度，人脸置信度等信息。
- **人脸质量检测**：判断人脸的遮挡、光照、模糊度、完整度等质量信息。可用于判断上传的人脸是否符合标准。
- **基于图片的活体检测**：基于单张图片，判断图片中的人脸是否为二次翻拍（举例：如用户A用手机拍摄了一张包含人脸的图片一，用户B翻拍了图片一得到了图片二，并用图片二伪造成用户A去进行识别操作，这种情况普遍发生在金融开户、实名认证等环节）。此能力可用于H5场景下的一些人脸采集场景中，增加人脸注册的安全性和真实性。

```
  std::string file_content;

Json::Value data;
Json::Value image_desp_1;
Json::Value image_desp_2;

aip::get_file_content("./assets/sample1.jpg", &file_content);
image_desp_1["image"] = aip::base64_encode(file_content.c_str(), (int) file_content.size());
image_desp_1["image_type"] = "BASE64";

aip::get_file_content("./assets/sample2.jpg", &file_content);
image_desp_2["image"] = aip::base64_encode(file_content.c_str(), (int) file_content.size());
image_desp_2["image_type"] = "BASE64";

data[0] = image_desp_1;
data[1] = image_desp_2;

Json::Value result = client->faceverify(data);
```

请求参数

参数	是否必选	类型	说明
image	是	string	图片信息(总数据大小应小于10M)，图片上传方式根据image_type来判断； **可以上传同一个用户的1张、3张或8张图片来进行活体判断，注：后端会选择每组照片中的最高分数作为整体分数。**
image_type	是	string	图片类型 **BASE64**:图片的base64值，base64编码后的图片数据，编码后的图片大小不超过2M； **URL**:图片的 URL地址(可能由于网络等原因导致下载图片时间过长)； **FACE_TOKEN**: 人脸图片的唯一标识，调用人脸检测接口时，会为每个人脸图片赋予一个唯一的FACE_TOKEN，同一张图片多次检测得到的FACE_TOKEN是同一个。
face_field	否	string	包括**age,expression,faceshape,gender,glasses,landmark,quality,facetype信息**，逗号分隔，默认只返回face_token、活体数、人脸框、概率和旋转角度

返回参数

参数	类型	是否必须	说明
face_liveness	是	float	活体分数值
thresholds	是	array	由服务端返回最新的阈值数据（随着模型的优化，阈值可能会变化），将此参数与返回的face_liveness进行比较，可以作为活体判断的依据。 frr_1e-4：万分之一误识率的阈值；frr_1e-3：千分之一误识率的阈值；frr_1e-2：百分之一误识率的阈值。误识率越低，准确率越高，相应的拒绝率也越高
face_list	是	array	每张图片的详细信息描述，如果只上传一张图片，则只返回一个结果。
+face_token	是	string	人脸图片的唯一标识
+location	是	array	人脸在图片中的位置
++left	是	double	人脸区域离左边界的距离
++top	是	double	人脸区域离上边界的距离

++top	是	double	人脸区域离上边界的距离
++width	是	double	人脸区域的宽度
++height	是	double	人脸区域的高度
++rotation	是	int64	人脸框相对于竖直方向的顺时针旋转角，[-180,180]
+face_probability	是	double	人脸置信度，范围【0~1】，代表这是一张人脸的概率，0最小、1最大。
+angle	是	array	人脸旋转角度参数
++yaw	是	double	三维旋转之左右旋转角[-90(左), 90(右)]
++pitch	是	double	三维旋转之俯仰角度[-90(上), 90(下)]
++roll	是	double	平面内旋转角[-180(逆时针), 180(顺时针)]
+age	否	double	年龄 ，当**face_field包含age时返回**
+expression	否	array	表情，当 **face_field包含expression时返回**
++type	否	string	**none**:不笑；**smile**:微笑；**laugh**:大笑
++probability	否	double	表情置信度，范围【0~1】，0最小、1最大。
+face_shape	否	array	脸型，当**face_field包含faceshape时返回**
++type	否	double	**square**: 正方形 **triangle**:三角形 **oval**: 椭圆 **heart**: 心形 **round**: 圆形
++probability	否	double	置信度，范围【0~1】，代表这是人脸形状判断正确的概率，0最小、1最大。
+gender	否	array	性别，**face_field包含gender时返回**
++type	否	string	male:**男性** female:**女性**
++probability	否	double	性别置信度，范围【0~1】，0代表概率最小、1代表最大。
+glasses	否	array	是否带眼镜，**face_field包含glasses时返回**
++type	否	string	**none**:无眼镜，**common**:普通眼镜，**sun**:墨镜
++probability	否	double	眼镜置信度，范围【0~1】，0代表概率最小、1代表最大。
+landmark	否	array	4个关键点位置，左眼中心、右眼中心、鼻尖、嘴中心。**face_field包含landmark时返回**
+landmark72	否	array	72个特征点位置 **face_field包含landmark时返回**
+quality	否	array	人脸质量信息。**face_field包含quality时返回**
++occlusion	否	array	人脸各部分遮挡的概率，范围[0~1]，0表示完整，1表示不完整
+++left_eye	否	double	左眼遮挡比例，[0-1] ， 1表示完全遮挡
+++righ			

+++right_eye	否	double	右眼遮挡比例，[0-1] ， 1表示完全遮挡
+++nose	否	double	鼻子遮挡比例，[0-1] ， 1表示完全遮挡
+++mouth	否	double	嘴巴遮挡比例，[0-1] ， 1表示完全遮挡
+++left_cheek	否	double	左脸颊遮挡比例，[0-1] ， 1表示完全遮挡
+++right_cheek	否	double	右脸颊遮挡比例，[0-1] ， 1表示完全遮挡
+++chin	否	double	下巴遮挡比例， ，[0-1] ， 1表示完全遮挡
++blur	否	double	人脸模糊程度，范围[0~1]，0表示清晰，1表示模糊
++illumination	否	double	取值范围在[0~255], 表示脸部区域的光照程度 越大表示光照越好
++completeness	否	int64	人脸完整度，0或1, 0为人脸溢出图像边界，1为人脸都在图像边界内
+parsing_info	否	string	人脸分层结果 结果数据是使用gzip压缩后再base64编码 使用前需base64解码后再解压缩 **原数据格式为string 形如0,0,0,0,0,1,1,1,1,1,1,2,2,2,2,2,2,2,2,2,…**

返回示例


```
{
"thresholds": {
    "frr_1e-4": 0.05,  //万分之一误拒率的阈值
    "frr_1e-3": 0.3,   //千分之一误拒率的阈值
    "frr_1e-2": 0.9    //百分之一误拒率的阈值
},
"face_liveness": 0.05532243927,
"face_list": [
    {
        "face_token": "df46f7c7db4aa09a093c26fb8d1a8d44",
        "location": {
            "left": 328.9026489,
            "top": 97.16340637,
            "width": 162,
            "height": 154,
            "rotation": 32
        },
        "face_probability": 1,
        "angle": {
            "yaw": 10.16196251,
            "pitch": 2.244354248,
            "roll": 33.82199097
        },
        "liveness": {
            "faceliveness": 0.004187555984,
            "livemapscore": 0.04492170034
        },
        "age": 23
    },
    {
        "face_token": "901d2c64274fccd687d311a6e6110a01",
        "location": {
```

```
            "left": 411.4876404,
            "top": 166.3593445,
            "width": 329,
            "height": 308,
            "rotation": 45
        },
        "face_probability": 0.9194830656,
        "angle": {
            "yaw": -1.716423035,
            "pitch": 7.344647408,
            "roll": 45.79914856
        },
        "liveness": {
            "faceliveness": 0.0001665892196,
            "livemapscore": 0.001787073661
        },
        "age": 23
    },
    {
        "face_token": "7d57e36981c48b4946eb97c8d838b02a",
        "location": {
            "left": 161.4559937,
            "top": 199.8726501,
            "width": 218,
            "height": 201,
            "rotation": -1
        },
        "face_probability": 1,
        "angle": {
            "yaw": -8.187754631,
            "pitch": 6.973727226,
            "roll": -1.25429821
        },
        "liveness": {
            "faceliveness": 0.02942637168,
            "livemapscore": 0.05532243927
        },
        "age": 23
    }
  ]
}
```


人脸对比

接口能力

- **两张人脸图片相似度对比**：比对两张图片中人脸的相似度，并返回相似度分值；
- **多种图片类型**：支持**生活照**、**证件照**、**身份证芯片照**、**带网纹照**四种类型的人脸对比；
- **活体检测**：基于图片中的破绽分析，判断其中的人脸是否为**二次翻拍**（举例：如用户A用手机拍摄了一张包含人脸的图片一，用户B翻拍了图片一得到了图片二，并用图片二伪造成用户A去进行识别操作，这种情况普遍发生在金融开户、实名认证等环节。）；
- **质量检测**：返回模糊、光照等质量检测信息，用于辅助判断图片是否符合识别要求；

业务应用

用于比对多张图片中的人脸相似度并返回两两比对的得分，可用于判断两张脸是否是同一人的可能性大小。

典型应用场景：如**人证合一验证**，**用户认证**等，可与您现有的人脸库进行比对验证。

```
  std::string file_content;

Json::Value data;
Json::Value image_desp_1;
Json::Value image_desp_2;

aip::get_file_content("./assets/sample1.jpg", &file_content);
image_desp_1["image"] = aip::base64_encode(file_content.c_str(), (int) file_content.size());
image_desp_1["image_type"] = "BASE64";

aip::get_file_content("./assets/sample2.jpg", &file_content);
image_desp_2["image"] = aip::base64_encode(file_content.c_str(), (int) file_content.size());
image_desp_2["image_type"] = "BASE64";

data[0] = image_desp_1;
data[1] = image_desp_2;

Json::Value result = client->match(data);
```

请求参数

参数	必选	类型	说明
image	是	string	图片信息(**总数据大小应小于10M**)，图片上传方式根据image_type来判断。 **两张图片通过json格式上传，格式参考表格下方示例**
image_type	是	string	图片类型 **BASE64**:图片的base64值，base64编码后的图片数据编码后的图片大小不超过2M；**URL**:图片的 URL地址(可能由于网络等原因导致下载图片时间过长)；**FACE_TOKEN**: 人脸图片的唯一标识，调用人脸检测接口时，会为每个人脸图片赋予一个唯一的FACE_TOKEN，同一张图片多次检测得到的FACE_TOKEN是同一个。
face_type	否	string	人脸的类型 **LIVE**表示生活照：通常为手机、相机拍摄的人像图片、或从网络获取的人像图片等，**IDCARD**表示身份证芯片照：二代身份证内置芯片中的人像照片， **WATERMARK**表示带水印证件照：一般为带水印的小图，如公安网小图 **CERT**表示证件照片：如拍摄的身份证、工卡、护照、学生证等证件图片 默认LIVE
quality_control	否	string	图片质量控制 **NONE**: 不进行控制 **LOW**:较低的质量要求 **NORMAL**: 一般的质量要求 **HIGH**: 较高的质量要求 **默认 NONE**
liveness_control	否	string	活体检测控制 **NONE**: 不进行控制 **LOW**:较低的活体要求(高通过率 低攻击拒绝率) **NORMAL**: 一般的活体要求(平衡的攻击拒绝率, 通过率) **HIGH**: 较高的活体要求(高攻击拒绝率 低通过率) **默认NONE**

返回参数

参数名	必选	类型	说明
score	是	float	人脸相似度得分
face_list	是	array	人脸信息列表
+face_token	是	string	人脸的唯一标志

返回示例

```
{
  "score": 44.3,
  "face_list": [ //返回的顺序与传入的顺序保持一致
    {
      "face_token": "fid1"
    },
    {
      "face_token": "fid2"
    }
  ]
}
```

人脸实名认证V4

能力介绍

1. 业务能力

- **质量检测（可选）**：判断图片中是否包含人脸，以及人脸在姿态、遮挡、模糊、光照等方面是否符合识别条件。
- **活体检测（可选）**：基于图片中的破绽分析，判断其中的人脸是否为**二次翻拍**（举例：如用户A用手机拍摄了一张包含人脸的图片一，用户B翻拍了图片一得到了图片二，并用图片二伪造成用户A去进行识别操作，这种情况普遍发生在金融开户、实名认证等环节）。
- **图片加密及风控（可选）**：当配合增强级采集SDK、增强级安全加固采集SDK、金融级采集SDK、金融级安全加固采集SDK版本使用，对采集SDK输出的加密图片进行解密（加密传输可以有效避免第三方非法黑产绕过APP模拟请求攻击云端接口的行为，Eg：脚本攻击等）；以及结合百度安全实验室大数据风控能力，对采集SDK的发起端设备进行风控识别，辨别是否为风险设备，Eg：ROM注入、视频劫持等；
- **人脸实名认证（必选）**：基于姓名和身份证号，调取公安权威数据源人脸图，将当前获取的人脸图片，与公安数据源人脸图进行对比，得出比对分数，并基于此进行业务判断是否为同一人。由于公安数据源人脸图具有最权威的身份证明作用，故对用户本人的验证结果可信度也最为合理。

2. 业务逻辑

- 上述能力，人脸实名认证能力为必选能力，质量检测、活体检测、图片加密及风控为可选能力，验证顺序为`人脸质量检测->活体检测->人脸实名认证`。
- **如选择了质量检测和活体检测能力，则有任意一个条件不通过，整个请求流程终止，并返回错误码，描述具体的不符合信息**。
- 基于此顺序串行验证逻辑，可以避免大量不符合条件的请求流转到人脸实名认证，节约您的成本。

3. 推荐阈值

- 此接口使用的对比算法，针对带水纹证件照采用了专项的模型处理，可保证水纹信息的影响降到尽可能低。
- 如比对成功，最终返回的有效数据为一个**对比分值**，在0~100之间，您可以设定具体的阈值来判断是否验证通过。
- **人证相似度的推荐阈值为80，对应的误识率为万分之一**。

```
 Json::Value result;

// 必填参数
std::string idCardNumber = "";
std::string name = "";
std::string image = "";

// 可选参数
std::map<std::string, std::string> options;
options["xxx"] = "xxx";

// 调用接口
result = client.faceMingJingVerify(idCardNumber, name, image, options, aip::json_null);
```

请求参数

参数	必选	类型	说明
app	否	string	APP端类型，配合采集SDK使用时须传入 **ios**：iOS端采集SDK **android**：安卓端采集SDK
sec_level	否	string	SDK安全级别，配合采集SDK使用时须传入，默认common **common**：配合4.1/4.1.5版本采集SDK，人脸图片未进行加密处理 **lite**: 配合5.2版本SDK
skey	否	string	使用5.2版本SDK请求时必填 **skey**：从SDK获取的密钥信息skey
x_device_id	否	string	使用5.2版本SDK请求时必填 **deviceId**：从SDK 获取的密钥信息deviceId
data	否	string	使用5.2版本SDK请求时必填， SDK输出的加密数据
id_card_number	是	string	身份证件号
name	是	string	姓名(需要是 utf8 编码)
liveness_control	否	string	活体控制参数 **NONE**: 不进行控制 **LOW**:较低的活体要求(高通过率 低攻击拒绝率) **NORMAL**: 一般的活体要求(平衡的攻击拒绝率, 通过率) **HIGH**: 较高的活体要求(高攻击拒绝率 低通过率) **默认为NONE**
spoofing_control	否	string	合成图控制参数 **NONE**: 不进行控制 **LOW**:较低的合成图阈值数值，由于合成图判定逻辑为大于阈值视为合成图攻击，该项代表**低通过率、高攻击拒绝率** **NORMAL**: 一般的合成图阈值数值，由于合成图判定逻辑为大于阈值视为合成图攻击，该项代表**平衡的攻击拒绝率, 通过率** **HIGH**: 较高的合成图阈值数值，由于合成图判定逻辑为大于阈值视为合成图攻击，该项代表**高通过率、低攻击拒绝率**) **默认为NONE**
quality_control	否	string	质量控制参数 **NONE**: 不进行控制 **LOW**:较低的质量要求 **NORMAL**: 一般的质量要求 **HIGH**: 较高的质量要求 **默认为NONE**
image	是	string	图片信息(数据大小应小于10M 分辨率应小于1920*1080)，**5.2版本SDK请求时已包含在加密数据data中，无需额外传入**
image_type	否	string	图片类型 **BASE64**：图片的base64值 **URL**：图片的 URL **FACE_TOKEN**：人脸标识 **默认 BASE64**

返回参数

参数	类型	说明
log_id	number	调用的日志id
result	jsonObject	认证返回的结果
+score	float	与公安数据源人脸图相似度可能性，用于验证生活照与公安数据源人脸图是否为同一人，有正常分数时为[0~100]，推荐阈值80，超过即判断为同一人
+verify_status	number	认证状态，取值如下： 0 ： 正常 1 ： 身份证号与姓名不匹配或该身份证号不存在 2 ： 公安网图片不存在或质量过低
dec_image	string	对SDK传入的加密图片进行解密。 仅APP场景且进行了图片加密时，此参数返回解密后的人脸图片信息
risk_level	string	判断设备是否发生过风险行为来判断风险级别，取值（数值由高到低）： 1 – 高危 2 – 嫌疑 3 – 普通 4 – 正常
risk_tag	string	风险标签，若判断为有风险，则会有风险标签json 数组告知风险类型 例如：general_inject

返回示例


```
{
  "log_id": 1370579072568000512,
  "result": {
     "score": 40.884,
     "verify_status": 0
  },
  "dec_image": "/9j/4AAQSkZJRgABAgAAAQABAAD",
  "risk_level": "3",
  "risk_tag": [
     "若判断为有风险，则会有风险标签json 数组告知风险类型，如：general_inject"
  ]
}
```


人脸比对V4

能力介绍

1. 接口能力

- **两张人脸图片相似度对比**：对比两张图片中人脸的相似度，并返回相似度分值；
- **多种图片类型**：支持**生活照**、**证件照**、**身份证芯片照**、**带网纹照**四种类型的人脸对比；
- **活体检测控制**：基于图片中的破绽分析，判断其中的人脸是否为**二次翻拍**（举例：如用户A用手机拍摄了一张包含人脸的图片一，用户B翻拍了图片一得到了图片二，并用图片二伪造成用户A去进行识别操作，这种情况普遍发生在金融开户、实名认证等环节。）；
- **质量检测控制**：分析图片的中人脸的模糊度、角度、光照强度等特征，判断图片质量；
- **图片加密及风控：**配合增强级、金融级采集SDK使用
  - 对采集SDK输出的加密图片进行解密（加密传输可以有效避免第三方非法黑产**绕过APP模拟请求攻击云端接口的行为**，Eg：脚本攻击等）；
  - 以及结合百度安全实验室大数据风控能力，对采集SDK的发起端设备进行风控识别，辨别是否为**风险设备**，Eg：ROM注入、视频劫持等；

2. 业务应用

- 用于对比多张图片中的人脸相似度并返回两两对比的得分，可用于判断两张脸是否是同一人的可能性大小。

- 典型应用场景：如**人证合一验证**，**用户认证**等，可与您现有的人脸库进行对比验证。

```
 Json::Value result;

// 必填参数
std::string image = "";
std::string imageType = "";
std::string registerImage = "";
std::string registerImageType = "";

// 可选参数
std::map<std::string, std::string> options;
options["xxx"] = "xxx";

// 调用接口
result = client.faceMingJingMatch(id_card_number, name, image, options, aip::json_null);
```

请求参数

参数	必选	类型	说明
app	否	string	APP类型，配合采集SDK使用时须传入 **ios**：iOS端采集SDK **android**：安卓端采集SDK
sec_level	否	string	SDK安全级别，配合采集SDK使用时须传入，默认common **common**：配合4.1/4.1.5版本SDK使用，人脸图片未进行加密处理 **lite**：配合5.2版本SDK使用
skey	否	string	使用5.2版本SDK请求时必填 从SDK获取的密钥信息
x_device_id	否	string	使用5.2版本SDK请求时必填 从SDK获取的密钥信息
data	否	string	使用5.2版本SDK请求时必填 SDK输出的加密数据
image	是	string	图片信息(数据大小应小于10M 分辨率应小于1920*1080)，***5.2版本SDK请求时已包含在加密数据data中，无需额外传入**
image_type	是	string	图片类型 **BASE64**：图片的base64值 **URL**：图片的URL **FACE_TOKEN**：人脸标识 默认 BASE64
face_type	否	string	人脸的类型 **LIVE**：表示生活照：通常为手机、相机拍摄的人像图片、或从网络获取的人像图片等 **IDCARD**：表示身份证芯片照：二代身份证内置芯片中的人像照片 **WATERMARK**：表示带水印证件照：一般为带水印的小图，如公安网小图 **CERT**：表示证件照片：如拍摄的身份证、工卡、护照、学生证等证件图片 **INFRARED** 表示红外照片：使用红外相机拍摄的照片 默认LIVE
quality_control	否	string	质量控制 **NONE**: 不进行控制 **LOW**:较低的质量要求 **NORMAL**: 一般的质量要求 **HIGH**: 较高的质量要求 **默认NONE**
liveness_control	否	string	活体控制 **NONE**: 不进行控制 **LOW**:较低的活体要求(高通过率 低攻击拒绝率) **NORMAL**: 一般的活体要求(平衡的攻击拒绝率, 通过率) **HIGH**: 较高的活体要求(高攻击拒绝率 低通过率) **默认NONE**
registerimage	是	string	图片信息(总数据大小应小于10M)，图片上传方式根据imagetype来判断。本图片特指客户服务器上传图片，非加密图片Base64值
registerimagetype	是	string	图片类型 **BASE64**：图片的base64值 **URL**：图片的 URL **FACE_TOKEN**：人脸标识 默认 BASE64
registerfacetype	否	string	人脸的类型
registerqualitycontrol	否	string	图片质量控制 **NONE**: 不进行控制 **LOW**:较低的质量要求 **NORMAL**: 一般的质量要求 **HIGH**: 较高的质量要求 **默认 NONE** 若图片质量不满足要求，则返回结果中会提示质量检测失败
registerlivenesscontrol	否	string	活体检测控制 **NONE**: 不进行控制 **LOW**:较低的活体要求(高通过率 低攻击拒绝率) **NORMAL**: 一般的活体要求(平衡的攻击拒绝率, 通过率) **HIGH**: 较高的活体要求(高攻击拒绝率 低通过率) **默认 NONE** 若活体检测结果不满足要求，则返回结果中会提示活体检测失败
facesorttype	否	int	人脸检测排序类型 0:代表检测出的人脸按照人脸面积从大到小排列 1:代表检测出的人脸按照距离图片中心从近到远排列 默认为0

返回参数

参数名	类型	说明
log_id	number	调用的日志id
result	jsonObject	认证返回的结果
+ score	number	人脸相似度得分，推荐阈值80分
+ face_list	jsonArray	人脸信息列表
++ face_token	string	人脸标志
dec_image	string	APP场景传入加密图片时，该项返回解密后的图片
risk_level	string	风控返回参数，只有在 risk_identify 为 true 时才返回，判断设备是否发生过风险行为来判断风险级别，取值（数值有高到低）： 1 – 高危 2 – 嫌疑 3 – 普通 4 – 正常
risk_tag	jsonArray	风控返回参数，只有在 risk_identify 为 true 时才返回，风险标签，若判断为有风险，则会有风险标签 json 数组告知风险类型
risk_warn_code	number	风控返回参数，只有在 risk_identify 为 true 时才返回，只有在风控服务异常的时候才返回这个字段

返回示例


```
{
  "log_id": 1370585066551377920,
  "result": {
    "score": 99.06919861,
    "face_list": [
      {
        "face_token": "549f9f1d1c7ec8c86931540b1939e8ed"
      },
      {
        "face_token": "1a319460ef89e8d27fb59062a28dbad7"
      }
    ]
  },
  "dec_image": "/9j/4AAQSkZJRgABAQAAAQABAAD",
  "risk_level": "3",
  "risk_tag": [
    "空"
  ]
}
```


在线图片活体V4

能力介绍

1. 接口能力

- **人脸基础信息**：包括人脸框位置，人脸空间旋转角度，人脸置信度等信息。
- **人脸质量检测**：判断人脸的遮挡、光照、模糊度、完整度等质量信息。可用于判断上传的人脸是否符合标准。
- **基于图片的活体检测**：基于单张图片，判断图片中的人脸是否为二次翻拍（举例：如用户A用手机拍摄了一张包含人脸的图片一，用户B翻拍了图片一得到了图片二，并用图片二伪造成用户A去进行识别操作，这种情况普遍发生在金融开户、实名

认证等环节）以及是否为合成图攻击。此能力可用于H5场景下的一些人脸采集场景中，增加人脸注册的安全性和真实性。

- **图片加密及风控**：配合采集SDK5.0版本使用，对采集SDK输出的加密图片进行解密（加密传输可以有效避免第三方非法黑产绕过APP模拟请求攻击云端接口的行为，Eg：脚本攻击等）；以及结合百度安全实验室大数据风控能力，对采集SDK的发起端设备进行风控识别，辨别是否为风险设备，Eg：ROM注入、视频劫持等；

```
 Json::Value result;

// 必填参数
std::string sdkVersion = "1";

// 可选参数
std::map<std::string, std::string> options;
options["xxx"] = "xxx";

// 调用接口
result = client.onlinePictureLiveV4(sdkVersion, options, aip::json_null);
```

请求参数

参数	是否必选	类型	说明
sdk_version	是	string	1 ：非加密图片，适用于4.1/4.1.5版本SDK、H5场景或纯服务端场景 4：适用于5.2版本SDK
face_field	否	string	包括**age,expression,face_shape,gender,glasses,landmark,quality,face_type,spoofing**信息，逗号分隔，默认只返回face_token、活体数、人脸框、概率和旋转角度
option	否	string	场景信息，程序会视不同的场景选用相对应的模型。当前支持的场景有COMMON(通用场景)，GATE(闸机场景)，FINANCE(金融场景)，LOGISTICS(物流场景)，INTERNET(泛互联网场景), **默认使用COMMON** 。注意：如果请求参数中存在多个option值时，则取第一个option的值，同时除通用场景外的其他场景需要联系工作人员对您所使用的appid进行配置
app	否	string	端类型 ios / android **5.2版本SDK必传该项**
s_key	否	string	端上提供的用于解密图片的skey **5.2版本SDK必传该项**
device_id	否	string	端上提供的用于解密图片的deviceId **5.2版本SDK必传该项**
data	否	string	端上提供的加密后的图片数组 **5.2版本SDK必传该项**
image_list	否	array	图片BASE64数组 4.1/4.1.5版本SDK、H5场景或纯服务端场景必填

返回参数

参数	是否必须	类型	说明
log_id	是	string	日志id
error_code	是	int	错误码状态，若为0则表示认证成功
error_msg	是	string	错误码说明，若为 success 则表示认证成功
result	是	object	活体结果
+face_liveness	是	float	活体分数值

+thresholds	是	array	由服务端返回最新的阈值数据（随着模型的优化，阈值可能会变化），将此参数与返回的face_liveness进行比较，可以作为活体判断的依据。 frr_1e-4：万分之一误识率的阈值； frr_1e-3：千分之一误识率的阈值； frr_1e-2：百分之一误识率的阈值。 误识率越低，准确率越高，相应的拒绝率也越高
+face_list	是	array	每张图片的详细信息描述，如果只上传一张图片，则只返回一个结果。
++face_token	是	string	人脸图片的唯一标识
++location	是	array	人脸在图片中的位置
+++left	是	double	人脸区域离左边界的距离
+++top	是	double	人脸区域离上边界的距离
+++width	是	double	人脸区域的宽度
+++height	是	double	人脸区域的高度
+++rotation	是	int64	人脸框相对于竖直方向的顺时针旋转角，[-180,180]
++face_probability	是	double	人脸置信度，范围【0~1】，代表这是一张人脸的概率，0最小、1最大。
++angel	是	array	人脸旋转角度参数
+++yaw	是	double	三维旋转之左右旋转角[-90(左), 90(右)]
+++pitch	是	double	三维旋转之俯仰角度[-90(上), 90(下)]
+++roll	是	double	平面内旋转角[-180(逆时针), 180(顺时针)]
++age	否	double	年龄 ，当**face_field包含age时返回**
++expression	否	array	表情，当 **face_field包含expression时返回**
+++type	否	string	**none**:不笑；**smile**:微笑；**laugh**:大笑
+++probability	否	double	表情置信度，范围【0~1】，0最小、1最大。
++face_shape	否	array	脸型，当**face_field包含face_shape时返回**
+++type	否	double	**square**: 正方形 **triangle**:三角形 **oval**: 椭圆 **heart**: 心形 **round**: 圆形
+++probability	否	double	置信度，范围【0~1】，代表这是人脸形状判断正确的概率，0最小、1最大。
++gender	否	array	性别，**face_field包含gender时返回**
+++type	否	string	male:**男性** female:**女性**

+++probability	否	double	性别置信度，范围【0~1】，0代表概率最小、1代表最大。
++glasses	否	array	是否带眼镜，**face_field包含glasses时返回**
++type	否	string	**none**:无眼镜，**common**:普通眼镜，**sun**:墨镜
+++probability	否	double	眼镜置信度，范围【0~1】，0代表概率最小、1代表最大。
++face_type	否	array	真实人脸/卡通人脸 **face_field包含face_type时返回**
+++type	否	string	**human**: 真实人脸 **cartoon**: 卡通人脸
+++probability	否	double	人脸类型判断正确的置信度，范围【0~1】，0代表概率最小、1代表最大。
++landmark	否	array	4个关键点位置，左眼中心、右眼中心、鼻尖、嘴中心。**face_field包含landmark时返回**
++landmark72	否	array	72个特征点位置 **face_field包含landmark时返回**
++quality	否	array	人脸质量信息。**face_field包含quality时返回**
+++occlusion	否	array	人脸各部分遮挡的概率，范围[0~1]，0表示完整，1表示不完整
++++left_eye	否	double	左眼遮挡比例，[0-1] ， 1表示完全遮挡
++++right_eye	否	double	右眼遮挡比例，[0-1] ， 1表示完全遮挡
++++nose	否	double	鼻子遮挡比例，[0-1] ， 1表示完全遮挡
++++mouth	否	double	嘴巴遮挡比例，[0-1] ， 1表示完全遮挡
++++left_cheek	否	double	左脸颊遮挡比例，[0-1] ， 1表示完全遮挡
++++right_cheek	否	double	右脸颊遮挡比例，[0-1] ， 1表示完全遮挡
++++chin	否	double	下巴遮挡比例，，[0-1] ， 1表示完全遮挡
+++blur	否	double	人脸模糊程度，范围[0~1]，0表示清晰，1表示模糊
+++illumination	否	double	取值范围在[0~255], 表示脸部区域的光照程度 越大表示光照越好
+++completeness	否	int64	人脸完整度，0或1, 0为人脸溢出图像边界，1为人脸都在图像边界内
++spoofing	否	double	合成图打分 判断图片是否为合成图 face_field包含时返回spoofing

risk_level	否	string	判断设备是否发生过风险行为来判断风险级别，取值（数值有高到低）：1 – 高危 2 – 嫌疑 3 – 普通 4 – 正常
risk_tag	否	string	风险标签，若判断为有风险，则会有风险标签 json 数组告知风险类型
dec_image	否	array	非安全加固增强级采集sdk、非安全加固金融级采集sdk、安全加固增强级采集sdk、安全加固金融级采集sdk会返回解密后的原图

返回示例


```
{
  "result": {
    "thresholds": {
      "frr_1e-3": 0.3,
      "frr_1e-2": 0.9,
      "frr_1e-4": 0.05
    },
    "face_liveness": 0.1976952702,
    "face_list": [
      {
        "liveness": {
          "livemapscore": 0.1976952702
        },
        "angle": {
          "roll": 0.31,
          "pitch": -2.07,
          "yaw": 0.85
        },
        "face_token": "abd1b4ce743099336cfed40193ff4944",
        "location": {
          "top": 161.74,
          "left": 79.04,
          "rotation": 1,
          "width": 318,
          "height": 333
        },
        "face_probability": 1
      },
      {
        "liveness": {
          "livemapscore": 0.1976952702
        },
        "angle": {
          "roll": 0.31,
          "pitch": -2.07,
          "yaw": 0.85
        },
        "face_token": "abd1b4ce743099336cfed40193ff4944",
        "location": {
          "top": 161.74,
          "left": 79.04,
          "rotation": 1,
          "width": 318,
          "height": 333
        },
        "face_probability": 1
      },
      {
        "liveness": {
          "livemapscore": 0.1976952702
        },
```

```
            "angle": {
                "roll": 0.31,
                "pitch": -2.07,
                "yaw": 0.85
            },
            "face_token": "abd1b4ce743099336cfed40193ff4944",
            "location": {
                "top": 161.74,
                "left": 79.04,
                "rotation": 1,
                "width": 318,
                "height": 333
            },
            "face_probability": 1
        },
        {
            "liveness": {
                "livemapscore": 0.1976952702
            },
            "angle": {
                "roll": 0.31,
                "pitch": -2.07,
                "yaw": 0.85
            },
            "face_token": "abd1b4ce743099336cfed40193ff4944",
            "location": {
                "top": 161.74,
                "left": 79.04,
                "rotation": 1,
                "width": 318,
                "height": 333
            },
            "face_probability": 1
        },
        {
            "liveness": {
                "livemapscore": 0.1976952702
            },
            "angle": {
                "roll": 0.31,
                "pitch": -2.07,
                "yaw": 0.85
            },
            "face_token": "abd1b4ce743099336cfed40193ff4944",
            "location": {
                "top": 161.74,
                "left": 79.04,
                "rotation": 1,
                "width": 318,
                "height": 333
            },
            "face_probability": 1
        },
        {
            "liveness": {
                "livemapscore": 0.1976952702
            },
            "angle": {
                "roll": 0.31,
                "pitch": -2.07,
                "yaw": 0.85
            },
            "face_token": "abd1b4ce743099336cfed40193ff4944"
```

```
            "face_token": "abd1b4ce743099336cfed40193ff4944",
            "location": {
                "top": 161.74,
                "left": 79.04,
                "rotation": 1,
                "width": 318,
                "height": 333
            },
            "face_probability": 1
        },
        {
            "liveness": {
                "livemapscore": 0.1976952702
            },
            "angle": {
                "roll": 0.31,
                "pitch": -2.07,
                "yaw": 0.85
            },
            "face_token": "abd1b4ce743099336cfed40193ff4944",
            "location": {
                "top": 161.74,
                "left": 79.04,
                "rotation": 1,
                "width": 318,
                "height": 333
            },
            "face_probability": 1
        },
        {
            "liveness": {
                "livemapscore": 0.1976952702
            },
            "angle": {
                "roll": 0.31,
                "pitch": -2.07,
                "yaw": 0.85
            },
            "face_token": "abd1b4ce743099336cfed40193ff4944",
            "location": {
                "top": 161.74,
                "left": 79.04,
                "rotation": 1,
                "width": 318,
                "height": 333
            },
            "face_probability": 1
        }
    ]
},
"risk_level": "3",
"log_id": 1423545654699779516,
"risk_tag": [
    "空"
],
"dec_image": [
        "/9j/4AAQ..."
]
}
```


错误信息

错误返回格式

若请求错误，服务器将返回的JSON文本包含以下参数：

- **error_code**：错误码。
- **error_msg**：错误描述信息，帮助理解和解决发生的错误。

错误码

错误码	错误信息	描述
1	Unknown error	服务器内部错误，请再次请求， 如果持续出现此类错误，请通过QQ群（860337848）或提交工单联系技术支持团队。
2	Service temporarily unavailable	服务暂不可用，请再次请求， 如果持续出现此类错误，请通过QQ群（860337848）或提交工单联系技术支持团队。
3	Unsupported openapi method	调用的API不存在，请检查请求URL后重新尝试，一般为URL中有非英文字符，如“-”，可手动输入重试
4	Open api request limit reached	集群超限额，请再次请求， 如果持续出现此类错误，请通过QQ群（860337848）或提交工单联系技术支持团队。
6	No permission to access data	无权限访问该用户数据，创建应用时未勾选相关接口
13	Get service token failed	获取token失败
14	IAM Certification failed	IAM 鉴权失败
15	app not exsits or create failed	应用不存在或者创建失败
17	Open api daily request limit reached	每天请求量超限额，可通过QQ群（860337848）联系群管、提交工单提升限额
18	Open api qps request limit reached	QPS超限额，可通过QQ群（860337848）联系群管、提交工单提升限额
19	Open api total request limit reached	请求总量超限额，可通过QQ群（860337848）联系群管、提交工单提升限额
100	Invalid parameter	无效的access_token参数，请检查后重新尝试
110	Access token invalid or no longer valid	access_token无效
111	Access token expired	access token过期
222001	param[] is null	必要参数未传入
222002	param[start] format error	参数格式错误
222003	param[length] format error	参数格式错误
222004	param[op_app_id_list] format error	参数格式错误
222005	param[group_id_list] format error	参数格式错误
222006	group_id format error	参数格式错误
222007	uid format error	参数格式错误
222008	face_id format error	参数格式错误

222008	face_id format error	参数格式错误
222009	quality_conf format error	参数格式错误
222010	user_info format error	参数格式错误
222011	param[uid_list] format error	参数格式错误
222012	param[op_app_id] format error	参数格式错误
222013	param[image] format error	参数格式错误
222014	param[app_id] format error	参数格式错误
222015	param[image_type] format error	参数格式错误
222016	param[max_face_num] format error	参数格式错误
222017	param[face_field] format error	参数格式错误
222018	param[user_id] format error	参数格式错误
222019	param[quality_control] format error	参数格式错误
222020	param[liveness_control] format error	参数格式错误
222021	param[max_user_num] format error	参数格式错误
222022	param[id_card_number] format error	参数格式错误
222023	param[name] format error	参数格式错误
222024	param[face_type] format error	参数格式错误
222025	param[face_token] format error	参数格式错误
222026	param[max_star_num] format error	参数格式错误
222201	network not available	服务端请求失败
222202	pic not has face	图片中没有人脸
222203	image check fail	无法解析人脸
222204	image_url_download_fail	从图片的url下载图片失败
222205	network not availablel	服务端请求失败
222206	rtse service return fail	服务端请求失败
222207	match user is not found	未找到匹配的用户
222208	the number of image	图片的数量错误

222208	is incorrect	图片的数量错误
222209	face token not exist	face token不存在
222300	add face fail	人脸图片添加失败
222301	get face fail	获取人脸图片失败
222302	system error	服务端请求失败
222303	get face fail	获取人脸图片失败
223100	group is not exist	操作的用户组不存在
223101	group is already exist	该用户组已存在
223102	user is already exist	该用户已存在
223103	user is not exist	找不到该用户
223104	group_list is too large	group_list包含组 数量过多
223105	face is already exist	该人脸已存在
223106	face is not exist	该人脸不存在
223110	uid_list is too large	uid_list包含数量过多
223111	dst group is not exist	目标用户组不存在
223112	quality_conf format error	quality_conf格式不正确
223113	face is covered	人脸有被遮挡
223114	face is fuzzy	人脸模糊
223115	face light is not good	人脸光照不好
223116	incomplete face	人脸不完整
223117	app_list is too large	app_list包含app数量 过多
223118	quality control error	质量控制项错误
223119	liveness control item error	活体控制项错误
223120	liveness check fail	活体检测未通过
223121	left eye is occlusion	质量检测未通过 左眼 遮挡程度过高
223122	right eye is occlusion	质量检测未通过 右眼 遮挡程度过高
223123	left cheek is occlusion	质量检测未通过 左脸 遮挡程度过高
223124	right cheek is occlusion	质量检测未通过 右脸 遮挡程度过高
223125	chin contour is occlusion	质量检测未通过 下巴遮挡程度过高
223126	nose is occlusion	质量检测未通过 鼻子遮挡程度过高
223127	mouth is occlusion	质量检测未通过 嘴巴 遮挡程度过高
	police picture is none or	公安网图片不存在或

222350	low quality	质量过低
222351	id number and name not match or id number not exist	身份证号与姓名不匹配或该身份证号不存在
222352	name format error	身份证名字格式错误
222353	id number format error	身份证号码格式错误
222354	id number not exist	公安库里不存在此身份证号
222355	police picture not exist	身份证号码正确，公安库里没有对应的照片
222360	invalid name or id number	身份证号码或名字非法（公安网校验不通过）
222901	system busy	系统繁忙
222902	system busy	系统繁忙
222903	system busy	系统繁忙
222904	system busy	系统繁忙
222905	system busy	系统繁忙
222906	system busy	系统繁忙
222907	system busy	系统繁忙
222908	system busy	系统繁忙
222909	system busy	系统繁忙
222910	system busy	系统繁忙
222911	system busy	系统繁忙
222912	system busy	系统繁忙
222913	system busy	系统繁忙
222914	system busy	系统繁忙
222915	system busy	系统繁忙
222916	system busy	系统繁忙
222361	system busy	系统繁忙

实名认证相关接口

人脸实名认证V3

人脸识别接口分为V3和V4两个版本，本文档为V3版本接口的说明文档。V3版本接口仅供老用户使用。

> 辨别接口版本的方法是：在百度云后台进入【应用列表】，点击【应用名称】，在【API列表】中可以看到【请求地址】，若请求地址中带有【V3】标识，则您具有的是V3权限，可以阅读本文档；若请求地址中带有【V2】标识，则您具有的是V2权限，应该去阅读V2文档。 如果您对文档内容有任何疑问，可以通过以下几种方式联系我们：

人脸实名认证V3

能力介绍

业务能力

- **质量检测（可选）**：判断图片中是否包含人脸，以及人脸在姿态、遮挡、模糊、光照等方面是否符合识别条件；

- **活体检测（可选）**：基于图片中的破绽分析，判断其中的人脸是否为**二次翻拍**（举例：如用户A用手机拍摄了一张包含人脸的图片一，用户B翻拍了图片一得到了图片二，并用图片二伪造成用户A去进行识别操作，这种情况普遍发生在金融开户、实名认证等环节。）；
- **人脸实名认证（必选）**：基于姓名和身份证号，调取公安权威数据源人脸图，将当前获取的人脸图片，与此公安数据源人脸图进行对比，得出比对分数，并基于此进行业务判断是否为同一人。由于公安数据源人脸图具有最权威的身份证明作用，故对用户本人的验证结果可信度也最为合理。

业务逻辑

- 上述三项能力为顺序串行验证，接口默认返回权威数据源身份对比分值，质量检测和活体检测为可选项。如选择了这两项，则验证顺序为`人脸质量检测->活体检测->人脸实名认证`。您也可以根据业务场景，选择这两项中的某一项或都不选择。
- **如选择了前两个环节，则有任意一个条件不通过，则整个请求流程终止，并返回错误码，描述具体的不符合信息**。
- 基于此顺序串行验证逻辑，可以避免大量不符合条件的请求流转到人脸实名认证，节约您的成本。

推荐阈值

- 此接口使用的对比算法，针对带水纹证件照采用了专项的模型处理，可保证水纹信息的影响降到尽可能低。
- 如比对成功，最终返回的有效数据为一个**对比分值**，在0~100之间，您可以设定具体的阈值来判断是否验证通过。**推荐阈值为80**

计费逻辑

- 前两个环节图片不符合校验规则，会以`error_code`形式反馈，属于正向业务判断，这两个环节的请求全部免费。真正请求到人脸实名认证这步并返回结果，才会进行计费。价格文档

在线调试 您可以在 示例代码中心 中调试该接口，可进行签名验证、查看在线调用的请求内容和返回结果、示例代码的自动生成。

调用方式

请求URL数据格式

向API服务地址使用POST发送请求，必须在URL中带上参数`access_token`，可通过后台的API Key和Secret Key生成，具体方式请参考“Access Token获取”。

示例代码

Bash
PHP
JAVA
Python
Cpp
C#
Node

```
##### !/bin/bash
curl -i -k 'https://aip.baidubce.com/oauth/2.0/token?grant_type=client_credentials&client_id=【百度云应用的AK】&client_secret=【百度云应用的SK】'
```

> **注意**：access_token的有效期为30天，**切记需要每30天进行定期更换，或者每次请求都拉取新token**；

例如此接口，使用HTTPS POST发送：

```
https://aip.baidubce.com/rest/2.0/face/v1/merge?access_token=24.f9ba9c5341b67688ab4added8bc91dec.2592000.1485570332.282335-8574074
```

POST中Body的参数，按照下方请求参数说明选择即可。

> **提示**：如果您为百度云老用户，正在使用其他非AI的服务，可以参考百度云AKSK鉴权方式发送请求，虽然**请求方式**和**鉴权方法**和本文所介绍的不同，但请求参数和返回结果一致。

请求说明

注意事项：

- **请求体格式化**：Content-Type为application/json，通过json格式化请求体。
- **Base64编码**：请求的图片需经过Base64编码，图片的base64编码指将图片数据编码成一串字符串，使用该字符串代替图像地址。您可以首先得到图片的二进制，然后用Base64格式编码即可。需要注意的是，图片的base64编码是不包含图片头的，如
- **图片格式**：现支持PNG、JPG、JPEG、BMP，**不支持GIF图片**
- **人脸识别接口分为V2和V3两个版本，本文档为V3版本接口的说明文档，请确认您在百度云后台获得的是V3版本接口权限，再来阅读本文档。**

> 辨别接口版本的方法是：在百度云后台进入【应用列表】，点击【应用名称】，在【API列表】中可以看到【请求地址】，若请求地址中带有【v3】标识，则您具有的是v3权限，可以阅读本文档；若请求地址中带有【v2】标识，则您具有的是v2权限，应该去阅读v2文档。

请求示例

HTTP方法：POST

请求URL： `https://aip.baidubce.com/rest/2.0/face/v3/person/verify`

URL参数：

参数	值
access_token	通过API Key和Secret Key获取的access_token,参考“Access Token获取”

Header：

参数	值
Content-Type	application/json

Body中放置请求参数，参数详情如下：

请求参数

参数	必选	类型	说明
image	是	string	图片信息，图片上传方式根据image_type来判断
image_type	是	string	图片类型 **BASE64**:图片的Base64值，Base64编码后的文件大小不超过2M；图片尺寸不超过1920*1080 **FACE_TOKEN**: 人脸图片的唯一标识，调用人脸检测接口时，会为每个人脸图片赋予一个唯一的FACE_TOKEN，同一张图片多次检测得到的FACE_TOKEN是同一个，FACE_TOKEN有效期为60min。
id_card_number	是	string	身份证号码
name	是	string	姓名（注：需要是UTF-8编码的中文）
quality_control	否	string	图片质量控制 **NONE**: 不进行控制 **LOW**:较低的质量要求 **NORMAL**: 一般的质量要求 **HIGH**: 较高的质量要求 **默认 NONE**
liveness_control	否	string	活体检测控制 **NONE**: 不进行控制 **LOW**:较低的活体要求(高通过率 低攻击拒绝率) **NORMAL**: 一般的活体要求(平衡的攻击拒绝率, 通过率) **HIGH**: 较高的活体要求(高攻击拒绝率 低通过率) **默认NONE**
spoofing_control	否	string	合成图控制参数 **NONE**: 不进行控制 **LOW**:较低的合成图阈值数值，由于合成图判定逻辑为大于阈值视为合成图攻击，该项代表**低通过率、高攻击拒绝率** **NORMAL**: 一般的合成图阈值数值，由于合成图判定逻辑为大于阈值视为合成图攻击，该项代表**平衡的攻击拒绝率, 通过率** **HIGH**: 较高的合成图阈值数值，由于合成图判定逻辑为大于阈值视为合成图攻击，该项代表**高通过率、低攻击拒绝率**) **默认为NONE**

示例代码

提示一：使用示例代码前，请记得替换其中的示例Token、图片地址或Base64信息。

提示二：部分语言依赖的类或库，请在代码注释中查看下载地址。

Bash

PHP

JAVA

Python

Cpp

C#

```
 身份验证
curl -i -k 'https://aip.baidubce.com/rest/2.0/face/v3/person/verify?access_token=【调用鉴权接口获取的token】' --data '{"image":"sfasq35sadvsvqwr5q...","image_type":"BASE64","id_card_number":"110...","name":"张三","quality_control":"LOW","liveness_control":"HIGH"}' -H 'Content-Type:application/json; charset=UTF-8'
```

返回说明

返回参数

参数	必须	类型	说明
log_id	是	uint64	日志id
score	是	float	与公安数据源人脸图相似度可能性，用于验证生活照与公安数据源人脸图是否为同一人，有正常分数时为[0~100]，推荐阈值80，超过即判断为同一人

返回示例

```
{
  "error_code": 0,
  "error_msg": "SUCCESS",
  "log_id": 4575553579001,
  "timestamp": 1626782958,
  "cached": 0,
  "result":
  {
     "score": 84.8769989
  }
}
```

- **质量控制参数说明**

> 不同的控制度下所对应的质量控制阈值，如果检测出来的质量信息某一项不符合控制阈值的要求，则会返回错误信息。

控制度	left_eye	right_eye	nose	mouth	left_cheek	right_cheek	chin_contour	illumination	blurdegree	completeness
LOW	0.8	0.8	0.8	0.8	0.8	0.8	0.8	20	0.8	0
NORMAL	0.6	0.6	0.6	0.6	0.6	0.6	0.6	40	0.6	0
HIGH	0.3	0.3	0.2	0.2	0.3	0.3	0.3	50	0.2	1

活体控制参数说明

> 不同的控制度下所对应的活体控制阈值，如果检测出来的活体分数小于控制阈值，则会返回错误信息。

控制度	阈值	说明
LOW	0.05	活体误拒率：万分之一；拒绝率：97.75%
NORMAL（推荐）	0.3	活体误拒率：千分之一；拒绝率：98.82%
HIGH	0.9	活体误拒率：百分之一；拒绝率：99.77%

> 1、误拒率: 把**真人识别为假人**的概率. 阈值越高，安全性越高, 要求也就越高, 对应的误识率就越高
> 2、通过率=1-误拒率

关于以上数值的概念介绍：

拒绝率（TRR）：如99%，代表100次作弊假体攻击，会有99次被拒绝。 误拒率（FRR）：如0.5%，指1000次真人请求，会有5次因为活体分数低于阈值被错误拒绝。 通过率（TAR）：如99%，指100次真人请求，会有99次因为活体分数高于阈值而通过。 阈值（Threshold）：高于此数值，则可判断为活体。

合成图控制参数说明

> 不同的控制度下所对应的合成图检测（PS、人脸融合等）阈值，如果检测出来的分数大于控制阈值，则会返回错误信息。

控制度	阈值	误拒率（FRR）	通过率	攻击拒绝率（TRR））
LOW	0.00023	5%	95%	94.93%
NORMAL（推荐）	0.00048	1%	99%	89.71%
HIGH	0.00109	0.1%	99.9%	84.57%

> 1、误拒率: 把**正常图片识别为合成图片**的概率. 阈值越低，安全性越高, 要求也就越高, 对应的误识率就越高
> 2、通过率=1-误拒率

关于以上数值的概念介绍：

阈值（Threshold）：高于此数值，则可判断为是合成图攻击。

身份证与名字比对

身份证与名字比对

能力介绍

验证用户输入的身份证号码和姓名是否匹配，用于判断用户信息是否真实。

在线调试

您可以在 示例代码中心 中调试该接口，可进行签名验证、查看在线调用的请求内容和返回结果、示例代码的自动生成。

调用方式

请求URL数据格式

向API服务地址使用POST发送请求，必须在URL中带上参数access_token，可通过后台的API Key和Secret Key生成，具体方式请参考“Access Token获取”。

示例代码

Bash

PHP

JAVA

Python

Cpp

C#

Node

```
##### !/bin/bash
curl -i -k 'https://aip.baidubce.com/oauth/2.0/token?grant_type=client_credentials&client_id=【百度云应用的AK】&client_secret=【百度云应用的SK】'
```

> **注意**：access_token的有效期为30天，**切记需要每30天进行定期更换，或者每次请求都拉取新token**；

例如此接口，使用HTTPS POST发送：

```
https://aip.baidubce.com/rest/2.0/face/v3/person/idmatch?access_token=24.f9ba9c5341b67688ab4added8bc91dec.2592000.1485570332.282335-8574074
```

POST中Body的参数，按照下方请求参数说明选择即可。

> **提示**：如果您为百度云老用户，正在使用其他非AI的服务，可以参考百度云AKSK鉴权方式发送请求，虽然**请求方式**和**鉴权**

> **方法**和本文所介绍的不同，但请求参数和返回结果一致。

请求说明

注意事项：

- **请求体格式化**：Content-Type为`application/json`，通过json格式化请求体。

请求示例

HTTP方法：`POST`

请求URL：`https://aip.baidubce.com/rest/2.0/face/v3/person/idmatch`

URL参数：

参数	值
access_token	通过API Key和Secret Key获取的access_token,参考“Access Token获取”。

Header：

参数	值
Content-Type	application/json

Body中放置请求参数，参数详情如下：

请求参数

参数	必选	类型	说明
id_card_number	是	string	身份证号
name	是	string	姓名（注：需要是UTF-8编码的中文）

请求示例

```
 {
"id_card_number": 12344,
"name": "张三"
}
```

返回说明

根据error_code判断，为0时表示匹配为同一个人。否则按错误码表的定义，如222351表示身份证号码与名字不匹配。

错误码

请参考人脸识别错误码

人脸实名认证（含有效期核验）

人脸实名认证（含有效期核验）接口

如果您对文档内容有任何疑问，可以通过以下几种方式联系我们：

- 在百度智能云-控制台内 提交工单，咨询问题类型请选择**人工智能服务**；
- 如有需要讨论的疑问，欢迎进入 AI社区 与其他开发者们一同交流。

能力介绍

业务能力

- **质量检测（可选）**：判断图片中是否包含人脸，以及人脸在姿态、遮挡、模糊、光照等方面是否符合识别条件。
- **活体检测（可选）**：基于图片中的破绽分析，判断其中的人脸是否为**二次翻拍**（举例：如用户A用手机拍摄了一张包含人脸的图片一，用户B翻拍了图片一得到了图片二，并用图片二伪造成用户A去进行识别操作，这种情况普遍发生在金融开户、实名认证等环节）。
- **人脸实名信息及有效期核验（必选）**：验证用户输入的身份证号码、姓名、身份证有效期是否匹配，判断用户信息是否真实。并基于姓名和身份证号，调取权威库人脸图，将当前获取的人脸图片，与权威库人脸图进行对比，得出比对分数，并基于此进行业务判断是否为同一人。由于权威库人脸图具有身份证明作用，故对用户本人的验证结果可信度也最为合理。

业务逻辑

- 上述能力，人脸实名认证能力为必选能力，质量检测、活体检测、图片加密及风控为可选能力，验证顺序为`人脸质量检测->活体检测->人脸实名信息及有效期核验`。
- **如选择了质量检测和活体检测能力，则有任意一个条件不通过，整个请求流程终止，并返回错误码，描述具体的不符合信息**。
- 基于此顺序串行验证逻辑，可以避免大量不符合条件的请求流转到人脸实名信息及有效期核验，节约您的成本。

推荐阈值

- 此接口使用的对比算法，针对带水纹证件照采用了专项的模型处理，可保证水纹信息的影响降到尽可能低。
- 如比对成功，最终返回的有效数据为一个**对比分值**，在0~100之间，您可以设定具体的阈值来判断是否验证通过。
- **人证相似度的推荐阈值为80，对应的误识率为万分之一**。

请求说明

注意事项：

- **请求体格式化**：Content-Type为`application/json`，通过`json`格式化请求体。
- **Base64编码**：请求的图片需经过`Base64编码`，图片的base64编码指将图片数据编码成一串字符串，使用该字符串代替图像地址。您可以首先得到图片的二进制，然后用Base64格式编码即可。需要注意的是，图片的base64编码是不包含图片头的，如``
- **图片格式**：现支持PNG、JPG、JPEG、BMP，**不支持GIF图片**

请求示例

HTTP方法：`POST`

请求URL： https://aip.baidubce.com/rest/2.0/face/v4/verify_date

URL参数：

参数	值
access_token	通过API Key和Secret Key获取的access_token,参考“Access Token获取”

Header：

参数	值
Content-Type	application/json

Body中放置请求参数，参数详情如下：

请求参数

参数	必选	类型	说明
name	是	string	姓名(需要是 utf8 编码)
id_card_number	是	string	身份证件号
start_date	是	string	身份证有效期起始日期：年月日8位数字完整传入（如20120422）； 起始日期不得早于19840101（第一代身份证签发日期）
end_date	是	string	身份证有效期截止日期：年月日8位数字完整传入（如20170422）， 1：截止日期须晚于起始日期5年以上（包含5年）； 2：有效期截止日期为 ”长期”时，须转换为“00000000”入参请求；
image	是	string	图片信息(数据大小应小于10M 分辨率应小于1920*1080)
image_type	是	string	图片类型 **BASE64**：图片的base64值
liveness_control	否	string	活体控制参数 **NONE**: 不进行控制 **LOW**:较低的活体要求(高通过率 低攻击拒绝率) **NORMAL**: 一般的活体要求(平衡的攻击拒绝率, 通过率) **HIGH**: 较高的活体要求(高攻击拒绝率 低通过率) **默认为NONE**
spoofing_control	否	string	合成图控制参数 **NONE**: 不进行控制 **LOW**:较低的合成图阈值数值，由于合成图判定逻辑为大于阈值视为合成图攻击，该项代表**低通过率、高攻击拒绝率** **NORMAL**: 一般的合成图阈值数值，由于合成图判定逻辑为大于阈值视为合成图攻击，该项代表**平衡的攻击拒绝率, 通过率** **HIGH**: 较高的合成图阈值数值，由于合成图判定逻辑为大于阈值视为合成图攻击，该项代表**高通过率、低攻击拒绝率**) **默认为NONE**
quality_control	否	string	质量控制参数 **NONE**: 不进行控制 **LOW**:较低的质量要求 **NORMAL**: 一般的质量要求 **HIGH**: 较高的质量要求 **默认为NONE**

请求示例

```
{
  "name": "黄*杰",
  "id_card_number": "460200********1396",
  "start_date": "20160304",
  "end_date": "20260304",
  "image_type": "BASE64",
  "image": "/9j/4AAQSkZJRgAB..."
}
```

返回参数

参数	类型	说明
error_code	number	错误码
error_msg	string	错误信息
log_id	number	调用的日志id
result	Object	认证返回的结果
+verify_status	number	认证状态字段 0：身份信息核验成功 1：身份信息无效，包含“身份证号与姓名不匹配、身份证号不存在、身份证有效期信息不匹配、身份证已挂失”4种情况 2： 公安网图片不存在或质量过低
+verify_score	number	仅在verify_status字段取值为0时返回，用于验证生活照与权威库人脸图是否为同一人，有正常分数时为[0~100]，推荐阈值80，超过即判断为同一人

返回示例

```
{
    "result": {
        "verify_status": 0,
        "verify_score": 39.138
    },
    "error_code": 0,
    "error_msg": "SUCCESS",
    "log_id": 1556871318792218696
}
```

参数说明

- **质量控制参数说明**

> 不同的控制度下所对应的质量控制阈值，如果检测出来的质量信息某一项不符合控制阈值的要求，则会返回错误信息。

控制度	left_eye	right_eye	nose	mouth	left_cheek	right_cheek	chin_contour	illumination	blurdegree	completeness
LOW	0.8	0.8	0.8	0.8	0.8	0.8	0.8	20	0.8	0
NORMAL	0.6	0.6	0.6	0.6	0.6	0.6	0.6	40	0.6	0
HIGH	0.3	0.3	0.2	0.2	0.3	0.3	0.3	50	0.2	1

活体控制参数说明

> 不同的控制度下所对应的活体控制阈值，如果检测出来的活体分数小于控制阈值，则会返回错误信息。

控制度	阈值	说明
LOW	0.05	活体误拒率：万分之一；拒绝率：97.75%
NORMAL（推荐）	0.3	活体误拒率：千分之一；拒绝率：98.82%
HIGH	0.9	活体误拒率：百分之一；拒绝率：99.77%

> 1、误拒率：把**真人识别为假人**的概率。 阈值越高，安全性越高，要求也就越高，对应的误识率就越高。 2、通过率=1-误

> 拒率

关于以上数值的概念介绍：

拒绝率（TRR）：如99%，代表100次作弊假体攻击，会有99次被拒绝。 误拒率（FRR）：如0.5%，指1000次真人请求，会有5次因为活体分数低于阈值被错误拒绝。 通过率（TAR）：如99%，指100次真人请求，会有99次因为活体分数高于阈值而通过。 阈值（Threshold）：高于此数值，则可判断为活体。

合成图控制参数说明

> 不同的控制度下所对应的合成图检测（PS、人脸融合等）阈值，如果检测出来的分数大于控制阈值，则会返回错误信息。

控制度	阈值	误拒率（FRR）	通过率	攻击拒绝率（TRR））
LOW	0.00023	5%	95%	94.93%
NORMAL（推荐）	0.00048	1%	99%	89.71%
HIGH	0.00109	0.1%	99.9%	84.57%

> 1、误拒率：把**正常图片识别为合成图片**的概率。阈值越低，安全性越高，要求也就越高，对应的误识率就越高。 2、通过率=1-误拒率

关于以上数值的概念介绍：

阈值（Threshold）：高于此数值，则可判断为是合成图攻击。

错误码

请参考错误码说明文档。

身份证与名字比对（含有效期核验）

身份证与名字比对（含有效期核验）接口

如果您对文档内容有任何疑问，可以通过以下几种方式联系我们：

- 在百度智能云-控制台内 提交工单，咨询问题类型请选择**人工智能服务**；
- 如有需要讨论的疑问，欢迎进入 AI社区 与其他开发者们一同交流。

能力介绍

业务能力

验证用户输入的身份证号码、姓名、身份证有效期是否匹配，判断用户信息是否真实。

请求说明

注意事项：

- **请求体格式化**：Content-Type为`application/json`，通过json格式化请求体。

请求示例

HTTP方法：`POST`

请求URL： https://aip.baidubce.com/rest/2.0/face/v4/idmatch_date

URL参数：

参数	值
access_token	通过API Key和Secret Key获取的access_token,参考“Access Token获取”

Header：

参数	值
Content-Type	application/json

Body中放置请求参数，参数详情如下：

请求参数

参数	必选	类型	说明
name	是	string	姓名（注：需要是UTF-8编码的中文）
id_card_number	是	string	身份证号
start_date	是	string	身份证有效期起始日期：年月日8位数字完整传入（如20120422）； 起始日期不得早于19840101（第一代身份证签发日期）
end_date	是	string	身份证有效期截止日期：年月日8位数字完整传入（如20170422） 1：截止日期应晚于起始日期5年以上（包含5年）； 2：有效期截止日期为 ”长期”时，须转换为“00000000”入参请求；

请求示例

```
 {
 "name": "黄*杰",
 "id_card_number": "460200********1396",
 "start_date": "20160304",
 "end_date": "20260304"
}
```

返回参数

参数	类型	说明
error_code	number	错误码
error_msg	string	错误信息
log_id	number	日志id
result	object	认证返回的结果
+verify_status	number	认证状态字段 0：身份信息核验成功 1：身份信息无效，包含“身份证号与姓名不匹配”、“身份证号不存在”、“身份证有效期信息不匹配”、“身份证已挂失”4种情况

返回示例

```
{
  "result": {
     "verify_status": 0
  },
  "error_code": 0,
  "error_msg": "SUCCESS",
  "log_id": 1556870623987097460
}
```


错误码

请参考错误码说明文档。

人证核验（含证件状态）

人证核验（含证件状态）接口

如果您对文档内容有任何疑问，可以通过以下几种方式联系我们：

- 在百度智能云-控制台内 提交工单，咨询问题类型请选择**人工智能服务**；
- 如有需要讨论的疑问，欢迎进入 AI社区 与其他开发者们一同交流。

能力介绍

业务能力

验证用户输入的身份证号码、姓名、身份证有效期**是否匹配**，判断用户信息**是否真实**。并返回当前用户证件、户籍状态，包括

- **证件状态**：是否最新、是否挂失、是否过期
- **户籍状态**：有效、暂时失效（户籍迁出，但未迁入）、无效（死亡、失踪、迁出、服兵役、出国境定居、消除重复登记人口、冻结户口、重载注销等

请求说明

注意事项：

- **请求体格式化**：Content-Type为`application/json`，通过`json`格式化请求体。

请求示例

HTTP方法：`POST`

请求URL： https://aip.baidubce.com/rest/2.0/face/v4/idmatch_status

URL参数：

参数	值
access_token	通过API Key和Secret Key获取的access_token,参考“Access Token获取”

Header：

参数	值
Content-Type	application/json

Body中放置请求参数，参数详情如下：

请求参数

参数	必选	类型	说明
name	是	string	姓名（注：需要是UTF-8编码的中文）
id_card_number	是	string	身份证号
start_date	是	string	身份证有效期起始日期：年月日8位数字完整传入（如20120422）； 起始日期不得早于19840101（第一代身份证签发日期）
end_date	是	string	身份证有效期截止日期：年月日8位数字完整传入（如20170422） 1：截止日期应晚于起始日期5年以上（包含5年）； 2：有效期截止日期为 ”长期”时，须转换为“00000000”入参请求；

请求示例

```
{
"name": "黄*杰",
"id_card_number": "460200********1396",
"start_date": "20160304",
"end_date": "20260304"
}
```

返回参数

参数	类型	说明
error_code	number	错误码
error_msg	string	错误信息
log_id	number	日志id
result	object	认证返回的结果
+verify_status	number	认证状态字段 0：身份信息核验成功，并返回当前证件、户籍状态信息 1：身份信息无效，包含“身份信息不存在”、“身份证号不存在”、2种情况。
+isNewest	number	证件是否最新，1 最新、0 非最新
+isLosted	number	证件是否挂失，1 挂失、0 非挂失
+isExpired	number	证件是否过期，1 过期、0 非过期
+hjzt	number	户籍状态，0有效、1暂时失效（户籍迁出，但未迁入）、2无效（死亡、失踪、迁出、服兵役、出国境定居、消除重复登记人口、冻结户口、重载注销等）

返回示例

```
{
  "result": {
     "verify_status": 0
     "isNewest": 1 //证件是否最新
     "isLosted": 0 //证件是否挂失
     "isExpired": 0 //证件是否过期
     "hjzt": 0 //户籍状态
  },
  "error_code": 0,
  "error_msg": "SUCCESS",
  "log_id": 1556870623987097460
}
```

错误码

请参考错误码说明文档。

人像特效相关文档

人脸融合

如果您对文档内容有任何疑问，可以通过以下几种方式联系我们：

- 在百度云控制台内 提交工单，咨询问题类型请选择**人工智能服务**
- 如有需要讨论的疑问，欢迎进入 AI社区 与其他开发者们一同交流
- 特效客户交流QQ群：583486416

能力介绍

对两张人脸进行融合处理，生成的人脸同时具备两张人脸的外貌特征（并不是换脸），此服务具有如下三个业务功能：

- **指定人脸**：当图片中有多张人脸时，可以指定某一张人脸与模板图进行融合
- **图像融合**：将检测到的两张人脸图片进行融合，输出一张融合后的人脸
- **黄反识别**：利用图像识别能力，判断图片中是否存在色情、暴恐血腥场景、敏感人物

在线调试

您可以在 示例代码中心 中调试该接口，可进行签名验证、查看在线调用的请求内容和返回结果、示例代码的自动生成。

调用方式

请求URL数据格式

向API服务地址使用POST发送请求，必须在URL中带上参数`access_token`，可通过后台的API Key和Secret Key生成，具体方式请参考“Access Token获取”。

示例代码

Bash
PHP
JAVA
Python
Cpp

C#

Node

```
##### !/bin/bash
curl -i -k 'https://aip.baidubce.com/oauth/2.0/token?grant_type=client_credentials&client_id=【百度云应用的AK】&client_secret=【百度云应用的SK】'
```

> **注意**：access_token的有效期为30天，**切记需要每30天进行定期更换，或者每次请求都拉取新token**；

POST中Body的参数，按照下方请求参数说明选择即可。

> **提示**：如果您为百度云老用户，正在使用其他非AI的服务，可以参考百度云AKSK鉴权方式发送请求，虽然**请求方式**和**鉴权方法**和本文所介绍的不同，但请求参数和返回结果一致。

请求说明

注意事项：

- **请求体格式化**：Content-Type为application/json，通过json格式化请求体。
- **Base64编码**：请求的图片需经过Base64编码，图片的base64编码指将图片数据编码成一串字符串，使用该字符串代替图像地址。您可以首先得到图片的二进制，然后用Base64格式编码即可。需要注意的是，图片的base64编码是不包含图片头的，如
- **图片格式**：现支持PNG、JPG、JPEG、BMP，**不支持GIF图片**

请求示例

HTTP方法：POST

请求URL：接口维护中，暂不支持新客户接入，详情请咨询您的商务经理，提交合作意向。

URL参数：

参数	值
access_token	通过API Key和Secret Key获取的access_token,参考“Access Token获取”

Header：

参数	值
Content-Type	application/json

Body中放置请求参数，参数详情如下：

请求参数：

参数	必选	类型	说明
version	否	string	服务版本 ，可选（1.0,2.0,3.0,4.0），如不传入该项则默认调用（1.0），2.0/3.0/4.0对merge_degree不生效，对融合效果要求较高可选择2.0（推荐版本），对融合结果的清晰度要求较高可选择3.0，**4.0为最新版本，清晰度及融合效果均有提升，页面功能演示为2.0版本效果**
alpha	否	float	融合参数，可选范围 0-1浮点数，保留两位小数，默认(0), 只在version=4.0时才有效。0代表与目标图人脸最大程度相似（完全换脸），1 代表完全不换脸保留模版图，中间值（如0.5）为进行一般的换脸效果。该参数主要用于连续使用制作一组换脸渐变图片
image_template	是	object	模板图信息，要求被融合的人脸 边缘需要与图片边缘保持一定距离， 图片要求见文档底部
+image	是	string	模板图信息 图片的分辨率要求在1920x1080以下
+image_type	是	string	图片类型 **BASE64**:图片的base64值; **FACE_TOKEN**: 人脸标识
+quality_control	否	string	质量控制 **NONE**: 不进行控制 **LOW**:较低的质量要求 **NORMAL**: 一般的质量要求 **HIGH**: 较高的质量要求 **默认NONE**
+face_location	否	string	指定模板图中进行人脸融合的人脸框位置 不指定时则默认使用最大的人脸 格式形如: `{\"left\": 111.4,\"top\": 96.56,\"width\": 98,\"height\": 98,\"rotation\": 3}` 当image_type为**FACE_TOKEN**时, 此参数无效, 会使用**FACE_TOKEN**对应的人脸
image_target	是	object	目标图信息，要求图片为清晰正脸
+image	是	string	目标图信息 图片的分辨率要求在1920x1080以下
+image_type	是	string	图片类型 **BASE64**:图片的base64值; **URL**:图片的 URL(下载图片时可能由于网络等原因导致下载图片时间过长) **FACE_TOKEN**: 人脸标识
+quality_control	否	string	质量控制 **NONE**: 不进行控制 **LOW**:较低的质量要求 **NORMAL**: 一般的质量要求 **HIGH**: 较高的质量要求 **默认NONE**
+face_location	否	string	指定目标图中进行人脸融合的人脸框位置 不指定时则默认使用最大的人脸 格式形如: `{\"left\": 111.4,\"top\": 96.56,\"width\": 98,\"height\": 98,\"rotation\": 3}` 当image_type为**FACE_TOKEN**时, 此参数无效, 会使用**FACE_TOKEN**对应的人脸
merge			融合度 关系到融合图与目标图的相似度 越高则越相似 LOW:较低的融合度 NORMAL: 一般的融合度

merge_degree	否	string	NORMAL: 一般的融合度 HIGH: 较高的融合度 COMPLETE: 完全融合 默认COMPLETE
position	否	int	水印的位置，取值如下： 0-右下角 1-左下角 2-左上角 3-右上角 默认0
language	否	int	水印的语言，取值如下： 0-中文（AI生成） 1-英文（Generated by AI） 默认0

示例代码

提示一：使用示例代码前，请记得替换其中的示例Token、图片地址或Base64信息。

提示二：部分语言依赖的类或库，请在代码注释中查看下载地址。

Bash

PHP

JAVA

Python

Cpp

C#

```
 人脸融合
curl -i -k 'https://aip.baidubce.com/rest/2.0/face/v1/merge?access_token=【调用鉴权接口获取的token】' --data '{"image_template":{"image":"sfasq35sadvsvqwr5q...","image_type":"BASE64","quality_control":"NONE"},"image_target":{"image":"sfasq35sadvsvqwr5q...","image_type":"BASE64","quality_control":"NONE"}}' -H 'Content-Type:application/json; charset=UTF-8'
```

返回说明

返回参数

- 返回结果

字段	类型	说明
merge_image	string	融合图的BASE64值

- 返回示例


```
{
  "error_code": 0,
  "error_msg": "SUCCESS",
  "log_id": 1234567890123,
  "timestamp": 1533094576,
  "cached": 0,
  "result": {
     "merge_image": "iVBORw0KGgoAAAANSUhEUgAAAeoAAAHqCAYAAADLb..."
  }
}
```


图片要求

目标图 目标图无严格限制， 建议选择 **正脸 清晰** 图像，如下面的手机自拍照

模板图 模板图（template_image），要求被融合的人脸边缘需要与图片边缘保持一定距离，保证被融合的人脸的的核心区域完全在图片中

推荐示意图

不合格示意图

错误码

请参考人脸识别错误码

特别提示

您在使用“人脸融合”服务时，应当遵守相关法律法规的要求以及本平台的相关服务协议，不得将本服务用于非法目的，不得利用本服务从事违法法律法规或侵犯他人合法权益的行为； 您在使用“人脸融合”服务时，应当确保您自身或者您应用的用户使用、上传、发布的照片、素材等信息内容，已取得相关权利人的合法授权，不存在侵犯他人肖像权、版权等合法权益的情形，亦不存在违反相关法律、法规、政策的规定以及公序良俗等情形。 您自行对您应用中由用户使用“人脸融合”服务产生的内容负责，保证其不违法相关法律、法规、政策的规定以及公序良俗等。 您违法上述要求引发的任何纠纷由您自行负责，与百度无关。百度有权立即停止您在本平台中的一切服务并保留追究您法律责任的权利。

人脸属性编辑

如果您对文档内容有任何疑问，可以通过以下几种方式联系我们：

- 在百度云控制台内提交工单，咨询问题类型请选择人工智能服务
- 如有需要讨论的疑问，欢迎进入AI社区 与其他开发者们一同交流
- 特效用户交流QQ群：583486416

能力介绍

对人脸属性特征进行编辑，实现性别互换、年龄改变等特效，为用户生成多种特效照片，此服务具有如下三个业务功能：

- **性别转换：**基于高密度的人脸关键点，改变男女性别面部特征，实现人物性别转换
- **变老人：**对人脸年龄改变过程进行预测，将人脸变为老人面孔
- **变小孩：**对人脸年龄改变过程进行推演，将人脸变为小孩面孔

在线调试

您可以在 示例代码中心 中调试该接口，可进行签名验证、查看在线调用的请求内容和返回结果、示例代码的自动生成。

调用方式

请求URL数据格式

向API服务地址使用POST发送请求，必须在URL中带上参数access_token，可通过后台的API Key和Secret Key生成，具体方式请参考“Access Token获取”。

示例代码

Bash
PHP
JAVA
Python

Cpp

C#

Node

```
##### !/bin/bash
curl -i -k 'https://aip.baidubce.com/oauth/2.0/token?grant_type=client_credentials&client_id=【百度云应用的AK】&client_secret=【百度云应用的SK】'
```

> **注意**：access_token的有效期为30天，**切记需要每30天进行定期更换，或者每次请求都拉取新token**；

例如此接口，使用HTTPS POST发送：

```
https://aip.baidubce.com/rest/2.0/face/v1/editattr?access_token=24.f9ba9c5341b67688ab4added8bc91dec.2592000.1485570332.282335-8574074
```

POST中Body的参数，按照下方请求参数说明选择即可。

> **提示**：如果您为百度云老用户，正在使用其他非AI的服务，可以参考百度云AKSK鉴权方式发送请求，虽然**请求方式**和**鉴权方法**和本文所介绍的不同，但请求参数和返回结果一致。

请求说明

注意事项 ：

- **请求体格式化**：Content-Type为application/json，通过json格式化请求体。
- **Base64编码**：请求的图片需经过Base64编码，图片的base64编码指将图片数据编码成一串字符串，使用该字符串代替图像地址。您可以首先得到图片的二进制，然后用Base64格式编码即可。需要注意的是，图片的base64编码是不包含图片头的，如
- **图片格式**：现支持PNG、JPG、JPEG、BMP，**不支持GIF图片**

请求示例：

HTTP方法：POST

请求URL：https://aip.baidubce.com/rest/2.0/face/v1/editattr

URL参数：

参数	值
access_token	通过API Key和Secret Key获取的access_token,参考“Access Token获取”

Header：

参数	值
Content-Type	application/json

Body中放置请求参数，参数详情如下：

请求参数：

参数	必选	类型	说明
image	是	string	原始图片信息 图片的分辨率要求在256*256以上、在4096*4096以下 大小在4M下 人脸区域要求在64*64以上
image_type	是	string	图片类型 **BASE64**:图片的base64值; **FACE_TOKEN**: 人脸标识
action_type	是	string	人脸编辑方式 **TO_KID**: V1版本变小孩 **TO_OLD**: V1版本变老人 **TO_FEMALE**: V1版本变女生 **TO_MALE**: V1版本变男生 **V2_AGE**：V2版本年龄变换，选择该项后可通过target参数指定年龄 **V2_GENDER**： v2版本性别变换，选择该项后需通过target进一步指定要转换的性别
target	否	int	该参数仅在action_type为V2_AGE或V2_GENDER时生效。 V2_GENDER范围：0或者1（”0”代表转换为男性，”1”代表转换为女性） V2_AGE动作值范围：1-85（代表目标年龄）
quality_control	否	string	质量控制 **NONE**: 不进行控制 **LOW**:较低的质量要求 **NORMAL**: 一般的质量要求 **HIGH**: 较高的质量要求 **默认NONE**
face_location	否	string	当图片中检测到多张人脸时，使用此参数来指定图片中需要编辑的人脸位置，不指定时则默认使用图中最大的人脸 格式形如: `{"left": 111.4,"top": 96.56,"width": 98,"height": 98,"rotation": 3}` 当image_type为**FACE_TOKEN**时，此参数无效，会使用**FACE_TOKEN**对应的人脸

示例代码 ：

```
{
  "image": "sfasq35sadvsvqwr5q...",
  "image_type": "BASE64",
  "quality_control": "NORMAL",
  "action_type": "TO_KID",
  "face_location": "{\"left\": 111.4,\"top\": 96.56,\"width\": 98,\"height\": 98,\"rotation\": 3}"
}
```

返回说明

返回参数

- 返回结果

字段	类型	说明
image	string	编辑后图片的BASE64值

- 返回示例


```
{
  "error_code": 0,
  "error_msg": "SUCCESS",
  "log_id": 1234567890123,
  "timestamp": 1533094576,
  "cached": 0,
  "result": {
    "image": "iVBORw0KGgoAAAANSUhEUgAAAeoAAAHqCAYAAADLb..."
  }
}
```


- 质量控制参数说明

> 不同的控制度下所对应的质量控制阈值 如果检测出来的质量信息某一项不符合控制阈值的要求，则会返回错误。

控制度	left_eye	right_eye	nose	mouth	left_cheek	right_cheek	chin_contour	illumination	blurdegree	completeness	yaw	pitch	roll
LOW	0.8	0.8	0.8	0.8	0.8	0.8	0.8	20	0.8	0	45	45	45
NORMAL	0.6	0.6	0.6	0.6	0.6	0.6	0.6	40	0.6	0	30	30	30
HIGH	0.2	0.2	0.2	0.2	0.2	0.2	0.2	100	0.2	1	20	20	20

参数说明

参数	说明
left_eye	左眼被遮挡的比例 [0-1] 1表示完全遮挡
right_eye	右眼被遮挡的比例 [0-1] 1表示完全遮挡
nose	鼻子被遮挡的比例 [0-1] 1表示完全遮挡
mouth	嘴巴被遮挡的比例 [0-1] 1表示完全遮挡
left_cheek	左脸颊被遮挡的比例 [0-1] 1表示完全遮挡
right_cheek	右脸颊被遮挡的比例 [0-1] 1表示完全遮挡
chin_contour	下巴被遮挡比例 [0-1] 1表示完全遮挡
illumination	光照 [0-255] 0表示光照不好
blurdegree	图片模糊度 [0-1] 1表示完全模糊
completeness	人脸完整度（0或1） 0为人脸溢出图像边界，1为人脸都在图像边界内
yaw	三维旋转之左右旋转角，范围：[-90（左），90（右）] 30阈值代表角度绝对值要求在30内
roll	平面内旋转角，范围：[-180（逆时针），180（顺时针）] 30阈值代表角度绝对值要求在30内
pitch	三维旋转之俯仰角度，范围：[-90（上），90（下）] 30阈值代表角度绝对值要求在30内

错误码

- **通用错误码**：请参考人脸识别错误码
- **特有错误码**：

错误码	错误信息	说明	处理建议
222309	image size is too small	图片尺寸过小，请使用清晰的图片	更换符合尺寸要求的图片
222213	face size is too small	人脸尺寸过小，请保证人脸区域在64*64以上	更换符合人脸区域尺寸要求的图片
222214	face are cartoon images	请使用非卡通的人脸图像	更换符合要求的图片
222215	face quality is not acceptable	人脸属性编辑处理该图像失败，请使用其他图片	请更换图片再进行尝试
222216	face edit attr fail	人脸属性编辑服务不可用，请重试	请重试，多次失败请提交工单

人脸关键点检测

如果您对文档内容有任何疑问，可以通过以下几种方式联系我们：

- 在百度云控制台内提交工单，咨询问题类型请选择人工智能服务
- 如有需要讨论的疑问，欢迎进入AI社区 与其他开发者们一同交流
- 特效用户交流QQ群：583486416

能力介绍

支持人脸 72关键点、150关键点、201 关键点检测，关键点包括人脸、眼睛、眉毛、嘴唇以及鼻子轮廓等，此服务具有如下三个业务功能：

- **人脸五官精准定位**：支持眉毛、眼睛、鼻子、嘴、腮位置的精准定位
- **人脸轮廓精准定位**：支持单人脸/多人脸的精准定位
- **人脸角度判断**：对图片中的人脸进行多种姿态角度判断

在线调试

您可以在 示例代码中心 中调试该接口，可进行签名验证、查看在线调用的请求内容和返回结果、示例代码的自动生成。

调用方式

请求URL数据格式

向API服务地址使用POST发送请求，必须在URL中带上参数access_token，可通过后台的API Key和Secret Key生成，具体方式请参考“Access Token获取”。

示例代码

Bash
PHP
JAVA
Python
Cpp
C#
Node

```
##### !/bin/bash
curl -i -k 'https://aip.baidubce.com/oauth/2.0/token?grant_type=client_credentials&client_id=【百度云应用的AK】&client_secret=【百度云应用的SK】'
```

> **注意**：access_token的有效期为30天，**切记需要每30天进行定期更换，或者每次请求都拉取新token**；

例如此接口，使用HTTPS POST发送：

```
https://aip.baidubce.com/rest/2.0/face/v1/landmark?access_token=24.f9ba9c5341b67688ab4added8bc91dec.2592000.1485570332.282335-8574074
```

POST中Body的参数，按照下方请求参数说明选择即可。

> **提示**：如果您为百度云老用户，正在使用其他非AI的服务，可以参考百度云AKSK鉴权方式发送请求，虽然**请求方式**和**鉴权方法**和本文所介绍的不同，但请求参数和返回结果一致。

请求说明

注意事项 ：

- **请求体格式化**：Content-Type为application/json，通过json格式化请求体。
- **Base64编码**：请求的图片需经过Base64编码，图片的base64编码指将图片数据编码成一串字符串，使用该字符串代替图像地址。您可以首先得到图片的二进制，然后用Base64格式编码即可。需要注意的是，图片的base64编码是不包含图片头的，如
- **图片格式**：现支持PNG、JPG、JPEG、BMP，**不支持GIF图片**

请求示例：

HTTP方法：POST

请求URL：https://aip.baidubce.com/rest/2.0/face/v1/landmark

URL参数：

参数	值
access_token	通过API Key和Secret Key获取的access_token,参考“Access Token获取”

Header：

参数	值
Content-Type	application/json

Body中放置请求参数，参数详情如下：

请求参数

参数	必选	类型	说明
image	是	string	图片信息(**数据大小应小于10M 分辨率应小于1920*1080**)
image_type	是	string	图片类型 **BASE64**:图片的base64值; **FACE_TOKEN**: 人脸标识
max_face_num	否	uint32	最多处理人脸的数目. **默认值为1（仅检测图片中面积最大的那个人脸） 最大值10**
face_field	否	string	包括age,gender,landmark4,landmark72,landmark150,landmark201信息，用逗号分隔. 默认只返回face_token、人脸框、概率和旋转角度

示例代码：

提示一：使用示例代码前，请记得替换其中的示例Token、图片地址或Base64信息。

提示二：部分语言依赖的类或库，请在代码注释中查看下载地址。

Python

```python
##### encoding:utf-8

import requests
import json, base64

'''
人脸检测与关键点检测
'''

request_url = "https://aip.baidubce.com/rest/2.0/face/v1/landmark"
filename = '[输入图片路径]'
file = open(filename, 'rb')
image = file.read()
file.close()
img_base64 = base64.b64encode(image)

access_token = '[调用鉴权接口获取的token]'
request_url = request_url + "?access_token=" + access_token
```

返回说明

返回参数

- 72点关键点返回结果

字段	必选	类型	说明
face_num	是	int	图片中的人脸数量
face_list	是	array	人脸信息列表 字段信息见下
face_token	是	string	人脸标志
location	是	array	人脸在图片中的位置
+left	是	double	人脸区域离左边界的距离
+top	是	double	人脸区域离上边界的距离
+width	是	double	人脸区域的宽度
+height	是	double	人脸区域的高度
+rotation	是	int64	人脸框相对于竖直方向的顺时针旋转角，[-180,180]
face_probability	是	double	人脸置信度，范围0-1
angle	是	array	人脸旋转参数 具体说明参考人脸空间姿态角参考
+yaw	是	double	三维旋转之左右旋转角[-90(左), 90(右)]
+pitch	是	double	三维旋转之俯仰角度[-90(上), 90(下)]
+roll	是	double	平面内旋转角[-180(逆时针), 180(顺时针)]
age	否	double	年龄 face_field包含age时返回
gender	否	array	性别 face_field包含gender时返回
+type	否	string	male:男性 female:女性
+probability	否	double	性别置信度，范围0~1
landmark72	否	array	72个特征点位置 **face_field包含landmark72时返回**

- 150点关键点返回结果

字段	必选	类型	说明
face_num	是	int	图片中的人脸数量
face_list	是	array	人脸信息列表 字段信息见下
face_token	是	string	人脸标志
location	是	array	人脸在图片中的位置
+left	是	double	人脸区域离左边界的距离
+top	是	double	人脸区域离上边界的距离
+width	是	double	人脸区域的宽度
+height	是	double	人脸区域的高度
+rotation	是	int64	人脸框相对于竖直方向的顺时针旋转角，[-180,180]
face_probability	是	double	人脸置信度，范围0-1
angle	是	array	人脸旋转参数 具体说明参考人脸空间姿态角参考
+yaw	是	double	三维旋转之左右旋转角[-90(左), 90(右)]
+pitch	是	double	三维旋转之俯仰角度[-90(上), 90(下)]
+roll	是	double	平面内旋转角[-180(逆时针), 180(顺时针)]
landmark150	否	object	150个特征点位置 **face_field包含landmark150时返回**

- 201点关键点返回结果

字段	必选	类型	说明
face_num	是	int	图片中的人脸数量
face_list	是	array	人脸信息列表 字段信息见下
face_token	是	string	人脸标志
location	是	array	人脸在图片中的位置
+left	是	double	人脸区域离左边界的距离
+top	是	double	人脸区域离上边界的距离
+width	是	double	人脸区域的宽度
+height	是	double	人脸区域的高度
+rotation	是	int64	人脸框相对于竖直方向的顺时针旋转角，[-180,180]
face_probability	是	double	人脸置信度，范围0-1
angle	是	array	人脸旋转参数 具体说明参考人脸空间姿态角参考
+yaw	是	double	三维旋转之左右旋转角[-90(左), 90(右)]
+pitch	是	double	三维旋转之俯仰角度[-90(上), 90(下)]
+roll	是	double	平面内旋转角[-180(逆时针), 180(顺时针)]
landmark201	否	object	201个特征点位置 **face_field包含landmark201时返回**

- 72点关键点返回示例


```
{
"error_code": 0,
"error_msg": "SUCCESS",
"log_id": 1476521144,
"timestamp": 1743659819,
"cached": 0,
"result": {
    "face_num": 1,
    "face_list": [
        {
            "face_token": "3e2d00fe72627db64f987e1c09978581",
            "location": {
                "left": 81.3,
                "top": 204.6,
                "width": 164,
                "height": 150,
                "rotation": 0
            },
            "face_probability": 1,
            "angle": {
                "yaw": -4.81,
                "pitch": 22.92,
                "roll": -0.13
            },
            "landmark72": [
                {
                    "x": 80.98,
                    "y": 223
                },
                {
                    "x": 82.93,
                    "y": 247.92
                },
                {
```

```
            "x": 88.63,
            "y": 272.79
        },
        {
            "x": 97.12,
            "y": 297.57
        },
        {
            "x": 113.36,
            "y": 322.46
        },
        {
            "x": 136.49,
            "y": 345.98
        },
        {
            "x": 161.76,
            "y": 355.2
        },
        {
            "x": 187.14,
            "y": 345.5
        },
        {
            "x": 210.67,
            "y": 322.32
        },
        {
            "x": 227.7,
            "y": 297.97
        },
        {
            "x": 236.96,
            "y": 273.21
        },
        {
            "x": 242.89,
            "y": 248.37
        },
        {
            "x": 245.55,
            "y": 223.46
        },
        {
            "x": 102.66,
            "y": 236.55
        },
        {
            "x": 111.68,
            "y": 231.81
        },
        {
            "x": 121.33,
            "y": 230.8
        },
        {
            "x": 130.88,
            "y": 233.37
        },
        {
            "x": 138.74,
            "y": 242.21
        }
```

```
        },
        {
            "x": 129.83,
            "y": 243.32
        },
        {
            "x": 119.54,
            "y": 244.11
        },
        {
            "x": 109.87,
            "y": 241.24
        },
        {
            "x": 123.76,
            "y": 236.71
        },
        {
            "x": 90.16,
            "y": 213.62
        },
        {
            "x": 102.93,
            "y": 204.62
        },
        {
            "x": 116.79,
            "y": 205.27
        },
        {
            "x": 129.74,
            "y": 208.66
        },
        {
            "x": 141.81,
            "y": 217.88
        },
        {
            "x": 128.49,
            "y": 216.39
        },
        {
            "x": 115.65,
            "y": 213.95
        },
        {
            "x": 102.71,
            "y": 212.46
        },
        {
            "x": 186.15,
            "y": 242.42
        },
        {
            "x": 194.05,
            "y": 233.94
        },
        {
            "x": 203.41,
            "y": 230.99
        },
        {
            "x": 213.23,
```

```
            "y": 231.92
        },
        {
            "x": 221.99,
            "y": 236.79
        },
        {
            "x": 215.03,
            "y": 241.68
        },
        {
            "x": 205.47,
            "y": 244.4
        },
        {
            "x": 195.19,
            "y": 243.54
        },
        {
            "x": 202.79,
            "y": 236.93
        },
        {
            "x": 181.73,
            "y": 217.62
        },
        {
            "x": 193.91,
            "y": 208.28
        },
        {
            "x": 206.73,
            "y": 205.08
        },
        {
            "x": 221.08,
            "y": 204.42
        },
        {
            "x": 234.11,
            "y": 213.37
        },
        {
            "x": 221.24,
            "y": 211.96
        },
        {
            "x": 208.16,
            "y": 213.9
        },
        {
            "x": 195.2,
            "y": 216.27
        },
        {
            "x": 151.27,
            "y": 243.53
        },
        {
            "x": 148.34,
            "y": 259.64
        },
```

```
            {
                "x": 145.44,
                "y": 276.05
            },
            {
                "x": 140.61,
                "y": 291.71
            },
            {
                "x": 150.28,
                "y": 294.71
            },
            {
                "x": 173.61,
                "y": 294.6
            },
            {
                "x": 183.39,
                "y": 292.01
            },
            {
                "x": 178.73,
                "y": 276.29
            },
            {
                "x": 176.04,
                "y": 259.83
            },
            {
                "x": 173.38,
                "y": 243.62
            },
            {
                "x": 161.73,
                "y": 287.67
            },
            {
                "x": 133.06,
                "y": 313.46
            },
            {
                "x": 147.86,
                "y": 312.64
            },
            {
                "x": 161.28,
                "y": 313.43
            },
            {
                "x": 175.04,
                "y": 313.08
            },
            {
                "x": 189.28,
                "y": 313.96
            },
            {
                "x": 177.04,
                "y": 325.97
            },
            {
                "x": 161.14,
```

```
                "y": 330.9
            },
            {
                "x": 144.91,
                "y": 325.94
            },
            {
                "x": 147.7,
                "y": 317.36
            },
            {
                "x": 161.33,
                "y": 319.51
            },
            {
                "x": 174.9,
                "y": 317.73
            },
            {
                "x": 174.73,
                "y": 318.57
            },
            {
                "x": 161.19,
                "y": 320.12
            },
            {
                "x": 147.56,
                "y": 318.51
            }
        ]
    }
  ]
 }
}
```


- 150点关键点返回示例


```
{
  "error_code": 0,
  "error_msg": "SUCCESS",
  "log_id": 3069319197,
  "timestamp": 1584341469,
  "cached": 0,
  "result": {
    "face_num": 1,
    "face_list": [
      {
        "face_token": "46c7fb0776721b391cc51c574095e2cd",
        "location": {
          "left": 30.95,
          "top": 121.14,
          "width": 88,
          "height": 84,
          "rotation": -5
        },
        "face_probability": 1,
        "angle": {
          "yaw": -2.64,
          "pitch": 11.24,
          "roll": -7.75
        },
```

```
        "landmark150": {
            "cheek_right_1": {
                "x": 31.09,
                "y": 135.51
            },
            "cheek_right_3": {
                "x": 33.85,
                "y": 150.02
            },
            "cheek_right_5": {
                "x": 37.83,
                "y": 164.53
            },
            "cheek_right_7": {
                "x": 44.83,
                "y": 178.83
            },
            "cheek_right_9": {
                "x": 58.79,
                "y": 191.43
            },
            "cheek_right_11": {
                "x": 73.89,
                "y": 198.59
            },
            "chin_2": {
                "x": 88.53,
                "y": 199.96
            },
            "cheek_left_11": {
                "x": 101.43,
                "y": 193.34
            },
            "cheek_left_9": {
                "x": 112.07,
                "y": 180.59
            },
            "cheek_left_7": {
                "x": 118.56,
                "y": 167.04
            },
            "cheek_left_5": {
                "x": 120.58,
                "y": 153.53
            },
            "cheek_left_3": {
                "x": 121.08,
                "y": 140.28
            },
            "cheek_left_1": {
                "x": 120.61,
                "y": 127.1
            },
            "eye_right_corner_right": {
                "x": 53.63,
                "y": 135.28
            },
            "eye_right_eyelid_upper_2": {
                "x": 57.74,
                "y": 132.45
            },
            "eye_right_eyelid_upper_4": {
```

```
            "x": 62.02,
            "y": 131.31
        },
        "eye_right_eyelid_upper_6": {
            "x": 66.21,
            "y": 131.8
        },
        "eye_right_corner_left": {
            "x": 69.98,
            "y": 134.82
        },
        "eye_right_eyelid_lower_6": {
            "x": 66.31,
            "y": 135.75
        },
        "eye_right_eyelid_lower_4": {
            "x": 62.27,
            "y": 136.34
        },
        "eye_right_eyelid_lower_2": {
            "x": 57.86,
            "y": 136.13
        },
        "eye_right_eyeball_center": {
            "x": 62.39,
            "y": 133.52
        },
        "eyebrow_right_corner_right": {
            "x": 45.44,
            "y": 124.85
        },
        "eyebrow_right_upper_2": {
            "x": 50.34,
            "y": 118.84
        },
        "eyebrow_right_upper_3": {
            "x": 56.93,
            "y": 117.58
        },
        "eyebrow_right_upper_4": {
            "x": 63.4,
            "y": 118.4
        },
        "eyebrow_right_corner_left": {
            "x": 69.55,
            "y": 122.35
        },
        "eyebrow_right_lower_3": {
            "x": 63.31,
            "y": 122.64
        },
        "eyebrow_right_lower_2": {
            "x": 57.22,
            "y": 122.76
        },
        "eyebrow_right_lower_1": {
            "x": 51.2,
            "y": 123.64
        },
        "eye_left_corner_right": {
            "x": 93.13,
            "y": 132.61
        },
```

```
        },
        "eye_left_eyelid_upper_2": {
            "x": 96.34,
            "y": 129.04
        },
        "eye_left_eyelid_upper_4": {
            "x": 100.29,
            "y": 127.74
        },
        "eye_left_eyelid_upper_6": {
            "x": 104.52,
            "y": 127.83
        },
        "eye_left_corner_left": {
            "x": 108.63,
            "y": 129.88
        },
        "eye_left_eyelid_lower_6": {
            "x": 105.2,
            "y": 131.28
        },
        "eye_left_eyelid_lower_4": {
            "x": 101.24,
            "y": 132.29
        },
        "eye_left_eyelid_lower_2": {
            "x": 97.1,
            "y": 132.55
        },
        "eye_left_eyeball_center": {
            "x": 100.54,
            "y": 129.79
        },
        "eyebrow_left_corner_right": {
            "x": 90.87,
            "y": 121.61
        },
        "eyebrow_left_upper_2": {
            "x": 96.07,
            "y": 116.47
        },
        "eyebrow_left_upper_3": {
            "x": 101.97,
            "y": 114.28
        },
        "eyebrow_left_upper_4": {
            "x": 108.55,
            "y": 113.94
        },
        "eyebrow_left_corner_left": {
            "x": 114.28,
            "y": 118.94
        },
        "eyebrow_left_lower_3": {
            "x": 108.78,
            "y": 118.93
        },
        "eyebrow_left_lower_2": {
            "x": 102.9,
            "y": 119.49
        },
        "eyebrow_left_lower_1": {
            "x": 97.04,
```

```
            "y": 120.66
        },
        "nose_right_contour_1": {
            "x": 76.06,
            "y": 134.79
        },
        "nose_right_contour_2": {
            "x": 74.9,
            "y": 142.66
        },
        "nose_right_contour_3": {
            "x": 73.79,
            "y": 150.49
        },
        "nose_right_contour_4": {
            "x": 70.72,
            "y": 158.93
        },
        "nose_right_contour_6": {
            "x": 77.66,
            "y": 160.41
        },
        "nose_left_contour_6": {
            "x": 92.13,
            "y": 159.04
        },
        "nose_left_contour_4": {
            "x": 97.49,
            "y": 156.25
        },
        "nose_left_contour_3": {
            "x": 93.15,
            "y": 148.51
        },
        "nose_left_contour_2": {
            "x": 90.26,
            "y": 141.06
        },
        "nose_left_contour_1": {
            "x": 87.39,
            "y": 133.65
        },
        "nose_tip": {
            "x": 85.25,
            "y": 155.76
        },
        "mouth_corner_right_outer": {
            "x": 67.23,
            "y": 172.68
        },
        "mouth_lip_upper_outer_3": {
            "x": 76.65,
            "y": 170.57
        },
        "mouth_lip_upper_outer_6": {
            "x": 85.6,
            "y": 169.93
        },
        "mouth_lip_upper_outer_9": {
            "x": 93.81,
            "y": 168.66
        },
```

```
"mouth_corner_left_outer": {
    "x": 101.41,
    "y": 169.23
},
"mouth_lip_lower_outer_9": {
    "x": 96.02,
    "y": 176.86
},
"mouth_lip_lower_outer_6": {
    "x": 87.07,
    "y": 180.42
},
"mouth_lip_lower_outer_3": {
    "x": 76.5,
    "y": 178.75
},
"mouth_lip_upper_inner_3": {
    "x": 76.87,
    "y": 172.37
},
"mouth_lip_upper_inner_6": {
    "x": 85.87,
    "y": 172.12
},
"mouth_lip_upper_inner_9": {
    "x": 93.97,
    "y": 170.48
},
"mouth_lip_lower_inner_9": {
    "x": 94.27,
    "y": 173.81
},
"mouth_lip_lower_inner_6": {
    "x": 86.45,
    "y": 176.01
},
"mouth_lip_lower_inner_3": {
    "x": 77.4,
    "y": 175.6
},
"cheek_right_2": {
    "x": 32.54,
    "y": 142.76
},
"cheek_right_4": {
    "x": 35.73,
    "y": 157.36
},
"cheek_right_6": {
    "x": 40.79,
    "y": 172.17
},
"cheek_right_8": {
    "x": 51.25,
    "y": 185.77
},
"cheek_right_10": {
    "x": 66.12,
    "y": 195.51
},
"chin_1": {
    "x": 81.12,
    "y": 200.23
```

```
        "y": 200.23
    },
    "chin_3": {
        "x": 95.6,
        "y": 197.66
    },
    "cheek_left_10": {
        "x": 107,
        "y": 187.2
    },
    "cheek_left_8": {
        "x": 115.79,
        "y": 174.06
    },
    "cheek_left_6": {
        "x": 119.94,
        "y": 160.41
    },
    "cheek_left_4": {
        "x": 121,
        "y": 146.91
    },
    "cheek_left_2": {
        "x": 120.94,
        "y": 133.6
    },
    "eyebrow_right_upper_1": {
        "x": 45.19,
        "y": 123.14
    },
    "eyebrow_right_upper_5": {
        "x": 69.47,
        "y": 120.23
    },
    "eyebrow_left_upper_1": {
        "x": 90.45,
        "y": 119.31
    },
    "eyebrow_left_upper_5": {
        "x": 114.14,
        "y": 117.36
    },
    "eye_right_eyelid_upper_1": {
        "x": 55.53,
        "y": 133.66
    },
    "eye_right_eyelid_upper_3": {
        "x": 59.77,
        "y": 131.68
    },
    "eye_right_eyelid_upper_5": {
        "x": 64.2,
        "y": 131.28
    },
    "eye_right_eyelid_upper_7": {
        "x": 68.28,
        "y": 132.99
    },
    "eye_right_eyelid_lower_7": {
        "x": 68.09,
        "y": 135.3
    },
    "eye_right_eyelid_lower_5": {
```

```
            "x": 64.33,
            "y": 136.24
        },
        "eye_right_eyelid_lower_3": {
            "x": 60.03,
            "y": 136.49
        },
        "eye_right_eyelid_lower_1": {
            "x": 55.75,
            "y": 135.79
        },
        "eye_right_eyeball_right": {
            "x": 58.72,
            "y": 134.1
        },
        "eye_right_eyeball_left": {
            "x": 65.72,
            "y": 133.6
        },
        "eye_left_eyelid_upper_1": {
            "x": 94.44,
            "y": 130.55
        },
        "eye_left_eyelid_upper_3": {
            "x": 98.16,
            "y": 128.15
        },
        "eye_left_eyelid_upper_5": {
            "x": 102.44,
            "y": 127.55
        },
        "eye_left_eyelid_upper_7": {
            "x": 106.7,
            "y": 128.58
        },
        "eye_left_eyelid_lower_7": {
            "x": 106.85,
            "y": 130.65
        },
        "eye_left_eyelid_lower_5": {
            "x": 103.28,
            "y": 132.02
        },
        "eye_left_eyelid_lower_3": {
            "x": 99.17,
            "y": 132.65
        },
        "eye_left_eyelid_lower_1": {
            "x": 95.15,
            "y": 132.56
        },
        "eye_left_eyeball_right": {
            "x": 97.24,
            "y": 130.52
        },
        "eye_left_eyeball_left": {
            "x": 103.95,
            "y": 129.67
        },
        "nose_bridge_1": {
            "x": 81.94,
            "y": 134.24
```

```
        },
        "nose_bridge_2": {
            "x": 83.08,
            "y": 141.89
        },
        "nose_bridge_3": {
            "x": 84.29,
            "y": 149.66
        },
        "nose_right_contour_5": {
            "x": 74.56,
            "y": 161.68
        },
        "nose_right_contour_7": {
            "x": 77.06,
            "y": 158.52
        },
        "nose_left_contour_7": {
            "x": 92.21,
            "y": 157.05
        },
        "nose_left_contour_5": {
            "x": 94.67,
            "y": 159.82
        },
        "nose_middle_contour": {
            "x": 85.35,
            "y": 163.03
        },
        "mouth_corner_right_inner": {
            "x": 68.55,
            "y": 172.8
        },
        "mouth_corner_left_inner": {
            "x": 100.4,
            "y": 169.56
        },
        "mouth_lip_upper_outer_1": {
            "x": 70.25,
            "y": 171.58
        },
        "mouth_lip_upper_outer_2": {
            "x": 73.42,
            "y": 170.96
        },
        "mouth_lip_upper_outer_4": {
            "x": 79.66,
            "y": 169.94
        },
        "mouth_lip_upper_outer_5": {
            "x": 82.54,
            "y": 169.79
        },
        "mouth_lip_upper_outer_7": {
            "x": 88.39,
            "y": 169.16
        },
        "mouth_lip_upper_outer_8": {
            "x": 91.02,
            "y": 168.74
        },
        "mouth_lip_upper_outer_10": {
```

```
            "x": 96.49,
            "y": 168.52
        },
        "mouth_lip_upper_outer_11": {
            "x": 99.01,
            "y": 168.64
        },
        "mouth_lip_lower_outer_11": {
            "x": 100.04,
            "y": 172.11
        },
        "mouth_lip_lower_outer_10": {
            "x": 98.33,
            "y": 174.75
        },
        "mouth_lip_lower_outer_8": {
            "x": 93.52,
            "y": 178.89
        },
        "mouth_lip_lower_outer_7": {
            "x": 90.48,
            "y": 180.17
        },
        "mouth_lip_lower_outer_5": {
            "x": 83.41,
            "y": 180.88
        },
        "mouth_lip_lower_outer_4": {
            "x": 79.76,
            "y": 180.32
        },
        "mouth_lip_lower_outer_2": {
            "x": 73.07,
            "y": 177.29
        },
        "mouth_lip_lower_outer_1": {
            "x": 70.05,
            "y": 175.12
        },
        "mouth_lip_upper_inner_1": {
            "x": 70.62,
            "y": 172.43
        },
        "mouth_lip_upper_inner_2": {
            "x": 73.67,
            "y": 172.31
        },
        "mouth_lip_upper_inner_4": {
            "x": 79.9,
            "y": 172.08
        },
        "mouth_lip_upper_inner_5": {
            "x": 82.79,
            "y": 171.97
        },
        "mouth_lip_upper_inner_7": {
            "x": 88.55,
            "y": 171.27
        },
        "mouth_lip_upper_inner_8": {
            "x": 91.2,
            "y": 170.78
        },
```

```
                "mouth_lip_upper_inner_10": {
                    "x": 96.53,
                    "y": 169.86
                },
                "mouth_lip_upper_inner_11": {
                    "x": 98.81,
                    "y": 169.52
                },
                "mouth_lip_lower_inner_11": {
                    "x": 99.1,
                    "y": 171.05
                },
                "mouth_lip_lower_inner_10": {
                    "x": 96.9,
                    "y": 172.53
                },
                "mouth_lip_lower_inner_8": {
                    "x": 91.76,
                    "y": 174.86
                },
                "mouth_lip_lower_inner_7": {
                    "x": 89.21,
                    "y": 175.55
                },
                "mouth_lip_lower_inner_5": {
                    "x": 83.42,
                    "y": 176.18
                },
                "mouth_lip_lower_inner_4": {
                    "x": 80.45,
                    "y": 176.03
                },
                "mouth_lip_lower_inner_2": {
                    "x": 73.9,
                    "y": 174.92
                },
                "mouth_lip_lower_inner_1": {
                    "x": 70.65,
                    "y": 173.95
                }
            }
        }
    ]
  }
}
```


- 201点关键点返回示例


```
{
 "error_code": 0,
 "error_msg": "SUCCESS",
 "log_id": 1928749930,
 "timestamp": 1587886328,
 "cached": 0,
 "result": {
    "face_num": 1,
    "face_list": [
        {
            "face_token": "d80fc864a9f11ca97f753b7fd6ef3092",
            "location": {
                "left": 148.2,
```

```
            "top": 207.76,
            "width": 293,
            "height": 265,
            "rotation": 0
        },
        "face_probability": 1,
        "angle": {
            "yaw": 19.13,
            "pitch": 17.13,
            "roll": -3.5
        },
        "landmark201": {
            "cheek_right_1": {
                "x": 42.31241607666,
                "y": 189.65449523926
            },
            "cheek_right_3": {
                "x": 44.54296875,
                "y": 197.21664428711
            },
            "cheek_right_5": {
                "x": 47.168300628662,
                "y": 204.71488952637
            },
            "cheek_right_7": {
                "x": 51.103427886963,
                "y": 212.02030944824
            },
            "cheek_right_9": {
                "x": 58.585376739502,
                "y": 217.99108886719
            },
            "cheek_right_11": {
                "x": 66.981811523438,
                "y": 220.88842773438
            },
            "chin_2": {
                "x": 74.937934875488,
                "y": 221.29644775391
            },
            "cheek_left_11": {
                "x": 81.538116455078,
                "y": 218.27543640137
            },
            "cheek_left_9": {
                "x": 86.794311523438,
                "y": 212.35699462891
            },
            "cheek_left_7": {
                "x": 89.269142150879,
                "y": 205.04515075684
            },
            "cheek_left_5": {
                "x": 89.34553527832,
                "y": 198.01556396484
            },
            "cheek_left_3": {
                "x": 89.059379577637,
                "y": 191.10192871094
            },
            "cheek_left_1": {
                "x": 88.503395080566,
                "y": 184.33065795898
```

```
        "y": 184.33065795898
    },
    "eye_right_corner_right": {
        "x": 55.116683959961,
        "y": 190.61100769043
    },
    "eye_right_eyelid_upper_2": {
        "x": 56.738288879395,
        "y": 188.59680175781
    },
    "eye_right_eyelid_upper_4": {
        "x": 59.01921081543,
        "y": 187.68321228027
    },
    "eye_right_eyelid_upper_6": {
        "x": 61.467914581299,
        "y": 188.02760314941
    },
    "eye_right_corner_left": {
        "x": 63.568134307861,
        "y": 189.63510131836
    },
    "eye_right_eyelid_lower_6": {
        "x": 61.694267272949,
        "y": 190.69076538086
    },
    "eye_right_eyelid_lower_4": {
        "x": 59.454444885254,
        "y": 191.33895874023
    },
    "eye_right_eyelid_lower_2": {
        "x": 57.118618011475,
        "y": 191.25410461426
    },
    "eye_right_eyeball_center": {
        "x": 58.797183990479,
        "y": 189.35336303711
    },
    "eyebrow_right_corner_right": {
        "x": 51.216777801514,
        "y": 188.38786315918
    },
    "eyebrow_right_upper_2": {
        "x": 53.870964050293,
        "y": 184.83016967773
    },
    "eyebrow_right_upper_3": {
        "x": 57.506542205811,
        "y": 183.5043182373
    },
    "eyebrow_right_upper_4": {
        "x": 61.183086395264,
        "y": 183.19314575195
    },
    "eyebrow_right_corner_left": {
        "x": 64.871238708496,
        "y": 184.73831176758
    },
    "eyebrow_right_lower_3": {
        "x": 61.479015350342,
        "y": 185.56793212891
    },
    "eyebrow_right_lower_2": {
```

```
            "x": 58.032566070557,
            "y": 186.2632598877
        },
        "eyebrow_right_lower_1": {
            "x": 54.63260269165,
            "y": 187.36543273926
        },
        "eye_left_corner_right": {
            "x": 74.069404602051,
            "y": 187.93475341797
        },
        "eye_left_eyelid_upper_2": {
            "x": 75.66828918457,
            "y": 186.65887451172
        },
        "eye_left_eyelid_upper_4": {
            "x": 77.592361450195,
            "y": 186.16188049316
        },
        "eye_left_eyelid_upper_6": {
            "x": 79.586471557617,
            "y": 186.45025634766
        },
        "eye_left_corner_left": {
            "x": 81.423141479492,
            "y": 187.45210266113
        },
        "eye_left_eyelid_lower_6": {
            "x": 79.689315795898,
            "y": 188.07566833496
        },
        "eye_left_eyelid_lower_4": {
            "x": 77.799751281738,
            "y": 188.36500549316
        },
        "eye_left_eyelid_lower_2": {
            "x": 75.900672912598,
            "y": 188.30387878418
        },
        "eye_left_eyeball_center": {
            "x": 77.943511962891,
            "y": 187.28001403809
        },
        "eyebrow_left_corner_right": {
            "x": 72.480133056641,
            "y": 183.96446228027
        },
        "eyebrow_left_upper_2": {
            "x": 75.453918457031,
            "y": 181.56910705566
        },
        "eyebrow_left_upper_3": {
            "x": 78.83179473877,
            "y": 181.00735473633
        },
        "eyebrow_left_upper_4": {
            "x": 82.155281066895,
            "y": 181.49905395508
        },
        "eyebrow_left_corner_left": {
            "x": 84.791572570801,
            "y": 184.30467224121
```

```
        },
        "eyebrow_left_lower_3": {
            "x": 82.042182922363,
            "y": 184.03210449219
        },
        "eyebrow_left_lower_2": {
            "x": 79.002876281738,
            "y": 183.73754882812
        },
        "eyebrow_left_lower_1": {
            "x": 75.751091003418,
            "y": 183.88179016113
        },
        "nose_right_contour_1": {
            "x": 66.593109130859,
            "y": 189.06278991699
        },
        "nose_right_contour_2": {
            "x": 66.620910644531,
            "y": 192.60729980469
        },
        "nose_right_contour_3": {
            "x": 66.599555969238,
            "y": 196.11846923828
        },
        "nose_right_contour_4": {
            "x": 65.618698120117,
            "y": 200.2027130127
        },
        "nose_right_contour_6": {
            "x": 68.844802856445,
            "y": 200.28895568848
        },
        "nose_left_contour_6": {
            "x": 74.86922454834,
            "y": 199.59733581543
        },
        "nose_left_contour_4": {
            "x": 77.114517211914,
            "y": 198.64364624023
        },
        "nose_left_contour_3": {
            "x": 74.99666595459,
            "y": 194.87409973145
        },
        "nose_left_contour_2": {
            "x": 73.466331481934,
            "y": 191.60952758789
        },
        "nose_left_contour_1": {
            "x": 71.929557800293,
            "y": 188.32957458496
        },
        "nose_tip": {
            "x": 72.110206604004,
            "y": 197.61848449707
        },
        "mouth_corner_right_outer": {
            "x": 64.238929748535,
            "y": 208.59043884277
        },
        "mouth_lip_upper_outer_3": {
```

```
            "x": 68.3440704345,
            "y": 206.58200073242
        },
        "mouth_lip_upper_outer_6": {
            "x": 72.647048950195,
            "y": 205.95881652832
        },
        "mouth_lip_upper_outer_9": {
            "x": 76.58984375,
            "y": 205.54756164551
        },
        "mouth_corner_left_outer": {
            "x": 80.137763977051,
            "y": 206.63317871094
        },
        "mouth_lip_lower_outer_9": {
            "x": 77.228042602539,
            "y": 209.13989257812
        },
        "mouth_lip_lower_outer_6": {
            "x": 73.251800537109,
            "y": 210.38688659668
        },
        "mouth_lip_lower_outer_3": {
            "x": 68.685707092285,
            "y": 210.19580078125
        },
        "mouth_lip_upper_inner_3": {
            "x": 68.588790893555,
            "y": 207.86560058594
        },
        "mouth_lip_upper_inner_6": {
            "x": 72.813514709473,
            "y": 207.4914855957
        },
        "mouth_lip_upper_inner_9": {
            "x": 76.605278015137,
            "y": 206.88005065918
        },
        "mouth_lip_lower_inner_9": {
            "x": 76.632255554199,
            "y": 207.44100952148
        },
        "mouth_lip_lower_inner_6": {
            "x": 72.939880371094,
            "y": 208.07493591309
        },
        "mouth_lip_lower_inner_3": {
            "x": 68.787719726562,
            "y": 208.41091918945
        },
        "cheek_right_2": {
            "x": 43.467124938965,
            "y": 193.40245056152
        },
        "cheek_right_4": {
            "x": 45.808784484863,
            "y": 200.98356628418
        },
        "cheek_right_6": {
            "x": 48.872074127197,
            "y": 208.6109161377
        },
```

```
        },
        "cheek_right_8": {
            "x": 54.450130462646,
            "y": 215.4328918457
        },
        "cheek_right_10": {
            "x": 62.691562652588,
            "y": 219.77143859863
        },
        "chin_1": {
            "x": 70.906677246094,
            "y": 221.53775024414
        },
        "chin_3": {
            "x": 78.58708190918,
            "y": 220.25169372559
        },
        "cheek_left_10": {
            "x": 84.438705444336,
            "y": 215.54095458984
        },
        "cheek_left_8": {
            "x": 88.421768188477,
            "y": 208.86682128906
        },
        "cheek_left_6": {
            "x": 89.404579162598,
            "y": 201.58218383789
        },
        "cheek_left_4": {
            "x": 89.226097106934,
            "y": 194.56100463867
        },
        "cheek_left_2": {
            "x": 88.776916503906,
            "y": 187.70640563965
        },
        "eyebrow_right_upper_1": {
            "x": 51.055225372314,
            "y": 187.41928100586
        },
        "eyebrow_right_upper_5": {
            "x": 64.620422363281,
            "y": 183.30424499512
        },
        "eyebrow_left_upper_1": {
            "x": 72.405311584473,
            "y": 182.5030670166
        },
        "eyebrow_left_upper_5": {
            "x": 84.738502502441,
            "y": 183.29322814941
        },
        "eye_right_eyelid_upper_1": {
            "x": 55.766471862793,
            "y": 189.49603271484
        },
        "eye_right_eyelid_upper_3": {
            "x": 57.783981323242,
            "y": 187.9764251709
        },
        "eye_right_eyelid_upper_5": {
            "x": 60.291412353516,
```

```
            "y": 187.6668548584
        },
        "eye_right_eyelid_upper_7": {
            "x": 62.610542297363,
            "y": 188.70217895508
        },
        "eye_right_eyelid_lower_7": {
            "x": 62.647716522217,
            "y": 190.19287109375
        },
        "eye_right_eyelid_lower_5": {
            "x": 60.609279632568,
            "y": 191.1389465332
        },
        "eye_right_eyelid_lower_3": {
            "x": 58.271293640137,
            "y": 191.43981933594
        },
        "eye_right_eyelid_lower_1": {
            "x": 56.100021362305,
            "y": 190.99014282227
        },
        "eye_right_eyeball_right": {
            "x": 56.948715209961,
            "y": 189.78875732422
        },
        "eye_right_eyeball_left": {
            "x": 60.76294708252,
            "y": 189.30876159668
        },
        "eye_left_eyelid_upper_1": {
            "x": 74.795600891113,
            "y": 187.22235107422
        },
        "eye_left_eyelid_upper_3": {
            "x": 76.582481384277,
            "y": 186.30328369141
        },
        "eye_left_eyelid_upper_5": {
            "x": 78.627487182617,
            "y": 186.19665527344
        },
        "eye_left_eyelid_upper_7": {
            "x": 80.57470703125,
            "y": 186.87817382812
        },
        "eye_left_eyelid_lower_7": {
            "x": 80.591354370117,
            "y": 187.83154296875
        },
        "eye_left_eyelid_lower_5": {
            "x": 78.77872467041,
            "y": 188.32012939453
        },
        "eye_left_eyelid_lower_3": {
            "x": 76.835243225098,
            "y": 188.42440795898
        },
        "eye_left_eyelid_lower_1": {
            "x": 74.967391967773,
            "y": 188.18258666992
        },
```

```
        "eye_left_eyeball_right": {
            "x": 76.36678314209,
            "y": 187.48699951172
        },
        "eye_left_eyeball_left": {
            "x": 79.485054016113,
            "y": 187.31430053711
        },
        "nose_bridge_1": {
            "x": 69.695106506348,
            "y": 188.73384094238
        },
        "nose_bridge_2": {
            "x": 70.618003845215,
            "y": 192.09872436523
        },
        "nose_bridge_3": {
            "x": 71.556343078613,
            "y": 195.47486877441
        },
        "nose_right_contour_5": {
            "x": 67.552391052246,
            "y": 201.55177307129
        },
        "nose_right_contour_7": {
            "x": 68.642906188965,
            "y": 199.08920288086
        },
        "nose_left_contour_7": {
            "x": 74.935028076172,
            "y": 198.34973144531
        },
        "nose_left_contour_5": {
            "x": 75.971466064453,
            "y": 200.47100830078
        },
        "nose_middle_contour": {
            "x": 72.127105712891,
            "y": 201.73022460938
        },
        "mouth_corner_right_inner": {
            "x": 64.899894714355,
            "y": 208.53944396973
        },
        "mouth_corner_left_inner": {
            "x": 79.593727111816,
            "y": 206.72438049316
        },
        "mouth_lip_upper_outer_1": {
            "x": 65.49520111084,
            "y": 207.76577758789
        },
        "mouth_lip_upper_outer_2": {
            "x": 66.83455657959,
            "y": 207.09317016602
        },
        "mouth_lip_upper_outer_4": {
            "x": 69.727005004883,
            "y": 206.0223236084
        },
        "mouth_lip_upper_outer_5": {
            "x": 71.136360168457,
            "y": 205.85467529297
```

```
            "y": 205.85467529297
        },
        "mouth_lip_upper_outer_7": {
            "x": 74.009178161621,
            "y": 205.50805664062
        },
        "mouth_lip_upper_outer_8": {
            "x": 75.26887512207,
            "y": 205.34454345703
        },
        "mouth_lip_upper_outer_10": {
            "x": 77.930633544922,
            "y": 205.73402404785
        },
        "mouth_lip_upper_outer_11": {
            "x": 79.06810760498,
            "y": 206.10731506348
        },
        "mouth_lip_lower_outer_11": {
            "x": 79.310043334961,
            "y": 207.62538146973
        },
        "mouth_lip_lower_outer_10": {
            "x": 78.422233581543,
            "y": 208.49403381348
        },
        "mouth_lip_lower_outer_8": {
            "x": 76.039436340332,
            "y": 209.91091918945
        },
        "mouth_lip_lower_outer_7": {
            "x": 74.759254455566,
            "y": 210.31077575684
        },
        "mouth_lip_lower_outer_5": {
            "x": 71.684158325195,
            "y": 210.71028137207
        },
        "mouth_lip_lower_outer_4": {
            "x": 70.194778442383,
            "y": 210.64688110352
        },
        "mouth_lip_lower_outer_2": {
            "x": 67.074501037598,
            "y": 209.91828918457
        },
        "mouth_lip_lower_outer_1": {
            "x": 65.643745422363,
            "y": 209.32096862793
        },
        "mouth_lip_upper_inner_1": {
            "x": 65.730621337891,
            "y": 208.26277160645
        },
        "mouth_lip_upper_inner_2": {
            "x": 67.079795837402,
            "y": 208.03302001953
        },
        "mouth_lip_upper_inner_4": {
            "x": 70.023719787598,
            "y": 207.62254333496
        },
        "mouth_lip_upper_inner_5": {
```

```
            "x": 71.351684570312,
            "y": 207.49478149414
        },
        "mouth_lip_upper_inner_7": {
            "x": 74.106018066406,
            "y": 207.1644744873
        },
        "mouth_lip_upper_inner_8": {
            "x": 75.296127319336,
            "y": 206.98078918457
        },
        "mouth_lip_upper_inner_10": {
            "x": 77.856018066406,
            "y": 206.71319580078
        },
        "mouth_lip_upper_inner_11": {
            "x": 78.921539306641,
            "y": 206.64031982422
        },
        "mouth_lip_lower_inner_11": {
            "x": 78.975936889648,
            "y": 207.04075622559
        },
        "mouth_lip_lower_inner_10": {
            "x": 77.91145324707,
            "y": 207.22946166992
        },
        "mouth_lip_lower_inner_8": {
            "x": 75.387634277344,
            "y": 207.6734161377
        },
        "mouth_lip_lower_inner_7": {
            "x": 74.222236633301,
            "y": 207.86814880371
        },
        "mouth_lip_lower_inner_5": {
            "x": 71.497695922852,
            "y": 208.21778869629
        },
        "mouth_lip_lower_inner_4": {
            "x": 70.19393157959,
            "y": 208.31753540039
        },
        "mouth_lip_lower_inner_2": {
            "x": 67.215522766113,
            "y": 208.55447387695
        },
        "mouth_lip_lower_inner_1": {
            "x": 65.80793762207,
            "y": 208.67227172852
        },
        "iris_left_1": {
            "x": 58.5608253479,
            "y": 188.5970916748
        },
        "iris_left_2": {
            "x": 58.339405059814,
            "y": 186.68266296387
        },
        "iris_left_3": {
            "x": 57.866024017334,
            "y": 186.83006286621
```

```
        },
        "iris_left_4": {
            "x": 57.384708404541,
            "y": 187.18560791016
        },
        "iris_left_5": {
            "x": 57.09496307373,
            "y": 187.73175048828
        },
        "iris_left_6": {
            "x": 57.025074005127,
            "y": 188.2507019043
        },
        "iris_left_7": {
            "x": 57.108268737793,
            "y": 188.8582611084
        },
        "iris_left_8": {
            "x": 57.340709686279,
            "y": 189.41612243652
        },
        "iris_left_9": {
            "x": 57.619876861572,
            "y": 189.85971069336
        },
        "iris_left_10": {
            "x": 57.992046356201,
            "y": 190.19557189941
        },
        "iris_left_11": {
            "x": 58.506317138672,
            "y": 190.3910369873
        },
        "iris_left_12": {
            "x": 59.06840133667,
            "y": 190.31591796875
        },
        "iris_left_13": {
            "x": 59.484573364258,
            "y": 190.00793457031
        },
        "iris_left_14": {
            "x": 59.801601409912,
            "y": 189.56636047363
        },
        "iris_left_15": {
            "x": 59.937400817871,
            "y": 189.02279663086
        },
        "iris_left_16": {
            "x": 59.994979858398,
            "y": 188.47799682617
        },
        "iris_left_17": {
            "x": 59.908092498779,
            "y": 187.91714477539
        },
        "iris_left_18": {
            "x": 59.693229675293,
            "y": 187.42846679688
        },
        "iris_left_19": {
```

```
            "x": 59.357315063477,
            "y": 186.98899841309
        },
        "iris_left_20": {
            "x": 58.83500289917,
            "y": 186.74018859863
        },
        "iris_right_1": {
            "x": 77.913948059082,
            "y": 186.52035522461
        },
        "iris_right_2": {
            "x": 77.751182556152,
            "y": 184.79627990723
        },
        "iris_right_3": {
            "x": 77.291893005371,
            "y": 184.87088012695
        },
        "iris_right_4": {
            "x": 76.81241607666,
            "y": 185.17738342285
        },
        "iris_right_5": {
            "x": 76.51399230957,
            "y": 185.63529968262
        },
        "iris_right_6": {
            "x": 76.399871826172,
            "y": 186.14614868164
        },
        "iris_right_7": {
            "x": 76.425010681152,
            "y": 186.71411132812
        },
        "iris_right_8": {
            "x": 76.59375,
            "y": 187.21925354004
        },
        "iris_right_9": {
            "x": 76.89958190918,
            "y": 187.63000488281
        },
        "iris_right_10": {
            "x": 77.306015014648,
            "y": 187.93461608887
        },
        "iris_right_11": {
            "x": 77.76798248291,
            "y": 188.10162353516
        },
        "iris_right_12": {
            "x": 78.355964660645,
            "y": 188.07786560059
        },
        "iris_right_13": {
            "x": 78.838737487793,
            "y": 187.83233642578
        },
        "iris_right_14": {
            "x": 79.188194274902,
            "y": 187.45365905762
        },
```

```
        "iris_right_15": {
            "x": 79.389739990234,
            "y": 186.96867370605
        },
        "iris_right_16": {
            "x": 79.451263427734,
            "y": 186.44125366211
        },
        "iris_right_17": {
            "x": 79.35481262207,
            "y": 185.87980651855
        },
        "iris_right_18": {
            "x": 79.112213134766,
            "y": 185.42335510254
        },
        "iris_right_19": {
            "x": 78.747856140137,
            "y": 185.04452514648
        },
        "iris_right_20": {
            "x": 78.234527587891,
            "y": 184.81198120117
        },
        "forehead_center": {
            "x": 63.854957580566,
            "y": 161.62615966797
        },
        "forehead_right_1": {
            "x": 56.716361999512,
            "y": 162.98860168457
        },
        "forehead_right_2": {
            "x": 50.169933319092,
            "y": 165.94499206543
        },
        "forehead_right_3": {
            "x": 45.235385894775,
            "y": 171.05480957031
        },
        "forehead_right_4": {
            "x": 42.774684906006,
            "y": 177.78215026855
        },
        "forehead_right_5": {
            "x": 42.371360778809,
            "y": 185.02729797363
        },
        "forehead_left_1": {
            "x": 70.29483795166,
            "y": 161.40893554688
        },
        "forehead_left_2": {
            "x": 76.429832458496,
            "y": 163.07518005371
        },
        "forehead_left_3": {
            "x": 81.494247436523,
            "y": 166.90705871582
        },
        "forehead_left_4": {
            "x": 85.017692565918,
```

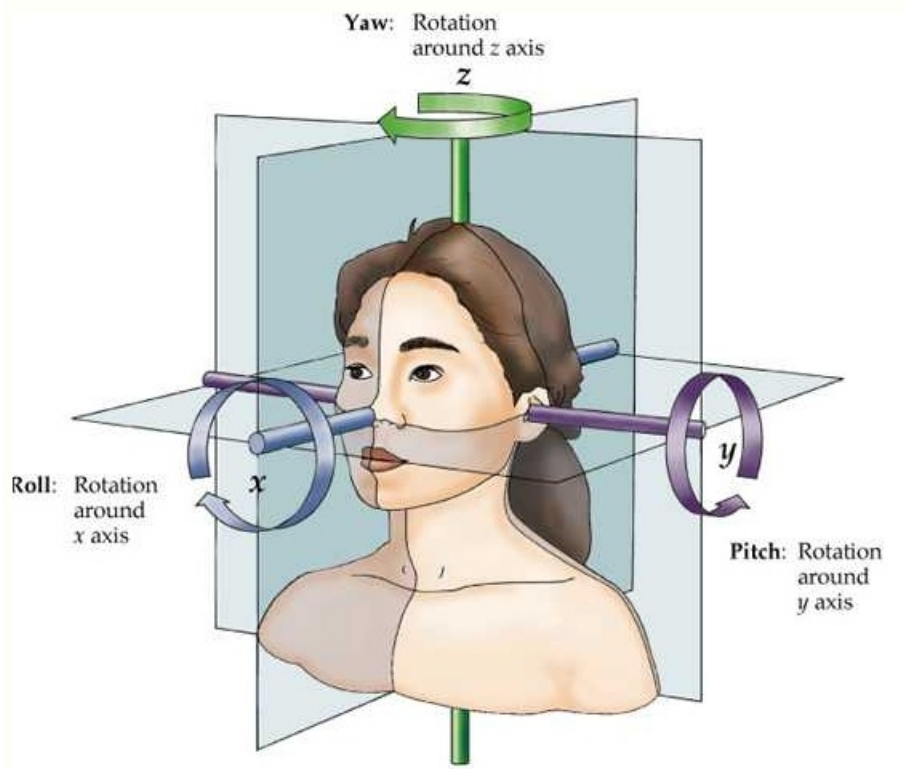
```
                "y": 172.2216796875
            },
            "forehead_left_5": {
                "x": 87.215446472168,
                "y": 178.24891662598
            }
        }
    }
  ]
 }
}
```


人脸空间姿态角参考

姿态角分为Pitch、Roll、Yaw，用于表示人脸在空间三维坐标系内的角度，常用于判断识别角度的界限值。 各角度阈值如下：

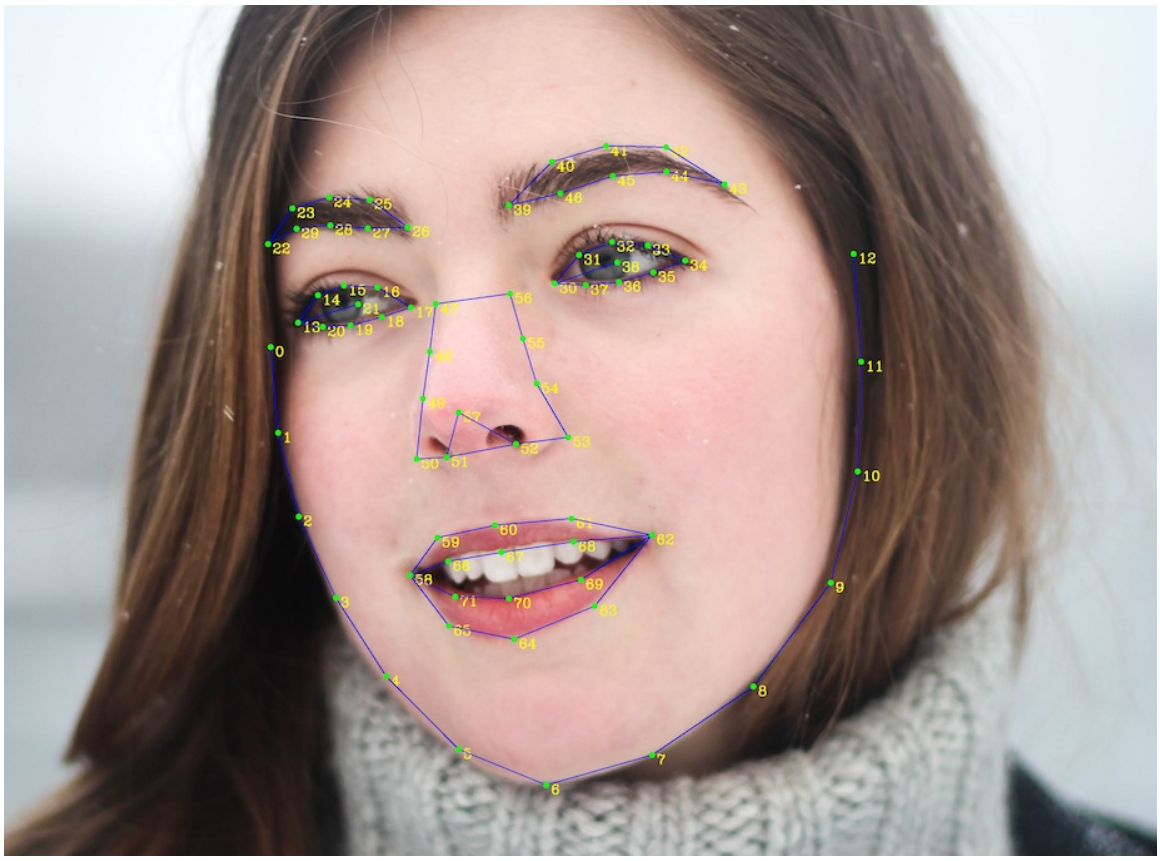
- Pitch：三维旋转之俯仰角度，范围：[-90（上），90（下）]，推荐俯仰角绝对值不大于20度；
- Roll：平面内旋转角，范围：[-180（逆时针），180（顺时针）]，推荐旋转角绝对值不大于20度；
- Yaw：三维旋转之左右旋转角，范围：[-90（左），90（右）]，推荐旋转角绝对值不大于20度；

各角度范围示意图如下：



示意图

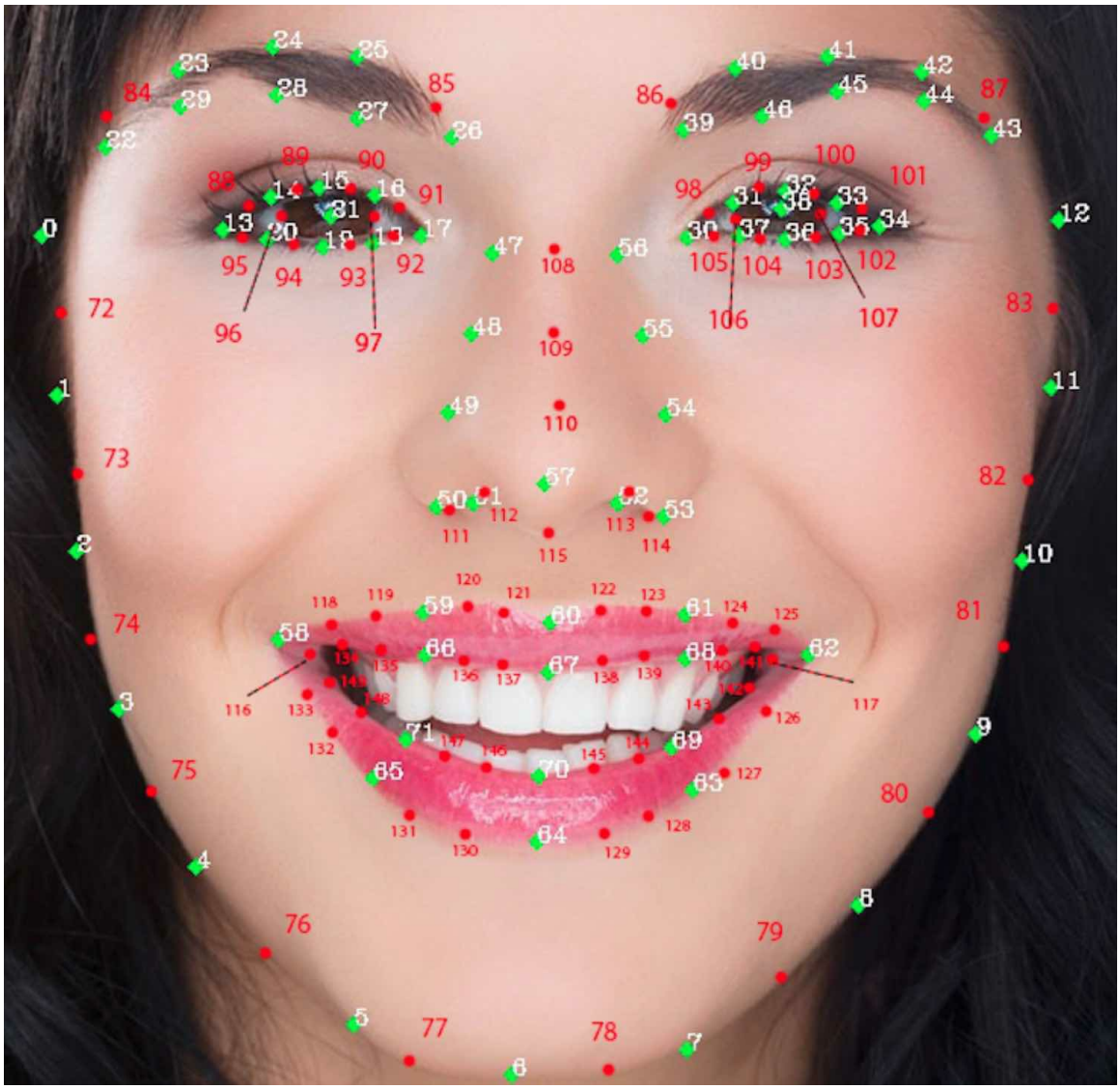
72个关键点示意图



对应landmark72个点的顺序，序号从0-71，且每个关键点有对应的英文命名作为参数名；关键点名称对应关系如下图

序号	命名	序号	命名
0	cheek_right_1	41	eyebrow_left_upper_3
1	cheek_right_3	42	eyebrow_left_upper_4
2	cheek_right_5	43	eyebrow_left_corner_left
3	cheek_right_7	44	eyebrow_left_lower_3
4	cheek_right_9	45	eyebrow_left_lower_2
5	cheek_right_11	46	eyebrow_left_lower_1
6	chin_2	47	nose_right_contour_1
7	cheek_left_11	48	nose_right_contour_2
8	cheek_left_9	49	nose_right_contour_3
9	cheek_left_7	50	nose_right_contour_4
10	cheek_left_5	51	nose_right_contour_6
11	cheek_left_3	52	nose_left_contour_6
12	cheek_left_1	53	nose_left_contour_4
13	eye_right_corner_right	54	nose_left_contour_3
14	eye_right_eyelid_upper_2	55	nose_left_contour_2
15	eye_right_eyelid_upper_4	56	nose_left_contour_1
16	eye_right_eyelid_upper_6	57	nose_tip
17	eye_right_corner_left	58	mouth_corner_right_outer
18	eye_right_eyelid_lower_6	59	mouth_lip_upper_outer_3
19	eye_right_eyelid_lower_4	60	mouth_lip_upper_outer_6
20	eye_right_eyelid_lower_2	61	mouth_lip_upper_outer_9
21	eye_right_eyeball_center	62	mouth_corner_left_outer
22	eyebrow_right_corner_right	63	mouth_lip_lower_outer_9
23	eyebrow_right_upper_2	64	mouth_lip_lower_outer_6
24	eyebrow_right_upper_3	65	mouth_lip_lower_outer_3
25	eyebrow_right_upper_4	66	mouth_lip_upper_inner_3
26	eyebrow_right_corner_left	67	mouth_lip_upper_inner_6
27	eyebrow_right_lower_3	68	mouth_lip_upper_inner_9
28	eyebrow_right_lower_2	69	mouth_lip_lower_inner_9
29	eyebrow_right_lower_1	70	mouth_lip_lower_inner_6
30	eye_left_corner_right	71	mouth_lip_lower_inner_3
31	eye_left_eyelid_upper_2	72	
32	eye_left_eyelid_upper_4	73	
33	eye_left_eyelid_upper_6	74	
34	eye_left_corner_left	75	
35	eye_left_eyelid_lower_6	76	
36	eye_left_eyelid_lower_4	77	
37	eye_left_eyelid_lower_2	78	
38	eye_left_eyeball_center	79	
39	eyebrow_left_corner_right	80	
40	eyebrow_left_upper_2	81	

150个关键点示意图



对应landmark150个点的顺序，序号从0-149，且每个关键点有对应的英文命名作为参数名；关键点名称对应关系如下图

序号	命名	序号	命名	序号	命名	序号	命名	序号	命名
0	cheek_right_1	31	eye_left_eyelid_upper_2	61	mouth_lip_upper_outer_9	91	eye_right_eyelid_upper_7	121	mouth_lip_upper_outer_5
1	cheek_right_3	32	eye_left_eyelid_upper_4	62	mouth_corner_left_outer	92	eye_right_eyelid_lower_7	122	mouth_lip_upper_outer_7
2	cheek_right_5	33	eye_left_eyelid_upper_6	63	mouth_lip_lower_outer_9	93	eye_right_eyelid_lower_5	123	mouth_lip_upper_outer_8
3	cheek_right_7	34	eye_left_corner_left	64	mouth_lip_lower_outer_6	94	eye_right_eyelid_lower_3	124	mouth_lip_upper_outer_10
4	cheek_right_9	35	eye_left_eyelid_lower_6	65	mouth_lip_lower_outer_3	95	eye_right_eyelid_lower_1	125	mouth_lip_upper_outer_11
5	cheek_right_11	36	eye_left_eyelid_lower_4	66	mouth_lip_upper_inner_3	96	eye_right_eyeball_right	126	mouth_lip_lower_outer_11
6	chin_2	37	eye_left_eyelid_lower_2	67	mouth_lip_upper_inner_6	97	eye_right_eyeball_left	127	mouth_lip_lower_outer_10
7	cheek_left_11	38	eye_left_eyeball_center	68	mouth_lip_upper_inner_9	98	eye_left_eyelid_upper_1	128	mouth_lip_lower_outer_8
8	cheek_left_9	39	eyebrow_left_corner_right	69	mouth_lip_lower_inner_9	99	eye_left_eyelid_upper_3	129	mouth_lip_lower_outer_7
9	cheek_left_7	40	eyebrow_left_upper_2	70	mouth_lip_lower_inner_6	100	eye_left_eyelid_upper_5	130	mouth_lip_lower_outer_5
10	cheek_left_5	41	eyebrow_left_upper_3	71	mouth_lip_lower_inner_3	101	eye_left_eyelid_upper_7	131	mouth_lip_lower_outer_4
11	cheek_left_3	42	eyebrow_left_upper_4	72	cheek_right_2	102	eye_left_eyelid_lower_7	132	mouth_lip_lower_outer_2
12	cheek_left_1	43	eyebrow_left_corner_left	73	cheek_right_4	103	eye_left_eyelid_lower_5	133	mouth_lip_lower_outer_1
13	eye_right_corner_right	44	eyebrow_left_lower_3	74	cheek_right_6	104	eye_left_eyelid_lower_3	134	mouth_lip_upper_inner_1
14	eye_right_eyelid_upper_2	45	eyebrow_left_lower_2	75	cheek_right_8	105	eye_left_eyelid_lower_1	135	mouth_lip_upper_inner_2
15	eye_right_eyelid_upper_4	46	eyebrow_left_lower_1	76	cheek_right_10	106	eye_left_eyeball_right	136	mouth_lip_upper_inner_4
16	eye_right_eyelid_upper_6	47	nose_right_contour_1	77	chin_1	107	eye_left_eyeball_left	137	mouth_lip_upper_inner_5
17	eye_right_corner_left	48	nose_right_contour_2	78	chin_3	108	nose_bridge_1	138	mouth_lip_upper_inner_7
18	eye_right_eyelid_lower_6	49	nose_right_contour_3	79	cheek_left_10	109	nose_bridge_2	139	mouth_lip_upper_inner_8
19	eye_right_eyelid_lower_4	50	nose_right_contour_4	80	cheek_left_8	110	nose_bridge_3	140	mouth_lip_upper_inner_10
20	eye_right_eyelid_lower_2	51	nose_right_contour_6	81	cheek_left_6	111	nose_right_contour_5	141	mouth_lip_upper_inner_11
21	eye_right_eyeball_center	52	nose_left_contour_6	82	cheek_left_4	112	nose_right_contour_7	142	mouth_lip_lower_inner_11
22	eyebrow_right_corner_right	53	nose_left_contour_4	83	cheek_left_2	113	nose_left_contour_7	143	mouth_lip_lower_inner_10
23	eyebrow_right_upper_2	54	nose_left_contour_3	84	eyebrow_right_upper_1	114	nose_left_contour_5	144	mouth_lip_lower_inner_8
24	eyebrow_right_upper_3	55	nose_left_contour_2	85	eyebrow_right_upper_5	115	nose_middle_contour	145	mouth_lip_lower_inner_7
25	eyebrow_right_upper_4	56	nose_left_contour_1	86	eyebrow_left_upper_1	116	mouth_corner_right_inner	146	mouth_lip_lower_inner_5
26	eyebrow_right_corner_left	57	nose_tip	87	eyebrow_left_upper_5	117	mouth_corner_left_inner	147	mouth_lip_lower_inner_4
27	eyebrow_right_lower_3	58	mouth_corner_right_outer	88	eye_right_eyelid_upper_1	118	mouth_lip_upper_outer_1	148	mouth_lip_lower_inner_2
28	eyebrow_right_lower_2	59	mouth_lip_upper_outer_3	89	eye_right_eyelid_upper_3	119	mouth_lip_upper_outer_2	149	mouth_lip_lower_inner_1
29	eyebrow_right_lower_1	60	mouth_lip_upper_outer_6	90	eye_right_eyelid_upper_5	120	mouth_lip_upper_outer_4		
30	eye_left_corner_right								

201个关键点示意图

对应landmark201个点的顺序，序号从0-200，且每个关键点有对应的英文命名作为参数名；关键点名称对应关系如下图

序号	命名	序号	命名	序号	命名	序号	命名	序号	命名
0	cheek_right_1	41	eyebrow_left_upper_3	82	cheek_left_4	123	mouth_lip_upper_outer_8	164	iris_left_15
1	cheek_right_3	42	eyebrow_left_upper_4	83	cheek_left_2	124	mouth_lip_upper_outer_10	165	iris_left_16
2	cheek_right_5	43	eyebrow_left_corner_left	84	eyebrow_right_upper_1	125	mouth_lip_upper_outer_11	166	iris_left_17
3	cheek_right_7	44	eyebrow_left_lower_3	85	eyebrow_right_upper_5	126	mouth_lip_lower_outer_11	167	iris_left_18
4	cheek_right_9	45	eyebrow_left_lower_2	86	eyebrow_left_upper_1	127	mouth_lip_lower_outer_10	168	iris_left_19
5	cheek_right_11	46	eyebrow_left_lower_1	87	eyebrow_left_upper_5	128	mouth_lip_lower_outer_8	169	iris_left_20
6	chin_2	47	nose_right_contour_1	88	eye_right_eyelid_upper_1	129	mouth_lip_lower_outer_7	170	iris_right_1
7	cheek_left_11	48	nose_right_contour_2	89	eye_right_eyelid_upper_3	130	mouth_lip_lower_outer_5	171	iris_right_2
8	cheek_left_9	49	nose_right_contour_3	90	eye_right_eyelid_upper_5	131	mouth_lip_lower_outer_4	172	iris_right_3
9	cheek_left_7	50	nose_right_contour_4	91	eye_right_eyelid_upper_7	132	mouth_lip_lower_outer_2	173	iris_right_4
10	cheek_left_5	51	nose_right_contour_6	92	eye_right_eyelid_lower_7	133	mouth_lip_lower_outer_1	174	iris_right_5
11	cheek_left_3	52	nose_left_contour_6	93	eye_right_eyelid_lower_5	134	mouth_lip_upper_inner_1	175	iris_right_6
12	cheek_left_1	53	nose_left_contour_4	94	eye_right_eyelid_lower_3	135	mouth_lip_upper_inner_2	176	iris_right_7
13	eye_right_corner_right	54	nose_left_contour_3	95	eye_right_eyelid_lower_1	136	mouth_lip_upper_inner_4	177	iris_right_8
14	eye_right_eyelid_upper_2	55	nose_left_contour_2	96	eye_right_eyeball_right	137	mouth_lip_upper_inner_5	178	iris_right_9
15	eye_right_eyelid_upper_4	56	nose_left_contour_1	97	eye_right_eyeball_left	138	mouth_lip_upper_inner_7	179	iris_right_10
16	eye_right_eyelid_upper_6	57	nose_tip	98	eye_left_eyelid_upper_1	139	mouth_lip_upper_inner_8	180	iris_right_11
17	eye_right_corner_left	58	mouth_corner_right_outer	99	eye_left_eyelid_upper_3	140	mouth_lip_upper_inner_10	181	iris_right_12
18	eye_right_eyelid_lower_6	59	mouth_lip_upper_outer_3	100	eye_left_eyelid_upper_5	141	mouth_lip_upper_inner_11	182	iris_right_13
19	eye_right_eyelid_lower_4	60	mouth_lip_upper_outer_6	101	eye_left_eyelid_upper_7	142	mouth_lip_lower_inner_11	183	iris_right_14
20	eye_right_eyelid_lower_2	61	mouth_lip_upper_outer_9	102	eye_left_eyelid_lower_7	143	mouth_lip_lower_inner_10	184	iris_right_15
21	eye_right_eyeball_center	62	mouth_corner_left_outer	103	eye_left_eyelid_lower_5	144	mouth_lip_lower_inner_8	185	iris_right_16
22	eyebrow_right_corner_right	63	mouth_lip_lower_outer_9	104	eye_left_eyelid_lower_3	145	mouth_lip_lower_inner_7	186	iris_right_17
23	eyebrow_right_upper_2	64	mouth_lip_lower_outer_6	105	eye_left_eyelid_lower_1	146	mouth_lip_lower_inner_5	187	iris_right_18
24	eyebrow_right_upper_3	65	mouth_lip_lower_outer_3	106	eye_left_eyeball_right	147	mouth_lip_lower_inner_4	188	iris_right_19
25	eyebrow_right_upper_4	66	mouth_lip_upper_inner_3	107	eye_left_eyeball_left	148	mouth_lip_lower_inner_2	189	iris_right_20
26	eyebrow_right_corner_left	67	mouth_lip_upper_inner_6	108	nose_bridge_1	149	mouth_lip_lower_inner_1	190	forehead_center
27	eyebrow_right_lower_3	68	mouth_lip_upper_inner_9	109	nose_bridge_2	150	iris_left_1	191	forehead_right_1
28	eyebrow_right_lower_2	69	mouth_lip_lower_inner_9	110	nose_bridge_3	151	iris_left_2	192	forehead_right_2
29	eyebrow_right_lower_1	70	mouth_lip_lower_inner_6	111	nose_right_contour_5	152	iris_left_3	193	forehead_right_3
30	eye_left_corner_right	71	mouth_lip_lower_inner_3	112	nose_right_contour_7	153	iris_left_4	194	forehead_right_4
31	eye_left_eyelid_upper_2	72	cheek_right_2	113	nose_left_contour_7	154	iris_left_5	195	forehead_right_5
32	eye_left_eyelid_upper_4	73	cheek_right_4	114	nose_left_contour_5	155	iris_left_6	196	forehead_left_1
33	eye_left_eyelid_upper_6	74	cheek_right_6	115	nose_middle_contour	156	iris_left_7	197	forehead_left_2
34	eye_left_corner_left	75	cheek_right_8	116	mouth_corner_right_inner	157	iris_left_8	198	forehead_left_3
35	eye_left_eyelid_lower_6	76	cheek_right_10	117	mouth_corner_left_inner	158	iris_left_9	199	forehead_left_4
36	eye_left_eyelid_lower_4	77	chin_1	118	mouth_lip_upper_outer_1	159	iris_left_10	200	forehead_left_5
37	eye_left_eyelid_lower_2	78	chin_3	119	mouth_lip_upper_outer_2	160	iris_left_11		
38	eye_left_eyeball_center	79	cheek_left_10	120	mouth_lip_upper_outer_4	161	iris_left_12		
39	eyebrow_left_corner_right	80	cheek_left_8	121	mouth_lip_upper_outer_5	162	iris_left_13		
40	eyebrow_left_upper_2	81	cheek_left_6	122	mouth_lip_upper_outer_7	163	iris_left_14		

错误码说明

若请求错误，服务器将返回的JSON文本包含以下参数：

- **error_code**：错误码。
- **error_msg**：错误描述信息，帮助理解和解决发生的错误。

例如Access Token失效返回：

```
{
 "error_code": 110,
 "error_msg": "Access token invalid or no longer valid"
}
```

上述问题重新获取新的Access Token再次请求即可。

接口流控及鉴权错误码

错误码	错误信息	描述	处理建议
2	Service temporarily unavailable	服务暂不可用	服务暂不可用，请再次请求，如果持续出现此类错误，请在控制台提交工单联系技术支持团队
4	Open api request limit reached	集群超限额	集群超限额，请再次请求，如果持续出现此类错误，请在控制台提交工单联系技术支持团队
6	no permission to access data	没有接口权限	请确认您调用的接口已经被赋权 常见问题是有V3版本权限，调用的是v2版本接口； 需要企业认证的，开通企业认证后一个小时左右即可使用。
17	Open api daily request limit reached	每天流量超限额	免费测试资源使用完毕，每天请求量超限额，已支持计费的接口，您可以在控制台人脸识别选择购买相关接口的次数包或开通按量后付费；邀测和未支持计费的接口，您可以在控制台提交工单申请提升限额
18	Open api qps request limit reached	QPS超限额	可直接自助购买更多QPS、联系商务接口人、或者提交工单
19	Open api total request limit reached	请求总量超限额	免费测试资源使用完毕，每天请求量超限额，已支持计费的接口，您可以在控制台人脸识别选择购买相关接口的次数包或开通按量后付费；邀测和未支持计费的接口，您可以在控制台提交工单申请提升限额
100	Invalid parameter	无效的access_token参数	token拉取失败，可以参考“Access Token获取”重新获取
110	Access token invalid or no longer valid	Access Token失效	**token有效期为30天，注意需要定期更换**，也可以每次请求都拉取新token
111	Access token expired	Access token过期	**token有效期为30天，注意需要定期更换**，也可以每次请求都拉取新token

通用及业务错误码

错误码	错误信息	说明	处理建议
222001	param[fieldName] is null	必要参数未传入	参考API说明文档，传入必须的fieldName字段
222002	param[start] format error	参数格式错误	参考API说明文档，修改参数
222003	param[length] format error	参数格式错误	参考API说明文档，修改参数

222004	param[op_app_id_list] format error	参数格式错误	参考API说明文档，修改参数
222005	param[group_id_list] format error	参数格式错误	参考API说明文档，修改参数
222006	group_id format error	参数格式错误	参考API说明文档，修改参数
222007	uid format error	参数格式错误	参考API说明文档，修改参数
222008	face_id format error	参数格式错误	参考API说明文档，修改参数
222009	quality_conf format error	参数格式错误	参考API说明文档，修改参数
222010	user_info format error	参数格式错误	参考API说明文档，修改参数
222011	param[uid_list] format error	参数格式错误	参考API说明文档，修改参数
222012	param[op_app_id] format error	参数格式错误	参考API说明文档，修改参数
222013	param[image] format error	参数格式错误	参考API说明文档，修改参数
222014	param[app_id] format error	参数格式错误	参考API说明文档，修改参数
222015	param[image_type] format error	参数格式错误	参考API说明文档，修改参数
222016	param[max_face_num] format error	参数格式错误	参考API说明文档，修改参数
222017	param[face_field] format error	参数格式错误	参考API说明文档，修改参数
222018	param[user_id] format error	参数格式错误	参考API说明文档，修改参数
222019	param[quality_control] format error	参数格式错误	参考API说明文档，修改参数
222020	param[liveness_control] format error	参数格式错误	参考API说明文档，修改参数
222021	param[max_user_num] format error	参数格式错误	参考API说明文档，修改参数
222022	param[id_card_number] format error	身份证号不符合要求，备注：中国大陆身份证号的每一位数字都有其生成规则，不可随意填写。	填写正确的身份证号

222023	param[name] format error	参数格式错误	请检查手机号、姓名、身份证号格式是否正确
222024	param[face_type] format error	参数格式错误	参考API说明文档，修改参数
222025	param[face_token] format error	参数格式错误	参考API说明文档，修改参数
222026	param[max_star_num] format error	参数格式错误	参考API说明文档，修改参数
222027	code length param error	验证码长度错误 (最小值大于最大值)	参考API说明文档，修改参数
222028	param[min_code_length] format error	参数格式错误	参考API说明文档，修改参数
222029	param[max_code_length] format error	参数格式错误	参考API说明文档，修改参数
222030	param[match_threshold] format error	参数格式错误	参考API说明文档，修改参数
222039	param[face_sort_type] format error	参数格式错误	参考API说明文档，修改参数
222200	request body should be json format	该接口需使用 application/json的 格式进行请求	请修改接口格式为： application/json
222201	network not available	服务端请求失败	重新尝试
222202	pic not has face	图片中没有人脸	检查图片质量
222203	image check fail	无法解析人脸	检查图片质量
222204	image_url_download_fail	从图片的url下载 图片失败	请确认url可公网访问
222205	network not availablel	服务端请求失败	重新尝试
222206	rtse service return fail	服务端请求失败	重新尝试
222207	match user is not found	未找到匹配的用户	请确认人脸库中 是否存在此用户
222208	the number of image is incorrect	图片的数量错误	多张图片请使用 json格式传输
222209	face token not exist	face token不存在	请确认您操作的 人脸已创建成功； 若face_token未注册到 人脸库则有效期只有1小时 注册人脸库的 face_token永久有效
222210	the number of user's faces is beyond the limit	人脸库中用户下的人脸数目超过限制	当前每个用户下限制人脸数目最大20张

222213	face size is too small	人脸尺寸过小，请保证人脸区域在64*64以上	请重新尝试， 请重新上传符合人脸尺寸要求的图片
222214	face are cartoon images	请使用非卡通的人脸图像	请重新尝试， 请重新上传非卡通的人脸图像
222215	face quality is not acceptable	人脸属性编辑处理该图像失败，请使用其他图片	请使用其他图片重新上传
222300	add face fail	人脸图片添加失败	重新尝试
222301	get face fail	获取人脸图片失败	请重新尝试， 如果持续出现此类错误， 请提交工单
222302	system error	服务端请求失败	重新尝试
222303	get face fail	获取人脸图片失败	请重新尝试， 如果持续出现此类错误，请提交工单
222309	image size is too small	图片尺寸过小，请使用清晰的图片	请重新尝试， 请重新上传符合尺寸要求的图片
222152	param[target] format error	人脸属性编辑，target参数错误	请修改参数后重试
222514	face editattrpro operation fail	人脸属性编辑v2调用服务失败，请重试	请重试
223100	group is not exist	操作的用户组不存在	请确认您操作的 用户组已创建成功
223101	group is already exist	该用户组已存在	请不要重复创建用户组
223102	user is already exist	该用户已存在	请不要重复创建用户
223103	user is not exist	找不到该用户	请确认您操作的 用户已创建成功
223104	group_list is too large	group_list包含组 数量过多	请按照文档提示 设置group_list参数
223105	face is already exist	该人脸已存在	请不要重复添加人脸
223106	face is not exist	该人脸不存在	请确认您操作的 人脸已创建成功； 若face_token未注册到 人脸库则有效期只有1小时， 注册人脸库的 face_token永久有效
223107	scene_type not same	人脸库中人脸复制时源组与目标组的scene_type不同	scene_type不同的组之间用户不能复制
223110	uid_list is too large	uid_list包含数量过多	请按照文档提示 设置user_list参数
223111	dst group not exist	目标用户组不存在	请确认您操作的 用户组已创建成功
223112	quality_conf format	quality_conf格式不正确	请按照文档提示设置

	error		quality_conf参数
223113	face is covered	人脸有被遮挡	提示用户请勿遮挡面部
223114	face is fuzzy	人脸模糊	人脸图片模糊， 前端页面可以提示 用户拍摄时不要晃动手机
223115	face light is not good	人脸光照不好	提示用户到光线适宜的地方拍摄
223116	incomplete face	人脸不完整	提示用户请勿遮挡面部
223117	app_list is too large	app_list包含app数量过多	请按照文档提示设置 app_list参数
223118	quality control error	质量控制项错误	请按照文档提示设置 质量控制参数
223119	liveness control item error	活体控制项错误	请按照文档提示设置 活体控制参数
223120	liveness check fail	活体检测未通过	此次活体检测结果为非活体
223121	left eye is occlusion	质量检测未通过 左眼遮挡程度过高	提示用户请勿遮挡左眼
223122	right eye is occlusion	质量检测未通过 右眼遮挡程度过高	提示用户请勿遮挡右眼
223123	left cheek is occlusion	质量检测未通过 左脸遮挡程度过高	提示用户请勿遮挡左脸颊
223124	right cheek is occlusion	质量检测未通过 右脸遮挡程度过高	提示用户请勿遮挡右脸颊
223125	chin contour is occlusion	质量检测未通过 下巴遮挡程度过高	提示用户请勿遮挡下巴
223126	nose is occlusion	质量检测未通过 鼻子遮挡程度过高	提示用户请勿遮挡鼻子
223127	mouth is occlusion	质量检测未通过 嘴巴遮挡程度过高	提示用户请勿遮挡嘴巴
222901	system busy	参数校验初始化失败	请重新尝试， 若尝试多次无效， 请提交工单咨询
222902	system busy	参数校验初始化失败	请重新尝试， 若尝试多次无效， 请提交工单咨询
222903	system busy	参数校验初始化失败	请重新尝试， 若尝试多次无效， 请提交工单咨询
222904	system busy	参数校验初始化失败	请重新尝试， 若尝试多次无效， 请提交工单咨询
222905	system busy	接口初始化失败	请重新尝试， 若尝试多次无效， 请提交工单咨询

222906	system busy	接口初始化失败	请重新尝试， 若尝试多次无效， 请提交工单咨询
222907	system busy	缓存处理失败	请重新尝试， 若尝试多次无效， 请提交工单咨询
222908	system busy	缓存处理失败	请重新尝试， 若尝试多次无效， 请提交工单咨询
222909	system busy	缓存处理失败	请重新尝试， 若尝试多次无效， 请提交工单咨询
222910	system busy	数据存储处理失败	请重新尝试， 若尝试多次无效， 请提交工单咨询
222911	system busy	数据存储处理失败	请重新尝试， 若尝试多次无效， 请提交工单咨询
222912	system busy	数据存储处理失败	请重新尝试， 若尝试多次无效， 请提交工单咨询
222913	system busy	接口初始化失败	请重新尝试， 若尝试多次无效， 请提交工单咨询
222914	system busy	接口初始化失败	请重新尝试， 若尝试多次无效， 请提交工单咨询
222915	system busy	后端服务连接失败	请重新尝试， 若尝试多次无效， 请提交工单咨询
222916	system busy	后端服务连接失败	请重新尝试， 若尝试多次无效， 请提交工单咨询
222304	image size is too large	图片尺寸太大	请确保图片尺寸在1920x1080以下
222305	pic storage not support	当前版本不支持图片存储	请确认face_token的使用符合规范
223128	group was deleting	正在清理该用户组的数据	请等该用户组清理完毕后再对 该组进行操作
223136	images exist in this group	该组内存在关联图片，无法新建相同名称组	需清理完该组下的数据后才能创建组
222361	system busy	公安服务连接失败	请重新尝试， 若尝试多次无效， 请提交工单咨询
222046	param[template_type] format error	参数格式错误	请参考接口文档修改

222101	param[merge_degree] format error	参数格式错误	请参考API说明文档，修改参数
222102	param[face_location] format error	参数格式错误	参考API说明文档，修改参数
222307	image illegal, reason: porn	图片非法 鉴黄未通过	请重新上传合法的图片
222308	image illegal, reason: sensitive person	图片非法 含有政治敏感人物	请重新上传合法的图片
222211	template image quality reject	人脸融合失败 模板图质量不合格	请检查模板图是否符合 人脸融合文档中的质量要求
222212	merge face fail	人脸融合失败	请更换素材后重新尝试， 如果持续出现此类错误，请提交工单
223129	face not forward	人脸未面向正前方 （人脸的角度信息大于30度）	请使用面向正前方的人脸图片
223130	spoofing_control item error	spoofing_control参数格式错误	请求参数spoofing_control的值不是接口文档指定的['NONE', 'LOW', 'NORMAL', 'HIGH'] 其中之一，修改请求参数 spoofing_control 值
223131	spoofing check fail	合成图检测未通过	此次活体检测结果为合成图攻击
223133	video extract image liveness check fail	视频提取图片活体检测失败	视频可能存在活体攻击
223052	action identify fail	视频中的动作验证未通过	请重新录制视频
223201	param[scene_type] format error	请求参数scene_type 格式错误	请检查请求参数
223202	scene_type does not match	识别时请求的scene_type与group设置的scene_type不匹配	检查请求参数 （此校验默认不开启）

人脸实名认证系列接口错误码

错误码	错误信息	说明	处理建议
222350	police picture is none or low quality	公安网图片不存在或 质量过低	此用户的信息没有 被公安数据覆盖到， 请将此次身份验证 转到人工进行处理
222351	id number and name not match or id number not exist	身份证号与姓名不匹配	可能身份证号码或 姓名填写错误， 请用户再次确认。 可能为作弊用户， 请增加审批环节
222022	param[id_card_number] format error	身份证号不符合格式要求	填写正确的身份证号
222023	param[name] format error	姓名格式错误	请检查姓名格式是否正确
222354	id number not exist	公安库里不存在此身份证号	对于公安数据缺失， 建议开设人工反馈渠道

222354	id number not exist	公安库里不存在此身份证号	建议开设人工反馈渠道， 进行人工认证处理
222355	ipolice picture not exist	身份证号码正确，公安库里没有对应的照片	对于公安数据缺失， 建议开设人工反馈渠道， 进行人工认证处理
222356	person picture is low quality	验证的人脸图片质量不符合要求	确认图片上的人脸是否清晰 不能有多张人脸 图片大小不能超过600KB 请重新拍摄
222357	picture file format error	待验证图片格式解析失败，不是jpeg、png、bmp文件格式	请修改上传的图片格式，重新尝试
222358	trigger risk interception	触发数据源风险拦截	数据源检测到业务使用相同身份信息，短时间内重复校验，请自查并进行调整
222361	network not available	公安服务连接失败	请重新尝试， 若尝试多次无效， 请提交工单咨询
282105	image decrypt error	图片解密失败	请重新尝试，确认下载SDK的appid与服务端请求appid是否一致 若尝试多次无效，请提交工单咨询
216201	image format error	图片格式失败	请检查传入的图片base64格式是否有误； 如您使用人脸离线采集SDK，可能为加密错误导致，请检查SDK文件
216100	invalid param	参数格式失败	请检查入参格式
282003	missing required parameter（s）	缺少必要参数	请检查入参是否完整
282000	internal error	服务器内部错误	请重新尝试， 若尝试多次无效， 请提交工单咨询
222041	param[start_date] format error	参数格式错误	参考API说明文档，修改参数
222042	param[end_date] format error	参数格式错误	参考API说明文档，修改参数

H5活体检测接口错误码列表

错误码	error_msg	错误信息	描述
216430	rtse/face service error	rtse/face 服务异常	请重新尝试
216431	voice service error	语音识别服务异常	请重新尝试
216432	video service call fail	视频解析服务调用失败	请重新尝试
216433	video service error	视频解析服务发生错误	请重新尝试，确认是否选择语音验证码但未上传音频，或语音验证码读取不正确
216500	code digit error	验证码位数错误	验证码错误， 请增加一层验证环节
216501	not found face	没有找到人脸	请查看上传视频是否包含人脸
216502	session lapse	当前会话已失效	请重新获取语音验证码
216505	redis connect error	redis连接失败	请重新尝试

216506	redis operation error	redis操作失败	请重新尝试
216508	not found video info	没有找到视频信息	请参考文档修改视频格式
216509	voice can not identify	视频中的声音无法识别 （声音过低或者有杂 音导致无法识别）	请重新录制视频
216510	video time is too long	动作活体模式验证时视频长度超过10s	请重新录制时长小于10s的视频
216511	voice file error	语音文件不符合要求	请重新录制视频
216512	action verify must post session_id	使用动作活体验证时必须使用会话id	先调用验证码下发接口获取动作类型的验证码
216513	detect_model param error	检测模型参数错误	根据接口文档要求传递参数
216612	system busy	系统繁忙	请重新尝试，若多次尝试无效请提交工单解决
216908	视频中人脸质量较差 (返回信息中包含具体原因)	视频中人脸质量过低 (返回的错误信息会包含 具体的错误信息包含 **illumiantion**(光照不足) **angle**(角度不好) **blur** (人脸模糊) **occlusion**(有遮挡) **too large**(人脸过大, **占屏幕2/3以上**) 等原因	请重新录制视频
216909	video all image detect over two face	检测所有视频抽帧图片的人脸数均超过2	确保录制的视频中仅有一人
222027	code length param error	验证码长度错误 (最小值大于最大值)	参考API说明文档，修改参数
222028	param[min_code_length] format error	参数格式错误	参考API说明文档，修改参数
222029	param[max_code_length] format error	参数格式错误	参考API说明文档，修改参数
222030	param[match_threshold] format error	参数格式错误	参考API说明文档，修改参数
223050	voice similarity low error	视频中的语音与语音验证码所对应数值相似度过低	请重新录制视频

人脸实名认证H5方案错误码列表

错误码	错误信息	描述
100	Invalid parameter	无效的参数
110	Access token invalid or no longer valid	访问链接无效或已过期，请重新生成
111	Access token expired	访问链接已过期，请重新生成
200	unsupported operation	不支持的操作
283456	图片为空或格式不正确	上传的图片为空或格式不正确
283400	服务异常，请稍后再试	服务异常，请稍后再试，如频繁出现，可提交工单反馈
283458	当前链接已失效，请重试	当前链接已失效，请重新生成链接（请检查verify_token是否生成或生成超过了2小

283458	当前链接已失效，请重试	时，核验成功或配置了核验失败1次后失效同样会触发该报错）
283459	请从手机端扫描二维码访问继续认证流程	请从手机端扫描二维码，以继续认证流程
283460	视频文件过大，核验请求超时	上传的视频文件过大，导致核验请求超时
222202	图片中没有人脸	上传的图片中没有检测到人脸
222203	无法解析人脸	无法解析上传的人脸图片
223113	人脸有被遮挡	上传的人脸图片被遮挡
223114	人脸模糊	上传的人脸图片模糊不清
223115	人脸光照不好	上传的人脸图片光照不好
223116	人脸不完整	上传的人脸图片不完整
223129	人脸未面向正前方	上传的人脸图片未面向正前方
223121	质量检测未通过 左眼遮挡程度过高	上传的人脸图片中左眼被遮挡程度过高
223122	质量检测未通过 右眼遮挡程度过高	上传的人脸图片中右眼被遮挡程度过高
223123	质量检测未通过 左脸遮挡程度过高	上传的人脸图片中左脸被遮挡程度过高
223124	质量检测未通过 右脸遮挡程度过高	上传的人脸图片中右脸被遮挡程度过高
223125	质量检测未通过 下巴遮挡程度过高	上传的人脸图片中
223126	质量检测未通过 鼻子遮挡程度过高	鼻子被遮挡，无法正常检测
223127	质量检测未通过 嘴巴遮挡程度过高	嘴巴被遮挡，无法正常检测
223131	合成图检测未通过	检测到图片为合成图，不符合要求
216434	活体检测未通过	无法通过活体检测，可能存在欺诈行为
216501	没有找到人脸	无法在视频或图像中找到人脸
216508	视频中有多张人脸	视频或图像中存在多张人脸，无法识别
216509	视频中的声音无法识别	视频中的声音质量过差，无法识别
216510	动作活体模式验证时视频长度超过10s	动作活体模式下，视频长度不能超过10秒
216511	语音文件不符合要求	语音文件质量不符合要求，无法识别
223050	语音验证未通过	语音验证未通过，无法确认身份
223051	唇语验证未通过	唇语验证未通过，无法确认身份
223052	动作验证未通过	动作验证未通过，无法确认身份
283738	颜色验证未通过	颜色验证未通过，无法确认身份
222350	公安网图片不存在或质量过低	公安网不存在此照片或照片质量过低
222351	身份证号与姓名不匹配	提交的身份证号与姓名不匹配
222022	身份证号不符合要求	填写正确的身份证号
222023	姓名格式错误	请检查姓名格式是否正确
	身份证号码正确，公安库里没	

222355	有对应的照片	身份证号码正确，但公安库里没有对应的照片
222356	验证的人脸图片质量不符合要求	提交的人脸图片质量不符合要求，无法确认身份
222360	身份核验未通过	身份核验未通过，无法确认身份
216600	身份证号码格式错误	身份证号码格式不正确
216601	身份证号和名字不匹配	提交的身份证号和姓名不匹配
283457	当前环境存在安全风险	当前环境检测触发了安全风控规则
283501	安全检验未通过	请求中的安全校验未通过，可能是请求被篡改或存在中间人攻击等风险。
283421	应用不存在。App not exist.	传入的应用 ID 不存在或已被删除。
216612	系统繁忙	服务端暂时无法处理请求，建议稍后再试。
283437	Token无效或已过期，请重新生成	认证 token 已过期或无效，请重新生成 token 后再进行请求。
283438	视频转码失败，请重试	视频文件转码失败，建议重新上传或选择其他视频文件。
283439	STS_Token 已经生成	STS Token 已经生成过，请勿重复请求。
283464	非法流程	当前请求的流程状态异常或非法。
283461	人脸和对比源不匹配	人脸图片和传入的对比源不匹配，建议核对传入的数据是否正确。
283462	比对源配置错误	对比源配置错误，建议核对传入的数据是否正确。
283463	人脸图片质量检测未通过	人脸图片质量检测未通过，建议上传更清晰的图片。
283465	人脸图片活体检测未通过	人脸图片活体检测未通过，建议重新采集图片或更换采集环境。
283467	该PLAN_ID下未查询到图片文件	传入的 plan_id 下未查询到图片文件，建议核对传入的 plan_id 是否正确。
283468	BOS文件上传失败	文件上传至 BOS 失败，建议重新上传或检查上传权限是否正确。
283469	用户请求的 body 是空	用户请求的 body 是空，请检查请求参数是否正确。
283435	方案不存在。Plan not exist.	传入的方案 ID 不存在或已被删除。
283440	身份证照片不符合要求，请重新上传	上传的身份证照片不符合要求，建议重新上传符合要求的身份证照片。
283442	身份证信息不合法，请重新填写	上传的身份证信息不合法，建议重新填写正确的身份证信息。
283443	不可使用语音验证码	当前环境下不支持使用语音验证码，建议使用其他方式进行验证。
283444	语音验证码生成失败，请重试	语音验证码生成失败，请检查参数是否正确或稍后再试。
283447	验证失败，请稍后重新尝试	验证失败，请稍后重新尝试。
283448	语音验证码不符合要求	语音验证码不符合要求，请重新生成验证码。
283449	活体检测视频不符合要求，请重新上传	上传的活体检测视频不符合要求，建议重新上传符合要求
283450	认证尚未开始。Verification is not started yet.	认证流程尚未开始。
283451	认证处理中。Verification is running.	认证流程正在处理中。
283453	不可使用照片活体	不支持使用照片进行活体检测。
283454	不可使用视频活体	不支持使用视频进行活体检测。
283455	超出查询有效期。Result is expired.	查询结果已经过期。

283436	Token生成失败，请重试	认证 token 生成失败，请重试。
283502	视频文件上传 bos 失败	视频文件上传至 BOS 失败，请重新上传或检查上传权限是否正确。
283503	对比源信息未传入	传入的对比源信息为空。
283504	请上传正确的身份证照片	上传的身份证照片不符合要求，请重新上传符合要求的身份证照片。
283505	请上传正确的身份证人像面	上传的身份证人像面不符合要求，请重新上传符合要求的身份证照片。
283506	请上传正确的身份证国徽面	上传的身份证国徽面不符合要求，请重新上传符合要求的身份证照片。
283507	不可使用身份证识别	不支持使用身份证进行人脸识别。
283601	重复推送错误信息	重复推送相同的错误信息。
222038	证件号或国籍参数格式错误	传入的证件号或国籍参数格式不正确。
300201	您已拒绝授权摄像头，如需继续，请点击重新拍摄	用户拒绝授权实时视频流获取，无法完成核验流程
300001	受当前环境限制（300001），请更换浏览器或设备重试	当前环境不兼容实时检测，且方案配置为不允许降级，无法完成核验流程
300002	受当前环境限制（300002），请更换浏览器或设备重试	当前环境不兼容实时视频录制，且方案配置为不允许降级，无法完成核验流程
999999	请确保是本人操作且正脸采集	请确保是本人进行操作且采集的人脸为正脸。
800001	采集超时	用户人脸采集流程超时
800002	炫瞳检测失败	用户炫瞳检测环节验证失败

常见问题及排查

注意事项

1. 在控制台里创建应用时，获取的API Key、Secret Key，可用来生成鉴权签名（Access Token）。请求鉴权接口时，参数client_id 就是指 API Key，client_secret 是指 Secret Key。
2. 同一账号下，同一接口服务所享有的免费额度是一定的，无论创建多少AppId，都共享这个额度。

获取Access Token错误

现象/报错	**常见错误/排查思路**
得到一个html页面	接口地址输入错误，比如末尾多了空格、逗号、斜线（/）等
提示invalid_client	1. client_id输入错误 2. client_secret输入错误
提示404	grant_type输入错误
提示unsupported_grant_type	没有传grant_type参数
提示invalid_request	1. 没有传client_id参数 2. 没有传client_secret参数

使用Access Token调用接口时受限

若请求错误，服务器将返回的JSON文本包含以下参数：

- error_code：错误码。
- error_msg：错误描述信息，帮助理解和解决发生的错误。

例如Access Token失效返回：

```
{
 "error_code": 110,
 "error_msg": "Access token invalid or no longer valid"
}
```

提示错误码6

错误码6：no permission to access data，表示没有接口权限。

核实步骤	操作方法
确认该应用是否有该接口的权限	1. 登录控制台 2. 点击左侧「应用列表」,点击正确的应用名称，进入详情页 3. 查看API列表中的「API」名称或「请求地址」，核实是否包含目标接口。如果没有，就说明该应用没有该接口的权限。

常见原因	解决方法
所使用的Access Token，其client_id和client_secret不是从正确的应用下获取的	在应用列表里核实，获取过 client_id（即 API Key）和 client_secret（即 Secret Key）的应用是否正确，如是否把正式应用和测试应用搞混了
未开通接口权限	1. 编辑应用，重新勾选所需要调用的接口 2. 或者在控制台的人脸页面里创建新应用，使用新的 client_id 和 client_secret
接口地址写错	1. 登录控制台 2. 点击左侧「应用列表」,点击正确的应用名称，进入详情页 3. 查看API列表中的「请求地址」，与接口文档对照，看是否错漏 注意：区分V2、V3版本，现在新用户开通的，默认是V3版本的接口
未完成企业认证，无法开通公安验证（包括开通企业认证，但未生效）	1. 个人用户：公安验证接口需要完成企业认证才开通权限 2. 企业用户：完成企业认证，需重新登录控制台，进入人脸识别页面，约30分钟后生效。

提示错误码17

错误码17：Open api daily request limit reached，表示流量超限额。

核实步骤	操作方法
确定接口的免费次数	1. 查看免费额度中，该接口的免费额度说明 2. 在控制台概览页，找到该接口，查看免费额度 3.如果已开通付费，核实账号余额或者可用代金券是否大于0
查看接口的已使用量	进入控制台概览页，点击左侧「监控报表」 1. 选择「全部应用」 2. 选择报错的API接口 3. 统计项选择「调用量」 4. 监控项勾选「调用成功」和「调用失败」 5. 时间段选择能够覆盖报错的时间起止（统计有2小时左右的延迟）

常见原因	操作方法
已开通付费：接口的免费额度已使用完毕，账户余额不足	给账户充值
未开通付费：接口的免费额度已使用完毕	开通付费并充值
测试接口：接口的免费额度已使用完毕，测试阶段未上计费	提交工单申请更高免费额度

提示错误码18

错误码18: Open api qps request limit reached，表示QPS超限

核实步骤	操作方法
确定接口的免费QPS额度	1. 查看免费额度中，该接口的免费额度说明 2. 在控制台概览页，找到该接口，查看免费额度 3.如果已开通付费，核实账号余额或者可用代金券是否大于0
查看接口的已使用QPS量	进入控制台概览页，点击左侧「监控报表」 1. 选择「全部应用」 2. 选择报错的API接口 3. 统计项选择「QPS」 4. 监控项勾选「QPS峰值」 5. 时间段选择能够覆盖报错的时间起止（统计有2小时左右的延迟）

常见原因	操作方法
QPS额度不足	1. 个人用户免费QPS是2，完成企业认证，免费额度会提升到10qps 2. 付费购买超限接口的QPS 3. 也可降低请求并发 注：QPS，Queries-per-second，即并发，是指每秒钟请求某服务的次数
公安验证接口已开通付费，但处于欠费状态	给账户充值，补齐欠费

人脸实名认证方案

方案介绍

方案简介

随着移动互联网的快速发展，全行业“实名制”已成为共识，尤其在一些涉及资金操作、敏感信息查阅的业务场景中，通过人脸识别的方式完成远程身份核验已非常普及。随之而来的，不少黑产份子开始针对人脸识别过程进行伪造攻击，对业务安全产生威胁。因此，人脸识别过程中加入活体检测、设备风控等安全能力，保障整体业务流程安全尤为重要。

百度人脸实名认证解决方案：具备OCR身份证识别、多样式实时活体检测、设备环境风控、数据安全加密、权威库人脸核验、自传人脸比对等多项组合能力，支持在**APP（含IOS、安卓、鸿蒙）及H5（含浏览器、微信、小程序）等业务场景**快速完成用户身份核验，提供安全可靠的防攻击能力，为企业业务安全保驾护航。

一、功能介绍

1、身份核验（含人脸）：将姓名、身份证号、人脸图片与**权威库**进行核验，得出比对分数，并基于此进行业务判断是否为同一人；同时支持**与系统已有人脸图片**进行1:1比对，得出相似度分数，判断用户身份；

2、活体检测：提供**实时炫瞳、动作、静默等多种活体检测能力，可按照实际业务需求灵活选择。基于云端多因子活体大模型：**针对**金融、物流、泛互联网**业务场景进行专项活体模型优化，适配不同业务需求的同时，提供高效的防御拦截能力，可抵挡屏幕、照片、视频、换脸、面具、3D模型等非活体攻击，并保证高通过率和识别率。

3、设备环境风控：基于SDK端采集的设备环境信息，对前端业务环境进行风险扫描，辨别是否为风险设备，返回风险等级及标签结果，可有效防御黑产批量虚拟机、病毒侵入等攻击手段。

4、端云数据加密：SDK端支持对人像图片信息进行加密，在云端接口进行信息解密，利用这种端云结合的加解密方案，可有效避免第三方非法黑产绕过APP模拟请求攻击云端接口的行为，如脚本攻击、ROM注入、视频劫持等，极大增加对黑产常用APP攻击的防御能力，保障端云双重维度的信息一致性。

二、方案优势

强安全：

- **权威库核验**：将实时采集的真人人脸图片、姓名、身份证号与权威库对比，确保操作者身份的真实性。避免身份证或人脸图像伪造等欺诈风险，权威可靠。
- **防作弊能力强**：提供多种活体检测方案，支持图片质量校验、多帧图片识别，有效抵御照片、合成图、视频翻拍、3D模型等作弊行为；业内首次推出合成图校验，有效应对PS、人脸融合后的图片/视频攻击。
- **加密防篡改**：摒弃传统人脸SDK直接采集摄像头信息的方式，升级为安全通道采集方式，从代码逻辑、采集方式、传输层多维度确保人脸信息采集真实有效。

体验优：

- **交互形式多样：**支持在APP、通用/微信H5等接入方式，产品形态全面丰富，灵活组合使用，满足各类场景需求
- **活体检测方式多样**：提供多种活体检测方案，支持实时炫瞳、动作、静默核验等多种防作弊手段，可根据不同业务需求灵活选择。

集成快：

- **可视化方案配置：**方案支持自定义设置人脸质量参数及活体检测动作，同时接口返回参数、阈值可根据业务需求灵活配置。
- **提供UI及demo：**提供包含UI交互的整套示例代码，修改参数配置即可使用，大大减少了开发集成成本。

三、适用场景

金融保险风控

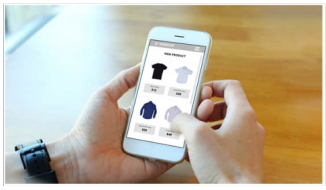
开户、授信、投保理赔、电子签约

服务人员身份监管

网约车司机、房产经纪人、快递员

社交直播

主播、用户交友实名、修改卡密



电子商务

海淘清关、商家实名、奖励提现

民事政务

访客登记、线上办事、企业报税

学生/考生身份核验

艺考/研究生招生、专业执照考试、招聘

- **金融风控** 在投资理财、保险理赔、证券交易等安全性要求高的金融场景，运用人脸实名认证方案，将线下业务转为线上自助模式，满足远程开户、保险回执单等业务需求。同时可以辅助密码找回等密保措施，降低用户身份信息被恶意篡改、顶替冒用等风险，提升信息安全管理
- **民事政务** 针对社保身份核实、政府访客登记、政务大厅自助窗口等场景，将原本繁琐费时的柜台业务办理，转为线上自助服务，无需窗口排队等待，提高业务处理效率
- **酒店入住** 酒店宾馆行业人员流动频繁，以往通过肉眼判断身份证件真假、是否本人持证的方式效率低下，准确度难以保证，无法满足企业和政府的监管需求。应用人脸实名认证技术，高精度核实住客身份信息，为住户提供更便捷的高质量服务
- **服务人员身份监管** 在货品运输、家政保洁等服务行业，对从业人员的身份真实性要求较高。运用人脸实名认证方案，从业人员可自助完成实名认证，防止身份被顶替、冒用，提升身份审核效率，保障业务安全
- **共享业务** 在共享班车、定制巴士、分时租赁等共享服务中，组合使用活体检测、身份证识别、人脸质量检测、人脸对比等能力，提升“注册-认证-审核-用车”全环节效率，实现驾驶全流程身份核验，保障信息安全，有效规避智能出行服务风险
- **考生身份核验** 目前招聘考试、专业执照考试、企业内部评测等考核逐渐转为在线操作，但信息作假、替考等风险较多，且难以系统化监管。引入人脸实名认证身方案，提供便捷、高效的身份核验服务，确保考生身份真实有效，有助于加强行业监管

价格说明

人脸实名认证解决方案

> **说明**：
> 1、使用人脸实名认证方案需完成企业认证
> 2、登录人脸识别控制台后，系统将自动发放免费测试额度
> 3、免费测试额度用尽后，可 购买次数包 或 开通按量后付费。

1. 人脸实名认证v4（与权威源比对）

- **功能说明**：通过APP/H5方案接入，可实现动作、打光等多种类型活体检测，并将用户姓名、身份证号、人脸信息与**权威数据源**中的身份信息进行比对，基于此核验用户身份。
- 以下费用包括**APP/H5方案**中的权威源核验、活体检测、合成图检测、质量检测功能，如需OCR身份证识别能力，需单独付费。

- **温馨提示：**完成企业认证后，服务并发量支持**2QPS**，正式付费后，并发量将扩充至**10QPS**。如业务需扩容更多QPS，可以**联系您的商务经理**，或通过合作咨询联系我们。

（1）按量后付费：

月调用量（次）	价格（元/次）
0 < n <= 10,000	0.8
10,000 < n <= 100,000	0.72
100,000 < n <=200,000	0.6
200,000 < n	0.48

（2）预付费资源包

次数包	价格	有效期
1,000 次	780 元	一年
5,000次	3,700元	一年
10,000 次	7,000 元	一年
50,000 次	33,000 元	一年
100,000 次	61,000 元	一年
500,000 次	275,000 元	一年
1000,000 次	500,000 元	一年

> **说明**
> 1、本接口**只计费成功调用**。失败调用会返回对应的错误码，详情说明可参考错误码文档
> 2、预付费包年次数包**有效期为1年**，支持叠加购买，按照购买时间顺序依次抵扣，全部使用完毕后自动切换为按量后付费模式。次数包购买时会自动开通**按量后付费，请勿在消耗次数包时终止后付费，避免影响正常使用**。次数包超出有效期未抵扣额度自动失效，无法继续使用，请根据实际业务需求酌情购买。
> 3、预付费包年次数包购买后**7天内未产生调用**，如有需要，您可对购买的QPS进行自助退款，您可前往百度智能云'控制台--财务--退订管理'页面进行自助退订，详细退订规则见退订说明， **超过7天**或**已产生调用的次数包**无法退款。

2. 人脸比对v4（与上传照片比对）

- **功能说明**：通过APP/H5方案接入，可实现动作、打光等多种类型活体检测，并将实时采集的人脸信息与您业务侧上传的人脸图片进行比对，基于此核验用户身份。
- 以下费用包括**APP/H5方案**中的自传照片核验、活体检测、合成图检测、质量检测功能，如需OCR身份证识别能力，需单独付费。
- **温馨提示：**完成企业认证后，服务并发量支持**2QPS**，正式付费后，并发量将扩充至**10QPS**。如业务需扩容更多QPS，可以**联系您的商务经理**，或通过合作咨询联系我们。

（1）按量后付费

分段阶梯计费：到达相应阶梯的计费调用量按照所在阶梯单价进行计费。如1月份内接口调用量为15000次，则费用为 ：10000 x 0.2 + 5000 x 0.18 = 2900 元.

月调用量（次）	价格（元/次）
0 < n <= 10,000	0.20
10,000 < n <= 100,000	0.18
100,000 < n <=500,000	0.15
500,000 < n	0.12

（2）预付费资源包

有效期为1年，次数包支持叠加购买，按照购买时间顺序依次抵扣，全部使用完毕后自动切换为按量后付费模式。超出有效期未抵扣额度自动失效，无法继续使用，请根据实际业务需求酌情购买。

次数包	价格	有效期
10,000 次	1,950 元	一年
100,000次	19,000元	一年
500,000 次	90,000 元	一年
1,000,000 次	160,000 元	一年
3,000,000 次	420,000 元	一年
5,000,000 次	600,000元	一年

（3）QPS叠加包

购买 QPS 数量	按月购买	按天购买
0<QPS<=10	270元/月/QPS	16元/天/QPS
10<QPS<=50	235元/月/QPS	14元/天/QPS
50<QPS<=100	170元/月/QPS	12元/天/QPS
100<QPS	135元/月/QPS	10元/天/QPS

> **说明**
>
> 1、本接口**只计费成功调用**。失败调用会返回对应的错误码，详情说明可参考错误码文档
>
> 2、预付费包年次数包**有效期为1年**，支持叠加购买，按照购买时间顺序依次抵扣，全部使用完毕后自动切换为按量后付费模式。次数包购买时会自动开通**按量后付费**，**请勿在消耗次数包时终止后付费，避免影响正常使用**。次数包超出有效期未抵扣额度自动失效，无法继续使用，请根据实际业务需求酌情购买。
>
> 3、预付费包年次数包购买后**7天内未产生调用**，如有需要，您可对购买的QPS进行自助退款，您可前往百度智能云'控制台--财务--退订管理'页面进行自助退订，详细退订规则见退订说明， **超过7天**或**已产生调用的次数包**无法退款。

3. 在线图片活体v4（仅活体检测）

- **功能说明**：通过APP/H5方案接入，可实现动作、打光等多种类型活体检测，判断用户真实性。拦截各类伪造攻击行为。
- 以下费用包括**APP/H5方案**中的多类活体检测、合成图检测、质量检测功能。
- **温馨提示：**完成企业认证后，服务并发量支持**2QPS**，正式付费后，并发量将扩充至**10QPS**。如业务需扩容更多QPS，可以**联系您的商务经理**，或通过合作咨询联系我们。

（1）按量后付费

开通按量后付费期间，QPS为10，按如下价格计费。

月调用量（次）	价格（元/次）
0 < n <= 10,000	0.100
10,000 < n <= 100,000	0.090
100,000 < n <=500,000	0.075
500,000 < n	0.060

（2）预付费资源包

> 1. 预付费包年次数包有效期为1年，支持叠加购买，按照购买时间顺序依次抵扣，全部使用完毕后自动切换为按量后付费模式。次数包购买时会自动开通**按量后付费**，**请勿在消耗次数包时终止后付费，避免影响正常使用**。次数包超出有效期未抵扣额度自动失效，无法继续使用，请根据实际业务需求酌情购买。
> 2. 预付费包年次数包购买后**7天内未产生调用**，如有需要，您可对购买的QPS进行自助退款，您可前往百度智能云'控制台--财 务--退订管理'页面进行自助退订，详细退订规则见退订说明， **超过7天**或**已产生调用的次数包**无法退款。

次数包	价格	有效期
10,000 次	980 元	一年
100,000 次	9,500 元	一年
500,000 次	45,000 元	一年
1,000,000 次	80,000 元	一年
3,000,000 次	210,000 元	一年
5,000,000 次	300,000 元	一年

（3）QPS叠加包

> 完成企业认证后接口并发支持默认赠送2QPS，开通付费后，并发支持将扩充到10QPS，如需更多QPS，可购买QPS叠加包。

购买 QPS 数量	按月购买	按天购买
0<QPS<=10	270元/月/QPS	16元/天/QPS
10<QPS<=50	235元/月/QPS	14元/天/QPS
50<QPS<=100	170元/月/QPS	12元/天/QPS
100<QPS	135元/月/QPS	10元/天/QPS

4. OCR身份证识别

详情请点击了解：https://ai.baidu.com/ai-doc/OCR/fk3h7xune#身份证识别

人脸实名认证方案-APP端

方案简介

方案简介

推出背景

- 现在，人脸识别技术被广泛应用在金融支付、用户注册、人脸登录等业务场景中。技术的进步方便用户的同时，黑灰产产业也开始对这些场景产生觊觎。并通过**屏幕攻击、照片、纸张、以及面具、头模**等方式进行非法攻击。随着黑产技术的进步，更是出现了通过**自动化脚本直接攻击云端API、ROM注入、视频劫持替换、批量虚拟机、病毒侵入**等新型攻击手段。使现有的人脸识别方案面临着巨大的安全挑战。
- 为提升人脸识别的安全性，保障客户的业务安全，便于客户在静默活体、动作活体、炫瞳活体多种活体验证方式中灵活切换，人脸实名认证产品团队与百度安全实验室联合推出**人脸实名认证APP方案**，在人脸登录、注册等环境加入层层保障，为您的业务保驾护航。

功能简介

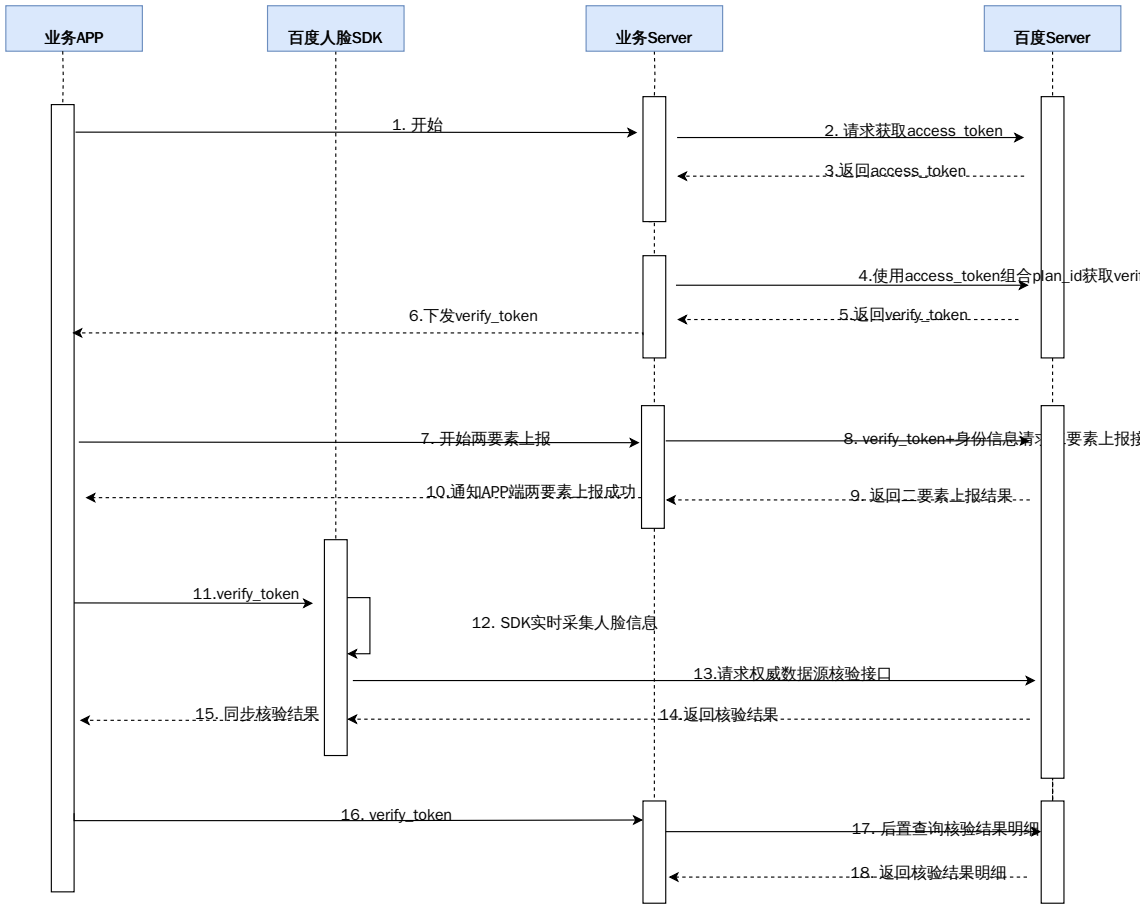
- 人脸实名认证APP方案提供标准化的人身核验流程，具有**人脸比对、活体检测、证件识别、人脸实名认证**等多项组合能力，以及端云配合高防攻击拦截能力，可抵挡**高清屏幕、照片、视频、AI换脸、高仿真面具、3D模型**的攻击。支持**静默活体、动作活体、炫瞳活体**等活体校验方式进行人脸采集，同时，加入**安全加密能力以及风控能力**，针对脚本攻击、ROM注入、视频劫持、批量虚拟机、病毒侵入等新型攻击手段进行强力有效防御。

> **人脸质量检测：**判断视频流中的图片帧中，哪些图片质量较佳，即人脸图像特征清晰（满足姿态角、光照、模糊度、遮挡等校验）。
>
> **人脸图像采集：**通过本地SDK能力，采集人脸图像，同时经过人脸质量检测，确保采集到的人脸图像符合各条件校验（满足姿态角、光照、模糊度、遮挡等校验），为设备前端获取有效可分析人脸的主要功能。
>
> **炫瞳活体检测：**通过屏幕上闪烁不同颜色的光线，判断当前用户是否真人操作。通过颜色活体进行面部反光鉴别的同时，百度特加入独有的瞳孔反光识别，提升整体的攻击拒绝率指标。
>
> **动作活体检测：**通过让用户做出指定人脸配合式的交互动作，识别当前操作者是否为活体，此功能为离线使用，可设定指定动作是否使用及应用顺序。动作包含：眨眨眼、张张嘴、向左转头、向右转头、向上抬头、向下低头、上下点头、左右摇头8个。
>
> **端云互验加密：**人脸实名认证APP方案集成文件中的采集SDK会对输出的图片进行加密（支持AES/国密），在云端接口进行解密。此端云配合的加密方式是百度专门针对市面黑产绕过采集SDK，攻击云端接口的攻击方式进行的功能升级。
>
> **风控能力：**云端接口接受SDK端传入的本地环境扫描设备指纹及安全信息，对SDK端进行设备风险识别，辨别是否为风险设备，返回识别结果。可有效防御黑产批量虚拟机、病毒侵入等攻击手段，降低第三方黑产攻破概率，提升业务安全性。

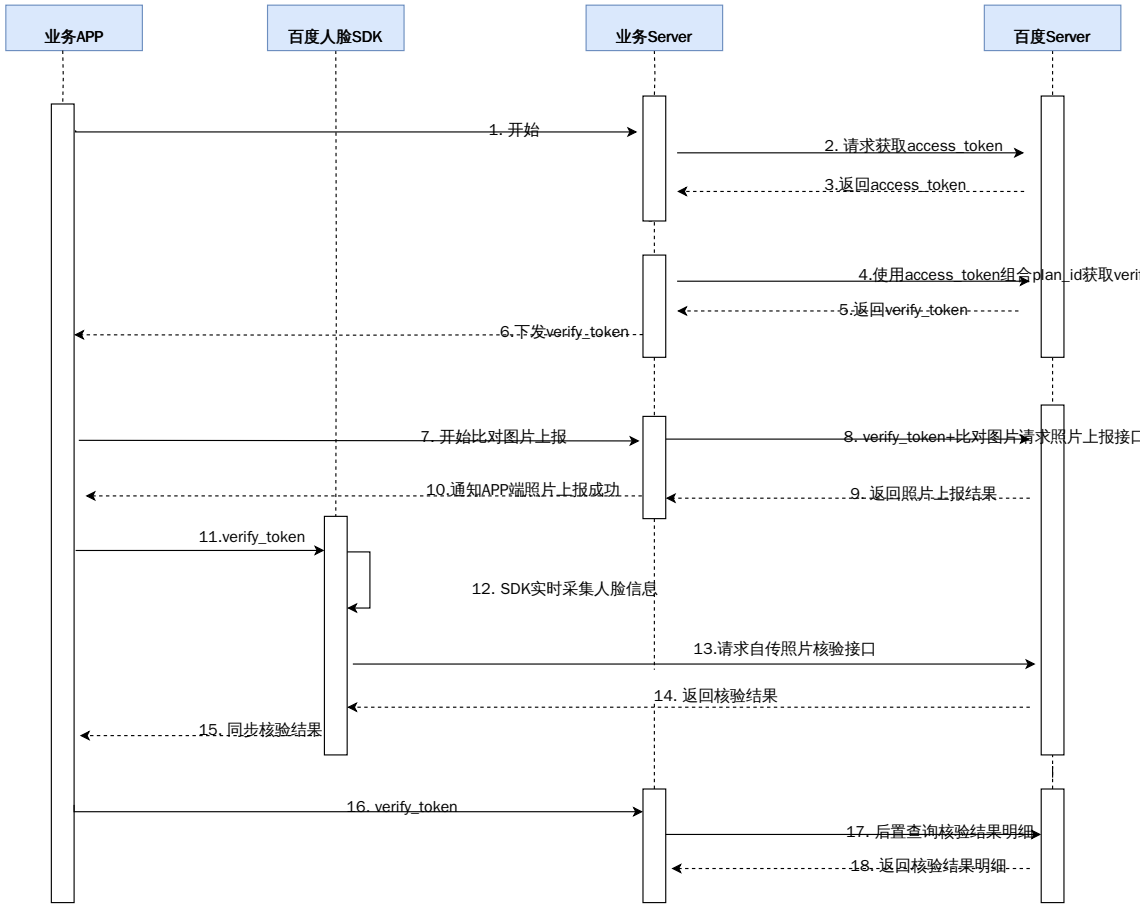
适用场景

- 人脸实名认证APP方案适用于**Android/IOS/HarmonyOS的APP场景**中实现用户实名认证、人脸比对、活体检测。如果您的业务场景是在微信小程序、公众号、APP内嵌H5等业务场景，推荐采用H5实名认证方案。

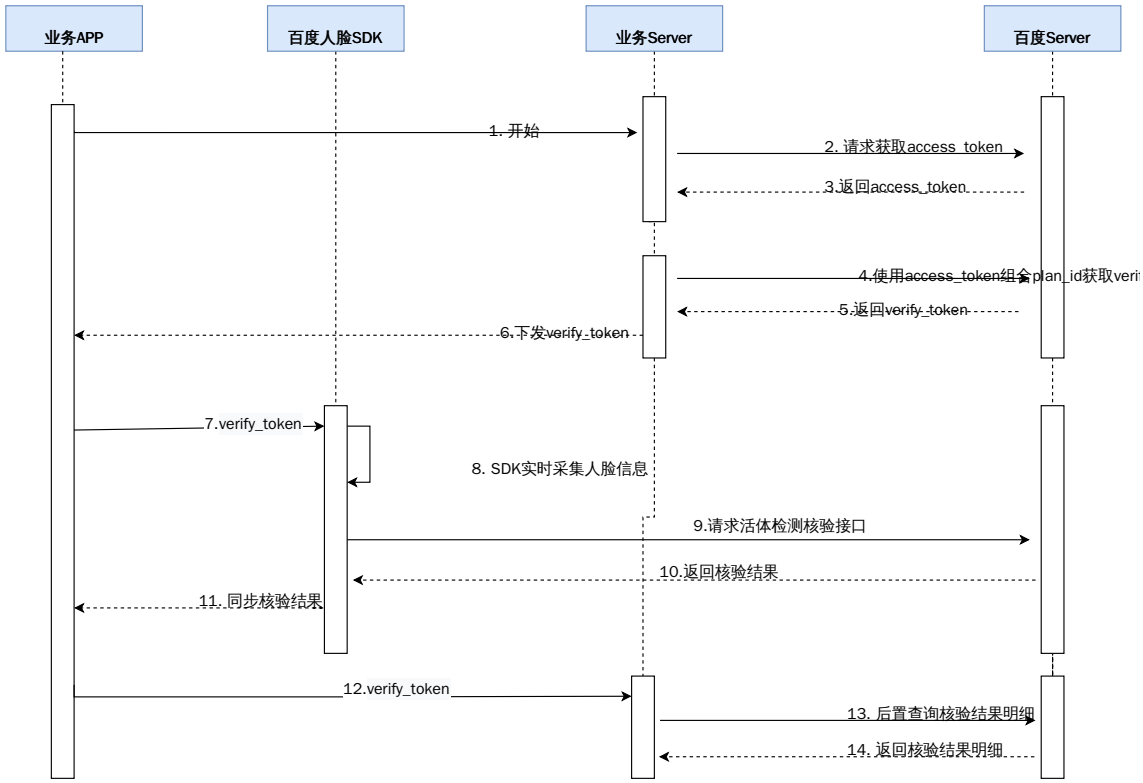
接入时序图 1. 人脸实名认证，人脸采集图片、姓名、身份证号码三要素核验



2. 人脸对比，人脸采集图片与自传图片的1:1比对



3. 活体检测，对人脸采集图片进行活体检测



重要：AccessToken有有效期，每次请求必须重新获取。出于安全性考虑，将AK、SK写到业务Server，请求百度服务器，间接获取AccessToken。关于AccessToken鉴权，请参考https://ai.baidu.com/ai-doc/REFERENCE/Ck3dwjhhu

方案接入步骤 人脸实名认证APP方案的具体接入步骤请参考

- Android方案接入指南
- iOS方案接入指南
- HarmonyOS方案接入指南

方案集成前准备

在正式集成前，需要做一些准备工作，完成一些账号、应用及配置，具体如下：

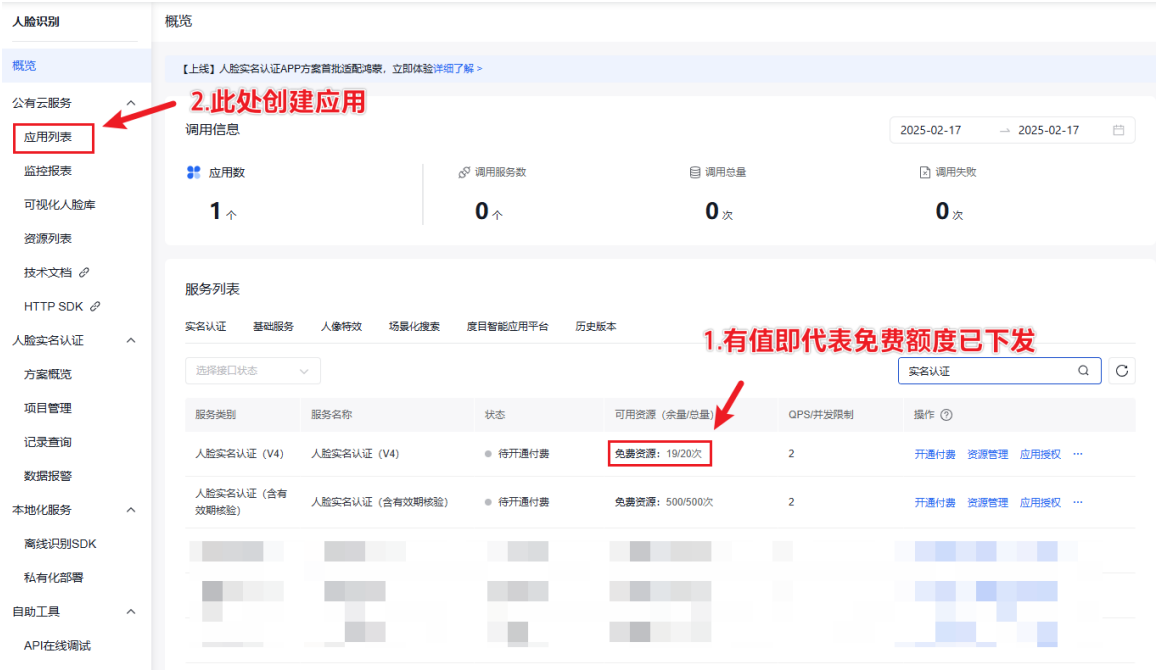
Step1: 注册成为开发者

> 在使用百度人脸实名认证方案之前，首先需注册百度云账号，账号注册方式请参考**账号注册指南**。
>
> 百度云账号注册完成以后，为顺利调用百度AI能力，需完成企业认证。具体认证方式请参考**企业认证指南**。

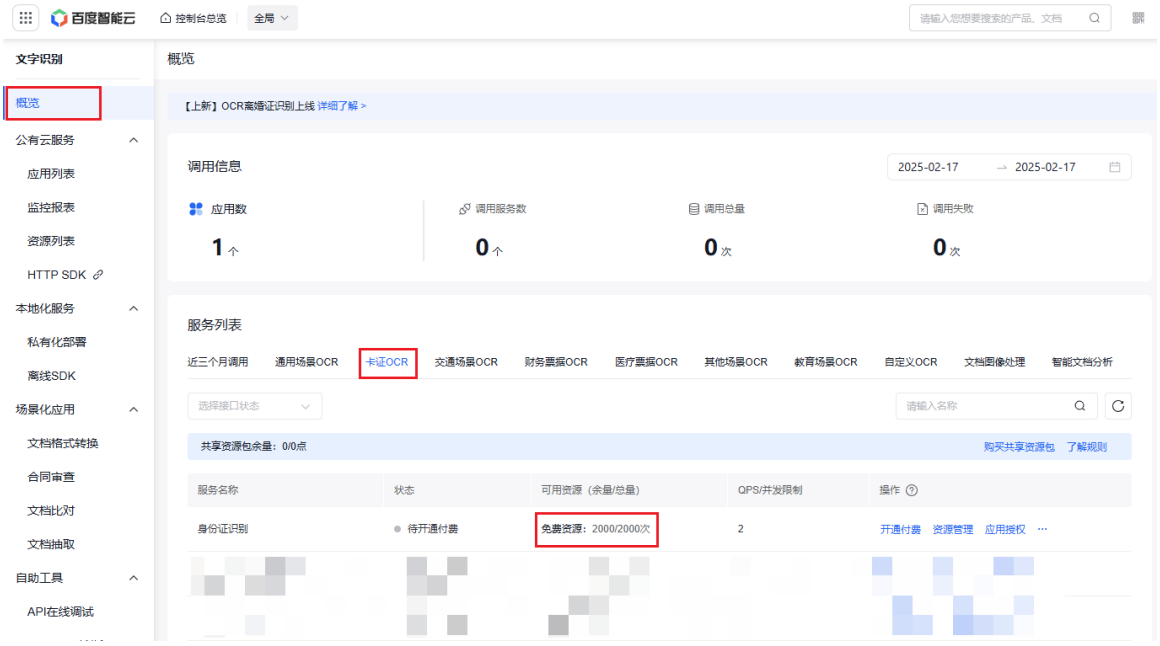
Step2：创建应用

2.1 输入应用名称，领取免费额度

- 创建好账号后，在正式调用AI 能力之前，需首先**创建应用**，应用是调用服务的基础能力单元，目前仅支持**人脸识别**方向下创建的应用。
- 确认接口所需的**免费调用额度**已经下发（在您完成实名认证后，首次进入控制台即自动领取。领取到的资源会在10分钟内发放至账户），用于接入测试。如下图所示：



- 除人脸服务接口的免费调用额度外，还需确认**身份证识别**接口的**免费调用额度**已经下发，用来调用身份证识别功能（必须领取，否则会报错服务异常），点击此处，按下图所示进行领取。



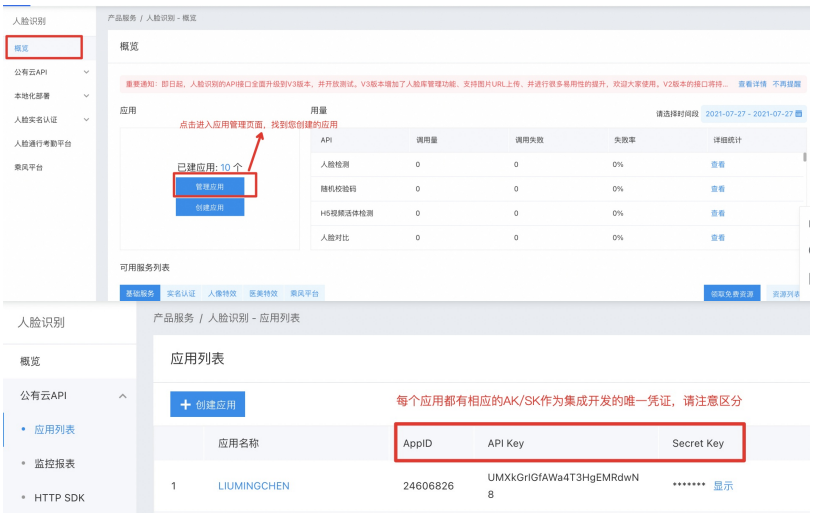
2.2 勾选所需接口

- **人脸识别服务相关接口已默认勾选**

- **注：**「接口选择」过程中，还**需勾选「文字识别」中的「身份证识别」接口**，用于实现身份核验流程中的身份证识别功能。如下图所示。

2.3 获取密钥信息（AK/SK）

完成应用创建后，平台将会分配给您此应用的相关凭证，主要为**AppID、API Key、Secret Key**，以上三个信息是您应用实际开发的主要凭证，每个应用之间各不相同，请您妥善保管。您可在控制台的**应用管理页面**找到以上信息。如下图所示



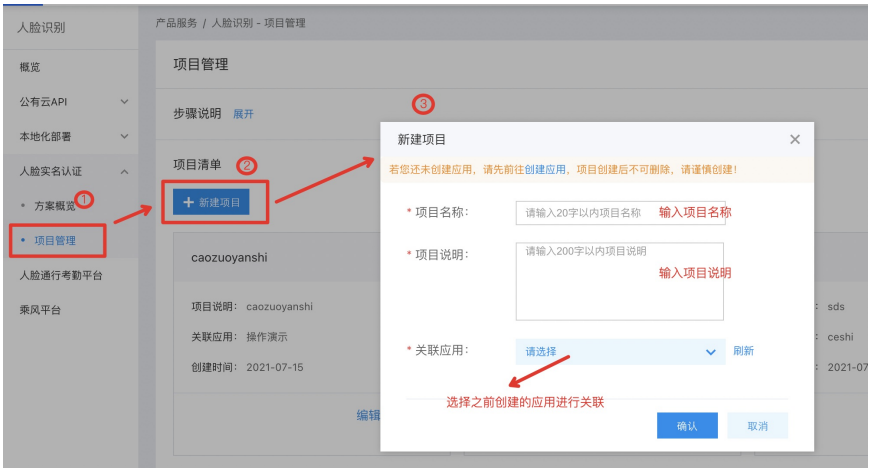
该AK/SK用于调用在线API 如：身份验证。**在之后下载的集成文件（示例工程）中需要填写正确的AK/SK以顺利集成。**

注：开发中请注意区分多份AK/SK（API Key、Secret Key），若填写的AK/SK与开发的应用不对应，会产生鉴权错误。

Step3：创建项目

- 进入控制台-人脸实名认证页面，选择『**项目管理**』页面，点击『**新建项目**』，进行项目创建，如下图所示。

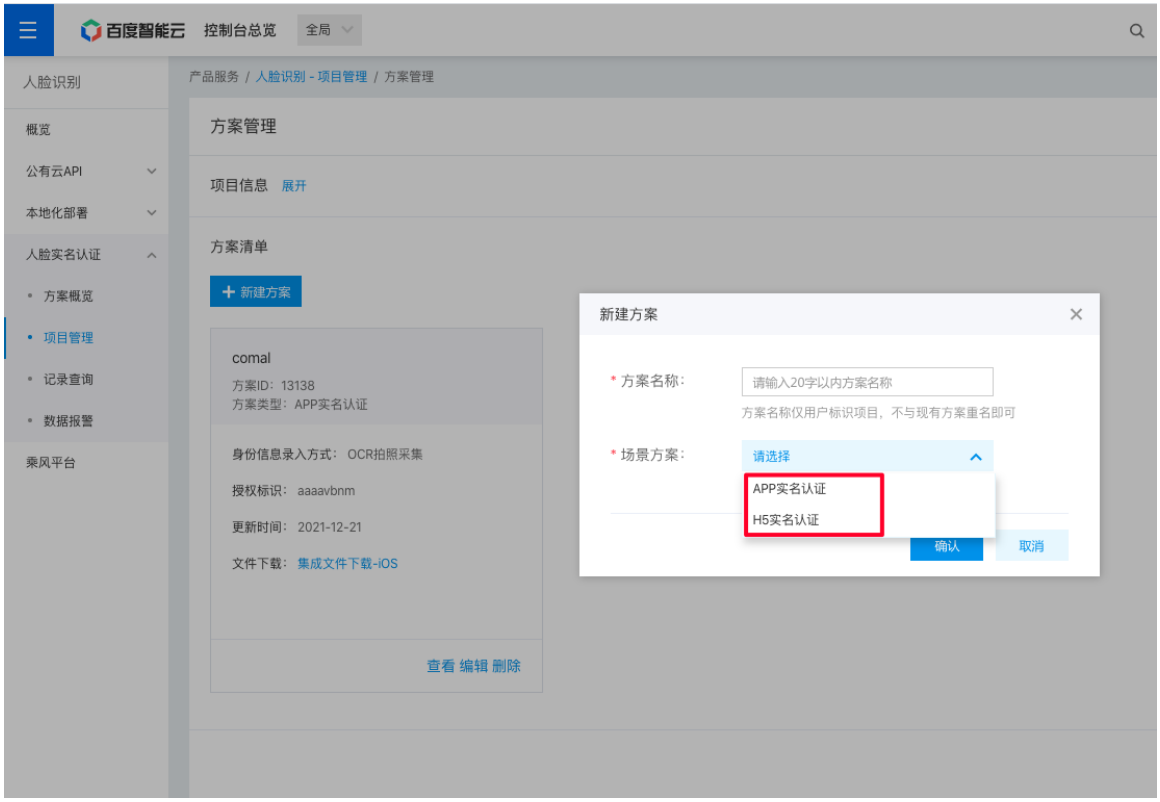
> 创建项目前，请确保您在应用控制台已创建应用，若您未创建应用，请参考STEP2创建应用后，再进行项目创建.



Step4：创建方案

- 项目创建完成后，点击「**方案管理**」进入方案管理页面，在这里您可以为您的项目创建不同的方案，如下图所示。

> 若您的场景为APP场景（Android/IOS/HarmonyOS系统），『场景方案』请选择APP实名认证方案。



4.1 身份信息录入

- 身份信息录入支持选择**用户手动输入**、**OCR拍照采集**及**业务调用时传入身份信息**，如下图所示

> **手动输入**：支持用户手动输入姓名+身份证号信息。
>
> **OCR拍照采集**：会在之后生成的APP方案示例工程内集成OCR采集SDK，在本地进行身份证质量校验，并判断是否为身份证证件。 （OCR拍照采集支持实时采集和相册上传两种形式。推荐您使用实时采集，可以在一定程度增加方案的安全性及便捷性。）
>
> **业务调用时传入身份信息**：指定用户姓名+身份证号进行活体检测及权威库验证流程，需要在采集流程开始前传入姓名+身份证号信息配合使用。具体操作方式可参考方案集成指南。
>
> 注：如您的业务场景无需使用身份信息录入功能，此项可先默认选择，在后续方案集成指南文档中将说明如何去除该部分功能

4.2 授权信息配置

- **授权标识：**自定义SDK的授权信息，作为授权文件的唯一标识，仅支持英文、数字、横线。
- **开发平台：**可多选，根据实际业务需要勾选iOS端或Android端，勾选后需填写对应包名及MD5（仅安卓）。
- **iOS包名：**Bundle ID。
- **Android包名：**Package Name。
- **Android签名MD5：**安卓包签名的keystore文件中私钥的数据摘要，什么是安卓签名MDS，如何获取
- **HarmonyOS包名**：鸿蒙应用bundleName
- **HarmonyOS应用指纹**：鸿蒙应用指纹，鸿蒙应用签名证书的签名文件，什么是鸿蒙应用指纹，如何获取

4.3 方案配置

- **风控：**风控功能开启后，接收SDK端传入的设备指纹信息，对SDK端进行设备风险识别，辨别是否为风险设备，返回识别结果。可有效防御黑产批量虚拟机、病毒侵入等攻击手段，降低第三方黑产攻破概率，提升业务安全性。**该项为必须开启**
- **图像质量控制（SDK）：**分为严格、正常、宽松三个等级，等级越严格，对采集图片的角度、模糊度、遮挡等信息参数把控越高，推荐使用正常。

> 此项配置为人脸采集SDK端对采集图片的质量要求，推荐实名认证场景选择严格或正常模式。图片质量越好，云端接口传输的通过率越高。

质量控制参数	「宽松」	「正常」	「严格」
光照最小值	30	40	60
光照最大值	240	220	200
遮挡-左眼	0.95	0.8	0.4
遮挡-右眼	0.95	0.8	0.4
遮挡-鼻子	0.95	0.8	0.4
遮挡-嘴巴	0.95	0.8	0.4
遮挡-左脸	0.95	0.8	0.4
遮挡-右脸	0.95	0.8	0.4
遮挡-下巴	0.95	0.8	0.4
姿态-俯仰角	30	20	15
姿态-左右角	18	18	15
姿态-旋转角	30	20	15
模糊度	0.8	0.6	0.4

- **活体检测设置（SDK）：**可选取炫瞳活体、动作活体、静默活体等活体检测方案并配置活体阈值。在该过程，SDK 会随机抓取几帧图像进行本地活体检测，检测通过后将图片传至云端进行下一步检测。动作活体支持配置检测动作列表及检测动作数量。
- **图像质量检测（云端）：**分为正常与宽松两个等级，等级设置越严格，对图片角度、模糊度、遮挡等信息参数把控越高，**推荐使用宽松。**
- **活体检测（云端）：**活体检测云端设置分为严格、正常、宽松三个等级，不同等级对应不同的活体检测阈值。等级设置越严格，对活体检测相关参数信息的把控越高。不同等级对应指标可参考下表，**推荐使用正常。**

> **活体检测阈值：**活体检测得分高于此阈值，即判断为活体
>
> **误拒率（FRR）：**指误将活体用户判断为非活体的概率。如误拒率为0.5%，指1000次真人请求，会有5次因为活体分数低于阈值被错误拒绝。

控制度	对应阈值	说明
宽松	0.05	万分之一活体误拒率
正常（推荐）	0.3	千分之一活体误拒率
严格	0.9	百分之一活体误拒率

- **阈值：**用户人脸图片与权威库中人脸的相似度得分阈值，得分超过此阈值，即被判断为同一人。阈值分数相关指标可参考下表，**推荐阈值为80。**

阈值分数	误识率	识别率
60	0.781615%	99.550128%
70	0.096534%	98.307626%
78	**0.015570%（万分之一）**	**95.672664%**
80 （推荐）	**0.009342%（低于万分之一）**	**94.323051%**

Step5：提交方案，获取示例工程

完成上述方案配置后，点击『提交』，进入方案管理页面，下载**Android/IOS/HarmonyOS集成文件（含示例工程）**进行SDK端集成使用。

> **Step4**中方案配置的参数会自动生成至集成文件（含示例工程）中，方便开发使用。
>
> **注意：请谨慎修改APP方案，修改后需要重新下载集成文件进行使用。同时，集成文件中的SDK授权文件（idl-license.face-ios/idl-license.face-android/idl-license.face-harmony）与SDK加密文件（idl-key.face-ios/idl-key.face-android/idl-key.face-harmony）需配套使用，请勿跨方案混用替换。**

至此，方案集成前的准备工作已完成，具体集成技术操作请参考方案集成指南。

Android-方案集成指南

前言

> 【本文】方案集成指南适用于需要集成APP方案，实现**权威库核验**、**自传照片人脸比对**、**仅活体检测**这3个业务需求时来参考。百度人脸实名认证APP方案默认开启风控功能，即使用人脸识别V4系列API接口接受SDK端传入本地环境扫描的设备指纹及安全信息，对SDK端进行设备风险识别，辨别是否为风险设备、并返回识别结果。**此方案可有效防御黑产批量虚拟机、病毒侵入等攻击手段，降低第三方黑产攻破概率，提升业务安全性。**

> 特殊的，如果控制台**风控选项**设定“**关闭**”，则需要配合开通人脸识别V3系列API接口实现**权威库核验**、**自传照片人脸比对**、**仅活体检测**这3个业务需求。

快速定位

- 人脸实名认证（权威源核验）实现方式
- 人脸比对（自传图片）实现方式
- 仅活体检测实现方式
- 核验及计费结果获取，服务端接口

1. 文档说明

文档名称	人脸实名认证APP方案 6.4版本集成文档
所属平台	Android
提交日期	2024-05-14

2. 版本说明

名称	版本号
名镜方案	6.4.0
系统支持	android 5.1+
架构支持	CPU架构平台，armeabi-v7a、arm64-v8a

3. SDK说明

开启大数据风控能力：

文件名称	版本号	说明
lib-LiantianStaticLiteAes-3.9.0.3-release.aar	3.9.0.3	大数据风控能力SDK，封装了设备指纹及安全检测功能
lib-FacePlatform-UI-6.4.3-release.aar	6.4.3	人脸实名认证SDK的UI层，封装活体检测相关UI，以及各平台so库、算法模型
lib-FaceAuthEnhanceSDK-3.0.1-release.aar	3.0 .1	人脸实名认证SDK业务逻辑层封装，封装了人脸实名认证、人脸对比、在线活体
ocr_ui-1.3.0-release.aar	1.3.0	百度OCR身份识别库

未开启大数据风控能力（不推荐）：

文件名称	版本号	说明
faceplatform-release-6.4.aar	6.4	人脸基础SDK
faceplatform-ui-release-6.4.aar	6.4	人脸实名认证SDK的UI层，封装活体检测相关UI，以及各平台so库、算法模型
faceauthbasicsdk-3.0.1-release.aar	3.0.1	人脸实名认证SDK业务逻辑层，封装了人脸实名认证、人脸对比、在线活体
ocr_ui-1.3.0-release.aar	1.3.0	百度OCR身份识别库

注：获取SDK可参考：获取示例工程

4. 配置签名

从百度云控制台下载工程之后，需要在build.gradle中配置签名信息，运行工程检查功能是否正常。

```
signingConfigs { NamedDomainObjectContainer<SigningConfig> it ->
    def alias : String = "androiddebugkey"        签名别名
    def password : String = "android"             签名密码
    def filePath : String = 'signature/facesdk-library.keystore'   签名路径

    debug {
        storeFile file(filePath)
        storePassword password
        keyAlias alias
        keyPassword password
    }
    release {
        storeFile file(filePath)
        storePassword password
        keyAlias alias
        keyPassword password
    }
}
```

5. SDK集成与授权

- 首先在app工程中增加**3. SDK说明**中的依赖库资源

> Notice: 如果需要使用OCR身份证识别能力，则需要增加此ocr-ui-release.aar，如果不使用则不需要增加。在app工程的build.gradle中添加相关依赖，然后点击运行。

- 其次将百度云控制台创建应用时获取的人脸授权文件(**idl-license.face-android**)、大数据风控密钥文件(**idl-key.face-android**)、本地配置文件(**local_config.json**)、质量控制配置文件(**quality_config.json**)放置于Assets目录下。

> 如果使用OCR身份证识别功能，请将OCR身份证识别授权文件（aip.license）也放置于Assets目录下，如下图所示。

- 最后参考控制台下载的工程以及相关接口说明完成SDK集成。

6. 人脸初始化接口（SDK）

初始化接口调用

返回值	API	描述
void	init(Context context, String licenseKey, String licenseName,FaceInitCallback FaceInitCallback)	人脸初始化接口

入参说明

参数	类型	说明
context	Context	上下文
licenseKey	String	授权Key
licenseName	String	授权文件名称

onCallback回调说明

参数	类型	含义	值
resultCode	int	错误码	1000为成功，其他为失败，详情参考resultCode错误码说明
resultMsg	String	详情见resultCode错误码说明	

resultCode错误码说明

resultCode	resultMsg	自查方案
1001	License未初始化	请按照集成文档说明完成SDK初始化
1002	License数据解密失败	请检查License文件是否正确
1003	Licesen数据格式错误	请检查license文件内容有被修改过
1004	License-Key校验错误	请检查工程代码初始化参数中的licenseId，和控制台下载工程里面的licenseId是否匹配
1006	MD5校验错误	请检查工程所使用的签名文件，和控制台填写的签名信息是否匹配
1008	包名（应用名校验错误）	请检查工程代码中的applicationId（包名）和控制台填写的包名是否匹配
1009	过期时间不正确	请提交工单或者线下联系百度产研人员
1011	授权已过期	请查看当前设备时间是否已不在授权文件有效期内
1012	本地文件读取失败	请检查授权文件名称以及路径
1013	远程数据拉取失败	本地鉴权失败，1）请检查工程代码中的applicationId（包名）和控制台填写包名是否匹配；2）请检查工程所使用的签名文件，和控制台填写的签名信息是否匹配
1014	本地时间校验错误	请检查当前设备时间是否早于实际时间

7. 获取verify_token接口（服务端）

本接口为实名认证APP方案的verify_token获取接口，利用所获取的verify_token进行实名认证流程的有效期为**2小时**。

在线调试 您可以在 示例代码中心 中调试该接口，可进行签名验证、查看在线调用的请求内容和返回结果、示例代码的自动生

成。

调用方式

请求URL数据格式

向API服务地址使用POST发送请求，必须在URL中带上参数access_token，可通过后台的API Key和Secret Key生成，具体方式请参考“Access Token获取”。

> **注意**：access_token的有效期为30天，**切记需要每30天进行定期更换，或者每次请求都拉取新token**；

POST中Body的参数，按照下方请求参数说明选择即可。

> **提示**：如果您为百度云老用户，正在使用其他非AI的服务，可以参考百度云AKSK鉴权方式发送请求，虽然**请求方式**和**鉴权方法**和本文所介绍的不同，但请求参数和返回结果一致。

请求说明

注意事项：

- **请求体格式化**：Content-Type为application/json，通过json格式化请求体。

请求示例

HTTP方法：POST

请求URL： https://aip.baidubce.com/rpc/2.0/brain/solution/faceprint/verifyToken/generate

URL参数：

参数	值
access_token	通过API Key和Secret Key获取的access_token,参考“Access Token获取”

Header：

参数	值
Content-Type	application/json

Body中放置请求参数，参数详情如下：

请求参数：

参数	必选	类型	说明
match_source	是	int	方案类型（比对源）；0：实名认证（权威库）；1：人脸比对（上传照片）；2：仅活体检测。
plan_id	是	string	方案的id信息，请在人脸实名认证控制台查看创建的APP方案ID信息

- 请求示例

```
 {
"match_source": 0, //以权威库核验场景为例，需要传入0
"plan_id": 16144,
}
```

返回参数

- 返回结果

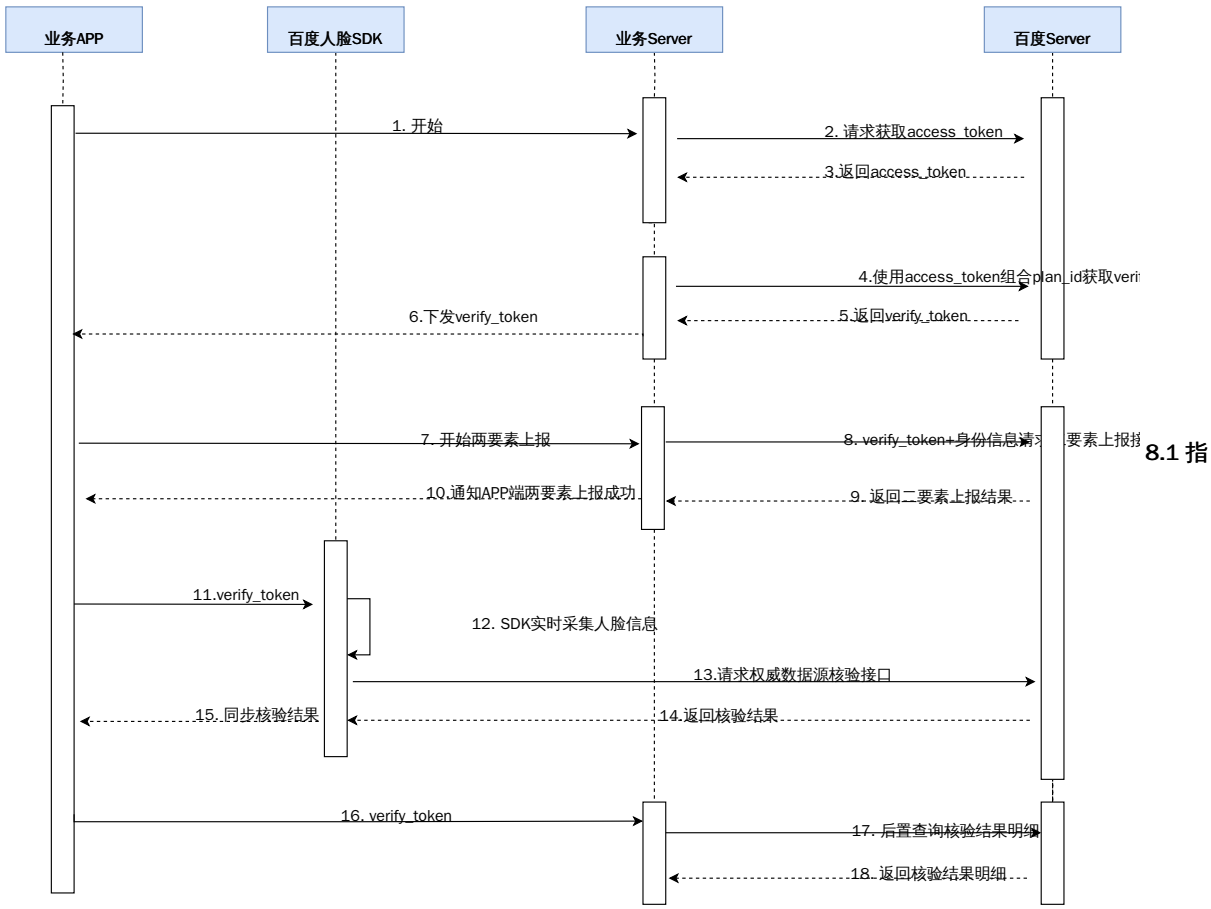
字段	必选	类型	说明
success	是	boolean	返回请求是否成功信息。 若请求成功返回ture; 请求失败则返回false
result	是	object	请求结果
+verify_token	是	string	请求获取的verify_token

- 返回示例

```
{
  "success": true,
  "result": {
     "verify_token": "Yz9rWITm4vak16PBAh5x8oG7"
  },
  "log_id": "1814798895"
}
```

8. 人脸实名认证实现方式

实现方式请参考如下时许图



8.1 指

定用户信息上报接口（服务端） 本接口用于，控制台APP方案中选择身份信息录入方式为**业务调用时传入**，需要先调用此接口输入指定用户的姓名+身份证号信息，再继续请求 **人脸权威库核验接口（SDK）**

在线调试 您可以在 示例代码中心 **中调试该接口**，可进行签名验证、查看在线调用的请求内容和返回结果、示例代码的自动生成。 **调用方式**

请求URL数据格式

向API服务地址使用POST发送请求，必须在URL中带上参数access_token，可通过后台的API Key和Secret Key生成，具体方式请参考“Access Token获取”。

> **注意**：access_token的有效期为30天，**切记需要每30天进行定期更换，或者每次请求都拉取新token**；

POST中Body的参数，按照下方请求参数说明选择即可。

> **提示**：如果您为百度云老用户，正在使用其他非AI的服务，可以参考百度云AKSK鉴权方式发送请求，虽然**请求方式**和**鉴权方法**和本文所介绍的不同，但请求参数和返回结果一致。

请求说明

注意事项：

- **请求体格式化**：Content-Type为application/json，通过json格式化请求体。

请求示例

HTTP方法：POST

请求URL：https://aip.baidubce.com/rpc/2.0/brain/solution/faceprint/idcard/submit

URL参数：

参数	值
access_token	通过API Key和Secret Key获取的access_token,参考“Access Token获取”

Header：

参数	值
Content-Type	application/json

Body中放置请求参数，参数详情如下：

参数	必选	类型	值
verify_token	是	string	通过access_token获取的verify_token
id_name	是	string	指定输入用户的姓名信息
id_no	是	string	指定输入用户的身份证件号信息
certificate_type	否	int	证件类型： 0 大陆居民二代身份证 4 港澳台居民居住证

- 请求示例：

```
{
  "verify_token": "2sF3nE5mXOHkx2aQwWG4n5WI",
  "id_name": "张三",
  "id_no": "500***********3390",
  "certificate_type": 0          // 证件类型：0:大陆居民二代身份证,4:港澳台居民居住证
}
```

返回参数

- 返回结果

字段	必选	类型	说明
success	是	boolean	返回请求是否成功信息。 若请求成功返回ture; 请求失败则返回false
result	是	int	请求结果，返回固定结果1，可忽略
log_id	是	string	本次查询接口请求的logid

- 返回示例

```
{
  "success": true,
  "result": 1,
  "log_id": "1244068892"
}
```

8.2 人脸权威库核验接口（SDK） 基于姓名、身份证号、当前SDK获取的人脸图片，与权威库进行对比，并得出比对分数，并基于此进行业务判断是否为同一人。

返回值	API	描述
void	startFaceVerify(Context context, Map params, FaceServiceCallbck FaceServiceCallbck)	SDK核验接口

入参params (HashMap类型) key值列表如下：

参数	类型	说明	必选
verify_token	String	需要APP服务端通过AK、SK获取access_token，参考 https://ai.baidu.com/docs#/Auth/top,然后通过access_token获取verify_token **【此处需要注意】verify_token有效期两个小时，建议每次调用接口重新获取verify_token**	是
plan_id	String	在控制台配置的方案Id	否
certificate_type	int	证件类型；0：中国居民二代身份证，1：港澳台来往内地通行证，2：外国人永久居留证，3：定居国外的中国公民护照，4：港澳台居民居住证。默认为0	否
match_source	int	方案类型（比对源）；0：实名认证（权威库）；1：人脸比对（上传照片）；2：仅活体检测；**这里默认传入0**	是

onCallback回调说明

参数	类型	含义	值
resultCode	int	错误码	0为成功，其他为失败，详情参考resultCode错误码说明
resultMap	HashMap	回调结果Map	详情见下表

resultMap key值列表说明：

Key值	类型	含义
resultMsg	String	详情见resultCode错误码说明
data	String	服务端返回结果json,只有resultCode为0时会返回 此字段

- resultCode和resultMsg说明

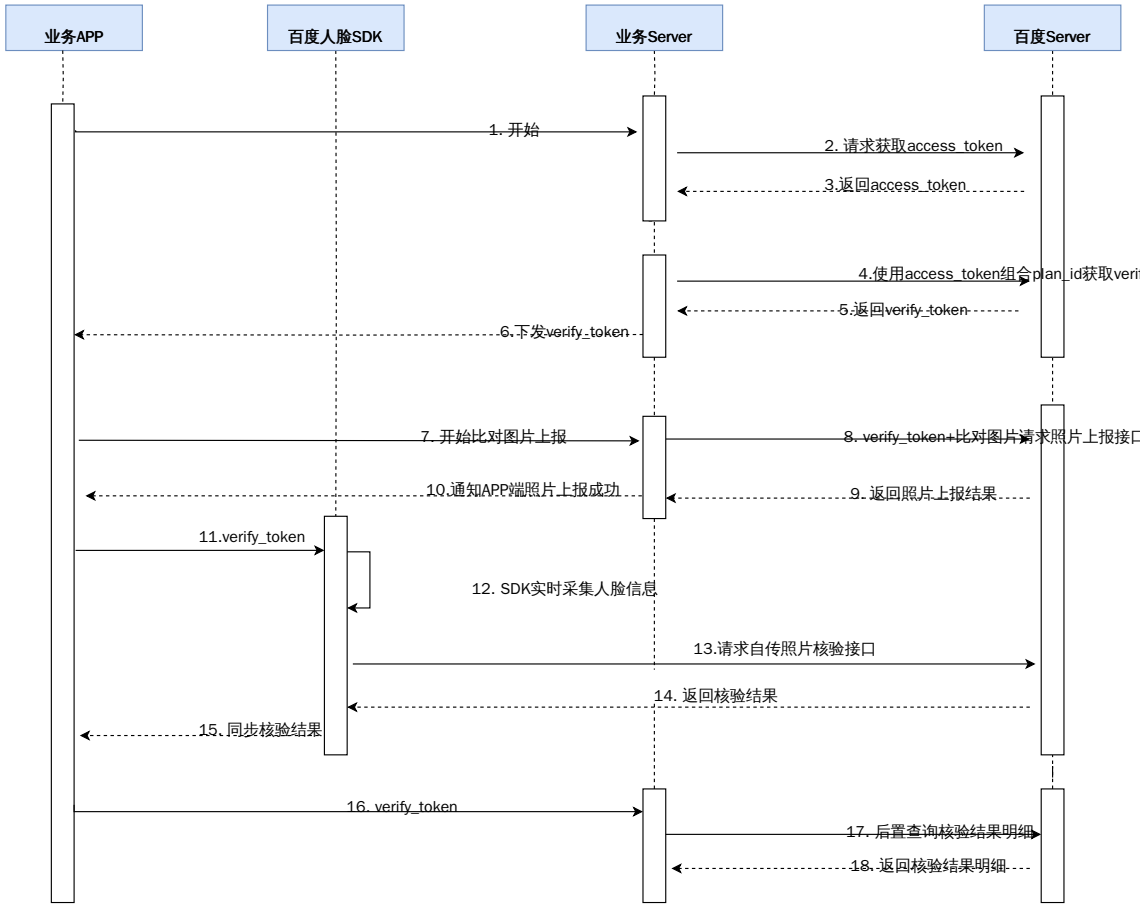
resultCode	resultMsg	自查方案
-102	采集流程取消	用户自行取消采集流程
-103	SDK未初始化	需要检查初始化方法有没有调用，建议在application里面调用
-303	视频录制错误	是否授权视频/音频
-310	摄像头调起失败	请检查摄像头权限
-401	超时	可通过设置延长超时时间限制
-402	炫瞳异常	炫瞳打光颜色与模型输出不匹配
-1001	网络异常	请检查网络通讯

- data返回示例

```
{
 "error_code": 0，//认证建议结果为通过，该结果仅供参考；可通过调用服务端getall接口获取最终认证结果
 "log_id": "1790651229035123218"
}
```

9. 人脸对比（自传照片）实现方式

实现方式请参考如下时许图



比图片上传接口（服务端） 本接口适用于人脸比对（自传照片）业务场景，为了保证用户信息安全，您需要提前通过服务端请求本接口来上报比对人脸信息。

在线调试 您可以在 示例代码中心 中调试该接口，可进行签名验证、查看在线调用的请求内容和返回结果、示例代码的自动生成。 **调用方式**

请求URL数据格式

向API服务地址使用POST发送请求，必须在URL中带上参数access_token，可通过后台的API Key和Secret Key生成，具体方式请参考“Access Token获取”。

> **注意**：access_token的有效期为30天，**切记需要每30天进行定期更换，或者每次请求都拉取新token**；

POST中Body的参数，按照下方请求参数说明选择即可。

> **提示**：如果您为百度云老用户，正在使用其他非AI的服务，可以参考百度云AKSK鉴权方式发送请求，虽然**请求方式**和**鉴权方法**和本文所介绍的不同，但请求参数和返回结果一致。

请求说明

注意事项：

- **请求体格式化**：Content-Type为application/json，通过json格式化请求体。

请求示例

HTTP方法：POST

请求URL： https://aip.baidubce.com/rpc/2.0/brain/solution/faceprint/uploadMatchImage

URL参数：

参数	值
access_token	通过API Key和Secret Key获取的access_token,参考“Access Token获取”

Header：

参数	值
Content-Type	application/json

Body中放置请求参数，参数详情如下：

参数	必选	类型	值
verify_token	是	string	通过access_token获取的verify_token
image	是	string	图片base64字符串，编码后的图片大小不超过10M，图片分辨率小于1920*1080
quality_control	否	string	“NONE”、“LOW”、“NORMAL”、“HIGH”； 质量控制参数，未主动传入时默认为“NONE”
liveness_control	否	string	“NONE”、“LOW”、“NORMAL”、“HIGH”； 活体控制参数，未主动传入时默认为“NONE”

- 请求示例：

```
{
 "verify_token": "2sF3nE5mXOHkx2aQwWG4n5WI",
 "image": "/9j/4AAQSkZJRgABAQAAAQABAAD/",
 "quality_control" : "LOW",
 "liveness_control" : "LOW",
}
```

返回参数

- 返回结果

字段	必选	类型	说明
success	是	boolean	返回请求是否成功信息。 若请求成功返回ture; 请求失败则返回false
log_id	是	string	logid
result	是	object	返回结果

- 返回示例

```
{
 "success": true,
 "result": null,
 "log_id": "1329130892"
}
```

9.2 人脸对比（SDK） 包含活体检测、质量检测、人脸1:1识别功能，可通过传入参数进行控制，如您的业务场景核心为人脸对比（自传照片）。

返回值	API	描述
void	startFaceVerify(Context context, FaceServiceCallbck FaceServiceCallbck)	SDK核验接口

入参params (HashMap类型) key值列表如下：

参数	类型	说明	必选
verify_token	String	需要APP服务端通过AK、SK获取access_token，参考 https://ai.baidu.com/docs#/Auth/top,然后通过access_token获取verify_token **【此处需要注意】verify_token有效期两个小时，建议每次调用接口重新获取verify_token**	是
match_source	int	方案类型（比对源）；0：实名认证（权威库）；1：人脸比对（上传照片）；2：仅活体检测；**这里默认传入1**	是

onCallback回调说明

参数	类型	含义	值
resultCode	int	错误码	0为成功，其他为失败，详情参考实名认证resultCode错误码说明
resultMap	HashMap	回调结果Map	详情见下表

resultMap key值列表说明：

Key值	类型	含义
resultMsg	String	详情见resultCode错误码说明
data	String	服务端返回结果json,只有resultCode为0时会返回 此字段

- resultCode和resultMsg说明

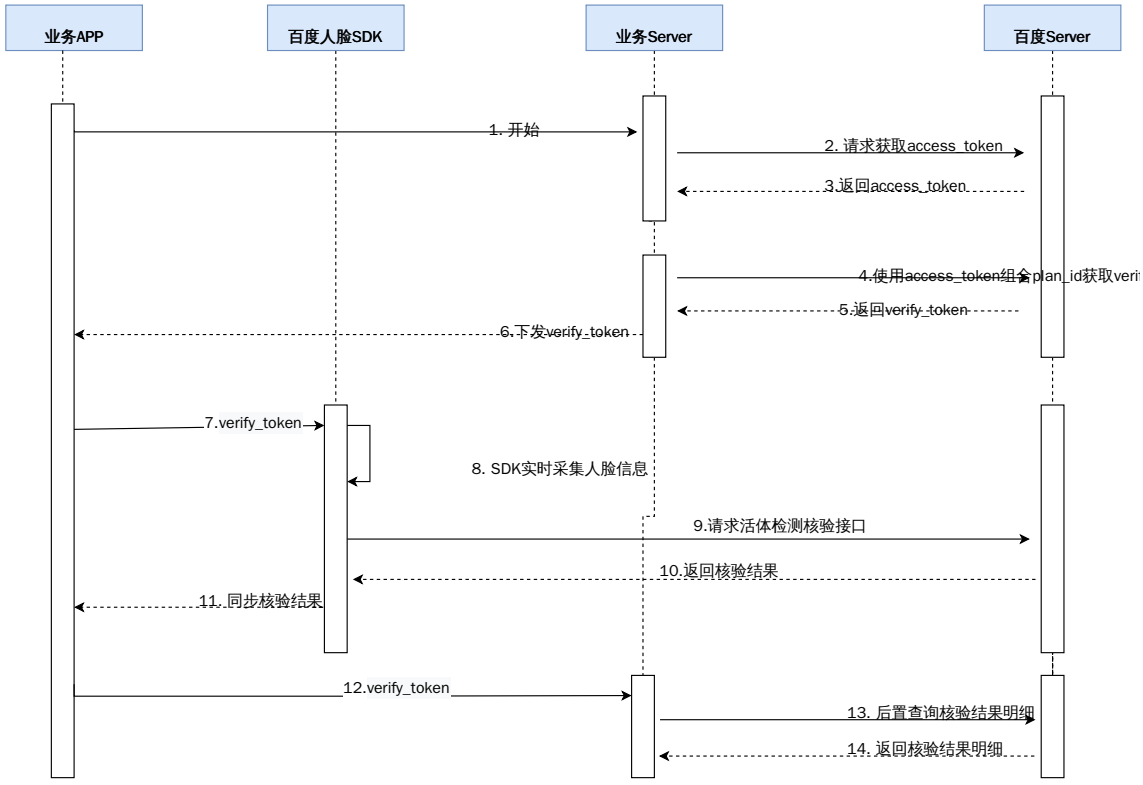
resultCode	resultMsg	自查方案
-102	采集流程取消	用户自行取消采集流程
-103	SDK未初始化	需要检查初始化方法有没有调用，建议在application里面调用
-303	视频录制错误	是否授权视频/音频
-310	摄像头调起失败	请检查摄像头权限
-401	超时	可通过设置延长超时时间限制
-402	炫瞳异常	炫瞳打光颜色与模型输出不匹配
-1001	网络异常	请检查网络通讯

- data返回示例

```
{
 "error_code": 0，//认证建议结果为通过，该结果仅供参考；可通过调用服务端getall接口获取最终认证结果
 "log_id": "1790651229035123218"
}
```

10. 人脸活体检测实现方式

实现方式请参考如下时许图



10.1 活体检测（SDK）

包含本地活体加云端活体，本地活体分静默活体、炫瞳活体、动作活体三种，云端活体可以判断图片中的人脸是否为二次翻拍以及是否为合成图攻击。实现二次验证采集图片是否存在假体攻击破绽的情况。
如您的业务场景核心为人脸实名认证（权威源），请直接参考 **8.2 人脸权威库核验接口（SDK）** 。

返回值	API	描述
void	startFaceVerify(Context context, Map params, FaceServiceCallbck FaceServiceCallbck)	SDK核验接口

入参params (HashMap类型) key值列表如下：

参数	类型	说明	必选
verify_token	String	需要APP服务端通过AK、SK获取access_token，参考 https://ai.baidu.com/docs#/Auth/top,然后通过access_token获取verify_token **【此处需要注意】verify_token有效期两个小时，建议每次调用接口重新获取verify_token**	是
match_source	int	方案类型（比对源）；0：实名认证（权威库）；1：人脸比对（上传照片）；2：仅活体检测；**这里默认传入2.**	是

onCallback回调说明

参数	类型	含义	值
resultCode	int	错误码	0为成功，其他为失败，详情参考实名认证resultCode错误码说明
resultMap	HashMap	回调结果Map	详情见下表

resultMap key值列表说明：

Key值	类型	含义
resultMsg	String	详情见resultCode错误码说明
data	String	服务端返回结果json,只有resultCode为0时会返回 此字段

- resultCode和resultMsg说明

resultCode	resultMsg	自查方案
-102	采集流程取消	用户自行取消采集流程
-103	SDK未初始化	需要检查初始化方法有没有调用，建议在application里面调用
-303	视频录制错误	是否授权视频/音频
-310	摄像头调起失败	请检查摄像头权限
-401	超时	可通过设置延长超时时间限制
-402	炫瞳异常	炫瞳打光颜色与模型输出不匹配
-1001	网络异常	请检查网络通讯

- data返回示例

```
{
 "error_code": 0, //认证建议结果为通过，该结果仅供参考；可通过调用服务端getall接口获取最终认证结果
 "log_id": "1790651229035123218"
}
```

11. 验证后查询接口

11.1 获取认证人脸接口（服务端）

本接口返回进行人脸实名认证过程中进行认证的最终采集的人脸信息。根据Verify_token返回的结果信息会在云端保留**3天**，您可根据需要在此期间进行调取查询。

在线调试 您可以在 示例代码中心 中调试该接口，可进行签名验证、查看在线调用的请求内容和返回结果、示例代码的自动生成。

调用方式

请求URL数据格式

向API服务地址使用POST发送请求，必须在URL中带上参数access_token，可通过后台的API Key和Secret Key生成，具体方式请参考“Access Token获取”。

> **注意**：access_token的有效期为30天，**切记需要每30天进行定期更换，或者每次请求都拉取新token**；

POST中Body的参数，按照下方请求参数说明选择即可。

> **提示**：如果您为百度云老用户，正在使用其他非AI的服务，可以参考百度云AKSK鉴权方式发送请求，虽然**请求方式**和**鉴权方法**和本文所介绍的不同，但请求参数和返回结果一致。

请求说明

注意事项：

- **请求体格式化**：Content-Type为application/json，通过json格式化请求体。

请求示例

HTTP方法：POST

请求URL： https://aip.baidubce.com/rpc/2.0/brain/solution/faceprint/result/simple

URL参数：

参数	值
access_token	通过API Key和Secret Key获取的access_token,参考“Access Token获取”

Header：

参数	值
Content-Type	application/json

Body中放置请求参数，参数详情如下：

参数	值
verify_token	通过access_token获取的verify_token

- 请求示例：

```
{
"verify_token":"clupeyP51sn28XzxGVTfYqoN"
}
```

返回参数

- 返回结果

字段	必选	类型	说明
success	是	boolean	返回请求是否成功信息。 若请求成功返回ture; 请求失败则返回false
result	是	object	请求结果
+image	是	string	返回采集的用户人脸信息

- 返回示例

```
{
  "success": true,
  "result": {
     "image":"https://brain.baidu.com/solution/faceprint/image/query?verify_token=xxxxxx"
  },
  "log_id": "1054986003"
}
```

11.2 查询认证结果接口

本接口为请求返回的认证结果信息查询，包含用户二次确认的身份证信息，活体检测信息、及用户对权威库图片进行比对的分数信息。（仅在认证成功时返回上述信息，认证失败返回错误码）根据Verify_token返回的结果信息会在云端保留**3天**，您可根据需要在此期间进行调取查询。

在线调试 您可以在 示例代码中心 中调试该接口，可进行签名验证、查看在线调用的请求内容和返回结果、示例代码的自动生成。 **调用方式**

请求URL数据格式

向API服务地址使用POST发送请求，必须在URL中带上参数`access_token`，可通过后台的API Key和Secret Key生成，具体方式请参考“Access Token获取”。

> **注意**：`access_token`的有效期为30天，**切记需要每30天进行定期更换，或者每次请求都拉取新token**；

POST中Body的参数，按照下方请求参数说明选择即可。

> **提示**：如果您为百度云老用户，正在使用其他非AI的服务，可以参考百度云AKSK鉴权方式发送请求，虽然**请求方式**和**鉴权方法**和本文所介绍的不同，但请求参数和返回结果一致。

请求说明

注意事项：

- **请求体格式化**：Content-Type为`application/json`，通过`json`格式化请求体。

请求示例

HTTP方法：`POST`

请求URL：`https://aip.baidubce.com/rpc/2.0/brain/solution/faceprint/result/detail`

URL参数：

参数	值
access_token	通过API Key和Secret Key获取的access_token,参考“Access Token获取”

Header：

参数	值
Content-Type	application/json

Body中放置请求参数，参数详情如下：

参数	值
verify_token	通过access_token获取的verify_token

请求示例：

```
 {
  "verify_token"："clupeyP51sn28XzxGVTfYqoN"
}
```

返回参数

- 返回结果

字段	必选	类型	说明
success	是	boolean	返回本次身份核验是否成功的结果。 成功返回ture; 失败则返回false
result	是	object	请求结果
+verify_result	是	object	认证返还信息
++verify_log_id	是	string	核验结果对应的日志ID
++liveness_score	是	float	**活体检测分数：** 活体验证通过时返回活体分数，不通过则返回0。
++score	是	float	人脸实名认证比对得分**（仅活体检测时返回为0）**
++risk_level	否	string	安全风控等级 方案配置启用安全风控，将返回该字段 值为1或2时表示触发了安全风险，值为3或4时无风险
++risk_tag	否	string	安全风控标签 方案配置启用安全风控，将返回该字段 当risk_level值为1或2时，返回具体风险类型，risk_level值为3或4时，为空
++spoofing	是	float	合成图分数 若未进行合成图检测，则返回0 若进行活体检测，则返回合成图检测分值
+idcard_confirm	是	object	用户二次确认的身份证信息
++name	是	string	姓名
++idcard_number	是	string	身份证号
code	否	int	错误码 **（仅认证失败时展示）**
message	否	string	错误信息 **（仅认证失败时展示）**
log_id	是	string	本次查询接口请求的日志ID

- 返回示例

```
{
 "success": true,
 "result": {
    "verify_result": {
        "verify_log_id": "1868893431661688963",
        "score": 94.57,
        "risk_level": "3",
        "risk_tag": "[]",
        "liveness_score": 0.99983007,
        "spoofing": 0.0
    },
    "idcard_confirm": {
        "idcard_number": "430419********4532",
        "name": "陈**"
    }
 },
 "log_id": "1868914111329362938"
}
```


- 返回失败示例


```
{
"success": false,
"result": {
    "verify_result": {
        "verify_log_id": "1866379694929155276",
        "score": 0.0052550267,
        "risk_level": "3",
        "risk_tag": "[]",
        "liveness_score": 0.0052550267,
        "spoofing": 0.0
    },
    "idcard_confirm": {
        "idcard_number": null,
        "name": null
    }
},
"code": 223120,
"message": "活体检测未通过",
"log_id": "1866379737400667552"
}
```


11.3 核验及计费信息获取（服务端）

根据Verify_token返回的信息会在云端保留**3天**，您可根据需要在此期间进行调取查询。

调用方式

请求URL数据格式

向API服务地址使用POST发送请求，必须在URL中带上参数`access_token`，可通过后台的API Key和Secret Key生成，具体方式请参考“Access Token获取”。

> **注意**：`access_token`的有效期为30天，**切记需要每30天进行定期更换，或者每次请求都拉取新token**；

POST中Body的参数，按照下方请求参数说明选择即可。

> **提示**：如果您为百度云老用户，正在使用其他非AI的服务，可以参考百度云AKSK鉴权方式发送请求，虽然**请求方式**和**鉴权**

> **方法**和本文所介绍的不同，但请求参数和返回结果一致。

在线调试 您可以在 示例代码中心 中调试该接口，可进行签名验证、查看在线调用的请求内容和返回结果、示例代码的自动生成。

请求说明

注意事项：

- **请求体格式化**：Content-Type为`application/json`，通过`json`格式化请求体。

请求示例

HTTP方法：`POST`

请求URL：`https://aip.baidubce.com/rpc/2.0/brain/solution/faceprint/result/getall`

URL参数：

参数	值
access_token	通过API Key和Secret Key获取的access_token,参考“Access Token获取”

Header：

参数	值
Content-Type	application/json

Body中放置请求参数，参数详情如下：

参数	值
verify_token	通过access_token获取的verify_token

请求示例：

```
{
 "verify_token" : "clupeyP51sn28XzxGVTfYqoN"
}
```

返回参数

- 返回结果

字段	必选	类型	说明
success	是	boolean	返回请求是否成功信息。 若请求成功返回ture; 请求失败则返回false
log_id	是	string	本次查询接口请求的logid
result	是	object	结果信息
+verify_count	是	int	当前verify_token对应的核验次数，核验次数为几，后面就有几个对象数组 如方案配置为“认证未通过时URL失效”，则该verify_token核验次数为1； 如方案配置为“认证未通过时URL不失效”，则该verify_token对应的核验次数可能大于1（用户核验失败后多次重试）
+ocr_charge_count	是	int	身份证识别OCR接口的计费次数

+match_charge_count	是	int	人脸对比接口的计费次数
+verify_charge_count	是	int	人脸实名认证接口的计费次数
+verify_result	是	object[]	核验结果信息，verify_count的值为几，就返回几个该对象
++order	是	int	代表本次核验结果在verify_token所有核验次数中的顺序，按时间先后排序，第一次核验返回值为 1，第二次核验返回值为 2，以此类推
++is_verify_passed	是	boolean	本次核验是否通过 核验成功返回true 核验失败返回false
++code	是	string	本次核验的错误码 核验成功时返回 0 核验失败返回非0错误码
++message	是	string	本次核验的错误信息 核验成功时返回“核验成功” 核验失败时返回具体错误原因，如“身份证姓名不匹配”
++verify_time	是	string	本次核验完成的时间点，包含年月日、时分秒，如“2023-6-20 18:00:00”
++is_charged	是	boolean	本次核验是否计费 计费返回true 不计费返回false
++charge_type	是	string	计费类型：**verify**(人脸实名认证)、**match**(人脸对比)、**live**(在线图片活体)
++charge_time	是	string	本次计费的时间点，如不计费则该字段为空，包含年月日、时分秒，如“2023-6-20 18:00:00”
++is_ocr_charge	否	boolean	身份证识别OCR是否计费 计费返回true 不计费返回false
++ocr_charge_time	否	string	ocr 收费时间点，如不收费则该字段为空，包含年月日、时分秒，如“2023-6-20 18:00:00”
++ocr_count	否	int	ocr次数
++verify_detail	是	object	核验的详细信息
+++face_image	是	string	本次核验流程中采集的人脸最佳质量图下载url
+++verify_log_id	是	string	本次核验流程中请求的人脸实名认证（方案配置权威库比对）、人脸对比（方案配置自建人脸库比对）、在线图片活体（仅活体检测）后端接口logid，支持在记录查询平台中通过logid查询到3天内的记录。 少数核验失败情况下，实际并未发生上述俩接口的请求，则该字段为空，如：用户拒绝摄像头授权且不允许降级
+++score	是	float	人脸相似度得分，大于等于方案设置阈值为同一人；当身份证姓名不匹配或活体不通过时，该项为空

			项为空
+++threshold	是	float	本次核验时，所使用的方案配置相似度阈值
+++liveness_score	是	float	活体检测得分，注：预留功能字段，当前返回为0
+++spoofing_score	否	float	方案配置使用合成图功能，将返回合成图得分，注：预留功能字段，当前返回为0
+++risk_level	否	string	安全风控等级 方案配置启用安全风控，将返回该字段 值为1或2时表示触发了安全风险，值为3或4时无风险
+++risk_tag	否	string	安全风控标签 方案配置启用安全风控，将返回该字段 当risk_level值为1或2时，返回具体风险类型，risk_level值为3或4时，为空
+++is_demote	否	boolean	H5方案相关，可忽略
++idcard_confirm	是	object	用户手动输入或二次确认的身份证信息
+++name	是	string	姓名
+++idcard_number	是	string	证件号
+++idcard_type	是	string	证件类型，大陆居民二代身份证返回0
++idcard_ocr_result	否	object	OCR采集的身份证信息，当方案配置使用OCR采集证件照时返回该参数
++idcard_images	否	object	返回采集的身份证图片信息 当人脸实名认证控制台设置为使用OCR识别时返回此参数信息

- 返回示例

```
{
  "success": true,
  "result": {
    "verify_count": 1,
    "ocr_charge_count": 0,
    "match_charge_count": 0,
    "verify_charge_count": 1,
    "verify_result": [
      {
        "order": 1,
        "code": "0",
        "message": "核验成功",
        "is_verify_passed": true,
        "verify_time": "2024-04-02 16:15:01",
        "is_charged": true,
        "charge_type": "verify",
        "charge_time": "2024-04-02 16:15:01",
        "is_ocr_charge": false,
        "ocr_count": 0,
        "ocr_charge_time": null,
        "verify_detail": {
          "score": 84.4000015258789,
          "threshold": 80.0,
          "app": true,
          "face_image": "https://bj.bcebos.com/v1/mingjing-qa/hb/h5-video-temp/temp/202404/16130/ZWCCBxXUz48Q9wuR/lgi1n0SpQRtzm9ySWLQwdwvn-00.jpg?authorization=bce-auth-v1%2F89e9208d0d5a497cbc8922a4d74d4f71%2F2024-04-02T08%3A19%3A23Z%2F86400%2F%2Fb3abf9eef0b7f897ccf0350d7d59bbfd4498ab214ef73d437a45e319e5028688",

          "verify_log_id": "1775074433891895605",
          "liveness_score": 0.99999183,
          "spoofing_score": null,
          "risk_level": null,
          "risk_tag": null,
          "is_demote": false
        },
        "idcard_confirm": {
          "name": "李**",
          "idcard_number": "14052219981234567",
          "idcard_type": "0",
          "idcard_ocr_result": null,
          "idcard_images": null
        }
      }
    ]
  },
  "log_id": "1775075543214005842"
}
```


12. 人脸释放接口（SDK）

人脸释放接口调用，实现对采集功能、模型的释放，减小内存。

返回值	API	描述
void	release()	人脸释放接口

13. OCR身份证识别相关接口

13.1 OCR身份证识别-初始化接口

返回值	API	描述
void	initAccessToken(OnResultListener listener, Context context)	OCR初始化接口

入参说明

参数	类型	含义
context	Context	上下文

onError回调参数说明：

参数	类型	含义
errorCode	int	服务端返回错误码，详情见鉴权机制接口：https://ai.baidu.com/ai-doc/REFERENCE/Ck3dwjhhu
errorMessage	String	服务端返回错误信息，详情见鉴权机制接口：https://ai.baidu.com/ai-doc/REFERENCE/Ck3dwjhhu

13.2 OCR身份证识别-识别接口

支持对二代居民身份证字段进行结构化识别，包括姓名、性别，调用参考OCR身份证识别接口文档。

返回值	API	描述
void	startOcrRecognize(Context context, OcrConfig ocrConfig, OcrRecognizeCallback ocrRecognizeCallback)	OCR识别接口

入参说明

参数	类型	含义
context	Context	上下文
ocrConfig	OcrConfig	OCR配置类

OcrConfig配置字段说明

参数	类型	含义	
ocrPageNavigationColor	int	OCR页面导航栏背景色，默认为白色	
ocrPageTitleText	String	OCR标题内容，默认为"身份信息采集"	
ocrPageTitleColor	int	OCR标题颜色，默认为黑色	
ocrPageTopText	String	OCR顶部扫描文字，默认为"请将您本人的\n身份证人像面放入框内"	
ocrPageTopTextColor	int	OCR 顶部扫描文字颜色，默认为白色	

onError回调参数说明

参数	类型	含义	值
errorCode	int	错误码	服务端返回错误码，详情见在线身份证识别接口：https://ai.baidu.com/ai-doc/OCR/rk3h7xzck
errorMessage	String	回调结果Map	服务端返回错误信息，详情见在线身份证识别接口：https://ai.baidu.com/ai-doc/OCR/rk3h7xzck

13.3 不使用OCR，只使用人脸相关能力 不使用OCR，可以删除ocr-ui.aar、aip.license，以及OCR初始化以及调用相关代码。

代码混淆

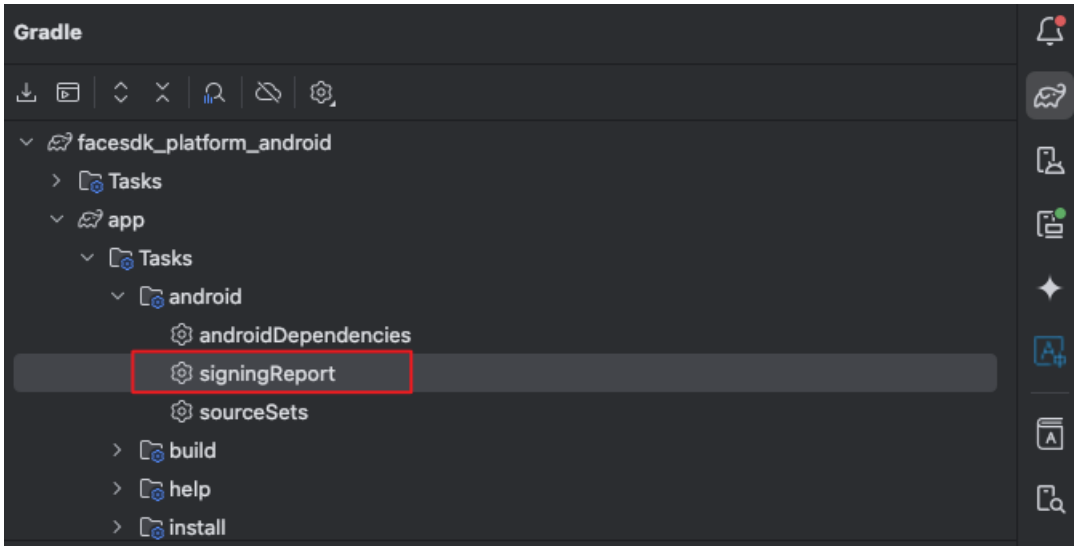
```
 -dontwarn com.baidu.idl.**
-keep class com.baidu.idl.** { *; }
-dontwarn com.baidu.vis.**
-keep class com.baidu.vis.** { *; }
-dontwarn com.baidu.liantian.**
-keep class com.baidu.liantian.** { *; }
-dontwarn com.baidu.protect.**
-keep class com.baidu.protect.** { *; }
-dontwarn com.baidu.ocr.**
-keep class com.baidu.ocr.** { *; }
```

权限

名称	说明	必选
需要动态申请的权限		
android.permission.CAMERA	拍照权限	是
android.permission.READ_EXTERNAL_STORAGE	读取手机外部存储权限（安全相关、OCR相关）	否
android.permission.READ_EXTERNAL_STORAGE	写入手机外部存储权限（安全相关、OCR相关）	否
不需要动态申请的权限		
android.permission.INTERNET	允许访问网络	是
android.permission.ACCESS_NETWORK_STATE	获取网络状态权限	是
android.permission.READ_PHONE_STATE	允许访问电话状态权限	是
android.hardware.camera.autofocus	允许相机对焦（OCR相关）	否
android.permission.ACCESS_WIFI_STATE	获取wifi权限	是
android.permission.WAKE_LOCK	屏幕常亮权限	是

常见问题

如何获取应用签名MD5 （1）方法1，通过Android Studio Gradle工具面板，运行signingReport方式获取，如下图



（2）方法2，通过运行keytool -list -v -keystore XXX.keystore命令方式来获取，如下图

```
keytool -list -v -keystore XXX.keystore
```

（3）方法3，通过运行keytool -printcert -jarfile XXX.apk命令方式来获取，如下图

```
keytool -printcert -jarfile XXX.apk
```

verify_token的详细说明 （1）access_token 有效期为 30 天，重复生成 access_token 的话，对之前的不影响，依旧能使用；access_token 与verify_token 是包含关系，即 verify_token 是由 access_token生成的，如果一个 verify_token 是和一个不匹配的 access_token 使用，会提示“Token无效或者已过期”

（2）verify_token 的核验有效期为 2 小时，在有效期内可以进行了核验动作，如果超过了 2 小时没有使用该 token 进行核验的话会提示“Token无效或者已过期”

（3） verify_token 核验完毕后（核验成功，或设置了认证未通过时URL失效），无法再进行核验，如果再次进行核验，会提示“Token无效或者已过期”

（4） verify_token 核验完毕后（核验成功，或设置了认证未通过时URL失效），支持对后验接口的查询，有效期均为 3 天，3 天后再次查询后验接口，会提示“Token无效或者已过期” **获取认证人脸、查询认证结果、核验及计费信息获取接口持续返回错误码18，提示openapi限制，且有返回logid 答：**检查APP方案依赖的“人脸实名认证V3/V4”、“人脸对比V3/V4”、“在线图片活体检测V3/V4” 三项付费接口，是否有免费额度，或者直接开通后付费，以消除该报错提示。开通后，再进行重试。

IOS-方案集成指南

前言

> 【本文】方案集成指南适用于需要集成APP方案，实现**权威库核验**、**自传照片人脸比对**、**仅活体检测**这3个业务需求时来参考。百度人脸实名认证APP方案默认开启风控功能，即使用人脸识别V4系列API接口接受SDK端传入本地环境扫描的设备指纹及安全信息，对SDK端进行设备风险识别，辨别是否为风险设备、并返回识别结果。**此方案可有效防御黑产批量虚拟机、病毒侵入等攻击手段，降低第三方黑产攻破概率，提升业务安全性。**

> 特殊的，如果控制台**风控选项**设定“**关闭**”，则需要配合开通人脸识别V3系列API接口实现**权威库核验**、**自传照片人脸比对**、**仅活体检测**这3个业务需求。

快速定位

- 人脸实名认证（权威库核验）实现方式
- 人脸比对（自传图片）
- 仅活体检测实现方式
- 核验及计费结果获取，服务端接口

1. 文档说明

文档名称	人脸实名认证APP方案 6.4版本集成文档
所属平台	iOS
提交日期	2024-09-25

2. 版本说明

名称	版本号
名镜方案	6.4.0
系统支持	iOS 9.0 +
架构	arm64
IDE	Xcode 14+

3. SDK说明

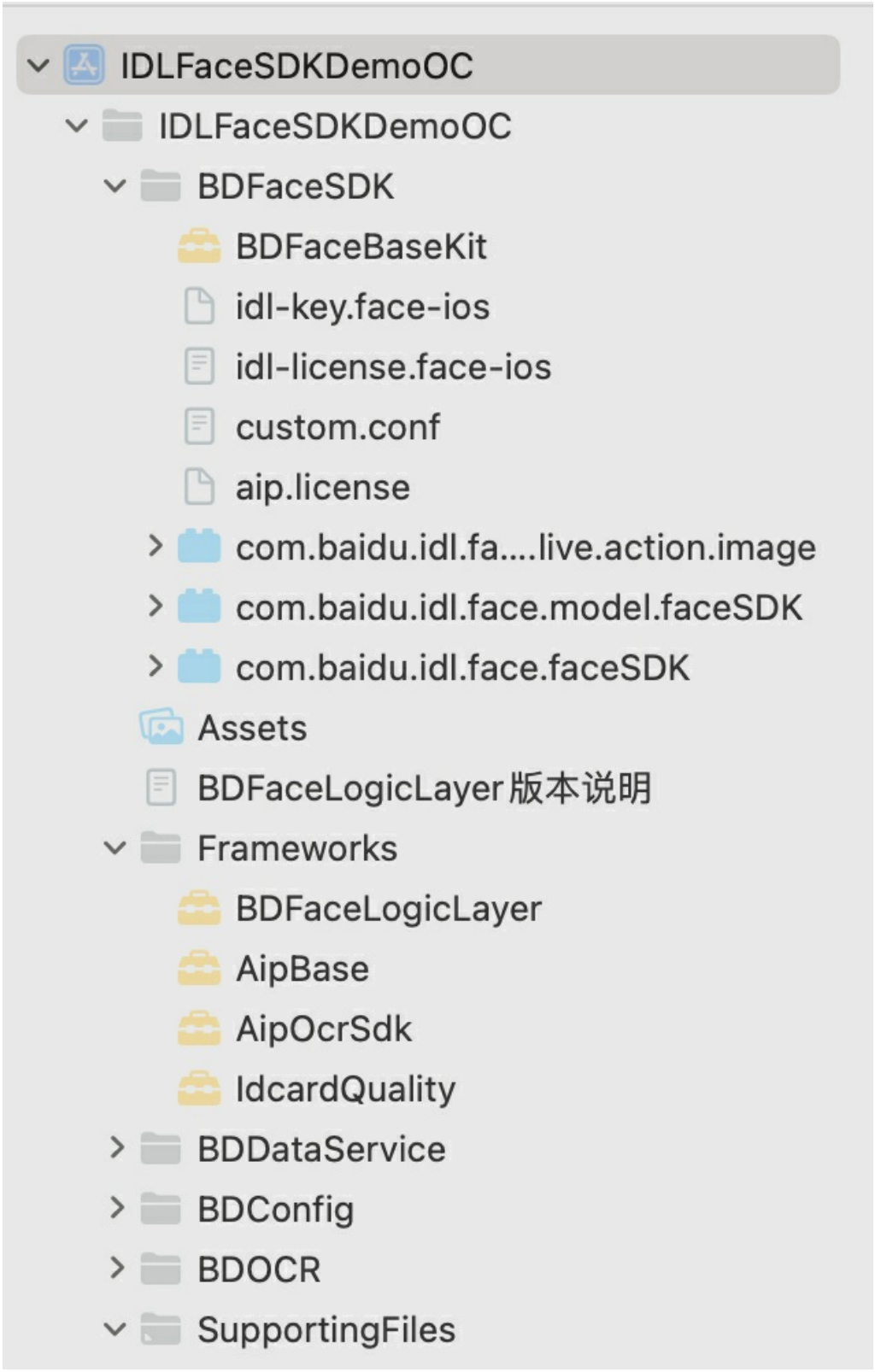
SDK	版本号	说明	类型
BDFaceLogicLayer.framework	1.0.0	名镜服务SDK，必须依赖	静态库
BDFaceBaseKit.framework	6.4.0	人脸采集SDK逻辑层，必须依赖	静态库
IDLFaceSDK.framework	1.0.0	人脸采集SDK算法层，非必须依赖	静态库
AipOcrSdk.framework	1.1.0	OCR识别SDK，非必须	动态库
IdcardQuality.framework	1.0.0	身份证质量控制SDK，非必须	动态库
AipBase.framework	1.0.0	基础工具类SDK，非必须	动态库

注：获取SDK可参考：获取示例工程

4. 运行项目工程

4.1 打开下载的iOS示例工程

如下图所示：



4.2 确认bundleId等信息是否正确

工程中的BundleID需要和方案配置中的包名一致，同时修改工程中的accessToken为方案绑定的应用生成的鉴权信息。

```
-ios | No Editor | m ViewContro...+MainSteps | h BDFaceLogicService | m ViewController | custom
IDLFaceSDKDemoOC 〉 IDLFaceSDKDemoOC 〉 SupportingFiles 〉 m ViewController+MainSteps 〉 C ViewController(Main_process)
30 @implementation ViewController (Main_process)
32 - (void)getAccessToken {
51         }
52     }];
53     [task resume];
54
55 #warning developer 用户需要替换上面的变量 BDTokenGetRequestUrl，具体Url是什么，可以参考注释。
       如果只是想调试，可以将token写死，如下面所示：设置self.accessToken=@"",如何获取token，
       参考：https://ai.baidu.com/ai-doc/REFERENCE/Ck3dwjhhu
56     ///  如下代码，只是调试，正式代码需要从url中及时获取，需要再每次进行人脸实名认证的时候，即时获取。
57     self.accessToken = @"[redacted]";
58 }
59
```

连接真机进行运行，之后可以看到如下界面：

4.3 点击开始身份认证

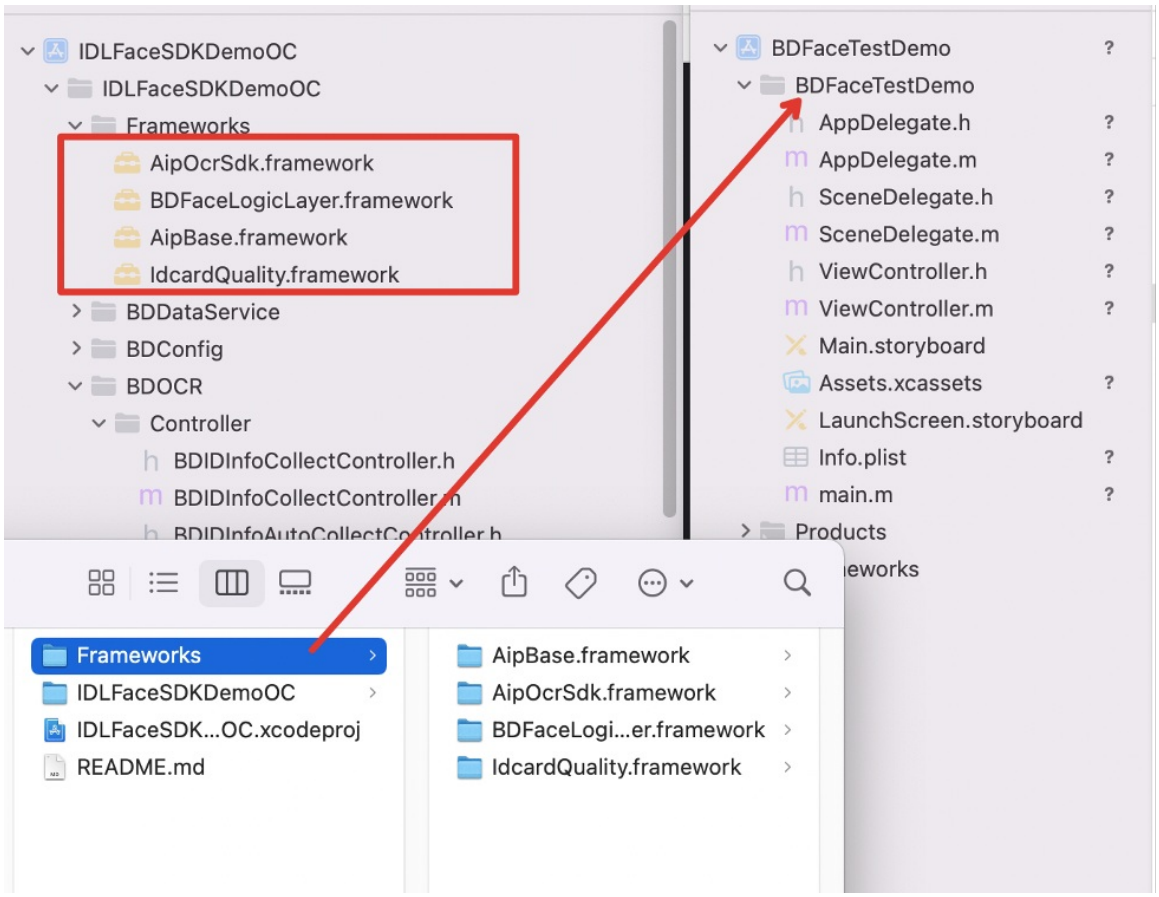
测试示例工程是否跑通，之后扫码身份证或输入身份证信息后，点击进行身份核验，验证成功可以看到如下界面：

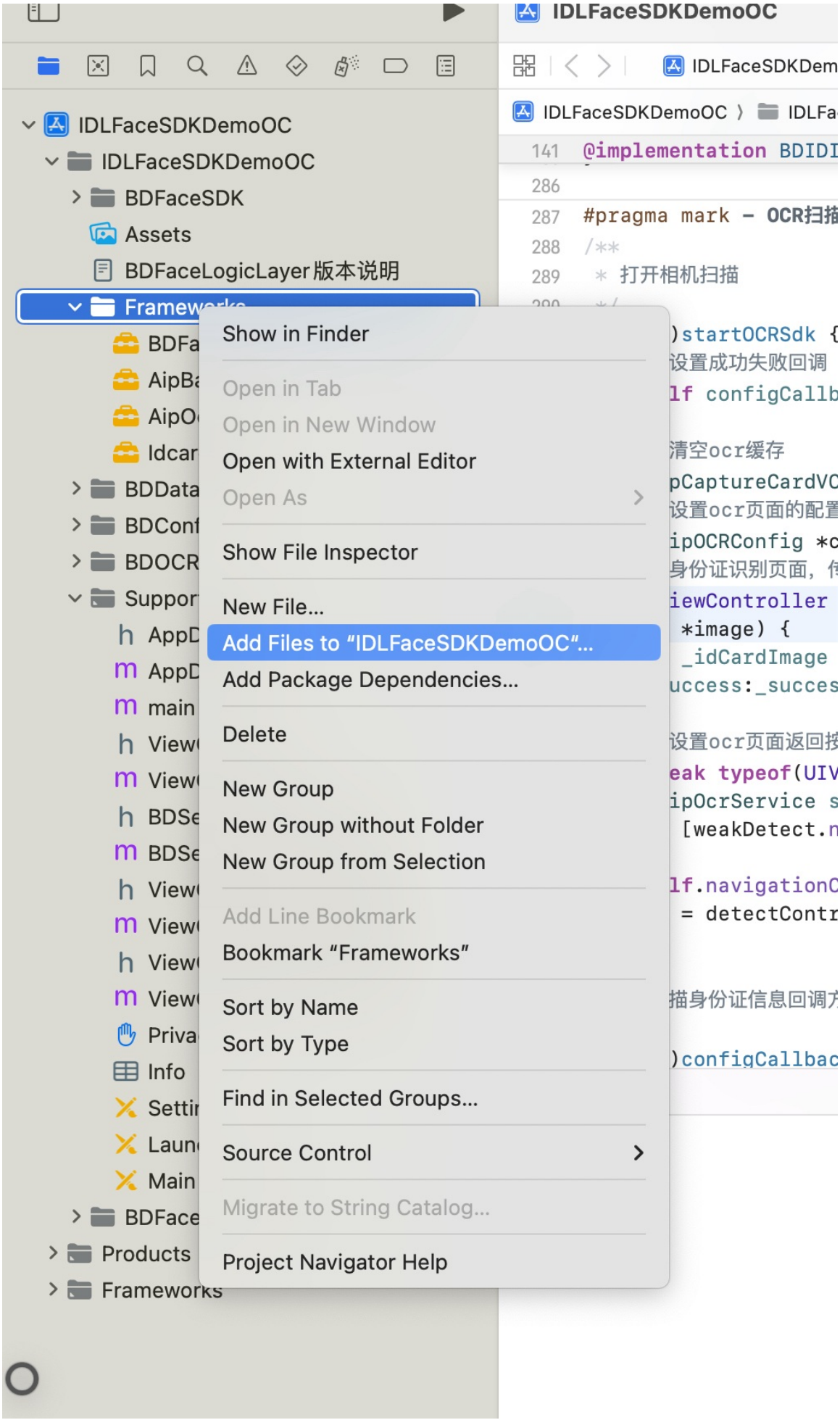
上图为示例工程运行成功，之后可以将示例工程代码集成到目标项目中。

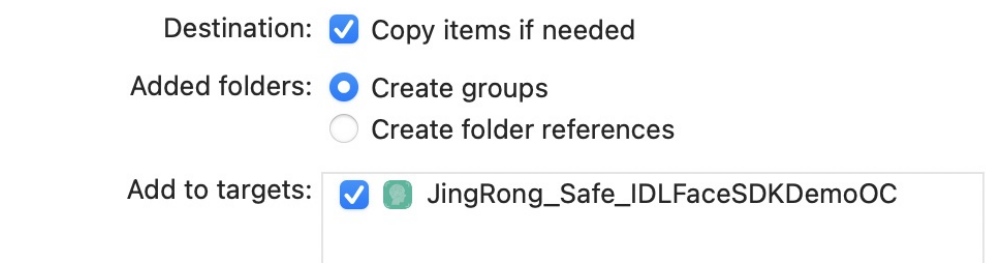
5. 集成步骤

5.1 将BDFaceLogicLayer.framework、AipBase.framework、 AipOcrSdk.framework和IdcardQuality.framework拖动到目标项目(在对应目录右键通过add files to xxx的方式添加)

如下所图示




IDLFaceSDKDemoOC
IDLFaceSDKDemoOC
IDLFaceSDKDemoOC
BDFaceSDK
Assets
BDFaceLogicLayer版本说明
Frameworks
Products
Frameworks
Show in Finder
Open in Tab
Open in New Window
Open with External Editor
Open As
Show File Inspector
New File...
Add Files to "IDLFaceSDKDemoOC"...
Add Package Dependencies...
Delete
New Group
New Group without Folder
New Group from Selection
Add Line Bookmark
Bookmark "Frameworks"
Sort by Name
Sort by Type
Find in Selected Groups...
Source Control
Migrate to String Catalog...
Project Navigator Help
141 @implementation BDIDI
286
287 #pragma mark - OCR扫描
288 /**
289 * 打开相机扫描

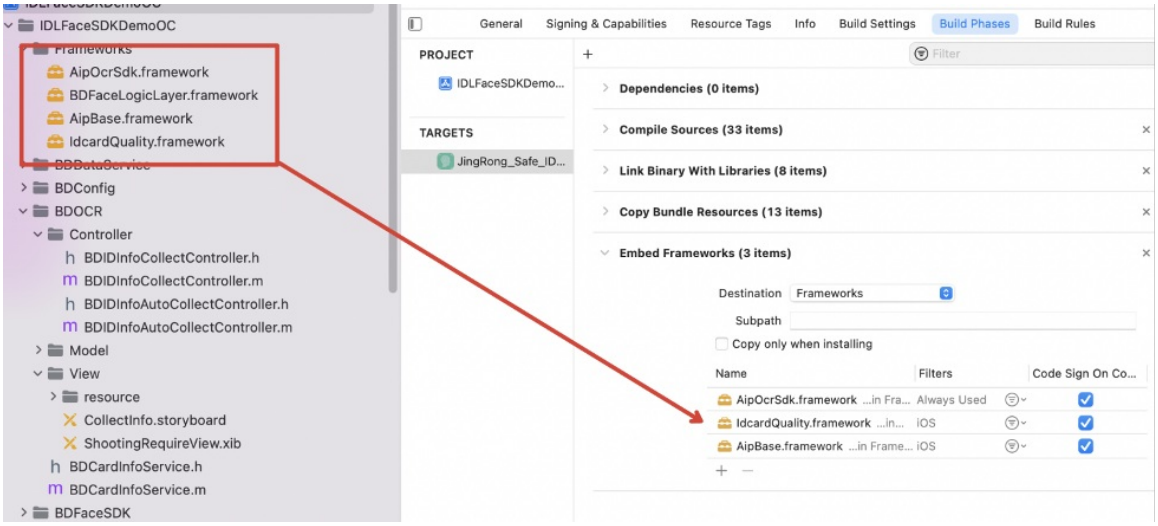


5.2 在Build Phases中点击下面的+号

选择NewCopyFilePhases,添加一个CopyFiles（可重新命名为Embed）

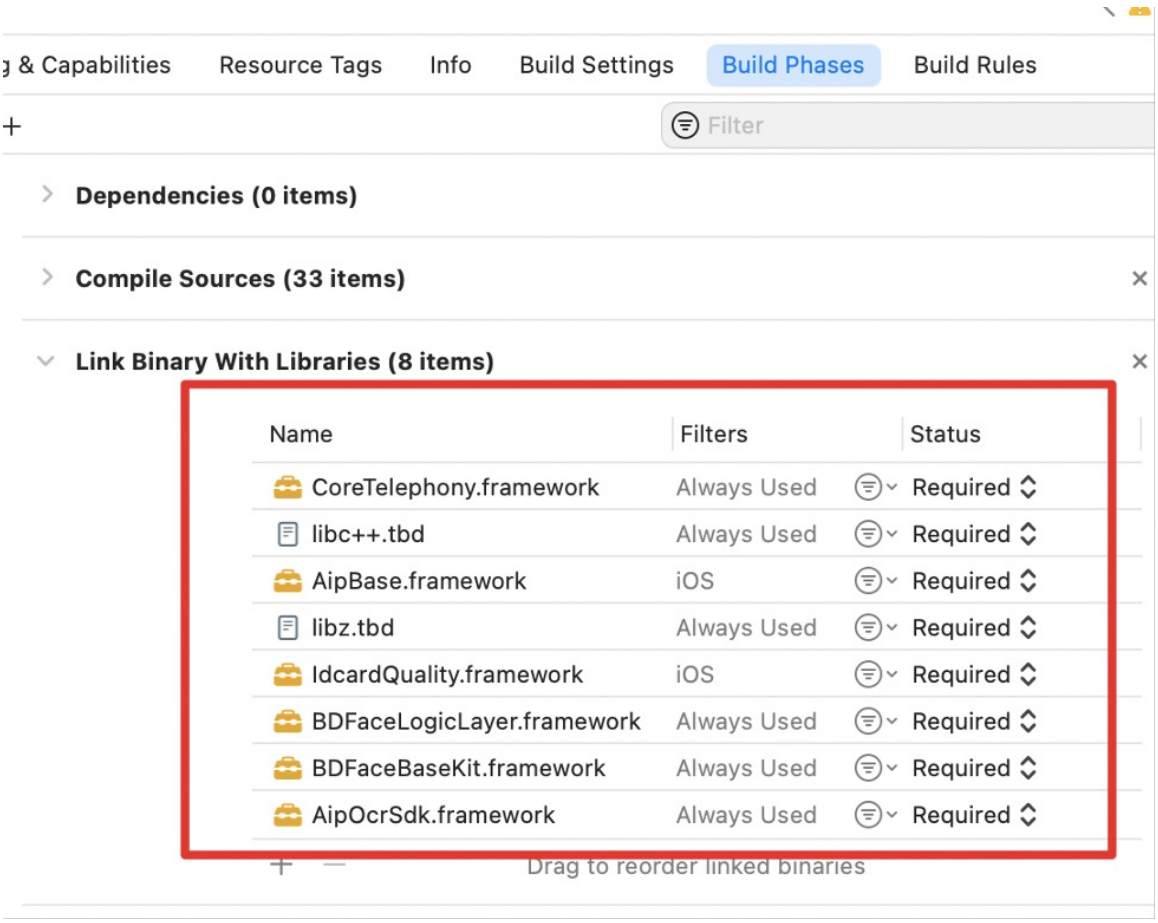
5.3 双击CopyFiles

改为Embed Frameworks，同时Destination改为framework，之后将AipBase.framework、AipOcrSdk.framework和IdcardQuality.framework 拖动到下面



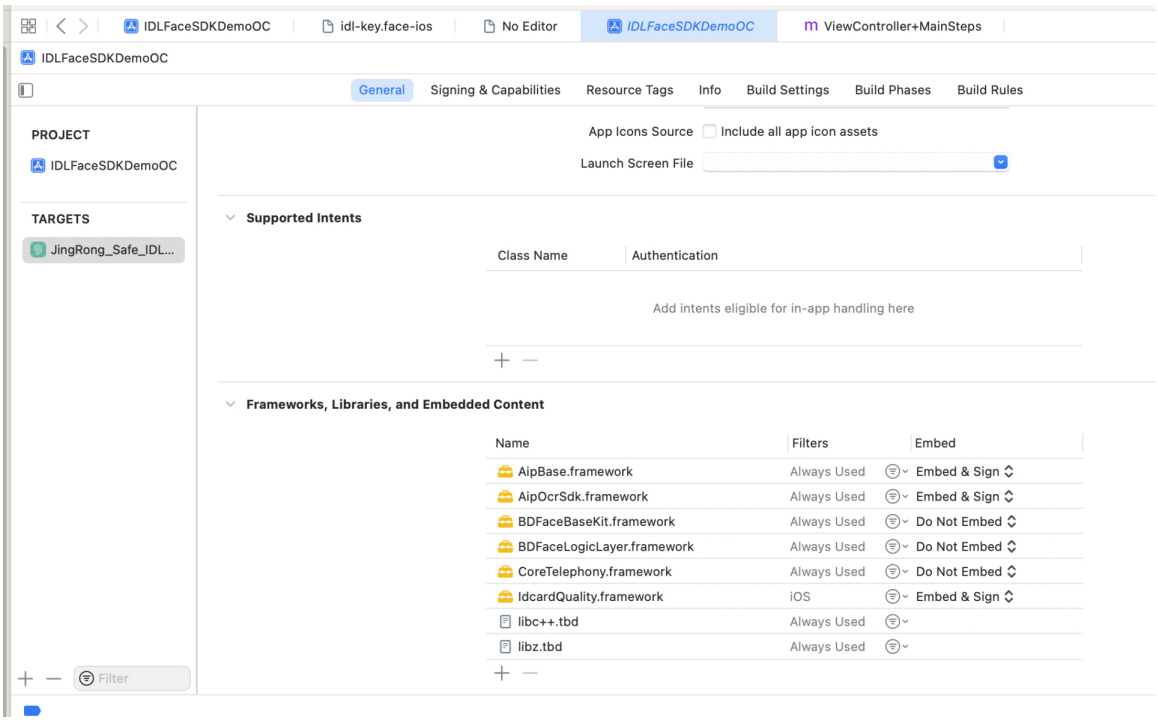
5.4 Link Binary With Libraries中添加libc++.tbd、libz.tbd 和 对应的framework

如下图所示:

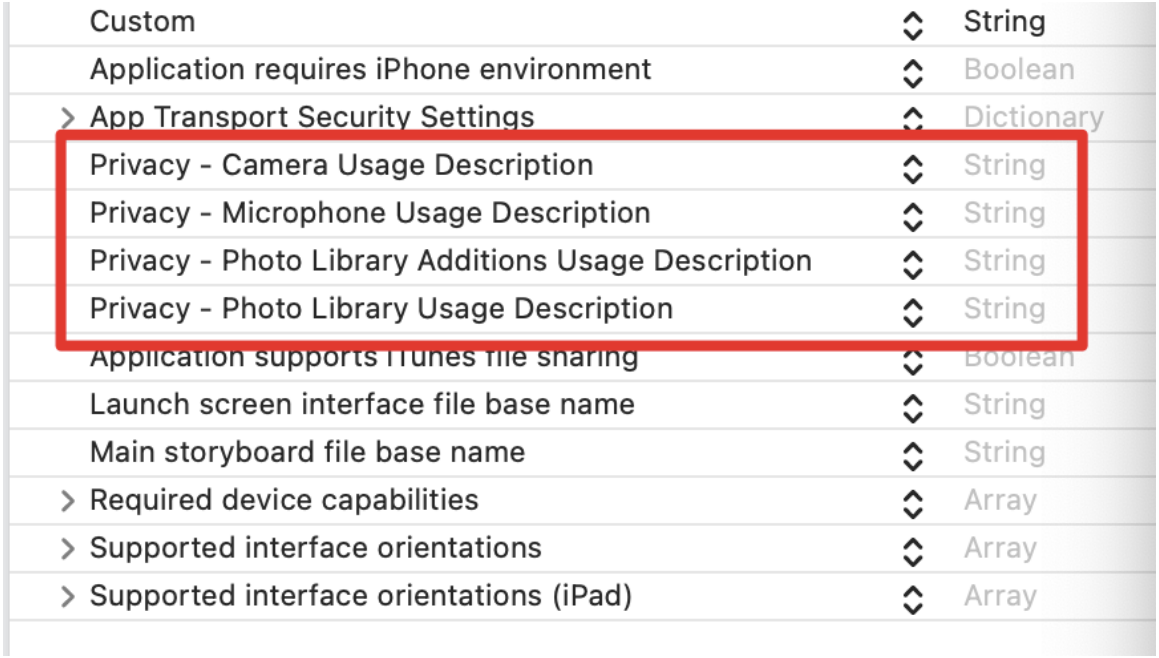


5.5 确认添加的库文件设置正确

如下图所示，点击general, 这三个库需要为Do Not Embed：BDFaceBaseKit.framework, BDFaceLogicLayer.framework，CoreTelephony.framework.



5.6 info.plist 文件中添加以下key（获取相机和照片权限）



最后之后可以将人脸核验示例工程代码代码结合自身工程项目，将相关代码集成到目标工程中。

6. 人脸初始化接口（SDK）

设置默认参数（动作活体参数，人脸配置参数，UI参数）

Step 1: 初始化人脸和OCR SDK

```
- (void)initFaceServiceAndInfoCollectService
```

Step 2: 初始化人脸SDK，进行参数配置。其中涉及活体方案相关的参数配置示例如下

```
- (void)initFaceSDK
/// 采集SDK初始化参数
[BDFaceBaseKitManager sharedInstance].paramsCustomConfigItem.isLivenessType = YES; //是否开启动作活体
[BDFaceBaseKitManager sharedInstance].paramsCustomConfigItem.isColorType = YES; //是否开启炫彩活体

//如果想实现更精细的活体方案配置
@property (nonatomic, assign) int     numOfLiveness;              // 活体动作池中有几个动作 （动作选取）
@property (nonatomic, assign) int     actionLiveSelectNum;          // 需要从动作活体动作池中抽取几个动作（动作数量）
@property (nonatomic, strong) NSMutableArray  *liveActionArray;       // 动作活体列表
```

API	描述
-(instancetype)initWithController:(UIViewController * _Nonnull)controller face:(void(^)(void))initFaceBlock ocr:(void(^)(void))initOcrBlock	传入当前controller，并初始化人脸和ocrSDK

入参说明

参数	类型	说明
controller	UIViewController *	当前controller
initFaceBlock	Block对象	人脸识别初始化Block
initOcrBlock	Block对象	OCR识别初始化Block

initFaceBlock中，使用以下函数进行人脸SDK初始化 -(void)initCollectWithLicenseID:(NSString *)licenseID andLocalLicenceName:(NSString *)licenseName andExtradata:(NSDictionary *)extradata callback:(FaceSDKInitResultBlock *)block;

回调状态码说明

参数	类型	说明
BDFaceInitRemindCode	枚举	初始化人脸的状态码,1000为成功，其他为失败，详情参考如下错误码说明

具体枚举值如下

枚举值	对应数字	说明	自查建议
BDFaceLICENSE_NOT_INIT_ERROR	1001	license未初始化	请按照集成文档说明完成SDK初始化
BDFaceLICENSE_DECRYPT_ERROR	1002	license数据解密失败	请检查License文件是否正确
BDFaceLICENSE_INFO_FORMAT_ERROR	1003	license数据格式错误	请检查license文件内容有被修改过
BDFaceLICENSE_KEY_CHECK_ERROR	1004	license-key(api-key)校验错误	请检查工程代码初始化参数中的licenseId，和申请license文件的licenseId是否匹配
BDFaceLICENSE_ALGORITHM_CHECK_ERROR	1005	算法ID校验错误	请提交工单或者线下联系百度产研人员
BDFaceLICENSE_MD5_CHECK_ERROR	1006	MD5校验错误	请检查工程所使用的签名文件，和申请license文件的签名信息是否匹配
BDFaceLICENSE_DEVICE_ID_CHECK_ERROR	1007	设备ID校验错误	采集SDK的授权模式不会出现这个错误码
BDFaceLICENSE_PACKAGE_NAME_CHECK_ERROR	1008	包名(应用名)校验错误	请检查工程代码中的applicationId（包名）和申请license文件的applicationId（包名）是否匹配
BDFaceLICENSE_EXPIRED_TIME_CHECK_ERROR	1009	授权过期时间有问题	请提交工单或者线下联系百度产研人员
BDFaceLICENSE_FUNCTION_CHECK_ERROR	1010	功能未授权	请查看授权文件中是否缺少必要的采集SDK功能声明（funclist参数），例如炫瞳活体
BDFaceLICENSE_TIME_EXPIRED	1011	授权已过期	请查看当前设备时间是否已不在授权文件有效期内
BDFaceLICENSE_LOCAL_FILE_ERROR	1012	本地授权文件读取失败	请检查授权文件名称以及路径
BDFaceLICENSE_REMOTE_DATA_ERROR	1013	远程授权文件拉取失败	本地鉴权失败之后，会远程拉取授权文件；若远程鉴权依然失败，可以关闭网络后重试
BDFaceLICENSE_LOCAL_TIME_ERROR	1014	本地时间校验错误	请检查当前设备时间是否早于实际时间

授权文件相关逻辑

> 人脸识别授权文件（idl-license.face-ios），OCR身份证识别授权文件（aip.license），从console平台下载完iOS项目，这些文件即包括在下载的示例项目中，不需要特殊处理，按上面第5.3步整体拖入目标项目中即可。

> 如果您下载集成方案的物料为控制台，此处可忽略；鉴权文件配置已自动生效。

配置授权信息： 导入license文件后，需配置FACE_LICENSE_NAME、FACE_LICENSE_ID等信息:

```
// 人脸license文件名
#define FACE_LICENSE_NAME    @"idl-license.face-ios"
//  (您申请的应用名称(appname)+「-face-ios」后缀，如申请的应用名称(appname)为test123，则此处填写test123-face-ios)
// 在后台 -> 产品服务 -> 人脸识别 -> 客户端SDK管理查看，如果没有的话就新建一个
#define FACE_LICENSE_ID         @"vis-var-license-5.0-face-ios"
```

具体参数使用请参照示例demo。

license文件使用注意事项：如出现鉴权失败无法调用人脸采集，请检查license id与license文件导入及配置是否正常。

（1）license id需要注意-face-ios后缀是否添加。

（2）如存在多种环境配置，请确保每个环境下的license id与license文件一一对应。tip：license文件存在运行缓存，切换测试环境请清空缓存，删除安装包后重新编译安装。

（3）确保license文件路径访问正常。

（4）如出现网络鉴权失败等提示，说明license文件与FACE_LICENSE_NAME未正确设置，请参考接入demo检查工程环境，确保参数无误。

7. 获取verify_token接口（服务端）

本接口为实名认证APP方案的verify_token获取接口，利用所获取的verify_token进行实名认证流程的有效期为**2小时**。

在线调试 您可以在 示例代码中心 中调试该接口，可进行签名验证、查看在线调用的请求内容和返回结果、示例代码的自动生成。

调用方式

请求URL数据格式

向API服务地址使用POST发送请求，必须在URL中带上参数`access_token`，可通过后台的API Key和Secret Key生成，具体方式请参考“Access Token获取”。

> **注意**：`access_token`的有效期为30天，**切记需要每30天进行定期更换，或者每次请求都拉取新token**；

POST中Body的参数，按照下方请求参数说明选择即可。

> **提示**：如果您为百度云老用户，正在使用其他非AI的服务，可以参考百度云AKSK鉴权方式发送请求，虽然**请求方式**和**鉴权方法**和本文所介绍的不同，但请求参数和返回结果一致。

请求说明

注意事项：

- **请求体格式化**：Content-Type为`application/json`，通过`json`格式化请求体。

请求示例

HTTP方法：`POST`

请求URL： `https://aip.baidubce.com/rpc/2.0/brain/solution/faceprint/verifyToken/generate`

URL参数：

参数	值
access_token	通过API Key和Secret Key获取的access_token,参考“Access Token获取”

Header：

参数	值
Content-Type	application/json

Body中放置请求参数，参数详情如下：

请求参数：

参数	必选	类型	说明
match_source	是	int	方案类型（比对源）；0：实名认证（权威库）；1：人脸比对（上传照片）；2：仅活体检测。
plan_id	是	string	方案的id信息，请在人脸实名认证控制台查看创建的APP方案ID信息

- 请求示例

```
{
"match_source": 0, //以权威库核验场景为例，需要传入0
"plan_id": 16144,
}
```

返回参数

- 返回结果

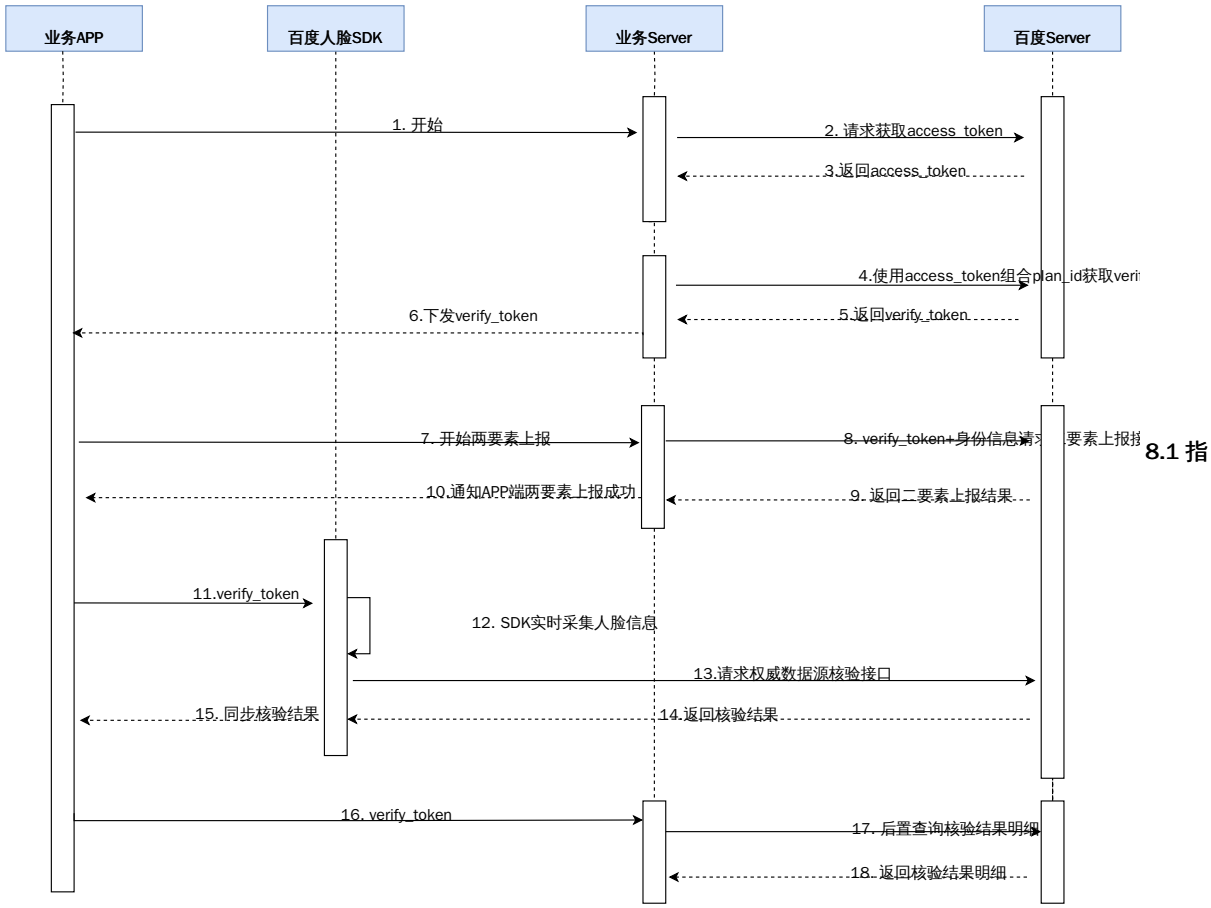
字段	必选	类型	说明
success	是	boolean	返回请求是否成功信息。 若请求成功返回ture; 请求失败则返回false
result	是	object	请求结果
+verify_token	是	string	请求获取的verify_token

- 返回示例

```
{
  "success": true,
  "result": {
     "verify_token": "Yz9rWITm4vak16PBAh5x8oG7"
  },
  "log_id": "1814798895"
}
```

8. 人脸实名认证实现方式

实现方式请参考如下时许图



定用户信息上报接口（服务端） 本接口用于，控制台APP方案中选择身份信息录入方式为**业务调用时传入**，需要先调用此接口输入指定用户的姓名+身份证号信息，再继续请求 **人脸权威库核验接口（SDK）**

在线调试 您可以在 示例代码中心 中调试该接口，可进行签名验证、查看在线调用的请求内容和返回结果、示例代码的自动生成。 **调用方式**

请求URL数据格式

向API服务地址使用POST发送请求，必须在URL中带上参数`access_token`，可通过后台的API Key和Secret Key生成，具体方式请参考“Access Token获取”。

> **注意**：`access_token`的有效期为30天，**切记需要每30天进行定期更换，或者每次请求都拉取新token**；

POST中Body的参数，按照下方请求参数说明选择即可。

> **提示**：如果您为百度云老用户，正在使用其他非AI的服务，可以参考百度云AKSK鉴权方式发送请求，虽然**请求方式**和**鉴权方法**和本文所介绍的不同，但请求参数和返回结果一致。

请求说明

注意事项：

- **请求体格式化**：Content-Type为`application/json`，通过`json`格式化请求体。

请求示例

HTTP方法：`POST`

请求URL：https://aip.baidubce.com/rpc/2.0/brain/solution/faceprint/idcard/submit

URL参数：

参数	值
access_token	通过API Key和Secret Key获取的access_token,参考“Access Token获取”

Header：

参数	值
Content-Type	application/json

Body中放置请求参数，参数详情如下：

参数	必选	类型	值
verify_token	是	string	通过access_token获取的verify_token
id_name	是	string	指定输入用户的姓名信息
id_no	是	string	指定输入用户的身份证件号信息
certificate_type	否	int	证件类型： 0 大陆居民二代身份证 4 港澳台居民居住证

- 请求示例：

```
{
  "verify_token": "2sF3nE5mXOHkx2aQwWG4n5WI",
  "id_name": "张三",
  "id_no": "500***********3390",
  "certificate_type": 0          // 证件类型：0:大陆居民二代身份证,4:港澳台居民居住证
}
```

返回参数

- 返回结果

字段	必选	类型	说明
success	是	boolean	返回请求是否成功信息。 若请求成功返回ture; 请求失败则返回false
result	是	int	请求结果，返回固定结果1，可忽略
log_id	是	string	本次查询接口请求的logid

- 返回示例

```
{
  "success": true,
  "result": 1,
  "log_id": "1244068892"
}
```

8.2 人脸权威库核验接口（SDK） 基于姓名、身份证号、当前SDK获取的人脸图片，与权威库进行对比，并得出比对分数，并基于此进行业务判断是否为同一人。

返回值	API	描述
void	startCheckLiveCommonAction:(NSDictionary *)dic callBack:(void(^)(int resultCode, NSDictionary resultDic))callBack;	SDK核验接口

入参dic值列表如下：

参数	类型	说明	必选
BDFaceLogicServiceTokenKey	String	key值为BDFaceLogicServiceTokenKey，value需要传入verify_token对应的具体值。需要APP服务端通过AK、SK获取access_token，参考 https://ai.baidu.com/docs#/Auth/top,然后通过access_token获取verify_token **【此处需要注意】verify_token有效期两个小时，建议每次调用接口重新获取verify_token**	是
match_source	int	方案类型（比对源）；0：实名认证（权威库）；1：人脸比对（上传照片）；2：仅活体检测；**这里默认传入0**	是

onCallback回调说明

参数	类型	含义	值
resultCode	int	错误码	0为成功，其他为失败，详情参考resultCode错误码说明
resultDic	NSDictionary	回调结果	详情见下表

resultDic key值列表说明：

Key值	类型	含义
resultMsg	String	详情见resultCode错误码说明
data	String	服务端返回结果json,只有resultCode为0时会返回 此字段

- resultCode和resultMsg说明

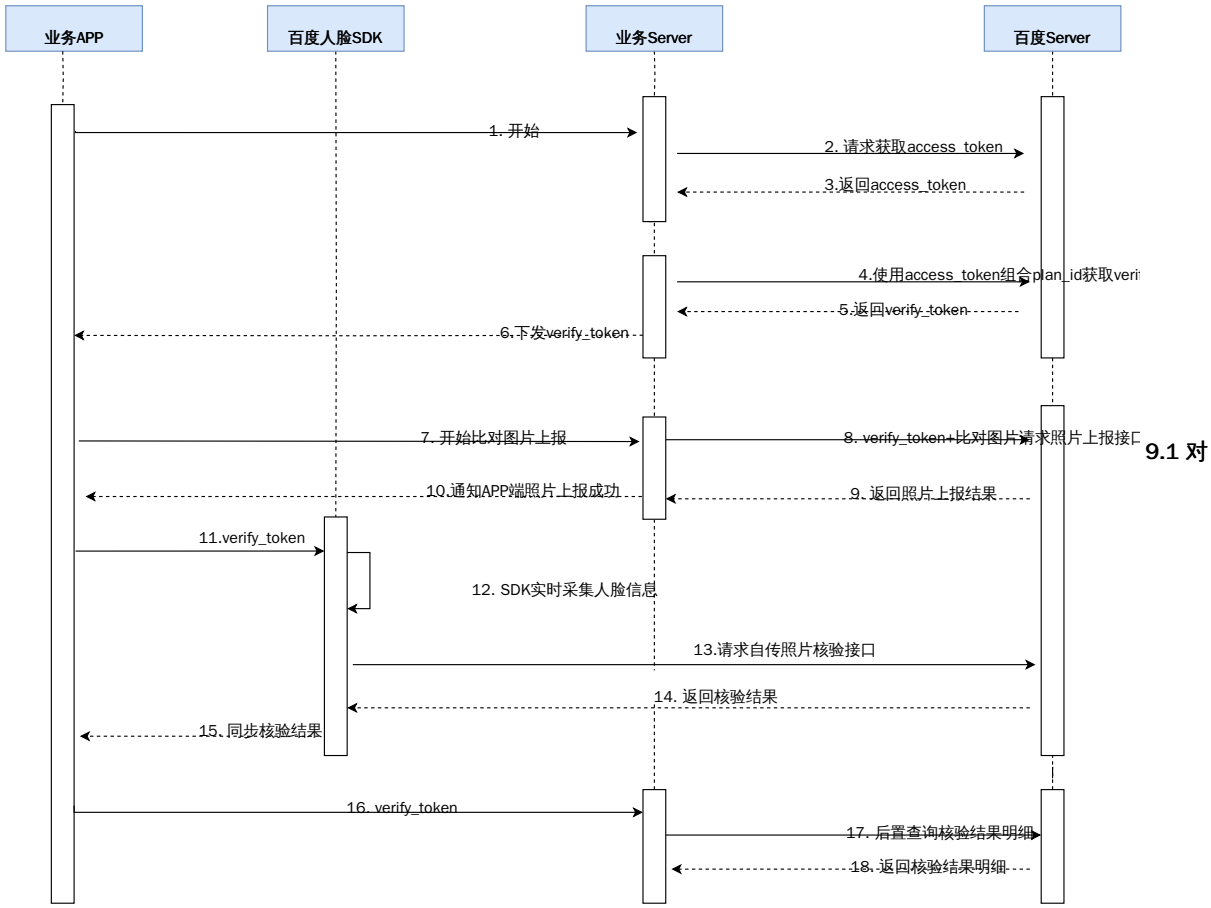
resultCode	resultMsg	自查方案
-102	采集流程取消	用户自行取消采集流程
-103	SDK未初始化	需要检查初始化方法有没有调用，建议在application里面调用
-303	视频录制错误	是否授权视频/音频
-310	摄像头调起失败	请检查摄像头权限
-401	超时	可通过设置延长超时时间限制
-402	炫瞳异常	炫瞳打光颜色与模型输出不匹配
-1001	网络异常	请检查网络通讯

- data返回示例

```
{
 "error_code": "0", //代表通过了百度内部的核验规则，建议使用服务端查询接口获得真实结果做业务判断
  "log_id": "1790651229035123218"
}
```

9. 人脸对比（自传照片）实现方式

实现方式请参考如下时许图



9.1 对

比图片上传接口（服务端） 本接口适用于人脸比对（自传照片）业务场景，为了保证用户信息安全，您需要提前通过服务端请求本接口来上报比对人脸信息。

在线调试 您可以在 示例代码中心 中调试该接口，可进行签名验证、查看在线调用的请求内容和返回结果、示例代码的自动生成。 **调用方式**

请求URL数据格式

向API服务地址使用POST发送请求，必须在URL中带上参数`access_token`，可通过后台的API Key和Secret Key生成，具体方式请参考“Access Token获取”。

> **注意**：`access_token`的有效期为30天，**切记需要每30天进行定期更换，或者每次请求都拉取新token**；

POST中Body的参数，按照下方请求参数说明选择即可。

> **提示**：如果您为百度云老用户，正在使用其他非AI的服务，可以参考百度云AKSK鉴权方式发送请求，虽然**请求方式**和**鉴权方法**和本文所介绍的不同，但请求参数和返回结果一致。

请求说明

注意事项：

- **请求体格式化**：Content-Type为`application/json`，通过`json`格式化请求体。

请求示例

HTTP方法：`POST`

请求URL： https://aip.baidubce.com/rpc/2.0/brain/solution/faceprint/uploadMatchImage

URL参数：

参数	值
access_token	通过API Key和Secret Key获取的access_token,参考“Access Token获取”

Header：

参数	值
Content-Type	application/json

Body中放置请求参数，参数详情如下：

参数	必选	类型	值
verify_token	是	string	通过access_token获取的verify_token
image	是	string	图片base64字符串，编码后的图片大小不超过10M，图片分辨率小于1920*1080
quality_control	否	string	“NONE”、“LOW”、“NORMAL”、“HIGH”； 质量控制参数，未主动传入时默认为“NONE”
liveness_control	否	string	“NONE”、“LOW”、“NORMAL”、“HIGH”； 活体控制参数，未主动传入时默认为“NONE”

- 请求示例：

```
{
 "verify_token": "2sF3nE5mXOHkx2aQwWG4n5WI",
 "image": "/9j/4AAQSkZJRgABAQAAAQABAAD/",
 "quality_control" : "LOW",
 "liveness_control" : "LOW",
}
```

返回参数

- 返回结果

字段	必选	类型	说明
success	是	boolean	返回请求是否成功信息。 若请求成功返回ture; 请求失败则返回false
log_id	是	string	logid
result	是	object	返回结果

- 返回示例

```
{
 "success": true,
 "result": null,
 "log_id": "1329130892"
}
```

9.2 人脸对比（SDK） 包含活体检测、质量检测、人脸1:1识别功能，可通过传入参数进行控制。

返回值	API	描述
void	startCheckLiveCommonAction:(NSDictionary *)dic callBack:(void(^)(int resultCode, NSDictionary* resultDic))callBack;	SDK核验接口

入参dic值列表如下：

参数	类型	说明	必选
BDFaceLogicServiceTokenKey	String	key值为BDFaceLogicServiceTokenKey，value需要传入verify_token对应的具体值。需要APP服务端通过AK、SK获取access_token，参考 https://ai.baidu.com/docs#/Auth/top,然后通过access_token获取verify_token **【此处需要注意】verify_token有效期两个小时，建议每次调用接口重新获取verify_token**	是
register_image	data	本地上传用来比对的图片源	是
match_source	int	方案类型（比对源）；0：实名认证（权威库）；1：人脸比对（上传照片）；2：仅活体检测；**这里默认传入1**	是

onCallback回调说明

参数	类型	含义	值
resultCode	int	错误码	0为成功，其他为失败，详情参考实名认证resultCode错误码说明
resultDic	Dictionary	回调结果	详情见下表

- resultDic key值列表说明：

Key值	类型	含义
resultMsg	String	详情见resultCode错误码说明
data	String	服务端返回结果json,只有resultCode为0时会返回 此字段

- resultCode和resultMsg说明

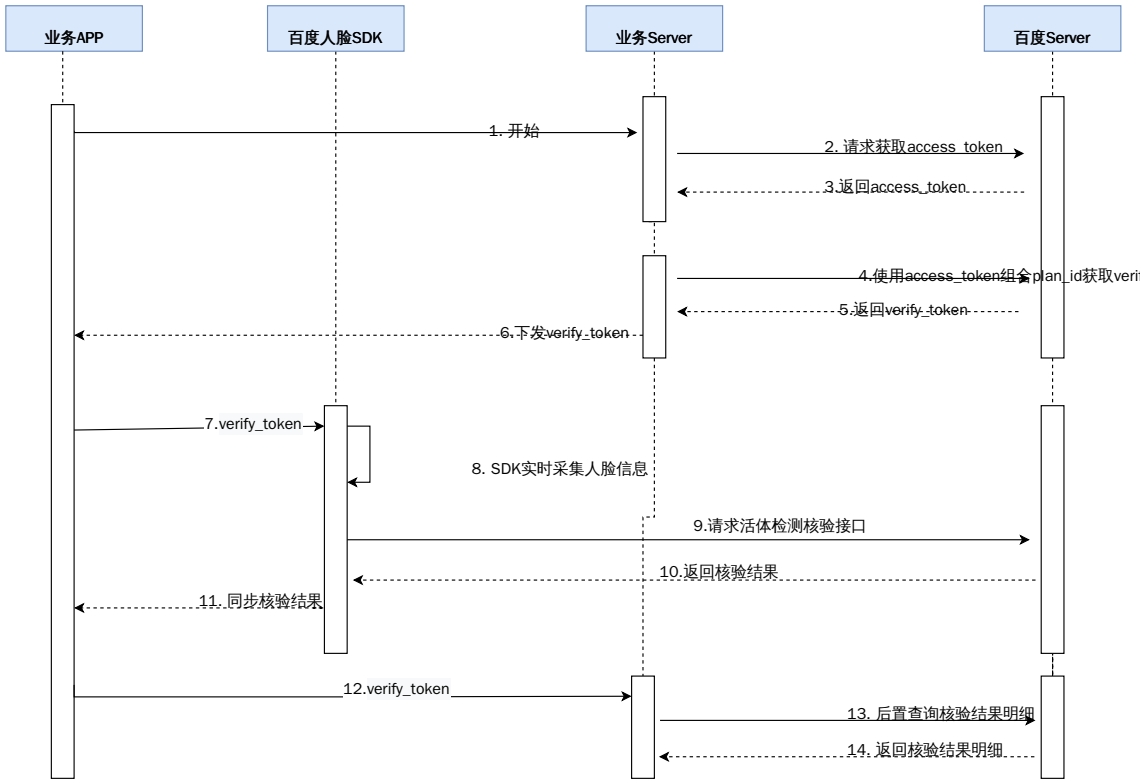
resultCode	resultMsg	自查方案
-102	采集流程取消	用户自行取消采集流程
-103	SDK未初始化	需要检查初始化方法有没有调用，建议在application里面调用
-303	视频录制错误	是否授权视频/音频
-310	摄像头调起失败	请检查摄像头权限
-401	超时	可通过设置延长超时时间限制
-402	炫瞳异常	炫瞳打光颜色与模型输出不匹配
-1001	网络异常	请检查网络通讯

- data返回示例

```
{
 "error_code": "0", //代表通过了百度内部的核验规则，建议使用服务端查询接口获得真实结果做业务判断
  "log_id": "1790651229035123218"
}
```

10. 人脸活体检测实现方式

实现方式请参考如下时许图



10.1 活体检测（SDK）

包含本地活体加云端活体，本地活体分静默活体、炫瞳活体、动作活体三种，云端活体可以判断图片中的人脸是否为二次翻拍以及是否为合成图攻击。实现二次验证采集图片是否存在假体攻击破绽的情况。
如您的业务场景核心为人脸实名认证（权威源），请直接参考 **8.2 人脸权威库核验接口（SDK）** 。

返回值	API	描述
void	startCheckLiveCommonAction:(NSDictionary *)dic callBack:(void(^)(int resultCode, NSDictionary* resultDic))callBack;	SDK核验接口

入参dic值列表如下：

参数	类型	说明	必选
BDFaceLogicServiceTokenKey	String	key值为BDFaceLogicServiceTokenKey，value需要传入verify_token对应的具体值。需要APP服务端通过AK、SK获取access_token，参考 https://ai.baidu.com/docs#/Auth/top,然后通过access_token获取verify_token **【此处需要注意】verify_token有效期两个小时，建议每次调用接口重新获取verify_token**	是
match_source	int	方案类型（比对源）；0：实名认证（权威库）；1：人脸比对（上传照片）；2：仅活体检测；**这里默认传入2.**	是

onCallback回调说明

参数	类型	含义	值
resultCode	int	错误码	0为成功，其他为失败，详情参考实名认证resultCode错误码说明
resultDic	NSDictionary	回调结果	详情见下表

- resultDic key值列表说明：

Key值	类型	含义
resultMsg	String	详情见resultCode错误码说明
data	String	服务端返回结果json,只有resultCode为0时会返回 此字段

- resultCode和resultMsg说明

resultCode	resultMsg	自查方案
-102	采集流程取消	用户自行取消采集流程
-103	SDK未初始化	需要检查初始化方法有没有调用，建议在application里面调用
-303	视频录制错误	是否授权视频/音频
-310	摄像头调起失败	请检查摄像头权限
-401	超时	可通过设置延长超时时间限制
-402	炫瞳异常	炫瞳打光颜色与模型输出不匹配
-1001	网络异常	请检查网络通讯

- data返回示例

```
{
 "error_code": "0", //代表通过了百度内部的核验规则，建议使用服务端查询接口获得真实结果做业务判断
  "log_id": "1790651229035123218"
}
```

11. 验证后查询接口

11.1 获取认证人脸接口（服务端）

本接口返回进行人脸实名认证过程中进行认证的最终采集的人脸信息。根据Verify_token返回的结果信息会在云端保留**3天**，您可根据需要在此期间进行调取查询。

在线调试 您可以在 示例代码中心 中调试该接口，可进行签名验证、查看在线调用的请求内容和返回结果、示例代码的自动生成。

调用方式

请求URL数据格式

向API服务地址使用POST发送请求，必须在URL中带上参数`access_token`，可通过后台的API Key和Secret Key生成，具体方式请参考“Access Token获取”。

> **注意**：`access_token`的有效期为30天，**切记需要每30天进行定期更换，或者每次请求都拉取新token**；

POST中Body的参数，按照下方请求参数说明选择即可。

> **提示**：如果您为百度云老用户，正在使用其他非AI的服务，可以参考百度云AKSK鉴权方式发送请求，虽然**请求方式**和**鉴权方法**和本文所介绍的不同，但请求参数和返回结果一致。

请求说明

注意事项：

- **请求体格式化**：Content-Type为`application/json`，通过`json`格式化请求体。

请求示例

HTTP方法：`POST`

请求URL： `https://aip.baidubce.com/rpc/2.0/brain/solution/faceprint/result/simple`

URL参数：

参数	值
access_token	通过API Key和Secret Key获取的access_token,参考“Access Token获取”

Header：

参数	值
Content-Type	application/json

Body中放置请求参数，参数详情如下：

参数	值
verify_token	通过access_token获取的verify_token

- 请求示例：

```
{
 "verify_token" : "clupeyP51sn28XzxGVTfYqoN"
}
```

返回参数

- 返回结果

字段	必选	类型	说明
success	是	boolean	返回请求是否成功信息。 若请求成功返回ture; 请求失败则返回false
result	是	object	请求结果
+image	是	string	返回采集的用户人脸信息

- 返回示例

```
{
  "success": true,
  "result": {
     "image":"https://brain.baidu.com/solution/faceprint/image/query?verify_token=xxxxxx"
  },
  "log_id": "1054986003"
}
```

11.2 查询认证结果接口

本接口为请求返回的认证结果信息查询，包含用户二次确认的身份证信息，活体检测信息、及用户对权威库图片进行比对的分数信息。（仅在认证成功时返回上述信息，认证失败返回错误码）根据Verify_token返回的结果信息会在云端保留**3天**，您可根据需要在此期间进行调取查询。

在线调试 您可以在 示例代码中心 中调试该接口，可进行签名验证、查看在线调用的请求内容和返回结果、示例代码的自动生成。 **调用方式**

请求URL数据格式

向API服务地址使用POST发送请求，必须在URL中带上参数access_token，可通过后台的API Key和Secret Key生成，具体方式请参考“Access Token获取”。

> **注意**：access_token的有效期为30天，**切记需要每30天进行定期更换，或者每次请求都拉取新token**；

POST中Body的参数，按照下方请求参数说明选择即可。

> **提示**：如果您为百度云老用户，正在使用其他非AI的服务，可以参考百度云AKSK鉴权方式发送请求，虽然**请求方式**和**鉴权方法**和本文所介绍的不同，但请求参数和返回结果一致。

请求说明

注意事项：

- **请求体格式化**：Content-Type为application/json，通过json格式化请求体。

请求示例

HTTP方法：POST

请求URL： https://aip.baidubce.com/rpc/2.0/brain/solution/faceprint/result/detail

URL参数：

参数	值
access_token	通过API Key和Secret Key获取的access_token,参考“Access Token获取”

Header：

参数	值
Content-Type	application/json

Body中放置请求参数，参数详情如下：

参数	值
verify_token	通过access_token获取的verify_token

请求示例：

```
 {
 "verify_token"："clupeyP51sn28XzxGVTfYqoN"
}
```

返回参数

- 返回结果

字段	必选	类型	说明
success	是	boolean	返回本次身份核验是否成功的结果。 成功返回ture; 失败则返回false
result	是	object	请求结果
+verify_result	是	object	认证返还信息
++verify_log_id	是	string	核验结果对应的日志ID
++liveness_score	是	float	**活体检测分数：** 活体验证通过时返回活体分数，不通过则返回0。
++score	是	float	人脸实名认证比对得分**（仅活体检测时返回为0）**
++risk_level	否	string	安全风控等级 方案配置启用安全风控，将返回该字段 值为1或2时表示触发了安全风险，值为3或4时无风险
++risk_tag	否	string	安全风控标签 方案配置启用安全风控，将返回该字段 当risk_level值为1或2时，返回具体风险类型，risk_level值为3或4时，为空
++spoofing	是	float	合成图分数 若未进行合成图检测，则返回0 若进行活体检测，则返回合成图检测分值
+idcard_confirm	是	object	用户二次确认的身份证信息
++name	是	string	姓名
++idcard_number	是	string	身份证号
code	否	int	错误码 **（仅认证失败时展示）**
message	否	string	错误信息 **（仅认证失败时展示）**
log_id	是	string	本次查询接口请求的日志ID

- 返回示例


```
{
 "success": true,
 "result": {
    "verify_result": {
        "verify_log_id": "1868893431661688963",
        "score": 94.57,
        "risk_level": "3",
        "risk_tag": "[]",
        "liveness_score": 0.99983007,
        "spoofing": 0.0
    },
    "idcard_confirm": {
        "idcard_number": "430419********4532",
        "name": "陈**"
    }
 },
 "log_id": "1868914111329362938"
}
```


- 返回失败示例

```
{
"success": false,
"result": {
    "verify_result": {
        "verify_log_id": "1866379694929155276",
        "score": 0.0052550267,
        "risk_level": "3",
        "risk_tag": "[]",
        "liveness_score": 0.0052550267,
        "spoofing": 0.0
    },
    "idcard_confirm": {
        "idcard_number": null,
        "name": null
    }
},
"code": 223120,
"message": "活体检测未通过",
"log_id": "1866379737400667552"
}
```


11.3 核验及计费信息获取（服务端）

根据Verify_token返回的信息会在云端保留**3天**，您可根据需要在此期间进行调取查询。

调用方式

请求URL数据格式

向API服务地址使用POST发送请求，必须在URL中带上参数`access_token`，可通过后台的API Key和Secret Key生成，具体方式请参考“Access Token获取”。

> **注意**：`access_token`的有效期为30天，**切记需要每30天进行定期更换，或者每次请求都拉取新token**；

POST中Body的参数，按照下方请求参数说明选择即可。

> **提示**：如果您为百度云老用户，正在使用其他非AI的服务，可以参考百度云AKSK鉴权方式发送请求，虽然**请求方式**和**鉴权方法**和本文所介绍的不同，但请求参数和返回结果一致。

在线调试 您可以在 示例代码中心 中调试该接口，可进行签名验证、查看在线调用的请求内容和返回结果、示例代码的自动生成。

请求说明

注意事项：

- **请求体格式化**：Content-Type为`application/json`，通过`json`格式化请求体。

请求示例

HTTP方法：`POST`

请求URL：`https://aip.baidubce.com/rpc/2.0/brain/solution/faceprint/result/getall`

URL参数：

参数	值
access_token	通过API Key和Secret Key获取的access_token,参考“Access Token获取”

Header：

参数	值
Content-Type	application/json

Body中放置请求参数，参数详情如下：

参数	值
verify_token	通过access_token获取的verify_token

请求示例：

```
{
  "verify_token" : "clupeyP51sn28XzxGVTfYqoN"
}
```

返回参数

- 返回结果

字段	必选	类型	说明
success	是	boolean	返回请求是否成功信息。 若请求成功返回ture; 请求失败则返回false
log_id	是	string	本次查询接口请求的logid
result	是	object	结果信息
+verify_count	是	int	当前verify_token对应的核验次数，核验次数为几，后面就有几个对象数组 如方案配置为“认证未通过时URL失效”，则该verify_token核验次数为1； 如方案配置为“认证未通过时URL不失效”，则该verify_token对应的核验次数可能大于1（用户核验失败后多次重试）
+ocr_charge_count	是	int	身份证识别OCR接口的计费次数
+match_charge_count	是	int	人脸对比接口的计费次数
+verify_charge_count	是	int	人脸实名认证接口的计费次数
+verify_result	是	object[]	核验结果信息，verify_count的值为几，就返回几个该对象
++order	是	int	代表本次核验结果在verify_token所有核验次数中的顺序，按时间先后排序，第一次核验返回值为 1，第二次核验返回值为 2，以此类推
++is_verify_passed	是	boolean	本次核验是否通过 核验成功返回true 核验失败返回false
			本次核验的错误码

++code	是	string	核验成功时返回 0 核验失败返回非0错误码
++message	是	string	本次核验的错误信息 核验成功时返回“核验成功” 核验失败时返回具体错误原因，如“身份证姓名不匹配”
++verify_time	是	string	本次核验完成的时间点，包含年月日、时分秒，如“2023-6-20 18:00:00”
++is_charged	是	boolean	本次核验是否计费 计费返回true 不计费返回false
++charge_type	是	string	计费类型：**verify**(人脸实名认证)、**match**(人脸对比)、**live**(在线图片活体)
++charge_time	是	string	本次计费的时间点，如不计费则该字段为空，包含年月日、时分秒，如“2023-6-20 18:00:00”
++is_ocr_charge	否	boolean	身份证识别OCR是否计费 计费返回true 不计费返回false
++ocr_charge_time	否	string	ocr 收费时间点，如不收费则该字段为空，包含年月日、时分秒，如“2023-6-20 18:00:00”
++ocr_count	否	int	ocr次数
++verify_detail	是	object	核验的详细信息
+++face_image	是	string	本次核验流程中采集的人脸最佳质量图下载url
+++verify_log_id	是	string	本次核验流程中请求的人脸实名认证（方案配置权威库比对）、人脸对比（方案配置自建人脸库比对）、在线图片活体（仅活体检测）后端接口logid，支持在记录查询平台中通过logid查询到3天内的记录。 少数核验失败情况下，实际并未发生上述俩接口的请求，则该字段为空，如：用户拒绝摄像头授权且不允许降级
+++score	是	float	人脸相似度得分，大于等于方案设置阈值为同一人；当身份证姓名不匹配或活体不通过时，该项为空
+++threshold	是	float	本次核验时，所使用的方案配置相似度阈值
+++liveness_score	是	float	活体检测得分，注：预留功能字段，当前返回为0
+++spoofing_score	否	float	方案配置使用合成图功能，将返回合成图得分，注：预留功能字段，当前返回为0
+++risk_level	否	string	安全风控等级 方案配置启用安全风控，将返回该字段 值为1或2时表示触发了安全风险，值为3或4时无风险
+++risk_tag	否	string	安全风控标签 方案配置启用安全风控，将返回该字段 当risk_level值为1或2时，返回具体风险类型，risk_level值为3或4时，为空
+++is_demote	否	boolean	H5方案相关，可忽略

++idcard_confirm	是	object	用户手动输入或二次确认的身份证信息
+++name	是	string	姓名
+++idcard_number	是	string	证件号
+++idcard_type	是	string	证件类型，大陆居民二代身份证返回0
++idcard_ocr_result	否	object	OCR采集的身份证信息，当方案配置使用OCR采集证件照时返回该参数
++idcard_images	否	object	返回采集的身份证图片信息 当人脸实名认证控制台设置为使用OCR识别时返回此参数信息

- 返回示例

```
{
  "success": true,
  "result": {
    "verify_count": 1,
    "ocr_charge_count": 0,
    "match_charge_count": 0,
    "verify_charge_count": 1,
    "verify_result": [
      {
        "order": 1,
        "code": "0",
        "message": "核验成功",
        "is_verify_passed": true,
        "verify_time": "2024-04-02 16:15:01",
        "is_charged": true,
        "charge_type": "verify",
        "charge_time": "2024-04-02 16:15:01",
        "is_ocr_charge": false,
        "ocr_count": 0,
        "ocr_charge_time": null,
        "verify_detail": {
          "score": 84.4000015258789,
          "threshold": 80.0,
          "app": true,
          "face_image": "https://bj.bcebos.com/v1/mingjing-qa/hb/h5-video-
temp/temp/202404/16130/ZWCCBxXUz48Q9wuR/lgi1n0SpQRtzm9ySWLQwdwvn-00.jpg?authorization=bce-auth-
v1%2F89e9208d0d5a497cbc8922a4d74d4f71%2F2024-04-
02T08%3A19%3A23Z%2F86400%2F%2Fb3abf9eef0b7f897ccf0350d7d59bbfd4498ab214ef73d437a45e319e5028688",

          "verify_log_id": "1775074433891895605",
          "liveness_score": 0.99999183,
          "spoofing_score": null,
          "risk_level": null,
          "risk_tag": null,
          "is_demote": false
        },
        "idcard_confirm": {
          "name": "李**",
          "idcard_number": "140522199812345678",
          "idcard_type": "0",
          "idcard_ocr_result": null,
          "idcard_images": null
        }
      }
    ]
  },
  "log_id": "1775075543214005842"
}
```


12. OCR身份证识别相关接口

12.1 OCR身份证识别初始化接口

API	描述
-(void) authWithLicenseFileData: (NSData *)licenseFileContent;	将aip.license文件的NSData数据传入OCR SDK进行初始化和鉴权

入参说明

参数	类型	含义
licenseFileContent	NSData	将aip.license文件读出来，转发NSData类型

注：如果aip.license文件鉴权文件正确，那么可以使用OCR识别成功识别身份证信息。

12.2 OCR身份证识别接口 支持对二代居民身份证字段进行结构化识别，包括姓名、性别，调用参考OCR身份证识别接口文档。

返回值	API	描述
UIViewController	-(UIViewController *)startOcrRecognize:(BDAipOCRConfig *)config didGetImage:(void(^)(UIImage *image))getImageAction success:(void(^)(NSDictionary *result))success failure:(void (^)(NSError *error))failure;	OCR识别接口

参数值列表如下：

参数	类型	说明	必选
config	BDAipOCRConfig	用于设置OCR页面相关UI	否
getImageAction	Block	获取到的图片信息回调	否
success	Block	获取身份信息成功回调	否
failure	Block	获取身份信息失败回调	否

config配置说明

属性	类型	说明
titleViewBgColor	UIColor	OCR页面导航栏背景色，默认为白色
pageTitle	NSString	OCR标题内容，默认为"身份信息采集"
pageTitleTextColor	UIColor	OCR标题颜色，默认为黑色
topTipText	NSString	OCR顶部扫描文字，默认为"请将您本人的\n身份证人像面放入框内"
topTipTextColor	UIColor	OCR 顶部文字颜色，默认为白色

权限

名称	用途
Privacy - Camera Usage Description	获取相机权限，需要使用相机做人脸识别和OCR识别
Privacy - Photo Library Additions Usage Description	iOS 11及以后保存照片到相册权限,OCR识别需要从相册选择照片
Privacy - Photo Library Usage Description	使用相册权限，OCR识别需要从相册选择照片
Privacy - Microphone Usage Description	人脸识别视频录制功能会使用该权限

只使用人脸采集功能，不使用OCR

- **运行示例工程代码**

点击开始身份核验，走完所有流程，可以出现如下界面：

- **删除库文件：AipOcrSdk.framework，AipBase.framework，IdcardQuality.framework**

- **全局搜索与AipOcrSDK/AipOcrSdk.h**

找到后并删除项目中所有引入对应SDK的代码，也可以注释，在删除后可能导致编译不通过，可以根据报错部分保证正常功能不受影响的同时进行相应的代码注释。

- **全局搜索 -(void) startOCRSdk 并注释该函数**

注意：其他编译报错部分调用的地方也需要注释或删除

```
// 打开相机扫描
 - (void)startOCRSdk {
[self configCallback];
// 身份证识别
[AipCaptureCardVC clearIdCard];
BDAipOCRConfig *config = [[BDAipOCRConfig alloc] init];
AipNavigationController *detectNavi = [AipCaptureCardVC viewControllerToDetectIdCard:config
                                    handler:^(UIImage *image) {
    _idCardImage = image;
    [[AipOcrService shardService] detectIdCardFrontFromImage:image
                                        withOptions:nil
                                      successHandler:^(id result) {
        _successHandler(result);
    } failHandler:_failHandler];
}];
UIViewController *detectRoot = [[UIViewController alloc] init];
detectRoot.view.backgroundColor = [UIColor whiteColor];
detectRoot.view.clipsToBounds = YES;
[detectRoot addChildViewController:detectNavi];
[detectRoot.view addSubview:detectNavi.view];

[self.navigationController pushViewController:detectRoot animated:YES];

_vc = detectRoot;
__weak typeof(detectRoot) weakDetectRoot = detectRoot;
detectNavi.captureController.backAction = ^{
    [weakDetectRoot.navigationController popViewControllerAnimated:YES];
};}
```

- **编译运行代码首页并将身份证OCR功能关闭或者使用代码**

```
 // 身份证获取方式  1 OCR扫描   0  手动输入  2代码传
+ (int)useOCR {
BDConfigDataService *service = [self sharedInstance];
NSNumber *value = service.sharedDic[BDConfigDataServiceKeyForSettingOcr];
if (value) {
    return value.intValue;
}
return [[BDConfigFileParser sharedInstance] useOCR];}
```

- **将userOCR的值设置为0**

保证程序只能手动输入身份信息来调用人脸，此部分功能模块不使用也可以对应删除。

运行工程

常见问题

verify_token的详细说明

（1）access_token 有效期为 30 天，重复生成 access_token 的话，对之前的不影响，依旧能使用；access_token 与 verify_token 是包含关系，即 verify_token 是由 access_token生成的，如果一个 verify_token 是和一个不匹配的 access_token 使用，会提示“Token无效或者已过期”

（2）verify_token 的核验有效期为 2 小时，在有效期内可以进行了核验动作，如果超过了 2 小时没有使用该 token 进行核验的话会提示“Token无效或者已过期”

（3） verify_token 核验完毕后（核验成功，或设置了认证未通过时URL失效），无法再进行核验，如果再次进行核验，会提示“Token无效或者已过期”

（4） verify_token 核验完毕后（核验成功，或设置了认证未通过时URL失效），支持对后验接口的查询，有效期均为 3 天，3 天后再次查询后验接口，会提示“Token无效或者已过期”

获取认证人脸、查询认证结果、核验及计费信息获取接口持续返回错误码18，提示openapi限制，且有返回logid 答：检查APP方案依赖的“人脸实名认证V3/V4”、“人脸对比V3/V4”、“在线图片活体检测V3/V4” 三项付费接口，是否有免费额度，或者直接开通后付费，以消除该报错提示。开通后，再进行重试。

HarmonyOS-方案集成指南

前言

> 【本文】方案集成指南适用于需要集成APP方案，实现**权威库核验**、**自传照片人脸比对**、**仅活体检测**这3个业务需求时来参考。百度人脸实名认证APP方案默认开启风控功能，即使用人脸识别V4系列API接口接受SDK端传入本地环境扫描的设备指纹及安全信息，对SDK端进行设备风险识别，辨别是否为风险设备、并返回识别结果。**此方案可有效防御黑产批量虚拟机、病毒侵入等攻击手段，降低第三方黑产攻破概率，提升业务安全性。**

> 特殊的，如果控制台**风控选项**设定“**关闭**”，则需要配合开通人脸识别V3系列API接口实现**权威库核验**、**自传照片人脸比对**、**仅活体检测**这3个业务需求。

快速定位

人脸实名认证（权威库核验）实现方式

人脸比对（自传图片）实现方式

仅活体检测实现方式

获取认证人脸接口（服务端）

核验及计费结果获取（服务端）

1. 文档说明

文档名称	**人脸实名认证APP方案 6.4版本集成文档**
所属平台	HarmonyOS
提交日期	2024-09-03

2. 版本说明

名称	**版本号**
名镜方案	6.4.0
系统支持	HarmonyOS NEXT.0.0.36

3. SDK说明

开启大数据风控

文件名称	版本号	说明
lib_Enhance.har	人脸实名认证SDK业务逻辑封装	包含人脸采集、安全风控等功能

未开启大数据风控（不推荐）

文件名称	版本号	说明
lib_Basic.har	人脸实名认证SDK业务逻辑封装	包含人脸采集等功能

注：获取SDK可参考：获取示例工程

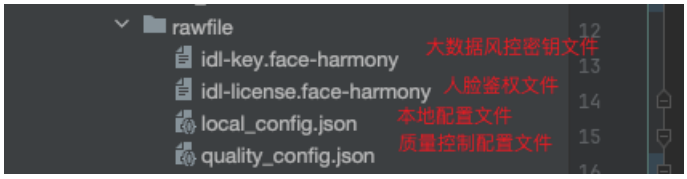
4. 配置包名和签名

- 从百度云控制台下载鸿蒙Demo之后，需要在build-profile.json5中配置好宿主应用的签名信息，更新宿主应用的所有签名信息material

```
],
"signingConfigs": [
  {
    "name": "default",
    "type": "HarmonyOS",
    "material": {
      "storePassword":"替换为签名密码",
      "certpath":"替换cert文件路径",
      "keyAlias":"替换为签名别名",
      "keyPassword":"替换为签名密码",
      "profile":"替换profile文件路径",
      "signAlg":"SHA256withECDSA",
      "storeFile":"替换为签名文件路径"
    }
  }
]
```

5. SDK集成与授权

- 首先在app工程中增加**3. SDK说明**中的依赖库资源
- 其次将百度云控制台创建应用时获取的人脸授权文件(**idl-license.face-harmony**)、大数据风控密钥文件(**idl-key.face-android**)、本地配置文件(**local_config.json**)、质量控制配置文件(**quality_config.json**)放置于rawfile目录下。



- 最后参考控制台下载的工程以及相关接口说明完成SDK集成。

6. 人脸初始化接口（SDK）

初始化接口调用

返回值	API	描述
void	init(context: Context, licenseKey: string, licenseName: string, keyName: string, faceInitCallback: FaceInitCallback)	人脸初始化接口

入参说明

参数	类型	说明
context	Context	上下文
licenseKey	String	授权Key
licenseName	String	授权文件名称
keyName	String	加解密文件名称

onCallback回调说明

参数	类型	含义	值
resultCode	int	错误码	1000为成功，其他为失败，详情参考resultCode错误码说明
resultMsg	String	详情见resultCode错误码说明	

resultCode错误码说明

resultCode	resultMsg	自查方案
1001	License未初始化	请按照集成文档说明完成SDK初始化
1002	License数据解密失败	请检查License文件是否正确
1003	Licesen数据格式错误	请检查license文件内容有被修改过
1004	License-Key校验错误	请检查工程代码初始化参数中的licenseId，和控制台下载工程里面的licenseId是否匹配
1006	鸿蒙应用指纹校验错误	请检查工程所使用的签名文件，和控制台填写的签名信息是否匹配
1008	鸿蒙bundleName应用名称校验错误	请检查工程代码中的bundleName（包名）和控制台填写的鸿蒙包名信息是否匹配
1009	过期时间不正确	请提交工单或者线下联系百度产研人员
1011	授权已过期	请查看当前设备时间是否已不在授权文件有效期内
1012	本地文件读取失败	请检查授权文件名称以及路径
1013	本地鉴权失败	1）请检查工程代码中的bundleName（包名）和控制台填写的鸿蒙包名是否匹配；2）请检查工程所使用的签名文件，和控制台填写的签名信息是否匹配
1014	本地时间校验错误	请检查当前设备时间是否早于实际时间

7. 获取verify_token接口（服务端）

本接口为实名认证APP方案的verify_token获取接口，利用所获取的verify_token进行实名认证流程的有效期为**2小时**。

在线调试 您可以在 示例代码中心 中调试该接口，可进行签名验证、查看在线调用的请求内容和返回结果、示例代码的自动生成。

调用方式

请求URL数据格式

向API服务地址使用POST发送请求，必须在URL中带上参数`access_token`，可通过后台的API Key和Secret Key生成，具体方式请参考“Access Token获取”。

> **注意**：`access_token`的有效期为30天，**切记需要每30天进行定期更换，或者每次请求都拉取新token**；

POST中Body的参数，按照下方请求参数说明选择即可。

> **提示**：如果您为百度云老用户，正在使用其他非AI的服务，可以参考百度云AKSK鉴权方式发送请求，虽然**请求方式**和**鉴权方法**和本文所介绍的不同，但请求参数和返回结果一致。

请求说明

注意事项：

- **请求体格式化**：Content-Type为`application/json`，通过`json`格式化请求体。

请求示例

HTTP方法：`POST`

请求URL：`https://aip.baidubce.com/rpc/2.0/brain/solution/faceprint/verifyToken/generate`

URL参数：

参数	值
access_token	通过API Key和Secret Key获取的access_token,参考“Access Token获取”

Header：

参数	值
Content-Type	application/json

Body中放置请求参数，参数详情如下：

请求参数：

参数	必选	类型	说明
match_source	是	int	方案类型（比对源）；0：实名认证（权威库）；1：人脸比对（上传照片）；2：仅活体检测。
plan_id	是	string	方案的id信息，请在人脸实名认证控制台查看创建的APP方案ID信息

- 请求示例

```
 {
"match_source": 0, //以权威库核验场景为例，需要传入0
"plan_id": 16144,
}
```

返回参数

- 返回结果

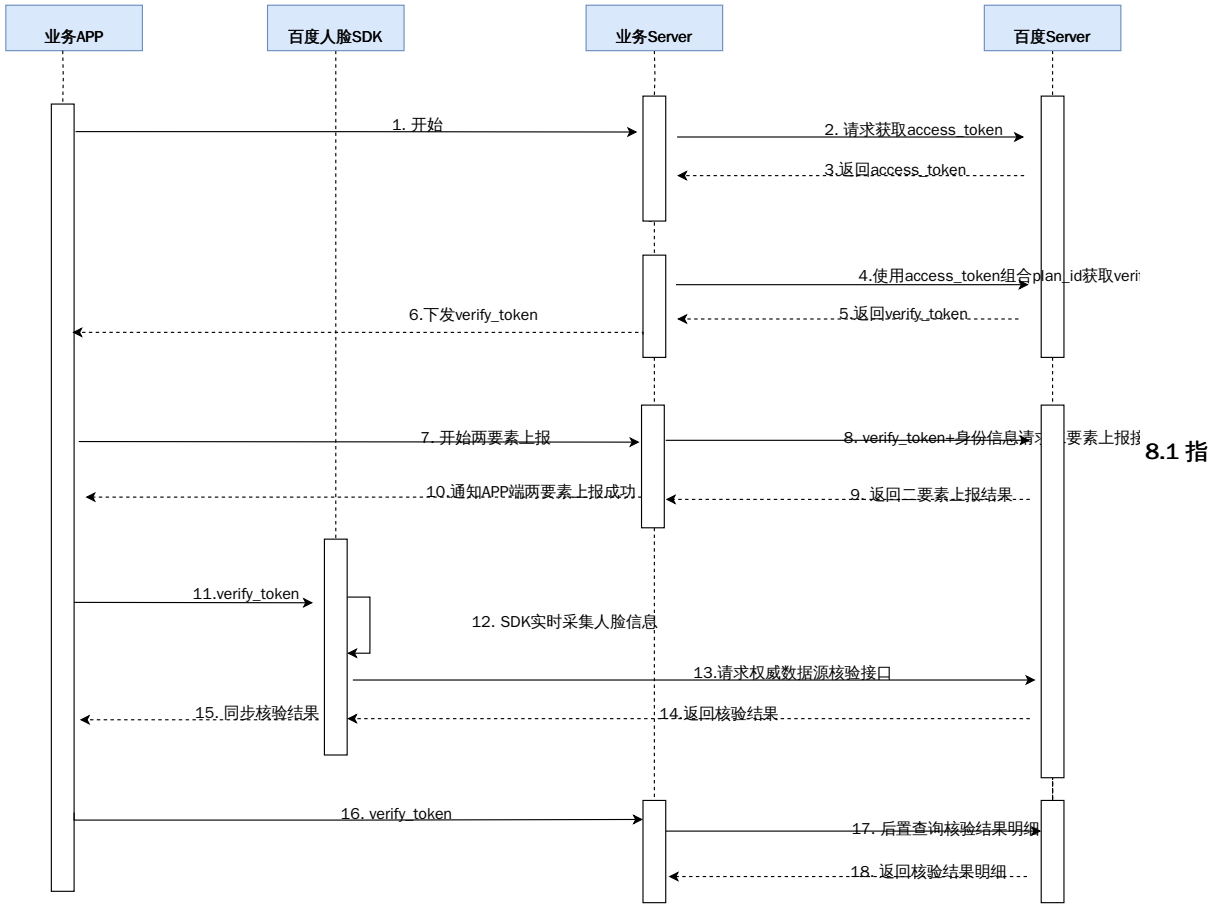
字段	必选	类型	说明
success	是	boolean	返回请求是否成功信息。 若请求成功返回ture; 请求失败则返回false
result	是	object	请求结果
+verify_token	是	string	请求获取的verify_token

- 返回示例

```
{
  "success": true,
  "result": {
     "verify_token": "Yz9rWITm4vak16PBAh5x8oG7"
  },
  "log_id": "1814798895"
}
```

8. 人脸实名认证实现方式

实现方式请参考如下时许图



8.1 指定用户信息上报接口（服务端） 本接口用于，控制台APP方案中选择身份信息录入方式为**业务调用时传入**，需要先调用此接口输入指定用户的姓名+身份证号信息，再继续请求 **人脸权威库核验接口（SDK）**

在线调试 您可以在 示例代码中心 中调试该接口，可进行签名验证、查看在线调用的请求内容和返回结果、示例代码的自动生成。 **调用方式**

请求URL数据格式

向API服务地址使用POST发送请求，必须在URL中带上参数`access_token`，可通过后台的API Key和Secret Key生成，具体方式请参考“Access Token获取”。

> **注意**：`access_token`的有效期为30天，**切记需要每30天进行定期更换，或者每次请求都拉取新token**；

POST中Body的参数，按照下方请求参数说明选择即可。

> **提示**：如果您为百度云老用户，正在使用其他非AI的服务，可以参考百度云AKSK鉴权方式发送请求，虽然**请求方式**和**鉴权方法**和本文所介绍的不同，但请求参数和返回结果一致。

请求说明

注意事项：

- **请求体格式化**：Content-Type为`application/json`，通过`json`格式化请求体。

请求示例

HTTP方法：`POST`

请求URL：https://aip.baidubce.com/rpc/2.0/brain/solution/faceprint/idcard/submit

URL参数：

参数	值
access_token	通过API Key和Secret Key获取的access_token,参考“Access Token获取”

Header：

参数	值
Content-Type	application/json

Body中放置请求参数，参数详情如下：

参数	必选	类型	值
verify_token	是	string	通过access_token获取的verify_token
id_name	是	string	指定输入用户的姓名信息
id_no	是	string	指定输入用户的身份证件号信息
certificate_type	否	int	证件类型： 0 大陆居民二代身份证 4 港澳台居民居住证

- 请求示例：

```
{
  "verify_token": "2sF3nE5mXOHkx2aQwWG4n5WI",
  "id_name": "张三",
  "id_no": "500***********3390",
  "certificate_type": 0          // 证件类型：0:大陆居民二代身份证,4:港澳台居民居住证
}
```

返回参数

- 返回结果

字段	必选	类型	说明
success	是	boolean	返回请求是否成功信息。 若请求成功返回ture; 请求失败则返回false
result	是	int	请求结果，返回固定结果1，可忽略
log_id	是	string	本次查询接口请求的logid

- 返回示例

```
{
  "success": true,
  "result": 1,
  "log_id": "1244068892"
}
```

8.2 人脸权威库核验接口（SDK） 基于姓名、身份证号、当前SDK获取的人脸图片，与权威库进行对比，并得出比对分数，并基于此进行业务判断是否为同一人。

返回值	API	描述
void	startFaceVerify(params: HashMap<string, Object>, faceServiceCallbck: FaceServiceCallbck)	SDK核验接口

入参params (HashMap类型) key值列表如下：

参数	类型	说明	必选
verify_token	String	需要APP服务端通过AK、SK获取access_token，参考 https://ai.baidu.com/docs#/Auth/top,然后通过access_token获取verify_token **【此处需要注意】verify_token有效期两个小时，建议每次调用接口重新获取verify_token**	是
plan_id	String	在控制台配置的方案Id	否
certificate_type	int	证件类型；0：中国居民二代身份证，1：港澳台来往内地通行证，2：外国人永久居留证，3：定居国外的中国公民护照，4：港澳台居民居住证。默认为0	否
match_source	int	方案类型（比对源）；0：实名认证（权威库）；1：人脸比对（上传照片）；2：仅活体检测；**这里默认传入0**	是

onCallback回调说明

参数	类型	含义	值
resultCode	int	错误码	0为成功，其他为失败，详情参考resultCode错误码说明
resultMap	HashMap	回调结果Map	详情见下表

resultMap key值列表说明：

Key值	类型	含义
resultMsg	String	详情见resultCode错误码说明
data	String	服务端返回结果json,只有错误码为0时会返回此字段

- resultCode和resultMsg说明

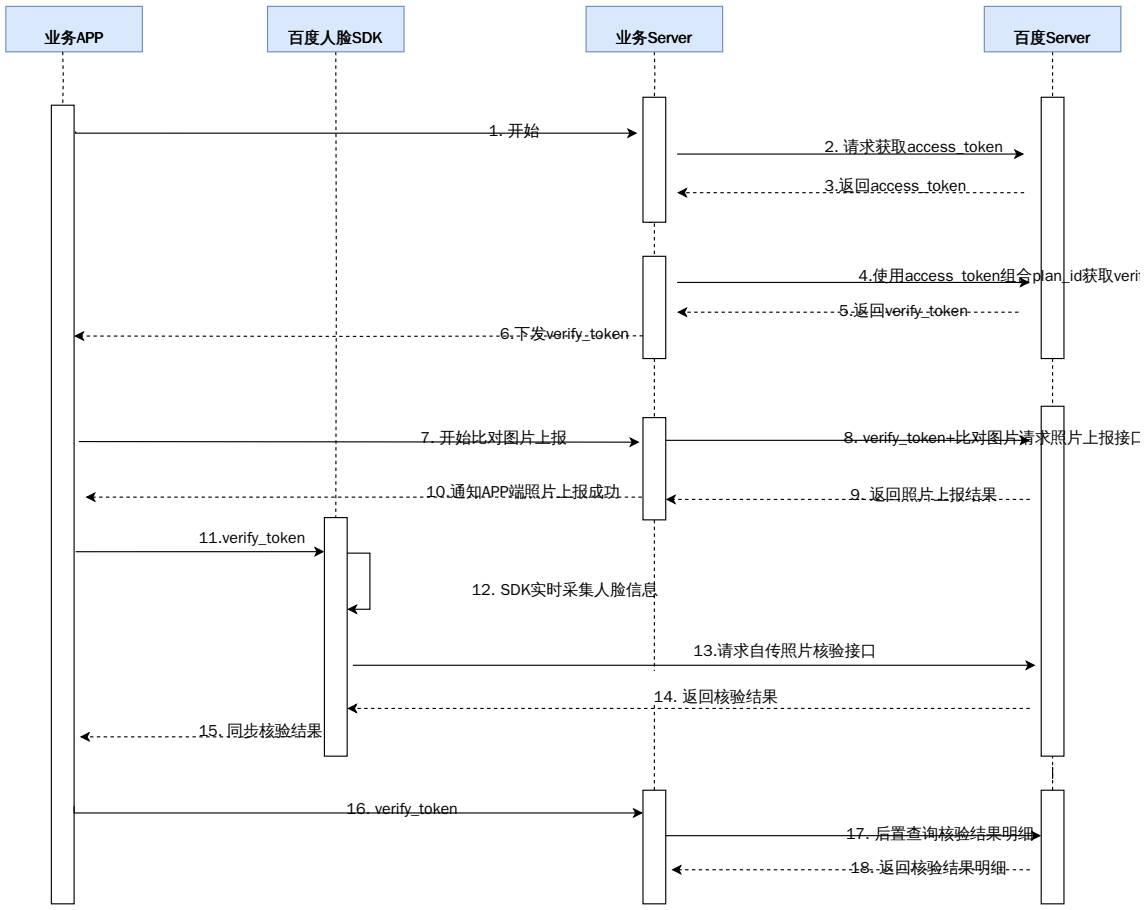
resultCode	resultMsg	自查方案
-102	采集流程取消	用户自行取消采集流程
-103	SDK未初始化	需要检查初始化方法有没有调用，建议在application里面调用
-303	视频录制错误	是否授权视频/音频
-310	摄像头调起失败	请检查摄像头权限
-401	超时	可通过设置延长超时时间限制
-402	炫瞳异常	炫瞳打光颜色与模型输出不匹配
-1001	网络异常	请检查网络通讯

- data返回示例

```
{
 "error_code": 0, //认证建议结果为通过，该结果仅供参考；可通过调用服务端getall接口获取最终认证结果
 "log_id": "1790651229035123218"
}
```

9. 人脸对比（自传照片）实现方式

实现方式请参考如下时许图



9.1 对

比图片上传接口（服务端） 本接口适用于人脸比对（自传照片）业务场景，为了保证用户信息安全，您需要提前通过服务端请求本接口来上报比对人脸信息。

在线调试 您可以在 示例代码中心 中调试该接口，可进行签名验证、查看在线调用的请求内容和返回结果、示例代码的自动生成。 **调用方式**

请求URL数据格式

向API服务地址使用POST发送请求，必须在URL中带上参数`access_token`，可通过后台的API Key和Secret Key生成，具体方式请参考“Access Token获取”。

> **注意**：`access_token`的有效期为30天，**切记需要每30天进行定期更换，或者每次请求都拉取新token**；

POST中Body的参数，按照下方请求参数说明选择即可。

> **提示**：如果您为百度云老用户，正在使用其他非AI的服务，可以参考百度云AKSK鉴权方式发送请求，虽然**请求方式**和**鉴权方法**和本文所介绍的不同，但请求参数和返回结果一致。

请求说明

注意事项：

- **请求体格式化**：Content-Type为`application/json`，通过`json`格式化请求体。

请求示例

HTTP方法：`POST`

请求URL： https://aip.baidubce.com/rpc/2.0/brain/solution/faceprint/uploadMatchImage

URL参数：

参数	值
access_token	通过API Key和Secret Key获取的access_token,参考“Access Token获取”

Header：

参数	值
Content-Type	application/json

Body中放置请求参数，参数详情如下：

参数	必选	类型	值
verify_token	是	string	通过access_token获取的verify_token
image	是	string	图片base64字符串，编码后的图片大小不超过10M，图片分辨率小于1920*1080
quality_control	否	string	“NONE”、“LOW”、“NORMAL”、“HIGH”； 质量控制参数，未主动传入时默认为“NONE”
liveness_control	否	string	“NONE”、“LOW”、“NORMAL”、“HIGH”； 活体控制参数，未主动传入时默认为“NONE”

- 请求示例：

```
{
 "verify_token": "2sF3nE5mXOHkx2aQwWG4n5WI",
 "image": "/9j/4AAQSkZJRgABAQAAAQABAAD/",
 "quality_control" : "LOW",
 "liveness_control" : "LOW",
}
```

返回参数

- 返回结果

字段	必选	类型	说明
success	是	boolean	返回请求是否成功信息。 若请求成功返回ture; 请求失败则返回false
log_id	是	string	logid
result	是	object	返回结果

- 返回示例

```
{
 "success": true,
 "result": null,
 "log_id": "1329130892"
}
```

9.2 人脸对比（SDK） 包含活体检测、质量检测、人脸1:1识别功能。

返回值	API	描述
void	startFaceVerify(params: HashMap<string, Object>, faceServiceCallbck: FaceServiceCallbck)	SDK核验接口

入参params (HashMap类型) key值列表如下：

参数	类型	说明	必选
verify_token	String	需要APP服务端通过AK、SK获取access_token，参考 https://ai.baidu.com/docs#/Auth/top,然后通过access_token获取verify_token **【此处需要注意】verify_token有效期两个小时，建议每次调用接口重新获取verify_token**	是
match_source	int	方案类型（比对源）；0：实名认证（权威库）；1：人脸比对（上传照片）；2：仅活体检测；**这里默认传入1**	是

onCallback回调说明

参数	类型	含义	值
resultCode	int	错误码	0为成功，其他为失败，详情参考实名认证resultCode错误码说明
resultMap	HashMap	回调结果Map	详情见下表

resultMap key值列表说明：

Key值	类型	含义
resultMsg	String	详情见resultCode错误码说明
data	String	服务端返回结果json,只有错误码为0时会返回此字段

- resultCode和resultMsg说明

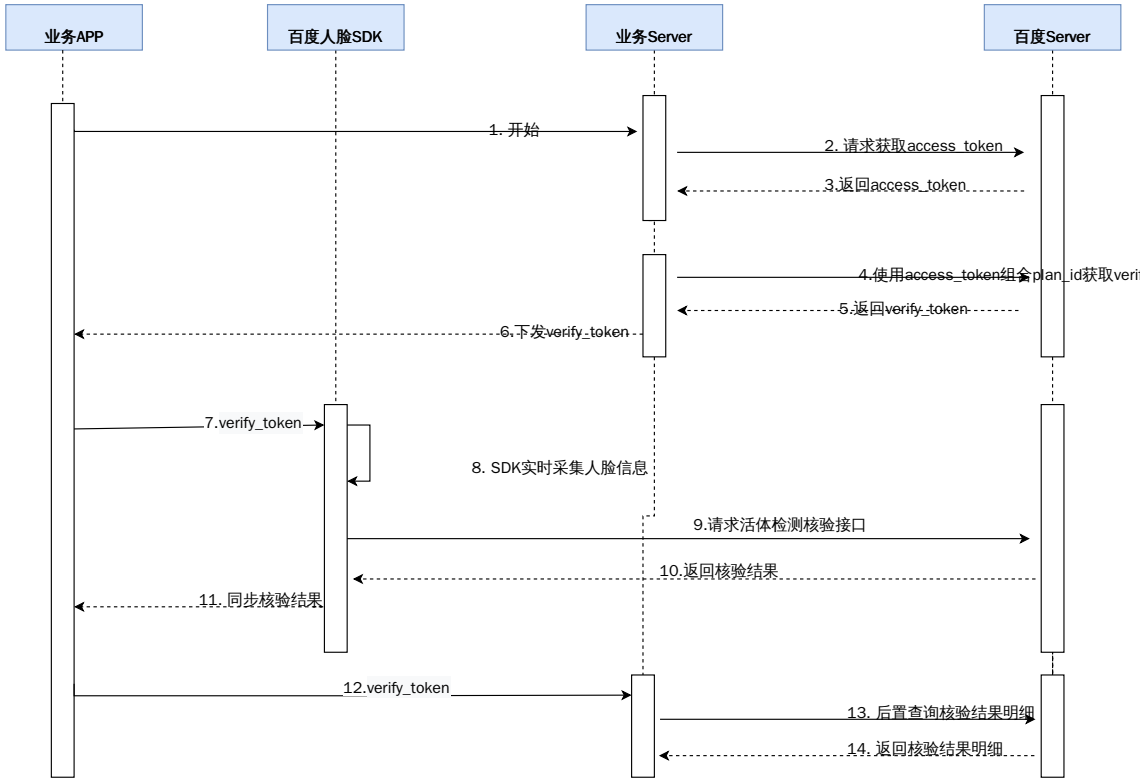
resultCode	resultMsg	自查方案
-102	采集流程取消	用户自行取消采集流程
-103	SDK未初始化	需要检查初始化方法有没有调用，建议在application里面调用
-303	视频录制错误	是否授权视频/音频
-310	摄像头调起失败	请检查摄像头权限
-401	超时	可通过设置延长超时时间限制
-402	炫瞳异常	炫瞳打光颜色与模型输出不匹配
-1001	网络异常	请检查网络通讯

- data返回示例

```
{
 "error_code": 0, //认证建议结果为通过，该结果仅供参考；可通过调用服务端getall接口获取最终认证结果
 "log_id": "1790651229035123218"
}
```

10. 人脸活体检测实现方式

实现方式请参考如下时许图



10.1 活体检测（SDK）

包含本地活体加云端活体，本地活体分静默活体、炫瞳活体、动作活体三种，云端活体可以判断图片中的人脸是否为二次翻拍以及是否为合成图攻击。实现二次验证采集图片是否存在假体攻击破绽的情况。
如您的业务场景核心为人脸实名认证（权威源），请直接参考 **8.2 人脸权威库核验接口（SDK）** 。

返回值	API	描述
void	startFaceVerify(Context context, Map params, FaceServiceCallbck FaceServiceCallbck)	SDK核验接口

入参params (HashMap类型) key值列表如下：

参数	类型	说明	必选
verify_token	String	需要APP服务端通过AK、SK获取access_token，参考 https://ai.baidu.com/docs#/Auth/top,然后通过access_token获取verify_token **【此处需要注意】verify_token有效期两个小时，建议每次调用接口重新获取verify_token**	是
match_source	int	方案类型（比对源）；0：实名认证（权威库）；1：人脸比对（上传照片）；2：仅活体检测；**这里默认传入2.**	是

onCallback回调说明

参数	类型	含义	值
resultCode	int	错误码	0为成功，其他为失败，详情参考实名认证resultCode错误码说明
resultMap	HashMap	回调结果Map	详情见下表

resultMap key值列表说明：

Key值	类型	含义
resultMsg	String	详情见resultCode错误码说明
data	String	服务端返回结果json,只有错误码为0时会返回此字段

- resultCode和resultMsg说明

resultCode	resultMsg	自查方案
-102	采集流程取消	用户自行取消采集流程
-103	SDK未初始化	需要检查初始化方法有没有调用，建议在application里面调用
-303	视频录制错误	是否授权视频/音频
-310	摄像头调起失败	请检查摄像头权限
-401	超时	可通过设置延长超时时间限制
-402	炫瞳异常	炫瞳打光颜色与模型输出不匹配
-1001	网络异常	请检查网络通讯

- data返回示例

```
{
 "error_code": 0, //认证建议结果为通过，该结果仅供参考；可通过调用服务端getall接口获取最终认证结果
 "log_id": "1790651229035123218"
}
```

11. 验证后查询接口

11.1 获取认证人脸接口（服务端）

本接口返回进行人脸实名认证过程中进行认证的最终采集的人脸信息。根据Verify_token返回的结果信息会在云端保留**3天**，您可根据需要在此期间进行调取查询。

在线调试 您可以在 示例代码中心 中调试该接口，可进行签名验证、查看在线调用的请求内容和返回结果、示例代码的自动生成。

调用方式

请求URL数据格式

向API服务地址使用POST发送请求，必须在URL中带上参数`access_token`，可通过后台的API Key和Secret Key生成，具体方式请参考“Access Token获取”。

> **注意**：`access_token`的有效期为30天，**切记需要每30天进行定期更换，或者每次请求都拉取新token**；

POST中Body的参数，按照下方请求参数说明选择即可。

> **提示**：如果您为百度云老用户，正在使用其他非AI的服务，可以参考百度云AKSK鉴权方式发送请求，虽然**请求方式**和**鉴权方法**和本文所介绍的不同，但请求参数和返回结果一致。

请求说明

注意事项：

- **请求体格式化**：Content-Type为`application/json`，通过`json`格式化请求体。

请求示例

HTTP方法：`POST`

请求URL：`https://aip.baidubce.com/rpc/2.0/brain/solution/faceprint/result/simple`

URL参数：

参数	值
access_token	通过API Key和Secret Key获取的access_token,参考“Access Token获取”

Header：

参数	值
Content-Type	application/json

Body中放置请求参数，参数详情如下：

参数	值
verify_token	通过access_token获取的verify_token

- 请求示例：

```
{
 "verify_token" : "clupeyP51sn28XzxGVTfYqoN"
}
```

返回参数

- 返回结果

字段	必选	类型	说明
success	是	boolean	返回请求是否成功信息。 若请求成功返回ture; 请求失败则返回false
result	是	object	请求结果
+image	是	string	返回采集的用户人脸信息

- 返回示例

```
{
  "success": true,
  "result":{
     "image":"https://brain.baidu.com/solution/faceprint/image/query?verify_token=xxxxxx"
  },
  "log_id": "1054986003"
}
```

11.2 查询认证结果接口

本接口为请求返回的认证结果信息查询，包含用户二次确认的身份证信息，活体检测信息、及用户对权威库图片进行比对的分数信息。（仅在认证成功时返回上述信息，认证失败返回错误码）根据Verify_token返回的结果信息会在云端保留**3天**，您可根据需要在此期间进行调取查询。

在线调试 您可以在 示例代码中心 中调试该接口，可进行签名验证、查看在线调用的请求内容和返回结果、示例代码的自动生成。 **调用方式**

请求URL数据格式

向API服务地址使用POST发送请求，必须在URL中带上参数access_token，可通过后台的API Key和Secret Key生成，具体方式请参考“Access Token获取”。

> **注意**：access_token的有效期为30天，**切记需要每30天进行定期更换，或者每次请求都拉取新token**；

POST中Body的参数，按照下方请求参数说明选择即可。

> **提示**：如果您为百度云老用户，正在使用其他非AI的服务，可以参考百度云AKSK鉴权方式发送请求，虽然**请求方式**和**鉴权方法**和本文所介绍的不同，但请求参数和返回结果一致。

请求说明

注意事项：

- **请求体格式化**：Content-Type为application/json，通过json格式化请求体。

请求示例

HTTP方法：POST

请求URL： https://aip.baidubce.com/rpc/2.0/brain/solution/faceprint/result/detail

URL参数：

参数	值
access_token	通过API Key和Secret Key获取的access_token,参考“Access Token获取”

Header：

参数	值
Content-Type	application/json

Body中放置请求参数，参数详情如下：

参数	值
verify_token	通过access_token获取的verify_token

请求示例：

```
 {
  "verify_token"："clupeyP51sn28XzxGVTfYqoN"
}
```

返回参数

- 返回结果

字段	必选	类型	说明
success	是	boolean	返回本次身份核验是否成功的结果。 成功返回ture; 失败则返回false
result	是	object	请求结果
+verify_result	是	object	认证返还信息
++verify_log_id	是	string	核验结果对应的日志ID
++liveness_score	是	float	**活体检测分数：** 活体验证通过时返回活体分数，不通过则返回0。
++score	是	float	人脸实名认证比对得分**（仅活体检测时返回为0）**
++risk_level	否	string	安全风控等级 方案配置启用安全风控，将返回该字段 值为1或2时表示触发了安全风险，值为3或4时无风险
++risk_tag	否	string	安全风控标签 方案配置启用安全风控，将返回该字段 当risk_level值为1或2时，返回具体风险类型，risk_level值为3或4时，为空
++spoofing	是	float	合成图分数 若未进行合成图检测，则返回0 若进行活体检测，则返回合成图检测分值
+idcard_confirm	是	object	用户二次确认的身份证信息
++name	是	string	姓名
++idcard_number	是	string	身份证号
code	否	int	错误码 **（仅认证失败时展示）**
message	否	string	错误信息 **（仅认证失败时展示）**
log_id	是	string	本次查询接口请求的日志ID

- 返回示例


```
{
 "success": true,
 "result": {
    "verify_result": {
        "verify_log_id": "1868893431661688963",
        "score": 94.57,
        "risk_level": "3",
        "risk_tag": "[]",
        "liveness_score": 0.99983007,
        "spoofing": 0.0
    },
    "idcard_confirm": {
        "idcard_number": "430419********4532",
        "name": "陈**"
    }
 },
 "log_id": "1868914111329362938"
}
```


- 返回失败示例

```
{
"success": false,
"result": {
    "verify_result": {
        "verify_log_id": "1866379694929155276",
        "score": 0.0052550267,
        "risk_level": "3",
        "risk_tag": "[]",
        "liveness_score": 0.0052550267,
        "spoofing": 0.0
    },
    "idcard_confirm": {
        "idcard_number": null,
        "name": null
    }
},
"code": 223120,
"message": "活体检测未通过",
"log_id": "1866379737400667552"
}
```


11.3 核验及计费信息获取（服务端）

根据Verify_token返回的信息会在云端保留**3天**，您可根据需要在此期间进行调取查询。

调用方式

请求URL数据格式

向API服务地址使用POST发送请求，必须在URL中带上参数`access_token`，可通过后台的API Key和Secret Key生成，具体方式请参考“Access Token获取”。

> **注意**：`access_token`的有效期为30天，**切记需要每30天进行定期更换，或者每次请求都拉取新token**；

POST中Body的参数，按照下方请求参数说明选择即可。

> **提示**：如果您为百度云老用户，正在使用其他非AI的服务，可以参考百度云AKSK鉴权方式发送请求，虽然**请求方式**和**鉴权方法**和本文所介绍的不同，但请求参数和返回结果一致。

在线调试 您可以在 示例代码中心 中调试该接口，可进行签名验证、查看在线调用的请求内容和返回结果、示例代码的自动生成。

请求说明

注意事项：

- **请求体格式化**：Content-Type为`application/json`，通过`json`格式化请求体。

请求示例

HTTP方法：`POST`

请求URL：`https://aip.baidubce.com/rpc/2.0/brain/solution/faceprint/result/getall`

URL参数：

参数	值
access_token	通过API Key和Secret Key获取的access_token,参考“Access Token获取”

Header：

参数	值
Content-Type	application/json

Body中放置请求参数，参数详情如下：

参数	值
verify_token	通过access_token获取的verify_token

请求示例：

```
{
 "verify_token" : "clupeyP51sn28XzxGVTfYqoN"
}
```

返回参数

- 返回结果

字段	必选	类型	说明
success	是	boolean	返回请求是否成功信息。 若请求成功返回ture; 请求失败则返回false
log_id	是	string	本次查询接口请求的logid
result	是	object	结果信息
+verify_count	是	int	当前verify_token对应的核验次数，核验次数为几，后面就有几个对象数组 如方案配置为“认证未通过时URL失效”，则该verify_token核验次数为1； 如方案配置为“认证未通过时URL不失效”，则该verify_token对应的核验次数可能大于1（用户核验失败后多次重试）
+ocr_charge_count	是	int	身份证识别OCR接口的计费次数 //鸿蒙APP方案暂不支持
+match_charge_count	是	int	人脸对比接口的计费次数
+verify_charge_count	是	int	人脸实名认证接口的计费次数
+verify_result	是	object[]	核验结果信息，verify_count的值为几，就返回几个该对象
++order	是	int	代表本次核验结果在verify_token所有核验次数中的顺序，按时间先后排序，第一次核验返回值为 1，第二次核验返回值为 2，以此类推
++is_verify_passed	是	boolean	本次核验是否通过 核验成功返回true 核验失败返回false
			本次核验的错误码

++code	是	string	核验成功时返回 0 核验失败返回非0错误码
++message	是	string	本次核验的错误信息 核验成功时返回“核验成功” 核验失败时返回具体错误原因，如“身份证姓名不匹配”
++verify_time	是	string	本次核验完成的时间点，包含年月日、时分秒，如“2023-6-20 18:00:00”
++is_charged	是	boolean	本次核验是否计费 计费返回true 不计费返回false
++charge_type	是	string	计费类型：**verify**(人脸实名认证)、**match**(人脸对比)、**live**(在线图片活体)
++charge_time	是	string	本次计费的时间点，如不计费则该字段为空，包含年月日、时分秒，如“2023-6-20 18:00:00”
++is_ocr_charge	否	boolean	身份证识别OCR是否计费 //鸿蒙APP方案暂不支持 计费返回true 不计费返回false
++ocr_charge_time	否	string	ocr 收费时间点，如不收费则该字段为空，包含年月日、时分秒，如“2023-6-20 18:00:00”
++ocr_count	否	int	ocr次数
++verify_detail	是	object	核验的详细信息
+++face_image	是	string	本次核验流程中采集的人脸最佳质量图下载url
+++verify_log_id	是	string	本次核验流程中请求的人脸实名认证（方案配置权威库比对）、人脸对比（方案配置自建人脸库比对）、在线图片活体（仅活体检测）后端接口logid，支持在记录查询平台中通过logid查询到3天内的记录。 少数核验失败情况下，实际并未发生上述俩接口的请求，则该字段为空，如：用户拒绝摄像头授权且不允许降级
+++score	是	float	人脸相似度得分，大于等于方案设置阈值为同一人；当身份证姓名不匹配或活体不通过时，该项为空
+++threshold	是	float	本次核验时，所使用的方案配置相似度阈值
+++liveness_score	是	float	活体检测得分，注：预留功能字段，当前返回为0
+++spoofing_score	否	float	方案配置使用合成图功能，将返回合成图得分，注：预留功能字段，当前返回为0
+++risk_level	否	string	安全风控等级 方案配置启用安全风控，将返回该字段 值为1或2时表示触发了安全风险，值为3或4时无风险
+++risk_tag	否	string	安全风控标签 方案配置启用安全风控，将返回该字段 当risk_level值为1或2时，返回具体风险类型，risk_level值为3或4时，为空
+++is_demote	否	boolean	H5方案相关，可忽略

++idcard_confirm	是	object	用户手动输入或二次确认的身份证信息
+++name	是	string	姓名
+++idcard_number	是	string	证件号
+++idcard_type	是	string	证件类型，大陆居民二代身份证返回0
++idcard_ocr_result	否	object	OCR采集的身份证信息，当方案配置使用OCR采集证件照时返回该参数 //鸿蒙APP方案暂不支持
++idcard_images	否	object	返回采集的身份证图片信息 当人脸实名认证控制台设置为使用OCR识别时返回此参数信息 //鸿蒙APP方案暂不支持

- 返回示例

```
{
  "success": true,
  "result": {
    "verify_count": 1,
    "ocr_charge_count": 0,
    "match_charge_count": 0,
    "verify_charge_count": 1,
    "verify_result": [
      {
        "order": 1,
        "code": "0",
        "message": "核验成功",
        "is_verify_passed": true,
        "verify_time": "2024-04-02 16:15:01",
        "is_charged": true,
        "charge_type": "verify",
        "charge_time": "2024-04-02 16:15:01",
        "is_ocr_charge": false,
        "ocr_count": 0,
        "ocr_charge_time": null,
        "verify_detail": {
          "score": 84.4000015258789,
          "threshold": 80.0,
          "app": true,
          "face_image": "https://bj.bcebos.com/v1/mingjing-qa/hb/h5-video-temp/temp/202404/16130/ZWCCBxXUz48Q9wuR/lgi1n0SpQRtzm9ySWLQwdwvn-00.jpg?authorization=bce-auth-v1%2F89e9208d0d5a497cbc8922a4d74d4f71%2F2024-04-02T08%3A19%3A23Z%2F86400%2F%2Fb3abf9eef0b7f897ccf0350d7d59bbfd4498ab214ef73d437a45e319e5028688",

          "verify_log_id": "1775074433891895605",
          "liveness_score": 0.99999183,
          "spoofing_score": null,
          "risk_level": null,
          "risk_tag": null,
          "is_demote": false
        },
        "idcard_confirm": {
          "name": "李**",
          "idcard_number": "140522199812345678",
          "idcard_type": "0",
          "idcard_ocr_result": null,
          "idcard_images": null
        }
      }
    ]
  },
  "log_id": "1775075543214005842"
}
```


12. 人脸释放接口（SDK）

人脸释放接口调用，实现对采集功能、模型的释放，减小内存。

返回值	API	描述
void	release()	人脸释放接口

权限

名称	说明	必选
需要动态申请的权限		
ohos.permission.CAMERA	拍照权限	是
不需要动态申请的权限		
ohos.permission.INTERNET	允许访问网络	是

常见问题

如何获取鸿蒙应用指纹 （1）方法1，测试证书

- “应用指纹”signatureInfo.fingerprint是鸿蒙应用签名证书(.cer文件)的SHA-256hash值，当前支持获取本应用的“应用指纹”。示例代码如下：

```
 import { bundleManager } from '@kit.AbilityKit';
import { hilog } from '@kit.PerformanceAnalysisKit';
import { BusinessError } from '@kit.BasicServicesKit';

let bundleFlags = bundleManager.BundleFlag.GET_BUNDLE_INFO_WITH_SIGNATURE_INFO;
try {
  bundleManager.getBundleInfoForSelf(bundleFlags).then((data) => {
    hilog.info(0x0000, 'testTag', 'getBundleInfoForSelf successfully. Data: %{public}s', JSON.stringify(data));
    //data里可以获取到signatureInfo，即应用的签名证书信息
  }).catch((err: BusinessError) => {
    hilog.error(0x0000, 'testTag', 'getBundleInfoForSelf failed. Cause: %{public}s', err.message);
  });
} catch (err) {
  let message = (err as BusinessError).message;
  hilog.error(0x0000, 'testTag', 'getBundleInfoForSelf failed: %{public}s', message);
}
```

（2）方法2，测试证书+发布证书

- 在项目级build-profile.json5文件中，signingConfigs字段内的profile的值即为签名文件的存储路径。
- 打开该签名文件（**后缀为.p7b**），打开后在文件内搜索“-certificate”，将“-----BEGIN CERTIFICATE-----”和“-----END CERTIFICATE-----”以及中间的信息拷贝到新的文本中，注意换行并去掉换行符，保存为一个新的.cer文件，如命名为xxx.cer。：

```
-----BEGIN CERTIFICATE-----
MIICMzCCAbegAwIBAgIEaOC/zDAMBggqhkjOPQQDAwUAMGMxCzAJBgNVBAYTAkNO
MRQwEgYDVQQKEwtPcGVuSGFybW9ueTEZMBcGA1UECxMQT3BlbkhhcmlvbnkgVGVh
bTEjMCEGA1UEAxMaT3BlbkhhcmlvbnkgQXBwbGljYXRpb24gQ0EwHhcNMjEwMjAy
MTIxOTMxWhcNNDkxMjMxMTIxOTMxWjBoMQswCQYDVQQGEwJDTjEUMBIGA1UEChML
T3BlbkhhcmlvbnkxGTAXBgNVBAsTEE9wZW5IYXJtb255IFRlYW0xKDAmBgNVBAMT
H09wZW5IYXJtb255IEFwcGxpY2F0aW9uIFJlbGVhc2UwWTATBgcqhkjOPQIBBggq
hkjOPQMBBwNCAATbYOCQQpW5fdkYHN45v0X3AHaxl2jPBdEDosFRIZleXmxOYzSG
JwMfsHhUU90E81I0TXYZnNmgM1sovubeQqATolIwUDAfBgNVHSMEGDAWgBTbhrci
FtULoUu33SV7ufEFfaItRzAOBgNVHQ8BAf8EBAMCB4AwHQYDVR0OBBYEFPtxruhl
cRBQsJdwcZqLu9oNUVgaMAwGCCqGSM49BAMDBQADaAAwZQIxAJta0PQ2p4DIu/ps
LMdLCDgQ5UHll0B4PGhBlMgdi2zf8nk9spazEQI/0XNwpft8QAIwHSuA2WelVi/o
zAlF08DnbJrOOtOnQq5wHOPlDYB4OtUzOYJk9scotrEnJxJzGsh/
-----END CERTIFICATE-----
```

- 使用keytool工具（在DevEco Studio安装目录下的jbr/bin文件夹内），执行如下命令通过.cer文件获取证书指纹的SHA256值。

  `keytool -printcert -file xxx.cer`

- 将证书指纹中SHA256的内容去掉冒号，即为最终要获得的签名指纹。

verify_token的详细说明 （1）access_token 有效期为 30 天，重复生成 access_token 的话，对之前的不影响，依旧能使用；access_token 与verify_token 是包含关系，即 verify_token 是由 access_token生成的，如果一个 verify_token 是和一个不匹配

的 access_token 使用，会提示“Token无效或者已过期”

（2）verify_token 的核验有效期为 2 小时，在有效期内可以进行了核验动作，如果超过了 2 小时没有使用该 token 进行核验的话会提示“Token无效或者已过期”

（3） verify_token 核验完毕后（核验成功，或设置了认证未通过时URL失效），无法再进行核验，如果再次进行核验，会提示“Token无效或者已过期”

（4） verify_token 核验完毕后（核验成功，或设置了认证未通过时URL失效），支持对后验接口的查询，有效期均为 3 天，3 天后再次查询后验接口，会提示“Token无效或者已过期” **获取认证人脸、查询认证结果、核验及计费信息获取接口持续返回错误码18，提示openapi限制，且有返回logid 答：**检查APP方案依赖的“人脸实名认证V3/V4”、“人脸对比V3/V4”、“在线图片活体检测V3/V4” 三项付费接口，是否有免费额度，或者直接开通后付费，以消除该报错提示。开通后，再进行重试。

SDK合规说明文档

前言

2023年2月工信部发布的**《工业和信息化部关于进一步提升移动互联网应用服务能力的通知》**对SDK、个人信息保护、服务体验等方面提出了具体要求，同时为了帮助开发者向最终用户提供相应功能及服务，特此起草本SDK合规使用指导。 您作为开发者为最终用户提供服务，需知悉并确认将遵守适用的法律法规和相关的标准规范，履行个人信息保护义务，并遵循合法、正当、必要和诚信的原则处理用户个人信息，包括但不限于《中华人民共和国个人信息保护法》、《中华人民共和国网络安全法》、《中华人民共和国数据安全法》以及其他适用的法律法规和相关的标准规范。 此文档用于帮助您更好地了解百度人脸安全采集SDK并合规使用本SDK服务，仅适用于开发者的业务区域为中国大陆地区的场景。

特别提示

本**《SDK合规指南》**是对本 SDK 的合规性和安全性描述与要求仅为我们向您提供的服务说明及使用建议，不构成也不应被视为我们对于任何法律法规及政策文件的及时的、完整的、全面的、甚至完全准确的分析，亦不构成我们对百度人脸安全采集SDK产品和/或服务的承诺和保证。本《指南》不能作为判断您与您所开发、运营的 App 是否满足合规与安全要求的可依赖的标准，亦不构成我们对前述事宜作出的任何担保或保证，您应当自行、独立地对前述 App 承担合规与安全责任。

目录

- SDK收集个人信息的频次、精度、使用目的、场景及对应选择的配置方式、示例
- SDK所需系统权限与功能关系，以及权限申请时机
- SDK扩展业务功能介绍及对应关闭的配置方式、示例
- SDK初始化及各项业务功能接口合规调用时机
- 联系我们

1. SDK收集个人信息不同频次、精度使用目的、场景及对应选择的配置方式、示例

为了帮助开发者向最终用户提供相应功能及服务，当最终用户使用相应功能及服务时，我们会通过开发者应用向系统申请最终用户设备的相应权限。开发者应确保最终用户可以随时通过取消系统授权开发者应用获取相应设备权限或其他开发者应用提供的授权设置，停止我们对最终用户个人信息的收集，之后最终用户可能将无法使用基于相应个人信息而提供的相关服务或功能，或者无法达到基于相应个人信息提供的相关服务拟达到的效果，但不会影响最终用户正常使用人脸安全采集SDK的其他不基于相应个人信息即可实现的业务功能。

同时，在您接入、使用【百度人脸安全采集SDK】服务前，我们要求您在隐私政策中向用户告知我们SDK的名称、SDK提供方名称、收集个人信息类型、使用目的、隐私政策链接，并获取用户的同意或取得其他合法性基础，在提供人脸采集服务时，涉及处理敏感个人信息/向中华人民共和国境外提供个人信息，建议您单独弹框获取用户的单独同意。您可以参考如下方式提供条款内容：

1.1 三方SDK信息

- 第三方SDK名称：百度人脸安全采集SDK
- 第三方公司名称：北京百度网讯科技有限公司 **1.2 收集个人信息类型以及使用目的**

序号	功能服务	个人信息类型	信息收集类型	适用系统版本
1.1	为实现设备风险环境检测、恶意应用检测、DNS劫持检测、网络安全检测等安全检测与防护功能，以及识别是否为真实设备及真人使用人脸安全采集SDK	设备信息（包括运营商信息、WiFi状态、WiFi参数、WiFi列表、系统设置、系统属性、设备型号、操作系统、正在运行的程序列表）、日志信息（包括设备信息）	必要个人信息，SDK直接采集	仅Android
1.2	为实现设备风险环境检测、恶意应用检测、DNS劫持检测、网络安全检测等安全检测与防护功能，以及识别是否为真实设备及真人使用人脸安全采集SDK	设备信息（包括IDFA、IDFV、运营商信息、WiFi状态、WiFi参数、系统设置、系统属性、设备型号、操作系统）、 位置信息（通过IP地址解析获取粗略位置信息）、日志信息（包括设备信息和IP地址）	必要个人信息，SDK直接采集	仅iOS
1.3	为实现设备风险环境检测、恶意应用检测、DNS劫持检测、网络安全检测等安全检测与防护功能，以及识别是否为真实设备及真人使用人脸安全采集SDK	设备信息（包括oaid、AAID、ODID、网络状态、网络参数、网络列表、设备型号、操作系统） 日志信息（包括设备信息）	必要个人信息，SDK直接采集	仅Harmony OS Next
2	**为帮助开发者向最终用户提供最佳质量人脸图片采集，以进行活体校验及身份核验的功能**	**人脸图片**	**可选个人信息，SDK直接采集**	**iOS、Android以及Harmony OS Next**
3.1	为帮助开发者收集最终用户输入的姓名、身份证件号码作为待核验数据，以实现向最终用户提供权威库身份核验功能	姓名、身份证件号码	可选个人信息，SDK直接采集	iOS、Android以及Harmony OS Next
3.2	为帮助开发者使用其已保存的最终用户姓名、身份证件号码作为待核验数据，以便实现无需最终用户手动输入即可向最终用户提供权威库身份核验功能	姓名、身份证件号码	可选人格信息，开发者传输共享	iOS、Android以及Harmony OS Next
3.3	为帮助开发者通过对身份证照片进行文字识别的方式，获取最终用户的姓名、身份证件号码作为待核验数据，以便实现无需最终用户手动输入即可向最终用户提供权威库身份核验功能	身份证照片	可选个人信息，SDK直接采集	iOS、Android以及Harmony OS Next
4	**为帮助开发者向最终用户提供视频录制功能，以便作为人脸图片活体校验的二次复核数据，实现活体校验二次复核功能**	**人脸视频信息（含语音）**	**可选个人信息，SDK直接采集**	**iOS、Android**

1.3 隐私政策链接 更多个人信息收集、使用、处理的细节请参考人脸安全采集SDK隐私政策

1.4 最终用户授权同意的建议方式

身份证、身份证件号码、人脸图片、人脸视频信息（含语音）为敏感个人信息，最终用户可以随时通过**【开启/取消系统授权开发者应用获取相应设备权限或其他开发者应用提供的授权设置或手动输入】**，启动或者停止我们对上述敏感个人信息的收集。若停止，最终用户可能将无法使用基于上述敏感个人信息而提供的相关服务或功能，或者无法达到基于上述敏感个人信息提供的相关服务拟达到的效果，但不会影响最终用户正常使用人脸安全采集SDK的其他不基于上述敏感个人信息即可实现的业务功能。

特别提示开发者及最终用户注意：

1. 仅在开发者集成人脸安全采集SDK且调用我们提供的公有云服务时，序号1.1及序号1.2所述信息才会上报至我们的服务器，否则我们不会也无法获取到前述信息。
2. 序号2、序号3.1、序号3.2及序号3.3所述信息将上报至开发者所部署的配合人脸安全采集SDK使用的后端人脸及安全私有化服务的服务器地址，我们不会也无法获取到前述信息。序号4所述信息将在本地加密处理后存储在应用指定路径下，我们不会也无法获取到前述信息，仅开发者可根据存储路径获取。若开发者在调用我们提供的公有云服务过程中委托我们处理最终用户的个人信息，我们将可能会获取相关个人信息，具体的收集、使用规则请参见公有云服务相关服务条款。
3. **序号3.1、序号3.2、序号3.3及序号4所述可选个人信息，可通过SDK扩展业务功能进行开启或关闭。相关功能介绍、配置方式以及实例参考本文 3. SDK扩展业务功能介绍及对应关闭的配置方式、示例**

2. SDK所需系统权限与功能关系，以及权限申请时机

为了保证最终用户能正常使用人脸安全采集SDK相应功能及服务，我们会通过开发者应用向系统申请最终用户设备的以下系统设备权限，申请前我们会征询最终用户的同意，最终用户可以选择“允许”或“禁止”权限申请。经过最终用户的授权后我们会开启相关权限，最终用户可以随时在系统中取消授权，最终用户取消授权会导致最终用户无法使用相关的业务功能，但不会导致最终用户无法使用其他业务功能。各项功能及功能对设备权限的调用情况如下：

Android系统版本

设备权限	功能及服务	权限授权方式
读取/写入外部存储卡	用于存储人脸安全采集SDK的模型信息、配置文件、以及保存安全风险诊断信息；从相册中读取身份证照片，以便识别身份证号和姓名，以实现权威库身份核验功能	授权方式由设备系统开发方以及开发者应用决定；当最终用户同意向开发者应用授予该权限时开启
获取粗略位置	用于采集设备IP地址检验设备环境及网络风险	授权方式由设备系统开发方以及开发者移动应用决定；当最终用户同意向开发者移动应用授予该权限时开启
访问网络	用于连接网络，进行数据传输，用作活体及身份核验	授权方式由设备系统开发方以及开发者产品决定；当最终用户同意向开发者产品授予该权限时开启
获取网络状态	用于获取当前网络信息，进行安全风险诊断，并保护网络传输	授权方式由设备系统开发方以及开发者产品决定；当最终用户同意向开发者产品授予该权限时开启
打开相机/摄像头	实现人脸图片采集、身份证照片采集和人脸视频录制功能，以便进行活体及身份校验	授权方式由设备系统开发方以及开发者产品决定；当最终用户同意向开发者产品授予该权限时开启。当且仅在人脸安全采集SDK初始化时进行相关权限申请，具体时机参考本文 4.SDK初始化及各项业务功能接口合规调用时机
麦克风权限	实现人脸视频录制功能（当开启视频录制功能时，需要记录语音），以便进行活体及身份校验	授权方式由设备系统开发方以及开发者产品决定；当最终用户同意向开发者产品授予该权限时开启

iOS系统版本

设备权限	功能服务	权限授权方式
打开相机/摄像头	实现人脸图片采集、身份证照片采集和人脸视频录制功能，以便进行活体及身份校验	授权方式由设备系统开发方以及开发者应用决定；当最终用户同意向开发者应用授予该权限时开启。当且仅在人脸安全采集SDK初始化时进行相关权限申请，具体时机参考本文 4.SDK初始化及各项业务功能接口合规调用时机
麦克风权限	实现人脸视频录制功能（当开启视频录制功能时，需要记录语音），以便进行活体及身份校验	授权方式由设备系统开发方以及开发者移动应用决定；当最终用户同意向开发者移动应用授予该权限时开启
访问相册	从相册中读取身份证照片，以便识别身份证号和姓名，以实现权威库身份核验功能	授权方式由设备系统开发方以及开发者产品决定；当最终用户同意向开发者产品授予该权限时开启

注：在不同设备中，权限显示方式及关闭方式可能有所不同，具体请最终用户参考设备及系统开发方说明或指引。

HarmonyOS Next系统版本

设备权限	功能及服务	权限授权方式
打开相机/摄像头	实现人脸图片采集、身份证照片采集和人脸视频录制功能，以便进行活体及身份校验	授权方式由设备系统开发方以及开发者应用决定；当最终用户同意向开发者应用授予该权限时开启。当且仅在人脸安全采集SDK初始化时进行相关权限申请，具体时机参考本文 4.SDK初始化及各项业务功能接口合规调用时机
访问网络	用于连接网络，进行数据传输，用作活体及身份核验	授权方式由设备系统开发方以及开发者产品决定；当最终用户同意向开发者产品授予该权限时开启

在不同设备中，权限显示方式及关闭方式可能有所不同，具体请最终用户参考设备及系统开发方说明或指引。

3. SDK扩展业务功能介绍及对应关闭的配置方式、示例

拓展业务功能	介绍	配置方式及示例
身份证照片文字识别	对二代居民身份证信息进行识别，可配合人脸对比、活体检测等云端服务连接权威库完成用户身份核验。	默认关闭，开启参考以下示例： iOS：authWithLicenseFileData 、 Andriod：initAccessToken
人脸视频录制	支持对采集流程中的视频进行录制并保存至本地环境，支持成功和失败下的多种场景录制。	默认关闭，开启参考关键字为"enableFaceFailVideo"的方法

4. SDK初始化及各项业务功能接口合规调用时机

为了避免您的应用在未获取用户的同意前SDK提前处理用户的个人信息，我们提供了SDK初始化的接口，请保证您的应用获取用户同意后才能调用此接口初始化SDK。

- iOS：initWithController
- Android：init
- HarmonyOS Next：init

5. 联系我们

人脸安全采集SDK的成长离不开各方开发者及最终用户的共同努力，我们非常感谢开发者及最终用户对人脸安全采集SDK数据更新、使用反馈方面的贡献。 开发者及最终用户可以通过百度云工单反馈开发者及最终用户对人脸安全采集SDK产品和服务的建议以及在使用过程中遇到的问题，以帮助我们优化产品功能及服务，使更多用户更加便捷的使用我们的产品和服务。

人脸实名认证方案-H5端（含小程序）

方案简介

方案简介

推出背景

- 现如今，人脸识别技术被广泛应用在金融支付、用户注册、人脸登录等业务场景中。黑灰产产业也开始对这些场景产生觊觎，通过**屏幕攻击、照片、纸张、以及面具、头模**等方式进行非法攻击。随着黑产技术的进步，更是出现了通过**自动化脚本直接攻击云端API、ROM注入、视频劫持替换、批量虚拟机、病毒侵入**等新型攻击手段，使现有的人脸识别方案面临着巨大的安全挑战。
- 为提升人脸识别的安全性，保障客户的业务安全，人脸实名认证产品团队与百度安全实验室联合推出**百度H5人脸实名认证方案**，在人脸登录、注册等环境加入层层保障，为您的业务保驾护航。

功能简介 百度H5人脸实名认证方案具备**OCR身份证识别（可选）**、**活体检测**、**人脸比对**、**人脸实名认证**等多项组合能力，提供**实时炫瞳视频**、**实时动作**、**视频录制**、**图片上传**等多种活体检测方式供您选择，并对金融、交通物流等不同的行业场景进行活体检测模型的优化，更有针对性的抵挡屏幕/照片/视频/换脸/面具/3D模型的攻击。同时，百度H5实名认证方案中加入**风控能力**，针对批量虚拟机、病毒侵入等新型攻击手段进行强力有效防御。

- **人脸实名认证：**基于姓名+身份证号+实时采集的人脸图片，与权威库三要素信息进行校验比对，得出比对分数，并基于此进行业务判断是否为同一人。
- **人脸比对（1:1）：**无需传入姓名或身份证号，实时采集人脸图片，与预先通过对比图片上传API上传的指定人脸图进行1：1比对。
- **活体检测：**

> **远近活体（实时检测）**：通过屏幕实时的交互动画提示，引导用户前后移动手机设备，由近及远或由远及近地进行多距离检测，配合眨眼动作验证，实现对3D头模、合成图、翻拍等活体攻击的拦截。相比于其他活体检测方式，对高清屏翻拍、AIGC合成图有更强的抵抗效果，适合对于安全性要求较高的业务场景。
>
> **炫瞳活体检测（实时检测）：**基于屏幕颜色打光的方式，通过面部反光及**瞳孔反光**对核验人员进行活体判断。相比于行业内传统的动作活体和视频活体检测方式，通过率大大提升，使用效率更加流畅便捷，有效拦截视频、图片伪造、3D面具、合成图等黑产攻击。 百度H5实名认证方案支持**实时检测**形式，**无需拍摄上传视频**，可直接在前端完成检测流程，提升整体核验流程的流畅度及用户体验。
>
> **动作活体检测（实时检测）：**通过让用户做出指定的配合式动作，识别当前操作者是否为活体，可根据场景需要选择动作列表及动作个数，并支持随机生成动作顺序，增加活体攻击难度。动作包含：眨眼、张嘴、向左转头、向右转头、向上抬头，向下低头等，可有效抵御高清图片、3D建模、视频等攻击。
>
> **静默活体检测（实时检测）：**针对视频流/图片，通过采集人像的破绽（摩尔纹、成像畸形等）来判断目标对象是否为活体，可有效防止屏幕二次翻拍等作弊攻击，可使用单张或多张判断逻辑。
>
> **非实时活体检测（录制视频）：**拍摄人脸图片/视频上传至云端接口进行核验，支持图片、静默、语音、动作等多种活体检测形式。

- **场景化模型专项优化：**提供**金融**、**交通物流**、**泛互联网**不同行业方案供您选择，针对不同的行业场景进行活体检测模型的专项训练和优化，在满足不同行业业务安全性的前提下，提高识别准确率和通过率。
- **风控能力：**人脸实名认证API接口接受SDK端传入的设备指纹信息，对JS-SDK端进行设备风险识别，辨别是否为风险设备，返回识别结果。可有效防御黑产批量虚拟机、病毒侵入等攻击手段，降低第三方黑产攻破概率，提升业务安全性。

适用场景

- 百度H5人脸实名认证方案适用于在**H5页面**、**微信公众号**、**微信小程序**等业务场景实现用户实名认证，如果您的场景是Android/iOS系统的APP场景，可以考虑通过WebView方式实现，但更推荐采用原生SDK方式，具体参考百度APP人脸实名认

证方案。

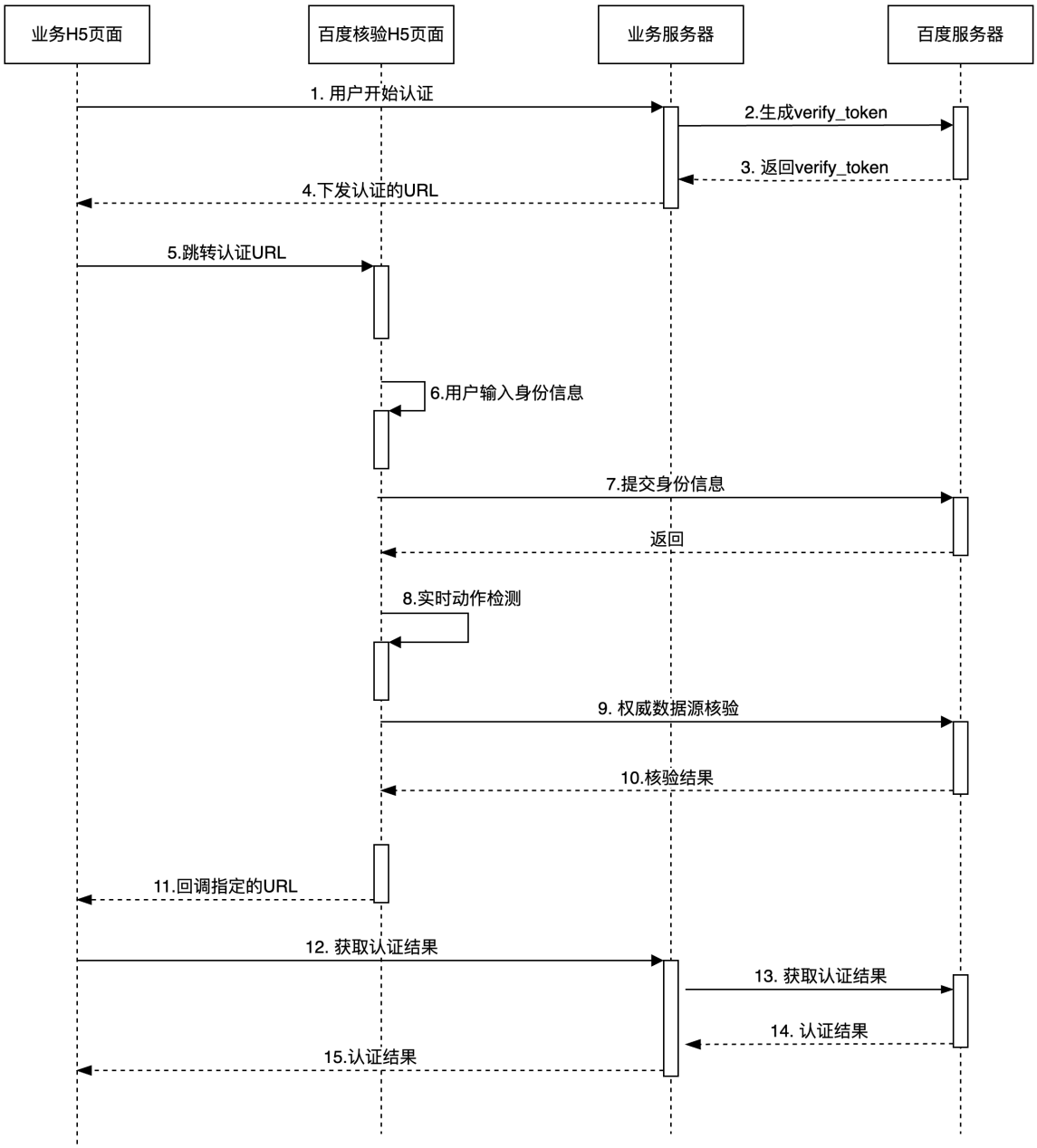
- 如您的业务场景是 **微信小程序** ，请联系专属商务客户经理或提交工单咨询接入方式。
- 实时活体检测**兼容性情况说明**：微信 版本8.0+；iOS 系统版本14.3+（微信内）、12+（微信外）；Android 系统版本9+，华为、小米、OPPO、VIVO、魅族等主流机型以及Chrome、Safari浏览器兼容性较好，个别小众安卓机型和老版本系统可能会自动降级为视频录制方案。

部署方式

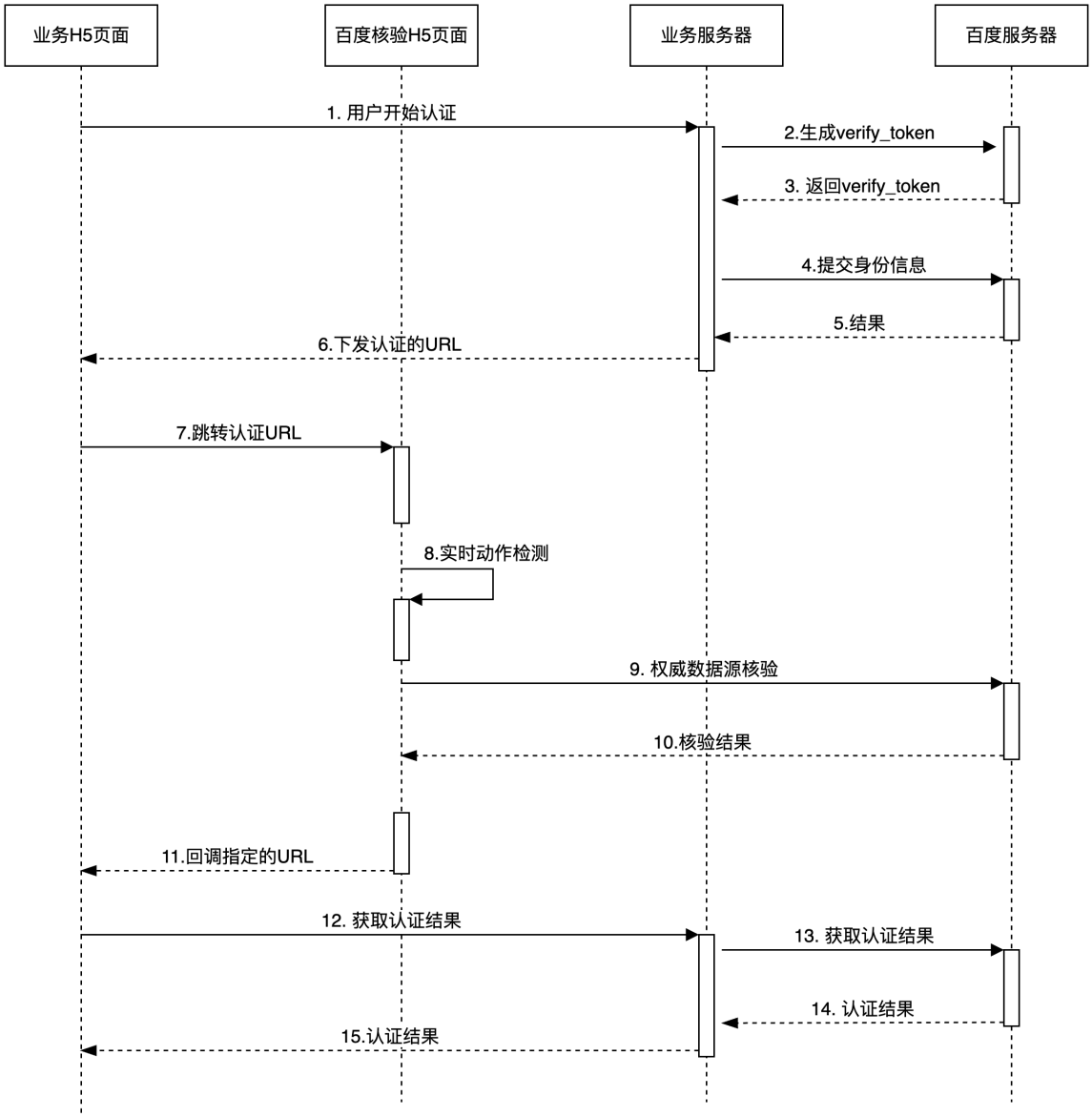
- **公有云**：通过可视化界面配置H5人脸实名认证方案，快速接入，生成含UI的链接，支持扫描二维码体验流程。创建人脸实名认证项目
- **私有化**：请联系专属商务客户经理咨询报价。

公有云方案接入时序图

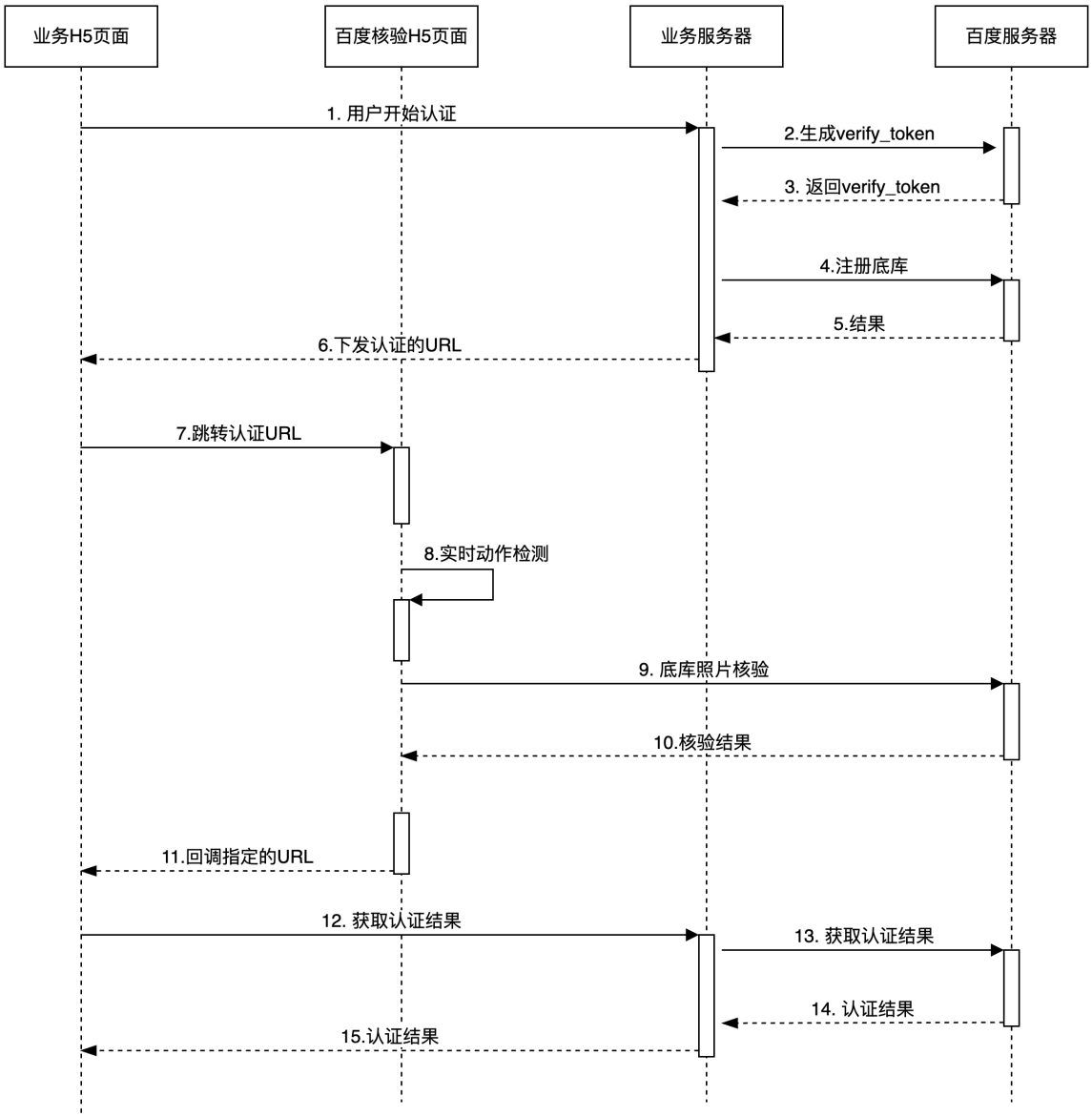
- **权威库-用户录入身份信息**：



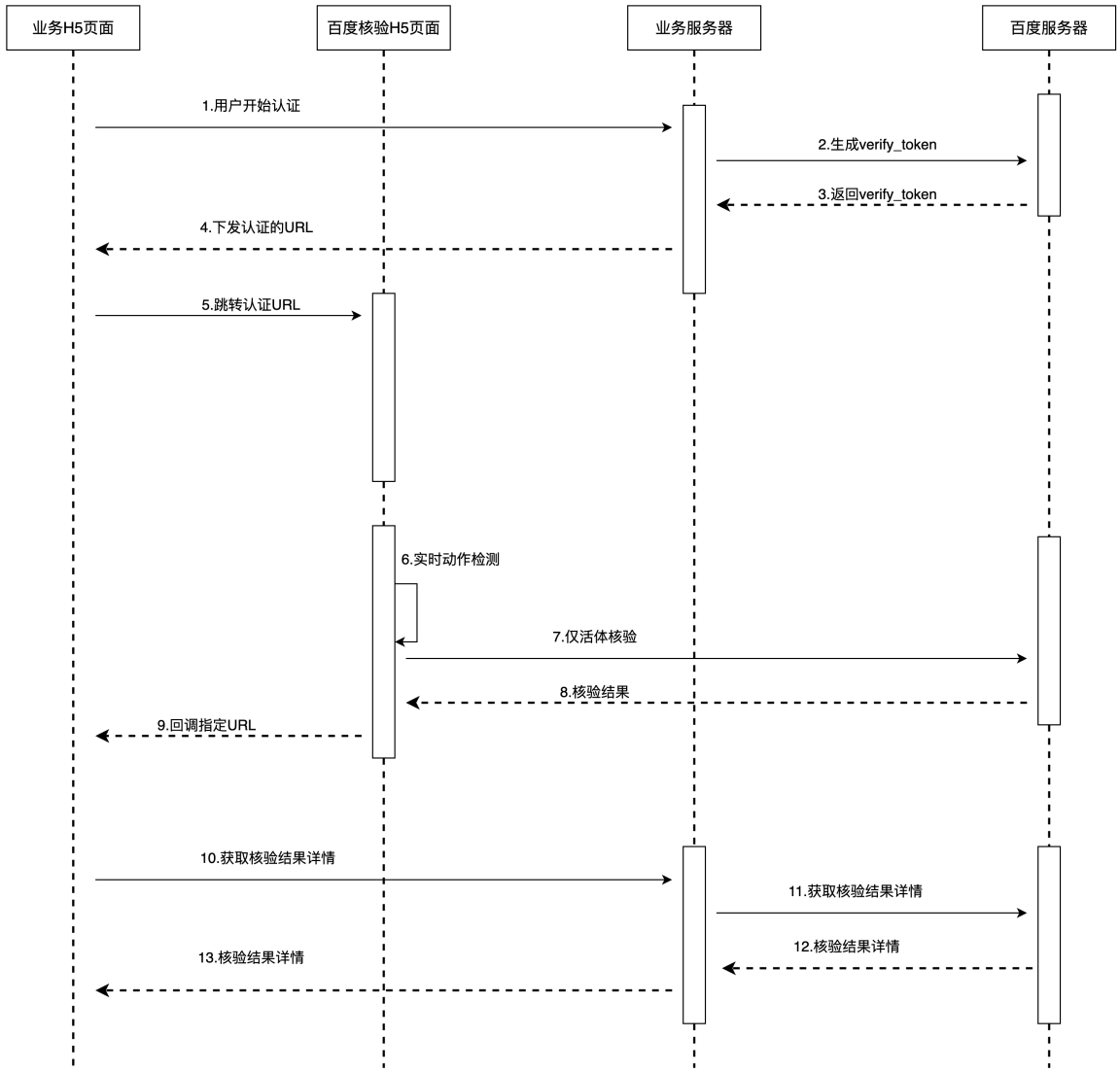
- **权威库-客户上报身份信息**：


业务H5页面
百度核验H5页面
业务服务器
百度服务器
1. 用户开始认证
2.生成verify_token
3. 返回verify_token
4.提交身份信息
5.结果
6.下发认证的URL
7.跳转认证URL
8.实时动作检测
9. 权威数据源核验
10.核验结果
11.回调指定的URL
12. 获取认证结果
13. 获取认证结果
14. 认证结果
15.认证结果

- **自建人脸库比对**：


业务H5页面
百度核验H5页面
业务服务器
百度服务器
1. 用户开始认证
2.生成verify_token
3. 返回verify_token
4.注册底库
5.结果
6.下发认证的URL
7.跳转认证URL
8.实时动作检测
9. 底库照片核验
10.核验结果
11.回调指定的URL
12. 获取认证结果
13. 获取认证结果
14. 认证结果
15.认证结果

- **仅活体检测**：



方案接入流程

- 百度H5人脸实名认证方案具体接入步骤请参考H5方案接入指南

方案接入指南

本文档将帮助您完成H5实名认证方案的创建及接入全流程。

一、 准备工作

在正式集成前，需要做一些准备工作，完成一些账号、应用及方案配置，具体如下：

Step1: 注册成为开发者

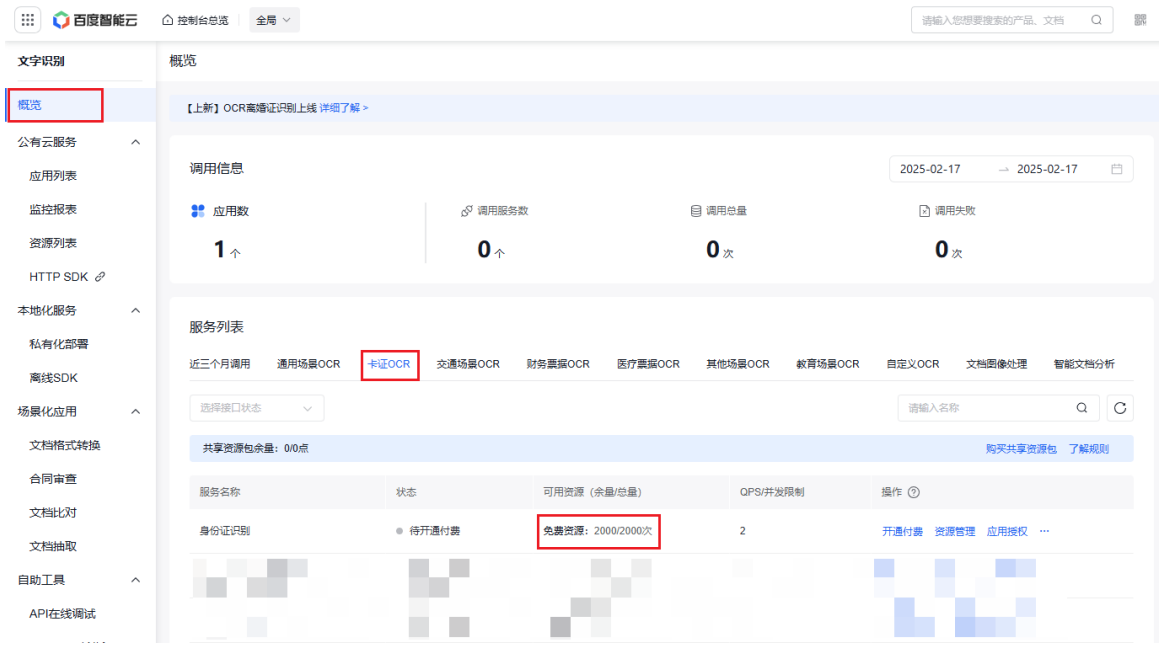
> 在使用百度人脸实名认证方案之前，首先需注册百度智能云账号，账号注册方式请参考**账号注册指南**。
>
> 百度智能云账号注册完成以后，为顺利调用百度AI能力，需完成企业认证。具体认证方式请参考**企业认证指南**。

Step2：创建应用

2.1 输入应用名称，领取免费额度

- 创建好账号后，在正式调用AI 能力之前，需首先**创建应用**，应用是调用服务的基础能力单元，目前仅支持**人脸识别**应用。
- 同时确认接口所需的**免费调用额度**已经下发（在您完成实名认证后，首次进入控制台即自动领取。领取到的资源会在10分钟内发放至账户），用于接入测试。如下图所示：

- 除人脸服务接口的免费调用额度外，还需确认**身份证识别**接口的**免费调用额度**已经下发，用来调用身份证识别功能（必须领取，否则会报错服务异常），点击此处，如下图所示：



2.2 勾选所需接口

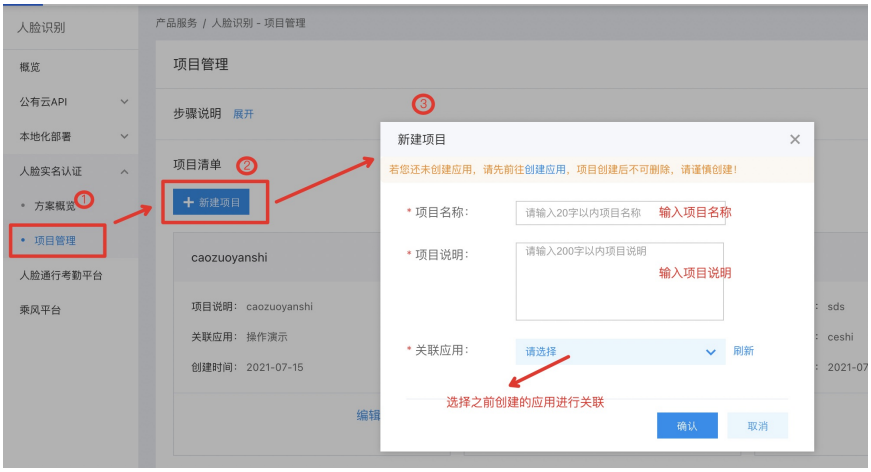
- 人脸识别服务相关接口已默认勾选。

- **注：**「接口选择」过程中，还**需勾选「文字识别」中的「身份证识别」接口**，用于实现身份核验流程中的身份证识别功能。如下图所示。

Step3：创建项目

- 进入控制台-人脸实名认证页面，选择**『项目管理』**页面，点击**『新建项目』**，进行项目创建，如下图所示。

> 创建项目前，请确保您在应用控制台已创建应用，若您未创建应用，请参考Step2创建应用后，再进行项目创建.



Step4：创建方案

- 项目创建完成后，点击「**方案管理**」进入方案管理页面，在这里您可以为您的项目创建不同的方案，如下图所示。

> 若您的场景为微信、H5页面，『场景方案』请选择H5实名认证方案。

4.1 标题设置

- **标题设置**：输入的标题文本内容，将作为您H5实名认证页面中，页面头部居中展示的标题。

例：当输入的内容为『标题名称创建演示』时，将于H5实名认证页面的头部进行展示，如下图

- **是否展示标题栏**：默认选择展示，如您的场景为APP内嵌H5页面，可能会存在APP自带标题栏与H5页面标题栏同时存在的情况，为避免“双标题”影响用户体验，您可设置不展示H5页面标题栏。

例：双标题情况如下图，可配置隐藏H5页面标题栏

4.2 授权声明配置

在这里您可以配置授权声明的信息，您也可以选择跳过此配置。

4.3 证件类型及信息录入配置

- **比对源选择**：

> **权威人脸库比对**：核验时需传入姓名及身份证号，实时采集人脸图片，与权威数据源进行一致性比对。
>
> **自建人脸库比对**：无需传入姓名或身份证号，实时采集人脸图片，与预先通过**对比图片上传API**上传的指定人脸图进行1：1比对。
>
> **仅活体检测**：无需传入姓名或身份证号，实时采集人脸图片进行真人检测，底层使用在线图片活体V4接口判断。

- **非大陆数据源**：默认不使用。配置打开后支持对中国居民二代身份证、港澳台居民往来内陆通行证、外国人永久居留证、定居国外的中国公民护照及港澳台居民居住证的验证。若不打开，则只支持中国居民二代身份证的验证。
- **身份信息录入方式**：支持身份证识别OCR、手动输入、指定用户身份核验三种。

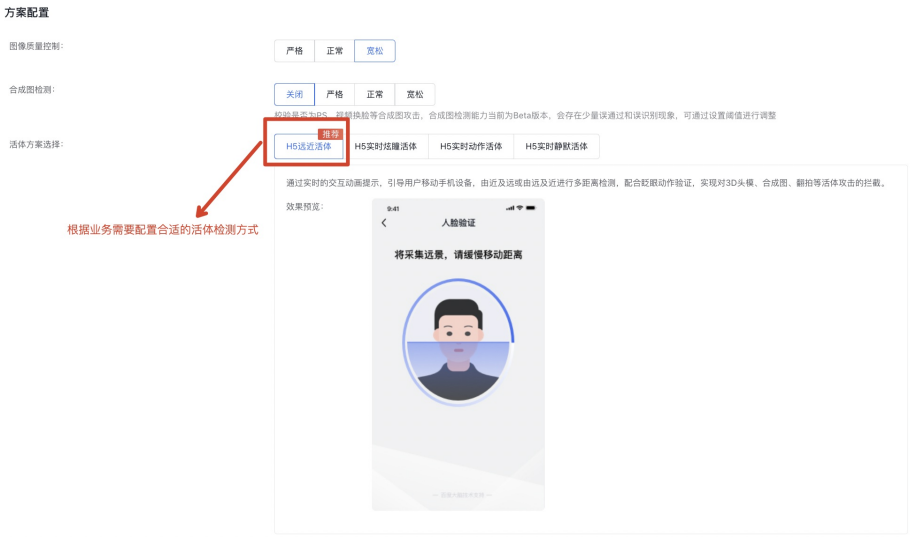
- 当选择**身份证识别OCR**后，支持配置**身份证风控功能**，开启后，对身份证的真伪进行判断，支持识别翻拍、PS伪造的身份信息，但开启后会存在少量误通过和误识别现象。同时支持配置身份证**人像面+国徽面及人像面的选择**，选择后通过查询接口可以查询到接口的返回信息。
- 当选择**手动输入**后，支持用户自己输入姓名及身份证号信息。
- 当选择**指定用户身份核验**后，支持指定用户的姓名+身份证号信息，用户侧无需输入，只需进行人脸采集及活体检测即可。 **注意：**若需要指定用户身份核验，则需要先请求 指定用户信息上报接口 再请求H5实名认证方案的URL。

4.4 方案配置

在此部分您可以根据业务属性配置适合的方案内容。

您可参考以下配置说明对图像质量、活体检测、合成图等参数进行灵活配置：

- **图像质量检测**：分为严格、正常、宽松三个等级，越严格，图片对角度、模糊度、遮挡等信息参数把控越高，推荐使用宽松。
- **活体检测**：分为严格、正常、宽松三个等级，越严格，对活体的检测等信息把控越高，推荐使用正常。
- **合成图检测**：分为严格、正常、宽松三个等级，越严格，对合成图的检测等信息把控越高，推荐使用正常。
- **活体检测方案**： 如下图所示，您可灵活选择适合自身业务的活体检测方式（如远近、炫瞳、动作等），同时H5实名认证方案采用**实时活体检测形式**，**无需用户拍摄上传视频**，直接在前端完成检测流程，提升整体核验流程的流畅度及用户体验。



- **远近活体（实时）**：通过屏幕实时的交互动画提示，引导用户前后移动手机设备，由近及远或由远及近地进行多距离检测，配合眨眼动作验证，实现对3D头模、AIGC合成图、翻拍等活体攻击的拦截。相比于其他活体检测方式，对高清屏翻拍、AIGC合成图有更强的抵抗效果，适合对于安全性要求较高的业务场景。
- **炫瞳活体检测（实时）**：基于屏幕颜色打光的方式，通过面部反光和**瞳孔反光**对核验人员进行活体判断。相比于行业内传统的动作活体和视频活体检测方式，通过率大大提升，使用效率更加流畅便捷，有效拦截视频、图片伪造、3D面具、合成图等黑产攻击。**炫瞳活体检测**一般搭配1-2个前置动作，进一步提升安全性。
- **动作活体检测（实时）**：通过用户做指定动作来验证当前拍摄视频的用户是否为活体。支持指定动作的个数进行验证，支持设置指定1-3个验证动作，推荐使用1或者2个动作进行验证。
- **静默视频活体检测（实时）**：通过用户录制一段视频来验证当前拍摄视频的用户是否为活体。
- **非实时活体检测（录制视频）：**拍摄人脸图片/视频上传至云端接口进行核验，支持图片、静默、语音、动作等多种活体检测形式。

附录：

活体检测阈值指标（高于此阈值即判断为活体）

控制度	对应阈值	说明
宽松	0.05	万分之一活体误拒率
正常（推荐）	0.3	千分之一活体误拒率
严格	0.9	百分之一活体误拒率

4.5 认证结果配置

通过上传「姓名+身份证号+人脸图片」三要素到人脸实名认证接口，通过调取权威数据源判断是否为本人操作。

- **提供的认证结果页面**：您可以选择是否使用百度提供的展示给用户认证结果的页面，若您选择自己开发，可选择不使用。
- **认证未通过时URL是否失效**：当用户认证未通过时，部分用户会出现多次重新请求验证的情况，您可通过此项配置控制用户认证数。
- **阈值**：此阈值设置的是用户上传图片与公安权威数据源图片进行比对后得分的阈值，高于此阈值即判断为用户本人。**阈值设置推荐为80**，您可通过实际业务场景继续调整。

4.6 降级活体方案配置

当您的活体方案选择的是实时炫瞳、实时动作、实时静默方案时，因实时音视频技术是在2017年提出的新型技术，对浏览器、手机系统以及APP内核存在兼容性要求，部分情况下无法进行实时检测。

您可选择不兼容时是否允许降级，并在允许降级时设置指定的活体检测方式。

- **【允许降级，优先保障用户完成认证流程】**：当选择此选项时，因浏览器、手机系统以及APP内核等环境因素导致实时活体检测不兼容时，自动降级至方案中配置的降级活体检测方式。您可事先选择降级后的活体检测方案，包括H5数字活体检测、H5动作活体检测、H5视频活体检测、在线图片检测四种降级方案。如下图所示：

- **【禁止降级，不兼容时中断流程跳转失败结果页】**：当选择此选项时，因浏览器、手机系统以及APP内核等环境因素导致实时活体检测不兼容时，当前核验流程结束，自动跳转至方案结果页。

Step5：提交方案，扫码预览 方案配置完成后，点击提交按钮，进入方案管理页面，鼠标移入「扫码体验」即可显示二维码信息，扫码即可体验H5实名认证流程，如下图所示。

> 以上体验流程仅当身份信息录入方式选择**身份证识别OCR**时生效，选择**手动输入、业务调用时传入身份信息**时则无法体验。 请谨慎修改上述H5方案配置，在点击「提交」后会实时对方案配置进行更新。若您需要修改方案，请在不影响线上业务的情况下进行调整。

二、 方案接入

Step1：获取token 通过获取verify_token接口获取verify_token信息

Step2: 确认用户信息上报方式

1. 如果选择用户手动输入或者拍照上传的方式，则跳过Step2，直接进入Step3
2. 如果选择通过API的形式传输用户信息，不需要用户手动输入，则通过指定用户信息上报接口上传对应的接口。 **Step3：跳转实名认证H5 URL，用户进行操作** 业务H5网页通过获取verify_token接口返回的verify_token信息请求认证H5页面，进行用户端流程操作。如已在获取Token时传入redirect_config，则无需配置callbackUrl、successUrl和failedUrl，**优先使用获取**

Token时传入的Url进行跳转。

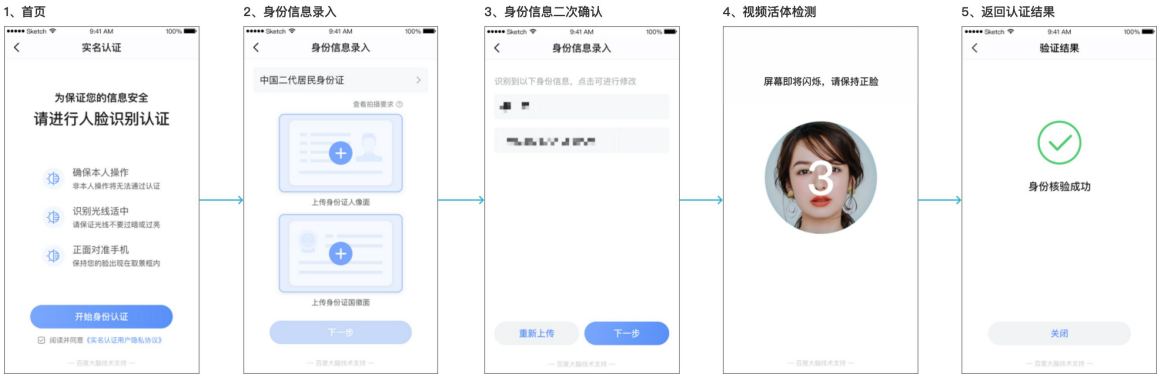
认证URL： https://brain.baidu.com/face/print/?token=xxx&successUrl=https://xxx&failedUrl=https://xxx%3CBr%3E

参数	含义	备注
token	填写verify_token，verify_token获取参考获取verify_token接口文档	
callbackUrl	通用跳转的网址，不区分成功与失败，用户通过token查询结果，callbackUrl与successUrl/failedUrl互斥（配置后successUrl、failedUrl不生效），网址需要加http/https前缀	如果流程回调需要携带token，建议传递https://www.callbackurl.com/xxxxxx?token=xxxx，核验结束后会直接回调该地址（预计2024年7月24日生效）
successUrl	请求成功跳转的网址，网址需要加http/https前缀	如果流程回调需要携带token，建议传递https://www.successurl.com/xxxxxx?token=xxxx，成功后会直接回调该地址
failedUrl	请求失败跳转的网址，网址需要加http/https前缀	如果流程回调需要携带token，建议传递https://www.failedUrl.com/xxxxxx?token=xxxx，失败后直接回调该地址

successUrl和failedUrl推荐使用encodeURIComponent进行转义，不然可能无法正确跳转，转义示例如下：

```
encodeURIComponent("https://ai.baidu.com")
```

操作流程参考如下：



Step4：获取认证结果及资料，返回用户认证信息 在用户完成认证（成功或者失败）后，我们会分别回调successUrl和failedUrl，传递?token=xxx参数，业务上可以根据token字段获取对应的数据，其中包括比对分数、图片等数据均需要通过后查询的接口进行查询。可以参考获取认证人脸接口文档、查询认证结果接口文档、查询统计结果接口文档查询用户人脸实名认证流程的相关信息，以进行后续业务集成开发。

三、 自有APP内嵌H5页面

如您需要在自有APP渠道中，通过webview方式内嵌H5页面，使用人脸实名认证H5方案。

我们为您提供了兼容性配置方法及其所需组件，以减少您可能遇到的兼容性问题，详细配置方法请参见APP内嵌H5兼容性配置文档。

如您使用该场景，为了更好地配合组件使用，请在您的认证URL后，增加参数&useNative=1，以使用js-bridge通信功能。如：https://brain.baidu.com/face/print/?token=xxx&successUrl=https://xxx&failedUrl=https://xxx%3CBr%3E&useNative=1

H5人脸实名认证方案配套接口

H5人脸实名认证方案提供多接口能力，实现以下功能：

- **对比图片上传**：上传用于1:1人脸比对的人脸底图，快速直达
- **指定用户信息上报**：上传用于权威库验证时的用户身份证号码、姓名，实现**无需用户拍照或手动输入**，直接进行活体及识别，快速直达

- **获取认证人脸**：获取认证成功最终采集的人脸信息，快速直达
- **查询认证结果**：获取身份证信息（仅在使用OCR的情况下返回）、活体检测分数、人脸相似度得分，快速直达
- **实时方案视频获取**：获取实时活体检测过程的视频/图片，快速直达
- **核验及计费信息获取**：获取对应verify_token下的所有核验信息（人脸、身份信息、认证结果），以及后端接口（人脸实名认证V4、人脸对比V4）的计费与否及计费时间， 快读直达

H5人脸实名认证方案的配套功能：

- **记录查询**：如您希望获取方案的记录信息，可开通记录查询功能进行可视化查看。
- **数据报警**：通过简单设置，即可对接口的安全风险监控，为业务保驾护航。当后台监测到接口受到攻击，或其他异常情况时，将向指定的邮箱发送报警邮件，提示及时处理问题，开通数据报警功能进行可视化查看。

一、方案功能接口

1.获取verify_token接口

本接口为H5实名认证方案的verify_token获取接口，利用所获取的verify_token进行实名认证流程的有效期为**2小时**。

在线调试 您可以在 示例代码中心 中调试该接口，可进行签名验证、查看在线调用的请求内容和返回结果、示例代码的自动生成。

调用方式

请求URL数据格式

向API服务地址使用POST发送请求，必须在URL中带上参数`access_token`，可通过后台的API Key和Secret Key生成，具体方式请参考“Access Token获取”。

> **注意**：`access_token`的有效期为30天，**切记需要每30天进行定期更换，或者每次请求都拉取新token**；

POST中Body的参数，按照下方请求参数说明选择即可。

> **提示**：如果您为百度云老用户，正在使用其他非AI的服务，可以参考百度云AKSK鉴权方式发送请求，虽然**请求方式**和**鉴权方法**和本文所介绍的不同，但请求参数和返回结果一致。

请求说明

注意事项：

- **请求体格式化**：Content-Type为`application/json`，通过`json`格式化请求体。

请求示例

HTTP方法：`POST`

请求URL： `https://aip.baidubce.com/rpc/2.0/brain/solution/faceprint/verifyToken/generate`

URL参数：

参数	值
access_token	通过API Key和Secret Key获取的access_token,参考“Access Token获取”

Header：

参数	值
Content-Type	application/json

Body中放置请求参数，参数详情如下：

请求参数：

参数	必选	类型	说明
plan_id	是	string	方案的id信息，请在人脸实名认证控制台查看创建的H5方案的方案ID信息
access_token	否	string	通过API Key和Secret Key获取的access_token
redirect_config	否	object	H5方案URL跳转链接重定向配置
+success_url	否	string	H5方案身份核验成功后跳转地址
+failed_url	否	string	H5方案身份核验失败后跳转地址

请求示例：

```
{
  "plan_id": 1,
  "redirect_config": {
     "success_url": "https://www.baidu.com",
     "failed_url": "https://www.baidu.com"
  },
  "access_token": "24.xxxxxxxxxxx"
}
```

返回参数

- 返回结果

字段	必选	类型	说明
success	是	boolean	返回请求是否成功信息。 若请求成功返回ture; 请求失败则返回false
result	是	object	请求结果
+verify_token	是	string	请求获取的verify_token
code	否	int	错误码（**仅认证失败时展示**）
message	否	string	错误信息（**仅认证失败时展示**）
log_id	是	string	本次查询接口请求的日志ID

- 返回成功示例

```
{
  "success": true,
  "result": {
     "verify_token": "Yz9rWITm4vak16PBAh5x8oG7"
  },
  "log_id": "1814798895"
}
```

- 返回失败示例

```
{
 "success": false,
 "result": null,
 "code": 283435,
 "message": "方案不存在。Plan not exist.",
 "log_id": "1864553256240296646"
}
```


2.指定用户信息上报接口 本接口用于，前端在方案中选择身份信息录入-身份信息录入方式-指定用户身份核验时，需要先调用此接口输入指定用户的姓名+身份证号信息，再请求url跳转页面。

在线调试 您可以在 示例代码中心 中调试该接口，可进行签名验证、查看在线调用的请求内容和返回结果、示例代码的自动生成。 **调用方式**

请求URL数据格式

向API服务地址使用POST发送请求，必须在URL中带上参数`access_token`，可通过后台的API Key和Secret Key生成，具体方式请参考“Access Token获取”。

> **注意**：`access_token`的有效期为30天，**切记需要每30天进行定期更换，或者每次请求都拉取新token**；

POST中Body的参数，按照下方请求参数说明选择即可。

> **提示**：如果您为百度云老用户，正在使用其他非AI的服务，可以参考百度云AKSK鉴权方式发送请求，虽然**请求方式**和**鉴权方法**和本文所介绍的不同，但请求参数和返回结果一致。

请求说明

注意事项：

- **请求体格式化**：Content-Type为`application/json`，通过`json`格式化请求体。

请求示例

HTTP方法：`POST`

请求URL：https://aip.baidubce.com/rpc/2.0/brain/solution/faceprint/idcard/submit

URL参数：

参数	值
access_token	通过API Key和Secret Key获取的access_token,参考“Access Token获取”

Header：

参数	值
Content-Type	application/json

Body中放置请求参数，参数详情如下：

参数	必选	类型	值
verify_token	是	string	通过access_token获取的verify_token
id_name	是	string	指定输入用户的姓名信息
id_no	是	string	指定输入用户的身份证件号信息
certificate_type	否	int	证件类型： 0 大陆居民二代身份证 1 港澳台居民来往内地通行证 2 外国人永久居留证 3 定居国外的中国公民护照 4 港澳台居民居住证 5 港澳居民来往内地通行证（非中国籍）

请求示例：

```
{
  "verify_token": "2sF3nE5mXOHkx2aQwWG4n5WI",
  "id_name": "张三",
  "id_no": "500***********3390",
  "certificate_type": 0          // 证件类型：0:大陆居民二代身份证,4:港澳台居民居住证
}
```

返回参数

- 返回结果

字段	必选	类型	说明
success	是	boolean	返回请求是否成功信息。 若请求成功返回ture; 请求失败则返回false
result	是	int	请求结果，返回固定结果1，可忽略
code	否	int	错误码（**仅认证失败时展示**）
message	否	string	错误信息（**仅认证失败时展示**）
log_id	是	string	本次查询接口请求的日志ID

- 返回成功示例

```
{
  "success": true,
  "result": 1,
  "log_id": "1244068892"
}
```

- 返回失败示例

```
{
"success": false,
"result": null,
"code": 283458,
"message": "当前链接已失效，请重试",
"log_id": "1864553719916441372"
}
```

3.对比图片上传接口 本接口用于，在H5方案中 比对源 选择为 与上传照片比对 时，需要先调用此接口上传待比对的自建人脸库中的指定人脸图片，再请求url跳转页面。

在线调试 您可以在 示例代码中心 中调试该接口，可进行签名验证、查看在线调用的请求内容和返回结果、示例代码的自动生成。 **调用方式**

请求URL数据格式

向API服务地址使用POST发送请求，必须在URL中带上参数`access_token`，可通过后台的API Key和Secret Key生成，具体方式请参考“Access Token获取”。

> **注意**：`access_token`的有效期为30天，**切记需要每30天进行定期更换，或者每次请求都拉取新token**；

POST中Body的参数，按照下方请求参数说明选择即可。

> **提示**：如果您为百度云老用户，正在使用其他非AI的服务，可以参考百度云AKSK鉴权方式发送请求，虽然**请求方式**和**鉴权方法**和本文所介绍的不同，但请求参数和返回结果一致。

请求说明

注意事项：

- **请求体格式化**：Content-Type为`application/json`，通过`json`格式化请求体。

请求示例

HTTP方法：`POST`

请求URL： https://aip.baidubce.com/rpc/2.0/brain/solution/faceprint/uploadMatchImage

URL参数：

参数	值
access_token	通过API Key和Secret Key获取的access_token,参考“Access Token获取”

Header：

参数	值
Content-Type	application/json

Body中放置请求参数，参数详情如下：

参数	必选	类型	值
verify_token	是	string	通过access_token获取的verify_token
image	是	string	图片base64字符串，编码后的图片大小不超过10M，图片分辨率小于1920*1080
quality_control	否	string	“NONE”、“LOW”、“NORMAL”、“HIGH”； 质量控制参数，未主动传入时默认为“NONE”
liveness_control	否	string	“NONE”、“LOW”、“NORMAL”、“HIGH”； 活体控制参数，未主动传入时默认为“NONE”

请求示例：

```
{
  "verify_token": "2sF3nE5mXOHkx2aQwWG4n5WI",
  "image": "/9j/4AAQSkZJRgABAQAAAQABAAD/",
  "quality_control" : "LOW",
  "liveness_control" : "LOW",
}
```

返回参数

- 返回结果

字段	必选	类型	说明
success	是	boolean	返回请求是否成功信息。 若请求成功返回ture; 请求失败则返回false
result	是	object	返回结果
code	否	int	错误码（**仅认证失败时展示**）
message	否	string	错误信息（**仅认证失败时展示**）
log_id	是	string	本次查询接口请求的日志ID

- 返回成功示例

```
{
 "success": true,
 "result": null,
 "log_id": "1329130892"
}
```

- 返回成功示例

```
{
 "success": false,
 "result": null,
 "code": 283400,
 "message": "服务异常，请稍后再试",
 "log_id": "1864553950812904684"
}
```

二、验证后查询接口

> 获取Token后，请先按照跳转实名认证H5 URL，用户进行操作后再查询接口，否则生成Token无法生效。

1.获取认证人脸接口

本接口返回进行人脸实名认证过程中进行认证的最终采集的人脸信息。根据Verify_token返回的结果信息会在云端保留**3天**，您可根据需要在此期间进行调取查询。

在线调试 您可以在 示例代码中心 中调试该接口，可进行签名验证、查看在线调用的请求内容和返回结果、示例代码的自动生成。

调用方式

请求URL数据格式

向API服务地址使用POST发送请求，必须在URL中带上参数`access_token`，可通过后台的API Key和Secret Key生成，具体方式请参考“Access Token获取”。

> **注意**：`access_token`的有效期为30天，**切记需要每30天进行定期更换，或者每次请求都拉取新token**；

POST中Body的参数，按照下方请求参数说明选择即可。

> **提示**：如果您为百度云老用户，正在使用其他非AI的服务，可以参考百度云AKSK鉴权方式发送请求，虽然**请求方式**和**鉴权方法**和本文所介绍的不同，但请求参数和返回结果一致。

请求说明

注意事项：

- **请求体格式化**：Content-Type为`application/json`，通过`json`格式化请求体。

请求示例

HTTP方法：`POST`

请求URL：`https://aip.baidubce.com/rpc/2.0/brain/solution/faceprint/result/simple`

URL参数：

参数	值
access_token	通过API Key和Secret Key获取的access_token,参考“Access Token获取”

Header：

参数	值
Content-Type	application/json

Body中放置请求参数，参数详情如下：

参数	值
verify_token	通过access_token获取的verify_token

请求示例：

```
 {
 "verify_token"："clupeyP51sn28XzxGVTfYqoN"
}
```

返回参数

- 返回结果

字段	必选	类型	说明
success	是	boolean	返回请求是否成功信息。 若请求成功返回ture; 请求失败则返回false
result	是	object	请求结果
+image	是	string	返回采集的用户人脸信息
code	否	int	错误码 **（仅认证失败时展示）**
message	否	string	错误信息 **（仅认证失败时展示）**
log_id	是	string	本次查询接口请求的日志ID

- 返回成功示例

```
{
  "success": true,
  "result": {
     "image":"https://brain.baidu.com/solution/faceprint/image/query?verify_token=xxxxxx"
  },
  "log_id": "1054986003"
}
```

- 返回失败示例

```
{
 "success": false,
 "result": null,
 "code": 283455,
 "message": "超出查询有效期。Result is expired.",
 "log_id": "1864554113296072563"
}
```

2.查询认证结果接口

本接口为请求返回的认证结果信息查询，包含身份证OCR识别信息、用户二次确认的身份证信息，活体检测信息、及用户对权威库图片进行比对的分数信息。（仅在认证成功时返回上述信息，认证失败返回错误码）根据Verify_token返回的结果信息会在云端保留**3天**，您可根据需要在此期间进行调取查询。

在线调试 您可以在 示例代码中心 **中调试该接口**，可进行签名验证、查看在线调用的请求内容和返回结果、示例代码的自动生成。 **调用方式**

请求URL数据格式

向API服务地址使用POST发送请求，必须在URL中带上参数`access_token`，可通过后台的API Key和Secret Key生成，具体方式请参考“Access Token获取”。

> **注意**：`access_token`的有效期为30天，**切记需要每30天进行定期更换，或者每次请求都拉取新token**；

POST中Body的参数，按照下方请求参数说明选择即可。

> **提示**：如果您为百度云老用户，正在使用其他非AI的服务，可以参考百度云AKSK鉴权方式发送请求，虽然**请求方式**和**鉴权方法**和本文所介绍的不同，但请求参数和返回结果一致。

请求说明

注意事项：

- **请求体格式化**：Content-Type为`application/json`，通过`json`格式化请求体。

请求示例

HTTP方法：`POST`

请求URL： `https://aip.baidubce.com/rpc/2.0/brain/solution/faceprint/result/detail`

URL参数：

参数	值
access_token	通过API Key和Secret Key获取的access_token,参考“Access Token获取”

Header：

参数	值
Content-Type	application/json

Body中放置请求参数，参数详情如下：

参数	值
verify_token	通过access_token获取的verify_token

请求示例：

```
 {
 "verify_token"："clupeyP51sn28XzxGVTfYqoN"
}
```

返回参数

- 返回结果

字段	必选	类型	说明
success	是	boolean	返回本次身份核验是否成功的结果。 成功返回ture; 失败则返回false
result	是	object	请求结果
+idcard_ocr_result	否	object	返回采集的身份证信息 当人脸实名认证控制台设置为使用OCR识别时返回此参数信息
++address	否	string	地址
++birthday	否	string	生日
++name	否	string	姓名
++id_card_number	否	string	身份证号
++gender	否	string	性别
++nation	否	string	民族
++expire_time	否	string	身份证失效日期
++issue_authority	否	string	身份证签发机关
++issue_time	否	string	身份证生效日期
+idcard_images	否	object	返回采集的身份证图片信息 当人脸实名认证控制台设置为使用OCR识别时返回此参数信息
++front_base64	否	string	身份证图片的正面信息
++back_base64	否	string	身份证图片的反面信息 当人脸实名认证控制台设置为使用OCR识别且为国徽面+人像面时返回此参数信息
+verify_result	是	object	认证返还信息
++liveness_score	是	float	**活体检测分数：** 活体验证通过时返回活体分数，不通过则返回0。
++score	是	float	人脸实名认证比对得分**（仅活体检测时返回为0）**
++spoofing	是	float	合成图分数 若未进行合成图检测，则返回0 若进行活体检测，则返回合成图检测分值
is_demote	否	bool	当配置为实时活体检测方案时是否降级，true为降级，false为不降级
+idcard_confirm	是	object	用户二次确认的身份证信息
++name	是	string	姓名
++idcard_number	是	string	身份证号
code	否	int	错误码 **（仅认证失败时展示）**
message	否	string	错误信息 **（仅认证失败时展示）**
log_id	是	string	本次查询接口请求的日志ID

- 返回示例

```
{
 "success": true,
 "result": {
    "verify_result": {
        "score": 93.7835,
        "liveness_score": 0.9672966,
        "spoofing": 0.0
    },
    "idcard_ocr_result": {
        "birthday": "19960216",
        "issue_authority": "胶南市公安局",
        "address": "山东省**********",
        "gender": "女",
        "nation": "汉",
        "expire_time": "20221103",
        "name": "柴*",
        "issue_time": "20121103",
        "id_card_number": "370**********5826"
    },
    "idcard_images": {
        "front_base64": "/9j/4AAQSkZJRgAB....",
        "back_base64": "/9j/4AAQSkZJRgAB...."
  },
    "idcard_confirm": {
        "idcard_number": "370**********5826",
        "name": "柴*"
    }
 },
 "log_id": "160931948204246"
}
```


- 返回失败示例


```
{
 "success": false,
 "result": null,
 "code": 283437,
 "message": "Token无效或已过期，请重新生成",
 "log_id": "1864555045677258973"
}
```


3.查询统计结果

根据Verify_token返回的结果信息会在云端保留**3天**，您可根据需要在此期间进行调取查询。

在线调试 您可以在 示例代码中心 中调试该接口，可进行签名验证、查看在线调用的请求内容和返回结果、示例代码的自动生成。

调用方式

请求URL数据格式

向API服务地址使用POST发送请求，必须在URL中带上参数`access_token`，可通过后台的API Key和Secret Key生成，具体方式请参考“Access Token获取”。

> **注意**：`access_token`的有效期为30天，**切记需要每30天进行定期更换，或者每次请求都拉取新token**；

POST中Body的参数，按照下方请求参数说明选择即可。

> **提示**：如果您为百度云老用户，正在使用其他非AI的服务，可以参考百度云AKSK鉴权方式发送请求，虽然**请求方式**和**鉴权方法**和本文所介绍的不同，但请求参数和返回结果一致。

请求说明

注意事项：

- **请求体格式化**：Content-Type为`application/json`，通过`json`格式化请求体。

请求示例

HTTP方法：`POST`

请求URL： `https://aip.baidubce.com/rpc/2.0/brain/solution/faceprint/result/stat`

URL参数：

参数	值
access_token	通过API Key和Secret Key获取的access_token,参考“Access Token获取”

Header：

参数	值
Content-Type	application/json

Body中放置请求参数，参数详情如下：

参数	值
verify_token	通过access_token获取的verify_token

请求示例：

```
 {
 "verify_token"："clupeyP51sn28XzxGVTfYqoN"
}
```

返回参数

- 返回结果

字段	必选	类型	说明
success	是	boolean	返回请求是否成功信息。 若请求成功返回ture; 请求失败则返回false
result	是	object	请求结果
+Verify_FIN	是	array	人脸实名认证接口请求的统计信息
++error_code	是	string	错误码编号 若为0则表示请求成功
++count	是	int	当前错误码的请求数量
+Sessioncode	是	array	随机校验码接口请求的统计信息
++error_code	是	string	错误码编号 若为0则表示请求成功
++count	是	int	当前错误码的请求数量
+OCR	是	array	OCR身份证识别接口请求的统计信息
++error_code	是	string	错误码编号 若为0则表示请求成功
++count	是	int	当前错误码的请求数量
+verify	是	array	人脸实名认证接口请求的统计信息
++error_code	是	string	错误码编号 若为0则表示请求成功
++count	是	int	当前错误码的请求数量
+VideoLiveness	是	array	视频活体检测接口请求的统计信息
++error_code	是	string	错误码编号 若为0则表示请求成功
++count	是	string	当前错误码的请求数量
+Livenesscolorful	是	array	炫瞳活体检测接口请求的统计信息
++error_code	是	string	错误码编号 若为0则表示请求成功
++count	是	string	当前错误码的请求数量
+VerifySec	是	array	人脸实名认证接口请求的统计信息
++error_code	是	string	错误码编号 若为0则表示请求成功
++count	是	string	当前错误码的请求数量
log_id	是	string	本次查询接口请求的日志ID

- 返回示例

```
{
  "success": true,
  "result": {
    "Verify_FIN": [
      {
        "error_code": 0,
        "count": 1
      }
    ],
     "Sessioncode": [
      {
        "error_code": 0,
        "count": 1
      }
    ],
    "OCR": [
      {
        "error_code": 0,
        "count": 2
      }
    ],
    "verify": [
      {
        "error_code": 0,
        "count": 1
      }
    ],
    "VideoLiveness": [
      {
        "error_code": 0,
        "count": 1
      }
    ]
    "Livenesscolorful": [
      {
        "error_code": 0,
        "count": 1
      }
    ]
    "VerifySec": [
      {
        "error_code": 0,
        "count": 1
      }
    ]
  },
  "log_id": "1405335905"
}
```


- 返回失败示例


```
{
 "success": false,
 "result": null,
 "code": 283437,
 "message": "Token无效或已过期，请重新生成",
 "log_id": "1864555045677258973"
}
```


4.实时方案视频获取

根据Verify_token返回的视频url会在云端保留1天，您可根据需要在此期间进行调取查询。

注意：H5方案中的视频录制功能默认不开启，调用本接口将无法获得视频数据，如您需要使用以上api，请联系商务经理或提交工单申请配置开通

调用方式

请求URL数据格式

向API服务地址使用POST发送请求，必须在URL中带上参数access_token，可通过后台的API Key和Secret Key生成，具体方式请参考“Access Token获取”。

> **注意**：access_token的有效期为30天，**切记需要每30天进行定期更换，或者每次请求都拉取新token**；

POST中Body的参数，按照下方请求参数说明选择即可。

> **提示**：如果您为百度云老用户，正在使用其他非AI的服务，可以参考百度云AKSK鉴权方式发送请求，虽然**请求方式**和**鉴权方法**和本文所介绍的不同，但请求参数和返回结果一致。

在线调试 您可以在 示例代码中心 中调试该接口，可进行签名验证、查看在线调用的请求内容和返回结果、示例代码的自动生成。

请求说明

注意事项：

- **请求体格式化**：Content-Type为application/json，通过json格式化请求体。

请求示例

HTTP方法：POST

请求URL： https://aip.baidubce.com/rpc/2.0/brain/solution/faceprint/result/media/query

URL参数：

参数	值
access_token	通过API Key和Secret Key获取的access_token,参考“Access Token获取”

Header：

参数	值
Content-Type	application/json

Body中放置请求参数，参数详情如下：

参数	值
verify_token	通过access_token获取的verify_token

请求示例：

```
 {
 "verify_token"："clupeyP51sn28XzxGVTfYqoN"
}
```

返回参数

- 返回结果

字段	必选	类型	说明
success	是	boolean	返回请求是否成功信息。 若请求成功返回ture; 请求失败则返回false
result	是	object	视频url链接
+processVideo	是	array	H5实时炫瞳活体、H5实时动作活体、H5实时静默活体方案视频链接数组，一个verify_token仅对应一个视频
+video	是	array	H5数字活体、H5动作活体、H5视频活体、在线图片活体方案视频链接数组，一个verify_token仅对应一个视频，**video和processVideo不会同时返回**
+images	是	array	图片链接数组，H5实时炫瞳活体、H5实时动作活体、H5实时静默活体方案所返回的4张人脸图片
+extInfo	是	string	扩展信息，如有则代表处理视频中的错误信息
log_id	是	string	本次查询接口请求的日志ID

- 返回示例

```
{
 "success": true,
 "result": {
     "processVideo": [
         "http://bj.bcebos.com/v1/aa.mp4"
     ],
     "video": [
         "http://bj.bcebos.com/v1/aa.mp4"
     ],
     "images": [
         "http://bj.bcebos.com/v1/aa.jpg"
     ],
     "extInfo": null
 },
 "log_id": "1534116864874049322"
}
```

- 返回失败示例

```
{
 "success": false,
 "result": null,
 "code": 283437,
 "message": "Token无效或已过期，请重新生成",
 "log_id": "1864555045677258973"
}
```

5.核验及计费信息获取

根据Verify_token返回的信息会在云端保留**3天**，您可根据需要在此期间进行调取查询。

调用方式

请求URL数据格式

向API服务地址使用POST发送请求，必须在URL中带上参数access_token，可通过后台的API Key和Secret Key生成，具体方式请参考“Access Token获取”。

> **注意**：access_token的有效期为30天，**切记需要每30天进行定期更换，或者每次请求都拉取新token**；

POST中Body的参数，按照下方请求参数说明选择即可。

> **提示**：如果您为百度云老用户，正在使用其他非AI的服务，可以参考百度云AKSK鉴权方式发送请求，虽然**请求方式**和**鉴权方法**和本文所介绍的不同，但请求参数和返回结果一致。

在线调试 您可以在 示例代码中心 中调试该接口，可进行签名验证、查看在线调用的请求内容和返回结果、示例代码的自动生成。

请求说明

注意事项：

- **请求体格式化**：Content-Type为application/json，通过json格式化请求体。

请求示例

HTTP方法：POST

请求URL： https://aip.baidubce.com/rpc/2.0/brain/solution/faceprint/result/getall

URL参数：

参数	值
access_token	通过API Key和Secret Key获取的access_token,参考“Access Token获取”

Header：

参数	值
Content-Type	application/json

Body中放置请求参数，参数详情如下：

参数	值
verify_token	通过access_token获取的verify_token

请求示例：

```
 {
 "verify_token"："clupeyP51sn28XzxGVTfYqoN"
}
```

返回参数

- 返回结果

字段	必选	类型	说明
success	是	boolean	返回请求是否成功信息。 若请求成功返回ture; 请求失败则返回false

log_id	是	string	本次查询接口请求的日志ID
result	是	object	结果信息
+verify_count	是	int	当前verify_token对应的核验次数，核验次数为几，后面就有几个对象数组 如方案配置为“认证未通过时URL失效”，则该verify_token核验次数为1； 如方案配置为“认证未通过时URL不失效”，则该verify_token对应的核验次数可能大于1（用户核验失败后多次重试）
+ocr_charge_count	是	int	身份证识别OCR接口的计费次数
+match_charge_count	是	int	人脸对比V4接口的计费次数 如方案配置为“认证未通过时URL失效”，计费次数最多为1； 如方案配置为“认证未通过时URL不失效”，则计费次数可能大于1（用户核验失败后多次重试）；
+verify_charge_count	是	int	人脸实名认证V4接口的计费次数 如方案配置为“认证未通过时URL失效”，计费次数最多为1； 如方案配置为“认证未通过时URL不失效”，则计费次数可能大于1（用户核验失败后多次重试）；
+verify_result	是	object[]	核验结果信息，verify_count的值为几，就返回几个该对象
++order	是	int	代表本次核验结果在verify_token所有核验次数中的顺序，按时间先后排序，第一次核验返回值为 1，第二次核验返回值为 2，以此类推
++is_verify_passed	是	boolean	本次核验是否通过 核验成功返回true 核验失败返回false
++code	是	string	本次核验的错误码 核验成功时返回 0 核验失败返回非0错误码
++message	是	string	本次核验的错误信息 核验成功时返回“核验成功” 核验失败时返回具体错误原因，如“身份证姓名不匹配”
++verify_time	是	string	本次核验完成的时间点，包含年月日、时分秒，如“2023-6-20 18:00:00”
++is_charged	是	boolean	本次核验是否计费 计费返回true 不计费返回false
++charge_type	是	string	计费类型：**verify**(人脸实名认证V4)、**match**(人脸对比V4)
++charge_time	是	string	本次计费的时间点，如不计费则该字段为空，包含年月日、时分秒，如“2023-6-20 18:00:00”
++is_ocr_charge	否	boolean	身份证识别OCR是否计费 计费返回true 不计费返回false
++ocr_charge_time	否	string	ocr 收费时间点，如不收费则该字段为空，包含年月日、时分秒，如“2023-6-20 18:00:00”
++ocr_count	否	int	ocr次数
++verify_detail	是	object	核验的详细信息
+++face_image	是	string	本次核验流程中采集的人脸最佳质量图下载url
			本次核验流程中请求的人脸实名认证V4（方案配置权威库比对）、人脸对比V4（方案配置

+++verify_log_id	是	string	自建人脸库比对）后端接口logid 可在记录查询平台中通过logid查询到3天内的记录。 少数核验失败情况下，实际并未发生上述俩接口的请求，则该字段为空，如：用户拒绝摄像头授权且不允许降级
+++score	是	float	人脸相似度得分，大于等于方案设置阈值为同一人；当身份证姓名不匹配或活体不通过时，该项为空
+++threshold	是	float	本次核验时，所使用的方案配置相似度阈值
+++liveness_score	是	float	活体检测得分，注：预留功能字段，当前返回为0
+++spoofing_score	否	float	方案配置使用合成图功能，将返回合成图得分，注：预留功能字段，当前返回为0
+++risk_level	否	string	安全风控等级 方案配置启用安全风控，将返回该字段 值为1或2时表示触发了安全风险，值为3或4时无风险
+++risk_tag	否	string	安全风控标签 方案配置启用安全风控，将返回该字段 当risk_level值为1或2时，返回具体风险类型，risk_level值为3或4时，为空
+++is_demote	否	boolean	方案配置为实时检测时，将返回该字段，代表本次核验是否出现降级 降级返回true，未降级返回false 注：当环境不支持实时检测，且方案配置允许降级时，将降级为录制视频上传
++idcard_confirm	是	object	用户手动输入或二次确认的身份证信息
+++name	是	string	姓名
+++idcard_number	是	string	证件号
+++idcard_type	是	string	证件类型，大陆居民二代身份证返回0
++idcard_ocr_result	否	object	OCR采集的身份证信息，当方案配置使用OCR采集证件照时返回该参数
+++address	否	string	地址
+++birthday	否	string	生日
+++name	否	string	姓名
+++idcard_number	否	string	证件号
+++idcard_type	否	string	证件类型，大陆居民二代身份证返回0
+++gender	否	string	性别
+++nation	否	string	民族
+++issue_time	否	string	身份证生效日期
+++expire_time	否	string	身份证失效日期
+++issue_authority	否	string	身份证签发机关
++idcard_images	否	object	返回采集的身份证图片信息 当人脸实名认证控制台设置为使用OCR识别时返回此参数信息

+++front_image	否	string	身份证正面图片的下载url
+++back_image	否	string	身份证背面图片的下载url

- 返回示例

```
{
  "success":true,
  "log_id":"123456",
  "result":{
     "verify_count":1,
     "ocr_charge_count":1,
     "match_charge_count":1,
     "verify_charge_count":1,
     "verify_result":[
        {
           "order":1,
           "is_verify_passed":true,
           "code":"0",
           "message":"",
           "verify_time":"2023-10-10 10:10:10",
           "is_charged":true,
           "charge_type":"verify",
           "charge_time":"2023-10-10 10:10:10",
           "is_ocr_charge":true,
           "ocr_count":2,
           "ocr_charge_time":"2023-10-10 10:10:10",
           "verify_detail":{
              "face_image":"xxx.jpg",
              "verify_log_id":"xxx",
              "score":89.9,
              "threshold":80,
              "liveness_score":80,
              "spoofing_score":80,
              "risk_level":"1",
              "risk_tag":"xxx",
              "is_demote":false
           },
           "idcard_confirm":{
              "name":"张三",
              "idcard_number":"36213219923232X",
              "idcard_type":"0",
              "idcard_ocr_result":{
                 "address":"江西省赣州市",
                 "birthday":"2020-10-10 10:10:10",
                 "name":"张三",
                 "idcard_number":"36213219923232X",
                 "idcard_type":"0",
                 "gender":"男",
                 "nation":"汉族",
                 "issue_time":"2020-10-10 10:10:10",
                 "expire_time":"2028-10-10 10:10:10",
                 "issue_authority":"江西省赣州市"
              },
              "idcard_images":{
                 "front_image":"xxx.jpg",
                 "back_image":"xxx.jpg"
              }
           }
        }
     ]
  }
}
```


- 返回失败示例

```
{
"success": false,
"result": null,
"code": 283437,
"message": "Token无效或已过期，请重新生成",
"log_id": "1864555045677258973"
}
```


三、常见问题

1. verify_token的详细说明 （1）access_token 有效期为 30 天，重复生成 access_token 的话，对之前的不影响，依旧能使用；access_token 与verify_token 是包含关系，即 verify_token 是由 access_token生成的，如果一个 verify_token 是和一个不匹配的 access_token 使用，会提示“Token无效或者已过期”

（2）verify_token 的核验有效期为 2 小时，在有效期内可以进行了核验动作，如果超过了 2 小时没有使用该 token 进行核验的话会提示“Token无效或者已过期”

（3） verify_token 核验完毕后（核验成功，或设置了认证未通过时URL失效），无法再进行核验，如果再次进行核验，会提示“Token无效或者已过期”

（4） verify_token 核验完毕后（核验成功，或设置了认证未通过时URL失效），支持对后验接口的查询，有效期均为 3 天，3 天后再次查询后验接口，会提示“Token无效或者已过期” **2. 获取认证人脸、查询认证结果、核验及计费信息获取接口持续返回错误码18，提示openapi限制，且有返回logid 答：**检查H5方案依赖的“人脸实名认证V4”、“人脸对比V4”两项付费接口，是否有免费额度，或者直接开通后付费，以消除该报错提示。开通后，再进行重试。

APP内嵌H5配置说明

APP内嵌H5兼容性配置

如您需要在自研APP中以webview的方式嵌入H5页面，请参照下文进行 iOS 及 Android 手机的兼容性适配。我们对一些需要做的配置进行了处理和封装，以framework（iOS）和Js-Bridge SDK（安卓）的形式提供，提升您的接入效率。

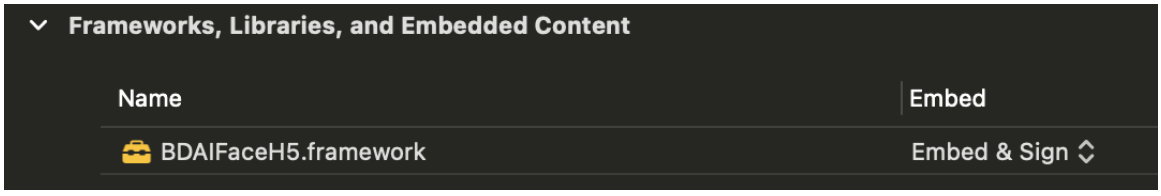
为了更好地配合组件使用，请在您的认证URL后，增加参数&useNative=1，以使用js-bridge通信功能。如：
https://brain.baidu.com/face/print/?token=xxx&successUrl=https://xxx&failedUrl=https://xxx%3CBr%3E&useNative=1

iOS

请您按如下步骤完成SDK集成，接着参照H5方案接入指南完成方案配置和token获取，生成认证H5 URL，使用生成的H5 URL替换示例代码中urlString。

1、配置依赖

- 将BDAIFaceH5.framework添加到工程中，同时配置依赖库为Embed & Sign



2、配置权限

- 在Info.plist添加权限配置，否则app运行会崩溃。

名称	是否必选	说明
Privacy - Camera Usage Description	是	相机权限
Privacy - Microphone Usage Description	是	麦克风权限

3、使用方法

- SDK结构

向集成方提供BDH5Browser类，BDH5Browser继承自UIViewController，对WKWebView进行了封装和适配，并提供了更为简洁的调用方法。

- 参数释义

方法名	返回值	说明	备注
homeUrl	无	加载H5方案主页地址	必要参数
isShowCloseButton	无	是否展示关闭按钮，默认不展示	非必要参数
js2NativeTags	无	与JS交互的约定key值，用于捕捉JS to Native的事件，通过BDH5BrowserDelegate回调自定义所需功能实现	非必要参数
bdH5BrowserDelegate	无	BDH5BrowserDelegate 代理	非必要参数

- 接口方法 通过navigationController推出browser。

```
 (void)showH5BrowserWithNavigationController:(UINavigationController *)navigationController animated:(BOOL)isAnimated;
```

通过viewController以present的形式推出browser。

```
 - (void)showH5BrowserWithPresentViewController:(UIViewController *)viewController isFullScreen:(BOOL)isFull animated:(BOOL)isAnimated completion:(void (^ __nullable)(void))completion;
```

销毁视图，completion只有在present模式下生效。

```
 - (void)dismissH5BrowserAnimated:(BOOL)isAnimated completion:(void (^ __nullable)(void))completion;
```

BDH5BrowserDelegate，用于客户自定义实现JS to Native事件回调，可根据自身业务选择实现。

```
 @protocol BDH5BrowserDelegate <NSObject>

@optional
- (void)bdH5BrowserUserContentController:(WKUserContentController *)userContentController didReceiveScriptMessage:(WKScriptMessage *)message;

@end
```

- 示例代码

在需要调用H5人脸的地方调用该示例代码即可

```
    // 百度人脸URL
    NSString *urlString = @"https://ai-face-h5-evaluation.weiyun.baidu.com/face/print/h5demo";

    if (self.urlTF.text.length > 0) {
        urlString = self.urlTF.text;
    }

    _desController = [[BDH5Browser alloc] init];

    // 设置URL，必传。否则页面无法加载
    _desController.homeUrl = urlString;

    // 设置与JS协商的key值，以实现 JS to Native 的事件回调。此处js2Native是与百度人脸JS端协商好的key值，用于JS与Native通信，切勿随意更改
    _desController.js2NativeTags = @[@"js2Native"];

    // webBrowser messageHandler 事件回调代理
    _desController.bdH5BrowserDelegate = self;

    // 是否展示关闭按钮，默认不展示
    _desController.isShowCloseButton = YES;

    // push or present
//    [_desController showH5BrowserWithNavigationController:self.navigationController animated:YES];

//    // push or present
    [_desController showH5BrowserWithPresentViewController:self isFullScreen:YES animated:YES completion:nil];
```

BDH5BrowserDelegate代理实现实例。

```
 - (void)bdH5BrowserUserContentController:(WKUserContentController *)userContentController didReceiveScriptMessage:(WKScriptMessage *)message {
    NSString *messageName = message.name;
    NSString *messageBody = message.body;

    NSLog(@"*****************:%@", message.name); // 方法名
    NSLog(@"*****************:%@", message.body); // 传递的数据

    // 可根据协商key值进行自定义native逻辑，以下仅做示例展示
    if ([messageName isEqualToString:@"js2Native"]) {

        dispatch_async(dispatch_get_main_queue(), ^{
            [self->_desController dismissH5BrowserAnimated:YES completion:nil];
            self->_desController = nil;

            UIAlertController *alert = [UIAlertController alertControllerWithTitle:messageName message:messageBody preferredStyle:UIAlertControllerStyleAlert];
            UIAlertAction *okAction = [UIAlertAction actionWithTitle:@"好的" style:UIAlertActionStyleDestructive handler:nil];
            [alert addAction:okAction];
            [self presentViewController:alert animated:YES completion:nil];
        });

    }

}
```

4、SDK及示例demo源码

- 链接: https://pan.baidu.com/s/1DsQoj94eGRLTZIEDSH258g

- 提取码: 请联系您的商务经理获取

安卓

请您先参考本文档完成下述步骤，接着参照H5方案接入指南完成方案配置和token获取，生成认证H5 URL，并将H5 URL填写在jsWebview的loadUrl方法。

1、配置依赖

- 下载jsbridge.aar，放入工程libs目录。 链接: https://pan.baidu.com/s/1w5rMLGGdXvlNDeyg2Zz7FA 提取码: 请联系您的商务经理获取
- 如果您使用androidx软件包，请下载以下版本，放入工程libs目录。 链接: https://pan.baidu.com/s/1po-gQdJvjAj47xeXt3opKw 提取码: 请联系您的商务经理获取
- 在app目录下的build.gradle文件中，配置jsbridge.aar依赖。

```
 2、repositories {
    flatDir {
        dirs 'libs'
    }
}
dependencies {
    // jsBridge-SDK.aar
    compile(name: 'lib-jsbridge-1.0-release', ext: 'aar')
}
```

2、权限

名称	是否必选	说明
android.permission.INTERNET	是	网络权限
android.permission.CAMERA	是	拍照权限
android.permission.RECORD_AUDIO	是	录音权限

3、使用方法

- JsWebview作为承载动态页面的容器，在JsBridge-SDK中用于加载web页面、相机&麦克风等相关权限申请功能、调起系统相机拍照&录像功能、返回键事件处理功能。

方法名	返回值	说明	备注
loadUrl()	无	加载h5方案主页地址	webview自带方法
checkPermisions()	boolean	相机&录音权限能力封装： 判断应用是否开启相机权限和录音权限，通过返回值判断是否已开启相关权限。 有权限则返回true，调用loadUrl()，来加载h5主页地址。 没有权限则返回false，方法内部会调用系统方法来申请权限。 Manifest.permission.CAMERA 相机权限 Manifest.permission.RECORD_AUDIO 录音权限	需要调用
setJs2NativeListener()	无	设置Js2Native的监听器，H5人脸实名认证会将核验完成事件进行回调，可以根据回调事件来做后续的逻辑处理。	
onKeyDown()	boolean	对返回键事件进行处理，用于H5页面回退场景，避免Activity销毁。 需要重写Activity的onKeyDown回调方法，并在onKeyDown方法内部调用此方法。 返回值：true，处理 返回值：false，不处理	需要调用
onPermisonResult()	boolean	相机权限申请回调：对申请权限的回调进行处理，重写系统的onRequestPermisionResult回调方法，并在onRequestPermisionResult方法内部调用此方法。 通过返回值判断用户是否开启相关权限，用户开启权限则返回true，则调用loadUrl()，来加载h5主页地址。用户未开启权限则返回false，提示申请权限失败。	需要调用
onCaptureResult()	无	系统相机拍照&相机录像能力回调：相机拍照&相机录像回调进行处理，重写系统的onActivityResult回调方法，并在onActivityResult方法内部调用此方法。无返回值。	需要调用

4、示例代码

- 在布局文件中，增加jswebview组件

```
<RelativeLayout
  android:id="@+id/js_layout"
  android:layout_width="match_parent"
  android:layout_height="match_parent">
  <com.baidu.ai.face.widget.JsWebView
     android:id="@+id/js_webview"
     android:layout_width="match_parent"
     android:layout_height="match_parent" />
</RelativeLayout>
```

- 在Activity中，导入com.baidu.ai.face.widget.JsWebView

```
import com.baidu.ai.face.widget.JsWebView
```

- 在Actvity中，找到jswebview组件，通过checkPermission方法来判断是否拥有相关权限，有权限则直接加载h5地址，无权限方法内部会申请相关权限。

```
 jsWebview = (JsWebView) findViewById(R.id.js_webview);
if (jsWebview.checkPermissions()) {
   // 加载H5页面
   jsWebview.loadUrl(mUrl);
}
```

- 在JsWebView上增加Js2NativeListener，H5人脸实名认证会将核验完成事件进行回调，可以根据回调事件来做相应的逻辑处理。

```
 // 通过JsBridge监听消息
jsWebView.setJs2NativeListener(new Js2NativeListener() {
    @Override
    public void js2Native(int code, String message) {
        Toast.makeText(jsWebView.getContext(),
                "js->Native:" + "code:" + code + ",value:" + message, Toast.LENGTH_LONG).show();
    }
});
```

- 在Activity的onRequestPermisionResult调用jswebview的onPermissionsResult方法，对权限回调进行处理，返回值为true，权限申请成功，直接加载h5地址。返回值为false，权限申请失败，申请失败需要增加自己的业务处理逻辑。

```
    @Override
    public void onRequestPermissionsResult(int requestCode, String[] permissions,
                                           int[] grantResults) {
        if (jsWebview.onPermissionsResult(requestCode, permissions, grantResults)) {
            // 加载H5页面
            jsWebview.loadUrl(mUrl);
        } else {
            // 权限申请失败，增加失败处理逻辑
        }
    }
```

- 在Activity的onKeyDown中调用jswebview的onKeyDown方法，内部封装了返回事件的处理。

```
    @Override
    public boolean onKeyDown(int keyCode, KeyEvent event) {
        return jsWebview.onKeyDown(JsActivity.this, keyCode, event);
    }
```

- 在Activity的onActivityResult中调用jswebview的onCaptureResult方法，内部封装了通过相机拍照，通过相机录像的回调处理逻辑。

```
    @Override
    protected void onActivityResult(int requestCode, int resultCode, Intent data) {
        super.onActivityResult(requestCode, resultCode, data);
        jsWebview.onCaptureResult(requestCode, resultCode, data);
    }
```

服务端接入

服务端接入-方案简介

方案简介

推出背景

- 现在，人脸识别技术被广泛应用在金融支付、用户注册、人脸登录等业务场景中。技术的进步方便用户的同时，黑灰产产业也开始对这些场景产生觊觎。并通过**屏幕攻击、照片、纸张、以及面具、头模**等方式进行非法攻击。随着黑产技术的进步，更是出现了通过**自动化脚本直接攻击云端API、ROM注入、视频劫持替换、批量虚拟机、病毒侵入**等新型攻击手段。使现有的人脸识别方案面临着巨大的安全挑战。
- 为提升人脸识别的安全性，保障客户的业务安全，便于客户在静默活体、动作活体、炫瞳活体多种活体验证方式中灵活切换，人脸实名认证产品团队与百度安全实验室联合推出**人脸实名认证APP方案**，在人脸登录、注册等环境加入层层保障，为您的业务保驾护航。

功能简介

- 人脸实名认证APP方案提供标准化的人身核验流程，具有**人脸比对、活体检测、证件识别、人脸实名认证**等多项组合能力，以及端云配合高防攻击拦截能力，可抵挡**高清屏幕、照片、视频、AI换脸、高仿真面具、3D模型**的攻击。支持**静默活体、动作活体、炫瞳活体**等活体校验方式进行人脸采集，同时，加入**安全加密能力以及风控能力**，针对脚本攻击、ROM注入、视频劫持、批量虚拟机、病毒侵入等新型攻击手段进行强力有效防御。

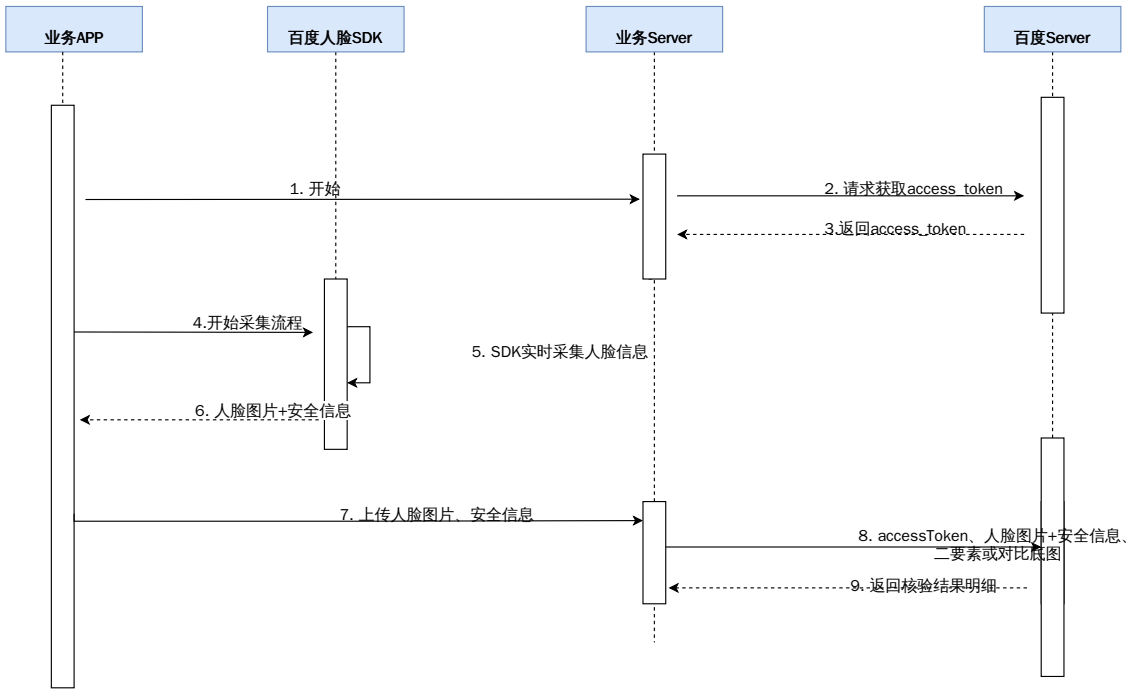
> **人脸质量检测：**判断视频流中的图片帧中，哪些图片质量较佳，即人脸图像特征清晰（满足姿态角、光照、模糊度、遮挡等校验）。
>
> **人脸图像采集：**通过本地SDK能力，采集人脸图像，同时经过人脸质量检测，确保采集到的人脸图像符合各条件校验（满足姿态角、光照、模糊度、遮挡等校验），为设备前端获取有效可分析人脸的主要功能。
>
> **炫瞳活体检测：**通过屏幕上闪烁不同颜色的光线，判断当前用户是否真人操作。通过颜色活体进行面部反光鉴别的同时，百度特加入独有的瞳孔反光识别，提升整体的攻击拒绝率指标。
>
> **动作活体检测：**通过让用户做出指定人脸配合式的交互动作，识别当前操作者是否为活体，此功能为离线使用，可设定指定动作是否使用及应用顺序。动作包含：眨眨眼、张张嘴、向左转头、向右转头、向上抬头、向下低头、上下点头、左右摇头8个。
>
> **端云互验加密：**人脸实名认证APP方案集成文件中的采集SDK会对输出的图片进行加密（支持AES/国密），在云端接口进行解密核验。此端云配合的加密方式是百度专门针对市面黑产绕过采集SDK，攻击云端接口的攻击方式进行的功能升级。
>
> **风控能力：**云接口同时接受SDK端传入的本地环境扫描设备指纹及安全信息，对SDK端进行设备风险识别，辨别是否为风险设备，返回识别结果。可有效防御黑产批量虚拟机、病毒侵入等攻击手段，降低第三方黑产攻破概率，提升业务安全性。

适用场景

- 人脸实名认证APP方案适用于**Android/IOS/HarmonyOS的APP场景**中实现用户实名认证、人脸比对、活体检测。如果您的业务场景是在微信小程序、公众号、APP内嵌H5等业务场景，推荐采用H5实名认证方案。

接入时序图

SDK-服务端接入方式是指通过SDK采集并获取加密人脸信息，经由服务端转发，配合百度云端接口实现实名认证、活体检测及人脸比对功能。如果您选择这种集成方案，端云交付的时序如下所示：



重要：AccessToken有有效期，每次请求必须重新获取。出于安全性考虑，将AK、SK写到业务Server，请求百度服务器，间接获取AccessToken。关于AccessToken鉴权，请参考https://ai.baidu.com/ai-doc/REFERENCE/Ck3dwjhhu

方案接入步骤 人脸实名认证APP方案具体接入步骤请参考

- Android-服务端接入指南
- iOS-服务端接入指南
- HarmonyOS-服务端接入指南

配套云端接口请参考：

- 权威数据源核验 人脸实名认证V4
- 本地图片无源比对 人脸对比V4
- 仅活体及深伪检测 在线图片活体V4

Android-服务端接入指南

1. 文档说明

文档名称	人脸实名认证APP方案 6.4版本集成文档
所属平台	Android
提交日期	2025-04-27

2. 版本说明

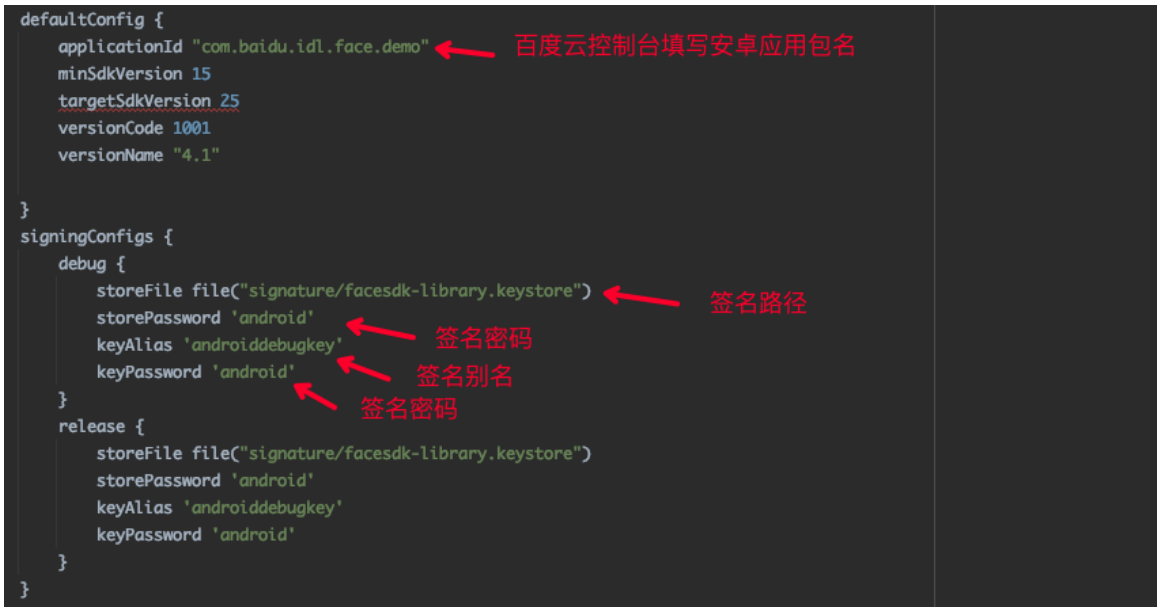
名称	版本号
名镜方案	6.4.3
系统支持	android 5.1+
架构支持	CPU架构平台，armeabi-v7a、arm64-v8a

3. SDK说明

文件名称	版本号	说明
lib-logic.aar	1.0.3	人脸实名认证SDK，业务逻辑封装
faceplatform-ui.aar	6.4	人脸SDK的UI层，封装采集和活体UI等功能，以及各平台so库
lib-liantian.aar	3.8.0.2	安全SDK
ocr_ui.aar	1.3.0	百度OCR身份识别库

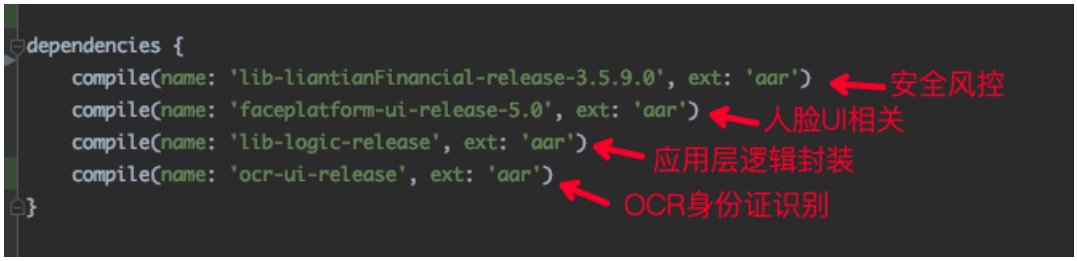
4. Demo运行

4.1 配置包名和签名 从百度云控制台下载Demo之后，需要在build.gradle中配置好包名和签名信息。



5. SDK集成

首先在app工程中增加lib-liantian.aar、faceplatform-ui.aar、lib-logic.aar、ocr-ui.aar。此处需要注意，如果需要使用OCR身份证识别能力，则需要增加此ocr-ui-release.aar，如果不使用则不需要增加。在app工程的build.gradle中添加相关依赖，然后点击运行。



6. 授权文件、加密文件

请将百度云控制台创建应用时获取的人脸授权文件(idl-license.face-android)、加密文件（idl-key.face-android）放置于Assets目录下。如果使用OCR身份证识别功能，请将OCR身份证识别授权文件（aip.license）也放置于Assets目录下，如下图所示。

7. 人脸相关接口

7.1 初始化接口

初始化接口调用

返回值	API	描述
void	init(Context context, String licenseKey, String licenseName,FaceInitCallback FaceInitCallback)	人脸初始化接口

入参说明

参数	类型	说明
context	Context	上下文
licenseKey	String	授权Key
licenseName	String	授权文件名称

onCallback回调说明

参数	类型	含义	值
resultCode	int	错误码	1000为成功，其他为失败，详情参考resultCode错误码说明
resultMsg	String	详情见resultCode错误码说明	

resultCode错误码说明

resultCode	resultMsg	自查方案
1001	License未初始化	请按照集成文档说明完成SDK初始化
1002	License数据解密失败	请检查License文件是否正确
1003	Licesen数据格式错误	请检查license文件内容有被修改过
1004	License-Key校验错误	请检查工程代码初始化参数中的licenseId，和控制台下载工程里面的licenseId是否匹配
1006	MD5校验错误	请检查工程所使用的签名文件，和控制台填写的签名信息是否匹配
1008	包名（应用名校验错误）	请检查工程代码中的applicationId（包名）和控制台填写的包名是否匹配
1009	过期时间不正确	请提交工单或者线下联系百度产研人员
1011	授权已过期	请查看当前设备时间是否已不在授权文件有效期内
1012	本地文件读取失败	请检查授权文件名称以及路径
1013	远程数据拉取失败	本地鉴权失败，1）请检查工程代码中的applicationId（包名）和控制台填写包名是否匹配；2）请检查工程所使用的签名文件，和控制台填写的签名信息是否匹配
1014	本地时间校验错误	请检查当前设备时间是否早于实际时间

7.2 人脸信息加密采集接口

包含本地质量和本地活体，本地质量可以确保采集到的人脸图像符合各条件校验（满足姿态角、光照、模糊度、遮挡等校验），本地活体分静默活体、炫瞳活体、动作活体三种。

- 此处最终采集到的数据经过加密处理，需要配合服务端的 人脸实名认证V4 或 人脸对比V4 或 在线图片活体V4 来使用。分别用来实现「权威数据源身份信息核验」、「本地图片无源比对」以及「仅活体检测」，适用于不同的业务场景需要。
- **sKey、xDeviceId、data** 为此接口的成功回调结果信息，作为上述3个服务端接口重要字段入参

返回值	API	描述
void	startFaceCollect(Context context, FaceServiceCallbck FaceServiceCallbck)	人脸采集接口

onCallback回调说明

参数	类型	含义	值
resultCode	int	错误码	0为成功，其他为失败，详情参考实名认证resultCode错误码说明
resultMap	HashMap	回调结果Map	详情见下表

resultMap key值列表说明：

Key值	类型	含义
resultMsg	String	详情见resultCode错误码说明
sKey	String	安全相关：sKey
xDeviceId	String	安全相关：xDeviceId
data	String	安全相关数据

7.3 人脸释放接口

人脸释放接口调用，实现对采集功能、模型的释放，减小内存。

返回值	API	描述
void	release()	人脸释放接口

8. OCR身份证识别相关接口

8.1 OCR身份证识别初始化接口

返回值	API	描述
void	initAccessToken(OnResultListener listener, Context context)	OCR初始化接口

入参说明

参数	类型	含义
context	Context	上下文

onError回调参数说明：

参数	类型	含义
errorCode	int	服务端返回错误码，详情见鉴权机制接口：https://ai.baidu.com/ai-doc/REFERENCE/Ck3dwjhhu
errorMessage	String	服务端返回错误信息，详情见鉴权机制接口：https://ai.baidu.com/ai-doc/REFERENCE/Ck3dwjhhu

8.2 OCR身份证识别接口

支持对二代居民身份证字段进行结构化识别，包括姓名、性别，调用参考OCR身份证识别接口文档。

返回值	API	描述
void	startOcrRecognize(Context context, OcrConfig ocrConfig, OcrRecognizeCallback ocrRecognizeCallback)	OCR识别接口

入参说明

参数	类型	含义
context	Context	上下文
ocrConfig	OcrConfig	OCR配置类

OcrConfig配置字段说明

参数	类型	含义	
ocrPageNavigationColor	int	OCR页面导航栏背景色，默认为白色	
ocrPageTitleText	String	OCR标题内容，默认为"身份信息采集"	
ocrPageTitleColor	int	OCR标题颜色，默认为黑色	
ocrPageTopText	String	OCR顶部扫描文字，默认为"请将您本人的\n身份证人像面放入框内"	
ocrPageTopTextColor	int	OCR 顶部扫描文字颜色，默认为白色	

onError回调参数说明

参数	类型	含义	值
errorCode	int	错误码	服务端返回错误码，详情见在线身份证识别接口：https://ai.baidu.com/ai-doc/OCR/rk3h7xzck
errorMessage	String	回调结果Map	服务端返回错误信息，详情见在线身份证识别接口：https://ai.baidu.com/ai-doc/OCR/rk3h7xzck

9. 代码混淆

```
 -dontwarn com.baidu.idl.**
-keep class com.baidu.idl.** { *; }
-dontwarn com.baidu.vis.**
-keep class com.baidu.vis.** { *; }
-dontwarn com.baidu.liantian.**
-keep class com.baidu.liantian.** { *; }
-dontwarn com.baidu.protect.**
-keep class com.baidu.protect.** { *; }
-dontwarn com.baidu.ocr.**
-keep class com.baidu.ocr.** { *; }
```

10. 权限

名称	说明	必选
需要动态申请的权限		
android.permission.CAMERA	拍照权限	是
android.permission.RECORD_AUDIO	录音权限（录制视频）	否
android.permission.READ_EXTERNAL_STORAGE	读取手机外部存储权限（安全相关、OCR相关）	否
android.permission.READ_EXTERNAL_STORAGE	写入手机外部存储权限（安全相关、OCR相关）	否
不需要动态申请的权限		
android.permission.INTERNET	允许访问网络	是
android.permission.ACCESS_NETWORK_STATE	获取网络状态权限	是
android.permission.READ_PHONE_STATE	允许访问电话状态权限	是
android.hardware.camera.autofocus	允许相机对焦（OCR相关）	否
android.permission.ACCESS_WIFI_STATE	获取wifi权限	是
android.permission.WAKE_LOCK	屏幕常亮权限	是

11. 不使用OCR，只使用人脸相关能力

不使用OCR，可以删除ocr-ui.aar、aip.license，以及OCR初始化以及调用相关代码。

iOS-服务端接入指南

1. 文档说明

文档名称	人脸实名认证APP方案 6.3版本集成文档
所属平台	iOS
提交日期	2023-11-09

2. 版本说明

名称	版本号
名镜方案	6.3.0
系统支持	iOS 9.0 +
架构	arm64
IDE	Xcode 14+

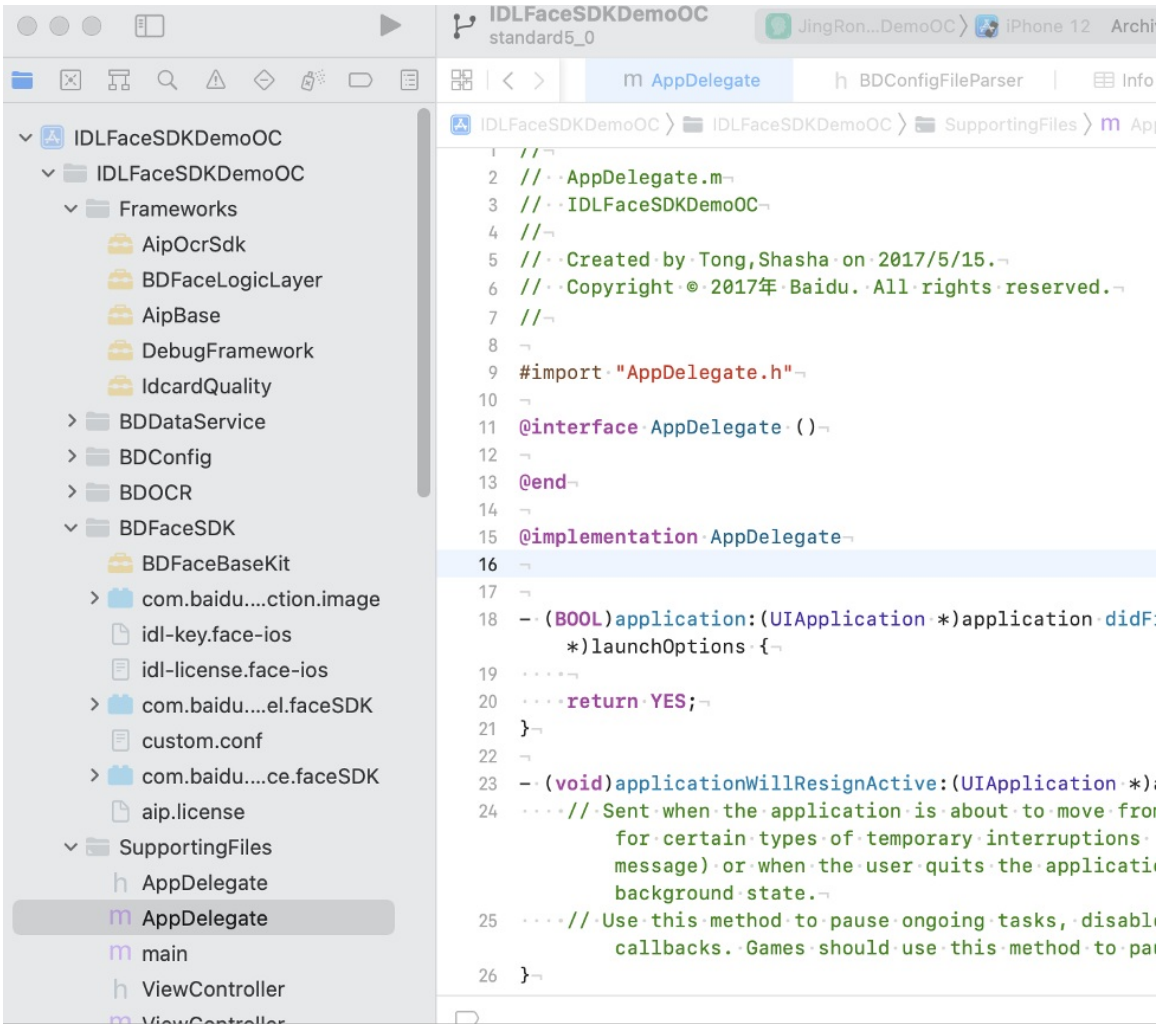
3. SDK说明

SDK	版本号	说明	类型
BDFaceBaseKit.framework	6.3.0	人脸采集SDK	静态库
AipOcrSdk.framework	1.1.0	OCR识别SDK	动态库
AipBase.framework	1.0.0	基础工具类SDK	动态库
IdcardQuality.framework	1.0.0	身份证质量控制SDK	动态库
BDFaceLogicLayer.framework	1.0.0	名镜服务SDK	静态库

4. 运行项目工程

4.1 打开下载的iOS示例工程

如下图所示：



4.2 确认bundleId等信息是否正确

连接真机进行运行，之后可以看到如下界面：

4.3 点击开始身份认证

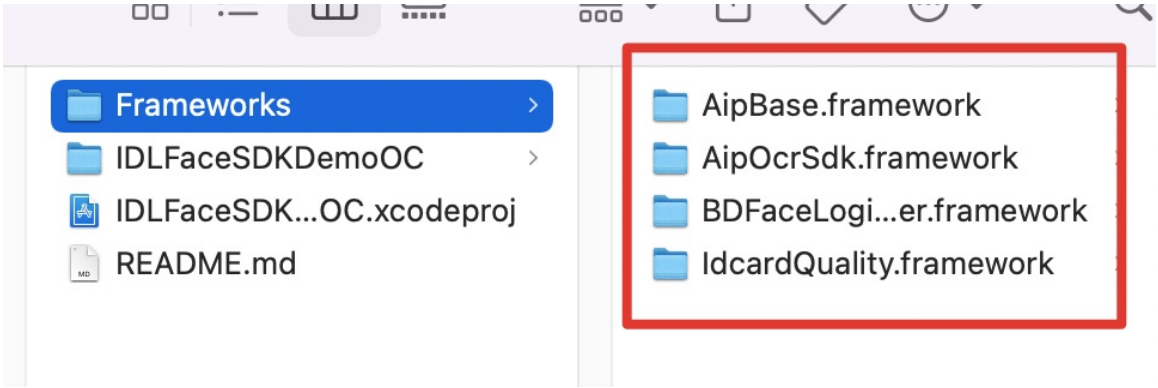
测试示例工程是否跑通，之后扫码身份证或输入身份证信息后，点击进行身份核验，验证成功可以看到如下界面：

上图为示例工程运行成功，之后可以将示例工程代码集成到目标项目中。

5. 集成步骤

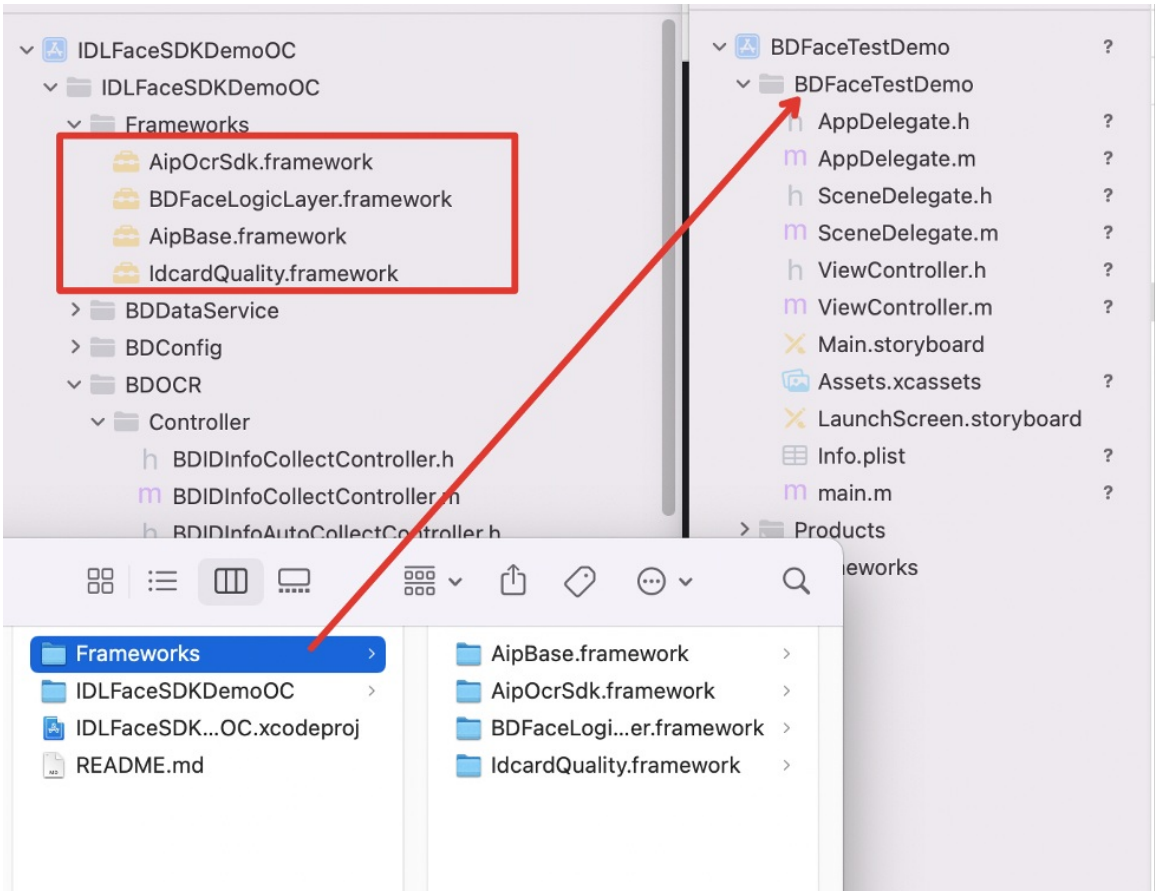
5.1 将BDFaceSDK文件夹下所有文件拖动到目标项目中

如下图所示

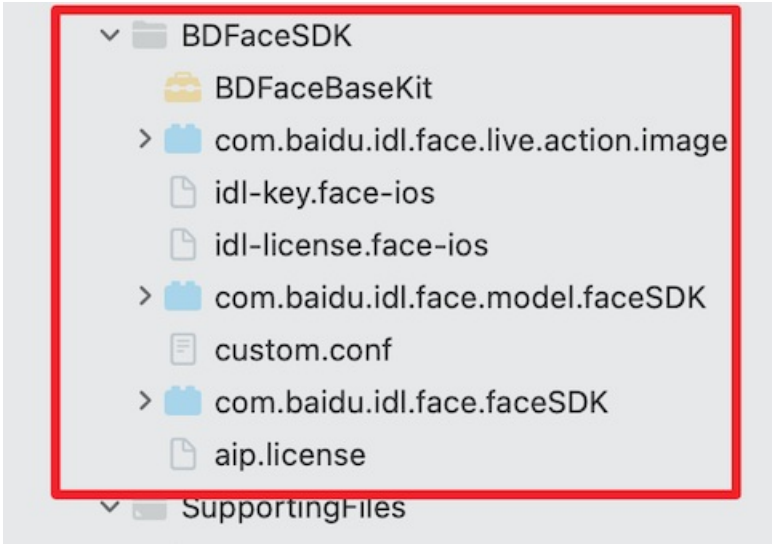


5.2 将AipBase.framework, AipOcrSdk.framework和IdcardQuality.framework DebugFramework.framework拖动到目标项目

如下所图示



5.3 将BDFaceSDK文件夹（如下图所示的文件夹）整体拖入到目标代码中；

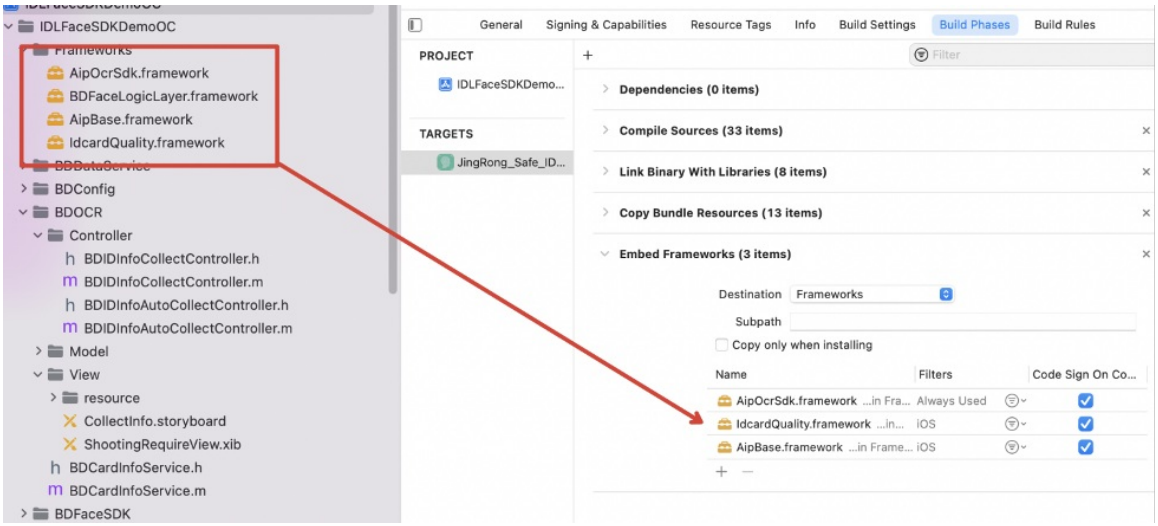


5.4 在Build Phases中点击下面的+号

选择NewCopyFilePhases,添加一个CopyFiles（可重新命名为Embed）

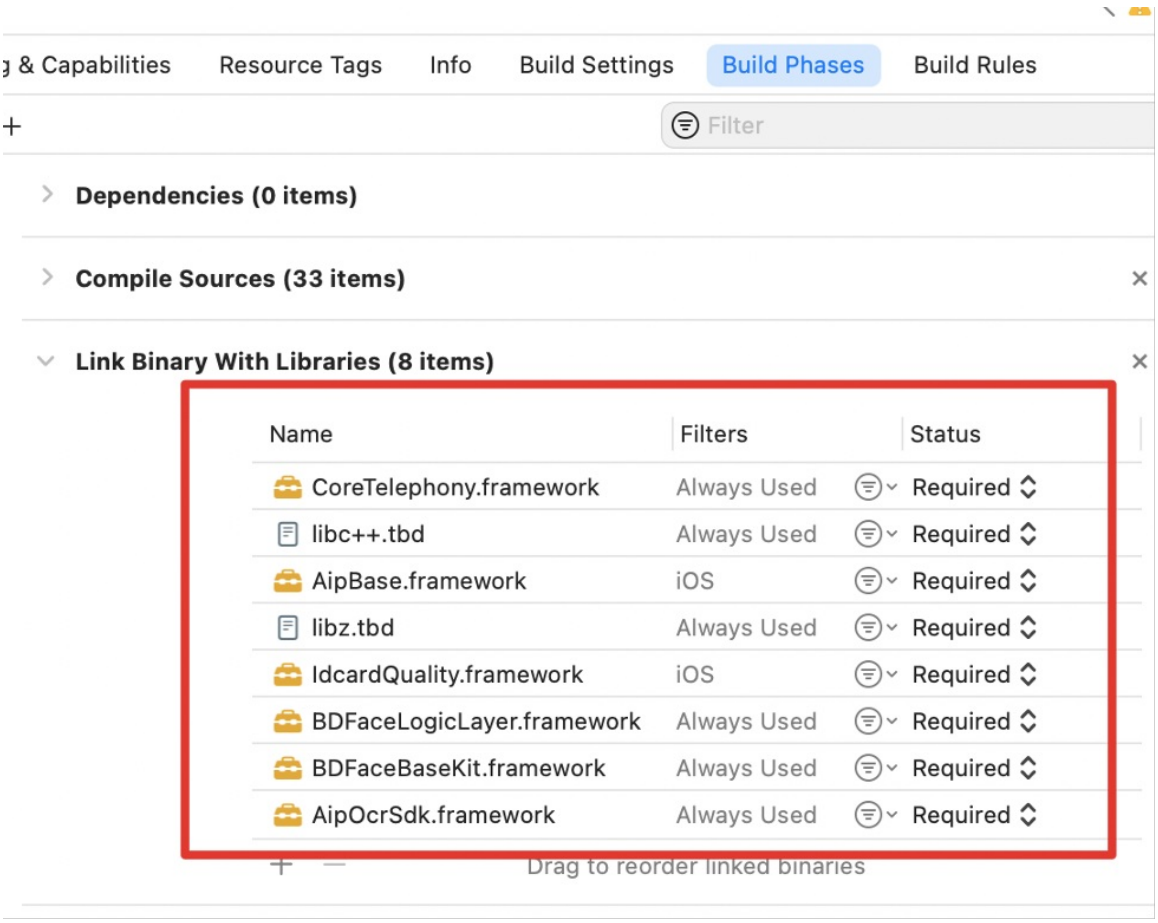
5.5 双击CopyFiles

改为Embed Frameworks，同时Destination改为framework，之后将AipBase.framework、AipOcrSdk.framework和IdcardQuality.framework 拖动到下面



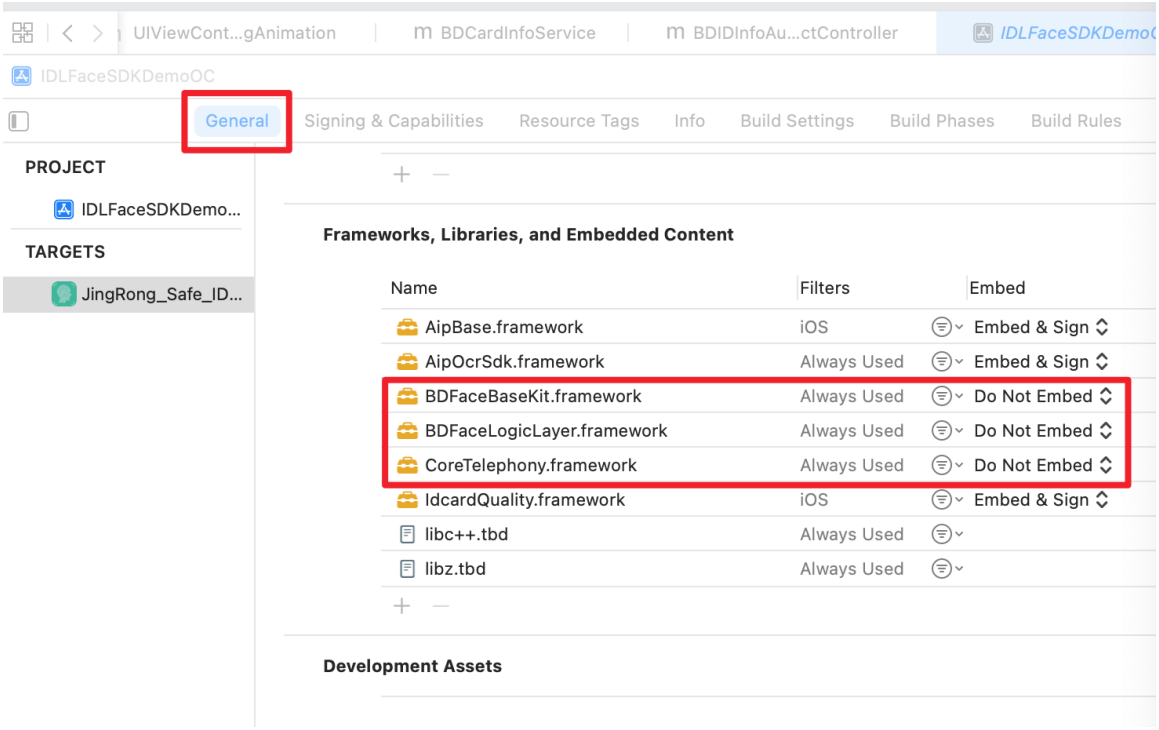
5.6 Link Binary With Libraries中添加libc++.tbd、libz.tbd 和 对应的framework

如下图所示:

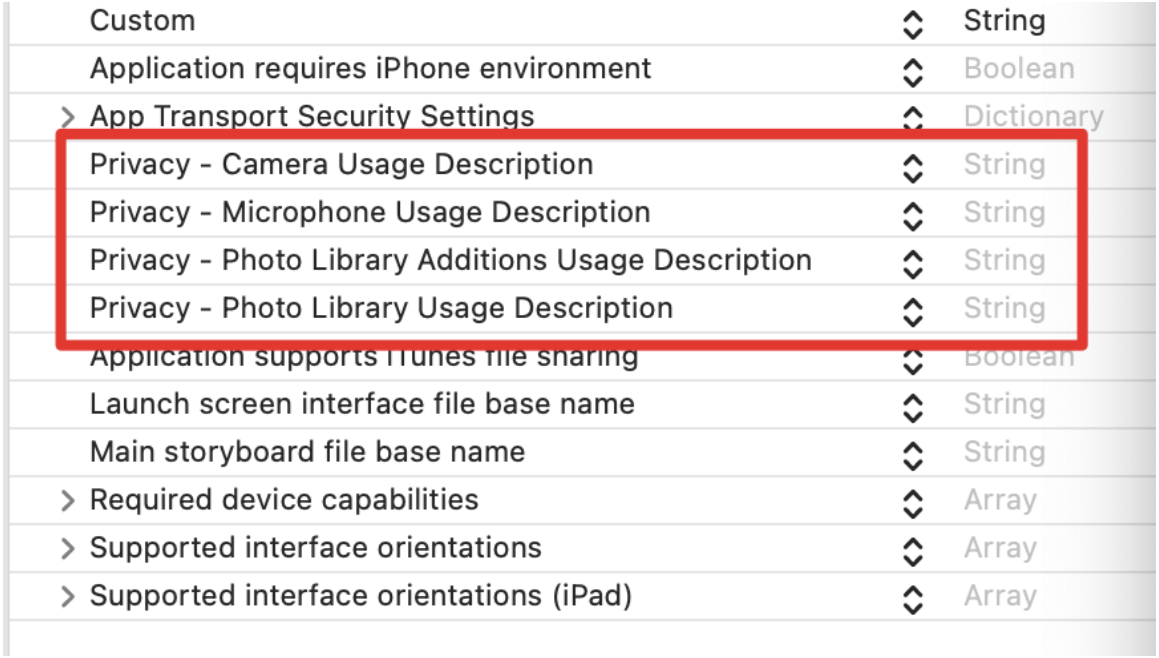


5.7 确认添加的库文件设置正确

如下图所示，点击general, 这三个库需要为Do Not Embed：BDFaceBaseKit.framework, BDFaceLogicLayer.framework，CoreTelephony.framework.



5.8 info.plist 文件中添加以下key（获取相机和照片权限）



最后之后可以将人脸核验示例工程代码代码结合自身工程项目，将相关代码集成到目标工程中。

6. 授权文件、加密文件

人脸识别授权文件（idl-license.face-ios），图片加密文件（idl-key.face-ios），OCR身份证识别授权文件（aip.license），从console平台下载完iOS项目，这些文件即包括在下载的示例项目中，不需要特殊处理，按上面第5.3步整体拖入目标项目中即可。

6.1 鉴权文件配置相关问题说明:

配置授权信息： 导入license文件后，需配置FACE_LICENSE_NAME、FACE_LICENSE_ID等信息:

```
// 人脸license文件名
#define FACE_LICENSE_NAME    @"idl-license.face-ios"
//  （您申请的应用名称(appname)+「-face-ios」后缀，如申请的应用名称(appname)为test123，则此处填写test123-face-ios)
// 在后台 -> 产品服务 -> 人脸识别 -> 客户端SDK管理查看，如果没有的话就新建一个
#define FACE_LICENSE_ID         @"vis-var-license-5.0-face-ios"
```

具体参数使用请参照示例demo。

license文件使用注意事项：如出现鉴权失败无法调用人脸采集，请检查license id与license文件导入及配置是否正常。

（1）license id需要注意-face-ios后缀是否添加。

（2）如存在多种环境配置，请确保每个环境下的license id与license文件一一对应。tip：license文件存在运行缓存，切换测试环境请清空缓存，删除安装包后重新编译安装。

（3）确保license文件路径访问正常。

（4）如出现网络鉴权失败等提示，说明license文件与FACE_LICENSE_NAME未正确设置，请才考接入demo检查工程环境，确保参数无误。

加解密key文件使用注意事项：根据2.1-2.2步骤添加加解密key文件到项目工程

（1）确保加解密key文件已重命名为idl-key.face-ios。

（2）确保idl-key.face-ios文件路径访问正常。

（3）若存在多中环境配置，请手动更新该对应文件，清空缓存后重新编译安装app包进行测试。

7. 人脸相关接口

7.1 设置默认参数（动作活体参数，人脸配置参数，UI参数）

初始化人脸和OCR SDK */

- (void)initFaceServiceAndInfoCollectService

初始化人脸SDK动作活体参数配置 */

- (void)initFaceSDK

7.2 初始化接口

初始化接口调用 用于初始化人脸识别和OCR识别

API	描述
-(instancetype)initWithController:(UIViewController * _Nonnull)controller face:(void(^)(void))initFaceBlock ocr:(void(^)(void))initOcrBlock	传入当前controller，并初始化人脸和ocrSDK

入参说明

参数	类型	说明
controller	UIViewController *	当前controller
initFaceBlock	Block对象	人脸识别初始化Block
initOcrBlock	Block对象	OCR识别初始化Block

initFaceBlock中，使用以下函数进行人脸SDK初始化 -(void)initCollectWithLicenseID:(NSString *)licenseID andLocalLicenceName:(NSString *)licenseName andExtradata:(NSDictionary *)extradata callback:(FaceSDKInitResultBlock *)block;

回调状态码说明

参数	类型	说明
BDFaceInitRemindCode	枚举	初始化人脸的状态码,1000为成功，其他为失败，详情参考如下错误码说明

具体枚举值如下

枚举值	对应数字	说明	自查建议
BDFaceLICENSE_NOT_INIT_ERROR	1001	license未初始化	请按照集成文档说明完成SDK初始化
BDFaceLICENSE_DECRYPT_ERROR	1002	license数据解密失败	请检查License文件是否正确
BDFaceLICENSE_INFO_FORMAT_ERROR	1003	license数据格式错误	请检查license文件内容有被修改过
BDFaceLICENSE_KEY_CHECK_ERROR	1004	license-key(api-key)校验错误	请检查工程代码初始化参数中的licenseId，和申请license文件的licenseId是否匹配
BDFaceLICENSE_ALGORITHM_CHECK_ERROR	1005	算法ID校验错误	请提交工单或者线下联系百度产研人员
BDFaceLICENSE_MD5_CHECK_ERROR	1006	MD5校验错误	请检查工程所使用的签名文件，和申请license文件的签名信息是否匹配
BDFaceLICENSE_DEVICE_ID_CHECK_ERROR	1007	设备ID校验错误	采集SDK的授权模式不会出现这个错误码
BDFaceLICENSE_PACKAGE_NAME_CHECK_ERROR	1008	包名(应用名)校验错误	请检查工程代码中的applicationId（包名）和申请license文件的applicationId（包名）是否匹配
BDFaceLICENSE_EXPIRED_TIME_CHECK_ERROR	1009	授权过期时间有问题	请提交工单或者线下联系百度产研人员
BDFaceLICENSE_FUNCTION_CHECK_ERROR	1010	功能未授权	请查看授权文件中是否缺少必要的采集SDK功能声明（funclist参数），例如炫瞳活体
BDFaceLICENSE_TIME_EXPIRED	1011	授权已过期	请查看当前设备时间是否已不在授权文件有效期内
BDFaceLICENSE_LOCAL_FILE_ERROR	1012	本地授权文件读取失败	请检查授权文件名称以及路径
BDFaceLICENSE_REMOTE_DATA_ERROR	1013	远程授权文件拉取失败	本地鉴权失败之后，会远程拉取授权文件；若远程鉴权依然失败，可以关闭网络后重试
BDFaceLICENSE_LOCAL_TIME_ERROR	1014	本地时间校验错误	请检查当前设备时间是否早于实际时间

7.3 人脸信息加密采集接口

- 包含本地质量和本地活体，本地质量可以确保采集到的人脸图像符合各条件校验（满足姿态角、光照、模糊度、遮挡等校验），本地活体分静默活体、炫瞳活体、动作活体三种。
- 此处最终采集到的数据经过加密处理，需要配合服务端的 人脸实名认证V4 或 人脸对比V4 或 在线图片活体V4 来使用。分别用来实现「权威数据源身份信息核验」、「本地图片无源比对」以及「仅活体检测」，适用于不同的业务场景需要。
- **sKey、xDeviceId、data** 为此接口的成功回调结果信息，作为上述3个服务端接口重要字段入参

返回值	API	描述
-(void)startFaceCollect:(void(^)(int resultCode, NSDictionary *resultDic))callBack;	人脸采集接口	

回调说明

参数	类型	含义	值
resultCode	int	错误码	0为成功，其他为失败，详情参考实名认证code错误码说明
resultDic	NSDictionary	回调结果	详情见下表

resultDic key值列表说明：

Key值	类型	含义
resultMsg	String	详情见code错误码说明
sKey	String	安全相关：sKey
xDeviceId	String	安全相关：xDeviceId
data	String	安全相关数据

8. OCR身份证识别相关接口

8.1 OCR身份证识别初始化接口

API	描述
-(void) authWithLicenseFileData: (NSData *)licenseFileContent;	将aip.license文件的NSData数据传入OCR SDK进行初始化和鉴权

入参说明

参数	类型	含义
licenseFileContent	NSData	将aip.license文件读出来，转发NSData类型

注：如果aip.license文件鉴权文件正确，那么可以使用OCR识别成功识别身份证信息。

8.2 OCR身份证识别接口 支持对二代居民身份证字段进行结构化识别，包括姓名、性别，调用参考OCR身份证识别接口文档。

返回值	API	描述
UIViewController	-(UIViewController *)startOcrRecognize:(BDAipOCRConfig *)config didGetImage:(void(^)(UIImage *image))getImageAction success:(void(^)(NSDictionary *result))success failure:(void (^)(NSError *error))failure;	OCR识别接口

参数值列表如下：

参数	类型	说明	必选
config	BDAipOCRConfig	用于设置OCR页面相关UI	否
getImageAction	Block	获取到的图片信息回调	否
success	Block	获取身份信息成功回调	否
failure	Block	获取身份信息失败回调	否

config配置说明

属性	类型	说明
titleViewBgColor	UIColor	OCR页面导航栏背景色，默认为白色
pageTitle	NSString	OCR标题内容，默认为"身份信息采集"
pageTitleTextColor	UIColor	OCR标题颜色，默认为黑色
topTipText	NSString	OCR顶部扫描文字，默认为"请将您本人的\n身份证人像面放入框内"
topTipTextColor	UIColor	OCR 顶部文字颜色，默认为白色

9. 权限

名称	用途
Privacy - Camera Usage Description	获取相机权限，需要使用相机做人脸识别和OCR识别
Privacy - Photo Library Additions Usage Description	iOS 11及以后保存照片到相册权限,OCR识别需要从相册选择照片
Privacy - Photo Library Usage Description	使用相册权限，OCR识别需要从相册选择照片
Privacy - Microphone Usage Description	人脸识别视频录制功能会使用该权限

10. 只使用人脸采集功能，不使用OCR

10.1 运行示例工程代码

点击开始身份核验，走完所有流程，可以出现如下界面：

10.2 删除库文件：AipOcrSdk.framework，AipBase.framework，IdcardQuality.framework

10.3 全局搜索与AipOcrSDK/AipOcrSdk.h

找到后并删除项目中所有引入对应SDK的代码，也可以注释，在删除后可能导致编译不通过，可以根据报错部分保证正常功能不受影响的同时进行相应的代码注释。

10.4 全局搜索 -(void) startOCRSdk 并注释该函数

注意：其他编译报错部分调用的地方也需要注释或删除

```
// 打开相机扫描
 - (void)startOCRSdk {
[self configCallback];
// 身份证识别
[AipCaptureCardVC clearIdCard];
BDAipOCRConfig *config = [[BDAipOCRConfig alloc] init];
AipNavigationController *detectNavi = [AipCaptureCardVC viewControllerToDetectIdCard:config
                                    handler:^(UIImage *image) {
    _idCardImage = image;
    [[AipOcrService shardService] detectIdCardFrontFromImage:image
                                        withOptions:nil
                                      successHandler:^(id result) {
        _successHandler(result);
    } failHandler:_failHandler];
}];
UIViewController *detectRoot = [[UIViewController alloc] init];
detectRoot.view.backgroundColor = [UIColor whiteColor];
detectRoot.view.clipsToBounds = YES;
[detectRoot addChildViewController:detectNavi];
[detectRoot.view addSubview:detectNavi.view];

[self.navigationController pushViewController:detectRoot animated:YES];

_vc = detectRoot;
__weak typeof(detectRoot) weakDetectRoot = detectRoot;
detectNavi.captureController.backAction = ^{
    [weakDetectRoot.navigationController popViewControllerAnimated:YES];
};}
```

10.5 .编译运行代码首页并将身份证OCR功能关闭或者使用代码

```
// 身份证获取方式  1 OCR扫描   0  手动输入  2代码传
+ (int)useOCR {
BDConfigDataService *service = [self sharedInstance];
NSNumber *value = service.sharedDic[BDConfigDataServiceKeyForSettingOcr];
if (value) {
    return value.intValue;
}
return [[BDConfigFileParser sharedInstance] useOCR];}
```

10.6 将userOCR的值设置为0

保证程序只能手动输入身份信息来调用人脸，此部分功能模块不使用也可以对应删除。

运行工程

HarmonyOS-服务端接入指南

前言

> 【本文】方案集成指南适用于需要集成APP方案，实现**权威库核验**、**自传照片人脸比对**、**仅活体检测**这3个业务需求时来参考。百度人脸实名认证APP方案默认开启风控功能，即使用人脸识别V4系列API接口接受SDK端传入本地环境扫描的设备指纹及安全信息，对SDK端进行设备风险识别，辨别是否为风险设备、并返回识别结果。**此方案可有效防御黑产批量虚拟机、病毒侵入等攻击手段，降低第三方黑产攻破概率，提升业务安全性。**

快速定位

1. 文档说明

文档名称	人脸实名认证APP方案 6.4版本集成文档
所属平台	HarmonyOS
提交日期	2024-09-03

2. 版本说明

名称	版本号
名镜方案	6.4.0
系统支持	HarmonyOS NEXT.0.0.36

3. SDK说明

文件名称	版本号	说明
lib_Enhance.har	人脸实名认证SDK业务逻辑封装	包含人脸采集、安全风控等功能

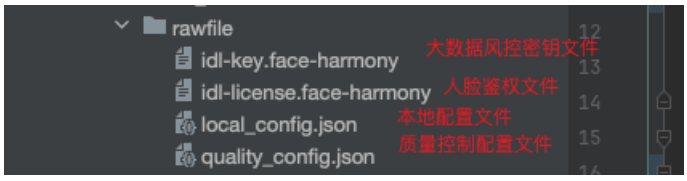
4. 配置包名和签名

- 从百度云控制台下载鸿蒙Demo之后，需要在build-profile.json5中配置好宿主应用的签名信息，更新宿主应用的所有签名信息material

```
],
"signingConfigs": [
  {
    "name": "default",
    "type": "HarmonyOS",
    "material": {
      "storePassword":"替换为签名密码",
      "certpath":"替换cert文件路径",
      "keyAlias":"替换为签名别名",
      "keyPassword":"替换为签名密码",
      "profile":"替换profile文件路径",
      "signAlg":"SHA256withECDSA",
      "storeFile":"替换为签名文件路径"
    }
  }
]
```

5. SDK集成与授权

- 首先在app工程中增加**3. SDK说明**中的依赖库资源
- 其次将百度云控制台创建应用时获取的人脸授权文件(**idl-license.face-harmony**)、大数据风控密钥文件(**idl-key.face-android**)、本地配置文件(**local_config.json**)、质量控制配置文件(**quality_config.json**)放置于rawfile目录下。



- 最后参考控制台下载的工程以及相关接口说明完成SDK集成。

6. 人脸相关接口

6.1 人脸初始化接口（SDK）

初始化接口调用

返回值	API	描述
void	init(context: Context, licenseKey: string, licenseName: string, keyName: string, faceInitCallback: FaceInitCallback)	人脸初始化接口

入参说明

参数	类型	说明
context	Context	上下文
licenseKey	String	授权Key
licenseName	String	授权文件名称
keyName	String	加解密文件名称

onCallback回调说明

参数	类型	含义	值
resultCode	int	错误码	1000为成功，其他为失败，详情参考resultCode错误码说明
resultMsg	String	详情见resultCode错误码说明	

resultCode错误码说明

resultCode	resultMsg	自查方案
1001	License未初始化	请按照集成文档说明完成SDK初始化
1002	License数据解密失败	请检查License文件是否正确
1003	Licesen数据格式错误	请检查license文件内容有被修改过
1004	License-Key校验错误	请检查工程代码初始化参数中的licenseId，和控制台下载工程里面的licenseId是否匹配
1006	鸿蒙应用指纹校验错误	请检查工程所使用的签名文件，和控制台填写的签名信息是否匹配
1008	鸿蒙bundleName应用名称校验错误	请检查工程代码中的bundleName（包名）和控制台填写的鸿蒙包名信息是否匹配
1009	过期时间不正确	请提交工单或者线下联系百度产研人员
1011	授权已过期	请查看当前设备时间是否已不在授权文件有效期内
1012	本地文件读取失败	请检查授权文件名称以及路径
1013	本地鉴权失败	1）请检查工程代码中的bundleName（包名）和控制台填写的鸿蒙包名是否匹配；2）请检查工程所使用的签名文件，和控制台填写的签名信息是否匹配
1014	本地时间校验错误	请检查当前设备时间是否早于实际时间

6.3 人脸信息加密采集接口

- 包含本地质量和本地活体，本地质量可以确保采集到的人脸图像符合各条件校验（满足姿态角、光照、模糊度、遮挡等校验），本地活体分静默活体、炫瞳活体、动作活体三种。
- 此处最终采集到的数据经过加密处理，需要配合服务端的 人脸实名认证V4 或 人脸对比V4 或 在线图片活体V4 来使用。分别用来实现「权威数据源身份信息核验」、「本地图片无源比对」以及「仅活体检测」，适用于不同的业务场景需要。
- **sKey、xDeviceId、data** 为此接口的成功回调结果信息，作为上述3个服务端接口重要字段入参

返回值	API	描述
void	startFaceCollect(params: HashMap<string, Object>, faceServiceCallbck: FaceServiceCallbck)	人脸采集接口

onCallback回调说明

参数	类型	含义	值
resultCode	int	错误码	0为成功，其他为失败，详情参考实名认证resultCode错误码说明
resultMap	HashMap	回调结果Map	详情见下表

resultMap key值列表说明：

Key值	类型	含义
resultMsg	String	详情见resultCode错误码说明
sKey	String	安全相关：sKey
xDeviceId	String	安全相关：xDeviceId
data	String	安全相关数据

7. 人脸释放接口（SDK）

在完成加密信息采集后，用于释放SDK，实现对采集功能、模型的释放，减小内存。

返回值	API	描述
void	release()	人脸释放接口

权限

名称	说明	必选
需要动态申请的权限		
ohos.permission.CAMERA	拍照权限	是
不需要动态申请的权限		
ohos.permission.INTERNET	允许访问网络	是

常见问题

如何获取鸿蒙应用指纹 （1）方法1，测试证书

- “应用指纹”signatureInfo.fingerprint是鸿蒙应用签名证书(.cer文件)的SHA-256hash值，当前支持获取本应用的“应用指纹”。示例代码如下：

```
import { bundleManager } from '@kit.AbilityKit';
import { hilog } from '@kit.PerformanceAnalysisKit';
import { BusinessError } from '@kit.BasicServicesKit';

let bundleFlags = bundleManager.BundleFlag.GET_BUNDLE_INFO_WITH_SIGNATURE_INFO;
try {
  bundleManager.getBundleInfoForSelf(bundleFlags).then((data) => {
    hilog.info(0x0000, 'testTag', 'getBundleInfoForSelf successfully. Data: %{public}s', JSON.stringify(data));
    //data里可以获取到signatureInfo，即应用的签名证书信息
  }).catch((err: BusinessError) => {
    hilog.error(0x0000, 'testTag', 'getBundleInfoForSelf failed. Cause: %{public}s', err.message);
  });
} catch (err) {
  let message = (err as BusinessError).message;
  hilog.error(0x0000, 'testTag', 'getBundleInfoForSelf failed: %{public}s', message);
}
```

（2）方法2，测试证书+发布证书

- 在项目级build-profile.json5文件中，signingConfigs字段内的profile的值即为签名文件的存储路径。
- 打开该签名文件（**后缀为.p7b**），打开后在文件内搜索“-certificate”，将“-----BEGIN CERTIFICATE-----”和“-----END CERTIFICATE-----”以及中间的信息拷贝到新的文本中，注意换行并去掉换行符，保存为一个新的.cer文件，如命名为xxx.cer。
  ：

```
-----BEGIN CERTIFICATE-----
MIICMzCCAbegAwIBAgIEaOC/zDAMBggqhkjOPQQDAwUAMGMxCzAJBgNVBAYTAkNO
MRQwEgYDVQQKEwtPcGVuSGFybW9ueTEZMBcGA1UECxMQT3BlbkhhcmlvbnkgVGVh
bTEjMCEGA1UEAxMaT3BlbkhhcmlvbnkgQXBwbGljYXRpb24gQ0EwHhcNMjEwMjAy
MTIxOTMxWhcNNDkxMjMxMTIxOTMxWjBoMQswCQYDVQQGEwJDTjEUMBIGA1UEChML
T3BlbkhhcmlvbnkxGTAXBgNVBAsTEE9wZW5IYXJtb255IFRlYW0xKDAmBgNVBAMT
H09wZW5IYXJtb255IEFwcGxpY2F0aW9uIFJlbGVhc2UwWTATBgcqhkjOPQIBBggq
hkjOPQMBBwNCAATbYOCQQpW5fdkYHN45v0X3AHaxl2jPBdEDosFRIZleXmxOYzSG
JwMfsHhUU90E8lI0TXYZnNmgMlsovubeQqATolIwUDAfBgNVHSMEGDAWgBTbhrci
FtULoUu33SV7ufEFfaItRzAOBgNVHQ8BAf8EBAMCB4AwHQYDVR0OBBYEFPtxruhl
cRBQsJdwcZqLu9oNUVgaMAwGCCqGSM49BAMDBQADaAAwZQIxAJta0PQ2p4DIu/ps
LMdLCDgQ5UHll0B4PGhBlMgdi2zf8nk9spazEQI/0XNwpft8QAIwHSuA2WelVi/o
zAlF08DnbJrOOtOnQq5wHOPlDYB4OtUzOYJk9scotrEnJxJzGsh/
-----END CERTIFICATE-----
```

- 使用keytool工具（在DevEco Studio安装目录下的jbr/bin文件夹内），执行如下命令通过.cer文件获取证书指纹的SHA256值。
  `keytool -printcert -file xxx.cer`
- 将证书指纹中SHA256的内容去掉冒号，即为最终要获得的签名指纹。

verify_token的详细说明 （1）access_token 有效期为 30 天，重复生成 access_token 的话，对之前的不影响，依旧能使用；access_token 与verify_token 是包含关系，即 verify_token 是由 access_token生成的，如果一个 verify_token 是和一个不匹配的 access_token 使用，会提示“Token无效或者已过期”

（2）verify_token 的核验有效期为 2 小时，在有效期内可以进行了核验动作，如果超过了 2 小时没有使用该 token 进行核验的话会提示“Token无效或者已过期”

（3） verify_token 核验完毕后（核验成功，或设置了认证未通过时URL失效），无法再进行核验，如果再次进行核验，会提示“Token无效或者已过期”

（4） verify_token 核验完毕后（核验成功，或设置了认证未通过时URL失效），支持对后验接口的查询，有效期均为 3 天，3 天后再次查询后验接口，会提示“Token无效或者已过期” **获取认证人脸、查询认证结果、核验及计费信息获取接口持续返回错误码18，提示openapi限制，且有返回logid 答：**检查APP方案依赖的“人脸实名认证V4”、“人脸对比V4”、“在线图片活体检测V4”三项付费接口，是否有免费额度，或者直接开通后付费，以消除该报错提示。开通后，再进行重试。

人脸实名认证V4

人脸实名认证（V4)接口

人脸识别接口分为V2、V3和V4三个版本，本文档为V4版本接口的说明文档，请确认您在百度智能云获得的是V4版本接口权限，再来阅读本文档。

> 辨别接口版本的方法是：在百度智能云-控制台进入【应用列表】，点击【应用名称】，在【API列表】中可以看到【请求地址】，若请求地址中带有【V4】标识，则您具有的是v4权限，可以阅读本文档；若请求地址中带有【V3】标识，则您具有的是V3权限，应该去阅读v3文档。

如果您对文档内容有任何疑问，可以通过以下几种方式联系我们：

- 在百度智能云-控制台内 提交工单，咨询问题类型请选择**人工智能服务**；
- 如有需要讨论的疑问，欢迎进入 AI社区 与其他开发者们一同交流。

能力介绍

业务能力

- **质量检测（可选）**：判断图片中是否包含人脸，以及人脸在姿态、遮挡、模糊、光照等方面是否符合识别条件。
- **活体检测（可选）**：基于图片中的破绽分析，判断其中的人脸是否为**二次翻拍**（举例：如用户A用手机拍摄了一张包含人脸的图片一，用户B翻拍了图片一得到了图片二，并用图片二伪造成用户A去进行识别操作，这种情况普遍发生在金融开户、实名认证等环节）。
- **图片加密及风控（可选）**：配合采集SDK使用，对采集SDK输出的加密图片进行解密（加密传输可以有效避免第三方非法黑产绕过APP模拟请求攻击云端接口的行为，Eg：脚本攻击等）；以及结合百度安全风控能力，对采集SDK的发起端设备进行风控识别，辨别是否为风险设备，Eg：ROM注入、视频劫持等；
- **人脸实名认证（必选）**：基于姓名、身份证号、当前获取的人脸图片，与权威库进行三要素一致性对比，得出比对分数，并基于此进行业务判断是否为同一人。由于权威库具有身份证明作用，故对用户本人的验证结果可信度也最为合理。

业务逻辑

- 上述能力，人脸实名认证能力为必选能力，质量检测、活体检测、图片加密及风控为可选能力，验证顺序为`人脸质量检测->活体检测->人脸实名认证`。
- **如选择了质量检测和活体检测能力，则有任意一个条件不通过，整个请求流程终止，并返回错误码，描述具体的不符合信息**。
- 基于此顺序串行验证逻辑，可以避免大量不符合条件的请求流转到人脸实名认证，节约您的成本。

推荐阈值

- 此接口使用的对比算法，针对带水纹证件照采用了专项的模型处理，可保证水纹信息的影响降到尽可能低。
- 如比对成功，最终返回的有效数据为一个**对比分值**，在0~100之间，您可以设定具体的阈值来判断是否验证通过。
- **人证相似度的推荐阈值为80，对应的误识率为万分之一**。

在线调试 您可以在 示例代码中心 中调试该接口，可进行签名验证、查看在线调用的请求内容和返回结果、示例代码的自动生成。

请求说明

注意事项：

- **请求体格式化**：Content-Type为`application/json`，通过`json`格式化请求体。

- **Base64编码**：请求的图片需经过Base64编码，图片的base64编码指将图片数据编码成一串字符串，使用该字符串代替图像地址。您可以首先得到图片的二进制，然后用Base64格式编码即可。需要注意的是，图片的base64编码是不包含图片头的，如``
- **图片格式**：现支持PNG、JPG、JPEG、BMP，**不支持GIF图片**

请求示例

HTTP方法：`POST`

请求URL： https://aip.baidubce.com/rest/2.0/face/v4/mingjing/verify

URL参数：

参数	值
access_token	通过API Key和Secret Key获取的access_token,参考“Access Token获取”

Header：

参数	值
Content-Type	application/json

Body中放置请求参数，参数详情如下：

请求参数

参数	必选	类型	说明
id_card_number	是	string	身份证件号
name	是	string	姓名(需要是 utf8 编码)
image	是	string	图片信息(数据大小应小于10M 分辨率应小于1920*1080)。**使用APP方案示例工程集成，即5.2+版本SDK时，图片信息已包含在加密数据data中，无需额外传入**
app	否	string	APP端类型，配合采集SDK使用时须传入 **ios**：iOS端采集SDK **android**：安卓端采集SDK **harmonyos**：鸿蒙Next端采集SDK
sec_level	否	string	SDK安全级别，配合采集SDK使用时须传入，默认common **common**：配合4.1/4.1.5版本采集SDK，人脸图片未进行加密处理 **lite**: 配合5.2+版本SDK
skey	否	string	使用5.2+版本SDK请求时必填 **skey**：从SDK获取的密钥信息skey
x_device_id	否	string	使用5.2+版本SDK请求时必填 **deviceId**：从SDK 获取的密钥信息deviceId
data	否	string	使用5.2+版本SDK请求时必填， SDK输出的加密数据
liveness_control	否	string	活体控制参数 **NONE**: 不进行控制 **LOW**:较低的活体要求(高通过率 低攻击拒绝率) **NORMAL**: 一般的活体要求(平衡的攻击拒绝率, 通过率) **HIGH**: 较高的活体要求(高攻击拒绝率 低通过率) **默认为NONE**
合成图控制参数			

spoofing_control	否	string	合成图控制参数 **NONE**: 不进行控制 **LOW**:较低的合成图阈值数值，由于合成图判定逻辑为大于阈值视为合成图攻击，该项代表**低通过率、高攻击拒绝率** **NORMAL**: 一般的合成图阈值数值，由于合成图判定逻辑为大于阈值视为合成图攻击，该项代表**平衡的攻击拒绝率, 通过率** **HIGH**: 较高的合成图阈值数值，由于合成图判定逻辑为大于阈值视为合成图攻击，该项代表**高通过率、低攻击拒绝率**) **默认为NONE**
quality_control	否	string	质量控制参数 **NONE**: 不进行控制 **LOW**:较低的质量要求 **NORMAL**: 一般的质量要求 **HIGH**: 较高的质量要求 **默认为NONE**
image_type	否	string	图片类型 **BASE64**：图片的base64值 **FACE_TOKEN**：人脸标识 **默认 BASE64**
get_liveness_score	否	string	是否返回活体分数 **0**（默认值）：不返回具体活体分数； 1：返回活体分数； **此字段生效前提为，请求字段liveness_control值不为NONE**
get_spoofing_score	否	string	是否返回合成图分数 **0**（默认值）：不返回合成图分数； 1：返回合成图分数； **此字段生效前提为，请求字段spoofing_control值不为NONE**

请求示例

```
// 纯API接入或配合4.1/4.1.5版本SDK使用
{
    "image_type": "BASE64",
    "image": "/9j/4AAQSkZJR",
    "id_card_number": "131102********0653",
    "name": "***",
    "quality_control": "LOW",
    "liveness_control": "LOW",
    "spoofing_control": "LOW"
}




// 5.2及更高版本SDK
{
    "app": "android",
    "sec_level": "lite",
    "x_device_id": "72c18da95552259dd7c4aaa52e9fa37f",
    "skey": "2fwYi/alzQBUJUc2TJn1oQ==",
    "data": "asdafasf",
    "image_type": "BASE64",
    "id_card_number": "131102********0653",
    "name": "***",
    "quality_control": "LOW",
    "liveness_control": "LOW",
    "spoofing_control": "LOW",
}
```

返回参数

参数	类型	说明
log_id	number	调用的日志id
result	jsonObject	认证返回的结果
+score	float	与权威库人脸图相似度可能性，用于验证生活照与权威库人脸图是否为同一人，有正常分数时为[0~100]，推荐阈值80，超过即判断为同一人
+verify_status	number	认证状态，取值如下： 0 ： 正常 1 ： 身份证号与姓名不匹配或该身份证号不存在 2 ： 公安网图片不存在或质量过低
+spoofing_score	number	合成图分数，默认不返回，仅当请求参数get_spoofing_score为1且spoofing_control不为none时返回
+liveness_score	number	活体分数，默认不返回，仅当请求参数get_liveness_score为1且liveness_control不为none时返回
dec_image	string	对SDK传入的加密图片进行解密。 仅APP场景且进行了图片加密时，此参数返回解密后的人脸图片信息
risk_level	string	判断设备是否发生过风险行为来判断风险级别，取值（数值由高到低）： 1 – 高危 2 – 嫌疑 3 – 普通 4 – 正常
risk_tag	string	风险标签，若判断为有风险，则会有风险标签json 数组告知风险类型 例如：general_inject

返回示例


```
{
  "log_id": 1370579072568000512,
  "result": {
     "score": 40.884,
     "verify_status": 0
  },
  "dec_image": "/9j/4AAQSkZJRgABAgAAAQABAAD",
  "risk_level": "3",
  "risk_tag": [
     "若判断为有风险，则会有风险标签json 数组告知风险类型，如：general_inject"
  ]
}
```


参数说明

- **质量控制参数说明**

> 不同的控制度下所对应的质量控制阈值，如果检测出来的质量信息某一项不符合控制阈值的要求，则会返回错误信息。

控制度	left_eye	right_eye	nose	mouth	left_cheek	right_cheek	chin_contour	illumination	blurdegree	completeness
LOW	0.8	0.8	0.8	0.8	0.8	0.8	0.8	20	0.8	0
NORMAL	0.6	0.6	0.6	0.6	0.6	0.6	0.6	40	0.6	0
HIGH	0.3	0.3	0.2	0.2	0.3	0.3	0.3	50	0.2	1

活体控制参数说明

> 不同的控制度下所对应的活体控制阈值，如果检测出来的活体分数小于控制阈值，则会返回错误信息。

控制度	阈值	说明
LOW	0.05	活体误拒率：万分之一；拒绝率：97.75%
NORMAL（推荐）	0.3	活体误拒率：千分之一；拒绝率：98.82%
HIGH	0.9	活体误拒率：百分之一；拒绝率：99.77%

> 1、误拒率：把**真人识别为假人**的概率。 阈值越高，安全性越高，要求也就越高，对应的误识率就越高。 2、通过率=1-误拒率

关于以上数值的概念介绍：

拒绝率（TRR）：如99%，代表100次作弊假体攻击，会有99次被拒绝。 误拒率（FRR）：如0.5%，指1000次真人请求，会有5次因为活体分数低于阈值被错误拒绝。 通过率（TAR）：如99%，指100次真人请求，会有99次因为活体分数高于阈值而通过。 阈值（Threshold）：高于此数值，则可判断为活体。

合成图控制参数说明

> 不同的控制度下所对应的合成图检测（PS、人脸融合等）阈值，如果检测出来的分数大于控制阈值，则会返回错误信息。

控制度	阈值	误拒率（FRR）	通过率	攻击拒绝率（TRR）
LOW	0.00023	5%	95%	94.93%
NORMAL（推荐）	0.00048	1%	99%	89.71%
HIGH	0.00109	0.1%	99.9%	84.57%

> 1、误拒率：把**正常图片识别为合成图片**的概率。阈值越低，安全性越高，要求也就越高，对应的误识率就越高。 2、通过率=1-误拒率

关于以上数值的概念介绍：

阈值（Threshold）：高于此数值，则可判断为是合成图攻击。

错误码

请参考错误码说明文档。

人脸对比V4

人脸对比(V4)

人脸识别接口分为V2、V3和V4三个版本，本文档为V4版本接口的说明文档，请确认您在百度云后台获得的是V4版本接口权

限，再来阅读本文档。

> 辨别接口版本的方法是：在百度云后台进入【应用列表】，点击【应用名称】，在【API列表】中可以看到【请求地址】，若请求地址中带有【V4】标识，则您具有的是v4权限，可以阅读本文档；若请求地址中带有【V3】标识，则您具有的是V3权限，应该去阅读v3文档。

如果您对文档内容有任何疑问，可以通过以下几种方式联系我们：

- 在百度云控制台内 提交工单，咨询问题类型请选择**人工智能服务**；
- 如有需要讨论的疑问，欢迎进入 AI社区 与其他开发者们一同交流。

能力介绍

接口能力

- **两张人脸图片相似度对比**：对比两张图片中人脸的相似度，并返回相似度分值；
- **多种图片类型**：支持**生活照**、**证件照**、**身份证芯片照**、**带网纹照**四种类型的人脸对比；
- **活体检测控制**：基于图片中的破绽分析，判断其中的人脸是否为**二次翻拍**（举例：如用户A用手机拍摄了一张包含人脸的图片一，用户B翻拍了图片一得到了图片二，并用图片二伪造成用户A去进行识别操作，这种情况普遍发生在金融开户、实名认证等环节。）；
- **质量检测控制**：分析图片的中人脸的模糊度、角度、光照强度等特征，判断图片质量；
- **图片加密及风控：配合人脸采集SDK使用
  - 对采集SDK输出的加密图片进行解密（加密传输可以有效避免第三方非法黑产**绕过APP模拟请求攻击云端接口的行为**，Eg：脚本攻击等）；
  - 以及结合百度安全风控能力，对采集SDK的发起端设备进行风控识别，辨别是否为**风险设备**，Eg：ROM注入、视频劫持等；

业务应用

- 用于对比多张图片中的人脸相似度并返回两两对比的得分，可用于判断两张脸是否是同一人的可能性大小。
- 典型应用场景：如**人证合一验证**，**用户认证**等，可与您现有的人脸库进行对比验证。

在线调试 您可以在 示例代码中心 中调试该接口，可进行签名验证、查看在线调用的请求内容和返回结果、示例代码的自动生成。

调用方式

请求URL数据格式

向API服务地址使用POST发送请求，必须在URL中带上参数`access_token`，可通过后台的API Key和Secret Key生成，具体方式请参考“Access Token获取”。

示例代码

Bash
PHP
JAVA
Python

Cpp

C#

Node

```
##### !/bin/bash
curl -i -k 'https://aip.baidubce.com/oauth/2.0/token?grant_type=client_credentials&client_id=【百度云应用的AK】&client_secret=【百度云应用的SK】'
```

> **注意**：access_token的有效期为30天，**切记需要每30天进行定期更换，或者每次请求都拉取新token**；

例如此接口，使用HTTPS POST发送：

```
https://aip.baidubce.com/rest/2.0/face/v4/mingjing/match?access_token=24.f9ba9c5341b67688ab4added8bc91dec.2592000.1485570332.282335-8574074
```

POST中Body的参数，按照下方请求参数说明选择即可。

> **提示**：如果您为百度云老用户，正在使用其他非AI的服务，可以参考百度云AKSK鉴权方式发送请求，虽然**请求方式**和**鉴权方法**和本文所介绍的不同，但请求参数和返回结果一致。

请求说明

注意事项：

- **请求体格式化**：Content-Type为application/json，通过json格式化请求体。
- **Base64编码**：请求的图片需经过Base64编码，图片的base64编码指将图片数据编码成一串字符串，使用该字符串代替图像地址。您可以首先得到图片的二进制，然后用Base64格式编码即可。需要注意的是，图片的base64编码是不包含图片头的，如
- **图片格式**：现支持PNG、JPG、JPEG、BMP，**不支持GIF图片**

请求示例

HTTP方法：POST

请求URL： https://aip.baidubce.com/rest/2.0/face/v4/mingjing/match

URL参数：

参数	值
access_token	通过API Key和Secret Key获取的access_token,参考“Access Token获取”

Header如下：

参数	值
Content-Type	application/json

Body中放置请求参数，参数详情如下：

请求参数

参数	必选	类型	说明
image	是	string	图片信息(数据大小应小于10M 分辨率应小于1920*1080)，**5.2+版本SDK请求时已包含在加密数据data中，无需额外传入**
image_type	是	string	图片类型 **BASE64**：图片的base64值 **FACE_TOKEN**：人脸标识 默认 BASE64
face_type	否	string	人脸的类型 **LIVE**：表示生活照：通常为手机、相机拍摄的人像图片、或从网络获取的人像图片等 **IDCARD**：表示身份证芯片照：二代身份证内置芯片中的人像照片 **WATERMARK**：表示带水印证件照：一般为带水印的小图，如公安网小图 **CERT**：表示证件照片：如拍摄的身份证、工卡、护照、学生证等证件图片 **INFRARED** 表示红外照片：使用红外相机拍摄的照片 默认LIVE
app	否	string	APP端类型，配合采集SDK使用时须传入 **ios**：iOS端采集SDK **android**：安卓端采集SDK **harmonyos**：鸿蒙Next端采集SDK
sec_level	否	string	SDK安全级别，配合采集SDK使用时须传入，默认common **common**：配合4.1/4.1.5版本SDK使用，人脸图片未进行加密处理 **lite**：配合5.2+版本SDK使用
skey	否	string	使用5.2+版本SDK请求时必填 从SDK获取的密钥信息
x_device_id	否	string	使用5.2+版本SDK请求时必填 从SDK获取的密钥信息
data	否	string	使用5.2+版本SDK请求时必填 SDK输出的加密数据
quality_control	否	string	质量控制 **NONE**: 不进行控制 **LOW**:较低的质量要求 **NORMAL**: 一般的质量要求 **HIGH**: 较高的质量要求 **默认NONE**
liveness_control	否	string	活体控制 **NONE**: 不进行控制 **LOW**:较低的活体要求(高通过率 低攻击拒绝率) **NORMAL**: 一般的活体要求(平衡的攻击拒绝率, 通过率)

			HIGH: 较高的活体要求(高攻击拒绝率 低通过率) **默认NONE**
spoofing_control	否	string	合成图控制参数 **NONE**: 不进行控制 **LOW**:较低的合成图阈值数值，由于合成图判定逻辑为大于阈值视为合成图攻击，该项代表**低通过率、高攻击拒绝率** **NORMAL**: 一般的合成图阈值数值，由于合成图判定逻辑为大于阈值视为合成图攻击，该项代表**平衡的攻击拒绝率, 通过率** **HIGH**: 较高的合成图阈值数值，由于合成图判定逻辑为大于阈值视为合成图攻击，该项代表**高通过率、低攻击拒绝率** **默认NONE**
register_image	是	string	图片信息(总数据大小应小于10M)，图片上传方式根据image_type来判断。本图片特指客户服务器上传图片，非加密图片Base64值
register_image_type	是	string	图片类型 **BASE64**：图片的base64值 **FACE_TOKEN**：人脸标识 默认 BASE64
register_face_type	否	string	人脸的类型 **LIVE**：表示生活照：通常为手机、相机拍摄的人像图片、或从网络获取的人像图片等 **IDCARD**：表示身份证芯片照：二代身份证内置芯片中的人像照片 **WATERMARK**：表示带水印证件照：一般为带水印的小图，如公安网小图 **CERT**：表示证件照片：如拍摄的身份证、工卡、护照、学生证等证件图片 **INFRARED**: 表示使用红外相机拍摄的照片 默认LIVE
register_quality_control	否	string	图片质量控制 **NONE**: 不进行控制 **LOW**:较低的质量要求 **NORMAL**: 一般的质量要求 **HIGH**: 较高的质量要求 **默认 NONE** 若图片质量不满足要求，则返回结果中会提示质量检测失败
register_liveness_control	否	string	活体检测控制 **NONE**: 不进行控制 **LOW**:较低的活体要求(高通过率 低攻击拒绝率) **NORMAL**: 一般的活体要求(平衡的攻击拒绝率, 通过率) **HIGH**: 较高的活体要求(高攻击拒绝率 低通过率) **默认 NONE** 若活体检测结果不满足要求，则返回结果中会提示活体检测失败
register_spoofing_control	否	string	合成图控制参数 **NONE**: 不进行控制 **LOW**:较低的合成图阈值数值，由于合成图判定逻辑为大于阈值视为合成图攻击，该项代表**低通过率、高攻击拒绝率** **NORMAL**: 一般的合成图阈值数值，由于合成图判定逻辑为大于阈值视为合成图攻击，该项代表**平衡的攻击拒绝率, 通过率** **HIGH**: 较高的合成图阈值数值，由于合成图判定逻辑为大于阈值视为合成图攻击，该项代表**高通过率、低攻击拒绝率** **默认 NONE**
			人脸检测排序类型

face_sort_type	否	int	0:代表检测出的人脸按照人脸面积从大到小排列 1:代表检测出的人脸按照距离图片中心从近到远排列 默认为0
get_liveness_score	否	string	是否返回活体分数 0（默认值）：不返回具体活体分数； 1：返回活体分数； **此字段生效前提为，请求字段liveness_control值不为NONE**
get_spoofing_score	否	string	是否返回合成图分数 0（默认值）：不返回合成图分数； 1：返回合成图分数； **此字段生效前提为，请求字段spoofing_control值不为NONE**
register_get_liveness_score	否	string	是否返回活体分数 0（默认值）：不返回具体活体分数； 1：返回活体分数； **此字段生效前提为，请求字段liveness_control值不为NONE**
register_get_spoofing_score	否	string	是否返回合成图分数 0（默认值）：不返回合成图分数； 1：返回合成图分数； **此字段生效前提为，请求字段spoofing_control值不为NONE**

请求示例

```
// 纯API接入或配合4.1/4.1.5版本SDK使用
{
    "image": "/9j/4AAQSkZJRgA",
    "image_type": "BASE64",
    "face_type": "LIVE",
    "quality_control": "LOW",
    "liveness_control": "LOW",
    "register_image": "/9j/4AAQSkZJRgA",
    "register_image_type": "BASE64",
    "register_face_type": "IDCARD",
    "register_quality_control": "LOW",
    "register_liveness_control": "LOW"
}

//// 5.2及更高版本SDK
{
    "app": "android",
    "sec_level": "lite",
    "x_device_id": "72c18da95552259dd7c4aaa52e9fa37f",
    "skey": "2fwYi/alzQBUJUc2TJn1oQ==",
    "data": "asdafasf",
    "image_type": "BASE64",
    "face_type": "LIVE",
    "quality_control": "LOW",
    "liveness_control": "LOW",

    "register_image": "/9j/4AAQSkZJRgA",
    "register_image_type": "BASE64",
    "register_face_type": "IDCARD",
    "register_quality_control": "LOW",
    "register_liveness_control": "LOW"
}
```

返回参数

参数名	类型	说明
log_id	number	调用的日志id
result	jsonObject	认证返回的结果
+ score	number	人脸相似度得分，推荐阈值80分
+ face_list	jsonArray	人脸信息列表
++ face_token	string	人脸标志
++spoofing_score	number	合成图分数，默认不返回，仅当请求参数get_spoofing_score/register_get_spoofing_score为1，且spoofing_control/register_spoofing_control不为none时返回
++liveness_score	number	活体分数，默认不返回，仅当请求参数get_liveness_score/register_get_liveness_score为1，且liveness_control/register_liveness_control不为none时返回
dec_image	string	APP场景传入加密图片时，该项返回解密后的图片
risk_level	string	风控返回参数，判断设备是否发生过风险行为来判断风险级别，取值（数值有高到低）： 1 – 高危 2 – 嫌疑 3 – 普通 4 – 正常
risk_tag	jsonArray	风控返回参数，风险标签，若判断为有风险，则会有风险标签 json 数组告知风险类型
risk_warn_code	number	风控返回参数，只有在风控服务异常的时候才返回这个字段

返回示例代码：

- 返回示例


```
{
 "log_id": 1370585066551377920,
 "result": {
    "score": 99.06919861,
    "face_list": [
       {
          "face_token": "549f9f1d1c7ec8c86931540b1939e8ed"
       },
       {
          "face_token": "1a319460ef89e8d27fb59062a28dbad7"
       }
    ]
 },
 "dec_image": "/9j/4AAQSkZJRgABAQAAAQABAAD",
 "risk_level": "3",
 "risk_tag": [
    "空"
 ]
}
```


参数说明

- **质量控制参数说明**

> 不同的控制度下所对应的质量控制阈值，如果检测出来的质量信息某一项不符合控制阈值的要求，则会返回错误信息。

遮挡情况的阈值

控制度	left_eye	right_eye	nose	mouth	left_cheek	right_cheek	chin_contour
LOW	0.8	0.8	0.8	0.8	0.8	0.8	0.8
NORMAL	0.6	0.6	0.6	0.6	0.6	0.6	0.6
HIGH	0.2	0.2	0.2	0.2	0.2	0.2	0.2

光照、模糊度、完整度的阈值

控制度	illumination	blurdegree	completeness
LOW	20	0.8	0
NORMAL	40	0.6	0
HIGH	100	0.2	1

活体控制参数说明

> 不同的控制度下所对应的活体控制阈值，如果检测出来的活体分数小于控制阈值，则会返回错误信息。

控制度	阈值	说明
LOW	0.05	活体误拒率：万分之一；拒绝率：97.75%
NORMAL	0.3	活体误拒率：千分之一；拒绝率：98.82%
HIGH	0.9	活体误拒率：百分之一；拒绝率：99.77%

> 1、误拒率: 把**真人识别为假人**的概率. 阈值越高，安全性越高, 要求也就越高, 对应的误识率就越高
> 2、通过率=1-误拒率

关于以上数值的概念介绍：

拒绝率（TRR）：如99%，代表100次作弊假体攻击，会有99次被拒绝。 误拒率（FRR）：如0.5%，指1000次真人请求，会有5次因为活体分数低于阈值被错误拒绝。 通过率（TAR）：如99%，指100次真人请求，会有99次因为活体分数高于阈值而通过。 阈值（Threshold）：高于此数值，则可判断为活体。

错误码

请参考人脸识别错误码

在线图片活体V4

人脸识别接口分为V2、V3、V4三个版本，本文档为V4版本接口的说明文档，请确认您在百度云后台获得的是V4版本接口权限，再来阅读本文档。

> 辨别接口版本的方法是：在百度云后台进入【应用列表】，点击【应用名称】，在【API列表】中可以看到【请求地址】，若请求地址中带有【V4】标识，则您具有的是V4权限，可以阅读本文档；若请求地址中带有【V3】标识，则您具有的是V3权限，应该去阅读V3文档。

如果您对文档内容有任何疑问，可以通过以下几种方式联系我们：

- 在百度云控制台内 提交工单，咨询问题类型请选择**人工智能服务**；
- 如有需要讨论的疑问，欢迎进入 AI社区 与其他开发者们一同交流。

能力介绍

接口能力

- **人脸基础信息**：包括人脸框位置，人脸空间旋转角度，人脸置信度等信息。
- **人脸质量检测**：判断人脸的遮挡、光照、模糊度、完整度等质量信息。可用于判断上传的人脸是否符合标准。
- **基于图片的活体检测**：基于单张图片，判断图片中的人脸是否为二次翻拍（举例：如用户A用手机拍摄了一张包含人脸的图片一，用户B翻拍了图片一得到了图片二，并用图片二伪造成用户A去进行识别操作，这种情况普遍发生在金融开户、实名认证等环节）以及是否为合成图攻击。此能力可用于H5场景下的一些人脸采集场景中，增加人脸注册的安全性和真实性。
- **图片加密及风控**：配合采集SDK5.0版本使用，对采集SDK输出的加密图片进行解密（加密传输可以有效避免第三方非法黑产绕过APP模拟请求攻击云端接口的行为，Eg：脚本攻击等）；以及结合百度安全风控能力，对采集SDK的发起端设备进行风控识别，辨别是否为风险设备，Eg：ROM注入、视频劫持等；

在线调试

您可以在 示例代码中心 中调试该接口，可进行签名验证、查看在线调用的请求内容和返回结果、示例代码的自动生成。

调用方式

请求URL数据格式

向API服务地址使用POST发送请求，必须在URL中带上参数`access_token`，可通过后台的API Key和Secret Key生成，具体方式请参考“Access Token获取”。

示例代码

Bash
PHP
JAVA
Python
Cpp
C#
Node

```
##### !/bin/bash
curl -i -k 'https://aip.baidubce.com/oauth/2.0/token?grant_type=client_credentials&client_id=【百度云应用的AK】&client_secret=【百度云应用的SK】'
```

> **注意**：access_token的有效期为30天，**切记需要每30天进行定期更换，或者每次请求都拉取新token**；

例如此接口，使用HTTPS POST发送：

```
https://aip.baidubce.com/rest/2.0/face/v4/faceverify?access_token=24.f9ba9c5341b67688ab4added8bc91dec.2592000.1485570332.282335-8574074
```

POST中Body的参数，按照下方请求参数说明选择即可。

> **提示**：如果您为百度云老用户，正在使用其他非AI的服务，可以参考百度云AKSK鉴权方式发送请求，虽然**请求方式**和**鉴权方法**和本文所介绍的不同，但请求参数和返回结果一致。

请求说明

注意事项：

- **请求体格式化**：Content-Type为application/json，通过json格式化请求体。
- **Base64编码**：请求的图片需经过Base64编码，图片的base64编码指将图片数据编码成一串字符串，使用该字符串代替图像地址。您可以首先得到图片的二进制，然后用Base64格式编码即可。需要注意的是，图片的base64编码是不包含图片头的，如
- **图片格式**：现支持PNG、JPG、JPEG、BMP，**不支持GIF图片**
- **人脸识别接口分为V2和V3、V4三个版本，本文档为V4版本接口的说明文档，请确认您在百度云后台获得的是V4版本接口权限，再来阅读本文档。**

> 辨别接口版本的方法是：在百度云后台进入【应用列表】，点击【应用名称】，在【API列表】中可以看到【请求地址】，若请求地址中带有【v4】标识，则您具有的是v4权限，可以阅读本文档；若请求地址中带有【v3】标识，则您具有的是v3权限，应该去阅读v3文档。

请求示例

HTTP方法：POST

请求URL： `https://aip.baidubce.com/rest/2.0/face/v4/faceverify`

URL参数：

参数	值
access_token	通过API Key和Secret Key获取的access_token,参考“Access Token获取”

Header：

参数	值
Content-Type	application/json

Body中放置请求参数，参数详情如下：

请求参数

参数	是否必选	类型	说明
sdk_version	是	string	1 ：非加密图片，适用于4.1/4.1.5版本SDK、H5场景或纯服务端场景 4：适用于5.2+版本SDK
face_field	否	string	包括**age,beauty,expression,face_shape,gender,glasses,landmark,quality,face_type,spoofing**信息，逗号分隔，默认只返回face_token、活体数、人脸框、概率和旋转角度
option	否	string	场景信息，程序会视不同的场景选用相对应的模型。当前支持的场景有COMMON(通用场景)，GATE(闸机场景)，FINANCE(金融场景)，LOGISTICS(物流场景)，INTERNET(泛互联网场景), **默认使用COMMON** 。注意：如果请求参数中存在多个option值时，则取第一个option的值，同时除通用场景外的其他场景需要联系工作人员对您所使用的appid进行配置
app	否	string	APP端类型，配合采集SDK使用时须传入 **ios**：iOS端采集SDK **android**：安卓端采集SDK **harmonyos**：鸿蒙Next端采集SDK
s_key	否	string	端上提供的用于解密图片的skey **5.2+版本SDK必传该项**
device_id	否	string	端上提供的用于解密图片的deviceId **5.2+版本SDK必传该项**
data	否	string	端上提供的加密后的图片数组 **5.2+版本SDK必传该项**
image_list	否	array	图片BASE64数组 4.1/4.1.5版本SDK、H5场景或纯服务端场景必填

请求示例

```
// 纯API接入或配合4.1/4.1.5版本SDK使用
{
  "option": "COMMON",
  "face_field": "spoofing",
  "image_list": [
    "/9j/4AAQSkZJR",
    "/9j/4AAQSkZJR"
  ]
}



// 5.2及更高版本SDK
{
    "sdk_version": "4",
    "s_key": "LNbv3Namn3Wg8e3Szq9Nwg==",
    "device_id": "dca010ccb8df410de90540038925c752",
    "data": "qipE4E10qFAr9m\/3U",
    "app": "android",
    "option": "COMMON",
    "face_field": "spoofing"
}
```

返回参数

参数	是否必须	类型	说明
log_id	是	string	日志id
error_code	是	int	错误码状态，若为0则表示认证成功
error_msg	是	string	错误码说明，若为 success 则表示认证成功
result	是	object	活体结果
+face_liveness	是	float	活体分数值
+thresholds	是	array	由服务端返回最新的阈值数据（随着模型的优化，阈值可能会变化），将此参数与返回的face_liveness进行比较，可以作为活体判断的依据。 frr_1e-4：万分之一误识率的阈值； frr_1e-3：千分之一误识率的阈值； frr_1e-2：百分之一误识率的阈值。 误识率越低，准确率越高，相应的拒绝率也越高
+face_list	是	array	每张图片的详细信息描述，如果只上传一张图片，则只返回一个结果。
++face_token	是	string	人脸图片的唯一标识
++location	是	array	人脸在图片中的位置
+++left	是	double	人脸区域离左边界的距离
+++top	是	double	人脸区域离上边界的距离
+++width	是	double	人脸区域的宽度
+++height	是	double	人脸区域的高度
+++rotation	是	int64	人脸框相对于竖直方向的顺时针旋转角，[-180,180]
++face_pro			

++face_probability	是	double	人脸置信度，范围【0~1】，代表这是一张人脸的概率，0最小、1最大。
++angel	是	array	人脸旋转角度参数
+++yaw	是	double	三维旋转之左右旋转角[-90(左), 90(右)]
+++pitch	是	double	三维旋转之俯仰角度[-90(上), 90(下)]
+++roll	是	double	平面内旋转角[-180(逆时针), 180(顺时针)]
++age	否	double	年龄 ，当**face_field包含age时返回**
++expression	否	array	表情，当 **face_field包含expression时返回**
+++type	否	string	**none**:不笑；**smile**:微笑；**laugh**:大笑
+++probability	否	double	表情置信度，范围【0~1】，0最小、1最大。
++face_shape	否	array	脸型，当**face_field包含face_shape时返回**
+++type	否	double	**square**: 正方形 **triangle**:三角形 **oval**: 椭圆 **heart**: 心形 **round**: 圆形
+++probability	否	double	置信度，范围【0~1】，代表这是人脸形状判断正确的概率，0最小、1最大。
++gender	否	array	性别，**face_field包含gender时返回**
+++type	否	string	male:**男性** female:**女性**
+++probability	否	double	性别置信度，范围【0~1】，0代表概率最小、1代表最大。
++glasses	否	array	是否带眼镜，**face_field包含glasses时返回**
++type	否	string	**none**:无眼镜，**common**:普通眼镜，**sun**:墨镜
+++probability	否	double	眼镜置信度，范围【0~1】，0代表概率最小、1代表最大。
++face_type	否	array	真实人脸/卡通人脸 **face_field包含face_type时返回**
+++type	否	string	**human**: 真实人脸 **cartoon**: 卡通人脸
+++probability	否	double	人脸类型判断正确的置信度，范围【0~1】，0代表概率最小、1代表最大。
++landmark	否	array	4个关键点位置，左眼中心、右眼中心、鼻尖、嘴中心。**face_field包含landmark时返回**
++landmark72	否	array	72个特征点位置 **face_field包含landmark时返回**
++quality	否	array	人脸质量信息。**face_field包含quality时返回**
+++occlusion	否	array	人脸各部分遮挡的概率，范围[0~1]，0表示完整，1表示不完整
++++left_eye	否	double	左眼遮挡比例，[0-1] ， 1表示完全遮挡
++++right_eye	否	double	右眼遮挡比例，[0-1] ， 1表示完全遮挡
++++nose	否	double	鼻子遮挡比例，[0-1] ， 1表示完全遮挡
++++mouth	否	double	嘴巴遮挡比例，[0-1] ， 1表示完全遮挡
++++left_cheek	否	double	左脸颊遮挡比例，[0-1] ， 1表示完全遮挡

++++right_cheek	否	double	右脸颊遮挡比例，[0-1] ， 1表示完全遮挡
++++chin	否	double	下巴遮挡比例，，[0-1] ， 1表示完全遮挡
+++blur	否	double	人脸模糊程度，范围[0~1]，0表示清晰，1表示模糊
+++illumination	否	double	取值范围在[0~255], 表示脸部区域的光照程度 越大表示光照越好
+++completeness	否	int64	人脸完整度，0或1, 0为人脸溢出图像边界，1为人脸都在图像边界内
++spoofing	否	double	合成图打分 判断图片是否为合成图 face_field包含时返回spoofing
risk_level	否	string	判断设备是否发生过风险行为来判断风险级别，取值（数值有高到低）：1 – 高危 2 – 嫌疑 3 – 普通 4 – 正常
risk_tag	否	string	风险标签，若判断为有风险，则会有风险标签 json 数组告知风险类型
dec_image	否	string	非安全加固增强级采集sdk、非安全加固金融级采集sdk、安全加固增强级采集sdk、安全加固金融级采集sdk会返回解密后的原图

返回示例


```
{
  "result": {
    "thresholds": {
      "frr_1e-3": 0.3,
      "frr_1e-2": 0.9,
      "frr_1e-4": 0.05
    },
    "face_liveness": 0.1976952702,
    "face_list": [
      {
        "liveness": {
          "livemapscore": 0.1976952702
        },
        "angle": {
          "roll": 0.31,
          "pitch": -2.07,
          "yaw": 0.85
        },
        "face_token": "abd1b4ce743099336cfed40193ff4944",
        "location": {
          "top": 161.74,
          "left": 79.04,
          "rotation": 1,
          "width": 318,
          "height": 333
        },
        "face_probability": 1
      },
      {
        "liveness": {
          "livemapscore": 0.1976952702
        },
        "angle": {
          "roll": 0.31,
          "pitch": -2.07,
          "yaw": 0.85
```

```
        },
        "face_token": "abd1b4ce743099336cfed40193ff4944",
        "location": {
            "top": 161.74,
            "left": 79.04,
            "rotation": 1,
            "width": 318,
            "height": 333
        },
        "face_probability": 1
    },
    {
        "liveness": {
            "livemapscore": 0.1976952702
        },
        "angle": {
            "roll": 0.31,
            "pitch": -2.07,
            "yaw": 0.85
        },
        "face_token": "abd1b4ce743099336cfed40193ff4944",
        "location": {
            "top": 161.74,
            "left": 79.04,
            "rotation": 1,
            "width": 318,
            "height": 333
        },
        "face_probability": 1
    },
    {
        "liveness": {
            "livemapscore": 0.1976952702
        },
        "angle": {
            "roll": 0.31,
            "pitch": -2.07,
            "yaw": 0.85
        },
        "face_token": "abd1b4ce743099336cfed40193ff4944",
        "location": {
            "top": 161.74,
            "left": 79.04,
            "rotation": 1,
            "width": 318,
            "height": 333
        },
        "face_probability": 1
    },
    {
        "liveness": {
            "livemapscore": 0.1976952702
        },
        "angle": {
            "roll": 0.31,
            "pitch": -2.07,
            "yaw": 0.85
        },
        "face_token": "abd1b4ce743099336cfed40193ff4944",
        "location": {
            "top": 161.74,
            "left": 79.04,
            "rotation": 1,
```

```
            "rotation": 1,
            "width": 318,
            "height": 333
        },
        "face_probability": 1
    },
    {
        "liveness": {
            "livemapscore": 0.1976952702
        },
        "angle": {
            "roll": 0.31,
            "pitch": -2.07,
            "yaw": 0.85
        },
        "face_token": "abd1b4ce743099336cfed40193ff4944",
        "location": {
            "top": 161.74,
            "left": 79.04,
            "rotation": 1,
            "width": 318,
            "height": 333
        },
        "face_probability": 1
    },
    {
        "liveness": {
            "livemapscore": 0.1976952702
        },
        "angle": {
            "roll": 0.31,
            "pitch": -2.07,
            "yaw": 0.85
        },
        "face_token": "abd1b4ce743099336cfed40193ff4944",
        "location": {
            "top": 161.74,
            "left": 79.04,
            "rotation": 1,
            "width": 318,
            "height": 333
        },
        "face_probability": 1
    },
    {
        "liveness": {
            "livemapscore": 0.1976952702
        },
        "angle": {
            "roll": 0.31,
            "pitch": -2.07,
            "yaw": 0.85
        },
        "face_token": "abd1b4ce743099336cfed40193ff4944",
        "location": {
            "top": 161.74,
            "left": 79.04,
            "rotation": 1,
            "width": 318,
            "height": 333
        },
        "face_probability": 1
    }
```

```
        ]
    },
    "risk_level": "3",
    "log_id": 1423545654699779516,
    "risk_tag": [
        "空"
    ],
    "dec_image": [
            "/9j/4AAQ..."
    ]
}
```

活体阈值参考

请务必在产品侧做好以下条件限制

- 检测的图片为二次采集，即通过相机当场拍摄，确保时间及操作条件的约束
- SDK输出的多帧情况，只要这些帧中，**任何一张通过了阈值，即可判断为活体，建议可用三帧情况**

关于活体检测faceliveness的判断阈值选择，可参考以下数值信息。（金融保险等对安全性要求较高的业务可适当调高活体阈值，如：0.5）：

阈值（Threshold）	误拒率（FRR）	通过率（TAR）	攻击拒绝率（TRR））
0.05	0.01%	99.99%	97.75%
0.1	0.05%	99.95%	98.33%
0.3 （**推荐**）	0.1%	99.9%	98.82%
0.5	0.5%	99.5%	99.67%
0.9	1%	99%	99.77%

关于以上数值的概念介绍：

- **拒绝率（TRR）**：如99%，代表100次作弊假体攻击，会有99次被拒绝。
- **误拒率（FRR）**：如0.5%，指1000次真人请求，会有5次因为活体分数低于阈值被错误拒绝。
- **通过率（TAR）**：如99%，指100次真人请求，会有99次因为活体分数高于阈值而通过。
- **阈值（Threshold）**：高于此数值，则可判断为活体。

合成图阈值参考

> 新推出合成图检测能力，在face_field字段中增加spoofing参数，进行判断，若spoofing分值高于合成图推荐阈值，则可判断为合成图攻击；此参数在face_liveness基础上进行合成图判断。

关于合成图检测spoofing的判断阈值选择，可参考以下数值信息：

阈值	误拒率（FRR）	通过率（TAR）	攻击拒绝率（TRR））
0.00023	5%	95%	94.93%
0.00048（**推荐**）	1%	99%	89.71%
0.00066	0.5%	99.5%	88.02%
0.00109	0.1%	99.9%	84.57%
0.00171	0.05%	99.95%	81.52%
0.00547	0.01%	99.99%	65.52%

- **阈值（Threshold）**：高于此数值，则可判断为是合成图攻击。

质量检测参考

指标	字段与解释	推荐数值界限
遮挡范围	**occlusion**，取值范围[0~1]，0为无遮挡，1是完全遮挡 含有多个具体子字段，表示脸部多个部位 通常用作判断头发、墨镜、口罩等遮挡	left_eye：0.6, #左眼被遮挡的阈值 right_eye：0.6, #右眼被遮挡的阈值 nose：0.7, #鼻子被遮挡的阈值 mouth：0.7, #嘴巴被遮挡的阈值 left_check：0.8, #左脸颊被遮挡的阈值 right_check：0.8, #右脸颊被遮挡的阈值 chin_contour：0.6, #下巴被遮挡阈值
模糊度范围	**blur**，取值范围[0~1]，0是最清晰，1是最模糊	小于0.7
光照范围	**illumination**，取值范围[0~255] 脸部光照的灰度值，0表示光照不好 以及对应客户端SDK中，YUV的Y分量	大于40
姿态角度	**Pitch**：三维旋转之俯仰角度[-90(上), 90(下)] **Roll**：平面内旋转角[-180(逆时针), 180(顺时针)] **Yaw**：三维旋转之左右旋转角[-90(左), 90(右)]	分别小于20度
人脸完整度	**completeness**（0或1），0为人脸溢出图像边界，1为人脸都在图像边界内	视业务逻辑判断
人脸大小	人脸部分的大小 建议长宽像素值范围：80*80~200*200	人脸部分不小于**100*100**像素

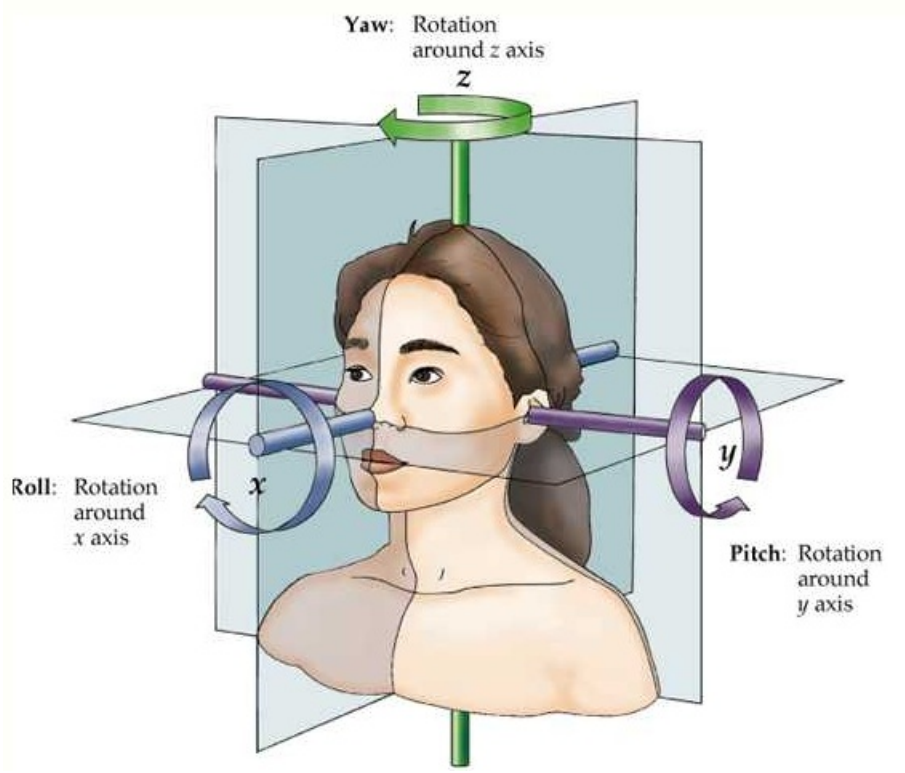
人脸空间姿态角参考

姿态角分为`Pitch`、`Roll`、`Yaw`，用于表示人脸在空间三维坐标系内的角度，常用于判断识别角度的界限值。

各角度阈值如下：

```
Pitch：三维旋转之俯仰角度，范围：[-90（上），90（下）]，推荐俯仰角绝对值不大于20度；
Roll：平面内旋转角，范围：[-180（逆时针），180（顺时针）]，推荐旋转角绝对值不大于20度；
Yaw：三维旋转之左右旋转角，范围：[-90（左），90（右）]，推荐旋转角绝对值不大于20度；
```

各角度范围示意图如下：



从姿态角度来看，**这三个值的绝对值越小越好**，这样代表人脸足够正视前方，最利于实际注册/识别使用。

错误码

请参考人脸识别错误码

常见问题

人脸实名认证常见问题

Q：实名认证支持核验的证件有哪些？

A：除二代居民身份证外，人脸实名认证API、身份证与名字比对API还支持以下证件，请提交工单或联系您的商务经理获得文档：

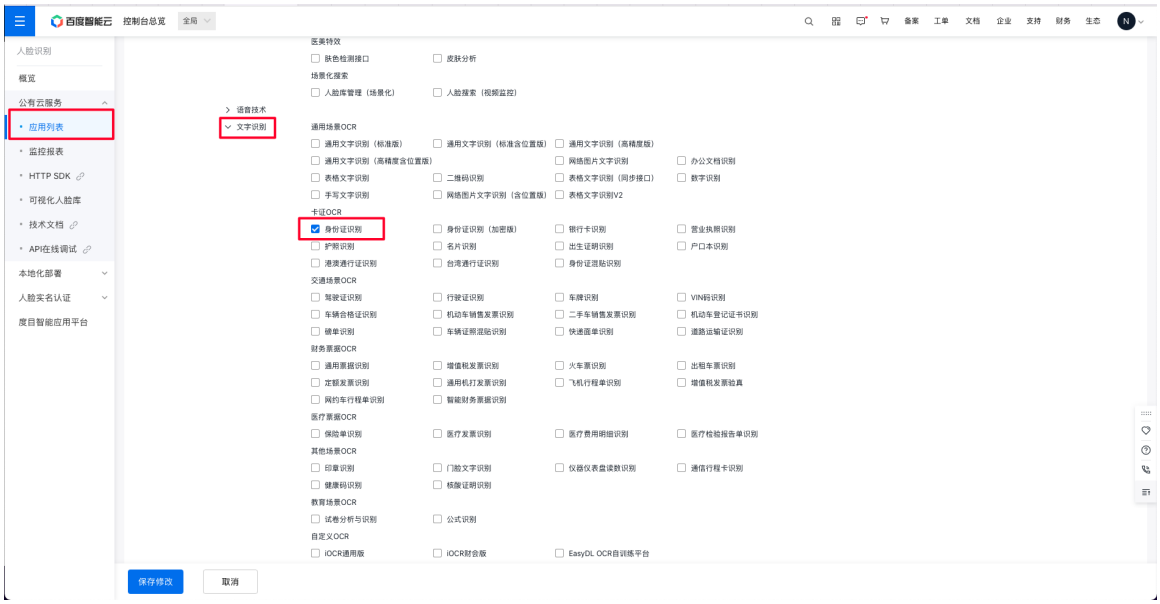
①港澳台居住证

Q：实名认证是否支持核验身份证的有效期？

A：可以的，支持核验+姓名+身份证号+证件起止有效期、人脸+姓名+身份证号+证件起止有效期，请提交工单或联系您的商务经理获得文档。

Q：使用人脸实名认证方案，是否需要开通OCR身份证识别API？

A：需要的，请参考文档在百度智能云控制台的应用中勾选身份证识别API、领取免费额度、开通后付费或购买次数包，避免因无额度影响正常使用。



H5方案常见问题

Q：H5方案配置完成后，无法通过百度智能云控制台生成二维码体验，如何解决？

A：当方案中选择「自建人脸库比对」、「指定用户身份核验」方式时，无法通过二维码体验，需集成H5方案接口上传信息后测试。

Q：H5方案获取verify_token一直报错“6”、“283400”，如何解决？

A：请确认是否已经创建H5方案，且access_token与**方案planid**是否对应，「人脸核真」系列API是否已获得权限。请注意，不要使用项目id。

Q：H5方案显示“服务异常，请稍后再试”，如何解决？

A：请先确认是否已经完成**企业认证**获得了「人脸实名认证」相关调用权限、免费额度或开通付费；另外请确认是否选择了「OCR身份证识别」能力，并已经**领取免费额度**或开通付费。

Q：业务审核需求要获取H5方案中用户的核验过程信息，如何拿到？

A：H5方案为了满足客户人工二次审核的需求，提供了两个后验接口。其中获取认证人脸可以查询报错的人脸信息，此接口支持两个小时有效期的查询，具体可参考simple文档。查询认证结果可查询验证成功/失败、身份证号、姓名、活体检测得分等。另外，百度智能云控制台已上线记录查询平台，在征得客户同意的情况下可查询三个自然日的核验记录。

Q：H5方案报错“283450 认证尚未开始”，如何解决

A：首先排查是否因账户欠费停服；如少量客户出现此报错信息，则可能是用户在核验过程中只输入身份证号和姓名，没有完成下一步的活体核验流程，导致日志中只有身份信息的录入，没有核验结果。

Q：H5方案是否可以嵌入APP内

A：确认可支持APP内webview，但推荐使用原生方案，通过集成SDK实现相关功能。

Q：在APP内集成H5方案，遇到不显示摄像头框的情况，如何解决

A：处理方式：请先确认是否获取到摄像头权限。

Q：H5方案在iOS环境遇到摄像头黑屏情况，如何解决

A：处理方式参考：

current context, possibly because the user denied permission.”；经测试发现，iOS升级到12.2后，如需在WKWebView中自动播放视频、或者想通过JS调用方法的方式播放视频，必须修改WKWebView中的两个配置：**“mediaTypesRequiringUserActionForPlayback”设置为“.all”，这个属性为设置无需用户操作自动播放视频；**

“allowsInlineMediaPlayback”设置为“true”，这个属性为设置为video播放视频不全屏(inline)；

```
let config = WKWebViewConfiguration()
config.allowsInlineMediaPlayback = true
config.mediaTypesRequiringUserActionForPlayback = .all
webview = WKWebView.init(frame: CGRect.zero, configuration: config)
```

并且确保plist中开启了使用相机和调用相册权限：

Information Property List	Dictionary	(5 items)
Appearance	String	Light
Privacy - Camera Usage Description	String	是否允许使用相机
Privacy - Photo Library Usage Description	String	是否允许调用相册
App Transport Security Settings	Dictionary	(1 item)
Application Scene Manifest	Dictionary	(1 item)

Q：H5方案在demo场景下可以使用实时活体检测，但集成后遇到降级为视频录制活体方式，如何解决

A：建议检查iframe的allow属性。

APP方案常见问题

Q：SDK的提示音是否支持其他语种？

A：可根据自身业务需求提供音频包和文案，具体请提交工单或联系您的商务经理。

Q：如何查看当前使用的SDK版本？

A：查询路径为Android ：FaceEnvironment.java；iOS：FaceSDKManager getVersion

Q：APP展示错误代码"SDK108"，如何解决

A：SDK108是网络连接出错，请检查运行环境是否能访问外网；如果可以访问外网，把超时时间设置较大后重试。

Q：APP是否支持横屏/后置摄像头

A：暂不支持。如有需求，请联系您的商务经理。

Q：APP方案生成后，无法下载对应的示例代码

A：未生成示例代码是由于OCR SDK未填写包名导致的，需要您到应用列表页，将项目关联的应用勾选身份证识别，并在下方的文字识别包名处切换至**需要**按钮，根据您的APP应用信息填写包名。

Step1：查看并进入项目关联应用

展开方案管理页面的项目信息，点击项目关联应用

Step2：编辑应用

进入关联应用后，点击左上角编辑操作，进入应用编辑页面

Step3：填写身份证识别包名信息

（1）进入应用编辑页面后，在接口选择中，展开文字识别类目，并勾选身份证识别选框

（2）

（2）勾选后，在底部文字识别包名处切换为需要按钮，并根据您的APP应用信息填写包名

Q：iOS SDK中的API_Key是Bundle ID么？ A：不是，Bundle ID是Xcode的`BundleIdentifier`，如下图所示：

Q：Android申请时需要填入打包签名的MD5，该MD5如何得来？

A：创建工程时生成签名文件`keystore.jks`（该文件会用于最终发布是打包，请认真对待），在命令行工具中使用`keytool -list -v -keystore keystore.jks`即可得到md5，删除冒号。

Q：人脸都用到那些授权？

A：OCR SDK需要使用授权文件aip.license，离线活体需要授权文件idl-license.face-android，在线接口需要ak/sk。

Q：Android SDK获取做动作过程中的图片，不带黑边的那种，具体怎么拿到？

A：参见`FaceDetectExpActivity`中当在`OnDetectCompletion`回调时候的`saveImage`方法。

错误码

若请求错误，服务器将返回的JSON文本包含以下参数：

- **error_code**：错误码。
- **error_msg**：错误描述信息，帮助理解和解决发生的错误。

通用错误码信息可查看：https://ai.baidu.com/ai-doc/FACE/5k37c1ujz

1、流控及鉴权通用错误码

错误码	错误信息	描述	处理建议
2	Service temporarily unavailable	服务暂不可用	服务暂不可用，请再次请求，如果持续出现此类错误，请在控制台提交工单联系技术支持团队
4	Open api request limit reached	集群超限额	集群超限额，请再次请求，如果持续出现此类错误，请在控制台提交工单联系技术支持团队
6	no permission to access data	没有接口权限	请确认您调用的接口已经被赋权 常见问题是有V3版本权限，调用的是v2版本接口； 需要企业认证的，开通企业认证后一个小时左右即可使用。
17	Open api daily request limit reached	每天流量超限额	免费测试资源使用完毕，每天请求量超限额，已支持计费的接口，您可以在控制台人脸识别选择购买相关接口的次数包或开通按量后付费；邀测和未支持计费的接口，您可以在控制台提交工单申请提升限额
18	Open api qps request limit reached	QPS超限额	可直接自助购买更多QPS、联系商务接口人、或者提交工单
19	Open api total request limit reached	请求总量超限额	免费测试资源使用完毕，每天请求量超限额，已支持计费的接口，您可以在控制台人脸识别选择购买相关接口的次数包或开通按量后付费；邀测和未支持计费的接口，您可以在控制台提交工单申请提升限额
100	Invalid parameter	无效的access_token参数	token拉取失败，可以参考“Access Token获取”重新获取
110	Access token invalid or no longer valid	Access Token失效	**token有效期为30天，注意需要定期更换**，也可以每次请求都拉取新token
111	Access token expired	Access token过期	**token有效期为30天，注意需要定期更换**，也可以每次请求都拉取新token

2、方案功能接口错误码

2.1 获取verify_token接口

错误码	错误信息	说明
283513	请求参数错误	请求体非 json，检查 body
283435	方案不存在。Plan not exist.	方案 id 不存在，检查请求参数
283512	服务异常，请稍后再试	redis 连接失败，请稍后尝试
283520	请传入核验方式	app 方案 match_source 参数未传入，检查请求参数
216100	invalid param	参数格式错误

2.2 指定用户信息上报接口

错误码	错误信息	说明
283442	身份证信息不合法，请重新填写	上传的身份证信息不合法，建议重新填写正确的身份证信息
283513	请求参数错误	请求体非 json，检查 body
283400	服务异常，请稍后再试	服务异常，请稍后再试，如频繁出现，可提交工单反馈
283458	当前链接已失效，请重试	当前链接已失效，请重新生成链接
216100	invalid param	参数格式错误
283437	Token无效或已过期，请重新生成	认证 token 已过期或无效，请重新生成 token 后再进行请求
222038	证件号或国籍参数格式错误	传入的证件号或国籍参数格式不正确

2.3 对比图片上传接口

错误码	错误信息	说明
283463	人脸图片质量检测未通过	人脸图片质量检测未通过，建议上传更清晰的图片
283400	服务异常，请稍后再试	服务异常，请稍后再试，如频繁出现，可提交工单反馈
222915	system busy	服务异常，请稍后再试，如频繁出现，可提交工单反馈
283513	请求参数错误	请求体非 json，检查 body
283462	比对源配置错误	对比源配置错误，建议核对传入的数据是否正确
283465	人脸图片活体检测未通过	非实时视频核验方案未通过，请重试
222202	图片中没有人脸	上传的图片中没有检测到人脸
283437	Token无效或已过期，请重新生成	认证 token 已过期或无效，请重新生成 token 后再进行请求
283456	图片为空或格式不正确	上传的图片为空或格式不正确
222203	无法解析人脸	无法解析上传的人脸图片

3、验证后查询接口错误码

3.1 获取认证人脸接口

错误码	错误信息	说明
223120	活体检测未通过	此次活体检测结果为非活体
17	Open api daily request limit reached	每天流量超限额
300201	您已拒绝授权摄像头，如需继续，请点击重新拍摄	用户拒绝授权实时视频流获取，无法完成核验流程
283450	认证尚未开始。Verification is not started yet.	认证流程尚未开始。
283451	认证处理中。Verification is running.	认证流程正在处理中。
800001	采集超时	用户人脸采集流程超时
222351	证件号与姓名不匹配	提交的身份证号与姓名不匹配
300001	受当前环境限制（300001），请更换浏览器或设备重试	当前环境不兼容实时检测，且方案配置为不允许降级，无法完成核验流程
223050	voice similarity low error	视频中的语音与语音验证码所对应数值相似度过低
999999	请确保是本人操作且正脸采集	请确保是本人进行操作且采集的人脸为正脸
800002	炫瞳检测失败	用户炫瞳检测环节验证失败
216509	视频中的声音无法识别	视频中的声音无法识别（声音过低或者有杂音导致无法识别）
223135	video extract image spoofing check fail	视频提取图片合成图检测失败

283437	Token无效或已过期，请重新生成	认证 token 已过期或无效，请重新生成 token 后再进行请求
223115	人脸光照不好	上传的人脸图片光照不好
223052	动作验证未通过	动作验证未通过，无法确认身份
223051	唇语验证未通过	唇语验证未通过，无法确认身份
223133	video extract image liveness check fail	视频提取图片活体检测失败
223116	人脸不完整	上传的人脸图片不完整
216908	视频中人脸质量较差	视频中人脸质量过低 返回的错误信息会包含 **illumiantion**(光照不足) **angle**(角度不好) **blur**(人脸模糊) **occlusion**(有遮挡) **too large**(人脸过大,**占屏幕2/3以上**)等原因
222202	图片中没有人脸	上传的图片中没有检测到人脸
223122	质量检测未通过 右眼遮挡程度过高	上传的人脸图片中右眼被遮挡程度过高
223114	人脸模糊	人脸模糊
216510	动作活体模式验证时视频长度超过10s	动作活体模式下，视频长度不能超过10秒
223125	质量检测未通过 下巴遮挡程度过高	上传的人脸图片中下巴遮挡程度过高
283516	留存视频上传失败	留存视频上传失败，请联系技术支持
223131	合成图检测未通过	合成图检测未通过
223126	质量检测未通过 鼻子遮挡程度过高	鼻子被遮挡，无法正常检测
223121	质量检测未通过 左眼遮挡程度过高	上传的人脸图片中左眼被遮挡程度过高
282000	请确保是本人操作且正脸采集	业务逻辑层服务异常，请稍后重试
283400	服务异常，请稍后再试	H5 内部服务异常，请稍后重试
800006	录制视频过大	非实时方案视频录制超过 20M
283512	服务异常，请稍后再试	redis 服务异常，请稍后再试
223123	质量检测未通过 左脸遮挡程度过高	上传的人脸图片中左脸被遮挡程度过高
283455	超出查询有效期。Result is expired.	查询结果已经过期
222023	名字格式错误	请检查姓名格式是否正确
223124	质量检测未通过 右脸遮挡程度过高	上传的人脸图片中右脸被遮挡程度过高
216909	请确保录制的视频中仅有一人	请确保录制的视频中仅有一人
223127	质量检测未通过 嘴巴遮挡程度过高	嘴巴被遮挡，无法正常检测
283457	当前环境存在安全风险	当前环境检测触发了安全风控规则
216502	当前会话已失效	请重新获取语音验证码
18	Open api qps request limit reached	Open API QPS请求次数限制已达到上限
222915	人脸服务异常，请稍后再试	人脸服务异常，请稍后再试
300002	受当前环境限制（300002），请更换浏览器或设备重试	当前环境不兼容实时视频录制，且方案配置为不允许降级，无法完成核验流程
300201	您已拒绝授权摄像头, 如需继续, 请点击重新拍摄	用户拒绝授权实时视频流获取，无法完成核验流程
216433	video service error	视频解析服务发生错误

283512	调用上游服务超时	调用上游服务超时，请再次尝试
2	调用OpenAPI服务超时	调用OpenAPI服务超时，请再次尝试
222013	param[image] format error	参数格式错误
216201	image format error	图片格式失败，请检查传入的图片base64格式是否有误；若使用的是安全版本SDK则是图片解密失败
222022	身份证号码格式错误	身份证号码格式错误
222350	公安网图片不存在或质量过低	此用户的信息没有被公安数据覆盖到，请将此次身份验证转到人工进行处理
216431	voice service error	语音识别服务异常
283513	请求参数错误	请求体非 json，检查 body
283511	请确保是本人操作且正脸采集	VMS 网关服务异常，请稍后重试
216100	invalid param	参数格式错误
216512	action verify must post session_id	使用动作活体验证时必须使用会话id
283449	活体检测视频不符合要求，请重新上传	上传的活体检测视频不符合要求，建议重新上传符合要求
222356	验证的人脸图片质量不符合要求	确认图片上的人脸是否清晰 不能有多张人脸 图片大小不能超过600KB 请重新拍摄
216612	系统繁忙	算子网关异常，请稍后再试
216498	长时间未检测到符合质量要求的人脸	长时间未检测到符合质量要求的人脸图片
216499	炫瞳检测超时	炫瞳环节检测超时
216530	认证尚未完成（新增）	认证流程尚未完成

3.2 查询认证结果接口

错误码	错误信息	说明
216100	invalid param	参数格式错误
283450	认证尚未开始。Verification is not started yet.	认证流程尚未开始
222351	证件号与姓名不匹配	提交的身份证号与姓名不匹配
223120	活体检测未通过	此次活体检测结果为非活体
800001	采集超时	用户人脸采集流程超时
800002	炫瞳检测失败	用户炫瞳检测环节验证失败
216509	视频中的声音无法识别	视频中的声音无法识别（声音过低或者有杂音导致无法识别）
223131	合成图检测未通过	合成图检测未通过
223051	唇语验证未通过	唇语验证未通过，无法确认身份
300001	受当前环境限制（300001），请更换浏览器或设备重试	当前环境不兼容实时检测，且方案配置为不允许降级，无法完成核验流程
223052	动作验证未通过	动作验证未通过，无法确认身份
283516	留存视频上传失败	留存视频上传失败，请联系技术支持
223116	人脸不完整	上传的人脸图片不完整
999999	请确保是本人操作且正脸采集	请确保是本人进行操作且采集的人脸为正脸
216510	动作活体模式验证时视频长度超过10s	动作活体模式下，视频长度不能超过10秒

216510	动作活体模式验证时视频长度超过10s	动作活体模式下，视频长度不能超过10秒
223050	语音验证未通过	视频中的语音与语音验证码所对应数值相似度过低
283451	认证处理中。Verification is running.	认证流程正在处理中
300201	您已拒绝授权摄像头, 如需继续, 请点击重新拍摄	用户拒绝授权实时视频流获取，无法完成核验流程
216201	服务异常，请稍后再试	图片格式失败，请检查传入的图片base64格式是否有误；若使用的是安全版本SDK则是图片解密失败
283400	服务异常，请稍后再试	H5 内部服务异常，请稍后重试
18	Open api qps request limit reached	Open API QPS请求次数限制已达到上限
223122	质量检测未通过 右眼遮挡程度过高	上传的人脸图片中右眼被遮挡程度过高
223114	人脸模糊	采集的人脸图片模糊
222023	名字格式错误	请检查姓名格式是否正确
283457	当前环境存在安全风险	当前环境检测触发了安全风控规则
222202	图片中没有人脸	上传的图片中没有检测到人脸
300002	受当前环境限制（300002），请更换浏览器或设备重试	当前环境不兼容实时视频录制，且方案配置为不允许降级，无法完成核验流程
223127	质量检测未通过 嘴巴遮挡程度过高	嘴巴被遮挡，无法正常检测
800006	录制视频过大	非实时方案视频录制超过 20M
223121	质量检测未通过 左眼遮挡程度过高	上传的人脸图片中左眼被遮挡程度过高
216908	视频中人脸质量较差	视频中人脸质量过低
283458	当前链接已失效，请重试	当前链接已失效，请重试
283512	服务异常，请稍后再试	redis 服务异常，请稍后再试
222350	公安网图片不存在或质量过低	此用户的信息没有被公安数据覆盖到，请将此次身份验证转到人工进行处理
223125	质量检测未通过 下巴遮挡程度过高	上传的人脸图片中下巴遮挡程度过高
282000	服务异常，请稍后再试	业务逻辑层服务异常，请稍后重试
223133	服务异常，请稍后再试	视频提取图片活体检测失败
216909	请确保录制的视频中仅有一人	请确保录制的视频中仅有一人
223115	人脸光照不好	上传的人脸图片光照不好
223123	质量检测未通过 左脸遮挡程度过高	上传的人脸图片中左脸被遮挡程度过高
216431	服务异常，请稍后再试	语音识别服务异常
222022	身份证号码格式错误	身份证号码格式错误
222915	服务异常，请稍后再试	人脸服务异常，请稍后再试
216205	服务异常，请稍后再试	业务逻辑层服务异常，请稍后重试
283437	Token无效或已过期，请重新生成	认证 token 已过期或无效，请重新生成 token 后再进行请求
283513	请求参数错误	请求体非 json，检查 body
223135	服务异常，请稍后再试	视频提取图片合成图检测失败
223126	质量检测未通过 鼻子遮挡程度过高	鼻子被遮挡，无法正常检测
223124	质量检测未通过 右脸遮挡程度过高	上传的人脸图片中右脸被遮挡程度过高
2	服务异常，请稍后再试	调用OpenAPI服务超时，请再次尝试
216502	当前会话已失效	请重新获取语音验证码

283511	调用上游服务超时	VMS 网关服务异常，请稍后重试
283518	数据不存在	核验结果留存异常，请稍后重试
216512	服务异常，请稍后再试	使用动作活体验证时必须使用会话id
283449	活体检测视频不符合要求，请重新上传	上传的活体检测视频不符合要求，建议重新上传符合要求
216433	服务异常，请稍后再试	视频解析服务发生错误
216612	系统繁忙	算子网关异常，请稍后再试
216498	长时间未检测到符合质量要求的人脸	长时间未检测到符合质量要求的人脸图片
216499	炫瞳检测超时	炫瞳环节检测超时
216530	认证尚未完成（新增）	认证流程尚未完成

3.3 查询统计结果接口

错误码	错误信息	说明
283437	Token无效或已过期，请重新生成	认证 token 已过期或无效，请重新生成 token 后再进行请求

3.4 实时方案视频获取

错误码	错误信息	说明
283450	认证尚未开始。Verification is not started yet.	认证流程尚未开始
283437	Token无效或已过期，请重新生成	认证 token 已过期或无效，请重新生成 token 后再进行请求
283512	服务异常，请稍后再试	redis 服务异常，请稍后再试
283451	认证处理中。Verification is running.	认证流程正在处理中。
283400	服务异常，请稍后再试	H5 内部服务异常，请稍后重试
283458	当前链接已失效，请重试	当前链接已失效，请重新生成链接

3.5 核验及计费信息获取

错误码	错误信息	说明
0	核验成功	核验成功
2	服务异常，请稍后再试	openapi 服务异常，请稍后再试
18	Open api qps request limit reached	用户流控qps限流
223120	活体检测未通过	采集图片未通过活体校验
999999	请确保是本人操作且正脸采集	请确保是本人进行操作且采集的人脸为正脸
283457	当前环境存在安全风险	当前环境检测触发了安全风控规则
223114	人脸模糊	上传的人脸图片模糊不清
222350	公安网图片不存在或质量过低	公安网不存在此照片或照片质量过低
222351	证件号与姓名不匹配	提交的身份证号与姓名不匹配，或户籍信息异常
223115	人脸光照不好	上传的人脸图片光照不好
800001	采集超时	用户人脸采集流程超时
223124	质量检测未通过 右脸遮挡程度过高	上传的人脸图片中右脸被遮挡程度过高
223052	动作验证未通过	动作验证未通过，无法确认身份
800006	录制视频过大	非实时方案视频录制超过 20M
800002	炫瞳检测失败	用户炫瞳检测环节验证失败

282000	服务异常，请稍后再试	业务逻辑层服务异常，请稍后重试
222202	图片中没有人脸	上传的图片中没有检测到人脸
222023	名字格式错误	名字格式不正确。名字需不超过48位，且不包含特殊字符和数字
223121	质量检测未通过 左眼遮挡程度过高	上传的人脸图片中左眼被遮挡程度过高
223131	合成图检测未通过	合成图检测未通过
283516	留存视频上传失败	留存视频上传失败，请联系技术支持
222013	服务异常，请稍后再试	安全服务异常，请稍后再试
223125	质量检测未通过 下巴遮挡程度过高	上传的人脸图片中下巴遮挡程度过高
223127	质量检测未通过 嘴巴遮挡程度过高	嘴巴被遮挡，无法正常检测
216509	视频中的声音无法识别	视频中的声音质量过差，无法识别
223122	质量检测未通过 右眼遮挡程度过高	上传的人脸图片中右眼被遮挡程度过高
216510	动作活体模式验证时视频长度超过10s	动作活体模式下，视频长度不能超过10秒
222356	验证的人脸图片质量不符合要求	提交的人脸图片质量不符合要求，无法确认身份
222022	身份证号码格式错误	身份证号码格式不正确
283512	服务异常，请稍后再试	redis 服务异常，请稍后再试
222915	服务异常，请稍后再试	人脸服务异常，请稍后再试
223126	质量检测未通过 鼻子遮挡程度过高	鼻子被遮挡，无法正常检测
216908	视频中人脸质量较差	视频中人脸质量过低 返回的错误信息会包含 **illumiantion**(光照不足) **angle**(角度不好) **blur**(人脸模糊) **occlusion**(有遮挡) **too large**(人脸过大,**占屏幕2/3以上**)等原因
223050	语音验证未通过	语音验证未通过，无法确认身份
222361	服务异常，请稍后再试	人脸服务异常，请稍后再试
300001	受当前环境限制（300001），请更换浏览器或设备重试	当前环境不兼容实时检测，且方案配置为不允许降级，无法完成核验流程
300201	您已拒绝授权摄像头, 如需继续, 请点击重新拍摄	用户拒绝授权实时视频流获取，无法完成核验流程
216909	请确保录制的视频中仅有一人	请确保录制的视频中仅有一人
283400	服务异常，请稍后重试	H5 内部服务异常，请稍后重试
283511	调用上游服务超时	VMS 网关服务异常，请稍后重试
216502	当前会话已失效	当前会话已失效，请重新获取语音验证码
216501	没有找到人脸	无法在视频或图像中找到人脸
223123	质量检测未通过 左脸遮挡程度过高	上传的人脸图片中左脸被遮挡程度过高
216612	系统繁忙	算子网关异常，请稍后再试
216498	长时间未检测到符合质量要求的人脸	长时间未检测到符合质量要求的人脸图片
216499	炫瞳检测超时	炫瞳环节检测超时
216530	认证尚未完成（新增）	认证流程尚未完成

人脸意愿核身方案

方案简介

方案简介

百度人脸意愿核身方案是将人脸核身、实时活体检测、语音识别等能力结合打造的一套**用于核验用户身份及办理业务的真实意愿**的解决方案。
此方案通过收集用户证件信息， 并配合动作、打光、距离等多种实时活体检测能力完成人脸核身后，通过语音问答模式核验用户真实意愿，同时保留整个意愿核身过程中的过程信息，为业务提供可追溯的核验依据。
当前方案可**支持H5、小程序**等多种渠道接入，为您的业务提供安全可靠，合规可信，体验优良的用户意愿核验能力。

功能介绍

1、用户身份核验：

- 权威库比对：收集用户姓名、身份证号以及人像信息，与**权威库**进行信息比对，基于比对得分判断用户信息真实性
- 自传照片比对：支持将收集的人像信息与**业务自传底图**进行1:1比对，得出两张图片的相似度分数，判断用户身份

2、活体检测：提供**实时炫瞳、动作、远近、静默等多种活体检测能力**，可按照实际业务需求灵活选择。**基于云端多因子活体大模型**针对**金融、物流、泛互联网**业务场景进行专项模型优化，适配不同业务需求的同时，提供高效的防御拦截能力，可抵挡屏幕、照片、视频、换脸、面具、3D模型等非活体攻击，并保证高通过率和识别率。

3、语音意愿确认：通过语音识别方式确认用户的业务办理意愿，并可获取核验过程中的音视频信息，为业务信息回溯提供参考依据。

4、设备环境风控：基于前端页面采集的环境信息，对浏览器等环境进行风险特征扫描，辨别是否为存在设备环境风险，返回风险等级及标签结果，可有效防御黑产批量虚拟机、病毒侵入等攻击手段。

5、端云数据加密：前端支持对人像图片信息进行加密，在云端接口进行信息解密，利用这种端云结合的加解密方案，可有效避免第三方非法黑产绕过前端采集过程，模拟业务请求，尝试攻击云端接口的行为，如脚本攻击、注入攻击、视频劫持等，极大增加对常见黑产攻击手段的防御能力，保障端云双重维度的信息一致性。

价格说明

人脸意愿核身方案

> **说明：**
> 1、使用人脸意愿核身方案需完成企业认证
> 2、登录人脸识别控制台后，系统将自动发放免费测试额度
> 3、免费测试额度用尽后，可 购买次数包 或 开通按量后付费。

1、意愿核身（权威库）

- **功能说明：**

- 实现动作、打光等多种类型的**实时活体检测**，拦截假体攻击
- 通过姓名、身份证号、人脸信息与**权威数据源**比对，核验用户身份
- 通过**语音识别**能力，确认用户的业务办理意愿，并可获取核验过程中的音视频信息

- **温馨提示**：完成企业认证**后**，服务并发量支持**2QPS**，正式付费后，并发量将扩充至**10QPS**。如业务需扩容更多QPS，可以**联系您的商务经理**，或通过合作咨询联系我们。

（1）按量后付费：

月调用量（次）	**价格（元/次）**
0 < n <= 10,000	1
10,000 < n <= 100,000	0.9
100,000 < n <=200,000	0.75
200,000 < n	0.6

（2）预付费资源包

次数包	**价格**	**有效期**
1,000 次	970 元	一年
5,000次	4,600元	一年
10,000 次	8,700 元	一年
50,000 次	41,000 元	一年
100,000 次	76,000 元	一年
500,000 次	340,000 元	一年
1000,000 次	620,000 元	一年

> **说明**
> 1、本接口**只计费成功调用**。失败调用会返回对应的错误码，详情说明可参考错误码文档
> 2、预付费包年次数包**有效期为1年**，支持叠加购买，按照购买时间顺序依次抵扣，全部使用完毕后自动切换为按量后付费模式。次数包购买时会自动开通**按量后付费**，**请勿在消耗次数包时终止后付费，避免影响正常使用**。次数包超出有效期未抵扣额度自动失效，无法继续使用，请根据实际业务需求酌情购买。
> 3、预付费包年次数包购买后**7天内未产生调用**，如有需要，您可对购买的QPS进行自助退款，您可前往百度智能云'控制台--财务--退订管理'页面进行自助退订，详细退订规则见退订说明， **超过7天**或**已产生调用的次数包**无法退款。

2、意愿核身（自传照片）

- **功能说明**：
  - 实现动作、打光等多种类型的**实时活体检测**，拦截假体攻击
  - 活体检测过程中采集的人像信息，**与业务侧上传的底图进行相似度比对**，判断是否为同一人
  - 通过**语音识别**能力，确认用户的业务办理意愿，并可获取核验过程中的音视频信息
- **温馨提示**：完成企业认证**后，服务并发量支持**2QPS**，正式付费后，并发量将扩充至**10QPS**。如业务需扩容更多QPS，可以**联系您的商务经理**，或通过合作咨询联系我们。

（1）按量后付费：

月调用量（次）	价格（元/次）
0 < n <= 10,000	0.3
10,000 < n <= 100,000	0.27
100,000 < n <=200,000	0.21
200,000 < n	0.18

（2）预付费资源包

次数包	价格	有效期
10,000 次	2,800 元	一年
50,000次	13,000元	一年
100,000 次	25,000 元	一年
200,000 次	48,000 元	一年
500,000 次	100,000元	一年
1,000,000 次	180,000 元	一年

> **说明**
>
> 1、本接口**只计费成功调用**。失败调用会返回对应的错误码，详情说明可参考错误码文档
>
> 2、预付费包年次数包**有效期为1年**，支持叠加购买，按照购买时间顺序依次抵扣，全部使用完毕后自动切换为按量后付费模式。次数包购买时会自动开通**按量后付费，请勿在消耗次数包时终止后付费，避免影响正常使用**。次数包超出有效期未抵扣额度自动失效，无法继续使用，请根据实际业务需求酌情购买。
>
> 3、预付费包年次数包购买后**7天内未产生调用**，如有需要，您可对购买的QPS进行自助退款，您可前往百度智能云'控制台--财务--退订管理'页面进行自助退订，详细退订规则见退订说明， **超过7天**或**已产生调用的次数包**无法退款。

方案接入指南

本文档将帮助您完成人脸实名认证-H5意愿核身方案的创建及接入全流程，有H5意愿核身需求请 提交工单 咨询

一、 准备工作

在正式集成前，需要做一些准备工作，完成一些账号、应用及方案配置，具体如下：

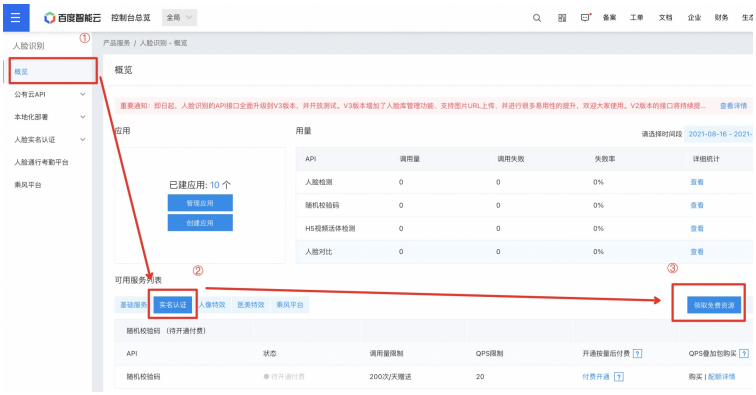
Step1: 注册成为开发者

> 在使用百度人脸实名认证方案之前，首先需注册百度智能云账号，账号注册方式请参考**账号注册指南**。
>
> 百度智能云账号注册完成以后，为顺利调用百度AI能力，需完成企业认证。具体认证方式请参考**企业认证指南**。

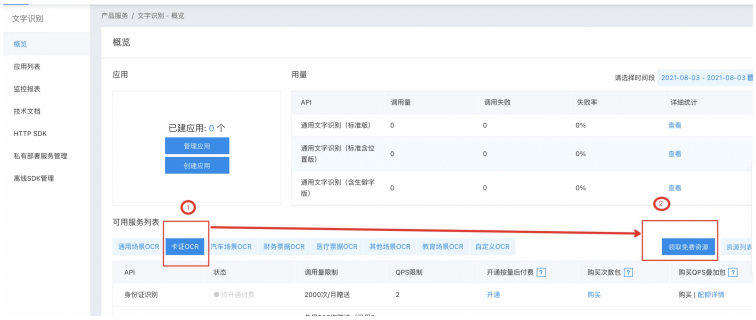
Step2：创建应用

2.1 输入应用名称，领取免费额度

- 创建好账号后，在正式调用AI 能力之前，需首先**创建应用**，应用是调用服务的基础能力单元，目前仅支持**人脸识别**应用。
- 同时领取接口所需的**免费调用额度**，用于接入测试。如下图所示：



- 除人脸服务接口的免费调用额度外，还需领取**身份证识别**接口的**免费调用额度**，用来调用身份证OCR识别功能（必须领取，否则会报错服务异常），点击此处，按下图所示进行领取。



- 如您之前已经领取过免费额度，无需重复领取，请跳至下一步骤。

2.2 勾选所需接口

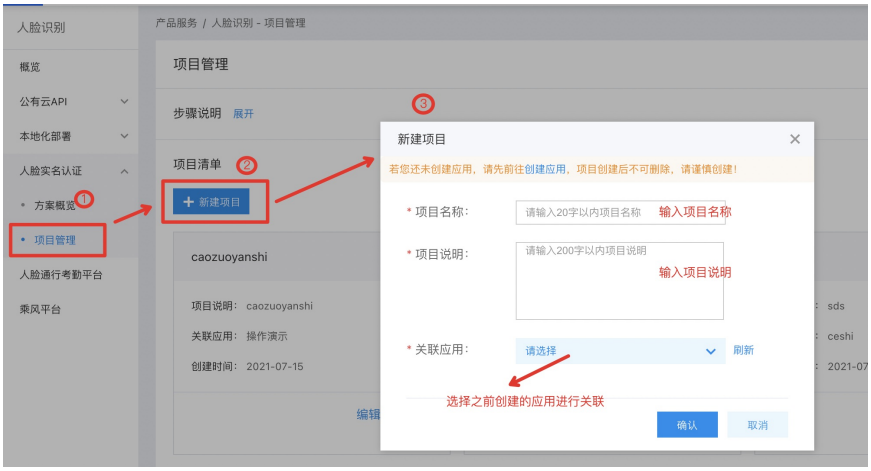
- 人脸识别服务相关接口已默认全部勾选，可根据自身业务进行调整。

- **注：**「接口选择」过程中，还**需勾选「文字识别」中的「身份证识别」接口**，用于实现身份核验流程中的身份证识别功能。如下图所示。

Step3：创建项目

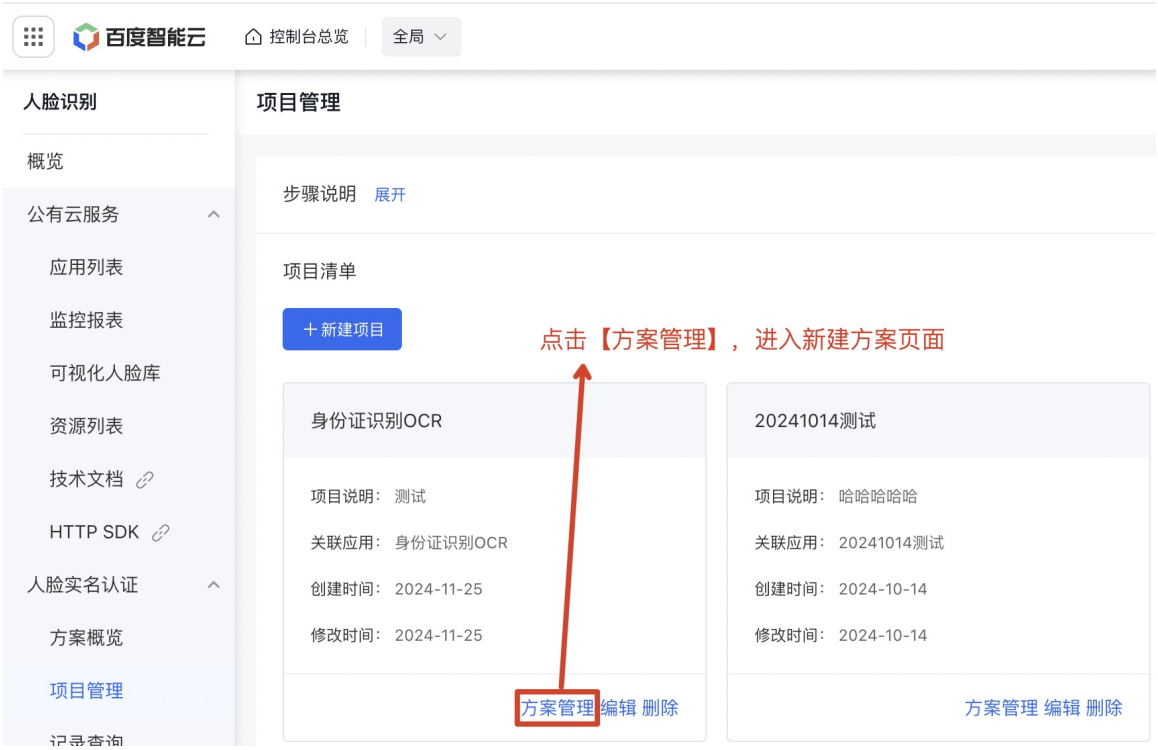
- 进入控制台-人脸实名认证页面，选择『**项目管理**』页面，点击『**新建项目**』，进行项目创建，如下图所示：

> 创建项目前，请确保您在应用控制台已创建应用，若您未创建应用，请参考Step2创建应用后，再进行项目创建.

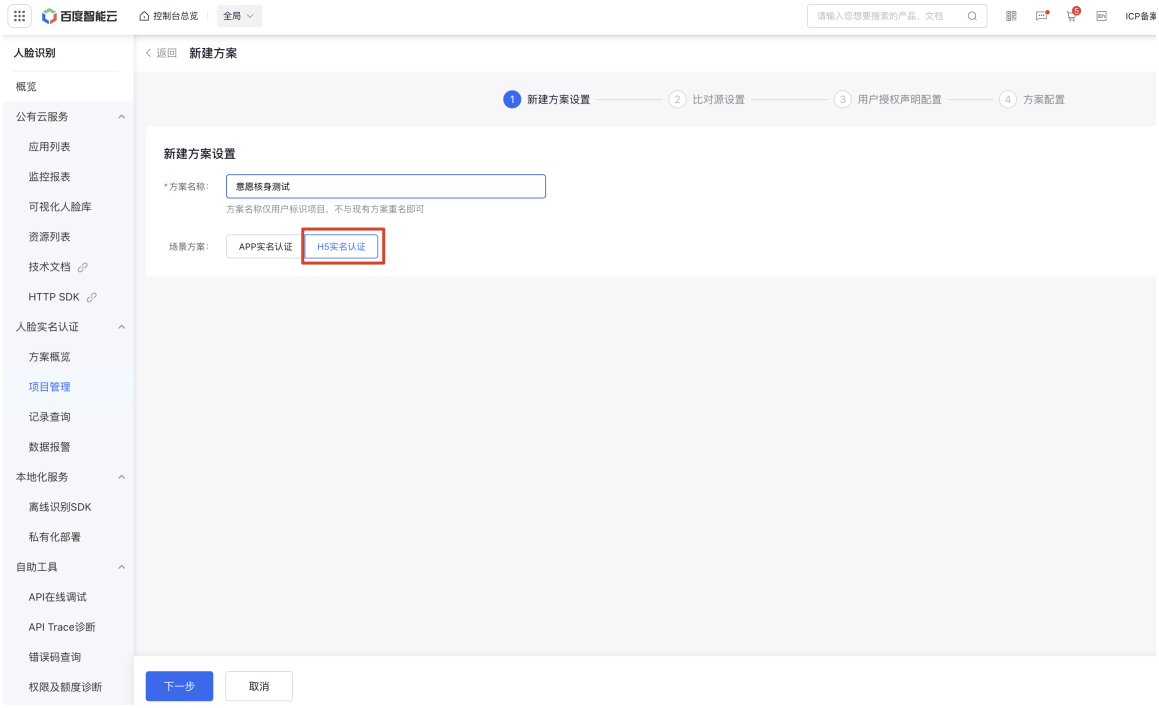


Step4：创建方案

- 项目创建完成后，点击「**方案管理**」进入方案管理页面，在这里您可以为您的项目创建不同的方案，如下图所示。



> 若您的场景为微信、H5页面，『场景方案』请选择H5实名认证方案。



4.1 比对源设置

- **比对源选择**：

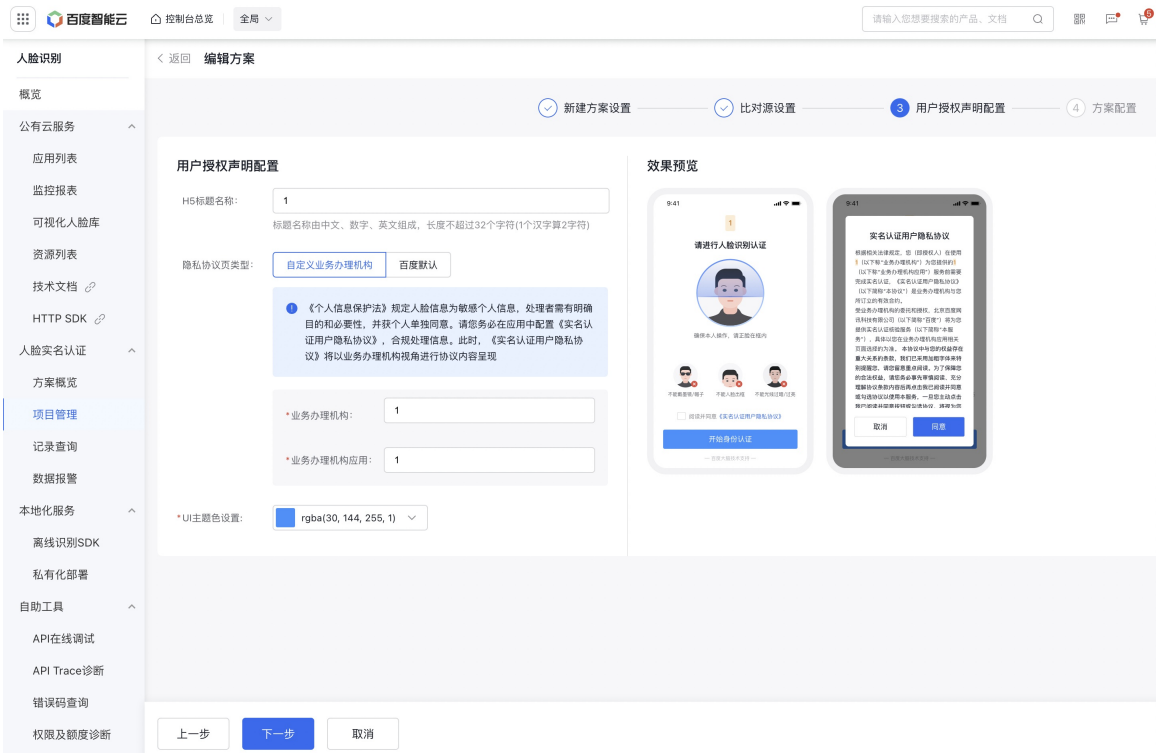
> **权威人脸库比对**：核验时需传入姓名及身份证号，实时采集人脸图片，与权威数据源进行一致性比对。
>
> **自建人脸库比对**：无需传入姓名或身份证号，实时采集人脸图片，与预先通过**对比图片上传API**上传的指定人脸图进行1：1比对。
>
> **仅活体检测**：无需传入姓名或身份证号，实时采集人脸图片进行真人检测，底层使用在线图片活体V4接口判断。

- **非大陆数据源**：默认不使用。配置打开后支持对中国居民二代身份证、港澳台居民往来内陆通行证、外国人永久居留证、定居国外的中国公民护照及港澳台居民居住证的验证。若不打开，则只支持中国居民二代身份证的验证。
- **身份信息录入方式**：支持身份证识别OCR、手动输入、指定用户身份核验三种。

> - 当选择**身份证识别OCR**后，支持配置**身份证风控功能**，开启后，对身份证的真伪进行判断，支持识别翻拍、PS伪造的身份信息，但开启后会存在少量误通过和误识别现象。同时支持配置身份证**人像面+国徽面及人像面的选择**，选择后通过查询接口可以查询到接口的返回信息。
> - 当选择**手动输入**后，支持用户自己输入姓名及身份证号信息。
> - 当选择**指定用户身份核验**后，支持指定用户的姓名+身份证号信息，用户侧无需输入，只需进行人脸采集及活体检测即可。 **注意：**若需要指定用户身份核验，则需要先请求 指定用户信息上报接口 再请求H5实名认证方案的URL。

4.2 用户授权声明配置

在这里您可以配置用户授权声明的信息，确保在得到用户授权的前提下，合规开展意愿核身业务。



4.3 方案配置

您可以根据自身业务使用情况,参考以下配置说明对图像质量、活体检测、合成图、意愿核身等参数进行灵活配置。

- **图像质量检测**：分为严格、正常、宽松三个等级，越严格，图片对角度、模糊度、遮挡等信息参数把控越高，推荐使用宽松。
- **活体检测**：分为严格、正常、宽松三个等级，越严格，对活体的检测等信息把控越高，推荐使用正常。
- **合成图检测**：分为严格、正常、宽松三个等级，越严格，对合成图的检测等信息把控越高，推荐使用正常。
- **活体方案选择**： 如下图所示，您可灵活选择适合自身业务的活体检测方式（如远近、炫瞳、动作等），同时也可针对业务场景选择不同的云端活体检测模型。并且，H5实名认证方案采用**实时活体检测形式**，**无需用户拍摄上传视频**，直接在前端完成检测流程，提升整体核验流程的流畅度及用户体验。

- **远近活体（实时）**：通过屏幕实时的交互动画提示，引导用户前后移动手机设备，由近及远或由远及近地进行多距离检测，配合眨眼动作验证，实现对3D头模、AIGC合成图、翻拍等活体攻击的拦截。相比于其他活体检测方式，对高清屏翻拍、AIGC合成图有更强的抵抗效果，适合对于安全性要求较高的业务场景。
- **炫瞳活体检测（实时）**：基于屏幕颜色打光的方式，通过面部反光和**瞳孔反光**对核验人员进行活体判断。相比于行业内传统的动作活体和视频活体检测方式，通过率大大提升，使用效率更加流畅便捷，有效拦截视频、图片伪造、3D面具、合成图等黑产攻击。**炫瞳活体检测**一般搭配1-2个前置动作，进一步提升安全性。
- **动作活体检测（实时）**：通过用户做指定动作来验证当前拍摄视频的用户是否为活体。支持指定动作的个数进行验证，支持设置指定1-3个验证动作，推荐使用1或者2个动作进行验证。
- **静默视频活体检测（实时）**：通过用户录制一段视频来验证当前拍摄视频的用户是否为活体。

附录：活体检测阈值指标（高于此阈值即判断为活体）

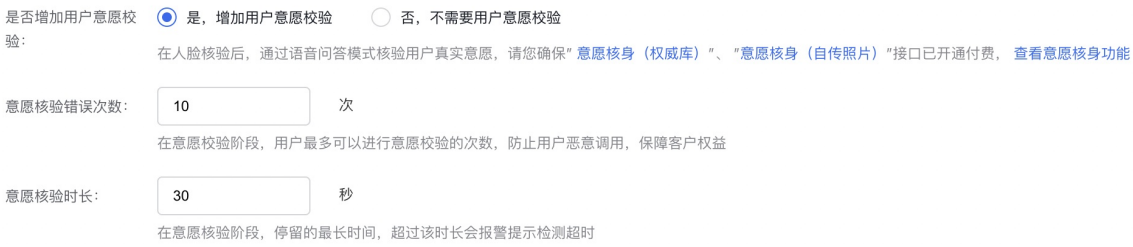
控制度	对应阈值	说明
宽松	0.05	万分之一活体误拒率
正常（推荐）	0.3	千分之一活体误拒率
严格	0.9	百分之一活体误拒率

附录：合成图检测阈值指标（高于此阈值即判断有合成图攻击风险）

控制度	阈值	误拒率（FRR）	通过率	攻击拒绝率（TRR）
宽松	0.00109	0.1%	99.9%	84.57%
正常（推荐）	0.00048	1%	99%	89.71%
严格	0.00023	5%	95%	94.93%

4.4 意愿核验配置

- **是否增加用户意愿检验**：意愿核身能力方案开关，选择「是，增加用户意愿校验」，将在活体校验环节后，通过语音朗读的模式校验用户真实意愿，请您确认“意愿核身（人脸实名认证）”、“意愿核身（人脸对比）”接口已开通付费。
- **意愿核验次数**：在意愿核验阶段，用户最多可以进行意愿核验的次数，防止用户恶意调用，保障客户权益。
- **意愿核验时长**：在意愿核验阶段，停留的最长时间，超过该时长会报警提示检测超时。



4.5 认证结果配置

通过上传「姓名+身份证号+人脸图片」三要素到人脸实名认证接口，通过调取权威库判断是否为本人操作；或者上传「自传图片+人脸图片」到人脸对比接口，获取2张图片的相似度分数判断是否为本人操作。

> - **提供的认证结果页面**：您可以选择是否使用百度提供的展示给用户认证结果的页面，若您选择自己开发，可选择不使用。
> - **认证未通过时URL是否失效**：当用户认证未通过时，部分用户会出现多次重新请求验证的情况，您可通过此项配置控制用户认证数。
> - **用户认证错误次数**：URL未失效时，用户最多可以进行活体检测的次数，防止用户恶意调用，保障客户权益。
> - **语音播报**：用户进行活体检测时，进行语音播报所做的动作。
> - **阈值**：此阈值设置的是用户上传图片与权威库图片进行比对后得分的阈值，高于此阈值即判断为用户本人。**阈值设置推荐为80**，您可通过实际业务场景继续调整。

4.6 意愿核身降级活体方案配置

- **意愿核身默认【禁止降级，不兼容时中断流程跳转失败结果页】**：当选择此选项时，因浏览器、手机系统以及APP内核等环境因素导致实时活体检测不兼容时，当前核验流程结束，自动跳转至方案结果页。

Step5：提交方案，扫码预览

方案配置完成后，点击提交按钮，进入方案管理页面，鼠标点击「查看二维码」即可显示二维码信息，扫码即可体验H5意愿核身流程，如下图所示。

> 以上体验流程仅当身份信息录入方式选择**身份证识别OCR**时生效，选择**手动输入、业务调用时传入身份信息**时则无法体验。 请谨慎修改上述H5方案配置，在点击「提交」后会实时对方案配置进行更新。若您需要修改方案，请在不影响线上业务的情况下进行调整。

二、 方案接入

Step1：获取token

通过获取Token接口获取verify_token信息

Step2: 确认用户信息上报方式

1. 如果选择用户手动输入或者拍照上传的方式，则跳过Step2，直接进入Step3
2. 如果选择通过API的形式传输用户信息，不需要用户手动输入，则通过指定用户信息上报接口上传对应的接口。

Step3：跳转实名认证H5 URL，用户进行操作

业务H5网页通过获取Token接口返回的verify_token信息请求认证H5页面，进行用户端流程操作。
认证URL： https://brain.baidu.com/face/print/?token=xxx&successUrl=https://xxx&failedUrl=https://xxx%3CBr%3E

参数	含义	备注
token	填写verify_token，verify_token获取参考获取verify_token参考文档	
callbackUrl	通用跳转的网址，不区分成功与失败，用户通过token查询结果，callbackUrl与successUrl/failedUrl互斥（配置后successUrl、failedUrl不生效），网址需要加http/https前缀	如果流程回调需要携带token，建议传递https://www.callbackurl.com/xxxxxx?token=xxxx，核验结束后会直接回调该地址（预计2024年7月24日生效）
successUrl	请求成功跳转的网址，网址需要加http/https前缀	如果流程回调需要携带token，建议传递https://www.successurl.com/xxxxxx?token=xxxx，成功后会直接回调该地址
failedUrl	请求失败跳转的网址，网址需要加http/https前缀	如果流程回调需要携带token，建议传递https://www.failedUrl.com/xxxxxx?token=xxxx, 失败后直接回调该地址

successUrl和failedUrl推荐使用encodeURIComponent进行转义，不然可能无法正确跳转，转义示例如下：
`encodeURIComponent("https://ai.baidu.com")`

Step4：获取认证结果及资料，返回用户认证信息

在用户完成认证（成功或者失败）后，我们会分别回调successUrl和failedUrl，传递?token=xxx参数，业务上可以根据token字段获取对应的数据，其中包括比对分数、图片等数据均需要通过后查询的接口进行查询。可以参考获取认证人脸接口文档、查询认证结果接口文档、查询统计结果接口文档查询用户人脸实名认证流程的相关信息，以进行后续业务集成开发。

三、 自有APP内嵌H5页面

如您需要在自有APP渠道中，通过webview方式内嵌H5页面，使用人脸实名认证H5方案。

我们为您提供了兼容性配置方法及其所需组件，以减少您可能遇到的兼容性问题，详细配置方法请参见APP内嵌H5兼容性配置文档。

如您使用该场景，为了更好地配合组件使用，请在您的认证URL后，增加参数&useNative=1，以使用js-bridge通信功能。如：https://brain.baidu.com/face/print/?token=xxx&successUrl=https://xxx&failedUrl=https://xxx%3CBr%3E&useNative=1

意愿核身方案配套接口

意愿核身（权威库）

- **获取verify_token+上传意愿核验文本信息**：意愿核身流程开始，初始化获取唯一的流程verify_token，同时上传意愿核验需要的朗读问题/答案列表文本
- **指定用户信息上报**：意愿核身流程中，提前上传用于权威库验证时的用户身份证号码、姓名，实现**无需用户拍照或手动输入**，直接进行活体及识别

意愿核身（自传图片）

- **获取verify_token+上传意愿核验文本信息**：意愿核身流程开始，初始化获取唯一的流程verify_token，同时上传意愿核验需要的朗读问题/答案列表文本
- **对比图片上传**：意愿核身流程中，提前上传用于1:1人脸比对的人脸底图

意愿核身结果查询

- **获取认证人脸**：获取认证成功最终采集的人脸信息
- **查询认证结果**：获取身份证信息（仅在使用OCR的情况下返回）、活体检测分数、人脸相似度得分、意愿核验结果
- **实时方案视频获取**：获取实时活体检测过程的视频/图片
- **核验及计费信息获取**：获取对应verify_token下的所有核验信息（人脸、身份信息、认证结果、意愿核验结果），以及后端接口「**意愿核身（权威库）**」、「**意愿核身（自传图片）**」的计费与否及计费时间

一、方案功能接口

1.获取verify_token+上传意愿核验文本信息接口

本接口为意愿核身方案的verify_token获取接口，利用所获取的verify_token进行意愿核身流程的有效期为**2小时**。

在线调试 您可以在 示例代码中心 中调试该接口，可进行签名验证、查看在线调用的请求内容和返回结果、示例代码的自动生成。

调用方式

请求URL数据格式

向API服务地址使用POST发送请求，必须在URL中带上参数`access_token`，可通过后台的API Key和Secret Key生成，具体方式请参考“Access Token获取”。

> **注意**：`access_token`的有效期为30天，**切记需要每30天进行定期更换，或者每次请求都拉取新token**；

POST中Body的参数，按照下方请求参数说明选择即可。

> **提示**：如果您为百度云老用户，正在使用其他非AI的服务，可以参考百度云AKSK鉴权方式发送请求，虽然**请求方式**和**鉴权方法**和本文所介绍的不同，但请求参数和返回结果一致。

请求说明

注意事项：

- **请求体格式化**：Content-Type为`application/json`，通过`json`格式化请求体。

请求示例

HTTP方法：POST

请求URL： https://aip.baidubce.com/rpc/2.0/brain/solution/faceprint/verifyToken/generate

URL参数：

参数	值
access_token	通过API Key和Secret Key获取的access_token,参考“Access Token获取”

Header：

参数	值
Content-Type	application/json

Body中放置请求参数，参数详情如下：

请求参数：

参数	必选	类型	说明
plan_id	是	string	方案的id信息，请在人脸实名认证控制台查看创建的H5意愿核身方案的方案ID信息
will_verify	是	object	意愿核验相关参数
+ verify_text	是	string	意愿核验系统播报问题文本内容，建议控制文本长度在50字以内。如「请确认本次业务是您本人自愿办理吗？请回答“是的，我确认”」
+ verify_answer	是	string	意愿核验回答问题答案，如「是的，我确认」
+ spd	否	string	问题文本播报语速，取值0-15，默认为5中语速，数值越高语速越快
+ pit	否	string	问题文本播报音调，取值0-15，默认为5中语调，数值越高语调越高
+ vol	否	string	问题文本播报音量，基础音库取值0-9，默认为5中音量（取值为0时为音量最小值，并非为无声）

请求示例：

```
{
 "plan_id": 21012,
 "will_verify": {
    "verify_text": "请问您本次业务是本人自愿办理吗",
    "verify_answer": "是的，我确认",
    "spd": "7", //通过传参适当调快语速
    "pit": "6", //通过传参适当调整音调
    "vol": "8" //通过传参适当调高音量
 }
}
```

返回参数

- 返回结果

字段	必选	类型	说明
success	是	boolean	返回请求是否成功信息 若请求成功返回ture 请求失败则返回false
result	是	object	请求结果
+verify_token	是	string	请求获取的verify_token
log_id	是	string	本次调用的日志id

- 返回示例

```
{
  "success": true,
  "result": {
     "verify_token": "Yz9rWITm4vak16PBAh5x8oG7"
  },
  "log_id": "1814798895"
}
```

2.指定用户信息上报接口

本接口用于，前端在方案中选择身份信息录入-身份信息录入方式-指定用户身份核验时，需要先调用此接口输入指定用户的姓名+身份证号信息，再请求url跳转页面。

在线调试 您可以在 示例代码中心 中调试该接口，可进行签名验证、查看在线调用的请求内容和返回结果、示例代码的自动生成。 **调用方式**

请求URL数据格式

向API服务地址使用POST发送请求，必须在URL中带上参数`access_token`，可通过后台的API Key和Secret Key生成，具体方式请参考“Access Token获取”。

> **注意**：`access_token`的有效期为30天，**切记需要每30天进行定期更换，或者每次请求都拉取新token**；

POST中Body的参数，按照下方请求参数说明选择即可。

> **提示**：如果您为百度云老用户，正在使用其他非AI的服务，可以参考百度云AKSK鉴权方式发送请求，虽然**请求方式**和**鉴权方法**和本文所介绍的不同，但请求参数和返回结果一致。

请求说明

注意事项：

- **请求体格式化**：Content-Type为`application/json`，通过`json`格式化请求体。

请求示例

HTTP方法：`POST`

请求URL：https://aip.baidubce.com/rpc/2.0/brain/solution/faceprint/idcard/submit

URL参数：

参数	值
access_token	通过API Key和Secret Key获取的access_token,参考“Access Token获取”

Header：

参数	值
Content-Type	application/json

Body中放置请求参数，参数详情如下：

参数	必选	类型	值
verify_token	是	string	通过access_token获取的verify_token
id_name	是	string	指定输入用户的姓名信息
id_no	是	string	指定输入用户的身份证件号信息
certificate_type	否	int	证件类型： 0 大陆居民二代身份证 1 港澳台居民来往内地通行证 2 外国人永久居留证 3 定居国外的中国公民护照 4 港澳台居民居住证 5 港澳居民来往内地通行证（非中国籍）

请求示例：

```
{
  "verify_token": "2sF3nE5mXOHkx2aQwWG4n5WI",
  "id_name": "张三",
  "id_no": "500***********3390",
  "certificate_type": 0
}
```

返回参数

- 返回结果

字段	必选	类型	说明
success	是	boolean	返回请求是否成功信息。 若请求成功返回ture; 请求失败则返回false
result	是	int	请求结果，返回固定结果1，可忽略
log_id	是	string	本次调用的日志id

- 返回示例

```
{
  "success": true,
  "result": 1,
  "log_id": "1244068892"
}
```

3.对比图片上传接口

本接口用于，前端在方案中选择比对源选择-自建人脸库比对时，需要先调用此接口上传待比对的自建人脸库中的指定人脸图

片，再请求url跳转页面。

在线调试 您可以在 示例代码中心 中调试该接口，可进行签名验证、查看在线调用的请求内容和返回结果、示例代码的自动生成。 **调用方式**

请求URL数据格式

向API服务地址使用POST发送请求，必须在URL中带上参数access_token，可通过后台的API Key和Secret Key生成，具体方式请参考“Access Token获取”。

> **注意**：access_token的有效期为30天，**切记需要每30天进行定期更换，或者每次请求都拉取新token**；

POST中Body的参数，按照下方请求参数说明选择即可。

> **提示**：如果您为百度云老用户，正在使用其他非AI的服务，可以参考百度云AKSK鉴权方式发送请求，虽然**请求方式**和**鉴权方法**和本文所介绍的不同，但请求参数和返回结果一致。

请求说明

注意事项：

- **请求体格式化**：Content-Type为application/json，通过json格式化请求体。

请求示例

HTTP方法：POST

请求URL： https://aip.baidubce.com/rpc/2.0/brain/solution/faceprint/uploadMatchImage

URL参数：

参数	值
access_token	通过API Key和Secret Key获取的access_token,参考“Access Token获取”

Header：

参数	值
Content-Type	application/json

Body中放置请求参数，参数详情如下：

参数	必选	类型	值
verify_token	是	string	通过access_token获取的verify_token
image	是	string	图片base64字符串，编码后的图片大小不超过10M，图片分辨率小于1920*1080
quality_control	否	string	“NONE”、“LOW”、“NORMAL”、“HIGH”； 质量控制参数，未主动传入时默认为“NONE”
liveness_control	否	string	“NONE”、“LOW”、“NORMAL”、“HIGH”； 活体控制参数，未主动传入时默认为“NONE”

请求示例：

```
{
  "verify_token": "2sF3nE5mXOHkx2aQwWG4n5WI",
  "image": "/9j/4AAQSkZJRgABAQAAAQABAAD/",
  "quality_control" : "LOW",
  "liveness_control" : "LOW",
}
```

返回参数

- 返回结果

字段	必选	类型	说明
success	是	boolean	返回请求是否成功信息。 若请求成功返回ture; 请求失败则返回false
result	是	object	返回结果
log_id	是	string	本次调用的日志id

- 返回示例

```
{
 "success": true,
 "result": null,
 "log_id": "1329130892"
}
```

二、验证后查询接口

> 获取verify_token后，请先按照跳转实名认证H5 URL，用户进行操作后再查询接口，否则生成verify_token无法生效,无法查询到结果。

1.获取认证人脸接口

本接口返回进行意愿核身过程中进行认证的最终采集的人脸信息。根据Verify_token返回的结果信息会在云端保留**3天**，您可根据需要在此期间进行调取查询。

在线调试 您可以在 示例代码中心 中调试该接口，可进行签名验证、查看在线调用的请求内容和返回结果、示例代码的自动生成。

调用方式

请求URL数据格式

向API服务地址使用POST发送请求，必须在URL中带上参数`access_token`，可通过后台的API Key和Secret Key生成，具体方式请参考“Access Token获取”。

> **注意**：`access_token`的有效期为30天，**切记需要每30天进行定期更换，或者每次请求都拉取新token**；

POST中Body的参数，按照下方请求参数说明选择即可。

> **提示**：如果您为百度云老用户，正在使用其他非AI的服务，可以参考百度云AKSK鉴权方式发送请求，虽然**请求方式**和**鉴权方法**和本文所介绍的不同，但请求参数和返回结果一致。

请求说明

注意事项：

- **请求体格式化**：Content-Type为`application/json`，通过`json`格式化请求体。

请求示例

HTTP方法：`POST`

请求URL：`https://aip.baidubce.com/rpc/2.0/brain/solution/faceprint/result/simple`

URL参数：

参数	值
access_token	通过API Key和Secret Key获取的access_token,参考“Access Token获取”

Header：

参数	值
Content-Type	application/json

Body中放置请求参数，参数详情如下：

参数	值
verify_token	通过access_token获取的verify_token

请求示例：

```
 {
 "verify_token" : "clupeyP51sn28XzxGVTfYqoN"
}
```

返回参数

- 返回结果

字段	必选	类型	说明
success	是	boolean	返回请求是否成功信息。 若请求成功返回ture; 请求失败则返回false
result	是	object	请求结果
+image	是	string	返回采集的用户人脸信息
log_id	是	string	本次调用的日志id

- 返回示例

```
{
  "success": true,
  "result": {
     "image":"https://brain.baidu.com/solution/faceprint/image/query?verify_token=xxxxxx"
  },
  "log_id": "1054986003"
}
```

2.查询认证结果接口（包含意愿核验结果）

本接口为请求返回的认证结果信息查询，包含身份证OCR识别信息、用户二次确认的身份证信息，活体检测信息、及用户对权威库图片进行比对的分数信息。（仅在认证成功时返回上述信息，认证失败返回错误码）根据Verify_token返回的结果信息会在云端保留**3天**，您可根据需要在此期间进行调取查询。

在线调试 您可以在 示例代码中心 中调试该接口，可进行签名验证、查看在线调用的请求内容和返回结果、示例代码的自动生成。 **调用方式**

请求URL数据格式

向API服务地址使用POST发送请求，必须在URL中带上参数`access_token`，可通过后台的API Key和Secret Key生成，具体方式请参考“Access Token获取”。

> **注意**：`access_token`的有效期为30天，**切记需要每30天进行定期更换，或者每次请求都拉取新token**；

POST中Body的参数，按照下方请求参数说明选择即可。

> **提示**：如果您为百度云老用户，正在使用其他非AI的服务，可以参考百度云AKSK鉴权方式发送请求，虽然**请求方式**和**鉴权方法**和本文所介绍的不同，但请求参数和返回结果一致。

请求说明

注意事项：

- **请求体格式化**：Content-Type为`application/json`，通过json格式化请求体。

请求示例

HTTP方法：`POST`

请求URL： `https://aip.baidubce.com/rpc/2.0/brain/solution/faceprint/result/detail`

URL参数：

参数	值
access_token	通过API Key和Secret Key获取的access_token,参考“Access Token获取”

Header：

参数	值
Content-Type	application/json

Body中放置请求参数，参数详情如下：

参数	值
verify_token	通过access_token获取的verify_token

请求示例：

```
 {
 "verify_token"："clupeyP51sn28XzxGVTfYqoN"
}
```

返回参数

- 返回结果

字段	必选	类型	说明
success	是	boolean	返回本次身份核验是否成功的结果。 成功返回ture; 失败则返回false
result	是	object	请求结果
+idcard_ocr_result	否	object	返回采集的身份证信息 当人脸实名认证控制台设置为使用OCR识别时返回此参数信息
++address	否	string	地址
++birthday	否	string	生日
++name	否	string	姓名
++id_card_number	否	string	身份证号
++gender	否	string	性别
++nation	否	string	民族
++expire_time	否	string	身份证失效日期
++issue_authority	否	string	身份证签发机关
++issue_time	否	string	身份证生效日期
+idcard_images	否	object	返回采集的身份证图片信息 当人脸实名认证控制台设置为使用OCR识别时返回此参数信息
++front_base64	否	string	身份证图片的正面信息
++back_base64	否	string	身份证图片的反面信息 当人脸实名认证控制台设置为使用OCR识别且为国徽面+人像面时返回此参数信息
+verify_result	是	object	认证返还信息
++liveness_score	是	float	**活体检测分数：** 活体验证通过时返回活体分数，不通过则返回0。
++score	是	float	意愿核身（权威库）、意愿核身（自传图片）的比对得分**（仅活体检测时返回为0）**
++spoofing	是	float	合成图分数 若未进行合成图检测，则返回0 若进行活体检测，则返回合成图检测分值
+idcard_confirm	是	object	用户二次确认的身份证信息
++name	是	string	姓名
++idcard_number	是	string	身份证号
+will_verify	是	object	意愿核身方案的核验结果
++asr_result	是	string	意愿核身方案的音频实时识别结果
++verify_status	是	string	意愿核身方案的核验结果状态
++audio	是	string	意愿核身方案的音频下载链接
++screenshot_url	是	string	意愿核身方案中的屏幕截图地址
log_id	是	string	本次调用的日志id

- 返回示例

```
{
 "success": true,
 "result": {
    "verify_result": {
        "score": 93.7835,
        "liveness_score": 0.9672966,
        "spoofing": 0.0
    },
    "idcard_ocr_result": {
        "birthday": "19960216",
        "issue_authority": "胶南市公安局",
        "address": "山东省**********",
        "gender": "女",
        "nation": "汉",
        "expire_time": "20221103",
        "name": "柴*",
        "issue_time": "20121103",
        "id_card_number": "370**********5826"
    },
    "idcard_images": {
        "front_base64": "/9j/4AAQSkZJRgAB....",
        "back_base64": "/9j/4AAQSkZJRgAB...."
    },
    "idcard_confirm": {
        "idcard_number": "370**********5826",
        "name": "柴*"
    },
    "will_verify": {
        "asr_result": "是的我确认",
        "verify_status": "0",
        "audio": "https://bj.bcebos.com",
        "screenshot_url": "https://bj.bcebos.com"
    }
 },
 "log_id": "160931948204246"
}
```


3.查询统计结果

根据Verify_token返回的结果信息会在云端保留**3天**，您可根据需要在此期间进行调取查询。

在线调试 您可以在 示例代码中心 中调试该接口，可进行签名验证、查看在线调用的请求内容和返回结果、示例代码的自动生成。

调用方式

请求URL数据格式

向API服务地址使用POST发送请求，必须在URL中带上参数`access_token`，可通过后台的API Key和Secret Key生成，具体方式请参考“Access Token获取”。

> **注意**：`access_token`的有效期为30天，**切记需要每30天进行定期更换，或者每次请求都拉取新token**；

POST中Body的参数，按照下方请求参数说明选择即可。

> **提示**：如果您为百度云老用户，正在使用其他非AI的服务，可以参考百度云AKSK鉴权方式发送请求，虽然**请求方式**和**鉴权方法**和本文所介绍的不同，但请求参数和返回结果一致。

请求说明

注意事项：

- **请求体格式化**：Content-Type为`application/json`，通过`json`格式化请求体。

请求示例

HTTP方法：`POST`

请求URL： `https://aip.baidubce.com/rpc/2.0/brain/solution/faceprint/result/stat`

URL参数：

参数	值
access_token	通过API Key和Secret Key获取的access_token,参考“Access Token获取”

Header：

参数	值
Content-Type	application/json

Body中放置请求参数，参数详情如下：

参数	值
verify_token	通过access_token获取的verify_token

请求示例：

```
 {
 "verify_token"："clupeyP51sn28XzxGVTfYqoN"
}
```

返回参数

- 返回结果

字段	必选	类型	说明
success	是	boolean	返回请求是否成功信息。 若请求成功返回ture; 请求失败则返回false
result	是	object	请求结果
+Verify_FIN	是	array	意愿核身2个接口请求的统计信息
++error_code	是	string	错误码编号 若为0则表示请求成功
++count	是	int	当前错误码的请求数量
+Sessioncode	是	array	随机校验码接口请求的统计信息
++error_code	是	string	错误码编号 若为0则表示请求成功
++count	是	int	当前错误码的请求数量
+OCR	是	array	OCR身份证识别接口请求的统计信息
++error_code	是	string	错误码编号 若为0则表示请求成功
++count	是	int	当前错误码的请求数量
+verify	是	array	意愿核身2个接口请求的统计信息
++error_code	是	string	错误码编号 若为0则表示请求成功
++count	是	int	当前错误码的请求数量
+VideoLiveness	是	array	视频活体检测接口请求的统计信息
++error_code	是	string	错误码编号 若为0则表示请求成功
++count	是	string	当前错误码的请求数量
+Livenesscolorful	是	array	炫瞳活体检测接口请求的统计信息
++error_code	是	string	错误码编号 若为0则表示请求成功
++count	是	string	当前错误码的请求数量
+VerifySec	是	array	意愿核身2个接口请求的统计信息
++error_code	是	string	错误码编号 若为0则表示请求成功
++count	是	string	当前错误码的请求数量
log_id	是	string	本次调用的日志id

- 返回示例

```
{
  "success": true,
  "result": {
     "Verify_FIN": [
        {
           "error_code": 0,
           "count": 1
        }
     ],
      "Sessioncode": [
        {
           "error_code": 0,
           "count": 1
        }
     ],
     "OCR": [
        {
           "error_code": 0,
           "count": 2
        }
     ],
     "verify": [
        {
           "error_code": 0,
           "count": 1
        }
     ],
     "VideoLiveness": [
        {
           "error_code": 0,
           "count": 1
        }
     ]
     "Livenesscolorful": [
        {
           "error_code": 0,
           "count": 1
        }
     ]
     "VerifySec": [
        {
           "error_code": 0,
           "count": 1
        }
     ]
  },
  "log_id": "1405335905"
}
```


4.实时方案视频获取

根据Verify_token返回的视频url会在云端保留1天，您可根据需要在此期间进行调取查询。

注意：H5方案中的视频录制功能默认不开启，调用本接口将无法获得视频数据，如您需要使用以上api，请联系商务经理或提交工单申请配置开通

调用方式

请求URL数据格式

向API服务地址使用POST发送请求，必须在URL中带上参数`access_token`，可通过后台的API Key和Secret Key生成，具体方式

请参考“Access Token获取”。

> **注意**：access_token的有效期为30天，**切记需要每30天进行定期更换，或者每次请求都拉取新token**；

POST中Body的参数，按照下方请求参数说明选择即可。

> **提示**：如果您为百度云老用户，正在使用其他非AI的服务，可以参考百度云AKSK鉴权方式发送请求，虽然**请求方式**和**鉴权方法**和本文所介绍的不同，但请求参数和返回结果一致。

在线调试 您可以在 示例代码中心 中调试该接口，可进行签名验证、查看在线调用的请求内容和返回结果、示例代码的自动生成。

请求说明

注意事项：

- **请求体格式化**：Content-Type为application/json，通过json格式化请求体。

请求示例

HTTP方法：POST

请求URL： https://aip.baidubce.com/rpc/2.0/brain/solution/faceprint/result/media/query

URL参数：

参数	值
access_token	通过API Key和Secret Key获取的access_token,参考“Access Token获取”

Header：

参数	值
Content-Type	application/json

Body中放置请求参数，参数详情如下：

参数	值
verify_token	通过access_token获取的verify_token

请求示例：

```
 {
 "verify_token"："clupeyP51sn28XzxGVTfYqoN"
}
```

返回参数

- 返回结果

字段	必选	类型	说明
success	是	boolean	返回请求是否成功信息。 若请求成功返回ture; 请求失败则返回false
result	是	object	视频url链接
+processVideo	是	array	实时核验视频链接，第一个为人脸核验视频，第二个是意愿核验视频
+video	是	array	意愿核验方案不使用该字段
+images	是	array	图片链接数组，H5实时炫瞳活体、H5实时动作活体、H5实时静默活体方案所返回的4张人脸图片
+extInfo	是	string	扩展信息，如有则代表处理视频中的错误信息
log_id	是	string	本次调用的日志id

- 返回示例

```
{
 "success": true,
 "result": {
    "processVideo": [
       "http://bj.bcebos.com/v1/aa.mp4"
    ],
    "video": [
       "http://bj.bcebos.com/v1/aa.mp4"
    ],
    "images": [
       "http://bj.bcebos.com/v1/aa.jpg"
    ],
    "extInfo": null
 },
 "log_id": "1534116864874049322"
}
```

5.核验及计费信息获取（包含意愿核验结果）

根据Verify_token返回的信息会在云端保留**3天**，您可根据需要在此期间进行调取查询。

调用方式

请求URL数据格式

向API服务地址使用POST发送请求，必须在URL中带上参数`access_token`，可通过后台的API Key和Secret Key生成，具体方式请参考“Access Token获取”。

> **注意**：`access_token`的有效期为30天，**切记需要每30天进行定期更换，或者每次请求都拉取新token**；

POST中Body的参数，按照下方请求参数说明选择即可。

> **提示**：如果您为百度云老用户，正在使用其他非AI的服务，可以参考百度云AKSK鉴权方式发送请求，虽然**请求方式**和**鉴权方法**和本文所介绍的不同，但请求参数和返回结果一致。

在线调试 您可以在 示例代码中心 中调试该接口，可进行签名验证、查看在线调用的请求内容和返回结果、示例代码的自动生成。

请求说明

注意事项：

- **请求体格式化**：Content-Type为`application/json`，通过json格式化请求体。

请求示例

HTTP方法：`POST`

请求URL： `https://aip.baidubce.com/rpc/2.0/brain/solution/faceprint/result/getall`

URL参数：

参数	值
access_token	通过API Key和Secret Key获取的access_token,参考“Access Token获取”

Header：

参数	值
Content-Type	application/json

Body中放置请求参数，参数详情如下：

参数	值
verify_token	通过access_token获取的verify_token

请求示例：

```
 {
 "verify_token"："clupeyP51sn28XzxGVTfYqoN"
}
```

返回参数

- 返回结果

字段	必选	类型	说明
success	是	boolean	返回请求是否成功信息。 若请求成功返回ture; 请求失败则返回false
log_id	是	string	本次查询接口请求的logid
result	是	object	结果信息
+verify_count	是	int	当前verify_token对应的核验次数，核验次数为几，后面就有几个对象数组 如方案配置为“认证未通过时URL失效”，则该verify_token核验次数为1； 如方案配置为“认证未通过时URL不失效”，则该verify_token对应的核验次数可能大于1（用户核验失败后多次重试）
+ocr_charge_count	是	int	身份证识别OCR接口的计费次数
+match_charge_count	是	int	意愿核身（自传图片）人的计费次数 如方案配置为“认证未通过时URL失效”，计费次数最多为1； 如方案配置为“认证未通过时URL不失效”，则计费次数可能大于1（用户核验失败后多次重试）；

+verify_charge_count	是	int	意愿核身（权威库）的计费次数 如方案配置为“认证未通过时URL失效”，计费次数最多为1； 如方案配置为“认证未通过时URL不失效”，则计费次数可能大于1（用户核验失败后多次重试）；
+will_verify	是	object	意愿核身结果
++audio	是	object	音频下载链接
++screenshot_url	是	object	屏幕截图地址
++asr_result	是	object	音频实时识别结果
++verify_status	是	object	意愿核验结果状态
+verify_result	是	object[]	核验结果信息，verify_count的值为几，就返回几个该对象
++order	是	int	代表本次核验结果在verify_token所有核验次数中的顺序，按时间先后排序，第一次核验返回值为 1，第二次核验返回值为 2，以此类推
++is_verify_passed	是	boolean	本次核验是否通过 核验成功返回true 核验失败返回false
++code	是	string	本次核验的错误码 核验成功时返回 0 核验失败返回非0错误码
++message	是	string	本次核验的错误信息 核验成功时返回“核验成功” 核验失败时返回具体错误原因，如“身份证姓名不匹配”
++verify_time	是	string	本次核验完成的时间点，包含年月日、时分秒，如“2023-6-20 18:00:00”
++is_charged	是	boolean	本次核验是否计费 计费返回true 不计费返回false
++charge_type	是	string	计费类型：**verify**(意愿核身（权威库）)、**match**(意愿核身（自传图片）)
++charge_time	是	string	本次计费的时间点，如不计费则该字段为空，包含年月日、时分秒，如“2023-6-20 18:00:00”
++is_ocr_charge	否	boolean	身份证识别OCR是否计费 计费返回true 不计费返回false
++ocr_charge_time	否	string	ocr 收费时间点，如不收费则该字段为空，包含年月日、时分秒，如“2023-6-20 18:00:00”
++ocr_count	否	int	ocr次数
++verify_detail	是	object	核验的详细信息
+++face_image	是	string	本次核验流程中采集的人脸最佳质量图下载url
+++verify_log_id	是	string	本次核验流程中请求的意愿核身（权威库）、意愿核身（自传图片）2个后端接口的logid 可在记录查询平台中通过logid查询到3天内的记录。 少数核验失败情况下，实际并未发生上述俩接口的请求，则该字段为空，如：用户拒绝摄像头授权
+++score	是	float	人脸相似度得分，大于等于方案设置阈值为同一人；当身份证姓名不匹配或活体不通过时，该项为空

+++threshold	是	float	本次核验时，所使用的方案配置相似度阈值
+++liveness_score	是	float	活体检测得分，注：预留功能字段，当前返回为0
+++spoofing_score	否	float	方案配置使用合成图功能，将返回合成图得分，注：预留功能字段，当前返回为0
+++risk_level	否	string	安全风控等级 方案配置启用安全风控，将返回该字段 值为1或2时表示触发了安全风险，值为3或4时无风险
+++risk_tag	否	string	安全风控标签 方案配置启用安全风控，将返回该字段 当risk_level值为1或2时，返回具体风险类型，risk_level值为3或4时，为空
+++is_demote	否	boolean	方案配置为实时检测时，将返回该字段，代表本次核验是否出现降级 降级返回true，未降级返回false 注：当环境不支持实时检测，且方案配置允许降级时，将降级为录制视频上传
++idcard_confirm	是	object	用户手动输入或二次确认的身份证信息
+++name	是	string	姓名
+++idcard_number	是	string	证件号
+++idcard_type	是	string	证件类型，大陆居民二代身份证返回0
++idcard_ocr_result	否	object	OCR采集的身份证信息，当方案配置使用OCR采集证件照时返回该参数
+++address	否	string	地址
+++birthday	否	string	生日
+++name	否	string	姓名
+++idcard_number	否	string	证件号
+++idcard_type	否	string	证件类型，大陆居民二代身份证返回0
+++gender	否	string	性别
+++nation	否	string	民族
+++issue_time	否	string	身份证生效日期
+++expire_time	否	string	身份证失效日期
+++issue_authority	否	string	身份证签发机关
++idcard_images	否	object	返回采集的身份证图片信息 当人脸实名认证控制台设置为使用OCR识别时返回此参数信息
+++front_image	否	string	身份证正面图片的下载url
+++back_image	否	string	身份证背面图片的下载url

- 返回示例

```
{
  "success": true,
  "result": {
    "verify_count": 1,
    "ocr_charge_count": 0,
    "match_charge_count": 0,
    "verify_charge_count": 1,
    "live_charge_count": 0,
    "verify_result": [
      {
        "order": 1,
        "code": "0",
        "message": "核验成功",
        "is_verify_passed": true,
        "verify_time": "2025-02-10 14:55:19",
        "is_charged": true,
        "charge_type": "verify",
        "charge_time": "2025-02-10 14:55:19",
        "is_ocr_charge": false,
        "ocr_count": 0,
        "ocr_charge_time": null,
        "verify_detail": {
          "score": 99.5199966430664,
          "threshold": 80.0,
          "face_image": "https://bj.bcebos.com",
          "verify_log_id": "xxx",
          "liveness_score": 0.99934685,
          "spoofing_score": 0.99934685,
          "risk_level": 3,
          "risk_tag": "xxx",
          "is_demote": false
        },
        "idcard_confirm": {
          "name": "***",
          "idcard_number": "xxx",
          "idcard_type": "0",
          "idcard_ocr_result": null,
          "idcard_images": null
        }
      }
    ],
    "will_verify": [
      {
        "audio": "https://bj.bcebos.com",
        "screenshot_url": "https://bj.bcebos.com",
        "asr_result": "愿意",
        "verify_status": "0"
      }
    ]
  },
  "log_id": "1889935299719517067"
}
```


三、常见问题

1. verify_token的详细说明

（1）access_token 有效期为 30 天，重复生成 access_token 的话，对之前的不影响，依旧能使用；access_token 与 verify_token 是包含关系，即 verify_token 是由 access_token生成的，如果一个 verify_token 是和一个不匹配的 access_token 使用，会提示“Token无效或者已过期”

（2）verify_token 的核验有效期为 2 小时，在有效期内可以进行了核验动作，如果超过了 2 小时没有使用该 token 进行核验的话会提示“Token无效或者已过期”

（3） verify_token 核验完毕后（核验成功，或设置了认证未通过时URL失效），无法再进行核验，如果再次进行核验，会提示“Token无效或者已过期”

（4） verify_token 核验完毕后（核验成功，或设置了认证未通过时URL失效），支持对后验接口的查询，有效期均为 3 天，3 天后再次查询后验接口，会提示“Token无效或者已过期”

2. 获取认证人脸、查询认证结果、核验及计费信息获取接口持续返回错误码18，提示openapi限制，且有返回logid

答：检查意愿核身方案依赖的“意愿核身（权威库）”、“意愿核身（自传图片）”两项付费接口，是否有免费额度，或者直接开通后付费，以消除该报错提示。开通后，再进行重试。

私有化部署

产品介绍

整体介绍

人脸私有化部署即可以支持部署到本地服务器的纯软件服务，也提供与GPU服务器组合的一体机全包服务，提供人脸检测与属性分析、人脸比对、人脸搜索、活体检测等基础功能，支持百万级超大型人脸库，毫秒级输出搜索结果。

提供分布式高并发部署方案，满足无网、弱网、专网等多种网络需求，适用于**楼宇监控、校园安防**等数据私密性要求较高的场景以及**地铁通行**等超大人脸库、高并发场景。

此软件包部署在开发者本地的服务器上，能够得到与在线API功能完全相同的接口（参数有少量区别，请参考接口文档）

如果您想将您的服务从公有云转移到本地化服务器请参考公有云转私有化文档。

如果您对本文档有任何疑问，欢迎在百度云控制台提交工单。

产品介绍

人脸识别私有化-通用版 人脸识别私有化部署包提供人脸检测与属性分析、人脸比对、人脸搜索、活体检测等基础功能，支持百万级超大型人脸库，可实现毫秒级响应。在产品性能方面，可以可根据自己的业务需求及显卡类型灵活选择授权QPS。

多场景模型

适用于人脸多种场景（视频监控、门禁通行、身份验证等），提供人脸检测、人脸比对、人脸搜索、活体检测等基础功能，支持GPU、CPU的部署方式

人脸私有化部署包支持本地服务器部署（包括单机部署、多机部署、集成部署等）及专有云服务器部署，您可以根据实际业务场景进行灵活选择。

> 注：
> 1、若您选取百度云GPU服务器的部署方式可以提交工单申请百度工作人员进行部署支持，服务器地区请选择**保定或苏州**
> 2、若您想获取免费测试/正式部署包，您可以到在百度云控制台发起人脸私有化部署包申请
> 关于价格方面的疑问请提交商务咨询

人脸识别私有化-定制版

人脸识别服务器一体机，提供搭载了人脸识别模型的GPU服务器，支持多种GPU型号可供选择。百度技术人员可上门安装交付，部署完成后即可在本地进行业务集成。

想了解定制版产品价格请提交商务咨询

部署方式

本地服务器部署

支持在本地进行单机部署、多机部署、集群部署等方式，提供一键部署工具，快速安装运行环境、容器及模型服务，最快半小时即可完成安装部署。

支持百万级超大型人脸库，可实现毫秒级响应。

具体部署细节请参考安装部署文档

专有云服务器部署

百度人脸私有化部署支持在专有云服务器上进行私有化部署，购买云端服务器和人脸识别私有化部署包授权后，可申请百度工作人员进行环境及服务的安装部署，您只需根据接口进行封装开发即可。

联合百度云GPU服务器购买人脸识别模型，服务器最低可享6折优惠，点此可立即咨询。

服务器地区请选择**保定或苏州**，若您需要其他地区百度云GPU服务器，请提交工单联系百度的工作人员

产品功能

1、人脸检测与属性分析

检测图中的人脸并返回人脸框。检测出人脸后，可对人脸进行分析，获得眼、口、鼻轮廓等72个关键点定位准确识别多种人脸属性，如性别、年龄、表情、情绪等信息。

该技术可适应大角度侧脸，遮挡，模糊，表情变化等各种实际环境。

具体人脸检测及属性分析可参考人脸检测接口文档

2、活体检测

- **在线图片活体检测** 基于图片中人像的破绽（摩尔纹、成像畸形等）来判断目标对象是否为活体，可有效防止屏幕二次翻拍等作弊攻击，可使用单张或多张判断逻辑。

3、人脸比对

- 生活照 通过提取人脸的特征，计算两张人脸的相似度，从而判断是否同一个人，并给出相似度评分。在已知用户ID的情况下帮助确认是否为用户本人的对比操作，即1:1身份验证。可用于真实身份验证、人证合一验证等场景
- 证件照 在“人脸比对-生活照”的基础上，针对身份证等证件照片的比对进行专项优化

> 注 ：百度人脸识别在LFW 测评准确度高达99.77%，处于业界领先的水平

4、人脸搜索

在一个指定人脸库中查找相似的人脸。给定一张照片，与指定人脸库中的N个人脸进行比对，找出最相似的一张或多张人脸。根据待识别人脸与现有人脸库中的人脸匹配程度，返回用户信息和匹配度，即1：N人脸检索。

可用于用户身份识别、身份验证相关场景。

5、人脸库管理

提供人脸注册、人脸更新、人脸删除、创建用户组、删除用户组等维度的人脸库管理接口。

与在线接口的人脸库管理功能完全相同。

产品性能

1、单GPU支持QPS调用量

图片长宽在600像素x600像素情况下，单张P4/GTX系列、T4、RTX系列显卡上最多支持50QPS；对最新推出的RTX系列显卡进行了性能优化提升，单张显卡最多可以达到80QPS，理论横向扩展无限制

> 注：
>
> 推荐单机4卡的方式进行部署，单机6卡或更多的情况下，GPU利用率会下降

2、人脸库查询速度

比对速度和查找速度，可以实现1s以内返回。当人脸底库在100w级别，单核可在1秒内完成一次比对查找，理论上支持无限水平扩展。

3、人脸库大小

人脸库理论上没有上限限制，存储人脸库需要消耗内存，单条人脸消耗 4KB。 人脸库总量大小影响比对速度，主要消耗CPU，单核1：100万 在1秒内完成，如果要在1秒内完成1：1000万需要10个核，理论横向扩展无限制。

4、识别准确率

与公有云在线接口的识别准确率基本相同。

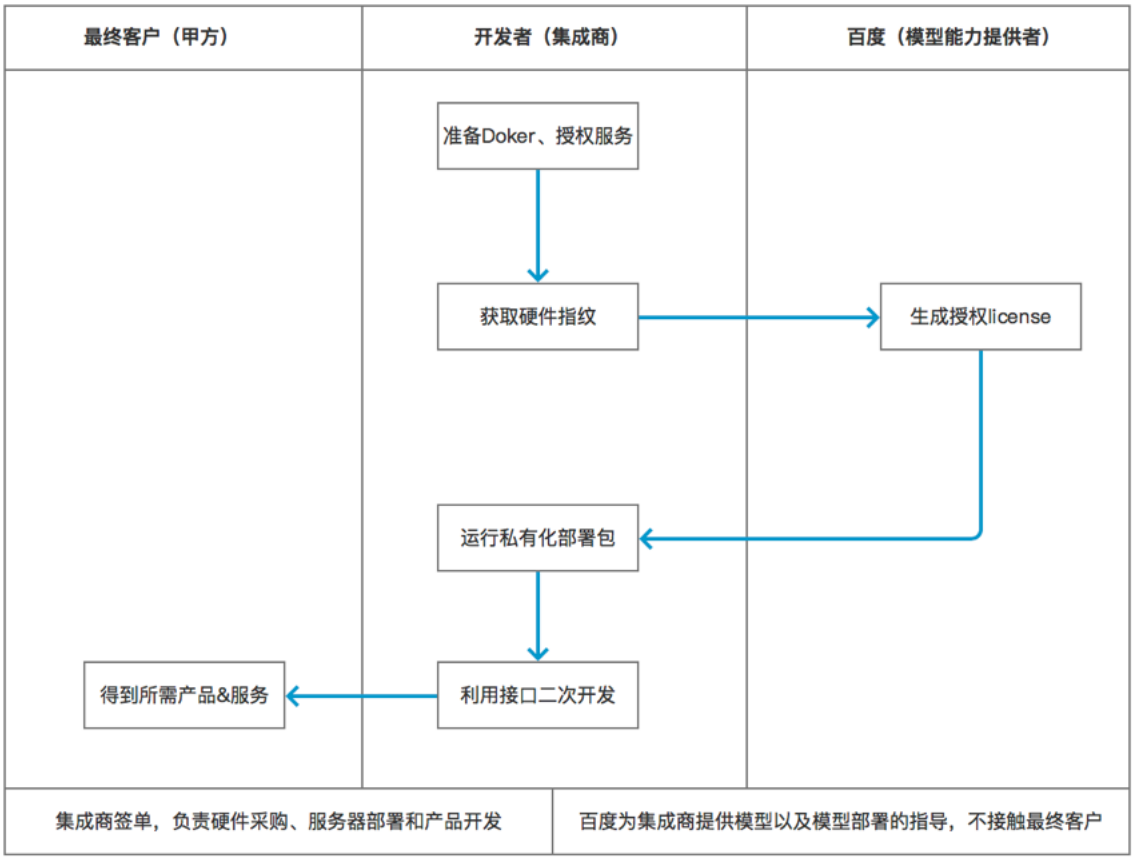
方案优势

- **纯离线**：满足无网、弱网、专网等多种网络需求，在数据私密的场景下也可以使用
- **纯软件**：纯软件方案可以快速测试、快速交付，且不与硬件进行捆绑，更灵活、利润也更高
- **超大人脸库**：支持百万级人脸库，毫秒级输出搜索结果，理论上人脸库大小无上限，且搜索响应时间不随人脸库增大而增加
- **识别效果领先**：支持百万级规模的人脸库管理和搜索，人脸识别在 LFW 测评准确度高达99.77%，检索速度业界领先，可应对各种业务需求
- **毫秒级响应**：人脸私有化部署包具备高并发、高吞吐、低时延等能力，百万级超大型数据库也可实现毫秒级响应，满足您的实时响应需求
- **性能选择灵活**：支持灵活选择各种业务需求场景的调用量需求

产品模式

百度在私有化方案中扮演底层模型能力输出者的角色，为集成商提供模型以及模型部署的指导，而不接触最终客户，由集成商与客户签单，负责硬件采购、服务器部署和产品开发。

> 注：百度只提供模型，需要客户自己解决硬件采购和部署开发的问题



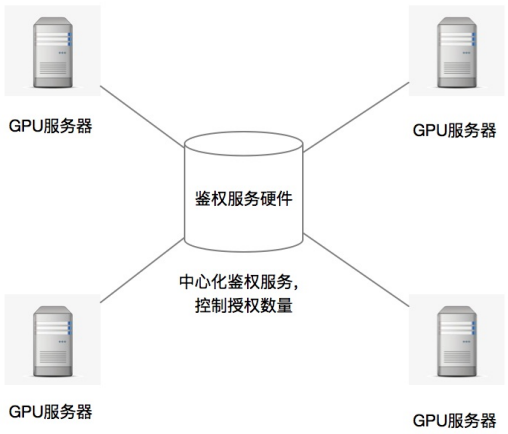
服务授权

1、授权方式

私有化方案需要百度授权后才能使用，需要您在运行授权服务的物理机提取硬件指纹并在申请人脸私有化部署包时上传指纹文件方可使用。

授权的申请数量与您使用的GPU显卡数量相同，您需要多少显卡申请多少授权即可。

> 注：运行授权服务的**物理机**的要求及**硬件指纹提取工具**使用请参考这里 授权服务可以运行在CPU设备上（比如一台固定的PC机），针对这台设备生成license，license文件中包含几个授权，就可以让人脸识别模型在多少个显卡上运行，只要保证被采集了指纹的硬件不被替换，授权也不受影响（即使替换GPU服务器也不会受影响）



2、授权有效期

正式部署包有效期至2038年，支持部署包升级。

> 注：
>
> 百度一直保持授权服务的兼容，同一场景化模型，若发现功能性质的问题，在百度优化后，开发者可以自助升级，无需额外付费。
>
> 但如果新增功能，形成了一个新的接口，则需要重新购买

使用前须知

本文档总结了客户在使用人脸识别私有化部署包过程中容易遇到的问题，以帮助您更好的实现业务需求

申请前须知

1、请提前准备好**鉴权物理机**和**GPU服务器**：

- 鉴权物理机用于安装百度提供的鉴权服务，是您运行人脸应用服务的基础，在申请时，您需要用指纹提取工具提取好指纹文件并上传才能完成申请，获取完整的部署包
- 不同的GPU显卡能够支持的性能不同，您可以根据您的业务需求购置所需显卡
  - P4/GTX、T4系列显卡最多支持50QPS（推荐T4显卡）
  - RTX系列显卡最多支持80QPS（推荐2080Ti显卡）

> 注：
>
> 1.鉴权所在服务器为linux系统（支持物理机、虚拟机或常见云服务器，不支持windows系统），具体要求请参考：指纹提取工具文档
>
> 2. GPU服务器（CPU、内存大小、硬盘、GPU显卡等）配置要求及推荐请参考：硬件配置及推荐文档
>
> 3. 申请入口（指纹提取工具在此获取）：立即申请
>
> 4. 若您想要在CPU部署私有化模型请提交工单联系百度工作人员

2、在一键安装过程中，会自动安装显卡驱动等环境，**无须手动安装**。
若您的机器中已安装显卡驱动请卸载后重新执行一键安装命令，以减少显卡驱动版本与要求版本不符导致的冲突与不兼容问题。

测试版转正式版须知

完成企业认证后，可以免费申请测试版授权，有效期为2个月。

测试版转正式版需要单独发起正式版申请，申请完成获取部署包后，只需在鉴权物理机中替换以下测试版的License文件即可，不需要重新部署人脸应用服务

> 替换License文件步骤请参考服务License更新说明或提交工单联系百度的工作人员

人脸应用服务使用须知

1. 注册图片存储须知

调用人脸注册接口时百度会存储人脸图片的特征值到数据库，但不会对原图片进行存储，建议您根据业务需要选择是否从业务端存储原图片到磁盘中进行备份

2. 抽帧服务须知

百度人脸私有化部署包不包含抽帧服务，若您需要直接对摄像头进行视频流处理，您可以在业务端进行抽帧处理或采用具有抽帧功能的摄像头

> 具有抽帧功能的摄像头推荐可参考这里

专有云服务器部署须知

百度人脸私有化部署支持本地服务器部署和专有云服务器部署，若您选取百度云GPU服务器的部署方式可以提交工单申请百度工作人员进行部署支持，服务器地区请选择**保定或苏州**

> 若您需要其他地区私有云服务器，请提交工单联系百度的工作人员

公有云转私有化

总体说明

如果您已经用百度的公有云接口完成产品验证，希望迁移到百度的私有化服务上，可以参考此说明文档。

> 因为线上线下依赖服务的不同，私有化版的人脸服务提供的能力会与公有云版有一些差异。 若您想了解更多公有云转私有化的内容，请提交工单联系百度的工作人员

注意事项

差异如下：

1、关于接口调用

- 私有化服务不需要通过AK/SK来获取token再请求，而是可以直接调用接口，具体调用方式参考接口说明文档
- 私有化的接口在请求时，需要替换成下接口地址并在请求中带上**appid**的参数。**appid**的值可以自由指定，但需要在请求接口时都使用同一个值。
- 不支持**image_type**参数中的**URL**，请使用**BASE64**和**FACE_TOKEN**参数（FACE_TOKEN参数使用也有一定限制见下文私有化FACE_TOKEN处理逻辑）

3、公有云服务使用Face_token，私有化时需要注意以下问题

Q1：如何使用face_token

- 私有化环境下想要使用Face_token,需要在生成Face_token的时候（即调用人脸检测、人脸注册接口时），要加face_field字段：feature，用于预先提取特征，方便使用face_token进行特征分析

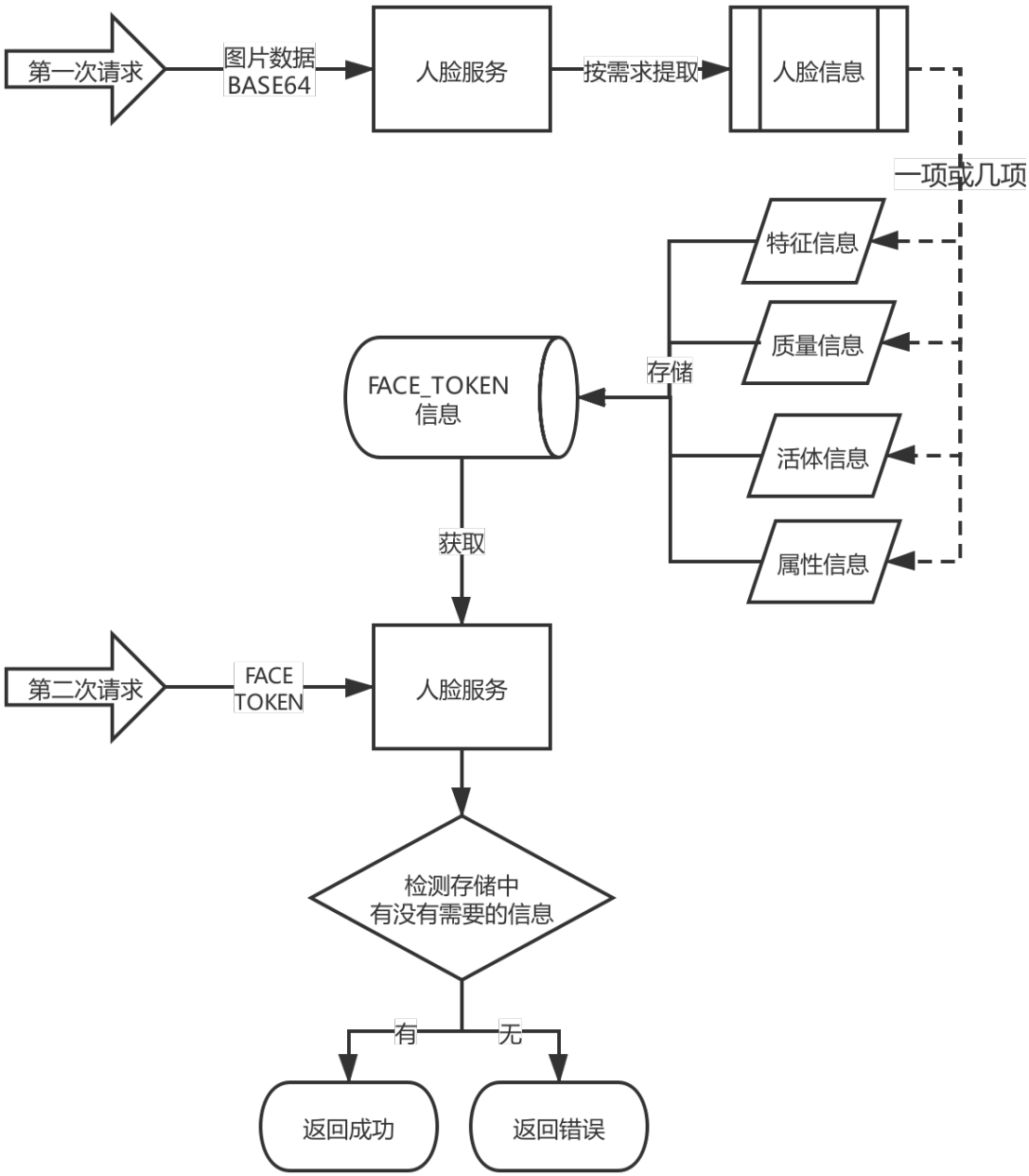
Q2：调用detect接口生成的face_token无法用来调用比对、搜索、活体等接口，应该怎么解决？

- 调用比对、搜索接口：调用detect、add接口检测时，要加face_field字段：feature，用于预先提取特征，方便使用face_token进行比对
- 调用活体接口：调用detect、add接口时，使用liveness_control参数，用于预先提取活体特征，进行活体分析
- 获取人脸属性：希望通过face_token获取什么属性，detect的阶段就需要在field参数中增加哪个参数

附：私有化Face_token处理逻辑

在使用图片进行请求时，人脸服务会将face_token和返回的人脸信息一起放入存储。后续使用face_token请求时，也只能获取当时存储人脸信息。

> 私有化时如果需要使用face_token请求人脸搜索接口，则需要在请求检测接口的**face_filed**中加入**feature**项



- 第一次请求(BASE64输入)
  - （1）生成FACE_TOKEN
  - （2）将检测的人脸信息(特征、质量、活体、属性等信息中的一项或几项)和FACE_TOKEN一起存储
  - （3）返回FACE_TOKEN
- 第二次请求(FACE_TOKEN输入)
  - （1）依据FACE_TOKEN获取上次检测出来的人脸信息
  - （2）判断存储的人脸信息能否覆盖该次请求中需要获取的信息
    - 能-返回成功
    - 否-返回失败(pic storage not support) 错误码222305

流程步骤

私有化接口调用一共分为如下几步：

Step1：准备好本地GPU服务器

GPU显卡类型与线上尽量保持一致，若您线上使用的是P4/GTX系列，则线下尽量也使用P4/GTX系列；若您线上使用的是T4/RTX系列，则线下尽量也使用T4/RTX系列。

Step2：申请私有化部署包

在百度云控制台发起私有化部署包申请，待百度工作人员审批后即可下载使用。

Step3：创建Appid

通过调用“创建用户组”接口来创建appid，即：在“创建用户组”接口中可以自定义一个Appid，当组创建成功后，此Appid即可生效。

Step4：参考接口文档进行接口调用

Step5：完成调用

快速开始

本文主要介绍人脸私有化部署的申请及使用流程，帮助您快速的使用本服务。

准备工作

请提前准备好**鉴权物理机**和**GPU服务器**：

- 鉴权物理机用于安装百度提供的鉴权服务，是您运行人脸应用服务的基础，在申请时，您需要用指纹提取工具提取好指纹文件并上传才能完成申请，获取完整的部署包
- 不同的GPU显卡能够支持的性能不同，您可以根据您的业务需求购置所需显卡
  - P4/GTX、T4系列显卡（推荐P4显卡）：单显卡最大可达到50QPS
  - RTX系列显卡（推荐2080Ti显卡）：单显卡最大可达到80QPS

> 注：
>
> 1. 鉴权物理机为Linux系统实体机（不支持虚拟机及Windows系统），具体要求请参考：指纹提取工具文档
> 2. GPU服务器（CPU、内存大小、硬盘、GPU显卡等）配置要求及推荐请参考：硬件配置及推荐文档
> 3. 申请入口（指纹提取工具在此获取）：立即申请
> 4. 若您希望在CPU部署人脸私有化模型，请提交工单联系百度的工作人员

Step1：提取硬件指纹

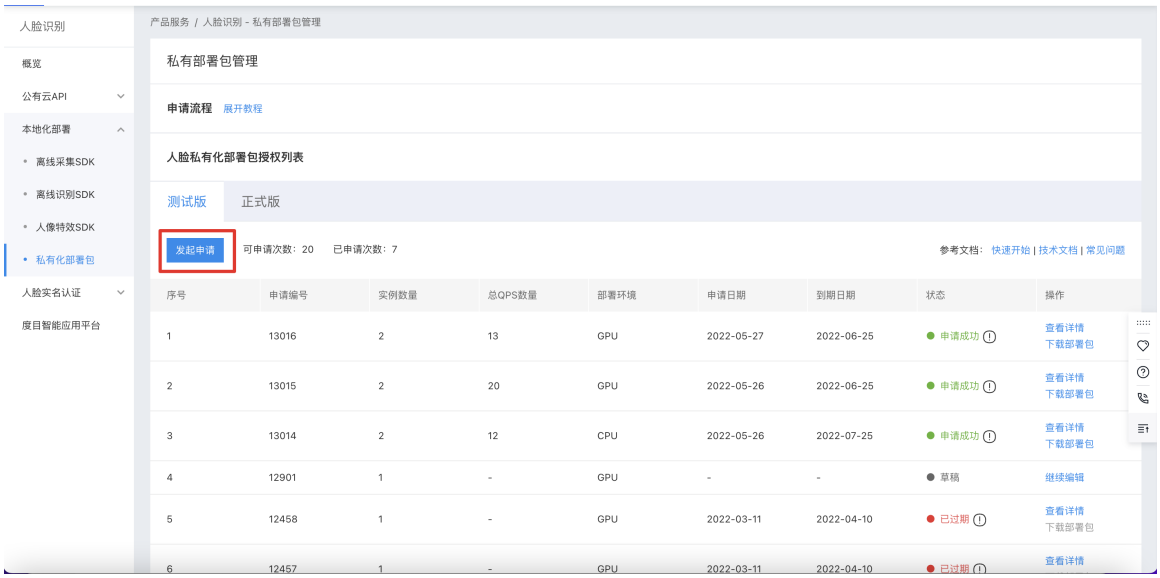
硬件指纹是您从物理机上获取的唯一标识，类似于机器码、Mac地址，百度将根据硬件指纹进行服务授权。

在申请人脸部署包时您需要上传硬件指纹以便百度为您授权。

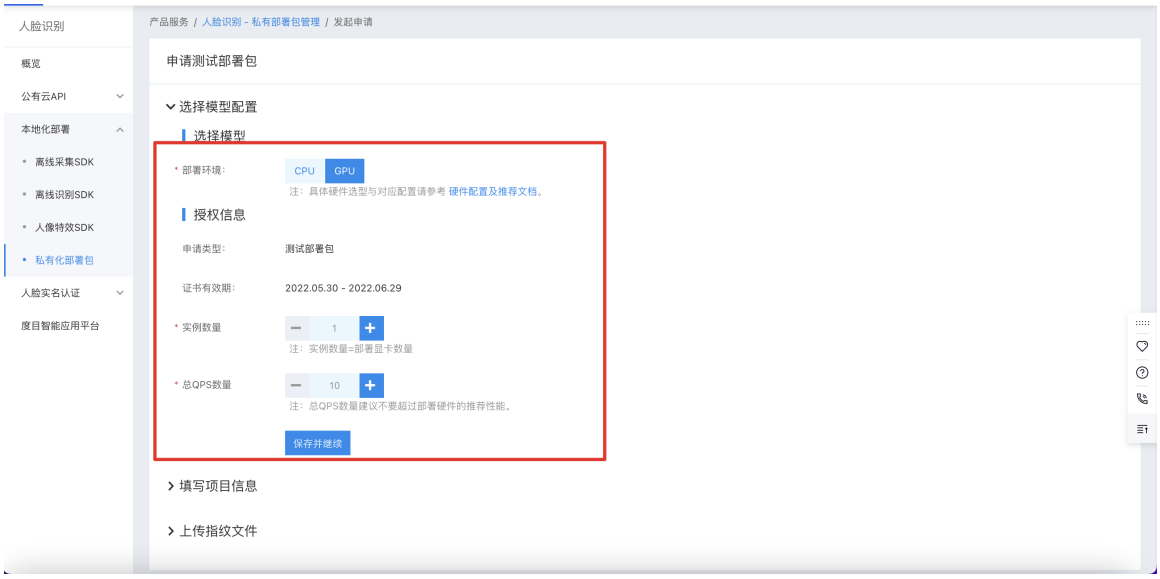
> 注：
>
> 1. 您需要提前准备好搭载Linux系统的PC物理机或者GPU服务器，关于物理机的要求以及指纹提取步骤请参考：**指纹提取工具**说明。
> 2. 指纹提取失败？请提交工单联系百度的工作人员

Step2：申请人脸部署包

- 在百度云控制台发起人脸部署包申请



- 按照您的部署环境和QPS需求申请对应模型&授权



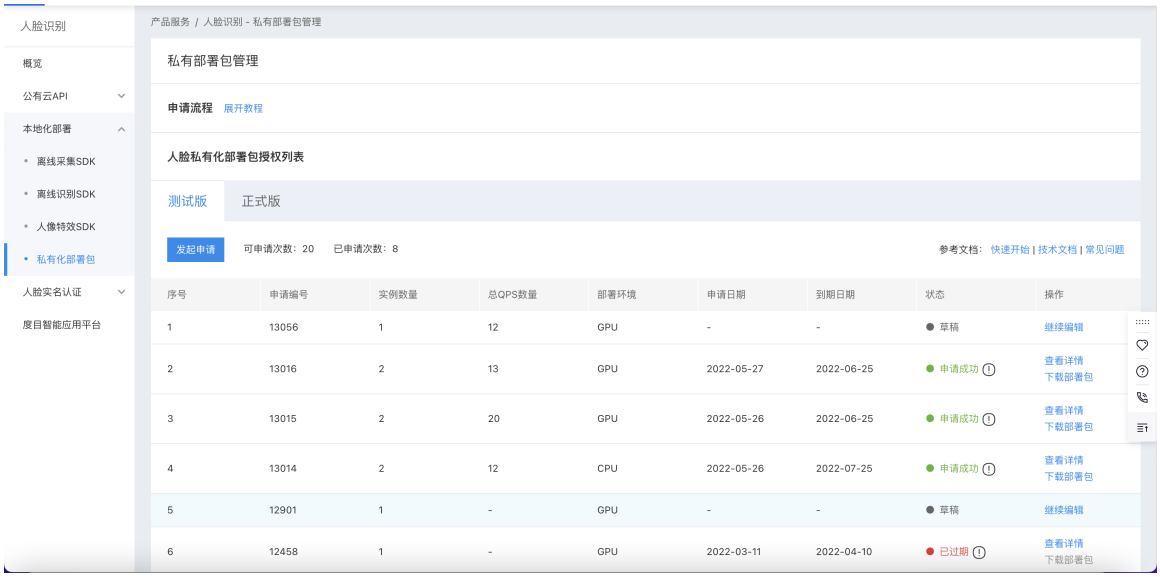
- 填写业务场景信息
- 上传指纹文件

测试版：有效期为1个月，有效期内免费，完成企业认证后即可进行免费试用

正式版：购买后可永久使用，适用于企业用户使用。

> 注：正式版申请线上购买支付完成即可获取部署包。

- 提交申请，等待1min左右刷新页面，下载部署包



Step3：安装部署包到GPU服务器

您可以一键安装鉴权服务和人脸应用服务，具体操作步骤请参考部署说明文档。

> 注：安装部署过程中遇到的问题可以查看常见问题文档或提交工单联系百度的工作人员。

Step4：调用人脸识别接口

服务部署成功后，可以调用人脸识别接口使用，具体操作步骤请参考：接口文档。

> 注：调用接口过程中遇到的问题，可以查看错误码说明文档和接口调用问题文档或提交工单联系百度的工作人员。

Step5：部署及运维问题排查

部署鉴权服务、人脸识别模型服务，以及使用人脸识别服务时遇到的问题可以查看常见问题文档。

查看服务是否部署成功，以及常用指令集可以查看运维支持文档

调用接口过程中遇到的问题，可以查看错误码说明。

百度会不定期进行模型的更新，您可以查看更新记录文档来决定是否需要升级您的模型。

若以上文档仍未解决您的问题，请提交工单联系百度的工作人员

更新记录

本文档记录了人脸私有化模型的更新记录，您可以在这里查看最新的更新记录。

若您想更新您现有的模型，需要到百度云控制台发起申请获取最新的部署包，在本地重新安装模型即可。

更新步骤

1. 卸载除数据库以外的服务（包括鉴权、驱动、人脸服务等）

```
python install.py rmall
```

2. 执行一键安装命令

```
python install.py inall
```

3. 提示安装成功后，可检查服务是否启动

```
docker ps
```

> 注：数据库服务不需要重新安装，重新安装会导致您的数据库数据丢失。
> 若您仍有疑问，请提交工单联系百度的工作人员

更新记录

日期	更新说明
2022.05	人脸私有化推出V4.0版本，并正式上线CPU模型，同时购买方式调整为白名单机制和下架历史部署包。此次更新内容包括CPU和GPU模型，具体优化如下： ① 进一步优化戴口罩识别效果 ②进一步优化图片活体检测效果 ③基于客户使用场景，针对近/远场景识别进行了深度优化
2020.05	人脸私有化推出V3.0版本 -- 上线多场景模型： 推出「通用版」、「视频监控版」、「闸机门禁版」、「安防布控版」模型 ① 通用版下正式开放CPU模型邀测②「安防版」正式更名为「视频监控版」，支持普通摄像头和抓拍机摄像头的选择切换 ③ 推出「闸机门禁版」模型，适用于轨道交通、园区通行，门禁等多种场景，并针对活体检测能力进行专项优化，通行更安全 ④「安防布控版」正式开放对外邀测，若您的场景是千万级底库，高QPS的搜索找人、以图搜图场景可以选择此模型
2020.05	推出数据库主从方案及主主方案，避免单点风险，实现数据库高可用，点击这里进行查看
2020.02	① 通用版模型优化戴口罩的人脸检测、戴口罩的人脸识别能力 ② 安防版模型优化戴口罩的人脸检测能力 口罩模型暂未更新至线上，需要请联系商务经理进行线下申请
2019.12	数据库增加导出数据库、清空特征脚本
2019.11	上线私有化2.0版本： 模型整体更新 ① 使用场景增加安防版，包含标准版所有的功能及性能规格 ② 增加最新主流显卡的适配，并在原来的基础版（上限20QPS）基础上，增加标准版（上限50QPS），高性能版（上限80QPS） ③ 修复了鉴权服务在网络不稳定状态下与人脸服务的自动重连机制
2019.10	① Aise性能优化，解决用户在数据量达到比较高的量级（eg：40W量级）以上时QPS性能较低的问题； ② 增加日志定期清理功能（15天）； ③ 增加数据库服务开机自启功能；（增加数据库开机自启功能请参考这里）
2018.10	添加M:N人脸搜索接口

接口文档

1.总体说明

私有化部署包部署成功后，即可获得与公有云基本完全相同的接口，人脸识别的相关接口将会启动，即可开始调用。

1.1 接口能力

在私有化部署包中会提供如下5类接口：

- **Appid管理**

业务能力

创建Appid：用于创建一个Appid

查询Appid：用于查询此业务中已经创建哪个Appid

> 注：创建方式为：通过调用“创建用户组”接口来创建appid，即：在“创建用户组”接口中可以自定义一个Appid，当组创建成功后，此Appid即可生效。

- **人脸检测与属性分析**

接口能力

① **人脸检测**：检测图片中的人脸并标记出位置信息;
② **人脸关键点**：展示人脸的核心关键点信息，及72个关键点信息;
③ **人脸属性值**：展示人脸属性信息，如年龄、性别等;
④ **人脸质量信息**：返回人脸各部分的遮挡、光照、模糊、完整度、置信度等信息。

典型应用场景：如**人脸属性分析**，**基于人脸关键点的加工分析**，**人脸营销活动**等。

- **人脸比对**

接口能力

① **两张人脸图片相似度对比**：比对两张图片中人脸的相似度，并返回相似度分值；
② **多种图片类型**：支持**生活照**、**证件照**、**身份证芯片照**、**带网纹照**四种类型的人脸对比；
③ **活体检测控制**：基于图片中的破绽分析，判断其中的人脸是否为**二次翻拍**（举例：如用户A用手机拍摄了一张包含人脸的图片一，用户B翻拍了图片一得到了图片二，并用图片二伪造成用户A去进行识别操作，这种情况普遍发生在金融开户、实名认证等环节。）；
④ **质量检测控制**：分析图片的中人脸的模糊度、角度、光照强度等特征，判断图片质量；

业务应用

用于比对多张图片中的人脸相似度并返回两两比对的得分，可用于判断两张脸是否是同一人的可能性大小。

典型应用场景：如**人证合一验证**，**用户认证**等，可与您现有的人脸库进行比对验证。

- **人脸搜索**

业务能力

① **1：N人脸搜索**：也称为1：N识别，在指定人脸集合中，找到最相似的人脸；
② **1：N人脸认证**：基于uid维度的1：N识别，由于uid已经锁定固定数量的人脸，所以检索范围更聚焦；

> **1：N人脸识别**与**1：N人脸认证**的差别在于：人脸搜索是在指定人脸集合中进行直接地人脸检索操作，而人脸认证是基于uid，先调取这个uid对应的人脸，再在这个uid对应的人脸集合中进行检索（因为每个uid通常对应的只有一张人脸，所以通常也就变为了1：1对比）；实际应用中，人脸认证需要用户或系统先输入id，这增加了验证安全度，但也增加了复杂度，具体使用哪个接口需要视您的业务场景判断。

> 提示：进行人脸查找相关操作前，建议先阅读**人脸库管理**相关内容。

- **人脸库管理**

业务能力

要完成1：N或者M：N识别，首先需要构建一个人脸库，用于存放所有人脸特征，相关接口如下：

人脸注册：向人脸库中添加人脸
① **人脸更新**：更新人脸库中指定用户下的人脸信息
② **人脸删除**：删除指定用户的某张人脸
③ **用户信息查询**：查询人脸库中某个用户的详细信息
④ **获取用户人脸列表**：获取某个用户组中的全部人脸列表
⑤ **获取用户列表**：查询指定用户组中的用户列表
⑥ **复制用户**：将指定用户复制到另外的人脸组
⑦ **创建用户组**：创建一个新的用户组
⑧ **删除用户组**：删除指定用户组
⑨ **组列表查询**：查询人脸库中用户组的列表

人脸库结构

人脸库、用户组、用户、用户下的人脸**层级关系**如下所示：

```
|- 人脸库(appid)
 |- 用户组一（group_id）
    |- 用户01（uid）
       |- 人脸（faceid）
    |- 用户02（uid）
       |- 人脸（faceid）
       |- 人脸（faceid）
       ....
     ....
 |- 用户组二（group_id）
 |- 用户组三（group_id）
 ....
```

关于人脸库的设置限制

① 每个私有化部署包建议对应1个appid，**每个appid对应一个人脸库**；
② 每个人脸库下，可以创建多个用户组，用户组（group）数量**没有限制**；
③ 每个用户组（group）下，可添加**无限**个user_id，**无限**张人脸（注：为了保证查询速度，单个group中的人脸容量上限建议为**80万**）；
④ 每个用户（user_id）所能注册的最大人脸数量**没有限制**；

> 提醒：每个人脸库对应一个appid，一定不要轻易删除appid，删除后则此人脸库将失效，无法进行查找！

1.2 接口格式说明

- **变量类型定义**

类型	定义
string	普通的字符串，可能会有长度要求，具体参见接口说明中的备注
uint32	整形数字，最大取值为4字节int。自然数
int64	整形数字，最大取值为8字节int。允许负数
json	无论是request还是response中某个字段定义为json，那么它其实是一个json格式的字符串，需要二次解析
array	request的query中表示array请使用key[] 。response的json中的array即为jsonArray
double	双精度，小数点后最大8位四舍五入
encryption	加密格式，一般用于人脸服务各模块间交互数据使用。

注意事项

① 所有ID定义必须为小于等于32字节的数字字母组合，尽量使用无意义的组合，并且不可以使用系统保留关键字：all、self、me、this、next。
② 所有接口POSTDATA 应当小于等于**8M**。
③ 单个group中的图片容量上限建议为**80万**
④ 人脸图片需要人脸像素在100x100以上，否则可能检测不出来人脸

- **请求方式**

请求方式统一使用`application/json`请求 **直接请求**
直接请求时需要将appid参数拼接在请求url中，不要放在json body中 注意：如果是直接对线上单台机器发起请求，需要在接口路径前加上`/api`的前缀

- **返回格式**

① **error_code**、**error_msg**即错误码和错误描述，详细含义请参考错误码表，**error_code为0代表请求成功**
② **result**是接口返回的详细信息，格式为数组；
③ **log_id**是请求的日志id，**13位长(bigint)**，用于定位请求。

```
{
  "error_code" : 0,  //错误码 0代表成功
  "error_msg"  : "SUCCESS", //错误信息
  "result" : {...} //返回结果 具体内容详见相关接口
  "log_id" : 3535325235 //请求的日志id
  "timestamp" : 1512391548 //请求到达的时间戳 精确到秒级
  "cached" : 0 //未启用 无需处理
}
```

- **阈值说明**

遮挡情况的阈值

控制度	left_eye	right_eye	nose	mouth	left_cheek	right_cheek	chin_contour
LOW	0.8	0.8	0.8	0.8	0.8	0.8	0.8
NORMAL	0.6	0.6	0.6	0.6	0.6	0.6	0.6
HIGH	0.2	0.2	0.2	0.2	0.2	0.2	0.2

模糊度、完整度的阈值

控制度	illumination	blurdegree	completeness
LOW	20	0.8	0
NORMAL	40	0.6	0
HIGH	100	0.2	1

- **参数说明**

参数	说明
left_eye	左眼被遮挡的比例 [0-1] 1表示完全遮挡
right_eye	右眼被遮挡的比例 [0-1] 1表示完全遮挡
nose	鼻子被遮挡的比例 [0-1] 1表示完全遮挡
mouth	嘴巴被遮挡的比例 [0-1] 1表示完全遮挡
left_cheek	左脸颊被遮挡的比例 [0-1] 1表示完全遮挡
right_cheek	右脸颊被遮挡的比例 [0-1] 1表示完全遮挡
chin_contour	下巴被遮挡比例 [0-1] 1表示完全遮挡
illumination	光照 [0-255] 0表示光照不好
blurdegree	图片模糊度 [0-1] 1表示完全模糊
completeness	人脸完整度（0或1） 0为人脸溢出图像边界，1为人脸都在图像边界内

- **活体控制阈值**

> 不同的控制度下所对应的活体控制阈值 如果检测出来的活体分数小于控制阈值，则会返回错误。

控制度	阈值	说明
LOW	0.05	0.01%活体误拒率 **对应的通过率为99.99% 攻击拒绝率为63.91%**
NORMAL	0.3	0.1%活体误拒率 **对应的通过率为99.9% 攻击拒绝率为90.32%**
HIGH	0.9	1%活体误拒率 **对应的通过率为99% 攻击拒绝率为97.56%**

> 误拒率（FRR）：如0.5%，指1000次真人请求，会有5次因为活体分数低于阈值被错误拒绝。 通过率（TAR）：如99%，指100次真人请求，会有99次因为活体分数高于阈值而通过。 攻击拒绝率（TRR）：如99%，代表100次作弊假体攻击，会有99次被拒绝。 误拒率越高， 其对攻击的防范能力也会越高

1.3 调用准备

1、调用接口的地址示例：[192.168.0.1]:8300/face-api/v3/face/detect，其中ip需要替换为用户自己服务器的ip，端口固定为：8300，接口地址需要替换为下述接口的地址。

2、调用接口时需要指定appid，此appid为用户自定的id，目前可以通过调用“创建用户组”接口来创建appid

2. 接口详情

2.1 基础能力接口

2.1.1 人脸检测

> 检测图片中的人脸并获得位置信息，属性信息，质量信息等

- 请求路径

/face-api/v3/face/detect

- URL请求参数

参数	必选	类型	说明
appid	是	string	app标识 （由数字、字母、下划线组成），长度限制48B

- Header：

参数	值
Content-Type	application/json

- Body请求参数

参数	必选	类型	说明
image	是	string	图片信息(**总数据大小应小于10M，分辨率应小于1920*1080**)，图片上传方式根据image_type来判断
image_type	是	string	图片类型 **BASE64**:图片的base64值，base64编码后的图片数据，编码后的图片大小不超过2M； **FACE_TOKEN**: 人脸图片的唯一标识，调用人脸检测接口时，会为每个人脸图片赋予一个唯一的FACE_TOKEN，同一张图片多次检测得到的FACE_TOKEN是同一个。
face_field	否	string	包括**age,expression,face_shape,gender,glasses,landmark,quality,face_type,parsing,eye_status,emotion,feature,spoofing信息** 逗号分隔. 默认只返回face_token、人脸框、概率和旋转角度
get_feature	否	string	是否保存特征值， 只有保存了特征值之后才能使用FACE_TOKEN，默认为NO YES 保存特征值 NO 不保存特征值 2019.12.12以后部署的模型提供此接口请求参数，之前部署的模型若需要保存特征值，请在face_field字段中添加feature字段 或者更新模型，更新方式请参考这里
max_face_num	否	uint32	最多处理人脸的数目，默认值为1，仅检测图片中面积最大的那个人脸；**最大值10**，检测图片中面积最大的几张人脸。若您想修改最大人脸检测数量数值，请参考接口调用问题文档
face_type	否	string	人脸的类型 **LIVE**表示生活照：通常为手机、相机拍摄的人像图片、或从网络获取的人像图片等 **IDCARD**表示身份证芯片照：二代身份证内置芯片中的人像照片 **WATERMARK**表示带水印证件照：一般为带水印的小图，如公安网小图 **CERT**表示证件照片：如拍摄的身份证、工卡、护照、学生证等证件图片 默认**LIVE**
liveness_control	否	string	**活体控制** 检测结果中不符合要求的人脸会被过滤 **NONE**: 不进行控制 **LOW**:较低的活体要求(高通过率 低攻击拒绝率) **NORMAL**: 一般的活体要求(平衡的攻击拒绝率, 通过率) **HIGH**: 较高的活体要求(高攻击拒绝率 低通过率) 默认**NONE**

若为**视频监控模型**，请求参数可以增加下列字段：

> 若您的前端摄像头是抓拍机抓拍并处理过的单人脸小图模式，推荐您在人脸检测/人脸注册/人脸更新/人脸搜索接口中使用SNAP参数，若为普通摄像头及混合场景模式，使用COMMON参数即可

参数	必选	类型	说明
image_spec	否	string	图片规格 服务会对不同规格的图片采用不同的图片处理方式 **COMMON**: 通用图片规格 默认的处理方式(如无适应的图片规格则使用此参数) **SNAP**: 从人脸抓拍机得到的抓拍图 适用于**小图片+单人脸**的情况 **默认COMMON**

- 请求示例

```
{
  "image": "027d8308a2ec665acb1bdf63e513bcb9",
  "image_type": "FACE_TOKEN",
  "face_field": "faceshape,facetype"
}
```

- 返回结果

字段	必选	类型	说明
face_num	是	int	检测到的图片中的人脸数量
face_list	是	array	人脸信息列表，具体包含的参数参考下面的列表。
+face_token	是	string	人脸图片的唯一标识
+location	是	array	人脸在图片中的位置
++left	是	double	人脸区域离左边界的距离
++top	是	double	人脸区域离上边界的距离
++width	是	double	人脸区域的宽度
++height	是	double	人脸区域的高度
++rotation	是	int64	人脸框相对于竖直方向的顺时针旋转角，[-180,180]
+face_probability	是	double	人脸置信度，范围【0~1】，代表这是一张人脸的概率，0最小、1最大。
+angel	是	array	人脸旋转角度参数
++yaw	是	double	三维旋转之左右旋转角[-90(左), 90(右)]
++pitch	是	double	三维旋转之俯仰角度[-90(上), 90(下)]
++roll	是	double	平面内旋转角[-180(逆时针), 180(顺时针)]
+age	否	double	年龄 ，当**face_field包含age时返回**
+expression	否	array	表情，当 **face_field包含expression时返回**
++type	否	string	**none**:不笑；**smile**:微笑；**laugh**:大笑
++probability	否	double	表情置信度，范围【0~1】，0最小、1最大。
+face_shape	否	array	脸型，当**face_field包含face_shape时返回**
++type	否	double	**square**: 正方形 **triangle**:三角形 **oval**: 椭圆 **heart**: 心形 **round**: 圆形
++probability	否	double	置信度，范围【0~1】，代表这是人脸形状判断正确的概率，0最小、1最大。
+gender	否	array	性别，**face_field包含gender时返回**
++type	否	string	male:**男性** female:**女性**
++probability	否	double	性别置信度，范围【0~1】，0代表概率最小、1代表最大。
+glasses	否	array	是否带眼镜，**face_field包含glasses时返回**
++type	否	string	**none**:无眼镜，**common**:普通眼镜，**sun**:墨镜
++probability	否	double	眼镜置信度，范围【0~1】，0代表概率最小、1代表最大。

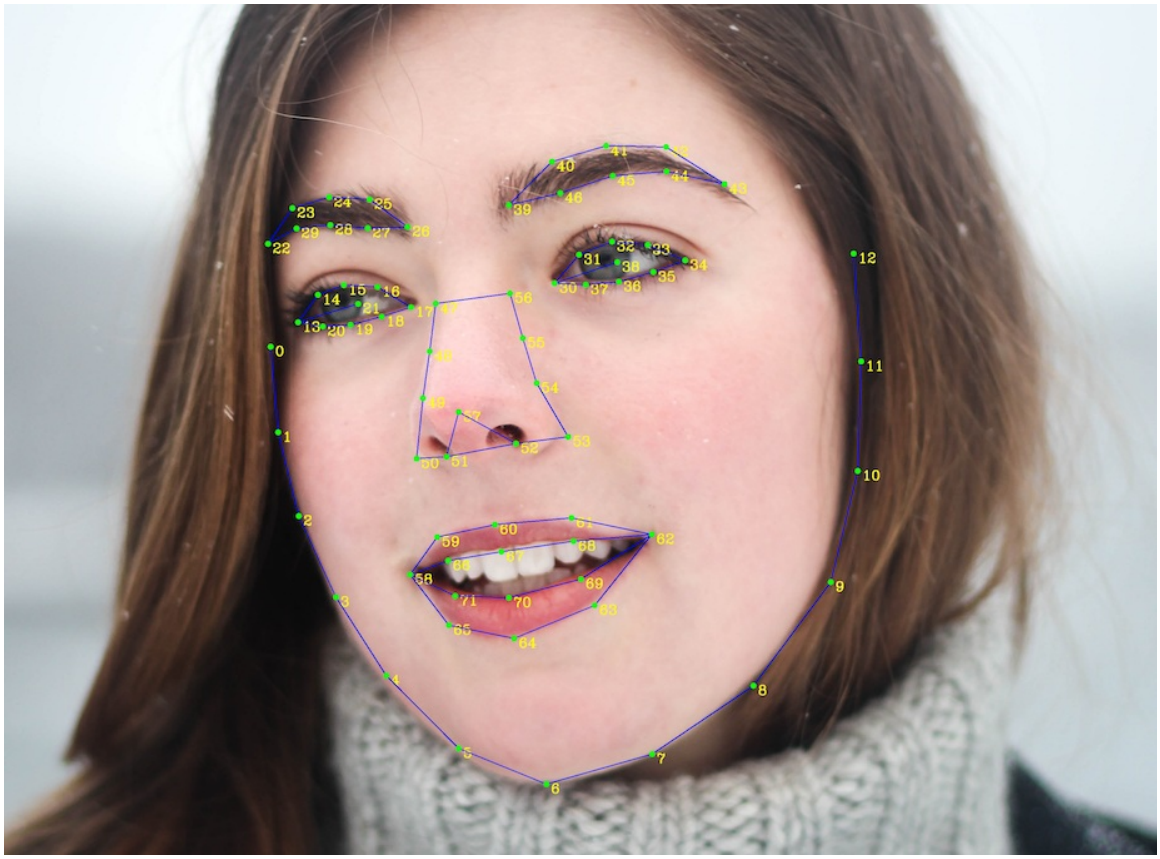
+eye_status	否	array	双眼状态（睁开/闭合） **face_field包含eye_status时返回**
++left_eye	否	double	左眼状态 [0,1]取值，越接近0闭合的可能性越大
++right_eye	否	double	右眼状态 [0,1]取值，越接近0闭合的可能性越大
+emotion	否	array	情绪 **face_field包含emotion时返回**
++type	否	string	**angry**:愤怒 **disgust**:厌恶 **fear**:恐惧 **happy**:高兴 **sad**:伤心 **surprise**:惊讶 **neutral**:无情绪
++probability	否	double	情绪置信度，范围0~1
+face_type	否	array	真实人脸/卡通人脸 **face_field包含face_type时返回**
++type	否	string	**human**: 真实人脸 **cartoon**: 卡通人脸
++probability	否	double	人脸类型判断正确的置信度，范围【0~1】，0代表概率最小、1代表最大。
+landmark	否	array	4个关键点位置，左眼中心、右眼中心、鼻尖、嘴中心。**face_field包含landmark时返回**
+landmark72	否	array	72个特征点位置 **face_field包含landmark72时返回**
+quality	否	array	人脸质量信息。**face_field包含quality时返回**
++occlusion	否	array	人脸各部分遮挡的概率，范围[0~1]，0表示完整，1表示不完整
+++left_eye	否	double	左眼遮挡比例，[0-1] ，1表示完全遮挡
+++right_eye	否	double	右眼遮挡比例，[0-1] ， 1表示完全遮挡
+++nose	否	double	鼻子遮挡比例，[0-1] ， 1表示完全遮挡
+++mouth	否	double	嘴巴遮挡比例，[0-1] ， 1表示完全遮挡
+++left_cheek	否	double	左脸颊遮挡比例，[0-1] ， 1表示完全遮挡
+++right_cheek	否	double	右脸颊遮挡比例，[0-1] ， 1表示完全遮挡
+++chin	否	double	下巴遮挡比例，[0-1] ， 1表示完全遮挡
++blur	否	double	人脸模糊程度，范围[0~1]，0表示清晰，1表示模糊
++illumination	否	double	取值范围在[0~255], 表示脸部区域的光照程度 越大表示光照越好
++completeness	否	int64	人脸完整度，0或1, 0为人脸溢出图像边界，1为人脸都在图像边界内
+spoofing	否	double	判断图片是否合成图功能
++char	否	string	人脸当前正在读出的文字（当前只支持对数字进行识别，返回内容为单个数字，范围0~9）
++probability	否	double	置信度，范围0~1
+parsing_info	否	string	人脸分层结果 结果数据是使用gzip压缩后再base64编码 使用前需base64解码后再解压缩 原数据格式为string 形如0,0,0,0,0,1,1,1,1,1,1,2,2,2,2,2,2,2,2,2,…

- 返回示例


```
{
"error_code": 0,
"error_msg": "SUCCESS",
"log_id": 1234567890123,
"timestamp": 1533094400,
"cached": 0,
"result": {
    "face_num": 1,
    "face_list": [
        {
            "face_token": "35235asfas21421fakghktyfdgh68bio",
            "location": {
                "left": 117,
                "top": 131,
                "width": 172,
                "height": 170,
                "rotation": 4
            },
            "face_probability": 1,
            "angle": {
                "yaw": -0.34859421849251,
                "pitch": 1.9135693311691,
                "roll": 2.3033397197723
            },
            "landmark": [
              {
                  "x": 61.67,
                  "y": 133.44
              },
              {
                  "x": 99.73,
                  "y": 129.72
              },
              {
                  "x": 85.88,
                  "y": 155.07
              },
              {
                  "x": 85.82,
                  "y": 173.16
              }
            ],
            "age": 29.298097610474,
            "expression": {
                "type": "smile",
                "probability": 0.5543018579483
            },
            "gender": {
                "type": "male",
                "probability": 0.99979132413864
            },
            "glasses": {
                "type": "sun",
                "probability": 0.99999964237213
            },
            "eye_status": {
                "left_eye": 0.9999974966,
                "right_eye": 0.9999724627
            }
```

```
            },
            "face_shape": {
                "type": "triangle",
                "probability": 0.5543018579483
            },
            "quality": {
                "occlusion": {
                    "left_eye": 0,
                    "right_eye": 0.015625,
                    "nose": 0,
                    "mouth": 0,
                    "left_cheek": 0.03078358248,
                    "right_cheek": 0.03356481344,
                    "chin_contour": 0.006372549105
                },
                "blur": 5.188863383e-7,
                "illumination": 131,
                "completeness": 1
            },
            "lib_language": {
                 "char": "0",
                 "probability": 0.67
            }
        }
    ]
}
}
```


72个关键点分布图（对应landmark72个点的顺序，序号从0-71）：



2.1.2 人脸对比

> 两张人脸图片相似度对比：比对两张图片中人脸的相似度，并返回相似度分值

- 请求路径

/face-api/v3/face/match

- URL请求参数

参数	必选	类型	说明
appid	是	string	app标识 （由数字、字母、下划线组成），长度限制48B

- Header：

参数	值
Content-Type	application/json

- Body请求参数

参数	必选	类型	说明
image	是	string	图片信息(**总数据大小应小于10M**)，图片上传方式根据image_type来判断。 **两张图片通过json格式上传，格式参考表格下方示例**
image_type	是	string	图片类型 **BASE64**:图片的base64值，base64编码后的图片数据，编码后的图片大小不超过2M； **FACE_TOKEN**: 人脸图片的唯一标识，调用人脸检测接口时，会为每个人脸图片赋予一个唯一的FACE_TOKEN，同一张图片多次检测得到的FACE_TOKEN是同一个。
face_type	否	string	人脸的类型 **LIVE**：表示生活照：通常为手机、相机拍摄的人像图片、或从网络获取的人像图片等， **IDCARD**：表示身份证芯片照：二代身份证内置芯片中的人像照片， **WATERMARK**：表示带水印证件照：一般为带水印的小图，如公安网小图 **CERT**：表示证件照片：如拍摄的身份证、工卡、护照、学生证等证件图片 默认LIVE
quality_control	否	string	图片质量控制 **NONE**: 不进行控制 **LOW**:较低的质量要求 **NORMAL**: 一般的质量要求 **HIGH**: 较高的质量要求 **默认 NONE** 若图片质量不满足要求，则返回结果中会提示质量检测失败
liveness_control	否	string	活体检测控制 **NONE**: 不进行控制 **LOW**:较低的活体要求(高通过率 低攻击拒绝率) **NORMAL**: 一般的活体要求(平衡的攻击拒绝率, 通过率) **HIGH**: 较高的活体要求(高攻击拒绝率 低通过率) **默认 NONE** 若活体检测结果不满足要求，则返回结果中会提示活体检测失败

- 请求示例

```
[
    {
        "image": "sfasq35sadvsvqwr5q...",
        "image_type": "BASE64",
        "face_type": "LIVE",
        "quality_control": "LOW",
        "liveness_control": "HIGH"
    },
    {
        "image": "sfasq35sadvsvqwr5q...",
        "image_type": "BASE64",
        "face_type": "IDCARD",
        "quality_control": "LOW",
        "liveness_control": "HIGH"
    }
]
```

> 注意：请求体要求为json格式，可以参考请求示例。

- 请求示例

```
[
  {
      "image": "sfasq35sadvsvqwr5q...",
      "image_type": "BASE64",
      "face_type": "LIVE",
      "quality_control": "LOW",
      "liveness_control": "HIGH"
  },
  {
      "image": "sfasq35sadvsvqwr5q...",
      "image_type": "BASE64",
      "face_type": "IDCARD",
      "quality_control": "LOW",
      "liveness_control": "HIGH"
  }
]
```

- 返回结果

参数名	必选	类型	说明
score	是	float	人脸相似度得分，推荐阈值80分
face_list	是	array	人脸信息列表
+face_token	是	string	人脸的唯一标志

- 返回示例

```
{
"error_code": 0,
"error_msg": "SUCCESS",
"log_id": 1234567890123,
"timestamp": 1533094576,
"cached": 0,
"result": {
    "score": 44.3,
    "face_list": [
        {
            "face_token": "fid1"
        },
        {
            "face_token": "fid2"
        }
    ]
}
}
```


2.1.3 人脸搜索

> 在指定人脸集合中，找到最相似的人脸

- 请求路径

/face-api/v3/face/identify

- URL请求参数

参数	必选	类型	说明
appid	是	string	app标识 （由数字、字母、下划线组成），长度限制48B

- Header：

参数	值
Content-Type	application/json

- Body请求参数

参数	必选	类型	说明
image	是	string	图片信息(**总数据大小应小于10M**)，图片上传方式根据image_type来判断
image_type	是	string	图片类型 **BASE64**:图片的base64值，base64编码后的图片数据，编码后的图片大小不超过2M； **FACE_TOKEN**: 人脸图片的唯一标识，调用人脸检测接口时，会为每个人脸图片赋予一个唯一的FACE_TOKEN，同一张图片多次检测得到的FACE_TOKEN是同一个。
group_id_list	是	string	从指定的group中进行查找 用逗号分隔，**上限10个**
quality_control	否	string	图片质量控制 **NONE**: 不进行控制 **LOW**:较低的质量要求 **NORMAL**: 一般的质量要求 **HIGH**: 较高的质量要求 **默认 NONE** 若图片质量不满足要求，则返回结果中会提示质量检测失败
liveness_control	否	string	活体检测控制 **NONE**: 不进行控制 **LOW**:较低的活体要求(高通过率 低攻击拒绝率) **NORMAL**: 一般的活体要求(平衡的攻击拒绝率, 通过率) **HIGH**: 较高的活体要求(高攻击拒绝率 低通过率) **默认NONE** 若活体检测结果不满足要求，则返回结果中会提示活体检测失败
user_id	否	string	当需要对特定用户进行比对时，指定user_id进行比对。即人脸认证功能。
match_threshold	否	int	匹配阈值（设置阈值后，score低于此阈值的用户信息将不会返回） 最大100 最小0 默认0 此阈值设置得越高，检索速度将会越快，推荐使用80的阈值
max_user_num	否	unit32	查找后返回的用户数量。返回相似度最高的几个用户，默认为1，最多返回50个,若您想要修改最大返回人脸数量，请参考4.如何控制1：N返回的最大人脸数问题文档

若为**视频监控模型**，请求参数可以增加下列字段：

> 若您的前端摄像头是抓拍机抓拍并处理过的单人脸小图模式，推荐您在人脸检测/人脸注册/人脸更新/人脸搜索接口中使用SNAP参数，若为普通摄像头及混合场景模式，使用COMMON参数即可

参数	必选	类型	说明
image_spec	否	string	图片规格 服务会对不同规格的图片采用不同的图片处理方式 **COMMON**: 通用图片规格 默认的处理方式(如无适应的图片规格则使用此参数) **SNAP**: 从人脸抓拍机得到的抓拍图 适用于**小图片+单人脸**的情况 **默认COMMON**

> 说明：如果使用base 64格式的图片，两张请求的图片请分别进行base64编码。

- 请求示例

```
{
  "image": "027d8308a2ec665acb1bdf63e513bcb9",
  "image_type": "FACE_TOKEN",
  "group_id_list": "group_repeat,group_233",
  "quality_control": "LOW",
  "liveness_control": "NORMAL"
}
```

- 返回结果

字段	必选	类型	说明
face_token	是	string	人脸标志
user_list	是	array	匹配的用户信息列表
+group_id	是	string	用户所属的group_id
+user_id	是	string	用户的user_id
+user_info	是	string	注册用户时携带的user_info
+score	是	float	用户的匹配得分，推荐阈值80分

```
{
  "error_code": 0,
  "error_msg": "SUCCESS",
  "log_id": 1234567890123,
  "timestamp": 1533094591,
  "cached": 0,
  "result": {
     "face_token": "fid",
     "user_list": [
        {
           "group_id": "test1",
           "user_id": "u333333",
           "user_info": "Test User",
           "score": 99.3
        }
     ]
  }
}
```

2.1.4 M:N 多人脸搜索

> 使用多人脸的图片，在指定人脸集合中，查找最相似的人脸

- 请求路径 /face-api/v3/face/midentify
- URL请求参数

参数	必选	类型	说明
appid	是	string	app标识 （由数字、字母、下划线组成），长度限制48B

- Header：

参数	值
Content-Type	application/json

- Body请求参数

参数	必选	类型	说明
image	是	string	图片信息(**数据大小应小于10M 分辨率应小于1920*1080**)
image_type	是	string	图片类型 **BASE64**:图片的base64值; **FACE_TOKEN**: face_token 人脸标识
group_id_list	是	string	从指定的group中进行查找 用逗号分隔，**上限10个**
max_face_num	否	int	最多处理人脸的数目 **默认值为1(仅检测图片中面积最大的那个人脸) 最大值10**
match_threshold	否	int	匹配阈值（设置阈值后，score低于此阈值的用户信息将不会返回） 最大100 最小0 默认80 **此阈值设置得越高，检索速度将会越快，推荐使用默认阈值80**
quality_control	否	string	质量控制(质量不符合要求的人脸不会出现在返回结果中) **NONE**: 不进行控制 **LOW**:较低的质量要求 **NORMAL**: 一般的质量要求 **HIGH**: 较高的质量要求 **默认NONE**
liveness_control	否	string	活体控制(活体分数不符合要求的人脸不会出现在返回结果中) **NONE**: 不进行控制 **LOW**:较低的活体要求(高通过率 低攻击拒绝率) **NORMAL**: 一般的活体要求(平衡的攻击拒绝率, 通过率) **HIGH**: 较高的活体要求(高攻击拒绝率 低通过率) **默认NONE**
max_user_num	否	unit32	识别返回的最大用户数，默认为1，最大值20

> 多人脸的情况下 如果设置了质量控制、活体控制参数，不合格的人脸将被过滤，不会出现在结果中。

- 请求示例

```
 {
"image": "/9j/4AAQSkZJRgABAQAAAQABAAD/2wBDAAgGBgcGBQgHBwcJCQgKD...",
"image_type": "BASE64",
"group_id_list": "group1",
"max_face_num" : 5,
"quality_control": "LOW",
"liveness_control": "NORMAL"
}
```

- 返回结果

字段	必选	类型	说明
face_num	是	int	图片中的人脸数量
face_list	是	array	人脸信息列表
+face_token	是	string	人脸标志
+location	是	array	人脸在图片中的位置
++left	是	double	人脸区域离左边界的距离
++top	是	double	人脸区域离上边界的距离
++width	是	double	人脸区域的宽度
++height	是	double	人脸区域的高度
++rotation	是	int64	人脸框相对于竖直方向的顺时针旋转角，[-180,180]
+user_list	是	array	匹配的用户信息列表
++group_id	是	string	用户所属的group_id
++user_id	是	string	用户的user_id
++user_info	是	string	注册用户时携带的user_info
++score	是	float	用户的匹配得分 **80分以上可以判断为同一人，此分值对应万分之一误识率**

- 返回示例

```
{
 "error_code": 0,
 "error_msg": "SUCCESS",
 "log_id": 240483475,
 "timestamp": 1535533440,
 "cached": 0,
 "result": {
    "face_num": 2,
    "face_list": [
       {
          "face_token": "6fe19a6ee0c4233db9b5bba4dc2b9233",
          "location": {
             "left": 31.95568085,
             "top": 120.3764267,
             "width": 87,
             "height": 85,
             "rotation": -5
          },
          "user_list": [
             {
                "group_id": "group1",
                "user_id": "5abd24fd062e49bfa906b257ec40d284",
                "user_info": "userinfo1",
                "score": 69.85684967041
             },
             {
                "group_id": "group1",
                "user_id": "2abf89cffb31473a9948268fde9e1c3f",
                "user_info": "userinfo2",
                "score": 66.586112976074
             }
          ]
       },
       {
          "face_token": "fde61e9c074f48cf2bbb319e42634f41",
          "location": {
             "left": 219.4467773,
             "top": 104.7486954,
             "width": 81,
             "height": 77,
             "rotation": 3
          },
          "user_list": [
             {
                "group_id": "group1",
                "user_id": "088717532b094c3990755e91250adf7d",
                "user_info": "userinfo",
                "score": 65.154159545898
             }
          ]
       }
    ]
 }
}
```


2.1.5 图片活体检测

> 多图活体检测接口，支持送入1/3/8张图片进行活体检测(如果传入一个人的多张图片，活体判断会更准确)

- 请求路径

/face-api/v3/face/liveness

- URL请求参数

参数	必选	类型	说明
appid	是	string	app标识 （由数字、字母、下划线组成），长度限制48B

- Header：

参数	值
Content-Type	application/json

- Body请求参数

参数名	必选	类型	说明
image	是	string	图片信息(总数据大小应小于10M)，图片上传方式根据image_type来判断； **可以上传同一个用户的1张、3张或8张图片来进行活体判断，注：后端会选择每组照片中的最高分数作为整体分数。**
image_type	是	string	图片类型 **BASE64**:图片的base64值，base64编码后的图片数据，需urlencode，编码后的图片大小不超过2M； **FACE_TOKEN**: 人脸图片的唯一标识，调用人脸检测接口时，会为每个人脸图片赋予一个唯一的FACE_TOKEN，同一张图片多次检测得到的FACE_TOKEN是同一个。
face_field	否	string	包括**age,expression,face_shape,gender,glasses,landmark,quality,face_type,parsing,eye_status,emotion,feature信息**，逗号分隔，默认只返回face_token、活体数、人脸框、概率和旋转角度
option	否	string	场景信息，程序会视不同的场景选用相对应的模型。当前支持的场景有COMMON(通用场景)，GATE(闸机场景)，默认使用COMMON

- 请求示例

```
[
  {
    "image": "sfasq35sadvsvqwr5q...",
    "image_type": "BASE64",
    "face_field": "age",
    "option": "COMMON"
  },
  {
    "image": "http://xxx.baidu.com/image1.png",
    "image_type": "URL",
    "face_field": "age",
    "option": "COMMON"
  },
  {
    "image": "9f30d19f86f89f2f07ce88b69557061a",
    "image_type": "FACE_TOKEN",
    "face_field": "age",
    "option": "COMMON"
  }
]
```

- 返回结果

参数	类型	是否必须	说明
face_liveness	是	float	活体分数值
thresholds	是	array	由服务端返回最新的阈值数据（随着模型的优化，阈值可能会变化），将此参数与返回的face_liveness进行比较，可以作为活体判断的依据。 frr_1e-4：万分之一误识率的阈值；frr_1e-3：千分之一误识率的阈值；frr_1e-2：百分之一误识率的阈值。误识率越低，准确率越高，相应的拒绝率也越高
face_list	是	array	每张图片的详细信息描述，如果只上传一张图片，则只返回一个结果。
+face_token	是	string	人脸图片的唯一标识
+location	是	array	人脸在图片中的位置
++left	是	double	人脸区域离左边界的距离
++top	是	double	人脸区域离上边界的距离
++width	是	double	人脸区域的宽度
++height	是	double	人脸区域的高度
++rotation	是	int64	人脸框相对于竖直方向的顺时针旋转角，[-180,180]
+face_probability	是	double	人脸置信度，范围【0~1】，代表这是一张人脸的概率，0最小、1最大。
+angle	是	array	人脸旋转角度参数
++yaw	是	double	三维旋转之左右旋转角[-90(左), 90(右)]
++pitch	是	double	三维旋转之俯仰角度[-90(上), 90(下)]
++roll	是	double	平面内旋转角[-180(逆时针), 180(顺时针)]
+age	否	double	年龄 ，当**face_field包含age时返回**
+expression	否	array	表情，当 **face_field包含expression时返回**
++type	否	string	**none**:不笑；**smile**:微笑；**laugh**:大笑
++probability	否	double	表情置信度，范围【0~1】，0最小、1最大。
+face_shape	否	array	脸型，当**face_field包含face_shape时返回**
++type	否	double	**square**: 正方形 **triangle**:三角形 **oval**: 椭圆 **heart**: 心形 **round**: 圆形
++probability	否	double	置信度，范围【0~1】，代表这是人脸形状判断正确的概率，0最小、1最大。
+gender	否	array	性别，**face_field包含gender时返回**
++type	否	string	male:**男性** female:**女性**

++probability	否	double	性别置信度，范围【0~1】，0代表概率最小、1代表最大。
+glasses	否	array	是否带眼镜，**face_field包含glasses时返回**
++type	否	string	**none**:无眼镜，**common**:普通眼镜，**sun**:墨镜
++probability	否	double	眼镜置信度，范围【0~1】，0代表概率最小、1代表最大。
+face_type	否	array	真实人脸/卡通人脸 **face_field包含face_type时返回**
++type	否	string	**human**: 真实人脸 **cartoon**: 卡通人脸
++probability	否	double	人脸类型判断正确的置信度，范围【0~1】，0代表概率最小、1代表最大。
+landmark	否	array	4个关键点位置，左眼中心、右眼中心、鼻尖、嘴中心。**face_field包含landmark时返回**
+landmark72	否	array	72个特征点位置 **face_field包含landmark时返回**
+quality	否	array	人脸质量信息。**face_field包含quality时返回**
++occlusion	否	array	人脸各部分遮挡的概率，范围[0~1]，0表示完整，1表示不完整
+++left_eye	否	double	左眼遮挡比例，[0-1] ， 1表示完全遮挡
+++right_eye	否	double	右眼遮挡比例，[0-1] ， 1表示完全遮挡
+++nose	否	double	鼻子遮挡比例，[0-1] ， 1表示完全遮挡
+++mouth	否	double	嘴巴遮挡比例，[0-1] ， 1表示完全遮挡
+++left_cheek	否	double	左脸颊遮挡比例，[0-1] ， 1表示完全遮挡
+++right_cheek	否	double	右脸颊遮挡比例，[0-1] ， 1表示完全遮挡
+++chin	否	double	下巴遮挡比例，，[0-1] ， 1表示完全遮挡
++blur	否	double	人脸模糊程度，范围[0~1]，0表示清晰，1表示模糊
++illumination	否	double	取值范围在[0~255], 表示脸部区域的光照程度 越大表示光照越好
++completeness	否	int64	人脸完整度，0或1, 0为人脸溢出图像边界，1为人脸都在图像边界内
+liveness	否	array	单张图片活体检测结果
++livemapscore	否	double	单张图片的活体打分 范围[0~1]

+parsing_info	否	string	人脸分层结果 结果数据是使用gzip压缩后再base64编码 使用前需base64解码后再解压缩 原数据格式为string 形如0,0,0,0,0,1,1,1,1,1,1,2,2,2,2,2,2,2,2,2,…
++char	否	string	人脸当前正在读出的文字（当前只支持对数字进行识别，返回内容为单个数字，范围0~9）
++probability	否	double	置信度，范围0~1

- 返回示例


```
 {
"error_code": 0,
"error_msg": "SUCCESS",
"log_id": 1234567890123,
"timestamp": 1533094517,
"cached": 0,
"result": {
    "thresholds": {
        "frr_1e-4": 0.05,
        "frr_1e-3": 0.3,
        "frr_1e-2": 0.9
    },
    "face_liveness": 0.05532243927,
    "face_list": [
        {
            "face_token": "df46f7c7db4aa09a093c26fb8d1a8d44",
            "location": {
                "left": 328.9026489,
                "top": 97.16340637,
                "width": 162,
                "height": 154,
                "rotation": 32
            },
            "face_probability": 1,
            "angle": {
                "yaw": 10.16196251,
                "pitch": 2.244354248,
                "roll": 33.82199097
            },
            "liveness": {
                "livemapscore": 0.04492170034
            },
            "age": 23
        },
        {
            "face_token": "901d2c64274fccd687d311a6e6110a01",
            "location": {
                "left": 411.4876404,
                "top": 166.3593445,
                "width": 329,
                "height": 308,
                "rotation": 45
            },
            "face_probability": 0.9194830656,
            "angle": {
                "yaw": -1.716423035,
                "pitch": 7.344647408,
                "roll": 45.79914856
            },
            "liveness": {
                "livemapscore": 0.001787073661
            },
```

```
            },
            "age": 23,
        },
        {
            "face_token": "7d57e36981c48b4946eb97c8d838b02a",
            "location": {
                "left": 161.4559937,
                "top": 199.8726501,
                "width": 218,
                "height": 201,
                "rotation": -1
            },
            "face_probability": 1,
            "angle": {
                "yaw": -8.187754631,
                "pitch": 6.973727226,
                "roll": -1.25429821
            },
            "liveness": {
                "livemapscore": 0.05532243927
            },
            "age": 23
        }
    ]
}
}
```


- 活体阈值说明

阈值	误拒率	通过率	攻击拒绝率
0.05	0.01%	99.99%	63.91%
0.3	0.1%	99.9%	90.33%
0.9	1%	99%	97.56%

- 阈值（Threshold）：高于此数值，则可判断为活体。（视用户的业务场景需求，选用合适的阈值）
- 误拒率（FRR）：如0.5%，指1000次真人请求，会有5次因为活体分数低于阈值被错误拒绝。
- 通过率（TAR）：如99%，指100次真人请求，会有99次因为活体分数高于阈值而通过。
- 攻击拒绝率（TRR）：如99%，代表100次作弊假体攻击，会有99次被拒绝。

2.2 人脸管理接口

2.2.1 人脸注册

> 向人脸库中添加人脸(如果group,uid不存在, 则会自动创建用户组和注册用户) 组下单个用户的人脸数目限制为20张（如果不同组下有同一个user_id， 每个组的user_id下的人脸数目都是限制20，不会合并计算） **一个用户组中只能有一个唯一的face_token**

- 请求路径:

/face-api/v3/face/add

- URL请求参数

参数	必选	类型	说明
appid	是	string	app标识 （由数字、字母、下划线组成），长度限制48B

- Header：

参数	值
Content-Type	application/json

- Body请求参数

参数	必选	类型	说明
image	是	string	图片信息(**总数据大小应小于10M**)，图片上传方式根据image_type来判断。 注：组内每个uid下的人脸图片数目上限为20张
image_type	是	string	图片类型 **BASE64**:图片的base64值，base64编码后的图片数据，编码后的图片大小不超过2M； **FACE_TOKEN**：人脸图片的唯一标识，调用人脸检测接口时，会为每个人脸图片赋予一个唯一的FACE_TOKEN，同一张图片多次检测得到的FACE_TOKEN是同一个。
group_id	是	string	用户组id，标识一组用户（由数字、字母、下划线组成），长度限制48B。 **产品建议**：根据您的业务需求，可以将需要注册的用户，按照业务划分，分配到不同的group下，例如按照会员手机尾号作为groupid，用于刷脸支付、会员计费消费等，这样可以尽可能控制每个group下的用户数与人脸数，提升检索的准确率 单个group建议80W人脸上限
user_id	是	string	用户id（由数字、字母、下划线组成），长度限制48B
user_info	否	string	用户资料，长度限制256B 默认空
quality_control	否	string	图片质量控制 **NONE**: 不进行控制 **LOW**:较低的质量要求 **NORMAL**: 一般的质量要求 **HIGH**: 较高的质量要求 **默认 NONE** 若图片质量不满足要求，则返回结果中会提示质量检测失败
liveness_control	否	string	活体检测控制 **NONE**: 不进行控制 **LOW**:较低的活体要求(高通过率 低攻击拒绝率) **NORMAL**: 一般的活体要求(平衡的攻击拒绝率, 通过率) **HIGH**: 较高的活体要求(高攻击拒绝率 低通过率) **默认NONE** 若活体检测结果不满足要求，则返回结果中会提示活体检测失败
action_type	否	string	**操作方式** APPEND: 当user_id在库中已经存在时，对此user_id重复注册时，新注册的图片默认会追加到该user_id下 REPLACE：当对此user_id重复注册时,则会用新图替换库中该user_id下所有图片 默认使用APPEND

若为**视频监控模型**，请求参数可以增加下列字段：

> 若您的前端摄像头是抓拍机抓拍并处理过的单人脸小图模式，推荐您在人脸检测/人脸注册/人脸更新/人脸搜索接口中使用

> SNAP参数，若为普通摄像头及混合场景模式，使用COMMON参数即可

参数	必选	类型	说明
image_spec	否	string	图片规格 服务会对不同规格的图片采用不同的图片处理方式 **COMMON**: 通用图片规格 默认的处理方式(如无适应的图片规格则使用此参数) **SNAP**: 从人脸抓拍机得到的抓拍图 适用于**小图片+单人脸**的情况 **默认COMMON**

> **说明**：人脸注册完毕后，生效时间**一般为5s以内**，之后便可以进行人脸搜索或认证操作。

- 请求示例

```
{
    "image": "027d8308a2ec665acb1bdf63e513bcb9",
    "image_type": "FACE_TOKEN",
    "group_id": "group_repeat",
    "user_id" : "user1",
    "user_info" : "abc",
    "quality_control": "LOW",
    "liveness_control": "NORMAL"
}
```

- 返回结果

字段	必选	类型	说明
face_token	是	string	人脸图片的唯一标识
location	是	array	人脸在图片中的位置
+left	是	double	人脸区域离左边界的距离
+top	是	double	人脸区域离上边界的距离
+width	是	double	人脸区域的宽度
+height	是	double	人脸区域的高度
+rotation	是	int64	人脸框相对于竖直方向的顺时针旋转角，[-180,180]

> **每个face_token在用户组下都是唯一的，即同一张人脸无法在同一个用户组下注册多次**

- 返回示例

```
{
"error_code": 0,
"error_msg": "SUCCESS",
"log_id": 1234567890123,
"timestamp": 1533094602,
"cached": 0,
"result": {
    "face_token": "2fa64a88a9d5118916f9a303782a97d3",
    "location": {
        "left": 117,
        "top": 131,
        "width": 172,
        "height": 170,
        "rotation": 4
    }
}
}
```


2.2.2 人脸更新

> 用于对人脸库中指定用户，更新其下的人脸图像。
> **说明**：针对一个user_id执行更新操作，新上传的人脸图像将覆盖该group_id中user_id的原有所有图像。

- 请求路径:

/face-api/v3/face/update

- URL请求参数

参数	必选	类型	说明
appid	是	string	app标识 （由数字、字母、下划线组成），长度限制48B

- Header：

参数	值
Content-Type	application/json

- Body请求参数

参数	必选	类型	说明
image	是	string	图片信息(**总数据大小应小于10M**)，图片上传方式根据image_type来判断
image_type	是	string	图片类型 **BASE64**:图片的base64值，base64编码后的图片数据，编码后的图片大小不超过2M； **FACE_TOKEN**: 人脸图片的唯一标识
group_id	是	string	用户组id，标识一组用户（由数字、字母、下划线组成），长度限制128B
user_id	是	string	用户id（由数字、字母、下划线组成），长度限制48B
user_info	否	string	用户资料，长度限制48B 默认空
quality_control	否	string	图片质量控制 **NONE**: 不进行控制 **LOW**:较低的质量要求 **NORMAL**: 一般的质量要求 **HIGH**: 较高的质量要求 **默认 NONE** 若图片质量不满足要求，则返回结果中会提示质量检测失败
liveness_control	否	string	活体检测控制 **NONE**: 不进行控制 **LOW**:较低的活体要求(高通过率 低攻击拒绝率) **NORMAL**: 一般的活体要求(平衡的攻击拒绝率, 通过率) **HIGH**: 较高的活体要求(高攻击拒绝率 低通过率) **默认NONE** 若活体检测结果不满足要求，则返回结果中会提示活体检测失败
action_type	否	string	**操作方式** UPDATE: 会使用新图替换库中该user_id下所有图片, 若user_id不存在则会报错 REPLACE：当user_id不存在时, 则会注册这个user_id的用户 默认使用UPDATE

若为**视频监控模型**，请求参数可以增加下列字段：

> 若您的前端摄像头是抓拍机抓拍并处理过的单人脸小图模式，推荐您在人脸检测/人脸注册/人脸更新/人脸搜索接口中使用SNAP参数，若为普通摄像头及混合场景模式，使用COMMON参数即可

参数	必选	类型	说明
image_spec	否	string	图片规格 服务会对不同规格的图片采用不同的图片处理方式 **COMMON**: 通用图片规格 默认的处理方式(如无适应的图片规格则使用此参数) **SNAP**: 从人脸抓拍机得到的抓拍图 适用于**小图片+单人脸**的情况 **默认COMMON**

- 请求示例

```
{
    "image": "027d8308a2ec665acb1bdf63e513bcb9",
    "image_type": "FACE_TOKEN",
    "group_id": "group_repeat",
    "user_id" : "user1",
    "user_info" : "cba",
    "quality_control": "LOW",
    "liveness_control": "NORMAL"
}
```

- 返回结果

字段	必选	类型	说明
log_id	是	uint64	请求标识码，随机数，唯一
face_token	是	string	人脸图片的唯一标识
location	是	array	人脸在图片中的位置
+left	是	double	人脸区域离左边界的距离
+top	是	double	人脸区域离上边界的距离
+width	是	double	人脸区域的宽度
+height	是	double	人脸区域的高度
+rotation	是	int64	人脸框相对于竖直方向的顺时针旋转角，[-180,180]

- 返回示例

```
{
 "error_code": 0,
 "error_msg": "SUCCESS",
 "log_id": 1234567890123,
 "timestamp": 1533094612,
 "cached": 0,
 "result": {
    "face_token": "2fa64a88a9d5118916f9a303782a97d3",
    "location": {
        "left": 117,
        "top": 131,
        "width": 172,
        "height": 170,
        "rotation": 4
    }
 }
}
```

2.2.3 人脸列表

> 获取一个用户下的人脸列表

- 请求路径

/face-api/v3/face/list

- URL请求参数

参数	必选	类型	说明
appid	是	string	app标识 （由数字、字母、下划线组成），长度限制48B

- Header：

参数	值
Content-Type	application/json

- Body请求参数

参数	必选	类型	说明
user_id	是	string	用户id（由数字、字母、下划线组成），长度限制4B
group_id	是	string	用户组id（由数字、字母、下划线组成） 长度限制48B

- 请求示例

```
{
  "user_id": "user1",
  "group_id": "group1"
}
```

- 返回结果

字段	必选	类型	说明
face_list	是	array	人脸信息列表
+face_token	是	string	人脸标志
+ctime	是	string	人脸创建时间

- 返回示例

```
{
  "error_code": 0,
  "error_msg": "SUCCESS",
  "log_id": 1234567890123,
  "timestamp": 1533094619,
  "cached": 0,
  "result": {
     "face_list": [
        {
           "face_token": "fid1",
           "ctime": "2018-01-01 00:00:00"
        },
        {
           "face_token": "fid2",
           "ctime": "2018-01-01 10:00:00"
        }
     ]
  }
}
```

2.2.4 删除人脸

> 删除用户下的某一张人脸 如果该用户下没有其他人脸了则同时删除用户

- 请求路径

/face-api/v3/face/delete

- URL请求参数

参数	必选	类型	说明
appid	是	string	app标识 （由数字、字母、下划线组成），长度限制48B

- Header：

参数	值
Content-Type	application/json

- Body请求参数

参数	必选	类型	说明
user_id	是	string	用户id（由数字、字母、下划线组成），长度限制48B
group_id	是	string	用户组id（由数字、字母、下划线组成） 长度限制48B
face_token	是	string	人脸id（由数字、字母、下划线组成）长度限制64B

- 请求示例

```
{
  "user_id": "user1",
  "group_id": "group1",
  "face_token": "2fa64a88a9d5118916f9a303782a97d3"
}
```

- 返回结果 通过返回的error_code判断是否成功 如失败则查看error_msg获得具体错误信息
- 通过返回的error_code判断是否成功 如失败则查看error_msg获得具体错误信息

```
{
  "error_code": 0,
  "error_msg": "SUCCESS",
  "log_id": 1234567890123,
  "timestamp": 1533094650,
  "cached": 0,
  "result": null
}
```

2.3 用户管理接口

2.3.1 复制用户

> 将已经存在于人脸库中的用户复制到一个新的组

- 请求路径

/face-api/v3/user/copy

- URL请求参数

参数	必选	类型	说明
appid	是	string	app标识 （由数字、字母、下划线组成），长度限制48B

- Header：

参数	值
Content-Type	application/json

- Body请求参数

参数	必选	类型	说明
user_id	是	string	用户id
src_group_id	是	string	从指定group里复制 长度限制48B
dst_group_id	是	string	需要添加用户的组id 长度限制48B

- 请求示例

```
{
  "user_id": "user1",
  "src_group_id": "group1",
  "dst_group_id": "group2"
}
```

- 返回结果 通过返回的error_code判断是否成功 如失败则查看error_msg获得具体错误信息

```
{
  "error_code": 0,
  "error_msg": "SUCCESS",
  "log_id": 1234567890123,
  "timestamp": 1533094714,
  "cached": 0,
  "result": null
}
```

2.3.2 获取用户信息

> 获取人脸库中某个用户的信息(user_info信息和用户所属的组)

- 请求路径

/face-api/v3/user/get

- URL请求参数

参数	必选	类型	说明
appid	是	string	app标识 （由数字、字母、下划线组成），长度限制48B

- Header：

参数	值
Content-Type	application/json

- Body请求参数

参数	必选	类型	说明
user_id	是	string	用户id（由数字、字母、下划线组成），长度限制48B
group_id	是	string	用户组id(由数字、字母、下划线组成，长度限制48B) 如传入**@ALL**则从所有组中查询用户信息

- 请求示例

```
{
  "user_id": "user1",
  "group_id": "group1"
}
```

- 返回结果

字段	必选	类型	说明
user_list	是	array	用户信息列表
+user_info	是	string	用户注册时携带的用户信息
+group_id	是	string	用户所属的group_id

- 返回示例

```
{
  "error_code": 0,
  "error_msg": "SUCCESS",
  "log_id": 1234567890123,
  "timestamp": 1533094729,
  "cached": 0,
  "result": {
     "user_list": [
        {
           "user_info": "user info ...",
           "group_id": "gid1"
        },
        {
           "user_info": "user info2 ...",
           "group_id": "gid2"
        }
     ]
  }
}
```

2.3.3 用户列表

> 获取用户组中的用户列表

- 请求路径

/face-api/v3/user/list

- URL请求参数

参数	必选	类型	说明
appid	是	string	app标识 （由数字、字母、下划线组成），长度限制48B

- Header：

参数	值
Content-Type	application/json

- Body请求参数

参数	必选	类型	说明
group_id	是	string	用户组id
start	否	uint32	默认值0，起始序号
length	否	uint32	返回数量，默认值100，最大值1000

- 请求示例

```
{
 "group_id": "group1"
}
```

- 返回结果

字段	必选	类型	说明
user_id_list	是	array	用户ID列表

- 返回示例

```
{
  "error_code": 0,
  "error_msg": "SUCCESS",
  "log_id": 1234567890123,
  "timestamp": 1533094739,
  "cached": 0,
  "result": {
     "user_id_list": [
        "uid1",
        "uid2"
     ]
  }
}
```

2.3.4 删除用户

> 将用户从某个组中删除

- 请求路径

/face-api/v3/user/delete

- URL请求参数

参数	必选	类型	说明
appid	是	string	app标识 （由数字、字母、下划线组成），长度限制48B

- Header：

参数	值
Content-Type	application/json

- Body请求参数

参数	必选	类型	说明
group_id	是	string	用户组id(由数字、字母、下划线组成，长度限制48B) 如传入**@ALL**则从所有组中删除用户
user_id	是	string	用户id(由数字、字母、下划线组成，长度限制48B)

- 请求示例

```
{
  "user_id": "user1",
  "group_id": "group1"
}
```

- 返回结果 通过返回的error_code判断是否成功 如失败则查看error_msg获得具体错误信息

```
{
  "error_code": 0,
  "error_msg": "SUCCESS",
  "log_id": 1234567890123,
  "timestamp": 1533094766,
  "cached": 0,
  "result": null
}
```

2.4 用户组管理接口

2.4.1 创建用户组

> 创建一个空的用户组
> **如果用户组已存在 则返回错误**

- 请求路径

/face-api/v3/group/add

- URL请求参数

参数	必选	类型	说明
appid	是	string	app标识 （由数字、字母、下划线组成），长度限制48B

- Header：

参数	值
Content-Type	application/json

- Body请求参数

参数	必选	类型	说明
group_id	是	string	group标识 （由数字、字母、下划线组成），长度限制48B

- 请求示例

```
{
  "group_id": "group1"
}
```

- 返回结果 通过返回的error_code判断是否成功 如失败则查看error_msg获得具体错误信息

```
{
  "error_code": 0,
  "error_msg": "SUCCESS",
  "log_id": 1234567890123,
  "timestamp": 1533094781,
  "cached": 0,
  "result": null
}
```

2.4.2 用户组列表

> 获取人脸库中用户组的列表

- 请求路径

/face-api/v3/group/list

- URL请求参数

参数	必选	类型	说明
appid	是	string	app标识 （由数字、字母、下划线组成），长度限制48B

- Header：

参数	值
Content-Type	application/json

- Body请求参数

参数	必选	类型	说明
start	否	uint32	默认值0，起始序号
length	否	uint32	返回数量，默认值100，最大值1000

- 请求示例

```
  {
    "start": 0,
    "length": 100
  }
```

- 返回结果

字段	必选	类型	说明
group_id_list	是	array	group

- 返回示例

```
{
 "error_code": 0,
 "error_msg": "SUCCESS",
 "log_id": 1234567890123,
 "timestamp": 1533094800,
 "cached": 0,
 "result": {
    "group_id_list": [
       "gid1",
       "gid2"
    ]
 }
}
```

2.4.3 删除用户组

> 删除用户组下所有的用户信息及人脸信息 **如果组不存在 则返回错误** 注：使用**删除组接口**前需要先绑定定时脚本

```
#### 组删除脚本(部署目录如果不是`/home/work`，记得修改到部署目录下)
*/10 * * * * cd /home/work/odp && ./hhvm/bin/hhvm app/face-api/script/DeleteGroup.php
```

- 路径

/face-api/v3/group/delete

- URL请求参数

参数	必选	类型	说明
appid	是	string	app标识 （由数字、字母、下划线组成），长度限制48B

- Header：

参数	值
Content-Type	application/json

- Body请求参数

请求参数	必选	类型	说明
group_id	是	string	group标识 （由数字、字母、下划线组成），长度限制48B

- 返回结果 通过返回的error_code判断是否成功 如失败则查看error_msg获得具体错误信息

> 注：
> 用户组删除为异步操作，通过定时任务每5min执行一次
> 1、删除1W人脸库大约需要70s；
> 2、删除5W人脸库大约需要3min；
> 3、删除10W人脸库大约需要4min；


```
{
  "error_code": 0,
  "error_msg": "SUCCESS",
  "log_id": 1234567890123,
  "timestamp": 1533094791,
  "cached": 0,
  "result": null
}
```


2.4.4 查询人脸组数量

- 请求路径: /face-api/facehub/statapp
- Header：

参数	值
Content-Type	application/json

- Body请求参数

请求参数	类型	说明
op_app_id_list	string	appid

- 返回结果


```
{
  "appid1": {
     "group_num": 1
}
```


批量获取组统计信息

- 请求路径 ：/face-api/facehub/statgroup
- Header：

参数	值
Content-Type	application/json

- Body请求参数

请求参数	类型	说明
op_app_id	string	appid标识
group_id_list	string	groupid列表 多个按,分隔 **限制20个**

- 返回结果


```
{
  "gid1": {
     "user_num": 3,
     "face_num": 5,
     "ctime": "2017-11-30 00:00:00",
     "utime": "2017-11-30 12:00:00"
  },
  "gid2": {
     "user_num": 3,
     "face_num": 5,
     "ctime": "2017-11-30 00:00:00",
     "utime": "2017-11-30 12:00:00"
  }
}
```


2.5 人脸库管理接口

2.5.1 创建人脸库（APPID）

> 人脸识别私有化服务中没有单独创建人脸库（Appid）的接口，如果需要创建人脸库，可以直接调用创建人脸组接口实现。创建方式为：在“创建用户组”接口中可以自定义一个Appid，当组创建成功后，此Appid即可生效。请参考「创建用户组」接口文档。

系统会默认生成一个default的人脸库Appid。

2.5.2 查询appid列表

> 列出所有的appid

- 请求路径 /face-api/v3/app/list
- 请求参数：无
- 返回结果

字段	必选	类型	说明
app_id_list	是	array	appid

- 返回示例

```
{
  "error_code": 0,
  "error_msg": "SUCCESS",
  "log_id": 774558169,
  "timestamp": 1559027574,
  "cached": 0,
  "result": {
    "app_id_list": [
      "516978270",
      "516978277",
      "516978100"
    ]
  }
}
```


3.错误码对照表

错误码	错误信息	说明	处理建议
222001	param[] is null	必要参数未传入	参考API说明文档，修改参数
222002	param[start] format error	参数格式错误	参考API说明文档，修改参数
222003	param[length] format error	参数格式错误	参考API说明文档，修改参数
222004	param[op_app_id_list] format error	参数格式错误	参考API说明文档，修改参数
222005	param[group_id_list] format error	参数格式错误	参考API说明文档，修改参数
222006	group_id format error	参数格式错误	参考API说明文档，修改参数
222007	uid format error	参数格式错误	参考API说明文档，修改参数
222008	face_id format error	参数格式错误	参考API说明文档，修改参数
222009	quality_conf format error	参数格式错误	参考API说明文档，修改参数
222010	user_info format error	参数格式错误	参考API说明文档，修改参数
222011	param[uid_list] format error	参数格式错误	参考API说明文档，修改参数
222012	param[op_app_id] format error	参数格式错误	参考API说明文档，修改参数
222013	param[image] format error	参数格式错误	参考API说明文档，修改参数
222014	param[app_id] format error	参数格式错误	参考API说明文档，修改参数
222015	param[image_type] format error	参数格式错误	参考API说明文档，修改参数
222016	param[max_face_num] format error	参数格式错误	参考API说明文档，修改参数
222017	param[face_field] format error	参数格式错误	参考API说明文档，修改参数
222018	param[user_id] format error	参数格式错误	参考API说明文档，修改参数
222019	param[quality_control] format error	参数格式错误	参考API说明文档，修改参数

	format error		
222020	param[liveness_control] format error	参数格式错误	参考API说明文档，修改参数
222021	param[max_user_num] format error	参数格式错误	参考API说明文档，修改参数
222022	param[id_card_number] format error	参数格式错误	参考API说明文档，修改参数
222023	param[name] format error	参数格式错误	参考API说明文档，修改参数
222024	param[face_type] format error	参数格式错误	参考API说明文档，修改参数
222025	param[face_token] format error	参数格式错误	参考API说明文档，修改参数
222026	param[max_star_num] format error	参数格式错误	参考API说明文档，修改参数
222027	code length param error	验证码长度错误 (最小值大于最大值)	参考API说明文档，修改参数
222028	param[min_code_length] format error	参数格式错误	参考API说明文档，修改参数
222029	param[max_code_length] format error	参数格式错误	参考API说明文档，修改参数
222030	param[match_threshold] format error	参数格式错误	参考API说明文档，修改参数
222200	request body should be json format	该接口需使用 application/json的 格式进行请求	请修改接口格式为： application/json
222201	network not available	服务端请求失败	重新尝试
222202	pic not has face	图片中没有人脸	检查图片质量
222203	image check fail	无法解析人脸	检查图片质量
222204	image_url_download_fail	从图片的url下载 图片失败	请确认url可公网访问
222205	network not availablel	服务端请求失败	重新尝试
222206	rtse service return fail	服务端请求失败	重新尝试
222207	match user is not found	未找到匹配的用户	请确认人脸库中 是否存在此用户
222208	the number of image is incorrect	图片的数量错误	多张图片请使用 json格式传输
222209	face token not exist	face token不存在	请确认您操作的 人脸已创建成功； 若face_token未注册到 人脸库则有效期只有1小时 注册人脸库的 face_token永久有效
222210	the number of user's faces is beyond the limit	人脸库中用户下的人脸数目超过限制	当前每个用户下限制人脸数目最大20张
222300	add face fail	人脸图片添加失败	重新尝试
222301	get face fail	获取人脸图片失败	请重新尝试， 如果持续出现此类错误， 请提交工单

222302	system error	服务端请求失败	重新尝试
222303	get face fail	获取人脸图片失败	请重新尝试， 如果持续出现此类错误，请提交工单
223100	group is not exist	操作的用户组不存在	请确认您操作的 用户组已创建成功
223101	group is already exist	该用户组已存在	请不要重复创建用户组
223102	user is already exist	该用户已存在	请不要重复创建用户
223103	user is not exist	找不到该用户	请确认您操作的 用户已创建成功
223104	group_list is too large	group_list包含组 数量过多	请按照文档提示 设置group_list参数
223105	face is already exist	该人脸已存在	请不要重复添加人脸
223106	face is not exist	该人脸不存在	请确认您操作的 人脸已创建成功； 若face_token未注册到 人脸库则有效期只有1小时， 注册人脸库的 face_token永久有效
223110	uid_list is too large	uid_list包含数量过多	请按照文档提示 设置user_list参数
223111	dst group not exist	目标用户组不存在	请确认您操作的 用户组已创建成功
223112	quality_conf format error	quality_conf格式不正确	请按照文档提示设置 quality_conf参数
223113	face is covered	人脸有被遮挡	提示用户请勿遮挡面部
223114	face is fuzzy	人脸模糊	人脸图片模糊， 前端页面可以提示 用户拍摄时不要晃动手机
223115	face light is not good	人脸光照不好	提示用户到光线适宜的地方拍摄
223116	incomplete face	人脸不完整	提示用户请勿遮挡面部
223117	app_list is too large	app_list包含app数量 过多	请按照文档提示设置 app_list参数
223118	quality control error	质量控制项错误	请按照文档提示设置 质量控制参数
223119	liveness control item error	活体控制项错误	请按照文档提示设置 活体控制参数
223120	liveness check fail	活体检测未通过	此次活体检测结果为非活体
223121	left eye is occlusion	质量检测未通过 左眼 遮挡程度过高	提示用户请勿遮挡左眼
223122	right eye is occlusion	质量检测未通过 右眼 遮挡程度过高	提示用户请勿遮挡右眼
223123	left cheek is occlusion	质量检测未通过 左脸 遮挡程度过高	提示用户请勿遮挡左脸颊
223124	right cheek is	质量检测未通过 右脸	提示用户请勿遮挡右脸颊

	occlusion	遮挡程度过高	提示用户请勿遮挡各脸颊
223125	chin contour is occlusion	质量检测未通过 下巴遮挡程度过高	提示用户请勿遮挡下巴
223126	nose is occlusion	质量检测未通过 鼻子遮挡程度过高	提示用户请勿遮挡鼻子
223127	mouth is occlusion	质量检测未通过 嘴巴遮挡程度过高	提示用户请勿遮挡嘴巴
222901	system busy	参数校验初始化失败	请重新尝试，若尝试多次无效，请提交工单咨询
222902	system busy	参数校验初始化失败	请重新尝试，若尝试多次无效，请提交工单咨询
222903	system busy	参数校验初始化失败	请重新尝试，若尝试多次无效，请提交工单咨询
222904	system busy	参数校验初始化失败	请重新尝试，若尝试多次无效，请提交工单咨询
222905	system busy	接口初始化失败	请重新尝试，若尝试多次无效，请提交工单咨询
222906	system busy	接口初始化失败	请重新尝试，若尝试多次无效，请提交工单咨询
222907	system busy	缓存处理失败	请重新尝试，若尝试多次无效，请提交工单咨询
222908	system busy	缓存处理失败	请重新尝试，若尝试多次无效，请提交工单咨询
222909	system busy	缓存处理失败	请重新尝试，若尝试多次无效，请提交工单咨询
222910	system busy	数据存储处理失败	请重新尝试，若尝试多次无效，请提交工单咨询
222911	system busy	数据存储处理失败	请重新尝试，若尝试多次无效，请提交工单咨询
222912	system busy	数据存储处理失败	请重新尝试，若尝试多次无效，请提交工单咨询
222913	system busy	接口初始化失败	请重新尝试，若尝试多次无效，请提交工单咨询

222914	system busy	接口初始化失败	请重新尝试， 若尝试多次无效， 请提交工单咨询
222915	system busy	后端服务连接失败	请重新尝试， 若尝试多次无效， 请提交工单咨询
222916	system busy	后端服务连接失败	请重新尝试， 若尝试多次无效， 请提交工单咨询
222304	image size is too large	图片尺寸太大	请确保图片尺寸在1920x1080以下 下
223128	group was deleting	正在清理该用户组的数据	请等该用户组清理完毕后再对 该组进行操作
222361	system busy	公安服务连接失败	请重新尝试， 若尝试多次无效， 请提交工单咨询
222046	param[template_type] format error	参数格式错误	请参考接口文档修改
222101	param[merge_degree] format error	参数格式错误	请参考API说明文档，修改参数
222102	param[face_location] format error	参数格式错误	参考API说明文档，修改参数
222307	image illegal, reason: porn	图片非法 鉴黄未通过	请重新上传合法的图片
222308	image illegal, reason: sensitive person	图片非法 含有政治敏感人物	请重新上传合法的图片
222211	template image quality reject	人脸融合失败 模板图质量不合格	请检查模板图是否符合 人脸融合文档中的质量要求
222212	merge face fail	人脸融合失败	请更换素材后重新尝试， 如果持续出现此类错误，请提交工单
223129	face not forward	人脸未面向正前方 （人脸的角度信息大于30度）	请使用面向正前方的人脸图片
222290	auth fail	feature服务鉴权失败	请检查授权服务运行正常，网络通畅
222291	qps exceed maximum	feature服务QPS超限	请降低服务访问频率
222292	auth expired	feature服务鉴权过期	请联系服务人员更新授权文件
222293	instance exceed maximum	feature服务实例数超限	请将运行实例数控制在授权数量内

错误码

私有化部署包部署成功后，即可获得与公有云基本完全相同的接口，人脸识别的相关接口将会启动，即可开始调用，若接口调用失败，则会返回Error_code，每个Error_code的说明和解决建议如下。

若您仍然有部署及调用问题，请提交工单联系百度的工作人员

总体说明

本文档整理了在部署过程中遇到的常见错误码及对应解决方法，主要分为以下几类问题：

- 返回222290错误
- 返回222915错误

- 返回222910错误
- 返回222207错误
- 错误码总览（222001--223140错误码）

常见错误码及解决方案

1. 返回222290错误

1、输入以下命令查看鉴权是否启动

```
netstat -apn|grep 8443
```

（1）如果没有启动，则启动鉴权

```
cd /home/baidu/work/c-offline-security-server/ && nohup bash start/c-offline-security-server-start.sh &
```

（2）若鉴权服务已启动，则继续排查

2、进入容器判断鉴权端口（8443端口）是否处于可访问状态

（1）进入容器

（2）curl端口是否可访问

```
curl 鉴权IP:8443
```

（3）若出现curl: (7) Failed connect to 10.233.42.40:8443; Connection refused，或者no route to host 则说明防火墙限制了8443端口，需要用户关掉防火墙，或者单独将8443端口放开

（4）若端口可以curl访问，继续排查问题

3、通过宿主机鉴权日志具体信息进一步查看鉴权问题

```
 cd /home/baidu/work/c-offline-security-server/log/  #进入鉴权日志目录
查看aipe_info+日期.log
```

具体日志排查参考常见问题文档。

若经过上述排查后，仍有问题，请提交工单联系百度工作人员进行解决

2. 返回222915错误

（1）**可能是由于防火墙的问题，容器内无法进行访问数据库服务**

排查方法：在容器中执行 curl 服务器IP:端口号

- 若出现curl: (7) Failed connect to 10.233.42.40:8443; Connection refused，则说明防火墙限制了8443端口，需要用户关掉防火墙，或者单独将8443端口放开
- 若出现下图所示内容，则说明容器能访问数据库服务，继续进行排查

（2）**也可能是指纹文件发生变化，导致鉴权失败**

排查方法：确认是否运行鉴权服务的物理机发生变化，如硬盘、网卡等

- 若发生变化，重新提取指纹，申请授权

- 如没有发生变化，则继续排查问题

（3）**检查鉴权服务是否启动**

排查方法：输入以下命令查看服务是否返回以下信息：

```
netstat -anp |grep 8443
```

- 如果启动失败则不返回任何结果，需要重新启动鉴权服务，输入以下命令进行重新启动

```
cd /home/baidu/work/c-offline-security-server/ && nohup bash start/c-offline-security-server-start.sh &
```

- 如果启动成功返回以下信息，继续进行问题排查

```
root@aipe-Inspiron-5675:~# netstat -anp |grep 8443
tcp        0      0 0.0.0.0:8443            0.0.0.0:*               LISTEN      2189/auth_server
```

（4）**检查是否超出feature-frame最大并发时，也会返回222915错误**

解决方案：用户根据自己的业务需求判断业务并发数是否超过最大并发支持

- 若超过，需要用户增加实例，减少运行模型的服务器数量、或实例数量
- 若没有超过，则继续进行问题排查

（5）**检查是否产品授权过期**

解决方案：如果您购买的是测试版的授权服务，请您到百度云控制台检查您的服务有效期

- 若已到期，可以申请延长试用期或申请正式版服务，正式版永久有效
- 若服务未到期，请提交工单联系百度工作人员进行解决

3. 返回222910错误

（1）**检查数据库服务是否启动**

解决方案：退出容器在服务器上执行以下命令

```
ps aux|grep databus
```

- 若未出现任何提示，则重新启动数据库服务
- 若出现以下内容，则说明数据库服务已启动，继续进行问题排查

```
[root@szth-gpu-szth-01 project-conf]# ps aux|grep databus
root     16746  0.0  0.0 107936   836 pts/5    S+   14:14   0:00 grep databus
idl-face  91999 39.0  1.0 11510808 2437548 ?    Sl   Aug22 6845:54 /home/idl-face/databus5535/libexec/mysqld --defaults-file=../etc/my.cnf --basedir=
/home/idl-face/databus5535 --datadir=/home/idl-face/databus5535/var --plugin-dir=/home/idl-face/databus5535/lib/plugin --log-error=/home/idl-face/dat
abus5535/log/mysql.err --pid-file=/home/idl-face/databus5535/var/mysql.pid --socket=/home/idl-face/databus5535/tmp/mysql.sock --port=5535
```

（2）**检查数据库地址或者端口是否填写错误**

解决方案：进入容器，输入以下命令查看地址和端口是否正确

```
cat /home/idl-face/odp/conf/db/cluster.conf
```

查看结果参考下图所示：

```
[..@IDC_rd]
ip          :   10.233.42.40
port        :   5535
[xdb]
```

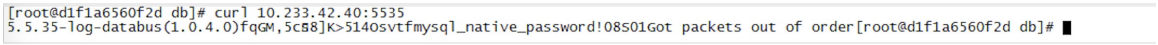
- 若地址和端口号错误，请将数据库的地址及端口号修改正确
- 若没有问题，请继续排查问题

> 所有ip和port都要改成数据库服务的地址和端口

（3）**可能是由于防火墙的问题，容器内无法访问数据库服务**

排查方法：在容器中执行 curl 服务器IP:端口号

- 若出现curl: (7) Failed connect to 10.233.42.40:5535; Connection refused，则说明防火墙限制了5535端口，需要用户关掉防火墙，或者单独将5335端口放开
- 若出现下图所示内容，则说明容器能访问数据库服务，继续进行排查



4. 返回222207错误（调用1：N接口时）

（1）**检查是否是空人脸组**

解决方案：查看该人脸组下是否有用户

- 如果人脸组下没有用户，请添加用户后继续重新调取接口访问
- 如果人脸组下有用户，或添加用户后仍然返回222207错误提示，则可能是数据库未正常连接，请参考下一条继续排查

（2）**检查数据库是否都正常连接**

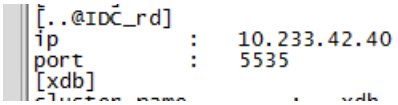
解决方案：按以下步骤查看数据库是否正常连接

（一）检查数据库地址或者端口是否填写错误

解决方案：进入容器，输入以下命令查看地址和端口是否正确

```
cat /home/idl-face/odp/conf/db/cluster.conf
```

查看结果参考下图所示：



- 若地址和端口号错误，请在文件中将数据库的地址及端口号修改正确
- 若没有问题，请继续排查问题

> 所有ip和port都要改成数据库服务的地址和端口

（二）查看数据库是否正常连接

解决方案：执行 ps aux|grep databus命令，查看数据库是否启动

```
ps aux|grep databus
```

- 如果出现图中内容，说明服务器没有启动数据库服务，如果没有启动需要手动启动数据库服务，并且重新启动容器

```
[root@nanhangface ~]# ps aux|grep databus
root      33312  0.0  0.0 112716  2392 pts/2    S+   16:59   0:00 grep --color=auto databus
```

手动启动数据库服务：

a. 切换到idl-face用户

```
su idl-face
```

b. 进入databus5535目录

```
cd /home/idl-face/databus5535/bin/
```

c. 启动服务

```
nohup ./mysqld_safe --defaults-file=../etc/my.cnf &
```

重新启动容器：

a. 使用以下命令查看容器ID

```
docker ps -a
```

回显的CONTAINER ID就是容器ID。

b. 使用以下命令重启服务

```
docker restart <容器ID>
```

- 若返回以下内容，则说明数据库连接正常，继续排查问题。

```
[root@szth-gpu-szth-01 project-conf]# ps aux|grep databus
root      16746  0.0  0.0 107936   836 pts/5    S+   14:14   0:00 grep databus
idl-face  91999 39.0  1.0 11510808 2437548 ?    Sl   Aug22 6845:54 /home/idl-face/databus5535/libexec/mysqld --defaults-file=../etc/my.cnf --basedir=
/home/idl-face/databus5535 --datadir=/home/idl-face/databus5535/var --plugin-dir=/home/idl-face/databus5535/lib/plugin --log-error=/home/idl-face/dat
abus5535/log/mysql.err --pid-file=/home/idl-face/databus5535/var/mysql.pid --socket=/home/idl-face/databus5535/tmp/mysql.sock --port=5535
```

（三）检查是否是数据异常

解决方案：需要进入/home/idl-face/public_bridge/log/目录查看bridge.INFO文件

- 若返回下图内容，则说明数据异常

```
[root@a0c5a22a78d1 log]# tail -n 100 bridge.INFO
E0904 14:48:29.879928   144 databus_bridge.cc:145] connect mysql databus faild, times = 426, 2003 Can't connect to MySQL server on '10.233.42.40' (111)
E0904 14:48:29.880087   144 databus_bridge.cc:90] reconnect to databus error
E0904 14:48:30.880309   144 databus_bridge.cc:145] connect mysql databus faild, times = 427, 2003 Can't connect to MySQL server on '10.233.42.40' (111)
E0904 14:48:30.880462   144 databus_bridge.cc:90] reconnect to databus error
E0904 14:48:31.880713   144 databus_bridge.cc:145] connect mysql databus faild, times = 428, 2003 Can't connect to MySQL server on '10.233.42.40' (111)
E0904 14:48:31.884215   144 databus_bridge.cc:90] reconnect to databus error
E0904 14:48:32.884446   144 databus_bridge.cc:145] connect mysql databus faild, times = 429, 2003 Can't connect to MySQL server on '10.233.42.40' (111)
E0904 14:48:32.884601   144 databus_bridge.cc:90] reconnect to databus error
E0904 14:48:33.884879   144 databus_bridge.cc:145] connect mysql databus faild, times = 430, 2003 Can't connect to MySQL server on '10.233.42.40' (111)
E0904 14:48:33.885094   144 databus_bridge.cc:90] reconnect to databus error
E0904 14:48:34.885412   144 databus_bridge.cc:145] connect mysql databus faild, times = 431, 2003 Can't connect to MySQL server on '10.233.42.40' (111)
E0904 14:48:34.885592   144 databus_bridge.cc:90] reconnect to databus error
E0904 14:48:35.885839   144 databus_bridge.cc:145] connect mysql databus faild, times = 432, 2003 Can't connect to MySQL server on '10.233.42.40' (111)
E0904 14:48:35.885965   144 databus_bridge.cc:90] reconnect to databus error
```

需要执行以下操作步骤：

1）进入容器,切换用户

```
 docker ps -a #查看所有容器
docker exec -it (container_id) /bin/bash #进入到container_id容器
切换用户: su idl-face
```

2）关闭bridge服务

```
cd /home/idl-face/public_bridge/bin
./bridge.sh stop
```

3）更新数据位置

```
cd /home/idl-face/public_bridge
bash update-pos.sh
```

4）启动bridge服务

```
cd /home/idl-face/public_bridge/bin
./bridge.sh start
```

（四）重启aise服务或者提交工单咨询百度工作人员

进入容器,切换用户

```
docker ps -a #查看所有容器
docker exec -it (container_id) /bin/bash #进入到container_id容器
切换用户: su idl-face
```

进入/home/idl-face/aise-service/bin目录，关闭aise服务

```
cd /home/idl-face/aise-service/bin
./face-aise_control stop
```

更新数据

```
bash update-data.sh
```

启动aise服务

```
./face-aise_control start
```

错误码总览

错误码	错误信息	说明	处理建议
222001	param[] is null	必要参数未传入	参考API说明文档，修改参数
222002	param[start] format error	参数格式错误	参考API说明文档，修改参数
222003	param[length] format error	参数格式错误	参考API说明文档，修改参数
222004	param[op_app_id_list] format error	参数格式错误	参考API说明文档，修改参数
222005	param[group_id_list] format error	参数格式错误	参考API说明文档，修改参数
222006	group_id format error	参数格式错误	参考API说明文档，修改参数
222007	uid format error	参数格式错误	参考API说明文档，修改参数
222008	face_id format error	参数格式错误	参考API说明文档，修改参数
222009	quality_conf format error	参数格式错误	参考API说明文档，修改参数

222010	user_info format error	参数格式错误	参考API说明文档，修改参数
222011	param[uid_list] format error	参数格式错误	参考API说明文档，修改参数
222012	param[op_app_id] format error	参数格式错误	参考API说明文档，修改参数
222013	param[image] format error	参数格式错误	参考API说明文档，修改参数
222014	param[app_id] format error	参数格式错误	参考API说明文档，修改参数
222015	param[image_type] format error	参数格式错误	参考API说明文档，修改参数
222016	param[max_face_num] format error	参数格式错误	参考API说明文档，修改参数
222017	param[face_field] format error	参数格式错误	参考API说明文档，修改参数
222018	param[user_id] format error	参数格式错误	参考API说明文档，修改参数
222019	param[quality_control] format error	参数格式错误	参考API说明文档，修改参数
222020	param[liveness_control] format error	参数格式错误	参考API说明文档，修改参数
222021	param[max_user_num] format error	参数格式错误	参考API说明文档，修改参数
222022	param[id_card_number] format error	参数格式错误	参考API说明文档，修改参数
222023	param[name] format error	参数格式错误	参考API说明文档，修改参数
222024	param[face_type] format error	参数格式错误	参考API说明文档，修改参数
222025	param[face_token] format error	参数格式错误	参考API说明文档，修改参数
222026	param[max_star_num] format error	参数格式错误	参考API说明文档，修改参数
222027	code length param error	验证码长度错误 (最小值大于最大值)	参考API说明文档，修改参数
222028	param[min_code_length] format error	参数格式错误	参考API说明文档，修改参数
222029	param[max_code_length] format error	参数格式错误	参考API说明文档，修改参数
222030	param[match_threshold] format error	参数格式错误	参考API说明文档，修改参数
222200	request body should be json format	该接口需使用 application/json的 格式进行请求	请修改接口格式为： application/json
222201	network not available	服务端请求失败	重新尝试
222202	pic not has face	图片中没有人脸	检查图片质量
222203	image check fail	无法解析人脸	检查图片质量
222204	image_url_download_fail	从图片的url下载 图片失败	请确认url可公网访问
222205	network not availablel	服务端请求失败	重新尝试

222206	rtse service return fail	服务端请求失败	重新尝试
222207	match user is not found	未找到匹配的用户	请确认人脸库中是否存在此用户
222208	the number of image is incorrect	图片的数量错误	多张图片请使用json格式传输
222209	face token not exist	face token不存在	请确认您操作的人脸已创建成功；若face_token未注册到人脸库则有效期只有1小时注册人脸库的face_token永久有效
222210	the number of user's faces is beyond the limit	人脸库中用户下的人脸数目超过限制	当前每个用户下限制人脸数目最大20张
222300	add face fail	人脸图片添加失败	重新尝试
222301	get face fail	获取人脸图片失败	请重新尝试，如果持续出现此类错误，请提交工单
222302	system error	服务端请求失败	重新尝试
222303	get face fail	获取人脸图片失败	请重新尝试，如果持续出现此类错误，请提交工单
223100	group is not exist	操作的用户组不存在	请确认您操作的用户组已创建成功
223101	group is already exist	该用户组已存在	请不要重复创建用户组
223102	user is already exist	该用户已存在	请不要重复创建用户
223103	user is not exist	找不到该用户	请确认您操作的用户已创建成功
223104	group_list is too large	group_list包含组数量过多	请按照文档提示设置group_list参数
223105	face is already exist	该人脸已存在	请不要重复添加人脸
223106	face is not exist	该人脸不存在	请确认您操作的人脸已创建成功；若face_token未注册到人脸库则有效期只有1小时，注册人脸库的face_token永久有效
223110	uid_list is too large	uid_list包含数量过多	请按照文档提示设置user_list参数
223111	dst group not exist	目标用户组不存在	请确认您操作的用户组已创建成功
223112	quality_conf format error	quality_conf格式不正确	请按照文档提示设置quality_conf参数
223113	face is covered	人脸有被遮挡	提示用户请勿遮挡面部
223114	face is fuzzy	人脸模糊	人脸图片模糊，前端页面可以提示

			用户拍摄时不要晃动手机
223115	face light is not good	人脸光照不好	提示用户到光线适宜的地方拍摄
223116	incomplete face	人脸不完整	提示用户请勿遮挡面部
223117	app_list is too large	app_list包含app数量过多	请按照文档提示设置app_list参数
223118	quality control error	质量控制项错误	请按照文档提示设置质量控制参数
223119	liveness control item error	活体控制项错误	请按照文档提示设置活体控制参数
223120	liveness check fail	活体检测未通过	此次活体检测结果为非活体
223121	left eye is occlusion	质量检测未通过 左眼遮挡程度过高	提示用户请勿遮挡左眼
223122	right eye is occlusion	质量检测未通过 右眼遮挡程度过高	提示用户请勿遮挡右眼
223123	left cheek is occlusion	质量检测未通过 左脸遮挡程度过高	提示用户请勿遮挡左脸颊
223124	right cheek is occlusion	质量检测未通过 右脸遮挡程度过高	提示用户请勿遮挡右脸颊
223125	chin contour is occlusion	质量检测未通过 下巴遮挡程度过高	提示用户请勿遮挡下巴
223126	nose is occlusion	质量检测未通过 鼻子遮挡程度过高	提示用户请勿遮挡鼻子
223127	mouth is occlusion	质量检测未通过 嘴巴遮挡程度过高	提示用户请勿遮挡嘴巴
222901	system busy	参数校验初始化失败	请重新尝试，若尝试多次无效，请提交工单咨询
222902	system busy	参数校验初始化失败	请重新尝试，若尝试多次无效，请提交工单咨询
222903	system busy	参数校验初始化失败	请重新尝试，若尝试多次无效，请提交工单咨询
222904	system busy	参数校验初始化失败	请重新尝试，若尝试多次无效，请提交工单咨询
222905	system busy	接口初始化失败	请重新尝试，若尝试多次无效，请提交工单咨询
222906	system busy	接口初始化失败	请重新尝试，若尝试多次无效，请提交工单咨询
222907	system busy	缓存处理失败	请重新尝试，若尝试多次无效，请提交工单咨询

222908	system busy	缓存处理失败	请重新尝试， 若尝试多次无效， 请提交工单咨询
222909	system busy	缓存处理失败	请重新尝试， 若尝试多次无效， 请提交工单咨询
222910	system busy	数据存储处理失败	请重新尝试， 若尝试多次无效， 请提交工单咨询
222911	system busy	数据存储处理失败	请重新尝试， 若尝试多次无效， 请提交工单咨询
222912	system busy	数据存储处理失败	请重新尝试， 若尝试多次无效， 请提交工单咨询
222913	system busy	接口初始化失败	请重新尝试， 若尝试多次无效， 请提交工单咨询
222914	system busy	接口初始化失败	请重新尝试， 若尝试多次无效， 请提交工单咨询
222915	system busy	后端服务连接失败	请重新尝试， 若尝试多次无效， 请提交工单咨询
222916	system busy	后端服务连接失败	请重新尝试， 若尝试多次无效， 请提交工单咨询
222304	image size is too large	图片尺寸太大	请确保图片尺寸在1920x1080以下下
223128	group was deleting	正在清理该用户组的数据	请等该用户组清理完毕后再对 该组进行操作
222361	system busy	公安服务连接失败	请重新尝试， 若尝试多次无效， 请提交工单咨询
222046	param[template_type] format error	参数格式错误	请参考接口文档修改
222101	param[merge_degree] format error	参数格式错误	请参考API说明文档，修改参数
222102	param[face_location] format error	参数格式错误	参考API说明文档，修改参数
222307	image illegal, reason: porn	图片非法 鉴黄未通过	请重新上传合法的图片
222308	image illegal, reason: sensitive person	图片非法 含有政治敏感人物	请重新上传合法的图片
222211	template image quality reject	人脸融合失败 模板图质量不合格	请检查模板图是否符合 人脸融合文档中的质量要求
222212	merge face fail	人脸融合失败	请更换素材后重新尝试， 如果持续出现此类错误，请提交工单
223129	face not forward	人脸未面向正前方 （人脸的角度信息大于30度）	请使用面向正前方的人脸图片
222290	auth fail	feature服务鉴权失败	请检查授权服务运行正常，网络通畅

222291	qps exceed maximum	feature服务QPS超限	请降低服务访问频率
222292	auth expired	feature服务鉴权过期	请联系服务人员更新授权文件
222293	instance exceed maximum	feature服务实例数超限	请将运行实例数控制在授权数量内
223140	group user is not exist	目标用户组下找不到该用户	请在目标用户组下添加该用户后重新调取接口访问

常见问题及排查

总体说明

常见问题及排查

私有化部署包部署成功后，获得与公有云基本完全相同的接口，人脸识别的相关接口将会启动，即可开始调用。

- 在私有化部署过程中遇到的问题，可以查看此文档进行解决。
- 部署完成后，人脸接口调用遇到的问题请参考错误码文档。

总体说明

本文档整理了在部署过程中常见的问题及解决方案，主要分为以下几类问题：

- 性能及环境问题
  - 性能问题
  - 环境问题
    - 容器docker问题
    - IP地址问题
    - 日志相关问题
- 鉴权失败问题
  - 一、如何判断鉴权失败
  - 二、快速排查
  - 三、常用检查项
  - 四、222915错误码排查鉴权问题
- 安装部署问题
  - 单机部署问题
    - 安装[人脸数据库服务]
    - 一键安装鉴权和人脸服务
      - （一）检查服务器环境
      - （二）安装Docker
      - （三）安装Nvidia驱动
      - （四）安装人脸服务
    - 检查服务部署成功及日志查看
  - 多机部署问题

- 接口调用问题
  - Face_token问题
  - 其他问题
- 其他常见问题
  - Error报错排查
  - 其他问题

性能及环境问题

在私有化部署过程中遇到的性能及环境相关问题，可以查看此文档进行解决。
若文档仍未解决您的问题，请提交工单联系百度的工作人员

性能问题

一.百度官方说可以达到20QPS，为什么我调用到10个并发就提示system busy?

这个是出于保障平响的角度设计的，并不影响QPS。并发是指同时发起请求的数量，一秒内进行五次10并发的请求，就是50QPS了，如果想了解并发与QPS的关系和区别，请点击查看GitHub文档

环境问题

容器docker问题

1.docker常用命令

```
 docker ps -a #查看所有容器
docker restart (container_id)#重启container_id容器
docker exec -it (container_id) /bin/bash #进入到container_id容器
docker images #查看所有镜像
docker rm -f  容器ID  #删除容器
```

2.Docker安装过程中发生冲突

Docker需要使用从百度下载的安装部署包中的Docker，如果您本地已有Docker，需要确认Docker是否可以卸载。（如果Docker中已经装了服务就先保存下来移植到新建的部署包中的Docker中）

3.部署过程中出现docker出现no space left on device的报错

方法1：在/etc/docker/daemon.json中添加如下，然后重启docker。注意检查配置文件的json格式不能出错。

```
{
  "graph":"/home/work/docker",//路径可修改，要保证有足够的空间
  "storage-driver":"overlay2"
}
```

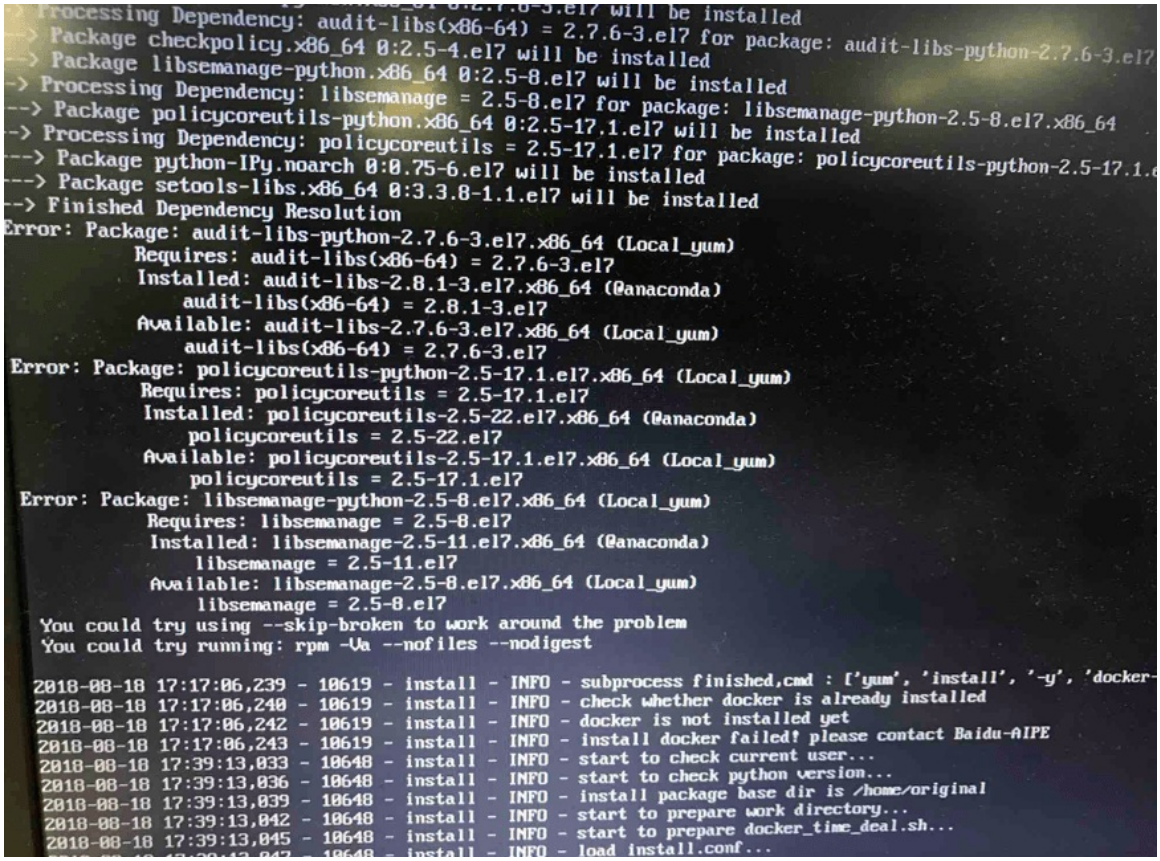
方法2：

```
 vim/usr/lib/systemd/system/docker.service
在ExecStart=/usr/bin/dockerd后面加上--graph /home/baidu/docker，保存
systemctl daemon-reload
systemctl restart docker
```

4.安装docker-ce时，依赖包冲突问题

docker-ce安装出错，出现需要安装依赖的版本比系统自带版本低的情况，导致安装失败，通过控制台日志可以看到Installed：

xxx的已安装版本信息，目前常见的依赖冲突有：audit-libs-2.7.6-3.el7.x86_64,libsemanage-2.5-8.el7.x86_64, policycoreutils-2.5-17.1.el7.x86_64，如下图所示：



解决方案：

a. 进入到/home/baidu/docker-packages目录，执行 rpm -Uvh –oldpackage {待安装的组件名.rpm}安装老版本，待安装的版本从日志Requires:xxxx(Local_yum)中获得；

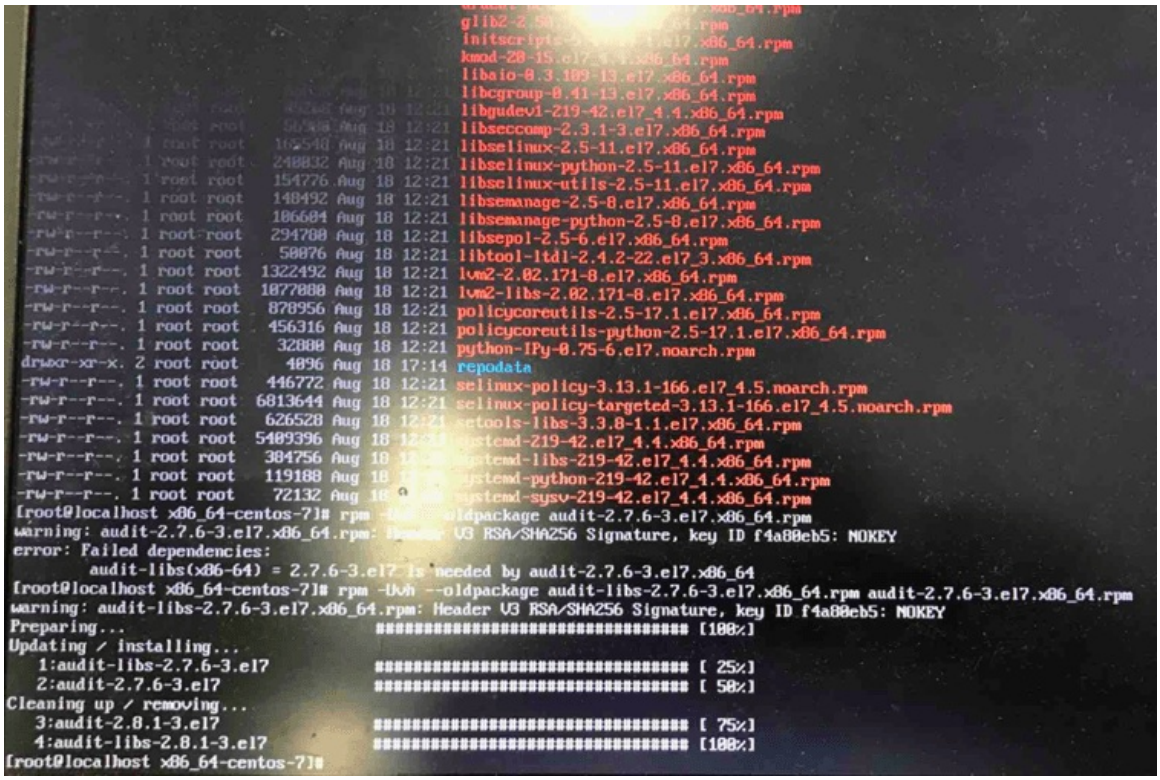
b. 如果执行rpm -Uvh –oldpackage {待安装的组件名.rpm}失败，查看日志，如果是出现日志{待安装的组件名} is needed by xxxxx,说明要删除版本被依赖着，不能直接删除，我们需要判断下依赖删除版本的服务是否在我们的本地源下.

如果是则安装替换的服务和依赖的他的服务，通过命令:

rpm -Uvh –oldpackage audit-libs-2.7.6-3.el7.x86_64.rpm audit-2.7.6-3.el7.x86_64.rpm 安装（自行替换红色的组件名）；

如果不是我们的本地源需要的，直接通过命令：

rpm -ivh --force --nodeps {待安装的组件名}进行替换版本，注意该操作有风险，需要谨慎判断。



5.部署过程中出现docker出现no space left on device的报错

两种方法，在/etc/docker/daemon.json中添加如下，然后重启docker。注意检查配置文件的json格式不能出错。 1、

```
 {
   "graph": "/home/work/docker",//路径可修改，要保证有足够的空间
   "storage-driver": "overlay2"
}
```

2、

```
 vim /usr/lib/systemd/system/docker.service
在ExecStart=/usr/bin/dockerd后面加上--graph /home/baidu/docker，保存
systemctl daemon-reload
systemctl restart docker
```

IP地址问题

1.**IP地址发生变化的问题**

（1）需要修改以下鉴权服务和人脸应用服务的IP地址

鉴权服务IP地址修改：/home/baidu/work/c-offline-security-server/conf/server.conf （修改后需要重启鉴权服务）

人脸服务IP地址修改：/home/baidu/work/face-server/project-conf/easypack_init.sh

人脸服务连接数据库IP地址修改：/home/baidu/work/face-server/project-conf/sconf/service.conf中MYSQL_SVC_SERVICE_HOST项

> 注：以上鉴权服务的修改针对离线鉴权服务有效，如果您使用的是在线BCC云服务器，请联系百度工作人员进行鉴权地址修改

（2）把原来的容器删掉

```
 docker ps -a #查看所有容器

docker rm -f  容器ID  #删除容器
```

（3）重新生成容器

```
 cd /home/baidu/work/face-server/project-conf/
bash multi_docker_start.sh
```

日志相关问题

1.如何查看日志

- 查看docker容器启动日志：

```
 docker logs (dockerid)
```

- 查看人脸服务启动日志：

```
 查看容器id：docker ps -a
然后进入容器查看日志: docker exec -it (dockerid) /bin/bash
容器内日志路径:
/home/idl-face/sys_start.log
/home/idl-face/feature-frame/log/service.log.wf
```

2.修改清理日志时长定时任务（默认为15天定期清理）

（1）进入容器

（2）切换idl-face用户

（3）执行命令crontab -e 修改天数

```
[idl-face@2c387c87d268 ~]$ crontab -l
#crontab
MAILTO=""

6 0 * * * find /home/idl-face/aise-service/logs/aise_service.* -type f -mtime +15 -exec rm -f {} \;
6 0 * * * find /home/idl-face/aise-service/aise-agent/logs/aise_agent.log.* -type f -mtime +15 -exec rm -f {} \;
7 0 * * * find /home/idl-face/relay_master/log/relay.log.* -type f -mtime +15 -exec rm -f {} \;
8 0 * * * find /home/idl-face/public_bridge/log/bridge.log.* -type f -mtime +15 -exec rm -f {} \;
9 0 * * * find /home/idl-face/odp/log/hhvm/hhvm-access.log.* -type f -mtime +15 -exec rm -f {} \;
9 0 * * * find /home/idl-face/odp/log/hhvm/hhvm-error.log.* -type f -mtime +15 -exec rm -f {} \;
10 0 * * * find /home/idl-face/odp/log/ral/ral* -type f -mtime +15 -exec rm -f {} \;
15 0 * * * find /home/idl-face/odp/log/face-api/face-api.log.* -type f -mtime +15 -exec rm -f {} \;
20 0 * * * find /home/idl-face/feature-frame/log/service.log.* -type f -mtime +15 -exec rm -f {} \;
*/5 * * * * cd /home/work/odp && ./hhvm/bin/hhvm app/face-api/script/DeleteGroup.php
```

鉴权失败问题

在私有化部署过程中遇到的鉴权相关问题，可以查看此文档进行解决。

若文档仍未解决您的问题，请提交工单联系百度的工作人员

一、如何判定鉴权失败

1)应用的日志显示authentication failed

2)一般容器一启动就退出，可通过docker logs 容器id 来查看容器日志，提示authentication failed字样

二、快速排查

查看/home/baidu/work/c-offline-security-server/log下的auth*server.log*和*aipe_info*日期.log。

鉴权日志	含义	解决办法
verify finger is invalid	license里的指纹错误	请确认运行鉴权服务的服务器是否有变化， 如硬盘、网卡等，如有变化， 请重新提取指纹、申请授权
verify product has not foud	产品未授权	此服务器未获得授权，无法运行人脸模型 请重新提取指纹、申请授权
verify product lc is expired	产品授权过期	请重新提取指纹、申请授权
instance_check reg fail,r_list full	实例池已满 运行模型的服务器数量 （或实例数量）超出授权上限	请减少运行模型的服务器数量、 或实例数量
Environment is unsafe	运行环境不安全	

三、常用检查项

1.检查鉴权服务进程是否存在

ps -ef|grep auth_server。服务正常启动后如下图所示：

```
root@***:~# ps -ef|grep auth_server
root 20609 20604 0 11:33 pts/2 00:00:00 sudo /home/baidu/work/c-offline-security-server/auth_server
root 20611 20609 0 11:33 pts/2 00:00:07 /home/baidu/work/c-offline-security-server/auth_server
root 22829 22616 0 12:08 pts/3 00:00:00 grep --color=auto auth_server
```

2.检查鉴权服务端口是否启动

netstat -apn|grep 8443。服务正常启动后如下图所示：

```
root@zhangwenkang-EX660:~# netstat -apn|grep 8443
tcp        0      0 0.0.0.0:8443            0.0.0.0:*               LISTEN      20611/auth_server
```

3.查看鉴权启动日志

从/home/baidu/work/c-offline-security-server/log/auth_server.log查看鉴权服务启动日志。

1）启动成功的日志如下图所示：

```
NOTICE: 07-04 15:20:15: Auth_Server * 14548 ---------- open log ----------
WARNING: 07-04 15:20:15: Auth_Server * 14548 ---------- open log.wf ----------
TRACE: 07-04 15:20:16: Auth_Server * 14548 baidu/prienv-security/auth-service-offline/http_server.cpp:279] Load config file success
TRACE: 07-04 15:20:16: Auth_Server * 14548 baidu/prienv-security/auth-service-offline/http_server.cpp:325] Aipe config: cluster conf =
192.168.1.100,192.168.1.101,192.168.1.102, ip = 12, inner port = 8091, data path = /home/baidu/work/c-offline-security-server/data, outer port = 6379
```

2）启动时常见的报错提示如下：

如下表示license文件不对，可能的原因是license的版本不对或者人为修改了license。

```
WARNING: 07-04 18:08:12: Auth_Server * 17018 baidu/prienv-security/auth-service-offline/util/io_utils.cpp:15] Open file license failed
NOTICE: 07-04 18:08:34: Auth_Server * 23584 ---------- open log ----------
WARNING: 07-04 18:08:34: Auth_Server * 23584 ---------- open log.wf ----------
WARNING: 07-04 18:08:35: Auth_Server * 23584 baidu/prienv-security/auth-service-offline/http_server.cpp:257] License decrypt error
WARNING: 07-04 18:08:35: Auth_Server * 23584 baidu/prienv-security/auth-service-offline/http_server.cpp:346] Start server, environment is unsaf
```

4.查看鉴权请求日志

从/home/baidu/work/c-offline-security-server/log/aipe*info*日期.log查看鉴权日志。根据应用里鉴权失败的时间或者根据产品名查找日志。

1）鉴权成功日志如下：

```
[2019-07-04 11:36:59.312][20611][020672][aipe_log][info] reg create instance,product=ocr,id=1eb7af0d-bece-4c80-965a-3b97555789c2,res=true
```

或者

```
[2019-07-04 17:16:35.166][16936][017535][aipe_log][info] instance_check success,product=classification-demo,id=b16480ca-92bb-4869-97e0-7da197cb2530
```

2）鉴权常见失败日志如下：

1. verify finger is invalid,product=... 机器指纹校验失败，license和当前机器指纹对不上。可能的原因是
   a. 申请license时用的指纹不是当前机器的指纹
   b. 当前机器硬件发生了变化，导致指纹变了
   c. 采集指纹时的用户和鉴权服务运行时的用户不一致
2. verify product has not foud,product=... license里没有该产品的授权。
3. verify product lc is expired,product=... 该产品授权已过期。
4. instance_check reg fail,r_list full,product=...,id=... 该产品授权的实例池已满。可能的原因是
   a. 该产品的容器数超过了授权的实例数
   b. 已停止的容器占用了授权实例池，等1分钟后实例池自动清理过期实例，再鉴权即可
   c. 应用每次重启后碰到这种情况，可能是鉴权客户端版本较老，没有把实例id持久化，需要升级应用
5. cache client get context error: Connection refused 鉴权服务连接本机缓存失败，可能的原因是
   a. 鉴权服务配置的ip和服务器实际ip不一致；
   b. 8991端口或6666端口被占用，导致缓存服务没起来；

若鉴权服务器IP地址与实际IP地址不一致，请参考以下方案进行修改： 输入以下命令，查看鉴权服务中的鉴权IP地址：

```
cat /home/baidu/work/c-offline-security-server/conf/server.conf
```

```
root@aipe-Inspiron-5675:/home/aipe# cat /home/baidu/work/c-offline-security-server/conf/server.conf
port:8443
AIPE_SEC_DT_CLUSTER_CONF:10.232.43.25
AIPE_SEC_DT_IP:10.232.43.25
AIPE_SEC_DT_PATH:/home/baidu/work/c-offline-security-server/data
```

（1）若填写IP地址与实际IP地址不相符，则修改为实际IP地址（不能填写127.0.0.1）并重启鉴权服务

停止服务：

```
cd /home/baidu/work/c-offline-security-server/ && bash start/c-offline-security-server-stop.sh
```

启动服务：

```
cd /home/baidu/work/c-offline-security-server/ && nohup bash start/c-offline-security-server-start.sh &
```

重启容器

（2）若填写IP为实际本机IP，则继续排查

5、若鉴权服务中的IP地址填写正确，则进入下面的路径查看人脸应用服务的鉴权IP地址是否填写正确

```
cat /home/baidu/work/face-server/project-conf/easypack_init.sh
```

![10e6a1ac52ce944af550a1ab72ecbd49](/Users/chaixue1/Library/Caches/BaiduMacHi/Share/images/10e6a1ac52ce944af550a1ab72ecbd49.png)

（1）若填写IP地址与实际IP地址不相符，则修改为实际IP地址（不能填写127.0.0.1）并重构容器

重构容器命令：

```
cd /home/baidu/work/face-server/project-conf/ &&bash multi_docker_start.sh
```

（2）若经过上述排查后，仍有问题，请提交工单联系百度工作人员进行解决

5.怎么重启鉴权服务

停止服务：

```
cd /home/baidu/work/c-offline-security-server/ && bash start/c-offline-security-server-stop.sh
```

启动服务：

```
cd /home/baidu/work/c-offline-security-server/ && nohup bash start/c-offline-security-server-start.sh &
```

四、222915错误码排查鉴权问题

（1）**可能是由于防火墙的问题，容器内无法进行访问数据库服务**

排查方法：在容器中执行 curl 服务器IP:端口号

- 若出现curl: (7) Failed connect to 10.233.42.40:8443; Connection refused，则说明防火墙限制了5535端口，需要用户关掉防火墙，或者单独将8443端口放开
- 若出现下图所示内容，则说明容器能访问数据库服务，继续进行排查

（2）**也可能是指纹文件发生变化，导致鉴权失败**

排查方法：确认是否运行鉴权服务的物理机发生变化，如硬盘、网卡等

- 若发生变化，重新提取指纹，申请授权
- 如没有发生变化，则继续排查问题

（3）**检查鉴权服务是否启动**

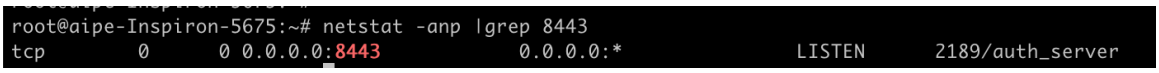
排查方法：输入以下命令查看服务是否返回以下信息：

```
netstat -anp |grep 8443
```

- 如果启动失败则不返回任何结果，需要重新启动鉴权服务，输入以下命令进行重新启动

```
cd /home/baidu/work/c-offline-security-server/ && nohup bash start/c-offline-security-server-start.sh &
```

- 如果启动成功返回以下信息，继续进行问题排查



（4）**检查是否超出feature-frame最大并发时，也会返回222915错误**

解决方案：用户根据自己的业务需求判断业务并发数是否超过最大并发支持

- 若超过，需要用户增加实例，减少运行模型的服务器数量、或实例数量
- 若没有超过，则继续进行问题排查

（5）**检查是否产品授权过期**

解决方案：如果您购买的是测试版的授权服务，请您到百度云控制台检查您的服务有效期

- 若已到期，可以申请延长试用期或申请正式版服务，正式版永久有效

- 若服务未到期，请提交工单联系百度工作人员进行解决

安装部署问题

在私有化部署过程中遇到的部署相关问题，可以查看此文档进行解决。
若文档仍未解决您的问题，请提交工单联系百度的工作人员

单机部署问题

安装[人脸数据库服务] 1、数据库连接失败，报错： Can't connect to local Mysql server through socket '/home/dil-face/databus5535/tmp/mysql.sock'

```
[root@minio-3 project-conf]# bash mysql_start.sh
更改用户 idl-face 的密码 .
passwd：所有的身份验证令牌已经成功更新。
databus start
221220 17:30:21 mysqld_safe Logging to '/home/idl-face/databus5535/log/mysql.err'.
221220 17:30:21 mysqld_safe Starting mysqld daemon with databases from /home/idl-face/databus5535/var
databus init start
databus grant ...
ERROR 2002 (HY000): Can't connect to local MySQL server through socket '/home/idl-face/databus5535/tmp/mysql.sock' (2)
ERROR 2002 (HY000): Can't connect to local MySQL server through socket '/home/idl-face/databus5535/tmp/mysql.sock' (2)
databus init finish
```

解决方案：

1）终止databus进程：

```
ps j -A |grep databus5535|grep -v "grep"
```

终止databus相关进程及其父进程
终止后请再次执行 `ps j -A |grep databus5535|grep -v "grep"` 核实，确保进程杀干净

2）卸载databus

```
##### 删除idl-face用户及其家目录，用于清理mysql底层数据
userdel -r idl-face
```

3）修改启动脚本

```
进入部署包内 original/package/Applications/face-server/project-conf/
##### 通过sed命令将sleep 40s替换为sleep 140s
sed -i 's/sleep 40s/sleep 140s/g' mysql_start.sh
```

修改后如下图所示：

```
61 cd /home/$mysql_user/databus5535/bin
62 ./mysqld_safe --defaults-file=../etc/my.cnf &
63 sleep 10s
64 exit
65 EOF
66
67 sleep 40s        更改为 sleep 140s
68
69 su idl-face -s /bin/bash ./databus_init.sh
70
71 databus_start="/home/idl-face/databus5535/sh/mysql_boot.sh"
72 if [ ! -f "/etc/rc.d/rc.local" ]
73 then
74         if [ ! -d "/etc/rc.d" ]
75         then
```

4）重新部署数据库服务

```
 bash mysql_start.sh
------------------------------出现下面结果表示安装mysql成功----------------------------
1. databus start
2. 190506 15:08:17 mysqld_safe Logging to '/home/idl-face/databus5535/log/mysql.err'.
3. 190506 15:08:17 mysqld_safe Starting mysqld daemon with databases from /home/idl-
4. face/databus5535/var
5. databus init start
6. databus grant ...
7. databus init start
8. databus init finish
```

2、数据库服务无法自动部署，如何进行手动部署并设置服务开机自动启动？

（1）首先以root用户创建idl-face新用户

```
 useradd idl-face
passwd 用户自定义
```

（2）解压original/package/Applications/face-server目录下的basepkg.tar.gz安装包，里面有databus5535目录，将其移到:/home/idl-face/databus5535/ （目录固定，不能为其他目录）

```
 mv ./databus5535 /home/idl-face/
```

（3）创建一个给mysql用的日志目录 /var/log/mariadb，并赋为 777 权限，并对 /home/idl-face/databus5535/ 赋权

```
 mkdir -p /var/log/mariadb
chmod 777 /var/log/mariadb -R
chown -R idl-face.idl-face  /home/idl-face/databus5535/
```

（4）切换到idl-face用户

```
 su idl-face
```

（5）进入databus5535目录

```
 cd /home/idl-face/databus5535/bin/
```

（6）启动服务

```
 nohup ./mysqld_safe --defaults-file=../etc/my.cnf &
```

（7）初始化数据库

```
 cd /home/idl-face/databus5535/
bin/mysql -uroot -p'Bs~XIsDDv4XcDGCt)S(+4*yjQ&8NJh' --default-character-set=utf8 < sql/database_and_grant.sql
bin/mysql -uroot -p'Bs~XIsDDv4XcDGCt)S(+4*yjQ&8NJh' --default-character-set=utf8 < sql/face.sql
```

（8）检查数据库是否启动成功

```
 ps -ef | grep mysql
```

（9）将数据库启动加入开机启动

```
 vim /etc/rd.d/rc.local
加入bash /home/idl-face/databus5535/sh/mysql_boot.sh
```

（10）将idl-face用户禁用登录 (非常重要)

```
 执行完成安装后，请一定要将idl-face用户禁用登录
方法：
1、切换到root用户
2、usermod -s /sbin/nologin idl-face
```

3、安装数据库报错error while loading shared libraries ：libncurses.so.5……

```
 报错为缺少对应的libncurses.so.5
可以yum install  libncurses.so.5或者yum install  libncurses.so.*进行安装
之后ps -ef | grep mysql 查看数据库进程id  kill掉
再执行userdel -r idl-face 将对应用户及文件删除
之后执行一键部署脚本或者手动部署即可
```

一键鉴权和人脸服务

在一键安装鉴权服务和人脸识别服务过程中主要会出现以下几类问题，大家可以根据实际部署中遇到的问题进行排查解决。

（一）检查服务器环境

1. 请保证8443端口（鉴权服务端口）不被占用

2. CPU支持AVX/AVX2以及BMI2指令集（12年后的CPU基本都支持）

环境检查问题及解决方案

类型	描述	解决方案
CPU内存环境检查	要求≥32GB，实际不满足要求	百万级人脸库内存＞16GB，执行continue命令即可
CPU指令集检查	环境检查结果显示当前系统不支持avx/avx2指令集以及bmi2指令集	需要更换硬件为可以支持avx/avx2指令集以及bmi2指令集的CPU
硬件环境检查	要求≥500GB，实际不满足要求	百万级人脸库硬盘≥100GB，执行continue命令即可
显卡检查	环境检查结果显示检查不到显卡	执行nvidia-smi确认显卡存在，执行continue命令即可
显卡驱动检查	环境检查结果显示显卡驱动不符合要求	卸载掉自己安装的显卡驱动，重新执行一键安装部署命令（一键安装过程中会安装驱动等环境，不建议客户自己安装驱动，驱动版本容易与人脸模型不兼容）

（二）安装Docker

1.Docker安装过程中发生冲突

Docker需要使用从百度下载的安装部署包中的Docker，如果您本地已有Docker，需要确认Docker是否可以卸载。（如果Docker中已经装了服务就先保存下来移植到新建的部署包中的Docker中）

2.docker启动失败， Failed to program NAT chain: ZONE_CONFLICT: 'docker0' already bound to a zone

问题现象：

通过journalctl -u docker --no-pager 命令发现发现docker报错

```
 failed to start daemon: Error initializing network controller: Error creating
default "bridge" network: Failed to program NAT chain: ZONE_CONFLICT: 'docker0'
already bound to a zone
```

解决方案：

该报错是因为防火墙的区域冲突，导致创建虚拟网络失败。解决方案如下：

```
 firewall-cmd --zone=trusted --remove-interface=docker0
firewall-cmd --reload
```

3. **部署过程中docker出现no space left on device的报错**

请参考 修改docker的默认存储路径

4. **安装docker-ce，依赖包冲突**

出现需要安装依赖的版本比系统自带版本低的情况，导致安装失败，详细报错如下

```
 --> 解决依赖关系完成
错误：软件包：libsemanage-python-2.5-8.el7.x86_64 (Local_yum)
        需要：libsemanage = 2.5-8.el7
        已安装: libsemanage-2.5-14.el7.x86_64 (@anaconda)
            libsemanage = 2.5-14.el7
        可用: libsemanage-2.5-8.el7.x86_64 (Local_yum)
            libsemanage = 2.5-8.el7
错误：软件包：audit-libs-python-2.7.6-3.el7.x86_64 (Local_yum)
        需要：audit-libs(x86-64) = 2.7.6-3.el7
        已安装: audit-libs-2.8.5-4.el7.x86_64 (@anaconda)
            audit-libs(x86-64) = 2.8.5-4.el7
        可用: audit-libs-2.7.6-3.el7.x86_64 (Local_yum)
            audit-libs(x86-64) = 2.7.6-3.el7
错误：软件包：policycoreutils-python-2.5-17.1.el7.x86_64 (Local_yum)
        需要：policycoreutils = 2.5-17.1.el7
        已安装: policycoreutils-2.5-34.el7.x86_64 (@anaconda)
            policycoreutils = 2.5-34.el7
        可用: policycoreutils-2.5-17.1.el7.x86_64 (Local_yum)
            policycoreutils = 2.5-17.1.el7
 您可以尝试添加 --skip-broken 选项来解决该问题
 您可以尝试执行：rpm -Va --nofiles --nodigest

2021-10-21 13:58:19,993 - 7100 - install - INFO - subprocess finished,cmd : ['yum', 'install', '-y', 'docker-ce']
```

解决方案： 服务器联网条件下，则可以通过在线安装解决。

```
 yum -y install docker-ce
```

如果yum安装 docker-ce 返回 no package docker-ce available，请使用如下方法解决：

```
 ##### 安装yum管理工具
yum install -y yum-utils

##### yum添加软件源
yum-config-manager \
--add-repo \
https://mirrors.ustc.edu.cn/docker-ce/linux/centos/docker-ce.repo

##### 刷新缓存
yum makecache fast

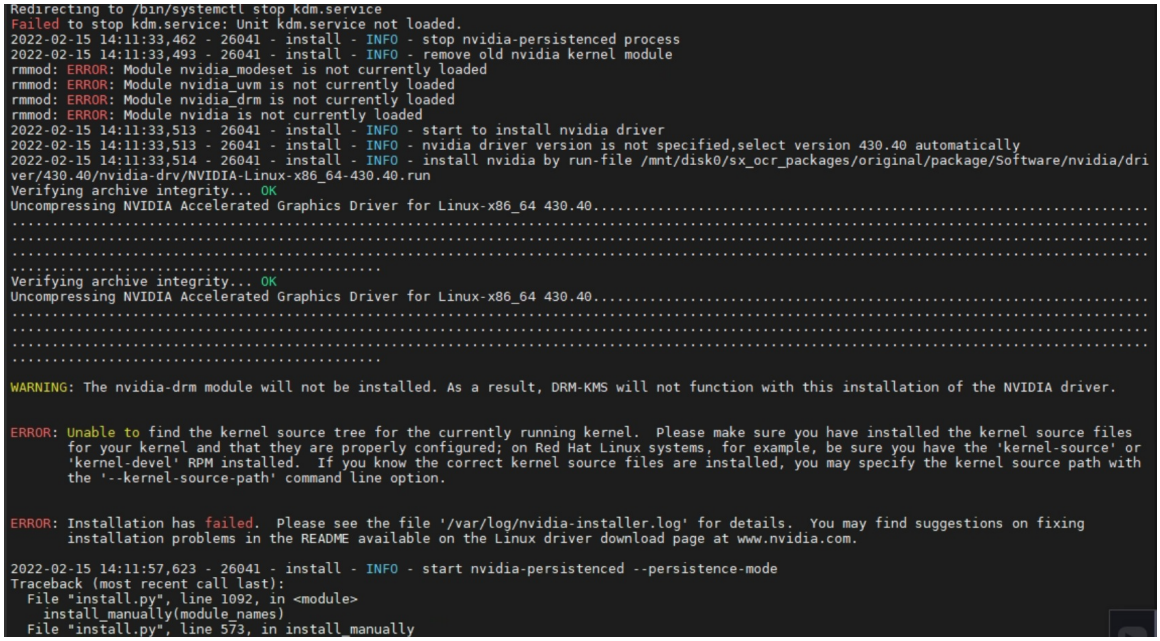
##### 安装docker-ce
yum install docker-ce
```

（三）安装Nvidia驱动

1. **安装nvidia驱动报错 unable to find the kernel source tree for the currently running kernel**

问题现象：

执行 `python install.py install nvidia` 报错如下图



解决方案：

1. 安装与内核版本对应的kernel-devel ，可在线安装：

```
yum install "kernel-devel-uname-r == $(uname -r)"
```

2）安装的时候在安装命令后面加上内核文件所在目录：

```
python install.py in nvidia --kernel-source-path=/usr/src/kernels/内核文件目录
```

3. 如果在线安装找不到包,可以访问 http://rpm.pbone.net/ 通过关键词搜索，选择与 `uname -r` 返回的内核版本一致的rpm包进行下载，安装方式

```
##### 假如 uname -r 返回 3.10.0-1160.59.1.el7.x86_64
##### 可下载对应的rpm包 kernel-devel-3.10.0-1160.59.1.el7.x86_64.rpm 进行安装
yum -y install kernel-devel-3.10.0-1160.59.1.el7.x86_64.rpm
##### 重新安装nvidia驱动
python install.py in nvidia --kernel-source-path=/usr/src/kernels/3.10.0-1160.59.1.el7.x86_64
```

2.出现ERROR: The Nouveau kernel driver is currently in use by your system

安装nvidia的过程中需要禁用nouveau，这个步骤需要重启才能生效。如果安装脚本执行完之后，最后的输出如下：

则需要重启机器。然后重新执行安装脚本即可。

3. **nvidia-docker2 安装失败**

如果当前服务器已经安装>17.06.2 版本的docker， 通过`python2 install.py inall` 或 `python2 install.py in nvidia` 安装nvidia-docker2 的话 一般会遇到与现有docker版本冲突的问题。

解决方案：

如果服务器可以联网的话,可以yum来在线安装

```
yum install -y nvidia-docker2
```

如果yum安装 nvidia-docker2 返回 `no package nvidia-docker2 available`，请使用如下方式解决：

```
distribution=$(source /etc/os-release;echo $ID$VERSION_ID) && curl -s -L https://nvidia.github.io/nvidia-docker/$distribution/nvidia-docker.repo | sudo tee /etc/yum.repos.d/nvidia-docker.repo

yum clean expire-cache
yum install -y nvidia-docker2
systemctl restart docker
```

（四）安装人脸服务

1. **安装过程中漏装人脸数据库服务**

人脸数据库服务一键安装部署请参考：安装部署方案 中单机部署-部署数据库服务。

2. **报错 face-server 安装失败，请尝试执行 cd /home/baidu/work/face-server/project-conf/ && bash multi_docker_start.sh**

问题现象：

执行`python install.py inall` 后报错 face-server 安装失败，请尝试执行 cd /home/baidu/work/face-server/project-conf/ && bash multi_docker_start.sh

解决方案：

请复制该命令手工执行，查看命令返回结果

```
cd /home/baidu/work/face-server/project-conf/ && bash multi_docker_start.sh
```

1）如果返回数据库连接失败

```
##### 重新安装数据库
ps -ef | grep mysql  # 找到对应的数据库进程kill 掉，或者pkill -u idl-face
userdel -r idl-face # 删除用户及家目录
cd original/package/Applications/face-server/project-conf/ && bash mysql_start.sh # 重新执行安装脚本
iptables -I INPUT -p tcp --dport 5535 -j ACCEPT  #（该步骤在部署mysql时有要求用户手动执行，避免遗漏，再次执行一遍）
##### 重新手动安装人脸服务
cd /home/baidu/work/face-server/project-conf/  && bash multi_docker_start.sh
```

2. 如果返回报错：“can not find docker image :300478125c1e, please check docker images”

```
##### 手动load 人脸镜像
docker load -i /home/baidu/work/face-server/docker.tar
##### 重新手动安装人脸服务
cd /home/baidu/work/face-server/project-conf/  && bash multi_docker_start.sh
```

3）如果返回报错：“can not find nvidia-smi”

```
##### 一般是nvidia安装有问题，可执行重新安装
python install.py rm nvidia && python install in nvidia
##### 重新手动安装人脸服务
cd /home/baidu/work/face-server/project-conf/  && bash multi_docker_start.sh
```

4)如果返回报错：“docker: error response from daemon: network easypack not found“

```
##### 手动创建network,名称为easypack
docker network create --driver bridge easypack
##### 重新手动安装人脸服务
cd /home/baidu/work/face-server/project-conf/  && bash multi_docker_start.sh
```

3. **"人脸应用部署后，face-service-0-0容器状态为Created"**

问题现象：

执行 `docker ps -a` 发现 face-service-0-0 容器状态为created

解决方案：

可以执行 `docker inspect ${人脸容器名}` 进一步查看顶部报错信息，如 返回 “network easypack not found”，可执行

```
docker network create --driver bridge easypack
docker restart face-service-0-0
```

多机部署问题

私有化多机部署出现问题请提交工单联系百度人员咨询解决。

文件下载问题

md5值校验失败

将部署包里refs.txt文件中下载失败的文件对应的md5改成-
再次执行bash download.sh命令即可

接口调用问题

在私有化部署过程中遇到的接口调用相关问题，可以查看此文档进行解决。

若文档仍未解决您的问题，请提交工单联系百度的工作人员

Face_token问题

1.私有化如何使用face_token?

私有化环境下想要使用Face_token,需要在生成Face_token的时候（即调用检测、注册接口时），要加face_field字段：feature，用于预先提取特征，方便使用face_token进行特征分析

2.调用detect接口生成的face_token无法用来调用比对、搜索、活体等接口，应该怎么解决？

（1）调用比对、搜索接口：调用detect、add接口检测时，要加face_field字段：feature，用于预先提取特征，方便使用face_token进行比对 （2）调用活体接口：调用detect、add接口时，使用liveness_control参数，用于预先提取活体特征，进行活体分析 （3）获取人脸属性：希望通过facetoken获取什么属性，detect的阶段就需要在field参数中增加哪个参数

3. 注册图片后，立即调用显示face_token不存在

注册后，存储face_token需要一段时间，这个时候立即调取图片，库内还未生成该face_token，需要开启redis服务存储图片特征值。

（1）修改 /home/idl-face/odp/conf/app/face-api/project/face.conf 将face_token_storage：db 改为 face_token_storage：cache

```
/home/idl-face/odp/conf/app/face-api/project
[root@823f05ee621e project]# cat face.conf
# 给appid默认值(仅私有化) 默认值default
default_appid : default

# 用户下人脸数目限制 0代表无限制
user_face_limit : 0

# 人脸图片尺寸大小限制(V3接口) 0代表无限制
# 16777216=4096*4096 2073600=1920*1080
image_size_limit : 0

# identify/midentify接口中的组数量大小限制数 0代表无限制
identify_group_limit : 10

# face_token存储位置 none代表不存储 db代表存储于数据库中 cache代表存储于缓存中
face_token_storage : cache

# face_token存储方式  0异步存储 1同步存储(接口返回耗时会增长)
face_token_sync : 0
```

（2）修改 /home/idl-face/odp/conf/app/face-api/project/service.conf 将cache：0 改为 cache：1

```
/home/idl-face/odp/conf/app/face-api/project
[root@823f05ee621e project]# cat service.conf
# 运行环境  私有化使用private
env : private

# 缓存开关(1代表开 0代表关)
cache : 1

# 人脸库图片存储开关(1代表开 0代表关)
storage : 0

# 数据同步配置 1:实时同步到AISE 0:异步方式同步
data_sync : 0
```

（3）进入 /home/idl-face/odp/conf/db/ 目录，新建redis.conf文件，修改文件权限 chmod 755 redis.conf

```
[root@823f05ee621e db]# ll
total 16
-rwxr-xr-x 1 idl-face idl-face 1259 Oct 22 16:33 cluster.conf
-rwxr-xr-x 1 idl-face idl-face 1583 Oct 22 16:33 cluster.conf.ctemplate
-rwxr-xr-x 1 idl-face idl-face   28 Dec  7  2017 global.conf
-rwxr-xr-x 1 idl-face idl-face   86 Oct 22 16:44 redis.conf
[root@823f05ee621e db]# pwd
/home/idl-face/odp/conf/db
```

（4）文件中添加以下内容，将红框中的ip改为redis服务的地址

> 注：单条cache有效时间为1个小时，每条占用空间约4~5K内存
>
> 若想修改Cache有效时间,修改/home/idl-face/odp/app/face-api/models/service/dao/cache/Face.php中 EXPIRE项的值即可

其他问题

1.修改max_face_num检测最大人脸数量参数

（1）进入容器

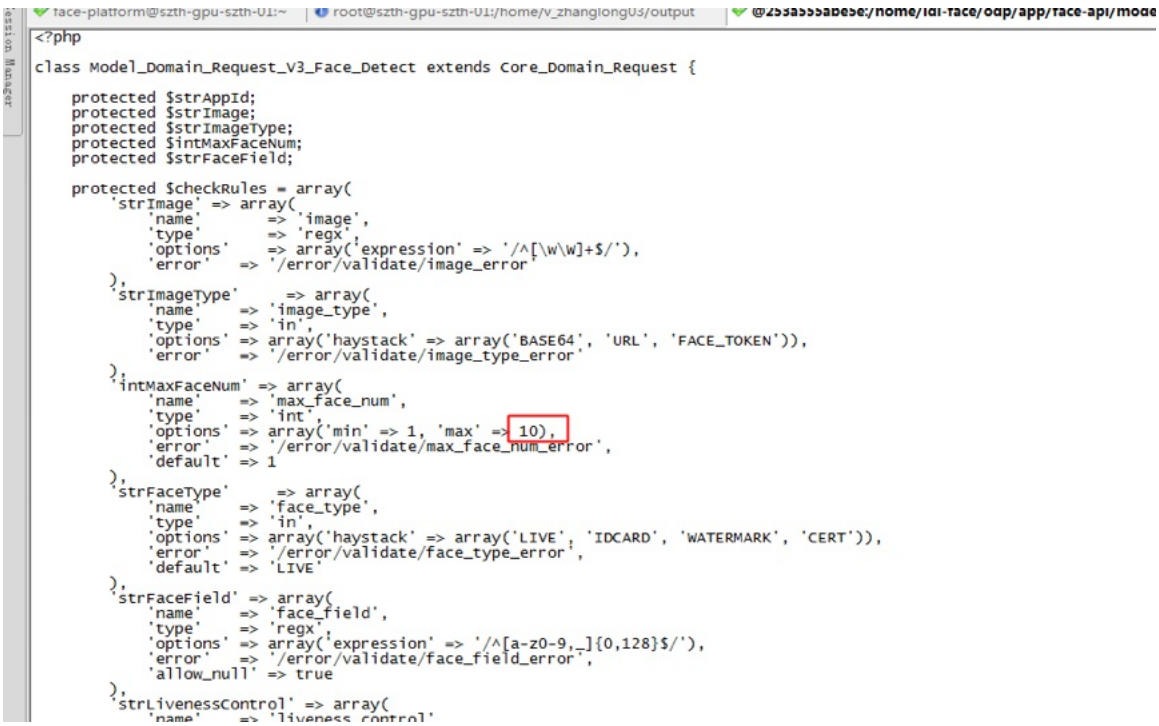
```
 docker ps -a #查看所有容器
docker exec -it (container_id) /bin/bash #进入到container_id容器
```

（2）修改文件内容

进入文件

```
cd /home/idl-face/odp/app/face-api/models/domain/request/v3/face/Detect.php
```

修改最大人脸数量阈值



```
<?php
class Model_Domain_Request_V3_Face_Detect extends Core_Domain_Request {

    protected $strAppId;
    protected $strImage;
    protected $strImageType;
    protected $intMaxFaceNum;
    protected $strFaceField;

    protected $checkRules = array(
        'strImage' => array(
            'name'       => 'image',
            'type'       => 'regx',
            'options'    => array('expression' => '/^[\w\W]+$/'),
            'error'      => '/error/validate/image_error'
        ),
        'strImageType'     => array(
            'name'    => 'image_type',
            'type'    => 'in',
            'options' => array('haystack' => array('BASE64', 'URL', 'FACE_TOKEN')),
            'error'   => '/error/validate/image_type_error'
        ),
        'intMaxFaceNum' => array(
            'name'    => 'max_face_num',
            'type'    => 'int',
            'options' => array('min' => 1, 'max' => 10),
            'error'   => '/error/validate/max_face_num_error',
            'default' => 1
        ),
        'strFaceType'     => array(
            'name'    => 'face_type',
            'type'    => 'in',
            'options' => array('haystack' => array('LIVE', 'IDCARD', 'WATERMARK', 'CERT')),
            'error'   => '/error/validate/face_type_error',
            'default' => 'LIVE'
        ),
        'strFaceField' => array(
            'name'    => 'face_field',
            'type'    => 'regx',
            'options' => array('expression' => '/^[a-z0-9,_]{0,128}$/'),
            'error'   => '/error/validate/face_field_error',
            'allow_null' => true
        ),
        'strLivenessControl' => array(
            'name'    => 'liveness_control'
```

（3）修改人脸检测接口中max_face_num的最大人脸数量

2.当访问接口时，会给出不同的错误提示

（1）当返回504时，一般是nginx错误，可以查看nginx服务是否启动，查看nginx的日志

（2）当返回502时，一般是hhvm错误，可以查看hhvm服务是否启动，查看hhvm的日志，查找问题原因

（3）当返回222915时，一般是feature_service错误，可以查看feature_service服务是否启动，查看feature_service的日志，查找问题原因

3.当访问接口时错误大概定位

```
如果提示504 则查看nginx是否有错误
如果提示502 500，则查看hhvm是否有错误，
如果提示222915，则查看feature_service是否有错误
```

4.如何控制1：N返回的最大人脸数

```
```javascript
进入docker容器
cd /home/idl-face/odp/app/face-api/models/domain/request/v3/face
vi Identify.php
第58行'options' => array('min' => 1, 'max' => 20),将max调大
```

注意：返回较多时，会影响检索速度

5. **若您想将数据库从一台服务器迁移到另一台服务器，请参考**数据库迁移方案

其他常见问题

其他问题

1.数据库迁移方案

您可以选择直接拷贝数据库到另一台服务器上，也可以选择将服务器中的内容导出到另一台数据库中

方法一：拷贝数据库

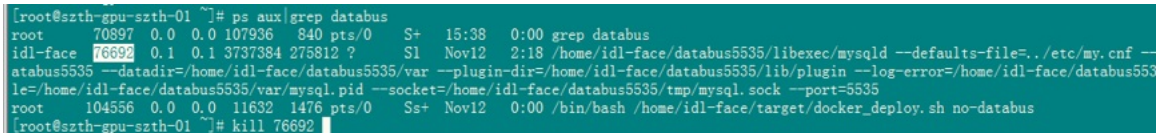
（1）查看服务器一是否有databus进程，如果有请kill掉databus进程

查看容器进程

```
ps aux|grep databus
```

杀掉进程（示例）

```
kill 76692（进程号）
```



（2）进入/home/idl-face/目录 将databus5535目录打包成databus5535.tar.gz，下载到本地

进入目录

```
cd /home/idl-face
```

将databus5535目录打包成databus5535.tar.gz

```
tar -czvf databus5535.tar.gz databus5535/
```

（3）服务器二中新建idl-face用户，查看是否有/home/idl-face/目录，如果没有则新建/home/idl-face/目录 新建idl-face用户
```

```
useradd idl-fac
```

新建/home/idl-face/目录

```
mkdir -p /home/idl-face/
```

（4）4、 将databus5535.tar.gz上传到服务器中/home/idl-face/目录，解压 tar –xzvf databus5535.tar.gz （5）5、 切换idl-face用户，进入/home/idl-face/databus5535/bin/目录，启动数据库 进入目录

```
cd /home/idl-face/databus5535/bin/
```

启动数据库服务

```
nohup ./mysqld_safe --defaults-file=../etc/my.cnf &
```

（6）切换root用户，将数据库加入自启动 打开文件

```
vim /etc/rc.local
```

加入数据库自启动

```
bash /home/idl-face/databus5535/sh/mysql_boot.sh
```

方法二：数据库导入 （1）在新部署的服务器上面使用自动化脚本部署数据库服务需要提前安装数据库服务 参考部署说明文档进行数据库服务的一键安装 （2）在原数据库服务器进入/home/idl-face/databus5535/bin/目录，执行导出数据库命令 进入目录

```
cd /home/idl-face/databus5535/bin/
```

导出数据库命令

```
./mysqldump -u root -p face > face.sql
```

输入密码Bs~XIsDDv4XcDGCt)S(+4*yjQ&8NJh

```
[root@szth-gpu-szth-01 bin]# ./mysqldump  -u root -p face >face.sql
Enter password:
[root@szth-gpu-szth-01 bin]# ll
total 110404
-rw-r--r-- 1 root     root     1217387 Nov 13 15:53 face.sql
```

（3）将face.sql下载上传到服务器二的/home/idl-face/目录 （4）进入服务器二/home/idl-face/databus5535/bin目录，进入命令行模式，删除face库 进入目录

```
cd /home/idl-face/databus5535/bin/
```

执行进入数据库命令

```
./mysql -u root -p
```

输入密码Bs~XIsDDv4XcDGCt)S(+4*yjQ&8NJh 删除数据库

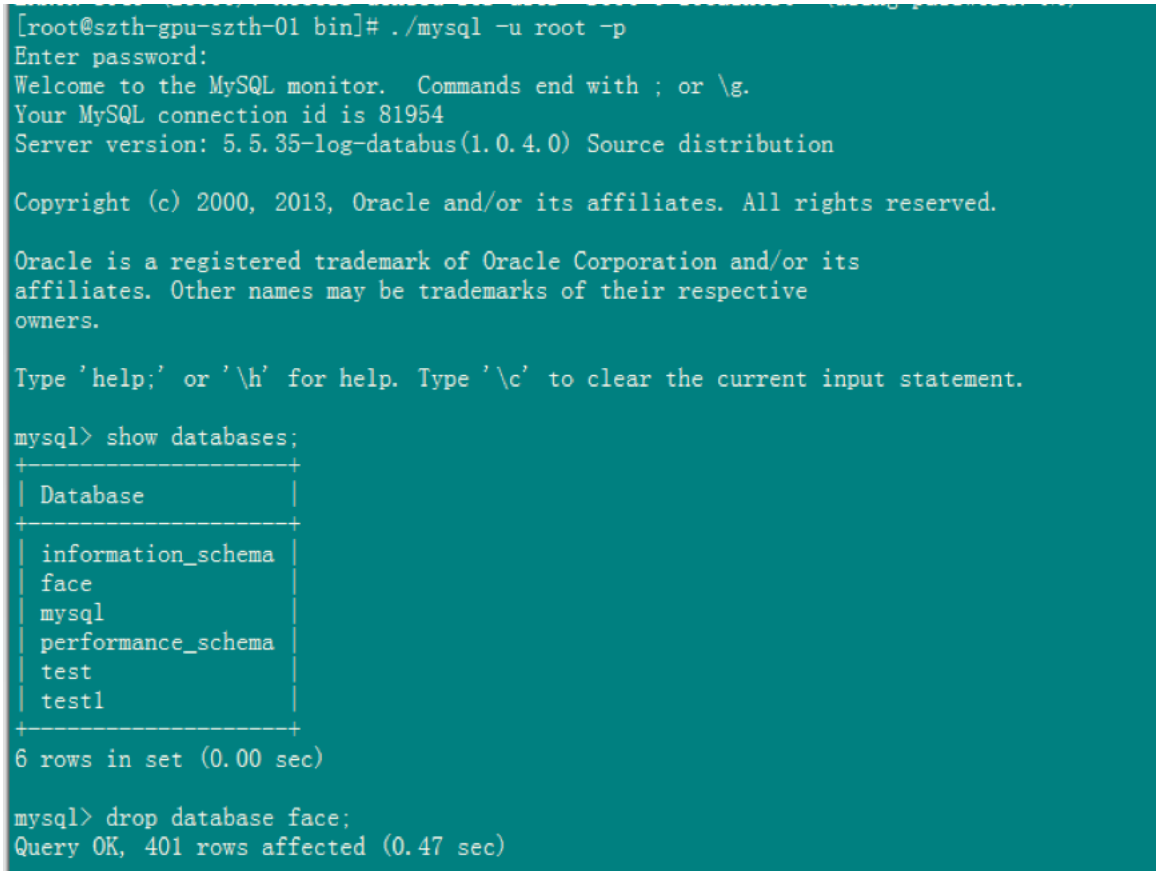
```
drop database face;
```

新建数据库

```
create database face;
```

导入数据库

```
source /home/idl-face/face.sql
```



（5）新建数据库，并执行导入操作

2.如何重新启动人脸识别服务？

第一步，使用以下命令查看容器ID

```
docker ps -a
```

回显的CONTAINER ID就是容器ID。

第二步，使用以下命令重启服务

```
docker restart <容器ID>
```

购买指南

价格说明

人脸私有化部署包针对服务器硬件进行授权，约定授权的并发数量，完成购买后，即可下载安装部署包，授权时间永久有效。

您可以在百度云控制台发起测试申请，完成企业认证后，可以发起免费测试申请，有效期为1个月。

> 授权数量和您的显卡数量有关，您需要多少GPU显卡，则购买/申请多少授权
>
> 如果您想申请测试版/正式版部署包可以在这里发起申请

产品定价

人脸识别私有化部署包提供人脸检测与属性分析、人脸比对、人脸搜索、活体检测等基础功能，支持百万级超大型人脸库，可实现毫秒级响应。

在产品性能方面，不同显卡有不同的性能表现，您可根据自己的业务需求及显卡类型灵活选择。

支持显卡类型	价格
P4/GTX：P4、P40、1060、1070、1070Ti、1080、1080Ti显卡 TESLA：T4、V100显卡 RTX：2060、2070、2080、2080Ti、3060、3070、3080显卡	联系商务

> 注：
>
> 1. 以上性能指标为图片长宽在600像素x600像素情况下的数据
> 2. 如果您需要CPU版本及其他GPU显卡类型请提交工单咨询

硬件配置及推荐

本文主要介绍了人脸检测私有化部署包部署所需的硬件配置，您可以按照如下文档准备硬件或检验已有硬件是否符合要求。

推荐使用**NVIDIA Tesla T4、2080Ti**，百度针对**NVIDIA Tesla T4、2080Ti**进行针对性优化，提高了GPU的利用率。

显卡选型建议

- **NVDIA Tesla P4/GTX系列**：P4、P40、1060、1070、1070Ti、1080、1080Ti 推荐使用**P4**卡（单张P4卡最多支持50QPS）

  > 注：P40显存大但运行模型的计算单元并不多，因此运行人脸识别模型的性能与P4基本没差别。

- **T4显卡、V100显卡**：T4、V100（单张T4卡最多支持50QPS）
- **NVDIA RTX系列**：2070、2080、2080Ti、3060、3070、3080、3090 推荐使用**2080Ti**卡（单张2080TI卡最多支持50QPS）

  显存需要6G以上

  > 注：1050及之前的显卡因为显存不足6GB，所以不支持人脸私有化模型部署。另外，百度的人脸识别私有化部署包对于GTX系列显卡也是兼容的，且GTX单卡支持的QPS数量比P4卡多，但因为GTX并非服务器专用显卡，所以请大家基于自身业务需求进行选择。

GPU服务器搭配推荐

≤100W以内的人脸库配置

本地服务器配置

> 百度推荐GPU服务器可参考这里

关键信息	要求	推荐值	推荐型号	备注
CPU	≥1核	单核	INTEL Xeon E5-2650 V4 12C 2.2GHZ*2	intel i7/i9等市场主流CPU均可，推荐使用志强系列 100W以内人脸库单核即可
内存	≥16GB	32（G）	DDR4-2666 32G*8	人脸查找阶段人脸库会进入内存，单条记录4KB所需内存：人脸库数量*4KB
硬盘	＞100GB	500（G）	SAS 10K*1	最小不得低于100GB，推荐使用500GB及以上
GPU		P4/T4/2080Ti	T4	单张T4卡1:N检索上限为50QPS

BCC云服务器配置

> 购买BCC云服务器请点击这里，购买云服务器和人脸部署包授权后（搭配购买BCC云服务器最低可享6折优惠）

关键信息	选择内容	备注
当前区域选择	苏州/保定	目前私有化部署服务器适配苏州/保定地区
架构	异构计算GPU／FPGA／百度昆仑	-
GPU计算型	实例规格：bcc.lgn1.c6m24.1p4（第10个）	显卡数量根据需求选择即可
操作系统	CentOS 7.6X86_64(64bit)	CentOS 7以上均可
系统盘	500GB	-
带宽需求	根据实际需求进行计算	-

100W以上人脸库配置

> 百度推荐GPU服务器可参考这里

关键信息	推荐型号	推荐值	备注
CPU	INTEL Xeon E5-2650 V4 12C 2.2GHZ*2	多核	人脸库总量大小影响比对速度，主要消耗CPU。 单核1：100万在1s内完成，如果要在1s内完成1：1000万需要10个核，理论横向扩展无限制
内存	DDR4-2666 32G	64（G）	人脸查找阶段人脸库会进入内存，单条记录4KB所需内存：人脸库数量*4KB
硬盘	SAS 10K*1	512（G）	人脸原图存储在硬盘，影响能够存储的人脸数量，依据人脸图片大小确定所需存储空间
GPU	Nvidia	P4/T4/2080Ti	单张T4卡1:N检索上限为50QPS

CPU服务器搭配推荐

> 本配置适用于在纯CPU上部署人脸私有化模型的场景，推荐20QPS以下需求客户考虑，20QPS及以上需求用户推荐GPU版本。

100W内人脸库

关键信息	≤15QPS	≤20QPS
型号	INTEL Xeon E5-2650 V4 12C 2.2GHZ*2	INTEL Xeon E5-2650 V4 12C 2.2GHZ*2
核数	≥4核	≥8核
内存	≥30GB	≥64GB
硬盘	＞100GB	＞100GB

硬件对服务的影响

名称	影响	计算方式
CPU	人脸库总量大小影响比对速度，主要消耗CPU	所需CPU数量=所需CPU内核数量÷每个CPU的核数=（人脸库大小÷100万）÷每个CPU的核数
内存	人脸库 需要消耗内存，单条记录4KB	所需内存大小=人脸服务所需内存+其他服务所需内存=人脸库大小x4（单位是：kb）+其他服务所需内存
硬盘	影响能够存储的人脸数量	所需存储空间大小=人脸服务所需存储+其他服务所需存储=人脸图片数量x单张人脸图片大小+其他服务所需存储
GPU	影响能够处理的并发请求的数量	所需T4显卡的数量=业务QPS量级÷50（显卡以Nvdia Tesla T4为例）

> 注：内存建议比计算值大一半以上，保证服务的稳定运行

部署运维

部署说明

部署环境准备（必看）

本文档介绍了鉴权服务及人脸应用服务部署的硬件及环境要求，请您在部署前**务必**参考此文档进行硬件及软件环境检查，以避免在安装部署过程中出现问题。

硬件环境要求

若您仍然不确定硬件选型，请提交工单联系百度的工作人员

名称	推荐	说明
CPU	E5-2620V4*2	支持avx/avx2指令集以及bmi2指令集
CPU内存	≥32GB	内存容量会影响可以创建的人脸库的大小，人脸查找时需要将人脸放到内存中（一条人脸占用4k内存）
主板主频	＞1833mHZ	推荐配置，不做强制要求
硬盘	SAS 10K*1 ≥512GB	推荐转速≥10K，硬盘转速会影响入库写入速度 （根据实际存储需求可调），**人脸服务默认安装到/home下**
GPU	NVIDIA Tesla P4、T4及RTX 2080Ti显卡	显存 ≥6G，不支持虚拟化的GPU

软件环境要求

您可以根据机器显卡版本自行安装显卡驱动

> 若您机器上已存在显卡驱动，建议您对显卡驱动版本进行查看，以减少因显卡驱动版本而导致的不兼容问题。
> 1.418.39以后（包括418.39）
> 2.v100、2080版本驱动418.74以后包括（418.74）
> 可以使用430.40版本显卡驱动

若您仍然不确定软件安装环境，请提交工单联系百度的工作人员

名称	说明
数据库	人脸私有化部署包内已自带Mysql(可参考：如何部署mysql，内置mysql集成Databus，Databus通过挖掘数据库日志的方式，将数据库变更实时、可靠的从数据库拉取出来，通过人脸容器内客户端获取数据变更同步至后方业务系统),**不支持对接外部mysql数据库。**
操作系统	centos7、ubuntu14 /16 /18、 redhat7.2、 suse12
Linux内核	>=3.10
linux桌面环境	确保系统禁用linux桌面环境（包括但不限于"lightdm", "gdm", "kdm" 等）
SELinux	确保系统禁用SELinux
CPU指令集	支持avx/avx2指令集以及bmi2指令集（必要条件）
Python	==2.7/3.6
curl	确保机器存在curl命令

安装部署前确保以下端口号开放（可通过设置防火墙白名单的方式）

> 端口号被占用会导致安装部署失败，请务必保证端口号处于可访问状态

名称	端口号
鉴权服务	8443
人脸应用服务	8300
数据库服务	5535

> 外部调用需要访问8300端口，若您有此方面业务需求，请确保8300端口处于可访问状态

各软件环境查看命令：

```
1. nvidia驱动查看: nvidia-smi
2. docker版本查看: docker version
3. docker信息查看: docker info
4. nvidia-docker2版本查看: nvidia-docker version
```

单机部署

百度为人脸服务的安装提供了一键部署工具，预置显卡驱动，最快半小时部署完成。

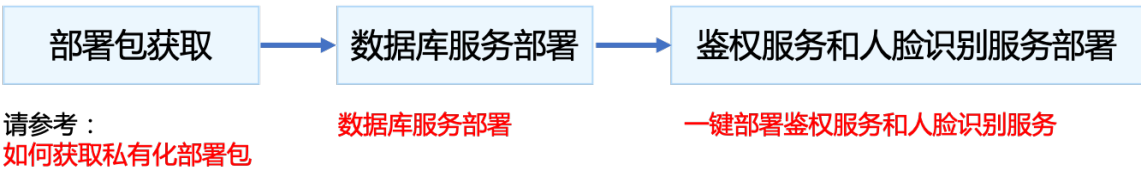
百度人脸服务预置负载均衡功能，您只需根据您的业务需求重复进行部署即可，多台机器部署可以调用同一个数据库。

- 若您需要集群部署服务，请提交工单联系百度工作人员。
- 若您需要修改License文件，请参考License更新说明文档，适用于以下几种情况：

  ① 鉴权物理机硬件指纹发生变化，需要重新申请部署包更换License

  ② 测试版转为正式版，需要更换正式版License文件

总体流程



部署环境准备

请参考部署环境准备文档，请您在部署前**务必**参考此文档进行硬件及软件环境检查，以避免在安装部署过程中出现问题。

Step1：获取部署包

1、获取部署包安装文件下载链接，下载部署包。

下载完成的文件示例如下：609859F08F4B4FB782948D669EE3CFE3.tar.gz；

2、将609859F08F4B4FB782948D669EE3CFE3.tar.gz上传到待部署的服务器中。

3、执行以下命令解压部署包

```
tar zxvf 609859F08F4B4FB782948D669EE3CFE3.tar.gz
```

4、解压后进入original目录执行**bash download.sh**命令获取全部安装文件，执行脚本后会自动下载以下安装文件：数据库服务安装包、鉴权服务安装包、应用服务安装包以及docker安装包等基础依赖环境。

```
bash download.sh
```

若您在此过程出现问题，请提交工单联系百度的工作人员

Step2：部署数据库服务

5、进入以下文件路径进行数据库服务的安装

```
original/package/Applications/face-server/
输入解压命令  tar -xf project-conf.tar
进入original/package/Applications/face-server/project-conf/
```

6、部署数据库服务（包含数据库服务开机自启功能）

```
bash mysql_start.sh
------------------------------出现下面结果表示安装mysql成功------------------------------
1. databus start
2. 190506 15:08:17 mysqld_safe Logging to '/home/idl-face/databus5535/log/mysql.err'.
3. 190506 15:08:17 mysqld_safe Starting mysqld daemon with databases from /home/idl-
4. face/databus5535/var
5. databus init start
6. databus grant ...
7. databus init start
8. databus init finish
```

7、检查数据库是否启动成功

```
ps -ef | grep mysql
```

> 注：如果一键部署数据库服务失败，则参考常见问题文档

8、开放数据库端口，并将命令加入rc.local

```
执行iptables -I INPUT -p tcp --dport 5535 -j ACCEPT
并将该命令加入/etc/rc.local
```

Step3：一键安装鉴权和人脸识别服务

9、数据库安装成功后，开始进行鉴权服务和人脸识别服务的一键部署。首先进入以下文件路径；

```
cd original/package/Install
```

10、执行一键部署命令，一键安装鉴权服务和人脸识别服务;

```
python install.py inall
```

> 注：一键安装过程中出现的问题请参考常见问题文档，其中包含：
> （1）检查服务器环境过程中遇到的问题
> （2）安装Docker和NvidiaDocker遇到的问题
> （3）安装人脸服务遇到的问题
> （4）其他过程中遇到的问题

11、安装过程中会自动进行环境检查，如果安装进程停住并提示"Environment checking failed! Please fix them before installation."表明环境检查失败,请先排查失败的环境检查项，再重新执行安装，或者输入continue强制继续安装；

12、安装过程中会提示输入**鉴权集群ip地址（鉴权服务部署时）、数据库ip地址、鉴权ip地址（人脸服务安装时）**，三个ip地址均输入本机实际网络地址即可。

> IP填写需要填写本机实际IP，如果机器没有做静态ip且为单机部署的情况下，鉴权集群ip地址可以输入127.0.0.1，数据库ip地址以及鉴权ip地址可以输入docker0的ip地址默认为172.17.0.1

13、安装过程中提示输入gpu_id，按照提示的gpu序号填写需要安装人脸应用服务的显卡序号即可。

> 注：
> ① 在只有一张显卡的情况下，gpu_id默认为0，填写0即可；
> ② 当存在多张显卡时，序号从0开始增加排序，按照需求选择对应的显卡填写即可。

若您在一键安装过程中出现问题，请提交工单联系百度的工作人员

▶ 如何单独部署某个模块

Step4：验证服务部署成功 基础应用健康检查 应用健康检查（或故障排查）脚本：trouble_shooting.tar

脚本能力：鉴权服务健康检测、容器状态检查、端口探活、网络联通性测试、容器关键报错日志输出等

使用方法: 将脚本上传至服务器任意目录（或在服务器直接下载），并解压后运行。

```
###### 解压
tar vxf trouble_shooting.tar
###### 执行
bash trouble_shooting.sh
```

人脸接口可用性验证 进入face-service容器：

```
docker ps -a #查看所有容器
docker exec -it ${face-service容器名或容器ID} /bin/bash #进入到container_id容器
```

切换目录

```
cd testtool
```

执行测试脚本：

```
/home/idl-face/odp/php/bin/php FaceApiV3Test.php 127.0.0.1 8300
```

如接口返回错误信息，请根据返回的错误码前往 错误码页面排查。

如所有接口返回正常，代表部署成功。

注：

如人脸注册接口返回 “eorror_code:223105, error_msg: face is already exist”,该报错是由于FaceApiV3Test.php测试脚本重复执行，导致人脸重复注册引起，可忽略。

如有其他问题请到运维文档自助进行排查或提交工单联系百度的工作人员

多机部署

百度为人脸服务的安装提供了一键部署工具，预置显卡驱动，最快半小时部署完成。

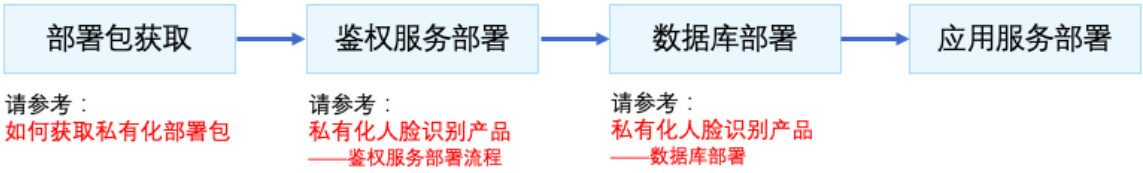
百度人脸服务预置负载均衡功能，您只需根据您的业务需求重复进行部署即可，多台机器部署可以调用同一个数据库。

- 若您需要单机部署服务，请参考单机部署文档。
- 若您需要集群部署服务，请提交工单联系百度工作人员。
- 若您需要修改License文件，请参考License更新说明文档，适用于以下几种情况：

  ① 鉴权物理机硬件指纹发生变化，需要重新申请部署包更换License

  ② 测试版转为正式版，需要更换正式版License文件
- 如果您在此部署过程中有任何问题，可以通过提交工单的方式咨询解决。

总体流程



部署环境准备

请参考部署环境准备文档，请您在部署前**务必**参考此文档进行硬件及软件环境检查，以避免在安装部署过程中出现问题。

Step1：部署鉴权服务

- 鉴权服务用于给人脸识别服务认证授权，如果不运行鉴权服务，后续的人脸服务也将无法启动。
- 鉴权服务要求必须在Linux物理机上运行，鉴权服务根据提取的指纹文件生成，指纹文件和机器硬件采用一机一码绑定。因此运行鉴权服务的机器硬件设备不允许改变，如有改变将导致服务终止，需要重新申请授权。
- 获取指纹请使用root用户执行。

鉴权服务安装步骤

1、获取部署包安装文件下载链接，下载部署包。

下载完成的文件示例如下：609859F08F4B4FB782948D669EE3CFE3.tar.gz；

2、执行以下命令解压部署包

```
tar zxvf 609859F08F4B4FB782948D669EE3CFE3.tar.gz
```

3、解压后进入original目录执行**bash download.sh**命令获取全部安装文件，执行脚本后会自动下载以下安装文件：鉴权服务安装包、人脸服务安装包以及docker安装包等基础依赖环境。

```
bash download.sh
```

4、将带有全部安装文件的original文件夹上传到待部署的服务器中。

5、进入以下文件路径；

```
cd original/package/Install/
```

6、执行以下命令完成鉴权服务的安装。

```
python install.py in c-offline-security-server
```

7、安装过程中会自动进行环境检查，如果安装进程停住并提示"Environment checking failed! Please fix them before installation."表明环境检查失败,请先排查失败的环境检查项，再重新执行安装，或者输入continue强制继续安装；

8、服务启动会监听8443端口，通过netstat -apn | grep 8443可以看端口是否已经被监听。

9、服务日志在解压后目录的/home/baidu/work/c-offline-security-server/log下，提供该目录下的所有日志文件，可以通过日志定位问题。

10、停止鉴权服务和启动鉴权服务的方法如下：

停止鉴权服务：

```
cd /home/baidu/work/c-offline-security-server/ && bash start/c-offline-security-server-stop.sh
```

启动鉴权服务：

```
cd /home/baidu/work/c-offline-security-server/ && bash start/c-offline-security-server-start.sh
```

Step2：部署数据库服务

数据库服务是人脸识别服务正常使用的必要条件

无论是单实例部署场景，多实例部署场景还是k8s集群部署场景，数据库服务均需要单独进行部署。

获取data bus部署包

```
wget -O basepkg.tar.gz http://face-package.bj.bcebos.com/basepkg/basepkg-0720.tar.gz
```

部署流程

1、以root用户创建idl-face新用户

```
useradd idl-face
passwd  用户自定义
```

2、将获取到的安装包basepkg.tar.gz解压，里面有databus5535目录，将其移到: /home/idl-face/databus5535/ （目录固定，不能为其他目录）

```
mv ./databus5535 /home/idl-face/
```

3、创建一个给mysql用的日志目录 /var/log/mariadb，并赋为 777 权限，并对 /home/idl-face/databus5535/ 赋权

```
mkdir -p /var/log/mariadb
chmod 777 /var/log/mariadb -R
chown -R idl-face.idl-face  /home/idl-face/databus5535/
```

4、切换到idl-face用户

```
su idl-face
```

5、进入databus5535目录

```
cd /home/idl-face/databus5535/bin/
```

6、启动服务

```
nohup ./mysqld_safe --defaults-file=../etc/my.cnf &
```

7、初始化数据库

```
 cd /home/idl-face/databus5535/
bin/mysql -uroot -p'Bs~XIsDDv4XcDGCt)S(+4*yjQ&8NJh' --default-character-set=utf8 < sql/database_and_grant.sql
bin/mysql -uroot -p'Bs~XIsDDv4XcDGCt)S(+4*yjQ&8NJh' --default-character-set=utf8 < sql/face.sql
```

8、检查数据库是否启动成功

```
ps -ef | grep mysql
```

Step3：部署应用服务

3.1 应用单实例部署说明

1、获取部署包安装文件下载链接，下载部署包。

下载完成的文件示例如下：609859F08F4B4FB782948D669EE3CFE3.tar.gz；

2、执行以下命令解压部署包

```
tar zxvf 609859F08F4B4FB782948D669EE3CFE3.tar.gz
```

3、解压后进入original目录执行**bash download.sh**命令获取全部安装文件，执行脚本后会自动下载以下安装文件：鉴权服务安装包、人脸服务安装包以及docker安装包等基础依赖环境。

```
bash download.sh
```

4、将带有全部安装文件的original文件夹上传到待部署的服务器中。

5、进入以下文件路径；

```
cd original/package/Install
```

6、查看可用模块，

```
python install.py se
```

7、查看已安装模块；

```
python install.py li
```

8、人脸服务部署；

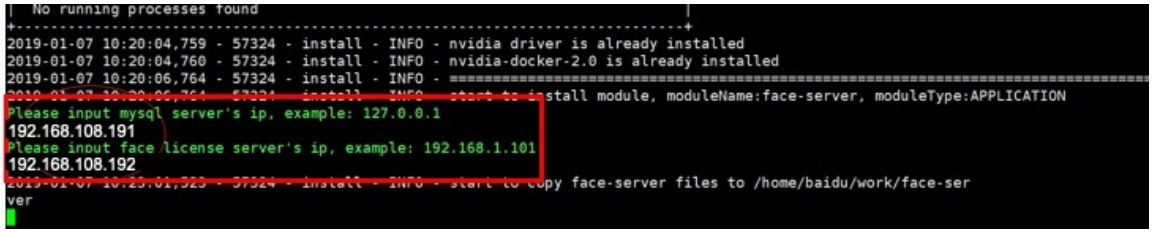
```
python install.py in face-server
```

9、安装过程中会自动进行环境检查，如果安装进程停住并提示"Environment checking failed! Please fix them before installation."表明环境检查失败,请先排查失败的环境检查项，再重新执行安装，或者输入continue强制继续安装；

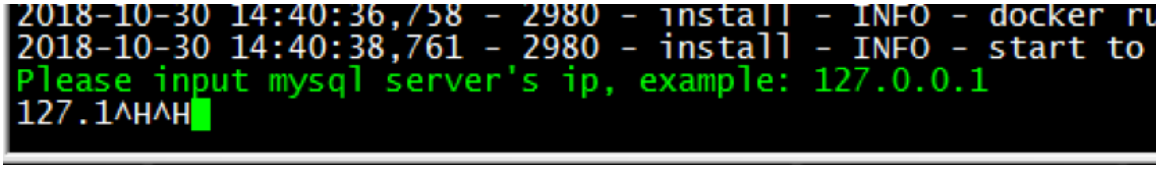
10、环境检查通过后自动继续安装，按照提示输入数据库以及本地ip。

其中mysql server's ip请填写数据库服务所在机器的实际网络地址；

face license server's ip请填写鉴权服务所在机器的实际网络地址。



11、因为操作系统对标准输入的处理无法在输入mysql和license server的ip时使用退格键，退格键被显示为 ^ H



用ctrl+backspace即可进行退格操作

12、提示安装成功后，可检查服务是否启动

```
docker ps ；
```

13、如果服务没有启动，可以使用docker ps -a, 查看容器id；

人脸识别服务启动后，**默认的http端口是8300**

注意事项

1、docker初始后，内部程序初次部署需要2分钟；

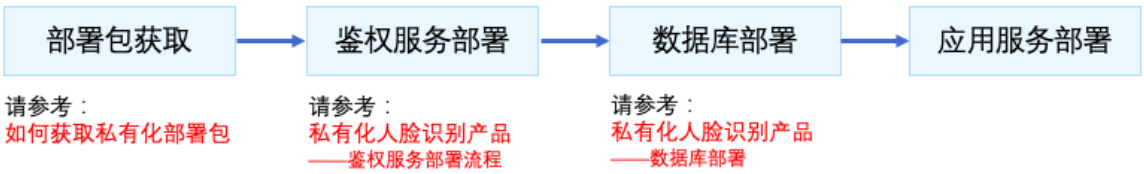
2、docker启动后，禁止用docker attach 命令进入容器，需要使用docker exec -it (dockerid) /bin/bash 命令登入容器

3.2 应用多实例部署说明

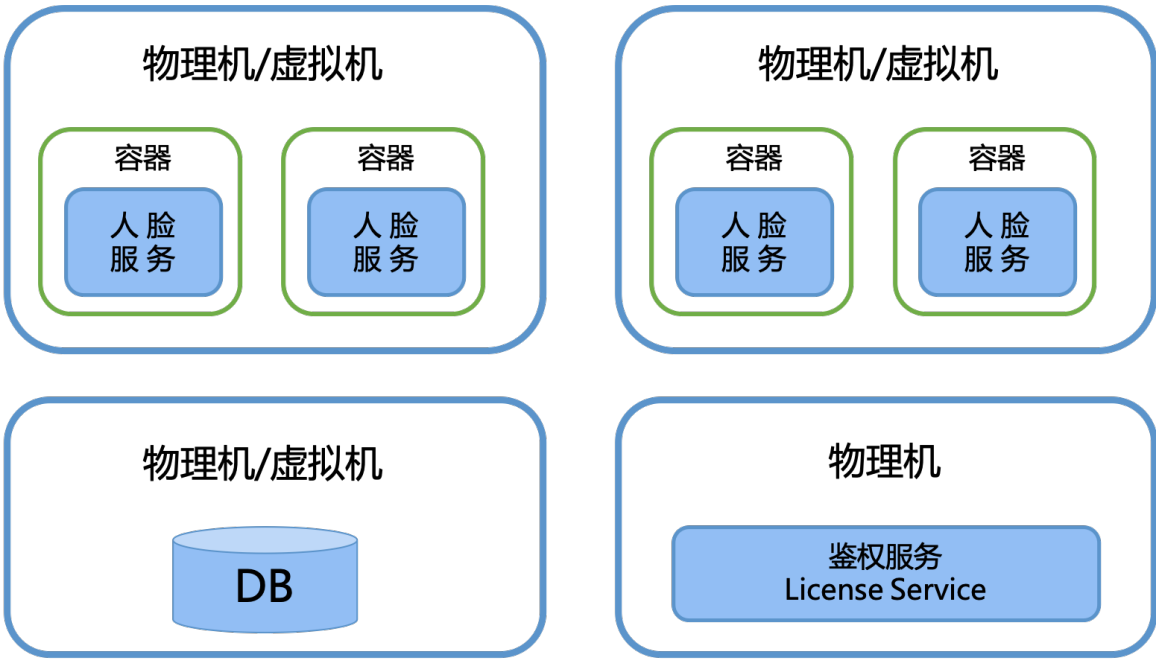
1、该部署说明适用于**单机多实例**部署场景，如**单机四卡四实例场景，每个实例均由单卡提供算力支持**。

2、**多机多实例部署场景即在多台机器上重复单机单实例或多机多实例的操作流程**。

总体流程



结构图



需要部署鉴权服务、databus数据库，多实例一键安装

可用于10实例左右的小规模集群

部署步骤

1、获取部署包安装文件下载链接，下载部署包。

下载完成的文件示例如下：609859F08F4B4FB782948D669EE3CFE3.tar.gz；

2、执行以下命令解压部署包

```
tar zxvf 609859F08F4B4FB782948D669EE3CFE3.tar.gz
```

3、解压后进入original目录执行**bash download.sh**命令获取全部安装文件，执行脚本后会自动下载以下安装文件：鉴权服务安装包、人脸服务安装包以及docker安装包等基础依赖环境。

```
bash download.sh
```

4、将带有全部安装文件的original文件夹上传到待部署的服务器中。

5、进入以下文件路径；

```
cd original/package/Install
```

6、检查配置文件module.json中是否有要安装各模块的名称

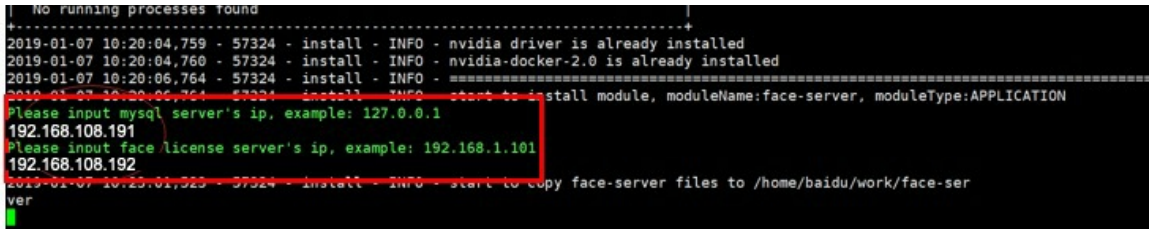
7、安装模块

```
python install.py in face-server
```

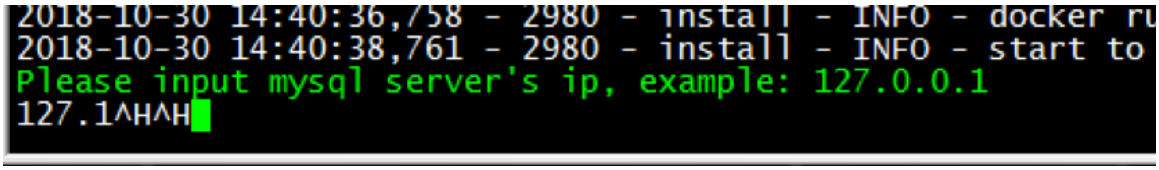
8、接下来输入数据库ip以及授权服务ip；

其中mysql server's ip请填写数据库服务所在机器的实际网络地址；

face license server's ip请填写鉴权服务所在机器的实际网络地址。



9、因为操作系统对标准输入的处理无法在输入mysql和license server的ip时使用退格键，退格键被显示为^H



用ctrl+backspace即可进行退格操作

10、 进入project-conf目录

```
cd /home/baidu/work/face-server/project-conf
```

11、如果要启动多个实例，重复执行：

```
bash multi_docker_start.sh
```

注意：确认无误输入y,GPU id输入GPU序号（如 0）即可

12、查看启动的容器

```
docker ps
```

13、http端口

```
启动后，每个实例都会有一个对应的http端口通过docker ps -a 查看容器可看到端口。
```

License更新说明

适用场景

适用于已经完成私有化部署的客户对License延期、License扩容、License新增产品授权的场景；

1、License延期：延长私有化部署包使用时间。

2、License扩容：增加私有化部署的应用实例数，使私有化部署包可在更多的设备上部署。

3、License新增产品授权：如部署完A产品后还需要在当前环境（同一台机器或同样的局域网环境）进行B产品的部署，A产品和新增的B产品使用同一套鉴权服务，需进行License变更。

如何获取更新后的License

1、License延期：在console端管理控制台**产品服务/XXX-本地部署包管理/查看详情**页面点击延期申请，发起License延期，审核通过后即可获得更新的License证书文件。或线下沟通后获取新的License证书文件。

2、License扩容：请线下商务沟通后获取新的License证书文件。

3、License新增产品授权：请线下商务沟通后获取新的License证书文件。

更新流程

EasyPack一键部署工具支持对私有化部署包的License证书进行一键更新，一键更新时不需要下载任何包及安装任何模块，请参考以下流程进行License证书更新。 1、获取到包含License的部署包后，解压缩，进入以下文件路径；

```
cd original/package/Install
```

2、执行以下命令完成License更新。

```
python install.py lu
```

3、操作成功后如下图所示，提示License update successfully即更新完成。

> 注：若您是2019.11月之前安装的服务，请在License更换完成后重启容器，再进行业务使用。

若您license更新失败，请请提交工单联系百度的工作人员

数据库集群部署

数据库主从方案 本文档介绍了人脸服务数据库的主从同步方案，当主库出现问题时，可以快速切换到从库提供服务，提高性能。 **主数据库**

1. 服务器二中新建idl-face用户，查看是否有/home/idl-face/目录，如果没有则新建/home/idl-face/目录

```
useradd idl-face
mkdir -p /home/idl-face/
```

2. 下载主数据库部署包到服务器中/home/idl-face/目录并解压

```
wget -O databus5535.tar.gz http://face-package.bj.bcebos.com/k8s/databus5535-master.tar.gz
tar –xvf databus5535.tar.gz
```

3. 改变目录用户权限

```
chown –R idl-face.idl-face /home/idl-face/
```

4. 切换idl-face用户，启动数据库

```
su idl-face
cd /home/idl-face/databus5535/
nohup ./bin/mysqld_safe --defaults-file=./etc/my.cnf &
```

5. 登录数据库

```
./bin/mysql -h127.0.0.1 -P5535 -uroot -p
输入密码：Bs~XIsDDv4XcDGCt)S(+4*yjQ&8NJh
```

6. 添加从库IP地址，用户 密码

```
grant REPLICATION SLAVE  on *.* to 'face'@'从xxx.xxx.xxx.xxx' identified by 'face';
```

从数据库

1. 服务器二中新建idl-face用户，查看是否有/home/idl-face/目录，如果没有则新建/home/idl-face/目录 新建idl-face用户

```
useradd idl-face
mkdir -p /home/idl-face/
```

2. 下载从数据库部署包到服务器中/home/idl-face/目录并解压

```
wget -O databus5535.tar.gz http://face-package.bj.bcebos.com/k8s/databus5535-slave.tar.gz
tar –zxvf databus5535.tar.gz
```

3. 修改目录用户权限

```
chown –R idl-face.idl-face /home/idl-face/
```

4. 切换idl-face用户，启动数据库

```
su idl-face
cd /home/idl-face/databus5535/
nohup ./bin/mysqld_safe --defaults-file=./etc/my.cnf &
```

5. 登录数据库

```
./bin/mysql -h127.0.0.1 -P5535 -uroot -p
输入密码：Bs~XIsDDv4XcDGCt)S(+4*yjQ&8NJh
```

6. 添加主库的ip用户密码和binlog文件名和数据位置

```
change master to master_host="主xxx.xxx.xxx.xxx", master_port=5535, master_user='face', master_password='face',
master_log_file='mysql-bin.000006', master_log_pos=0;
```

（mysql_log_file和master_log_pos从主库上执行 show master status中获取）

7. 开启主从同步

```
start slave;
```

8. 查看状态

```
show slave status\G;
```

主从切换 主数据库服务挂了，切到从库，其中人脸服务，鉴权服务还是用主库服务器，需要修改主库配置文件

1. 修改主库人脸服务数据库配置文件的IP地址，改成从库的IP地址

```
vim /home/baidu/work/face-server/project-conf/sconf/service.conf
###### databus
MYSQL_SVC_SERVICE_HOST=xxx.xxx.xxx.xxx
```

将xxx.xxx.xxx.xxx修改为从数据库的IP地址，其余配置项不要修改

2. 删除原来容器并重新构建新的容器

```
cd /home/baidu/work/face-server/project-conf & bash multi_docker_start.sh
```

数据库主主方案

主库1

1. 下载数据库部署包

```
wget -O databus5535.tar.gz http://face-package.bj.bcebos.com/k8s/databus5535-master.tar.gz
```

2. 将部署包移动到/home/idl-face/目录,并解压。

3. 改变目录用户权限

```
chown -R idl-face.idl-face /home/idl-face/databus5535
```

4. 启动数据库服务

```
su idl-face
cd /home/idl-face/databus5535/
nohup ./bin/mysqld_safe --defaults-file=./etc/my.cnf &
```

5. 登录数据库

```
./bin/mysql -h127.0.0.1 -P5535 -uroot -p
输入密码：Bs~XIsDDv4XcDGCt)S(+4*yjQ&8NJh
```

6. 添加从库ip 用户 密码

```
grant REPLICATION SLAVE  on *.* to 'face'@'10.233.42.1' identified by 'face';
10.233.42.1为从库ip
```

主库2

1. 下载部署包

```
wget -O databus5535.tar.gz https://face-package.bj.bcebos.com/k8s/databus5535-slave.tar.gz
```

2. 将部署包移动到/home/idl-face/目录

3. 修改/home/idl-face/databus5535/etc/ mysqld.cnf，将server-id=1改为=2

4. 改变目录用户权限

```
chown -R idl-face.idl-face /home/idl-face/databus5535
```

5. 启动数据库服务

```
 su idl-face
cd /home/idl-face/databus5535/
nohup ./bin/mysqld_safe --defaults-file=./etc/my.cnf &
```

6. 登录数据库

```
 ./bin/mysql -h127.0.0.1 -P5535 -uroot -p
输入密码：Bs~XIsDDv4XcDGCt)S(+4*yjQ&8NJh
```

7. 添加从库ip 用户 密码

```
 grant REPLICATION SLAVE  on *.* to 'face'@'10.233.43.2' identified by 'face';
```

8. 添加主库的ip用户密码和binlog文件名和数据位置

```
 change master to master_host="10.233.42.2", master_port=5535, master_user='face', master_password='face',
master_log_file='mysql-bin.000006', master_log_pos=0;
```

9. start slave;

10. 查看状态

```
 show slave status\G;
```

主库1

1. 添加主库的ip用户密码和binlog文件名和数据位置

```
 change master to master_host="10.233.42.1", master_port=5535, master_user='face', master_password='face',
master_log_file='mysql-bin.000006', master_log_pos=0;
```

2. start slave;

3. 查看状态

```
 show slave status\G;
```

部署问题 数据库启动报错 1.数据库构建过程中 启动数据库报错

```
 ./bin/mysqld_safe: /home/idl-face/databus5535/bin/my_print_defaults: /opt/compiler/gcc-4.8.2/lib64/ld-linux-x86-
64.so.2: bad ELF interpreter: No such file or directory
```

查看是否存在/opt/compiler/gcc-4.8.2/lib64/ld-linux-x86-64.so.2文件 如果不存在 则 01)切换到/opt/compiler下 下载文件并解压

```
 cd /opt/compiler
wget https://face-package.bj.bcebos.com/k8s/gcc-8.2.tar.gz
wget https://face-package.bj.bcebos.com/k8s/gcc-4.8.2.bpkg-r4.tar.gz
tar -zxvf gcc-4.8.2.bpkg-r4.tar.gz
tar -zxvf gcc-8.2.tar.gz
```

02)修改gcc-4.8.2.bpkg-r4.tar.gz解压后文件的名称

```
 cd gcc-4.8.2.bpkg-r4/
mv gcc-4.8.2.bpkg-r4/ ../gcc-4.8.2
```

03)重新启动数据库

数据库开机无法自启 开机启动脚本不能正常执行可修改脚本为

```
 可将原数据库启动命令更换为
/bin/bash /home/idl-face/databus5535/server start
```

数据库主从重启失效 重启服务器后查看主从状态

```
 show slave status\G

Slave_IO_Running:NO
Slave_SQL_Running:NO
```

可修改开机自启脚本/home/idl-face/databus5535/sh/mysql_boot.sh 添加如下内容

```
 /home/idl-face/databus5535/sh/mysql -h127.0.0.1 -P5535 -uroot -p'Bs~XIsDDv4XcDGCt)S(+4*yjQ&8NJh' -e "stop slave;start slave;"
```

运维支持

运维相关的问题可以在这里进行查看。

若您仍然有部署及调用问题，请提交工单联系百度的工作人员

常用Docker命令

1.docker常用命令

```
 docker ps -a #查看所有容器
docker restart (container_id)#重启container_id容器
docker exec -it (container_id) /bin/bash #进入到container_id容器
docker images #查看所有镜像
docker rm -f  容器ID  #删除容器
```

人脸服务运维

1. **如何查看人脸服务是否部署成功？**

如果执行部署命令后，未出现错误提示，需要确认服务是否正常：

- 在宿主机上执行命令：nvidia-smi, 查看GPU进程是否启动
- 或者进入容器，执行测试脚本:

进入容器：

```
 docker ps -a #查看所有容器
docker exec -it (container_id) /bin/bash #进入到container_id容器
```

切换目录

```
cd testtool
```

执行测试脚本：

```
/home/idl-face/odp/php/bin/php FaceApiV3Test.php 127.0.0.1 8300
```

2.怎么重启鉴权服务

停止服务：

```
cd /home/baidu/work/c-offline-security-server/ && bash start/c-offline-security-server-stop.sh
```

启动服务：

```
cd /home/baidu/work/c-offline-security-server/ && nohup bash start/c-offline-security-server-start.sh &
```

鉴权服务运维

1. **检查鉴权服务是否启动**

```
ps -ef|grep auth_server
```

服务正常启动后如下图所示：

```
root@***:~# ps -ef|grep auth_server
root 20609 20604 0 11:33 pts/2 00:00:00 sudo /home/baidu/work/c-offline-security-server/auth_server
root 20611 20609 0 11:33 pts/2 00:00:07 /home/baidu/work/c-offline-security-server/auth_server
root 22829 22616 0 12:08 pts/3 00:00:00 grep --color=auto auth_server
```

2.检查鉴权服务端口是否启动

netstat -apn|grep 8443。服务正常启动后如下图所示：

```
root@zhangwenkang-EX660:~# netstat -apn|grep 8443
tcp        0      0 0.0.0.0:8443            0.0.0.0:*               LISTEN      20611/auth_server
```

3.查看鉴权启动日志

从/home/baidu/work/c-offline-security-server/log/auth_server.log查看鉴权服务启动日志。

1）启动成功的日志如下图所示：

```
NOTICE: 07-04 15:20:15: Auth_Server * 14548 ---------- open log ----------
WARNING: 07-04 15:20:15: Auth_Server * 14548 ---------- open log.wf ----------
TRACE: 07-04 15:20:16: Auth_Server * 14548 baidu/prienv-security/auth-service-offline/http_server.cpp:279] Load config file success
TRACE: 07-04 15:20:16: Auth_Server * 14548 baidu/prienv-security/auth-service-offline/http_server.cpp:325] Aipe config: cluster conf =
192.168.1.100,192.168.1.101,192.168.1.102, ip = 12, inner port = 8091, data path = /home/baidu/work/c-offline-security-server/data, outer port = 6379
```

2）启动时常见的报错提示如下：

如下表示license文件不对，可能的原因是license的版本不对或者人为修改了license。

```
WARNING: 07-04 18:08:12: Auth_Server * 17018 baidu/prienv-security/auth-service-offline/util/io_utils.cpp:15] Open file license failed
NOTICE: 07-04 18:08:34: Auth_Server * 23584 ---------- open log ----------
WARNING: 07-04 18:08:34: Auth_Server * 23584 ---------- open log.wf ----------
WARNING: 07-04 18:08:35: Auth_Server * 23584 baidu/prienv-security/auth-service-offline/http_server.cpp:257] License decrypt error
WARNING: 07-04 18:08:35: Auth_Server * 23584 baidu/prienv-security/auth-service-offline/http_server.cpp:346] Start server, environment is unsaf
```

4.**查看鉴权请求日志**

从/home/baidu/work/c-offline-security-server/log/aipe*info*日期.log查看鉴权日志。根据应用里鉴权失败的时间或者根据产品名查找日志。

1）鉴权成功日志如下：

```
[2019-07-04 11:36:59.312][20611][020672][aipe_log][info] reg create instance,product=ocr,id=1eb7af0d-bece-4c80-965a-3b97555789c2,res=true
```

或者

```
[2019-07-04 17:16:35.166][16936][017535][aipe_log][info] instance_check success,product=classification-demo,id=b16480ca-92bb-4869-97e0-7da197cb2530
```

2）鉴权常见失败日志如下：

鉴权失败日志	**原因**
verify finger is invaliid,product=...	机器指纹校验失败，license和当前指纹对不上 可能原因： （1）申请License时用的指纹不是当前机器的指纹 （2）当前机器硬件发生了变化 （3）采集指纹时的用户和鉴权服务运行时的用户不一致
verify product has not found,product=...	License里没有该产品的授权
verify product lc is expired,product=...	该产品授权已过期
instance_check reg fail,r_list full,product=...,id=...	该产品授权的实例池已满。可能原因： （1）该产品的容器超过了授权的实例数 （2）已停止的容器占用了授权实例池，等1分钟后实例池自动清理过期实例，再鉴权即可 （3）应用每次重启后碰到的这种情况，可能是鉴权客户端版本较老，没有把实例id持久化，需要升级应用
cache client get context error:Connection refused	鉴权服务连接本机缓存失败，可能的原因是： （1）鉴权服务配置的ip和服务器实际ip不一致； （2）8991端口或6666端口被占用，导致缓存服务没起来

5.**重启鉴权服务**

停止服务：

```
cd /home/baidu/work/c-offline-security-server/ && bash start/c-offline-security-server-stop.sh
```

启动服务：

```
cd /home/baidu/work/c-offline-security-server/ && nohup bash start/c-offline-security-server-start.sh &
```

日志查看及修改

1. 如何查看日志

（1）查看所有服务启动的日志

进入容器，查看文件

```
/home/idl-face/sys_start.log
```

（2）查看人脸服务的日志

进入容器，查看文件

```
/home/idl-face/feature-frame/log/service.log.wf
```

2.修改清理日志时长定时任务（默认为15天定期清理）

（1）进入容器

（2）切换idl-face用户

（3）执行命令crontab -e 修改天数

```
[idl-face@2c387c87d268 ~]$ crontab -l
#crontab
MAILTO=""

6 0 * * * find /home/idl-face/aise-service/logs/aise_service.* -type f -mtime +15 -exec rm -f {} \;
6 0 * * * find /home/idl-face/aise-service/aise-agent/logs/aise_agent.log.* -type f -mtime +15 -exec rm -f {} \;
7 0 * * * find /home/idl-face/relay_master/log/relay.log.* -type f -mtime +15 -exec rm -f {} \;
8 0 * * * find /home/idl-face/public_bridge/log/bridge.log.* -type f -mtime +15 -exec rm -f {} \;
9 0 * * * find /home/idl-face/odp/log/hhvm/hhvm-access.log.* -type f -mtime +15 -exec rm -f {} \;
9 0 * * * find /home/idl-face/odp/log/hhvm/hhvm-error.log.* -type f -mtime +15 -exec rm -f {} \;
10 0 * * * find /home/idl-face/odp/log/ral/ral* -type f -mtime +15 -exec rm -f {} \;
15 0 * * * find /home/idl-face/odp/log/face-api/face-api.log.* -type f -mtime +15 -exec rm -f {} \;
20 0 * * * find /home/idl-face/feature-frame/log/service.log.* -type f -mtime +15 -exec rm -f {} \;
*/5 * * * * cd /home/work/odp && ./hhvm/bin/hhvm app/face-api/script/DeleteGroup.php
```

数据库运维

1.数据库服务加入开机自启

- 在部署过程中一键安装数据库加入开启自启功能
- 手动将数据库启动加入开机启动

```
vim /etc/rc.local
加入bash /home/idl-face/databus5535/sh/mysql_boot.sh
```

2.数据库迁移方案

您可以选择直接拷贝数据库到另一台服务器上，也可以选择将服务器中的内容导出到另一台数据库中

方法一：拷贝数据库

（1）查看服务器一是否有databus进程，如果有请kill掉databus进程

查看容器进程

```
ps aux|grep databus
```

杀掉进程（示例）

```
kill 76692（进程号）
```

```
[root@szth-gpu-szth-01 ~]# ps aux|grep databus
root      70897  0.0  0.0 107936   840 pts/0    S+   15:38   0:00 grep databus
idl-face  76692  0.1  0.1 3737384 275812 ?      Sl   Nov12   2:18 /home/idl-face/databus5535/libexec/mysqld --defaults-file=../etc/my.cnf --
atabus5535 --datadir=/home/idl-face/databus5535/var --plugin-dir=/home/idl-face/databus5535/lib/plugin --log-error=/home/idl-face/databus553
le=/home/idl-face/databus5535/var/mysql.pid --socket=/home/idl-face/databus5535/tmp/mysql.sock --port=5535
root     104556  0.0  0.0  11632  1476 pts/0    Ss+  Nov12   0:00 /bin/bash /home/idl-face/target/docker_deploy.sh no-databus
[root@szth-gpu-szth-01 ~]# kill 76692
```

（2）进入/home/idl-face/目录 将databus5535目录打包成databus5535.tar.gz，下载到本地

进入目录

```
cd /home/idl-face
```

将databus5535目录打包成databus5535.tar.gz

```
tar -czvf databus5535.tar.gz databus5535/
```

（3）服务器二中新建idl-face用户，查看是否有/home/idl-face/目录，如果没有则新建/home/idl-face/目录 新建idl-face用户

```
useradd idl-face
```

新建/home/idl-face/目录

```
mkdir -p /home/idl-face/
```

（4）4、 将databus5535.tar.gz上传到服务器中/home/idl-face/目录，解压 tar –xzvf databus5535.tar.gz （5）5、 切换idl-face用户，进入/home/idl-face/databus5535/bin/目录，启动数据库 进入目录

```
cd /home/idl-face/databus5535/bin/
```

启动数据库服务

```
nohup ./mysqld_safe --defaults-file=../etc/my.cnf &
```

（6）切换root用户，将数据库加入自启动 打开文件

```
vim /etc/rc.local
```

加入数据库自启动

```
bash /home/idl-face/databus5535/sh/mysql_boot.sh
```

方法二：数据库导入

（1）在新部署的服务器上面使用自动化脚本部署数据库服务需要提前安装数据库服务 参考部署说明文档进行数据库服务的一键安装

（2）在原数据库服务器进入/home/idl-face/databus5535/bin/目录，执行导出数据库命令 进入目录

```
cd /home/idl-face/databus5535/bin/
```

导出数据库命令

```
./mysqldump -u root -p face > face.sql
```

输入密码Bs~XIsDDv4XcDGCt)S(+4*yjQ&8NJh

```
[root@szth-gpu-szth-01 bin]# ./mysqldump  -u root -p face >face.sql
Enter password:
[root@szth-gpu-szth-01 bin]# ll
total 110404
-rw-r--r-- 1 root     root     1217387 Nov 13 15:53 face.sql
```

（3）将face.sql下载上传到服务器二的/home/idl-face/目录 （4）进入服务器二/home/idl-face/databus5535/bin目录，进入

命令行模式，删除face库 进入目录

```
cd /home/idl-face/databus5535/bin/
```

执行进入数据库命令

```
./mysql -u root -p
```

输入密码Bs~XIsDDv4XcDGCt)S(+4*yjQ&8NJh 删除数据库

```
drop database face;
```

新建数据库

```
create database face;
```

导入数据库

```
source /home/idl-face/face.sql
```

```
[root@szth-gpu-szth-01 bin]# ./mysql -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 81954
Server version: 5.5.35-log-databus(1.0.4.0) Source distribution

Copyright (c) 2000, 2013, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| face               |
| mysql              |
| performance_schema |
| test               |
| test1              |
+--------------------+
6 rows in set (0.00 sec)

mysql> drop database face;
Query OK, 401 rows affected (0.47 sec)
```

（5）新建数据库，并执行导入操作

```
mysql> create database face;
Query OK, 1 row affected (0.00 sec)

mysql> use face;
Database changed
mysql> source /home/idl-face/face.sql
```

3. **数据库的备份与备份删除**

（1）在服务器上/home/idl-face/databus5535/sh/目录下查看mysql_export.sh脚本

（2） 执行crontab –e命令

```
crontab –e
```

在crontab –e命令里面加入 执行数据库备份命令（eg：每周一上午2点执行数据库备份）

```
0 2 * * 1 bash /home/idl-face/databus5535/sh/mysql_export.sh
```

> 使用此脚本前需要先进行修改
> FILE_PATH 此变量为sql备份地址，可进行自定义。如修改位置。备份定时清理地址需同步进行修改
> FILE环境变量需要修改为$FILE*PATH/face*$TIME.sql 修改后为FILE=$FILE*PATH/face*$TIME.sql

> 注：若想对备份时间点进行修改，修改前五位数字即可：
> 前五位代表 分钟 时 天 月 周 ，中间用空格替代

（3）如果您想对备份的数据库进行定时清除，请执行以下操作

执行crontab –e命令，在里面加入执行数据库备份删除命令（eg：每天晚上00：06会将15天前备份的数据库删除）

```
6 0 * * * find /home/idl-face/sql/face_* -type f -mtime +15 -exec rm -f {} \;
```

（4）保存

4. **清除数据库图片特征值（慎用）**

> 注：图片特征值与Face_Token有关，若您后续需要采用face_token进行人脸注册、人脸比对、人脸搜索，不建议您清除

（1）下载脚本mysql_truncate.sh并且放到服务器/home/idl-face/databus5535/sh/目录

```
cd /home/idl-face/databus5535/sh/
```

（2）当需要执行清空特征值时，执行bash /home/idl-face/databus5535/sh/mysql_truncate.sh命令

```
bash /home/idl-face/databus5535/sh/mysql_truncate.sh
```

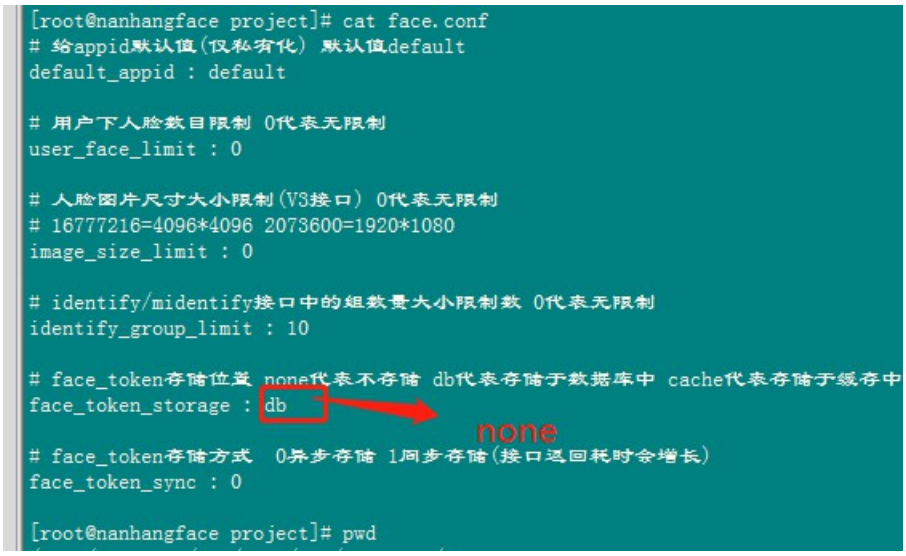
5. **关闭数据库存储图片特征值**

（1）进入目录

```
cd /home/idl-face/odp/conf/app/face-api/project/face.conf
```

（2）将下图db修改为none

> 注：关闭后只能通过图片进行搜索，无法使用face_token,若您后续需要采用face_token进行人脸注册、人脸比对、人脸搜索，不建议您关闭



更改人脸缓存数据同步策略

为保障人脸1:N搜索接口查询效率，人脸容器内置AISE缓存服务,默认情况下采用异步方式更新缓存数据，如果在人脸注册成功后，无法及时通过1:N搜索查询到对应人脸信息，可将数据更新策略更改为同步方式。更改方式如下：

1、切到face-service容器bash终端

```
##### 人脸容器名可通过docker ps|grep face 查询
docker exec -it <人脸容器名> bash
```

2、修改配置文件,将 data_sync值改为1

```
vi /home/idl-face/odp/conf/app/face-api/project/service.conf
##### data_sync
##### 1表示实时同步，0表示异步方式同步
```

3、退出容器bash终端并重启人脸容器

```
exit
docker restart <人脸容器名>
```

IP地址修改

1. **服务器的网络环境发生变化，IP地址如何修改？**

（1）删除容器

（2）修改鉴权的IP地址

鉴权IP地址所在的配置文件

```
vi /home/baidu/work/c-offline-security-server/conf/server.conf
```

修改后需重启鉴权服务

(3) 修改人脸IP地址

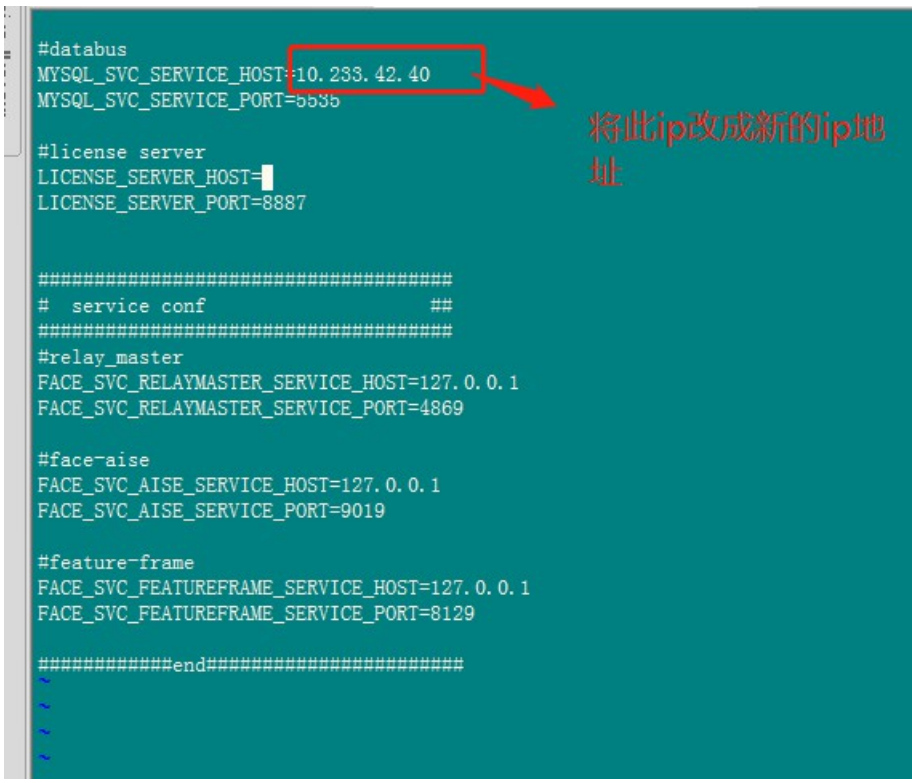
```
vim /home/baidu/work/face-server/project-conf/easypack_init.sh
```

（4）修改连接数据库的IP地址

数据库IP地址所在的文件

```
vim /home/baidu/work/face-server/project-conf/sconf/service.conf
```

修改数据库IP地址



（4）重新生成容器

```
cd /home/baidu/work/face-server/project-conf/

bash  multi_docker_start.sh
```

监控特征抽取服务QPS

> 该监控项用于统计特征抽取服务的并发量和qps等信息。

查询方式 1、通过命令行查询qps

```
 ##### 进入人脸应用容器
docker exec -it ${人脸容器名} bash
##### 调用接口
curl 0.0.0.0:8815/status
```

返回如下：

```
 [root@ad013d40bff0 testtool]# curl 0.0.0.0:8815/status
non_service_error: 0
connection_count: 5
max_concurrency: 15
[baidu.face.GeneralClassifyService]
process (FeatureRequest) returns (FeatureResponse)
count: 23
error: 0
latency: 1186817
latency_50: 1186817
latency_90: 1186817
latency_99: 1186817
latency_999: 1186817
latency_9999: 1186817
max_latency: 1186817
qps: 0
processing: 1
```

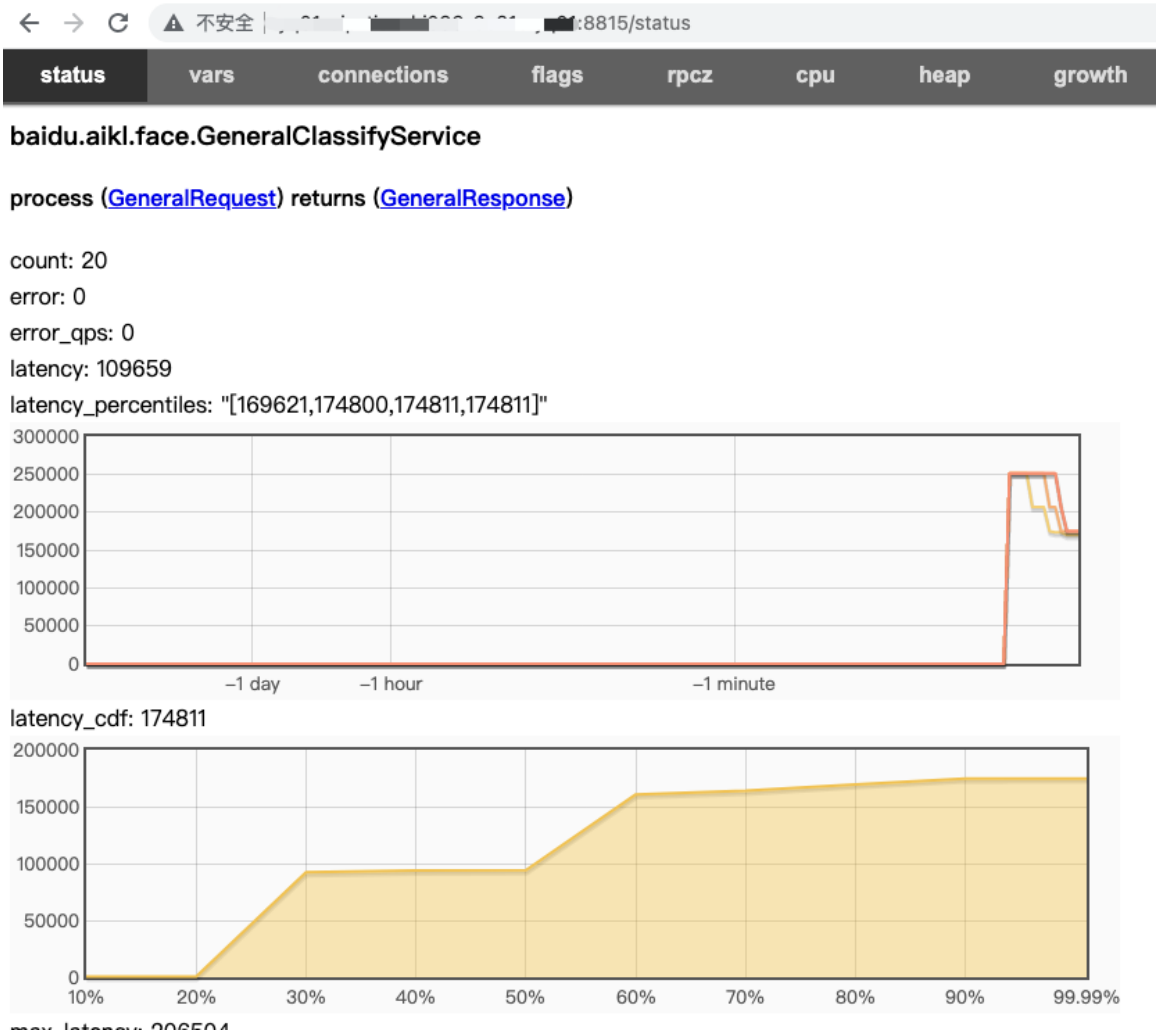
2、通过浏览器可视化监控QPS变化趋势 1）将容器内feature-servide 服务端口号8815在宿主机做映射：

```
 ##### 停用当前face-service 人脸容器
docker stop ${人脸容器名}
docker rm ${人脸容器名}
##### 更新容器启动脚本
cd /home/baidu/work/face-server/project-conf/
##### 需注意下行命令中/附近的空格，避免漏掉
sed -i  's/8300/& -p 8815:8815 /g' docker_init.sh
##### 启动容器
bash multi_docker_start.sh
```

2. 查看人脸容器运行情况

```
 [root@ad013d40bff0 testtool]# bash-4.2$ docker ps -a|grep face
0895e9270bc4   300478125c1e                                                  "/home/idl-face/targ…"   46
seconds ago   Up 43 seconds
0.0.0.0:8815->8815/tcp, 0.0.0.0:8300->8300/tcp,
face-service-0-0
```

3. 通过浏览器访问 `${服务器IP地址}:8815/status`



参数说明

- non_service_error: "non"修饰的是“service_error"，后者即是分列在各个服务下的error，此外的error都计入non_service_error。服务处理 过程中client断开连接导致无法成功写回response就算non_service_error。而服务内部对后端的连接断开属于服务内部逻辑，只要最终服务成功 地返回了response，即使错误也是计入该服务的error，而不是non_service_error。
- connection_count: 向该server发起请求的连接个数，不包含对外连接的个数。
- max_concurrency: 最大并发数
- count: 成功处理的请求总个数。
- error: 失败的请求总个数。
- latency: 在web界面下从右到左分别是过去60秒，60分钟，24小时，30天的平均延时。在文本界面下是10秒内(-bvar_dump_interval控制) 的平均延时。
- latency_percentiles: 是延时的50%, 90%, 99%, 99.9%分位值，统计窗口默认10秒(-bvar_dump_interval控制)，web界面下有曲线。
- latency_cdf: 是分位值的另一种展现形式，类似histogram，只能在web界面下查看。
- max_latency:在web界面下从右到左分别是过去60秒，60分钟，24小时，30天的最大延时。在文本界面下是10秒内(-bvar_dump_interval控 制)的最大延时。
- qps:在web界面下从右到左分别是过去60秒，60分钟，24小时，30天的平均qps。在文本界面下是10秒内(-bvar_dump_interval控制)的平均 qps。
- processing: 正在处理的请求个数。如果持续不为0(特别是在压力归0后)，应考虑程序是否有bug。

License更新步骤

EasyPack一键部署工具支持对私有化部署包的License证书进行一键更新，一键更新时**不需要执行download.sh 或 download.bat,并且不需要安装任何模块，在新申请的部署包中执行更新操作即可**。具体请参考以下流程进行License证书更新。

1、获取到包含License的部署包后，解压缩，进入以下文件路径.

```
cd original/package/Install
```

2、执行以下命令完成License更新。

```
python install.py lu
```

3、操作成功后如下图所示，提示License update successfully即更新完成。

物理机-将机器指纹方式鉴权更换为加密狗硬件鉴权

以下步骤仅适用于物理机部署场景。 加密狗硬件鉴权包获取请您线下联系商务经理。

如何区分离线鉴权服务不同版本？

```
 cd original/package/Install/
python install.py search
##### 或 python install.py se
```

如果c-offline-security-server 版本号 返回 `with-dog` 表示 **加密狗硬件鉴权**

```
模块名: c-offline-security-server， 版本号：with-dog， 内置版本 xxx，依赖模块 []
```

如果c-offline-security-server 版本号 `no-dog` 表示 **x86-机器指纹方式鉴权**

```
模块名: c-offline-security-server， 版本号：no-dog， 内置版本 x, 依赖模块 []
```

替换步骤 1、将部署包解压后进入original目录执行 `bash download.sh`命令获取全部安装文件

```
cd original && bash download.sh
```

2、将旧的机器指纹方式鉴权服务卸载

```
 ##### 进入新的部署包 （加密狗硬件鉴权部署包）
cd package/Install
python install.py remove c-offline-security-server
##### 或 python install.py rm c-offline-security-server
##### 检查/home/baidu/work/c-offline-security-server 是否存在，如存在将其删除
rm -rf /home/baidu/work/c-offline-security-server
```

3、安装加密狗硬件离线鉴权服务

```
python install.py install c-offline-security-server
##### 或 python install.py in c-offline-security-server
```

4、耐心等待几分钟后，执行私有化应用健康检查（或故障排查）脚本：trouble_shooting.tar 来验证 c-offline-sercurity-server 服务

```
##### 解压
tar vxf trouble_shooting.tar
##### 执行
bash trouble_shooting.sh
```

检查加密狗驱动是否运行，正常情况下返回 CodeMeter Server is running.

```
service codemeter status
```

检查加密狗硬件是否被机器识别

```
cmu -x
```

识别成功的结果如下

```
cmu - CodeMeter Universal Support Tool.
Version 6.70 of 2018-Jul-19 (Build 3152) for Linux
Copyright (C) 2007-2018 by WIBU-SYSTEMS AG. All rights reserved.
List all locally connected CmContainers:
- CmContainer with Serial Number 3-4512221 and version 4.10
...
Result: 1 CmContainer(s) listed.
```

识别失败的结果如下：

```
cmu - CodeMeter Universal Support Tool.Version 6.70 of 2018-Jul-19 (Build 3152) for LinuxCopyright (C) 2007-2018 by WIBU-SYSTEMS AG. All rights reserved.List all locally connected CmContainers:Result: 0 CmContainer(s) listed.
```

人脸识别私有化部署模型更新/回滚操作步骤

更新模型操作 以人脸模型从V1升级至V2 举例说明：

1、终止databus数据库相关进程

首先通过如下命令查询databus相关进程信息

```
ps j -A |grep databus5535|grep -v "grep"

PPID   PID  PGID   SID TTY      TPGID STAT   UID   TIME COMMAND
1 21518 21513 21513 ?           -1 S     1008   0:00 /bin/sh /home/idl-face/databus5535/bin/mysqld_safe --defaults-file=/home/idl-face/databus5535/etc/my.cnf
21518 23095 21513 21513 ?           -1 Sl    1008   0:00 /home/idl-face/databus5535/libexec/mysqld --defaults-file=/home/idl-face/databus5535/etc/my.cnf --basedir=/home/idl-face/databus5535 --datadir=/home/idl-face/databus5535/var --plugin-dir=/home/idl-face/databus5535/lib/plugin --log-error=/home/idl-face/databus5535/log/mysql.err --pid-file=/home/idl-face/databus5535/var/mysql.pid --socket=/home/idl-face/databus5535/tmp/mysql.sock --port=5535
```

除了使用 PPID 和 PID 表示的父子进程关系外，进程间还有其他两种关系：

- 用 PGID 表示的进程组
- 用 SID 表示的会话

进程组（PGID）是一个或多个进程（通常与一个作业关联）的集合，我们可以使用该 PGID，通过 kill 命令向整个进程组发送信号：

```
##### 如上面命令返回PGID=21513 ,此处使用负数 -21513 表示PGID（进程组）
kill -SIGTERM -- -21513
```

我们用一个负数 `-21513` 向进程组发送信号。如果我们传递的是一个正数，这个数将被视为进程 ID 用于终止进程。如果我们传递的是一个负数，它被视为 PGID，用于终止整个进程组。

2、删除idl-face用户及其家目录，用于清理mysql底层数据（该操作会删除数据库内已注册的人脸数据，如库内无重要数据可忽略提示。）

```
userdel -r idl-face
```

3、进入新的部署包（V2）内，重新部署数据库

```
cd original/package/Applications/face-server/

输入解压命令  tar -xf project-conf.tar
进入original/package/Applications/face-server/project-conf/
bash mysql_start.sh
```

4、进入新的部署包(V2)内，更新人脸模型

```
cd original/package/Install
python2 install.py remove face-server
python2 install.py install face-server
```

5、进入新的部署包(V2)内，更新授权（可选）

```
cd original/package/Install
python2 install.py lu
```

回退模型操作 以人脸模型从V2回退至V1 举例说明：

1、终止databus数据库相关进程

首先通过如下命令查询databus相关进程信息

```
ps j -A |grep databus5535|grep -v "grep"

PPID   PID  PGID   SID TTY      TPGID STAT   UID   TIME COMMAND
1 21518 21513 21513 ?           -1 S     1008   0:00 /bin/sh /home/idl-face/databus5535/bin/mysqld_safe --defaults-file=/home/idl-face/databus5535/etc/my.cnf
21518 23095 21513 21513 ?           -1 Sl    1008   0:00 /home/idl-face/databus5535/libexec/mysqld --defaults-file=/home/idl-face/databus5535/etc/my.cnf --basedir=/home/idl-face/databus5535 --datadir=/home/idl-face/databus5535/var --plugin-dir=/home/idl-face/databus5535/lib/plugin --log-error=/home/idl-face/databus5535/log/mysql.err --pid-file=/home/idl-face/databus5535/var/mysql.pid --socket=/home/idl-face/databus5535/tmp/mysql.sock --port=5535
```

除了使用 PPID 和 PID 表示的父子进程关系外，进程间还有其他两种关系：

- 用 PGID 表示的进程组
- 用 SID 表示的会话

进程组（PGID）是一个或多个进程（通常与一个作业关联）的集合，我们可以使用该 PGID，通过 kill 命令向整个进程组发送信号：

```
##### 如上面命令返回PGID=21513 ,此处使用负数 -21513 表示PGID（进程组）
kill -SIGTERM -- -21513
```

我们用一个负数 `-21513` 向进程组发送信号。如果我们传递的是一个正数，这个数将被视为进程 ID 用于终止进程。如果我们传递的是一个负数，它被视为 PGID，用于终止整个进程组。

2、删除idl-face用户及其家目录，用于清理mysql底层数据（该操作会删除数据库内已注册的人脸数据，如库内无重要数据可忽略提示。）

```
userdel -r idl-face
```

3、进入新的部署包（V1）内，重新部署数据库

```
cd original/package/Applications/face-server/

输入解压命令  tar -xf project-conf.tar
进入original/package/Applications/face-server/project-conf/
bash mysql_start.sh
```

4、进入新的部署包(V1)内，更新人脸模型

```
cd original/package/Install
python2 install.py remove face-server
python2 install.py install face-server
```

5、进入新的部署包(V1)内，更新授权（可选）

```
cd original/package/Install
python2 install.py lu
```

若您仍然有部署及调用问题，请提交工单联系百度的工作人员

人脸离线识别SDK

概览

快速导览

- 功能介绍
- 激活授权
- 硬件选型
- 应用策略
- 产品定价
- Android-SDK

- WIN-C++-SDK
- Linux-ARM-SDK
- Android-SDK-常见问题
- 人脸识别特征值同步接口

获得支持

如果您对文档内容有任何疑问，可以通过以下几种方式联系我们：

- 在百度云控制台内 提交工单，咨询问题类型请选择**人工智能服务**；
- 如有需要讨论的疑问，欢迎进入 AI社区 与其他开发者们一同交流。

离线识别SDK是一个刚刚来到市场的小家伙，还有很多不足，请各位开发大大不吝指正批评，我们会努力快速优化，做到更好。

SDK基础信息介绍

产品概述

人脸离线识别SDK，包含人脸采集、活体检测、人脸对比/识别、人脸库管理等能力，并全部离线化、本地化。此SDK一经授权激活，可完全在无网环境下工作，所有数据皆在设备本地运行处理，可根据业务需要进行灵活的上层业务开发。核心能力分布如下图所示，后文会详细介绍。

适用场景特点

- **网络**：无网、局域网等情况，无法连接公网。如政府单位、金融保险、教育机构等。
- **安全**：行业特点所带来的人脸数据敏感性，即使可以连接公网也不可请求。
- **速度**：由于各地网络线路、机房部署等诸多原因，网络请求速度存在不可控因素。
- **稳定**：需要尽可能避免网络抖动、机房故障等影响，进一步控制可用性影响因素。

规格信息

- 包大小：~ 100M（包含全部功能，可根据需要自定义剪裁）
- 最小人脸检测大小：30px * 30px
- 可识别人脸角度：yaw ≤ ±30°, pitch ≤ ±30°
- 检测速度：70ms 720p*
- 追踪速度：20ms 720p*
- 人脸检测耗时：< 100ms

- RGB图片特征抽取耗时：< 150ms
- RGB活体检测耗时：< 100ms
- 近红外活体检测耗时：< 20ms
- 3D结构光活体检测耗时：< 10ms
- 5万本地人脸库检索速度：< 20ms

如RK3288，检测+活体+识别全流程耗时 < 300ms

> 备注：以上指标，以RK3288作为参考，由最新版SDK运行在真实设备上，采用真实数据集所得，但**算法性能受实际运行设备、实际数据集等情况影响**，以上数字仅供参考。

兼容性

Android 5.1+

授权方式

按设备授权

离线识别SDK授权方式为以设备维度为主，每台硬件设备需要一个独立的授权，此授权的校验是基于设备的硬件指纹（指纹的获取SDK初始化时会自动读取并展示），被授权的设备，将在有效期内可以运行SDK。

重新拉取授权的情况：设备授权不变，仅需要重新激活而已。

- 删除SDK或基于SDK开发的应用
- 刷安卓系统

授权失效的情况：需要重新购买序列号，之前的序列号失效。

- 激活一台设备后，此设备硬件变更
- 硬件损坏

序列号

序列号为管理授权的依据。每台被授权的设备，都将对应一个序列号，用于标识对应的设备信息及授权记录。序列号的形式为16位随机英文数字组合，如：`3G59-M5JK-889A-7LQA`。您在管理后台购买SDK授权时，购买成功后系统将会发放对应数量的序列号。序列号不限制平台版本，任何开发平台的离线SDK，都可以使用此序列号激活。序列号不限制账号，可供任何设备激活使用。

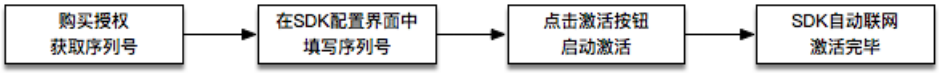
激活

已购买的序列号，是用于激活的唯一凭证，激活流程主要是将序列号与具体的硬件进行绑定（硬件指纹），从而生成对应硬件设备的授权文件（License），SDK运行前，将会校验授权文件是否和实际硬件信息相匹配。

> **注意**：激活时，设备系统时间需要和当前时间一致，如果差距太大（例如偏差5min以上），激活则无法完成。

联网激活

此种激活方式，适用于设备可首次联网的情况，优势在于激活过程极为简单。您只需将SDK安装到需要激活的设备上，然后填写已经购买的序列号，在界面上点击激活即可（为使用方便，我们为您设计了一个简单的激活用户界面）

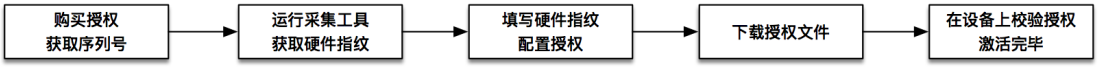


1. 获取序列号：从管理后台购买获取序列号。
2. SDK中填写序列号：将序列号填写到SDK的可视化界面中。
3. 启动激活：点击「激活」按钮。
4. SDK自动激活：SDK自动拉取授权文件并完成授权，激活完毕。

如激活成功，将会立即在界面上有明确的弹窗提示，请留意查看；如激活失败，也会反馈具体的错误信息。

离线激活（安卓后续版本支持）

此种激活方式，适用于设备完全不可联网的情况，优势在于可避免联网激活，满足业务对网络的严格要求，以及设备批量注册需求。您需要在后台配置好硬件指纹并完成和序列号的绑定，然后将授权文件放到SDK的指定位置。



1. 获取序列号：从管理后台购买获取序列号。
2. 采集硬件指纹：将SDK置于设备上，运行激活程序，获取硬件指纹。
3. 配置授权：在后台将硬件指纹绑定到具体序列号上。
4. 下载授权文件：绑定成功后下载授权文件。
5. 设备激活：将授权文件放到SDK中，并初始化SDK完成授权。

授权有效期

申请通过后，每个账户给**2个测试序列号**，用于激活及SDK试用，有效期为**自激活日期后的3个月**。这两个序列号在有效期内完全免费，您可以用于进行产品试用。试用期到期后也可以在后台申请延期，填写具体延期理由即可。

正式购买

正式购买的序列号，试用期限为**永久有效**。此「永久」是指绑定到具体设备维度，但如已绑定的硬件设备变更后，授权则可能会失效。

定价方式

离线SDK的授权基于设备维度，每个序列号仅可以授权一台设备。每个账号购买的序列号会累计计算，**累计购买量**越多，单价越便宜。具体如下所示：

累计购买授权数	每个授权单价
第100-999个	119元/个
第1000-4999个	99元/个
第5000个及以上	79元/个

立即去购买

核心功能

离线人脸检测与追踪

可在设备端，离线实时检测视频流中的人脸。同时支持处理静态图片或者视频流，并对当前检测到的人脸持续跟踪，动态定位

人脸轮廓，稳定贴合人脸。

离线质量控制

在人脸检测及追踪过程中，实时校验人脸的姿态角度、遮挡、清晰度、光照条件，符合质量条件的人脸图片才会被采集。

离线人脸采集

针对视频流实时完成人脸图片采集，并输出满足质量过滤条件的人脸图片，可自定义采集人脸大小，采集频率，采集质量等设置。

离线RGB可见光活体检测

针对视频流/图片，通过采集人像的破绽（摩尔纹、成像畸形等）来判断目标对象是否为活体，可有效防止屏幕二次翻拍等作弊攻击，可使用单张或多张判断逻辑。

离线NIR近红外活体检测

针对视频流/图片，利用近红外成像原理，实现夜间或无自然光条件下的活体判断。其成像特点（如屏幕无法成像，不同材质反射率不同等）可以实现高鲁棒性的活体判断。

离线Depth深度图像（3D结构光）活体检测

通过3D建模判断目标对象是否为活体，基于3D结构光成像原理，可强效防御图片、视频、屏幕、模具等攻击。

离线1：1对比

提供本地化的1：1人脸对比功能，高鲁棒性算法，可对应各种姿态、肤色、光照等场景，可有效应用于人证比对、身份核验等场景。

示例工程中包含：

- 图片与图片的比对：两张人脸图片的1：1对比，并返回相似度分值。
- 图片与视频流比对：一张预设的人脸图片，和摄像头实时采集的符合条件的人脸图片进行对比。

离线1：N搜索

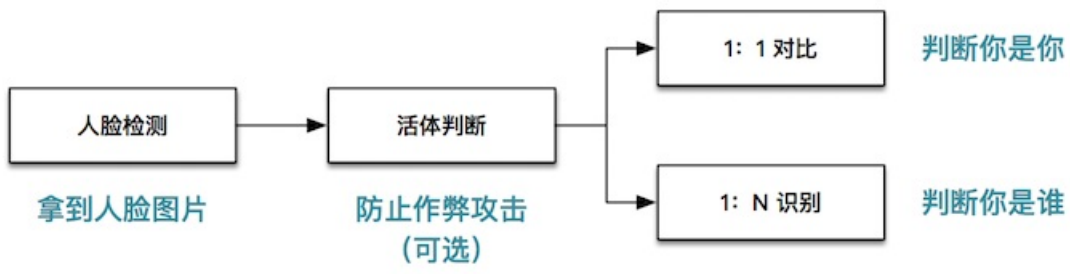
提供本地化1：N人脸搜索功能，即在指定人脸集合中，找到最相似的人脸。人脸库可存储于本地数据库中，进行人脸特征值比对。支持通过视频流实时采集人脸，并与人脸库中预设的人脸进行一一对比，返回相似度最高的user及对应分值。

离线人脸库管理

支持人脸库、人脸组、用户、Face4个维度的增删改查设置，**人脸库推荐5w以内，实际应用中1W以下最佳，可根据业务需要适当调整**。

业务应用策略

通用流程概述



如上图所示，人脸识别的核心业务流程可以分为三个步骤。

1. 检测采集：通过视频流实时检测跟踪，并采集到符合质量要求的人脸图片，用于后续的识别。
2. 活体判断：为可选步骤，主要保障业务操作者为真人，避免业务作弊。加上这步的校验，即只有满足活体判断通过，人脸图片才会被采集。
3. 对比识别：1：1对比主要是判断「你是你」，用于核实身份的真实性；1：N搜索主要判断「你是谁」，用于明确身份的具体所属。

活体检测

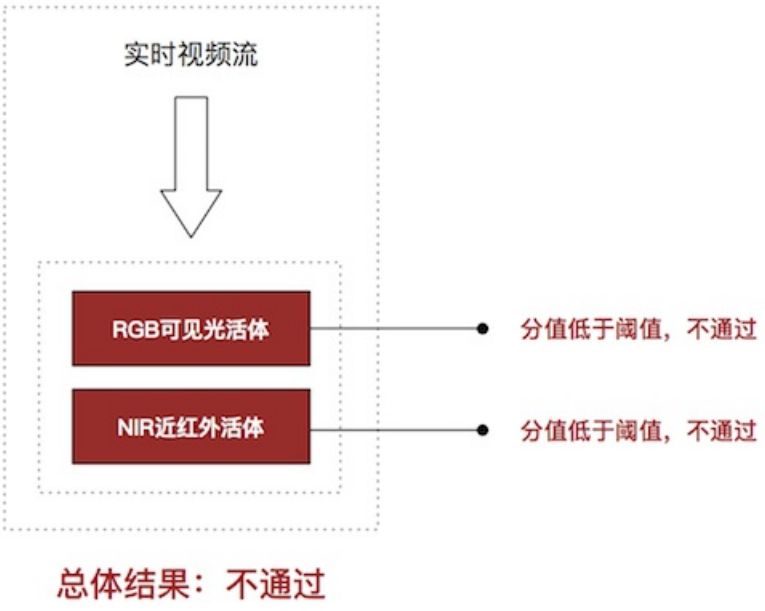
活体基础原理

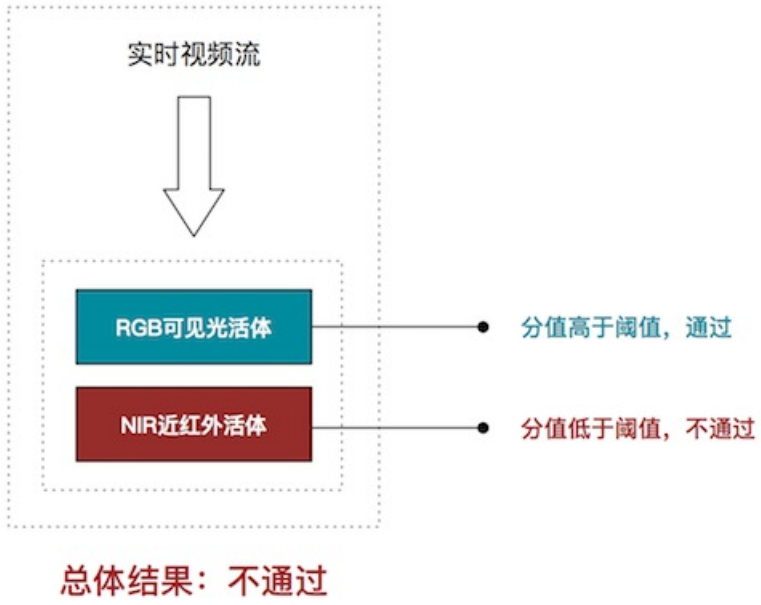
- RGB可见光活体：主要基于图片破绽，判断目标对象是否为活体。例如图像中的屏幕反光、成像畸形等，最主要的应用情形为屏幕的二次翻拍等攻击防御。此种活体对于待检测图片的要求，主要需要满足画面中除了人脸以外，要尽可能保留一些背景内容，用于查找破绽，通常建议人脸与屏幕的长宽比为1：3。为控制达到此比例，建议通过调整最小检测人脸参数，控制采集的人脸不可过大，避免人脸面积占比过高，而导致图片中没有多少背景信息。同时RGB活体受光线影响较大，所以在强光、暗光等场景，容易数值波动较大，主要影响的是「通过率」，产品策略上需要通过适当的补光、使用宽动态镜头抵消逆光等方式缓解。
- NIR近红外活体：主要基于近红外光线反射成像原理，通过人脸呈现来判断是否为活体。即使是夜间或者没有自然光的情况下，依然可以判断活体。因为其成像特点，对于屏幕、图片等攻击形式，基本可以达到近似于100%的活体防御。同时近红外设备的成本，相对性价比更高。
- Depth深度活体（3D结构光）：结构光原理是通过主动光发射，在物体上形成光栅，并接受此信息进行活体分析。同样可以不需要自然光。3D结构光的成像更为稳定，抗攻击能力更强，对图像噪声的抗干扰能力也更强，是相对更加安全和稳定的方案，但相应的硬件设备造价也较高，需要根据实际业务成本预算，进行综合考虑。

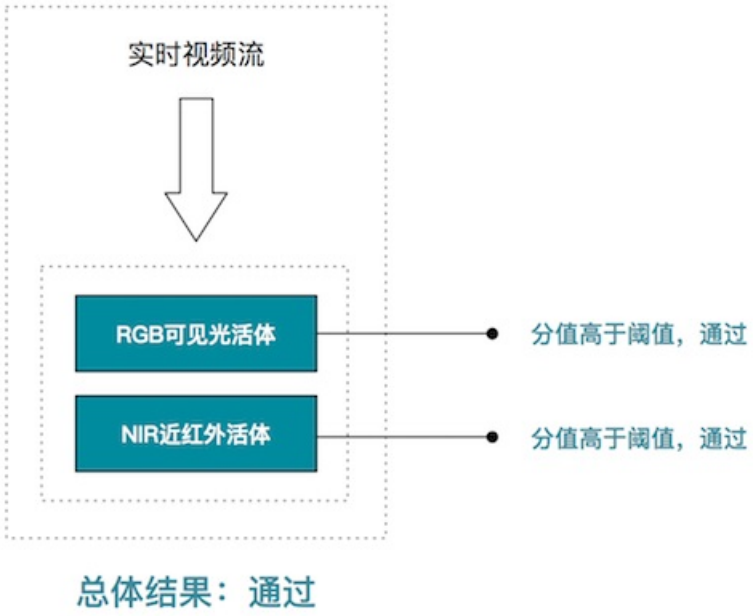
在实际业务场景中，需要根据场景特点，灵活组合使用以上几种活体方案。当然，为保证业务安全，还需要制定一系列的辅助措施，如证件信息读取、密码、其他生物特征识别等，达到更安全的核验。

产品应用策略

活体判断逻辑简单理解为：满足活体条件才可以采集图片，否则一直反复判断，直到满足活体条件为止。因为我们最终送去识别的图片是RGB图片，所以需要**保证活体通过时，在同一时间采集RGB图片，才可真正防止作弊攻击**。当多种活体叠加使用时，需要满足所有活体都通过，才能出发此操作，如果有任一活体没有通过，都不可进入识别步骤。







场景及应用方案

- 通行场景：此场景通常保障通行速度为主，确保不影响通行秩序和效率。所以建议无需使用三重活体检测，可仅用NIR活体或Depth活体，保障效率同时仍可保证安全性。
- 身份核验场景：此场景通常保障业务安全性为主，可尽可能提供更加安全的活体方案。如RGB+NIR，或者RGB+Depth，乃至RGB+NIR+Depth，最大程度确保对攻击的拒绝率较高。
- 强光/暗光场景：光线较强的场景，RGB活体会受影响比较严重，建议使用NIR或者Depth活体，同时尽量通过产品策略避免这两种情况的光线，例如添加补光灯、配备遮光板等。

应用方案及模组选择

- 无需活体：如有人值守的场景下，活体检测并无太大的必要（活体检测也会增加额外的业务耗时），现有设备的单目USB摄像头即可。
- 仅用RGB可见光活体：可使用设备自带的USB单目摄像头，输出的RGB图像可直接用于RGB可见光活体算法识别。
- RGB可见光+NIR近红外活体：需要配备能够同时获取RGB、NIR近红外数据的镜头使用。
- RGB可见光+Depth深度活体：需要配备能够同时获取RGB、Depth深度图像数据的镜头使用。

- RGB可见光+NIR近红外+Depth深度活体：需要配备能够同时获取RGB、NIR、Depth深度图像数据的镜头使用。

指标

活体检测存在几个标准的指标，如下所示：

- 拒绝率（TRR）：如99%，代表100次作弊假体攻击，会有99次被拒绝。
- 误拒率（FRR）：如0.5%，指1000次真人请求，会有5次因为活体分数低于阈值被错误拒绝。
- 通过率（TAR）：如99%，指100次真人请求，会有99次因为活体分数高于阈值而通过。
- 阈值（Threshold）：高于此数值，则可判断为活体。

温馨提示：此SDK涉及多种离线活体检测模型，加上镜头模组效果各异，使用环境也较为复杂，以下给出综合测试值，仅供参考：

- 拒绝率：> 99.5%
- 误拒率：< 1%
- 通过率：> 99%

阈值选择推荐

活体分值区间为[0~1]，大部分情况的活体攻击，活体分值近似于接近0.0，建议阈值如下，高于此阈值的即可判断为活体。

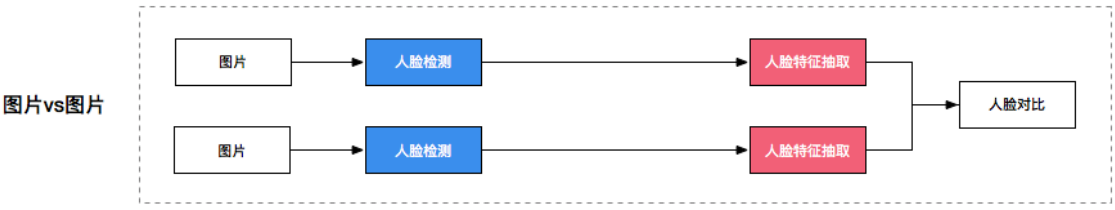
- RGB可见光活体：0.8
- NIR近红外活体：0.8
- Depth深度活体（3D结构光）：0.8

人脸1：1对比

人脸对比分为两种常见形态，具体如下：

对比类型：图片vs图片

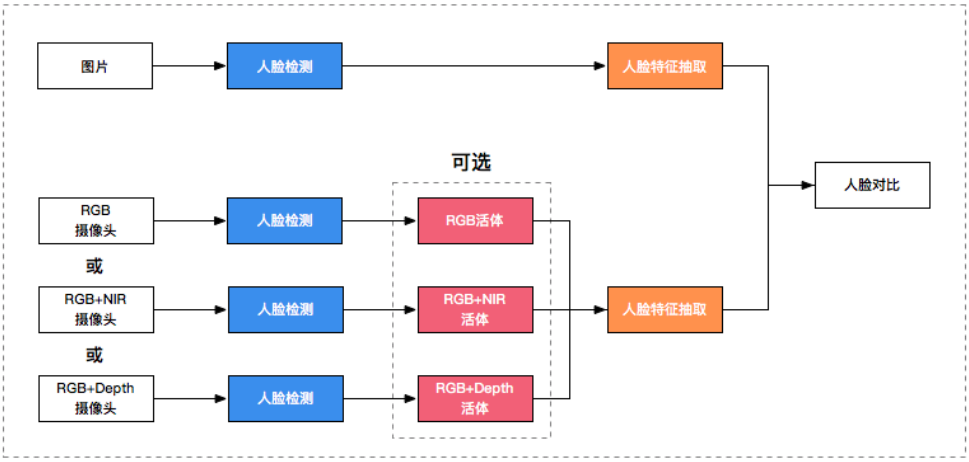
如果有两张待对比的图片（如证件照图片、与事先获取到的人脸生活照等），则可以直接通过离线SDK进行对比，业务流程如下图所示：



对比类型：实时视频流vs图片

最常见的1：1对比类型，一张事先获取的图片（通常为身份证芯片照、证件照片等），与现场人脸采集的图片进行对比。如果实际应用于为无人看守场景，则需要配备活体检测保障业务安全性。

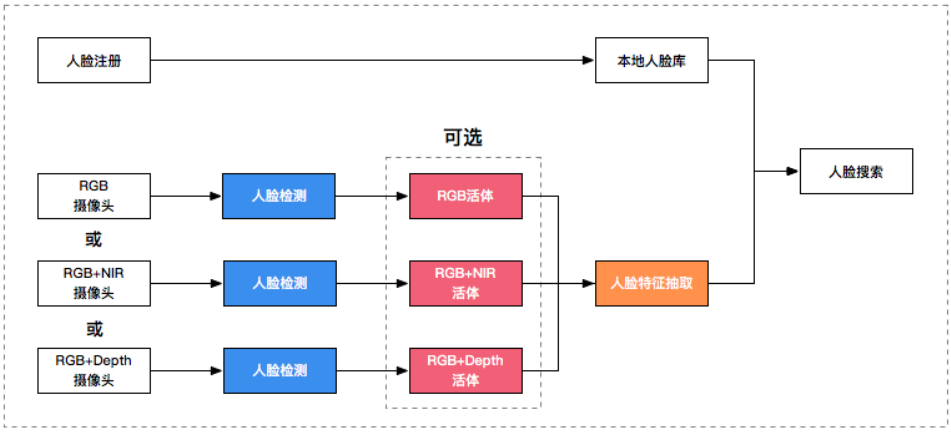
视频流vs图片



人脸1：N搜索

将需要识别的人脸图片集注册到本地人脸库中，当有用户需要识别身份时，从视频流中实时采集人脸图片，与人脸库中的人脸集合对比，得到搜索结果。如果需要为无人看守，则需要配备活体检测保障业务真实性和安全性。

视频流vs人脸库



设备选型

镜头模组选择

****无活体或仅用RGB单目活体****

uvc免驱单目USB摄像头，推荐**视派尔**单目摄像头，默认为3mm焦距镜头，可以选择6mm镜头，可以检测识别更远的人脸。

- **视派尔C-EP28WD400**，[产品规格书]

****RGB可见光+NIR近红外活体****

推荐使用如下品牌的近红外摄像头，可根据场景举例选择合适焦距摄像头。

- **迪威泰DV-BD4044S305AD**，[产品规格书]；联系人：杨先生；联系电话：18643209187
- **迪威泰DV-BD4053S305AD**，[产品规格书]；联系人：杨先生；联系电话：18643209187
- **视派尔C-EP35WDLDIR**，[产品规格书]；联系人：焦先生；联系电话：18676660044

NIR近红外实际检测效果回显示例：

** RGB可见光+Depth深度活体（3D结构光）**

推荐使用如下品牌的深度摄像头，可根据场景举例选择合适焦距摄像头。

- **奥比中光Astra Mini / Mini S**，[产品规格书]；联系人：牛妍；联系电话：13571984079；邮箱：niuyan@orbbec.com

Depth实际检测效果回显示例：

是否可以使用其他品牌镜头？

如果您需要使用NIR近红外或者Depth活体，则一定要配备支持对应图像格式类型的镜头模组。目前百度离线SDK适配的镜头类型有限，难以覆盖市面上所有双目模组。您可以查看下方的图像适配指标文档，只需满足文档中的图像/数据要求，即可自行进行模组适配。

[镜头模组图像适配指标.pdf]

我们也诚挚欢迎各镜头模组厂商参与到SDK的合作中来，我们会积极与您进行产品、技术、商务等多方面对接。测评效果较佳的模组，将会作为SDK官方适配模组，并在所有技术资料中优先推荐用户使用。您可以将合作申请发送到下方咨询邮箱，我们接到您的申请后会尽快与您取得联系，感谢您的信任。

合作邮件：ai#baidu.com（ #替换成@符号 ）

开发板选择（Android）

推荐使用RK3399，推荐硬件厂商及型号

- Firefly AIO-3399J，[产品规格介绍]，[固件支持]，[联系Firefly] ，联系电话：4001511533

机器选择（Windows）

- 系统：Windows 7、Windows 10系统的设备
- 处理器：推荐Intel i3及以上
- 内存：4.00 GB及以上
- 系统类型：32位、64位
- 具备2个及以上的USB插口
- 安装vs2015环境（不保证其他vs版本兼容性）

方案选型

离线SDK具备离线人脸检测、跟踪、质量校验、图像采集、RGB/NIR/3D结构光活体、离线对比识别、离线人脸库管理，仅需一个SDK即可完成业务能力覆盖。

目前，百度AI开放平台，也支持人脸离线采集SDK搭配公有云人脸比对/人脸搜索API能力的云端方案，以及人脸私有化部署包搭配人脸离线识别SDK的完全本地化方案，您可以根据实际的业务需要，选择合适的产品应用方案，以达到最佳的应用效果。

使用离线SDK

适用的场景特点

- 网络条件不稳定
- 无网络
- 数据安全性高，不允许联网
- 人脸库较小，且变更不频繁，如1w人以下
- 单台设备、单人脸检测与识别

> 注意：此版本离线SDK暂不支持多线程处理，主要适用于单人脸核验场景，不建议用于多人脸抓拍识别。

典型场景及设备类型

- 人脸门禁
- 人脸闸机
- 人证核验机
- 自助柜机
- 人脸考勤机

- 会场签到

> 提示：对于离线1：N场景，需要您基于人脸离线识别SDK构造上层的人脸库数据更新业务逻辑，例如小区门口的闸机为纯离线，但是住户的变更信息，如何同步到每一个大门的闸机中。

使用人脸离线采集SDK+人脸比对/人脸搜索API接口

适用的场景特点

- 网络条件良好
- 人脸库量级庞大
- 需要跨地域同步人脸库
- 人脸库更新频繁
- 用户APP产品形态

> 提示：人脸离线采集SDK具备本地化的人脸检测跟踪、质量校验、人脸采集等功能，与人脸离线识别SDK相比，活体和识别操作使用云端服务完成。

典型场景及设备类型

- 零售会员识别
- 人脸支付
- 手机APP
- 移动考勤
- 远程开户

使用人脸离线识别SDK+人脸私有化部署包

适用的场景特点

- 网络条件较差，或无网、局域网等
- 人脸库庞大，变更频繁（5W以上人脸库）
- 业务常为集中时段的高并发（如考勤、地铁通行）

典型场景及设备类型

- 园区刷脸考勤通行
- 社区闸机
- 楼宇门禁
- 轨道交通
- 机场、火车站安检
- 医疗挂号

功能介绍

产品概述

人脸离线识别SDK，包含人脸采集、活体检测、人脸对比/识别、人脸库管理等能力，并全部离线化、本地化。此SDK一经授权激活，可完全在无网环境下工作，所有数据皆在设备本地运行处理，可根据业务需要进行灵活的上层业务开发。核心能力分布如下图所示，后文会详细介绍。

适用场景特点

- **网络**：无网、局域网等情况，无法连接公网。如政府单位、金融保险、教育机构等。
- **安全**：行业特点所带来的人脸数据敏感性，即使可以连接公网也不可请求。
- **速度**：由于各地网络线路、机房部署等诸多原因，网络请求速度存在不可控因素。
- **稳定**：需要尽可能避免网络抖动、机房故障等影响，进一步控制可用性影响因素。

规格信息

- 包大小：~ 100M
- 最小人脸检测大小：50px * 50px
- 可识别人脸角度：yaw ≤ ±30°, pitch ≤ ±30°

- 检测速度：100ms 720p*
- 追踪速度：30ms 720p*
- 人脸检测耗时：< 100ms
- RGB图片特征抽取耗时：< 300ms
- RGB活体检测耗时：< 200ms
- 近红外活体检测耗时：< 50ms
- 3D结构光活体检测耗时：< 50ms
- 1万本地人脸库检索速度：< 400ms

> 备注：以上指标，由最新版SDK运行在真实设备上，采用真实数据集所得，但**算法性能受实际运行设备、实际数据集等情况影响**，以上数字仅供参考。

兼容性

Android 4.4+

硬件适配情况请详见后文：1.4 硬件选型

核心功能

离线人脸检测与追踪

可在设备端，离线实时检测视频流中的人脸。同时支持处理静态图片或者视频流，并对当前检测到的人脸持续跟踪，动态定位人脸轮廓，稳定贴合人脸。

离线质量控制

在人脸检测及追踪过程中，实时校验人脸的姿态角度、遮挡、清晰度、光照条件，符合质量条件的人脸图片才会被采集。

离线人脸采集

针对视频流实时完成人脸图片采集，并输出满足质量过滤条件的人脸图片，可自定义采集人脸大小，采集频率，采集质量等设置。

离线RGB可见光活体检测

针对视频流/图片，通过采集人像的破绽（摩尔纹、成像畸形等）来判断目标对象是否为活体，可有效防止屏幕二次翻拍等作弊攻击，可使用单张或多张判断逻辑。

离线NIR近红外活体检测

针对视频流/图片，利用近红外成像原理，实现夜间或无自然光条件下的活体判断。其成像特点（如屏幕无法成像，不同材质反射率不同等）可以实现高鲁棒性的活体判断。

离线Depth深度图像（3D结构光）活体检测

通过3D建模判断目标对象是否为活体，基于3D结构光成像原理，可强效防御图片、视频、屏幕、模具等攻击。

离线1：1对比

提供本地化的1：1人脸对比功能，高鲁棒性算法，可对应各种姿态、肤色、光照等场景，可有效应用于人证比对、身份核验等场景。

示例工程中包含：

- 图片与图片的比对：两张人脸图片的1：1对比，并返回相似度分值。
- 图片与视频流比对：一张预设的人脸图片，和摄像头实时采集的符合条件的人脸图片进行对比。

离线1：N搜索

本地数据库中保留所有人脸特征值（如需要保留原图，可根据业务需要自行修改工程）。

- 视频流采集的人脸在人脸库中搜索：视频流中实时采集人脸，并与人脸库中预设的人脸库进行一一对比，返回相似度最高的user及对应分值。

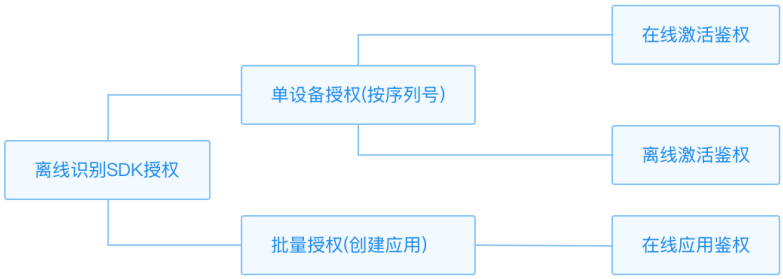
离线人脸库管理

人脸数量不做上限，可根据业务需要适当调整，支持人脸库、人脸组、用户、Face几个维度的增删改查设置。

激活授权

人脸离线识别SDK的激活方式分为「软授权」和「硬授权」两种：

- 软授权详情如下所示，主要分为单设备授权（按序列号）和批量授权（创建应用）两种形式，其中单设备授权支持在线和离线两种方式



- 硬授权主要是加密芯片授权，需要焊接加密芯片完成永久安全授权操作，加密芯片型号atsha204a，如需规格书请线下联系PM/PMM。

序列号

序列号为管理授权的依据。每台被授权的设备，都将对应一个序列号，用于标识对应的设备信息及授权记录。序列号的形式为16位随机英文数字组合，如：`3G59-M5JK-889A-7LQA`。您在管理后台购买SDK授权时，购买成功后系统将会发放对应数量的序列号。序列号不限制平台版本，任何开发平台的离线SDK，都可以使用此序列号激活。序列号不限制账号，可供任何设备激活使用。

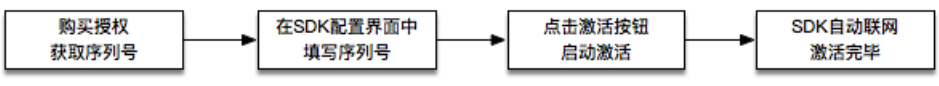
什么是激活

已购买的序列号，是用于激活的唯一凭证，激活流程主要是将序列号与具体的硬件进行绑定（硬件指纹），从而生成对应硬件设备的授权文件（License），SDK运行前，将会校验授权文件是否和实际硬件信息相匹配。

> **注意**：激活时，设备系统时间需要和当前时间一致，如果差距太大（例如偏差5min以上），激活则无法完成。

联网激活

此种激活方式，适用于设备可首次联网的情况，优势在于激活过程极为简单。您只需将SDK安装到需要激活的设备上，然后填写已经购买的序列号，在界面上点击激活即可（为使用方便，我们为您设计了一个简单的激活用户界面）



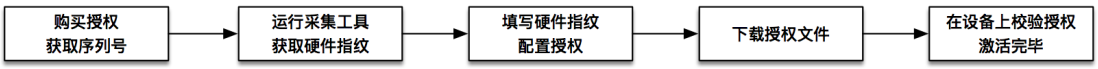
1. 获取序列号：从管理后台购买获取序列号。

2. SDK中填写序列号：将序列号填写到SDK的可视化界面中。
3. 启动激活：点击「激活」按钮。
4. SDK自动激活：SDK自动拉取授权文件并完成授权，激活完毕。

如激活成功，将会立即在界面上有明确的弹窗提示，请留意查看；如激活失败，也会反馈具体的错误信息。

单设备离线激活

此种激活方式，适用于设备完全不可联网的情况，优势在于可避免联网激活，满足业务对网络的严格要求，以及设备批量注册需求。您需要在后台配置好硬件指纹并完成和序列号的绑定，然后将授权文件放到SDK的指定位置。



1. 获取序列号：从管理后台购买获取序列号。
2. 采集硬件指纹：将SDK置于设备上，运行激活程序，获取硬件指纹。
3. 配置授权：在后台将硬件指纹绑定到具体序列号上。
4. 下载授权文件：绑定成功后下载授权文件。
5. 设备激活：将授权文件放到SDK中，并初始化SDK完成授权。

授权有效期

申请通过后，每个账户给**2个测试序列号**，用于激活及SDK试用，有效期为**自激活日期后的3个月**。这两个序列号在有效期内完全免费，您可以用于进行产品试用。试用期到期后也可以在后台申请延期，填写具体延期理由即可。

正式购买

正式购买的序列号，试用期限为**永久有效**。此「永久」是指绑定到具体设备维度，但如已绑定的硬件设备变更后，授权则可能会失效。

激活鉴权常见问题

1. 扩容硬盘后，出现指纹变更，经过USB外接硬盘测试，硬件指纹并没有发生变化。
2. 切换网络如有线网络切换至无线网络，硬件指纹发生变更，经过切换网络测试，禁用网络测试，设置时候用网络随机地址，测试，指纹信息没有发生改变。
3. 使用Windwos工控机，拔掉继电器，导致指纹信息变更。
4. 使用虚拟机，重启之后发生了指纹变更。
5. 重装系统后，导致指纹信息变更。
6. 系统时间变更，导致授权失败。
7. 插上U盘，指纹信息发生变更，通过本地测试，未复现。
8. CPU、网卡等硬件损坏导致硬件指纹变更。
9. 已经激活的设备硬件变更、固件升级导致指纹变化。
10. 设备系统升级、SDK系统升级（小概率）。

常见问题解决方案详见Android-sdk-常见问题

硬件选型

镜头模组选择

一、无活体或仅用RGB单目活体

UVC免驱单目USB摄像头，默认推荐3mm、3.5mm焦距镜头，可以选择6mm镜头，可以检测识别更远的人脸。

- **WEGO摄像头+百度人脸识别SDK序列码套装**：[立即购买]
- **WEGO摄像头C系列+百度离线人脸识别SDK激活码**：[立即购买]
- **视派尔C-EP28WD400**：[立即购买]
- **视派尔C-EP28WD600**：[立即购买]
- **视派尔C-EP28WD305**：[立即购买]
- **视派尔-RP28WD305**：[立即购买]

二、RGB可见光+NIR近红外活体

推荐使用如下品牌的近红外摄像头，可根据场景举例选择合适焦距摄像头。

- **WG-SS-A1双目镜头+百度SDK激活码套装**：[立即购买]
- **迪威泰DV-BD4044S305AD**：[立即购买]
- **视派尔-RP22PWDLDIRV403**：[立即购买]
- **视派尔-EP17WDLMIR305**：[立即购买]
- **视派尔C-EP36WDLDIR**：[立即购买]

NIR近红外实际检测效果回显示例：

三、RGB可见光+Depth深度活体（结构光 or ToF）

推荐使用如下品牌的深度摄像头，可根据场景举例选择合适焦距摄像头。

- **奥比中光 Astra Atlas**：[立即购买]
- **奥比中光-大白**：[立即购买]
- **奥比中光 Astra Pro S**：[立即购买]
- **奥比中光 Astra Deeyea**：[立即购买]
- **奥比中光 Astra Mini / Mini S**：[立即购买]
- **奥比中光 Petrel 海燕3D结构光模组**：[立即购买]
- **华捷艾米 IMI A200+mini 3D结构光模组**
- **PicoZense TOF DCAM305**：[立即购买]
- **艾芯智能 TOF M5**：[立即购买]

Depth实际检测效果回显示例（结构光或ToF镜头）：

四、使用其他品牌镜头

如果您需要使用NIR近红外或者Depth活体，**一定要配备支持对应图像格式类型的镜头模组，如果没有进行硬件对齐等调整，无法保证效果**。目前百度离线SDK适配的镜头类型有限，难以覆盖市面上所有镜头模组。您可以查看下方的图像适配指标文档，只需满足文档中的图像/数据要求，即可自行进行模组适配。

[镜头模组图像适配指标.pdf]

我们也诚挚欢迎各镜头模组厂商参与到SDK的合作中来，我们会积极与您进行产品、技术、商务等多方面对接。测评效果较佳的模组，将会作为SDK官方适配模组，并在所有技术资料中优先推荐用户使用。您可以将合作申请发送到下方咨询邮箱，我们接到您的申请后会尽快与您取得联系，感谢您的信任。

合作邮件：ai#baidu.com（ #替换成@符号 ）

开发板选择（Android）

- 内存：2G最佳

- 主频：1.3GHz以上
- 芯片：RockChip 3399/3288、高通8953，其他芯片型号需要您自行适配

推荐硬件厂商及型号：

- 百度人脸识别开发套件—壁虎：[立即购买]
- 创百RK3288人脸识别终端：[立即购买]
- 创百 RK3399 工控主板：[立即购买]
- 创百 RK3399 工控主板：[立即购买]
- Firefly RK3399 工控主板，[了解详情]
- Firefly AIO-3399J 行业主板，[了解详情]
- 音诺恒 RK3399 工控主板，[立即购买]

机器选择（Windows）

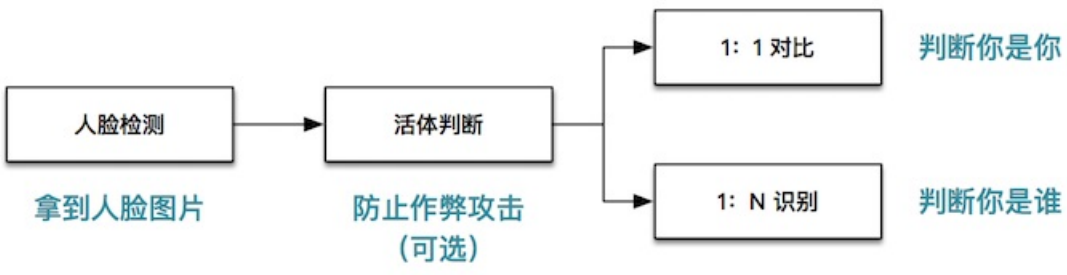
- 系统：Windows 7、Windows 10系统的设备，不支持Windows Server
- 处理器：推荐Intel i3及以上
- 内存：4GB及以上
- 系统类型：32位、64位
- 具备2个及以上的USB插口
- 安装VS2015及以上环境（不保证其他vs版本兼容性）

机器选择（Linux）

- 系统：Ubuntu 14.04及以上、Centos:6.3及以上
- 处理器：根据业务实际需求调整
- 内存：4GB及以上
- 系统类型：64位
- gcc g++ 4.8.2及以上

应用策略

通用流程概述



如上图所示，人脸识别的核心业务流程可以分为三个步骤。

1. 检测采集：通过视频流实时检测跟踪，并采集到符合质量要求的人脸图片，用于后续的识别。

2. 活体判断：为可选步骤，主要保障业务操作者为真人，避免业务作弊。加上这步的校验，即只有满足活体判断通过，人脸图片才会被采集。
3. 对比识别：1：1对比主要是判断「你是你」，用于核实身份的真实性；1：N搜索主要判断「你是谁」，用于明确身份的具体所属。

活体检测

活体基础原理

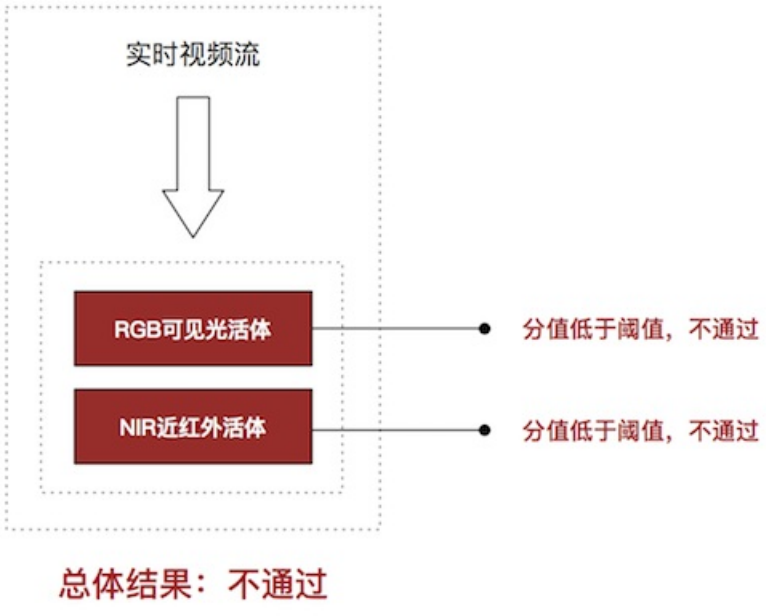
- RGB可见光活体：主要基于图片破绽，判断目标对象是否为活体。例如图像中的屏幕反光、成像畸形等，最主要的应用情形为屏幕的二次翻拍等攻击防御。此种活体对于待检测图片的要求，主要需要满足画面中除了人脸以外，要尽可能保留一些背景内容，用于查找破绽，通常建议人脸与屏幕的长宽比为1：3。为控制达到此比例，建议通过调整最小检测人脸参数，控制采集的人脸不可过大，避免人脸面积占比过高，而导致图片中没有多少背景信息。同时RGB活体受光线影响较大，所以在强光、暗光等场景，容易数值波动较大，主要影响的是「通过率」，产品策略上需要通过适当的补光、使用宽动态镜头抵消逆光等方式缓解。
- NIR近红外活体：主要基于近红外光线反射成像原理，通过人脸呈现来判断是否为活体。即使是夜间或者没有自然光的情况下，依然可以判断活体。因为其成像特点，对于屏幕、图片等攻击形式，基本可以达到近似于100%的活体防御。同时近红外设备的成本，相对性价比更高。
- Depth深度活体（3D结构光）：结构光原理是通过主动光发射，在物体上形成光栅，并接受此信息进行活体分析。同样可以不需要自然光。3D结构光的成像更为稳定，抗攻击能力更强，对图像噪声的抗干扰能力也更强，是相对更加安全和稳定的方案，但相应的硬件设备造价也较高，需要根据实际业务成本预算，进行综合考虑。

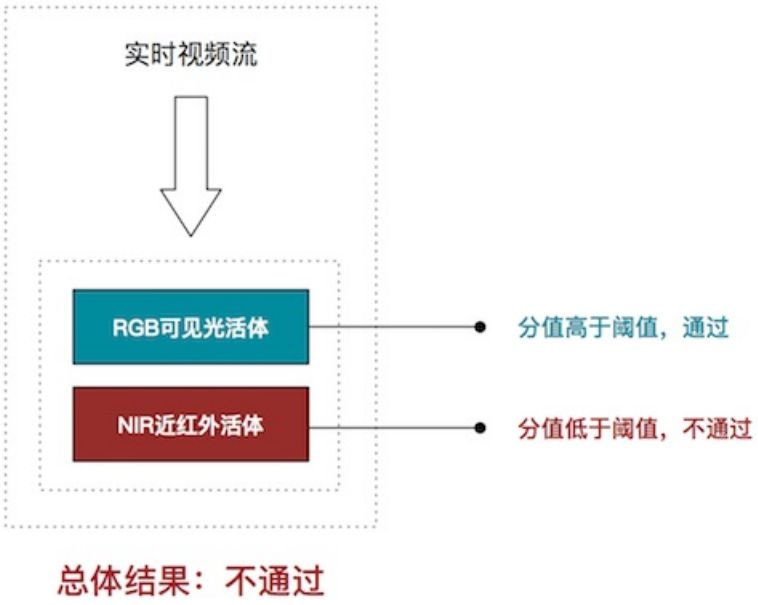
以上，无论活体是否可以通过，我们业务的最终目的，还是要进行「对比识别」，所以**拿到合适的RGB图像还是最为关键**，保障符合条件的RGB图像获取也是需要在活体判断同时，最为需要关注的事情。

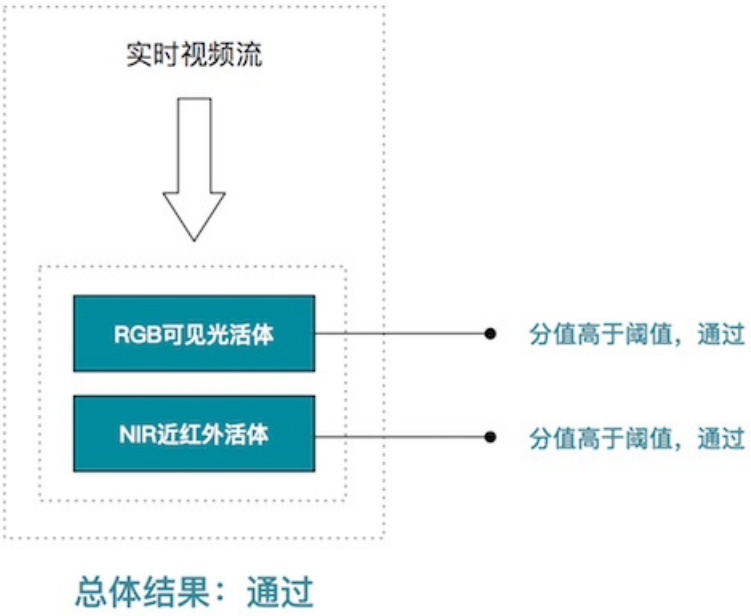
在实际业务场景中，需要根据场景特点，灵活组合使用以上几种活体方案。当然，实际业务中，**没有绝对100%的安全**，在应用过程中，还需要根据业务流程特点，指定一系列的辅助措施，如证件信息读取、密码、其他生物特征识别等，达到刚安全的核验。

产品应用策略

活体判断逻辑简单理解为：满足活体条件才可以采集图片，否则一直反复判断，直到满足活体条件为止。因为我们最终送去识别的图片是RGB图片，所以需要**保证活体通过时，在同一时间采集RGB图片，才可真正防止作弊攻击**。当多种活体叠加使用时，需要满足所有活体都通过，才能出发此操作，如果有任一活体没有通过，都不可进入识别步骤。







场景及应用方案

- 通行场景：此场景通常保障通行速度为主，确保不影响通行秩序和效率。所以建议无需使用三重活体检测，可仅用NIR活体或Depth活体，保障效率同时仍可保证安全性。
- 身份核验场景：此场景通常保障业务安全性为主，可尽可能提供更加安全的活体方案。如RGB+NIR，或者RGB+Depth，乃至RGB+NIR+Depth，最大程度确保对攻击的拒绝率较高。
- 强光/暗光场景：光线较强的场景，RGB活体会受影响比较严重，建议使用NIR或者Depth活体，同时尽量通过产品策略避免这两种情况的光线，例如添加补光灯、配备遮光板等。

应用方案及模组选择

- 无需活体：如有人值守的场景下，活体检测并无太大的必要（活体检测也会增加额外的业务耗时），现有设备的单目USB摄像头即可。
- 仅用RGB可见光活体：如果您的设备已经配备了USB单目摄像头，则无需更换，输出的RGB图像可直接用于RGB可见光活体算法识别。
- RGB可见光+NIR近红外活体：需要配备能够同时获取RGB、NIR近红外数据的镜头使用。

- RGB可见光+Depth深度活体：需要配备能够同时获取RGB、Depth深度图像数据的镜头使用。
- RGB可见光+NIR近红外+Depth深度活体：此种方式安全度最高，目前SDK正在模组适配中，后续会陆续推出已适配的模组方案，敬请期待。

指标

活体检测存在几个标准的指标，如下所示：

- 拒绝率（TRR）：如99%，代表100次作弊假体攻击，会有99次被拒绝。
- 误拒率（FRR）：如0.5%，指1000次真人请求，会有5次因为活体分数低于阈值被错误拒绝。
- 通过率（TAR）：如99%，指100次真人请求，会有99次因为活体分数高于阈值而通过。
- 阈值（Threshold）：高于此数值，则可判断为活体。

温馨提示：此SDK涉及多种离线活体检测模型，加上镜头模组效果各异，使用环境也较为复杂，以下给出综合测试值，仅供参考：

- 拒绝率：> 99.5%
- 误拒率：< 1%
- 通过率：> 99%

阈值选择推荐

活体分值区间为[0~1]，大部分情况的活体攻击，活体分值近似于接近0.0，建议阈值如下，高于此阈值的即可判断为活体。

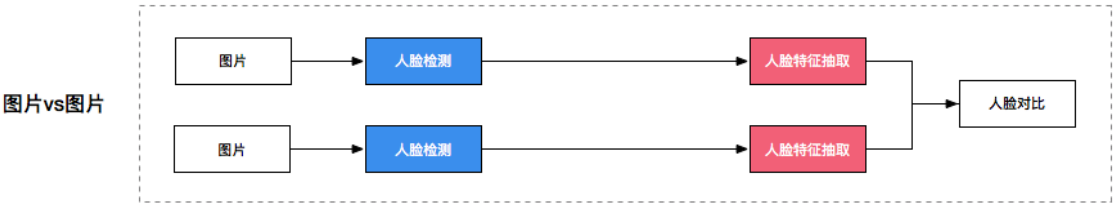
- RGB可见光活体：0.8
- NIR近红外活体：0.8
- Depth深度活体（3D结构光）：0.8

人脸1：1对比

人脸对比分为两种常见形态，具体如下：

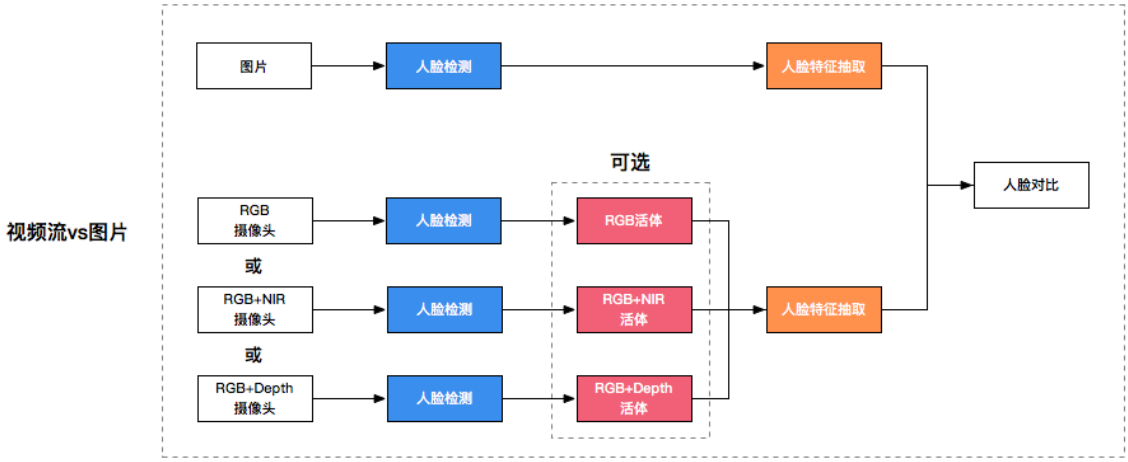
对比类型：图片vs图片

如果有两张待对比的图片（如证件图片、以事先获取到的人脸生活照等），则可以直接通过离线SDK进行对比，业务流程如下图所示：



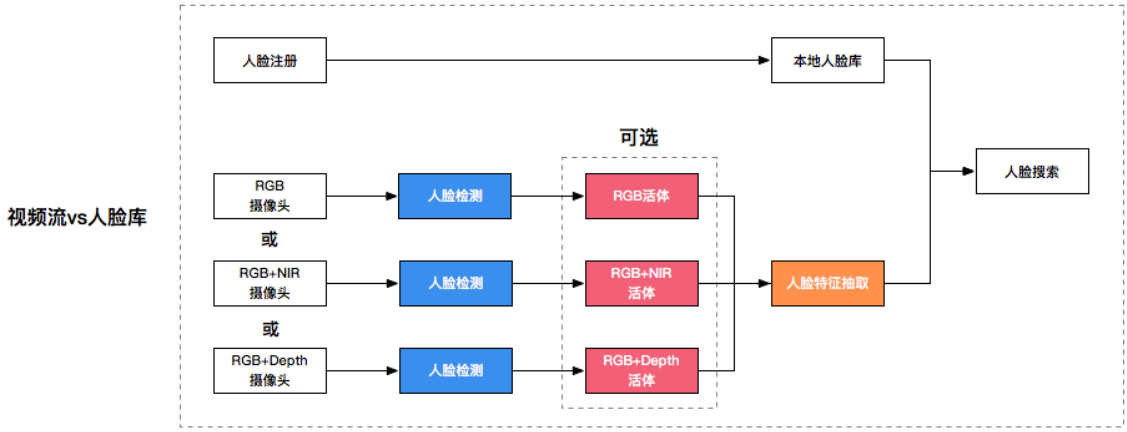
对比类型：实时视频流vs图片

这是最为常见的1：1对比类型，一张事先获取的图片（通常为身份证芯片照、证件照片等），与现场人脸采集的图片进行对比。如果需要为无人看守，则需要配备活体检测保障业务真实性和安全性。



人脸1：N搜索

将需要识别的人脸图片集注册到本地人脸库中，当有用户需要识别身份时，从视频流中实时采集人脸图片，与人脸库中的人脸集合对比，得到搜索结果。如果需要为无人看守，则需要配备活体检测保障业务真实性和安全性。



产品定价

计费简介

人脸离线SDK根据设备授权数量收取费用，按累计阶梯计费，随着购买数量的增加，授权单价将会降低。购买授权并激活后，即可下载授权文件，永久有效。 每个账号下默认分配**5个试用版序列号**，有效期为三个月，供您前期测试。

价目表

价格为阶梯计费，可根据您的累计购买数量，查看具体价格

独立购买SDK授权数N	每个授权单价
100≤N<1000	119元/个
1000≤N<5000	99元/个
5000≤N<10000	79元/个
N≥10000	商务咨询

购买说明

1. 正式版授权有效期为永久，即如果设备信息没有变化（硬件指纹不变化），授权一直有效可用（如软件重装，原授权文件仍然有效，重新激活即可）
2. 按照所需授权个数购买后，可通过同一个授权池下发给两种不同的授权模式：单设备（序列号）授权和批量（应用）授权，并支持对授权进行回收

3. 授权不受系统平台限制，支持在Android、Windows、Linux等多平台激活
4. 序列号的购买是「独立阶梯计费」，例如您需购买900个授权，采购区间对应100-1000的之间，所以需付费119元/个 x 900个 = 107,100元； 当您购买2000个授权，采购区间对应1000-4999之间，则付费 99元/个 x 2000=198,000元；若您的购买量达到10000个及以上，请您商务咨询，我们会有商务团队跟您线下对接
5. SDK购买激活后，或购买超过7天，不支持退款

Android-SDK

概述

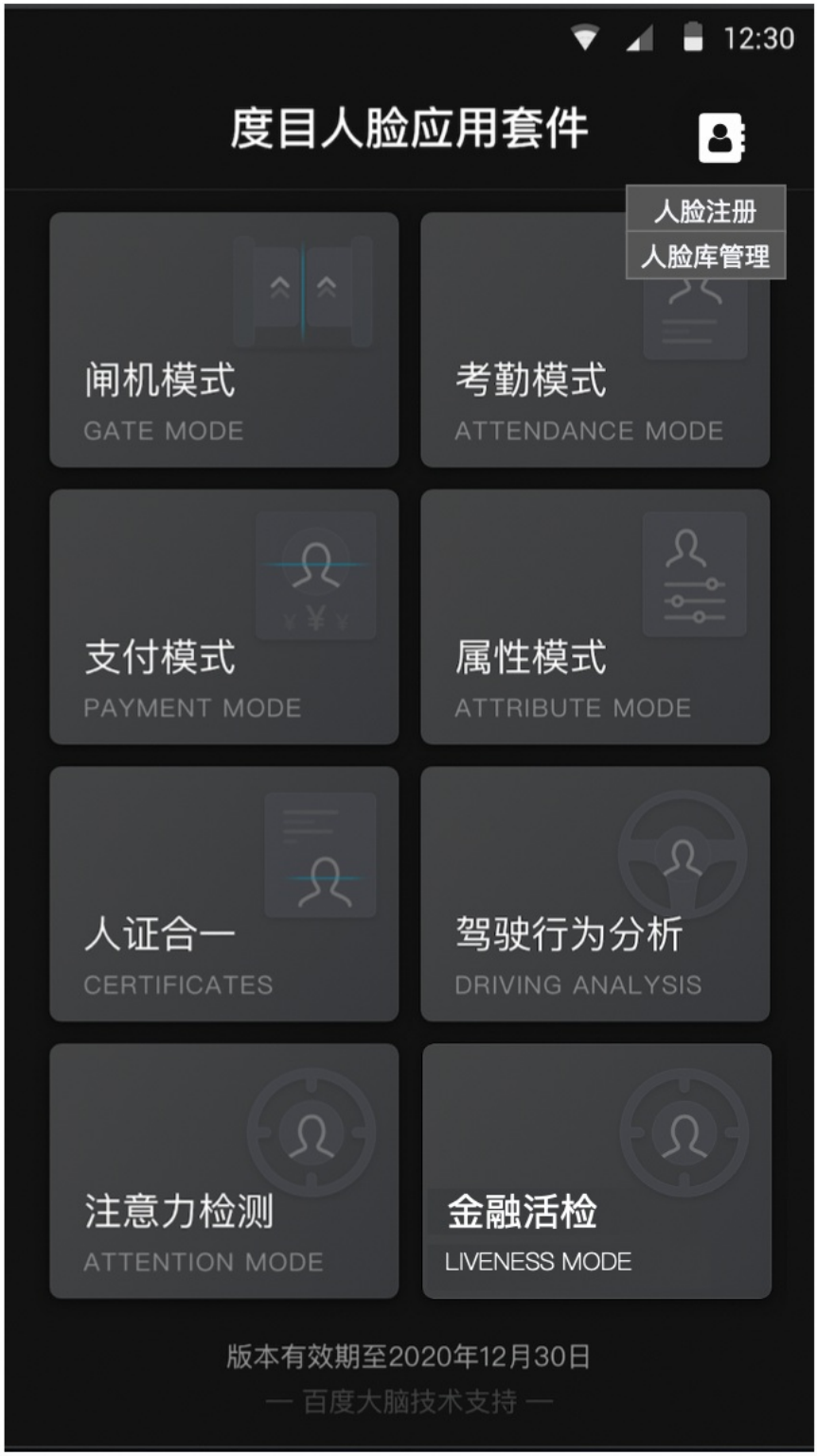
- 百度人脸离线识别SDK Android版本，主要支持RK3288、3399芯片平台，且已验证支持高通450/8953、MTK6763/8788（可登陆百度智能云控制台console后台获取SDK部署包）
- 针对其他安卓平台的芯片，例如RK3568/3566、晶晨S905D3(CPU/NPU)、全志A63/T509，为了保障每个芯片平台上算法效果最优，都有专项适配调优的SDK版本，请线下联系商务咨询获取专版SDK。

版本日志

版本	日期	更新说明
V8.2.0	2024.06.07	1、优化了人脸特征提取模型； 2、考勤模块下新增了多人脸识别功能并支持UI横屏识别
V8.1.2	2023.03.02	1、接入动态底库策略，解决由于底库质量差带来的误识别问题； 2、优化口罩检测、安全帽检测模型； 3、添加活体多帧策略，配合阈值设备同解决多镜头泛化问题； 4、添加加密芯片授权支持，软硬授权并线支持； 5、支持多款3D结构光模组兼容适配
v8.1	2022.11.18	1、修复日志信息错误bug； 2、去除应用层的网络判断逻辑，解决每次设备断网重启重新激活的问题； 3、添加安卓11系统兼容性配置
v8.0	2022.04.20	1、检测、活体、识别更新至8.0最新算法版本； 2、解决旧版本偶发崩溃问题，支持边注册、边识别； 3、隐藏情绪识别功能，优化Demo中人脸追踪框的效果
v7.0	2021.08.24	1、更新人脸检测模型，解决部分场景下非人脸误检测的问题； 2、全面更新多模态活体检测模型，提升对新材质活体攻击道具的防御能力； 3、更新人脸识别模型，实现单模型同时支持戴口罩人脸识别和外国人识别； 4、增加业务层二次开发接口，降低SDK集成难度，保障端到端算法效果； 5、完善示例工程功能，优化示例工程交互； 6、修复部分已知Bug
v6.0	2021.02.26	1、更新人脸检测模型，提升在强光、暗光、逆光、阴阳光等复杂光线场景下人脸检测的召回率； 2、更新RGB、NIR和Depth三种模态的活体检测模型，提升在复杂光线场景下真人活体检测通过率，同步提升高清2D和高质量3D道具的活体攻击防御效果； 3、更新人脸识别模型，提升复杂光线场景下人脸识别的准确率； 4、新增暗光恢复功能，对暗光场景下的图片进行亮度提升，提高检测和识别的准确率； 5、区分『通用版』和『口罩版』，通用版适用于非戴口罩场景下的人脸识别，口罩版适用于在戴口罩场景下完成人脸识别； 6、修复部分已知Bug
v5.1	2021.01.01	1、新增『金融活检』示例工程，该示例工程集成了多帧检测等活体检测策略，提升活体检测准确率； 2、新增『RGB+NIR』混合模态人脸识别示例工程，解决暗光人脸识别问题，提升弱光人脸识别准确率； 3、优化示例工程的功能和代码结构，更有利于直接使用和进行二次开发； 4、修复部分已知Bug

		4、修复部分已知Bug
v5.0	2020.10.16	1、升级了了人脸检测模型，重点优化了复杂光线场景下的人脸检测能力； 2、升级了RGB、NIR和Depth三种模态的活体检测模型，优化后2D照片攻击防御能力和复杂光线场景下的真人活体检测能力提升明显； 3、升级了最新的人脸特征抽取与比对模型（人脸识别模型），升级后识别模型对老人、儿童等极端年龄群体的泛化行提升明显； 4、在RK3288主板上，端到端全流程耗时缩短到<300ms； 5、增加了最优图像帧功能，提升识别准确率； 6、增加多线程的示例工程，支持边注册、边识别和多模态同时检测功能； 7、解决了部分已知Bug。 **注：由于该版本的SDK更换了特征抽取与比对的模型，因此从旧版本升级到该版本的用户需要重新注册人脸底库（刷库）**
v4.2.1	2020.07.07	1、增加场景化示例工程，针对「闸机通行」、「考勤打卡」、「金融支付」、「属性检测」、「人证核验」、「驾驶员行为分析」和「注意力检测」7大应用场景提供定制化的交互流程和交互界面； 2、优化示例工程中的注册激活功能，提供快速、便捷的「在线激活」、「离线激活」和「批量激活」功能； 3、优化示例工程中人脸库管理功能和设置功能； 4、解决部分已知Bug **注：4.2.1版本主要针对示例工程进行了全面重构和优化，功能和接口与4.1.1版本保持一致**
v4.1.1	2020.03.12	1、新增注意力检测模型和接口； 2、新增驾驶行为分析模型和接口； 3、优化口罩检测模型； 4、解决部分已知Bug **注：若使用4.1.1版本中注意力检测功能，请参考章节1.6的内容；若使用4.1.1版本中驾驶员行为分析功能，请参考章节1.7的内容**
v4.0.0	2020.02.25	1、新增NIR检测、识别模态； 2、新增RGBD识别模态； 3、优化检测、对齐、活体、证件照/生活照识别模型； 4、优化端到端全流程耗时； 5、新增人脸是否戴口罩分类能力和接口； 6、新增抠图能力和接口 7、解决部分已知Bug **注：若使用4.0版本中口罩检测能力，请务必参考章节1.11.1的内容**
v3.2.0	2019.12.10	1、底层支持多线程； 2、增加睁闭眼、张闭嘴模型； 3、优化NIR界面适配效果； 4、修复部分已知Bug **注：由3.0/3.1版本升级到3.2的客户，请按照章节1.11.2的方案进行初始化修改**
v3.1.0	2019.08.13	1、优化人脸检测性能； 2、新增离线人脸属性模型，包括年龄、种族、性别、7种细分情绪和3种分类情绪、佩戴眼镜等属性
v3.0.0	2019.07.25	1、新版检测模型，人脸检测及跟踪速度大幅提升； 2、新版RGB、NIR、Depth活体模型，优化活体检测的有效人脸大小尺寸范围，活体通过率大幅提升。增加更多攻击类型样本数据，非活体拒绝率大幅提升； 3、新版生活照模型，特征抽取速度大幅提升； 4、新版接口实现，功能逻辑精简优化，更易集成； 5、新版示例工程，优化功能逻辑，新增大量自定义设置项，效果调节更简单； 6、新增「按应用批量鉴权」，应用初始化时自动联网激活。
v2.0.3	2019.06.14	1、更新硬件指纹获取模块，优化特殊环境下指纹变更的问题

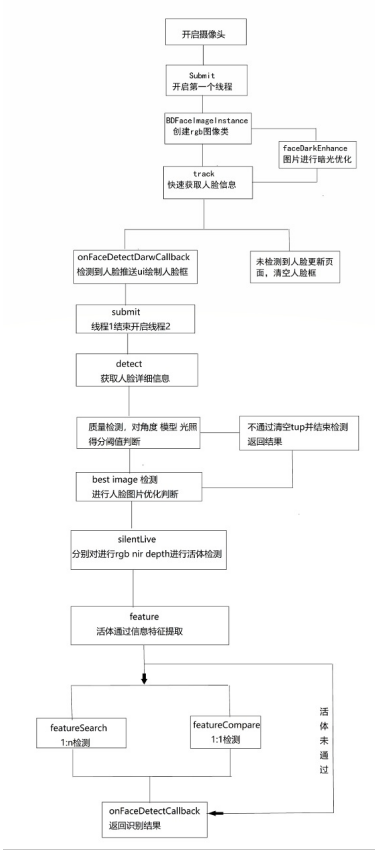
v2.0.2	2019.04.01	1、全新人脸检测模型，检测追踪更流畅； 2、全新证件照模型：体积更小，速度更快 ； 3、部分接口细节优化
v2.0.1	2019.03.14	1、接口设计优化； 2、增加几款结构光镜头支持； 3、已知bug修复
v2.0.0	2019.01.10	1、优化生活照模型精度及速度 2、优化检测模型及策略 3、优化接口设计 4、优化活体检测速度
v1.1.0	2018.09.03	1、增加离线证件照特征抽取模型，可有效处理芯片照、证件照比对需求 2、增加离线人脸属性模型，支持性别、年龄等属性分析 3、增加多线程支持 4、增加离线激活支持，可导入授权文件无网激活 5、增加对奥比中光Astra Pro深度图镜头模组支持 6、增加对华捷艾米深度镜头模组支持 7、其他细节优化及已知bug修复
v1.0.1	2018.08.03	1、修复设备指纹发生变化bug 2、替换近红外活体模型，优化效果 3、修复批量注册人脸到人脸库，失败问题 4、修复注册、图片人脸检测、视频返回图片抽取特征失败，经常出现检测不到人脸问题
v1.0.0	2018.06.29	初版，包括离线人脸采集、离线活体检测、离线对比识别、离线人脸库管理等功能



此技术文档主要是针对Android 8.0版本SDK进行说明，从接口文档和示例工程DEMO两个角度详尽地阐述了8.0版本的强大功能及使用方法。

1、SDK示例工程说明

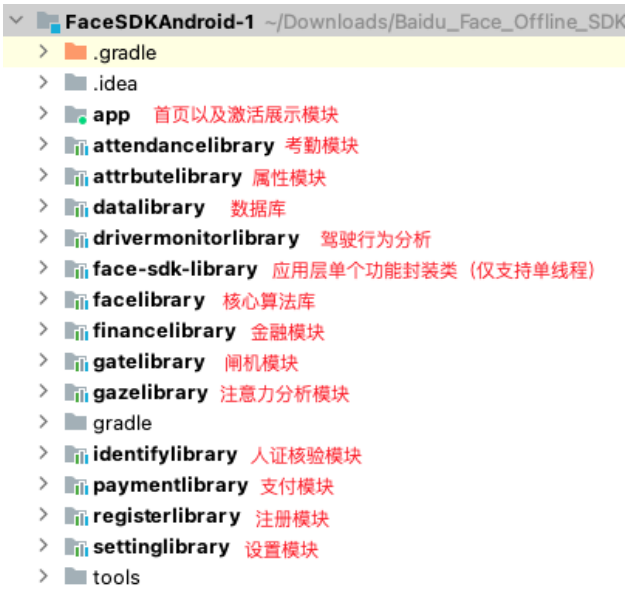
1.1 识别流程图



1.2 工程及接口说明

1.2.1 工程目录图

（1）attendancelibrary 考勤模块
（2）attrbutelibrary 属性模块
（3）drivermonitorlibrary 驾驶行为模块
（4）financelibrary 金融模块
（5）gatelibrary 闸机模块
（6）gazelibrary 注意力模块
（7）identifylibrary 认证核验模块
（8）paymentlibrary 支付模块
（9）registerlibrary 人脸注册和人脸管理模块
（人脸 1:N 识别功能请参考考勤，闸机，支付模块）
（人脸 1:1 识别功能请参考人证核验模块）



1.2.2 faceLibrary核心库

所有识别逻辑都要依赖于此库，且其他model逻辑都是基于此库中的接口进行的应用层封装

其中lib为算法库，assets中为检测识别模型包含：

1.align 对齐模型

2.attribute 属性模型

3.best_image 最优人脸模型

4.blur 质量模糊模型

5.detect 人脸检测模型

6.driver_monitor 驾驶行为模型

7.eyes_close 眼睛闭合模型

8.feature 识别模型

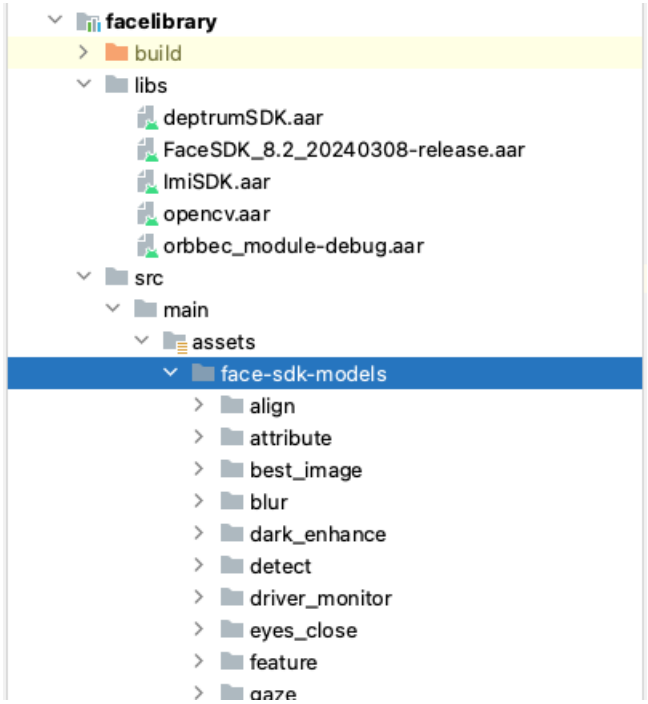
9.gaze 注意力模型

10. mouth_close 嘴巴闭合模型

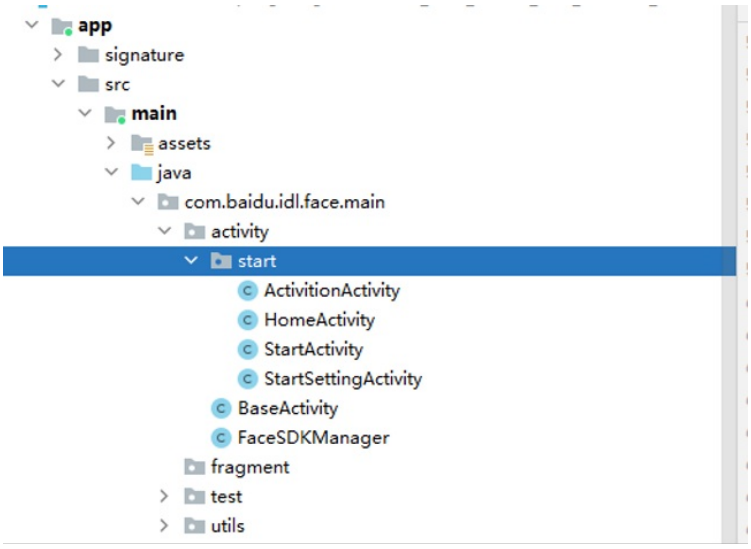
11. mouth_mask 口罩模型

12. occlusion 遮挡模型

13. silent_live 活检模型



1.2.3 app model



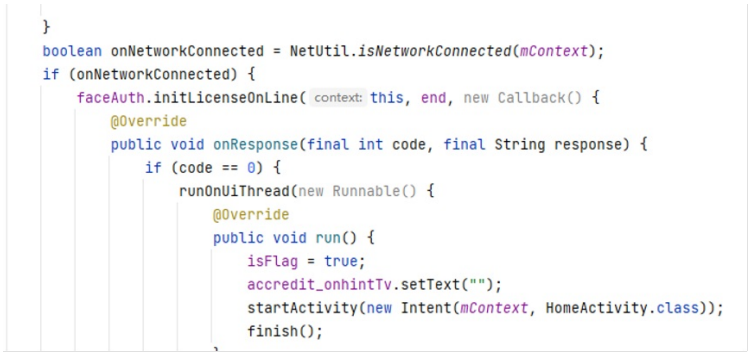
此model包含StartActivity启动页，sdk在线、离线、批量激活接口和激活方式和Home首页

SDK激活接口：

离线激活

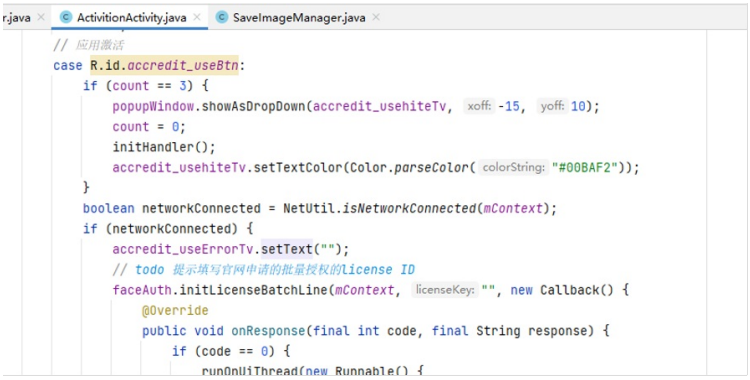
```
// 离线激活
case R.id.accredit_offBtn:
    faceAuth.initLicenseOffLine( context: this, new Callback() {
        @Override
        public void onResponse(final int code, final String response) {
            if (code == 0) {
                runOnUiThread(new Runnable() {
                    @Override
                    public void run() {
                        isFlag = true;
                        // 授权成功跳转功能入口页面
```

在线激活



```
}
boolean onNetworkConnected = NetUtil.isNetworkConnected(mContext);
if (onNetworkConnected) {
    faceAuth.initLicenseOnLine( context: this, end, new Callback() {
        @Override
        public void onResponse(final int code, final final String response) {
            if (code == 0) {
                runOnUiThread(new Runnable() {
                    @Override
                    public void run() {
                        isFlag = true;
                        accredit_onhintTv.setText("");
                        startActivity(new Intent(mContext, HomeActivity.class));
                        finish();
```

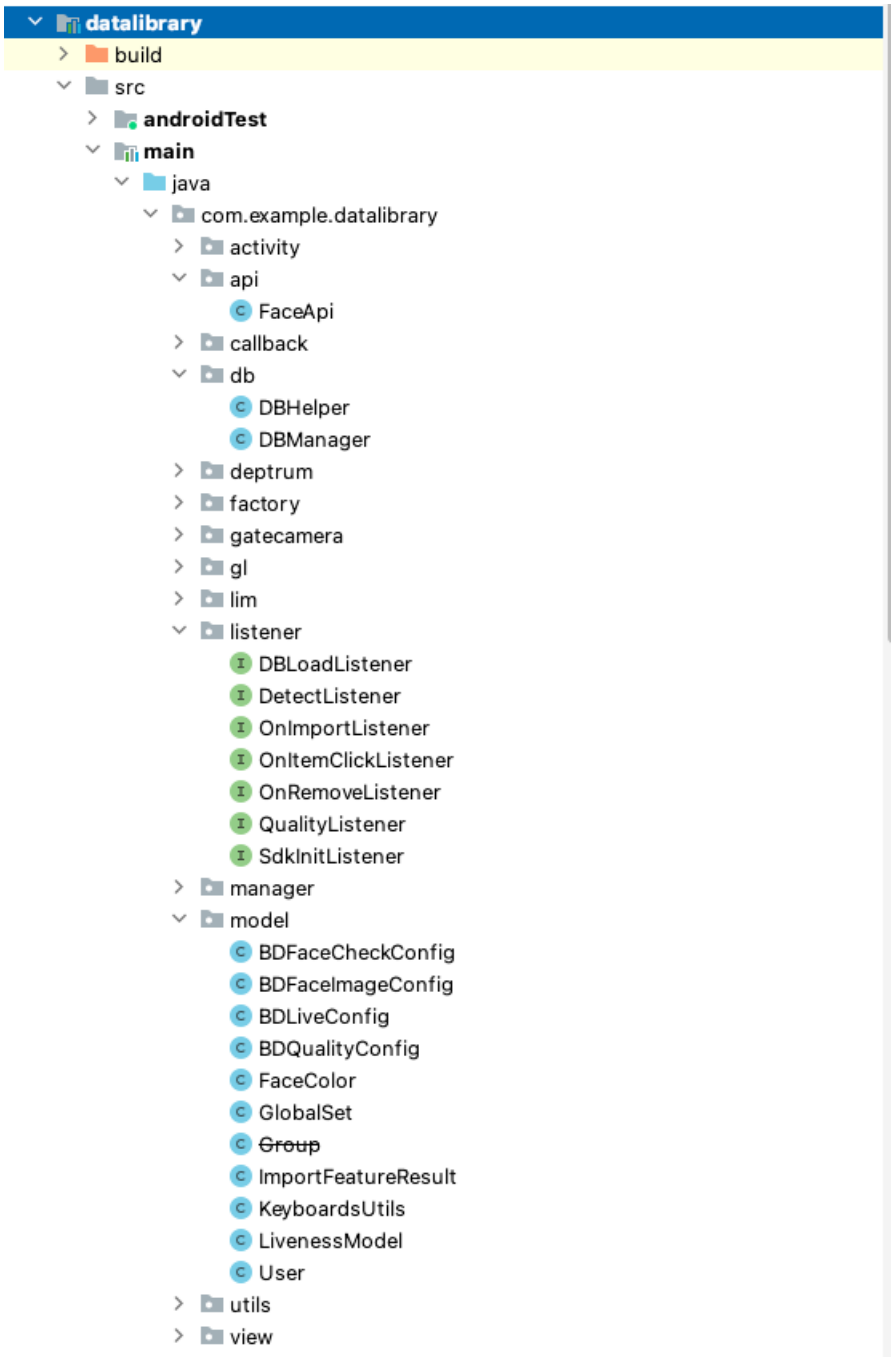
批量激活



```
r.java  ActivitionActivity.java  SaveImageManager.java
// 应用激活
case R.id.accredit_useBtn:
    if (count == 3) {
        popupWindow.showAsDropDown(accredit_usehiteTv, xoff: -15, yoff: 10);
        count = 0;
        initHandler();
        accredit_usehiteTv.setTextColor(Color.parseColor( colorString: "#00BAF2"));
    }
    boolean networkConnected = NetUtil.isNetworkConnected(mContext);
    if (networkConnected) {
        accredit_useErrorTv.setText("");
        // todo 提示填写官网申请的批量授权的license ID
        faceAuth.initLicenseBatchLine(mContext, licenseKey: "", new Callback() {
            @Override
            public void onResponse(final int code, final String response) {
                if (code == 0) {
                    runOnUiThread(new Runnable() {
```

1.2.4 dataLibrary 数据库 mode

datalibrary 模块为数据库管理模快，使用的是原生sqLite数据库（注：demo案例除百度识别sdk与3D结构光摄像头驱动外其他均为原生代码包括摄像头开启）



1.2.5 face-sdk-library high-level api 算法接口封装

face-sdk-library 内部集成了识别流程，其他模块继承 faceLibrary 只需要调取一个方法即可完成整体识别流程，目前只集成了活体检测，人脸特征提取，人脸比对，人脸属性分析，驾驶员行为分析 6 种接口方式（且目前仅支持单线程）

接口类型	接口需求	接口参数
人脸活体检测接口	将**人脸检测与对齐、质量检测**和**多模态活体检测**功能封装到一个接口内，用户可以通过调用该接口完成图像的活体判断。活体判断的结果为多模态联合判断，即：各种模态结果全部为真则结果为真人，任何一种模态判断为假则结果为假。	人脸检测：『最小检测人脸』 质量检测：『是否开启质量检测』『质量检测阈值』 活体检测：『活体检测模态』『活体检测阈值』
人脸特征抽取接口	将**人脸检测与对齐、质量检测、多模态活体检测**和**人脸特征抽取**功能封装到一个接口内，用户可以通过调用该接口完成图像中人脸特征抽取。可以通过制定的特征值抽取模型完成人脸特征的抽取。	人脸检测：『最小检测人脸』 质量检测：『是否开启质量检测』『质量检测阈值』 活体检测：『活体检测模态』『活体检测阈值』 特征抽取：『特征抽取模型』
1:1人脸比对接口	基于**『人脸特征值抽取接口』**完成人脸的1:1比对，用户通过调用该接口可以完成两个图像人脸相似度的比较，返回结果为两个图像的人脸相似度。	人脸检测：『最小检测人脸』 质量检测：『是否开启质量检测』『质量检测阈值』 活体检测：『活体检测模态』『活体检测阈值』 特征抽取：『特征抽取模型』 特征比对：『特征比对阈值』
人脸属性分析接口	将**人脸检测与对齐、人脸属性分析**功能封装到一个接口内，用户可以通过调用该接口得到图像中人脸的年龄、性别、是否佩戴眼镜、是否佩戴口罩等属性信息	人脸检测：『最小检测人脸』
驾驶员行为分析接口	将**人脸检测与对齐、驾驶员行为分析**功能封装到一个接口内，用户可以通过调用该接口得到图象中人脸的驾驶员相关属性：抽烟、进食、饮水、使用手机、注意力、安全带等	人脸检测：『最小检测人脸』

1.2.6 人脸识别Android SDK接入前置基础配置

1.2.6.1 官网sdk示例使用

如果可以直接使用我们官方文档中给出的demo示例，则进入网站进行下载https://console.bce.baidu.com/ai/#/ai/face/offline/index

集成以及sdk接口文档地址：https://ai.baidu.com/ai-doc/FACE/pk37c1mqu

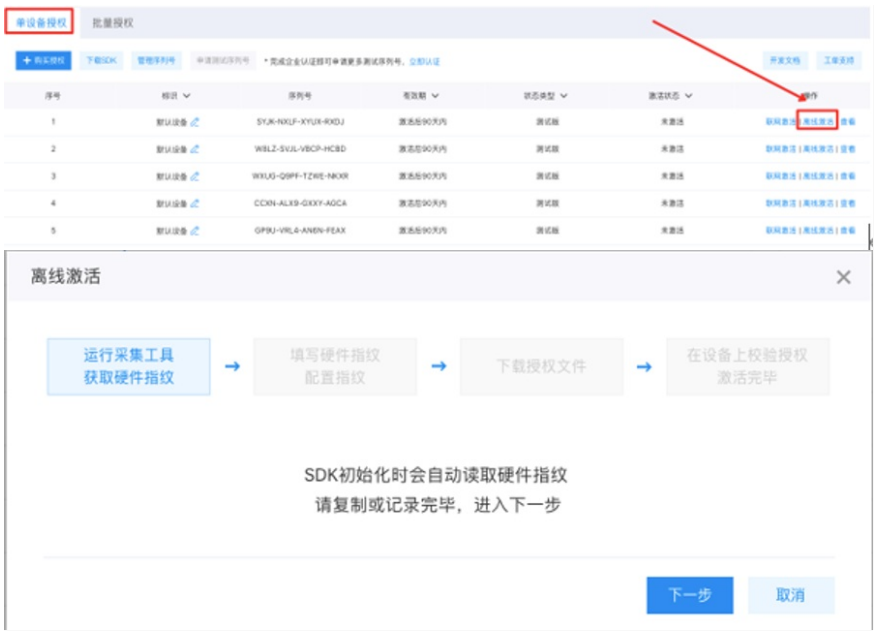
1.3 1:n 识别功能-简单集成方法使用说明：(以闸机模式为例)

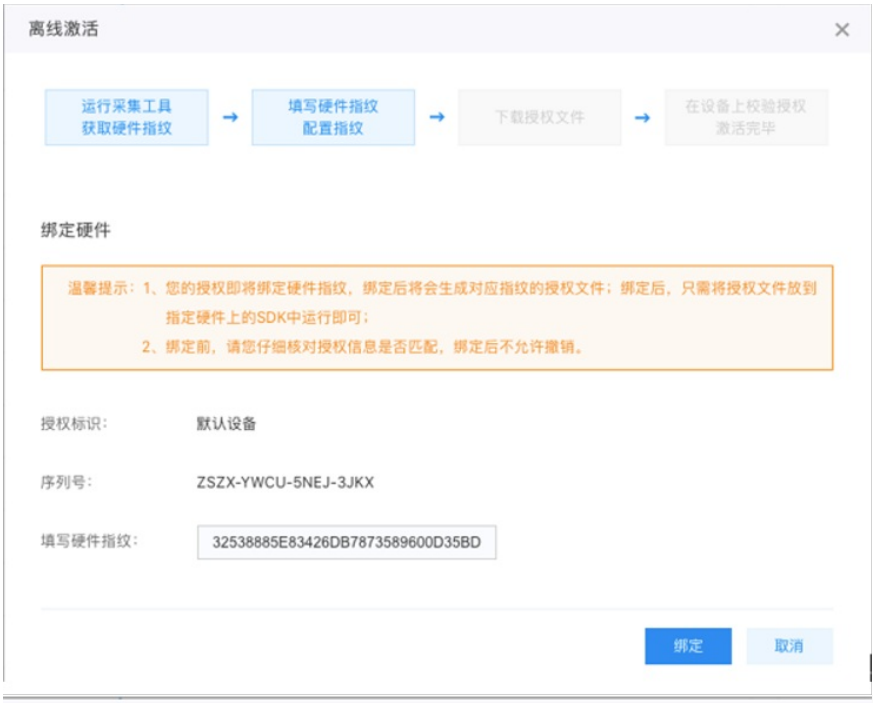
激活说明

在线激活


单设备授权
批量授权
离线激活
运行采集工具
获取硬件指纹
填写硬件指纹
配置指纹
下载授权文件
在设备上校验授权
激活完毕
SDK初始化时会自动读取硬件指纹
请复制或记录完毕，进入下一步
下一步
取消
联网激活
在SDK配置界面
中填写序列号
点击激活按钮
启动激活
SDK自动联网
激活完毕
联网激活成功
// 在线激活方式
// licenseID : 官网申请的licenseID
faceAuth.initLicenseOnLine(context: this, licenseID: "xxxx-xxxx-xxxx-xxxx", new Callback() {
@Override
public void onResponse(int code, String s) {
if(code == 0){
// 在线激活成功，跳转页面
}
}
});

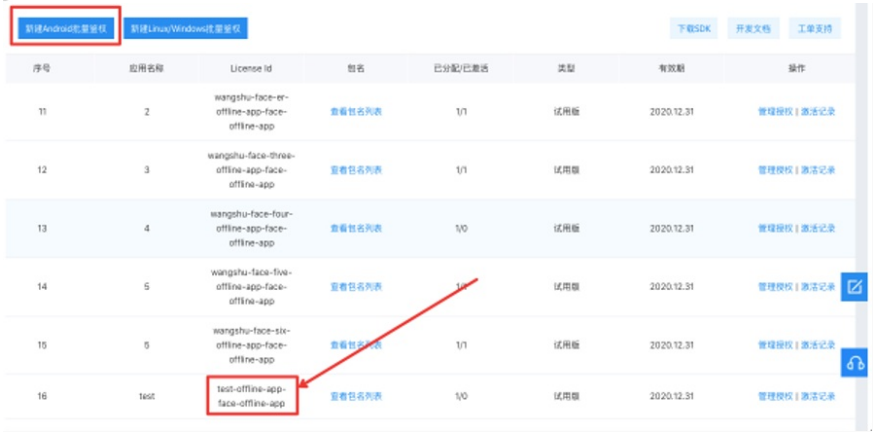
离线激活


单设备授权
批量授权
离线激活
运行采集工具
获取硬件指纹
填写硬件指纹
配置指纹
下载授权文件
在设备上校验授权
激活完毕
SDK初始化时会自动读取硬件指纹
请复制或记录完毕，进入下一步
下一步
取消



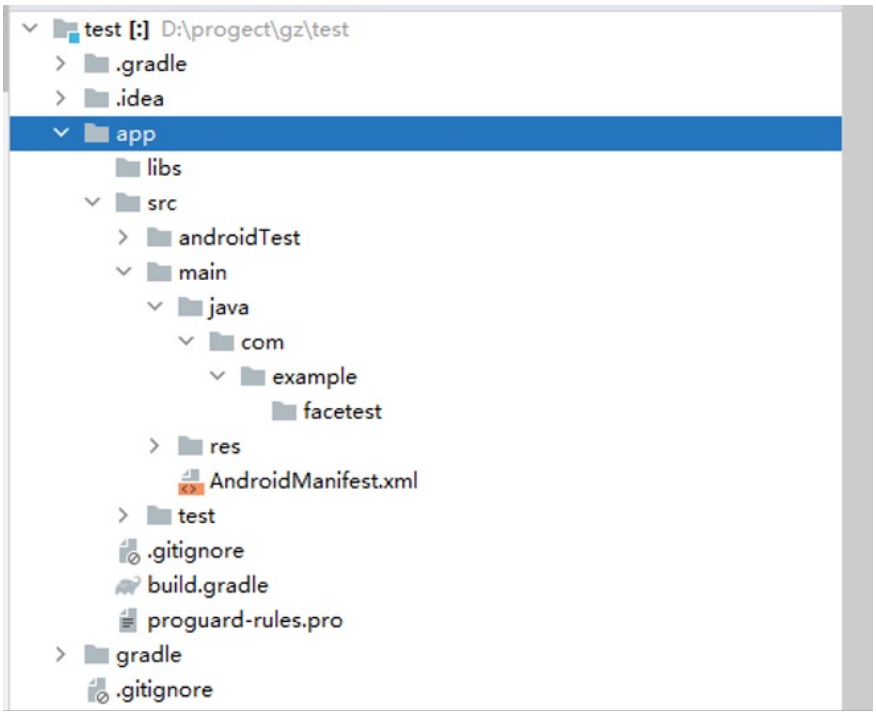

```
//  离线激活方式
faceAuth.initLicenseOffLine( context: this, new Callback() {
    @Override
    public void onResponse(int code, String s) {
        if(code == 0){
            // 离线激活成功，跳转页面
        }
    }
});
```

- #### 批量激活（属于在线激活）

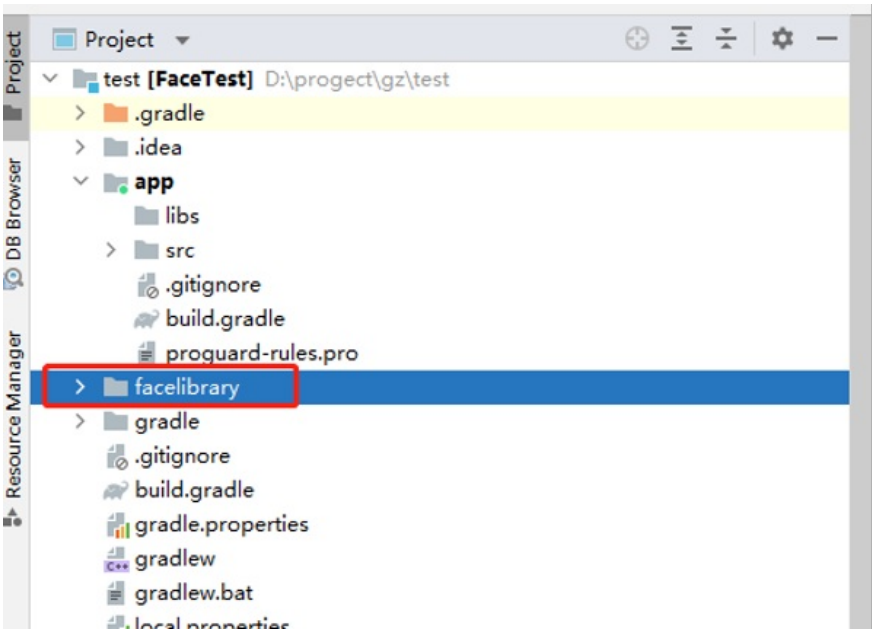


```
// 应用激活方式
// licenseID : 官网申请的licensekey
faceAuth.initLicenseBatchLine( context: this, licenseKey: "官网申请的自定义licenseID", new Callback() {
    @Override
    public void onResponse(int code, String s) {
        if(code == 0){
            // 应用激活成功，跳转页面
        }
    }
});
```

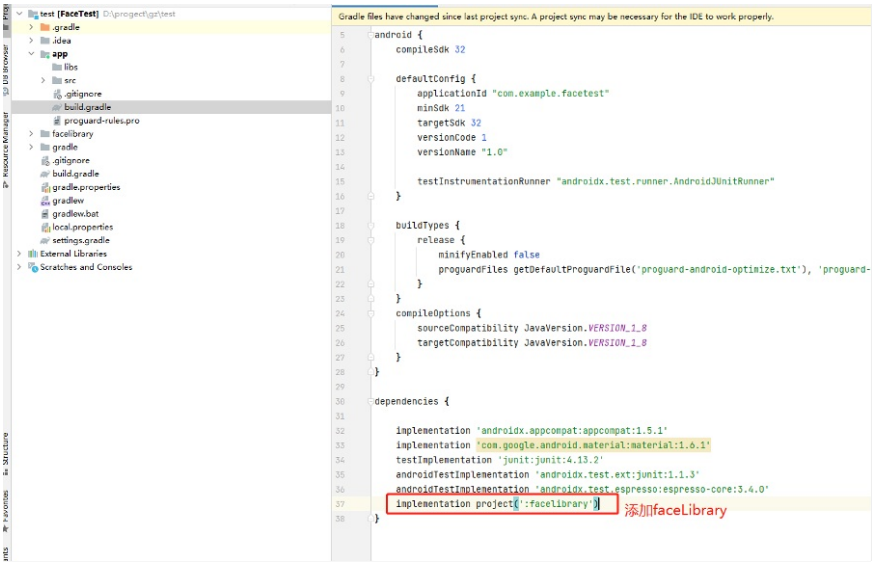
1.3.1 新建项目



1.3.2 导入faceLibrary库

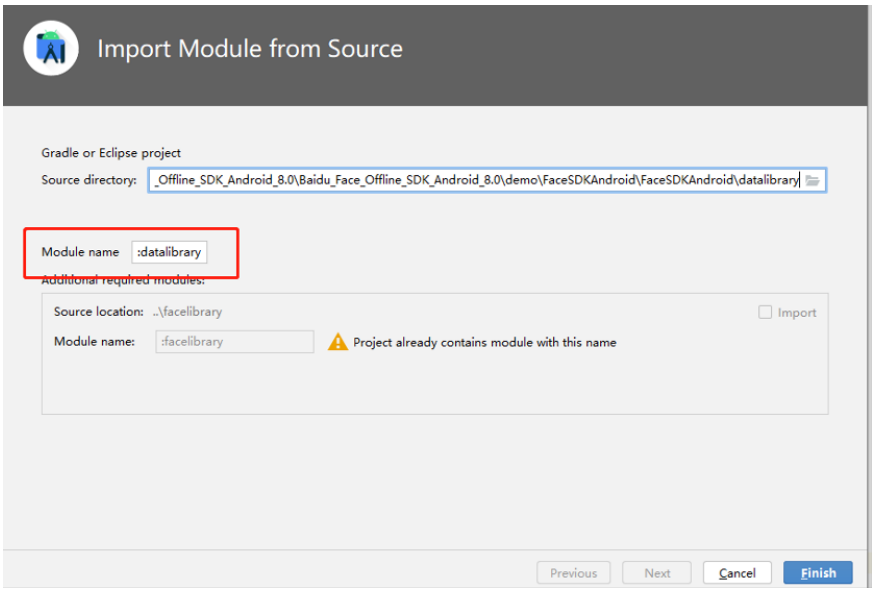


1.3.2.1 FaecLibrary放入app.build中



1.3.2.2 添加ndk

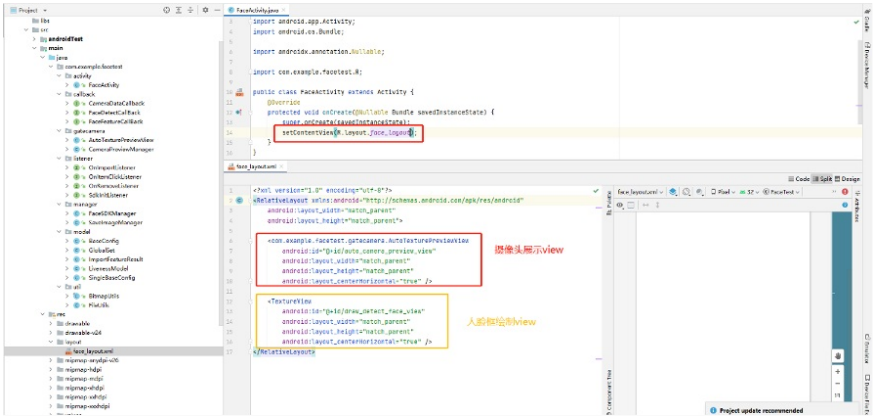
1.3.3 导入dataLibrary 数据库（同上）



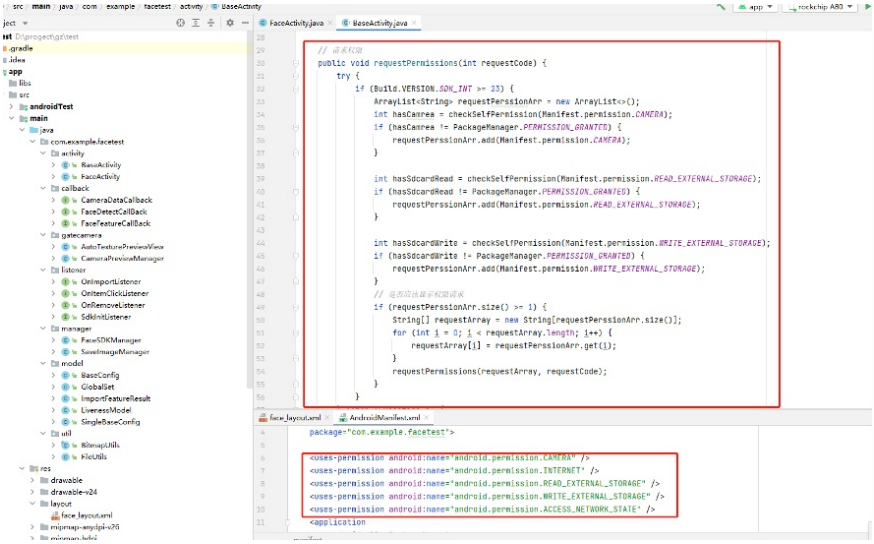
1.3.4 将闸机模式部分封装代码放入项目中

将model和utils中的文件全部复制到工程中，Activity根据实际的识别场景选择使用即可

1.3.5 创建activity以及layout



1.3.5.1 添加权限

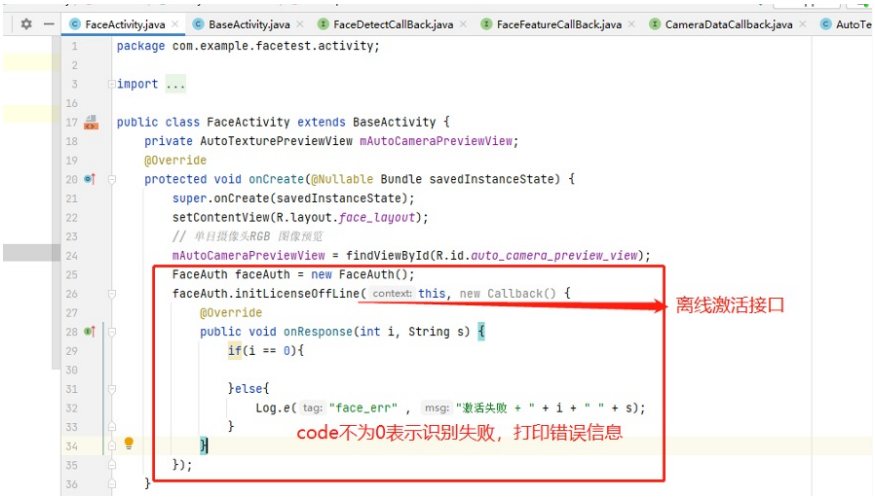


1.3.6 开启摄像头

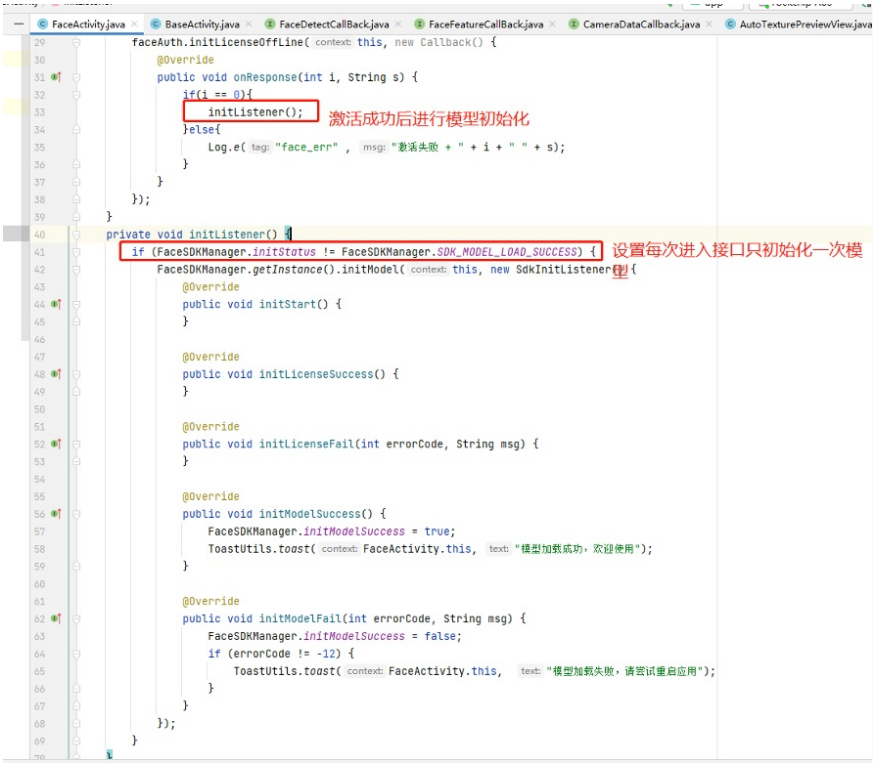
```java
import androidx.annotation.Nullable;

import com.example.facetest.R;
import com.example.facetest.callback.CameraDataCallback;
import com.example.facetest.gatecamera.AutoTexturePreviewView;
import com.example.facetest.gatecamera.CameraPreviewManager;
import com.example.facetest.model.SingleBaseConfig;

public class FaceActivity extends BaseActivity {
    private AutoTexturePreviewView mAutoCameraPreviewView;
    @Override
    protected void onCreate(@Nullable Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.face_layout);
        // 单目摄像头RGB 图像预览
        mAutoCameraPreviewView = findViewById(R.id.auto_camera_preview_view);
    }

    @Override
    protected void onResume() {
        super.onResume();


        CameraPreviewManager.getInstance().setCameraFacing(0);

        CameraPreviewManager.getInstance().startPreview( context: this, mAutoCameraPreviewView,
                SingleBaseConfig.getBaseConfig().getRgbAndNirWidth(),
                SingleBaseConfig.getBaseConfig().getRgbAndNirHeight(),
                new CameraDataCallback() {
                    @Override
                    public void onGetCameraData(byte[] data, Camera camera, int width, int height) {
                    }
                });
    }
}
```


运行展示效果（注：当前有可能出现摄像头展示角度出现问题）

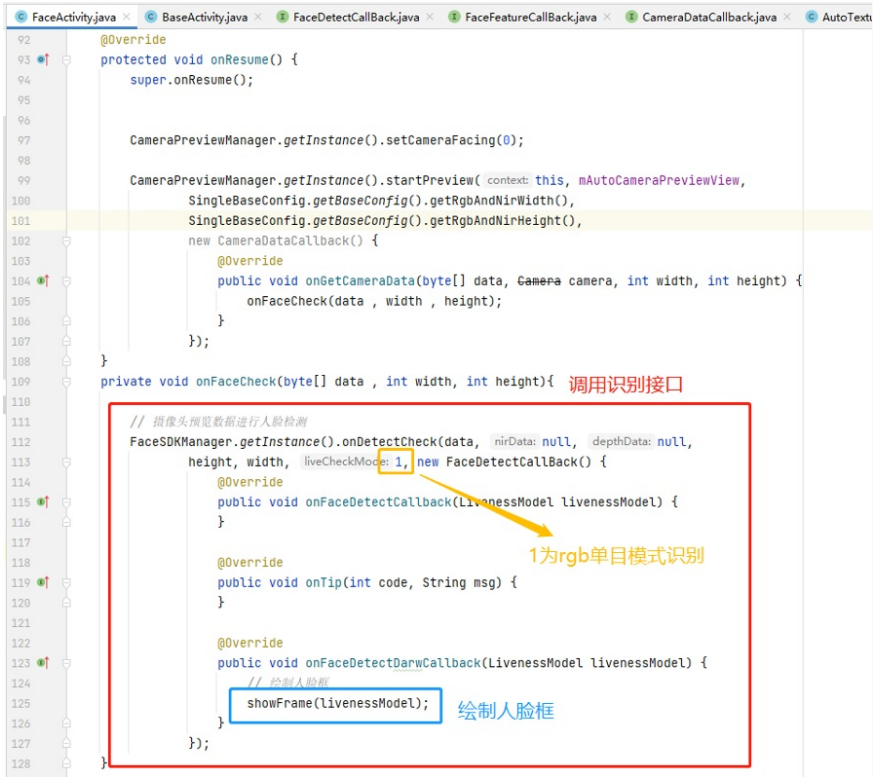
1.3.7 添加sdk激活接口（以离线激活为例）



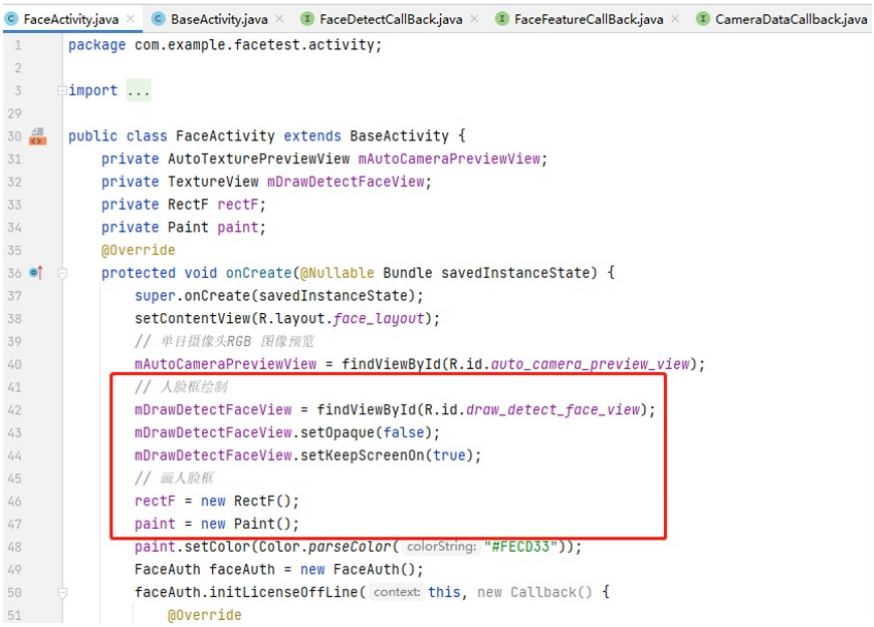
1.3.8 添加模型初始化

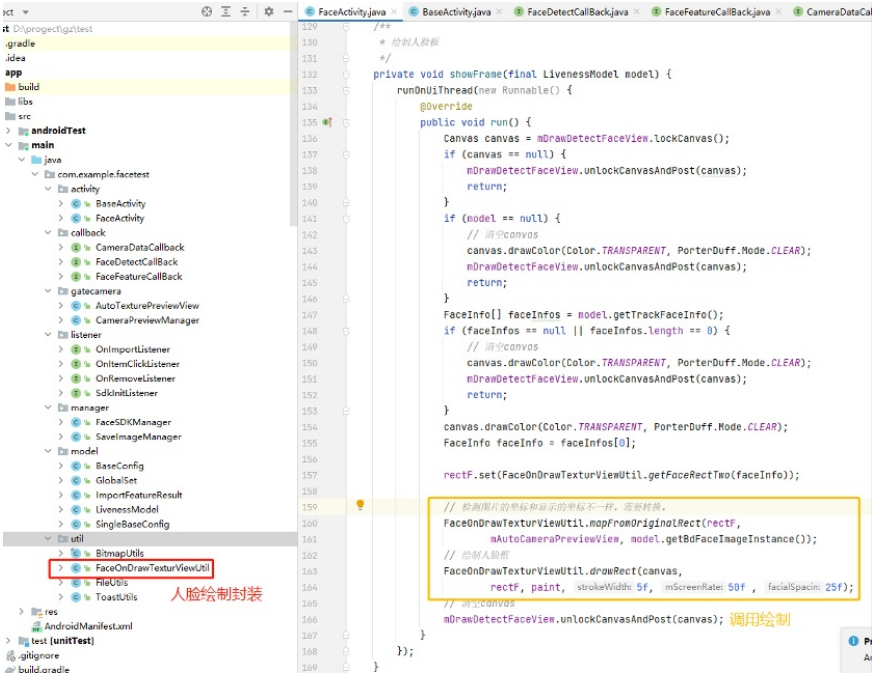


1.3.9 添加检测识别接口和人脸框绘制

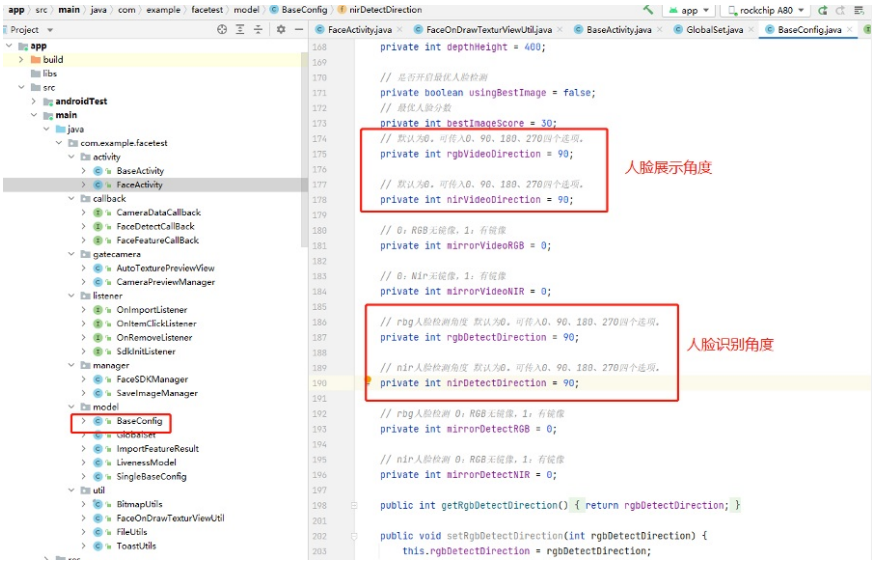


1.3.9.1 绘制人脸





1.3.9.2 设置人脸展示角度与识别角度(设备不同角度会有不同，建议先跑通官方demo确认两种角度具体数据)



效果展示

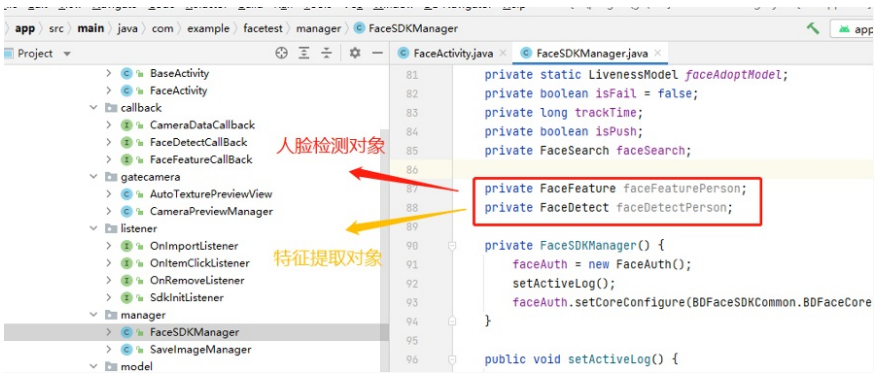
（注：部分设备人脸框会出现镜像，将人脸框展示view旋转180°）

1.3.10 注册人脸（以图片注册且导入+识别双线程举例）

1.3.10.1 添加初始化数据库

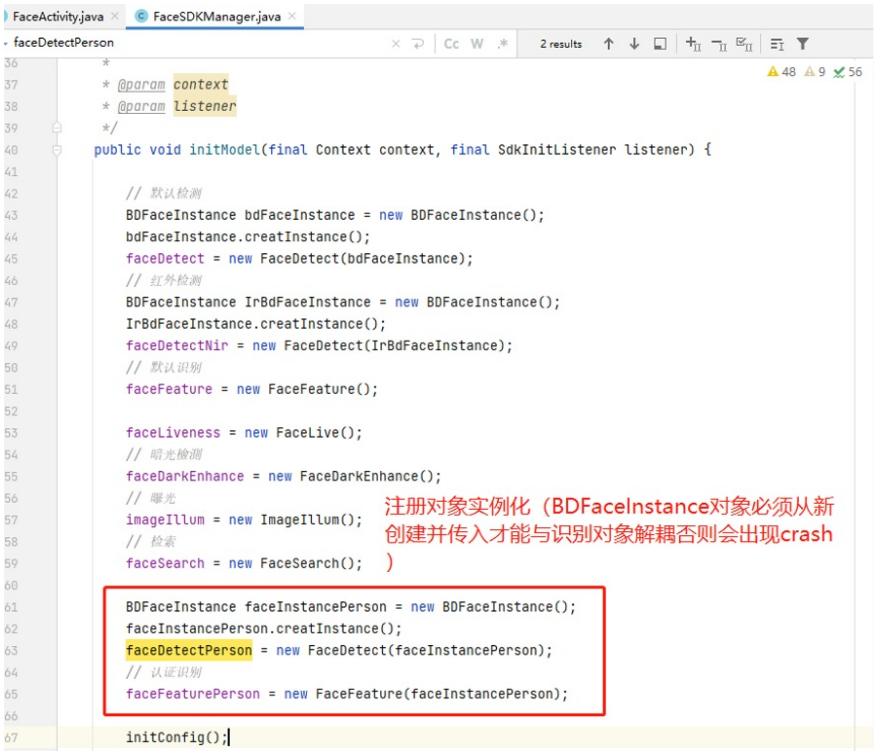
```java
package com.example.facetest.activity;

import ...

public class FaceActivity extends BaseActivity {
    private AutoTexturePreviewView mAutoCameraPreviewView;
    private TextureView mDrawDetectFaceView;
    private RectF rectF;
    private Paint paint;
    @Override
    protected void onCreate(@Nullable Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.face_layout);
        FaceSDKManager.getInstance().initDataBases( context: this);
        // 单目摄像头RGB 图像预览
        mAutoCameraPreviewView = findViewById(R.id.auto_camera_preview_view);
        // 人脸框绘制
        mDrawDetectFaceView = findViewById(R.id.draw_detect_face_view);
        mDrawDetectFaceView.setOpaque(false);
        mDrawDetectFaceView.setKeepScreenOn(true);
```

1.3.10.2 添加注册模型以及检测方法

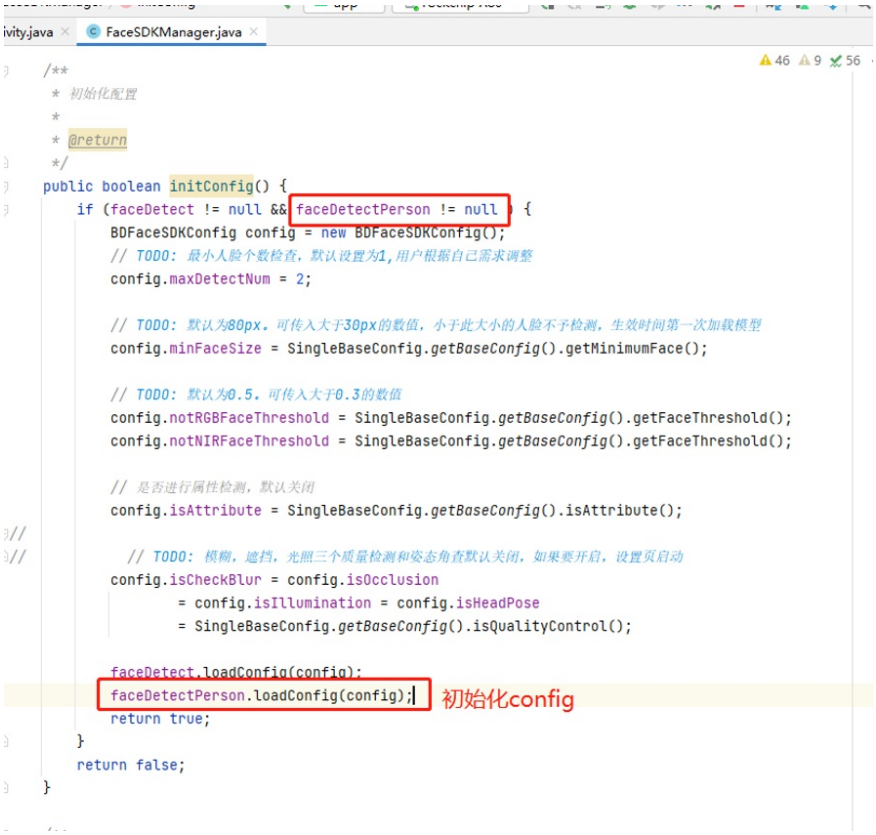


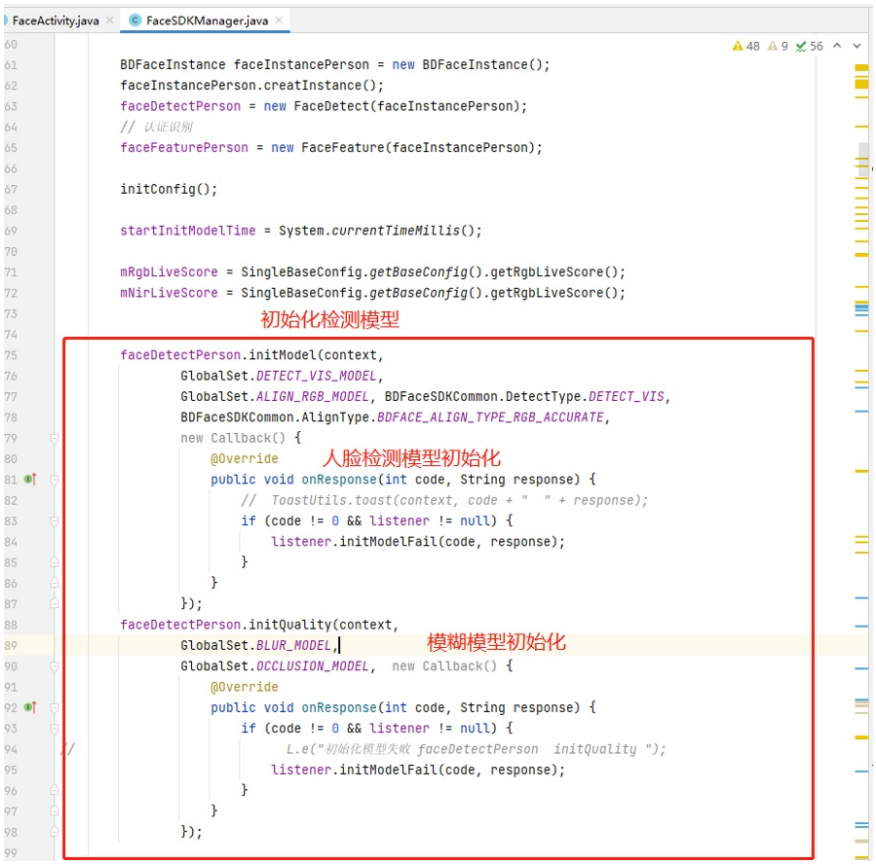

```java
private static LivenessModel faceAdoptModel;
private boolean isFail = false;
private long trackTime;
private boolean isPush;
private FaceSearch faceSearch;

private FaceFeature faceFeaturePerson;
private FaceDetect faceDetectPerson;

private FaceSDKManager() {
    faceAuth = new FaceAuth();
    setActiveLog();
    faceAuth.setCoreConfigure(BDFaceSDKCommon.BDFaceCore
}

public void setActiveLog() {
```

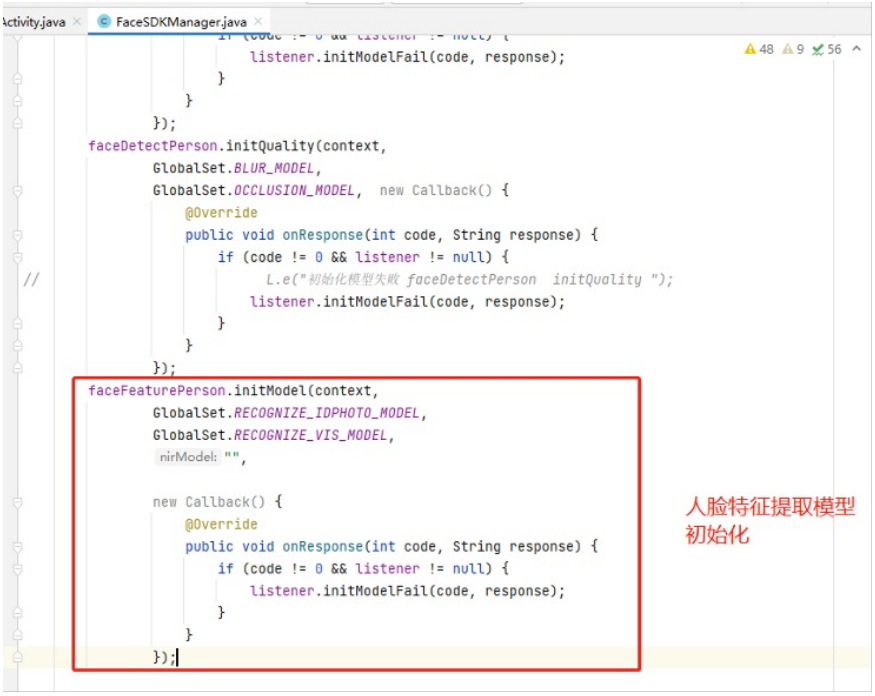
1.3.10.3 添加注册模型初始化




```java
/**
 * @param context
 * @param listener
 */
public void initModel(final Context context, final SdkInitListener listener) {

    // 默认检测
    BDFaceInstance bdFaceInstance = new BDFaceInstance();
    bdFaceInstance.creatInstance();
    faceDetect = new FaceDetect(bdFaceInstance);
    // 红外检测
    BDFaceInstance IrBdFaceInstance = new BDFaceInstance();
    IrBdFaceInstance.creatInstance();
    faceDetectNir = new FaceDetect(IrBdFaceInstance);
    // 默认识别
    faceFeature = new FaceFeature();

    faceLiveness = new FaceLive();
    // 暗光检测
    faceDarkEnhance = new FaceDarkEnhance();
    // 曝光
    imageIllum = new ImageIllum();
    // 检索
    faceSearch = new FaceSearch();

    BDFaceInstance faceInstancePerson = new BDFaceInstance();
    faceInstancePerson.creatInstance();
    faceDetectPerson = new FaceDetect(faceInstancePerson);
    // 认证识别
    faceFeaturePerson = new FaceFeature(faceInstancePerson);

    initConfig();
```


ivity.java
FaceSDKManager.java
46 9 56
/**
* 初始化配置
*
* @return
*/
public boolean initConfig() {
if (faceDetect != null && faceDetectPerson != null) {
BDFaceSDKConfig config = new BDFaceSDKConfig();
// TODO: 最小人脸个数检查，默认设置为1,用户根据自己需求调整
config.maxDetectNum = 2;
// TODO: 默认为80px，可传入大于30px的数值，小于此大小的人脸不予检测，生效时间第一次加载模型
config.minFaceSize = SingleBaseConfig.getBaseConfig().getMinimumFace();
// TODO: 默认为0.5，可传入大于0.3的数值
config.notRGBFaceThreshold = SingleBaseConfig.getBaseConfig().getFaceThreshold();
config.notNIRFaceThreshold = SingleBaseConfig.getBaseConfig().getFaceThreshold();
// 是否进行属性检测，默认关闭
config.isAttribute = SingleBaseConfig.getBaseConfig().isAttribute();
// TODO: 模糊，遮挡，光照三个质量检测和姿态角查默认关闭，如果要开启，设置页启动
config.isCheckBlur = config.isOcclusion
= config.isIllumination = config.isHeadPose
= SingleBaseConfig.getBaseConfig().isQualityControl();
faceDetect.loadConfig(config);
faceDetectPerson.loadConfig(config);
初始化config
return true;
}
return false;
}

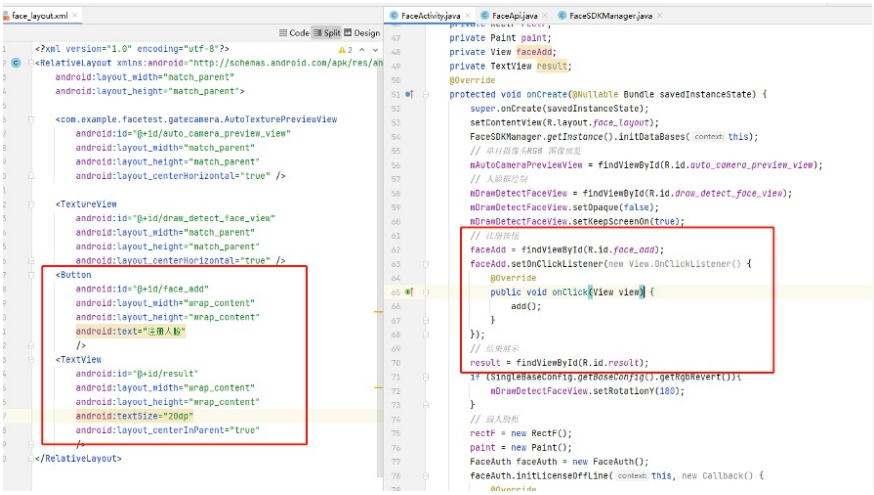

FaceActivity.java
FaceSDKManager.java
48 9 56
BDFaceInstance faceInstancePerson = new BDFaceInstance();
faceInstancePerson.creatInstance();
faceDetectPerson = new FaceDetect(faceInstancePerson);
// 认证识别
faceFeaturePerson = new FaceFeature(faceInstancePerson);
initConfig();
startInitModelTime = System.currentTimeMillis();
mRgbLiveScore = SingleBaseConfig.getBaseConfig().getRgbLiveScore();
mNirLiveScore = SingleBaseConfig.getBaseConfig().getRgbLiveScore();
初始化检测模型
faceDetectPerson.initModel(context,
GlobalSet.DETECT_VIS_MODEL,
GlobalSet.ALIGN_RGB_MODEL, BDFaceSDKCommon.DetectType.DETECT_VIS,
BDFaceSDKCommon.AlignType.BDFACE_ALIGN_TYPE_RGB_ACCURATE,
new Callback() {
@Override
人脸检测模型初始化
public void onResponse(int code, String response) {
// ToastUtils.toast(context, code + " " + response);
if (code != 0 && listener != null) {
listener.initModelFail(code, response);
}
}
});
faceDetectPerson.initQuality(context,
GlobalSet.BLUR_MODEL,
模糊模型初始化
GlobalSet.OCCLUSION_MODEL, new Callback() {
@Override
public void onResponse(int code, String response) {
if (code != 0 && listener != null) {
L.e("初始化模型失败 faceDetectPerson initQuality ");
listener.initModelFail(code, response);
}
}
});



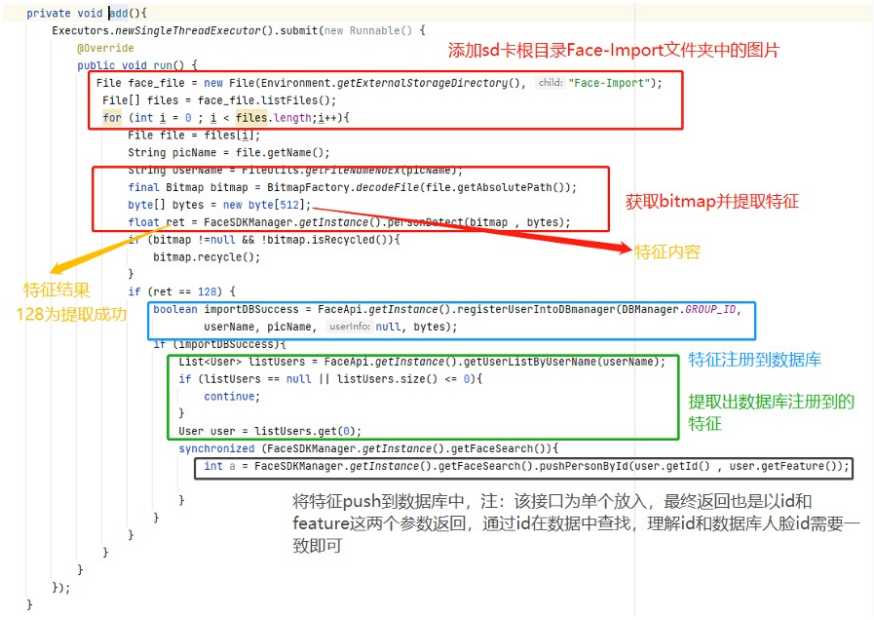
1.3.10.4 注册方法添加

1.3.10.5 添加注册逻辑

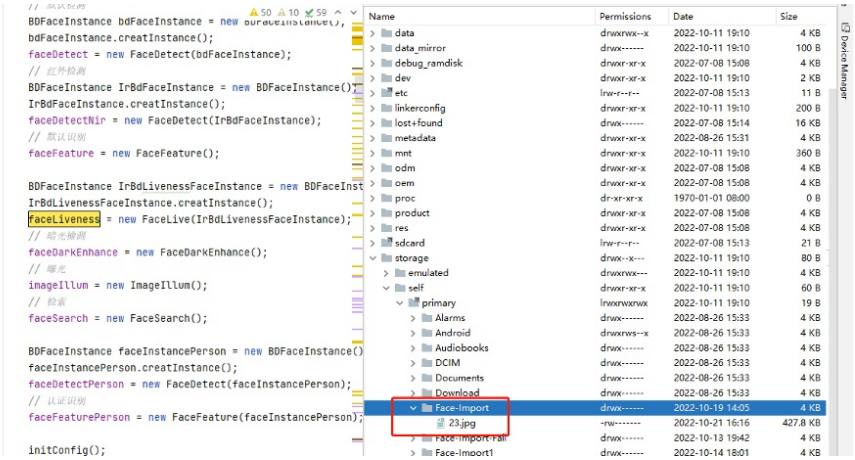
1.3.10.5.1 添加结果展示和注册按钮



1.3.10.5.2 添加注册方法

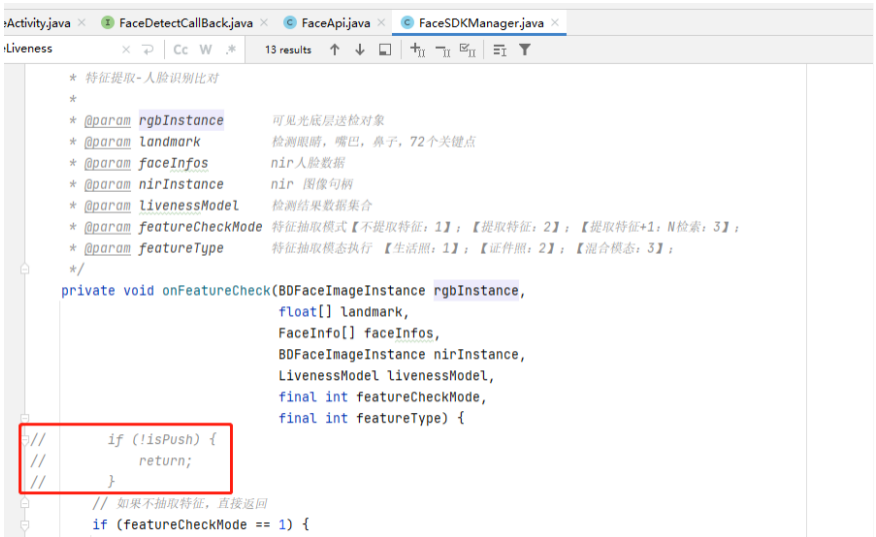


本地sd卡Face-Import文件夹下放入人脸图片

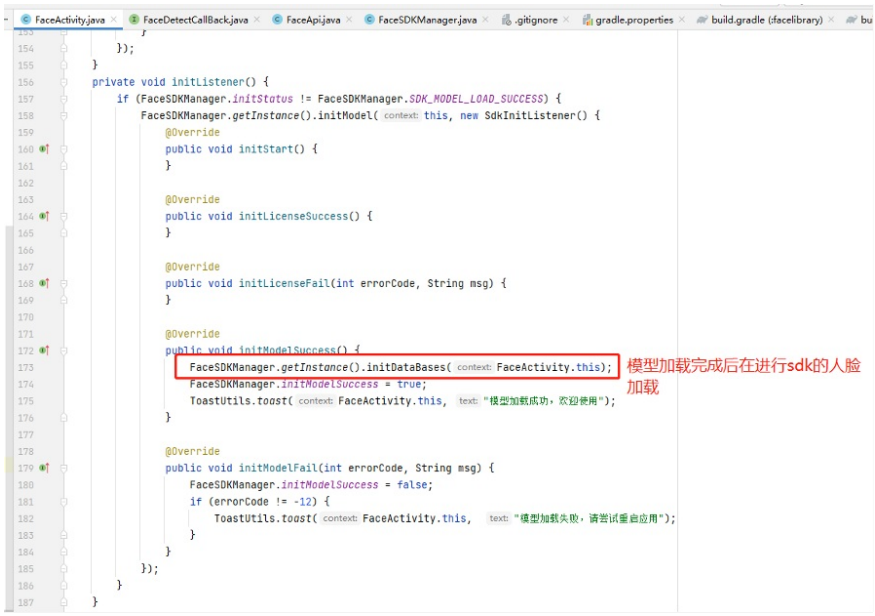


进入apk点击注册人脸查看识别效果

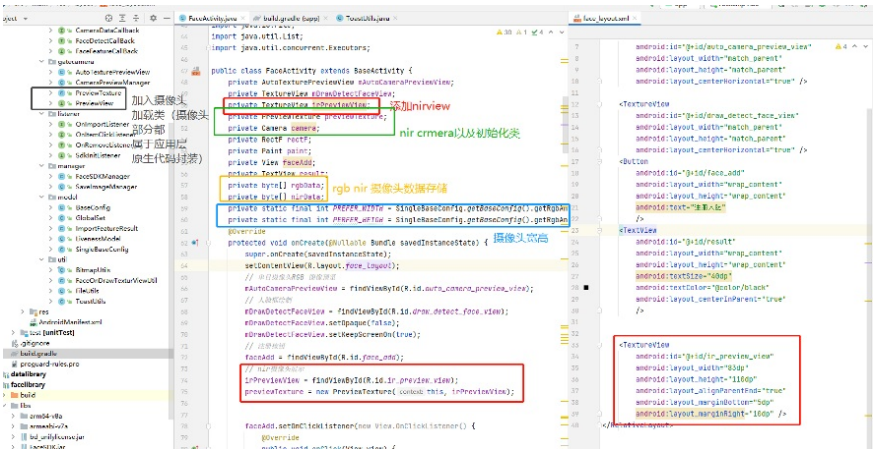
部分设备此时可能会出现一直识别不过的现象，可以先将这部分注释



这时会发现从新进入程序后仍然不能识别到人脸，是因为数据库没有正常加载导致，加入数据库加载代码



1.3.11 nir活体摄像头添加（仅限于含有rgb+nir双目摄像头设备）




```
@Override
protected void onResume() {
    super.onResume();


    CameraPreviewManager.getInstance().setCameraFacing(0);

    CameraPreviewManager.getInstance().startPreview( context: this, mAutoCameraPreviewView,
            PREFER_WIDTH, PERFER_HEIGH, new CameraDataCallback() {
                @Override
                public void onGetCameraData(byte[] data, Camera camera, int width, int height) {
                    dealRgb(data);
                }
            });
    // 开启nir摄像头
    camera = Camera.open(1);
    int cameraRotation = SingleBaseConfig.getBaseConfig().getNirVideoDirection();
    camera.setDisplayOrientation(cameraRotation);
    previewTexture.setCamera(camera, PREFER_WIDTH, PERFER_HEIGH);
    camera.setPreviewCallback(new Camera.PreviewCallback() {
        @Override
        public void onPreviewFrame(byte[] data, Camera camera) {
            dealIr(data);
        }
    });

}
private void dealRgb(byte[] rgbData){
    this.rgbData = rgbData;
    onFaceCheck();
}
private void dealIr(byte[] nirData){
    this.nirData = nirData;
}
```

开启nir摄像头

设置nir摄像头角度

调用识别

记录nir 和 rgb 数据

```
    onFaceCheck();
}
private void dealIr(byte[] nirData){
    this.nirData = nirData;
}
private void onFaceCheck(){

    if (FaceSDKManager.initStatus != FaceSDKManager.SDK_MODEL_LOAD_SUCCESS) {
        return;
    }
    if (rgbData == null || nirData == null){
        return;
    }
    // 摄像头预览数据进行人脸检测
    FaceSDKManager.getInstance().onDetectCheck(rgbData, nirData, depthData: null,
            PERFER_HEIGH, PREFER_WIDTH, liveCheckMode: 2, new FaceDetectCallBack() {
                @Override
                public void onFaceDetectCallback(LivenessModel livenessModel) {
                    checkResult(livenessModel);
                }

                @Override
                public void onTip(int code, String msg) {
                }

                @Override
                public void onFaceDetectDarwCallback(LivenessModel livenessModel) {
                    // 绘制人脸框
                    showFrame(livenessModel);
                }
            });

}
private void  checkResult(LivenessModel livenessModel){
    // 当未检测到人脸UI显示
```

nir数据传入

2为rgb+nir识别

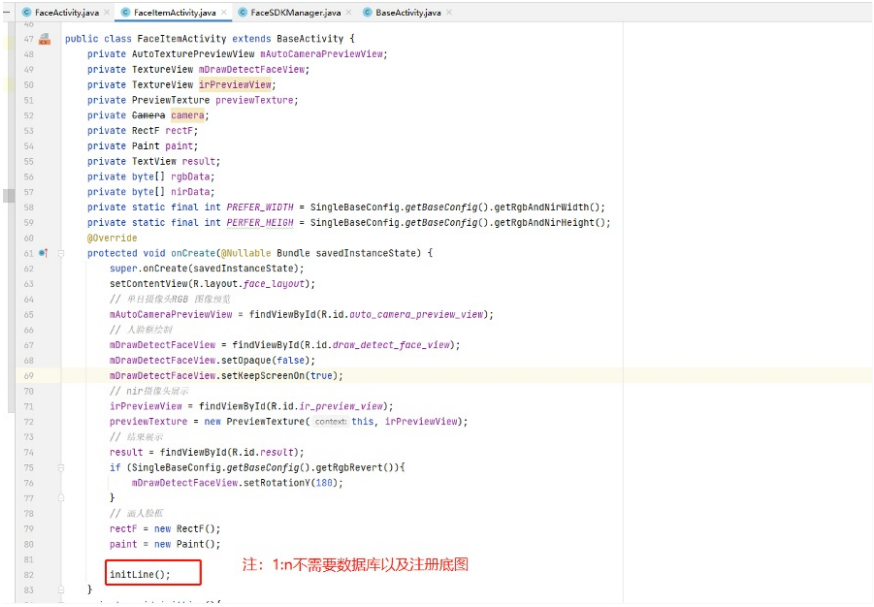
```
private void  checkResult(LivenessModel livenessModel){
    // 当未检测到人脸UI显示
    runOnUiThread(new Runnable() {
        @Override
        public void run() {
            if (livenessModel == null){
                return;
            }
            if (! livenessModel.isQualityCheck()){
                result.setText("请正视摄像头");
            }else if( livenessModel.getRgbLivenessScore() < SingleBaseConfig.getBaseConfig().getRgbLiveScore() ||
                    livenessModel.getIrLivenessScore() < SingleBaseConfig.getBaseConfig().getNirLiveScore()){

                result.setText("活体检测未通过 ，rgb活体得分 ： " + livenessModel.getRgbLivenessScore() +
                        " nir活体得分 : " + livenessModel.getIrLivenessScore());
            }else if (livenessModel.getUser() == null){

                result.setText("识别未通过");
            }else {
                result.setText("识别通过 人员姓名 ： " + livenessModel.getUser().getUserName());

            }
```

nir结果判断

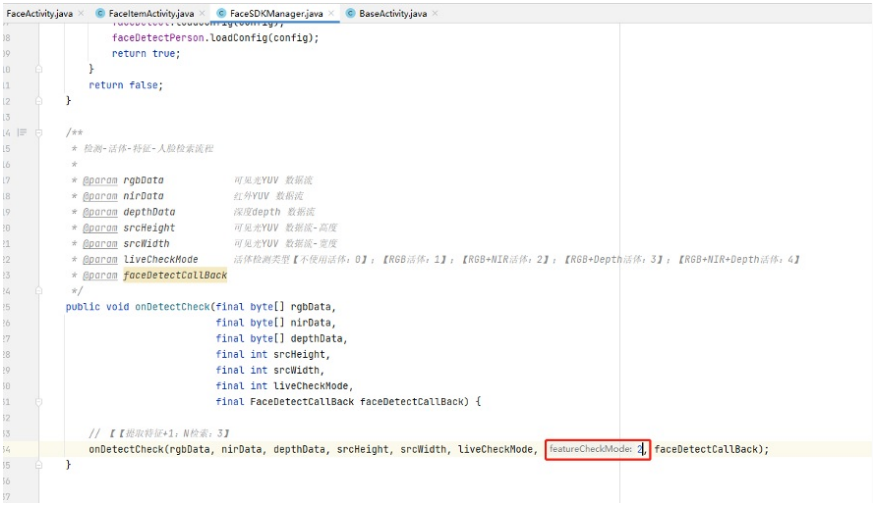
效果展示

1.4 1:1 识别功能-简单集成方法使用说明：（同上方1：n识别demo继续）

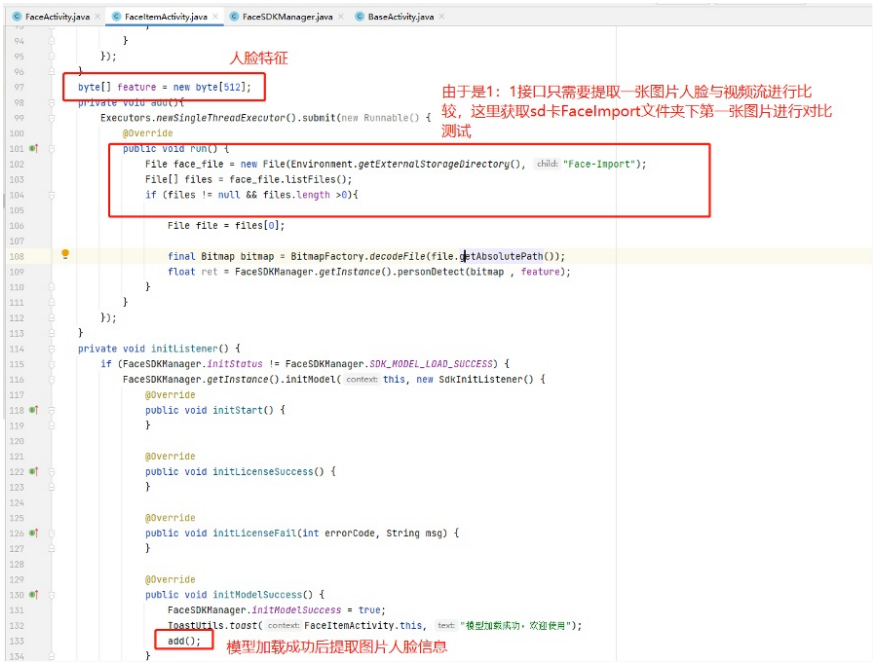
1.4.1 创建1：1 activity 功能与1：n同理（1：1不需要数据库）



1.4.2 调整FaceSDKManager类为只进行特征提取不进行1：n比较(且去掉3.10.5.2 部分的isPush判断)



1.4.3 提取图片人脸特征



1.4.4 添加1：1比较接口

```
                @Override
                public void onTip(int code, String msg) {
                }

                @Override
                public void onFaceDetectDarwCallback(LivenessModel livenessModel) {
                    // 绘制人脸框
                    showFrame(livenessModel);
                }
            });
    }
    private void  checkResult(LivenessModel livenessModel){
        // 当未检测到人脸UI显示
        runOnUiThread(new Runnable() {
            @Override
            public void run() {
                if (livenessModel == null){
                    return;
                }
                if (! livenessModel.isQualityCheck()){
                    result.setText("请正视摄像头");
                }else if( livenessModel.getRgbLivenessScore() < SingleBaseConfig.getBaseConfig().getRgbLiveScore() ||
                        livenessModel.getIrLivenessScore() < SingleBaseConfig.getBaseConfig().getNirLiveScore()){

                    result.setText("活体检测未通过 ，rgb活体得分 ： " + livenessModel.getRgbLivenessScore() +
                            " nir活体得分 ：" + livenessModel.getIrLivenessScore());
                }else {
                    float score = FaceSDKManager.getInstance().getFaceSearch().compare(BDFaceSDKCommon.FeatureType.BDFACE_FEATURE_TYPE_LIVE_PHOTO ,
                            livenessModel.getFeature() , feature , isPercent: true);
                    if (score >= SingleBaseConfig.getBaseConfig().getLiveThreshold()){

                        result.setText("识别通过");
                    }else {
                        result.setText("识别未通过");
                    }
                }
            }
        });
    }
```

1：1人脸特征比较接口

效果展示

1.5 驾驶行为模块使用说明

驾驶行为检测，只有单纯的检测功能和RGB活检，用于检测行为和注意力。不做识别功能（包括金融、属性、注意力模式均补进行识别功能）

```
private void checkData() {
    if (rgbData != null ) {
        FaceSDKManager.getInstance().onAttrDetectCheck(rgbData, irData, depthData: null, RGB_HEIGHT,
                RGB_WIDTH, SingleBaseConfig.getBaseConfig().getType() , new FaceDetectCallBack() {
                    5 usages
                    @Override
                    public void onFaceDetectCallback(final LivenessModel livenessModel) {
                        if (isRelease) {
                            showDetectImage(livenessModel);
                            showFaceMessage(livenessModel);
                        } else {
                            showFaceMessage(livenessModel);
                        }

                        // 检测结果输出
                        if (future3 != null && !future3.isDone()) {
                            return;
                        }

                        future3 = es3.submit((Runnable) () -> {
                                if (livenessModel == null) {
                                    return;
                                }
                                long gazeStartTime = System.currentTimeMillis();
                                bdFaceGazeInfo = livenessModel.getBdFaceGazeInfo();
                                long gazeEndTime = System.currentTimeMillis() - gazeStartTime;

                                if (irData != null) {
                                    long driverStartTime = System.currentTimeMillis();
                                    driverInfo = livenessModel.getDriverInfo();
                                    driverEndTime = System.currentTimeMillis() - driverStartTime;
                                } else {
                                    driverEndTime = 0;
                                }
                                showDriverMessage(bdFaceGazeInfo, driverInfo, gazeEndTime, driverEndTime);

                        });
```

人脸信息获取+活体检测

活体信息展示

驾驶行为信息获取

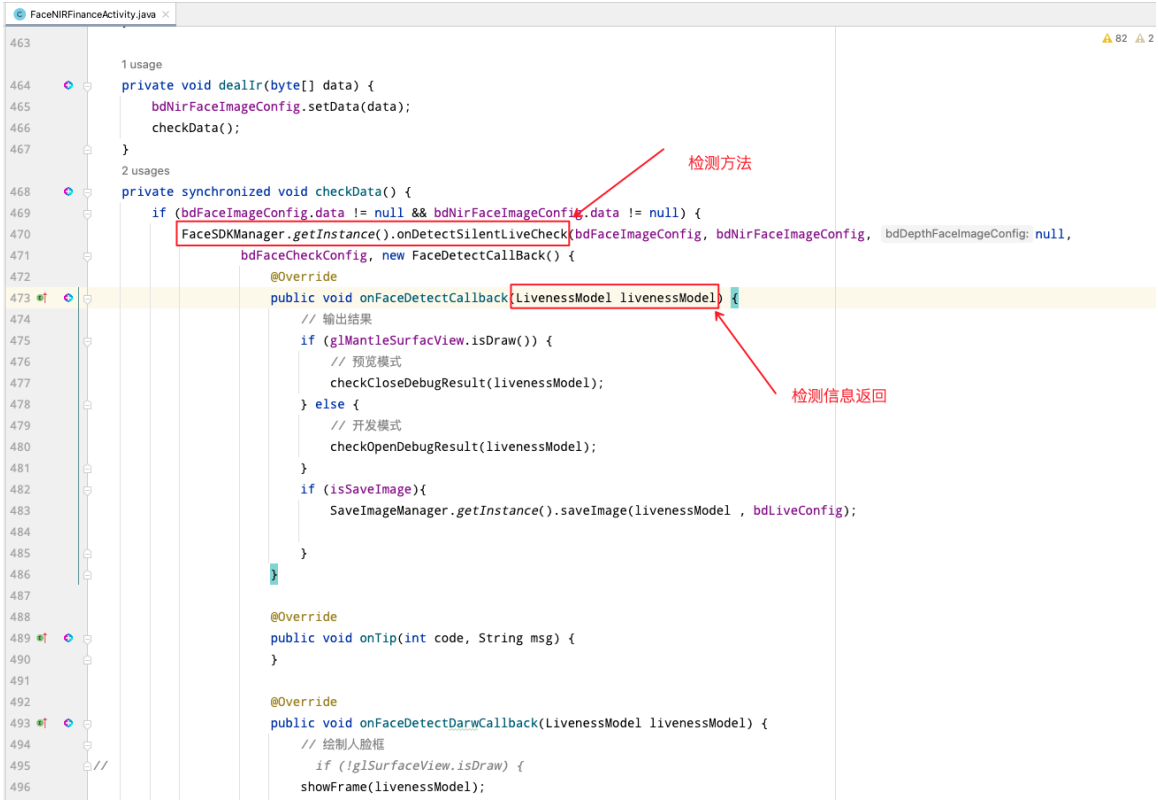
aceRGBGateActivity.java | FaceRGBPersonActivity.java | datalibrary/.../FaceSDKManager.java | DriverMonitorActivityDrivermonitor.java | BDFaceDriverMonitorInfo.class | drivermonitor/.../FaceSDK

```
} else {
    dmReleaseMsg.setText("");
}
if (driverInfo != null && driverInfo.getBdFaceDriverMonitorInfo() != null) {

    if (driverInfo.getBdFaceDriverMonitorInfo().isCalling()) {
        drivierMsg.append("打电话,");
    }
    if (driverInfo.getBdFaceDriverMonitorInfo().isDrinking()) {
        drivierMsg.append("喝水,");
    }
    if (driverInfo.getBdFaceDriverMonitorInfo().isEating()) {
        drivierMsg.append("吃东西,");
    }
    if (driverInfo.getBdFaceDriverMonitorInfo().isSmoking()) {
        drivierMsg.append("抽烟,");
    }

    String detailsMsg = drivierMsg.toString();
    if (detailsMsg.length() != 0) {
        if (detailsMsg.endsWith(",")) {
            detailsMsg = detailsMsg.substring(0, detailsMsg.length() - 1);
        }
        if (isRelease) {
            dmRelLinerMsg.setVisibility(View.VISIBLE);
        } else {
            driverDebugMsg.setVisibility(View.VISIBLE);
        }
        dmPreTx.setText(detailsMsg);
        dmReleaseMsg.setText(detailsMsg);
    }

    dmDriverCallScore.setText("打电话分数: " + driverInfo.getBdFaceDriverMonitorInfo().calling);
    dmDriverDrinkScore.setText("喝水分数: " + driverInfo.getBdFaceDriverMonitorInfo().drinking);
    dmDriverEatScore.setText("吃东西分数: " + driverInfo.getBdFaceDriverMonitorInfo().eating);
    dmDriverSmokeScore.setText("抽烟分数: " + driverInfo.getBdFaceDriverMonitorInfo().smoking);
    dmDriverNormalScore.setText("正常分数: " + driverInfo.getBdFaceDriverMonitorInfo().normal);
} else {
    dmPreTx.setText("");
```

驾驶行为判断

驾驶行为得分展示

1.6 金融活检模块使用说明




```
1 usage
private void dealIr(byte[] data) {
    bdNirFaceImageConfig.setData(data);
    checkData();
}
2 usages
private synchronized void checkData() {
    if (bdFaceImageConfig.data != null && bdNirFaceImageConfig.data != null) {
        FaceSDKManager.getInstance().onDetectSilentLiveCheck(bdFaceImageConfig, bdNirFaceImageConfig, bdDepthFaceImageConfig: null,
                bdFaceCheckConfig, new FaceDetectCallBack() {
                    @Override
                    public void onFaceDetectCallback(LivenessModel livenessModel) {
                        // 输出结果
                        if (glMantleSurfaceView.isDraw()) {
                            // 预览模式
                            checkCloseDebugResult(livenessModel);
                        } else {
                            // 开发模式
                            checkOpenDebugResult(livenessModel);
                        }
                        if (isSaveImage){
                            SaveImageManager.getInstance().saveImage(livenessModel , bdLiveConfig);

                        }
                    }

                    @Override
                    public void onTip(int code, String msg) {
                    }

                    @Override
                    public void onFaceDetectDarwCallback(LivenessModel livenessModel) {
                        // 绘制人脸框
//                        if (!glSurfaceView.isDraw) {
                        showFrame(livenessModel);
```

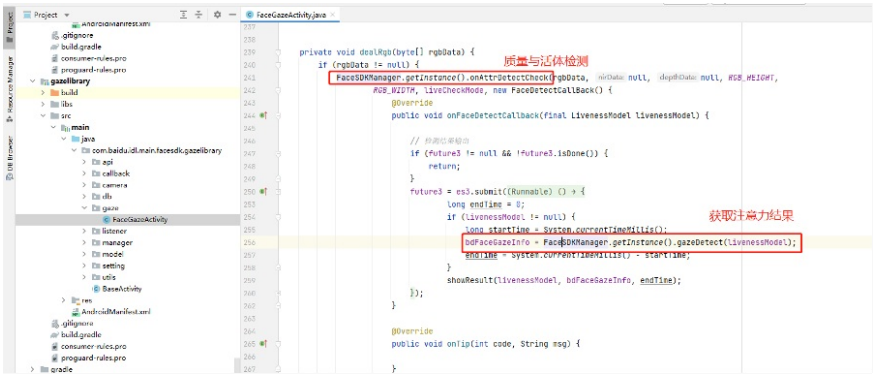
与闸机模式相同但金融活检不会有数据库和识别操作，只走到了活体接口

```
livenessModel.setTestBdFaceImageInstanceDuration(System.currentTimeMillis() - startTime);
onTrack(
    rgbInstance,
    livenessModel,
    new DetectListener() {
        2 usages
        @Override
        public void onDetectSuccess(FaceInfo[] faceInfos, BDFaceImageInstance rgbInstance) {

            // 保存人脸特征点
            livenessModel.setLandmarks(faceInfos[0].landmarks);
            // 保存人脸图片
            livenessModel.setBdFaceImageInstance(rgbInstance.getImage());
            // 调用绘制人脸框接口
            if (faceDetectCallBack != null) {
                faceDetectCallBack.onFaceDetectDarwCallback(livenessModel);
            }
            // 送检识别
            onSilentLivenessCheck(
                rgbInstance,
                bdNirFaceImageConfig,
                bdDepthFaceImageConfig,
                bdFaceCheckConfig,
                livenessModel,
                startTime,
                faceDetectCallBack,
                faceInfos);
        }
```

此模块支持检测帧数调整，默认为1帧（可在BDLiveConfig类中进行修改）,无特征提取和对比识别功能。

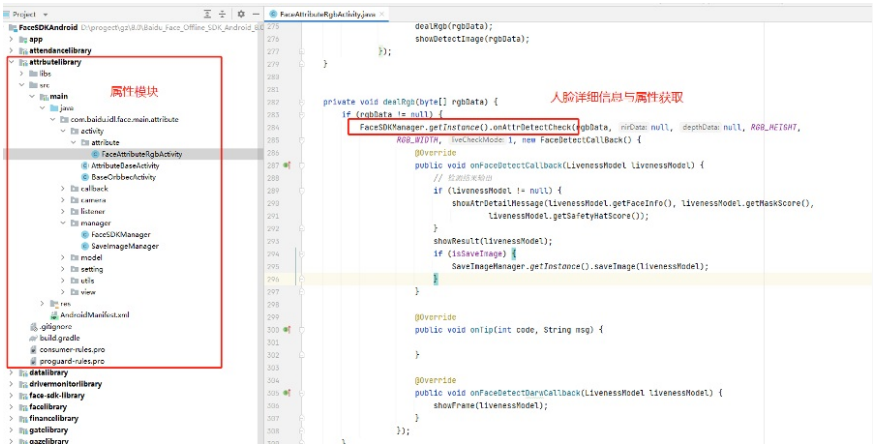
```
// TODO 活体检测
BDLiveConfig bdLiveConfig = bdFaceCheckConfig.bdLiveConfig;
boolean isLiveCheck = bdFaceCheckConfig.bdLiveConfig != null;
if (isLiveCheck) {
    long startRgbTime = System.currentTimeMillis();
    boolean rgbLiveStatus =
            faceModel
                    .getFaceLive()
                    .strategySilentLive(
                            BDFaceSDKCommon.LiveType.BDFACE_SILENT_LIVE_TYPE_RGB,
                            rgbInstance,
                            faceInfos[0],
                            bdLiveConfig.framesThreshold,
                            bdLiveConfig.rgbLiveScore);
    livenessModel.setRGBLiveStatus(rgbLiveStatus);
    livenessModel.setRgbLivenessDuration(
            System.currentTimeMillis() - startRgbTime);
}
```

1.7 注意力模式使用说明



注意力检测，主要检测包括左右眼、向左向右、向上向下与睁闭眼状态

1.8 属性模式说明



属性模式，主要包括口罩、眼镜、年龄与性别检测

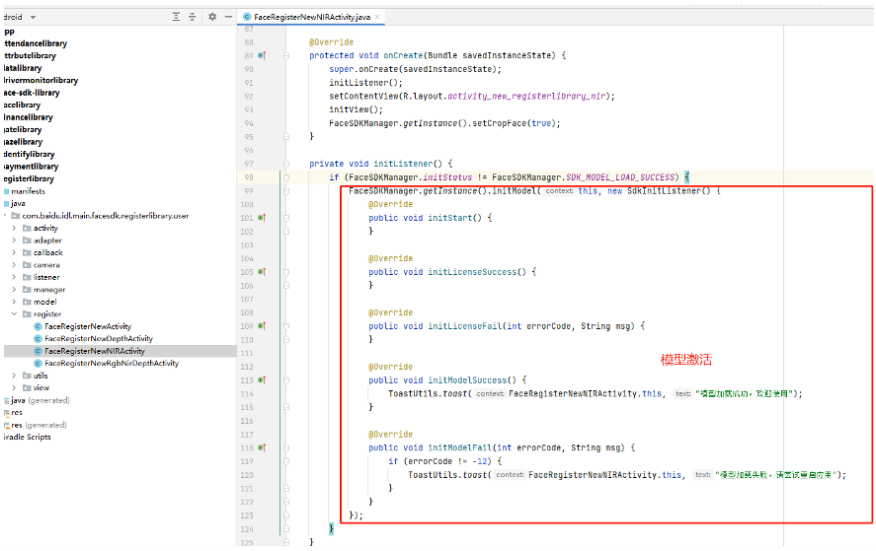
1.9 注册人脸库模块说明

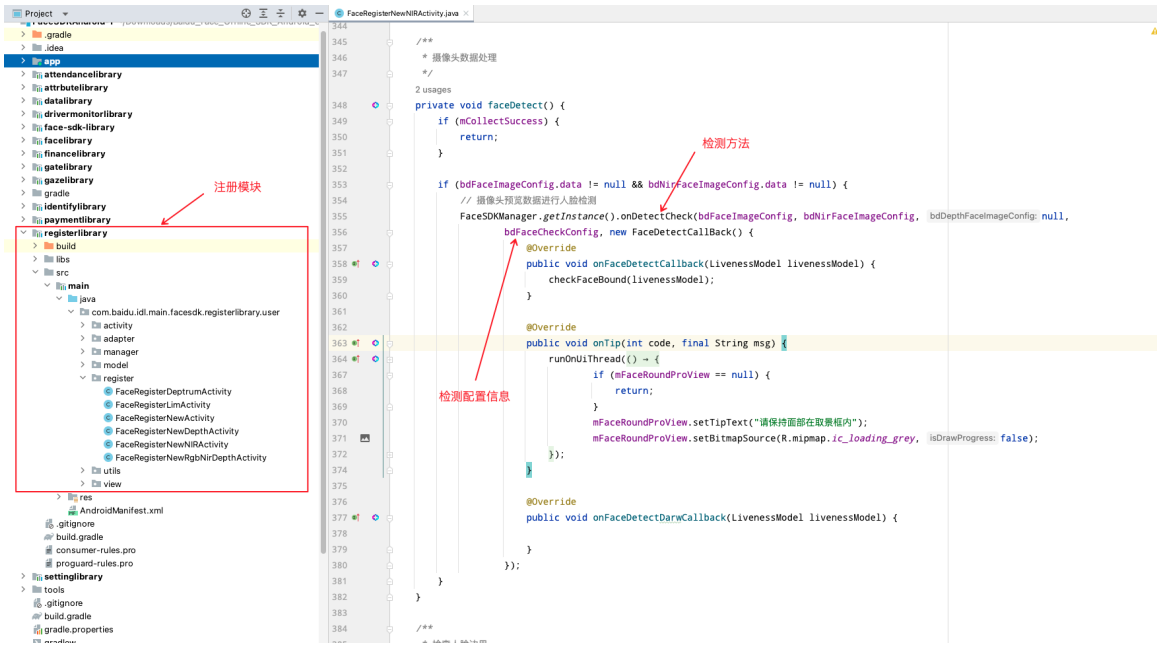
1.9.1视频流检测人脸并注册

1.9.1.1人脸注册

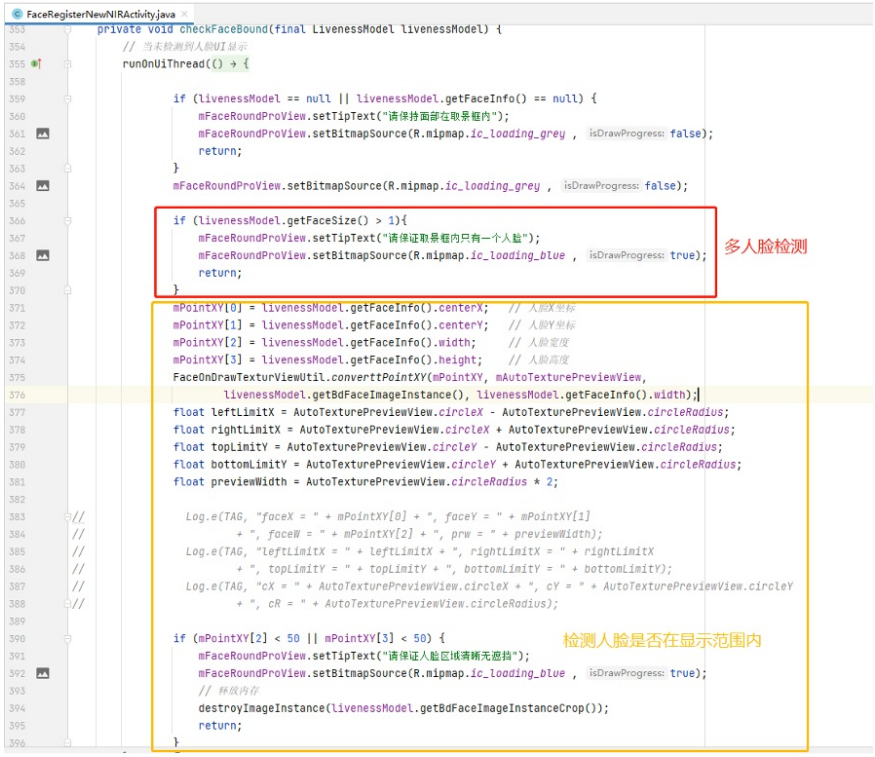
有3种镜头模式选择注册（注意人脸注册镜头模式以考勤，闸机，支付，认证核验模块其中最后一个设置的镜头模式应用于人脸注册，**且demo注册阈值设置为强偶合，比如闸机、考勤模式阈值在程序中修改后注册模式也会对应修改，并且目前不支持其他模式与注册同步线程使用。多线程清参考上方3.10)**

1.9.1.2检测部分





1.9.1.3人脸范围、质量、活体判断



```
mPointXY[0] = livenessModel.getFaceInfo().centerX;   // 人脸X坐标
mPointXY[1] = livenessModel.getFaceInfo().centerY;   // 人脸Y坐标
mPointXY[2] = livenessModel.getFaceInfo().width;     // 人脸宽度
mPointXY[3] = livenessModel.getFaceInfo().height;    // 人脸高度
FaceOnDrawTexturViewUtil.converttPointXY(mPointXY, mAutoTexturePreviewView,
        livenessModel.getBdFaceImageInstance(), livenessModel.getFaceInfo().width);
float leftLimitX = AutoTexturePreviewView.circleX - AutoTexturePreviewView.circleRadius;
float rightLimitX = AutoTexturePreviewView.circleX + AutoTexturePreviewView.circleRadius;
float topLimitY = AutoTexturePreviewView.circleY - AutoTexturePreviewView.circleRadius;
float bottomLimitY = AutoTexturePreviewView.circleY + AutoTexturePreviewView.circleRadius;
float previewWidth = AutoTexturePreviewView.circleRadius * 2;
if (mPointXY[2] < 50 || mPointXY[3] < 50) {
    mFaceRoundProView.setTipText("请保证人脸区域清晰无遮挡");
    mFaceRoundProView.setBitmapSource(R.mipmap.ic_loading_blue , isDrawProgress: true);
    // 释放内存
    destroyImageInstance(livenessModel.getBdFaceImageInstanceCrop());
    return;
}

if (mPointXY[2] > previewWidth || mPointXY[3] > previewWidth) {
    mFaceRoundProView.setTipText("请保证人脸区域清晰无遮挡");
    mFaceRoundProView.setBitmapSource(R.mipmap.ic_loading_blue , isDrawProgress: true);
    // 释放内存
    destroyImageInstance(livenessModel.getBdFaceImageInstanceCrop());
    return;
}

if (mPointXY[0] - mPointXY[2] / 2 < leftLimitX
        || mPointXY[0] + mPointXY[2] / 2 > rightLimitX
        || mPointXY[1] - mPointXY[3] / 2 < topLimitY
        || mPointXY[1] + mPointXY[3] / 2 > bottomLimitY) {
    mFaceRoundProView.setTipText("请保证人脸区域清晰无遮挡");
    mFaceRoundProView.setBitmapSource(R.mipmap.ic_loading_blue , isDrawProgress: true);
    // 释放内存
    destroyImageInstance(livenessModel.getBdFaceImageInstanceCrop());
    return;
}
```

判断人脸是否在展示范围内

```
/**
 * 检验活体分值
 *
 * @param livenessModel LivenessModel实体
 */
private void checkLiveScore(LivenessModel livenessModel) {
    if (livenessModel == null || livenessModel.getFaceInfo() == null) {
        mFaceRoundProView.setTipText("请保持面部在取景框内");
        return;
    }

    float rgbLivenessScore = livenessModel.getRgbLivenessScore();
    float irLivenessScore = livenessModel.getIrLivenessScore();
    float liveThreadHold = SingleBaseConfig.getBaseConfig().getRgbLiveScore();
    float liveIrThreadHold = SingleBaseConfig.getBaseConfig().getNirLiveScore();
    Log.e(TAG,  msg: "score = " + rgbLivenessScore + ", ns = " + irLivenessScore);
    if (! livenessModel.isQualityCheck()){
        mFaceRoundProView.setTipText("请保证人脸区域清晰无遮挡");
        mFaceRoundProView.setBitmapSource(R.mipmap.ic_loading_blue , isDrawProgress: true);
        return;
    } else if (rgbLivenessScore < liveThreadHold || irLivenessScore < liveIrThreadHold) {
        mFaceRoundProView.setTipText("请保证采集对象为真人");
        mFaceRoundProView.setBitmapSource(R.mipmap.ic_loading_blue , isDrawProgress: true);
        // 释放内存
        destroyImageInstance(livenessModel.getBdFaceImageInstanceCrop());
        return;
    }
    // 提取特征值
    getFeatures(livenessModel);
}
```

rgb和nir活体得分

rgb nir活体阈值

判断质量检测是否通过

判断活体检测是否通过

全部通过进行人脸特征提取注册

1.9.1.4提取完毕的特征人脸信息，注册到本地数据库

FaceRegisterNewNIRActivity.java

```
507             displayCompareResult(ret, model.getFeature(), model);
508         }
509
510         // 根据特征抽取的结果 注册人脸
            1 usage
511         private void displayCompareResult(float ret, byte[] faceFeature, LivenessModel model) {
512             if (model == null) {
513                 mFaceRoundProView.setTipText("请保持面部在取景框内");
514                 mFaceRoundProView.setBitmapSource(R.mipmap.ic_loading_grey , isDrawProgress: false);
515                 return;
516             }
517
518             // 特征提取成功
519             if (ret == 128) {
520                 BDFaceImageInstance cropInstance =
521                         FaceSDKManager.getInstance().getCopeFace(
522                                 BitmapUtils.getInstaceBmp(model.getBdFaceImageInstance()) ,
523                                 model.getLandmarks(),
524                                 initialValue: 0
525                         ); ;
526                 if (cropInstance == null) {
527                     mFaceRoundProView.setTipText("抠图失败");
528                     mFaceRoundProView.setBitmapSource(R.mipmap.ic_loading_blue , isDrawProgress: true);
529                     return;
530                 }
531                 mCropBitmap = BitmapUtils.getInstaceBmp(cropInstance);
532                 // 获取头像
533                 if (mCropBitmap != null) {
534                     mCollectSuccess = true;
535                     mCircleHead.setImageBitmap(mCropBitmap);
536                 }
537                 cropInstance.destory();
538
539                 mRelativeCollectSuccess.setVisibility(View.VISIBLE);
540                 mRelativePreview.setVisibility(View.GONE);
541                 mFaceRoundProView.setTipText("");
542
543                 for (int i = 0; i < faceFeature.length; i++) {
544                     mFeatures[i] = faceFeature[i];
545                 }
```

提取的特征值，512byte数组

判断是否成功

保存特征

```
    } else if (id == R.id.btn_collect_confirm) {   // 用户名注册
        String userName = mEditName.getText().toString();
            if (TextUtils.isEmpty(userName)) {
                ToastUtils.toast(getApplicationContext(), "请先输入用户名");
                return;
            }
            if (userName.length() > 10) {
                ToastUtils.toast(getApplicationContext(), "用户名长度不得大于10位");
                return;
            }
        // 姓名过滤
        String nameResult = FaceApi.getInstance().isValidName(userName);
        if (!"0".equals(nameResult)) {
            ToastUtils.toast(getApplicationContext(), nameResult);
            return;
        }
        String imageName = userName + ".jpg";
        // 注册到人脸库
        boolean isSuccess = FaceApi.getInstance().registerUserIntoDBmanager( groupName: null,
                userName, imageName, userInfo: null, mFeatures);
        if (isSuccess) {
            // 保存人脸图片
            File faceDir = FileUtils.getBatchImportSuccessDirectory();
            File file = new File(faceDir, imageName);
            FileUtils.saveBitmap(file, mCropBitmap);
            // 数据变化，更新内存
              FaceSDKManager.getInstance().initDatabases();
            // 更新UI
            mRelativeCollectSuccess.setVisibility(View.GONE);
            mRelativeRegisterSuccess.setVisibility(View.VISIBLE);
            mCircleRegSucHead.setImageBitmap(mCropBitmap);
        } else {
            ToastUtils.toast(getApplicationContext(), text: "保存数据库失败，" +
                    "可能是用户名格式不正确");
        }
```

人脸姓名以及特征存储

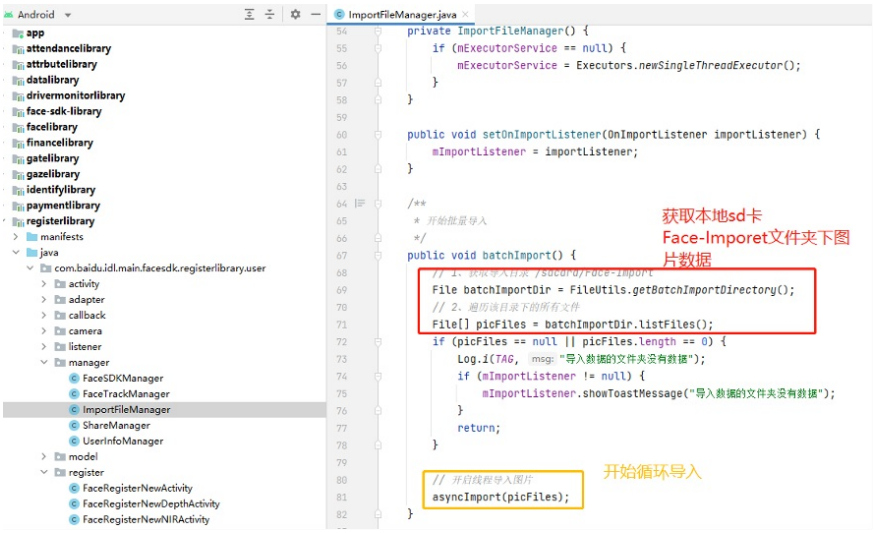
注：

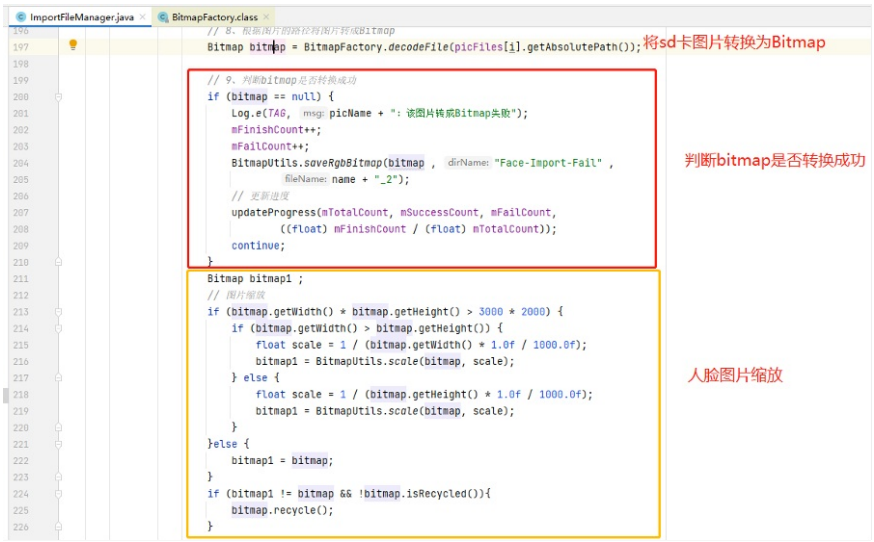
1.demo用户名为必填项，支持英文、汉字、数字和下划线。

2.注册成功后的图片命名和格式为设置名.jpg,保存在sd卡下的Success-Import文件夹下，用于人脸库的图片显示。

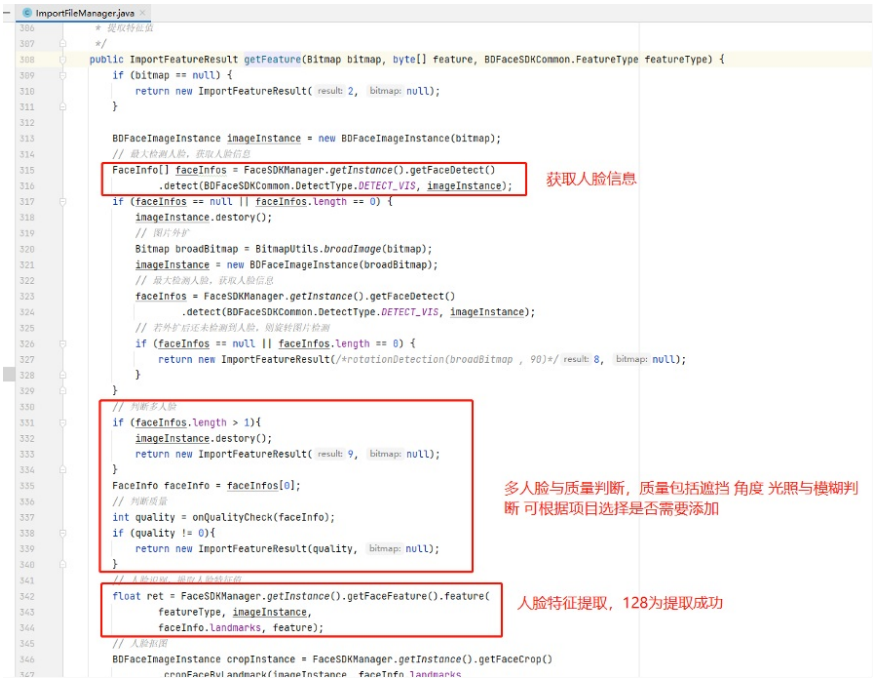
3.demo使用安卓原生数据库，开发者可根据自己的业务需求场景使用其他安卓开源数据库，添加保存人脸信息。

1.9.2 图片bitmap人脸注册（也可也以参考上方3.10进行图片注册）





特征提取



1.10 支付模块UI二次开发指引

问题：

对识别SDK进入具体模块后对应UI二次开发指引

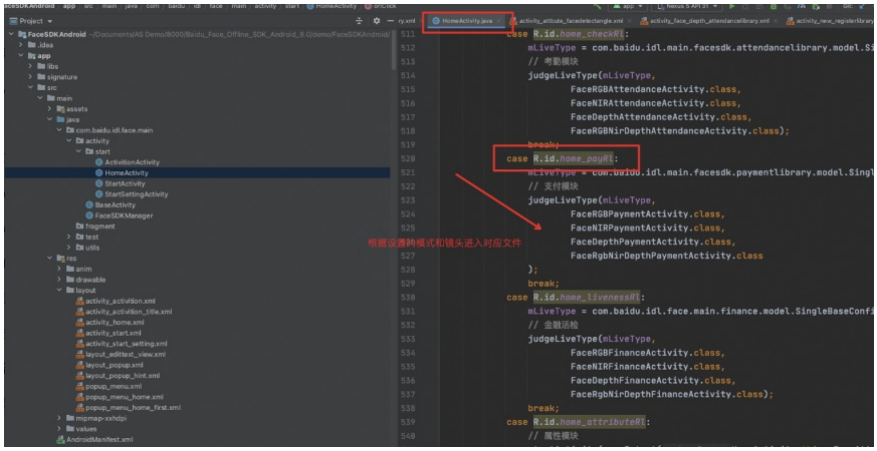
解决：

以支付模块为例，对进入模块后的UI与布局进行分析，并结合用户业务需求进行二次开发指引

在主页面中，点击进入具体模块，如这里点击进入支付模式

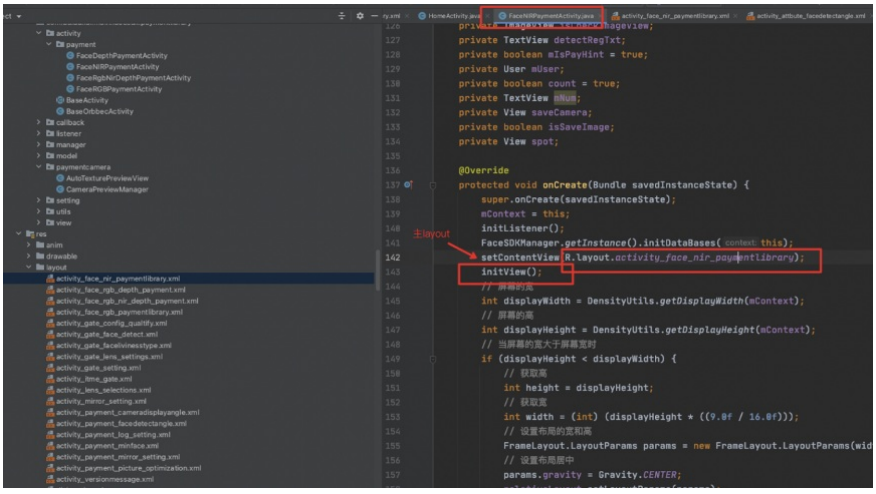
<" width=70%/>

对应于HomeActivity页面点击支付模块跳转分页面，这里有一点需注意：这里共分了四个跳转文件，具体会跳转进入哪一个取决于SDK中在单双目中设置的哪个模式和摄像头的设置，这里是以双目rgb+nir的模式为例进行展开讲解。

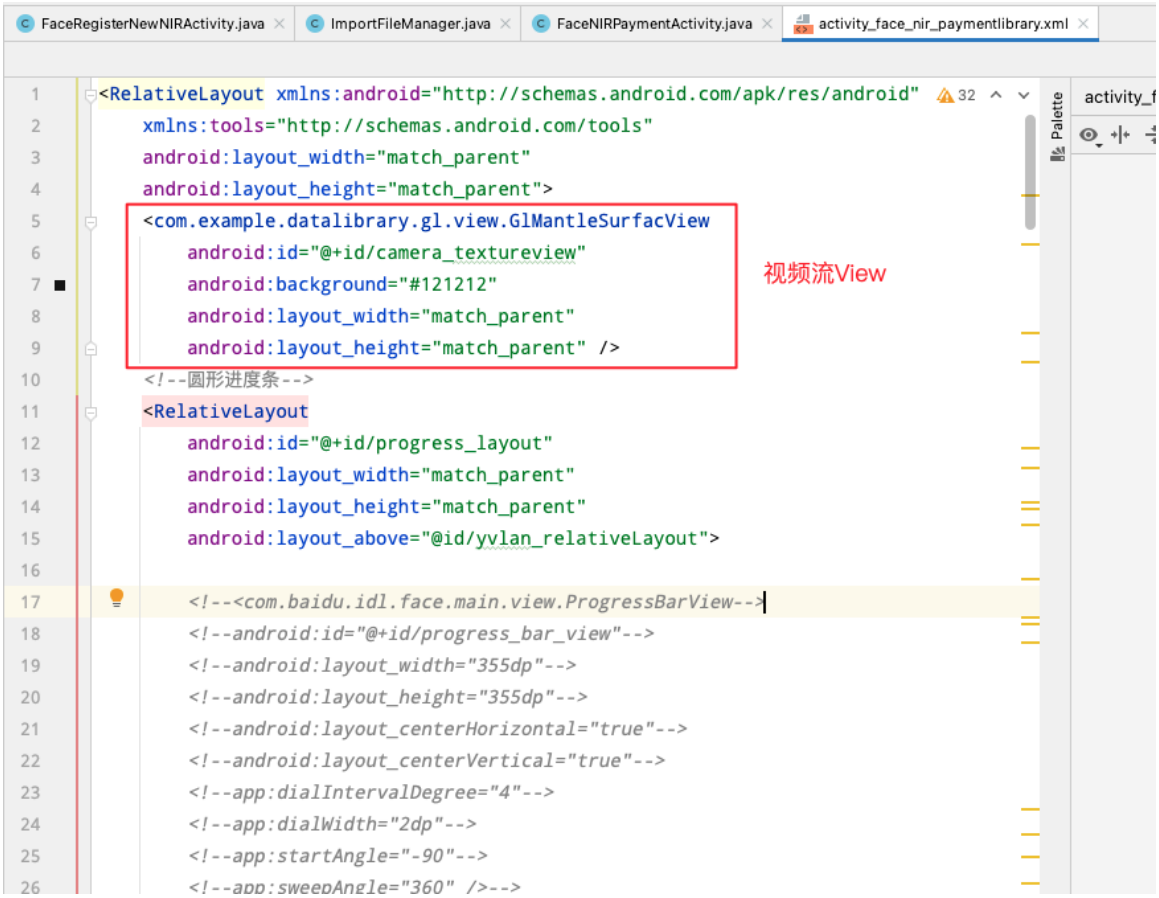


通过点击来到支付模块的主页面

对应布局逻辑如下



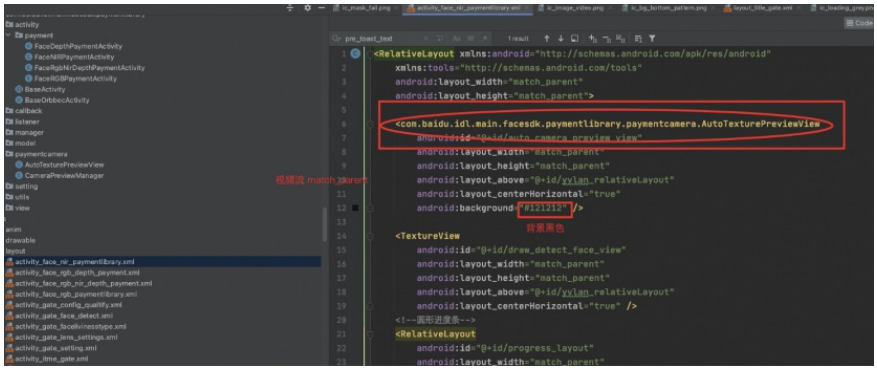
支付模块主页面对应的布局文件为 activity_face_nir_paymentlibrary（依然是同上以rgb+nir双目为例）



总体思路是将所有的UI都全部放在一份layout布局中，并通过visible和嵌套layout的方式控制哪些显示哪些不显示，之后在逻辑中通过click的visible来控制显示，包括字体颜色和画面以及控件。

下面进行详细剖析：

视频进入并填充整个画面



引入圆形进度条

```
        android:layout_centerHorizontal="true" />
    <!--圆形进度条-->
    <RelativeLayout
        android:id="@+id/progress_layout"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_above="@id/yvlan_relativeLayout">

        <!--<com.baidu.idl.face.main.view.ProgressBarView-->
        <!--android:id="@+id/progress_bar_view"-->
        <!--android:layout_width="355dp"-->
        <!--android:layout_height="355dp"-->
        <!--android:layout_centerHorizontal="true"-->
        <!--android:layout_centerVertical="true"-->
        <!--app:dialIntervalDegree="4"-->
        <!--app:dialWidth="2dp"-->
        <!--app:startAngle="-90"-->
        <!--app:sweepAngle="360" />-->
        <ImageView
            android:id="@+id/progress_bar_view"
            android:layout_width="355dp"
            android:layout_height="355dp"
            android:layout_centerHorizontal="true"
            android:layout_centerVertical="true"
            android:src="@mipmap/ic_loading_grey" />
    </RelativeLayout>
```

圆形进度条图片对应UI

填充文字部分，设置颜色、布局方向、字体大小

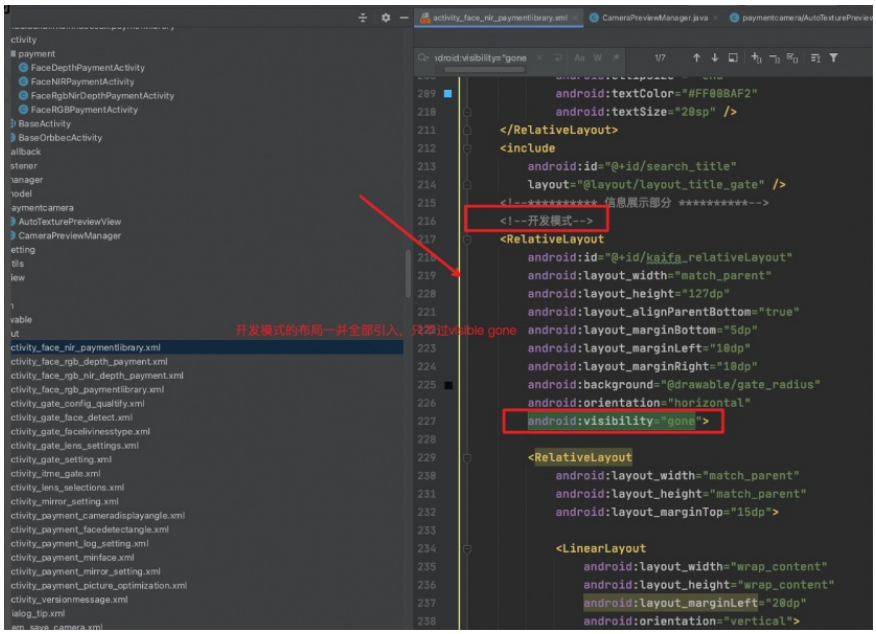
```
    <TextView
        android:id="@+id/pre_toast_text"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_above="@id/yvlan_relativeLayout"
        android:gravity="center_horizontal"
        android:text="请保持面部在取景框内"
        android:textColor="#ffffff"
        android:textSize="22sp" />

    <RelativeLayout
        android:id="@+id/pay_hintRl"
        android:layout_width="match_parent"
```

预览模式和开发模式的布局一并全部引入，只不过使用visible gone控制

预览模式为默认模式显示

```
        </RelativeLayout>

    </RelativeLayout>

    <!--预览模式-->
    <RelativeLayout
        android:id="@+id/yvlan_relativeLayout"
        android:layout_width="match_parent"
        android:layout_height="200dp"
        android:layout_alignParentBottom="true"
        android:background="@mipmap/ic_bg_bottom_pattern"
        android:gravity="center_horizontal"
        android:visibility="visible">

        <TextView
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_alignParentBottom="true"
            android:layout_marginBottom="15dp"
            android:alpha="0.4"
            android:gravity="center_horizontal"
            android:text="— 百度大脑技术支持 —"
            android:textColor="#666666"
            android:textSize="12sp" />
    </RelativeLayout>

</RelativeLayout>
```

开发模式显示也在该layout中全部引入，通过属性控制不显示



其中包括search_title，由于这是一个通用多次用到的布局，所以单独放在一个layout中

该layout_title_gate真实布局如下

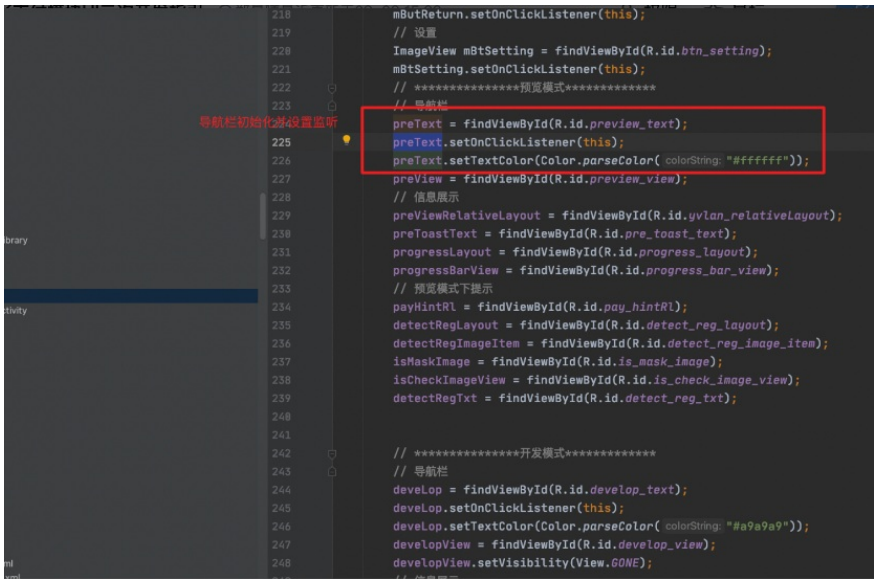
```
    android:background="#121212">

    <ImageView
        android:id="@+id/btn_back"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginLeft="15dp"
        android:layout_centerVertical="true"
        android:padding="5dp"
        android:src="@mipmap/ic_return"
        android:text="返回"
        android:textColor="#ffffff" />
    各个模块的点击也在click中完成

    <ImageView
        android:id="@+id/preview_view"
        android:layout_width="match_parent"
        android:layout_height="12dp"
        android:layout_below="@+id/preview_text"
        android:layout_alignLeft="@id/preview_text"
        android:layout_alignRight="@id/preview_text"
        android:layout_marginTop="-12dp"
        android:src="@mipmap/ic_highlight" />
    <TextView
        android:id="@+id/preview_text"
        android:layout_width="wrap_content"
        android:layout_height="35dp"
        android:layout_marginStart="50dp"
        android:layout_toEndOf="@+id/btn_back"
        android:layout_centerVertical="true"
        android:text="预览模式"
        android:textColor="#d3d3d3"
        android:textSize="12sp"
        tools:ignore="RtlCompat" />
```

显示了每次模块中最顶部的UI,对应UI为

以上是静态UI布局，在完成初始化后通过visible展示

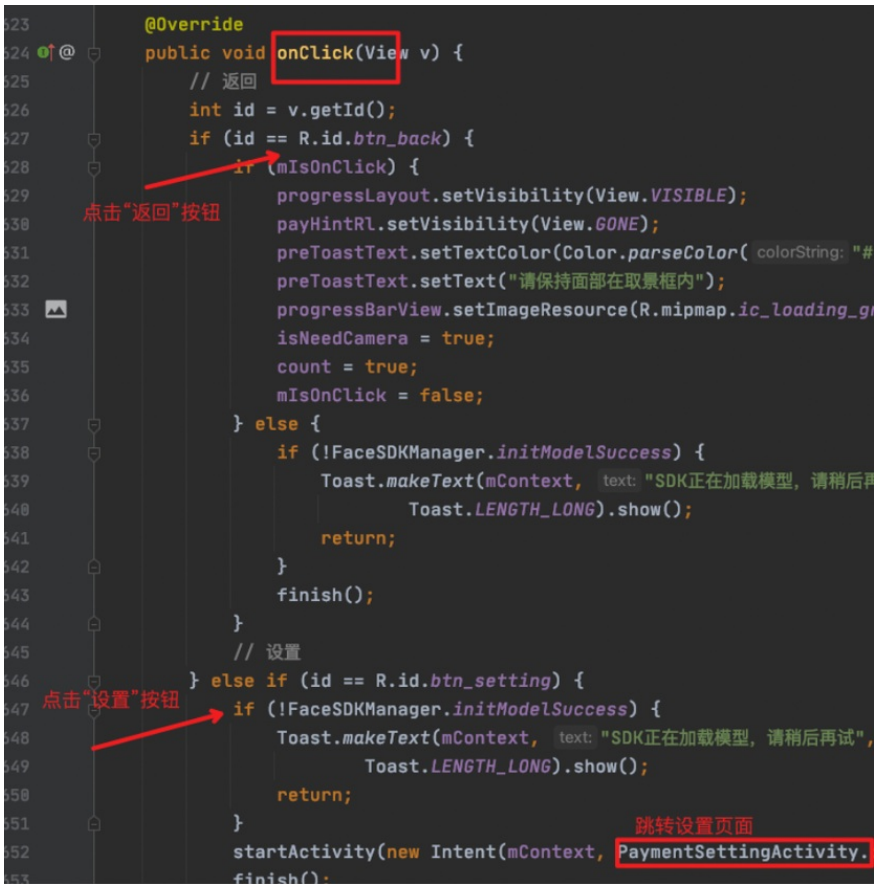
```
    */
    private void initView() {

        // 获取整个布局
        relativeLayout = findViewById(R.id.all_relative);
        // 画人脸框
        rectF = new RectF();
        paint = new Paint();
        paintBg = new Paint();
        mDrawDetectFaceView = findViewById(R.id.draw_detect_face_view);
        mDrawDetectFaceView.setOpaque(false);
        mDrawDetectFaceView.setKeepScreenOn(true);
        if (SingleBaseConfig.getBaseConfig().getRgbRevert()){
            mDrawDetectFaceView.setRotationY(180);
        }
        // 双目摄像头RGB 图像预览
        mAutoCameraPreviewView = findViewById(R.id.auto_camera_preview_view);
        mAutoCameraPreviewView.setVisibility(View.VISIBLE);
        mAutoCameraPreviewView.isDraw = true;

        // 返回
```

导航栏设置监听



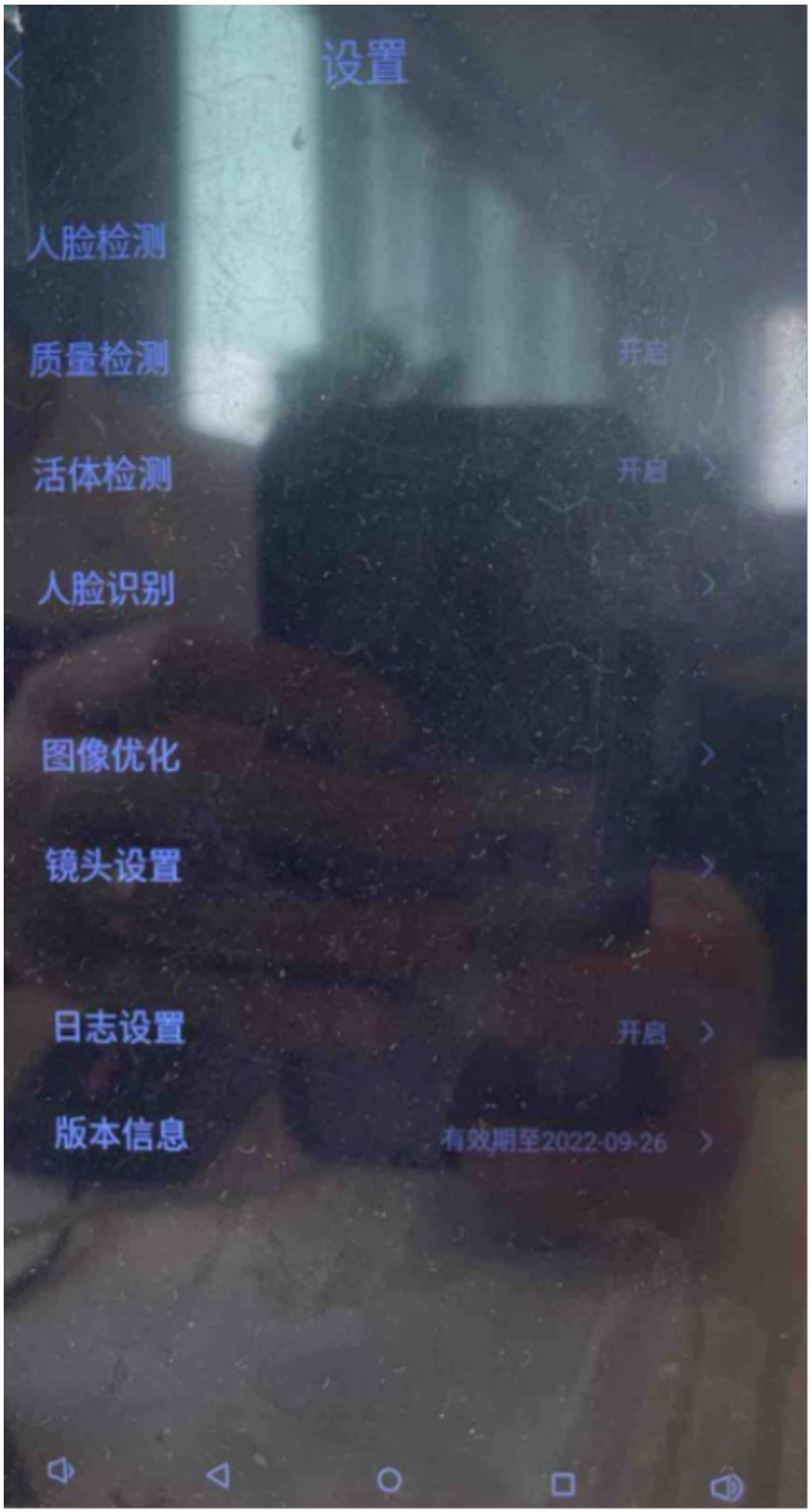

```
        mButReturn.setOnClickListener(this);
        // 设置
        ImageView mBtSetting = findViewById(R.id.btn_setting);
        mBtSetting.setOnClickListener(this);
        // ***************预览模式*************
        // 导航栏
        preText = findViewById(R.id.preview_text);
        preText.setOnClickListener(this);
        preText.setTextColor(Color.parseColor( colorString: "#ffffff"));
        preView = findViewById(R.id.preview_view);
        // 信息展示
        preViewRelativeLayout = findViewById(R.id.yvlan_relativeLayout);
        preToastText = findViewById(R.id.pre_toast_text);
        progressLayout = findViewById(R.id.progress_layout);
        progressBarView = findViewById(R.id.progress_bar_view);
        // 预览模式下提示
        payHintRl = findViewById(R.id.pay_hintRl);
        detectRegLayout = findViewById(R.id.detect_reg_layout);
        detectRegImageItem = findViewById(R.id.detect_reg_image_item);
        isMaskImage = findViewById(R.id.is_mask_image);
        isCheckImageView = findViewById(R.id.is_check_image_view);
        detectRegTxt = findViewById(R.id.detect_reg_txt);


        // ***************开发模式*************
        // 导航栏
        develop = findViewById(R.id.develop_text);
        develop.setOnClickListener(this);
        develop.setTextColor(Color.parseColor( colorString: "#a9a9a9"));
        developView = findViewById(R.id.develop_view);
        developView.setVisibility(View.GONE);
```

一旦点击对应按钮跳转对应逻辑

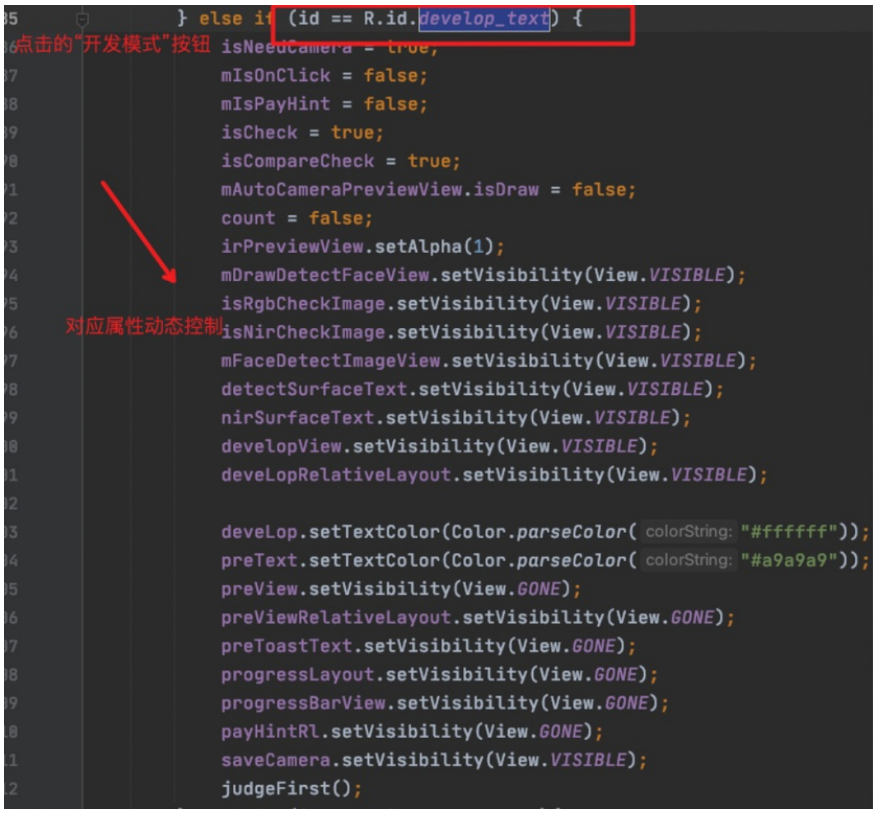



```
    @Override
    public void onClick(View v) {
        // 返回
        int id = v.getId();
        if (id == R.id.btn_back) {
            if (mIsOnClick) {
                progressLayout.setVisibility(View.VISIBLE);
                payHintRl.setVisibility(View.GONE);
                preToastText.setTextColor(Color.parseColor( colorString: "#
                preToastText.setText("请保持面部在取景框内");
                progressBarView.setImageResource(R.mipmap.ic_loading_gr
                isNeedCamera = true;
                count = true;
                mIsOnClick = false;
            } else {
                if (!FaceSDKManager.initModelSuccess) {
                    Toast.makeText(mContext, text: "SDK正在加载模型，请稍后再
                            Toast.LENGTH_LONG).show();
                    return;
                }
                finish();
            }
            // 设置
        } else if (id == R.id.btn_setting) {
            if (!FaceSDKManager.initModelSuccess) {
                Toast.makeText(mContext, text: "SDK正在加载模型，请稍后再试",
                        Toast.LENGTH_LONG).show();
                return;
            }
            startActivity(new Intent(mContext, PaymentSettingActivity.
            finish();
```

点击“设置“按钮，跳转UI



若从默认的“预览模式”点击“开发模式”按钮，根据点击将对应UI反向Visible及对应gone

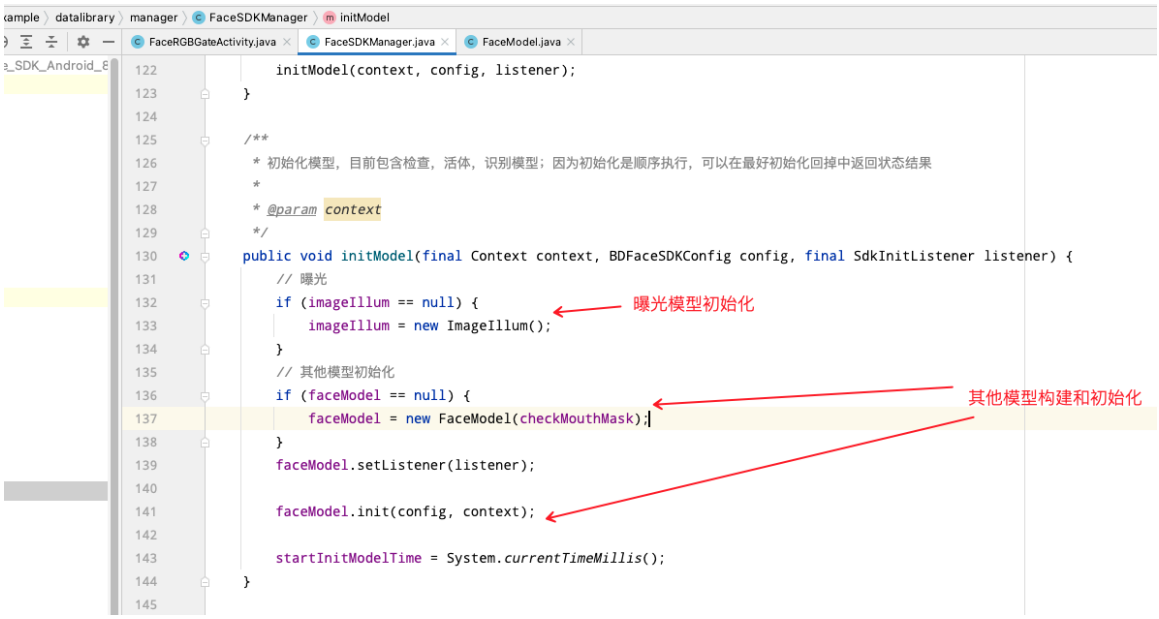


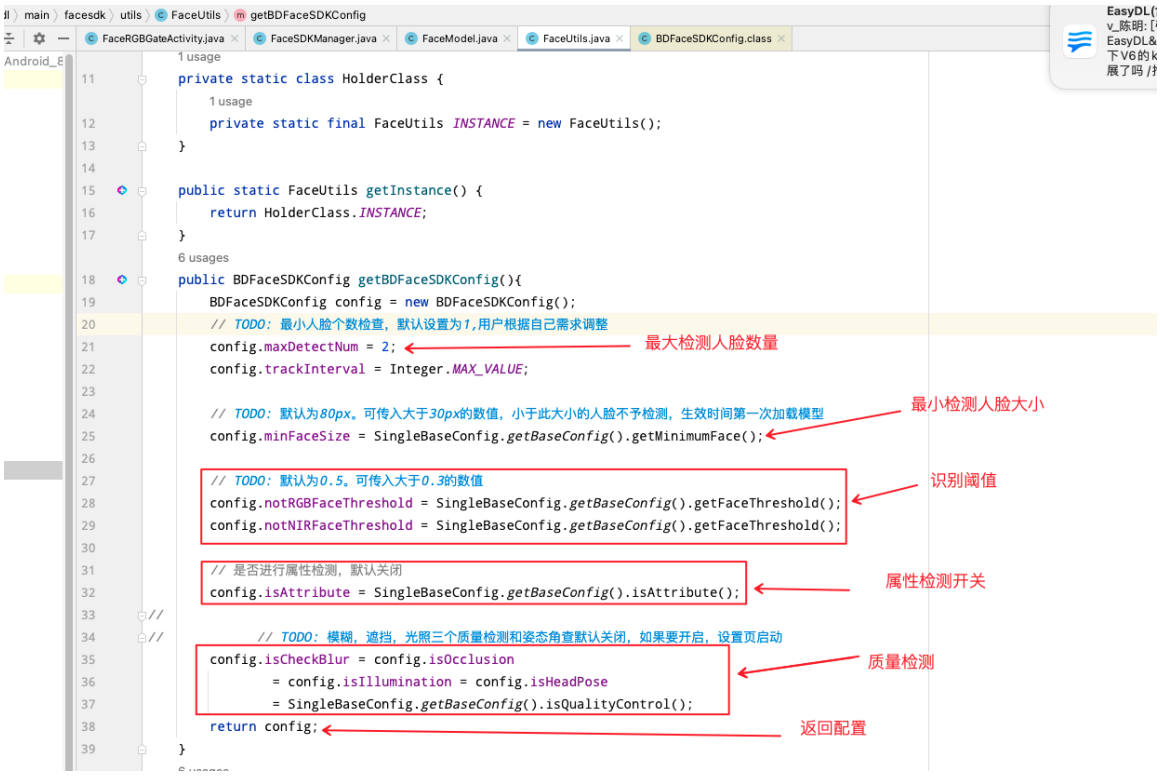
1.11 识别流程方法和阈值说明

FaceSDKManager.class

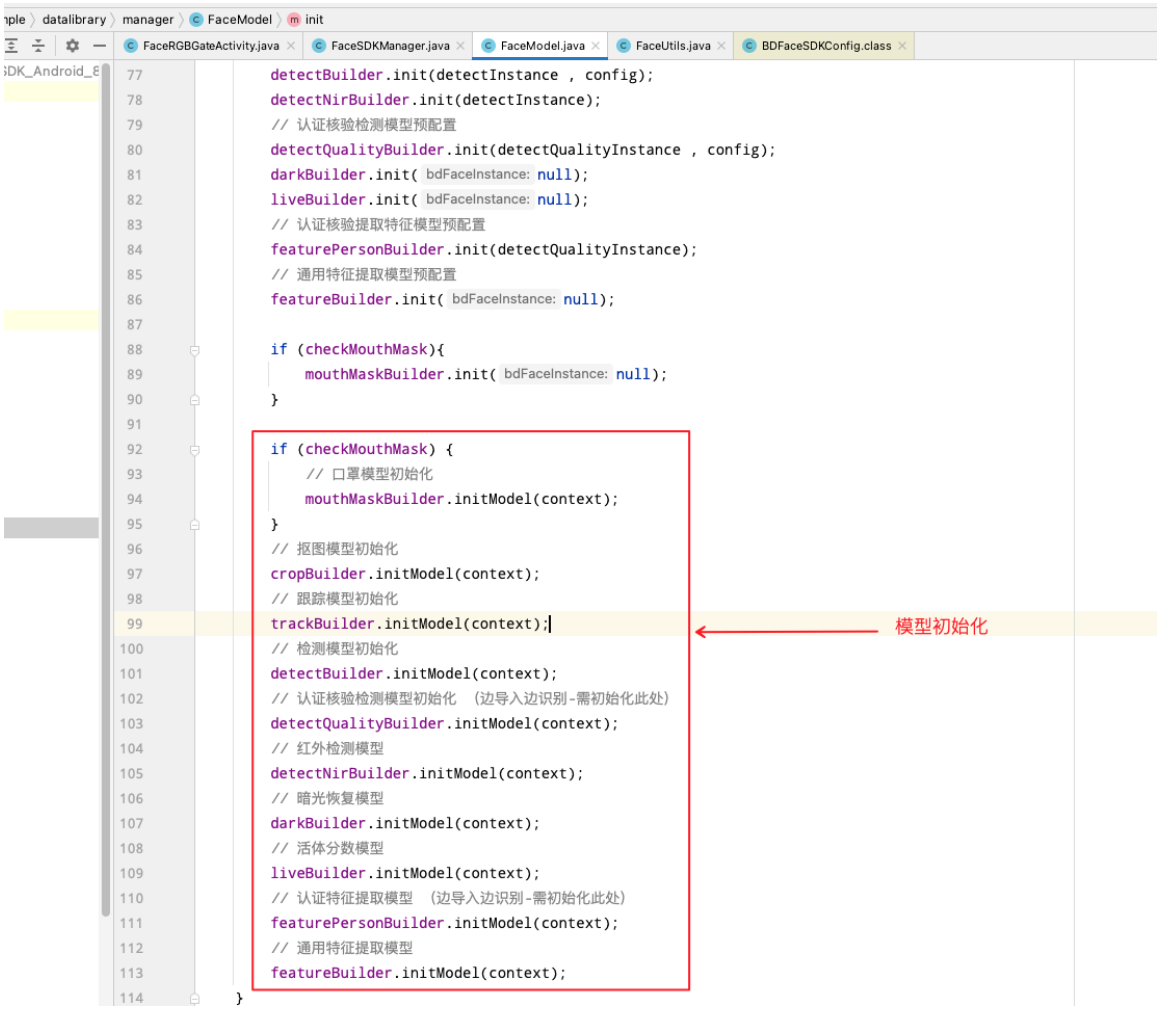
每个模块都有相应的FaceSDKManager.class。

1.11.1 模型初始化

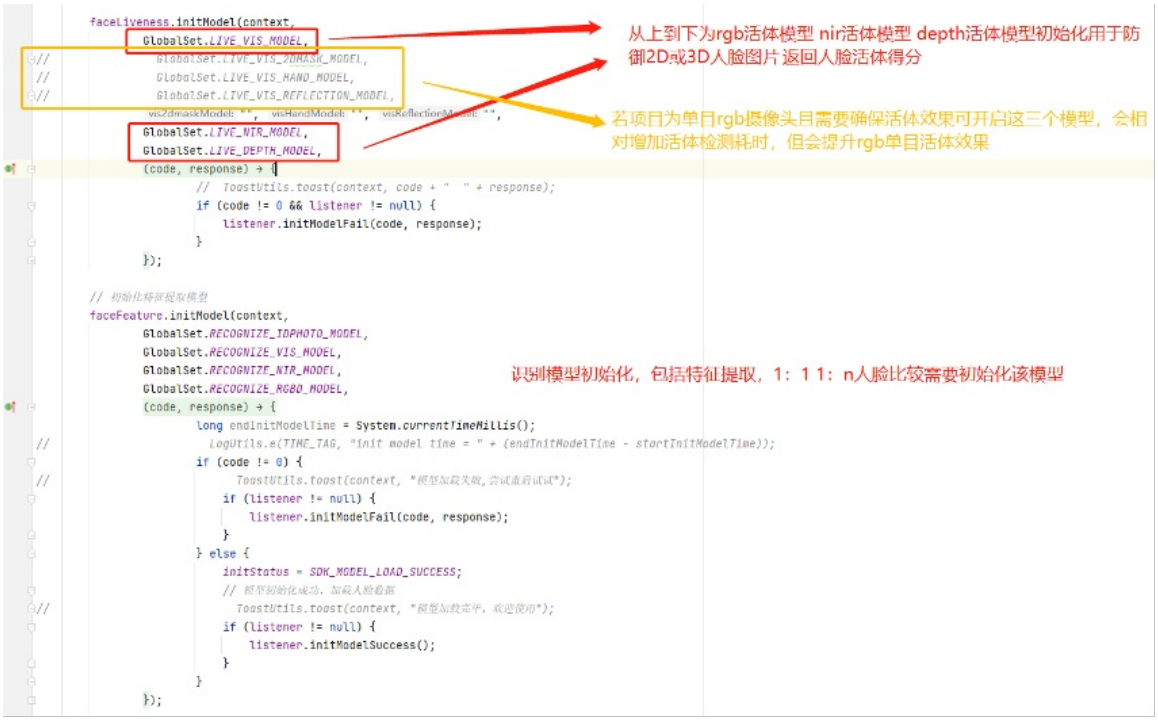




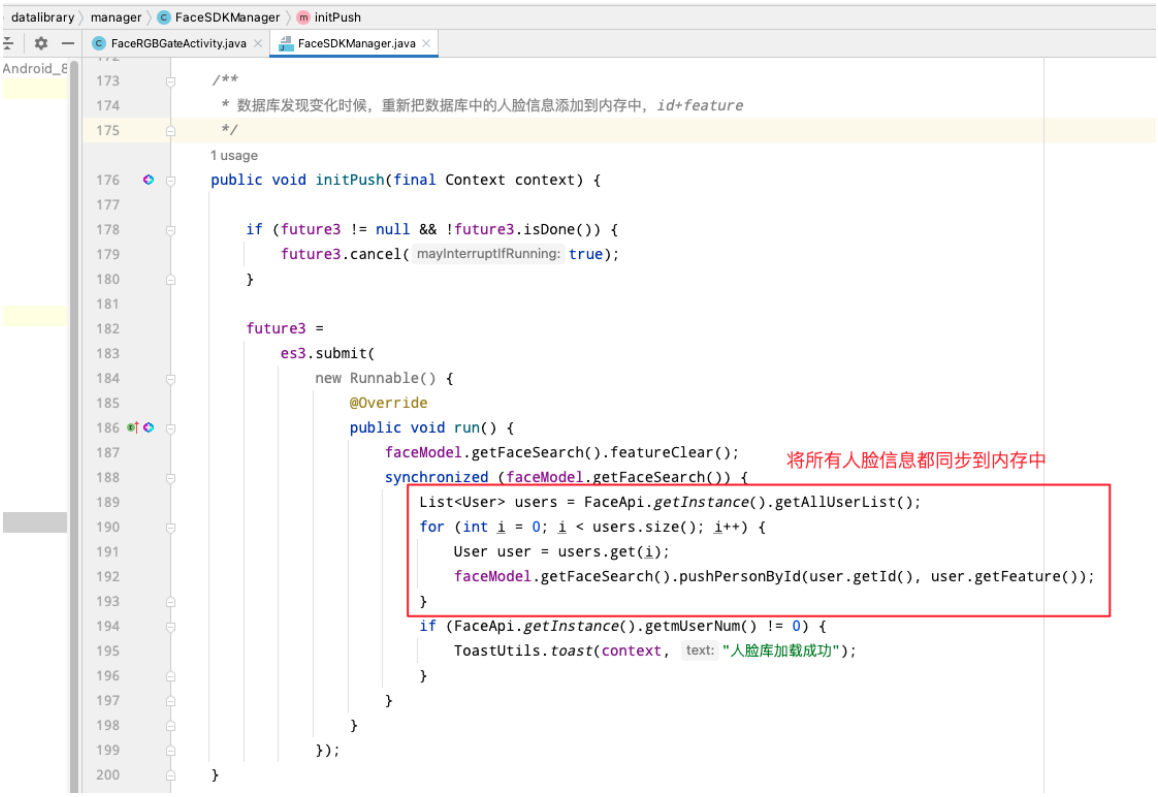

```
1 usage
private static class HolderClass {
    1 usage
    private static final FaceUtils INSTANCE = new FaceUtils();
}

public static FaceUtils getInstance() {
    return HolderClass.INSTANCE;
}
6 usages
public BDFaceSDKConfig getBDFaceSDKConfig(){
    BDFaceSDKConfig config = new BDFaceSDKConfig();
    // TODO: 最小人脸个数检查，默认设置为1,用户根据自己需求调整
    config.maxDetectNum = 2;
    config.trackInterval = Integer.MAX_VALUE;

    // TODO: 默认为80px。可传入大于30px的数值，小于此大小的人脸不予检测，生效时间第一次加载模型
    config.minFaceSize = SingleBaseConfig.getBaseConfig().getMinimumFace();

    // TODO: 默认为0.5。可传入大于0.3的数值
    config.notRGBFaceThreshold = SingleBaseConfig.getBaseConfig().getFaceThreshold();
    config.notNIRFaceThreshold = SingleBaseConfig.getBaseConfig().getFaceThreshold();

    // 是否进行属性检测，默认关闭
    config.isAttribute = SingleBaseConfig.getBaseConfig().isAttribute();
//
//          // TODO: 模糊，遮挡，光照三个质量检测和姿态角查默认关闭，如果要开启，设置页启动
    config.isCheckBlur = config.isOcclusion
            = config.isIllumination = config.isHeadPose
            = SingleBaseConfig.getBaseConfig().isQualityControl();
    return config;
}
```




```
detectBuilder.init(detectInstance , config);
detectNirBuilder.init(detectInstance);
// 认证核验检测模型预配置
detectQualityBuilder.init(detectQualityInstance , config);
darkBuilder.init( bdFaceInstance: null);
liveBuilder.init( bdFaceInstance: null);
// 认证核验提取特征模型预配置
featurePersonBuilder.init(detectQualityInstance);
// 通用特征提取模型预配置
featureBuilder.init( bdFaceInstance: null);

if (checkMouthMask){
    mouthMaskBuilder.init( bdFaceInstance: null);
}

if (checkMouthMask) {
    // 口罩模型初始化
    mouthMaskBuilder.initModel(context);
}
// 抠图模型初始化
cropBuilder.initModel(context);
// 跟踪模型初始化
trackBuilder.initModel(context);
// 检测模型初始化
detectBuilder.initModel(context);
// 认证核验检测模型初始化 （边导入边识别-需初始化此处）
detectQualityBuilder.initModel(context);
// 红外检测模型
detectNirBuilder.initModel(context);
// 暗光恢复模型
darkBuilder.initModel(context);
// 活体分数模型
liveBuilder.initModel(context);
// 认证特征提取模型 （边导入边识别-需初始化此处）
featurePersonBuilder.initModel(context);
// 通用特征提取模型
featureBuilder.initModel(context);
}
```



1.11.2 将人脸注册数据放进sdk内存



1.11.3 识别流程调用

图1

```
1 usage
private void startTestOpenDebugRegisterFunction() {
    if (SingleBaseConfig.getBaseConfig().getRBGCameraId() != -1) {
        CameraPreviewManager.getInstance().setCameraFacing(SingleBaseConfig.getBaseConfig().getRBGCameraId());
    } else {
        CameraPreviewManager.getInstance().setCameraFacing(CameraPreviewManager.CAMERA_USB);
    }
    int[] cameraSize = CameraPreviewManager.getInstance().initCamera();
    initFaceConfig(cameraSize[1], cameraSize[0]);
    isPause = true;
    CameraPreviewManager.getInstance().setmCameraDataCallback(new CameraDataCallback() {
        @Override
        public void onGetCameraData(byte[] data, Camera camera, int width, int height) {
            glMantleSurfacView.setFrame();

            bdFaceImageConfig.setData(data);
            FaceSDKManager.getInstance().onDetectCheck(bdFaceImageConfig, bdNirFaceImageConfig: null, bdDepthFaceImageConfig: null,
                    bdFaceCheckConfig, new FaceDetectCallBack() {
                @Override
                public void onFaceDetectCallback(LivenessModel livenessModel) {
                    // 预览模式
                    checkCloseDebugResult(livenessModel);
//                    String str = "";
//                    Toast.makeText(mContext, str.length() + "", Toast.LENGTH_SHORT).show();
                    // 开发模式
                    checkOpenDebugResult(livenessModel);
                    if (isSaveImage) {
                        SaveImageManager.getInstance().saveImage(livenessModel, bdLiveConfig);
                    }
                }

                @Override
                public void onTip(int code, String msg) {
                }
```

图2

```
    */
public void onDetectCheck(final BDFaceImageConfig bdFaceImageConfig, final BDFaceImageConfig bdNirFaceImageConfig,
    final BDFaceImageConfig bdDepthFaceImageConfig, final BDFaceCheckConfig bdFaceCheckConfig,
    final FaceDetectCallBack faceDetectCallBack) {
    if (!FaceSDKManager.initModelSuccess) {
        return;
    }
    long startTime = System.currentTimeMillis();
    // 创建检测结果存储数据
    LivenessModel livenessModel = new LivenessModel();
    // 创建检测对象，如果原始数据YUV，转为算法检测的图片BGR
    // TODO: 用户调整旋转角度和是否镜像，手机和开发版需要动态适配
    BDFaceImageInstance rgbInstance = getBdImage(bdFaceImageConfig, bdFaceCheckConfig.darkEnhance);
    livenessModel.setTestBDFaceImageInstanceDuration(System.currentTimeMillis() - startTime);
    onTrack(
        rgbInstance,
        livenessModel,
        new DetectListener() {
```

图3

```
6 usages
private BDFaceImageInstance getBdImage(BDFaceImageConfig bdFaceImageConfig, boolean darkEnhance) {
    BDFaceImageInstance rgbInstance =
        new BDFaceImageInstance(
            bdFaceImageConfig.data,
            bdFaceImageConfig.srcHeight,
            bdFaceImageConfig.srcWidth,
            bdFaceImageConfig.bdFaceImageType,
            bdFaceImageConfig.direction,
            bdFaceImageConfig.mirror);
    BDFaceImageInstance rgbInstanceOne;
    // 判断暗光恢复
    if (darkEnhance) {
        rgbInstanceOne = faceModel.getDark().faceDarkEnhance(rgbInstance);
        rgbInstance.destory();
    } else {
        rgbInstanceOne = rgbInstance;
    }
    return rgbInstanceOne;
}
```

图片暗光优化，此处会返回新的ImageInstance，
所以需要对之前的Image进行内存清理（与bitmap同理）

图4

```
2 usages
private void onTrack(BDFaceImageInstance rgbInstance, LivenessModel livenessModel, DetectListener detectListener) {

    long startDetectTime = System.currentTimeMillis();

    livenessModel.setRgbDetectDuration(System.currentTimeMillis() - startDetectTime);
    // track
    FaceInfo[] faceInfos = null;
    // 多人采用检测，不能跟踪
    if (isMultiIdentify){
        faceInfos = getDetectCheck(rgbInstance);
    }
    else{
        faceInfos = getTrackCheck(rgbInstance);
    }

    // 检测结果判断
    if (faceInfos == null || faceInfos.length == 0) {
        detectListener.onDetectFail();
        return;
    }
```

图5

```
1 usage
private FaceInfo[] getTrackCheck(BDFaceImageInstance rgbInstance) {

    // 快速检测获取人脸信息，仅用于绘制人脸框，详细人脸数据后续获取
    FaceInfo[] faceInfos =
        faceModel
            .getFaceTrack()
            .track(
                BDFaceSDKCommon.DetectType.DETECT_VIS,
                BDFaceSDKCommon.AlignType.BDFACE_ALIGN_TYPE_RGB_FAST,
                rgbInstance);
    return faceInfos;
}
```

track快速获取人脸，FaceInfo为人脸信息，
包括人脸宽高位置关键点和模糊遮挡角度光照等得分
（此接口为快速获取，只会返回人脸宽高位置）

图6

```
        livenessModel,
        new DetectListener() {
            2 usages
            @Override
            public void onDetectSuccess(FaceInfo[] faceInfos, BDFaceImageInstance rgbInstance) {
                // 多帧判断
                if (!frameSelect(faceInfos[0])) {
                    livenessModel.setBdFaceImageInstance(rgbInstance.getImage());
                    if (faceDetectCallBack != null && faceAdoptModel != null) {
                        faceDetectCallBack.onFaceDetectDarwCallback(livenessModel);
                        faceDetectCallBack.onFaceDetectCallback(faceAdoptModel);
                    }
                    rgbInstance.destory();

                    return;
                }
                // 保存人脸特征点
                livenessModel.setLandmarks(faceInfos[0].landmarks);
                // 保存人脸图片
                livenessModel.setBdFaceImageInstance(rgbInstance.getImage());
                // 口罩检测数据
                if (checkMouthMask) {
                    FaceMouthMask mouthMask = getFaceMouthMask();
                    if (mouthMask != null) {
```

注：视频流创建的BDFaceImageInstance需要通过
.getImage才能进获取到图片信息，但内存清理只需要对
原有的对象进行destory即可，清理后get后的不需要清理

FaceSDKManager.java | FaceInfo.class | BDFaceOcclusion.class | FaceRGBGateActivity.java

Decompiled .class file, bytecode version: 51.0 (Java 7)

```
import ...

public class FaceInfo {
    public int faceID;
    public float centerX;
    public float centerY;
    public float width;
    public float height;
    public float angle;
    public float score;
    public float[] landmarks;
    public float yaw;
    public float roll;
    public float pitch;
    public float blurriness;
    public float illum;
    public BDFaceOcclusion occlusion;
    public int age;
    public BDFaceRace race;
    public BDFaceGlasses glasses;
    public BDFaceGender gender;
    Deprecated
```

人脸中心点位置

人脸宽高

人脸得分，越高越接近人脸

人脸左右 平面 上下角度得分

模糊得分

人脸亮度得分

人脸部位遮挡得分

```
BDFaceImageInstance rgbInstance = getBdImage(bdFaceImageConfig, bdFaceCheckConfig.darkEnhance);
livenessModel.setTestBDFaceImageInstanceDuration(System.currentTimeMillis() - startTime);
onTrack(
    rgbInstance,
    livenessModel,
    new DetectListener() {
        2 usages
        @Override
        public void onDetectSuccess(FaceInfo[] faceInfos, BDFaceImageInstance rgbInstance) {
            // 多帧判断
            if (!frameSelect(faceInfos[0])) {
                livenessModel.setBdFaceImageInstance(rgbInstance.getImage());
                if (faceDetectCallBack != null && faceAdoptModel != null) {
                    faceDetectCallBack.onFaceDetectDarwCallback(livenessModel);
                    faceDetectCallBack.onFaceDetectCallback(faceAdoptModel);
                }
                rgbInstance.destory();

                return;
            }
            // 保存人脸特征点
```

```
public void onLivenessCheck(
    final BDFaceImageInstance rgbInstance,
    final BDFaceImageConfig nirBDFaceImageConfig,
    final BDFaceImageConfig depthBDFaceImageConfig,
    final BDFaceCheckConfig bdFaceCheckConfig,
    final LivenessModel livenessModel,
    final long startTime,
    final FaceDetectCallBack faceDetectCallBack,
    final FaceInfo[] fastFaceInfos) {

    if (future2 != null && !future2.isDone()) {
        // 流程结束销毁图片，开始下一帧图片检测，否着内存泄露
        rgbInstance.destory();
        return;
    }

    future2 =
        es2.submit(
            new Runnable() {

                @Override
                public void run() {

                    // 获取BDFaceCheckConfig配置信息
                    if (bdFaceCheckConfig == null) {
                        rgbInstance.destory();

                        return;
                    }
                    onDetect(bdFaceCheckConfig, rgbInstance, fastFaceInfos, livenessModel, new DetectListener() {
```

若该线程在识别中，清理图片缓存结束调用

获取人脸信息

```
2 usages
private void onDetect(
    BDFaceCheckConfig bdFaceCheckConfig,
    BDFaceImageInstance rgbInstance,
    FaceInfo[] fastFaceInfos,
    LivenessModel livenessModel,
    DetectListener detectListener) {

    long accurateTime = System.currentTimeMillis();
    FaceInfo[] faceInfos;

    if (bdFaceCheckConfig != null) {
        bdFaceCheckConfig.bdFaceDetectListConfig.usingQuality = true;
        faceInfos =
            faceModel
                .getFaceDetect()
                .detect(
                    BDFaceSDKCommon.DetectType.DETECT_VIS,
                    BDFaceSDKCommon.AlignType.BDFACE_ALIGN_TYPE_RGB_ACCURATE,
                    rgbInstance,
                    fastFaceInfos,
                    bdFaceCheckConfig.bdFaceDetectListConfig);
    } else {
        faceInfos = faceModel.getFaceDetect().detect(BDFaceSDKCommon.DetectType.DETECT_VIS, rgbInstance);
    }
```

获取人脸详细信息，该接口会返回
模糊角度模糊光照和人脸数据等信息

- 最优人脸

```
// 最优人脸控制
if (!onBestImageCheck(livenessModel, bdFaceCheckConfig, faceDetectCallBack)) {
    livenessModel.setQualityCheck(true);
    livenessModel.clearIdentifyResults();
    rgbInstance.destory();
    if (faceDetectCallBack != null) {
        faceDetectCallBack.onFaceDetectCallback(livenessModel);
    }
    return;
}
```

```
/**
 * 最优人脸控制
 *
 * @param livenessModel
 * @param faceDetectCallBack
 * @return
 */
2 usages
public boolean onBestImageCheck(
    LivenessModel livenessModel, BDFaceCheckConfig bdFaceCheckConfig, FaceDetectCallBack faceDetectCallBack) {
    boolean isBestImageCheck = false;
    if (livenessModel != null && livenessModel.getFaceInfos() != null) {
        FaceInfo[] faceInfos = livenessModel.getFaceInfos();
        int size = faceInfos.length;
        for (int i = 0; i < size; i++) {
            float bestImageScore = faceInfos[i].bestImageScore;
            if (bestImageScore > 0.5) {
                isBestImageCheck = true;
            }
        }
    }
    return isBestImageCheck;
}
```

- 质量检测

```
// 质量检测未通过,销毁BDFaceImageInstance, 结束函数
if (!onQualityCheck(faceInfos, bdFaceCheckConfig.bdQualityConfig, faceDetectCallBack)) {
    livenessModel.setQualityCheck(true);
    livenessModel.clearIdentifyResults();
    rgbInstance.destory();
    if (faceDetectCallBack != null) {
        faceDetectCallBack.onFaceDetectCallback(livenessModel);
    }

    return;
}
```

```
// 角度过滤
if (Math.abs(faceInfo.yaw) > bdQualityConfig.gesture) {
    faceDetectCallBack.onTip( code: -1,  msg: "人脸左右偏转角超出限制");
    checkItem = false;
} else if (Math.abs(faceInfo.roll) > bdQualityConfig.gesture) {
    faceDetectCallBack.onTip( code: -1,  msg: "人脸平行平面内的头部旋转角超出限制");
    checkItem = false;
} else if (Math.abs(faceInfo.pitch) > bdQualityConfig.gesture) {
    faceDetectCallBack.onTip( code: -1,  msg: "人脸上下偏转角超出限制");
    checkItem = false;
}

// 模糊结果过滤
float blur = faceInfo.bluriness;
if (blur > bdQualityConfig.blur) {
    faceDetectCallBack.onTip( code: -1,  msg: "图片模糊");
    checkItem = false;
}

// 光照结果过滤
float illum = faceInfo.illum;
Log.e( tag: "illum",  msg: "illum = " + illum);
if (illum < bdQualityConfig.illum) {
    faceDetectCallBack.onTip( code: -1,  msg: "图片光照不通过");
    checkItem = false;
}
```

```
// 口罩识别不进行质量判断
if (!checkMouthMask) {
    // 遮挡结果过滤
    if (faceInfo.occlusion != null) {
        BDFaceOcclusion occlusion = faceInfo.occlusion;

        if (occlusion.leftEye > bdQualityConfig.leftEye) {
            // 左眼遮挡置信度
            faceDetectCallBack.onTip( code: -1,  msg: "左眼遮挡");
            checkItem = false;
        } else if (occlusion.rightEye > bdQualityConfig.rightEye) {
            // 右眼遮挡置信度
            faceDetectCallBack.onTip( code: -1,  msg: "右眼遮挡");
            checkItem = false;
        } else if (occlusion.nose > bdQualityConfig.nose) {
            // 鼻子遮挡置信度
            faceDetectCallBack.onTip( code: -1,  msg: "鼻子遮挡");
            checkItem = false;
        } else if (occlusion.mouth > bdQualityConfig.mouth) {
            // 嘴巴遮挡置信度
            faceDetectCallBack.onTip( code: -1,  msg: "嘴巴遮挡");
            checkItem = false;
        } else if (occlusion.leftCheek > bdQualityConfig.leftCheek) {
            // 左脸遮挡置信度
            faceDetectCallBack.onTip( code: -1,  msg: "左脸遮挡");
            checkItem = false;
        } else if (occlusion.rightCheek > bdQualityConfig.rightCheek) {
            // 右脸遮挡置信度
            faceDetectCallBack.onTip( code: -1,  msg: "右脸遮挡");
            checkItem = false;
        } else if (occlusion.chin > bdQualityConfig.chinContour) {
            // 下巴遮挡置信度
            faceDetectCallBack.onTip( code: -1,  msg: "下巴遮挡");
        }
    }
}
```

- 质量检测阈值设置

StartActivity.java | FaceAuth.class | FaceRGBGateActivity.java | FaceSDKManager.java | BDQualityConfig.java

```java
public class BDQualityConfig {
    6 usages
    public BDQualityConfig(float blur , float illum , float gesture ,
                           float leftEye , float rightEye ,
                           float nose , float mouth , float leftCheek ,
                           float rightCheek , float chinContour){
        this.blur = blur;
        this.illum = illum;
        this.gesture = gesture;
        this.leftEye = leftEye;
        this.rightEye = rightEye;
        this.nose = nose;
        this.mouth = mouth;
        this.leftCheek = leftCheek;
        this.rightCheek = rightCheek;
        this.chinContour = chinContour;
    }
    // 模糊度设置，默认0.8。取值范围[0~1]，0是最清晰，1是最模糊
    public float blur;
    // 光照设置，默认0.8.取值范围[0~1]，数值越大，光线越强
    public float illum;
    // 姿态阈值
    public float gesture;
    // 左眼被遮挡的阈值，默认0.8
    public float leftEye;
    // 右眼被遮挡的阈值，默认0.8
    public float rightEye;
    // 鼻子被遮挡的阈值，默认0.8
    public float nose;
    // 嘴巴被遮挡的阈值，默认0.8
    public float mouth;
    // 左脸颊被遮挡的阈值，默认0.8
    public float leftCheek;
    // 右脸颊被遮挡的阈值，默认0.8
    public float rightCheek;
    // 下巴被遮挡阈值，默认为0.8
    public float chinContour;
}
```

在每个模块的FaceUtils类中赋值，取BaseConfig中的默认值

StartActivity.java | FaceAuth.class | FaceRGBGateActivity.java | FaceSDKManager.java | BDQualityConfig.java | FaceUtils.java

```java
                    SingleBaseConfig.getBaseConfig().getActiveModel() ,
                bdFaceDetectListConfig , bdQualityConfig , bdLiveConfig
                );
    }

    7 usages
    public BDLiveConfig getBDLiveConfig(){
        return SingleBaseConfig.getBaseConfig().isLivingControl() ?
                new BDLiveConfig(SingleBaseConfig.getBaseConfig().getRgbLiveScore() ,
                SingleBaseConfig.getBaseConfig().getNirLiveScore() ,
                SingleBaseConfig.getBaseConfig().getDepthLiveScore()) : null;
    }

    1 usage
    private BDQualityConfig getBDQualityConfig(){
        return !SingleBaseConfig.getBaseConfig().isQualityControl() ?
                null : new BDQualityConfig(SingleBaseConfig.getBaseConfig().getBlur() ,
                SingleBaseConfig.getBaseConfig().getIllumination() ,
                SingleBaseConfig.getBaseConfig().getGesture() , SingleBaseConfig.getBaseConfig().getLeftEye() ,
                SingleBaseConfig.getBaseConfig().getRightEye() , SingleBaseConfig.getBaseConfig().getNose() ,
                SingleBaseConfig.getBaseConfig().getMouth() , SingleBaseConfig.getBaseConfig().getLeftCheek() ,
                SingleBaseConfig.getBaseConfig().getRightCheek() , SingleBaseConfig.getBaseConfig().getChinContour());
    }
```

- 活体检测

- rgb活体检测

```
// TODO 活体检测
float[] rgbScores = {-1};
BDLiveConfig bdLiveConfig = bdFaceCheckConfig.bdLiveConfig;
boolean isLiveCheck = bdFaceCheckConfig.bdLiveConfig != null;

if (isLiveCheck) {
    long startRgbTime = System.currentTimeMillis();
    rgbScores =
        silentLives(
            rgbInstance,
            BDFaceSDKCommon.LiveType.BDFACE_SILENT_LIVE_TYPE_RGB,
            faceInfos,
            mRgbLiveList,
            bdLiveConfig.rgbLiveScore);
    /* if (rgbScores != null){
        int size = rgbScores.length;
        Log.d("Attend", "score size:" + size);
        for (int i = 0; i < size; i++) {
            Log.d("Attend", "score:" + rgbScores[i]);
        }
    } */

    if (faceInfos.length == 1) {
        livenessModel.setRgbLivenessScore(rgbScores[0]);
    }
    livenessModel.setRgbLivenessScores(rgbScores);
    livenessModel.setRgbLivenessDuration(System.currentTimeMillis() - startRgbTime);
}
```

```
/**
 *  活体检测
 * @param rgbInstance
 * @param type
 * @param faceInfos
 * @param list
 * @param liveScore
 * @return
 */
1 usage
private float[] silentLives(
    BDFaceImageInstance rgbInstance,
    BDFaceSDKCommon.LiveType type,
    FaceInfo[] faceInfos,
    List<Boolean> list,
    float liveScore) {
    float[] scores = {0, 0, 0}; // 最多检测3个人脸
    if (faceInfos != null) {
        synchronized (faceModel.getFaceLive()) {
            Log.e( tag: "test_camera", msg: rgbInstance.getImage().data.length + "开始");
            int size = faceInfos.length;

            for (int i = 0; i < size; i++) {
                scores[i] =
                    faceModel.getFaceLive().silentLive(type, rgbInstance, faceInfos[i].landmarks, liveScore);
            }
        }
    }
    return scores;
}
```

- nirInstance获取方法

```
// TODO nir活体检测
float nirScore = -1;
FaceInfo[] faceInfosIr = null;
BDFaceImageInstance nirInstance = null;
boolean isHaveNirImage = nirBDFaceImageConfig != null && isLiveCheck;
if (isHaveNirImage) {
    // 创建检测对象，如果原始数据YUV-IR，转为算法检测的图片BGR
    // TODO: 用户调整旋转角度和是否镜像，手机和开发版需要动态适配
    long nirInstanceTime = System.currentTimeMillis();
    nirInstance = getBdImage(nirBDFaceImageConfig, darkEnhance: false);

    livenessModel.setBdNirFaceImageInstance(nirInstance.getImage());
    livenessModel.setNirInstanceTime(System.currentTimeMillis() - nirInstanceTime);

    // 避免RGB检测关键点在IR对齐活体稳定，增加红外检测
    long startIrDetectTime = System.currentTimeMillis();
    BDFaceDetectListConf bdFaceDetectListConf = new BDFaceDetectListConf();
    bdFaceDetectListConf.usingDetect = true;
    faceInfosIr =
        faceModel
            .getFaceNirDetect()
            .detect(
                BDFaceSDKCommon.DetectType.DETECT_NIR,
                BDFaceSDKCommon.AlignType.BDFACE_ALIGN_TYPE_NIR_ACCURATE,
                nirInstance,
                faceInfos: null,
                bdFaceDetectListConf);
    livenessModel.setIrLivenessDuration(
        System.currentTimeMillis() - startIrDetectTime);
    //                  LogUtils.e(TIME_TAG, "detect ir time = " + livenessModel.getIrLivenessDuration());
    if (faceInfosIr != null && faceInfosIr.length > 0) {
        FaceInfo faceInfoIr = faceInfosIr[0];
        nirScore =
            silentLive(
                nirInstance,
                BDFaceSDKCommon.LiveType.BDFACE_SILENT_LIVE_TYPE_NIR,
                faceInfoIr.landmarks,
                mNirLiveList,
```

- getBdImage方法

```
6 usages
private BDFaceImageInstance getBdImage(BDFaceImageConfig bdFaceImageConfig, boolean darkEnhance) {
    BDFaceImageInstance rgbInstance =
        new BDFaceImageInstance(
            bdFaceImageConfig.data,
            bdFaceImageConfig.srcHeight,
            bdFaceImageConfig.srcWidth,
            bdFaceImageConfig.bdFaceImageType,
            bdFaceImageConfig.direction,
            bdFaceImageConfig.mirror);
    BDFaceImageInstance rgbInstanceOne;
    // 判断暗光恢复
    if (darkEnhance) {
        rgbInstanceOne = faceModel.getDark().faceDarkEnhance(rgbInstance);
        rgbInstance.destory();
    } else {
        rgbInstanceOne = rgbInstance;
    }
    return rgbInstanceOne;
}
```

- nir活体得分获取方法

```
1 usage
private float silentLive(
    BDFaceImageInstance rgbInstance,
    BDFaceSDKCommon.LiveType type,
    float[] landmarks,
    List<Boolean> list,
    float liveScore) {
    float score = 0;
    if (landmarks != null) {
        synchronized (faceModel.getFaceLive()) {
            Log.e( tag: "test_camera",  msg: rgbInstance.getImage().data.length + "开始");
            score = faceModel.getFaceLive().silentLive(type, rgbInstance, landmarks, liveScore);
            Log.e( tag: "test_camera",  msg: "活体结束");
        }
        list.add(score > liveScore);
    }
    while (list.size() > 6) {
        list.remove( index: 0);
    }
    if (list.size() > 2) {
        int rgbSum = 0;
        for (Boolean b : list) {
            if (b) {
                rgbSum++;
            }
        }
        if (1.0 * rgbSum / list.size() > 0.6) {
            if (score < liveScore) {
                score = liveScore + (1 - liveScore) * new Random().nextFloat();
            }
        } else {
            if (score > liveScore) {
                score = new Random().nextFloat() * liveScore;
            }
        }
    }
    return score;
}
```

- depth活体检测和活体得分获取方法

```
// TODO depth活体检测
float depthScore = -1;
boolean isHaveDepthImage = depthBDFaceImageConfig != null && isLiveCheck;
if (depthBDFaceImageConfig != null) {
    // TODO: 用户调整旋转角度和是否镜像，适配Atlas 镜头，目前宽和高400*640，其他摄像头需要动态调整,人脸72 个关键点x 坐标向左移动80个
    float[] depthLandmark = new float[faceInfos[0].landmarks.length];
    BDFaceImageInstance depthInstance;
    if (bdFaceCheckConfig.cameraType == 1) {
        System.arraycopy(
            faceInfos[0].landmarks,
            srcPos: 0,
            depthLandmark,
            destPos: 0,
            faceInfos[0].landmarks.length);
        for (int i = 0; i < 144; i = i + 2) {
            depthLandmark[i] -= 80;
        }
    } else {
        depthLandmark = faceInfos[0].landmarks;
    }
    depthInstance = getBdImage(depthBDFaceImageConfig, darkEnhance: false);
    livenessModel.setBdDepthFaceImageInstance(depthInstance.getImage());
    // 创建检测对象，如果原始数据Depth
    long startDepthTime = System.currentTimeMillis();
    depthScore =
        faceModel
            .getFaceLive()
            .silentLive(
                BDFaceSDKCommon.LiveType.BDFACE_SILENT_LIVE_TYPE_DEPTH,
                depthInstance,
                depthLandmark);
    livenessModel.setDepthLivenessScore(depthScore);
    livenessModel.setDepthtLivenessDuration(
        System.currentTimeMillis() - startDepthTime);
    //                    LogUtils.e(TIME_TAG, "live depth time = " + livenessModel.getDepthtLivenessDuration());
    depthInstance.destory();
}
```

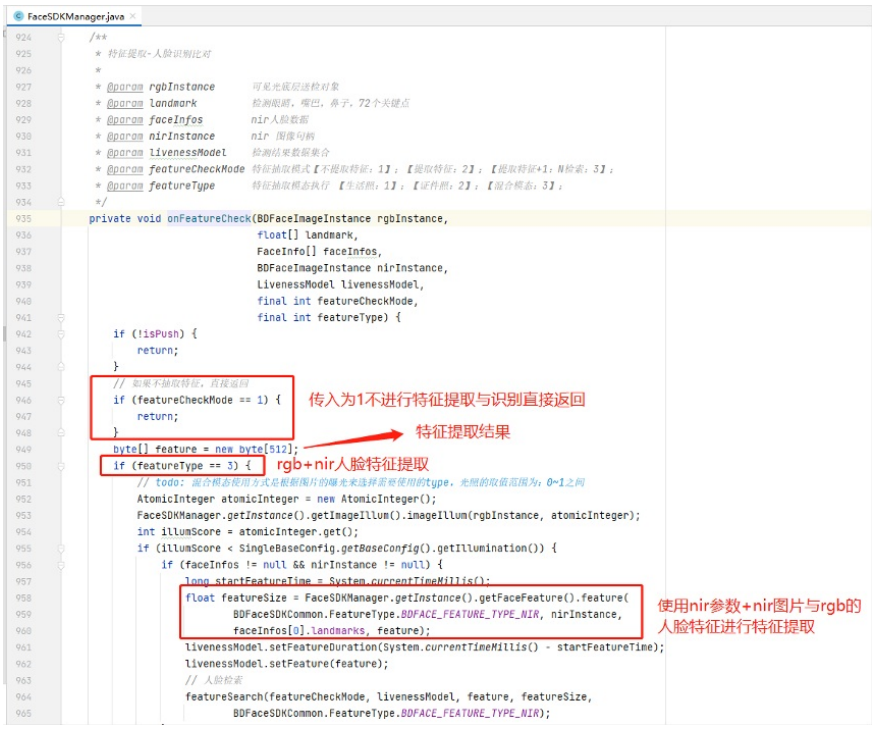
depth活体检测

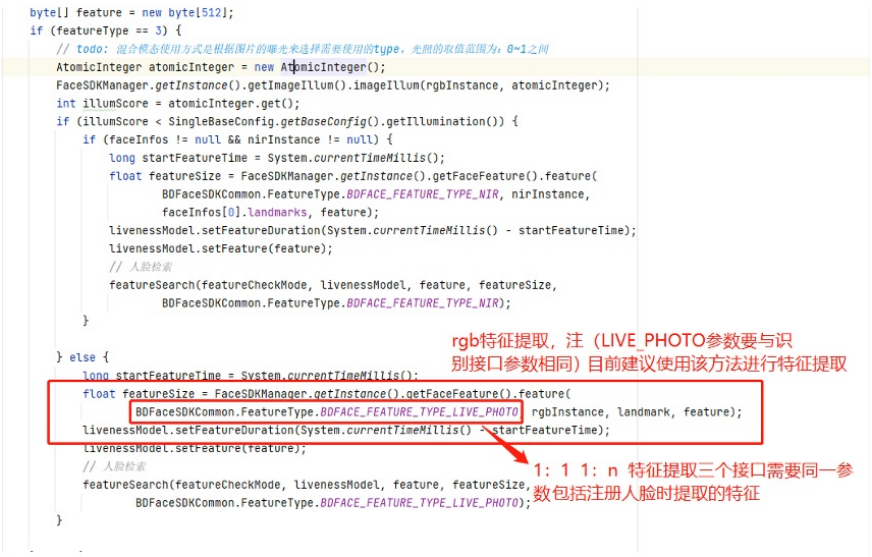
StartActivity.java | FaceAuth.class | FaceRGBGateActivity.java | FaceSDKManager.java | FaceLive.class

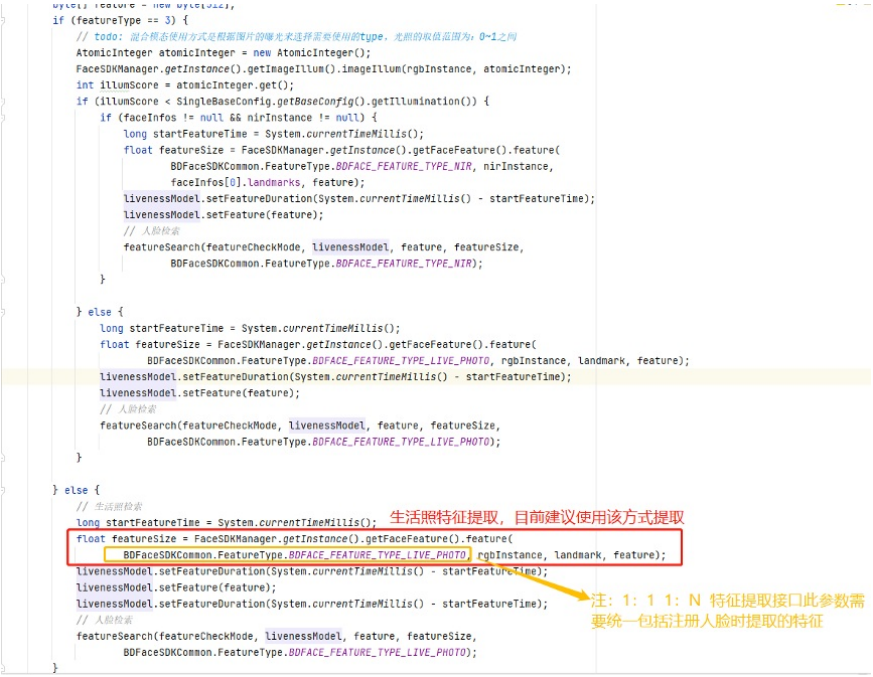
Decompiled .class file, bytecode version: 51.0 (Java 7) Download Sources Choose So

```
                    } else {
                        callback.onResponse( i: 0, s: "活体模型加载成功");
                    }

                    Log.e( tag: "bdface", msg: "FaceLive initModel");
                }
            }
        };
        FaceQueue.getInstance().execute(runnable);
    }

    8 usages
    public float silentLive(BDFaceSDKCommon.LiveType type, BDFaceImageInstance bdFaceImageInstance, float[] landmarks) {
        if (type != null && bdFaceImageInstance != null && landmarks != null) {
            long instanceIndex = this.bdFaceInstance.getIndex();
            return instanceIndex == 0L ? -1.0F : this.nativeSilentLive(instanceIndex, type.ordinal(), bdFaceImageInstance, landmarks, var6: 0.8F);
        } else {
            Log.v(TAG, msg: "Parameter is null");
            return -1.0F;
        }
    }
```

- 识别


FaceSDKManager.java
/**
* 特征提取-人脸识别比对
*
* @param rgbInstance 可见光底层送检对象
* @param landmark 检测眼睛，嘴巴，鼻子，72个关键点
* @param faceInfos nir人脸数据
* @param nirInstance nir 图像句柄
* @param livenessModel 检测结果数据集合
* @param featureCheckMode 特征抽取模式【不提取特征：1】；【提取特征：2】；【提取特征+1：N检索：3】；
* @param featureType 特征抽取模态执行 【生活照：1】；【证件照：2】；【混合模态：3】；
*/
private void onFeatureCheck(BDFaceImageInstance rgbInstance,
float[] landmark,
FaceInfo[] faceInfos,
BDFaceImageInstance nirInstance,
LivenessModel livenessModel,
final int featureCheckMode,
final int featureType) {
if (!isPush) {
return;
}
// 如果不抽取特征，直接返回
if (featureCheckMode == 1) {
return;
}
byte[] feature = new byte[512];
if (featureType == 3) {
// todo: 混合模态使用方式是根据图片的曝光来选择需要使用的type，光照的取值范围为：0~1之间
AtomicInteger atomicInteger = new AtomicInteger();
FaceSDKManager.getInstance().getImageIllum().imageIllum(rgbInstance, atomicInteger);
int illumScore = atomicInteger.get();
if (illumScore < SingleBaseConfig.getBaseConfig().getIllumination()) {
if (faceInfos != null && nirInstance != null) {
long startFeatureTime = System.currentTimeMillis();
float featureSize = FaceSDKManager.getInstance().getFaceFeature().feature(
BDFaceSDKCommon.FeatureType.BDFACE_FEATURE_TYPE_NIR, nirInstance,
faceInfos[0].landmarks, feature);
livenessModel.setFeatureDuration(System.currentTimeMillis() - startFeatureTime);
livenessModel.setFeature(feature);
// 人脸检索
featureSearch(featureCheckMode, livenessModel, feature, featureSize,
BDFaceSDKCommon.FeatureType.BDFACE_FEATURE_TYPE_NIR);
传入为1不进行特征提取与识别直接返回
特征提取结果
rgb+nir人脸特征提取
使用nir参数+nir图片与rgb的人脸特征进行特征提取


byte[] feature = new byte[512];
if (featureType == 3) {
// todo: 混合模态使用方式是根据图片的曝光来选择需要使用的type，光照的取值范围为：0~1之间
AtomicInteger atomicInteger = new AtomicInteger();
FaceSDKManager.getInstance().getImageIllum().imageIllum(rgbInstance, atomicInteger);
int illumScore = atomicInteger.get();
if (illumScore < SingleBaseConfig.getBaseConfig().getIllumination()) {
if (faceInfos != null && nirInstance != null) {
long startFeatureTime = System.currentTimeMillis();
float featureSize = FaceSDKManager.getInstance().getFaceFeature().feature(
BDFaceSDKCommon.FeatureType.BDFACE_FEATURE_TYPE_NIR, nirInstance,
faceInfos[0].landmarks, feature);
livenessModel.setFeatureDuration(System.currentTimeMillis() - startFeatureTime);
livenessModel.setFeature(feature);
// 人脸检索
featureSearch(featureCheckMode, livenessModel, feature, featureSize,
BDFaceSDKCommon.FeatureType.BDFACE_FEATURE_TYPE_NIR);
}
} else {
long startFeatureTime = System.currentTimeMillis();
float featureSize = FaceSDKManager.getInstance().getFaceFeature().feature(
BDFaceSDKCommon.FeatureType.BDFACE_FEATURE_TYPE_LIVE_PHOTO, rgbInstance, landmark, feature);
livenessModel.setFeatureDuration(System.currentTimeMillis() - startFeatureTime);
livenessModel.setFeature(feature);
// 人脸检索
featureSearch(featureCheckMode, livenessModel, feature, featureSize,
BDFaceSDKCommon.FeatureType.BDFACE_FEATURE_TYPE_LIVE_PHOTO);
}
rgb特征提取，注（LIVE_PHOTO参数要与识别接口参数相同）目前建议使用该方法进行特征提取
1：1 1：n 特征提取三个接口需要同一参数包括注册人脸时提取的特征


if (featureType == 3) {
// todo: 混合模态使用方式是根据图片的曝光来选择需要使用的type，光照的取值范围为：0~1之间
AtomicInteger atomicInteger = new AtomicInteger();
FaceSDKManager.getInstance().getImageIllum().imageIllum(rgbInstance, atomicInteger);
int illumScore = atomicInteger.get();
if (illumScore < SingleBaseConfig.getBaseConfig().getIllumination()) {
if (faceInfos != null && nirInstance != null) {
long startFeatureTime = System.currentTimeMillis();
float featureSize = FaceSDKManager.getInstance().getFaceFeature().feature(
BDFaceSDKCommon.FeatureType.BDFACE_FEATURE_TYPE_NIR, nirInstance,
faceInfos[0].landmarks, feature);
livenessModel.setFeatureDuration(System.currentTimeMillis() - startFeatureTime);
livenessModel.setFeature(feature);
// 人脸检索
featureSearch(featureCheckMode, livenessModel, feature, featureSize,
BDFaceSDKCommon.FeatureType.BDFACE_FEATURE_TYPE_NIR);
}
} else {
long startFeatureTime = System.currentTimeMillis();
float featureSize = FaceSDKManager.getInstance().getFaceFeature().feature(
BDFaceSDKCommon.FeatureType.BDFACE_FEATURE_TYPE_LIVE_PHOTO, rgbInstance, landmark, feature);
livenessModel.setFeatureDuration(System.currentTimeMillis() - startFeatureTime);
livenessModel.setFeature(feature);
// 人脸检索
featureSearch(featureCheckMode, livenessModel, feature, featureSize,
BDFaceSDKCommon.FeatureType.BDFACE_FEATURE_TYPE_LIVE_PHOTO);
}
} else {
// 生活照检索
long startFeatureTime = System.currentTimeMillis();
float featureSize = FaceSDKManager.getInstance().getFaceFeature().feature(
BDFaceSDKCommon.FeatureType.BDFACE_FEATURE_TYPE_LIVE_PHOTO, rgbInstance, landmark, feature);
livenessModel.setFeatureDuration(System.currentTimeMillis() - startFeatureTime);
livenessModel.setFeature(feature);
livenessModel.setFeatureDuration(System.currentTimeMillis() - startFeatureTime);
// 人脸检索
featureSearch(featureCheckMode, livenessModel, feature, featureSize,
BDFaceSDKCommon.FeatureType.BDFACE_FEATURE_TYPE_LIVE_PHOTO);
}
生活照特征提取，目前建议使用该方式提取
注：1：1 1：N 特征提取接口此参数需要统一包括注册人脸时提取的特征

- 1：n接口

```
// 如果只提去特征，不做检索，此处返回

if (featureCheckMode == 2) {
    livenessModel.setFeatureCode(featureSize);
    return;
}

// 如果提取特征+检索，调用search 方法
if (featureSize == FEATURE_SIZE / 4) {
    long startFeature = System.currentTimeMillis();
    // 特征提取成功

    // TODO 阈值可以根据不同模型调整
    if (featureCheckMode == 3) {

        List<? extends Feature> featureResult =
                faceModel.getFaceSearch().search(type, bdFaceCheckConfig.scoreThreshold, topNum: 1, feature, isPercent: false);

        // TODO 返回top num = 1 个数据集合，此处可以任意设置，会返回比对从大到小排序的num 个数据集合
        if (featureResult != null && featureResult.size() > 0) {

            // 获取第一个数据
            Feature topFeature = featureResult.get(0);
            // 判断第一个阈值是否大于设定阈值，如果大于，检索成功
            threholdScore = bdFaceCheckConfig.scoreThreshold;
            if (topFeature != null && topFeature.getScore() > threholdScore) {
                // 当前featureEntity 只有id+feature 索引，在数据库中查到完整信息
```

1:n识别接口，会以识别得分从高到底排序返回
数量设置为1默认只返回得分最高的人脸

```
        // TODO 返回top num = 1 个数据集合，此处可以任意设置，会返回比对从大到小排序的num 个数据集合
        if (featureResult != null && featureResult.size() > 0) {

            // 获取第一个数据
            Feature topFeature = featureResult.get(0);
            // 判断第一个阈值是否大于设定阈值，如果大于，检索成功
            threholdScore = bdFaceCheckConfig.scoreThreshold;
            if (topFeature != null && topFeature.getScore() > threholdScore) {
                // 当前featureEntity 只有id+feature 索引，在数据库中查到完整信息

                User user = FaceApi.getInstance().getUserListById(topFeature.getId());
                if (user != null) {
                    IdentifyResult idResult = new IdentifyResult(user, index, topFeature.getScore());
                    // Log.d("Attend", "add user:" + user.getUserInfo() + " index:" + index);
                    livenessModel.addIdentifyResult(idResult);
                    livenessModel.setUser(user);
                    livenessModel.setFeatureScore(topFeature.getScore());

                    setFail(livenessModel);
                } else {
                    IdentifyResult idResult = new IdentifyResult(user, index, topFeature.getScore());
                    // Log.d("Attend", "add user222:" + user.getUserInfo() + " index:" + index);
                    setFail(livenessModel);
                }
            } else {
                setFail(livenessModel);
            }
        } else {
            // Log.d("Attend", "featureSearchs 333333: " + index + " featureResult:0");
            setFail(livenessModel);
        }
```

识别得分最高的人脸数据，该接口会返回人脸
的id得分和特征值，id为注册人脸放到内存时的id

通过ID查找对应用户

设置得分

1.12 人脸识别SDK 主页面UI布局修改指引

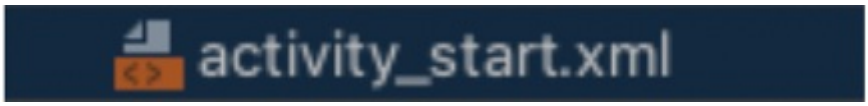
问题：

对识别SDK从启动——鉴权——主页面——具体模块内部，各UI对应布局修改地址进行梳理

解决：

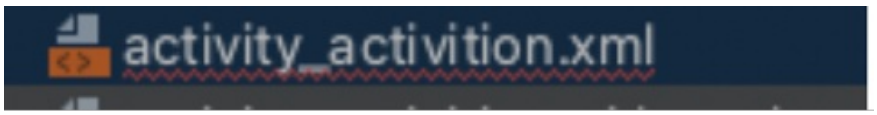
1.启动页面

启动页面对应UI布局修改路径： activity_start.xml



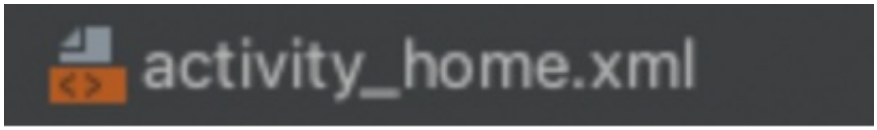
2.激活授权页面

激活授权页面对应UI布局修改路径：activity_activition.xml



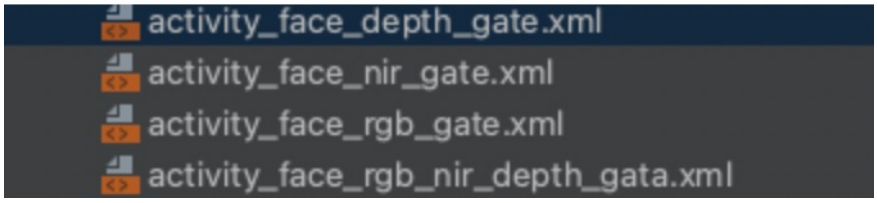
3.主页面

主页面对应UI布局修改路径：activity_home.xml



4.闸机模式页面

闸机模式页面对应UI布局修改路径：（根据活体检测的单/双目形式跳转对应逻辑）



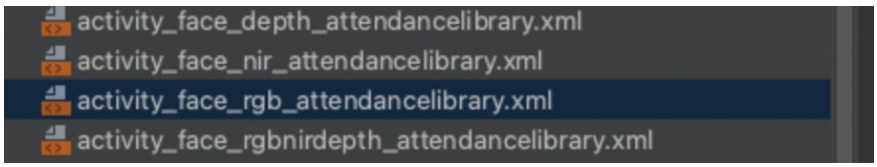
（注：若出现摄像头角度旋转或检测不出人脸点击右上角设置按钮-镜头设置-调整人脸检测角度与视频流回显角度后再次测试）


预览模式
开发模式
设置
人脸检测
质量检测
开启
活体检测
开启
人脸识别
图像优化
镜头设置
日志设置
开启
版本信息
有效期至2037-01-01


镜头相关设置
人脸检测图像设置
视频流回显图像设置
人脸检测框镜像
关闭
如果出现人脸检测不到或摄像头角度不正确
先进入这里修改人脸与回显角度后进行测试

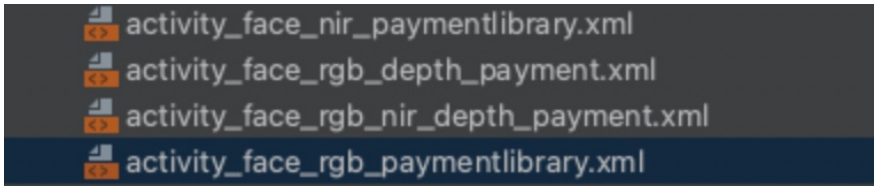
5.考勤模式页面

考勤模式页面对应UI布局修改路径：（根据活体检测的单/双目形式跳转对应逻辑）



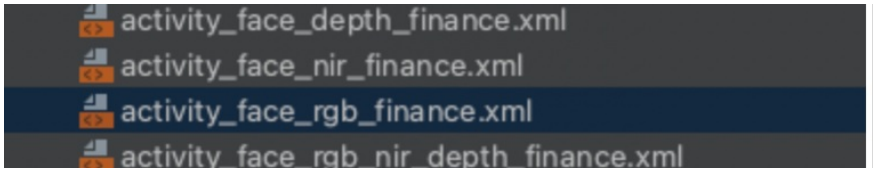
6.支付模式

支付模式页面对应UI布局修改路径：（根据活体检测的单/双目形式跳转对应逻辑）

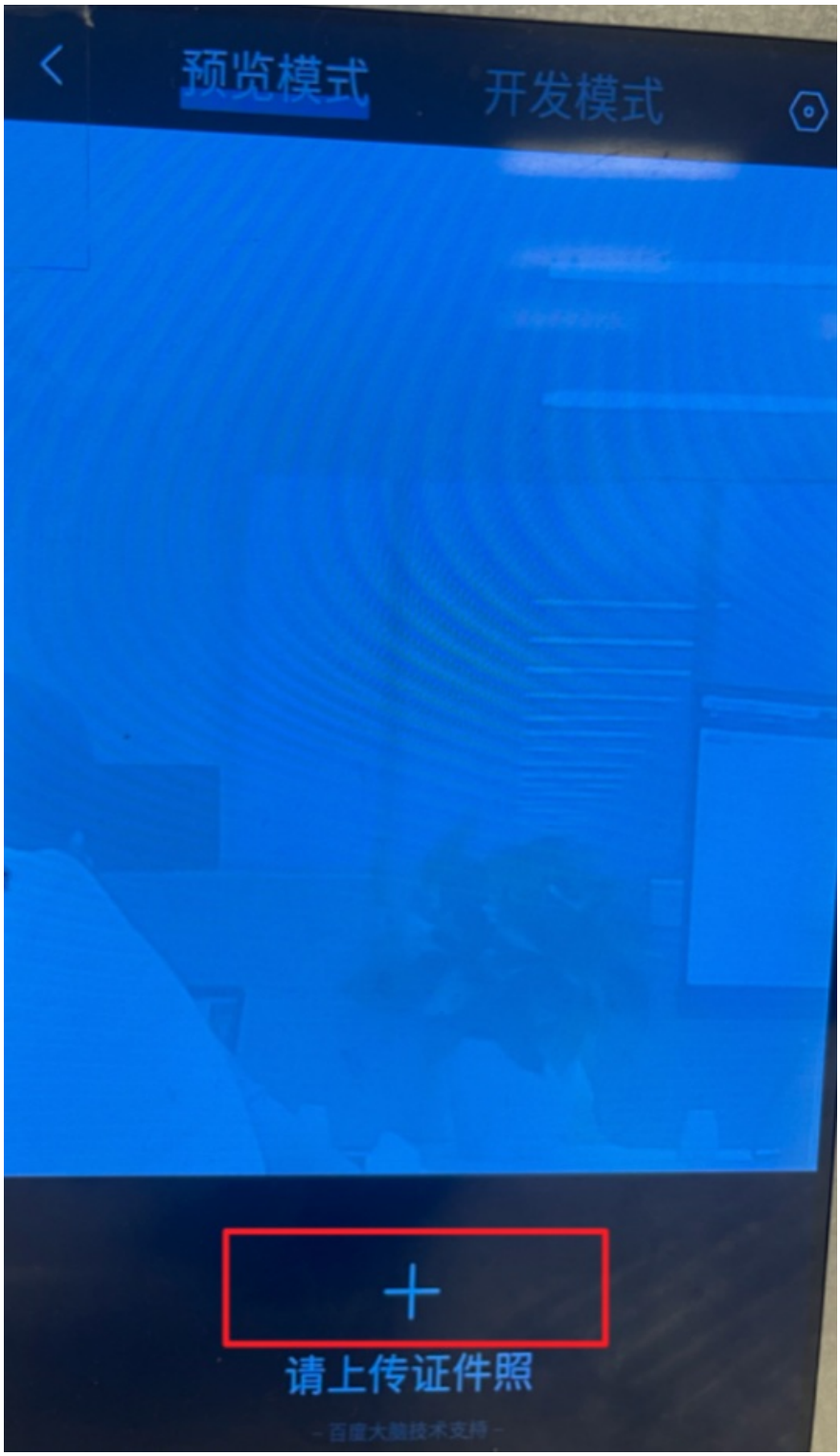


7.金融活检

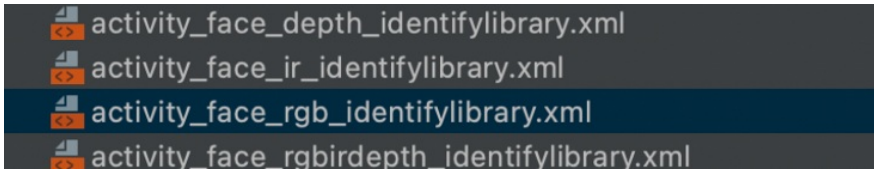
金融活检页面对应UI布局修改路径：（根据活体检测的单/双目形式跳转对应逻辑）



8.人证核验



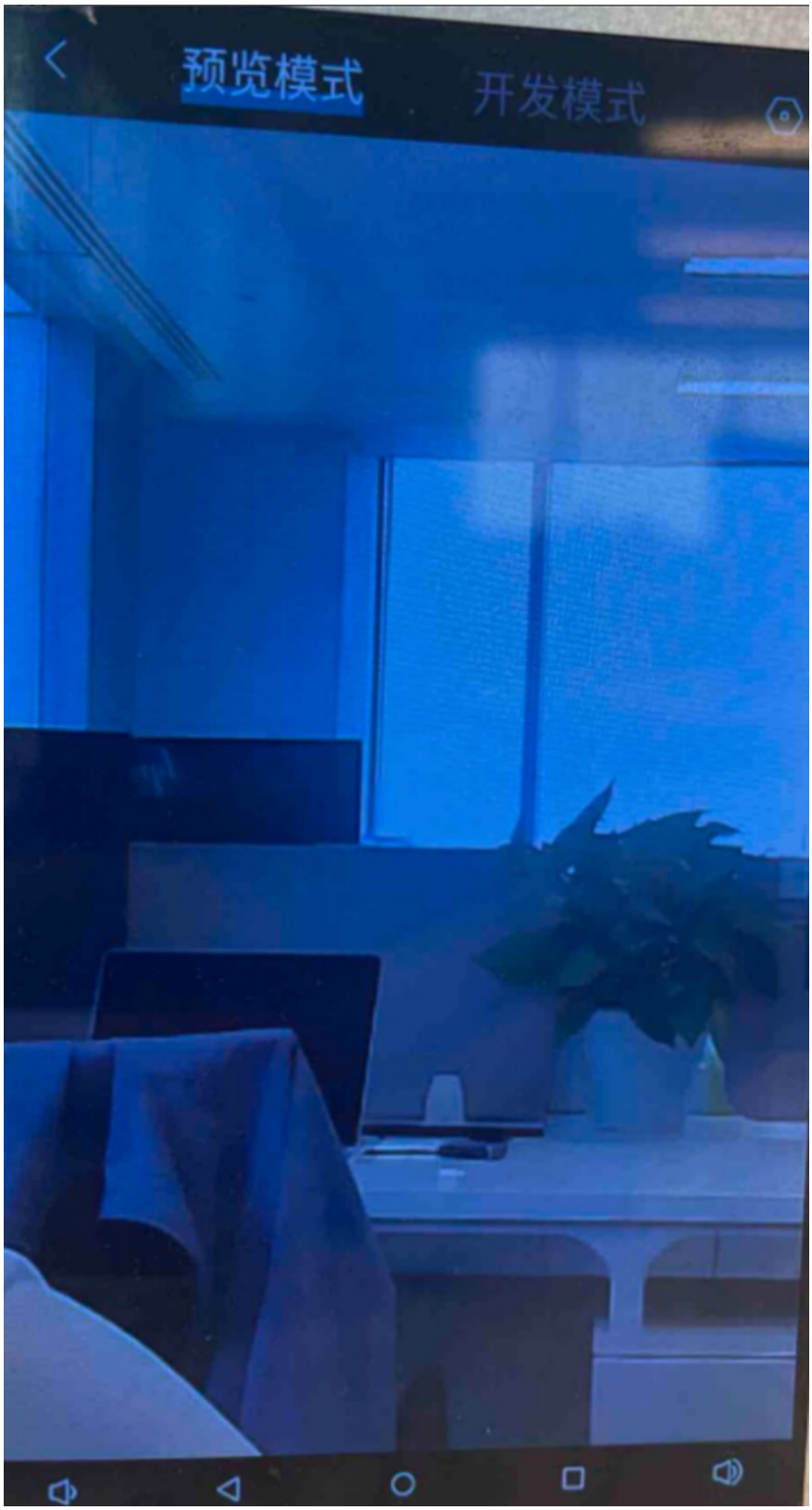
人证核验页面对应UI布局修改路径：（根据活体检测的单/双目形式跳转对应逻辑）



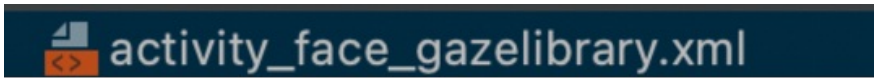
9.驾驶行为分析

驾驶行为分析页面对应UI布局修改路径：activity_nir_drivermonitor.xml

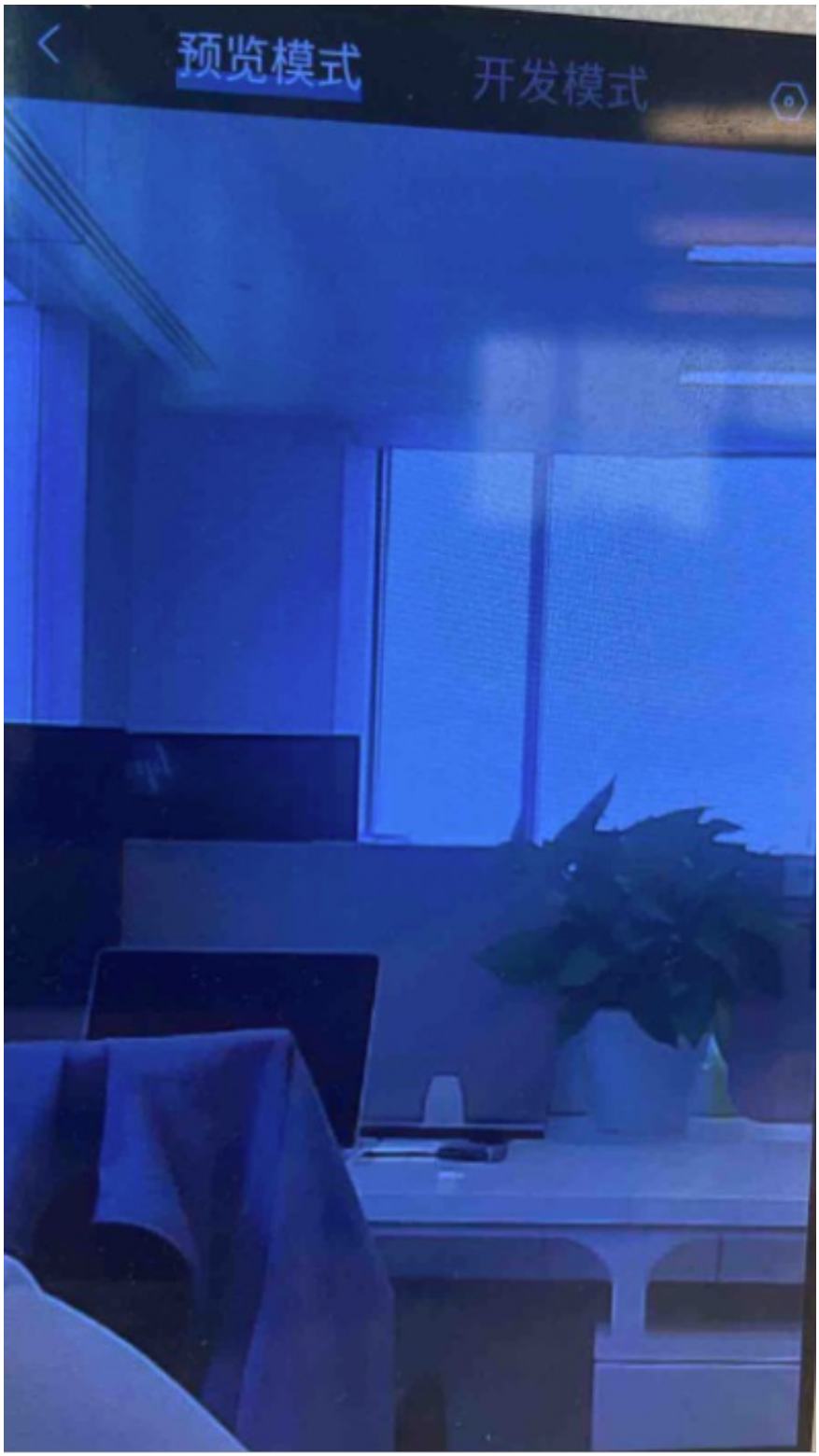
10.注意力检测



注意力检测页面对应UI布局修改路径：activity_face_gazelibrary.xml



11.属性模式

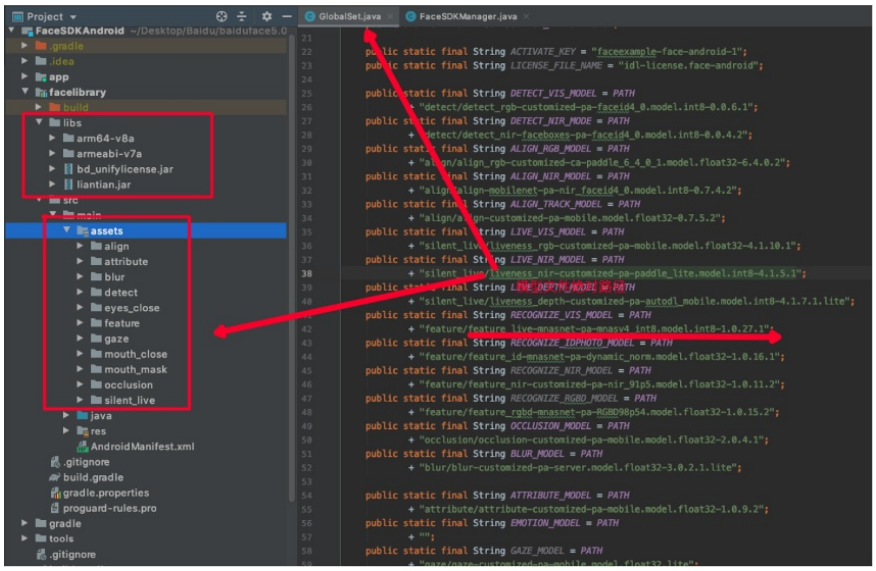


属性检测页面对应UI布局修改路径：activity_face_rgb_attribute.xml

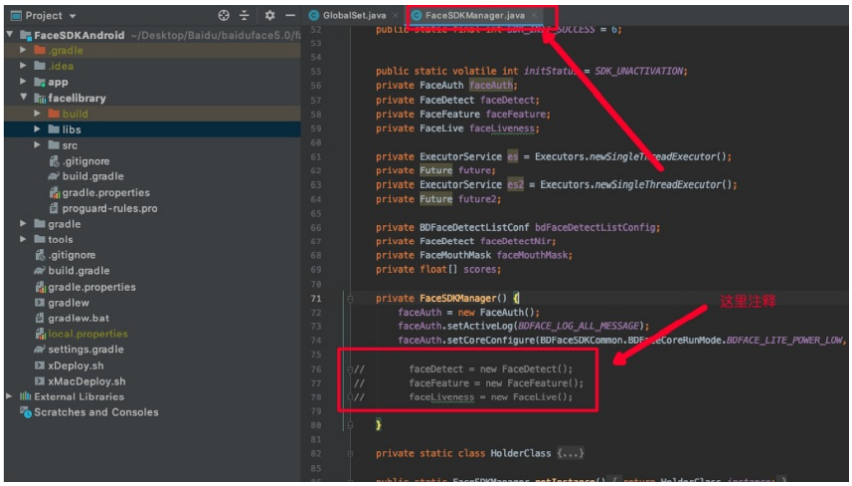
1.13 升级指导

注意（更新版本要刷新人脸底库，重新导入人脸）

1.13.1 3.v – 4.v



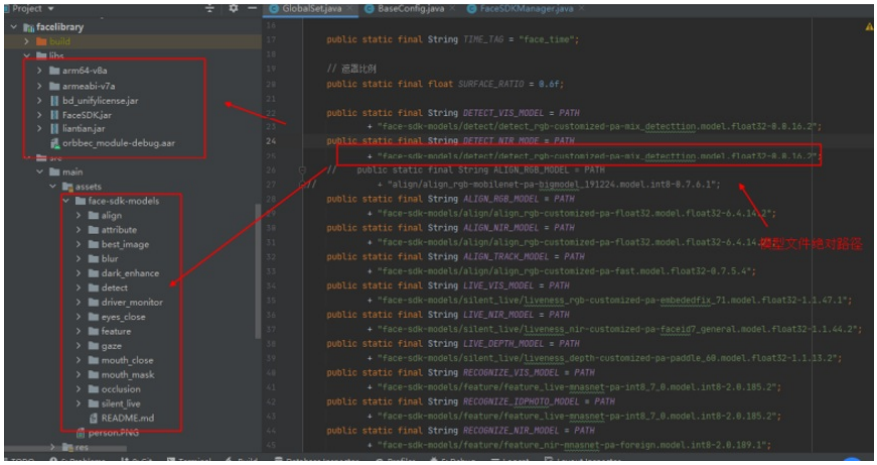
1. libs 下所有 jar 和文件替换成 4.0 的 2.assets 下所有模型文件替换成 4.0 的, GlobalSet 模型路径要对应到 assets 下相应的文件夹



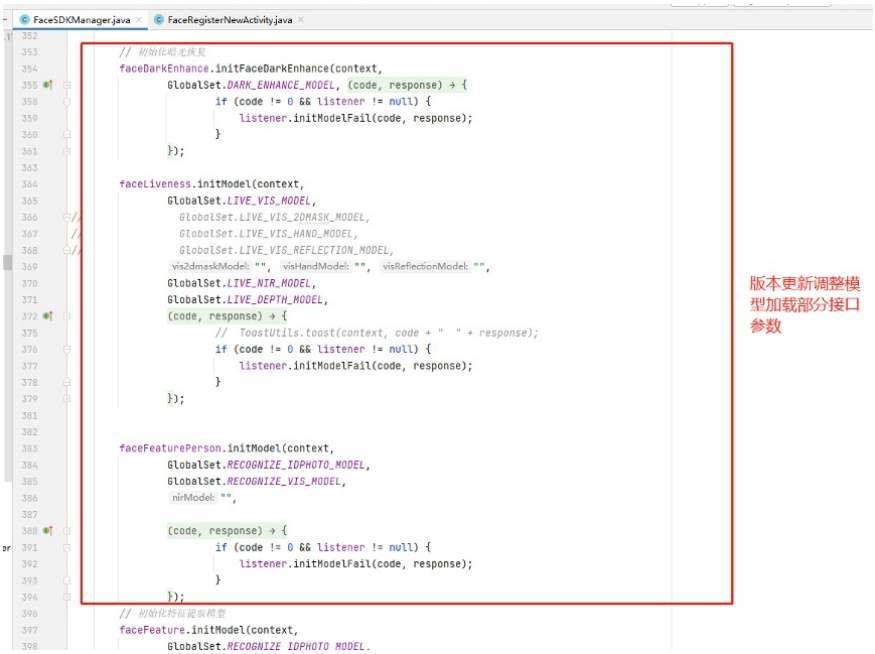
1.13.2 4.v – 5.v、5.v – 6.v、6.v – 7.v

（更新版本要刷新人脸底库，重新导入人脸）

4.0到7.0升级同理，替换sdk与模型路径文件

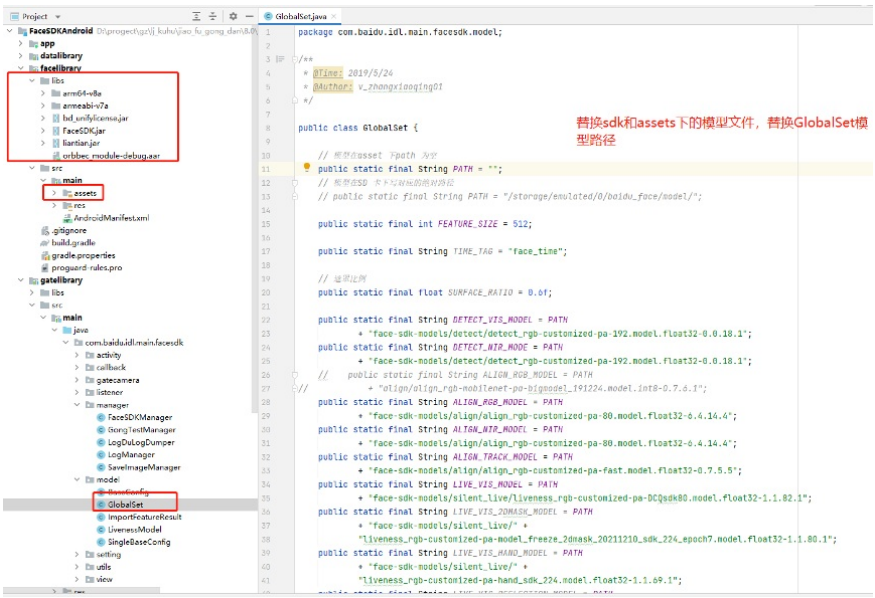


调整模型加载部分接口逻辑

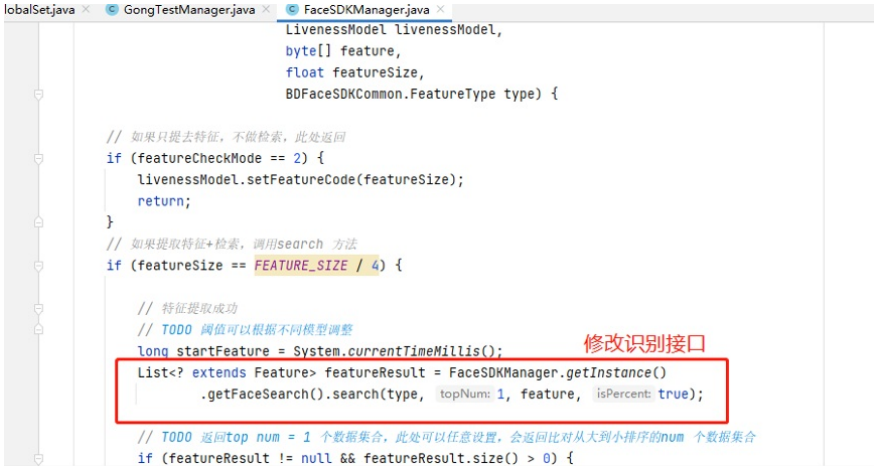


1.13.3 7.v – 8.v

（更新版本要刷新人脸底库，重新导入人脸）



添加最新识别对象



2、接口说明

2.1 FaceAuth鉴权接口

2.1.1 鉴权-在线鉴权 说明：用户通过申请授权码，在线授权，激活设备

```
void initLicenseOnLine(final Context context, final String licenseKey, final AuthCallback callback)
```

参数名	含义
context	当前上下文
licenseKey	AIPE 鉴权码
callback	鉴权结果 void onResponse(int code, String response) code 0：成功；code 1 加载失败 response 结果信息

2.1.2 鉴权-离线授权 说明：用户申请鉴权文件，放在SD 卡下，点击按钮直接鉴权

```
void initLicenseOffLine(final Context context, final Callback callback)
```

参数名	含义
context	当前上下文
callback	鉴权结果 void onResponse(int code, String response) code 0：成功；code 1 加载失败 response 结果信息

2.1.3 鉴权-在线按应用批量授权 说明：用户通过申请在线licenseID，不需要输入任何信息，直接网络请求获取鉴权文件

```
void initLicenseBatchLine(final Context context, final String licenseKey, final Callback callback)
```

参数名	含义
context	当前上下文
licenseKey	鉴权文件Key
callback	鉴权结果 void onResponse(int code, String response) code 0：成功；code 1 加载失败 response 结果信息

2.1.4 开启底层Log输出

说明：用于Debug时候输出Log详细信息

```
void setActiveLog(BDFaceSDKCommon.BDFaceLogInfo logInfo, int isLog)
```

参数名	含义
BDFaceLogInfo	底层log 打印 BDFACE_LOG_ERROR_MESSAGE // 打印输出错误日志 BDFACE_LOG_VALUE_MESSAGE // 打印输出值日志 BDFACE_LOG_TYPE_PERF // 打印性能日志 BDFACE_LOG_TYPE_ALL // 打印全部日志 BDFACE_LOG_TYPE_DEBUG // 打印debug日志
isLog	0：不输出日志；1：输出日志

2.1.5 设置核数

说明：根据开发板类型，设置加速对CPU核数依赖，调整参数，提高性能

```
setCoreConfigure(BDFaceSDKCommon.BDFaceCoreRunMode runMode, int coreNum)
```

参数名	含义
runMode	推荐使用0， 1， 3；如果有需要绑核的话，用0和1；如果不需要绑核，系统自动调度的话，用3 BDFACE_LITE_POWER_HIGH=0, 绑定大核运行模式。如果ARM CPU支持big.LITTLE，则优先使用并绑定Big cluster。如果设置的线程数大于大核数量，则会将线程数自动缩放到大核数量。如果系统不存在大核或者在一些手机的低电量情况下会出现绑核失败，如果失败则进入不绑核模式。 BDFACE_LITE_POWER_LOW=1,绑定小核运行模式。如果ARM CPU支持big.LITTLE，则优先使用并绑定Little cluster。如果设置的线程数大于小核数量，则会将线程数自动缩放到小核数量。如果找不到小核，则自动进入不绑核模式。 BDFACE_LITE_POWER_FULL=2,大小核混用模式。线程数可以大于大核数量。当线程数大于核心数量时，则会自动将线程数缩放到核心数量。 BDFACE_LITE_POWER_NO_BIND=3,不绑核运行模式（推荐）。系统根据负载自动调度任务到空闲的CPU核心上。 BDFACE_LITE_POWER_RAND_HIGH=4,轮流绑定大核模式。如果Big cluster有多个核心，则每预测10次后切换绑定到下一个核心。 BDFACE_LITE_POWER_RAND_LOW=5,轮流绑定小核模式。如果Little cluster有多个核心，则每预测10次后切换绑定到下一个核心。
coreNum	根据如下命令，查看cpu 核数，选择线程数 adb shell cat /proc/cpuinfo

2.1.6 Code返回值

说明: 根据code返回值来判断返回错误信息

参数名	含义
code	code == 0 成功； code == 1 context 为null； code == -1 非法的参数； code == -2 内存分配失败； code == -3 实例对象为空； code == -4 模型内容为空；code == -5 不支持的能力类型； code == -6 不支持预测类型；code == -7 预测库对象创建失败； code == -8 预测库初始化失败；code == -9 图像数据为空； code == -10 人脸能力初始化失败；code == -11 能力未加载 ； code == -12 人脸能力已加载；code == -13 未授权； code == -14 人脸能力运行异常 ；code == -15 不支持的图像类型； code == -16 图像转换失败；

2.2 FaceDetect 检测接口

构造方法：

```
 // 无参构造调用（执行默认创建的instance ）
public FaceDetect()
// 有参构造调用（执行自己创建的instance ）
public FaceDetect(BDFaceInstance thisBdFaceInstance)
```

参数名	含义
thisBdFaceInstance	绑定指定的BDFaceInstance实例，否则使用默认的BDFaceInstance实例

2.2.1 可见光检测对齐模型加载

说明：检测对齐模型加载，支持可见光模型

```
 void initModel(final Context context, final String visModel,final String nirModel,final String alignModel, final Callback callback)
```

参数名	含义
context	上下文context
visModel	可见光图片检测模型
nirModel	红外图片检测模型（非必要参数，可以为空）
alignModel	对齐类型
callback	模型加载结果 void onResponse(int code, String response) code：请参照此文档 1.1.6 code返回值 response 结果信息

2.2.2 检测对齐模型加载

说明：检测对齐模型加载，支持可见光、近红外检测模型

```
public void initModel(final Context context,final String detectModel,final String alignModel,final BDFaceSDKCommon.DetectType detectType,final BDFaceSDKCommon.AlignType alignType,final Callback callback)
```

参数名	含义
context	上下文context
detectModel	检测模型
alignModel	对齐模型
detectType	检测类型 DETECT_VIS 为可见光； DETECT_NIR为近红外
alignType	对齐类型 BDFACE_ALIGN_TYPE_RGB_ACCURATE 为可见光对齐类型; BDFACE_ALIGN_TYPE_NIR_ACCURATE 为 近红外对齐类型
callback	模型加载结果 void onResponse(int code, String response) code：请参照此文档 1.1.6 code返回值 response 结果信息

2.2.3 质量检测模型加载

说明：质量检测模型加载，判断人脸遮挡信息，光照信息，模糊信息，模型包含模糊模型，遮挡信息，作用于质量检测接口

```
void initQuality(final Context context, final String blurModel, final String occlurModel, final Callback callback)
```

参数名	含义
context	上下文context
blurModel	模糊检测模型
occlurModel	遮挡检测模型
callback	鉴权结果 void onResponse(int code, String response) code 0：成功；code 1 加载失败 response 结果信息

2.2.4 属性情绪模型加载

说明：人脸属性（年龄，性别，戴眼镜等），情绪（喜怒哀乐）模型初始化

```
void initAttrbute(final Context context, final String atttibuteModel, final Callback callback)
```

参数名	含义
context	上下文context
atttibuteModel	属性检测模型
emotionModel	7种情绪检测模型
callback	鉴权结果 void onResponse(int code, String response) code 0：成功；code 1 加载失败 response 结果信息

2.2.5 眼睛闭合，嘴巴闭合模型加载

说明：人脸眼睛闭合，嘴巴闭合模型初始化

```
void initFaceClose(final Context context,
final String eyecloseModel,final String mouthcloseModel,final Callback callback)
```

参数名	含义
context	上下文context
eyecloseModel	眼睛闭合检测模型
mouthcloseModel	嘴巴闭合检测模型
callback	鉴权结果 void onResponse(int code, String response) code：请参照此文档 1.1.6 code返回值 response 结果信息

2.2.6 最优人脸模型加载

说明：最优人脸模型初始化

```
void initBestImage(final Context context,final String bestModel,final Callback callback)
```

参数名	含义
context	上下文context
bestModel	最优人脸模型地址
callback	鉴权结果 void onResponse(int code, String response) code：请参照此文档 1.1.6 code返回值 response 结果信息

2.2.7 配置信息加载

说明：检测最小人脸，是否开启内部质量检测，检测或者追踪时间间隔等配置（只适用于3.2及以下版本的detect、track接口）

```
void loadConfig(BDFaceSDKConfig config)
```

参数名	含义
config	参数配置实体

2.2.8 人脸框检测

说明：人脸框检测，每一帧图片都会检测，返回基本人脸信息和72 关键点，可以绘制人脸框，描绘眼耳鼻嘴关键点，也可作用于后续活体，特征抽取入参。

2.2.8.1 根据图片进行检测

```
FaceInfo[] detect(BDFaceSDKCommon.DetectType type,BDFaceImageInstance imageInstance)
```

参数名	含义
detectType	检测类型
imageInstance	图片数据信息
FaceInfo[]	返回参数(人脸参数信息)

2.2.8.2 根据传入的人脸框信息进行检测

说明：可灵活配置的人脸检测接口，使用输入的faceInfos通过配置bdFaceDetectListConfig可以控制是否进行以下能力的预测：人脸检测，关键点提取，头部姿态角，光照，模糊，属性，情绪，闭眼，闭嘴。输出返回值为预测后的faceInfos，可作用于后续活体，特征抽取入参。（faceInfos[]可配置为空，即为第一帧检测）

```
FaceInfo[] detect(BDFaceSDKCommon.DetectType type,BDFaceSDKCommon.AlignType alignType, BDFaceImageInstance imageInstance,FaceInfo[] faceInfos,BDFaceDetectListConf bdFaceDetectListConfig)
```

参数名	含义
detectType	检测类型
alignType	对齐类型
imageInstance	图片数据信息
faceInfos[]	人脸框数据(可以通过人脸追踪能力获取人脸框)
bdFaceDetectListConfig	功能开关
FaceInfo[]	返回参数(人脸参数信息)

2.2.9 人脸跟踪-多人脸检测（接口只支持RGB追踪）

说明：视频人脸跟踪检测，追踪图片中多个人脸信息，通过参数 `num` 配置，接口包含检测和跟踪功能，返回基本人脸信息和72 关键点，可以绘制人脸框，描绘眼耳鼻嘴关键点，也可作用于后续活体，特征抽取入参。 该接口只支持RGB检测和对齐；

```
FaceInfo[] track(DetectType detectType, BDFaceImageInstance imageInstance)
```

参数名	含义
detectType	检测类型（DETECT_VIS ）
imageInstance	图片数据信息
FaceInfo[]	返回参数(人脸参数信息)

2.2.10 人脸跟踪-多人脸检测(接口只支持RGB跟踪)

说明：视频人脸跟踪检测，追踪图片中多个人脸信息，通过参数 `num` 配置，接口包含检测和跟踪功能，返回基本人脸信息和72 关键点，可以绘制人脸框，描绘眼耳鼻嘴关键点，也可作用于后续活体，特征抽取入参。

```
FaceInfo[] track(BDFaceSDKCommon.DetectType detectType, BDFaceSDKCommon.AlignType
alignType,BDFaceImageInstance imageInstance)
```

参数名	含义
detectType	检测类型
alignType	对齐类型
imageInstance	图片数据信息
FaceInfo[]	返回参数(人脸参数信息)

2.2.11 人脸抠图

说明：根据人脸检测结果扣图，扣图结果为矫正之后的人脸信息

```
public BDFaceImageInstance cropFace(BDFaceImageInstance imageInstance, float[] landmark)
```

参数名	含义
imageInstance	图片数据信息
landmark	检测后产出数据
	扣图图像信息，包含宽，高，图片类型，图片数据

返回值：扣图图像信息，包含宽，高，图片类型，图片数据

```
BDFaceImageInstance
```

2.2.12 检测模型卸载

```
void uninitModel()
```

2.2.13 口罩模型加载 说明：检测人脸的口罩。

```
void initMouthMask(final Context context, final String mouthMaskModel, final Callback callback)
```

参数名	含义
context	上下文context
mouthMaskModel	口罩检测模型
callback	鉴权结果 void onResponse(int code, String response) code:请参照此文档 1.1.6 code返回值 response 结果信息

2.3 FaceLive 活体接口

构造方法：

```
// 无参构造调用（执行默认创建的instance ）
public FaceLive()
// 有参构造调用（执行自己创建的instance ）
public FaceLive(BDFaceInstance thisBdFaceInstance)
```

参数名	含义
thisBdFaceInstance	绑定指定的BDFaceInstance实例，否则使用默认的BDFaceInstance实例

2.3.1 活体模型加载

说明：静默活体检测模型初始化，可见光活体模型，深度活体，近红外活体模型初始化

```
 void initModel(final Context context, final String visModel, final String vis2dmaskModel, final String visHandModel, final String visReflectionModel, final String nirModel, final String depthModel, final Callback callback)
```

参数名	含义
context	上下文context
visModel	可见光图片活体模型
vis2dmaskModel	优化rgb活体攻破模型，可以为空
visHandModel	同上
visReflectionModel	同上
depthModel	深度图片活体模型
nirModel	红外图片活体模型
callback	模型加载结果 void onResponse(int code, String response) code：请参照此文档 1.1.6 code返回值 response 结果信息

2.3.2 人脸静默活体检测

说明：静默活体分值检测，返回0-1结果，建议超过0.9 为活体

```
 public float silentLive(LiveType type, BDFaceImageInstance bdFaceImageInstance, float[] landmarks)
```

参数名	含义
type	BDFACE_SILENT_LIVE_TYPE_RGB 可见光图像静默活体检测 BDFACE_SILENT_LIVE_TYPE_NIR 红外图像静默活体检测 BDFACE_SILENT_LIVE_TYPE_DEPTH 深度图像静默活体检测
bdFaceImageInstance	图像对象
landmarks	检查后landmark

2.3.3 活体模型卸载

说明：静默活体模型卸载

```
 void uninitModel()
```

2.4 FaceFeature 特征抽取接口

构造方法：

```
 // 无参构造调用（执行默认创建的instance ）
public FaceFeature()
// 有参构造调用（执行自己创建的instance ）
public FaceFeature(BDFaceInstance thisBdFaceInstance)
```

参数名	含义
thisBdFaceInstance	绑定指定的BDFaceInstance实例，否则使用默认的BDFaceInstance实例

2.4.1 特征模型加载

说明：离线特征获取模型加载，目前支持可见光模型，近红外检测模型（非必要参数，可以为空），证件照模型；用户根据自己场景，选择相应场景模型

2.4.1.1 旧版本模型加载（3.2及以下版本）

说明：只能使用证件照，生活照初始化

```
initModel(final Context context,final String idPhotoModel,final String visModel,final String nirModel,final Callback callback)
```

参数名	含义
context	上下文context
idPhotoModel	证件照图片模型
visModel	可见光图片模型
nirModel	红外图片模型（非必要参数，可以为空）
callback	模型加载结果 void onResponse(int code, String response) code：请参照此文档 1.1.6 code返回值 response 结果信息

2.4.1.2 新版本模型加载（4.0及以上版本）

```
initModel(final Context context,final String idPhotoModel,final String visModel,final String nirModel,final String rgbdModel, final Callback callback)
```

参数名	含义
context	上下文context
idPhotoModel	证件照图片模型
visModel	可见光图片模型
nirModel	红外图片模型
rgbdModel	RGBD图片模型
callback	模型加载结果 void onResponse(int code, String response) code：请参照此文档 1.1.6 code返回值 response 结果信息

2.4.2 特征提取

说明：离线特征提取接口，通过featureType 提取不同图片特征数据，函数返回特征个数，特征存储在feature 参数中

2.4.2.1 RGB图片（生活照、证件照）、NIR图片特征提取

说明：该接口支持RGB证件照、RGB生活照、NIR图片的识别，仅对一张图片进行特征提取

```
float feature(FeatureType featureType, BDFaceImageInstance imageInstance,float[] landmarks, byte[] feature)
```

参数名	含义
featureType	BDFACE_FEATURE_TYPE_LIVE_PHOTO 生活照 BDFACE_FEATURE_TYPE_ID_PHOTO 证件照 BDFACE_FEATURE_TYPE_NIR 红外
imageInstance	图像信息
landmarks	检测后产出的数据
feature	出参：人脸特征 feature 数组，默认初始化512空字节
float	返回128个特征数据

2.4.2.2 深度图片特征抽取（3.2及以下版本不支持）

说明：该接口仅支持RGBD识别，需要对两张图片进行特征提取（RGB图片、Depth图片）

```
float featureRGBD(FeatureType featureType, BDFaceImageInstance imageInstance,BDFaceImageInstance imageInstance_depth, float[] landmarks, byte[] feature)
```

参数名	含义
featureType	BDFACE_FEATURE_TYPE_LIVE_PHOTO //生活照 BDFACE_FEATURE_TYPE_ID_PHOTO // 证件照 BDFACE_FEATURE_TYPE_NIR // 红外 BDFACE_FEATURE_TYPE_RGBD // RGBD特征提取
imageInstance	RGB图像信息
imageInstance	Depth图像信息
landmarks	检测后产出的数据
feature	出参：人脸特征 feature 数组，默认初始化512空字节
float	返回256个特征数据

2.4.3 特征比对

说明：离线特征比对结果，分值为0--100 之间

```
float compare(FeatureType featureType, byte[] feature1, byte[] feature2, boolean isPercent)
```

参数名	含义
featureType	FeatureType.FEATURE_VIS生活照 FeatureType.FEATURE_ID_PHOTO证件照照
feature1	特征1
feature2	特征2
isPercent	控制参数：true返回0~100数值；false 返回0~1
float	比对结果

2.4.4 1:N特征设置

说明：特征集合预加载接口，继承Feature，必须初始化id 和 feature 字段，用于1：N 内部实现和数据返回。

```
public int pushPersonById(int pointID, byte[] feature)
```

参数名	含义
pointID	人员ID
features	人员特征

2.4.5 1:N特征比对

说明：当前feature和预加载Feature 集合比对，返回预加载Feature集合中命中的id，feature 字段和比对分值score；用户可以通过id 在数据库中查找全量信息。

```
public List<? extends Feature> search(BDFaceSDKCommon.FeatureType featureType,
                                      float threshold,
                                      int topNum,
                                      byte[] feature,
                                      boolean isPercent)
```

参数名	含义
featureType	FeatureType.FEATURE_VIS生活照 FeatureType.FEATURE_ID_PHOTO证件照照
threshold	比对阈值
topNum	获取前num 个feature+id映射数组
feature	当前检查人脸特征值
isPercent	控制参数：true返回0~100数值；false 返回0~1

2.4.6 卸载特征模型

```
void uninitModel()
```

2.5 BDFaceImageInstance 图片接口

2.5.1 图片构造

说明：创建图片对象

```
BDFaceImageInstance(byte[] data, int height, int width,BDFaceSDKCommon.BDFaceImageType imageType, float angle, int isMbyteArrayror)
```

参数名	含义
data	图片字节数
height	图片高
width	图片宽
imageType	图片类型
angle	图片旋转角度
isMbyteArrayror	是否镜像

2.5.2 图片构造

说明：创建图片对象

```
BDFaceImageInstance(Bitmap bitmap)
```

参数名	含义
bitmap	图片bitmap

2.5.3 图像句柄创建（YUV,DEPTH图像数据流）

说明：创建图片对象

```
BDFaceImageInstance(byte[] data, int height, int width,BDFaceSDKCommon.BDFaceImageType imageType, float angle, int isMbyteArrayror)
```

参数名	含义
data	图像数据流流
height	图像数据流高
width	图像数据流宽
imageType	图像数据流类型
angle	图像数据流旋转角度（YUV 数据生效）
isMbyteArrayror	图像数据流是否镜像（YUV 数据生效）

2.5.4 图像句柄创建（Bitmap 图像信息）

说明：创建图片对象

```
BDFaceImageInstance(Bitmap bitmap)
```

参数名	含义
bitmap	图片bitmap

2.5.5 图片信息获取（送检）

说明：如果需要校验底层图片，通过该接口获取底层送检图片信息数据

```
BDFaceImageInstance getImage()
```

2.5.6 图片销毁

说明：销毁图片对象，每一帧图像数据使用完之后，必须调用，否着会出现内存溢出

```
int destory();
```

2.6 FaceGaze注意力检测接口

构造方法：

```
// 无参构造调用（执行默认创建的instance ）
public FaceGaze()
// 有参构造调用（执行自己创建的instance ）
public FaceGaze(BDFaceInstance thisBdFaceInstance)
```

参数名	含义
thisBdFaceInstance	绑定指定的BDFaceInstance实例，否则使用默认的BDFaceInstance实例

2.6.1 注意力模型加载 说明：眼睛状态检测，同时判断出左眼，右眼6种状态，分别为向上，向下，向左，向右，向前，闭合

```
public void initModel(final Context context, final String gazeModel, final Callback callback)
```

参数名	含义
context	上下文context
gazeModel	注意力模型
callback	模型加载结果 void onResponse(int code, String response) code：请参照此文档 1.1.6 code返回值 response 结果信息

2.6.2 注意力状态获取

说明：通过图片和关键点获取眼睛注意力状态信息

```
public BDFaceGazeInfo gaze(BDFaceImageInstance imageInstance, float[] landmarks)
```

参数名	含义
bdFaceImageInstance	图像对象
landmarks	检测后landmark

2.6.3 注意力模型卸载

说明：注意力模型卸载

```
void uninitGazeModel()
```

说明：0表示成功，非0失败

2.7 FaceDriverMonitor 驾驶行为监测

构造方法：

```
 // 无参构造调用（执行默认创建的instance ）
FaceDriverMonitor()
// 有参构造调用（执行自己创建的instance ）
FaceDriverMonitor(BDFaceInstance thisBdFaceInstance)
```

2.7.1 驾驶行为监测能力加载 说明：加载驾驶行为监测能力

```
void initDriverMonitor(final Context context, final String driverMonitorModel, final Callback callback)
```

参数名	含义
context	上下文context
driverMonitorModel	驾驶行为监测模型路径
callback	模型加载结果 void onResponse(int code, String response) code：请参照此文档 1.1.6code返回值 response 结果信息

2.7.2 驾驶行为监测能力能力卸载

```
void uninitDriverMonitor()
```

2.7.3 驾驶行为监测检测

```
public BDFaceDriverMonitorInfo driverMonitor(BDFaceImageInstance imageInstance,
                                             FaceInfo faceinfo)
```

参数名	含义
imageInstance	图片数据信息
FaceInfo	人脸参数信息（需要包含人脸检测的人脸框数据）
BDFaceDriverMonitorInfo	返回参数：驾驶行为监测结果

2.8 FaceCrop抠图能力接口

说明：可以根据人脸框或者人脸关键点进行抠图（该功能3.2及以下版本不支持）

构造方法：

```
// 无参构造调用（执行默认创建的instance ）
FaceCrop()
// 有参构造调用（执行自己创建的instance ）
FaceCrop(BDFaceInstance thisBdFaceInstance)
```

2.8.1 initFaceCrop抠图能力加载

```
void initFaceCrop(final Callback callback)
```

参数名	含义
callback	鉴权结果 void onResponse(int code, String response) code：请参照此文档 1.1.6 code返回值 response 结果信息

2.8.2 uninitFaceCrop抠图能力卸载

```
int uninitFaceCrop()
```

参数名	含义
int	0 success；other 参数异常；

2.8.3 cropFaceByBox使用人脸框进行人脸抠图

```
public static BDFaceImageInstance cropFaceByBox(BDFaceImageInstance imageInstance, FaceInfo faceinfo, float enlargeRatio, AtomicInteger isOutofBoundary)
```

参数名	含义
imageInstance	图片数据信息
faceinfo	包含人脸框的人脸信息
enlargeRatio	抠图放大倍数
isOutofBoundary	抠图是否：是否超出图像范围（是否有黑边） 0为未超出，1为超出

2.8.4 cropFaceByLandmark使用人脸关键点进行人脸抠图

```
public BDFaceImageInstance cropFaceByLandmark(BDFaceImageInstance imageInstance,
                                              float[] landmark,
                                              float enlargeRatio,
                                              boolean correction,
                                              AtomicInteger isOutofBoundary)
```

参数名	含义
imageInstance	图片数据信息
landmark	检测后产出数据
enlargeRatio	抠图放大倍数
correction	是否进行人脸矫正
isOutofBoundary	抠图是否：是否超出图像范围（是否有黑边） 0为未超出，1为超出

2.9 FaceMouthMask口罩检测接口

说明：4.0及以上版本支持该功能

构造方法：

```
 // 无参构造调用（执行默认创建的instance ）
public FaceMouthMask()
// 有参构造调用（执行自己创建的instance ）
public FaceMouthMask(BDFaceInstance thisBdFaceInstance)
```

参数名	含义
thisBdFaceInstance	绑定指定的BDFaceInstance实例，否则使用默认的BDFaceInstance实例

2.9.1 口罩检测模型加载

说明：加载口罩检测模型

```
int initModel(final Context context, final String mouthMaskModel, final Callback callback)
```

参数名	含义
context	上下文context
mouthMaskModel	口罩检测模型
callback	鉴权结果 void onResponse(int code, String response) code：请参照此文档 1.1.6 code返回值 response 结果信息

2.9.2 口罩检测结果获取

说明：通过图片和人脸框数据获取口罩检测置信度数据，0代表不戴口罩，1代表戴口罩

```
float[] checkMask(BDFaceImageInstance imageInstance, FaceInfo[] faceInfos)
```

参数名	含义
imageInstance	图像信息
faceInfos	人脸框数据
float[]	返回的戴口罩置信度

2.9.3 口罩检测模型卸载

说明：卸载口罩检测模型

```
int uninitModel()
```

说明：0表示成功，非0失败

2.10 ImageIllum 图片光照检测接口

```
int imageIllum(BDFaceImageInstance imageInstance, AtomicInteger illumScore)
```

参数名	含义
imageInstance	图像信息
illumScore	图片光照强度

2.11 FaceSafetyHat安全帽检测

构造方法：

// 无参构造调用（执行默认创建的instance ）
public FaceSafetyHat()
// 有参构造调用（执行自己创建的instance ）
public FaceSafetyHat(BDFaceInstance thisBdFaceInstance)

参数名	含义
thisBdFaceInstance	绑定指定的BDFaceInstance实例，否则使用默认的BDFaceInstance实例

2.11.1 模型加载 public void initModel(final Context context, final String safetyHatModel, final Callback callback)

参数名	含义
context	上下文context
safetyHatModel	安全帽模型
callback	模型加载结果 void onResponse(int code, String response) code：请参照此文档 2.1.7 code返回值response 结果信息

2.11.2 安全帽检测 public float[] checkHat(BDFaceImageInstance bdFaceImageInstance, FaceInfo[] faceInfos)

参数名	含义
imageInstance	图像信息
faceInfos	人脸信息

2.12 FaceDarkEnhance暗光恢复检测

2.12.1 模型加载

```
public void initFaceDarkEnhance(final Context context, final String FaceDarkEnhanceModel, final Callback callback)
```

参数名	含义
context	上下文context
FaceDarkEnhanceModel	暗光恢复模型
callback	模型加载结果 void onResponse(int code, String response) code：请参照此文档 1.1.6 code返回值 response 结果信息

2.12.2 暗光检测

```
public BDFaceImageInstance faceDarkEnhance(BDFaceImageInstance imageInstance)
```

参数名	含义
imageInstance	图像信息
返回值	暗光增强后的图像信息

2.12.3 能力卸载

```
public int uninitFaceDarkEnhance()
```

2.13 实体类说明

2.13.1 FaceInfo 实体类

```
public class FaceInfo {

  /**
   * ---------detect---------**
   * 人脸索引值，标记连续视频帧追踪中人脸ID
   */
  public int faceID;

  /**
   * 人脸中心点x坐标
   */
  public float centerX;

  /**
   * 人脸中心点y坐标
   */
  public float centerY;

  /**
   * 人脸宽度
```

```
     */
    public float width;

    /**
     * 人脸高度
     */
    public float height;

    /**
     * 人脸角度
     */
    public float angle;

    /**
     * 人脸置信度
     */
    public float score;

    /**
     * 人脸72个关键点数据（鼻子，眼镜，嘴巴，眉毛）
     */
    public float[] landmarks;

    /** -------- config head pose ture------**/

    /**
     * 人脸左右偏转角
     */
    public float yaw;

    /**
     * 人脸平行平面内的头部旋转角
     */
    public float roll;

    /**
     * 人脸上下偏转角
     */
    public float pitch;

    /** ---------config quality blur illum occluton ture---------**/

    /**
     * 人脸模糊度信息
     */
    public float bluriness;

    /**
     * 人脸光照信息
     */
    public int illum;

    /**
     * 人脸遮挡信息
     */
    public BDFaceOcclusion occlusion;

  /**---------config Attribute ture-------**/

    /**
     * 人脸年龄
     */
    public int age;
```

```
    public int age;
    /**
     * 人脸种族（黄，白，黑，印度）
     */
    public BDFaceSDKCommon.BDFaceRace race;
    /**
     * 人脸佩戴眼镜状态（无眼镜，有眼镜，墨镜）
     */
    public BDFaceSDKCommon.BDFaceGlasses glasses;
    /**
     * 人脸性别（男女）状态
     */
    public BDFaceSDKCommon.BDFaceGender gender;
    /**
     * 人脸3种情绪(中性，微笑，大笑)
     */
    public BDFaceSDKCommon.BDFaceEmotion emotionThree;
    /**---------config emotion ture-------**/
    /**
     * 人脸7种情绪（生气，恶心，害怕，开心，伤心，惊讶，无情绪）
     */
    public BDFaceSDKCommon.BDFaceEmotionEnum emotionSeven;
     /**---------config isMouthClose ture-------**/

    /**
     * 嘴巴闭合置信度
     */
    public float mouthclose;

    /**---------config isEyeClose ture-------**/

    /**
     * 左眼闭合的置信度
     */
    public float leftEyeclose;
    /**
     * 右眼闭合的置信度
     */
    public float rightEyeclose;
    /**
     * 最优人脸得分
     */
    public float bestImageScore;

}
```

2.13.2 BDFaceSDKCommon 枚举类

```
public class BDFaceSDKCommon {

    public enum BDFaceImageType {
        BDFACE_IMAGE_TYPE_RGB,
        BDFACE_IMAGE_TYPE_BGR,
        BDFACE_IMAGE_TYPE_RGBA,
        BDFACE_IMAGE_TYPE_BGRA,
        BDFACE_IMAGE_TYPE_GRAY,
        BDFACE_IMAGE_TYPE_DEPTH,
        BDFACE_IMAGE_TYPE_YUV_NV21, //  YYYYVUVU
        BDFACE_IMAGE_TYPE_YUV_NV12, // YYYYUVUV
        BDFACE_IMAGE_TYPE_YUV_YV12, // YYYYVVUU
    }
```

```
// 检测类型
public enum DetectType {
    DETECT_VIS, // 可见光图像
    DETECT_NIR  // 红外图像
}

//  人脸关键点类型枚举
public enum AlignType {
    BDFACE_ALIGN_TYPE_RGB_ACCURATE,  // 精确RGB对齐
    BDFACE_ALIGN_TYPE_RGB_FAST,      // 快速RGB对齐
    BDFACE_ALIGN_TYPE_NIR_ACCURATE,  // 精确NIR对齐
}

/**
 * 图片检测类型，目前支持红外，深度图，可见光
 */
public enum LiveType {
    BDFACE_SILENT_LIVE_TYPE_RGB,
    BDFACE_SILENT_LIVE_TYPE_NIR,
    BDFACE_SILENT_LIVE_TYPE_DEPTH,
}

/**
 * 特征提取图片类型，证件照，可见光，红外，RGBD
 */
public enum FeatureType {
    BDFACE_FEATURE_TYPE_LIVE_PHOTO,    // 生活照特征提取
    BDFACE_FEATURE_TYPE_ID_PHOTO,      // 证件照特征提取
    BDFACE_FEATURE_TYPE_NIR,           // 近红外特征提取
    BDFACE_FEATURE_TYPE_RGBD,          // RGBD特征提取
}

// 质量检测类型
public enum FaceQualityType {
    BLUR,        // 模糊
    OCCLUSION,   // 遮挡
    ILLUMINATION  // 光照
}

 // 表情类型
public enum BDFaceEmotion {
    BDFACE_EMOTION_FROWN,          // 皱眉
    BDFACE_EMOTION_SMILE,          // 笑
    BDFACE_EMOTION_CALM,           // 平静
}

// 情绪
public enum BDFaceEmotionEnum {
    BDFACE_EMOTIONS_ANGRY,         // 生气
    BDFACE_EMOTIONS_DISGUST,       // 恶心
    BDFACE_EMOTIONS_FEAR,          // 害怕
    BDFACE_EMOTIONS_HAPPY,         // 开心
    BDFACE_EMOTIONS_SAD,           // 伤心
    BDFACE_EMOTIONS_SURPRISE,      // 惊讶
    BDFACE_EMOTIONS_NEUTRAL,       // 无情绪
}

// 人脸属性种族
public enum BDFaceRace {
    BDFACE_RACE_YELLOW,      // 黄种人
    BDFACE_RACE_WHITE,       // 白种人
    BDFACE_RACE_BLACK,       // 黑种人
```

```
        BDFACE_RACE_INDIAN,      // 印度人
    }

    // 戴眼镜状态
    public enum BDFaceGlasses {
        BDFACE_NO_GLASSES,  // 无眼镜
        BDFACE_GLASSES,     // 有眼镜
        BDFACE_SUN_GLASSES, // 墨镜
    }

    // 性别
    public enum BDFaceGender {
        BDFACE_GENDER_FEMALE, // 女性
        BDFACE_GENDER_MALE,  // 男性
    }

    // 凝视方向
    public enum BDFaceGazeDirection {
        BDFACE_GACE_DIRECTION_UP,          // 向上看
        BDFACE_GACE_DIRECTION_DOWN,        // 向下看
        BDFACE_GACE_DIRECTION_RIGHT,       // 向右看
        BDFACE_GACE_DIRECTION_LEFT,        // 向左看
        BDFACE_GACE_DIRECTION_FRONT,       // 向前看
        BDFACE_GACE_DIRECTION_EYE_CLOSE,   // 闭眼
    }

    public enum BDFaceActionLiveType {
        BDFace_ACTION_LIVE_BLINK,          // 眨眨眼
        BDFACE_ACTION_LIVE_OPEN_MOUTH,     // 张张嘴
        BDFACE_ACTION_LIVE_NOD,            // 点点头
        BDFACE_ACTION_LIVE_SHAKE_HEAD,     // 摇摇头
        BDFACE_ACTION_LIVE_LOOK_UP,        // 抬头
        BDFACE_ACTION_LIVE_TURN_LEFT,      // 向左转
        BDFACE_ACTION_LIVE_TURN_RIGHT,     // 向右转
    }

    /**
     * log种类枚举
     */
    public enum BDFaceLogInfo {
        BDFACE_LOG_ERROR_MESSAGE,  // 打印输出错误日志
        BDFACE_LOG_VALUE_MESSAGE,  // 打印输出值日志
        BDFACE_LOG_TYPE_PERF,  //打印输出性能日志
        BDFACE_LOG_TYPE_ALL,  //打印输出全部日志
        BDFACE_LOG_TYPE_DEBUG,   //打印输出debug日志
    }

    public enum BDFaceCoreRunMode {
        BDFACE_LITE_POWER_HIGH,     //大核运行模式
        BDFACE_LITE_POWER_LOW,      //  小核运行模式
        BDFACE_LITE_POWER_FULL,     // 大小核混用模式
        BDFACE_LITE_POWER_NO_BIND,  // 不绑核运行模式 （推荐）
        BDFACE_LITE_POWER_RAND_HIGH, // 轮流绑定大核模式
        BDFACE_LITE_POWER_RAND_LOW   // 轮流绑定小核模式
    }
}
```

2.13.3 BDFaceOcclusion 实体类

```
public class BDFaceOcclusion {
public float leftEye;    // 左眼遮挡置信度
public float rightEye;   // 右眼遮挡置信度
public float nose;       // 鼻子遮挡置信度
public float mouth;      // 嘴巴遮挡置信度
public float leftCheek;  // 左脸遮挡置信度
public float rightCheek; // 右脸遮挡置信度
public float chin;       // 下巴遮挡置信度
}
```

2.13.4 BDFaceSDKConfig 配置实体类

```
public class BDFaceSDKConfig {

  /**
   * 输入图像的缩放系数
   */
  public float scaleRatio = -1;

  /**
   * 需要检测的最大人脸数目
   */
  public int maxDetectNum = 10;

  /**
   * 需要检测的最小人脸大小
   */
  public int minFaceSize = 0;

  /**
   * 人脸置信度阈值（检测分值大于该阈值认为是人脸）
   * RGB
   */
  public float notRGBFaceThreshold = 0.5f;
  /**
   * 人脸置信度阈值（检测分值大于该阈值认为是人脸）
   * NIR
   */
  public float notNIRFaceThreshold = 0.5f;

  /**
   * 未跟踪到人脸前的检测时间间隔
   */
  public float detectInterval = 0;

  /**
   * 已跟踪到人脸后的检测时间间隔
   */
  public float trackInterval = 500;

  /**
   * 质量检测模糊，默认不做质量检测
   */
  public boolean isCheckBlur = false;

  /**
   * 质量检测遮挡，默认不做质量检测
   */
  public boolean isOcclusion = false;

  /**
```

```
    * 质量检测光照，默认不做质量检测
    */
   public boolean isIllumination = false;

   /**
    * 姿态角检测，获取yaw(左右偏转角)，roll(人脸平行平面内的头部旋转角)，pitch(上下偏转角),默认不检测
    */
   public boolean isHeadPose = false;

  /**
    * 属性检查，获取年龄，种族，是否戴眼镜等信息，默认不检测
    */
   public boolean isAttribute = false;

   /**
    * 7种情绪信息获取，默认不检测（3.2及以下版本不支持）
    */
   private boolean isEmotion = false;

   /**
    * 是否扣图，默认不扣图（3.2及以下版本不支持）
    */
   public boolean isCropFace = false;

   /**
    * 是否检测眼睛闭合，默认不检测
    */
   private boolean isEyeClose = false;

   /**
    * 是否检测嘴巴闭合，默认不检测
    */
   private boolean isMouthClose = false;

   /**
    * 是否检测最优人脸，默认不检测
    */
   public boolean isBestImage = false;

}
```

2.13.5 Feature 实体类

```
 public class Feature {
   private int id;
   private String faceToken = "";
   private byte[] feature;
   private String userId = "";
   private String groupId = "";
   private long ctime;
   private long updateTime;
   private String imageName = "";
   private String userName = "";
   private String cropImageName = "";
   private boolean isChecked;
   private float score;
}
```

2.13.6 BDFaceGazeInfo 实体类

```
public class BDFaceGazeInfo {
public float leftEyeConf;                    // 左眼的置信度
public float rightEyeConf;                   // 右眼的置信度
public float softmaxEyeConf;
public BDFaceGazeDirection leftEyeGaze;      // 左眼的注意力信息
public BDFaceGazeDirection rightEyeGaze;     // 右眼的注意力信息
public BDFaceGazeDirection softmaxEyeGaze;
}
```

2.13.7 BDFaceDetectListConf 实体类

```
public class BDFaceDetectListConf {
/**
 * 检测（在track 精度不足时开启）默认不做检测
 */
public boolean usingDetect = false;
/**
 * 检测（在track 精度不足时开启）默认开启对齐
 */
public boolean usingAlign = true;
/**
 * 质量检测模糊，默认不做质量检测
 */
public boolean usingQuality = false;
/**
 * 姿态角检测，获取yaw(左右偏转角)，roll(人脸平行平面内的头部旋转角)，pitch(上下偏转角),默认不检测
 */
public boolean usingHeadPose = false;
/**
 * 属性检查，获取年龄，种族，是否戴眼镜等信息，默认不检测
 */
public boolean usingAttribute = false;
/**
 * 7种情绪信息获取，默认不检测
 */
public boolean usingEmotion = false;
/**
 * 是否检测眼睛闭合，默认不检测
 */
public boolean usingEyeClose = false;
/**
 * 是否检测嘴巴闭合，默认不检测
 */
public boolean usingMouthClose = false;
/**
 * 是否检测最优人脸，默认不检测
 */
public boolean usingBestImage = false;
/**
 * 是否检测戴口罩，默认不检测
 */
public boolean usingMouthMask = false;
}
```

2.13.8 BDFaceDriverMonitorInfo 驾驶检测结果信息

```
public class BDFaceDriverMonitorInfo {
    float normal = 0;    // 行为正常
    float calling = 0;   // 打电话
    float drinking = 0;  // 喝水
    float eating = 0;    // 吃东西
    float smoking = 0;   // 抽烟
}
```

3、示例参考

3.1 功能介绍

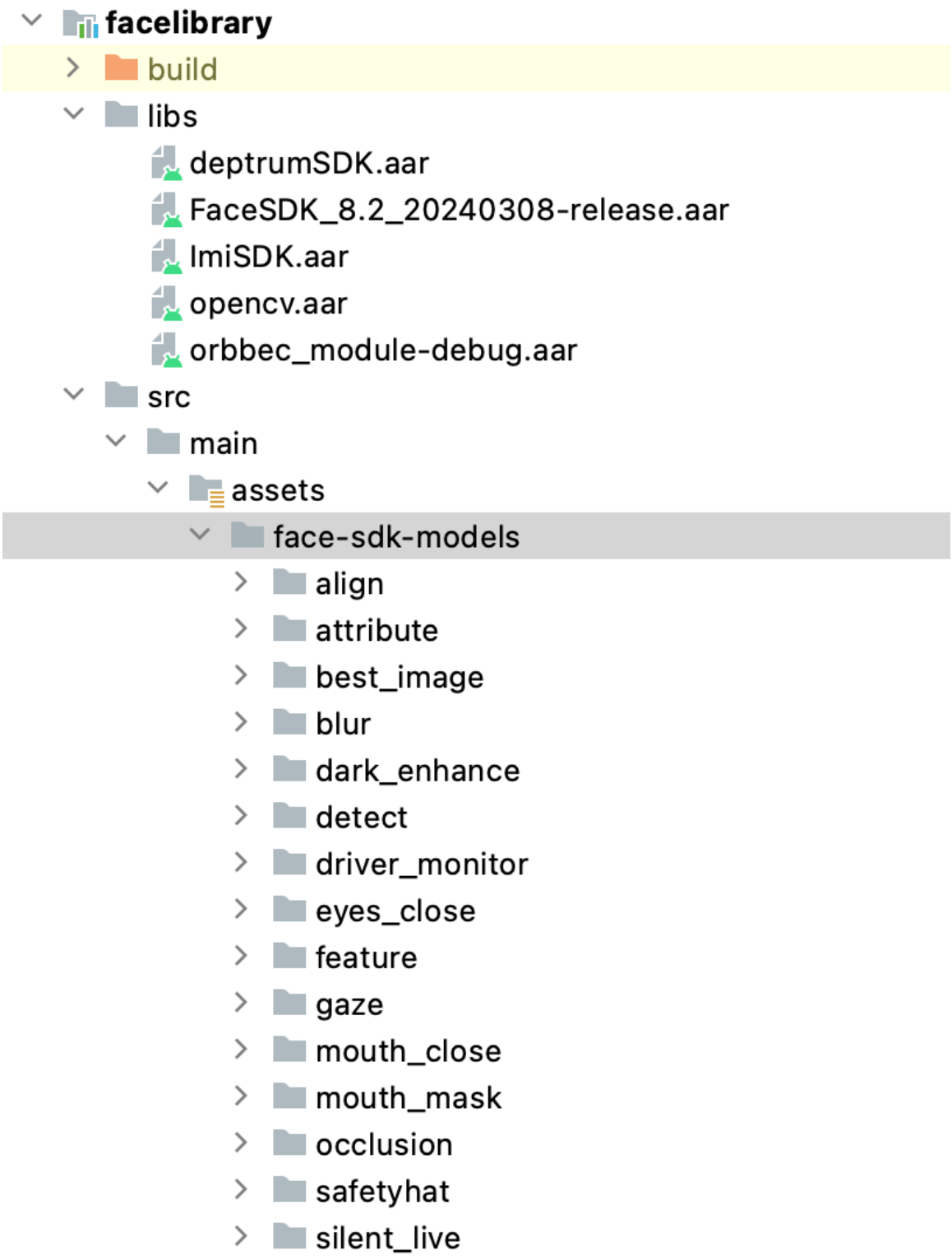
为了方便您快速体验新版SDK的能力，我们为您构建了一个简易的示例工程，包含了人脸识别端上的常用能力，并配以带有交互的UI界面，希望能更直接地让您理解此版SDK的功能。同时，您也可以基于此示例工程进行二次开发，快速构建Demo或业务工程。

1. **多种鉴权模式**：提供序列号、离线License文件授权的可视化方式。
2. **人脸注册**：提供全屏跟踪人脸注册，和固定区域人脸注册，满足不同业务场景要求。
3. **1：N识别**：提供完整的1：N识别功能和界面设计。
4. **调试模式**：提供多种模态的预览图界面、各模型的实际应用耗时，方便用于评估应用性能。
5. **多模态活体检测**：提供预设的无活体、RGB活体、RGB+NIR活体、RGB+Depth四种活体设置，并预设适配镜头能力。
6. **人脸库管理**：提供库、组、用户的三层人脸库管理逻辑，并支持可视化批量导入功能。
7. **人证比对**：提供证件照模型，并预设在1：1功能中，支持实时采集和照片两种录入模式。
8. **高级设置**：支持质量检测、活体模态、检测模态等多项可视化配置能力，方便直接上手调节使用。

3.2 代码结构

3.2.1 核心库介绍

facelibrary是SDK的依赖库。



lib目录为动态库so，包含**arm64-v8a**和**armeabi-v7a**两个平台。

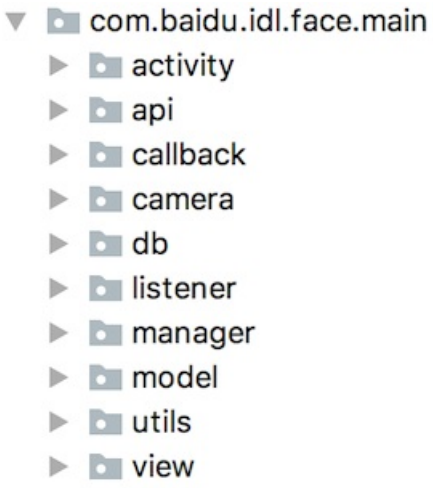
名称	功能
libbdface_sdk.so	人脸核心库
libbd_license.so、libliantian.so	授权核心库
libaikl_calc_arm.so、libaikl_cluster_arm.so	加速算法核心库

新版sdk以上so库文件都合并至FaceSDK_8.2_20240308-release.so中

assets目录为模型文件。

名称	功能	是否必须
detect	人脸检测核心模型	是
align	人脸对齐核心模型	是
blur、occlusion	质量检测模型	否
slient_live	活体检测模型	是
feature	识别核心模型	是

java目录为用户组管理、人脸SDK操作、视频流、图片等操作辅助类



3.2.2 示例代码结构

名称	功能
MainActivity	主功能页面，包含人脸检索，认证比对，功能设置，授权激活功能入口
FaceRegisterNewActivity	人脸注册
BatchImportActivity	人脸库管理，批量导入功能
FaceDetectAngleActivity	人脸检测角度设置，用于调整实际送去人脸检测的图片的角度，包括0,90,180,270。SDK只能识别人脸朝上的人脸。
CameraDisplayAngleActivity	摄像头视频流回显角度
GateMinFaceActivity	最小人脸设置
FaceLivinessType	活体类型设置，分为无活体、RGB活体、RGB+NIR活体、RGB+Depth活体、RGB+NIR+Depth活体。默认为不使用活体。示例工程里面进行活体设置后，后续的人脸注册、人脸1：1，1：n等操作需选择相应Activity。无活体和rgb活体需要使用单目usb摄像头，rgb+ir活体需要使用rgb+ir双目摄像头。RGB+Depth活体，目前只能使用RGB+Depth活体功能。RGB+NIR+Depth的活体硬件设备适配还在开发中，敬请期待。
MirrorSettingActivity	镜像调节页面

3.3 开始集成

3.3.1 鉴权

鉴权初始化

```
public void init(final Context context, final SdkInitListener listener)
```

- 参数说明：

参数	含义
context	上下文
listener	初始化回调

- 接口调用：

```
FaceSDKManager.getInstance().init(mContext, new SdkInitListener())
```

模型初始化

说明：包含两种检测模型（RGB+NIR）、对齐、质量检测、活体3种（RGB+NIR+Depth）、特征提取两种（生活照+证件照）共

10个模型

```
public void initModel(final Context context, BDFaceSDKConfig config, boolean isLog, final SdkInitListener listener)
```

- 参数说明：

参数	含义
context	上下文
config	检测实体类
isLog	是否打印日志
listener	初始化回调

- 接口调用：

```
FaceSDKManager.getInstance().initModel(mContext, new SdkInitListener())
```

方式一：使用序列号形式的在线鉴权

```
public void initLicenseOnLine(final Context context, final String licenseID, final Callback callback)
```

- 参数说明：

参数	含义
context	上下文
licenseID	后台购买或申请的序列号
callback	授权结果 void onResponse(int code, String response) code 0：成功；code 1 加载失败

- 接口调用：

```
FaceAuth faceAuth = new FaceAuth();
faceAuth.initLicenseOnLine(context, licenseID, new callback)
```

方式二：使用本地文件形式的离线鉴权

```
public void initLicenseOffLine(final Context context, final Callback callback)
```

- 参数说明：

参数	含义
context	上下文
callback	授权结果 void onResponse(int code, String response)code 0：成功；code 1 加载失败

- 接口调用：

```
FaceAuth faceAuth = new FaceAuth();
faceAuth.initLicenseOffLine(context, licenseID, new callback)
```

方式三：使用按应用信息的批量鉴权

```
public void initLicenseBatchLine(final Context context, final String licenseKey, final Callback callback)
```

- 参数说明：

参数	含义
context	上下文
licenseKey	需百度负责人提供
callback	授权结果 void onResponse(int code, String response)code 0：成功；code 1 加载失败

- 接口调用：

```
 FaceAuth faceAuth = new FaceAuth();
faceAuth.initLicenseBatchLine(context, licenseKey, new callback)
```

3.3.2 人脸注册

进入该功能之后首先需要录入输入用户名+组名，用户名要求不能含有特殊符号，要求不能超过30个字符，填入成功之后才可进入视频流实时采集页面。**FaceRegisterActivity**是注册页面。采集页面将会执行人脸检测、人脸活体检测、特征值提取功能。前面都通过即执行注册，注册成功后并将注册成功的图片保存到本地。您可以根据实际使用的硬件进行选择。

> **注一**：为了使注册的图片达到较高的质量，注册默认开启质量检测与远近距离校验。设置里的质量检测开关对注册页面不起作用。

> **注二**：为了使注册效果更好。所有活体状态下注册默认开启固定区域检测，设置里的检测跟踪策略对注册页面不起作用。

1）注册采集，可选择以下3种方式返回人脸

- FaceRegisterNewActivity：无活体或RGB活体（活体检测成功后，注册人脸）
- FaceRegisterNewNIRActivity：进行RGB+NIR活体成功后注册人脸
- FaceRegisterNewDepthActivity：进行RGB+Depth活体成功后注册人脸（目前仅支持奥比中光AtIas镜头）

下面是检测相关代码：

```
    private void faceDetect(byte[] data, final int width, final int height) {

        FaceSDKManager.getInstance().onDetectCheck(bdFaceImageConfig, null, null,
                bdFaceCheckConfig , new FaceDetectCallBack() {
                    @Override
                    public void onFaceDetectCallback(LivenessModel livenessModel) {
                        // 输出结果

                        checkFaceBound(livenessModel);
                    }

                    @Override
                    public void onTip(final int code, String msg) {
                        runOnUiThread(new Runnable() {
                            @Override
                            public void run() {
                                if (mFaceRoundProView == null) {
                                    return;
                                }
                                if (code == 0){
                                    mFaceRoundProView.setTipText("请保持面部在取景框内");
                                    mFaceRoundProView.setBitmapSource(R.mipmap.ic_loading_grey , false);
                                }else {
                                    mFaceRoundProView.setTipText("请保证人脸区域清晰无遮挡");
                                    mFaceRoundProView.setBitmapSource(R.mipmap.ic_loading_blue , true);
                                }
                            }
                        });
                    }

                    @Override
                    public void onFaceDetectDarwCallback(LivenessModel livenessModel) {
                        showFrame(livenessModel);
                    }
                });
    }
```

下面是活体判断相关代码：

```
    private void checkLiveScore(LivenessModel livenessModel) {
      if (livenessModel == null || livenessModel.getFaceInfo() == null) {
          mFaceRoundProView.setTipText("请保持面部在取景框内");
          return;
      }

      // 获取活体类型
      int liveType = SingleBaseConfig.getBaseConfig().getType();
      // int liveType = 2;

      if (livenessModel.isQualityCheck()){
          mFaceRoundProView.setTipText("请保证人脸区域清晰无遮挡");
          mFaceRoundProView.setBitmapSource(R.mipmap.ic_loading_blue , true);
          return;
      } else if (bdLiveConfig == null) {          // 无活体
          getFeatures(livenessModel);
      } else { // RGB活体检测
          float rgbLivenessScore = livenessModel.getRgbLivenessScore();
          float liveThreadHold = SingleBaseConfig.getBaseConfig().getRgbLiveScore();
          // Log.e(TAG, "score = " + rgbLivenessScore);
          if (rgbLivenessScore < liveThreadHold) {
              mFaceRoundProView.setTipText("请保证采集对象为真人");
              mFaceRoundProView.setBitmapSource(R.mipmap.ic_loading_blue , true);
              return;
          }
          // 提取特征值
          getFeatures(livenessModel);
      }
  }
```

下面是人脸特征提取的代码：

```
    private void displayCompareResult(float ret, byte[] faceFeature, LivenessModel model) {
        if (model == null) {
            mFaceRoundProView.setTipText("请保持面部在取景框内");
            mFaceRoundProView.setBitmapSource(R.mipmap.ic_loading_grey , false);
            return;
        }

        // 特征提取成功
        if (ret == 128) {
            // 抠图
            BDFaceImageInstance cropInstance =
                    FaceSDKManager.getInstance().getCopeFace(
                            BitmapUtils.getInstaceBmp(model.getBdFaceImageInstance()) ,
                             model.getLandmarks(),
                             0
                    ); ;
            if (cropInstance == null) {
                mFaceRoundProView.setTipText("抠图失败");
                mFaceRoundProView.setBitmapSource(R.mipmap.ic_loading_blue , true);
                return;
            }
            mCropBitmap = BitmapUtils.getInstaceBmp(cropInstance);
            // 获取头像
            if (mCropBitmap != null) {
                mCollectSuccess = true;
                mCircleHead.setImageBitmap(mCropBitmap);
            }
            cropInstance.destory();

            mRelativeCollectSuccess.setVisibility(View.VISIBLE);
            mRelativePreview.setVisibility(View.GONE);
            mFaceRoundProView.setTipText("");

            for (int i = 0; i < faceFeature.length; i++) {
                mFeatures[i] = faceFeature[i];
            }
        } else {
            mFaceRoundProView.setTipText("特征提取失败");
            mFaceRoundProView.setBitmapSource(R.mipmap.ic_loading_blue , true);
        }
    }
```

下面是将特征值及用户信息注册到人脸库的代码：

```
 String userName = mEditName.getText().toString();
// 姓名过滤
String nameResult = FaceApi.getInstance().isValidName(userName);
if (!"0".equals(nameResult)) {
    ToastUtils.toast(getApplicationContext(), nameResult);
    return;
}
String imageName = userName + ".jpg";
// 注册到人脸库
boolean isSuccess = FaceApi.getInstance().registerUserIntoDBmanager(null,userName, imageName, null, mFeatures);
if (isSuccess) {
    // 保存人脸图片
    File faceDir = FileUtils.getBatchImportSuccessDirectory();
    File file = new File(faceDir, imageName);
    FileUtils.saveBitmap(file, mCropBitmap);
    // 数据变化，更新内存
    // FaceSDKManager.getInstance().initDatabases();
    // 更新UI
    mRelativeCollectSuccess.setVisibility(View.GONE);
    mRelativeRegisterSuccess.setVisibility(View.VISIBLE);
    mCircleRegSucHead.setImageBitmap(mCropBitmap);
} else {
    ToastUtils.toast(getApplicationContext(), "保存数据库失败，" +"可能是用户名格式不正确");
}
```

参数填写时请注意：

1. 用户名为必填项，支持英文与数字。
2. 用户组为可选项，不填默认生成名为default 的用户组；
3. 用户信息可不填，用于其他需求；
4. 注册成功后生成的图片命名格式为group-username.jpg,保存在sdcard下的Success-Import文件夹下，用于人脸库的图片显示。

3.3.3 1：N识别

RGB搜索

- 打开预览：

```
public void startPreview(GlMantleSurfacView textureView, int videoDirection ,int width, int height)
```

- 参数说明：

参数	含义
textureView	帧图像预览数据
videoDirection	视频流旋转角度
width	图片的宽
height	图片的高

- 接口调用：

```
 // 设置相机的ID
if (SingleBaseConfig.getBaseConfig().getRBGCameraId() != -1) {
    CameraPreviewManager.getInstance().setCameraFacing(SingleBaseConfig.getBaseConfig().getRBGCameraId());
} else {
    CameraPreviewManager.getInstance().setCameraFacing(CameraPreviewManager.CAMERA_USB);
}
// 开启预览
CameraPreviewManager.getInstance().startPreview(glMantleSurfacView,
    SingleBaseConfig.getBaseConfig().getRgbVideoDirection() , PREFER_WIDTH, PERFER_HEIGH);
```

- 人脸检索(包含检测、活体、特征、人脸搜索全流程)

```
public void onDetectCheck(final BDFaceImageConfig bdFaceImageConfig,
                          final BDFaceImageConfig bdNirFaceImageConfig,
                          final BDFaceImageConfig bdDepthFaceImageConfig,
                          final BDFaceCheckConfig bdFaceCheckConfig,
                          final FaceDetectCallBack faceDetectCallBack)
```

- 参数说明：

参数	含义
bdFaceImageConfig	视频流数据以及相关配置
bdNirFaceImageConfig	红外YUV数据流以及相关配置
bdDepthFaceImageConfig	深度depth数据流以及相关配置
bdFaceCheckConfig	特征提取模式、质量和活体阈值以及相机类型等配置类
faceDetectCallBack	检测结果回调

接口调用：

```
FaceSDKManager.getInstance().onDetectCheck(data, null, null,height, width, mLiveTyenew, FaceDetectCallBack()
```

RGB+NIR搜索

- 非Debug模式：FaceRGBIRCloseDebugSearchActivity
- Debug模式：FaceRGBIROpenDebugSearchActivity

```
public void onDetectCheck(final byte[] rgbData,
                          final byte[] nirData,
                          final byte[] depthData,
                          final int srcHeight,
                          final int srcWidth,
                          final int liveCheckMode,
                          final FaceDetectCallBack faceDetectCallBack)
```

- 参数说明：

参数	含义
rgbData	可见光YUV 数据流
nirData	红外YUV 数据流
depthData	深度depth 数据流
srcHeight	可见光YUV 数据流-高度
srcWidth	可见光YUV 数据流-宽度
liveCheckMode	活体检测类型： 不使用活体：1； RGB活体：2； RGB+NIR活体：3； RGB+Depth活体：4
faceDetectCallBack	检测结果回调

- 接口调用 ：

```
 FaceSDKManager.getInstance().onDetectCheck(rgbData, irData, null, PERFER_HEIGH, PREFER_WIDTH, 3, new
FaceDetectCallBack())
```

RGB+Depth搜索

- 非Debug模式：FaceRGBDepthCloseDebugSearchActivity
- Debug模式：FaceRGBDepthOpenDebugSearchActivity

```
public void onDetectCheck(final byte[] rgbData,
                          final byte[] nirData,
                          final byte[] depthData,
                          final int srcHeight,
                          final int srcWidth,
                          final int liveCheckMode,
                          final FaceDetectCallBack faceDetectCallBack)
```

- 参数说明：

参数	含义
rgbData	可见光YUV 数据流
nirData	红外YUV 数据流
depthData	深度depth 数据流
srcHeight	可见光YUV 数据流-高度
srcWidth	可见光YUV 数据流-宽度
liveCheckMode	活体检测类型： 不使用活体：1； RGB活体：2； RGB+NIR活体：3； RGB+Depth活体：4
faceDetectCallBack	检测结果回调

- 接口调用 ：

```
FaceSDKManager.getInstance().onDetectCheck(rgbData, null, depthData, RGB_HEIGHT, RGB_WIDTH, 4, new
FaceDetectCallBack()
```

3.3.4 人脸库管理

批量导入

本功能下支持批量导入人脸图片数据，用于将图片录入到人脸库中，即批量注册。该过程不进行活体检测，仅执行批量人脸特征值生成录入，每张图片提取一张最大人脸做为注册人脸，人脸图片较多时可能会耗时较长，请耐心等待。

```
// 走人脸SDK接口，通过人脸检测、特征提取拿到人脸特征值
 ret = FaceApi.getInstance().getFeature(bitmap, bytes,
 BDFaceSDKCommon.FeatureType.BDFACE_FEATURE_TYPE_LIVE_PHOTO);
 LogUtils.i(TAG, "live_photo = " + ret);
```

使用步骤：

1. 在SD卡根目录，创建一个文件夹，命名为`Face-Import`。
2. 将需要导入的每个图片，文件名示例为 `username_group.jpg` ，即用户名-用户组名的文件命名方式，用户名建议4-20位字符，支持字母、数字、下划线组合。
3. 新建一个压缩文件，命名为`Face.zip`，将人脸图片全部放到该zip下，并将zip文件放置到`Face-Import`目录下。
4. 点击下方按钮开始解压操作，批量注册时间较长，请耐心等待，不要关闭此窗口。

其中：

批量导入的接口如下：

```
public void batchImport();
```

获取压缩包导入目录相关代码：

```
// 获取导入目录 /sdcard/Face-Import
 File batchImportDir = FileUtils.getBatchImportDirectory();
```

遍历该目录下的所有文件相关代码：

```
// 遍历该目录下的所有文件
 File[] picFiles = batchImportDir.listFiles();
 if (picFiles == null || picFiles.length == 0) {
    Log.i(TAG, "导入数据的文件夹没有数据");
    if (mImportListener != null) {
        mImportListener.showToastMessage("导入数据的文件夹没有数据");
        }
        return;
    }
```

判断Face.zip是否存在的相关代码：

```
// 判断Face.zip是否存在
  File zipFile = FileUtils.isFileExist(batchFaceDir.getPath(),
        "Face.zip");
  if (zipFile == null) {
      LogUtils.i(TAG, "导入数据的文件夹没有Face.zip");
      if (mImportListener != null) {
          mImportListener.showToastMessage("搜索失败，
              请检查操作步骤并重试");
      }
      return;
  }
```

如果Face.zip文件存在，并且解压成功之后，就可以进行导入了。 导入过程中会首先判断图片后缀是不是满足要求：

```
// 获取图片名
 String picName = picFiles[i].getName();
 // 判断图片后缀
 if (!picName.endsWith(".jpg") && !picName.endsWith(".png")) {
  LogUtils.i(TAG, "图片后缀不满足要求");
     mFinishCount++;
     mFailCount++;
     // 更新进度
     updateProgress(mFinishCount, mSuccessCount, mFailCount,
      ((float) mFinishCount / (float) mTotalCount));
     continue;
 }
```

如果满足要求，则判断图片命名是否满足要求：

```
  // 获取不带后缀的图片名
   String picNameNoEx = FileUtils.getFileNameNoEx(picName);
   // 通过既定的图片名格式，按照“-”分割，获取组名和用户名
   String[] picNames = picNameNoEx.split("-");
   // 如果分割失败，则该图片命名不满足要求
   if (picNames.length != 2) {
    LogUtils.e(TAG, "图片命名格式不符合要求");
       mFinishCount++;
       mFailCount++;
       continue;
   }

根据姓名查询数据库与文件中对应的姓名是否相等，如果相等，则直接过滤

 // 根据姓名查询数据库与文件中对应的姓名是否相等，如果相等，则直接过滤
   List<User> listUsers = FaceApi.getInstance()
    .getUserListByUserName(groupName, userName);
   if (listUsers != null && listUsers.size() > 0) {
    LogUtils.i(TAG, "与之前图片名称相同");
       mFinishCount++;
       mFailCount++;
       // 更新进度
       updateProgress(mFinishCount, mSuccessCount, mFailCount,
        ((float) mFinishCount / (float) mTotalCount));
       continue;
   }
```

导入过程中都会有log提示，出现导入失败情况要根据log提示定位出问题原因。

组列表信息管理

该页面主要是查询、删除组列表操作。

- 获取组列表信息：

```
  public List<Group> getGroupListByGroupId(String groupId)
```

参数：

参数类型	参数名称	取值范围	说明
String	groupId	不可为空	要查询的组名称，为null则返回null

用户详情信息管理

该页面主要有用户详细信息展示、删除以及用户图片替换功能。

- 用户删除：

```
  public boolean userDelete(String userId);
```

参数：

参数类型	参数名称	取值范围	说明
String	userId	不可为空	当前用户id

- 更换图片：

```
public boolean userUpdate(String userName, String imageName, byte[] feature);
```

参数：

参数类型	参数名称	取值范围	说明
String	userName	不可为空	当前用户名
String	imageName	不可为空	当前图片名称
byte[]	feature	不可为空	人脸特征值

3.3.5 人证比对

模拟真实场景下，人脸图片与证件照图片（如小图、身份证芯片照）对比的业务流程，特征抽取默认使用「证件照模型」，以处理对比过程中的证件照图片特征抽取的要求（证件照图片普遍像素较低）。配套使用的活体检测功能，需要在设置-->镜头及活体检测模式中单独选择对应的已经适配的镜头。

比对完成返回核验结果与相似度分值，分值大于相似度阈值则表示核验通过，反之核验不通过。

> **注**：SDK只检测人脸朝上的人脸。

从视频流中采集两张人脸图片进行对比

此种方式的人脸图片需要从视频流中实时采集，如果为无人值守情况，还需配备活体检测以保障业务安全。`FaceIdCompareActivity`是比对页面，根据活体策略选择相应的实现，开发者可以根据实际使用的硬件进行选择。

采集人脸，可选择以下3种方式返回人脸

- FaceRGBPersonActivity：无活体或RGB活体（活体检测成功后，返回检测到的人脸）
- FaceIRTestimonyActivity：进行RGB+NIR活体成功后返回检测到RGB人脸
- FaceDepthTestimonyActivity：进行RGB+Depth活体成功后返回检测到RGB人脸（奥比中光Atlas镜头）

1）根据返回人脸抽取特征

```
 final Bitmap bitmap = BitmapFactory.decodeStream(getContentResolver().openInputStream(uri1));
if (bitmap != null) {
    byte[] secondFeature = new byte[512];
    // 提取特征值
    float ret = FaceSDKManager.getInstance().personDetect(bitmap, secondFeature,
            FaceUtils.getInstance().getBDFaceCheckConfig() , this);
    // 提取特征值
    // 上传图片有人脸显示
    hintShowIv.setVisibility(View.VISIBLE);
    testimonyShowImg.setVisibility(View.VISIBLE);
    hintShowIv.setImageBitmap(bitmap);
    testimonyShowImg.setImageBitmap(bitmap);
    if (ret != -1) {
        isFace = false;
        // 判断质量检测，针对模糊度、遮挡、角度
        if (ret == 128) {
            bdFaceCheckConfig.setSecondFeature(secondFeature);
            toast("图片特征抽取成功");
            hintShowRl.setVisibility(View.VISIBLE);
            testimonyShowRl.setVisibility(View.VISIBLE);
            testimonyAddIv.setVisibility(View.GONE);
            testimonyUploadFilesTv.setVisibility(View.GONE);
            developmentAddRl.setVisibility(View.GONE);
        } else {
            ToastUtils.toast(mContext, "图片特征抽取失败");
        }
    } else {
        isFace = true;
        // 上传图片无人脸隐藏
        hintShowIv.setVisibility(View.GONE);
        testimonyShowImg.setVisibility(View.GONE);
        hintShowRl.setVisibility(View.VISIBLE);
        testimonyShowRl.setVisibility(View.VISIBLE);
        testimonyAddIv.setVisibility(View.GONE);
        testimonyUploadFilesTv.setVisibility(View.GONE);
        developmentAddRl.setVisibility(View.GONE);
    }
}
```

2）比对两张人脸图片

```
 if (featureCheckMode == 4) {
    // 目前仅支持
    float score = faceModel.getFaceSearch().compare(
                        BDFaceSDKCommon.FeatureType.BDFACE_FEATURE_TYPE_ID_PHOTO,
                        livenessModel.getFeature(), secondFeature, true);
    livenessModel.setScore(score);
    if (score > threholdScore) {
        /*faceId = livenessModel.getFaceInfo().faceID;
        trackTime = System.currentTimeMillis();
        faceAdoptModel = livenessModel;
        failNumber = 0;
        isFail = false;*/
        setFail(livenessModel);
    } else {
        setFail(livenessModel);
    }
}
```

3.3.6 功能设置

我们将常用功能设置预设在工程中，sdcard根目录下的 faceConfig.txt 文件保存了人脸识别的各项基础配置，主要功能可以调整人脸镜像，人脸角度等相关配置（详细说明可进入配置页里查看说明）。您也可以基于预设配置，进行自定义配置修改。

配置信息初始化（从sdcard根目录的faceConfig.txt的文件里读取配置）

```
  isConfigExit = ConfigUtils.isConfigExit();
isInitConfig = ConfigUtils.initConfig();
if (isInitConfig && isConfigExit) {
  Toast.makeText(MainActivity.this, "初始配置加载成功", Toast.LENGTH_SHORT).show();
} else {
  Toast.makeText(MainActivity.this, "初始配置失败,将重置文件内容为默认配置", Toast.LENGTH_SHORT).show();
  ConfigUtils.modityJson();
  }
```

每次修改配置之后，需调用以下方法，更新配置文件并重新读取数据。

```
ConfigUtils.modityJson();
```

3.4 核心类说明

FaceSDKManager

功能：负责初始检测类FaceAuth、FaceDetector、FaceFeature、FaceLiveness com.baidu.idl.face.main.manager

FaceLiveness

功能：人脸活体相关操作封装类，包含人脸rgb、ir、depth活体检测 com.baidu.idl.main.facesdk

ImportFileManager

功能：批量导入的相关操作 com.baidu.idl.face.main.manager

UserInfoManager

功能：人脸库管理相关操作 com.baidu.idl.face.main.manager

DBManager

功能：数据库相关操作 com.baidu.idl.face.main.db

WIN-C++-SDK

版本日志

- 全方面兼容Windows全系统平台，支持win7/win10/XP系统，同时兼容X32、X64系统，深度适配X32系统工控机「赛扬」J1900、「奔腾」G4400等旧设备CPU，已推出C++/C#/Java多语言版本
- 此页面仅把C++语言版本作为示例，若需获取C#、Java版本SDK及相关技术文档，请登陆百度智能云控制台
- Windows 版本SDK官网介绍：点击跳转
- Windows 版本SDK常见问题：点击跳转

> 备注：由于部分Windows设备机型较旧、性能较弱，全流程耗时会增加

版本	日期	更新说明
v8.2	2022.1	1. 优化多线程 2. 删除facebox中的无用姿态角angle

v8.2	2.20	3. 对容易引发内存泄漏的示例做说明，引导客户注意使用 4. 更新授权库，支持多网卡按顺序排序读取
v8.1	2022.06.21	1. 模型升级最新8.0版本，并基于8.0增加业务层二次开发接口 2. 提升对儿童、老人、外国人和戴口罩场景的识别准确率，优化活体算法对复杂光线泛化性
v6.3	2021.12.28	1. 依赖库拆减、使sdk主库不依赖镜头模组库，镜头模组库需要时候单独添加； 2. sdk自动授权激活； 3. 新增设备指纹接口； 4. 人脸库优化，支持中文路径、用户信息支持中文参数，人脸图片入库等； 5. 模型可根据需要删减、模型能力可定制化设置； 6. 新增特征值多端对齐示例代码。
v6.2	2021.12.07	基于6.1业务层接口，开发带UI界面（QT编译）demo。
v6.1	2021.08.25	基于V6.0版本增加如下业务层二次开发接口，帮助客户快速集成上手 一、人脸检测跟踪接口（传入opencv转换的mat） 1. 人脸检测track 2. 人脸跟踪detect 二、人脸管理接口 （传入opencv转换的mat） 1. 人脸注册接口 2. 人脸更新接口 3. 人脸删除接口 4. 用户删除接口 5. 创建用户组接口 6. 用户组删除 7. 用户信息查询接口 8. 用户组列表查询接口 9. 组列表查询接口 三、人脸对比及识别接口（传入opencv转换的mat） 1. 人脸识别identify接口 2. 特征值提取接口 3. 特征值比较接口 四、活体检测接口 （传入opencv转换的mat） 1. 近红外(NIR)活体检测接口 2. 可见光(RGB)活体检测接口 3. 可见光(RGB)&深度(Depth)活体检测接口 4. 有动作活体检测接口（action_live） 五、属性及质量接口（传入opencv转换的mat） 1. 人脸属性接口 六、其他接口能力 1. 检测参数可定制化，包括最大检测人数、最小人脸大小，跟踪时候的检测间隔等 2. 暗光恢复 3. 最佳人脸 4. 人脸遮挡、光照、模糊度 5. 人脸扣图 6. 头角度
v6.0 多语言 通用版	2021.06.22	1. 更新人脸检测模型，提升在强光、暗光、逆光、阴阳光等复杂光线场景下人脸检测的召回率； 2. 更新RGB、NIR和Depth三种模态的活体检测模型，提升在复杂光线场景下真人活体检测通过率，同步提升高清2D和高质量3D道具的活体攻击防御效果； 3. 更新人脸识别模型，提升复杂光线场景下人脸识别的准确率； 4. 新增暗光恢复功能，对暗光场景下的图片进行亮度提升，提高检测和识别的准确率；

		5. 更新「证件照」模型，深度优化人证1:1比对场景准确率和泛化性; 6. 适配兼容奔腾G4400、赛扬J1900老旧CPU，支持X32/X64多操作系统，支持win7/win10/XP系统。
v4.2	2021.03.30	1. 增加端到端人脸识别示例工程，提供快速验证和测试的工具及二次开发的范例参考； 2. 增加人脸识别设置示例工程，提供人脸识别相关参数配置功能的参考可配合人脸识别工程使用。
v4.1	2020.07.10	1. 优化人脸检测、关键点检测模型，提升人脸检测准确率； 2. 增加口罩检测功能，支持对用户是否佩戴口罩这个属性进行检测； 3. 新增激活工具，该工具支持在连网状态下通过授权序列号完成激活; 4. 完善各功能接口调用的示例工程。
v4.0	2020.02.27	1. 全面升级人脸检测、关键点检测模型，提升人脸检测和追踪的准确率，解决部分场景下的人脸漏检问题； 2. 全面升级三模态活体检测模型，提升RGB、NIR、Depth三模态活体检测的准确率； 3. 全面优化人脸特征抽取和特征比对模型，提升人脸识别模型的准确率和泛化性; 4. 全面重构SDK的接口设计，降低二次开发难度的同时提升了二次开发的灵活性。

目 录

1、设计背景
2、名词解释
3、sdk简介
  3.1 功能架构
  3.2 人脸技术流程
  3.3 版本及兼容性
  3.4 依赖库及运行环境
4、sdk包结构
5、授权激活
  5.1 sdk自动激活
  5.2 激活工具激活(LicenseTool.exe)
  5.3 官网离线激活
6、sdk集成及demo示例工程
  6.1 sdk的集成
  6.2 sdk工程的集成及库文件说明
7、模型能力加载及模型说明
  7.1 模型删减说明
  7.2 模型路径的定制化
  7.3 能力定制化说明
    7.3.1 detect.json （人脸检测能力定制配置文件）
    7.3.2 track.json （人脸追踪能力定制配置文件）
    7.3.3 action_live.json （动作活体能力定制配置文件）
    7.3.4 crop.json (人脸抠图能力定制配置文件)
8、功能接口
  8.1 人脸检测detect接口
  8.2 人脸跟踪track接口
  8.3 清除人脸跟踪历史接口
  8.4 人脸关键点接口
  8.5 注意力检测接口
  8.6 人脸属性检测接口
  8.7 人脸抠图接口
  8.8 暗光恢复接口
  8.9 眼部状态检测接口
  8.10 嘴巴闭合检测接口
  8.11 口罩佩戴检测接口
  8.12 人脸质量
    8.12.1 人脸姿态角接口
    8.12.2 人脸光照检测接口
    8.12.3 人脸遮挡检测接口
    8.12.4 人脸模糊度检测接口
    8.12.5 最优人脸检测接口
  8.13 特征值及人脸比对（1:1）

8.13.1 人脸特征值接口
8.13.2 人脸活体特征值接口
8.13.3 深度人脸特征值接口
8.13.4 特征值比对接口
8.13.5 人脸1:1比对接口
8.14 动作活体和静默活体
8.14.1 动作活体接口
8.14.2 清除动作活体历史接口
8.14.3 rgb静默活体接口
8.14.4 nir静默活体接口
8.14.5 rgb+depth双目静默活体接口
8.15 人脸库管理
8.15.1 人脸注册接口(通过传入图片帧)
8.15.2 人脸注册接口(通过传入特征值)
8.15.3 人脸更新接口(传入opencv图片帧)
8.15.4 用户删除接口
8.15.5 创建用户组接口
8.15.6 用户组删除接口
8.15.7 用户信息查询接口
8.15.8 用户人脸图片查询接口
8.15.9 用户组列表查询接口
8.15.10 群组列表查询接口
8.15.11 人脸库人脸数量查询
8.15.12 人脸识别接口(1:N) (传入opencv图片帧)
8.15.13 人脸识别接口(1:N) (传入特征值)
8.15.14 人脸识别接口(1:N) (传入opencv图片帧)
8.15.15 人脸识别接口(1:N) (传入特征值)
8.16 sdk系统信息接口
8.16.1 获取sdk版本号接口
8.16.2 获取设备指纹接口
9、功能接口对应结构体描述
9.1 人脸跟踪信息结构体
9.2 人脸框信息结构体
9.3 人脸关键点信息结构体
9.4 人脸特征值结构体
9.5 人脸姿态角结构体
9.6 人脸属性信息结构体
9.7 嘴巴闭合结构体
9.8 口罩佩戴结构体
9.9 最优人脸置信度结构体
9.10 人脸模糊度置信度结构体
9.11 人脸光照置信度结构体
9.12 人脸遮挡置信度结构体
9.13 人眼闭合状态结构体
9.14 注意力结构体
9.15 静默活体置信度结构体
10、多端特征值对齐
11、适配的深度双目摄像头特别说明
12、错误码及错误信息
13、常见问题

1、设计背景

(1) **场景特点：**

- 网络：对于无网、局域网等情况，无法连接公网，API方式无法运作。如政府单位、金融保险、教育机构等，其中内网情况最为常见，私有化部署是项目开展的前提条件。
- 安全：即使可以连接外网，因为人脸数据的敏感性，许多客户不希望将人脸数据传入百度服务器，如大学学生照片、部分企业员工数据等，API形式也往往不被接受。

- 速度：由于各地网络线路、机房部署、图片采集方式等诸多原因，API形式往往耗时较高，容易存在部分请求耗时过长的情况，容易影响业务正常运转。
- 稳定：API形式容易受网络抖动、机房故障、线上连带bug等影响，存在一定的不稳定因素，可用性保障，往往成为在线调用最容易出现问题的地方。

(2) **客户特点**：

- 1：N-小型人脸库检索：多为通道通行、固定区域人群验证等需求，如写字楼闸机门禁、企业考勤打卡等，人脸库范围较小，且不易经常变动。
- 1：1-自有数据源对比：将当前采集的人脸，与其他数据源中的人脸进行对比，如身份证芯片照、教务系统图片、档案图片等，进行快速的1：1对比验证。

(3) **核心需求**：

- 基础的人脸采集：包含人脸检测、跟踪、捕获、质量校验等基础功能，获取符合识别条件的人脸。为之前的客户端SDK的标准功能，离线版本SDK保留以上所有能力。
- 本地特征抽取：所有在SDK中运行的人脸图片，都可以完成本地特征抽取，以便进行对比或识别操作。
- 1：1对比：支持两张图片的相似度对比，可直接传入图片，也可调用本地某个人脸特征；
- 1：N搜索：支持一定库大小的人脸查找，在指定的人脸集合中查找最相似的人脸，并返回相似度分值；

2、名词解释

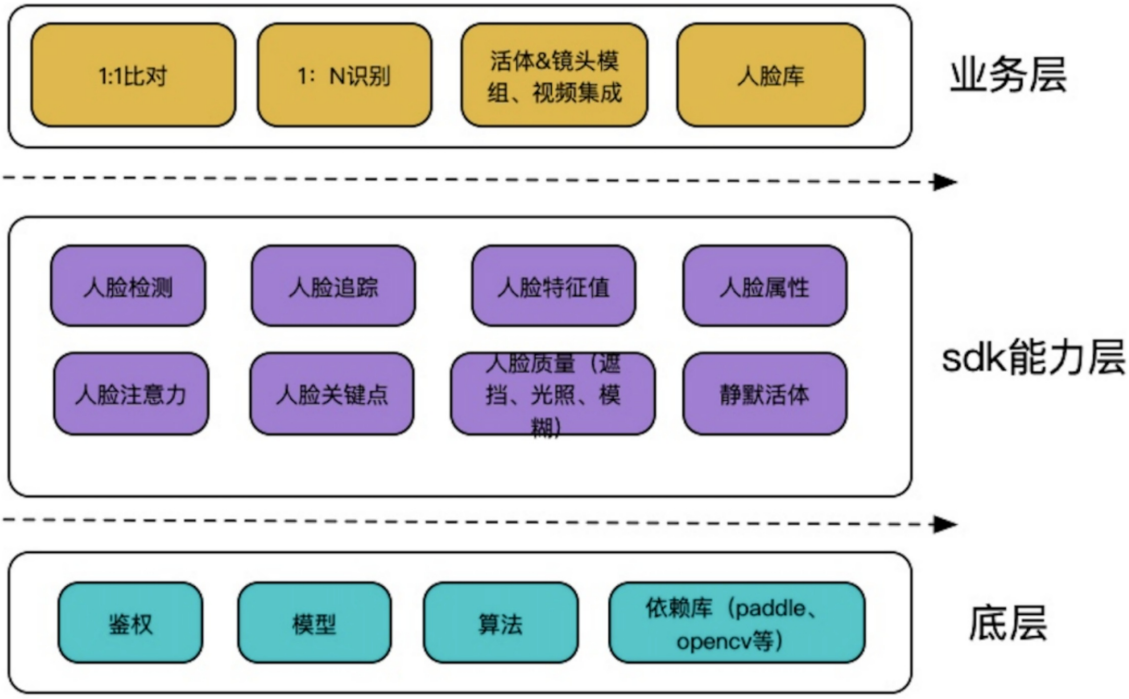
名词	定义
sdk	windows c++离线人脸识别sdk（须支持c++11）
vs2015	微软的开发工具visual studio 2015 (推荐安装vs community 2015)
license	人脸识别激活所需要的激活文件,文档介绍了三种激活方式
key	人脸激活所需的序列号，可从百度AI官网申请（ai.baidu.com）
feature	人脸特征值，用来表示人脸特征的128个float浮点值
landmark	人脸关键点（72个关键点）
face_token	对应人脸图片的唯一编码，若一个人上传了2张不同图片，则可能有2个不同的face_token,它和图片一一对应

3、sdk简介

本sdk适应于windows平台下的人脸识别系统，为支持c++语言开发的sdk,开发者可在vs2015下面进行开发（推荐使用，不保证其他版本vs都兼容）。sdk采用导出动态库dll的方式提供接口，另外随sdk附带一个示例工程FaceOfflineSdk，提供了sdk的各种能力及调用示例。

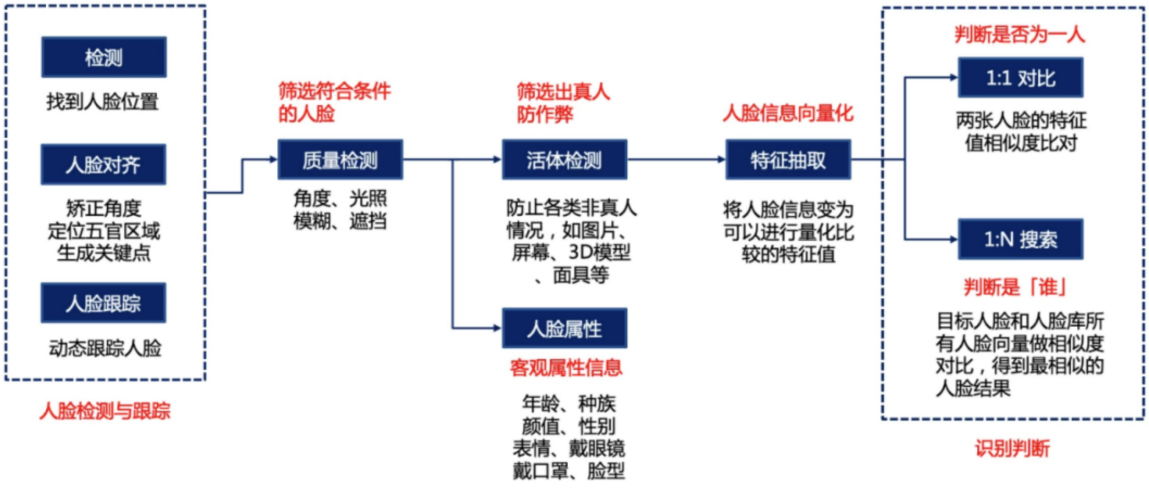
3.1 功能架构

sdk具有人脸检测、追踪、特征值、静默活体、人脸库、镜头模组集成等诸多功能。架构如下图所示：



3.2 人脸技术流程

sdk的人脸识别技术流程可如下图所示，通过人脸检测进行人脸特征值的提取，同时，在人脸检测或特征值提取之前可通过人脸质量判断和活体检测进行按需过滤。人脸信息变成可量化的特征值后，可实现人脸特征值的1:1比对和人脸的1：N识别（N个人脸特征信息可保存在人脸库中）



3.3 版本及兼容性

本sdk支持x86、x64两个平台的版本。

支持win7、win10操作系统（windows server能用但不推荐）。

针对赛扬、奔腾等cpu我们有noavx版本（通用版不适应这类cpu）。

推荐使用vs2015 community或professional或vs2015以上版本的开发工具。

sdk未提供debug的库文件，所以请选择release模式开发测试。

3.4 依赖库及运行环境

本sdk在vs环境下编译会在x64或x86（根据您选择的版本不同）下生成exe可执行文件。

依赖的其他dll库文件包括如opencv等都在该exe目录，请勿删除。

若exe执行提示找不到依赖dll库文件，可根据您的版本在sdk的`tools->vc_redist`目录执

行vc_redist.x64.exe或vc_redist.x86.exe进行安装（该vc_redst为vs2015的依赖）。

4、sdk包结构

sdk包结构如下图所示：

5、授权激活

sdk需要授权激活后才能正常使用，在sdk初始化后，若报错误码-13（错误码参考文档最后定义），一般为没有通过授权。

通常，sdk分按设备授权和按应用授权两种方式，大部分采用按设备授权的方式，按应用授权可针对批量大规模客户使用（文档中先只介绍按设备授权，按应用授权可工单或联系百度商务我们提供另外的文档或技术支持）。

按设备授权的方式中，sdk自动激活和激活工具激活需要设备能联网，若设备不能连外网，可采用官网离线激活的方式。

5.1 sdk自动激活

在sdk的目录中有license文件夹，里面存放了2个文件，license.key和license.ini，分别是授权key和授权文件，若在百度官网申请了授权系列号key（16位），可按sdk中的license文件夹中license.key原格式样子覆盖填写您申请的key。

在设备能联网的情况下，运行sdk会自动授权激活并拉取新的授权文件license.ini覆盖sdk中的旧文件。

5.2 激活工具激活(LicenseTool.exe)

在百度官网申请系列号key（16位）后，在FaceOfflineSdk的tools->license_tool目录夹下，双击运行LicenseTool.exe激活工具，可见如下所示界面。

在输入框中输入16位的系列号后，点击激活按钮，稍等片刻，即可见弹出的激活成功对话框，若激活失败，请参考对话框中弹出的message信息查找原因。激活成功后会生成license.zip文件,解压后获取license.ini和license.key文件，拿这2个文件拷贝覆盖到sdk中的license文件夹，覆盖对应旧文件，重启sdk，即可完成授权激活。

5.3 官网离线激活

若设备不能连外网、通过授权还有另外一种方法，即通过如上5.2的激活工具，获取到设备硬件指纹信息，通过百度官网填入指纹信息和申请到的系列号key，可完成激活并下载获取到license.ini文件和license.key文件，把这2个文件拷贝到FaceOfflineSdk的license目录下，重新运行sdk亦可通过授权激活。

6、sdk集成及demo示例工程

6.1 sdk的集成

sdk集成的include需包含sdk的接口头文件：baidu_face_api.h；

引入库lib文件：BaiduFaceApi.lib、face_sdk.lib；

运行动态库dll文件以及部分底层算法库文件：BaiduFaceApi.dll、face_sdk.dll、bd_license.dll、libcurl.dll、libeay32.dll、libopenblas.dll、libiomp5md.dll、mkldnn.dll、mklml.dll、ssleay32.dll。（这些库文件在sdk中的win32或x64目录）；

除此之外，第三方头文件及库文件也建议保留,如json、opencv。

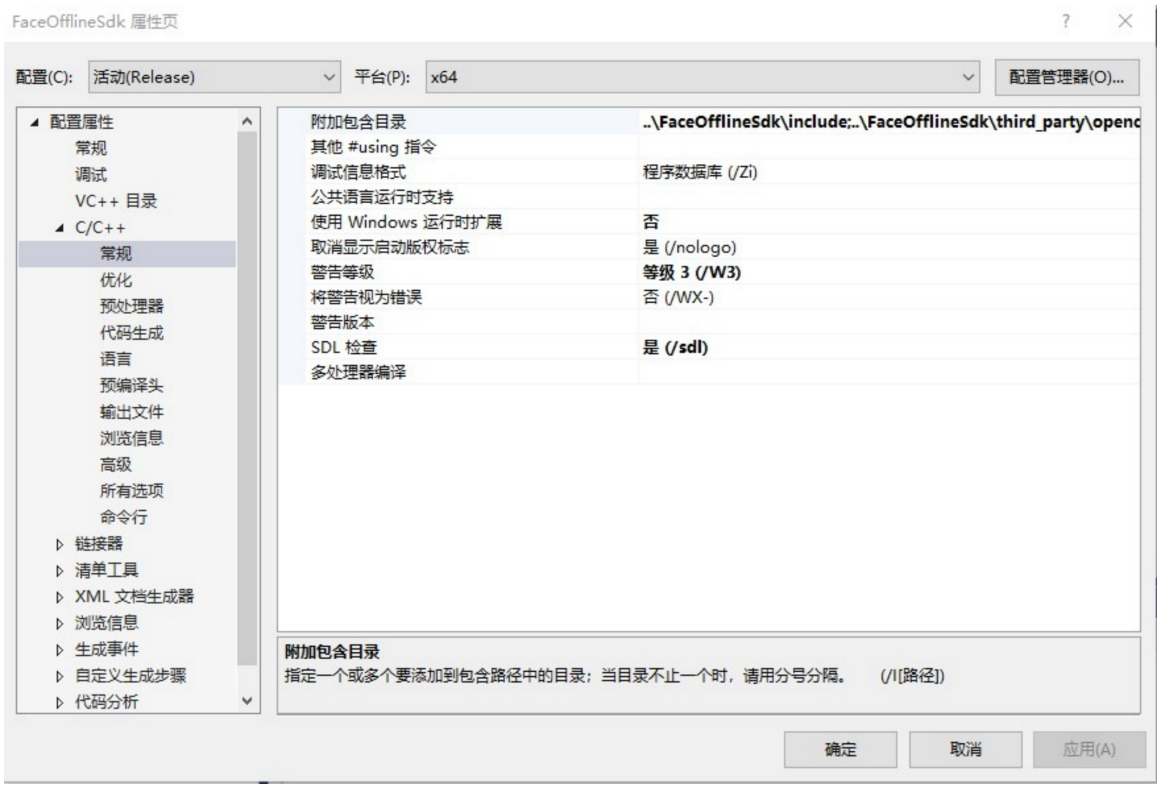
6.2 sdk工程的集成及库文件说明

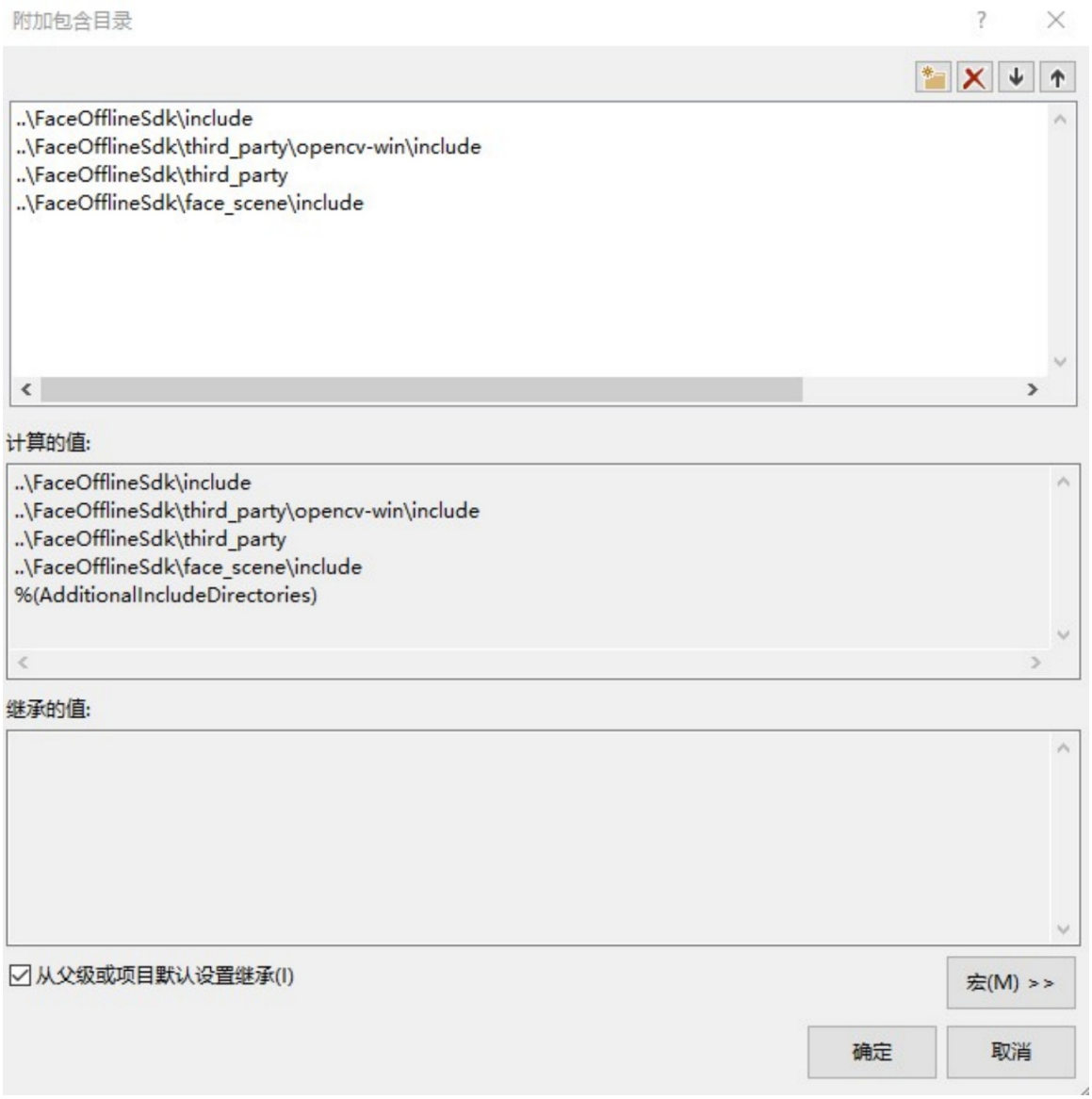
sdk工程FaceOfflineSdk包含了sdk接口及demo示例工程。其中：sdk接口头文件为前述include目录的baidu_face_api.h，该文件定义了sdk的各api接口方法。sdk接口的lib库文件为前述lib目录。根据x64和x86分别放置在不同目录。其中的lib文件描述如下列表：

lib文件名称	lib文件说明	对应动态库dll	是否可删除
BaiduFaceApi.lib	百度人脸库引入文件	BaiduFaceApi.dll	否
face_sdk.lib	百度人脸库引入文件	face_sdk.dll	否
opencv_world320.lib	opencv库文件，用来显示图片，视频等	opencv_ffmpeg320.dll	否
json_vc71_libmt.lib	json cpp库文件，用来做json解析	无	是（若不使用人脸库等用到json的部分，可删除该库）
OrbeCamera.lib	3d结构光镜头：奥比中光引入库文件	对应的动态库在camera_driver文件夹的orbe文件夹里面，可根据x64或x86选择64位和32位，使用的时候需要把dll的文件夹里面的内容全部拷贝到exe所在目录	是（若不用奥比中光3d结构光做深度活体检测，可删除该库），另外include里面的头文件orbe_camera.h也可删除
AimiCamera.lib	3d结构光镜头：华捷艾米引入库文件	对应的动态库在camera_driver文件夹的hjimi文件夹里面，可根据x64或x86选择64位和32位，使用的时候需要把dll的文件夹里面的内容全部拷贝到exe所在目录	是（若不用奥比中光3d结构光做深度活体检测，可删除该库），另外include里面的头文件aimi_camera.h也可删除

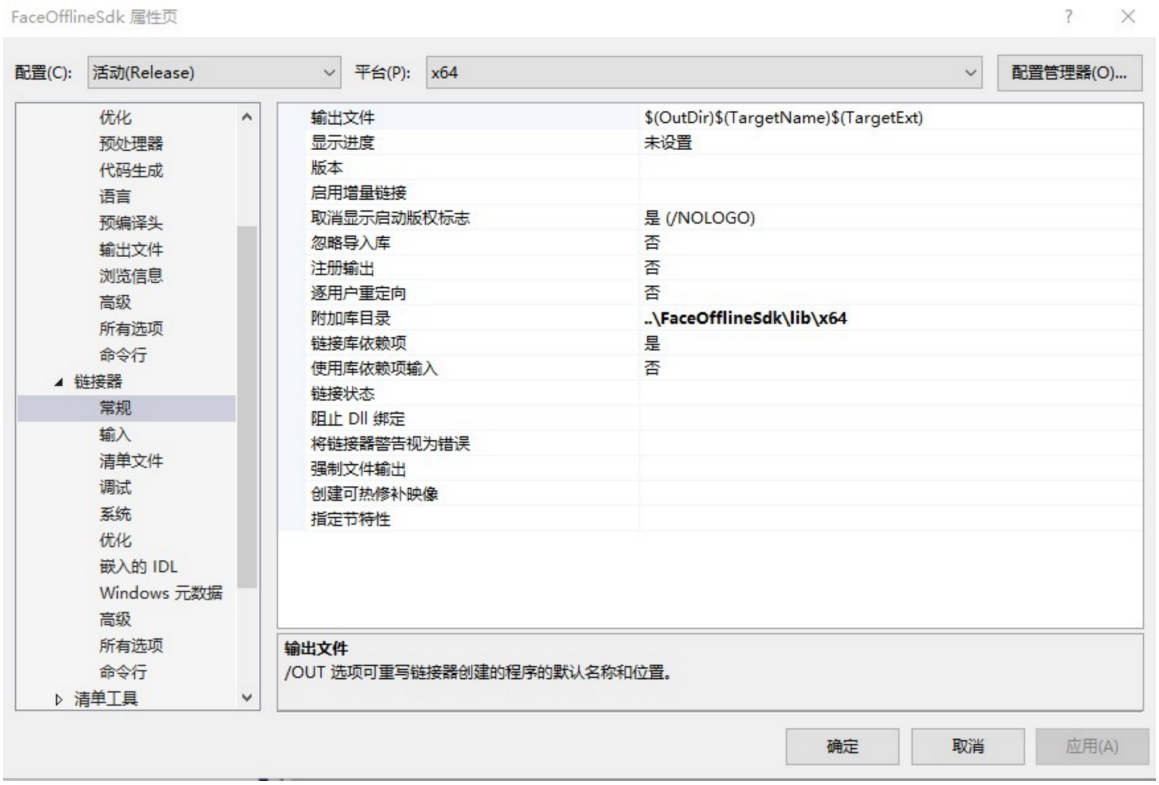
用vs2015打开sln工程文件后，默认便已设置好工程的`include`及`lib`库路径。若需要手动设置，则`include`如下图所示：

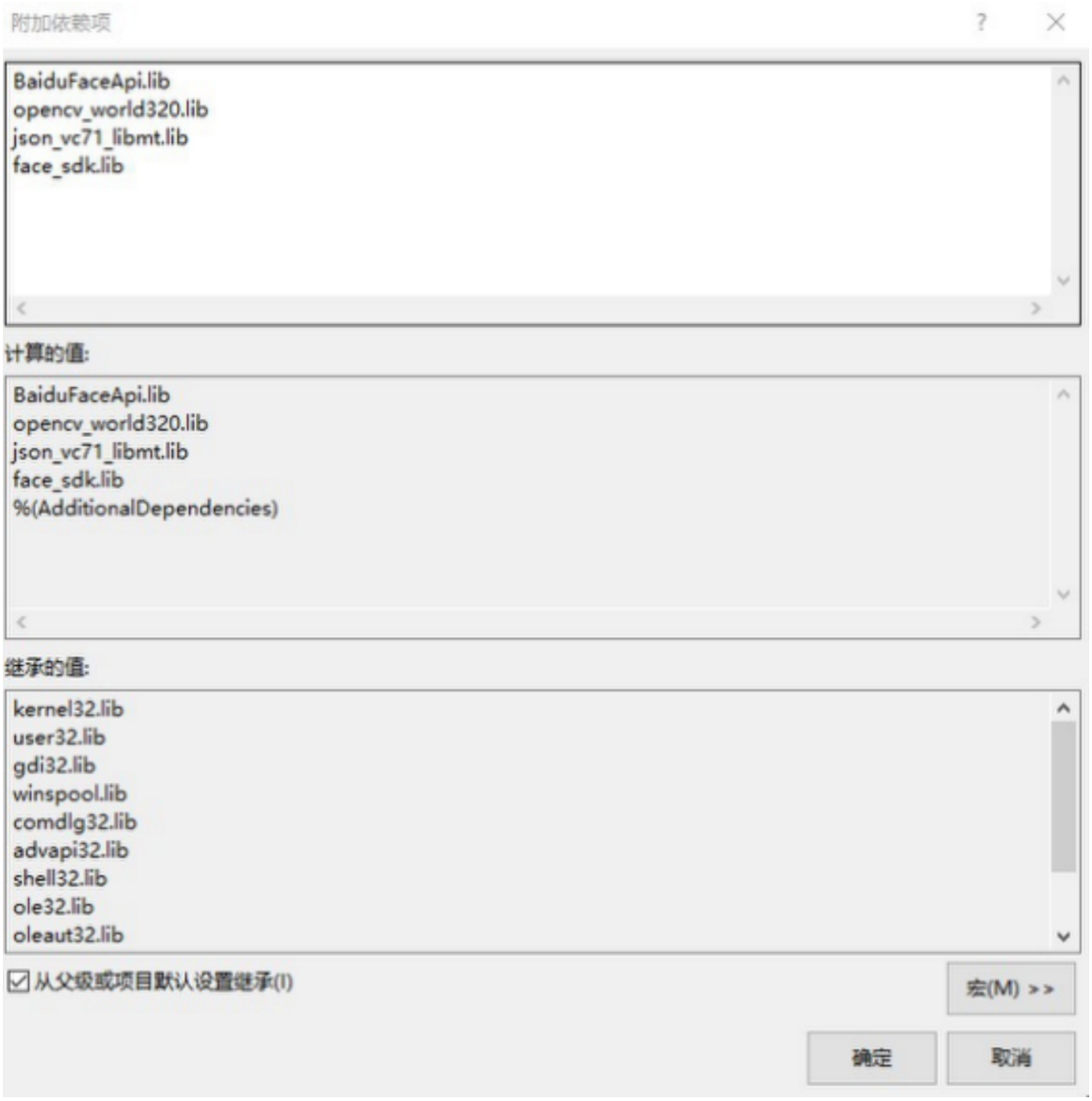
右键工程->属性，在c++常规中添加包含目录。(`include`的文件夹的头文件为sdk的使用头文件，建议接入客户自己工程时候附带，`third_party`中为`json`和`opencv`的头文件，也建议引入)





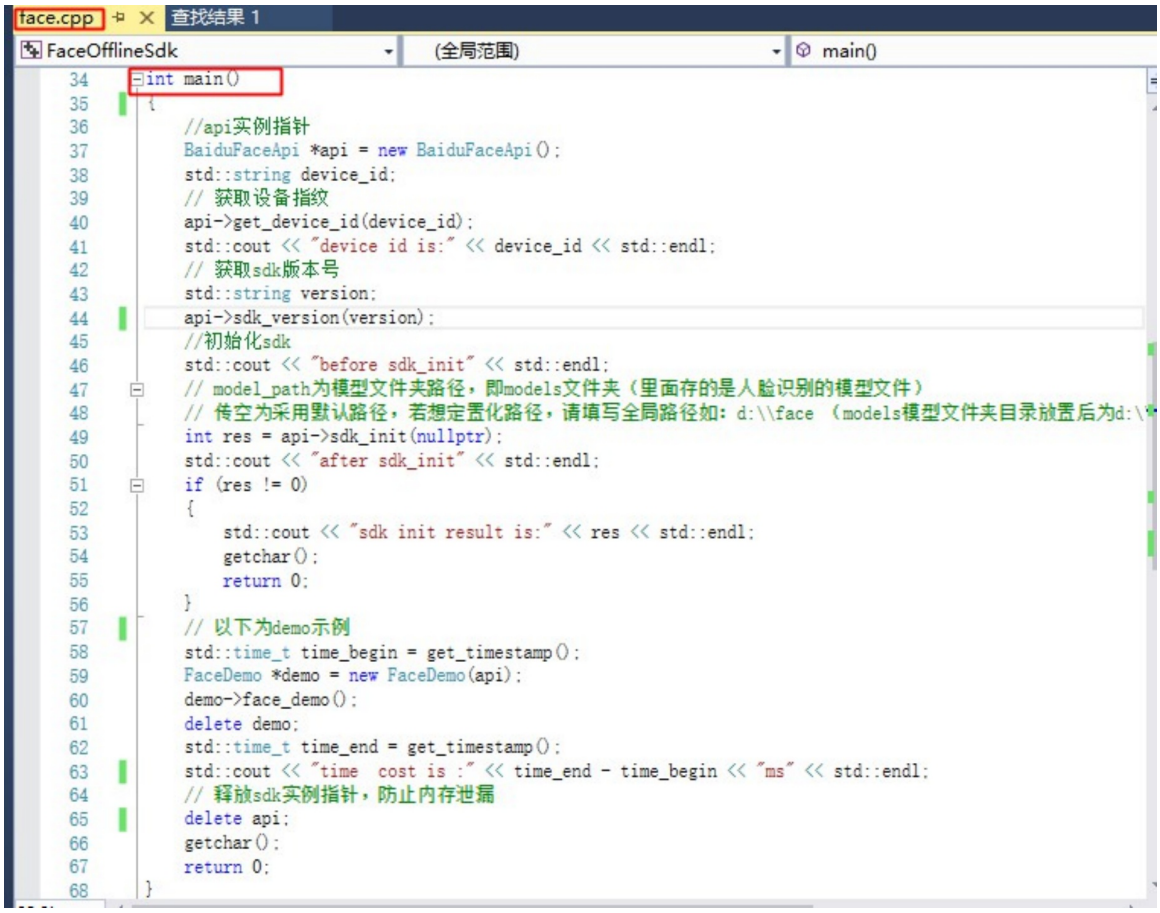
同样：右键工程->属性，链接器->常规中输入如（假设64位，32位请输入x86）





demo示例工程FaceOfflineSdk展示了如何集成百度人脸识别离线sdk，并调用sdk的方法及各示例demo等。

在FaceOfflineSdk中的face.cpp的main()方法中，有使用sdk的各个接口方法示例。接入sdk及其简单，如下图及解释：



sdk使用主要三步：1）初始化实例指针；2）初始化sdk；3) 调用示例demo，实现功能。

sdk末尾需要释放sdk实例化指针`delete api`。初始化sdk返回值若不为0，可通过文档查询对应错误码，查找sdk初始化失败原因。

另外，可通过`is_auth()`方法查看是否通过了授权，通常，需要通过授权，才可以使用sdk的各种能力。

示例工程中：分别有以下几个文件夹放置文件对应几个常用sdk的调用demo。

解析如下：

文件夹或文件名	文件说明
face.cpp	sdk总入口，初始化，模型加载，销毁等,见main方法
face_demo.cpp	demo总入口，可打开注释运行各类demo示例
face_detect	人脸检测接口及示例demo
face_track	人脸跟踪接口及示例demo
face_feature	人脸特征值接口及示例demo
face_compare	人脸1:1 1:N 比对，特征值比较等接口及示例demo
face_liveness	可见光RGB、红外IR活体检测、RGB&IR双目摄像头静默活体检测，RGB&DEPTH双目摄像头静默活体检测等
face_manager	人脸库管理类，包括人脸注册、删除，更新、组管理，人脸信息查询等
face_attr	人脸属性(年龄、性别、种族等) 接口及示例
face_head_pose	人脸姿态角接口及示例
face_illumination	人脸光照检测接口及示例
face_occlusion	人脸遮挡度检测接口及示例
face_blur	人脸模糊度检测接口及示例
face_action_live	动作活体检测接口及示例
face_best	优人脸接口及示例
face_crop	人脸抠图接口及示例
face_gaze	双眼注意力检测接口及示例
face_eye_close	眼睛闭合检测接口及示例
face_mouth_close	嘴巴闭合检测接口及示例
face_mouth_mask	是否佩戴口罩检测接口及示例
face_scene	该示例亦展示了如何使用sdk原子接口等，如不用该示例，则引入的头文件处 face_scene\include可删除

7、模型能力加载及模型说明

7.1 模型删减说明

sdk支持按需配置模型和加载能力，若sdk有某部分功能不需要使用，可尝试删除一些模型，删除后模型即不会加载也不会占用内存。sdk中models文件夹里的模型及是否可删减说明如下表:

模型文件夹名称	说明	是否可删减	删减说明
detect	人脸检测模型	是	若不使用nir近红外功能，可删除detect_nir开头的模型文件
align	人脸关键点模型	是	若不使用nir近红外功能，可删除align_nir开头的模型文件
attribute	人脸属性	是	若不使用人脸属性检测功能，该文件夹可删除
best_image	最佳人脸	是	若不使用最佳人脸检测功能，该文件夹可删除
blur	人脸质量模糊度检测	是	若不使用该功能，该文件夹可删除
dark_enhance	暗光恢复	是	若不使用该功能，该文件夹可删除
eye_close	眼睛闭合	是	若不使用眼睛闭合及动作活体功能，该文件夹可删除
feature	人脸特征值	是	若仅使用rgb可见光人脸特征值，除feature_life_float32_paddle.encrypted文件外，该文件夹其他文件可删除
gaze	人脸注意力检测	是	若不使用该功能，该文件夹可删除
mouth_close	嘴巴闭合检测	是	若不使用嘴巴闭合及动作活体功能，该文件夹可删除
mouth_mask	口罩佩戴检测	是	若不使用该功能，该文件夹可删除
occlusion	人脸遮挡检测	是	若不使用该功能，该文件夹可删除
silent_live	静默活体检测	是	若只使用rgb可见光单目静默活体，则除silent_live_rgb_float32_paddle.encrypted文件外其他皆可删除

7.2 模型路径的定制化

sdk支持模型文件夹`models`的路径自定义，当sdk初始化`api->sdk_init(nullptr)`传null时候，为sdk支持模型文件夹路径在默认路径，即`models`在sdk现有位置。同时也支持models通过`sdk_init`中传入绝对路径。若把`models`文件夹拷贝到d盘的face文件夹下面，则可定义：

```
api->sdk_init(“d:\\face”);
```

此时，授权文件夹`license`也需要随之变为d盘的`face`文件夹下面。否则会出现授权不通过的问题。 另外，若使用了能力自定义的`config`文件夹，也需要拷贝到d盘的`face`文件夹下面。否则能力定制化也不会生效而是使用的系统默认。

7.3 能力定制化说明

sdk支持能力自定义、通过读取配置文件的方式进行能力定制化。sdk默认能力加载无需定制化，若需要定制化，请把sdk根目录里面的`conf`文件夹重命名为`config`文件夹。并且在sdk中，把里面的`json`配置按如下说明做修改，可起到定制化能力加载的效果，配置文件简要说明如下（若需定制化修改，请参考示例`json`，修改`json`字段的值来达到定制化的目的）

7.3.1 detect.json （人脸检测能力定制配置文件） 配置文件名：`detect.json`

说明：人脸检测自定义能力配置文件

```
{
  "max_detect_num":5,
  "min_face_size":0,
  "scale_ratio":-1,
  "not_face_thr":0.5
}
```

参数	类型	说明
max_detect_num	int	最大检测的人脸数量，最多支持50，最小1,默认5
min_face_size	int	默认0，可用来设置检测的最小人脸，比如可设为10，则表示小于10*10的人脸，sdk检测不到
scale_radio	int	默认-1，表示进行人脸检测时候的图片缩放比率。建议用-1表示传入原图sdk自己缩放（为保证检测效果，该参数建议用默认）
not_face_thr	float	默认0.5，表示非人脸的阈值，取值范围0-1

7.3.2 track.json （人脸追踪能力定制配置文件） 配置文件名：track.json

说明：人脸追踪自定义能力配置文件

```
 {
   "detect_intv_before_track":0.02,
   "detect_intv_during_track":0.02
}
```

参数	类型	说明
detect_intv_before_track	float	表示人脸追踪前进行人脸检测的时间间隔（单位毫秒）
detect_intv_during_track	float	表示人脸追踪时候进行人脸检测的时间间隔（单位毫秒）

7.3.3 action_live.json （动作活体能力定制配置文件） 配置文件名：action_live.json

说明：人脸动作活体自定义能力配置文件

```
 {
   "eye_open_threshold":0.5,
   "eye_close_threshold":0.5,
   "mouth_open_threshold":0.5,
   "mouth_close_threshold":0.5,
   "look_up_threshold":0.5,
   "look_down_threshold":0.5,
   "turn_left_threshold":0.5,
   "turn_right_threshold":0.5,
   "nod_threshold":0.5,
   "shake_threshold":0.5,
   "max_cache_num":1
}
```

参数	类型	说明
eye_open_threshold	float	表示人脸动作活体眼睛睁开的置信度阈值
eye_close_threshold	float	表示人脸动作活体眼睛闭合的置信度阈值
mouth_open_threshold	float	表示人脸动作活体嘴巴张开的置信度阈值
mouth_close_threshold	float	表示人脸动作活体嘴巴闭合的置信度阈值
look_up_threshold	float	表示人脸动作活体抬头的置信度阈值
look_down_threshold	float	表示人脸动作活体低头的置信度阈值
turn_left_threshold	float	表示人脸动作活体向左转头的置信度阈值
turn_right_threshold	float	表示人脸动作活体向右转头的置信度阈值
nod_threshold	float	表示人脸动作活体点头的置信度阈值
shake_threshold	float	表示人脸动作活体摇头的置信度阈值
max_cache_num	float	最大缓存数、推荐为1不做修改

7.3.4 crop.json (人脸抠图能力定制配置文件) 配置文件名：`crop.json`

说明：人脸抠图自定义能力配置文件

```
 {
   "is_flat":0,
"crop_size":200,
"enlarge_ratio":1
}
```

参数	类型	说明
is_flat	int	默认为0，表示是否是镜像，该参数目前无效
crop_size	int	表示抠图的大小，如200，则表示抠出来的是200*200的图片
enlarge_ratio	float	默认是1，若小于1，比如0.8，图片有点放大，会稍微模糊，建议自定义可设为1

8、功能接口

sdk功能接口的调用可参考各示例cpp文件，接口定义在`baidu_face_api.h`

sdk实现的主要功能有人脸实时跟踪检测、人脸特征值提取、动作活体、RGB&IR静默活体检测、RGB&DEPTH静默活体检测、人脸注册、人脸更新、组管理、用户管理以及1:1人脸对比,1:N人脸识别、特征值的比对和通过usb或笔记本自带摄像头检测视频帧，返回识别出的人脸信息、人脸属性等，另外支持对人脸检测进行能力加载设置，达到根据设置进行识别的目的。

各接口功能及传入参数和返回结果等定义如下：

8.1 人脸检测detect接口

方法名：`detect`

说明：人脸检测，返回人脸信息

函数：`int detect(std::vector<FaceBox> &out, const cv::Mat* mat, int type = 0)`

请求参数	说明	必须	类型	示例描述
out	通过引用返回的人脸结构体数组	是	std::vector	人脸框信息结构体
mat	传入的opencv视频帧	是	Opencv mat	请参考示例
type	检测类型（传0表示rgb可见光人脸检测，1表示nir）	否	int	0或1，不传默认为0
返回信息	函数的返回	是	int	<=0 为未检测到人脸或错误码 >0时候为检测到的人脸数量

8.2 人脸跟踪track接口

方法名：`track`

说明：人脸跟踪，返回人脸信息

函数：`int track(std::vector<TrackFaceInfo>&out, const cv::Mat* mat, int type = 0)`

请求参数	说明	必须	类型	示例描述
out	通过引用返回的人脸结构体数组	是	std::vector	人脸跟踪结构体信息
mat	传入的opencv视频帧	是	Opencv mat	请参考示例
type	检测类型（传0表示rgb可见光人脸检测，1表示nir）	否	int	0或1，不传默认为0
返回信息	函数的返回	是	int	<=0 为未检测到人脸或错误码 >0时候为检测到的人脸数量

8.3 清除人脸跟踪历史接口

方法名：`clear_track_history`

说明：清除人脸跟踪的历史信息

函数：`int clear_track_history(int type = 0)`

请求参数	说明	必须	类型	示例描述
type	检测类型（传0表示rgb可见光人脸检测，1表示nir）	否	int	0或1，不传默认为0
返回信息	函数的返回	是	void	

8.4 人脸关键点接口

方法名：`face_landmark`

说明：人脸关键点，返回人脸关键点信息

函数：`int face_landmark(std::vector<Landmarks>&out, const cv::Mat* mat, int type)`

请求参数	说明	必须	类型	示例描述
out	通过引用返回的人脸关键点数组	是	std::vector	人脸关键点信息结构体
mat	传入的opencv视频帧	是	Opencv mat	请参考示例
type	检测类型（传0可见光生活照特征值，1、可见光证件照特征值 2、表示近红外特征值）	是	int	
返回信息	函数的返回	是	int	<=0 为未检测到人脸或错误码 >0时候为检测到的人脸数量

8.5 注意力检测接口

方法名：`face_gaze`

说明：人脸注意力检测

函数：`int face_gaze(std::vector<GazeInfo>& out, const cv::Mat* mat)`

请求参数	说明	必须	类型	示例描述
out	通过引用返回的人脸注意力结构体数组	是	std::vector	注意力结构体
mat	传入的opencv视频帧	是	Opencv mat	请参考示例
返回信息	函数的返回	是	int	<=0 为未检测到人脸或错误码 >0时候为检测到的人脸数量

8.6 人脸属性检测接口

方法名：`face_attr`

说明：人脸属性检测

函数：`int face_attr(std::vector<Attribute>& out, const cv::Mat *mat)`

请求参数	说明	必须	类型	示例描述
out	通过引用返回的人脸属性结构体数组	是	std::vector	人脸属性结构体
mat	传入的opencv视频帧	是	Opencv mat	请参考示例
返回信息	函数的返回	是	int	<=0 为未检测到人脸或错误码 >0时候为检测到的人脸数量

8.7 人脸抠图接口

方法名：`face_crop`

说明：人脸扣图，返回人脸扣图（仅支持单人脸抠图）

函数：`int face_crop(cv::Mat&out, const cv::Mat* mat)`

请求参数	说明	必须	类型	示例描述
out	抠图结果图片帧	是	Opencv mat	
mat	传入的opencv视频帧	是	Opencv mat	
返回信息	函数的返回	是	int	返回：<0 为未检测到人脸或错误码 =0 表示抠图成功

8.8 暗光恢复接口

方法名：`dark_enhance`

说明：暗光恢复，用于图片比较暗的使之变亮利于人类检测（仅支持单人脸抠图）

函数：`int dark_enhance(cv::Mat&out, const cv::Mat* mat)`

请求参数	说明	必须	类型	示例描述
out	暗光恢复结果图片帧	是	Opencv mat	
mat	传入的opencv视频帧	是	Opencv mat	
返回信息	函数的返回	int	返回：<0 为错误码 =0表示成功	

8.9 眼部状态检测接口

方法名：`face_eye_close`

说明：人脸眼部状态检测

函数：`int face_eye_close(std::vector<EyeClose> &out, const cv::Mat* mat)`

请求参数	说明	必须	类型	示例描述
out	通过引用返回的人脸眼部状态结构体数组	是	std::vector	眼部状态信息结构体
mat	传入的opencv视频帧	是	Opencv mat	请参考示例
返回信息	函数的返回	是	int	<=0 为未检测到人脸或错误码 >0时候为检测到的人脸数量

8.10 嘴巴闭合检测接口

方法名：`face_mouth_close`

说明：人脸嘴巴闭合检测

函数：`int face_mouth_close(std::vector<MouthClose>& out, const cv::Mat* mat)`

请求参数	说明	必须	类型	示例描述
out	通过引用返回的人脸嘴巴闭合结构体数组	是	std::vector	嘴巴闭合结构体
mat	传入的opencv视频帧	是	Opencv mat	请参考示例
返回信息	函数的返回	是	int	<=0 为未检测到人脸或错误码 >0时候为检测到的人脸数量

8.11 口罩佩戴检测接口

方法名：face_mouth_mask

说明：人脸口罩佩戴检测

函数：int face_mouth_mask(std::vector<MouthMask>& out, const cv::Mat* mat)

请求参数	说明	必须	类型	示例描述
out	通过引用返回的人脸口罩佩戴结构体数组	是	std::vector	佩戴口罩结构体
mat	传入的opencv视频帧	是	Opencv mat	请参考示例
返回信息	函数的返回	是	int	<=0 为未检测到人脸或错误码 >0时候为检测到的人脸数量

8.12 人脸质量

人脸质量判断可由以下几个因素自由组合综合判断，如姿态角、光照、遮挡、模糊以及最佳人脸。

8.12.1 人脸姿态角接口 方法名：face_head_pose

说明：人脸姿态角检测（yaw：左右边转角 roll：屏幕旋转角 pitch：上下偏转角 姿态角推荐范围 -15~15，越大角度越不正不合格）

函数：int face_head_pose(std::vector<HeadPose>&out, const cv::Mat* mat)

请求参数	说明	必须	类型	示例描述
out	通过引用返回的人脸姿态角数组	是	std::vector	人脸姿态角信息结构体
mat	传入的opencv视频帧	是	Opencv mat	请参考示例
返回信息	函数的返回	是	int	<=0 为未检测到人脸或错误码 >0时候为检测到的人脸数量

8.12.2 人脸光照检测接口 方法名：face_illumination

说明：人脸光照检测、（光照分值0-255、分值越大光照越强，推荐阈值100）

函数：int face_illumination(std::vector<Illumination>&out, const cv::Mat* mat)

请求参数	说明	必须	类型	示例描述
out	通过引用返回的人脸光照信息数组	是	std::vector	光照置信度结构体
mat	传入的opencv视频帧	是	Opencv mat	请参考示例
返回信息	函数的返回	是	int	<=0 为未检测到人脸或错误码 >0时候为检测到的人脸数量

8.12.3 人脸遮挡检测接口 方法名：face_occlusion

说明：人脸遮挡检测（遮挡分值0-1，分值越大遮挡度越高，推荐阈值0.6）

函数：int face_occlusion(std::vector<Occlusion>& out, const cv::Mat* mat)

请求参数	说明	必须	类型	示例描述
out	通过引用返回的人脸遮挡结构体数组	是	std::vector	遮挡置信度结构体
mat	传入的opencv视频帧	是	Opencv mat	请参考示例
返回信息	函数的返回	是	int	<=0 为未检测到人脸或错误码 >0时候为检测到的人脸数量

8.12.4 人脸模糊度检测接口 方法名：face_blur

说明：人脸模糊检测(模糊度分值0-1，分值越大模糊度越高，推荐阈值0.6)

函数：int face_blur(std::vector<Blur> &out, const cv::Mat * mat)

请求参数	说明	必须	类型	示例描述
out	通过引用返回的人脸模糊结构体数组	是	std::vector	模糊度结构体
mat	传入的opencv视频帧	是	Opencv mat	请参考示例
返回信息	函数的返回	是	int	<=0 为未检测到人脸或错误码 >0时候为检测到的人脸数量

8.12.5 最优人脸检测接口 方法名：face_best

说明：最佳人脸检测(最优人脸分值0-100，分值越高，最佳人脸图片得分越高，推荐阈值50)

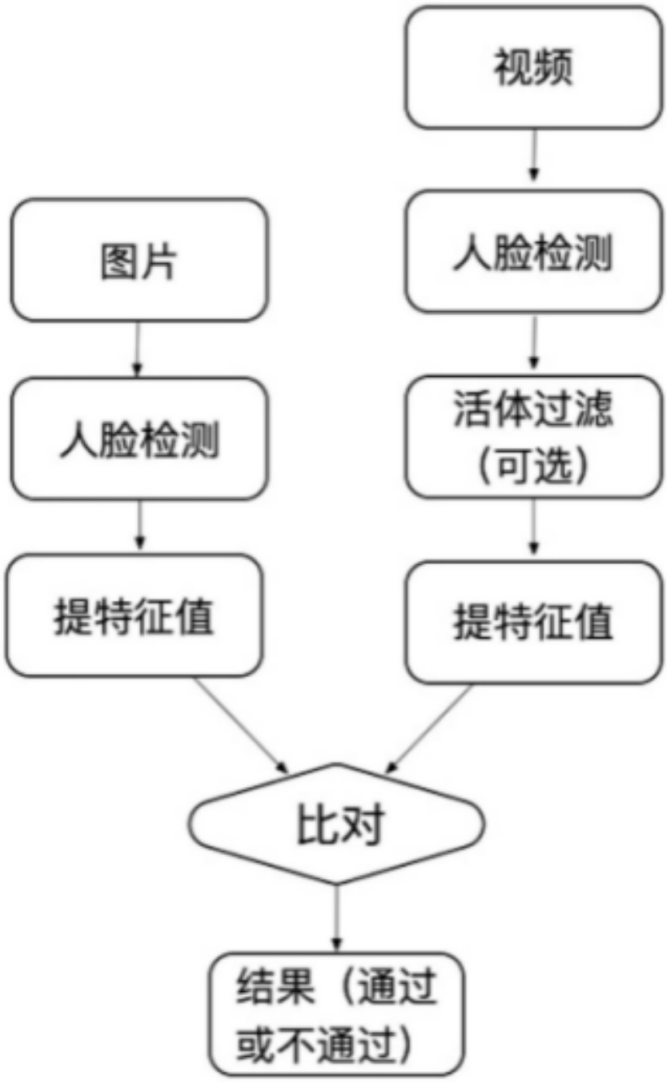
函数：int face_best(std::vector<Best> &out, const cv::Mat *mat)

请求参数	说明	必须	类型	示例描述
out	通过引用返回的最佳人脸结构体数组	是	std::vector	最优人脸结构体
mat	传入的opencv视频帧	是	Opencv mat	请参考示例
返回信息	函数的返回	是	int	<=0 为未检测到人脸或错误码 >0时候为检测到的人脸数量

8.13 特征值及人脸比对（1:1）

人脸比对的原理实际是特征值比对，通过提取图片中的人脸特征值，根据特征值调用compare_feature接口进行比对，推荐比对分值超过80分为同一人，可根据实际检测比对情况动态调整。

人脸1:1比对流程可如下图，实现如人证比对功能。（人的照片和实时视频比对，可根据使用情况选择是否启用质量检测）



8.13.1 人脸特征值接口 方法名：face_feature

说明：人脸特征值提取，并返回人脸信息

函数：int face_feature(std::vector<Feature> &out_fea, std::vector<FaceBox> &out_box, const cv::Mat *mat , int type = 0)

请求参数	说明	必须	类型	示例描述
out_fea	通过引用返回的人脸特征值结构体数组	是	std::vector	人脸特征值结构体信息
out_box	通过引用返回的人脸框结构体数组	是	std::vector	人脸框结构体信息
mat	传入的opencv视频帧	是	Opencv mat	请参考示例
type	检测类型（传0可见光生活照特征值，包含1寸照，1、网格证件照特征值2、表示近红外特征值）	否	int	0或1、2,未传默认为0
返回信息	函数的返回	是	int	<=0 为未检测到人脸或错误码 >0时候为检测到的人脸数量

8.13.2 人脸活体特征值接口 方法名：liveness_feature

说明：人脸特征值提取，并返回活体分值，人脸信息

函数：int liveness_feature(std::vector<Feature> &out_fea, std::vector<FaceBox> &out_box, float& score, const cv::Mat *mat , int type = 0)

请求参数	说明	必须	类型	示例描述
out_fea	通过引用返回的人脸特征值结构体数组	是	std::vector	人脸特征值结构体信息
out_box	通过引用返回的人脸框结构体数组	是	std::vector	人脸框结构体信息
mat	传入的opencv视频帧	是	Opencv mat	请参考示例
score	返回的活体分值	是	float	
type	检测类型（传0可见光特征值，1表示近红外特征值）	否	int	0或1,未传默认为0
返回信息	函数的返回	是	int	<=0 为未检测到人脸或错误码 >0时候为检测到的人脸数量

8.13.3 深度人脸特征值接口 方法名：rgbd_feature

说明：人脸特征值提取，返回人脸信息

函数：int rgbd_feature(std::vector<Feature> &out_fea, std::vector<Feature> &out_box, const cv::Mat* rgb_mat, const cv::Mat* depth_mat)

请求参数	说明	必须	类型	示例描述
out_fea	通过引用返回的人脸特征值结构体数组	是	std::vector	人脸特征值结构体信息
out_box	通过引用返回的人脸框结构体数组	是	std::vector	人脸框结构体信息
rgb_mat	传入可见光的opencv视频帧	是	Opencv mat	请参考示例
depth_mat	传入深度的opencv视频帧	是	int	请参考示例
返回信息	函数的返回	是	int	<=0 为未检测到人脸或错误码 >0时候为检测到的人脸数量

8.13.4 特征值比对接口 方法名：compare_feature

说明：人脸跟踪，返回人脸信息

函数：float compare_feature(Feature* f1, Feature *f2, int type)

请求参数	说明	必须	类型	示例描述
f1	人脸特征值结构体	是	Feature*	人脸特征值结构体信息(仅支持单个结构体的比对)
f2	人脸特征值结构体	是	Feature*	人脸特征值结构体信息(仅支持单个结构体的比对)
type	特征值类型（0,1,2）	否	int	0：生活照、包含1寸照 1：网格证件照 2：近红外特征值 未传默认为0(1寸或2寸的证据照也推荐当作生活照模式)
返回信息	函数的返回	是	float	特征值比对分值

8.13.5 人脸1:1比对接口 方法名：match

说明：人脸比对，返回人脸比对分值

函数：`int match(const cv::Mat& img1, const cv::Mat& img2, int type)`

请求参数	说明	必须	类型	示例描述
img1	传入可见光的opencv图片帧	是	cv::Mat	
img2	传入可见光的opencv图片帧	是	cv::Mat	
type	比对类型（0,1,2）	否	int	0：生活照、包含1寸照 1：网格证件照 2：近红外特征值 未传默认为0(1寸或2寸的证据照也推荐当作生活照模式)
返回信息	函数的返回	是	int	人脸比对分值（同特征值比对分值结果应该是float，这里返回int是做了四舍五入）

8.14 动作活体和静默活体

8.14.1 动作活体接口 方法名：`face_action_live`

说明：动作活体（眨眨眼、张张嘴等动作校验活体）

函数：`int face_action_live(std::vector<FaceBox>& out, int action_type, int& action_result, const cv::Mat* mat)`

请求参数	说明	必须	类型	示例描述
out	返回的人脸框结构体数组	是	std::vector	人脸框结构体
action_type	传入的活体动作	是	int	动作活体类型，如0表示眨眨眼，1表示张嘴等
action_result	动作活体的返回结果	是	int	返回结果为1表示存在该动作活体
mat	传入的opencv视频帧	是	Mat	视频帧
返回字段描述				
返回字段	int，>=0返回人脸个数，<0时候为错误码			
返回示例				

8.14.2 清除动作活体历史接口 方法名：`action_live_clear_history`

说明：清除动作活体历史

函数：`int action_live_clear_history()`

请求参数	说明	必须	类型	示例描述
返回字段描述				
返回字段	0表示成功，其他错误码 int			
返回示例				

8.14.3 rgb静默活体接口 方法名：`rgb_liveness`

说明：rgb可见光单目静默活体（推荐rgb+nir或rgb+depth进行双目活体校验）

函数：`int rgb_liveness(std::vector<TrackFaceInfo>& out, float &score, const cv::Mat* mat)`

请求参数	说明	必须	类型	示例描述
out	返回的人脸框结构体数组	是	std::vector	人脸框结构体（多人则返回最大人脸）
score	返回的动作活体分值	是	float	
mat	传入的opencv视频帧	是	Mat	视频帧

返回字段描述	
返回字段	int，>=0返回人脸个数，<0时候为错误码
返回示例	

8.14.4 nir静默活体接口 方法名：`nir_liveness`

说明：Nir近红外单目静默活体（推荐`rgb+nir`或`rgb+depth`进行双目活体校验）

函数：`int nir_liveness(std::vector<TrackFaceInfo>& out, float &score, const cv::Mat* mat)`

请求参数	说明	必须	类型	示例描述
out	返回的人脸框结构体数组	是	std::vector	人脸框结构体（多人则返回最大人脸）
score	返回的动作活体分值	是	float	
mat	传入的opencv视频帧	是	Mat	视频帧

返回字段描述	
返回字段	int，>=0返回人脸个数，<0时候为错误码
返回示例	

8.14.5 rgb+depth双目静默活体接口 方法名：`rgb_depth_liveness`

说明：`rgb+depth`双目静默活体

函数：`int rgb_depth_liveness(std::vector<TrackFaceInfo> &out, float &rgbscore, float &depthcore, const cv::Mat * rgb, const cv::Mat* depth)`

请求参数	说明	必须	类型	示例描述
out	返回的人脸框结构体数组	是	std::vector	人脸框结构体（多人则返回最大人脸）
rgbscore	返回的rgb动作活体分值	是	float	
depthscore	返回的depth动作活体分值	是	float	
rgb	传入的opencv视频帧	是	Mat	视频帧
depth	传入的opencv视频帧	是	Mat	视频帧

返回字段描述	
返回字段	int，>=0返回人脸个数，<0时候为错误码
返回示例	

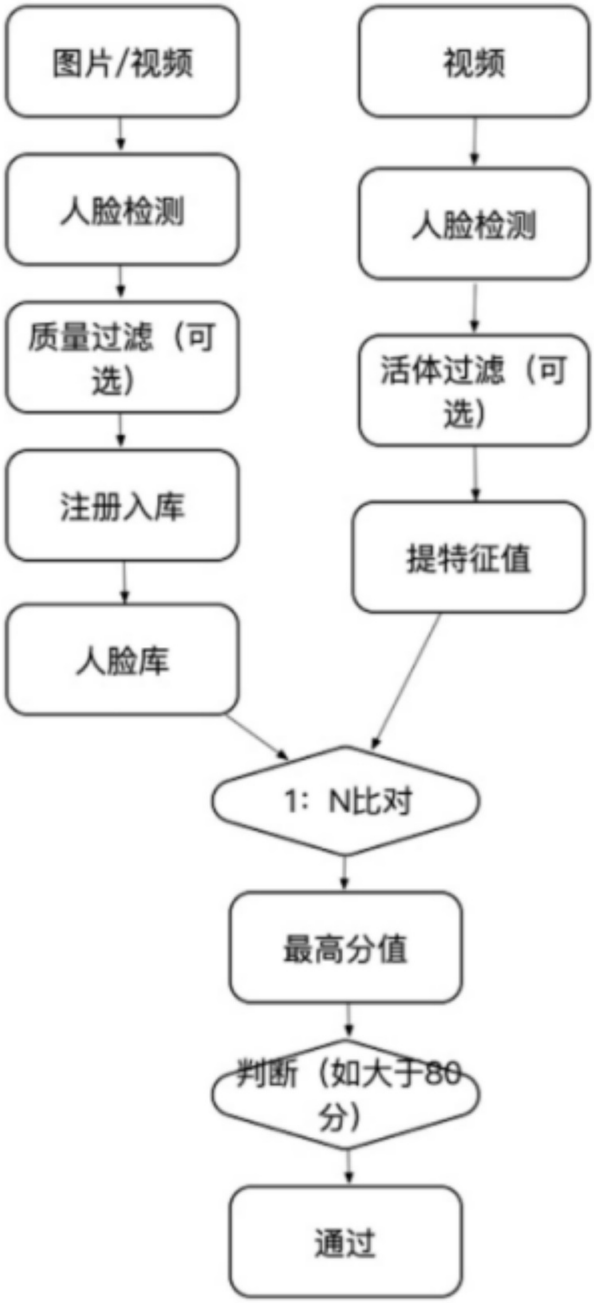
8.15 人脸库管理

sdk提供支持5万以下的人脸库管理，采用的是`sqlite`数据库，sdk启动后会自动生成`db`文件夹和`face.db`文件（人脸库数据文件）、`db`文件夹可手动删除，删除后人脸数据库即被整体删除，sdk重启后会自动重新创建新库。人脸库数据结构可采用`sqlLiteExpert`等可视化工具查看人脸库表结构。人脸库创建后有三张表,`feature`表（用来保存人脸特征值），`user`表（用来保存人脸用户信息，如`userid`，`groupid`以及人脸图片信息，用户信息等）以及`user_group`表（用户组表）。

人脸库可按组(`group_id`)划分,组就好比一个集团的子公司，人脸注册或查找既可以按整个库查找，也可以按组（子公司）查找，按组查找速度更快（范围小）。用户`id（user_id）`是用来标识人脸用户的唯一id，组id（`group_id`）是用来标识组（子公司）的唯一id。用户信息（`user_info`）可作为用户id的说明，如标识用户名称、住址等信息，也可不填写。人脸的比对或识

别、归根结底是人脸特征值的比对。人脸库的保存实际上是保存了对应用户user_id的特征值在人脸库上，同时保存了用户user_id和group_id及其对应关系（一个用户对应一张人脸、一个特征值数据）。除user_info字段（用户信息）支持中文外，user_id和group_id仅支持英文、数字和下划线的参数组合。

人脸库1：N识别可如如下图所示流程：



8.15.1 人脸注册接口(通过传入图片帧) 方法名：user_add

说明：用户注册，该接口支持传入opencv图片帧（通常指生活照，1寸或2寸的证件照也可以用这种类型）（图片帧注册会把人脸图片缩略图入库）

函数：void user_add(std::string& res, const cv::Mat *img, const char* user_id, const char* group_id, const char* user_info="")

请求参数	说明	必须	类型	示例描述
res	返回的结果	是	string json	数据
img	opencv图片帧指针	是	cv::Mat *	人脸图片opencv图片帧指针
user_id	用户id	是	const char*	用户id（由数字、字母、下划线组成），长度限制128B
group_id	组id	是	const char*	用户组id，标识一组用户（由数字、字母、下划线组成），长度限制128B
user_info	用户资料	否	const char*	256个字符以内（中文数减半）

返回字段描述(json)		
errno及msg映射	errno	Msg
	0	Success
	<0	失败的原因
返回示例		

8.15.2 人脸注册接口(通过传入特征值) 方法名：`user_add`

说明：用户注册，该接口通过传入提取的人脸图片特征值注册（特征值注册无法把人脸缩略图入库）

函数：`void user_add(std::string& res, Feature* f1, const char* user_id, const char* group_id, const char* user_info="")`

请求参数	说明	必须	类型	示例描述
res	返回的结果	是	string json	数据
f1	特征值结构体	是	Feature*	人脸特征值结构体
user_id	用户id	是	const char*	用户id（由数字、字母、下划线组成），长度限制128B
group_id	组id	是	const char*	用户组id，标识一组用户（由数字、字母、下划线组成），长度限制128B
user_info	用户资料	否	const char*	256个字符以内(中文数减半)

返回字段描述(json)		
errno及msg映射	errno	Msg
	0	Success
	<0	失败的原因
返回示例		

8.15.3 人脸更新接口(传入opencv图片帧) 方法名：`user_update`

说明：人脸更新

函数：`void user_update(std::string& res, const cv::Mat* img, const char* user_id, const char* group_id, const char* user_info="")`

请求参数	说明	必须	类型	示例描述
res	返回的结果	是	string json	数据
img	opencv图片帧指针	是	cv::Mat *	人脸图片opencv图片帧指针
user_id	用户id	是	const char*	用户id（由数字、字母、下划线组成），长度限制128B
group_id	组id	是	const char*	用户组id，标识一组用户（由数字、字母、下划线组成），长度限制128B
user_info	用户信息	否	const char*	用户资料，长度限制256个字符以内(中文数减半)

返回字段描述(json)		
errno及message映射	errno	Msg
	0	Success
	<0	失败的原因
返回示例		

8.15.4 用户删除接口 方法名：`user_delete`

说明：用户删除（用户删除后，人脸数据也随即被删除，一个用户对应一张人脸）

函数：`void user_delete(std::string&res, const char* user_id, const char* group_id)`

请求参数	说明	必须	类型	示例描述
res	返回的结果	是	string json	数据
user_id	用户id	是	const char*	用户id（由数字、字母、下划线组成），长度限制128B
group_id	组id	是	const char*	用户组id，标识一组用户（由数字、字母、下划线组成），长度限制128B

返回字段描述(json)			
errno及msg映射	errno	Msg	
	0	Success	
	<0	失败的原因	
data字段	log_id	string	请求日志标识
返回示例			

8.15.5 创建用户组接口 方法名：`group_add`

说明：创建用户组

函数：`void group_add(std::string&res, const char* gourp_id)`

请求参数	说明	必须	类型	示例描述
res	返回的结果	是	string	json 数据
group_id	用户组id	是	const char*	用户组id，标识一组用户（由数字、字母、下划线组成），长度限制128B

返回字段描述(json)			
errno及msg映射	errno	msg	
	0	Success	
	<0	失败的原因	
data字段	log_id	string	请求日志标识
返回示例			

8.15.6 用户组删除接口 方法名：group_delete

说明：用户组删除

函数：void group_delete(std::string&res, const char* group_id)

请求参数	说明	必须	类型	示例描述
res	返回的结果	是	string	json 数据
group_id	组id	是	const char*	用户组id，标识一组用户（由数字、字母、下划线组成），长度限制128B

返回字段描述(json)			
errno及msg映射	errno	msg	
	0	Success	
	<0	失败的原因	
data字段	log_id	string	请求日志标识
返回示例			

8.15.7 用户信息查询接口 方法名：get_user_info

说明：用户信息查询接口

函数：void get_user_info(std::string&res, const char* user_id, const char* group_id)

请求参数	说明	必须	类型	示例描述
res	返回的结果	是	string	json 数据
user_id	用户id	是	const char*	用户id（由数字、字母、下划线组成），长度限制128B
group_id	组id	是	const char*	用户组id，标识一组用户（由数字、字母、下划线组成），长度限制128B

返回字段描述(json)			
errno及msg映射	errno	msg	
	0	Success	
	<0	失败的原因	
data字段	log_id	string	请求日志标识
	result	array	识别结果列表
	group_id	string	组id
	face_token	string	人脸特征的唯一标识
	user_info	string	用户信息
	create_time	string	人脸首次注册时间
返回示例			

8.15.8 用户人脸图片查询接口 方法名：get_user_image

说明：用户人脸图片查询接口

函数：`int get_user_image(cv::Mat & img, const char* user_id, const char* group_id)`

请求参数	说明	必须	类型	示例描述
img	通过引用返回的图片结果	是	cv::Mat &	
user_id	用户id	是	const char*	用户id（由数字、字母、下划线组成），长度限制128B
group_id	组id	是	const char*	用户组id，标识一组用户（由数字、字母、下划线组成），长度限制128B
返回字段描述	返回int，0为成功，其他为错误码			

8.15.9 用户组列表查询接口 方法名：`get_user_list`

说明：用户组列表查询接口

函数：`void get_user_list(std::string&res, const char* group_id, int start = 0, int length = 100)`

请求参数	说明	必须	类型	示例描述
res	返回的结果	是	string	json 数据
group_id	用户组id	是	const char*	用户组id
start	查询起始序号	否	int	默认值为0
length	返回数量	否	int	默认值100，最大值1000

返回字段描述(json)			
errno及msg映射	errno	msg	
	0	Success	
	<0	失败的原因	
data字段	log_id	string	请求日志标识
	user_id_list	array	user_id列表数组
返回示例			

8.15.10 群组列表查询接口 方法名：`get_group_list`

说明：组列表查询

函数：`void get_group_list(std::string& res, int start = 0, int length = 100)`

请求参数	说明	必须	类型	示例描述
res	返回的结果	是	string	json 数据
start	起始序号	否	Int	默认值为0，从0开始
length	返回数量	否	Int	默认值为100，最大为1000

返回字段描述(json)			
errno及msg映射	errno	msg	
	0	Success	
	<0	失败的原因	
data字段	log_id	string	请求日志标识
	user_id_list	array	user_id列表数组
返回示例			

8.15.11 人脸库人脸数量查询 方法名：`db_face_count`

说明：查询数据库人脸数量 (传入组id表示查询该组都人脸数量，null表示查整个库的人脸数量)

函数：`int db_face_count(const char* group_id = nullptr)`

请求参数	说明	必须	类型	示例描述
group_id	组id	是	const char*	人脸分组的组id(传null表示查询整个库的数量)
返回示例	int >=0 (返回的数量)			

8.15.12 人脸识别接口(1:N) (传入opencv图片帧) 方法名：`identify`

说明：人脸识别（1：N）,可传入人脸组，比对某个组的所有人脸（人脸库可按单位分组，缩小组范围，减少识别比对时间）

函数：`void identify(std::string&res, const cv::Mat *img, const char* group_id_list, const char* user_id, int type = 0)`

参数	说明	必须	类型	示例描述
res	返回的结果	是	string	json 数据
img	opencv图片帧指针	是	cv::Mat *	人脸opencv图片帧指针
group_id_list	组列表	是	const char*	组列表
user_id	用户id	否	const char*	用户id
type	特征值类型	否	shi	特征值类型（0、是生活照、1、证件照2、近红外，参考特征值提取），未传默认为0

返回字段描述		
返回字段	比对分值	float
返回示例		

8.15.13 人脸识别接口(1:N) (传入特征值) 方法名：`identify`

说明：人脸识别（1：N）,可传入人脸组，比对某个组的所有人脸（人脸库可按单位分组，缩小组范围，减少识别比对时间）

函数：`void identify(std::string&res, Feature* f1, const char* group_id_list, const char* user_id, int type = 0)`

参数	说明	必须	类型	示例描述
res	返回的结果	是	string	json 数据
f1	起始序号	是	Feature*	特征值结构体
group_id_list	组列表	是	const char*	组列表
user_id	用户id	否	const char*	用户id
type	特征值类型	否	shi	特征值类型： 0、是生活照，包含1寸照 1、网格证件照 2、近红外，参考特征值提取） 未传默认为0

返回字段描述		
返回字段	比对分值	float
返回示例		

8.15.14 人脸识别接口(1:N) (传入opencv图片帧) 方法名：identify_with_all

说明：人脸识别（1：N）,和整个人类库比对

函数：void identify_with_all(std::string& res, const cv::Mat *img, int type = 0)

参数	说明	必须	类型	示例描述
res	返回的结果	是	string	json 数据
img	opencv图片帧	是	cv::Mat *	人脸opencv图片帧
type	特征值类型	否	int	特征值类型： 0、是生活照，包含1寸照 1、网格证件照 2、近红外，参考特征值提取） 未传默认为0

返回字段描述		
返回字段	比对分值	float
返回示例		

8.15.15 人脸识别接口(1:N) (传入特征值) 方法名：identify_with_all

说明：人脸识别（1：N）,和整个人类库比对

函数：void identify_with_all(std::string& res, Feature *f1, int type = 0)

参数	说明	必须	类型	示例描述
res	返回的结果	是	string	json 数据
f1	起始序号	是	Feature*	特征值结构体
type	特征值类型	否	int	特征值类型： 0、是生活照，包含1寸照 1、网格证件照 2、近红外，参考特征值提取） 未传默认为0

返回字段描述		
返回字段	比对分值	float
返回示例		

8.16 sdk系统信息接口

8.16.1 获取sdk版本号接口 方法名：`sdk_version`

说明：通过引用返回sdk版本号

函数：`void sdk_version(std::string& version)`

参数	说明	必须	类型	示例描述
version	返回的sdk版本号结果	是	string	如：6.3.3
返回字段描述				

8.16.2 获取设备指纹接口 方法名：`get_device_id`

说明：通过引用返回sdk的设备指纹信息（可用来标识sdk的唯一性）

函数：`void get_device_id(std::string& version)`

参数	说明	必须	类型	示例描述
device_id	返回的sdk设备指纹结果	是	string	如：0BAF96961A8377AB74797B05F7F2C805
返回字段描述				

9、功能接口对应结构体描述

9.1 人脸跟踪信息结构体

结构体名：`TrackFaceInfo`

说明：人脸跟踪信息结构体

```
  struct TrackFaceInfo
{
long face_id;
FaceBox box;
Landmarks facelandmarks;
}
```

参数	类型	说明
face_id	long	人脸id
box	FaceBox	人脸框结构体
facelandmarks	Landmarks	人脸关键点结构体

9.2 人脸框信息结构体

结构体名：`FaceBox`

说明：人脸框信息结构体

```
 struct FaceBox
{
int index;
float center_x;
float center_y;
float width;
float height;
float angle;
float score;
}
```

参数	类型	说明
index	int	人脸索引值
center_x	float	人脸中心点x坐标
center_y	float	人脸中心点y坐标
width	float	人脸宽度
height	float	人脸高度
angle	float	人脸角度
score	float	人脸置信度

9.3 人脸关键点信息结构体

结构体名：`Landmarks`

说明：人脸关键点信息结构体

```
 struct Landmarks
{
int index;
int size;
float data[144];
float score;
}
```

参数	类型	说明
index	int	人脸关键点索引
size	int	人脸关键点数量（size的值通常为144,72个x，y坐标）
data	float[144]	人脸关键点坐标数组（144）（72个x，y坐标）
score	float	人脸关键点置信度

9.4 人脸特征值结构体

结构体名：`Feature`

说明：人脸特征值结构体

```
 struct Feature
{
int size;
float data[128];
}
```

参数	类型	说明
size	int	特征值大小（通常为128个float浮点值数据）
data	float[]	128个浮点值数组

9.5 人脸姿态角结构体

结构体名：`HeadPose`

说明：人脸姿态角结构体

```
 struct HeadPose
{
float yaw;
float roll;
float pitch;
}
```

参数	类型	说明
yaw	float	左右偏转角
roll	float	与人脸平行平面内的头部旋转角
pitch	float	上下偏转角

9.6 人脸属性信息结构体

结构体名：`Attribute`

说明：人脸属性信息结构体

```
 struct Attribute
{
int age;
Race race;
AttributeEmotionType emotion;
Glasses glasses;
Gender gender;
}
```

参数	类型	说明
age	int	年龄

race	Race	种族
		enum Race
		{
		BDFace_RACE_YELLOW = 0, // 黄种人
		BDFace_RACE_WHITE = 1, //白种人
		BDFace_RACE_BLACK = 2, // 黑种人
		BDFace_RACE_INDIAN = 3, // 印第安人
		}

emotion	AttributeEmotionType	表情
		enum AttributeEmotionType
		{
		BDFACE_ATTRIBUTE_EMOTION_FROWN = 0,//皱眉
		BDFACE_ATTRIBUTE_EMOTION_SMILE = 1, //笑
		BDFACE_ATTRIBUTE_EMOTION_CALM = 2, //平静
		}

glasses	Glasses	戴眼镜状态
		enum Glasses
		{
		BDFACE_NO_GLASSES = 0 , // 无眼镜
		BDFACE_GLASSES = 1, // 有眼镜
		BDFACE_SUN_GLASSES = 2 //墨镜
		}

gender	Gender	性别
		enum Gender
		{
		BDFACE_GENDER_FEMILE = 0, // 女性
		BDFACE_GENDER_MALE = 1, //
		}

9.7 嘴巴闭合结构体

结构体名：`MouthClose`

说明：嘴巴闭合置信度结构体

```
  struct MouthClose
{
float score;
}
```

参数	类型	说明
score	float	置信度分值

9.8 口罩佩戴结构体、

结构体名：`MouthMask`

说明：口罩佩戴置信度结构体

```
 struct MouthMask
{
float score;
}
```

参数	类型	说明
score	float	置信度分值

9.9 最优人脸置信度结构体

结构体名：`Best`

说明：最优人脸置信度结构体

```
 struct Best
{
float score;
}
```

参数	类型	说明
score	float	置信度分值

9.10 人脸模糊度置信度结构体

结构体名：`Blur`

说明：人脸模糊度置信度结构体

```
 struct Blur
{
float score;
}
```

参数	类型	说明
score	float	置信度分值

9.11 人脸光照置信度结构体

结构体名：`Illumination`

说明：光照置信度结构体

```
 struct Illumination
{
int score;
}
```

参数	类型	说明
score	int	置信度分值

9.12 人脸遮挡置信度结构体

结构体名：`Illumination`

说明：人脸遮挡置信度结构体

```
 struct Illumination
{
float left_eye;
float right_eye;
float nose;
float mouth;
float left_cheek;
float right_cheek;
float chin;
}
```

参数	类型	说明
left_eye	float	左眼遮挡置信度
right_eye	float	右眼遮挡置信度
nose	float	鼻子遮挡置信度
mouth	float	嘴巴遮挡置信度
left_cheek	float	左脸遮挡置信度
right_cheek	float	右脸遮挡置信度
chin	float	下巴遮挡置信度

9.13 人眼闭合状态结构体

结构体名：`EyeClose`

说明：人眼闭合状态结构体

```
 struct EyeClose
{
float left_eye_close_conf;
float right_eye_close_conf;
}
```

参数	类型	说明
float	left_eye_close_conf	float
float	right_eye_close_conf	float

9.14 注意力结构体

结构体名：`GazeInfo`

说明：注意力结构体

```
  struct GazeInfo
{
Gaze left_eye;
Gaze right_eye;
}
struct Gaze
{
GazeDirection direction; // 凝视方向
float confidence; // 置信度
}
struct GazeDirection
{
BDFACE_GAZE_DIRECTION_UP = 0, // 向上看
BDFACE_GAZE_DIRECTION_DOWN = 1, // 向下看
BDFACE_GAZE_DIRECTION_RIGHT = 2, // 向右看
BDFACE_GZE_DIRECTION_LEFT = 3, // 向左看
BDFACE_GAZE_DIRECTION_FRONT = 4, // 向前看
BDFACE_GAZE_DIRECTION_EYE_CLOSE = 5, // 闭眼
}
```

参数	类型	说明
left_eye	Gaze	左眼注意力
right_eye	Gaze	右眼注意力

9.15 静默活体置信度结构体

结构体名：`LivenessScore`

说明：活体信度结构体

```
  struct LivenessScore
{
int score;
}
```

参数	类型	说明
score	float	置信度分值

10、多端特征值对齐

对于windows sdk来说，支持和安卓特征值对齐，windows sdk6.x系列，特征值对齐安卓6.x系列，但和如安卓7.x系列或5.x系列，则不对齐，且只支持rgb可见光特征值对齐。

在windows c++ sdk中，可通过提取的特征值128个`float`数组，保存成二进制数据，即和安卓中的512个byte保存成字节流的数据对齐。

sdk中提供了`float`数组转换为二进制`buffer`的示例代码以及二进制`buffer`数据转换为特征值`float`数组的示例代码。可参考`util`文件夹中的`FeatureAlign`类。

11、适配的深度双目摄像头特别说明

sdk自带了奥比中光以及华捷艾米的双目摄像头，支持`rgb+depth`，同时也支持迪威泰或视派尔的`rgb+nir`双目摄像头，奥比中光或华捷艾米的双目摄像头，需要安装驱动，在sdk目录的`tools->camera_driver`里面，分别双击安装即可(奥比的驱动对应`orbe`目录，华捷艾米的对应`hjimi`，注意双击安装后可能需要重启设备生效)，安装完毕后，可参考示例demo接入奥比或华捷的双目摄像头进行活体检测。因为sdk需要用动态库`dll`，所以需要根据您的x64或x86平台，分别拷贝奥比或华捷的dll里面的所有文件到sdk的`exe`目录所在文件夹里面，才能保证镜头模组能够正常运行(若未拷贝，运行sdk会提示找不到对应的如奥比或华捷的动态库`dll`文件)。若运行镜头模组的时候还提示找不到依赖，可选择奥比的镜头时候安装一下`vc_redist.exe(vs2013`的依赖库，可微

软官网查找下载)。华捷艾米的则可能需要安装vc_redist.exe(vs2010的依赖库，可微软官网查找下载)

12、错误码及错误信息

各接口返回结果error_code及msg信息如下：

错误码	错误内容	错误描述
0	SUCCESS	成功
-1	ILLEGAL_PARAMS	失败或非法参数
-2	MEMORY_ALLOCATION_FAILED	内存分配失败
-3	INSTANCE_IS_EMPTY	实例对象为空
-4	MODEL_IS_EMPTY	模型内容为空
-5	UNSUPPORT_ABILITY_TYPE	不支持的能力类型
-6	UNSUPPORT_INFER_TYPE	不支持的预测库类型
-7	NN_CREATE_FAILED	预测库对象创建失败
-8	NN_INIT_FAILED	预测库对象初始化失败
-9	IMAGE_IS_EMPTY	图像数据为空
-10	ABILITY_INIT_FAILED	人脸能力初始化失败
-11	ABILITY_UNLOAD	人脸能力未加载
-12	ABILITY_ALREADY_LOADED	人脸能力已加载
-13	NOT_AUTHORIZED	未授权
-14	ABILITY_RUN_EXCEPTION	人脸能力运行异常
-15	UNSUPPORT_IMAGE_TYPE	不支持的图像类型
-16	IMAGE_TRANSFORM_FAILED	图像转换失败
-1001	SYSTEM_ERROR	系统错误
-1002	PARARM_ERROR	参数错误
-1003	DB_OP_FAILED	数据库操作失败
-1004	NO_DATA	没有数据
-1005	RECORD_UNEXIST	记录不存在
-1006	RECORD_ALREADY_EXIST	记录已经存在
-1007	FILE_NOT_EXIST	文件不存在
-1008	GET_FEATURE_FAIL	提取特征值失败
-1009	FILE_TOO_BIG	文件太大
-1010	FACE_RESOURCE_NOT_EXIST	人脸资源文件不存在
-1011	FEATURE_LEN_ERROR	特征值长度错误
-1012	DETECT_NO_FACE	未检测到人脸
-1013	CAMERA_ERROR	摄像头错误或不存在
-1014	FACE_INSTANCE_ERROR	人脸引擎初始化错误
-1015	LICENSE_FILE_NOT_EXIST	授权文件不存在
-1016	LICENSE_KEY_EMPTY	授权序列号为空
-1017	LICENSE_KEY_INVALID	授权序列号无效
-1018	LICENSE_KEY_EXPIRE	授权序序列号过期

-1019	LICENSE_ALREADY_USED	授权序列号已被使用
-1020	DEVICE_ID_EMPTY	设备指纹为空
-1021	NETWORK_TIMEOUT	网络超时
-1022	NETWORK_ERROR	网络错误

13、常见问题：

1. sdk推荐使用vs2015 community或professional版本进行开发。若未安装该ide想运行sdk中的exe，可能需要安装vc运行环境，请参考sdk目录的vc_redist文件夹的exe双击安装，根据平台选择x64或x86安装。
2. 工程运行过程中，若不能正常运行功能，可在sdk的x64或win32目录下，把face_conf.json中字段false修改为true为打开日志模式，通过日志输出及各接口返回的错误码判断问题所在。
3. 模型文件可定制化：在main方法入口，可在sdk_init方法中传入模型文件夹的绝对路径，达到模型文件定制化的目的。若不定制化路径，sdk_init中传入null即可，默认模型文件在bin目录的models文件夹里面，无需更改。
4. 激活后是否可以把激活文件license.ini和license.key拷贝到其他设备运行？
   不能，离线sdk和设备绑定，每个设备对应一个key和一个license文件，换设备无法运行。但对同一台设备，可把Release下的license.ini和license.key拷贝到本电脑的另外sdk，该设备也等同于激活，可以使用。
5. 是否支持debug模式？
   只支持Release模式，不支持debug模式
6. sdk支持x64和x86两种模式，推荐使用x64模式。支持运行在win7或win10系统中。
7. 人脸库不支持中文参数？
   用户信息user_info支持中文，其他人脸管理参数目前暂只支持英文、数字下划线组合模式，工程所在的路径也建议不用放入中文路径中，可能影响人脸库创建。
8. 特征值多端是否对齐？
   在人脸6.0系列sdk中，win端，安卓和linux端生活照模式下的特征值都是对齐的。win中请把提取出来的128个float数据保存成二进制数据即可和安卓的512个byte二进制数据对齐。
9. 老版本的接口如4.1、4.2和新版本6.0为啥差别比较大？
   新版本对原子接口进行了封装，更易用，若想使用老版本的接口，可参考face_scene.cpp文件。
10. sdk的依赖库是否可以删减？
    新版支持sdk的依赖库进行精简，详细可参考文档说明。
11. sdk是否支持多线程运行？
    sdk不推荐采用多线程运行。
12. sdk是否支持二进制转opencv mat以及base64字符串的转换？
    sdk中的util文件夹里提供了大量转换示例代码，可参考查看。
13. 生活照和证件照模式有啥区别，是否可切换使用？
    sdk中，即使是1寸，2寸的证件照，比对或提取特征值也推荐用生活照模式处理，sdk中的证件照模式通常用作有网纹（比如身份证上带网格的），sdk中选择用了生活照模式，则所有接口都必须用该模式，

Linux-ARM-SDK

概述

- 百度人脸离线识别SDK Linux arm 版本，主要支持RK3288、3399芯片ubuntu平台，可登陆百度智能云控制台console后台获取SDK部署包
- 针对海思3516DV300、RV1109/1126、晶视CV1825等Linux芯片平台，有单独的专版SDK，请线下联系商务咨询获取专版

SDK

-海思3516DV300 官网地址：https://ai.baidu.com/tech/face/offline-sdk/hisilicon

-RV1109/RV1126 官网地址：https://ai.baidu.com/tech/face/offline-sdk/rv1109

版本日志

版本	日期	更新说明
v7.1	2021.12.28	添加人脸库、业务层封装等
v7.0	2021.08.30	模型能力升级到7.0，修改设备指纹的小部分bug等
v5.0	2021.04.30	1、升级了人脸检测模型，重点优化了复杂光线场景下的人脸检测能力； 2、升级了RGB、NIR和Depth三种模态的活体检测模型，优化后2D照片攻击防御能力和复杂光线场景真人活体检测能力提升明显； 3、升级了最新的人脸特征抽取与比对模型（人脸识别模型），升级后识别模型对老人、儿童等极端年龄群体的泛化行提升明显； 4、在RK3288主板上，端到端全流程耗时缩短到<300ms； 5、增加了最优图像帧功能，提升识别准确率； 6、修复跨平台第三方依赖缺失问题； 7、兼容性拓展：兼容三星4418、6818开发板，同时在支持ubuntu系统的RK3288验证开发； 注：由于该版本的SDK更换了特征抽取与比对的模型，因此从旧版本升级到该版本的用户需要重新注册人脸底库进行刷库
v4.1	2020.07.20	1、升级Paddle Lite版本，降低端到端全流程人脸识别耗时； 2、增加口罩检测功能，支持对用户是否佩戴口罩这个属性进行检测； 3、新增激活工具，该工具支持在连网状态下通过授权序列号完成激活
v4.0	2020.02.27	1、全面升级人脸检测、关键点检测模型，提升人脸检测和追踪的准确率，解决部分场景下的人脸漏检问题； 2、全面升级三模态活体检测模型，提升RGB、NIR、Depth三模态活体检测的准确率； 3、全面优化人脸特征抽取和特征比对模型，提升人脸识别模型的准确率和泛化性； 4、全面重构SDK的接口设计，降低二次开发难度的同时提升了二次开发的灵活性

目录

1 设计背景
2 名词解释
3 SDK简介
  3.1 功能架构
  3.2 版本及兼容性
  3.3 直接编译和运行
  3.4 交叉编译
4 SDK工程结构
5 授权激活
  5.1 SDK自动激活
  5.2 激活工具激活(LICENSE_TOOL)
  5.3 官网离线激活(LICENSE_TOOL)
6 SDK集成及DEMO示例工程
  6.1 SDK的集成
  6.2 SDK的使用示例
7 模型能力加载及模型说明
  7.1 模型删减说明
  7.2 模型路径的定制化
  7.3 能力定制化说明
    7.3.1 detect.json （人脸检测能力定制配置文件）
    7.3.2 track.json （人脸追踪能力定制配置文件）

7.3.3 action_live.json （动作活体能力定制配置文件）
7.3.4 crop.json (人脸抠图能力定制配置文件)
8 功能接口
8.1 人脸检测DETECT接口
8.2 人脸跟踪TRACK接口
8.3 清除人脸跟踪历史接口
8.4 人脸关键点接口
8.5 注意力检测接口
8.6 人脸属性检测接口
8.7 人脸扣图接口
8.8 暗光恢复接口
8.9 眼部状态检测接口
8.10 嘴巴闭合检测接口
8.11 口罩佩戴检测接口
8.12 人脸质量
8.12.1 人脸姿态角接口
8.12.2 人脸光照检测接口
8.12.3 人脸遮挡检测接口
8.12.4 人脸模糊度检测接口
8.12.5 最优人脸检测接口
8.13 特征值及人脸比对（1:1）
8.13.1 人脸特征值接口
8.13.2 人脸活体特征值接口
8.13.3 深度人脸特征值接口
8.13.4 特征值比对接口
8.13.5 人脸1：1比对接口
8.14 动作活体和静默活体
8.14.1 动作活体接口
8.14.2 清除动作活体历史接口
8.14.3 rgb静默活体接口
8.14.4 nir静默活体接口
8.14.5 rgb+depth双目静默活体接口
8.15 人脸库管理
8.15.1 人脸注册接口(通过传入图片帧)
8.15.2 人脸注册接口(通过传入特征值)
8.15.3 人脸更新接口
8.15.4 用户删除接口
8.15.5 创建用户组接口
8.15.6 用户组删除接口
8.15.7 用户信息查询接口
8.15.8 用户人脸图片查询接口
8.15.9 用户组列表查询接口
8.15.10 人脸库人脸数量查询
8.15.11 群组列表查询接口
8.15.12 人脸识别接口(1:N) (传入opencv图片帧)
8.15.13 人脸识别接口(1:N)(传入特征值)
8.15.14 人脸识别接口(1:N) (传入opencv图片帧)
8.15.15 人脸识别接口(1:N) (传入特征值)
8.16 SDK系统信息接口
8.16.1 获取sdk版本号接口
8.16.2 获取设备指纹接口
8.17 安全驾驶接口
8.17.1 驾驶行为检测接口
8.17.2 安全带佩戴检测接口
9 结构体描述
9.1 人脸跟踪信息结构体
9.2 人脸框信息结构体
9.3 人脸关键点信息结构体
9.4 人脸特征值结构体
9.5 人脸姿态角结构体
9.6 人脸属性信息结构体
9.7 嘴巴闭合结构体

9.8 口罩佩戴结构体
9.9 最优人脸置信度结构体
9.10 人脸模糊度置信度结构体
9.11 人脸光照置信度结构体
9.12 人脸遮挡置信度结构体
9.13 人眼闭合状态结构体
9.14 注意力结构体
9.15 静默活体置信度结构体
9.16 驾驶行为结构体
9.17 安全带佩戴结构体
10 多端特征值对齐
11 错误码及错误信息
12 常见问题

1. 设计背景

- 场景特点:

-- **网络**:对于无网、局域网等情况，无法连接公网，API 方式无法运作。如政府单 位、金融保险、教育机构等，其中内网情况最为常见，私有化部署是项目开展的前提条件。
-- **安全**:即使可以连接外网，因为人脸数据的敏感性，许多客户不希望将人脸数据传入百度服务器，如大学学生照片、部分企业员工数据等，API 形式也往往不被接受。
-- **速度**:由于各地网络线路、机房部署、图片采集方式等诸多原因，API 形式往往耗时较高，容易存在部分请求耗时过长的情况，容易影响业务正常运转。
-- **稳定**:API 形式容易受网络抖动、机房故障、线上连带 bug 等影响，存在一定的不稳定因素，可用性保障，往往成为在线调用最容易出现问题的地方。

- 客户特点:

-- **1:N-小型人脸库检索**:多为通道通行、固定区域人群验证等需求，如写字楼闸机 门禁、企业考勤打卡等，人脸库范围较小，且不易经常变动。
-- **1:1-自有数据源对比**:将当前采集的人脸，与其他数据源中的人脸进行对比，如 身份证芯片照、教务系统图片、档案图片等，进行快速的 1:1 对比验证。

- 核心需求:

-- **基础的人脸采集**:包含人脸检测、跟踪、捕获、质量校验等基础功能，获取符合识 别条件的人脸。为之前的客户端 SDK 的标准功能，离线版本 SDK 保留以上所有能力。
-- **本地特征抽取**:所有在 SDK 中运行的人脸图片，都可以完成本地特征抽取，以便进行对比或识别操作。
-- **1:1 对比**:支持两张图片的相似度对比，可直接传入图片，也可调用本地某个人脸特征;
-- **1:N 搜索**:支持一定库大小的人脸查找，在指定的人脸集合中查找最相似的人脸，并返回相似度分值;

2. 名词解释

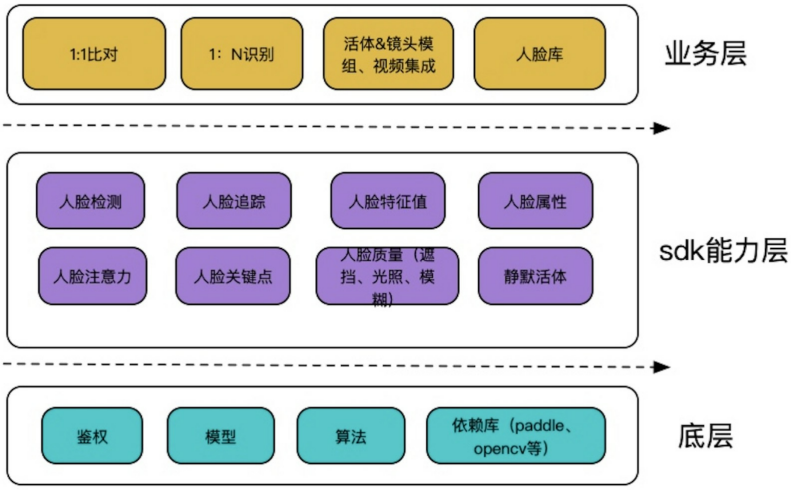
名词	定义
sdk	linux arm C++离线人脸识别 sdk
g++	linux c++编译工具，如 rk3288、3399 等系统一般自带
license	人脸识别激活所需要的激活文件,文档介绍了三种激活方式
key	人脸激活所需的序列号，可从百度 AI 官网申请(ai.baidu.com)
feature	人脸特征值，用来表示人脸特征的 128 个 float 浮点值
landmark	人脸关键点(72 个关键点)
face_token	对应人脸图片的唯一编码，若一个人上传了 2 张不同图片，则可能有 2 个不同 的 face_token,它和图片一一对应

3. sdk 简介

本 sdk 适应于 linux arm 平台下的人脸识别系统,为支持 c++语言开发的 sdk,开发 者可在 linux arm 平台下面进行开发(支持 armv7hf、armv8 平台)。sdk 采用导出动 态库 dll 的方式提供接口，sdk 附带一个示例工程 face_offline_sdk，提供了 sdk 的各种能力及调用示例。

3.1 功能架构

sdk 具有人脸检测、追踪、特征值、静默活体、人脸库、镜头模组集成等诸多功能。架构如下图所示:



3.2 版本及兼容性

本 sdk 支持 armv7hf、armv8 两种 liunx arm 平台。各平台推荐代表如: armv7hf 架构的如 rk3288 系列。
armv8 架构的如 rk3399 系列。
推荐使用 linux arm 平台上直接编译。

3.3 直接编译和运行

sdk 支持 ssh 网络登录开发板直接命令行编译。可参考 sdk 中的编译 txt 文件进行 编译运行。
在 sdk 工程目录:

```
 mkdir build            #新建 build 目录
cd build               #进入 build 目录
cmake-DARCH_ABI=armv8.. #根据平台分别用命令armv7hf或armv8
make -j4               #或 make 编译、根据开发板是否支持多核
cd ..                  #回退到 sdk 工程目录
./run.sh armv8         #不同平台请分别修改为如 armv7hf 或 armv8。
```

4. sdk 工程结构

```
face_offline_sdk          #sdk工程
├----CMakeLists.txt       #cmake编译文件
├---- images              #demo示例的图片文件夹
├---- include             #包含include文件夹
├---- lib                 #库文件夹
├---- license             #授权文件夹
├---- models              #模型文件夹
├---- run.sh              #执行脚本
├---- src                 #示例demo的源文件夹
├---- third_party         #引入的第三方文件夹
├---- doc                 #sdk使用文档文件夹
└---- conf                #能力定制化文件夹
```

5. 授权激活

sdk 需要授权激活后才能正常使用，在 sdk 初始化后，若报错误码-13(错误码参考文档最后定义)，一般为没有通过授权。通常，sdk 分按设备授权和按应用授权两种方式，大部分采用按设备授权的方式，按应用授权可针对批量大规模客户使用(文档中先只介绍按设备授权，按应用授权可工单或联系百度商务我们提供另外的文档或技术支持)。按设备授权的方式中，sdk 自动激活和激活工具激活需要设备能联网，若设备不能连外网，可采用官网离线激活的方式。

5.1 sdk 自动激活

在 sdk 的目录中有 license 文件夹，里面存放了 2 个文件，license.key 和 license.ini，分别是授权 key 和授权文件，若在百度官网申请了授权系列号 key(16 位)，可按 sdk 中的 license 文件夹中 license.key 原格式样子覆盖填写您申请的 key。在设备能联网的情况下，运行 sdk 会自动授权激活并拉取新的授权文件 license.ini 覆盖 sdk 中的旧文件。

5.2 官网离线激活

若设备不能连外网、通过授权还有另外一种方法，即通过运行SDK中的get_device_id来获取硬件指纹信息，获取到设备硬件指纹信息，通过百度官网填入指纹信息和申请到的系列号 key，可完成激活并下载获取到 license.ini 文件和 license.key 文件，把这 2 个文件拷贝到 face_offline_sdk 的 license 目录下，重新运行 sdk 亦可通过授权激活。

6. SDK 集成及 demo 示例工程

6.1 SDK 的集成

sdk 集成的 include 需包含 sdk 的接口头文件:baidu_face_api.h、以及 struct_info.h(定义 sdk 的结构体类)两个文件。(若需要使用 face_scene 文件夹里 面的原子方法，则需要引入整个 sdk 文件夹的 include 目录)。动态库文件在 lib 文件 夹下，根据平台区分 armv7hf 或 armv8 文件夹里面，动态库需要集成时候全部引入。 各动态库文件说明如下

so 文件名称	**so 文件说明**	**是否可删除**
libbaidu_face_api.so	百度人脸库文件	否
libface_sdk.lib	百度人脸库文件	否
libopencv_world.so	opencv 库文件，用来 显示图片，视频等	否、与之一起的有 so.4.1、 so.4.1.0
libcurl.so	联网库文件	否、与之一起的有 so.4、so.4.1
libssl.so	openssl 库 文 件 (https 联网)	否、与之一起的有 so.1.1
libcrypto.so	openssl 库 文 件 (https 联网)	否、与之一起的有 so.1.1

6.2 sdk 的使用示例

sdk 工程 face_offline_sdk 包含了 sdk 接口及 demo 示例工程。其中:sdk 接口头 文件为前述 include 目录的 baidu_face_api.h，提供了 sdk 的各接口方法。sdk 接口 的 lib 库文件为前述 lib 目录。根据平台分别放置在 armv7hf/armv8 目录。其 中:baidu_face_api.so 和 face_sdk.so 为人脸 sdk 的 lib 库文件。其他为 curl、 opencv 库文件等。
demo 示例工程 face_offline_sdk 展示了如何集成百度人脸识别离线 sdk，并调用 sdk 的方法及使用场景化示例等。
在 face_offline_sdk 中的 face.cpp 的 main()方法中，有使用 sdk 的各个接口方法 示例。接入 sdk 及其简单，如下图及解释:

```
 // 入口函数
int main()
{
//api 实例指针
BaiduFaceApi *api = new BaiduFaceApi();
//初始化 sdk
std::cout << "before sdk_init" << std::endl; int res = api->sdk_init(nullptr);
std::cout << "after sdk_init" << std::endl; if (res != 0)
{
std::cout << "sdk init result is:" << res << std::endl; getchar();
return 0;
}
std::time_t time_begin = get_timestamp();
FaceDemo *demo = new FaceDemo(api);
demo->face_demo();
delete demo;
std::time_t time_end = get_timestamp();
std::cout << "time cost is :" << time_end - time_begin << "ms" << std::en dl;
std::cout << "before delete api" << std::endl; // 释放 sdk 实例指针，防止内存泄漏
delete api;
getchar();
std::cout << "end main" << std::endl;
return 0;
}
```

sdk 使用主要三步:

1)初始化实例指针

2)初始化 sdk

第 3 步即可调用示例 demo，实现需要的功能。末尾需要释放 sdk 实例化指针 api。 另外，可通过 is_auth()方法查看是否通过了授权，通常，需要通过授权，才可以使用 sdk 的各种能力。

示例工程中:分别有以下几个文件夹放置文件对应几个常用 sdk 的调用 demo。

解析如下:

文件夹或文件名	文件说明
face.cpp	sdk 总入口，初始化，模型加载，销毁等,见 main 方法
face_demo.cpp	demo 总入口，可打开注释运行各类 demo 示例
face_detect	人脸检测接口及示例 demo
face_track	人脸跟踪接口及示例 demo
face_feature	人脸特征值接口及示例 demo
face_compare	人脸 1:1&1:N(1:N 需要先人脸入库) 比对，特征值比较等接口及示例 demo
face_liveness	可见光 RGB、红外 IR 活体检测、RGB&IR 双目摄像头静默活 体检测，RGB&DEPTH 双目摄像头静默活体检测等
face_manager	人脸库管理类，包括人脸注册、删除，更新、组管理，人脸信息查询等
face_attr	人脸属性(年龄、性别、种族等) 接口及示例
face_head_pose	人脸姿态角接口及示例
face_illumination	人脸光照检测接口及示例
face_occlusion	人脸遮挡度检测接口及示例
face_blur	人脸模糊度检测接口及示例
face_crop	人脸扣图接口及示例
face_gaze	双眼注意力检测接口及示例
face_eye_close	眼睛闭合检测接口及示例
face_mouth_close	嘴巴闭合检测接口及示例
face_mouth_mask	是否佩戴口罩检测接口及示例
face_scene	原子接口综合示例(该示例亦展示了如何使用 sdk 原子接口等)
sdk_info	实现如读取 sdk 版本号、设备指纹等接口示例
driver_monitor	驾驶行为检测示例，如打电话、吃东西等
safety_belt	安全带佩戴检测示例
face_action_live	动作活体检测示例

7. 模型能力加载及模型说明

7.1 模型删减说明

sdk 支持按需配置模型和加载能力，若 sdk 有某部分功能不需要使用，可尝试删除 一些模型，删除后模型即不会加载也不会占用内存。sdk 中 models 文件夹里的模型及 是否可删减说明如下表:

模型文件夹名称	说明	是否可删减	删减说明
detect	人脸检测模型	否	该文件夹不能删减
align	人脸关键点模型	否	该文件夹不能删减
attribute	人脸属性	是	若不使用该功能，该文件夹可删除
best_image	最佳人脸	是	若不使用该功能，该文件夹可删除
blur	人脸质量模糊度检测	是	若不使用该功能，该文件夹可删除
dark_enhance	暗光恢复	是	若不使用该功能，该文件夹可删除
eye_close	眼睛闭合	是	若不使用眼睛闭合及动作活体功能，该 文件夹可删除
feature	人脸特征值	是	若不用nir近红外的人脸特征值，可删 除 feature_nir 开头的文件
gaze	人脸注意力检测	是	若不使用该功能，该文件夹可删除
mouth_close	嘴巴闭合检测	是	若不使用嘴巴闭合及动作活体功能，该 文件夹可删除
mouth_mask	口罩佩戴检测	是	若不使用该功能，该文件夹可删除
occlusion	人脸遮挡检测	是	若不使用该功能，该文件夹可删除
silent_live	静默活体检测	是	若只使用rgb可见光单目静默活体，则 除 liveness_rgb 开头的文件外其他皆可删除
driver_monitor	安全驾驶	是	若不使用驾驶行为检测和安全带佩戴 检测，该文件夹可删除

7.2 模型路径的定制化

sdk 支持模型文件夹 models 的路径自定义，当 sdk 初始化 api->sdk_init(nullptr)传 null 时候，为 sdk 支持模型文件夹路径在默认路径，即 models 在 sdk 现有位置。同时也支持 models 通过 sdk_init 中传入绝对路径。若把 models 文件夹拷贝到如/home/face 文件夹下面，则可定义:

api->sdk_init(“/home/face”);

此时，授权文件夹 license 也需要随之变为/home/face 文件夹下面。否则会出现授权 不通过的问题。

另外，若使用了能力自定义的 config 文件夹，也需要拷贝到/home/face 文件夹下面。 否则能力定制化也不会生效而是使用的系统默认。

7.3 能力定制化说明

sdk 支持能力自定义、通过读取配置文件的方式进行能力定制化。sdk 默认能力加 载无需定制化，若需要定制化，请把 sdk 根目录里面的 conf 文件夹重命名为 config 文件夹。并且在 sdk 中，把里面的 json 配置按如下说明做修改，可起到定制化能力加载的效果，配置文件简要说明如下(若需定制化修改，请参考示例 json，修改 json 字段的值来达到定制化的目的)

7.3.1 detect.json (人脸检测能力定制配置文件)

配置文件名	detect.json

说明	人脸检测自定义能力配置文件
	{
	"max_detect_num":5, "min_face_size":0, "scale_ratio":-1, "not_face_thr":0.5
	}

参数	类型	说明
max_detect_num	int	最大检测的人脸数量，最多支持 50，最小1,默认 5
min_face_size	int	默认 0，可用来设置检测的最小人脸，比如可设为 10，则 表示小于 10*10 的人脸，sdk 检测不到
scale_radio	int	默认-1，表示进行人脸检测时候的图片缩放比率。建议用-1 表示传入原图 sdk 自己缩放(为保证检测效果，该参数建议用默认)
not_face_thr	float	默认 0.5，表示非人脸的阈值，取值范围 0-1

7.3.2 track.json (人脸追踪能力定制配置文件)

配置文件名	track.json

说明	人脸追踪自定义能力配置文件
	{
	"detect_intv_before_track":0.02, "detect_intv_during_track":0.02
	}

参数	类型	说明
detect_intv_before_track	float	表示人脸追踪前进行人脸检测的时间间隔(单位毫秒)
detect_intv_duringtrack	float	表示人脸追踪时候进行人脸检测的时间间隔(单位毫秒)

7.3.3 action_live.json (动作活体能力定制配置文件)

配置文件名	action_live.json

说明	人脸动作活体自定义能力配置文件
	{
	"eye_open_threshold":0.5,
	"eye_close_threshold":0.5,
	"mouth_open_threshold":0.5,
	"mouth_close_threshold":0.5,
	"look_up_threshold":0.5,
	"look_down_threshold":0.5,
	"turn_left_threshold":0.5,
	"turn_right_threshold":0.5,
	"nod_threshold":0.5,
	"shake_threshold":0.5,
	"max_cache_num":1
	}

参数	类型	说明
eye_open_threshold	float	表示人脸动作活体眼睛睁开的置信度阈值
eye_close_threshold	float	表示人脸动作活体眼睛闭合的置信度阈值
mouth_open_threshold	float	表示人脸动作活体嘴巴张开的置信度阈值
mouth_close_threshold	float	表示人脸动作活体嘴巴闭合的置信度阈值
look_up_threshold	float	表示人脸动作活体抬头的置信度阈值
look_down_threshold	float	表示人脸动作活体低头的置信度阈值
turn_left_threshold	float	表示人脸动作活体向左转头的置信度阈值
turn_right_threshold	float	表示人脸动作活体向右转头的置信度阈值
nod_threshold	float	表示人脸动作活体点头的置信度阈值
shake_threshold	float	表示人脸动作活体摇头的置信度阈值
max_cache_num	float	最大缓存数、推荐为 1 不做修改

7.3.4 crop.json (人脸抠图能力定制配置文件)

配置文件名	crop.json

说明	人脸抠图自定义能力配置文件
	{
	"is_flat":0,
	"crop_size":200,
	"enlarge_ratio":1
	}

参数	类型	说明
is_flat	int	默认为 0，表示是否是镜像，该参数目前无效
crop_size	int	表示抠图的大小，如 200，则表示抠出来的是 200*200 的图 片
enlarge_ratio	float	默认是 1，若小于 1，比如 0.8，图片有点放大，会稍微模 糊，建议自定义可设为 1

8. 功能接口

sdk 功能接口的调用可参考各示例 cpp 文件，接口定义在 baidu_face_api.h。 sdk 实现的主要功能有人脸实时跟踪检测、人脸特征值提取、动作活体、RGB&IR 静默 活体检测、RGB&DEPTH 静默活体检测、人脸注册、人脸更新、组管理、用户管理以及 1:1 人脸对比,1:N 人脸识别、特征值的比对和通过 usb 或笔记本自带摄像头检测视频 帧，返回识别出的人脸信息、人脸属性等，另外支持对人脸检测进行能力加载设置， 达到根据设置进行识别的目的。
各接口功能及传入参数和返回结果等定义如下:

8.1 人脸检测 detect 接口

方法名	detect
说明	人脸检测，返回人脸信息
函数	int detect(std::vector<FaceBox> &out, const cv::Mat* mat, int type = 0)

请求参数	说明	必须	类型	示例描述
out	通过引用返回的人脸结构体数组	是	std::vector<FaceBox>	人脸框信息结构体
mat	传入的 opencv 视频帧	是	Opencv mat	请参考示例
type	检测类型(传 0 表示 rgb 可见光人脸检测， 1 表示 nir)		int	0 或 1，不传默认为0
返回信息	函数的返回	是	int	<=0 为未检测到人脸或错误码; >0 时候为检测到的人脸数量

8.2 人脸跟踪 track 接口

方法名	track
说明	人脸跟踪，返回人脸信息
函数	int track(std::vector<TrackFaceInfo>&out, const cv::Mat* mat, int type = 0)

请求参数	说明	必须	类型	示例描述
out	通过引用返回的人脸结构体数组	是	std::vector<TrackFaceInfo>	人脸跟踪结构体信息
mat	传入的 opencv 视频帧	是	Opencv mat	请参考示例
type	检测类型(传 0 表示 rgb 可见光人脸检测，1 表示 nir)	否	int	0 或 1，不传默认为0
返回信息	函数的返回	是	int	<=0 为未检测到人脸或错误码； >0 时候为检测到的人脸数量

8.3 清除人脸跟踪历史接口

方法名	clear_track_history
说明	清除人脸跟踪的历史信息
函数	int clear_track_history(int type = 0)

请求参数	说明	必须	类型	示例描述
type	检测类型(传 0 表示 rgb 可见光人脸检测，1 表示 nir)	否	int	0 或 1，不传默认为 0
返回信息	函数的返回	是	void	

8.4 人脸关键点接口

方法名	face_landmark
说明	人脸关键点，返回人脸关键点信息
函数	int face_landmark(std::vector<Landmarks>&out, const cv::Mat* mat, int type)

请求参数	说明	必须	类型	示例描述
out	通过引用返回的人脸关键点数组	是	std::vector<Landmarks>	人脸关键点信息结构体
mat	传入的 opencv 视频帧	是	Opencv mat	请参考示例
type	检测类型（传 0 可见光生活照，1、表示近红外）	是	int	
返回信息	函数的返回	是	int	<=0 为未检测到人脸或错误码； >0时候为检测到的人脸数量

8.5 注意力检测接口

方法名	face_gaze
说明	人脸注意力检测，返回人脸注意力信息
函数	int face_gaze(std::vector<GazeInfo>& out, const cv::Mat* mat)

请求参数	说明	必须	类型	
out	通过引用返回的人脸注意力结构体数组	是	std::vector<GazeInfo>	注意力结构体
mat	传入的 opencv 视频帧	是	Opencv mat	请参考示例
返回信息	函数的返回	是	int	<=0 为未检测到人脸或错误码; >0时候为检测到的人脸数量

8.6 人脸属性检测接口

方法名	face_attr
说明	人脸属性检测，返回人脸属性信息
函数	int face_attr(std::vector<Attribute>& out, const cv::Mat *mat)

请求参数	说明	必须	类型	示例描述
out	通过引用返回的人脸属性结构体数组	是	std::vector<Attribute>	人脸属性结构体
mat	传入的 opencv 视频帧	是	Opencv mat	请参考示例
返回信息	函数的返回	是	int	<=0 为未检测到人脸或错误码 ； >0时候为检测到的人脸数量

8.7 人脸扣图接口

方法名	face_crop
说明	人脸扣图，返回人脸扣图（仅支持单人脸抠图）
函数	int face_crop(cv::Mat&out, const cv::Mat* mat)

请求参数	说明	必须	类型	示例描述
out	抠图结果图片帧	是	Opencv mat	
mat	传入的 opencv 视频帧	是	Opencv mat	
返回信息	函数的返回	是	int	<=0 为未检测到人脸或错误码 ；>0时候为检测到的人脸数量

8.8 暗光恢复接口

方法名	dark_enhance
说明	暗光恢复，用于图片比较暗的使之变量利于人类检测(仅支持单人脸抠 图)
函数	int dark_enhance(cv::Mat&out, const cv::Mat* mat)

请求参数	说明	必须	类型	示例描述
out	暗光恢复结果图片帧	是	Opencv mat	
mat	传入的 opencv 视频帧	是	Opencv mat	
返回信息	函数的返回	是	int	返回：<0 为错误码; =0表示成功

8.9 眼部状态检测接口

方法名	face_eye_close
说明	人脸眼部状态检测，返回人脸眼部状态信息
函数	int face_eye_close(std::vector<EyeClose> &out, const cv::Mat* mat)

请求参数	说明	必须	类型	示例描述
out	通过引用返回的人脸眼部状态结构体数组	是	std::vector<EyeClose>	眼部状态信息结构体
mat	传入的 opencv 视频帧	是	Opencv mat	请参考示例
返回信息	函数的返回	是	int	<=0 为未检测到人脸或错误码；>0时候为检测到的人脸数量

8.10 嘴巴闭合检测接口

方法名	face_mouth_close
说明	人脸嘴巴闭合检测，返回人脸嘴巴闭合信息
函数	int face_mouth_close(std::vector<MouthClose>& out, const cv::Mat* mat)

请求参数	说明	必须	类型	示例描述
out	通过引用返回的人脸嘴巴闭合结构体数组	是	std::vector<M outhClose>	嘴巴闭合结构体
mat	传入的 opencv 视频帧	是	Opencv mat	请参考示例
返回信息	函数的返回	是	int	<=0 为未检测到人脸或错误码；>0 时候为检测到的人脸数量

8.11 口罩佩戴检测接口

方法名	face_mouth_mask
说明	人脸口罩佩戴检测，返回人脸口罩佩戴信息
函数	int face_mouth_mask(std::vector<MouthMask>& out, const cv::Mat* mat)

请求参数	说明	必须	类型	示例描述
out	通过引用返回的人脸口罩佩戴结构体数组	是	std::vector<Mo uthMask>	佩戴口罩结构体
mat	传入的 opencv 视频帧	是	Opencv mat	请参考示例
返回信息	函数的返回	是	int	<=0 为未检测到人脸或错误码；>0时候为检测到的人脸数量

8.12 人脸质量

人脸质量判断可由以下几个因素自由组合综合判断，如姿态角、光照、遮挡、模 糊以及最佳人脸。人脸质量判断的推荐阈值: 人脸检测置信度>0.63、遮挡度 <0.75、 人脸姿态角 <20 、人脸模糊度 <0.8、最优人脸 >50。 **8.12.1 人脸姿态角接口** 方法名 |face_head_pose | --- | --- | 说明 |人脸姿态角检测，返回人脸姿态角信息 函数 |int face_head_pose(std::vector<HeadPose>&out, const cv::Mat* mat)

请求参数	说明	必须	类型	示例描述
out	通过引用返回的人脸姿态角数组	是	std::vector<HeadPose>	人脸姿态角信息结构体
mat	传入的 opencv 视频帧	是	Opencv mat	请参考示例
返回信息	函数的返回	是	int	<=0 为未检测到人脸或错误码；>0 时候为检测到的人脸数量

8.12.2 人脸光照检测接口 方法名 |face_illumination | --- | --- |
说明 | 人脸光照检测、（光照分值 0-255、分值越大光照越强） 函数 |int face_illumination(std::vector<Illumination>&out, const cv::Mat* mat)

请求参数	说明	必须	类型	示例描述
out	通过引用返回的人脸光照信息数组	是	std::vector<Illum ination>	光照置信度结构体
mat	传入的 opencv 视频帧	是	Opencv mat	请参考示例
返回信息	函数的返回	是	int	<=0 为未检测到人脸或错误码;>0时候为检测到的人脸数量

8.12.3 人脸遮挡检测接口 方法名 | face_occlusion | --- | ---
说明 | 人脸遮挡检测（遮挡分值 0-1，分值越大遮挡度越高） 函数 | int face_occlusion(std::vector<Occlusion>& out, const cv::Mat* mat)

请求参数	说明	必须	类型	示例描述
out	通过引用返回的人脸遮挡结构体数组	是	std::vector<Occlusion>	遮挡置信度结构体
mat	传入的 opencv 视频帧	是	Opencv mat	请参考示例
返回信息	函数的返回	是	int	<=0 为未检测到人脸或错误码；>0 时候为检测到的人脸数量

8.12.4 人脸模糊度检测接口 方法名 |face_blur | --- | ---

说明 | 人脸模糊检测(模糊度分值 0-1，分值越大模糊度越高) 函数 | int face_blur(std::vector<Blur> &out, const cv::Mat * mat)

请求参数	说明	必须	类型	示例描述
out	通过引用返回的人脸模糊结构体数组	是	std::vector<Blur>	模糊度结构体
mat	传入的 opencv 视频帧	是	Opencv mat	请参考示例
返回信息	函数的返回	是	int	<=0 为未检测到人脸或错误码 ；>0时候为检测到的人脸数量

8.12.5 最优人脸检测接口 方法名 | face_best | --- | ---

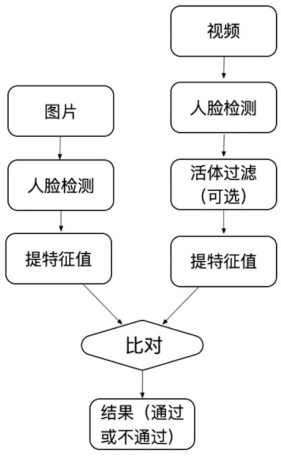
说明 | 最佳人脸检测(最优人脸分值 0-100，分值越高，最佳人脸图片得分越高) 函数 | int face_best(std::vector<Best> &out, const cv::Mat *mat)

请求参数	说明	必须	类型	示例描述
out	通过引用返回的最佳人脸结构体数组	是	std::vector<Best>	最优人脸结构体
mat	传入的 opencv 视频帧	是	Opencv mat	请参考示例
返回信息	函数的返回	是	int	<=0 为未检测到人脸或错误码 ；>0时候为检测到的人脸数量

8.13 特征值及人脸比对(1:1)

人脸比对的原理实际是特征值比对，通过提取图片中的人脸特征值，根据特征值 调用 compare_feature 接口进行比对，推荐比对分值超过 80 分为同一人，可根据实际 检测比对情况动态调整。

人脸 1:1 比对流程可如下图，实现如人证比对功能。(人的照片和实时视频比对， 可根据使用情况选择是否启用质量检测)



8.13.1 人脸特征值接口 方法名 | face_feature | --- | --- 说明 | 人脸特征值提取，并返回人脸信息 函数 | int face_feature(std::vector<Feature> &out_fea, std::vector<FaceBox> &out_box, const cv::Mat *mat，int type = 0)

请求参数	说明	必须	类型	示例描述
out_fea	通过引用返回的人脸特征值结构体数组	是	std::vector<Feature>	人脸特征值结构体信息
out_box	通过引用返回的人脸框结构体数组	是	std::vector<FaceBox>	人脸框结构体信息
mat	传入的 opencv 视频帧	是	Opencv mat	请参考示例
type	检测类型（传 0 可见光生活照特征值 1、近红外特征值）	否	int	0或1,未传默认为0
返回信息	函数的返回	是	int	<=0 为未检测到人脸或错误码； >0 时候为检测到的人脸数量

8.13.2 人脸活体特征值接口 方法名 |liveness_feature | --- | --- 说明 | 人脸特征值提取，同时有活体校验、并返回活体分值，人脸信息 函数 | int liveness_feature(std::vector<Feature> &out_fea, std::vector<FaceBox> &out_box, float& score, const cv::Mat *mat , int type = 0)

请求参数	说明	必须	类型	示例描述
out_fea	通过引用返回的人脸特征值结构体数组	是	std::vector<Feature>	人脸特征值结构体信息
out_box	通过引用返回的人脸框结构体数组	是	std::vector<FaceBox>	人脸框结构体信息
mat	传入的 opencv 视频帧	是	Opencv mat	请参考示例
score	返回的活体分值	是	float	
type	检测类型（传 0 可见光特征值，1 表示近红外特征值）	否	int	0或1,未传默认为0
返回信息	函数的返回	是	int	<=0 为未检测到人脸或错误码; >0时候为检测到的人脸数量

8.13.3 深度人脸特征值接口 方法名 |rgbd_feature | --- | --- | 说明 |人脸特征值提取，返回人脸信息 函数 |int rgbd_feature(std::vector<Feature> &out_fea, std::vector<Feature> &out_box, const cv::Mat *rgb_mat, char* depth_mat)

请求参数 |说明| 必须 |类型 |示例描述 | --- | --- | --- | --- | --- out_fea|通过引用返回的人脸特征值结构体数组|是 | std::vector<Feature> | 人脸特征值结构体信息 out_box |通过引用返回的人脸框结构体数组 |是 |std::vector<FaceBox>|人脸框结构体信息 rgb_mat |传入可见光的 opencv 视频帧 |是 | Opencv mat |请参考示例 depth_mat | 传入深度的二进制流数据 |是 |char* |请参考示例 返回信息 |函数的返回|是 |int |<=0 为未检测到人脸或错误码；
>0时候为检测到的人脸数量

8.13.4 特征值比对接口 方法名 |compare_feature | --- |--- 说明 |人脸跟踪，返回人脸信息 函数| float compare_feature(Feature *f1, Feature* f2, int type)

请求参数	说明	必须	类型	示例描述
f1	人脸特征值结构体	是	Feature*	人脸特征值结构体信息(仅支持单个结构体的比对)
f2	人脸特征值结构体	是	Feature*	人脸特征值结构体信息(仅支持单个结构体的比对)
type	特征值类型（0,1）	否	int	0：生活照 1：近红外特征值，未传默认为0
返回信息	函数的返回	是	float	特征值比对分值

8.13.5 人脸1：1比对接口 方法名| match | --- | --- 说明 |人脸比对，返回人脸比对分值 函数 |int match(const cv::Mat& img1,

const cv::Mat& img2, int type)

请求参数	说明	必须	类型	示例描述
img1	传入可见光的opencv图片帧	是	cv::Mat	
img2	传入可见光的opencv图片帧	是	cv::Mat	
type	比对类型（0,1）	否	int	0：生活照模式、包含1寸照 1：近红外特征值 未传默认为0(1寸或2寸的证据照也推荐当作生活照模式)
返回信息	函数的返回	是	int	人脸比对分值（同特征值比对分值结果应该是float，这里返回int是做了四舍五入）

8.14 动作活体和静默活体

8.14.1 动作活体接口 方法名 |face_action_live | --- | --- 说明 |动作活体（眨眨眼、张张嘴等动作校验活体） 函数 |int face_action_live(std::vector& out, int action_type, int& action_result, const cv::Mat* mat)

参数	说明	必须	类型	示例描述
out	返回的人脸框结构体数组	是	std::vector	人脸框结构体
action_type	传入的活体动作	是	int	动作活体类型，如0表示眨眨眼，1表示张嘴等
action_result	动作活体的返回结果	是	int	返回结果为1表示存在该动作活体
mat	传入的opencv视频帧	是	Mat	视频帧

返回字段描述	
返回字段	int，>=0返回人脸个数，<0时候为错误码
返回示例	

8.14.2 清除动作活体历史接口 方法名 | action_live_clear_history | --- | --- 说明 | 清除动作活体历史 函数| int action_live_clear_history()

返回字段描述	
返回字段	返回int，0:表示成功，其他为错误码，参考文档后续
返回示例	

8.14.3 rgb静默活体接口 方法名 |rgb_liveness | --- | ---| 说明| rgb可见光单目静默活体（推荐rgb+nir或rgb+depth进行双目活体校验） 函数| int rgb_liveness(std::vector& out, float &score, const cv::Mat* mat)

参数	说明	必须	类型	示例描述
out	返回的人脸框结构体数组	是	std::vector	人脸框结构体（多人则返回最大人脸）
score	返回的动作活体分值	是	float	
mat	传入的opencv视频帧	是	Mat	视频帧

返回字段描述	
返回字段	int，>=0返回人脸个数，<0时候为错误码
返回示例	

8.14.4 nir静默活体接口 方法名 |nir_liveness | --- | ---| 说明| Nir近红外单目静默活体（推荐rgb+nir或rgb+depth进行双目活体校

验） 函数 |int nir_liveness(std::vector& out, float &score, const cv::Mat* mat)

参数	说明	必须	类型	示例描述
out	返回的人脸框结构体数组	是	std::vector	人脸框结构体（多人则返回最大人脸）
score	返回的动作活体分值	是	float	
mat	传入的opencv视频帧	是	Mat	视频帧

返回字段描述	
返回字段	int，>=0返回人脸个数，<0时候为错误码
返回示例	

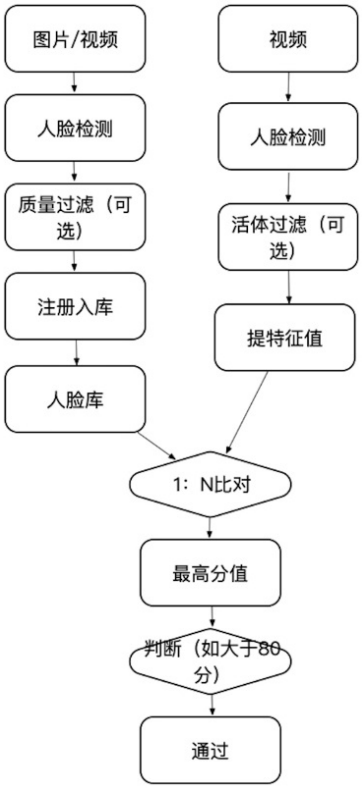
8.14.5 rgb+depth双目静默活体接口 方法名| rgb_depth_liveness | --- | --- 说明 |rgb+depth双目静默活体 函数 |int rgb_depth_liveness(std::vector &out, float &rgbscore, float &depthcore, const cv::Mat *rgb, char* depth)

参数	说明	必须	类型	示例描述
out	返回的人脸框结构体数组	是	std::vector	人脸框结构体（多人则返回最大人脸）
rgbscore	返回的rgb动作活体分值	是	float	
depthscore	返回的depth动作活体分值	是	float	
rgb	传入的opencv视频帧	是	Mat	视频帧
depth	传入的二进制深度数据	是	Char *	视频帧

返回字段描述	
返回字段	int，>=0返回人脸个数，<0时候为错误码
返回示例	

8.15 人脸库管理

sdk提供支持5万以下的人脸库管理，采用的是sqlite数据库，sdk启动后会自动生成db文件夹和face.db文件（人脸库数据文件）、db文件夹可手动删除，删除后人脸数据库即被整体删除，sdk重启后会自动重新创建新库。人脸库数据结构可采用sqlLiteExpert等可视化工具查看人脸库表结构。人脸库创建后有三张表,feature表（用来保存人脸特征值），user表（用来保存人脸用户信息，如userid，groupid以及人脸图片信息，用户信息等）以及user_group表（用户组表）。
人脸库可按组(group_id)划分,组就好比一个集团的子公司，人脸注册或查找既可以按整个库查找，也可以按组（子公司）查找，按组查找速度更快（范围小）。用户id（user_id）是用来标识人脸用户的唯一id，组id（group_id）是用来标识组（子公司）的唯一id。用户信息（user_info）可作为用户id的说明，如标识用户名称、住址等信息，也可不填写。人脸的比对或识别、归根结底是人脸特征值的比对。人脸库的保存实际上是保存了对应用户user_id的特征值在人脸库上，同时保存了用户user_id和group_id及其对应关系（一个用户对应一张人脸、一个特征值数据）。除user_info字段（用户信息）支持中文外，user_id和group_id仅支持英文、数字和下划线的参数组合。人脸1：N识别返回识别的最高分和用户信息，推荐分值超过80为识别成功。
人脸库1：N识别可如如下图所示流程：



8.15.1 人脸注册接口(通过传入图片帧) 方法名| user_add | --- | --- 说明 |用户注册，该接口支持传入opencv图片帧（通常指生活照，1寸或2寸的证件照也可以用这种类型）（图片帧注册会把人脸图片缩略图入库） 函数 |void user_add(std::string& res, const cv::Mat *img, const char* user_id, const char *group_id, const char* user_info="")

参数	说明	必须	类型	示例描述
res	返回的结果	是	string	json 数据
img	Opencv图片帧指针	是	cv::Mat *	人脸图片opencv图片帧指针
user_id	用户id	是	const char*	用户id（由数字、字母、下划线组成），长度限制128B
group_id	组id	是	const char*	用户组id，标识一组用户（由数字、字母、下划线组成），长度限制128B
user_info	用户资料	否	const char*	256个字符以内

返回字段描述(json)	
errno及msg映射	errno Msg 0 Success <0 失败的原因

8.15.2 人脸注册接口(通过传入特征值) 方法名| user_add | --- | --- 说明 |用户注册，该接口通过传入提取的人脸图片特征值注册（特征值注册无法把人脸缩略图入库） 函数 |void user_add(std::string& res, Feature *f1, const char* user_id, const char *group_id, const char* user_info="")

参数	说明	必须	类型	示例描述
res	返回的结果	是	string	json 数据
f1	特征值结构体	是	Feature*	人脸特征值结构体
user_id	用户id	是	const char*	用户id（由数字、字母、下划线组成），长度限制128B
group_id	组id	是	const char*	用户组id，标识一组用户（由数字、字母、下划线组成），长度限制128B
user_info	用户资料	否	const char* 256个字符以内(中文数减半)	

返回字段描述(json)	
errno及msg映射	errno Msg 0 Success <0 失败的原因

8.15.3 人脸更新接口 方法名 |user_update | --- | --- 说明 |人脸更新 函数 |void user_update(std::string& res, const cv::Mat *img, const char* user_id, const char *group_id, const char* user_info="")

参数	说明	必须	类型	示例描述
res	返回的结果	是	string	json 数据
img	opencv图片帧指针	是	cv::Mat*	人脸图片opencv图片帧指针
user_id	用户id	是	const char*	用户id（由数字、字母、下划线组成），长度限制128B
group_id	组id	是	const char*	用户组id，标识一组用户（由数字、字母、下划线组成），长度限制128B
user_info	用户信息	否	const char*	用户资料，长度限制256B

返回字段描述(json)	
errno及message映射	errno Msg 0 Success <0 失败的原因

8.15.4 用户删除接口 方法名 |user_delete | --- | --- 说明 |用户删除 函数| void user_delete(std::string&res, const char *user_id, const char* group_id)

参数	说明	必须	类型	示例描述
res	返回的结果	是	string	json 数据
user_id	用户id	是	const char*	用户id（由数字、字母、下划线组成），长度限制128B
group_id	组id	是	const char*	用户组id，标识一组用户（由数字、字母、下划线组成），长度限制128B

返回字段描述(json)	
errno及msg映射	errno Msg 0 Success <0 失败的原因

8.15.5 创建用户组接口 方法名 | group_add | --- | --- 说明 | 创建用户组 函数| void group_add(std::string&res, const char*

gourp_id)

参数	说明	必须	类型	示例描述
res	返回的结果	是	string	json 数据
group_id	用户组id	是	const char*	用户组id，标识一组用户（由数字、字母、下划线组成），长度限制128B

返回字段描述(json)	
errno及msg映射	errno msg 0 Success <0 失败的原因

8.15.6 用户组删除接口 方法名| group_delete | --- | --- 说明 |用户组删除 函数| void group_delete(std::string&res, const char* group_id)

参数	说明	必须	类型	示例描述
res	返回的结果	是	string	json 数据
group_id	组id	是	const char*	用户组id，标识一组用户（由数字、字母、下划线组成），长度限制128B

返回字段描述(json)	
errno及msg映射	Errno Msg 0 Success <0 失败的原因

8.15.7 用户信息查询接口 方法名| get_user_info | --- | --- 说明 |用户信息查询接口 函数 |void get_user_info(std::string&res, const char *user_id, const char* group_id)

参数	说明	必须	类型	示例描述
res	返回的结果	是	string	json 数据
user_id	用户id	是	const char*	用户id（由数字、字母、下划线组成），长度限制128B
group_id	组id	是	const char*	用户组id，标识一组用户（由数字、字母、下划线组成），长度限制128B

返回字段描述(json)	
errno及msg映射	errno Msg 0 Success <0 失败的原因
data字段	log_id string 请求日志标识 result array 识别结果列表 group_id string 组id face_token string 人脸特征的唯一标识 user_info string 用户信息 create_time string 人脸首次注册时间

8.15.8 用户人脸图片查询接口 方法名| get_user_image | --- | --- 说明 |用户人脸图片查询接口 函数 |int get_user_image(cv::Mat & img, const char *user_id, const char* group_id)

参数	说明	必须	类型	示例描述
img	通过引用返回的图片结果	是	cv::Mat &	
user_id	用户id	是	const char*	用户id（由数字、字母、下划线组成），长度限制128B
group_id	组id	是	const char*	v用户组id，标识一组用户（由数字、字母、下划线组成），长度限制128B

返回字段描述：返回int，0为成功，其他为错误码

8.15.9 用户组列表查询接口 方法名 |get_user_list | --- | --- 说明 |用户组列表查询接口 函数| void get_user_list(std::string&res, const char* group_id, int start = 0, int length = 100)

参数	说明	必须	类型	示例描述
res	返回的结果	是	string	json 数据
group_id	用户组id	是	const char*	用户组id
start	查询起始序号	否	int	默认值为0
length	返回数量	否	int	默认值100，最大值1000

返回字段描述(json)	
errno及msg映射	Errno Msg 0 Success <0 失败的原因
data字段	log_id string 请求日志表识 user_id_list array user_id列表数组

8.15.10 人脸库人脸数量查询 方法名 |db_face_count | --- | --- 说明 |查询数据库人脸数量 (传入组id表示查询该组都人脸数量，null表示查整个库的人脸数量) 函数 |int db_face_count(const char* group_id = nullptr)

参数	说明	必须	类型	示例描述
group_id	组id	是	const char*	人脸分组的组id(传null表示查询整个库的数量)

返回示例：int >=0 (返回的数量)

8.15.11 群组列表查询接口 方法名| get_group_list | --- | --- 说明 |组列表查询 函数 |void get_group_list(std::string& res, int start = 0, int length = 100)

参数	说明	必须	类型	示例描述
res	返回的结果	是	string	json 数据
start	起始序号	否	Int	默认值为0，从0开始
length	返回数量	否	Int	默认值为100，最大为1000

返回字段描述(json)	
errno及msg映射	errno Msg 0 Success <0 失败的原因
data字段	log_id string 请求日志标识 group_id_list array group_id列表数组

8.15.12 人脸识别接口(1:N) (传入opencv图片帧) 方法名|identify | --- | --- 说明 |人脸识别（1：N）,可传入人脸组，比对某个组的所有人脸（人脸库可按单位分组，缩小组范围，减少识别比对时间）、返回最高分的用户及信息 函数 |void identify(std::string&res, const cv::Mat *img, const char* group_id_list, const char* user_id, int type = 0)

参数	说明	必须	类型	示例描述
res	返回的结果	是	string	json 数据
img	opencv图片帧指针	是	cv::Mat *	人脸opencv图片帧指针
group_id_list	组列表	是	const char*	组列表
user_id	用户id	否	const char*	用户id
type	特征值类型	否	shi	特征值类型（0、是生活照、1、证件照2、近红外，参考特征值提取），未传默认为0

返回字段描述	
返回字段	比对分值 float

8.15.13 人脸识别接口(1:N)(传入特征值) 方法名 |identify | --- | --- 说明 |人脸识别（1：N）,可传入人脸组，比对某个组的所有人脸（人脸库可按单位分组，缩小组范围，减少识别比对时间）、返回最高分的用户及信息 函数 |void identify(std::string&res, Feature *f1, const char* group_id_list, const char* user_id, int type = 0)

参数	说明	必须	类型	示例描述
res	返回的结果	是	string	json 数据
f1	起始序号	是	Feature*	特征值结构体
group_id_list	组列表	是	const char*	组列表
user_id	用户id	否	const char*	用户id
type	特征值类型	否	shi	特征值类型（0、是生活照、1、证件照2、近红外，参考特征值提取），未传默认为0

返回字段描述	
返回字段	比对分值 float

8.15.14 人脸识别接口(1:N) (传入opencv图片帧) 方法名 |identify_with_all | --- | --- 说明 |人脸识别（1：N）,和整个人类库比对、返回最高分的用户及信息 函数| void identify_with_all(std::string& res, const cv::Mat *img, int type = 0)

参数	说明	必须	类型	示例描述
res	返回的结果	是	string	json 数据
img	opencv图片帧	是	cv::Mat *	人脸opencv图片帧
type	特征值类型	否	int	特征值类型（0、是生活照，包含1寸照、1、网格证件照2、近红外，参考特征值提取），未传默认为0

返回字段描述	
返回字段	比对分值 float

8.15.15 人脸识别接口(1:N) (传入特征值) 方法名 |identify_with_all | --- | --- 说明 |人脸识别（1：N）,和整个人类库比对 、返回最高分的用户及信息 函数 |void identify_with_all(std::string& res, Feature *f1, int type = 0)

参数	说明	必须	类型	示例描述
res	返回的结果	是	string	json 数据
f1	起始序号	是	Feature *	特征值结构体
type	特征值类型	否	int	特征值类型（0、是生活照、1、证件照2、近红外，参考特征值提取），未传默认为0

返回字段描述	
返回字段	比对分值 float

8.16 sdk系统信息接口

8.16.1 获取sdk版本号接口 方法名 |sdk_version | --- | --- 说明| 通过引用返回sdk版本号 函数 |void sdk_version(std::string& version)

参数	说明	必须	类型	示例描述
version	返回的sdk版本号结果	是	string	如：6.3.3

8.16.2 获取设备指纹接口 方法名 |get_device_id | --- | --- 说明 |通过引用返回sdk的设备指纹信息（可用来标识sdk的唯一性） 函数 |void get_device_id(std::string& version)

参数	说明	必须	类型	示例描述
device_id	返回的sdk设备指纹结果	是	string	如：0BAF96961A8377AB74797B05F7F2C805

8.17 安全驾驶接口

8.17.1 驾驶行为检测接口 方法名| driver_monitor | --- | --- 说明 |驾驶行为检测接口，返回驾驶行为信息 函数 |int driver_monitor(std::vector &out, const cv::Mat *mat)

参数	说明	必须	类型	示例描述
out	返回的驾驶行为结构体数组信息	是	std::vector	DriverMonitor结构体见后续结构体定义
mat	传入的opencv图片帧	是	cv::Mat *	

返回字段描述: int，>=0返回人脸个数，<0时候为错误码

8.17.2 安全带佩戴检测接口 方法名 |safety_belt | --- | --- 说明 |安全带佩戴检测接口，返回佩戴置信度分值 函数 |int safety_belt(std::vector &out, const cv::Mat *mat)

参数	说明	必须	类型	示例描述
out	返回的安全带佩戴结构体数组信息	是	std::vector	SafeBelt结构体见后续结构体定义
mat	传入的opencv图片帧	是	cv::Mat *	

返回字段描述：int，>=0返回人脸个数，<0时候为错误码

9. 结构体描述

9.1 人脸跟踪信息结构体

结构体名	TrackFaceInfo

说明	人脸跟踪信息结构体
	struct TrackFaceInfo
	{
	long face_id;
	FaceBox box;
	Landmarks facelandmarks;
	}

参数	类型	说明
face_id	long	人脸id
box	FaceBox	人脸框结构体
facelandmarks	Landmarks	人脸关键点结构体

9.2 人脸框信息结构体

结构体名	FaceBox

说明	人脸框信息结构体
	struct FaceBox
	{
	int index;
	float center_x;
	float center_y;
	float width;
	float height;
	float angle;
	float score;
	}

参数	类型	说明
index	int	人脸索引值
center_x	float	人脸中心点x坐标
center_y	float	人脸中心点y坐标
width	float	人脸宽度
height	float	人脸高度
angle	float	人脸角度
score	float	人脸置信度

9.3 人脸关键点信息结构体

结构体名	Landmarks

说明	人脸关键点信息结构体
	struct Landmarks
	{
	int index;
	int size;
	float data[144];
	float score;
	}

参数	类型	说明
index	int	人脸关键点索引
size	int	人脸关键点数量（size的值通常为144,72个x，y坐标）
data	float[144]	人脸关键点坐标数组（144）（72个x，y坐标）
score	float	人脸关键点置信度

9.4 人脸特征值结构体

结构体名	Feature

说明	人脸特征值结构体
	struct Feature
	{
	int size;
	float data[128];
	}

参数	类型	说明
size	int	特征值大小（通常为128个float浮点值数据）
data	float[]	128个浮点值数组

9.5 人脸姿态角结构体

结构体名	HeadPose

说明	人脸姿态角结构体
	struct HeadPose
	{
	float yaw;
	float roll;
	float pitch;
	}

参数	类型	说明
yaw	float	左右偏转角
roll	float	与人脸平行平面内的头部旋转角
pitch	float	上下偏转角

9.6 人脸属性信息结构体

结构体名	Attribute

说明	人脸属性信息结构体
	struct Attribute
	{
	int age;
	Race race;
	AttributeEmotionType emotion;
	Glasses glasses;
	Gender gender;
	}

参数	类型	说明
age	int	年龄

race	Race	种族
		enum Race
		{
		BDFace_RACE_YELLOW = 0, // 黄种人<brBDFace_RACE_WHITE = 1, //白种人
		BDFace_RACE_BLACK = 2, // 黑种人
		BDFace_RACE_INDIAN = 3, // 印第安人
		}

emotion	AttributeEmotionType	表情
		enum AttributeEmotionType
		{
		BDFACE_ATTRIBUTE_EMOTION_FROWN = 0,//皱眉
		BDFACE_ATTRIBUTE_EMOTION_SMILE = 1, //笑
		BDFACE_ATTRIBUTE_EMOTION_CALM = 2, //平静
		}

glasses	Glasses	戴眼镜状态
		enum Glasses
		{
		BDFACE_NO_GLASSES = 0 , // 无眼镜
		BDFACE_GLASSES = 1, // 有眼镜
		BDFACE_SUN_GLASSES = 2 //墨镜
		}

gender	Gender	性别
		enum Gender
		{
		BDFACE_GENDER_FEMILE = 0, // 女性
		BDFACE_GENDER_MALE = 1, //
		}

9.7 嘴巴闭合结构体

结构体名	MouthClose

说明	嘴巴闭合置信度结构体
	struct MouthClose
	{
	float score;
	}

参数	类型	说明
score	float	置信度分值

9.8 口罩佩戴结构体

结构体名	MouthMask

说明	口罩佩戴置信度结构体
	struct MouthMask
	{
	float score;
	}

参数	类型	说明
score	float	置信度分值

9.9 最优人脸置信度结构体

结构体名	Best

说明	最优人脸置信度结构体
	struct Best
	{
	float score;
	}

参数	类型	说明
score	float	置信度分值

9.10 人脸模糊度置信度结构体

结构体名	Blur

说明	人脸模糊度置信度结构体
	struct Blur
	{
	float score;
	}

参数	类型	说明
score	float	置信度分值

9.11 人脸光照置信度结构体

结构体名	Illumination

说明	光照置信度结构体
	struct Illumination
	{
	int score;
	}

参数	类型	说明
score	int	置信度分值

9.12 人脸遮挡置信度结构体

结构体名	Illumination

说明	人脸遮挡置信度结构体
	struct Illumination
	{
	float left_eye;
	float right_eye;
	float nose;
	float mouth;
	float left_cheek;
	float right_cheek;
	float chin;
	}

参数	类型	说明
left_eye	float	左眼遮挡置信度
right_eye	float	右眼遮挡置信度
nose	float	鼻子遮挡置信度
mouth	float	嘴巴遮挡置信度
left_cheek	float	左脸遮挡置信度
right_cheek	float	右脸遮挡置信度
chin	float	下巴遮挡置信度

9.13 人眼闭合状态结构体

结构体名	EyeClose

说明	人眼闭合状态结构体
	struct EyeClose
	{
	float left_eye_close_conf;
	float
	right_eye_close_conf;
	}

参数	类型	说明
float left_eye_close_conf	float	置信度分值
float right_eye_close_conf	float	置信度分值

9.14 注意力结构体

结构体名	GazeInfo

说明	注意力结构体
	struct GazeInfo
	{
	Gaze left_eye;
	Gaze right_eye;
	}
	struct Gaze
	{
	GazeDirection direction; // 凝视方向
	float confidence; // 置信度
	}
	struct GazeDirection
	{
	BDFACE_GAZE_DIRECTION_UP = 0, // 向上看
	BDFACE_GAZE_DIRECTION_DOWN = 1, // 向下看
	BDFACE_GAZE_DIRECTION_RIGHT = 2, // 向右看
	BDFACE_GZE_DIRECTION_LEFT = 3, // 向左看
	BDFACE_GAZE_DIRECTION_FRONT = 4, // 向前看
	BDFACE_GAZE_DIRECTION_EYE_CLOSE = 5, // 闭眼
	}

参数	类型	说明
left_eye	Gaze	左眼注意力
right_eye	Gaze	右眼注意力

9.15 静默活体置信度结构体

结构体名	LivenessScore

说明	活体信度结构体
	struct LivenessScore
	{
	int score;
	}

参数	类型	说明
score	float	置信度分值

9.16 驾驶行为结构体

结构体名	DriverMonitor

说明	驾驶行为结构体
	struct DriverMonitor
	{
	float normal;
	float calling;
	float drinking;
	float eating;
	float smoking;
	}

参数	类型	说明
normal	float	行为正常的置信度分值（置信度范围0-1）
calling	float	打电话行为的置信度分值（置信度范围0-1）
drinking	float	喝水的置信度分值（置信度范围0-1）
eating	float	吃东西的置信度分值（置信度范围0-1）
smoking	float	抽烟的置信度分值（置信度范围0-1）

9.17 安全带佩戴结构体

结构体名	SafetyBelt

说明	安全带佩戴行为结构体
	struct SafetyBelt
	{
	float score;
	}

参数	类型	说明
score	float	安全带佩戴的置信度分值（置信度范围0-1）

10. 多端特征值对齐

对于linux arm sdk来说，支持和安卓特征值对齐，linux arm sdk7.x系列，特征值对齐安卓7.x系列，但和如安卓6.x系列或5.x系列，则不对齐，且只支持rgb可见光特征值对齐。
在linux arm sdk中，可通过提取的特征值128个float数组，保存成二进制数据，即和安卓中的512个byte保存成字节流的数据对齐。
sdk中提供了float数组转换为二进制buffer的示例代码以及二进制buffer数据转换为特征值float数组的示例代码。可参考util文件夹中的FeatureAlign类。

11. 错误码及错误信息

各接口返回结果error_code及msg信息如下：

错误码	错误内容	错误描述
0	SUCCESS	成功
-1	ILLEGAL_PARAMS	失败或非法参数
-2	MEMORY_ALLOCATION_FAILED	内存分配失败
-3	INSTANCE_IS_EMPTY	实例对象为空
-4	MODEL_IS_EMPTY	模型内容为空

-5	UNSUPPORT_ABILITY_TYPE	不支持的能力类型
-6	UNSUPPORT_INFER_TYPE	不支持的预测库类型
-7	NN_CREATE_FAILED	预测库对象创建失败
-8	NN_INIT_FAILED	预测库对象初始化失败
-9	IMAGE_IS_EMPTY	图像数据为空
-10	ABILITY_INIT_FAILED	人脸能力初始化失败
-11	ABILITY_UNLOAD	人脸能力未加载
-12	ABILITY_ALREADY_LOADED	人脸能力已加载
-13	NOT_AUTHORIZED	未授权
-14	ABILITY_RUN_EXCEPTION	人脸能力运行异常
-15	UNSUPPORT_IMAGE_TYPE	不支持的图像类型
-16	IMAGE_TRANSFORM_FAILED	图像转换失败
-1001	SYSTEM_ERROR	系统错误
-1002	PARARM_ERROR	参数错误
-1003	DB_OP_FAILED	数据库操作失败
-1004	NO_DATA	没有数据
-1005	RECORD_UNEXIST	记录不存在
-1006	RECORD_ALREADY_EXIST	记录已经存在
-1007	FILE_NOT_EXIST	文件不存在
-1008	GET_FEATURE_FAIL	提取特征值失败
-1009	FILE_TOO_BIG	文件太大
-1010	FACE_RESOURCE_NOT_EXIST	人脸资源文件不存在
-1011	FEATURE_LEN_ERROR	特征值长度错误
-1012	DETECT_NO_FACE	未检测到人脸
-1013	CAMERA_ERROR	摄像头错误或不存在
-1014	FACE_INSTANCE_ERROR	人脸引擎初始化错误
-1015	LICENSE_FILE_NOT_EXIST	授权文件不存在
-1016	LICENSE_KEY_EMPTY	授权序列号为空
-1017	LICENSE_KEY_INVALID	授权序列号无效
-1018	LICENSE_KEY_EXPIRE	授权序序列号过期
-1019	LICENSE_ALREADY_USED	授权序列号已被使用
-1020	DEVICE_ID_EMPTY	设备指纹为空
-1021	NETWORK_TIMEOUT	网络超时
-1022	NETWORK_ERROR	网络错误

12. 常见问题：

12.1 sdk推荐使用交叉编译或在开发板如rk3288上直接编译。

12.2 工程运行过程中，若不能正常运行功能，可在build目录下，生成face_conf.json文件，内容为{"log_open":true}，通过这个json配置文件，可打开sdk的日志模式，运行后会输出及各接口返回的错误码日志等判断问题所在。

12.3 模型文件可定制化：在main方法入口，可在sdk_init方法中传入模型文件夹的绝对路径，达到模型文件定制化的目的。若

不定制化路径，sdk_init中传入null即可，默认模型文件在sdk的models文件夹里面，无需更改。

12.4 激活后是否可以把激活文件license.ini和license.key拷贝到其他设备运行？

不能，离线sdk和设备绑定，每个设备对应一个key和一个license文件，换设备无法运行。但对同一台设备，可把Release下的license.ini和license.key拷贝到本电脑的另外sdk，该设备也等同于激活，可以使用。

12.5 是否支持debug模式？只支持Release模式，不支持debug模式。

12.6 sdk支持armv7hf、armv8等平台，请根据对应平台运行。

12.7 人脸库不支持中文参数？用户信息user_info支持中文，其他人脸管理参数目前暂只支持英文、数字下划线组合模式，工程所在的路径也建议不用放入中文路径中，可能影响人脸库创建。

12.8 特征值多端是否对齐？在人脸7.0系列sdk中，安卓和linux端生活照模式下的特征值都是对齐的。linux中请把提取出来的128个float数据保存成二进制数据（float数组转二进制保存）即可和安卓的512个byte二进制数据对齐。linux中人脸库保存的数据是进行了base64编码保存的。

12.9 sdk是否支持多线程运行？sdk不推荐采用多线程运行。

12.10 海思等其他开发板是否支持？sdk推荐用在rk3288开发板上，其他开发板是armv7hf或armv8平台，也可支持运行、部分第三库如opencv等，若报错，可推荐在对应开发板上编译产出库文件，opencv推荐使用4.1版本。

人脸识别特征值同步接口

人脸识别特征值同步接口

人脸识别特征值同步接口可以实现人脸特征抽取和人脸库的构建。

该接口主要用于在服务端（云端）提取与人脸离线识别SDK通用的人脸特征值，通过调用该接口获取到的人脸特征值及构建的人脸库可以直接导入离线设备端作为人脸离线识别SDK的底库。

特征抽取接口

说明：该接口用于检测图片中的人脸并获得人脸位置信息及特征数据

1. 请求参数

1.1. 请求方法

HTTP方法：`POST`

请求URL： `https://aip.baidubce.com/rest/2.0/face/v1/feature`

URL参数：

参数	值
access_token	通过API Key和Secret Key获取的access_token,参考“Access Token获取”

Header：

参数	值
Content-Type	application/json

1.2. 参数列表

参数	是否必选	类型	说明
image	是	string	图片信息(数据大小应小于10M)
image_type	是	string	图片类型: BASE64:图片的base64值 URL:图片的 URL(下载图片时可能由于网络等原因导致下载图片时间过长)
platform_version	否	string	平台版本（私有化部署版本）可选值有Android_4300、Android_4501、Android_5101、Android_6001、Android_60011、Android_6002、Android_6003、Android_8001、Android_8002、HiSilicon_2000、HiSilicon_2001、Windows_8001、Windows_8002
version	是	string	服务版本（公有云接口版本），当前支持【4300、4501、5101、6001、60011、6002、Android_7001、Android_7002、Android_7003、Android_905D3、Android_8001、Android_8002、Windows_8001、Windows_8002、HiSilicon_2000、HiSilicon_2001、HiSilicon_2003、HiSilicon_3000、RV1109_2000、RV1109_2001、RV1109_2003、RV1109_3000】 4300对应安卓sdk的4.3.0.1版本; 4501对应安卓sdk4.5.0.1版本; 5101对应安卓sdk5.1.0.1版本; 6001: 对应安卓/Windows 6.0通行版; 60011:对应安卓6.0通行红外照版; 6002: 对应安卓/Windows 6.0口罩版； 6003:对应安卓6.0证件照识别模型； Android_7001:对应7.0通用版RGB识别模型； Android_7002:对应7.0通用版RGB&NIR识别模型； Android_7003:对应7.0海外版RGB识别模型； Android_905D3: 对应905D3 1.0 RGB识别模型； Android_8001:对应8.0通用版RGB识别模型； Android_8002:对应8.0通用版RGB&NIR识别模型； Windows_8001:对应8.0通用版RGB识别模型； Windows_8002:对应8.0通用版RGB&NIR识别模型； HiSilicon_2000:对应海思2.0通用版RGB识别模型； HiSilicon_2001:对应海思2.0通用版RGB&NIR识别模型； HiSilicon_2003:对应海思2.0海外版RGB识别模型； HiSilicon_3000:对应HiSilicon 3.0通用版RGB识别模型 RV1109_2000:对应RV1109 2.0通用版RGB识别模型； RV1109_2001:对应RV1109 2.0通用版RGB&NIR识别模型； RV1109_2003:对应RV1109 2.0海外版RGB识别模型； RV1109_3000:对应RV1109 3.0通用版RGB识别模型 备注：只有大版本号一致才能满足不同系统版本SDK的特征值互通，例如安卓6.x和Windows 6.x，新版本安卓8.x和Windows 8.x
max_face_num	否	uint32	最多处理人脸的数目. 默认值为1（仅检测图片中面积最大的那个人脸） 最大值100
prob_threshold	否	float	人脸检测置信度过滤阈值 范围0~1 默认值为0.5
min_face_size	否	uint32	人脸大小过滤阈值 默认值为50(仅检测人脸区域宽度在50以上的人脸)

1.3. 请求示例

```
 {
  "image": "/9j/4AAQSkZJRgABAQAAAQABAAD/2wBDAAgGBgcGBQgHB...",
  "image_type": "BASE64",
  "version": 4300
}
```

2.返回结果

2.1. 参数列表

字段	是否必选	类型	说明
face_num	是	int	图片中的人脸数量
face_list	是	array	人脸信息列表 字段信息见下
location	是	array	人脸在图片中的位置
+center_x	是	float	人脸区域中心点离左边界的距离
+center_y	是	float	人脸区域中心点离上边界的距离
+width	是	float	人脸区域的宽度
+height	是	float	人脸区域的高度
+rotation	是	int64	人脸框相对于竖直方向的顺时针旋转角，[-180,180]
face_probability	是	float	人脸置信度，范围0-1
landmark72	否	array	72个特征点位置 face_field包含landmark时返回 具体对应位置见72个关键点示意图
landmark72_probability	是	float	人脸置信度，范围0-1
feature	是	string	人脸特征信息

2.2. 返回示例

```
 {
"error_code": 0,
"error_msg": "SUCCESS",
"log_id": 2858021767,
"timestamp": 1586767658,
"cached": 0,
"result": {
    "face_num": 1,
    "face_list": [
        {
            "location": {
                "center_x": 77.58989716,
                "center_y": 142.822937,
                "width": 88.65403748,
                "height": 111.686615,
                "rotation": 0
            },
            "landmark72_probability": 0.9992024302,
            "landmark72": [
                {
                    "x": 31.85945511,
                    "y": 134.6939392
                },
```

```
        },
        {
            "x": 34.70507431,
            "y": 149.3773804
        },
        {
            "x": 38.71540833,
            "y": 163.9017944
        },
        {
            "x": 45.37094498,
            "y": 178.0964508
        },
        {
            "x": 58.14535522,
            "y": 190.8776245
        },
        {
            "x": 73.57247925,
            "y": 198.5220337
        },
        {
            "x": 88.43057251,
            "y": 200.3898926
        },
        {
            "x": 101.6749039,
            "y": 194.270813
        },
        {
            "x": 113.2898407,
            "y": 181.7249756
        },
        {
            "x": 120.3853683,
            "y": 167.4402008
        },
        {
            "x": 122.5986481,
            "y": 153.5954437
        },
        {
            "x": 123.1974487,
            "y": 139.8652344
        },
        {
            "x": 122.687851,
            "y": 126.2104416
        },
        {
            "x": 52.40658951,
            "y": 135.2301331
        },
        {
            "x": 56.85477448,
            "y": 132.2627563
        },
        {
            "x": 61.40156555,
            "y": 131.2002563
        },
        {
            "x": 65.94801331,
```

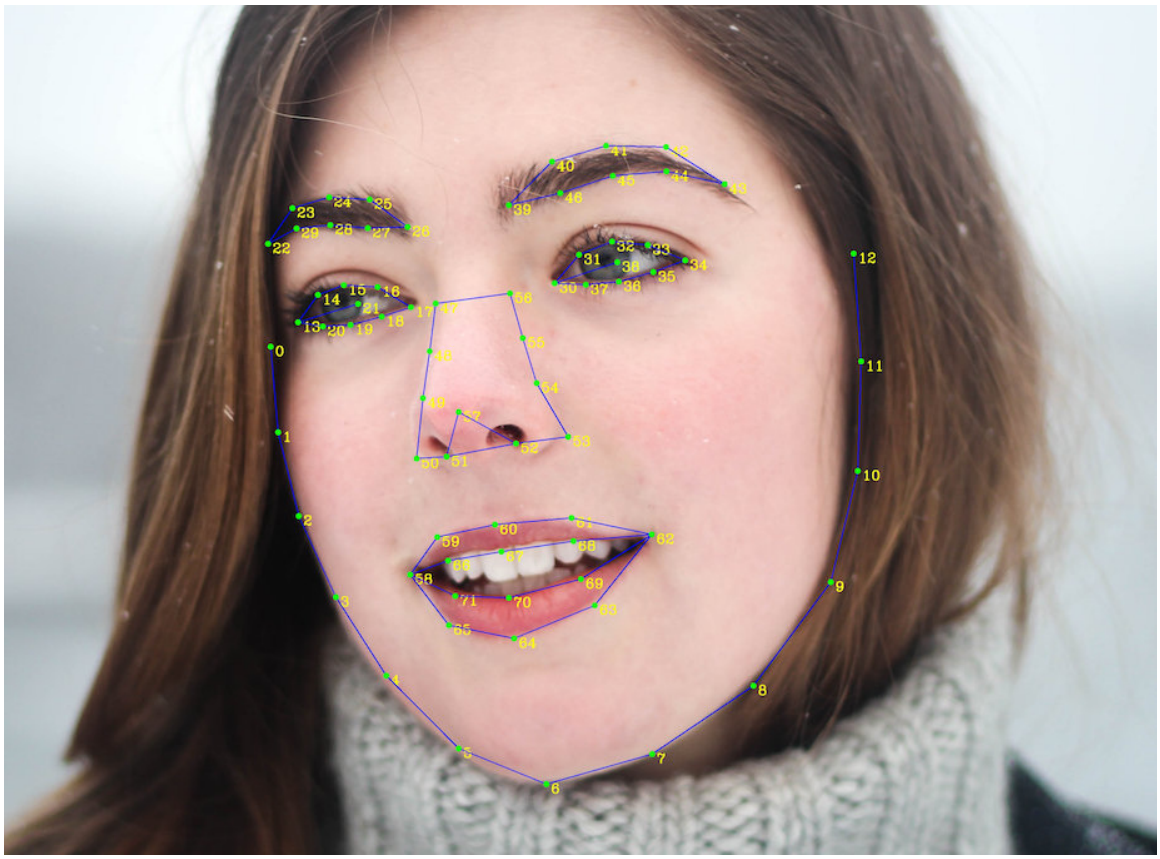
```
            "y": 131.8759308
        },
        {
            "x": 70.11144257,
            "y": 135.0392303
        },
        {
            "x": 65.93842316,
            "y": 135.842926
        },
        {
            "x": 61.48118591,
            "y": 136.3649902
        },
        {
            "x": 56.84508896,
            "y": 136.0989227
        },
        {
            "x": 61.41680908,
            "y": 133.6108093
        },
        {
            "x": 44.35185242,
            "y": 125.7888184
        },
        {
            "x": 50.08662415,
            "y": 120.1439972
        },
        {
            "x": 57.18831635,
            "y": 118.5079346
        },
        {
            "x": 64.31583405,
            "y": 118.9860992
        },
        {
            "x": 71.02742767,
            "y": 122.8297577
        },
        {
            "x": 64.24365234,
            "y": 123.2055359
        },
        {
            "x": 57.56406403,
            "y": 123.3519287
        },
        {
            "x": 50.91134644,
            "y": 124.1832886
        },
        {
            "x": 92.80820465,
            "y": 132.9268494
        },
        {
            "x": 96.26389313,
            "y": 128.844574
        },
```

```
        {
            "x": 100.5915298,
            "y": 127.3882294
        },
        {
            "x": 105.1667709,
            "y": 127.6879349
        },
        {
            "x": 109.6202545,
            "y": 129.9836884
        },
        {
            "x": 105.8541565,
            "y": 131.58078
        },
        {
            "x": 101.5338593,
            "y": 132.6605225
        },
        {
            "x": 97.05542755,
            "y": 132.9283447
        },
        {
            "x": 100.1235352,
            "y": 129.9213257
        },
        {
            "x": 90.2924881,
            "y": 121.1221771
        },
        {
            "x": 96.0561142,
            "y": 116.0347748
        },
        {
            "x": 102.5889816,
            "y": 114.3559265
        },
        {
            "x": 109.3289413,
            "y": 114.6903839
        },
        {
            "x": 115.2678986,
            "y": 119.429985
        },
        {
            "x": 109.318779,
            "y": 118.7310181
        },
        {
            "x": 103.117363,
            "y": 119.074173
        },
        {
            "x": 96.82205963,
            "y": 120.2095337
        },
        {
            "x": 76.32652283,
```

```
            "y": 135.2134247
        },
        {
            "x": 75.47886658,
            "y": 142.9312134
        },
        {
            "x": 74.66484833,
            "y": 150.6994934
        },
        {
            "x": 71.87775421,
            "y": 159.0964966
        },
        {
            "x": 78.33350372,
            "y": 160.4037628
        },
        {
            "x": 91.78503418,
            "y": 159.0786133
        },
        {
            "x": 96.69219971,
            "y": 156.4392853
        },
        {
            "x": 92.56558228,
            "y": 148.8883667
        },
        {
            "x": 90.01399231,
            "y": 141.4549561
        },
        {
            "x": 87.45940399,
            "y": 134.0280151
        },
        {
            "x": 85.36231995,
            "y": 155.7476807
        },
        {
            "x": 66.46245575,
            "y": 172.3835449
        },
        {
            "x": 76.1668396,
            "y": 170.0975342
        },
        {
            "x": 85.70652771,
            "y": 169.4825439
        },
        {
            "x": 94.3141861,
            "y": 168.2900391
        },
        {
            "x": 102.5781555,
            "y": 168.4827271
        },
        {
```

```
            {
                "x": 96.42169952,
                "y": 176.848175
            },
            {
                "x": 86.68830109,
                "y": 180.6125488
            },
            {
                "x": 75.61171722,
                "y": 179.1507874
            },
            {
                "x": 76.39985657,
                "y": 172.5153046
            },
            {
                "x": 85.97942352,
                "y": 172.4254913
            },
            {
                "x": 94.55542755,
                "y": 170.6518555
            },
            {
                "x": 94.6856842,
                "y": 173.494873
            },
            {
                "x": 86.11052704,
                "y": 175.6858673
            },
            {
                "x": 76.53229523,
                "y": 175.497345
            }
        ],
        "feature":
"azwBEORLwZYRJ+YYPHcnHAFyfII8LSUEuYZtCQfWXQ5mrycxpiT2NB7fnLiiCkW9pmX8oOL+eqSUorLXysaPrMijcdDwf/lW2bmoW
,
        "face_probability": 0.9996234179
    }
  ]
}
}
```


3. 72个关键点示意图



Android-SDK-常见问题

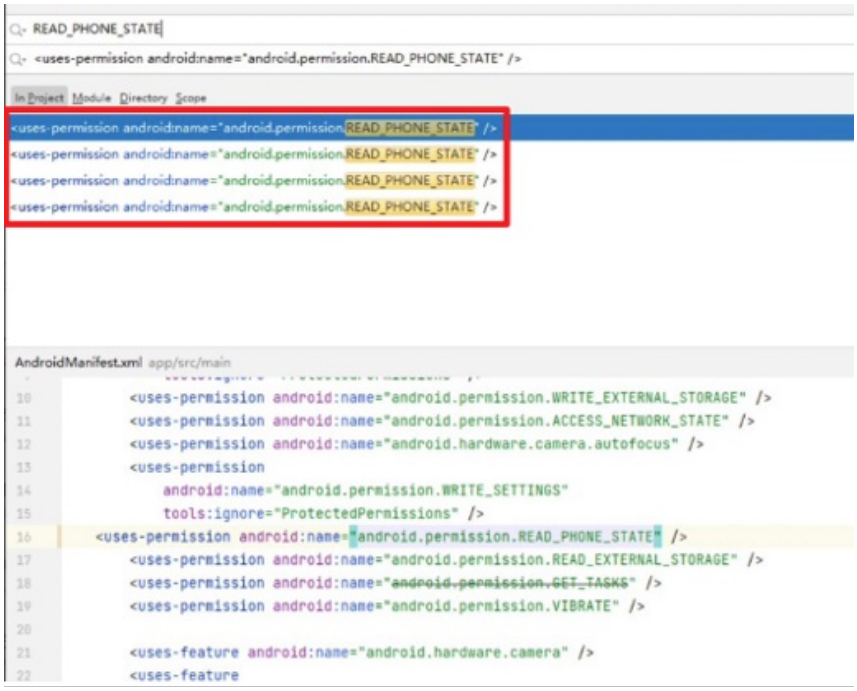
1. Android 8.0人脸离线识别SDK常见问题Q&A

1.1 授权相关问题

- **人脸识别Android SDK no enough params 报错解决方案**

  问题产生原因：no enough params报错出现原因是设备的硬件指纹获取不到，若获取不到硬件指纹设备无法完成激活授权操作，因此也就无法正常使用人脸识别SDK。

  解决方案：将READ_PHONE_STATE 这个权限全局搜索并注释，操作方法如图：



  将注释该权限后的工程重新编译生成apk进行安装。

- **人脸识别Android SDK激活授权时是否需要设备联网**

  离线识别Android SDK共3种激活方式：在线激活、离线激活、批量激活（应用激活）。

需要设备联网的方式是：在线激活、批量激活方式；

无需设备联网的方式是：离线激活方式。

无论选择哪种方式完成的激活授权，在首次激活成功后均可在弱网或无网环境中使用。

- **人脸识别Android SDK时间校验机制原理**

  在完成人脸识别Android SDK的激活鉴权后，由于各种因素如出现设备的时间发生了变化，可能导致SDK报错需重新激活。但序列号仍在有效期内，为什么会出现鉴权失败的问题？

  问题产生原因：出现该问题的原因是设备在鉴权时，SDK内部会保存当前激活的RTC时间，在后续在正常使用的过程中设备的时间仅能往后计数（即实时显示时间大于等于首次激活的时间），若由于设备原因或人为调整时间因素，而出现设备时间小于首次激活时间，SDK会认为该设备还未进行激活鉴权，即当前时刻对应的RTC时间之前从未识别到SDK有过激活授权操作，导致会出现无法使用人脸识别能力的情况。

  解决方案：恢复设备时间为系统时间，并按照文档中的激活流程重新执行一次鉴权流程即可，同时您请在后续使用中也保持系统时间不手动调整时间。

- **批量激活不成功解决方案**

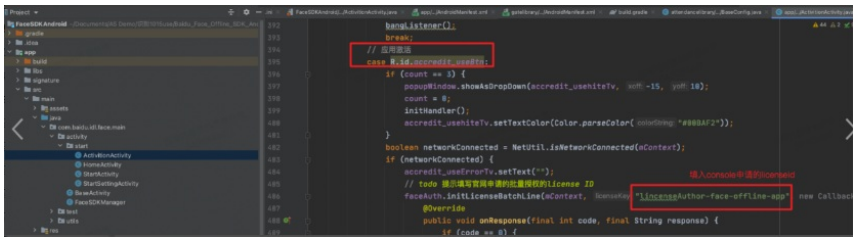
  解决方案：批量激活首先需设备连接网络，在激活过程中实际比对的是控制台自定义的licenseid和应用包名。因此，您先保持设备连接网络，并在控制台新建应用，如图

填入您应用名称、license ID、包名（注意您这里填写的包名需与工程中实际使用的包名一致）

填入完成之后点击创建应用，进入SDK工程的路径：

FaceSDKAndroid/app/src/main/java/com/baidu/idl/face/main/activity/start/ActivitionActivity，配置控制台刚配置的

license Id，如图：



最后编译生成apk推进设备，选择应用激活方式，点击“立即激活”，即可完成授权。

- **设备存储卡与激活授权的关系**

  在完成激活鉴权后，会有激活文件保存到存储卡内，若拔走储存卡安装在另外的设备上会提示该序列号已激活，因为卡内存的是上一个设备的信息，不建议这样操作。

- **人脸识别SDK序列号回收**

  问题背景：如您在控制台已分配了序列号，且序列号对应激活状态是未激活，则您可通过以下流程回收该序列号。

  解决方案：测试序列号暂无法回收，可回收正式序列号。正式序列号回收方法如下：

- **人脸识别Android SDK 激活失败原因**

  您在对人脸识别Android SDK进行激活时，若激活错误，您可参考以下激活错误码对应说明，请参考：

ErrorCode	常量值	说明
SUCCESS	0	成功
LICENSE_NOT_INIT_ERROR	1	license未初始化
LICENSE_DECRYPT_ERROR	2	license数据解密失败
LICENSE_INFO_FORMAT_ERROR	3	license数据格式错误
LICENSE_KEY_CHECK_ERROR	4	license-key(api-key)校验错误
LICENSE_ALGORITHM_CHECK_ERROR	5	算法ID校验错误
LICENSE_MD5_CHECK_ERROR	6	MD5校验错误
LICENSE_DEVICE_ID_CHECK_ERROR	7	设备ID校验错误
LICENSE_PACKAGE_NAME_CHECK_ERROR	8	包名(应用名)校验错误
LICENSE_EXPIRED_TIME_CHECK_ERROR	9	过期时间不正确
LICENSE_FUNCTION_CHECK_ERROR	10	功能未授权
LICENSE_TIME_EXPIRED	11	授权已过期
LICENSE_LOCAL_FILE_ERROR	12	本地文件读取失败
LICENSE_REMOTE_DATA_ERROR	13	远程数据拉取失败
LICENSE_LOCAL_TIME_ERROR	14	本地时间校验错误
OTHER_ERROR	0xff	其他错误

- **使用人脸识别Android SDK新版本始终报错鉴权问题**

  问题背景：如您的该台设备之前已安装过人脸识别Android SDK旧版本，当前使用新版本始终无法完成激活流程。

  解决方案：您可尝试清除设备缓存，重新安装SDK进行激活。

1.2 工程集成问题

- **CPU架构兼容问题**

  SDK支持CPU架构如下：armeabi，arm64-v8a，armeabi-v7a，x86。 如您在使用过程中，出现设备CPU架构的报错，您可根据实际硬件支持的架构选择对应型号即可，请参考：

```
//flavorDimensions "default"
ndk {
    // 设置支持的 SO 库构架
    abiFilters  'armeabi-v7a','armeabi','x86'
    //abiFilters  'armeabi-v7a','armeabi','x86' ,'arm64-v8a'
```

- **离线识别Android SDK存储的人脸特征值大概占多少内存**

  5万的人脸图片底库大概占用内存大小60M~70M。

- **二次开发后查看人脸识别Andorid SDK版本号方法**

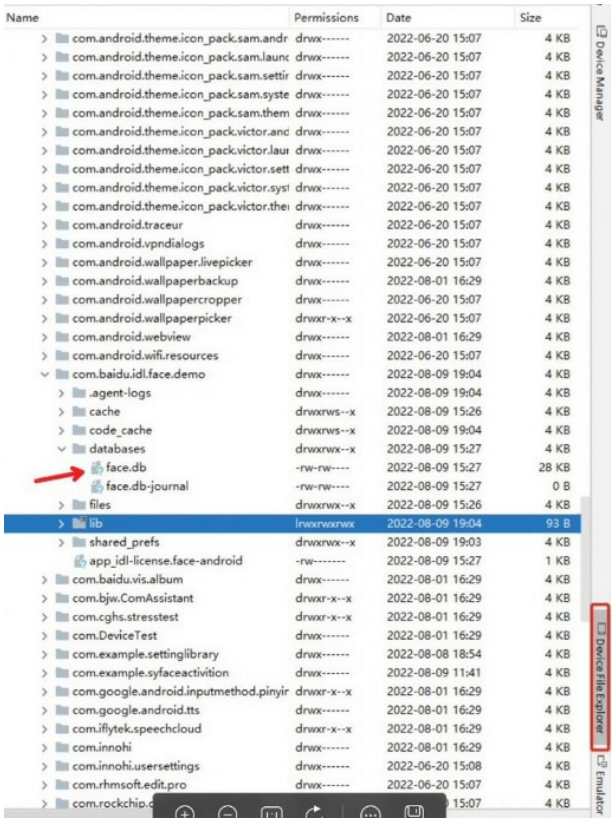
  在您的工程中找到FaceSDK.jar

  通过命令certutil -hashfile 文件名称.文件类型 MD5 ，查看生成的MD5（即为该jar包对应的MD5）。使用相同的方式获取您身边识别SDK所有DEMO版本FaceSDK.jar对应的MD5，其中一致MD5对应的DEMO版本号即为您当前工程中集成使用的人脸识别DEMO版本号。

- **人脸识别Android SDK本地数据库迁移**

  问题背景：如您基于SDK的已有人脸数据库，需将本地设备上的人脸数据迁移到其他设备中，完成人脸库的迁移，您可参考以下方案。

  解决方案：在Android Studio中点击右下角Device File Explorer，找到路径com.baidu.idl.face.demo文件夹下databases/face.db，即可将该文件导入另一设备中同路径下完成人脸库的迁移。



1.3 报错修复问题

- **屏幕宽高layout空指针报错问题**

  问题产生报错日志如下：

  该问题是由setLayoutParams空指针导致。

  解决方案：您注释relativeLayout.setLayoutParams(params);即可，请参考:

```
setContentView(R.layout.activity_face_rgb_gate);
initView();
// 屏幕的宽
int displayWidth = DensityUtils.getDisplayWidth(mContext);
// 屏幕的高
int displayHeight = DensityUtils.getDisplayHeight(mContext);
// 当屏幕的宽大于屏幕宽时
if (displayHeight < displayWidth) {
    // 获取高
    int height = displayHeight;
    // 获取宽
    int width = (int) (displayHeight * ((9.0f / 16.0f)));
    // 设置布局的宽和高
    FrameLayout.LayoutParams params = new FrameLayout.LayoutParams(width, height);
    // 设置布局居中
    params.gravity = Gravity.CENTER;
    relativeLayout.setLayoutParams(params);
}
```

- **人脸识别Android SDK相机码流的照片像素，推荐尺寸**

  人脸识别Android SDK相机码流照片像素推荐尺寸为640×480。

- **人脸框不与人脸实时跟踪问题**

  题背景：您在使用Android 人脸识别SDK中若发现视频流标注人脸框与人脸未实时跟踪对应，您可操作以下方案。

  解决方案：在您使用的模块点击右上角设置-镜头设置-人脸检测框镜像-开启检测框镜像。

- **人脸识别Android SDK人脸注册成功后，重启SDK人脸数据丢失**

  问题背景：若您二次开发后在使用人脸库批量导入功能，若出现重启SDK人脸数据丢失的问题，可通过以下流程排查

  排查方案：排查在人脸批量导入后，是否有调用FaceApi.getInstance().init，如图：

```
131     private void initDBApi(){
132         if (future != null && !future.isDone()) {
133             future.cancel(true);
134         }
135         isDBLoad = false;
136         future = Executors.newSingleThreadExecutor().submit(new Runnable() {
137             @Override
138             public void run() {
139                 FaceApi.getInstance().init(new DBLoadListener() {
140
141                     @Override
142                     public void onStart(int successCount) {
```

  排查这里是否正常执行，若没有则手动调用执行重新编译。

- **人脸识别Android SDK修改topNum参数导致人脸比对不同步问题**

  问题背景：若基于DEMO修改了featureResult对应list获取的topNum参数，导致在进行人脸1:N比对时出现人脸库未实时同步问题。

  解决方案：正常情况在DEMO中，这部分对应的topNum参数是1，如图：

```
// 如果提取特征+检索，调用search 方法
if (featureSize == FEATURE_SIZE / 4) {

    // 特征提取成功
    // TODO 阈值可以根据不同模型调整
    long startFeature = System.currentTimeMillis();
    List<? extends Feature> featureResult = FaceSDKManager.getInstance()
            .getFaceSearch().search(type, 1, feature, true);

    // TODO 返回top num = 1 个数据集合，此处可以任意设置，会返回比对从大到小排序的num 个数据集合
    if (featureResult != null && featureResult.size() > 0) {
```

  这里本身可不用修改，若修改则只能修改为allUserList.size，请参考：

```
.getFaceSearch().search(type, allUserList.size(), feature, isPercent: true);
```

  或者修改为FaceApi.getInstance().getAllUserList().size()，请参考：

```
.getFaceSearch().search(type, FaceApi.getInstance().getAllUserList().size(), feature,
```

- **人脸识别Android SDK livenessmodel为null问题**

  您若在使用过程中发现您设备中打印的livenessmodel始终为null，您可排查设备的视频流是否接入正常，以及确认设备的镜头是否安装倒置。

- **人脸识别Android SDK 导入图片失败原因**

您在对人脸识别Android SDK进行人脸图片导入时，若报错，您可参考以下图片导入失败错误码对应说明，请参考：

导入失败错误码列表

错误码	错误原因
1	图像格式不符合要求
2	图像格式转化失败
3	图像角度不符合要求
4	人脸角度不符合要求
5	人脸图像模糊
6	人脸遮挡
7	光照不符合
8	未检测到人脸
9	检测到多人脸
10	其它

- **人脸识别Android SDK批量激活，控制台报错授权过期**

  问题背景：如您在控制台新建应用对人脸识别Android SDK 批量激活时，报错授权已过期，如下图：

  解决方案：这需为您延长该应用的有效期，您提交工单，我们为您在后台延长应用的有效期即可。

- **人脸识别Android SDK 戴口罩无法识别通过**

  解决方案：在录入人脸时需不带口罩进行录入人脸，注册成功后，您可通过以下两种方式解决戴口罩无法识别问题：

  1.关闭质量检测；

  2.质量检测开关打开：左右脸颊设置1，鼻子设置1，嘴巴设置1，下巴设置1。

  以上两种方式均可。

- **人脸识别Android SDK批量导入方式，识别效果有所欠缺**

  问题背景：如您在使用人脸识别Android SDK人脸图片批量导入方式批量注册人脸，在识别时出现识别效果不是很好，该情况的出现是由于以批量导入方式受限于图片的内容和具体分辨率，可能会有识别效果没有现场人脸录入的效果那么好。

  解决方案：您可尝试通过裁剪图片的背景部分仅保留清晰人脸、筛选模糊人脸图片提高分辨率。

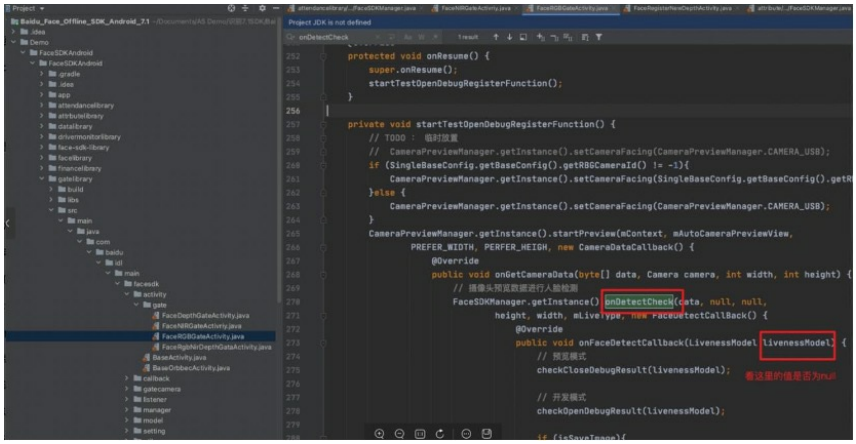
1.4 二次开发指引

- **如何判断人脸识别SDK视频流中已出现人脸**

  人脸识别Android SDK共分包括闸机模式、考勤模式等8个功能模块，应用什么模块，就在对应模块内部进行查看，这里以闸机模式gatelibrary为例进行分析：

  首先进入路径

  FaceSDKAndroid/gatelibrary/src/main/java/com/baidu/idl/main/facesdk/activity/gate/FaceRGBGateActivity，找到FaceSDKManager.getInstance().onDetectCheck，如图：

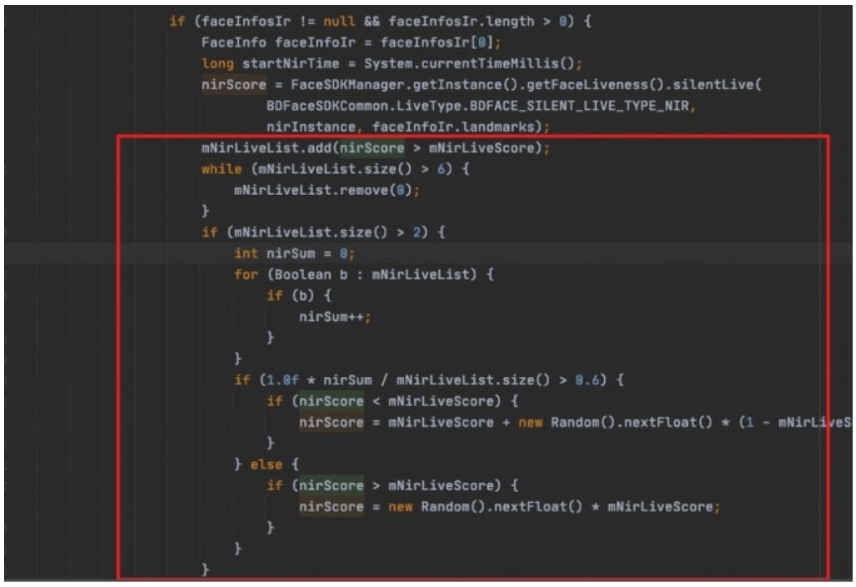
  

  判断图中标注livenessModel是否为null：livenessModel为null代表此时视频流中未检测到人脸；livenessModel不为null代表此时视频流中已检测出人脸。

- **nirlivelist优化问题**

  问题背景：若您在进行二次开发过程中，有nirlivelist报错或有需求简化nirlivelist部分，您可进行以下操作。

  解决方案：您可将SDK中nirlivelist如下对应部分删除，同时rgblivelist也可对应删除，删除后对活体的检测不会有影响。

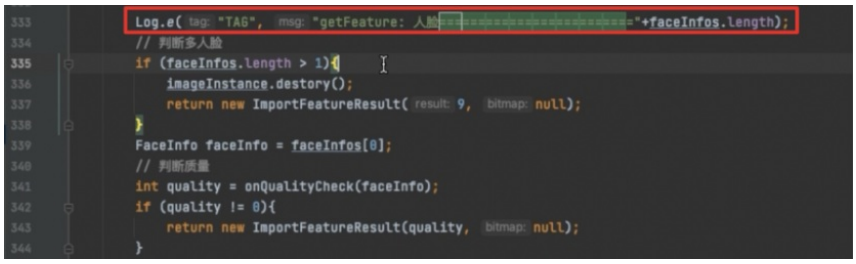


- **Android人脸识别SDK如何判断注册时识别到多少张人脸**

  进入路径

  FaceSDKAndroid/registerlibrary/src/main/java/com/baidu/idl/main/facesdk/registerlibrary/user/manager/ImportFileManager，

  根据图中标注添加日志，如图：

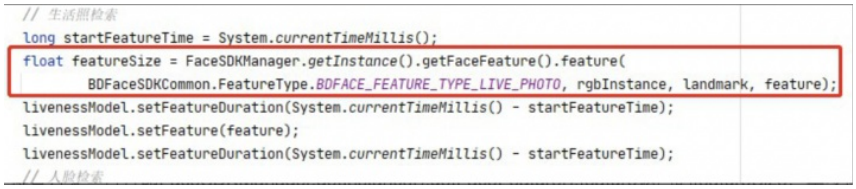


  根据打印结果可查看对应识别到的人脸数。

- **Android人脸识别SDK特征值提取 ret返回-11问题**

  问题背景：正常情况下人脸特征值提取成功返回的ret状态是128，若返回ret -11，应排查以下方法的执行是否正常

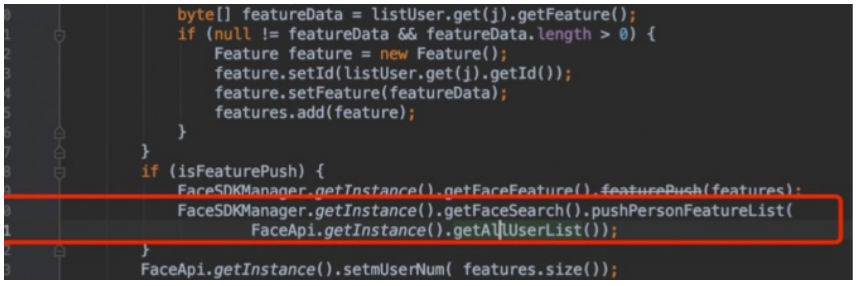
  排查方案：可打印FaceSDKManager.getInstance().getFaceFeature().feature返回的featureSize，单步排查流程执行逻辑



- **人脸识别Android SDK调用pushPersonFeatureList未返回**

  问题背景：在识别SDK中调用pushPersonFeatureList方法没有返回，可通过以下流程排查

  排查方案：打印日志确认getFaceSearch方法返回的FaceSearch对象的属性是否不为空，且已经正常执行进入getFaceSearch方法，如图：



- **人脸识别Android SDK人脸库图片对应路径**

  对于在SDK人脸库中注册成功的人脸库，可在sd卡根目录Success-Import文件夹中进行查看。

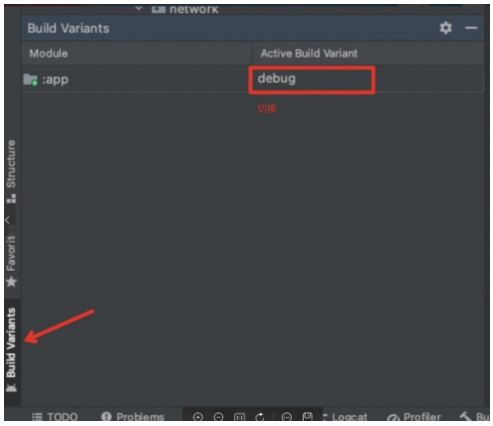
1.5 编译常见问题

- **Android Studio编译报错 Task 'assembleDebug' not found in project ':app'**

如使用Android Studio编译SDK出现如下报错

```
Task 'assembleDebug' not found in project ':app'.
```

解决方案：点击Android Studio左下角Build Variants，切换打包方式（debug与release切换）重新编译



- **人脸识别Android SDK抵御活体攻击方法**

  当前人脸识别Android SDK在抵御活体攻击方面具备以下策略，您可结合您的需求设置：

  1.若您对于安全等级要求较高，您请开启双目模式，包括rgb+nir双目模式或rgb+depth双目模式；

  2.若您需抵御照片翻拍攻击、提前录制视频攻击等方式：

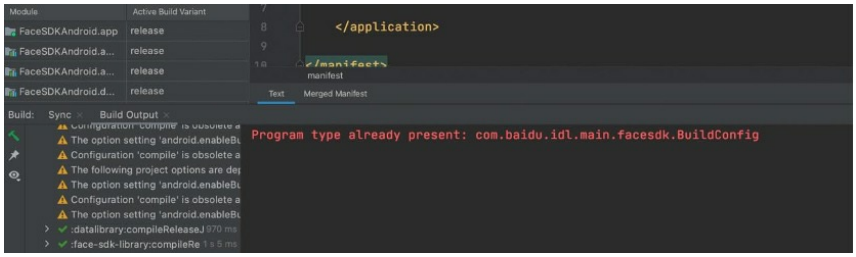
  您可开启rgb+nir双目模式，该模式会对视频中的人像进行nir近红外活体识别，仅对于红外活体过滤通过的人脸会通过进行下一步的特征值提取及比对，对照片、视频等一系列平面的攻击方式有较好效果；

  3.若您需抵御佩戴人脸模具等攻击方式

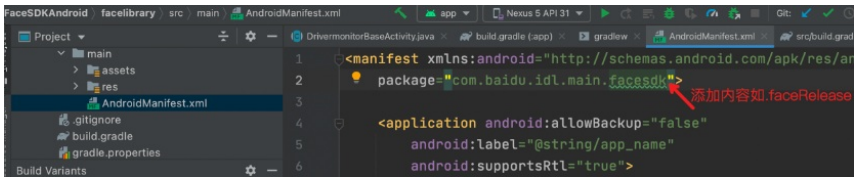
  您可开启rgb+depth双目模式，该模式对于人脸模具等立体结构的攻击有较好效果。

- **人脸识别Android SDK release版本编译报错"Program type already present: com.baidu.idl.main.facesdk.BuildConfig"**

  问题背景：如您在使用人脸识别Android SDK对release版本编译时报错"Program type already present: com.baidu.idl.main.facesdk.BuildConfig"，如下：



  解决方案：正式包会进行混淆并且需要手动录入基础信息，因此您需修改package信息，这里以Demo包名为例，您进入facelibrary->src->main->AndroidManifest.xml按以下修改，重新编译即可。



1.6 特征值端云同步问题

- 人脸特征值在多平台通用问题

  问题背景：调用人脸特征值同步接口URL： https://aip.baidubce.com/rest/2.0/face/v1/feature，该接口返回的结果feature如下

  该接口返回的feature若需与Android、Windows等平台进行通用，需通过解密流程才能完成。

解密方法：调用Base64.decode(contentBase.getBytes(),Base64.DEFAULT)方法解密，详细操作如图：

- **人脸图片的特征值提取**

  问题：

  如何获取对应人脸图片的特征值

解决方案：

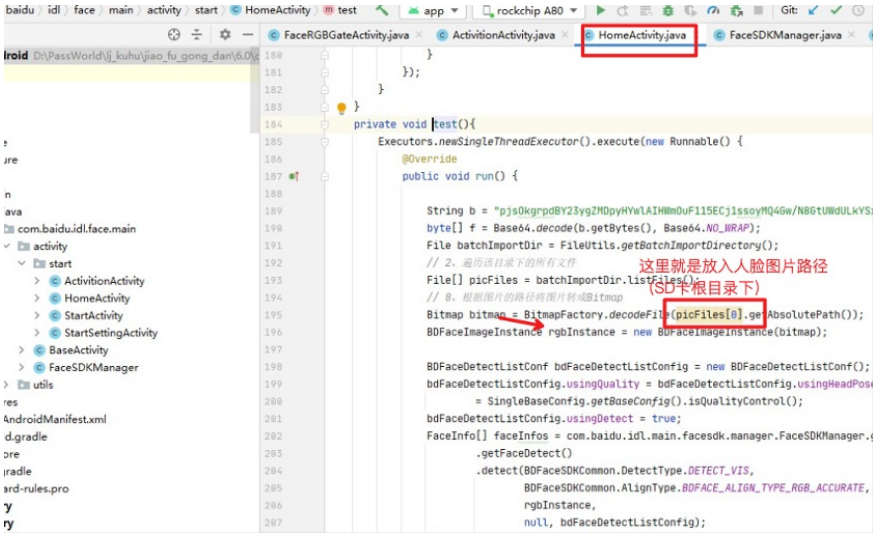
首先在这里下载工程

链接: https://pan.baidu.com/s/1EFi-CW2FNZOL1R-iRh2F5g 提取码: abq6

1.导入待测试图片

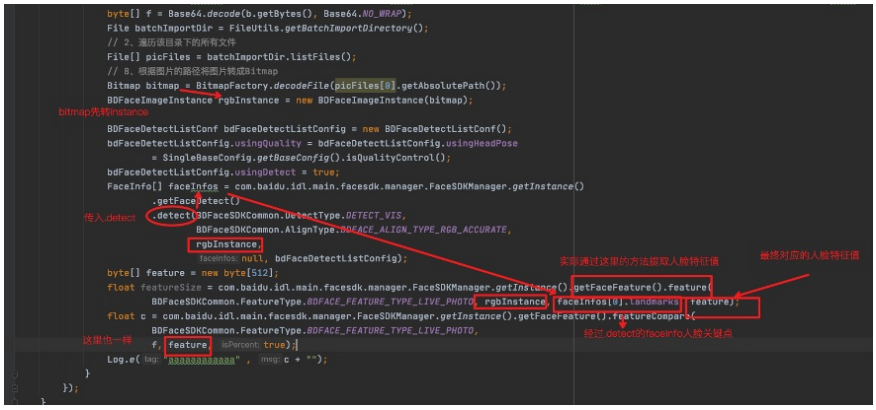
将您需提取特征值的人脸图片，放入SD卡根目录test文件夹下（如果没有test文件夹您手动新建并放入即可）

2.进入工程HomeActivity文件

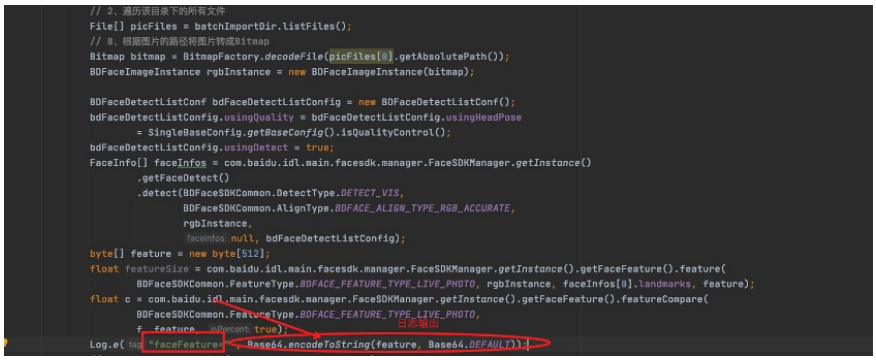


3.实际获取人脸图片特征值

以下只是对提取特征值的逻辑进行介绍，不用实际操作



4.输出打印特征值



这里打印输出的faceFeature即为传入人脸图片对应的特征值信息。

2. Andorid人脸识别SDK人证核验无法通过解决方案

问题：

人脸识别Android SDK使用人证核验模式无法比对通过问题。

解决方案如下：

（以下方案依次执行即可，不用全部操作）

1.关闭质量检测，将人脸最小识别px调至30px

2.增大读卡器读取的身份证图片分辨率，如您是通过读卡器等外置设备，提取身份证、卡证等人脸图片，您可适当提高人脸图片的分辨率。

3.阈值调整

证件照1:1比对，推荐阈值70（对于老年图片或照片遮挡模糊度较高还可继续往下调整）；

证件照1:N比对，推荐阈值80。

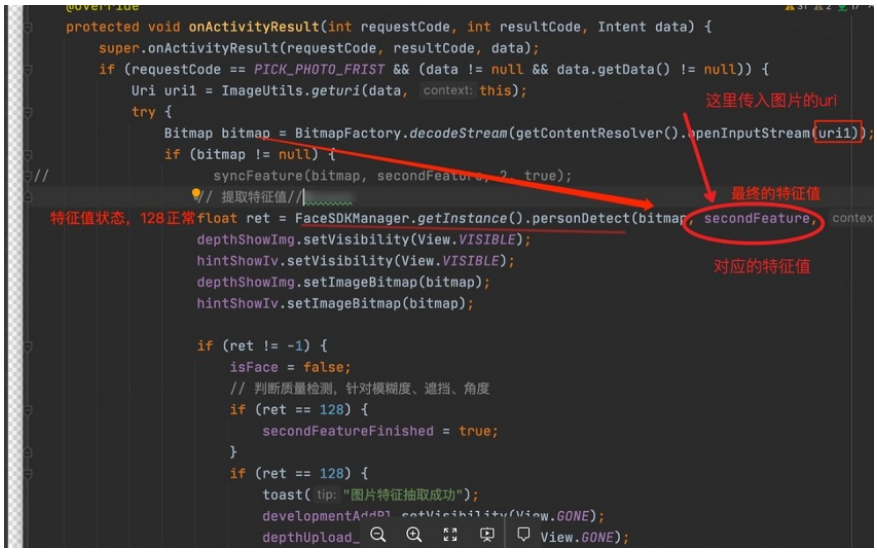
4.手动拍摄身份证照片传入SDK

5.打印log，看逻辑 对具体流程进行比对。

以下是对于仍不能认证通过的排查步骤：

（a）排查特征值提取是否正常

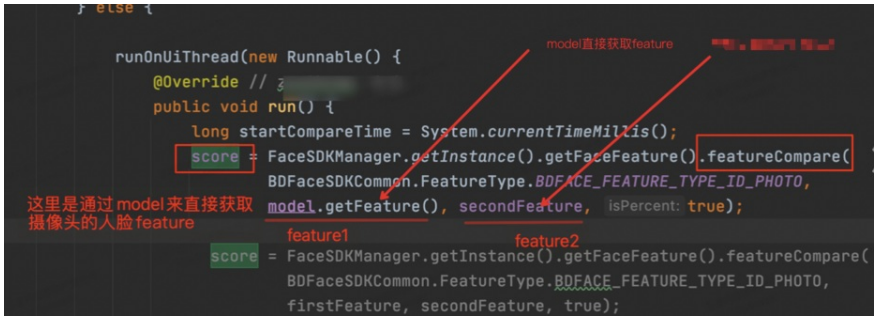
identifylibrary模块，onActivityResult中uir1中传入证件照片，返回的secondFeature为对应的人脸特征值



ret返回128，secondFeature有值，则证件特征值提取成功。

（b）排查证件照与人脸视频流比对是否正常比对

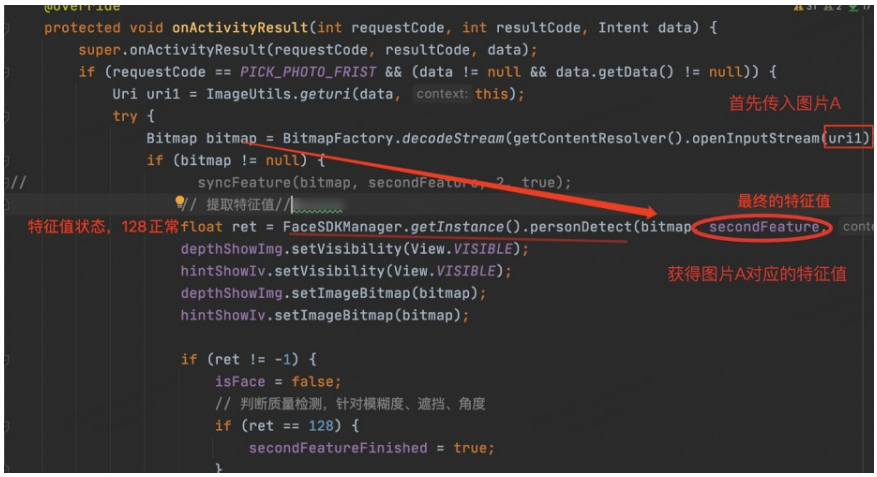
传入刚刚获取的图片特征值secondfeature，视频流中人脸的特征值SDK会自动获取并保存。确定图中标注视频流人脸特征值feature1与证件特征值feature2是否正常



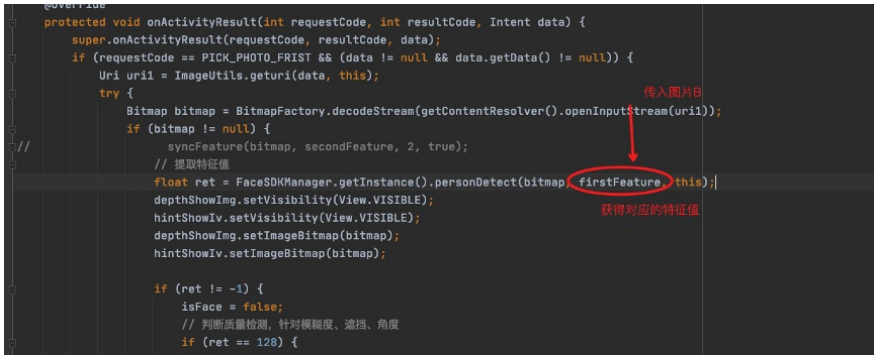
使用featureCompare进行识别比对，根据返回的score与自定义阈值比较，判定是否比对通过。

（c）若用户使用两张图片格式进行人脸比对，则根据demo进行二次开发：

首先获得人脸图片1的特征值，对应secondFeature：



相同方式获得人脸图片2的特征值，对应firstFeature:



两张图片进行比对，返回score，根据阈值进行自定义结果返回

此外，若用户上传的证件照为生活照格式，则这里也可对应进行替换

```
// 人证核验
public float personDetect(final Bitmap bitmap, final byte[] feature, Context context) {
    BDFaceImageInstance rgbInstance = new BDFaceImageInstance(bitmap);
    float ret = -1;
    BDFaceDetectListConf bdFaceDetectListConfig = new BDFaceDetectListConf();
    bdFaceDetectListConfig.usingQuality = bdFaceDetectListConfig.usingHeadPose
            = SingleBaseConfig.getBaseConfig().isQualityControl();
    bdFaceDetectListConfig.usingAttribute = SingleBaseConfig.getBaseConfig().isAttribute();
    bdFaceDetectListConfig.usingDetect = true;

    FaceInfo[] faceInfos = faceDetectPerson.detect(BDFaceSDKCommon.DetectType.DETECT_VIS,
            BDFaceSDKCommon.AlignType.BDFACE_ALIGN_TYPE_RGB_ACCURATE,
            rgbInstance,
            null, bdFaceDetectListConfig);

    if (faceInfos != null && faceInfos.length > 0) {
        // 判断质量检测，针对模糊度、遮挡、角度
        if (qualityCheck(faceInfos[0], context)) {
            ret = faceFeaturePerson.feature(BDFaceSDKCommon.FeatureType.
                    BDFACE_FEATURE_TYPE_ID_PHOTO, rgbInstance, faceInfos[0].landmarks, feature);
        }
    } else {
```

替换为

```
BDFACE_FEATURE_TYPE_LIVE_PHOTO,
```

3. SDK报错Gradle文件适配问题解决方案

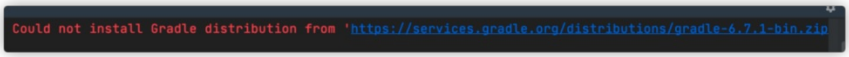
问题：

在运行AS build时报错gradle文件不匹配问题。

解决方案如下：

在使用Android Studio过程中遇到的Gradle 报错适配问题可参考如下解决方案： **Gradle 适配**

在Mac 版Android Studio打开工程时可能会遇到以下报错（windows版流程一样 路径参照修改）

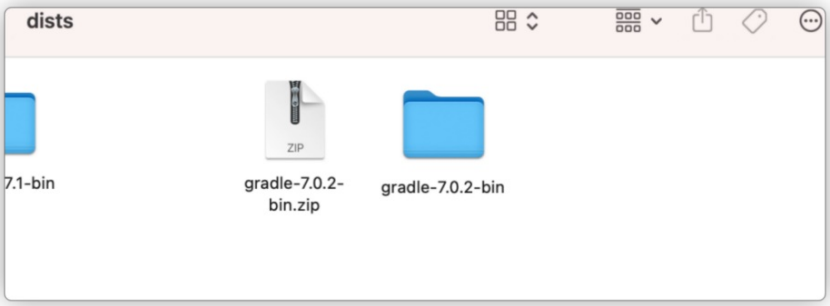


解决方式如下：

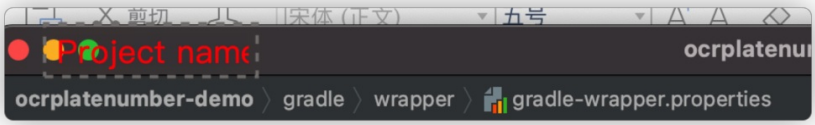
1.在https://services.gradle.org/distributions/gradle-7.0.2-bin.zip 中，将文件下载到本地。

2.下载下来后，在命令终端执行如下命令打开Android Studio根路径配置的gradle，其中标红换为自己的user

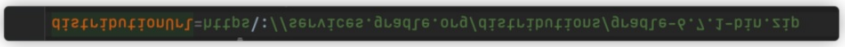
3.打开后将网上下载下来的gradle-7.0.2-bin.zip复制到以上dists目录下并解压



4.在项目路径下—gradle—wapper—gradle-wrapper.properties

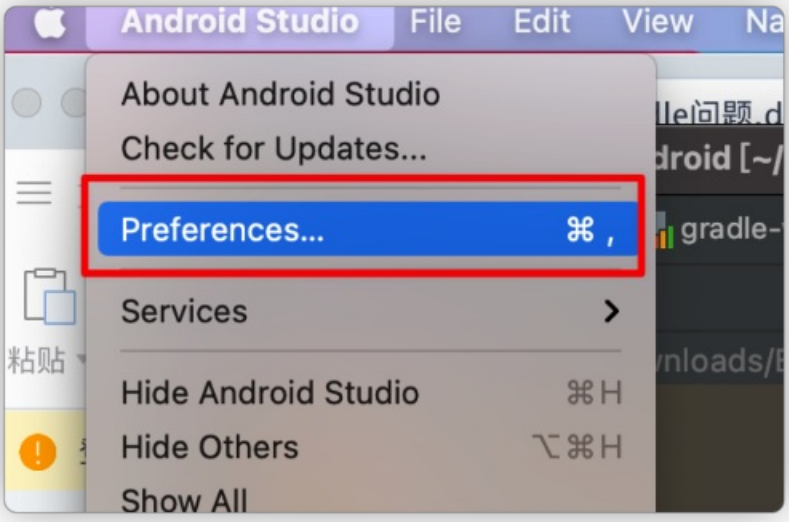


修改以下配置

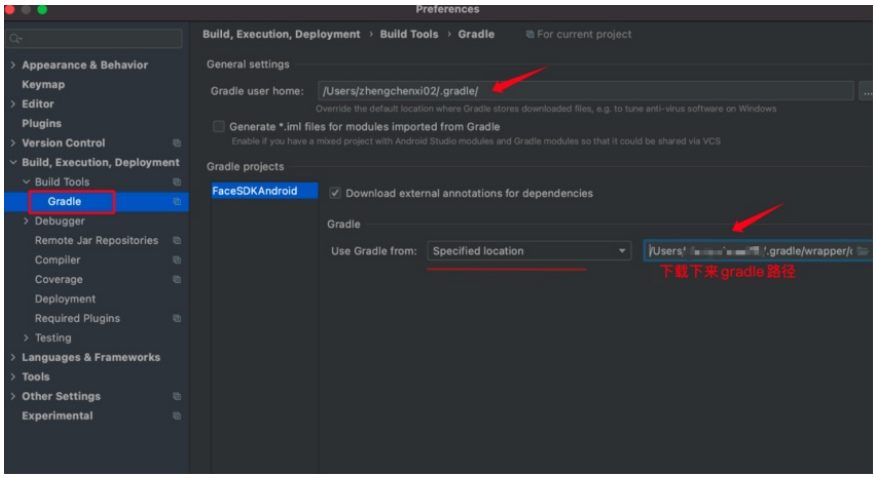


对应修改为以下参数

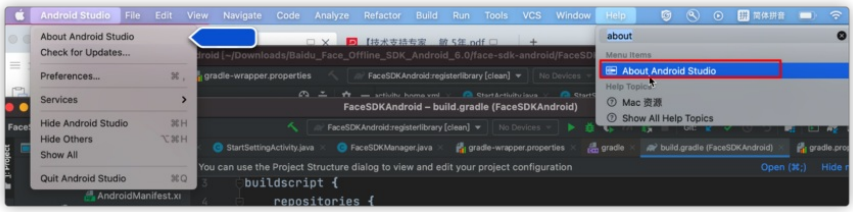
5.打开preferences



将找到gradle配置项进行修改，选择apply

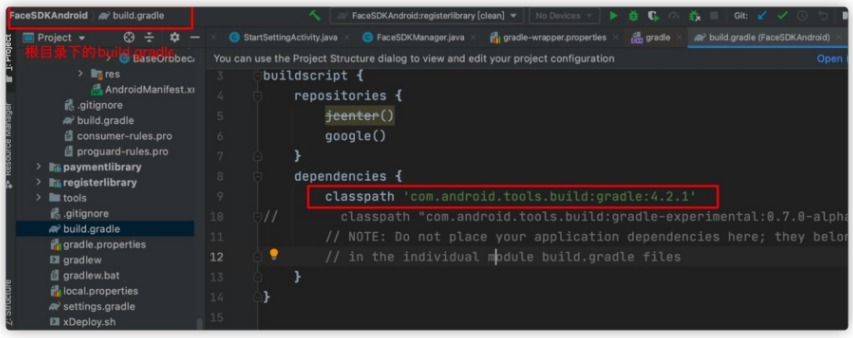


6.至此完成gradle文件的适配，但同时需要更改dependences

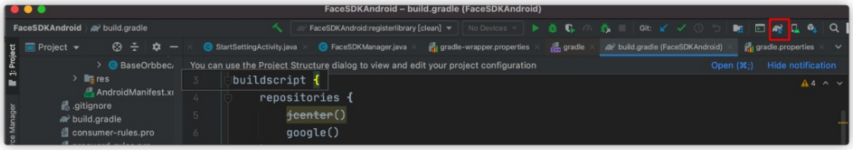


可以看到改版本对应为4.2.1

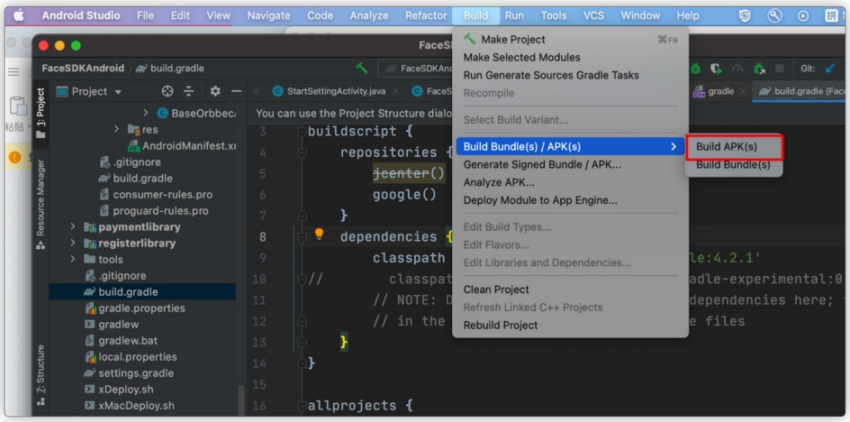
修改dependences指向为目标版本进行对应



7.找到右上角Sync Projects with Gradle Filles,运行一下



8.同步成功后，build apks，将编译没问题的工程build为apk，可以直接推进手机。



完成

此外如果需要引用sdk，首先将sdk copy到根目录，并在根目录/build.gradle文件中添加以下部分：

dependencies {compile fileTree(include: ['*.jar'],dir: 'libs')}

4. 人脸识别7.X 版本SDK重启设备需重新激活解决方案

问题：

在7.X版本的人脸识别Android SDK，存在问题：设备通过在线激活成功后，断网重启的情况下需要重新激活授权问题。

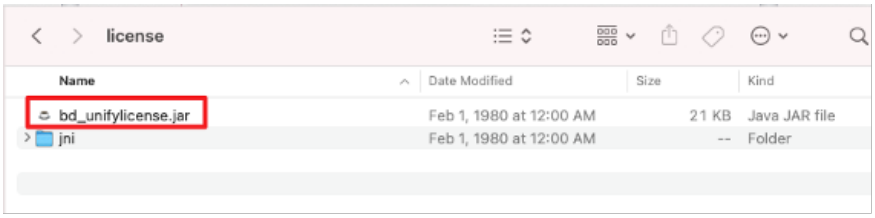
解决方案：

首先下载license文件，下载地址：https://ai-toolbox.bj.bcebos.com/AndroidTool/licenseFix.rar?authorization=bce-auth-v1/2a5115b018924a7bb2405efb5392855f/2022-09-21T02%3A27%3A17Z/-1/host/0311b5a4babfb7fb6347f52c07022086083716e9cadd8fbb1ad15e1b6e8b4410

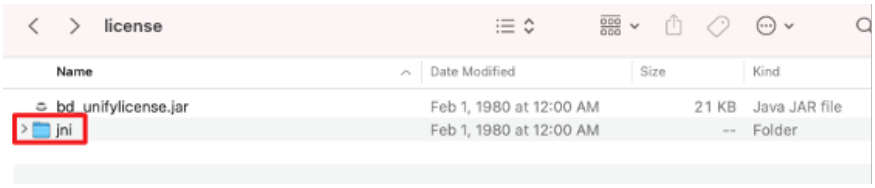
解压后关注两处文件：

（1）jar包

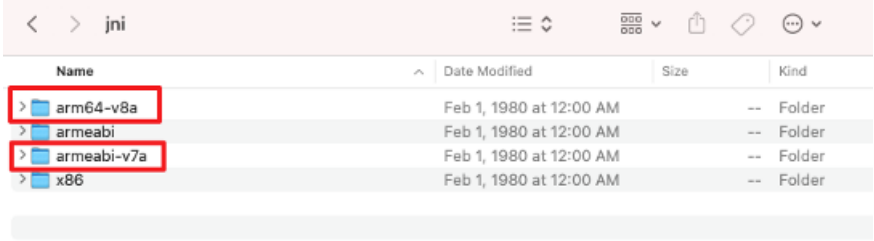
进行拷贝bd_unifylicense.jar，如图：



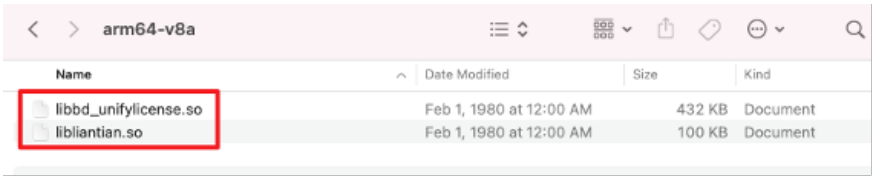
（2）进入jni文件

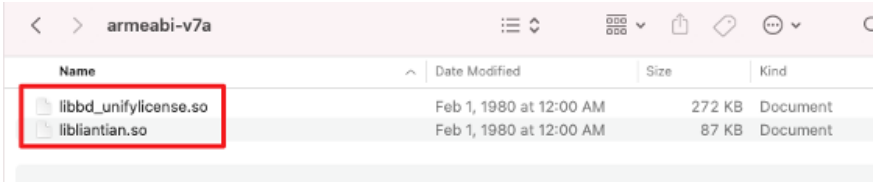


再次进入arm64-v8a与armeabi-v7a



分别进入，拷贝以下文件，如图：

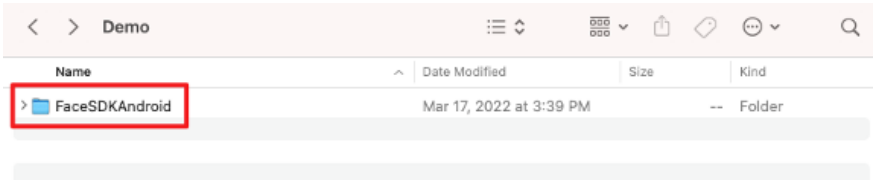




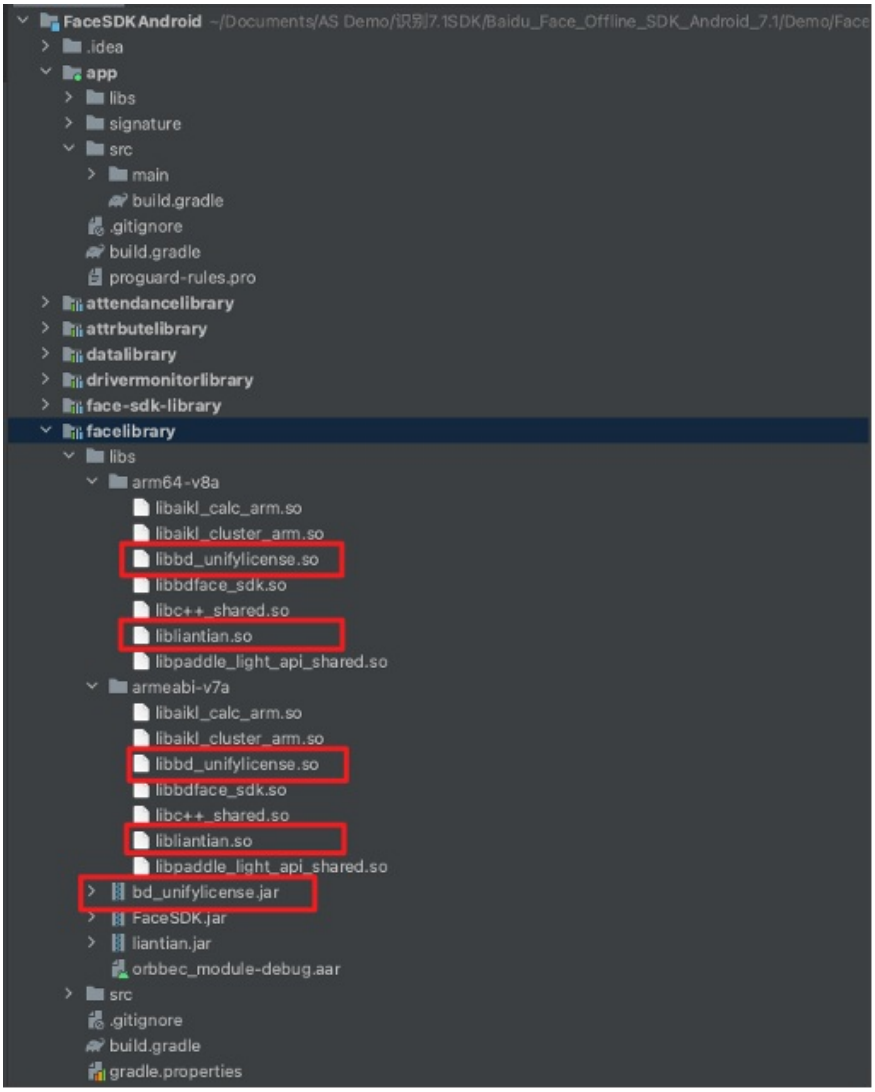
共拷贝这三处文件，至此完成需要替换文件的拷贝。

进入原正在使用的SDK工程路径，如图

再进入FaceSDKAndroid



使用Android Studio打开工程，并将以上三份文件替换到如下文件中，如图： （注意仅替换文件，不是整体文件夹）



替换后在AS上方，点击Build-Clean Project

执行成功后，编译APK推进设备安装，该APK即可修复在首次激活成功后，重启设备即使在无网络环境下也无需再次激活授权流程。

5. 人脸识别Android SDK镜头问题解决方案

问题：

用户在人脸识别SDK中， 切换前后镜头（或多个镜头设备切换指定镜头） 未生效； 点击旋转镜头角度 未生效。

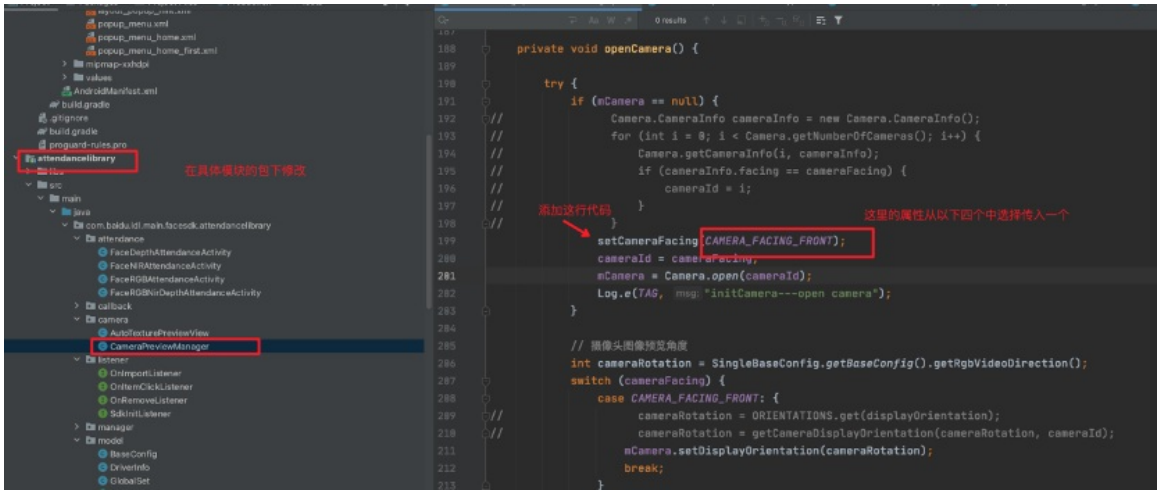
解决方案：

以下分两个问题解决：

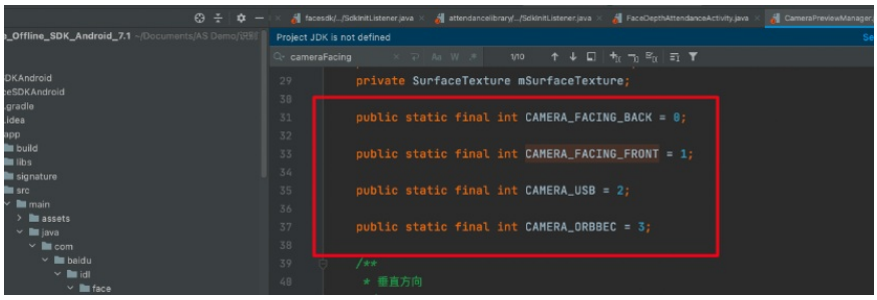
1.切换前后置镜头问题

解决方案

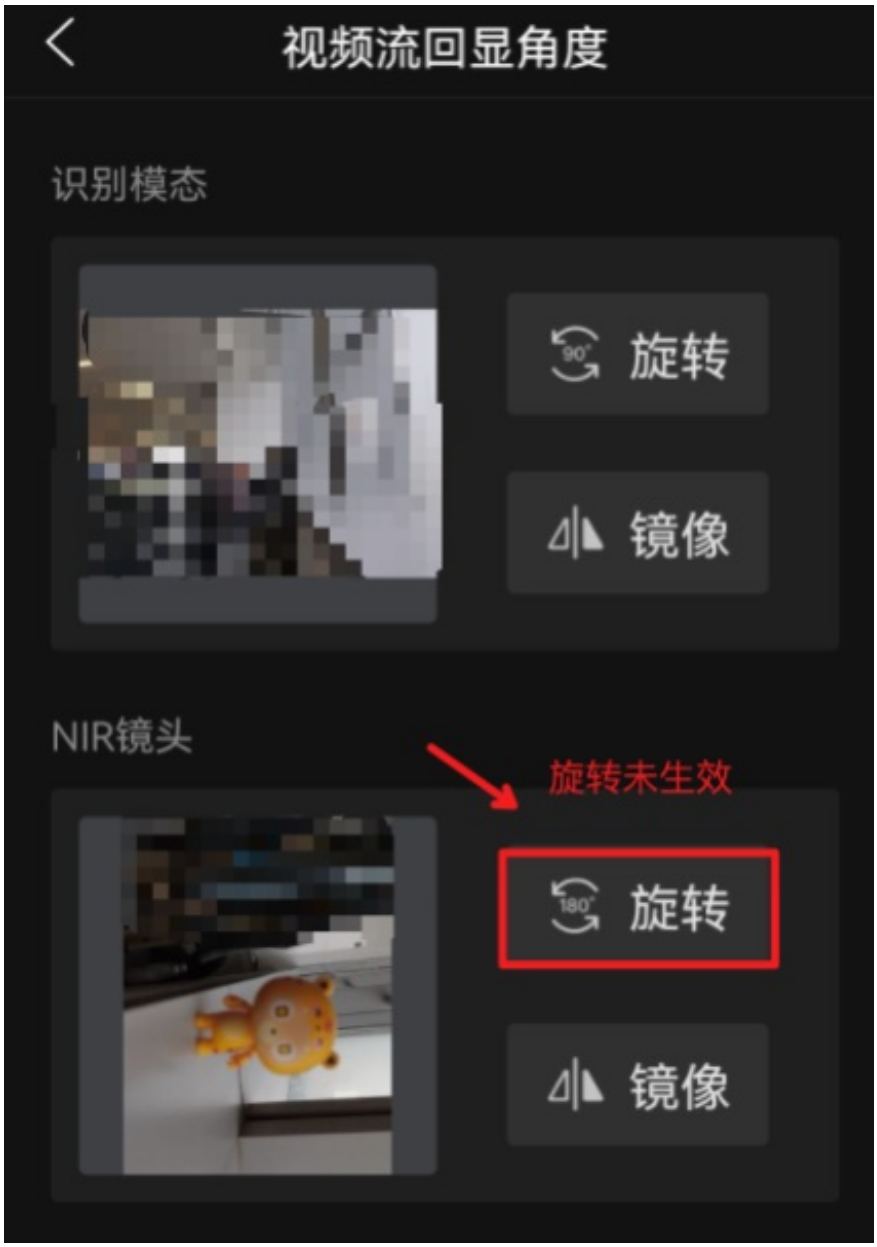
在具体模块（ 这里以考勤模式attendancelibrary为例）CameraPreviewManager添加以下代码



标红框中的参数替换为以下之一（具体哪一个与设备的实际情况有关，需要实际测试）

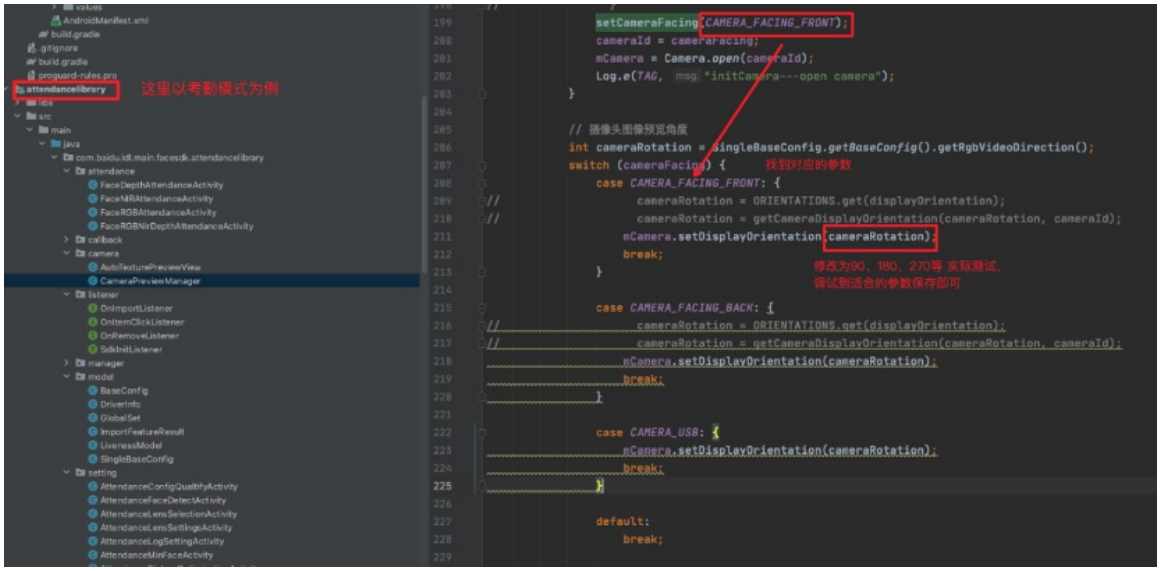


2.切换显示角度，修改回显角度不生效，镜头属于旋转90度问题



SDK中角度对应的实际代码位置如下

重要：本处是举例修改8个模块中具体一个，这里是考勤模式attendace下的CameraPreviewManager文件，实际上共有8个模块可修改（使用哪个模块修改哪个模块对应的文件即可）



同时若在人脸注册时视频流仍是旋转的，对应找到需注册模块 registerlibrary的CameraPreviewManager文件，同步修改。

注意：您的每次修改都需将设备中历史SDK删除，并通过重新安装方式保证参数修改生效。

6. 人脸识别SDK 1：N人脸搜索逻辑分析

人脸识别SDK1:N整体流程如下：

init资源——暗光恢复——.track绘制人脸框——.detect进一步获取详细人脸信息——最优人脸控制——质量检测——.silentlive活体检测（rgb活体校验、nir活体校验、depth活体校验）——.feature提取特征值——.search人脸搜索1：N返回结果。

1.initModel初始化资源

code=0 代表资源加载成功，否则根据错误码排查模型初始化问题。

```
private void initListener() {
    if (FaceSDKManager.initStatus != FaceSDKManager.SDK_MODEL_LOAD_SUCCESS) {
        FaceSDKManager.getInstance().initModel(this, new SdkInitListener() {
            @Override
            public void initStart() {
            }
```

在内部初始化各类模型资源

各类模型初始化

```
faceDetect.initModel(context,
        GlobalSet.DETECT_VIS_MODEL,
        GlobalSet.ALIGN_RGB_MODEL, BDFaceSDKCommon.DetectType.DETECT_VIS,
        BDFaceSDKCommon.AlignType.BDFACE_ALIGN_TYPE_RGB_ACCURATE,
        new Callback() {
            @Override
            public void onResponse(int code, String response) {
                //  ToastUtils.toast(context, code + "  " + response);
                if (code != 0 && listener != null) {
                    listener.initModelFail(code, response);
                }
            }
        });

faceDetectNir.initModel(context,
        GlobalSet.DETECT_NIR_MODE,
        GlobalSet.ALIGN_NIR_MODEL, BDFaceSDKCommon.DetectType.DETECT_NIR,
        BDFaceSDKCommon.AlignType.BDFACE_ALIGN_TYPE_NIR_ACCURATE,
        new Callback() {
            @Override
            public void onResponse(int code, String response) {
                //  ToastUtils.toast(context, code + "  " + response);
                if (code != 0 && listener != null) {
                    listener.initModelFail(code, response);
                }
            }
        });
```

onDetectCheck 开始检测识别

rgbData对应为使用rgb摄像头获取人脸数据信息、irData对应为使用nir摄像头获取人脸数据信息

```
}
if (rgbData != null && irData != null) {
    FinanceFaceSDKManager.getInstance().onDetectCheck(rgbData, irData,  depthData: null,
            PERFER_HEIGH, PREFER_WIDTH,  liveCheckMode: 2, new FaceDetectCallBack() {
        @Override
        public void onFaceDetectCallback(LivenessModel livenessModel) {
            // 输出结果
            if (mAutoCameraPreviewView.isDraw) {
                // 预览模式
                checkCloseDebugResult(livenessModel);
            } else {
                // 开发模式
                checkOpenDebugResult(livenessModel);
            }
        }
```

拿到摄像头获取图片数据

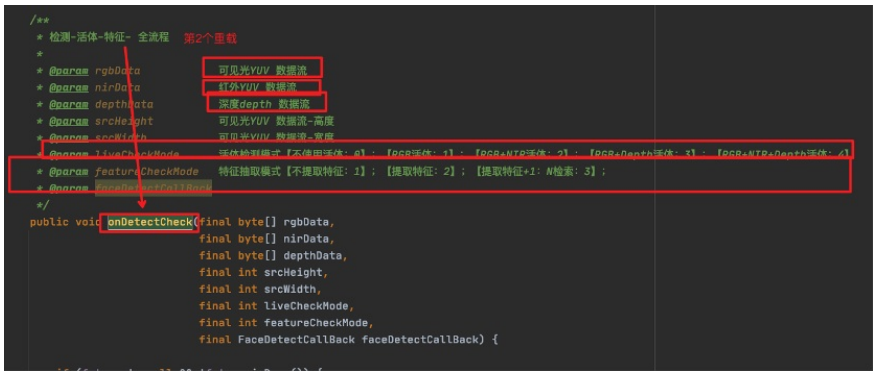
onDetectCheck 检测-活体-特征-人脸检索流程，两个入口

获取faceInfo[0]最优质人脸信息

a.

```
/**
 * 检测-活体-特征-人脸检索流程
 *
 * @param rgbData          可见光YUV 数据流
 * @param nirData          红外YUV 数据流
 * @param depthData        深度Depth 数据流
 * @param srcHeight        可见光YUV 数据流-高度
 * @param srcWidth         可见光YUV 数据流-宽度
 * @param liveCheckMode    活体检测类型【不使用活体: 0】; 【RGB活体: 1】; 【RGB+NIR活体: 2】; 【RGB+Depth活体: 3】; 【RGB+NIR+Depth活体: 4】
 * @param faceDetectCallBack
 */
public void onDetectCheck(final byte[] rgbData,
                          final byte[] nirData,
                          final byte[] depthData,
                          final int srcHeight,
                          final int srcWidth,
                          final int liveCheckMode,
                          final FaceDetectCallBack faceDetectCallBack) {
```

b.



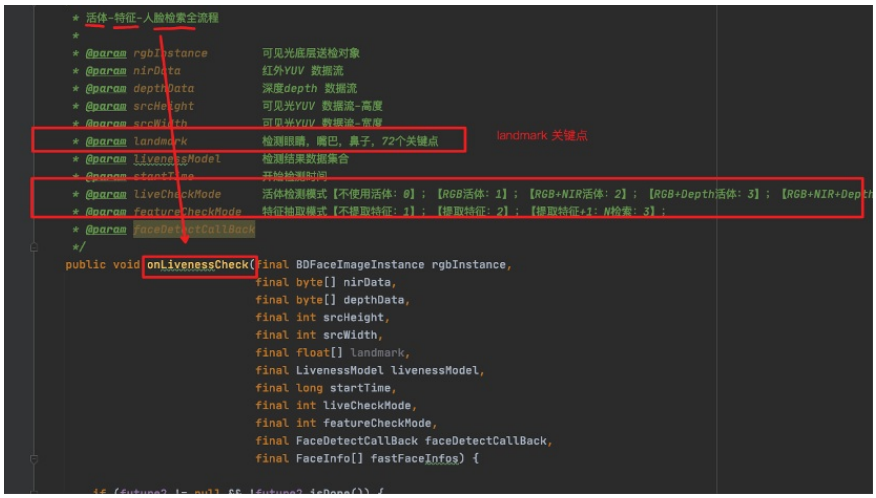
在该函数内部，首先rgbData转Instance

2.判断暗光恢复

3.绘制人脸框

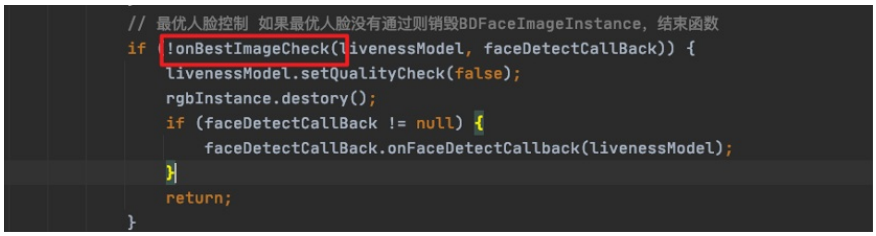
track()获 取到的人脸初步信息存入FaceInfo[]数组，且绘制人脸框。

onLivenessCheck



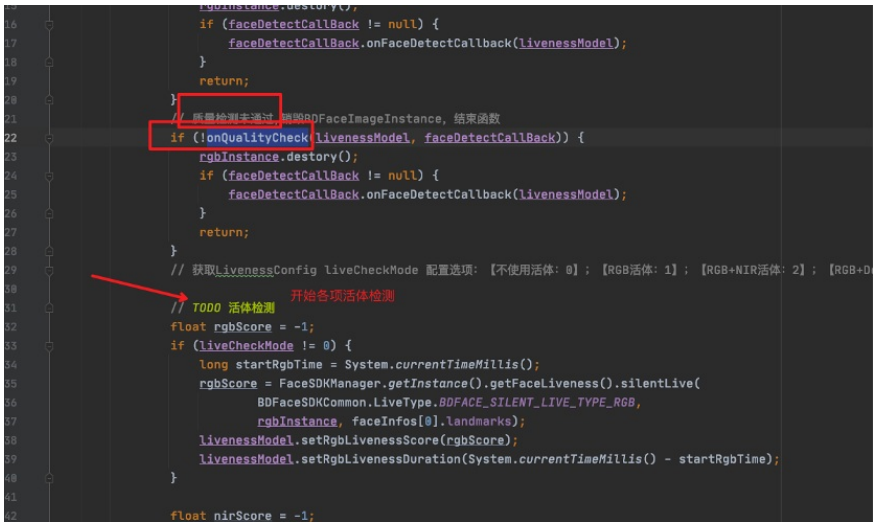
4.进一步获取详细人脸信息

5.最优人脸控制



6.质量检测

完成光照、遮挡、属性部位的质量过滤



7.进行各项活体检测过滤

（1）rgb活体校验

```
// TODO 活体检测
float rgbScore = -1;
if (liveCheckMode != 0) {
    long startRgbTime = System.currentTimeMillis();
    rgbScore = FaceSDKManager.getInstance().getFaceLiveness().silentLive(
            BDFaceSDKCommon.LiveType.BDFACE_SILENT_LIVE_TYPE_RGB,
            rgbInstance, faceInfos[0].landmarks);
    livenessModel.setRgbLivenessScore(rgbScore);
    livenessModel.setRgbLivenessDuration(System.currentTimeMillis() - startRgbTime);
}

float nirScore = -1;
FaceInfo[] faceInfosIr = null;
BDFaceImageInstance nirInstance = null;
if (liveCheckMode == 2 || liveCheckMode == 4 && nirData != null) {
    // 创建检测对象，如果原始数据YUV-IR，转为算法检测的图片BGR
    // TODO: 用户调整旋转角度和是否镜像，手机和开发版需要动态适配
    nirInstance = new BDFaceImageInstance(nirData, srcHeight,
            srcWidth, BDFaceSDKCommon.BDFaceImageType.BDFACE_IMAGE_TYPE_YUV_NV21,
            SingleBaseConfig.getBaseConfig().getDetectDirection(),
            SingleBaseConfig.getBaseConfig().getMirrorNIR());

    // 避免RGB检测关键点在IR对齐活体稳定，增加红外检测
    long startIrDetectTime = System.currentTimeMillis();
    BDFaceDetectListConf bdFaceDetectListConf = new BDFaceDetectListConf();
    bdFaceDetectListConf.usingDetect = true;
    faceInfosIr = faceDetectNir.detect(BDFaceSDKCommon.DetectType.DETECT_NIR,
            BDFaceSDKCommon.AlignType.BDFACE_ALIGN_TYPE_NIR_ACCURATE,
            nirInstance, faceInfos: null, bdFaceDetectListConf);
    bdFaceDetectListConf.usingDetect = false;
    livenessModel.setIrLivenessDuration(System.currentTimeMillis() - startIrDetectTime);
      LogUtils.e(TIME_TAG, "detect ir time = " + livenessModel.getIrLivenessDuration());
```

（2）nir活体校验

```
//              LogUtils.e(TIME_TAG, "detect ir time = " + livenessModel.getIrLivenessDuration());

            if (faceInfosIr != null && faceInfosIr.length > 0) {
                FaceInfo faceInfoIr = faceInfosIr[0];
                long startNirTime = System.currentTimeMillis();
                nirScore = FaceSDKManager.getInstance().getFaceLiveness().silentLive(
                        BDFaceSDKCommon.LiveType.BDFACE_SILENT_LIVE_TYPE_NIR,
                        nirInstance, faceInfoIr.landmarks);
                livenessModel.setIrLivenessScore(nirScore);
                livenessModel.setIrLivenessDuration(System.currentTimeMillis() - startNirTime);
//                LogUtils.e(TIME_TAG, "live ir time = " + livenessModel.getIrLivenessDuration());
            }
        }
```

（3）depth活体校验

```
                    SingleBaseConfig.getBaseConfig().getDepthHeight(),
                    BDFaceSDKCommon.BDFaceImageType.BDFACE_IMAGE_TYPE_DEPTH,
                     angle: 0,  isMbyteArrayror: 0);
        } else {
            depthInstance = new BDFaceImageInstance(depthData,
                    SingleBaseConfig.getBaseConfig().getDepthHeight(),
                    SingleBaseConfig.getBaseConfig().getDepthWidth(),
                    BDFaceSDKCommon.BDFaceImageType.BDFACE_IMAGE_TYPE_DEPTH,
                     angle: 0,  isMbyteArrayror: 0);
        }

        // 创建检测对象，如果原始数据Depth
        long startDepthTime = System.currentTimeMillis();
        if (SingleBaseConfig.getBaseConfig().getCameraType() == 1) {
            depthScore = FaceSDKManager.getInstance().getFaceLiveness().silentLive(
                    BDFaceSDKCommon.LiveType.BDFACE_SILENT_LIVE_TYPE_DEPTH,
                    depthInstance, depthLandmark);
        } else {
            depthScore = FaceSDKManager.getInstance().getFaceLiveness().silentLive(
                    BDFaceSDKCommon.LiveType.BDFACE_SILENT_LIVE_TYPE_DEPTH,
                    depthInstance, faceInfos[0].landmarks);
```

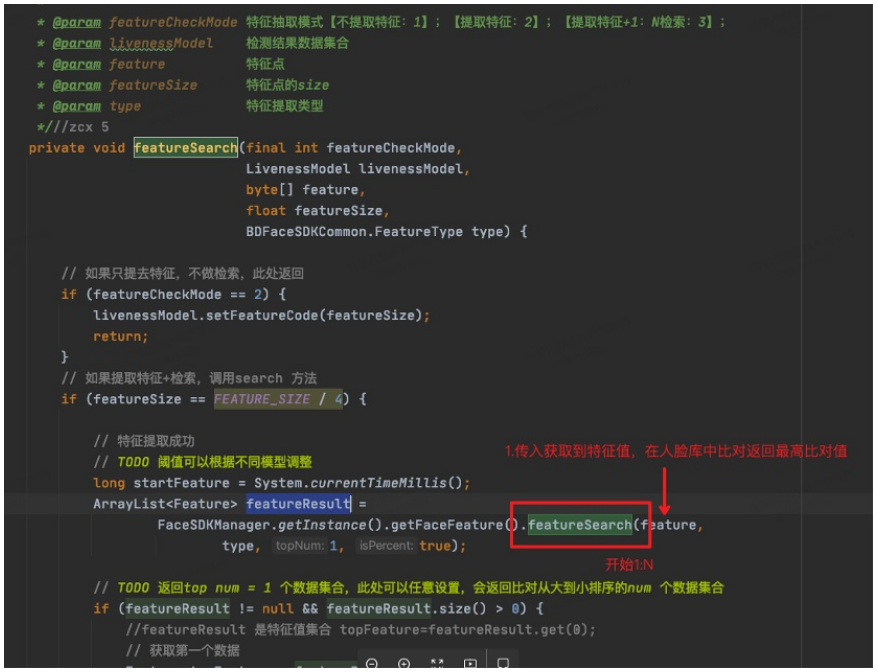
特征提取+人脸检索

```
        depthInstance.destory();
    }

    // TODO 特征提取+人脸检索
    if (liveCheckMode == 0) {
        onFeatureCheck(rgbInstance, faceInfos[0].landmarks, faceInfosIr,
                nirInstance, livenessModel, featureCheckMode,
                SingleBaseConfig.getBaseConfig().getActiveModel());
    } else {
        if (liveCheckMode == 1 && rgbScore > SingleBaseConfig.getBaseConfig().getRgbLiveScore()) {
            onFeatureCheck(rgbInstance, faceInfos[0].landmarks, faceInfosIr,
                    nirInstance, livenessModel, featureCheckMode,
                    SingleBaseConfig.getBaseConfig().getActiveModel());
        } else if (liveCheckMode == 2 && rgbScore > SingleBaseConfig.getBaseConfig().getRgbLiveScore()
                && nirScore > SingleBaseConfig.getBaseConfig().getNirLiveScore()) {
            onFeatureCheck(rgbInstance, faceInfos[0].landmarks, faceInfosIr, nirInstance,
                    livenessModel, featureCheckMode,
                    SingleBaseConfig.getBaseConfig().getActiveModel());
        } else if (liveCheckMode == 3 && rgbScore > SingleBaseConfig.getBaseConfig().getRgbLiveScore()
                && depthScore > SingleBaseConfig.getBaseConfig().getDepthLiveScore()) {
            onFeatureCheck(rgbInstance, faceInfos[0].landmarks, faceInfosIr, nirInstance,
                    livenessModel, featureCheckMode,
                    SingleBaseConfig.getBaseConfig().getActiveModel());
        } else if (liveCheckMode == 4 && rgbScore > SingleBaseConfig.getBaseConfig().getRgbLiveScore()
                && nirScore > SingleBaseConfig.getBaseConfig().getNirLiveScore()
                && depthScore > SingleBaseConfig.getBaseConfig().getDepthLiveScore()) {
            onFeatureCheck(rgbInstance, faceInfos[0].landmarks, faceInfosIr, nirInstance,
                    livenessModel, featureCheckMode,
                    SingleBaseConfig.getBaseConfig().getActiveModel());
```

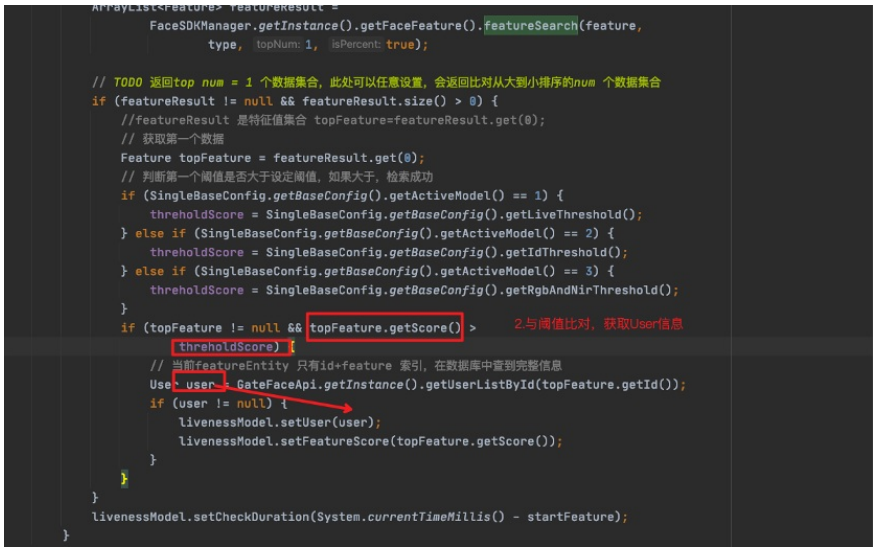
重点在这里 color="#0000dd">landmarks —> feature （人脸关键点转人脸特征值）

8.提取特征值

9.featureSearch人脸检索1:N



搜索结果与设定阈值比较



若比对通过返回人脸信息

综合以上步骤，人脸搜索1:N全流程 ：

init资源——暗光恢复——.track绘制人脸框——.detect进一步获取详细人脸信息——最优人脸控制——质量检测——.silentlive活体检测（rgb活体校验、nir活体校验、depth活体校验）——.feature提取特征值——.search人脸搜索1：N返回结果。

7. 人脸识别SDK去除本地时间校验解决方案

问题：

在控制台下载并使用人脸SDK后，根据在线激活、离线激活、批量激活方式完成授权后，在后续使用过程中报错： license use times has reached the upper limit问题

解决方案：

（该方案是去除本地时间校验和在线激活每次均需联网的方案，间接性地解决了激活次数上限问题）

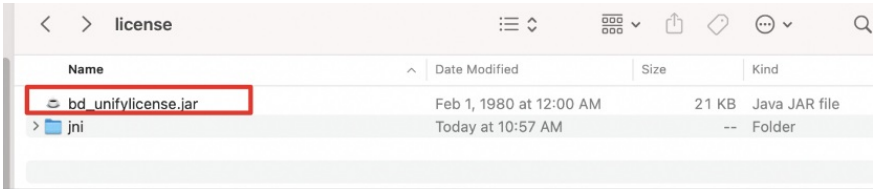
根据链接下载license文件，下载链接如下：

链接: https://pan.baidu.com/s/1PUOXKuH8693kmUmDSs4Jeg 提取码: v51d

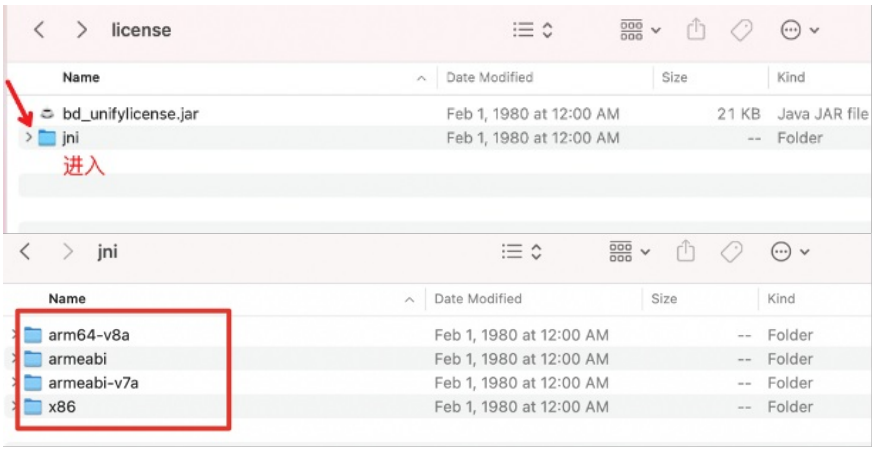
解压后关注两处文件：

（1）jar包

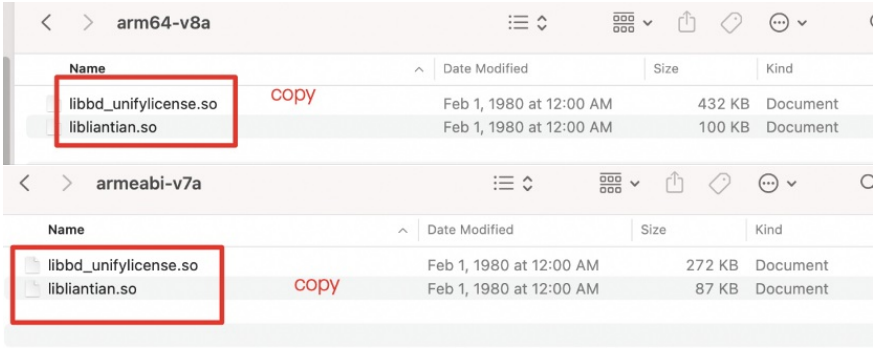
进行拷贝



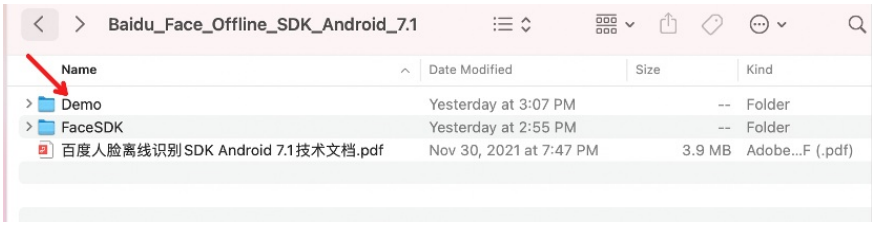
（2）so库



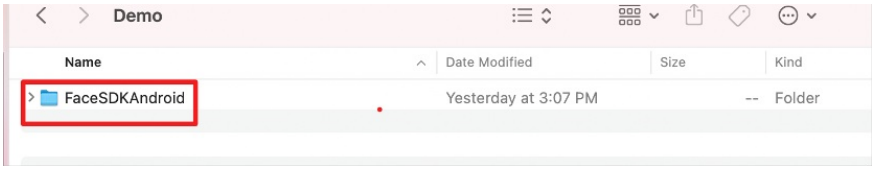
分别进入，拷贝



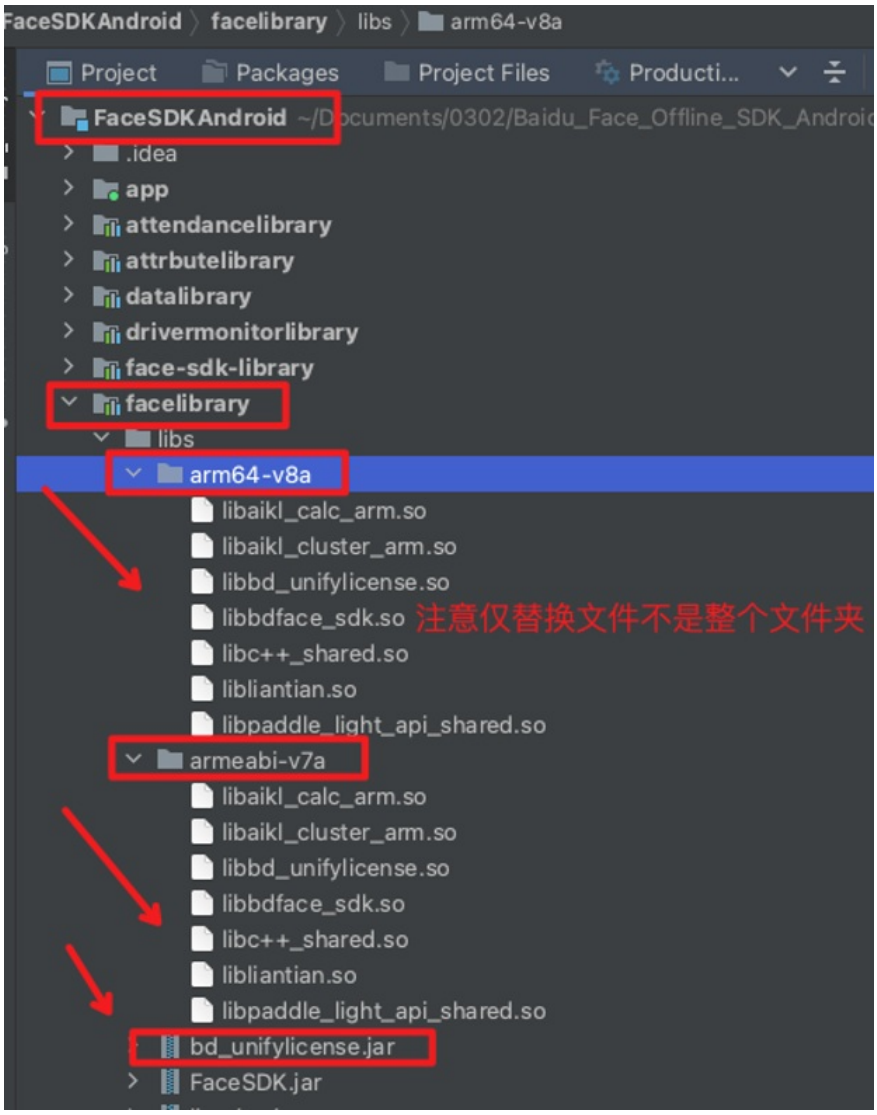
拷贝so文件后，进入使用的SDK 工程路径



再进入



将以上文件替换到如下文件中（注意仅替换文件，不是整体文件夹）



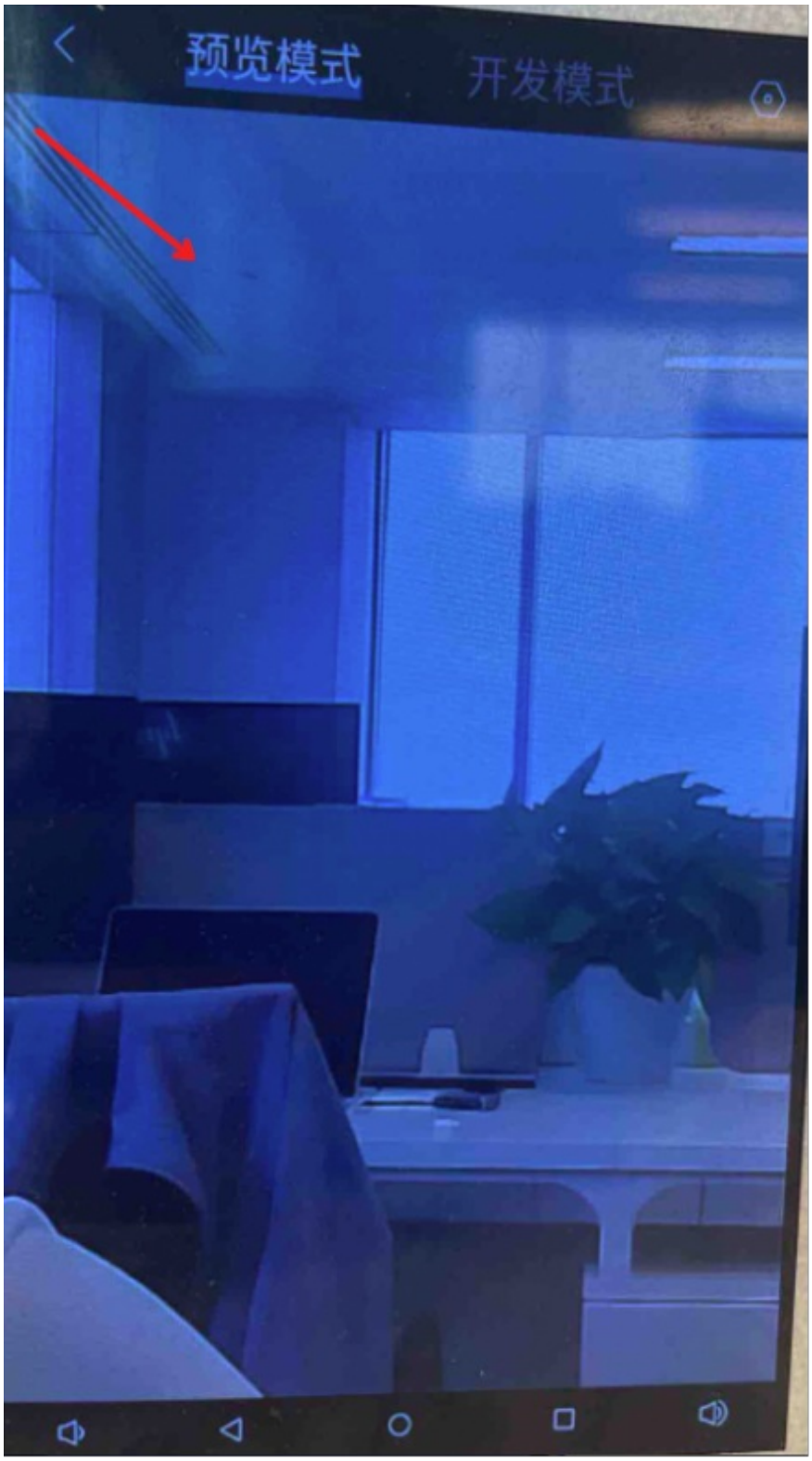
替换后在AS上方，点击Build-Clean Project

执行成功后，编译APK推进设备安装，该APK即可完成在激活后，去除本地时间校验报错license use times has reached the upper limit问题。

8. 识别SDK视频流屏蔽与控制的解决方案

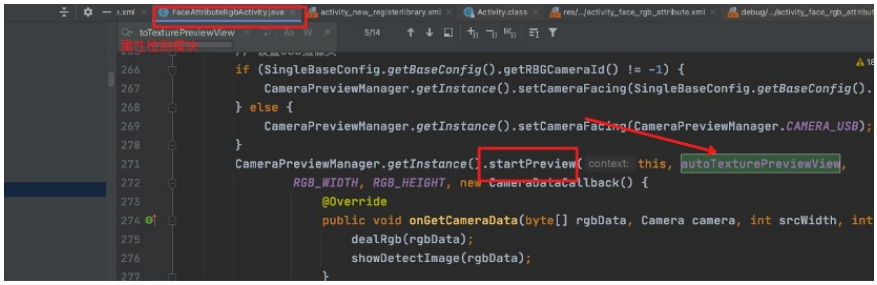
问题：

屏蔽视频流、控制SDK视频流大小，保证功能运行+接入三方业务UI



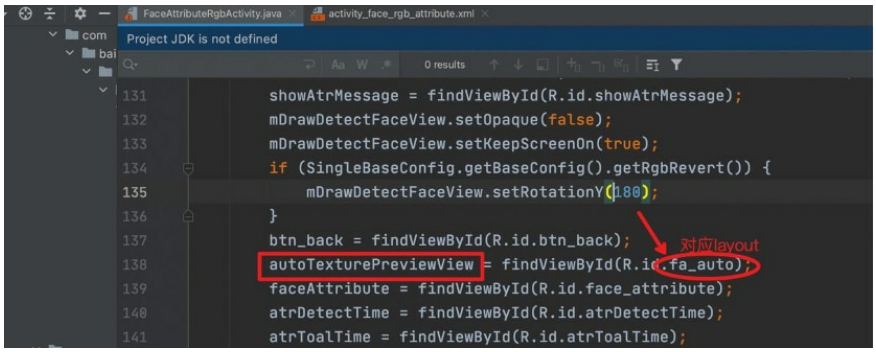
解决方案：

在具体模块（指如闸机模块、考勤模块）中找到starpreview函数执行，这里以属性检测模块中代码为例进行分析，如图

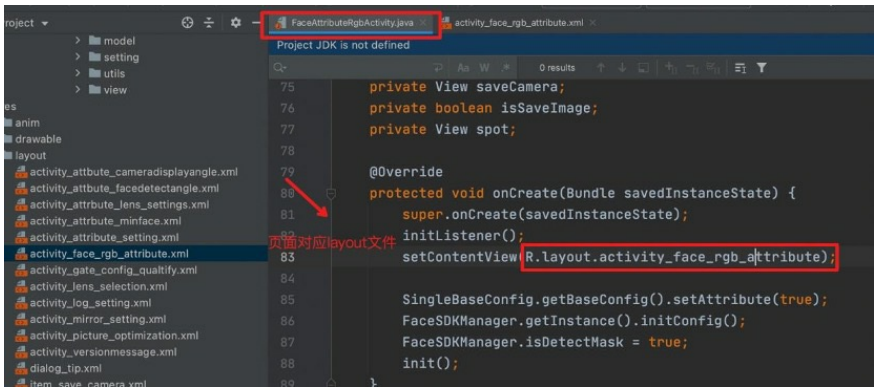


startPreview是SDK开启摄像头的逻辑，视频流真正开启时，对应的UI控制和显示就在这里

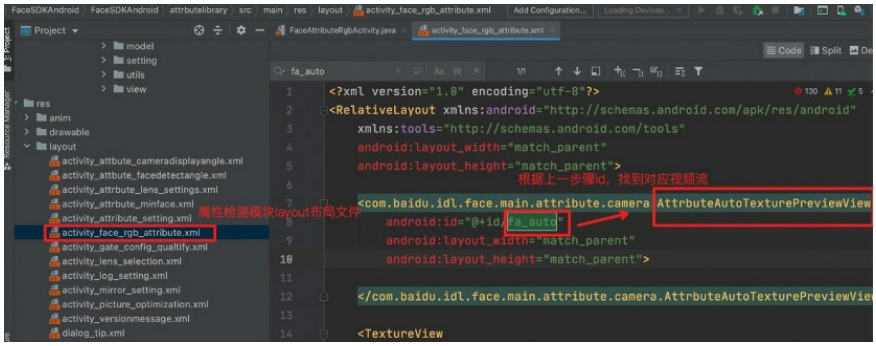
在该Activity中找到 autoTexturePreviewView对应layout .fa_auto



找到该Activity对应layout文件



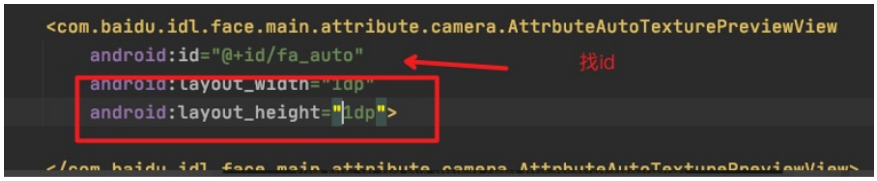
进入页面布局文件中，根据id找到对应视频流UI com.baidu.idl.face.main.attribute.camera.AttrbuteAutoTexturePreviewView



1、解决屏蔽SDK视频流显示：

在这里修改宽、高就可以了，width、height修改为1dp，视频流完成屏蔽不会再显示。

（除可以修改宽高，也可自己写入其他view进行覆盖）

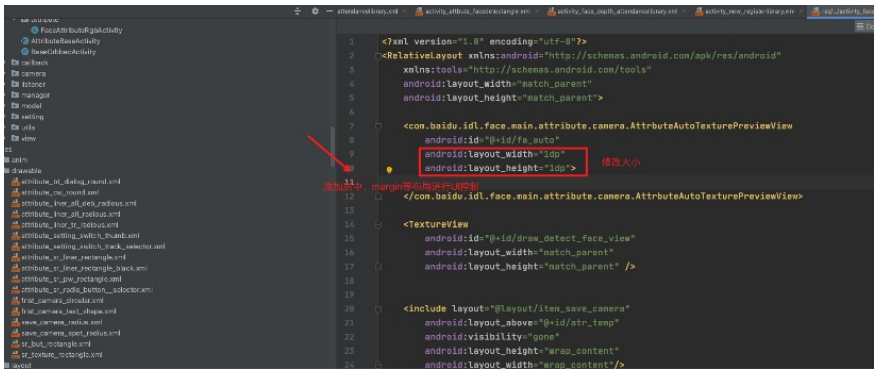


或者透明度修改为0也可实现

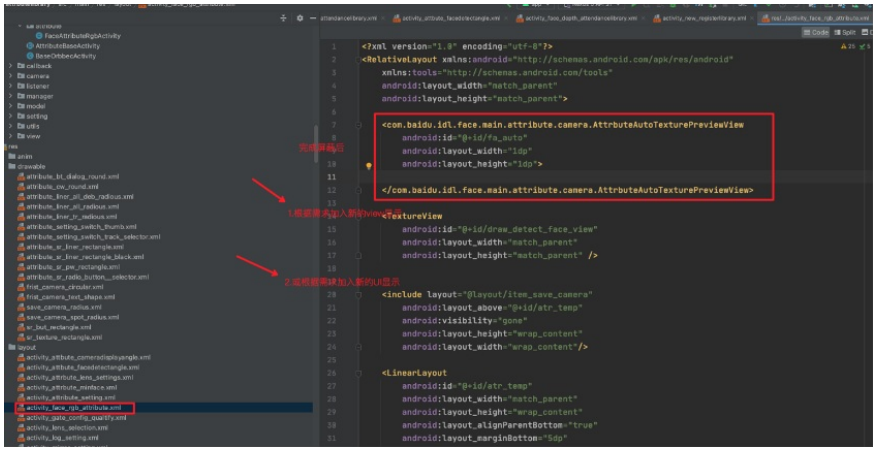
2、控制SDK视频流大小

这里如果将width和height修改为91dp、99dp（ 具体宽高大小根据业务需求进行更改即可 ），可对画面中视频流的大小进行控制

此外， 在这里也可加入居中等其他排版完成对视频大小和方位的控制。



覆盖或屏蔽视频流后在在该布局文件中根据实际需求添加新的UI，或引入自行的view进行显示即可。



同时也可以使用以下方式实现：自定义设定宽高

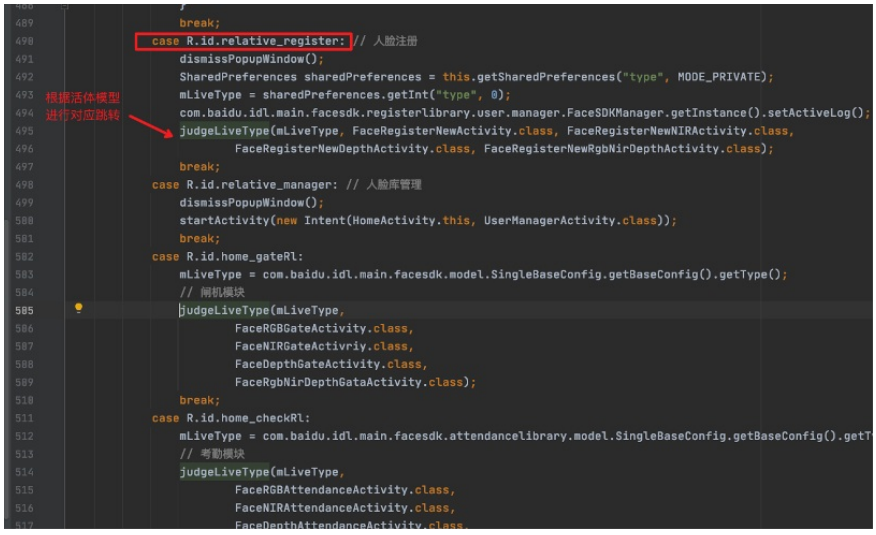
```
        }
        ViewGroup.LayoutParams layoutParams = irPreviewView.getLayoutParams();
        int w = layoutParams.width;
        int h = layoutParams.height;
        int cameraRotation = SingleBaseConfig.getBaseConfig().getNirVideoDirection();
        mCamera[1].setDisplayOrientation(cameraRotation);
        if (cameraRotation == 90 || cameraRotation == 270) {
            layoutParams.height = w;
            layoutParams.width = h;
            // 旋转90度或者270，需要调整宽高
        } else {
                layoutParams.height = h;
                layoutParams.width = w;
        }
        irPreviewView.setLayoutParams(layoutParams);
        mPreview[1].setCamera(mCamera[1],  PREFER_WIDTH , PERFER_HEIGH);
        mCamera[1].setPreviewCallback(new Camera.PreviewCallback() {
```

9. 人脸注册添加自定义质量检测解决方案

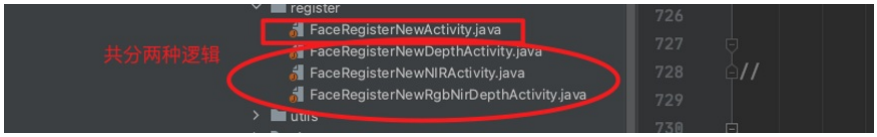
解决：人脸注册时结合用户三方业务需求，添加自定义逻辑方案 人脸注册模块代码对应位置如下：

根据活体模型执行对应跳转：

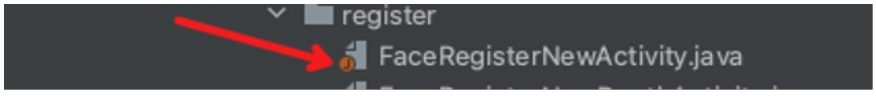
不使用活体、RGB活体、RGB+NIR活体、RGB+DEPTH活体、RGB+NIR+DEPTH活体



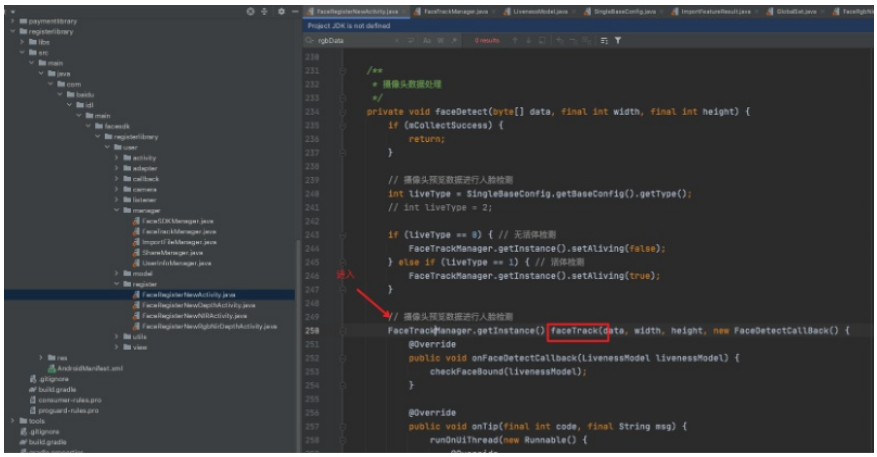
虽然有4种方式，但根据跳转的进入方式，只分两种逻辑



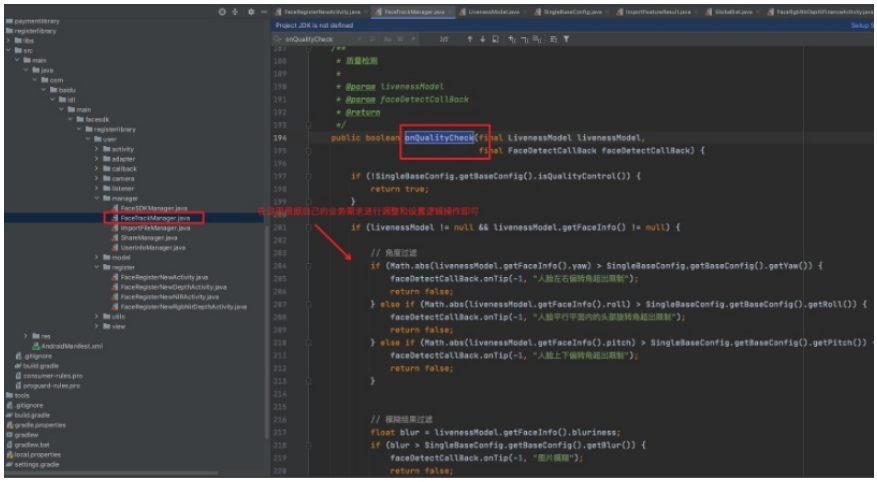
1、若进入的是RGB单目活体方式



进入FaceTrackManager



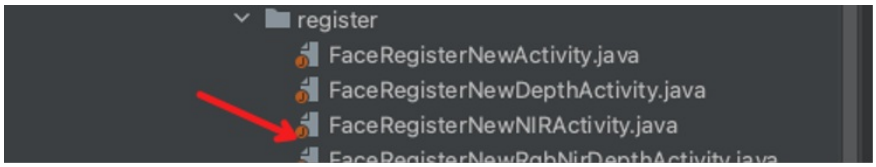
找到实际进行质量检测的逻辑onQualityCheck



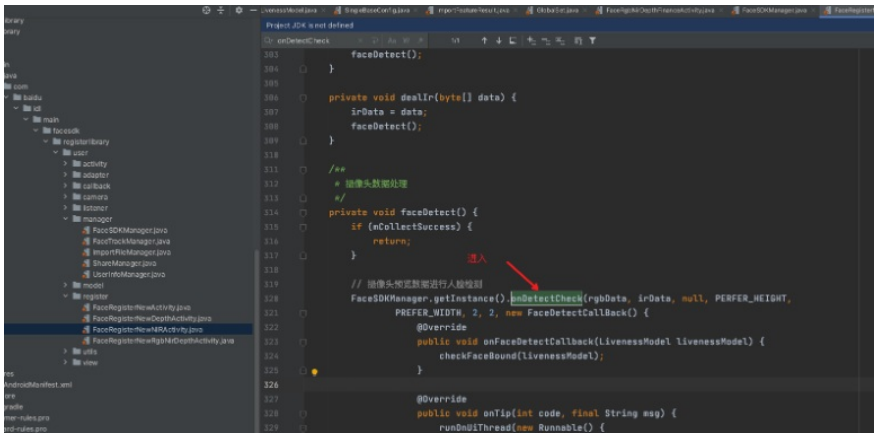
结合用户三方业务需求，添加自定义逻辑方案，

如左眼达到遮挡条件时进行提示、不再进行注册、或其他自行需求逻辑执行

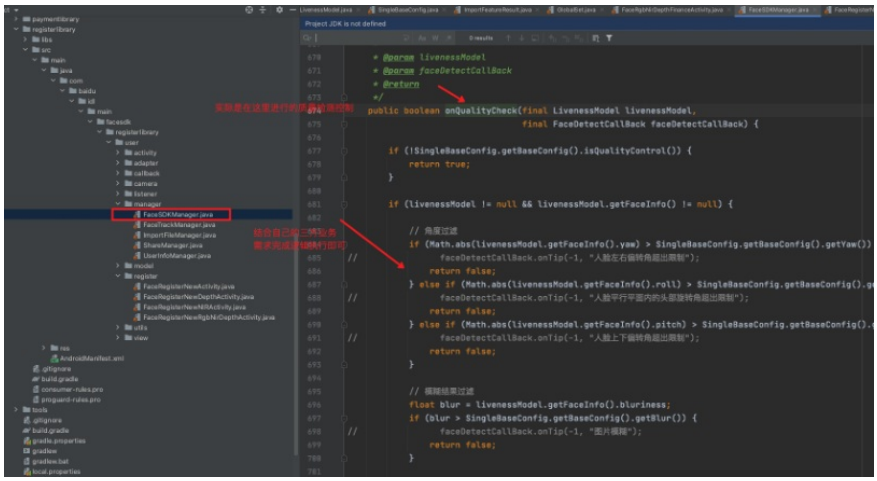
2、若进入的是除RGB单目方式之外的逻辑



进入对应逻辑FaceSDKManager onDetectCheck

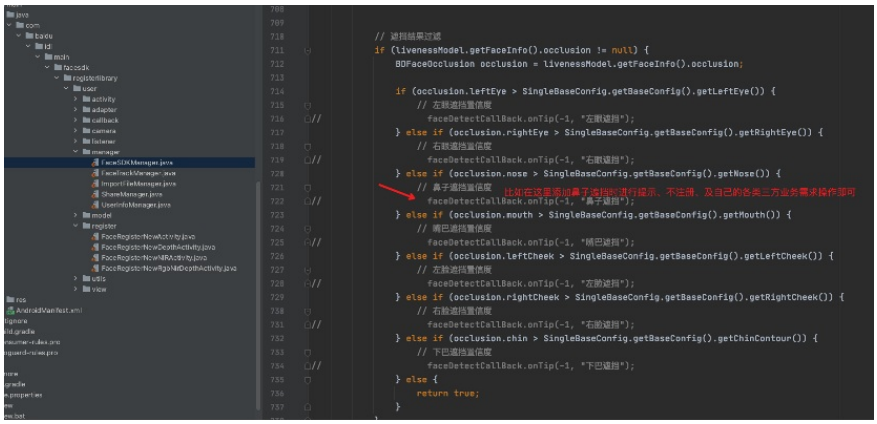


找到实际进行质量检测的逻辑onQualityCheck



结合用户三方业务需求，添加自定义逻辑方案，

比如鼻子遮挡后进行提示、不再进行注册、或其他自行需求逻辑执行



10. 人脸识别 Android SDK 批量激活去除重复激活解决方案

问题：

人脸识别Android SDK，若您使用的批量激活方式激活，在授权后，重启设备会重新进入激活授权界面，需要您点击批量激活重新完成激活授权过程，通过以下方案您可去除该重复操作步骤，仅需一次批量激活流程。

解决方案：

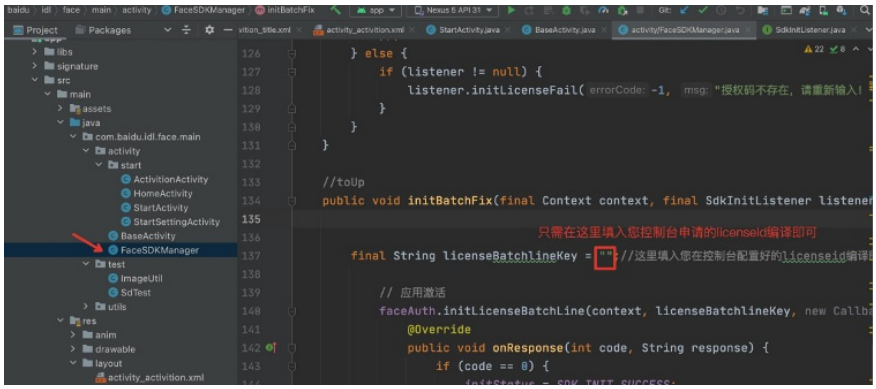
（由于该修改过程较为繁琐，因此我们已经为您修改代码，将修复后的工程上传到云端，您可下载直接使用）

您请先在这里下载修复工程包：

https://ai-toolbox.bj.bcebos.com/AndroidTool/BatchUPFaceSDKAndroid.zip?authorization=bce-auth-v1/2a5115b018924a7bb2405efb5392855f/2022-12-05T08%3A36%3A46Z/-1/host/8227aea41483654bd2fcf569d1e4ef9dfa42b30e81d479f862a0d0c06f28369b

下载解压后。打开找到以下路径文件，如图：

app/src/main/java/com.baidu.idl.face.main/activity/FaceSDKManager



找到图中标注 licenseBatchlineKey 赋值处，在该处填写控制台配置的批量激活licenseId，如图：

将图中您申请的License Id填入以上licenseBatchlineKey中，点击 build 重新编译生成 apk，即可安装使用。该版本 apk 可在首次完成激活授权后，在退出、重启设备等操作下无需再次激活流程。

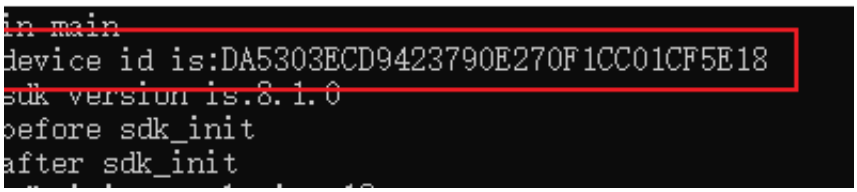
Windows-SDK-常见问题

1. 激活说明

1.1 获取硬件指纹

- LicenseTool工具获取：在SDK解压目录~\tools\license_tool\ 双击LicenseTool.exe，可以看到设备的硬件指纹

- 运行SDK获取：使用vs打开工程，选择Relase x64运行，同样可以得到指纹信息，如下 device_id即为指纹信息

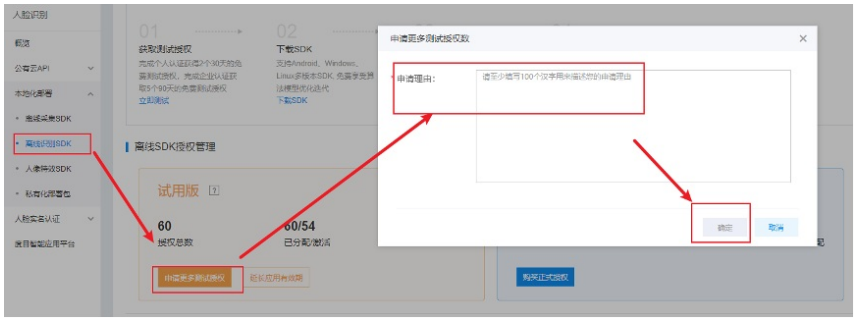


1.2 获取序列号

申请授权：

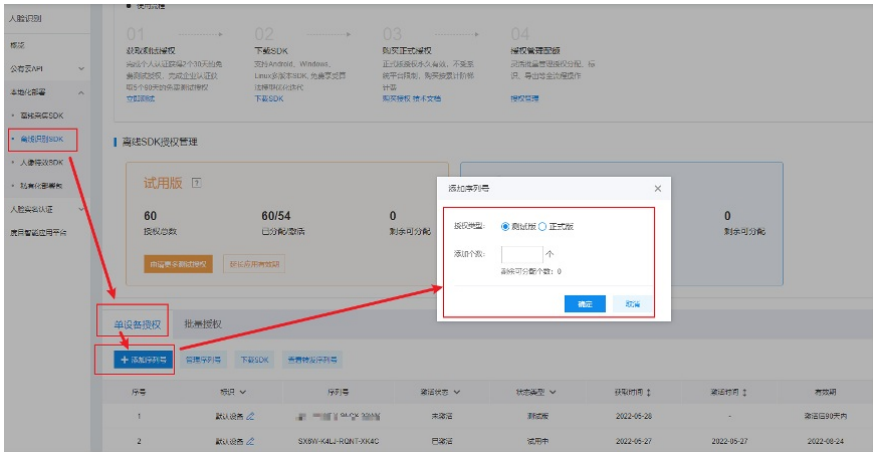
- 申请链接：https://console.bce.baidu.com/ai/?fromai=1#/ai/face/offline/index
- 查看剩余：

- 申请更多测试授权：依次选择： 离线识别SDK ->申请更多测试授权 -> 填写申请理由即可 -> 确定 ，预计24小时通过



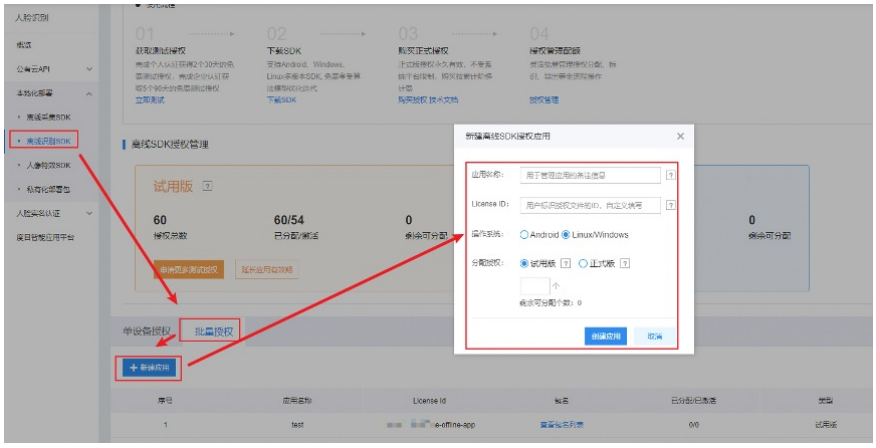
单台设备授权序列号：

依次点击： 单设备授权 -> 添加序列号 -> 测试版 -> 添加个数填写1 -> 确定



页面稍后自动刷新，即可看到生成的16位序列号，以及激活状态和状态类型

** **批量授权应用：**** 依次点击：批量授权 -> 新建应用 -> 填写应用名称（自定义）-> 填写License ID （自定义，需要英文）-> 选择操作系统（Linux/Windows）-> 试用版 -> 添加个数填写2 -> 创建应用



页面稍后自动刷新，即可看到自定义的license id，如下截图

1.3 激活

在线激活 单设备激活

- 方法1：在SDK解压目录~\tools\license_tool\ 双击LicenseTool.exe，填写单台设备授权的16位序列号，点击激活，会提示激活成功，同时会自动生成license.key 和 license.ini文件，在控制台的单台设备授权中的序列号状态为已激活。
- 方法2：在SDK解压目录~\license\打开license.key文件，删除里面内容，填写16位序列号，然后运行工程即可。运行工程时会自动拉取license.ini文件。

批量激活

在SDK解压目录~\license\打开license.key文件，删除里面内容，填写在 批量授权应用中自定义的license id

在设备运行sdk时，可以自动拉取license.ini文件，对应的授权应用中的数量减1，直到用完为止

注意：license文件内容（license.key和license.ini）需要放置~\license\目录下

离线激活 在获取硬件指纹后，可在控制台中的序列号列表右侧点击离线激活，按照操作，输入硬件指纹（ 尽可能复制粘贴，手动输入容易出错 ），绑定之后，可下载授权文件

将授权文件解压后替换sdk 中的license文件夹即可

注意：license文件内容（license.key和license.ini）需要放置~\license\目录下

2. 授权失败相关问题

2.1 常见错误码

错误码	错误描述	错误中文描述	处理办法
-4	MODEL_IS_EMPTY	模型内容为空	1、在二次开发过程中，models文件夹和license文件夹与程序运行目录的相对路径需要跟SDK demo保持同样的相对路径 2、如果自行调整路径，则models文件夹和license文件夹要在同一级目录，如均在D://face目录下；②sdk_init函数传参应该为D://face
-13	NOT_AUTHORIZED	未授权	1、可以使用licensetool.exe进行激活（需要联网） 2、可以登录控制台控制台进行激活 3、可以直接修改license.key文件，将内容替换成要激活的序列号（需要联网）
-1015	LICENSE_FILE_NOT_EXIST	授权文件不存在	sdk_init传入的路径下没有license文件夹
-1016	LICENSE_KEY_EMPTY	授权序列号为空	检查license.key文件是否为空
-1017	LICENSE_KEY_INVALID	授权序列号无效	检查license.key文件是否与控制台的序列号格式一致
-1018	LICENSE_KEY_EXPIRE	授权序序列号过期	检查控制台授权序列号是否是测试序列号，测试序列号的授权是有时效的，可以在控制台查看序列号的失效时间，也有可能已被其他设备使用
-1019	LICENSE_ALREADY_USED	授权序列号已被使用	该序列号已在其它设备上激活，检查当前设备的device_id是否与控制台对应序列号的硬件指纹信息一致，也有可能出现序列号过期
-1020	DEVICE_ID_EMPTY	设备指纹为空	远程激活时，输入指纹ID为空，一般不会出现该错误
-1021	NETWORK_TIMEOUT	网络超时	远程激活时，网络超时，可稍后再试，一般不会出现该错误
-1022	NETWORK_ERROR	网络错误	远程激活时，设备没有网络，联网后进行激活，如果仍然失败，可在控制台进行离线激活，下载对应的license文件

2.2 通用排查方法

- 检查模型文件和license文件是否在同一级目录下，如果自行调整位置，sdk_init需要修改路径，例：均在D://face目录下，则sdk_init函数传参应该为D://face

```
//初始化sdk
std::cout << "before sdk_init" << std::endl;
// model_path为模型文件夹路径，即models文件夹（里面存的是人脸识别的模型文件）
// 传空为采用默认路径，若想定置化路径，请填写全局路径如：d:\\face （models模型文件夹目录放置后为d:\\face\\models）
int res = api->sdk_init(nullptr);
std::cout << "after sdk_init" << std::endl;
if (res != 0)
{
    std::cout << "sdk init result is:" << res << std::endl;
    getchar();
    return 0;
}
```

- 检查license文件夹下的文件是否被替换成已激活的license文件，如果已经替换，查看控制台输出的license.key是否与激活的序列号一致
- 检查设备的指纹信息与控制台激活的序列号中的指纹信息是否一致

- 网络超时或者网络错误，需要检查网络问题
- 设备重启之后，时间可能被修改，需要调整为当前时间（当前北京时间）

2.3 指纹信息变更原因

硬件指纹是指通过算法，提取到设备的CPU，GPU，硬盘，内存条，网卡等信息，加密成唯一字符串，因此在使用过程中不要更换CPU，GPU，硬盘，内存条，网卡等硬件设备，更换会导致指纹变更，重装系统也会导致指纹变更。常见的指纹变更原因如下：

- 扩容硬盘后，出现指纹变化
- 使用虚拟机，重启之后发生了指纹变化
- 重装系统后，导致指纹信息变化
- 系统时间变更，导致授权失败
- 个别工控机设备重启之后，或者外接硬件插拔（如继电器），导致指纹信息发生变化
- 个别网络环境切换时，导致指纹发生变化
- 个别设备外接U盘，指纹信息发生变化
- 个别设备存在多个网卡，可能会导致指纹信息发生变化

3. 运行问题

3.1 运行环境

- 支持Win7，Win10系统，其他系统可能存在兼容性问题，推荐使用x64运行
- Windows Server系统上可能缺失一些动态库，不建议使用
- 不建议使用虚拟机，虚拟机重启可能会导致指纹发生变化

3.2 DLLNotFound异常

此DllNotFound异常可能是系统环境必要的DLL缺失导致，也可能是此设备的CPU没有AVX指令集所导致(编译运行离线SDK需要AVX指令进行协助的，可以下载CPU-Z工具查询是否有AVX指令集)，需要从这两方面排查一下:

1. 安装依赖

- 可以先安装一下基础环境的动态链接库DLL包，可以去微软官网下载vc_redist或者提工单支持
- 离线SDK中~/tools/vc_redist，根据开发设备的选择安装vc_redist
- 安装vc2013和vc2015

2. 安装CPU-Z

- 下载链接：https://www.cpuid.com/softwares/cpu-z.html
- 检查是否缺少AVX指令集，如果没有AVX指令集，则可以在控制台下载NOAVX版本的SDK

3.3 内存占用较高

SDK demo在运行时，加载全部模型，大约会占用800M左右的内存

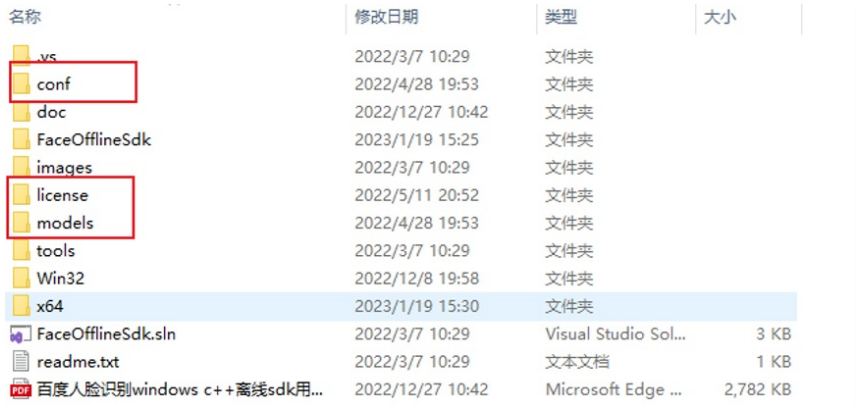
SDK在使用过程中（如持续循环调用人脸检测）会出现短暂的内存增加，随后内存会保持稳定，不再增加

4. 二次开发

4.1 必要文件

db文件无需复制，sdk初始化时会自动生成，license、models、conf文件夹要放到同一级目录中，dll文件放到根目录或者编译生成的exe目录

- 授权文件：license文件夹
- 模型文件：models文件夹
- 配置文件：conf文件夹
- dll库文件：SDK运行目录下所有的dll文件
  以c++ SDK为例，如下图



4.2 输出日志配置

在exe运行目录下创建配置文件 face_conf.json，且其中写入 json 字段 {“log_open”: 1}为console 日志输出模式，若字段值为 2，则可输出日志文件 face_sdk.log（同 exe 所在目录）。通过日志输出可更详细判断各接口返回的错误码及问题所在。若该 conf 配置文件不存在或字段值为 1 或 2 之外的其他字段，则不输出日志。

4.3 模型缩减

sdk 支持按需配置模型和加载能力，若 sdk 有某部分功能不需要使用，可尝试删除一些模型，删除后模型既不会加载也不会占用内存。sdk 中 models 文件夹里的模型及是否可删减说明如下表:

模型文件夹名称	说明	是否可删减	删减说明
detect	人脸检测模型	是	若不使用nir近红外功能，可删除detect_nir开头的模型文件
align	人脸关键点模型	是	若不使用nir近红外功能，可删除align_nir开头的模型文件
attribute	人脸属性	是	若不使用人脸属性检测功能，该文件夹可删除
best_image	最佳人脸	是	若不使用最佳人脸检测功能，该文件夹可删除
blur	人脸质量模糊度检测	是	若不使用该功能，该文件夹可删除
dark_enhance	暗光恢复	是	若不使用该功能，该文件夹可删除
eye_close	眼睛闭合	是	若不使用眼睛闭合及动作活体功能，该文件夹可删除
feature	人脸特征值	是	若仅使用rgb可见光人脸特征值，feature_nir文件可删除（nir特征值）
gaze	人脸注意力检测	是	若不使用该功能，该文件夹可删除
mouth_close	嘴巴闭合检测	是	若不使用嘴巴闭合及动作活体功能，该文件夹可删除
mouth_mask	口罩佩戴检测	是	若不使用该功能，该文件夹可删除
occlusion	人脸遮挡检测	是	若不使用该功能，该文件夹可删除
silent_live	静默活体检测	是	若只使用rgb可见光单目静默活体，则liveniss_nir和live_depth文件可删除

4.4 阈值配置

sdk 支持能力自定义、通过读取配置文件的方式进行能力定制化。sdk 默认能力加载无需定制化，若需要定制化，请把 sdk 根目录里面的 conf 文件夹重命名为 config 文件夹。并且在 sdk 中，把里面的 json 配置按如下说明做修改，可起到定制化能力加载的效果，配置文件简要说明如下（若需定制化修改，请参考示例 json，修改 json 字段的值来达到定制化的目的）

- detect.json (人脸检测配置)

配置文件名	detect.json	

说明	人脸检测自定义能力配置文件
	{
	"max_detect_num":5,
	"min_face_size":0,
	"scale_ratio":-1,
	"not_face_thr":0.5
	}	

参数	类型	说明
max_detect_num	int	最大检测的人脸数量，最多支持 50，最小 1,默认 1
min_face_size	int	默认 0，可用来设置检测的最小人脸，比如可设为 10，则表示小于 10*10 的人脸，sdk 检测不到
scale_radio	int	默认-1，表示进行人脸检测时候的图片缩放比率。建议用-1 表示传入原图 sdk 自己缩放（为保证检测效果，该参数建议用默认）
not_face_thr	float	默认 0.5，表示非人脸的阈值，取值范围 0-1

- track.json (人脸追踪配置)

配置文件名	track.json	

说明	人脸追踪自定义能力配置文件
	{
	"detect_intv_before_track":0.02,
	"detect_intv_during_track":0.02
	}	

参数	类型	说明
detect_intv_before_track	float	表示人脸追踪前进行人脸检测的时间间隔（单位毫秒）
detect_intv_during_track	float	表示人脸追踪时候进行人脸检测的时间间隔（单位毫秒）

- action_live.json (动作活体配置)

配置文件名	action_live.json	

说明	人脸动作活体自定义能力配置文件
	{
	"eye_open_threshold":0.5,
	"eye_close_threshold":0.5,
	"mouth_open_threshold":0.5,
	"mouth_close_threshold":0.5,
	"look_up_threshold":0.5,
	"look_down_threshold":0.5,
	"turn_left_threshold":0.5,
	"turn_right_threshold":0.5,
	"nod_threshold":0.5,
	"shake_threshold":0.5,
	"max_cache_num":1
	}	

参数	类型	说明
eye_open_threshold	float	表示人脸动作活体眼睛睁开的置信度阈值
eye_close_threshold	float	表示人脸动作活体眼睛闭合的置信度阈值
mouth_open_threshold	float	表示人脸动作活体嘴巴张开的置信度阈值
mouth_close_threshold	float	表示人脸动作活体嘴巴闭合的置信度阈值
look_up_threshold	float	表示人脸动作活体抬头的置信度阈值
look_down_threshold	float	表示人脸动作活体低头的置信度阈值
turn_left_threshold	float	表示人脸动作活体向左转头的置信度阈值
turn_right_threshold	float	表示人脸动作活体向右转头的置信度阈值
nod_threshold	float	表示人脸动作活体点头的置信度阈值
shake_threshold	float	表示人脸动作活体摇头的置信度阈值
max_cache_num	float	最大缓存数、推荐为 1 不做修改

- crop.json (人脸抠图配置)

配置文件名	crop.json	

说明	人脸抠图自定义能力配置文件
	{
	"is_flat":0,
	"crop_size":200,
	"enlarge_ratio":1
	}	

参数	类型	说明
is_flat	int	默认为 0，表示是否是镜像，该参数目前无效
crop_size	int	表示抠图的大小，如 200，则表示抠出来的是 200*200 的图片
enlarge_ratio	float	默认是 1，若小于 1，比如 0.8，图片有点放大，会稍微模糊，建议自定义可设为 1

4.5 视频流卡顿

如果视频流看起来卡顿，是由于在循环读取视频帧时，进行了如人脸检测、特征值提取等接口调用。一般这种场景，是需要抽帧处理的，如每2秒进行一次人脸检测或特征值提取

4.6 多人脸图片检测

- 默认检测人脸数为 1
- 若需要检测多人脸，可修改配置文件，将conf文件修改为config，再修改detect.json，修改对应的max_detect_num参数

```
{
    "max_detect_num":10,
    "min_face_size":0,
    "scale_ratio":-1,
    "not_face_thr":0.5
}
```

4.7 多人脸检测返回顺序

多人脸检测修改好配置后，人脸顺序是按照从大到小的顺序返回，包括人脸检测，人脸抠图等接口。如下图返回的Width和height可以看出，是从大到小的顺序返回

```
after sdk_init
-----face_num  is-------2
detect score is:0.999996
detect mWidth is:246.241
face height is:316.976
detect mCenter_x is:332.83
detect mCenter_y is:371.251
detect score is:0.999986
detect mWidth is:224.529
face height is:293.194
detect mCenter_x is:928.3
detect mCenter_y is:386.735
time  cost is :1407ms
before delete api
end main
```

5. 效果问题

5.1 外国人群人脸识别

目前最新的8.0版本已经针对外国人群（黑人、白人、阿拉伯人、东南亚人等）做了专项泛化性优化，在综合评测集下识别准

确率达到97.5%。

5.2 多年龄人群人脸识别

目前已经针对老人（社区、养老院场景）、小孩（包括幼儿园、K12人群）多年龄人群做了专项泛化性优化。

5.3 NIR人脸识别

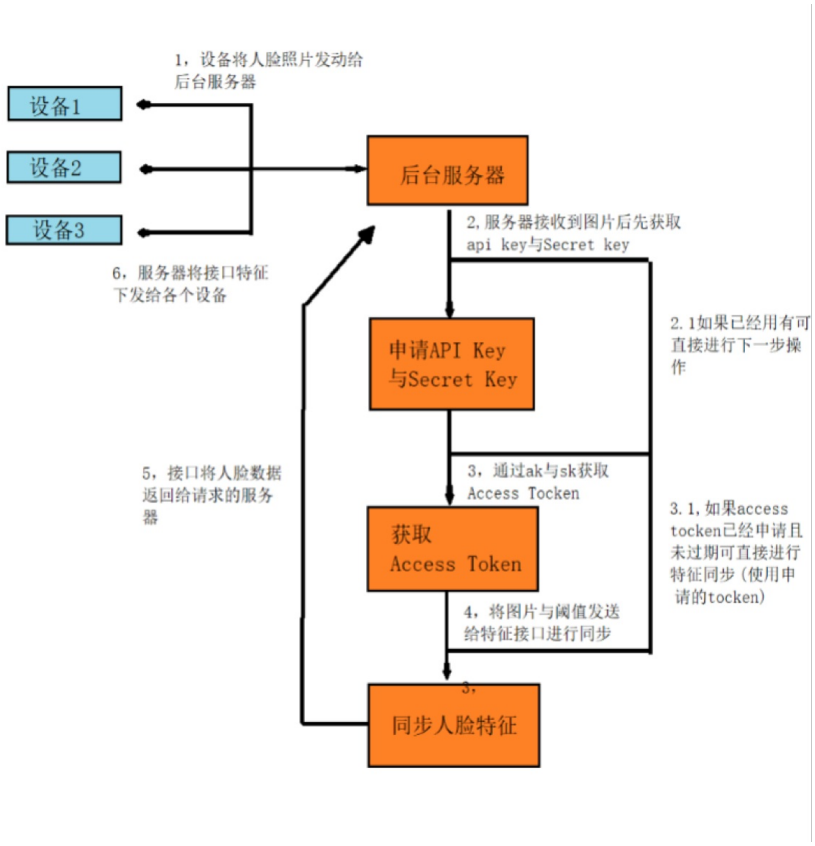
纯nir识别技术暂时不支持，目前支持RGB单模态识别、RGB/NIR混合模态识别；NIR模态在活体检测比较成熟，针对视频流/图片，利用近红外成像原理，实现夜间或无自然光条件下的活体判断。其成像特点（如屏幕无法成像，不同材质反射率不同等）可以实现高鲁棒性的活体判断，降低照片/视频攻击的成功率。

5.4 端云同步

为了解决客户需要在每台设备上需重复建设本地化人脸特征库的问题，提供端云同步工具，支持公有云/私有化部署两种形式；
支持在云端统一抽取特征值，通过access token直接下发特征值数据给到设备端，省去图片传输成本及设备端算力消耗，同时保持了端云模型的精度对齐，整体应用更高效、更安全、更省流量

- 特征值同步公有云接口：
  - 接口文档：https://ai.baidu.com/ai-doc/FACE/Okg7edktq
  - 测试地址：https://console.bce.baidu.com/ai/#/ai/face/app/create（有免费QPS额度可用于测试）
- 特征值私有化部署：
  - 前期配置信息参考（硬件要求/环境要求/单机部署）
    https://ai.baidu.com/ai-doc/FACE/nk38rh8p2 硬件要求
    https://ai.baidu.com/ai-doc/FACE/lk37c1is5 环境要求
    https://ai.baidu.com/ai-doc/FACE/tk4m61y1q 单机部署

业务流程参考如下：



历史版本

API文档-v2

人脸检测

人脸识别接口分为V2和V3两个版本，本文档为V2版本接口的说明文档，请确认您在百度云后台获得的是V2版本接口权限，再来阅读本文档。

> 辨别接口版本的方法是：在百度云后台进入【应用列表】，点击【应用名称】，在【API列表】中可以看到【请求地址】，若请求地址中带有【v2】标识，则您具有的是v2权限，可以阅读本文档；若请求地址中带有【v3】标识，则您具有的是v3权限，应该去阅读v3文档。

如果您对文档内容有任何疑问，可以通过以下几种方式联系我们：

- 在百度云控制台内 提交工单，咨询问题类型请选择**人工智能服务**；
- 如有需要讨论的疑问，欢迎进入 AI社区 与其他开发者们一同交流。

能力介绍

接口能力

- **人脸检测**：检测图片中的人脸并标记出位置信息;
- **人脸关键点**：展示人脸的核心关键点信息，及72个关键点信息。
- **人脸属性值**：展示人脸属性信息，如年龄、性别等。
- **人脸质量信息**：返回人脸各部分的遮挡、光照、模糊、完整度、置信度等信息。

业务应用

典型应用场景：如**人脸属性分析**，**基于人脸关键点的加工分析**，**人脸营销活动**等。

> 说明：检测响应速度，与图片中人脸数量相关，人脸数量较多时响应时间会有些许延长。

质量检测

如果需要判断一张图片中的人脸，是否符合后续识别或者对比的条件，可以使用此接口，在请求时在face_fields参数中请求qualities。基于返回结果qualities中，以下字段及对应阈值，进行质量检测的判断，以保证人脸质量符合后续业务操作要求。

指标	字段与解释	推荐数值界限
遮挡范围	**occlusion**，取值范围[0~1]，0为无遮挡，1是完全遮挡 含有多个具体子字段，表示脸部多个部位 通常用作判断头发、墨镜、口罩等遮挡	left_eye：0.6, #左眼被遮挡的阈值 right_eye：0.6, #右眼被遮挡的阈值 nose：0.7, #鼻子被遮挡的阈值 mouth：0.7, #嘴巴被遮挡的阈值 left_check：0.8, #左脸颊被遮挡的阈值 right_check：0.8, #右脸颊被遮挡的阈值 chin_contour：0.6, #下巴被遮挡阈值
模糊度范围	**blur**，取值范围[0~1]，0是最清晰，1是最模糊	小于0.7
光照范围	**illumination**，取值范围[0~255] 脸部光照的灰度值，0表示光照不好 以及对应客户端SDK中，YUV的Y分量	大于40
姿态角度	**Pitch**：三维旋转之俯仰角度[-90(上), 90(下)] **Roll**：平面内旋转角[-180(逆时针), 180(顺时针)] **Yaw**：三维旋转之左右旋转角[-90(左), 90(右)]	分别小于20度
人脸完整度	**completeness**（0或1），0为人脸溢出图像边界，1为人脸都在图像边界内	视业务逻辑判断
人脸大小	人脸部分的大小 建议长宽像素值范围：80*80~200*200	人脸部分不小于**100*100**像素

调用方式

请求URL数据格式

向API服务地址使用POST发送请求，必须在URL中带上参数`access_token`，可通过后台的API Key和Secret Key生成，具体方式请参考“Access Token获取”。

获取access_token的示例代码

{% AccessToken %}

> **注意**：`access_token`的有效期为30天，**切记需要每30天进行定期更换，或者每次请求都拉取新token**；

例如此接口，使用HTTPS POST发送：

```
https://aip.baidubce.com/rest/2.0/face/v2/detect?
access_token=24.f9ba9c5341b67688ab5added8bc91dec.2592000.1485570332.282335-8574074
```

POST中Body的参数，按照下方请求参数说明选择即可。

> **提示**：如果您为百度云老用户，正在使用其他非AI的服务，可以参考百度云AKSK鉴权方式发送请求，虽然**请求方式**和**鉴权方法**和本文所介绍的不同，但请求参数和返回结果一致。

请求说明

注意事项：

- **请求体格式化**：Content-Type为`application/x-www-form-urlencoded`，通过`urlencode`格式化请求体。

- **Base64编码**：请求的图片需经过Base64编码，图片的base64编码指将图片数据编码成一串字符串，使用该字符串代替图像地址。您可以首先得到图片的二进制，然后用Base64格式编码即可。需要注意的是，图片的base64编码是不包含图片头的，如
- **图片格式**：现支持PNG、JPG、JPEG、BMP，**不支持GIF图片**
- **人脸识别接口分为V2和V3两个版本，本文档为V2版本接口的说明文档，请确认您在百度云后台获得的是V2版本接口权限，再来阅读本文档。**

> 辨别接口版本的方法是：在百度云后台进入【应用列表】，点击【应用名称】，在【API列表】中可以看到【请求地址】，若请求地址中带有【v2】标识，则您具有的是v2权限，可以阅读本文档；若请求地址中带有【v3】标识，则您具有的是v3权限，应该去阅读v3文档。

请求示例

HTTP方法：POST

请求URL： https://aip.baidubce.com/rest/2.0/face/v2/detect

URL参数：

参数	值
access_token	通过API Key和Secret Key获取的access_token,参考“Access Token获取”

Header：

参数	值
Content-Type	application/x-www-form-urlencoded

Body中放置请求参数，参数详情如下：

请求参数

参数	必选	类型	说明
image	是	string	**base64编码后的图片数据，需urlencode，编码后的图片大小不超过2M**
max_face_num	否	uint32	最多处理人脸的数目，默认值为1，仅检测图片中面积最大的那个人脸
face_fields	否	string	包括**age,beauty,expression,faceshape,gender,glasses,landmark,qualities**信息，逗号分隔，**默认只返回人脸框、概率和旋转角度**

> 说明：face_fields参数，默认只返回人脸框、概率和旋转角度，age等更多属性，请在此参数中添加。

请求示例代码

提示一：使用示例代码前，请记得替换其中的示例Token、图片地址或Base64信息。

提示二：部分语言依赖的类或库，请在代码注释中查看下载地址。

{% Face-API-Detect %}

返回说明

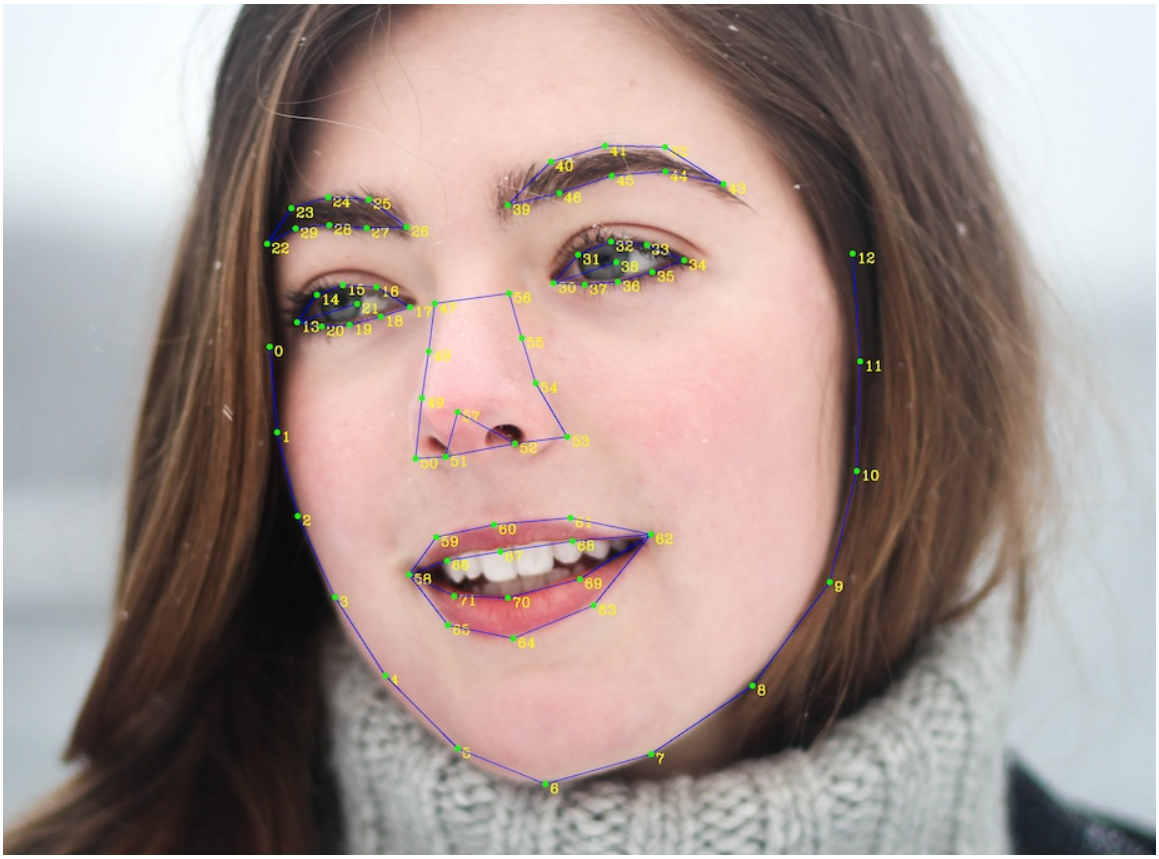
返回参数

参数	类型	必选	说明

参数	类型	必选	说明
log_id	uint64	是	日志id
result_num	uint32	是	人脸数目
result	object[]	是	人脸属性对象的集合
+age	double	否	年龄。face_fields包含age时返回
+beauty	double	否	美丑打分，范围0-100，越大表示越美。face_fields包含beauty时返回
+location	object	是	人脸在图片中的位置
++left	uint32	是	人脸区域离左边界的距离
++top	uint32	是	人脸区域离上边界的距离
++width	uint32	是	人脸区域的宽度
++height	uint32	是	人脸区域的高度
+face_probability	double	是	人脸置信度，范围0-1
+rotation_angle	int32	是	人脸框相对于竖直方向的顺时针旋转角，[-180,180]
+yaw	double	是	三维旋转之左右旋转角[-90(左), 90(右)]
+pitch	double	是	三维旋转之俯仰角度[-90(上), 90(下)]
+roll	double	是	平面内旋转角[-180(逆时针), 180(顺时针)]
+expression	uint32	否	表情，0，不笑；1，微笑；2，大笑。face_fields包含expression时返回
+expression_probability	double	否	表情置信度，范围0~1。face_fields包含expression时返回
+faceshape	object[]	否	脸型置信度。face_fields包含faceshape时返回
++type	string	是	脸型：square/triangle/oval/heart/round
++probability	double	是	置信度：0~1
+gender	string	否	male、female。face_fields包含gender时返回
+gender_probability	double	否	性别置信度，范围[0~1]，face_fields包含gender时返回
+glasses	uint32	否	是否带眼镜，0-无眼镜，1-普通眼镜，2-墨镜。face_fields包含glasses时返回
+glasses_probability	double	否	眼镜置信度，范围[0~1]，face_fields包含glasses时返回
+landmark	object[]	否	4个关键点位置，左眼中心、右眼中心、鼻尖、嘴中心。face_fields包含landmark时返回
++x	uint32	否	x坐标
++y	uint32	否	y坐标
+landmark72	object[]	否	72个特征点位置，face_fields包含landmark时返回
++x	uint32	否	x坐标
++y	uint32	否	y坐标
+qualities	object	否	人脸质量信息。face_fields包含qualities时返回
++occlusion	object	是	人脸各部分遮挡的概率，范围[0~1]，0表示完整，1表示不完整
+++left_eye	double	是	左眼遮挡比例
+++right_eye	double	是	右眼遮挡比例
+++nose	double	是	鼻子遮挡比例
+++mouth	double	是	嘴巴遮挡比例
+++left_cheek	double	是	左脸颊遮挡比例
+++right_cheek	double	是	右脸颊遮挡比例

+++right_cheek	double	是	右脸颊遮挡比例
+++chin	double	是	下巴遮挡比例
++blur	double	是	人脸模糊程度，范围[0~1]，0表示清晰，1表示模糊
++illumination	-	是	取值范围在[0~255],表示脸部区域的光照程度
++completeness	-	是	人脸完整度，0或1, 0为人脸溢出图像边界，1为人脸都在图像边界内
++type	object	是	真实人脸/卡通人脸置信度
+++human	-	是	真实人脸置信度，[0~1]，大于0.5可以判断为人脸
+++cartoon	-	是	卡通人脸置信度，[0~1]

72个关键点分布图（对应landmark72个点的顺序，序号从0-71）：



返回示例

```
{
  "result_num": 1,
  "result": [
    {
      "location": {
        "left": 117,
        "top": 131,
        "width": 172,
        "height": 170
      },
      "face_probability": 1,
      "rotation_angle": 2,
      "yaw": -0.34859421849251,
      "pitch": 2.3033397197723,
      "roll": 1.9135693311691,
      "landmark": [
        {
          "x": 161.74819946289,
          "y": 163.30244445801
        },
        ...
      ],
      "landmark72": [
        {
          "x": 115.86531066895,
          "y": 170.0546875
        } ,
        ...
      ],
      "age": 29.298097610474,
      "beauty": 55.128883361816,
      "expression": 1,
      "expression_probablity": 0.5543018579483,
      "gender": "male",
      "gender_probability": 0.99979132413864,
      "glasses": 0,
      "glasses_probability": 0.99999964237213,
      "qualities": {
        "occlusion": {
          "left_eye": 0,
          "right_eye": 0,
          "nose": 0,
          "mouth": 0,
          "left_cheek": 0.0064102564938366,
          "right_cheek": 0.0057411273010075,
          "chin": 0
        },
        "blur": 1.1886881756684e-10,
        "illumination": 141,
        "completeness": 1,
        "type": {
          "human": 0.99935841560364,
          "cartoon": 0.00064159056637436
        }
      }
    }
  ],
  "log_id": 2493878179101621
}
```


错误码

请参考人脸识别错误码

人脸对比

人脸识别接口分为V2和V3两个版本，本文档为V2版本接口的说明文档，请确认您在百度云后台获得的是V2版本接口权限，再来阅读本文档。

> 辨别接口版本的方法是：在百度云后台进入【应用列表】，点击【应用名称】，在【API列表】中可以看到【请求地址】，若请求地址中带有【v2】标识，则您具有的是v2权限，可以阅读本文档；若请求地址中带有【v3】标识，则您具有的是v3权限，应该去阅读v3文档。 如果您对文档内容有任何疑问，可以通过以下几种方式联系我们：

- 在百度云控制台内 提交工单，咨询问题类型请选择**人工智能服务**；
- 如有需要讨论的疑问，欢迎进入 AI社区 与其他开发者们一同交流。

能力介绍

接口能力

- **两张人脸图片相似度对比**：比对两张图片中人脸的相似度，并返回相似度分值；
- **多种图片类型**：支持**生活照**、**证件照**、**身份证芯片照**、**带网纹照**四种类型的人脸对比；
- **活体检测**：基于图片中的破绽分析，判断其中的人脸是否为**二次翻拍**（举例：如用户A用手机拍摄了一张包含人脸的图片一，用户B翻拍了图片一得到了图片二，并用图片二伪造成用户A去进行识别操作，这种情况普遍发生在金融开户、实名认证等环节。）；
- **质量检测**：返回模糊、光照等质量检测信息，用于辅助判断图片是否符合识别要求；

业务应用

用于比对多张图片中的人脸相似度并返回两两比对的得分，可用于判断两张脸是否是同一人的可能性大小。

典型应用场景：如**人证合一验证**，**用户认证**等，可与您现有的人脸库进行比对验证。

调用方式

请求URL数据格式

向API服务地址使用POST发送请求，必须在URL中带上参数`access_token`，可通过后台的API Key和Secret Key生成，具体方式请参考“Access Token获取”。

获取access_token的示例代码

{% AccessToken %}

> **注意**：`access_token`的有效期为30天，**切记需要每30天进行定期更换，或者每次请求都拉取新token**；

例如此接口，使用HTTPS POST发送：

```
https://aip.baidubce.com/rest/2.0/face/v2/match?
access_token=24.f9ba9c5341b67688ab5added8bc91dec.2592000.1485570332.282335-8574075
```

POST中Body的参数，按照下方请求参数说明选择即可。

> **提示**：如果您为百度云老用户，正在使用其他非AI的服务，可以参考百度云AKSK鉴权方式发送请求，虽然**请求方式**和**鉴权**

> **方法**和本文所介绍的不同，但请求参数和返回结果一致。

请求说明

注意事项：

- **请求体格式化**：Content-Type为`application/x-www-form-urlencoded`，通过`urlencode`格式化请求体。
- **Base64编码**：请求的图片需经过`Base64编码`，图片的base64编码指将图片数据编码成一串字符串，使用该字符串代替图像地址。您可以首先得到图片的二进制，然后用Base64格式编码即可。需要注意的是，图片的base64编码是不包含图片头的，如``
- **图片格式**：现支持PNG、JPG、JPEG、BMP，**不支持GIF图片**
- **人脸识别接口分为V2和V3两个版本，本文档为V2版本接口的说明文档，请确认您在百度云后台获得的是V2版本接口权限，再来阅读本文档。**

> 辨别接口版本的方法是：在百度云后台进入【应用列表】，点击【应用名称】，在【API列表】中可以看到【请求地址】，若请求地址中带有【v2】标识，则您具有的是v2权限，可以阅读本文档；若请求地址中带有【v3】标识，则您具有的是v3权限，应该去阅读v3文档。

请求示例

HTTP方法：`POST`

请求URL： `https://aip.baidubce.com/rest/2.0/face/v2/match`

URL参数：

参数	值
access_token	通过API Key和Secret Key获取的access_token,参考“Access Token获取”

Header：

参数	值
Content-Type	application/x-www-form-urlencoded

Body中放置请求参数，参数详情如下：

请求参数

参数	必选	类型	说明
images	是	string	**分别base64编码后的2张图片数据，需urlencode，半角逗号分隔，单次请求最大不超过20M**
ext_fields	否	string	返回质量信息，取值固定，目前支持qualities(质量检测)(对所有图片都会做改处理)
image_liveness	否	string	返回的活体信息，“faceliveness,faceliveness” 表示对比对的两张图片都做活体检测；“,faceliveness” 表示对第一张图片不做活体检测、第二张图做活体检测；“faceliveness,” 表示对第一张图片做活体检测、第二张图不做活体检测； **注：需要用于判断活体的图片，图片中的人脸像素面积需要不小于100px*100px，人脸长宽与图片长宽比例，不小于1/3**
types	否	string	请求对比的两张图片的类型，示例：“7，13” **7**表示生活照：通常为手机、相机拍摄的人像图片、或从网络获取的人像图片等 **11**表示身份证芯片照：二代身份证内置芯片中的人像照片 **12**表示带水印证件照：一般为带水印的小图，如公安网小图 **13**表示证件照片：如拍摄的身份证、工卡、护照、学生证等证件图片，**注**：需要确保人脸部分不可太小，通常为100px*100px

> 说明：两张请求的图片请分别进行base64编码。

请求代码示例

提示一：使用示例代码前，请记得替换其中的示例Token、图片地址或Base64信息。

提示二：部分语言依赖的类或库，请在代码注释中查看下载地址。

{% Face-API-Match %}

返回说明

返回参数

字段	必选	类型	说明
log_id	是	uint64	请求唯一标识码，随机数
result_num	是	uint32	返回结果数目，即：result数组中元素个数
result	是	array(object)	结果数据，index和请求图片index对应。数组元素为每张图片的匹配得分数组，top n。得分范围[0,100.0]
+index_i	是	uint32	比对图片1的index
+index_j	是	uint32	比对图片2的index
+score	是	double	比对得分，推荐**80分**作为阈值，80分以上可以判断为同一人，此分值对应万分之一误识率
ext_info	否	array（dict）	对应参数中的ext_fields
+qualities	否	string	质量相关的信息，无特殊需求可以不使用
+faceliveness	否	string	活体检测分数，单帧活体检测参考阈值**0.3**，超过此分值以上则可认为是活体。**注意：活体检测接口主要用于判断是否为二次翻拍，需要限制用户为当场拍照获取图片**；推荐配合客户端SDK有动作校验活体使用

返回示例

```
 //请求两张图片
{
    "log_id": 73473737,
    "result_num":1,
    "result": [
        {
            "index_i": 0,
            "index_j": 1,
            "score": 44.3
        }
    ]
}
```

关于活体检测faceliveness的判断阈值选择，可参考以下数值信息：

阈值（Threshold）	误拒率（FRR）	通过率（TAR）	攻击拒绝率（TRR））
0.05	0.01%	99.99%	97.75%
0.1	0.05%	99.95%	98.33%
0.3 （推荐）	0.1%	99.9%	98.82%
0.5	0.5%	99.5%	99.67%
0.9	1%	99%	99.77%

关于以上数值的概念介绍：

- **拒绝率（TRR）**：如99%，代表100次作弊假体攻击，会有99次被拒绝。
- **误拒率（FRR）**：如0.5%，指1000次真人请求，会有5次因为活体分数低于阈值被错误拒绝。
- **通过率（TAR）**：如99%，指100次真人请求，会有99次因为活体分数高于阈值而通过。
- **阈值（Threshold）**：高于此数值，则可判断为活体。

错误码

请参考人脸识别错误码

人脸查找

概述

人脸识别接口分为V2和V3两个版本，本文档为V2版本接口的说明文档，请确认您在百度云后台获得的是V2版本接口权限，再来阅读本文档。

> 辨别接口版本的方法是：在百度云后台进入【应用列表】，点击【应用名称】，在【API列表】中可以看到【请求地址】，若请求地址中带有【v2】标识，则您具有的是v2权限，可以阅读本文档；若请求地址中带有【v3】标识，则您具有的是v3权限，应该去阅读v3文档。 如果您对文档内容有任何疑问，可以通过以下几种方式联系我们：

- 在百度云控制台内 提交工单，咨询问题类型请选择**人工智能服务**；
- 如有需要讨论的疑问，欢迎进入 AI社区 与其他开发者们一同交流。

能力介绍

业务能力

- **1：N人脸识别**：也称为1：N识别，在指定人脸集合中，找到最相似的人脸；
- **1：N人脸认证**：基于uid维度的1：N识别，由于uid已经锁定固定数量的人脸，所以检索范围更聚焦；
- **M：N多人脸识别**：也称为M：N识别，待识别图片中含有多个人脸时，在指定人脸集合中，找到这多个人脸分别最相似的人脸；

> **1：N人脸识别**与**1：N人脸认证**的差别在于：人脸识别是在指定人脸集合中进行直接地人脸检索操作，而人脸认证是基于uid，先调取这个uid对应的人脸，再在这个uid对应的人脸集合中进行检索（因为每个uid通常对应的只有一张人脸，所以通常也就变为了1：1对比）；实际应用中，人脸认证需要用户或系统先输入id，这增加了验证安全度，但也增加了复杂度，具体使用哪个接口需要视您的业务场景判断。

> M：N识别的原理，相当于在多个人脸的图片中，先分别找出所有人脸，然后分别在待查找的人脸集合中，分别做1：N识别，最后将识别结果汇总在一起进行返回。

> 提示：进行人脸查找相关操作前，建议先阅读 人脸库管理 相关内容。

调用方式

请求URL数据格式

向API服务地址使用POST发送请求，必须在URL中带上参数`access_token`，可通过后台的API Key和Secret Key生成，具体方式请参考“Access Token获取”。

获取access_token的示例代码

{% AccessToken %}

> **注意**：`access_token`的有效期为30天，**切记需要每30天进行定期更换，或者每次请求都拉取新token**；

例如此接口，使用HTTPS POST发送：

```
https://aip.baidubce.com/rest/2.0/face/v2/identify?
access_token=24.f9ba9c5341b67688ab4added8bc91dec.2592000.1485570332.282335-8574074
```

POST中Body的参数，按照下方请求参数说明选择即可。

> **提示**：如果您为百度云老用户，正在使用其他非AI的服务，可以参考百度云AKSK鉴权方式发送请求，虽然**请求方式**和**鉴权方法**和本文所介绍的不同，但请求参数和返回结果一致。

人脸识别

接口描述

用于计算指定组内用户，与上传图像中人脸的相似度。识别前提为您已经创建了一个人脸库。

典型应用场景：如**人脸闸机**，**考勤签到**，**安防监控**等。

> **说明**：人脸识别返回值不直接判断是否是同一人，只返回用户信息及相似度分值。

> 提示：进行人脸查找相关操作前，建议先阅读 人脸库管理 相关内容。

请求说明

注意事项：

- **请求体格式化**：Content-Type为`application/x-www-form-urlencoded`，通过`urlencode`格式化请求体。
- **Base64编码**：请求的图片需经过`Base64编码`，图片的base64编码指将图片数据编码成一串字符串，使用该字符串代替图像地址。您可以首先得到图片的二进制，然后用Base64格式编码即可。需要注意的是，图片的base64编码是不包含图片头的，如``
- **图片格式**：现支持PNG、JPG、JPEG、BMP，**不支持GIF图片**
- **人脸识别接口分为V2和V3两个版本，本文档为V2版本接口的说明文档，请确认您在百度云后台获得的是V2版本接口权限，再来阅读本文档。**

> 辨别接口版本的方法是：在百度云后台进入【应用列表】，点击【应用名称】，在【API列表】中可以看到【请求地址】，若请求地址中带有【v2】标识，则您具有的是v2权限，可以阅读本文档；若请求地址中带有【v3】标识，则您具有的是v3权限，应该去阅读v3文档。

请求示例

HTTP方法：`POST`

请求URL：`https://aip.baidubce.com/rest/2.0/face/v2/identify`

URL参数：

参数	值
access_token	通过API Key和Secret Key获取的access_token,参考“Access Token获取”

Header：

参数	值
Content-Type	application/x-www-form-urlencoded

Body中放置请求参数，参数详情如下：

请求参数

参数	必选	类型	说明
image	是	string	**base64编码后的图片数据，需urlencode，每次只支持单张图片，编码后的图片大小不超过10M**
group_id	是	string	用户组id（由数字、字母、下划线组成），长度限制128B，如果需要查询多个用户组id，用逗号分隔
ext_fields	否	string	特殊返回信息，多个用逗号分隔，取值固定: 目前支持faceliveness(活体检测)。**注：需要用于判断活体的图片，图片中的人脸像素面积需要不小于100px*100px，人脸长宽与图片长宽比例，不小于1/3**
user_top_num	否	uint32	识别后返回的用户top数，默认为1，最多返回5个

请求代码示例

提示一：使用示例代码前，请记得替换其中的示例Token、图片地址或Base64信息。

提示二：部分语言依赖的类或库，请在代码注释中查看下载地址。

{% Face-API-Identify %}

返回说明

返回参数

字段	必选	类型	说明
log_id	是	uint64	请求唯一标识码，随机数
result_num	是	uint32	返回结果数目，即：result数组中元素个数
ext_info	否	array	对应参数中的ext_fields
+faceliveness	否	string	活体检测分数，单帧活体检测参考阈值**0.3**，超过此分值以上则可认为是活体。**注意：活体检测接口主要用于判断是否为二次翻拍，需要限制用户为当场拍照获取图片**；推荐配合客户端SDK有动作校验活体使用
result	是	array(object)	结果数组
+group_id	是	string	对应的这个用户的group_id
+uid	是	string	匹配到的用户id
+user_info	是	string	注册时的用户信息
+scores	是	array(double)	结果数组，数组元素为匹配得分，top n。得分[0,100.0]。推荐**80分**作为阈值，80分以上可以判断为同一人，此分值对应万分之一误识率。

返回示例

```
{
  "log_id": 73473737,
  "result_num":1,
  "result": [
     {
        "group_id" : "test1",
        "uid": "u333333",
        "user_info": "Test User",
        "scores": [
              99.3,
              83.4
        ]
     }
  ]
}
```

关于活体检测faceliveness的判断阈值选择，可参考以下数值信息：

阈值（Threshold）	误拒率（FRR）	通过率（TAR）	攻击拒绝率（TRR））
0.05	0.01%	99.99%	97.75%
0.1	0.05%	99.95%	98.33%
0.3 （**推荐**）	0.1%	99.9%	98.82%
0.5	0.5%	99.5%	99.67%
0.9	1%	99%	99.77%

关于以上数值的概念介绍：

- **拒绝率（TRR）**：如99%，代表100次作弊假体攻击，会有99次被拒绝。
- **误拒率（FRR）**：如0.5%，指1000次真人请求，会有5次因为活体分数低于阈值被错误拒绝。
- **通过率（TAR）**：如99%，指100次真人请求，会有99次因为活体分数高于阈值而通过。
- **阈值（Threshold）**：高于此数值，则可判断为活体。

人脸认证

接口描述

用于识别上传的图片是否为指定用户，即查找前需要先确定要查找的用户在人脸库中的id。

典型应用场景：如**人脸登录**，**人脸签到**等

> 提示：进行人脸查找相关操作前，建议先阅读 人脸库管理 相关内容。

请求说明

注意事项：

- **请求体格式化**：Content-Type为`application/x-www-form-urlencoded`，通过`urlencode`格式化请求体。
- **Base64编码**：请求的图片需经过`Base64编码`，图片的base64编码指将图片数据编码成一串字符串，使用该字符串代替图像地址。您可以首先得到图片的二进制，然后用Base64格式编码即可。需要注意的是，图片的base64编码是不包含图片头的，如``
- **图片格式**：现支持PNG、JPG、JPEG、BMP，**不支持GIF图片**

请求示例

HTTP方法：`POST`

请求URL：`https://aip.baidubce.com/rest/2.0/face/v2/verify`

URL参数：

参数	值
access_token	通过API Key和Secret Key获取的access_token,参考“Access Token获取”

Header如下：

参数	值
Content-Type	application/x-www-form-urlencoded

Body中放置请求参数，参数详情如下：

请求参数

参数	必选	类型	说明
uid	是	string	用户id（由数字、字母、下划线组成），长度限制128B
image	是	string	**base64编码后的图片数据，需urlencode，每次只支持单张图片，编码后的图片大小不超过10M**
group_id	是	string	用逗号分隔，表示从指定的group中查找
top_num	否	uint32	返回匹配得分top数，默认为1
ext_fields	否	string	特殊返回信息，多个用逗号分隔，取值固定: 目前支持faceliveness(活体检测)。**注：需要用于判断活体的图片，图片中的人脸像素面积需要不小于100px*100px，人脸长宽与图片长宽比例，不小于1/3**

请求代码示例

提示一：使用示例代码前，请记得替换其中的示例Token、图片地址或Base64信息。

提示二：部分语言依赖的类或库，请在代码注释中查看下载地址。

{% Face-API-Verify %}

返回说明

返回参数

字段	必选	类型	说明
log_id	是	uint64	请求唯一标识码，随机数
result_num	是	uint32	返回结果数目，即：result数组中元素个数
result	是	array(double)	结果数组，数组元素为匹配得分，top n。 得分范围[0,100.0]。超过**80分**可认为认证成功
ext_info	否	array	对应参数中的ext_fields
+faceliveness	否	string	活体检测分数，单帧活体检测参考阈值**0.3**，超过此分值以上则可认为是活体。**活体检测接口主要用于判断是否为二次翻拍，需要限制用户为当场拍照获取图片**；推荐配合客户端SDK有动作校验活体使用

返回示例

```
{
  "log_id": 73473737,
  "result_num":2,
  "result": [
        99.3,
        83.6
  ]
}
```

关于活体检测faceliveness的判断阈值选择，可参考以下数值信息：

阈值（Threshold）	误拒率（FRR）	通过率（TAR）	攻击拒绝率（TRR））
0.05	0.01%	99.99%	97.75%
0.1	0.05%	99.95%	98.33%
0.3 （推荐）	0.1%	99.9%	98.82%
0.5	0.5%	99.5%	99.67%
0.9	1%	99%	99.77%

关于以上数值的概念介绍：

- **拒绝率（TRR）**：如99%，代表100次作弊假体攻击，会有99次被拒绝。
- **误拒率（FRR）**：如0.5%，指1000次真人请求，会有5次因为活体分数低于阈值被错误拒绝。
- **通过率（TAR）**：如99%，指100次真人请求，会有99次因为活体分数高于阈值而通过。
- **阈值（Threshold）**：高于此数值，则可判断为活体。

M:N 识别

待识别的图片中，存在多张人脸的情况下，支持在一个人脸库中，一次请求，同时返回图片中所有人脸的识别结果。

> 提示：进行人脸查找相关操作前，建议先阅读 人脸库管理 相关内容。

请求说明

注意事项：

- **请求体格式化**：Content-Type为`application/x-www-form-urlencoded`，通过`urlencode`格式化请求体。
- **Base64编码**：请求的图片需经过`Base64编码`，图片的base64编码指将图片数据编码成一串字符串，使用该字符串代替图像地址。您可以首先得到图片的二进制，然后用Base64格式编码即可。需要注意的是，图片的base64编码是不包含图片头的，如``
- **图片格式**：现支持PNG、JPG、JPEG、BMP，**不支持GIF图片**

请求示例

HTTP方法：POST

请求URL： `https://aip.baidubce.com/rest/2.0/face/v2/multi-identify`

URL参数：

参数	值
access_token	通过API Key和Secret Key获取的access_token,参考“Access Token获取”

Header如下：

参数	值
Content-Type	application/x-www-form-urlencoded

Body中放置请求参数，参数详情如下：

请求参数

参数	必选	类型	说明
image	是	string	**base64编码后的图片数据，需urlencode，每次只支持单张图片，编码后的图片大小不超过10M**
group_id	是	string	用户组id（由数字、字母、下划线组成），长度限制128B，用户组id，如果有多个，用逗号分隔，最多支持10个，不允许出现空ID或重复ID
ext_fields	否	string	特殊返回信息，多个用逗号分隔，取值固定: 目前支持faceliveness(活体检测)。**注：需要用于判断活体的图片，图片中的人脸像素面积需要不小于100px*100px，人脸长宽与图片长宽比例，不小于1/3**
detect_top_num	否	uint32	检测多少个人脸进行比对，默认值1（最对返回10个）
user_top_num	否	uint32	返回识别结果top数，当同一个人有多张图片时，只返回比对最高的1个分数（即，scores参数只有一个值），默认为1（最多返回20个）

返回说明

返回参数

参数	字段	必选	类型	说明
log_id	-	是	uint32	请求标识码，随机数，唯一
result_num	-	是	float	返回结果数目，即：result数组中元素个数（PS：最终返回的个数是小于等于 “实际检测到的人脸数” * “每个人脸匹配的top人数”）
ext_info	-	否	array	对应参数中的ext_fields
-	faceliveness	否	string	活体检测分数，单帧活体检测参考阈值**0.3**，超过此分值以上则可认为是活体。**注意：活体检测接口主要用于判断是否为二次翻拍，需要限制用户为当场拍照获取图片**；推荐配合客户端SDK有动作校验活体使用
result	-	是	array(object)	-
-	group_id	是	string	对应的这个用户的group_id
-	uid	是	string	匹配到的用户id
-	user_info	是	string	注册时的用户信息
-	scores	是	array(double)	结果数组，数组元素为匹配得分，得分[0,100.0]；个数取决于user_top_num的设定。推荐**80分**以上即可判断为同一人
-	position	是	object	人脸位置，如{top:111,left:222,width:333,height:444,degree:20}

返回示例

```
{
  "log_id": 73473737,
  "result_num":1,
  "result": [
     {
        "group_id" : "test1",
        "uid": "u333333",
        "user_info": "Test User",
        "position":{"left":726.99188232422,"top":288.37701416016,"width":44,"height":42,"degree":-
4,"prob":0.91117089986801},
        "scores": [
              99.3
        ]
     },
     {
        "group_id" : "test1",
        "uid": "u2222222",
        "user_info": "Test User",
        "position":{"left":726.99188232422,"top":288.37701416016,"width":44,"height":42,"degree":-
4,"prob":0.91117089986801},
        "scores": [
              82.3
        ]
     }
  ]
}
```


关于活体检测faceliveness的判断阈值选择，可参考以下数值信息：

阈值（Threshold）	误拒率（FRR）	通过率（TAR）	攻击拒绝率（TRR））
0.05	0.01%	99.99%	97.75%
0.1	0.05%	99.95%	98.33%
0.3 （推荐）	0.1%	99.9%	98.82%
0.5	0.5%	99.5%	99.67%
0.9	1%	99%	99.77%

关于以上数值的概念介绍：

- **拒绝率（TRR）**：如99%，代表100次作弊假体攻击，会有99次被拒绝。
- **误拒率（FRR）**：如0.5%，指1000次真人请求，会有5次因为活体分数低于阈值被错误拒绝。
- **通过率（TAR）**：如99%，指100次真人请求，会有99次因为活体分数高于阈值而通过。
- **阈值（Threshold）**：高于此数值，则可判断为活体。

错误码

请参考人脸识别错误码

人脸库管理

概述

人脸识别接口分为V2和V3两个版本，本文档为V2版本接口的说明文档，请确认您在百度云后台获得的是V2版本接口权限，再来阅读本文档。

> 辨别接口版本的方法是：在百度云后台进入【应用列表】，点击【应用名称】，在【API列表】中可以看到【请求地址】，若请求地址中带有【v2】标识，则您具有的是v2权限，可以阅读本文档；若请求地址中带有【v3】标识，则您具有的是v3权限，应该去阅读v3文档。 如果您对文档内容有任何疑问，可以通过以下几种方式联系我们：

- 在百度云控制台内 提交工单，咨询问题类型请选择**人工智能服务**；
- 如有需要讨论的疑问，欢迎进入 AI社区 与其他开发者们一同交流。

能力介绍

业务能力

人脸库管理相关接口，要完成1：N或者M：N识别，首先需要构建一个人脸库，用于存放所有人脸特征，相关接口如下：

- **人脸注册**：向人脸库中添加人脸
- **人脸更新**：更新人脸库中，指定用户下的人脸信息
- **人脸删除**：删除人脸库中的某个用户
- **用户信息查询**：查询人脸库中某个用户的详细信息
- **组列表查询**：查询人脸库中用户组的列表
- **组内用户列表查询**：查询指定用户组中的用户列表
- **组间复制用户**：将已经存在于人脸库中的用户复制到一个新的组
- **组内删除用户**：将指定用户从某个组中删除，但不会删除用户在其它组的信息

人脸库结构

人脸库、用户组、用户、用户下的人脸**层级关系**如下所示：

```
|- 人脸库(appid)
 |- 用户组一（group_id）
    |- 用户01（uid）
       |- 人脸（faceid）
    |- 用户02（uid）
       |- 人脸（faceid）
       |- 人脸（faceid）
       ....
     ....
 |- 用户组二（group_id）
 |- 用户组三（group_id）
 ....
```

关于人脸库的设置限制

- 每个开发者账号可以创建100个appid；
- **每个appid对应一个人脸库，且不同appid之间，人脸库互不相通**；
- 每个人脸库下，可以创建多个用户组，用户组（group）数量**没有限制**；
- 每个用户组（group）下，可添加最多**无限**张人脸，**无限**个uid；
- 每个用户（uid）所能注册的最大人脸数量**20**；

> 提醒：每个人脸库对应一个appid，一定确保不要轻易删除后台应用列表中的appid，删除后则此人脸库将失效，无法进行

> 任何查找！

质量判断

为了保证识别效果，请控制注册人脸的质量（通过/detect人脸检测接口判断），通过人脸检测接口，基于以下字段和对应阈值，进行质量检测的判断，以保证人脸质量符合后续业务操作要求，具体参数可详见下表所示：

指标	字段与解释	推荐数值界限
遮挡范围	**occlusion**（0~1），0为无遮挡，1是完全遮挡 含有多个具体子字段，表示脸部多个部位 通常用作判断头发、墨镜、口罩等遮挡	left_eye：0.6, #左眼被遮挡的阈值 right_eye：0.6, #右眼被遮挡的阈值 nose：0.7, #鼻子被遮挡的阈值 mouth：0.7, #嘴巴被遮挡的阈值 left_check：0.8, #左脸颊被遮挡的阈值 right_check：0.8, #右脸颊被遮挡的阈值 chin_contour：0.6, #下巴被遮挡阈值
模糊度范围	**Blur**（0~1），0是最清晰，1是最模糊	小于0.7
光照范围	**illumination**（0~255） 脸部光照的灰度值，0表示光照不好 以及对应客户端SDK中，YUV的Y分量	大于40
姿态角度	**Pitch**：三维旋转之俯仰角度[-90(上), 90(下)] **Roll**：平面内旋转角[-180(逆时针), 180(顺时针)] **Yaw**：三维旋转之左右旋转角[-90(左), 90(右)]	分别小于20度
人脸完整度	**completeness**（0或1），0为人脸溢出图像边界，1为人脸都在图像边界内	视业务逻辑判断
人脸大小	人脸部分的大小 建议长宽像素值范围：80*80~200*200	人脸部分不小于**100*100**像素

调用方式

请求URL数据格式

向API服务地址使用POST发送请求，必须在URL中带上参数access_token，可通过后台的API Key和Secret Key生成，具体方式请参考“Access Token获取”。

获取access_token的示例代码

{% AccessToken %}

> **注意**：access_token的有效期为30天，**切记需要每30天进行定期更换，或者每次请求都拉取新token**；

例如此接口，使用HTTPS POST发送：

```
https://aip.baidubce.com/rest/2.0/face/v2/faceset/user/add?
access_token=24.f9ba9c5341b67688ab6added8bc91dec.2592000.1485570332.282335-8574074
```

POST中Body的参数，按照下方请求参数说明选择即可。

> **提示**：如果您为百度云老用户，正在使用其他非AI的服务，可以参考百度云AKSK鉴权方式发送请求，虽然**请求方式**和**鉴权**

> **方法**和本文所介绍的不同，但请求参数和返回结果一致。

人脸注册

接口描述

用于从人脸库中新增用户，可以设定多个用户所在组，及组内用户的人脸图片，

典型应用场景：构建您的人脸库，如**会员人脸注册**，**已有用户补全人脸信息**等。

请求说明

注意事项：

- **请求体格式化**：Content-Type为`application/x-www-form-urlencoded`，通过`urlencode`格式化请求体。
- **Base64编码**：请求的图片需经过`Base64编码`，图片的base64编码指将图片数据编码成一串字符串，使用该字符串代替图像地址。您可以首先得到图片的二进制，然后用Base64格式编码即可。需要注意的是，图片的base64编码是不包含图片头的，如``
- **图片格式**：现支持PNG、JPG、JPEG、BMP，**不支持GIF图片**
- **人脸识别接口分为V2和V3两个版本，本文档为V2版本接口的说明文档，请确认您在百度云后台获得的是V2版本接口权限，再来阅读本文档。**

> 辨别接口版本的方法是：在百度云后台进入【应用列表】，点击【应用名称】，在【API列表】中可以看到【请求地址】，若请求地址中带有【v2】标识，则您具有的是v2权限，可以阅读本文档；若请求地址中带有【v3】标识，则您具有的是v3权限，应该去阅读v3文档。 **请求示例**

HTTP方法：`POST`

请求URL： `https://aip.baidubce.com/rest/2.0/face/v2/faceset/user/add`

URL参数：

参数	值
access_token	通过API Key和Secret Key获取的access_token,参考“Access Token获取”

Header如下：

参数	值
Content-Type	application/x-www-form-urlencoded

Body中放置请求参数，参数详情如下：

请求参数

参数	必选	类型	说明
uid	是	string	用户id（由数字、字母、下划线组成），长度限制128B。
group_id	是	string	用户组id，标识一组用户（由数字、字母、下划线组成），长度限制128B。如果需要将一个uid注册到多个group下，group_id需要用多个逗号分隔，每个group_id长度限制为48个英文字符。**注：group无需单独创建，注册用户时则会自动创建group。** **产品建议**：根据您的业务需求，可以将需要注册的用户，按照业务划分，分配到不同的group下，例如按照会员手机尾号作为groupid，用于刷脸支付、会员计费消费等，这样可以尽可能控制每个group下的用户数与人脸数，提升检索的准确率
image	是	string	**base64编码后的图片数据，需urlencode，每次只支持单张图片，编码后的图片大小不超过10M**。为保证后续识别的效果较佳，建议注册的人脸，为用户正面人脸，且保持人脸都在图片之内
user_info	是	string	用户资料，长度限制256B
action_type	否	string	参数包含app、replace。**如果为"replace"，则每次注册时进行替换replace（新增或更新）操作，默认为append操作**。例如：uid在库中已经存在时，对此uid重复注册时，新注册的图片默认会**追加**到该uid下，如果手动选择`action_type:replace`，则会用新图替换库中该uid下所有图片。

> **说明**：人脸注册完毕后，生效时间**一般为5s以内**，之后便可以进行识别或认证操作。

请求代码示例

提示一：使用示例代码前，请记得替换其中的示例Token、图片地址或Base64信息。

提示二：部分语言依赖的类或库，请在代码注释中查看下载地址。

{% Face-API-Add %}

返回说明

返回参数

字段	必选	类型	说明
log_id	是	uint64	请求标识码，随机数，唯一

返回示例

```
 // 注册成功
{
    "log_id": 73473737,
}

// 注册发生错误
{
    "error_code": 216616,
    "log_id": 674786177,
    "error_msg": "image exist"
}
```

人脸更新

接口描述

用于对人脸库中指定用户，更新其下的人脸图像。

> **说明**：针对一个uid执行更新操作，新上传的人脸图像将覆盖此uid原有所有图像。

> **说明**：在uid不存在时，调用此接口，会报错。如果希望此uid不在人脸库中时，创建这个uid，请添加action_type:replace,注册该uid，操作结果等同注册新用户。

请求说明

注意事项：

- **请求体格式化**：Content-Type为application/x-www-form-urlencoded，通过urlencode格式化请求体。
- **Base64编码**：请求的图片需经过Base64编码，图片的base64编码指将图片数据编码成一串字符串，使用该字符串代替图像地址。您可以首先得到图片的二进制，然后用Base64格式编码即可。需要注意的是，图片的base64编码是不包含图片头的，如
- **图片格式**：现支持PNG、JPG、JPEG、BMP，**不支持GIF图片**

请求示例

HTTP方法：POST

请求URL： https://aip.baidubce.com/rest/2.0/face/v2/faceset/user/update

URL参数：

参数	值
access_token	通过API Key和Secret Key获取的access_token,参考“Access Token获取”

Header如下：

参数	值
Content-Type	application/x-www-form-urlencoded

Body中放置请求参数，参数详情如下：

请求参数

参数	必选	类型	说明
uid	是	string	用户id（由数字、字母、下划线组成），长度限制128B
image	是	string	**base64编码后的图片数据，需urlencode，每次只支持单张图片，编码后的图片大小不超过10M**
user_info	是	string	用户信息，长度限制256位
group_id	是	string	更新指定groupid下uid对应的信息
action_type	否	string	**在uid不存在时，调用此接口，会报错。 如果希望此uid不在人脸库中时，创建这个uid，请添加replace参数， 注册该uid，操作结果等同注册新用户**

请求代码示例

提示一：使用示例代码前，请记得替换其中的示例Token、图片地址或Base64信息。

提示二：部分语言依赖的类或库，请在代码注释中查看下载地址。

{% Face-API-Update %}

返回说明

返回参数

字段	必选	类型	说明
log_id	是	uint64	请求标识码，随机数，唯一

返回示例

```
 // 更新成功
{
   "log_id": 73473737,
}
// 更新发生错误
{
  "error_code": 216612,
  "log_id": 1137508902,
  "error_msg": "user not exist"
}
```

人脸删除

接口描述

用于从人脸库中删除一个用户。

人脸删除注意事项：

- 删除的内容，包括用户所有图像和身份信息；
- 如果一个uid存在于多个用户组内，将会同时将从各个组中把用户删除
- 如果指定了group_id，则只删除此group下的uid相关信息

请求说明

请求示例

HTTP方法：`POST`

请求URL： `https://aip.baidubce.com/rest/2.0/face/v2/faceset/user/delete`

URL参数：

参数	值
access_token	通过API Key和Secret Key获取的access_token,参考“Access Token获取”

Header如下：

参数	值
Content-Type	application/x-www-form-urlencoded

Body中放置请求参数，参数详情如下：

请求参数

参数	必选	类型	说明
uid	是	string	用户id（由数字、字母、下划线组成），长度限制128B
group_id	否	string	删除指定group_id中的uid信息

请求代码示例

提示一：使用示例代码前，请记得替换其中的示例Token、图片地址或Base64信息。

提示二：部分语言依赖的类或库，请在代码注释中查看下载地址。

{% Face-API-Delete %}

返回说明

返回参数

字段	必选	类型	说明
log_id	是	uint64	请求标识码，随机数，唯一

返回示例

```
 // 删除成功
{
   "log_id": 73473737,
}
// 删除发生错误
{
  "error_code": 216612,
  "log_id": 1382953199,
  "error_msg": "user not exist"
}
```

用户信息查询

接口描述

用于查询人脸库中某用户的详细信息。

请求说明

请求示例

HTTP方法：`POST`

请求URL： `https://aip.baidubce.com/rest/2.0/face/v2/faceset/user/get`

URL参数：

参数	值
access_token	通过API Key和Secret Key获取的access_token,参考“Access Token获取”

Header如下：

参数	值
Content-Type	application/x-www-form-urlencoded

Body中放置请求参数，参数详情如下：

请求参数

参数	必选	类型	说明
uid	是	string	用户id（由数字、字母、下划线组成），长度限制128B
group_id	否	string	选择指定group_id则只查找group列表下的uid内容，如果不指定则查找所有group下对应uid的信息

请求代码示例

提示一：使用示例代码前，请记得替换其中的示例Token、图片地址或Base64信息。

提示二：部分语言依赖的类或库，请在代码注释中查看下载地址。

{% Face-API-UserGet %}

返回说明

返回参数

字段	必选	类型	说明
log_id	是	uint64	请求标识码，随机数，唯一
result	是	array(double)	结果数组
+uid	是	string	匹配到的用户id
+user_info	是	string	注册时的用户信息
+groups	是	array(string)	用户所属组列表

返回示例

```
 {
“result_num” : 2
   "result": {
      [
         "uid": "testuser2",
         "user_info": "registed user info ...",
         "group_id": "grep1",
      ],
      [
         "uid": "testuser2",
         "user_info": "registed user info2 ...",
         "group_id": "grep2",
      ],
   },
   "log_id": 2979357502
}
```

组列表查询

接口描述

用于查询用户组的列表。

请求说明

请求示例

HTTP方法：`POST`

请求URL： `https://aip.baidubce.com/rest/2.0/face/v2/faceset/group/getlist`

URL参数：

参数	值
access_token	通过API Key和Secret Key获取的access_token,参考“Access Token获取”

Header如下：

参数	值
Content-Type	application/x-www-form-urlencoded

Body中放置请求参数，参数详情如下：

请求参数

参数	必选	类型	说明
start	否	uint32	默认值0，起始序号
num	否	uint32	返回数量，默认值100，最大值1000

请求代码示例

提示一：使用示例代码前，请记得替换其中的示例Token、图片地址或Base64信息。

提示二：部分语言依赖的类或库，请在代码注释中查看下载地址。

{% Face-API-GroupGetlist %}

返回说明

返回参数

字段	必选	类型	说明
log_id	是	uint64	请求标识码，随机数，唯一
result_num	是	uint32	返回个数
result	是	array(string)	group_id列表

返回示例

```
{
  "result_num": 2,
  "result": [
     "grp1",
     "grp2"
  ],
  "log_id": 3314921889
}
```

组内用户列表查询

接口描述

用于查询指定用户组中的用户列表。

请求说明

请求示例

HTTP方法：`GET`

请求URL： `https://aip.baidubce.com/rest/2.0/face/v2/faceset/group/getusers`

URL参数：

参数	值
access_token	通过API Key和Secret Key获取的access_token,参考“Access Token获取”

Header如下：

参数	值
Content-Type	application/x-www-form-urlencoded

Body中放置请求参数，参数详情如下：

请求参数

参数	必选	类型	说明
group_id	是	string	用户组id
start	否	uint32	默认值0，起始序号
num	否	uint32	返回数量，默认值100，最大值1000

请求代码示例

提示一：使用示例代码前，请记得替换其中的示例Token、图片地址或Base64信息。

提示二：部分语言依赖的类或库，请在代码注释中查看下载地址。

{% Face-API-GroupGetusers %}

返回说明

返回参数

字段	必选	类型	说明
log_id	是	uint64	请求标识码，随机数，唯一
result_num	是	uint32	返回个数
result	是	array(object)	user列表
+uid	是	string	用户id
+user_info	是	string	用户信息

返回示例

```
{
  "log_id": 3314921889,
  "result_num": 2,
  "result": [
    {
      "uid": "uid1",
      "user_info": "user info 1"
    },
    {
      "uid": "uid2",
      "user_info": "user info 2"
    }
  ]
}
```


组间复制用户

接口描述

用于将已经存在于人脸库中的用户**复制到一个新的组**。

> **说明**：并不是向一个指定组内添加用户，而是直接从其它组复制用户信息 如果需要注册用户，请直接使用**人脸注册接口**

请求说明

请求示例

HTTP方法：`POST`

请求URL： `https://aip.baidubce.com/rest/2.0/face/v2/faceset/group/adduser`

URL参数：

参数	值
access_token	通过API Key和Secret Key获取的access_token,参考“Access Token获取”

Header如下：

参数	值
Content-Type	application/x-www-form-urlencoded

Body中放置请求参数，参数详情如下：

请求参数

参数	必选	类型	说明
group_id	是	string	需要添加信息的组id，多个的话用逗号分隔
uid	是	string	用户id
src_group_id	是	string	从指定group里复制信息

请求代码示例

提示一：使用示例代码前，请记得替换其中的示例Token、图片地址或Base64信息。

提示二：部分语言依赖的类或库，请在代码注释中查看下载地址。

{% Face-API-GroupAdduser %}

返回说明

返回参数

字段	必选	类型	说明
log_id	是	uint64	请求标识码，随机数，唯一

返回示例

```
 // 正确返回值
{
   "log_id": 3314921889,
}
// 发生错误时返回值
{
  "error_code": 216100,
  "log_id": 3111284097,
  "error_msg": "already add"
}
```

组内删除用户

接口描述

用于将用户从某个组中删除，但不会删除用户在其它组的信息。

> **说明**：当用户仅属于单个分组时，本接口将返回错误，请使用**人脸删除接口**

请求说明

请求示例

HTTP方法：`POST`

请求URL： `https://aip.baidubce.com/rest/2.0/face/v2/faceset/group/deleteuser`

URL参数：

参数	值
access_token	通过API Key和Secret Key获取的access_token,参考“Access Token获取”

Header如下：

参数	值
Content-Type	application/x-www-form-urlencoded

Body中放置请求参数，参数详情如下：

请求参数

参数	必选	类型	说明
group_id	是	string	用户组id，多个的话用逗号分隔
uid	是	string	用户id

请求代码示例

提示一：使用示例代码前，请记得替换其中的示例Token、图片地址或Base64信息。

提示二：部分语言依赖的类或库，请在代码注释中查看下载地址。

{% Face-API-GroupDeleteuser %}

返回说明

返回参数

字段	必选	类型	说明
log_id	是	uint64	请求标识码，随机数，唯一

返回示例

```
 // 正确返回值
{
   "log_id": 3314921889,
}
// 发生错误时返回值
{
  "error_code": 216619,
  "log_id": 815967402,
  "error_msg": "user must be in one group at least"
}
```

错误码

请参考人脸识别错误码

身份验证

人脸识别接口分为V2和V3两个版本，本文档为V2版本接口的说明文档，请确认您在百度云后台获得的是V2版本接口权限，再来阅读本文档。

> 辨别接口版本的方法是：在百度云后台进入【应用列表】，点击【应用名称】，在【API列表】中可以看到【请求地址】，若请求地址中带有【v2】标识，则您具有的是v2权限，可以阅读本文档；若请求地址中带有【v3】标识，则您具有的是v3权限，应该去阅读v3文档。 如果您对文档内容有任何疑问，可以通过以下几种方式联系我们：

- 在百度云控制台内 提交工单，咨询问题类型请选择**人工智能服务**；
- 如有需要讨论的疑问，欢迎进入 AI社区 与其他开发者们一同交流。

能力介绍

业务能力

- **质量检测（可选）**：判断图片中是否包含人脸，以及人脸在姿态、遮挡、模糊、光照等方面是否符合识别条件；
- **活体检测（可选）**：基于图片中的破绽分析，判断其中的人脸是否为**二次翻拍**（举例：如用户A用手机拍摄了一张包含人脸的图片一，用户B翻拍了图片一得到了图片二，并用图片二伪造成用户A去进行识别操作，这种情况普遍发生在金融开户、实名认证等环节。）；
- **公安验证（必选）**：基于姓名和身份证号，调取公民身份证小图（源自公安系统），将当前获取的人脸图片，与此证件小图进行对比，得出比对分数，并基于此进行业务判断是否为同一人。由于公安系统小图，具有最权威的身份证明作用，故对用户本人的验证结果可信度也最为合理。

业务逻辑

- 上述三项能力为顺序串行验证，接口默认返回公安身份对比分值，质量检测和活体检测为可选项。如选择了这两项，则验证顺序为`人脸质量检测->活体检测->公安身份验证`。您也可以根据业务场景，选择这两项中的某一项或都不选择。
- **如选择了前两个环节，则有任意一个条件不通过，则整个请求流程终止，并返回错误码，描述具体的不符合信息**。
- 基于此顺序串行验证逻辑，可以避免大量不符合条件的请求流转到公安验证，节约您的成本。

推荐阈值

- 此接口使用的对比算法，针对带水纹证件照采用了专项的模型处理，可保证水纹信息的影响降到尽可能低。
- 如比对成功，最终返回的有效数据为一个**对比分值**，在0~1之间，您可以设定具体的阈值来判断是否验证通过。
- **人证相似度的推荐阈值为0.8，对应的误识率为万分之一**。

计费逻辑

- 前两个环节图片不符合校验规则，会以`error_code`形式反馈，属于正向业务判断，这两个环节的请求全部免费。真正请求到公安验证这步并返回结果，才会进行计费。价格文档

调用方式

请求URL数据格式

向API服务地址使用POST发送请求，必须在URL中带上参数`access_token`，可通过后台的API Key和Secret Key生成，具体方式请参考“Access Token获取”。

获取access_token的示例代码

{% AccessToken %}

> **注意**：`access_token`的有效期为30天，**切记需要每30天进行定期更换，或者每次请求都拉取新token**；

例如此接口，使用HTTPS POST发送：

```
https://aip.baidubce.com/rest/2.0/face/v2/person/verify?
access_token=24.f9ba9c5341b67688ab4added8bc91dec.2592000.1485570332.282335-8574074
```

POST中Body的参数，按照下方请求参数说明选择即可。

> **提示**：如果您为百度云老用户，正在使用其他非AI的服务，可以参考百度云AKSK鉴权方式发送请求，虽然**请求方式**和**鉴权方法**和本文所介绍的不同，但请求参数和返回结果一致。

请求说明

注意事项：

- **请求体格式化**：Content-Type为`application/x-www-form-urlencoded`，通过`urlencode`格式化请求体。
- **Base64编码**：请求的图片需经过`Base64编码`，图片的base64编码指将图片数据编码成一串字符串，使用该字符串代替图像地址。您可以首先得到图片的二进制，然后用Base64格式编码即可。需要注意的是，图片的base64编码是不包含图片头的，如``
- **图片格式**：现支持PNG、JPG、JPEG、BMP，**不支持GIF图片**

- **人脸识别接口分为V2和V3两个版本，本文档为V2版本接口的说明文档，请确认您在百度云后台获得的是V2版本接口权限，再来阅读本文档。**

> 辨别接口版本的方法是：在百度云后台进入【应用列表】，点击【应用名称】，在【API列表】中可以看到【请求地址】，若请求地址中带有【v2】标识，则您具有的是v2权限，可以阅读本文档；若请求地址中带有【v3】标识，则您具有的是v3权限，应该去阅读v3文档。

请求示例

HTTP方法：`POST`

请求URL： `https://aip.baidubce.com/rest/2.0/face/v2/person/verify`

URL参数：

参数	值
access_token	通过API Key和Secret Key获取的access_token,参考“Access Token获取”

Header：

参数	值
Content-Type	application/x-www-form-urlencoded

Body中放置请求参数，参数详情如下：

请求参数

参数	必选	类型	说明
image	是	string	**base64编码后的图片数据，需urlencode，编码后的图片大小不超过5M，图片尺寸不超过1920*1080**
id_card_number	是	uint32	身份证号（真实身份证号号码）。我们的服务端会做格式校验，并通过错误码返回，但是为了您的产品反馈体验更及时，建议在产品前端做一下号码格式校验与反馈
name	是	string	utf8，姓名（真实姓名，和身份证号匹配）
quality	否	string	判断图片中的人脸质量是否符合条件。use表示需要做质量控制，**质量不符合条件的照片会被直接拒绝**

quality_conf	否	string	人脸质量检测中每一项指标的具体阈值设定，json串形式，当指定quality:use时生效。**默认使用的阈值**如下：
			{
			left_eye：0.6，左眼被遮挡的比例，取值范围[0~1]，数值越大遮挡越多；
			right_eye：0.6，右眼被遮挡的比例，取值范围[0~1]，数值越大遮挡越多；
			nose：0.7，鼻子被遮挡的比例，取值范围[0~1]，数值越大遮挡越多；
			mouth：0.7，嘴巴被遮挡的比例，取值范围[0~1]，数值越大遮挡越多；
			left_check：0.8，左脸颊被遮挡的比例，取值范围[0~1]，数值越大遮挡越多；
			right_check：0.8，右脸颊被遮挡的比例，取值范围[0~1]，数值越大遮挡越多；
			chin_contour：0.6，下巴被遮挡的比例，取值范围[0~1]，数值越大遮挡越多；
			illumination：30，脸部光照的灰度值阈值，取值范围[0~255]，数值越大光照越强；
			blurdegree：0.7，人脸模糊度阈值，取值范围[0~1]，数值越大越模糊；
			}

faceliveness	否	sting	判断活体值是否达标。use表示需要做活体检测，**低于活体阈值的照片会直接拒绝**

faceliveness_conf	否	string	人脸活体检测的阈值设定，json串形式，当指定faceliveness:use时生效。**默认使用的阈值**如下：
			{
			faceliveness：0.3，
			//单帧图片活体检测阈值，取值范围[0~1]，大于阈值即可判断为活体
			}

ext_fields	否	string	可选项为faceliveness，qualities。选择具体的项，则返回参数中将会显示相应的扩展字段。 如faceliveness表示返回结果中包含活体相关内容，qualities表示返回结果中包含质量检测相关内容

返回说明

返回参数

参数	必须	类型	说明
log_id	是	uint64	日志id
result	是	float	与公安小图相似度可能性，用于验证生活照与公安小图是否为同一人，有正常分数时为[0~1]，推荐阈值0.8，超过即判断为同一人
ext_info	否	string	拓展信息json串，只有选择了ext_fields时才会返回具体信息。选择faceliveness返回具体活体分值信息，选择qualities返回人脸质量检测信息。两者可以同时选择，半角逗号分割。
+faceliveness	否	string	活体检测值，单帧图片建议阈值，小于此值则认为不是活体，超过则判断为活体
+qualities	否	string	质量检测结果
++occlusion	否	string	人脸遮挡情况
+++left_eye	否	string	左眼被遮挡的比例，取值范围[0~1]，数值越大遮挡越多；小于阈值时有返回，默认阈值0.6
+++right_eye	否	string	右眼被遮挡的比例，取值范围[0~1]，数值越大遮挡越多；小于阈值时有返回，默认阈值0.6
+++nose	否	string	鼻子被遮挡的比例，取值范围[0~1]，数值越大遮挡越多；小于阈值时有返回，默认阈值0.7
+++mouth	否	string	嘴巴被遮挡的比例，取值范围[0~1]，数值越大遮挡越多；小于阈值时有返回，默认阈值0.7
+++left_cheek	否	string	左脸颊被遮挡的比例，取值范围[0~1]，数值越大遮挡越多；小于阈值时有返回，默认阈值0.8
+++right_cheek	否	string	右脸颊被遮挡的比例，取值范围[0~1]，数值越大遮挡越多；小于阈值时有返回，默认阈值0.8
+++chin	否	string	下巴被遮挡的比例，取值范围[0~1]，数值越大遮挡越多；小于阈值时有返回，默认阈值0.8
++blur	否	string	人脸模糊度阈值，取值范围[0~1]，数值越大越模糊；小于阈值时有返回，默认阈值0.7
++illumination	否	string	脸部光照的灰度值阈值，取值范围[0~255]，数值越大光照越强；大于阈值时有返回，默认阈值30
++completeness	否	string	人脸完整度，0或1，0为人脸溢出图像边界，1为人脸都在图像边界内

返回示例

```
{
  "result":0.03419,
  "ext_info":{
    "faceliveness":0.47
  },
  "log_id":772889134072410
}
```

错误码

请参考人脸识别错误码

在线活体检测

人脸识别接口分为V2和V3两个版本，本文档为V2版本接口的说明文档，请确认您在百度云后台获得的是V2版本接口权限，再来阅读本文档。

> 辨别接口版本的方法是：在百度云后台进入【应用列表】，点击【应用名称】，在【API列表】中可以看到【请求地址】，若请求地址中带有【v2】标识，则您具有的是v2权限，可以阅读本文档；若请求地址中带有【v3】标识，则您具有的是v3权限，应该去阅读v3文档。 如果您对文档内容有任何疑问，可以通过以下几种方式联系我们：

- 在百度云控制台内 提交工单，咨询问题类型请选择**人工智能服务**；
- 如有需要讨论的疑问，欢迎进入 AI社区 与其他开发者们一同交流。

能力介绍

接口能力

- **人脸基础信息**：包括人脸框位置，人脸空间旋转角度，人脸置信度等信息。
- **人脸质量检测**：判断人脸的遮挡、光照、模糊度、完整度等质量信息。可用于判断上传的人脸是否符合标准。
- **基于图片的活体检测**：基于单张图片，判断图片中的人脸是否为二次翻拍（举例：如用户A用手机拍摄了一张包含人脸的图片一，用户B翻拍了图片一得到了图片二，并用图片二伪造成用户A去进行识别操作，这种情况普遍发生在金融开户、实名认证等环节）。此能力可用于H5场景下的一些人脸采集场景中，增加人脸注册的安全性和真实性。

调用方式

请求URL数据格式

向API服务地址使用POST发送请求，必须在URL中带上参数`access_token`，可通过后台的API Key和Secret Key生成，具体方式请参考“Access Token获取”。

示例代码

Bash
PHP
JAVA
Python
Cpp
C#
Node

```
##### !/bin/bash
curl -i -k 'https://aip.baidubce.com/oauth/2.0/token?grant_type=client_credentials&client_id=【百度云应用的AK】&client_secret=【百度云应用的SK】'
```

> **注意**：access_token的有效期为30天，**切记需要每30天进行定期更换，或者每次请求都拉取新token**；

例如此接口，使用HTTPS POST发送：

```
https://aip.baidubce.com/rest/2.0/face/v2/faceverify?access_token=24.f9ba9c5341b67688ab5added8bc91dec.2592000.1485570332.282335-8574075
```

POST中Body的参数，按照下方请求参数说明选择即可。

> **提示**：如果您为百度云老用户，正在使用其他非AI的服务，可以参考百度云AKSK鉴权方式发送请求，虽然**请求方式**和**鉴权方法**和本文所介绍的不同，但请求参数和返回结果一致。

请求说明

注意事项：

- **请求体格式化**：Content-Type为application/x-www-form-urlencoded，通过urlencode格式化请求体。
- **Base64编码**：请求的图片需经过Base64编码，图片的base64编码指将图片数据编码成一串字符串，使用该字符串代替图像地址。您可以首先得到图片的二进制，然后用Base64格式编码即可。需要注意的是，图片的base64编码是不包含图片头的，如
- **图片格式**：现支持PNG、JPG、JPEG、BMP，**不支持GIF图片**
- **人脸识别接口分为V2和V3两个版本，本文档为V2版本接口的说明文档，请确认您在百度云后台获得的是V2版本接口权限，再来阅读本文档。**

> 辨别接口版本的方法是：在百度云后台进入【应用列表】，点击【应用名称】，在【API列表】中可以看到【请求地址】，若请求地址中带有【v2】标识，则您具有的是v2权限，可以阅读本文档；若请求地址中带有【v3】标识，则您具有的是v3权限，应该去阅读v3文档。

请求示例

HTTP方法：POST

请求URL： `https://aip.baidubce.com/rest/2.0/face/v2/faceverify`

URL参数：

参数	值
access_token	通过API Key和Secret Key获取的access_token,参考“Access Token获取”

Header：

参数	值
Content-Type	application/x-www-form-urlencoded

Body中放置请求参数，参数详情如下：

请求参数

参数	是否必选	类型	说明
image	是	string	**base64编码后的图片数据，需urlencode，编码后的图片大小不超过2M**
max_face_num	否	uint32	最多处理人脸数目，默认值1
face_fields	否	string	如不选择此项，返回结果默认只有人脸框、概率和旋转角度。可选参数为qualities、faceliveness。**qualities**：图片质量相关判断；**faceliveness**：活体判断。如果两个参数都需要选择，请使用半角逗号分隔。

返回参数

参数	类型	是否必须	说明
log_id	uint64	是	日志id
result_num	uint32	是	人脸数目
result	object[]	是	人脸属性对象的集合
+faceliveness	float	否	活体分数，face_fields包括qualities时返回
+location	bject	是	人脸在图片中的位置
++left	uint32	是	人脸区域离左边界的距离
++top	uint32	是	人脸区域离上边界的距离
++width	uint32	是	人脸区域的宽度
++height	uint32	是	人脸区域的高度
+face_probability	double	是	人脸置信度，范围0-1
+rotation_angle	int32	是	人脸框相对于竖直方向的顺时针旋转角，[-180,180]
+yaw	double	是	三维旋转之左右旋转角[-90(左), 90(右)]
+pitch	double	是	三维旋转之俯仰角度[-90(上), 90(下)]
+roll	double	是	平面内旋转角[-180(逆时针), 180(顺时针)]
+qualities	object	否	人脸质量信息。face_fields包含qualities时返回
++occlusion	object	是	人脸各部分遮挡的概率，区间[0, 1]
+++left_eye	double	是	左眼
+++right_eye	double	是	右眼
+++nose	double	是	鼻子
+++mouth	double	是	嘴
+++left_cheek	double	是	左脸颊
+++right_cheek	double	是	右脸颊
+++chin	double	是	下巴
++blur	double	是	人脸模糊程度，[0, 1]。0表示清晰，1表示模糊
++illumination	double	是	取值范围在[0,255],表示脸部区域的光照程度
++completeness	double	是	人脸完整度，[0, 1]。0表示完整，1表示不完整
++type	object	是	真实人脸/卡通人脸置信度
+++human	double	是	真实人脸置信度，[0, 1]
+++cartoon	double	是	卡通人脸置信度，[0, 1]

返回示例

```
{
  log_id: 1900901488032821,
  result_num: 1,
  result: [
    {
      rotation_angle: 10,
      yaw: 11.357421875,
      faceliveness: 0.06,
      location: {
        width: 96,
        top: 73,
        height: 96,
        left: 283
      },
      qualities: {
        illumination: 211,
        occlusion: {
          right_eye: 0,
          left_eye: 0.039751552045345,
          left_cheek: 0.12623985111713,
          mouth: 0,
          nose: 0,
          chin: 0.037661049515009,
          right_cheek: 0.024720622226596
        },
        completeness: 1,
        type: {
          cartoon: 0,
          human: 0
        },
        blur: 2.5251445032182e-11
      },
      pitch: 1.0063140392303,
      roll: 12.760620117188,
      face_probability: 1
    }
  ]
}
```


活体阈值参考

请务必在产品侧做好以下条件限制

- 检测的图片为二次采集，即通过相机当场拍摄，确保时间及操作条件的约束；
- SDK输出的多帧情况，只要这些帧中，**任何一张通过了阈值，即可判断为活体，建议可用三帧情况**；
- 推荐分值采用**99.5%**

关于活体检测faceliveness的判断阈值选择，可参考以下数值信息：

阈值（Threshold）	**误拒率（FRR）**	**通过率（TAR）**	**攻击拒绝率（TRR）**
0.05	0.01%	99.99%	97.75%
0.1	0.05%	99.95%	98.33%
0.3 （**推荐**）	0.1%	99.9%	98.82%
0.5	0.5%	99.5%	99.67%
0.9	1%	99%	99.77%

关于以上数值的概念介绍：

- **拒绝率（TRR）**：如99%，代表100次作弊假体攻击，会有99次被拒绝。
- **误拒率（FRR）**：如0.5%，指1000次真人请求，会有5次因为活体分数低于阈值被错误拒绝。
- **通过率（TAR）**：如99%，指100次真人请求，会有99次因为活体分数高于阈值而通过。
- **阈值（Threshold）**：高于此数值，则可判断为活体。

质量检测参考

指标	字段与解释	推荐数值界限
遮挡范围	**occlusion**，取值范围[0~1]，0为无遮挡，1是完全遮挡 含有多个具体子字段，表示脸部多个部位 通常用作判断头发、墨镜、口罩等遮挡	left_eye：0.6, #左眼被遮挡的阈值 right_eye：0.6, #右眼被遮挡的阈值 nose：0.7, #鼻子被遮挡的阈值 mouth：0.7, #嘴巴被遮挡的阈值 left_check：0.8, #左脸颊被遮挡的阈值 right_check：0.8, #右脸颊被遮挡的阈值 chin_contour：0.6, #下巴被遮挡阈值
模糊度范围	**blur**，取值范围[0~1]，0是最清晰，1是最模糊	小于0.7
光照范围	**illumination**，取值范围[0~255] 脸部光照的灰度值，0表示光照不好 以及对应客户端SDK中，YUV的Y分量	大于40
姿态角度	**Pitch**：三维旋转之俯仰角度[-90(上), 90(下)] **Roll**：平面内旋转角[-180(逆时针), 180(顺时针)] **Yaw**：三维旋转之左右旋转角[-90(左), 90(右)]	分别小于20度
人脸完整度	**completeness**（0或1），0为人脸溢出图像边界，1为人脸都在图像边界内	视业务逻辑判断
人脸大小	人脸部分的大小 建议长宽像素值范围：80*80~200*200	人脸部分不小于**100*100**像素

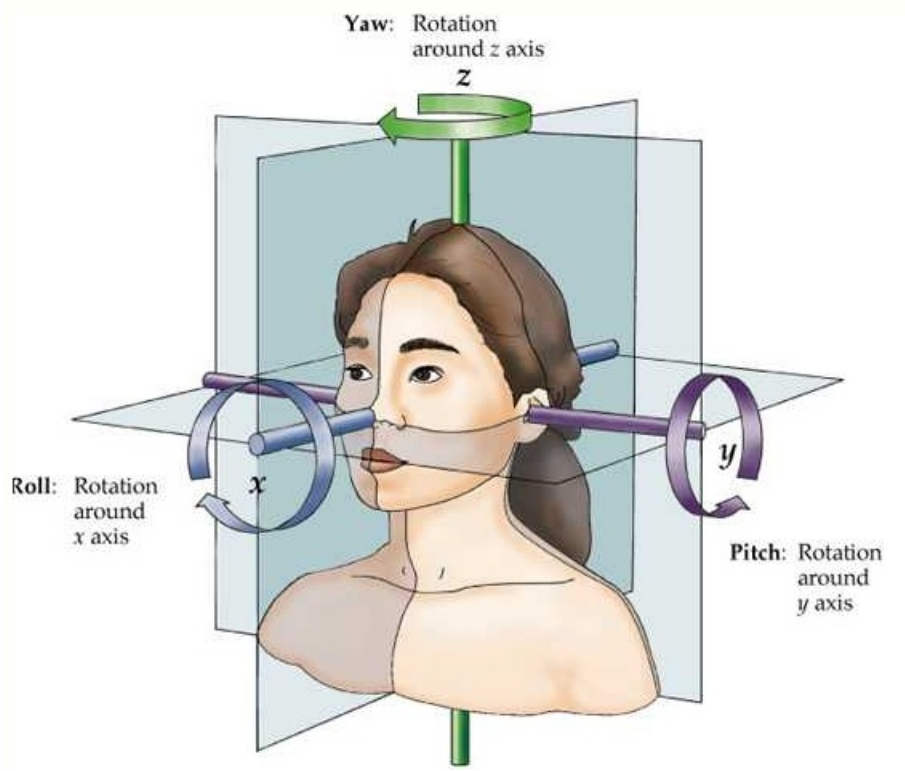
人脸空间姿态角参考

姿态角分为`Pitch`、`Roll`、`Yaw`，用于表示人脸在空间三维坐标系内的角度，常用于判断识别角度的界限值。

各角度阈值如下：

```
Pitch：三维旋转之俯仰角度，范围：[-90（上），90（下）]，推荐俯仰角绝对值不大于20度；
Roll：平面内旋转角，范围：[-180（逆时针），180（顺时针）]，推荐旋转角绝对值不大于20度；
Yaw：三维旋转之左右旋转角，范围：[-90（左），90（右）]，推荐旋转角绝对值不大于20度；
```

各角度范围示意图如下：



从姿态角度来看，**这三个值的绝对值越小越好**，这样代表人脸足够正视前方，最利于实际注册/识别使用。

但在实际应用场景中，由于摄像头的布设位置，往往无法拿到正对人脸的图片，主要分为以下几种情况：

1. **监控摄像头**：此类摄像头一般置于室内棚顶/室外架杆顶端，垂直向下倾斜一定角度，水平方向有一定的拍摄广度，一般Pitch和Yaw角度变化范围较大，所以采集到的人脸往往存在大量的俯角过大、侧脸等，导致识别效果不佳。这种情况通常是调整摄像头角度（摄像头与水平面夹角减小）、调整最小检测人脸（在行人靠近摄像头时尽可能早些拿到人脸）、增加摄像头数量（不同角度互相补充，避免采集死角），实际项目实施中，还是要通过实地考察，基于现场环境一点点调节摄像头角度；
2. **闸机/门禁**：闸机方面，摄像头往往置于机器顶部，摄像头向上仰一定角度，通常低于平均成人身高，所以行人路过时，一般需要微微低头看闸机上的摄像头/屏幕，如果摆放不当，会造成采集到的人脸Pitch角度过大；门禁方面，摄像头往往置于成人身高平视高度，但多为门框侧面，如果行人朝向正门，侧面对视摄像头，如摆放不当，会造成采集的人脸Yaw角度过大。以上两种主要场景，需要基于实际场景情况动态调整安装位置，尽量避免角度过大的同时，避免用户动作不要太大。
3. **USB摄像头**：线下场景中，USB摄像头常用于近场场景录入人脸（如柜台、大屏、自助柜机等），这种情况下为了拿到角度最好的图片（用户正视屏幕），在应用UI方面做一个前置页面提示，往往能最小成本地提高操作标准度。

错误码

请参考人脸识别错误码

H5活体检测

如果您对文档内容有任何疑问，可以通过以下几种方式联系我们：

- 在百度云控制台内 提交工单，咨询问题类型请选择**人工智能服务**；
- 如有需要讨论的疑问，欢迎进入 AI社区 与其他开发者们一同交流。

能力介绍

业务能力

H5视频活体检测产品，是由两个接口组合而成，主要用于在H5场景下，通过用户新录制并上传一个视频，来进行活体检测的判断。相对于APP有动作校验+静默图片活体、静默图片活体这两种方式，H5视频活体方案比APP方案更加灵活，同时比单张图片活体方式更加安全。其主要功能如下所示：

- 语音校验码：为防止用户提交非当前操作的视频，在录制视频时，随机分配一个数字，用户需要读出这个数字，声音存于视频当中，并在后续识别时校验，以判断是否为此次会话。语音校验码作为辅助性质的验证条件，是一个可选项，如果应用场景比较嘈杂或方言口音比较重，可以不使用语音验证。
- 视频多帧活体检测：录制并上传的视频，会在云端进行随机抽帧分析，并得出最终的活体检测分数。

> 以上两项能力，分为两个接口，使用顺序为`语音校验码`->`视频多帧校验`，具体调用逻辑可以参考我们的。Demo体验地址（电脑端用谷歌、火狐浏览器）：
> Mobile：https://ai.baidu.com/face-verify/mobile.html
> PC：https://ai.baidu.com/face-verify/pc.html

主要适用场景

- 微信服务号：用于对操作用户真实性要求严格的场景，用于依托于微信服务号的金融开户、实名认证、账户信息变更二次验证等服务。
- APP内Webview：对于如Cordova架构开发的APP，或者APP内变更频繁的身份信息页等情况，可以采用此方案完成活体检测。
- 浏览器：如果仅是一个H5宣传页，或者Wap版本网页等，可以通过此方法快速集成更加安全的活体检测功能。

此方案的优劣势

- 优势：相对于APP有动作校验、单张图片静默判断，此方法在没有APP情况下，可以更快速、轻量级地实现活体检测，同时保障一定安全性。
- 劣势：由于视频较大，上传时间可能较长，另由于不同手机的浏览器内核差异较大，容易出现兼容性问题。

调用方式

请求URL数据格式

向API服务地址使用POST发送请求，必须在URL中带上参数`access_token`，可通过后台的API Key和Secret Key生成，具体方式请参考“Access Token获取”。

获取access_token的示例代码

{% AccessToken %}

> **注意**：`access_token`的有效期为30天，**切记需要每30天进行定期更换，或者每次请求都拉取新token**；

例如此接口，使用HTTPS POST发送：

```
https://aip.baidubce.com/rest/2.0/face/v2/identify?
access_token=24.f9ba9c5341b67688ab4added8bc91dec.2592000.1485570332.282335-8574074
```

POST中Body的参数，按照下方请求参数说明选择即可。

语音校验码接口

接口描述

此接口主要用于生成随机码，用于视频的语音识别校验使用，以判断视频的即时性，而非事先录制的，提升作弊的难度。

请求说明

请求示例

HTTP方法：`POST`

请求URL： `https://aip.baidubce.com/rest/2.0/face/v1/faceliveness/sessioncode`

URL参数：

参数	值
access_token	通过API Key和Secret Key获取的access_token,参考“Access Token获取”

Header：

参数	值
Content-Type	application/x-www-form-urlencoded

Body中放置请求参数，参数详情如下：

请求参数

参数	必选	类型	说明
appid	否	string	百度云创建应用时的唯一标识ID

返回说明

返回参数

字段	必选	类型	说明
session_id	是	string	语音校验码会话id，有效期5分钟，请提示用户在五分钟内完成全部操作
code	是	string	语音验证码，数字形式，3~6位数字

返回示例

```
 {
"err_no": 0,
"err_msg": "SUCCESS",
"result": {
 "session_id": "S59faeeebb9111890355690",
 "code": "9940"
},
"timestamp": 1509617387,
"cached": 0,
"serverlogid": "0587756642"
}
```

视频活体检测接口

接口描述

此接口一方面通过语音识别得到校验码，通过session code来判断视频是否作弊。另一方面进行视频抽帧，判断是否为活体。

请求说明

请求示例

HTTP方法：`POST`

请求URL： `https://aip.baidubce.com/rest/2.0/face/v1/faceliveness/verify`

URL参数：

参数	值
access_token	通过API Key和Secret Key获取的access_token,参考“Access Token获取”

Header：

参数	值
Content-Type	application/x-www-form-urlencoded

Body中放置请求参数，参数详情如下：

请求参数

参数	必选	类型	说明
session_id	否	string	语音校验码会话id，使用此接口的前提是已经调用了语音校验码接口。 语音校验码作为辅助性质的验证条件，是一个可选项， 如果应用场景比较嘈杂或方言口音比较重，可以不使用语音验证。
video_base64	是	string	base64编码后的视频数据（视频限制：最佳为上传5-15s的mp4文件。 视频编码方式：h264编码；音频编码格式：aac，pcm均可。）

返回说明

返回参数

字段	必选	类型	说明
score	是	float	活体检测分数
thresholds	是	array	阈值参考，实际业务应用中，请以score>阈值判定通过，可直接选择不同误识别率的阈值，无需对应具体的分值，选择阈值参数即可。
code	是	array	语音校验码信息
create	是	string	生成的校验码，通过create和identify两个字段的对比，可以判断上传的视频是否为目标视频。
identify	是	string	语音识别出来的校验码
pic_list	是	array	抽取图片信息列表
pic_list[i].face_id	是	string	face唯一ID
pic_list[i].pic	是	string/encryption	base64编码后的图片信息

返回示例

```
 {

err_no:0,
err_msg: 'success',
result: {
 score: 0.984654366,
 thresholds: {
  "frr_1e-4": 0.05, //万分之一误识别率的阈值
  "frr_1e-3": 0.3,  //千分之一误识别率的阈值
  "frr_1e-2": 0.9   //百分之一误识别率的阈值
    },
    code: {
      "create": "5789",
      "identify": "5789"
    },
    pic_list: [
     {
       "face_id": 5745745747,
       "pic": "gsagaheryzxv..."
     },
     {
       "face_id": 5745745747,
       "pic": "gsagaheryzxv..."
     }
    ]
 },
  "timestamp": 1509611848,
  "cached": 0,
  "serverlogid": "2248375729"
}
```

错误码列表

错误码	错误信息	描述
216506	redis操作失败	
216505	redis连接失败	
216430	rtse/face 服务异常	
216502	当前会话已失效	
216501	没有找到人脸	
216508	没有找到视频信息	
216434	活体检测失败	
216908	视频中人脸质量过低(返回的错误信息 会包含具体的错误信息包含 illumiantion(光照不足)angle(角度) blur(人脸模糊) occlusion(遮挡)too large(人脸过大) 等原因	
216507	视频中有多张人脸	
216433	视频解析服务发生错误	
216432	视频解析服务调用失败	
216431	语音识别服务异常	
216500	验证码位数错误	

错误码

若请求错误，服务器将返回的JSON文本包含以下参数：

- **error_code**：错误码。
- **error_msg**：错误描述信息，帮助理解和解决发生的错误。

例如Access Token失效返回：

```
{
 "error_code": 110,
 "error_msg": "Access token invalid or no longer valid"
}
```

上述问题重新获取新的Access Token再次请求即可。

接口流控及鉴权错误码

错误码	错误信息	描述	处理建议
4	Open api request limit reached	集群超限额	可直接提交工单，咨询问题类型请选择**人工智能服务**；
6	no permission to access data	没有接口权限	请确认您调用的接口已经被赋权 常见问题是有V3版本权限， 调用的是v2版本接口；
17	Open api daily request limit reached	每天流量超限额，或者免费赠送的调用次数已经用完	可提交工单，咨询问题类型请选择**人工智能服务**；
18	Open api qps request limit reached	QPS超限额	可直接自助购买更多QPS、联系商务接口人、或者提交工单，咨询问题类型请选择**人工智能服务**；
19	Open api total request limit reached	请求总量超限额	可提交工单，咨询问题类型请选择**人工智能服务**；
100	Invalid parameter	无效的access_token参数	token拉取失败，可以参考“Access Token获取”重新获取
110	Access token invalid or no longer valid	Access Token失效	**token有效期为30天，注意需要定期更换**，也可以每次请求都拉取新token
111	Access token expired	Access token过期	**token有效期为30天，注意需要定期更换**，也可以每次请求都拉取新token

通用及业务错误码

错误码	错误信息	描述	处理建议
216100	invalid param	参数异常，具体异常原因详见error_msg	根据msg信息具体排查
216101	not enough param	缺少必须的参数，具体异常原因 详见error_msg	检查入参中必选参数是否都填写
216102	service not support	请求了不支持的服务	请检查调用的url
216103	param too long	请求超长	一般为一次传入图片个数超过系统限制
216110	appid not exist	appid不存在	请重新检查后台应用列表中的应用信息
216111	invalid userid	userid信息非法	请检查对应的参数
216200	empty image	图片为空或者base64解码错误	检查base64编码是否正确，base64后是否做了urlencode
216201	image format error	图片格式错误	检查base64编码是否正确，base64后是否做了urlencode
216202	image size error	图片大小错误	检查图片大小是否符合要求，注意图片大小是指base64编码后的大小
216300	db error	数据库异常	少量发生时重试即可
216400	backend error	后端识别服务异常	可以根据具体msg查看错误原因
216402	face not found	未找到人脸	请检查图片是否含有人脸
216500	unknown error	未知错误	少量发生时重试即可
216611	user not exist	用户不存在	请确认该用户是否注册或注册已经生效，注册到生效的最长延时为5s
216613	fail to delete user record	删除用户图片记录失败	重试即可
216614	not enough images	两两比对中图片数少于2张，无法比较	检查传入的图片数量
216615	fail to process images	服务处理该图片失败	重试即可
216616	image existed	图片已存在	无需重复注册
216617	fail to add user	新增用户图片失败	重试，或检查图片是否编码正确
216618	no user in group	组内用户为空	确认该group是否存在或已经生效
216631	request add user overlimit	本次请求添加的用户数量超限	检查注册的uid数量

公安验证接口错误码

错误码	错误信息	描述	处理建议
216501	face not found	传入的生活照中没有找到人脸	建议提示用户重新操作，确保人脸置于镜头前方
216600	id number format error	身份证格式错误，请检查后重新输入	建议提示用户重新仔细检查身份证号中的数字及字母信息
216601	id number and name not match or id number not exist	身份证号码与姓名不匹配，或无法找到此身份证号码	建议提示用户重新核查姓名和身份证号是否输入有误，如确实为作弊情况，可在多次提示后锁住一段时间操作。如用户确定内容正确，则可能在身份查询系统找不到
216602	face is blocked	输入生活照人脸遮挡，质量检测不通过	建议提示用户不要佩戴墨镜、口罩，避免大面积头发遮挡等
216603	face light is not good	人脸光照不好，质量检测不通过	建议提示用户不要在光线过强、逆光等情况操作，同时确保脸部光线一般明亮即可
216604	face is incomplete	人脸不完整，质量检测不通过	建议提示用户将脸部置于镜头前方，确保脸部都在画面之内
216605	人脸置信度低，质量检测不通过	-	建议提示用户检查将人脸置于镜头前方，确保光线正常
216606	face is fuzzy	人脸模糊，质量检测不通过	建议提示用户保持镜头稳定，拍摄时不要突然晃动头部
216607	police picture is none or low quality	公安库无此人图片或公安小图质量不佳	对于数据缺失，建议开设人工反馈渠道，进行人工认证处理
216608	face liveness check fail	输入的生活照活体校验不通过	建议提示用户确保人脸全部置于画面中，拍摄时不要抖动造成拖影，镜头不要逆光，周边光线不要太强，人脸部分的光线保持正常室内光即可
216609	人脸左右角度过大	-	建议提示用户人脸正视屏幕
216610	人脸俯仰角度过大	-	建议提示用户人脸正视屏幕

REST API SDK-v2

JAVA语言

简介

Hi，您好，欢迎使用百度人脸识别服务。

本文档主要针对Java开发者，描述百度人脸识别接口服务的相关技术内容。如果您对文档内容有任何疑问，可以通过以下几种方式联系我们：

- 在百度云控制台内提交工单，咨询问题类型请选择**人工智能服务**；
- 如有疑问，进入AI社区交流：http://ai.baidu.com/forum/topic/list/165

接口能力

接口名称	接口能力简要描述
人脸检测	检测人脸并定位，返回五官关键点，及人脸各属性值
人脸比对	返回两两比对的人脸相似值
人脸查找	在一个人脸集合中找到找到相似的人脸，由一系列接口组成，包括人脸识别、人脸认证、人脸库管理相关接口（人脸注册、人脸更新、人脸删除、用户信息查询、组列表查询、组内用户列表查询、组间复制用户、组内删除用户）

版本更新记录

上线日期	版本号	更新内容
2018.4.3	4.2.0	新增人脸在线活体检测、身份验证接口
2018.1.11	4.1.0	新增人脸比对M:N接口
2017.12.22	4.0.0	接口统一升级
2017.11.14	3.3.2	人脸检测接口升级v2版本
2017.10.18	3.2.1	使用proxy问题修复
2017.8.25	3.0.0	更新sdk打包方式：所有AI服务集成一个SDK
2017.7.14	1.3.6	更新SDK打包方式
2017.4.27	1.3.4	人脸比对、识别、认证和人脸库设置接口升级为v2版本
2017.4.20	1.3.3	AI SDK同步版本更新
2017.4.13	1.3.2	AI SDK同步版本更新
2017.3.23	1.3	兼容Android环境
2017.3.2	1.2	上线人脸查找接口，增加对图片参数要求限制的检查，增加设置超时接口
2017.1.20	1.1	上线人脸比对接口，同时修复部分云用户调用不成功的错误
2017.1.6	1.0	初始版本，上线人脸属性识别接口

快速入门

安装Face Java SDK

Face Java SDK目录结构

```
com.baidu.aip
    ├─── auth                    //签名相关类
    ├─── http                    //Http通信相关类
    ├─── client                  //公用类
    ├─── exception                 //exception类
    ├─── face
    │      └─── AipFace        //AipFace类
    └─── util                    //工具类
```

支持 JAVA版本：1.7+

查看源码 Java SDK代码现已公开，您可以查看代码、或者在License范围内修改和编译SDK以适配您的环境。 github链接：https://github.com/Baidu-AIP/java-sdk

使用maven依赖：

添加以下依赖即可。其中版本号可在maven官网查询

```
<dependency>
  <groupId>com.baidu.aip</groupId>
  <artifactId>java-sdk</artifactId>
  <version>${version}</version>
</dependency>
```

直接使用JAR包步骤如下：

1.在官方网站下载Java SDK压缩工具包。

2.将下载的`aip-java-sdk-version.zip`解压后，复制到工程文件夹中。

3.在Eclipse右键“工程 -> Properties -> Java Build Path -> Add JARs”。

4.添加SDK工具包`aip-java-

PHP语言

简介

Hi，您好，欢迎使用百度人脸识别服务。

本文档主要针对PHP开发者，描述百度人脸识别接口服务的相关技术内容。如果您对文档内容有任何疑问，可以通过以下几种方式联系我们：

- 在百度云控制台内提交工单，咨询问题类型请选择**人工智能服务**；
- 如有疑问，进入AI社区交流：http://ai.baidu.com/forum/topic/list/165

接口能力

接口名称	接口能力简要描述
人脸检测	检测人脸并定位，返回五官关键点，及人脸各属性值
人脸比对	返回两两比对的人脸相似值
人脸查找	在一个人脸集合中找到找到相似的人脸，由一系列接口组成，包括人脸识别、人脸认证、人脸库管理相关接口（人脸注册、人脸更新、人脸删除、用户信息查询、组列表查询、组内用户列表查询、组间复制用户、组内删除用户）

版本更新记录

上线日期	版本号	更新内容
2018.4.9	2.2.2	新增身份验证，在线活体检测接口
2018.01.12	2.1.0	新增M：N多人脸识别
2017.12.22	2.0.0	SDK代码重构
2017.5.11	1.0.0	人脸识别服务上线

快速入门

安装人脸识别 PHP SDK

人脸识别 PHP SDK目录结构

```
├─── AipFace.php          //人脸识别
└─── lib
   ├─── AipHttpClient.php      //内部http请求类
   ├─── AipBCEUtil.php         //内部工具类
   └─── AipBase               //Aip基类
```

支持PHP版本：5.3+

使用PHP SDK开发骤如下：

1.在官方网站下载php SDK压缩包。

2.将下载的`aip-php-sdk-version.zip`解压后，复制AipFace.php以及lib/*到工程文件夹中。

3.引入AipFace.php

新建AipFace

AipFace是人脸识别的PHP SDK客户端，为使用人脸识别的开发人员提供了一系列的交互方法。

参考如下代码新建一个AipFace：

```
 require_once 'AipFace.php';

// 你的 APPID AK SK
const APP_ID = '你的 App ID';
const API_KEY = '你的 Api Key';
const SECRET_KEY = '你的 Secret Key';

$client = new AipFace(APP_ID, API_KEY, SECRET_KEY);
```

在上面代码中，常量`APP_ID`在百度云控制台中创建，常量`API_KEY`与`SECRET_KEY`是在创建完毕应用后，系统分配给用户的，均为字符串，用于标识用户，为访问做签名验证，可在AI服务控制台中的**应用列表**中查看。

注意：如您以前是百度云的老用户，其中`API_KEY`对应百度云的“Access Key ID”，`SECRET_KEY`对应百度云的“Access Key Secret”。

配置AipFace

如果用户需要配置AipFace的网络请求参数(一般不需要配置)，可以在构造AipFace之后调用接口设置参数，目前只支持以下参数：

接口	说明
setConnectionTimeoutInMillis	建立连接的超时时间（单位：毫秒)
setSocketTimeoutInMillis	通过打开的连接传输数据的超时时间（单位：毫秒）

接口说明

人脸检测

检测请求图片中的人脸，返回人脸位置、72个关键点坐标、及人脸相关属性信息。

检测响应速度，与图片中人脸数量相关，人脸数量较多时响应时间会有些许延长。

典型应用场景：如**人脸属性分析**，**基于人脸关键点的加工分析**，**人脸营销活动**等。

> 五官位置会标记具体坐标；72个关键点坐标也包含具体坐标，但不包含对应位置的详细位置描述。

```
  $image = file_get_contents('example.jpg');

// 调用人脸检测
$client->detect($image);

// 如果有可选参数
$options = array();
$options["max_face_num"] = 2;
$options["face_fields"] = "age";

// 带参数调用人脸检测
$client->detect($image, $options);
```

人脸检测 请求参数详情

参数名称	是否必选	类型	默认值	说明
image	是	string		图像数据，base64编码，要求base64编码后大小不超过4M，最短边至少15px，最长边最大4096px,支持jpg/png/bmp格式
max_face_num	否	string	1	最多处理人脸数目，默认值1
face_fields	否	string		包括age,beauty,expression,faceshape,gender,glasses,landmark,qualities信息，逗号分隔，默认只返回人脸框、概率和旋转角度

人脸检测 返回数据参数详情

参数	类型	必选	说明
log_id	uint64	是	日志id
result_num	uint32	是	人脸数目
result	object[]	是	人脸属性对象的集合
+age	double	否	年龄。face_fields包含age时返回
+beauty	double	否	美丑打分，范围0-100，越大表示越美。face_fields包含beauty时返回
+location	object	是	人脸在图片中的位置
++left	uint32	是	人脸区域离左边界的距离
++top	uint32	是	人脸区域离上边界的距离
++width	uint32	是	人脸区域的宽度
++height	uint32	是	人脸区域的高度
+face_probability	double	是	人脸置信度，范围0-1
+rotation_angle	int32	是	人脸框相对于竖直方向的顺时针旋转角，[-180,180]
+yaw	double	是	三维旋转之左右旋转角[-90(左), 90(右)]
+pitch	double	是	三维旋转之俯仰角度[-90(上), 90(下)]
+roll	double	是	平面内旋转角[-180(逆时针), 180(顺时针)]
+expression	uint32	否	表情，0，不笑；1，微笑；2，大笑。face_fields包含expression时返回
+expression_probability	double	否	表情置信度，范围0~1。face_fields包含expression时返回
+faceshape	object[]	否	脸型置信度。face_fields包含faceshape时返回
++type	string	是	脸型：square/triangle/oval/heart/round

++probability	double	是	置信度：0~1
+gender	string	否	male、female。face_fields包含gender时返回
+gender_probability	double	否	性别置信度，范围0~1。face_fields包含gender时返回
+glasses	uint32	否	是否带眼镜，0-无眼镜，1-普通眼镜，2-墨镜。face_fields包含glasses时返回
+glasses_probability	double	否	眼镜置信度，范围0~1。face_fields包含glasses时返回
+landmark	object[]	否	4个关键点位置，左眼中心、右眼中心、鼻尖、嘴中心。face_fields包含landmark时返回
++x	uint32	否	x坐标
++y	uint32	否	y坐标
+landmark72	object[]	否	72个特征点位置，示例图 。face_fields包含landmark时返回
++x	uint32	否	x坐标
++y	uint32	否	y坐标
+qualities	object	否	人脸质量信息。face_fields包含qualities时返回
++occlusion	object	是	人脸各部分遮挡的概率,[0, 1],0表示完整，1表示不完整
+++left_eye	double	是	左眼
+++right_eye	double	是	右眼
+++nose	double	是	鼻子
+++mouth	double	是	嘴
+++left_cheek	double	是	左脸颊
+++right_cheek	double	是	右脸颊
+++chin	double	是	下巴
++blur	double	是	人脸模糊程度，[0, 1]。0表示清晰，1表示模糊
++illumination	-	是	取值范围在[0,255],表示脸部区域的光照程度
++completeness	-	是	人脸完整度，[0, 1]。0表示完整，1表示不完整
++type	object	是	真实人脸/卡通人脸置信度
+++human	-	是	真实人脸置信度，[0, 1]
+++cartoon	-	是	卡通人脸置信度，[0, 1]

人脸检测 返回示例

```
{
  "result_num": 1,
  "result": [
    {
      "location": {
        "left": 117,
        "top": 131,
        "width": 172,
        "height": 170
      },
      "face_probability": 1,
      "rotation_angle": 2,
      "yaw": -0.34859421849251,
      "pitch": 2.3033397197723,
      "roll": 1.9135693311691,
      "landmark": [
        {
          "x": 161.74819946289,
          "y": 163.30244445801
        },
        ...
      ],
      "landmark72": [
        {
          "x": 115.86531066895,
          "y": 170.0546875
        } ,
        ...
      ],
      "age": 29.298097610474,
      "beauty": 55.128883361816,
      "expression": 1,
      "expression_probablity": 0.5543018579483,
      "gender": "male",
      "gender_probability": 0.99979132413864,
      "glasses": 0,
      "glasses_probability": 0.99999964237213,
      "qualities": {
        "occlusion": {
          "left_eye": 0,
          "right_eye": 0,
          "nose": 0,
          "mouth": 0,
          "left_cheek": 0.0064102564938366,
          "right_cheek": 0.0057411273010075,
          "chin": 0
        },
        "blur": 1.1886881756684e-10,
        "illumination": 141,
        "completeness": 1,
        "type": {
          "human": 0.99935841560364,
          "cartoon": 0.00064159056637436
        }
      }
    }
  ],
  "log_id": 2493878179101621
}
```


质量判断

可通过人脸检测接口，基于以下字段和对应阈值，进行质量检测的判断，以保证人脸质量符合后续业务操作要求。

指标	字段与解释	推荐数值界限
遮挡范围	**occlusion**（0~1），0为无遮挡，1是完全遮挡 含有多个具体子字段，表示脸部多个部位 通常用作判断头发、墨镜、口罩等遮挡	left_eye：0.6, #左眼被遮挡的阈值 right_eye：0.6, #右眼被遮挡的阈值 nose：0.7, #鼻子被遮挡的阈值 mouth：0.7, #嘴巴被遮挡的阈值 left_check：0.8, #左脸颊被遮挡的阈值 right_check：0.8, #右脸颊被遮挡的阈值 chin_contour：0.6, #下巴被遮挡阈值
模糊度范围	**Blur**（0~1），0是最清晰，1是最模糊	小于0.7
光照范围	**illumination**（0~255） 脸部光照的灰度值，0表示光照不好 以及对应客户端SDK中，YUV的Y分量	大于40
姿态角度	**Pitch**：三维旋转之俯仰角度[-90(上), 90(下)] **Roll**：平面内旋转角[-180(逆时针), 180(顺时针)] **Yaw**：三维旋转之左右旋转角[-90(左), 90(右)]	分别小于20度
人脸完整度	**completeness**（0或1），0代表完整，1代表不完整	小于0.4
人脸大小	人脸部分的大小 建议长宽像素值范围：80*80~200*200	人脸部分不小于**100*100**像素

人脸比对

该请求用于比对多张图片中的人脸相似度并返回两两比对的得分，可用于判断两张脸是否是同一人的可能性大小。

典型应用场景：如**人证合一验证**，**用户认证**等，可与您现有的人脸库进行比对验证。

> **说明：**支持对比对的两张图片做在线活体检测

```
 $images = array(
   file_get_contents('example0.jpg'),
   file_get_contents('example1.jpg'),
);

// 调用人脸比对
$client->match($images);

// 如果有可选参数
$options = array();
$options["ext_fields"] = "qualities";
$options["image_liveness"] = ",faceliveness";
$options["types"] = "7,13";

// 带参数调用人脸比对
$client->match($images, $options);
```

人脸比对 请求参数详情

参数名称	是否必选	类型	可选值范围	说明
images	是	string		base64编码后的多张图片数据，半角逗号分隔，单次请求总共最大20M
ext_fields	否	string		返回质量信息，取值固定:目前支持qualities(质量检测)。(对所有图片都会做改处理)
image_liveness	否	string	faceliveness,faceliveness - 对比对的两张图片都做活体检测 ,faceliveness - 对第一张图片不做活体检测、第二张图做活体检测 faceliveness, - 对第一张图片做活体检测、第二张图不做活体检测	返回的活体信息，“faceliveness,faceliveness” 表示对比对的两张图片都做活体检测；“,faceliveness” 表示对第一张图片不做活体检测、第二张图做活体检测；“faceliveness,” 表示对第一张图片做活体检测、第二张图不做活体检测； **注：需要用于判断活体的图片，图片中的人脸像素面积需要不小于100px*100px，人脸长宽与图片长宽比例，不小于1/3**
types	否	string		请求对比的两张图片的类型，示例：“7,13” **12**表示带水印证件照：一般为带水印的小图，如公安网小图 **7**表示生活照：通常为手机、相机拍摄的人像图片、或从网络获取的人像图片等 **13**表示证件照片：如拍摄的身份证、工卡、护照、学生证等证件图片，**注**：需要确保人脸部分不可太小，通常为100px*100px

人脸比对 返回数据参数详情

字段	必选	类型	说明
log_id	是	uint64	请求唯一标识码，随机数
result_num	是	uint32	返回结果数目，即：result数组中元素个数
result	是	array(object)	结果数据，index和请求图片index对应。数组元素为每张图片的匹配得分数组，top n。 得分[0,100.0]
+index_i	是	uint32	比对图片1的index
+index_j	是	uint32	比对图片2的index
+score	是	double	比对得分
ext_info	否	array（dict）	对应参数中的ext_fields
+qualities	否	string	质量相关的信息，无特殊需求可以不使用
+faceliveness	否	string	活体分数，如0.49999。单帧活体检测参考阈值**0.393241**，超过此分值以上则可认为是活体。**注意：活体检测接口主要用于判断是否为二次翻拍，需要限制用户为当场拍照获取图片**；推荐配合客户端SDK有动作校验活体使用

人脸比对 返回示例

```
 //请求两张图片
{
    "log_id": 73473737,
    "result_num":1,
    "result": [
        {
            "index_i": 0,
            "index_j": 1,
            "score": 44.3
        }
    ]
}
```

人脸识别

用于计算指定组内用户，与上传图像中人脸的相似度。识别前提为您已经创建了一个**人脸库**。

典型应用场景：如**人脸闸机**，**考勤签到**，**安防监控**等。

> **说明：**人脸识别返回值不直接判断是否是同一人，只返回用户信息及相似度分值。

> **说明：**推荐可判断为同一人的相似度分值为**80**，您也可以根据业务需求选择更合适的阈值。

```
 $groupId = "group1,group2";

$image = file_get_contents('example.jpg');

// 调用人脸识别
$client->identifyUser($groupId, $image);

// 如果有可选参数
$options = array();
$options["ext_fields"] = "faceliveness";
$options["user_top_num"] = 3;

// 带参数调用人脸识别
$client->identifyUser($groupId, $image, $options);
```

人脸识别 请求参数详情

参数名称	是否必选	类型	默认值	说明
group_id	是	string		用户组id，标识一组用户（由数字、字母、下划线组成），长度限制128B。如果需要将一个uid注册到多个group下，group_id需要用多个逗号分隔，每个group_id长度限制为48个英文字符。**注：group无需单独创建，注册用户时则会自动创建group。** **产品建议**：根据您的业务需求，可以将需要注册的用户，按照业务划分，分配到不同的group下，例如按照会员手机尾号作为groupid，用于刷脸支付、会员计费消费等，这样可以尽可能控制每个group下的用户数与人脸数，提升检索的准确率
image	是	string		图像数据，base64编码，要求base64编码后大小不超过4M，最短边至少15px，最长边最大4096px,支持jpg/png/bmp格式
ext_fields	否	string		特殊返回信息，多个用逗号分隔，取值固定: 目前支持faceliveness(活体检测)。**注：需要用于判断活体的图片，图片中的人脸像素面积需要不小于100px*100px，人脸长宽与图片长宽比例，不小于1/3**
user_top_num	否	string	1	返回用户top数，默认为1，最多返回5个

人脸识别 返回数据参数详情

字段	必选	类型	说明
log_id	是	uint64	请求唯一标识码，随机数
result_num	是	uint32	返回结果数目，即：result数组中元素个数
ext_info	否	array	对应参数中的ext_fields
+faceliveness	否	string	活体分数，如0.49999。单帧活体检测参考阈值0.393241，超过此分值以上则可认为是活体。**注意：活体检测接口主要用于判断是否为二次翻拍，需要限制用户为当场拍照获取图片**；推荐配合客户端SDK有动作校验活体使用
result	是	array(object)	结果数组
+group_id	是	string	对应的这个用户的group_id
+uid	是	string	匹配到的用户id
+user_info	是	string	注册时的用户信息
+scores	是	array(double)	结果数组，数组元素为匹配得分，top n。得分[0,100.0]

人脸识别 返回示例

```
{
  "log_id": 73473737,
  "result_num":1,
  "result": [
    {
      "group_id" : "test1",
      "uid": "u333333",
      "user_info": "Test User",
      "scores": [
            99.3,
            83.4
      ]
    }
  ]
}
```

人脸认证

用于识别上传的图片是否为指定用户，即查找前需要先确定要查找的用户在人脸库中的id。

典型应用场景：如**人脸登录**，**人脸签到**等

> **说明：**人脸认证与人脸识别的差别在于：人脸识别需要指定一个待查找的人脸库中的组；而人脸认证需要指定具体的用户id即可，不需要指定具体的人脸库中的组；实际应用中，人脸认证需要用户或系统先输入id，这增加了验证安全度，但也增加了复杂度，具体使用哪个接口需要视您的业务场景判断。

> **说明：**请求参数中，新增在线活体检测

```
$uid = "user1";

$groupId = "group1,group2";

$image = file_get_contents('example.jpg');

// 调用人脸认证
$client->verifyUser($uid, $groupId, $image);

// 如果有可选参数
$options = array();
$options["top_num"] = 3;
$options["ext_fields"] = "faceliveness";

// 带参数调用人脸认证
$client->verifyUser($uid, $groupId, $image, $options);
```

人脸认证 请求参数详情

参数名称	是否必选	类型	默认值	说明
uid	是	string		用户id（由数字、字母、下划线组成），长度限制128B
group_id	是	string		用户组id，标识一组用户（由数字、字母、下划线组成），长度限制128B。如果需要将一个uid注册到多个group下，group_id需要用多个逗号分隔，每个group_id长度限制为48个英文字符。**注：group无需单独创建，注册用户时则会自动创建group。** **产品建议**：根据您的业务需求，可以将需要注册的用户，按照业务划分，分配到不同的group下，例如按照会员手机尾号作为groupid，用于刷脸支付、会员计费消费等，这样可以尽可能控制每个group下的用户数与人脸数，提升检索的准确率
image	是	string		图像数据，base64编码，要求base64编码后大小不超过4M，最短边至少15px，最长边最大4096px,支持jpg/png/bmp格式
top_num	否	string	1	返回用户top数，默认为1
ext_fields	否	string		特殊返回信息，多个用逗号分隔，取值固定: 目前支持faceliveness(活体检测)。**注：需要用于判断活体的图片，图片中的人脸像素面积需要不小于100px*100px，人脸长宽与图片长宽比例，不小于1/3**

人脸认证 返回数据参数详情

字段	必选	类型	说明
log_id	是	uint64	请求唯一标识码，随机数
result_num	是	uint32	返回结果数目，即：result数组中元素个数
result	是	array(double)	结果数组，数组元素为匹配得分，top n。 得分范围[0,100.0]。推荐得分超过80可认为认证成功
ext_info	否	array	对应参数中的ext_fields
+faceliveness	否	string	活体分数，如0.49999。单帧活体检测参考阈值0.393241，超过此分值以上则可认为是活体。**活体检测接口主要用于判断是否为二次翻拍，需要限制用户为当场拍照获取图片**；推荐配合客户端SDK有动作校验活体使用

人脸认证 返回示例

```
{
  "log_id": 73473737,
  "result_num":2,
  "result": [
        99.3,
        83.6
  ]
}
```

M:N 识别

待识别的图片中，存在多张人脸的情况下，支持在一个人脸库中，一次请求，同时返回图片中所有人脸的识别结果。

```
 $groupId = "group1,group2";

$image = file_get_contents('example.jpg');

// 调用M:N 识别
$client->multiIdentify($groupId, $image);

// 如果有可选参数
$options = array();
$options["ext_fields"] = "faceliveness";
$options["detect_top_num"] = 3;
$options["user_top_num"] = 2;

// 带参数调用M:N 识别
$client->multiIdentify($groupId, $image, $options);
```

M:N 识别 请求参数详情

参数名称	是否必选	类型	默认值	说明
group_id	是	string		用户组id，标识一组用户（由数字、字母、下划线组成），长度限制128B。如果需要将一个uid注册到多个group下，group_id需要用多个逗号分隔，每个group_id长度限制为48个英文字符。**注：group无需单独创建，注册用户时则会自动创建group。** **产品建议**：根据您的业务需求，可以将需要注册的用户，按照业务划分，分配到不同的group下，例如按照会员手机尾号作为groupid，用于刷脸支付、会员计费消费等，这样可以尽可能控制每个group下的用户数与人脸数，提升检索的准确率
image	是	string		图像数据，base64编码，要求base64编码后大小不超过4M，最短边至少15px，最长边最大4096px,支持jpg/png/bmp格式
ext_fields	否	string		特殊返回信息，多个用逗号分隔，取值固定: 目前支持faceliveness(活体检测)。**注：需要用于判断活体的图片，图片中的人脸像素面积需要不小于100px*100px，人脸长宽与图片长宽比例，不小于1/3**
detect_top_num	否	string	1	检测多少个人脸进行比对，默认值1（最对返回10个）
user_top_num	否	string	1	返回识别结果top人数”，当同一个人有多张图片时，只返回比对最高的1个分数（即，scores参数只有一个值），默认为1（最多返回20个）

M:N 识别 返回数据参数详情

参数	字段	必选	类型	说明
log_id	-	是	uint32	请求标识码，随机数，唯一
result_num	-	是	float	返回结果数目，即：result数组中元素个数（PS：最终返回的个数是小于等于 “实际检测到的人脸数” * “每个人脸匹配的top人数”）
ext_info	-	否	array	对应参数中的ext_fields
-	faceliveness	否	string	活体检测分数，单帧活体检测参考阈值**0.393241**，超过此分值以上则可认为是活体。**注意：活体检测接口主要用于判断是否为二次翻拍，需要限制用户为当场拍照获取图片**；推荐配合客户端SDK有动作校验活体使用
result	-	是	array(object)	-
-	group_id	是	string	对应的这个用户的group_id
-	uid	是	string	匹配到的用户id
-	user_info	是	string	注册时的用户信息
-	scores	是	array(double)	结果数组，数组元素为匹配得分，得分[0,100.0]；个数取决于user_top_num的设定。推荐**80分**以上即可判断为同一人
-	position	是	object	人脸位置，如{top:111,left:222,width:333,height:444,degree:20}

M:N 识别 返回示例


```
{
  "log_id": 73473737,
  "result_num":1,
  "result": [
     {
        "group_id" : "test1",
        "uid": "u333333",
        "user_info": "Test User",
        "position":{"left":726.99188232422,"top":288.37701416016,"width":44,"height":42,"degree":-
4,"prob":0.91117089986801},
        "scores": [
              99.3
        ]
     },
     {
        "group_id" : "test1",
        "uid": "u2222222",
        "user_info": "Test User",
        "position":{"left":726.99188232422,"top":288.37701416016,"width":44,"height":42,"degree":-
4,"prob":0.91117089986801},
        "scores": [
              82.3
        ]
     }
  ]
}
```


人脸注册

用于从人脸库中新增用户，可以设定多个用户所在组，及组内用户的人脸图片，

典型应用场景：构建您的人脸库，如**会员人脸注册**，**已有用户补全人脸信息**等。

人脸库、用户组、用户、用户下的人脸**层级关系**如下所示：

```
|- 人脸库
 |- 用户组一
   |- 用户01
     |- 人脸
   |- 用户02
     |- 人脸
     |- 人脸
     ....
    ....
 |- 用户组二
 |- 用户组三
 |- 用户组四
 ....
```

关于人脸库的设置限制

- 每个开发者账号可以创建100个appid；
- **每个appid对应一个人脸库，且不同appid之间，人脸库互不相通**；
- 每个人脸库下，可以创建多个用户组，用户组（group）数量**没有限制**；
- 每个用户组（group）下，可添加最多**无限**张人脸，**无限**个uid；
- 每个用户（uid）所能注册的最大人脸数量**没有限制**；

为了保证识别效果，请控制注册人脸的质量（通过`/detect`人脸检测接口判断），具体参数可详见下表所示：

质量判断

可通过人脸检测接口，基于以下字段和对应阈值，进行质量检测的判断，以保证人脸质量符合后续业务操作要求。

指标	字段与解释	推荐数值界限
遮挡范围	**occlusion**（0~1），0为无遮挡，1是完全遮挡 含有多个具体子字段，表示脸部多个部位 通常用作判断头发、墨镜、口罩等遮挡	left_eye：0.6, #左眼被遮挡的阈值 right_eye：0.6, #右眼被遮挡的阈值 nose：0.7, #鼻子被遮挡的阈值 mouth：0.7, #嘴巴被遮挡的阈值 left_check：0.8, #左脸颊被遮挡的阈值 right_check：0.8, #右脸颊被遮挡的阈值 chin_contour：0.6, #下巴被遮挡阈值
模糊度范围	**Blur**（0~1），0是最清晰，1是最模糊	小于0.7
光照范围	**illumination**（0~255） 脸部光照的灰度值，0表示光照不好 以及对应客户端SDK中，YUV的Y分量	大于40
姿态角度	**Pitch**：三维旋转之俯仰角度[-90(上), 90(下)] **Roll**：平面内旋转角[-180(逆时针), 180(顺时针)] **Yaw**：三维旋转之左右旋转角[-90(左), 90(右)]	分别小于20度
人脸完整度	**completeness**（0或1），0为人脸溢出图像边界，1为人脸都在图像边界内	视业务逻辑判断
人脸大小	人脸部分的大小 建议长宽像素值范围：80*80~200*200	人脸部分不小于**100*100**像素

```
 $uid = "user1";

$userInfo = "user's info";

$groupId = "group1,group2";

$image = file_get_contents('example.jpg');

// 调用人脸注册
$client->addUser($uid, $userInfo, $groupId, $image);

// 如果有可选参数
$options = array();
$options["action_type"] = "replace";

// 带参数调用人脸注册
$client->addUser($uid, $userInfo, $groupId, $image, $options);
```

人脸注册 请求参数详情

参数名称	是否必选	类型	默认值	说明
uid	是	string		用户id（由数字、字母、下划线组成），长度限制128B
user_info	是	string		用户资料，长度限制256B
group_id	是	string		用户组id，标识一组用户（由数字、字母、下划线组成），长度限制128B。如果需要将一个uid注册到多个group下，group_id需要用多个逗号分隔，每个group_id长度限制为48个英文字符。**注：group无需单独创建，注册用户时则会自动创建group。** **产品建议**：根据您的业务需求，可以将需要注册的用户，按照业务划分，分配到不同的group下，例如按照会员手机尾号作为groupid，用于刷脸支付、会员计费消费等，这样可以尽可能控制每个group下的用户数与人脸数，提升检索的准确率
image	是	string		图像base64编码，**每次仅支持单张图片，图片编码后大小不超过10M**。为保证后续识别的效果较佳，建议注册的人脸，为用户正面人脸。
action_type	否	string	append	参数包含append、replace。**如果为“replace”，则每次注册时进行替换replace（新增或更新）操作，默认为append操作**。例如：uid在库中已经存在时，对此uid重复注册时，新注册的图片默认会**追加**到该uid下，如果手动选择`action_type:replace`，则会用新图替换库中该uid下所有图片。

人脸注册 返回数据参数详情

字段	是否必选	类型	说明
log_id	是	uint64	请求唯一标识码，随机数

人脸注册 返回示例

```
 // 注册成功
{
   "log_id": 73473737,
}
// 注册发生错误
{
  "error_code": 216616,
  "log_id": 674786177,
  "error_msg": "image exist"
}
```

人脸更新

用于对人脸库中指定用户，更新其下的人脸图像。

> **说明：**针对一个uid执行更新操作，新上传的人脸图像将覆盖此uid原有所有图像。

> **说明：**执行更新操作，如果该uid不存在时，会返回错误。如果添加了action_type:replace,则不会报错，并自动注册该uid，操作结果等同注册新用户。

```
 $uid = "user1";

$userInfo = "user's info";

$groupId = "group1";

$image = file_get_contents('example.jpg');

// 调用人脸更新
$client->updateUser($uid, $userInfo, $groupId, $image);

// 如果有可选参数
$options = array();
$options["action_type"] = "replace";

// 带参数调用人脸更新
$client->updateUser($uid, $userInfo, $groupId, $image, $options);
```

人脸更新 请求参数详情

参数名称	是否必选	类型	默认值	说明
uid	是	string		用户id（由数字、字母、下划线组成），长度限制128B
user_info	是	string		用户资料，长度限制256B
group_id	是	string		更新指定groupid下uid对应的信息
image	是	string		图像数据，base64编码，要求base64编码后大小不超过4M，最短边至少15px，最长边最大4096px,支持jpg/png/bmp格式
action_type	否	string	append	目前仅支持replace，uid不存在时，不报错，会自动变为注册操作；未选择该参数时，如果uid不存在会提示错误

人脸更新 返回数据参数详情

字段	是否必选	类型	说明
log_id	是	uint64	请求唯一标识码，随机数

人脸更新 返回示例

```
 // 更新成功
{
   "log_id": 73473737,
}
// 更新发生错误
{
  "error_code": 216612,
  "log_id": 1137508902,
  "error_msg": "user not exist"
}
```

人脸删除

用于从人脸库中删除一个用户。

人脸删除注意事项：

- 删除的内容，包括用户所有图像和身份信息；
- 如果一个uid存在于多个用户组内，将会同时将从各个组中把用户删除
- 如果指定了group_id，则只删除此group下的uid相关信息

```
 $uid = "user1";

// 调用人脸删除
$client->deleteUser($uid);

// 如果有可选参数
$options = array();
$options["group_id"] = "group1";

// 带参数调用人脸删除
$client->deleteUser($uid, $options);
```

人脸删除 请求参数详情

参数名称	是否必选	类型	说明
uid	是	string	用户id（由数字、字母、下划线组成），长度限制128B
group_id	否	string	删除指定groupid下uid对应的信息

人脸删除 返回数据参数详情

字段	是否必选	类型	说明
log_id	是	uint64	请求唯一标识码，随机数

人脸删除 返回示例

```
 // 删除成功
{
    "log_id": 73473737,
}
// 删除发生错误
{
  "error_code": 216612,
  "log_id": 1137508902,
  "error_msg": "user not exist"
}
```

用户信息查询

用于查询人脸库中某用户的详细信息。

```
 $uid = "user1";

// 调用用户信息查询
$client->getUser($uid);

// 如果有可选参数
$options = array();
$options["group_id"] = "group1";

// 带参数调用用户信息查询
$client->getUser($uid, $options);
```

用户信息查询 请求参数详情

参数名称	是否必选	类型	说明
uid	是	string	用户id（由数字、字母、下划线组成），长度限制128B
group_id	否	string	选择指定group_id则只查找group列表下的uid内容，如果不指定则查找所有group下对应uid的信息

用户信息查询 返回数据参数详情

字段	是否必选	类型	说明
log_id	是	uint64	请求标识码，随机数，唯一
result	是	array(double)	结果数组
+uid	是	string	匹配到的用户id
+user_info	是	string	注册时的用户信息
+groups	是	array(string)	用户所属组列表

用户信息查询 返回示例

```
 {
“result_num” : 2
   "result": {
      [
         "uid": "testuser2",
         "user_info": "registed user info ...",
         "group_id": "grep1",
      ],
      [
         "uid": "testuser2",
         "user_info": "registed user info2 ...",
         "group_id": "grep2",
      ],
   },
   "log_id": 2979357502
}
```

组列表查询

用于查询用户组的列表。

```
 // 调用组列表查询
$client->getGroupList();

// 如果有可选参数
$options = array();
$options["start"] = 0;
$options["num"] = 50;

// 带参数调用组列表查询
$client->getGroupList(, $options);
```

组列表查询 请求参数详情

参数名称	是否必选	类型	默认值	说明
start	否	string	0	默认值0，起始序号
num	否	string	100	返回数量，默认值100，最大值1000

组列表查询 返回数据参数详情

字段	是否必选	类型	说明
result_num	是	number	返回个数
result	是	array(string)	group_id列表

组列表查询 返回示例

```
{
  "result_num": 2,
  "result": [
    "grp1",
    "grp2"
  ],
  "log_id": 3314921889
}
```

组内用户列表查询

用于查询指定用户组中的用户列表。

```
 $groupId = "group1";

// 调用组内用户列表查询
$client->getGroupUsers($groupId);

// 如果有可选参数
$options = array();
$options["start"] = 0;
$options["num"] = 50;

// 带参数调用组内用户列表查询
$client->getGroupUsers($groupId, $options);
```

组内用户列表查询 请求参数详情

参数名称	是否必选	类型	默认值	说明
group_id	是	string		用户组id（由数字、字母、下划线组成），长度限制128B
start	否	string	0	默认值0，起始序号
num	否	string	100	返回数量，默认值100，最大值1000

组内用户列表查询 返回数据参数详情

字段	是否必选	类型	说明
log_id	是	uint64	请求标识码，随机数，唯一
result_num	是	uint32	返回个数
result	是	array(object)	user列表
+uid	是	string	用户id
+user_info	是	string	用户信息

组内用户列表查询 返回示例

```
{
  "log_id": 3314921889,
  "result_num": 2,
  "result": [
    {
      "uid": "uid1",
      "user_info": "user info 1"
    },
    {
      "uid": "uid2",
      "user_info": "user info 2"
    }
  ]
}
```

组间复制用户

用于将已经存在于人脸库中的用户**复制到一个新的组**。

> **说明：**并不是向一个指定组内添加用户，而是直接从其它组复制用户信息 如果需要注册用户，请直接使用**人脸注册接口**

```
 $srcGroupId = "group1";

$groupId = "group1,group2";

$uid = "user1";

// 调用组间复制用户
$client->addGroupUser($srcGroupId, $groupId, $uid);
```

组间复制用户 请求参数详情

参数名称	是否必选	类型	说明
src_group_id	是	string	从指定group里复制信息
group_id	是	string	用户组id，标识一组用户（由数字、字母、下划线组成），长度限制128B。如果需要将一个uid注册到多个group下，group_id需要用多个逗号分隔，每个group_id长度限制为48个英文字符。**注：group无需单独创建，注册用户时则会自动创建group。** **产品建议**：根据您的业务需求，可以将需要注册的用户，按照业务划分，分配到不同的group下，例如按照会员手机尾号作为groupid，用于刷脸支付、会员计费消费等，这样可以尽可能控制每个group下的用户数与人脸数，提升检索的准确率
uid	是	string	用户id（由数字、字母、下划线组成），长度限制128B

组间复制用户 返回数据参数详情

字段	是否必选	类型	说明
log_id	是	uint64	请求标识码，随机数，唯一

组间复制用户 返回示例

```
 // 正确返回值
{
    "log_id": 3314921889,
}
// 发生错误时返回值
{
  "error_code": 216100,
  "log_id": 3111284097,
  "error_msg": "already add"
}
```

组内删除用户

用于将用户从某个组中删除，但不会删除用户在其它组的信息。

> **说明：**当用户仅属于单个分组时，本接口将返回错误，请使用**人脸删除接口**

```
 $groupId = "group1,group2";

$uid = "user1";

// 调用组内删除用户
$client->deleteGroupUser($groupId, $uid);
```

组内删除用户 请求参数详情

参数名称	是否必选	类型	说明
group_id	是	string	用户组id，标识一组用户（由数字、字母、下划线组成），长度限制128B。如果需要将一个uid注册到多个group下，group_id需要用多个逗号分隔，每个group_id长度限制为48个英文字符。**注：group无需单独创建，注册用户时则会自动创建group。** **产品建议**：根据您的业务需求，可以将需要注册的用户，按照业务划分，分配到不同的group下，例如按照会员手机尾号作为groupid，用于刷脸支付、会员计费消费等，这样可以尽可能控制每个group下的用户数与人脸数，提升检索的准确率
uid	是	string	用户id（由数字、字母、下划线组成），长度限制128B

组内删除用户 返回数据参数详情

字段	是否必选	类型	说明
log_id	是	uint64	请求标识码，随机数，唯一

组内删除用户 返回示例

```
 // 正确返回值
{
   "log_id": 3314921889,
}
// 发生错误时返回值
{
  "error_code": 216619,
  "log_id": 815967402,
  "error_msg": "user must be in one group at least"
}
```

身份验证

质量检测（可选）活体检测（可选）公安验证（必选）

```
 $image = file_get_contents('example.jpg');
$idCardNumber = "110233112299822211";

$name = "张三";

// 调用身份验证
$client->personVerify($image, $idCardNumber, $name);

// 如果有可选参数
$options = array();
$options["quality"] = "use";
$options["quality_conf"] = "{\"left_eye\": 0.6, \"right_eye\": 0.6}";
$options["faceliveness"] = "use";
$options["faceliveness_conf"] = "{\"faceliveness\": 0.834963}";
$options["ext_fields"] = "qualities";

// 带参数调用身份验证
$client->personVerify($image, $idCardNumber, $name, $options);
```

身份验证 请求参数详情

参数名称	是否必选	类型	说明
image	是	string	图像数据，base64编码，要求base64编码后大小不超过4M，最短边至少15px，最长边最大4096px,支持jpg/png/bmp格式
id_card_number	是	string	身份证号（真实身份证号号码）。我们的服务端会做格式校验，并通过错误码返回，但是为了您的产品反馈体验更及时，建议在产品前端做一下号码格式校验与反馈
name	是	string	utf8，姓名（真实姓名，和身份证号匹配）
quality	否	string	判断图片中的人脸质量是否符合条件。use表示需要做质量控制，质量不符合条件的照片会被直接拒绝
quality_conf	否	string	人脸质量检测中每一项指标的具体阈值设定，json串形式，当指定quality:use时生效
faceliveness	否	string	判断活体值是否达标。use表示需要做活体检测，低于活体阈值的照片会直接拒绝
faceliveness_conf	否	string	人脸活体检测的阈值设定，json串形式，当指定faceliveness:use时生效。默认使用的阈值如下：{faceliveness：0.834963}
ext_fields	否	string	可选项为faceliveness，qualities。选择具体的项，则返回参数中将会显示相应的扩展字段。如faceliveness表示返回结果中包含活体相关内容，qualities表示返回结果中包含质量检测相关内容

身份验证 返回数据参数详情

参数	必须	类型	说明
log_id	是	uint64	日志id
result	是	float	与公安小图相似度可能性，用于验证生活照与公安小图是否为同一人，有正常分数时为[0~1]，推荐阈值0.8，超过即判断为同一人
ext_info	否	string	拓展信息json串，只有选择了ext_fields时才会返回具体信息。选择faceliveness返回具体活体分值信息，选择qualities返回人脸质量检测信息。两者可以同时选择，半角逗号分割。
+faceliveness	否	string	活体检测值，单帧图片建议阈值，小于此值则认为不是活体，超过则判断为活体
+qualities	否	string	质量检测结果
++occlusion	否	string	人脸遮挡情况
+++left_eye	否	string	左眼被遮挡的比例，取值范围[0~1]，数值越大遮挡越多；小于阈值时有返回，默认阈值0.6
+++right_eye	否	string	右眼被遮挡的比例，取值范围[0~1]，数值越大遮挡越多；小于阈值时有返回，默认阈值0.6
+++nose	否	string	鼻子被遮挡的比例，取值范围[0~1]，数值越大遮挡越多；小于阈值时有返回，默认阈值0.7
+++mouth	否	string	嘴巴被遮挡的比例，取值范围[0~1]，数值越大遮挡越多；小于阈值时有返回，默认阈值0.7
+++left_cheek	否	string	左脸颊被遮挡的比例，取值范围[0~1]，数值越大遮挡越多；小于阈值时有返回，默认阈值0.8
+++right_cheek	否	string	右脸颊被遮挡的比例，取值范围[0~1]，数值越大遮挡越多；小于阈值时有返回，默认阈值0.8
+++chin	否	string	下巴被遮挡的比例，取值范围[0~1]，数值越大遮挡越多；小于阈值时有返回，默认阈值0.8
++blur	否	string	人脸模糊度阈值，取值范围[0~1]，数值越大越模糊；小于阈值时有返回，默认阈值0.7
++illumination	否	string	脸部光照的灰度值阈值，取值范围[0~255]，数值越大光照越强；大于阈值时有返回，默认阈值30
++completeness	否	string	人脸完整度，0或1, 0为人脸溢出图像边界，1为人脸都在图像边界内

身份验证 返回示例

```
{
 "result":0.03419,
 "ext_info":{
   "faceliveness":0.834963
 },
 "log_id":772889134072410
}
```

在线活体检测

人脸基础信息，人脸质量检测，基于图片的活体检测

```
 $image = file_get_contents('example.jpg');

// 调用在线活体检测
$client->faceverify($image);

// 如果有可选参数
$options = array();
$options["max_face_num"] = 2;
$options["face_fields"] = "qualities";

// 带参数调用在线活体检测
$client->faceverify($image, $options);
```

在线活体检测 请求参数详情

参数名称	是否必选	类型	默认值	说明
image	是	string		图像数据，base64编码，要求base64编码后大小不超过4M，最短边至少15px，最长边最大4096px,支持jpg/png/bmp格式
max_face_num	否	string	1	最多处理人脸数目，默认值1
face_fields	否	string		如不选择此项，返回结果默认只有人脸框、概率和旋转角度。可选参数为qualities、faceliveness。qualities：图片质量相关判断；faceliveness：活体判断。如果两个参数都需要选择，请使用半角逗号分隔。

在线活体检测 返回数据参数详情

参数	类型	是否必须	说明
log_id	uint64	是	日志id
result_num	uint32	是	人脸数目
result	object[]	是	人脸属性对象的集合
+faceliveness	float	否	活体分数，face_fields包括qualities时返回
+location	bject	是	人脸在图片中的位置
++left	uint32	是	人脸区域离左边界的距离
++top	uint32	是	人脸区域离上边界的距离
++width	uint32	是	人脸区域的宽度
++height	uint32	是	人脸区域的高度
+face_probability	double	是	人脸置信度，范围0-1
+rotation_angle	int32	是	人脸框相对于竖直方向的顺时针旋转角，[-180,180]
+yaw	double	是	三维旋转之左右旋转角[-90(左), 90(右)]
+pitch	double	是	三维旋转之俯仰角度[-90(上), 90(下)]
+roll	double	是	平面内旋转角[-180(逆时针), 180(顺时针)]
+qualities	object	否	人脸质量信息。face_fields包含qualities时返回
++occlusion	object	是	人脸各部分遮挡的概率，区间[0, 1]
+++left_eye	double	是	左眼
+++right_eye	double	是	右眼
+++nose	double	是	鼻子
+++mouth	double	是	嘴
+++left_cheek	double	是	左脸颊
+++right_cheek	double	是	右脸颊
+++chin	double	是	下巴
++blur	double	是	人脸模糊程度，[0, 1]。0表示清晰，1表示模糊
++illumination	double	是	取值范围在[0,255],表示脸部区域的光照程度
++completeness	double	是	人脸完整度，[0, 1]。0表示完整，1表示不完整
++type	object	是	真实人脸/卡通人脸置信度
+++human	double	是	真实人脸置信度，[0, 1]
+++cartoon	double	是	卡通人脸置信度，[0, 1]

在线活体检测 返回示例

```
{
  log_id: 1900901488032821,
  result_num: 1,
  result: [
    {
      rotation_angle: 10,
      yaw: 11.357421875,
      faceliveness: 8.1253347161692e-05,
      location: {
        width: 96,
        top: 73,
        height: 96,
        left: 283
      },
      qualities: {
        illumination: 211,
        occlusion: {
          right_eye: 0,
          left_eye: 0.039751552045345,
          left_cheek: 0.12623985111713,
          mouth: 0,
          nose: 0,
          chin: 0.037661049515009,
          right_cheek: 0.024720622226596
        },
        completeness: 1,
        type: {
          cartoon: 0,
          human: 0
        },
        blur: 2.5251445032182e-11
      },
      pitch: 1.0063140392303,
      roll: 12.760620117188,
      face_probability: 1
    }
  ]
}
```


错误信息

错误返回格式

若请求错误，服务器将返回的JSON文本包含以下参数：

- **error_code**：错误码。
- **error_msg**：错误描述信息，帮助理解和解决发生的错误。

错误码

服务端返回的错误码

错误码	错误信息	描述
4	Open api request limit reached	集群超限额
14	IAM Certification failed	IAM鉴权失败，建议用户参照文档自查生成sign的方式是否正确，或换用控制台中ak sk的方式调用
17	Open api daily request limit reached	每天流量超限额
18	Open api qps request limit reached	QPS超限额
19	Open api total request limit reached	请求总量超限额
100	Invalid parameter	无效参数
110	Access token invalid or no longer valid	Access Token失效
111	Access token expired	Access token过期
216015	module closed	模块关闭
216100	invalid param	参数异常
216101	not enough param	缺少必须的参数
216102	service not support	请求了不支持的服务，请检查调用的url
216103	param too long	请求超长，一般为一次传入图片个数超过系统限制
216110	appid not exist	appid不存在，请重新检查后台应用列表中的应用信息
216111	invalid userid	userid信息非法，请检查对应的参数
216200	empty image	图片为空或者base64解码错误
216201	image format error	图片格式错误
216202	image size error	图片大小错误
216300	db error	数据库异常，**少量发生时重试即可**
216400	backend error	后端识别服务异常，可以根据具体msg查看错误原因
216401	internal error	内部错误
216402	face not found	未找到人脸，请检查图片是否含有人脸
216500	unknown error	未知错误
216611	user not exist	用户不存在，请确认该用户是否注册或注册已经生效(**需要已经注册超过5s**)
216613	fail to delete user record	删除用户图片记录失败，重试即可
216614	not enough images	两两比对中图片数少于2张，无法比较
216615	fail to process images	服务处理该图片失败，发生后重试即可
216616	image existed	图片已存在
216617	fail to add user	新增用户图片失败
216618	no user in group	组内用户为空，确认该group是否存在或已经生效(需要已经注册超过5s)
216631	request add user overlimit	本次请求添加的用户数量超限

Python语言

简介

Hi，您好，欢迎使用百度人脸识别服务。

本文档主要针对Python开发者，描述百度人脸识别接口服务的相关技术内容。如果您对文档内容有任何疑问，可以通过以下几种方式联系我们：

- 在百度云控制台内提交工单，咨询问题类型请选择**人工智能服务**；
- 如有疑问，进入AI社区交流：http://ai.baidu.com/forum/topic/list/165

接口能力

接口名称	接口能力简要描述
人脸检测	检测人脸并定位，返回五官关键点，及人脸各属性值
人脸比对	返回两两比对的人脸相似值
人脸查找	在一个人脸集合中找到找到相似的人脸，由一系列接口组成，包括人脸识别、人脸认证、人脸库管理相关接口（人脸注册、人脸更新、人脸删除、用户信息查询、组列表查询、组内用户列表查询、组间复制用户、组内删除用户）

版本更新记录

上线日期	版本号	更新内容
2018.4.9	2.2.2	新增身份验证，在线活体检测接口
2018.01.12	2.1.0	新增M：N多人脸识别
2017.12.22	2.0.0	SDK代码重构
2017.5.11	1.0.0	人脸识别服务上线

快速入门

安装人脸识别 Python SDK

人脸识别 Python SDK目录结构

```
 ├─── README.md
├─── aip                  //SDK目录
│    ├─── __init__.py      //导出类
│    ├─── base.py          //aip基类
│    ├─── http.py          //http请求
│    └─── face.py //人脸识别
└─── setup.py             //setuptools安装
```

支持Python版本：2.7.+ ,3.+

安装使用Python SDK有如下方式：

- 如果已安装pip，执行`pip install baidu-aip`即可。
- 如果已安装setuptools，执行`python setup.py install`即可。

新建AipFace

AipFace是人脸识别的Python SDK客户端，为使用人脸识别的开发人员提供了一系列的交互方法。

参考如下代码新建一个AipFace：

```
 from aip import AipFace

""" 你的 APPID AK SK """
APP_ID = '你的 App ID'
API_KEY = '你的 Api Key'
SECRET_KEY = '你的 Secret Key'

client = AipFace(APP_ID, API_KEY, SECRET_KEY)
```

在上面代码中，常量APP_ID在百度云控制台中创建，常量API_KEY与SECRET_KEY是在创建完毕应用后，系统分配给用户的，均为字符串，用于标识用户，为访问做签名验证，可在AI服务控制台中的**应用列表**中查看。

注意：如您以前是百度云的老用户，其中API_KEY对应百度云的“Access Key ID”，SECRET_KEY对应百度云的“Access Key Secret”。

配置AipFace

如果用户需要配置AipFace的网络请求参数(一般不需要配置)，可以在构造AipFace之后调用接口设置参数，目前只支持以下参数：

接口	说明
setConnectionTimeoutInMillis	建立连接的超时时间（单位：毫秒
setSocketTimeoutInMillis	通过打开的连接传输数据的超时时间（单位：毫秒）

接口说明

人脸检测

检测请求图片中的人脸，返回人脸位置、72个关键点坐标、及人脸相关属性信息。

检测响应速度，与图片中人脸数量相关，人脸数量较多时响应时间会有些许延长。

典型应用场景：如**人脸属性分析**，**基于人脸关键点的加工分析**，**人脸营销活动**等。

> 五官位置会标记具体坐标；72个关键点坐标也包含具体坐标，但不包含对应位置的详细位置描述。

```
 """ 读取图片 """
def get_file_content(filePath):
    with open(filePath, 'rb') as fp:
        return fp.read()

image = get_file_content('example.jpg')

""" 调用人脸检测 """
client.detect(image);

""" 如果有可选参数 """
options = {}
options["max_face_num"] = 2
options["face_fields"] = "age"

""" 带参数调用人脸检测 """
client.detect(image, options)
```

人脸检测 请求参数详情

参数名称	是否必选	类型	默认值	说明
image	是	string		图像数据，base64编码，要求base64编码后大小不超过4M，最短边至少15px，最长边最大4096px,支持jpg/png/bmp格式
max_face_num	否	string	1	最多处理人脸数目，默认值1
face_fields	否	string		包括age,beauty,expression,faceshape,gender,glasses,landmark,qualities信息，逗号分隔，默认只返回人脸框、概率和旋转角度

人脸检测 返回数据参数详情

参数	类型	必选	说明
log_id	uint64	是	日志id
result_num	uint32	是	人脸数目
result	object[]	是	人脸属性对象的集合
+age	double	否	年龄。face_fields包含age时返回
+beauty	double	否	美丑打分，范围0-100，越大表示越美。face_fields包含beauty时返回
+location	object	是	人脸在图片中的位置
++left	uint32	是	人脸区域离左边界的距离
++top	uint32	是	人脸区域离上边界的距离
++width	uint32	是	人脸区域的宽度
++height	uint32	是	人脸区域的高度
+face_probability	double	是	人脸置信度，范围0-1
+rotation_angle	int32	是	人脸框相对于竖直方向的顺时针旋转角，[-180,180]
+yaw	double	是	三维旋转之左右旋转角[-90(左), 90(右)]
+pitch	double	是	三维旋转之俯仰角度[-90(上), 90(下)]
+roll	double	是	平面内旋转角[-180(逆时针), 180(顺时针)]
+expression	uint32	否	表情，0，不笑；1，微笑；2，大笑。face_fields包含expression时返回
+expression_probability	double	否	表情置信度，范围0~1。face_fields包含expression时返回
+faceshape	object[]	否	脸型置信度。face_fields包含faceshape时返回
++type	string	是	脸型：square/triangle/oval/heart/round
++probability	double	是	置信度：0~1
+gender	string	否	male、female。face_fields包含gender时返回
+gender_probability	double	否	性别置信度，范围0~1。face_fields包含gender时返回
+glasses	uint32	否	是否带眼镜，0-无眼镜，1-普通眼镜，2-墨镜。face_fields包含glasses时返回
+glasses_probability	double	否	眼镜置信度，范围0~1。face_fields包含glasses时返回
+landmark	object[]	否	4个关键点位置，左眼中心、右眼中心、鼻尖、嘴中心。face_fields包含landmark时返回
++x	uint32	否	x坐标
++y	uint32	否	y坐标
+landmark72	object[]	否	72个特征点位置，示例图 。face_fields包含landmark时返回

++x	uint32	否	x坐标
++y	uint32	否	y坐标
+qualities	object	否	人脸质量信息。face_fields包含qualities时返回
++occlusion	object	是	人脸各部分遮挡的概率,[0, 1],0表示完整，1表示不完整
+++left_eye	double	是	左眼
+++right_eye	double	是	右眼
+++nose	double	是	鼻子
+++mouth	double	是	嘴
+++left_cheek	double	是	左脸颊
+++right_cheek	double	是	右脸颊
+++chin	double	是	下巴
++blur	double	是	人脸模糊程度，[0, 1]。0表示清晰，1表示模糊
++illumination	-	是	取值范围在[0,255],表示脸部区域的光照程度
++completeness	-	是	人脸完整度，[0, 1]。0表示完整，1表示不完整
++type	object	是	真实人脸/卡通人脸置信度
+++human	-	是	真实人脸置信度，[0, 1]
+++cartoon	-	是	卡通人脸置信度，[0, 1]

人脸检测 返回示例

```
{
  "result_num": 1,
  "result": [
    {
      "location": {
        "left": 117,
        "top": 131,
        "width": 172,
        "height": 170
      },
      "face_probability": 1,
      "rotation_angle": 2,
      "yaw": -0.34859421849251,
      "pitch": 2.3033397197723,
      "roll": 1.9135693311691,
      "landmark": [
        {
          "x": 161.74819946289,
          "y": 163.30244445801
        },
        ...
      ],
      "landmark72": [
        {
          "x": 115.86531066895,
          "y": 170.0546875
        } ,
        ...
      ],
      "age": 29.298097610474,
      "beauty": 55.128883361816,
      "expression": 1,
      "expression_probablity": 0.5543018579483,
      "gender": "male",
      "gender_probability": 0.99979132413864,
      "glasses": 0,
      "glasses_probability": 0.99999964237213,
      "qualities": {
        "occlusion": {
          "left_eye": 0,
          "right_eye": 0,
          "nose": 0,
          "mouth": 0,
          "left_cheek": 0.0064102564938366,
          "right_cheek": 0.0057411273010075,
          "chin": 0
        },
        "blur": 1.1886881756684e-10,
        "illumination": 141,
        "completeness": 1,
        "type": {
          "human": 0.99935841560364,
          "cartoon": 0.00064159056637436
        }
      }
    }
  ],
  "log_id": 2493878179101621
}
```


质量判断

可通过人脸检测接口，基于以下字段和对应阈值，进行质量检测的判断，以保证人脸质量符合后续业务操作要求。

指标	字段与解释	推荐数值界限
遮挡范围	**occlusion**（0~1），0为无遮挡，1是完全遮挡 含有多个具体子字段，表示脸部多个部位 通常用作判断头发、墨镜、口罩等遮挡	left_eye：0.6, #左眼被遮挡的阈值 right_eye：0.6, #右眼被遮挡的阈值 nose：0.7, #鼻子被遮挡的阈值 mouth：0.7, #嘴巴被遮挡的阈值 left_check：0.8, #左脸颊被遮挡的阈值 right_check：0.8, #右脸颊被遮挡的阈值 chin_contour：0.6, #下巴被遮挡阈值
模糊度范围	**Blur**（0~1），0是最清晰，1是最模糊	小于0.7
光照范围	**illumination**（0~255） 脸部光照的灰度值，0表示光照不好 以及对应客户端SDK中，YUV的Y分量	大于40
姿态角度	**Pitch**：三维旋转之俯仰角度[-90(上), 90(下)] **Roll**：平面内旋转角[-180(逆时针), 180(顺时针)] **Yaw**：三维旋转之左右旋转角[-90(左), 90(右)]	分别小于20度
人脸完整度	**completeness**（0或1），0代表完整，1代表不完整	小于0.4
人脸大小	人脸部分的大小 建议长宽像素值范围：80*80~200*200	人脸部分不小于**100*100**像素

人脸比对

该请求用于比对多张图片中的人脸相似度并返回两两比对的得分，可用于判断两张脸是否是同一人的可能性大小。

典型应用场景：如**人证合一验证**，**用户认证**等，可与您现有的人脸库进行比对验证。

> **说明：**支持对比对的两张图片做在线活体检测

```
 """ 读取图片 """
def get_file_content(filePath):
    with open(filePath, 'rb') as fp:
        return fp.read()

images = [
    get_file_content('example0.jpg'),
    get_file_content('example1.jpg'),
]

""" 调用人脸比对 """
client.match(images);

""" 如果有可选参数 """
options = {}
options["ext_fields"] = "qualities"
options["image_liveness"] = ",faceliveness"
options["types"] = "7,13"

""" 带参数调用人脸比对 """
client.match(images, options)
```

人脸比对 请求参数详情

参数名称	是否必选	类型	可选值范围	说明
images	是	string		base64编码后的多张图片数据，半角逗号分隔，单次请求总共最大20M
ext_fields	否	string		返回质量信息，取值固定:目前支持qualities(质量检测)。(对所有图片都会做改处理)
image_liveness	否	string	faceliveness,faceliveness - 对比对的两张图片都做活体检测 ,faceliveness - 对第一张图片不做活体检测、第二张图做活体检测 faceliveness, - 对第一张图片做活体检测、第二张图不做活体检测	返回的活体信息，“faceliveness,faceliveness” 表示对比对的两张图片都做活体检测；“,faceliveness” 表示对第一张图片不做活体检测、第二张图做活体检测；“faceliveness,” 表示对第一张图片做活体检测、第二张图不做活体检测； **注：需要用于判断活体的图片，图片中的人脸像素面积需要不小于100px*100px，人脸长宽与图片长宽比例，不小于1/3**
types	否	string		请求对比的两张图片的类型，示例：“7,13” **12**表示带水印证件照：一般为带水印的小图，如公安网小图 **7**表示生活照：通常为手机、相机拍摄的人像图片、或从网络获取的人像图片等 **13**表示证件照片：如拍摄的身份证、工卡、护照、学生证等证件图片，**注**：需要确保人脸部分不可太小，通常为100px*100px

人脸比对 返回数据参数详情

字段	必选	类型	说明
log_id	是	uint64	请求唯一标识码，随机数
result_num	是	uint32	返回结果数目，即：result数组中元素个数
result	是	array(object)	结果数据，index和请求图片index对应。数组元素为每张图片的匹配得分数组，top n。 得分[0,100.0]
+index_i	是	uint32	比对图片1的index
+index_j	是	uint32	比对图片2的index
+score	是	double	比对得分
ext_info	否	array（dict）	对应参数中的ext_fields
+qualities	否	string	质量相关的信息，无特殊需求可以不使用
+faceliveness	否	string	活体分数，如0.49999。单帧活体检测参考阈值**0.393241**，超过此分值以上则可认为是活体。**注意：活体检测接口主要用于判断是否为二次翻拍，需要限制用户为当场拍照获取图片**；推荐配合客户端SDK有动作校验活体使用

人脸比对 返回示例

```
 //请求两张图片
{
    "log_id": 73473737,
    "result_num":1,
    "result": [
        {
            "index_i": 0,
            "index_j": 1,
            "score": 44.3
        }
    ]
}
```

人脸识别

用于计算指定组内用户，与上传图像中人脸的相似度。识别前提为您已经创建了一个**人脸库**。

典型应用场景：如**人脸闸机**，**考勤签到**，**安防监控**等。

> **说明：**人脸识别返回值不直接判断是否是同一人，只返回用户信息及相似度分值。

> **说明：**推荐可判断为同一人的相似度分值为**80**，您也可以根据业务需求选择更合适的阈值。

```
 groupId = "group1,group2"

""" 读取图片 """
def get_file_content(filePath):
    with open(filePath, 'rb') as fp:
        return fp.read()

image = get_file_content('example.jpg')

""" 调用人脸识别 """
client.identifyUser(groupId, image);

""" 如果有可选参数 """
options = {}
options["ext_fields"] = "faceliveness"
options["user_top_num"] = 3

""" 带参数调用人脸识别 """
client.identifyUser(groupId, image, options)
```

人脸识别 请求参数详情

参数名称	是否必选	类型	默认值	说明
group_id	是	string		用户组id，标识一组用户（由数字、字母、下划线组成），长度限制128B。如果需要将一个uid注册到多个group下，group_id需要用多个逗号分隔，每个group_id长度限制为48个英文字符。**注：group无需单独创建，注册用户时则会自动创建group。** **产品建议**：根据您的业务需求，可以将需要注册的用户，按照业务划分，分配到不同的group下，例如按照会员手机尾号作为groupid，用于刷脸支付、会员计费消费等，这样可以尽可能控制每个group下的用户数与人脸数，提升检索的准确率
image	是	string		图像数据，base64编码，要求base64编码后大小不超过4M，最短边至少15px，最长边最大4096px,支持jpg/png/bmp格式
ext_fields	否	string		特殊返回信息，多个用逗号分隔，取值固定: 目前支持faceliveness(活体检测)。**注：需要用于判断活体的图片，图片中的人脸像素面积需要不小于100px*100px，人脸长宽与图片长宽比例，不小于1/3**
user_top_num	否	string	1	返回用户top数，默认为1，最多返回5个

人脸识别 返回数据参数详情

字段	必选	类型	说明
log_id	是	uint64	请求唯一标识码，随机数
result_num	是	uint32	返回结果数目，即：result数组中元素个数
ext_info	否	array	对应参数中的ext_fields
+faceliveness	否	string	活体分数，如0.49999。单帧活体检测参考阈值0.393241，超过此分值以上则可认为是活体。**注意：活体检测接口主要用于判断是否为二次翻拍，需要限制用户为当场拍照获取图片**；推荐配合客户端SDK有动作校验活体使用
result	是	array(object)	结果数组
+group_id	是	string	对应的这个用户的group_id
+uid	是	string	匹配到的用户id
+user_info	是	string	注册时的用户信息
+scores	是	array(double)	结果数组，数组元素为匹配得分，top n。得分[0,100.0]

人脸识别 返回示例

```
{
  "log_id": 73473737,
  "result_num":1,
  "result": [
    {
      "group_id" : "test1",
      "uid": "u333333",
      "user_info": "Test User",
      "scores": [
            99.3,
            83.4
      ]
    }
  ]
}
```

人脸认证

用于识别上传的图片是否为指定用户，即查找前需要先确定要查找的用户在人脸库中的id。

典型应用场景：如**人脸登录**，**人脸签到**等

> **说明：**人脸认证与人脸识别的差别在于：人脸识别需要指定一个待查找的人脸库中的组；而人脸认证需要指定具体的用户id即可，不需要指定具体的人脸库中的组；实际应用中，人脸认证需要用户或系统先输入id，这增加了验证安全度，但也增加了复杂度，具体使用哪个接口需要视您的业务场景判断。

> **说明：**请求参数中，新增在线活体检测

```
uid = "user1"

groupId = "group1,group2"

""" 读取图片 """
def get_file_content(filePath):
    with open(filePath, 'rb') as fp:
        return fp.read()

image = get_file_content('example.jpg')

""" 调用人脸认证 """
client.verifyUser(uid, groupId, image);

""" 如果有可选参数 """
options = {}
options["top_num"] = 3
options["ext_fields"] = "faceliveness"

""" 带参数调用人脸认证 """
client.verifyUser(uid, groupId, image, options)
```

人脸认证 请求参数详情

参数名称	是否必选	类型	默认值	说明
uid	是	string		用户id（由数字、字母、下划线组成），长度限制128B
group_id	是	string		用户组id，标识一组用户（由数字、字母、下划线组成），长度限制128B。如果需要将一个uid注册到多个group下，group_id需要用多个逗号分隔，每个group_id长度限制为48个英文字符。**注：group无需单独创建，注册用户时则会自动创建group。** **产品建议**：根据您的业务需求，可以将需要注册的用户，按照业务划分，分配到不同的group下，例如按照会员手机尾号作为groupid，用于刷脸支付、会员计费消费等，这样可以尽可能控制每个group下的用户数与人脸数，提升检索的准确率
image	是	string		图像数据，base64编码，要求base64编码后大小不超过4M，最短边至少15px，最长边最大4096px,支持jpg/png/bmp格式
top_num	否	string	1	返回用户top数，默认为1
ext_fields	否	string		特殊返回信息，多个用逗号分隔，取值固定: 目前支持faceliveness(活体检测)。**注：需要用于判断活体的图片，图片中的人脸像素面积需要不小于100px*100px，人脸长宽与图片长宽比例，不小于1/3**

人脸认证 返回数据参数详情

字段	必选	类型	说明
log_id	是	uint64	请求唯一标识码，随机数
result_num	是	uint32	返回结果数目，即：result数组中元素个数
result	是	array(double)	结果数组，数组元素为匹配得分，top n。 得分范围[0,100.0]。推荐得分超过80可认为认证成功
ext_info	否	array	对应参数中的ext_fields
+faceliveness	否	string	活体分数，如0.49999。单帧活体检测参考阈值0.393241，超过此分值以上则可认为是活体。**活体检测接口主要用于判断是否为二次翻拍，需要限制用户为当场拍照获取图片**；推荐配合客户端SDK有动作校验活体使用

人脸认证 返回示例

```
{
  "log_id": 73473737,
  "result_num":2,
  "result": [
        99.3,
        83.6
  ]
}
```

M:N 识别

待识别的图片中，存在多张人脸的情况下，支持在一个人脸库中，一次请求，同时返回图片中所有人脸的识别结果。

```
 groupId = "group1,group2"

""" 读取图片 """
def get_file_content(filePath):
    with open(filePath, 'rb') as fp:
        return fp.read()

image = get_file_content('example.jpg')

""" 调用M:N 识别 """
client.multiIdentify(groupId, image);

""" 如果有可选参数 """
options = {}
options["ext_fields"] = "faceliveness"
options["detect_top_num"] = 3
options["user_top_num"] = 2

""" 带参数调用M:N 识别 """
client.multiIdentify(groupId, image, options)
```

M:N 识别 请求参数详情

参数名称	是否必选	类型	默认值	说明
group_id	是	string		用户组id，标识一组用户（由数字、字母、下划线组成），长度限制128B。如果需要将一个uid注册到多个group下，group_id需要用多个逗号分隔，每个group_id长度限制为48个英文字符。**注：group无需单独创建，注册用户时则会自动创建group。** **产品建议**：根据您的业务需求，可以将需要注册的用户，按照业务划分，分配到不同的group下，例如按照会员手机尾号作为groupid，用于刷脸支付、会员计费消费等，这样可以尽可能控制每个group下的用户数与人脸数，提升检索的准确率
image	是	string		图像数据，base64编码，要求base64编码后大小不超过4M，最短边至少15px，最长边最大4096px,支持jpg/png/bmp格式
ext_fields	否	string		特殊返回信息，多个用逗号分隔，取值固定: 目前支持faceliveness(活体检测)。**注：需要用于判断活体的图片，图片中的人脸像素面积需要不小于100px*100px，人脸长宽与图片长宽比例，不小于1/3**
detect_top_num	否	string	1	检测多少个人脸进行比对，默认值1（最对返回10个）
user_top_num	否	string	1	返回识别结果top人数”，当同一个人有多张图片时，只返回比对最高的1个分数（即，scores参数只有一个值），默认为1（最多返回20个）

M:N 识别 返回数据参数详情

参数	字段	必选	类型	说明
log_id	-	是	uint32	请求标识码，随机数，唯一
result_num	-	是	float	返回结果数目，即：result数组中元素个数（PS：最终返回的个数是小于等于 “实际检测到的人脸数” * “每个人脸匹配的top人数”）
ext_info	-	否	array	对应参数中的ext_fields
-	faceliveness	否	string	活体检测分数，单帧活体检测参考阈值**0.393241**，超过此分值以上则可认为是活体。**注意：活体检测接口主要用于判断是否为二次翻拍，需要限制用户为当场拍照获取图片**；推荐配合客户端SDK有动作校验活体使用
result	-	是	array(object)	-
-	group_id	是	string	对应的这个用户的group_id
-	uid	是	string	匹配到的用户id
-	user_info	是	string	注册时的用户信息
-	scores	是	array(double)	结果数组，数组元素为匹配得分，得分[0,100.0]；个数取决于user_top_num的设定。推荐**80分**以上即可判断为同一人
-	position	是	object	人脸位置，如{top:111,left:222,width:333,height:444,degree:20}

M:N 识别 返回示例


```
{
  "log_id": 73473737,
  "result_num":1,
  "result": [
     {
        "group_id" : "test1",
        "uid": "u333333",
        "user_info": "Test User",
        "position":{"left":726.99188232422,"top":288.37701416016,"width":44,"height":42,"degree":-
4,"prob":0.91117089986801},
        "scores": [
              99.3
        ]
     },
     {
        "group_id" : "test1",
        "uid": "u2222222",
        "user_info": "Test User",
        "position":{"left":726.99188232422,"top":288.37701416016,"width":44,"height":42,"degree":-
4,"prob":0.91117089986801},
        "scores": [
              82.3
        ]
     }
  ]
}
```


人脸注册

用于从人脸库中新增用户，可以设定多个用户所在组，及组内用户的人脸图片，

典型应用场景：构建您的人脸库，如**会员人脸注册**，**已有用户补全人脸信息**等。

人脸库、用户组、用户、用户下的人脸**层级关系**如下所示：

```
|- 人脸库
 |- 用户组一
   |- 用户01
     |- 人脸
   |- 用户02
     |- 人脸
     |- 人脸
     ....
    ....
 |- 用户组二
 |- 用户组三
 |- 用户组四
 ....
```

关于人脸库的设置限制

- 每个开发者账号可以创建100个appid；
- **每个appid对应一个人脸库，且不同appid之间，人脸库互不相通**；
- 每个人脸库下，可以创建多个用户组，用户组（group）数量**没有限制**；
- 每个用户组（group）下，可添加最多**无限**张人脸，**无限**个uid；
- 每个用户（uid）所能注册的最大人脸数量**没有限制**；

为了保证识别效果，请控制注册人脸的质量（通过`/detect`人脸检测接口判断），具体参数可详见下表所示：

质量判断

可通过人脸检测接口，基于以下字段和对应阈值，进行质量检测的判断，以保证人脸质量符合后续业务操作要求。

指标	字段与解释	推荐数值界限
遮挡范围	**occlusion**（0~1），0为无遮挡，1是完全遮挡 含有多个具体子字段，表示脸部多个部位 通常用作判断头发、墨镜、口罩等遮挡	left_eye：0.6, #左眼被遮挡的阈值 right_eye：0.6, #右眼被遮挡的阈值 nose：0.7, #鼻子被遮挡的阈值 mouth：0.7, #嘴巴被遮挡的阈值 left_check：0.8, #左脸颊被遮挡的阈值 right_check：0.8, #右脸颊被遮挡的阈值 chin_contour：0.6, #下巴被遮挡阈值
模糊度范围	**Blur**（0~1），0是最清晰，1是最模糊	小于0.7
光照范围	**illumination**（0~255） 脸部光照的灰度值，0表示光照不好 以及对应客户端SDK中，YUV的Y分量	大于40
姿态角度	**Pitch**：三维旋转之俯仰角度[-90(上), 90(下)] **Roll**：平面内旋转角[-180(逆时针), 180(顺时针)] **Yaw**：三维旋转之左右旋转角[-90(左), 90(右)]	分别小于20度
人脸完整度	**completeness**（0或1），0为人脸溢出图像边界，1为人脸都在图像边界内	视业务逻辑判断
人脸大小	人脸部分的大小 建议长宽像素值范围：80*80~200*200	人脸部分不小于**100*100**像素

```
 uid = "user1"

userInfo = "user's info"

groupId = "group1,group2"

""" 读取图片 """
def get_file_content(filePath):
    with open(filePath, 'rb') as fp:
        return fp.read()

image = get_file_content('example.jpg')

""" 调用人脸注册 """
client.addUser(uid, userInfo, groupId, image);

""" 如果有可选参数 """
options = {}
options["action_type"] = "replace"

""" 带参数调用人脸注册 """
client.addUser(uid, userInfo, groupId, image, options)
```

人脸注册 请求参数详情

参数名称	是否必选	类型	默认值	说明
uid	是	string		用户id（由数字、字母、下划线组成），长度限制128B
user_info	是	string		用户资料，长度限制256B
group_id	是	string		用户组id，标识一组用户（由数字、字母、下划线组成），长度限制128B。如果需要将一个uid注册到多个group下，group_id需要用多个逗号分隔，每个group_id长度限制为48个英文字符。**注：group无需单独创建，注册用户时则会自动创建group。** **产品建议**：根据您的业务需求，可以将需要注册的用户，按照业务划分，分配到不同的group下，例如按照会员手机尾号作为groupid，用于刷脸支付、会员计费消费等，这样可以尽可能控制每个group下的用户数与人脸数，提升检索的准确率
image	是	string		图像base64编码，**每次仅支持单张图片，图片编码后大小不超过10M**。为保证后续识别的效果较佳，建议注册的人脸，为用户正面人脸。
action_type	否	string	append	参数包含append、replace。**如果为“replace”，则每次注册时进行替换replace（新增或更新）操作，默认为append操作**。例如：uid在库中已经存在时，对此uid重复注册时，新注册的图片默认会**追加**到该uid下，如果手动选择`action_type:replace`，则会用新图替换库中该uid下所有图片。

人脸注册 返回数据参数详情

字段	是否必选	类型	说明
log_id	是	uint64	请求唯一标识码，随机数

人脸注册 返回示例

```
 // 注册成功
{
   "log_id": 73473737,
}
// 注册发生错误
{
  "error_code": 216616,
  "log_id": 674786177,
  "error_msg": "image exist"
}
```

人脸更新

用于对人脸库中指定用户，更新其下的人脸图像。

> **说明：**针对一个uid执行更新操作，新上传的人脸图像将覆盖此uid原有所有图像。

> **说明：**执行更新操作，如果该uid不存在时，会返回错误。如果添加了action_type:replace,则不会报错，并自动注册该uid，操作结果等同注册新用户。

```
 uid = "user1"

userInfo = "user's info"

groupId = "group1"

""" 读取图片 """
def get_file_content(filePath):
    with open(filePath, 'rb') as fp:
        return fp.read()

image = get_file_content('example.jpg')

""" 调用人脸更新 """
client.updateUser(uid, userInfo, groupId, image);

""" 如果有可选参数 """
options = {}
options["action_type"] = "replace"

""" 带参数调用人脸更新 """
client.updateUser(uid, userInfo, groupId, image, options)
```

人脸更新 请求参数详情

参数名称	是否必选	类型	默认值	说明
uid	是	string		用户id（由数字、字母、下划线组成），长度限制128B
user_info	是	string		用户资料，长度限制256B
group_id	是	string		更新指定groupid下uid对应的信息
image	是	string		图像数据，base64编码，要求base64编码后大小不超过4M，最短边至少15px，最长边最大4096px,支持jpg/png/bmp格式
action_type	否	string	append	目前仅支持replace，uid不存在时，不报错，会自动变为注册操作；未选择该参数时，如果uid不存在会提示错误

人脸更新 返回数据参数详情

字段	是否必选	类型	说明
log_id	是	uint64	请求唯一标识码，随机数

人脸更新 返回示例

```
 // 更新成功
{
    "log_id": 73473737,
}
// 更新发生错误
{
  "error_code": 216612,
  "log_id": 1137508902,
  "error_msg": "user not exist"
}
```

人脸删除

用于从人脸库中删除一个用户。

人脸删除注意事项：

- 删除的内容，包括用户所有图像和身份信息；
- 如果一个uid存在于多个用户组内，将会同时将从各个组中把用户删除
- 如果指定了group_id，则只删除此group下的uid相关信息

```
 uid = "user1"

""" 调用人脸删除 """
client.deleteUser(uid);

""" 如果有可选参数 """
options = {}
options["group_id"] = "group1"

""" 带参数调用人脸删除 """
client.deleteUser(uid, options)
```

人脸删除 请求参数详情

参数名称	是否必选	类型	说明
uid	是	string	用户id（由数字、字母、下划线组成），长度限制128B
group_id	否	string	删除指定groupid下uid对应的信息

人脸删除 返回数据参数详情

字段	是否必选	类型	说明
log_id	是	uint64	请求唯一标识码，随机数

人脸删除 返回示例

```
 // 删除成功
{
    "log_id": 73473737,
}
// 删除发生错误
{
  "error_code": 216612,
  "log_id": 1137508902,
  "error_msg": "user not exist"
}
```

用户信息查询

用于查询人脸库中某用户的详细信息。

```
 uid = "user1"

""" 调用用户信息查询 """
client.getUser(uid);

""" 如果有可选参数 """
options = {}
options["group_id"] = "group1"

""" 带参数调用用户信息查询 """
client.getUser(uid, options)
```

用户信息查询 请求参数详情

参数名称	是否必选	类型	说明
uid	是	string	用户id（由数字、字母、下划线组成），长度限制128B
group_id	否	string	选择指定group_id则只查找group列表下的uid内容，如果不指定则查找所有group下对应uid的信息

用户信息查询 返回数据参数详情

字段	是否必选	类型	说明
log_id	是	uint64	请求标识码，随机数，唯一
result	是	array(double)	结果数组
+uid	是	string	匹配到的用户id
+user_info	是	string	注册时的用户信息
+groups	是	array(string)	用户所属组列表

用户信息查询 返回示例

```
 {
“result_num” : 2
   "result": {
      [
         "uid": "testuser2",
         "user_info": "registed user info ...",
         "group_id": "grep1",
      ],
      [
         "uid": "testuser2",
         "user_info": "registed user info2 ...",
         "group_id": "grep2",
      ],
   },
   "log_id": 2979357502
}
```

组列表查询

用于查询用户组的列表。

```
 """ 调用组列表查询 """
client.getGroupList();

""" 如果有可选参数 """
options = {}
options["start"] = 0
options["num"] = 50

""" 带参数调用组列表查询 """
client.getGroupList(, options)
```

组列表查询 请求参数详情

参数名称	是否必选	类型	默认值	说明
start	否	string	0	默认值0，起始序号
num	否	string	100	返回数量，默认值100，最大值1000

组列表查询 返回数据参数详情

字段	是否必选	类型	说明
result_num	是	number	返回个数
result	是	array(string)	group_id列表

组列表查询 返回示例

```
{
  "result_num": 2,
  "result": [
    "grp1",
    "grp2"
  ],
  "log_id": 3314921889
}
```

组内用户列表查询

用于查询指定用户组中的用户列表。

```
 groupId = "group1"

""" 调用组内用户列表查询 """
client.getGroupUsers(groupId);

""" 如果有可选参数 """
options = {}
options["start"] = 0
options["num"] = 50

""" 带参数调用组内用户列表查询 """
client.getGroupUsers(groupId, options)
```

组内用户列表查询 请求参数详情

参数名称	是否必选	类型	默认值	说明
group_id	是	string		用户组id（由数字、字母、下划线组成），长度限制128B
start	否	string	0	默认值0，起始序号
num	否	string	100	返回数量，默认值100，最大值1000

组内用户列表查询 返回数据参数详情

字段	是否必选	类型	说明
log_id	是	uint64	请求标识码，随机数，唯一
result_num	是	uint32	返回个数
result	是	array(object)	user列表
+uid	是	string	用户id
+user_info	是	string	用户信息

组内用户列表查询 返回示例

```
{
  "log_id": 3314921889,
  "result_num": 2,
  "result": [
    {
      "uid": "uid1",
      "user_info": "user info 1"
    },
    {
      "uid": "uid2",
      "user_info": "user info 2"
    }
  ]
}
```

组间复制用户

用于将已经存在于人脸库中的用户**复制到一个新的组**。

> **说明：**并不是向一个指定组内添加用户，而是直接从其它组复制用户信息 如果需要注册用户，请直接使用**人脸注册接口**

```
 srcGroupId = "group1"

groupId = "group1,group2"

uid = "user1"

""" 调用组间复制用户 """
client.addGroupUser(srcGroupId, groupId, uid);
```

组间复制用户 请求参数详情

参数名称	是否必选	类型	说明
src_group_id	是	string	从指定group里复制信息
group_id	是	string	用户组id，标识一组用户（由数字、字母、下划线组成），长度限制128B。如果需要将一个uid注册到多个group下，group_id需要用多个逗号分隔，每个group_id长度限制为48个英文字符。**注：group无需单独创建，注册用户时则会自动创建group。** **产品建议**：根据您的业务需求，可以将需要注册的用户，按照业务划分，分配到不同的group下，例如按照会员手机尾号作为groupid，用于刷脸支付、会员计费消费等，这样可以尽可能控制每个group下的用户数与人脸数，提升检索的准确率
uid	是	string	用户id（由数字、字母、下划线组成），长度限制128B

组间复制用户 返回数据参数详情

字段	是否必选	类型	说明
log_id	是	uint64	请求标识码，随机数，唯一

组间复制用户 返回示例

```
 // 正确返回值
{
    "log_id": 3314921889,
}
// 发生错误时返回值
{
  "error_code": 216100,
  "log_id": 3111284097,
  "error_msg": "already add"
}
```

组内删除用户

用于将用户从某个组中删除，但不会删除用户在其它组的信息。

> **说明：**当用户仅属于单个分组时，本接口将返回错误，请使用**人脸删除接口**

```
 groupId = "group1,group2"

uid = "user1"

""" 调用组内删除用户 """
client.deleteGroupUser(groupId, uid);
```

组内删除用户 请求参数详情

参数名称	是否必选	类型	说明
group_id	是	string	用户组id，标识一组用户（由数字、字母、下划线组成），长度限制128B。如果需要将一个uid注册到多个group下，group_id需要用多个逗号分隔，每个group_id长度限制为48个英文字符。**注：group无需单独创建，注册用户时则会自动创建group。** **产品建议**：根据您的业务需求，可以将需要注册的用户，按照业务划分，分配到不同的group下，例如按照会员手机尾号作为groupid，用于刷脸支付、会员计费消费等，这样可以尽可能控制每个group下的用户数与人脸数，提升检索的准确率
uid	是	string	用户id（由数字、字母、下划线组成），长度限制128B

组内删除用户 返回数据参数详情

字段	是否必选	类型	说明
log_id	是	uint64	请求标识码，随机数，唯一

组内删除用户 返回示例

```
 // 正确返回值
{
   "log_id": 3314921889,
}
// 发生错误时返回值
{
  "error_code": 216619,
  "log_id": 815967402,
  "error_msg": "user must be in one group at least"
}
```

身份验证

质量检测（可选）活体检测（可选）公安验证（必选）

```
 """ 读取图片 """
def get_file_content(filePath):
    with open(filePath, 'rb') as fp:
        return fp.read()

image = get_file_content('example.jpg')
idCardNumber = "110233112299822211"

name = "张三"

""" 调用身份验证 """
client.personVerify(image, idCardNumber, name);

""" 如果有可选参数 """
options = {}
options["quality"] = "use"
options["quality_conf"] = "{\"left_eye\": 0.6, \"right_eye\": 0.6}"
options["faceliveness"] = "use"
options["faceliveness_conf"] = "{\"faceliveness\": 0.834963}"
options["ext_fields"] = "qualities"

""" 带参数调用身份验证 """
client.personVerify(image, idCardNumber, name, options)
```

身份验证 请求参数详情

参数名称	是否必选	类型	说明
image	是	string	图像数据，base64编码，要求base64编码后大小不超过4M，最短边至少15px，最长边最大4096px,支持jpg/png/bmp格式
id_card_number	是	string	身份证号（真实身份证号号码）。我们的服务端会做格式校验，并通过错误码返回，但是为了您的产品反馈体验更及时，建议在产品前端做一下号码格式校验与反馈
name	是	string	utf8，姓名（真实姓名，和身份证号匹配）
quality	否	string	判断图片中的人脸质量是否符合条件。use表示需要做质量控制，质量不符合条件的照片会被直接拒绝
quality_conf	否	string	人脸质量检测中每一项指标的具体阈值设定，json串形式，当指定quality:use时生效
faceliveness	否	string	判断活体值是否达标。use表示需要做活体检测，低于活体阈值的照片会直接拒绝
faceliveness_conf	否	string	人脸活体检测的阈值设定，json串形式，当指定faceliveness:use时生效。默认使用的阈值如下：{faceliveness：0.834963}
ext_fields	否	string	可选项为faceliveness，qualities。选择具体的项，则返回参数中将会显示相应的扩展字段。如faceliveness表示返回结果中包含活体相关内容，qualities表示返回结果中包含质量检测相关内容

身份验证 返回数据参数详情

参数	必须	类型	说明
log_id	是	uint64	日志id
result	是	float	与公安小图相似度可能性，用于验证生活照与公安小图是否为同一人，有正常分数时为[0~1]，推荐阈值0.8，超过即判断为同一人
ext_info	否	string	拓展信息json串，只有选择了ext_fields时才会返回具体信息。选择faceliveness返回具体活体分值信息，选择qualities返回人脸质量检测信息。两者可以同时选择，半角逗号分割。
+faceliveness	否	string	活体检测值，单帧图片建议阈值，小于此值则认为不是活体，超过则判断为活体
+qualities	否	string	质量检测结果
++occlusion	否	string	人脸遮挡情况
+++left_eye	否	string	左眼被遮挡的比例，取值范围[0~1]，数值越大遮挡越多；小于阈值时有返回，默认阈值0.6
+++right_eye	否	string	右眼被遮挡的比例，取值范围[0~1]，数值越大遮挡越多；小于阈值时有返回，默认阈值0.6
+++nose	否	string	鼻子被遮挡的比例，取值范围[0~1]，数值越大遮挡越多；小于阈值时有返回，默认阈值0.7
+++mouth	否	string	嘴巴被遮挡的比例，取值范围[0~1]，数值越大遮挡越多；小于阈值时有返回，默认阈值0.7
+++left_cheek	否	string	左脸颊被遮挡的比例，取值范围[0~1]，数值越大遮挡越多；小于阈值时有返回，默认阈值0.8
+++right_cheek	否	string	右脸颊被遮挡的比例，取值范围[0~1]，数值越大遮挡越多；小于阈值时有返回，默认阈值0.8
+++chin	否	string	下巴被遮挡的比例，取值范围[0~1]，数值越大遮挡越多；小于阈值时有返回，默认阈值0.8
++blur	否	string	人脸模糊度阈值，取值范围[0~1]，数值越大越模糊；小于阈值时有返回，默认阈值0.7
++illumination	否	string	脸部光照的灰度值阈值，取值范围[0~255]，数值越大光照越强；大于阈值时有返回，默认阈值30
++completeness	否	string	人脸完整度，0或1，0为人脸溢出图像边界，1为人脸都在图像边界内

身份验证 返回示例

```
{
 "result":0.03419,
 "ext_info":{
   "faceliveness":0.834963
 },
 "log_id":772889134072410
}
```

在线活体检测

人脸基础信息，人脸质量检测，基于图片的活体检测

```
  """ 读取图片 """
def get_file_content(filePath):
    with open(filePath, 'rb') as fp:
        return fp.read()

image = get_file_content('example.jpg')

""" 调用在线活体检测 """
client.faceverify(image);

""" 如果有可选参数 """
options = {}
options["max_face_num"] = 2
options["face_fields"] = "qualities"

""" 带参数调用在线活体检测 """
client.faceverify(image, options)
```

在线活体检测 请求参数详情

参数名称	是否必选	类型	默认值	说明
image	是	string		图像数据，base64编码，要求base64编码后大小不超过4M，最短边至少15px，最长边最大4096px,支持jpg/png/bmp格式
max_face_num	否	string	1	最多处理人脸数目，默认值1
face_fields	否	string		如不选择此项，返回结果默认只有人脸框、概率和旋转角度。可选参数为qualities、faceliveness。qualities：图片质量相关判断；faceliveness：活体判断。如果两个参数都需要选择，请使用半角逗号分隔。

在线活体检测 返回数据参数详情

参数	类型	是否必须	说明
log_id	uint64	是	日志id
result_num	uint32	是	人脸数目
result	object[]	是	人脸属性对象的集合
+faceliveness	float	否	活体分数，face_fields包括qualities时返回
+location	bject	是	人脸在图片中的位置
++left	uint32	是	人脸区域离左边界的距离
++top	uint32	是	人脸区域离上边界的距离
++width	uint32	是	人脸区域的宽度
++height	uint32	是	人脸区域的高度
+face_probability	double	是	人脸置信度，范围0-1
+rotation_angle	int32	是	人脸框相对于竖直方向的顺时针旋转角，[-180,180]
+yaw	double	是	三维旋转之左右旋转角[-90(左), 90(右)]
+pitch	double	是	三维旋转之俯仰角度[-90(上), 90(下)]
+roll	double	是	平面内旋转角[-180(逆时针), 180(顺时针)]
+qualities	object	否	人脸质量信息。face_fields包含qualities时返回
++occlusion	object	是	人脸各部分遮挡的概率，区间[0, 1]
+++left_eye	double	是	左眼
+++right_eye	double	是	右眼
+++nose	double	是	鼻子
+++mouth	double	是	嘴
+++left_cheek	double	是	左脸颊
+++right_cheek	double	是	右脸颊
+++chin	double	是	下巴
++blur	double	是	人脸模糊程度，[0, 1]。0表示清晰，1表示模糊
++illumination	double	是	取值范围在[0,255],表示脸部区域的光照程度
++completeness	double	是	人脸完整度，[0, 1]。0表示完整，1表示不完整
++type	object	是	真实人脸/卡通人脸置信度
+++human	double	是	真实人脸置信度，[0, 1]
+++cartoon	double	是	卡通人脸置信度，[0, 1]

在线活体检测 返回示例

```
{
    log_id: 1900901488032821,
    result_num: 1,
    result: [
        {
            rotation_angle: 10,
            yaw: 11.357421875,
            faceliveness: 8.1253347161692e-05,
            location: {
                width: 96,
                top: 73,
                height: 96,
                left: 283
            },
            qualities: {
                illumination: 211,
                occlusion: {
                    right_eye: 0,
                    left_eye: 0.039751552045345,
                    left_cheek: 0.12623985111713,
                    mouth: 0,
                    nose: 0,
                    chin: 0.037661049515009,
                    right_cheek: 0.024720622226596
                },
                completeness: 1,
                type: {
                    cartoon: 0,
                    human: 0
                },
                blur: 2.5251445032182e-11
            },
            pitch: 1.0063140392303,
            roll: 12.760620117188,
            face_probability: 1
        }
    ]
}
```


错误信息

错误返回格式

若请求错误，服务器将返回的JSON文本包含以下参数：

- **error_code**：错误码。
- **error_msg**：错误描述信息，帮助理解和解决发生的错误。

错误码

服务端返回的错误码

错误码	错误信息	描述
4	Open api request limit reached	集群超限额
14	IAM Certification failed	IAM鉴权失败，建议用户参照文档自查生成sign的方式是否正确，或换用控制台中ak sk的方式调用
17	Open api daily request limit reached	每天流量超限额
18	Open api qps request limit reached	QPS超限额
19	Open api total request limit reached	请求总量超限额
100	Invalid parameter	无效参数
110	Access token invalid or no longer valid	Access Token失效
111	Access token expired	Access token过期
216015	module closed	模块关闭
216100	invalid param	参数异常
216101	not enough param	缺少必须的参数
216102	service not support	请求了不支持的服务，请检查调用的url
216103	param too long	请求超长，一般为一次传入图片个数超过系统限制
216110	appid not exist	appid不存在，请重新检查后台应用列表中的应用信息
216111	invalid userid	userid信息非法，请检查对应的参数
216200	empty image	图片为空或者base64解码错误
216201	image format error	图片格式错误
216202	image size error	图片大小错误
216300	db error	数据库异常，**少量发生时重试即可**
216400	backend error	后端识别服务异常，可以根据具体msg查看错误原因
216401	internal error	内部错误
216402	face not found	未找到人脸，请检查图片是否含有人脸
216500	unknown error	未知错误
216611	user not exist	用户不存在，请确认该用户是否注册或注册已经生效(**需要已经注册超过5s**)
216613	fail to delete user record	删除用户图片记录失败，重试即可
216614	not enough images	两两比对中图片数少于2张，无法比较
216615	fail to process images	服务处理该图片失败，发生后重试即可
216616	image existed	图片已存在
216617	fail to add user	新增用户图片失败
216618	no user in group	组内用户为空，确认该group是否存在或已经生效(需要已经注册超过5s)
216631	request add user overlimit	本次请求添加的用户数量超限

C#语言

简介

Hi，您好，欢迎使用百度人脸识别服务。

本文档主要针对C#开发者，描述百度人脸识别接口服务的相关技术内容。如果您对文档内容有任何疑问，可以通过以下几种方式联系我们：

- 在百度云控制台内提交工单，咨询问题类型请选择**人工智能服务**；
- 如有疑问，进入AI社区交流：http://ai.baidu.com/forum/topic/list/165

接口能力

接口名称	接口能力简要描述
人脸检测	检测人脸并定位，返回五官关键点，及人脸各属性值
人脸比对	返回两两比对的人脸相似值
人脸查找	在一个人脸集合中找到找到相似的人脸，由一系列接口组成，包括人脸识别、人脸认证、人脸库管理相关接口（人脸注册、人脸更新、人脸删除、用户信息查询、组列表查询、组内用户列表查询、组间复制用户、组内删除用户）

版本更新记录

上线日期	版本号	更新内容
2018.4.9	3.4.0	新增.Net Core支持；新增身份证验证接口，在线活体检测接口
2018.1.11	3.3.1	新增人脸识别N:M接口
2017.12.21	3.3.0	接口升级
2017.11.14	3.2.1	人脸检测接口升级
2017.9.12	3.0.0	更新SDK打包方式：所有AI服务集成一个SDK
2017.4.1	1.0	第一版！

快速入门

安装人脸 C# SDK

C# SDK 现已开源! https://github.com/Baidu-AIP/dotnet-sdk

支持平台：.Net Framework 3.5 4.0 4.5， .Net Core 2.0

方法一：使用Nuget管理依赖 （推荐） 在NuGet中搜索 `Baidu.AI`，安装最新版即可。

packet地址 https://www.nuget.org/packages/Baidu.AI/

方法二：下载安装

人脸 C# SDK目录结构

```
Baidu.Aip
  ├─── net35
  │    ├─── AipSdk.dll           // 百度AI服务 windows 动态库
  │    ├─── AipSdk.xml           // 注释文件
  │    └─── Newtonsoft.Json.dll  // 第三方依赖
  ├─── net40
  ├─── net45
  └─── netstandard2.0
     ├─── AipSdk.deps.json
     └─── AipSdk.dll
```

> 如果需要在 Unity 平台使用，可引用工程源码自行编译。

安装

1.在官方网站下载C# SDK压缩工具包。

2.解压后，将 `AipSdk.dll` 和 `Newtonsoft.Json.dll` 中添加为引用。

新建交互类

Baidu.Aip.Face.Face是人脸的交互类，为使用人脸的开发人员提供了一系列的交互方法。

用户可以参考如下代码新建一个交互类：

```
 // 设置APPID/AK/SK
var APP_ID = "你的 App ID";
var API_KEY = "你的 Api Key";
var SECRET_KEY = "你的 Secret Key";

var client = new Baidu.Aip.Face.Face(API_KEY, SECRET_KEY);
client.Timeout = 60000;  // 修改超时时间
```

在上面代码中，常量`APP_ID`在百度云控制台中创建，常量`API_KEY`与`SECRET_KEY`是在创建完毕应用后，系统分配给用户的，均为字符串，用于标识用户，为访问做签名验证，可在AI服务控制台中的**应用列表**中查看。

注意：如您以前是百度云的老用户，其中`API_KEY`对应百度云的“Access Key ID”，`SECRET_KEY`对应百度云的“Access Key Secret”。

接口说明

人脸检测

检测请求图片中的人脸，返回人脸位置、72个关键点坐标、及人脸相关属性信息。

检测响应速度，与图片中人脸数量相关，人脸数量较多时响应时间会有些许延长。

典型应用场景：如**人脸属性分析**，**基于人脸关键点的加工分析**，**人脸营销活动**等。

> 五官位置会标记具体坐标；72个关键点坐标也包含具体坐标，但不包含对应位置的详细位置描述。

```
 public void DetectDemo() {
var image = File.ReadAllBytes("图片文件路径");
// 调用人脸检测，可能会抛出网络等异常，请使用try/catch捕获
var result = client.Detect(image);
Console.WriteLine(result);
// 如果有可选参数
var options = new Dictionary<string, object>{
    {"max_face_num", 2},
    {"face_fields", "age"}
};
// 带参数调用人脸检测
result = client.Detect(image, options);
Console.WriteLine(result);
}
```

人脸检测 请求参数详情

参数名称	是否必选	类型	默认值	说明
image	是	byte[]		二进制图像数据
max_face_num	否	string	1	最多处理人脸数目，默认值1
face_fields	否	string		包括age,beauty,expression,faceshape,gender,glasses,landmark,qualities信息，逗号分隔，默认只返回人脸框、概率和旋转角度

人脸检测 返回数据参数详情

参数	类型	必选	说明
log_id	uint64	是	日志id
result_num	uint32	是	人脸数目
result	object[]	是	人脸属性对象的集合
+age	double	否	年龄。face_fields包含age时返回
+beauty	double	否	美丑打分，范围0-100，越大表示越美。face_fields包含beauty时返回
+location	object	是	人脸在图片中的位置
++left	uint32	是	人脸区域离左边界的距离
++top	uint32	是	人脸区域离上边界的距离
++width	uint32	是	人脸区域的宽度
++height	uint32	是	人脸区域的高度
+face_probability	double	是	人脸置信度，范围0-1
+rotation_angle	int32	是	人脸框相对于竖直方向的顺时针旋转角，[-180,180]
+yaw	double	是	三维旋转之左右旋转角[-90(左), 90(右)]
+pitch	double	是	三维旋转之俯仰角度[-90(上), 90(下)]
+roll	double	是	平面内旋转角[-180(逆时针), 180(顺时针)]
+expression	uint32	否	表情，0，不笑；1，微笑；2，大笑。face_fields包含expression时返回
+expression_probability	double	否	表情置信度，范围0~1。face_fields包含expression时返回
+faceshape	object[]	否	脸型置信度。face_fields包含faceshape时返回
++type	string	是	脸型：square/triangle/oval/heart/round
++probability	double	是	置信度：0~1
+gender	string	否	male、female。face_fields包含gender时返回
+gender_probability	double	否	性别置信度，范围0~1。face_fields包含gender时返回
+glasses	uint32	否	是否带眼镜，0-无眼镜，1-普通眼镜，2-墨镜。face_fields包含glasses时返回
+glasses_probability	double	否	眼镜置信度，范围0~1。face_fields包含glasses时返回
+landmark	object[]	否	4个关键点位置，左眼中心、右眼中心、鼻尖、嘴中心。face_fields包含landmark时返回
++x	uint32	否	x坐标
++y	uint32	否	y坐标
+landmark72	object[]	否	72个特征点位置，示例图 。face_fields包含landmark时返回
++x	uint32	否	x坐标

++y	uint32	否	y坐标
+qualities	object	否	人脸质量信息。face_fields包含qualities时返回
++occlusion	object	是	人脸各部分遮挡的概率,[0, 1],0表示完整，1表示不完整
+++left_eye	double	是	左眼
+++right_eye	double	是	右眼
+++nose	double	是	鼻子
+++mouth	double	是	嘴
+++left_cheek	double	是	左脸颊
+++right_cheek	double	是	右脸颊
+++chin	double	是	下巴
++blur	double	是	人脸模糊程度，[0, 1]。0表示清晰，1表示模糊
++illumination	-	是	取值范围在[0,255],表示脸部区域的光照程度
++completeness	-	是	人脸完整度，[0, 1]。0表示完整，1表示不完整
++type	object	是	真实人脸/卡通人脸置信度
+++human	-	是	真实人脸置信度，[0, 1]
+++cartoon	-	是	卡通人脸置信度，[0, 1]

人脸检测 返回示例

```
{
  "result_num": 1,
  "result": [
    {
      "location": {
        "left": 117,
        "top": 131,
        "width": 172,
        "height": 170
      },
      "face_probability": 1,
      "rotation_angle": 2,
      "yaw": -0.34859421849251,
      "pitch": 2.3033397197723,
      "roll": 1.9135693311691,
      "landmark": [
        {
          "x": 161.74819946289,
          "y": 163.30244445801
        },
        ...
      ],
      "landmark72": [
        {
          "x": 115.86531066895,
          "y": 170.0546875
        } ,
        ...
      ],
      "age": 29.298097610474,
      "beauty": 55.128883361816,
      "expression": 1,
      "expression_probablity": 0.5543018579483,
      "gender": "male",
      "gender_probability": 0.99979132413864,
      "glasses": 0,
      "glasses_probability": 0.99999964237213,
      "qualities": {
        "occlusion": {
          "left_eye": 0,
          "right_eye": 0,
          "nose": 0,
          "mouth": 0,
          "left_cheek": 0.0064102564938366,
          "right_cheek": 0.0057411273010075,
          "chin": 0
        },
        "blur": 1.1886881756684e-10,
        "illumination": 141,
        "completeness": 1,
        "type": {
          "human": 0.99935841560364,
          "cartoon": 0.00064159056637436
        }
      }
    }
  ],
  "log_id": 2493878179101621
}
```


质量判断

可通过人脸检测接口，基于以下字段和对应阈值，进行质量检测的判断，以保证人脸质量符合后续业务操作要求。

指标	字段与解释	推荐数值界限
遮挡范围	**occlusion**（0~1），0为无遮挡，1是完全遮挡 含有多个具体子字段，表示脸部多个部位 通常用作判断头发、墨镜、口罩等遮挡	left_eye：0.6, #左眼被遮挡的阈值 right_eye：0.6, #右眼被遮挡的阈值 nose：0.7, #鼻子被遮挡的阈值 mouth：0.7, #嘴巴被遮挡的阈值 left_check：0.8, #左脸颊被遮挡的阈值 right_check：0.8, #右脸颊被遮挡的阈值 chin_contour：0.6, #下巴被遮挡阈值
模糊度范围	**Blur**（0~1），0是最清晰，1是最模糊	小于0.7
光照范围	**illumination**（0~255） 脸部光照的灰度值，0表示光照不好 以及对应客户端SDK中，YUV的Y分量	大于40
姿态角度	**Pitch**：三维旋转之俯仰角度[-90(上), 90(下)] **Roll**：平面内旋转角[-180(逆时针), 180(顺时针)] **Yaw**：三维旋转之左右旋转角[-90(左), 90(右)]	分别小于20度
人脸完整度	**completeness**（0或1），0代表完整，1代表不完整	小于0.4
人脸大小	人脸部分的大小 建议长宽像素值范围：80*80~200*200	人脸部分不小于**100*100**像素

人脸比对

该请求用于比对多张图片中的人脸相似度并返回两两比对的得分，可用于判断两张脸是否是同一人的可能性大小。

典型应用场景：如**人证合一验证**，**用户认证**等，可与您现有的人脸库进行比对验证。

> **说明：**支持对比对的两张图片做在线活体检测

```
 public void MatchDemo() {
var images = new[]
  {
      File.ReadAllBytes("图片文件路径"),
      File.ReadAllBytes("图片文件路径")
  };
// 调用人脸比对，可能会抛出网络等异常，请使用try/catch捕获
var result = client.Match(images);
Console.WriteLine(result);
// 如果有可选参数
var options = new Dictionary<string, object>{
   {"ext_fields", "qualities"},
   {"image_liveness", ",faceliveness"},
   {"types", "7,13"}
};
// 带参数调用人脸比对
result = client.Match(images, options);
Console.WriteLine(result);
}
```

人脸比对 请求参数详情

参数名称	是否必选	类型	可选值范围	说明
images	是	IEnumerable<byte[]>		二进制图像数据
ext_fields	否	string		返回质量信息，取值固定:目前支持qualities(质量检测)。(对所有图片都会做改处理)
image_liveness	否	string	faceliveness,faceliveness - 对比对的两张图片都做活体检测 ,faceliveness - 对第一张图片不做活体检测、第二张图做活体检测 faceliveness, - 对第一张图片做活体检测、第二张图不做活体检测	返回的活体信息，“faceliveness,faceliveness” 表示对比对的两张图片都做活体检测；“,faceliveness” 表示对第一张图片不做活体检测、第二张图做活体检测；“faceliveness,” 表示对第一张图片做活体检测、第二张图不做活体检测； **注：需要用于判断活体的图片，图片中的人脸像素面积需要不小于100px*100px，人脸长宽与图片长宽比例，不小于1/3**
types	否	string		请求对比的两张图片的类型，示例：“7,13” **12**表示带水印证件照：一般为带水印的小图，如公安网小图 **7**表示生活照：通常为手机、相机拍摄的人像图片、或从网络获取的人像图片等 **13**表示证件照片：如拍摄的身份证、工卡、护照、学生证等证件图片，**注**：需要确保人脸部分不可太小，通常为100px*100px

人脸比对 返回数据参数详情

字段	必选	类型	说明
log_id	是	uint64	请求唯一标识码，随机数
result_num	是	uint32	返回结果数目，即：result数组中元素个数
result	是	array(object)	结果数据，index和请求图片index对应。数组元素为每张图片的匹配得分数组，top n。 得分[0,100.0]
+index_i	是	uint32	比对图片1的index
+index_j	是	uint32	比对图片2的index
+score	是	double	比对得分
ext_info	否	array（dict）	对应参数中的ext_fields
+qualities	否	string	质量相关的信息，无特殊需求可以不使用
+faceliveness	否	string	活体分数，如0.49999。单帧活体检测参考阈值**0.393241**，超过此分值以上则可认为是活体。**注意：活体检测接口主要用于判断是否为二次翻拍，需要限制用户为当场拍照获取图片**；推荐配合客户端SDK有动作校验活体使用

人脸比对 返回示例

```
 //请求两张图片
{
    "log_id": 73473737,
    "result_num":1,
    "result": [
        {
            "index_i": 0,
            "index_j": 1,
            "score": 44.3
        }
    ]
}
```

人脸识别

用于计算指定组内用户，与上传图像中人脸的相似度。识别前提为您已经创建了一个**人脸库**。

典型应用场景：如**人脸闸机**，**考勤签到**，**安防监控**等。

> **说明：**人脸识别返回值不直接判断是否是同一人，只返回用户信息及相似度分值。

> **说明：**推荐可判断为同一人的相似度分值为**80**，您也可以根据业务需求选择更合适的阈值。

```
 public void IdentifyDemo() {
 var groupId = "group1,group2";

 var image = File.ReadAllBytes("图片文件路径");
 // 调用人脸识别，可能会抛出网络等异常，请使用try/catch捕获
 var result = client.Identify(groupId, image);
 Console.WriteLine(result);
 // 如果有可选参数
 var options = new Dictionary<string, object>{
    {"ext_fields", "faceliveness"},
    {"user_top_num", 3}
 };
 // 带参数调用人脸识别
 result = client.Identify(groupId, image, options);
 Console.WriteLine(result);
}
```

人脸识别 请求参数详情

参数名称	是否必选	类型	默认值	说明
group_id	是	string		用户组id，标识一组用户（由数字、字母、下划线组成），长度限制128B。如果需要将一个uid注册到多个group下，group_id需要用多个逗号分隔，每个group_id长度限制为48个英文字符。**注：group无需单独创建，注册用户时则会自动创建group。** **产品建议**：根据您的业务需求，可以将需要注册的用户，按照业务划分，分配到不同的group下，例如按照会员手机尾号作为groupid，用于刷脸支付、会员计费消费等，这样可以尽可能控制每个group下的用户数与人脸数，提升检索的准确率
image	是	byte[]		二进制图像数据
ext_fields	否	string		特殊返回信息，多个用逗号分隔，取值固定: 目前支持faceliveness(活体检测)。**注：需要用于判断活体的图片，图片中的人脸像素面积需要不小于100px*100px，人脸长宽与图片长宽比例，不小于1/3**
user_top_num	否	string	1	返回用户top数，默认为1，最多返回5个

人脸识别 返回数据参数详情

字段	必选	类型	说明
log_id	是	uint64	请求唯一标识码，随机数
result_num	是	uint32	返回结果数目，即：result数组中元素个数
ext_info	否	array	对应参数中的ext_fields
+faceliveness	否	string	活体分数，如0.49999。单帧活体检测参考阈值0.393241，超过此分值以上则可认为是活体。**注意：活体检测接口主要用于判断是否为二次翻拍，需要限制用户为当场拍照获取图片**；推荐配合客户端SDK有动作校验活体使用
result	是	array(object)	结果数组
+group_id	是	string	对应的这个用户的group_id
+uid	是	string	匹配到的用户id
+user_info	是	string	注册时的用户信息
+scores	是	array(double)	结果数组，数组元素为匹配得分，top n。得分[0,100.0]

人脸识别 返回示例

```
{
  "log_id": 73473737,
  "result_num":1,
  "result": [
    {
      "group_id" : "test1",
      "uid": "u333333",
      "user_info": "Test User",
      "scores": [
            99.3,
            83.4
      ]
    }
  ]
}
```

人脸认证

用于识别上传的图片是否为指定用户，即查找前需要先确定要查找的用户在人脸库中的id。

典型应用场景：如**人脸登录**，**人脸签到**等

> **说明：**人脸认证与人脸识别的差别在于：人脸识别需要指定一个待查找的人脸库中的组；而人脸认证需要指定具体的用户id即可，不需要指定具体的人脸库中的组；实际应用中，人脸认证需要用户或系统先输入id，这增加了验证安全度，但也增加了复杂度，具体使用哪个接口需要视您的业务场景判断。

> **说明：**请求参数中，新增在线活体检测

```
 public void VerifyDemo() {
var uid = "user1";

var groupId = "group1,group2";

var image = File.ReadAllBytes("图片文件路径");
// 调用人脸认证，可能会抛出网络等异常，请使用try/catch捕获
var result = client.Verify(uid, groupId, image);
Console.WriteLine(result);
// 如果有可选参数
var options = new Dictionary<string, object>{
    {"top_num", 3},
    {"ext_fields", "faceliveness"}
};
// 带参数调用人脸认证
result = client.Verify(uid, groupId, image, options);
Console.WriteLine(result);
}
```

人脸认证 请求参数详情

参数名称	是否必选	类型	默认值	说明
uid	是	string		用户id（由数字、字母、下划线组成），长度限制128B
group_id	是	string		用户组id，标识一组用户（由数字、字母、下划线组成），长度限制128B。如果需要将一个uid注册到多个group下，group_id需要用多个逗号分隔，每个group_id长度限制为48个英文字符。**注：group无需单独创建，注册用户时则会自动创建group。** **产品建议**：根据您的业务需求，可以将需要注册的用户，按照业务划分，分配到不同的group下，例如按照会员手机尾号作为groupid，用于刷脸支付、会员计费消费等，这样可以尽可能控制每个group下的用户数与人脸数，提升检索的准确率
image	是	byte[]		二进制图像数据
top_num	否	string	1	返回用户top数，默认为1
ext_fields	否	string		特殊返回信息，多个用逗号分隔，取值固定: 目前支持faceliveness(活体检测)。**注：需要用于判断活体的图片，图片中的人脸像素面积需要不小于100px*100px，人脸长宽与图片长宽比例，不小于1/3**

人脸认证 返回数据参数详情

字段	必选	类型	说明
log_id	是	uint64	请求唯一标识码，随机数
result_num	是	uint32	返回结果数目，即：result数组中元素个数
result	是	array(double)	结果数组，数组元素为匹配得分，top n。 得分范围[0,100.0]。推荐得分超过80可认为认证成功
ext_info	否	array	对应参数中的ext_fields
+faceliveness	否	string	活体分数，如0.49999。单帧活体检测参考阈值0.393241，超过此分值以上则可认为是活体。**活体检测接口主要用于判断是否为二次翻拍，需要限制用户为当场拍照获取图片**；推荐配合客户端SDK有动作校验活体使用

人脸认证 返回示例

```
{
  "log_id": 73473737,
  "result_num":2,
  "result": [
        99.3,
        83.6
  ]
}
```

M:N 识别

待识别的图片中，存在多张人脸的情况下，支持在一个人脸库中，一次请求，同时返回图片中所有人脸的识别结果。

```
 public void MultiIdentifyDemo() {
 var groupId = "group1,group2";

 var image = File.ReadAllBytes("图片文件路径");
 // 调用M:N 识别，可能会抛出网络等异常，请使用try/catch捕获
 var result = client.MultiIdentify(groupId, image);
 Console.WriteLine(result);
 // 如果有可选参数
 var options = new Dictionary<string, object>{
    {"ext_fields", "faceliveness"},
    {"detect_top_num", 3},
    {"user_top_num", 2}
 };
 // 带参数调用M:N 识别
 result = client.MultiIdentify(groupId, image, options);
 Console.WriteLine(result);
}
```

M:N 识别 请求参数详情

参数名称	是否必选	类型	默认值	说明
group_id	是	string		用户组id，标识一组用户（由数字、字母、下划线组成），长度限制128B。如果需要将一个uid注册到多个group下，group_id需要用多个逗号分隔，每个group_id长度限制为48个英文字符。**注：group无需单独创建，注册用户时则会自动创建group。** **产品建议**：根据您的业务需求，可以将需要注册的用户，按照业务划分，分配到不同的group下，例如按照会员手机尾号作为groupid，用于刷脸支付、会员计费消费等，这样可以尽可能控制每个group下的用户数与人脸数，提升检索的准确率
image	是	byte[]		二进制图像数据
ext_fields	否	string		特殊返回信息，多个用逗号分隔，取值固定: 目前支持faceliveness(活体检测)。**注：需要用于判断活体的图片，图片中的人脸像素面积需要不小于100px*100px，人脸长宽与图片长宽比例，不小于1/3**
detect_top_num	否	string	1	检测多少个人脸进行比对，默认值1（最对返回10个）
user_top_num	否	string	1	返回识别结果top人数”，当同一个人有多张图片时，只返回比对最高的1个分数（即，scores参数只有一个值），默认为1（最多返回20个）

M:N 识别 返回数据参数详情

参数	字段	必选	类型	说明
log_id	-	是	uint32	请求标识码，随机数，唯一
result_num	-	是	float	返回结果数目，即：result数组中元素个数（PS：最终返回的个数是小于等于 “实际检测到的人脸数” * “每个人脸匹配的top人数”）
ext_info	-	否	array	对应参数中的ext_fields
-	faceliveness	否	string	活体检测分数，单帧活体检测参考阈值**0.393241**，超过此分值以上则可认为是活体。**注意：活体检测接口主要用于判断是否为二次翻拍，需要限制用户为当场拍照获取图片**；推荐配合客户端SDK有动作校验活体使用
result	-	是	array(object)	-
-	group_id	是	string	对应的这个用户的group_id
-	uid	是	string	匹配到的用户id
-	user_info	是	string	注册时的用户信息
-	scores	是	array(double)	结果数组，数组元素为匹配得分，得分[0,100.0]；个数取决于user_top_num的设定。推荐**80分**以上即可判断为同一人
-	position	是	object	人脸位置，如{top:111,left:222,width:333,height:444,degree:20}

M:N 识别 返回示例


```
{
  "log_id": 73473737,
  "result_num":1,
  "result": [
     {
        "group_id" : "test1",
        "uid": "u333333",
        "user_info": "Test User",
        "position":{"left":726.99188232422,"top":288.37701416016,"width":44,"height":42,"degree":-
4,"prob":0.91117089986801},
        "scores": [
              99.3
        ]
     },
     {
        "group_id" : "test1",
        "uid": "u2222222",
        "user_info": "Test User",
        "position":{"left":726.99188232422,"top":288.37701416016,"width":44,"height":42,"degree":-
4,"prob":0.91117089986801},
        "scores": [
              82.3
        ]
     }
  ]
}
```


人脸注册

用于从人脸库中新增用户，可以设定多个用户所在组，及组内用户的人脸图片，

典型应用场景：构建您的人脸库，如**会员人脸注册**，**已有用户补全人脸信息**等。

人脸库、用户组、用户、用户下的人脸**层级关系**如下所示：

```
|- 人脸库
 |- 用户组一
   |- 用户01
     |- 人脸
   |- 用户02
     |- 人脸
     |- 人脸
     ....
    ....
 |- 用户组二
 |- 用户组三
 |- 用户组四
 ....
```

关于人脸库的设置限制

- 每个开发者账号可以创建100个appid；
- **每个appid对应一个人脸库，且不同appid之间，人脸库互不相通**；
- 每个人脸库下，可以创建多个用户组，用户组（group）数量**没有限制**；
- 每个用户组（group）下，可添加最多**无限**张人脸，**无限**个uid；
- 每个用户（uid）所能注册的最大人脸数量**没有限制**；

为了保证识别效果，请控制注册人脸的质量（通过`/detect`人脸检测接口判断），具体参数可详见下表所示：

质量判断

可通过人脸检测接口，基于以下字段和对应阈值，进行质量检测的判断，以保证人脸质量符合后续业务操作要求。

指标	字段与解释	推荐数值界限
遮挡范围	**occlusion**（0~1），0为无遮挡，1是完全遮挡 含有多个具体子字段，表示脸部多个部位 通常用作判断头发、墨镜、口罩等遮挡	left_eye：0.6, #左眼被遮挡的阈值 right_eye：0.6, #右眼被遮挡的阈值 nose：0.7, #鼻子被遮挡的阈值 mouth：0.7, #嘴巴被遮挡的阈值 left_check：0.8, #左脸颊被遮挡的阈值 right_check：0.8, #右脸颊被遮挡的阈值 chin_contour：0.6, #下巴被遮挡阈值
模糊度范围	**Blur**（0~1），0是最清晰，1是最模糊	小于0.7
光照范围	**illumination**（0~255） 脸部光照的灰度值，0表示光照不好 以及对应客户端SDK中，YUV的Y分量	大于40
姿态角度	**Pitch**：三维旋转之俯仰角度[-90(上), 90(下)] **Roll**：平面内旋转角[-180(逆时针), 180(顺时针)] **Yaw**：三维旋转之左右旋转角[-90(左), 90(右)]	分别小于20度
人脸完整度	**completeness**（0或1），0为人脸溢出图像边界，1为人脸都在图像边界内	视业务逻辑判断
人脸大小	人脸部分的大小 建议长宽像素值范围：80*80~200*200	人脸部分不小于**100*100**像素

```
 public void UserAddDemo() {
var uid = "user1";

var userInfo = "user's info";

var groupId = "group1,group2";

var image = File.ReadAllBytes("图片文件路径");
// 调用人脸注册，可能会抛出网络等异常，请使用try/catch捕获
var result = client.UserAdd(uid, userInfo, groupId, image);
Console.WriteLine(result);
// 如果有可选参数
var options = new Dictionary<string, object>{
    {"action_type", "replace"}
};
// 带参数调用人脸注册
result = client.UserAdd(uid, userInfo, groupId, image, options);
Console.WriteLine(result);
}
```

人脸注册 请求参数详情

参数名称	是否必选	类型	默认值	说明
uid	是	string		用户id（由数字、字母、下划线组成），长度限制128B
user_info	是	string		用户资料，长度限制256B
group_id	是	string		用户组id，标识一组用户（由数字、字母、下划线组成），长度限制128B。如果需要将一个uid注册到多个group下，group_id需要用多个逗号分隔，每个group_id长度限制为48个英文字符。**注：group无需单独创建，注册用户时则会自动创建group。** **产品建议**：根据您的业务需求，可以将需要注册的用户，按照业务划分，分配到不同的group下，例如按照会员手机尾号作为groupid，用于刷脸支付、会员计费消费等，这样可以尽可能控制每个group下的用户数与人脸数，提升检索的准确率
image	是	byte[]		二进制图像数据
action_type	否	string	append	参数包含append、replace。**如果为“replace”，则每次注册时进行替换replace（新增或更新）操作，默认为append操作**。例如：uid在库中已经存在时，对此uid重复注册时，新注册的图片默认会**追加**到该uid下，如果手动选择`action_type:replace`，则会用新图替换库中该uid下所有图片。

人脸注册 返回数据参数详情

字段	是否必选	类型	说明
log_id	是	uint64	请求唯一标识码，随机数

人脸注册 返回示例

```
 // 注册成功
{
   "log_id": 73473737,
}
// 注册发生错误
{
  "error_code": 216616,
  "log_id": 674786177,
  "error_msg": "image exist"
}
```

人脸更新

用于对人脸库中指定用户，更新其下的人脸图像。

> **说明：**针对一个uid执行更新操作，新上传的人脸图像将覆盖此uid原有所有图像。

> **说明：**执行更新操作，如果该uid不存在时，会返回错误。如果添加了action_type:replace,则不会报错，并自动注册该uid，操作结果等同注册新用户。

```
 public void UserUpdateDemo() {
var uid = "user1";

var userInfo = "user's info";

var groupId = "group1";

var image = File.ReadAllBytes("图片文件路径");
// 调用人脸更新，可能会抛出网络等异常，请使用try/catch捕获
var result = client.UserUpdate(uid, userInfo, groupId, image);
Console.WriteLine(result);
// 如果有可选参数
var options = new Dictionary<string, object>{
    {"action_type", "replace"}
};
// 带参数调用人脸更新
result = client.UserUpdate(uid, userInfo, groupId, image, options);
Console.WriteLine(result);
}
```

人脸更新 请求参数详情

参数名称	是否必选	类型	默认值	说明
uid	是	string		用户id（由数字、字母、下划线组成），长度限制128B
user_info	是	string		用户资料，长度限制256B
group_id	是	string		更新指定groupid下uid对应的信息
image	是	byte[]		二进制图像数据
action_type	否	string	append	目前仅支持replace，uid不存在时，不报错，会自动变为注册操作；未选择该参数时，如果uid不存在会提示错误

人脸更新 返回数据参数详情

字段	是否必选	类型	说明
log_id	是	uint64	请求唯一标识码，随机数

人脸更新 返回示例

```
 // 更新成功
{
    "log_id": 73473737,
}
// 更新发生错误
{
  "error_code": 216612,
  "log_id": 1137508902,
  "error_msg": "user not exist"
}
```

人脸删除

用于从人脸库中删除一个用户。

人脸删除注意事项：

- 删除的内容，包括用户所有图像和身份信息；

- 如果一个uid存在于多个用户组内，将会同时将从各个组中把用户删除
- 如果指定了group_id，则只删除此group下的uid相关信息

```
 public void UserDeleteDemo() {
 var uid = "user1";

 // 调用人脸删除，可能会抛出网络等异常，请使用try/catch捕获
 var result = client.UserDelete(uid);
 Console.WriteLine(result);
 // 如果有可选参数
 var options = new Dictionary<string, object>{
    {"group_id", "group1"}
 };
 // 带参数调用人脸删除
 result = client.UserDelete(uid, options);
 Console.WriteLine(result);
}
```

人脸删除 请求参数详情

参数名称	是否必选	类型	说明
uid	是	string	用户id（由数字、字母、下划线组成），长度限制128B
group_id	否	string	删除指定groupid下uid对应的信息

人脸删除 返回数据参数详情

字段	是否必选	类型	说明
log_id	是	uint64	请求唯一标识码，随机数

人脸删除 返回示例

```
 // 删除成功
{
   "log_id": 73473737,
}
// 删除发生错误
{
  "error_code": 216612,
  "log_id": 1137508902,
  "error_msg": "user not exist"
}
```

用户信息查询

用于查询人脸库中某用户的详细信息。

```
 public void UserGetDemo() {
var uid = "user1";

// 调用用户信息查询，可能会抛出网络等异常，请使用try/catch捕获
var result = client.UserGet(uid);
Console.WriteLine(result);
// 如果有可选参数
var options = new Dictionary<string, object>{
    {"group_id", "group1"}
};
// 带参数调用用户信息查询
result = client.UserGet(uid, options);
Console.WriteLine(result);
}
```

用户信息查询 请求参数详情

参数名称	是否必选	类型	说明
uid	是	string	用户id（由数字、字母、下划线组成），长度限制128B
group_id	否	string	选择指定group_id则只查找group列表下的uid内容，如果不指定则查找所有group下对应uid的信息

用户信息查询 返回数据参数详情

字段	是否必选	类型	说明
log_id	是	uint64	请求标识码，随机数，唯一
result	是	array(double)	结果数组
+uid	是	string	匹配到的用户id
+user_info	是	string	注册时的用户信息
+groups	是	array(string)	用户所属组列表

用户信息查询 返回示例

```
 {
“result_num” : 2
   "result": {
      [
         "uid": "testuser2",
         "user_info": "registed user info ...",
         "group_id": "grep1",
      ],
      [
         "uid": "testuser2",
         "user_info": "registed user info2 ...",
         "group_id": "grep2",
      ],
   },
   "log_id": 2979357502
}
```

组列表查询

用于查询用户组的列表。

```
 public void GroupGetlistDemo() {
// 调用组列表查询，可能会抛出网络等异常，请使用try/catch捕获
var result = client.GroupGetlist();
Console.WriteLine(result);
// 如果有可选参数
var options = new Dictionary<string, object>{
    {"start", 0},
    {"num", 50}
};
// 带参数调用组列表查询
result = client.GroupGetlist(options);
Console.WriteLine(result);
}
```

组列表查询 请求参数详情

参数名称	是否必选	类型	默认值	说明
start	否	string	0	默认值0，起始序号
num	否	string	100	返回数量，默认值100，最大值1000

组列表查询 返回数据参数详情

字段	是否必选	类型	说明
result_num	是	number	返回个数
result	是	array(string)	group_id列表

组列表查询 返回示例

```
{
  "result_num": 2,
  "result": [
     "grp1",
     "grp2"
  ],
  "log_id": 3314921889
}
```

组内用户列表查询

用于查询指定用户组中的用户列表。

```
 public void GroupGetusersDemo() {
var groupId = "group1";

// 调用组内用户列表查询，可能会抛出网络等异常，请使用try/catch捕获
var result = client.GroupGetusers(groupId);
Console.WriteLine(result);
// 如果有可选参数
var options = new Dictionary<string, object>{
    {"start", 0},
    {"num", 50}
};
// 带参数调用组内用户列表查询
result = client.GroupGetusers(groupId, options);
Console.WriteLine(result);
}
```

组内用户列表查询 请求参数详情

参数名称	是否必选	类型	默认值	说明
group_id	是	string		用户组id（由数字、字母、下划线组成），长度限制128B
start	否	string	0	默认值0，起始序号
num	否	string	100	返回数量，默认值100，最大值1000

组内用户列表查询 返回数据参数详情

字段	是否必选	类型	说明
log_id	是	uint64	请求标识码，随机数，唯一
result_num	是	uint32	返回个数
result	是	array(object)	user列表
+uid	是	string	用户id
+user_info	是	string	用户信息

组内用户列表查询 返回示例

```
{
  "log_id": 3314921889,
  "result_num": 2,
  "result": [
    {
      "uid": "uid1",
      "user_info": "user info 1"
    },
    {
      "uid": "uid2",
      "user_info": "user info 2"
    }
  ]
}
```

组间复制用户

用于将已经存在于人脸库中的用户**复制到一个新的组**。

> **说明：**并不是向一个指定组内添加用户，而是直接从其它组复制用户信息 如果需要注册用户，请直接使用**人脸注册接口**

```
 public void GroupAdduserDemo() {
var srcGroupId = "group1";

var groupId = "group1,group2";

var uid = "user1";

// 调用组间复制用户，可能会抛出网络等异常，请使用try/catch捕获
var result = client.GroupAdduser(srcGroupId, groupId, uid);
Console.WriteLine(result);
}
```

组间复制用户 请求参数详情

参数名称	是否必选	类型	说明
src_group_id	是	string	从指定group里复制信息
group_id	是	string	用户组id，标识一组用户（由数字、字母、下划线组成），长度限制128B。如果需要将一个uid注册到多个group下，group_id需要用多个逗号分隔，每个group_id长度限制为48个英文字符。**注：group无需单独创建，注册用户时则会自动创建group。** **产品建议**：根据您的业务需求，可以将需要注册的用户，按照业务划分，分配到不同的group下，例如按照会员手机尾号作为groupid，用于刷脸支付、会员计费消费等，这样可以尽可能控制每个group下的用户数与人脸数，提升检索的准确率
uid	是	string	用户id（由数字、字母、下划线组成），长度限制128B

组间复制用户 返回数据参数详情

字段	是否必选	类型	说明
log_id	是	uint64	请求标识码，随机数，唯一

组间复制用户 返回示例

```
 // 正确返回值
{
    "log_id": 3314921889,
}
// 发生错误时返回值
{
  "error_code": 216100,
  "log_id": 3111284097,
  "error_msg": "already add"
}
```

组内删除用户

用于将用户从某个组中删除，但不会删除用户在其它组的信息。

> **说明：**当用户仅属于单个分组时，本接口将返回错误，请使用**人脸删除接口**

```
 public void GroupDeleteuserDemo() {
 var groupId = "group1,group2";

 var uid = "user1";

 // 调用组内删除用户，可能会抛出网络等异常，请使用try/catch捕获
 var result = client.GroupDeleteuser(groupId, uid);
 Console.WriteLine(result);
}
```

组内删除用户 请求参数详情

参数名称	是否必选	类型	说明
group_id	是	string	用户组id，标识一组用户（由数字、字母、下划线组成），长度限制128B。如果需要将一个uid注册到多个group下，group_id需要用多个逗号分隔，每个group_id长度限制为48个英文字符。**注：group无需单独创建，注册用户时则会自动创建group。** **产品建议**：根据您的业务需求，可以将需要注册的用户，按照业务划分，分配到不同的group下，例如按照会员手机尾号作为groupid，用于刷脸支付、会员计费消费等，这样可以尽可能控制每个group下的用户数与人脸数，提升检索的准确率
uid	是	string	用户id（由数字、字母、下划线组成），长度限制128B

组内删除用户 返回数据参数详情

字段	是否必选	类型	说明
log_id	是	uint64	请求标识码，随机数，唯一

组内删除用户 返回示例

```
 // 正确返回值
{
    "log_id": 3314921889,
}
// 发生错误时返回值
{
  "error_code": 216619,
  "log_id": 815967402,
  "error_msg": "user must be in one group at least"
}
```

身份验证

质量检测（可选）活体检测（可选）公安验证（必选）

```
 public void PersonVerifyDemo() {
var image = File.ReadAllBytes("图片文件路径");
var idCardNumber = "110233112299822211";

var name = "张三";

// 调用身份验证，可能会抛出网络等异常，请使用try/catch捕获
var result = client.PersonVerify(image, idCardNumber, name);
Console.WriteLine(result);
// 如果有可选参数
var options = new Dictionary<string, object>{
    {"quality", "use"},
    {"quality_conf", "{\"left_eye\": 0.6, \"right_eye\": 0.6}"},
    {"faceliveness", "use"},
    {"faceliveness_conf", "{\"faceliveness\": 0.834963}"},
    {"ext_fields", "qualities"}
};
// 带参数调用身份验证
result = client.PersonVerify(image, idCardNumber, name, options);
Console.WriteLine(result);
}
```

身份验证 请求参数详情

参数名称	是否必选	类型	说明
image	是	byte[]	二进制图像数据
id_card_number	是	string	身份证号（真实身份证号号码）。我们的服务端会做格式校验，并通过错误码返回，但是为了您的产品反馈体验更及时，建议在产品前端做一下号码格式校验与反馈
name	是	string	utf8，姓名（真实姓名，和身份证号匹配）
quality	否	string	判断图片中的人脸质量是否符合条件。use表示需要做质量控制，质量不符合条件的照片会被直接拒绝
quality_conf	否	string	人脸质量检测中每一项指标的具体阈值设定，json串形式，当指定quality:use时生效
faceliveness	否	string	判断活体值是否达标。use表示需要做活体检测，低于活体阈值的照片会直接拒绝
faceliveness_conf	否	string	人脸活体检测的阈值设定，json串形式，当指定faceliveness:use时生效。默认使用的阈值如下：{faceliveness：0.834963}
ext_fields	否	string	可选项为faceliveness，qualities。选择具体的项，则返回参数中将会显示相应的扩展字段。如faceliveness表示返回结果中包含活体相关内容，qualities表示返回结果中包含质量检测相关内容

身份验证 返回数据参数详情

参数	必须	类型	说明
log_id	是	uint64	日志id
result	是	float	与公安小图相似度可能性，用于验证生活照与公安小图是否为同一人，有正常分数时为[0~1]，推荐阈值0.8，超过即判断为同一人
ext_info	否	string	拓展信息json串，只有选择了ext_fields时才会返回具体信息。选择faceliveness返回具体活体分值信息，选择qualities返回人脸质量检测信息。两者可以同时选择，半角逗号分割。
+faceliveness	否	string	活体检测值，单帧图片建议阈值，小于此值则认为不是活体，超过则判断为活体
+qualities	否	string	质量检测结果
++occlusion	否	string	人脸遮挡情况
+++left_eye	否	string	左眼被遮挡的比例，取值范围[0~1]，数值越大遮挡越多；小于阈值时有返回，默认阈值0.6
+++right_eye	否	string	右眼被遮挡的比例，取值范围[0~1]，数值越大遮挡越多；小于阈值时有返回，默认阈值0.6
+++nose	否	string	鼻子被遮挡的比例，取值范围[0~1]，数值越大遮挡越多；小于阈值时有返回，默认阈值0.7
+++mouth	否	string	嘴巴被遮挡的比例，取值范围[0~1]，数值越大遮挡越多；小于阈值时有返回，默认阈值0.7
+++left_cheek	否	string	左脸颊被遮挡的比例，取值范围[0~1]，数值越大遮挡越多；小于阈值时有返回，默认阈值0.8
+++right_cheek	否	string	右脸颊被遮挡的比例，取值范围[0~1]，数值越大遮挡越多；小于阈值时有返回，默认阈值0.8
+++chin	否	string	下巴被遮挡的比例，取值范围[0~1]，数值越大遮挡越多；小于阈值时有返回，默认阈值0.8
++blur	否	string	人脸模糊度阈值，取值范围[0~1]，数值越大越模糊；小于阈值时有返回，默认阈值0.7
++illumination	否	string	脸部光照的灰度值阈值，取值范围[0~255]，数值越大光照越强；大于阈值时有返回，默认阈值30
++completeness	否	string	人脸完整度，0或1，0为人脸溢出图像边界，1为人脸都在图像边界内

身份验证 返回示例


```
{
 "result":0.03419,
 "ext_info":{
   "faceliveness":0.834963
 },
 "log_id":772889134072410
}
```


在线活体检测

人脸基础信息，人脸质量检测，基于图片的活体检测

```
 public void FaceverifyDemo() {
var image = File.ReadAllBytes("图片文件路径");
// 调用在线活体检测，可能会抛出网络等异常，请使用try/catch捕获
var result = client.Faceverify(image);
Console.WriteLine(result);
// 如果有可选参数
var options = new Dictionary<string, object>{
    {"max_face_num", 2},
    {"face_fields", "qualities"}
};
// 带参数调用在线活体检测
result = client.Faceverify(image, options);
Console.WriteLine(result);
}
```

在线活体检测 请求参数详情

参数名称	是否必选	类型	默认值	说明
image	是	byte[]		二进制图像数据
max_face_num	否	string	1	最多处理人脸数目，默认值1
face_fields	否	string		如不选择此项，返回结果默认只有人脸框、概率和旋转角度。可选参数为qualities、faceliveness。qualities：图片质量相关判断；faceliveness：活体判断。如果两个参数都需要选择，请使用半角逗号分隔。

在线活体检测 返回数据参数详情

参数	类型	是否必须	说明
log_id	uint64	是	日志id
result_num	uint32	是	人脸数目
result	object[]	是	人脸属性对象的集合
+faceliveness	float	否	活体分数，face_fields包括qualities时返回
+location	bject	是	人脸在图片中的位置
++left	uint32	是	人脸区域离左边界的距离
++top	uint32	是	人脸区域离上边界的距离
++width	uint32	是	人脸区域的宽度
++height	uint32	是	人脸区域的高度
+face_probability	double	是	人脸置信度，范围0-1
+rotation_angle	int32	是	人脸框相对于竖直方向的顺时针旋转角，[-180,180]
+yaw	double	是	三维旋转之左右旋转角[-90(左), 90(右)]
+pitch	double	是	三维旋转之俯仰角度[-90(上), 90(下)]
+roll	double	是	平面内旋转角[-180(逆时针), 180(顺时针)]
+qualities	object	否	人脸质量信息。face_fields包含qualities时返回
++occlusion	object	是	人脸各部分遮挡的概率，区间[0, 1]
+++left_eye	double	是	左眼
+++right_eye	double	是	右眼
+++nose	double	是	鼻子
+++mouth	double	是	嘴
+++left_cheek	double	是	左脸颊
+++right_cheek	double	是	右脸颊
+++chin	double	是	下巴
++blur	double	是	人脸模糊程度，[0, 1]。0表示清晰，1表示模糊
++illumination	double	是	取值范围在[0,255],表示脸部区域的光照程度
++completeness	double	是	人脸完整度，[0, 1]。0表示完整，1表示不完整
++type	object	是	真实人脸/卡通人脸置信度
+++human	double	是	真实人脸置信度，[0, 1]
+++cartoon	double	是	卡通人脸置信度，[0, 1]

在线活体检测 返回示例

```
{
  log_id: 1900901488032821,
  result_num: 1,
  result: [
    {
      rotation_angle: 10,
      yaw: 11.357421875,
      faceliveness: 8.1253347161692e-05,
      location: {
        width: 96,
        top: 73,
        height: 96,
        left: 283
      },
      qualities: {
        illumination: 211,
        occlusion: {
          right_eye: 0,
          left_eye: 0.039751552045345,
          left_cheek: 0.12623985111713,
          mouth: 0,
          nose: 0,
          chin: 0.037661049515009,
          right_cheek: 0.024720622226596
        },
        completeness: 1,
        type: {
          cartoon: 0,
          human: 0
        },
        blur: 2.5251445032182e-11
      },
      pitch: 1.0063140392303,
      roll: 12.760620117188,
      face_probability: 1
    }
  ]
}
```


FAQ

1. throw exception "fail to fetch token: 基础连接已关闭"

- 检查网络连接
- 检查网络是否有代理

2. SSL报错 "The authentication or decryption has failed" 可能由于网络代理等原因导致证书不正确，属于常见的网络问题，可以参考这个答案

3. 调用接口，等待一段时间后出现超时 可以设置Timeout参数。

4. 在.Net Core下运行报错 'GBK' is not a supported encoding name' 添加System.Text.Encoding.CodePages引用，并注册。`Encoding.RegisterProvider(CodePagesEncodingProvider.Instance)`

错误信息

错误返回格式

若请求错误，服务器将返回的JSON文本包含以下参数：

- **error_code**：错误码。
- **error_msg**：错误描述信息，帮助理解和解决发生的错误。

错误码

错误码	错误信息	描述
4	Open api request limit reached	集群超限额
14	IAM Certification failed	IAM鉴权失败，建议用户参照文档自查生成sign的方式是否正确，或换用控制台中ak sk的方式调用
17	Open api daily request limit reached	每天流量超限额
18	Open api qps request limit reached	QPS超限额
19	Open api total request limit reached	请求总量超限额
100	Invalid parameter	无效参数
110	Access token invalid or no longer valid	Access Token失效
111	Access token expired	Access token过期
216015	module closed	模块关闭
216100	invalid param	参数异常
216101	not enough param	缺少必须的参数
216102	service not support	请求了不支持的服务，请检查调用的url
216103	param too long	请求超长，一般为一次传入图片个数超过系统限制
216110	appid not exist	appid不存在，请重新检查后台应用列表中的应用信息
216111	invalid userid	userid信息非法，请检查对应的参数
216200	empty image	图片为空或者base64解码错误
216201	image format error	图片格式错误
216202	image size error	图片大小错误
216300	db error	数据库异常，**少量发生时重试即可**
216400	backend error	后端识别服务异常，可以根据具体msg查看错误原因
216401	internal error	内部错误
216402	face not found	未找到人脸，请检查图片是否含有人脸
216500	unknown error	未知错误
216611	user not exist	用户不存在，请确认该用户是否注册或注册已经生效(**需要已经注册超过5s**)
216613	fail to delete user record	删除用户图片记录失败，重试即可
216614	not enough images	两两比对中图片数少于2张，无法比较
216615	fail to process images	服务处理该图片失败，发生后重试即可
216616	image existed	图片已存在
216617	fail to add user	新增用户图片失败
216618	no user in group	组内用户为空，确认该group是否存在或已经生效(需要已经注册超过5s)
216631	request add user overlimit	本次请求添加的用户数量超限

C++语言

简介

Hi，您好，欢迎使用百度人脸识别服务。

本文档主要针对C++开发者，描述百度人脸识别接口服务的相关技术内容。如果您对文档内容有任何疑问，可以通过以下几种方式联系我们：

- 在百度云控制台内提交工单，咨询问题类型请选择**人工智能服务**；
- 如有疑问，进入AI社区交流：http://ai.baidu.com/forum/topic/list/165

接口能力

接口名称	接口能力简要描述
人脸检测	检测人脸并定位，返回五官关键点，及人脸各属性值
人脸比对	返回两两比对的人脸相似值
人脸查找	在一个人脸集合中找到找到相似的人脸，由一系列接口组成，包括人脸识别、人脸认证、人脸库管理相关接口（人脸注册、人脸更新、人脸删除、用户信息查询、组列表查询、组内用户列表查询、组间复制用户、组内删除用户）

版本更新记录

上线日期	版本号	更新内容
2018.4.9	0.6.0	新增身份验证，在线活体检测接口
2018.1.12	0.5.0	新增M：N多人脸识别接口
2017.12.21	0.4.0	更改了人脸认证,更新用户,人脸更新,组件复制用户的接口参数
2017.11.14	0.3.1	人脸检测接口升级到v2
2017.11.9	0.3.0	初始化参数修改
2017.10.31	0.1.0	人脸识别第一版

快速入门

安装人脸识别 C++ SDK

人脸识别 C++ SDK目录结构

```
├─── base
│   ├─── base.h                    // 请求客户端基类
│   ├─── base64.h                  // base64加密相关类
│   ├─── http.h                    // http请求封装类
│   └─── utils.h                   // 工具类
└─── face.h                  // 人脸识别 交互类
```

最低支持 C++ 11+

直接使用开发包步骤如下：

1.在官方网站下载C++ SDK压缩包。

2.将下载的`aip-cpp-sdk-version.zip`解压，其中文件为包含实现代码的头文件。

3.安装依赖库libcurl（需要支持https） openssl jsoncpp(>1.6.2版本，0.x版本将不被支持)。

4.编译工程时添加 C++11 支持 (gcc/clang 添加编译参数 -std=c++11), 添加第三方库链接参数 lcurl, lcrypto, ljsoncpp。

5.在源码中include face.h ，引入压缩包中的头文件以使用aip命名空间下的类和方法。

新建client

client是人脸识别的C++客户端，为使用人脸识别的开发人员提供了一系列的交互方法。当您引入了相应头文件后就可以新建一个client对象

用户可以参考如下代码新建一个client：

```
 #include "face.h"

// 设置APPID/AK/SK
std::string app_id = "你的 App ID";
std::string api_key = "你的 Api key";
std::string secret_key = "你的 Secret Key";

aip::Face client(app_id, api_key, secret_key);
```

在上面代码中，常量APP_ID在百度云控制台中创建，常量API_KEY与SECRET_KEY是在创建完毕应用后，系统分配给用户的，均为字符串，用于标识用户，为访问做签名验证，可在AI服务控制台中的**应用列表**中查看。

注意：如您以前是百度云的老用户，其中API_KEY对应百度云的“Access Key ID”，SECRET_KEY对应百度云的“Access Key Secret”。

接口说明

人脸检测

检测请求图片中的人脸，返回人脸位置、72个关键点坐标、及人脸相关属性信息。

检测响应速度，与图片中人脸数量相关，人脸数量较多时响应时间会有些许延长。

典型应用场景：如**人脸属性分析**，**基于人脸关键点的加工分析**，**人脸营销活动**等。

> 五官位置会标记具体坐标；72个关键点坐标也包含具体坐标，但不包含对应位置的详细位置描述。

```
 Json::Value result;

std::string image;
aip::get_file_content("/assets/sample.jpg", &image);

// 调用人脸检测
result = client.detect(image, aip::null);

// 如果有可选参数
std::map<std::string, std::string> options;
options["max_face_num"] = "2";
options["face_fields"] = "age";

// 带参数调用人脸检测
result = client.detect(image, options);
```

人脸检测 请求参数详情

参数名称	是否必选	类型	默认值	说明
image	是	std::string		图片数据的二进制字符串，可以使用aip::get_file_content函数获取
max_face_num	否	std::string	1	最多处理人脸数目，默认值1
face_fields	否	std::string		包括age,beauty,expression,faceshape,gender,glasses,landmark,qualities信息，逗号分隔，默认只返回人脸框、概率和旋转角度

人脸检测 返回数据参数详情

参数	类型	必选	说明
log_id	uint64	是	日志id
result_num	uint32	是	人脸数目
result	object[]	是	人脸属性对象的集合
+age	double	否	年龄。face_fields包含age时返回
+beauty	double	否	美丑打分，范围0-100，越大表示越美。face_fields包含beauty时返回
+location	object	是	人脸在图片中的位置
++left	uint32	是	人脸区域离左边界的距离
++top	uint32	是	人脸区域离上边界的距离
++width	uint32	是	人脸区域的宽度
++height	uint32	是	人脸区域的高度
+face_probability	double	是	人脸置信度，范围0-1
+rotation_angle	int32	是	人脸框相对于竖直方向的顺时针旋转角，[-180,180]
+yaw	double	是	三维旋转之左右旋转角[-90(左), 90(右)]
+pitch	double	是	三维旋转之俯仰角度[-90(上), 90(下)]
+roll	double	是	平面内旋转角[-180(逆时针), 180(顺时针)]
+expression	uint32	否	表情，0，不笑；1，微笑；2，大笑。face_fields包含expression时返回
+expression_probability	double	否	表情置信度，范围0~1。face_fields包含expression时返回
+faceshape	object[]	否	脸型置信度。face_fields包含faceshape时返回
++type	string	是	脸型：square/triangle/oval/heart/round
++probability	double	是	置信度：0~1
+gender	string	否	male、female。face_fields包含gender时返回
+gender_probability	double	否	性别置信度，范围0~1。face_fields包含gender时返回
+glasses	uint32	否	是否带眼镜，0-无眼镜，1-普通眼镜，2-墨镜。face_fields包含glasses时返回
+glasses_probability	double	否	眼镜置信度，范围0~1。face_fields包含glasses时返回
+landmark	object[]	否	4个关键点位置，左眼中心、右眼中心、鼻尖、嘴中心。face_fields包含landmark时返回
++x	uint32	否	x坐标
++y	uint32	否	y坐标
+landmark72	object[]	否	72个特征点位置，示例图 。face_fields包含landmark时返回

++x	uint32	否	x坐标
++y	uint32	否	y坐标
+qualities	object	否	人脸质量信息。face_fields包含qualities时返回
++occlusion	object	是	人脸各部分遮挡的概率,[0, 1],0表示完整，1表示不完整
+++left_eye	double	是	左眼
+++right_eye	double	是	右眼
+++nose	double	是	鼻子
+++mouth	double	是	嘴
+++left_cheek	double	是	左脸颊
+++right_cheek	double	是	右脸颊
+++chin	double	是	下巴
++blur	double	是	人脸模糊程度，[0, 1]。0表示清晰，1表示模糊
++illumination	-	是	取值范围在[0,255],表示脸部区域的光照程度
++completeness	-	是	人脸完整度，[0, 1]。0表示完整，1表示不完整
++type	object	是	真实人脸/卡通人脸置信度
+++human	-	是	真实人脸置信度，[0, 1]
+++cartoon	-	是	卡通人脸置信度，[0, 1]

人脸检测 返回示例

```
{
  "result_num": 1,
  "result": [
    {
      "location": {
        "left": 117,
        "top": 131,
        "width": 172,
        "height": 170
      },
      "face_probability": 1,
      "rotation_angle": 2,
      "yaw": -0.34859421849251,
      "pitch": 2.3033397197723,
      "roll": 1.9135693311691,
      "landmark": [
        {
          "x": 161.74819946289,
          "y": 163.30244445801
        },
        ...
      ],
      "landmark72": [
        {
          "x": 115.86531066895,
          "y": 170.0546875
        } ,
        ...
      ],
      "age": 29.298097610474,
      "beauty": 55.128883361816,
      "expression": 1,
      "expression_probablity": 0.5543018579483,
      "gender": "male",
      "gender_probability": 0.99979132413864,
      "glasses": 0,
      "glasses_probability": 0.99999964237213,
      "qualities": {
        "occlusion": {
          "left_eye": 0,
          "right_eye": 0,
          "nose": 0,
          "mouth": 0,
          "left_cheek": 0.0064102564938366,
          "right_cheek": 0.0057411273010075,
          "chin": 0
        },
        "blur": 1.1886881756684e-10,
        "illumination": 141,
        "completeness": 1,
        "type": {
          "human": 0.99935841560364,
          "cartoon": 0.00064159056637436
        }
      }
    }
  ],
  "log_id": 2493878179101621
}
```


质量判断

可通过人脸检测接口，基于以下字段和对应阈值，进行质量检测的判断，以保证人脸质量符合后续业务操作要求。

指标	字段与解释	推荐数值界限
遮挡范围	**occlusion**（0~1），0为无遮挡，1是完全遮挡 含有多个具体子字段，表示脸部多个部位 通常用作判断头发、墨镜、口罩等遮挡	left_eye：0.6, #左眼被遮挡的阈值 right_eye：0.6, #右眼被遮挡的阈值 nose：0.7, #鼻子被遮挡的阈值 mouth：0.7, #嘴巴被遮挡的阈值 left_check：0.8, #左脸颊被遮挡的阈值 right_check：0.8, #右脸颊被遮挡的阈值 chin_contour：0.6, #下巴被遮挡阈值
模糊度范围	**Blur**（0~1），0是最清晰，1是最模糊	小于0.7
光照范围	**illumination**（0~255） 脸部光照的灰度值，0表示光照不好 以及对应客户端SDK中，YUV的Y分量	大于40
姿态角度	**Pitch**：三维旋转之俯仰角度[-90(上), 90(下)] **Roll**：平面内旋转角[-180(逆时针), 180(顺时针)] **Yaw**：三维旋转之左右旋转角[-90(左), 90(右)]	分别小于20度
人脸完整度	**completeness**（0或1），0代表完整，1代表不完整	小于0.4
人脸大小	人脸部分的大小 建议长宽像素值范围：80*80~200*200	人脸部分不小于**100*100**像素

人脸比对

该请求用于比对多张图片中的人脸相似度并返回两两比对的得分，可用于判断两张脸是否是同一人的可能性大小。

典型应用场景：如**人证合一验证**，**用户认证**等，可与您现有的人脸库进行比对验证。

> **说明：**支持对比对的两张图片做在线活体检测

```
 Json::Value result;

std::vector<std::string> images;
std::string temp_file_str;
aip::get_file_content("/assets/sample1.jpg", &temp_file_str);
images.emplace_back(std::move(temp_file_str));
aip::get_file_content("/assets/sample2.jpg", &temp_file_str);
images.emplace_back(std::move(temp_file_str));

// 调用人脸比对
result = client.match(images, aip::null);

// 如果有可选参数
std::map<std::string, std::string> options;
options["ext_fields"] = "qualities";
options["image_liveness"] = ",faceliveness";
options["types"] = "7,13";

// 带参数调用人脸比对
result = client.match(images, options);
```

人脸比对 请求参数详情

参数名称	是否必选	类型	可选值范围	说明
images	是	std::vector<std::string>		图片数据的二进制字符串组成的vector，
可以使用aip::get_file_content函数获取单个图片信息，然后加入vector容器				
ext_fields	否	std::string		返回质量信息，取值固定:目前支持qualities(质量检测)。(对所有图片都会做改处理)
image_liveness	否	std::string	faceliveness,faceliveness - 对比对的两张图片都做活体检测,faceliveness - 对第一张图片不做活体检测、第二张图做活体检测faceliveness, - 对第一张图片做活体检测、第二张图不做活体检测	返回的活体信息，“faceliveness,faceliveness” 表示对比对的两张图片都做活体检测；“,faceliveness” 表示对第一张图片不做活体检测、第二张图做活体检 测；“faceliveness,” 表示对第一张图片做活体检测、第二张图不做活体检测； **注：需要用于判断活体的图片，图片中的人脸像素面积需要不小于100px*100px，人脸长宽与图片长宽比例，不小于1/3**
types	否	std::string		请求对比的两张图片的类型，示例：“7,13” **12**表示带水印证件照：一般为带水印的小图，如公安网小图 **7**表示生活照：通常为手机、相机拍摄的人像图片、或从网络获取的人像图片等 **13**表示证件照片：如拍摄的身份证、工卡、护照、学生证等证件图片，**注**：需要确保人脸部分不可太小，通常为100px*100px

人脸比对 返回数据参数详情

字段	必选	类型	说明
log_id	是	uint64	请求唯一标识码，随机数
result_num	是	uint32	返回结果数目，即：result数组中元素个数
result	是	array(object)	结果数据，index和请求图片index对应。数组元素为每张图片的匹配得分数组，top n。 得分[0,100.0]
+index_i	是	uint32	比对图片1的index
+index_j	是	uint32	比对图片2的index
+score	是	double	比对得分
ext_info	否	array（dict）	对应参数中的ext_fields
+qualities	否	string	质量相关的信息，无特殊需求可以不使用
+faceliveness	否	string	活体分数，如0.49999。单帧活体检测参考阈值**0.393241**，超过此分值以上则可认为是活体。**注意：活体检测接口主要用于判断是否为二次翻拍，需要限制用户为当场拍照获取图片**；推荐配合客户端SDK有动作校验活体使用

人脸比对 返回示例

```
 //请求两张图片
{
   "log_id": 73473737,
   "result_num":1,
   "result": [
      {
         "index_i": 0,
         "index_j": 1,
         "score": 44.3
      }
   ]
}
```

人脸识别

用于计算指定组内用户，与上传图像中人脸的相似度。识别前提为您已经创建了一个**人脸库**。

典型应用场景：如**人脸闸机**，**考勤签到**，**安防监控**等。

> **说明：**人脸识别返回值不直接判断是否是同一人，只返回用户信息及相似度分值。

> **说明：**推荐可判断为同一人的相似度分值为**80**，您也可以根据业务需求选择更合适的阈值。

```
 Json::Value result;

std::string group_id = "group1,group2";

std::string image;
aip::get_file_content("/assets/sample.jpg", &image);

// 调用人脸识别
result = client.identify(group_id, image, aip::null);

// 如果有可选参数
std::map<std::string, std::string> options;
options["ext_fields"] = "faceliveness";
options["user_top_num"] = "3";

// 带参数调用人脸识别
result = client.identify(group_id, image, options);
```

人脸识别 请求参数详情

参数名称	是否必选	类型	默认值	说明
group_id	是	std::string		用户组id，标识一组用户（由数字、字母、下划线组成），长度限制128B。如果需要将一个uid注册到多个group下，group_id需要用多个逗号分隔，每个group_id长度限制为48个英文字符。**注：group无需单独创建，注册用户时则会自动创建group。** **产品建议**：根据您的业务需求，可以将需要注册的用户，按照业务划分，分配到不同的group下，例如按照会员手机尾号作为groupid，用于刷脸支付、会员计费消费等，这样可以尽可能控制每个group下的用户数与人脸数，提升检索的准确率
image	是	std::string		图片数据的二进制字符串，可以使用aip::get_file_content函数获取
ext_fields	否	std::string		特殊返回信息，多个用逗号分隔，取值固定: 目前支持faceliveness(活体检测)。**注：需要用于判断活体的图片，图片中的人脸像素面积需要不小于100px*100px，人脸长宽与图片长宽比例，不小于1/3**
user_top_num	否	std::string	1	返回用户top数，默认为1，最多返回5个

人脸识别 返回数据参数详情

字段	必选	类型	说明
log_id	是	uint64	请求唯一标识码，随机数
result_num	是	uint32	返回结果数目，即：result数组中元素个数
ext_info	否	array	对应参数中的ext_fields
+faceliveness	否	string	活体分数，如0.49999。单帧活体检测参考阈值0.393241，超过此分值以上则可认为是活体。**注意：活体检测接口主要用于判断是否为二次翻拍，需要限制用户为当场拍照获取图片**；推荐配合客户端SDK有动作校验活体使用
result	是	array(object)	结果数组
+group_id	是	string	对应的这个用户的group_id
+uid	是	string	匹配到的用户id
+user_info	是	string	注册时的用户信息
+scores	是	array(double)	结果数组，数组元素为匹配得分，top n。得分[0,100.0]

人脸识别 返回示例

```
{
  "log_id": 73473737,
  "result_num":1,
  "result": [
     {
        "group_id" : "test1",
        "uid": "u333333",
        "user_info": "Test User",
        "scores": [
              99.3,
              83.4
        ]
     }
  ]
}
```

人脸认证

用于识别上传的图片是否为指定用户，即查找前需要先确定要查找的用户在人脸库中的id。

典型应用场景：如**人脸登录**，**人脸签到**等

> **说明：**人脸认证与人脸识别的差别在于：人脸识别需要指定一个待查找的人脸库中的组；而人脸认证需要指定具体的用户id即可，不需要指定具体的人脸库中的组；实际应用中，人脸认证需要用户或系统先输入id，这增加了验证安全度，但也增加了复杂度，具体使用哪个接口需要视您的业务场景判断。

> **说明：**请求参数中，新增在线活体检测

```
 Json::Value result;

std::string uid = "user1";

std::string group_id = "group1,group2";

std::string image;
aip::get_file_content("/assets/sample.jpg", &image);

// 调用人脸认证
result = client.verify(uid, group_id, image, aip::null);

// 如果有可选参数
std::map<std::string, std::string> options;
options["top_num"] = "3";
options["ext_fields"] = "faceliveness";

// 带参数调用人脸认证
result = client.verify(uid, group_id, image, options);
```

人脸认证 请求参数详情

参数名称	是否必选	类型	默认值	说明
uid	是	std::string		用户id（由数字、字母、下划线组成），长度限制128B
group_id	是	std::string		用户组id，标识一组用户（由数字、字母、下划线组成），长度限制128B。如果需要将一个uid注册到多个group下，group_id需要用多个逗号分隔，每个group_id长度限制为48个英文字符。**注：group无需单独创建，注册用户时则会自动创建group。** **产品建议**：根据您的业务需求，可以将需要注册的用户，按照业务划分，分配到不同的group下，例如按照会员手机尾号作为groupid，用于刷脸支付、会员计费消费等，这样可以尽可能控制每个group下的用户数与人脸数，提升检索的准确率
image	是	std::string		图片数据的二进制字符串，可以使用aip::get_file_content函数获取
top_num	否	std::string	1	返回用户top数，默认为1
ext_fields	否	std::string		特殊返回信息，多个用逗号分隔，取值固定: 目前支持faceliveness(活体检测)。**注：需要用于判断活体的图片，图片中的人脸像素面积需要不小于100px*100px，人脸长宽与图片长宽比例，不小于1/3**

人脸认证 返回数据参数详情

字段	必选	类型	说明
log_id	是	uint64	请求唯一标识码，随机数
result_num	是	uint32	返回结果数目，即：result数组中元素个数
result	是	array(double)	结果数组，数组元素为匹配得分，top n。 得分范围[0,100.0]。推荐得分超过80可认为认证成功
ext_info	否	array	对应参数中的ext_fields
+faceliveness	否	string	活体分数，如0.49999。单帧活体检测参考阈值0.393241，超过此分值以上则可认为是活体。**活体检测接口主要用于判断是否为二次翻拍，需要限制用户为当场拍照获取图片**；推荐配合客户端SDK有动作校验活体使用

人脸认证 返回示例

```
{
  "log_id": 73473737,
  "result_num":2,
  "result": [
        99.3,
        83.6
  ]
}
```

M:N 识别

待识别的图片中，存在多张人脸的情况下，支持在一个人脸库中，一次请求，同时返回图片中所有人脸的识别结果。

```
 Json::Value result;

std::string group_id = "group1,group2";

std::string image;
aip::get_file_content("/assets/sample.jpg", &image);

// 调用M:N 识别
result = client.multi_identify(group_id, image, aip::null);

// 如果有可选参数
std::map<std::string, std::string> options;
options["ext_fields"] = "faceliveness";
options["detect_top_num"] = "3";
options["user_top_num"] = "2";

// 带参数调用M:N 识别
result = client.multi_identify(group_id, image, options);
```

M:N 识别 请求参数详情

参数名称	是否必选	类型	默认值	说明
group_id	是	std::string		用户组id，标识一组用户（由数字、字母、下划线组成），长度限制128B。如果需要将一个uid注册到多个group下，group_id需要用多个逗号分隔，每个group_id长度限制为48个英文字符。**注：group无需单独创建，注册用户时则会自动创建group。** **产品建议**：根据您的业务需求，可以将需要注册的用户，按照业务划分，分配到不同的group下，例如按照会员手机尾号作为groupid，用于刷脸支付、会员计费消费等，这样可以尽可能控制每个group下的用户数与人脸数，提升检索的准确率
image	是	std::string		图片数据的二进制字符串，可以使用aip::get_file_content函数获取
ext_fields	否	std::string		特殊返回信息，多个用逗号分隔，取值固定: 目前支持faceliveness(活体检测)。**注：需要用于判断活体的图片，图片中的人脸像素面积需要不小于100px*100px，人脸长宽与图片长宽比例，不小于1/3**
detect_top_num	否	std::string	1	检测多少个人脸进行比对，默认值1（最对返回10个）
user_top_num	否	std::string	1	返回识别结果top人数”，当同一个人有多张图片时，只返回比对最高的1个分数（即，scores参数只有一个值），默认为1（最多返回20个）

M:N 识别 返回数据参数详情

参数	字段	必选	类型	说明
log_id	-	是	uint32	请求标识码，随机数，唯一
result_num	-	是	float	返回结果数目，即：result数组中元素个数（PS：最终返回的个数是小于等于 “实际检测到的人脸数” * “每个人脸匹配的top人数”）
ext_info	-	否	array	对应参数中的ext_fields
-	faceliveness	否	string	活体检测分数，单帧活体检测参考阈值**0.393241**，超过此分值以上则可认为是活体。**注意：活体检测接口主要用于判断是否为二次翻拍，需要限制用户为当场拍照获取图片**；推荐配合客户端SDK有动作校验活体使用
result	-	是	array(object)	-
-	group_id	是	string	对应的这个用户的group_id
-	uid	是	string	匹配到的用户id
-	user_info	是	string	注册时的用户信息
-	scores	是	array(double)	结果数组，数组元素为匹配得分，得分[0,100.0]；个数取决于user_top_num的设定。推荐**80分**以上即可判断为同一人
-	position	是	object	人脸位置，如{top:111,left:222,width:333,height:444,degree:20}

M:N 识别 返回示例

```
{
  "log_id": 73473737,
  "result_num":1,
  "result": [
     {
        "group_id" : "test1",
        "uid": "u333333",
        "user_info": "Test User",
        "position":{"left":726.99188232422,"top":288.37701416016,"width":44,"height":42,"degree":-
4,"prob":0.91117089986801},
        "scores": [
              99.3
        ]
     },
     {
        "group_id" : "test1",
        "uid": "u2222222",
        "user_info": "Test User",
        "position":{"left":726.99188232422,"top":288.37701416016,"width":44,"height":42,"degree":-
4,"prob":0.91117089986801},
        "scores": [
              82.3
        ]
     }
  ]
}
```


人脸注册

用于从人脸库中新增用户，可以设定多个用户所在组，及组内用户的人脸图片，

典型应用场景：构建您的人脸库，如**会员人脸注册**，**已有用户补全人脸信息**等。

人脸库、用户组、用户、用户下的人脸**层级关系**如下所示：

```
|- 人脸库
 |- 用户组一
   |- 用户01
     |- 人脸
   |- 用户02
     |- 人脸
     |- 人脸
     ....
    ....
 |- 用户组二
 |- 用户组三
 |- 用户组四
 ....
```

关于人脸库的设置限制

- 每个开发者账号可以创建100个appid；
- **每个appid对应一个人脸库，且不同appid之间，人脸库互不相通**；
- 每个人脸库下，可以创建多个用户组，用户组（group）数量**没有限制**；
- 每个用户组（group）下，可添加最多**无限**张人脸，**无限**个uid；
- 每个用户（uid）所能注册的最大人脸数量**没有限制**；

为了保证识别效果，请控制注册人脸的质量（通过/detect人脸检测接口判断），具体参数可详见下表所示：

质量判断

可通过人脸检测接口，基于以下字段和对应阈值，进行质量检测的判断，以保证人脸质量符合后续业务操作要求。

指标	字段与解释	推荐数值界限
遮挡范围	**occlusion**（0~1），0为无遮挡，1是完全遮挡 含有多个具体子字段，表示脸部多个部位 通常用作判断头发、墨镜、口罩等遮挡	left_eye：0.6, #左眼被遮挡的阈值 right_eye：0.6, #右眼被遮挡的阈值 nose：0.7, #鼻子被遮挡的阈值 mouth：0.7, #嘴巴被遮挡的阈值 left_check：0.8, #左脸颊被遮挡的阈值 right_check：0.8, #右脸颊被遮挡的阈值 chin_contour：0.6, #下巴被遮挡阈值
模糊度范围	**Blur**（0~1），0是最清晰，1是最模糊	小于0.7
光照范围	**illumination**（0~255） 脸部光照的灰度值，0表示光照不好 以及对应客户端SDK中，YUV的Y分量	大于40
姿态角度	**Pitch**：三维旋转之俯仰角度[-90(上), 90(下)] **Roll**：平面内旋转角[-180(逆时针), 180(顺时针)] **Yaw**：三维旋转之左右旋转角[-90(左), 90(右)]	分别小于20度
人脸完整度	**completeness**（0或1），0为人脸溢出图像边界，1为人脸都在图像边界内	视业务逻辑判断
人脸大小	人脸部分的大小 建议长宽像素值范围：80*80~200*200	人脸部分不小于**100*100**像素

```
 Json::Value result;

std::string uid = "user1";

std::string user_info = "user's info";

std::string group_id = "group1,group2";

std::string image;
aip::get_file_content("/assets/sample.jpg", &image);

// 调用人脸注册
result = client.user_add(uid, user_info, group_id, image, aip::null);

// 如果有可选参数
std::map<std::string, std::string> options;
options["action_type"] = "replace";

// 带参数调用人脸注册
result = client.user_add(uid, user_info, group_id, image, options);
```

人脸注册 请求参数详情

参数名称	是否必选	类型	默认值	说明
uid	是	std::string		用户id（由数字、字母、下划线组成），长度限制128B
user_info	是	std::string		用户资料，长度限制256B
group_id	是	std::string		用户组id，标识一组用户（由数字、字母、下划线组成），长度限制128B。如果需要将一个uid注册到多个group下，group_id需要用多个逗号分隔，每个group_id长度限制为48个英文字符。**注：group无需单独创建，注册用户时则会自动创建group。** **产品建议**：根据您的业务需求，可以将需要注册的用户，按照业务划分，分配到不同的group下，例如按照会员手机尾号作为groupid，用于刷脸支付、会员计费消费等，这样可以尽可能控制每个group下的用户数与人脸数，提升检索的准确率
image	是	std::string		图片数据的二进制字符串，可以使用aip::get_file_content函数获取
action_type	否	std::string	append	参数包含append、replace。**如果为“replace”，则每次注册时进行替换replace（新增或更新）操作，默认为append操作**。例如：uid在库中已经存在时，对此uid重复注册时，新注册的图片默认会**追加**到该uid下，如果手动选择`action_type:replace`，则会用新图替换库中该uid下所有图片。

人脸注册 返回数据参数详情

字段	是否必选	类型	说明
log_id	是	uint64	请求唯一标识码，随机数

人脸注册 返回示例

```
 // 注册成功
{
   "log_id": 73473737,
}
// 注册发生错误
{
  "error_code": 216616,
  "log_id": 674786177,
  "error_msg": "image exist"
}
```

人脸更新

用于对人脸库中指定用户，更新其下的人脸图像。

> **说明：**针对一个uid执行更新操作，新上传的人脸图像将覆盖此uid原有所有图像。

> **说明：**执行更新操作，如果该uid不存在时，会返回错误。如果添加了action_type:replace,则不会报错，并自动注册该uid，操作结果等同注册新用户。

```
 Json::Value result;

std::string uid = "user1";

std::string user_info = "user's info";

std::string group_id = "group1";

std::string image;
aip::get_file_content("/assets/sample.jpg", &image);

// 调用人脸更新
result = client.user_update(uid, user_info, group_id, image, aip::null);

// 如果有可选参数
std::map<std::string, std::string> options;
options["action_type"] = "replace";

// 带参数调用人脸更新
result = client.user_update(uid, user_info, group_id, image, options);
```

人脸更新 请求参数详情

参数名称	是否必选	类型	默认值	说明
uid	是	std::string		用户id（由数字、字母、下划线组成），长度限制128B
user_info	是	std::string		用户资料，长度限制256B
group_id	是	std::string		更新指定groupid下uid对应的信息
image	是	std::string		图片数据的二进制字符串，可以使用aip::get_file_content函数获取
action_type	否	std::string	append	目前仅支持replace，uid不存在时，不报错，会自动变为注册操作；未选择该参数时，如果uid不存在会提示错误

人脸更新 返回数据参数详情

字段	是否必选	类型	说明
log_id	是	uint64	请求唯一标识码，随机数

人脸更新 返回示例

```
 // 更新成功
{
    "log_id": 73473737,
}
// 更新发生错误
{
  "error_code": 216612,
  "log_id": 1137508902,
  "error_msg": "user not exist"
}
```

人脸删除

用于从人脸库中删除一个用户。

人脸删除注意事项：

- 删除的内容，包括用户所有图像和身份信息；
- 如果一个uid存在于多个用户组内，将会同时将从各个组中把用户删除
- 如果指定了group_id，则只删除此group下的uid相关信息

```
 Json::Value result;

std::string uid = "user1";

// 调用人脸删除
result = client.user_delete(uid, aip::null);

// 如果有可选参数
std::map<std::string, std::string> options;
options["group_id"] = "group1";

// 带参数调用人脸删除
result = client.user_delete(uid, options);
```

人脸删除 请求参数详情

参数名称	是否必选	类型	说明
uid	是	std::string	用户id（由数字、字母、下划线组成），长度限制128B
group_id	否	std::string	删除指定groupid下uid对应的信息

人脸删除 返回数据参数详情

字段	是否必选	类型	说明
log_id	是	uint64	请求唯一标识码，随机数

人脸删除 返回示例

```
 // 删除成功
{
   "log_id": 73473737,
}
// 删除发生错误
{
  "error_code": 216612,
  "log_id": 1137508902,
  "error_msg": "user not exist"
}
```

用户信息查询

用于查询人脸库中某用户的详细信息。

```
 Json::Value result;

std::string uid = "user1";

// 调用用户信息查询
result = client.user_get(uid, aip::null);

// 如果有可选参数
std::map<std::string, std::string> options;
options["group_id"] = "group1";

// 带参数调用用户信息查询
result = client.user_get(uid, options);
```

用户信息查询 请求参数详情

参数名称	是否必选	类型	说明
uid	是	std::string	用户id（由数字、字母、下划线组成），长度限制128B
group_id	否	std::string	选择指定group_id则只查找group列表下的uid内容，如果不指定则查找所有group下对应uid的信息

用户信息查询 返回数据参数详情

字段	是否必选	类型	说明
log_id	是	uint64	请求标识码，随机数，唯一
result	是	array(double)	结果数组
+uid	是	string	匹配到的用户id
+user_info	是	string	注册时的用户信息
+groups	是	array(string)	用户所属组列表

用户信息查询 返回示例

```
 {
“result_num” : 2
   "result": {
      [
         "uid": "testuser2",
         "user_info": "registed user info ...",
         "group_id": "grep1",
      ],
      [
         "uid": "testuser2",
         "user_info": "registed user info2 ...",
         "group_id": "grep2",
      ],
   },
   "log_id": 2979357502
}
```

组列表查询

用于查询用户组的列表。

```
 Json::Value result;

// 调用组列表查询
result = client.group_getlist(aip::null);

// 如果有可选参数
std::map<std::string, std::string> options;
options["start"] = "0";
options["num"] = "50";

// 带参数调用组列表查询
result = client.group_getlist(options);
```

组列表查询 请求参数详情

参数名称	是否必选	类型	默认值	说明
start	否	std::string	0	默认值0，起始序号
num	否	std::string	100	返回数量，默认值100，最大值1000

组列表查询 返回数据参数详情

字段	是否必选	类型	说明
result_num	是	number	返回个数
result	是	array(string)	group_id列表

组列表查询 返回示例

```
{
  "result_num": 2,
  "result": [
     "grp1",
     "grp2"
  ],
  "log_id": 3314921889
}
```

组内用户列表查询

用于查询指定用户组中的用户列表。

```
 Json::Value result;

std::string group_id = "group1";

// 调用组内用户列表查询
result = client.group_getusers(group_id, aip::null);

// 如果有可选参数
std::map<std::string, std::string> options;
options["start"] = "0";
options["num"] = "50";

// 带参数调用组内用户列表查询
result = client.group_getusers(group_id, options);
```

组内用户列表查询 请求参数详情

参数名称	是否必选	类型	默认值	说明
group_id	是	std::string		用户组id（由数字、字母、下划线组成），长度限制128B
start	否	std::string	0	默认值0，起始序号
num	否	std::string	100	返回数量，默认值100，最大值1000

组内用户列表查询 返回数据参数详情

字段	是否必选	类型	说明
log_id	是	uint64	请求标识码，随机数，唯一
result_num	是	uint32	返回个数
result	是	array(object)	user列表
+uid	是	string	用户id
+user_info	是	string	用户信息

组内用户列表查询 返回示例

```
{
  "log_id": 3314921889,
  "result_num": 2,
  "result": [
    {
      "uid": "uid1",
      "user_info": "user info 1"
    },
    {
      "uid": "uid2",
      "user_info": "user info 2"
    }
  ]
}
```

组间复制用户

用于将已经存在于人脸库中的用户**复制到一个新的组**。

> **说明：**并不是向一个指定组内添加用户，而是直接从其它组复制用户信息 如果需要注册用户，请直接使用**人脸注册接口**

```
 Json::Value result;

std::string src_group_id = "group1";

std::string group_id = "group1,group2";

std::string uid = "user1";

// 调用组间复制用户
result = client.group_adduser(src_group_id, group_id, uid, aip::null);
```

组间复制用户 请求参数详情

参数名称	是否必选	类型	说明
src_group_id	是	std::string	从指定group里复制信息
group_id	是	std::string	用户组id，标识一组用户（由数字、字母、下划线组成），长度限制128B。如果需要将一个uid注册到多个group下，group_id需要用多个逗号分隔，每个group_id长度限制为48个英文字符。**注：group无需单独创建，注册用户时则会自动创建group。** **产品建议**：根据您的业务需求，可以将需要注册的用户，按照业务划分，分配到不同的group下，例如按照会员手机尾号作为groupid，用于刷脸支付、会员计费消费等，这样可以尽可能控制每个group下的用户数与人脸数，提升检索的准确率
uid	是	std::string	用户id（由数字、字母、下划线组成），长度限制128B

组间复制用户 返回数据参数详情

字段	是否必选	类型	说明
log_id	是	uint64	请求标识码，随机数，唯一

组间复制用户 返回示例

```
 // 正确返回值
{
   "log_id": 3314921889,
}
// 发生错误时返回值
{
  "error_code": 216100,
  "log_id": 3111284097,
  "error_msg": "already add"
}
```

组内删除用户

用于将用户从某个组中删除，但不会删除用户在其它组的信息。

> **说明：**当用户仅属于单个分组时，本接口将返回错误，请使用**人脸删除接口**

```
 Json::Value result;

std::string group_id = "group1,group2";

std::string uid = "user1";

// 调用组内删除用户
result = client.group_deleteuser(group_id, uid, aip::null);
```

组内删除用户 请求参数详情

参数名称	是否必选	类型	说明
group_id	是	std::string	用户组id，标识一组用户（由数字、字母、下划线组成），长度限制128B。如果需要将一个uid注册到多个group下，group_id需要用多个逗号分隔，每个group_id长度限制为48个英文字符。**注：group无需单独创建，注册用户时则会自动创建group。** **产品建议**：根据您的业务需求，可以将需要注册的用户，按照业务划分，分配到不同的group下，例如按照会员手机尾号作为groupid，用于刷脸支付、会员计费消费等，这样可以尽可能控制每个group下的用户数与人脸数，提升检索的准确率
uid	是	std::string	用户id（由数字、字母、下划线组成），长度限制128B

组内删除用户 返回数据参数详情

字段	是否必选	类型	说明
log_id	是	uint64	请求标识码，随机数，唯一

组内删除用户 返回示例

```
 // 正确返回值
{
   "log_id": 3314921889,
}
// 发生错误时返回值
{
  "error_code": 216619,
  "log_id": 815967402,
  "error_msg": "user must be in one group at least"
}
```

身份验证

质量检测（可选）活体检测（可选）公安验证（必选）

```
 Json::Value result;

std::string image;
aip::get_file_content("/assets/sample.jpg", &image);

std::string id_card_number = "110233112299822211";

std::string name = "张三";

// 调用身份验证
result = client.person_verify(image, id_card_number, name, aip::null);

// 如果有可选参数
std::map<std::string, std::string> options;
options["quality"] = "use";
options["quality_conf"] = "{\"left_eye\": 0.6, \"right_eye\": 0.6}";
options["faceliveness"] = "use";
options["faceliveness_conf"] = "{\"faceliveness\": 0.834963}";
options["ext_fields"] = "qualities";

// 带参数调用身份验证
result = client.person_verify(image, id_card_number, name, options);
```

身份验证 请求参数详情

参数名称	是否必选	类型	说明
image	是	std::string	图片数据的二进制字符串，可以使用aip::get_file_content函数获取
id_card_number	是	std::string	身份证号（真实身份证号号码）。我们的服务端会做格式校验，并通过错误码返回，但是为了您的产品反馈体验更及时，建议在产品前端做一下号码格式校验与反馈
name	是	std::string	utf8，姓名（真实姓名，和身份证号匹配）
quality	否	std::string	判断图片中的人脸质量是否符合条件。use表示需要做质量控制，质量不符合条件的照片会被直接拒绝
quality_conf	否	std::string	人脸质量检测中每一项指标的具体阈值设定，json串形式，当指定quality:use时生效
faceliveness	否	std::string	判断活体值是否达标。use表示需要做活体检测，低于活体阈值的照片会直接拒绝
faceliveness_conf	否	std::string	人脸活体检测的阈值设定，json串形式，当指定faceliveness:use时生效。默认使用的阈值如下：{faceliveness：0.834963}
ext_fields	否	std::string	可选项为faceliveness，qualities。选择具体的项，则返回参数中将会显示相应的扩展字段。如faceliveness表示返回结果中包含活体相关内容，qualities表示返回结果中包含质量检测相关内容

身份验证 返回数据参数详情

参数	必须	类型	说明
log_id	是	uint64	日志id
result	是	float	与公安小图相似度可能性，用于验证生活照与公安小图是否为同一人，有正常分数时为[0~1]，推荐阈值0.8，超过即判断为同一人
ext_info	否	string	拓展信息json串，只有选择了ext_fields时才会返回具体信息。选择faceliveness返回具体活体分值信息，选择qualities返回人脸质量检测信息。两者可以同时选择，半角逗号分割。
+faceliveness	否	string	活体检测值，单帧图片建议阈值，小于此值则认为不是活体，超过则判断为活体
+qualities	否	string	质量检测结果
++occlusion	否	string	人脸遮挡情况
+++left_eye	否	string	左眼被遮挡的比例，取值范围[0~1]，数值越大遮挡越多；小于阈值时有返回，默认阈值0.6
+++right_eye	否	string	右眼被遮挡的比例，取值范围[0~1]，数值越大遮挡越多；小于阈值时有返回，默认阈值0.6
+++nose	否	string	鼻子被遮挡的比例，取值范围[0~1]，数值越大遮挡越多；小于阈值时有返回，默认阈值0.7
+++mouth	否	string	嘴巴被遮挡的比例，取值范围[0~1]，数值越大遮挡越多；小于阈值时有返回，默认阈值0.7
+++left_cheek	否	string	左脸颊被遮挡的比例，取值范围[0~1]，数值越大遮挡越多；小于阈值时有返回，默认阈值0.8
+++right_cheek	否	string	右脸颊被遮挡的比例，取值范围[0~1]，数值越大遮挡越多；小于阈值时有返回，默认阈值0.8
+++chin	否	string	下巴被遮挡的比例，取值范围[0~1]，数值越大遮挡越多；小于阈值时有返回，默认阈值0.8
++blur	否	string	人脸模糊度阈值，取值范围[0~1]，数值越大越模糊；小于阈值时有返回，默认阈值0.7
++illumination	否	string	脸部光照的灰度值阈值，取值范围[0~255]，数值越大光照越强；大于阈值时有返回，默认阈值30
++completeness	否	string	人脸完整度，0或1, 0为人脸溢出图像边界，1为人脸都在图像边界内

身份验证 返回示例

```
{
 "result":0.03419,
 "ext_info":{
   "faceliveness":0.834963
 },
 "log_id":772889134072410
}
```

在线活体检测

人脸基础信息，人脸质量检测，基于图片的活体检测

```
 Json::Value result;

std::string image;
aip::get_file_content("/assets/sample.jpg", &image);

// 调用在线活体检测
result = client.faceverify(image, aip::null);

// 如果有可选参数
std::map<std::string, std::string> options;
options["max_face_num"] = "2";
options["face_fields"] = "qualities";

// 带参数调用在线活体检测
result = client.faceverify(image, options);
```

在线活体检测 请求参数详情

参数名称	是否必选	类型	默认值	说明
image	是	std::string		图片数据的二进制字符串，可以使用aip::get_file_content函数获取
max_face_num	否	std::string	1	最多处理人脸数目，默认值1
face_fields	否	std::string		如不选择此项，返回结果默认只有人脸框、概率和旋转角度。可选参数为qualities、faceliveness。qualities：图片质量相关判断；faceliveness：活体判断。如果两个参数都需要选择，请使用半角逗号分隔。

在线活体检测 返回数据参数详情

参数	类型	是否必须	说明
log_id	uint64	是	日志id
result_num	uint32	是	人脸数目
result	object[]	是	人脸属性对象的集合
+faceliveness	float	否	活体分数，face_fields包括qualities时返回
+location	bject	是	人脸在图片中的位置
++left	uint32	是	人脸区域离左边界的距离
++top	uint32	是	人脸区域离上边界的距离
++width	uint32	是	人脸区域的宽度
++height	uint32	是	人脸区域的高度
+face_probability	double	是	人脸置信度，范围0-1
+rotation_angle	int32	是	人脸框相对于竖直方向的顺时针旋转角，[-180,180]
+yaw	double	是	三维旋转之左右旋转角[-90(左), 90(右)]
+pitch	double	是	三维旋转之俯仰角度[-90(上), 90(下)]
+roll	double	是	平面内旋转角[-180(逆时针), 180(顺时针)]
+qualities	object	否	人脸质量信息。face_fields包含qualities时返回
++occlusion	object	是	人脸各部分遮挡的概率，区间[0, 1]
+++left_eye	double	是	左眼
+++right_eye	double	是	右眼
+++nose	double	是	鼻子
+++mouth	double	是	嘴
+++left_cheek	double	是	左脸颊
+++right_cheek	double	是	右脸颊
+++chin	double	是	下巴
++blur	double	是	人脸模糊程度，[0, 1]。0表示清晰，1表示模糊
++illumination	double	是	取值范围在[0,255],表示脸部区域的光照程度
++completeness	double	是	人脸完整度，[0, 1]。0表示完整，1表示不完整
++type	object	是	真实人脸/卡通人脸置信度
+++human	double	是	真实人脸置信度，[0, 1]
+++cartoon	double	是	卡通人脸置信度，[0, 1]

在线活体检测 返回示例

```
{
    log_id: 1900901488032821,
    result_num: 1,
    result: [
        {
            rotation_angle: 10,
            yaw: 11.357421875,
            faceliveness: 8.1253347161692e-05,
            location: {
                width: 96,
                top: 73,
                height: 96,
                left: 283
            },
            qualities: {
                illumination: 211,
                occlusion: {
                    right_eye: 0,
                    left_eye: 0.039751552045345,
                    left_cheek: 0.12623985111713,
                    mouth: 0,
                    nose: 0,
                    chin: 0.037661049515009,
                    right_cheek: 0.024720622226596
                },
                completeness: 1,
                type: {
                    cartoon: 0,
                    human: 0
                },
                blur: 2.5251445032182e-11
            },
            pitch: 1.0063140392303,
            roll: 12.760620117188,
            face_probability: 1
        }
    ]
}
```


错误信息

错误返回格式

若请求错误，服务器将返回的JSON文本包含以下参数：

- **error_code**：错误码。
- **error_msg**：错误描述信息，帮助理解和解决发生的错误。

错误码

错误码	错误信息	描述
4	Open api request limit reached	集群超限额
14	IAM Certification failed	IAM鉴权失败，建议用户参照文档自查生成sign的方式是否正确，或换用控制台中ak sk的方式调用
17	Open api daily request limit reached	每天流量超限额
18	Open api qps request limit reached	QPS超限额
19	Open api total request limit reached	请求总量超限额
100	Invalid parameter	无效参数
110	Access token invalid or no longer valid	Access Token失效
111	Access token expired	Access token过期
216015	module closed	模块关闭
216100	invalid param	参数异常
216101	not enough param	缺少必须的参数
216102	service not support	请求了不支持的服务，请检查调用的url
216103	param too long	请求超长，一般为一次传入图片个数超过系统限制
216110	appid not exist	appid不存在，请重新检查后台应用列表中的应用信息
216111	invalid userid	userid信息非法，请检查对应的参数
216200	empty image	图片为空或者base64解码错误
216201	image format error	图片格式错误
216202	image size error	图片大小错误
216300	db error	数据库异常，**少量发生时重试即可**
216400	backend error	后端识别服务异常，可以根据具体msg查看错误原因
216401	internal error	内部错误
216402	face not found	未找到人脸，请检查图片是否含有人脸
216500	unknown error	未知错误
216611	user not exist	用户不存在，请确认该用户是否注册或注册已经生效(**需要已经注册超过5s**)
216613	fail to delete user record	删除用户图片记录失败，重试即可
216614	not enough images	两两比对中图片数少于2张，无法比较
216615	fail to process images	服务处理该图片失败，发生后重试即可
216616	image existed	图片已存在
216617	fail to add user	新增用户图片失败
216618	no user in group	组内用户为空，确认该group是否存在或已经生效(需要已经注册超过5s)
216631	request add user overlimit	本次请求添加的用户数量超限

Nodejs语言

简介

Hi，您好，欢迎使用百度人脸识别服务。

本文档主要针对Nodejs开发者，描述百度人脸识别接口服务的相关技术内容。如果您对文档内容有任何疑问，可以通过以下几种方式联系我们：

- 在百度云控制台内提交工单，咨询问题类型请选择**人工智能服务**；
- 如有疑问，进入AI社区交流：http://ai.baidu.com/forum/topic/list/165

接口能力

接口名称	接口能力简要描述
人脸检测	检测人脸并定位，返回五官关键点，及人脸各属性值
人脸比对	返回两两比对的人脸相似值
人脸查找	在一个人脸集合中找到找到相似的人脸，由一系列接口组成，包括人脸识别、人脸认证、人脸库管理相关接口（人脸注册、人脸更新、人脸删除、用户信息查询、组列表查询、组内用户列表查询、组间复制用户、组内删除用户）

版本更新记录

上线日期	版本号	更新内容
2018.4.9	2.2.0	新增身份验证，在线活体检测接口
2018.1.12	2.1.0	新增M：N多人脸识别接口
2017.12.21	2.0.0	实现代码重构，接口返回标准promise对象
2017.11.14	1.5.1	人脸检测接口升级到v2
2017.5.11	1.0.0	初版

快速入门

安装人脸识别 Node SDK

人脸识别 Node SDK目录结构

```
├─── src
|   ├─── auth                    //授权相关类
|   ├─── http                    //Http通信相关类
|   ├─── client                  //公用类
|   ├─── util                    //工具类
|   └─── const                   //常量类
├─── AipFace.js                 //人脸识别交互类
├─── index.js                   //入口文件
└─── package.json               //npm包描述文件
```

支持 node 版本 4.0+

查看源码 Nodejs SDK代码已开源，您可以查看代码、或者在License范围内修改和编译SDK以适配您的环境。 github链接：https://github.com/Baidu-AIP/nodejs-sdk

直接使用node开发包步骤如下：

1.在官方网站下载node SDK压缩包。

2.将下载的`aip-node-sdk-version.zip`解压后，复制到工程文件夹中。

3.进入目录，运行npm install安装sdk依赖库

4.把目录当做模块依赖

其中，`version`为版本号，添加完成后，用户就可以在工程中使用人脸识别 Node SDK。

直接使用npm安装依赖：

`npm install baidu-aip-sdk`

新建AipFaceClient

AipFaceClient是人脸识别的node客户端，为使用人脸识别的开发人员提供了一系列的交互方法。

用户可以参考如下代码新建一个AipFaceClient：

```
 var AipFaceClient = require("baidu-aip-sdk").face;

// 设置APPID/AK/SK
var APP_ID = "你的 App ID";
var API_KEY = "你的 Api Key";
var SECRET_KEY = "你的 Secret Key";

// 新建一个对象，建议只保存一个对象调用服务接口
var client = new AipFaceClient(APP_ID, API_KEY, SECRET_KEY);
```

为了使开发者更灵活的控制请求，模块提供了设置全局参数和全局请求拦截器的方法；本库发送网络请求依赖的是request模块，因此参数格式与request模块的参数相同 更多参数细节您可以参考request官方参数文档。

```
 var HttpClient = require("baidu-aip-sdk").HttpClient;

// 设置request库的一些参数，例如代理服务地址，超时时间等
// request参数请参考 https://github.com/request/request#requestoptions-callback
HttpClient.setRequestOptions({timeout: 5000});

// 也可以设置拦截每次请求（设置拦截后，调用的setRequestOptions设置的参数将不生效），
// 可以按需修改request参数（无论是否修改，必须返回函数调用参数）
// request参数请参考 https://github.com/request/request#requestoptions-callback
HttpClient.setRequestInterceptor(function(requestOptions) {
    // 查看参数
    console.log(requestOptions)
    // 修改参数
    requestOptions.timeout = 5000;
    // 返回参数
    return requestOptions;
});
```

在上面代码中，常量`APP_ID`在百度云控制台中创建，常量`API_KEY`与`SECRET_KEY`是在创建完毕应用后，系统分配给用户的，均为字符串，用于标识用户，为访问做签名验证，可在AI服务控制台中的**应用列表**中查看。

注意：如您以前是百度云的老用户，其中`API_KEY`对应百度云的“Access Key ID”，`SECRET_KEY`对应百度云的“Access Key Secret”。

接口说明

人脸检测

检测请求图片中的人脸，返回人脸位置、72个关键点坐标、及人脸相关属性信息。

检测响应速度，与图片中人脸数量相关，人脸数量较多时响应时间会有些许延长。

典型应用场景：如**人脸属性分析**，**基于人脸关键点的加工分析**，**人脸营销活动**等。

> 五官位置会标记具体坐标；72个关键点坐标也包含具体坐标，但不包含对应位置的详细位置描述。

```
 var fs = require('fs');

var image = fs.readFileSync("assets/example.jpg").toString("base64");

// 调用人脸检测
client.detect(image).then(function(result) {
    console.log(JSON.stringify(result));
}).catch(function(err) {
    // 如果发生网络错误
    console.log(err);
});

// 如果有可选参数
var options = {};
options["max_face_num"] = "2";
options["face_fields"] = "age";

// 带参数调用人脸检测
client.detect(image, options).then(function(result) {
    console.log(JSON.stringify(result));
}).catch(function(err) {
    // 如果发生网络错误
    console.log(err);
});;
```

人脸检测 请求参数详情

参数名称	是否必选	类型	默认值	说明
image	是	string		图像数据，base64编码，要求base64编码后大小不超过4M，最短边至少15px，最长边最大4096px,支持jpg/png/bmp格式
max_face_num	否	string	1	最多处理人脸数目，默认值1
face_fields	否	string		包括age,beauty,expression,faceshape,gender,glasses,landmark,qualities信息，逗号分隔，默认只返回人脸框、概率和旋转角度

人脸检测 返回数据参数详情

参数	类型	必选	说明
log_id	uint64	是	日志id
result_num	uint32	是	人脸数目
result	object[]	是	人脸属性对象的集合
+age	double	否	年龄。face_fields包含age时返回
+beauty	double	否	美丑打分，范围0-100，越大表示越美。face_fields包含beauty时返回
+location	object	是	人脸在图片中的位置
++left	uint32	是	人脸区域离左边界的距离
++top	uint32	是	人脸区域离上边界的距离
++width	uint32	是	人脸区域的宽度
++height	uint32	是	人脸区域的高度

+face_probability	double	是	人脸置信度，范围0-1
+rotation_angle	int32	是	人脸框相对于竖直方向的顺时针旋转角，[-180,180]
+yaw	double	是	三维旋转之左右旋转角[-90(左), 90(右)]
+pitch	double	是	三维旋转之俯仰角度[-90(上), 90(下)]
+roll	double	是	平面内旋转角[-180(逆时针), 180(顺时针)]
+expression	uint32	否	表情，0，不笑；1，微笑；2，大笑。face_fields包含expression时返回
+expression_probability	double	否	表情置信度，范围0~1。face_fields包含expression时返回
+faceshape	object[]	否	脸型置信度。face_fields包含faceshape时返回
++type	string	是	脸型：square/triangle/oval/heart/round
++probability	double	是	置信度：0~1
+gender	string	否	male、female。face_fields包含gender时返回
+gender_probability	double	否	性别置信度，范围0~1。face_fields包含gender时返回
+glasses	uint32	否	是否带眼镜，0-无眼镜，1-普通眼镜，2-墨镜。face_fields包含glasses时返回
+glasses_probability	double	否	眼镜置信度，范围0~1。face_fields包含glasses时返回
+landmark	object[]	否	4个关键点位置，左眼中心、右眼中心、鼻尖、嘴中心。face_fields包含landmark时返回
++x	uint32	否	x坐标
++y	uint32	否	y坐标
+landmark72	object[]	否	72个特征点位置，示例图 。face_fields包含landmark时返回
++x	uint32	否	x坐标
++y	uint32	否	y坐标
+qualities	object	否	人脸质量信息。face_fields包含qualities时返回
++occlusion	object	是	人脸各部分遮挡的概率,[0, 1],0表示完整，1表示不完整
+++left_eye	double	是	左眼
+++right_eye	double	是	右眼
+++nose	double	是	鼻子
+++mouth	double	是	嘴
+++left_cheek	double	是	左脸颊
+++right_cheek	double	是	右脸颊
+++chin	double	是	下巴
++blur	double	是	人脸模糊程度，[0, 1]。0表示清晰，1表示模糊
++illumination	-	是	取值范围在[0,255],表示脸部区域的光照程度
++completeness	-	是	人脸完整度，[0, 1]。0表示完整，1表示不完整
++type	object	是	真实人脸/卡通人脸置信度
+++human	-	是	真实人脸置信度，[0, 1]
+++cartoon	-	是	卡通人脸置信度，[0, 1]

人脸检测 返回示例

```
{
  "result_num": 1,
  "result": [
    {
      "location": {
        "left": 117,
        "top": 131,
        "width": 172,
        "height": 170
      },
      "face_probability": 1,
      "rotation_angle": 2,
      "yaw": -0.34859421849251,
      "pitch": 2.3033397197723,
      "roll": 1.9135693311691,
      "landmark": [
        {
          "x": 161.74819946289,
          "y": 163.30244445801
        },
        ...
      ],
      "landmark72": [
        {
          "x": 115.86531066895,
          "y": 170.0546875
        } ,
        ...
      ],
      "age": 29.298097610474,
      "beauty": 55.128883361816,
      "expression": 1,
      "expression_probablity": 0.5543018579483,
      "gender": "male",
      "gender_probability": 0.99979132413864,
      "glasses": 0,
      "glasses_probability": 0.99999964237213,
      "qualities": {
        "occlusion": {
          "left_eye": 0,
          "right_eye": 0,
          "nose": 0,
          "mouth": 0,
          "left_cheek": 0.0064102564938366,
          "right_cheek": 0.0057411273010075,
          "chin": 0
        },
        "blur": 1.1886881756684e-10,
        "illumination": 141,
        "completeness": 1,
        "type": {
          "human": 0.99935841560364,
          "cartoon": 0.00064159056637436
        }
      }
    }
  ],
  "log_id": 2493878179101621
}
```


质量判断

可通过人脸检测接口，基于以下字段和对应阈值，进行质量检测的判断，以保证人脸质量符合后续业务操作要求。

指标	字段与解释	推荐数值界限
遮挡范围	**occlusion**（0~1），0为无遮挡，1是完全遮挡 含有多个具体子字段，表示脸部多个部位 通常用作判断头发、墨镜、口罩等遮挡	left_eye：0.6, #左眼被遮挡的阈值 right_eye：0.6, #右眼被遮挡的阈值 nose：0.7, #鼻子被遮挡的阈值 mouth：0.7, #嘴巴被遮挡的阈值 left_check：0.8, #左脸颊被遮挡的阈值 right_check：0.8, #右脸颊被遮挡的阈值 chin_contour：0.6, #下巴被遮挡阈值
模糊度范围	**Blur**（0~1），0是最清晰，1是最模糊	小于0.7
光照范围	**illumination**（0~255） 脸部光照的灰度值，0表示光照不好 以及对应客户端SDK中，YUV的Y分量	大于40
姿态角度	**Pitch**：三维旋转之俯仰角度[-90(上), 90(下)] **Roll**：平面内旋转角[-180(逆时针), 180(顺时针)] **Yaw**：三维旋转之左右旋转角[-90(左), 90(右)]	分别小于20度
人脸完整度	**completeness**（0或1），0代表完整，1代表不完整	小于0.4
人脸大小	人脸部分的大小 建议长宽像素值范围：80*80~200*200	人脸部分不小于**100*100**像素

人脸比对

该请求用于比对多张图片中的人脸相似度并返回两两比对的得分，可用于判断两张脸是否是同一人的可能性大小。

典型应用场景：如**人证合一验证**，**用户认证**等，可与您现有的人脸库进行比对验证。

> **说明：**支持对比对的两张图片做在线活体检测

```
 var fs = require('fs');
var images = [fs.readFileSync("assets/example1.jpg").toString("base64"),
fs.readFileSync("assets/example2.jpg").toString("base64")];

// 调用人脸比对
client.match(images).then(function(result) {
    console.log(JSON.stringify(result));
}).catch(function(err) {
    // 如果发生网络错误
    console.log(err);
});

// 如果有可选参数
var options = {};
options["ext_fields"] = "qualities";
options["image_liveness"] = ",faceliveness";
options["types"] = "7,13";

// 带参数调用人脸比对
client.match(images, options).then(function(result) {
    console.log(JSON.stringify(result));
}).catch(function(err) {
    // 如果发生网络错误
    console.log(err);
});;
```

人脸比对 请求参数详情

参数名称	是否必选	类型	可选值范围	说明
images	是	string		base64编码后的多张图片数据，半角逗号分隔，单次请求总共最大20M
ext_fields	否	string		返回质量信息，取值固定:目前支持qualities(质量检测)。(对所有图片都会做改处理)
image_liveness	否	string	faceliveness,faceliveness - 对比对的两张图片都做活体检测 ,faceliveness - 对第一张图片不做活体检测、第二张图做活体检测 faceliveness, - 对第一张图片做活体检测、第二张图不做活体检测	返回的活体信息，“faceliveness,faceliveness” 表示对比对的两张图片都做活体检测；“,faceliveness” 表示对第一张图片不做活体检测、第二张图做活体检测；“faceliveness,” 表示对第一张图片做活体检测、第二张图不做活体检测； **注：需要用于判断活体的图片，图片中的人脸像素面积需要不小于100px*100px，人脸长宽与图片长宽比例，不小于1/3**
types	否	string		请求对比的两张图片的类型，示例：“7,13” **12**表示带水印证件照：一般为带水印的小图，如公安网小图 **7**表示生活照：通常为手机、相机拍摄的人像图片、或从网络获取的人像图片等 **13**表示证件照片：如拍摄的身份证、工卡、护照、学生证等证件图片，**注**：需要确保人脸部分不可太小，通常为100px*100px

人脸比对 返回数据参数详情

字段	必选	类型	说明
log_id	是	uint64	请求唯一标识码，随机数
result_num	是	uint32	返回结果数目，即：result数组中元素个数
result	是	array(object)	结果数据，index和请求图片index对应。数组元素为每张图片的匹配得分数组，top n。 得分[0,100.0]
+index_i	是	uint32	比对图片1的index
+index_j	是	uint32	比对图片2的index
+score	是	double	比对得分
ext_info	否	array（dict）	对应参数中的ext_fields
+qualities	否	string	质量相关的信息，无特殊需求可以不使用
+faceliveness	否	string	活体分数，如0.49999。单帧活体检测参考阈值**0.393241**，超过此分值以上则可认为是活体。**注意：活体检测接口主要用于判断是否为二次翻拍，需要限制用户为当场拍照获取图片**；推荐配合客户端SDK有动作校验活体使用

人脸比对 返回示例

```
 //请求两张图片
{
    "log_id": 73473737,
    "result_num":1,
    "result": [
        {
            "index_i": 0,
            "index_j": 1,
            "score": 44.3
        }
    ]
}
```

人脸识别

用于计算指定组内用户，与上传图像中人脸的相似度。识别前提为您已经创建了一个**人脸库**。

典型应用场景：如**人脸闸机**，**考勤签到**，**安防监控**等。

> **说明：**人脸识别返回值不直接判断是否是同一人，只返回用户信息及相似度分值。

> **说明：**推荐可判断为同一人的相似度分值为**80**，您也可以根据业务需求选择更合适的阈值。

```
 var fs = require('fs');

var groupId = "group1,group2";

var image = fs.readFileSync("assets/example.jpg").toString("base64");

// 调用人脸识别
client.identifyUser(groupId, image).then(function(result) {
    console.log(JSON.stringify(result));
}).catch(function(err) {
    // 如果发生网络错误
    console.log(err);
});

// 如果有可选参数
var options = {};
options["ext_fields"] = "faceliveness";
options["user_top_num"] = "3";

// 带参数调用人脸识别
client.identifyUser(groupId, image, options).then(function(result) {
    console.log(JSON.stringify(result));
}).catch(function(err) {
    // 如果发生网络错误
    console.log(err);
});;
```

人脸识别 请求参数详情

参数名称	是否必选	类型	默认值	说明
group_id	是	string		用户组id，标识一组用户（由数字、字母、下划线组成），长度限制128B。如果需要将一个uid注册到多个group下，group_id需要用多个逗号分隔，每个group_id长度限制为48个英文字符。**注：group无需单独创建，注册用户时则会自动创建group。** **产品建议**：根据您的业务需求，可以将需要注册的用户，按照业务划分，分配到不同的group下，例如按照会员手机尾号作为groupid，用于刷脸支付、会员计费消费等，这样可以尽可能控制每个group下的用户数与人脸数，提升检索的准确率
image	是	string		图像数据，base64编码，要求base64编码后大小不超过4M，最短边至少15px，最长边最大4096px,支持jpg/png/bmp格式
ext_fields	否	string		特殊返回信息，多个用逗号分隔，取值固定: 目前支持faceliveness(活体检测)。**注：需要用于判断活体的图片，图片中的人脸像素面积需要不小于100px*100px，人脸长宽与图片长宽比例，不小于1/3**
user_top_num	否	string	1	返回用户top数，默认为1，最多返回5个

人脸识别 返回数据参数详情

字段	必选	类型	说明
log_id	是	uint64	请求唯一标识码，随机数
result_num	是	uint32	返回结果数目，即：result数组中元素个数
ext_info	否	array	对应参数中的ext_fields
+faceliveness	否	string	活体分数，如0.49999。单帧活体检测参考阈值0.393241，超过此分值以上则可认为是活体。**注意：活体检测接口主要用于判断是否为二次翻拍，需要限制用户为当场拍照获取图片**；推荐配合客户端SDK有动作校验活体使用
result	是	array(object)	结果数组
+group_id	是	string	对应的这个用户的group_id
+uid	是	string	匹配到的用户id
+user_info	是	string	注册时的用户信息
+scores	是	array(double)	结果数组，数组元素为匹配得分，top n。得分[0,100.0]

人脸识别 返回示例

```
{
  "log_id": 73473737,
  "result_num":1,
  "result": [
     {
        "group_id" : "test1",
        "uid": "u333333",
        "user_info": "Test User",
        "scores": [
              99.3,
              83.4
        ]
     }
  ]
}
```

人脸认证

用于识别上传的图片是否为指定用户，即查找前需要先确定要查找的用户在人脸库中的id。

典型应用场景：如**人脸登录**，**人脸签到**等

> **说明：**人脸认证与人脸识别的差别在于：人脸识别需要指定一个待查找的人脸库中的组；而人脸认证需要指定具体的用户id即可，不需要指定具体的人脸库中的组；实际应用中，人脸认证需要用户或系统先输入id，这增加了验证安全度，但也增加了复杂度，具体使用哪个接口需要视您的业务场景判断。

> **说明：**请求参数中，新增在线活体检测

```
 var fs = require('fs');

var uid = "user1";

var groupId = "group1,group2";

var image = fs.readFileSync("assets/example.jpg").toString("base64");

// 调用人脸认证
client.verifyUser(uid, groupId, image).then(function(result) {
    console.log(JSON.stringify(result));
}).catch(function(err) {
    // 如果发生网络错误
    console.log(err);
});

// 如果有可选参数
var options = {};
options["top_num"] = "3";
options["ext_fields"] = "faceliveness";

// 带参数调用人脸认证
client.verifyUser(uid, groupId, image, options).then(function(result) {
    console.log(JSON.stringify(result));
}).catch(function(err) {
    // 如果发生网络错误
    console.log(err);
});;
```

人脸认证 请求参数详情

参数名称	是否必选	类型	默认值	说明
uid	是	string		用户id（由数字、字母、下划线组成），长度限制128B
group_id	是	string		用户组id，标识一组用户（由数字、字母、下划线组成），长度限制128B。如果需要将一个uid注册到多个group下，group_id需要用多个逗号分隔，每个group_id长度限制为48个英文字符。**注：group无需单独创建，注册用户时则会自动创建group。** **产品建议**：根据您的业务需求，可以将需要注册的用户，按照业务划分，分配到不同的group下，例如按照会员手机尾号作为groupid，用于刷脸支付、会员计费消费等，这样可以尽可能控制每个group下的用户数与人脸数，提升检索的准确率
image	是	string		图像数据，base64编码，要求base64编码后大小不超过4M，最短边至少15px，最长边最大4096px,支持jpg/png/bmp格式
top_num	否	string	1	返回用户top数，默认为1
ext_fields	否	string		特殊返回信息，多个用逗号分隔，取值固定: 目前支持faceliveness(活体检测)。**注：需要用于判断活体的图片，图片中的人脸像素面积需要不小于100px*100px，人脸长宽与图片长宽比例，不小于1/3**

人脸认证 返回数据参数详情

字段	必选	类型	说明
log_id	是	uint64	请求唯一标识码，随机数
result_num	是	uint32	返回结果数目，即：result数组中元素个数
result	是	array(double)	结果数组，数组元素为匹配得分，top n。 得分范围[0,100.0]。推荐得分超过80可认为认证成功
ext_info	否	array	对应参数中的ext_fields
+faceliveness	否	string	活体分数，如0.49999。单帧活体检测参考阈值0.393241，超过此分值以上则可认为是活体。**活体检测接口主要用于判断是否为二次翻拍，需要限制用户为当场拍照获取图片**；推荐配合客户端SDK有动作校验活体使用

人脸认证 返回示例

```
{
  "log_id": 73473737,
  "result_num":2,
  "result": [
        99.3,
        83.6
  ]
}
```

M:N 识别

待识别的图片中，存在多张人脸的情况下，支持在一个人脸库中，一次请求，同时返回图片中所有人脸的识别结果。

```
 var fs = require('fs');

var groupId = "group1,group2";

var image = fs.readFileSync("assets/example.jpg").toString("base64");

// 调用M:N 识别
client.multiIdentify(groupId, image).then(function(result) {
    console.log(JSON.stringify(result));
}).catch(function(err) {
    // 如果发生网络错误
    console.log(err);
});

// 如果有可选参数
var options = {};
options["ext_fields"] = "faceliveness";
options["detect_top_num"] = "3";
options["user_top_num"] = "2";

// 带参数调用M:N 识别
client.multiIdentify(groupId, image, options).then(function(result) {
    console.log(JSON.stringify(result));
}).catch(function(err) {
    // 如果发生网络错误
    console.log(err);
});;
```

M:N 识别 请求参数详情

参数名称	是否必选	类型	默认值	说明
group_id	是	string		用户组id，标识一组用户（由数字、字母、下划线组成），长度限制128B。如果需要将一个uid注册到多个group下，group_id需要用多个逗号分隔，每个group_id长度限制为48个英文字符。**注：group无需单独创建，注册用户时则会自动创建group。** **产品建议**：根据您的业务需求，可以将需要注册的用户，按照业务划分，分配到不同的group下，例如按照会员手机尾号作为groupid，用于刷脸支付、会员计费消费等，这样可以尽可能控制每个group下的用户数与人脸数，提升检索的准确率
image	是	string		图像数据，base64编码，要求base64编码后大小不超过4M，最短边至少15px，最长边最大4096px,支持jpg/png/bmp格式
ext_fields	否	string		特殊返回信息，多个用逗号分隔，取值固定: 目前支持faceliveness(活体检测)。**注：需要用于判断活体的图片，图片中的人脸像素面积需要不小于100px*100px，人脸长宽与图片长宽比例，不小于1/3**
detect_top_num	否	string	1	检测多少个人脸进行比对，默认值1（最对返回10个）
user_top_num	否	string	1	返回识别结果top人数”，当同一个人有多张图片时，只返回比对最高的1个分数（即，scores参数只有一个值），默认为1（最多返回20个）

M:N 识别 返回数据参数详情

参数	字段	必选	类型	说明
log_id	-	是	uint32	请求标识码，随机数，唯一
result_num	-	是	float	返回结果数目，即：result数组中元素个数（PS：最终返回的个数是小于等于 “实际检测到的人脸数” * “每个人脸匹配的top人数”）
ext_info	-	否	array	对应参数中的ext_fields
-	faceliveness	否	string	活体检测分数，单帧活体检测参考阈值**0.393241**，超过此分值以上则可认为是活体。**注意：活体检测接口主要用于判断是否为二次翻拍，需要限制用户为当场拍照获取图片**；推荐配合客户端SDK有动作校验活体使用
result	-	是	array(object)	-
-	group_id	是	string	对应的这个用户的group_id
-	uid	是	string	匹配到的用户id
-	user_info	是	string	注册时的用户信息
-	scores	是	array(double)	结果数组，数组元素为匹配得分，得分[0,100.0]；个数取决于user_top_num的设定。推荐**80分**以上即可判断为同一人
-	position	是	object	人脸位置，如{top:111,left:222,width:333,height:444,degree:20}

M:N 识别 返回示例

```
{
  "log_id": 73473737,
  "result_num":1,
  "result": [
     {
        "group_id" : "test1",
        "uid": "u333333",
        "user_info": "Test User",
        "position":{"left":726.99188232422,"top":288.37701416016,"width":44,"height":42,"degree":-
4,"prob":0.91117089986801},
        "scores": [
              99.3
        ]
     },
     {
        "group_id" : "test1",
        "uid": "u2222222",
        "user_info": "Test User",
        "position":{"left":726.99188232422,"top":288.37701416016,"width":44,"height":42,"degree":-
4,"prob":0.91117089986801},
        "scores": [
              82.3
        ]
     }
  ]
}
```


人脸注册

用于从人脸库中新增用户，可以设定多个用户所在组，及组内用户的人脸图片，

典型应用场景：构建您的人脸库，如**会员人脸注册**，**已有用户补全人脸信息**等。

人脸库、用户组、用户、用户下的人脸**层级关系**如下所示：

```
|- 人脸库
 |- 用户组一
   |- 用户01
     |- 人脸
   |- 用户02
     |- 人脸
     |- 人脸
     ....
    ....
 |- 用户组二
 |- 用户组三
 |- 用户组四
 ....
```

关于人脸库的设置限制

- 每个开发者账号可以创建100个appid；
- **每个appid对应一个人脸库，且不同appid之间，人脸库互不相通**；
- 每个人脸库下，可以创建多个用户组，用户组（group）数量**没有限制**；
- 每个用户组（group）下，可添加最多**无限**张人脸，**无限**个uid；
- 每个用户（uid）所能注册的最大人脸数量**没有限制**；

为了保证识别效果，请控制注册人脸的质量（通过`/detect`人脸检测接口判断），具体参数可详见下表所示：

质量判断

可通过人脸检测接口，基于以下字段和对应阈值，进行质量检测的判断，以保证人脸质量符合后续业务操作要求。

指标	字段与解释	推荐数值界限
遮挡范围	**occlusion**（0~1），0为无遮挡，1是完全遮挡 含有多个具体子字段，表示脸部多个部位 通常用作判断头发、墨镜、口罩等遮挡	left_eye：0.6, #左眼被遮挡的阈值 right_eye：0.6, #右眼被遮挡的阈值 nose：0.7, #鼻子被遮挡的阈值 mouth：0.7, #嘴巴被遮挡的阈值 left_check：0.8, #左脸颊被遮挡的阈值 right_check：0.8, #右脸颊被遮挡的阈值 chin_contour：0.6, #下巴被遮挡阈值
模糊度范围	**Blur**（0~1），0是最清晰，1是最模糊	小于0.7
光照范围	**illumination**（0~255） 脸部光照的灰度值，0表示光照不好 以及对应客户端SDK中，YUV的Y分量	大于40
姿态角度	**Pitch**：三维旋转之俯仰角度[-90(上), 90(下)] **Roll**：平面内旋转角[-180(逆时针), 180(顺时针)] **Yaw**：三维旋转之左右旋转角[-90(左), 90(右)]	分别小于20度
人脸完整度	**completeness**（0或1），0为人脸溢出图像边界，1为人脸都在图像边界内	视业务逻辑判断
人脸大小	人脸部分的大小 建议长宽像素值范围：80*80~200*200	人脸部分不小于**100*100**像素

```
  var fs = require('fs');

var uid = "user1";

var userInfo = "user's info";

var groupId = "group1,group2";

var image = fs.readFileSync("assets/example.jpg").toString("base64");

// 调用人脸注册
client.addUser(uid, userInfo, groupId, image).then(function(result) {
    console.log(JSON.stringify(result));
}).catch(function(err) {
    // 如果发生网络错误
    console.log(err);
});

// 如果有可选参数
var options = {};
options["action_type"] = "replace";

// 带参数调用人脸注册
client.addUser(uid, userInfo, groupId, image, options).then(function(result) {
    console.log(JSON.stringify(result));
}).catch(function(err) {
    // 如果发生网络错误
    console.log(err);
});;
```

人脸注册 请求参数详情

参数名称	是否必选	类型	默认值	说明
uid	是	string		用户id（由数字、字母、下划线组成），长度限制128B
user_info	是	string		用户资料，长度限制256B
group_id	是	string		用户组id，标识一组用户（由数字、字母、下划线组成），长度限制128B。如果需要将一个uid注册到多个group下，group_id需要用多个逗号分隔，每个group_id长度限制为48个英文字符。**注：group无需单独创建，注册用户时则会自动创建group。** **产品建议**：根据您的业务需求，可以将需要注册的用户，按照业务划分，分配到不同的group下，例如按照会员手机尾号作为groupid，用于刷脸支付、会员计费消费等，这样可以尽可能控制每个group下的用户数与人脸数，提升检索的准确率
image	是	string		图像base64编码，**每次仅支持单张图片，图片编码后大小不超过10M**。为保证后续识别的效果较佳，建议注册的人脸，为用户正面人脸。
action_type	否	string	append	参数包含append、replace。**如果为“replace”，则每次注册时进行替换replace（新增或更新）操作，默认为append操作**。例如：uid在库中已经存在时，对此uid重复注册时，新注册的图片默认会**追加**到该uid下，如果手动选择`action_type:replace`，则会用新图替换库中该uid下所有图片。

人脸注册 返回数据参数详情

字段	是否必选	类型	说明
log_id	是	uint64	请求唯一标识码，随机数

人脸注册 返回示例

```
 // 注册成功
{
    "log_id": 73473737,
}
// 注册发生错误
{
  "error_code": 216616,
  "log_id": 674786177,
  "error_msg": "image exist"
}
```

人脸更新

用于对人脸库中指定用户，更新其下的人脸图像。

> **说明：**针对一个uid执行更新操作，新上传的人脸图像将覆盖此uid原有所有图像。

> **说明：**执行更新操作，如果该uid不存在时，会返回错误。如果添加了action_type:replace,则不会报错，并自动注册该uid，操作结果等同注册新用户。

```
 var fs = require('fs');

var uid = "user1";

var userInfo = "user's info";

var groupId = "group1";

var image = fs.readFileSync("assets/example.jpg").toString("base64");

// 调用人脸更新
client.updateUser(uid, userInfo, groupId, image).then(function(result) {
    console.log(JSON.stringify(result));
}).catch(function(err) {
    // 如果发生网络错误
    console.log(err);
});

// 如果有可选参数
var options = {};
options["action_type"] = "replace";

// 带参数调用人脸更新
client.updateUser(uid, userInfo, groupId, image, options).then(function(result) {
    console.log(JSON.stringify(result));
}).catch(function(err) {
    // 如果发生网络错误
    console.log(err);
});;
```

人脸更新 请求参数详情

参数名称	是否必选	类型	默认值	说明
uid	是	string		用户id（由数字、字母、下划线组成），长度限制128B
user_info	是	string		用户资料，长度限制256B
group_id	是	string		更新指定groupid下uid对应的信息
image	是	string		图像数据，base64编码，要求base64编码后大小不超过4M，最短边至少15px，最长边最大4096px,支持jpg/png/bmp格式
action_type	否	string	append	目前仅支持replace，uid不存在时，不报错，会自动变为注册操作；未选择该参数时，如果uid不存在会提示错误

人脸更新 返回数据参数详情

字段	是否必选	类型	说明
log_id	是	uint64	请求唯一标识码，随机数

人脸更新 返回示例

```
 // 更新成功
{
   "log_id": 73473737,
}
// 更新发生错误
{
  "error_code": 216612,
  "log_id": 1137508902,
  "error_msg": "user not exist"
}
```

人脸删除

用于从人脸库中删除一个用户。

人脸删除注意事项：

- 删除的内容，包括用户所有图像和身份信息；
- 如果一个uid存在于多个用户组内，将会同时将从各个组中把用户删除
- 如果指定了group_id，则只删除此group下的uid相关信息

```
 var uid = "user1";

// 调用人脸删除
client.deleteUser(uid).then(function(result) {
    console.log(JSON.stringify(result));
}).catch(function(err) {
    // 如果发生网络错误
    console.log(err);
});

// 如果有可选参数
var options = {};
options["group_id"] = "group1";

// 带参数调用人脸删除
client.deleteUser(uid, options).then(function(result) {
    console.log(JSON.stringify(result));
}).catch(function(err) {
    // 如果发生网络错误
    console.log(err);
});;
```

人脸删除 请求参数详情

参数名称	是否必选	类型	说明
uid	是	string	用户id（由数字、字母、下划线组成），长度限制128B
group_id	否	string	删除指定groupid下uid对应的信息

人脸删除 返回数据参数详情

字段	是否必选	类型	说明
log_id	是	uint64	请求唯一标识码，随机数

人脸删除 返回示例

```
 // 删除成功
{
    "log_id": 73473737,
}
// 删除发生错误
{
  "error_code": 216612,
  "log_id": 1137508902,
  "error_msg": "user not exist"
}
```

用户信息查询

用于查询人脸库中某用户的详细信息。

```
 var uid = "user1";

// 调用用户信息查询
client.getUser(uid).then(function(result) {
    console.log(JSON.stringify(result));
}).catch(function(err) {
    // 如果发生网络错误
    console.log(err);
});

// 如果有可选参数
var options = {};
options["group_id"] = "group1";

// 带参数调用用户信息查询
client.getUser(uid, options).then(function(result) {
    console.log(JSON.stringify(result));
}).catch(function(err) {
    // 如果发生网络错误
    console.log(err);
});;
```

用户信息查询 请求参数详情

参数名称	是否必选	类型	说明
uid	是	string	用户id（由数字、字母、下划线组成），长度限制128B
group_id	否	string	选择指定group_id则只查找group列表下的uid内容，如果不指定则查找所有group下对应uid的信息

用户信息查询 返回数据参数详情

字段	是否必选	类型	说明
log_id	是	uint64	请求标识码，随机数，唯一
result	是	array(double)	结果数组
+uid	是	string	匹配到的用户id
+user_info	是	string	注册时的用户信息
+groups	是	array(string)	用户所属组列表

用户信息查询 返回示例

```
 {
“result_num” : 2
   "result": {
      [
         "uid": "testuser2",
         "user_info": "registed user info ...",
         "group_id": "grep1",
      ],
      [
         "uid": "testuser2",
         "user_info": "registed user info2 ...",
         "group_id": "grep2",
      ],
   },
   "log_id": 2979357502
}
```

组列表查询

用于查询用户组的列表。

```
 // 调用组列表查询
client.getGrouplist().then(function(result) {
    console.log(JSON.stringify(result));
}).catch(function(err) {
    // 如果发生网络错误
    console.log(err);
});

// 如果有可选参数
var options = {};
options["start"] = "0";
options["num"] = "50";

// 带参数调用组列表查询
client.getGrouplist(, options).then(function(result) {
    console.log(JSON.stringify(result));
}).catch(function(err) {
    // 如果发生网络错误
    console.log(err);
});;
```

组列表查询 请求参数详情

参数名称	是否必选	类型	默认值	说明
start	否	string	0	默认值0，起始序号
num	否	string	100	返回数量，默认值100，最大值1000

组列表查询 返回数据参数详情

字段	是否必选	类型	说明
result_num	是	number	返回个数
result	是	array(string)	group_id列表

组列表查询 返回示例

```
{
    "result_num": 2,
    "result": [
        "grp1",
        "grp2"
    ],
    "log_id": 3314921889
}
```

组内用户列表查询

用于查询指定用户组中的用户列表。

```
var groupId = "group1";

// 调用组内用户列表查询
client.getGroupUsers(groupId).then(function(result) {
    console.log(JSON.stringify(result));
}).catch(function(err) {
    // 如果发生网络错误
    console.log(err);
});

// 如果有可选参数
var options = {};
options["start"] = "0";
options["num"] = "50";

// 带参数调用组内用户列表查询
client.getGroupUsers(groupId, options).then(function(result) {
    console.log(JSON.stringify(result));
}).catch(function(err) {
    // 如果发生网络错误
    console.log(err);
});;
```

组内用户列表查询 请求参数详情

参数名称	是否必选	类型	默认值	说明
group_id	是	string		用户组id（由数字、字母、下划线组成），长度限制128B
start	否	string	0	默认值0，起始序号
num	否	string	100	返回数量，默认值100，最大值1000

组内用户列表查询 返回数据参数详情

字段	是否必选	类型	说明
log_id	是	uint64	请求标识码，随机数，唯一
result_num	是	uint32	返回个数
result	是	array(object)	user列表
+uid	是	string	用户id
+user_info	是	string	用户信息

组内用户列表查询 返回示例

```
{
  "log_id": 3314921889,
  "result_num": 2,
  "result": [
    {
      "uid": "uid1",
      "user_info": "user info 1"
    },
    {
      "uid": "uid2",
      "user_info": "user info 2"
    }
  ]
}
```

组间复制用户

用于将已经存在于人脸库中的用户**复制到一个新的组**。

> **说明：**并不是向一个指定组内添加用户，而是直接从其它组复制用户信息 如果需要注册用户，请直接使用**人脸注册接口**

```
 var srcGroupId = "group1";

var groupId = "group1,group2";

var uid = "user1";

// 调用组间复制用户
client.addGroupUsers(srcGroupId, groupId, uid).then(function(result) {
    console.log(JSON.stringify(result));
}).catch(function(err) {
    // 如果发生网络错误
    console.log(err);
});
```

组间复制用户 请求参数详情

参数名称	是否必选	类型	说明
src_group_id	是	string	从指定group里复制信息
group_id	是	string	用户组id，标识一组用户（由数字、字母、下划线组成），长度限制128B。如果需要将一个uid注册到多个group下，group_id需要用多个逗号分隔，每个group_id长度限制为48个英文字符。**注：group无需单独创建，注册用户时则会自动创建group。** **产品建议**：根据您的业务需求，可以将需要注册的用户，按照业务划分，分配到不同的group下，例如按照会员手机尾号作为groupid，用于刷脸支付、会员计费消费等，这样可以尽可能控制每个group下的用户数与人脸数，提升检索的准确率
uid	是	string	用户id（由数字、字母、下划线组成），长度限制128B

组间复制用户 返回数据参数详情

字段	是否必选	类型	说明
log_id	是	uint64	请求标识码，随机数，唯一

组间复制用户 返回示例

```
 // 正确返回值
{
    "log_id": 3314921889,
}
// 发生错误时返回值
{
  "error_code": 216100,
  "log_id": 3111284097,
  "error_msg": "already add"
}
```

组内删除用户

用于将用户从某个组中删除，但不会删除用户在其它组的信息。

> **说明：**当用户仅属于单个分组时，本接口将返回错误，请使用**人脸删除接口**

```
 var groupId = "group1,group2";

var uid = "user1";

// 调用组内删除用户
client.deleteGroupUsers(groupId, uid).then(function(result) {
    console.log(JSON.stringify(result));
}).catch(function(err) {
    // 如果发生网络错误
    console.log(err);
});
```

组内删除用户 请求参数详情

参数名称	是否必选	类型	说明
group_id	是	string	用户组id，标识一组用户（由数字、字母、下划线组成），长度限制128B。如果需要将一个uid注册到多个group下，group_id需要用多个逗号分隔，每个group_id长度限制为48个英文字符。**注：group无需单独创建，注册用户时则会自动创建group。** **产品建议**：根据您的业务需求，可以将需要注册的用户，按照业务划分，分配到不同的group下，例如按照会员手机尾号作为groupid，用于刷脸支付、会员计费消费等，这样可以尽可能控制每个group下的用户数与人脸数，提升检索的准确率
uid	是	string	用户id（由数字、字母、下划线组成），长度限制128B

组内删除用户 返回数据参数详情

字段	是否必选	类型	说明
log_id	是	uint64	请求标识码，随机数，唯一

组内删除用户 返回示例

```
 // 正确返回值
{
    "log_id": 3314921889,
}
// 发生错误时返回值
{
  "error_code": 216619,
  "log_id": 815967402,
  "error_msg": "user must be in one group at least"
}
```

身份验证

质量检测（可选）活体检测（可选）公安验证（必选）

```
 var fs = require('fs');

var image = fs.readFileSync("assets/example.jpg").toString("base64");
var idCardNumber = "110233112299822211";

var name = "张三";

// 调用身份验证
client.personVerify(image, idCardNumber, name).then(function(result) {
    console.log(JSON.stringify(result));
}).catch(function(err) {
    // 如果发生网络错误
    console.log(err);
});

// 如果有可选参数
var options = {};
options["quality"] = "use";
options["quality_conf"] = "{\"left_eye\": 0.6, \"right_eye\": 0.6}";
options["faceliveness"] = "use";
options["faceliveness_conf"] = "{\"faceliveness\": 0.834963}";
options["ext_fields"] = "qualities";

// 带参数调用身份验证
client.personVerify(image, idCardNumber, name, options).then(function(result) {
    console.log(JSON.stringify(result));
}).catch(function(err) {
    // 如果发生网络错误
    console.log(err);
});;
```

身份验证 请求参数详情

参数名称	是否必选	类型	说明
image	是	string	图像数据，base64编码，要求base64编码后大小不超过4M，最短边至少15px，最长边最大4096px,支持jpg/png/bmp格式
id_card_number	是	string	身份证号（真实身份证号号码）。我们的服务端会做格式校验，并通过错误码返回，但是为了您的产品反馈体验更及时，建议在产品前端做一下号码格式校验与反馈
name	是	string	utf8，姓名（真实姓名，和身份证号匹配）
quality	否	string	判断图片中的人脸质量是否符合条件。use表示需要做质量控制，质量不符合条件的照片会被直接拒绝
quality_conf	否	string	人脸质量检测中每一项指标的具体阈值设定，json串形式，当指定quality:use时生效
faceliveness	否	string	判断活体值是否达标。use表示需要做活体检测，低于活体阈值的照片会直接拒绝
faceliveness_conf	否	string	人脸活体检测的阈值设定，json串形式，当指定faceliveness:use时生效。默认使用的阈值如下：{faceliveness：0.834963}
ext_fields	否	string	可选项为faceliveness，qualities。选择具体的项，则返回参数中将会显示相应的扩展字段。如faceliveness表示返回结果中包含活体相关内容，qualities表示返回结果中包含质量检测相关内容

身份验证 返回数据参数详情

参数	必须	类型	说明
log_id	是	uint64	日志id
result	是	float	与公安小图相似度可能性，用于验证生活照与公安小图是否为同一人，有正常分数时为[0~1]，推荐阈值0.8，超过即判断为同一人
ext_info	否	string	拓展信息json串，只有选择了ext_fields时才会返回具体信息。选择faceliveness返回具体活体分值信息，选择qualities返回人脸质量检测信息。两者可以同时选择，半角逗号分割。
+faceliveness	否	string	活体检测值，单帧图片建议阈值，小于此值则认为不是活体，超过则判断为活体
+qualities	否	string	质量检测结果
++occlusion	否	string	人脸遮挡情况
+++left_eye	否	string	左眼被遮挡的比例，取值范围[0~1]，数值越大遮挡越多；小于阈值时有返回，默认阈值0.6
+++right_eye	否	string	右眼被遮挡的比例，取值范围[0~1]，数值越大遮挡越多；小于阈值时有返回，默认阈值0.6
+++nose	否	string	鼻子被遮挡的比例，取值范围[0~1]，数值越大遮挡越多；小于阈值时有返回，默认阈值0.7
+++mouth	否	string	嘴巴被遮挡的比例，取值范围[0~1]，数值越大遮挡越多；小于阈值时有返回，默认阈值0.7
+++left_cheek	否	string	左脸颊被遮挡的比例，取值范围[0~1]，数值越大遮挡越多；小于阈值时有返回，默认阈值0.8
+++right_cheek	否	string	右脸颊被遮挡的比例，取值范围[0~1]，数值越大遮挡越多；小于阈值时有返回，默认阈值0.8
+++chin	否	string	下巴被遮挡的比例，取值范围[0~1]，数值越大遮挡越多；小于阈值时有返回，默认阈值0.8
++blur	否	string	人脸模糊度阈值，取值范围[0~1]，数值越大越模糊；小于阈值时有返回，默认阈值0.7
++illumination	否	string	脸部光照的灰度值阈值，取值范围[0~255]，数值越大光照越强；大于阈值时有返回，默认阈值30
++completeness	否	string	人脸完整度，0或1, 0为人脸溢出图像边界，1为人脸都在图像边界内

身份验证 返回示例

```
{
 "result":0.03419,
 "ext_info":{
   "faceliveness":0.834963
 },
 "log_id":772889134072410
}
```

在线活体检测

人脸基础信息，人脸质量检测，基于图片的活体检测

```
  var fs = require('fs');

var image = fs.readFileSync("assets/example.jpg").toString("base64");

// 调用在线活体检测
client.faceverify(image).then(function(result) {
    console.log(JSON.stringify(result));
}).catch(function(err) {
    // 如果发生网络错误
    console.log(err);
});

// 如果有可选参数
var options = {};
options["max_face_num"] = "2";
options["face_fields"] = "qualities";

// 带参数调用在线活体检测
client.faceverify(image, options).then(function(result) {
    console.log(JSON.stringify(result));
}).catch(function(err) {
    // 如果发生网络错误
    console.log(err);
});;
```

在线活体检测 请求参数详情

参数名称	是否必选	类型	默认值	说明
image	是	string		图像数据，base64编码，要求base64编码后大小不超过4M，最短边至少15px，最长边最大4096px,支持jpg/png/bmp格式
max_face_num	否	string	1	最多处理人脸数目，默认值1
face_fields	否	string		如不选择此项，返回结果默认只有人脸框、概率和旋转角度。可选参数为qualities、faceliveness。qualities：图片质量相关判断；faceliveness：活体判断。如果两个参数都需要选择，请使用半角逗号分隔。

在线活体检测 返回数据参数详情

参数	类型	是否必须	说明
log_id	uint64	是	日志id
result_num	uint32	是	人脸数目
result	object[]	是	人脸属性对象的集合
+faceliveness	float	否	活体分数，face_fields包括qualities时返回
+location	bject	是	人脸在图片中的位置
++left	uint32	是	人脸区域离左边界的距离
++top	uint32	是	人脸区域离上边界的距离
++width	uint32	是	人脸区域的宽度
++height	uint32	是	人脸区域的高度
+face_probability	double	是	人脸置信度，范围0-1
+rotation_angle	int32	是	人脸框相对于竖直方向的顺时针旋转角，[-180,180]
+yaw	double	是	三维旋转之左右旋转角[-90(左), 90(右)]
+pitch	double	是	三维旋转之俯仰角度[-90(上), 90(下)]
+roll	double	是	平面内旋转角[-180(逆时针), 180(顺时针)]
+qualities	object	否	人脸质量信息。face_fields包含qualities时返回
++occlusion	object	是	人脸各部分遮挡的概率，区间[0, 1]
+++left_eye	double	是	左眼
+++right_eye	double	是	右眼
+++nose	double	是	鼻子
+++mouth	double	是	嘴
+++left_cheek	double	是	左脸颊
+++right_cheek	double	是	右脸颊
+++chin	double	是	下巴
++blur	double	是	人脸模糊程度，[0, 1]。0表示清晰，1表示模糊
++illumination	double	是	取值范围在[0,255],表示脸部区域的光照程度
++completeness	double	是	人脸完整度，[0, 1]。0表示完整，1表示不完整
++type	object	是	真实人脸/卡通人脸置信度
+++human	double	是	真实人脸置信度，[0, 1]
+++cartoon	double	是	卡通人脸置信度，[0, 1]

在线活体检测 返回示例

```
{
  log_id: 1900901488032821,
  result_num: 1,
  result: [
    {
      rotation_angle: 10,
      yaw: 11.357421875,
      faceliveness: 8.1253347161692e-05,
      location: {
        width: 96,
        top: 73,
        height: 96,
        left: 283
      },
      qualities: {
        illumination: 211,
        occlusion: {
          right_eye: 0,
          left_eye: 0.039751552045345,
          left_cheek: 0.12623985111713,
          mouth: 0,
          nose: 0,
          chin: 0.037661049515009,
          right_cheek: 0.024720622226596
        },
        completeness: 1,
        type: {
          cartoon: 0,
          human: 0
        },
        blur: 2.5251445032182e-11
      },
      pitch: 1.0063140392303,
      roll: 12.760620117188,
      face_probability: 1
    }
  ]
}
```


错误信息

错误返回格式

若请求错误，服务器将返回的JSON文本包含以下参数：

- **error_code**：错误码。
- **error_msg**：错误描述信息，帮助理解和解决发生的错误。

错误码

错误码	错误信息	描述
4	Open api request limit reached	集群超限额
14	IAM Certification failed	IAM鉴权失败，建议用户参照文档自查生成sign的方式是否正确，或换用控制台中ak sk的方式调用
17	Open api daily request limit reached	每天流量超限额
18	Open api qps request limit reached	QPS超限额
19	Open api total request limit reached	请求总量超限额
100	Invalid parameter	无效参数
110	Access token invalid or no longer valid	Access Token失效
111	Access token expired	Access token过期
216015	module closed	模块关闭
216100	invalid param	参数异常
216101	not enough param	缺少必须的参数
216102	service not support	请求了不支持的服务，请检查调用的url
216103	param too long	请求超长，一般为一次传入图片个数超过系统限制
216110	appid not exist	appid不存在，请重新检查后台应用列表中的应用信息
216111	invalid userid	userid信息非法，请检查对应的参数
216200	empty image	图片为空或者base64解码错误
216201	image format error	图片格式错误
216202	image size error	图片大小错误
216300	db error	数据库异常，**少量发生时重试即可**
216400	backend error	后端识别服务异常，可以根据具体msg查看错误原因
216401	internal error	内部错误
216402	face not found	未找到人脸，请检查图片是否含有人脸
216500	unknown error	未知错误
216611	user not exist	用户不存在，请确认该用户是否注册或注册已经生效(**需要已经注册超过5s**)
216613	fail to delete user record	删除用户图片记录失败，重试即可
216614	not enough images	两两比对中图片数少于2张，无法比较
216615	fail to process images	服务处理该图片失败，发生后重试即可
216616	image existed	图片已存在
216617	fail to add user	新增用户图片失败
216618	no user in group	组内用户为空，确认该group是否存在或已经生效(需要已经注册超过5s)
216631	request add user overlimit	本次请求添加的用户数量超限

离线采集SDK

Android SDK4.0

Android端人脸采集SDK4.0使用文档（4.6.0.5）

目录

1 简介
1.1 功能介绍
1.2 兼容性
1.3 开发包说明
2 集成指南
2.1 Sample示例
2.2 准备工作
2.2.1 申请license
2.2.2 下载SDK
2.3 运行示例工程
2.3.1 运行自动配置授权信息的示例工程
2.3.2 运行未配置授权信息的示例工程
2.4添加SDK到工程
2.5权限声明
3 功能使用
3.1 活体识别
3.2人脸采集
3.3质量校验设置
3.4 界面定制说明
3.4.1 修改faceplatform_ui界面
4 接口设计说明
4.1人脸功能管理器
4.1.1 创建实例
4.1.2人脸功能管理器初始化
4.1.3设置人脸功能控制参数
4.1.4取得人脸图像采集功能接口
4.1.5取得活体检测功能接口
4.2人脸图像采集器
4.2.1设置人脸图像采集功能参数
4.2.2人脸图像采集
4.3活体检测器
4.3.1设置人脸功能控制参数
4.3.2活体检测
4.4人脸图像采集界面
4.5 活体检测界面
5 常见问题

1、简介

百度Face SDK Android 版是一种面向 Android 移动设备人脸技术开发包，此版SDK包含人脸检测、动作活体识别等功能，以aar包+动态链接库的形式发布。 基于该方案，开发者可以轻松的构建包含人脸检测、采集和活体识别的应用。在您使用SDK之前，我们首先为您介绍以下SDK本身及相关人脸能力，以便您能更方便使用人脸服务。

1.1 功能介绍

此版SDK所包含的能力如下：

- **本地版活体检测**：通过让用户做出指定人脸配合式的交互动作，识别当前操作者是否为活体。此功能为离线使用，可设定指定动作是否使用及应用顺序。动作包含：眨眼、张嘴、左摇头，右摇头，摇摇头、向上抬头，向下低头。可有效抵御高清图片、3D建模、视频等攻击。**说明：用户完成指定动作即判断为活体，不返回相应分数。**

> 建议搭配线上的在线活体检测接口配合使用，在本地做动作活体检测，云端进行视频翻拍、模型攻击以及合成图攻击的二次校验

- **本地版人脸质量检测**：判断视频流中的图片帧中，哪些图片质量较佳，即人脸图像特征清晰（满足角度、姿态、光照、模糊度等校验）。

- **本地版人脸图像采集**：通过本地SDK能力，采集人脸图像，同时经过人脸质量检测，确保采集到的人脸图像符合各条件校验（角度、姿态、光照、模糊度等），为设备前端获取有效可分析人脸的主要功能。
- **离线授权**：SDK的授权判断，授权介质也称为license，在SDK使用中，需要通过license向授权服务器发起请求，判断SDK的使用合法性及使用有效期。

> 此版SDK**全部功能为离线版本，所有功能均本地化使用**，主要用于在客户端（Android）获取人脸，实际业务使用中，可以按照业务需要，配合在线API完成全流程的业务集成。

1.2 兼容性

系统：支持 Android 4.0.3(API Level 15)及以上系统。需要开发者通过 minSdkVersion来保证支持系统的检测。

机型：手机，平板暂不支持

构架：支持 CPU架构平台【arm-v7，arm-64，x86】

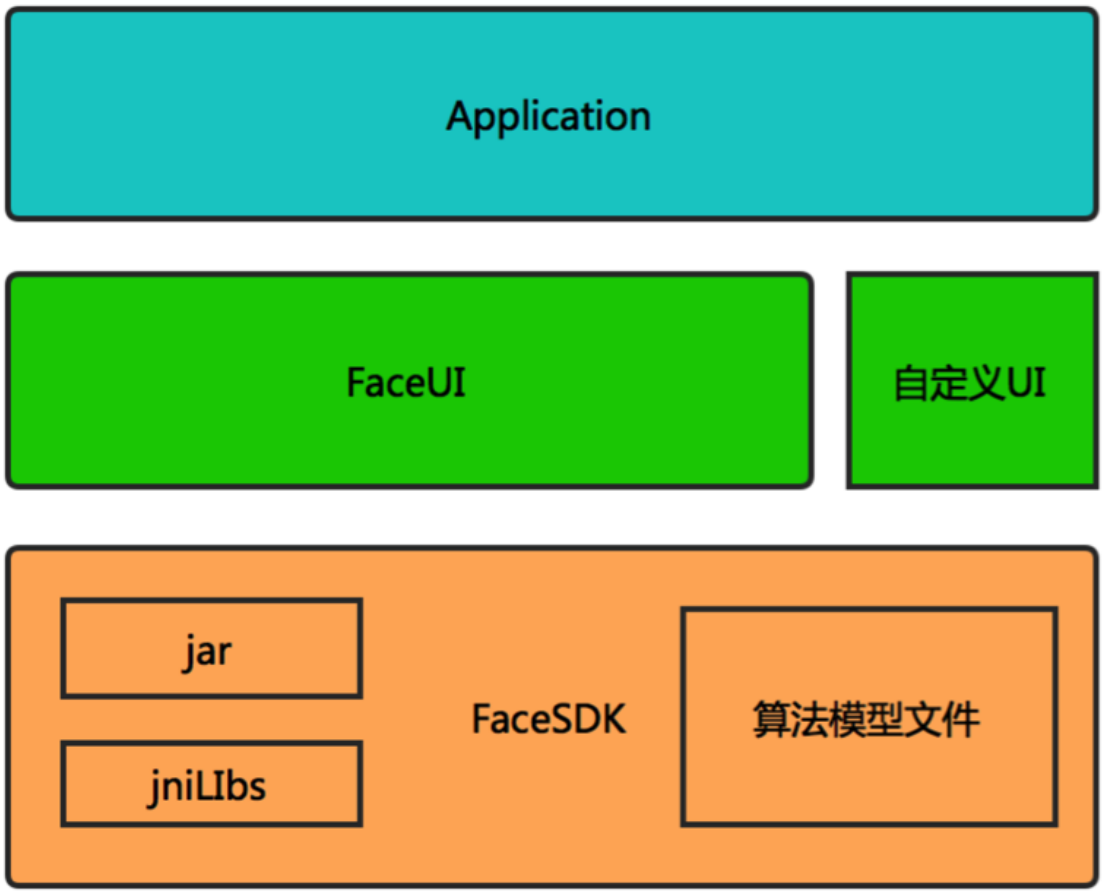
网络：支持 WIFI 及移动网络,移动网络支持使用NET 网关及 WAP 网关(CMWAP、CTWAP、UNIWAP、3GWAP)。

1.3 开发包说明

文件/文件夹名	说明
/faceplatform-release	SDK lib 库相关代码的 aar
/faceplatform-ui	SDK的UI库，封装拍照裁剪等功能,以及各平台的so库。so包含以下几个平台如果关注包大小，请自行删减。/armeabi-v7a/arm64-v8a/x86，
/app	DEMO工程

2、集成指南

本章将进行 Step-By-Step的讲解,如何快速的集成 人脸Sdk到现有应用中。一个完整的Demo 请参考开发包中的示例程序 FacePlatform。方案架构参考下图：

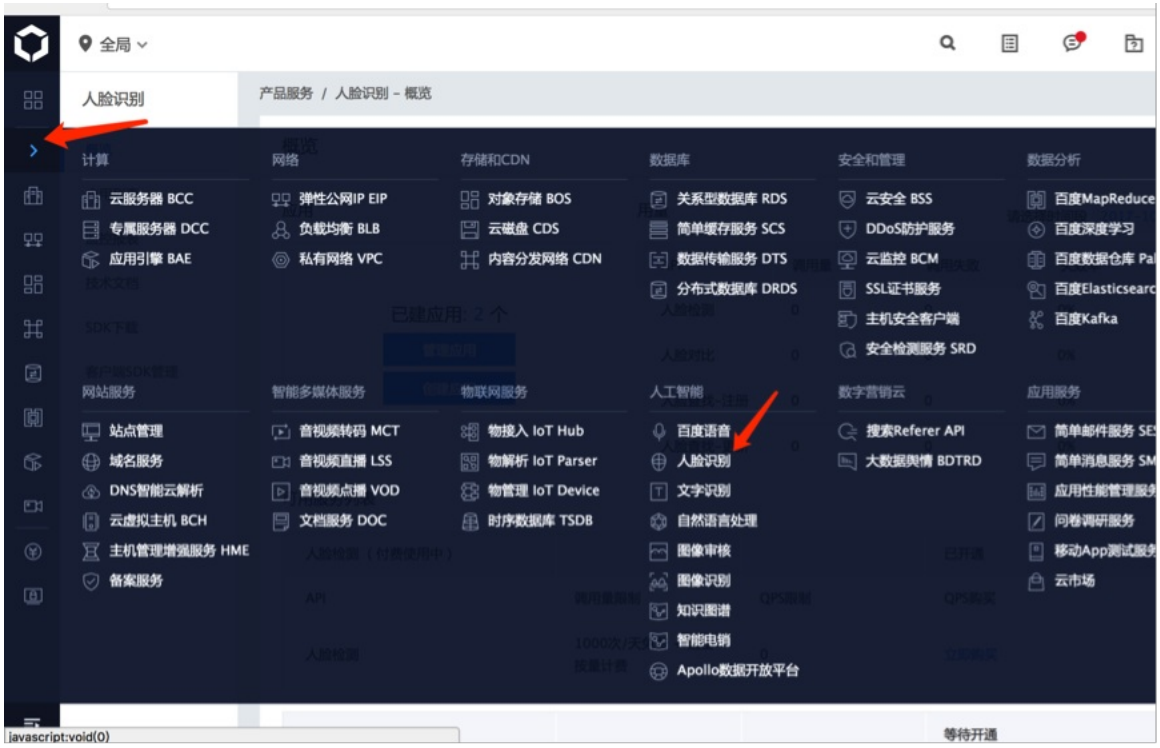


2.1 准备工作

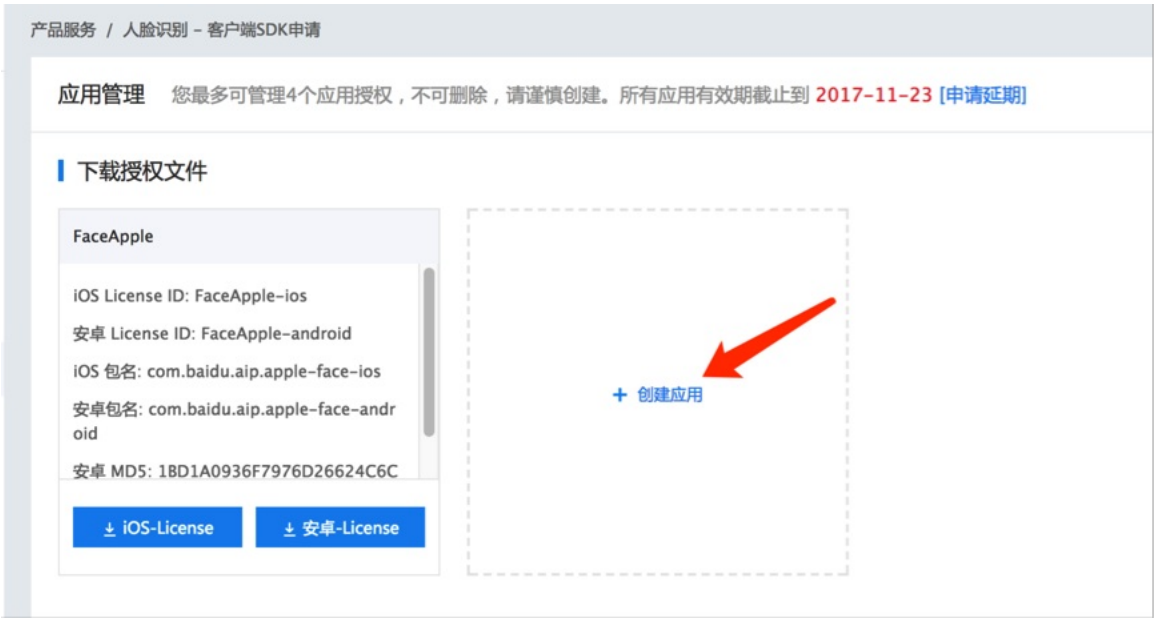
2.1.1 申请license

人脸SDK License：此license用于SDK离线功能使用，在您的申请人脸SDK的后台页面，全局->产品服 务->人脸识别->本地化部署->离线采集SDK申请

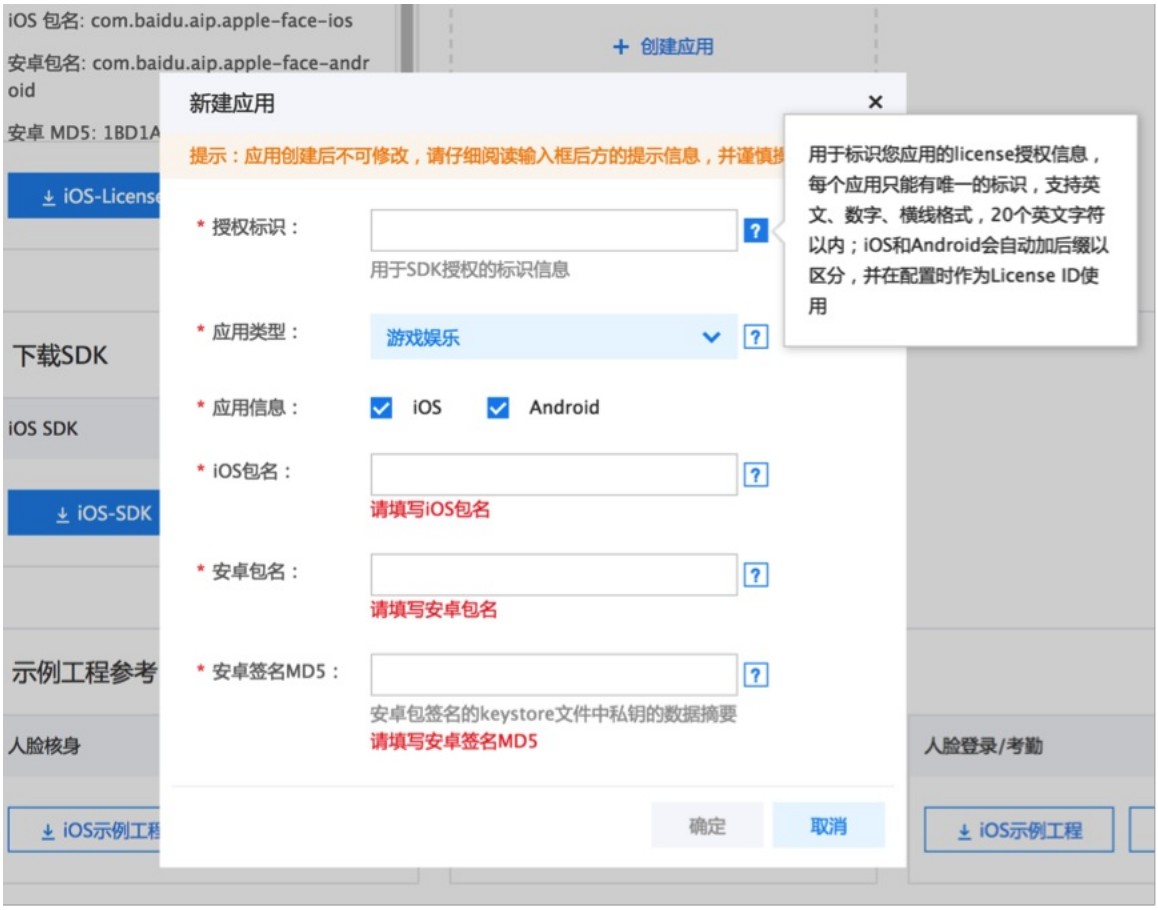
人脸控制台路径如下：



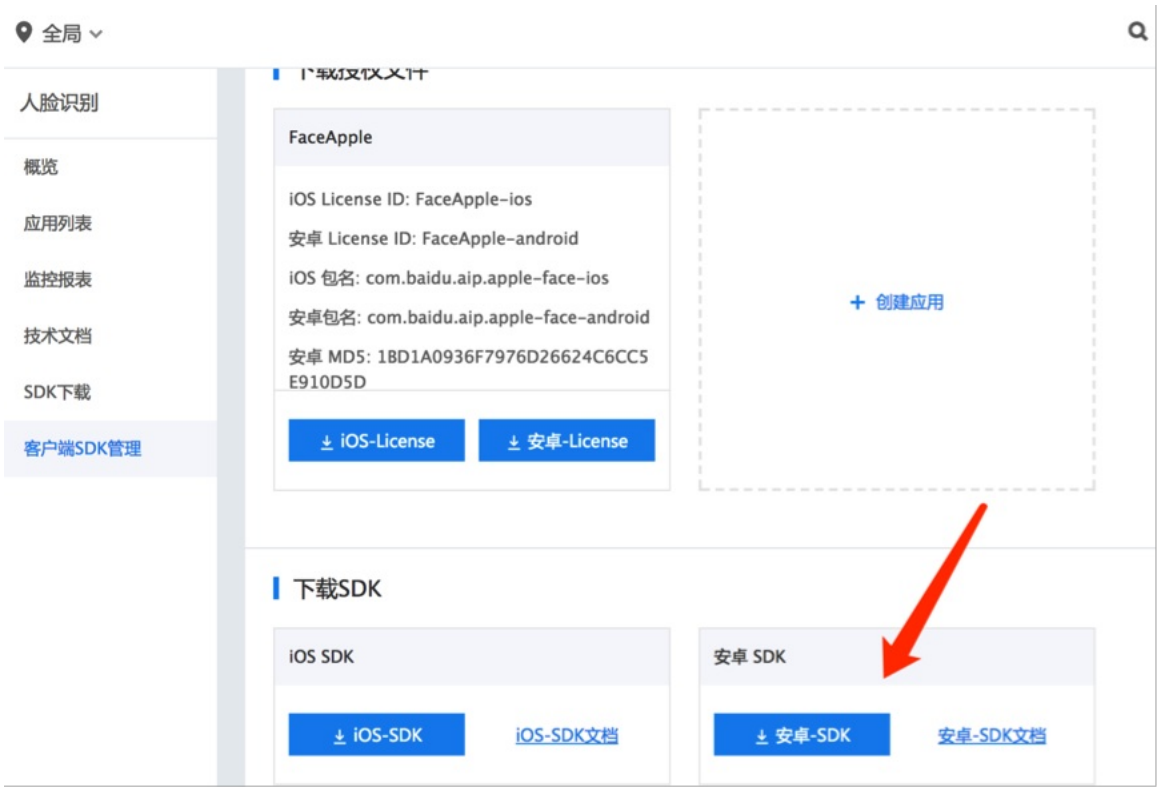
点击客户端SDK管理，弹出如下图：创建应用（这里创建应用是为了使用离线SDK，上面创建应用为了使用人脸在线接口，如注册、识别等）



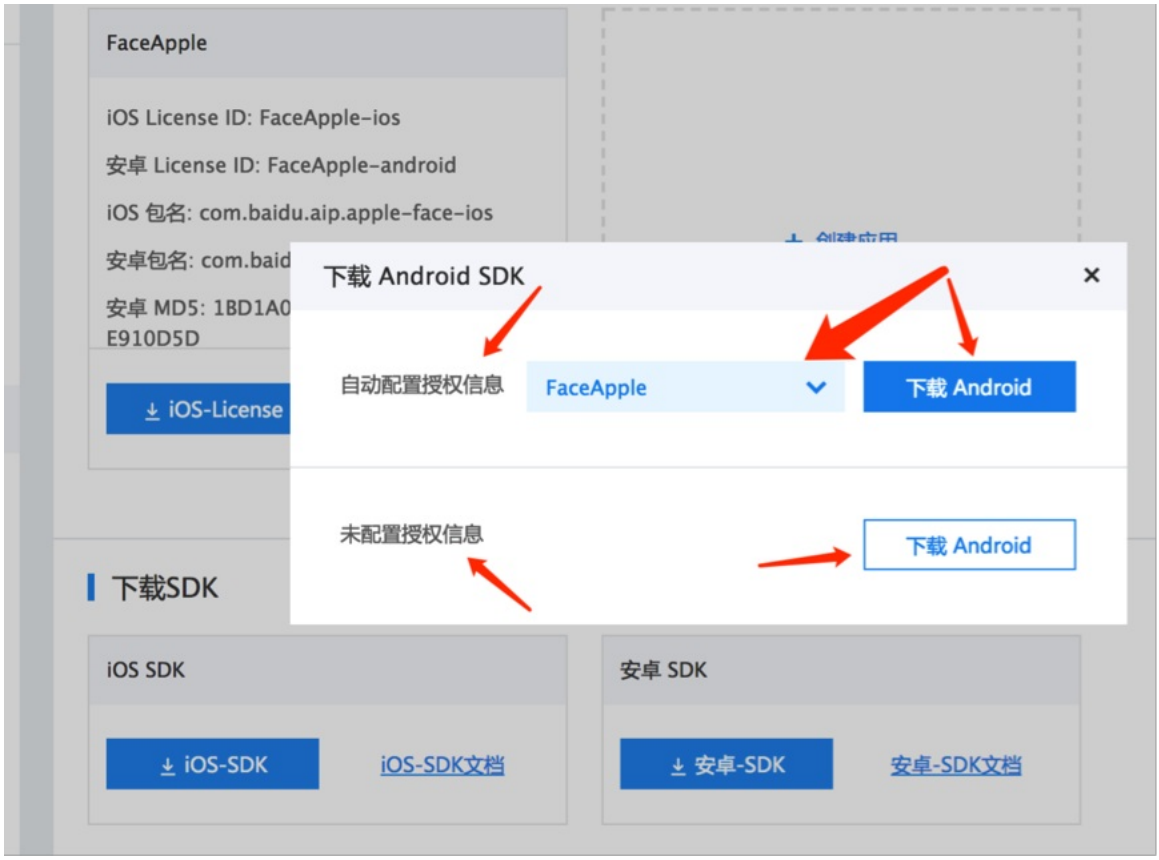
在弹出的框中输入授权标识，选择应用类型，应用系统，以及包名、MD5签名，详情请查看输入框右边提示



2.1.2 下载SDK



下载SDK分为自动配置授权信息（创建license后就可以选择为该应用，下载后SDK自动帮您配置授权，不用下载license拷贝到工程中，初始化参数licenseID,包名也帮您配置好了）和未配置授权信息两种方式：



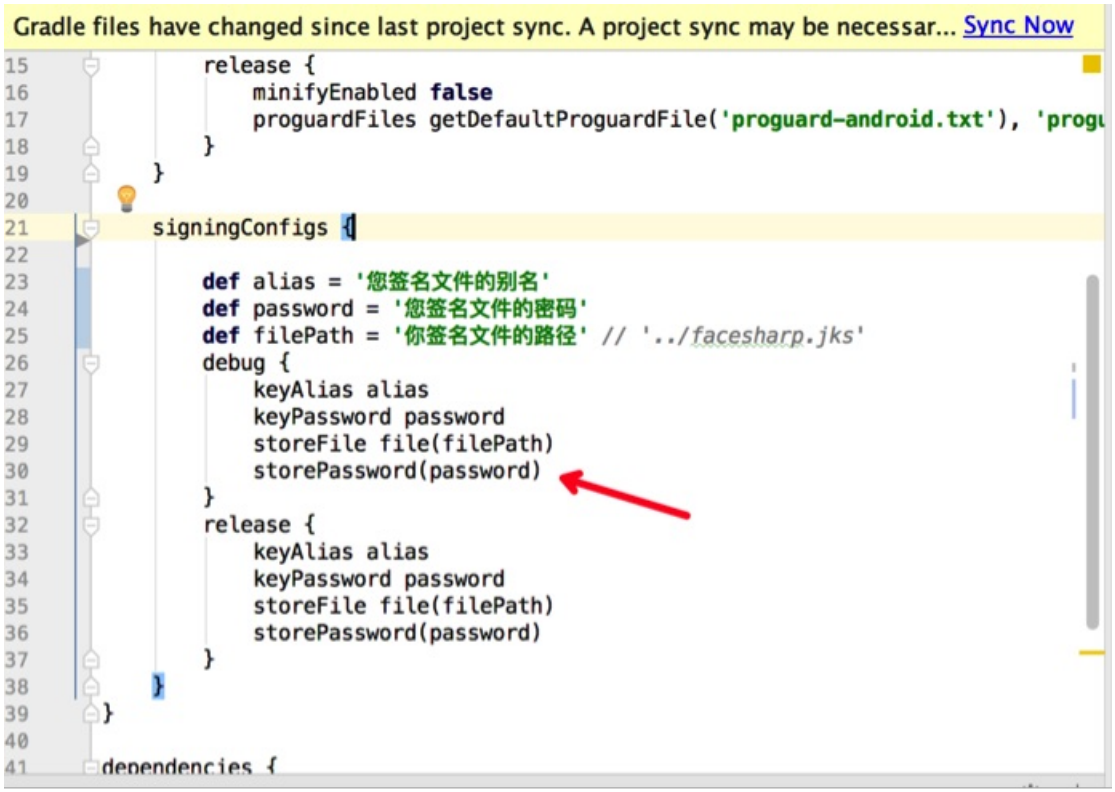
2.2 运行示例工程

2.3.1 运行自动配置授权信息的示例工程

该下载的示例工程，已经帮你改好了license和包名

- Android Studio导入下载的示例工程

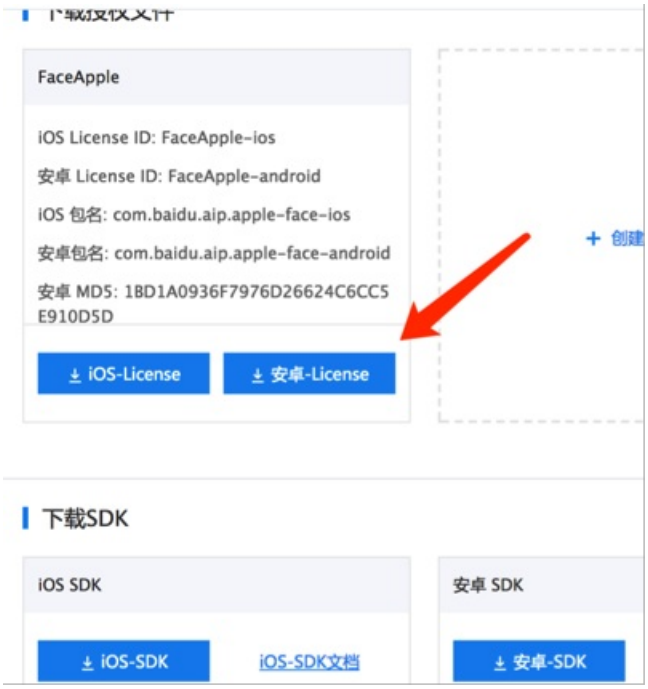
- 配置打包签名文件，由于SDK运行时会校验签名文件的MD5是否和申请时填入一样。为了便于debug能正常使用SDK，需要配置打包签名文件。
- 运行示例工程

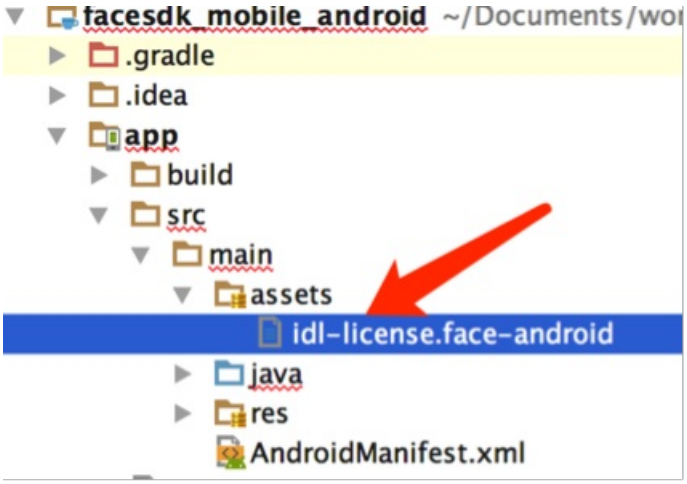


2.3.2 运行未配置授权信息的示例工程

（1）Android Studio导入下载的示例工程

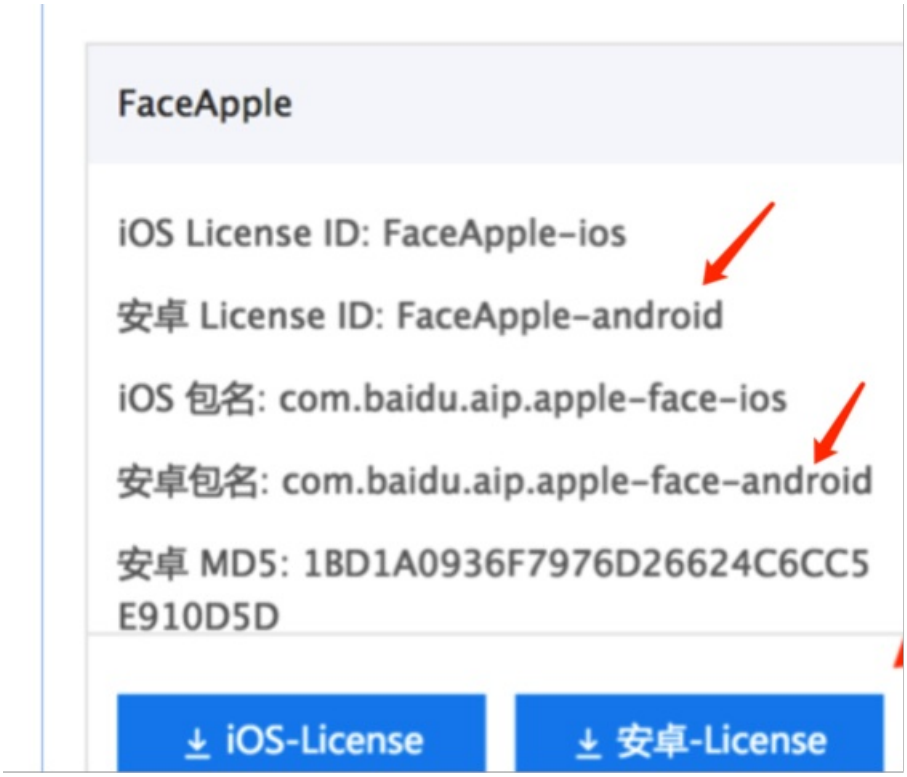
（2）下载license拷贝到工程的assets目录


FaceApple
iOS License ID: FaceApple-ios
安卓 License ID: FaceApple-android
iOS 包名: com.baidu.aip.apple-face-ios
安卓包名: com.baidu.aip.apple-face-android
安卓 MD5: 1BD1A0936F7976D26624C6CC5E910D5D
iOS-License
安卓-License
下载SDK
iOS SDK
安卓 SDK
iOS-SDK
iOS-SDK文档
安卓-SDK

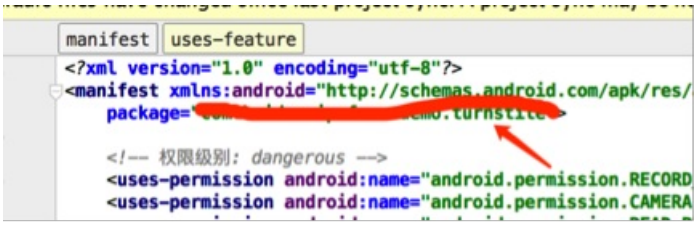

facesdk_mobile_android ~/Documents/wo
.gradle
.idea
app
build
src
main
assets
idl-license.face-android
java
res
AndroidManifest.xml

（3）修改Config类

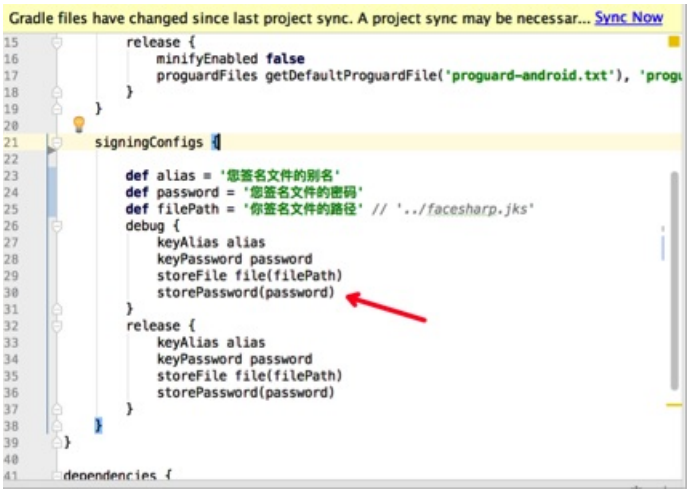

public class Config {
查看申请license信息，里面包含licenseID
public static String licenseID = 替换为你的licenseID,后台SDK管理界面中,已经生成的licenseID,如:test-face-android;
public static String licenseFileName = "idl-license.face-android";
}
放到assets目录下的license文件的名字



（4）修改app.gradle和AndroidManifest里面的包名为申请license填入的包名，如上图安卓包名。



（5）配置打包签名文件，由于SDK运行时会校验签名文件的MD5是否和申请时填入一样。为了便于debug能正常使用SDK，需要配置打包签名文件。



（6）运行示例工程。如果无法正常体验，请查看logcat日志。是否有 **FaceSDK-License LICENSE_INFO_CHECK_ERROR**日志。如果有说明授权没有成功，可以查看本文档最后的常见问题进行解决。

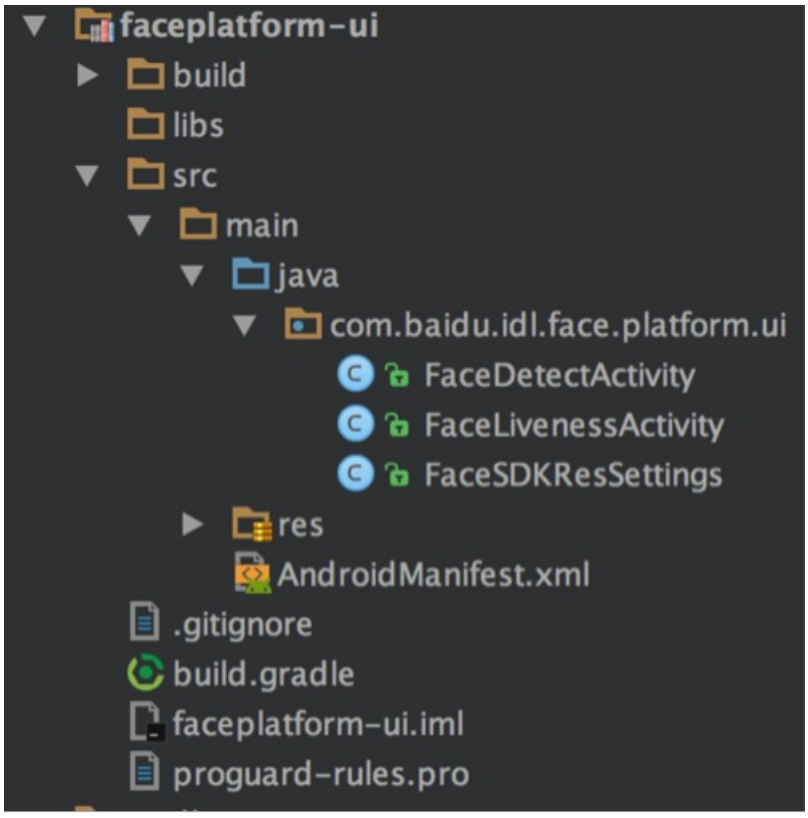
2.4 添加SDK到工程

FaceSdk以androidstudio开发方式提供，以下介绍在android studio开发工具导入FaceSdk

（1）将开发包中的faceplatform-release库Copy 到工程根目录。

（2）将开发包中的faceplatform-ui库Copy 到工程根目录。

（3）SDK提供的了开源的faceplatform-ui库，把活体检测和人脸图像采集功能等功能进行了封装，适配了主流机型机型。如果需要使用，请添加faceplatform-ui模块到的工程中。faceplatform-ui目录结构如下图



（4）在build.gradle使用compile project引人faceplatform-ui库工程。

```
dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    compile "com.android.support:recyclerview-v7:+"
    compile project(path: ':faceplatform-ui')
}
```

（5）Setting.gradle中include faceplatfrom-ui和facepaltfrom-release

```
include ':app', ':faceplatform-release'
include ':faceplatform-ui'
```

（6）从官网下载授权文件license，复制到app/src/main/assets目录下。

（7）申请的license已经和打包签名key进行了绑定（申请时用到了签名的md5，为了便于debug模式也能调用SDK的功能，需要把debug的key改成申请license的key。

- 把key拷贝到项目根目录下
- 主appbuild.gradle android 下面添加(修改)signingConfigs相关的配置。如下图。

```
    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'
        }
    }

    signingConfigs {
        debug {
            keyAlias 'facesharp'
            keyPassword [redacted]
            storeFile file('../facesharp.jks')
            storePassword([redacted])
        }
        release {
            keyAlias [redacted]
            keyPassword [redacted]
            storeFile file('../facesharp.jks')
            storePassword([redacted])
        }
    }
```

2.5 权限声明

名称	用途
android.permission.INTERNET	允许应用联网,SDK联网授权。
android.permission.READ_PHONE_STATE	获取用户手机的 IMEI,用来唯一的标识用户
android.permission.CAMERA	允许调用相机进行拍照
android.hardware.camera.autofocus	允许相机对焦
android.permission.WRITE_EXTERNAL_STORAGE	图片裁剪临时存储
android.permission.INTERNET	允许访问网络
android.permission.WRITE_SETTINGS	允许修改系统设置

3、功能使用

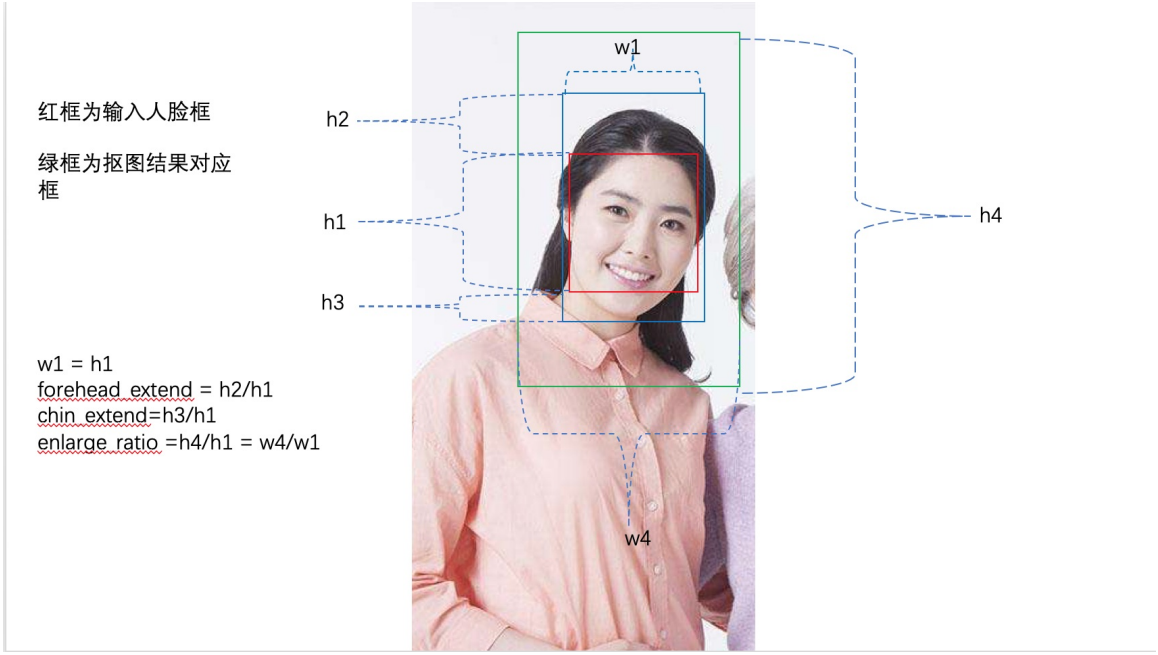
3.1 人脸采集（包含动作活体）

（1）初始化SDK 调用FaceSDKManager.getIntance().initialize(Context context, String licenseID, String licenseFileName, IInitCallback callback); **Demo中此段代码在HomeActivity中。**

（2）初始化参数设置

```
        FaceConfig config = FaceSDKManager.getInstance()
        .getFaceConfig();
       // 设置可检测的最小人脸阈值
       config.setMinFaceSize(FaceEnvironment.VALUE_MIN_FACE_SIZE);
       // 设置可检测到人脸的阈值
       config.setNotFaceValue(FaceEnvironment.VALUE_NOT_FACE_THRESHOLD);
       // 设置模糊度阈值
       config.setBlurnessValue(FaceEnvironment.VALUE_BLURNESS);
       // 设置光照阈值（范围0-255）
       config.setBrightnessValue(FaceEnvironment.VALUE_BRIGHTNESS);
       // 设置遮挡阈值
       config.setOcclusionValue(FaceEnvironment.VALUE_OCCLUSION);
       // 设置人脸姿态角阈值
       config.setHeadPitchValue(FaceEnvironment.VALUE_HEAD_PITCH);
       config.setHeadYawValue(FaceEnvironment.VALUE_HEAD_YAW);
       // 设置闭眼阈值
       config.setEyeClosedValue(FaceEnvironment.VALUE_CLOSE_EYES);
       // 设置图片缓存数量
       config.setCacheImageNum(FaceEnvironment.VALUE_CACHE_IMAGE_NUM);
       // 设置活体动作，通过设置list，LivenessTypeEunm.Eye, LivenessTypeEunm.Mouth,
       // LivenessTypeEunm.HeadUp, LivenessTypeEunm.HeadDown, LivenessTypeEunm.HeadLeft,
       // LivenessTypeEunm.HeadRight, LivenessTypeEunm.HeadLeftOrRight
       config.setLivenessTypeList(ExampleApplication.livenessList);
       // 设置动作活体是否随机
       config.setLivenessRandom(ExampleApplication.isLivenessRandom);
       // 设置开启提示音
       config.setSound(ExampleApplication.isOpenSound);
   // 原图缩放系数
       config.setScale(FaceEnvironment.VALUE_SCALE);
       // 抠图高的设定，为了保证好的抠图效果，我们要求高宽比是4：3，所以会在内部进行计算，只需要传入高即可
       config.setCropHeight(FaceEnvironment.VALUE_CROP_HEIGHT);
       // 加密类型，0：Base64加密，上传时image_sec传false；1：百度加密文件加密，上传时image_sec传true
       config.setSecType(FaceEnvironment.VALUE_SEC_TYPE);
       FaceSDKManager.getInstance().setFaceConfig(config);
```

关于抠图内部实现方案如下图所示：



（3） startActivity(new Intent(this, FaceLivenessExpActivity.class))，开启预览。

（4）调用FaceSDKManager.getInstance().getLivenessStrategyModule()获得ILivenessStrategy对象。(该方法每次调用都会返回一个新对象).

（6）调用ILivenessStrategy.setPreviewDegree();设置预览图片的旋转角度。调用setDetectStrategySoundEnable设置是否开启语音。调用setDetectStrategyConfig设置，预览图的大小，人脸检测框的坐标和回调。

（7）多次调用livenessStrategy进行人脸图片采集，人脸跟踪、质量检测、动作活体检测。

（8）实现ILivenessStrategyCallback的onLivenessCompletion并处理结果。**其中base64ImageCropMap为存放最佳人脸抠图图片，base64ImageSrcMap为存放最佳人脸原图图片**可以查看起父类FaceLivenessActivity的saveImage和base64ToImage方法，获取对于的bitamap。

```
public class FaceLivenessExpActivity extends FaceLivenessActivity {

    private DefaultDialog mDefaultDialog;

    @Override
    public void onCreate(Bundle savedInstanceState) { super.onCreate(savedInstanceState); }

    @Override
    public void onLivenessCompletion(FaceStatusEnum status, String message, HashMap<String, String> base64ImageMap) {
        super.onLivenessCompletion(status, message, base64ImageMap);
        if (status == FaceStatusEnum.OK && mIsCompletion) {
            showMessageDialog("活体检测", "检测成功");
        } else if (status == FaceStatusEnum.Error_DetectTimeout ||
                status == FaceStatusEnum.Error_LivenessTimeout ||
                status == FaceStatusEnum.Error_Timeout) {
            showMessageDialog("活体检测", "采集超时");
        }
    }
```

（9）Demo中针对最优抠图或者最优原图，都会调用SecRequest类下的sendMessage(Context, String secBase64);方法将加密后的base64发送到服务端。

3.2 人脸采集（不包含动作活体）

（1）调用FaceSDKManager.getIntance().initialize(context,Config.licenseID, Config.licenseFileName, IInitCallback callback);初始化SDK.。

（2）初始化SDK参数

```
        FaceConfig config = FaceSDKManager.getInstance()
    .getFaceConfig();
        // 设置可检测的最小人脸阈值
        config.setMinFaceSize(FaceEnvironment.VALUE_MIN_FACE_SIZE);
        // 设置可检测到人脸的阈值
        config.setNotFaceValue(FaceEnvironment.VALUE_NOT_FACE_THRESHOLD);
        // 设置模糊度阈值
        config.setBlurnessValue(FaceEnvironment.VALUE_BLURNESS);
        // 设置光照阈值（范围0-255）
        config.setBrightnessValue(FaceEnvironment.VALUE_BRIGHTNESS);
        // 设置遮挡阈值
        config.setOcclusionValue(FaceEnvironment.VALUE_OCCLUSION);
        // 设置人脸姿态角阈值
        config.setHeadPitchValue(FaceEnvironment.VALUE_HEAD_PITCH);
        config.setHeadYawValue(FaceEnvironment.VALUE_HEAD_YAW);
        // 设置闭眼阈值
        config.setEyeClosedValue(FaceEnvironment.VALUE_CLOSE_EYES);
        // 设置图片缓存数量
        config.setCacheImageNum(FaceEnvironment.VALUE_CACHE_IMAGE_NUM);
        // 设置活体动作，通过设置list，LivenessTypeEunm.Eye, LivenessTypeEunm.Mouth,
        // LivenessTypeEunm.HeadUp, LivenessTypeEunm.HeadDown, LivenessTypeEunm.HeadLeft,
        // LivenessTypeEunm.HeadRight, LivenessTypeEunm.HeadLeftOrRight
        config.setLivenessTypeList(ExampleApplication.livenessList);
        // 设置动作活体是否随机
        config.setLivenessRandom(ExampleApplication.isLivenessRandom);
        // 设置开启提示音
        config.setSound(ExampleApplication.isOpenSound);
        // 原图缩放系数
        config.setScale(FaceEnvironment.VALUE_SCALE);
        // 抠图高的设定，为了保证好的抠图效果，我们要求高宽比是4：3，所以会在内部进行计算，只需要传入高即可
        config.setCropHeight(FaceEnvironment.VALUE_CROP_HEIGHT);
        // 加密类型，0：Base64加密，上传时image_sec传false；1：百度加密文件加密，上传时image_sec传true
        config.setSecType(FaceEnvironment.VALUE_SEC_TYPE);
        FaceSDKManager.getInstance().setFaceConfig(config);
```

（3）startActivity(new Intent(this, FaceDetectExpActivity.class))，开启预览。

（4）调用FaceSDKManager.getInstance().getDetectStrategyModule()获得IDetectStrategy对象。(该方法每次调用都会返回一个新对象).

（5）调用IDetectStrategy.setPreviewDegree();设置预览图片的旋转角度。调用setDetectStrategySoundEnable设置是否开启语音。调用setDetectStrategyConfig设置，预览图的大小，人脸检测框的坐标和回调。

（6） 多次调用detectStrategy进行人脸图片采集。（不带动作活体中支持口罩检测，当口罩分值大于0.7时便认为戴口罩，这时将不会进行遮挡检测）

（7） 实现IDetectStrategyCallback的onDetectCompletion并处理结果。其中**base64ImageCropMap为存放最佳抠图人脸，base64ImageSrcMap为存放最佳原图人脸**。可以查看起父类FaceDetectActivity的saveImage和base64ToImage方法，获取对于的bitmap。

```
public class FaceDetectExpActivity extends FaceDetectActivity {

    private DefaultDialog mDefaultDialog;

    @Override
    public void onCreate(Bundle savedInstanceState) { super.onCreate(savedInstanceState); }

    @Override
    public void onDetectCompletion(FaceStatusEnum status, String message, HashMap<String, String> base64ImageMap) {
        super.onDetectCompletion(status, message, base64ImageMap);
        if (status == FaceStatusEnum.OK && mIsCompletion) {
            showMessageDialog("人脸图像采集", "采集成功");
        } else if (status == FaceStatusEnum.Error_DetectTimeout ||
                status == FaceStatusEnum.Error_LivenessTimeout ||
                status == FaceStatusEnum.Error_Timeout) {
            showMessageDialog("人脸图像采集", "采集超时");
        }
    }

    private void showMessageDialog(String title, String message) {
```

（8）Demo中针对最优抠图或者最优原图，都会调用SecRequest类下的sendMessage(Context, String secBase64);方法将加密后的base64发送到服务端。

3.3 质量校验设置

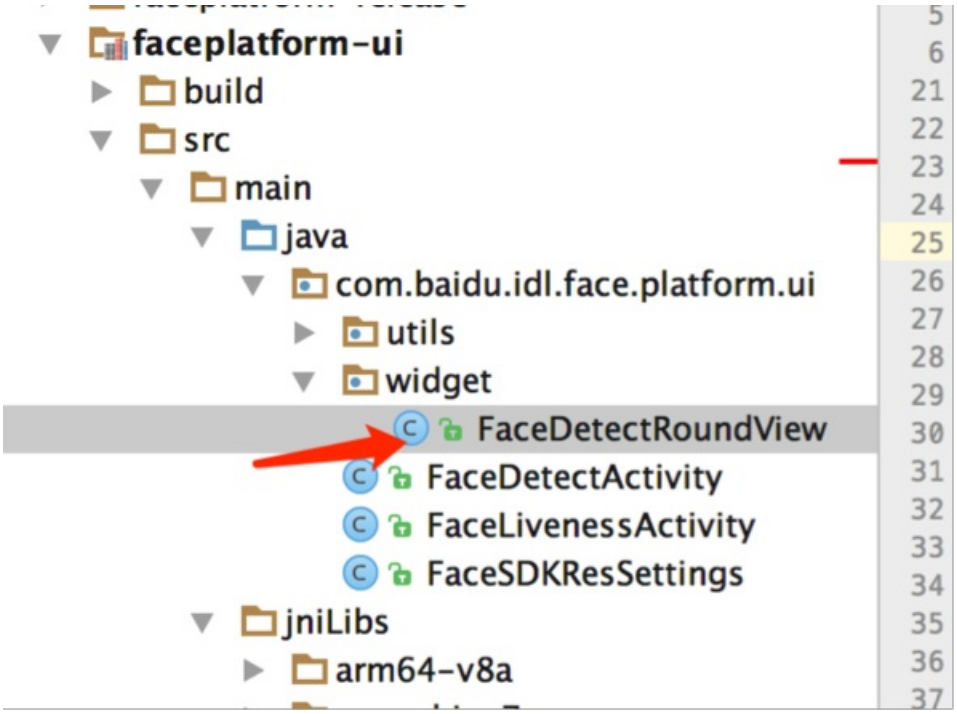
com.baidu.idl.face.platform.FaceConfig类用于人脸检测参数设置。

参数	名称	默认值	取值范围
minFaceSize	最小人脸阈值	200	
notFaceValue	非人脸阈值	0.6f	0~1.0f
brightnessValue	图片光照阈值	82f	0-255f
blurnessValue	图像模糊阈值	0.3f	0~1.0f
occlusionValue	人脸遮挡阈值	0.5f	0~1.0f
headPitchValue	低头抬头角度	9	0~45
headYawValue	左右摇头角度	9	0~45
eyeClosedValue	闭眼阈值	0.7f	0~1.0f
cacheImageNum	图片缓存数量（非动作活体使用）	3	

3.4 界面定制说明

3.4.1 修改faceplatform_ui界面

（1）如果SDK自带的UI和您的APP不统一，您可以自行修改FaceDetectRoundView。



（2）修改提示语音音频文件，有两种方式。

a、直接替换FaceUI工程raw下的mp3文件和string.xml。

b、FaceEnvironment 提供了setSoundId(FaceStatusNewEnum status, int soundId); 设置提示音资源。 FaceStatusNewEnum为不同的状态。soundId为资源文件所对应的resource id。

4、接口设计说明#

4.1 人脸功能管理器

4.1.1创建实例

- 方法

```
FaceSDKManager getInstance()
```

- 参数

  无

- 返回

  人脸功能管理器

- 说明

  创建人脸功能管理器

4.1.2 人脸功能管理器初始化

- 方法

```
public void initialize(final Context context, String licenseID, StringlicenseFileName, IInitCallback callback)
```

- 参数

  context 上下文环境，licenseID 传入申请License时获取的应用名称+_face_android后缀，licenseFileName 鉴权文件名称，callback 鉴权成功与否回调

- 返回

  无

- 说明

  初始化人脸检测功能。进行人脸检测功能License鉴权验证。

4.1.3 设置人脸功能控制参数

- 方法

```
void setFaceConfig(FaceConfig config)
```

- 参数

  config 人脸功能控制参数对象

- 返回

  无

- 说明

  设置人脸功能控制参数对象。

  FaceConfig对象参数：

  光照阈值

  图像模糊阈值

  人脸遮挡阈值

  头部姿态角度

  最小人脸检测阈值

  人脸检测精度阈值

  截取人脸图片大小

  进行活体检测的动作类型列表

  是否进行人脸图片质量检测

4.1.4 取得人脸图像采集功能接口

- 方法

```
IDetectStrategy getDetectStrategyModule()
```

- 参数

  无

- 返回

  人脸图像采集功能接口

- 说明

  取得人脸图像采集功能接口。人脸图像采集接口完成，解析图片人脸信息，返回检测结果。

4.1.5 取得活体检测功能接口

- 方法

```
ILivenessStrategy getLivenessStrategyModule()
```

- 参数

  无

- 返回

  活体检测功能接口

- 说明

  取得活体检测功能接口。活体检测功能接口完成，解析图片人脸信息，返回活体检测结果。

4.1.6 内存释放接口

- 方法

```
void release()
```

- 参数

  无

- 返回

  无

- 说明

  主要针对模型的释放，以减少内存

4.2 人脸图像采集器

人脸图像采集器IDetectStrategy，检测图片中人脸信息，返回人脸检测状态，完成人脸图像采集。

4.2.1 设置人脸图像采集功能参数

- 方法

```
void setDetectStrategyConfig(Rect previewRect, Rect detectRect,IDetectStrategyCallback callback)
```

- 参数

  previewRect 人脸图片大小，类型：Rect

  detectRect 人脸检测区域大小，类型：Rect

  callback 人脸图像采集功能状态监听器

- 返回

  无

- 说明

  设置人脸功能控制参数对象。

4.2.2 人脸图像采集

- 方法

```
void detectStrategy(byte[] imageData)
```

- 参数

  imageData 图片信息

- 返回 无

- 说明

  检测图片中的人脸信息，完成人脸图像采集，返回检测状态和结果。

4.3 活体检测器

活体检测器LivenessStrategy，检测图片人脸信息，活体检测结果状态。

4.3.1 设置人脸功能控制参数

- 方法

```
voidsetLivenessStrategyConfig(
    List<LivenessTypeEnum> livenessList,
    Rect previewRect,
    Rect detectRect,
    ILivenessStrategyCallback callback);
```

- 参数

  livenessList 活体动作列表

  previewRect 人脸图片大小，类型：Rect

  detectRect 人脸检测区域大小，类型：Rect

  callback 人脸图像采集功能状态监听器

- 返回

  无

- 说明

  设置活体检测功能参数对象。

4.3.2 活体检测

- 方法

```
void livenessStrategy(byte[] imageData);
```

- 参数

  imageData 图片信息

- 返回

  无

- 说明

  检测图片中的人脸信息，完成人脸图像采集和人脸活体检测，返回检测状态和结果。

4.4 人脸图像采集界面

人脸图像采集器界面FaceDetectActivity，包括UI界面，系统相机控制，使用人脸图像采集器IDetectStrategy处理相机采集到的图像。

4.5 活体检测界面

活体检测界面FaceLivenessActivity，包括UI界面，系统相机控制，使用活体检测器ILivenessStrategy处理相机采集到的图像，完

成活体检测功能。

4.6 图片加密请求类

图片加密请求类SecRequest，里面的sendMessage方法是为了测试图片加密之后secBase64是否可以通过云端解密的一个示例代码

5、常见问题

（1）license文件有什么用，该放在什么地方？

license文件需要申请，目的是作为sdk校验开发者的使用合法性，license文件放置位置不对或未放置license文件会导致没法使用sdk，一般应先申请license文件，并把申请得到的license文件，放置在assets目录下面。

（2）**FaceSDK-LicenseLICENSE_INFO_CHECK_ERROR =license id**

licenseID为您申请时填appname+"_face_android"。如下图demo-turnstile-face-android为license里面的licenseID, **demo-turnstile-face-android1**为app运行时Config.licenseID，两者必须一致

```
E/FaceSDK: FaceSDK-License LICENSE_INFO_CHECK_ERROR =license id  demo-turnstile-face-android  demo-turnstile-face-android1
```

（3）**FaceSDK-LicenseLICENSE_INFO_CHECK_ERROR =signature md5**

md5不一致错误，签名的为license里面的md5，后面的为app运行时获取的签名文件的md5，这两个md5必须一致且区分大小写。 `E/FaceSDK: FaceSDK-License LICENSE_INFO_CHECK_ERROR =signature md5  F5846C60804CC6042D55D09F1AB82364  4357A3EDBC0CA02EA8B5E0578E58D(`

（4）**FaceSDK-License LICENSE_INFO_CHECK_ERROR =package name**

PackageName不一致错误。License里面的packagename为申请license时填的，需要保证和app里面的packagename一致。

（5）活体检测常见有那些动作？是否可配置？

常见有7个动作，眨眼、张嘴、左摇头，右摇头，摇摇头、向上抬头，向下低头。 sdk提供FaceConfig参数设置类，如活体检测角度、光线，检测动作，检测动作数量等设置。

（6）使用sdk一般会用到活体检测拍照等功能，有什么需要注意？

Android 6.0+，需要注意相机拍摄权限问题。如没申请权限，可能导致没法调起相机。

（7）在有些机型上出现特别卡或出现无响应？

SDK在**armeabi**上性能非常差，建议删掉其他so只留下armeabi-v7a，包括使用的其他第三方so。因为如果其他so有armeabi，根据android系统查找so的逻辑，在armeabi的机型上只会去该目录下查找so，而人脸SDK没有，就会出现找不到so。

（8）license 文件失效了，不能用了怎么办？

license文件申请时候有期限，如过期会导致校验失效，需要在后台申请延期。

IOS SDK4.0

目录

目录~~
1、简介
1.1 功能介绍
1.2 兼容性
1.3 开发包说明
2、集成指南
2.1 准备工作
2.1.1 申请license
2.1.2 下载SDK
2.2 运行示例工程
2.2.1 自动配置授权信息集成
2.2.2 未使用自动配置授权信息的集成
2.3 添加SDK到工程
2.4 权限声明
3、接口说明
3.1 FaceSDK 鉴权初始化
3.1.1 设置鉴权功能
3.1.2 鉴权成功的凭证
3.2 FaceSDK 功能初始化
3.2.1 FaceSDK 参数配置
3.2.2 FaceSDK 初始化
3.2.3 FaceSDK 释放
3.3 人脸采集
3.4 动作采集
3.5 活体动作设置
4、常见问题

1、简介

1.1 功能介绍

百度Face SDK IOS 版是一种面向 IOS 移动设备人脸技术开发包，此版SDK包含人脸检测、活体识别等功能。

基于该方案，开发者可以轻松的构建包含人脸检测、采集和活体识别的应用。在您使用SDK之前，我们首先为您介绍以下SDK本身及相关人脸能力，以便您能更方便使用人脸服务。

此版SDK所包含的能力如下：

- **本地版活体检测**：通过让用户做出指定人脸配合式的交互动作，识别当前操作者是否为活体。此功能为离线使用，可设定指定动作是否使用及应用顺序。动作包含：眨眼、张嘴、左摇头，右摇头，摇摇头、向上抬头，向下低头。可有效抵御高清图片、3D建模、视频等攻击。**说明：用户完成相应动作即判断为活体，不返回相应分数。**

> 建议搭配线上的在线活体检测接口配合使用，在本地做动作活体检测，云端进行视频翻拍、 模型攻击以及合成图攻击的二次校验

- **本地版人脸质量检测**：判断视频流中的图片帧中，哪些图片质量较佳，即人脸图像特征清晰（满足角度、姿态、光照、模糊度等校验）。
- **本地版人脸图像采集**：通过本地SDK能力，采集人脸图像，同时经过人脸质量检测，确保采集到的人脸图像符合各条件校验（角度、姿态、光照、模糊度等），为设备前端获取有效可分析人脸的主要功能。
- **离线授权**：SDK的授权判断，授权介质也称为license，在SDK使用中，需要通过license向授权服务器发起请求，判断SDK的使用合法性及使用有效期。

> 此版SDK**全部功能为离线版本，所有功能均本地化使用**，主要用于在客户端（IOS）获取人脸，实际业务使用中，可以按照

> 业务需要，配合在线API完成全流程的业务集成。

1.2 兼容性

- **系统**：支持iOS8以上系统。需要开发者通过Deployment Target 来保证支持系统的检测。
- **机型**：手机，平板暂不支持
- **网络**：支持 WIFI 及移动网络,移动网络支持使用 NET 网关及WAP 网关(CMWAP、CTWAP、UNIWAP、3GWAP)。

1.3 开发包说明

文件/文件夹名	说明
FaceSDK	FaceSDK 包、bundle 资源文件、bundle 模型文件、鉴权文件
Public/Common	视频流处理类
Public/Utils	图像处理类
UI/Controller	DEMO工程
UI/View	View控制类
FaceParameterConfig.h	配置信息

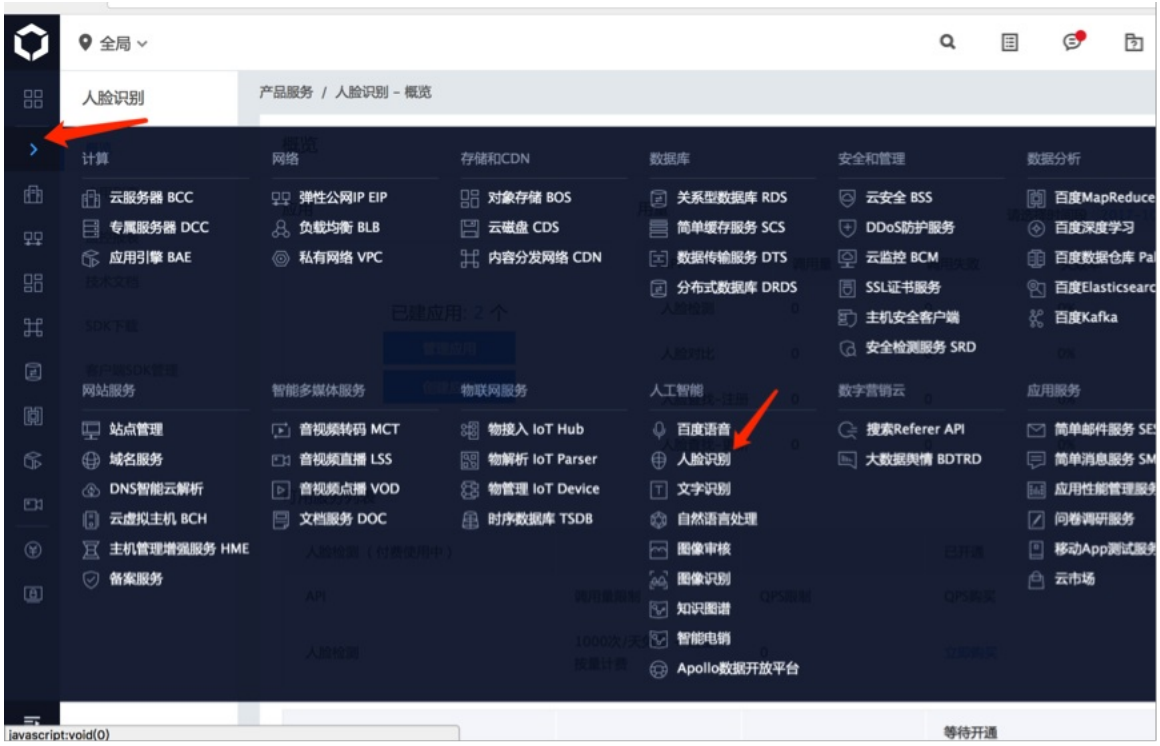
2、集成指南

2.1 准备工作

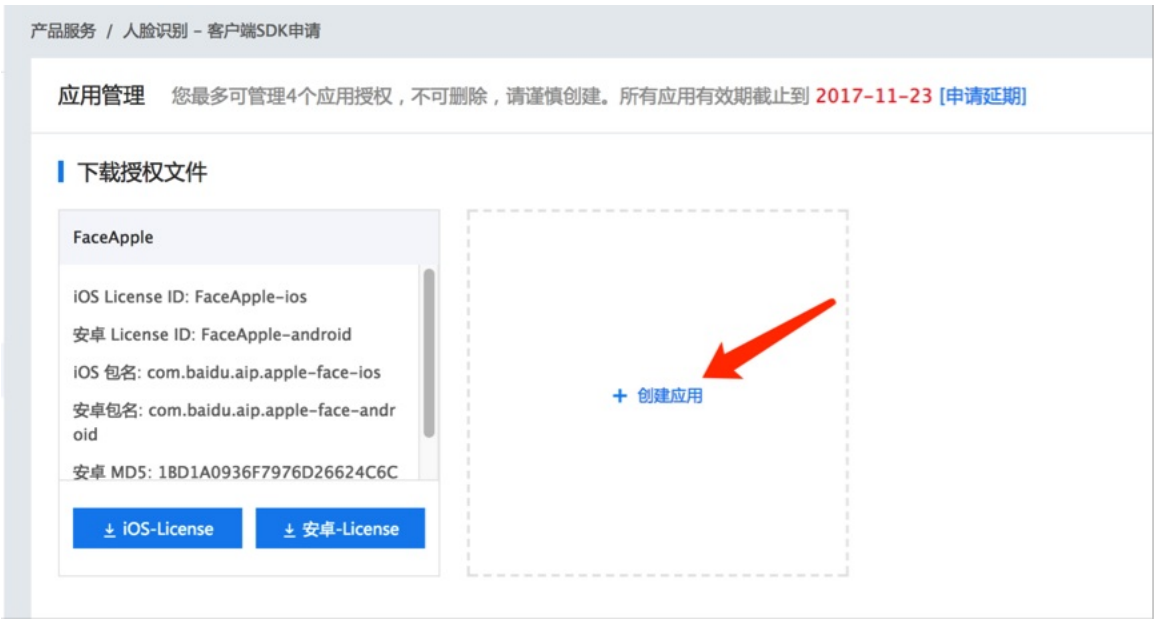
2.1.1 申请license

人脸SDK License：此license用于SDK离线功能使用，在您的申请人脸SDK的后台页面，全局->产品服 务->人脸识别->本地化部署->离线采集SDK申请

人脸控制台路径如下：

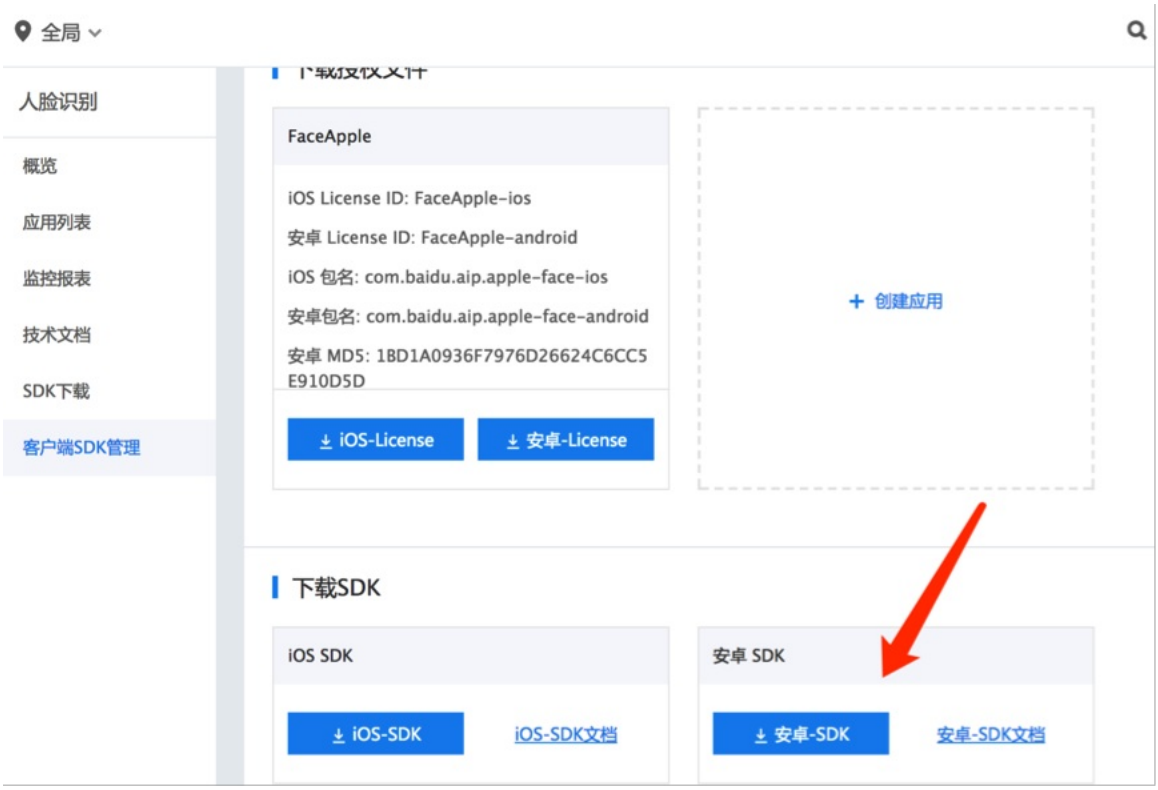


点击客户端SDK管理，弹出如下图：创建应用（这里创建应用是为了使用离线SDK，上面创建应用为了使用人脸在线接口，如注册、识别等）

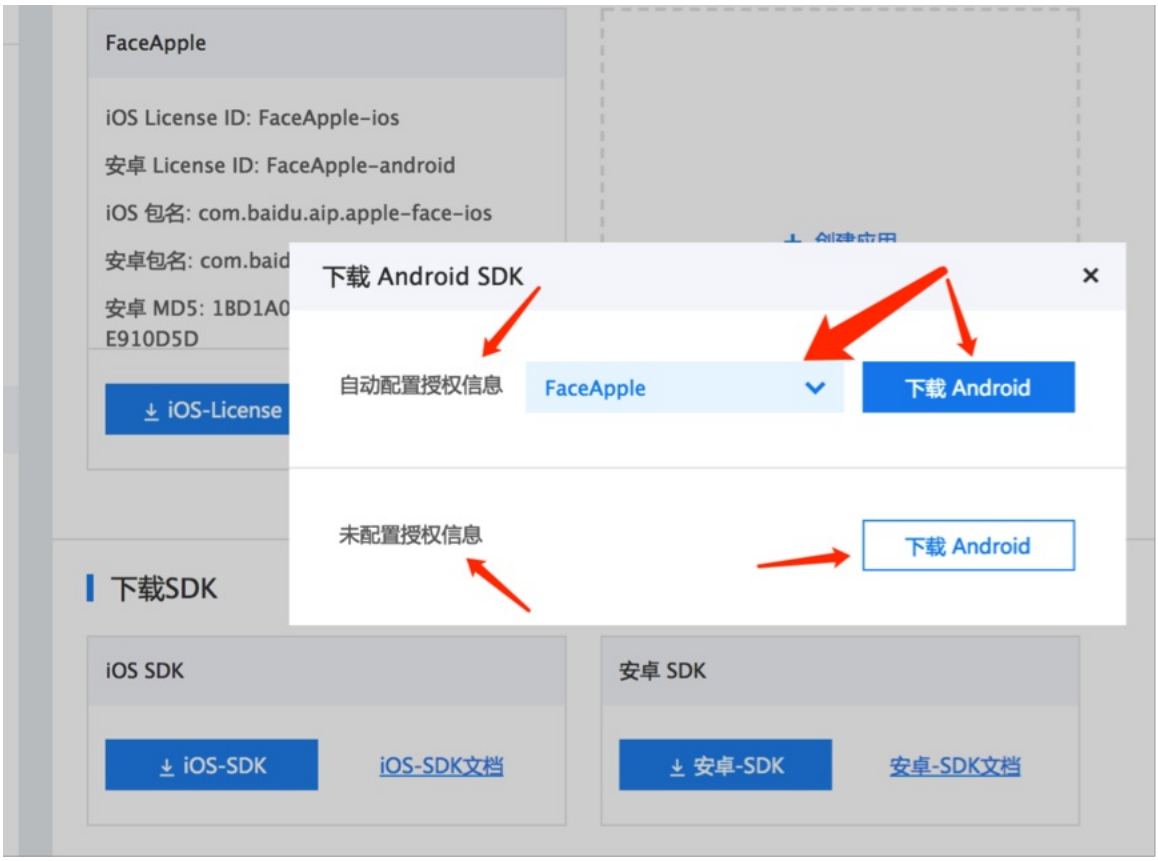


在弹出的框中输入授权标识，选择应用类型，应用系统，以及包名、MD5签名，详情请查看输入框右边提示

2.1.2 下载SDK



下载SDK分为自动配置授权信息（创建license后就可以选择为该应用，下载后SDK自动帮您配置授权，不用下载license拷贝到工程中，初始化参数licenseID,包名也帮您配置好了）和未配置授权信息两种方式：



2.2 运行示例工程

2.2.1 自动配置授权信息集成

如果您是通过自动配置授权信息下载的示例工程，只需配置好证书即可。查看下项目中的FaceParameterConfig.h文件，已经自动配置

```
11
12 // 人脸license文件名
13 #define FACE_LICENSE_NAME    @"idl-license"
14
15 // 人脸license后缀
16 #define FACE_LICENSE_SUFFIX  @"face-ios"
17
18 //  (您申请的应用名称(appname)+「-face-ios」后缀，如申请的应用名称(appname)为test123，则此处填写test123-face-ios)
19 #define FACE_LICENSE_ID        @"BaiduFaceDemo-face-ios"
20
```

示例图

配置好证书，即可运行。注意已经设置好的bundle id不要随意改动。

2.2.2 未使用自动配置授权信息的集成

手动导入授权信息。 需要手动导入license文件，配置好bundleID、licenseID等信息:

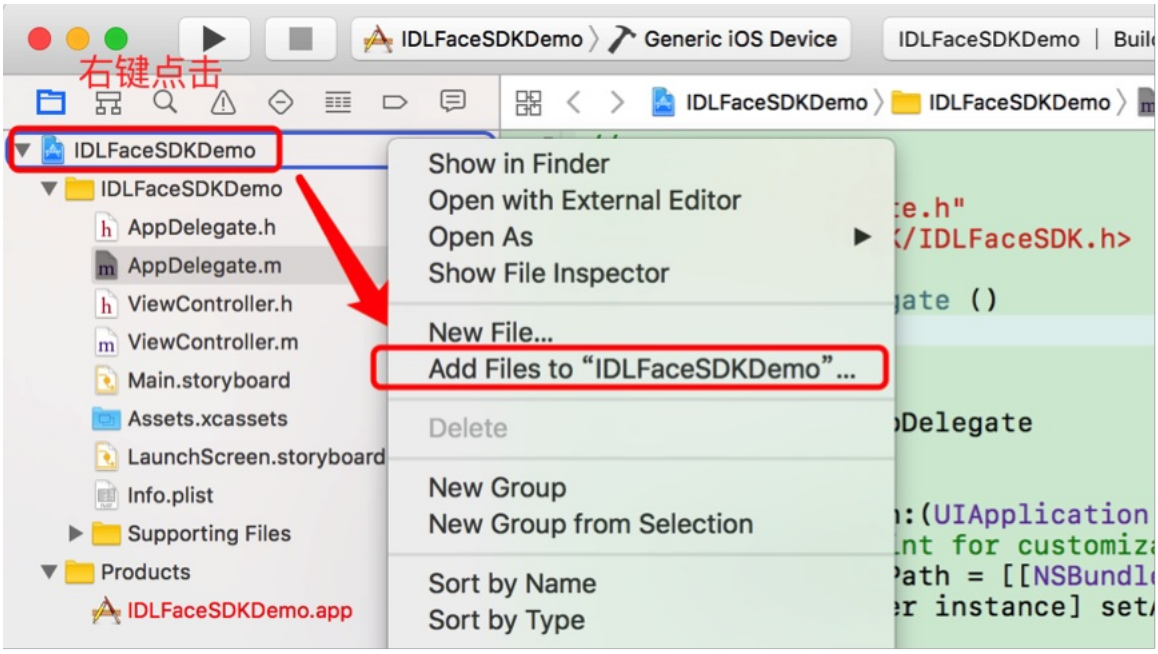
```
11
12 // 人脸license文件名
13 #define FACE_LICENSE_NAME    @"idl-license"
14
15 // 人脸license后缀
16 #define FACE_LICENSE_SUFFIX  @"face-ios"
17
18 //  (您申请的应用名称(appname)+「-face-ios」后缀，如申请的应用名称(appname)为test123，则此处填写test123-face-ios)
19 #define FACE_LICENSE_ID        @"BaiduFaceDemo-face-ios"
20
```

示例图

2.3 添加SDK到工程

1. 打开或者新建一个项目。

2. 右键点击项目，会出现一个添加菜单，在菜单中选择『Add Files to“此处是你的项目名字”…… 』,如下图所示：

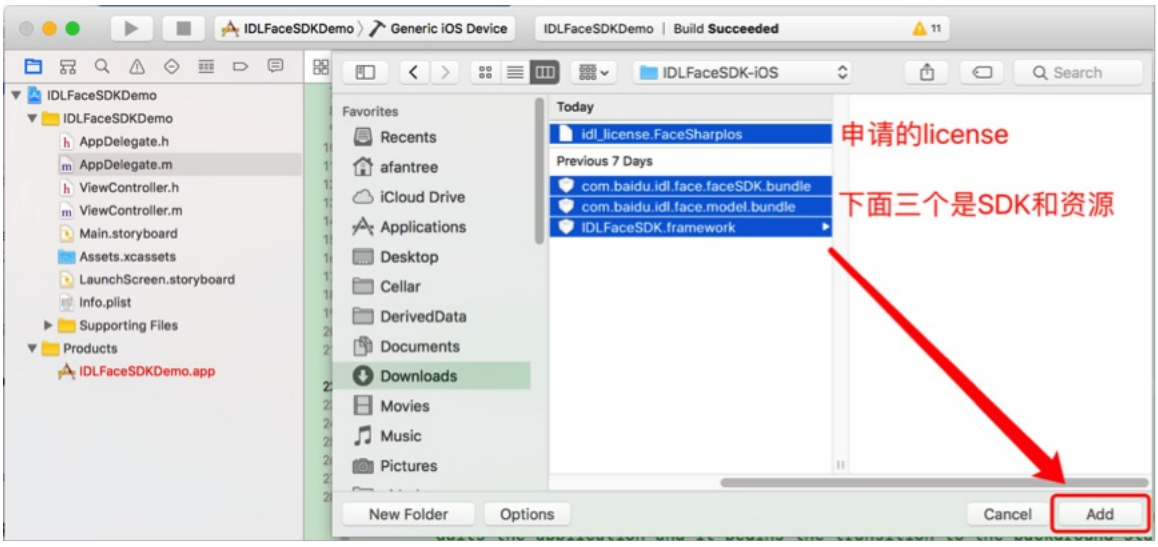


3. 在添加文件弹出框里面选择申请到的license和SDK添加进来。如下图：

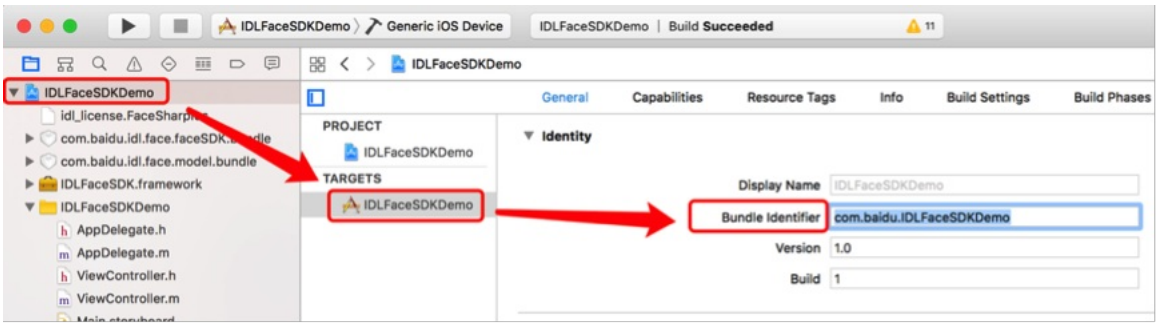
   注意：license为百度官方提供的。

   SDK包含下面三个文件:

   - IDLFaceSDK.framework
   - com.baidu.idl.face.faceSDK.bundle
   - com.baidu.idl.face.model.faceSDK.bundle



4. 确认下Bundle Identifier 是否是申请license时填报的那一个，注意：license和Bundle Identifier是一一对应关系，填错了会导致SDK不能用。



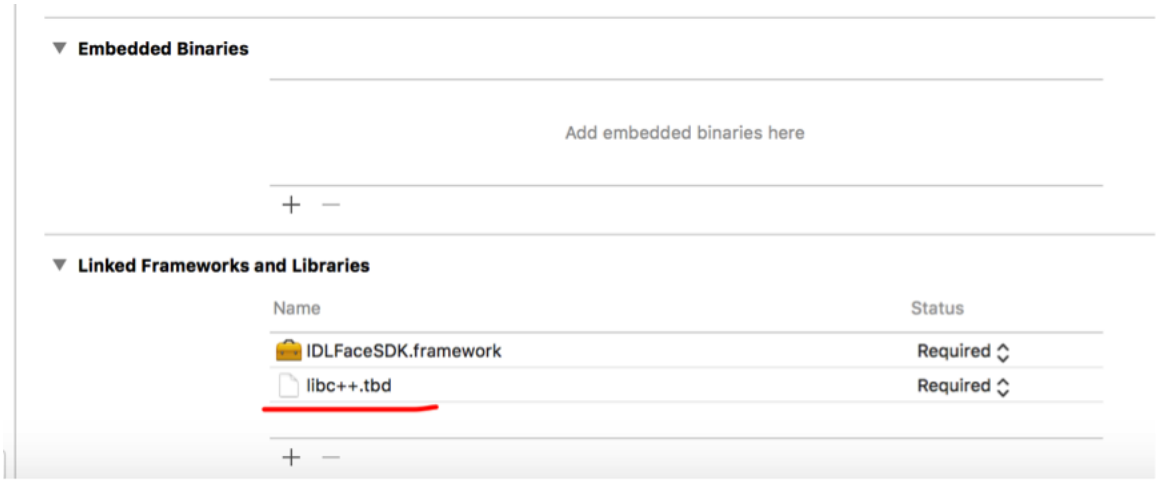
5. 填写正确的FACE_LICENSE_ID。

（即后台展示的LicenseID）

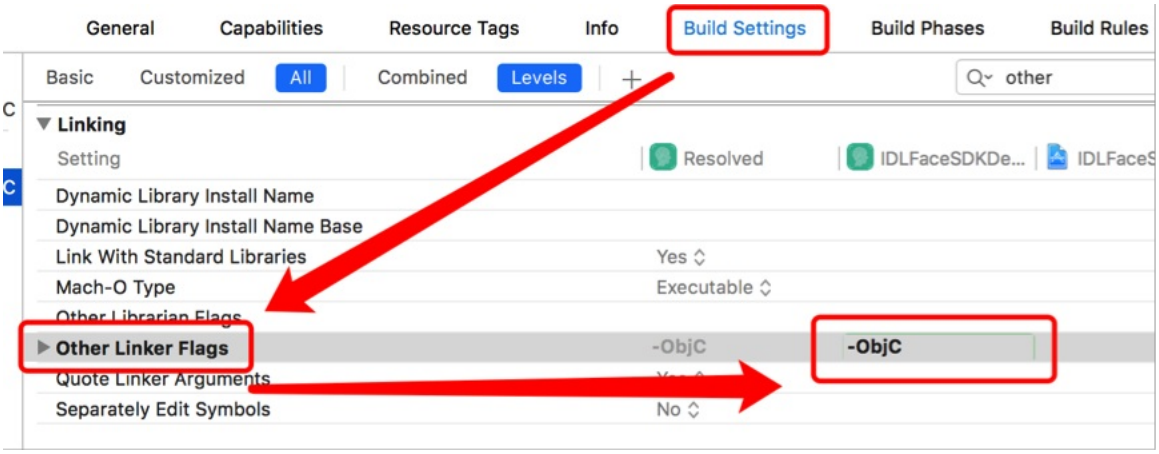
在FaceParameterConfig.h文件里面填写拼接好的FACE_LICENSE_ID。

```
//  (您申请的应用名称(appname)+「-face-ios」后缀，如申请的应用名称(appname)为test123，则此处填写test123-face-ios)
#define FACE_LICENSE_ID        @"BaiduFaceDemo-face-ios"
```

6. 选择链接C++标准库。



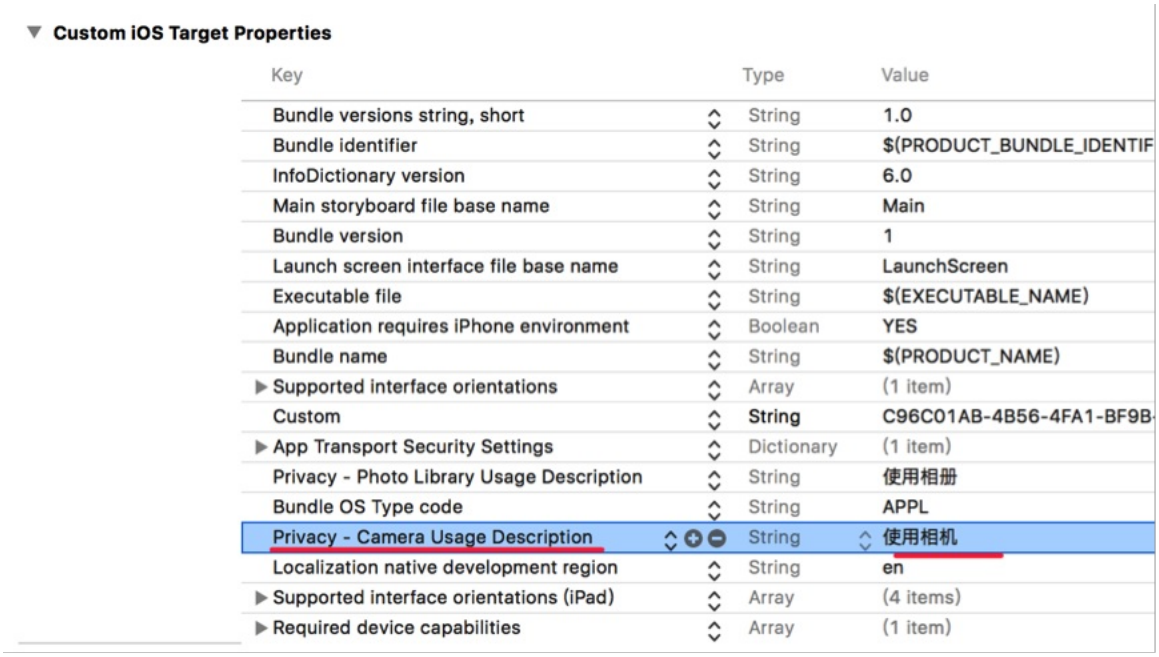
7. 如果没有使用pod管理第三方库的话，请在Build Setting >Linking > Other Linker Flags 上面加入 –ObjC 选项。如果用了pod请忽略，因为pod会自动添加上。



2.4 权限声明

需要使用相机权限：编辑Info.plist文件，添加

Privacy- Camera Usage Description 的Key值，Value为使用相机时候的提示语，可以填写：『使用相机』。



3、接口说明

3.1 FaceSDK 鉴权初始化

3.1.1 设置鉴权功能

```
 - (void)setLicenseID:(NSString *)licenseID andLocalLicenceFile:(NSString *)licensePath andRemoteAuthorize:(BOOL)remoteAuthorize;
```

参数：

- licenseId：平台申请的 licenseID
- localLicencePath：鉴权文件路径
- remoteAuthorize：是否开启网络鉴权

返回：

- 无

参考AppDelegate.m 实现，使用方法如下：

```
    NSString* licensePath = [[NSBundle mainBundle] pathForResource:FACE_LICENSE_NAME ofType:FACE_LICENSE_SUFFIX];
    NSAssert([[NSFileManager defaultManager] fileExistsAtPath:licensePath], @"license文件路径不对，请仔细查看文档");
    [[FaceSDKManager sharedInstance] setLicenseID:FACE_LICENSE_ID andLocalLicenceFile:licensePath andRemoteAuthorize:true];
```

3.1.2 鉴权成功的凭证

```
 - (BOOL)canWork
```

参数：

- 无

返回：

- True代表成功，false代表失败

参考ViewController.m 实现，判断鉴权是否通过：

```
 if (![[FaceSDKManager sharedInstance] canWork]){
  NSLog(@"授权失败，请检测ID 和 授权文件是否可用");
  return;
}
```

3.2 FaceSDK 功能初始化

3.2.1 FaceSDK 参数配置

具体方法详见如下：

```
 /**
 * 设置预测库耗能模式
 * 默认 LITE_POWER_NO_BIND
 */
- (void)setLitePower:(int)litePower;

/**
 * 需要检测的最大人脸数目
 * 默认1
 */
- (void)setMaxDetectNum:(int)detectNum；

/**
 * 需要检测的最小人脸大小
 * 默认40
 */
- (void)setMinFaceSize:(int)width;

/**
 * 人脸置信度阈值（检测分值大于该阈值认为是人脸
 * RGB
 * 默认 0.5f
 */
- (void)setNotFaceThreshold:(CGFloat)thr；

/**
 * 质量检测遮挡阈值
 * 默认0.5
 */
- (void)setOccluThreshold:(CGFloat)thr；

/**
 * 质量检测光照阈值
 * 默认100
 */
- (void)setIllumThreshold:(CGFloat)thr；

/**
 * 质量检测模糊阈值
 * 默认0.5
 */
- (void)setBlurThreshold:(CGFloat)thr；

/**
 * 姿态检测阈值
```

```
 * 姿态检测阈值
 * 默认pitch=12，yaw=12，row=10
 */
- (void)setEulurAngleThrPitch:(float)pitch yaw:(float)yaw roll:(float)roll；

/**
 * 输出图像个数
 * 默认3
 */
- (void)setMaxCropImageNum:(int)imageNum；

/**
 * 输出图像宽，设置为有效值(大于0)则对图像进行缩放，否则输出原图抠图结果
 * 默认 480
 */
- (void)setCropFaceSizeWidth:(CGFloat)width；

/**
 * 输出图像高，设置为有效值(大于0)则对图像进行缩放，否则输出原图抠图结果
 * 默认 680
 */
- (void)setCropFaceSizeHeight:(CGFloat)height；

/**
 * 输出图像，下巴扩展，大于等于0，0：不进行扩展
 * 默认0.1
 */
- (void)setCropChinExtend:(CGFloat)chinExtend；

/**
 * 输出图像，额头扩展，大于等于0，0：不进行扩展
 * 默认0.2
 */
- (void)setCropForeheadExtend:(CGFloat)foreheadExtend；

/**
 * 输出图像，人脸框与背景比例，大于等于1，1：不进行扩展
 * 默认1.5f
 */
- (void)setCropEnlargeRatio:(float)cropEnlargeRatio;

/**
 * 动作超时配置
 */
- (void)setConditionTimeout:(CGFloat)timeout；

/**
 * 语音间隔提醒配置
 */
- (void)setIntervalOfVoiceRemind:(CGFloat)timeout;

/**
 * 是否开启静默活体，默认false
 */
- (void)setIsCheckSilentLive:(BOOL)isCheck;

/**
 * 口罩检测阈值配置，默认0.8。
 * 大于阈值判定为戴口罩，低于阈值判定为未戴口罩
 */
- (void)setMouthMaskThreshold:(CGFloat)thr；

/**
```

```
 * 设置原始图片缩放比例，默认1不缩放，scale 阈值0~1
 */
- (void)setImageWithScale:(CGFloat)scale;

/**
 * 设置图片加密类型，type=0 基于base64 加密；type=1 基于百度安全算法加密
 */
- (void)setImageEncrypteWithType:(int) type;
```

3.2.2 FaceSDK 初始化

```
 - (int)initCollect;
```

参数:

- 无

返回：

- 无

参考ViewController.m initSDK 方法实现：

```
- (void) initSDK {

    if (![[FaceSDKManager sharedInstance] canWork]){
        NSLog(@"授权失败，请检测ID 和 授权文件是否可用");
        return;
    }

    // 初始化SDK配置参数，可使用默认配置
    // 设置最小检测人脸阈值
    [[FaceSDKManager sharedInstance] setMinFaceSize:200];
    // 设置截取人脸图片高
    [[FaceSDKManager sharedInstance] setCropFaceSizeWidth:400];
    // 设置截取人脸图片宽
    [[FaceSDKManager sharedInstance] setCropFaceSizeHeight:640];
    // 设置人脸遮挡阈值
    [[FaceSDKManager sharedInstance] setOccluThreshold:0.5];
    // 设置亮度阈值
    [[FaceSDKManager sharedInstance] setIllumThreshold:40];
    // 设置图像模糊阈值
    [[FaceSDKManager sharedInstance] setBlurThreshold:0.3];
    // 设置头部姿态角度
    [[FaceSDKManager sharedInstance] setEulurAngleThrPitch:10 yaw:10 roll:10];
    // 设置人脸检测精度阈值
    [[FaceSDKManager sharedInstance] setNotFaceThreshold:0.6];
    // 设置抠图的缩放倍数
    [[FaceSDKManager sharedInstance] setCropEnlargeRatio:3.0];
    // 设置照片采集张数
    [[FaceSDKManager sharedInstance] setMaxCropImageNum:6];
    // 设置超时时间
    [[FaceSDKManager sharedInstance] setConditionTimeout:1500];
    // 设置开启口罩检测，非动作活体检测可以采集戴口罩图片
    [[FaceSDKManager sharedInstance] setIsCheckMouthMask:true];
    // 设置开启口罩检测情况下，非动作活体检测口罩过滤阈值，默认0.8 不需要修改
    [[FaceSDKManager sharedInstance] setMouthMaskThreshold:0.8f];
    // 设置原始图缩放比例
    [[FaceSDKManager sharedInstance] setImageWithScale:0.8f];
    // 设置图片加密类型，type=0 基于base64 加密；type=1 基于百度安全算法加密
    [[FaceSDKManager sharedInstance] setImageEncrypteType:0];
    // 初始化SDK功能函数
    [[FaceSDKManager sharedInstance] initCollect];
}
```

3.2.3 FaceSDK 释放

```
- (int)uninitCollect
```

参数:

- 无

返回：

- 无

3.3 人脸采集

调用IDLFaceDetectionManager类的:

```
 /**
 * 人脸采集，成功之后返回扣图图片，原始图片
 * @param image 镜头拿到的图片
 * @param previewRect 预览的Rect
 * @param detectRect 检测的Rect
 * return completion 回调信息
 */
- (void)detectStratrgyWithNormalImage:(UIImage *)image previewRect:(CGRect)previewRect detectRect:(CGRect)detectRect completionHandler:(DetectStrategyCompletion)completion;

typedef void (^DetectStrategyCompletion) (FaceInfo * faceinfo,NSDictionary * images, DetectRemindCode remindCode);
```

温馨提示：NSDictionary * images 返回FaceCropImageInfo 对象包含带黑边的图片，主要是为了配合在线 API 方式的活体检测使用，本地算法会对图片做一定的预处理操作；而不带黑边的图片，如果无需使用活体，则可使用不带黑边的图片采集方式。

参数说明：

`previewRect`与`detctRect`是为了做距离检测而定义的，为了在上层封装判断脸是否在框内／离太远／离太近。

- image：相机获取的图片
- previewRect： 人脸图片大小，间接定义的最大距离的 maxRect 和最小距离的 minRect。
- detectRect：人脸检测区域大小，实际采集区域
- completion：完成后返回照片和状态结果

参考BDFaceDetectionViewController.m 实现，使用方法如下：

```
        __weak typeof(self) weakSelf = self;
    [[IDLFaceDetectionManager sharedInstance] detectStratrgyWithNormalImage:image previewRect:self.previewRect
detectRect:self.detectRect completionHandler:^(FaceInfo *faceinfo, NSDictionary *images, DetectRemindCode
remindCode) {
        switch (remindCode) {
          case DetectRemindCodeOK: {
              weakSelf.hasFinished = YES;
              [self warningStatus:CommonStatus warning:@"非常好"];
              if (images[@"image"] != nil && [images[@"image"] count] != 0) {

                  NSArray *imageArr = images[@"image"];
                  for (FaceCropImageInfo * image in imageArr) {
                      NSLog(@"cropImageWithBlack %f %f", image.cropImageWithBlack.size.height,
image.cropImageWithBlack.size.width);
                      NSLog(@"originalImage %f %f", image.originalImage.size.height, image.originalImage.size.width);
                  }

                  FaceCropImageInfo * bestImage = imageArr[0];
                  [[BDFaceImageShow sharedInstance] setSuccessImage:bestImage.originalImage];
                  [[BDFaceImageShow sharedInstance] setSilentliveScore:bestImage.silentliveScore];
                  // 公安验证接口测试
                  [self request:bestImage.cropImageWithBlackEncryptStr];
                  dispatch_async(dispatch_get_main_queue(), ^{
                      UIViewController* fatherViewController = weakSelf.presentingViewController;
                      [weakSelf dismissViewControllerAnimated:YES completion:^{
                          BDFaceSuccessViewController *avc = [[BDFaceSuccessViewController alloc] init];

                          [fatherViewController presentViewController:avc animated:YES completion:nil];
                      }];

                  });
              }
              [self singleActionSuccess:true];
              break;
```

3.4 动作采集

调用IDLFaceLivenessManager类的:

```
 /**
 *  人脸活体验证，成功之后扣图图片，原始图片三种
 * @param image 镜头拿到的图片
 * @param previewRect 预览的Rect
 * @param detectRect 检测的Rect
 * return completion 回调信息
 */
-(void) livenessNormalWithImage:(UIImage *)image previewRect:(CGRect)previewRect detectRect:(CGRect)detectRect
completionHandler:(LivenessNormalCompletion)completion;

typedef void (^LivenessNormalCompletion) (NSDictionary * images, FaceInfo *faceInfo, LivenessRemindCode
remindCode);
```

温馨提示：NSDictionary * images 返回FaceCropImageInfo 对象包含带黑边的图片，主要是为了配合在线 API 方式的活体检测使用，本地算法会对图片做一定的预处理操作；而不带黑边的图片，如果无需使用活体，则可使用不带黑边的图片采集方式。

参数说明：

`previewRect`与`detctRect`是为了做距离检测而定义的，为了在上层封装判断脸是否在框内／离太远／离太近。

- image：相机获取的图片
- previewRect： 人脸图片大小，间接定义的最大距离的 maxRect 和最小距离的 minRect。
- detectRect：人脸检测区域大小，实际采集区域
- completion：完成后返回照片和状态结果

说明：

- 检测图片中的人脸信息，完成人脸图像采集和人脸活体检测，返回检测状态和结果。

```
    [[IDLFaceLivenessManager sharedInstance] livenessNormalWithImage:image previewRect:self.previewRect
detectRect:self.detectRect completionHandler:^(NSDictionary *images, FaceInfo *faceInfo, LivenessRemindCode
remindCode) {

        switch (remindCode) {
            case LivenessRemindCodeOK: {
                weakSelf.hasFinished = YES;
                [self warningStatus:CommonStatus warning:@"非常好"];
                if (images[@"image"] != nil && [images[@"image"] count] != 0) {

                    NSArray *imageArr = images[@"image"];
                    for (FaceCropImageInfo * image in imageArr) {
                        NSLog(@"cropImageWithBlack %f %f", image.cropImageWithBlack.size.height,
image.cropImageWithBlack.size.width);
                        NSLog(@"originalImage %f %f", image.originalImage.size.height, image.originalImage.size.width);
                    }

                    FaceCropImageInfo * bestImage = imageArr[0];
```

3.5 活体动作设置

调用IDLFaceLivenessManager类的:

```
- (void)livenesswithList:(NSArray *)array order:(BOOL)ordernumberOfLiveness:(NSInteger)numberOfLiveness
```

参数：

- array: 活体动作列表
- order: 是否按顺序进行活体动作
- numberOfLiveness: 活体动作数目（array为nil是起作用）

返回：

- 无

说明：

- 活体动作设置

4、常见问题

Q：鉴权问题。提示「验证失败」 A：先确定网络情况是否正常，本地鉴权文件失效了才走网络鉴权。定位错误码，排查鉴权失败的原因。一般是 licenseID 和 bundleID 配置不一致导致的鉴权失败。请注意上线前授权文件一定要更新。

Q：license 文件失效了，不能用了怎么办？ A：License 文件申请时候有期限，如过期会导致校验失效，需要在后台进行申请延期。

Q：使用 iOS 采集端，采集到的图片是斜着的，这个正常吗，会影响识别吗？ A：不会影响识别。有黑边和倾斜是因为图片质量算法造成的，我们是按 1:3 对图像进行背景填充使人脸居中，为的是更好的识别图像。这个版本提供了`detectStratrgyWithQualityControlImage`和`detectStratrgyWithNormalImage`两种方法供选择。

更多问题请点击 [常见问题](

V3.3.0.0版本

安卓-基础版

目录

1 简介
1.1 功能介绍
1.2 兼容性
1.3 开发包说明
2 集成指南
2.1 Sample示例工程说明
2.2 准备工作
2.2.1 申请license
2.2.2 下载SDK
2.3 运行示例工程
2.3.1 运行自动配置授权信息的示例工程
2.3.2 运行未配置授权信息的示例工程
2.4添加SDK到工程
2.5权限声明
3 功能使用
3.1人脸检测
3.2质量校验设置
3.3 界面定制说明
3.3.1 修改ui界面
4 接口设计说明
4.1人脸功能管理器
4.1.1 创建实例
4.1.2人脸功能管理器初始化
4.1.3设置人脸功能控制参数
4.1.4取得人脸图像采集功能接口
5常见问题

1、简介

百度Face SDK Android 版是一种面向 Android 移动设备人脸技术开发包，此版SDK包含人脸检测、活体识别等功能，以aar包+动态链接库的形式发布。基于该方案，开发者可以轻松的构建包含人脸检测、采集和活体识别的应用。在您使用SDK之前，我们首先为您介绍以下SDK本身及相关人脸能力，以便您能更方便使用人脸服务。

1.1 功能介绍

此版SDK所包含的能力如下：

- **本地版活体检测**：通过让用户做出指定人脸配合式的交互动作，识别当前操作者是否为活体，此功能为离线使用，可设定指定动作是否使用及应用顺序。动作包含：眨眼、张嘴、左摇头，右摇头，摇摇头、向上抬头，向下低头七个。可有效抵御高清图片、3D建模、视频等攻击。
- **本地版人脸质量检测**：判断视频流中的图片帧中，哪些图片质量较佳，即人脸图像特征清晰（满足角度、姿态、光照、模糊度等校验）。
- **本地版人脸图像采集**：通过本地SDK能力，采集人脸图像，同时经过人脸质量检测，确保采集到的人脸图像符合各条件校验（角度、姿态、光照、模糊度等），为设备前端获取有效可分析人脸的主要功能。
- **离线授权**：SDK的授权判断，授权介质也称为license，在SDK使用中，需要通过license向授权服务器发起请求，判断SDK的

使用合法性及使用有效期。

> 此版SDK**全部功能为离线版本，所有功能均本地化使用**，主要用于在客户端（Android）获取人脸，实际业务使用中，可以按照业务需要，配合在线API完成全流程的业务集成。

为了方便您的开发，我们已经为您准备了多种场景的示例工程，您可以根据业务需要，在后台进行直接下载，目前支持【人脸核身】【人脸闸机/门禁】【人脸登录/考勤】【多人脸识别】,示例工程参考下图：

1.2 兼容性

系统：支持 Android 4.0.3(API Level 15)及以上系统。需要开发者通过 minSdkVersion来保证支持系统的检测。

机型：手机和平板皆可

构架：支持 CPU架构平台【arm-v7，arm-64，x86】

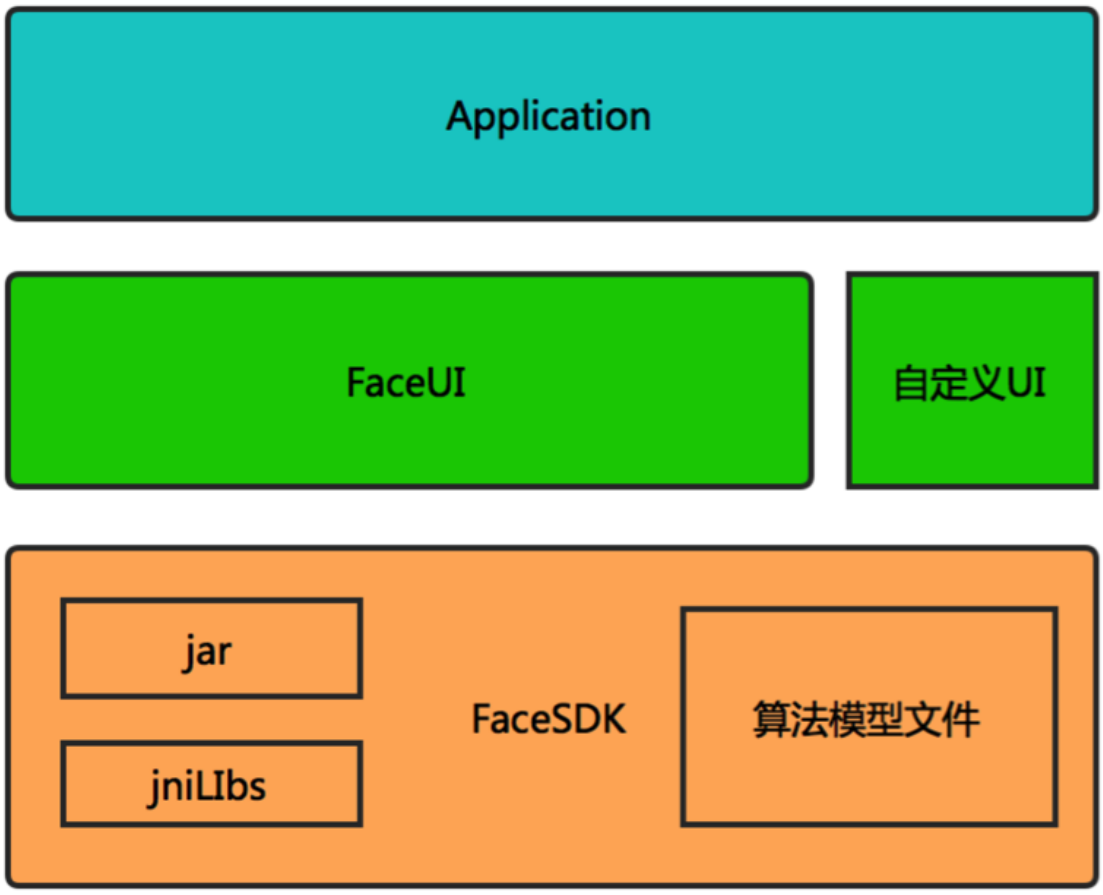
网络：支持 WIFI 及移动网络,移动网络支持使用NET 网关及 WAP 网关(CMWAP、CTWAP、UNIWAP、3GWAP)。

1.3 开发包说明

文件/文件夹名	说明
/faceplatform-release	SDK lib 库相关代码的 aar
/faceplatform-ui	SDK的UI库，封装拍照裁剪等功能,以及各平台的so库。so包含以下几个平台如果关注包大小，请自行删减。/armeabi-v7a/arm64-v8a/x86，
FacePlatform/	DEMO工程

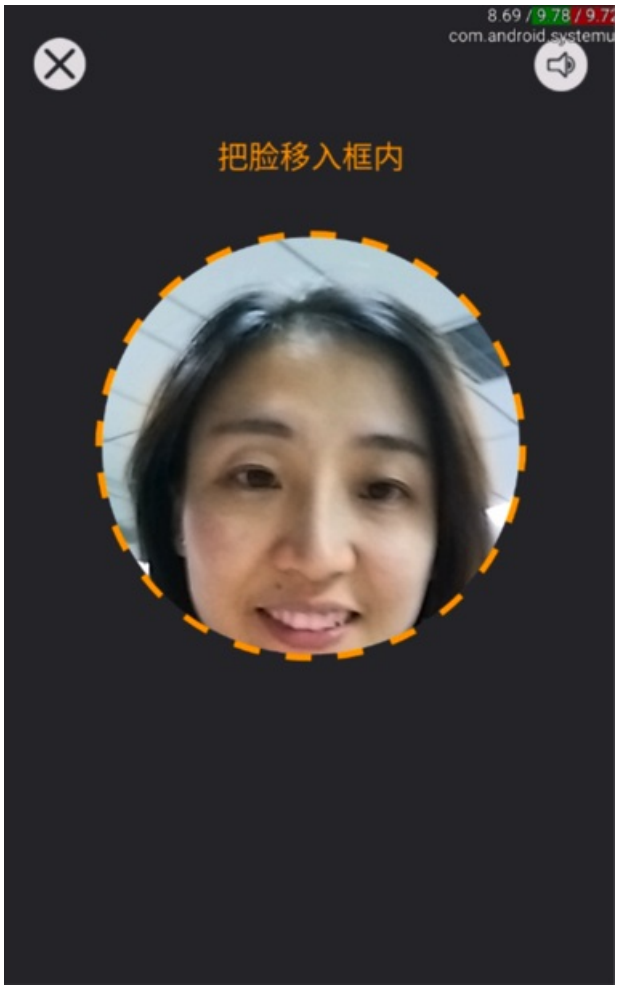
2、集成指南

本章将进行 Step-By-Step的讲解,如何快速的集成 人脸Sdk到现有应用中。一个完整的Demo 请参考开发包中的示例程序 FacePlatform。方案架构参考下图：

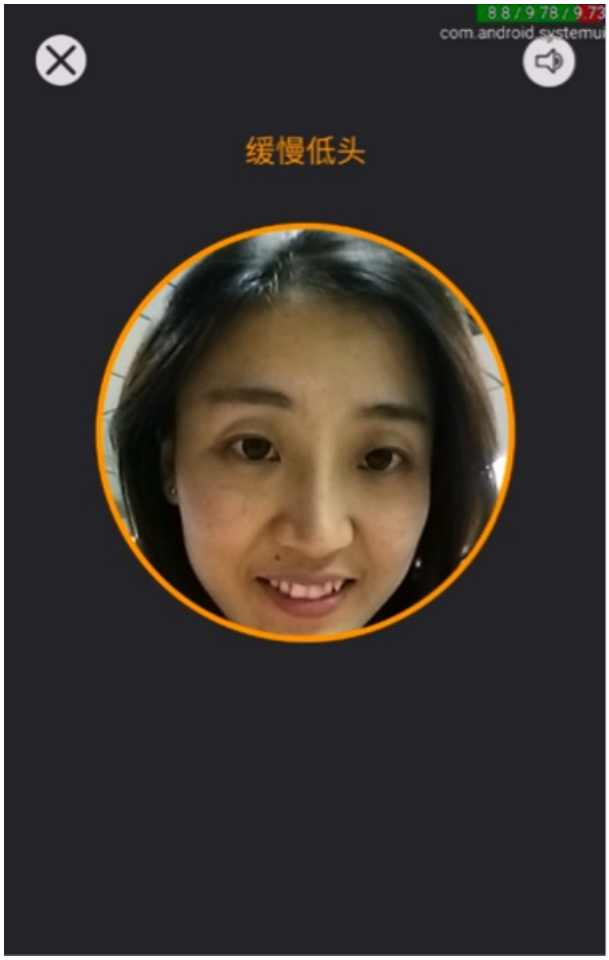


2.1 Sample示例

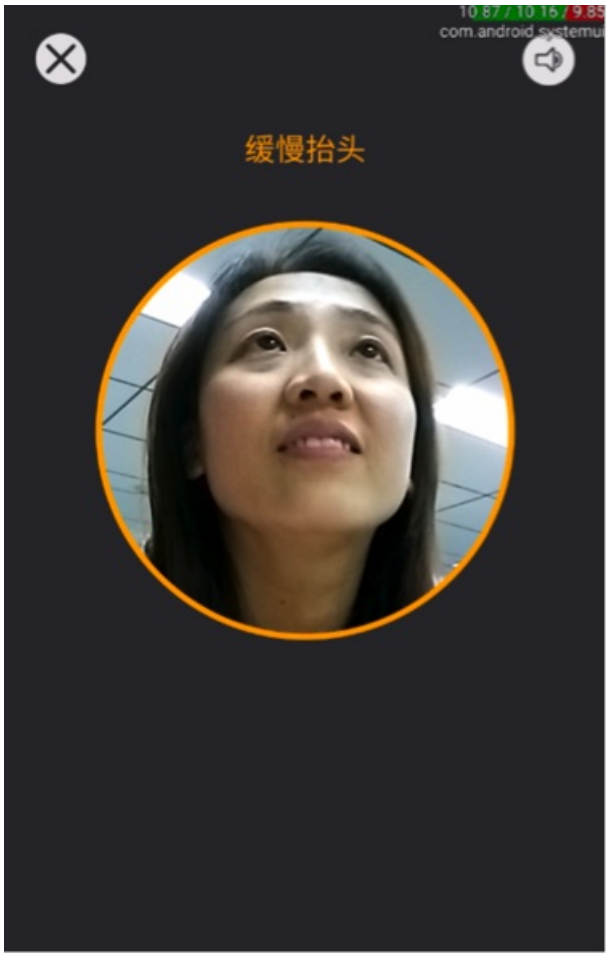
- 把脸移入框内



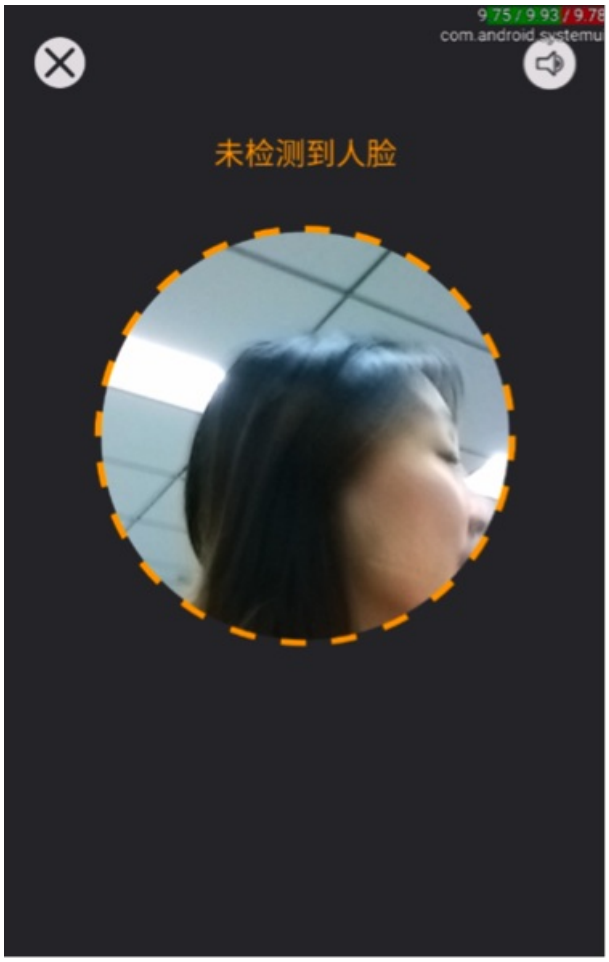
- 慢慢低头



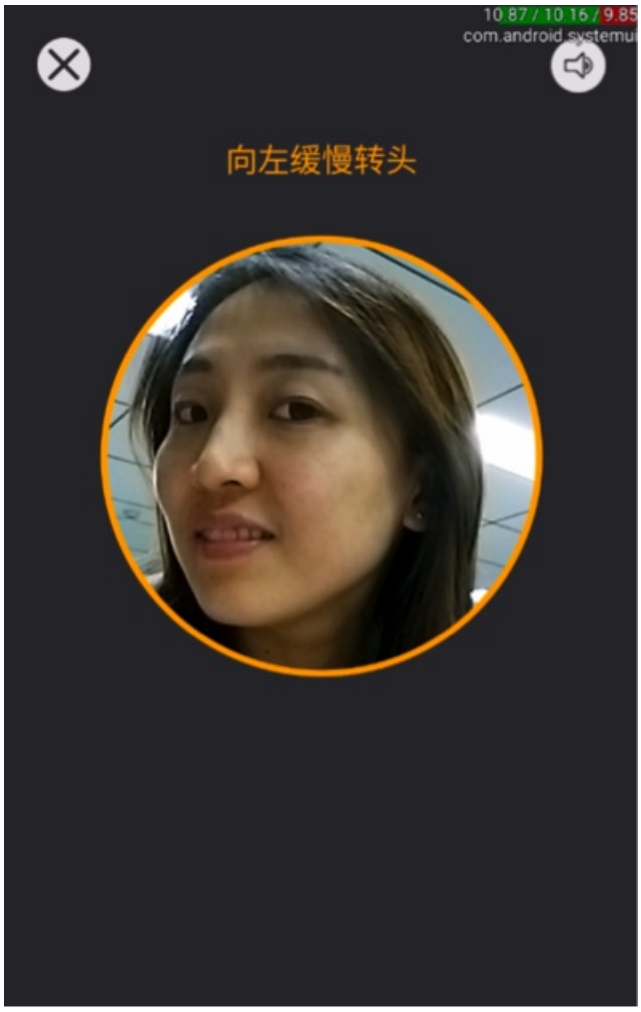
- 慢慢抬头



- 未检测到人脸



- 向左缓慢摇头

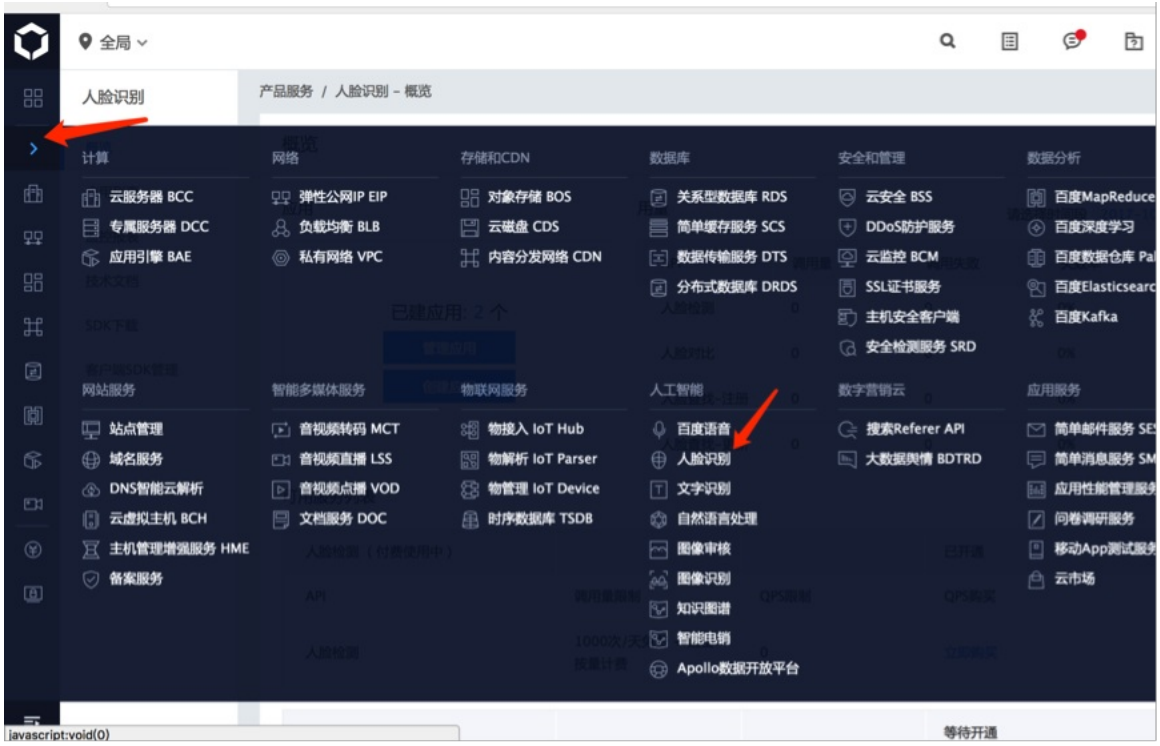


2.2 准备工作

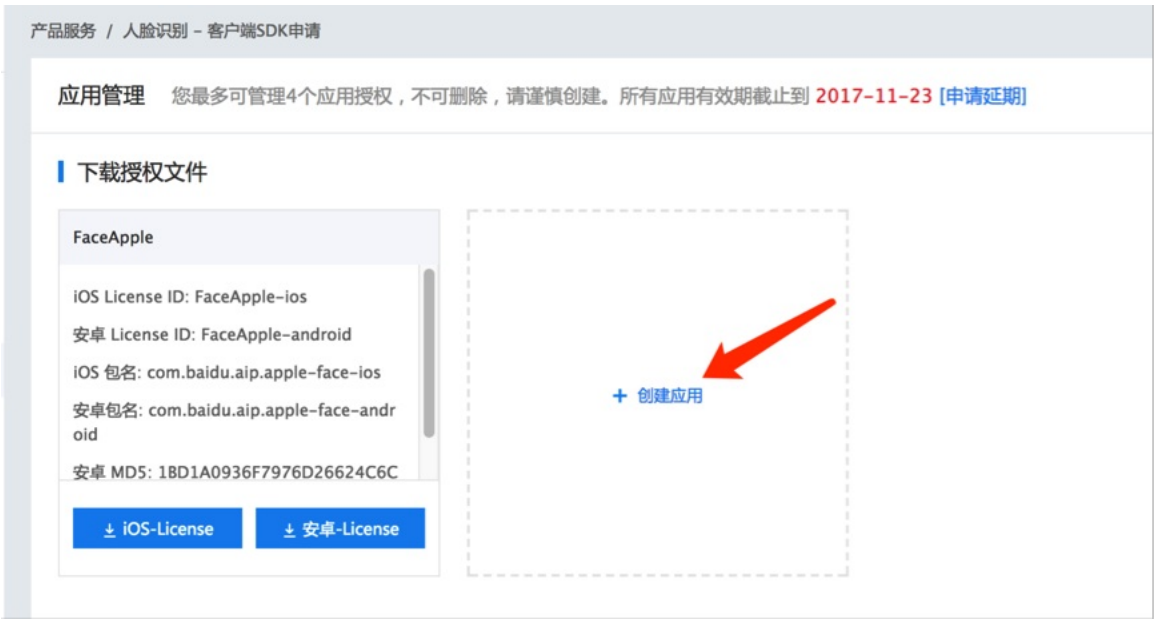
2.2.1 申请license

人脸SDK License：此license用于SDK离线功能使用，在您的申请人脸SDK的后台页面，全局->产品服务->人脸识别->客户端SDK申请

人脸控制台路径如下：

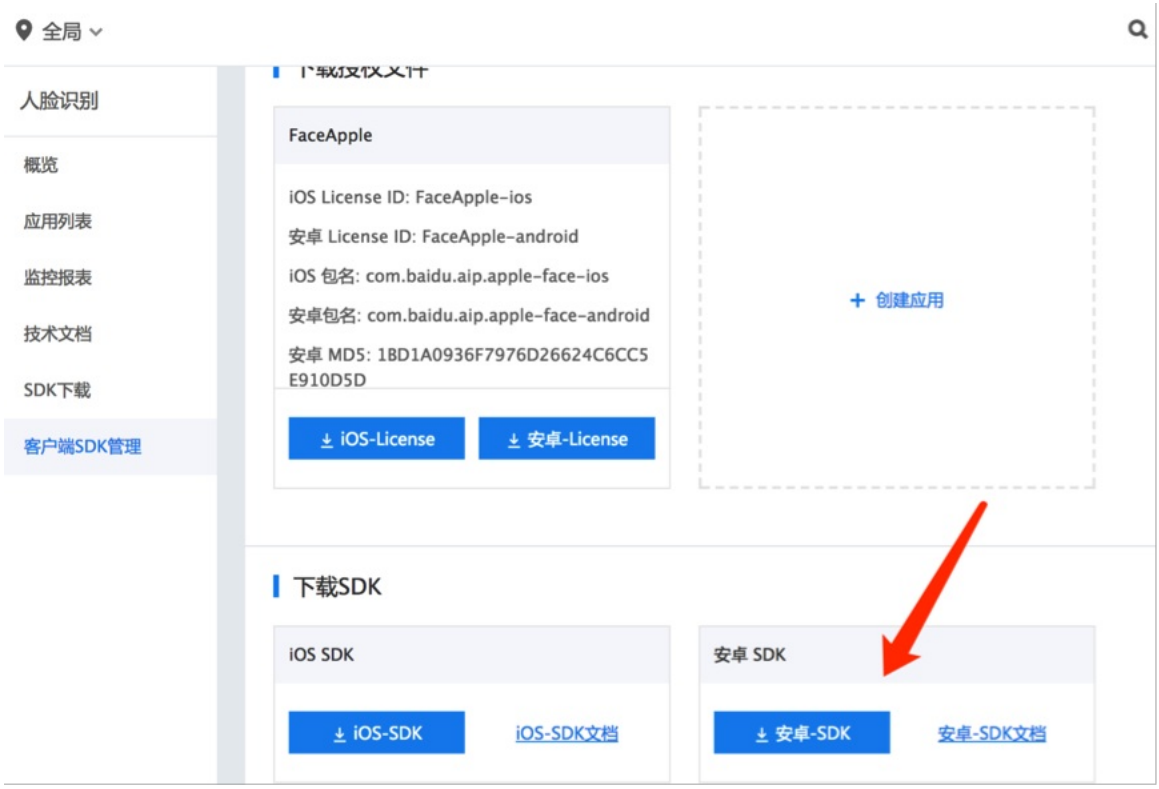


点击客户端SDK管理，弹出如下图：创建应用（这里创建应用是为了使用离线SDK，上面创建应用为了使用人脸在线接口，如注册、识别等）

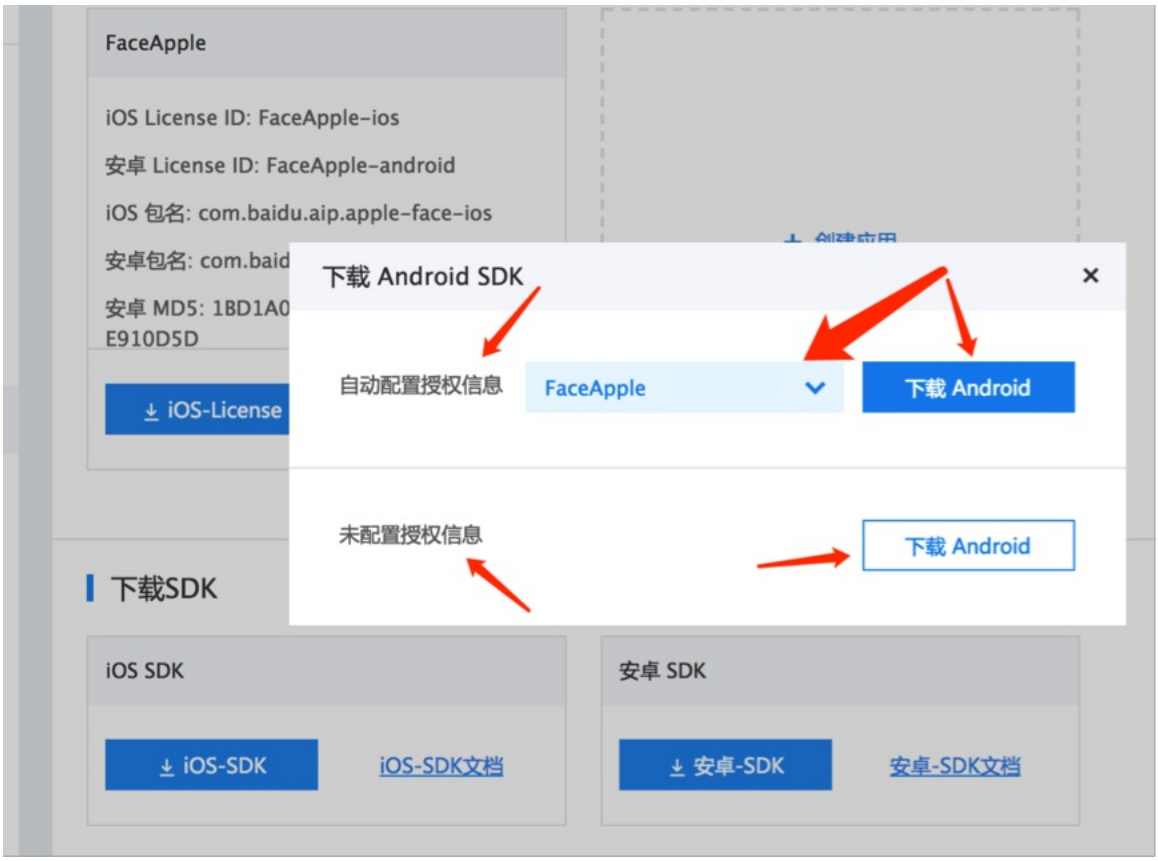


在弹出的框中输入授权标识，选择应用类型，应用系统，以及包名、MD5签名，详情请查看输入框右边提示

2.2.2 下载SDK



下载SDK分为自动配置授权信息（创建license后就可以选择为该应用，下载后SDK自动帮您配置授权，不用下载license拷贝到工程中，初始化参数licenseID,包名也帮您配置好了）和未配置授权信息两种方式：



2.3 运行示例工程

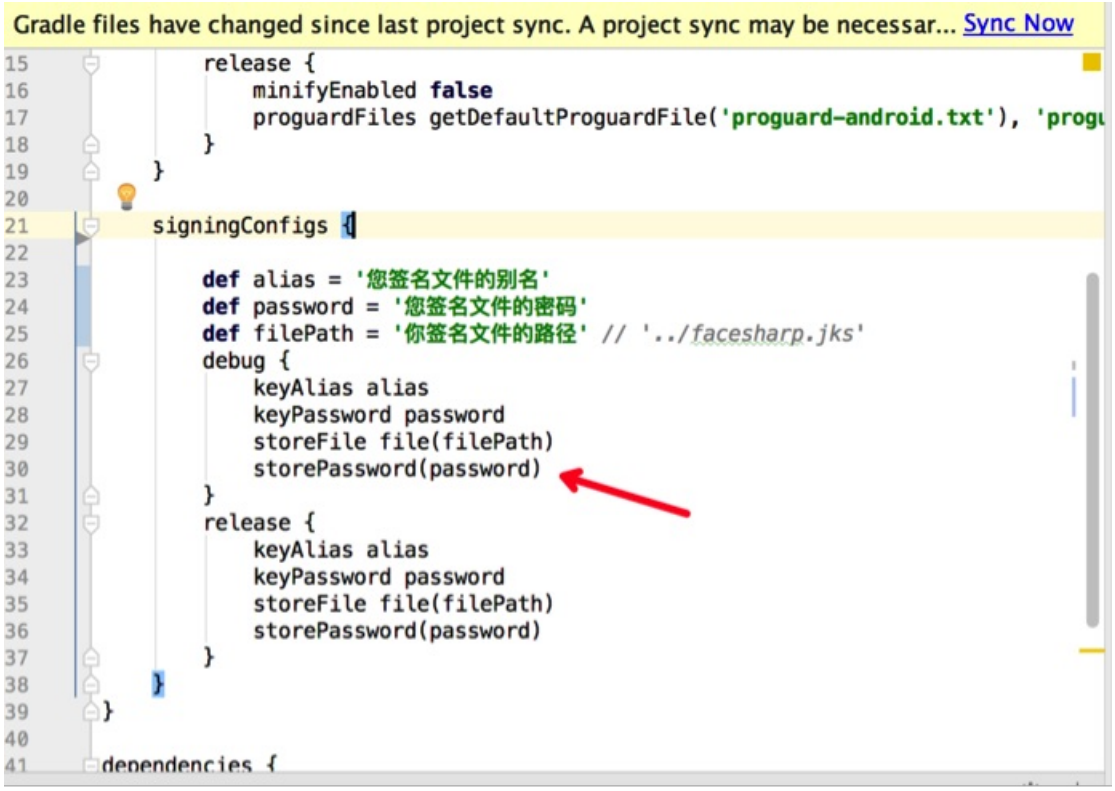
2.3.1 运行自动配置授权信息的示例工程

该下载的示例工程，已经帮你改好了license和包名

（1）Android Studio导入下载的示例工程

（2）配置打包签名文件，由于SDK运行时会校验签名文件的MD5是否和申请时填入一样。为了便于debug能正常使用SDK，需要配置打包签名文件。

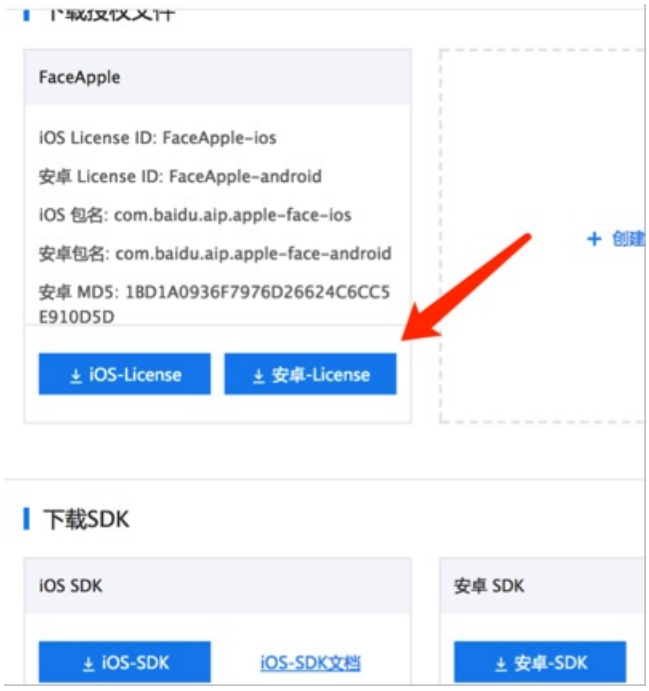
（3）运行示例工程

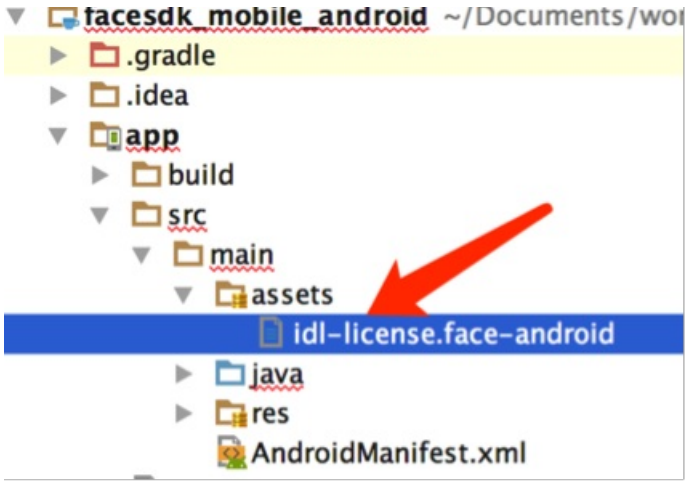


2.3.2 运行未配置授权信息的示例工程

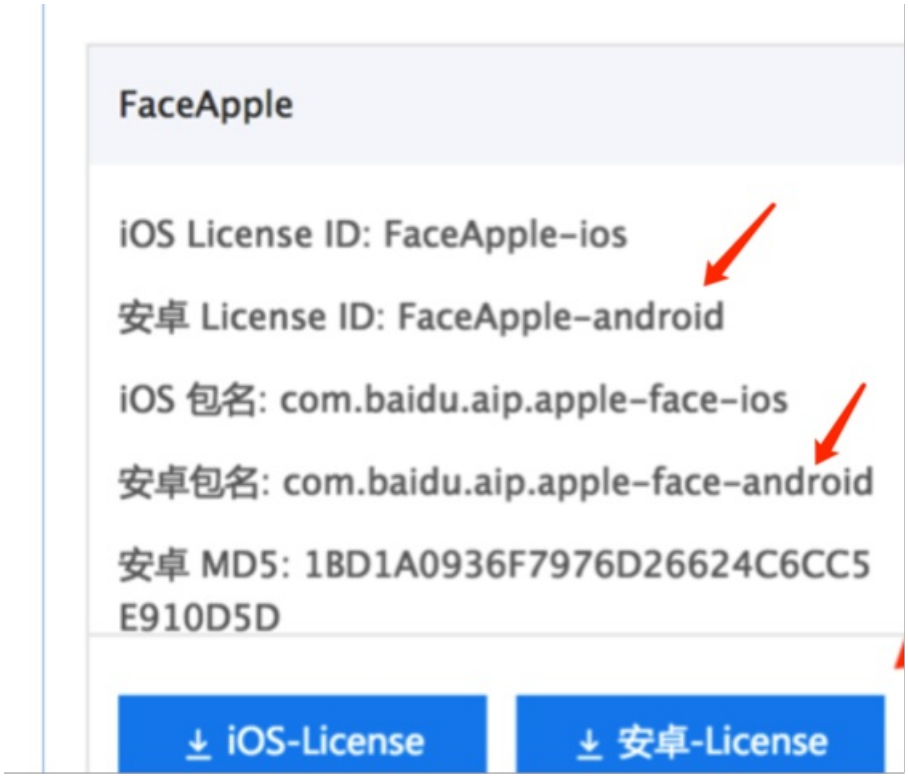
（1）Android Studio导入下载的示例工程

（2）下载license拷贝到工程的assets目录



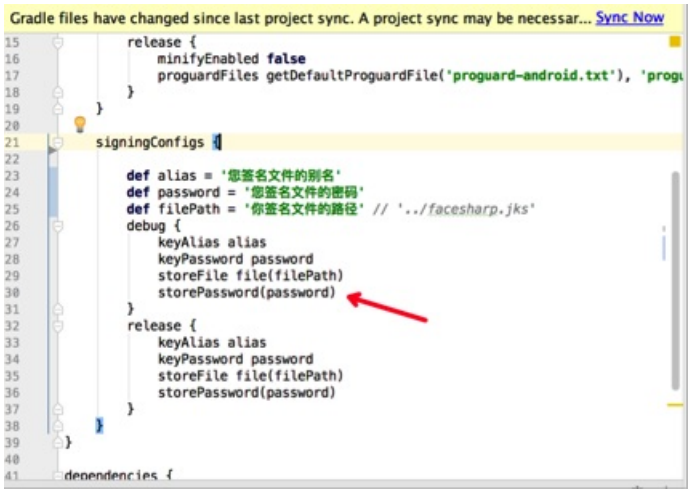


（3）修改Config类



（4）修改app.gradle和AndroidManifest里面的包名为申请license填入的包名，如上图安卓包名。

（5）配置打包签名文件，由于SDK运行时会校验签名文件的MD5是否和申请时填入一样。为了便于debug能正常使用SDK，需要配置打包签名文件。



（6）运行示例工程。如果无法正常体验，请查看logcat日志。是否有**FaceSDK-License LICENSE_INFO_CHECK_ERROR** 日志。如果有说明授权没有成功，可以查看本文档最后的常见问题进行解决。

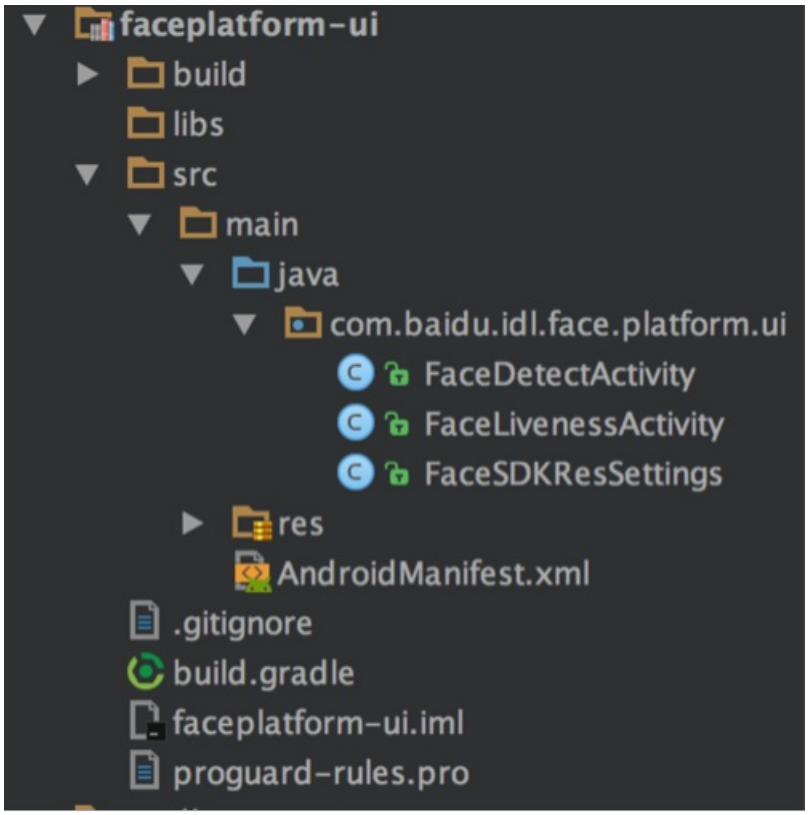
2.4 添加SDK到工程

FaceSdk以androidstudio开发方式提供，以下介绍在android studio开发工具导入FaceSdk

（1）将开发包中的faceplatform-release库Copy 到工程根目录。

（2）将开发包中的faceplatform-ui库Copy 到工程根目录。

（3）SDK提供的了开源的faceplatform-ui库，把活体检测和人脸图像采集功能等功能进行了封装，适配了主流机型机型。如果需要使用，请添加faceplatform-ui模块到的工程中。faceplatform-ui目录结构如下图



（4）在build.gradle使用compile project引人faceplatform-ui库工程。

```
dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    compile "com.android.support:recyclerview-v7:+"
    compile project(path: ':faceplatform-ui')
}
```

（5）Setting.gradle中include faceplatfrom-ui和facepaltfrom-release

```
include ':app', ':faceplatform-release'
include ':faceplatform-ui'
```

（6）从官网下载授权文件license，复制到app/src/main/assets目录下。

（7）申请的license已经和打包签名key进行了绑定（申请时用到了签名的md5，为了便于debug模式也能调用SDK的功能，需要把debug的key改成申请license的key。

a、把key拷贝到项目根目录下

b、主appbuild.gradle android 下面添加(修改)signingConfigs相关的配置。如下图。

2.5 权限声明

名称	用途
android.permission.INTERNET	允许应用联网,SDK联网授权。
android.permission.READ_PHONE_STATE	获取用户手机的 IMEI,用来唯一的标识用户
android.permission.CAMERA	允许调用相机进行拍照
android.hardware.camera.autofocus	允许相机对焦
android.permission.WRITE_EXTERNAL_STORAGE	图片裁剪临时存储

3、功能使用

3.1 人脸检测

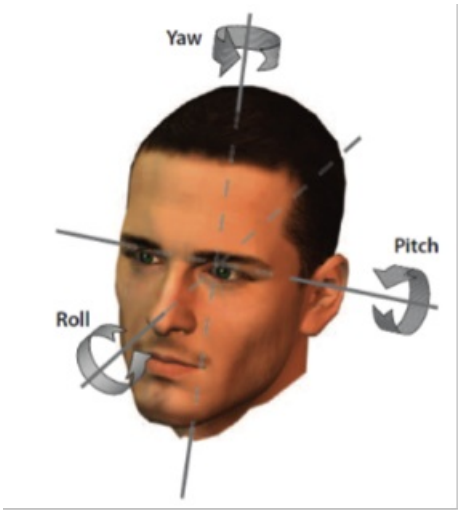
（1） 调用FaceSDKManager.getIntance().initialize(context,Config.licenseID, Config.licenseFileName);

初始化SDK,demo中此代码在Demoapplication中。

(2) 初始化参数设置

```
FaceConfig config = FaceSDKManager.getInstance().getFaceConfig();
// SDK初始化已经设置完默认参数（推荐参数），您也根据实际需求进行数值调整
// 设置活体动作，通过设置list LivenessTypeEnum.Eye, LivenessTypeEnum.Mouth, LivenessTyp
// LivenessTypeEnum.HeadDown, LivenessTypeEnum.HeadLeft, LivenessTypeEnum.HeadRight,
// LivenessTypeEnum.HeadLeftOrRight
config.setLivenessTypeList(ExampleApplication.livenessList);
// 设置 活体动作是否随机 boolean
config.setLivenessRandom(ExampleApplication.isLivenessRandom);
// 模糊度范围 (0-1) 推荐小于0.7
config.setBlurnessValue(FaceEnvironment.VALUE_BLURNESS);
// 光照范围 (0-1) 推荐大于40
config.setBrightnessValue(FaceEnvironment.VALUE_BRIGHTNESS);
// 裁剪人脸大小
config.setCropFaceValue(FaceEnvironment.VALUE_CROP_FACE_SIZE);
// 人脸yaw,pitch,row 角度，范围（-45，45），推荐-15-15
config.setHeadPitchValue(FaceEnvironment.VALUE_HEAD_PITCH);
config.setHeadRollValue(FaceEnvironment.VALUE_HEAD_ROLL);
config.setHeadYawValue(FaceEnvironment.VALUE_HEAD_YAW);
// 最小检测人脸（在图片人脸能够被检测到最小值）80-200， 越小越耗性能，推荐120-200
config.setMinFaceSize(FaceEnvironment.VALUE_MIN_FACE_SIZE);
//
config.setNotFaceValue(FaceEnvironment.VALUE_NOT_FACE_THRESHOLD);
// 人脸遮挡范围 （0-1） 推荐小于0.5
config.setOcclusionValue(FaceEnvironment.VALUE_OCCLUSION);
// 是否进行质量检测
config.setCheckFaceQuality(true);
// 人脸检测使用线程数
config.setFaceDecodeNumberOfThreads(2);
// 是否开启提示音
config.setSound(true);

FaceSDKManager.getInstance().setFaceConfig(config);
```

人脸旋转角度参考下图：



（3） FaceConfig config =FaceSDKManager.*getInstance*().getFaceConfig();

FaceConfig .setSound(boolean)相应的方法设置，提示音&提示语资源。(非必须)

（4）Activity中初始化相机可见示例TrackActivity中的initCamera中，主要用到2个类FaceDetectManager及CameraImageSource等。(该部分代码虽然在facesdk中，但是开源代码，可根据自身需要修改。)代码如：

```
*/
private void initCamera() {
    // 初始化相机图片资源
    final CameraImageSource cameraImageSource = new CameraImageSource( context: this);
    // 设置预览界面
    cameraImageSource.setPreviewView(previewView);

    // 设置人脸检测图片资源
    faceDetectManager.setImageSource(cameraImageSource);
    // 设置人脸检测回调,其中 retCode为人脸检测回调值（0通常为检测到人脸),infos为人脸信息，frame为相机回调图片资源
    faceDetectManager.setOnFaceDetectListener(new FaceDetectManager.OnFaceDetectListener() {
        @Override
        public void onDetectFace(final int retCode, FaceInfo[] infos, ImageFrame frame) {

            if (infos != null && infos[0] != null) {
                showFrame(infos[0]);
            } else {
                showFrame( info: null);
            }
        }
    });

    cameraImageSource.getCameraControl().setPermissionCallback(onRequestPermission() → {
            ActivityCompat.requestPermissions( activity: TrackActivity.this,
                    new String[]{Manifest.permission.CAMERA}, requestCode: 100);
            return true;
    });


    ICameraControl control = cameraImageSource.getCameraControl();
    control.setPreviewView(previewView);
    // 设置检测裁剪处理器
    faceDetectManager.addPreProcessor(cropProcessor);

    // 获取相机屏幕方向
    int orientation = getResources().getConfiguration().orientation;
    mIsPortrait = (orientation == Configuration.ORIENTATION_PORTRAIT);
```

（5）在faceDetectManager的监听回调方法onDetectFace中，可以检测得到人脸信息FaceInfo,retCode为人脸检测值，，ImageFrame为相机返回的预览图片，一般为0的时候表示合格人脸图片，可用这时候的图片进行人脸注册、登录等事宜。亦可如后描述7在onTrack中取图片进行。

（6）sdk自带可适配前置、后置及usb摄像头，可在如下方法中进行调整。

```
/**
 * 摄像头类型设置，可根据自己需求设置前置、后置及usb摄像头
 *
 * @param cameraImageSource
 */
private void setCameraType(CameraImageSource cameraImageSource) {
    // TODO 选择使用前置摄像头
    cameraImageSource.getCameraControl().setCameraFacing(ICameraControl.CAMERA_FACING_FRONT);

    // TODO 选择使用usb摄像头
    //  cameraImageSource.getCameraControl().setCameraFacing(ICameraControl.CAMERA_USB);
    // 如果不设置，人脸框会镜像，显示不准
    //  previewView.getTextureView().setScaleX(-1);

    // TODO 选择使用后置摄像头
    // cameraImageSource.getCameraControl().setCameraFacing(ICameraControl.CAMERA_FACING_BACK);
    // previewView.getTextureView().setScaleX(-1);
}
```

（7）多次调用采集的人脸图片也可保存在本地，如示例demo中得DetectActivity中的faceDetectManager的回调方法onTrack中，可以利用回调参数FaceFilter.TrackedModel model中进行model.cropFace获取回调的图片并进行保存（Bitmap face = model.cropFace();）。

（8） 以上为完成人脸检测采集，若需要调用在线API,可以使用第7部获取的bitmap可用于调用百度人脸云服务（如：注册、识别等。具体参见 https://ai.baidu.com/docs#/Face-API/top）。调用在线服务通常只需要传最佳人脸（bestimage0）

注意：调用在线api功能，为了上传人脸更快，可以把人脸图片压缩到200*200~300*300。如果调用在线活体功能，需要保证上传图片中人脸不小于100px*100px，长宽占图片的三分之一左右。

3.2 质量校验设置

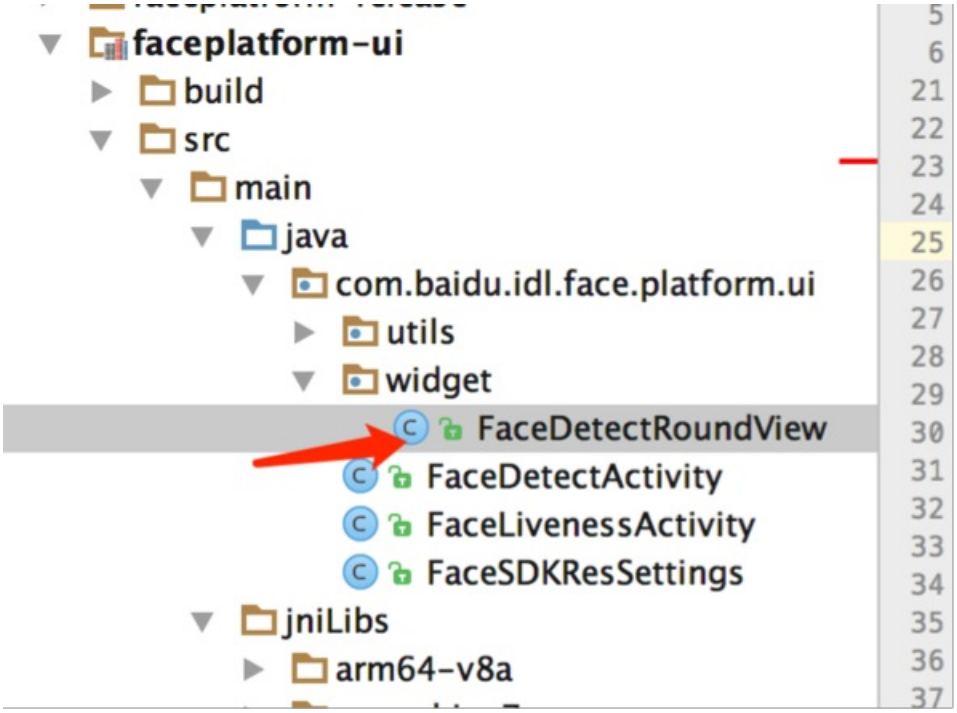
com.baidu.idl.face.platform.FaceConfig类用于人脸检测参数设置。

参数	名称	默认值	取值范围
brightnessValue	图片爆光度	40f	
blurnessValue	图像模糊度	0.5f	0~1.0f
occlusionValue	人脸遮挡阈值	0.5f	0~1.0f
headPitchValue	低头抬头角度	10	0~45
headYawValue	左右角度	10	0~45
headRollValue	偏头角度	10	0~45
cropFaceValue	裁剪图片大小	400	
minFaceSize	最小人脸检测值 小于此值的人脸将检测不出来。最小值为80.	200	
notFaceValue	人脸置信度	0.6f	0~1.0f
isCheckFaceQuality	是否检测人脸质量	true	true/flase

3.3 界面定制说明

3.3.1 修改faceplatform_ui界面

（1）如果SDK自带的UI和您的APP不统一，您可以自行修改FaceDetectRoundView。



（2）修改提示语音音频文件，有两种方式。

a、直接替换FaceUI工程raw下的mp3文件和string.xml。

b、FaceEnvironment 提供了setSoundId(FaceStatusEnum status, int soundId); 设置提示音资源。 FaceStatusEnum为不同的状态。soundId为资源文件所对应的resource id. //TODO 都支持哪些音频格式？ 和 setTipsId(FaceStatusEnumstatus, int tipsId); 设置提示语。

4、接口设计说明

4.1 人脸功能管理器

4.1.1创建实例

- 方法

FaceSDKManager getInstance()

- 参数

无

- 返回

人脸功能管理器

- 说明

创建人脸功能管理器

4.1.2 人脸功能管理器初始化

- 方法

```
public void initialize(final Context context, String licenseID, String licenseFileName)
```

- 参数

context 上下文环境，licenseID 传入申请License时获取的应用名称+_face_android后缀

- 返回

无

- 说明

初始化人脸检测功能。进行人脸检测功能License鉴权验证。

4.1.3 设置人脸功能控制参数

- 方法

```
void setFaceConfig(FaceConfig config)
```

- 参数

config 人脸功能控制参数对象

- 返回

无

- 说明

设置人脸功能控制参数对象。

FaceConfig对象参数：

光照阈值

图像模糊阈值

人脸遮挡阈值

头部姿态角度

最小人脸检测阈值

人脸检测精度阈值

截取人脸图片大小

进行活体检测的动作类型列表

是否进行人脸图片质量检测

4.1.4 取得人脸图像采集功能接口

- 方法

```
IDetectStrategy getDetectStrategyModule()
```

- 参数

无

- 返回

人脸图像采集功能接口

- 说明

取得人脸图像采集功能接口。人脸图像采集接口完成，解析图片人脸信息，返回检测结果。

5、常见问题

（1）**license**文件有什么用，该放在什么地方？

license文件需要申请，目的是作为sdk校验开发者的使用合法性，license文件放置位置不对或未放置license文件会导致没法使用sdk，一般应先申请license文件，并把申请得到的license文件，放置在assets目录下面。

（2）**FaceSDK-LicenseLICENSE_INFO_CHECK_ERROR =license id**

licenseID为您申请时填appname+“_face_android”。如下图demo-turnstile-face-android为license里面的licenseID, **demo-turnstile-face-android1**为app运行时Config.licenseID，两者必须一致

```
E/FaceSDK: FaceSDK-License LICENSE_INFO_CHECK_ERROR =license id  demo-turnstile-face-android  demo-turnstile-face-android1
```

（3）**FaceSDK-LicenseLICENSE_INFO_CHECK_ERROR =signature md5**

md5不一致错误，签名的为license里面的md5，后面的为app运行时获取的签名文件的md5，这两个md5必须一致且区分大小写。 `E/FaceSDK: FaceSDK-License LICENSE_INFO_CHECK_ERROR =signature md5  F5846C60804CC6042D55D09F1AB82364  4357A3EDBC0CA02EA8B5E0578E58D`

（4） **FaceSDK-License LICENSE_INFO_CHECK_ERROR =package name**

PackageName不一致错误。License里面的packagename为申请license时填的，需要保证和app里面的packagename一致。

（5） 在有些机型上出现特别卡或出现无响应？

SDK在**armeabi**上性能非常差，建议删掉其他so只留下armeabi-v7a，包括使用的其他第三方so。因为如果其他so有armeabi，根据android系统查找so的逻辑，在armeabi的机型上只会去该目录下查找so，而人脸SDK没有，就会出现找不到.so文件。

（6） **license** 文件失效了，不能用了怎么办？

license文件申请时候有期限，如过期会导致校验失效，需要在后台申请延期。

安卓-有动作活体版

目录

1 简介
1.1 功能介绍
1.2 兼容性
1.3 开发包说明
2 集成指南
2.1 Sample示例
2.2 准备工作
2.2.1 申请license
2.2.2 下载SDK
2.3 运行示例工程
2.3.1 运行自动配置授权信息的示例工程
2.3.2 运行未配置授权信息的示例工程
2.4添加SDK到工程
2.5权限声明
3 功能使用
3.1 活体识别
3.2人脸采集
3.3质量校验设置
3.4 界面定制说明
3.4.1 修改faceplatform_ui界面
4 接口设计说明
4.1人脸功能管理器
4.1.1 创建实例
4.1.2人脸功能管理器初始化
4.1.3设置人脸功能控制参数
4.1.4取得人脸图像采集功能接口
4.1.5取得活体检测功能接口
4.2人脸图像采集器
4.2.1设置人脸图像采集功能参数
4.2.2人脸图像采集
4.3活体检测器
4.3.1设置人脸功能控制参数
4.3.2活体检测
4.4人脸图像采集界面
4.5 活体检测界面
5 常见问题

1、简介

百度Face SDK Android 版是一种面向 Android 移动设备人脸技术开发包，此版SDK包含人脸检测、活体识别等功能，以aar包+动态链接库的形式发布。基于该方案，开发者可以轻松的构建包含人脸检测、采集和活体识别的应用。在您使用SDK之前，我们首先为您介绍以下SDK本身及相关人脸能力，以便您能更方便使用人脸服务。

1.1 功能介绍

此版SDK所包含的能力如下：

- **本地版活体检测**：通过让用户做出指定人脸配合式的交互动作，识别当前操作者是否为活体，此功能为离线使用，可设定指定动作是否使用及应用顺序。动作包含：眨眼、张嘴、左摇头，右摇头，摇摇头、向上抬头，向下低头七个。可有效抵御高清图片、3D建模、视频等攻击。
- **本地版人脸质量检测**：判断视频流中的图片帧中，哪些图片质量较佳，即人脸图像特征清晰（满足角度、姿态、光照、模糊度等校验）。
- **本地版人脸图像采集**：通过本地SDK能力，采集人脸图像，同时经过人脸质量检测，确保采集到的人脸图像符合各条件校验（角度、姿态、光照、模糊度等），为设备前端获取有效可分析人脸的主要功能。
- **离线授权**：SDK的授权判断，授权介质也称为license，在SDK使用中，需要通过license向授权服务器发起请求，判断SDK的使用合法性及使用有效期。

> 此版SDK**全部功能为离线版本，所有功能均本地化使用**，主要用于在客户端（Android）获取人脸，实际业务使用中，可以按照业务需要，配合在线API完成全流程的业务集成。

为了方便您的开发，我们已经为您准备了多种场景的示例工程，您可以根据业务需要，在后台进行直接下载，目前支持【人脸核身】【人脸闸机/门禁】【人脸登录/考勤】【多人脸识别】,示例工程参考下图：

1.2 兼容性

系统：支持 Android 4.0.3(API Level 15)及以上系统。需要开发者通过 minSdkVersion来保证支持系统的检测。

机型：手机和平板皆可

构架：支持 CPU架构平台【arm-v7，arm-64，x86】

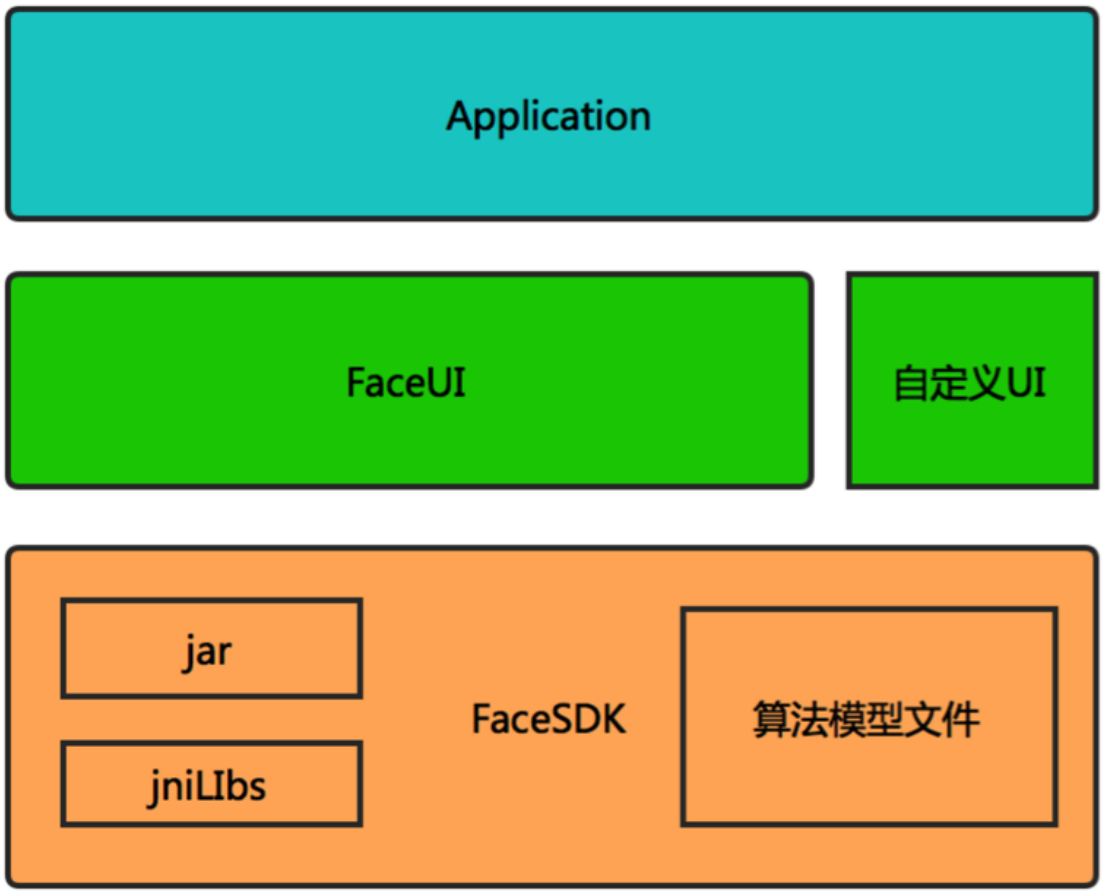
网络：支持 WIFI 及移动网络,移动网络支持使用NET 网关及 WAP 网关(CMWAP、CTWAP、UNIWAP、3GWAP)。

1.3 开发包说明

文件/文件夹名	说明
/faceplatform-release	SDK lib 库相关代码的 aar
/faceplatform-ui	SDK的UI库，封装拍照裁剪等功能,以及各平台的so库。so包含以下几个平台如果关注包大小，请自行删减。/armeabi-v7a/arm64-v8a/x86，
FacePlatform/	DEMO工程

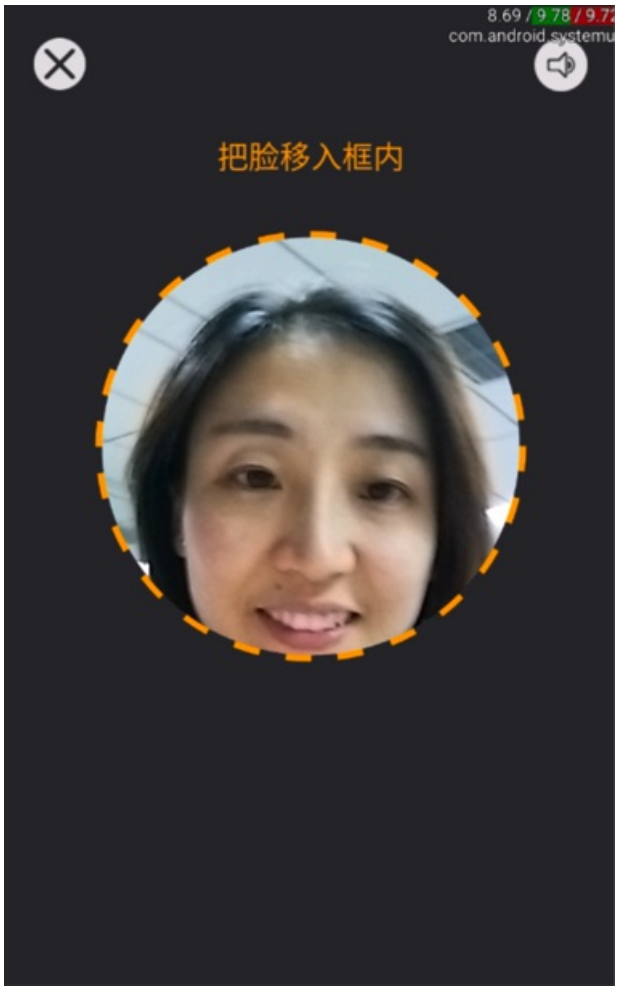
2、集成指南

本章将进行 Step-By-Step的讲解,如何快速的集成 人脸Sdk到现有应用中。一个完整的Demo 请参考开发包中的示例程序 FacePlatform。方案架构参考下图：

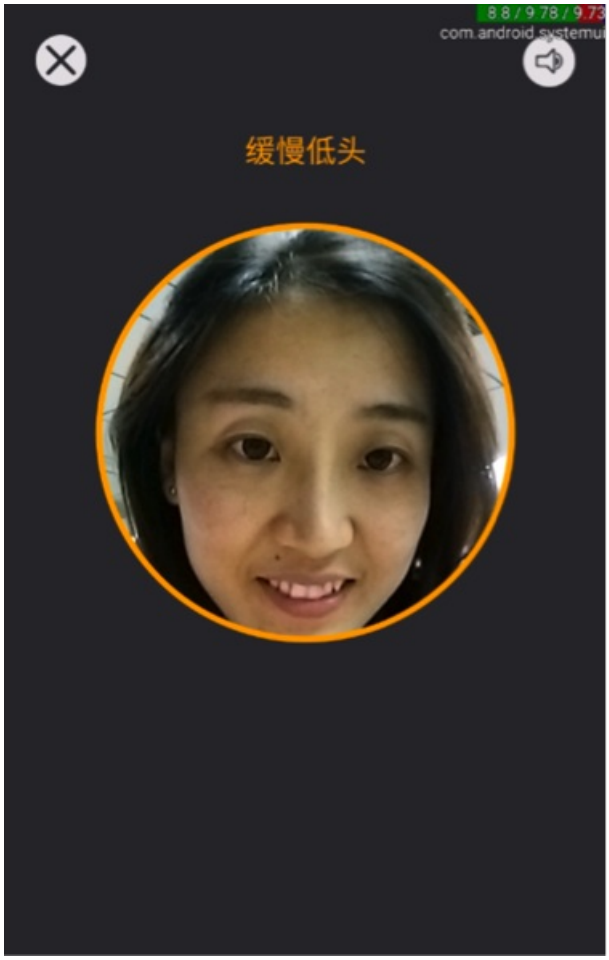


2.1 Sample示例

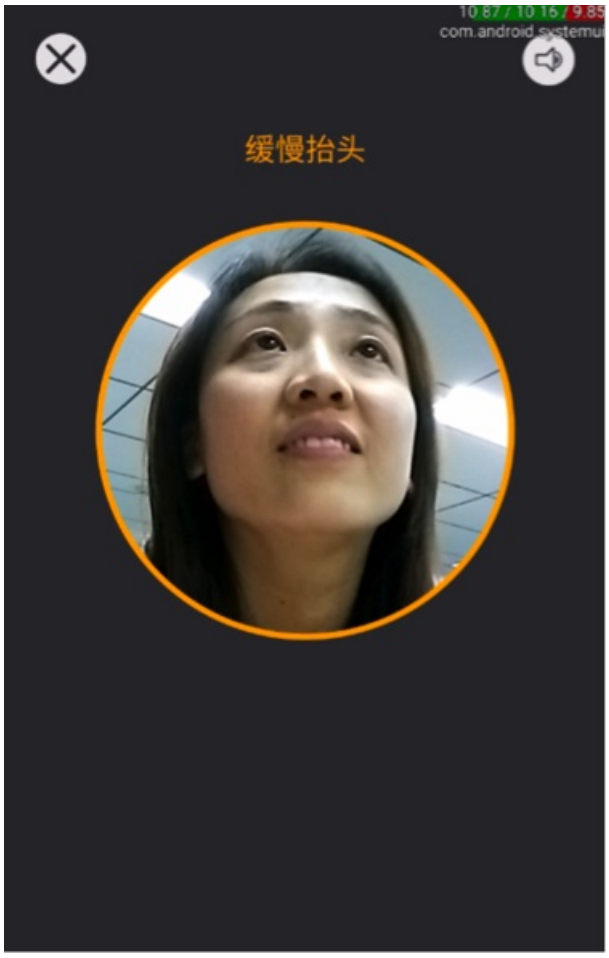
- 把脸移入框内



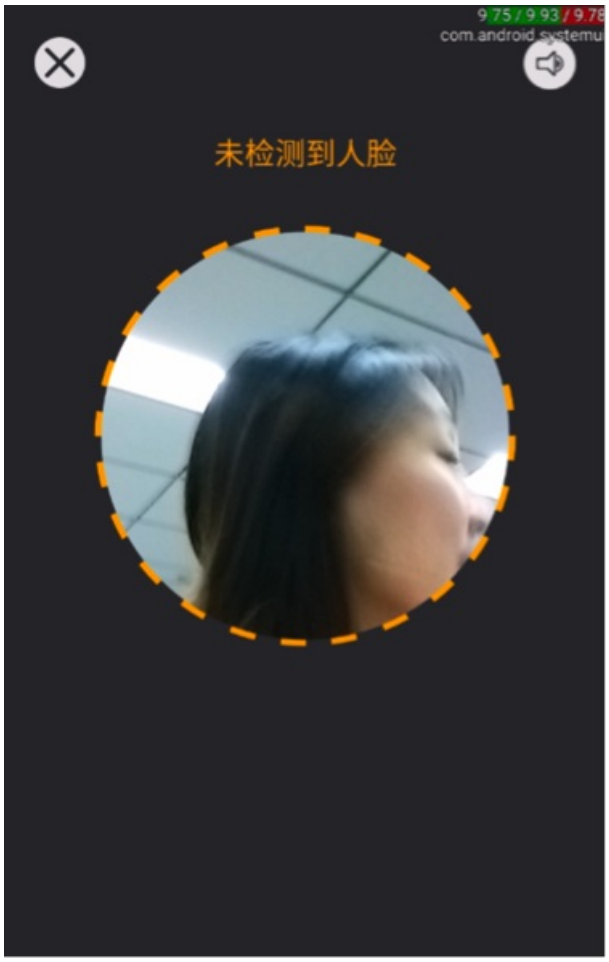
- 慢慢低头



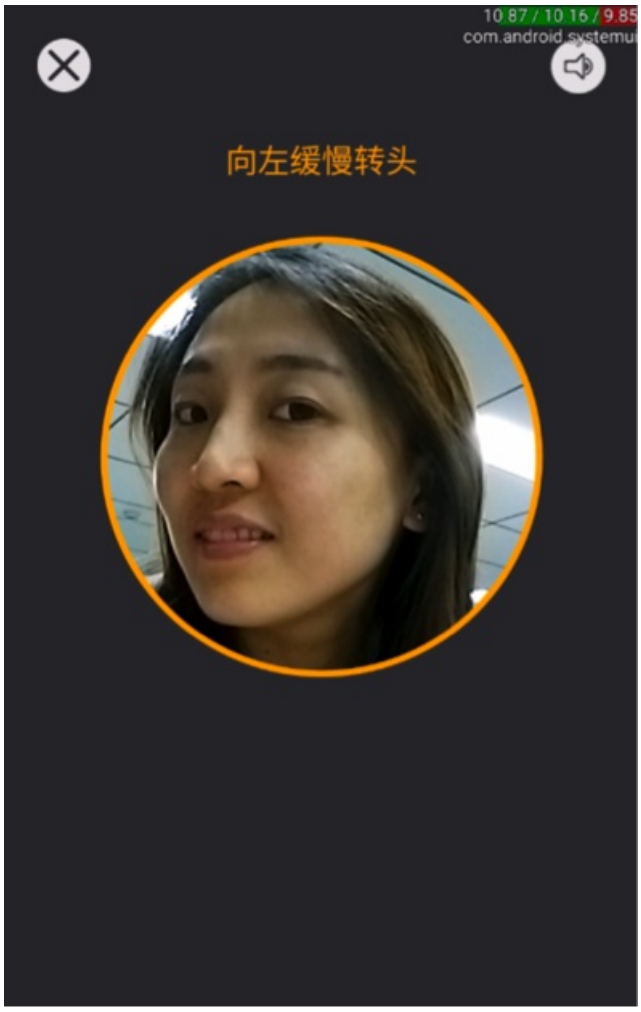
- 慢慢抬头



- 未检测到人脸



- 向左缓慢摇头

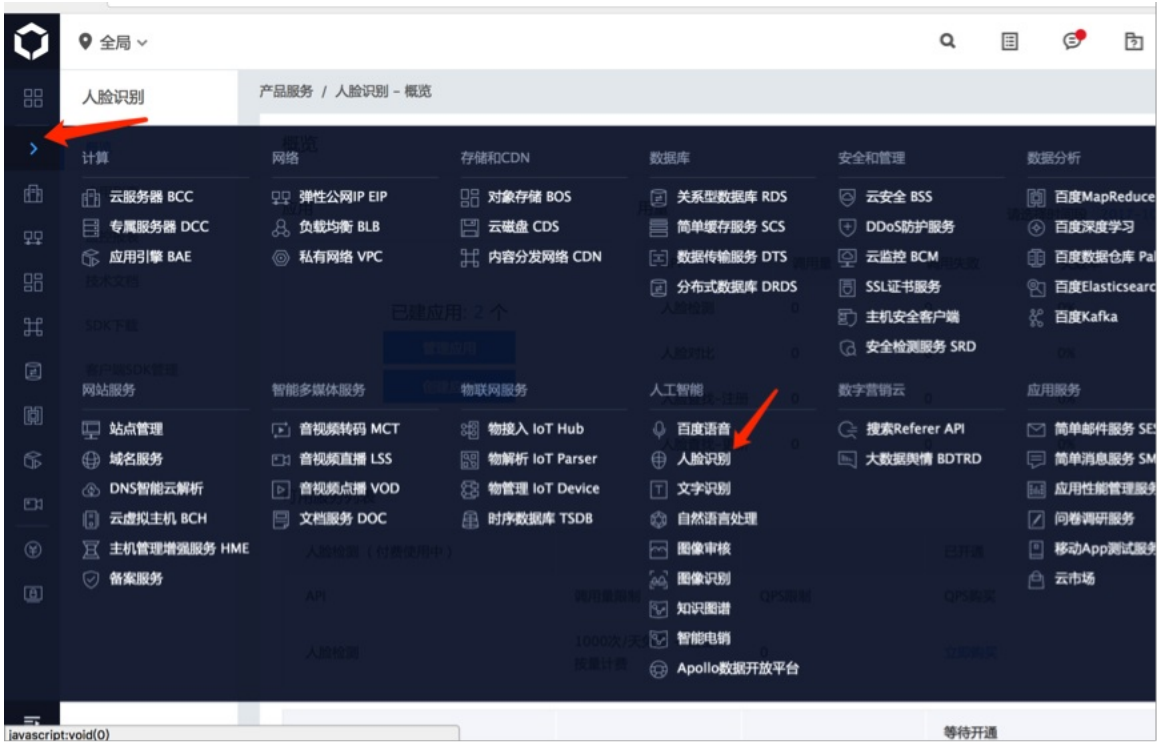


2.2 准备工作

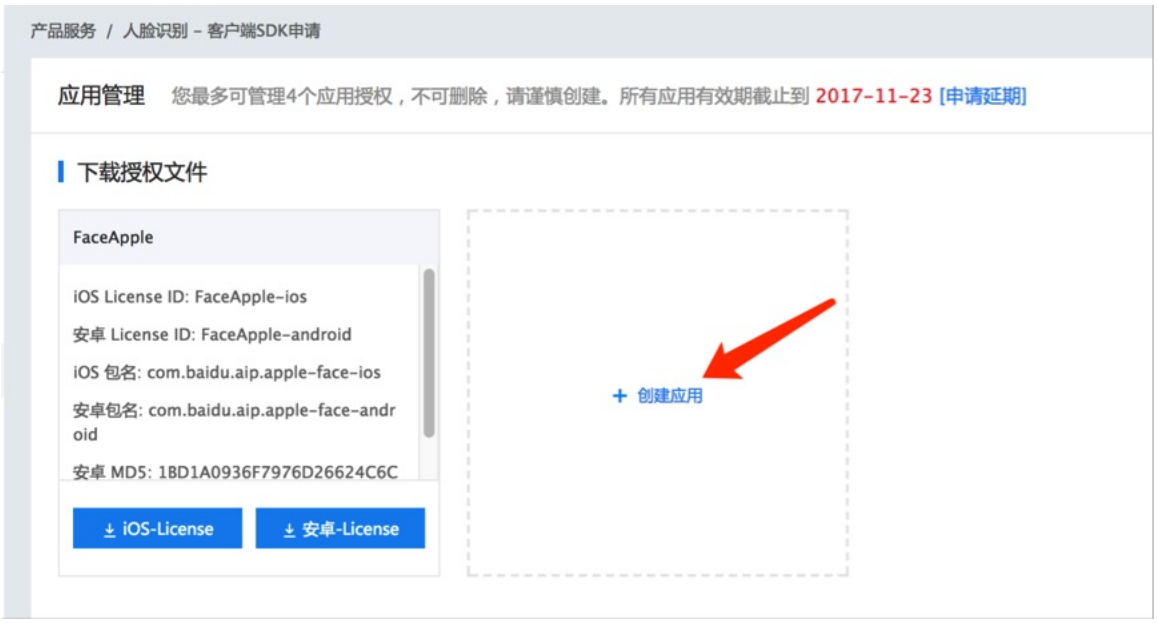
2.2.1 申请license

人脸SDK License：此license用于SDK离线功能使用，在您的申请人脸SDK的后台页面，全局->产品服务->人脸识别->客户端SDK申请

人脸控制台路径如下：

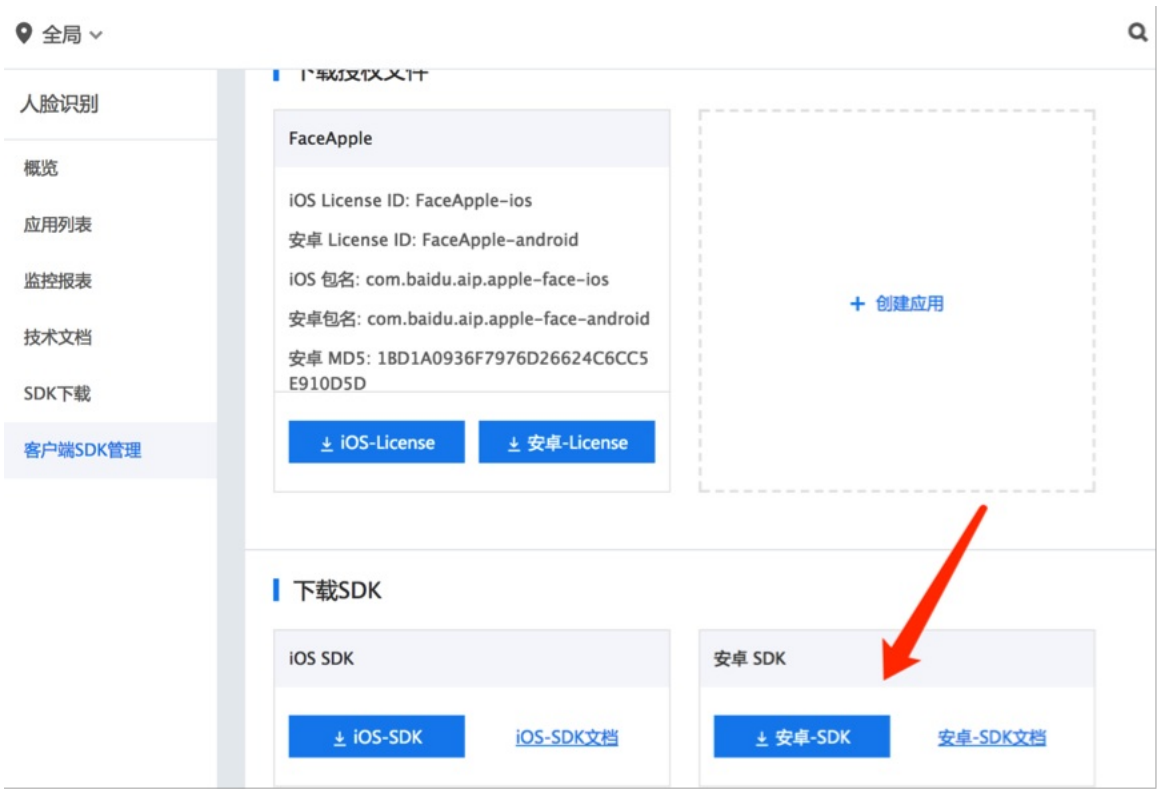


点击客户端SDK管理，弹出如下图：创建应用（这里创建应用是为了使用离线SDK，上面创建应用为了使用人脸在线接口，如注册、识别等）

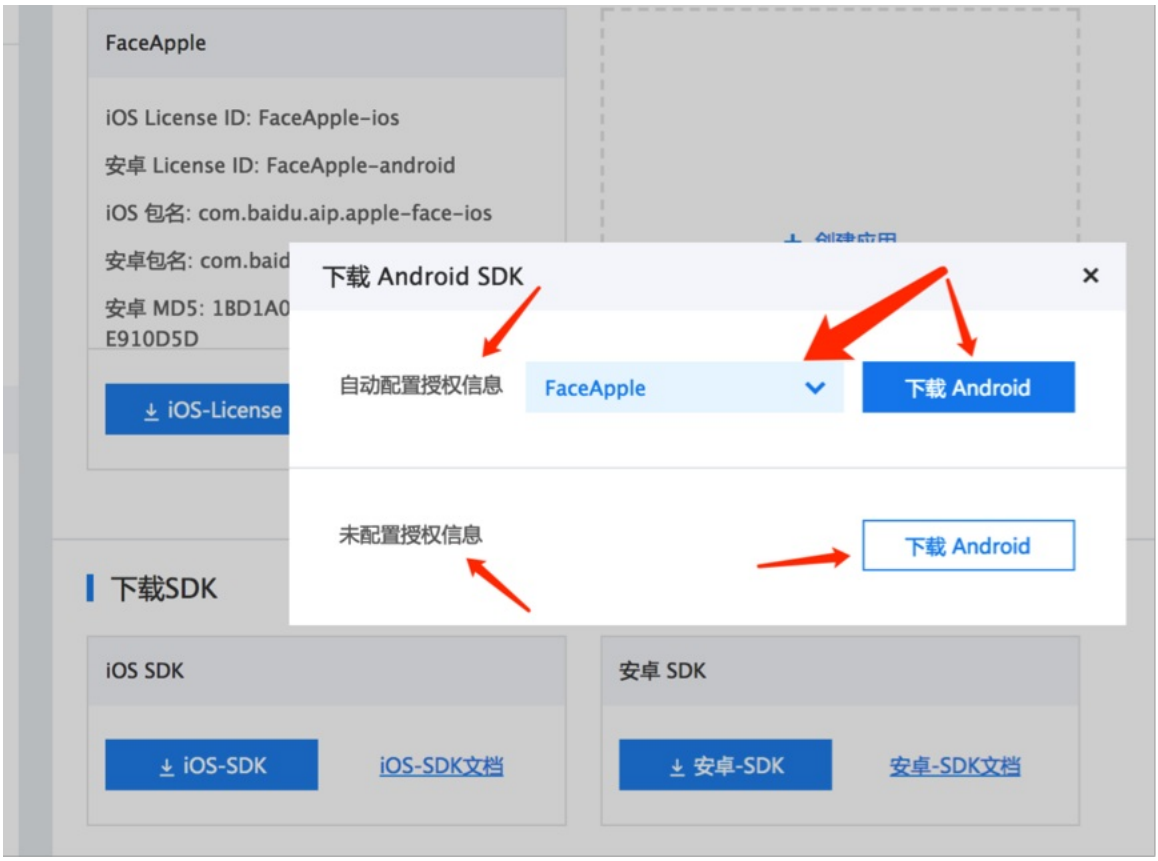


在弹出的框中输入授权标识，选择应用类型，应用系统，以及包名、MD5签名，详情请查看输入框右边提示

2.2.2 下载SDK



下载SDK分为自动配置授权信息（创建license后就可以选择为该应用，下载后SDK自动帮您配置授权，不用下载license拷贝到工程中，初始化参数licenseID,包名也帮您配置好了）和未配置授权信息两种方式：

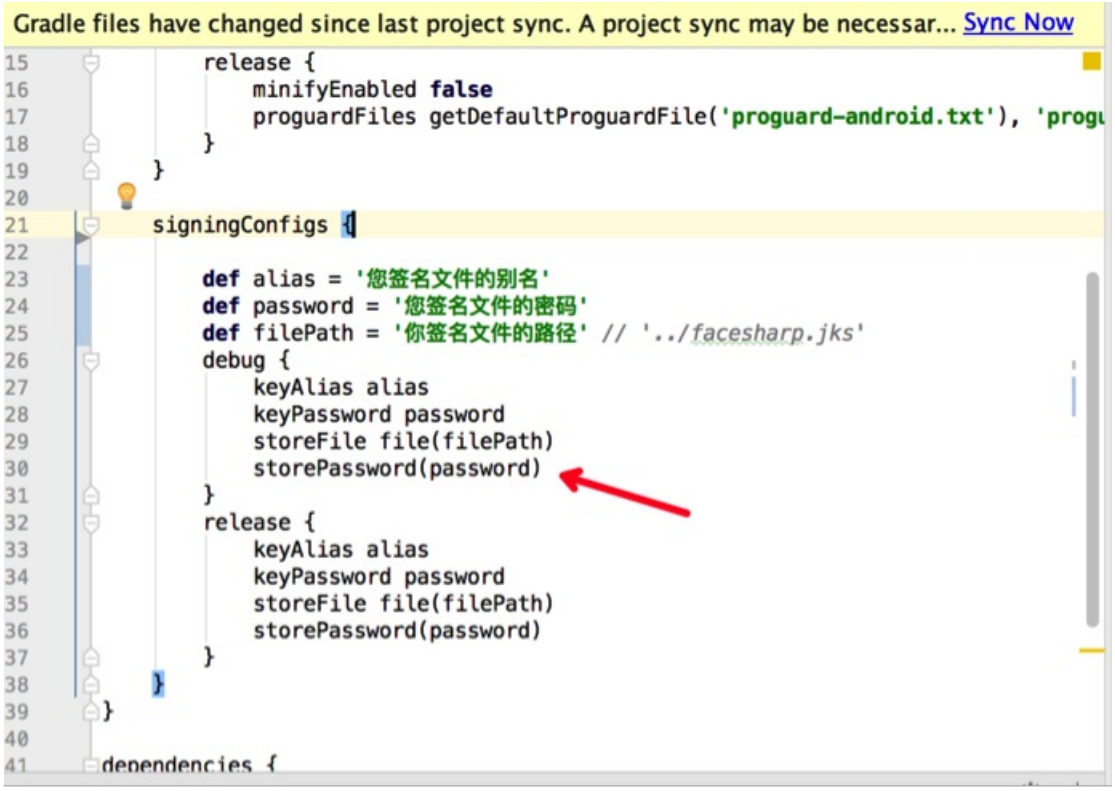


2.3 运行示例工程

2.3.1 运行自动配置授权信息的示例工程

该下载的示例工程，已经帮你改好了license和包名

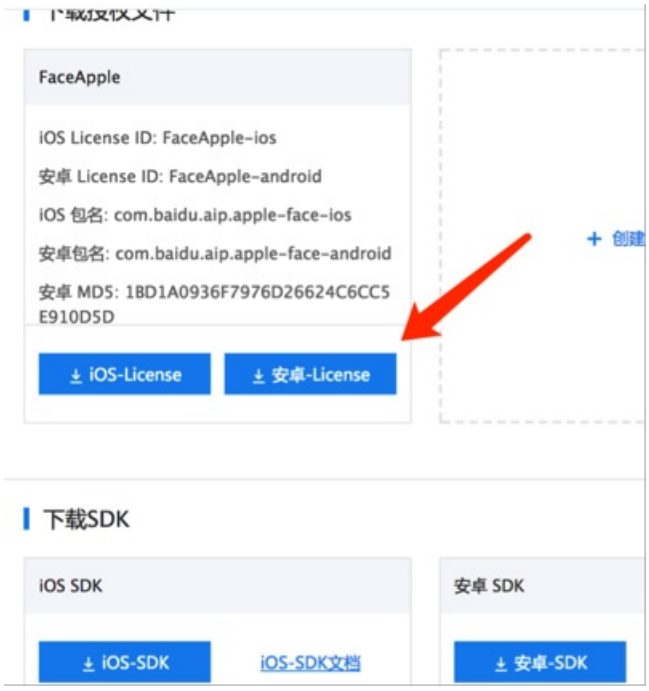
- Android Studio导入下载的示例工程
- 配置打包签名文件，由于SDK运行时会校验签名文件的MD5是否和申请时填入一样。为了便于debug能正常使用SDK，需要配置打包签名文件。
- 运行示例工程

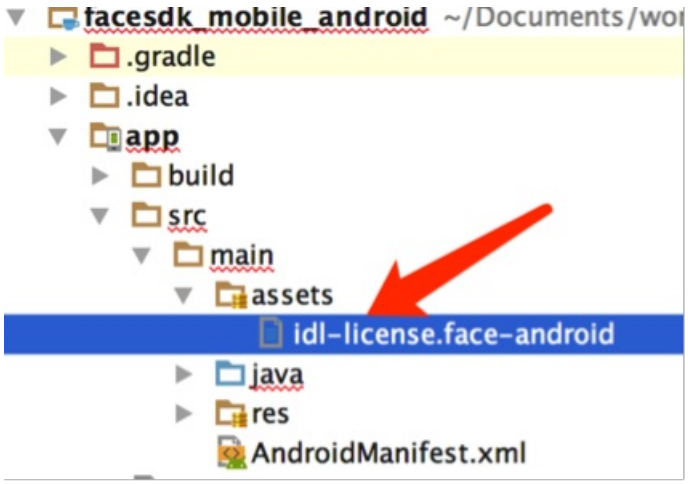


2.3.2 运行未配置授权信息的示例工程

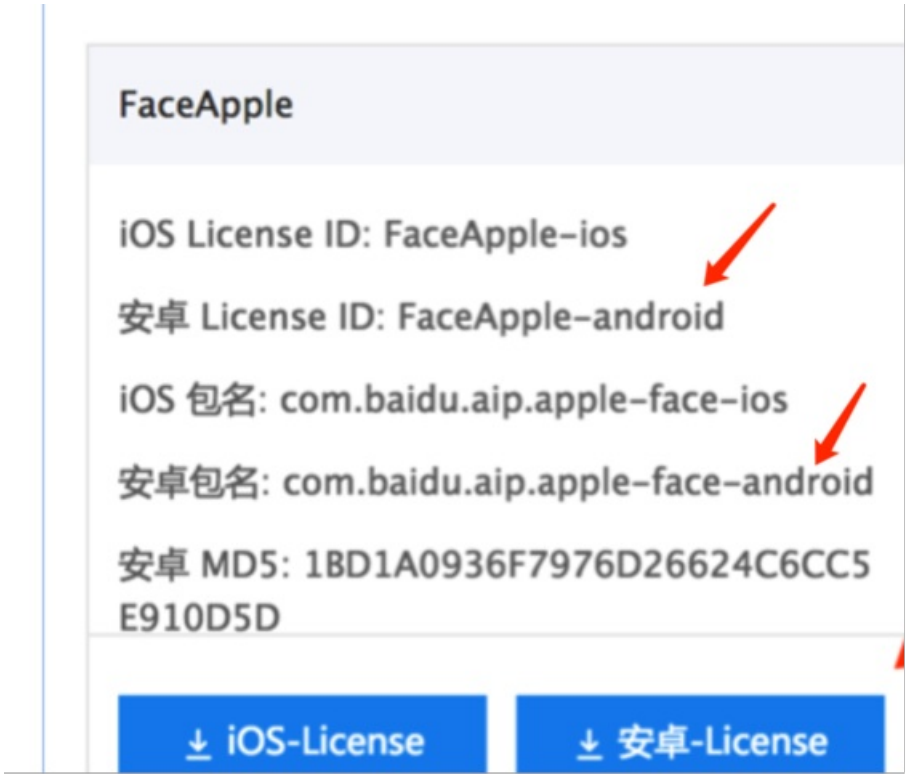
（1）Android Studio导入下载的示例工程

（2）下载license拷贝到工程的assets目录



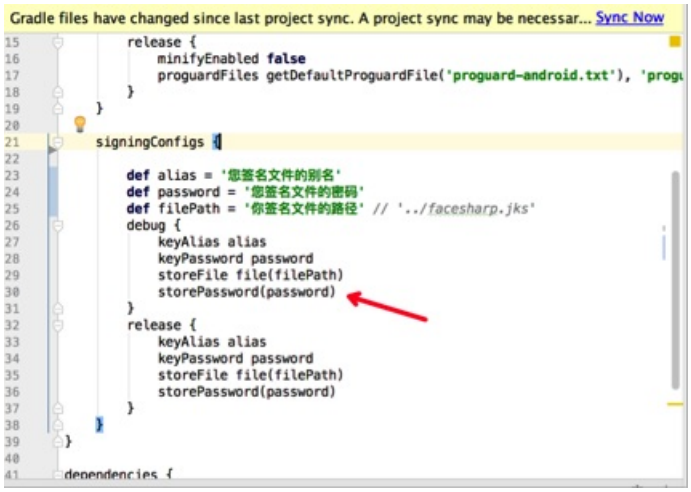


（3）修改Config类



（4）修改app.gradle和AndroidManifest里面的包名为申请license填入的包名，如上图安卓包名。

（5）配置打包签名文件，由于SDK运行时会校验签名文件的MD5是否和申请时填入一样。为了便于debug能正常使用SDK，需要配置打包签名文件。



（6）运行示例工程。如果无法正常体验，请查看logcat日志。是否有 **FaceSDK-License LICENSE_INFO_CHECK_ERROR**日志。如果有说明授权没有成功，可以查看本文档最后的常见问题进行解决。

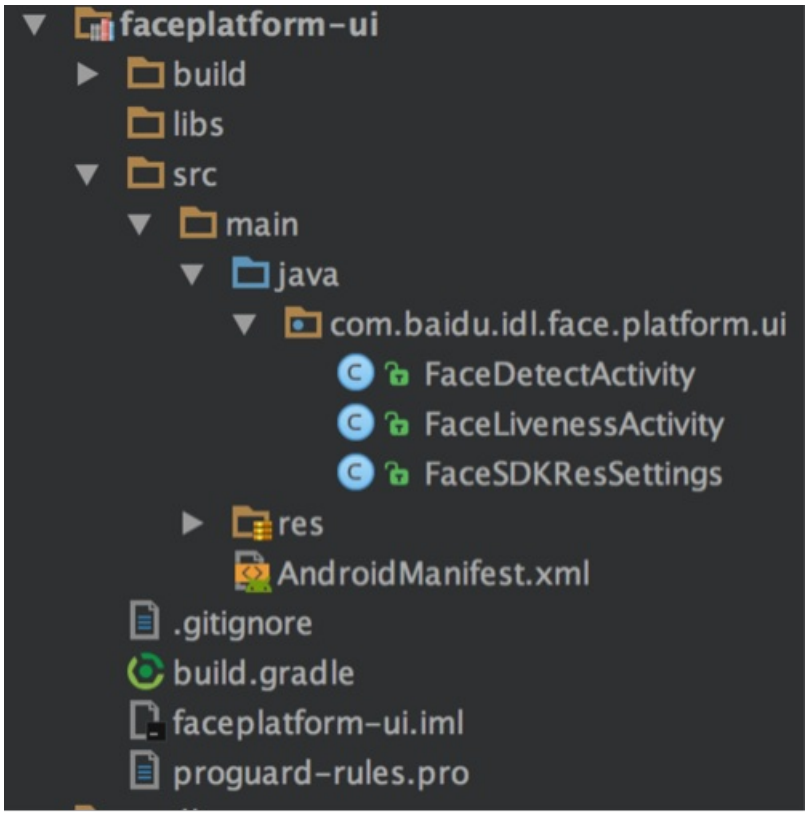
2.4 添加SDK到工程

FaceSdk以androidstudio开发方式提供，以下介绍在android studio开发工具导入FaceSdk

（1）将开发包中的faceplatform-release库Copy 到工程根目录。

（2）将开发包中的faceplatform-ui库Copy 到工程根目录。

（3）SDK提供的了开源的faceplatform-ui库，把活体检测和人脸图像采集功能等功能进行了封装，适配了主流机型机型。如果需要使用，请添加faceplatform-ui模块到的工程中。faceplatform-ui目录结构如下图



（4）在build.gradle使用compile project引人faceplatform-ui库工程。

```
dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    compile "com.android.support:recyclerview-v7:+"
    compile project(path: ':faceplatform-ui')
}
```

（5）Setting.gradle中include faceplatfrom-ui和facepaltfrom-release

```
include ':app', ':faceplatform-release'
include ':faceplatform-ui'
```

（6）从官网下载授权文件license，复制到app/src/main/assets目录下。

（7）申请的license已经和打包签名key进行了绑定（申请时用到了签名的md5，为了便于debug模式也能调用SDK的功能，需要把debug的key改成申请license的key。

- 把key拷贝到项目根目录下
- 主appbuild.gradle android 下面添加(修改)signingConfigs相关的配置。如下图。

2.5 权限声明

名称	用途
android.permission.INTERNET	允许应用联网,SDK联网授权。
android.permission.READ_PHONE_STATE	获取用户手机的 IMEI,用来唯一的标识用户
android.permission.CAMERA	允许调用相机进行拍照
android.hardware.camera.autofocus	允许相机对焦
android.permission.WRITE_EXTERNAL_STORAGE	图片裁剪临时存储

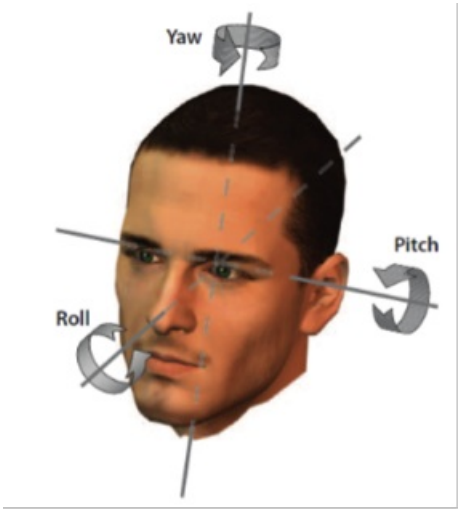
3、功能使用

3.1 活体识别

（1） 调用FaceSDKManager.getIntance().initialize(context,Config.licenseID, Config.licenseFileName);初始化SDK。**Demo中此段代码在MainActivity中。**

（2） 初始化参数设置

```
FaceConfig config = FaceSDKManager.getInstance().getFaceConfig();
// SDK初始化已经设置完默认参数（推荐参数），您也根据实际需求进行数值调整
// 设置活体动作，通过设置list LivenessTypeEnum.Eye, LivenessTypeEnum.Mouth, LivenessTyp
// LivenessTypeEnum.HeadDown, LivenessTypeEnum.HeadLeft, LivenessTypeEnum.HeadRight,
// LivenessTypeEnum.HeadLeftOrRight
config.setLivenessTypeList(ExampleApplication.livenessList);
// 设置 活体动作是否随机 boolean
config.setLivenessRandom(ExampleApplication.isLivenessRandom);
// 模糊度范围 (0-1) 推荐小于0.7
config.setBlurnessValue(FaceEnvironment.VALUE_BLURNESS);
// 光照范围 (0-1) 推荐大于40
config.setBrightnessValue(FaceEnvironment.VALUE_BRIGHTNESS);
// 裁剪人脸大小
config.setCropFaceValue(FaceEnvironment.VALUE_CROP_FACE_SIZE);
// 人脸yaw,pitch,row 角度，范围（-45，45），推荐-15-15
config.setHeadPitchValue(FaceEnvironment.VALUE_HEAD_PITCH);
config.setHeadRollValue(FaceEnvironment.VALUE_HEAD_ROLL);
config.setHeadYawValue(FaceEnvironment.VALUE_HEAD_YAW);
// 最小检测人脸（在图片人脸能够被检测到最小值）80-200， 越小越耗性能，推荐120-200
config.setMinFaceSize(FaceEnvironment.VALUE_MIN_FACE_SIZE);
//
config.setNotFaceValue(FaceEnvironment.VALUE_NOT_FACE_THRESHOLD);
// 人脸遮挡范围 （0-1） 推荐小于0.5
config.setOcclusionValue(FaceEnvironment.VALUE_OCCLUSION);
// 是否进行质量检测
config.setCheckFaceQuality(true);
// 人脸检测使用线程数
config.setFaceDecodeNumberOfThreads(2);
// 是否开启提示音
config.setSound(true);

FaceSDKManager.getInstance().setFaceConfig(config);
```



（3） FaceConfig config =FaceSDKManager.*getInstance*().getFaceConfig();

FaceConfig .setSound(boolean)相应的方法设置，提示音&提示语资源。(非必须)

（4） startActivity(new Intent(this, FaceLivenessExpActivity.class))，开启预览。

（5）调用FaceSDKManager.getInstance().getLivenessStrategyModule()获得ILivenessStrategy对象。(该方法每次调用都会返回一个新对象).

（6）调用ILivenessStrategy.setPreviewDegree();设置预览图片的旋转角度。调用setDetectStrategySoundEnable设置是否开启语音。调用setDetectStrategyConfig设置，预览图的大小，人脸检测框的坐标和回调。

（7）多次调用livenessStrategy进行人脸图片采集，人脸跟踪。

（8）实现ILivenessStrategyCallback的onLivenessCompletion并处理结果。**其中base64ImageMap为存放最佳人脸和每个活体动作的图片。**可以查看起父类FaceLivenessActivity的saveImage和base64ToImage方法，获取对于的bitamap。

```
public class FaceLivenessExpActivity extends FaceLivenessActivity {

    private DefaultDialog mDefaultDialog;

    @Override
    public void onCreate(Bundle savedInstanceState) { super.onCreate(savedInstanceState); }

    @Override
    public void onLivenessCompletion(FaceStatusEnum status, String message, HashMap<String, String> base64ImageMap) {
        super.onLivenessCompletion(status, message, base64ImageMap);
        if (status == FaceStatusEnum.OK && mIsCompletion) {
            showMessageDialog("活体检测", "检测成功");
        } else if (status == FaceStatusEnum.Error_DetectTimeout ||
                status == FaceStatusEnum.Error_LivenessTimeout ||
                status == FaceStatusEnum.Error_Timeout) {
            showMessageDialog("活体检测", "采集超时");
        }
    }
```

（9）以为完成本地活体验证，如需要调用在线服务，可以使用第8部获取的bitmap可用于调用百度人脸云服务（如：注册、识别等。具体参见 https://ai.baidu.com/docs#/Face-API/top），调用在线服务通常只需要传最佳人脸（bestimage0）

3.2 人脸采集

（1）调用FaceSDKManager.getIntance().initialize(context,Config.licenseID, Config.licenseFileName);初始化SDK.。

（2）初始化SDK参数

```
FaceConfig config = FaceSDKManager.getInstance().getFaceConfig();
// SDK初始化已经设置完默认参数（推荐参数），您也根据实际需求进行数值调整
// 设置活体动作，通过设置list LivenessTypeEnum.Eye, LivenessTypeEnum.Mouth, LivenessTyp
// LivenessTypeEnum.HeadDown, LivenessTypeEnum.HeadLeft, LivenessTypeEnum.HeadRight,
// LivenessTypeEnum.HeadLeftOrRight
config.setLivenessTypeList(ExampleApplication.livenessList);
// 设置 活体动作是否随机 boolean
config.setLivenessRandom(ExampleApplication.isLivenessRandom);
// 模糊度范围 (0-1) 推荐小于0.7
config.setBlurnessValue(FaceEnvironment.VALUE_BLURNESS);
// 光照范围 (0-1) 推荐大于40
config.setBrightnessValue(FaceEnvironment.VALUE_BRIGHTNESS);
// 裁剪人脸大小
config.setCropFaceValue(FaceEnvironment.VALUE_CROP_FACE_SIZE);
// 人脸yaw,pitch,row 角度，范围（-45，45），推荐-15-15
config.setHeadPitchValue(FaceEnvironment.VALUE_HEAD_PITCH);
config.setHeadRollValue(FaceEnvironment.VALUE_HEAD_ROLL);
config.setHeadYawValue(FaceEnvironment.VALUE_HEAD_YAW);
// 最小检测人脸（在图片人脸能够被检测到最小值）80-200， 越小越耗性能，推荐120-200
config.setMinFaceSize(FaceEnvironment.VALUE_MIN_FACE_SIZE);
//
config.setNotFaceValue(FaceEnvironment.VALUE_NOT_FACE_THRESHOLD);
// 人脸遮挡范围 （0-1） 推荐小于0.5
config.setOcclusionValue(FaceEnvironment.VALUE_OCCLUSION);
// 是否进行质量检测
config.setCheckFaceQuality(true);
// 人脸检测使用线程数
config.setFaceDecodeNumberOfThreads(2);
// 是否开启提示音
config.setSound(true);

FaceSDKManager.getInstance().setFaceConfig(config);
```

（3）FaceConfig config =FaceSDKManager.*getInstance*().getFaceConfig();

FaceConfig .setSound(boolean)相应的方法设置，提示音&提示语资源。(非必须)

（4）startActivity(new Intent(this, FaceLivenessExpActivity.class))，开启预览。

（5）调用FaceSDKManager.getInstance().getDetectStrategyModule()获得IDetectStrategy对象。(该方法每次调用都会返回一个新对象).

（6）调用IDetectStrategy.setPreviewDegree();设置预览图片的旋转角度。调用setDetectStrategySoundEnable设置是否开启语音。调用setDetectStrategyConfig设置，预览图的大小，人脸检测框的坐标和回调。

（7） 多次调用detectStrategy进行人脸图片采集。

（8） 实现IDetectStrategyCallback的onDetectCompletion并处理结果。其中base64ImageMap为存放最佳人脸和每个活体动作的图片。可以查看起父类FaceLivenessActivity的saveImage和base64ToImage方法，获取对于的bitamap。

```
public class FaceDetectExpActivity extends FaceDetectActivity {

    private DefaultDialog mDefaultDialog;

    @Override
    public void onCreate(Bundle savedInstanceState) { super.onCreate(savedInstanceState); }

    @Override
    public void onDetectCompletion(FaceStatusEnum status, String message, HashMap<String, String> base64ImageMap) {
        super.onDetectCompletion(status, message, base64ImageMap);
        if (status == FaceStatusEnum.OK && mIsCompletion) {
            showMessageDialog("人脸图像采集", "采集成功");
        } else if (status == FaceStatusEnum.Error_DetectTimeout ||
                status == FaceStatusEnum.Error_LivenessTimeout ||
                status == FaceStatusEnum.Error_Timeout) {
            showMessageDialog("人脸图像采集", "采集超时");
        }
    }

    private void showMessageDialog(String title, String message) {
```

（9） 以上为完成人脸采集，若需要调用在线API,可以使用第8部获取的bitmap可用于调用百度人脸云服务（如：注册、识别等。具体参见 https://ai.baidu.com/docs#/Face-API/top）。调用在线服务通常只需要传最佳人脸（bestimage0）

注意：调用在线api功能，为了上传人脸更快，可以把人脸图片压缩到 200 *200~300* 300。如果调用在线活体功能，需要保证上传图片中人脸不小于 100px * 100px，长宽占图片的三分之一左右

3.3 质量校验设置

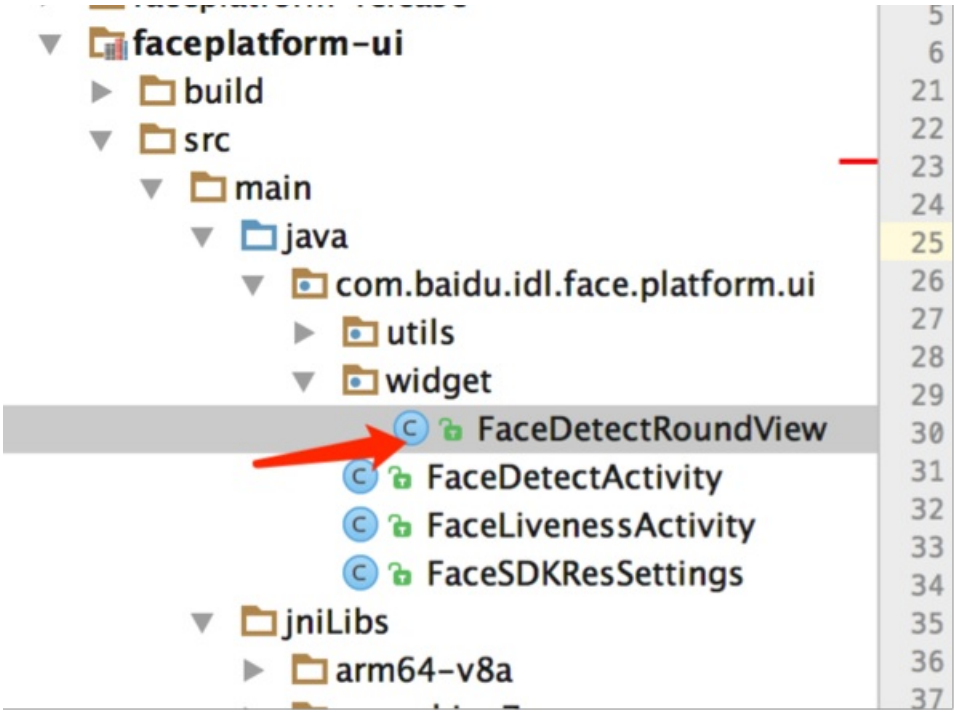
com.baidu.idl.face.platform.FaceConfig类用于人脸检测参数设置。

参数	名称	默认值	公安验证推荐值	取值范围
brightnessValue	图片爆光度	40f	100f	
blurnessValue	图像模糊度	0.5f	0.35f	0~1.0f
occlusionValue	人脸遮挡阈值	0.5f	0.3f	0~1.0f
headPitchValue	低头抬头角度	10	10	0~45
headYawValue	左右角度	10	10	0~45
headRollValue	偏头角度	10	10	0~45
cropFaceValue	裁剪图片大小	400	400	
minFaceSize	最小人脸检测值 小于此值的人脸将检测不出来。最小值为80.	200	200	
notFaceValue	人脸置信度	0.6f	0.6f	0~1.0f
isCheckFaceQuality	是否检测人脸质量	true	true	true/flase

3.4 界面定制说明

3.4.1 修改faceplatform_ui界面

（1）如果SDK自带的UI和您的APP不统一，您可以自行修改FaceDetectRoundView。



（2）修改提示语音音频文件，有两种方式。

a、直接替换FaceUI工程raw下的mp3文件和string.xml。

b、FaceEnvironment 提供了setSoundId(FaceStatusEnum status, int soundId); 设置提示音资源。 FaceStatusEnum为不同的状态。soundId为资源文件所对应的resource id. //TODO 都支持哪些音频格式？ 和 setTipsId(FaceStatusEnumstatus, int tipsId); 设置提示语。

4、接口设计说明#

4.1 人脸功能管理器

4.1.1创建实例

- 方法

FaceSDKManager getInstance()

- 参数

无

- 返回

人脸功能管理器

- 说明

创建人脸功能管理器

4.1.2 人脸功能管理器初始化

- 方法

```
public void initialize(final Context context, String licenseID, String licenseFileName)
```

- 参数

context 上下文环境，licenseID 传入申请License时获取的应用名称+_face_android后缀

- 返回

无

- 说明

初始化人脸检测功能。进行人脸检测功能License鉴权验证。

4.1.3 设置人脸功能控制参数

- 方法

```
void setFaceConfig(FaceConfig config)
```

- 参数

config 人脸功能控制参数对象

- 返回

无

- 说明

设置人脸功能控制参数对象。

FaceConfig对象参数：

光照阈值

图像模糊阈值

人脸遮挡阈值

头部姿态角度

最小人脸检测阈值

人脸检测精度阈值

截取人脸图片大小

进行活体检测的动作类型列表

是否进行人脸图片质量检测

4.1.4 取得人脸图像采集功能接口

- 方法

```
IDetectStrategy getDetectStrategyModule()
```

- 参数

无

- 返回

人脸图像采集功能接口

- 说明

取得人脸图像采集功能接口。人脸图像采集接口完成，解析图片人脸信息，返回检测结果。

4.1.5 取得活体检测功能接口

- 方法

```
ILivenessStrategy getLivenessStrategyModule()
```

- 参数

无

- 返回

活体检测功能接口

- 说明

取得活体检测功能接口。活体检测功能接口完成，解析图片人脸信息，返回活体检测结果。

4.2 人脸图像采集器

人脸图像采集器IDetectStrategy，检测图片中人脸信息，返回人脸检测状态，完成人脸图像采集。

4.2.1 设置人脸图像采集功能参数

- 方法

```
voidsetDetectStrategyConfig(Rect previewRect, Rect detectRect,IDetectStrategyCallback callback)
```

- 参数

previewRect 人脸图片大小，类型：Rect

detectRect 人脸检测区域大小，类型：Rect

callback 人脸图像采集功能状态监听器

- 返回

无

- 说明

设置人脸功能控制参数对象。

4.2.2 人脸图像采集

- 方法

```
void detectStrategy(byte[] imageData)
```

- 参数

imageData 图片信息

- 返回

无

- 说明

检测图片中的人脸信息，完成人脸图像采集，返回检测状态和结果。

4.3 活体检测器

活体检测器LivenessStrategy，检测图片人脸信息，活体检测结果状态。

4.3.1 设置人脸功能控制参数

- 方法

```
voidsetLivenessStrategyConfig(
        List<LivenessTypeEnum> livenessList,
        Rect previewRect,
        Rect detectRect,
        ILivenessStrategyCallback callback);
```

- 参数

livenessList 活体动作列表

previewRect 人脸图片大小，类型：Rect

detectRect 人脸检测区域大小，类型：Rect

callback 人脸图像采集功能状态监听器

- 返回

无

- 说明

设置活体检测功能参数对象。

4.3.2 活体检测

- 方法

void livenessStrategy(byte[] imageData);

- 参数

imageData 图片信息

- 返回

无

- 说明

检测图片中的人脸信息，完成人脸图像采集和人脸活体检测，返回检测状态和结果。

4.4 人脸图像采集界面

人脸图像采集器界面FaceDetectActivity，包括UI界面，系统相机控制，使用人脸图像采集器IDetectStrategy处理相机采集到的图像。

4.5 活体检测界面

活体检测界面FaceLivenessActivity，包括UI界面，系统相机控制，使用活体检测器ILivenessStrategy处理相机采集到的图像，完成活体检测功能。

5、常见问题

（1）**license**文件有什么用，该放在什么地方？

license文件需要申请，目的是作为sdk校验开发者的使用合法性，license文件放置位置不对或未放置license文件会导致没法使用sdk，一般应先申请license文件，并把申请得到的license文件，放置在assets目录下面。

（2）**FaceSDK-LicenseLICENSE_INFO_CHECK_ERROR =license id**

licenseID为您申请时填appname+"_face_android"。如下图demo-turnstile-face-android为license里面的licenseID, **demo-turnstile-face-android1**为app运行时Config.licenseID，两者必须一致

```
E/FaceSDK: FaceSDK-License LICENSE_INFO_CHECK_ERROR =license id  demo-turnstile-face-android  demo-turnstile-face-android1
```

（3）**FaceSDK-LicenseLICENSE_INFO_CHECK_ERROR =signature md5**

md5不一致错误，签名的为license里面的md5，后面的为app运行时获取的签名文件的md5，这两个md5必须一致且区分大小写。 `E/FaceSDK: FaceSDK-License LICENSE_INFO_CHECK_ERROR =signature md5  F5846C60804CC6042D55D09F1AB82364  4357A3EDBC0CA02EA8B5E0578E58D`

（4）**FaceSDK-License LICENSE_INFO_CHECK_ERROR =package name**

PackageName不一致错误。License里面的packagename为申请license时填的，需要保证和app里面的packagename一致。

（5）活体检测常见有那些动作？是否可配置？

常见有6个动作，眨眼，张大嘴，向上抬头，向下低头，向左摇头，向右摇头等。 sdk提供FaceConfig参数设置类，如活体检测角度、光线，检测动作，检测动作数量等设置。

（6）使用sdk一般会用到活体检测拍照等功能，有什么需要注意？

Android 6.0+，需要注意相机拍摄权限问题。如没申请权限，可能导致没法调起相机。

（7）在有些机型上出现特别卡或出现无响应？

SDK在**armeabi**上性能非常差，建议删掉其他so只留下armeabi-v7a，包括使用的其他第三方so。因为如果其他so有armeabi，根据android系统查找so的逻辑，在armeabi的机型上只会去该目录下查找so，而人脸SDK没有，就会出现找不到so。

（8）**license** 文件失效了，不能用了怎么办？

license文件申请时候有期限，如过期会导致校验失效，需要在后台申请延期。

IOS-基础版

版本记录

版本	日期	更新说明
v3.3.0	2018.06.15	增加多人脸检测功能；性能优化，修复部分bug

目录

1 基础信息
1.1 产品概述
1.2 规格信息
1.3 兼容性
1.4 开发包说明
2 业务介绍
2.1 功能简介
2.2 业务流程
3 快速集成
3.1 创建SDK使用的鉴权文件
3.2 下载SDK
3.3 快速测试
4 接口说明
4.1 人脸采集
4.2 多人脸追踪
4.3 FaceSDK鉴权方法
4.4 鉴权成功的凭证
4.5 设置人脸功能控制参数
4.6 人脸质量检测配置
4.7 人脸图像检测
4.8 人脸精准对齐
4.9 人脸追踪
4.10 人脸姿态获取
4.11 人脸姿态获取(常用方法)
4.12 多人脸追踪
4.13 人脸追踪识别
4.14 最大人脸追踪识别
4.15 人脸图片质量检测
4.16 人脸抠图
5 常见问题

1、基础信息

1.1 产品概述

百度FaceSDK iOS基础版是一种面向iOS移动设备人脸技术开发包，此版SDK包含人脸检测、活体识别等API。基于该方案，开发者可以轻松的构建包含人脸检测、采集的应用。在您使用SDK之前，我们首先为您介绍以下SDK本身及相关人脸能力，以便您能更方便使用人脸服务。

1.2 规格信息

- 包大小：~10M
- 最小人脸检测大小：50px * 50px
- 可识别人脸角度：yaw ≤ ±30°, pitch ≤ ±30°
- 检测速度：100ms 720p*
- 追踪速度：30ms 720p*
- 人脸检测耗时：< 100ms

> 备注：以上指标，由最新版SDK运行在真实设备上，采用真实数据集所得，但**算法性能受实际运行设备、实际数据集等情况影响**，以上数字仅供参考。

1.3 兼容性

- **系统**：支持**iOS 8**及以上系统。需要开发者通过Deployment Target来保证支持系统的检测。

- **机型**：手机和平板皆可
- **网络**：支持 WIFI 及移动网络,移动网络支持使用 NET 网关及WAP 网关(CMWAP、CTWAP、UNIWAP、3GWAP)。

1.4 开发包说明

文件/文件夹名	说明
FaceSDK	FaceSDK 包、bundle资源文件、模型文件、鉴权文件
Core	相机视频流处理类
Utils	图片坐标转换、图片资源加载类
ViewController	DEMO工程
FaceParameterConfig.h	配置信息

1.4 授权介绍

FaceSDK iOS版本，基于产品线授权，

2、业务介绍

2.1 功能简介

2.1.1 人脸检测与跟踪

可在设备端，离线实时检测视频流中的人脸。同时支持处理静态图片或者视频流，并对当前检测到的人脸持续跟踪，动态定位人脸轮廓，稳定贴合人脸。

2.1.2 质量控制

在人脸检测及追踪过程中，实时校验人脸的姿态角度、遮挡、清晰度、光照条件，符合质量条件的人脸图片才会被采集。

2.1.3 人脸采集

针对视频流实时完成人脸图片采集，并输出满足质量过滤条件的人脸图片，可自定义采集人脸大小，采集频率，采集质量等设置。

2.2 业务流程

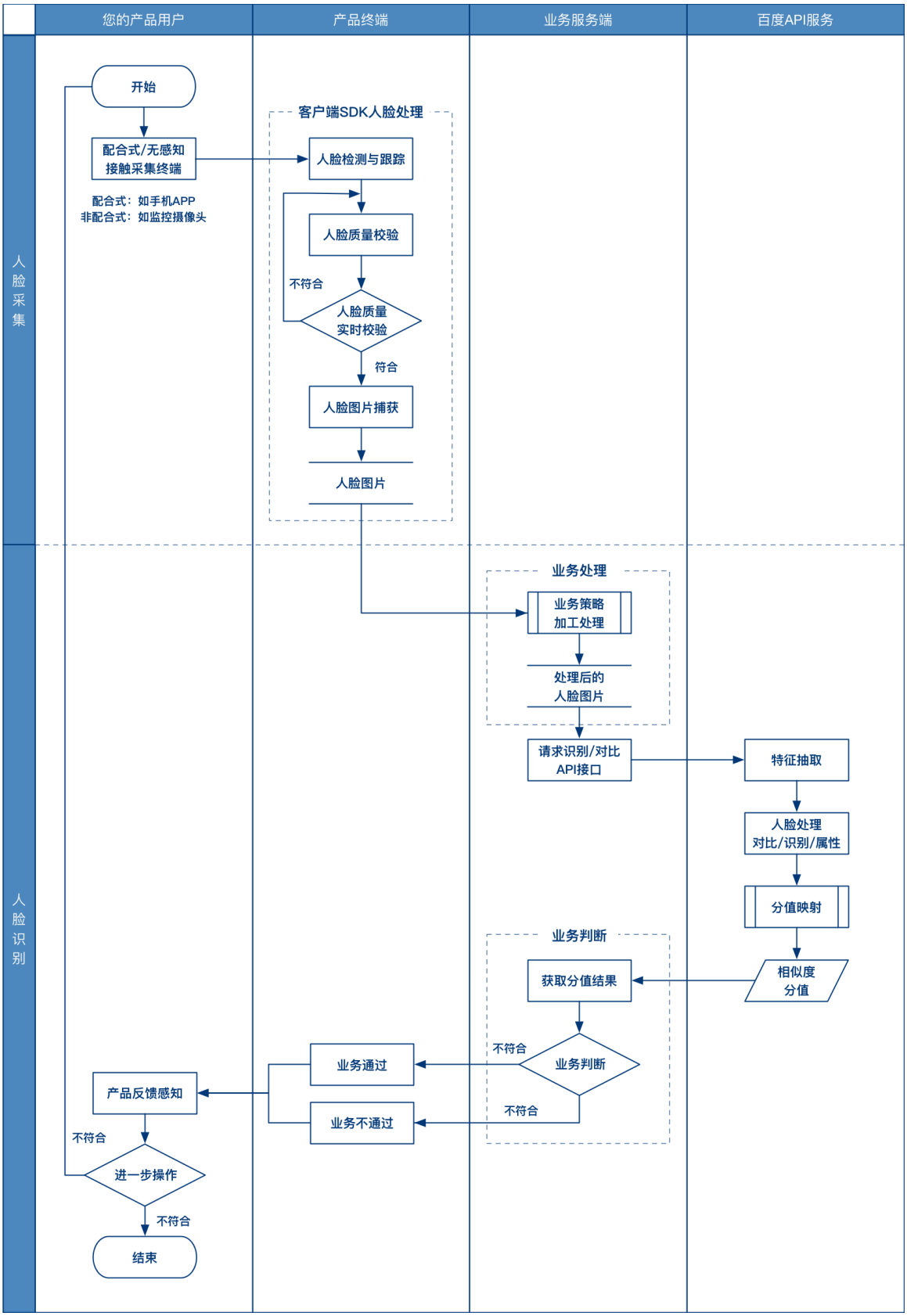
人脸识别的应用场景，核心可以分为三大类：

1. **身份核验**：即1:1对比，判断两张脸的相似度，判断`你是你`，通常用于需要验证用户身份真实性的场景，如人证对比。
2. **身份识别**：即1:N识别，在一个人脸集合中找到最相似的人脸，判断`你是谁`，通常用于判断用户身份是否存在，及身份信息内容的场景，如人脸门禁、人脸支付等。
3. **属性分析**：即人脸属性分析，基于人脸信息，返回年龄、性别等属性值，通常用于客群分析、娱乐营销等场景，如统计线下客群年龄分布。

而以上场景的几乎所有业务过程，核心可以分为两个步骤：

1. **人脸采集**：人脸识别的前置步骤，即获取到人脸图片，用于对比、识别、属性分析等操作。
2. **人脸分析**：包括人脸图片的加工处理，特征抽取与对比，结果返回等一系列操作，也是通常理解为的人脸识别操作。

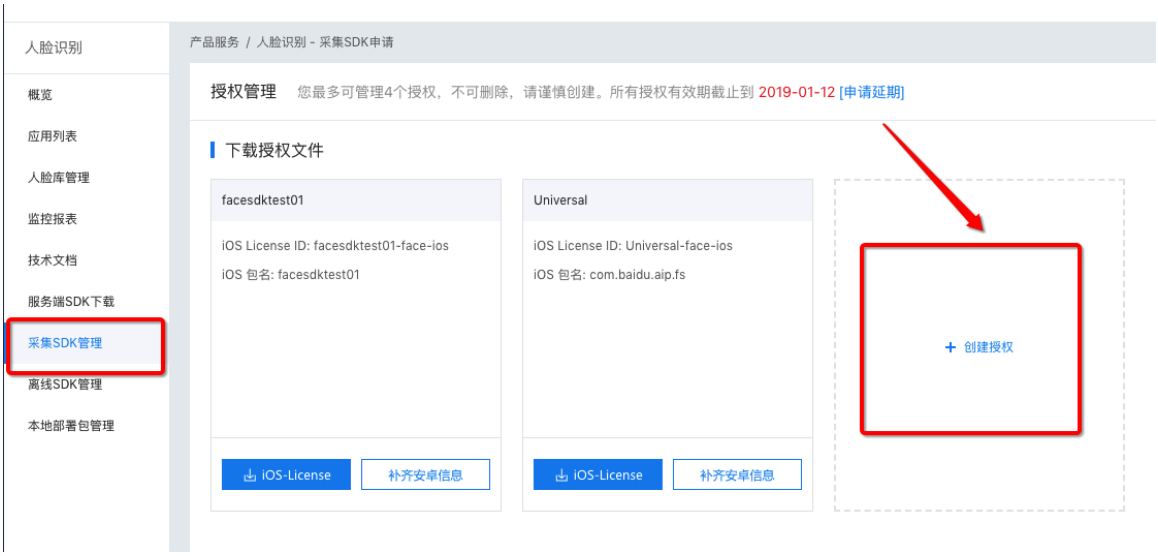
要想确保人脸识别的应用效果得到保障，最为核心的一个环节即人脸的获取，即**人脸采集**。目前市面上所有人脸识别应用落地，面临的主要问题就是**应用环境复杂**，包括光照、遮挡、作弊攻击等一系列环境因素干扰，涉及产品策略、硬件选型、施工方案等多个维度地综合作用，才能不断提升最终效果。



3、快速集成

3.1 创建SDK使用的鉴权文件

人脸SDK License：此license用于SDK离线功能使用，在您的申请人脸SDK的后台页面，**全局->产品服务->人脸识别->采集SDK**，点击创建授权文件。



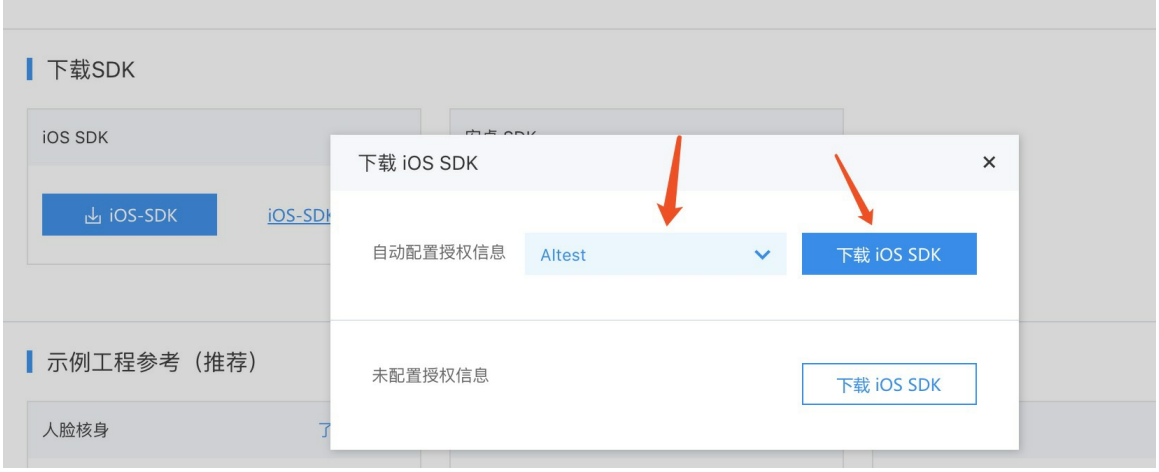
在弹出的框中输入授权标识，选择应用类型，应用系统，以及包名，详情请查看输入框右边提示。

3.2 下载SDK

基础采集SDK下载请点击SDK的下载地址

SDK下载分两种：

1、自动配置授权信息。（license创建后可以选择该方式，下载SDK后自动帮您配置授权，不用单独导入license，初始化参数licenseID、包名自动配置）



2、手动导入授权信息。 需要手动导入license文件，配置好bundleID、licenseID等信息；

```
14
15 // 人脸license文件名
16 #define FACE_LICENSE_NAME    @"idl-license"
17 // 人脸license后缀
18 #define FACE_LICENSE_SUFFIX  @"face-ios"
19 //  (您申请的应用名称(appname)+「-face-ios」后缀，如申请的应用名称(appname)为test123，则
       此处填写test123-face-ios)
20 // 在后台 -> 产品服务 -> 人脸识别 -> 客户端SDK管理查看，如果没有的话就新建一个
21 #define FACE_LICENSE_ID        @"facesdktest01-face-ios"
```

3.3 快速测试

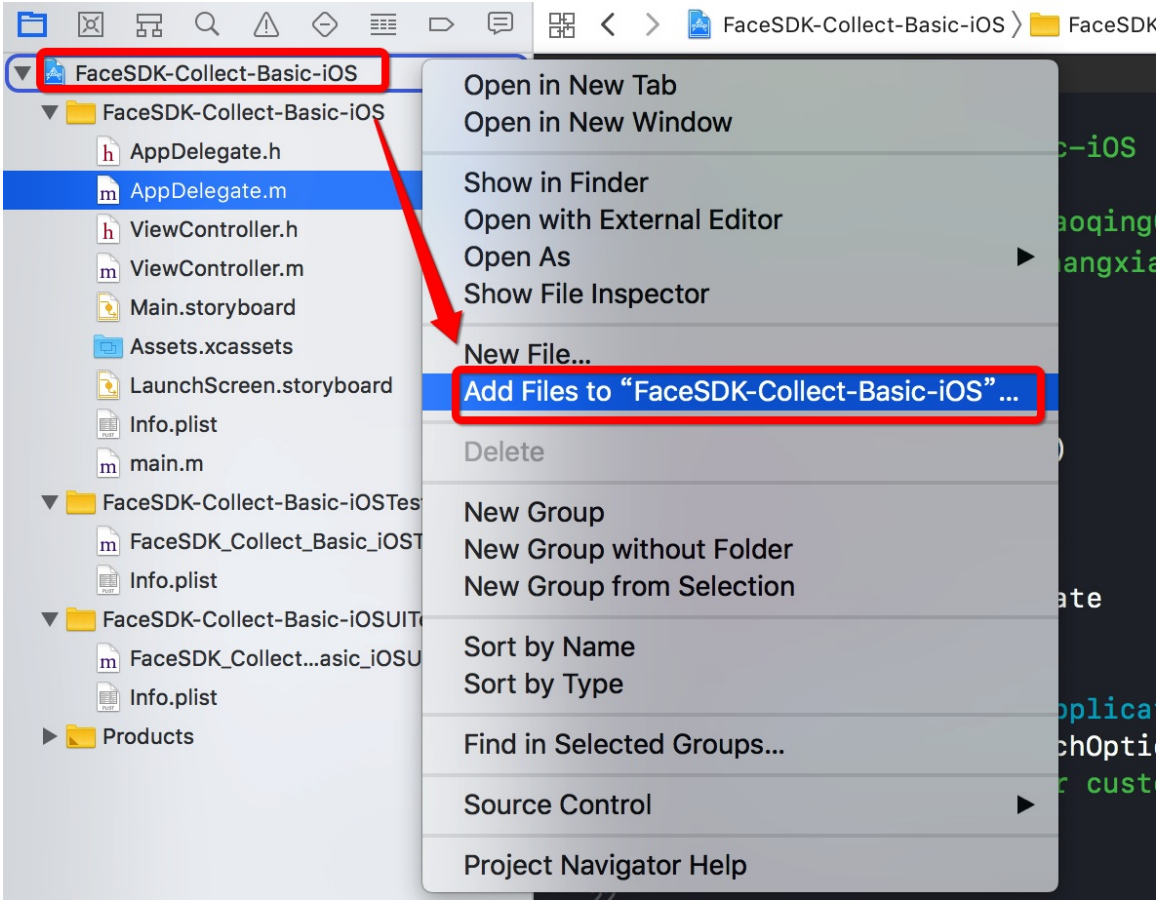
3.3.1 自动配置授权信息集成

如果您是通过自动配置授权信息下载的示例工程，只需配置好证书即可。查看下项目中的`FaceParameterConfig.h`文件，已经自动配置。配置好证书，即可运行。注意已经设置好的bundle id不要随意改动。

3.3.2 手动导入授权信息的集成

打开或者新建一个项目。

右键点击项目，会出现一个添加菜单，在菜单中选择『Add Files to“此处是你的项目名字”…… 』,在添加文件弹出框里面选择申请到的license添加进来 。下载license文件，根据要求配置项目中的FaceParameterConfig.h文件中的内容。 如下图所示：

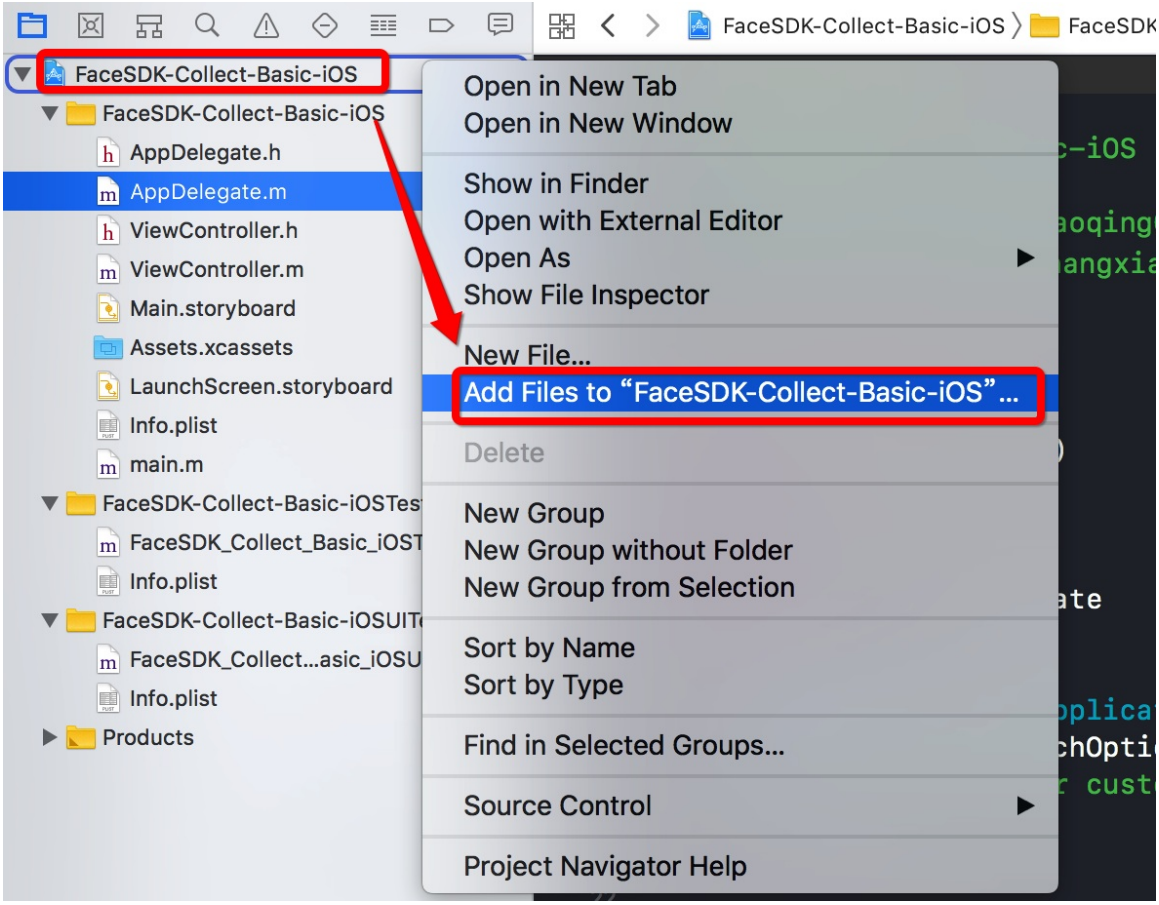


3.3.3 添加SDK到工程

1、打开或者新建一个项目。

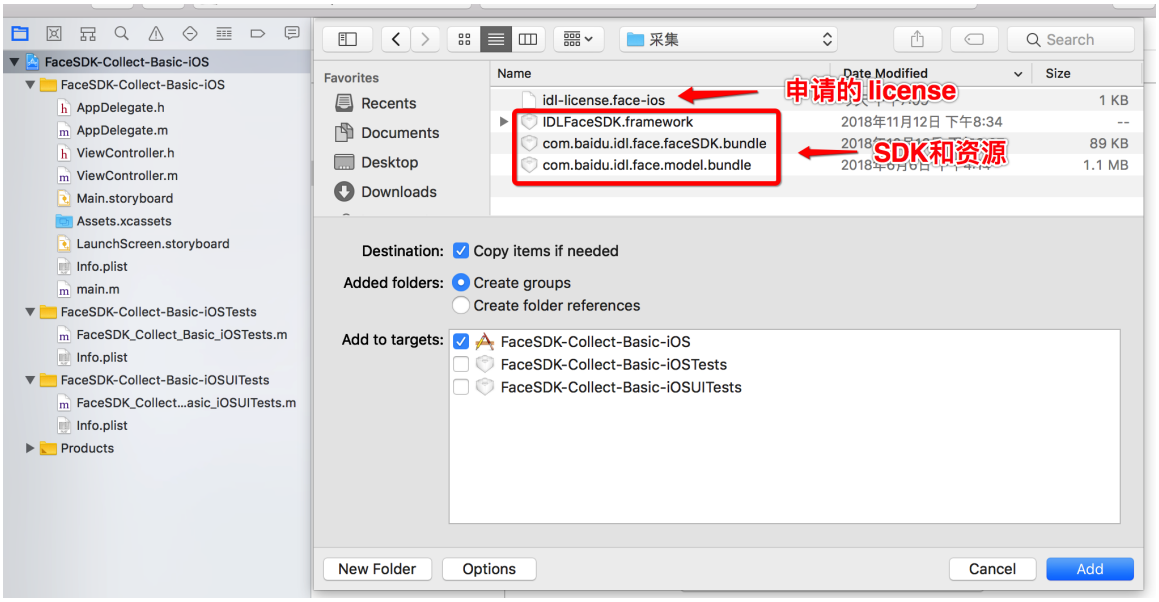
2、右键点击项目，会出现一个添加菜单，在菜单中选择『Add Files to“此处是你的项目名字”…… 』，在添加文件弹出框里面的FaceSDK开发工具包 添加进来。SDK的下载地址

如下图所示：



SDK包含下面3个文件:

文件名/包名	说明
com.baidu.idl.face.faceSDK.bundle	图片音频资源包
com.baidu.idl.face.model.bundle	模型捆绑包
IDLFaceSDK.framework	FaceSDK



3、确认下Bundle Identifier 是否是申请license时填报的那一个，注意：license和Bundle Identifier是一一对应关系，填错了会导致SDK不能用。

```
19
20 //  (您申请的应用名称(appname)+「-face-ios」后缀，如申请的应用名称(appname)为test123，则此处填写test123-face-ios)
21 // 在后台 -> 产品服务 -> 人脸识别 -> 客户端SDK管理查看，如果没有的话就新建一个
22 #define FACE_LICENSE_ID         @"facesdktest01-face-ios"
23
```

4、填写正确的FACE_LICENSE_ID。

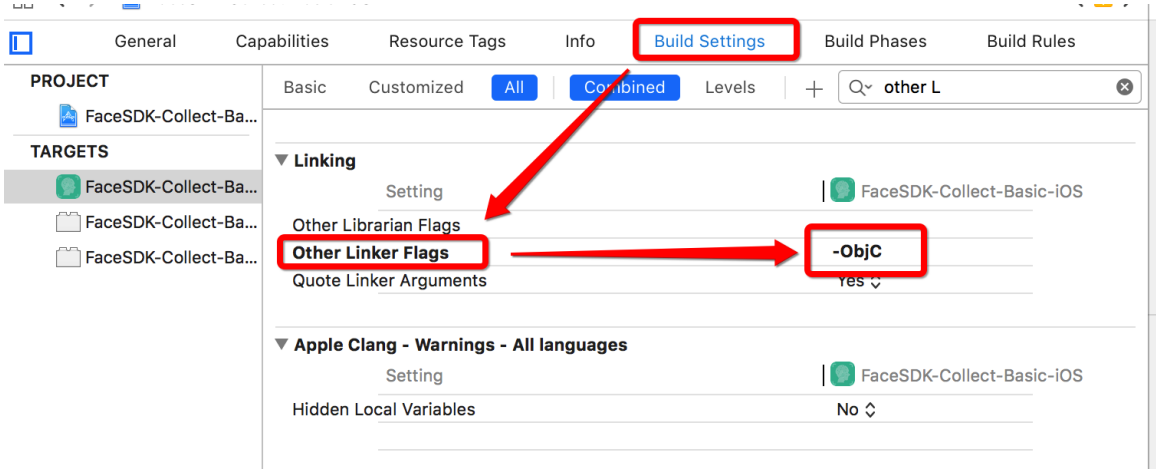
（即后台展示的LicenseID）

在FaceParameterConfig.h文件里面填写拼接好的FACE_LICENSE_ID。

```
19
20 //  (您申请的应用名称(appname)+「-face-ios」后缀，如申请的应用名称(appname)为test123，则此处填写test123-face-ios)
21 // 在后台 -> 产品服务 -> 人脸识别 -> 客户端SDK管理查看，如果没有的话就新建一个
22 #define FACE_LICENSE_ID         @"facesdktest01-face-ios"
23
```

5、选择链接C++标准库。

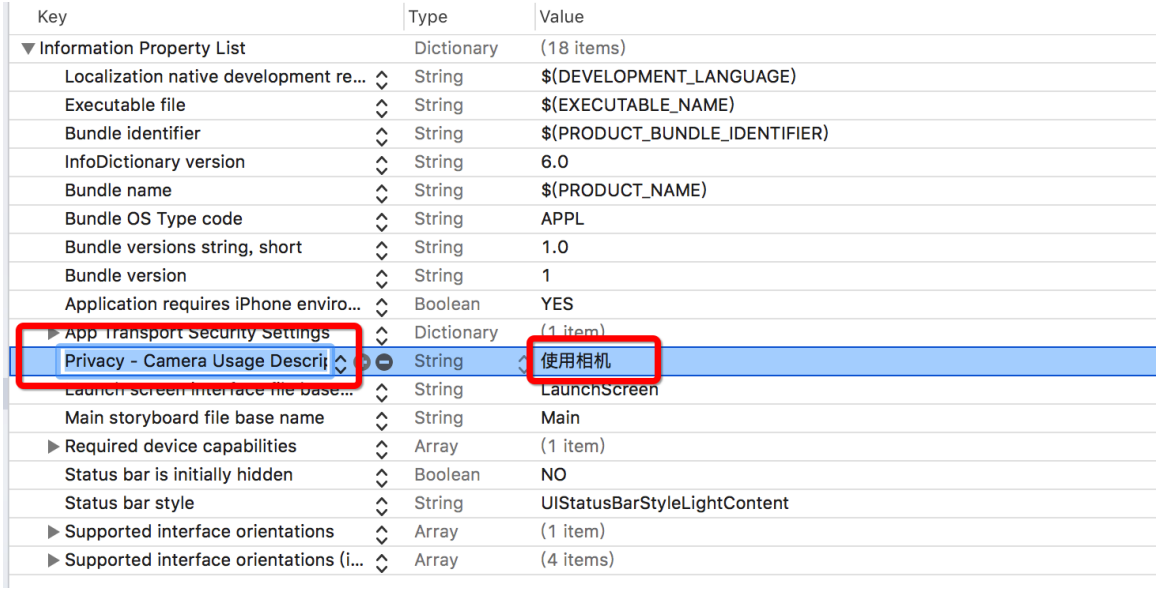
6、如果没有使用pod管理第三方库的话，请在Build Setting >Linking > Other Linker Flags 上面加入–ObjC选项。如果用了pod请忽略，因为pod会自动添加上。



3.3.4 权限声明

需要使用相机权限：编辑Info.plist文件，添加

Privacy- Camera Usage Description的Key值，Value为使用相机时候的提示语，可以填写:「使用相机」



Key	Type	Value
Information Property List	Dictionary	(18 items)
Localization native development re...	String	$(DEVELOPMENT_LANGUAGE)
Executable file	String	$(EXECUTABLE_NAME)
Bundle identifier	String	$(PRODUCT_BUNDLE_IDENTIFIER)
InfoDictionary version	String	6.0
Bundle name	String	$(PRODUCT_NAME)
Bundle OS Type code	String	APPL
Bundle versions string, short	String	1.0
Bundle version	String	1
Application requires iPhone enviro...	Boolean	YES
App Transport Security Settings	Dictionary	(1 item)
Privacy - Camera Usage Descri	String	使用相机
Launch screen interface file base...	String	LaunchScreen
Main storyboard file base name	String	Main
Required device capabilities	Array	(1 item)
Status bar is initially hidden	Boolean	NO
Status bar style	String	UIStatusBarStyleLightContent
Supported interface orientations	Array	(1 item)
Supported interface orientations (i...	Array	(4 items)

4、接口说明

4.1 人脸采集

```
 /*带黑边的方法-通过图片质量控制*/
- (void)detectStratrgyWithQualityControlImage:(UIImage *)image previewRect:(CGRect)previewRect detectRect:(CGRect)detectRect completionHandler:(DetectStrategyCompletion)completion;

/*不带黑边*/
- (void)detectStratrgyWithNormalImage:(UIImage *)image previewRect:(CGRect)previewRect detectRect:(CGRect)detectRect completionHandler:(DetectStrategyCompletion)completion;
```

> **温馨提示**：带黑边的图片，主要是为了配合在线API方式的活体检测使用，本地算法会对图片做一定的预处理操作；而不带黑边的图片，如果无需使用活体，则可使用不带黑边的图片采集方式。

参数说明：

`previewRect`与`detctRect`是为了做距离检测而定义的，为了在上层封装判断脸是否在框内／离太远／离太近。

- image：相机获取的图片
- previewRect：间接定义的最大距离的maxRect和最小距离的minRect。
- detectRect：实际采集区域
- completion：回调

使用方法见下图：

```
    /*不论带不带黑边，取图片都是：images[@"bestImage"]*/
    //带黑边的方法
    __weak typeof(self) weakSelf = self;
    [[IDLFaceDetectionManager sharedInstance]
        detectStratrgyWithQualityControlImage:image
        previewRect:self.previewRect detectRect:self.detectRect
        completionHandler:^(FaceInfo *faceinfo, NSDictionary *images,
        DetectRemindCode remindCode) {
        switch (remindCode) {
            case DetectRemindCodeOK: {
                weakSelf.hasFinished = YES;
                [self warningStatus:CommonStatus warning:@"非常好"];
                if (images[@"bestImage"] != nil && [images[@"bestImage"] count]
                    != 0) {
                    NSData* data = [[NSData alloc] initWithBase64EncodedString:
                        [images[@"bestImage"] lastObject]
                        options:NSDataBase64DecodingIgnoreUnknownCharacters];
                    UIImage* bestImage = [UIImage imageWithData:data];
                    NSLog(@"bestImage = %@",bestImage);
                }
                dispatch_async(dispatch_get_main_queue(), ^{
                    [UIView animateWithDuration:0.5 animations:^{
                        weakSelf.animaView.alpha = 1;
                    } completion:^(BOOL finished) {
                        [UIView animateWithDuration:0.5 animations:^{
                            weakSelf.animaView.alpha = 0;
                        } completion:^(BOOL finished) {
                            [weakSelf closeAction];
                        }];
                    }];
                });
                [self singleActionSuccess:true];
                break;
```

4.2 多人脸追踪

- (void)detectMultiFacesImage:(UIImage *)image withMaxFaceCount:(NSInteger)maxFaceCount handler:(TrackDetectStrategyCompletion)completion;

参数说明：

- image：相机获取的图片
- maxFaceCount：定义追踪的人脸数目
- completion：回调

使用方法见下图：

```
    __weak typeof(self) weakSelf = self;
    [[IDLFaceDetectionManager sharedInstance]detectMultiFacesImage:image withMaxFaceCount:2
        handler:^(FaceInfo *faceinfo, TrackDetectRemindCode remindCode) {
            switch (remindCode) {
            case TrackDetectRemindCodeOK: {
                [self warningStatus:PoseStatus warning:@"成功"];
                NSLog(@"追踪到的人脸数据：%@",faceinfo)
            }
            case TrackDetectRemindCodeImageBlured: {
                [self warningStatus:PoseStatus warning:@"图像模糊"];
            }
```

4.3 FaceSDK鉴权方法

- (void)initLicenseID:(NSString *)licenseId localLicenceFile:(NSString *)localLicencePath remoteAuthorize:(BOOL)remoteAuthorize;

参数：

- licenseId：平台申请的 licenseID
- localLicencePath：鉴权文件路径
- remoteAuthorize：是否开启网络鉴权

返回：

- 无

4.4 鉴权成功的凭证

```
- (FaceLicenseErrorCode)canWork;
```

参数：

- 无

返回：

- FaceLicenseErrorCode 返回值

4.5 设置人脸功能控制参数

具体方法详见如下：

```
// 设置鉴权
-(void)setLicenseID:(NSString )licenseID andLocalLicenceFile:(NSString)localLicencePath;
// 最小检测人脸阈值
-(void)setMinFaceSize:(NSInteger)width;
// 截取人脸图片大小
-(void)setCropFaceSizeWidth:(CGFloat)width;
// 人脸检测精度阈值
-(void)setNotFaceThreshold:(CGFloat)th;
// 人脸遮挡阈值
-(void)setOccluThreshold:(CGFloat)thr;
// 亮度阈值
-(void)setIllumThreshold:(NSInteger)thr;
// 图像模糊阈值
-(void)setBlurThreshold:(CGFloat)thr;
// 头部姿态角度
-(void)setEulurAngleThrPitch:(NSInteger)pitch yaw:(NSInteger)yawroll:(NSInteger)roll;
// 是否进行人脸图片质量检测
-(void)setIsCheckQuality:(BOOL)isCheck;
```

4.6 人脸质量检测配置

```
- (void)setQualityCheckAbility:(FaceQualityType)type isOpen:(BOOL)open;
```

> 注意：质量检测默认关闭，要想开启对应功能需要先调用setIsCheckQuality接口方法参数:

参数：

- type: 质量检测类型 FaceQualityType
- open：是否开启

返回：

- 无

4.7 人脸图像检测

```
- (void)detectFaceInImage:(UIImage *)image
         withMethodType:(FaceSDKDetectMethodType)methodType
        andGetFaceRects:( NSArray *_Nullable*_Nullable)faceRects
          andFaceNumber:(NSInteger *)faceNumber
      andMinimumFaceSize:(NSInteger)minSize;
```

参数:

- image 图像
- methodType 检测算法模型 CNN（可见光图片检测）、NIR（近红外图片检测）
- minSize 最小检测人脸size

返回：

- faceRects output 人脸框的位置
- faceNumber 检测出的人脸的数目

4.8 人脸精准对齐

```
- (void)fineAlignFaceInImage:(UIImage *)image
            withMethodType:(FaceSDKMethodType)methodType
              andFaceShape:(NSArray *_Nullable*_Nullable)faceShape
            andNumOfPoints:(NSInteger)numberOfPoints;
```

参数：

- image 图像
- methodType 识别算法模型 SDM / CDNN（建议用）

返回：

- faceShape input/output人脸特征点
- numberOfPoints 人脸特征点个数

4.9 人脸追踪

```
- (void)trackFaceInImage:(UIImage *)image
          withMethodType:(FaceSDKMethodType)methodType
            andFaceShape:( NSArray *_Nullable*_Nullable)faceShape
          andNumOfPoints:(NSInteger )numberOfPoints
               andScore:(CGFloat *)score;
```

参数：

- image 图像
- methodType 识别算法模型 SDM / CDNN（建议用）
- faceShape 72个人脸特征点坐标数组

- numberOfPoints 人脸特征点个数，目前只支持72个点

返回：

- score output置信度

4.10 人脸姿态获取

```
- (NSDictionary *)headPostEstimationWithFaceShape:(NSArray *)faceShape
                                  andImgDataRowNum:(NSInteger)rows
                                    andImgColumNum:(NSInteger)colums
                                    andNumOfPoints:(NSInteger )numberOfPoints;
```

参数：

- faceShape input and output 人脸特征点
- numberOfPoints 人脸特征点个数
- rows 图像宽
- colums 图像高

返回：

- NSDictionary pitch、row、yaw

4.11 人脸姿态获取(常用方法)

```
- (NSDictionary *)headPostEstimationWithImage:(UIImage *)image
                                 andFaceShape:(NSArray *)faceShape
                                andNumOfPoints:(NSInteger )numberOfPoints;
```

参数：

- image 图像
- faceShape input and output 人脸特征点
- numberOfPoints 人脸特征点个数

返回：

- NSDictionary pitch、row、yaw

4.12 多人脸追踪

```
- (void)trackWithImage:(UIImage *)image andMaxFaceCount:(NSUInteger)count;
```

参数：

- image 图像
- count 追踪人脸数目

返回：

- 无

4.13 人脸追踪识别

最先检测到人脸

```
 - (FaceVerifierErrorCode)prepareDataWithImage:(UIImage *)image
andActionType:(FaceVerifierActionType)actionType;
```

参数：

- image 图像
- actionType 采集方式（FaceVerifierActionTypeRecognition）

返回：

- FaceVerifierErrorCode 人脸识别结果

4.14 最大人脸追踪识别

```
 - (FaceVerifierErrorCode)prepareDataForMaxFaceWithImage:(UIImage *)image andActionType:
(FaceVerifierActionType)actionType;
```

参数：

- image 图像
- actionType 采集方式（FaceVerifierActionTypeRecognition）

返回：

- FaceVerifierErrorCode 最大人脸识别结果

4.15 人脸图片质量检测

一般用于trackImage方法后用

```
- (void)imageQualityWith:(UIImage *)image
          andFaceShape:(NSArray *_Nullable*_Nullable)faceShape
       andNumOfPoints:(NSInteger)numberOfPoints
            bluriness:(CGFloat *)bluriness
                illum:(NSInteger *)illum
            occlusion:(NSArray *_Nullable*_Nullable)occlusion
           nOccluPart:(NSInteger *)nOccluPart;
```

参数：

- image 图像
- faceShape 人脸特征点
- numberOfPoints 人脸特征点个数

返回：

- bluriness 人脸模糊值
- illum 光照值
- occlusion 人脸部位
- nOccluPart 人脸遮挡部位数量

4.16 人脸抠图

一般用于trackImage方法后用

```
- (void)cropFaceImageWith:(UIImage *)image
                FaceShape:(NSArray *_Nullable*_Nullable)faceShape
              numOfPoints:(NSInteger)numberOfPoints
           faceImageWidth:(NSInteger)width
          faceImageHeight:(NSInteger)height
                cropImage:(UIImage *_Nullable*_Nullable)cropImage
                cropShaps:(NSArray *_Nullable*_Nullable)cropShaps;
```

参数：

- image 图像
- faceShape 人脸特征点
- numberOfPoints 人脸特征点个数

返回：

- width 抠图宽
- height 抠图高
- cropImage 抠图结果图片
- cropShaps 抠图结果图片特征点

5、常见问题

Q：鉴权问题。提示「验证失败」
A：先确定网络情况是否正常，本地鉴权文件失效了才走网络鉴权。定位错误码，排查鉴权失败的原因。一般是licenseID 和bundleID配置不一致导致的鉴权失败。请注意上线前授权文件一定要更新。

Q：license文件失效了，不能用了怎么办？
A：License文件申请时候有期限，如过期会导致校验失效，需要在后台进行申请延期。

Q：使用iOS采集端，采集到的图片是斜着的，这个正常吗，会影响识别吗？
A：不会影响识别。有黑边和倾斜是因为图片质量算法造成的，我们是按1:3对图像进行背景填充使人脸居中，为的是更好的识别图像。这个版本提供了`detectStratrgyWithQualityControlImage`和`detectStratrgyWithNormalImage`两种方法供选择。

更多问题请点击 常见问题

IOS-有动作活体版

Face SDK iOS开发文档

目录

1 简介
1.1 功能介绍
1.2 兼容性
1.3 开发包说明
2 集成指南
2.1 Sample示例
2.2 准备工作
2.2.1 申请license
2.2.2 下载SDK
2.3 运行示例工程
2.3.1自动配置授权信息集成
2.3.2 未使用自动配置授权信息的集成
2.4添加SDK到工程
2.5权限声明
3 功能使用
3.1 活体识别
3.2人脸采集
3.3质量校验设置
3.4 界面定制说明
3.5接口设计说明
3.5.1 DetectRemindCode
3.5.2 FaceLivenessActionType
3.5.3 LivenessRemindCode
3.6创建实例
3.7 设置人脸功能控制参数
3.8 取得人脸图像采集功能接口
3.9 取得活体检测功能接口
3.10 设置人脸图像采集功能参数
3.11 设置人脸功能控制参数
3.12 活体动作设置
4 常见问题

1、简介

1.1 功能介绍

百度Face SDK IOS 版是一种面向 IOS 移动设备人脸技术开发包，此版SDK包含人脸检测、活体识别等功能。基于该方案，开发者可以轻松的构建包含人脸检测、采集和活体识别的应用。在您使用SDK之前，我们首先为您介绍以下SDK本身及相关人脸能力，以便您能更方便使用人脸服务。

此版SDK所包含的能力如下：

- **本地版活体检测**：通过让用户做出指定人脸配合式的交互动作，识别当前操作者是否为活体，此功能为离线使用，可设定指定动作是否使用及应用顺序。动作包含：眨眼、张嘴、左摇头，右摇头，摇摇头、向上抬头，向下低头七个。可有效抵御高清图片、3D建模、视频等攻击。
- **本地版人脸质量检测**：判断视频流中的图片帧中，哪些图片质量较佳，即人脸图像特征清晰（满足角度、姿态、光照、模糊度等校验）。
- **本地版人脸图像采集**：通过本地SDK能力，采集人脸图像，同时经过人脸质量检测，确保采集到的人脸图像符合各条件校验（角度、姿态、光照、模糊度等），为设备前端获取有效可分析人脸的主要功能。
- **离线授权**：SDK的授权判断，授权介质也称为license，在SDK使用中，需要通过license向授权服务器发起请求，判断SDK的使用合法性及使用有效期。

> 此版SDK**全部功能为离线版本，所有功能均本地化使用**，主要用于在客户端（Android）获取人脸，实际业务使用中，可以按照业务需要，配合在线API完成全流程的业务集成。

为了方便您的开发，我们已经为您准备了多种场景的示例工程，您可以根据业务需要，在后台进行直接下载，目前支持【人脸核身】【人脸闸机/门禁】【人脸登录/考勤】【多人脸识别】,示例工程参考下图：

1.2 兼容性

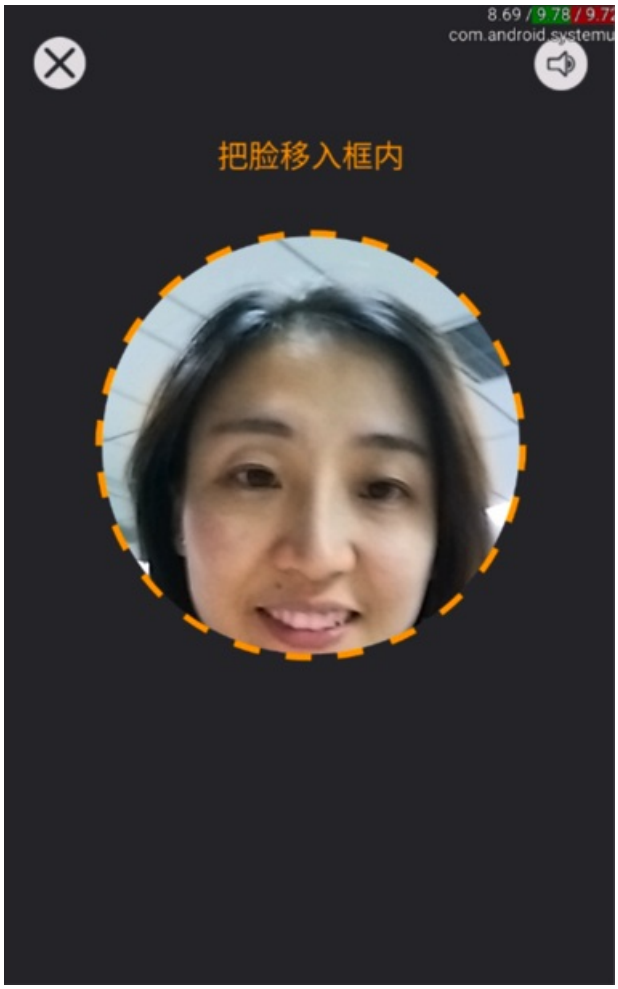
- **系统**：支持iOS8以上系统。需要开发者通过Deployment Target 来保证支持系统的检测。
- **机型**：手机和平板皆可
- **网络**：支持 WIFI 及移动网络,移动网络支持使用 NET 网关及WAP 网关(CMWAP、CTWAP、UNIWAP、3GWAP)。

1.3 开发包说明##

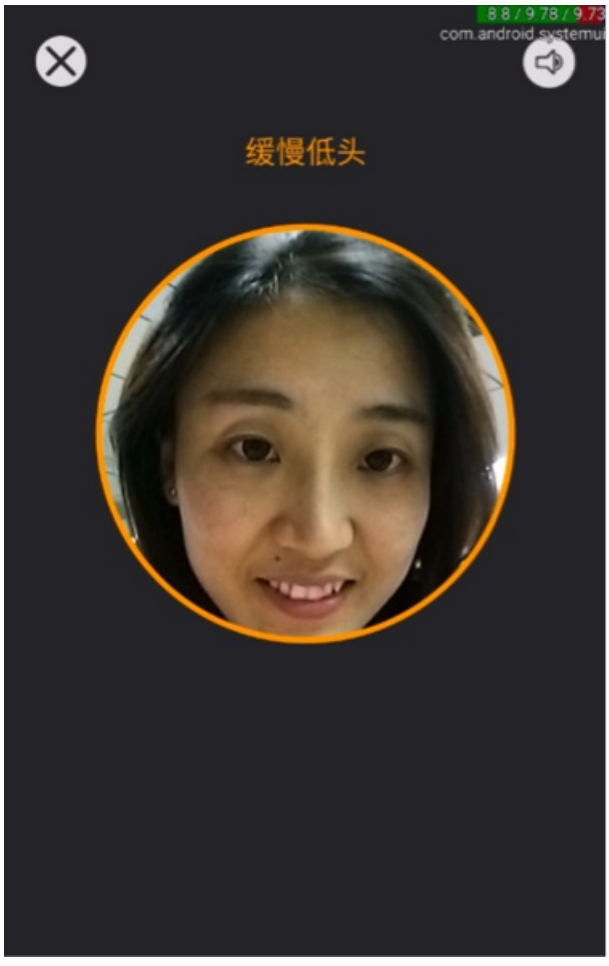
2、集成指南

2.1 Sample示例

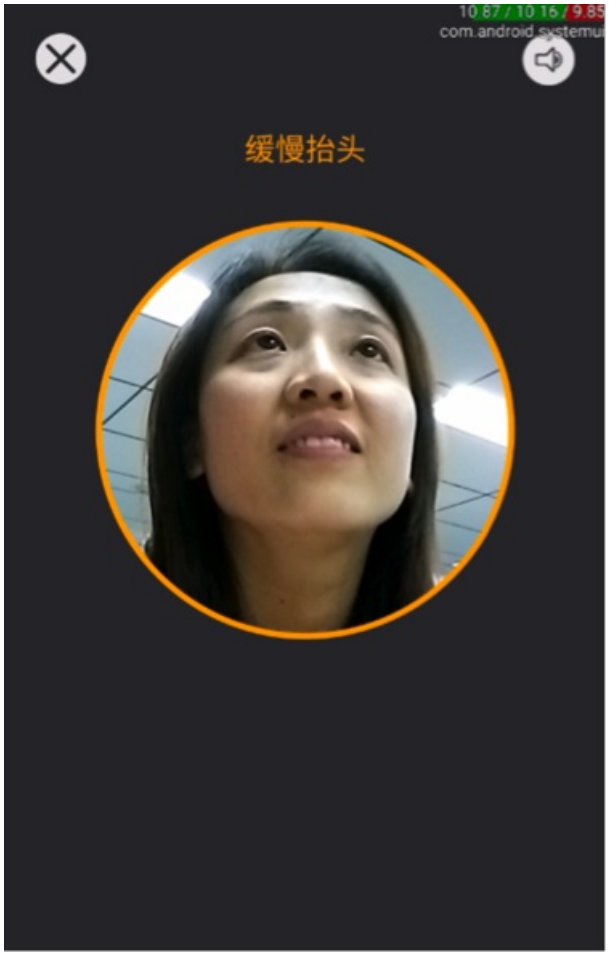
- 把脸移入框内



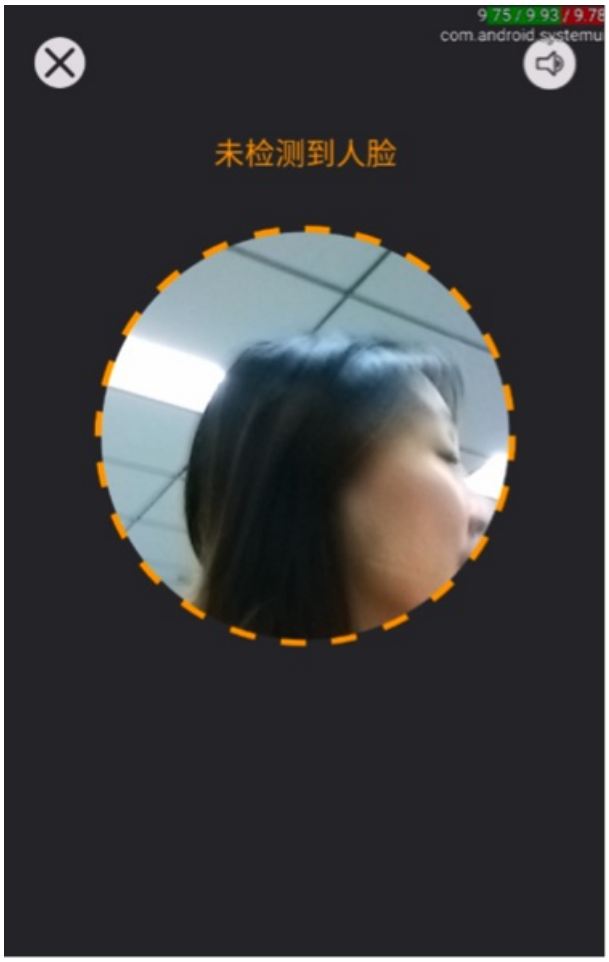
- 慢慢低头



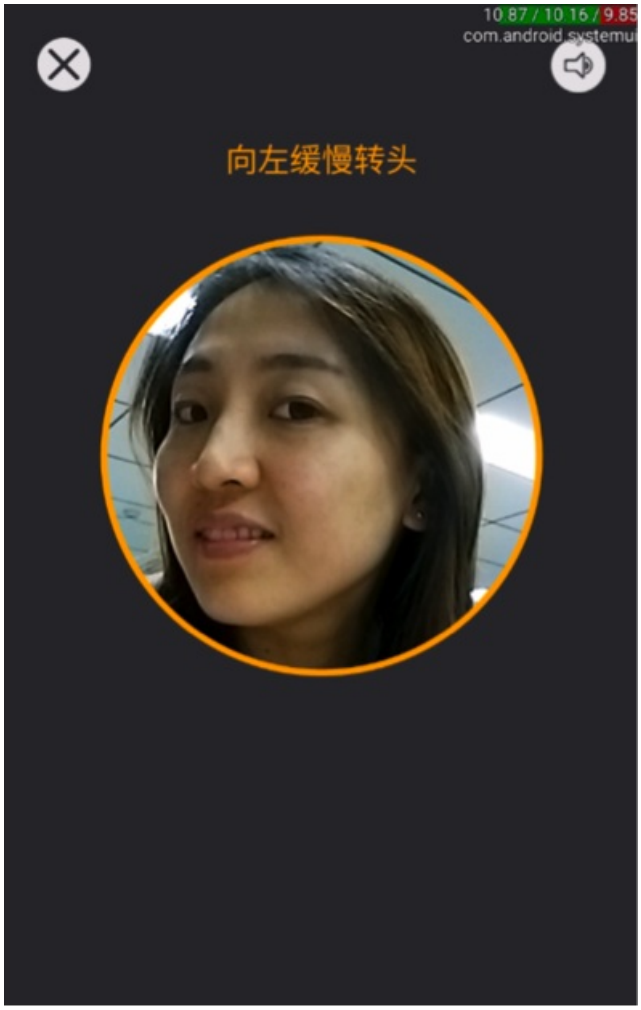
- 慢慢抬头



- 未检测到人脸



- 向左缓慢摇头

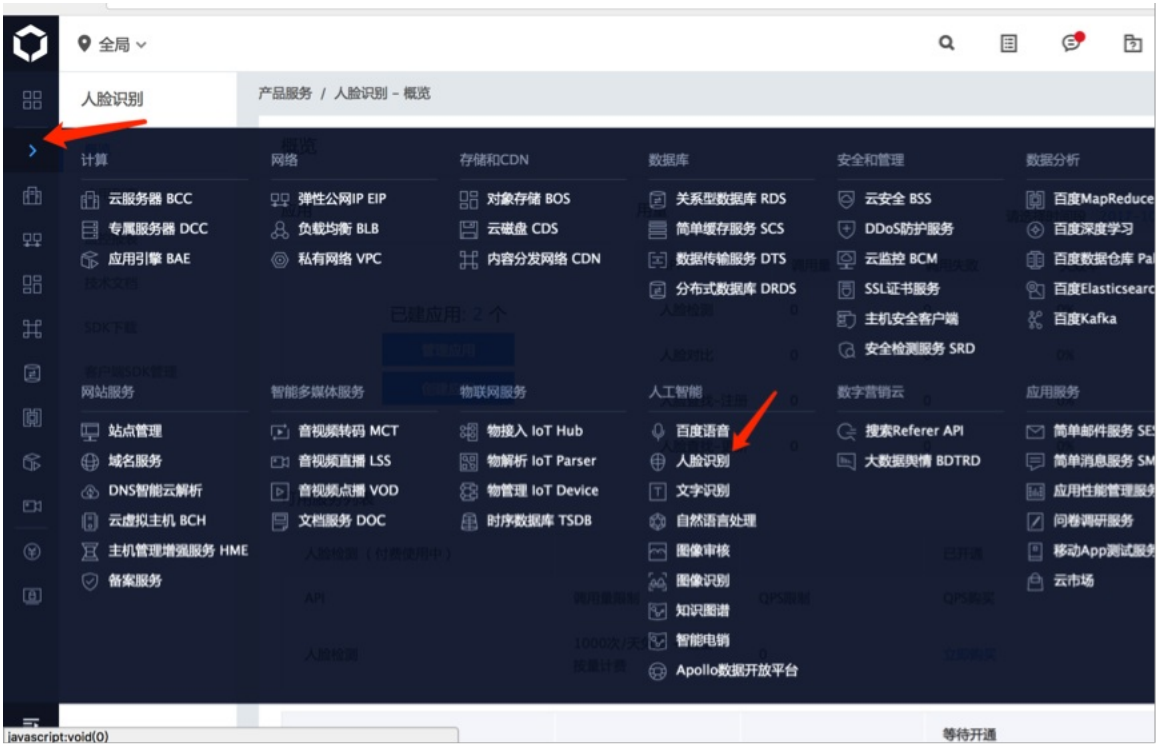


2.2 准备工作

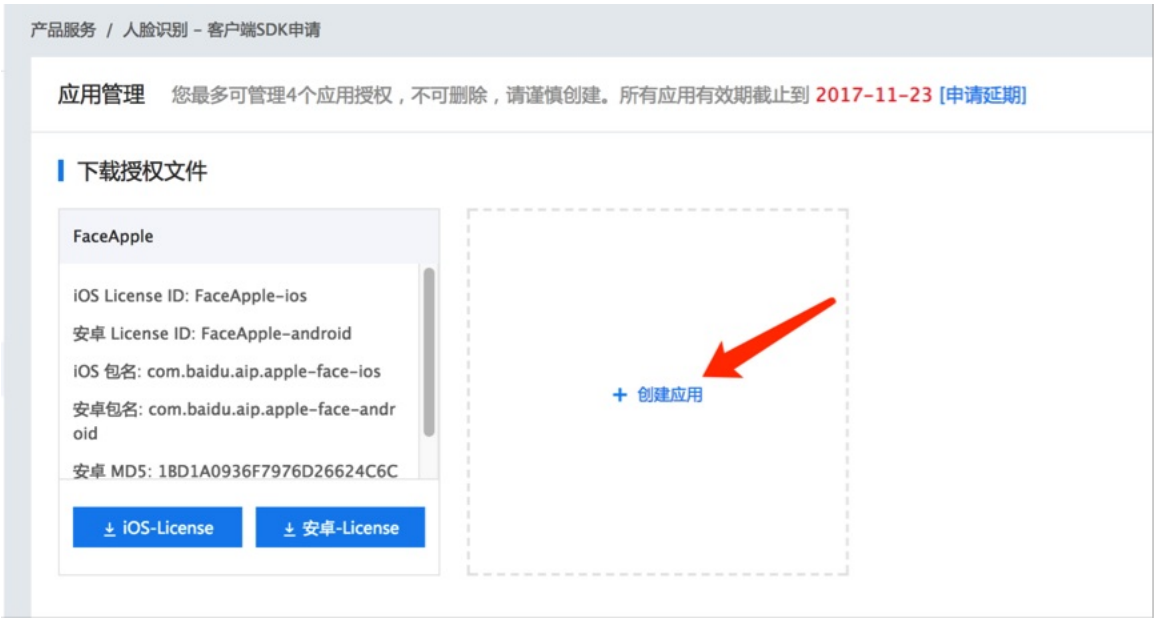
2.2.1 申请license

人脸SDK License：此license用于SDK离线功能使用，在您的申请人脸SDK的后台页面，全局->产品服务->人脸识别->客户端SDK申请

人脸控制台路径如下：



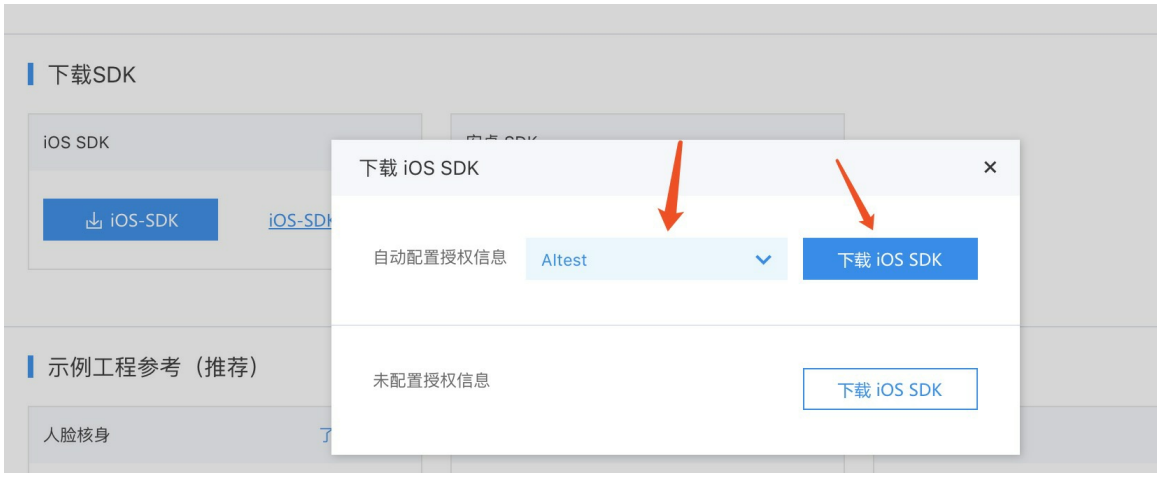
点击客户端SDK管理，弹出如下图：创建应用（这里创建应用是为了使用离线SDK，上面创建应用为了使用人脸在线接口，如注册、识别等）



在弹出的框中输入授权标识，选择应用类型，应用系统，以及包名，详情请查看输入框右边提示

2.2.2 下载SDK

下载SDK分为自动配置授权信息（创建license后就可以选择为该应用，下载后SDK自动帮您配置授权，不用下载license拷贝到工程中，初始化参数licenseID,包名也帮您配置好了）和未配置授权信息两种方式：



2.3 运行示例工程

2.3.1 自动配置授权信息集成

如果您是通过自动配置授权信息下载的示例工程，只需配置好证书即可。查看下项目中的FaceParameterConfig.h文件，已经自动配置

```
11
12 // 人脸license文件名
13 #define FACE_LICENSE_NAME    @"idl-license"
14
15 // 人脸license后缀
16 #define FACE_LICENSE_SUFFIX  @"face-ios"
17
18 //  (您申请的应用名称(appname)+「-face-ios」后缀，如申请的应用名称(appname)为test123，则此处填写test123-face-ios)
19 #define FACE_LICENSE_ID        @"BaiduFaceDemo-face-ios"
20
```

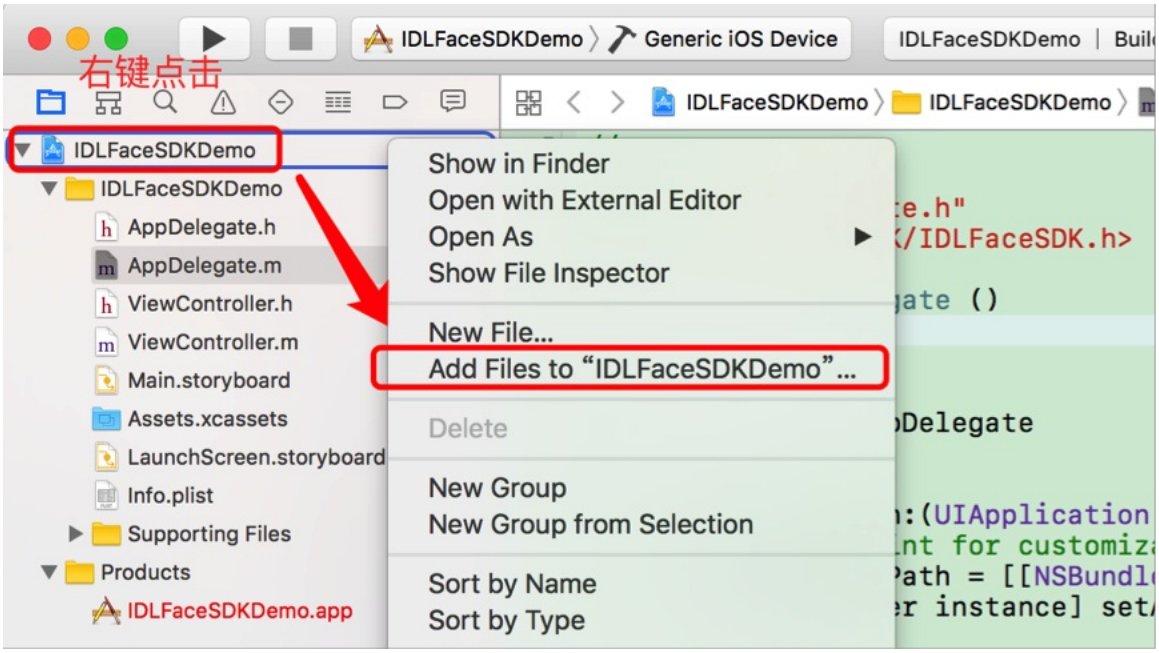
示例图

配置好证书，即可运行。注意已经设置好的bundle id不要随意改动。

2.3.2 未使用自动配置授权信息的集成

打开或者新建一个项目。

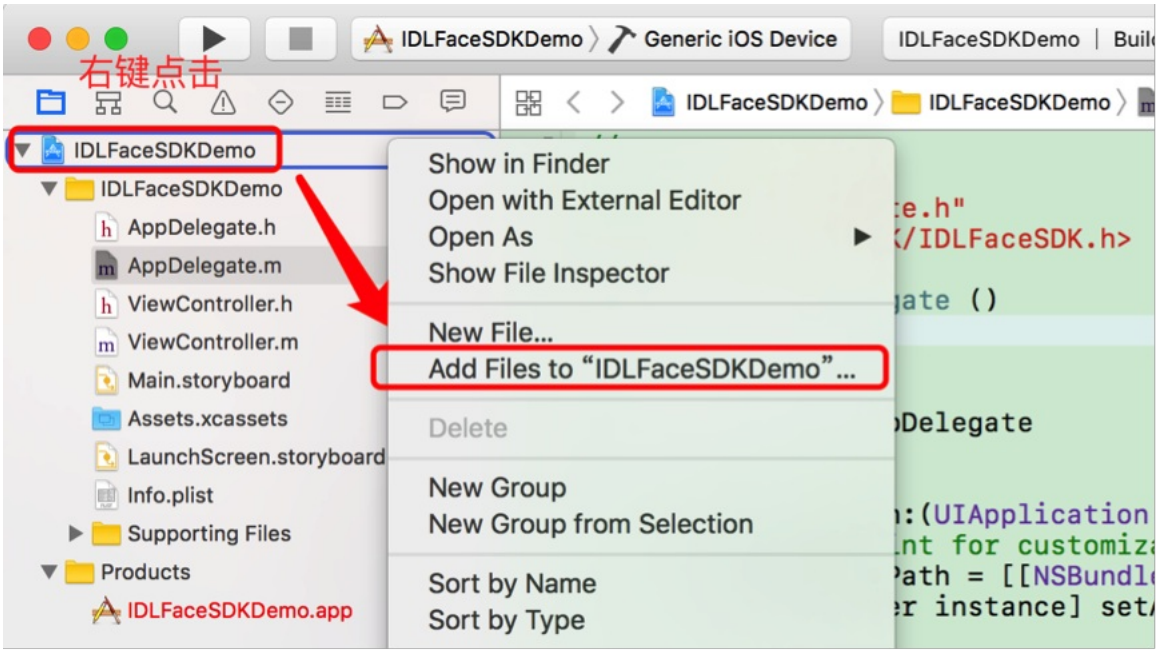
右键点击项目，会出现一个添加菜单，在菜单中选择『Add Files to“此处是你的项目名字”…… 』,如下图所示：



2.4 添加SDK到工程##

1. 打开或者新建一个项目。

2. 右键点击项目，会出现一个添加菜单，在菜单中选择『Add Files to“此处是你的项目名字”…… 』,如下图所示：

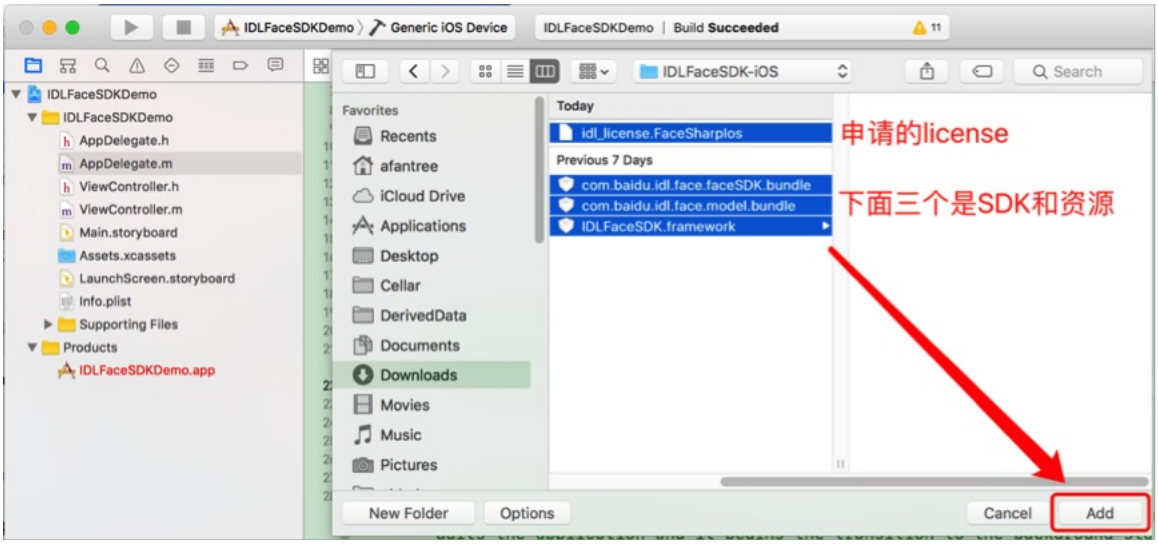


3. 在添加文件弹出框里面选择申请到的license和SDK添加进来。如下图：

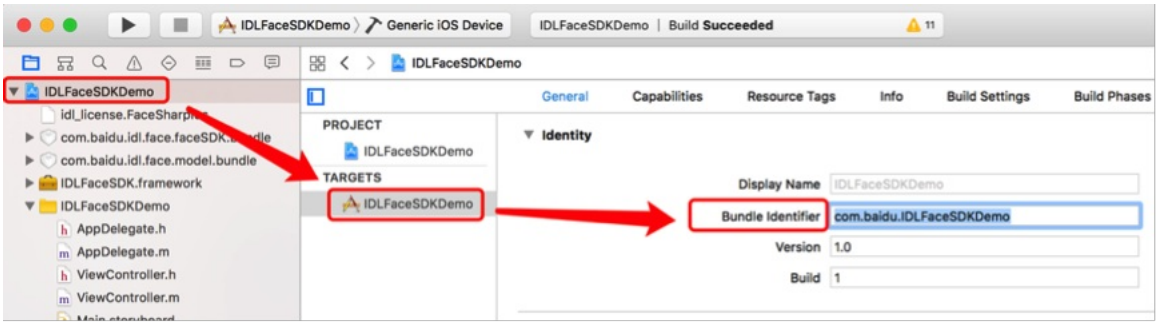
注意：license为百度官方提供的。

SDK包含下面三个文件:

- IDLFaceSDK.framework
- com.baidu.idl.face.faceSDK.bundle
- com.baidu.idl.face.model.bundle



4. 确认下Bundle Identifier 是否是申请license时填报的那一个，注意：license和Bundle Identifier是一一对应关系，填错了会导致SDK不能用。



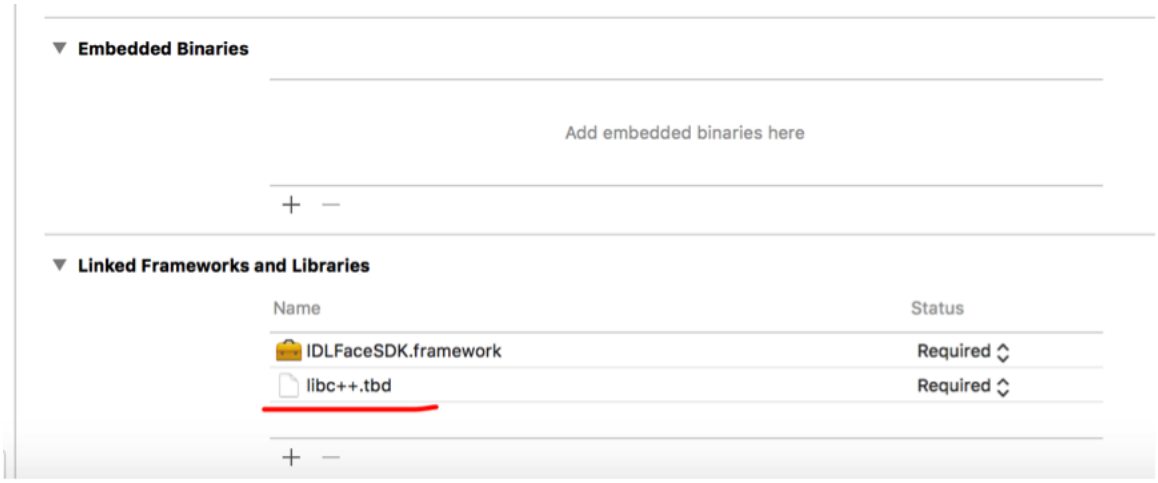
5. 填写正确的FACE_LICENSE_ID。

（即后台展示的LicenseID）

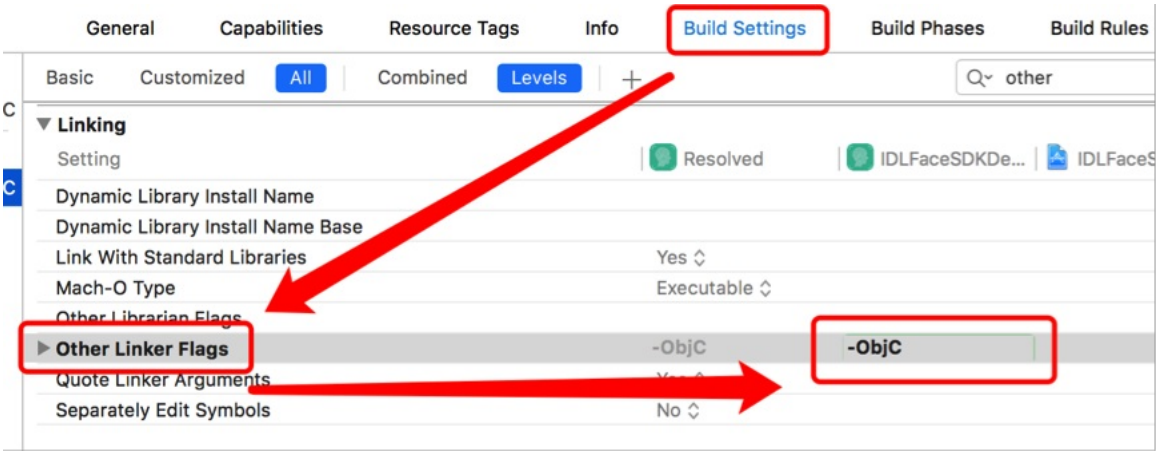
在FaceParameterConfig.h文件里面填写拼接好的FACE_LICENSE_ID。

```
//  (您申请的应用名称(appname)+「-face-ios」后缀，如申请的应用名称(appname)为test123，则此处填写test123-face-ios)
#define FACE_LICENSE_ID        @"BaiduFaceDemo-face-ios"
```

6. 选择链接C++标准库。



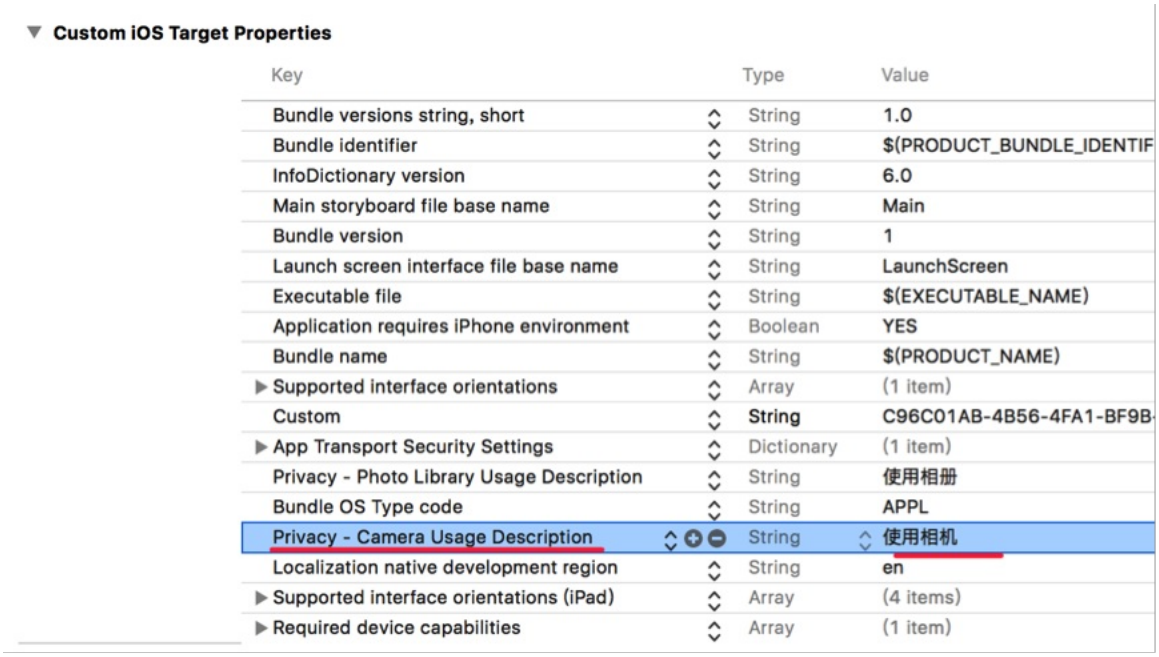
7. 如果没有使用pod管理第三方库的话，请在Build Setting >Linking > Other Linker Flags 上面加入 –ObjC 选项。如果用了pod请忽略，因为pod会自动添加上。



2.5 权限声明

需要使用相机权限：编辑Info.plist文件，添加

Privacy- Camera Usage Description 的Key值，Value为使用相机时候的提示语，可以填写：『使用相机』。



3、功能使用

3.1 活体识别

使用IDLFaceLivenessManager类的:

```
(void)livenessStratrgyWithImage:(UIImage*)image previewRect:(CGRect)previewRectdetectRect:(CGRect)detectRect completionHandler:(LivenessStrategyCompletion)completion
```

方法进行活体识别功能。使用方法见下图：

```
61 - (void)faceProcesss:(UIImage *)image {
62     if (self.hasFinished) {
63         return;
64     }
65     __weak typeof(self) weakSelf = self;
66     [[IDLFaceLivenessManager sharedInstance] livenessStratrgyWithImage:image
           previewRect:self.previewRect detectRect:self.detectRect completionHandler:^(NSDictionary
           *images, LivenessRemindCode remindCode) {
67         switch (remindCode) {
68             case LivenessRemindCodeOK: {
69                 weakSelf.hasFinished = YES;
70                 [self warningStatus:CommonStatus warning:@"非常好"];
71                 if (images[@"bestImage"] != nil && [images[@"bestImage"] count] != 0) {
72
73                     NSData* data = [[NSData alloc] initWithBase64EncodedString:[images[@"bestImage"]
                           lastObject] options:NSDataBase64DecodingIgnoreUnknownCharacters];
74                     UIImage* bestImage = [UIImage imageWithData:data];
75                     NSLog(@"bestImage = %@",bestImage);
76                 }
```

3.2 人脸采集

使用IDLFaceDetectionManager类的

```
(void)detectStratrgyWithImage:(UIImage*)image previewRect:(CGRect)previewRect  detectRect:(CGRect)detectRectcompletionHandler:(DetectStrategyCompletion)completion;
```

方法，来进行人脸采集。使用方法见下图：

```
- (void)faceProcesss:(UIImage *)image {
    if (self.hasFinished) {
        return;
    }

    __weak typeof(self) weakSelf = self;
    [[IDLFaceDetectionManager sharedInstance] detectStratrgyWithImage:image
        previewRect:self.previewRect detectRect:self.detectRect completionHandler:^(NSDictionary
        *images, DetectRemindCode remindCode) {
        switch (remindCode) {
            case DetectRemindCodeOK: {
                weakSelf.hasFinished = YES;
                [self warningStatus:CommonStatus warning:@"非常好"];
                if (images[@"bestImage"] != nil && [images[@"bestImage"] count] != 0) {
                    NSData* data = [[NSData alloc] initWithBase64EncodedString:[images[@"bestImage"]
                        lastObject] options:NSDataBase64DecodingIgnoreUnknownCharacters];
                    UIImage* bestImage = [UIImage imageWithData:data];
                    NSLog(@"bestImage = %@",bestImage);
                }
                dispatch_async(dispatch_get_main_queue(), ^{
                    [UIView animateWithDuration:0.5 animations:^{
                        weakSelf.animaView.alpha = 1;
```

3.3 质量校验设置

```
// 设置最小检测人脸阈值
[[FaceSDKManager sharedInstance] setMinFaceSize:200];
// 设置截取人脸图片大小
[[FaceSDKManager sharedInstance] setCropFaceSizeWidth:400];
// 设置人脸遮挡阀值
[[FaceSDKManager sharedInstance] setOccluThreshold:0.5];
// 设置亮度阀值
[[FaceSDKManager sharedInstance] setIllumThreshold:40];
// 设置图像模糊阀值
[[FaceSDKManager sharedInstance] setBlurThreshold:0.7];
// 设置头部姿态角度
[[FaceSDKManager sharedInstance] setEulurAngleThrPitch:10 yaw:10 roll:10];
// 设置是否进行人脸图片质量检测
[[FaceSDKManager sharedInstance] setIsCheckQuality:YES];
// 设置超时时间
[[FaceSDKManager sharedInstance] setConditionTimeout:10];
// 设置人脸检测精度阀值
[[FaceSDKManager sharedInstance] setNotFaceThreshold:0.6];
// 设置照片采集张数
[[FaceSDKManager sharedInstance] setMaxCropImageNum:1];
```

参数	名称	默认值	公安验证推荐值	取值范围
brightnessValue	图片爆光度	40f	100f	
blurnessValue	图像模糊度	0.5f	0.35f	0~1.0f
occlusionValue	人脸遮挡阈值	0.5f	0.3f	0~1.0f
headPitchValue	低头抬头角度	10	10	0~45
headYawValue	左右角度	10	10	0~45
headRollValue	偏头角度	10	10	0~45
cropFaceValue	裁剪图片大小	400	400	
minFaceSize	最小人脸检测值 小于此值的人脸将检测不出来。最小值为80.	200	200	
notFaceValue	人脸置信度	0.6f	0.6f	0~1.0f
isCheckFaceQuality	是否检测人脸质量	true	true	true/flase

3.4 界面定制说明

如果需要自定义UI界面，可以在FaceBaseViewController类的viewDidLoad方法里面进行修改。

3.5 接口设计说明

3.5.1 DetectRemindCode

类型名	DetectRemindCode	
名称	人脸检测返回类型	
所属头文件	IDLFaceDetectionManager.h	
类型说明	Enum 枚举类型	
枚举名	说明	值
DetectRemindCodeOK	正常	0
DetectRemindCodePitchOutofDownRange	头部偏低	1
DetectRemindCodePitchOutofUpRange	头部偏高	3
DetectRemindCodeYawOutofLeftRangeE	头部偏左	3
DetectRemindCodeYawOutofRightRange	头部偏右	4
DetectRemindCodePoorIllumination	光照不足	5
DetectRemindCodeNoFaceDetected	没有检测到人脸	6
DetectRemindCodeImageBlured	图像模糊	7
DetectRemindCodeOcclusionLeftEye	左眼有遮挡	8
DetectRemindCodeOcclusionRightEye	右眼有遮挡	9
DetectRemindCodeOcclusionNose	鼻子有遮挡	10
DetectRemindCodeOcclusionMouth	嘴巴有遮挡	11
DetectRemindCodeOcclusionLeftContour	左脸颊有遮挡	12
DetectRemindCodeOcclusionRightContour	右脸颊有遮挡	13
DetectRemindCodeOcclusionChinCoutour	下颚有遮挡	14
DetectRemindCodeTooClose	太近	15
DetectRemindCodeTooFar	太远	16
DetectRemindCodeBeyondPreviewFrame	出框	17
DetectRemindCodeVerifyInitError	鉴权失败	18
DetectRemindCodeVerifyDecryptError	鉴权失败	19
DetectRemindCodeVerifyInfoFormatError	鉴权失败	20
DetectRemindCodeVerifyExpired	鉴权失败	21
DetectRemindCodeVerifyMissRequiredInfo	鉴权失败	22
DetectRemindCodeVerifyInfoCheckError	鉴权失败	23
DetectRemindCodeVerifyLocalFileError	鉴权失败	24
DetectRemindCodeVerifyRemoteDataError	鉴权失败	25
DetectRemindCodeTimeout	超时	26
DetectRemindCodeConditionMeet	条件满足	27

3.5.2 FaceLivenessActionType

类型名	FaceLivenessActionType	
名称	活体检测命令类型	
所属头文件	FaceSDKManager.h	
类型说明	Enum 枚举类型	
枚举名	说明	值
FaceLivenessActionTypeLiveEye	眨眨眼	0
FaceLivenessActionTypeLiveMouth	张张嘴	1
FaceLivenessActionTypeLiveYawRight	向右摇头	2
FaceLivenessActionTypeLiveYawLeft	向左摇头	3
FaceLivenessActionTypeLivePitchUp	抬头	4
FaceLivenessActionTypeLivePitchDown	低头	5
FaceLivenessActionTypeLiveYaw	摇头	6
FaceLivenessActionTypeNoAction	没有动作	7

3.5.3 LivenessRemindCode

类型名	LivenessRemindCode	
名称	活体检测提醒类型	
所属头文件	IDLFaceLivenessManager.h	
类型说明	Enum 枚举类型	
枚举名	**说明**	**值**
LivenessRemindCodeOK	正常	0
LivenessRemindCodePitchOutofDownRange	头部偏低	1
LivenessRemindCodePitchOutofUpRange	头部偏高	2
LivenessRemindCodeYawOutofLeftRange	头部偏左	3
LivenessRemindCodeYawOutofRightRange	头部偏右	4
LivenessRemindCodePoorIllumination	光照不足	5
LivenessRemindCodeNoFaceDetected	没有检测到人脸	6
LivenessRemindCodeImageBlured	图像模糊	7
LivenessRemindCodeOcclusionLeftEye	左眼有遮挡	8
LivenessRemindCodeOcclusionRightEye	右眼有遮挡	9
LivenessRemindCodeOcclusionNose	鼻子有遮挡	10
LivenessRemindCodeOcclusionMouth	嘴巴有遮挡	11
LivenessRemindCodeOcclusionLeftContour	左脸颊有遮挡	12
LivenessRemindCodeOcclusionRightContour	右脸颊有遮挡	13
LivenessRemindCodeOcclusionChinCoutour	下颚有遮挡	14
LivenessRemindCodeTooClose	太近	15
LivenessRemindCodeTooFar	太远	16
LivenessRemindCodeBeyondPreviewFrame	出框	17
LivenessRemindCodeLiveEye	眨眨眼	18
LivenessRemindCodeLiveMouth	张大嘴	19

LivenessRemindCodeLiveYawRight	向左摇头	20
LivenessRemindCodeLiveYawLeft	向右摇头	21
LivenessRemindCodeLivePitchUp	向上抬头	22
LivenessRemindCodeLivePitchDown	向下低头	23
LivenessRemindCodeLiveYaw	摇摇头	24
LivenessRemindCodeSingleLivenessFinished	完成一个活体动作	25
LivenessRemindCodeVerifyInitError	鉴权失败	26
LivenessRemindCodeVerifyDecryptError	鉴权失败	27
LivenessRemindCodeVerifyInfoFormatError	鉴权失败	28
LivenessRemindCodeVerifyExpired	鉴权失败	29
LivenessRemindCodeVerifyMissRequiredInfo	鉴权失败	30
LivenessRemindCodeVerifyInfoCheckError	鉴权失败	31
LivenessRemindCodeVerifyLocalFileError	鉴权失败	32
LivenessRemindCodeVerifyRemoteDataError	鉴权失败	33
LivenessRemindCodeTimeout	超时	34
LivenessRemindCodeConditionMeet	条件满足	35

3.6 创建实例

- 方法：FaceSDKManager.instance
- 参数：无
- 返回：人脸功能管理器
- 说明：创建人脸功能管理器。

3.7 设置人脸功能控制参数

- 方法

```
  // 设置鉴权
(void)setLicenseID:(NSString )licenseID andLocalLicenceFile:(NSString)localLicencePath;
// 最小检测人脸阈值
(void)setMinFaceSize:(NSInteger)width;
// 截取人脸图片大小
(void)setCropFaceSizeWidth:(CGFloat)width;
// 人脸检测精度阈值
(void)setNotFaceThreshold:(CGFloat)th;
// 人脸遮挡阈值
(void)setOccluThreshold:(CGFloat)thr;
// 亮度阈值
(void)setIllumThreshold:(NSInteger)thr;
// 图像模糊阈值
(void)setBlurThreshold:(CGFloat)thr;
// 头部姿态角度
(void)setEulurAngleThrPitch:(NSInteger)pitch yaw:(NSInteger)yawroll:(NSInteger)roll;
// 是否进行人脸图片质量检测
(void)setIsCheckQuality:(BOOL)isCheck;
```

3.8 取得人脸图像采集功能接口

- 方法

```
(void)detectWithImage:(UIImage *)image completion: (void (^)(FaceInfo *faceinfo, ResultCode resultCode))completion;
```

- 参数

image: 采集到的图像

- 返回

faceinfo: FaceInfo 人脸检测信息

resultCode: ResultCode 返回执行结果码

- 说明

取得人脸图像采集功能接口。人脸图像采集接口完成，解析图片人脸信息，返回检测结果。

3.9 取得活体检测功能接口

- 方法

```
(void)livenessWithImage:(UIImage )image completion:(void (^)(FaceInfo faceinfo, LivenessState* state, ResultCode resultCode))completion;
```

- 参数

image: 采集到的图像

liveAction: 活体动作类型

- 返回

aceinfo: FaceInfo 人脸检测信息

livenessState:LivenessState 活体状态码

resultCode: ResultCode 返回执行结果码

- 说明

取得活体检测功能接口。活体检测功能接口完成，解析图片人脸信息，返回活体检测结果。

3.10 设置人脸图像采集功能参数

- 方法

```
(void)detectStratrgyWithImage:(UIImage *)image previewRect:(CGRect)previewRectdetectRect:(CGRect)detectRectcompletionHandler:(DetectStrategyCompletion)completion;
```

- 参数

previewRect 人脸图片大小，类型：Rect

detectRect 人脸检测区域大小，类型：Rect

completion 完成后返回照片和状态结果

- 说明

检测图片中的人脸信息，完成人脸图像采集，返回检测状态和结果。

3.11 设置人脸功能控制参数

- 方法

```
(void)livenessStratrgyWithImage:(UIImage *)imagepreviewRect:(CGRect)previewRect detectRect:
(CGRect)detectRectcompletionHandler:(LivenessStrategyCompletion)completion;
```

- 参数

previewRect 人脸图片大小，类型：Rect

detectRect 人脸检测区域大小，类型：Rect

completion 完成后返回照片和状态结果

- 说明

检测图片中的人脸信息，完成人脸图像采集和人脸活体检测，返回检测状态和结果。

3.12 活体动作设置

- 方法

```
(void)livenesswithList:(NSArray *)array order:(BOOL)ordernumberOfLiveness:(NSInteger)numberOfLiveness
```

- 参数

array: 活体动作列表

order: 是否按顺序进行活体动作

numberOfLiveness: 活体动作数目（array为nil是起作用）

- 返回

无

- 说明

活体动作设置

4、常见问题

1、license文件有什么用，该放在什么地方？

license文件需要申请，目的是作为sdk校验开发者的使用合法性，license文件放置位置 不对或未放置license文件会导致没法使用sdk，一般应先申请license文件，并把申请得到的license文件，放置在assets目录下面。****

2、活体检测常见有那些动作？是否可配置？

常见有6个动作，眨眼，张大嘴，向上抬头，向下低头，向左摇头，向右摇头等。 sdk提供FaceConfig参数设置类，如活体检测角度、光线，检测动作，检测动作数量等设置。

3、license文件失效了，不能用了怎么办？

License文件申请时候有期限，如过期会导致校验失效，需要在后台进行申请延期。

windows版本

windows-基础版

> 温馨提示：2020年4月离线采集SDK Win&Linux版本正式下线，若您需要此版本请到离线识别SDK进行申请；老客户转正式版请提交工单申请。

目录

1 基础信息
1.1 产品概述
1.2 规格信息
1.3 兼容性
2 业务介绍
2.1 功能简介
2.2 业务流程
3 快速集成
3.1 名词介绍
3.2 SDK文件结构
3.3 授权激活
3.4 运行Demo工程
4 接口说明
4.1 人脸检测track接口（传入图片）
4.2 人脸检测track接口（传入二进制图片buffer）
4.3 人脸检测track_max_face接口
4.4 人脸检测track_max_face接口（传入二进制图片buffer）
4.5 人脸检测设置接口
4.6 USB摄像头检测（实时视频帧人脸检测）
4.7 人脸检测track接口（传入opencv的mat）
4.8 人脸质量接口（通过传入图片）
5 错误码及错误信息
6 常见问题

1、基础信息

1.1 产品概述

百度FaceSDK Windows基础版是一种面向Windows设备的人脸技术开发包，此版SDK包含人脸检测追踪等方法。基于该方案，开发者可以轻松的构建包含人脸检测、采集的应用。在您使用SDK之前，我们首先为您介绍一下SDK本身及相关人脸能力，以便您能更方便使用人脸服务。

1.2 规格信息

- 包大小：~500M（相关库文件较大）
- 最小人脸检测大小：50px * 50px
- 可识别人脸角度：yaw ≤ ±30°, pitch ≤ ±30°
- 检测速度：100ms 720p*
- 追踪速度：30ms 720p*
- 人脸检测耗时：< 100ms

> 备注：以上指标，由最新版SDK运行在真实设备上，采用真实数据集所得，但**算法性能受实际运行设备、实际数据集等情况影响**，以上数字仅供参考。

1.3 兼容性

- x86、x64两个版本，支持Win7、Win10

- 推荐基于vs2015进行开发

2、业务介绍

2.1 功能简介

2.1.1 人脸检测与跟踪

可在设备端，离线实时检测视频流中的人脸。同时支持处理静态图片或者视频流，并对当前检测到的人脸持续跟踪，动态定位人脸轮廓，稳定贴合人脸。

2.1.2 质量控制

在人脸检测及追踪过程中，实时校验人脸的姿态角度、遮挡、清晰度、光照条件，符合质量条件的人脸图片才会被采集。

2.1.3 人脸采集

针对视频流实时完成人脸图片采集，并输出满足质量过滤条件的人脸图片，可自定义采集人脸大小，采集频率，采集质量等设置。

2.2 业务流程

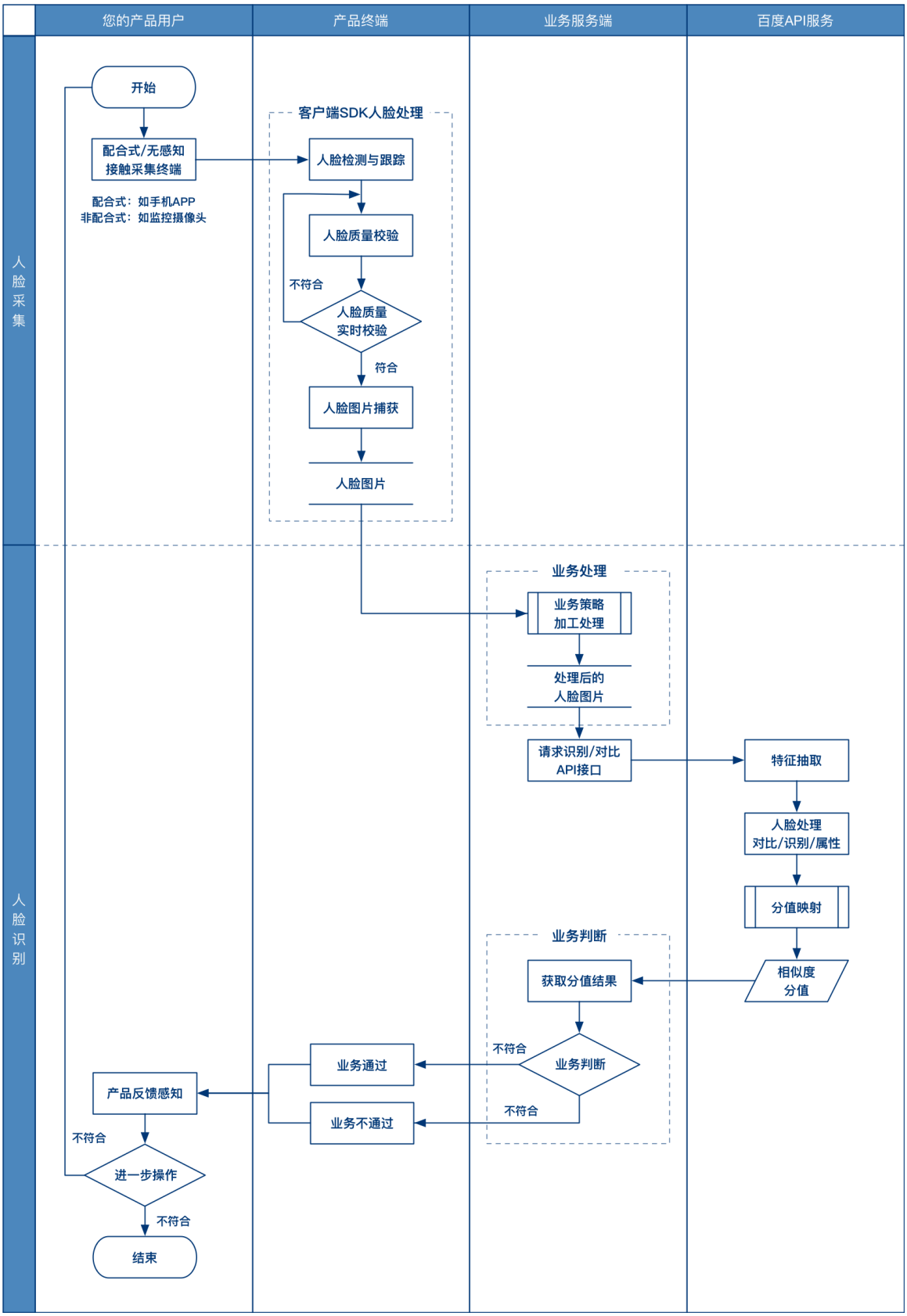
人脸识别的应用场景，核心可以分为三大类：

1. **身份核验**：即1:1对比，判断两张脸的相似度，判断`你是你`，通常用于需要验证用户身份真实性的场景，如人证对比。
2. **身份识别**：即1:N识别，在一个人脸集合中找到最相似的人脸，判断`你是谁`，通常用于判断用户身份是否存在，及身份信息内容的场景，如人脸门禁、人脸支付等。
3. **属性分析**：即人脸属性分析，基于人脸信息，返回年龄、性别等属性值，通常用于客群分析、娱乐营销等场景，如统计线下客群年龄分布。

而以上场景的几乎所有业务过程，核心可以分为两个步骤：

1. **人脸采集**：人脸识别的前置步骤，即获取到人脸图片，用于对比、识别、属性分析等操作。
2. **人脸分析**：包括人脸图片的加工处理，特征抽取与对比，结果返回等一系列操作，也是通常理解为的人脸识别操作。

要想确保人脸识别的应用效果得到保障，最为核心的一个环节即人脸的获取，即**人脸采集**。目前市面上所有人脸识别应用落地，面临的主要问题就是**应用环境复杂**，包括光照、遮挡、作弊攻击等一系列环境因素干扰，涉及产品策略、硬件选型、施工方案等多个维度地综合作用，才能不断提升最终效果。



3、快速集成

3.1 名词介绍

名词	定义
采集SDK	Windows离线人脸识别SDK
vs2015	微软的开发工具visual studio 2015 (推荐安装vs community 2015)
License	人脸识别激活所需要的激活文件,可利用激活工具生成
License_key	创建授权时需要传入的授权标识字段

3.2 SDK文件结构

SDK提供动态库BaiduFaceApi.dll、BaiduFaceApi.lib及头文件face.h。另外有附带的demo工程BaiduFaceCollect以及鉴权激活工具LicenseTool.exe。

在此之外，还附带支撑SDK使用的人脸识别模型文件夹face-resource，该文件夹在BaiduFaceCollect示例工程里面，存放路径请参考BaiduFaceCollect工程，默认存放路径为您要开发的exe（如demo工程BaiduFaceCollect.exe）所在路径的上级目录下。若存放不对，可能会影响sdk正常使用，另外请勿删除该文件夹。

为运行exe，需要用到一些底层的库文件（dll），相关库文件在BaiduFaceCollect的win32和x64文件夹（分别对应x86和x64平台）中均已提供。此外，随BaiduFaceCollect还有一些辅助文件或开源库，如opencv及头文件，base64文件，这些都是开源文件，随demo工程根据用户需要可自行修改，另外就是一些示例demo文件。

3.3 授权激活

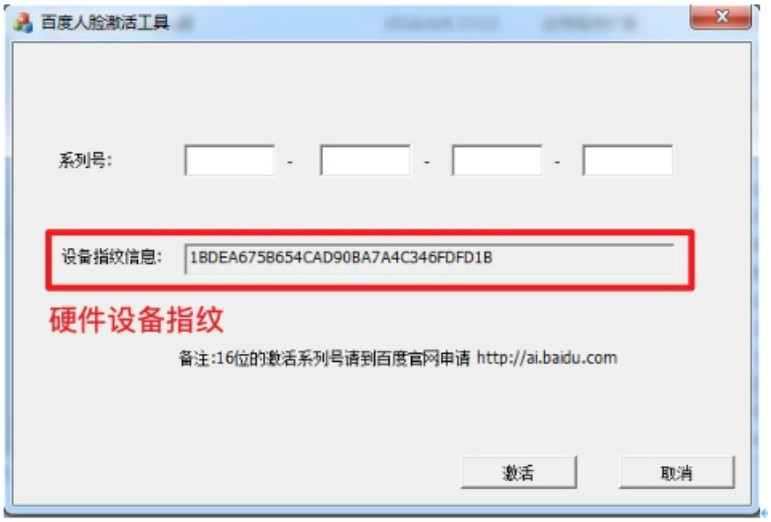
激活分测试阶段**按设备单机授权**、和**按产品线授权**。

- **按设备单机授权**：调试阶段的一种授权方式，主要用于指定设备上反复测试程序功能，激活状态只适用于授权的设备硬件。
- **按产品线授权**：针对发布的exe，该exe可应用于所有设备。

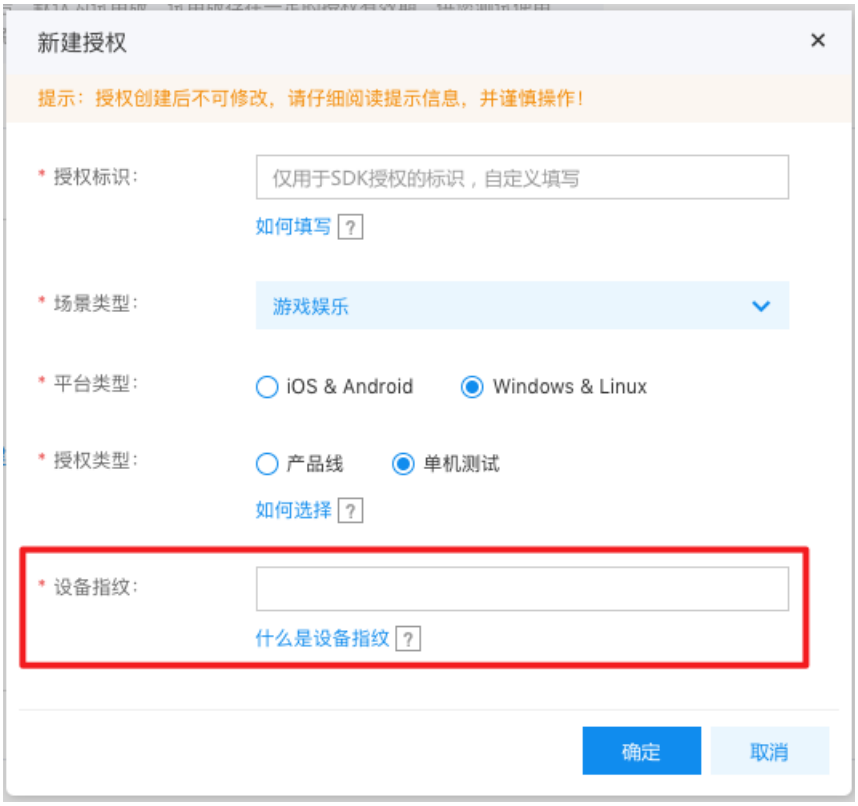
3.3.1 按设备单机授权

该方式授权，主要用于在测试开发阶段，在测试机器上反复调试编译，所以提供按照设备维度的授权。

第一步：SDK附带一个LicenseTool.exe，双击可运行，弹出如图所示的设备指纹信息。



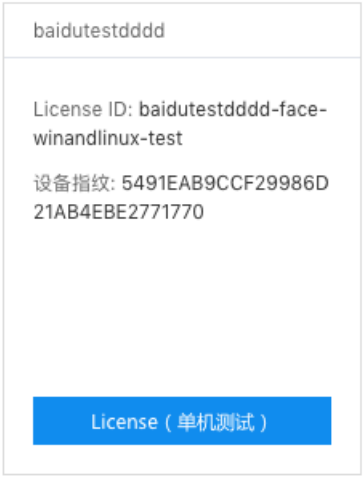
第二步：复制此硬件指纹信息，在后台填写到下图所在的位置，并创建授权，下载对应的License文件。



第三步（重要！！）：在SDK中，您需要在初始化的时候传入license_key，此license_key为您在后台填写的「授权标识」，添加一个-face-winadnlinux-test的后缀，如下图所示，**授权标识填写的为「baidutest」，则license_key为：baidutest-face-winadnlinux-test。**


```
 // 传入license_key，牢记在后台填写的「授权标识」，并在此授权标识基础上添加 -face-winadnlinux-test 的后缀
// 如您填写的授权标识填写的为「baidutest」，则license_key为：baidutest-face-winadnlinux-test
std::string license_key = "baidutest-face-winadnlinux-test";

//初始化sdk
sdk_init(license_key.c_str());
```

第四步：在官网下载License文件，如下图所示：



需要重命名为：license.ini，并**拷贝到exe所在路径同一目录位置**，如SDK的win32或x64目录下。

以上四步完毕后，您的exe就可以通过授权激活运行。

3.3.2 按产品线授权

该方式授权，主要用于发布正式的exe安装包，授权将会识别具体的exe程序，确保指定exe的正常运行。

> **注**：所以每次程序升级，需要单独创建一个License文件进行更新。

第一步：在后台新建授权中表单中，平台类型选择「Windows & Linux」，授权类型选择「产品线」，并填写以下几个内容：

1. 授权标识：自定义用于标识您的license授权信息，每个授权只能有唯一的标识，支持英文、数字、横线，20个英文字符以内；
2. 产品名：填写exe可执行程序的具体名称，例如face.exe，则产品名为face.exe；
3. 程序文件的md5：**此项稍后填写**，exe文件的md5：如face.exe，请填写此程序的md5值

第二步：在SDK初始化的时候传入license_key，此license_key为您在后台填写的「授权标识」，添加一个-face-win的后缀，如下图所示，**授权标识填写的为「baidutest」，则license_key为：baidutest-face-win。**

```
// 传入license_key，牢记在后台填写的「授权标识」，并在此授权标识基础上添加 -face-win 的后缀
// 如您填写的授权标识填写的为「baidutest」，则license_key为：baidutest-face-win
std::string license_key = "baidutest-face-win";

//初始化sdk
sdk_init(license_key.c_str());
```

第三步：完成License代码配置后，再编译生成exe文件，用md5命令或工具生成该exe的md5，并填写到第一步的表单中，提交表单，并下载生成的License文件。

> **注**：exe一旦编译后，则license_key和exe不能再编译改变，否则该license文件无法通过激活。

第四步：若按上述步骤生成license文件后，重命名为:license.ini，并拷贝到exe所在路径同一目录位置，如SDK的win32或x64目录下。则可运行exe通过授权激活。

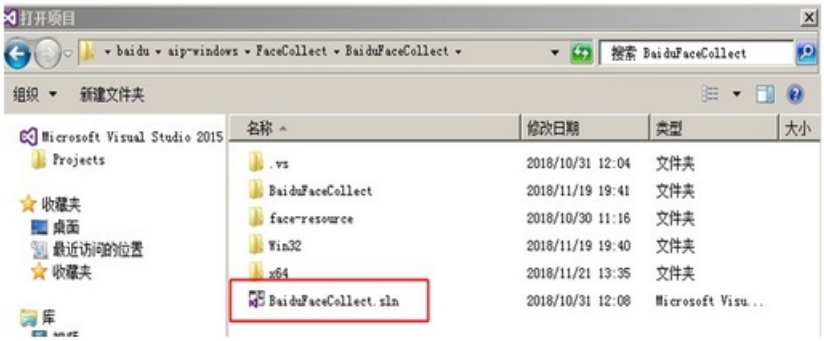
3.4 运行Demo工程

Demo示例工程BaiduFaceCollect展示了如何集成百度人脸识别采集SDK。即导入**BaiduFaceApi.dll**，**BaiduFaceApi.lib**及头文件**face.h**。另外为了示例视频人脸跟踪等，用到了一些opencv的库文件以及一些实现demo的支持文件，如image_base64等。

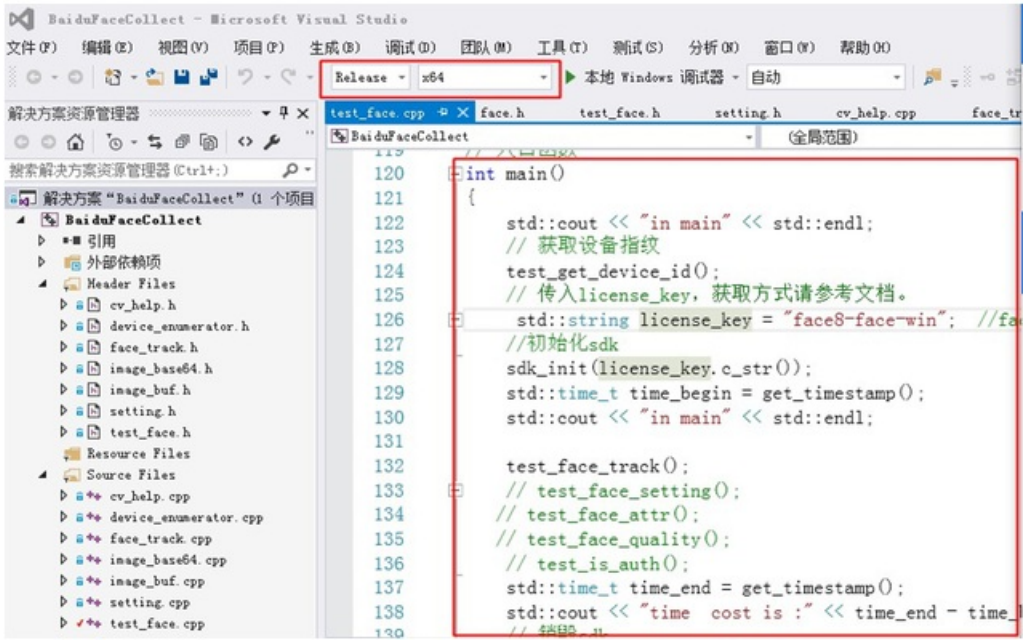
> **注**：这些支持文件均为代码开源或是开源库

在BaiduFaceCollect中的test_face.cpp的main()方法中，有使用sdk的各个接口方法示例。可通过vs2015双击打开如下图的sln文

件。



后工程展开为（入口函数在test_face.cpp的main方法，请编译release版本的工程，x64或x86）：



SDK实现的主要功能有人脸实时跟踪检测、人脸属性、人脸质量，返回识别出的人脸信息等，另外支持对人脸检测进行设置，达到根据设置进行识别的目的。各接口功能及传入参数和返回结果（返回结果一般为json格式的字符串）详见下方接口说明。

4、接口说明

4.1 人脸检测track接口（传入图片）

方法名

`track`

方法说明

人脸检测，返回人脸信息

请求信息

参数	必须	类型	描述
out	是	std::vector	**返回的检测到的人脸图片信息结构体** std::vector *& out 通过传入TrackFaceInfo的结构体指针引用，在检测到人脸信息后，用该引用返回人脸信息。TrackedFaceInfo结构体为检测到的人脸信息，其中：box为检测到的人脸框（为FaceInfo结构体,包含人脸框大小，中心点坐标，人脸宽度等信息）landmarks为检测到的人脸关键点，一般为144个。score为人脸打分headPose的向量组为人脸x,y,z的三个角度
image	是	string	人脸图片信息，根据image_type，传入图片内容
img_type	是	int	传入的图片类型。为1时候表示base64编码的图片，为2时候表示传入图片的本地路径。BASE64：图片的base64值；FACE_FILE：图片的本地文件路径地址；
maxTrackObjNum	是	int	最多检测人脸数量，默认为1，最大不超过5

返回信息

返回结果为int，表示检测到几个人脸信息，1为1个人，0为没检测到人脸。

4.2 人脸检测track接口（传入二进制图片buffer）

方法名

`track_by_buf`

方法说明

人脸检测，返回人脸信息

请求信息

参数	必须	类型	描述
out	是	std::vector	**返回的检测到的人脸图片信息结构体** std::vector *& out 通过传入TrackFaceInfo的结构体指针引用，在检测到人脸信息后，用该引用返回人脸信息。TrackedFaceInfo结构体为检测到的人脸信息，其中：box为检测到的人脸框（为FaceInfo结构体,包含人脸框大小，中心点坐标，人脸宽度等信息）landmarks为检测到的人脸关键点，一般为144个。score为人脸打分headPose的向量组为人脸x,y,z的三个角度
image	是	Unsigned char *	人脸图片信息，二进制图片buffer内容
size	是	int	二进制图片的大小
maxTrackObjNum	是	int	最多检测人脸数量，默认为1，最大不超过5

返回信息

返回结果为int，表示检测到几个人脸信息，1为1个人，0为没检测到人脸。

4.3 人脸检测track_max_face接口

方法名

`track_max_face`

方法说明

检测最大人脸。

请求信息

参数	必须	类型	描述
out	是	std::vector<TrackFaceInfo>	**返回的检测到的人脸图片信息结构体** std::vector *& out 通过传入TrackFaceInfo的结构体指针引用，在检测到人脸信息后，用该引用返回人脸信息。TrackedFaceInfo结构体为检测到的人脸信息，其中：box为检测到的人脸框（为FaceInfo结构体,包含人脸框大小，中心点坐标，人脸宽度等信息）；landmarks为检测到的人脸关键点，一般为144个；score为人脸打分；headPose的向量组为人脸x,y,z的三个角度
image	是	string	图片信息（数据大小应小于10M）
image_type	是	int	为1时候表示base64编码的图片；为2时候表示传入图片的本地路径。 BASE64：图片的base64值；FACE_FILE：图片的本地文件路径地址；

返回信息

返回结果为int，表示检测到几个人脸信息，1为1个人，0为没检测到人脸。

4.4 人脸检测track_max_face接口（传入二进制图片buffer）

方法名

`track_max_face_by_buf`

方法说明

检测最大人脸。

请求信息

参数	必须	类型	描述
out	是	std::vector<TrackFaceInfo>	**返回的检测到的人脸图片信息结构体** std::vector *& out 通过传入TrackFaceInfo的结构体指针引用，在检测到人脸信息后，用该引用返回人脸信息。TrackedFaceInfo结构体为检测到的人脸信息，其中：box为检测到的人脸框（为FaceInfo结构体,包含人脸框大小，中心点坐标，人脸宽度等信息）；landmarks为检测到的人脸关键点，一般为144个；score为人脸打分；headPose的向量组为人脸x,y,z的三个角度
image	是	Unsigned char *	人脸图片信息，二进制图片buffer内容
size	是	int	二进制图片的大小

返回信息

返回结果为int，表示检测到几个人脸信息，1为1个人，0为没检测到人脸。

4.5 人脸检测设置接口

请参考`TestFaceApi`工程中的代码示例及头文件`BaiduFaceApi.h`中的定义及`setting.cpp`里的代码注释。

4.6 USB摄像头检测（实时视频帧人脸检测）

通过打开usb摄像头，返回实时检测到的视频帧人脸信息。请参考`TestFaceApi`中的示例`liveness.cpp`中的`usb_track_face_info`，该方法中用到了人脸检测track视频帧，接口如下4.5：

4.7 人脸检测track接口（传入opencv的mat）

方法名

`Track`

方法说明

人脸检测，返回人脸信息。

请求信息

参数	必须	类型	描述
out	是	`std::vector<TrackFaceInfo>`	**返回的检测到的人脸图片信息结构体** std::vector *& out 通过传入TrackFaceInfo的结构体指针引用，在检测到人脸信息后，用该引用返回人脸信息。TrackedFaceInfo结构体为检测到的人脸信息，其中：box为检测到的人脸框（为FaceInfo结构体,包含人脸框大小，中心点坐标，人脸宽度等信息）；landmarks为检测到的人脸关键点，一般为144个；score为人脸打分；headPose的向量组为人脸x,y,z的三个角度
Mat	是	Opencv格式的单帧图片mat	人脸图片信息
maxTrackObjNum	是	int	最多检测人脸数量。默认为1，最大不超过5

返回信息

返回结果为TrackedFaceInfo的向量指针,向量组为0时候表示没检测到人脸。TrackedFaceInfo结构体为检测到的人脸信息，其中：box为检测到的人脸框（为FaceInfo结构体,包含人脸框大小，中心点坐标，人脸宽度等信息）；landmarks为检测到的人脸关键点，一般为144个；score为人脸打分；headPose的向量组为人脸x,y,z的三个角度。

4.8 人脸质量接口（通过传入图片）

方法名

`face_quality`

方法说明

人脸质量检测接口

请求信息

参数	必须	类型	示例描述
image	是	string	需要检测的人脸图片，小于10M, 图片类型根据image_type参数定
imgae_type	是	int	图片类型，必选择以下2种形式之一。 Image_type为1代表BASE64为2代表FACE_FILE。 **BASE64**：图片的base64值； **FACE_FILE**：图片的本地文件路径地址；

返回信息

errno信息：

- errno=0：Success
- errno>0：错误码对应的详细msg

data信息:

字段	类型	说明
bluriness	float	模糊度
illum	float	光照
occl_l_eye	float	左眼遮挡度
occl_r_eye	float	右眼遮挡度
occl_nose	float	鼻子遮挡度
occl_mouth	float	嘴巴遮挡度
occl_l_contour	float	左脸遮挡度
occl_r_contour	float	右脸遮挡度
occl_chin	float	下巴遮挡度

5、错误码及错误信息

错误码	错误内容	错误描述
0	SUCCESS	成功
1	SYSTEM ERROR	系统错误
2	UNKNOWN ERROR	未知错误
1001	NOT_AUTH	授权校验失败
1002	REQUEST PARAMS ERROR	请求参数错误
1003	DB_OP_FAILED	数据库操作失败
1004	NO_DATA	没有数据
1005	RECORD_UNEXIST	记录不存在
1006	RECORD_ALREADY_EXIST	记录已经存在
1007	FILE_NOT_EXIST	文件不存在
1008	GET_FEATURE_FAIL	提取特征值失败
1009	FILE_TOO_BIG	文件太大
1010	FACE_RESOURCE_NOT_EXIST	人脸资源文件不存在
1011	FEATURE_LEN_ERROR	特征值长度错误
1012	DETECT_NO_FACE	未检测到人脸

6、常见问题

Q：编译demo工程时候出现如opencv-win\include\opencv2\flann\logger.h(66): error C4996: 'fopen': This function or variable may be unsafe的错误提示

A：这种错误一般是因为vs强制要求安全等级提高所致，可以通过右键工程-属性-C/C++ -- 预处理器—预处理器定义中加入：`_CRT_SECURE_NO_WARNINGS`后重新编译即可解决问题。

Q：工程运行过程中，提示face-resource不存在

A：此SDK需要一些模型文件，在demo工程的`face-resource`文件夹中，该文件夹需要放置在exe所在路径的上级目录下。若放置不正确，可能出现找不到模型文件，没法进行人脸识别。

Q：exe不能运行或者崩溃

A：编译的工程exe，需要和dll等在一个文件夹里面，如示例工程win32和x64就有不少动态库dll文件，这是exe运行所需的库文件，不放在一起的话，可能exe不能运行或者崩溃。

Q：激活后是否可以把激活文件license.ini拷贝到其他设备运行？

A：如果是按设备激活的，则license.ini和设备绑定，只能用于该设备测试，若是按产品方式激活，则该激活文件可随产品可执行文件一起拷贝到任意设备上运行。

Q：激活工具license.exe运行不起来，还有其他方法获取测试用的设备指纹吗？

A：直接运行SDK的exe如BaiduFaceCollect.exe，查看console的输出页面device id:字样后面的串，该串即设备指纹。

Linux版本

linux-基础版

> 温馨提示：2020年4月离线采集SDK Win&Linux版本正式下线，若您需要此版本请到离线识别SDK进行申请；老客户转正式版请提交工单申请。

目录

1 基础信息
1.1 产品概述
1.2 规格信息
1.3 兼容性
2 业务介绍
2.1 功能简介
2.2 业务流程
3 快速集成
3.1 名词介绍
3.2 SDK文件结构
3.3 授权激活
3.4 运行Demo工程
4 接口说明
4.1 人脸检测track接口（传入图片）
4.2 人脸检测track接口（传入二进制图片buffer）
4.3 人脸检测track_max_face接口
4.4 人脸检测track_max_face接口（传入二进制图片buffer）
4.5 人脸检测设置接口
4.6 USB摄像头检测（实时视频帧人脸检测）
4.7 人脸检测track接口（传入opencv的mat）
4.8 人脸质量接口（通过传入图片）
5 错误码及错误信息
6 常见问题

1、基础信息

1.1 产品概述

百度FaceSDK Linux基础版是一种面向Linux设备的人脸技术开发包，此版SDK包含人脸检测追踪等方法。基于该方案，开发者可以轻松的构建包含人脸检测、采集的应用。在您使用SDK之前，我们首先为您介绍一下SDK本身及相关人脸能力，以便您能更方便使用人脸服务。

1.2 规格信息

- 包大小：~500M（相关库文件较大）
- 最小人脸检测大小：50px * 50px
- 可识别人脸角度：yaw ≤ ±30°, pitch ≤ ±30°
- 检测速度：100ms 720p*
- 追踪速度：30ms 720p*

- 人脸检测耗时：< 100ms

> 备注：以上指标，由最新版SDK运行在真实设备上，采用真实数据集所得，但**算法性能受实际运行设备、实际数据集等情况影响**，以上数字仅供参考。

1.3 兼容性

- Centos6.3 gcc4.8.2 以及 Ubuntu16.04 gcc5.4.0上编译

2、业务介绍

2.1 功能简介

2.1.1 人脸检测与跟踪

可在设备端，离线实时检测视频流中的人脸。同时支持处理静态图片或者视频流，并对当前检测到的人脸持续跟踪，动态定位人脸轮廓，稳定贴合人脸。

2.1.2 质量控制

在人脸检测及追踪过程中，实时校验人脸的姿态角度、遮挡、清晰度、光照条件，符合质量条件的人脸图片才会被采集。

2.1.3 人脸采集

针对视频流实时完成人脸图片采集，并输出满足质量过滤条件的人脸图片，可自定义采集人脸大小，采集频率，采集质量等设置。

2.2 业务流程

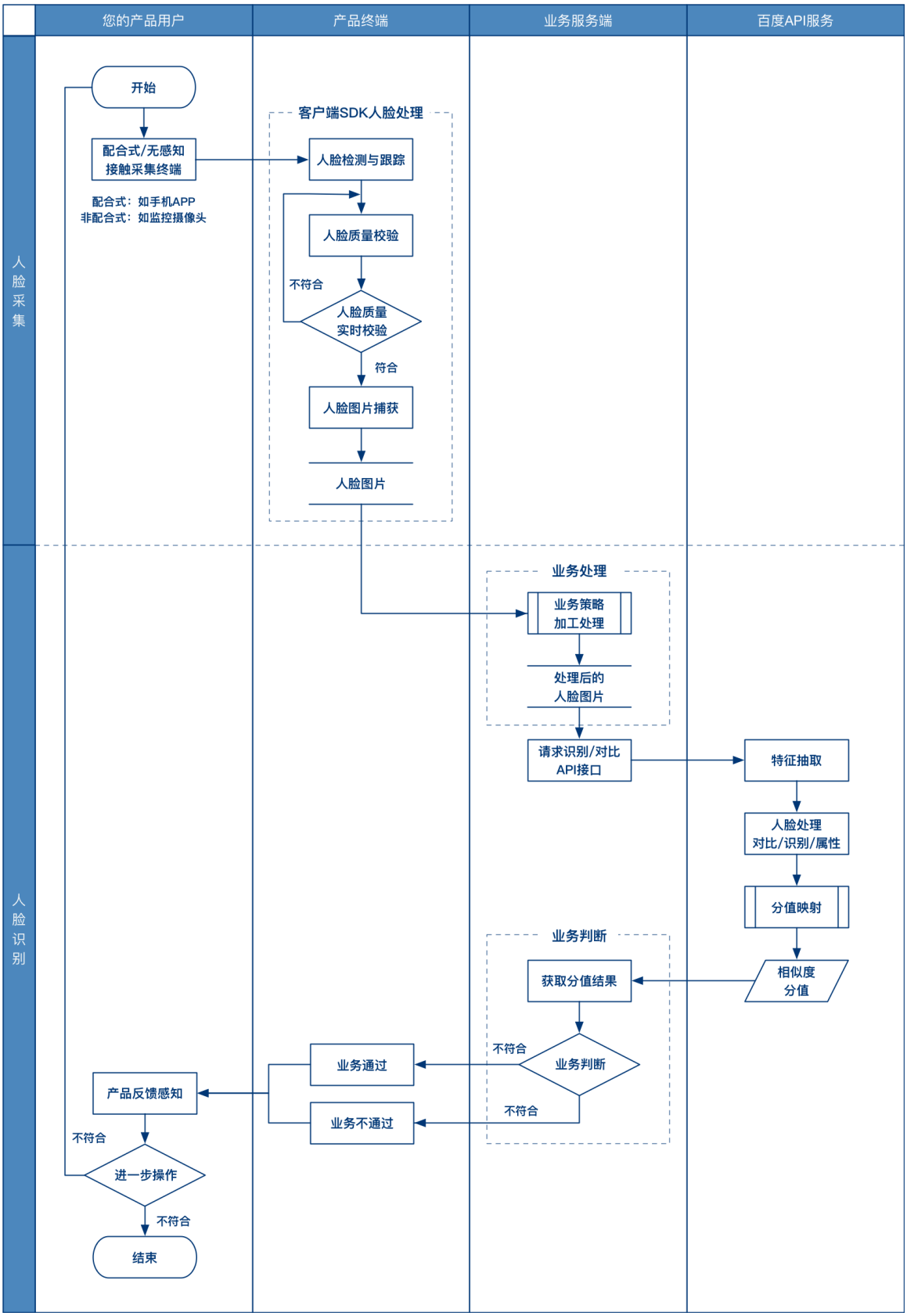
人脸识别的应用场景，核心可以分为三大类：

1. **身份核验**：即1:1对比，判断两张脸的相似度，判断`你是你`，通常用于需要验证用户身份真实性的场景，如人证对比。
2. **身份识别**：即1:N识别，在一个人脸集合中找到最相似的人脸，判断`你是谁`，通常用于判断用户身份是否存在，及身份信息内容的场景，如人脸门禁、人脸支付等。
3. **属性分析**：即人脸属性分析，基于人脸信息，返回年龄、性别等属性值，通常用于客群分析、娱乐营销等场景，如统计线下客群年龄分布。

而以上场景的几乎所有业务过程，核心可以分为两个步骤：

1. **人脸采集**：人脸识别的前置步骤，即获取到人脸图片，用于对比、识别、属性分析等操作。
2. **人脸分析**：包括人脸图片的加工处理，特征抽取与对比，结果返回等一系列操作，也是通常理解为的人脸识别操作。

要想确保人脸识别的应用效果得到保障，最为核心的一个环节即人脸的获取，即**人脸采集**。目前市面上所有人脸识别应用落地，面临的主要问题就是**应用环境复杂**，包括光照、遮挡、作弊攻击等一系列环境因素干扰，涉及产品策略、硬件选型、施工方案等多个维度地综合作用，才能不断提升最终效果。



3、快速集成

3.1 名词介绍

名词	定义
采集SDK	Windows离线人脸识别SDK
gcc	Linux平台c++编译器
License	人脸识别激活所需要的激活文件,可利用激活工具生成
License_key	创建授权时需要传入的授权标识字段
device_id	设备指纹，产品激活可按产品线批量授权，也可按设备指纹只授权一个设备

3.2 SDK文件结构

Sdk提供动态库`libBaiduFaceApi.so`（centos和ubuntu的so文件名不同，请参考SDK）及头文件`face.h`。

在此之外，还附带支撑SDK使用的人脸识别模型文件夹`face-resource`，该文件夹在BaiduFaceCollect示例工程里面，存放路径请参考BaiduFaceCollect工程。默认存放路径为您要开发的exe可执行文件路径同一目录下。若存放不对，可能会影响sdk正常使用，另外请勿删除该文件夹。

为运行您开发的可执行文件，需要用到一些底层的库文件支持，相关库文件在baidu-`facecollect`的`lib3`目录中（主要有opencv库，ffmpeg库，json，curl库等）。此外，随工程还有一些编译用的Makefile文件及sh脚本文件等，可通过Makefile编译工程，sh脚本文件运行编译的可执行文件。Makefile文件分别命名为`Makefile_centos6.3`及`Makefile_ubuntu`，若要编译相应平台版本，请将其重命名为Makefile。

3.3 授权激活

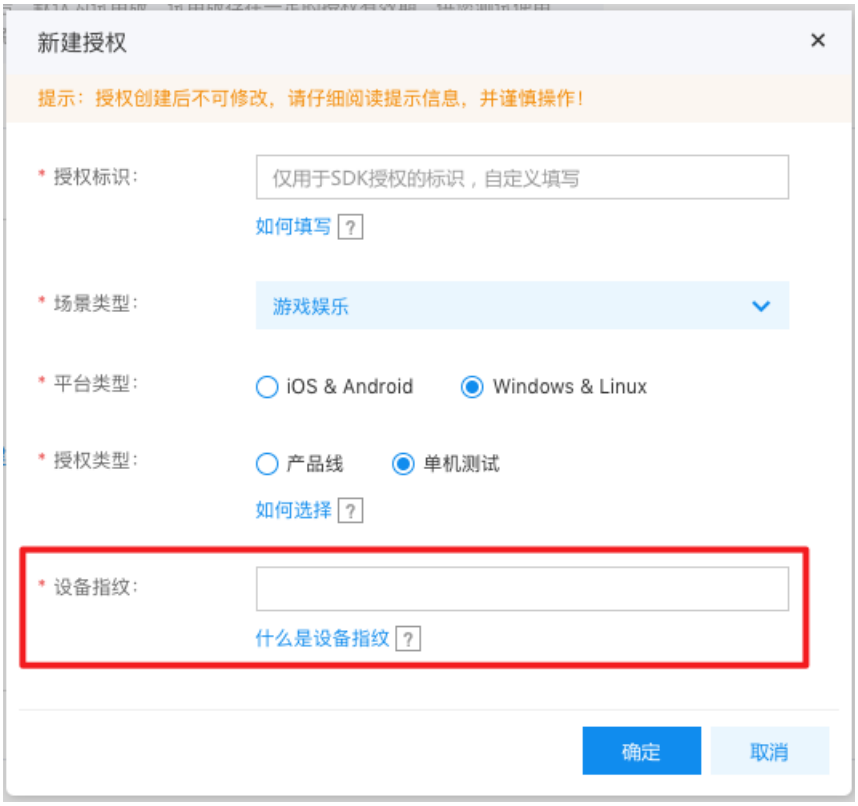
激活分测试阶段**按设备单机授权**、和**按产品线授权**。

- **按设备单机授权**：调试阶段的一种授权方式，主要用于指定设备上反复测试程序功能，激活状态只适用于授权的设备硬件。
- **按产品线授权**：针对发布的exe，该exe可应用于所有设备。

3.3.1 按设备单机授权

第一步：通过脚本sh face_ubuntu.sh(若为centos,则执行sh face_centos.sh)，则可在页面输出device id的设备指纹信息。

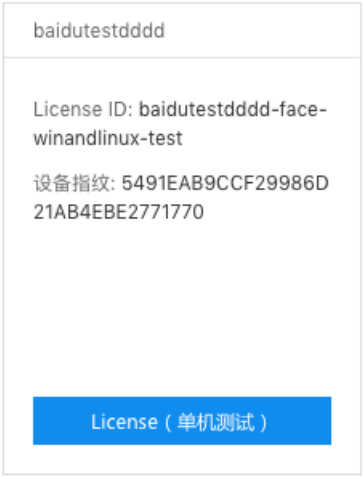
第二步：复制此硬件指纹信息，在后台填写到下图所在的位置，并创建授权，下载对应的License文件。



第三步（重要！！）：在SDK中，您需要在初始化的时候传入license_key，此license_key为您在后台填写的「授权标识」，添加一个-face-winadnlinux-test的后缀，如下图所示，**授权标识填写的为「baidutest」，则license_key为：baidutest-face-winadnlinux-test。**


```
 // 传入license_key，牢记在后台填写的「授权标识」，并在此授权标识基础上添加 -face-winadnlinux-test 的后缀
// 如您填写的授权标识填写的为「baidutest」，则license_key为：baidutest-face-winadnlinux-test
std::string license_key = "baidutest-face-winadnlinux-test";

//初始化sdk
sdk_init(license_key.c_str());
```

第四步：在官网下载License文件，如下图所示：



需要重命名为：license.ini，并**拷贝到可执行文件所在路径同一目录位置**。

以上四步完毕后，您的可执行文件就可以通过授权激活运行。

3.3.2 按产品线授权

该方式授权，主要用于发布正式的exe安装包，授权将会识别具体的exe程序，确保指定exe的正常运行。

> **注**：所以每次程序升级，需要单独创建一个License文件进行更新。

第一步：在后台新建授权中表单中，平台类型选择「Windows & Linux」，授权类型选择「产品线」，并填写以下几个内容：

1. 授权标识：自定义用于标识您的license授权信息，每个授权只能有唯一的标识，支持英文、数字、横线，20个英文字符以内；

2. 产品名：填写exe可执行程序的具体名称，例如face.exe，则产品名为face.exe；

3. 程序文件的md5：**此项稍后填写**，exe文件的md5：如face.exe，请填写此程序的md5值

第二步：在SDK初始化的时候传入license_key，此license_key为您在后台填写的「授权标识」，添加一个-face-linux的后缀，如下图所示，**授权标识填写的为「baidutest」，则license_key为：baidutest-face-linux。**

```
 // 传入license_key，牢记在后台填写的「授权标识」，并在此授权标识基础上添加 -face-linux 的后缀
// 如您填写的授权标识填写的为「baidutest」，则license_key为：baidutest-face-linux
std::string license_key = "baidutest-face-linux";

//初始化sdk
sdk_init(license_key.c_str());
```

第三步：完成License代码配置后，再编译生成可执行文件，用md5命令或工具生成该可执行文件的md5，并填写到第一步的表单中，提交表单，并下载生成的License文件。

> **注**：可执行文件一旦编译后，则license_key和文件本身不能再编译改变，否则该license文件无法通过激活。

第四步：若按上述步骤生成license文件后，重命名为:license.ini，并拷贝到可执行文件所在路径同一目录位置，则可运行可执行文件通过授权激活。

3.4 运行Demo工程

Demo示例工程BaiduFaceCollect展示了如何集成百度人脸识别采集SDK。即导入`so`及头文件`face.h`。另外为了示例视频人脸跟踪等，用到了一些opencv的库文件以及一些实现demo的支持文件，如image_base64等。

> **注**：这些支持文件均为代码开源或是开源库

在BaiduFaceCollect中的test_face.cpp的main()方法中，有使用sdk的各个接口方法示例。

SDK实现的主要功能有人脸实时跟踪检测、人脸属性、人脸质量，返回识别出的人脸信息等，另外支持对人脸检测进行设置，达到根据设置进行识别的目的。各接口功能及传入参数和返回结果（返回结果一般为json格式的字符串）详见下方接口说明。

4、接口说明

4.1 人脸检测track接口（传入图片）

方法名

`track`

方法说明

人脸检测，返回人脸信息

请求信息

参数	必须	类型	描述
out	是	std::vector	**返回的检测到的人脸图片信息结构体** std::vector *& out 通过传入TrackFaceInfo的结构体指针引用，在检测到人脸信息后，用该引用返回人脸信息。TrackedFaceInfo结构体为检测到的人脸信息，其中：box为检测到的人脸框（为FaceInfo结构体,包含人脸框大小，中心点坐标，人脸宽度等信息）landmarks为检测到的人脸关键点，一般为144个。score为人脸打分headPose的向量组为人脸x,y,z的三个角度
image	是	string	人脸图片信息，根据image_type，传入图片内容
img_type	是	int	传入的图片类型。为1时候表示base64编码的图片，为2时候表示传入图片的本地路径。BASE64：图片的base64值；FACE_FILE：图片的本地文件路径地址；
maxTrackObjNum	是	int	最多检测人脸数量，默认为1，最大不超过5

返回信息

返回结果为int，表示检测到几个人脸信息，1为1个人，0为没检测到人脸。

4.2 人脸检测track接口（传入二进制图片buffer）

方法名

`track_by_buf`

方法说明

人脸检测，返回人脸信息

请求信息

参数	必须	类型	描述
out	是	std::vector	**返回的检测到的人脸图片信息结构体** std::vector *& out 通过传入TrackFaceInfo的结构体指针引用，在检测到人脸信息后，用该引用返回人脸信息。TrackedFaceInfo结构体为检测到的人脸信息，其中：box为检测到的人脸框（为FaceInfo结构体,包含人脸框大小，中心点坐标，人脸宽度等信息）landmarks为检测到的人脸关键点，一般为144个。score为人脸打分headPose的向量组为人脸x,y,z的三个角度
image	是	Unsigned char *	人脸图片信息，二进制图片buffer内容
size	是	int	二进制图片的大小
maxTrackObjNum	是	int	最多检测人脸数量，默认为1，最大不超过5

返回信息

返回结果为int，表示检测到几个人脸信息，1为1个人，0为没检测到人脸。

4.3 人脸检测track_max_face接口

方法名

`track_max_face`

方法说明

检测最大人脸。

请求信息

参数	必须	类型	描述
out	是	std::vector<TrackFaceInfo>	**返回的检测到的人脸图片信息结构体** std::vector *& out 通过传入TrackFaceInfo的结构体指针引用，在检测到人脸信息后，用该引用返回人脸信息。TrackedFaceInfo结构体为检测到的人脸信息，其中：box为检测到的人脸框（为FaceInfo结构体,包含人脸框大小，中心点坐标，人脸宽度等信息）；landmarks为检测到的人脸关键点，一般为144个；score为人脸打分；headPose的向量组为人脸x,y,z的三个角度
image	是	string	图片信息（数据大小应小于10M）
image_type	是	int	为1时候表示base64编码的图片；为2时候表示传入图片的本地路径。 BASE64：图片的base64值；FACE_FILE：图片的本地文件路径地址；

返回信息

返回结果为int，表示检测到几个人脸信息，1为1个人，0为没检测到人脸。

4.4 人脸检测track_max_face接口（传入二进制图片buffer）

方法名

`track_max_face_by_buf`

方法说明

检测最大人脸。

请求信息

参数	必须	类型	描述
out	是	std::vector<TrackFaceInfo>	**返回的检测到的人脸图片信息结构体** std::vector *& out 通过传入TrackFaceInfo的结构体指针引用，在检测到人脸信息后，用该引用返回人脸信息。TrackedFaceInfo结构体为检测到的人脸信息，其中：box为检测到的人脸框（为FaceInfo结构体,包含人脸框大小，中心点坐标，人脸宽度等信息）；landmarks为检测到的人脸关键点，一般为144个；score为人脸打分；headPose的向量组为人脸x,y,z的三个角度
image	是	Unsigned char *	人脸图片信息，二进制图片buffer内容
size	是	int	二进制图片的大小

返回信息

返回结果为int，表示检测到几个人脸信息，1为1个人，0为没检测到人脸。

4.5 人脸检测设置接口

请参考`TestFaceApi`工程中的代码示例及头文件`BaiduFaceApi.h`中的定义及`setting.cpp`里的代码注释。

4.6 USB摄像头检测（实时视频帧人脸检测）

通过打开usb摄像头，返回实时检测到的视频帧人脸信息。请参考`TestFaceApi`中的示例`liveness.cpp`中的`usb_track_face_info`，该方法中用到了人脸检测track视频帧，接口如下4.5：

4.7 人脸检测track接口（传入opencv的mat）

方法名

`Track`

方法说明

人脸检测，返回人脸信息。

请求信息

参数	必须	类型	描述
out	是	`std::vector<TrackFaceInfo>`	**返回的检测到的人脸图片信息结构体** std::vector *& out 通过传入TrackFaceInfo的结构体指针引用，在检测到人脸信息后，用该引用返回人脸信息。TrackedFaceInfo结构体为检测到的人脸信息，其中：box为检测到的人脸框（为FaceInfo结构体,包含人脸框大小，中心点坐标，人脸宽度等信息）；landmarks为检测到的人脸关键点，一般为144个；score为人脸打分；headPose的向量组为人脸x,y,z的三个角度
Mat	是	Opencv格式的单帧图片mat	人脸图片信息
maxTrackObjNum	是	int	最多检测人脸数量。默认为1，最大不超过5

返回信息

返回结果为TrackedFaceInfo的向量指针,向量组为0时候表示没检测到人脸。TrackedFaceInfo结构体为检测到的人脸信息，其中：box为检测到的人脸框（为FaceInfo结构体,包含人脸框大小，中心点坐标，人脸宽度等信息）；landmarks为检测到的人脸关键点，一般为144个；score为人脸打分；headPose的向量组为人脸x,y,z的三个角度。

4.8 人脸质量接口（通过传入图片）

方法名

`face_quality`

方法说明

人脸质量检测接口

请求信息

参数	必须	类型	示例描述
image	是	string	需要检测的人脸图片，小于10M, 图片类型根据image_type参数定
imgae_type	是	int	图片类型，必选择以下2种形式之一。 Image_type为1代表BASE64为2代表FACE_FILE。 **BASE64**：图片的base64值； **FACE_FILE**：图片的本地文件路径地址；

返回信息

errno信息：

- errno=0：Success
- errno>0：错误码对应的详细msg

data信息:

字段	类型	说明
bluriness	float	光照
illum	float	模糊度
occl_l_eye	float	左眼遮挡度
occl_r_eye	float	右眼遮挡度
occl_nose	float	鼻子遮挡度
occl_mouth	float	嘴巴遮挡度
occl_l_contour	float	左脸遮挡度
occl_r_contour	float	右脸遮挡度
occl_chin	float	下巴遮挡度

5、错误码及错误信息

错误码	错误内容	错误描述
0	SUCCESS	成功
1	SYSTEM ERROR	系统错误
2	UNKNOWN ERROR	未知错误
1001	NOT_AUTH	授权校验失败
1002	REQUEST PARAMS ERROR	请求参数错误
1003	DB_OP_FAILED	数据库操作失败
1004	NO_DATA	没有数据
1005	RECORD_UNEXIST	记录不存在
1006	RECORD_ALREADY_EXIST	记录已经存在
1007	FILE_NOT_EXIST	文件不存在
1008	GET_FEATURE_FAIL	提取特征值失败
1009	FILE_TOO_BIG	文件太大
1010	FACE_RESOURCE_NOT_EXIST	人脸资源文件不存在
1011	FEATURE_LEN_ERROR	特征值长度错误
1012	DETECT_NO_FACE	未检测到人脸

6、常见问题

Q：激活后是否可以把激活文件license.ini拷贝到其他设备运行？
A：如果是按设备激活的，则license.ini和设备绑定，只能用于该设备测试，若是按产品方式激活，则该激活文件可随产品可执行文件一起拷贝到任意设备上运行。

安全加固采集SDK升级文档

Android

本文档针对已接入线上增强级采集SDK、需升级到安全版本的客户提供升级指导。

安全增强级采集SDK是目前线上增强级采集SDK的安全加固版本，**针对ROM注入、相机劫持、视频流替换等攻击方式进行安全升级，符合国家网信办安全标准**，安全升级点如下:

- **代码保护:** 保护人脸识别过程中，图像数据不被Hook替换

- **图像保护:** 对图像本身增加多重加密及签名验证
- **网络保护:**增强风险网络环境检测能力，避免中间人攻击
- **逻辑保护:**避免人脸识别业务流程逻辑被攻击
- **风险环境扫描:**提升Root检测、重打包、云手机、生物探针等设备风险检测能力

升级方式请参考以下步骤

1.删除项目的对faceplatform 和faceplatform-ui的依赖

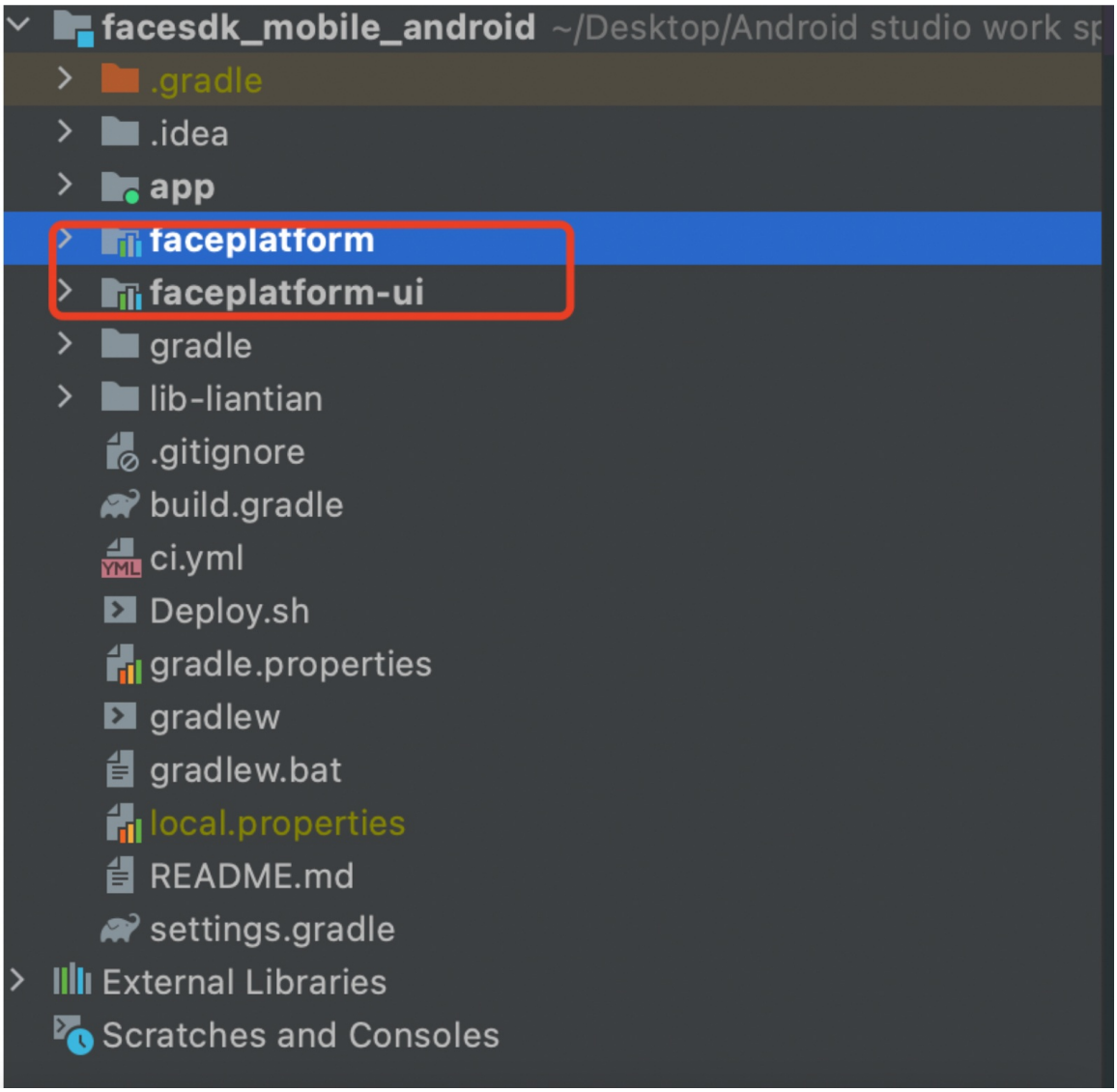
根目录下**settings.gradle** 文件 删除依赖

```
include ':faceplatform-ui'
include ':faceplatform'
```

app module 下**build.gradle**文件注释掉如下依赖，第三步后重新打开

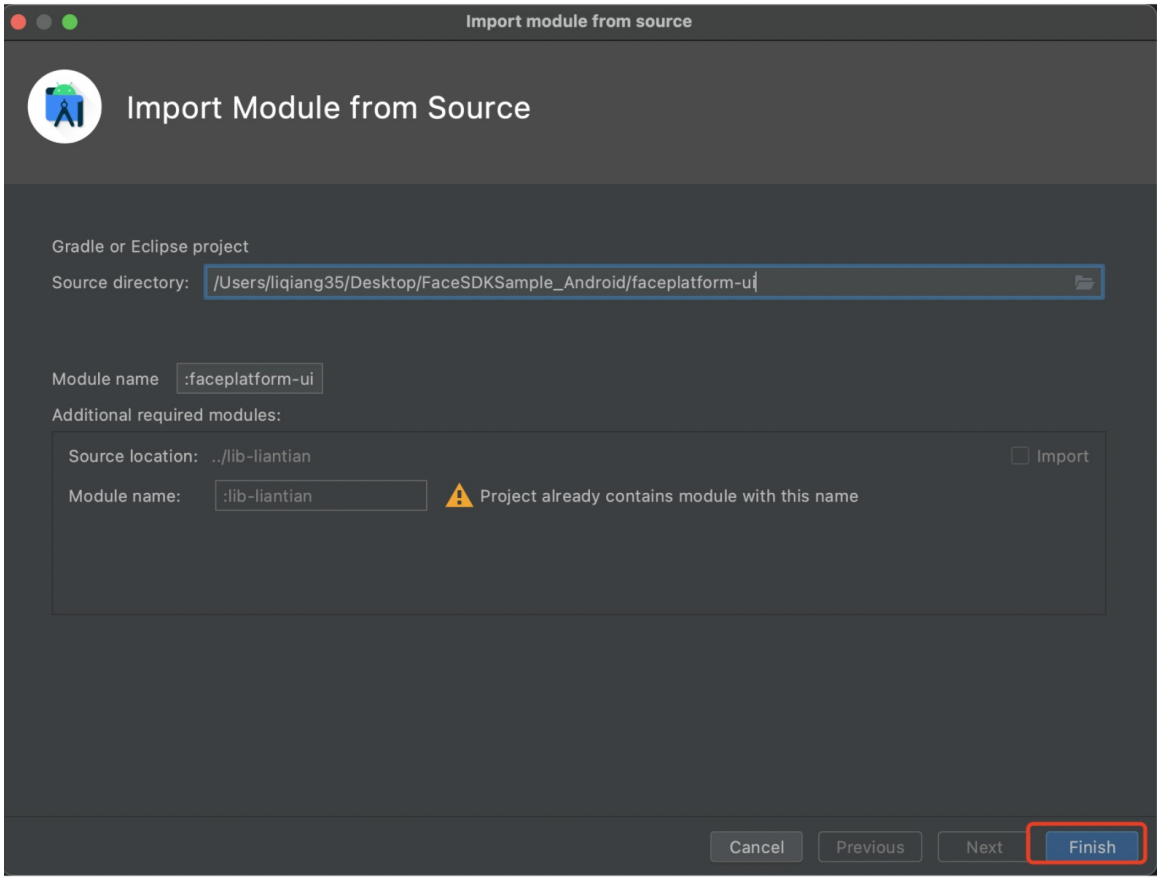
```
compile project(path: ':faceplatform-ui')
```

重新进行**sync**操作，如下图所示



2.导入第一步下载安全SDKDemo里面的faceplatform-ui 及 lib-liantian module

点击**file ->new ->import module** 点击**finish**



3.打开app module 下build.gradle文件添加如下依赖并重新build

```
compile project(path: ':faceplatform-ui')
```

此时 **setting.gradle**包含如下**module**

```
include ':faceplatform-ui'
include ':lib-liantian'
```

4.app 目录下AndroidManifest.xml **文件，升级了安全SDK后，将之前的**liantian**类删除掉，如下都可以删除**

```
<!--安全设备指纹接入 start-->
    <activity
        android:name="com.baidu.liantian.LiantianActivity"
        android:exported="true"
        android:theme="@android:style/Theme.Translucent"
        android:excludeFromRecents="true"
        android:launchMode="standard">
        <intent-filter>
            <action android:name="com.baidu.action.Liantian.VIEW" />
            <category android:name="com.baidu.category.liantian"/>
            <category android:name="android.intent.category.DEFAULT"/>
        </intent-filter>
    </activity>

    <receiver android:name="com.baidu.liantian.LiantianReceiver" android:exported="false">
        <intent-filter>
            <action android:name="com.baidu.action.Liantian.VIEW" />
            <category android:name="com.baidu.category.liantian"/>
            <category android:name="android.intent.category.DEFAULT"/>
        </intent-filter>
        <intent-filter android:priority="2147483647">
            <action android:name="android.intent.action.BOOT_COMPLETED"/>
        </intent-filter>
    </receiver>

    <!--将com.baidu.idl.face.demo替换成您工程的包名-->
    <provider android:authorities="com.baidu.idl.face.demo.liantian.ac.provider"
android:name="com.baidu.liantian.LiantianProvider" android:exported="true"/>

    <service
        android:name="com.baidu.liantian.LiantianService"
        android:exported="false">
        <intent-filter>
            <action android:name="com.baidu.action.Liantian.VIEW" />
            <category android:name="com.baidu.category.liantian"/>
            <category android:name="android.intent.category.DEFAULT"/>
        </intent-filter>
    </service>
```

5.FaceLivenessExpActivity onLivenessCompletion **接口修改为** onCollectCompletion****

```
@Override
public void onLivenessCompletion(FaceStatusNewEnum status, String message,
                                 HashMap<String, ImageInfo> base64ImageCropMa
                                 HashMap<String, ImageInfo> base64ImageSrcMap
    super.onLivenessCompletion(status, message, base64ImageCropMap, base64Ima
    if (status == FaceStatusNewEnum.OK && mIsCompletion) {
        // 获取最优图片
        getBestImage(base64ImageCropMap, base64ImageSrcMap);
    } else if (status == FaceStatusNewEnum.DetectRemindCodeTimeout) {
        if (mViewBg != null) {
            mViewBg.setVisibility(View.VISIBLE);
        }
        showMessageDialog();
    }
}
```

6.在Application**里初始化 LH安全类,填写自己的**apiKey**及**secretKey****

```
public void onCreate() {
    super.onCreate();
    LH.setAgreePolicy(getApplicationContext(), true);
    LH.init(getApplicationContext(), "vis-var-license-face-android",
            "xYBW9IAzPR6YVy1qsve8SPfc",
            "ZxA7mMPc9Xvn2S4tRZKbjWSQNun4PSjd");

}
```

如有其他问题可参考 安全加固采集SDK接入文档

iOS

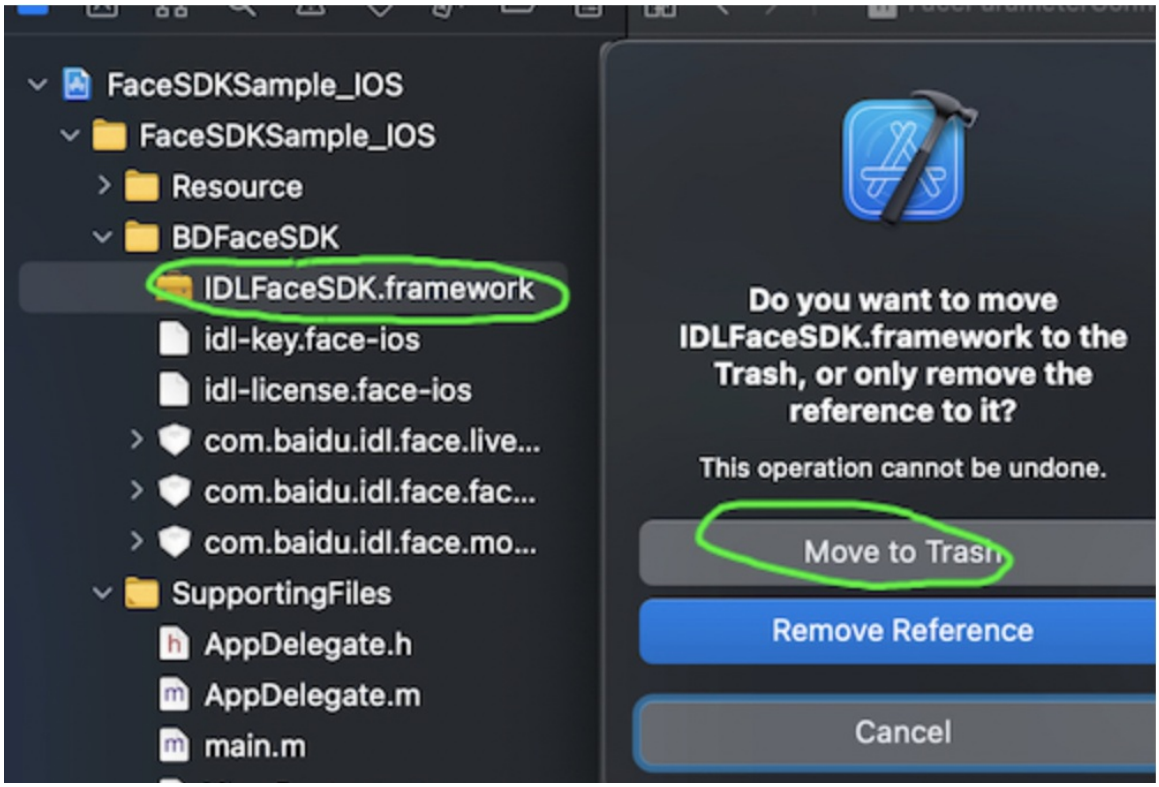
本文档针对已接入线上增强级采集SDK、需升级到安全版本的客户提供升级指导。

安全增强级采集SDK是目前线上增强级采集SDK的安全加固版本，**针对ROM注入、相机劫持、视频流替换等攻击方式进行安全升级，符合国家网信办安全标准**，安全升级点如下:

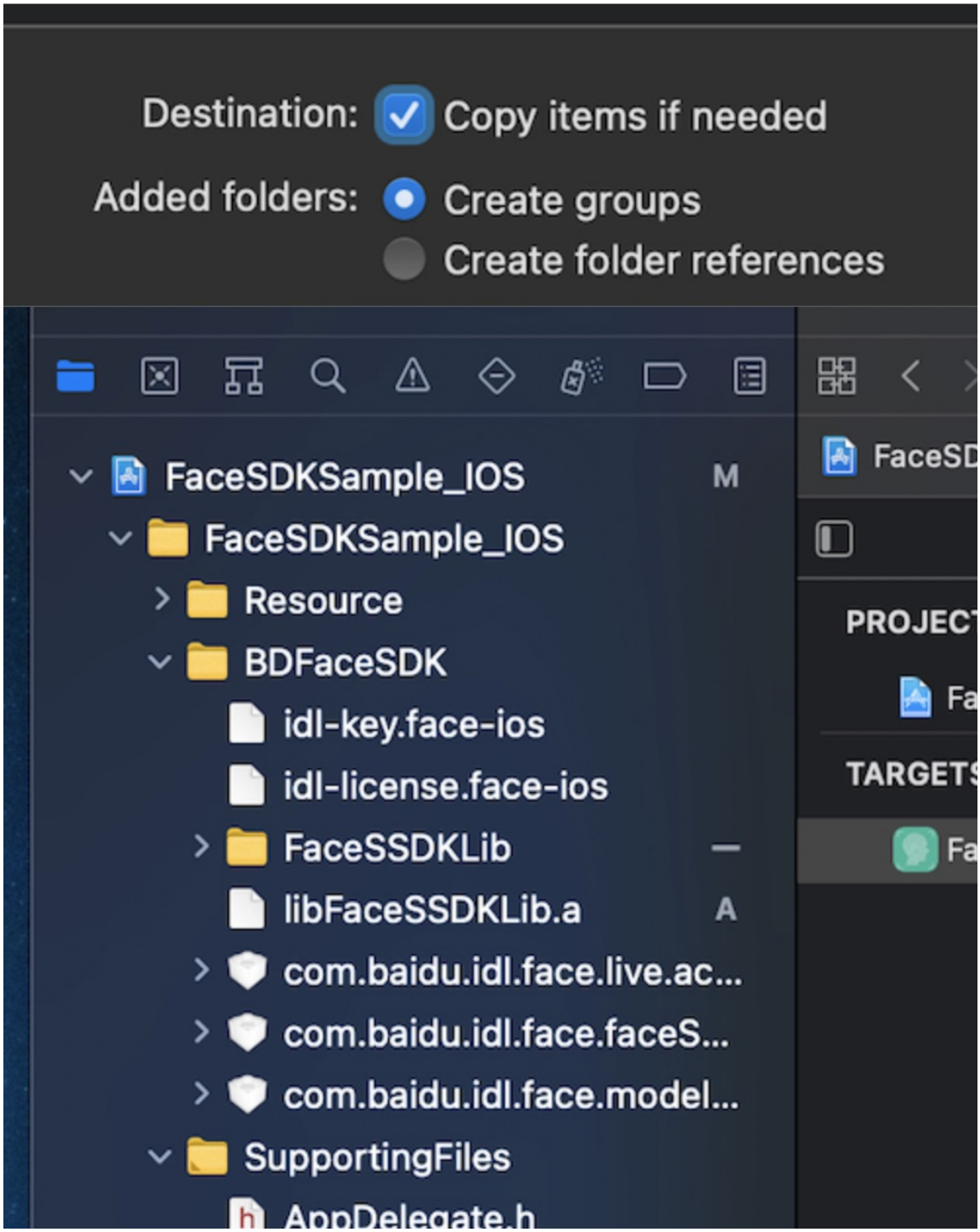
- **代码保护:** 保护人脸识别过程中，图像数据不被Hook替换
- **图像保护:** 对图像本身增加多重加密及签名验证
- **网络保护:**增强风险网络环境检测能力，避免中间人攻击
- **逻辑保护:**避免人脸识别业务流程逻辑被攻击
- **风险环境扫描:**提升Root检测、重打包、云手机、生物探针等设备风险检测能力

升级方式请参考以下步骤

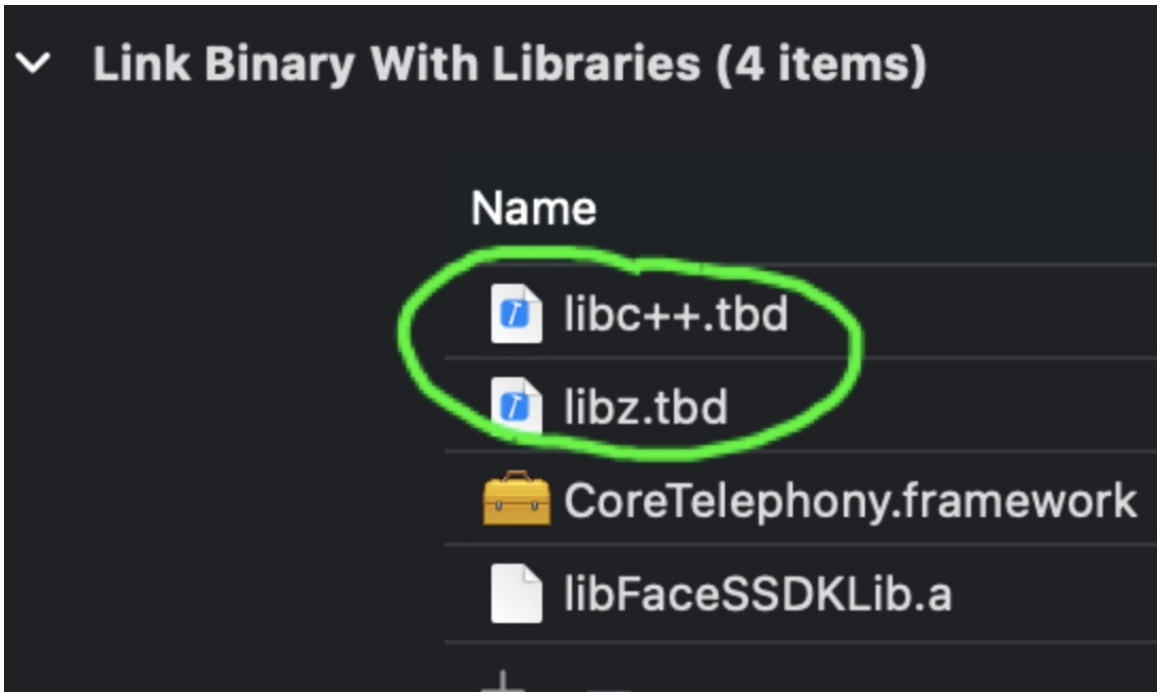
1、将 IDLFaceSDK.framework 从索引中删除，如下图所示，选择moveToTrash;



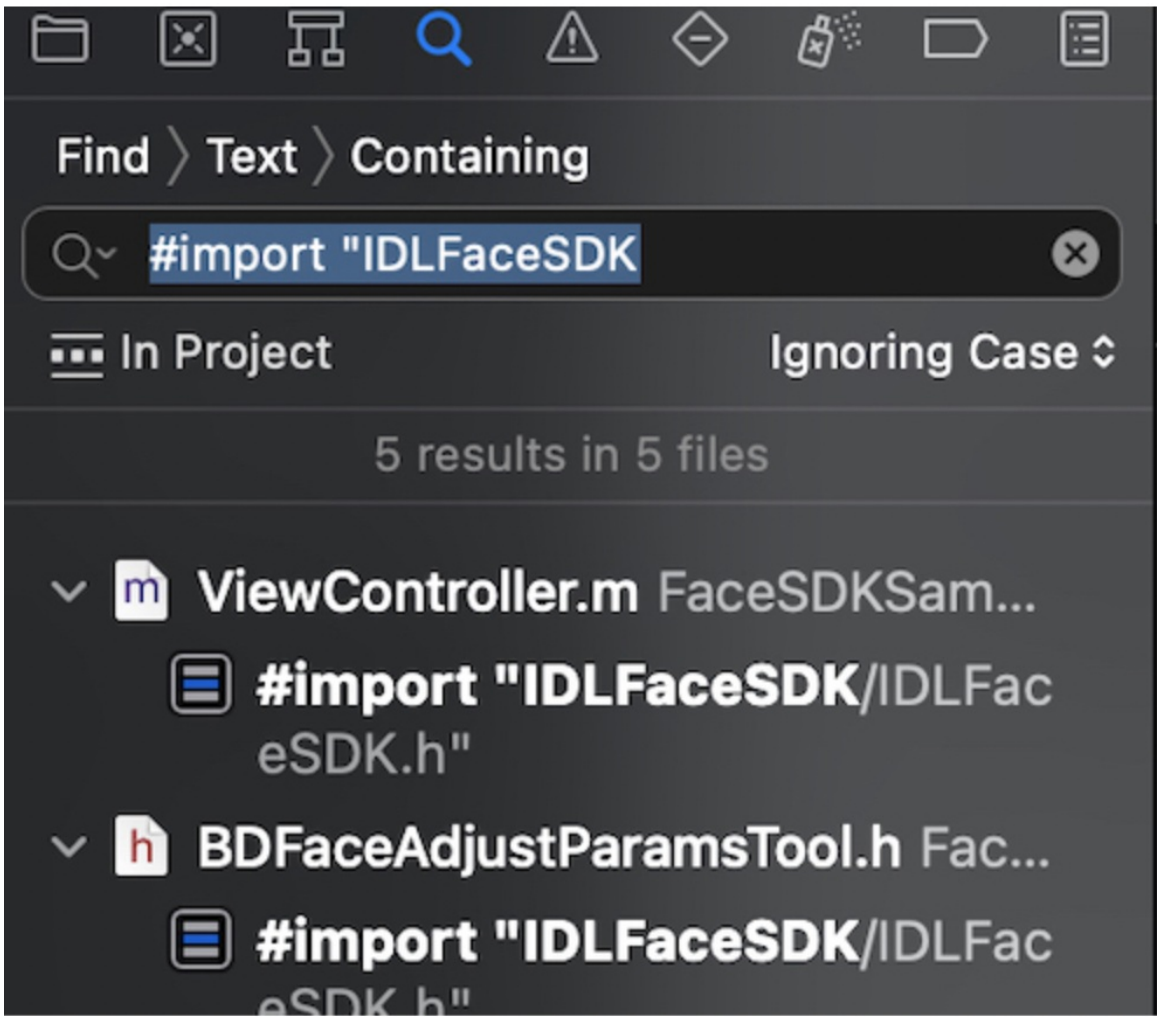
2、将安全增强级SDK的 FaceSSDKLib 头文件文件夹和libFaceSSDKLib.a 拖入刚才的BDFaceSDK文件夹索引中，如下图所示：



3、添加依赖库 libz.tbd 和 libc++.tbd ，如下图所示：



4、在工程中全局搜索#import "IDLFaceSDK"或#import<<IDLFaceSDK,并将引入字符串全部替换为#import "SSFaceSDK.h"，如下图所示:



5、将所有 IDLFace 开头的类，替换为 SSFace 开头 安全增强级SDK将所有的类名称做了修改，原类名以**IDLFace**开头，安全增强级为做区分改为以**SSFace**开头，如原类名 [IDLFaceDetectionManager sharedInstance] 替换为 [SSFaceDetectionManager sharedInstance]

部分接口没有了，注释掉即可，如:

[[IDLFaceDetectionManager sharedInstance] startInitial];

[[IDLFaceDetectionManager sharedInstance] reset];

6、重要差异总结

(1) idl-key.face-ios 文件不一致了，需要重新改文件；

(2) 回调差异：原SDK中， `- (void)faceProcesss:(UIImage *)image` 为识别离线采集识别完成的回调函数，包括了识别中途的各种姿态错误的回调和最终采集是否成功的回调，但是新的SDK中，该方法的逻辑分成了两个回调

一个是活体检测状态的回调函数 `- (void)livenessActionDidFinishWithCode:(LivenessRemindCode)code;` 一个是人脸流程回调函数 `- (void)faceSessionCompletionWithStatus:(BDFaceCompletionStatus)status result:(NSDictionary *)result;`

因为差异比较大，这里只对容易出错的 BDFaceCompletionStatus 的状态问题作出解释：

```
 BDFaceCompletionStatusSuccess = 1,

BDFaceCompletionStatusNoRisk = 2,

BDFaceCompletionStatusImagesSuccess = 3,
```

上面三个，都是采集流程正常的回调，BDFaceCompletionStatusNoRisk 表示设备没有风险；BDFaceCompletionStatusSuccess代表着整个流程的走通，表示人脸离线检测和最终人证核验流程的结束；BDFaceCompletionStatusImagesSuccess代表了动作采集流程完成，但最终的人证核验检测还没有跑完。

(3) 在调用如下函数进行人脸识别之前，需先调用 initCollect 方法进行安全增强级SDK的初始化。 `[[SSFaceSDKManager sharedInstance] livenesswithList:livenessArray order:order numberOfLiveness:numberOfLiveness];`

（4）clientId和clientSecret分别对应的未升级版本中的face_api_key 和face_secret_key，仅更改名称

其余问题可参考 安全加固采集SDK接入文档

安全加固采集SDK接入文档

Android

1 简介

1.1 产品介绍

百度安全增强级采集 SDK Android 版是一种面向 Android 移动设备的人脸技术开发包，此版SDK包含人脸检测、活体识别等功能，以aar包+动态链接库的形式发布。基于该方案，开发者可以轻松的构建包含基于人脸采集和活体识别的实名认证和人脸比对流程的应用。

为了解决人脸的采集和使用过程中的安全问题，本SDK在实名认证和人脸比对流程中应用了多种安全解决方案，使第三方应用能更简单地集成安全的实名认证和人脸比对能力，本SDK为面向API level 22以上设备的Android APP提供流程安全的人脸采集，及采集后的实名认证和人脸比对流程。

1.2 功能介绍

此版SDK基于在线的安全能力封装为人脸比对和实名认证两个核心流程，主要用于在客户端执行这两种流程。因为这两种流程最终需要向服务器发起请求以完成认证，因此无法在离线环境下使用。

此版SDK所包含的能力如下：

- **动作活体检测**：通过让用户做出指定人脸配合式的交互动作，识别当前操作者是否为活体，此功能为离线使用，可设定指定动作是否使用及应用顺序。动作包含：眨眨眼、张闭嘴、向左摇头、向右摇头、向上抬头，向下低头6个。可有效抵御高清图片、3D建模、视频等攻击。
- **人脸质量检测**：判断视频流中的图片帧中，哪些图片质量较佳，即人脸图像特征清晰（满足姿态角、光照、模糊度、遮挡等校验）。

- **人脸图像采集**：通过本地SDK能力，采集人脸图像，同时经过人脸质量检测，确保采集到的人脸图像符合各条件校验（满足姿态角、光照、模糊度、遮挡等校验），为设备前端获取有效可分析人脸的主要功能。
- **授权**：SDK的授权判断，授权介质也称为license，在SDK使用中，需要通过license向授权服务器发起请求，判断SDK的使用合法性及使用有效期。
- **人脸比对和实名认证流程**：通过将原子操作封装为安全的流程，在流程中加入多种安全防护能力，能够抵御大多数针对人脸流程的攻击。因为不用自行维护流程，应用程序使用人脸比对和实名认证功能也变得更加简便。

1.3 兼容性

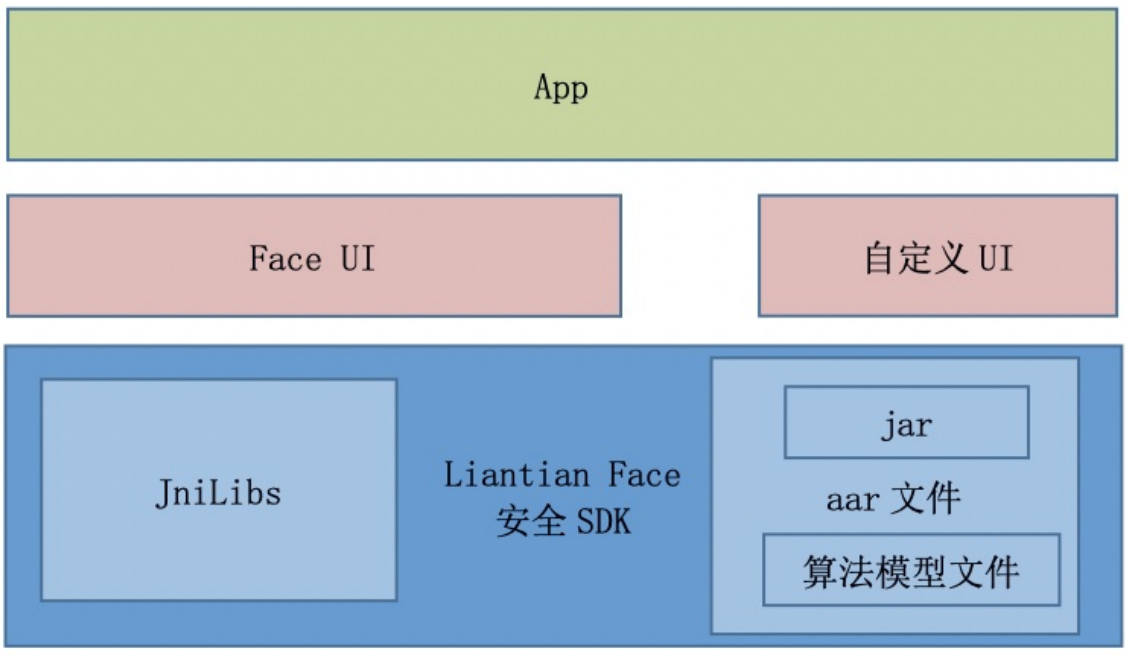
- 系统：支持 Android 5.1(API Level 22)及以上系统。需要开发者通过 minSdkVersion来保证支持系统的检测。
- 机型：手机和平板皆可（暂不支持横屏）
- 构架：支持 CPU架构平台【armeabi-v7a、arm64-v8a】
- 网络：支持 WIFI 及移动网络。

1.4 开发包说明

文件/文件夹名	说明
/lib-liantian	SDK lib库、模型文件以及流程逻辑代码打包的aar
/faceplatform-ui	SDK的UI库，封装采集和动作活体UI等功能，以及各平台的so库。so包含以下几个平台如果关注包大小，请自行删减。【armeabi-v7a、arm64-v8a】
/app	DEMO工程（包含首页面、采集成功失败页面、设置页面等）

2 集成指南

本章将进行 Step-By-Step的讲解,如何快速的集成 人脸Sdk到现有应用中。一个完整的Demo 请参考开发包中的示例程序 FacePlatform。方案架构参考下图：



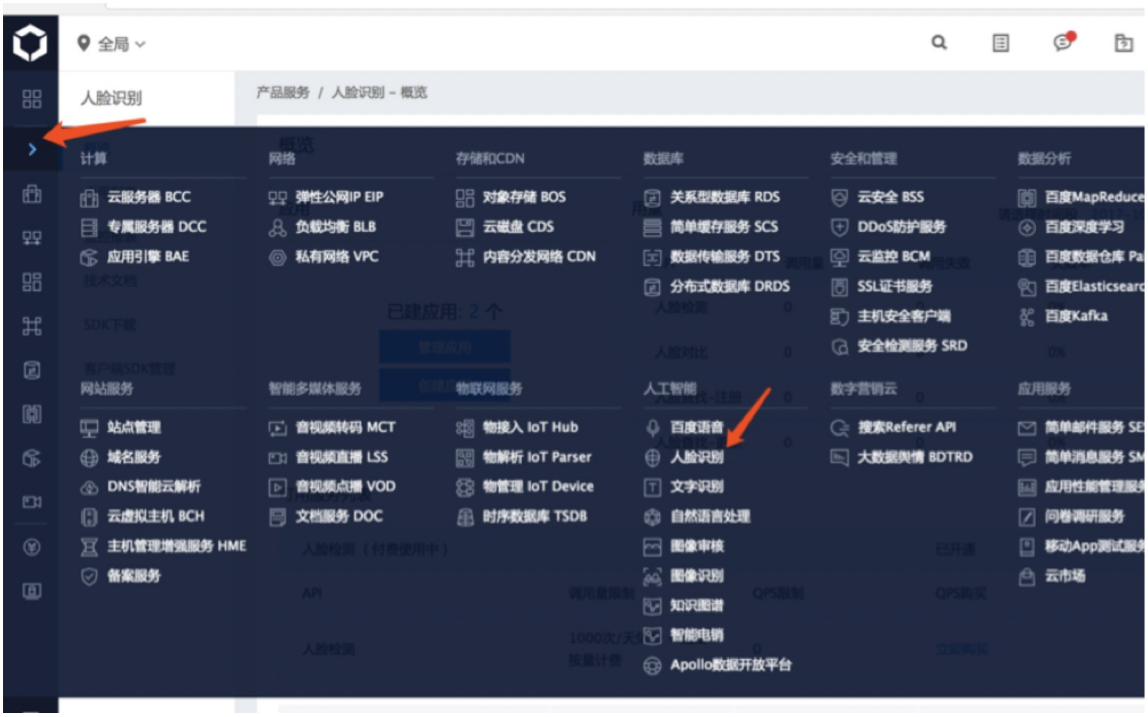
2.1 准备工作

2.1.1 申请License

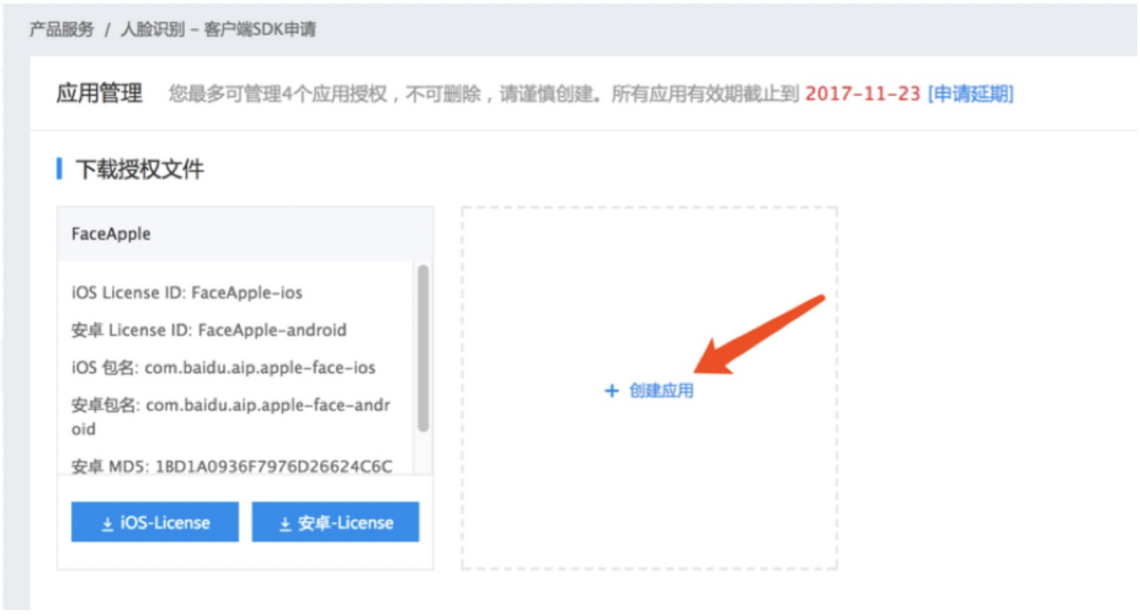
人脸SDK License：此license用于SDK离线功能使用，在您的申请人脸SDK的后台页面，全局->产品服务->人脸识别->客户端

SDK申请

人脸控制台路径如下：

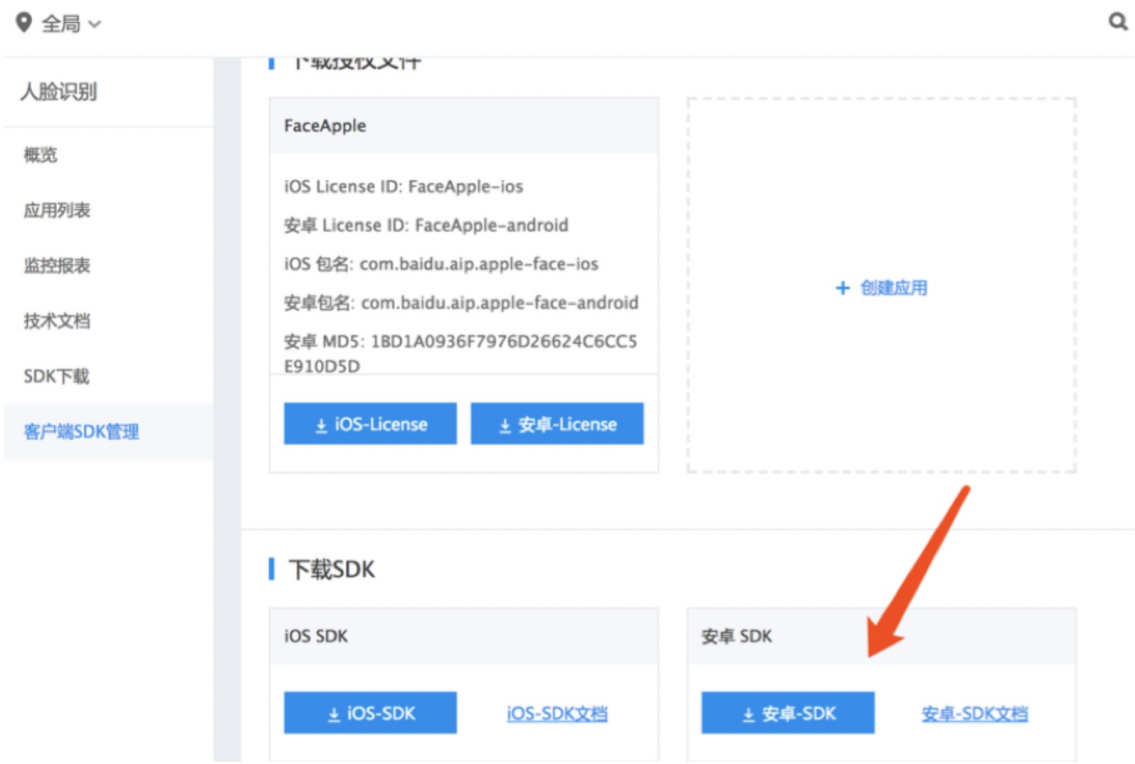


点击客户端SDK管理，弹出如下图：创建应用（这里创建应用是为了使用离线SDK，上面创建应用为了使用人脸在线接口，如注册、识别等）

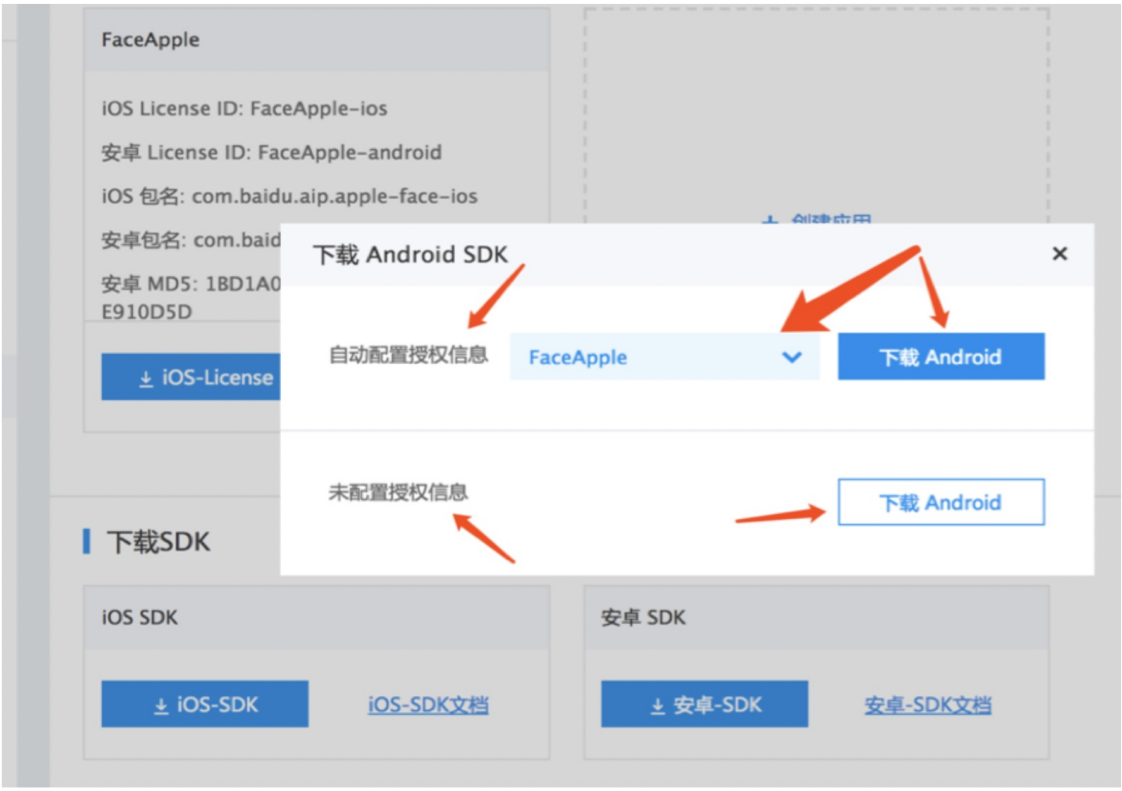


在弹出的框中输入授权标识，选择应用类型，应用系统，以及包名，详情请查看输入框右边提示

2.1.2 下载SDK



下载SDK分为自动配置授权信息（创建license后就可以选择为该应用，下载后SDK自动帮您配置授权，不用下载license拷贝到工程中，初始化参数licenseID,包名也帮您配置好了）和未配置授权信息两种方式：



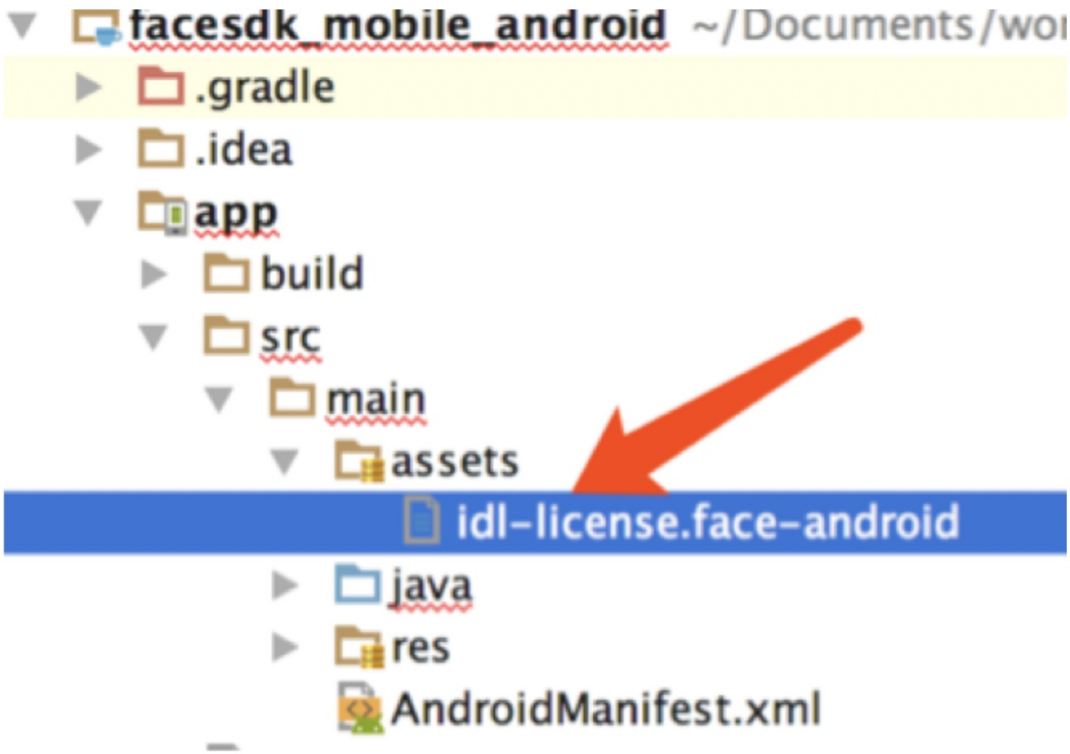
2.2 运行示例工程

2.2.1 运行自动配置授权信息的示例工程

该下载的示例工程，已经帮您改好了license和包名 • Android Studio导入下载的示例工程 • 配置打包签名文件，由于SDK运行时会校验签名文件的MD5是否和申请时填入一样。为了便于debug能正常使用SDK，需要配置打包签名文件。 • 运行示例工程

2.2.2 运行未配置授权信息的示例工程

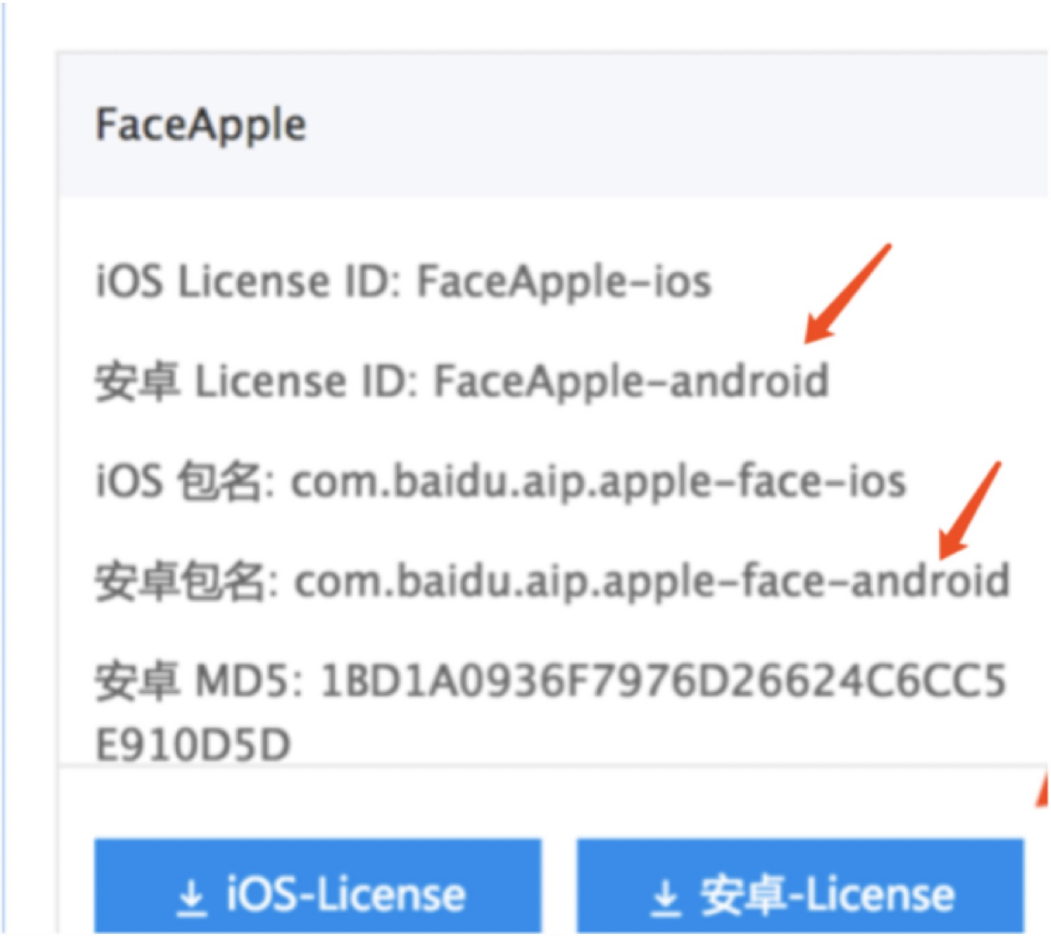
（1）Android Studio导入下载的示例工程 （2）下载license拷贝到工程的assets目录



（3）修改HomeActivity.java的initialize方法参数

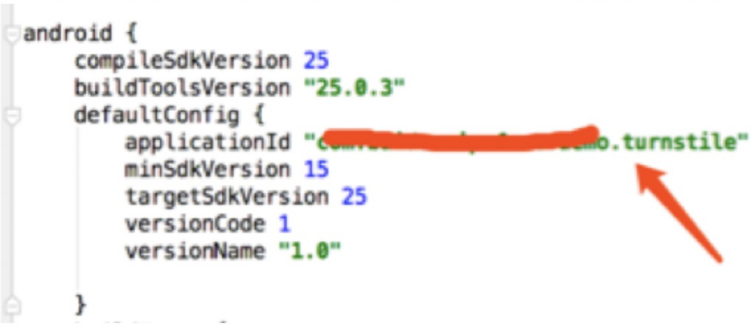

```
// 为了android和ios 区分授权，appId=appname_face_android ,其中appname为申请sdk时的应用名
// 应用上下文
// 申请License取得的APPID
// assets目录下License文件名
FaceSDKManager.getInstance().initialize(mContext, licenseID: "...",
        licenseFileName: "...", new IInitCallback() {
            @Override
            public void initSuccess() {
                runOnUiThread(() -> {
                        Log.e(TAG, msg: "初始化成功");
                        showToast( msg: "初始化成功");
                        mIsInitSuccess = true;
                });
            }

            @Override
            public void initFailure(final int errCode, final String errMsg) {
                runOnUiThread(() -> {
                        Log.e(TAG, msg: "初始化失败 = " + errCode + " " + errMsg);
                        showToast( msg: "初始化失败 = " + errCode + ", " + errMsg);
                        mIsInitSuccess = false;
                });
            }
        });
```

（4）修



改app.gradle里面的包名为申请license填入的包名，如上图安卓包名。

（5）配

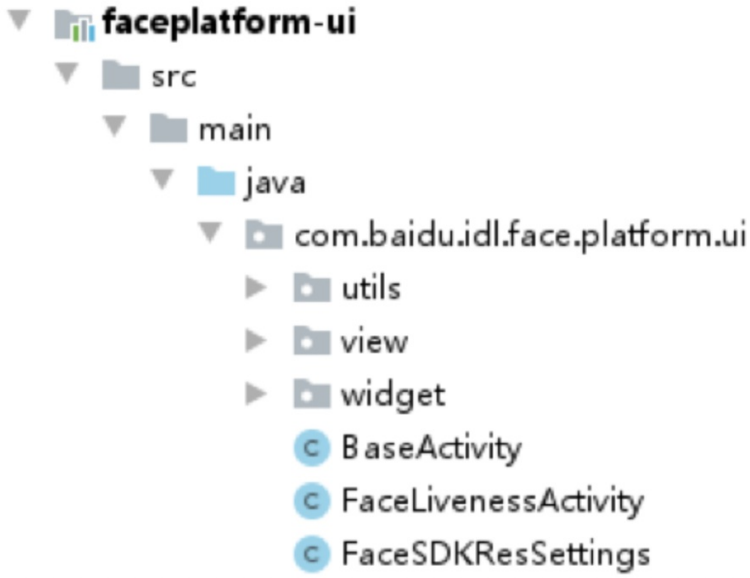


置打包签名文件，由于SDK运行时会校验签名文件的MD5是否和申请时填入一样。为了便于debug能正常使用SDK，需要配置打包签名文件。

（6）运行示例工程。如果无法正常体验，请查看logcat日志。是否有 FaceSDK-License LICENSE_INFO_CHECK_ERROR日志。如果有说明授权没有成功，可以查看本文档最后的常见问题进行解决。

2.3 添加SDK到工程

安全增强级采集SDK以androidstudio开发方式提供，以下介绍在android studio开发工具导入FaceSdk （1）将开发包中的lib-liantian库Copy 到工程根目录。 （2）将开发包中的faceplatform-ui库Copy 到工程根目录。 （3）SDK提供的了开源的faceplatform-ui库，把活体检测和人脸图像采集功能等功能进行了封装，适配了主流机型机型。如果需要使用，请添加faceplatform-ui模块到的工程中。faceplatform-ui目录结构如下图



（4）在build.gradle使用compile project引人faceplatform-ui库和lib-liantian库。 （5）Setting.gradle中include faceplatfrom-ui和lib-

liantian

```
include ':lib-liantian'
include ':faceplatform-ui'
```

（6）从官网下载授权文件license，复制到app/src/main/assets目录下。 （7）申请的license已经和打包签名key进行了绑定（申请时用到了签名的md5，为了便于debug模式也能调用SDK的功能，需要把debug的key改成申请license的key。 • 把key拷贝到项目根目录下 • 主appbuild.gradle android 下面添加(修改)signingConfigs相关的配置。如下图。

```
buildTypes {
    release {
        minifyEnabled false
        proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'
    }
}

signingConfigs {
    debug {
        keyAlias 'facesharp'
        keyPassword [redacted]
        storeFile file('../facesharp.jks')
        storePassword([redacted])
    }
    release {
        keyAlias [redacted]
        keyPassword [redacted]
        storeFile file('../facesharp.jks')
        storePassword([redacted])
    }
}
```

（8）将app module下的AndroidManifest.xml中的liantian库的注册包名改成您工程中的包名

```
<!--将com.baidu.idl.face.demo替换成您工程的包名-->
<provider android:authorities="[redacted].liantian.ac.provider" android:name="com.baidu.l

<service
    android:name="com.baidu.liantian.LiantianService"
    android:exported="false">
    <intent-filter>
        <action android:name="com.baidu.action.Liantian.VIEW" />
        <category android:name="com.baidu.category.liantian"/>
        <category android:name="android.intent.category.DEFAULT"/>
    </intent-filter>
</service>

<meta-data android:name="seckey_avscan" android:value= "660346260f8a841a04ec2a56815b421b"/>
<meta-data android:name="appkey_avscan" android:value= "100034"/>
<!--安全设备指纹接入 end-->
```

2.4 权限声明

名称	用途
需要动态申请的权限	
android.permission.READ_EXTERNAL_STORAGE	读取手机外部存储权限
android.permission.WRITE_EXTERNAL_STORAGE	写入手机外部存储权限
android.permission.CAMERA	拍照权限
不需要动态申请的权限	
android.permission.READ_PHONE_STATE	获取用户手机的 IMEI,用来唯一的标识用户
android.hardware.camera.autofocus	允许相机对焦
android.permission.INTERNET	允许访问网络
android.permission.WRITE_SETTINGS	允许修改系统设置
android.permission.WAKE_LOCK	屏幕常亮权限

2.5 混淆设置

如果工程需要做混淆处理的话，则在工程的proguard-rules.pro文件中添加如下配置。aar文件中已经包含了混淆设置，如果正常使用aar集成，可以不用单独进行混淆设置： -keep class com.baidu.vis.unified.license. *{;} -keep class com.baidu.liantian.* {;} -keep class com.baidu.baidusec. *{;} -keep class com.baidu.idl.main.facesdk.* {;}

3 功能使用

3.1 实名认证/人脸比对

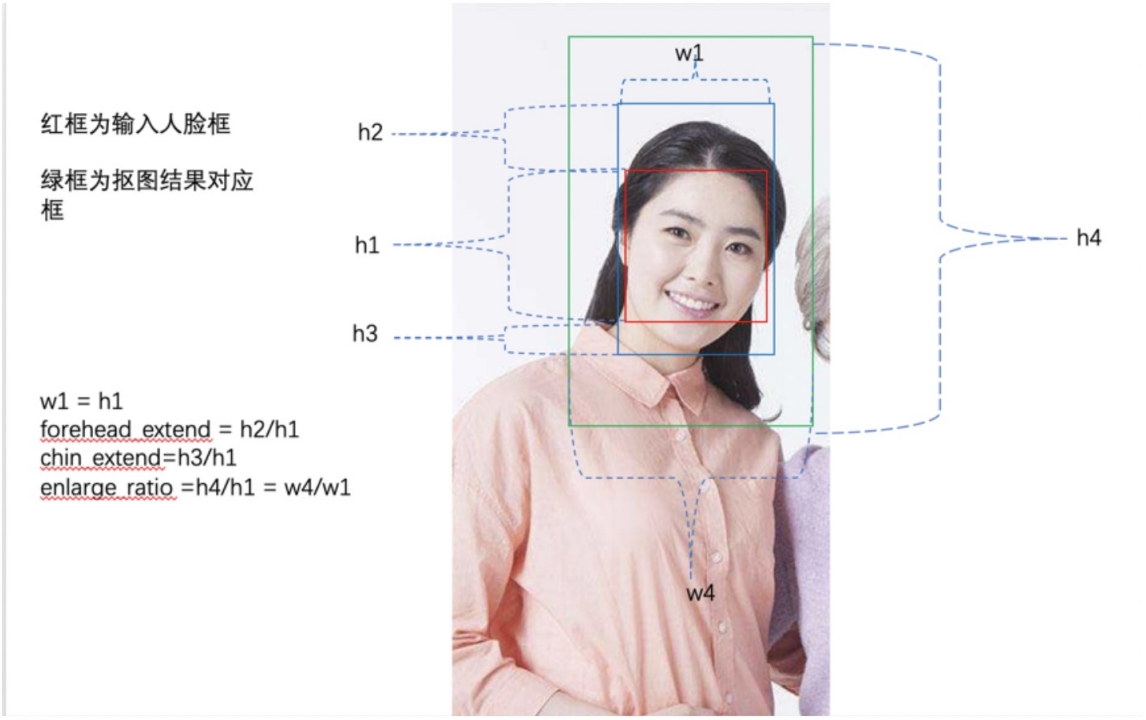
（1）同意隐私协议。 需要在用户同意隐私协议后调用LH.setAgreePolicy。用户未同意隐私协议前请勿调用初始化方法，以避免可能触发的隐私API调用。

（2）初始化SDK。 本SDK使用时需要同时初始化安全增强级采集SDK自身和Face采集模块。 为达到较好的安全效果，请在应用启动后尽早执行本SDK初始化，并设置同意隐私协议。 首先调用LH.init方法初始化安全模块。随后调用FaceSDKManager.getIntance().initialize(Context context, String licenseID, String licenseFileName, IInitCallback callback); 参数分别表示：当前上下文、鉴权key、鉴权名称、回调参数 。 Demo中此段代码在HomeActivity中。同时需要 （3） 初始化参数设置。

```
 FaceConfig config = FaceSDKManager.getInstance().getFaceConfig();
// SDK初始化已经设置完默认参数（推荐参数），也可以根据实际需求进行数值调整
// 质量等级（0：正常、1：宽松、2：严格、3：自定义）
// 获取保存的质量等级
SharedPreferencesUtil util = new SharedPreferencesUtil(mContext);
int qualityLevel = (int) util.getSharedPreference(Const.KEY_QUALITY_LEVEL_SAVE, -1);
if (qualityLevel == -1) {
    qualityLevel = ExampleApplication.qualityLevel;
}
// 根据质量等级获取相应的质量值（注：第二个参数要与质量等级的set方法参数一致）
QualityConfigManager manager = QualityConfigManager.getInstance();
manager.readQualityFile(mContext.getApplicationContext(), qualityLevel);
QualityConfig qualityConfig = manager.getConfig();
if (qualityConfig == null) {
    return false;
}
// 设置模糊度阈值
config.setBlurnessValue(qualityConfig.getBlur());
// 设置最小光照阈值（范围0-255）
config.setBrightnessValue(qualityConfig.getMinIllum());
```

```
// 设置最大光照阈值（范围0-255）
config.setBrightnessMaxValue(qualityConfig.getMaxIllum());
// 设置左眼遮挡阈值
config.setOcclusionLeftEyeValue(qualityConfig.getLeftEyeOcclusion());
// 设置右眼遮挡阈值
config.setOcclusionRightEyeValue(qualityConfig.getRightEyeOcclusion());
// 设置鼻子遮挡阈值
config.setOcclusionNoseValue(qualityConfig.getNoseOcclusion());
// 设置嘴巴遮挡阈值
config.setOcclusionMouthValue(qualityConfig.getMouseOcclusion());
// 设置左脸颊遮挡阈值
config.setOcclusionLeftContourValue(qualityConfig.getLeftContourOcclusion());
// 设置右脸颊遮挡阈值
config.setOcclusionRightContourValue(qualityConfig.getRightContourOcclusion());
// 设置下巴遮挡阈值
config.setOcclusionChinValue(qualityConfig.getChinOcclusion());
// 设置人脸姿态角阈值
config.setHeadPitchValue(qualityConfig.getPitch());
config.setHeadYawValue(qualityConfig.getYaw());
config.setHeadRollValue(qualityConfig.getRoll());
// 设置可检测的最小人脸阈值
config.setMinFaceSize(FaceEnvironment.VALUE_MIN_FACE_SIZE);
// 设置可检测到人脸的阈值
config.setNotFaceValue(FaceEnvironment.VALUE_NOT_FACE_THRESHOLD);
// 设置闭眼阈值
config.setEyeClosedValue(FaceEnvironment.VALUE_CLOSE_EYES);
// 设置图片缓存数量
config.setCacheImageNum(FaceEnvironment.VALUE_CACHE_IMAGE_NUM);
// 设置活体动作，通过设置list，LivenessTypeEunm.Eye, LivenessTypeEunm.Mouth,
// LivenessTypeEunm.HeadUp, LivenessTypeEunm.HeadDown,
// LivenessTypeEunm.HeadLeft, LivenessTypeEunm.HeadRight
config.setLivenessTypeList(ExampleApplication.livenessList);
// 设置动作活体是否随机
config.setLivenessRandom(ExampleApplication.isLivenessRandom);
// 设置开启提示音
config.setSound(ExampleApplication.isOpenSound);
// 原图缩放系数
config.setScale(FaceEnvironment.VALUE_SCALE);
// 抠图宽高的设定，为了保证好的抠图效果，建议高宽比是4：3
config.setCropHeight(FaceEnvironment.VALUE_CROP_HEIGHT);
config.setCropWidth(FaceEnvironment.VALUE_CROP_WIDTH);
// 抠图人脸框与背景比例
config.setEnlargeRatio(FaceEnvironment.VALUE_CROP_ENLARGERATIO);
// 检测超时设置
config.setTimeDetectModule(FaceEnvironment.TIME_DETECT_MODULE);
// 检测框远近比率
config.setFaceFarRatio(FaceEnvironment.VALUE_FAR_RATIO);
config.setFaceClosedRatio(FaceEnvironment.VALUE_CLOSED_RATIO);
FaceSDKManager.getInstance().setFaceConfig(config);
```

关于抠图内部实现方案如下图所示：



（4）

startActivity(new Intent(this, FaceLivenessExpActivity.class))启动界面。

（5）实现FaceProcessCallback，对流程中的回调事件进行处理。

（6）调用LH.startFaceVerify启动实名认证流程，或调用LH.startFaceCompare启动人脸比对流程。

（7）在onConfigCamera回调中，配置摄像头参数，以及设置预览图大小，人脸检测框的坐标。调用LH.setSoundEnable设置是否开启语音。

（8）在整个流程中，根据FaceProcessCallback的回调事件进行界面处理。在进入采集页面后，于希望开始流程的时机调用startFaceVerify或startFaceCompare开始流程。 该界面至少应当包含一个用于显示摄像头预览图像的SurfaceView，但不需要从开始就显示。请保证在onDeviceCheckResult回调后，直到onEnd回调为止的过程中，该SurfaceView可见。

一次正常的采集流程： 调用startFaceProcess -> onBegin被回调 -> 设备环境扫描 -> onDeviceCheckResult被回调-> onBeginCollectFaceInfo被回调 -> 开启摄像头 -> onConfigCamera被回调，用户配置摄像头 -> 开始预览 -> onCollectCompletion被回调（多次，需要更新界面，最后一次回调status为OK或DetectRemindCodeTimeout)-> onBeginRequestRemote 被回调 -> 请求云端进行人脸比对或实名认证 -> onEnd被回调。

一次异常的采集流程： onEnd可能在onBegin后的任何时间被回调，其他中间步骤可能会缺失。此种情况下，onEnd会回调错误码。

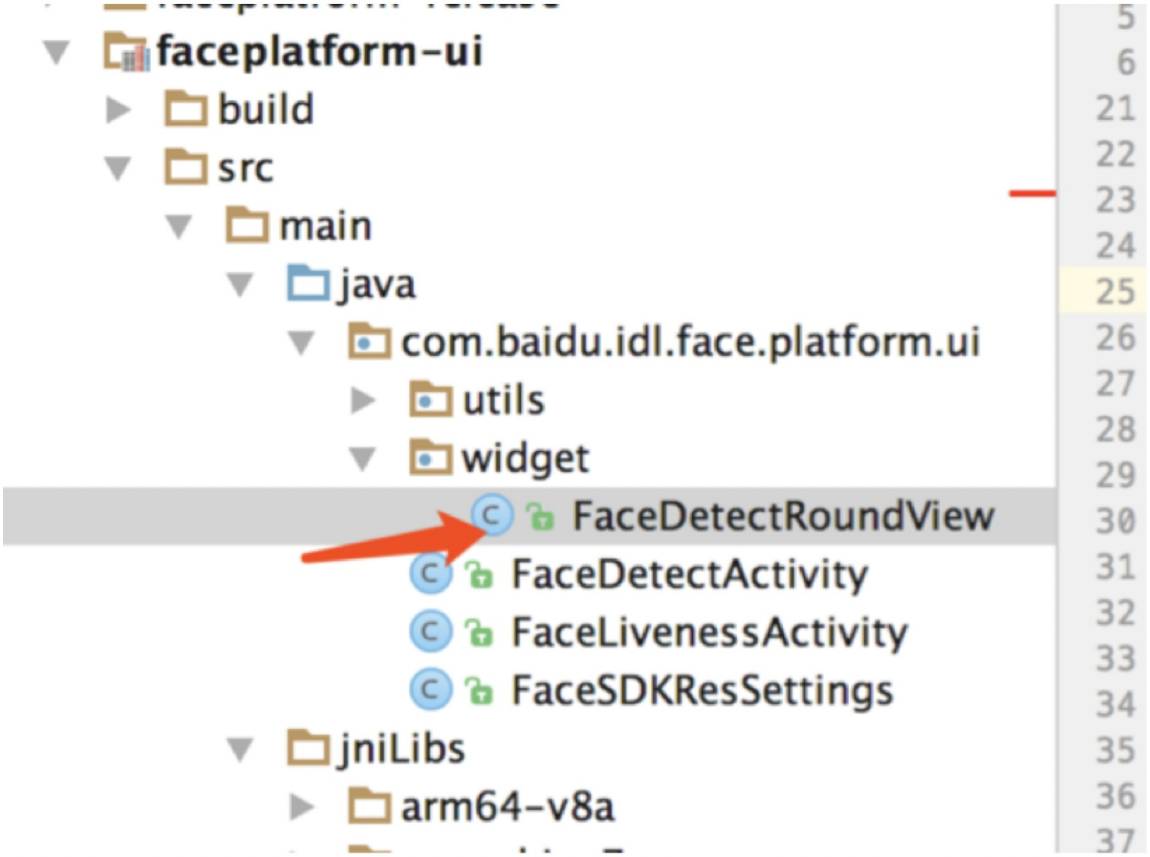
（9）采集完成后，会自动将数据发送到在onEnd中获取从服务器获取的实名认证/人脸比对结果。

3.2 质量校验设置

参数	名称	默认值	取值范围
minFaceSize	最小人脸阈值	200	
notFaceValue	非人脸阈值	0.6f	0~1.0f
brightnessValue	图片最小光照阈值	宽松：30、正常：40、严格：60	0-255f
brightnessMaxValue	图片最大光照阈值	宽松：240、正常：220、严格：200	0-255f
blurnessValue	图像模糊阈值	宽松：0.8f、正常：0.6f、严格：0.4f	0~1.0f
occlusionLeftEyeValue	左眼遮挡阀值	宽松：0.95f、正常：0.8f、严格：0.4f	0~1.0f
occlusionRightEyeValue	右眼遮挡阀值	宽松：0.95f、正常：0.8f、严格：0.4f	0~1.0f
occlusionNoseValue	鼻子遮挡阀值	宽松：0.95f、正常：0.8f、严格：0.4f	0~1.0f
occlusionMouthValue	嘴巴遮挡阀值	宽松：0.95f、正常：0.8f、严格：0.4f	0~1.0f
occlusionLeftContourValue	左脸颊遮挡阀值	宽松：0.95f、正常：0.8f、严格：0.4f	0~1.0f
occlusionRightContourValue	右脸颊遮挡阀值	宽松：0.95f、正常：0.8f、严格：0.4f	0~1.0f
occlusionChinValue	下巴遮挡阀值	宽松：0.95f、正常：0.8f、严格：0.4f	0~1.0f
headPitchValue	低头抬头角度	宽松：30、正常：20、严格：15	0~45
headYawValue	左右摇头角度	宽松：18、正常：18、严格：15	0~45
headRollValue	偏头角度	宽松：30、正常：20、严格：15	0~45
eyeClosedValue	闭眼阈值	0.7f	0~1.0f
cacheImageNum	图片缓存数量	3	建议 3~6

3.3 界面定制说明

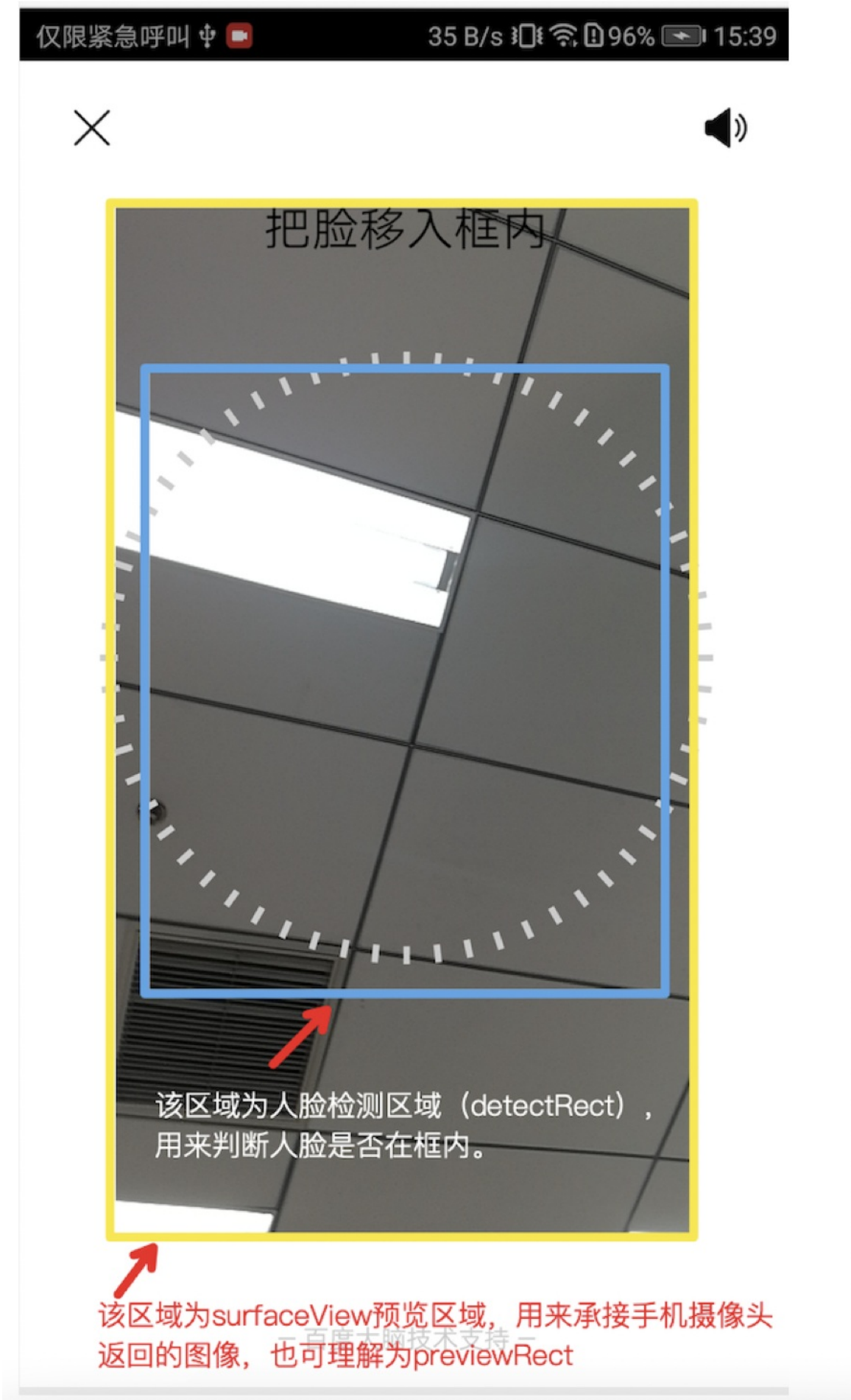
（1）如果SDK自带的UI和您的APP不统一，您可以自行修改FaceDetectRoundView。



（2）修改提示语音音频文件，有两种方式。 a、直接替换FaceUI工程raw下的mp3文件和string.xml。 b、FaceEnvironment 提供了setSoundId(FaceStatusNewEnum status, int soundId); 设置提示音资源。FaceStatusNewEnum为不同的状态。soundId为资源文件所对应的resource id。

3.4 采集中的预览区域和检测区域说明

如下图所示（图中背景设置为透明了，可以看出这两个区域）：



其中：

surfaceView宽高为屏幕的宽高； previewRect的宽高为手机摄像头的宽高； detectRect的宽高是以previewRect为基准计算出来的。 计算方式参照FaceDetectRoundView.java的Rect getPreviewDetectRect(int w, int pw, int ph)方法。 在onConfigCamera中，传入了previewRect和detectRect对象，可以在此处对其进行配置。

4 API 接口

4.1 安全增强级采集SDK初始化接口

安全增强级采集SDK的初始化接口。 如果在同意隐私协议前调用初始化接口，初始化过程会被中止，并在调用同意隐私协议方法后恢复。 在开始采集人脸的流程前，请保证调用了本初始化方法并同意了隐私协议，否则采集流程将无法启动。

```
void LH.init(Context context, String licenseId, String apiKey, String secretKey);
```

参数说明：
licenseId：Face为App分配的许可Id，在Face云端平台注册App时获取。
apiKey：百度AI开放平台授权的ApiKey，在Face云端平台注册App时获取。
secretKey：百度AI开放平台授权的secrecKey，在Face云端平台注册App时获取。

4.2 设置用户同意隐私协议接口

通过调用此接口告知安全增强级采集SDK用户是否同意了隐私协议。 在同意隐私协议前，所有对安全增强级采集SDK的调用均会失败，直到同意了隐私协议。

```
void LH.setAgreePolicy(Context context, boolean agree)
```

参数说明：
agree：为用户是否同意了隐私协议，true为同意，false为不同意。

4.3 人脸功能管理器FaceSdkManager

1. 获取实例 通过调用getInstance()获取人脸功能管理器的实例。

2. 人脸功能管理器初始化

   ```
   public void initialize(final Context context, String licenseID, String licenseFileName, IInitCallback callback)
   ```

   参数说明：
   licenseID： 传入申请License时获取的应用名称+_face_android后缀。
   licenseFileName： 鉴权文件名称。
   callback ：鉴权成功与否回调。

3. 设置人脸功能控制参数

   ```
   void setFaceConfig(FaceConfig config)
   ```

   参数说明： config：人脸功能控制参数对象。请按照4.1（3）初始化参数配置中的配置项按需进行配置。 **4.4 开始实名认证流程**

   开始实名认证流程。该方法是异步方法，调用后，安全增强级采集SDK自身执行人脸采集和实名认证逻辑。

```
void LH.startFaceVerify (Activity activity, SurfaceHolder previewSurfaceHolder, FaceProcessCallback processCallback, int deviceCheckTimeout, FaceVerifyInfo info, boolean liveness)
```

参数说明：
activity：执行人脸采集的Activity。
previewSurfaceHolder：用于展示摄像头预览的SurfaceView的Holder。
processCallback：流程回调，见“FaceProcessCallback接口”。
deviceCheckTimeout：安全检测超时时间，传入正数时，若在该时间内（单位：秒）未完成检测，则跳过。传入-1跳过检测，传入0后台检测。
info：实名认证信息，包含身份证号，姓名等信息。
liveness：是否进行活体检测。

4.5开始人脸比对流程

开始人脸比对流程。该方法是异步方法，调用后，安全增强级采集SDK自身执行人脸采集和人脸比对逻辑。

```
void LH.startFaceCompare (Activity activity, SurfaceHolder
previewSurfaceHolder, FaceProcessCallback processCallback，int deviceCheckTimeout，FaceCompareInfo info，boolean
liveness)
```

参数说明：

activity：执行人脸采集的Activity。

previewSurfaceHolder：用于展示摄像头预览的SurfaceView的Holder。

processCallback：流程回调，见“FaceProcessCallback接口”。

deviceCheckTimeout：安全检测超时时间，传入正数时，若在该时间内（单位：秒）未完成检测，则跳过。传入-1跳过检测，传入0后台检测。

info：人脸比对信息，包含比对用图片等信息。

liveness：是否进行活体检测。

4.6 取消正在进行的流程

取消正在进行的人脸比对或实名认证流程。调用此接口后，安全增强级采集SDK取消整个流程。请在Activity的onPause中调用此方法，保证界面离开顶端时流程被取消。下次进入界面应当重新开始人脸验证过程。

```
void LH.cancelFaceProcess()
```

4.7 采集过程中设置语音开关

设置是否开启人脸采集过程的语音提示。只有采集过程中可以通过此方法开启或关闭语音提示。采集前调用此方法不会生效。采集前若要开启或关闭全局语音提示，请在Face采集SDK的配置项中进行配置。

```
void LH.setSoundEnable (boolean enable)
```

4.8 FaceProcessCallback接口

回调类，回调方法顺序随人脸采集流程自上向下：

1. **onBegin ()**
   必有。
   回调流程开始事件。

2. **onDeviceCheckResult (int status)**
   回调设备风险检测结果。无安全插件会认为是未知。无风险或未知会进入人脸采集流程。
   若有风险，则不会进行人脸采集流程。此回调方法return后，会直接回调onEnd方法。
   若在云端配置了采集后置摄像头的功能，则在本回调开始，会将后置摄像头的图像预览视频流输出到传入的SurfaceHolder上。请从此处开始保证SurfaceView处于可见状态。
   status：1代表无风险，-1代表有风险。

3. **onBeginCollectFaceInfo();**
   将要开始采集人脸信息，此方法回调后，请保证SurfaceView处于可见状态。

4. **onConfigCamera(Camera camera，Rect previewRect，Rect detectRect)**
   此时已经开启前置摄像头，但还未开始预览。此回调方法中可以对camera进行配置，并可以对人脸识别模块的预览Rect和检测Rect进行设置。此回调方法返回后，将开始预览前置摄像头图像。

5. **onCollectCompletion(FaceStatusNewEnum status, String message, HashMap<String, ImageInfo> base64ImageCropMap, HashMap<String, ImageInfo> base64ImageSrcMap,int currentLivenessCount);**

回调人脸采集过程的阶段性结果，供APP刷新界面。

status：按FaceStatusNewEnum的枚举值返回的当前状态。

message：需要在界面展示的提示。

base64ImageCropMap：完成时的抠图图像信息。只有status为OK时才会包含此信息。

base64ImageSrcMap：完成时的原图图像信息。只有status为OK时才会包含此信息。

currentLivenessCount：当前完成的活体验证步骤数，用于界面展示验证进度。

6. **void onBeginRequestRemote();**

   开始向服务器发起实名认证/人脸比对请求。

   此方法回调后，SurfaceView可以处于不可见状态。

7. **onEnd (int status , String resultJson)**

   必有，包括被Cancel的情况。

   回调流程结束事件。

   status：1代表正常结束， <0的值均表示未正常结束。

   resultJson：云端返回的结果。只有status为1时此字段有值。其他情况为空字符串。

onEnd的部分status值列举如下：

1：正常结束，返回云端验证结果。

-1：已经有一个采集验证流程在运行。

-2：云端验证过程异常。

-3：风控验证失败。

-4：更严格情形下的风控验证失败。

-5：摄像头异常。

-6：流程被取消。

-7：线程异常。

-8：筛选图像异常。

-9：采集前流程异常。

-10：活体验证步骤异常。

-11：预览异常。

-12：采集后流程异常。

-13：未初始化安全增强级采集SDK。

-14：未同意隐私协议。

8. **void setCurrentLiveType(LivenessTypeEnum livenessType);**

9. **void viewReset();**

10. **void animStop();**

11. **void setFaceInfo(FaceExtInfo faceExtInfo);** 此4个回调方法为Face采集SDK的ILivenessViewCallback接口的原有方法，本SDK在接到这些回调时会透传给本Callback类的实现。

4.9 实名认证信息类FaceVerifyInfo

实名认证信息类，请调用以下构造方法获取实例。 FaceVerifyInfo(String idCardNumber, String name, int verifyType, String nation, FaceEnum.LivenessControl livenessControl, FaceEnum.SpoofingControl spoofingControl, FaceEnum.QualityControl qualityControl, String phoneNumber)

参数含义：

idCardNumber：证件号。

name：姓名，不必进行编码。组织数据时会对此字段进行URLEncode。

verifyType：证件类型，0:大陆身份证(default);1:港澳居民来往内地通行证;2:外国人永久居留身份证;3:定居国外的中国公民护照。

nation：国家或地区三位大写英文字母缩写。当verify_type为1、2、3时填写国籍3位(国家名英文缩写，除港澳以外参照

ISO3166 标准，定居国外的中国公民和港澳居民使用 CHN) **livenessControl**：活体控制配置项

spoofingControl：合成图控制配置项

qualityControl：质量控制配置项

phoneNumber：需要风控的电话号码

4.10 人脸比对信息类FaceCompareInfo

人脸比对信息类，请调用以下构造方法获取实例。 **FaceCompareInfo(FaceEnum.QualityControl qualityControl, FaceEnum.LivenessControl livenessControl, FaceEnum.FaceType faceType, int faceSortType, String registerImage, FaceEnum.ImageType registerImageType, FaceEnum.FaceType registerFaceType, FaceEnum.QualityControl registerQualityControl, FaceEnum.LivenessControl registerLivenessControl, String phoneNumber)**

参数含义：

qualityControl：质量控制配置项

livenessControl：活体控制配置项

faceType：人脸类型 **faceSortType**：人脸检测排序类型。 0:代表检测出的人脸按照人脸面积从大到小排列；1:代表检测出的人脸按照距离图片中心从近到远排列

registerImage：比对用图片信息

registerImageType：比对用图片类型

registerFaceType：比对用人脸类型

registerQualityControl：比对用质量控制配置项

registerLivenessControl：比对用活体控制配置项

phoneNumber：需要风控的电话号码

4.11 枚举类FaceEnum

FaceVerifyInfo和FaceCompareInfo中使用的枚举类均在FaceEnum中定义。

public enum LivenessControl { NONE, // 不进行控制 LOW, // 较低的活体要求(高通过率 低攻击拒绝率) NORMAL, // 一般的活体要求(平衡的攻击拒绝率, 通过率) (default) HIGH; // 较高的活体要求(高攻击拒绝率 低通过率) }

public enum SpoofingControl { NONE, // 不进行控制 LOW, // 较低的合成图检测要求(高通过率 低攻击拒绝率) NORMAL, // 一般的合成图检测要求(平衡的攻击拒绝率, 通过率) (default) HIGH; // 较高的合成图检测要求(平衡的攻击拒绝率, 通过率) }

public enum QualityControl { NONE, // 不进行控制(default) LOW, // 较低的质量要求 NORMAL, // 一般的质量要求 HIGH; // 较高的质量要求 }

public enum FaceType { LIVE, // 表示生活照：通常为手机、相机拍摄的人像图片、或从网络获取的人像图片等 (default) IDCARD, // 表示身份证芯片照：二代身份证内置芯片中的人像照片 WATERMARK, // 表示带水印证件照：一般为带水印的小图，如公安网小图 CERT, // 表示证件照片：如拍摄的身份证、工卡、护照、学生证等证件图片 INFRARED; // 表示红外照片：使用红外相机拍摄的照片 }

public enum ImageType { BASE64, // 图片的base64值，base64编码后的图片数据，编码后的图片大小不超过2M URL, // 图片的 URL地址(可能由于网络等原因导致下载图片时间过长) FACE_TOKEN; // 人脸图片的唯一标识，调用人脸检测接口时，会为每个人脸图片赋予一个唯一的FACE_TOKEN，同一张图片多次检测得到的FACE_TOKEN是同一个 }

5 多进程支持 安全增强级采集SDK支持在指定进程中运行。注意，负责采集和验证Activity应该与安全增强级采集SDK的配置进程在同一进程。

a) 如果需要配置安全增强级采集SDK运行在其它进程，请将如下所有组件都通过 Android:process属性配置在同一个进程中。

```
<activityandroid:name="com.baidu.liantian.LiantianActivity"android:exported="false"android:theme="@android:style/Theme
><intent-filter><action android:name="com.baidu.action.Liantian.VIEW"/><category
android:name="com.baidu.category.liantian"/><category android:name="android.intent.category.DEFAULT"/></intent-
filter></activity><serviceandroid:name="com.baidu.liantian.LiantianService"android:exported="false"><intent-filter>
<action android:name="com.baidu.action.Liantian.VIEW"/><category android:name="com.baidu.category.liantian"/>
<category android:name="android.intent.category.DEFAULT"/></intent-filter></service><provider android:authorities="应
用包名.liantian.ac.provider" android:name="com.baidu.liantian.LiantianProvider"
android:exported="false"tools:replace="android:authorities"/>
```

b) 请不要为本SDK的provider配置android:multprocess属性。 c) 为了给APP提供更安全的采集和验证流程，请尽可能在APP主进程初始化sdk和进行采集验证。

IOS

1、简介

1.1 功能介绍

百度安全增强级采集 SDK iOS 版是一种面向 iOS 移动设备的人脸技术开发包，此版SDK包含人脸检测、活体识别等功能。基于该方案，开发者可以轻松的构建包含人脸检测、采集和活体识别的应用。在您使用SDK之前，我们首先为您介绍以下SDK本身及相关人脸能力，以便您能更方便使用人脸服务。

此版SDK基于在线的安全能力封装为人脸比对和实名认证两个核心流程，主要用于在客户端执行这两种流程。因为这两种流程最终需要向服务器发起请求以完成认证，因此无法在离线环境下使用。

此版SDK所包含的能力如下：

- **动作活体检测**：通过让用户做出指定人脸配合式的交互动作，识别当前操作者是否为活体，此功能为离线使用，可设定指定动作是否使用及应用顺序。动作包含：眨眼、张嘴、左摇头，右摇头，向上抬头，向下低头六个。可有效抵御高清图片、3D建模、视频等攻击。
- **人脸质量检测**：判断视频流中的图片帧中，哪些图片质量较佳，即人脸图像特征清晰（满足遮挡、姿态、光照、模糊度等校验）。
- **人脸图像采集**：通过本地SDK能力，采集人脸图像，同时经过人脸质量检测，确保采集到的人脸图像符合各条件校验（遮挡、姿态、光照、模糊度等），为设备前端获取有效可分析人脸的主要功能。
- **授权**：SDK的授权判断，授权介质也称为license，在SDK使用中，需要通过license向授权服务器发起请求，判断SDK的使用合法性及使用有效期。
- **人脸比对和实名认证流程**：通过将原子操作封装为安全的流程，在流程中加入多种安全防护能力，能够抵御大多数针对人脸流程的攻击。因为不用自行维护流程，应用程序使用人脸比对和实名认证功能也变得更加简便。

为了方便您的开发，我们已经为您准备了多种场景的示例工程，您可以根据业务需要，在后台进行直接下载，目前支持【人脸核身】

【人脸闸机/门禁】【人脸登录/考勤】【多人脸识别】,示例工程参考下图：

1.2 兼容性

- **系统**：支持iOS9以上系统。需要开发者通过Deployment Target 来保证支持系统的检测。
- **机型**：手机和平板皆可（暂不支持横屏）
- **架构**：arm64、armv7
- **网络**：支持 WIFI 及移动网络,移动网络支持使用 NET 网关及WAP 网关(CMWAP、CTWAP、UNIWAP、3GWAP)。

1.3 开发包说明

文件/文件夹名	说明
FaceSDK	FaceSDK 包、bundle 资源文件、bundle 模型文件、鉴权文件
Public/Common	视频流处理类
Public/Utils	图像处理类
UI/Controller	DEMO工程
UI/View	View控制类
FaceParameterConfig.h	配置信息

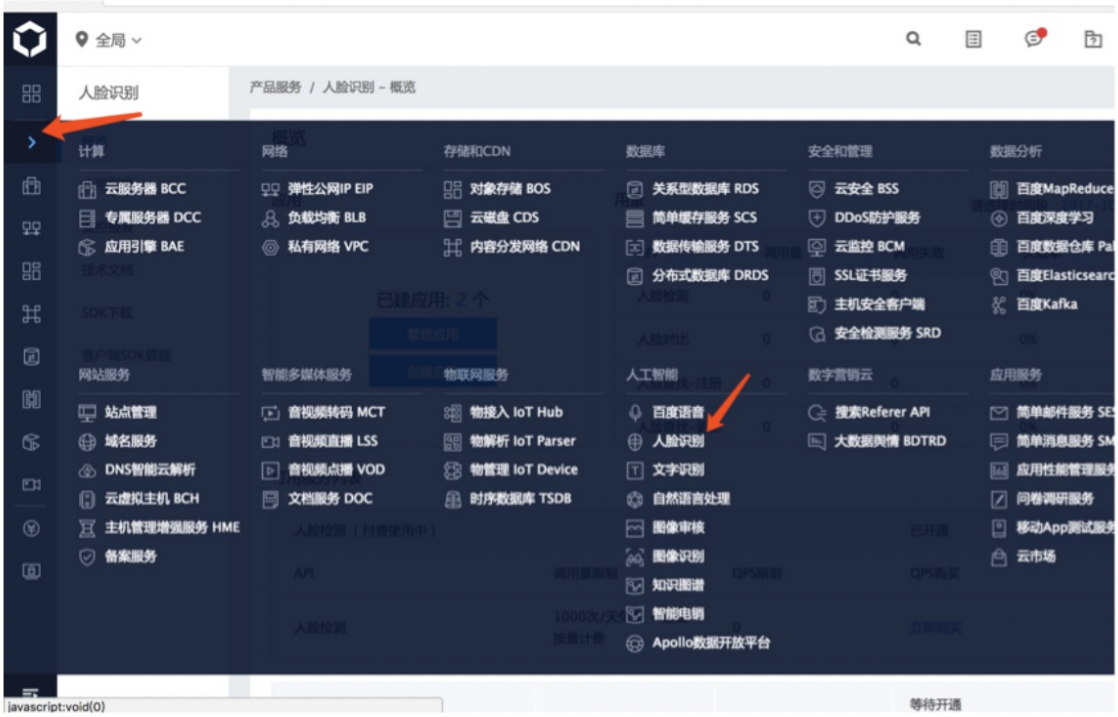
2、集成指南

2.1 准备工作

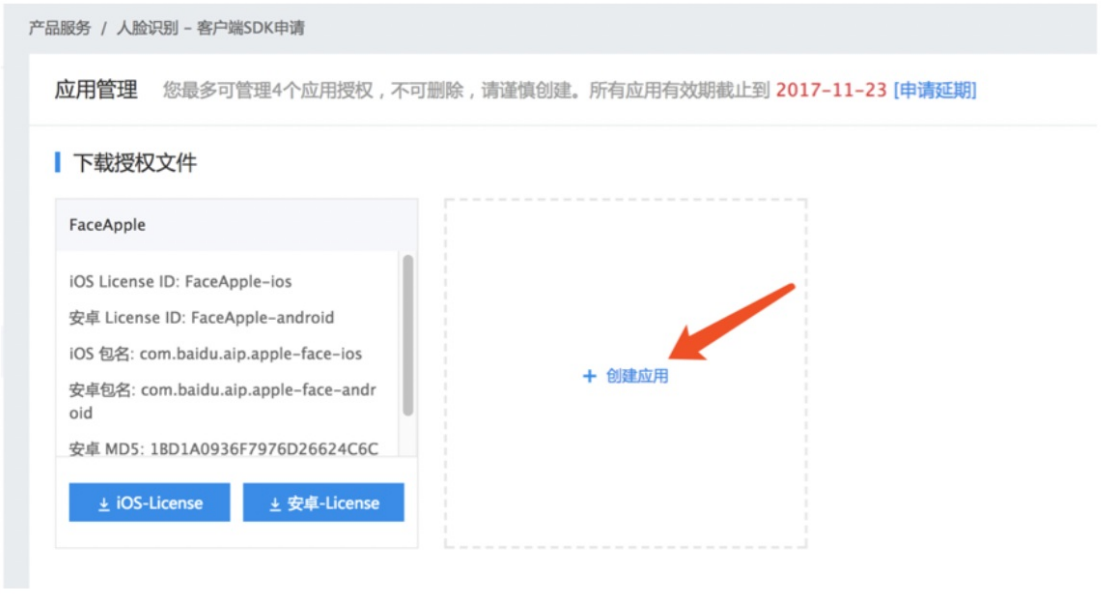
2.1.1 申请license

人脸SDK License：此license用于SDK离线功能使用，在您的申请人脸SDK的后台页面，全局->产品服务->人脸识别->客户端SDK申请

人脸控制台路径如下：

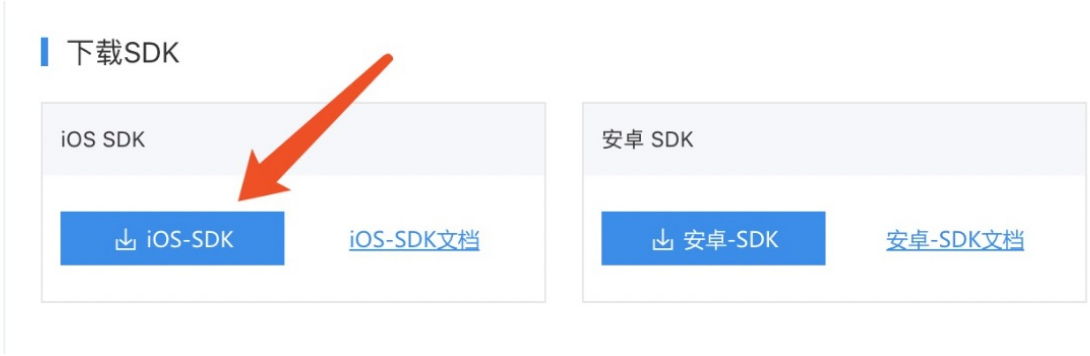


点击客户端SDK管理，弹出如下图：创建应用（这里创建应用是为了使用离线SDK，上面创建应用为了使用人脸在线接口，如注册、识别等）

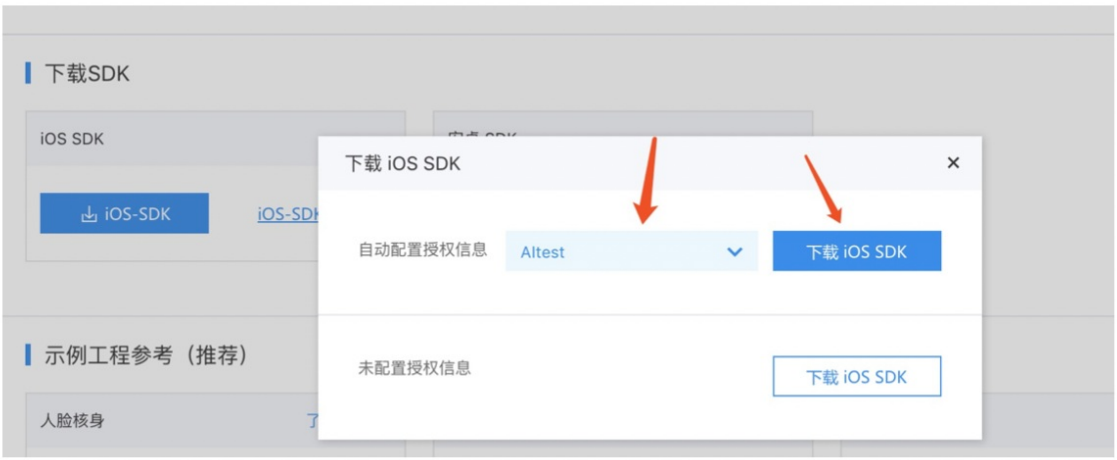


在弹出的框中输入授权标识，选择应用类型，应用系统，以及包名，详情请查看输入框右边提示

2.1.2 下载SDK



下载SDK分为自动配置授权信息（创建license后就可以选择为该应用，下载后SDK自动帮您配置授权，不用下载license拷贝到工程中，初始化参数licenseID,包名也帮您配置好了）和未配置授权信息两种方式：



2.2 运行示例工程

2.2.1 自动配置授权信息集成

如果您是通过自动配置授权信息下载的示例工程，只需配置好证书即可。查看下项目中的FaceParameterConfig.h文件，已经自动配置

```
11
12 // 人脸license文件名
13 #define FACE_LICENSE_NAME    @"idl-license"
14
15 // 人脸license后缀
16 #define FACE_LICENSE_SUFFIX  @"face-ios"
17
18 //  (您申请的应用名称(appname)+「-face-ios」后缀，如申请的应用名称(appname)为test123，则此处填写test123-face-ios)
19 #define FACE_LICENSE_ID        @"BaiduFaceDemo-face-ios"
20
```

示例图

配置好证书，即可运行。注意已经设置好的bundle id不要随意改动。

2.2.2 未使用自动配置授权信息的集成

手动导入授权信息。 需要手动导入license文件，配置好bundleID、licenseID等信息:

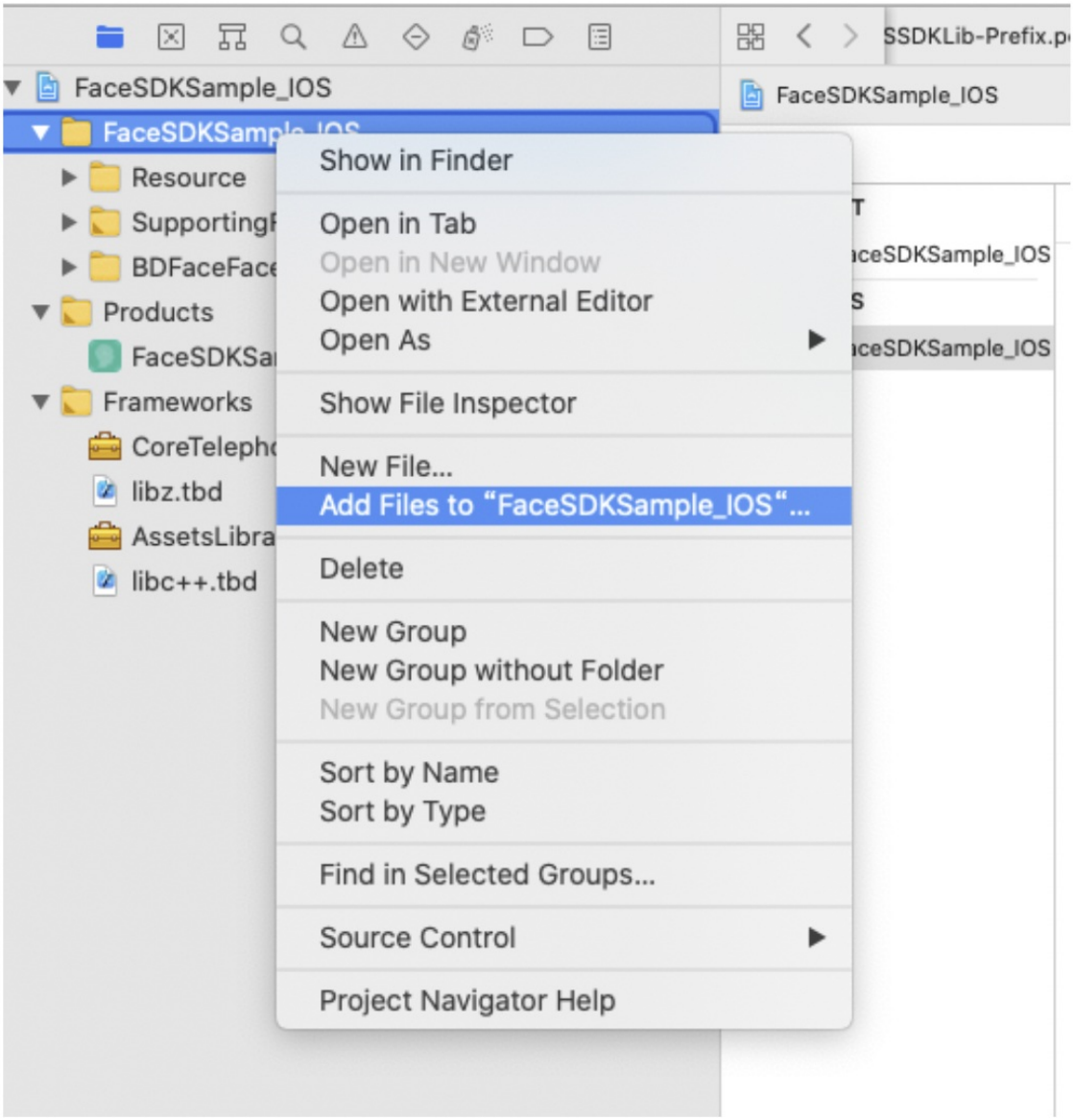
```
11
12 // 人脸license文件名
13 #define FACE_LICENSE_NAME    @"idl-license"
14
15 // 人脸license后缀
16 #define FACE_LICENSE_SUFFIX  @"face-ios"
17
18 //  (您申请的应用名称(appname)+「-face-ios」后缀，如申请的应用名称(appname)为test123，则此处填写test123-face-ios)
19 #define FACE_LICENSE_ID        @"BaiduFaceDemo-face-ios"
20
```

示例图

2.3 添加SDK到工程

1、 打开或者新建一个项目。

2、 右键点击项目，会出现一个添加菜单，在菜单中选择『Add Files to“此处是你的项目名字”…… 』,如下图所示：

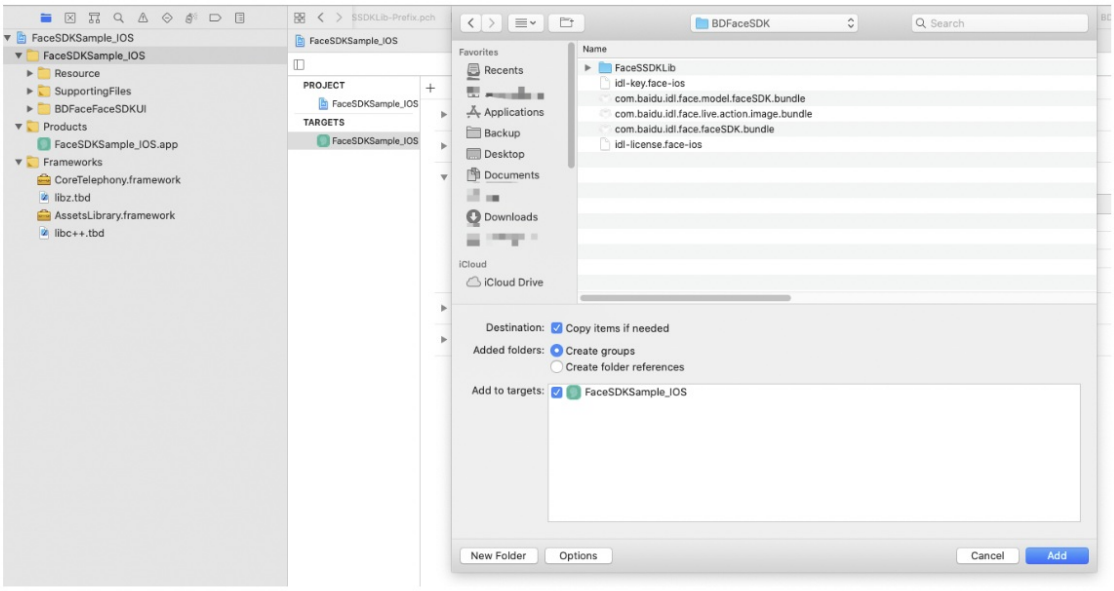


3、 在添加文件弹出框里面选择申请到的license和SDK添加进来。如下图：

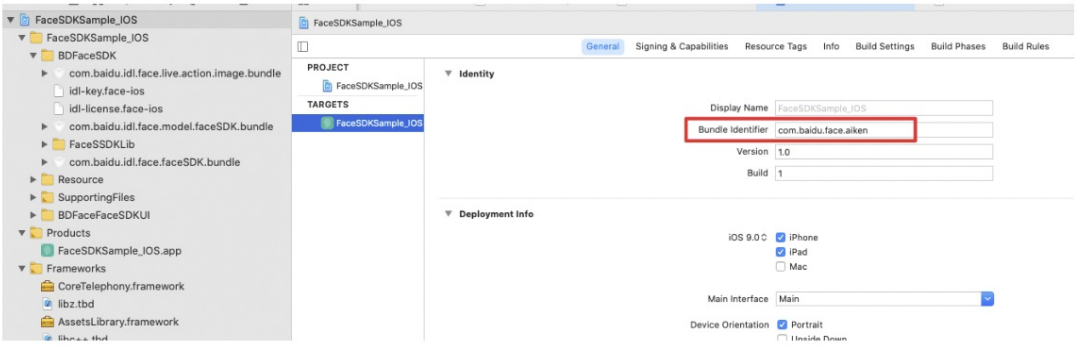
注意：license为百度官方提供的。

BDFaceSDK文件夹下包含下面文件:

- com.baidu.idl.face.faceSDK.bundle
- com.baidu.idl.face.live.action.image.bundle
- com.baidu.idl.face.model.faceSDK.bundle
- idl-key.face-ios
- idl-license.face-ios
- FaceSSDKLib（libFaceSSDKLib.a、SSDKLib.h、SSFaceDetectionManager.h、SSFaceSDK.h、SSFaceSDKManager.h）



4、确认下Bundle Identifier 是否是申请license时填报的那一个，注意：license和Bundle Identifier是一一对应关系，填错了会导致SDK不能用。



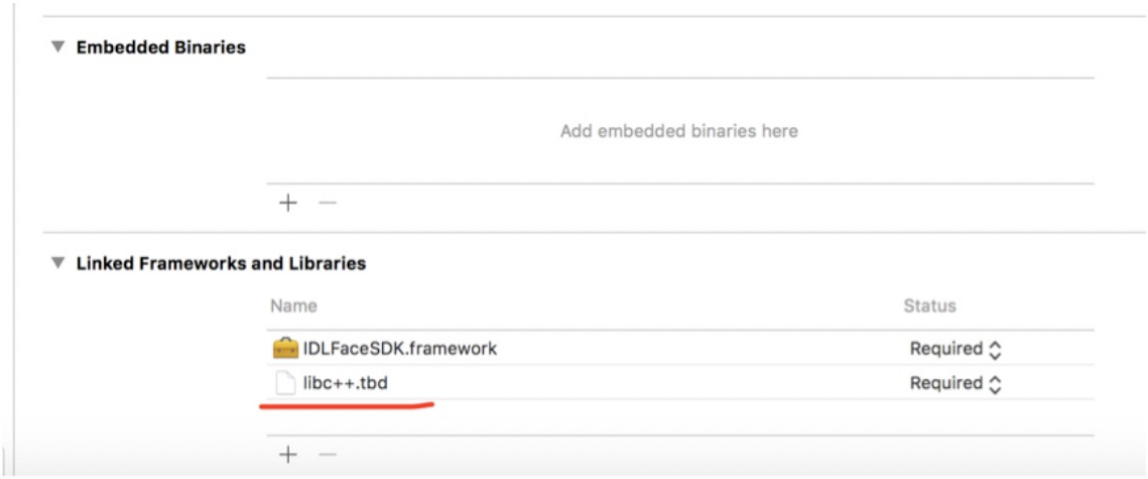
5、 填写正确的FACE_LICENSE_ID。

（即后台展示的LicenseID）

在FaceParameterConfig.h文件里面填写拼接好的FACE_LICENSE_ID。

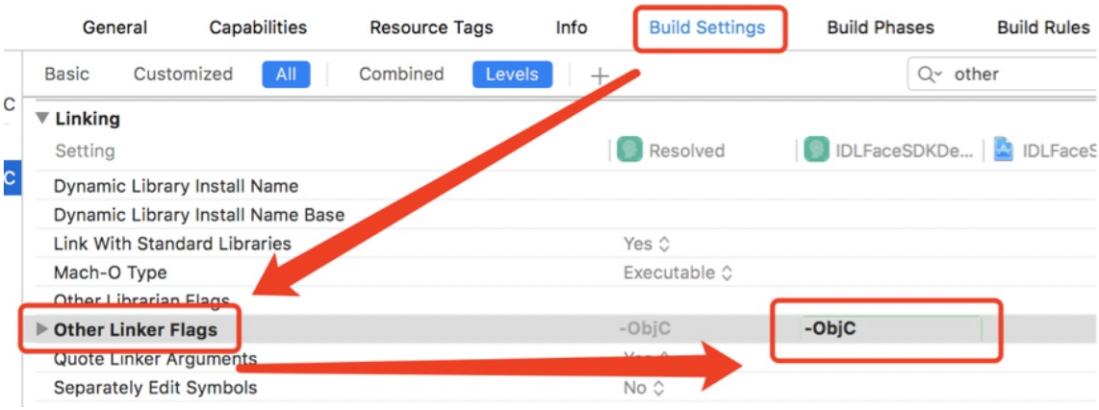
```
//  (您申请的应用名称(appname)+「-face-ios」后缀，如申请的应用名称(appname)为test123，则此处填写test123-face-ios)
#define FACE_LICENSE_ID        @"BaiduFaceDemo-face-ios"
```

6、 选择链接C++标准库。



7、 如果没有使用pod管理第三方库的话，请在Build Setting >Linking > Other Linker Flags 上面加入 –ObjC 选项。如果用了pod请忽

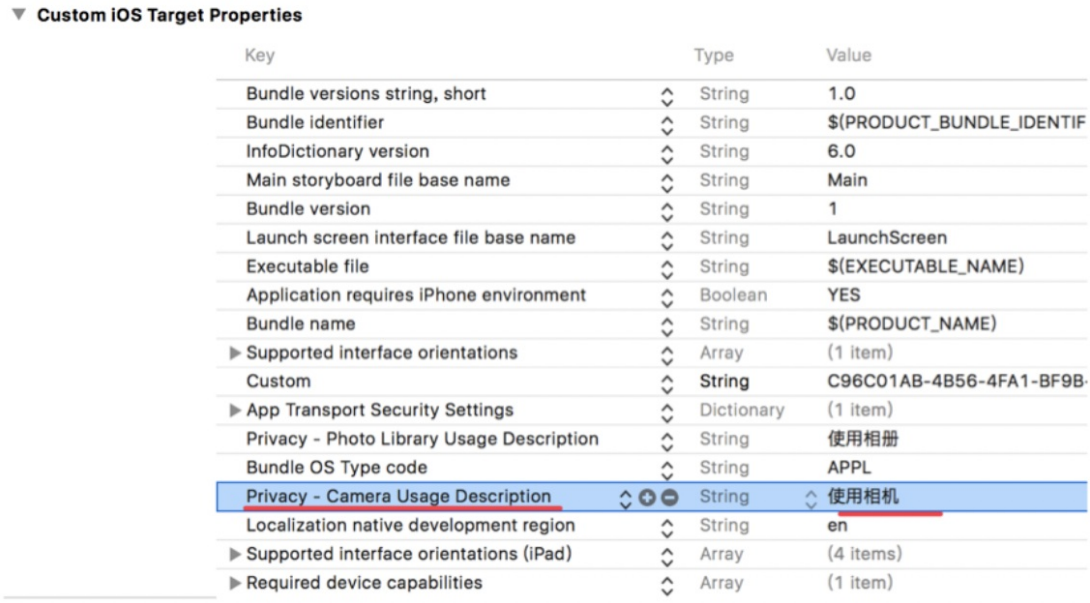
略，因为pod会自动添加上。



2.4 权限声明

需要使用相机权限：编辑Info.plist文件，添加

Privacy- Camera Usage Description 的Key值，Value为使用相机时候的提示语，可以填写：『使用相机』。



3、接口说明

3.1 FaceSDK 鉴权初始化

3.1.1 设置鉴权功能

```
 - (void)setLicenseID:(NSString *)licenseID andLocalLicenceFile:(NSString *)licensePath andRemoteAuthorize:(BOOL)remoteAuthorize;
```

参数：

- licenseId：平台申请的 licenseID
- localLicencePath：鉴权文件路径
- remoteAuthorize：是否开启网络鉴权

返回：

- 无

参考AppDelegate.m 实现，使用方法如下：

```
    NSString* licensePath = [[NSBundle mainBundle] pathForResource:FACE_LICENSE_NAME
ofType:FACE_LICENSE_SUFFIX];
    NSAssert([[NSFileManager defaultManager] fileExistsAtPath:licensePath], @"license文件路径不对，请仔细查看文档");
    [[SSFaceSDKManager sharedInstance] setLicenseID:FACE_LICENSE_ID andLocalLicenceFile:licensePath
andRemoteAuthorize:true];
```

3.1.2 鉴权成功的凭证

```
 - (BOOL)canWork
```

参数：

- 无

返回：

- True代表成功，false代表失败

参考ViewController.m 实现，判断鉴权是否通过：

```
 if (![[FaceSDKManager sharedInstance] canWork]){
  NSLog(@"授权失败，请检测ID 和 授权文件是否可用");
  return;
}
```

3.2 FaceSDK 功能初始化

3.2.1 SSFaceSDKManager参数配置

具体方法详见如下：

```
 /**
 * 获取版本号
 */
- (NSString *)getVersion;

/**
 *  重置计数器
 */
- (void)reset;

/**
 * 获取设备zid 公安验证上传
 */
- (NSString *)getZtoken;

/**
 *  SDK鉴权方法-文件授权
 *  SDK鉴权方法 必须在使用其他方法之前设置，否则会导致SDK不可用
 *
 *  @param licenseID 授权ID
 *  @param licensePath 本地鉴权文件路径
 *  @param remoteAuthorize 是否远程更新过期鉴权文件
 */
- (void)setLicenseID:(NSString *)licenseID andLocalLicenceFile:(NSString *)licensePath andRemoteAuthorize:
(BOOL)remoteAuthorize;

/**
 *  SDK云端校验设置
 *  需要云端校验，需提前申请id和secret
```

```
 * 需要云端校验，需提前申请Id和secret
 *
 * @param clientId api key
 * @param clientSecret api secret
 */
- (void)setBCEClientId:(NSString *)clientId clientSecret:(NSString *)clientSecret;

/**
 * 初始化采集功能
 */
- (int) initCollect;

/**
 * 卸载采集功能
 */
- (int)uninitCollect;

/**
 * 判断授权是否通过，true 表示通过，false 表示不通过
 */
- (BOOL)canWork;

/**
 * 设置预测库耗能模式
 * 默认 LITE_POWER_NO_BIND
 */
- (void)setLitePower:(int)litePower;

/**
 * 需要检测的最大人脸数目
 * 默认1
 */
- (void)setMaxDetectNum:(int)detectNum ;

/**
 * 需要检测的最小人脸大小
 * 默认40
 */
- (void)setMinFaceSize:(int)width;

/**
 * 人脸置信度阈值（检测分值大于该阈值认为是人脸
 * RGB
 * 默认 0.5f
 */
- (void)setNotFaceThreshold:(CGFloat)thr ;

/**
 * 质量检测遮挡阈值
 * 默认0.5
 */
- (void)setOccluThreshold:(CGFloat)thr ;

/**
 * 质量检测遮挡阈值-左眼遮挡置信度
 * 默认0.31
 */
-(void)setOccluLeftEyeThreshold:(CGFloat)thr ;

/**
 * 质量检测遮挡阈值- 右眼遮挡置信度
 * 默认0.31
 */
```

```
-(void)setOccluRightEyeThreshold:(CGFloat)thr；

/**
 * 质量检测遮挡阈值-鼻子遮挡置信度
 * 默认0.27
 */
-(void)setOccluNoseThreshold:(CGFloat)thr；

/**
 * 质量检测遮挡阈值-嘴巴遮挡置信度
 * 默认0.2
 */
-(void)setOccluMouthThreshold:(CGFloat)thr；

/**
 * 质量检测遮挡阈值-左脸遮挡置信度
 * 默认0.48
 */
-(void)setOccluLeftCheekThreshold:(CGFloat)thr；

/**
 * 质量检测遮挡阈值-右脸遮挡置信度
 * 默认0.48
 */
-(void)setOccluRightCheekThreshold:(CGFloat)thr；

/**
 * 质量检测遮挡阈值-下巴遮挡置信度
 * 默认0.4
 */
-(void)setOccluChinThreshold:(CGFloat)thr；


/**
 * 最大光照阈值
 */
- (void)setMaxIllumThreshold:(CGFloat)thr;

/**
 * 最小光照阈值
 */
- (void)setMinIllumThreshold:(CGFloat)thr；

/**
 *  质量检测模糊阈值
 *  默认0.5
 */
- (void)setBlurThreshold:(CGFloat)thr；

/**
 *  姿态检测阈值
 *  默认pitch=12，yaw=12，row=10
 */
- (void)setEulurAngleThrPitch:(float)pitch yaw:(float)yaw roll:(float)roll；

/**
 * 输出图像个数
 * 默认3
 */
- (void)setMaxCropImageNum:(int)imageNum；

/**
```

```
 * 输出图像宽，设置为有效值(大于0)则对图像进行缩放，否则输出原图抠图结果
 * 默认 480
 */
- (void)setCropFaceSizeWidth:(CGFloat)width；

/**
 * 输出图像高，设置为有效值(大于0)则对图像进行缩放，否则输出原图抠图结果
 * 默认 680
 */
- (void)setCropFaceSizeHeight:(CGFloat)height；

/**
 * 输出图像，下巴扩展，大于等于0，0：不进行扩展
 * 默认0.1
 */
- (void)setCropChinExtend:(CGFloat)chinExtend；

/**
 * 输出图像，额头扩展，大于等于0，0：不进行扩展
 * 默认0.2
 */
- (void)setCropForeheadExtend:(CGFloat)foreheadExtend；

/**
 * 输出图像，人脸框与背景比例，大于等于1，1：不进行扩展
 * 默认1.5f
 */
- (void)setCropEnlargeRatio:(float)cropEnlargeRatio;

/**
 * 动作超时配置
 */
- (void)setConditionTimeout:(CGFloat)timeout；

/**
 * 语音间隔提醒配置
 */
- (void)setIntervalOfVoiceRemind:(CGFloat)timeout;

/**
 * 是否开启口罩检测，非动作活体检测模型true，动作活体检测模型false
 */
- (void)setIsCheckMouthMask:(BOOL)isCheck;

/**
 * 口罩检测阈值配置，默认0.8。
 * 大于阈值判定为戴口罩，低于阈值判定为未戴口罩
 */
- (void)setMouthMaskThreshold:(CGFloat)thr；

/**
 * 设置原始图片缩放比例，默认1不缩放，scale 阈值0~1
 */
- (void)setImageWithScale:(CGFloat)scale;

/**
 * 设置图片加密类型，type=0 基于base64 加密；type=1 基于百度安全算法加密
 */
- (void)setImageEncrypteWithType:(int) type;

/**
 *  人脸过远框比例 默认：0.4
```

```
 */
- (void)setMinRect:(float) minRectScale;
/**
 * 采集动作验证
 * @param image 检测的图片
 * @param isOriginal 是否返回原始图片
 * @param completion 判断采集是否完成，人脸信息状态是否正常
 */
- (void)detectWithImage:(UIImage *)image isRreturnOriginalValue:(BOOL) isOriginal completion:(void (^)(FaceInfo *faceinfo,
ResultCode resultCode))completion;

/**
 * 动作活体动作验证
 * @param image 检测的图片
 * @param actionLiveType 当前要求做的动作
 * @param completion 判断当前动作是否完成，人脸信息状态是否正常
 */
- (void)livenessWithImage:(UIImage *)image withAction:(FaceLivenessActionType)actionLiveType completion:(void (^)
(FaceInfo *faceinfo, FaceLivenessState *state, ResultCode resultCode))completion;

/**
 * 设置活体动作
 * @param array 包含活体动作种类
 * @param order 是否顺序执行
 * @param numberOfLiveness 活体数量
 */
- (void)livenesswithList:(NSArray *)array order:(BOOL)order numberOfLiveness:(NSInteger)numberOfLiveness;

/**
 * 活体检测过程中，返回活体总数，当前成功个数，当前活体类型
 */
-(void)livenessProcessHandler:(LivenessProcess)processHandler;
```

3.2.2 FaceSDK 初始化

```
 - (int)initCollect;
```

参数:

- 无

返回：

- 无

参考ViewController.m initSDK 方法实现：

```
- (void) initSDK {
    if (![[SSFaceSDKManager sharedInstance] canWork]){
        NSLog(@"授权失败，请检测ID 和 授权文件是否可用");
        return;
    }

    // 初始化SDK配置参数，可使用默认配置
    // 设置最小检测人脸阈值
    [[SSFaceSDKManager sharedInstance] setMinFaceSize:200];
    // 设置截取人脸图片高
    [[SSFaceSDKManager sharedInstance] setCropFaceSizeWidth:480];
    // 设置截取人脸图片宽
    [[SSFaceSDKManager sharedInstance] setCropFaceSizeHeight:640];
    // 设置人脸遮挡阈值
    [[SSFaceSDKManager sharedInstance] setOccluThreshold:0.5];
    // 设置亮度阈值
    [[SSFaceSDKManager sharedInstance] setMinIllumThreshold:40];
    [[SSFaceSDKManager sharedInstance] setMaxIllumThreshold:240];
    // 设置图像模糊阈值
    [[SSFaceSDKManager sharedInstance] setBlurThreshold:0.3];
    // 设置头部姿态角度
    [[SSFaceSDKManager sharedInstance] setEulurAngleThrPitch:10 yaw:10 roll:10];
    // 设置人脸检测精度阈值
    [[SSFaceSDKManager sharedInstance] setNotFaceThreshold:0.6];
    // 设置抠图的缩放倍数
    [[SSFaceSDKManager sharedInstance] setCropEnlargeRatio:2.5];
    // 设置照片采集张数
    [[SSFaceSDKManager sharedInstance] setMaxCropImageNum:3];
    // 设置超时时间
    [[SSFaceSDKManager sharedInstance] setConditionTimeout:15];
    // 设置开启口罩检测，非动作活体检测可以采集戴口罩图片
    [[SSFaceSDKManager sharedInstance] setIsCheckMouthMask:true];
    // 设置开启口罩检测情况下，非动作活体检测口罩过滤阈值，默认0.8 不需要修改
    [[SSFaceSDKManager sharedInstance] setMouthMaskThreshold:0.8f];
    // 设置原始图缩放比例
    [[SSFaceSDKManager sharedInstance] setImageWithScale:0.8f];
    // 初始化SDK功能函数
    [[SSFaceSDKManager sharedInstance] initCollect];
    // 设置人脸过远框比例
    [[SSFaceSDKManager sharedInstance] setMinRect:0.4];

    /// 设置用户设置的配置参数
    [BDFaceAdjustParamsTool setDefaultConfig];
}
```

3.2.3 FaceSDK 释放

```
- (int)uninitCollect
```

参数:

- 无

返回：

- 无

3.3 人脸采集和活体识别

人脸采集和活体识别由SSFaceDetectionManager类实现，类结构如下：

```
 /**
 * 流程返回结果类型
 */
typedef NS_ENUM(NSInteger, BDFaceCompletionStatus) {
    BDFaceCompletionStatusSuccess = 1,  // 成功
    BDFaceCompletionStatusNoRisk = 2,   // 无风险
    BDFaceCompletionStatusImagesSuccess = 3, // 图像采集成功

    BDFaceCompletionStatusIsRunning = -1,   // 正在采集图像
    BDFaceCompletionStatusResultFail = -2,  // 云端服务执行失败
    BDFaceCompletionStatusIsRiskDevice = -3,    // 风险设备
    BDFaceCompletionStatusCameraError = -5,    // 没有授权镜头
    BDFaceCompletionStatusTimeout = -6, // 超时
    BDFaceCompletionStatusCancel = -7, // 取消
    BDFaceCompletionStatusSDKNotInit = -13, // SDK未初始化
    BDFaceCompletionStatusLicenseFail = -15, // 授权错误
    BDFaceCompletionStatusNetworkError = -16,    // 网络错误
};

@protocol SSCaptureDataOutputProtocol <NSObject>

// 帧图像回传，用于刷新UI使用
- (void)captureOutputSampleBuffer:(UIImage *)image;

// 本次人脸流程回调
- (void)faceSessionCompletionWithStatus:(BDFaceCompletionStatus)status result:(NSDictionary *)result;

// 活体检测状态
- (void)livenessActionDidFinishWithCode:(LivenessRemindCode)code;

// 人脸识别检测
- (void)detectionActionDidFinishWithCode:(DetectRemindCode)code;

@end
@interface SSFaceDetectionManager : NSObject

// 图像返回帧处理代理
@property (nonatomic, weak) id<SSCaptureDataOutputProtocol> delegate;

// 采集流程运行状态
@property (nonatomic, assign, readonly) BOOL runningStatus;

// 风险检测超时时间，默认3秒
@property (nonatomic, assign) NSInteger riskDetectionSetting;

// AVCaptureSessionPreset类型枚举，支持低版本所以使用NSString
@property (nonatomic, copy) NSString *sessionPresent;

// 采集图像区域
@property (nonatomic, assign) CGRect previewRect;

// 探测区域
@property (nonatomic, assign) CGRect detectRect;

// 是否开启声音提醒
@property (nonatomic, assign) BOOL enableSound;

// 返回图片类型
@property (nonatomic, assign) BDFaceOutputImageType outputImageType;


+ (instancetype)sharedInstance;
```

```
+ (instancetype)sharedInstance;

- (void)connectPreviewLayer:(AVCaptureVideoPreviewLayer*)pLayer;
/**
 * 创建用于实名认证的参数
 */
- (NSDictionary *)creatFaceVerifyParameters:(NSString *)idCardNumber
                                       name:(NSString *)name
                                 verifyType:(FaceIdCardType)cardType
                                     nation:(NSString *)nation
                                phoneNumber:(NSString *)phoneNumber
                             livenessControl:(FaceLivenessControlType)livenessControl
                             spoofingControl:(FaceSpoofingControlType)spoofingControl
                              qualityControl:(FaceQualityControlType)qualityControl;

/**
 * 创建用于人脸比对的参数
 */
- (NSDictionary *)createFaceMatchParametersWithRegisterImage:(NSString *)registerImageBase64
                                          registerImageType:(FaceRegisterImageType)registerImageType
                                          registerFaceType:(FaceFaceType)registerFaceType
                                                faceType:(FaceFaceType)faceType
                                            faceSortType:(FaceSortype)faceSortType
                                             phoneNumber:(NSString *)phoneNumber
                                          livenessControl:(FaceLivenessControlType)livenessControl
                                           qualityControl:(FaceQualityControlType)qualityControl
                                    registerLivenessControl:(FaceLivenessControlType)registerLivenessControl
                                     registerQualityControl:(FaceQualityControlType)registerQualityControl;


/**
 * 开始当前人脸校验流程
 * @param detectionType 业务流程，采集信息用于实名认证、人脸比对
 * @param parameters 流程需要的参数
 * @param flowType 操作流程，人脸采集、人脸活体
 * @param vc 用于进行人脸信息采集的ViewController
 */
- (void)startSessionWithType:(BDFaceResultReportType)detectionType parameters:(NSDictionary *)parameters faceFlow:
(BDFaceFlowType)flowType viewController:(UIViewController *)vc;

/**
 * 取消当前人脸校验流程
 */
- (void)cancel;

@end
```

3.3.1 流程介绍

整体流程如下：

- 通过createFaceVerify/Match 两个方法，创建用于进行人脸识别和人脸比对的参数
- 调用startSessionWithType方法启动人脸流程
- SSCaptureDataOutputProtocol代理的方法会收到回调，来通知具体步骤和结果
- 最终在faceSessionCompletionWithStatus回调方法中通知人脸过程成功或失败

具体过程参数和结果中的枚举，请参考.h头文件中的注释，及Demo中的代码。

3.3.2 参数介绍

previewRect与detctRect是为了做距离检测而定义的，为了在上层封装判断脸是否在框内／离太远／离太近。

- delegate：处理在镜头打开过程中的回调，包括视频图像和人脸识别的状态结果
- previewRect： 人脸图片大小，间接定义的最大距离的 maxRect 和最小距离的 minRect。
- detectRect：人脸检测区域大小，实际采集区域
- riskDetectionSetting：风险检测超时时间，默认3秒，设置为0时跳过风险检测
- sessionPresent：视频流输出格式，使用AVCaptureSessionPreset类型枚举
- enableSound：采集过程中是否开启声音通知
- outputImageType：返回图片类型，可以返回原图或抠图

3.3.3 视频流展示方式

视频流返回后有两种UI展现方式：

（1）UIImage展示

- (void)captureOutputSampleBuffer:(UIImage *)image;通过此方法回调的UIImage，渲染到主屏幕，实时在主线程刷新即可。

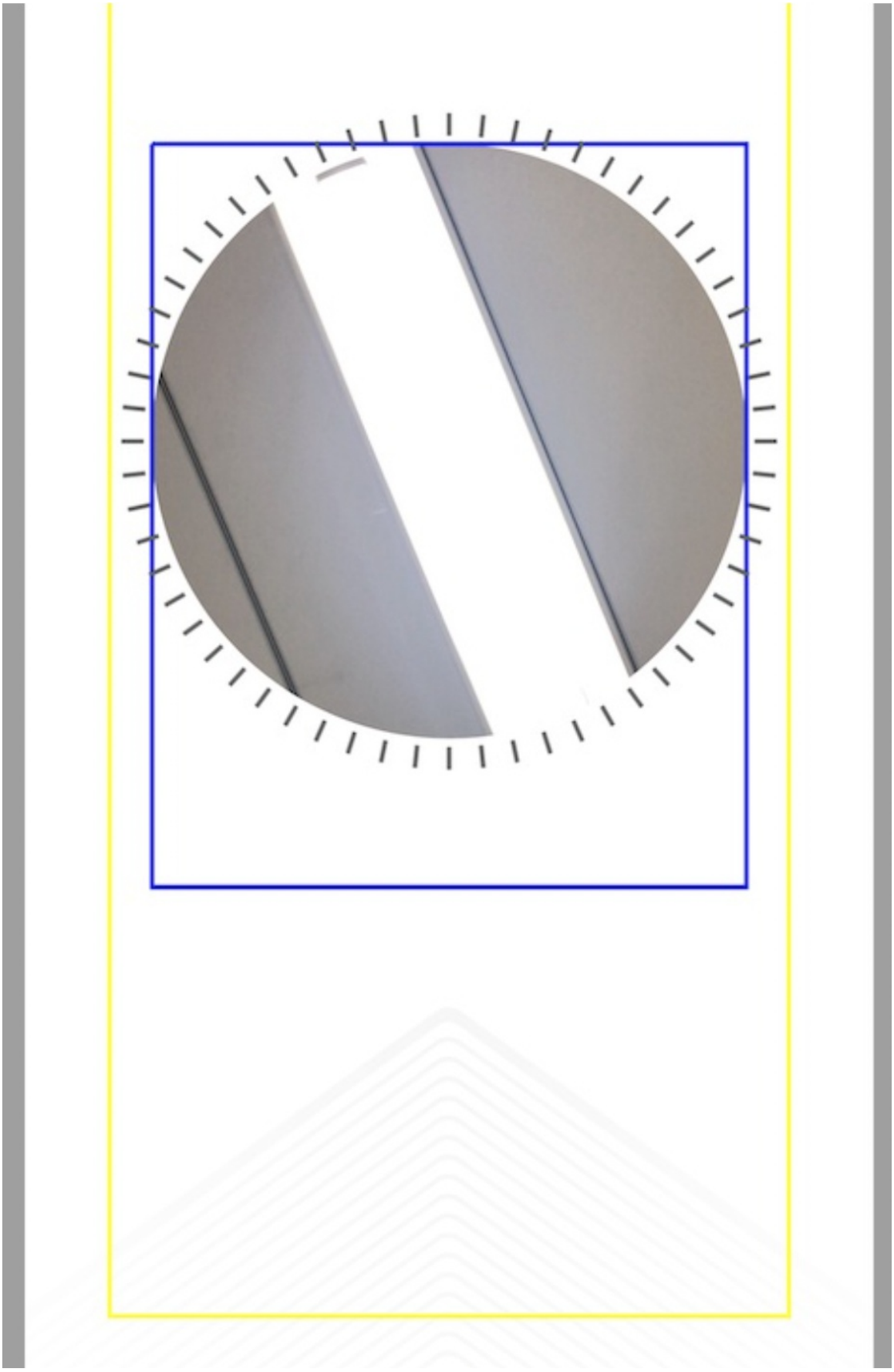
（2）PreviewLayer展示（建议方式）

使用[SSFaceDetectionManager sharedInstance];创建完detectionManager后，通过设置- (void)connectPreviewLayer:(AVCaptureVideoPreviewLayer*)pLayer;来进行PreviewLayer关联，后续镜头采集到的图像自动穿到到主界面的UI上展示。

3.3.4 Demo代码片段

参考BDFaceBaseViewController.m和BDFaceLivingConfigViewController.m实现，使用方法如下：

```
 // 初始化Layer
self.videoPreview = [[UIView alloc]initWithFrame:self.view.frame];
self.vPreviewLayer = [[AVCaptureVideoPreviewLayer alloc]init];
self.vPreviewLayer.frame = self.view.frame;
self.vPreviewLayer.videoGravity = AVLayerVideoGravityResizeAspect;

[self.videoPreview.layer addSublayer:self.vPreviewLayer];
[self.view addSubview:self.videoPreview];

// 初始化采集类
self.videoCapture = [SSFaceDetectionManager sharedInstance];
self.videoCapture.previewRect = self.previewRect;
self.videoCapture.detectRect = self.detectRect;
self.videoCapture.delegate = self;
self.videoCapture.riskDetectionSetting = 3;
self.videoCapture.enableSound = NO;
[self.videoCapture connectPreviewLayer:self.vPreviewLayer];

// 创建人脸识别参数
NSDictionary *parameters =  [self.videoCapture creatFaceVerifyParameters:@"身份证号" name:@"姓名"
verifyType:KFaceIdCardTypeDefault nation:nil phoneNumber:@"电话" livenessControl:nil spoofingControl:nil
qualityControl:nil];

// 执行人脸识别流程
[self.videoCapture startSessionWithType:BDFaceResultReportTypeVerifySec parameters:parameters
faceFlow:BDFaceDetectionTypeLiveness viewController:self];

// 流程回调处理
- (void)faceCallbackWithCode:(BDFaceCompletionStatus)status result:(NSDictionary *)result {
    if (status == BDFaceCompletionStatusSuccess) {
       // 流程成功
    } else if(status < 0) {
      // 流程异常
    }
}
```



说明：

- 检测图片中的人脸信息，完成人脸图像采集和人脸活体检测，返回检测状态和结果，通过SSCaptureDataOutputProtocol进行回调。

```
 // 帧图像回传，用于刷新UI使用
- (void)captureOutputSampleBuffer:(UIImage *)image;

// 本次人脸流程回调
- (void)faceSessionCompletionWithStatus:(BDFaceCompletionStatus)status result:(NSDictionary *)result;

// 活体检测状态
- (void)livenessActionDidFinishWithCode:(LivenessRemindCode)code;

// 人脸识别检测
- (void)detectionActionDidFinishWithCode:(DetectRemindCode)code;
```

4、常见问题

Q：鉴权问题。提示「验证失败」 A：先确定网络情况是否正常，本地鉴权文件失效了才走网络鉴权。定位错误码，排查鉴权失败的原因。一般是 licenseID 和 bundleID 配置不一致导致的鉴权失败。请注意上线前授权文件一定要更新。

Q：license 文件失效了，不能用了怎么办？ A：License 文件申请时候有期限，如过期会导致校验失效，需要在后台进行申请延期。

Q：使用 iOS 采集端，采集到的图片是斜着的，这个正常吗，会影响识别吗？ A：不会影响识别。有黑边和倾斜是因为图片质量算法造成的，我们是按 1:3 对图像进行背景填充使人脸居中，为的是更好的识别图像。

更多问题请点击 [常见问题](

增强级SDK

Android

采集SDK4.1版本兼容增强级及标准级的SDK版本，故版本号都是公用的4.1版本，增强级方案中需要配合增强级接口使用才能支持数据加密及大数据风控功能。

1、简介

百度Face SDK Android 版是一种面向 Android 移动设备人脸技术开发包，此版SDK包含人脸检测、活体识别等功能，以aar包+动态链接库的形式发布。基于该方案，开发者可以轻松的构建包含人脸检测、采集和活体识别的应用。在您使用SDK之前，我们首先为您介绍以下SDK本身及相关人脸能力，以便您能更方便使用人脸服务。

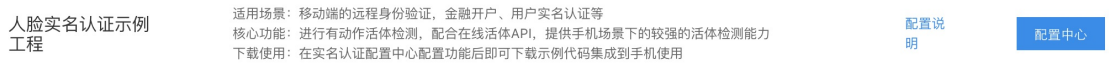
1.1 功能介绍

此版SDK所包含的能力如下：

- **离线动作活体检测**：通过让用户做出指定人脸配合式的交互动作，识别当前操作者是否为活体，此功能为离线使用，可设定指定动作是否使用及应用顺序。动作包含：眨眨眼、张闭嘴、向左摇头、向右摇头、向上抬头，向下低头6个。可有效抵御高清图片、3D建模、视频等攻击。
- **离线人脸质量检测**：判断视频流中的图片帧中，哪些图片质量较佳，即人脸图像特征清晰（满足姿态角、光照、模糊度、遮挡等校验）。
- **离线人脸图像采集**：通过本地SDK能力，采集人脸图像，同时经过人脸质量检测，确保采集到的人脸图像符合各条件校验（满足姿态角、光照、模糊度、遮挡等校验），为设备前端获取有效可分析人脸的主要功能。
- **离线授权**：SDK的授权判断，授权介质也称为license，在SDK使用中，优先验证本地离线鉴权文件，验证通过可离线使用；在本地鉴权失败情况下需要向授权服务器发起请求，远程拉取授权进行验证。

> 此版SDK**全部功能为离线版本，所有功能均本地化使用**，主要用于在客户端（Android）获取人脸，实际业务使用中，可以按照业务需要，配合在线API完成全流程的业务集成。

为了方便您的开发，我们已经为您准备了【人脸实名认证】场景的示例工程，您可以根据业务需要，在后台进行直接下载, 示例工程参考下图：



1.2 兼容性

系统：支持 Android 5.1(API Level 22)及以上系统。需要开发者通过 minSdkVersion来保证支持系统的检测。

机型：手机和平板皆可（暂不支持横屏）

构架：支持 CPU架构平台【armeabi-v7a、arm64-v8a】

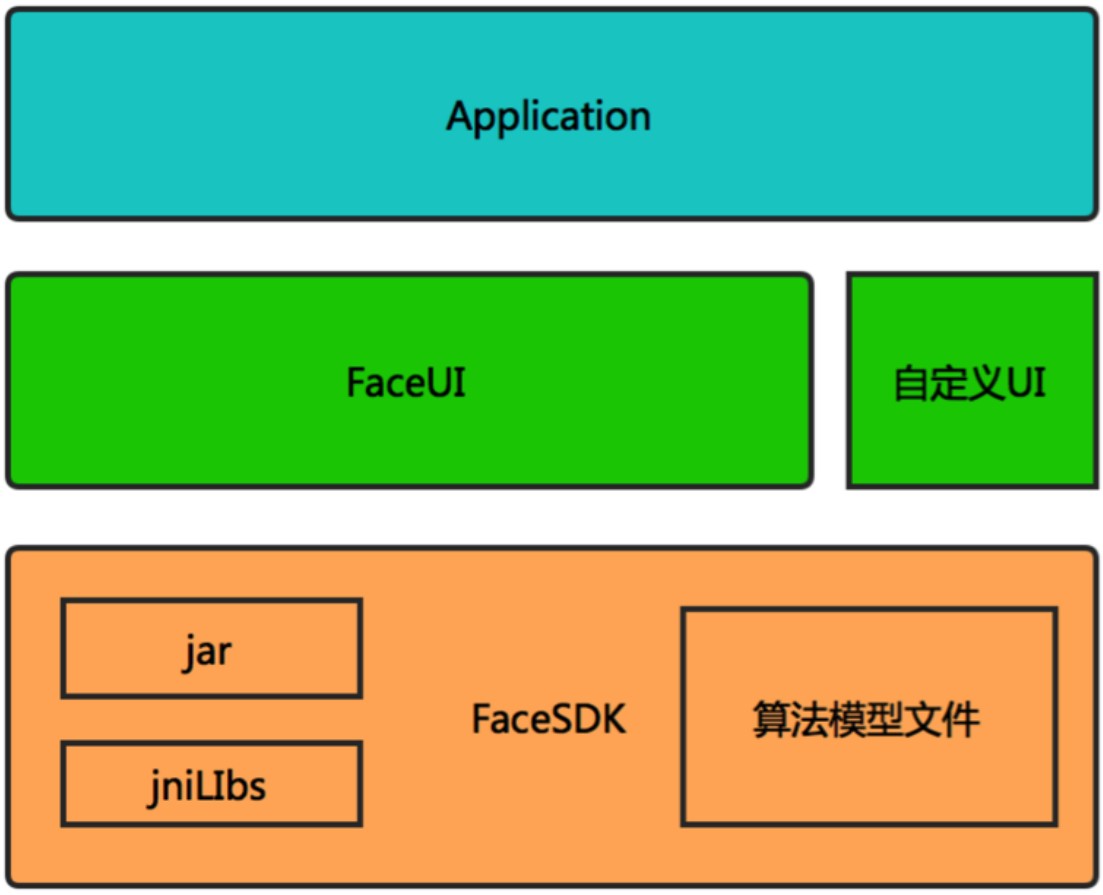
网络：支持 WIFI 及移动网络。

1.3 开发包说明

文件/文件夹名	说明
/faceplatform-release	SDK lib库、模型文件以及采集逻辑代码打包的aar
/faceplatform-ui	SDK的UI库，封装采集和动作活体UI等功能，以及各平台的so库。so包含以下几个平台如果关注包大小，请自行删减。【armeabi-v7a、arm64-v8a】
/app	DEMO工程（包含首页面、采集成功失败页面、设置页面等）

2、集成指南

本章将进行 Step-By-Step的讲解,如何快速的集成 人脸Sdk到现有应用中。一个完整的Demo 请参考开发包中的示例程序 FacePlatform。方案架构参考下图：

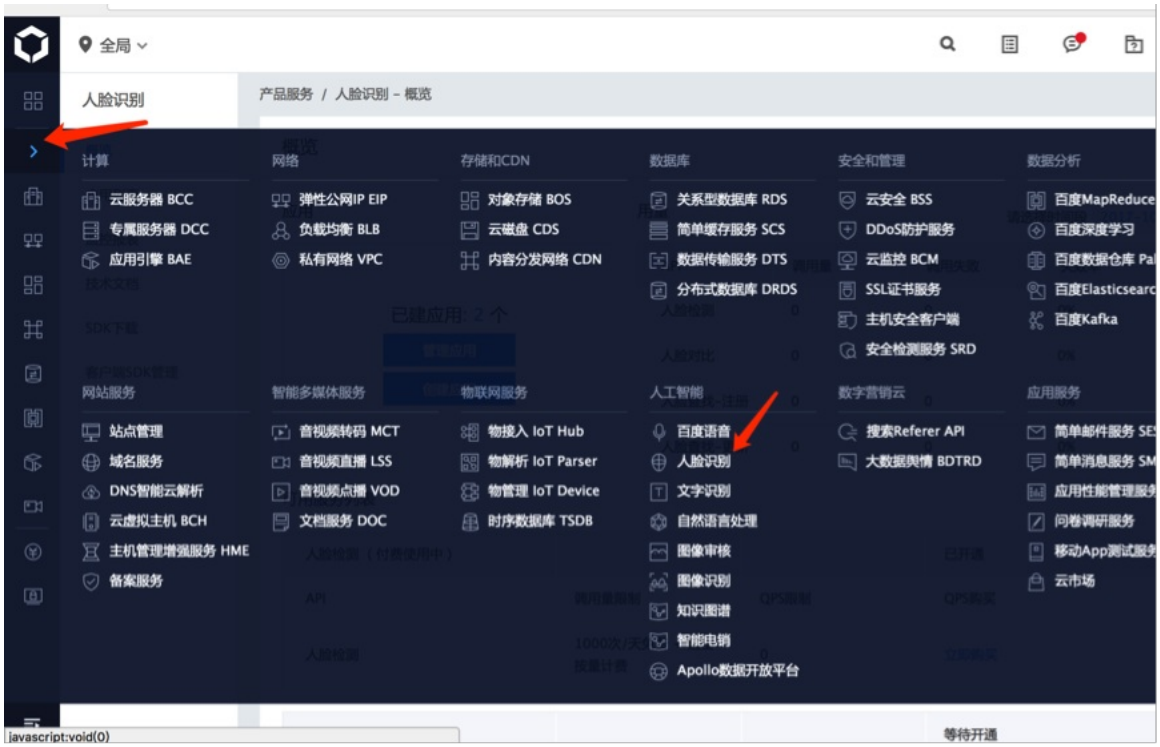


2.1 准备工作

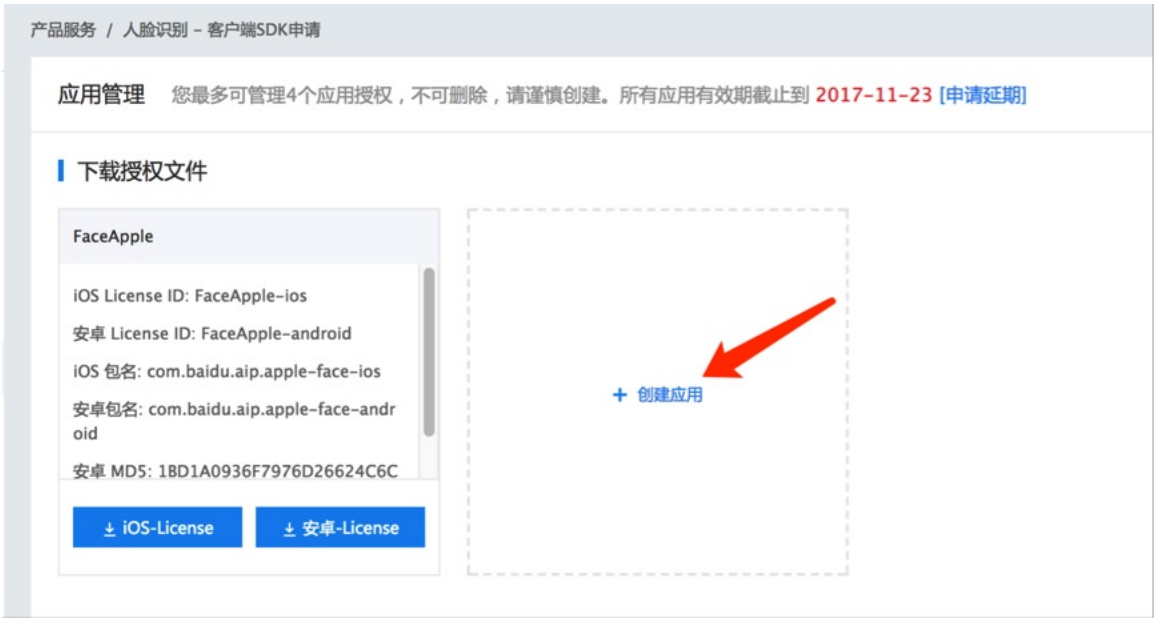
2.1.1 申请license

人脸SDK License：此license用于SDK离线功能使用，在您的申请人脸SDK的后台页面，全局->产品服务->人脸识别->客户端SDK申请

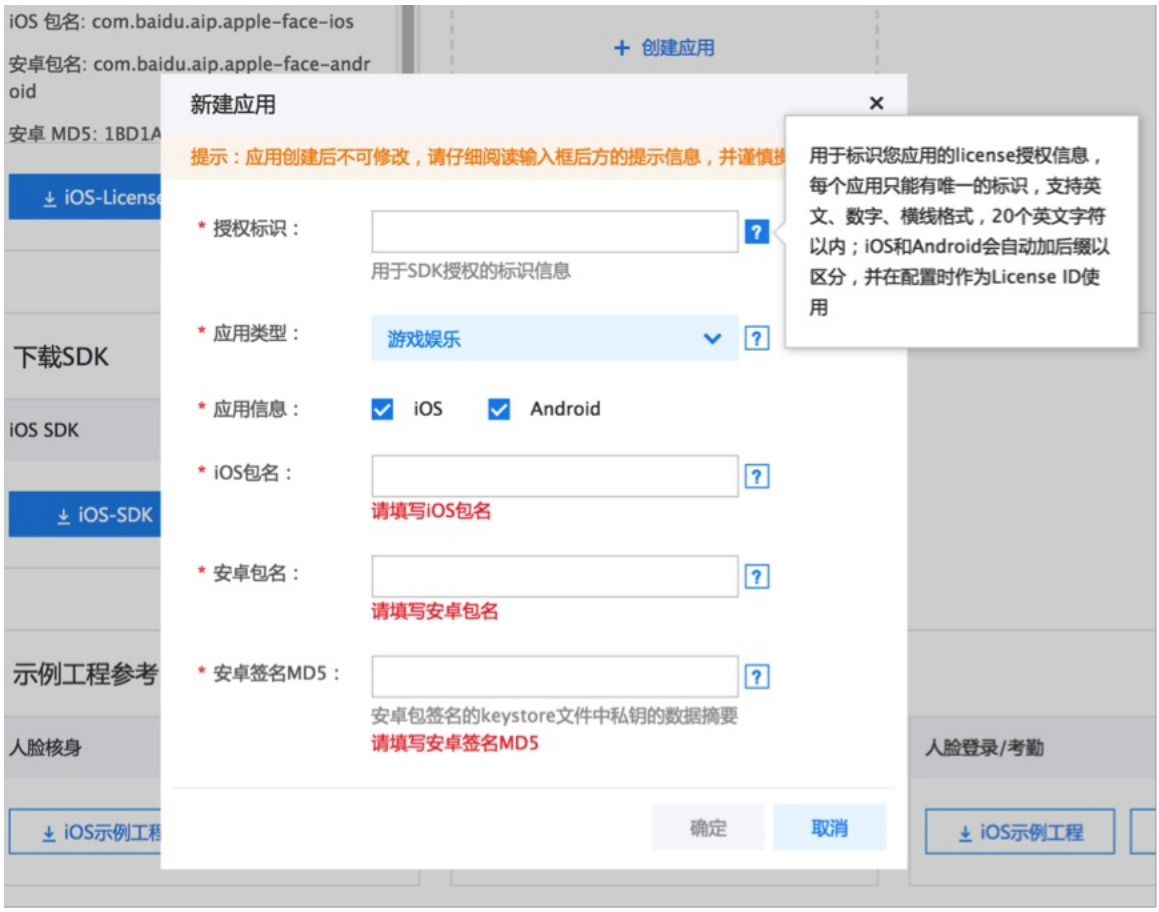
人脸控制台路径如下：



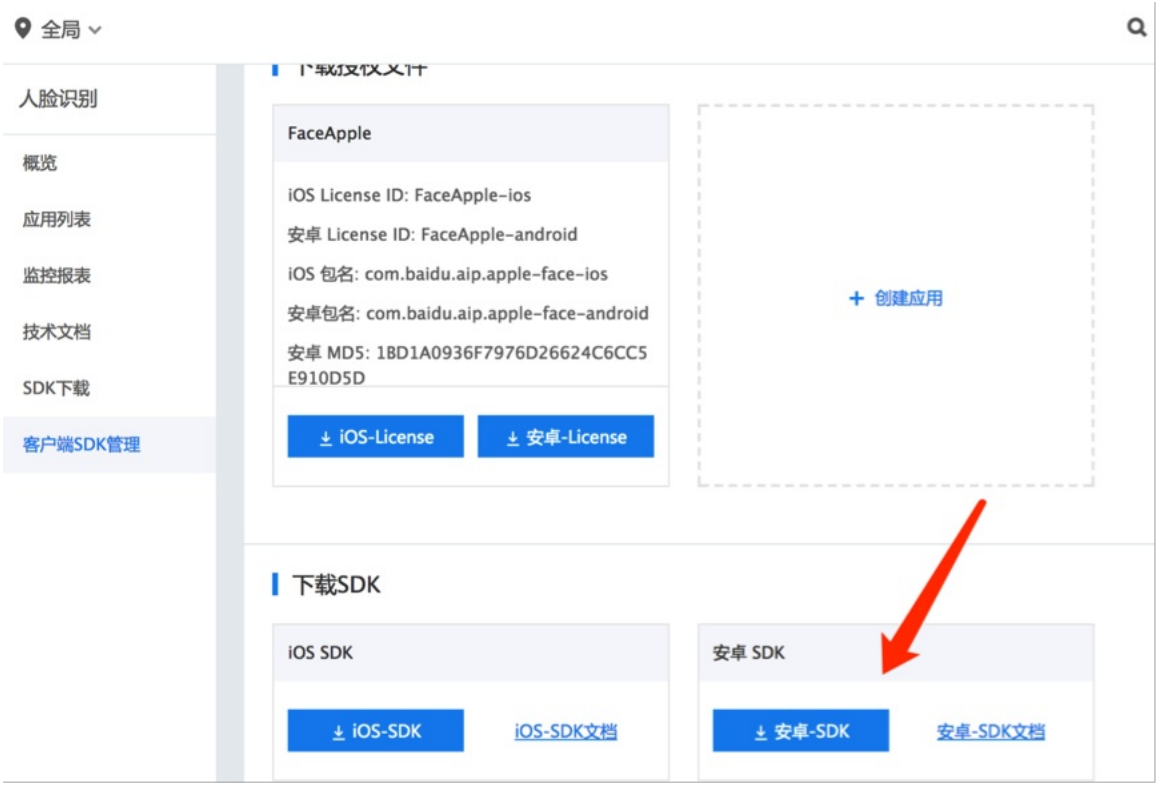
点击客户端SDK管理，弹出如下图：创建应用（这里创建应用是为了使用离线SDK，上面创建应用为了使用人脸在线接口，如注册、识别等）



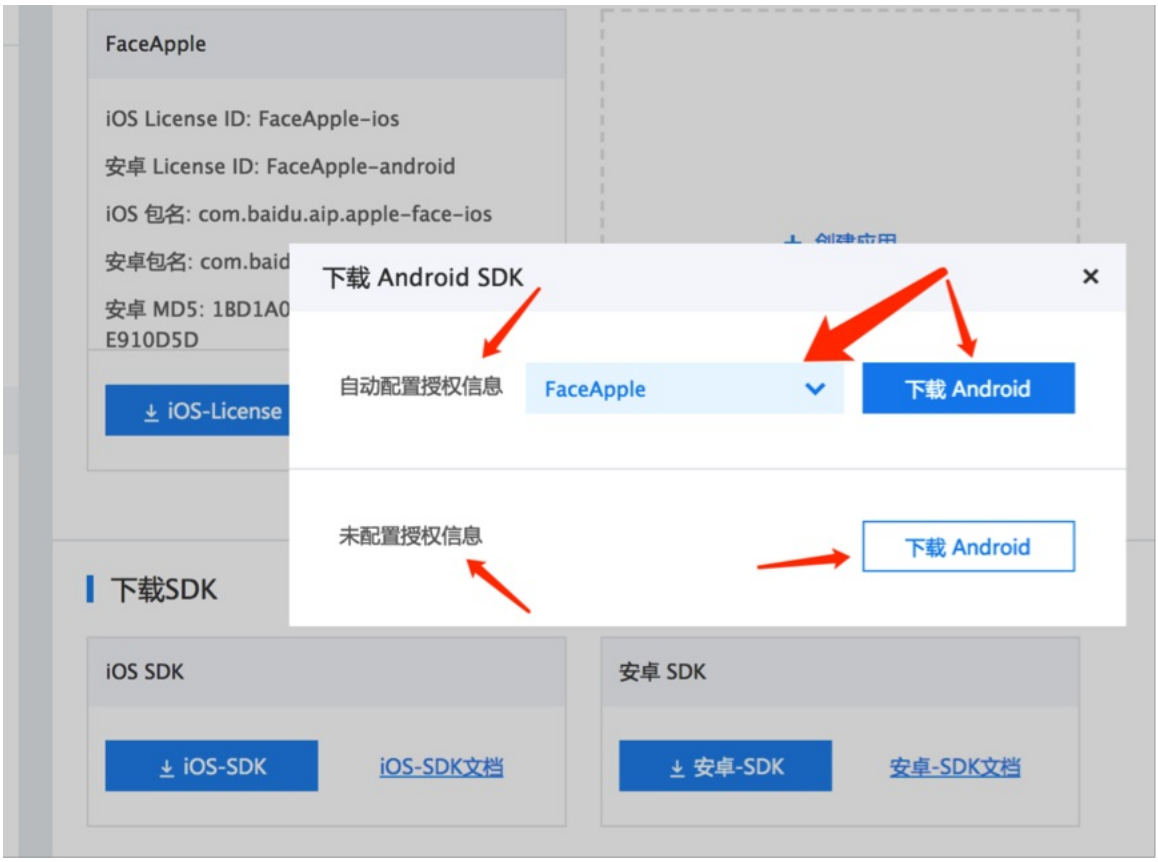
在弹出的框中输入授权标识，选择应用类型，应用系统，以及包名、MD5签名，详情请查看输入框右边提示



2.1.2 下载SDK



下载SDK分为自动配置授权信息（创建license后就可以选择为该应用，下载后SDK自动帮您配置授权，不用下载license拷贝到工程中，初始化参数licenseID,包名也帮您配置好了）和未配置授权信息两种方式：



2.2 运行示例工程

2.3.1 运行自动配置授权信息的示例工程

该下载的示例工程，已经帮你改好了license和包名

- Android Studio导入下载的示例工程
- 配置打包签名文件，由于SDK运行时会校验签名文件的MD5是否和申请时填入一样。为了便于debug能正常使用SDK，需要

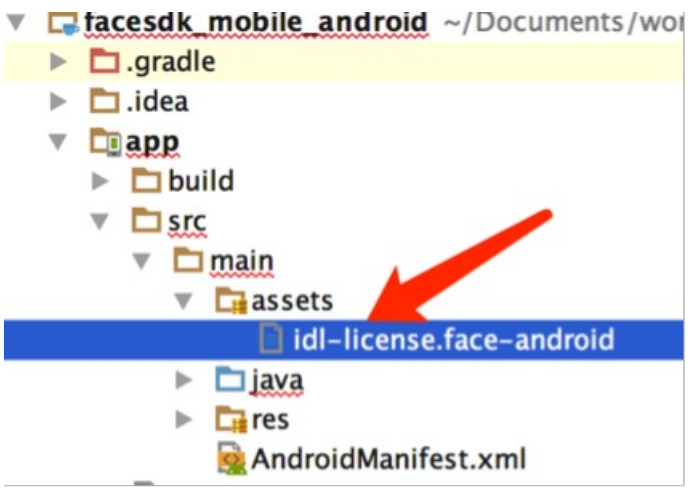
配置打包签名文件。

- 运行示例工程

2.3.2 运行未配置授权信息的示例工程

（1）Android Studio导入下载的示例工程

（2）下载license拷贝到工程的assets目录

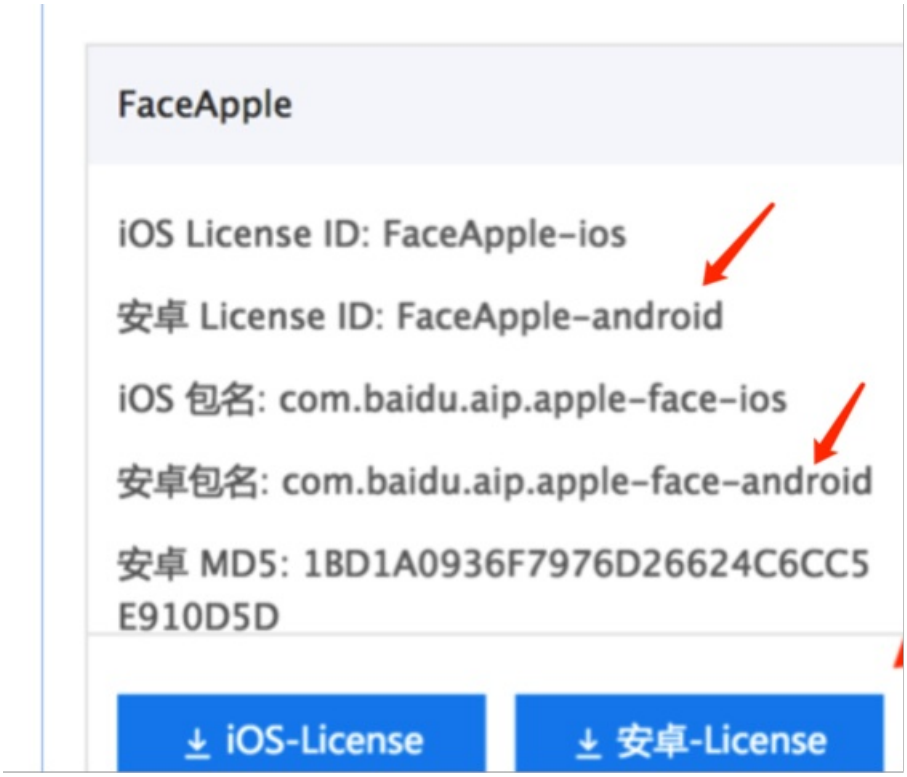


（3）修改HomeActivity.java的initialize方法参数

```
// 为了android和ios 区分授权，appId=appname_face_android ,其中appname为申请sdk时的应用名
// 应用上下文
// 申请License取得的APPID
// assets目录下License文件名
FaceSDKManager.getInstance().initialize(mContext,  licenseID: "[redacted]",
        licenseFileName: "[redacted]", new IInitCallback() {
            @Override
            public void initSuccess() {
                runOnUiThread(() -> {
                        Log.e(TAG,  msg: "初始化成功");
                        showToast( msg: "初始化成功");
                        mIsInitSuccess = true;
                });
            }

            @Override
            public void initFailure(final int errCode, final String errMsg) {
                runOnUiThread(() -> {
                        Log.e(TAG,  msg: "初始化失败 = " + errCode + " " + errMsg);
                        showToast( msg: "初始化失败 = " + errCode + ", " + errMsg);
                        mIsInitSuccess = false;
                });
            }
        });
```

查看申请license信息，里面包含licenseId

填入放到assets目录下的授权文件名称

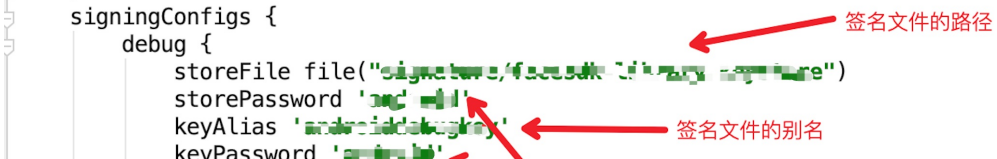


（4）修改app.gradle里面的包名为申请license填入的包名，如上图安卓包名。

```
android {
    compileSdkVersion 25
    buildToolsVersion "25.0.3"
    defaultConfig {
        applicationId "[redacted].turnstile"
        minSdkVersion 15
        targetSdkVersion 25
        versionCode 1
        versionName "1.0"

    }
```

（5）配置打包签名文件，由于SDK运行时会校验签名文件的MD5是否和申请时填入一样。为了便于debug能正常使用SDK，需要配置打包签名文件。

```
    }
    signingConfigs {
        debug {
            storeFile file("[illegible]")
            storePassword '[illegible]'
            keyAlias '[illegible]'
            keyPassword '[illegible]'
        }
        release {
            storeFile file("[illegible]")
            storePassword '[illegible]'
            keyAlias '[illegible]'
            keyPassword '[illegible]'
        }
    }
    buildTypes {
```



（6）运行示例工程。如果无法正常体验，请查看logcat日志。是否有 **FaceSDK-License LICENSE_INFO_CHECK_ERROR**日志。如果有说明授权没有成功，可以查看本文档最后的常见问题进行解决。

2.4 添加SDK到工程

FaceSdk以androidstudio开发方式提供，以下介绍在android studio开发工具导入FaceSdk

（1）将开发包中的faceplatform-release库Copy 到工程根目录。

（2）将开发包中的faceplatform-ui库Copy 到工程根目录。

（3）SDK提供的了开源的faceplatform-ui库，把活体检测和人脸图像采集功能等功能进行了封装，适配了主流机型机型。如果需要使用，请添加faceplatform-ui模块到的工程中。faceplatform-ui目录结构如下图

```
    ![img](http://bjhw-bce-online-public0.bjhw.baidu.com:8109/proxy-external/?url=https://ai.bdstatic.com/file/A22B3618EB134243ADD6A54691D28791)
```

（4）在build.gradle使用compile project引人faceplatform-ui库工程。

```
dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    compile "com.android.support:recyclerview-v7:+"
    compile project(path: ':faceplatform-ui')
}
```

（5）Setting.gradle中include faceplatfrom-ui和facepaltfrom-release

```
include ':app', ':faceplatform-release'
include ':faceplatform-ui'
```

（6）从官网下载授权文件license，复制到app/src/main/assets目录下。

（7）申请的license已经和打包签名key进行了绑定（申请时用到了签名的md5，为了便于debug模式也能调用SDK的功能，需要把debug的key改成申请license的key。

- 把key拷贝到项目根目录下
- 主appbuild.gradle android 下面添加(修改)signingConfigs相关的配置。如下图。

```
    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'
        }
    }

    signingConfigs {
        debug {
            keyAlias 'facesharp'
            keyPassword [redacted]
            storeFile file('../facesharp.jks')
            storePassword([redacted])
        }
        release {
            keyAlias [redacted]
            keyPassword [redacted]
            storeFile file('../facesharp.jks')
            storePassword([redacted])
        }
    }
```

2.5 权限声明

名称	用途
需要动态申请的权限	
android.permission.READ_EXTERNAL_STORAGE	读取手机外部存储权限
android.permission.WRITE_EXTERNAL_STORAGE	写入手机外部存储权限
android.permission.CAMERA	拍照权限
不需要动态申请的权限	
android.permission.READ_PHONE_STATE	获取用户手机的 IMEI,用来唯一的标识用户
android.hardware.camera.autofocus	允许相机对焦
android.permission.INTERNET	允许访问网络
android.permission.WRITE_SETTINGS	允许修改系统设置
android.permission.WAKE_LOCK	屏幕常亮权限

2.6、混淆设置

如果工程需要做混淆处理的话，则在工程的proguard-rules.pro文件中添加如下配置：

```
-keep class com.baidu.vis.unified.license.** {*;}

-keep class com.baidu.liantian.** {*;}

-keep class com.baidu.baidusec.** {*;}

-keep class com.baidu.idl.main.facesdk.** {*;}
```

3、功能使用

3.1 人脸采集（包含动作活体）

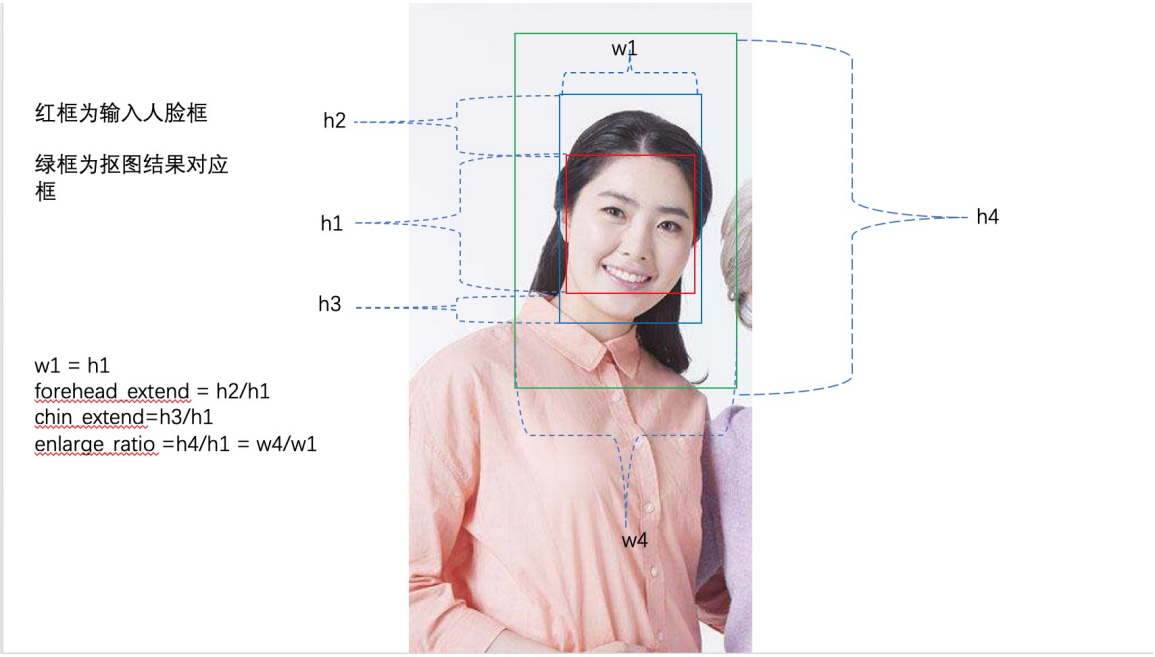
（1）初始化SDK **参数分别表示：当前上下文、鉴权key、鉴权名称、回调参数** 调用 FaceSDKManager.getIntance().initialize(Context context, String licenseID, String licenseFileName, IInitCallback callback); **Demo中此段代码在HomeActivity中。**

（2） 初始化参数设置

```
 FaceConfig config = FaceSDKManager.getInstance().getFaceConfig();
// SDK初始化已经设置完默认参数（推荐参数），也可以根据实际需求进行数值调整
// 质量等级（0：正常、1：宽松、2：严格、3：自定义）
// 获取保存的质量等级
SharedPreferencesUtil util = new SharedPreferencesUtil(mContext);
int qualityLevel = (int) util.getSharedPreference(Const.KEY_QUALITY_LEVEL_SAVE, -1);
if (qualityLevel == -1) {
    qualityLevel = ExampleApplication.qualityLevel;
}
// 根据质量等级获取相应的质量值（注：第二个参数要与质量等级的set方法参数一致）
QualityConfigManager manager = QualityConfigManager.getInstance();
manager.readQualityFile(mContext.getApplicationContext(), qualityLevel);
QualityConfig qualityConfig = manager.getConfig();
if (qualityConfig == null) {
    return false;
}
// 设置模糊度阈值
config.setBlurnessValue(qualityConfig.getBlur());
// 设置最小光照阈值（范围0-255）
config.setBrightnessValue(qualityConfig.getMinIllum());
// 设置最大光照阈值（范围0-255）
config.setBrightnessMaxValue(qualityConfig.getMaxIllum());
// 设置左眼遮挡阈值
config.setOcclusionLeftEyeValue(qualityConfig.getLeftEyeOcclusion());
// 设置右眼遮挡阈值
config.setOcclusionRightEyeValue(qualityConfig.getRightEyeOcclusion());
// 设置鼻子遮挡阈值
config.setOcclusionNoseValue(qualityConfig.getNoseOcclusion());
// 设置嘴巴遮挡阈值
config.setOcclusionMouthValue(qualityConfig.getMouseOcclusion());
// 设置左脸颊遮挡阈值
config.setOcclusionLeftContourValue(qualityConfig.getLeftContourOcclusion());
// 设置右脸颊遮挡阈值
config.setOcclusionRightContourValue(qualityConfig.getRightContourOcclusion());
// 设置下巴遮挡阈值
config.setOcclusionChinValue(qualityConfig.getChinOcclusion());
// 设置人脸姿态角阈值
config.setHeadPitchValue(qualityConfig.getPitch());
config.setHeadYawValue(qualityConfig.getYaw());
config.setHeadRollValue(qualityConfig.getRoll());
// 设置可检测的最小人脸阈值
config.setMinFaceSize(FaceEnvironment.VALUE_MIN_FACE_SIZE);
// 设置可检测到人脸的阈值
config.setNotFaceValue(FaceEnvironment.VALUE_NOT_FACE_THRESHOLD);
```

```
// 设置闭眼阈值
config.setEyeClosedValue(FaceEnvironment.VALUE_CLOSE_EYES);
// 设置图片缓存数量
config.setCacheImageNum(FaceEnvironment.VALUE_CACHE_IMAGE_NUM);
// 设置活体动作，通过设置list，LivenessTypeEunm.Eye, LivenessTypeEunm.Mouth,
// LivenessTypeEunm.HeadUp, LivenessTypeEunm.HeadDown, LivenessTypeEunm.HeadLeft,
// LivenessTypeEunm.HeadRight
config.setLivenessTypeList(ExampleApplication.livenessList);
// 设置动作活体是否随机
config.setLivenessRandom(ExampleApplication.isLivenessRandom);
// 设置开启提示音
config.setSound(ExampleApplication.isOpenSound);
// 原图缩放系数
config.setScale(FaceEnvironment.VALUE_SCALE);
// 抠图宽高的设定，为了保证好的抠图效果，建议高宽比是4：3
config.setCropHeight(FaceEnvironment.VALUE_CROP_HEIGHT);
config.setCropWidth(FaceEnvironment.VALUE_CROP_WIDTH);
// 抠图人脸框与背景比例
config.setEnlargeRatio(FaceEnvironment.VALUE_CROP_ENLARGERATIO);
// 加密类型，0：非加密原图Base64，上传时image_sec传false；1：百度加密文件加密，上传时image_sec传true
config.setSecType(FaceEnvironment.VALUE_SEC_TYPE);
// 检测超时设置
config.setTimeDetectModule(FaceEnvironment.TIME_DETECT_MODULE);
// 检测框远近比率
config.setFaceFarRatio(FaceEnvironment.VALUE_FAR_RATIO);
config.setFaceClosedRatio(FaceEnvironment.VALUE_CLOSED_RATIO);
FaceSDKManager.getInstance().setFaceConfig(config);
```

关于抠图内部实现方案如下图所示：



（3） startActivity(new Intent(this, FaceLivenessExpActivity.class))，开启预览。

（4）调用FaceSDKManager.getInstance().getLivenessStrategyModule()获得ILivenessStrategy对象。(该方法每次调用都会返回一个新对象).

（6）调用ILivenessStrategy.setPreviewDegree();设置预览图片的旋转角度。调用setDetectStrategySoundEnable设置是否开启语音。调用setDetectStrategyConfig设置，预览图的大小，人脸检测框的坐标和回调。

（7）多次调用livenessStrategy进行人脸图片采集，人脸跟踪、质量检测、动作活体检测。

（8）实现ILivenessStrategyCallback的onLivenessCompletion并处理结果。**其中base64ImageCropMap为存放人脸抠图图片集，base64ImageSrcMap为存放人脸原图图片集**并通过调用getBestImage()方法，通过从优到差的质量排序，获取最优的

bitmap，代码如下图所示：

```
main 〉 java 〉 com 〉 baidu 〉 idl 〉 face 〉 example 〉 FaceLivenessExpActivity
ategyExtModule.java × | SoundPoolHelper.java × | SoundPlayer.java × | FaceLivenessExpActivity.java × | FileUtils.java ×

@Override
public void onLivenessCompletion(FaceStatusNewEnum status, String message,
                                 HashMap<String, ImageInfo> base64ImageCropMap,
                                 HashMap<String, ImageInfo> base64ImageSrcMap, int currentLivenessCount) {
    super.onLivenessCompletion(status, message, base64ImageCropMap, base64ImageSrcMap, currentLivenessCount);
    if (status == FaceStatusNewEnum.OK && mIsCompletion) {
        // 获取最优图片
        getBestImage(base64ImageCropMap, base64ImageSrcMap);
    } else if (status == FaceStatusNewEnum.DetectRemindCodeTimeout) {
        if (mViewBg != null) {
            mViewBg.setVisibility(View.VISIBLE);
        }
        showMessageDialog();
    }
}


    /**
     * 获取最优图片
     * @param imageCropMap 抠图集合
     * @param imageSrcMap  原图集合
     */
    private void getBestImage(HashMap<String, ImageInfo> imageCropMap, HashMap<String, ImageInfo> imageSrcMap) {
        String bmpStr = null;
        // 将抠图集合中的图片按照质量降序排序，最终选取质量最优的一张抠图图片
        if (imageCropMap != null && imageCropMap.size() > 0) {
            List<Map.Entry<String, ImageInfo>> list1 = new ArrayList<>(imageCropMap.entrySet());
            Collections.sort(list1, new Comparator<Map.Entry<String, ImageInfo>>() {

                @Override
                public int compare(Map.Entry<String, ImageInfo> o1,
                                   Map.Entry<String, ImageInfo> o2) {
                    String[] key1 = o1.getKey().split("_");
                    String score1 = key1[2];
                    String[] key2 = o2.getKey().split("_");
                    String score2 = key2[2];
                    // 降序排序
                    return Float.valueOf(score2).compareTo(Float.valueOf(score1));
                }
            });

            // 获取抠图中的加密或非加密的base64
//            int secType = mFaceConfig.getSecType();
//            String base64;
//            if (secType == 0) {
//                base64 = list1.get(0).getValue().getBase64();
//            } else {
//                base64 = list1.get(0).getValue().getSecBase64();
//            }
        }


        // 将原图集合中的图片按照质量降序排序，最终选取质量最优的一张原图图片
        if (imageSrcMap != null && imageSrcMap.size() > 0) {
            List<Map.Entry<String, ImageInfo>> list2 = new ArrayList<>(imageSrcMap.entrySet());
            Collections.sort(list2, new Comparator<Map.Entry<String, ImageInfo>>() {

                @Override
                public int compare(Map.Entry<String, ImageInfo> o1,
                                   Map.Entry<String, ImageInfo> o2) {
                    String[] key1 = o1.getKey().split("_");
                    String score1 = key1[2];
                    String[] key2 = o2.getKey().split("_");
                    String score2 = key2[2];
                    // 降序排序
                    return Float.valueOf(score2).compareTo(Float.valueOf(score1));
                }
            });
            bmpStr = list2.get(0).getValue().getBase64();

            // 获取原图中的加密或非加密的base64
//            int secType = mFaceConfig.getSecType();
//            String base64;
//            if (secType == 0) {
//                base64 = mBmpStr;
//            } else {
//                base64 = list2.get(0).getValue().getBase64();
//            }
        }

        // 页面跳转
        IntentUtils.getInstance().setBitmap(bmpStr);
        Intent intent = new Intent(FaceLivenessExpActivity.this,
                CollectionSuccessActivity.class);
        intent.putExtra("destroyType", "FaceLivenessExpActivity");
        startActivity(intent);
    }
```

（9）Demo中的最优抠图或者最优原图，可以调用SecRequest类下的sendMessage(Context context, String secBase64, int

secType);将加密后的base64发送到服务端，如需风控，需要把SecRequest.java下的risk_identify参数置为true。

3.2 人脸采集（不包含动作活体）

（1）初始化SDK **参数分别表示：当前上下文、鉴权key、鉴权名称、回调参数** 调用 FaceSDKManager.getIntance().initialize(Context context, String licenseID, String licenseFileName, IInitCallback callback); **Demo中此段代码在HomeActivity中。**

（2） 初始化参数设置

```
 FaceConfig config = FaceSDKManager.getInstance().getFaceConfig();
// SDK初始化已经设置完默认参数（推荐参数），也可以根据实际需求进行数值调整
// 质量等级（0：正常、1：宽松、2：严格、3：自定义）
// 获取保存的质量等级
SharedPreferencesUtil util = new SharedPreferencesUtil(mContext);
int qualityLevel = (int) util.getSharedPreference(Const.KEY_QUALITY_LEVEL_SAVE, -1);
if (qualityLevel == -1) {
    qualityLevel = ExampleApplication.qualityLevel;
}
// 根据质量等级获取相应的质量值（注：第二个参数要与质量等级的set方法参数一致）
QualityConfigManager manager = QualityConfigManager.getInstance();
manager.readQualityFile(mContext.getApplicationContext(), qualityLevel);
QualityConfig qualityConfig = manager.getConfig();
if (qualityConfig == null) {
    return false;
}
// 设置模糊度阈值
config.setBlurnessValue(qualityConfig.getBlur());
// 设置最小光照阈值（范围0-255）
config.setBrightnessValue(qualityConfig.getMinIllum());
// 设置最大光照阈值（范围0-255）
config.setBrightnessMaxValue(qualityConfig.getMaxIllum());
// 设置左眼遮挡阈值
config.setOcclusionLeftEyeValue(qualityConfig.getLeftEyeOcclusion());
// 设置右眼遮挡阈值
config.setOcclusionRightEyeValue(qualityConfig.getRightEyeOcclusion());
// 设置鼻子遮挡阈值
config.setOcclusionNoseValue(qualityConfig.getNoseOcclusion());
// 设置嘴巴遮挡阈值
config.setOcclusionMouthValue(qualityConfig.getMouseOcclusion());
// 设置左脸颊遮挡阈值
config.setOcclusionLeftContourValue(qualityConfig.getLeftContourOcclusion());
// 设置右脸颊遮挡阈值
config.setOcclusionRightContourValue(qualityConfig.getRightContourOcclusion());
// 设置下巴遮挡阈值
config.setOcclusionChinValue(qualityConfig.getChinOcclusion());
// 设置人脸姿态角阈值
config.setHeadPitchValue(qualityConfig.getPitch());
config.setHeadYawValue(qualityConfig.getYaw());
config.setHeadRollValue(qualityConfig.getRoll());
// 设置可检测的最小人脸阈值
config.setMinFaceSize(FaceEnvironment.VALUE_MIN_FACE_SIZE);
// 设置可检测到人脸的阈值
config.setNotFaceValue(FaceEnvironment.VALUE_NOT_FACE_THRESHOLD);
// 设置闭眼阈值
config.setEyeClosedValue(FaceEnvironment.VALUE_CLOSE_EYES);
// 设置图片缓存数量
config.setCacheImageNum(FaceEnvironment.VALUE_CACHE_IMAGE_NUM);
// 设置活体动作，通过设置list，LivenessTypeEunm.Eye, LivenessTypeEunm.Mouth,
// LivenessTypeEunm.HeadUp, LivenessTypeEunm.HeadDown, LivenessTypeEunm.HeadLeft,
// LivenessTypeEunm.HeadRight
config.setLivenessTypeList(ExampleApplication.livenessList);
```

```
// 设置动作活体是否随机
config.setLivenessRandom(ExampleApplication.isLivenessRandom);
// 设置开启提示音
config.setSound(ExampleApplication.isOpenSound);
// 原图缩放系数
config.setScale(FaceEnvironment.VALUE_SCALE);
// 抠图宽高的设定，为了保证好的抠图效果，建议高宽比是4：3
config.setCropHeight(FaceEnvironment.VALUE_CROP_HEIGHT);
config.setCropWidth(FaceEnvironment.VALUE_CROP_WIDTH);
// 抠图人脸框与背景比例
config.setEnlargeRatio(FaceEnvironment.VALUE_CROP_ENLARGERATIO);
// 加密类型，0：非加密原图Base64，上传时image_sec传false；1：百度加密文件加密，上传时image_sec传true
config.setSecType(FaceEnvironment.VALUE_SEC_TYPE);
// 检测超时设置
config.setTimeDetectModule(FaceEnvironment.TIME_DETECT_MODULE);
// 检测框远近比率
config.setFaceFarRatio(FaceEnvironment.VALUE_FAR_RATIO);
config.setFaceClosedRatio(FaceEnvironment.VALUE_CLOSED_RATIO);
FaceSDKManager.getInstance().setFaceConfig(config);
```

（3）startActivity(new Intent(this, FaceDetectExpActivity.class))，开启预览。

（4）调用FaceSDKManager.getInstance().getDetectStrategyModule()获得IDetectStrategy对象。(该方法每次调用都会返回一个新对象).

（5）调用IDetectStrategy.setPreviewDegree();设置预览图片的旋转角度。调用setDetectStrategySoundEnable设置是否开启语音。调用setDetectStrategyConfig设置，预览图的大小，人脸检测框的坐标和回调。

（6） 多次调用detectStrategy进行人脸图片采集。（不带动作活体中支持口罩检测，当口罩分值大于0.7时便认为戴口罩，这时将不会进行遮挡检测）

（7） 实现IDetectStrategyCallback的onDetectCompletion并处理结果。**其中base64ImageCropMap为存放人脸抠图图片集，base64ImageSrcMap为存放人脸原图图片集**并通过调用getBestImage()方法，通过从优到差的质量排序，获取最优的bitmap，代码如下图所示：

ctStrategyExtModule.java | SoundPoolHelper.java | SoundPlayer.java | FaceDetectExpActivity.java

```
@Override
public void onDetectCompletion(FaceStatusNewEnum status, String message,
                               HashMap<String, ImageInfo> base64ImageCropMap,
                               HashMap<String, ImageInfo> base64ImageSrcMap) {
    super.onDetectCompletion(status, message, base64ImageCropMap, base64ImageSrcMap);
    if (status == FaceStatusNewEnum.OK && mIsCompletion) {
        // 获取最优图片
        getBestImage(base64ImageCropMap, base64ImageSrcMap);
    } else if (status == FaceStatusNewEnum.DetectRemindCodeTimeout) {
        if (mViewBg != null) {
            mViewBg.setVisibility(View.VISIBLE);
        }
        showMessageDialog();
    }
}
```

```
    /**
     * 获取最优图片
     * @param imageCropMap 抠图集合
     * @param imageSrcMap  原图集合
     */
    private void getBestImage(HashMap<String, ImageInfo> imageCropMap, HashMap<String, ImageInfo> imageSrcMap) {
        String bmpStr = null;
        // 将抠图集合中的图片按照质量降序排序，最终选取质量最优的一张抠图图片
        if (imageCropMap != null && imageCropMap.size() > 0) {
            List<Map.Entry<String, ImageInfo>> list1 = new ArrayList<>(imageCropMap.entrySet());
            Collections.sort(list1, new Comparator<Map.Entry<String, ImageInfo>>() {

                @Override
                public int compare(Map.Entry<String, ImageInfo> o1,
                                   Map.Entry<String, ImageInfo> o2) {
                    String[] key1 = o1.getKey().split("_");
                    String score1 = key1[2];
                    String[] key2 = o2.getKey().split("_");
                    String score2 = key2[2];
                    // 降序排序
                    return Float.valueOf(score2).compareTo(Float.valueOf(score1));
                }
            });

            // 获取抠图中的加密或非加密的base64
//            int secType = mFaceConfig.getSecType();
//            String base64;
//            if (secType == 0) {
//                base64 = list1.get(0).getValue().getBase64();
//            } else {
//                base64 = list1.get(0).getValue().getSecBase64();
//            }
        }


        // 将原图集合中的图片按照质量降序排序，最终选取质量最优的一张原图图片
        if (imageSrcMap != null && imageSrcMap.size() > 0) {
            List<Map.Entry<String, ImageInfo>> list2 = new ArrayList<>(imageSrcMap.entrySet());
            Collections.sort(list2, new Comparator<Map.Entry<String, ImageInfo>>() {

                @Override
                public int compare(Map.Entry<String, ImageInfo> o1,
                                   Map.Entry<String, ImageInfo> o2) {
                    String[] key1 = o1.getKey().split("_");
                    String score1 = key1[2];
                    String[] key2 = o2.getKey().split("_");
                    String score2 = key2[2];
                    // 降序排序
                    return Float.valueOf(score2).compareTo(Float.valueOf(score1));
                }
            });
            bmpStr = list2.get(0).getValue().getBase64();

            // 获取原图中的加密或非加密的base64
//            int secType = mFaceConfig.getSecType();
//            String base64;
//            if (secType == 0) {
//                base64 = mBmpStr;
//            } else {
//                base64 = list2.get(0).getValue().getBase64();
//            }
        }

        // 页面跳转
        IntentUtils.getInstance().setBitmap(bmpStr);
        Intent intent = new Intent(FaceLivenessExpActivity.this,
                CollectionSuccessActivity.class);
        intent.putExtra("destroyType", "FaceLivenessExpActivity");
        startActivity(intent);
    }
```

（8）Demo中的最优抠图或者最优原图，可以调用SecRequest类下的sendMessage(Context context, String secBase64, int secType);将加密后的base64发送到服务端，如需风控，需要把SecRequest.java下的risk_identify参数置为true。

3.3 质量校验设置

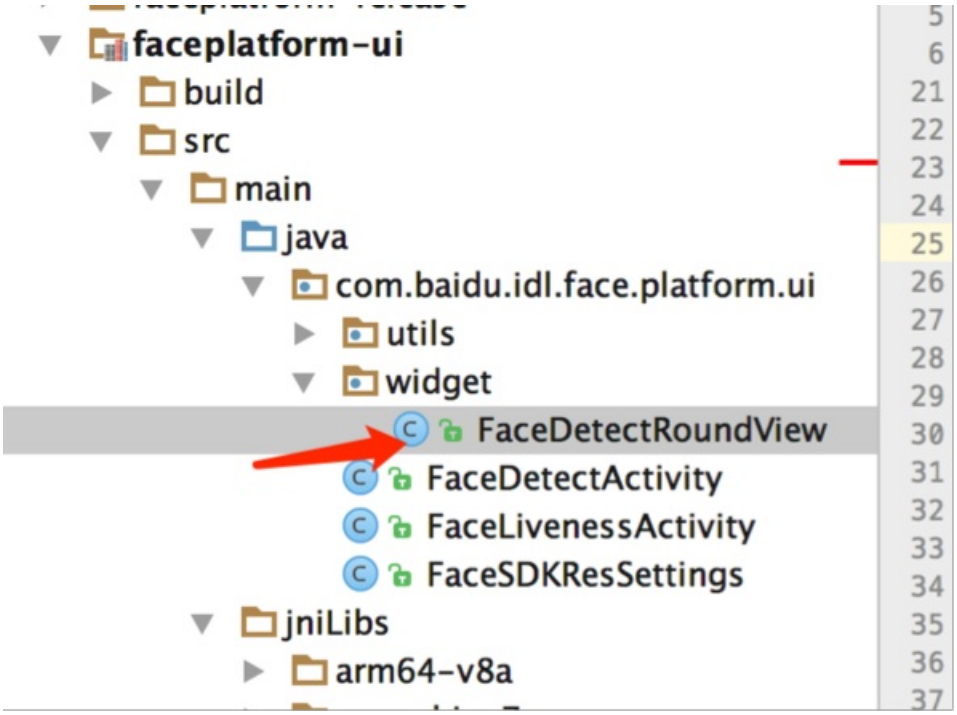
com.baidu.idl.face.platform.FaceConfig类用于人脸检测参数设置。

参数	名称	默认值	取值范围
minFaceSize	最小人脸阈值	200	
notFaceValue	非人脸阈值	0.6f	0~1.0f
brightnessValue	图片最小光照阈值	宽松：30、正常：40、严格：60	0-255f
brightnessMaxValue	图片最大光照阈值	宽松：240、正常：220、严格：200	0-255f
blurnessValue	图像模糊阈值	宽松：0.8f、正常：0.6f、严格：0.4f	0~1.0f
occlusionLeftEyeValue	左眼遮挡阈值	宽松：0.95f、正常：0.8f、严格：0.4f	0~1.0f
occlusionRightEyeValue	右眼遮挡阈值	宽松：0.95f、正常：0.8f、严格：0.4f	0~1.0f
occlusionNoseValue	鼻子遮挡阈值	宽松：0.95f、正常：0.8f、严格：0.4f	0~1.0f
occlusionMouthValue	嘴巴遮挡阈值	宽松：0.95f、正常：0.8f、严格：0.4f	0~1.0f
occlusionLeftContourValue	左脸颊遮挡阈值	宽松：0.95f、正常：0.8f、严格：0.4f	0~1.0f
occlusionRightContourValue	右脸颊遮挡阈值	宽松：0.95f、正常：0.8f、严格：0.4f	0~1.0f
occlusionChinValue	下巴遮挡阈值	宽松：0.95f、正常：0.8f、严格：0.4f	0~1.0f
headPitchValue	低头抬头角度	宽松：30、正常：20、严格：15	0~45
headYawValue	左右摇头角度	宽松：18、正常：18、严格：15	0~45
headRollValue	偏头角度	宽松：30、正常：20、严格：15	0~45
eyeClosedValue	闭眼阈值	0.7f	0~1.0f
cacheImageNum	图片缓存数量	3	建议3~6

3.4 界面定制说明

3.4.1 修改faceplatform_ui界面

（1）如果SDK自带的UI和您的APP不统一，您可以自行修改FaceDetectRoundView。



（2）修改提示语音音频文件，有两种方式。

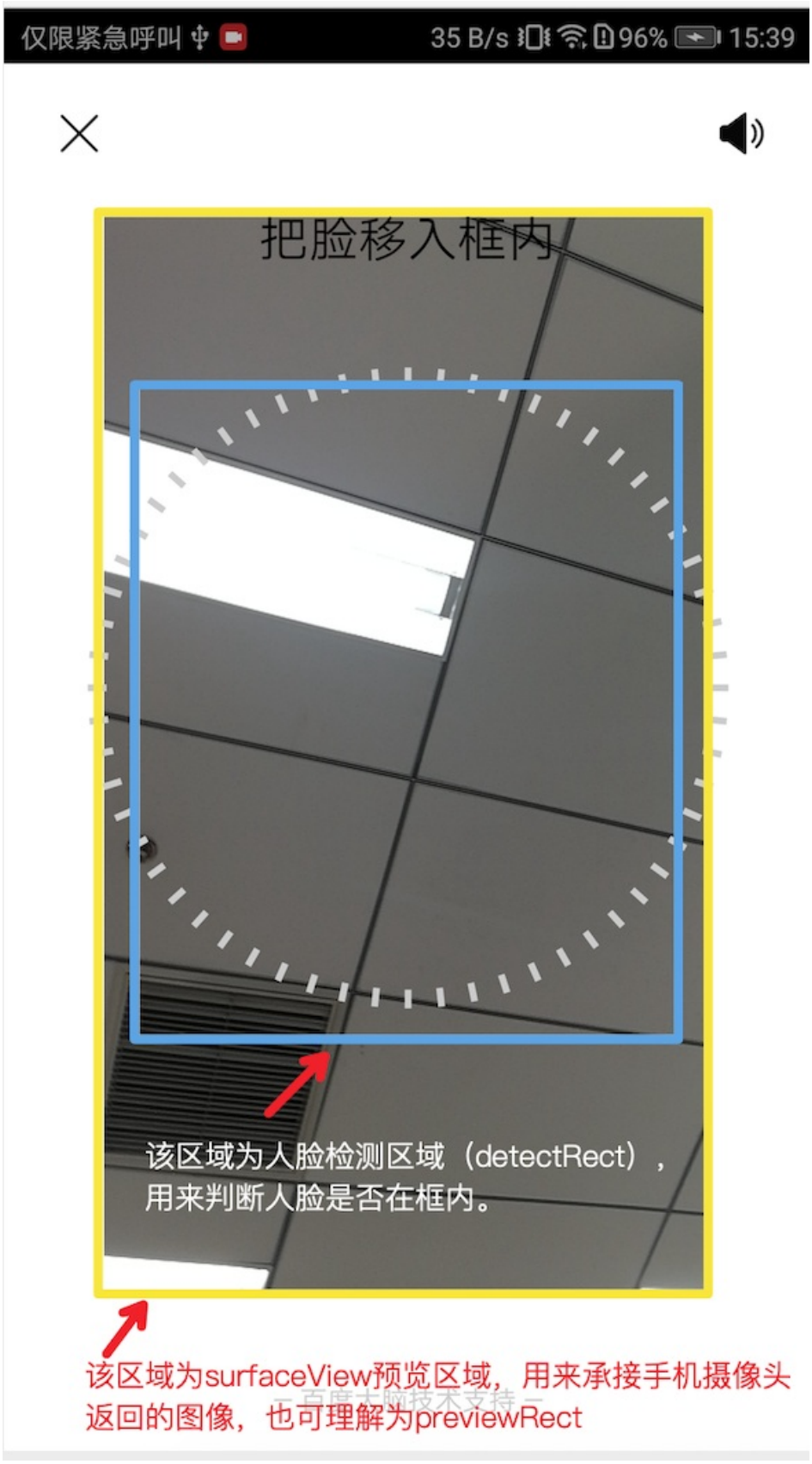
a、直接替换FaceUI工程raw下的mp3文件和string.xml。

b、FaceEnvironment 提供了setSoundId(FaceStatusNewEnum status, int soundId); 设置提示音资源。 FaceStatusNewEnum为

不同的状态。soundId为资源文件所对应的resource id。

3.5 采集中的预览区域和检测区域说明

如下图所示（图中背景设置为透明了，可以看出这两个区域）：



其中： surfaceView宽高为屏幕的宽高； previewRect的宽高为手机摄像头的宽高； detectRect的宽高是以previewRect为基准计算出来的。 计算方式参照FaceDetectRoundView.java的Rect getPreviewDetectRect(int w, int pw, int ph)方法

4、接口设计说明#

4.1 人脸功能管理器

4.1.1创建实例

- 方法

```
FaceSDKManager getInstance()
```

- 参数

```
无
```

- 返回

```
人脸功能管理器
```

- 说明

```
创建人脸功能管理器
```

4.1.2 人脸功能管理器初始化

- 方法

```
public void initialize(final Context context, String licenseID, String licenseFileName, IInitCallback callback)
```

- 参数

```
context 上下文环境，licenseID 传入申请License时获取的应用名称+_face_android后缀，licenseFileName 鉴权文件名称， callback 鉴权成功与否回调
```

- 返回

```
无
```

- 说明

```
初始化人脸检测功能。进行人脸检测功能License鉴权验证。
```

4.1.3 设置人脸功能控制参数

- 方法

```
void setFaceConfig(FaceConfig config)
```

- 参数

```
config 人脸功能控制参数对象
```

- 返回

```
无
```

- 说明

设置人脸功能控制参数对象。

FaceConfig对象参数：

最小光照阈值

最大光照阈值

模糊阈值

遮挡阈值

头部姿态角度阈值

最小人脸检测阈值

非人脸阈值

人脸抠图宽高设置

进行活体检测的动作类型列表

超时时间设置

检测框远近比率设置等

4.1.4 取得人脸图像采集功能接口

- 方法

```
IDetectStrategy getDetectStrategyModule()
```

- 参数

  无

- 返回

  人脸图像采集功能接口

- 说明

  取得人脸图像采集功能接口。人脸图像采集接口完成，解析图片人脸信息，返回检测结果。

4.1.5 取得活体检测功能接口

- 方法

```
ILivenessStrategy getLivenessStrategyModule()
```

- 参数

  无

- 返回

  活体检测功能接口

- 说明

  取得活体检测功能接口。活体检测功能接口完成，解析图片人脸信息，返回活体检测结果。

4.1.6 内存释放接口

- 方法

```
void release()
```

- 参数

  无

- 返回

  无

- 说明

  主要针对模型的释放，以减少内存

4.2 人脸图像采集器

人脸图像采集器IDetectStrategy，检测图片中人脸信息，返回人脸检测状态，完成人脸图像采集。

4.2.1 设置人脸图像采集功能参数

- 方法

```
void setDetectStrategyConfig(Rect previewRect, Rect detectRect,IDetectStrategyCallback callback)
```

- 参数

  previewRect 人脸图片大小，类型：Rect

  detectRect 人脸检测区域大小，类型：Rect

  callback 人脸图像采集功能状态监听器

- 返回

  无

- 说明

  设置人脸功能控制参数对象。

4.2.2 人脸图像采集

- 方法

```
void detectStrategy(byte[] imageData)
```

- 参数

imageData 图片信息

- 返回

无

- 说明

检测图片中的人脸信息，完成人脸图像采集，返回检测状态和结果。

4.3 活体检测器

活体检测器LivenessStrategy，检测图片人脸信息，活体检测结果状态。

4.3.1 设置人脸功能控制参数

- 方法

```
voidsetLivenessStrategyConfig(
        List<LivenessTypeEnum> livenessList,
        Rect previewRect,
        Rect detectRect,
        ILivenessStrategyCallback callback);
```

- 参数

livenessList 活体动作列表

previewRect 人脸图片大小，类型：Rect

detectRect 人脸检测区域大小，类型：Rect

callback 人脸图像采集功能状态监听器

- 返回

无

- 说明

设置活体检测功能参数对象。

4.3.2 活体检测

- 方法

```
void livenessStrategy(byte[] imageData);
```

- 参数

imageData 图片信息

- 返回

```
无
```

- 说明

```
检测图片中的人脸信息，完成人脸图像采集和人脸活体检测，返回检测状态和结果。
```

4.4 人脸图像采集界面

人脸图像采集器界面FaceDetectActivity，包括UI界面，系统相机控制，使用人脸图像采集器IDetectStrategy处理相机采集到的图像。

4.5 活体检测界面

活体检测界面FaceLivenessActivity，包括UI界面，系统相机控制，使用活体检测器ILivenessStrategy处理相机采集到的图像，完成活体检测功能。

4.6 图片加密请求类

图片加密请求类SecRequest，里面的sendMessage方法是为了测试图片加密之后secBase64是否可以通过云端解密的一个示例代码。 **4.7 ZID参数获取** 调用增强级实名认证接口配合增强级SDK使用时，需传入zid参数打开大数据风控功能。

获取方法：

```
FaceSDKManager.getInstance().getZid(Context)
```

5、常见问题

（1）**license**文件有什么用，该放在什么地方？

license文件需要申请，目的是作为sdk校验开发者的使用合法性，license文件放置位置不对或未放置license文件会导致没法使用sdk，一般应先申请license文件，并把申请得到的license文件，放置在assets目录下面。 授权延期后要重新替换到SDK中，否则会因为授权过期导致无法使用。

（2）鉴权初始化的错误码对应的信息

ErrorCode	常量值	说明
SUCCESS	0	成功
LICENSE_NOT_INIT_ERROR	1	license未初始化
LICENSE_DECRYPT_ERROR	2	license数据解密失败
LICENSE_INFO_FORMAT_ERROR	3	license数据格式错误
LICENSE_KEY_CHECK_ERROR	4	license-key(api-key)校验错误
LICENSE_ALGORITHM_CHECK_ERROR	5	算法ID校验错误
LICENSE_MD5_CHECK_ERROR	6	MD5校验错误
LICENSE_DEVICE_ID_CHECK_ERROR	7	设备ID校验错误
LICENSE_PACKAGE_NAME_CHECK_ERROR	8	包名(应用名)校验错误
LICENSE_EXPIRED_TIME_CHECK_ERROR	9	过期时间不正确
LICENSE_FUNCTION_CHECK_ERROR	10	功能未授权
LICENSE_TIME_EXPIRED	11	授权已过期
LICENSE_LOCAL_FILE_ERROR	12	本地文件读取失败
LICENSE_REMOTE_DATA_ERROR	13	远程数据拉取失败
LICENSE_LOCAL_TIME_ERROR	14	本地时间校验错误
OTHER_ERROR	0xff	其他错误

（3）**FaceSDK-LicenseLICENSE_INFO_CHECK_ERROR =license id**

licenseID为您申请时填appname+“_face_android”。如下图demo-turnstile-face-android为license里面的licenseID, **demo-turnstile-face-android1**为app运行时Config.licenseID，两者必须一致

```
E/FaceSDK: FaceSDK-License LICENSE_INFO_CHECK_ERROR =license id  demo-turnstile-face-android  demo-turnstile-face-android1
```

（4）**FaceSDK-LicenseLICENSE_INFO_CHECK_ERROR =signature md5**

md5不一致错误，签名的为license里面的md5，后面的为app运行时获取的签名文件的md5，这两个md5必须一致且区分大小写。 `E/FaceSDK: FaceSDK-License LICENSE_INFO_CHECK_ERROR =signature md5  F5846C60804CC6042D55D09F1AB82364  4357A3EDBC0CA02EA8B5E0578E58D`

（5）**FaceSDK-License LICENSE_INFO_CHECK_ERROR =package name**

PackageName不一致错误。License里面的packagename为申请license时填的，需要保证和app里面的packagename一致。

（6）活体检测常见有那些动作？是否可配置？

常见有6个动作，眨眼，张大嘴，向上抬头，向下低头，向左摇头，向右摇头等。 sdk提供FaceConfig参数设置类，如活体检测角度、光线，检测动作，检测动作数量等设置。

（7）使用sdk一般会用到活体检测拍照等功能，有什么需要注意？

Android 6.0+，需要注意相机拍摄权限问题。如没申请权限，可能导致没法调起相机。

（8）在有些机型上出现特别卡或出现无响应？

SDK在**armeabi**上性能非常差，建议删掉其他so只留下armeabi-v7a，包括使用的其他第三方so。因为如果其他so有armeabi，根据android系统查找so的逻辑，在armeabi的机型上只会去该目录下查找so，而人脸SDK没有，就会出现找不到so。

（9）**license** 文件失效了，不能用了怎么办？

license文件申请时候有期限，如过期会导致校验失效，需要在后台申请延期。

IOS

1.简介

1.1 功能介绍

百度Face SDK IOS 版是一种面向 IOS 移动设备人脸技术开发包，此版SDK包含人脸检测、活体识别等功能。基于该方案，开发者可以轻松的构建包含人脸检测、采集和活体识别的应用。在您使用SDK之前，我们首先为您介绍以下SDK本身及相关人脸能力，以便您能更方便使用人脸服务。

此版SDK所包含的能力如下：

- **离线动作活体检测**：通过让用户做出指定人脸配合式的交互动作，识别当前操作者是否为活体，此功能为离线使用，可设定指定动作是否使用及应用顺序。动作包含：眨眼、张嘴、左摇头，右摇头，向上抬头，向下低头六个。可有效抵御高清图片、3D建模、视频等攻击。
- **离线人脸质量检测**：判断视频流中的图片帧中，哪些图片质量较佳，即人脸图像特征清晰（满足遮挡、姿态、光照、模糊度等校验）。
- **离线人脸图像采集**：通过本地SDK能力，采集人脸图像，同时经过人脸质量检测，确保采集到的人脸图像符合各条件校验（遮挡、姿态、光照、模糊度等），为设备前端获取有效可分析人脸的主要功能。
- **离线授权**：SDK的授权判断，授权介质也称为license，在SDK使用中，优先验证本地离线鉴权文件，验证通过可离线使用；在本地鉴权失败情况下需要向授权服务器发起请求，远程拉取授权进行验证。

1.2 兼容性

- **系统**：支持iOS9以上系统。需要开发者通过Deployment Target 来保证支持系统的检测。
- **机型**：手机和平板皆可（暂不支持横屏）
- **架构**：arm64、armv7
- **网络**：支持 WIFI 及移动网络,移动网络支持使用 NET 网关及WAP 网关(CMWAP、CTWAP、UNIWAP、3GWAP)。

1.3 开发包说明

文件/文件夹名	说明
FaceSDK	FaceSDK 包、bundle 资源文件、bundle 模型文件、鉴权文件
Public/Common	视频流处理类
Public/Utils	图像处理类
UI/Controller	DEMO工程
UI/View	View控制类
FaceParameterConfig.h	配置信息

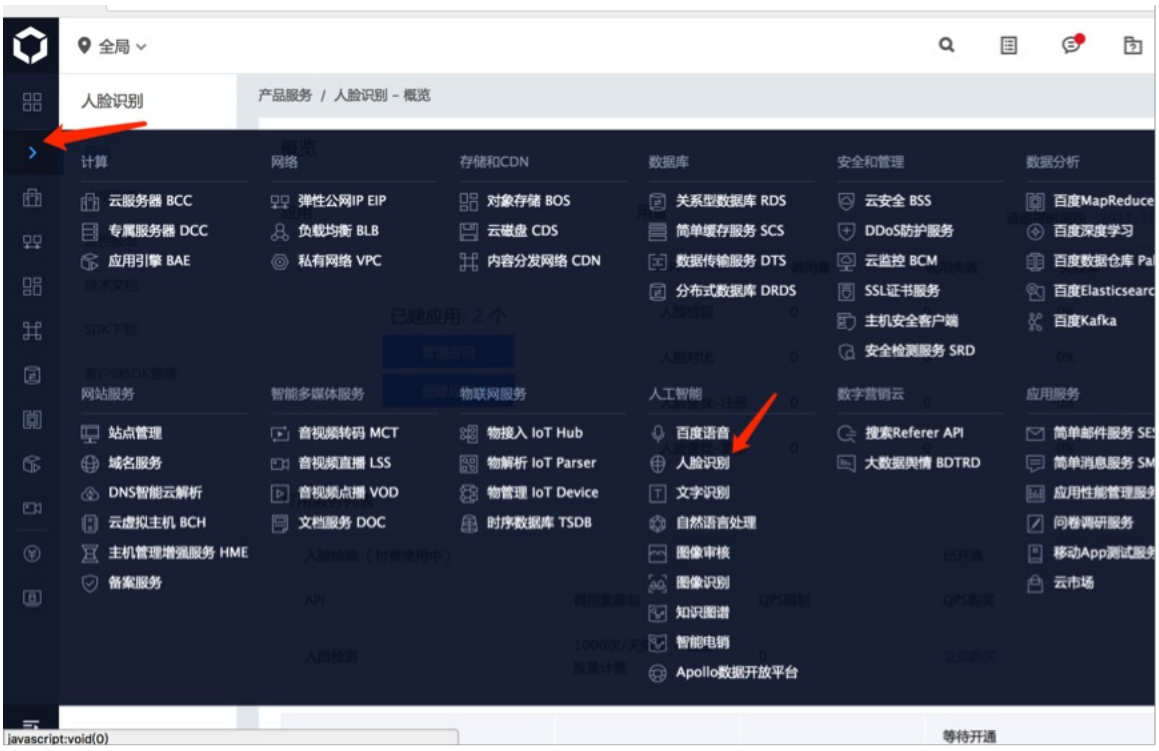
2.集成指南

2.1 准备工作

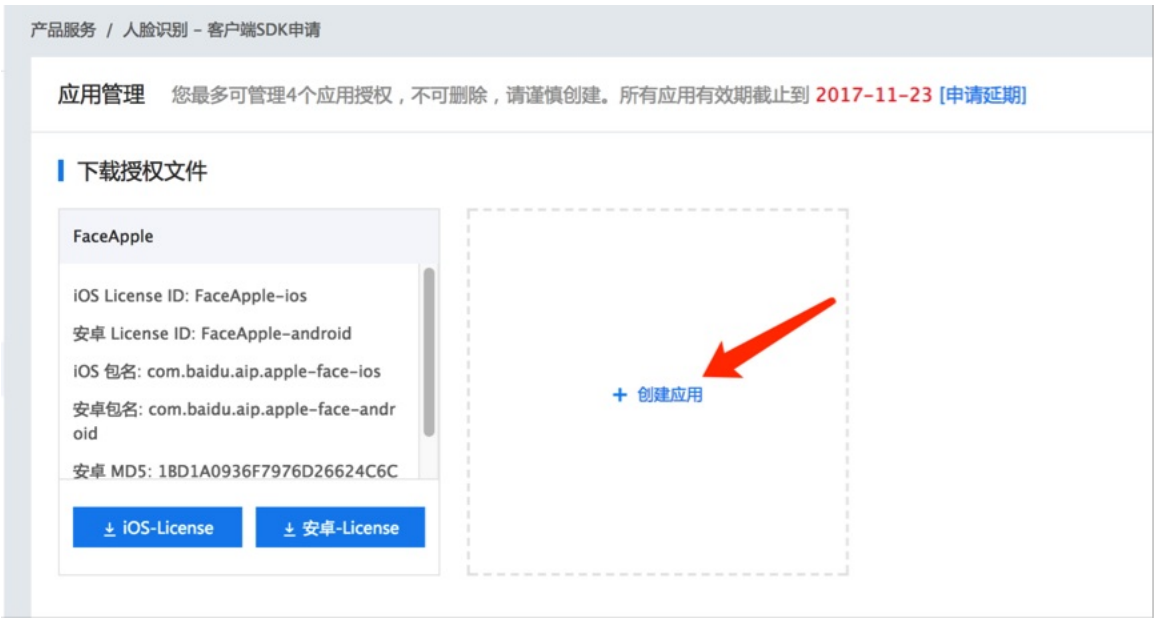
2.1.1 申请license

人脸SDK License：此license用于SDK离线功能使用，在您的申请人脸SDK的后台页面，全局->产品服务->人脸识别->客户端SDK申请

人脸控制台路径如下：



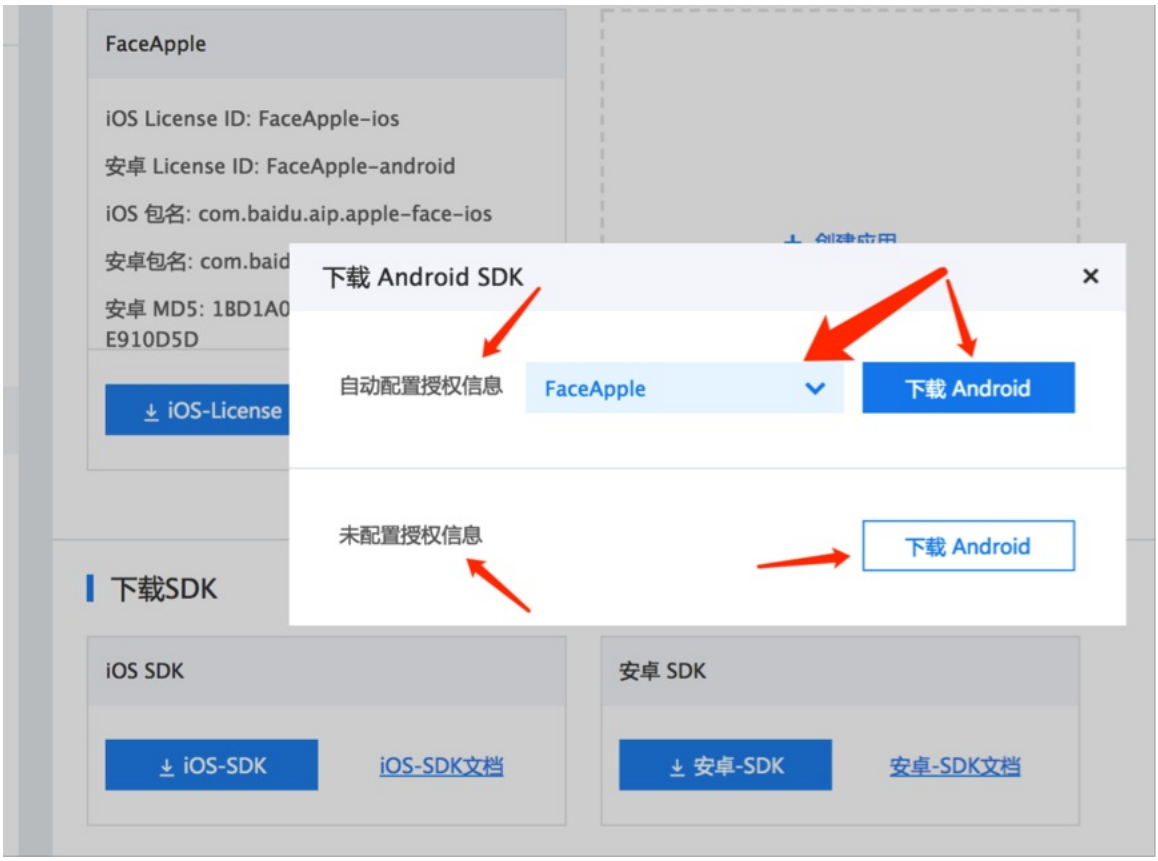
点击客户端SDK管理，弹出如下图：创建应用（这里创建应用是为了使用离线SDK，上面创建应用为了使用人脸在线接口，如注册、识别等）



在弹出的框中输入授权标识，选择应用类型，应用系统，以及包名，详情请查看输入框右边提示

2.1.2 下载SDK

下载SDK分为自动配置授权信息（创建license后就可以选择为该应用，下载后SDK自动帮您配置授权，不用下载license拷贝到工程中，初始化参数licenseID,包名也帮您配置好了）和未配置授权信息两种方式：



2.2 运行示例工程

2.2.1 自动配置授权信息集成

如果您是通过自动配置授权信息下载的示例工程，只需配置好证书即可。查看下项目中的FaceParameterConfig.h文件，已经自动配置

```
11
12 // 人脸license文件名
13 #define FACE_LICENSE_NAME    @"idl-license"
14
15 // 人脸license后缀
16 #define FACE_LICENSE_SUFFIX  @"face-ios"
17
18 //  (您申请的应用名称(appname)+「-face-ios」后缀，如申请的应用名称(appname)为test123，则此处填写test123-face-ios)
19 #define FACE_LICENSE_ID        @"BaiduFaceDemo-face-ios"
20
```

示例图

配置好证书，即可运行。注意已经设置好的bundle id不要随意改动。

2.2.2 未使用自动配置授权信息的集成

手动导入授权信息。 需要手动导入license文件，配置好bundleID、licenseID等信息:

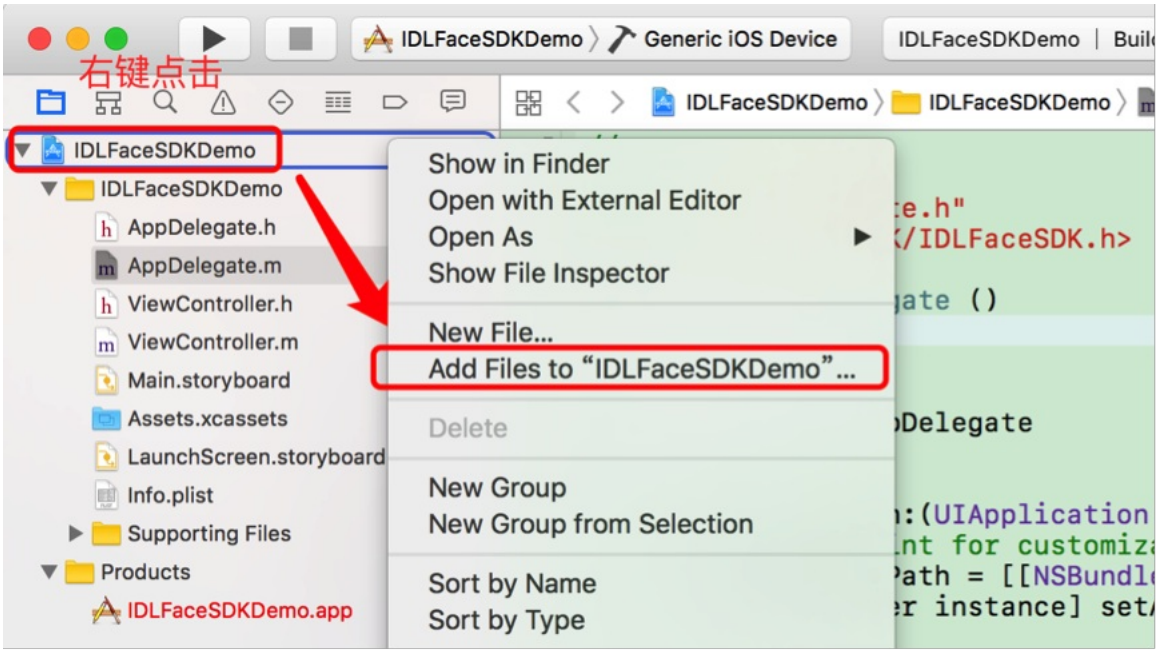
```
11
12 // 人脸license文件名
13 #define FACE_LICENSE_NAME    @"idl-license"
14
15 // 人脸license后缀
16 #define FACE_LICENSE_SUFFIX  @"face-ios"
17
18 //  (您申请的应用名称(appname)+「-face-ios」后缀，如申请的应用名称(appname)为test123，则此处填写test123-face-ios)
19 #define FACE_LICENSE_ID        @"BaiduFaceDemo-face-ios"
20
```

示例图

2.3 添加SDK到工程

1. 打开或者新建一个项目。

2. 右键点击项目，会出现一个添加菜单，在菜单中选择『Add Files to“此处是你的项目名字”…… 』,如下图所示：

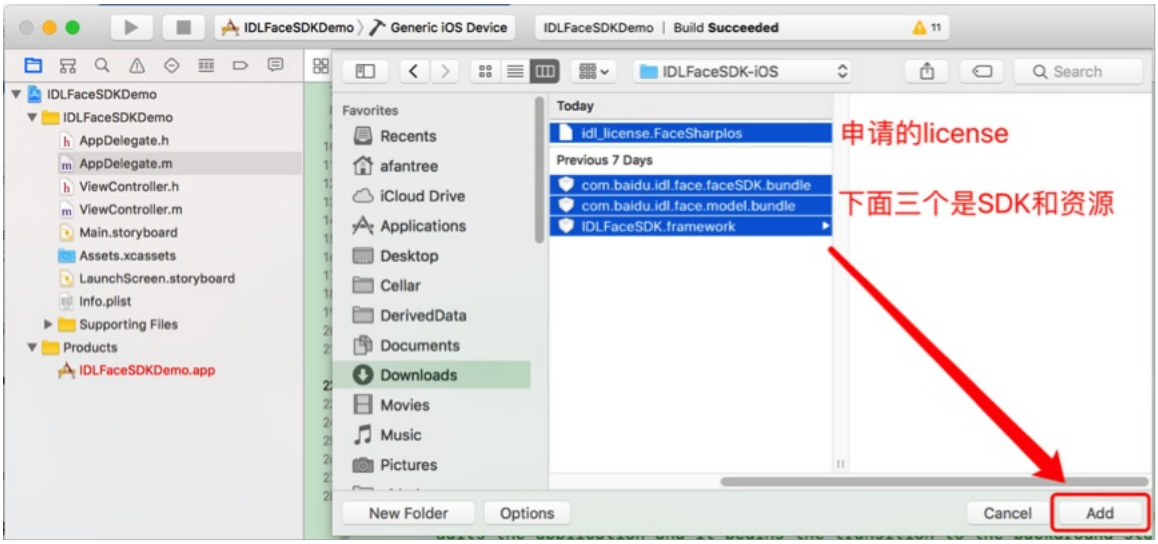


3. 在添加文件弹出框里面选择申请到的license和SDK添加进来。如下图：

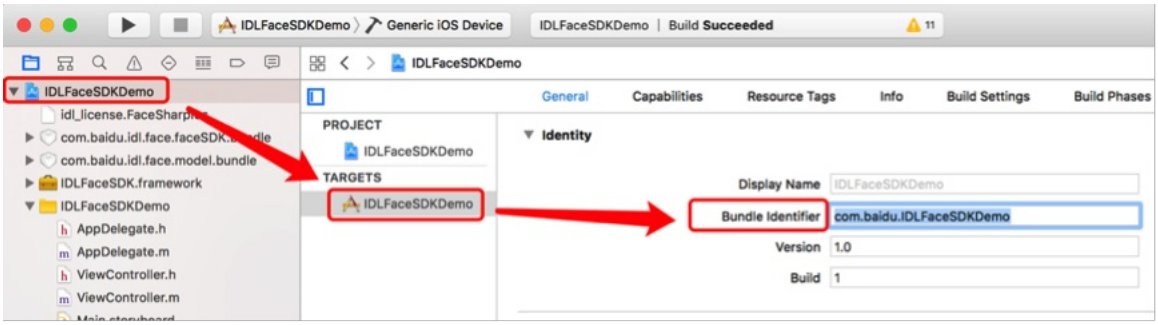
   注意：license为百度官方提供的。

   SDK包含下面三个文件:

   - IDLFaceSDK.framework
   - com.baidu.idl.face.faceSDK.bundle
   - com.baidu.idl.face.model.faceSDK.bundle



4. 确认下Bundle Identifier 是否是申请license时填报的那一个，注意：license和Bundle Identifier是一一对应关系，填错了会导致SDK不能用。



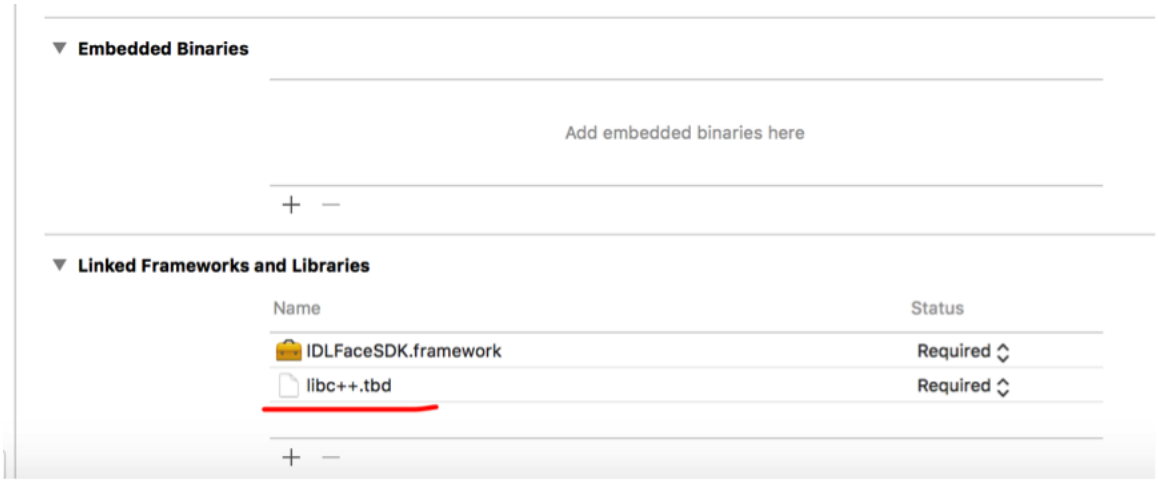
5. 填写正确的FACE_LICENSE_ID。

（即后台展示的LicenseID）

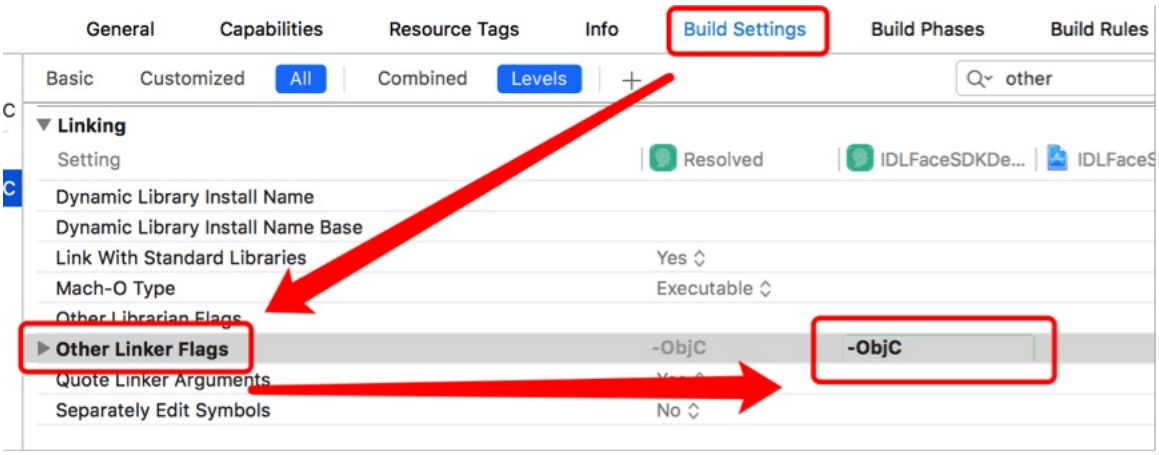
在FaceParameterConfig.h文件里面填写拼接好的FACE_LICENSE_ID。

```
//  (您申请的应用名称(appname)+「-face-ios」后缀，如申请的应用名称(appname)为test123，则此处填写test123-face-ios)
#define FACE_LICENSE_ID        @"BaiduFaceDemo-face-ios"
```

6. 选择链接C++标准库。



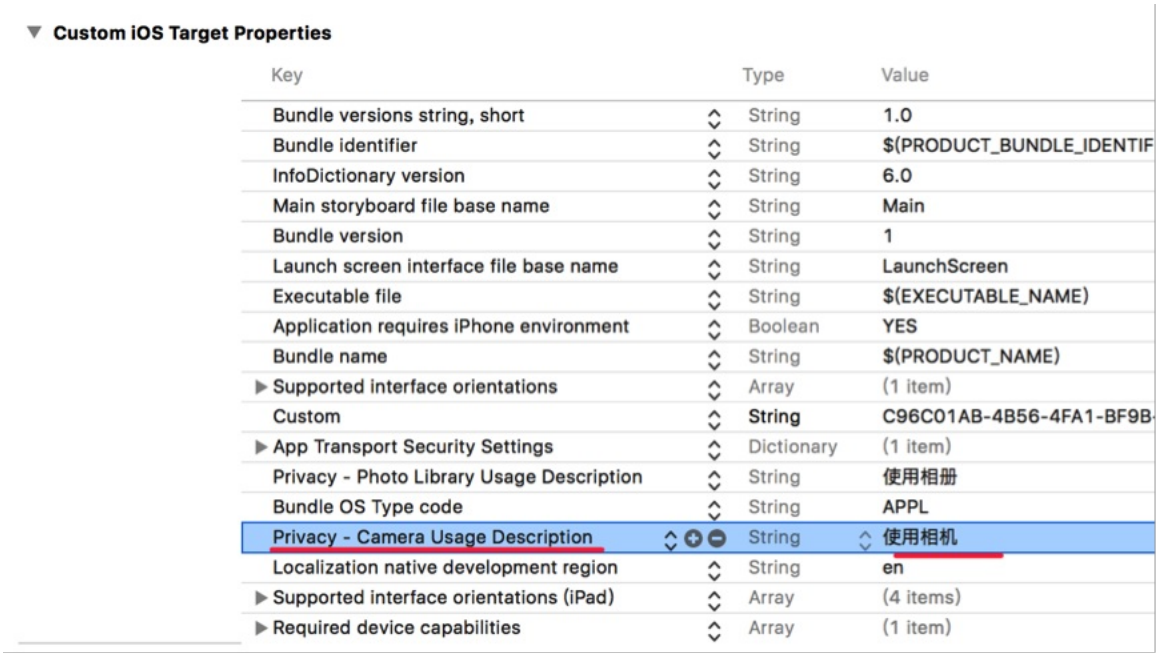
7. 如果没有使用pod管理第三方库的话，请在Build Setting >Linking > Other Linker Flags 上面加入 –ObjC 选项。如果用了pod请忽略，因为pod会自动添加上。



2.4 权限声明

需要使用相机权限：编辑Info.plist文件，添加

Privacy- Camera Usage Description 的Key值，Value为使用相机时候的提示语，可以填写：『使用相机』。



3.接口说明

3.1 FaceSDK 鉴权初始化

3.1.1 设置鉴权功能

```
 - (void)setLicenseID:(NSString *)licenseID andLocalLicenceFile:(NSString *)licensePath andRemoteAuthorize:(BOOL)remoteAuthorize;
```

参数：

- licenseId：平台申请的 licenseID
- localLicencePath：鉴权文件路径
- remoteAuthorize：是否开启网络鉴权

返回：

- 无

参考AppDelegate.m 实现，使用方法如下：

```
    NSString* licensePath = [[NSBundle mainBundle] pathForResource:FACE_LICENSE_NAME ofType:FACE_LICENSE_SUFFIX];
    NSAssert([[NSFileManager defaultManager] fileExistsAtPath:licensePath], @"license文件路径不对，请仔细查看文档");
    [[FaceSDKManager sharedInstance] setLicenseID:FACE_LICENSE_ID andLocalLicenceFile:licensePath andRemoteAuthorize:true];
```

3.1.2 鉴权成功的凭证

```
 - (BOOL)canWork
```

参数：

- 无

返回：

- True代表成功，false代表失败

参考ViewController.m 实现，判断鉴权是否通过：

```
 if (![[FaceSDKManager sharedInstance] canWork]){
  NSLog(@"授权失败，请检测ID 和 授权文件是否可用");
  return;
}
```

3.2 FaceSDK 功能初始化

3.2.1 FaceSDK 参数配置

具体方法详见如下：

```
 /**
 * 设置预测库耗能模式
 * 默认 LITE_POWER_NO_BIND
 */
- (void)setLitePower:(int)litePower;

/**
 * 需要检测的最大人脸数目
 * 默认1
 */
- (void)setMaxDetectNum:(int)detectNum；

/**
 * 需要检测的最小人脸大小
 * 默认40
 */
- (void)setMinFaceSize:(int)width;

/**
 * 人脸置信度阈值（检测分值大于该阈值认为是人脸
 * RGB
 * 默认 0.5f
 */
- (void)setNotFaceThreshold:(CGFloat)thr；

/**
 * 质量检测遮挡阈值
 * 默认0.5
 */
- (void)setOccluThreshold:(CGFloat)thr；

/**
 * 质量检测光照阈值
 * 默认100
 */
- (void)setIllumThreshold:(CGFloat)thr；

/**
 * 质量检测模糊阈值
 * 默认0.5
 */
- (void)setBlurThreshold:(CGFloat)thr；

/**
 * 姿态检测阈值
 * 默认pitch=12，yaw=12，row=10
 */
- (void)setEulurAngleThrPitch:(float)pitch yaw:(float)yaw roll:(float)roll；
```

```
/**
 * 输出图像个数
 * 默认3
 */
- (void)setMaxCropImageNum:(int)imageNum；

/**
 * 输出图像宽，设置为有效值(大于0)则对图像进行缩放，否则输出原图抠图结果
 * 默认 480
 */
- (void)setCropFaceSizeWidth:(CGFloat)width；

/**
 * 输出图像高，设置为有效值(大于0)则对图像进行缩放，否则输出原图抠图结果
 * 默认 680
 */
- (void)setCropFaceSizeHeight:(CGFloat)height；

/**
 * 输出图像，下巴扩展，大于等于0，0：不进行扩展
 * 默认0.1
 */
- (void)setCropChinExtend:(CGFloat)chinExtend；

/**
 * 输出图像，额头扩展，大于等于0，0：不进行扩展
 * 默认0.2
 */
- (void)setCropForeheadExtend:(CGFloat)foreheadExtend；

/**
 * 输出图像，人脸框与背景比例，大于等于1，1：不进行扩展
 * 默认1.5f
 */
- (void)setCropEnlargeRatio:(float)cropEnlargeRatio;

/**
 * 动作超时配置
 */
- (void)setConditionTimeout:(CGFloat)timeout；

/**
 * 语音间隔提醒配置
 */
- (void)setIntervalOfVoiceRemind:(CGFloat)timeout;

/**
 * 是否开启静默活体，默认false
 */
- (void)setIsCheckSilentLive:(BOOL)isCheck;

/**
 * 口罩检测阈值配置，默认0.8。
 * 大于阈值判定为戴口罩，低于阈值判定为未戴口罩
 */
- (void)setMouthMaskThreshold:(CGFloat)thr；

/**
 * 设置原始图片缩放比例，默认1不缩放，scale 阈值0~1
 */
- (void)setImageWithScale:(CGFloat)scale;
```

```
/**
 * 设置图片加密类型，type=0 非加密原图base64 ；type=1 基于百度安全算法加密
 */
- (void)setImageEncrypteWithType:(int) type;

/**
 *  人脸过远框比例 默认：0.4
 */
- (void)setMinRect:(float) minRectScale;
```

3.2.2 FaceSDK 初始化

```
 - (int)initCollect;
```

参数:

- 无

返回：

- 无

参考ViewController.m initSDK 方法实现：

```
- (void) initSDK {

    if (![[FaceSDKManager sharedInstance] canWork]){
        NSLog(@"授权失败，请检测ID 和 授权文件是否可用");
        return;
    }

    // 初始化SDK配置参数，可使用默认配置
    // 设置最小检测人脸阈值
    [[FaceSDKManager sharedInstance] setMinFaceSize:200];
    // 设置截取人脸图片高
    [[FaceSDKManager sharedInstance] setCropFaceSizeWidth:480];
    // 设置截取人脸图片宽
    [[FaceSDKManager sharedInstance] setCropFaceSizeHeight:640];
    // 设置人脸遮挡阈值
    [[FaceSDKManager sharedInstance] setOccluThreshold:0.5];
    // 设置亮度阈值
    [[FaceSDKManager sharedInstance] setIllumThreshold:40];
    // 设置图像模糊阈值
    [[FaceSDKManager sharedInstance] setBlurThreshold:0.3];
    // 设置头部姿态角度
    [[FaceSDKManager sharedInstance] setEulurAngleThrPitch:10 yaw:10 roll:10];
    // 设置人脸检测精度阈值
    [[FaceSDKManager sharedInstance] setNotFaceThreshold:0.6];
    // 设置抠图的缩放倍数
    [[FaceSDKManager sharedInstance] setCropEnlargeRatio:2.5];
    // 设置照片采集张数
    [[FaceSDKManager sharedInstance] setMaxCropImageNum:3];
    // 设置超时时间
    [[FaceSDKManager sharedInstance] setConditionTimeout:1500];
    // 设置开启口罩检测，非动作活体检测可以采集戴口罩图片
    [[FaceSDKManager sharedInstance] setIsCheckMouthMask:true];
    // 设置开启口罩检测情况下，非动作活体检测口罩过滤阈值，默认0.8 不需要修改
    [[FaceSDKManager sharedInstance] setMouthMaskThreshold:0.8f];
    // 设置原始图缩放比例
    [[FaceSDKManager sharedInstance] setImageWithScale:0.8f];
    // 设置图片加密类型，type=0 非加密原图base64 ；type=1 基于百度安全算法加密
    [[FaceSDKManager sharedInstance] setImageEncrypteType:0];
    // 设置人脸过远框比例
    [[FaceSDKManager sharedInstance] setMinRect:0.4];
    // 初始化SDK功能函数
    [[FaceSDKManager sharedInstance] initCollect];
}
```

3.2.3 FaceSDK 释放

```
- (int)uninitCollect
```

参数:

- 无

返回：

- 无

3.3 人脸采集

调用IDLFaceDetectionManager类的:

```
 /**
 * 人脸采集，成功之后返回扣图图片，原始图片
 * @param image 镜头拿到的图片
 * @param previewRect 预览的Rect
 * @param detectRect 检测的Rect
 * return completion 回调信息
 */
- (void)detectStratrgyWithNormalImage:(UIImage *)image previewRect:(CGRect)previewRect detectRect:(CGRect)detectRect completionHandler:(DetectStrategyCompletion)completion;

typedef void (^DetectStrategyCompletion) (FaceInfo * faceinfo,NSDictionary * images, DetectRemindCode remindCode);
```

温馨提示：NSDictionary * images 返回FaceCropImageInfo 对象包含带黑边的图片，主要是为了配合在线 API 方式的活体检测使用，本地算法会对图片做一定的预处理操作；而不带黑边的图片，如果无需使用活体，则可使用不带黑边的图片采集方式。

参数说明：

`previewRect`与`detctRect`是为了做距离检测而定义的，为了在上层封装判断脸是否在框内／离太远／离太近。

- image：相机获取的图片
- previewRect： 人脸图片大小，间接定义的最大距离的 maxRect 和最小距离的 minRect。
- detectRect：人脸检测区域大小，实际采集区域
- completion：完成后返回照片和状态结果

参考BDFaceDetectionViewController.m 实现，使用方法如下：

```
        __weak typeof(self) weakSelf = self;
    [[IDLFaceDetectionManager sharedInstance] detectStratrgyWithNormalImage:image previewRect:self.previewRect
detectRect:self.detectRect completionHandler:^(FaceInfo *faceinfo, NSDictionary *images, DetectRemindCode
remindCode) {
        switch (remindCode) {
          case DetectRemindCodeOK: {
              weakSelf.hasFinished = YES;
              [self warningStatus:CommonStatus warning:@"非常好"];
              if (images[@"image"] != nil && [images[@"image"] count] != 0) {

                  NSArray *imageArr = images[@"image"];
                  for (FaceCropImageInfo * image in imageArr) {
                      NSLog(@"cropImageWithBlack %f %f", image.cropImageWithBlack.size.height,
image.cropImageWithBlack.size.width);
                      NSLog(@"originalImage %f %f", image.originalImage.size.height, image.originalImage.size.width);
                  }

                  FaceCropImageInfo * bestImage = imageArr[0];
                  [[BDFaceImageShow sharedInstance] setSuccessImage:bestImage.originalImage];
                  [[BDFaceImageShow sharedInstance] setSilentliveScore:bestImage.silentliveScore];
                  // 公安验证接口测试
                  [self request:bestImage.cropImageWithBlackEncryptStr];
                  dispatch_async(dispatch_get_main_queue(), ^{
                      UIViewController* fatherViewController = weakSelf.presentingViewController;
                      [weakSelf dismissViewControllerAnimated:YES completion:^{
                          BDFaceSuccessViewController *avc = [[BDFaceSuccessViewController alloc] init];

                          [fatherViewController presentViewController:avc animated:YES completion:nil];
                      }];

                  });
              }
              [self singleActionSuccess:true];
              break;
```

3.4 动作采集

调用IDLFaceLivenessManager类的:

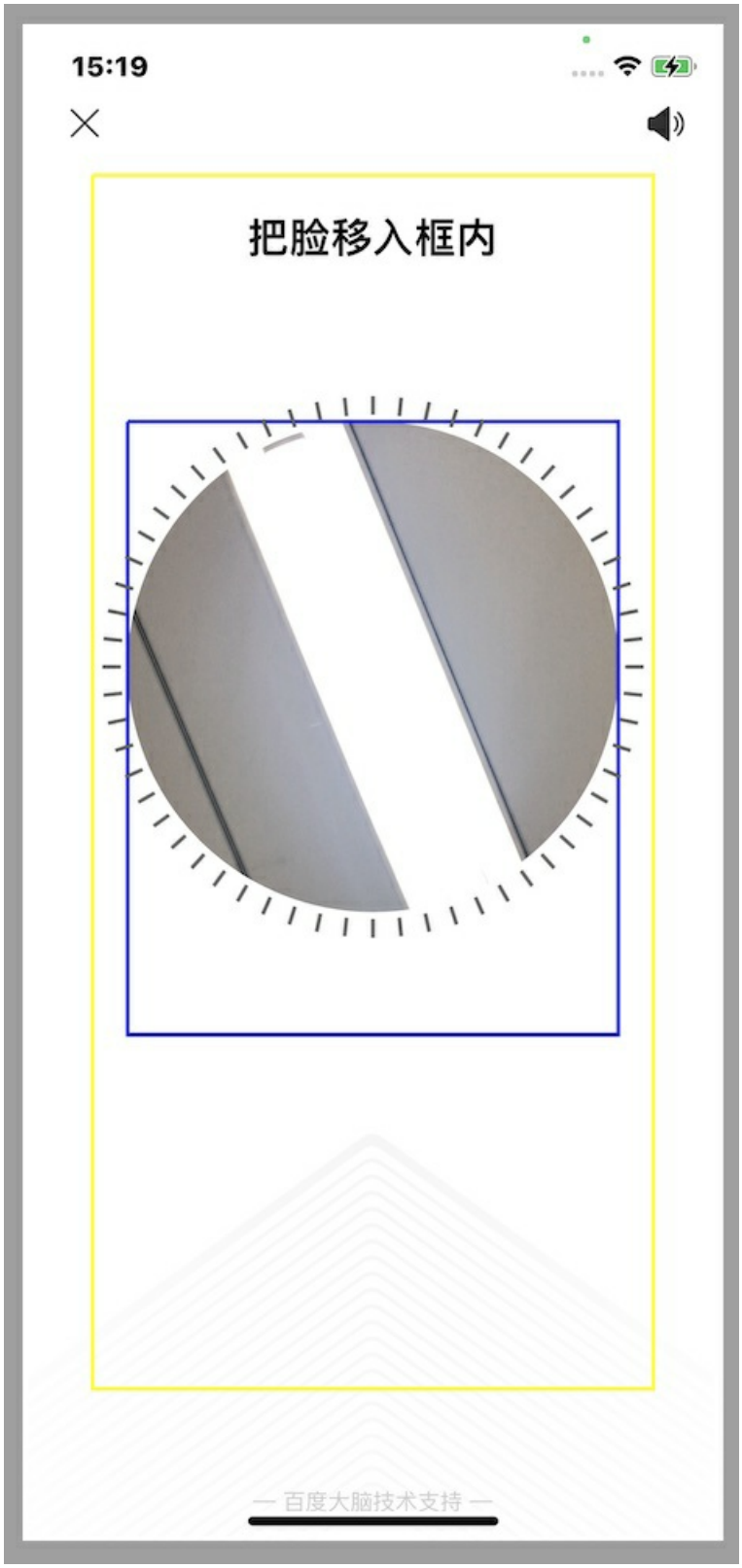
```
 /**
 *  人脸活体验证，成功之后扣图图片，原始图片三种
 * @param image 镜头拿到的图片
 * @param previewRect 预览的Rect
 * @param detectRect 检测的Rect
 * return completion 回调信息
 */
-(void) livenessNormalWithImage:(UIImage *)image previewRect:(CGRect)previewRect detectRect:(CGRect)detectRect
completionHandler:(LivenessNormalCompletion)completion;

typedef void (^LivenessNormalCompletion) (NSDictionary * images, FaceInfo *faceInfo, LivenessRemindCode
remindCode);
```

温馨提示：NSDictionary * images 返回FaceCropImageInfo 对象包含带黑边的图片，主要是为了配合在线 API 方式的活体检测使用，本地算法会对图片做一定的预处理操作；而不带黑边的图片，如果无需使用活体，则可使用不带黑边的图片采集方式。

参数说明：

`previewRect`与`detctRect`是为了做距离检测而定义的，为了在上层封装判断脸是否在框内／离太远／离太近。

- image：相机获取的图片
- previewRect： 人脸图片大小(下图黄色框)
- detectRect：人脸检测区域大小，实际采集区域，间接定义的最大距离的 maxRect 和最小距离的 minRect。（下图蓝色框）
- completion：完成后返回照片和状态结果



说明：

- 检测图片中的人脸信息，完成人脸图像采集和人脸活体检测，返回检测状态和结果。

```
    [[IDLFaceLivenessManager sharedInstance] livenessNormalWithImage:image previewRect:self.previewRect
detectRect:self.detectRect completionHandler:^(NSDictionary *images, FaceInfo *faceInfo, LivenessRemindCode
remindCode) {

        switch (remindCode) {
            case LivenessRemindCodeOK: {
                weakSelf.hasFinished = YES;
                [self warningStatus:CommonStatus warning:@"非常好"];
                if (images[@"image"] != nil && [images[@"image"] count] != 0) {

                    NSArray *imageArr = images[@"image"];
                    for (FaceCropImageInfo * image in imageArr) {
                        NSLog(@"cropImageWithBlack %f %f", image.cropImageWithBlack.size.height,
image.cropImageWithBlack.size.width);
                        NSLog(@"originalImage %f %f", image.originalImage.size.height, image.originalImage.size.width);
                    }

                    FaceCropImageInfo * bestImage = imageArr[0];
```

3.5 活体动作设置

调用IDLFaceLivenessManager类的:

```
- (void)livenesswithList:(NSArray *)array order:(BOOL)ordernumberOfLiveness:(NSInteger)numberOfLiveness
```

参数：

- array: 活体动作列表
- order: 是否按顺序进行活体动作
- numberOfLiveness: 活体动作数目（array为nil是起作用）

返回：

- 无

说明：

- 活体动作设置 **3.6 zid参数获取** 调用增强级实名认证接口配合增强级SDK使用时，需传入zid参数打开大数据风控功能。

获取方法：

```
    [[FaceSDKManager sharedInstance] getZtoken]
```

4、常见问题

Q：鉴权问题。提示「验证失败」 A：先确定网络情况是否正常，本地鉴权文件失效了才走网络鉴权。定位错误码，排查鉴权失败的原因。一般是 licenseID 和 bundleID 配置不一致导致的鉴权失败。请注意上线前授权文件一定要更新。

Q：license 文件失效了，不能用了怎么办？ A：License 文件申请时候有期限，如过期会导致校验失效，需要在后台进行申请延期。

Q：使用 iOS 采集端，采集到的图片是斜着的，这个正常吗，会影响识别吗？ A：不会影响识别。有黑边和倾斜是因为图片质量算法造成的，我们是按 1:3 对图像进行背景填充使人脸居中，为的是更好的识别图像。这个版本提供了`detectStratrgyWithQualityControlImage`和`detectStratrgyWithNormalImage`两种方法供选择。

更多问题请点击 [常见问题](

金融级SDK

Android-1.0

Android端人脸采集SDK金融1.0使用文档

1.简介

百度Face SDK Android 版是一种面向 Android 移动设备人脸技术开发包，此版SDK包含人脸检测、炫瞳活体识别等功能，以aar包+动态链接库的形式发布。

基于该方案，开发者可以轻松的构建包含人脸检测、采集和活体识别的应用。在您使用SDK之前，我们首先为您介绍以下SDK本身及相关人脸能力，以便您能更方便使用人脸服务。

1.1 功能介绍

此版SDK所包含的能力如下：

- **离线炫瞳活体检测**：通过屏幕上闪烁不同颜色的光线，判断当前用户是否真人操作。通过颜色活体进行面部反光鉴别的同时，百度特加入独有的瞳孔反光识别，提升整体的攻击拒绝率指标，可有效抵御高清图片、3D建模、视频等攻击。
- **端云互验加密：**在采集SDK端加入端云互验加密功能，在采集SDK端对数据进行加密，云端进行解密，从而可以有效识别第三方非法黑产通过绕过APP用脚本攻击接口的行为发生。
- **大数据风控：**与百度安全实验室联合推出大数据风控功能，与云端接口配合对设备端进行环境校验，可有效识别ROM注入、视频劫持、云手机等行为。
- **离线人脸质量检测**：判断视频流中的图片帧中，哪些图片质量较佳，即人脸图像特征清晰（满足姿态角、光照、模糊度、遮挡等校验）。
- **离线人脸图像采集**：通过本地SDK能力，采集人脸图像，同时经过人脸质量检测，确保采集到的人脸图像符合各条件校验（满足姿态角、光照、模糊度、遮挡等校验），为设备前端获取有效可分析人脸的主要功能。
- **离线授权**：SDK的授权判断，授权介质也称为license，在SDK使用中，优先验证本地离线鉴权文件，验证通过可离线使用；在本地鉴权失败情况下需要向授权服务器发起请求，远程拉取授权进行验证。

> 此版SDK**全部功能为离线版本，所有功能均本地化使用**，主要用于在客户端（Android）获取人脸，实际业务使用中，可以按照业务需要，配合在线API完成全流程的业务集成。

为了方便您的开发，我们已经为您准备了【人脸实名认证】场景的示例工程，您可以根据业务需要，在后台进行直接下载，示例工程参考下图：

人脸实名认证示例工程

适用场景：移动端的远程身份验证，金融开户、用户实名认证等
核心功能：进行有动作活体检测，配合在线活体API，提供手机场景下的较强的活体检测能力
下载使用：在实名认证配置中心配置功能后即可下载示例代码集成到手机使用

配置说明 配置中心

1.2 兼容性

系统：支持 Android 5.1(API Level 22)及以上系统。需要开发者通过 minSdkVersion来保证支持系统的检测。

机型：手机和平板皆可（暂不支持横屏）

构架：支持 CPU架构平台【armeabi-v7a、arm64-v8a】

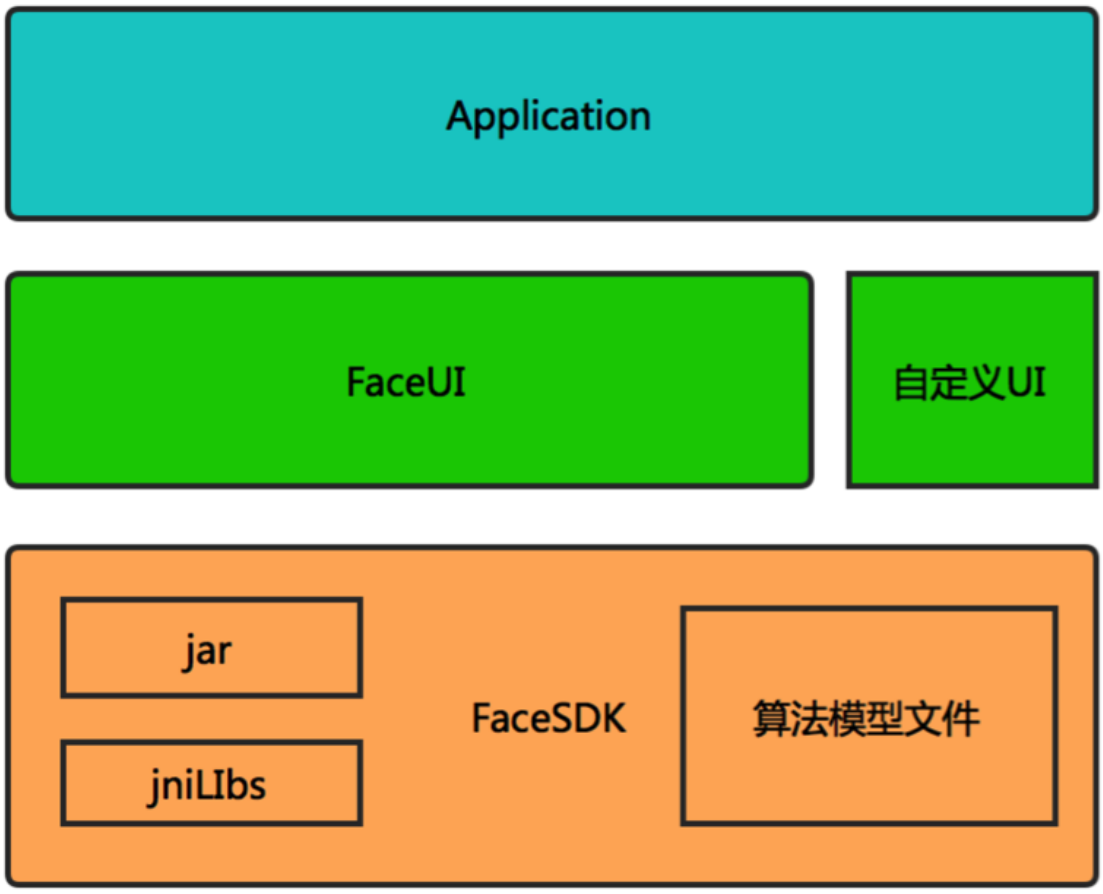
网络：支持 WIFI 及移动网络。

1.3 开发包说明

文件/文件夹名	说明
/faceplatform-release	SDK lib库、模型文件以及采集逻辑代码打包的aar
/faceplatform-ui	SDK的UI库，封装采集和动作活体UI等功能，以及各平台的so库。so包含以下几个平台如果关注包大小，请自行删减。【armeabi-v7a、arm64-v8a】
/app	DEMO工程（包含首页面、采集成功失败页面、设置页面等）

2.集成指南

本章将进行 Step-By-Step的讲解,如何快速的集成 人脸Sdk到现有应用中。一个完整的Demo 请参考开发包中的示例程序FacePlatform。方案架构参考下图：

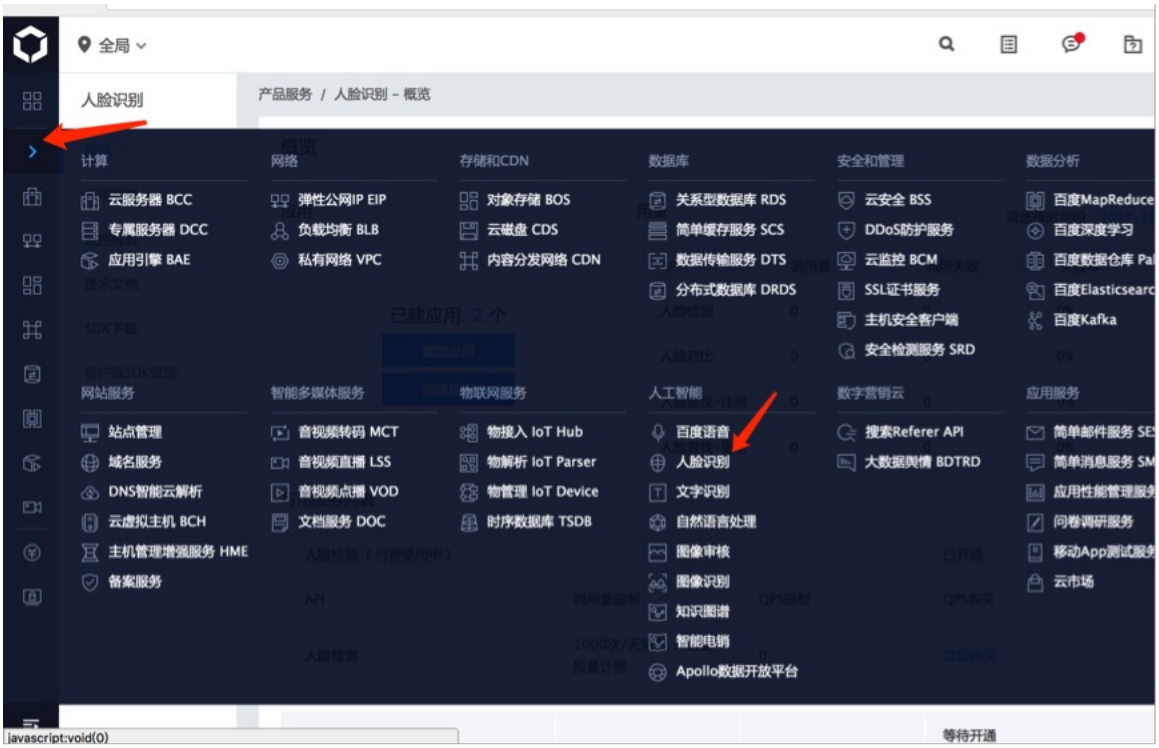


2.1 准备工作

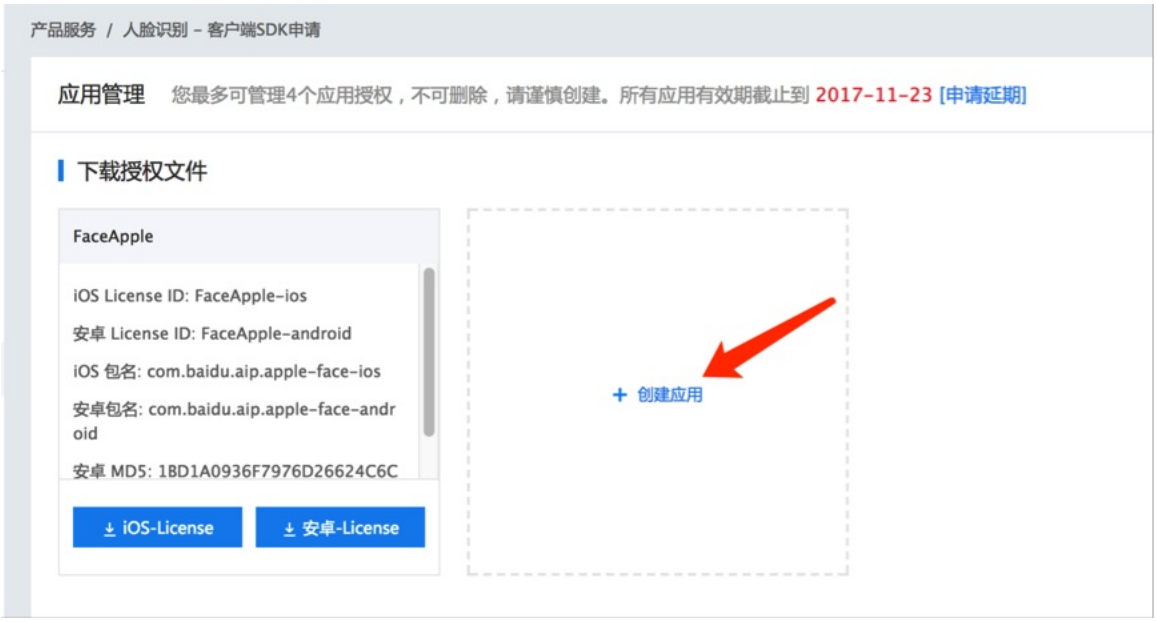
2.1.1 申请license

人脸SDK License：此license用于SDK离线功能使用，在您的申请人脸SDK的后台页面，全局->产品服务->人脸识别->客户端SDK申请

人脸控制台路径如下：

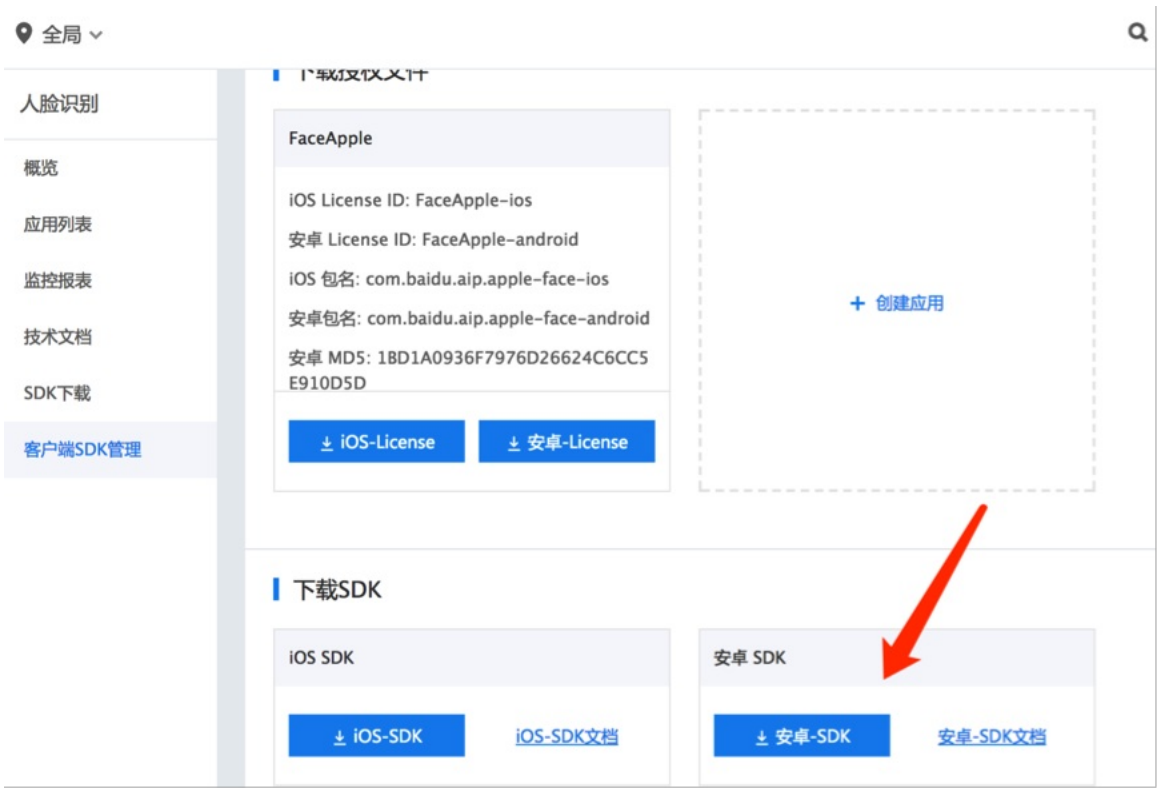


点击客户端SDK管理，弹出如下图：创建应用（这里创建应用是为了使用离线SDK，上面创建应用为了使用人脸在线接口，如注册、识别等）

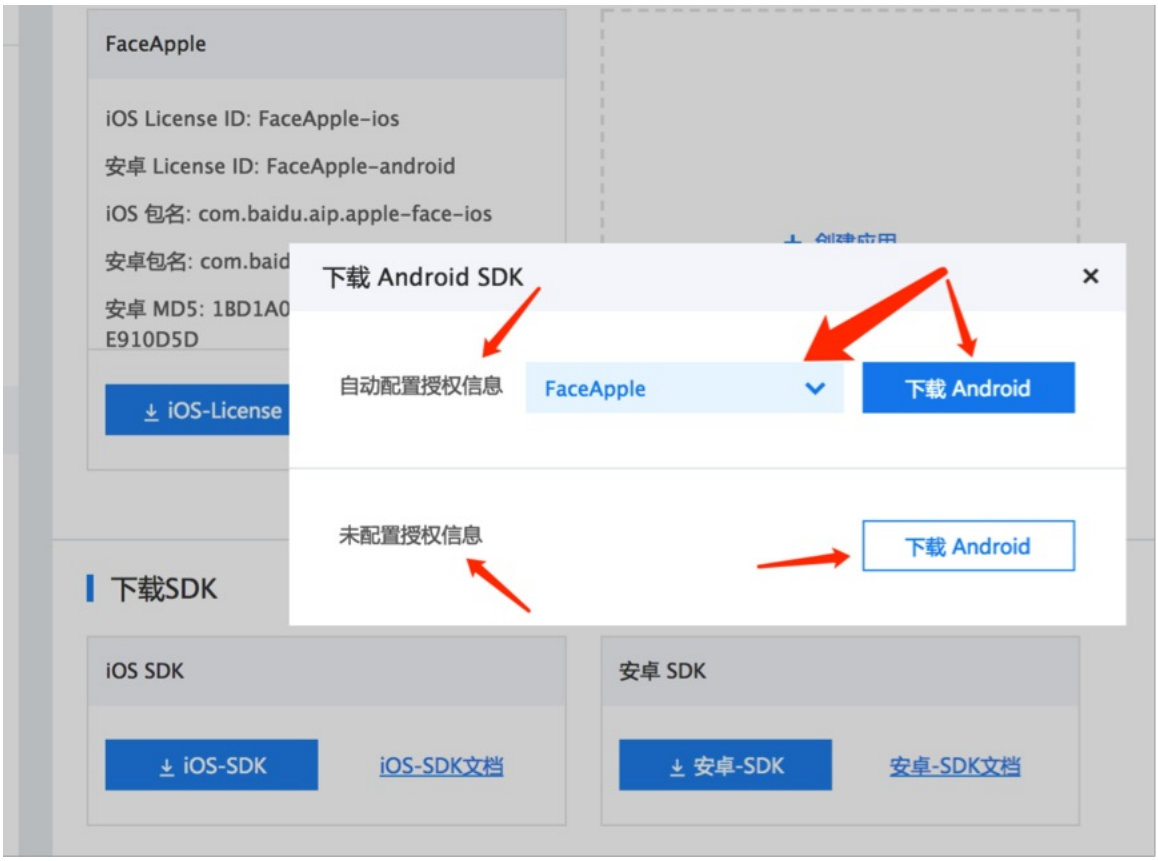


在弹出的框中输入授权标识，选择应用类型，应用系统，以及包名、MD5签名，详情请查看输入框右边提示

2.1.2 下载SDK



下载SDK分为自动配置授权信息（创建license后就可以选择为该应用，下载后SDK自动帮您配置授权，不用下载license拷贝到工程中，初始化参数licenseID,包名也帮您配置好了）和未配置授权信息两种方式：



2.2 运行示例工程

2.3.1 运行自动配置授权信息的示例工程

该下载的示例工程，已经帮你改好了license和包名

- Android Studio导入下载的示例工程
- 配置打包签名文件，由于SDK运行时会校验签名文件的MD5是否和申请时填入一样。为了便于debug能正常使用SDK，需要配置打包签名文件。
- 运行示例工程

```
    }
    signingConfigs {
        debug {
            storeFile file("[illegible]")
            storePassword '[illegible]'
            keyAlias '[illegible]'
            keyPassword '[illegible]'
        }
        release {
            storeFile file("[illegible]")
            storePassword '[illegible]'
            keyAlias '[illegible]'
            keyPassword '[illegible]'
        }
    }
    buildTypes {
```

签名文件的路径

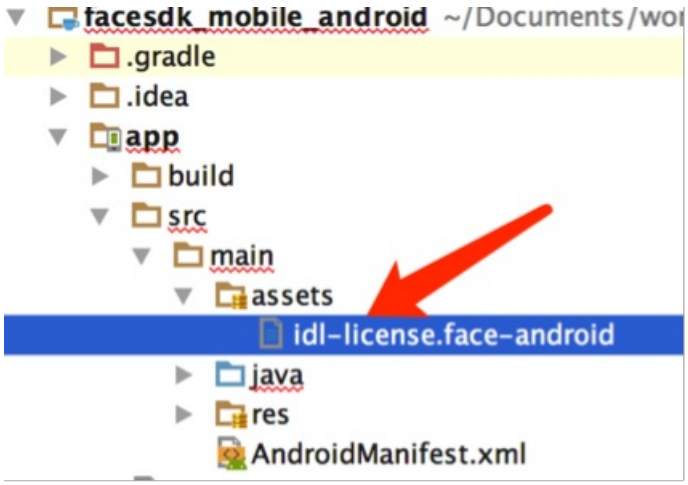
签名文件的别名

签名文件的密码

2.3.2 运行未配置授权信息的示例工程

（1）Android Studio导入下载的示例工程

（2）下载license拷贝到工程的assets目录

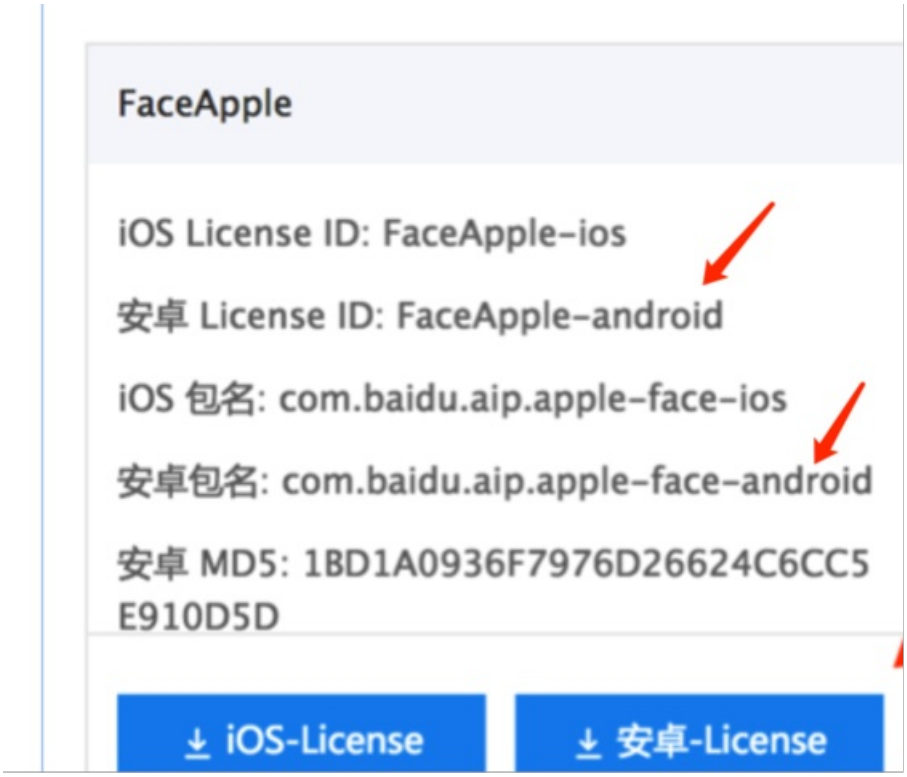


（3）修改HomeActivity.java的initialize方法参数

```
// 为了android和ios 区分授权，appId=appname_face_android ,其中appname为申请sdk时的应用名
// 应用上下文
// 申请License取得的APPID
// assets目录下License文件名
FaceSDKManager.getInstance().initialize(mContext, licenseID: "[illegible]",
        licenseFileName: "[illegible]", new IInitCallback() {
            @Override
            public void initSuccess() {
                runOnUiThread(() -> {
                        Log.e(TAG, msg: "初始化成功");
                        showToast( msg: "初始化成功");
                        mIsInitSuccess = true;
                });
            }

            @Override
            public void initFailure(final int errCode, final String errMsg) {
                runOnUiThread(() -> {
                        Log.e(TAG, msg: "初始化失败 = " + errCode + " " + errMsg);
                        showToast( msg: "初始化失败 = " + errCode + ", " + errMsg);
                        mIsInitSuccess = false;
                });
            }
        });
```

查看申请license信息，里面包含licenseId

填入放到assets目录下的授权文件名称



（4）修改app.gradle里面的包名为申请license填入的包名，如上图安卓包名。

（5）配置打包签名文件，由于SDK运行时会校验签名文件的MD5是否和申请时填入一样。为了便于debug能正常使用SDK，需要配置打包签名文件。

（6）运行示例工程。如果无法正常体验，请查看logcat日志。是否有 **FaceSDK-License LICENSE_INFO_CHECK_ERROR**日志。如果有说明授权没有成功，可以查看本文档最后的常见问题进行解决。

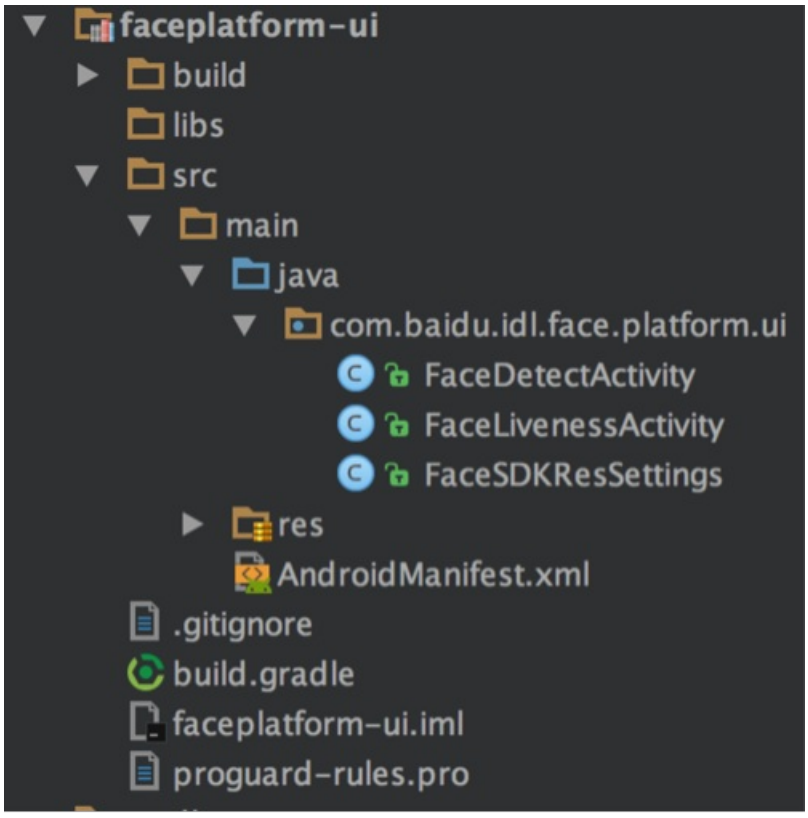
2.4 添加SDK到工程

FaceSdk以androidstudio开发方式提供，以下介绍在android studio开发工具导入FaceSdk

（1）将开发包中的faceplatform-release库Copy 到工程根目录。

（2）将开发包中的faceplatform-ui库Copy 到工程根目录。

（3）SDK提供的了开源的faceplatform-ui库，把活体检测和人脸图像采集功能等功能进行了封装，适配了主流机型机型。如果需要使用，请添加faceplatform-ui模块到的工程中。faceplatform-ui目录结构如下图



（4）在build.gradle使用compile project引人faceplatform-ui库工程。

```
dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    compile "com.android.support:recyclerview-v7:+"
    compile project(path: ':faceplatform-ui')
}
```

（5）Setting.gradle中include faceplatfrom-ui和facepaltfrom-release

```
include ':app', ':faceplatform-release'
include ':faceplatform-ui'
```

（6）从官网下载授权文件license，复制到app/src/main/assets目录下。

（7）申请的license已经和打包签名key进行了绑定（申请时用到了签名的md5，为了便于debug模式也能调用SDK的功能，需要把debug的key改成申请license的key。

- 把key拷贝到项目根目录下
- 主appbuild.gradle android 下面添加(修改)signingConfigs相关的配置。如下图。

```
    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'
        }
    }

    signingConfigs {
        debug {
            keyAlias 'facesharp'
            keyPassword [redacted]
            storeFile file('../facesharp.jks')
            storePassword([redacted])
        }
        release {
            keyAlias [redacted]
            keyPassword [redacted]
            storeFile file('../facesharp.jks')
            storePassword([redacted])
        }
    }
```

2.5 权限声明

名称	用途
需要动态申请的权限	
android.permission.READ_EXTERNAL_STORAGE	读取手机外部存储权限
android.permission.WRITE_EXTERNAL_STORAGE	写入手机外部存储权限
android.permission.CAMERA	拍照权限
不需要动态申请的权限	
android.permission.READ_PHONE_STATE	获取用户手机的 IMEI,用来唯一的标识用户
android.hardware.camera.autofocus	允许相机对焦
android.permission.INTERNET	允许访问网络
android.permission.WRITE_SETTINGS	允许修改系统设置
android.permission.WAKE_LOCK	屏幕常亮权限

2.6、混淆设置

如果工程需要做混淆处理的话，则在工程的proguard-rules.pro文件中添加如下配置：

```
-keep class com.baidu.vis.unified.license.** {*;}

-keep class com.baidu.liantian.** {*;}

-keep class com.baidu.baidusec.** {*;}

-keep class com.baidu.idl.main.facesdk.** {*;}
```

2.7、模拟器功能

需要在app下的build.gradle的android{}增加下面的话，保证模拟器能安装

```
splits {
    abi {
        enable true
        reset()
        include 'x86', 'armeabi-v7a','arm64-v8a'
        universalApk true
    }
}
```

3.功能使用

3.1 活体检测接口说明（ILivenessStrategy.java）

```
    // 活体检测策略功能接口方法
    void setLivenessStrategyConfig(List<LivenessTypeEnum> livenessList,
List<String> livenessColorList, Rect previewRect,
Rect detectRect, ILivenessStrategyCallback callback);
    // 设置是否开启语音播报
    void setLivenessStrategySoundEnable(boolean flag);
    // 用于视频录制时出现手机不兼容采用抽帧的方式
    void setIsFrameExtraction(boolean frameExtraction);
    // 开始采集（传入图像视频流，用于非录制情况）
    void livenessStrategy(byte[] imageData);
    // 开始采集（传入图片，用于录制情况，开启后会录制用户操作过程中的视频，并存于本地）
    void livenessStrategy(Bitmap bitmap);
    // 设置图片角度
    void setPreviewDegree(int degree);
    // 释放、重置
    void reset();
```

3.2 采集中关于View更新的回调（ILivenessViewCallback.java）

```
    // 获取动作活体类型，用于弹出动作的动画
    void setCurrentLiveType(LivenessTypeEnum liveType);
    // 重置当前完成动作活体个数（当前版本已不使用）
    void viewReset();
    // 设置动画停止的时机
    void animStop();
    // 用于传递faceInfo，调试画框使用
    void setFaceInfo(FaceExtInfo faceInfo);
    // 开始切换炫彩颜色（做炫彩效果时使用）
    void setBackgroundColor(int currentColor, int preColor);
    // 开始录制视频（lostFaceId为是否丢失人脸，如果丢失，则重新录制）
    void startRecordVideo(boolean lostFaceId);
```

3.3 人脸采集（包含动作活体（可关闭）、炫彩活体）

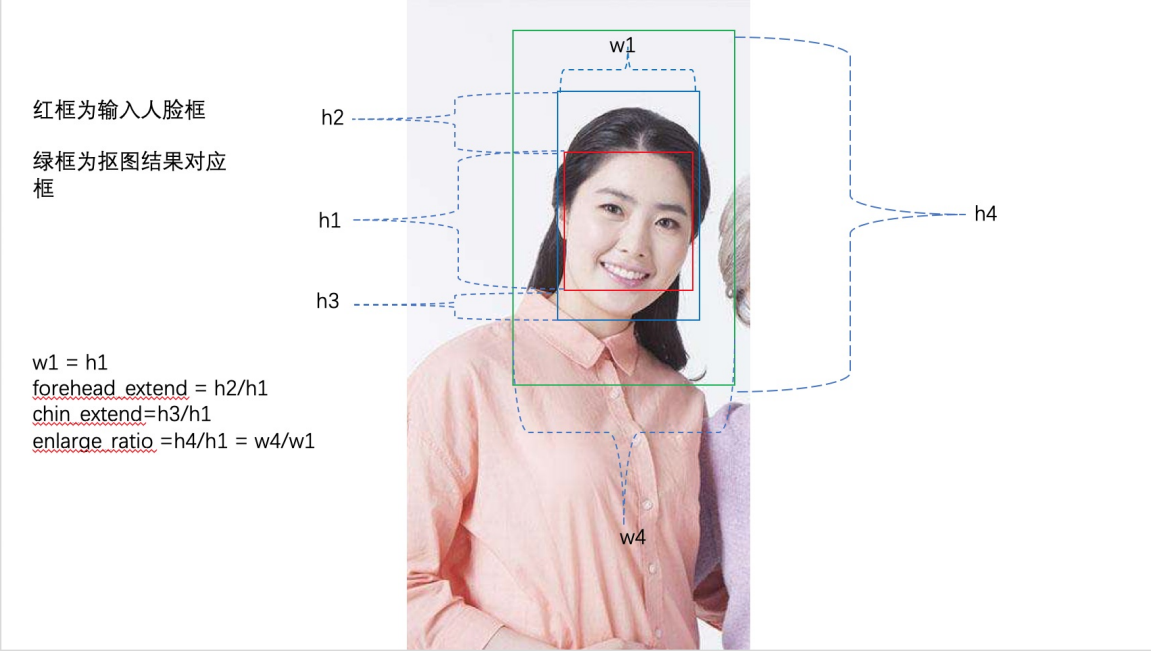
（1）初始化SDK **参数分别表示：当前上下文、鉴权key、鉴权名称、回调参数** 调用 FaceSDKManager.getIntance().initialize(Context context, String licenseID, String licenseFileName, IInitCallback callback); **Demo中此段代码在HomeActivity中。**

（2） 初始化参数设置

```
 FaceConfig config = FaceSDKManager.getInstance().getFaceConfig();
// SDK初始化已经设置完默认参数（推荐参数），也可以根据实际需求进行数值调整
// 质量等级（0：正常、1：宽松、2：严格、3：自定义）
// 获取保存的质量等级
SharedPreferencesUtil util = new SharedPreferencesUtil(mContext);
int qualityLevel = (int) util.getSharedPreference(Const.KEY_QUALITY_LEVEL_SAVE, -1);
if (qualityLevel == -1) {
    qualityLevel = ExampleApplication.qualityLevel;
}
// 根据质量等级获取相应的质量值（注：第二个参数要与质量等级的set方法参数一致）
QualityConfigManager manager = QualityConfigManager.getInstance();
manager.readQualityFile(mContext.getApplicationContext(), qualityLevel);
QualityConfig qualityConfig = manager.getConfig();
if (qualityConfig == null) {
    return false;
}
// 设置模糊度阈值
config.setBlurnessValue(qualityConfig.getBlur());
// 设置最小光照阈值（范围0-255）
config.setBrightnessValue(qualityConfig.getMinIllum());
// 设置最大光照阈值（范围0-255）
config.setBrightnessMaxValue(qualityConfig.getMaxIllum());
// 设置左眼遮挡阈值
config.setOcclusionLeftEyeValue(qualityConfig.getLeftEyeOcclusion());
// 设置右眼遮挡阈值
config.setOcclusionRightEyeValue(qualityConfig.getRightEyeOcclusion());
// 设置鼻子遮挡阈值
config.setOcclusionNoseValue(qualityConfig.getNoseOcclusion());
// 设置嘴巴遮挡阈值
config.setOcclusionMouthValue(qualityConfig.getMouseOcclusion());
// 设置左脸颊遮挡阈值
config.setOcclusionLeftContourValue(qualityConfig.getLeftContourOcclusion());
// 设置右脸颊遮挡阈值
config.setOcclusionRightContourValue(qualityConfig.getRightContourOcclusion());
// 设置下巴遮挡阈值
config.setOcclusionChinValue(qualityConfig.getChinOcclusion());
// 设置人脸姿态角阈值
config.setHeadPitchValue(qualityConfig.getPitch());
config.setHeadYawValue(qualityConfig.getYaw());
config.setHeadRollValue(qualityConfig.getRoll());
// 设置可检测的最小人脸阈值
config.setMinFaceSize(FaceEnvironment.VALUE_MIN_FACE_SIZE);
// 设置可检测到人脸的阈值
config.setNotFaceValue(FaceEnvironment.VALUE_NOT_FACE_THRESHOLD);
// 设置闭眼阈值
config.setEyeClosedValue(FaceEnvironment.VALUE_CLOSE_EYES);
// 设置图片缓存数量
config.setCacheImageNum(FaceEnvironment.VALUE_CACHE_IMAGE_NUM);
// 设置活体动作，通过设置list，LivenessTypeEunm.Eye, LivenessTypeEunm.Mouth,
// LivenessTypeEunm.HeadUp, LivenessTypeEunm.HeadDown, LivenessTypeEunm.HeadLeft,
// LivenessTypeEunm.HeadRight
config.setLivenessTypeList(ExampleApplication.livenessList);
// 设置动作活体是否随机
// config.setLivenessRandom(ExampleApplication.isLivenessRandom);
// 设置开启提示音
```

```
config.setSound(ExampleApplication.isOpenSound);
// 原图缩放系数
config.setScale(FaceEnvironment.VALUE_SCALE);
// 抠图宽高的设定，为了保证好的抠图效果，建议高宽比是4：3
config.setCropHeight(FaceEnvironment.VALUE_CROP_HEIGHT);
config.setCropWidth(FaceEnvironment.VALUE_CROP_WIDTH);
// 抠图人脸框与背景比例
config.setEnlargeRatio(FaceEnvironment.VALUE_CROP_ENLARGERATIO);
// 检测超时设置
config.setTimeDetectModule(FaceEnvironment.TIME_DETECT_MODULE);
// 检测框远近比率
config.setFaceFarRatio(FaceEnvironment.VALUE_FAR_RATIO);
config.setFaceClosedRatio(FaceEnvironment.VALUE_CLOSED_RATIO);
// 是否开启动作活体
config.setOpenActionLive(ExampleApplication.isActionLive);
// 设置动作活体颜色类型列表
config.setLivenessColorTypeList(ExampleApplication.livenessColorList);
// 设置活体阈值
config.setLivenessValue(FaceEnvironment.VALUE_LIVENESS_SCORE);
// 设置炫彩活体颜色分数阈值
config.setLivenessColorValue(FaceEnvironment.VALUE_LIVENESS_COLOR_SCORE);
// 设置视频录制时间
config.setRecordVideoTime(FaceEnvironment.TIME_RECORD_VIDEO);
// 视频录制中出现分辨率改变的手机或其它不兼容的手机统计
config.setPhoneList(ExampleApplication.phoneList);
FaceSDKManager.getInstance().setFaceConfig(config);
```

关于抠图内部实现方案如下图所示：



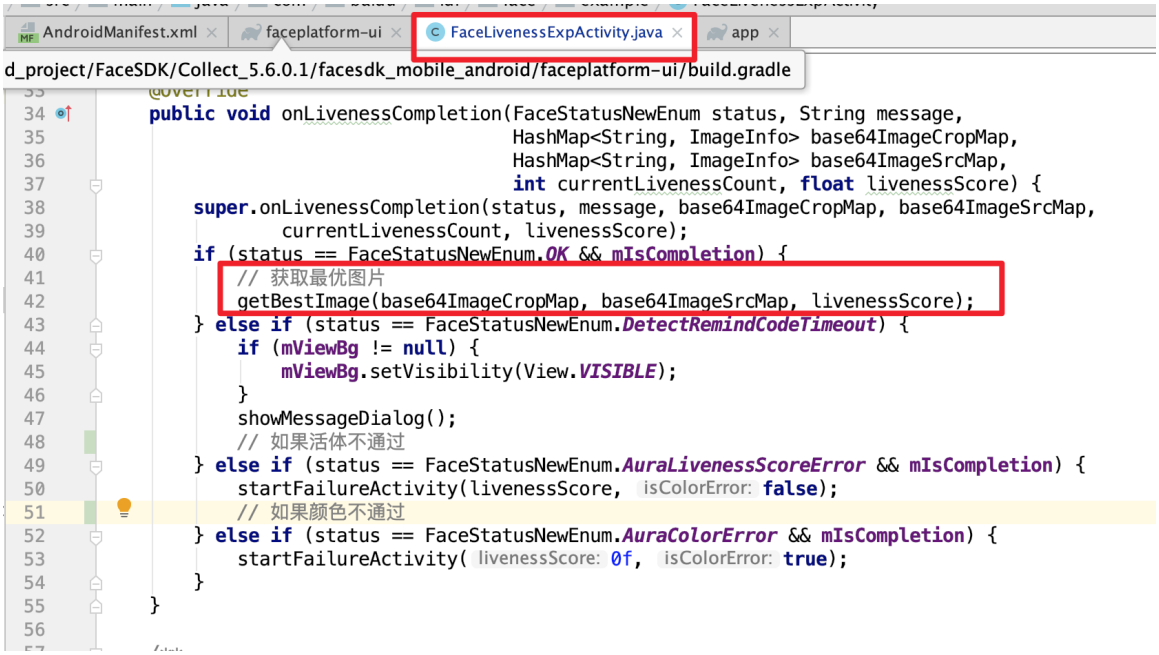
（3） startActivity(new Intent(this, FaceLivenessExpActivity.class))，开启预览。

（4）调用FaceSDKManager.getInstance().getLivenessStrategyModule()获得ILivenessStrategy对象。(该方法每次调用都会返回一个新对象).

（6）调用ILivenessStrategy.setPreviewDegree();设置预览图片的旋转角度。调用setDetectStrategySoundEnable设置是否开启语音。调用setDetectStrategyConfig设置，预览图的大小，人脸检测框的坐标和回调。

（7）多次调用livenessStrategy进行人脸图片采集，人脸跟踪、质量检测、动作活体检测、炫彩检测等。

（8）实现ILivenessStrategyCallback的onLivenessCompletion并处理结果。**其中base64ImageCropMap为存放人脸抠图图片集，base64ImageSrcMap为存放人脸原图图片集**并通过调用getBestImage()方法，通过从优到差的质量排序，获取最优的bitmap，代码如下图所示：




```
        @Override
        public void onLivenessCompletion(FaceStatusNewEnum status, String message,
                                         HashMap<String, ImageInfo> base64ImageCropMap,
                                         HashMap<String, ImageInfo> base64ImageSrcMap,
                                         int currentLivenessCount, float livenessScore) {
            super.onLivenessCompletion(status, message, base64ImageCropMap, base64ImageSrcMap,
                    currentLivenessCount, livenessScore);
            if (status == FaceStatusNewEnum.OK && mIsCompletion) {
                // 获取最优图片
                getBestImage(base64ImageCropMap, base64ImageSrcMap, livenessScore);
            } else if (status == FaceStatusNewEnum.DetectRemindCodeTimeout) {
                if (mViewBg != null) {
                    mViewBg.setVisibility(View.VISIBLE);
                }
                showMessageDialog();
                // 如果活体不通过
            } else if (status == FaceStatusNewEnum.AuraLivenessScoreError && mIsCompletion) {
                startFailureActivity(livenessScore, isColorError: false);
                // 如果颜色不通过
            } else if (status == FaceStatusNewEnum.AuraColorError && mIsCompletion) {
                startFailureActivity( livenessScore: 0f,  isColorError: true);
            }
        }
```

```
    /**
     * 获取最优图片
     * @param imageCropMap 抠图集合
     * @param imageSrcMap  原图集合
     */
    private void getBestImage(HashMap<String, ImageInfo> imageCropMap, HashMap<String, ImageInfo> imageSrcMap) {
        String bmpStr = null;
        // 将抠图集合中的图片按照质量降序排序，最终选取质量最优的一张抠图图片
        if (imageCropMap != null && imageCropMap.size() > 0) {
            List<Map.Entry<String, ImageInfo>> list1 = new ArrayList<>(imageCropMap.entrySet());
            Collections.sort(list1, new Comparator<Map.Entry<String, ImageInfo>>() {

                @Override
                public int compare(Map.Entry<String, ImageInfo> o1,
                                   Map.Entry<String, ImageInfo> o2) {
                    String[] key1 = o1.getKey().split("_");
                    String score1 = key1[2];
                    String[] key2 = o2.getKey().split("_");
                    String score2 = key2[2];
                    // 降序排序
                    return Float.valueOf(score2).compareTo(Float.valueOf(score1));
                }
            });

            // 获取抠图中的加密或非加密的base64
//            int secType = mFaceConfig.getSecType();
//            String base64;
//            if (secType == 0) {
//                base64 = list1.get(0).getValue().getBase64();
//            } else {
//                base64 = list1.get(0).getValue().getSecBase64();
//            }
        }
```

```
        // 将原图集合中的图片按照质量降序排序，最终选取质量最优的一张原图图片
        if (imageSrcMap != null && imageSrcMap.size() > 0) {
            List<Map.Entry<String, ImageInfo>> list2 = new ArrayList<>(imageSrcMap.entrySet());
            Collections.sort(list2, new Comparator<Map.Entry<String, ImageInfo>>() {

                @Override
                public int compare(Map.Entry<String, ImageInfo> o1,
                                   Map.Entry<String, ImageInfo> o2) {
                    String[] key1 = o1.getKey().split("_");
                    String score1 = key1[2];
                    String[] key2 = o2.getKey().split("_");
                    String score2 = key2[2];
                    // 降序排序
                    return Float.valueOf(score2).compareTo(Float.valueOf(score1));
                }
            });
            bmpStr = list2.get(0).getValue().getBase64();

            // 获取原图中的加密或非加密的base64
//            int secType = mFaceConfig.getSecType();
//            String base64;
//            if (secType == 0) {
//                base64 = mBmpStr;
//            } else {
//                base64 = list2.get(0).getValue().getBase64();
//            }
        }

        // 页面跳转
        IntentUtils.getInstance().setBitmap(bmpStr);
        Intent intent = new Intent(FaceLivenessExpActivity.this,
                CollectionSuccessActivity.class);
        intent.putExtra("destroyType", "FaceLivenessExpActivity");
        startActivity(intent);
    }
```

另外，livenessScore表示炫彩活体的当前分数，用户可以看到自己的炫彩活体分数。

（9）Demo中的最优抠图或者最优原图，可以调用SecRequest类下的sendMessage(Context context, String secBase64, int secType);将加密后的base64发送到服务端，如需风控，需要把SecRequest.java下的risk_identify参数置为true。

3.4 带视频录制或图片抽帧的人脸采集功能 （1）如果想要测试视频录制功能，则需要在**ExampleApplication.java**中的**ublic static boolean isOpenVideoRecord = false**修改为true。 （2）在录制视频过程中，有可能会出现部分手机不兼容的情况，例如打不开摄像头，那么需要在**HomeActivity.java**中加入白名单处理，如下：

```
private void addVideoPhone() {
    // 统计视频录制中出现不兼容情况的手机
    ExampleApplication.phoneList.clear();
    ExampleApplication.phoneList.add("OPPO_PBET00");            // OPPO R17
    ExampleApplication.phoneList.add("HONOR_EBG-AN00");         // 荣耀30 PRO
    ExampleApplication.phoneList.add("Lenovo_Lenovo L38041");   // 联想K5 PRO
    ExampleApplication.phoneList.add("vivo_V1963A");            // vivo Z6
    ExampleApplication.phoneList.add("HUAWEI_SEA-AL00");        // 华为 nova5
    ExampleApplication.phoneList.add("OPPO_PBCM30");            // OPPO K1
    ExampleApplication.phoneList.add("OPPO_OPPO R11s");         // OPPO R11s
}
```

加入白名单后，则会采用图片抽帧的方式留存。

视频录制的路径为：**/data/data/包名/file/vrecord/XXX.mp4** 抽帧图片的路径为：**/data/data/包名/file/image/XXX.jpg**

（2）初始化SDK **参数分别表示：当前上下文、鉴权key、鉴权名称、回调参数** 调用FaceSDKManager.getIntance().initialize(Context context, String licenseID, String licenseFileName, IInitCallback callback); **Demo中此段代码在HomeActivity中。**

（3）初始化参数设置（参照3.2（2））

（4） startActivity(new Intent(this, FaceLivenessVideoExpActivity.class))，开启预览，开启录制。（如果采用选帧策略，则是跳转到FaceLivenessExpActivity中）

（4）调用FaceSDKManager.getInstance().getLivenessStrategyModule()获得ILivenessStrategy对象。(该方法每次调用都会返回一个新对象).

（5）调用ILivenessStrategy.setPreviewDegree();设置预览图片的旋转角度。调用setDetectStrategySoundEnable设置是否开启

语音。调用setDetectStrategyConfig设置，预览图的大小，人脸检测框的坐标和回调。

（6）如果采用图片选帧留存方式，则需要调用void setIsFrameExtraction(boolean frameExtraction);方法，传入true

（7）多次调用livenessStrategy进行人脸图片采集，人脸跟踪、质量检测、动作活体检测、炫彩检测等。

（8）实现ILivenessStrategyCallback的onLivenessCompletion并处理结果。**其中base64ImageCropMap为存放人脸抠图图片集，base64ImageSrcMap为存放人脸原图图片集**并通过调用getBestImage()方法，通过从优到差的质量排序，获取最优的bitmap，代码如下图所示：

```
@Override
public void onLivenessCompletion(final FaceStatusNewEnum status, String message,
                                 final HashMap<String, ImageInfo> base64ImageCropMap,
                                 final HashMap<String, ImageInfo> base64ImageSrcMap,
                                 int currentLivenessCount, final float livenessScore) {
    super.onLivenessCompletion(status, message, base64ImageCropMap, base64ImageSrcMap,
            currentLivenessCount, livenessScore);
    runOnUiThread(() -> {
            if (status == FaceStatusNewEnum.OK && mIsCompletion) {
                // 获取最优图片
                getBestImage(base64ImageCropMap, base64ImageSrcMap, livenessScore);
            } else if (status == FaceStatusNewEnum.DetectRemindCodeTimeout) {
                if (mViewBg != null) {
                    mViewBg.setVisibility(View.VISIBLE);
                }
                showMessageDialog();
            } else if (status == FaceStatusNewEnum.AuraLivenessScoreError && mIsCompletion) {
                startFailureActivity(livenessScore,  isColorError: false);
            } else if (status == FaceStatusNewEnum.AuraColorError && mIsCompletion) {
                startFailureActivity( livenessScore: 0f,  isColorError: true);
            }
    });
}
```

```
/**
 * 获取最优图片
 * @param imageCropMap 抠图集合
 * @param imageSrcMap  原图集合
 */
private void getBestImage(HashMap<String, ImageInfo> imageCropMap, HashMap<String, ImageInfo> imageSrcMap) {
    String bmpStr = null;
    // 将抠图集合中的图片按照质量降序排序，最终选取质量最优的一张抠图图片
    if (imageCropMap != null && imageCropMap.size() > 0) {
        List<Map.Entry<String, ImageInfo>> list1 = new ArrayList<>(imageCropMap.entrySet());
        Collections.sort(list1, new Comparator<Map.Entry<String, ImageInfo>>() {

            @Override
            public int compare(Map.Entry<String, ImageInfo> o1,
                               Map.Entry<String, ImageInfo> o2) {
                String[] key1 = o1.getKey().split("_");
                String score1 = key1[2];
                String[] key2 = o2.getKey().split("_");
                String score2 = key2[2];
                // 降序排序
                return Float.valueOf(score2).compareTo(Float.valueOf(score1));
            }
        });

        // 获取抠图中的加密或非加密的base64
//          int secType = mFaceConfig.getSecType();
//          String base64;
//          if (secType == 0) {
//              base64 = list1.get(0).getValue().getBase64();
//          } else {
//              base64 = list1.get(0).getValue().getSecBase64();
//          }
    }
```

```
        // 将原图集合中的图片按照质量降序排序，最终选取质量最优的一张原图图片
        if (imageSrcMap != null && imageSrcMap.size() > 0) {
            List<Map.Entry<String, ImageInfo>> list2 = new ArrayList<>(imageSrcMap.entrySet());
            Collections.sort(list2, new Comparator<Map.Entry<String, ImageInfo>>() {

                @Override
                public int compare(Map.Entry<String, ImageInfo> o1,
                                   Map.Entry<String, ImageInfo> o2) {
                    String[] key1 = o1.getKey().split("_");
                    String score1 = key1[2];
                    String[] key2 = o2.getKey().split("_");
                    String score2 = key2[2];
                    // 降序排序
                    return Float.valueOf(score2).compareTo(Float.valueOf(score1));
                }
            });
            bmpStr = list2.get(0).getValue().getBase64();
            // 获取原图中的加密或非加密的base64
//            int secType = mFaceConfig.getSecType();
//            String base64;
//            if (secType == 0) {
//                base64 = mBmpStr;
//            } else {
//                base64 = list2.get(0).getValue().getBase64();
//            }
        }

        // 页面跳转
        IntentUtils.getInstance().setBitmap(bmpStr);
        Intent intent = new Intent(FaceLivenessExpActivity.this,
                CollectionSuccessActivity.class);
        intent.putExtra("destroyType", "FaceLivenessExpActivity");
        startActivity(intent);
    }
```

另外，livenessScore表示炫彩活体的当前分数，用户可以看到自己的炫彩活体分数。

（9）Demo中的最优抠图或者最优原图，可以调用SecRequest类下的sendMessage(Context context, String secBase64, int secType);将加密后的base64发送到服务端，如需风控，需要把SecRequest.java下的risk_identify参数置为true。

3.5 质量校验设置

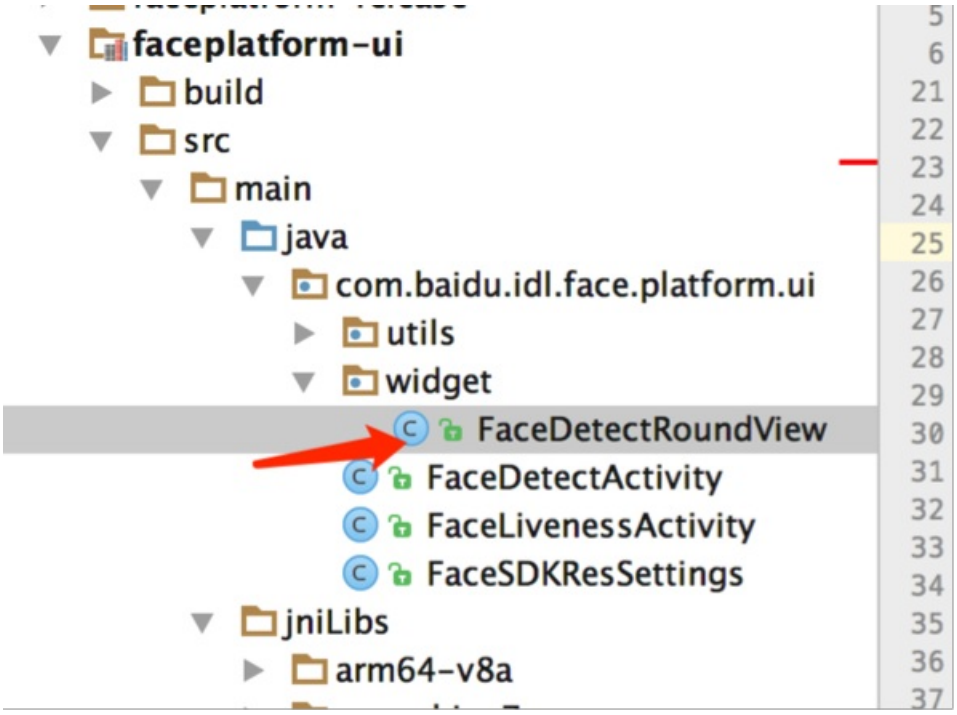
com.baidu.idl.face.platform.FaceConfig类用于人脸检测参数设置。

参数	名称	默认值	取值范围
minFaceSize	最小人脸阈值	200	
notFaceValue	非人脸阈值	0.6f	0~1.0f
brightnessValue	图片最小光照阈值	宽松：30、正常：40、严格：60	0-255f
brightnessValue	图片最大光照阈值	宽松：240、正常：220、严格：200	0-255f
blurnessValue	图像模糊阈值	宽松：0.8f、正常：0.6f、严格：0.4f	0~1.0f
occlusionLeftEyeValue	左眼遮挡阈值	宽松：0.95f、正常：0.8f、严格：0.4f	0~1.0f
occlusionRightEyeValue	右眼遮挡阈值	宽松：0.95f、正常：0.8f、严格：0.4f	0~1.0f
occlusionNoseValue	鼻子遮挡阈值	宽松：0.95f、正常：0.8f、严格：0.4f	0~1.0f
occlusionMouthValue	嘴巴遮挡阈值	宽松：0.95f、正常：0.8f、严格：0.4f	0~1.0f
occlusionLeftContourValue	左脸颊遮挡阈值	宽松：0.95f、正常：0.8f、严格：0.4f	0~1.0f
occlusionRightContourValue	右脸颊遮挡阈值	宽松：0.95f、正常：0.8f、严格：0.4f	0~1.0f
occlusionChinValue	下巴遮挡阈值	宽松：0.95f、正常：0.8f、严格：0.4f	0~1.0f
headPitchValue	低头抬头角度	宽松：30、正常：20、严格：15	0~45
headYawValue	左右摇头角度	宽松：18、正常：18、严格：15	0~45
headRollValue	偏头角度	宽松：30、正常：20、严格：15	0~45
eyeClosedValue	闭眼阈值	0.7f	0~1.0f
cacheImageNum	图片缓存数量	3	建议3~6
livenessValue	活体分数阈值	0.9f	0~1.0f

3.6 界面定制说明

3.6.1 修改faceplatform_ui界面

（1）如果SDK自带的UI和您的APP不统一，您可以自行修改FaceDetectRoundView。



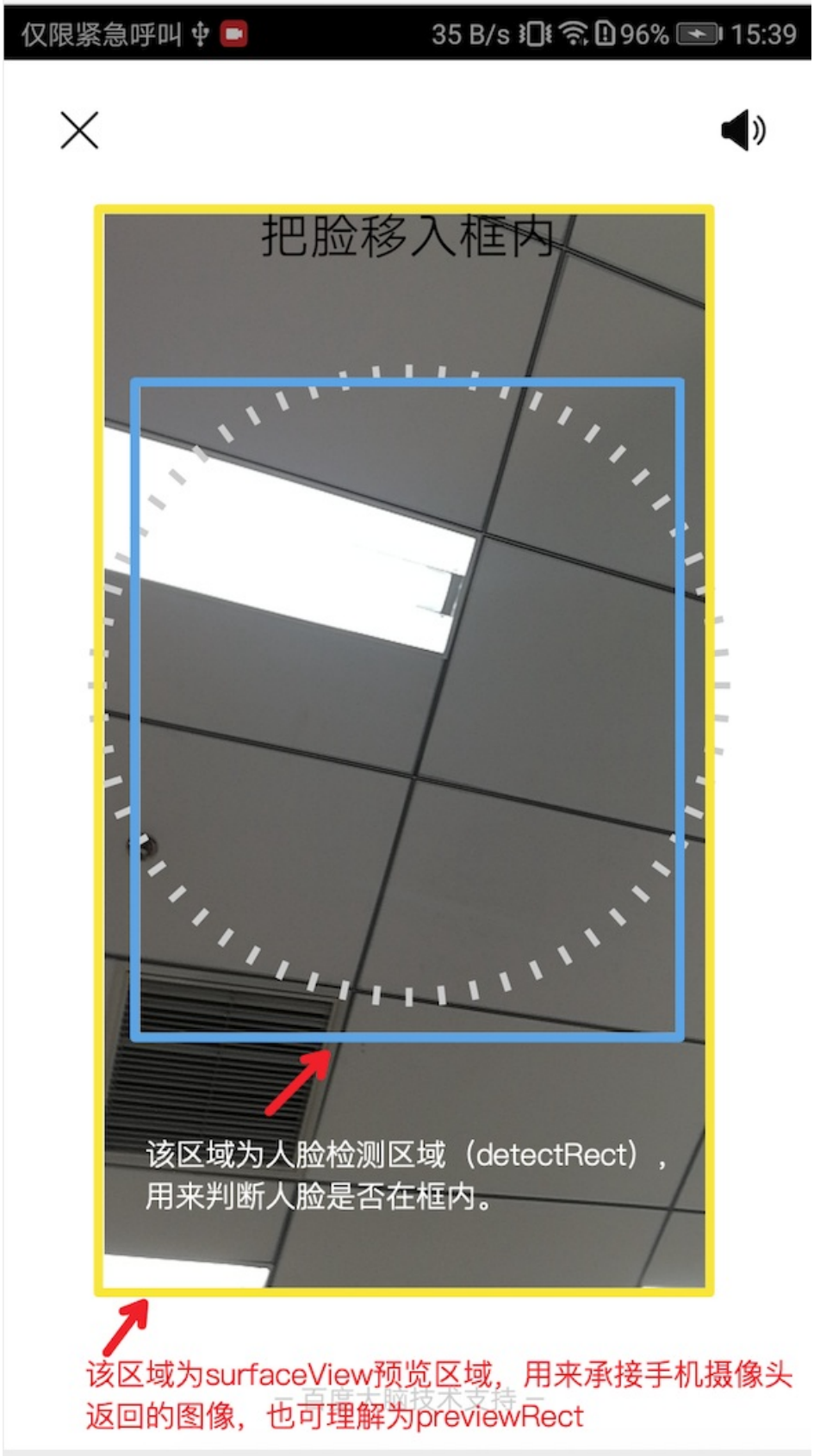
（2）修改提示语音音频文件，有两种方式。

a、直接替换FaceUI工程raw下的mp3文件和string.xml。

b、FaceEnvironment 提供了setSoundId(FaceStatusNewEnum status, int soundId); 设置提示音资源。 FaceStatusNewEnum为不同的状态。soundId为资源文件所对应的resource id。

3.7 采集中的预览区域和检测区域说明

如下图所示（图中背景设置为透明了，可以看出这两个区域）：



其中： surfaceView宽高为屏幕的宽高； previewRect的宽高为手机摄像头的宽高； detectRect的宽高是以previewRect为基准计算出来的。 计算方式参照FaceDetectRoundView.java的Rect getPreviewDetectRect(int w, int pw, int ph)方法

4.接口设计说明

4.1 人脸功能管理器

4.1.1创建实例

- 方法

FaceSDKManager getInstance()

- 参数

无

- 返回

人脸功能管理器

- 说明

> 创建人脸功能管理器

4.1.2 人脸功能管理器初始化

- 方法

```
public void initialize(final Context context, String licenseID, String licenseFileName, IInitCallback callback)
```

- 参数

context 上下文环境，licenseID 传入申请License时获取的应用名称+_face_android后缀，licenseFileName 鉴权文件名称，callback 鉴权成功与否回调

- 返回

无

- 说明

初始化人脸检测功能。进行人脸检测功能License鉴权验证。

4.1.3 设置人脸功能控制参数

- 方法

```
void setFaceConfig(FaceConfig config)
```

- 参数

config 人脸功能控制参数对象

- 返回

无

- 说明

设置人脸功能控制参数对象。

FaceConfig对象参数：

最小光照阈值

最大光照阈值

模糊阈值

遮挡阈值

头部姿态角度阈值

最小人脸检测阈值

非人脸阈值

人脸抠图宽高设置

进行活体检测的动作类型列表

超时时间设置

检测框远近比率设置等

4.1.4 取得人脸图像采集功能接口

- 方法

```
IDetectStrategy getDetectStrategyModule()
```

- 参数

无

- 返回

人脸图像采集功能接口

- 说明

取得人脸图像采集功能接口。人脸图像采集接口完成，解析图片人脸信息，返回检测结果。

4.1.5 取得活体检测功能接口

- 方法

```
ILivenessStrategy getLivenessStrategyModule()
```

- 参数

无

- 返回

活体检测功能接口

- 说明

取得活体检测功能接口。活体检测功能接口完成，解析图片人脸信息，返回活体检测结果。

4.1.6 内存释放接口

- 方法

```
void release()
```

- 参数

无

- 返回

无

- 说明

主要针对模型的释放，以减少内存

4.2 人脸图像采集器

人脸图像采集器IDetectStrategy，检测图片中人脸信息，返回人脸检测状态，完成人脸图像采集。

4.2.1 设置人脸图像采集功能参数

- 方法

```
void setDetectStrategyConfig(Rect previewRect, Rect detectRect,IDetectStrategyCallback callback)
```

- 参数

previewRect 人脸图片大小，类型：Rect

detectRect 人脸检测区域大小，类型：Rect

callback 人脸图像采集功能状态监听器

- 返回

无

- 说明

设置人脸功能控制参数对象。

4.2.2 人脸图像采集

- 方法

```
void detectStrategy(byte[] imageData)
```

- 参数

imageData 图片信息

- 返回

无

- 说明

检测图片中的人脸信息，完成人脸图像采集，返回检测状态和结果。

4.3 活体检测器

活体检测器LivenessStrategy，检测图片人脸信息，活体检测结果状态。

4.3.1 设置人脸功能控制参数

- 方法

```
voidsetLivenessStrategyConfig(
        List<LivenessTypeEnum> livenessList,
        Rect previewRect,
        Rect detectRect,
        ILivenessStrategyCallback callback);
```

- 参数

livenessList 活体动作列表

previewRect 人脸图片大小，类型：Rect

detectRect 人脸检测区域大小，类型：Rect

callback 人脸图像采集功能状态监听器

- 返回

无

- 说明

设置活体检测功能参数对象。

4.3.2 活体检测

- 方法

void livenessStrategy(byte[] imageData);

- 参数

imageData 图片信息

- 返回

无

- 说明

检测图片中的人脸信息，完成人脸图像采集和人脸活体检测，返回检测状态和结果。

4.4 人脸图像采集界面

人脸图像采集器界面FaceDetectActivity，包括UI界面，系统相机控制，使用人脸图像采集器IDetectStrategy处理相机采集到的图像。

4.5 活体检测界面

活体检测界面FaceLivenessActivity，包括UI界面，系统相机控制，使用活体检测器ILivenessStrategy处理相机采集到的图像，完成活体检测功能。

4.6 图片加密请求类 图片加密请求类SecRequest，里面的sendMessage方法是为了测试图片加密之后secBase64是否可以通过云端解密的一个示例代码。 **4.7 ZID参数获取** 调用金融级实名认证接口配合金融级SDK使用时，需传入zid参数打开大数据风控

功能。

获取方法：

```
FaceSDKManager.getInstance().getZid(Context)
```

5.常见问题

（1）**license**文件有什么用，该放在什么地方？

license文件需要申请，目的是作为sdk校验开发者的使用合法性，license文件放置位置不对或未放置license文件会导致没法使用sdk，一般应先申请license文件，并把申请得到的license文件，放置在assets目录下面。

（2）鉴权初始化的错误码对应的信息

ErrorCode	常量值	说明
SUCCESS	0	成功
LICENSE_NOT_INIT_ERROR	1	license未初始化
LICENSE_DECRYPT_ERROR	2	license数据解密失败
LICENSE_INFO_FORMAT_ERROR	3	license数据格式错误
LICENSE_KEY_CHECK_ERROR	4	license-key(api-key)校验错误
LICENSE_ALGORITHM_CHECK_ERROR	5	算法ID校验错误
LICENSE_MD5_CHECK_ERROR	6	MD5校验错误
LICENSE_DEVICE_ID_CHECK_ERROR	7	设备ID校验错误
LICENSE_PACKAGE_NAME_CHECK_ERROR	8	包名(应用名)校验错误
LICENSE_EXPIRED_TIME_CHECK_ERROR	9	过期时间不正确
LICENSE_FUNCTION_CHECK_ERROR	10	功能未授权
LICENSE_TIME_EXPIRED	11	授权已过期
LICENSE_LOCAL_FILE_ERROR	12	本地文件读取失败
LICENSE_REMOTE_DATA_ERROR	13	远程数据拉取失败
LICENSE_LOCAL_TIME_ERROR	14	本地时间校验错误
OTHER_ERROR	0xff	其他错误

（3）**FaceSDK-LicenseLICENSE_INFO_CHECK_ERROR =license id**

licenseID为您申请时填appname+“_face_android”。如下图demo-turnstile-face-android为license里面的licenseID, **demo-turnstile-face-android1**为app运行时Config.licenseID，两者必须一致

```
E/FaceSDK: FaceSDK-License LICENSE_INFO_CHECK_ERROR =license id  demo-turnstile-face-android  demo-turnstile-face-android1
```

（4）**FaceSDK-LicenseLICENSE_INFO_CHECK_ERROR =signature md5**

md5不一致错误，签名的为license里面的md5，后面的为app运行时获取的签名文件的md5，这两个md5必须一致且区分大小写。 `E/FaceSDK: FaceSDK-License LICENSE_INFO_CHECK_ERROR =signature md5  F5846C60804CC6042D55D09F1AB82364  4357A3EDBC0CA02EA8B5E0578E58D`

（5）**FaceSDK-License LICENSE_INFO_CHECK_ERROR =package name**

PackageName不一致错误。License里面的packagename为申请license时填的，需要保证和app里面的packagename一致。

（6）活体检测常见有那些动作？是否可配置？

常见有6个动作，眨眼，张大嘴，向上抬头，向下低头，向左摇头，向右摇头等。 sdk提供FaceConfig参数设置类，如活体检测

角度、光线，检测动作，检测动作数量等设置。

（7）使用sdk一般会用到活体检测拍照等功能，有什么需要注意？

Android 6.0+，需要注意相机拍摄权限问题。如没申请权限，可能导致没法调起相机。

（8）在有些机型上出现特别卡或出现无响应？

SDK在**armeabi**上性能非常差，建议删掉其他so只留下armeabi-v7a，包括使用的其他第三方so。因为如果其他so有armeabi，根据android系统查找so的逻辑，在armeabi的机型上只会去该目录下查找so，而人脸SDK没有，就会出现找不到so。

（9）**license** 文件失效了，不能用了怎么办？

license文件申请时候有期限，如过期会导致校验失效，需要在后台申请延期。

IOS-1.0

1.简介

1.1 功能介绍

百度Face SDK IOS 版是一种面向 IOS 移动设备人脸技术开发包，此版SDK包含人脸检测、活体识别等功能。基于该方案，开发者可以轻松的构建包含人脸检测、采集和活体识别的应用。在您使用SDK之前，我们首先为您介绍以下SDK本身及相关人脸能力，以便您能更方便使用人脸服务。

此版SDK所包含的能力如下：

- **离线炫瞳活体检测**：通过屏幕上闪烁不同颜色的光线，判断当前用户是否真人操作。通过颜色活体进行面部反光鉴别的同时，百度特加入独有的瞳孔反光识别，提升整体的攻击拒绝率指标，可有效抵御高清图片、3D建模、视频等攻击。
- **端云互验加密：**在采集SDK端加入端云互验加密功能，在采集SDK端对数据进行加密，云端进行解密，从而可以有效识别第三方非法黑产通过绕过APP用脚本攻击接口的行为发生。
- **大数据风控：**与百度安全实验室联合推出大数据风控功能，与云端接口配合对设备端进行环境校验，可有效识别ROM注入、视频劫持、云手机等行为。
- **离线人脸质量检测**：判断视频流中的图片帧中，哪些图片质量较佳，即人脸图像特征清晰（满足姿态角、光照、模糊度、遮挡等校验）。
- **离线人脸图像采集**：通过本地SDK能力，采集人脸图像，同时经过人脸质量检测，确保采集到的人脸图像符合各条件校验（满足姿态角、光照、模糊度、遮挡等校验），为设备前端获取有效可分析人脸的主要功能。
- **离线授权**：SDK的授权判断，授权介质也称为license，在SDK使用中，优先验证本地离线鉴权文件，验证通过可离线使用；在本地鉴权失败情况下需要向授权服务器发起请求，远程拉取授权进行验证。

为了方便您的开发，我们已经为您准备了多种场景的示例工程，您可以根据业务需要，在后台进行直接下载，目前支持【人脸核身】【人脸闸机/门禁】【人脸登录/考勤】【多人脸识别】,示例工程参考下图：

1.2 兼容性

- **系统**：支持iOS9以上系统。需要开发者通过Deployment Target 来保证支持系统的检测。
- **机型**：手机和平板皆可（暂不支持横屏）
- **架构**：arm64、armv7
- **网络**：支持 WIFI 及移动网络,移动网络支持使用 NET 网关及WAP 网关(CMWAP、CTWAP、UNIWAP、3GWAP)。

1.3 开发包说明

文件/文件夹名	说明
FaceSDK	FaceSDK 包、bundle 资源文件、bundle 模型文件、鉴权文件
Public/Common	视频流处理类
Public/Utils	图像处理类
UI/Controller	DEMO工程
UI/View	View控制类
FaceParameterConfig.h	配置信息

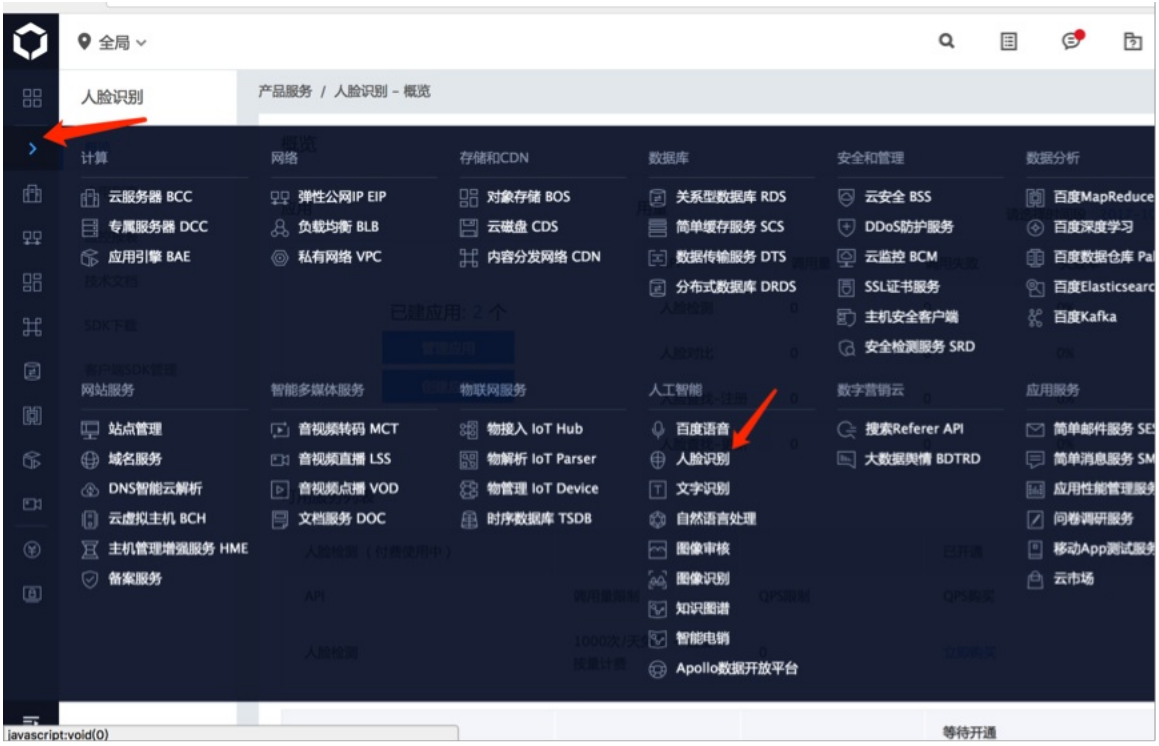
2.集成指南

2.1 准备工作

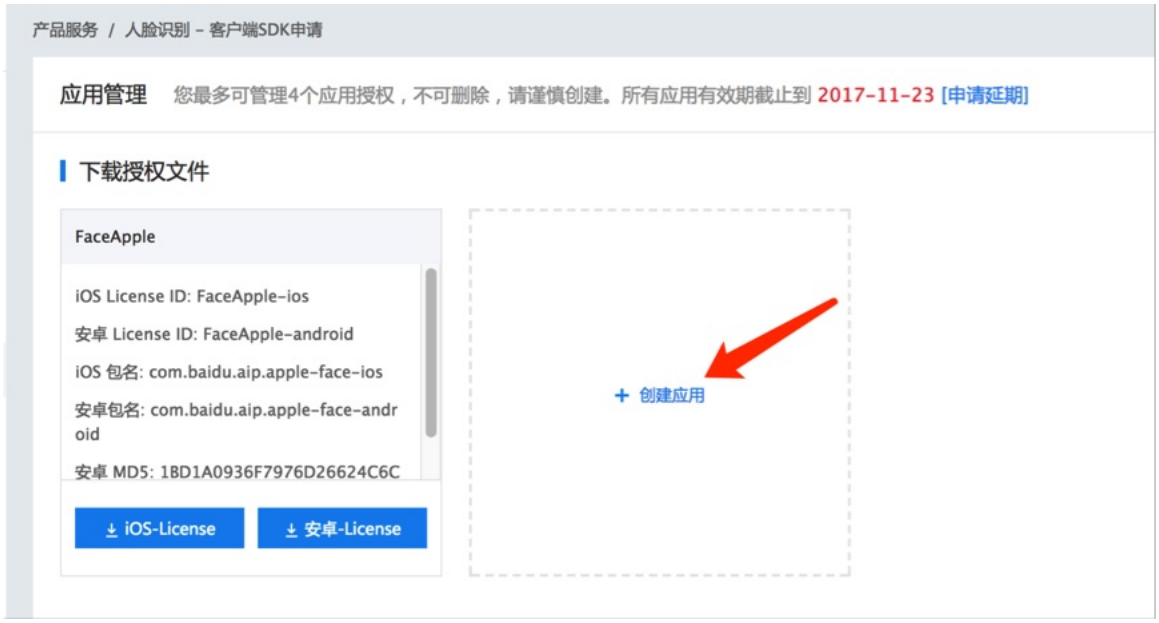
2.1.1 申请license

人脸SDK License：此license用于SDK离线功能使用，在您的申请人脸SDK的后台页面，全局->产品服务->人脸识别->客户端SDK申请

人脸控制台路径如下：



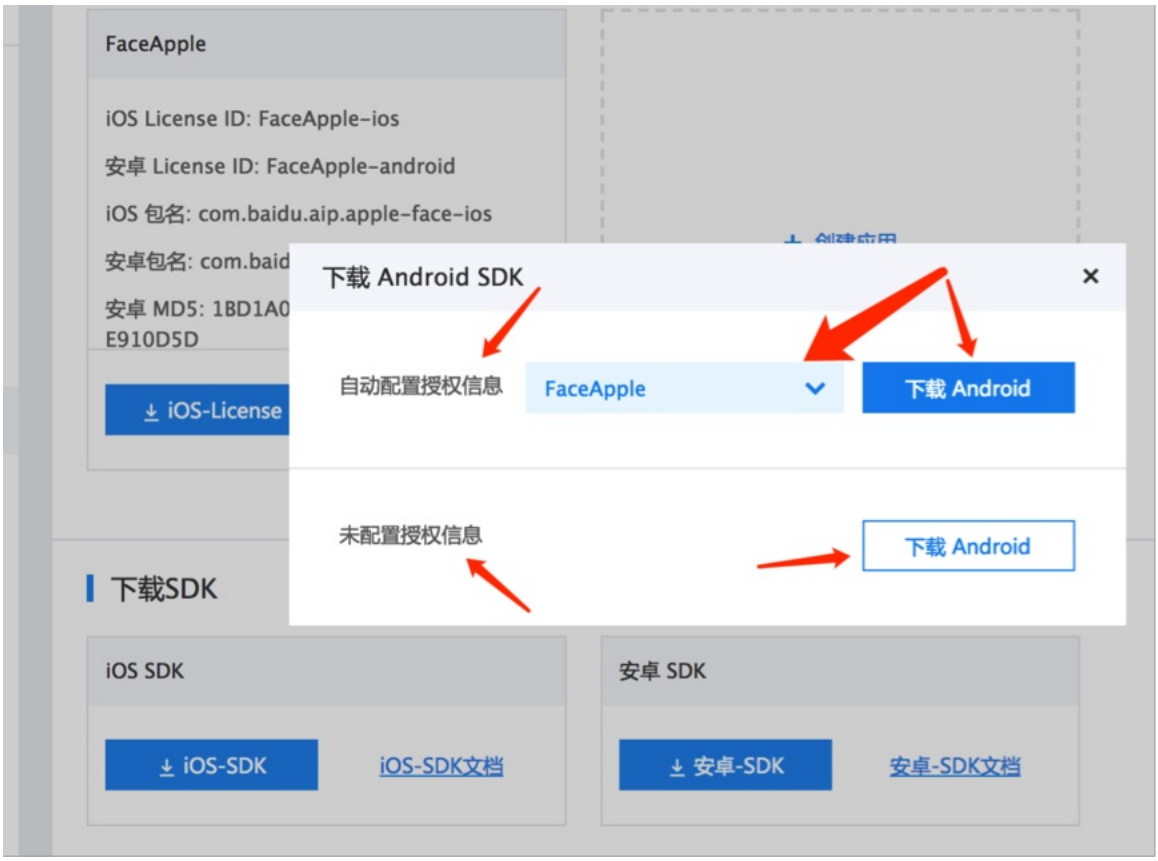
点击客户端SDK管理，弹出如下图：创建应用（这里创建应用是为了使用离线SDK，上面创建应用为了使用人脸在线接口，如注册、识别等）



在弹出的框中输入授权标识，选择应用类型，应用系统，以及包名，详情请查看输入框右边提示

2.1.2 下载SDK

下载SDK分为自动配置授权信息（创建license后就可以选择为该应用，下载后SDK自动帮您配置授权，不用下载license拷贝到工程中，初始化参数licenseID,包名也帮您配置好了）和未配置授权信息两种方式：



2.2 运行示例工程

2.2.1 自动配置授权信息集成

如果您是通过自动配置授权信息下载的示例工程，只需配置好证书即可。查看下项目中的FaceParameterConfig.h文件，已经自动配置

```
11
12 // 人脸license文件名
13 #define FACE_LICENSE_NAME    @"idl-license"
14
15 // 人脸license后缀
16 #define FACE_LICENSE_SUFFIX  @"face-ios"
17
18 //  (您申请的应用名称(appname)+「-face-ios」后缀，如申请的应用名称(appname)为test123，则此处填写test123-face-ios)
19 #define FACE_LICENSE_ID        @"BaiduFaceDemo-face-ios"
20
```

示例图

配置好证书，即可运行。注意已经设置好的bundle id不要随意改动。

2.2.2 未使用自动配置授权信息的集成

手动导入授权信息。 需要手动导入license文件，配置好bundleID、licenseID等信息:

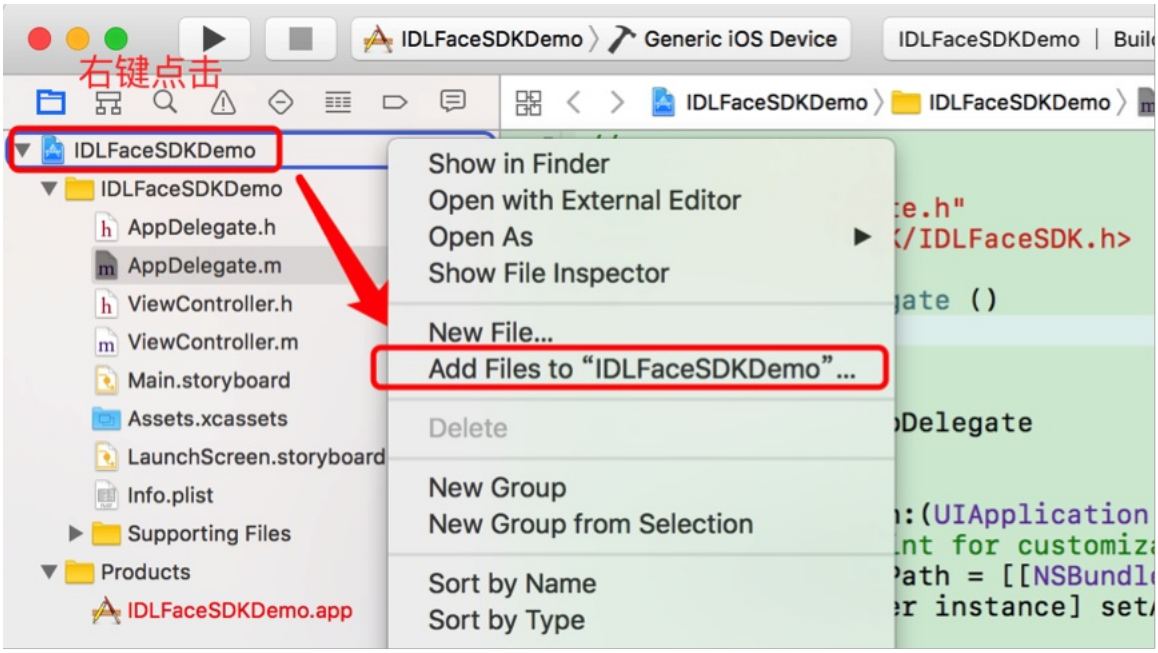
```
11
12 // 人脸license文件名
13 #define FACE_LICENSE_NAME    @"idl-license"
14
15 // 人脸license后缀
16 #define FACE_LICENSE_SUFFIX  @"face-ios"
17
18 //  (您申请的应用名称(appname)+「-face-ios」后缀，如申请的应用名称(appname)为test123，则此处填写test123-face-ios)
19 #define FACE_LICENSE_ID        @"BaiduFaceDemo-face-ios"
20
```

示例图

2.3 添加SDK到工程

1. 打开或者新建一个项目。

2. 右键点击项目，会出现一个添加菜单，在菜单中选择『Add Files to“此处是你的项目名字”…… 』,如下图所示：

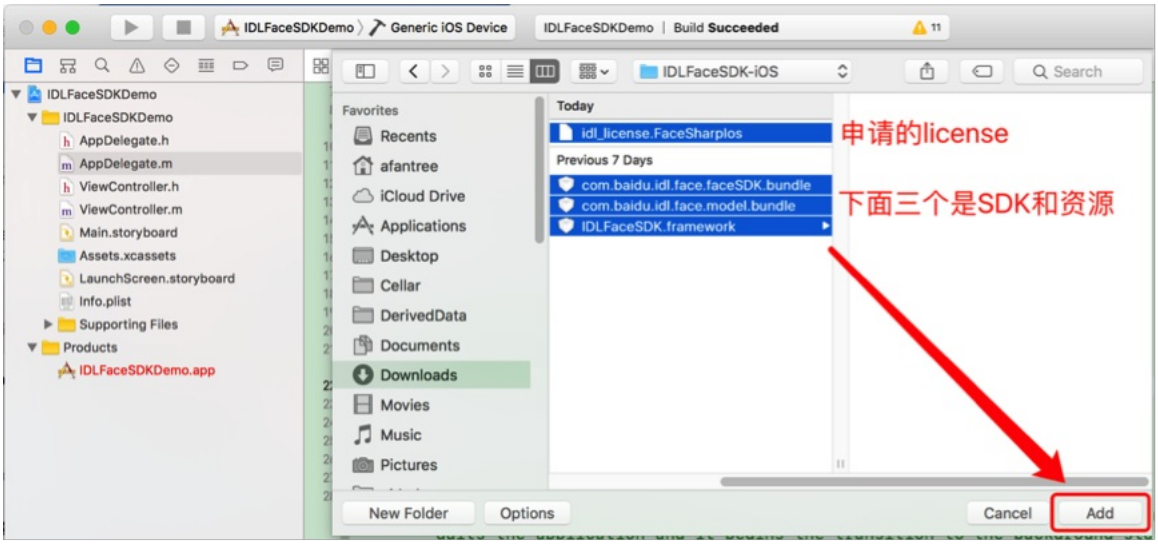


3. 在添加文件弹出框里面选择申请到的license和SDK添加进来。如下图：

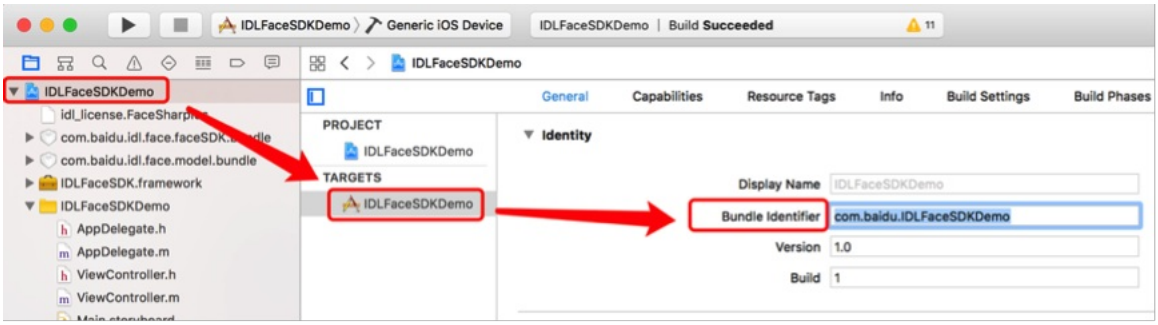
   注意：license为百度官方提供的。

   SDK包含下面三个文件:

   - IDLFaceSDK.framework
   - com.baidu.idl.face.faceSDK.bundle
   - com.baidu.idl.face.model.faceSDK.bundle



4. 确认下Bundle Identifier 是否是申请license时填报的那一个，注意：license和Bundle Identifier是一一对应关系，填错了会导致SDK不能用。



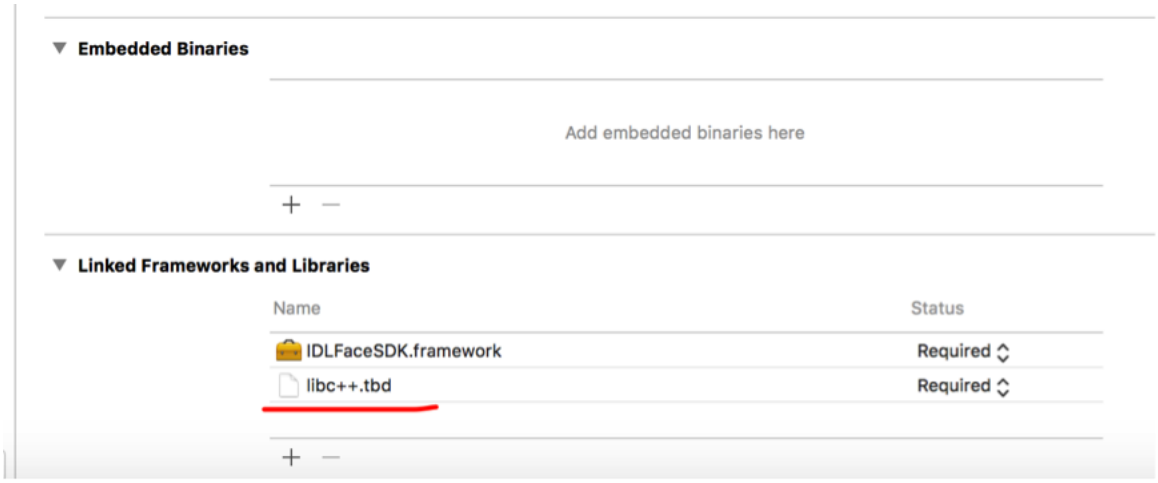
5. 填写正确的FACE_LICENSE_ID。

（即后台展示的LicenseID）

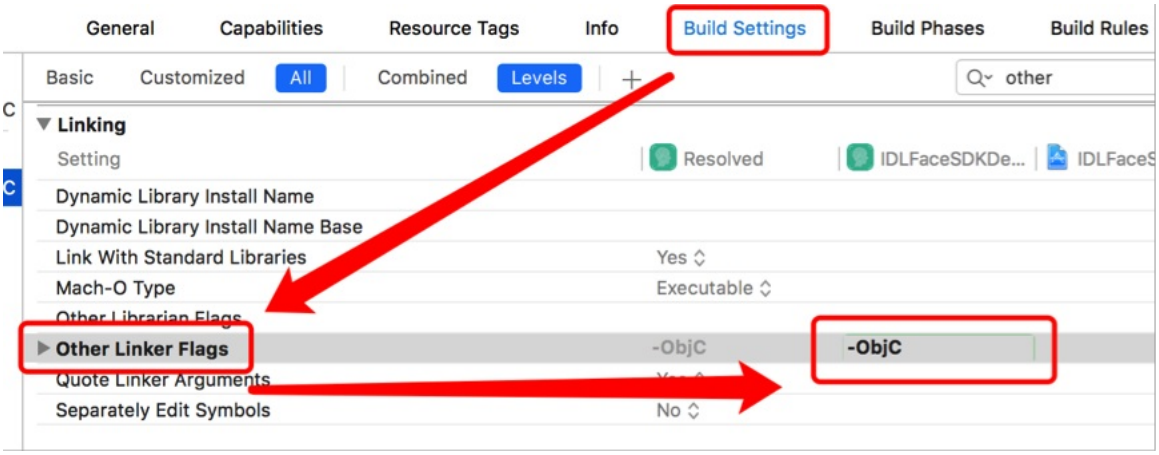
在FaceParameterConfig.h文件里面填写拼接好的FACE_LICENSE_ID。

```
//  (您申请的应用名称(appname)+「-face-ios」后缀，如申请的应用名称(appname)为test123，则此处填写test123-face-ios)
#define FACE_LICENSE_ID        @"BaiduFaceDemo-face-ios"
```

6. 选择链接C++标准库。



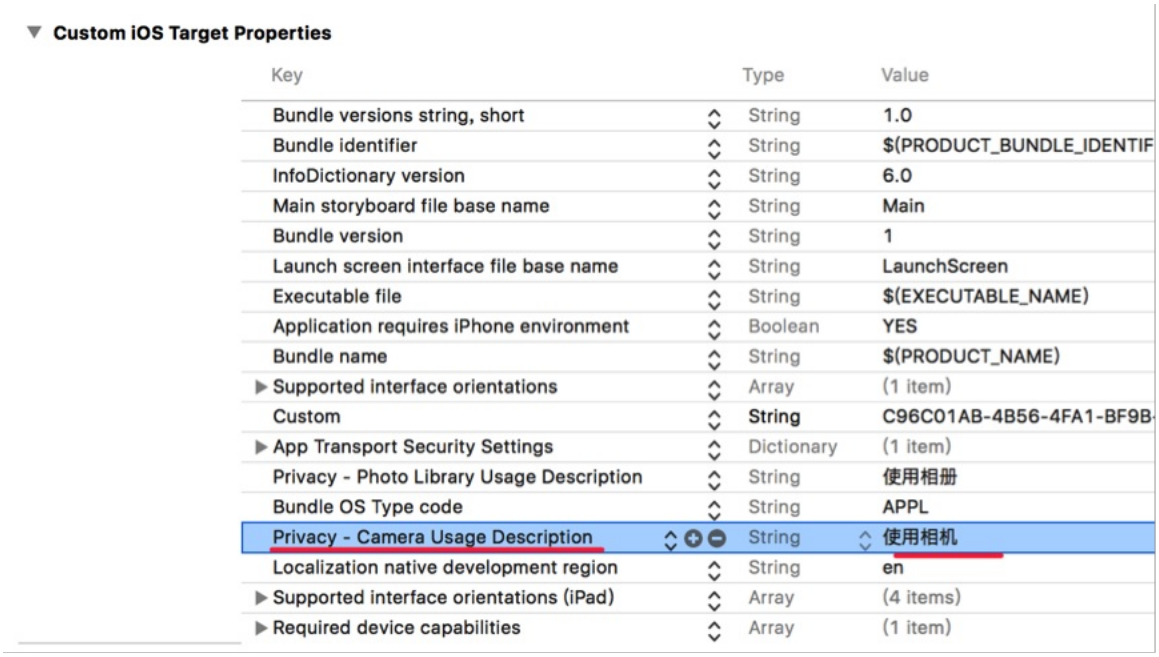
7. 如果没有使用pod管理第三方库的话，请在Build Setting >Linking > Other Linker Flags 上面加入 –ObjC 选项。如果用了pod请忽略，因为pod会自动添加上。



2.4 权限声明

需要使用相机权限：编辑Info.plist文件，添加

Privacy- Camera Usage Description 的Key值，Value为使用相机时候的提示语，可以填写：『使用相机』。



3.接口说明

3.1 FaceSDK 鉴权初始化

3.1.1 设置鉴权功能

```
 - (void)setLicenseID:(NSString *)licenseID andLocalLicenceFile:(NSString *)licensePath andRemoteAuthorize:(BOOL)remoteAuthorize;
```

参数：

- licenseId：平台申请的 licenseID
- localLicencePath：鉴权文件路径
- remoteAuthorize：是否开启网络鉴权

返回：

- 无

参考AppDelegate.m 实现，使用方法如下：

```
    NSString* licensePath = [[NSBundle mainBundle] pathForResource:FACE_LICENSE_NAME ofType:FACE_LICENSE_SUFFIX];
    NSAssert([[NSFileManager defaultManager] fileExistsAtPath:licensePath], @"license文件路径不对，请仔细查看文档");
    [[FaceSDKManager sharedInstance] setLicenseID:FACE_LICENSE_ID andLocalLicenceFile:licensePath andRemoteAuthorize:true];
```

3.1.2 鉴权成功的凭证

```
 - (BOOL)canWork
```

参数：

- 无

返回：

- True代表成功，false代表失败

参考ViewController.m 实现，判断鉴权是否通过：

```
 if (![[FaceSDKManager sharedInstance] canWork]){
  NSLog(@"授权失败，请检测ID 和 授权文件是否可用");
  return;
}
```

3.2 FaceSDK 功能初始化

3.2.1 FaceSDK 参数配置

具体方法详见如下：

```
 /**
 * 设置预测库耗能模式
 * 默认 LITE_POWER_NO_BIND
 */
- (void)setLitePower:(int)litePower;

/**
 * 需要检测的最大人脸数目
 * 默认1
 */
- (void)setMaxDetectNum:(int)detectNum；

/**
 * 需要检测的最小人脸大小
 * 默认40
 */
- (void)setMinFaceSize:(int)width;

/**
 * 人脸置信度阈值（检测分值大于该阈值认为是人脸
 * RGB
 * 默认 0.5f
 */
- (void)setNotFaceThreshold:(CGFloat)thr；

/**
 * 质量检测遮挡阈值
 * 默认0.5
 */
- (void)setOccluThreshold:(CGFloat)thr；

/**
 * 质量检测光照阈值
 * 默认100
 */
- (void)setIllumThreshold:(CGFloat)thr；

/**
 * 质量检测模糊阈值
 * 默认0.5
 */
- (void)setBlurThreshold:(CGFloat)thr；

/**
 * 姿态检测阈值
 * 默认pitch=12，yaw=12，row=10
 */
- (void)setEulurAngleThrPitch:(float)pitch yaw:(float)yaw roll:(float)roll；
```

```
/**
 * 输出图像个数
 * 默认3
 */
- (void)setMaxCropImageNum:(int)imageNum；

/**
 * 输出图像宽，设置为有效值(大于0)则对图像进行缩放，否则输出原图抠图结果
 * 默认 480
 */
- (void)setCropFaceSizeWidth:(CGFloat)width；

/**
 * 输出图像高，设置为有效值(大于0)则对图像进行缩放，否则输出原图抠图结果
 * 默认 680
 */
- (void)setCropFaceSizeHeight:(CGFloat)height；

/**
 * 输出图像，下巴扩展，大于等于0，0：不进行扩展
 * 默认0.1
 */
- (void)setCropChinExtend:(CGFloat)chinExtend；

/**
 * 输出图像，额头扩展，大于等于0，0：不进行扩展
 * 默认0.2
 */
- (void)setCropForeheadExtend:(CGFloat)foreheadExtend；

/**
 * 输出图像，人脸框与背景比例，大于等于1，1：不进行扩展
 * 默认1.5f
 */
- (void)setCropEnlargeRatio:(float)cropEnlargeRatio;

/**
 * 动作超时配置
 */
- (void)setConditionTimeout:(CGFloat)timeout；

/**
 * 语音间隔提醒配置
 */
- (void)setIntervalOfVoiceRemind:(CGFloat)timeout;

/**
 * 是否开启静默活体，默认false
 */
- (void)setIsCheckSilentLive:(BOOL)isCheck;

/**
 * 口罩检测阈值配置，默认0.8。
 * 大于阈值判定为戴口罩，低于阈值判定为未戴口罩
 */
- (void)setMouthMaskThreshold:(CGFloat)thr；

/**
 * 设置原始图片缩放比例，默认1不缩放，scale 阈值0~1
 */
- (void)setImageWithScale:(CGFloat)scale;
```

```
/**
 * 设置图片加密类型，type=0 非加密原图base64 ；type=1 基于百度安全算法加密
 */
- (void)setImageEncrypteWithType:(int) type;

/**
 *  人脸过远框比例 默认：0.4
 */
- (void)setMinRect:(float) minRectScale;

/**
 *  活体检测阈值 默认：0.8
 */
- (void)setliveThresholdValue:(float) liveThresholdValue;

/**
 *  视频录制能力 默认：关闭
 */
- (void)setRecordAbility:(BOOL) recordAbility;

/**
 *  炫彩颜色判断能力 默认：关闭
 */
- (void)setColorJudgeAbility:(BOOL) colorJudgeAbility;
```

3.2.2 FaceSDK 初始化

```
- (int)initCollect;
```

参数:

- 无

返回：

- 无

参考ViewController.m initSDK 方法实现：

```
- (void) initSDK {

    if (![[FaceSDKManager sharedInstance] canWork]){
        NSLog(@"授权失败，请检测ID 和 授权文件是否可用");
        return;
    }

    // 初始化SDK配置参数，可使用默认配置
    // 设置最小检测人脸阈值
    [[FaceSDKManager sharedInstance] setMinFaceSize:200];
    // 设置截取人脸图片高
    [[FaceSDKManager sharedInstance] setCropFaceSizeWidth:480];
    // 设置截取人脸图片宽
    [[FaceSDKManager sharedInstance] setCropFaceSizeHeight:640];
    // 设置人脸遮挡阈值
    [[FaceSDKManager sharedInstance] setOccluThreshold:0.5];
    // 设置亮度阈值
    [[FaceSDKManager sharedInstance] setIllumThreshold:40];
    // 设置图像模糊阈值
    [[FaceSDKManager sharedInstance] setBlurThreshold:0.3];
    // 设置头部姿态角度
    [[FaceSDKManager sharedInstance] setEulurAngleThrPitch:10 yaw:10 roll:10];
    // 设置人脸检测精度阈值
    [[FaceSDKManager sharedInstance] setNotFaceThreshold:0.6];
    // 设置抠图的缩放倍数
    [[FaceSDKManager sharedInstance] setCropEnlargeRatio:2.5];
    // 设置照片采集张数
    [[FaceSDKManager sharedInstance] setMaxCropImageNum:3];
    // 设置超时时间
    [[FaceSDKManager sharedInstance] setConditionTimeout:1500];
    // 设置开启口罩检测，非动作活体检测可以采集戴口罩图片
    [[FaceSDKManager sharedInstance] setIsCheckMouthMask:true];
    // 设置开启口罩检测情况下，非动作活体检测口罩过滤阈值，默认0.8 不需要修改
    [[FaceSDKManager sharedInstance] setMouthMaskThreshold:0.8f];
    // 设置原始图缩放比例
    [[FaceSDKManager sharedInstance] setImageWithScale:0.8f];
    // 设置图片加密类型，type=0 非加密原图base64 ；type=1 基于百度安全算法加密
    [[FaceSDKManager sharedInstance] setImageEncrypteType:0];
    // 设置人脸过远框比例
    [[FaceSDKManager sharedInstance] setMinRect:0.4];
    // 活体检测阈值
    [[FaceSDKManager sharedInstance] setliveThresholdValue:0.8];
    //视频录制能力
    [[FaceSDKManager sharedInstance] setRecordAbility:NO];
    //炫彩颜色判断能力
    [[FaceSDKManager sharedInstance] setColorJudgeAbility:NO];
    // 初始化SDK功能函数
    [[FaceSDKManager sharedInstance] initCollect];
}
```

3.2.3 FaceSDK 释放

```
- (int)uninitCollect
```

3.3 炫瞳活体采集

调用IDLFaceColorfulManager类的:

```
 /**
 * 人脸采集，成功之后返回扣图图片，原始图片
 * @param image 镜头拿到的图片
 * @param detectRect 预览的Rect
 * @param previewRect 检测的Rect
 * @param colorQuality 炫彩当前颜色是否通过质量检测
 * return completion 回调信息
 */
- (void)colorStratrgyWithNormalImage:(UIImage *)image previewRect:(CGRect)previewRect detectRect:(CGRect)detectRect
isColorQuality:(BOOL)colorQuality completionHandler:(ColorStrategyCompletion)completion;
```

温馨提示：NSDictionary * images 返回FaceCropImageInfo 对象包含带黑边的图片，主要是为了配合在线 API 方式的活体检测使用，本地算法会对图片做一定的预处理操作；而不带黑边的图片，如果无需使用活体，则可使用不带黑边的图片采集方式。

参数说明：

previewRect与detctRect是为了做距离检测而定义的，为了在上层封装判断脸是否在框内／离太远／离太近。

- image：相机获取的图片
- previewRect：人脸图片大小，间接定义的最大距离的 maxRect 和最小距离的 minRect。
- detectRect：人脸检测区域大小，实际采集区域
- completion：完成后返回照片和状态结果

参考BDFaceColorfulViewController 实现，使用方法如下：

```
    [[IDLFaceColorfulManager sharedInstance] colorStratrgyWithNormalImage:image previewRect:weakSelf.previewRect
detectRect:weakSelf.detectRect isColorQuality:(BOOL)weakSelf.colorQuality completionHandler:^(FaceInfo *faceinfo,
NSDictionary *images, ColorRemindCode remindCode){
            switch (remindCode) {
              case ColorRemindCodeOK: {
                  [weakSelf singleActionSuccess:true];
                  break;
              }
```

3.4 动作采集

调用IDLFaceLivenessManager类的:

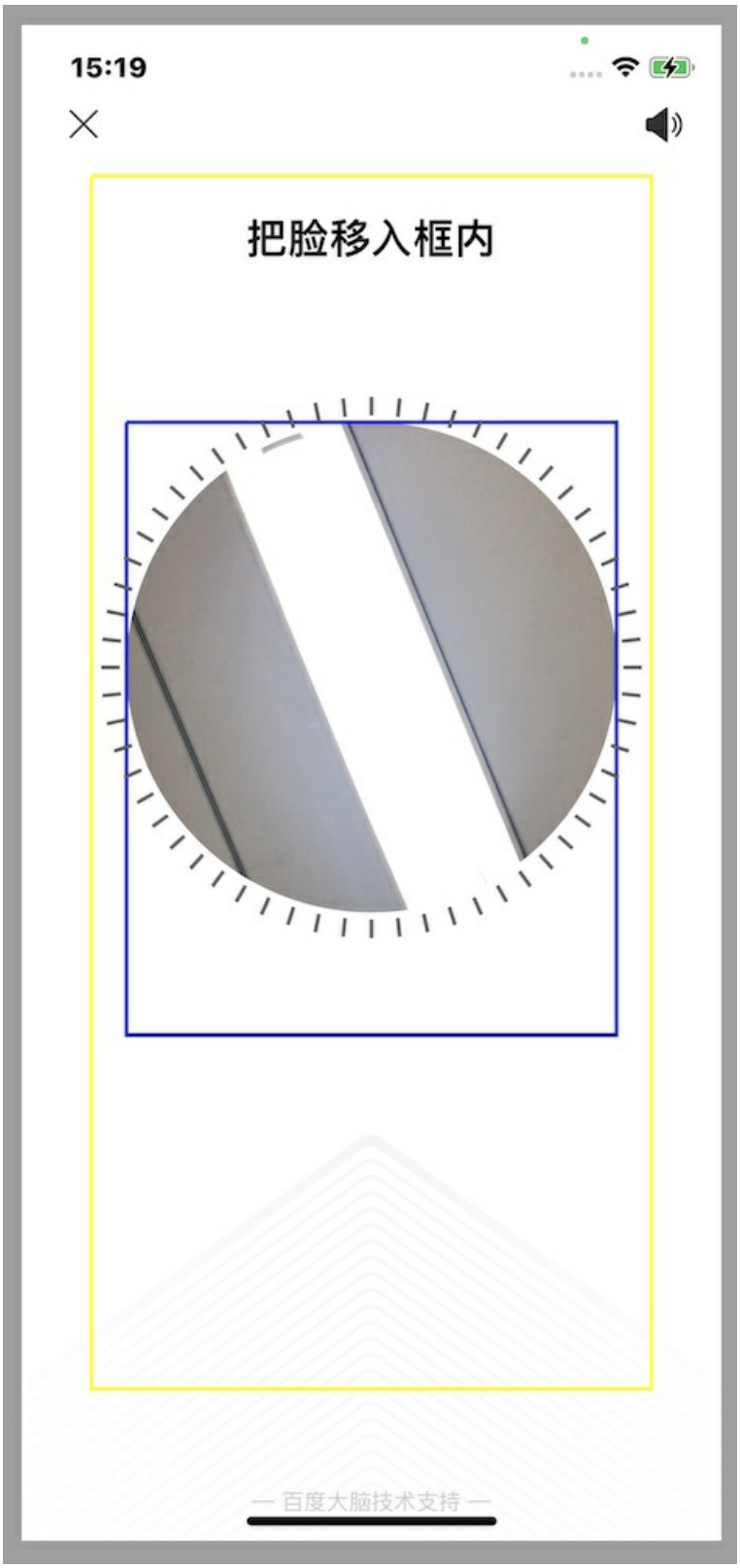
```
 /**
 *  人脸活体验证，成功之后扣图图片，原始图片三种
 * @param image 镜头拿到的图片
 * @param previewRect 预览的Rect
 * @param detectRect 检测的Rect
 * return completion 回调信息
 */
-(void) livenessNormalWithImage:(UIImage *)image previewRect:(CGRect)previewRect detectRect:(CGRect)detectRect
completionHandler:(LivenessNormalCompletion)completion;

typedef void (^LivenessNormalCompletion) (NSDictionary * images, FaceInfo *faceInfo, LivenessRemindCode
remindCode);
```

温馨提示：NSDictionary * images 返回FaceCropImageInfo 对象包含带黑边的图片，主要是为了配合在线 API 方式的活体检测使用，本地算法会对图片做一定的预处理操作；而不带黑边的图片，如果无需使用活体，则可使用不带黑边的图片采集方式。

参数说明：

previewRect与detctRect是为了做距离检测而定义的，为了在上层封装判断脸是否在框内／离太远／离太近。

- image：相机获取的图片
- previewRect： 人脸图片大小(下图黄色框)
- detectRect：人脸检测区域大小，实际采集区域，间接定义的最大距离的 maxRect 和最小距离的 minRect。（下图蓝色框）
- completion：完成后返回照片和状态结果



说明：

- 检测图片中的人脸信息，完成人脸图像采集和人脸活体检测，返回检测状态和结果。

```
    [[IDLFaceLivenessManager sharedInstance] livenessNormalWithImage:image previewRect:self.previewRect
detectRect:self.detectRect completionHandler:^(NSDictionary *images, FaceInfo *faceInfo, LivenessRemindCode
remindCode) {

        switch (remindCode) {
            case LivenessRemindCodeOK: {
                weakSelf.hasFinished = YES;
                [self warningStatus:CommonStatus warning:@"非常好"];
                if (images[@"image"] != nil && [images[@"image"] count] != 0) {

                    NSArray *imageArr = images[@"image"];
                    for (FaceCropImageInfo * image in imageArr) {
                        NSLog(@"cropImageWithBlack %f %f", image.cropImageWithBlack.size.height,
image.cropImageWithBlack.size.width);
                        NSLog(@"originalImage %f %f", image.originalImage.size.height, image.originalImage.size.width);
                    }

                    FaceCropImageInfo * bestImage = imageArr[0];
```

3.5 ZID参数获取 调用金融级实名认证接口配合金融级SDK使用时，需传入zid参数打开大数据风控功能。

获取方法：

```
    [[FaceSDKManager sharedInstance] getZtoken]
```

4.常见问题

Q：鉴权问题。提示「验证失败」 A：先确定网络情况是否正常，本地鉴权文件失效了才走网络鉴权。定位错误码，排查鉴权失败的原因。一般是 licenseID 和 bundleID 配置不一致导致的鉴权失败。请注意上线前授权文件一定要更新。

Q：license 文件失效了，不能用了怎么办？ A：License 文件申请时候有期限，如过期会导致校验失效，需要在后台进行申请延期。

Q：使用 iOS 采集端，采集到的图片是斜着的，这个正常吗，会影响识别吗？ A：不会影响识别。有黑边和倾斜是因为图片质量算法造成的，我们是按 1:3 对图像进行背景填充使人脸居中，为的是更好的识别图像。这个版本提供了`detectStratrgyWithQualityControlImage`和`detectStratrgyWithNormalImage`两种方法供选择。

Q：鉴权初始化的错误码对应的信息 A： | ErrorCode | 常量值 | 说明 | | ---------- | ------ | ---------- | | SUCCESS | 0 | 成功 | | LICENSE_NOT_INIT_ERROR | 1 | license未初始化 | | LICENSE_DECRYPT_ERROR | 2 | license数据解密失败 | | LICENSE_INFO_FORMAT_ERROR | 3 | license数据格式错误 | | LICENSE_KEY_CHECK_ERROR | 4 | license-key(api-key)校验错误 | | LICENSE_ALGORITHM_CHECK_ERROR | 5 | 算法ID校验错误 | | LICENSE_MD5_CHECK_ERROR | 6 | MD5校验错误 | | LICENSE_DEVICE_ID_CHECK_ERROR | 7 | 设备ID校验错误 | | LICENSE_PACKAGE_NAME_CHECK_ERROR | 8 | 包名(应用名)校验错误 | | LICENSE_EXPIRED_TIME_CHECK_ERROR | 9 | 过期时间不正确 | | LICENSE_FUNCTION_CHECK_ERROR | 10 | 功能未授权 | | LICENSE_TIME_EXPIRED | 11 | 授权已过期 | | LICENSE_LOCAL_FILE_ERROR | 12 | 本地文件读取失败 | | LICENSE_REMOTE_DATA_ERROR | 13 | 远程数据拉取失败 | | LICENSE_LOCAL_TIME_ERROR | 14 | 本地时间校验错误 | | OTHER_ERROR | 0xff | 其他错误 |

更多问题请点击 常见问题

离线识别SDK

Android-SDK-v2.0说明

版本日志

版本	日期	更新说明
v2.0.3	2019.06.14	1、更新硬件指纹获取模块，优化特殊环境下指纹变更的问题
v2.0.2	2019.04.01	1、全新人脸检测模型，检测追踪更流畅； 2、全新证件照模型：体积更小，速度更快 ； 3、部分接口细节优化
v2.0.1	2019.03.14	1、接口设计优化； 2、增加几款结构光镜头支持； 3、已知bug修复
v2.0.0	2019.01.10	1、优化生活照模型精度及速度 2、优化检测模型及策略 3、优化接口设计 4、优化活体检测速度
v1.1.0	2018.09.03	1、增加离线证件照特征抽取模型，可有效处理芯片照、证件照比对需求 2、增加离线人脸属性模型，支持性别、年龄等属性分析 3、增加多线程支持 4、增加离线激活支持，可导入授权文件无网激活 5、增加对奥比中光Astra Pro深度图镜头模组支持 6、增加对华捷艾米深度镜头模组支持 7、其他细节优化及已知bug修复
v1.0.1	2018.08.03	1、修复设备指纹发生变化bug 2、替换近红外活体模型，优化效果 3、修复批量注册人脸到人脸库，失败问题 4、修复注册、图片人脸检测、视频返回图片抽取特征失败，经常出现检测不到人脸问题
v1.0.0	2018.06.29	初版，包括离线人脸采集、离线活体检测、离线对比识别、离线人脸库管理等功能

目录

1、FaceAuth-鉴权接口
1.1 鉴权-在线授权
1.2 鉴权-SDK 压缩文件，本地鉴权
1.3 开启底层Log输出
1.4 设置Anakin核数
2、FaceDetect-检测接口
2.1 检测对齐模型加载
2.2 质量检测模型加载
2.3 配置信息加载
2.4 YUV图片转换ARGB
2.5 人脸框检测
2.6 人脸跟踪-最大人脸接口
2.7 人脸跟踪-第一个人脸接口
2.8 人脸跟踪-多人脸检测
2.9 人脸图像质量检测
2.10 检测方法类型设置
2.11 人脸图片信息清理
3、FaceLive-活体接口
3.1 活体模型加载
3.2 人脸静默活体检测-RGB可见光
3.3 人脸静默活体检测-NIR近红外或Depth深度图
4、FaceFeature-特征接口
4.1 特征模型加载
4.2 人脸特征提取
4.3 人脸特征比对
5、FaceAttributes-属性接口
5.1 属性情绪模型加载
5.2 人脸属性检测
5.3 人脸表情检测
6、人脸信息实体类
6.1 基础信息实体类
6.2 扩展信息实体类

1、FaceAuth鉴权接口

1.1 鉴权-在线授权

说明：用户通过申请授权码，在线授权，激活设备

```
void initLicenseOnLine(final Context context, final String licenseKey, final AuthCallback callback)
```

参数名	含义
context	当前上下文
licenseKey	AIPE 鉴权码
callback	鉴权结果 void onResponse(int code, String response) code 0：成功；code 1 加载失败 response 结果信息

1.2 鉴权-SDK 压缩文件，本地鉴权

说明：用户通过申请鉴权文件，存储在SD 卡根目录下，离线鉴权，激活设备

```
void initLicenseOffLine(final Context context, final AuthCallback callback)
```

参数名	含义
context	当前上下文
callback	鉴权结果 void onResponse(int code, String response) code 0：成功；code 1 加载失败 response 结果信息

1.3 开启底层Log输出

说明：用于Debug 时候输出LOG 详细信息

```
void setActiveLog(BDFaceLogInfo logInfo)
```

参数名	含义
BDFaceLogInfo	底层log 打印 BDFACE_LOG_VALUE_MESSAGE, // 打印输出值日志 BDFACE_LOG_ERROR_MESSAGE, // 打印输出错误日志 BDFACE_LOG_ALL_MESSAGE, // 打印所有日志

1.4 设置Anakin核数

说明：根据开发板类型，设置加速对Cpu 核数依赖，调整参数，提高性能

```
void setAnakinThreadsConfigure(int flagsThreads ,int flagsCluster)
```

参数名	含义
flagsThreads	大核个数（建议：3288传入2，3399传入4）
flagsCluster	小核个数（建议：传入0）

2、FaceDetect检测接口

2.1 检测对齐模型加载

说明：检测模型加载，目前支持可见光模型，近红外检测模型（非必要参数，可以为空），对齐模型

```
void initModel(final Context context, final String visModel,final String nirModel,final String alignModel, final Callback callback)
```

参数名	含义
context	上下文context
visModel	可见光图片检测模型
nirModel	红外图片检测模型（非必要参数，可以为空）
alignModel	对齐类型
callback	模型加载结果 void onResponse(int code, String response) code 0：成功；code 1 加载失败 response 结果信息

2.2 质量检测模型加载

说明：质量检测模型加载，判断人脸遮挡信息，光照信息，模糊信息，模型包含模糊模型，遮挡信息，作用于质量检测接口

```
void initQuality(final Context context, final String blurModel, final String occlurModel, final Callback callback)
```

参数名	含义
context	上下文context
blurModel	模糊检测模型
occlurModel	遮挡检测模型
callback	鉴权结果 void onResponse(int code, String response) code 0：成功；code 1 加载失败 response 结果信息

2.3 配置信息加载

说明：检测最小人脸，是否开启内部质量检测，检测或者追踪时间间隔等配置

```
void loadConfig(BDFaceSDKConfig config)
```

参数名	含义
config	参数配置实体

2.4 YUV图片转换ARGB

说明：摄像头回调数据格式转化，YUV 转为ARGB 用于检测

```
int getDataFromYUVimg(byte[] dataYUV, int[] imageData, int width, int height, int angle, int flip)
```

参数名	含义
dataYUV	YUV 图片字节数
imageData	RGBA 图片返回值
width	图片宽
height	图片高
angle	原图的旋转角度
flip	原图是否镜像（0 非镜像 1 镜像）

2.5 人脸框检测

说明：人脸框检测，每一帧图片都会检测，返回基本人脸信息，可以人脸框绘制

```
FaceInfo[] detect(int[] imageData, int height, int width,int minFaceSize)
```

参数名	含义
imageData	ARGB图片像素点
height	图片高
width	图片宽
minFaceSize	需要检测的最小人脸尺寸

返回
成功则返回 FaceInfo[] 数组，包含：人脸框width，人脸angle，人脸框中心坐标x,y，人脸可信度mConf

```
        if (faceDetect == null) {
            Toast.makeText(mContext,
                    "未初始化检测模型", Toast.LENGTH_SHORT).show();
            return;
        }
        faceDetect.setDetectMethodType(FaceDetect.DetectType.DETECT_VIS);
        FaceInfo[] faceInfos = faceDetect.detect(
                liveVisImg.data, liveVisImg.height, liveVisImg.width, 50);
        if (faceInfos != null && faceInfos.length > 0) {
            FaceInfo faceInfo = faceInfos[0];
            StringBuilder info = new StringBuilder();
            info.append("face info length:").append(faceInfos.length)
                    .append(" face_id:").append(faceInfo.face_id)
                    .append(" width:").append(faceInfo.mWidth)
                    .append(" score:").append(faceInfo.mConf);
            Toast.makeText(mContext,
                    info,
                    Toast.LENGTH_SHORT).show();
            Log.e("handler", info.toString());
        } else {
            String info = " face info length 0";
            Toast.makeText(mContext,
                    info, Toast.LENGTH_SHORT).show();
            Log.e("handler", info);
        }
```

2.6 人脸跟踪-最大人脸接口

说明：人脸跟踪检测，追踪图片中最大人脸信息，接口包含检测和跟踪功能，返回基本人脸信息和72 关键点，可以绘制人脸框，描绘眼耳鼻嘴关键点，也可作用于后续活体，特征抽取入参。

```
FaceInfo[] trackMaxFace(int[] imageData, int height, int width)
```

参数名	含义
imageData	ARGB图片像素点
height	图片高
width	图片宽

返回
成功则返回 FaceInfo[] 数组，包含：人脸框width，人脸angle，人脸框中心坐标x,y，人脸可信度mConf，72个关键点landmark，人脸face_id，三个姿态角度headPose，动作活体数据is_live，用户根据自己需求选择

```
 if (faceDetect == null) {
    Toast.makeText(mContext,
          "未初始化检测模型", Toast.LENGTH_SHORT).show();
    return;
}
faceDetect.setDetectMethodType(FaceDetect.DetectType.DETECT_VIS);
FaceInfo[] faceInfos = faceDetect.trackMaxFace(
      liveVisImg.data, liveVisImg.height, liveVisImg.width
);
if (faceInfos != null && faceInfos.length > 0) {
    FaceInfo faceInfo = faceInfos[0];
    StringBuilder info = new StringBuilder();
    info.append("face info length:").append(faceInfos.length)
          .append(" face_id:").append(faceInfo.face_id)
          .append(" width:").append(faceInfo.mWidth)
          .append(" score:").append(faceInfo.mConf);
    if (faceInfo.occlu != null && faceInfo.occlu.length == 7) {
        info.append("face occlusion: ")
              .append(" L eye:").append(faceInfo.occlu[0])
              .append(" R eye:").append(faceInfo.occlu[1])
              .append(" nose:").append(faceInfo.occlu[2])
              .append(" mouth:").append(faceInfo.occlu[3])
              .append(" L contour:").append(faceInfo.occlu[4])
              .append(" R contour:").append(faceInfo.occlu[5])
              .append(" chin contour:").append(faceInfo.occlu[6]);
        info.append("face blur: ").append(faceInfo.blur);
        info.append("face illum: ").append(faceInfo.illum);
    }
    Toast.makeText(mContext,
          info,
          Toast.LENGTH_SHORT).show();
    Log.e("handler", info.toString());
} else {
    String info = " face info length 0";
    Toast.makeText(mContext,
          info, Toast.LENGTH_SHORT).show();
    Log.e("handler", info);
}
faceDetect.clearTrackedFaces();
```

2.7 人脸跟踪-第一个人脸接口

说明：人脸跟踪检测，追踪图片中第一个人脸信息，接口包含检测和跟踪功能，返回基本人脸信息和72 关键点，可以绘制人脸框，描绘眼耳鼻嘴关键点，也可作用于后续活体，特征抽取入参。

```
FaceInfo[] trackFirstFace(int[] imageData, int height, int width)
```

参数名	含义
imageData	ARGB图片像素点
height	图片高
width	图片宽

返回
成功则返回 FaceInfo[] 数组，包含：人脸框width，人脸angle，人脸框中心坐标x,y，人脸可信度mConf，72个关键点landmark，人脸face_id，三个姿态角度headPose，动作活体数据is_live，用户根据自己需求选择

2.8 人脸跟踪-多人脸检测

说明：人脸跟踪检测，追踪图片中多个人脸信息，通过参数num 配置，接口包含检测和跟踪功能，返回基本人脸信息和72 关键点，可以绘制人脸框，描绘眼耳鼻嘴关键点，也可作用于后续活体，特征抽取入参。

```
FaceInfo[] track(int[] imageData, int height, int width, int num)
```

参数名	含义
imageData	ARGB图片像素点
height	图片高
width	图片宽
num	最大跟踪人脸个数

返回
成功则返回 FaceInfo[] 数组，包含：人脸框width，人脸angle，人脸框中心坐标x,y，人脸可信度mConf，72个关键点landmark，人脸face_id，三个姿态角度headPose，动作活体数据is_live，用户根据自己需求选择

2.9 人脸图像质量检测

说明：单个原子方法质量检测方法（也可以配置BDFaceSDKConfig 中的 isCheckBlur 等三个参数，在trackMaxFace 和 trackFirstFace 方法内部生效）

```
int imgQuality(int[] imageData, int height, int width, int[] landmarks,float[] bluriness, int[] illum, float[] occlusion, int[] nOccluPart)
```

参数名	含义
imageData	图片像素点
height	图片高
width	图片宽
landmark	人脸72个关键点FaceInfo.landmark
bluriness	模糊结果
illum	光照结果
occlusion	遮挡结果
nOccluPart	遮挡部位

返回
成功则返回 1

```
 if (faceDetect == null) {
    Toast.makeText(mContext,
          "未初始化检测模型", Toast.LENGTH_SHORT).show();
    return;
}
faceDetect.setDetectMethodType(FaceDetect.DetectType.DETECT_VIS);
FaceInfo[] faceInfos = faceDetect.trackMaxFace(
      liveVisImg.data, liveVisImg.height, liveVisImg.width);
if (faceInfos != null && faceInfos.length > 0) {
    FaceInfo faceInfo = faceInfos[0];
    // 质量检测通过加入人脸列表
    float[] bluriness = new float[1];
    int[] illum = new int[1];
    float[] occlusion = new float[7];
    int[] nOccluPart = new int[1];
    faceDetect.imgQuality(liveVisImg.data, liveVisImg.height,
          liveVisImg.width, faceInfo.landmarks, bluriness, illum, occlusion, nOccluPart);
    StringBuilder info = new StringBuilder();
    info.append("face occlusion: ")
          .append(" L eye:").append(occlusion[0])
          .append(" R eye:").append(occlusion[1])
          .append(" nose:").append(occlusion[2])
          .append(" mouth:").append(occlusion[3])
          .append(" L contour:").append(occlusion[4])
          .append(" R contour:").append(occlusion[5])
          .append(" chin contour:").append(occlusion[6]);
    info.append("face blur: ").append(bluriness[0]);
    info.append("face illum: ").append(illum[0]);
    Toast.makeText(mContext,
          info,
          Toast.LENGTH_SHORT).show();
    Log.e("handler", info.toString());
} else {
    String info = " face info length 0";
    Toast.makeText(mContext,
          info, Toast.LENGTH_SHORT).show();
    Log.e("handler", info);
}
faceDetect.clearTrackedFaces();
```

2.10 检测方法类型设置

说明：检测方法类型设置，建议FACEBOX_VIS 可见光检测

```
void setDetectMethodType(DetectType detectMethodType)
```

参数名	含义
detectMethodType	图片检测类型(目前两种支持FACEBOX_VIS和FACEBOX_NIR，建议FACEBOX_VIS 可见光检测)

2.11 人脸图片信息清理

说明：清空trackMaxFace 和 trackFirstFace 接口检测缓存数据，在注册或者识别成功之后，可以调用该接口

```
void clearTrackedFaces()
```

3、FaceLive活体接口

3.1 活体模型加载

说明：静默活体检测模型初始化，可见光活体模型，近红外活体模型，深度活体模型初始化

```
void initModel(final Context context,final String visModel,final String nirModel,final String depthModel,final Callback callback)
```

参数名	含义
context	上下文context
visModel	可见光图片活体模型
nirModel	红外图片活体模型
depthModel	深度图活体模型
callback	模型加载结果 void onResponse(int code, String response) code 0：成功；code 1 加载失败 response 结果信息

3.2 人脸静默活体检测-RGB可见光

说明：可见光静默活体分值检测，返回0-1结果，建议超过0.9 为活体

```
float silentLive(LiveType liveType, int[] imageData, int height, int width, int[] landmarks)
```

参数名	含义
liveType	FaceLive.LiveType.LIVEID_VIS 可见光图像静默活体检测<br /
imageData	ARGB图片像素点
height	图片高
width	图片宽
landmark	人脸72个关键点FaceInfo.landmark

返回
成功则返回活体分值

```
        if (faceDetect == null) {
            Toast.makeText(mContext,
                    "未初始化检测模型", Toast.LENGTH_SHORT).show();
            return;
        }

        if (faceLive == null) {
            Toast.makeText(mContext,
                    "未初始化活体模型", Toast.LENGTH_SHORT).show();
            return;
        }
        faceDetect.setDetectMethodType(FaceDetect.DetectType.DETECT_VIS);

        FaceInfo[] faceInfos = faceDetect.trackMaxFace(
                liveVisImg.data, liveVisImg.height, liveVisImg.width
        );

        if (faceInfos != null && faceInfos.length > 0) {
            FaceInfo faceInfo = faceInfos[0];

            float visScore = faceLive.silentLive(FaceLive.LiveType.LIVEID_VIS,
                    liveVisImg.data, liveVisImg.height, liveVisImg.width,
                    faceInfo.landmarks);
            float nirScore = faceLive.silentLive(FaceLive.LiveType.LIVEID_NIR,
                    liveVisImg.dataByte, liveVisImg.height, liveVisImg.width,
                    faceInfo.landmarks);
            float depthScore = faceLive.silentLive(FaceLive.LiveType.LIVEID_DEPTH,
                    liveDepthImg.dataByte, liveVisImg.height, liveVisImg.width,
                    faceInfo.landmarks);
            StringBuilder builder = new StringBuilder();
            builder.append("live-vis").append(visScore)
                    .append("  live-nir").append(nirScore)
                    .append("  live-depth").append(depthScore);
```

3.3 人脸静默活体检测-NIR近红外或Depth深度图

说明：近红外和深度静默活体分值检测，返回0-1结果，建议超过0.9 为活体；红外和深度为byte 字节数组

```
float silentLive(LiveType liveType, byte[] imageData, int height, int width, int[] landmarks)
```

参数名	含义
liveType	FaceLive.LiveType.LIVEID_NIR近红外图像静默活体检测 FaceLive.LiveType.LIVEID_DEPTH 深度图静默活体检测
imageData	灰度图片像素点或深度图片像素点
height	图片高
width	图片宽
landmark	人脸72个关键点FaceInfo.landmark

返回
成功则返回活体分值

4、FaceFeature特征接口

4.1 特征模型加载

说明：离线特征获取模型加载，目前支持可见光模型，近红外检测模型（非必要参数，可以为空），证件照模型；用户根据自

己场景，选择相应场景模型

```
initModel(final Context context,final String idPhotoModel,final String visModel,final String nirModel,final Callback callback)
```

参数名	含义
context	上下文context
idPhotoModel	证件照图片模型
visModel	可见光图片模型
nirModel	红外图片模型（非必要参数，可以为空）
callback	模型加载结果 void onResponse(int code, String response) code 0：成功；code 1 加载失败 response 结果信息

4.2 人脸特征提取

说明：离线特征提取接口，通过featureType 提取不同图片特征数据，函数返回特征个数，特征存储在feature 参数中

```
float feature(FeatureType featureType, int[] imageData, int height, int width, int[] landmarks, byte[] feature)
```

参数名	含义
featureType	FeatureType.FEATURE_VIS生活照 FeatureType.FEATURE_ID_PHOTO证件照照
imageData	图片像素点
height	图片高
width	图片宽
landmark	人脸72个关键点FaceInfo.landmark
feature	人脸特征 feature 数组，默认初始化512空字节

返回
成功则返回512 特征点

```
 if (faceDetect == null) {
    Toast.makeText(mContext,
          "未初始化检测模型", Toast.LENGTH_SHORT).show();
    return;
}

if (faceFeature == null) {
    Toast.makeText(mContext,
          "未初始化特征模型", Toast.LENGTH_SHORT).show();
    return;
}
faceDetect.setDetectMethodType(FaceDetect.DetectType.DETECT_VIS);

FaceInfo[] faceInfos = faceDetect.trackMaxFace(
      featureIdPImg.data, featureIdPImg.height, featureIdPImg.width
);

byte[] feature1 = new byte[512];
if (faceInfos != null && faceInfos.length > 0) {
    FaceInfo faceInfo = faceInfos[0];
    float length = faceFeature.feature(FaceFeature.FeatureType.FEATURE_VIS,
          featureIdPImg.data, featureIdPImg.height, featureIdPImg.width, faceInfo.landmarks, feature1);
    Log.e("handler", "VIS feature 1 length " + length);
}
faceDetect.clearTrackedFaces();

FaceInfo[] faceInfos2 = faceDetect.trackMaxFace(
      featureVisImg.data, featureVisImg.height, featureVisImg.width
);

byte[] feature2 = new byte[512];
if (faceInfos2 != null && faceInfos2.length > 0) {
    FaceInfo faceInfo2 = faceInfos2[0];
    float length = faceFeature.feature(FaceFeature.FeatureType.FEATURE_VIS,
          featureVisImg.data, featureVisImg.height, featureVisImg.width, faceInfo2.landmarks, feature2);
    Log.e("handler", "VIS feature 2 length " + length);
}
faceDetect.clearTrackedFaces();

float result = faceFeature.featureCompare(FaceFeature.FeatureType.FEATURE_VIS,
      feature1, feature2);
```

4.3 人脸特征比对

说明：离线特征比对结果，分值为0--100 之间

```
float featureCompare(FeatureType featureType, byte[] feature1, byte feature2)
```

参数名	含义
featureType	FeatureType.FEATURE_VIS生活照 FeatureType.FEATURE_ID_PHOTO证件照照
feature1	特征1
feature2	特征2

返回
成功则返回 比对结果（0--100）

5、FaceAttributes属性接口

5.1 属性情绪模型加载

说明：人脸属性（年龄，性别，戴眼镜等），情绪（喜怒哀乐）模型初始化

```
void initModel(final Context context,final String atttibuteModel,final String emotionModel,final Callback callback)
```

参数名	含义
context	上下文context
atttibuteModel	属性模型
emotionModel	情绪模型
callback	模型加载结果 void onResponse(int code, String response) code 0：成功；code 1 加载失败 response 结果信息

5.2 人脸属性检测

说明：获取人脸属性信息，包含年龄，表情，种族，性别，是否佩戴眼镜

```
BDFaceSDKAttribute attribute(int[] imageData, int height, int width, int[] landmarks)
```

参数名	含义
imageData	ARGB图片像素点
height	图片高
width	图片宽
landmark	人脸72个关键点FaceInfo.landmark

返回
成功则返回 BDFaceSDKAttribute

```
        if (faceDetect == null) {
            Toast.makeText(mContext,
                    "未初始化检测模型", Toast.LENGTH_SHORT).show();
            return;
        }

        if (faceAttributes == null) {
            Toast.makeText(mContext,
                    "未初始化属性模型", Toast.LENGTH_SHORT).show();
            return;
        }

        faceDetect.setDetectMethodType(FaceDetect.DetectType.DETECT_VIS);

        FaceInfo[] faceInfos = faceDetect.trackMaxFace(
                liveVisImg.data, liveVisImg.height, liveVisImg.width);

        if (faceInfos != null && faceInfos.length > 0) {
            FaceInfo faceInfo = faceInfos[0];
            BDFaceSDKAttribute attribute = faceAttributes.attribute(liveVisImg.data,
                    liveVisImg.height, liveVisImg.width,
                    faceInfo.landmarks);

            BDFaceSDKEmotions emotions = faceAttributes.emotions(liveVisImg.data,
                    liveVisImg.height, liveVisImg.width,
                    faceInfo.landmarks);

            final StringBuilder builder = new StringBuilder();
            if (attribute != null) {
                builder.append(" attribute:");
                builder.append(attribute.age);
                builder.append("    ").append(attribute.emotion);
                builder.append("    ").append(attribute.gender);
                builder.append("    ").append(attribute.glasses);
            }
            if (emotions != null) {
                builder.append("\n emotion:");
                builder.append("    ").append(emotions.emotion);
                builder.append("    ").append(emotions.expression_conf);
            }
            Log.e("handler", "attribute  " + builder.toString());

        }
        faceDetect.clearTrackedFaces();
```

5.3 人脸表情检测

说明：获取人脸表情信息，包含生气，开心，厌恶，害怕，惊讶等

```
BDFaceSDKEmotions emotions(int[] imageData, int height, int width, int[] landmarks)
```

参数名	含义
imageData	ARGB图片像素点
height	图片高
width	图片宽
landmark	人脸72个关键点FaceInfo.landmark

返回
成功则返回BDFaceSDKEmotions

6、人脸信息实体类

6.1 基础信息实体类

```
 /**
 * 人脸信息实体类
 */

public class FaceInfo {
    public float mWidth; // rectangle width
    public float mAngle; // rectangle tilt angle [-45 45] in degrees
    public float mCenter_y;   // rectangle center y
    public float mCenter_x;  // rectangle center x
    public float mConf; // face detection score

    public int[] landmarks;
    public int face_id;
    public float[] headPose;
    public int[] is_live;

    public float illum = 0f;
    public float blur = 0f;
    public float[] occlu;

    public FaceInfo(float width, float angle, float y, float x, float conf) {
        mWidth = width;
        mAngle = angle;
        mCenter_y = y;
        mCenter_x = x;
        mConf = conf;
        landmarks = null;
        face_id = 0;
    }

    public FaceInfo(float width, float angle, float y, float x, float conf, int track_id, int[] ldmks) {
        mWidth = width;
        mAngle = angle;
        mCenter_y = y;
        mCenter_x = x;
        mConf = conf;
        landmarks = ldmks;
        face_id = track_id;
    }

    public FaceInfo(float width, float angle, float y, float x, float conf,
                    int track_id, int[] ldmks, float[] pose, int[] livestatus,
                    float illum,float blur,float[] occlu
    ){
        mWidth = width;
        mAngle = angle;
        mCenter_y = y;
        mCenter_x = x;
        mConf = conf;
        landmarks = ldmks;
        face_id = track_id;
        headPose = pose;
        is_live = livestatus;
        this.illum = illum;
        this.blur = blur;
        this.occlu = occlu;
    }
}
```

6.2 扩展信息实体类

```
 // 图片信息类
public class BDFaceSDKImageInfo {
    public int height;                          // 图片高度
    public int widht;                           // 图片宽度
    public int[] data;                          // 图片数据
    public BDFaceSDKCommon.BDFaceImageType type;  // 图片格式

    public BDFaceSDKImageInfo(int height, int widht, int[] data, int type) {
        this.height = height;
        this.widht = widht;
        this.data = data;
        this.type = BDFaceSDKCommon.BDFaceImageType.values()[type];
    }
}

public class BDFaceSDKConfig {
    /**
     * 最小人脸检测大小 建议50
     */
    public int minFaceSize = 50;

    /**
     * 最大人脸检测大小 建议-1(不做限制)
     */
    public int maxFaceSize = -1;

    /**
     * 人脸跟踪，检测的时间间隔 默认 500ms
     */
    public int trackInterval = 0;

    /**
     * 人脸跟踪，跟踪时间间隔 默认 1000ms
     */
    public int detectInterval = 0;

    /**
     * 人脸置信度阈值，建议值0.5
     */
    public float noFaceSize = 0.5f;

    /**
     * 人脸姿态角 pitch,yaw,roll
     */
    public int pitch;
    public int yaw;
    public int roll;

    /**
     * 质量检测模糊，遮挡，光照，默认不做质量检测
     */
    public boolean isCheckBlur = true;
    public boolean isOcclusion = true;
    public boolean isIllumination = true;

    /**
     * 检测图片类型，可见光或者红外
     */
    public FaceDetect.DetectType detectMethodType
            = FaceDetect.DetectType.DETECT_VIS;
}
```

```
// 人脸属性
public class BDFaceSDKAttribute {
    public int age;                                 // 年龄
    public BDFaceSDKCommon.BDFaceRace race;         // 种族
    public BDFaceSDKCommon.BDFaceEmotion emotion;   // 表情
    public BDFaceSDKCommon.BDFaceGlasses glasses;   // 戴眼镜状态
    public BDFaceSDKCommon.BDFaceGender gender;     // 性别

    public BDFaceSDKAttribute(int age, int race, int emotion, int glasses, int gender) {
        this.age = age;
        this.race = BDFaceSDKCommon.BDFaceRace.values()[race];
        this.emotion = BDFaceSDKCommon.BDFaceEmotion.values()[emotion];
        this.gender = BDFaceSDKCommon.BDFaceGender.values()[gender];
        this.glasses = BDFaceSDKCommon.BDFaceGlasses.values()[glasses];
    }
}

// 人脸情绪
public class BDFaceSDKEmotions {
    public BDFaceSDKCommon.BDFaceEmotionEnum emotion;
    public float expression_conf;
    public float[] expression_conf_list;

    public BDFaceSDKEmotions(int emotion, float expression_conf, float[] expression_conf_list) {
        this.emotion = BDFaceSDKCommon.BDFaceEmotionEnum.values()[emotion];
        this.expression_conf = expression_conf;
        this.expression_conf_list = expression_conf_list;
    }
}
    // 图像类型
    public enum BDFaceImageType {
        BD_FACE_IMAGE_TYPE_RGB,         // rgb图像
        BD_FACE_IMAGE_TYPE_BGR,         // bgr图像
        BD_FACE_IMAGE_TYPE_RGBA,        // rgba图像
        BD_FACE_IMAGE_TYPE_BGRA,        // bgra图像
        BD_FACE_IMAGE_TYPE_GRAY,        // 灰度图
        BD_FACE_IMAGE_TYPE_DEPTH,       // 深度图
        BD_FACE_IMAGE_TYPE_YUV422,      // YUV422图像
        BD_FACE_IMAGE_TYPE_YUV_411,     // YUV 411图像
        BD_FACE_IMAGE_TYPE_YUV_420,     // YUV 411图像
        BD_FACE_IMAGE_TYPE_YUV_YUYV,    // YUV YUYV图像
        BD_FACE_IMAGE_TYPE_YUV_YU12,    // YUV YU12图像
        BD_FACE_IMAGE_TYPE_YUV_NV12,    // YUV NV12图像
        BD_FACE_IMAGE_TYPE_YUV_NV21,    // YUV NV21图像
    }

    // 表情类型
    public enum BDFaceEmotion {
        BDFACE_EMOTION_NEUTRAL,         // 中性表情
        BDFACE_EMOTION_SMILE,           // 微笑
        BDFACE_EMOTION_BIG_SMILE,       // 大笑
    }

    // 情绪
    public enum BDFaceEmotionEnum {
        BDFACE_EMOTIONS_ANGRY,          // 生气
        BDFACE_EMOTIONS_DISGUST,        // 恶心
        BDFACE_EMOTIONS_FEAR,           // 害怕
        BDFACE_EMOTIONS_HAPPY,          // 开心
        BDFACE_EMOTIONS_SAD,            // 伤心
        BDFACE_EMOTIONS_SURPRISE,       // 惊讶
        BDFACE_EMOTIONS_NEUTRAL,        // 无情绪
    }
```

```
};

// 人脸属性种族
public enum BDFaceRace {
    BDFACE_RACE_YELLOW,     // 黄种人
    BDFACE_RACE_WHITE,      // 白种人
    BDFACE_RACE_BLACK,      // 黑种人
    BDFACE_RACE_INDIAN,     // 印度人
}

// 戴眼镜状态
public enum BDFaceGlasses {
    BDFACE_NO_GLASSES,  // 无眼镜
    BDFACE_GLASSES,     // 有眼镜
    BDFACE_SUN_GLASSES, // 墨镜
}

// 性别
public enum BDFaceGender {
    BDFACE_GENDER_FEMALE, // 女性
    BDFACE_GENDER_MALE,   // 男性
}
/**
 * log种类枚举
 */
public enum BDFaceLogInfo {
    BDFACE_LOG_VALUE_MESSAGE,   // 打印输出值日志
    BDFACE_LOG_ERROR_MESSAGE,   // 打印输出错误日志
    BDFACE_LOG_ALL_MESSAGE,     // 打印所有日志
};
```

Android-人脸通行工程

版本日志

版本	日期	说明
v2.0.2	2019.04.01	1、全新人脸检测模型，检测追踪更流畅； 2、部分接口细节优化 3、版本号与主线SDK对齐
v1.1.0	2019.03.14	1、增加几款结构光镜头支持；2、已知bug修复
V1.0.0	2019.01.10	人脸通行示例工程初版

1、方案介绍

1.1 方案概述

闸机/门禁是人脸识别技术的重点应用场景，人脸门禁/闸机在校园、景区、社区、展会等业务场景中应用广泛，为了便于开发者根据自己的场景化业务需求快速进行开发，提高人脸SDK/API的调用开发效率，我们基于场景化开发者对于人脸通行业务上的需求，为广大开发者提供基于业务层考虑的示例代码封装。您所需要的功能，都可以在我们提供的示例工程中快速找到该功能的实现模块并参考我们的实现方案进行开发使用。（注：本方案结合**人脸离线识别SDK**进行示例工程的开发）

> 人脸离线识别SDK包含人脸检测、活体检测、人脸比对等功能，SDK一经授权激活，可完全在无网络环境下工作，所有数据皆可在设备本地运行处理，可根据业务需求进行灵活的上层业务开发，核心能力分布如下图所示。

1.2 适用场景

适用场景特点

- **网络**：业务仅运行在局域网中，无法连接公有网络。如政府单位、金融保险、教育机构等。
- **安全**：处于业务安全考虑，人脸数据具有敏感性，离线局域网方案适用于数据安全要求较高的场景。
- **速度**：离线方案运行于离线嵌入式设备，处理性能相对服务器较弱，但网络延迟影响小。
- **稳定**：设备端运行离线人脸库，避免了网络抖动、机房故障带来的稳定性影响因素。

典型场景举例

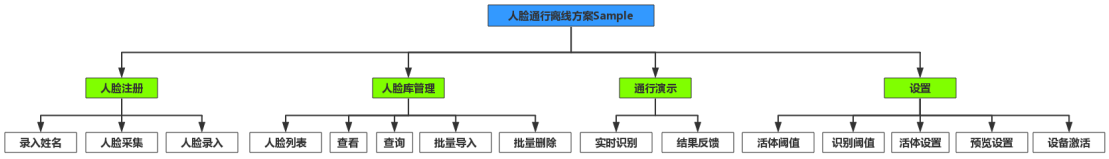
- **智慧校园**：使用人脸识别系统结合传统校园一卡通，实现公寓门禁、食堂计费、浴室扣费等一些列刷脸功能。
- **智慧票务**：将人脸识别系统与票务系统关联，使用人脸完成用户注册、购票、检票等票务全流程，适用于景区、剧场等。
- **智慧社区**：方案适用于智慧社区等小型安防场景，集成与社区监控系统中完成人流统计、人脸通行等核心功能。
- **企业管理**：方案可用于智能企业管理，可在此基础上根据业务需求实现人脸考勤、人脸门禁、访客签到等功能。
- **智慧酒店**：利用人脸识别能力辅助完成客人实名入住登记、人脸门锁、账户计费等功能。
- **.....**

1.3 名词解释

- **活体检测**：利用生物特征进行活体判断，这里主要利用人脸识别技术判断图像中是否为真人人脸，用户抵抗面具、高清图片等攻击。
- **阈值**： 阈值又叫临界值，是指一个效应能够产生的最低值或最高值，人脸活体和识别一般算法输出高于阈值才会判定人脸为活体或为同一个人。
- **嵌入式设备端**：嵌入式设备主要指配备在闸机中作为人脸分析前端的中小型设备（比服务器小），包括人脸抓拍机、人脸识别平板、人脸分析盒子等。
- **人脸服务端**：人脸识别服务器，分为百度云端识别服务器或本地私有化部署人脸服务器。
- **管理服务端**：对业务场景下的人脸库、通行记录等进行管理等终端。

1.4 功能介绍

本节主要介绍示例工程中的核心基础业务功能，如下图为示例工程的功能结构：

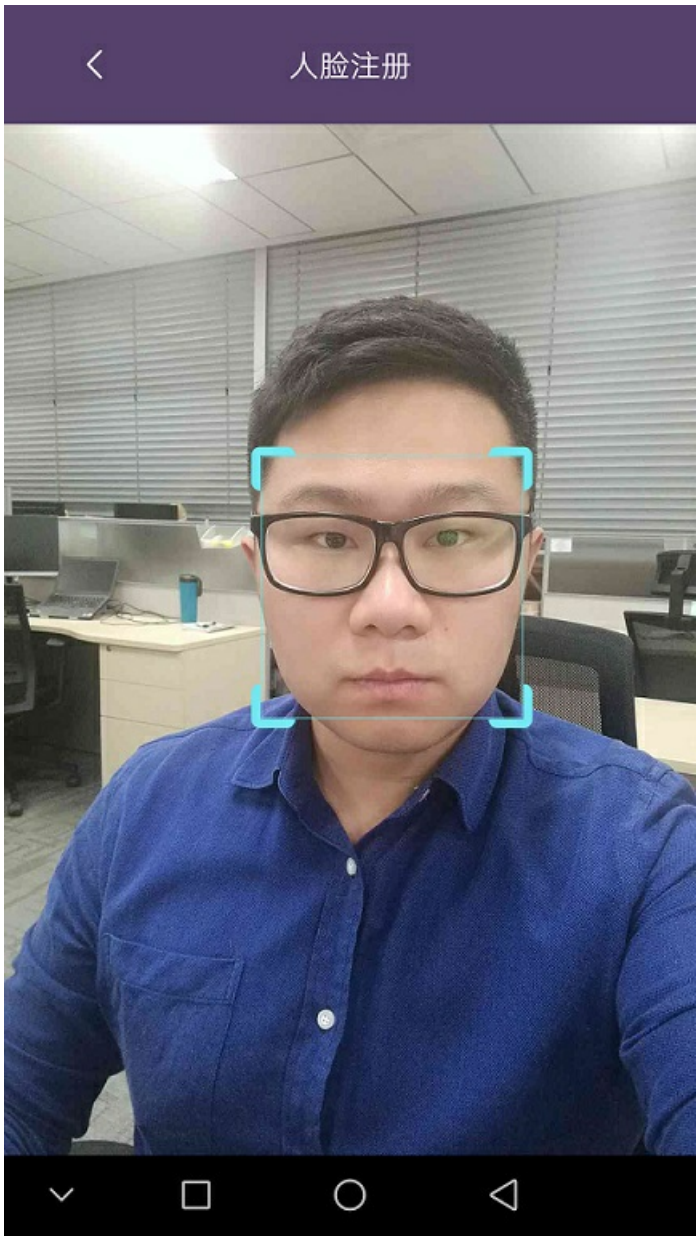


1.4.1 人脸注册

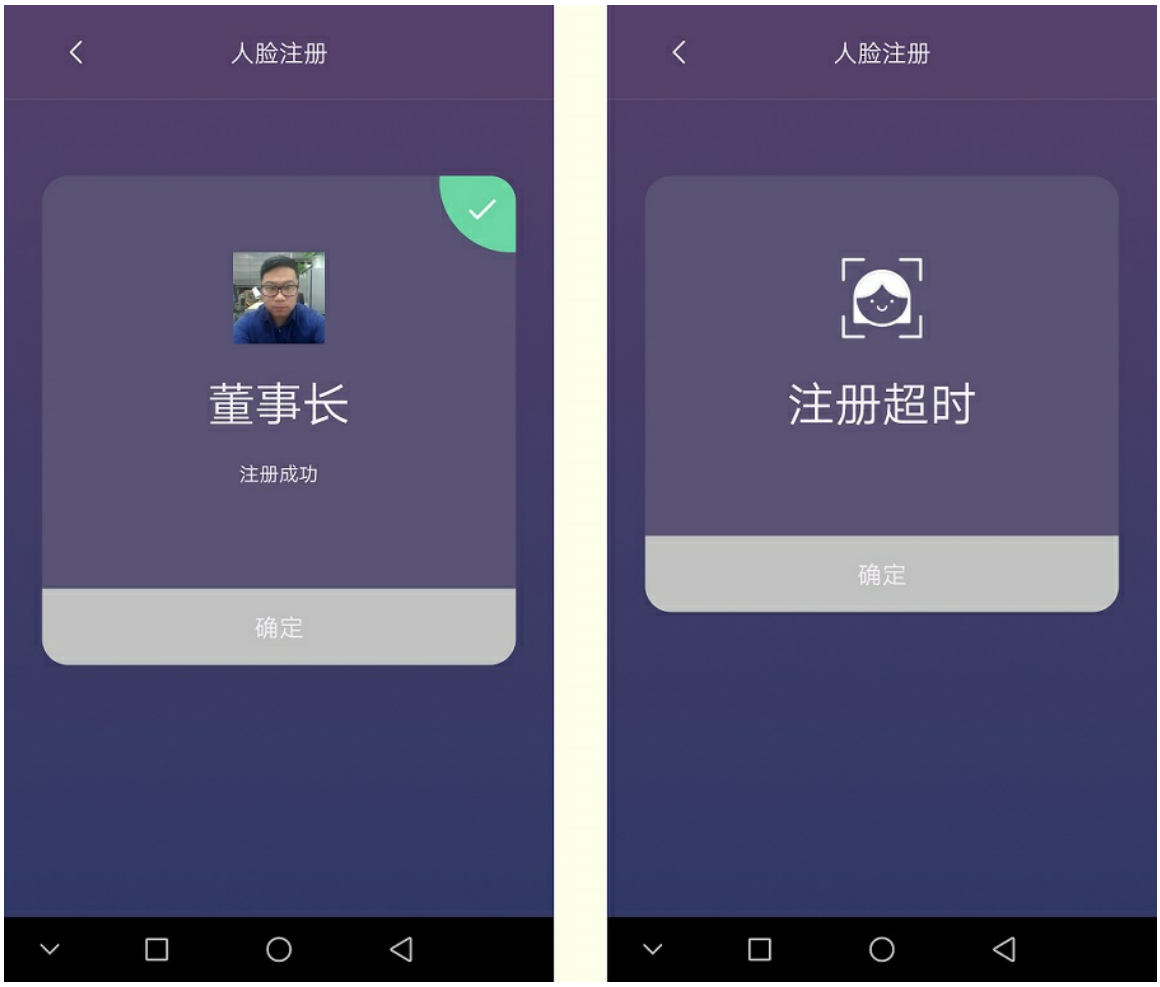
人脸注册即通过设备连接的摄像头抓取的人脸图像进行人脸检测以及特征提取将用户的人脸图片及特征值录入人脸库的过程：

- **录入姓名**：示例工程当前设计的人脸数据库存储结构为“姓名”+“人脸图片”，用户需要输入姓名并配合完成人脸抓拍方可完成人脸的录入，即人脸注册。第一步进行姓名的录入，点击“下一步”箭头图标进入下一步人脸采集。

- **人脸采集**：通过摄像头采集图像并通过算法自动完成对图片中人脸进行检测及特征提取，并会将人脸图像及对应人脸特征值存至数据库中，完成信息录入。



- **注册结果**：注册结果通知包括“注册成功”或“注册超时”信息。

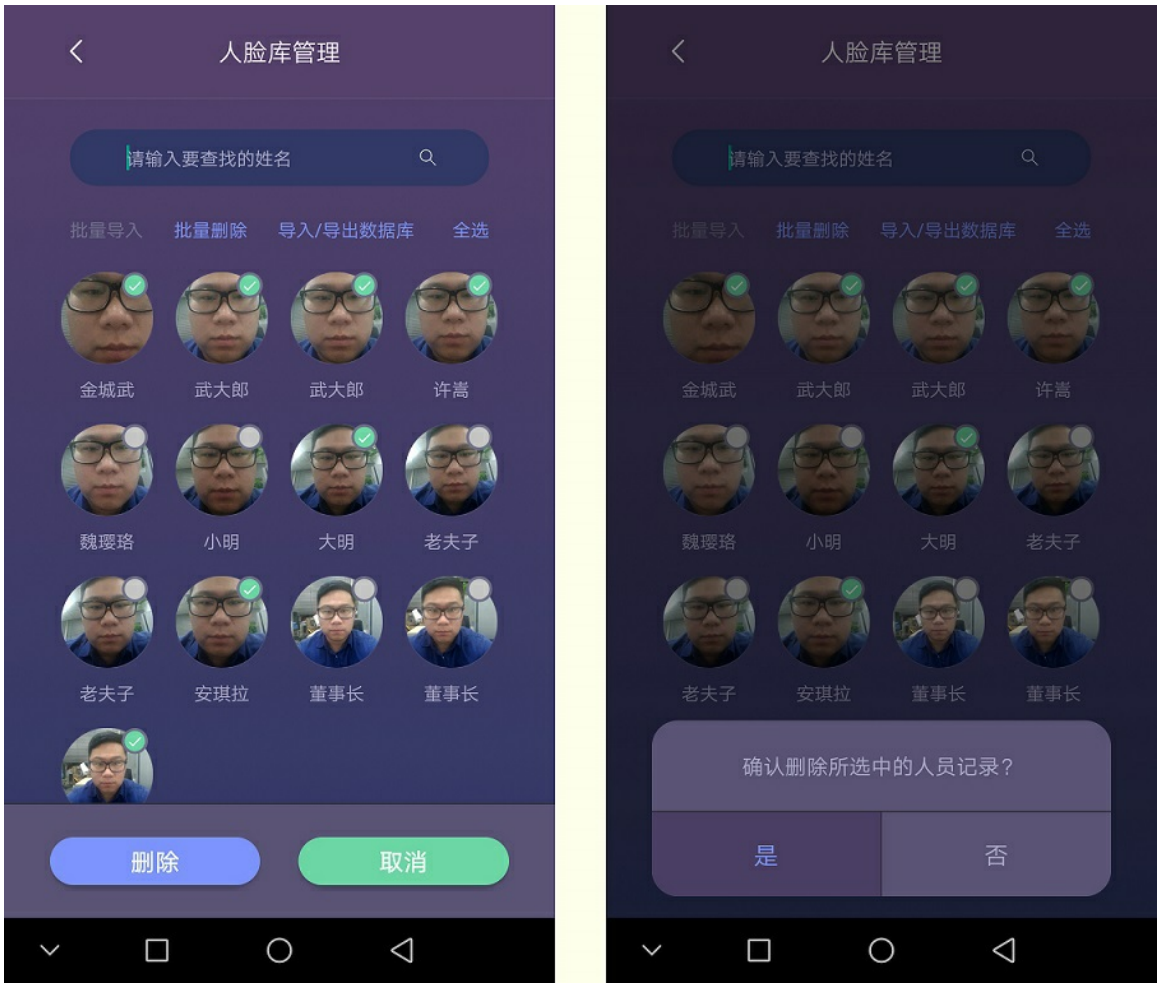


1.4.2 人脸库管理

“人脸库管理”功能演示模块包含人脸列表管理、注册图片查看、人员查询、批量导入（批量注册）、批量删除功能。

- **人脸列表**：列表展示用户通过“人脸注册”功能或“批量导入（批量注册）”功能录入到人脸库的姓名信息及头像切图。

- **注册图片查看**：点击上图中头像可查看原始注册图像。
- **人员查询**：在搜索框中输入人员姓名，可查询筛选出对应人员。（Sample中仅实现了全字段匹配检索，模糊检索开发者可根据产品或项目需求自行实现）
- **批量导入（批量注册）**：点击人脸列表上方的“批量导入”进入批量注册功能，可实现人脸批量注册，请按照示例工程APP文案提示进行操作。
- **批量删除**：人脸库管理支持数据的批量删除，点击人脸列表上方的“批量删除”后点选要删除的图片或点击“全选”选择全部图片。删除操作会将对应人员的姓名信息及图像信息全部删除。

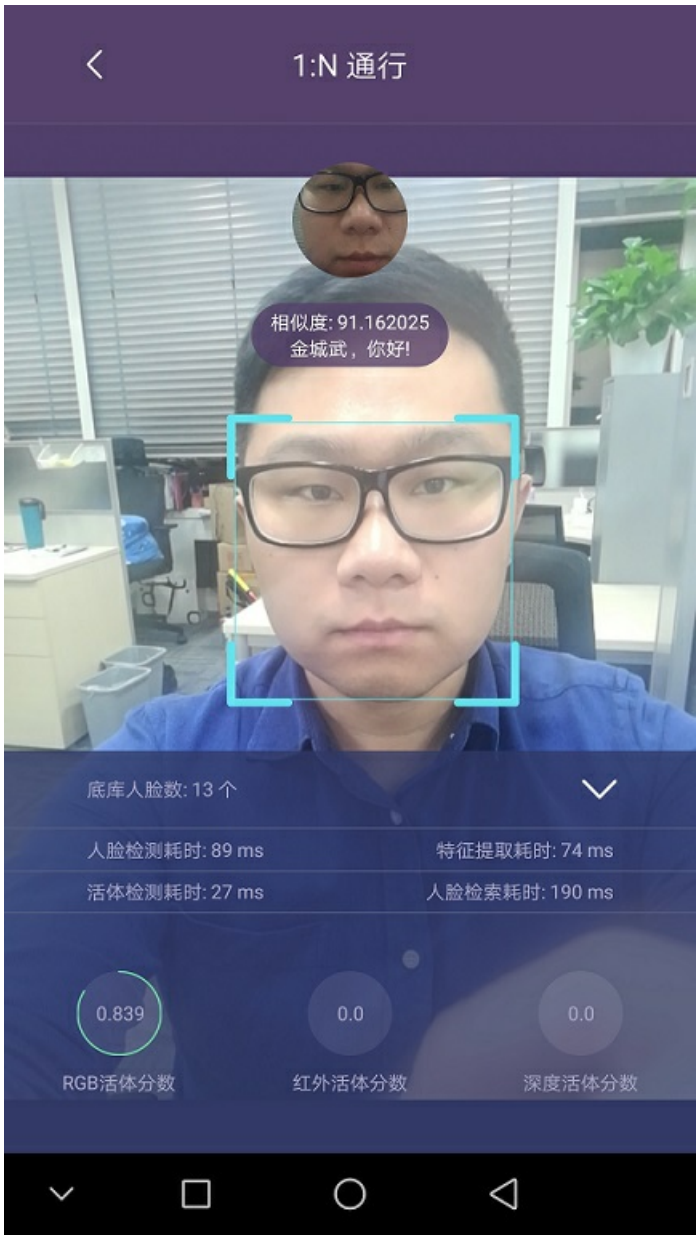


- **数据库导入/导出**：示例工程提供了方便开发者直接进行数据库导入导出操作的功能，以节省开发者在不同机器上进行大数据量底库检索耗时测试时进行批量注册的时间，用户可在一次批量注册后导出数据库以供其他机器测试使用。

1.4.3 通行演示

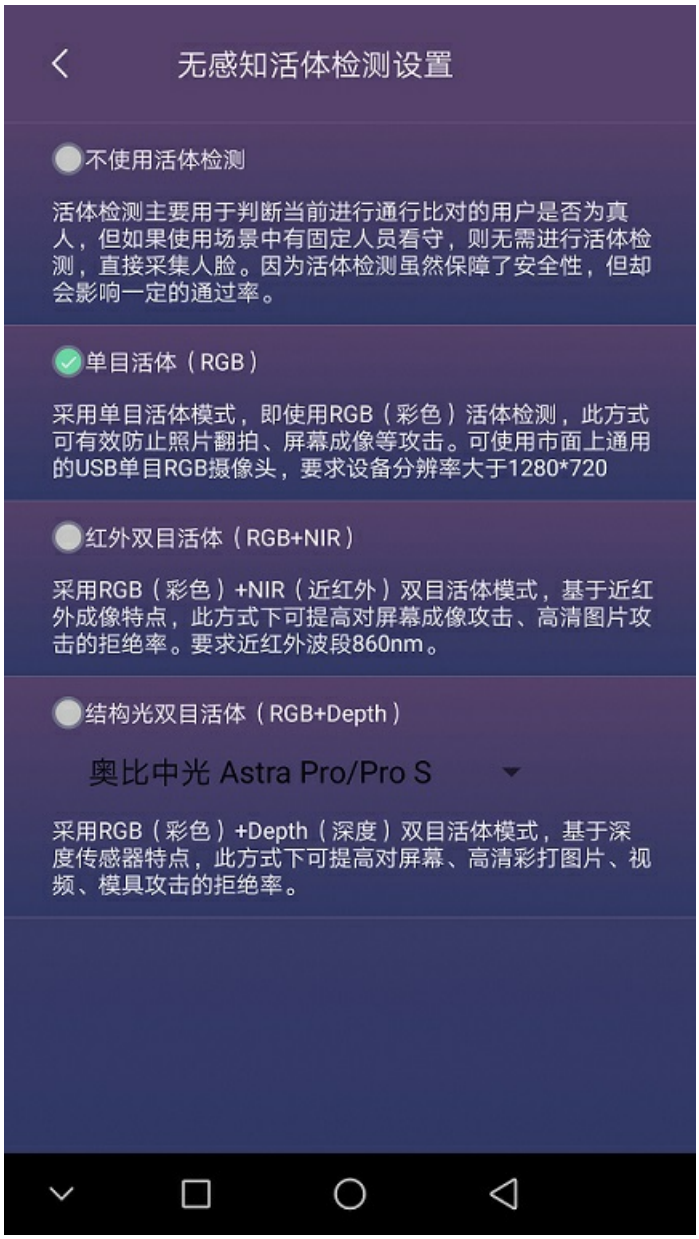
通行演示功能为示例工程重点功能，为百度官方提供的闸机/门禁通行场景的参考业务实现，支持“万”级别底库人像检索，企业开发者可参考示例工程代码逻辑对自己的项目进行功能开发。该功能支持外接单目摄像头、双目红外摄像头、双目结构光摄像头，以及对应的活体校验逻辑。为方便开发者测试当前耗时情况，我们在界面上为用户提供了如下信息：

- 底库人脸数
- 人脸检测耗时
- 特征提取耗时
- 活体检测耗时
- 人脸检索耗时（1:N检索耗时）
- RGB活体分数
- 红外活体分数
- 深度（结构光）活体分数



1.4.4 设置

- **无感知活体阈值设置**：调节阈值以控制活体攻击防范程度，包括“可见光活体阈值”、“红外活体阈值”、“深度活体阈值”三项设置。
- **人脸识别阈值设置**：针对应用场景来调节人脸识别阈值，人脸对比分数高于阈值则认为是同一个人。阈值设置越高则识别准确率越高，但会造成召回率下降，即本来为同一个人的两张图片被判定为非同一个人的情况会增多。建议开发者根据业务场景进行适当的实验以确定最佳阈值。
- **无感知活体检测设置**：我们为开发者提供了四种活体方案，包括“不使用活体检测”、“单目活体（RGB）”、“红外双目活体（RGB+NIR）”、“结构光双目活体（RGB+Depth）”，其中结构光活体由于需要加载驱动需要选择对应连接的摄像头型号。



- **摄像头预览旋转角度设置**：摄像头输出视频流的实际预览界面角度，一经设置，将会作用于回显画面上，用于纠正摄像头原本输出图像，避免出现图像中人头方向未朝上的情况，以确保能够检测到人脸。用户可通过查看“通行演示”模块中预览图像的实际展示情况做角度调整。

1.5 授权激活

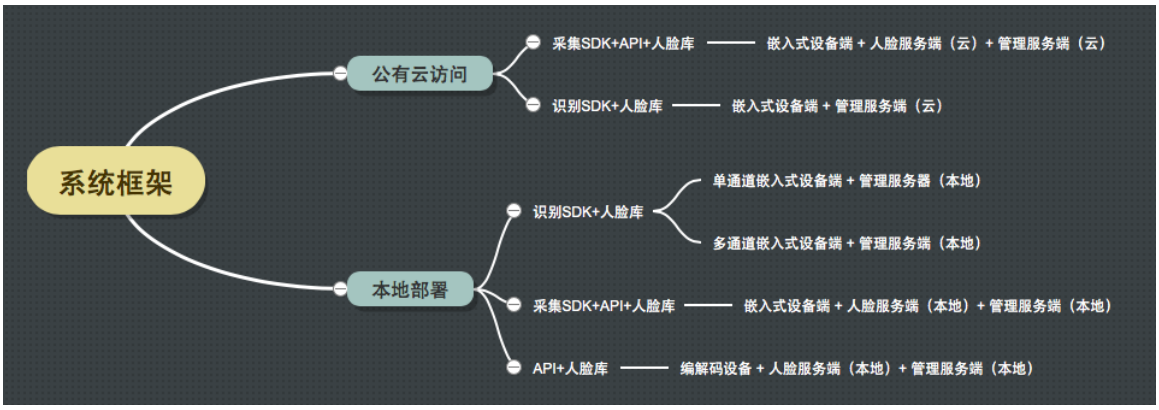
人脸通行演示方案示例工程对开发者免费提供上层应用代码，但由于目前离线识别SDK采用阶梯收费标准，并在用户成功申请离线识别SDK后发放一定数量的试用序列号，用户使用试用序列号对本示例工程中的人脸SDK进行授权及设备激活后方可进行相关测试或开发。SDK相关授权或定价可参考 [设备激活]

1.6 硬件选型

具体硬件设备选型请参考SDK技术文档中的 [设备选型]

1.7 方案组合

人脸闸机/门禁由于不同场景中业务需求存在一定差异，并且依据具体情况可能需要定制化开发，综合考虑人脸识别能力在服务器、嵌入式设备端不同的表现，我们梳理了如下几种不同的技术实现方案：

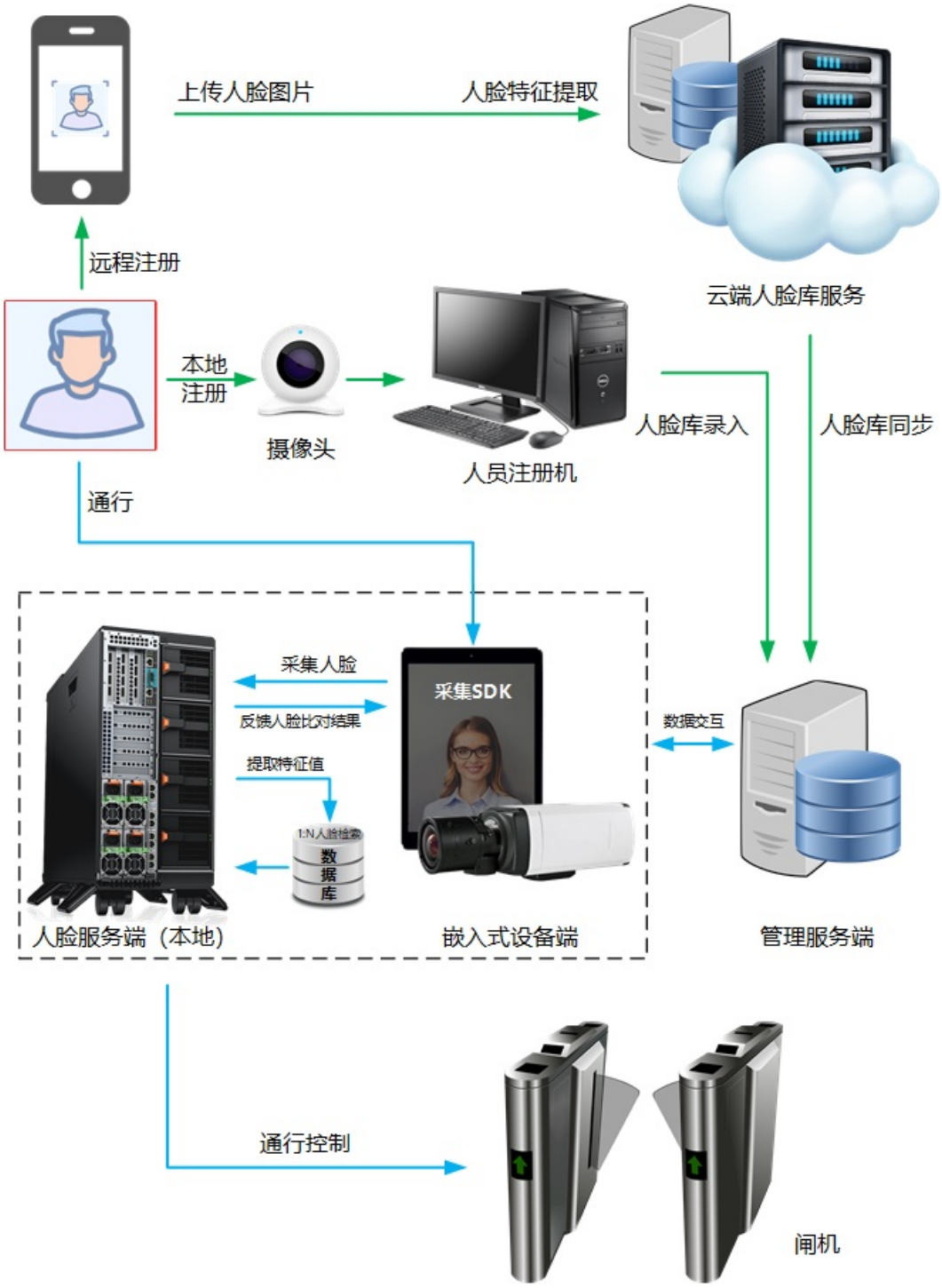


- **云端识别方案**：

- **嵌入式识别方案**：


上传人脸图片
人脸特征提取
远程注册
云端人脸库服务
本地
注册
摄像头
人员注册机
人脸库录入
人脸库同步
通行
识别SDK
通行
控制
数据交互
1:N人脸检索
提取
特征值
数
据
库
反馈人脸
比对结果
闸机
嵌入式设备端
管理服务端

- **私有化部署方案**：



2、集成指南

2.1 准备工作

2.1.1 注册开发者

1. STEP1：点击百度AI 开放平台导航右侧的控制台，页面跳转到百度云登录界面，登录完毕后，将会进入到百度云后台，点击「控制台」进入百度云控制台页面；您也可以在官网直接点击免费试用，登录完毕后将自动进入到百度云控制台。
2. STEP2：使用百度账号完成登录，如您还未持有百度账户，可以点击此处注册百度账户。
3. STEP3：进入百度云欢迎页面，填写企业和个人基本信息，注册完毕，至此成为开发者。(如您之前已经是百度云用户或百度开发者中心用户，STEP3 可略过。)
4. STEP4：进入百度云控制台，找到人工智能相关服务面板。
5. STEP5：点击进入「人脸识别」

6. STEP6：点击进入「离线SDK管理」

2.1.2 设备激活

首次运行示例工程，会显示“设备激活”页面，激活方式分别是：在线激活、离线激活。

（1）在线激活：

1为设备指纹，自动读取；2为序列号，手动输入。点击激活（激活需要联网，采用的https请求，https要求设备系统时间跟请求服务器时间差距不大，否则请求授权服务器会失败），激活以后使用不需要在连接网络，一个序列号绑定一台设备，卸载应用后重新安装可能需要重新激活，可以使用同一个序列号。可以在你的控制台查看设备号对应的序列号。在线激活界面如下图所示：

（2）离线激活：

离线激活需要登录百度云控制台，根据设备指纹，下载离线激活文件License.zip，将该激活文件保存在手机的SD卡根目录下，点击“离线激活”，SDK会自动进行激活处理。

2.2 代码结构

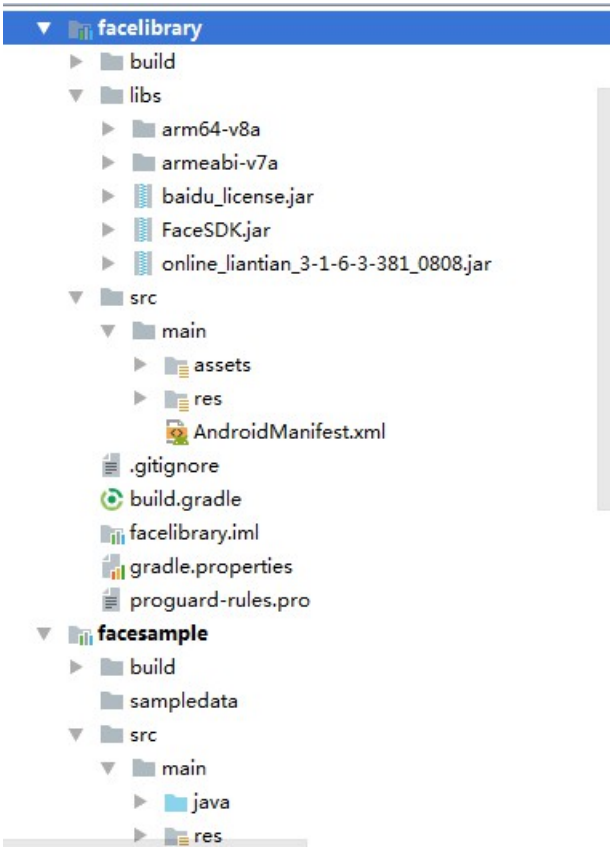
2.2.1 核心库介绍

faceLibrary库主要用于存放SDK相关的动态库以及模型文件。

-- lib目录为动态库so和jar包

-- assets目录为模型文件

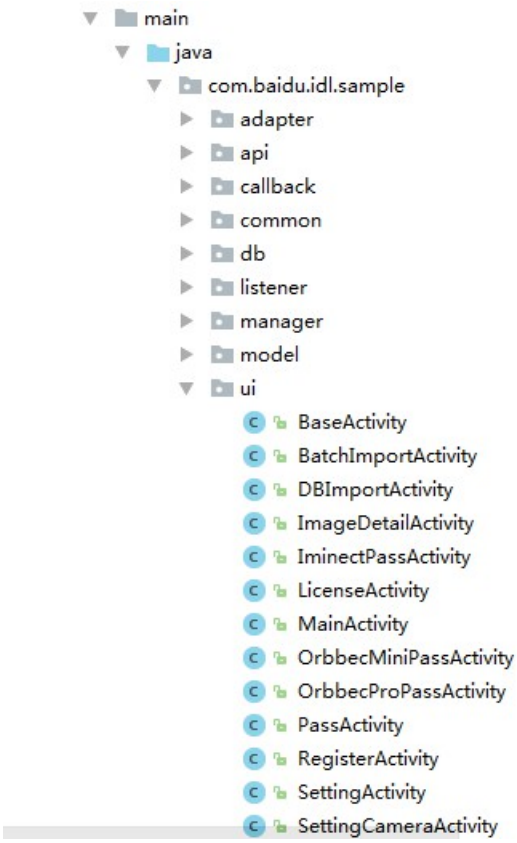
facesample主要是java代码，主要包含用户组管理、人脸SDK操作、视频流、图片等操作辅助类。



2.2.2 示例代码介绍

参数名	含义
BatchImportActivity	提供批量注册人脸到人脸库。
DBImportActivity	数据库文件的导入导出功能。
ImageDetailActivity	用于查看图片。
IminectPassActivity	带RGB+Depth活体，视频流人脸实时检测，并和某个分组的人脸库进行1：n比对。需要使用RGB+Depth的双目摄像头。如华捷艾米双目摄像头。
LicenseActivity	用于初始化鉴权操作。
OrbbecMiniPassActivity	带RGB+Depth活体，视频流人脸实时检测，并和某个分组的人脸库进行1：n比对。需要使用RGB+Depth的双目摄像头。如奥比中光mini双目摄像头。
OrbbecProPassActivity	带RGB+Depth活体，视频流人脸实时检测，并和某个分组的人脸库进行1：n比对。需要使用RGB+Depth的双目摄像头。如奥比中光Pro双目摄像头。
PassActivity	手机RGB或外接RGB活体或带RGB+NIR活体，视频流人脸实时检测，并和某个分组的人脸库进行1：n比对。需要使用RGB+NIR的双目摄像头。
RegisterActivity	用于进行人脸注册。
SettingActivity	设置功能，用于摄像头配置以及SDK检测参数设置。
SettingCameraActivity	无感知检测活体设置，用于设置是否检测活体，以及设置采用单目、双目活体。
SettingCameraPreviewAngleActivity	摄像头预览旋转角度设置，用于调整预览界面人脸是否摆正。
SettingDetectActivity	人脸质量检测阈值设置。
SettingFeatureActivity	人脸识别阈值设置。
SettingLiveActivity	无感知活体阈值设置。
UserActivity	用户管理，主要用于用户的导入、查找、删除。

以上所述文件位置如下图所示：



2.3 开始集成

接下来我们详细介绍以下集成步骤。

2.3.1 人脸库集成

1、把facelibrary库下的lib、asset目录下的so库、jar包以及模型文件添加到自己的工程中。

2、根据需要把app里面的示例代码添加到自己的工程。

2.3.2 按设备授权（授权过程中对模型进行初始化）

（1）在线激活：

```
 FaceAuth faceAuth = new FaceAuth();
faceAuth.initLicenseOnLine(this, key, new AuthCallback() {
        @Override
        public void onResponse(final int code, final String response, String licenseKey) {
            if (code == 0) {
                // 如果激活成功
                // 初始化模型
                FaceSDKManager.getInstance().initModel(mContext);
                // 初始化数据库
                DBManager.getInstance().init(getApplicationContext());
                finish();
            } else {
                ToastUtils.toast(mContext, code + "  " + response);
            }
        }
    });
```

初始化模型的相关代码如下：

```
 public void initModel(final Context context) {
// 加载人脸检测模型
   faceDetector.initModel(context, "enc_detect_vgg_anakin.lite.bin",
           "small_detect.model",
           "enc_net_deploy_stageI.anakin.bin", new FaceCallback() {
               @Override
               public void onResponse(int code, String response) {
                   ToastUtils.toast(context, code + "  " + response);
               }
           });
// 设定人脸检测配置参数，该方法采取默认参数
   faceDetector.loadConfig(getFaceEnvironmentConfig());
// 加载特征提取模型
   faceFeature.initModel(context, "enc_sphere_resnet34_128_580_pytorch_anakin.bin",
           "enc_fsp_fc128_160_pytorch_anakin.bin",
           "", new FaceCallback() {
               @Override
               public void onResponse(int code, String response) {
                   ToastUtils.toast(context, code + "  " + response);
                   ;
               }
           });
// 加载活体检测模型
   faceLiveness.initModel(context, "enc_model_rgb_145_anakin.bin",
           "enc_nir_anakin.lite.bin",
           "enc_depth_anakin.lite.bin", new FaceCallback() {
               @Override
               public void onResponse(int code, String response) {
                   ToastUtils.toast(context, code + "  " + response);
               }
           });
}
```

如果需要设置SDK的具体参数：

```
 FaceEnvironment faceEnvironment = new FaceEnvironment();
faceEnvironment.setMinFaceSize(50);
faceEnvironment.setMaxFaceSize(-1);
faceEnvironment.setDetectInterval(200);
faceEnvironment.setTrackInterval(500);
faceEnvironment.setNoFaceSize(0.5f);
faceEnvironment.setPitch(30);
faceEnvironment.setYaw(30);
faceEnvironment.setRoll(30);
faceEnvironment.setCheckBlur(true);
faceEnvironment.setOcclusion(true);
faceEnvironment.setIllumination(true);
faceEnvironment.setDetectMethodType(FaceDetect.DetectType.DETECT_VIS);
```

详细参数设置参考如下表格所示：

参数	名称	默认值
minFaceSize	检测最小人脸值	50
maxFaceSize	检测最大人脸值	-1（不做限制）
trackInterval	人脸跟踪，检测的时间间隔	500
detectInterval	人脸跟踪，跟踪时间间隔	1000
noFaceSize	非人脸阈值	0.5
pitch	抬头低头角度	15
yaw	左右摇头角度	15
yaw	左右摇头角度	15
roll	偏头角度	15
isCheckBlur	是否进行模糊检测	false
isIllumination	是否进行光照检测	false
isOcclusion	是否进行遮挡检测	false
detectMethodType	图片检测类型	VIS

（2）离线激活

相关代码如下：

```
faceAuth.initLicenseOffLine(this, new AuthCallback() {
        @Override
        public void onResponse(final int code, final String response, String licenseKey) {
            if (code == 0) {
                GlobalSet.FACE_AUTH_STATUS = 0;
                // 初始化人脸
                FaceSDKManager.getInstance().initModel(mContext);
                // 初始化数据库
                DBManager.getInstance().init(getApplicationContext());
                finish();
            } else {
                ToastUtils.toast(mContext, code + "  " + response);
            }
        }
    });
```

2.3.3 人脸注册（RegisterActivity）

进入该功能之后首先需要录入注册人的姓名，姓名要求不能含有特殊符号，要求不能超过10个字符，填入成功之后才可进入视频流实时采集页面。在采集页面如果30s未检测到人脸，则显示人脸注册超时。采集页面将会执行人脸检测、人脸活体检测、特征值提取功能。当返回正常时，说明注册成功。并将注册成功的图片保存到本地。

下面是活体检测相关代码：

```
FaceInfo[] faceInfos = null;
  if (mRgbArray != null) {
// 进行人脸检测
      faceInfos = FaceSDKManager.getInstance().getFaceDetector().trackMaxFace(mRgbArray, width, height);
  }
// 保存相关图片信息
  LivenessModel livenessModel = new LivenessModel();
  livenessModel.setRgbDetectDuration(System.currentTimeMillis() - startTime);
  livenessModel.getImageFrame().setArgb(mRgbArray);
  livenessModel.getImageFrame().setWidth(width);
  livenessModel.getImageFrame().setHeight(height);
  livenessModel.setLiveType(type);

  if (faceInfos != null && faceInfos.length > 0) {
      livenessModel.setTrackFaceInfo(faceInfos);
      FaceInfo faceInfo = faceInfos[0];
      livenessModel.setFaceInfo(faceInfo);
      livenessModel.setLandmarks(faceInfo.landmarks);

      // 返回检测到人脸的信息，用于画人脸框
      if (faceDetectCallBack != null) {
          faceDetectCallBack.onFaceDetectCallback(true, (int) faceInfo.mWidth,
                  (int) faceInfo.mWidth, (int) faceInfo.mCenter_x, (int) faceInfo.mCenter_y,
                  width, height);
      }

      if (livenessCallBack != null) {
          livenessCallBack.onTip(1, "活体判断中");
      }

      Log.e("lth_id",""+curFaceID+"   "+livenessModel.getFaceInfo().face_id);

// 活体检测，获取活体分值
      float rgbScore = 0;
      if ((type & MASK_RGB) == MASK_RGB) {
          startTime = System.currentTimeMillis();
          rgbScore = rgbLiveness(mRgbArray, width, height, faceInfos[0].landmarks);
          livenessModel.setRgbLivenessScore(rgbScore);
          livenessModel.setRgbLivenessDuration(System.currentTimeMillis() - startTime);
      }
      ......
  }
```

下面是人脸特征提取，并将相关信息保存起来的过程：

```
public void registFace(LivenessModel livenessModel) {
// 人脸特征提取
  byte[] bytes = new byte[512];
  float length = FaceSDKManager.getInstance().getFaceFeature().extractFeature(
          livenessModel.getImageFrame().getArgb(),
          livenessModel.getImageFrame().getHeight(),
          livenessModel.getImageFrame().getWidth(),
          bytes,
          livenessModel.getLandmarks());
// 如果特征值提取正常
  if (length == 128) {
      Feature feature = new Feature();
      feature.setCtime(System.currentTimeMillis());
      feature.setFeature(bytes);
      feature.setUserName(registNickName);
      final String uid = UUID.randomUUID().toString();
```

```
        feature.setUserId(uid);
        feature.setGroupId("0");

        int imgWidth = livenessModel.getImageFrame().getWidth();
        int imgHeight = livenessModel.getImageFrame().getHeight();
        Bitmap registBmp = registBmp = Bitmap.createBitmap(imgWidth,
                imgHeight, Bitmap.Config.ARGB_8888);
        registBmp.setPixels(livenessModel.getImageFrame().getArgb(), 0, imgWidth, 0, 0, imgWidth, imgHeight);

        // 保存图片到新目录中
        File facePicDir = FileUtils.getFacePicDirectory();
        // 保存抠图图片到新目录中
        File faceCropDir = FileUtils.getFaceCropPicDirectory();
        String picFile = "regist_" + uid + "_rgb.png";

        if (facePicDir != null) {
            File savePicPath = new File(facePicDir, picFile);
            if (FileUtils.saveFile(savePicPath, registBmp)) {
                Log.i(TAG, "图片保存成功");
                feature.setImageName(picFile);
            }
        }

        Bitmap cropBitmap = null;
        String cropImgName = null;
        // 人脸抠图
        FaceInfo faceInfo = livenessModel.getFaceInfo();
        if (faceInfo != null) {
            cropBitmap = ImageUtils.noBlackBoundImgCrop(faceInfo.landmarks,
                    livenessModel.getImageFrame().getHeight(), livenessModel.getImageFrame().getWidth(),
                    livenessModel.getImageFrame().getArgb());

            if (cropBitmap == null) {
                cropBitmap = registBmp;
            }

            cropImgName = "crop_" + picFile;
        }
        if (faceCropDir != null && cropBitmap != null) {
            File saveCropPath = new File(faceCropDir, cropImgName);
            if (FileUtils.saveFile(saveCropPath, cropBitmap)) {
                Log.i(TAG, "抠图图片保存成功");
                feature.setCropImageName(cropImgName);
            }
        }

      // 将人脸信息保存数据库
        if (FaceApi.getInstance().featureAdd(feature)) {
     // 注册成功，返回结果
            returnRegistResult(0, livenessModel, cropBitmap);
        }
        return;
    }
}
```

2.3.4 通行演示（1:N）

1:N 通行主要是通过视频流的形式，将摄像头采集到的人脸与人脸库中的N张人脸图片进行特征值比对。如果比对成功，则允许通行。

您可以根据实际硬件选择活体策略，有下面几种实现：

- PassActivity 无活体或rgb活体或rgb+ir活体
- OrbbecProPassActivity rgb+depth活体
- OrbbecMiniPassActivity rgb+depth活体
- IminectPassActivity rgb+depth活体

以PassActivity实现为例

1）跟据是否是红外摄像头选择相关view

```
 // 如果是NIR摄像头，则用BinocularView加载
if (GlobalSet.getLiveStatusValue() == GlobalSet.LIVE_STATUS.RGN_NIR) {
        mBinocularView = new BinocularView(mContext);
        mBinocularView.setImageView(mImageTrack);
        mBinocularView.setLivenessCallBack(this);
        mCameraView.addView(mBinocularView, layoutParams);
} else {  // 如果是RGB摄像头，则用MonocularView加载
        mMonocularView = new MonocularView(mContext);
        mMonocularView.setImageView(mImageTrack);
        mMonocularView.setLivenessCallBack(this);
        mCameraView.addView(mMonocularView, layoutParams);
}
```

2）以RGB为例，在MonocularView中实现人脸检测、活体检测、特征值对比功能

```
 // 开始调用人脸相关接口
private synchronized void checkData() {
    if (rgbData != null) {
        FaceSDKManager.getInstance().getFaceLiveness().setNirRgbInt(null);
        FaceSDKManager.getInstance().getFaceLiveness().setRgbInt(rgbData);
        FaceSDKManager.getInstance().getFaceLiveness().setIrData(null);
        FaceSDKManager.getInstance().getFaceLiveness().livenessCheck(mTrackImageWidth, mTrackImageHeight,
                GlobalSet.getLiveStatusValue() == GlobalSet.LIVE_STATUS.RGB ? 0X0001 : 0X0000);
        rgbData = null;
    }
}

// 实现人脸检测、活体检测
private boolean onLivenessCheck(int width, int height, int type) {
    boolean isLiveness = false;

    long startTime = System.currentTimeMillis();
    FaceInfo[] faceInfos = null;
    if (mRgbArray != null) {
  // 人脸检测，获取人脸信息
        faceInfos = FaceSDKManager.getInstance().getFaceDetector().trackMaxFace(mRgbArray, width, height);
    }
    LivenessModel livenessModel = new LivenessModel();
    livenessModel.setRgbDetectDuration(System.currentTimeMillis() - startTime);
    livenessModel.getImageFrame().setArgb(mRgbArray);
    livenessModel.getImageFrame().setWidth(width);
    livenessModel.getImageFrame().setHeight(height);
    livenessModel.setLiveType(type);

    if (faceInfos != null && faceInfos.length > 0
            && faceInfos[0].mConf >= GlobalSet.getDetectConfValue()) {
        livenessModel.setTrackFaceInfo(faceInfos);
        FaceInfo faceInfo = faceInfos[0];
        livenessModel.setFaceInfo(faceInfo);
```

```
        livenessModel.setLandmarks(faceInfo.landmarks);

        // 返回检测到人脸的事件
        if (faceDetectCallBack != null) {
            faceDetectCallBack.onFaceDetectCallback(true, (int) faceInfo.mWidth,
                    (int) faceInfo.mWidth, (int) faceInfo.mCenter_x, (int) faceInfo.mCenter_y,
                    width, height);
        }

        if (livenessCallBack != null) {
            livenessCallBack.onTip(1, "活体判断中");
        }

        if (currentTaskType == TASK_TYPE_ONETON && livenessModel.getFaceInfo().face_id == curFaceID) {
            return true;
        }

        float rgbScore = 0;
        if ((type & MASK_RGB) == MASK_RGB) {
            startTime = System.currentTimeMillis();
    // 活体检测
            rgbScore = rgbLiveness(mRgbArray, width, height, faceInfos[0].landmarks);
            livenessModel.setRgbLivenessScore(rgbScore);
            livenessModel.setRgbLivenessDuration(System.currentTimeMillis() - startTime);
        }

        if (livenessCallBack != null) {
            switch (currentTaskType) {
                case TASK_TYPE_ONETON:
                    filterFeature(livenessModel);
                    break;
                default:
                    break;
            }

        }
    } else {
        // 返回检测到人脸的事件
        if (faceDetectCallBack != null) {
            faceDetectCallBack.onFaceDetectCallback(false, 0,
                    0, 0, 0, 0, 0);
        }
        if (livenessCallBack != null) {
            livenessCallBack.onCallback(1, null);
            livenessCallBack.onTip(1, "未检测到人脸");
        }
    }
    livenessModel.setTrackFaceInfo(faceInfos);
    if (livenessCallBack != null) {
        livenessCallBack.onCanvasRectCallback(livenessModel);
    }
    return isLiveness;
}

// 特征值提取
public boolean onFeatureCheck(LivenessModel livenessModel) {
    if ((GlobalSet.getLiveStatusValue() == GlobalSet.LIVE_STATUS.NO) ||
            (GlobalSet.getLiveStatusValue() == GlobalSet.LIVE_STATUS.RGB
                    && livenessModel.getRgbLivenessScore() > GlobalSet.getLiveRgbValue())) {
        byte[] visFeature = new byte[512];
        long sTime = System.currentTimeMillis();
    // 特征值提取
        float length = FaceSDKManager.getInstance().getFaceFeature().extractFeature(
```

```
        float length = FaceSDKManager.getInstance().getFaceFeature().extractFeature(
                livenessModel.getImageFrame().getArgb(),
                livenessModel.getImageFrame().getHeight(),
                livenessModel.getImageFrame().getWidth(),
                visFeature,
                livenessModel.getLandmarks());
        livenessModel.setFeatureDuration(System.currentTimeMillis() - sTime);
        if (length == 128) {
            livenessModel.setFeatureByte(visFeature);
            return true;
        }
    }
    return false;
}
```

```
 // 将拿到的特征进行比对，并将结果返回
public void filterFeature(LivenessModel livenessModel) {
    if (onFeatureCheck(livenessModel)) {
        long sTime = System.currentTimeMillis();
 // 方法内部实现比对，并将结果存入数据库
        Feature feature = FaceSDKManager.getInstance().getFeature(
                FaceFeature.FeatureType.FEATURE_VIS, livenessModel.getFeatureByte(), livenessModel);
        livenessModel.setCheckDuration(System.currentTimeMillis() - sTime);
        Log.e("lth_sc", "" + livenessModel.getCheckDuration());
        if (feature != null) {
            curFaceID = livenessModel.getFaceInfo().face_id;
            livenessModel.setFeature(feature);
            livenessCallBack.onCallback(0, livenessModel);
            return;
        }
    }
    livenessCallBack.onCallback(1, livenessModel);
}
```

3）摄像头配置

```
 public void onResume() {
   if (mCameraNum == 0) {
       Toast.makeText(context, "未检测到摄像头", Toast.LENGTH_LONG).show();
       return;
} else {
       try {
           Camera.CameraInfo info = new Camera.CameraInfo();
           int numCameras = Camera.getNumberOfCameras();
           for (int i = 0; i < numCameras; i++) {
  // 获取摄像头信息，判断是前置摄像头还是后置摄像头
               Camera.getCameraInfo(i, info);
               if (info.facing == Camera.CameraInfo.CAMERA_FACING_FRONT) {
                   mCamera = Camera.open(i);
                   break;
               }
           }
           // 如果没有前置摄像头，则打开默认的后置摄像头
           if (mCamera == null) {
               mCamera = Camera.open(0);
           }
           // 没有摄像头时，抛出异常
           if (mCamera == null) {
               throw new RuntimeException("Unable to open camera");
           }

  // 设置摄像头预览角度
           mCamera.setDisplayOrientation(previewAngle);
           mPreview.setCamera(mCamera, PREFER_WIDTH, PERFER_HEIGH);
           mCamera.setPreviewCallback(new Camera.PreviewCallback() {
               @Override
               public void onPreviewFrame(byte[] data, Camera camera) {
                   dealRgb(data);
               }
           });
       } catch (RuntimeException e) {
           // Log.e(TAG, e.getMessage());
       }
   }

   if (mTextureViewOne != null) {
       mTextureViewOne.setOpaque(false);
   }
}
```

2.3.5 人脸库管理

1）查找功能

通过姓名查找库中对应的此人照片

```
 // 搜索按钮
if (v == mImageSearch) {
   String searchContent = mEditContent.getText().toString().trim();
   // 根据关键字查找人脸库
   UserInfoManager.getInstance().getFeatureInfo(searchContent, mUserInfoListener);
}
```

2）批量导入

本功能下支持批量导入人脸图片数据，用于将图片录入到人脸库中，即批量注册。该过程不进行活体检测，仅执行批量人脸特

征值生成录入，每张图片提取一张最大人脸做为注册人脸，人脸图片较多时可能会耗时较长，请耐心等待！

```
 // 检测人脸，提取人脸特征值
ret = FaceApi.getInstance().getFeature(argbImg, bytes,
        FaceFeature.FeatureType.FEATURE_VIS, environment, faceInfos);
FaceInfo faceInfo = faceInfos[0];
Log.e(TAG, "live_photo = " + ret);
```

PC通过USB连接本机，找到本机跟目录，该目录下"Import Faces"目录，该目录下包含"Faces List.txt" 及 "FacePictures"目录。请不要对此目录进行更改。批量导入时会判断此目录：

```
 // 获取导入图片目录 /sdcard/Import Faces/FacePictures
File batchPicDir = FileUtils.getBatchFacePicDirectory();
String[] picFiles = batchPicDir.list();
if (picFiles == null || picFiles.length == 0) {
    if (mImportListener != null) {
        mImportListener.showToastMessage("导入图片的文件夹没有图片");
    }
    return;
}
```

"FacePictures"目录下放人脸注册图片，可通过PC拷贝放入。在"Faces List.txt" 中如下将姓名与相对应图片名称按照对应格式填入或自行编写脚本写入

姓名	图片名
刘德华	刘德华.jpg
岳云鹏	岳云鹏.jpg
王大锤	WDC.png

如导入时库中已有相同则会过滤此条导入(本程序安装后自带100张图片的批量导入示例，可直接点击体验)，准备就绪，开始导入吧！

```
 // 根据姓名查询数据库与文件中对应的姓名是否相等，如果相等，则直接过滤
List<Feature> listFeatures = DBManager.getInstance().queryFeatureByName(userName);
if (listFeatures != null && listFeatures.size() > 0) {
    String msg = "与之前图片对应的姓名相同";
    // 保存失败信息
    logBuilder.append(userName + "\t" + picFile + "\t"
            + "失败" + "\t" + msg + "\n");
```

批量导入过程中如有特征未能提取，检测失败的图片，请查看日志信息，如：

```
 Log.e(TAG, picFile + "未检测到人脸，可能原因：人脸太小");
String msg = "未检测到人脸，可能原因：人脸太小"
        + "（必须大于最小检测人脸minFaceSize），"
        + "或者人脸角度太大，人脸不是朝上";
logBuilder.append(userName + "\t" + picFile + "\t"
        + "失败" + "\t" + msg + "\n");
```

3）批量删除

批量删除可自由选择删除任意数据，或者全选所有。

```
 // 从数据库中删除
UserInfoManager.getInstance().batchRemoveFeatureInfo(mListFeatureInfo, mUserInfoListener,
        mSelectCount);
```

4）导入/导出数据库

本功能支持将批量导入人脸图片数据从手机内存导出到SD卡中，或者将SD卡的数据文件导入到手机内存当中。如下：其中outputDBSuccess()为导出监听，importDBSuccess()为导入监听,showErrMsg()为异常监听。

```
 // 复制数据库到内存
public void copyDBFileToData() {
    String sqlPath = "/sdcard/bdface/import/bdface.db";
    String newPath = "/data/data/com.baidu.idl.face.demo/databases/bdface.db";
    if (FileUtils.copyFile(sqlPath, newPath)){
        if (mDBFileListener != null) {
            mDBFileListener.importDBSuccess();
        }
    } else {
        if (mDBFileListener != null) {
            mDBFileListener.showErrMsg();
        }
    }
}
```

```
 // 复制数据库到SD卡
public void copyDBFileToSDCard() {
    String sqlPath = "/data/data/com.baidu.idl.face.demo/databases/bdface.db";
    String newPath = "/sdcard/bdface/output/bdface.db";
    if (FileUtils.copyFile(sqlPath, newPath)){
        if (mDBFileListener != null) {
            mDBFileListener.outputDBSuccess();
        }
    } else {
        if (mDBFileListener != null) {
            mDBFileListener.showErrMsg();
        }
    }
}
```

（1）导入数据库：将数据库文件放置SD卡bdface/import目录下，并命名为bdface.db，点击"导入数据库"按钮，完成导入功能。

（2）导出数据库：点击"导出数据库"按钮，数据库文件(bdface.db)将会导出到bdface/outout目录下。

2.3.6 设置

此功能中所有改动都是将值存入SP（PreferencesUtil），等待SDK中拿到的值来进行比较。

与设置的值相对应的key在 GlobalSet 类中列出。如：

```
 // 质量参数阈值类型
public static final String TYPE_BLURR_THRESHOLD = "TYPE_BLURR_THRESHOLD";
public static final String TYPE_OCCLUSION_THRESHOLD = "TYPE_OCCLUSION_THRESHOLD";
public static final String TYPE_ILLUMINATION_THRESHOLD = "TYPE_ILLUMINATION_THRESHOLD";
```

1）无感知活体阈值设置

a.可见光活体（默认值0.8）

通过算法对可见光图像中人脸进行活体判断，当算法输出的人脸活体分数大于“阈值”是判定活体校验通过。

b.红外活体（默认值0.8）

通过算法对近红外图像中人脸进行活体判断，当算法输出的人脸活体分数大于“阈值”是判定活体校验通过。

c.深度活体（默认值0.8）

通过算法对深度图像中人脸进行活体判断，当算法输出的人脸活体分数大于“阈值”是判定活体校验通过。

2）人脸识别阈值设置

a.可见光特征（默认值90）

通过算法对两张人脸进行对比得出相似度，当相似度大于“阈值”时判定两张照片为同一个人。

b.证件照特征（默认值90）

通过算法对两张人脸进行比对并得出相似度，当相似度大于“阈值”时判定两张照片为同一个人。

3）无感知活体检测设置（四选一）

a.不使用活体检测

活体检测主要用于判断当前进行对比的用户是否为真人，但如果使用场景中有固定人员看守，则无需进行活体检测，直接采集人脸。因为国体检测虽然保障了安全性，但却会影响一定的通过率。

b.单目活体（RGB）

采用单目活体模式，即使用RGB（彩色）活体检测，此方式可有效防止照片翻拍，屏幕成像等攻击，可使用市面上通用的USB单目摄像头，要求设备分辨率大于1280*720。

c.红外双目活体（RGB+NIR）

采用RGB（彩色）+NIR（近红外）双目活体模式，基于近红外成像特点，此方式下可提高对屏幕成像攻击、高清图片攻击的拒绝率。要求近红外波段860nm。

d.结构光双目活体（RGB+Depth）

采用RGB（彩色）+Depth（深度）双目活体模式，基于深度传感器特点，此方式下可提高对屏幕、高清彩打图片、视频、模具攻击的拒绝率。（三种结构光摄像头可选：奥比中光 Astra Pro/Pro S，奥比中光 Astra MiNi/MiNi S，华捷艾米 A100S+MINI）。

```
  if (GlobalSet.getStructuredLightValue() == GlobalSet.STRUCTURED_LIGHT.OBI_ASTRA_PRO) {
    intent = new Intent(this, OrbbecProPassActivity.class);
} else if (GlobalSet.getStructuredLightValue() == GlobalSet.STRUCTURED_LIGHT.OBI_ASTRA_MINI) {
    intent = new Intent(this, OrbbecMiniPassActivity.class);
} else if (GlobalSet.getStructuredLightValue() == GlobalSet.STRUCTURED_LIGHT.HUAJIE_AMY_MINI) {
    intent = new Intent(this, IminectPassActivity.class);
}
```

注：请设置与您设备相对应的方式！

4）摄像头预览旋转角度设置

此项设置用于管理：摄像头输出视频流的实际预览界面角度，一经设置，将会作用于所有类型的回显画面（RGB、NIR、Depth回显）可用于纠正摄像头原本输出图像中，人脸并没有水平朝上的问题，避免人脸检测不到的现象。旋转角度分为四种0,90,180,270。（顺时针方向）

5）设备激活

同上2.3.2按设备授权（授权过程中对模型进行初始化）

2.4 核心类

2.4.1 FaceSDKManager

- 功能：负责初始检测类FaceDetector、FaceFeature、FaceLiveness、FaceEnvironment
- com.baidu.idl.sample.manager

2.4.2 FaceDetector

- 功能：人脸检测封装类，包含人脸检测功能初始化、人脸检测接口调用
- com.baidu.idl.sample.manager

2.4.3 FaceFeatures

- 功能：人脸特征抽取封装类，包含人脸特征抽取功能初始化、人脸特征抽取接口调用
- com.baidu.idl.sample.manager

2.4.4 FaceLiveness

- 功能：人脸活体相关操作封装类，包含人脸rgb、ir、depth活体检测
- com.baidu.idl.sample.manager

2.4.5 FaceEnvironment

- 功能：人脸配置相关操作封装类，包含人脸检测的相关参数及默认值
- com.baidu.idl.sample.manager

Android-人证核验工程

版本日志

版本	日期	说明
v2.0.2	2019.04.01	1、全新人脸检测模型，检测追踪更流畅； 2、全新证件照模型，体积更小，速度更快； 3、部分接口细节优化； 3、版本号与主线SDK对齐
v1.1.0	2019.03.14	1、增加几款镜头模组支持；2、已知bug修复
V1.0.0	2019.01.10	人证核验示例工程初版

1、方案介绍

1.1 方案概述

人证核验方案，用于常见的1：1身份核验场景，用于证明用户的身份是否符实（即证明你是你）。常见的核验方式为以下几种：

- 身份证信息 vs 本人身份：常见操作为终端设备上存在身份证读卡器，用于读取身份证内部的芯片照，再与本人人脸信息进行1：1比对，确定当前待核验用户为证件本人。如交通卡口验证、身份核验机等。
- 系统ID信息 vs 本人身份：系统中记录的指定ID信息，通过人脸方式进行本人身份核验，如访客管理、会员验证、支付验证等。

核验终端设备，常置于无人看守场景，需要确保用户不可作弊，则在本人人脸获取过程中，增加**活体检测**能力，确保是**真人操**

作。不同类型技术方案的活体检测，需要依赖不同的前端镜头模组，则衍生出多种硬件设备适配方案。

此示例工程旨在提供通用的人证核验功能及配套设置，并提供一定的软硬件搭配方案，帮助开发者快速完成人证核验的业务原型搭建，及效果评估。

1.2 适用场景

此示例工程所提供的方案，可广泛应用于1：1身份核验场景，用于满足用户**是真人且是本人**的业务诉求。

- **人证核验机**
- **自助柜机**
- **交通卡口身份核验**
- **酒店前台核验机**
- **社区门禁**
- **访客管理**
- **.....**

1.3 功能介绍

1.3.1 设备激活

示例工程中提供两种工程授权激活方式，皆是基于设备维度的授权：

- 在线激活：只需手动填写激活码，便可快速完成联网激活（仅需操作一次）
- 离线激活：在设备完全不可联网的情况下，可将授权文件置于SD卡中，工程会自动寻找授权文件并完成激活。

1.3.2 功能首页

工程首页中，提供几种核心功能的入口和功能说明，方便快速进行演示操作和业务逻辑验证。

1.3.3 人证比对演示

模拟真实场景下，人脸图片与证件照图片对比的业务流程，特征抽取默认使用「证件照模型」，以处理对比过程中的证件照图片特征抽取的要求（证件照图片普遍像素较低）。配套使用的活体检测功能，需要在参数设置中单独选择对应的已经适配的镜头。

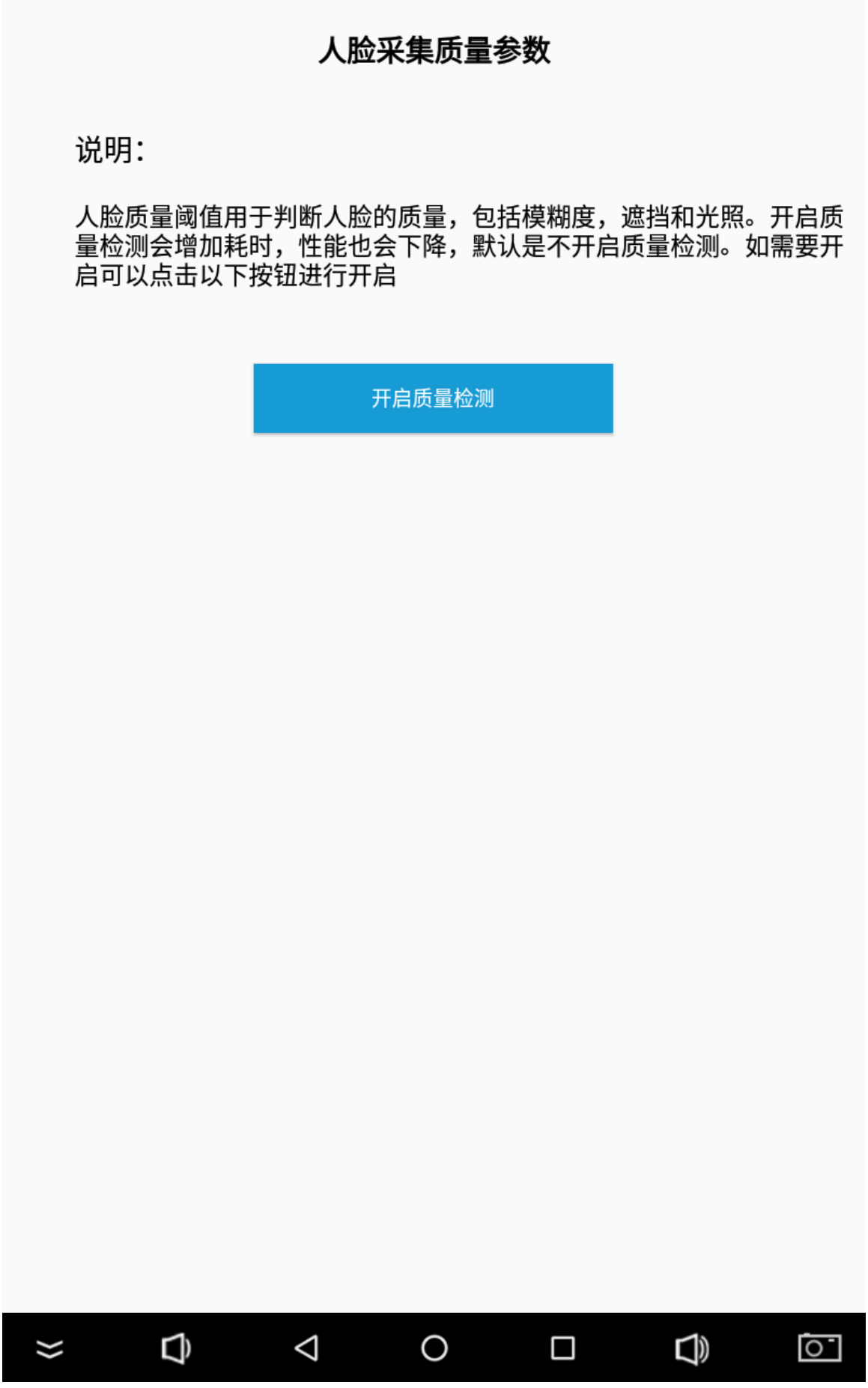
1.3.4 活体检测演示

用于体验和测试活体检测效果，界面内会持续显示实时的活体检测结果，可用于评估硬件及算法在真实场景下的真实表现。具体的活体模式可以在参数设置中按需选择。

例如选择RGB+NIR活体模式，界面内则会实时显示两种图像回显，用于反馈实时的视频流画面。同时显示活体判断结果、各项检测耗时、图片预览等。可直观地用于模拟真实应用状态下的活体效果。

1.3.5 参数设置

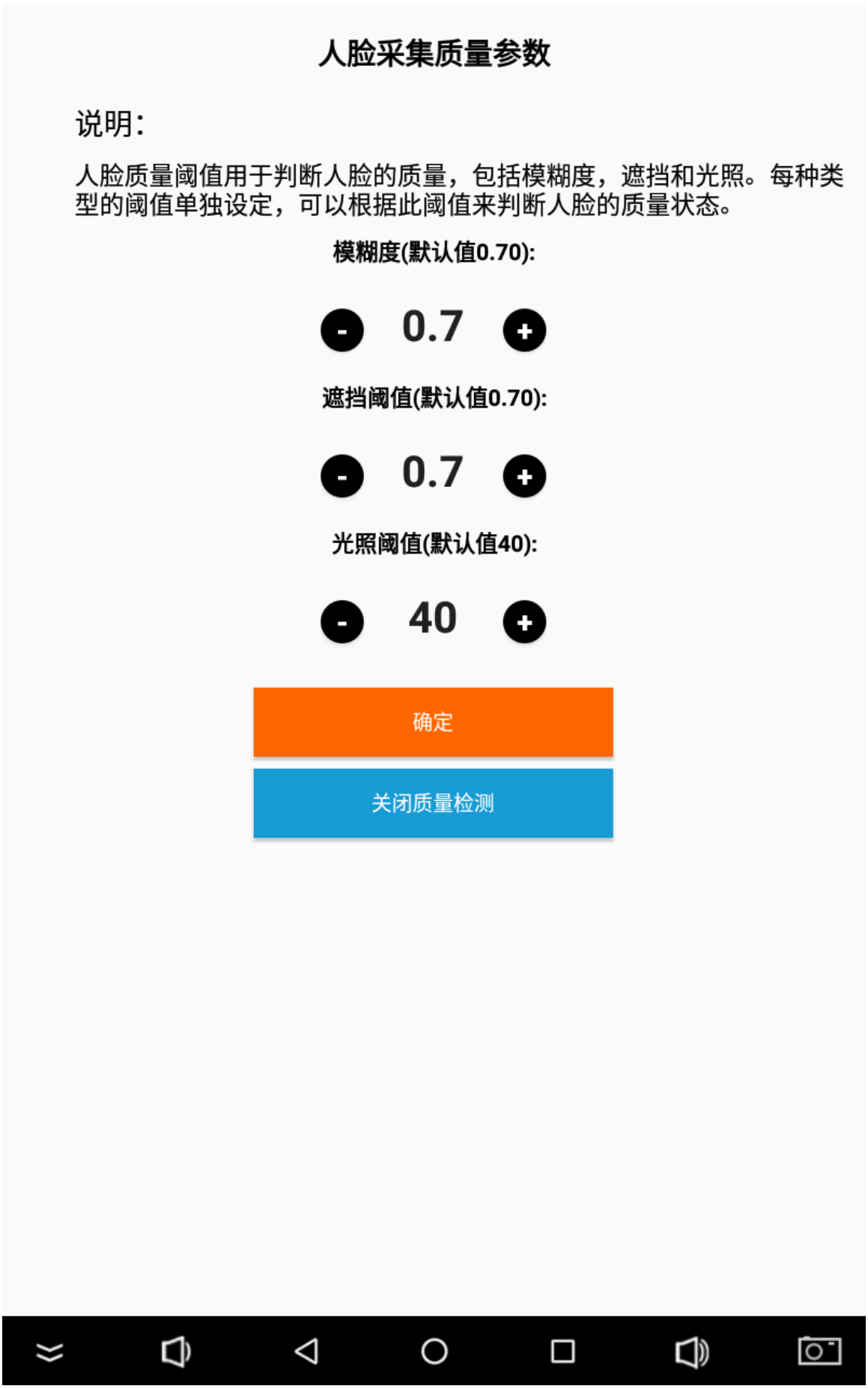
人脸采集质量参数



如开启此项，则在人脸检测过程中，将会增加实时的人脸质量检测，包括模糊度判断、遮挡判断、光照判断。如实时图片帧不满足这几项中的任何一项阈值，则此图片帧将会被舍弃，不会送到活体步骤判断。

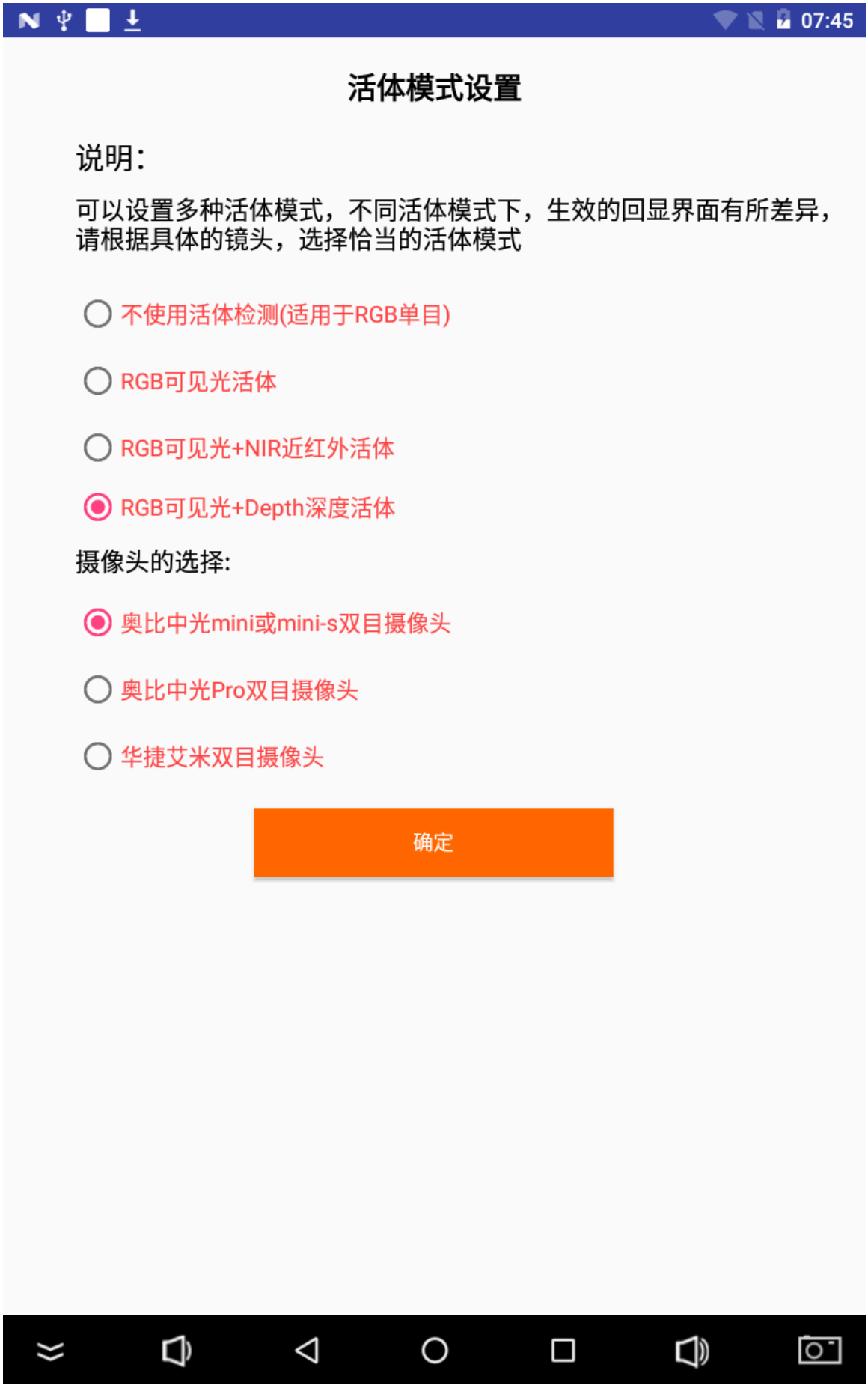
开启此项后，由于处理任务增加，性能开销也会对应增大。

点击「开启质量检测」，则可开启此项。



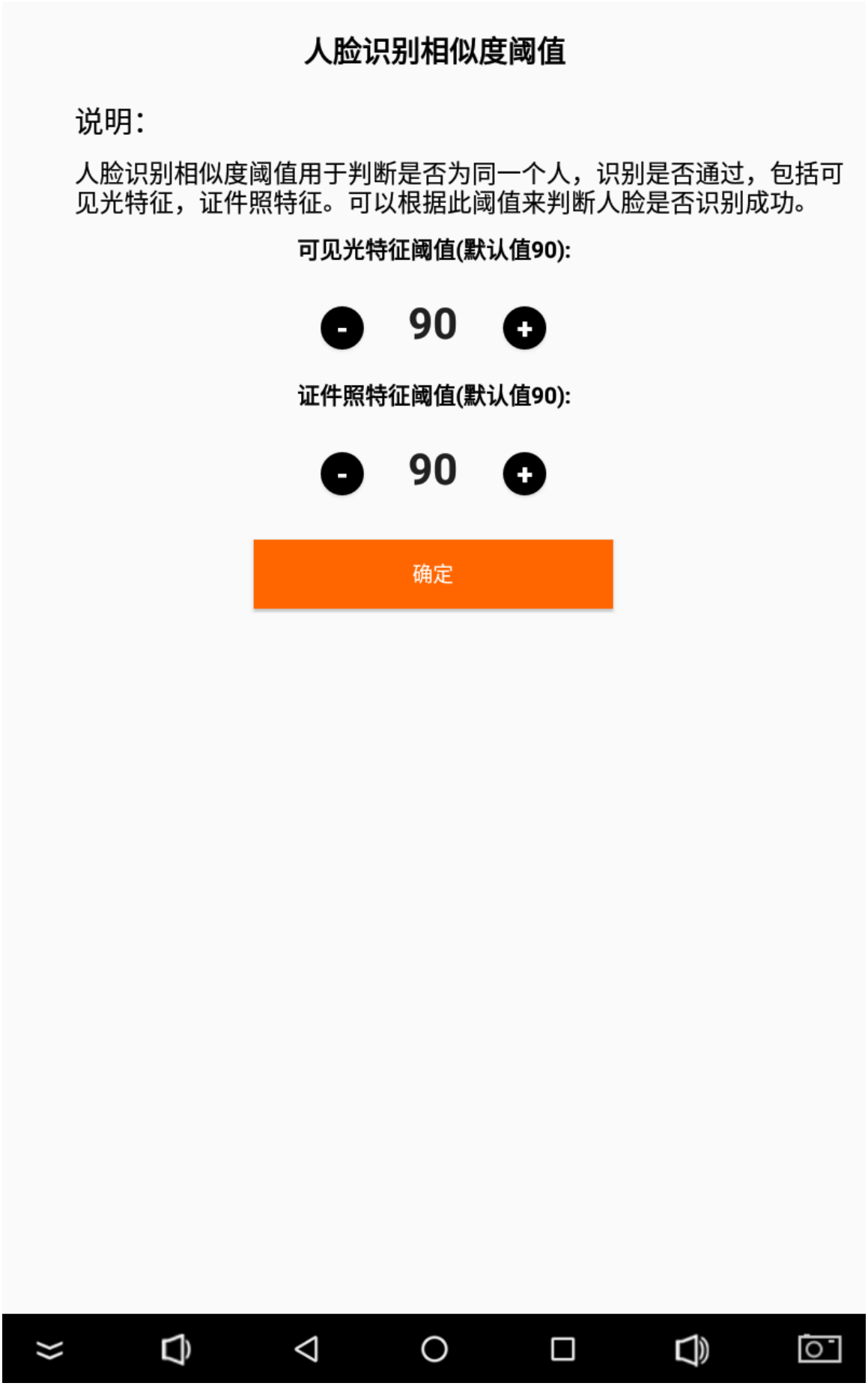
开启质量检测后，则可以自定义三个检测项的阈值，小于此阈值的图片帧将会被过滤掉。

活体模式设置



工程中提供四种活体模式配置，可根据业务需要自由选择。详细活体策略介绍，请参考 业务策略说明

识别相似度阈值



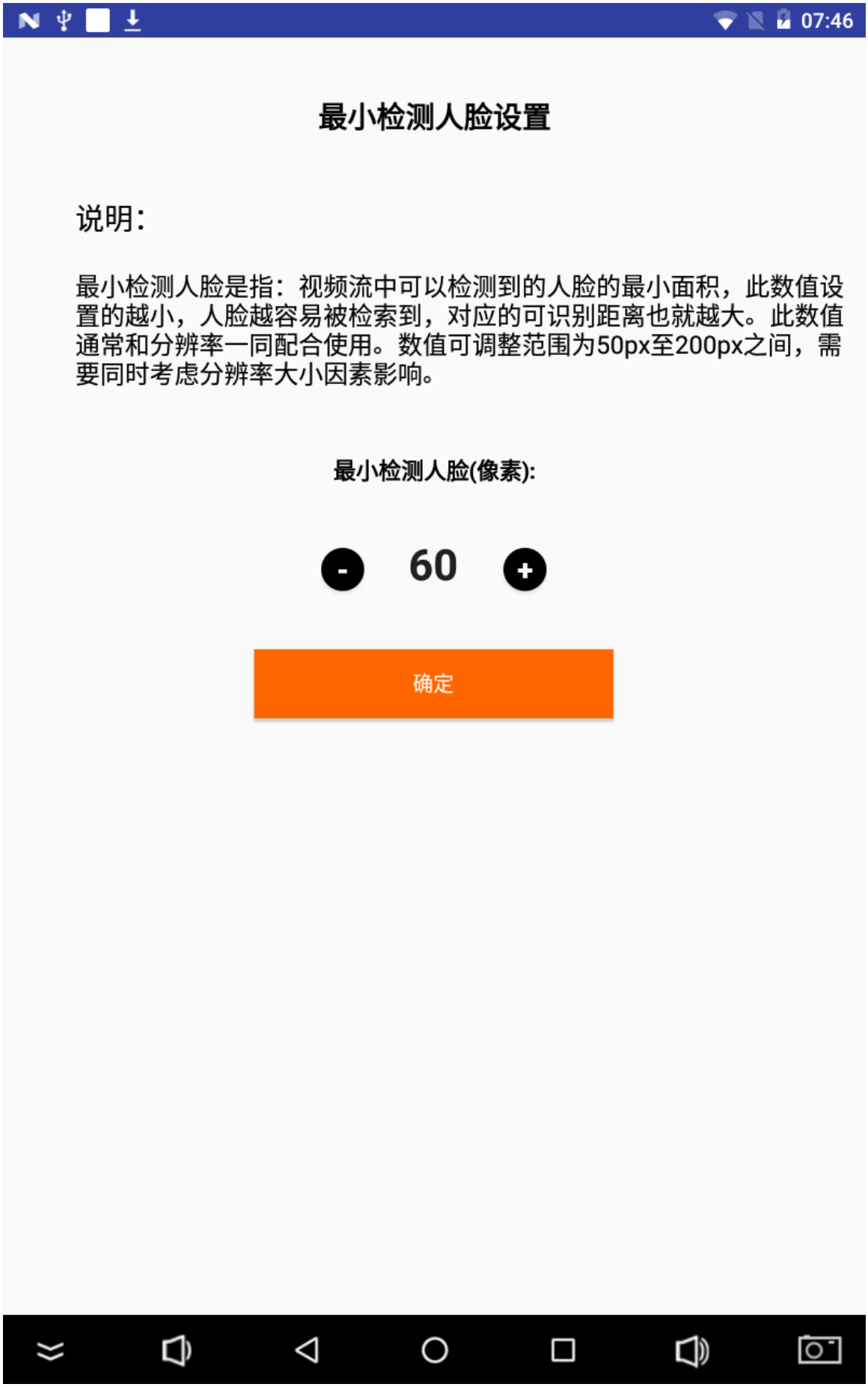
用于设置对比相似度的阈值，实际对比得分超过已设置的阈值，则可判断为同一人，推荐阈值80分或90分。

活体阈值设置

三种活体的阈值设定，超过此阈值则判断为活体通过。

> 注意：为业务安全性考虑，如果选择多种活体方式，则每一项活体得分都超过阈值，才会判断活体通过。

最小检测人脸设置



最小检测人脸是指：视频流中可以检测到的人脸的最小面积，相同镜头焦距情况下，此数值设置的越小，人脸越容易被检索到，对应的可识别距离也就越大。

此数值通常和分辨率一同配合使用。数值可调整范围为50px至200px之间，需要同时考虑分辨率大小因素影响。

人脸检测角度设置

此项设置用于管理：从视频流中实际检测人脸的方向，分为0、90、180、270四个角度。选择固定方向后，则只有人脸在此方向角度才会被检测到（RGB）

摄像头预览旋转角度设置

此项设置用于管理：摄像头输出视频流的实际预览界面角度，一经设置，将会作用于所有类型的回显画面（RGB、NIR、Depth回显）

可用于纠正摄像头原本输出图像中，人脸并没有水平朝上的问题，避免人脸检测不到的现象。

1.4 授权激活

详细说明请参考 授权说明

1.5 硬件选型

详细说明请参考 设备选型

2、集成指南

2.1 准备工作

2.1.1 注册开发者

1. STEP1：点击百度AI开放平台导航右侧的控制台，页面跳转到百度云登录界面，登录完毕后，将会进入到百度云后台，点击「控制台」进入百度云控制台页面；您也可以在官网直接点击免费试用，登录完毕后将自动进入到百度云控制台。
2. STEP2：使用百度账号完成登录，如您还未持有百度账户，可以点击此处注册百度账户。
3. STEP3：进入百度云欢迎页面，填写企业和个人基本信息，注册完毕，至此成为开发者。(如您之前已经是百度云用户或百度开发者中心用户，STEP3 可略过)
4. STEP4：进入百度云控制台，找到人工智能相关服务面板。
5. STEP5：点击进入「人脸识别」
6. STEP6：点击进入「离线SDK管理」

2.1.2 设备激活

首次运行示例工程，会弹出激活窗口。1为设备号，自动读取；2为序列号，手动输入。点击激活（激活需要联网，采用的https请求，https要求设备系统时间跟请求服务器时间差距不大，否则请求授权服务器会失败），激活以后使用不需要在连接网络，一个序列号绑定一台设备，卸载应用后重新安装可能需要重新激活，可以使用同一个序列号。可以在你的控制台查看设备号对应的序列号。

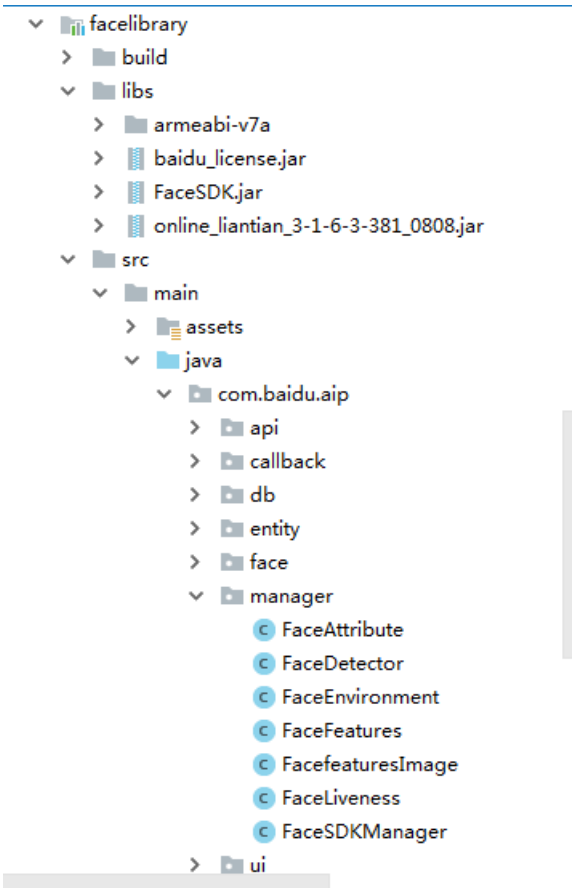
SDK的激活界面如下图所示：

2.2 代码结构

2.2.1 核心库介绍

facelibrary是SDK的依赖库。

- lib目录为动态库so和jar包
- assets目录为模型文件
- java目录为用户组管理、人脸SDK操作、视频流、图片等操作辅助类

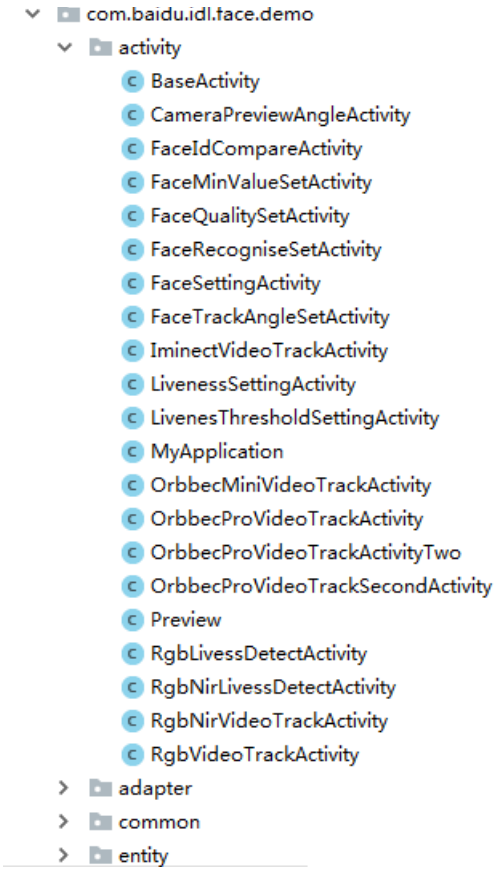


2.2.2 示例代码介绍

model名称	功能说明
FaceIdCompareActivity	人证对比，包括选择证件照图片和视频流实时采集人脸进行人脸识别比对，根据比对的结果分数确认是否为同一个人，核验是否通过。
LivenessSettingActivity	活体类型设置，分为无活体、RGB活体、RGB+NIR活体、RGB+Depth活体、RGB+NIR+Depth活体。默认为不使用活体。示例工程里面进行活体设置后，后续的人脸注册、人脸1：1，1：n等操作需选择相应Activity。无活体和rgb活体需要使用单目usb摄像头，rgb+ir活体需要使用rgb+ir双目摄像头（如迪威泰、视派尔双目摄像头，具体推荐型号详见上文设备选型介绍）。RGB+Depth需要使用奥比中光mini/mini-s/Pro-S、或华捷艾米摄像头，且目前只能使用RGB+Depth活体功能。RGB+NIR+Depth的活体硬件设备适配还在开发中。
CameraPreviewAngleActivity	摄像头预览角度设置，可以根据实际预览的情况进行摄像头预览画面角度的旋转调整，包括0,90,180,270
FaceMinValueSetActivity	最小检测人脸设置，此数值设置的越小，人脸越容易被检索到，对应的可识别距离也就越大，可调整数值范围为50px~200px
FaceQualitySetActivity	人脸采集质量参数设置，包括模糊度，遮挡和光照的设置。开启质量检测会增加耗时，性能也会下降，默认情况下是不开启质量检测。

FaceRecogniseSetActivity	人脸识别相似度阈值设置，默认值是90。根据次阈值可以判读人脸是否识别成功。
MainActivity	示例主界面入口
FaceSettingActivity	参数设置界面入口
FaceTrackAngleSetActivity	人脸检测角度设置，用于调整实际送去人脸检测的图片的角度，包括0,90,180,270。SDK只能识别人脸朝上的人脸。
LivenesThresholdSettingActivity	活体阈值设置，包括RGB活体阈值，NIR活体阈值，Depth活体阈值。默认值是0.9,活体分数达到0.9左右可以判断活体通过
IminectVideoTrackActivity	华捷艾米摄像头视频预览实时人脸活体检测特征提取
OrbbecMiniVideoTrackActivity	奥比中光mini摄像头视频预览实时人脸活体检测特征提取
OrbbecProVideoTrackSecondActivity	奥比中光Pro摄像头视频预览实时人脸活体检测特征提取
RgbNirVideoTrackActivity	双目近红外摄像头视频预览实时人脸活体检测特征提取
RgbVideoTrackActivity	单目RGB可见光摄像头视频预览实时人脸活体检测特征提取

以上所述文件位置如下图所示：



2.3 开始集成

接下来我们详细介绍以下集成步骤。

2.3.1 人脸库集成

1. 把facelibrary库添加到自己的工程中：（1）`settings.gradle` 添加‘:facelibrary’；（2）app->build.gradle->dependencies->compile project(":facelibrary")。

2. 根据需要把app里面的示例代码添加到自己的工程。

2.3.2 SDK初始化

采用默认参数进行初始化：

```
private void initSDK() {
    FaceSDKManager.getInstance().init( context: this, new FaceSDKManager.SdkInitListener() {
        @Override
        public void initStart() {
            toast( text: "sdk init start");
        }

        @Override
        public void initSuccess() {
            toast( text: "sdk init success");
        }

        @Override
        public void initFail(int errorCode, String msg) {
            toast( text: "sdk init fail:" + msg);
        }
    });
}
```

如果需要设置SDK的具体参数：

```
public FaceEnvironment getFaceEnvironmentConfig() {
    String faceMinValue = PreferencesUtil.getString(GlobalSet.TYPE_MIN_FACE_SET, String.valueOf(60));
    int qualityType = PreferencesUtil.getInt(TYPE_QUALITY, GlobalSet.TYPE_QUALITY_CLOSE);
    faceEnvironment.setMinFaceSize(Integer.parseInt(faceMinValue));
    faceEnvironment.setMaxFaceSize(-1);
    faceEnvironment.setDetectInterval(1000);
    faceEnvironment.setTrackInterval(500);
    faceEnvironment.setNoFaceSize(0.5f);
    faceEnvironment.setPitch(30);
    faceEnvironment.setYaw(30);
    faceEnvironment.setRoll(30);
    if (qualityType == GlobalSet.TYPE_QUALITY_CLOSE) {
        faceEnvironment.setCheckBlur(false);
        faceEnvironment.setOcclusion(false);
        faceEnvironment.setIllumination(false);
    } else if (qualityType == GlobalSet.TYPE_QUALITY_OPEN) {
        faceEnvironment.setCheckBlur(true);
        faceEnvironment.setOcclusion(true);
        faceEnvironment.setIllumination(true);
    }
    faceEnvironment.setDetectMethodType(FaceDetect.DetectType.DETECT_VIS);
    return faceEnvironment;
}
```

详细参数设置参考如下表格所示：

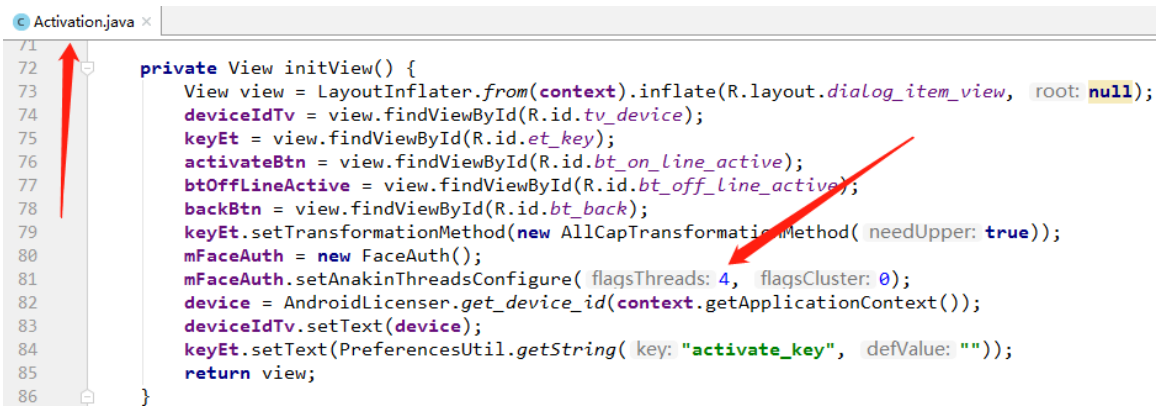
参数	名称	默认值	取值范围
brightnessValue	图片爆光度	40f	50~255
blurnessValue	图像模糊度	0.7f	0~1.0f
occlusionValue	人脸遮挡阈值	0.5f	0~1.0f
headPitchValue	低头抬头角度	15	0~45
headYawValue	左右角度	15	0~45
headRollValue	偏头角度	15	0~45
minFaceSize	最小人脸检测值，小于此值的人脸将检测不出来，最小值为50	50	50~200
notFaceSize	人脸置信度	0.8f	0~1.0f
isCheckBlur	是否检测人脸模糊度	False	True/Flase
isOcclusion	是否检测人脸遮挡	False	True/Flase
isIllumination	是否检测人脸曝光度	False	True/Flase

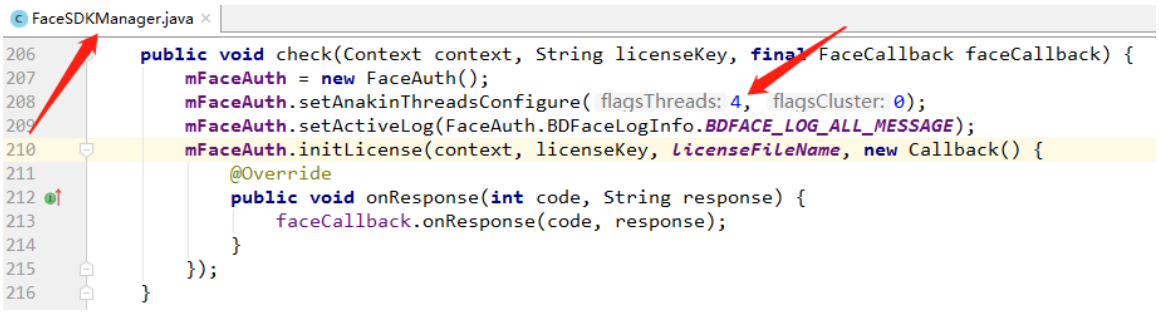
2.3.3 按设备授权

SDK初始化的时候会检测设备授权，如果没有授权，会弹出授权框，填入在平台上创建的序列号。授权成功SDK才可初始化成功。

2.3.4 设置Anakin核数说明

SDK在授权初始化模型之前会进行Anakin大小核数的设置，共有两处地方需要修改。 在3399板子上运行，需要设置为4个大核0个小核。在3288板子的上运行，需要设置为2个大核0小核。 示例代码中默认的是是设置为4个大核0个小核的，用户需要根据自己板子情况进行修改，参考下图：





2.3.5 人脸采集质量参数设置

模型介绍

人脸质量阈值用于判断人脸的质量，包括模糊度，遮挡和光照。当质量检测通过之后，才进行活体的检测特征抽取比对识别。

```
float[] occlu = faceInfo.occlu;
if (occlu != null && occlu.length > 0) {
    float leftEye = occlu[0];
    float rightEye = occlu[1];
    float nose = occlu[2];
    float mouth = occlu[3];
    float lContour = occlu[4];
    float rContour = occlu[5];
    float chinContour = occlu[6];
    if ((faceInfo.blur < blurrValue) && (leftEye < occlusionValue) && (rightEye < occlusionValue)
            && (nose < occlusionValue) && (mouth < occlusionValue) && (lContour < occlusionValue)
            && (rContour < occlusionValue) && (chinContour < occlusionValue)
            && (faceInfo.illum > illuminateValue)) {
        livingCheck(width, height, type, faceInfos, livenessModel);
    } else {
        if (livenessCallBack != null) {
            livenessCallBack.onTip( code: 0, msg: "人脸质量差");
            livenessCallBack.onCallback( livenessModel: null);
        }
    }
} else {
    livingCheck(width, height, type, faceInfos, livenessModel);
}
```

注意事项

温馨提示：开启质量检测会增加耗时，性能也会下降，默认是不开启质量检测。

2.3.6 无感知活体检测模式

活体介绍

详见文档1.3.2部分

使用方法

model名称	功能说明
LivenessSettingActivity	活体类型设置，分为无活体、RGB活体、RGB+NIR活体、RGB+Depth活体、RGB+NIR+Depth活体。默认为不使用活体。示例工程里面进行活体设置后，后续的人脸注册、人脸1：1，1：n等操作需选择相应Activity。无活体和rgb活体需要使用单目usb摄像头，rgb+ir活体需要使用rgb+ir双目摄像头（如迪威泰、视派尔双目摄像头，具体推荐型号详见上文设备选型介绍）。RGB+Depth需要使用奥比中光mini/mini-s/Pro-S、或华捷艾米摄像头，且目前只能使用RGB+Depth活体功能。RGB+NIR+Depth的活体硬件设备适配还在开发中。

温馨提示：业务流程中，只能使用一种活体设置。

2.3.7 人脸识别相似度阈值（FaceRecogniseSetActivity）

人脸识别相似度阈值用于判断是否为同一个人，识别是否通过，包括可见光特征，证件照特征。可以根据此阈值来判断人脸是否识别成功，核验是否通过。

```
float rgbFeatureValue = Float.parseFloat(PreferencesUtil.getString(TYPE_RGB_FEATURE_THRESHOLD, String.valueOf(90)));
float idFeatureValue = Float.parseFloat(PreferencesUtil.getString(TYPE_ID_FEATURE_THRESHOLD, String.valueOf(90)));
float score = 0;
score = FaceApi.getInstance().matchIdFeaturePhoto(firstFeature, secondFeature);
Log.e( tag: "chaixiaogang", msg: "compare score"+score);
if (score > idFeatureValue) {
    tvState.setTextColor(getResources().getColor(R.color.green));
    tvState.setText("核验通过");
} else {
    tvState.setTextColor(getResources().getColor(R.color.red));
    tvState.setText("核验不通过");
}
tvScore.setText("相似度：" + score);
```

2.3.8 活体阈值设置（LivenesThresholdSettingActivity） 活体阈值用于判断是否活体通过，每种活体模型的阈值单独设定，可以根据次阈值来判断活体的通过率和拒绝率。通常情况下，真人的活体得分极大接近于1，非活体极大接近于0.0，阈值设定视安全性而定。

```
if ((type & FaceLiveness.MASK_DEPTH) == FaceLiveness.MASK_DEPTH) {
    tvDepthTime.setVisibility(View.VISIBLE);
    tvDepthScore.setVisibility(View.VISIBLE);
    tvDepthScore.setText("Depth活体得分：" + livenessModel.getDepthLivenessScore());
    tvDepthTime.setText("Depth活体耗时：" + livenessModel.getDetphtLivenessDuration());
    currentDepthScore = livenessModel.getDepthLivenessScore();
    Log.i( tag: "chaixiaogang",  msg: "depth live score" + currentDepthScore);
}
if (currentRgbScore > rgbThreshold) {
    tvSuccessState.setVisibility(View.VISIBLE);
    tvFailState.setVisibility(View.GONE);
    tvSuccessState.setText("可见光活体：通过");
} else {
    tvSuccessState.setVisibility(View.GONE);
    tvFailState.setVisibility(View.VISIBLE);
    tvFailState.setText("可见光活体：不通过");
}

if (currentDepthScore > depthThreshold) {
    tvDepthSuccessState.setVisibility(View.VISIBLE);
    tvDepthFailState.setVisibility(View.GONE);
    tvDepthSuccessState.setText("深度活体：通过");
} else {
    tvDepthSuccessState.setVisibility(View.GONE);
    tvDepthFailState.setVisibility(View.VISIBLE);
    tvDepthFailState.setText("深度活体：不通过");
}
```

2.3.9 最小检测人脸设置(FaceMinValueSetActivity) 最小检测人脸是指：视频流中可以检测到的人脸的最小面积，此数值设置的越小，人脸越容易被检索到，对应的可识别距离也就越大。此数值通常和分辨率一同配合使用。数值可调整范围为50px至200px之间，需要同时考虑分辨率大小因素影响。

2.4.0 人脸检测角度设置（FaceTrackAngleSetActivity） 此项设置用于管理：从视频流中实际检测人脸的方向，分为0、90、180、270四个角度。选择固定方向后，则只有人脸在此方向角度才会被检测到（RGB）。

注意事项

温馨提示：SDK只检测人脸朝上的人脸，需要确保送去检测的人脸角度是正的，朝上的。

2.4.1 摄像头预览旋转角度设置（CameraPreviewAngleActivity） 此项设置用于管理：摄像头输出视频流的实际预览界面角度，分为0、90、180、270四个角度。一经设置，将会作用于所有类型的回显画面（RGB、NIR、Depth回显）可用于纠正摄像头原本输出图像中，人脸并没有水平朝上的问题，避免人脸检测不到的现象。

2.4.2 人证对比

2.4.2.1 从相册里选择两张图片进行对比

此种方法无需使用人脸采集及活体等功能。

注意事项

> 温馨提示：SDK只检测人脸朝上的人脸。

2.4.2.2 从视频流中采集两张人脸图片进行对比

此种方式的人脸图片需要从视频流中实时采集，如果为无人值守情况，还需配备活体检测以保障业务安全。FaceIdCompareActivity根据活体策略选择相应的实现，开发者可以根据实际使用的硬件进行选择。

1）获取人脸，可选择以下5种方式返回人脸

- RgbLivessDetectActivity：无活体或RGB活体（活体检测成功后，返回检测到的人脸）
- RgbNirLivessDetectActivity：进行RGB+NIR活体成功后返回检测到RGB人脸
- OrbbecMiniVideoTrackActivity：进行RGB+Depth活体成功后返回检测到RGB人脸（奥比中光mini镜头）
- OrbbecProVideoTrackSecondActivity：进行RGB+Depth活体成功后返回检测到RGB人脸（奥比中光Pro镜头）
- IminectVideoTrackActivity：进行RGB+Depth活体成功后返回检测到RGB人脸（华捷艾米镜头）

```
private void collectImage(LivenessModel model) {
    FaceInfo faceInfo = model.getFaceInfo();
    ImageFrame imageFrame = model.getImageFrame();
    float[] headPose = faceInfo.headPose;
    if (Math.abs(headPose[0]) > 30 || Math.abs(headPose[1]) > 30 || Math.abs(headPose[2]) > 30) {
        runOnUiThread(() -> { tvTopShowState.setText("人脸置角度太大，请正对屏幕"); });
    } else {
        Bitmap bitmap = FaceCropper.getFace(imageFrame.getArgb(),
                faceInfo, imageFrame.getWidth());
        try {
            // 其他来源保存到临时目录
            final File file = File.createTempFile( prefix: UUID.randomUUID().toString() + "",  suffix: ".jpg");
            // 人脸识别不需要整张图片。可以对人脸区别进行裁剪。减少流量消耗和，网络传输占用的时间消耗。
            ImageUtils.resize(bitmap, file,  maxWidth: 300,  maxHeight: 300);
            Intent intent = new Intent();
            intent.putExtra( name: "file_path", file.getAbsolutePath());
            setResult(Activity.RESULT_OK, intent);
            success = true;
            finish();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

2）根据返回人脸抽取特征

```
private void syncFeature(final Bitmap bitmap, final byte[] feature, final int index) {
    float ret = 0;
    ret = FaceApi.getInstance().extractIdPhotoFeature(bitmap, feature);
    Log.i( tag: "chaixiaogang",  msg: "ret:" + ret);
    if (ret == 128 && index == 1) {
        firstFeatureFinished = true;
    } else if (ret == 128 && index == 2) {
        secondFeatureFinished = true;
    }
    if (ret == 128) {
        toast( tip: "图片" + index + "特征抽取成功");
    } else if (ret == -100) {
        toast( tip: "未完成人脸比对，可能原因，图片1为空");
    } else if (ret == -101) {
        toast( tip: "未完成人脸比对，可能原因，图片2为空");
    } else if (ret == -102) {
        toast( tip: "未完成人脸比对，可能原因，图片1未检测到人脸");
    } else if (ret == -103) {
        toast( tip: "未完成人脸比对，可能原因，图片2未检测到人脸");
    } else {
        toast( tip: "未完成人脸比对，可能原因，"
                + "人脸太小（小于sdk初始化设置的最小检测人脸）"
                + "人脸不是朝上，sdk不能检测出人脸");
    }
}
```

3）比对两张人脸图片

```
if (!firstFeatureFinished) {
    toast( tip: "图片1特征没有抽取成功");
    return;
}

if (!secondFeatureFinished) {
    toast( tip: "图片2特征没有抽取成功");
    return;
}
float rgbFeatureValue = Float.parseFloat(PreferencesUtil.getString(TYPE_RGB_FEATURE_THRESHOLD, String.valueOf(90)));
float idFeatureValue = Float.parseFloat(PreferencesUtil.getString(TYPE_ID_FEATURE_THRESHOLD, String.valueOf(90)));
float score = 0;
score = FaceApi.getInstance().matchIdFeaturePhoto(firstFeature, secondFeature);
Log.e( tag: "chaixiaogang", msg: "compare score"+score);
if (score > idFeatureValue) {
    tvState.setTextColor(getResources().getColor(R.color.green));
    tvState.setText("核验通过");
} else {
    tvState.setTextColor(getResources().getColor(R.color.red));
    tvState.setText("核验不通过");
}
tvScore.setText("相似度：" + score);
```

2.4.3 活体检测（视频流）

您可以根据实际硬件选择活体策略，有下面几种实现：

- RgbVideoTrackActivity 无活体或rgb活体(单目RGB镜头)
- RgbNirVideoTrackActivity rgb+ir活体(双目近红外镜头)
- OrbbecMiniVideoTrackActivity rgb+depth活体(奥比中光mini镜头)
- OrbbecProVideoTrackSecondActivity rgb+depth活体(奥比中光Pro镜头)
- IminectVideoTrackActivity rgb+depth活体(华捷艾米镜头)

开发者可以根据活体策略和实际使用的硬件（连接的摄像头）选择相应的实现。

2.4.3.1 RgbVideoTrackActivity

1）初始化视频流检测

```
private void initLoadConfig() {
    faceDetectManager = new FaceDetectManager(getApplicationContext());
    // 从系统相机获取图片帧。
    final CameraImageSource cameraImageSource = new CameraImageSource( context: this);
    // 可以通过 camera.getParameters().getSupportedPreviewSizes()查看支持列表。
    cameraImageSource.getCameraControl().setPreferredPreviewSize( width: 640,  height: 480);
    cameraImageSource.getCameraControl().setDisplayOrientation(previewAngle);
    // 设置预览
    cameraImageSource.setPreviewView(previewView);
    // 设置图片源
    faceDetectManager.setImageSource(cameraImageSource);
    // 设置人脸过滤角度，角度越小，人脸越正，比对时分数越高
    faceDetectManager.getFaceFilter().setAngle(20);
    textureRectView.setOpaque(false);
    // 不需要屏幕自动变黑。
    textureRectView.setKeepScreenOn(true);
    boolean isPortrait = getResources().getConfiguration().orientation == Configuration.ORIENTATION_PORTRAIT;
    if (isPortrait) {
        previewView.setScaleType(PreviewView.ScaleType.FIT_WIDTH);
        // 相机竖屏模式
        cameraImageSource.getCameraControl().setDisplayOrientation(CameraView.ORIENTATION_PORTRAIT);
    } else {
        previewView.setScaleType(PreviewView.ScaleType.FIT_HEIGHT);
        // 相机横屏模式
        cameraImageSource.getCameraControl().setDisplayOrientation(CameraView.ORIENTATION_HORIZONTAL);
    }
    setCameraType(cameraImageSource);
}
```

2）选择摄像头类型

```
private void setCameraType(CameraImageSource cameraImageSource) {
    // TODO 选择使用前置摄像头
    // cameraImageSource.getCameraControl().setCameraFacing(ICameraControl.CAMERA_FACING_FRONT);

    // TODO 选择使用usb摄像头
    cameraImageSource.getCameraControl().setCameraFacing(ICameraControl.CAMERA_USB);
    // 如果不设置，人脸框会镜像，显示不准
    previewView.getTextureView().setScaleX(-1);

    // TODO 选择使用后置摄像头
      cameraImageSource.getCameraControl().setCameraFacing(ICameraControl.CAMERA_FACING_BACK);
      previewView.getTextureView().setScaleX(-1);
}
```

3）设置视频流人脸检测回调，在回调进行比对

```
private void addListener() {
    // 设置回调，回调人脸检测结果。
    faceDetectManager.setOnFaceDetectListener(new FaceDetectManager.OnFaceDetectListener() {
        @Override
        public void onDetectFace(FaceInfo[] infos, ImageFrame frame) {
            setTrackAngle(frame); // 显示检测的图片。用于调试，如果人脸sdk检测的人脸需要朝上，可以通过该图片判断。实际应用中可注释掉
            myHandler.sendEmptyMessage( what: 1);
            if (infos != null && infos.length > 0) {
                myHandler.sendEmptyMessage( what: 2);
                showFrameRect(frame, infos);
                syncIdentity(frame, infos);
            } else {
                myHandler.sendEmptyMessage( what: 3);
                showFrameRect(frame, infos);
            }
        }
    });
}
```

4）活体检测

```
private float rgbLiveness(ImageFrame imageFrame, FaceInfo faceInfo) {
    long starttime = System.currentTimeMillis();
    final float rgbScore = FaceSDKManager.getInstance().getFaceLiveness().rgbLiveness(imageFrame.getArgb(), imageFrame
            .getWidth(), imageFrame.getHeight(), faceInfo.landmarks);
    Log.e( tag: "chaixiaogang",  msg: "rgb live score" + rgbScore);
    final long duration = System.currentTimeMillis() - starttime;
    mRgbScore = rgbScore;
    runOnUiThread(new Runnable() {
        @Override
        public void run() {
            tvRgbLivingTime.setVisibility(View.VISIBLE);
            tvRgbLivingScore.setVisibility(View.VISIBLE);
            tvRgbLivingTime.setText("RGB活体耗时：" + duration);
            tvRgbLivingScore.setText("RGB活体得分：" + rgbScore);
            float currentRgbScore = rgbScore;
            if (currentRgbScore > rgbThreshold) {
                tvSuccessState.setVisibility(View.VISIBLE);
                tvFailState.setVisibility(View.GONE);
                tvSuccessState.setText("可见光活体：通过");
            } else {
                tvSuccessState.setVisibility(View.GONE);
                tvFailState.setVisibility(View.VISIBLE);
                tvFailState.setText("可见光活体：不通过");
            }
        }
    });
    return rgbScore;
}
```

5）特征抽取

```
private void identity(ImageFrame imageFrame, FaceInfo faceInfo) {
    float raw = Math.abs(faceInfo.headPose[0]);
    float patch = Math.abs(faceInfo.headPose[1]);
    float roll = Math.abs(faceInfo.headPose[2]);
    // 人脸的三个角度大于20不进行识别
    if (raw > 20 || patch > 20 || roll > 20) {
        return;
    }
    long starttime = System.currentTimeMillis();
    int[] argb = imageFrame.getArgb();
    int rows = imageFrame.getHeight();
    int cols = imageFrame.getWidth();
    int[] landmarks = faceInfo.landmarks;
    byte[] imageFrameFeature = new byte[512];
    FaceSDKManager.getInstance().getFaceFeature().extractFeatureForIDPhoto(argb, rows, cols,
            imageFrameFeature, landmarks);
    displayTip( text: "特征抽取对比耗时:" + (System.currentTimeMillis() - starttime), tvFeatureCompareTime);
}
```

2.4.3.2 RgbNirVideoTrackActivity

1）初始化视频流：RGB+NIR双目摄像头通过系统api出来的视频流都是yuv420数据，需要区分出RGB还是NIR数据。通过选取

一点数量点的取平均值比较，大的为RGB，小的为NIR。

```
try {
    mCamera[0] = Camera.open(0);
    mCamera[1] = Camera.open(1);

    mCamera[0].setDisplayOrientation(previewAngle);
    mCamera[1].setDisplayOrientation(previewAngle);

    mPreview[0].setCamera(mCamera[0], PREFER_WIDTH, PERFER_HEIGH);
    mPreview[1].setCamera(mCamera[1], PREFER_WIDTH, PERFER_HEIGH);

    mCamera[0].setPreviewCallback((data, camera) -> {
            if (rgbOrIrConfirm) {
                choiceRgbOrIrType( index: 0, data);
            } else if (camemra1DataMean == 0) {
                rgbOrIr( index: 0, data);
            }
    });
    mCamera[1].setPreviewCallback((data, camera) -> {
            if (rgbOrIrConfirm) {
                choiceRgbOrIrType( index: 1, data);
            } else if (camemra2DataMean == 0) {
                rgbOrIr( index: 1, data);
            }
    });
} catch (RuntimeException e) {
    Log.e(TAG, e.getMessage());
```

2）人脸活体检测

```
private synchronized void checkData() {
    if (rgbData != null && irData != null) {
        FaceSDKManager.getInstance().getFaceLiveness().setRgbInt(rgbData);
        FaceSDKManager.getInstance().getFaceLiveness().setIrData(irData);
        if (trackAngle == 90 || trackAngle == 270) {
            FaceSDKManager.getInstance().getFaceLiveness().livenessCheck(PERFER_HEIGH, PREFER_WIDTH,  type: 0x0011);
        } else {
            FaceSDKManager.getInstance().getFaceLiveness().livenessCheck(PREFER_WIDTH, PERFER_HEIGH,  type: 0x0011);
        }
        rgbData = null;
        irData = null;
    }
}
```

3）特征抽取（与RgbVideoTrackActivity实现相同）

2.4.3.3 OrbbecMiniVideoTrackActivity

1）初始化视频流+人脸检测+活体：采用奥比中光双目摄像头，Depth数据为16位数据

```
    synchronized (sync) {

        mDepthGLView.update(depthStream, com.orbbec.utils.GlobalDef.TYPE_DEPTH);
        mRgbGLView.update(rgbStream, com.orbbec.utils.GlobalDef.TYPE_COLOR);

        ByteBuffer depthByteBuf = depthStream.readFrame().getData();
        ByteBuffer colorByteBuf = rgbStream.readFrame().getData();
        int depthLen = depthByteBuf.remaining();
        int rgbLen = colorByteBuf.remaining();

        byte[] depthByte = new byte[depthLen];
        byte[] rgbByte = new byte[rgbLen];

        depthByteBuf.get(depthByte);
        colorByteBuf.get(rgbByte);

        final Bitmap bitmap = ImageUtils.RGB2Bitmap(rgbByte, mWidth, mHeight);
        FaceSDKManager.getInstance().getFaceLiveness().setRgbBitmap(bitmap);
        FaceSDKManager.getInstance().getFaceLiveness().setDepthData(depthByte);
        FaceSDKManager.getInstance().getFaceLiveness().livenessCheck(mWidth, mHeight,  type: 0X0101);
    }
}
```

2）活体检测回调

```
    FaceSDKManager.getInstance().getFaceLiveness().setLivenessCallBack(new ILivenessCallBack() {
        @Override
        public void onCallback(LivenessModel livenessModel) {
            checkResult(livenessModel);
            mModel = livenessModel;
        }

        @Override
        public void onTip(int code, final String msg) {
            runOnUiThread(new Runnable() {
                @Override
                public void run() {
                    tvTopShowState.setText(msg);
                }
            });
        }

        @Override
        public void onFaceTrackCallback(LivenessModel livenessModel) {
            showFrameRect(mModel);
        }

    });
}
```

3）特征抽取（与RgbVideoTrackActivity实现相同）

2.4.3.4 IminectVideoTrackActivity

1）初始化视频流+人脸检测+活体：采用华捷艾米双目深度摄像头，Depth数据为16位数据

```
if (colorFrame == null) {
    colorFrame = mCamera.readNextFrame( timeout: 40);
}
if (null == depthFrame || null == colorFrame) {
    continue;
}

mColorBuffer = colorFrame.getData();
mDepthBuffer = depthFrame.getData();
ByteBuffer depthframeData;
depthframeData = Utils.depth2RGB888(depthFrame, histogram: true, extractDepth: false);
int rgbLen = mColorBuffer.remaining();
byte[] rgbByte = new byte[rgbLen];
mColorBuffer.get(rgbByte);
final Bitmap bitmap = ImageUtils.RGB2Bitmap(rgbByte, width: 640, height: 480);
mColorSurfaceView.updateVertices(mColorBuffer);
mDepthSurfaceView.updateVertices(depthframeData);
int depthLen = mDepthBuffer.remaining();
byte[] depthByte = new byte[depthLen];
mDepthBuffer.get(depthByte);
FaceSDKManager.getInstance().getFaceLiveness().setRgbBitmap(bitmap);
FaceSDKManager.getInstance().getFaceLiveness().setDepthData(depthByte);
FaceSDKManager.getInstance().getFaceLiveness().livenessCheck( width: 640, height: 480, type: 0X0101);
colorFrame = null;
depthFrame = null;
```

2）活体检测回调（与OrbbecMiniVideoTrackActivity实现相同）

3）特征抽取（与RgbVideoTrackActivity实现相同）

2.4 核心类

2.4.1 FaceSDKManager

- 功能：**负责授权激活、初始检测类FaceDetector、FaceFeature、FaceLiveness**
- 位于：`com.baidu.aip.manager`

2.4.2 FaceDetector

- 功能：**人脸检测封装类，包含人脸检测功能初始化、人脸检测接口调用**
- 位于：`com.baidu.aip.manager`

2.4.3 FaceFeature

- 功能：**人脸特征抽取封装类，包含人脸特征抽取功能初始化、人脸特征抽取接口调用**
- 位于：`com.baidu.aip.manager`

2.4.4 FaceLiveness

- 功能：**人脸活体相关操作封装类，包含人脸rgb、ir、depth活体检测**
- 位于：`com.baidu.aip.manager`

2.4.5 FaceEnvironment

- 功能：**人脸配置相关操作封装类，包含人脸检测的相关参数及默认值**
- 位于：`com.baidu.aip.manager`

2.4.6 FaceDetectManager

- 功能：**人脸图片和视频人脸检测封装类，可以接受CameraImageSource和FileImageSource源**
- 位于：`com.baidu.aip.face`

2.4.7 FaceAttribute

- 功能：**人脸属性相关操作封装类，包含年龄，性别，表情等**
- 位于：`com.baidu.aip.manager`

2.4.8 FacefeaturesImage

- 功能：**人脸特征抽取封装类，包含人脸特征抽取功能初始化、人脸特征抽取接口调用**
- 位于：`com.baidu.aip.manager`

3、常见问题

Q：提示硬件指纹变化，导致激活失效？
A：v1.0.1版本以上的SDK，已经修复了硬件指纹变化问题。但由于硬件本身的多样化原因，可能某些情况下，仍会导致指纹变化。

Q：哪些情况可能导致指纹变化？
A：刷机、更换硬件设备将会导致指纹变化。但安卓系统升级、APP卸载重装、恢复系统出厂值并不会导致硬件指纹变化。

Q：同一台设备可以被多个序列号多次激活么？
A：可以。一台设备可以被多个序列号激活，没有限制。

Q：调用`getFeature`接口对图片进行特征提取，经常会出现特征提取失败的情况，错误码为6。
A：主要可能为在人脸检测的过程中没有检测到人脸，建议调整设置下人脸的大小`set_min_face_size`的值。

Q：视频流人脸检测和图片检测是否可以同时或间隔进行？
A：人脸SDK为单例，同一时间只能进行一个图像源。另外进行人脸检测具体追踪功能，视频流进行检测时，不能插入其他图像帧。想`VideoMatchImageActivity`、`RgbIrVideoMatchImageActivity`、`OrbbecVideoMatchImageActivity`，需要先把图片进行人脸检测后，在把打开视频流检测。中间需要使用`FaceSDKManager.getInstance().getFaceDetector().clearTrackedFaces();`清除数据

Q：人脸检测检测不到人脸？
A：人脸SDK检测需要传入检测的人脸图片是人脸朝上，预览和实际传给SDK检测的图片方向不一定相同，需要把实际检测的数据转成（**argb->bitmap**）图片，显示确定人脸是否朝上。

Q：如何调整人脸检测识别距离，以及调节检测的最小人脸？
A：主要方法有三种，详情如下：

1. 调整FaceDetector初始化时最小检测人脸大小（FaceEnvironment VALUE_MIN_FACE_SIZE = 100;），可选范围为：50~200（50*50px-200*200px），最小检测人脸越小，能检测到人脸越小。最小检测人脸越小，性能消耗越大。
2. 调整人脸检测传入的图像分辨率，分辨率越大，能检测越越远。鉴于目前端设备性能，建议选择640*480，1280*720。分辨率越大，性能消耗越大。
3. 选择更大焦距的摄像头，相当于把人脸拉近。同样距离，大焦距的镜头，图像视觉越小，人脸占比越大。通常USB摄像头为3mm、6mm焦距。对性能影响小，调整人脸检测距离明显。

Q：人脸检测返回值问题？

A：一般反馈OK(0)表示检测到合格的人脸，当传入检测数据间隔时间较长上，超过了追踪的时间，会返回DATA_HIT_LAST(9)。所有返回9也是检测到了合格的人脸，如下所示：

```
public static enum ErrCode {
    OK,
    PITCH_OUT_OF_DOWN_MAX_RANGE,
    PITCH_OUT_OF_UP_MAX_RANGE,
    YAW_OUT_OF_LEFT_MAX_RANGE,
    YAW_OUT_OF_RIGHT_MAX_RANGE,
    POOR_ILLUMINATION,
    NO_FACE_DETECTED,
    DATA_NOT_READY,
    DATA_HIT_ONE,
    DATA_HIT_LAST,
    IMG_BLURED,
    OCCLUSION_LEFT_EYE,
    OCCLUSION_RIGHT_EYE,
    OCCLUSION_NOSE,
    OCCLUSION_MOUTH,
    OCCLUSION_LEFT_CONTOUR,
    OCCLUSION_RIGHT_CONTOUR,
    OCCLUSION_CHIN_CONTOUR,
    FACE_NOT_COMPLETE,
    UNKNOW_TYPE;

    private ErrCode() {
    }
}
```

Q：授权失败？

A：一个序列号只能对应一台设备，一个设备可以绑定多个序列号，测试期间的序列号有使用时间，会过期，过期后需要到AI平台上进行延期，正式使用的序列号是永久授权的。授权不过，人脸SDK将无法返回正确结果。

Q：so加载问题？

A：很多开发者反馈找不到so库，原因是前面只提供了armeabi-v7a的库，但开发者基本加了其他第三方的库arm64-v8a、armeabi、armeabi-v7a和x86等都加进去了。so的加载原理是先加载当前CPU对应的so库，比如64位的手机会先加载arm64-v8a，只有在没有arm64-v8a目录才会去其他目录（如armeabi-v7a）下找，所有就算只留个空arm64-v8a目录也不行，因为这样他只会在arm64-v8a目录下找，这就要求每个目录下的so齐全一致。同时也不能把armeabi-v7a里面的so拷到其他目录，不要看名字一样。同时加入arm64-v8a和armeabi-v7a库。这样会导致打出来的包大不少。所以如果觉的包太大，只留armeabi-v7a是可以，他兼容其他cpu架构。**注意：aar里面可能包含so，注意检查。**

Android-广告屏分析工程

版本日志

日期	版本	更新说明
2019.07.24	V1.1	更新底层SDK； 增加批量鉴权接口； 开放RGB活体检测接口；开放证件照比对接口
2019.01.10	V1.0	人脸广告屏分析方案初版

1. 简介

您好，欢迎使用百度人脸广告屏解决方案。

1.1 方案概述

此方案提供完整的线下广告智能投放、效果监测等一整套解决方案，颠覆传统的粗放投放模式，实现品效合一、数据驱动，赋能广告主科学制定投放决策，助力线下媒体提升点位价值。

1.2 功能介绍

1.2.1 受众检测与识别

离线识别SDK，通过分析线下媒体的摄像头拍摄的视频图像，利用人脸检测、质量检测、人脸识别等功能，为视频中出现的每个受众建立唯一的受众ID。同时，离线识别SDK可以对检测到的每个受众人脸进行抠图，通过与在线API配合，将抠出的人脸图上传云端，建立人脸库，可以实现对受众的管理和运营。

1.2.2 受众画像分析

离线识别SDK，通过对线下媒体的摄像头拍摄的视频图像进行处理，实时分析得到视频中出现单个或多个受众的画像。画像内容包括：受众ID、年龄、性别、称谓、表情、眼镜、观看时长、关注度。结合地理位置、时间等信息，建立完备的受众画像，实现线下广告投放精细化和个性化。

1.2.3 投放效果分析

离线识别SDK，通过对指定时段内线下媒体的摄像头拍摄的视频图像进行分析，监测该时段内，线下媒体所在区域的人流热度，智能分析受众的动作和行为，并进行语义化解析，形成多维度投放效果报表，帮助广告主改善广告内容，优化投放决策。

1.3 名词解释

受众画像中，称谓由年龄和性别的结果而产生，具体产生标准为：

表1 称谓产生的标准

性别	年龄	称呼
男	$0 < X \leqslant 12$	小正太
	$12 < X \leqslant 20$	小哥哥
	$20 < X \leqslant 35$	大哥哥
	$35 < X \leqslant 50$	欧巴
	>50	大叔
女	$0 < X \leqslant 12$	小萝莉
	$12 < X \leqslant 20$	美少女
	$20 < X \leqslant 35$	小姐姐
	$35 < X \leqslant 50$	欧尼
	>50	阿姨

- 观看行为：系统检测到人脸后，该人脸头部上下（Pitch）、左右（Yaw）偏转角度在一定范围之内，视为一次观看行为。其中，上下（Pitch）、左右（Yaw）偏转角度的默认范围均为-15°~15°，此范围可根据实际情况进行调整。
- 观看时长：受众所有观看行为的持续时间，单位为秒。
- 关注度：观看时长/检测时长*100%
- 触达人数：从摄像头视频流中检测出的人脸，经过去重后，对应的受众数量。
- 观看人数：触达人数中有观看行为的受众数量。
- 检测时长：系统检测到人脸并进行人脸跟踪的全部持续时间，单位为秒。
- 关注度：观看时长/检测时长*100%
- 深度观看人数：观看度大于某个值的受众数量，默认值为50%，可根据实际情况进行调整。
- 点位热度：衡量广告媒体点位的流量热度，其计算公式为：**点位热度= 触达人数*0.3 + 观看人数*0.3 + 深度观看人数*0.4**

1.4 应用场景

1.4.1 线下媒体广告投放

无网状态离线运行，精准分析广告触达人数、观看人数、收视率等指标，建立流量漏斗模型，利用数据帮助广告主科学优化投放策略。

1.4.2 广告精准营销

获取线下媒体屏前受众画像，打通线上线下数据，基于画像完成千人千面的广告精准投放。

1.4.3 有效剔除非真人图片造成的数据影响

对广告效果进行监播时，利用活体检测能力，剔除掉背景中的人物图片，人物海报等非真人图片造成的影响，使得监播的数据更为真实可靠。

1.4.4 提升新零售场景业务效率

零售店、酒店、商场等多种新零售场景，打通广告营销系统与场景业务系统，借助人脸识别完成酒店认证核验、人脸支付等业务，提升新零售场景的业务效率。

1.5 方案优势

- 多人脸场景：提供基于多人脸场景的各项人脸检测和识别功能，能够同时支持对4-8人进行处理。
- 离线/在线通用：能够在离线环境完成各项人脸检测和识别功能，也支持与在线API联动，以建立更大的受众库。
- 业务功能自定义：可以自定义观看行为等业务功能的标准，极大拓展跨场景的适用性。

2、集成指南

2.1 准备工作

在正式集成前，需要做一些准备工作，完成一些账号、应用及配置，具体如下：

2.1.1 注册开发者

- STEP1：点击百度AI开放平台导航右侧的控制台，页面跳转到百度云登录界面，登录完毕后，将会进入到百度云后台，点击「控制台」进入百度云控制台页面；您也可以在官网直接点击免费试用，登录完毕后将自动进入到百度云控制台。

- STEP2：使用百度账号完成登录，如您还未持有百度账户，可以点击此处注册百度账户。
- STEP3：进入百度云欢迎页面，填写企业和个人基本信息，注册完毕，至此成为开发者。（如您之前已经是百度云用户或百度开发者中心用户，STEP3 可略过。）
- STEP4：进入百度云控制台，找到人工智能相关服务面板。
- STEP5：点击进入「人脸识别」模块。

2.1.2 获取序列号

点击“离线识别SDK管理”选项，进入离线SDK管理页面。在此页面，用户将获得五个免费试用的序列号，每个序列号经过激活后，可以在一台设备上使用一个离线SDK。用户可以点击“下载SDK”按钮，下载离线识别SDK。

完成授权请参考 激活教程

2.1.3 在线批量鉴权

采用在线批量鉴权方案，请线下联系百度与您对接的商务接口人

2.2 集成逻辑

2.2.1 阈值选择

1. 观看行为定义

头部偏转方向	阈值
上下（Pitch）	-15°~15°
左右（Pitch）	-15°~15°

2. 遮挡检测

指标	说明	阈值
left_eye	取值范围[0~1]，0为无遮挡，1是完全遮挡	0.6
right_eye	取值范围[0~1]，0为无遮挡，1是完全遮挡	0.6
nose	取值范围[0~1]，0为无遮挡，1是完全遮挡	0.6
mouth	取值范围[0~1]，0为无遮挡，1是完全遮挡	0.6
left_cheek	取值范围[0~1]，0为无遮挡，1是完全遮挡	0.6
right_cheek	取值范围[0~1]，0为无遮挡，1是完全遮挡	0.6
chin_contour	取值范围[0~1]，0为无遮挡，1是完全遮挡	0.6

3. 质量检测

指标	说明	阈值
illumination	取值范围[0~255]，脸部光照的灰度值，0表示光照不好。以及对应客户端SDK中，YUV的Y分量。	40
blurdegree	取值范围[0~1]，0是最清晰，1是最模糊。	0.6
completeness	0或1，0为人脸溢出图像边界，1为人脸都在图像边界内。	0

2.3 安卓集成说明

2.3.1 工程配置

1. 修改包名app->build.gradle->android->defaultConfig -> 您申请license时填的包名

```
defaultConfig {
        applicationId ""
        minSdkVersion 15
        targetSdkVersion 26
        versionCode 1
        versionName "1.0"

        ndk {
            moduleName "facesdk"
            ldLibs "log"
            abiFilters "armeabi-v7a" , "x86", "arm64-v8a" // "armeabi", "x86", "arm64-v8a"
        }
    }
```

2. 配置权限

```
 <uses-permission android:name="android.permission.CAMERA" />
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.MOUNT_UNMOUNT_FILESYSTEMS" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.hardware.camera.autofocus" />
<uses-permission android:name="android.permission.WRITE_SETTINGS" />
<uses-permission android:name="android.permission.READ_PHONE_STATE" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.GET_TASKS" />
```

2.3.2 FaceAuth鉴权接口

1. 在线鉴权

备注：鉴权时需手动输入序列号，在有网络的情况下进行鉴权

```
 /*
* 初始化鉴权,鉴权方式:通过AIPE 序列码在线激活鉴权
*  @param context
* @param licenseKey
* @param callback
*/

public void initLicenseOnLine(final Context context, final String licenseKey, final AuthCallback callback)
```

- 方法功能：初始化鉴权，进行远程网络鉴权
- 方法参数：

字段名称	含义	参考值	类型	必传
context	当前上下文	context 上下文，传当前的 application 即可	Context	Y
licenseKey	申请的鉴权码	申请的序列码	String	Y
callback	鉴权结果 void Response(int code, String response)	code：0：成功；1：加载失败； response：结果信息	AuthCallback	Y

- 接口调用：

```
 faceAuth = new FaceAuth();
faceAuth.initLicenseOnLine(this, key, new Callback() {
    @Override
    public void onResponse(final int code, final String response) {

    }
});
```

2. 离线鉴权

```
 /**
* 初始化鉴权，鉴权方式：离线激活（通过license.zip 文件进行鉴权）
* @param context
* @param callback
*/
public void initLicenseOffLine(final Context context, final Callback callback)
```

- 方法功能：进行离线鉴权
- 方法参数：

字段名称	含义	参考值	类型	必传
context	当前上下文	context上下文，传当前的 application 即可	Context	Y
callback	鉴权结果 void Response(int code, String response)	code：0：成功；1：加载失败； response：结果信息	AuthCallback	Y

- 接口调用：

```
 faceAuth = new FaceAuth();
faceAuth.initLicenseOffLine(this, new Callback() {
    @Override
    public void onResponse(final int code, final String response) {

    }
});
```

2. 在线批量鉴权

```
 /**
 * 初始化鉴权，鉴权方式：在线批量授权
 * @param context
 * @param callback
 */
void initLicenseBatchLine(final Context context, final String licenseID, final Callback callback)
```

- 方法功能：在线批量鉴权
- 方法参数：

参数名	含义
context	当前上下文
licenseID	鉴权码 （sk）
callback	鉴权结果 void onResponse(int code, String response) code 0：成功；code 1 加载失败 response 结果信息

- 接口调用：

```
 faceAuth = new FaceAuth();
faceAuth.initLicenseBatchLine(this, "", new Callback() {
      @Override
      public void onResponse(final int code, final String response) {

      }
 });
```

2.3.3 初始化SDK

```
 // 初始化SDK
 public static BDFaceSDKManager mFaceSDKManager = BDFaceSDKManager.getInstance();
```

- 加载模型

注: 共包括 检测模型、质量检测模型、表情情绪模型、 特征提取模型、活体模型； 2.进行特征比对时必须加载 检测模型、质量检测模型、表情情绪模型、特征提取模型 3.进行活体检测时必须加载 检测模型、质量检测模型、表情情绪模型、活体检测模型

```
 // 特征提取模型， true 为加载该模型，false为不加载该模型， 默认为true
 public boolean isFeatureStatus = true;
 // 活体检测模型， true 为加载模型， false 为加载该模型
 public boolean isLiveStatus = true;
```

- 接口调用:

人脸分析、特征提取模型加载：

```
 BDFaceSDKManager.getInstance().isLiveStatus = false;
BDFaceSDKManager.getInstance().initSDK(this);
```

- 活体检测模型加载：

```
 BDFaceSDKManager.getInstance().isFeatureStatus = false;
BDFaceSDKManager.getInstance().initSDK(this);
```

2.3.4 人脸分析

```
 // 人脸分析
public void faceTrack(final Handler handler, BDFaceImageInstance imageInstance)
```

- 方法功能：实现人脸检测，获取活体分值及人脸属性

- 方法参数：

字段名称	含义	类型	必传
handler	利用消息机制进行人脸框的绘制	Handler	Y
imageInstance	将获取到的帧图片送检	BDFaceImageInstance	Y

- 接口调用：

```
 //实现人脸检测，获取活体分数及人脸属性

BDFaceSDKManager mFaceSDKManager = BDFaceSDKManager.getInstance();

private void decodeRun(final byte[] data) {
    Runnable runnable = new Runnable() {
        @Override
        public void run() {
            bdFaceImageInstance = new BDFaceImageInstance(data, CameraUtils.DEFAULT_HEIGHT,
                    CameraUtils.DEFAULT_WIDTH, BDFaceSDKCommon.BDFaceImageType.BDFACE_IMAGE_TYPE_YUV_420,
                    0, 0);
            mFaceSDKManager.faceTrack(handler, bdFaceImageInstance);
            frame_stack_counter--;
            if (bdFaceImageInstance != null){
                bdFaceImageInstance.destory();
            }
        }
    };
    handler.post(runnable);

}
```

2.3.5 证件照比对

特征值获取

```
 // 特征提取
public byte[] extractFeature(Context context, String imgPath)
```

- 方法功能：获取图像的特征数据

- 方法参数：

字段名称	含义	参考值	类型	必传
context	当前上下文	context 上下文，传当前的application即可	Context	Y
imgPath	图像数据	图片地址	String	Y

- 接口调用：

```
 BDFaceSDKManager bdFaceSDKManager = BDFaceSDKManager.getInstance();
first = bdFaceSDKManager.extractFeature(FaceFeatureActivity.this, "pic/camera1.jpg");
```

获取提征值的返回值

```
 public FaceInfo[] getFaceInfos()
{
      return faceInfos;
  }
```

- 接口调用：

```
 BDFaceSDKManager bdFaceSDKManager = BDFaceSDKManager.getInstance();
FaceInfo firstFaceInfo = bdFaceSDKManager.getFaceInfos()[0];
```

特征比对

```
 // 特征比对
public float featureCompare(byte[] first, byte[] second)
```

- 方法功能：将获取到的获取到的图像进行比对；
- 方法参数：

字段名称	含义	参考值	类型	必传
first	生活照图片	特征分值	byte[]	Y
second	证件照图片	特征分值	byte[]	Y

- 接口调用：

```
 BDFaceSDKManager bdFaceSDKManager = BDFaceSDKManager.getInstance();
bdFaceSDKManager.featureCompare(first, second)
```

2.3.6 活体检测

```
 // 活体检测
public void getFaceLive(Context context, String imagePath)
```

- 方法功能：通过此方法检测静默活体的分值
- 方法参数：

字段名称	含义	参考值	类型	必传
context	当前上下文	context 上下文，传当前的application即可	Context	Y
imgPath	图像数据	图片地址	String	Y

- 接口调用：

```
 bdFaceSDKManager.getFaceLive(mContext, "pic/camera1.jpg");
```

2.3.7 Face Info（检测获取到的数据）

检测到的总人数、观看广告的总人数、 所有观看的总时长

```
 private int totalPeople; // 触达人数
private int viewPeople;  // 观看人数
private long totalViewTime; // 所有人观看的总时长(ms)
```

- 接口调用：

```
 // 检测后的数据

FaceInfo faceInfo = new FaceInfo();
// 触达人数
int totalPeople = faceinfo.getTotalPeople();
// 观看人数
int viewPeople = faceInfo.getViewPeople();
// 观看时长
long totalViewTime = faceInfo.getTotalViewTime();
```

2.3.8 BDFaceInfoManager

获取深度观看人数

```
/**
* 获取深度观看人数
*/
public int getDepPerson()
```

- 接口调用：

```
BDFaceInfoManager.getInstance().getDepPerson()
```

获取点位热度

```
 /**
* 点位热度
*/
public double getPointHeatdegree(FaceInfo faceInfo)
```

- 方法参数：

字段名称	含义	参考值	类型	必传
faceInfo	得到观看人数	人脸检测得到的数据	FaceInfo	Y

- 接口调用：

```
BDFaceInfoManager.getInstance().getPointHeatdegree(faceinfo);
```

获取男性人数

```
/**
 * 男性人数
 */
public int getMaleCount()
```

- 接口调用：

```
BDFaceInfoManager.getInstance().getMaleCount();
```

获取女性人数

```
/**
 * 女性人数
 */
public int getFemaleCount()
```

- 接口调用：
-

```
BDFaceInfoManager.getInstance().getFemaleCount();
```

姿态角过滤

```
/**
   * 抬头值为负值 默认为-15
   * 低头值为正值 默认为15
   * 左偏值为负值 默认为-15
   * 右偏值为正值 默认为15
   */
  public void setAttitudeAngle(int lookUp, int lowHead, int leftDeviation, int rightDeviation)
```

- 方法参数：

字段名称	含义	参考值	类型	必传
LookUp	抬头值	负值 默认为 -15	int	Y
lowHead	低头值	正值 默认为 15	int	Y
leftDeviation	左偏值	负值 默认为 -15	int	Y
rightDeviation	右偏值	正值 默认为 15	int	Y

- 接口调用：

```
BDFaceInfoManager.getInstance().setAttitudeAngle(-15,15,-15,15);
```

清空历史数据

```
/**
* 清空历史记录
*/
public void clearRecord()
```

- 接口调用：

```
BDFaceInfoManager.getInstance().clearRecord();
```

2.3.9 BDFacePerson（人脸基本信息）

faceID

```
 /**
* 人脸ID
*/
public  int getFaceID(){
 return faceID;
}
```

- 接口调用：

```
 for (BDFacePerson person : bdFacePersonList){
 person.getFaceID();
}
```

年龄

```
 /**
* 获取年龄
*/
public float getAge(){
 return age;
}
```

- 接口调用：

```
 for (BDFacePerson person  : bdFacePersonList){
 person.getAge();
}
```

性别

```
 /**
* 获取性别
*/
 public int getGender() {
        return gender;
    }
```

- 接口调用：

```
 for (BDFacePerson person  : bdFacePersonList){
 /**
 * 返回值为0 男性
 * 返回值为 1 为女性
 */
 person.getGender();
}
```

是否戴眼镜

```
 /**
* 是否带眼睛
*/
 public int getGlasses() {
        return glasses;
    }
```

- 接口调用：

```
 for (BDFacePerson person ：bdFacePersonList){
 /**
 * 返回值为0 不戴眼睛， 否则为戴眼睛
 */
 person.getGlasses();
}
```

情绪

```
 /**
* 情绪
*/
public int getEmotion() {
        return emotion;
    }
```

- 接口调用：

```
 for (BDFacePerson person ：bdFacePersonList){
 /**
 * 返回值为0 生气
 * 返回值为1 恶心
 * 返回值为2 恐惧
 * 返回值为3 开心
 * 返回值为4 伤心
 * 返回值为5 惊讶
 * 返回值为6 中性
 */
 person.getEmotion();
}
```

关注度

```
  /**
  * 关注度
  */
    public int getGazePrecent()
```

- 接口调用：

```
 for (BDFacePerson person ：bdFacePersonList){
 person.getGazePrecent();
}
```

观看时长

```
  /**
  * 返回个人观看时间
  */
  public long getViewTime()
```

- 接口调用：

```
 for (BDFacePerson person ：bdFacePersonList){
 person.getViewTime();
}
```

2.3.10 FaceManager

获取指定时间段内的监播数据

```
 /**
   * 获取指定时间段内的监播数据 * @param start 开始时间(ms) * @param end 结束时间(ms)
   *
   * @return
   */
  public FaceInfo getRange(long start, long end) {
```

- 方法参数：

字段名称	含义	类型	必传
start	指定时间的开始时间	long	Y
end	指定时间的结束时间	long	Y

- 接口调用:

```
  bdFaceManager = FaceManager.getInstance();
FaceInfo faceInfoI = bdFaceManager.getRange(mFirstTime, System.currentTimeMillis());
```

获取当前时间往前推 playTime 内的监播信息

```
/**
   * 获取当时时间往前推 playTime 内的监播信息 * @param playTime
   *
   * @return
   */
  public FaceInfo getRange(long playTime)
```

- 方法参数：

字段名称	含义	类型	必传
playTime	往前的推的时间	long	Y

- 接口调用：

```
 // 向前推3秒内的监播信息
bdFaceManager = FaceManager.getInstance();
FaceInfo faceInfo = bdFaceManager.getRange(3000);
```

Win-SDK-C++（历史版本）

版本日志

版本	日期	更新说明
v1.1.0	2018.08.31	1、增加离线证件照特征抽取模型，可有效处理芯片照、证件照比对需求 2、增加离线人脸属性模型，支持性别、年龄等属性分析 3、增加深度图活体检测支持，适配奥比中光Astra Mini/Mini S、华捷艾米深度镜头模组 4、图片入参支持buffer二进制模式 5、其他细节优化及已知bug修复
v1.0.0	2018.06.29	初版，包括离线人脸采集、离线活体检测、离线对比识别、离线人脸库管理等功能

目录

1、简介
1.1 SDK基础信息
1.1.1 产品概述
1.1.2 规格信息
1.1.3 兼容性
1.1.4 授权方式
1.1.5 产品定价
1.2 功能介绍
1.2.1 人脸检测与追踪
1.2.2 质量控制
1.2.3 人脸采集
1.2.4 离线RGB可见光活体检测
1.2.5 离线NIR近红外活体检测
1.2.6 离线Depth深度图像活体检测（3D结构光）
1.2.7 离线1：1对比
1.2.8 离线1：N搜索
1.2.9 离线人脸库管理
1.3 业务逻辑
1.3.1 通用流程概述
1.3.2 活体检测
1.3.3 人脸对比
1.3.4 人脸搜索
1.4 设备选型
1.4.1 镜头模组选择
1.4.2 机器选择
1.5 方案选型
1.5.1 使用离线SDK
1.5.2 使用采集SDK+API
1.5.3 使用离线SDK+API
2、SDK详细介绍
2.1 名词解释
2.2 SDK简介
2.3 SDK文件结构
2.4 激活工具
2.5 Demo示例工程
2.6 特征抽取模型选择
3、功能接口
3.1 人脸检测及设置
3.1.1 人脸检测track接口(传入图片)
3.1.2 人脸检测track接口(传入二进制图片buffer)
3.1.3 人脸检测track_max_face接口
3.1.4 人脸检测track_max_face接口(传入二进制图片buffer)
3.1.5 人脸检测设置接口
3.1.6 USB摄像头检测
3.1.7 人脸检测track接口(传入opencv的mat)
3.2 人脸管理接口
3.2.1 人脸注册接口
3.2.2 人脸注册接口（传入二进制图片buffer）
3.2.3 人脸更新接口

3.2.4 人脸更新接口（传入二进制图片buffer）
3.2.5 人脸删除接口
3.2.6 用户删除接口
3.2.7 创建用户组接口
3.2.8 用户组删除
3.2.9 用户信息查询接口
3.2.10 用户组列表查询接口
3.2.11 组列表查询接口
3.3 人脸对比及识别接口
3.3.1 人脸对比接口
3.3.2 人脸对比接口（传入二进制图片buffer）
3.3.3 人脸识别identify接口
3.3.4 人脸识别identify接口（传入二进制图片buffer）
3.3.5 特征值提取接口（通过传入图片）
3.3.6 特征值提取接口（通过传入图片）
3.3.7 特征值提取接口（通过传入opencv的视频帧）
3.3.8 特征值比较接口
3.4 活体检测接口
3.4.1 近红外(NIR)活体检测接口（通过传入图片）
3.4.2 近红外(NIR)活体检测接口（通过传入二进制图片buffer）
3.4.3 近红外(NIR)活体检测接口（通过传入opencv的视频帧）
3.4.4 可见光(RGB)活体检测接口（通过传入图片）
3.4.5 可见光(RGB)活体检测接口（通过传入二进制图片buffer）
3.4.6 可见光(RGB)活体检测接口（通过传入opencv的视频帧）
3.4.7 可见光(RGB)&近红外(NIR)活体检测接口（通过传入二进制图片buffer）
3.4.8 可见光(RGB)&近红外(NIR)活体检测接口（通过传入opencv视频帧）
3.4.9 可见光(RGB)&深度(Depth)活体检测接口（通过传入二进制图片buffer）
3.4.10 可见光(RGB)&深度(Depth)活体检测接口（通过传入opencv视频帧）
3.5 属性及质量接口
3.5.1 人脸属性（通过传入图片）
3.5.2 人脸质量接口（通过传入图片）
4、错误码及错误信息
5、常见问题

1、简介

1.1 SDK基础信息

1.1.1 产品概述

人脸离线识别SDK，包含人脸采集、活体检测、人脸对比/识别、人脸库管理等能力，并全部离线化、本地化。此SDK一经授权激活，可完全在无网环境下工作，所有数据皆在设备本地运行处理，可根据业务需要进行灵活的上层业务开发。核心能力分布如下图所示，后文会详细介绍。

适用场景特点

- **网络**：无网、局域网等情况，无法连接公网。如政府单位、金融保险、教育机构等。
- **安全**：行业特点所带来的人脸数据敏感性，即使可以连接公网也不可请求。
- **速度**：由于各地网络线路、机房部署等诸多原因，网络请求速度存在不可控因素。
- **稳定**：需要尽可能避免网络抖动、机房故障等影响，进一步控制可用性影响因素。

1.1.2 规格信息

- 包大小：~ 600M
- 最小人脸检测大小：50px * 50px
- 可识别人脸角度：yaw ≤ ±30°, pitch ≤ ±30°
- 检测速度：100ms 720p*
- 追踪速度：30ms 720p*
- 人脸检测耗时：< 100ms

- RGB图片特征抽取耗时：< 300ms
- RGB活体检测耗时：< 200ms
- 近红外活体检测耗时：< 50ms
- 1万本地人脸库检索速度：< 400ms

> 备注：以上指标，由最新版SDK运行在真实设备上，采用真实数据集所得，但**算法性能受实际运行设备、实际数据集等情况影响**，以上数字仅供参考。

- SDK采用动态库dll方式提供
- 提供一个鉴权激活工具：LicenseTool.exe

1.1.3 兼容性

- x86、x64两个版本，支持Win7、Win10
- 推荐基于vs2015进行开发

1.1.4 授权方式

按设备授权

离线识别SDK授权方式为以设备维度为主，每台硬件设备需要一个独立的授权，此授权的校验是基于设备的硬件指纹（指纹的获取SDK初始化时会自动读取并展示），被授权的设备，将在有效期内可以运行SDK。

重新拉取授权的情况：设备授权不变，仅需要重新激活而已。

- 删除SDK或基于SDK开发的应用
- 重新安装Windows系统

授权失效的情况：需要重新购买序列号，之前的序列号失效。

- 激活一台设备后，此设备硬件变更
- 硬件损坏

序列号

序列号为管理授权的依据。每台被授权的设备，都将对应一个序列号，用于标识对应的设备信息及授权记录。序列号的形式为16位随机英文数字组合，如：`3G59-M5JK-889A-7LQA`。您在管理后台购买SDK授权时，购买成功后系统将会发放对应数量的序列号。序列号不限制平台版本，任何开发平台的离线SDK，都可以使用此序列号激活。序列号不限制账号，可供任何设备激活使用。

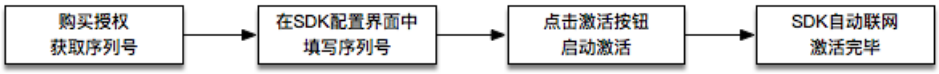
激活

已购买的序列号，是用于激活的唯一凭证，激活流程主要是将序列号与具体的硬件进行绑定（硬件指纹），从而生成对应硬件设备的授权文件（License），SDK运行前，将会校验授权文件是否和实际硬件信息相匹配。

> **注意**：激活时，设备系统时间需要和当前时间一致，如果差距太大（例如偏差5min以上），激活则无法完成。

联网激活

此种激活方式，适用于设备可首次联网的情况，优势在于激活过程极为简单。您只需将SDK安装到需要激活的设备上，然后填写已经购买的序列号，在界面上点击激活即可（为使用方便，我们为您设计了一个简单的激活用户界面）

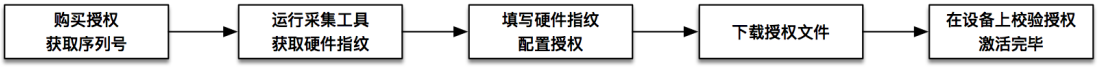


1. 获取序列号：从管理后台购买获取序列号。
2. SDK中填写序列号：将序列号填写到SDK的可视化界面中。
3. 启动激活：点击「激活」按钮。
4. SDK自动激活：SDK自动拉取授权文件并完成授权，激活完毕。

如激活成功，将会立即在界面上有明确的弹窗提示，请留意查看；如激活失败，也会反馈具体的错误信息。

离线激活

此种激活方式，适用于设备完全不可联网的情况，优势在于可避免联网激活，满足业务对网络的严格要求，以及设备批量注册需求。您需要在后台配置好硬件指纹并完成和序列号的绑定，然后将授权文件放到SDK的指定位置。



1. 获取序列号：从管理后台购买获取序列号。
2. 采集硬件指纹：将SDK置于设备上，运行激活程序，获取硬件指纹。
3. 配置授权：在后台将硬件指纹绑定到具体序列号上。
4. 下载授权文件：绑定成功后下载授权文件。
5. 设备激活：将授权文件放到SDK中，并初始化SDK完成授权。

授权有效期

申请通过后，每个账户给**2个测试序列号**，用于激活及SDK试用，有效期为**自激活日期后的3个月**。这两个序列号在有效期内完全免费，您可以用于进行产品试用。试用期到期后也可以在后台申请延期，填写具体延期理由即可。

正式购买

正式购买的序列号，试用期限为**永久有效**。此「永久」是指绑定到具体设备维度，但如已绑定的硬件设备变更后，授权则可能会失效。

1.1.5 定价方式

离线SDK的授权基于设备维度，每个序列号仅可以授权一台设备。每个账号购买的序列号会累计计算，**累计购买量**越多，单价越便宜。具体如下所示：

累计购买授权数量	每个授权单价
0~1000	299元/个
1001~5000	249元/个
>5000	199元/个

立即去购买

1.2 功能介绍

1.2.1 人脸检测与追踪

可在设备端，离线实时检测视频流中的人脸。同时支持处理静态图片或者视频流，并对当前检测到的人脸持续跟踪，动态定位

人脸轮廓，稳定贴合人脸。

1.2.2 质量控制

在人脸检测及追踪过程中，实时校验人脸的姿态角度、遮挡、清晰度、光照条件，符合质量条件的人脸图片才会被采集。

1.2.3 人脸采集

针对视频流实时完成人脸图片采集，并输出满足质量过滤条件的人脸图片，可自定义采集人脸大小，采集频率，采集质量等设置。

1.2.4 离线RGB可见光活体检测

针对视频流/图片，通过采集人像的破绽（摩尔纹、成像畸形等）来判断目标对象是否为活体，可有效防止屏幕二次翻拍等作弊攻击，可使用单张或多张判断逻辑。

1.2.5 离线NIR近红外活体检测

针对视频流/图片，利用近红外成像原理，实现夜间或无自然光条件下的活体判断。其成像特点（如屏幕无法成像，不同材质反射率不同等）可以实现高鲁棒性的活体判断。

1.2.6 离线Depth深度图像活体检测（3D结构光）

通过3D建模判断目标对象是否为活体，基于3D结构光成像原理，可强效防御图片、视频、屏幕、模具等攻击。

1.2.7 离线1：1对比

提供本地化的1：1人脸对比功能，高鲁棒性算法，可对应各种姿态、肤色、光照等场景，可有效应用于人证比对、身份核验等场景。

示例工程中包含：

- 图片与图片的比对：两张人脸图片的1：1对比，并返回相似度分值。
- 图片与视频流比对：一张预设的人脸图片，和摄像头实时采集的符合条件的人脸图片进行对比。

1.2.8 离线1：N搜索

本地数据库中保留所有人脸特征值（如需要保留原图，可根据业务需要自行修改工程）。

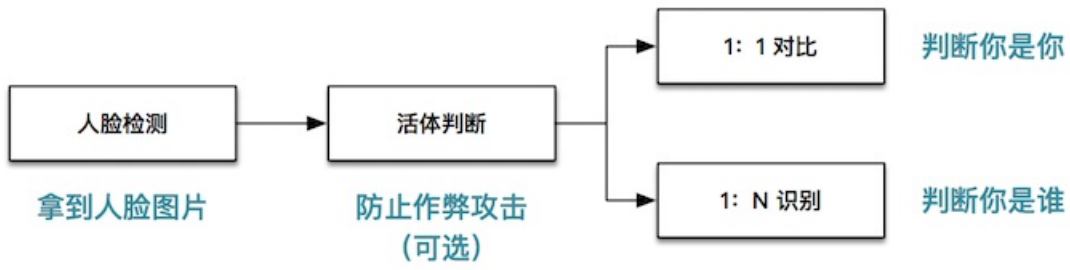
- 视频流采集的人脸在人脸库中搜索：视频流中实时采集人脸，并与人脸库中预设的人脸库进行一一对比，返回相似度最高的user及对应分值。

1.2.9 离线人脸库管理

人脸数量不做上限，可根据业务需要适当调整，支持人脸库、人脸组、用户、Face几个维度的增删改查设置。

1.3 业务流程

1.3.1 通用流程概述



如上图所示，人脸识别的核心业务流程可以分为三个步骤。

1. 检测采集：通过视频流实时检测跟踪，并采集到符合质量要求的人脸图片，用于后续的识别。
2. 活体判断：为可选步骤，主要保障业务操作者为真人，避免业务作弊。加上这步的校验，即只有满足活体判断通过，人脸图片才会被采集。
3. 对比识别：1：1对比主要是判断「你是你」，用于核实身份的真实性；1：N搜索主要判断「你是谁」，用于明确身份的具体所属。

1.3.2 活体检测

> 提示：此版本仅支持RGB可见光活体、NIR近红外活体。Depth深度活体将会尽快推出

1.3.2.1 基础原理

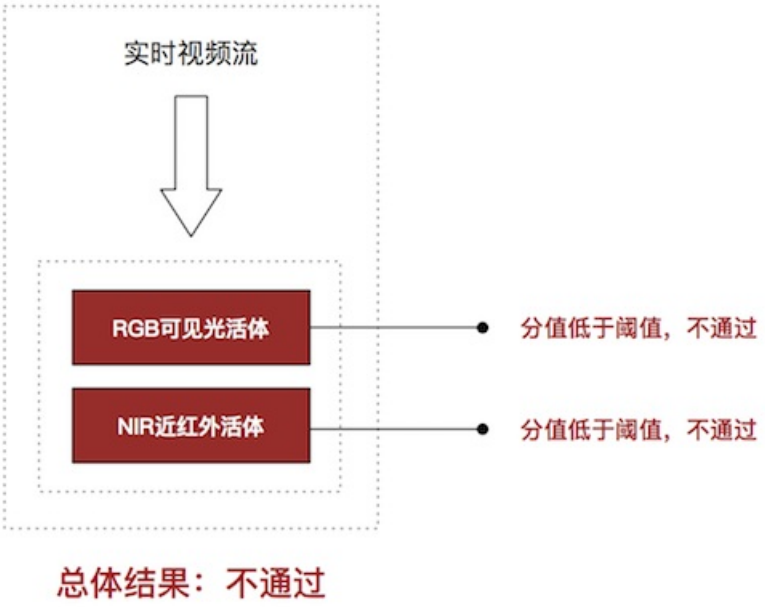
- RGB可见光活体：主要基于图片破绽，判断目标对象是否为活体。例如图像中的屏幕反光、成像畸形等，最主要的应用情形为屏幕的二次翻拍等攻击防御。此种活体对于待检测图片的要求，主要需要满足画面中除了人脸以外，要尽可能保留一些背景内容，用于查找破绽，通常建议人脸与屏幕的长宽比为1：3。为控制达到此比例，建议通过调整最小检测人脸参数，控制采集的人脸不可过大，避免人脸面积占比过高，而导致图片中没有多少背景信息。同时RGB活体受光线影响较大，所以在强光、暗光等场景，容易数值波动较大，主要影响的是「通过率」，产品策略上需要通过适当的补光、使用宽动态镜头抵消逆光等方式缓解。
- NIR近红外活体：主要基于近红外光线反射成像原理，通过人脸呈现来判断是否为活体。即使是夜间或者没有自然光的情况下，依然可以判断活体。因为其成像特点，对于屏幕、图片等攻击形式，基本可以达到近似于100%的活体防御。同时近红外设备的成本，相对性价比更高。
- Depth深度活体（3D结构光）：结构光原理是通过主动光发射，在物体上形成光栅，并接受此信息进行活体分析。同样可以不需要自然光。3D结构光的成像更为稳定，抗攻击能力更强，对图像噪声的抗干扰能力也更强，是相对更加安全和稳定的方案，但相应的硬件设备造价也较高，需要根据实际业务成本预算，进行综合考虑。

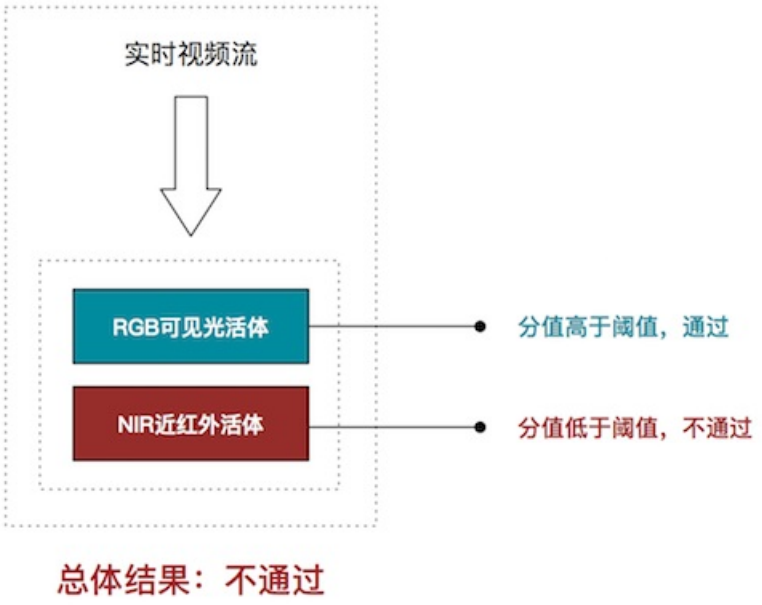
以上，无论活体是否可以通过，我们业务的最终目的，还是要进行「对比识别」，所以**拿到合适的RGB图像还是最为关键**，保障符合条件的RGB图像获取也是需要在活体判断同时，最为需要关注的事情。

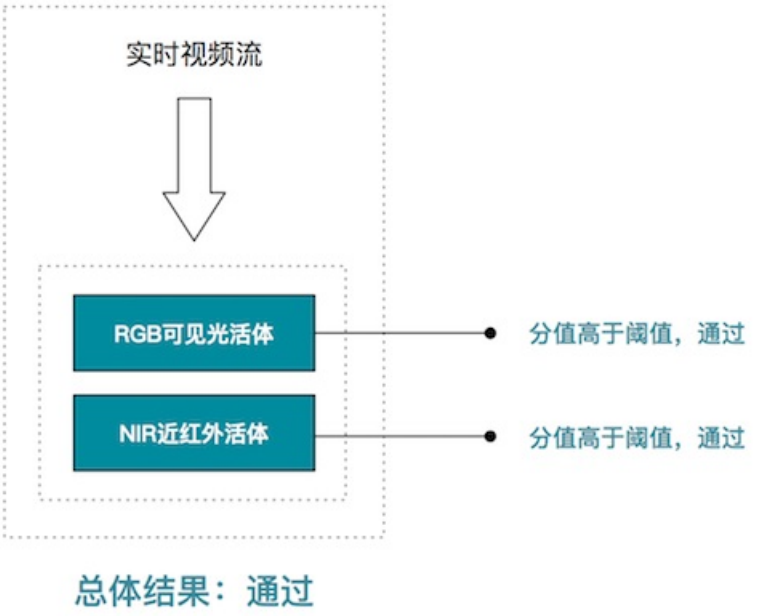
在实际业务场景中，需要根据场景特点，灵活组合使用以上几种活体方案。当然，实际业务中，**没有绝对100%的安全**，在应用过程中，还需要根据业务流程特点，指定一系列的辅助措施，如证件信息读取、密码、其他生物特征识别等，达到刚安全的核验。

1.3.2.2 产品应用策略

活体判断逻辑简单理解为：满足活体条件才可以采集图片，否则一直反复判断，直到满足活体条件为止。因为我们最终送去识别的图片是RGB图片，所以需要**保证活体通过时，在同一时间采集RGB图片，才可真正防止作弊攻击**。当多种活体叠加使用时，需要满足所有活体都通过，才能出发此操作，如果有任一活体没有通过，都不可进入识别步骤。







1.3.2.3 场景及应用方案

- 通行场景：此场景通常保障通行速度为主，确保不影响通行秩序和效率。所以建议无需使用三重活体检测，可仅用NIR活体或Depth活体，保障效率同时仍可保证安全性。
- 身份核验场景：此场景通常保障业务安全性为主，可尽可能提供更加安全的活体方案。如RGB+NIR，或者RGB+Depth，乃至RGB+NIR+Depth，最大程度确保对攻击的拒绝率较高。
- 强光/暗光场景：光线较强的场景，RGB活体会受影响比较严重，建议使用NIR或者Depth活体，同时尽量通过产品策略避免这两种情况的光线，例如添加补光灯、配备遮光板等。

1.3.2.4 应用方案及模组选择

- 无需活体：如有人值守的场景下，活体检测并无太大的必要（活体检测也会增加额外的业务耗时），现有设备的单目USB摄像头即可。
- 仅用RGB可见光活体：如果您的设备已经配备了USB单目摄像头，则无需更换，输出的RGB图像可直接用于RGB可见光活体算法识别。
- RGB可见光+NIR近红外活体：需要配备能够同时获取RGB、NIR近红外数据的镜头使用。
- RGB可见光+Depth深度活体：需要配备能够同时获取RGB、Depth深度图像数据的镜头使用。
- RGB可见光+NIR近红外+Depth深度活体：此种方式安全度最高，目前SDK正在模组适配中，**后续会陆续推出已适配的模组方案，敬请期待。**

1.3.2.4 指标

活体检测存在几个标准的指标，如下所示：

- 拒绝率（TRR）：如99%，代表100次作弊假体攻击，会有99次被拒绝。
- 误拒率（FRR）：如0.5%，指1000次真人请求，会有5次因为活体分数低于阈值被错误拒绝。
- 通过率（TAR）：如99%，指100次真人请求，会有99次因为活体分数高于阈值而通过。
- 阈值（Threshold）：高于此数值，则可判断为活体。

温馨提示：此SDK涉及多种离线活体检测模型，加上镜头模组效果各异，使用环境也较为复杂，以下给出综合测试值，仅供参考：

- 拒绝率：> 99.5%
- 误拒率：< 1%
- 通过率：> 99%

1.3.2.4 阈值选择推荐

活体分值区间为[0~1]，大部分情况的活体攻击，活体分值近似于接近0.0，建议阈值如下，高于此阈值的即可判断为活体。

- RGB可见光活体：0.8
- NIR近红外活体：0.8
- Depth深度活体（3D结构光）：0.8（后续版本即将支持）

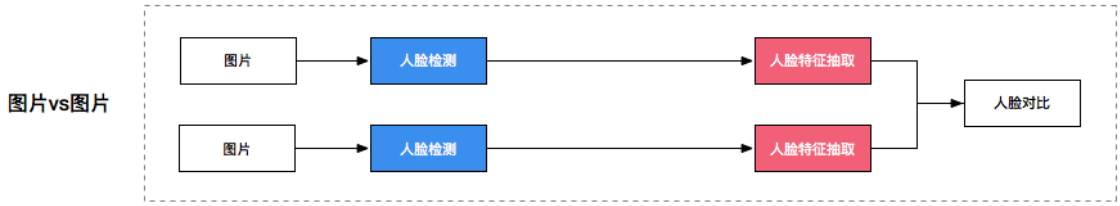
1.3.3 人脸1：1对比

人脸对比分为两种常见形态，具体如下：

1.3.3.1 对比类型：图片vs图片

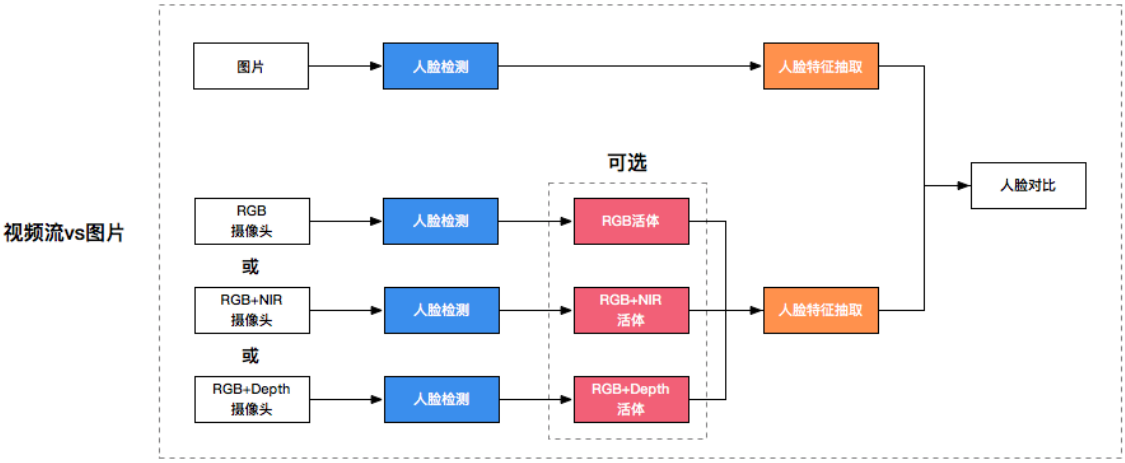
如果有两张待对比的图片（如证件图片、以事先获取到的人脸生活照等），则可以直接通过离线SDK进行对比，业务流程如下

图所示：



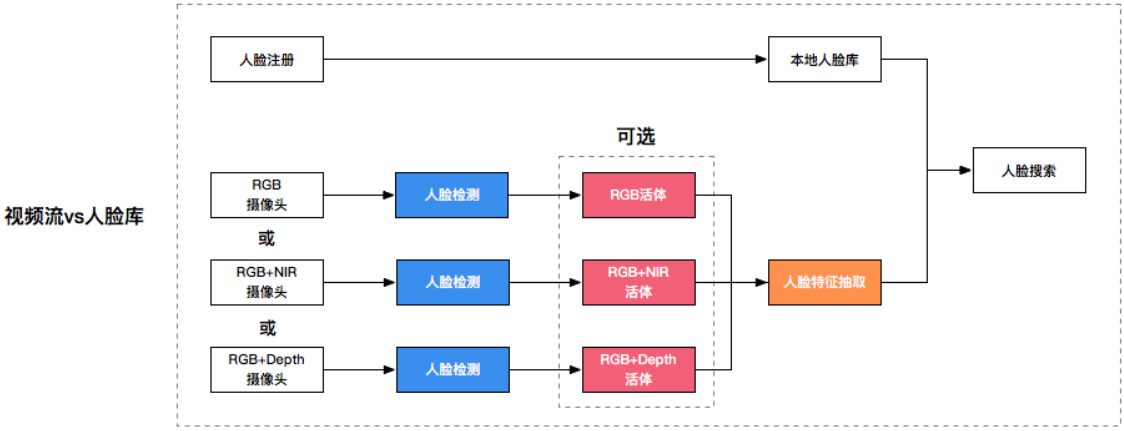
1.3.3.2 对比类型：实时视频流vs图片

这是最为常见的1：1对比类型，一张事先获取的图片（通常为身份证芯片照、证件照片等），与现场人脸采集的图片进行对比。如果需要为无人看守，则需要配备活体检测保障业务真实性和安全性。



1.3.4 人脸1：N搜索

将需要识别的人脸图片集注册到本地人脸库中，当有用户需要识别身份时，从视频流中实时采集人脸图片，与人脸库中的人脸集合对比，得到搜索结果。如果需要为无人看守，则需要配备活体检测保障业务真实性和安全性。



1.4 设备选型

1.4.1 镜头模组选择

1.4.1.1 无活体或仅用RGB单目活体

uvc免驱单目USB摄像头，推荐**视派尔**单目摄像头，默认为3mm焦距镜头，可以选择6mm镜头，可以检测识别更远的人脸。

- **视派尔C-EP28WD400**，[产品规格书]

1.4.1.2 RGB可见光+NIR近红外活体

推荐使用如下品牌的近红外摄像头，可根据场景举例选择合适焦距摄像头。

- **视派尔C-EP35WDLDIR**，[产品规格书] | [去AI市场购买]
- **迪威泰DV-BD4044S305AD**，[产品规格书]；联系人：杨先生；联系电话：18643209187
- **迪威泰DV-BD4053S305AD**，[产品规格书]；联系人：杨先生；联系电话：18643209187

NIR近红外实际检测效果回显示例：

1.4.1.3 RGB可见光+Depth深度活体（3D结构光）

推荐使用如下品牌的深度摄像头，可根据场景举例选择合适焦距摄像头。

- **奥比中光Astra Mini / Mini S**，[产品规格书]；[AI市场购买链接]
- **华捷艾米A100S+MINI**，[产品规格书]；[AI市场购买链接]

1.4.1.4 是否可以使用其他品牌镜头？

如果您需要使用NIR近红外或者Depth活体，则一定要配备支持对应图像格式类型的镜头模组。目前百度离线SDK适配的镜头类型有限，难以覆盖市面上所有双目模组。您可以查看下方的图像适配指标文档，只需满足文档中的图像/数据要求，即可自行进行模组适配。

[镜头模组图像适配指标]

我们也诚挚欢迎各镜头模组厂商参与到SDK的合作中来，我们会积极与您进行产品、技术、商务等多方面对接。测评效果较佳的模组，将会作为SDK官方适配模组，并在所有技术资料中优先推荐用户使用。您可以将合作申请发送到下方咨询邮箱，我们接到您的申请后会尽快与您取得联系，感谢您的信任。

合作邮件：ai#baidu.com（ #替换成@符号 ）

1.4.2 机器选择

- 系统：Windows 7、Windows 10系统的设备
- 处理器：推荐Intel i3及以上
- 内存：4.00 GB及以上
- 系统类型：32位、64位
- 具备2个及以上的USB插口

- 安装vs2015环境（不保证其他vs版本的兼容性）

1.5 方案选型

离线SDK具备离线人脸检测、跟踪、质量校验、图像采集、RGB/NIR/3D结构光活体、离线对比识别、离线人脸库管理。可基本cover人脸识别所需要的全部能力，仅需一个SDK即可完成业务能力覆盖。但相对来说仍然具备一定的场景及业务使用限制。

目前AI开放平台也同时开放采集SDK和API能力，建议您根据实际的业务需要，选择合适的产品应用方案，以达到最佳的应用效果。

1.5.1 使用离线SDK

以下特点及场景，适合仅适用人脸离线SDK

适用的场景特点

- 网络条件不稳定
- 无网络
- 数据安全性高，不允许联网
- 人脸库较小，且变更不频繁，如1w人以下
- 单台设备、单人脸检测与识别

> 注意：此版本离线SDK暂不支持多线程处理，主要适用于单人脸核验场景，不建议用于多人脸抓拍识别。

典型场景及设备类型

- 人脸门禁
- 人脸闸机
- 人证核验机
- 自助柜机
- 人脸考勤机
- 会场签到

> 提示：对于离线1：N场景，需要您基于SDK构造上层的人脸库数据更新业务逻辑，例如小区门口的闸机为纯离线，但是住户的变更信息，如何同步到每一个大门的闸机中。

1.5.2 使用采集SDK+API接口

以下特点及场景，适合使用采集SDK+API接口

适用的场景特点

- 网络条件良好
- 人脸库量级庞大
- 需要跨地域同步人脸库
- 人脸库更新频繁

- 用户APP产品形态

> 提示：采集SDK具备本地化的人脸检测跟踪、质量校验、人脸采集等功能，与离线SDK相比，活体和识别操作使用云端服务完成。

典型场景及设备类型

- 零售会员识别
- 人脸支付
- 手机APP
- 移动考勤
- 远程开户

1.5.3 使用离线SDK+API接口

以下特点及场景，适合使用离线SDK+API接口

适用的场景特点

- 网络条件较好
- 人脸库较小，但变更频繁
- 业务常为集中时段的高并发（如考勤）

典型场景及设备类型

- 上下班刷脸考勤
- 社区闸机
- 楼宇门禁

离线人脸库处理固定人员的识别请求，将在本地抵消掉很多API请求量；同时API处理变动人员的请求，方便及时同步更新，从而总体节省QPS资源的成本消耗。

2、SDK详细介绍

2.1 名词解释

名词	定义
SDK	Windows离线人脸识别sdk
vs2015	微软的开发工具visual studio 2015 (推荐安装vs community 2015)
License	人脸识别激活所需要的激活文件,可利用激活工具生成
key	人脸激活所需的序列号，可从百度官网申请及购买（ai.baidu.com）
feature	人脸特征值，用来表示人脸特征的512个浮点值
face_token	对应人脸图片的唯一编码，若一个人上传了2张不同图片，则可能有2个不同的face_token,它和图片一一对应

2.2 SDK简介

本sdk适应于windows平台下的人脸识别系统,分x86和x64两个版本,开发者可在vs2015下面进行开发（推荐使用，不保证vs其他版本都兼容）。Sdk采用动态库dll方式提供。

另外随sdk附带一个鉴权激活工具（LicenseTool.exe），此激活工具为采用微软MFC开发,通过该激活工具可生成正常接入sdk的

激活license文件(生成两个文件license.ini和license.key)达到通过鉴权，正常使用sdk的目的。

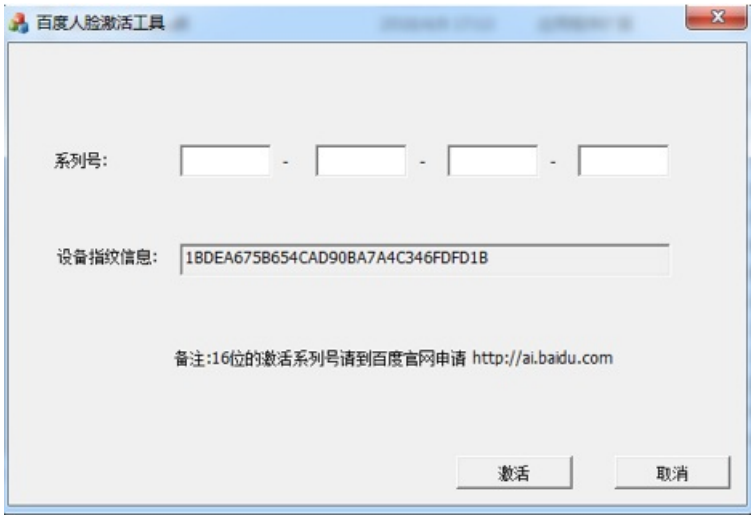
2.3 SDK文件结构

Sdk提供动态库BaiduFaceApi.dll、BaiduFaceApi.lib及头文件baidu_face_api.h。另外有附带的demo工程TestFaceApi以及鉴权激活工具LicenseTool.exe。在此之外，还附带支撑SDK使用的人脸识别模型文件等在文件夹face-resource中，该文件夹在TestFaceApi例子工程里面(该文件夹存放路径请参考TestFaceApi工程，默认存放路径为您要开发的exe(如demo工程TestFaceApi.exe)所在路径的上级目录下。若存放不对，可能会影响sdk正常使用，另外请勿删除该文件夹)。

为运行exe，需要用到一些底层的库文件(dll)，相关库文件在TestFaceApi的win32和x64文件夹(分别对应x86和x64平台)中均已提供。此外，随TestFaceApi还有一些辅助文件或开源库，如opencv及头文件，base64文件，这些都是开源文件，随demo工程可以根据需要可自行修改，另外一些示例demo文件，如人脸注册、删除等人脸库管理类manager，人脸检测、活体检测及识别类livensess，人脸比较类compare，人脸参数设置类setting等的头文件及cpp文件。

2.4 激活工具（LicenseTool.exe）

在百度官网申请或购买序列号后，在TestFaceApi的win32或x64(由您编译的版本决定)目录下，双击运行LicenseTool.exe激活工具，可见如下所示界面。在输入框中输入16位的系列号后，点击激活按钮，稍等片刻，即可见弹出的激活成功对话框，若激活失败，请参考对话框中弹出的message信息查找原因。激活成功后会在激活目录生成license.ini以及license.key文件，这2个文件是作为sdk通过鉴权使用的配置文件，请勿删除。



如果你不方便使用网络，可以通过离线激活进行操作：

通过如上的激活工具，获取到设备指纹信息，通过序列号管理后台找到需要绑定的序列号，选择「离线激活」，填入指纹设备信息，即可下载获取到license.ini文件和license.key文件，把这2个文件拷贝到TestFaceApi的Win32或x64目录下，运行TestFaceApi.exe亦可通过鉴权。

2.5 Demo示例工程

demo示例工程TestFaceApi展示了如何集成百度人脸识别离线sdk。即导入BaiduFaceApi.dll,BaiduFaceApi.lib及头文件BaiduFaceApi.h。另外为了示例视频人脸跟踪等，用到了一些opencv的库文件以及一些实现demo的支持文件，如image_base64等。（这些支持文件均为代码开源或是开源库）

在TestFaceApi中的test_face.cpp的main()方法中，有使用sdk的各个接口方法示例。接入sdk及其简单，分3步3行代码。如下：

```
 BaiduFace *api = new BaiduFaceApi();
// 第一步：实例化人脸SDK

api->sdk_init();
// 第二步：初始化人脸SDK

std::string res = api->user_add("方法的传入参数，此处省略");
// 第三步，调用功能接口，user_add为人脸注册接口，res为调用功能接口后的返回。
```

如上，即为调用sdk功能的最简单3步3行代码，当然调用SDK后，在程序退出后，需要释放sdk防止内存泄漏，需要删除sdk实例化的指针`delete api;`

示例工程中：分别有以下几个示例cpp文件对应几个常用sdk的调用demo。

文件名	功能
setting.cpp	人脸检测、识别等参数设置
compare.cpp	人脸1:1比对、1:N 搜索，人脸特征值提取，特征值对比等
liveness.cpp	USB摄像头视频信息人脸实时检测，可见光RGB、红外NIR活体检测、RGB\NIR 双目摄像头活体检测等
manager.cpp	人脸库管理类，包括人脸注册、删除，更新、组管理，人脸信息查询等
cv_help.cpp	绘制人脸跟踪框、人脸关键点位等的工具类

2.6 特征抽取模型选择

模型介绍

v1.1版本起，SDK提供生活照和证件照两种特征抽取模型，主要适用场景如下:

- **生活照模型**：如手机拍摄的图片、较为清晰的证件照片、USB镜头实时采集的图片、网络摄像头实时采集的图片等。
- **证件照模型**：如身份证芯片照、各类证件照（工卡、学生卡、会员卡照片等）、人脸的像素普遍小于80px的图片等。

使用方法

在SDK初始化方法中控制：sdk_init(true); //**传入true为使用证件照模型，传入fasle为普通生活照模型**

注意事项

温馨提示：一经选择一个模型，则所有业务流程的特征抽取处理，都会使用此模型，两个模型不可同时作用。**如业务中设计证件照的特征抽取，请务必选择使用证件照模型**。

3、功能接口

SDK实现的主要功能有人脸实时跟踪检测、人脸特征值提取、RGB\IR活体检测、人脸注册、人脸更新、组管理、用户管理以及1:1人脸对比,1:N人脸识别、特征值的比对和通过USB或笔记本自带摄像头检测视频帧，返回识别出的人脸信息等，另外支持对人脸检测进行设置，达到根据设置进行识别的目的。各接口功能及传入参数和返回结果（返回结果一般为json格式的字符串）描述如下：

3.1 人脸检测及设置

3.1.1 人脸检测track接口(传入图片)

方法名

`track`

方法说明

人脸检测，返回人脸信息

请求信息

参数	必须	类型	描述
out	是	std::vector	**返回的检测到的人脸图片信息结构体** std::vector *& out 通过传入TrackFaceInfo的结构体指针引用，在检测到人脸信息后，用该引用返回人脸信息。TrackedFaceInfo结构体为检测到的人脸信息，其中：box为检测到的人脸框（为FaceInfo结构体,包含人脸框大小，中心点坐标，人脸宽度等信息）landmarks为检测到的人脸关键点，一般为144个。score为人脸打分headPose的向量组为人脸x,y,z的三个角度
image	是	string	人脸图片信息，根据image_type，传入图片内容
img_type	是	int	传入的图片类型。为1时候表示base64编码的图片，为2时候表示传入图片的本地路径。BASE64：图片的base64值；FACE_FILE：图片的本地文件路径地址；
maxTrackObjNum	是	int	最多检测人脸数量，默认为1，最大不超过5

返回信息

返回结果为int，表示检测到几个人脸信息，1为1个人，0为没检测到人脸。

3.1.2 人脸检测track接口(传入二进制图片buffer)

方法名

`track_by_buf`

方法说明

人脸检测，返回人脸信息

请求信息

参数	必须	类型	描述
out	是	std::vector	**返回的检测到的人脸图片信息结构体** std::vector *& out 通过传入TrackFaceInfo的结构体指针引用，在检测到人脸信息后，用该引用返回人脸信息。TrackedFaceInfo结构体为检测到的人脸信息，其中：box为检测到的人脸框（为FaceInfo结构体,包含人脸框大小，中心点坐标，人脸宽度等信息）landmarks为检测到的人脸关键点，一般为144个。score为人脸打分headPose的向量组为人脸x,y,z的三个角度
image	是	Unsigned char *	人脸图片信息，二进制图片buffer内容
size	是	int	二进制图片的大小
maxTrackObjNum	是	int	最多检测人脸数量，默认为1，最大不超过5

返回信息

返回结果为int，表示检测到几个人脸信息，1为1个人，0为没检测到人脸。

3.1.3 人脸检测track_max_face接口

方法名

`track_max_face`

方法说明

检测最大人脸。

请求信息

参数	必须	类型	描述
out	是	std::vector<TrackFaceInfo>	**返回的检测到的人脸图片信息结构体** std::vector *& out 通过传入TrackFaceInfo的结构体指针引用，在检测到人脸信息后，用该引用返回人脸信息。TrackedFaceInfo结构体为检测到的人脸信息，其中：box为检测到的人脸框（为FaceInfo结构体,包含人脸框大小，中心点坐标，人脸宽度等信息）；landmarks为检测到的人脸关键点，一般为144个；score为人脸打分；headPose的向量组为人脸x,y,z的三个角度
image	是	string	图片信息（数据大小应小于10M）
image_type	是	int	为1时候表示base64编码的图片；为2时候表示传入图片的本地路径。 BASE64：图片的base64值；FACE_FILE：图片的本地文件路径地址；

返回信息

返回结果为int，表示检测到几个人脸信息，1为1个人，0为没检测到人脸。

3.1.4 人脸检测track_max_face接口(传入二进制图片buffer)

方法名

`track_max_face_by_buf`

方法说明

检测最大人脸。

请求信息

参数	必须	类型	描述
out	是	std::vector<TrackFaceInfo>	**返回的检测到的人脸图片信息结构体** std::vector *& out 通过传入TrackFaceInfo的结构体指针引用，在检测到人脸信息后，用该引用返回人脸信息。TrackedFaceInfo结构体为检测到的人脸信息，其中：box为检测到的人脸框（为FaceInfo结构体,包含人脸框大小，中心点坐标，人脸宽度等信息）；landmarks为检测到的人脸关键点，一般为144个；score为人脸打分；headPose的向量组为人脸x,y,z的三个角度
image	是	Unsigned char *	人脸图片信息，二进制图片buffer内容
size	是	int	二进制图片的大小

返回信息

返回结果为int，表示检测到几个人脸信息，1为1个人，0为没检测到人脸。

3.1.5 人脸检测设置接口

请参考`TestFaceApi`工程中的代码示例及头文件`BaiduFaceApi.h`中的定义及`setting.cpp`里的代码注释。

3.1.6 USB摄像头检测

通过打开usb摄像头，返回实时检测到的视频帧人脸信息。请参考`TestFaceApi`中的示例`liveness.cpp`中的`usb_track_face_info`，该方法中用到了人脸检测track视频帧，接口如下3.1.7。

3.1.7 人脸检测track接口(传入opencv的mat)

方法名

`Track`

方法说明

人脸检测，返回人脸信息。

请求信息

参数	必须	类型	描述
out	是	`std::vector<TrackFaceInfo>`	**返回的检测到的人脸图片信息结构体** std::vector *& out 通过传入TrackFaceInfo的结构体指针引用，在检测到人脸信息后，用该引用返回人脸信息。TrackedFaceInfo结构体为检测到的人脸信息，其中：box为检测到的人脸框（为FaceInfo结构体,包含人脸框大小，中心点坐标，人脸宽度等信息）；landmarks为检测到的人脸关键点，一般为144个；score为人脸打分；headPose的向量组为人脸x,y,z的三个角度
Mat	是	Opencv格式的单帧图片mat	人脸图片信息
maxTrackObjNum	是	int	最多检测人脸数量。默认为1，最大不超过5

返回信息

返回结果为TrackedFaceInfo的向量指针,向量组为0时候表示没检测到人脸。TrackedFaceInfo结构体为检测到的人脸信息，其中：box为检测到的人脸框（为FaceInfo结构体,包含人脸框大小，中心点坐标，人脸宽度等信息）；landmarks为检测到的人脸关键点，一般为144个；score为人脸打分；headPose的向量组为人脸x,y,z的三个角度。

3.2 人脸管理接口

3.2.1 人脸注册接口

方法名

`user_add`

方法说明

用户注册，该接口支持传入base64编码的图片或传入图片文件路径进行注册。如下所述，image_type为1时候表示传入base64编码的图片，为2时候表示传入本地图片文件地址

请求信息

参数	必须	类型	示例描述
user_id	是	string	用户id，字母、数字、下划线组成，最多128个字符
group_id	是	string	用户组id，标识一组用户（由数字、字母、下划线组成），长度限制128B。用户组和user_id之间，仅为映射关系。如传入的groupid并未事先创建完毕，则注册用户的同时，直接完成group的创建
image	是	string	图片信息，须小于10M
image_type	是	int	图片类型。1：为图片的base64值；2：图片的本地文件地址
user_info	否	string	用户资料，字母、数字、下划线组成，256个字符以内

返回信息

errno信息：

- errno=0：Success
- errno>0：错误码对应的详细msg

data信息:

字段	类型	说明
log_id	string	请求日志标识
face_token	string	注册图片对应的face_token

3.2.2 人脸注册接口（传入二进制图片buffer）

方法名

`user_add_by_buf`

方法说明

用户注册，该接口支持传入二进制的图片buffer。

请求信息

参数	必须	类型	示例描述
user_id	是	string	用户id，字母、数字、下划线组成，最多128个字符
group_id	是	string	用户组id，标识一组用户（由数字、字母、下划线组成），长度限制128B。用户组和user_id之间，仅为映射关系。如传入的groupid并未事先创建完毕，则注册用户的同时，直接完成group的创建
image	是	Unsigned char*	二进制图片内容、须小于10M
size	是	int	二进制图片大小
user_info	否	string	用户资料，256个字符以内

返回信息

errno信息：

- errno=0：Success
- errno>0：错误码对应的详细msg

data信息:

字段	类型	说明
log_id	string	请求日志标识
face_token	string	注册图片对应的face_token

3.2.3 人脸更新接口

方法名

`user_update`

方法说明

人脸更新

请求信息

参数	必须	类型	示例描述
user_id	是	string	用户id（由数字、字母、下划线组成），长度限制128B
group_id	是	string	用户组id，标识一组用户（由数字、字母、下划线组成），长度限制128B
image	是	string	图片信息，数据大小应小于10M，每次仅支持单张图片
image_type	是	int	图片类型，必选择以下2种形式之一。image_type为1代表BASE64为2代表FACE_FILE。 **BASE64**：图片的base64值； **FACE_FILE**：图片的本地文件路径地址
user_info	否	string	用户资料，长度限制256个字符以内

返回信息

errno信息：

- errno=0：Success
- errno>0：错误码对应的详细msg

data信息:

字段	类型	说明
log_id	string	请求日志标识
face_token	string	注册图片对应的face_token

3.2.4 人脸更新接口（传入二进制图片buffer）

方法名

`user_update_by_buf`

方法说明

人脸更新，该接口支持传入二进制的图片buffer。

请求信息

参数	必须	类型	示例描述
user_id	是	string	用户id（由数字、字母、下划线组成），长度限制128B
group_id	是	string	用户组id，标识一组用户（由数字、字母、下划线组成），长度限制128B
image	是	Unsigned char *	二进制图片内容、须小于10M
size	是	int	二进制图片大小
user_info	否	string	用户资料，256个字符以内

返回信息

errno信息：

- errno=0：Success
- errno>0：错误码对应的详细msg

data信息:

字段	类型	说明
log_id	string	请求日志标识
face_token	string	注册图片对应的face_token

3.2.5 人脸删除接口

方法名

`user_face_delete`

方法说明

人脸删除

请求信息

参数	必须	类型	示例描述
user_id	是	string	用户id（由数字、字母、下划线组成），长度限制128B
group_id	是	string	用户组id，标识一组用户（由数字、字母、下划线组成），长度限制128B
face_token	是	string	人脸id（由数字、字母、下划线组成）长度限制128B

返回信息

errno信息：

- errno=0：Success
- errno>0：错误码对应的详细msg

data信息:

字段	类型	说明
log_id	string	请求日志标识

3.2.6 用户删除接口

方法名

`user_delete`

方法说明

用户删除

请求信息

参数	必须	类型	示例描述
user_id	是	string	用户id（由数字、字母、下划线组成），长度限制128B
group_id	是	string	用户组id，标识一组用户（由数字、字母、下划线组成），长度限制128B

返回信息

errno信息：

- errno=0：Success
- errno>0：错误码对应的详细msg

data信息:

字段	类型	说明
log_id	string	请求日志标识

3.2.7 创建用户组接口

方法名

`group_add`

方法说明

创建用户组

请求信息

参数	必须	类型	示例描述
group_id	是	string	用户组id，标识一组用户（由数字、字母、下划线组成），长度限制128B

返回信息

errno信息：

- errno=0：Success
- errno>0：错误码对应的详细msg

data信息:

字段	类型	说明
log_id	string	请求日志标识

3.2.8 用户组删除

方法名

`group_delete`

方法说明

用户组删除

请求信息

参数	必须	类型	示例描述
group_id	是	string	用户组id，标识一组用户（由数字、字母、下划线组成），长度限制128B

返回信息

errno信息：

- errno=0：Success
- errno>0：错误码对应的详细msg

data信息:

字段	类型	说明
log_id	string	请求日志标识

3.2.9 用户信息查询接口

方法名

`get_user_info`

方法说明

用户信息查询

请求信息

参数	必须	类型	示例描述
user_id	是	string	用户id（由数字、字母、下划线组成），长度限制128B
group_id	是	string	用户组id，标识一组用户（由数字、字母、下划线组成），长度限制128B

返回信息

errno信息：

- errno=0：Success
- errno>0：错误码对应的详细msg

data信息:

字段	类型	说明
log_id	string	请求日志标识
result	array	识别结果列表
group_id	string	组id
face_token	string	人脸特征的唯一标识
user_info	string	用户信息
create_time	string	人脸首次注册时间

3.2.10 用户组列表查询接口

方法名

`get_user_list`

方法说明

用户组列表查询

请求信息

参数	必须	类型	示例描述
group_id	是	string	用户组id
start	否	int	默认值为0
length	否	int	默认值100，最大值1000

返回信息

errno信息：

- errno=0：Success
- errno>0：错误码对应的详细msg

data信息:

字段	类型	说明
log_id	string	请求日志表识
user_id_list	array	user_id列表数组

3.2.11 组列表查询接口

方法名

`get_group_list`

方法说明

组列表查询

请求信息

参数	必须	类型	示例描述
start	否	int	默认值为0，从0开始
length	否	int	默认值为100，最大为1000

返回信息

errno信息：

- errno=0：Success
- errno>0：错误码对应的详细msg

data信息:

字段	类型	说明
log_id	string	请求日志标识
group_id_list	array	group_id列表数组

3.3 人脸对比及识别接口

3.3.1 人脸对比接口

方法名

`match`

方法说明

人脸对比接口（本接口中的image_type为1表示base64图片编码，为2表示文件路径，为3表示face_token）

请求信息

参数	必须	类型	示例描述
image1	是	string	需要对比的第一张图片，小于10M
imgae1_type	是	int	图片1类型，必选择以下三种形式之一 **BASE64**：图片的base64值； **FACE_FILE**：图片的本地文件路径地址； **FACE_TOKEN**：face_token 人脸标识；
image2	是	string	需要对比的第二张图片，小于10M
image2_type	是	int	图片2类型，必选择以下三种形式之一 **BASE64**：图片的base64值； **FACE_FILE**：图片的本地文件路径地址； **FACE_TOKEN**：face_token 人脸标识；

返回信息

errno信息：

- errno=0：Success
- errno>0：错误码对应的详细msg

data信息:

字段	类型	说明
log_id	string	请求日志标识
result	array	识别结果列表
score	string	相似度分值，0-100，百分制，保留后2位小数点

3.3.2 人脸对比接口（传入二进制图片buffer）

方法名

`match_by_buf`

方法说明

人脸对比接口（传入二进制图片buffer）

请求信息

参数	必须	类型	示例描述
image1	是	Unsigned char *	需要对比的第一张图片，小于10M
size	是	int	图片1的大小
image2	是	Unsigned char *	需要对比的第二张图片，小于10M
size	是	int	图片2的大小

返回信息

errno信息：

- errno=0：Success
- errno>0：错误码对应的详细msg

data信息:

字段	类型	说明
log_id	string	请求日志标识
result	array	识别结果列表
score	string	相似度分值，0-100，百分制，保留后2位小数点

3.3.3 人脸识别identify接口

方法名

`identify`

方法说明

人脸识别，提供1：N查找 (本接口中的image_type为1表示base64图片编码，为2表示文件路径，为3表示face_token)

请求信息

参数	必须	类型	示例描述
image	是	string	图片信息，数据大小小于10M
image_type	是	int	图片类型，必选择以下三种形式之一 **BASE64**：图片的base64值； **FACE_FILE**：图片的本地文件路径地址； **FACE_TOKEN**：face_token 人脸标识；
group_id_list	是	string	组id列表。默认至少填写一个group_id，从指定的group中进行查找。需要同时查询多个group，用逗号分隔，上限10个
user_id	是	string	用户id，若指定了某个user，则只会与指定group下的这个user进行对比；若user_id传空字符串""，则会与此group下的所有user进行1：N识别
user_top_num	否	int	识别后返回的用户top数，默认为1，最多返回50个结果

返回信息

errno信息：

- errno=0：Success
- errno>0：错误码对应的详细msg

data信息:

字段	类型	说明
log_id	string	请求日志标识
face_token	string	传入识别的image的face_token值
result_num	int	返回结果数量
result	array	识别结果列表
group_id	string	用户组id
user_id	string	用户id
score	string	相似度分值,0-100,百分制，保留小数点后两位

3.3.4 人脸识别identify接口（传入二进制图片buffer）

方法名

`identify_by_buf`

方法说明

人脸识别，提供1：N查找 (传入二进制图片buffer)

请求信息

参数	必须	类型	示例描述
image	是	Unsigned char *	二进制图片信息，数据大小小于10M
size	是	int	图片大小
group_id_list	是	string	组id列表。默认至少填写一个group_id，从指定的group中进行查找。需要同时查询多个group，用逗号分隔，上限10个
user_id	是	string	用户id，若指定了某个user，则只会与指定group下的这个user进行对比；若user_id传空字符串”“，则会与此group下的所有user进行1：N识别
user_top_num	否	int	识别后返回的用户top数，默认为1，最多返回50个结果

返回信息

errno信息：

- errno=0：Success
- errno>0：错误码对应的详细msg

data信息:

字段	类型	说明
log_id	string	请求日志标识
face_token	string	传入识别的image的face_token值
result_num	int	返回结果数量
result	array	识别结果列表
group_id	string	用户组id
user_id	string	用户id
score	string	相似度分值,0-100,百分制，保留小数点后两位

3.3.5 特征值提取接口（通过传入图片）

方法名

`get_face_feature`

方法说明

提取人脸特征值，为512个浮点值，已加密 (本接口 image_type 为1表示base64图片编码，为2表示文件路径)

请求信息

参数	必须	类型	示例描述
image	是	string	图片信息，数据大小小于10M
image_type	是	Int	图片类型，必选择以下2种形式之一(image_type值为1或2，分别对应 BASE64：图片的base64值； FACE_FILE：图片的本地文件路径地址；)
feature	是	`const float*`	提取特征值，通过传入const float*指针的引用，来返回提取的人脸特征值

返回信息

int 如512，正常提取成功的特征值向量个数，若不为512则为返回的错误码。

3.3.6 特征值提取接口（通过传入二进制图片buffer）

方法名

`get_face_feature_by_buf`

方法说明

提取人脸特征值，为512个浮点值，已加密 (本接口 image_type 为1表示base64图片编码，为2表示文件路径)

请求信息

参数	必须	类型	示例描述
image	是	Unsigned char *	二进制图片信息，数据大小小于10M
size	是	int	二进制图片大小
feature	是	`const float *`	提取特征值，通过传入const float*指针的引用，来返回提取的人脸特征值

返回信息

int 如512，正常提取成功的特征值向量个数，若不为512则为返回的错误码。

3.3.7 特征值提取接口（通过传入opencv的视频帧）

方法名

`get_face_feature`

方法说明

提取人脸特征值，为512个浮点值，已加密 (本接口 image_type 为1表示base64图片编码，为2表示文件路径)

请求信息

参数	必须	类型	示例描述
Mat	是	Cv::Mat	Opencv视频帧，即需要提取特征值的视频帧（单帧）
feature	是	const float*	提取的特征值，通过传入const float*指针的引用，来返回提取的人脸特征值

返回信息

int 如512，正常提取成功的特征值向量个数，若不为512则为返回的错误码。

3.3.8 特征值比较接口

方法名

compare_feature

方法说明

对人脸特征值进行比较，可返回人脸特征相似分值（百分制）

请求信息

参数	必须	类型	示例描述
feature1	是	std::vector<float>	512个浮点型的特征值，传入const的特征值引用
feature2	是	std::vector<float>	512个浮点型的特征值，传入const的特征值引用

返回信息

返回值	类型	说明
score	float	比较结果，百分制的分值

3.4 活体检测接口

3.4.1 近红外(NIR)活体检测接口（通过传入图片）

方法名

ir_liveness_check

方法说明

人脸红外活体检测接口（image_type 为1表示base64图片编码 为2表示文件路径）

请求信息

参数	必须	类型	示例描述
image	是	string	需要检测的图片，小于10M, 图片类型根据image_type参数定
imgae_type	是	int	图片类型，必选择以下2种形式之一。image_type为1代表BASE64为2代表FACE_FILE。 **BASE64**：图片的base64值； **FACE_FILE**：图片的本地文件路径地址

返回信息

errno信息：

- errno=0：Success
- errno>0：错误码对应的详细msg

data信息:

字段	类型	说明
score	string	活体检测分值，范围为0-1，实际为浮点型，开发者可自行转换为百分制。

3.4.2 近红外(NIR)活体检测接口（通过传入二进制图片buffer）

方法名

`ir_liveness_check_by_buf`

方法说明

人脸近红外活体检测接口（传入二进制图片buffer）

请求信息

参数	必须	类型	示例描述
image	是	Unsigned char *	需要检测的图片，二级制格式，小于10M
size	是	int	二进制图片大小

返回信息

errno信息：

- errno=0：Success
- errno>0：错误码对应的详细msg

data信息:

字段	类型	说明
score	string	活体检测分值，范围为0-1，实际为浮点型，开发者可自行转换为百分制。

3.4.3 近红外(NIR)活体检测接口（通过传入opencv的视频帧）

方法名

`ir_liveness_check`

方法说明

人脸红外活体检测接口（本接口传入opencv的视频帧mat）

请求信息

参数	必须	类型	示例描述
Mat	是	Cv::Mat	需要检测的视频帧（单帧）

返回信息

errno信息：

- errno=0：Success
- errno>0：错误码对应的详细msg

data信息:

字段	类型	说明
score	string	活体检测分值，范围为0-1，实际为浮点型，开发者可自行转换为百分制。

3.4.4 可见光(RGB)活体检测接口（通过传入图片）

方法名

`rgb_liveness_check`

方法说明

人脸可见光活体检测接口（image_type 为1表示base64图片编码 为2表示文件路径）

请求信息

参数	必须	类型	示例描述
image	是	string	需要检测的图片，小于10M,图片类型根据image_type参数定
imgae_type	是	int	图片类型，必选择以下2种形式之一。 Image_type为1代表BASE64为2代表FACE_FILE。 **BASE64**：图片的base64值； **FACE_FILE**：图片的本地文件路径地址；

返回信息

errno信息：

- errno=0：Success
- errno>0：错误码对应的详细msg

data信息:

字段	类型	说明
score	string	活体检测分值，范围为0-1，实际为浮点型，开发者可自行转换为百分制。

3.4.5 可见光(RGB)活体检测接口（通过传入二进制图片buffer）

方法名

`rgb_liveness_check_by_buf`

方法说明

人脸可见光活体检测接口（image_type 为1表示base64图片编码 为2表示文件路径）

请求信息

参数	必须	类型	示例描述
image	是	Unsigned char *	需要检测的图片，二级制格式，小于10M
size	是	int	二进制图片大小

返回信息

errno信息：

- errno=0：Success

- errno>0：错误码对应的详细msg

data信息:

字段	类型	说明
score	string	活体检测分值，范围为0-1，实际为浮点型，开发者可自行转换为百分制。

3.4.6 可见光(RGB)活体检测接口（通过传入opencv的视频帧）

方法名

`rgb_liveness_check`

方法说明

人脸可见光活体检测接口（本接口传入opencv的视频帧mat）

请求信息

参数	必须	类型	示例描述
mat	是	Cv::Mat	需要检测的视频帧（单帧）

返回信息

errno信息：

- errno=0：Success
- errno>0：错误码对应的详细msg

data信息:

字段	类型	说明
score	string	活体检测分值，范围为0-1，实际为浮点型，开发者可自行转换为百分制。

3.4.7 可见光(RGB)&近红外(NIR)活体检测接口（通过传入二进制图片buffer）

方法名

`rgb_ir_liveness_check_by_buf`

方法说明

可见光和近红外（RGB & NIR）同时活体检测

请求信息

参数	必须	类型	示例描述
rgb_img	是	Unsigned char*	可见光图片的二进制图片信息（rgb），须小于10M
rgb_size	是	int	图片大小
ir_img	是	Unsigned char*	近红外图片的二进制图片信息（ir），须小于10M
ir_size	是	int	图片大小

返回信息

errno信息：

- errno=0：Success

- errno>0：错误码对应的详细msg

data信息:

字段	类型	说明
rgb_score	string	RGB活体检测分值，范围为0-1，实际为浮点型，可自行转换为百分制。
Ir_score	string	NIR活体检测分值，范围为0-1，实际为浮点型，可自行转换为百分制。

3.4.8 可见光(RGB)&近红外(NIR)活体检测接口（通过传入opencv视频帧）

方法名

`rgb_ir_liveness_check`

方法说明

可见光和近红外（RGB & NIR）同时活体检测

请求信息

参数	必须	类型	示例描述
rgb_mat	是	mat	可见光（rgb）视频帧mat
ir_mat	是	Mat	红外（ir）视频帧mat

返回信息

errno信息：

- errno=0：Success
- errno>0：错误码对应的详细msg

data信息:

字段	类型	说明
rgb_score	string	RGB活体检测分值，范围为0-1，实际为浮点型，可自行转换为百分制。
ir_score	string	NIR活体检测分值，范围为0-1，实际为浮点型，可自行转换为百分制。

3.4.9 可见光(RGB)&深度(Depth)活体检测接口（通过传入二进制图片buffer）

方法名

`rgb_depth_liveness_check_by_buf`

方法说明

可见光和深度图（RGB & Depth）同时活体检测

本sdk目前适配了奥比中光的双目深度摄像头以及华捷艾米的双目深度摄像头。在sdk中分别以动态库的形式接入，如orbe_camera.dll以及hjimi_camera.dll，用于快速接入，不需要可直接移除。

另外若要适配奥比中光的摄像头，在win32下有个orbe_install目录，**该目录有个sensorDriver.exe文件需要双击安装后重启电脑，才能保证适配奥比中光摄像头成功**。该两款摄像头适配及调用demo详见liveness类的rgb_depth_track_by_mat_with_orbe方法和rgb_depth_track_by_mat_with_hjimi方法。

请求信息

参数	必须	类型	示例描述
rgb_img	是	Unsigned char*	可见光图片的二进制图片信息（rgb），须小于10M
rgb_size	是	int	图片大小
depth_img	是	Unsigned char*	深度图片的二进制图片信息（ir），须小于10M
depth_size	是	int	图片大小

返回信息

errno信息：

- errno=0：Success
- errno>0：错误码对应的详细msghttp://news.family.baidu.com/specialTopicDetail?subjectId=186429&subjectPathNodeId=2103

data信息:

字段	类型	说明
Rgb_score	string	RGB活体检测分值，范围为0-1，实际为浮点型，可自行转换为百分制。
depth_score	string	NIR活体检测分值，范围为0-1，实际为浮点型，可自行转换为百分制。

3.4.10 可见光(RGB)&深度(Depth)活体检测接口（通过传入opencv视频帧）

方法名

`rgb_ir_liveness_check`

方法说明

可见光和近红外（RGB & NIR）同时活体检测

本sdk目前适配了奥比中光的双目深度摄像头以及华捷艾米的双目深度摄像头。在sdk中分别以动态库的形式接入，如orbe_camera.dll以及hjimi_camera.dll，用于快速接入，不需要可直接移除。

另外若要适配奥比中光的摄像头，在win32下有个orbe_install目录，**该目录有个sensorDriver.exe文件需要双击安装后重启电脑，才能保证适配奥比中光摄像头成功**。该两款摄像头适配及调用demo详见liveness类的rgb_depth_track_by_mat_with_orbe方法和rgb_depth_track_by_mat_with_hjimi方法。

请求信息

参数	必须	类型	示例描述
rgb_mat	是	mat	可见光（rgb）视频帧mat
depth_mat	是	Mat	深度（depth）视频帧mat

返回信息

errno信息：

- errno=0：Success
- errno>0：错误码对应的详细msg

data信息:

字段	类型	说明
rgb_score	string	RGB活体检测分值，范围为0-1，实际为浮点型，可自行转换为百分制。
depth_score	string	深度活体检测分值，范围为0-1，实际为浮点型，可自行转换为百分制。

3.5 属性及质量接口

3.5.1 人脸属性（通过传入图片）

方法名

face_attr

方法说明

人脸属性检测接口(本接口 image_type 为1表示base64图片编码 为2表示文件路径)

请求信息

参数	必须	类型	示例描述
image	是	string	需要检测的人脸图片，小于10M, 图片类型根据image_type参数定
imgae_type	是	int	图片类型，必选择以下2种形式之一。 Image_type为1代表BASE64为2代表FACE_FILE。 **BASE64**：图片的base64值； **FACE_FILE**：图片的本地文件路径地址；

返回信息

errno信息：

- errno=0：Success
- errno>0：错误码对应的详细msg

data信息:

字段	类型	说明
age（年龄）	int	人脸年龄范围0-100，若为-1表示不完整的人脸。
race（种族）	int	正常人脸标注 0：黄种人 1：白种人 2：黑人 3：阿拉伯人，不完整的人脸标注-1
expression(表情)	int	0：中性表情，1：微笑，2：大笑
gender(性别)	int	0: 女 female，1: 男 male，-1: 婴儿或不好辨别性别
glasses(是否戴眼镜)	int	0：不带眼镜 no glasses，1：普通透明眼镜 glasses，2：太阳镜 sunGlasses

3.5.2 人脸质量接口（通过传入图片）

方法名

face_quality

方法说明

人脸质量检测接口

请求信息

参数	必须	类型	示例描述
image	是	string	需要检测的人脸图片，小于10M, 图片类型根据image_type参数定
imgae_type	是	int	图片类型，必选择以下2种形式之一。 Image_type为1代表BASE64为2代表FACE_FILE。 **BASE64**：图片的base64值； **FACE_FILE**：图片的本地文件路径地址；

返回信息

errno信息：

- errno=0：Success
- errno>0：错误码对应的详细msg

data信息:

字段	类型	说明
bluriness	float	模糊度
illum	float	光照
occl_l_eye	float	左眼遮挡度
occl_r_eye	float	右眼遮挡度
occl_nose	float	鼻子遮挡度
occl_mouth	float	嘴巴遮挡度
occl_l_contour	float	左脸遮挡度
occl_r_contour	float	右脸遮挡度
occl_chin	float	下巴遮挡度

4、错误码及错误信息

错误码	错误内容	错误描述
0	SUCCESS	成功
1	SYSTEM ERROR	系统错误
2	UNKNOWN ERROR	未知错误
1001	NOT_AUTH	授权校验失败
1002	REQUEST PARAMS ERROR	请求参数错误
1003	DB_OP_FAILED	数据库操作失败
1004	NO_DATA	没有数据
1005	RECORD_UNEXIST	记录不存在
1006	RECORD_ALREADY_EXIST	记录已经存在
1007	FILE_NOT_EXIST	文件不存在
1008	GET_FEATURE_FAIL	提取特征值失败
1009	FILE_TOO_BIG	文件太大
1010	FACE_RESOURCE_NOT_EXIST	人脸资源文件不存在
1011	FEATURE_LEN_ERROR	特征值长度错误
1012	DETECT_NO_FACE	未检测到人脸

5、常见问题

Q：为什么会报db_operation_error错误？

A：请检查sdk路径是否带中文，这会导致数据库创建失败然后提示db_operation_error

Q：编译demo工程时候出现如opencv-win\include\opencv2\flann\logger.h(66): error C4996: 'fopen': This function or variable may be unsafe的错误提示

A：这种错误一般是因为vs强制要求安全等级提高所致，可以通过右键工程-属性-C/C++ -- 预处理器—预处理器定义中加入：_CRT_SECURE_NO_WARNINGS 后重新编译即可解决问题。

Q：工程运行过程中，提示face-resource不存在。

A：人脸sdk，需要一些模型文件，在demo工程的face-resource文件夹中，该文件夹需要放置在exe所在路径的上级目录下。若放置不正确，可能出现找不到模型文件，没法进行人脸识别。

Q：激活后是否可以把激活文件license.ini和license.key拷贝到其他设备运行？

A：不能，离线sdk和设备绑定，每个设备对应一个key和一个license文件，换设备无法运行。但对同一台设备，可把win32下的license.ini和license.key拷贝到x64下，则x64环境，该设备也等同于激活，可以使用。

Q：exe不能运行或崩溃

A：编译的工程exe，需要和dll等在一个文件夹里面，如示例工程win32和x64就有不少动态库dll文件，这是exe运行所需的库文件，不放在一起的话，可能exe不能运行或者崩溃。

Q：vs2010或vs2017打开工程编译提示错误不能通过编译？

A：开发文档官方指定使用vs2015 comunity版本，因vs2010不支持c++11，但sdk使用了c++11的新功能，vs2017若编译提示json库错误，如json::reader的错误，建议更新为v1.1版本的SDK即可解决。

Q：SDK是否支持多线程？

A：v1.1版本已经支持多线程，请在后台下载最新版的SDK。

Q：Windows离线SDK是否支持C#，Java语言？

A：Windows SDK采用C++语言编写，支持用vs2015版本开发，推荐用vs2015 Comunity版本。SDK提供动态库dll，可用于实现Java调用和C#调用，预计在9-10月份，百度会推出官方版本。

Q：Windows离线sdk在debug版本报错？

A：目前仅提供x86和x64的release版本的库，不支持debug版本，需要调试可在release版本下通过添加日志调试。

Q：SDK中的激活工具licenseTool.exe和sdk的demo TestFaceApi.exe都不能运行，崩溃或提示缺少dll？

A：此SDK基于C++编写，需要安装对应运行环境（推荐使用vs2015），崩溃或提示缺少dll是因为缺少运行环境。

Q：证件照等图片检测不到人脸，但实际是有人脸的？

A：默认可检测最小人脸大小是**100**，若检测不到，可通过设置最小人脸大小调整，例如调整最小检测人脸为30px，可使用api->set_min_face_size(30)，达到调整最小人脸检测大小的目的，然后再调用检测，这样能检测到比较小得图片如证件照等。v1.1版本中已经集成了证件照模型，推荐您升级SDK版本解决此问题。

Win-SDK-C#（历史版本）

版本日志

版本	日期	更新说明
v1.1.0	2018.08.31	1、增加离线证件照特征抽取模型，可有效处理芯片照、证件照比对需求 2、增加离线人脸属性模型，支持性别、年龄等属性分析 3、增加深度图活体检测支持，适配奥比中光Astra Mini/Mini S、华捷艾米深度镜头模组 4、图片入参支持buffer二进制模式 5、其他细节优化及已知bug修复
v1.0.0	2018.06.29	初版，包括离线人脸采集、离线活体检测、离线对比识别、离线人脸库管理等功能

目录

1、简介
1.1 SDK基础信息
1.1.1 产品概述
1.1.2 规格信息
1.1.3 兼容性
1.1.4 授权方式
1.1.5 产品定价
1.2 功能介绍
1.2.1 人脸检测与追踪
1.2.2 质量控制
1.2.3 人脸采集
1.2.4 离线RGB可见光活体检测
1.2.5 离线NIR近红外活体检测
1.2.6 离线Depth深度图像活体检测（3D结构光）
1.2.7 离线1：1对比
1.2.8 离线1：N搜索
1.2.9 离线人脸库管理
1.3 业务逻辑
1.3.1 通用流程概述
1.3.2 活体检测
1.3.3 人脸对比
1.3.4 人脸搜索
1.4 设备选型
1.4.1 镜头模组选择
1.4.2 机器选择
1.5 方案选型
1.5.1 使用离线SDK
1.5.2 使用采集SDK+API
1.5.3 使用离线SDK+API
2、SDK详细介绍
2.1 名词解释
2.2 SDK简介
2.3 SDK文件结构
2.4 激活工具
2.5 Demo示例工程
2.6 特征抽取模型选择
3、功能接口
3.1 人脸检测及设置
3.1.1 人脸检测track接口(传入图片)
3.1.2 人脸检测track接口(传入二进制图片buffer)
3.1.3 人脸检测track_max_face接口
3.1.4 人脸检测track_max_face接口(传入二进制图片buffer)
3.1.5 人脸检测设置接口
3.1.6 USB摄像头检测
3.2 人脸管理接口
3.2.1 人脸注册接口
3.2.2 人脸注册接口（传入二进制图片buffer）
3.2.3 人脸更新接口
3.2.4 人脸更新接口（传入二进制图片buffer）
3.2.5 人脸删除接口
3.2.6 用户删除接口
3.2.7 创建用户组接口
3.2.8 用户组删除
3.2.9 用户信息查询接口
3.2.10 用户组列表查询接口
3.2.11 组列表查询接口
3.3 人脸对比及识别接口
3.3.1 人脸对比接口
3.3.2 人脸对比接口（传入二进制图片buffer）
3.3.3 人脸对比接口（传入二进制图片buffer和特征值feature比对）

3.3.4 人脸识别identify接口
3.3.5 人脸识别identify接口（传入二进制图片buffer）
3.3.6 特征值提取接口（通过传入图片）
3.3.7 特征值提取接口（传入二进制图片buffer）
3.3.8 特征值比较接口
3.4 活体检测接口
3.4.1 可见光(RGB)活体检测接口（通过传入图片）
3.4.2 可见光(RGB)活体检测接口（通过传入二进制图片buffer）
3.4.3 可见光(RGB)&近红外(NIR)活体检测接口（通过传入二进制图片buffer）
3.4.4 可见光(RGB)&近红外(NIR)活体检测接口（通过sdk底层打开opencv视频帧）
3.4.5 可见光(RGB)&深度(Depth)活体检测接口（通过传入二进制图片buffer）
3.4.6 可见光(RGB)&深度(Depth)活体检测接口（通过sdk底层打开opencv视频帧）
3.5 质量接口
3.5.1 人脸质量接口（通过传入图片）
3.5.2 人脸质量接口（通过传入图片二进制buffer）
4、适配的深度双目摄像头特别说明
5、错误码及错误信息
6、常见问题

1、简介

1.1 SDK基础信息

1.1.1 产品概述

人脸离线识别SDK，包含人脸采集、活体检测、人脸对比/识别、人脸库管理等能力，并全部离线化、本地化。此SDK一经授权激活，可完全在无网环境下工作，所有数据皆在设备本地运行处理，可根据业务需要进行灵活的上层业务开发。核心能力分布如下图所示，后文会详细介绍。

适用场景特点

- **网络**：无网、局域网等情况，无法连接公网。如政府单位、金融保险、教育机构等。
- **安全**：行业特点所带来的人脸数据敏感性，即使可以连接公网也不可请求。
- **速度**：由于各地网络线路、机房部署等诸多原因，网络请求速度存在不可控因素。
- **稳定**：需要尽可能避免网络抖动、机房故障等影响，进一步控制可用性影响因素。

1.1.2 规格信息

- 包大小：~ 500M
- 最小人脸检测大小：50px * 50px
- 可识别人脸角度：yaw ≤ ±30°, pitch ≤ ±30°
- 检测速度：100ms 720p*
- 追踪速度：30ms 720p*
- 人脸检测耗时：< 100ms

- RGB图片特征抽取耗时：< 300ms
- RGB活体检测耗时：< 200ms
- 近红外活体检测耗时：< 50ms
- 1万本地人脸库检索速度：< 400ms

> 备注：以上指标，由最新版SDK运行在真实设备上，采用真实数据集所得，但**算法性能受实际运行设备、实际数据集等情况影响**，以上数字仅供参考。

- SDK采用动态库dll方式提供
- 提供一个鉴权激活工具：LicenseTool.exe

1.1.3 兼容性

- x86、x64两个版本，支持Win7、Win10
- 推荐基于**vs2015**进行开发

1.1.4 授权方式

按设备授权

离线识别SDK授权方式为以设备维度为主，每台硬件设备需要一个独立的授权，此授权的校验是基于设备的硬件指纹（指纹的获取SDK初始化时会自动读取并展示），被授权的设备，将在有效期内可以运行SDK。

重新拉取授权的情况：设备授权不变，仅需要重新激活而已。

- 删除SDK或基于SDK开发的应用
- 重新安装Windows系统

授权失效的情况：需要重新购买序列号，之前的序列号失效。

- 激活一台设备后，此设备硬件变更
- 硬件损坏

序列号

序列号为管理授权的依据。每台被授权的设备，都将对应一个序列号，用于标识对应的设备信息及授权记录。序列号的形式为16位随机英文数字组合，如：`3G59-M5JK-889A-7LQA`。您在管理后台购买SDK授权时，购买成功后系统将会发放对应数量的序列号。序列号不限制平台版本，任何开发平台的离线SDK，都可以使用此序列号激活。序列号不限制账号，可供任何设备激活使用。

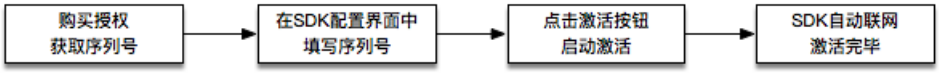
激活

已购买的序列号，是用于激活的唯一凭证，激活流程主要是将序列号与具体的硬件进行绑定（硬件指纹），从而生成对应硬件设备的授权文件（License），SDK运行前，将会校验授权文件是否和实际硬件信息相匹配。

> **注意**：激活时，设备系统时间需要和当前时间一致，如果差距太大（例如偏差5min以上），激活则无法完成。

联网激活

此种激活方式，适用于设备可首次联网的情况，优势在于激活过程极为简单。您只需将SDK安装到需要激活的设备上，然后填写已经购买的序列号，在界面上点击激活即可（为使用方便，我们为您设计了一个简单的激活用户界面）



1. 获取序列号：从管理后台购买获取序列号。

2. SDK中填写序列号：将序列号填写到SDK的可视化界面中。

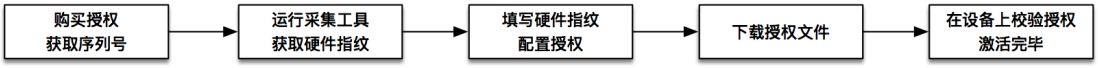
3. 启动激活：点击「激活」按钮。

4. SDK自动激活：SDK自动拉取授权文件并完成授权，激活完毕。

如激活成功，将会立即在界面上有明确的弹窗提示，请留意查看；如激活失败，也会反馈具体的错误信息。

离线激活

此种激活方式，适用于设备完全不可联网的情况，优势在于可避免联网激活，满足业务对网络的严格要求，以及设备批量注册需求。您需要在后台配置好硬件指纹并完成和序列号的绑定，然后将授权文件放到SDK的指定位置。



1. 获取序列号：从管理后台购买获取序列号。

2. 采集硬件指纹：将SDK置于设备上，运行激活程序，获取硬件指纹。

3. 配置授权：在后台将硬件指纹绑定到具体序列号上。

4. 下载授权文件：绑定成功后下载授权文件。

5. 设备激活：将授权文件放到SDK中，并初始化SDK完成授权。

授权有效期

申请通过后，每个账户给**2个测试序列号**，用于激活及SDK试用，有效期为**自激活日期后的3个月**。这两个序列号在有效期内完全免费，您可以用于进行产品试用。试用期到期后也可以在后台申请延期，填写具体延期理由即可。

正式购买

正式购买的序列号，试用期限为**永久有效**。此「永久」是指绑定到具体设备维度，但如已绑定的硬件设备变更后，授权则可能会失效。

1.1.5 定价方式

离线SDK的授权基于设备维度，每个序列号仅可以授权一台设备。每个账号购买的序列号会累计计算，**累计购买量**越多，单价越便宜。具体如下所示：

累计购买授权数量	每个授权单价
0~1000	299元/个
1001~5000	249元/个
>5000	199元/个

立即去购买

1.2 功能介绍

1.2.1 人脸检测与追踪

可在设备端，离线实时检测视频流中的人脸。同时支持处理静态图片或者视频流，并对当前检测到的人脸持续跟踪，动态定位

人脸轮廓，稳定贴合人脸。

1.2.2 质量控制

在人脸检测及追踪过程中，实时校验人脸的姿态角度、遮挡、清晰度、光照条件，符合质量条件的人脸图片才会被采集。

1.2.3 人脸采集

针对视频流实时完成人脸图片采集，并输出满足质量过滤条件的人脸图片，可自定义采集人脸大小，采集频率，采集质量等设置。

1.2.4 离线RGB可见光活体检测

针对视频流/图片，通过采集人像的破绽（摩尔纹、成像畸形等）来判断目标对象是否为活体，可有效防止屏幕二次翻拍等作弊攻击，可使用单张或多张判断逻辑。

1.2.5 离线NIR近红外活体检测

针对视频流/图片，利用近红外成像原理，实现夜间或无自然光条件下的活体判断。其成像特点（如屏幕无法成像，不同材质反射率不同等）可以实现高鲁棒性的活体判断。

1.2.6 离线Depth深度图像活体检测（3D结构光）

通过3D建模判断目标对象是否为活体，基于3D结构光成像原理，可强效防御图片、视频、屏幕、模具等攻击。

1.2.7 离线1：1对比

提供本地化的1：1人脸对比功能，高鲁棒性算法，可对应各种姿态、肤色、光照等场景，可有效应用于人证比对、身份核验等场景。

示例工程中包含：

- 图片与图片的比对：两张人脸图片的1：1对比，并返回相似度分值。
- 图片与视频流比对：一张预设的人脸图片，和摄像头实时采集的符合条件的人脸图片进行对比。

1.2.8 离线1：N搜索

本地数据库中保留所有人脸特征值（如需要保留原图，可根据业务需要自行修改工程）。

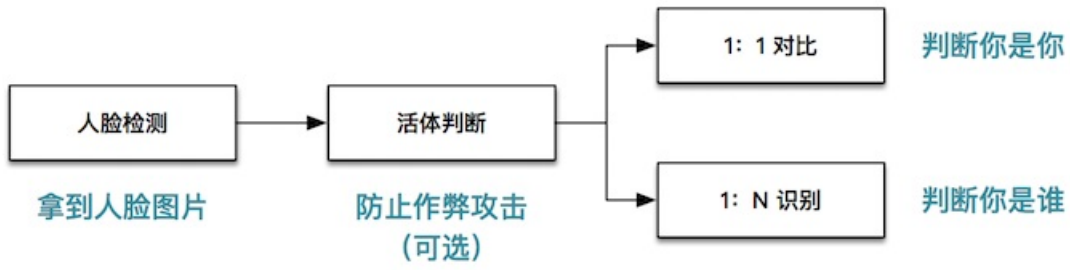
- 视频流采集的人脸在人脸库中搜索：视频流中实时采集人脸，并与人脸库中预设的人脸库进行一一对比，返回相似度最高的user及对应分值。

1.2.9 离线人脸库管理

人脸数量不做上限，可根据业务需要适当调整，支持人脸库、人脸组、用户、Face几个维度的增删改查设置。

1.3 业务流程

1.3.1 通用流程概述



如上图所示，人脸识别的核心业务流程可以分为三个步骤。

1. 检测采集：通过视频流实时检测跟踪，并采集到符合质量要求的人脸图片，用于后续的识别。
2. 活体判断：为可选步骤，主要保障业务操作者为真人，避免业务作弊。加上这步的校验，即只有满足活体判断通过，人脸图片才会被采集。
3. 对比识别：1：1对比主要是判断「你是你」，用于核实身份的真实性；1：N搜索主要判断「你是谁」，用于明确身份的具体所属。

1.3.2 活体检测

> 提示：此版本仅支持RGB可见光活体、NIR近红外活体。Depth深度活体将会尽快推出

1.3.2.1 基础原理

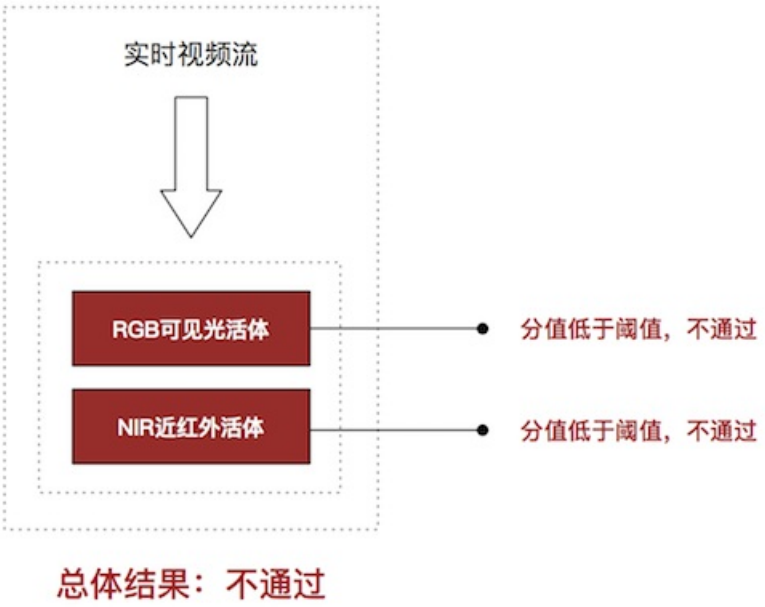
- RGB可见光活体：主要基于图片破绽，判断目标对象是否为活体。例如图像中的屏幕反光、成像畸形等，最主要的应用情形为屏幕的二次翻拍等攻击防御。此种活体对于待检测图片的要求，主要需要满足画面中除了人脸以外，要尽可能保留一些背景内容，用于查找破绽，通常建议人脸与屏幕的长宽比为1：3。为控制达到此比例，建议通过调整最小检测人脸参数，控制采集的人脸不可过大，避免人脸面积占比过高，而导致图片中没有多少背景信息。同时RGB活体受光线影响较大，所以在强光、暗光等场景，容易数值波动较大，主要影响的是「通过率」，产品策略上需要通过适当的补光、使用宽动态镜头抵消逆光等方式缓解。
- NIR近红外活体：主要基于近红外光线反射成像原理，通过人脸呈现来判断是否为活体。即使是夜间或者没有自然光的情况下，依然可以判断活体。因为其成像特点，对于屏幕、图片等攻击形式，基本可以达到近似于100%的活体防御。同时近红外设备的成本，相对性价比更高。
- Depth深度活体（3D结构光）：结构光原理是通过主动光发射，在物体上形成光栅，并接受此信息进行活体分析。同样可以不需要自然光。3D结构光的成像更为稳定，抗攻击能力更强，对图像噪声的抗干扰能力也更强，是相对更加安全和稳定的方案，但相应的硬件设备造价也较高，需要根据实际业务成本预算，进行综合考虑。

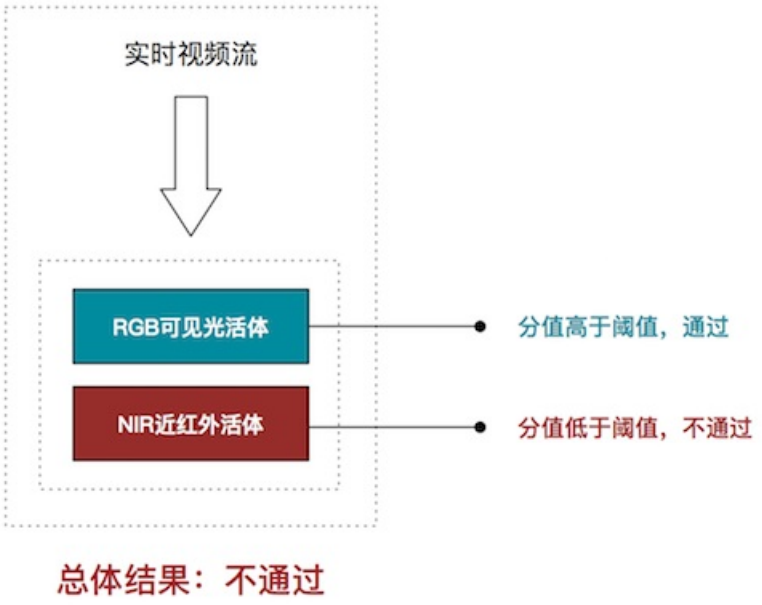
以上，无论活体是否可以通过，我们业务的最终目的，还是要进行「对比识别」，所以**拿到合适的RGB图像还是最为关键**，保障符合条件的RGB图像获取也是需要在活体判断同时，最为需要关注的事情。

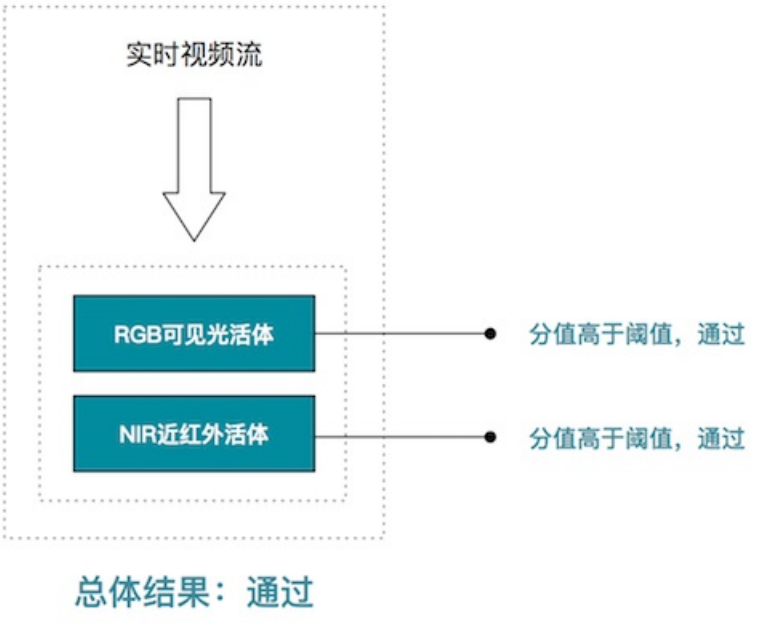
在实际业务场景中，需要根据场景特点，灵活组合使用以上几种活体方案。当然，实际业务中，**没有绝对100%的安全**，在应用过程中，还需要根据业务流程特点，指定一系列的辅助措施，如证件信息读取、密码、其他生物特征识别等，达到刚安全的核验。

1.3.2.2 产品应用策略

活体判断逻辑简单理解为：满足活体条件才可以采集图片，否则一直反复判断，直到满足活体条件为止。因为我们最终送去识别的图片是RGB图片，所以需要**保证活体通过时，在同一时间采集RGB图片，才可真正防止作弊攻击**。当多种活体叠加使用时，需要满足所有活体都通过，才能出发此操作，如果有任一活体没有通过，都不可进入识别步骤。







1.3.2.3 场景及应用方案

- 通行场景：此场景通常保障通行速度为主，确保不影响通行秩序和效率。所以建议无需使用三重活体检测，可仅用NIR活体或Depth活体，保障效率同时仍可保证安全性。
- 身份核验场景：此场景通常保障业务安全性为主，可尽可能提供更加安全的活体方案。如RGB+NIR，或者RGB+Depth，乃至RGB+NIR+Depth，最大程度确保对攻击的拒绝率较高。
- 强光/暗光场景：光线较强的场景，RGB活体会受影响比较严重，建议使用NIR或者Depth活体，同时尽量通过产品策略避免这两种情况的光线，例如添加补光灯、配备遮光板等。

1.3.2.4 应用方案及模组选择

- 无需活体：如有人值守的场景下，活体检测并无太大的必要（活体检测也会增加额外的业务耗时），现有设备的单目USB摄像头即可。
- 仅用RGB可见光活体：如果您的设备已经配备了USB单目摄像头，则无需更换，输出的RGB图像可直接用于RGB可见光活体算法识别。
- RGB可见光+NIR近红外活体：需要配备能够同时获取RGB、NIR近红外数据的镜头使用。
- RGB可见光+Depth深度活体：需要配备能够同时获取RGB、Depth深度图像数据的镜头使用。
- RGB可见光+NIR近红外+Depth深度活体：此种方式安全度最高，目前SDK正在模组适配中，**后续会陆续推出已适配的模组方案，敬请期待。**

1.3.2.4 指标

活体检测存在几个标准的指标，如下所示：

- 拒绝率（TRR）：如99%，代表100次作弊假体攻击，会有99次被拒绝。
- 误拒率（FRR）：如0.5%，指1000次真人请求，会有5次因为活体分数低于阈值被错误拒绝。
- 通过率（TAR）：如99%，指100次真人请求，会有99次因为活体分数高于阈值而通过。
- 阈值（Threshold）：高于此数值，则可判断为活体。

温馨提示：此SDK涉及多种离线活体检测模型，加上镜头模组效果各异，使用环境也较为复杂，以下给出综合测试值，仅供参考：

- 拒绝率：> 99.5%
- 误拒率：< 1%
- 通过率：> 99%

1.3.2.4 阈值选择推荐

活体分值区间为[0~1]，大部分情况的活体攻击，活体分值近似于接近0.0，建议阈值如下，高于此阈值的即可判断为活体。

- RGB可见光活体：0.8
- NIR近红外活体：0.8
- Depth深度活体（3D结构光）：0.8（后续版本即将支持）

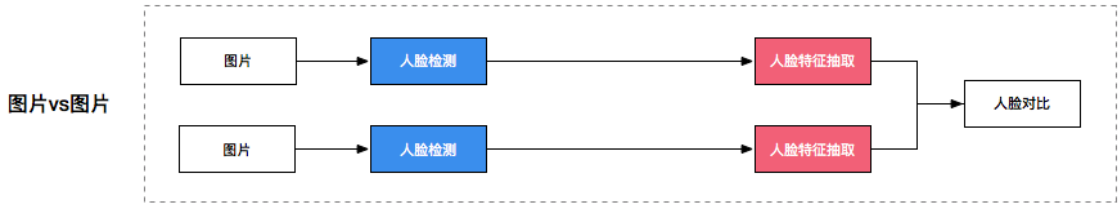
1.3.3 人脸1：1对比

人脸对比分为两种常见形态，具体如下：

1.3.3.1 对比类型：图片vs图片

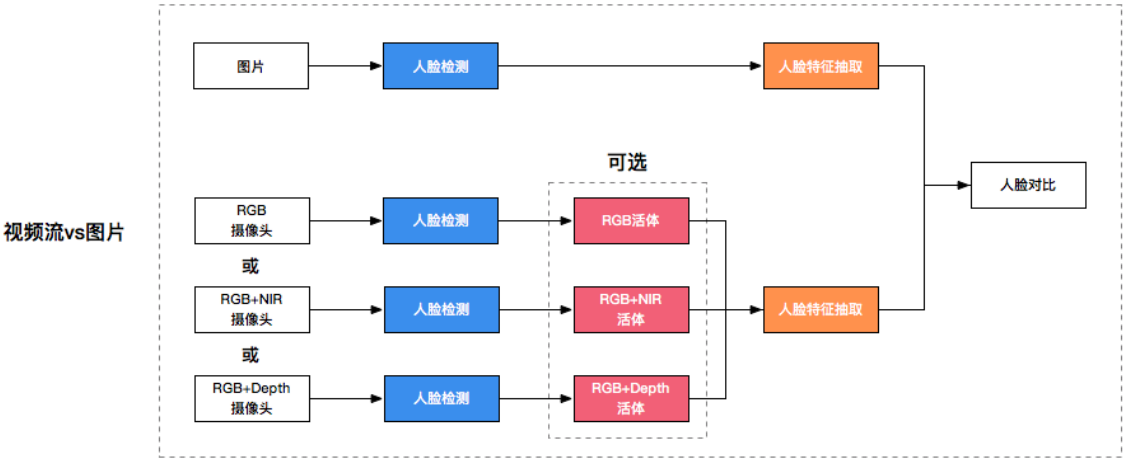
如果有两张待对比的图片（如证件图片、以事先获取到的人脸生活照等），则可以直接通过离线SDK进行对比，业务流程如下

图所示：



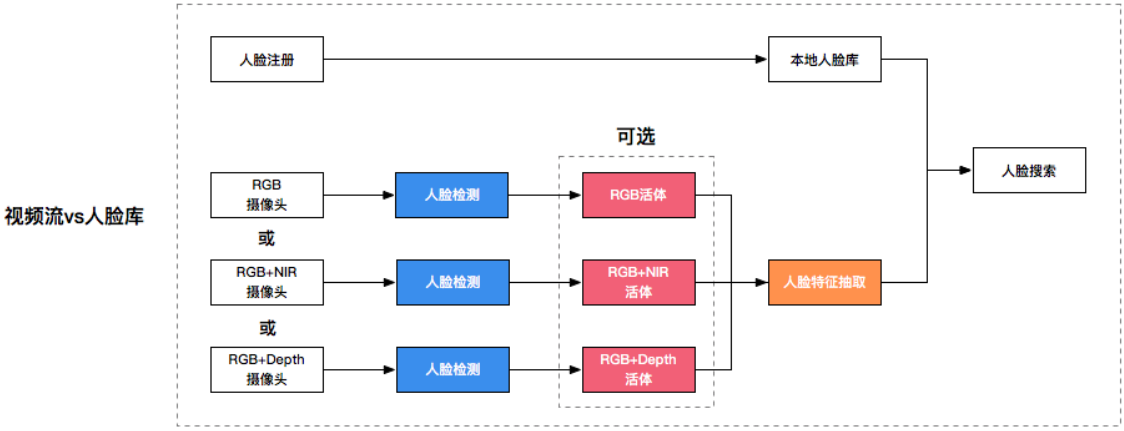
1.3.3.2 对比类型：实时视频流vs图片

这是最为常见的1：1对比类型，一张事先获取的图片（通常为身份证芯片照、证件照片等），与现场人脸采集的图片进行对比。如果需要为无人看守，则需要配备活体检测保障业务真实性和安全性。



1.3.4 人脸1：N搜索

将需要识别的人脸图片集注册到本地人脸库中，当有用户需要识别身份时，从视频流中实时采集人脸图片，与人脸库中的人脸集合对比，得到搜索结果。如果需要为无人看守，则需要配备活体检测保障业务真实性和安全性。



1.4 设备选型

1.4.1 镜头模组选择

1.4.1.1 无活体或仅用RGB单目活体

uvc免驱单目USB摄像头，推荐**视派尔**单目摄像头，默认为3mm焦距镜头，可以选择6mm镜头，可以检测识别更远的人脸。

- **视派尔C-EP28WD400**，[产品规格书]

1.4.1.2 RGB可见光+NIR近红外活体

推荐使用如下品牌的近红外摄像头，可根据场景举例选择合适焦距摄像头。

- **视派尔C-EP35WDLDIR**，[产品规格书] | [去AI市场购买]
- **迪威泰DV-BD4044S305AD**，[产品规格书]；联系人：杨先生；联系电话：18643209187
- **迪威泰DV-BD4053S305AD**，[产品规格书]；联系人：杨先生；联系电话：18643209187

NIR近红外实际检测效果回显示例：

1.4.1.3 RGB可见光+Depth深度活体（3D结构光）

推荐使用如下品牌的深度摄像头，可根据场景举例选择合适焦距摄像头。

- **奥比中光Astra Mini / Mini S**，[产品规格书]；[AI市场购买链接]

1.4.1.4 是否可以使用其他品牌镜头？

如果您需要使用NIR近红外或者Depth活体，则一定要配备支持对应图像格式类型的镜头模组。目前百度离线SDK适配的镜头类型有限，难以覆盖市面上所有双目模组。您可以查看下方的图像适配指标文档，只需满足文档中的图像/数据要求，即可自行进行模组适配。

[镜头模组图像适配指标]

我们也诚挚欢迎各镜头模组厂商参与到SDK的合作中来，我们会积极与您进行产品、技术、商务等多方面对接。测评效果较佳的模组，将会作为SDK官方适配模组，并在所有技术资料中优先推荐用户使用。您可以将合作申请发送到下方咨询邮箱，我们接到您的申请后会尽快与您取得联系，感谢您的信任。

合作邮件：ai#baidu.com（ #替换成@符号 ）

1.4.2 机器选择

- 系统：Windows 7、Windows 10系统的设备
- 处理器：推荐Intel i3及以上
- 内存：4.00 GB及以上
- 系统类型：32位、64位
- 具备2个及以上的USB插口
- 安装vs2015环境（不保证其他vs版本的兼容性）

1.5 方案选型

离线SDK具备离线人脸检测、跟踪、质量校验、图像采集、RGB/NIR/3D结构光活体、离线对比识别、离线人脸库管理。可基本cover人脸识别所需要的全部能力，仅需一个SDK即可完成业务能力覆盖。但相对来说仍然具备一定的场景及业务使用限制。

目前AI开放平台也同时开放采集SDK和API能力，建议您根据实际的业务需要，选择合适的产品应用方案，以达到最佳的应用效果。

1.5.1 使用离线SDK

以下特点及场景，适合仅适用人脸离线SDK

适用的场景特点

- 网络条件不稳定
- 无网络
- 数据安全性高，不允许联网
- 人脸库较小，且变更不频繁，如1w人以下
- 单台设备、单人脸检测与识别

> 注意：此版本离线SDK暂不支持多线程处理，主要适用于单人脸核验场景，不建议用于多人脸抓拍识别。

典型场景及设备类型

- 人脸门禁
- 人脸闸机
- 人证核验机
- 自助柜机
- 人脸考勤机
- 会场签到

> 提示：对于离线1：N场景，需要您基于SDK构造上层的人脸库数据更新业务逻辑，例如小区门口的闸机为纯离线，但是住户的变更信息，如何同步到每一个大门的闸机中。

1.5.2 使用采集SDK+API接口

以下特点及场景，适合使用采集SDK+API接口

适用的场景特点

- 网络条件良好
- 人脸库量级庞大
- 需要跨地域同步人脸库
- 人脸库更新频繁
- 用户APP产品形态

> 提示：采集SDK具备本地化的人脸检测跟踪、质量校验、人脸采集等功能，与离线SDK相比，活体和识别操作使用云端服务完成。

典型场景及设备类型

- 零售会员识别
- 人脸支付
- 手机APP
- 移动考勤
- 远程开户

1.5.3 使用离线SDK+API接口

以下特点及场景，适合使用离线SDK+API接口

适用的场景特点

- 网络条件较好
- 人脸库较小，但变更频繁
- 业务常为集中时段的高并发（如考勤）

典型场景及设备类型

- 上下班刷脸考勤
- 社区闸机
- 楼宇门禁

离线人脸库处理固定人员的识别请求，将在本地抵消掉很多API请求量；同时API处理变动人员的请求，方便及时同步更新，从而总体节省QPS资源的成本消耗。

2、SDK详细介绍

2.1 名词解释

名词	定义
SDK	Windows离线人脸识别sdk
vs2015	微软的开发工具visual studio 2015 (推荐安装vs community 2015)
License	人脸识别激活所需要的激活文件,可利用激活工具生成
key	人脸激活所需的序列号，可从百度官网申请及购买（ai.baidu.com）
feature	人脸特征值，用来表示人脸特征的512个浮点值
landmark	人脸关键点坐标
face_token	对应人脸图片的唯一编码，若一个人上传了2张不同图片，则可能有2个不同的face_token,它和图片一一对应

2.2 SDK简介

本sdk适应于windows平台下的人脸识别系统，为在c++版本基础上进行c#接口支持的sdk，开发者可在vs2015下面进行开发（推荐使用，不保证vs其他版本都兼容）

SDK采用C#调用C++的动态库dll的方式，另外随SDK附带一个鉴权激活工具（LicenseTool.exe），此激活工具为采用微软MFC

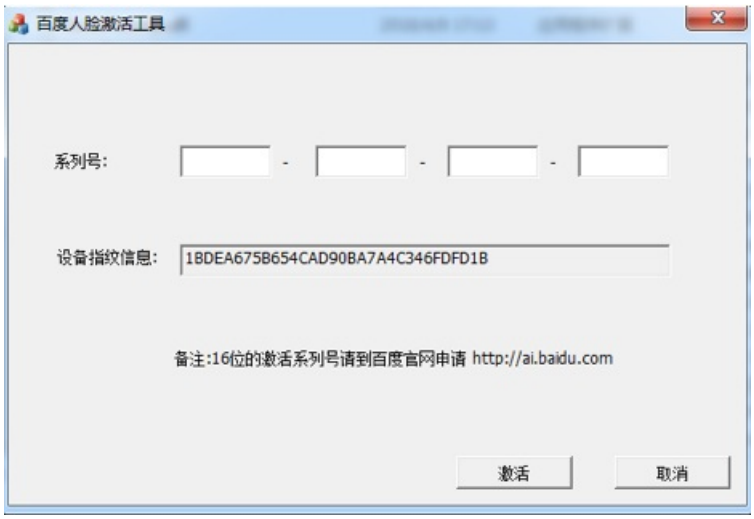
开发，通过该激活工具可生成正常接入SDK的激活license文件（生成两个文件license.ini和license.key）达到通过鉴权，正常使用sdk的目的。

2.3 SDK文件结构

SDK提供C++的动态库BaiduFaceApi.dll，C#调用方法如TestFace工程各demo示例。另外有附带的鉴权激活工具LicenseTool.exe。在此之外，还附带支撑SDK使用的人脸识别模型文件等在文件夹face-resource中，该文件夹在TestFace例子工程里面（该文件夹存放路径请参考TestFace工程，默认存放路径为您要开发的exe(如demo工程TestFace.exe)所在路径的上级目录下。若存放不对，可能会影响sdk正常使用,另外请勿删除该文件夹）。为运行exe，需要用到一些底层的库文件(dll)，相关库文件在TestFace的bin\Release文件夹中已提供。

2.4 激活工具（LicenseTool.exe）

在百度官网申请或购买序列号后，在TestFace的bin\Release目录夹下，双击运行LicenseTool.exe激活工具，可见如下所示界面。在输入框中输入16位的系列号后，点击激活按钮，稍等片刻，即可见弹出的激活成功对话框，若激活失败，请参考对话框中弹出的message信息查找原因。激活成功后会在激活目录生成license.ini以及license.key文件，这2个文件是作为SDK通过鉴权使用的配置文件，请勿删除。



如果你不方便使用网络，可以通过离线激活进行操作：

通过如上的激活工具，获取到设备指纹信息，通过序列号管理后台找到需要绑定的序列号，选择「离线激活」，填入指纹设备信息，即可下载获取到license.ini文件和license.key文件，把这2个文件拷贝到TestFaceApi的Win32或x64目录下，运行TestFaceApi.exe亦可通过鉴权。

2.5 Demo示例工程

Demo示例工程TestFace展示了如何集成百度人脸识别离线sdk。即通过DllImport c++的库文件BaiduFaceApi.dll，引入sdk的方法。

在TestFace中的Face.cs的Main()方法中，有使用SDK的各个接口方法示例。接入SDK及其简单，分2步2行代码。如下：

```
 sdk_init(false);
// 第一步：实例化人脸SDK

test_get_device_id();
// 第二步：调用sdk的功能，本句示例：获取设备指纹的示例代码
```

如上，即为调用sdk功能的最简单2步2行代码，当然调用sdk在程序退出后，需要释放sdk防止内存泄漏，需要删除sdk实例化的c++指针sdk_destroy();

示例工程中：分别有以下几个示例cs文件对应几个常用sdk的调用demo。

文件名	文件说明
FaceSetting.cs	人脸检测、识别等参数设置
FaceCompare.cs	人脸1:1 1:N 比对，人脸特征值提取，特征值比较等
FaceLiveness.cs	可见光RGB、红外IR活体检测、RGB&IR双目摄像头活体检测，RGB&DEPTH双目摄像头活体检测等
FaceTrack.cs	USB摄像头视频信息人脸实时检测、最大人脸检测等
FaceManager.cs	人脸库管理类，包括人脸注册、删除，更新、组管理，人脸信息查询等
FaceAttr.cs	人脸属性(年龄、性别、种族等)
FaceQuality.cs	人脸质量

2.6 特征抽取模型选择

模型介绍

v1.1版本起，SDK提供生活照和证件照两种特征抽取模型，主要适用场景如下:

- **生活照模型**：如手机拍摄的图片、较为清晰的证件照片、USB镜头实时采集的图片、网络摄像头实时采集的图片等。
- **证件照模型**：如身份证芯片照、各类证件照（工卡、学生卡、会员卡照片等）、人脸的像素普遍小于80px的图片等。

使用方法

在SDK初始化方法中控制：sdk_init(true); //**传入true为使用证件照模型，传入fasle为普通生活照模型**

注意事项

温馨提示：一经选择一个模型，则所有业务流程的特征抽取处理，都会使用此模型，两个模型不可同时作用。**如业务中设计证件照的特征抽取，请务必选择使用证件照模型**。

3、功能接口

SDK实现的主要功能有人脸实时跟踪检测、人脸特征值提取、RGB\IR活体检测、人脸注册、人脸更新、组管理、用户管理以及1:1人脸对比,1:N人脸识别、特征值的比对和通过USB或笔记本自带摄像头检测视频帧，返回识别出的人脸信息等，另外支持对人脸检测进行设置，达到根据设置进行识别的目的。各接口功能及传入参数和返回结果（返回结果一般为json格式的字符串）描述如下：

3.1 人脸检测及设置

3.1.1 人脸检测track接口(传入图片)

方法名

`track`

方法说明

人脸检测，返回人脸信息

请求信息

参数	必须	类型	描述
image	是	string	人脸图片信息，传入图片文件路径
max_track_num	是	int	最多检测人脸数量，默认为1，最大不超过5

返回信息

返回结果为json，示例如下:

```
 {
 ”count”: 1,
 data: {
  “width”: 200,
  ”angle”: 5,
  ”center_x”: 20.3,
  ”center_y”: 30.2,
  ”face_id”: 0,
  ”score”: 0.80,
  ”headpos”: [5, 3, 2],
  ”landmarks”: […]
 }
},

依次为

count: 检测的人脸数量
width: 人脸宽度， angle： 角度
center_x, center_y： 人脸中心位置坐标
face_id: 人脸id
score：人脸得分
headpos: 人脸三个角度（ x， y， z方向） landmark： 人脸轮廓关键点坐标
```

3.1.2 人脸检测track接口(传入二进制图片buffer)

方法名

`track_by_buf`

方法说明

人脸检测，返回人脸信息

请求信息

参数	必须	类型	描述
image	是	Unsigned char *	人脸图片信息，二进制图片buffer内容
size	是	int	二进制图片的大小
max_track_num	是	int	最多检测人脸数量，默认为1，最大不超过5

返回信息

返回结果为json，示例如下:

```
 {
 ”count”: 1,
 data: {
  “width”: 200,
  ”angle”: 5,
  ”center_x”: 20.3,
  ”center_y”: 30.2,
  ”face_id”: 0,
  ”score”: 0.80,
  ”headpos”: [5, 3, 2],
  ”landmarks”: […]
 }
},
```

依次为

count: 检测的人脸数量
width: 人脸宽度， angle： 角度
center_x, center_y： 人脸中心位置坐标
face_id: 人脸id
score：人脸得分
headpos: 人脸三个角度（ x， y， z方向） landmark： 人脸轮廓关键点坐标

3.1.3 人脸检测track_max_face接口

方法名

`track_max_face`

方法说明

检测最大人脸。

请求信息

参数	必须	类型	描述
image	是	string	图片信息（数据大小应小于10M），文件指定路径

返回信息

返回结果为json，示例如下:

```
 {
 ”count”: 1,
 data: {
  “width”: 200,
  ”angle”: 5,
  ”center_x”: 20.3,
  ”center_y”: 30.2,
  ”face_id”: 0,
  ”score”: 0.80,
  ”headpos”: [5, 3, 2],
  ”landmarks”: […]
 }
},
```

依次为

count: 检测的人脸数量
width: 人脸宽度， angle： 角度
center_x, center_y： 人脸中心位置坐标
face_id: 人脸id
score：人脸得分
headpos: 人脸三个角度（ x， y， z方向） landmark： 人脸轮廓关键点坐标

3.1.4 人脸检测track_max_face接口(传入二进制图片buffer)

方法名

`track_max_face_by_buf`

方法说明

检测最大人脸。

请求信息

参数	必须	类型	描述
image	是	Unsigned char *	人脸图片信息，二进制图片buffer内容
size	是	int	二进制图片的大小

返回信息

返回结果为json，示例如下:

```
 {
 ”count”: 1,
 data: {
  “width”: 200,
  ”angle”: 5,
  ”center_x”: 20.3,
  ”center_y”: 30.2,
  ”face_id”: 0,
  ”score”: 0.80,
  ”headpos”: [5, 3, 2],
  ”landmarks”: […]
 }
},

依次为

count: 检测的人脸数量
width: 人脸宽度， angle： 角度
center_x, center_y： 人脸中心位置坐标
face_id: 人脸id
score：人脸得分
headpos: 人脸三个角度（ x， y， z方向） landmark： 人脸轮廓关键点坐标
```

3.1.5 人脸检测设置接口

请参考`TestFace`工程中的代码示例及`FaceSetting.cs`的代码注释。

3.1.6 USB摄像头检测

方法名

`usb_track_face_info`

方法说明

人脸检测，返回人脸信息（sdk内部使用opencv打开摄像头检测）

请求信息

参数	必须	类型	描述
FaceCallbackt	是	FaceCallback	C#的回调方法。FaceCallback(byte[] buf, int size, string res);回调中返回3个参数，1、buf表示回调的实时视频帧图片二进制buffer；2、size为视频帧大小；3、res为检测到的人脸返回信息，同track方法中的返回json

3.2 人脸管理接口

3.2.1 人脸注册接口

方法名

`user_add`

方法说明

用户注册，该接口支持传入本地图片文件地址。

请求信息

参数	必须	类型	示例描述
user_id	是	string	用户id，字母、数字、下划线组成，最多128个字符
group_id	是	string	用户组id，标识一组用户（由数字、字母、下划线组成），长度限制128B。用户组和user_id之间，仅为映射关系。如传入的groupid并未事先创建完毕，则注册用户的同时，直接完成group的创建
image	是	string	图片信息，须小于10M，传入图片的本地文件地址
user_info	否	string	用户资料，256个字符以内

返回信息

errno信息：

- errno=0：Success
- errno>0：错误码对应的详细msg

data信息:

字段	类型	说明
log_id	string	请求日志标识
face_token	string	注册图片对应的face_token

3.2.2 人脸注册接口（传入二进制图片buffer）

方法名

`user_add_by_buf`

方法说明

用户注册，该接口支持传入二进制的图片buffer。

请求信息

参数	必须	类型	示例描述
user_id	是	string	用户id，字母、数字、下划线组成，最多128个字符
group_id	是	string	用户组id，标识一组用户（由数字、字母、下划线组成），长度限制128B。用户组和user_id之间，仅为映射关系。如传入的groupid并未事先创建完毕，则注册用户的同时，直接完成group的创建
image	是	Unsigned char*	二进制图片内容、须小于10M
size	是	int	二进制图片大小
user_info	否	string	用户资料，256个字符以内

返回信息

errno信息：

- errno=0：Success
- errno>0：错误码对应的详细msg

data信息:

字段	类型	说明
log_id	string	请求日志标识
face_token	string	注册图片对应的face_token

3.2.3 人脸更新接口

方法名

`user_update`

方法说明

人脸图片及信息更新

请求信息

参数	必须	类型	示例描述
user_id	是	string	用户id（由数字、字母、下划线组成），长度限制128B
group_id	是	string	用户组id，标识一组用户（由数字、字母、下划线组成），长度限制128B
image	是	string	图片信息，数据大小应小于10M，传入本地图片文件地址，每次仅支持单张图片
user_info	否	string	用户资料，长度限制256个字符以内

返回信息

errno信息：

- errno=0：Success
- errno>0：错误码对应的详细msg

data信息:

字段	类型	说明
log_id	string	请求日志标识
face_token	string	注册图片对应的face_token

3.2.4 人脸更新接口（传入二进制图片buffer）

方法名

`user_update_by_buf`

方法说明

人脸更新，该接口支持传入二进制的图片buffer。

请求信息

参数	必须	类型	示例描述
user_id	是	string	用户id（由数字、字母、下划线组成），长度限制128B
group_id	是	string	用户组id，标识一组用户（由数字、字母、下划线组成），长度限制128B
image	是	Unsigned char *	二进制图片内容、须小于10M
size	是	int	二进制图片大小
user_info	否	string	用户资料，256个字符以内

返回信息

errno信息：

- errno=0：Success
- errno>0：错误码对应的详细msg

data信息:

字段	类型	说明
log_id	string	请求日志标识
face_token	string	注册图片对应的face_token

3.2.5 人脸删除接口

方法名

`user_face_delete`

方法说明

人脸删除

请求信息

参数	必须	类型	示例描述
user_id	是	string	用户id（由数字、字母、下划线组成），长度限制128B
group_id	是	string	用户组id，标识一组用户（由数字、字母、下划线组成），长度限制128B
face_token	是	string	人脸id（由数字、字母、下划线组成）长度限制128B

返回信息

errno信息：

- errno=0：Success
- errno>0：错误码对应的详细msg

data信息:

字段	类型	说明
log_id	string	请求日志标识

3.2.6 用户删除接口

方法名

`user_delete`

方法说明

用户删除

请求信息

参数	必须	类型	示例描述
user_id	是	string	用户id（由数字、字母、下划线组成），长度限制128B
group_id	是	string	用户组id，标识一组用户（由数字、字母、下划线组成），长度限制128B

返回信息

errno信息：

- errno=0：Success
- errno>0：错误码对应的详细msg

data信息:

字段	类型	说明
log_id	string	请求日志标识

3.2.7 创建用户组接口

方法名

`group_add`

方法说明

创建用户组

请求信息

参数	必须	类型	示例描述
group_id	是	string	用户组id，标识一组用户（由数字、字母、下划线组成），长度限制128B

返回信息

errno信息：

- errno=0：Success
- errno>0：错误码对应的详细msg

data信息:

字段	类型	说明
log_id	string	请求日志标识

3.2.8 用户组删除

方法名

`group_delete`

方法说明

用户组删除

请求信息

参数	必须	类型	示例描述
group_id	是	string	用户组id，标识一组用户（由数字、字母、下划线组成），长度限制128B

返回信息

errno信息：

- errno=0：Success
- errno>0：错误码对应的详细msg

data信息:

字段	类型	说明
log_id	string	请求日志标识

3.2.9 用户信息查询接口

方法名

`get_user_info`

方法说明

用户信息查询

请求信息

参数	必须	类型	示例描述
user_id	是	string	用户id（由数字、字母、下划线组成），长度限制128B
group_id	是	string	用户组id，标识一组用户（由数字、字母、下划线组成），长度限制128B

返回信息

errno信息：

- errno=0：Success
- errno>0：错误码对应的详细msg

data信息:

字段	类型	说明
log_id	string	请求日志标识
result	array	识别结果列表
group_id	string	组id
face_token	string	人脸特征的唯一标识
user_info	string	用户信息
create_time	string	人脸首次注册时间

3.2.10 用户组列表查询接口

方法名

`get_user_list`

方法说明

用户组列表查询

请求信息

参数	必须	类型	示例描述
group_id	是	string	用户组id
start	否	int	默认值为0
length	否	int	默认值100，最大值1000

返回信息

errno信息：

- errno=0：Success
- errno>0：错误码对应的详细msg

data信息:

字段	类型	说明
log_id	string	请求日志表识
user_id_list	array	user_id列表数组

3.2.11 组列表查询接口

方法名

`get_group_list`

方法说明

组列表查询

请求信息

参数	必须	类型	示例描述
start	否	int	默认值为0，从0开始
length	否	int	默认值为100，最大为1000

返回信息

errno信息：

- errno=0：Success
- errno>0：错误码对应的详细msg

data信息:

字段	类型	说明
log_id	string	请求日志标识
group_id_list	array	group_id列表数组

3.3 人脸对比及识别接口

3.3.1 人脸对比接口

方法名

`match`

方法说明

人脸对比接口（传入图片文件路径）

请求信息

参数	必须	类型	示例描述
image1	是	string	需要对比的第一张图片，小于10M，传入图片文件路径
image2	是	string	需要对比的第二张图片，小于10M，传入图片文件路径

返回信息

errno信息：

- errno=0：Success
- errno>0：错误码对应的详细msg

data信息:

字段	类型	说明
log_id	string	请求日志标识
result	array	识别结果列表
score	string	相似度分值，0-100，百分制，保留后2位小数点

3.3.2 人脸对比接口（传入二进制图片buffer）

方法名

`match_by_buf`

方法说明

人脸对比接口（传入二进制图片buffer）

请求信息

参数	必须	类型	示例描述
image1	是	Unsigned char *	需要对比的第一张图片，小于10M
size	是	int	图片1的大小
image2	是	Unsigned char *	需要对比的第二张图片，小于10M
size	是	int	图片2的大小

返回信息

errno信息：

- errno=0：Success
- errno>0：错误码对应的详细msg

data信息:

字段	类型	说明
log_id	string	请求日志标识
result	array	识别结果列表
score	string	相似度分值，0-100，百分制，保留后2位小数点

3.3.3 人脸对比接口（传入二进制图片buffer和特征值feature比对）

方法名

`match_by_feature`

方法说明

人脸对比接口(传入二进制图片buffer)

请求信息

参数	必须	类型	示例描述
feature1	是	Byte[]	需要对比的特征值
Fea_len1	是	int	特征值长度，一般为2048个byte
image2	是	Byte[]	需要对比的第二张图片，小于10M
size	是	int	图片2的大小

返回信息

errno信息：

- errno=0：Success
- errno>0：错误码对应的详细msg

data信息:

字段	类型	说明
log_id	string	请求日志标识
result	array	识别结果列表
score	string	相似度分值，0-100，百分制，保留后2位小数点

3.3.4 人脸识别identify接口

方法名

`identify`

方法说明

人脸识别，提供1：N查找 (传入图片文件路径)

请求信息

参数	必须	类型	示例描述
image	是	string	图片信息，数据大小小于10M，传入图片文件路径
group_id_list	是	string	组id列表。默认至少填写一个group_id，从指定的group中进行查找。需要同时查询多个group，用逗号分隔，上限10个
user_id	是	string	用户id，若指定了某个user，则只会与指定group下的这个user进行对比；若user_id传空字符串""，则会与此group下的所有user进行1：N识别
user_top_num	否	int	识别后返回的用户top数，默认为1，最多返回50个结果

返回信息

errno信息：

- errno=0：Success
- errno>0：错误码对应的详细msg

data信息:

字段	类型	说明
log_id	string	请求日志标识
face_token	string	传入识别的image的face_token值
result_num	int	返回结果数量
result	array	识别结果列表
group_id	string	用户组id
user_id	string	用户id
score	string	相似度分值,0-100,百分制，保留小数点后两位

3.3.5 人脸识别identify接口（传入二进制图片buffer）

方法名

`identify_by_buf`

方法说明

人脸识别，提供1：N查找 (传入二进制图片buffer)

请求信息

参数	必须	类型	示例描述
image	是	Unsigned char *	二进制图片信息，数据大小小于10M
size	是	int	图片大小
group_id_list	是	string	组id列表。默认至少填写一个group_id，从指定的group中进行查找。需要同时查询多个group，用逗号分隔，上限10个
user_id	是	string	用户id，若指定了某个user，则只会与指定group下的这个user进行对比；若user_id传空字符串""，则会与此group下的所有user进行1：N识别
user_top_num	否	int	识别后返回的用户top数，默认为1，最多返回50个结果

返回信息

errno信息：

- errno=0：Success
- errno>0：错误码对应的详细msg

data信息:

字段	类型	说明
log_id	string	请求日志标识
face_token	string	传入识别的image的face_token值
result_num	int	返回结果数量
result	array	识别结果列表
group_id	string	用户组id
user_id	string	用户id
score	string	相似度分值,0-100,百分制，保留小数点后两位

3.3.6 特征值提取接口（通过传入图片）

方法名

`get_face_feature`

方法说明

提取人脸特征值，为512个浮点值，已加密

请求信息

参数	必须	类型	示例描述
image	是	string	图片信息，数据大小小于10M，传入图片文件路径
length	是	int	通过引用返回特征值的长度，若为2048表示提取正确，其他值表示提取了错误的特征值

返回信息

byte[]， 若为空，表示提取特征值失败，不为空，则为长度为2048的byte[]。

3.3.7 特征值提取接口（通过传入二进制图片buffer）

方法名

`get_face_feature_by_buf`

方法说明

提取人脸特征值，为2048个byte (传入二进制图片buffer)

请求信息

参数	必须	类型	示例描述
image	是	Unsigned char *	二进制图片信息，数据大小小于10M
size	是	int	二进制图片大小
length	是	int	通过引用返回特征值的长度，若为2048表示提取正确，其他值表示提取了错误的特征值

返回信息

byte[]， 若为空，表示提取特征值失败，不为空，则为长度为2048的byte[]。

3.3.8 特征值比较接口

方法名

`compare_feature`

方法说明

对人脸特征值进行比较，可返回人脸特征相似分值（百分制）

请求信息

参数	必须	类型	示例描述
feature1	是	byte[]	2048个byte数组的特征值
fea_len1	是	int	特征值1的长度
feature2	是	byte[]	2048个byte数组的特征值
fea_len2	是	int	特征值2的长度

返回信息

返回值	类型	说明
score	float	比较结果，百分制的分值

3.4 活体检测接口

3.4.1 可见光(RGB)活体检测接口（通过传入图片）

方法名

`rgb_liveness_check`

方法说明

人脸可见光活体检测接口（image_type 为1表示base64图片编码 为2表示文件路径）

请求信息

参数	必须	类型	示例描述
image	是	string	需要检测的图片，小于10M，图片的本地文件路径地址

返回信息

errno信息：

- errno=0：Success
- errno>0：错误码对应的详细msg

data信息:

字段	类型	说明
score	string	活体检测分值，范围为0-1，实际为浮点型，开发者可自行转换为百分制。

3.4.2 可见光(RGB)活体检测接口（通过传入二进制图片buffer）

方法名

`rgb_liveness_check_by_buf`

方法说明

人脸可见光活体检测接口（image_type 为1表示base64图片编码 为2表示文件路径）

请求信息

参数	必须	类型	示例描述
image	是	byte[]	需要检测的图片，二级制格式，小于10M
size	是	int	二进制图片大小

返回信息

errno信息：

- errno=0：Success
- errno>0：错误码对应的详细msg

data信息:

字段	类型	说明
score	string	活体检测分值，范围为0-1，实际为浮点型，开发者可自行转换为百分制。

3.4.3 可见光(RGB)&近红外(NIR)活体检测接口（通过传入二进制图片buffer）

方法名

`rgb_ir_liveness_check_by_buf`

方法说明

可见光和近红外（RGB & NIR）同时活体检测

请求信息

参数	必须	类型	示例描述
rgb_img	是	byte[]	可见光图片的二进制图片信息（rgb），须小于10M
rgb_size	是	int	图片大小
ir_img	是	byte[]	近红外图片的二进制图片信息（ir），须小于10M
ir_size	是	int	图片大小

返回信息

errno信息：

- errno=0：Success

- errno>0：错误码对应的详细msg

data信息:

字段	类型	说明
rgb_score	string	RGB活体检测分值，范围为0-1，实际为浮点型，可自行转换为百分制。
Ir_score	string	NIR活体检测分值，范围为0-1，实际为浮点型，可自行转换为百分制。

3.4.4 可见光(RGB)&近红外(NIR)活体检测接口（通过sdk底层打开opencv视频帧）

方法名

`rgb_ir_liveness_check_by_opencv`

方法说明

可见光和近红外（RGB & NIR）同时活体检测

请求信息

参数	必须	类型	示例描述
FaceCallback	是		FaceCallback(byte[] buf, int size, string res);回调中返回3个参数；1、buf表示回调的实时视频帧图片二进制buffer；2、size为视频帧大小；3、res为检测到的人脸返回信息，同track方法中的返回json

3.4.5 可见光(RGB)&深度(Depth)活体检测接口（通过传入二进制图片buffer）

方法名

`rgb_depth_liveness_check_by_buf`

方法说明

可见光和深度图（RGB & Depth）同时活体检测

请求信息

参数	必须	类型	示例描述
rgb_img	是	Unsigned char*	可见光图片的二进制图片信息（rgb），须小于10M
rgb_size	是	int	图片大小
depth_img	是	Unsigned char*	深度图片的二进制图片信息（ir），须小于10M
depth_size	是	int	图片大小

返回信息

errno信息：

- errno=0：Success
- errno>0：错误码对应的详细msghttp://news.family.baidu.com/specialTopicDetail?subjectId=186429&subjectPathNodeId=2103

data信息:

字段	类型	说明
Rgb_score	string	RGB活体检测分值，范围为0-1，实际为浮点型，可自行转换为百分制。
depth_score	string	NIR活体检测分值，范围为0-1，实际为浮点型，可自行转换为百分制。

3.4.6 可见光(RGB)&深度(Depth)活体检测接口（通过sdk底层打开opencv视频帧）

方法名

`rgb_ir_liveness_check`

方法说明

可见光和近红外（RGB & NIR）同时活体检测

请求信息

参数	必须	类型	示例描述
FaceCallback	是		FaceCallback(byte[] buf, int size, string res);回调中返回3个参数；1、buf表示回调的实时视频帧图片二进制buffer；2、size为视频帧大小；3、res为检测到的人脸返回信息，同track方法中的返回json

3.5 质量接口

3.5.1 人脸质量接口（通过传入图片）

方法名

`face_quality`

方法说明

人脸质量检测接口

请求信息

参数	必须	类型	示例描述
image	是	string	需要检测的人脸图片，小于10M, 传入图片文件的地址

返回信息

errno信息：

- errno=0：Success
- errno>0：错误码对应的详细msg

data信息:

字段	类型	说明
bluriness	float	模糊度
illum	float	光照
occl_l_eye	float	左眼遮挡度
occl_r_eye	float	右眼遮挡度
occl_nose	float	鼻子遮挡度
occl_mouth	float	嘴巴遮挡度
occl_l_contour	float	左脸遮挡度
occl_r_contour	float	右脸遮挡度
occl_chin	float	下巴遮挡度

3.5.2 人脸质量接口（通过传入图片二进制buffer）

方法名

`face_quality_by_buf`

方法说明

人脸质量检测接口

请求信息

参数	必须	类型	示例描述
image	是	byte[]	需要检测的图片二进制buffer
size	是	int	buffer大小

返回信息

errno信息：

- errno=0：Success
- errno>0：错误码对应的详细msg

data信息:

字段	类型	说明
bluriness	float	模糊度
illum	float	光照
occl_l_eye	float	左眼遮挡度
occl_r_eye	float	右眼遮挡度
occl_nose	float	鼻子遮挡度
occl_mouth	float	嘴巴遮挡度
occl_l_contour	float	左脸遮挡度
occl_r_contour	float	右脸遮挡度
occl_chin	float	下巴遮挡度

4、适配的深度双目摄像头特别说明

本SDK目前适配了奥比中光的双目深度摄像头。

在SDK中分别以动态库的形式接入，如orbe_camera.dll，是为了用户使用方便以及快速接入或不需要则可以移除。另外若要适配奥比中光的摄像头，在bin\Release下有个orbe_install目录，该目录有个sensorDriver.exe文件**需要双击安装后重启电脑，才能保证适配奥比中光摄像头成功**。

5、错误码及错误信息

错误码	错误内容	错误描述
0	SUCCESS	成功
1	SYSTEM ERROR	系统错误
2	UNKNOWN ERROR	未知错误
1001	NOT_AUTH	授权校验失败
1002	REQUEST PARAMS ERROR	请求参数错误
1003	DB_OP_FAILED	数据库操作失败
1004	NO_DATA	没有数据
1005	RECORD_UNEXIST	记录不存在
1006	RECORD_ALREADY_EXIST	记录已经存在
1007	FILE_NOT_EXIST	文件不存在
1008	GET_FEATURE_FAIL	提取特征值失败
1009	FILE_TOO_BIG	文件太大
1010	FACE_RESOURCE_NOT_EXIST	人脸资源文件不存在
1011	FEATURE_LEN_ERROR	特征值长度错误
1012	DETECT_NO_FACE	未检测到人脸
1013	CAMERA_ERROR	摄像头错误或不存在

6、常见问题

Q：工程运行过程中，提示face-resource不存在。

A：人脸sdk，需要一些模型文件，在demo工程的face-resource文件夹中，该文件夹需要放置在exe所在路径的上级目录下。若放置不正确，可能出现找不到模型文件，没法进行人脸识别。

Q：激活后是否可以把激活文件license.ini和license.key拷贝到其他设备运行？

A：不能，离线sdk和设备绑定，每个设备对应一个key和一个license文件，换设备无法运行。但对同一台设备，可把win32下的license.ini和license.key拷贝到x64下，则x64环境，该设备也等同于激活，可以使用。

Q：证件照等图片检测不到人脸，但实际是有人脸的？

A：默认可检测最小人脸大小是**100**，若检测不到，可通过设置最小人脸大小调整，例如调整最小检测人脸为30px，可使用api->set_min_face_size(30)，达到调整最小人脸检测大小的目的，然后再调用检测，这样能检测到比较小得图片如证件照等。v1.1版本中已经集成了证件照模型，推荐您升级SDK版本解决此问题。·

Linux-X86-SDK（历史版本）

版本日志

版本	日期	更新说明
v1.0.0	2018.09.21	Linux SDK初版，包括离线人脸采集、特征抽取、人脸比对、人脸属性等功能

目录

1、简介
1.1 产品概述
1.2 规格信息
1.3 兼容性
1.4 授权方式
1.5 产品定价
2、SDK详细介绍
2.1 名词解释
2.2 SDK简介
2.3 SDK文件结构
2.4 激活工具
2.5 Demo示例工程
2.6 特征抽取模型选择
3、功能接口
3.1 人脸检测及设置
3.1.1 人脸检测track接口(传入图片)
3.1.2 人脸检测track接口(传入二进制图片buffer)
3.1.3 人脸检测track_max_face接口
3.1.4 人脸检测track_max_face接口(传入二进制图片buffer)
3.1.5 人脸检测设置接口
3.1.6 USB摄像头检测
3.1.7 人脸检测track接口(传入opencv的mat)
3.2 人脸对比及识别接口
3.2.1 人脸对比接口
3.2.2 人脸对比接口（传入二进制图片buffer）
3.3 特征值提取接口
3.3.1 特征值提取接口（通过传入图片）
3.3.2 特征值提取接口（通过传入二进制图片buffer）
3.3.3 特征值提取接口（通过传入opencv的视频帧）
3.3.4 特征值比对接口
3.4 属性及质量接口
3.4.1 人脸属性（通过传入图片）
3.4.2 人脸属性（通过传入二进制图片buffer）
3.4.3 人脸质量接口（通过传入图片）
3.4.4 人脸质量接口（通过传入二进制图片buffer）
4、错误码及错误信息
5、常见问题

1、简介

1.1 产品概述

人脸离线识别SDK，包含人脸采集、活体检测、人脸对比/识别、人脸库管理等能力，并全部离线化、本地化。此SDK一经授权激活，可完全在无网环境下工作，所有数据皆在设备本地运行处理，可根据业务需要进行灵活的上层业务开发。核心能力分布如下图所示，后文会详细介绍。

适用场景特点

- **网络**：无网、局域网等情况，无法连接公网。如政府单位、金融保险、教育机构等。
- **安全**：行业特点所带来的人脸数据敏感性，即使可以连接公网也不可请求。
- **速度**：由于各地网络线路、机房部署等诸多原因，网络请求速度存在不可控因素。
- **稳定**：需要尽可能避免网络抖动、机房故障等影响，进一步控制可用性影响因素。

1.2 规格信息

- 包大小：~500M
- 最小人脸检测大小：50px * 50px
- 可识别人脸角度：yaw ≤ ±30°, pitch ≤ ±30°
- 检测速度：100ms 720p*
- 追踪速度：30ms 720p*
- 人脸检测耗时：~= 100ms

> 备注：以上指标，由最新版SDK运行在真实设备上，采用真实数据集所得，但**算法性能受实际运行设备、实际数据集等情况影响**，以上数字仅供参考。

- 提供一个鉴权激活工具：tool_ubuntu.sh、tool_centos.sh，可根据平台选择对应的sh文件运行

1.3 兼容性

- Centos6.3 gcc4.8.2 以及 Ubuntu16.04 gcc5.4.0上编译
- 推荐基于vs2015进行开发

1.4 授权方式

按设备授权

离线识别SDK授权方式为以设备维度为主，每台硬件设备需要一个独立的授权，此授权的校验是基于设备的硬件指纹（指纹的获取SDK初始化时会自动读取并展示），被授权的设备，将在有效期内可以运行SDK。

重新拉取授权的情况：设备授权不变，仅需要重新激活而已。

- 删除SDK或基于SDK开发的应用
- 重新安装Windows系统

授权失效的情况：需要重新购买序列号，之前的序列号失效。

- 激活一台设备后，此设备硬件变更
- 硬件损坏

序列号

序列号为管理授权的依据。每台被授权的设备，都将对应一个序列号，用于标识对应的设备信息及授权记录。序列号的形式为16位随机英文数字组合，如：`3G59-M5JK-889A-7LQA`。您在管理后台购买SDK授权时，购买成功后系统将会发放对应数量的序列号。序列号不限制平台版本，任何开发平台的离线SDK，都可以使用此序列号激活。序列号不限制账号，可供任何设备激活使用。

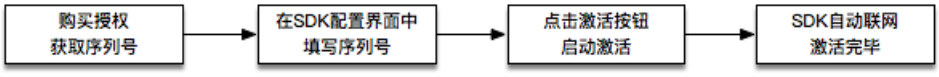
激活

已购买的序列号，是用于激活的唯一凭证，激活流程主要是将序列号与具体的硬件进行绑定（硬件指纹），从而生成对应硬件设备的授权文件（License），SDK运行前，将会校验授权文件是否和实际硬件信息相匹配。

> **注意**：激活时，设备系统时间需要和当前时间一致，如果差距太大（例如偏差5min以上），激活则无法完成。

联网激活

此种激活方式，适用于设备可首次联网的情况，优势在于激活过程极为简单。您只需将SDK安装到需要激活的设备上，然后填写已经购买的序列号，在界面上点击激活即可（为使用方便，我们为您设计了一个简单的激活用户界面）



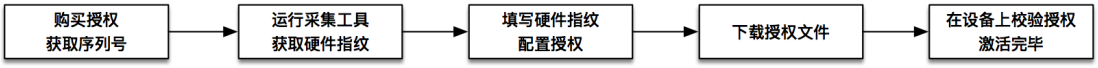
1. 获取序列号：从管理后台购买获取序列号。
2. SDK中填写序列号：将序列号填写到SDK的可视化界面中。
3. 启动激活：点击「激活」按钮。

4. SDK自动激活：SDK自动拉取授权文件并完成授权，激活完毕。

如激活成功，将会立即在界面上有明确的弹窗提示，请留意查看；如激活失败，也会反馈具体的错误信息。

离线激活

此种激活方式，适用于设备完全不可联网的情况，优势在于可避免联网激活，满足业务对网络的严格要求，以及设备批量注册需求。您需要在后台配置好硬件指纹并完成和序列号的绑定，然后将授权文件放到SDK的指定位置。



1. 获取序列号：从管理后台购买获取序列号。

2. 采集硬件指纹：将SDK置于设备上，运行激活程序，获取硬件指纹。

3. 配置授权：在后台将硬件指纹绑定到具体序列号上。

4. 下载授权文件：绑定成功后下载授权文件。

5. 设备激活：将授权文件放到SDK中，并初始化SDK完成授权。

授权有效期

申请通过后，每个账户给**2个测试序列号**，用于激活及SDK试用，有效期为**自激活日期后的3个月**。这两个序列号在有效期内完全免费，您可以用于进行产品试用。试用期到期后也可以在后台申请延期，填写具体延期理由即可。

正式购买

正式购买的序列号，试用期限为**永久有效**。此「永久」是指绑定到具体设备维度，但如已绑定的硬件设备变更后，授权则可能会失效。

1.5 定价方式

离线SDK的授权基于设备维度，每个序列号仅可以授权一台设备。每个账号购买的序列号会累计计算，**累计购买量**越多，单价越便宜。具体如下所示：

累计购买授权数量	每个授权单价
0~1000	299元/个
1001~5000	249元/个
>5000	199元/个

立即去购买

2、SDK详细介绍

2.1 名词解释

名词	定义
SDK	Linux离线人脸识别sdk
gcc	Linux平台下c++代码编译器
License	人脸识别激活所需要的激活文件,可利用激活工具生成
key	人脸激活所需的序列号，可从百度官网申请及购买（ai.baidu.com）
feature	人脸特征值，用来表示人脸特征的512个浮点值
face_token	对应人脸图片的唯一编码，若一个人上传了2张不同图片，则可能有2个不同的face_token,它和图片一一对应

2.2 SDK简介

本SDK适应于Linux（Centos、Ubuntu）平台下的人脸识别系统,分别在Centos6.3 gcc4.8.2 以及Ubuntu16.04 gcc5.4.0上编译（其他版本或平台的linux不保证兼容）。SDK采用动态库so方式提供给开发者，另外随sdk附带一个鉴权激活工具（可运行脚本 tool-ubuntu.sh或tool-centos运行，这2文件分别对应不同的OS），通过该激活工具可生成正常接入SDK的激活license文件（生成两个文件license.ini和license.key）达到通过鉴权，正常使用SDK的目的。

2.3 SDK文件结构

Sdk提供动态库BaiduFaceApi.so及头文件baidu_face_api.h。另外有附带的demo例子代码test-face-api以及鉴权激活工具tool。在此之外，还附带支撑sdk使用的人脸识别模型文件等在文件夹face-resource中，该文件夹在test-face-api例子工程里面(该文件夹存放路径请参考test-face-api，默认存放路径为您要开发的可执行文件路径同一目录下。若存放不对，可能会影响sdk正常使用,另外请勿删除该文件夹)。

为运行您开发的可执行文件，需要用到一些底层的库文件支持,相关库文件在test-face-api的lib3目录中（主要有opencv库，ffmpeg库，json，curl库等）。此外，随工程还有一些编译用的Makefile文件及sh脚本文件等，可通过Makefile编译工程，sh脚本文件运行编译的可执行文件。Makefile文件分别命名为Makefile_centos6.3及Makefile_ubuntu，若要编译相应平台版本，请把他重命名为Makefile。

2.4 激活工具（tool）

在百度官网申请系列号Key后，在test-face-api工程下有个tool_ubuntu.sh和tool_centos.sh。可根据您的OS平台选择对应sh文件运行，在运行之前，先修改脚本文件的最后一行，把如05QU-ATMA-SCWY-CAUB字样的key修改为您在百度官网申请到的key，然后再运行sh脚本文件（若是centos平台，输入sh tool_centos.sh即可）。激活成功后会在激活目录生成license.ini以及license.key文件，这2个文件是作为sdk通过鉴权使用的配置文件，请勿删除。

如果你不方便使用网络，可以通过离线激活进行操作：

通过如上的激活工具，获取到设备指纹信息，通过序列号管理后台找到需要绑定的序列号，选择「离线激活」，填入指纹设备信息，即可下载获取到license.ini文件和license.key文件，把这2个文件拷贝到test-face-api目录下，运行可执行文件亦可通过鉴权。

2.5 Demo示例工程

Demo示例工程代码在test-face-api中的cpp目录里面，其中的test_face.cpp里面有入口方法main()，展示了如何集成百度人脸识别离线SDK。即正确编写Makefile文件及引入头文件include以及对应的一些库文件。另外为了示例视频人脸跟踪等，用到了一些opencv的库文件以及一些实现demo的支持文件，如json等。（这些支持文件均为代码开源或是开源库）

在test-face-api中的test_face.cpp的main()方法中，有使用sdk的各个接口方法示例。接入sdk及其简单，分3步3行代码。如下：

```
 BaiduFace *api = new BaiduFaceApi();
// 第一步：实例化人脸SDK

api->sdk_init();
// 第二步：初始化人脸SDK

std::string res = api->face_attr("方法的传入参数，此处省略");
// 第三步，调用功能接口，user_add为人脸注册接口，res为调用功能接口后的返回。
```

如上，即为调用sdk功能的最简单3步3行代码，当然调用SDK后，在程序退出后，需要释放sdk防止内存泄漏，需要删除SDK实例化的指针delete api;

示例工程中：分别有以下几个示例cpp文件对应几个常用sdk的调用demo。

文件名	功能
setting.cpp	人脸检测、识别等参数设置
compare.cpp	人脸1:1比对、1:N 搜索，人脸特征值提取，特征值对比等
liveness.cpp	USB摄像头视频信息人脸实时检测，图片检测人脸信息等
cv_help.cpp	绘制人脸跟踪框、人脸关键点位等的工具类

2.6 特征抽取模型选择

模型介绍

v1.1版本起，SDK提供生活照和证件照两种特征抽取模型，主要适用场景如下:

- **生活照模型**：如手机拍摄的图片、较为清晰的证件照片、USB镜头实时采集的图片、网络摄像头实时采集的图片等。
- **证件照模型**：如身份证芯片照、各类证件照（工卡、学生卡、会员卡照片等）、人脸的像素普遍小于80px的图片等。

使用方法

在SDK初始化方法中控制：sdk_init(true); //**传入true为使用证件照模型，传入fasle为普通生活照模型**

注意事项

温馨提示：一经选择一个模型，则所有业务流程的特征抽取处理，都会使用此模型，两个模型不可同时作用。**如业务中设计证件照的特征抽取，请务必选择使用证件照模型**。

3、功能接口

SDK实现的主要功能有人脸实时跟踪检测、人脸特征值提取、1:1人脸对比、特征值的比对和通过USB或笔记本自带摄像头检测视频帧，返回识别出的人脸信息等，另外支持对人脸检测进行设置，达到根据设置进行识别的目的。各接口功能及传入参数和返回结果（返回结果一般为json格式的字符串）描述如下：

3.1 人脸检测及设置

3.1.1 人脸检测track接口(传入图片)

方法名

`track`

方法说明

人脸检测，返回人脸信息

请求信息

参数	必须	类型	描述
out	是	std::vector	**返回的检测到的人脸图片信息结构体** std::vector *& out 通过传入TrackFaceInfo的结构体指针引用，在检测到人脸信息后，用该引用返回人脸信息。TrackedFaceInfo结构体为检测到的人脸信息，其中：box为检测到的人脸框（为FaceInfo结构体,包含人脸框大小，中心点坐标，人脸宽度等信息）landmarks为检测到的人脸关键点，一般为144个。score为人脸打分headPose的向量组为人脸x,y,z的三个角度
image	是	string	人脸图片信息，根据image_type，传入图片内容
img_type	是	int	传入的图片类型。为1时候表示base64编码的图片，为2时候表示传入图片的本地路径。BASE64：图片的base64值；FACE_FILE：图片的本地文件路径地址；
maxTrackObjNum	是	int	最多检测人脸数量，默认为1，最大不超过5

返回信息

返回结果为int，表示检测到几个人脸信息，1为1个人，0为没检测到人脸。

3.1.2 人脸检测track接口(传入二进制图片buffer)

方法名

`track_by_buf`

方法说明

人脸检测，返回人脸信息

请求信息

参数	必须	类型	描述
out	是	std::vector	**返回的检测到的人脸图片信息结构体** std::vector *& out 通过传入TrackFaceInfo的结构体指针引用，在检测到人脸信息后，用该引用返回人脸信息。TrackedFaceInfo结构体为检测到的人脸信息，其中：box为检测到的人脸框（为FaceInfo结构体,包含人脸框大小，中心点坐标，人脸宽度等信息）landmarks为检测到的人脸关键点，一般为144个。score为人脸打分headPose的向量组为人脸x,y,z的三个角度
image	是	Unsigned char *	人脸图片信息，二进制图片buffer内容
size	是	int	二进制图片的大小
maxTrackObjNum	是	int	最多检测人脸数量，默认为1，最大不超过5

返回信息

返回结果为int，表示检测到几个人脸信息，1为1个人，0为没检测到人脸。

3.1.3 人脸检测track_max_face接口

方法名

`track_max_face`

方法说明

检测最大人脸。

请求信息

参数	必须	类型	描述
out	是	std::vector<TrackFaceInfo>	**返回的检测到的人脸图片信息结构体** std::vector *& out 通过传入TrackFaceInfo的结构体指针引用，在检测到人脸信息后，用该引用返回人脸信息。TrackedFaceInfo结构体为检测到的人脸信息，其中：box为检测到的人脸框（为FaceInfo结构体,包含人脸框大小，中心点坐标，人脸宽度等信息）；landmarks为检测到的人脸关键点，一般为144个；score为人脸打分；headPose的向量组为人脸x,y,z的三个角度
image	是	string	图片信息（数据大小应小于10M）
image_type	是	int	为1时候表示base64编码的图片；为2时候表示传入图片的本地路径。 BASE64：图片的base64值；FACE_FILE：图片的本地文件路径地址；

返回信息

返回结果为int，表示检测到几个人脸信息，1为1个人，0为没检测到人脸。

3.1.4 人脸检测track_max_face接口(传入二进制图片buffer)

方法名

`track_max_face_by_buf`

方法说明

检测最大人脸。

请求信息

参数	必须	类型	描述
out	是	std::vector<TrackFaceInfo>	**返回的检测到的人脸图片信息结构体** std::vector *& out 通过传入TrackFaceInfo的结构体指针引用，在检测到人脸信息后，用该引用返回人脸信息。TrackedFaceInfo结构体为检测到的人脸信息，其中：box为检测到的人脸框（为FaceInfo结构体,包含人脸框大小，中心点坐标，人脸宽度等信息）；landmarks为检测到的人脸关键点，一般为144个；score为人脸打分；headPose的向量组为人脸x,y,z的三个角度
image	是	Unsigned char *	人脸图片信息，二进制图片buffer内容
size	是	int	二进制图片的大小

返回信息

返回结果为int，表示检测到几个人脸信息，1为1个人，0为没检测到人脸。

3.1.5 人脸检测设置接口

请参考SDK工程中的代码示例及头文件`baidu_face_api.h`中的定义及`setting.cpp`里的代码注释。

3.1.6 USB摄像头检测

请参考TestFaceApi中的示例liveness.cpp中的`usb_track_face_info`，该方法中用到了人脸检测track视频帧，接口如下3.1.7：

3.1.7 人脸检测track接口(传入opencv的mat)

方法名

`Track`

方法说明

人脸检测，返回人脸信息。

请求信息

参数	必须	类型	描述
out	是	`std::vector<TrackFaceInfo>`	**返回的检测到的人脸图片信息结构体** std::vector *& out 通过传入TrackFaceInfo的结构体指针引用，在检测到人脸信息后，用该引用返回人脸信息。TrackedFaceInfo结构体为检测到的人脸信息，其中：box为检测到的人脸框（为FaceInfo结构体,包含人脸框大小，中心点坐标，人脸宽度等信息）；landmarks为检测到的人脸关键点，一般为144个；score为人脸打分；headPose的向量组为人脸x,y,z的三个角度
Mat	是	Opencv格式的单帧图片mat	人脸图片信息
maxTrackObjNum	是	int	最多检测人脸数量。默认为1，最大不超过5

返回信息

返回结果为TrackedFaceInfo的向量指针,向量组为0时候表示没检测到人脸。TrackedFaceInfo结构体为检测到的人脸信息，其中：box为检测到的人脸框（为FaceInfo结构体,包含人脸框大小，中心点坐标，人脸宽度等信息）；landmarks为检测到的人脸关键点，一般为144个；score为人脸打分；headPose的向量组为人脸x,y,z的三个角度。

3.2 人脸对比及识别接口

3.2.1 人脸对比接口

方法名

`match`

方法说明

人脸对比接口（本接口中的image_type为1表示base64图片编码，为2表示文件路径，为3表示face_token）

请求信息

参数	必须	类型	示例描述
image1	是	string	需要对比的第一张图片，小于10M
imgae1_type	是	int	图片1类型，必选择以下三种形式之一 **BASE64**：图片的base64值； **FACE_FILE**：图片的本地文件路径地址； **FACE_TOKEN**：face_token 人脸标识；
image2	是	string	需要对比的第二张图片，小于10M
image2_type	是	int	图片2类型，必选择以下三种形式之一 **BASE64**：图片的base64值； **FACE_FILE**：图片的本地文件路径地址； **FACE_TOKEN**：face_token 人脸标识；

返回信息

errno信息：

- errno=0：Success
- errno>0：错误码对应的详细msg

data信息:

字段	类型	说明
log_id	string	请求日志标识
result	array	识别结果列表
score	string	相似度分值，0-100，百分制，保留后2位小数点

3.2.2 人脸对比接口（传入二进制图片buffer）

方法名

`match_by_buf`

方法说明

人脸对比接口（传入二进制图片buffer）

请求信息

参数	必须	类型	示例描述
image1	是	Unsigned char *	需要对比的第一张图片，小于10M
size	是	int	图片1的大小
image2	是	Unsigned char *	需要对比的第二张图片，小于10M
size	是	int	图片2的大小

返回信息

errno信息：

- errno=0：Success
- errno>0：错误码对应的详细msg

data信息:

字段	类型	说明
log_id	string	请求日志标识
result	array	识别结果列表
score	string	相似度分值，0-100，百分制，保留后2位小数点

3.3 特征值提取接口

3.3.1 特征值提取接口（通过传入图片）

方法名

`get_face_feature`

方法说明

提取人脸特征值，为512个浮点值，已加密 (本接口image_type 为1表示base64图片编码 为2表示文件路径)

请求信息

参数	必须	类型	示例描述
image	是	string	需要检测的人脸图片，小于10M, 图片类型根据image_type参数定
imgae_type	是	int	图片类型，必选择以下2种形式之一。 image_type为1代表BASE64，为2代表FACE_FILE。 **BASE64**：图片的base64值； **FACE_FILE**：图片的本地文件路径地址；
feature	是	const float*	通过传入const float*指针的引用，来返回提取的人脸特征值

返回信息

errno信息：

- errno=0：Success
- errno>0：错误码对应的详细msg

data信息:

int类型的特征抽取向量数。正常提取成功则返回特征值向量个数（512），若不为512则为返回的错误码.

3.3.2 特征值提取接口（通过传入二进制图片buffer）

方法名

`get_face_feature_by_buf`

方法说明

提取人脸特征值，为512个浮点值，已加密 (传入二进制图片buffer)

请求信息

参数	必须	类型	示例描述
image	是	Unsigned char*	需要检测的人脸图片，小于10M
size	是	int	二进制图片大小
feature	是	const float*	通过传入const float*指针的引用，来返回提取的人脸特征值

返回信息

errno信息：

- errno=0：Success
- errno>0：错误码对应的详细msg

data信息:

int类型的特征抽取向量数。正常提取成功则返回特征值向量个数（512），若不为512则为返回的错误码.

3.3.3 特征值提取接口（通过传入opencv的视频帧）

方法名

`get_face_feature`

方法说明

提取人脸特征值，为512个浮点值，已加密 (本接口image_type 为1表示base64图片编码 为2表示文件路径)

请求信息

参数	必须	类型	示例描述
Mat	是	string	需要提取特征值的OpenCV视频帧（单帧）
feature	是	const float*	通过传入const float*指针的引用，来返回提取的人脸特征值

返回信息

errno信息：

- errno=0：Success
- errno>0：错误码对应的详细msg

data信息:

int类型的特征抽取向量数。正常提取成功则返回特征值向量个数（512），若不为512则为返回的错误码.

3.3.4 特征值比对接口

方法名

`compare_feature`

方法说明

对人脸特征值进行比较，可返回人脸特征相似分值（百分制）

请求信息

参数	必须	类型	示例描述
feature1	是	std::vector	512个浮点型的特征值，传入const的特征值引用
feature2	是	std::vector	512个浮点型的特征值，传入const的特征值引用

返回信息

errno信息：

- errno=0：Success
- errno>0：错误码对应的详细msg

data信息:

相似度结果：float类型，百分制的分值

3.4 属性及质量检测接口

3.4.1 人脸属性（通过传入图片）

方法名

`face_attr`

方法说明

人脸属性检测接口(本接口 image_type 为1表示base64图片编码 为2表示文件路径)

请求信息

参数	必须	类型	示例描述
image	是	string	需要检测的人脸图片，小于10M, 图片类型根据image_type参数定
imgae_type	是	int	图片类型，必选择以下2种形式之一。 Image_type为1代表BASE64为2代表FACE_FILE。 **BASE64**：图片的base64值； **FACE_FILE**：图片的本地文件路径地址；

返回信息

errno信息：

- errno=0：Success
- errno>0：错误码对应的详细msg

data信息:

字段	类型	说明
age（年龄）	int	人脸年龄范围0-100，若为-1表示不完整的人脸。
race（种族）	int	正常人脸标注 0：黄种人 1：白种人 2：黑人 3：阿拉伯人，不完整的人脸标注-1
expression(表情)	int	0：中性表情，1：微笑，2：大笑
gender(性别)	int	0: 女 female，1: 男 male，-1: 婴儿或不好辨别性别
glasses(是否戴眼镜)	int	0：不带眼镜 no glasses，1：普通透明眼镜 glasses，2：太阳镜 sunGlasses

3.4.2 人脸属性（通过传入二进制图片buffer)

方法名

`face_attr_by_buf`

方法说明

人脸属性检测接口

请求信息

参数	必须	类型	示例描述
image	是	unsigned char *	需要检测的图片，小于10M，二进制图片buffer
size	是	int	二进制图片大小

返回信息

errno信息：

- errno=0：Success
- errno>0：错误码对应的详细msg

data信息:

字段	类型	说明
age（年龄）	int	人脸年龄范围0-100，若为-1表示不完整的人脸。
race（种族）	int	正常人脸标注 0：黄种人 1：白种人 2：黑人 3：阿拉伯人，不完整的人脸标注-1
expression(表情)	int	0：中性表情，1：微笑，2：大笑
gender(性别)	int	0: 女 female，1: 男 male，-1: 婴儿或不好辨别性别
glasses(是否戴眼镜)	int	0：不带眼镜 no glasses，1：普通透明眼镜 glasses，2：太阳镜 sunGlasses

3.4.3 人脸质量接口（通过传入图片）

方法名

face_quality

方法说明

人脸质量检测接口

请求信息

参数	必须	类型	示例描述
image	是	string	需要检测的人脸图片，小于10M, 图片类型根据image_type参数定
imgae_type	是	int	图片类型，必选择以下2种形式之一。 Image_type为1代表BASE64为2代表FACE_FILE。 **BASE64**：图片的base64值； **FACE_FILE**：图片的本地文件路径地址；

返回信息

errno信息：

- errno=0：Success
- errno>0：错误码对应的详细msg

data信息:

字段	类型	说明
bluriness	float	光照
illum	float	模糊度
occl_l_eye	float	左眼遮挡度
occl_r_eye	float	右眼遮挡度
occl_nose	float	鼻子遮挡度
occl_mouth	float	嘴巴遮挡度
occl_l_contour	float	左脸遮挡度
occl_r_contour	float	右脸遮挡度
occl_chin	float	下巴遮挡度

3.4.4 人脸质量接口（通过传入二进制图片buffer）

方法名

face_quality_by_buf

方法说明

人脸质量检测接口

请求信息

参数	必须	类型	示例描述
image	是	unsigned char *	需要检测的图片，小于10M，二进制图片buffer
size	是	int	二进制图片大小

返回信息

errno信息：

- errno=0：Success
- errno>0：错误码对应的详细msg

data信息:

字段	类型	说明
bluriness	float	光照
illum	float	模糊度
occl_l_eye	float	左眼遮挡度
occl_r_eye	float	右眼遮挡度
occl_nose	float	鼻子遮挡度
occl_mouth	float	嘴巴遮挡度
occl_l_contour	float	左脸遮挡度
occl_r_contour	float	右脸遮挡度
occl_chin	float	下巴遮挡度

4、错误码及错误信息

错误码	错误内容	错误描述
0	SUCCESS	成功
1	SYSTEM ERROR	系统错误
2	UNKNOWN ERROR	未知错误
1001	NOT_AUTH	授权校验失败
1002	REQUEST PARAMS ERROR	请求参数错误
1003	DB_OP_FAILED	数据库操作失败
1004	NO_DATA	没有数据
1005	RECORD_UNEXIST	记录不存在
1006	RECORD_ALREADY_EXIST	记录已经存在
1007	FILE_NOT_EXIST	文件不存在
1008	GET_FEATURE_FAIL	提取特征值失败
1009	FILE_TOO_BIG	文件太大
1010	FACE_RESOURCE_NOT_EXIST	人脸资源文件不存在
1011	FEATURE_LEN_ERROR	特征值长度错误
1012	DETECT_NO_FACE	未检测到人脸

5、常见问题

Q：激活工具激活时候，提示timeout或者unsupport protocl

A：若提示timeout，请确认设备是否能联网；若提示unsupport protocl，请确认是否安装了openssl。

Q：工程运行过程中，提示face-resource不存在

A：SDK需要一些模型文件，在demo工程的face-resource文件夹中，该文件夹需要放置在exe所在路径的上级目录下。若放置不正确，可能出现找不到模型文件，没法进行人脸识别。

Q：工程激活过程中，提示segment fault

A：这是因为工程若没有激活，便直接执行脚本文件运行可执行文件，请先完成激活操作。

Q：激活后是否可以把激活文件license.ini和license.key拷贝到其他设备运行？

A：不能，离线sdk和设备绑定，每个设备对应一个key和一个license文件，换设备无法运行。

人脸注册工具平台

人脸注册工具平台

平台概述

人脸识别落地的共有三个核心环节：采集人脸图片注册人脸库—>获取用户现场照片进行人脸比对—>基于比对结果进行业务处理。人脸注册工具平台可以帮助开发者高效完成第一个环节，通过可视化方式快速生成进行用户信息采集的H5和微信小程序页面，用户填写的信息后将通过百度的数据回传服务转发至开发者的服务器上，保证开发者可以可以像自己开发一套完整服务一样拥有对数据的控制权。

接下来将讲述数据回传服务的配置方式。

数据回传配置说明

1、数据接收服务配置

为方便开发者接收人脸注册工具所采集数据，对接系统业务，开发者需按照此规范来开发数据接收服务。 用户提供的http或https请求地址（在编辑项目第五步时指定）且接收地址需要为可访问的公网地址，此地址以POST方式接收数据。

> 注意：
>
> 1、为了保证数据安全以及方便区分不同项目之间的数据，我们定义了项目加密token，在创建完成项目后，在项目列表页可以看到。
>
> 2、不同项目定义的参数名称不同，在设计数据接收服务时，请参考编辑项目第五步右侧的参数列表，当项目更新时，此参数列表也会同步更新

接口地址：用户提供公网可访问 请求方式：POST 请求格式：application/json

百度侧输出的的参数说明：

参数名称	数据类型	必填	备注	样例
logId	Long	是	唯一推送操作ID	153267755774325
projectToken	String	是	唯一方案token， 可用于与客户系统项目关联， 创建完成项目后， 在项目列表页可以看到	F004231C63ED40FD8FB092B497E7CB5B
optType	Int	是	操作类型 1：审核通过 2：驳回 3：删除	1
applyId	Long	是	唯一报名ID	83
uUserId	String	是	自定义用户ID	15555555555-0000000001
appId	Long	是	人脸库ID	1482513467
groupId	String	是	人脸组名称	test_g_2
channelType	Int	是	渠道类型ID，1：H5 2：微信小程序	1
name	String	是	用户姓名	小度
telephone	String	是	用户注册手机号	15555555555
faceImage	String	是	用户人脸图链接	http://bj.bcebos.com/v1/aip-web/48EBCB80FF7C4DCA9E013942
item	String	是	动态表单项，具体字段名称 请参照会议创建页面	item1

请求示例：

```
{
"logId": 153267755774325,
"projectToken": "F004231C63ED40FD8FB092B497E7CB5B",
"optType": 1,
"applyId": 83,
"uUserId": 20,
"appId": 1482513467,
"groupId": "test_g_2",
"channelType": 1,
"name": "小度",
"telephone": "15555555555",
"faceImage": "http://bj.bcebos.com/v1/aip-web/48EBCB80FF7C4DCA9E0139425B76F078?authorization=bce-auth-v1/f86a2044998643b5abc89b59158bad6d/2018-07-27T07:45:30Z/-1//07fd154d3456eea9104b7c467efc0d43aceb7609d1635246d8fdb1272b0cea30",
"item1": "篮球,足球",
"item2": "0000000001"
}
```


2、数据接收状态反馈

为保障数据推送的可靠性，需要开发者在接收到数据后给百度侧反馈接收状态，若接收失败，百度侧将按照一定周期进行重复推动，直到达到推动次数的上限(目前可以推送10次)。

具体的反馈方式是：在成功接收到数据后需要返回json格式的响应结果。

Header：

参数	值
Content-Type	application/json

Body中放置返回参数，参数详情如下：

参数名称	数据类型	是否必须	备注	样例
logId	Long	是	唯一推送操作ID原样返回	153267755774325
errorCode	Int	是	0同步成功，非0同步失败	0
errorMsg	String	否	提示信息	同步成功

响应示例：


```
{
"logId": 153267755774325,
"errorCode": 0,
"errorMsg": "同步成功"
}
```


人脸产品套件-壁虎

产品和硬件介绍

感谢您关注百度人脸识别开发套件——壁虎。

壁虎是一款旨在提升人脸识别端开发效率的产品，默认配备了镜头模组、主板、算法SDK、各类外设接口，并进行了软硬件的专项调优。应用壁虎套件，您只需专注于自己的业务应用开发，基于套件构建一个外壳即可投入实际应用。

此文档主要介绍套件的基础能力和构造，方便对产品有一个完整的了解。

业务能力

常规人脸设备开发遇到哪些问题

- **软硬件选型困难**：对于初步接触的集成商或设备商，需要做大量的算法和板卡选型，这部分要综合 考虑成本、供货能力、算法和硬件的适配性等，调研周期较长。即使是老牌的设备厂商，也要考虑替换算法过程中的一些联动硬件变更（例如镜头）。
- **设备成本高**：目前基于3288板卡，一个基础门门禁的整体造价成本在1000元上下；板卡+镜头的单项成本大概要700左右。使用MTK、高通、海思等方案，研发成本比较高，整体硬件下来也要至少500起，还不一定能够保证实际效果。以上还仅是为硬件成本，不算人力研发、部署安装的成本。生产单量没到k以上，一般没有什么能力自己压价，初期采购成本不可避免的较高。
- **端效果调试困难**：由于研发队伍一般没有人脸方面的专家，所以效果调优方面面，如果算法厂商的 产品没有非常清晰的指导，整个效果调优过程往往会踩很多坑，研发人员消耗比较大大。
- **上位机集成困难**：分为自有系统，或者用户既有的系统。如果是后者，设备的数据通信，与系统之间的集成耗时也很麻烦，如果SDK没有提供较好的上位机接口，这部分需要完全自己重新构造。

壁虎怎么解决这些问题

1. 硬件成本的降低：在保障效果的同时，通过集中的采购，尽可能压低硬件价格，保持产品成本相对具备竞争力。
2. 集成周期的缩短：预设标准化软硬件调优方案，开机配置可用，减小选型和调试的成本。
3. 产品效果的保障：标准化软硬件选型，避免去学习人脸识别相关参数含义，可让算法发挥出符合业务要求的效果。
4. 产品技术的可信：百度品牌背书，在技术可信度和稳定性方面，为产品增值加分。

产品构成及功能

- 双目近红外镜头
- 7寸屏幕
- RK3288主板
- 外设接口
- 百度人脸识别离线SDK及Sample工程

SDK具体介绍详见：**SDK开发说明**

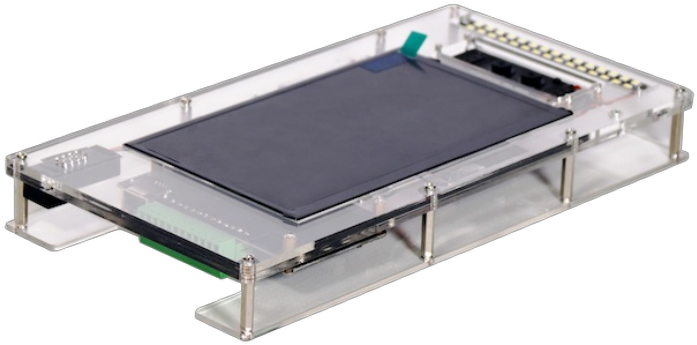
如何使用

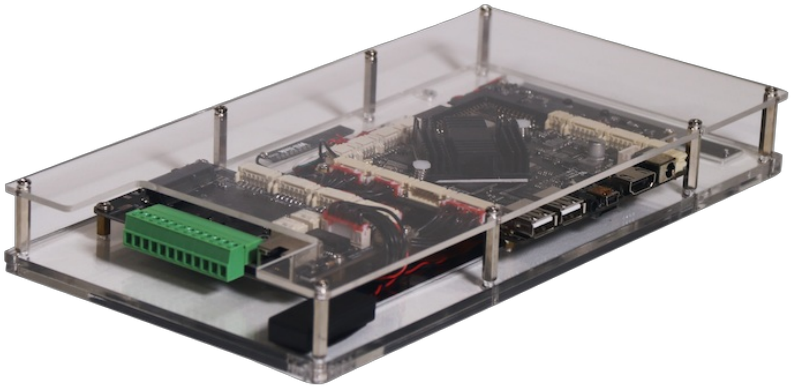
1. 购买壁虎
2. 开机验证业务功能（预装软件已激活）
3. 连接调试口（mini-usb线）
4. SD卡中预设了SDK工程，直接copy到本地，并进行项目开发

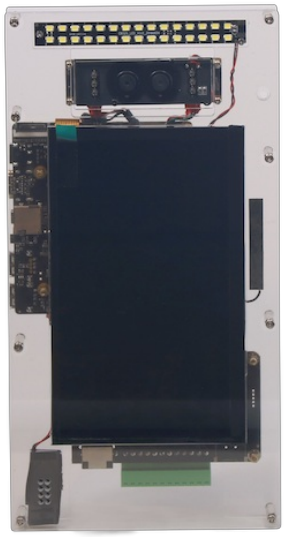
产品特色

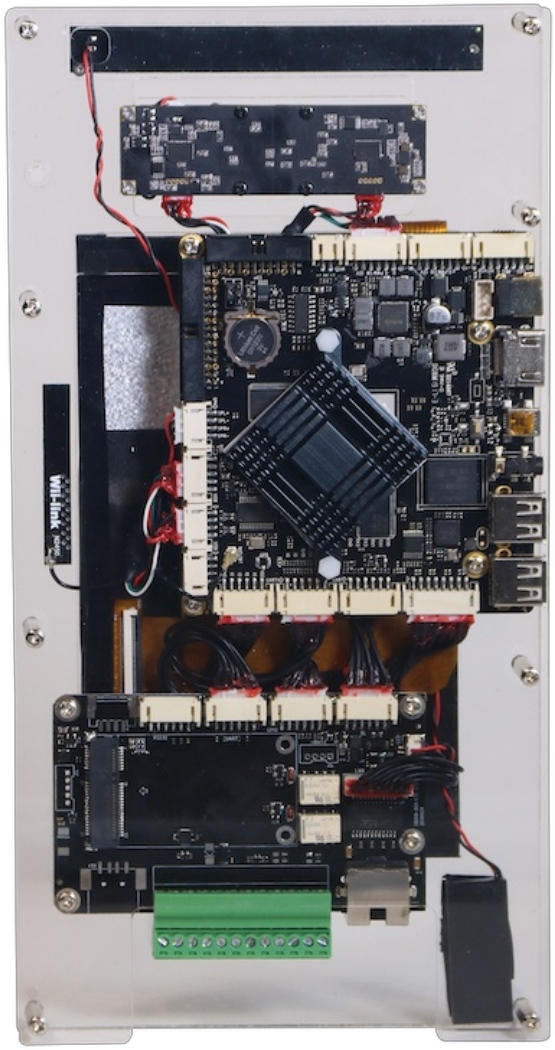
- **开机即用**：直接快速体验离线识别的功能及效果。
- **软硬调优**：硬件已经针对百度SDK进行适配优化，效果无需较大改动。
- **丰富接口**：预留闸机、门禁等产品常用的各类外设接口。
- **高性价比**：软硬件一整套售价，仅需899元。

产品外观









硬件参数

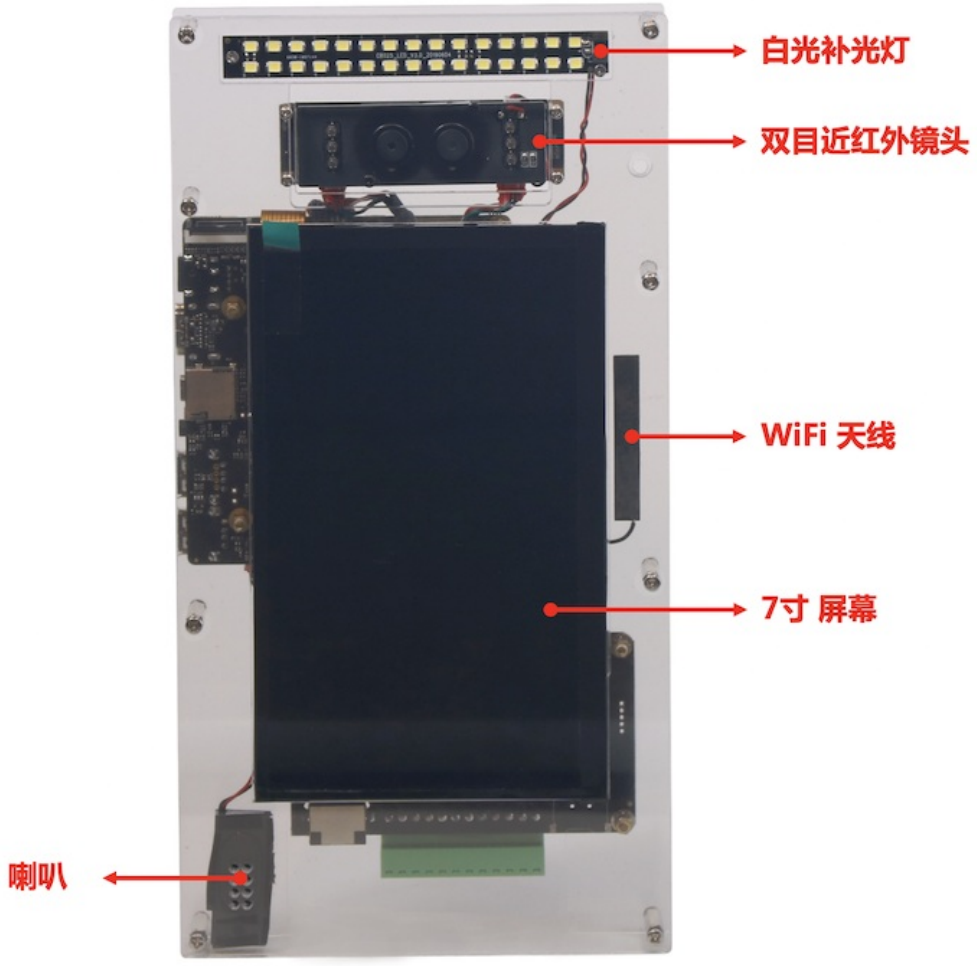
名称	功能描述
CPU	RK3288四核Cortex-A17，高性能28nm HKMG工艺
GPU	Mali-T764 GPU,支持AFBC(帧缓冲压缩）,支持 OpenGL ES 1.1/2.0/3.1, 内嵌高性能2D 加速硬件OpenCL, DirectX9.3
内存	DDR3 2G
存储容量	eMMC 8GB
显示屏	MIPI 接口；1024*600
网络	具备百兆网口，支持Ethernet 支持2.4G WiFi，Wi-Fi 支持802.11b/g/n 协议。 具备蓝牙功能 V2.1+EDR/Bluetooth3.0/3.0+HS/4.0
摄像头	双200W摄像头（RGB+IR），高亮红外补光灯
实时时钟	内置实时时钟供电电池
外设	支持1路**485**、2路**RS232** 支持 2路继电器（默认Open） 支持1路韦根 支持3个 USB HOST、1 个 USB OTG 支持TF 卡，最大支持 64GB 内置单喇叭 3W*4R 支持耳机接口输出 ，支持麦克风
输入适配器	输入：AC100-240V.50-60HZ 输出： DC12V/5.5mm 内芯2.1 mm DC头 2A－5A （要求浪涌电压小于18V，纹波电压小于100mV） 上电自启动

软件参数

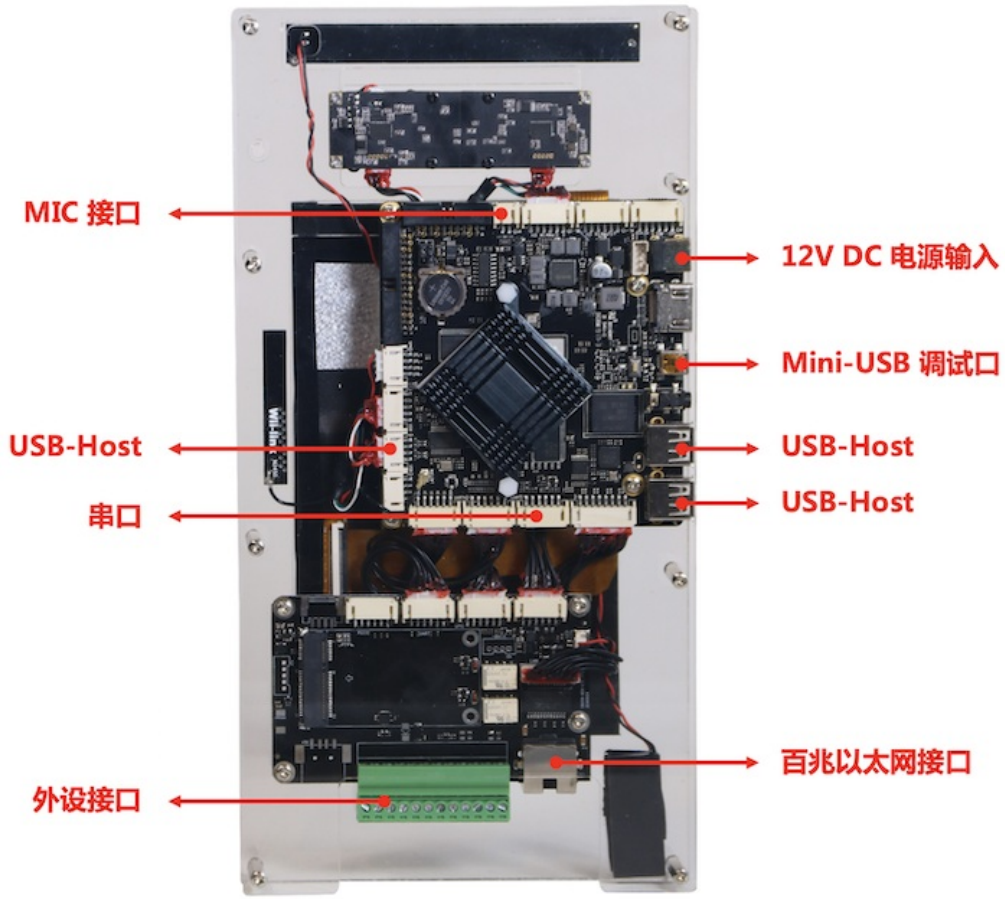
软件项	详细内容
操作系统	默认Android 5.1
基本软件功能	相机，网页浏览、资源管理器
音效模式	时钟、闹钟、录音
语言支持	多国语言
录音	支持MP3、WMA格式录音
文书处理	EPUB/WORD/EXCEL/POWERPOINT/PDF/TXT
电子书	PDF/TXT/CHM/DOC/EXCEL/EPUB/RTF/FB2
日程	日历
输入法	标准 Andriod 键盘，可选第三方输入法（中文、韩文、日文等）
软件信息	百度人脸识别离线SDK 3.0系列SDK 原生态 Android 系统，开放 root 权限，可进行产品定制开发 APK 安装器 System Setting

外设预览

产品正面



产品反面



产品定价

899元/件，无阶梯价。现在去 **立即购买**

包装说明

- 包装盒一个
- 壁虎套件一个
- 电源一个
- Mini-USB线一个

售后保障

如软硬件出现问题，可以通过以下几种方式联系我们：

- 在百度云控制台内 **提交工单**，咨询问题类型请选择 **人工智能服务**；
- 如有需要讨论的疑问，欢迎进入 **AI社区** 与其他开发者们一同交流。

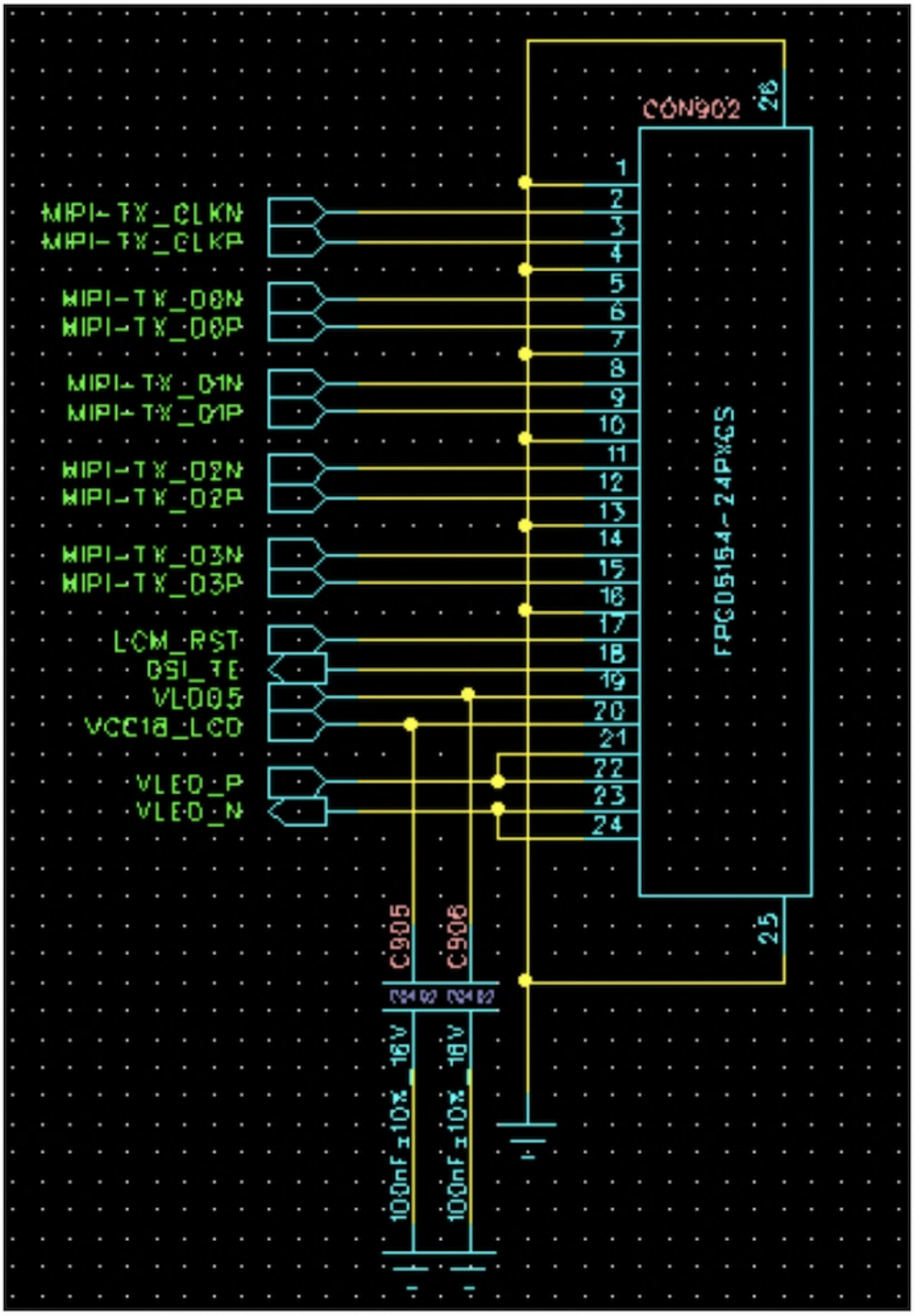
接口定义说明

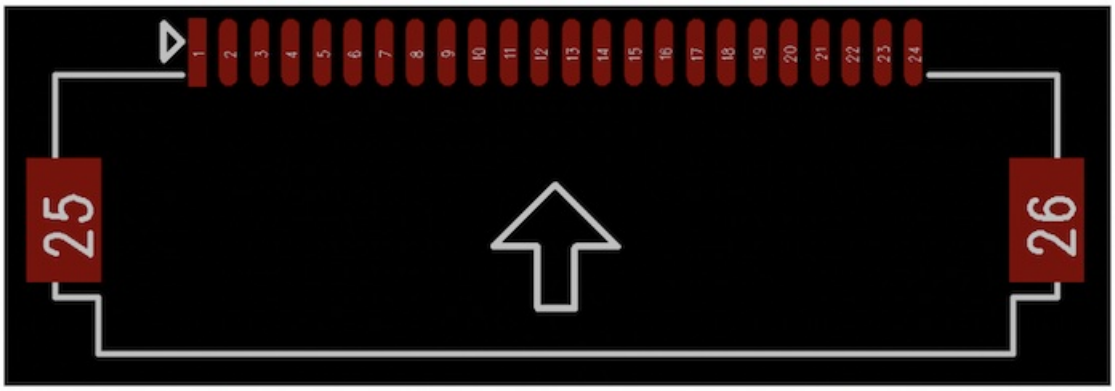
感谢您关注百度人脸识别开发套件——壁虎。

壁虎是一款旨在提升人脸识别端开发效率的产品，默认配备了镜头模组、主板、算法SDK、各类外设接口，并进行了软硬件的专项调优。应用壁虎套件，您只需专注于自己的业务应用开发，基于套件构建一个外壳即可投入实际应用。

此文档主要介绍套件的各项常用接口定义，用于进行硬件的二次开发和外设连接。

MIPI 屏幕接口


CON902
26
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
FPC05154-24PXSS
MIPI_TX_CLKN
MIPI_TX_CLKP
MIPI_TX_D0N
MIPI_TX_D0P
MIPI_TX_D1N
MIPI_TX_D1P
MIPI_TX_D2N
MIPI_TX_D2P
MIPI_TX_D3N
MIPI_TX_D3P
LCM_RST
DSI_TE
VLDO5
VCC18_LCD
VLED_P
VLED_N
C905
C906
100nF ±10% 16V
100nF ±10% 16V

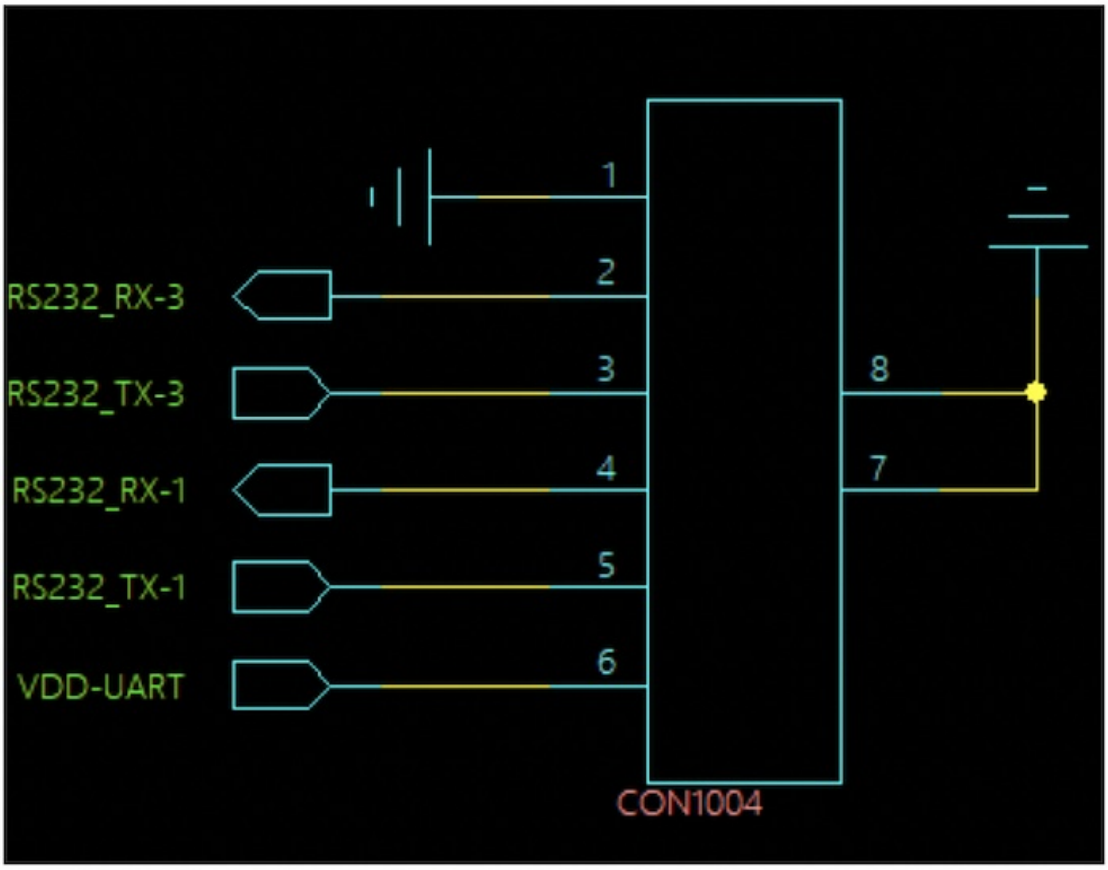

25
26

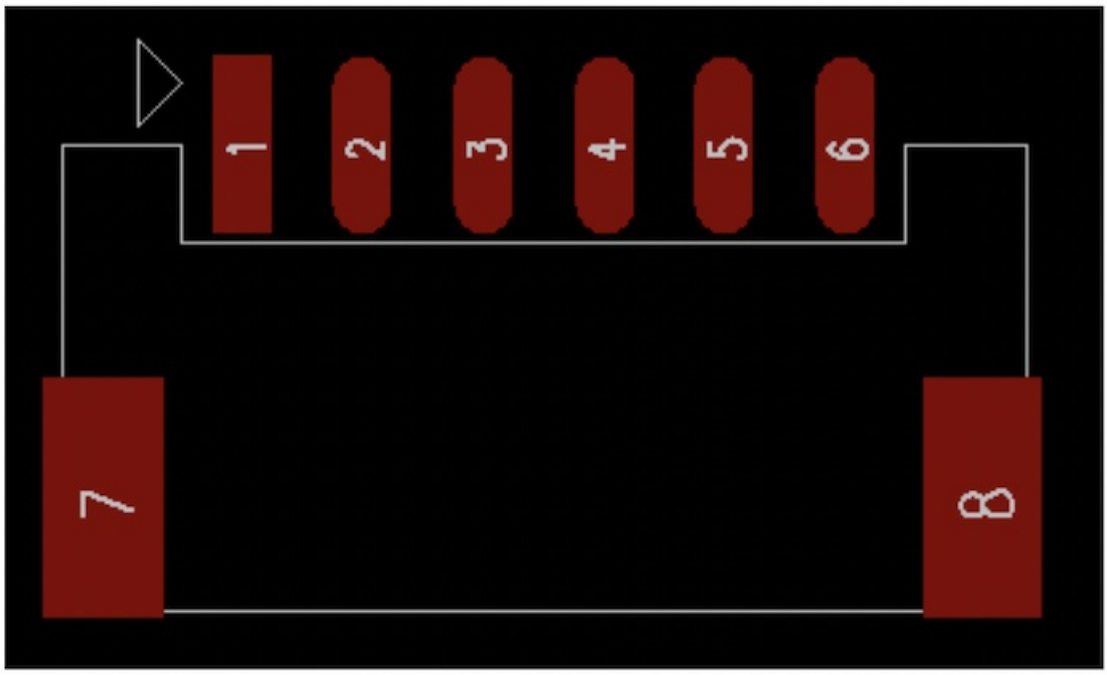
序号	定义	属性	描述
1	GND	地线	地线
2	MIPI_TX_CLKN	输出	时钟
3	MIPI_TX_CLKP	输出	时钟
4	GND	地线	地线
5	MIPI_TX_D0N	输出	数据
6	MIPI_TX_D0P	输出	数据
7	GND	地线	地线
8	MIPI_TX_D1N	输出	数据
9	MIPI_TX_D1P	输出	数据
10	GND	地线	地线
11	MIPI_TX_D2N	输出	数据
12	MIPI_TX_D2P	输出	数据
13	GND	地线	地线
14	MIPI_TX_D3N	输出	数据
15	MIPI_TX_D3P	输出	数据
16	GND	地线	地线
17	LCM_RST	输出	LCD复位脚(GPIO5_B4)
18	NC	悬空	悬空
19	VIO28_PMU	输出	LCD 2.8V输入电压
20	VIO18_PMU	输出	LCD 1.8V输入电压
21	VLED_P	地线	背光正极
22	VLED_P	输出	背光正极
23	VLED_N	输入	背光负极
24	VLED_N	地线	背光负极

串口

UART1和UART3 串口接口(UART1 UART3 JACK)（默认RS232）

接线示例如下图所示：(矩形引脚为第一脚)



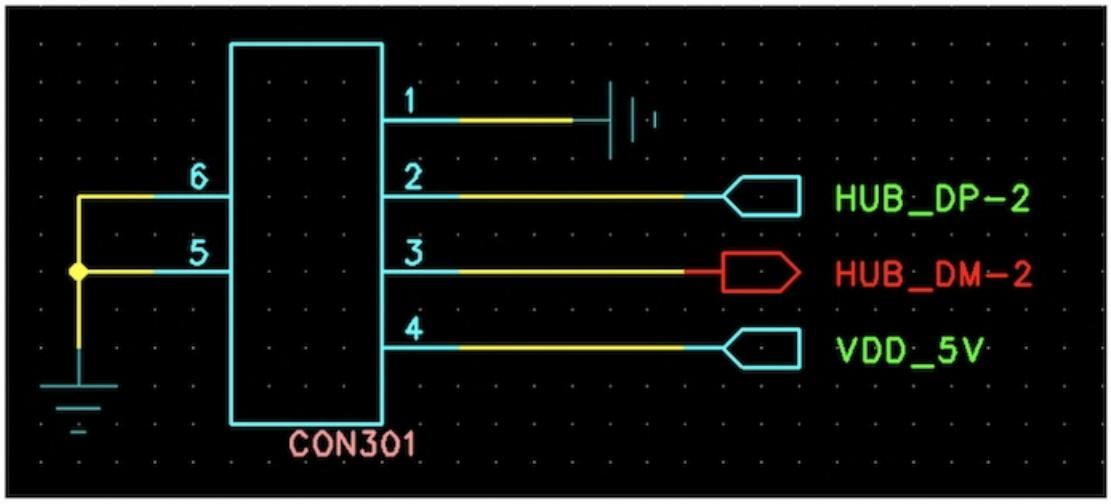


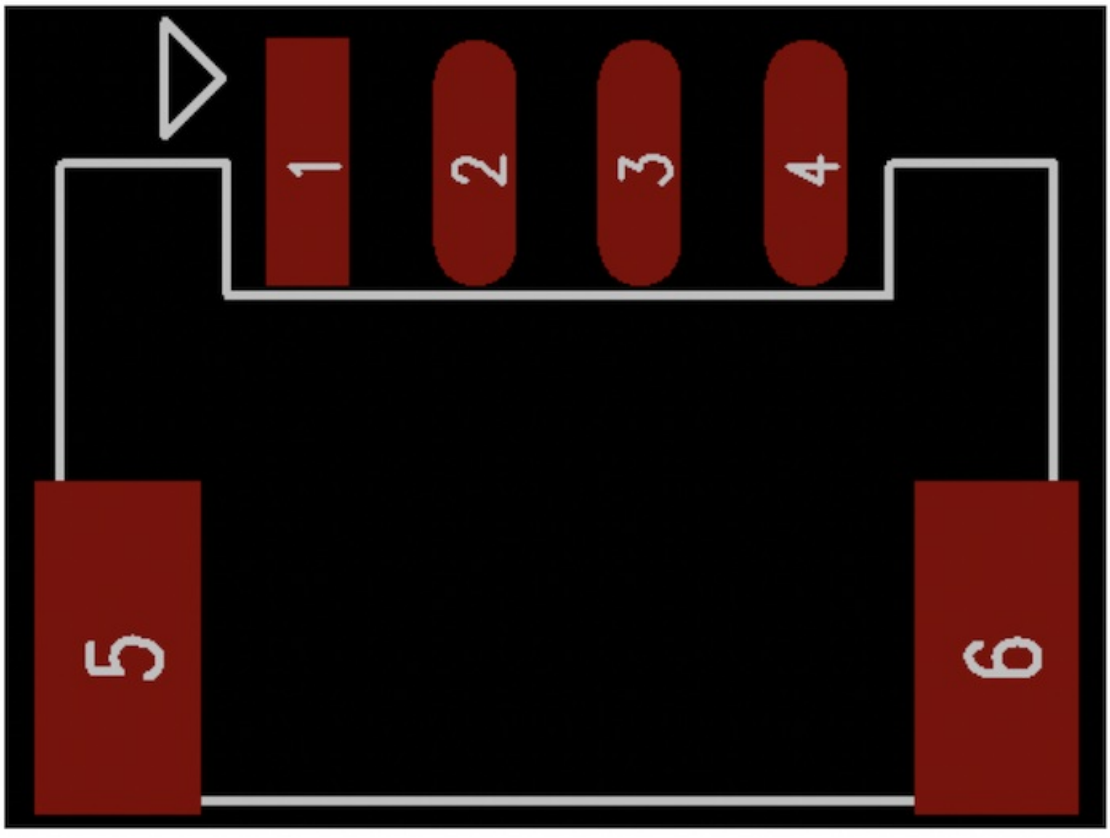
序号	RS232功能	CMOS电平	属性	描述
1	GND	GND	输入	地线
2	RS232_RX-3	UART_RX-3	输入	缺省为RS232_RX-3信号，如需配置为3.3V或者5V串口须更改硬件跳线电阻
3	RS232_TX-3	UART_TX-3	输出	缺省为RS232_TX-3信号，如需配置为3.3V或者5V串口须更改硬件跳线电阻
4	RS232_RX-1	UART_RX-1	输入	缺省为RS232_RX-1信号，如需配置为3.3V或者5V串口须更改硬件跳线电阻
5	RS232_TX-1	UART_TX-1	输出	缺省为RS232_TX-1信号，如需配置为3.3V或者5V串口须更改硬件跳线电阻
6	VDD	VDD	输出	5V 电压输出，可拉低GPIO-5-C0的电平来关闭此电源的输出

USB-HOST

CON6 USB-HUB-2 接口（USB-HOST JACK）

接线示例如下图所示：(矩形引脚为第一脚)



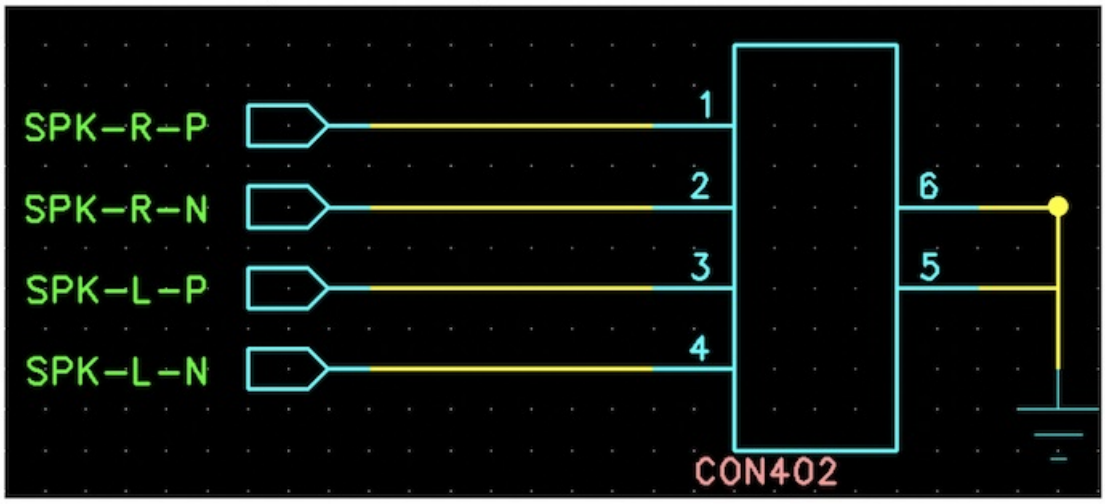


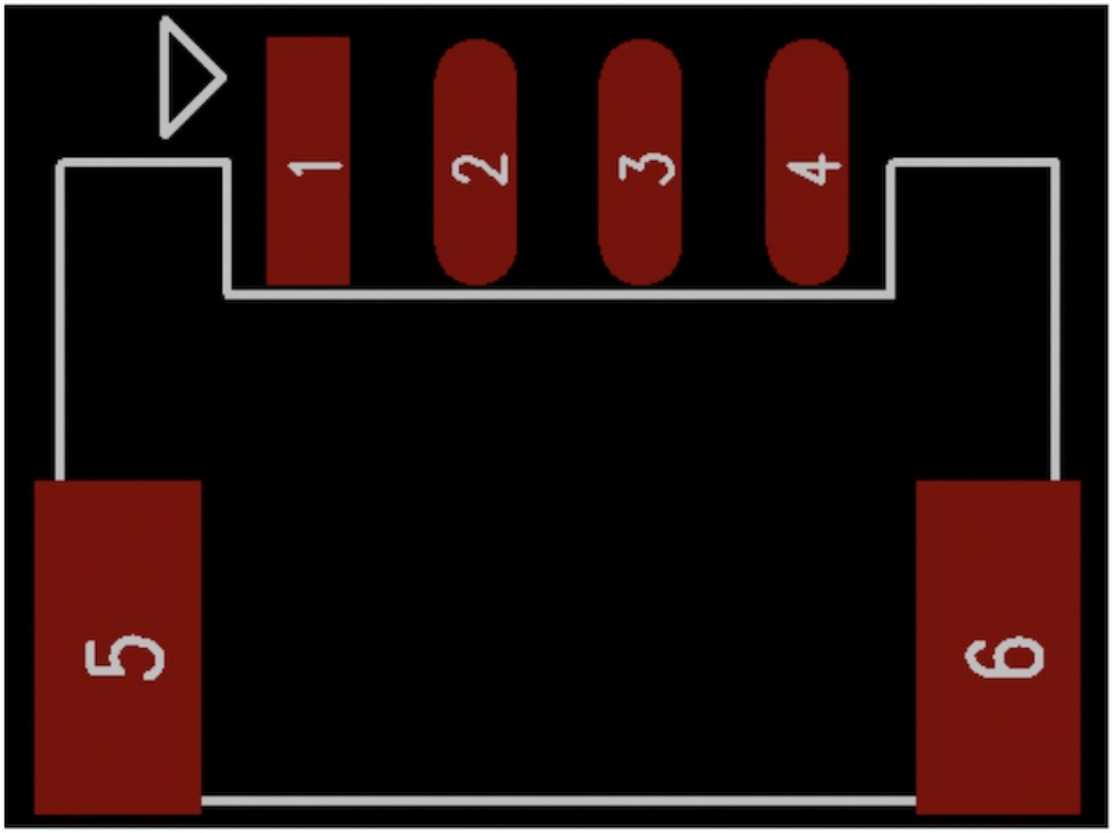
序号	定义	属性	描述
1	GND	地线	地线
2	HUB_DP-2	输入	HUB_DP-2
3	HUB_DM-2	输入	HUB_DM-2
4	5V	输出	5V 电压输出，此电源可由GPIO3-B0开关

喇叭

喇叭输出接口（SPEAKER OUT JACK）

接线示例如下图所示：(矩形引脚为第一脚)



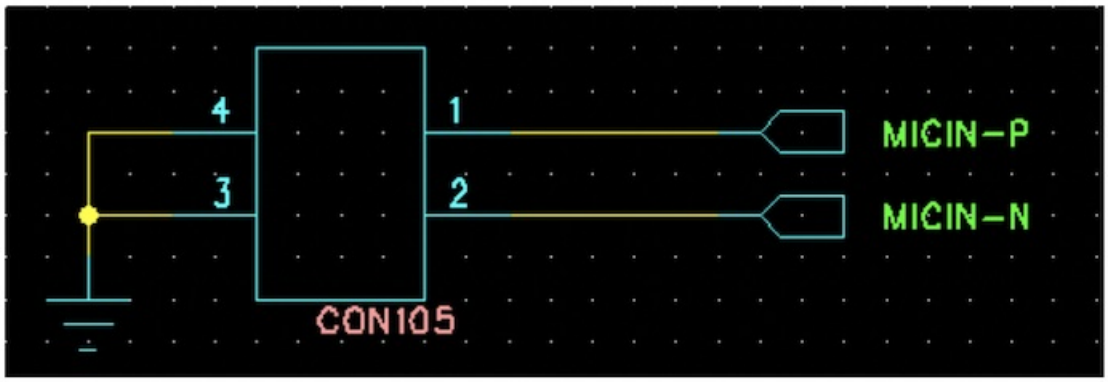


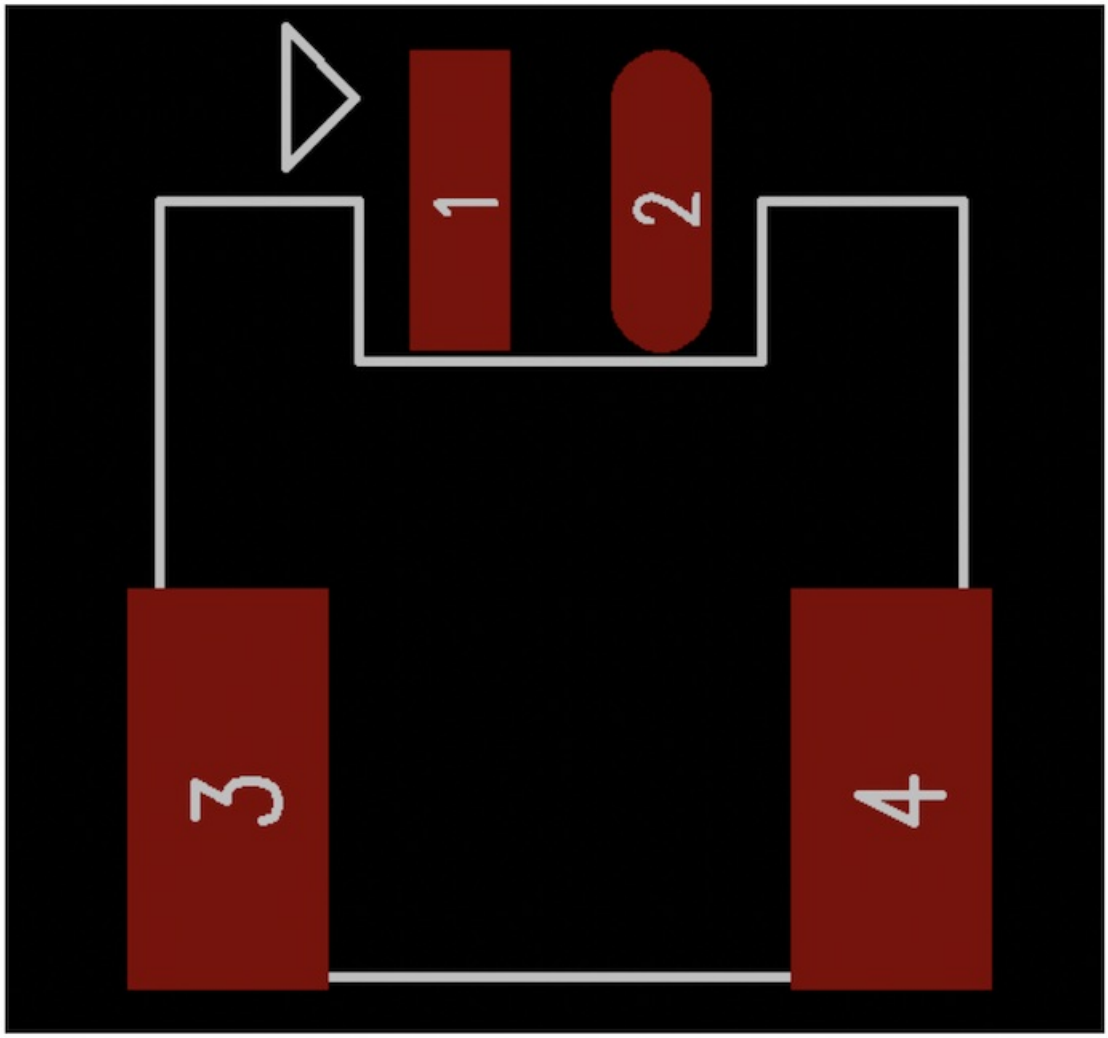
序号	定义	属性	描述
1	SPK-R-P	输出	右声道喇叭输出正极
2	SPK-R-N	输出	右声道喇叭输出负极
3	SPK-L-P	输出	左声道喇叭输出正极
4	SPK-L-N	输出	左声道喇叭输出负极

MIC

麦克风接口（MIC JACK）

接线示例如下图所示：(矩形引脚为第一脚)



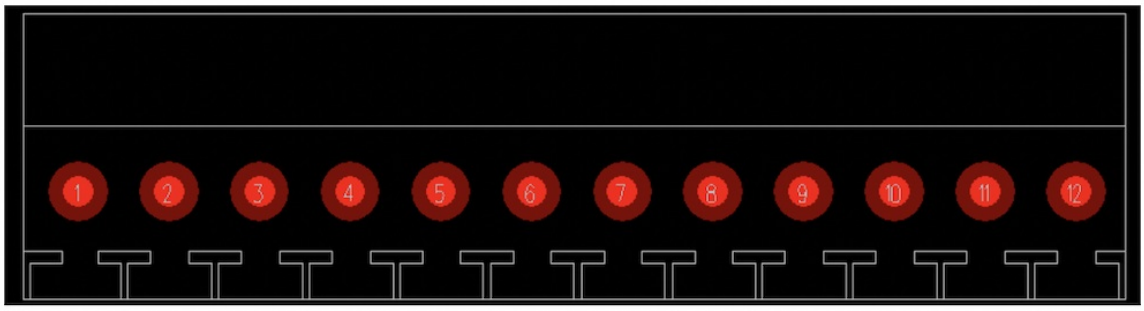


序号	定义	属性	描述
1	MIC+	输入	MIC 正极输入
2	MIC-	输入	MIC 负极输入

外设接口

外设接口（3.81端子口）

接线示例如下图所示：



序号	定义	属性	描述
1	NC	空	空
2	GND	地线	地线
3	RS485-B	RS485-B	RS485-B
4	RS485-A	RS485-A	RS485-A
5	GND	地线	地线
6	NC	空	空
7	韦根D0	韦根D0	韦根D0
8	韦根D1	韦根D1	韦根D1
9	继电器1NO	继电器1NO	继电器1NO
10	继电器1COM	继电器1COM	继电器1COM
11	继电器2NO	继电器2NO	继电器2NO
12	继电器2COM	继电器2COM	继电器2COM

实名认证

增强级人脸实名认证

增强级人脸实名认证接口

能力介绍

> 本接口需要配合增强级人脸采集SDK使用，增强级人脸采集SDK输出的图片会进行加密，在增强级人脸实名认证接口进行解密，以这种端云配合的方式防止信息传输过程中泄露或遭到第三方非法攻击。

业务能力

- **质量检测（可选）**：判断图片中是否包含人脸，以及人脸在姿态、遮挡、模糊、光照等方面是否符合识别条件。
- **活体检测（可选）**：基于图片中的破绽分析，判断其中的人脸是否为**二次翻拍**（举例：如用户A用手机拍摄了一张包含人脸的图片一，用户B翻拍了图片一得到了图片二，并用图片二伪造成用户A去进行识别操作，这种情况普遍发生在金融开户、

实名认证等环节。）。

- **人脸实名认证（必选）**：基于姓名和身份证号，调取公安权威数据源人脸图，将当前获取的人脸图片，与公安数据源人脸图进行对比，得出比对分数，并基于此进行业务判断是否为同一人。由于公安数据源人脸图具有最权威的身份证明作用，故对用户本人的验证结果可信度也最为合理。

业务逻辑

- 上述三项能力为顺序串行验证，人脸实名认证能力为必选能力，质量检测和活体检测为可选能力，如这两项可选能力都选择，则验证顺序为`人脸质量检测->活体检测->人脸实名认证`。您也可以根据业务场景，选择这两项中的某一项或都不选择。
- **如选择了质量检测和活体检测能力，则有任意一个条件不通过，整个请求流程终止，并返回错误码，描述具体的不符合信息**。
- 基于此顺序串行验证逻辑，可以避免大量不符合条件的请求流转到人脸实名认证，节约您的成本。

推荐阈值

- 此接口使用的对比算法，针对带水纹证件照采用了专项的模型处理，可保证水纹信息的影响降到尽可能低。
- 如比对成功，最终返回的有效数据为一个**对比分值**，在0~1之间，您可以设定具体的阈值来判断是否验证通过。
- **人证相似度的推荐阈值为0.8，对应的误识率为万分之一**。

计费逻辑

- 前两个环节图片不符合校验规则，会以`error_code`形式反馈，属于正向业务判断，这两个环节的请求全部免费。真正请求到人脸实名认证这步并返回结果，才会进行计费。计费标准请参考价格文档

调用方式

请求URL数据格式

向API服务地址使用POST发送请求，必须在URL中带上参数`access_token`，可通过后台的API Key和Secret Key生成，具体方式请参考“Access Token获取”。

示例代码

Bash
PHP
JAVA
Python
Cpp
C#
Node

```
##### !/bin/bash
curl -i -k 'https://aip.baidubce.com/oauth/2.0/token?grant_type=client_credentials&client_id=【百度云应用的AK】&client_secret=【百度云应用的SK】'
```

> **注意**：access_token的有效期为30天，**切记需要每30天进行定期更换，或者每次请求都拉取新token**；

例如此接口，使用HTTPS POST发送：

```
https://aip.baidubce.com/rest/2.0/face/v3/person/verifySec?access_token=24.f9ba9c5341b67688ab4added8bc91dec.2592000.1485570332.282335-8574074
```

POST中Body的参数，按照下方请求参数说明选择即可。

> **提示**：如果您为百度云老用户，正在使用其他非AI的服务，可以参考百度云AKSK鉴权方式发送请求，虽然**请求方式**和**鉴权方法**和本文所介绍的不同，但请求参数和返回结果一致。

请求说明

注意事项：

- **请求体格式化**：Content-Type为application/json，通过json格式化请求体。
- **Base64编码**：请求的图片需经过Base64编码，图片的base64编码指将图片数据编码成一串字符串，使用该字符串代替图像地址。您可以首先得到图片的二进制，然后用Base64格式编码即可。需要注意的是，图片的base64编码是不包含图片头的，如
- **图片格式**：现支持PNG、JPG、JPEG、BMP，**不支持GIF图片**

APP场景

适合于增强级采集SDK搭配请求公安数据源，验证用户的姓名、身份证号与现场采集的人脸图片是否一致的场景使用。

强调：

（1）必须搭配增强级采集SDK使用，以保证image是加密过的，否则会报错图片解密失败

（2）增强级采集SDK与接口获取token的AKSK需要来自同一应用，否则也会导致图片解密失败

请求示例

HTTP方法：POST

请求URL： `https://aip.baidubce.com/rest/2.0/face/v3/person/verifySec`

URL参数：

参数	值
access_token	通过API Key和Secret Key获取的access_token,参考“Access Token获取”

Header：

参数	值
Content-Type	application/json

Body中放置请求参数，参数详情如下：

请求参数

参数	必选	类型	说明
scene_type	否	string	APP ： APP场景，与采集SDK一起使用
image	是	string	图片信息(总数据大小应小于10M)，图片上传方式根据image_type来判断
image_type	是	string	图片类型 BASE64:图片的base64值，base64编码后的图片数据，编码后的图片大小不超过2M；图片尺寸不超过1920*1080 URL
app	是	string	ios android
name	是	string	姓名（注：需要是UTF-8编码的中文）
id_card_number	是	string	身份证号码
quality_control	否	string	图片质量控制 NONE: 不进行控制 LOW:较低的质量要求 NORMAL: 一般的质量要求 HIGH: 较高的质量要求 默认 NONE
liveness_control	否	string	活体检测控制 NONE: 不进行控制 LOW:较低的活体要求(高通过率 低攻击拒绝率) NORMAL: 一般的活体要求(平衡的攻击拒绝率, 通过率) HIGH: 较高的活体要求(高攻击拒绝率 低通过率) 默认NONE
spoofing_control	否	string	合成图控制参数 **NONE**: 不进行控制 **LOW**:较低的合成图阈值数值，由于合成图判定逻辑为大于阈值视为合成图攻击，该项代表**低通过率、高攻击拒绝率** **NORMAL**: 一般的合成图阈值数值，由于合成图判定逻辑为大于阈值视为合成图攻击，该项代表**平衡的攻击拒绝率, 通过率** **HIGH**: 较高的合成图阈值数值，由于合成图判定逻辑为大于阈值视为合成图攻击，该项代表**高通过率、低攻击拒绝率**) **默认为NONE**
risk_identify	否	boolean	Ture：打开大数据风控功能。（接受SDK端传入的指纹信息，返回识别结果） False：关闭大数据风控功能。 默认False
zid	否	string	SDK 唯一标识 ZID，从增强级SDK中获取；必选，端内活动 zid 为空会直接拒绝，请重视此参数
ip	否	string	需要风控判别的ip地址
phone	否	string	需要风控判别的手机号

请求示例：

APP场景:图片上传比对使用json格式，如下:

```
{
  "scene_type":"APP",
  "image":"/9j/4AAQSkZJRg...",  //采集SDK上传的加密后的图片
  "image_type":"BASE64",
  "id_card_number":"370***********5826",
  "name":"柴*",

 "risk_identify":"Ture",    //仅当APP场景时生效
 "ip": "...ip", //填写IP地址
 "phone": "...phone"   //填写手机号
}
```

- **返回参数**

参数	必须	类型	说明
log_id	是	uint64	日志id
verify_status	是	int	认证状态，取值： 0 姓名与身份证匹配，可进行人脸比对 1 身份证号与姓名不匹配或该身份证号不存在 2 公安网图片不存在或质量过低
score	否	float	与公安数据源人脸图相似度可能性，用于验证生活照与公安数据源人脸图是否为同一人，有正常分数时为[0~100]，推荐阈值80，超过即判断为同一人
dec_image	否	string	对SDK传入的加密图片进行解密。 当场景为APP时，此参数为解密后的人脸图片信息
risk_level	否	string	判断设备是否发生过风险行为来判断风险级别，取值（数值有高到低）： 1 – 高危 2 – 嫌疑 3 – 普通 4 – 正常
risk_tag	否	string	风险标签，若判断为有风险，则会有风险标签json 数组告知风险类型
risk_warn_code	否	int	风控服务异常，请重试若多次重试后无效，请提交工单咨询

返回示例

```
{
  "result": {
     "score": 83.04692078,
     "verify_status": 0
  },
  "log_id": 1381618138428211200,
  "dec_image": "/9j/4AAQSkZJRgABA...",
   "risk_level": "1",
  "log_id": 1349004812288524288,
  "risk_tag": [
     "general_inject"
  ]
}
```

参数说明

- **质量控制参数说明**

> 不同的控制度下所对应的质量控制阈值，如果检测出来的质量信息某一项不符合控制阈值的要求，则会返回错误信息。

控制度	left_eye	right_eye	nose	mouth	left_cheek	right_cheek	chin_contour	illumination	blurdegree	completeness
LOW	0.8	0.8	0.8	0.8	0.8	0.8	0.8	20	0.8	0
NORMAL	0.6	0.6	0.6	0.6	0.6	0.6	0.6	40	0.6	0
HIGH	0.3	0.3	0.2	0.2	0.3	0.3	0.3	50	0.2	1

活体控制参数说明

> 不同的控制度下所对应的活体控制阈值，如果检测出来的活体分数小于控制阈值，则会返回错误信息。

控制度	阈值	说明
LOW	0.05	活体误拒率：万分之一；拒绝率：97.75%
NORMAL（推荐）	0.3	活体误拒率：千分之一；拒绝率：98.82%
HIGH	0.9	活体误拒率：百分之一；拒绝率：99.77%

> 1、误拒率: 把**真人识别为假人**的概率. 阈值越高，安全性越高, 要求也就越高, 对应的误识率就越高
> 2、通过率=1-误拒率

关于以上数值的概念介绍：

拒绝率（TRR）：如99%，代表100次作弊假体攻击，会有99次被拒绝。 误拒率（FRR）：如0.5%，指1000次真人请求，会有5次因为活体分数低于阈值被错误拒绝。 通过率（TAR）：如99%，指100次真人请求，会有99次因为活体分数高于阈值而通过。 阈值（Threshold）：高于此数值，则可判断为活体。

合成图控制参数说明

> 不同的控制度下所对应的合成图检测（PS、人脸融合等）阈值，如果检测出来的分数大于控制阈值，则会返回错误信息。

控制度	阈值	误拒率（FRR）	通过率	攻击拒绝率（TRR）
LOW	0.00023	5%	95%	94.93%
NORMAL（推荐）	0.00048	1%	99%	89.71%
HIGH	0.00109	0.1%	99.9%	84.57%

> 1、误拒率: 把**正常图片识别为合成图片**的概率. 阈值越低，安全性越高, 要求也就越高, 对应的误识率就越高
> 2、通过率=1-误拒率

关于以上数值的概念介绍：

阈值（Threshold）：高于此数值，则可判断为是合成图攻击。

错误码

请参考人脸识别错误码

金融级人脸实名认证

金融级人脸实名认证接口

能力介绍

> 本接口需要配合金融级人脸采集SDK使用，金融级人脸采集SDK输出的图片会进行加密，在此接口进行解密。以此防止信息传输过程中泄露或遭到第三方非法攻击。
>
> 配合金融级采集SDK可以对炫瞳活体输出的图片进一步进行增强配合认证，大大提升了整体方案的非真人的拦截能力。

业务能力

- **质量检测（可选）**：判断图片中是否包含人脸，以及人脸在姿态、遮挡、模糊、光照等方面是否符合识别条件。
- **活体检测（可选）**：基于图片中的破绽分析，判断其中的人脸是否为**二次翻拍**（举例：如用户A用手机拍摄了一张包含人脸的图片一，用户B翻拍了图片一得到了图片二，并用图片二伪造成用户A去进行识别操作，这种情况普遍发生在金融开户、实名认证等环节）。
- **人脸实名认证（必选）**：基于姓名和身份证号，调取公安权威数据源人脸图，将当前获取的人脸图片，与公安数据源人脸图进行对比，得出比对分数，并基于此进行业务判断是否为同一人。由于公安数据源人脸图具有最权威的身份证明作用，故对用户本人的验证结果可信度也最为合理。

业务逻辑

- 上述三项能力为顺序串行验证，人脸实名认证能力为必选能力，质量检测和活体检测为可选能力，如这两项可选能力都选择，则验证顺序为`人脸质量检测->活体检测->人脸实名认证`。您也可以根据业务场景，选择这两项中的某一项或都不选择。
- **如选择了质量检测和活体检测能力，则有任意一个条件不通过，整个请求流程终止，并返回错误码，描述具体的不符合信息**。
- 基于此顺序串行验证逻辑，可以避免大量不符合条件的请求流转到人脸实名认证，节约您的成本。

推荐阈值

- 此接口使用的对比算法，针对带水纹证件照采用了专项的模型处理，可保证水纹信息的影响降到尽可能低。
- 如比对成功，最终返回的有效数据为一个**对比分值**，在0~1之间，您可以设定具体的阈值来判断是否验证通过。
- **人证相似度的推荐阈值为0.8，对应的误识率为万分之一**。

计费逻辑

- 前两个环节图片不符合校验规则，会以`error_code`形式反馈，属于正向业务判断，这两个环节的请求全部免费。真正请求到人脸实名认证这步并返回结果，才会进行计费。计费标准请参考价格文档

请求说明

注意事项：

- **请求体格式化**：Content-Type为`application/json`，通过`json`格式化请求体。
- **Base64编码**：请求的图片需经过`Base64编码`，图片的base64编码指将图片数据编码成一串字符串，使用该字符串代替图像地址。您可以首先得到图片的二进制，然后用Base64格式编码即可。需要注意的是，图片的base64编码是不包含图片头的，如``
- **图片格式**：现支持PNG、JPG、JPEG、BMP，**不支持GIF图片**

APP场景

适合于金融级采集SDK搭配请求公安数据源，验证用户的姓名、身份证号与现场采集的人脸图片是否一致的场景使用。

强调：

（1）金融级采集SDK使用时，需要保证image必须是加密过的，否则会报错图片解密失败

（2）金融级采集SDK与接口获取token的AKSK需要来自同一应用，否则也会导致图片解密失败

请求示例

HTTP方法：`POST`

请求URL： https://aip.baidubce.com/rest/2.0/face/v3/person/verify_fin

URL参数：

参数	值
access_token	通过API Key和Secret Key获取的access_token,参考“Access Token获取”

Header：

参数	值
Content-Type	application/json

Body中放置请求参数，参数详情如下：

请求参数

参数	必选	类型	说明
image	是	string	图片信息(总数据大小应小于10M)，图片上传方式根据image_type来判断
image_type	是	string	图片类型 BASE64:图片的base64值，base64编码后的图片数据，编码后的图片大小不超过2M；图片尺寸不超过1920*1080
app	是	string	ios android
name	是	string	姓名（注：需要是UTF-8编码的中文）
id_card_number	是	string	身份证号码
quality_control	否	string	图片质量控制 NONE: 不进行控制 LOW:较低的质量要求 NORMAL: 一般的质量要求 HIGH: 较高的质量要求 默认 NONE
liveness_control	否	string	活体检测控制 NONE: 不进行控制 LOW:较低的活体要求(高通过率 低攻击拒绝率) NORMAL: 一般的活体要求(平衡的攻击拒绝率, 通过率) HIGH: 较高的活体要求(高攻击拒绝率 低通过率) 默认NONE
spoofing_control	否	string	合成图控制参数 **NONE**: 不进行控制 **LOW**:较低的合成图阈值数值，由于合成图判定逻辑为大于阈值视为合成图攻击，该项代表**低通过率、高攻击拒绝率** **NORMAL**: 一般的合成图阈值数值，由于合成图判定逻辑为大于阈值视为合成图攻击，该项代表**平衡的攻击拒绝率, 通过率** **HIGH**: 较高的合成图阈值数值，由于合成图判定逻辑为大于阈值视为合成图攻击，该项代表**高通过率、低攻击拒绝率**) **默认为NONE**
risk_identify	否	boolean	Ture：打开大数据风控功能。（接受SDK端传入的指纹信息，返回识别结果） False：关闭大数据风控功能。 默认False
zid	是	string	SDK 唯一标识 ZID，从增强级SDK中获取；必选，端内活动 zid 为空会直接拒绝，请重视此参数
ip	否	string	需要风控判别的ip地址
phone	否	string	需要风控判别的手机号

请求示例

APP场景:图片上传比对使用json格式，如下:

```
{
  "image":"/9j/4AAQSkZJRg...", //金融级采集SDK上传的加密后的图片
  "image_type":"BASE64",
  "id_card_number":"110284********5828",
  "name":"张三",

 "risk_identify":"Ture",
 "ip": "...ip", //填写IP地址
 "phone": "...phone"  //填写手机号
}
```

- **返回参数**

参数	必须	类型	说明
log_id	是	uint64	日志id
result	是	jsonObject	认证返回的结果
verify_status	是	int	认证状态，取值： 0 姓名与身份证匹配，可进行人脸比对 1 身份证号与姓名不匹配或该身份证号不存在 2 公安网图片不存在或质量过低
score	否	float	与公安数据源人脸图相似度可能性，用于验证生活照与公安数据源人脸图是否为同一人，有正常分数时为[0~100]，推荐阈值80，超过即判断为同一人
dec_image	否	string	对SDK传入的加密图片进行解密。 当场景为APP时，此参数为解密后的人脸图片信息
risk_level	否	string	判断设备是否发生过风险行为来判断风险级别，取值（数值由高到低）： 1 – 高危 2 – 嫌疑 3 – 普通 4 – 正常
risk_tag	否	string	风险标签，若判断为有风险，则会有风险标签json 数组告知风险类型 例如：general_inject
risk_warn_code	否	int	风控返回参数，只有在 risk_identify 为 true 时才返回

只有在风控服务异常的时候才返回这个字段 |

返回示例

```
{
  "log_id": 1370579072568000512,
  "result": {
     "score": 40.884,
     "verify_status": 0
  },
  "dec_image": "/9j/4AAQSkZJRgABAgAAAQABAAD",
  "risk_level": "3",
  "risk_tag": [
     "general_inject"
  ]
}
```

H5场景 如您需在微信公众号等H5场景中调用此接口，请接入百度金融级人脸实名认证方案，接入方式可参考接入文档。

参数说明

- **质量控制参数说明**

> 不同的控制度下所对应的质量控制阈值，如果检测出来的质量信息某一项不符合控制阈值的要求，则会返回错误信息。

控制度	left_eye	right_eye	nose	mouth	left_cheek	right_cheek	chin_contour	illumination	blurdegree	completeness
LOW	0.8	0.8	0.8	0.8	0.8	0.8	0.8	20	0.8	0
NORMAL	0.6	0.6	0.6	0.6	0.6	0.6	0.6	40	0.6	0
HIGH	0.3	0.3	0.2	0.2	0.3	0.3	0.3	50	0.2	1

活体控制参数说明

> 不同的控制度下所对应的活体控制阈值，如果检测出来的活体分数小于控制阈值，则会返回错误信息。

控制度	阈值	说明
LOW	0.05	活体误拒率：万分之一；拒绝率：97.75%
NORMAL（推荐）	0.3	活体误拒率：千分之一；拒绝率：98.82%
HIGH	0.9	活体误拒率：百分之一；拒绝率：99.77%

> 1、误拒率: 把**真人识别为假人**的概率. 阈值越高，安全性越高, 要求也就越高, 对应的误识率就越高
> 2、通过率=1-误拒率

关于以上数值的概念介绍：

拒绝率（TRR）：如99%，代表100次作弊假体攻击，会有99次被拒绝。 误拒率（FRR）：如0.5%，指1000次真人请求，会有5次因为活体分数低于阈值被错误拒绝。 通过率（TAR）：如99%，指100次真人请求，会有99次因为活体分数高于阈值而通过。 阈值（Threshold）：高于此数值，则可判断为活体。

合成图控制参数说明

> 不同的控制度下所对应的合成图检测（PS、人脸融合等）阈值，如果检测出来的分数大于控制阈值，则会返回错误信息。

控制度	阈值	误拒率（FRR）	通过率	攻击拒绝率（TRR））
LOW	0.00023	5%	95%	94.93%
NORMAL（推荐）	0.00048	1%	99%	89.71%
HIGH	0.00109	0.1%	99.9%	84.57%

> 1、误拒率: 把**正常图片识别为合成图片**的概率. 阈值越低，安全性越高, 要求也就越高, 对应的误识率就越高
> 2、通过率=1-误拒率

关于以上数值的概念介绍：

阈值（Threshold）：高于此数值，则可判断为是合成图攻击。

错误码

请参考错误码说明文档。

增强级人脸比对

增强级人脸比对接口

能力介绍

接口能力

- **两张人脸图片相似度对比**：比对两张图片中人脸的相似度，并返回相似度分值；
- **多种图片类型**：支持**生活照**、**证件照**、**身份证芯片照**、**带网纹照**四种类型的人脸对比；
- **活体检测控制**：基于图片中的破绽分析，判断其中的人脸是否为**二次翻拍**（举例：如用户A用手机拍摄了一张包含人脸的图片一，用户B翻拍了图片一得到了图片二，并用图片二伪造成用户A去进行识别操作，这种情况普遍发生在金融开户、实名认证等环节。）；
- **质量检测控制**：分析图片的中人脸的模糊度、角度、光照强度等特征，判断图片质量；
- **图片加密及风控：**配合增强级采集SDK版本使用（SDK下载请点击这里）
  - 对采集SDK输出的加密图片进行解密（加密传输可以有效避免第三方非法黑产**绕过APP模拟请求攻击云端接口的行为**，Eg：脚本攻击等）；
  - 以及结合百度安全实验室大数据风控能力，对采集SDK的发起端设备进行风控识别，辨别是否为**风险设备**，Eg：ROM注入、视频劫持等。

业务应用

- 用于比对多张图片中的人脸相似度并返回两两比对的得分，可用于判断两张脸是否是同一人的可能性大小。
- 典型应用场景：如**人证合一验证**，**用户认证**等，可与您现有的人脸库进行比对验证。

调用方式

请求URL数据格式

向API服务地址使用POST发送请求，必须在URL中带上参数`access_token`，可通过后台的API Key和Secret Key生成，具体方式请参考“Access Token获取”。

示例代码

Bash
PHP
JAVA
Python
Cpp
C#
Node

```
##### !/bin/bash
curl -i -k 'https://aip.baidubce.com/oauth/2.0/token?grant_type=client_credentials&client_id=【百度云应用的AK】&client_secret=【百度云应用的SK】'
```

> **注意**：access_token的有效期为30天，**切记需要每30天进行定期更换，或者每次请求都拉取新token**；

例如此接口，使用HTTPS POST发送：

```
https://aip.baidubce.com/rest/2.0/face/v1/merge?access_token=24.f9ba9c5341b67688ab4added8bc91dec.2592000.1485570332.282335-8574074
```

POST中Body的参数，按照下方请求参数说明选择即可。

> **提示**：如果您为百度云老用户，正在使用其他非AI的服务，可以参考百度云AKSK鉴权方式发送请求，虽然**请求方式**和**鉴权方法**和本文所介绍的不同，但请求参数和返回结果一致。

请求说明

注意事项：

- **请求体格式化**：Content-Type为application/json，通过json格式化请求体。
- **Base64编码**：请求的图片需经过Base64编码，图片的base64编码指将图片数据编码成一串字符串，使用该字符串代替图像地址。您可以首先得到图片的二进制，然后用Base64格式编码即可。需要注意的是，图片的base64编码是不包含图片头的，如
- **图片格式**：现支持PNG、JPG、JPEG、BMP，**不支持GIF图片**
- **人脸识别接口分为V2和V3两个版本，本文档为V3版本接口的说明文档，请确认您在百度云后台获得的是V3版本接口权限，再来阅读本文档。**

> 辨别接口版本的方法是：在百度云后台进入【应用列表】，点击【应用名称】，在【API列表】中可以看到【请求地址】，若请求地址中带有【v3】标识，则您具有的是v3权限，可以阅读本文档；若请求地址中带有【v2】标识，则您具有的是v2权限，应该去阅读v2文档。

APP场景

适合于增强级采集SDK与留存图片比对的场景使用。

强调：

（1）增强级采集SDK使用时，需要保证image必须是加密过的，否则会报错图片解密失败

（2）增强级采集SDK与接口获取token的AKSK需要来自同一应用，否则也会导致图片解密失败

请求示例

HTTP方法：`POST`

请求URL： `https://aip.baidubce.com/rest/2.0/face/v3/matchsec`

URL参数：

参数	值
access_token	通过API Key和Secret Key获取的access_token,参考“Access Token获取”

Header如下：

参数	值
Content-Type	application/json

Body中放置请求参数，参数详情如下：

请求参数

参数	必选	类型	说明
scene_type	否	string	APP ： APP场景，与采集SDK一起使用
image	是	string	采集SDK4.1版本上传的加密图片Base64信息
image_type	是	string	图片类型 **BASE64**:图片的base64值，base64编码后的图片数据，编码后的图片大小不超过2M；
app	是	string	ios android
face_type	否	string	人脸的类型 **LIVE**：表示生活照：通常为手机、相机拍摄的人像图片、或从网络获取的人像图片等， **IDCARD**：表示身份证芯片照：二代身份证内置芯片中的人像照片， **WATERMARK**：表示带水印证件照：一般为带水印的小图，如公安网小图 **CERT**：表示证件照片：如拍摄的身份证、工卡、护照、学生证等证件图片 **INFRARED** 表示红外照片：使用红外相机拍摄的照片 默认LIVE
quality_control	否	string	图片质量控制 **NONE**: 不进行控制 **LOW**:较低的质量要求 **NORMAL**: 一般的质量要求 **HIGH**: 较高的质量要求 **默认 NONE** 若图片质量不满足要求，则返回结果中会提示质量检测失败
liveness_control	否	string	活体检测控制 **NONE**: 不进行控制 **LOW**:较低的活体要求(高通过率 低攻击拒绝率) **NORMAL**: 一般的活体要求(平衡的攻击拒绝率, 通过率) **HIGH**: 较高的活体要求(高攻击拒绝率 低通过率) **默认 NONE** 若活体检测结果不满足要求，则返回结果中会提示活体检测失败
register_image	是	string	图片信息(**总数据大小应小于10M**)，图片上传方式根据image_type来判断。本图片特指客户服务器上传图片，非加密图片Base64值

			服务器上传图片，非加密图片Base64值
register_image_type	是	string	图片类型 **BASE64**:图片的base64值，base64编码后的图片数据，编码后的图片大小不超过2M； **URL**:图片的 URL地址(可能由于网络等原因导致下载图片时间过长)； **FACE_TOKEN**: 人脸图片的唯一标识，调用人脸检测接口时，会为每个人脸图片赋予一个唯一的FACE_TOKEN，同一张图片多次检测得到的FACE_TOKEN是同一个。
register_face_type	否	string	人脸的类型 **LIVE**：表示生活照：通常为手机、相机拍摄的人像图片、或从网络获取的人像图片等， **IDCARD**：表示身份证芯片照：二代身份证内置芯片中的人像照片， **WATERMARK**：表示带水印证件照：一般为带水印的小图，如公安网小图 **CERT**：表示证件照片：如拍摄的身份证、工卡、护照、学生证等证件图片 **INFRARED** 表示红外照片：使用红外相机拍摄的照片 默认LIVE
register_quality_control	否	string	图片质量控制 **NONE**: 不进行控制 **LOW**:较低的质量要求 **NORMAL**: 一般的质量要求 **HIGH**: 较高的质量要求 **默认 NONE** 若图片质量不满足要求，则返回结果中会提示质量检测失败
register_liveness_control	否	string	活体检测控制 **NONE**: 不进行控制 **LOW**:较低的活体要求(高通过率 低攻击拒绝率) **NORMAL**: 一般的活体要求(平衡的攻击拒绝率, 通过率) **HIGH**: 较高的活体要求(高攻击拒绝率 低通过率) **默认 NONE** 若活体检测结果不满足要求，则返回结果中会提示活体检测失败
risk_identify	否	boolean	Ture：打开大数据风控功能。（接受SDK端传入的指纹信息，返回识别结果） False：关闭大数据风控功能。 默认False
ip	否	string	需要风控判别的ip地址
phone	否	string	需要风控判别的手机号

请求的示例代码如下所示：

```
{
  "scene_type":"APP",

 "image": "/9j/4AAQSk... ...",  //采集SDK传入的加密后的图片BASE64信息
 "image_type": "BASE64",
 "face_type": "LIVE",
 "quality_control": "NORMAL",
 "liveness_control": "NORMAL",

 "register_image": "/9j/4AAQSk... ...", //服务端传入的非加密图片的BASE64
 "register_image_type": "BASE64",
 "register_face_type": "LIVE",
 "register_quality_control": "NORMAL",
 "register_liveness_control": "NORMAL",

 "risk_identify": "Ture",    //仅当APP场景时生效
 "ip": "...ip", //填写IP地址
 "phone": "...phone"   //填写手机号
}
```

返回参数

参数名	必选	类型	说明
score	是	float	人脸相似度得分，推荐阈值80分
face_list	是	array	人脸信息列表
+face_token	是	string	人脸的唯一标志
dec_image	否	string	对SDK传入的加密图片进行解密。 当场景为APP时，此参数为解密后的人脸图片信息
risk_level	否	string	判断设备是否发生过风险行为来判断风险级别，取值（数值有高到低）： 1 – 高危 2 – 嫌疑 3 – 普通 4 – 正常
risk_tag	否	string	风险标签，若判断为有风险，则会有风险标签json 数组告知风险类型
risk_warn_code	否	int	风控服务异常，请重试若多次重试后无效，请提交工单咨询

返回示例代码：

- 返回示例


```
{
  "log_id": 1381603218697486336,
  "result": {
     "score": 0,
     "face_list": [
        {
           "face_token": "1670763fa23353d0c6e8f9c9e7e85f34"
        },
        {
           "face_token": "05f934809b7a96c2ddc53462f6e7ad42"
        }
     ],
  },
  "dec_image": "/9j/4AAQSkZJRgABA...",
  "risk_level": "1",
  "log_id": 1349004812288524288,
  "risk_tag": [
     "general_inject"
  ]
}
```


错误码

请参考人脸识别错误码

金融级人脸比对

金融级人脸比对

能力介绍

接口能力

- **两张人脸图片相似度对比**：比对两张图片中人脸的相似度，并返回相似度分值；
- **多种图片类型**：支持**生活照**、**证件照**、**身份证芯片照**、**带网纹照**四种类型的人脸对比；
- **活体检测控制**：基于图片中的破绽分析，判断其中的人脸是否为**二次翻拍**（举例：如用户A用手机拍摄了一张包含人脸的图

片一，用户B翻拍了图片一得到了图片二，并用图片二伪造成用户A去进行识别操作，这种情况普遍发生在金融开户、实名认证等环节。）；

- **质量检测控制**：分析图片的中人脸的模糊度、角度、光照强度等特征，判断图片质量；
- **图片加密及风控：**配合金融级采集SDK版本使用（SDK请与您的客户经理申请）
  - 对采集SDK输出的加密图片进行解密（加密传输可以有效避免第三方非法黑产**绕过APP模拟请求攻击云端接口的行为**，Eg：脚本攻击等）；
  - 以及结合百度安全实验室大数据风控能力，对采集SDK的发起端设备进行风控识别，辨别是否为**风险设备**，Eg：ROM注入、视频劫持等；
  - 搭配金融级SDK进行炫瞳活体的认证，提升非活体的拦截能力。

业务应用

- 用于比对多张图片中的人脸相似度并返回两两比对的得分，可用于判断两张脸是否是同一人的可能性大小。
- 典型应用场景：如**人证合一验证**，**用户认证**等，可与您现有的人脸库进行比对验证。

调用方式

请求URL数据格式

向API服务地址使用POST发送请求，必须在URL中带上参数`access_token`，可通过后台的API Key和Secret Key生成，具体方式请参考“Access Token获取”。

示例代码

Bash
PHP
JAVA
Python
Cpp
C#
Node

```
##### !/bin/bash
curl -i -k 'https://aip.baidubce.com/oauth/2.0/token?grant_type=client_credentials&client_id=【百度云应用的AK】&client_secret=【百度云应用的SK】'
```

> **注意**：access_token的有效期为30天，**切记需要每30天进行定期更换，或者每次请求都拉取新token**；

例如此接口，使用HTTPS POST发送：

```
https://aip.baidubce.com/rest/2.0/face/v1/merge?access_token=24.f9ba9c5341b67688ab4added8bc91dec.2592000.1485570332.282335-8574074
```

POST中Body的参数，按照下方请求参数说明选择即可。

> **提示**：如果您为百度云老用户，正在使用其他非AI的服务，可以参考百度云AKSK鉴权方式发送请求，虽然**请求方式**和**鉴权方法**和本文所介绍的不同，但请求参数和返回结果一致。

请求说明

注意事项：

- **请求体格式化**：Content-Type为application/json，通过json格式化请求体。
- **Base64编码**：请求的图片需经过Base64编码，图片的base64编码指将图片数据编码成一串字符串，使用该字符串代替图像地址。您可以首先得到图片的二进制，然后用Base64格式编码即可。需要注意的是，图片的base64编码是不包含图片头的，如
- **图片格式**：现支持PNG、JPG、JPEG、BMP，**不支持GIF图片**
- **人脸识别接口分为V2和V3两个版本，本文档为V3版本接口的说明文档，请确认您在百度云后台获得的是V3版本接口权限，再来阅读本文档。**

> 辨别接口版本的方法是：在百度云后台进入【应用列表】，点击【应用名称】，在【API列表】中可以看到【请求地址】，若请求地址中带有【v3】标识，则您具有的是v3权限，可以阅读本文档；若请求地址中带有【v2】标识，则您具有的是v2权限，应该去阅读v2文档。

APP场景使用

适合于增强级采集SDK与留存图片比对的场景使用。

强调：

（1）增强级采集SDK使用时，需要保证image必须是加密过的，否则会报错图片解密失败

（2）增强级采集SDK与接口获取token的AKSK需要来自同一应用，否则也会导致图片解密失败

请求示例

HTTP方法：`POST`

请求URL： https://aip.baidubce.com/rest/2.0/face/v3/match_fin

URL参数：

参数	值
access_token	通过API Key和Secret Key获取的access_token,参考“Access Token获取”

Header如下：

参数	值
Content-Type	application/json

Body中放置请求参数，参数详情如下：

请求参数

参数	必选	类型	说明
image	是	string	采集SDK4.1版本上传的加密图片Base64信息
image_type	是	string	图片类型 **BASE64**:图片的base64值，base64编码后的图片数据，编码后的图片大小不超过2M；
app	是	string	ios android
face_type	否	string	人脸的类型 **LIVE**：表示生活照：通常为手机、相机拍摄的人像图片、或从网络获取的人像图片等， **IDCARD**：表示身份证芯片照：二代身份证内置芯片中的人像照片， **WATERMARK**：表示带水印证件照：一般为带水印的小图，如公安网小图 **CERT**：表示证件照片：如拍摄的身份证、工卡、护照、学生证等证件图片 **INFRARED** 表示红外照片：使用红外相机拍摄的照片 默认LIVE
quality_control	否	string	图片质量控制 **NONE**: 不进行控制 **LOW**:较低的质量要求 **NORMAL**: 一般的质量要求 **HIGH**: 较高的质量要求 **默认 NONE** 若图片质量不满足要求，则返回结果中会提示质量检测失败
liveness_control	否	string	活体检测控制 **NONE**: 不进行控制 **LOW**:较低的活体要求(高通过率 低攻击拒绝率) **NORMAL**: 一般的活体要求(平衡的攻击拒绝率, 通过率) **HIGH**: 较高的活体要求(高攻击拒绝率 低通过率) **默认 NONE** 若活体检测结果不满足要求，则返回结果中会提示活体检测失败
register_image	是	string	图片信息(**总数据大小应小于10M**)，图片上传方式根据image_type来判断。本图片特指客户服务器上传图片，非加密图片Base64值
			图片类型

register_image_type	是	string	图片类型 **BASE64**:图片的base64值，base64编码后的图片数据，编码后的图片大小不超过2M； **URL**:图片的 URL地址(可能由于网络等原因导致下载图片时间过长)； **FACE_TOKEN**: 人脸图片的唯一标识，调用人脸检测接口时，会为每个人脸图片赋予一个唯一的FACE_TOKEN，同一张图片多次检测得到的FACE_TOKEN是同一个。
register_face_type	否	string	人脸的类型 **LIVE**：表示生活照：通常为手机、相机拍摄的人像图片、或从网络获取的人像图片等， **IDCARD**：表示身份证芯片照：二代身份证内置芯片中的人像照片， **WATERMARK**：表示带水印证件照：一般为带水印的小图，如公安网小图 **CERT**：表示证件照片：如拍摄的身份证、工卡、护照、学生证等证件图片 **INFRARED** 表示红外照片：使用红外相机拍摄的照片 默认LIVE
register_quality_control	否	string	图片质量控制 **NONE**: 不进行控制 **LOW**:较低的质量要求 **NORMAL**: 一般的质量要求 **HIGH**: 较高的质量要求 **默认 NONE** 若图片质量不满足要求，则返回结果中会提示质量检测失败
register_liveness_control	否	string	活体检测控制 **NONE**: 不进行控制 **LOW**:较低的活体要求(高通过率 低攻击拒绝率) **NORMAL**: 一般的活体要求(平衡的攻击拒绝率, 通过率) **HIGH**: 较高的活体要求(高攻击拒绝率 低通过率) **默认 NONE** 若活体检测结果不满足要求，则返回结果中会提示活体检测失败
risk_identify	否	boolean	Ture：打开大数据风控功能。（接受SDK端传入的指纹信息，返回识别结果） False：关闭大数据风控功能。 默认False
ip	否	string	需要风控判别的ip地址
phone	否	string	需要风控判别的手机号

请求的示例代码如下所示：

```
{
    "scene_type":"APP",

 "image": "/9j/4AAQSk... ...",  //采集SDK传入的加密后的图片BASE64信息
 "image_type": "BASE64",
 "face_type": "LIVE",
 "quality_control": "NORMAL",
 "liveness_control": "NORMAL",

 "register_image": "/9j/4AAQSk... ...", //服务端传入的非加密图片的BASE64
 "register_image_type": "BASE64",
 "register_face_type": "LIVE",
 "register_quality_control": "NORMAL",
 "register_liveness_control": "NORMAL",

 "risk_identify": "Ture",
 "ip": "...ip", //填写IP地址
 "phone": "...phone"   //填写手机号
}
```

返回参数

参数名	必选	类型	说明
score	是	float	人脸相似度得分，推荐阈值80分
face_list	是	array	人脸信息列表
+face_token	是	string	人脸的唯一标志
dec_image	否	string	对SDK传入的加密图片进行解密。 当场景为APP时，此参数为解密后的人脸图片信息
risk_level	否	string	判断设备是否发生过风险行为来判断风险级别，取值（数值有高到低）： 1 – 高危 2 – 嫌疑 3 – 普通 4 – 正常
risk_tag	否	string	风险标签，若判断为有风险，则会有风险标签jason 数组告知风险类型
risk_warn_code	否	int	风控服务异常，请重试若多次重试后无效，请提交工单咨询

返回示例代码：

- 返回示例


```
{
  "log_id": 1381603218697486336,
  "result": {
     "score": 0,
     "face_list": [
        {
           "face_token": "1670763fa23353d0c6e8f9c9e7e85f34"
        },
        {
           "face_token": "05f934809b7a96c2ddc53462f6e7ad42"
        }
     ],
  },
  "dec_image": "/9j/4AAQSkZJRgABA...",
  "risk_level": "1",
  "log_id": 1349004812288524288,
  "risk_tag": [
     "general_inject"
  ]
}
```


参数说明

- **质量控制参数说明**

> 不同的控制度下所对应的质量控制阈值，如果检测出来的质量信息某一项不符合控制阈值的要求，则会返回错误信息。

遮挡情况的阈值

控制度	left_eye	right_eye	nose	mouth	left_cheek	right_cheek	chin_contour
LOW	0.8	0.8	0.8	0.8	0.8	0.8	0.8
NORMAL	0.6	0.6	0.6	0.6	0.6	0.6	0.6
HIGH	0.2	0.2	0.2	0.2	0.2	0.2	0.2

光照、模糊度、完整度的阈值

控制度	illumination	blurdegree	completeness
LOW	20	0.8	0
NORMAL	40	0.6	0
HIGH	100	0.2	1

活体控制参数说明

> 不同的控制度下所对应的活体控制阈值，如果检测出来的活体分数小于控制阈值，则会返回错误信息。

控制度	阈值	说明
LOW	0.05	活体误拒率：万分之一；拒绝率：97.75%
NORMAL	0.3	活体误拒率：千分之一；拒绝率：98.82%
HIGH	0.9	活体误拒率：百分之一；拒绝率：99.77%

> 1、误拒率: 把**真人识别为假人**的概率. 阈值越高，安全性越高, 要求也就越高, 对应的误识率就越高
> 2、通过率=1-误拒率

关于以上数值的概念介绍：

拒绝率（TRR）：如99%，代表100次作弊假体攻击，会有99次被拒绝。 误拒率（FRR）：如0.5%，指1000次真人请求，会有5次因为活体分数低于阈值被错误拒绝。 通过率（TAR）：如99%，指100次真人请求，会有99次因为活体分数高于阈值而通过。 阈值（Threshold）：高于此数值，则可判断为活体。

错误码

请参考人脸识别错误码

活体检测

如果您对文档内容有任何疑问，可以通过以下几种方式联系我们：

- 在百度云控制台内 提交工单，咨询问题类型请选择**人工智能服务**；
- 如有需要讨论的疑问，欢迎进入 AI社区 与其他开发者们一同交流。

一、视频活体检测

能力介绍

业务能力

视频活体检测产品，是由两个接口组合而成，主要用于在H5场景下，通过用户新录制并上传一个视频，来进行活体检测的判断。相对于APP有`动作校验+静默图片活体`、`静默图片活体`这两种方式，视频活体检测方案比APP方案更加灵活，同时比单张图片活体方式更加安全。其主要功能如下所示：

- **视频多帧活体检测**：录制并上传的视频，会在云端进行随机抽帧分析，并得出最终的活体检测分数。
- **随机校验码**：（用于在语音/动作活体检测中生成随机数字/动作）
  - 为防止用户提交非当前操作的视频，选择随机验证码后，即可在录制视频时，随机生成数字/动作，用户需要读出数字/做出相应动作，在后续识别时校验，以判断视频是否为现场录制。

- 随机校验码作为辅助性质的验证条件，是一个可选项，可根据业务具体应用场景来选择是否使用，以及**根据业务场景决定选择语音/动作活体检测方式**。如业务场景比较嘈杂或方言口音比较重，可不使用语音活体检测方式，选择动作活体检测方式进行校验。

- **唇语识别Beta版**：
  - 对用户上传的视频进行唇语识别，返回唇语识别是否通过，返回结果为单独字段，与视频活体与语音/动作校验不冲突。
  - 若需要使用唇语识别，需要先使用随机校验码和视频活体检测接口，且活体检测方式应配置为语音活体检测。
  - **唇语识别能力当前为Beta版本，识别准确率较低，仅用于辅助 语音活体检测方式 进行验证，您可以通过接口的入参来设置是否使用该能力**
- **合成图识别Beta版**：
  - 对用户上传的视频抽帧进行合成图像识别，能识别出AI变脸、AI换脸等合成图，让业务更加安全。
  - **合成图识别能力当前为Beta版本，仅用于辅助验证，您可以通过接口的入参来设置是否使用该能力**

> 以上四项能力，分为两个接口，使用顺序为`随机校验码接口`->`视频活体检测接口`，具体调用逻辑可以参考文档 接口文档

主要适用场景

- 微信服务号：用于对操作用户真实性要求严格的场景，用于依托于微信服务号的金融开户、实名认证、账户信息变更二次验证等服务。
- APP内Webview：对于如Cordova架构开发的APP，或者APP内变更频繁的身份信息页等情况，可以采用此方案完成活体检测。
- 浏览器：如果仅是一个H5宣传页，或者Wap版本网页等，可以通过此方法快速集成更加安全的活体检测功能。

此方案的优劣势

- 优势：相对于APP有动作校验、单张图片静默判断，此方法在没有APP情况下，可以更快速、轻量级地实现活体检测，同时保障一定安全性。
- 劣势：由于视频较大，上传时间可能较长，另由于不同手机的浏览器内核差异较大，容易出现兼容性问题。

调用方式

请求URL数据格式

向API服务地址使用POST发送请求，必须在URL中带上参数`access_token`，可通过后台的API Key和Secret Key生成，具体方式请参考“Access Token获取”。

示例代码

Bash
PHP
JAVA
Python
Cpp
C#

Node

```
##### !/bin/bash
curl -i -k 'https://aip.baidubce.com/oauth/2.0/token?grant_type=client_credentials&client_id=【百度云应用的AK】&client_secret=【百度云应用的SK】'
```

> **注意**：access_token的有效期为30天，**切记需要每30天进行定期更换，或者每次请求都拉取新token**；

例如此接口，使用HTTPS POST发送：

```
https://aip.baidubce.com/rest/2.0/face/v1/merge?access_token=24.f9ba9c5341b67688ab4added8bc91dec.2592000.1485570332.282335-8574074
```

POST中Body的参数，按照下方请求参数说明选择即可。

> **提示**：如果您为百度云老用户，正在使用其他非AI的服务，可以参考百度云AKSK鉴权方式发送请求，虽然**请求方式**和**鉴权方法**和本文所介绍的不同，但请求参数和返回结果一致。

1.1 随机校验码接口（原语音验证码接口）

接口描述

此接口主要用于生成随机码，用于视频的语音/动作识别校验使用，以判断视频的即时性，而非事先录制的，提升作弊的难度。

在线调试 您可以在 API Explorer 中调试该接口，可进行签名验证、查看在线调用的请求内容和返回结果、示例代码的自动生成。

请求说明

请求示例

HTTP方法：POST

请求URL： https://aip.baidubce.com/rest/2.0/face/v1/faceliveness/sessioncode

URL参数：

参数	值
access_token	通过API Key和Secret Key获取的access_token,参考“Access Token获取”

Header：

参数	值
Content-Type	application/x-www-form-urlencoded

Body中放置请求参数，参数详情如下：

- 请求参数

参数名	必选	类型	说明
type	否	int	0为语音验证和唇语验证步骤， 1为视频动作活体 默认0
min_code_length	否	int	当type=0时，语音和唇语生成的验证码最小长度：最大6 最小3 默认3 当type=1时，视频动作活体 的验证码最小长度：最大3 最小1 默认1
max_code_length	否	int	当type=0时，语音和唇语生成的验证码最大长度：最大6 最小3 默认6 当type=1时，视频动作活体 的验证码最大长度：最大3 最小1 默认3

> **说明**：
>
> - 当传参为1-1-3时，代表随机生成1-3个动作进行核验；
> - 当传参为1-1-1时，代表随机生成1个动作进行核验；
> - 当传参为1-2-2时，代表随机生成2个动作进行核验；
> - 当传参为1-3-3时，代表随机生成3个动作进行核验。

返回说明

返回参数

字段	必选	类型	说明
session_id	是	string	随机校验码会话id，有效期5分钟，请提示用户在五分钟内完成全部操作 验证码使用过即失效，每次使用视频活体前请重新拉取验证码
code	是	string	随机验证码，数字形式，1~6位数字； 若为动作活体时，返回数字表示的动作对应关系为：0:眨眼 2:右转 3:左转 4:抬头 5:低头

返回示例

```
 {
  "err_no": 0,
  "err_msg": "SUCCESS",
  "result": {
      "session_id": "S59faeeebb9111890355690", //会话ID
      "code": "034" //当为视频动作活体时，返回值的代表所需动作和动作顺序。 0:眨眼  2:右转 3:左转  4:抬头  5:低头
  },
  "timestamp": 1509617387,
  "cached": 0,
  "serverlogid": "0587756642",
  "error_code": 0,
   "error_message": "SUCCESS"
}
```

> error_code为0即代表成功, 其他情况则为失败

1.2 视频活体检测接口

接口描述

此接口一方面通过随机验证码接口获取语音/动作校验码，通过session code来判断视频是否作弊。另一方面进行视频抽帧，判断是否为活体。

在线调试 您可以在 API Explorer 中调试该接口，可进行签名验证、查看在线调用的请求内容和返回结果、示例代码的自动生成。

请求说明

请求示例

HTTP方法：`POST`

请求URL： `https://aip.baidubce.com/rest/2.0/face/v1/faceliveness/verify`

URL参数：

参数	值
access_token	通过API Key和Secret Key获取的access_token,参考“Access Token获取”

Header：

参数	值
Content-Type	application/x-www-form-urlencoded

Body中放置请求参数，参数详情如下：

请求参数

参数名	必选	类型	说明
type_identify	否	string	voice为语音验证，action为视频动作活体验证，默认为voice （若您需要静默视频活体验证，此参数无需传入）
video_base64	是	string	base64 编码的视频数据（编码前建议先将视频进行转码，h.264编码，mp4封装）需要注意的是，视频的base64编码是不包含视频头的，如 data:video/mp4;base64,； 建议视频长度控制在01s-10s之间，视频大小建议在2M以内（视频大小强制要求在20M以内，推荐使用等分辨率压缩，压缩分辨率建议不小于640*480） 视频大小分辨率建议限制在16~2032之间
session_id	否	string	会话ID （当此字段为空时，为静默视频活体检测） 当使用语音验证及视频动作活体验证时，此字段必须传入，且获取验证码时要填入对应的验证类型
lip_identify	否	string	辅助语音验证进行，用于判断验证码是否为当事人读出 取值COMMON/STRICT/OFF, COMMON代表使用唇语识别，STRICT代表使用唇语识别并使用更加严格的策略判断是否通过 OFF代表关闭唇语识别，默认OFF
face_field	否	string	需要使用合成图功能时，此项传入**spoofing**

> 唇语识别中，使用STRICT策略会比使用COMMON策略通过率降低，但攻击拒绝率会提升 建议视频大小控制在10M/1min以内

返回说明

返回参数

参数名	类型	说明
score	float	活体检测的总体打分 范围[0,1]，分数越高则活体的概率越大
maxspoofing	float	返回的1-8张图片中合成图检测得分的最大值 范围[0,1]，分数越高则概率越大
spoofing_score	float	返回的1-8张图片中合成图检测得分的中位数 范围[0,1]，分数越高则概率越大
thresholds	array	阈值 按活体检测分数>阈值来判定活体检测是否通过(阈值视产品需求选择其中一个)
code	array	验证码信息
+create	string	生成的验证码
+identify	string	验证码的语音识别结果
+similarity	float	验证码相似度 取值0~1 1代表完全一致 0代表完全不一致 推荐阈值**0.75**
lip_language	string	唇语识别结果 pass代表唇语验证通过，fail代表唇语验证未通过，当存在请求字段lip_identify字段值为COMMON 或 STRICT时返回
action_verify	string	动作识别结果 pass代表动作验证通过，fail代表动作验证未通过，当存在请求字段type_identify字段值为action时返回
best_image	array	质量最佳图片
+face_token	string	人脸图片的唯一标识
+pic	string	base64编码后的图片信息
+liveness_score	float	此图片的活体分数，范围[0,1]
pic_list	array	返回1-8张抽取出来的图片信息
+face_token	string	人脸图片的唯一标识
+spoofing	float	此图片的合成图分数，范围[0,1]

返回示例

```
 {
"err_no": 0,
"err_msg": "SUCCESS",
"result": {
    "score": 0.18,
    "maxspoofing": 0.0002082588035,
    "spoofing_score": 0.00018671568975,
    "code": {
        "create": "853",
        "identify": "23456789",
        "similarity": 0.13
    },
    "lip_language": "fail",     //lip_identify 为 COMMON 或 STRICT 时返回该字段值，值为pass 或 false
    "action_verify": "fail",    //type_identify 为 action 时返回该字段值，值为pass 或 false【注意：action_verify与lip_language是互斥的】
    "thresholds": {
         "frr_1e-4": 0.05,     //万分之一误拒率的阈值
         "frr_1e-3": 0.3,      //千分之一误拒率的阈值
         "frr_1e-2": 0.9,      //百分之一误拒率的阈值
    },
    "best_image": {
        "pic": "图片base64值",
        "face_token": "0839b921224816fb558b0a74ee6284fb",
        "face_id": "0839b921224816fb558b0a74ee6284fb",
```

```
        "liveness_score": 0.9634260269,
        "spoofing": 0.0001962436945
    },
    "pic_list": [ //默认返回8张图片
        {
            "pic" : "gAQTGF2YzU4LjkyLjEwMAD", //图片base64编码后的值
            "face_token": "f043b6c7d202cb25e8dfc24fccf37553",
            "face_id": "f043b6c7d202cb25e8dfc24fccf37553",
            "liveness_score": 0.18,
            "spoofing": 0.000179775554
        },
        {
            "pic" : "gAQTGF2YzU4LjkyLjEwMAD", //图片base64编码后的值
            "face_token": "d12b7e32069d337dc5d57cd3d15e2935",
            "face_id": "d12b7e32069d337dc5d57cd3d15e2935",
            "liveness_score": 0.06,
            "spoofing": 0.0002082588035
        },
        {
            "pic" : "gAQTGF2YzU4LjkyLjEwMAD", //图片base64编码后的值
            "face_token": "6411f95f491fb665c389de03a33f12a4",
            "face_id": "6411f95f491fb665c389de03a33f12a4",
            "liveness_score": 0.06,
            "spoofing": 0.0001938246714
        },
        {
            "pic" : "gAQTGF2YzU4LjkyLjEwMAD", //图片base64编码后的值
            "face_token": "d7fb09b7942555bf1e4b8d47a63c2e4b",
            "face_id": "d7fb09b7942555bf1e4b8d47a63c2e4b",
            "liveness_score": 0.05,
            "spoofing": 0.0001579090458
        },
        {
            "pic" : "gAQTGF2YzU4LjkyLjEwMAD", //图片base64编码后的值
            "face_token": "b14f94951a1d1b0fb8770bb1ef2bf2e7",
            "face_id": "b14f94951a1d1b0fb8770bb1ef2bf2e7",
            "liveness_score": 0.05,
            "spoofing": 0.0001889262057
        },
        {
            "pic" : "gAQTGF2YzU4LjkyLjEwMAD", //图片base64编码后的值
            "face_token": "f7f363d0e71e91906dbdcfec0573de70",
            "face_id": "f7f363d0e71e91906dbdcfec0573de70",
            "liveness_score": 0.05,
            "spoofing": 0.0001999060332
        },
        {
            "pic" : "gAQTGF2YzU4LjkyLjEwMAD", //图片base64编码后的值
            "face_token": "388bdccbd70200f62c5f46a84b012691",
            "face_id": "388bdccbd70200f62c5f46a84b012691",
            "liveness_score": 0.04,
            "spoofing": 0.0001798788871
        },
        {
            "pic" : "gAQTGF2YzU4LjkyLjEwMAD", //图片base64编码后的值
            "face_token": "5fa4e1116f00912f66dba9b42a6a739e",
            "face_id": "5fa4e1116f00912f66dba9b42a6a739e",
            "liveness_score": 0.04,
            "spoofing": 0.0001976373605
        }
    ]
},
```

```
"timestamp": 1597148814,
"cached": 0,
"serverlogid": 1614796453,
"error_code": 0,
"error_msg": "SUCCESS"
}
```


活体阈值参考

请务必在产品侧做好以下条件限制

- 检测的图片为二次采集，即通过相机当场拍摄，确保时间及操作条件的约束；
- SDK输出的多帧情况，只要这些帧中，**任何一张通过了阈值，即可判断为活体，建议可用三帧情况**；
- 推荐分值采用**99.5%**

关于活体检测faceliveness的判断阈值选择，可参考以下数值信息：

阈值（Threshold）	误拒率（FRR）	通过率（TAR）	攻击拒绝率（TRR））
0.05	0.01%	99.99%	97.75%
0.1	0.05%	99.95%	98.33%
0.3 （**推荐**）	0.1%	99.9%	98.82%
0.5	0.5%	99.5%	99.67%
0.9	1%	99%	99.77%

关于以上数值的概念介绍：

- **拒绝率（TRR）**：如99%，代表100次作弊假体攻击，会有99次被拒绝。
- **误拒率（FRR）**：如0.5%，指1000次真人请求，会有5次因为活体分数低于阈值被错误拒绝。
- **通过率（TAR）**：如99%，指100次真人请求，会有99次因为活体分数高于阈值而通过。
- **阈值（Threshold）**：高于此数值，则可判断为活体。

合成图阈值参考

> 新推出合成图检测能力，在入参字段中增加spoofing参数，进行判断，若spoofing分值高于合成图推荐阈值，则可判断为合成图攻击

关于合成图检测spoofing的判断阈值选择，可参考以下数值信息：

阈值	误拒率（FRR）	通过率（TAR）	攻击拒绝率（TRR））
0.00023	5%	95%	94.93%
0.00048（**推荐**）	1%	99%	89.71%
0.00066	0.5%	99.5%	88.02%
0.00109	0.1%	99.9%	84.57%
0.00171	0.05%	99.95%	81.52%
0.00547	0.01%	99.99%	65.52%

- **阈值（Threshold）**：高于此数值，则可判断为是合成图攻击。

错误码列表

错误码	error_msg	错误信息	描述
216430	rtse/face service error	rtse/face 服务异常	请重新尝试
216431	voice service error	语音识别服务异常	请重新尝试
216432	video service call fail	视频解析服务调用失败	请重新尝试
216433	video service error	视频解析服务发生错误	请重新尝试
216434	liveness check fail	活体检测失败	请重新尝试
216500	code digit error	验证码位数错误	验证码错误， 请增加一层验证环节
216501	not found face	没有找到人脸	请查看上传视频是否包含人脸，可能因为人脸过大导致，建议用户调整距离重新录制视频
216502	session lapse	当前会话已失效	请重新获取语音验证码
216505	redis connect error	redis连接失败	请重新尝试
216506	redis operation error	redis操作失败	请重新尝试
216507	found many faces	视频中有多张人脸	请重新录制视频
216508	not found video info	没有找到视频信息	请参考文档修改视频格式
216509	voice can not identify	视频中的声音无法识别 （声音过低或者有杂 音导致无法识别）	请重新录制视频
216908	视频中人脸质量较差 (返回信息中包含具体原因)	视频中人脸质量过低 (返回的错误信息会包含 具体的错误信息包含 **illumiantion**(光照不足) **angle**(角度不好) **blur** (人脸模糊) **occlusion**(有遮挡) **too large**(人脸过大, **占屏幕2/3以上**) 等原因	请重新录制视频
222027	code length param error	验证码长度错误 (最小值大于最大值)	参考API说明文档，修改参数
222028	param[min_code_length] format error	参数格式错误	参考API说明文档，修改参数
222029	param[max_code_length] format error	参数格式错误	参考API说明文档，修改参数
222030	param[match_threshold] format error	参数格式错误	参考API说明文档，修改参数

人脸识别接口分为V2和V3两个版本，本文档为V3版本接口的说明文档，请确认您在百度云后台获得的是V3版本接口权限，再来阅读本文档。

> 辨别接口版本的方法是：在百度云后台进入【应用列表】，点击【应用名称】，在【API列表】中可以看到【请求地址】，若请求地址中带有【v3】标识，则您具有的是v3权限，可以阅读本文档；若请求地址中带有【v2】标识，则您具有的是v2权限，应该去阅读v2文档。 如果您对文档内容有任何疑问，可以通过以下几种方式联系我们：

- 在百度云控制台内 提交工单，咨询问题类型请选择**人工智能服务**；

- 如有需要讨论的疑问，欢迎进入 AI社区 与其他开发者们一同交流。

二、在线图片活体检测

能力介绍

接口能力

- **人脸基础信息**：包括人脸框位置，人脸空间旋转角度，人脸置信度等信息。
- **人脸质量检测**：判断人脸的遮挡、光照、模糊度、完整度等质量信息。可用于判断上传的人脸是否符合标准。
- **基于图片的活体检测**：基于单张图片，判断图片中的人脸是否为二次翻拍（举例：如用户A用手机拍摄了一张包含人脸的图片一，用户B翻拍了图片一得到了图片二，并用图片二伪造成用户A去进行识别操作，这种情况普遍发生在金融开户、实名认证等环节）以及是否为合成图攻击。此能力可用于H5场景下的一些人脸采集场景中，增加人脸注册的安全性和真实性。

在线调试 您可以在 API Explorer 中调试该接口，可进行签名验证、查看在线调用的请求内容和返回结果、示例代码的自动生成。

调用方式

请求URL数据格式

向API服务地址使用POST发送请求，必须在URL中带上参数`access_token`，可通过后台的API Key和Secret Key生成，具体方式请参考“Access Token获取”。

示例代码

Bash

PHP

JAVA

Python

Cpp

C#

Node

```
##### !/bin/bash
curl -i -k 'https://aip.baidubce.com/oauth/2.0/token?grant_type=client_credentials&client_id=【百度云应用的AK】&client_secret=【百度云应用的SK】'
```

> **注意**：access_token的有效期为30天，**切记需要每30天进行定期更换，或者每次请求都拉取新token**；

例如此接口，使用HTTPS POST发送：

```
https://aip.baidubce.com/rest/2.0/face/v1/merge?
access_token=24.f9ba9c5341b67688ab4added8bc91dec.2592000.1485570332.282335-8574074
```

POST中Body的参数，按照下方请求参数说明选择即可。

> **提示**：如果您为百度云老用户，正在使用其他非AI的服务，可以参考百度云AKSK鉴权方式发送请求，虽然**请求方式**和**鉴权方法**和本文所介绍的不同，但请求参数和返回结果一致。

请求说明

注意事项：

- **请求体格式化**：Content-Type为application/json，通过json格式化请求体。
- **Base64编码**：请求的图片需经过Base64编码，图片的base64编码指将图片数据编码成一串字符串，使用该字符串代替图像地址。您可以首先得到图片的二进制，然后用Base64格式编码即可。需要注意的是，图片的base64编码是不包含图片头的，如
- **图片格式**：现支持PNG、JPG、JPEG、BMP，**不支持GIF图片**
- **人脸识别接口分为V2和V3两个版本，本文档为V3版本接口的说明文档，请确认您在百度云后台获得的是V3版本接口权限，再来阅读本文档。**

> 辨别接口版本的方法是：在百度云后台进入【应用列表】，点击【应用名称】，在【API列表】中可以看到【请求地址】，若请求地址中带有【v3】标识，则您具有的是v3权限，可以阅读本文档；若请求地址中带有【v2】标识，则您具有的是v2权限，应该去阅读v2文档。

请求示例

HTTP方法：POST

请求URL： https://aip.baidubce.com/rest/2.0/face/v3/faceverify

URL参数：

参数	值
access_token	通过API Key和Secret Key获取的access_token,参考“Access Token获取”

Header：

参数	值
Content-Type	application/json

Body中放置请求参数，参数详情如下：

请求参数

参数	是否必选	类型	说明
image	是	string	图片信息(总数据大小应小于10M)，图片上传方式根据image_type来判断； **可以上传同一个用户的1张、3张或8张图片来进行活体判断** 注： (1)后端会选择每组照片中的最高分数作为整体分数。 图片通过json格式上传，格式参考表格下方示例 (2)支持1、3、8张图片输入进行计算，请求格式为数组格式
image_type	是	string	图片类型 **BASE64**:（**推荐**）图片的base64值，base64编码后的图片数据，需urlencode，编码后的图片大小不超过2M；**FACE_TOKEN**: 人脸图片的唯一标识，调用人脸检测接口时，会为每个人脸图片赋予一个唯一的FACE_TOKEN，同一张图片多次检测得到的FACE_TOKEN是同一个。
face_field	否	string	包括**age,beauty,expression,face_shape,gender,glasses,landmark,quality,face_type,spoofing信息**，逗号分隔，默认只返回face_token、活体数、人脸框、概率和旋转角度
option	否	string	场景信息，程序会视不同的场景选用相对应的模型。当前支持的场景有COMMON(通用场景)，GATE(闸机场景)，**默认使用COMMON**

说明：图片的上传使用json格式，发送内容为数组，（即 [{XXXXX}，{XXXXX}] 这种格式）具体如下：

```
[
  {
    "image": "sfasq35sadvsvqwr5q...",
    "image_type": "BASE64",
     "face_field": "age,beauty,spoofing",
     "option": "COMMON"
  },
  {
    "image": "http://xxx.baidu.com/image1.png",
    "image_type": "URL",
    "face_field": "age,beauty,spoofing",
    "option": "COMMON"
  },
  {
    "image": "9f30d19f86f89f2f07ce88b69557061a",
    "image_type": "FACE_TOKEN",
    "face_field": "age,beauty,spoofing",
    "option": "COMMON"
  }
]
```

请求示例

提示一：使用示例代码前，请记得替换其中的示例Token、图片地址或Base64信息。

提示二：部分语言依赖的类或库，请在代码注释中查看下载地址。

Bash

PHP

JAVA

Python

Cpp

C#

在线活体检测

```
curl -i -k 'https://aip.baidubce.com/rest/2.0/face/v3/faceverify?access_token=【调用鉴权接口获取的token】' --data '[{"image":"sfasq35sadvsvqwr5q...","image_type":"BASE64","face_field":"age,beauty,expression"},{"image":"http://xxx.baidu.com/image1.png","image_type":"URL","face_field":"age,beauty"}]' -H 'Content-Type:application/json; charset=UTF-8'
```

返回说明 返回参数

参数	是否必须	类型	说明
face_liveness	是	float	活体分数值
thresholds	是	array	由服务端返回最新的阈值数据（随着模型的优化，阈值可能会变化），将此参数与返回的face_liveness进行比较，可以作为活体判断的依据。 frr_1e-4：万分之一误识率的阈值；frr_1e-3：千分之一误识率的阈值；frr_1e-2：百分之一误识率的阈值。误识率越低，准确率越高，相应的拒绝率也越高
face_list	是	array	每张图片的详细信息描述，如果只上传一张图片，则只返回一个结果。
+face_token	是	string	人脸图片的唯一标识
+location	是	array	人脸在图片中的位置
++left	是	double	人脸区域离左边界的距离
++top	是	double	人脸区域离上边界的距离
++width	是	double	人脸区域的宽度
++height	是	double	人脸区域的高度
++rotation	是	int64	人脸框相对于竖直方向的顺时针旋转角，[-180,180]
+face_probability	是	double	人脸置信度，范围【0~1】，代表这是一张人脸的概率，0最小、1最大。
+angel	是	array	人脸旋转角度参数
++yaw	是	double	三维旋转之左右旋转角[-90(左), 90(右)]

++pitch	是	double	三维旋转之俯仰角度[-90(上), 90(下)]
++roll	是	double	平面内旋转角[-180(逆时针), 180(顺时针)]
+age	否	double	年龄 ，当**face_field包含age时返回**
+beauty	否	int64	美丑打分，范围0-100，越大表示越美。当**face_fields包含beauty时返回**
+expression	否	array	表情，当 **face_field包含expression时返回**
++type	否	string	**none**:不笑；**smile**:微笑；**laugh**:大笑
++probability	否	double	表情置信度，范围【0~1】，0最小、1最大。
+face_shape	否	array	脸型，当**face_field包含face_shape时返回**
++type	否	double	**square**: 正方形 **triangle**:三角形 **oval**: 椭圆 **heart**: 心形 **round**: 圆形
++probability	否	double	置信度，范围【0~1】，代表这是人脸形状判断正确的概率，0最小、1最大。
+gender	否	array	性别，**face_field包含gender时返回**
++type	否	string	male:**男性** female:**女性**
++probability	否	double	性别置信度，范围【0~1】，0代表概率最小、1代表最大。
+glasses	否	array	是否带眼镜，**face_field包含glasses时返回**
++type	否	string	**none**:无眼镜，**common**:普通眼镜，**sun**:墨镜
++probability	否	double	眼镜置信度，范围【0~1】，0代表概率最小、1代表最大。
+face_type	否	array	真实人脸/卡通人脸 **face_field包含face_type时返回**
++type	否	string	**human**: 真实人脸 **cartoon**: 卡通人脸
++probability	否	double	人脸类型判断正确的置信度，范围【0~1】，0代表概率最小、1代表最大。
+landmark	否	array	4个关键点位置，左眼中心、右眼中心、鼻尖、嘴中心。**face_field包含landmark时返回**
+landmark72	否	array	72个特征点位置 **face_field包含landmark时返回**
+quality	否	array	人脸质量信息。**face_field包含quality时返回**
++occlusion	否	array	人脸各部分遮挡的概率，范围[0~1]，0表示完整，1表示不完整
+++left_eye	否	double	左眼遮挡比例，[0-1] ， 1表示完全遮挡
+++right_eye	否	double	右眼遮挡比例，[0-1] ， 1表示完全遮挡
+++nose	否	double	鼻子遮挡比例，[0-1] ， 1表示完全遮挡

+++mouth	否	double	嘴巴遮挡比例，[0-1] ， 1表示完全遮挡
+++left_cheek	否	double	左脸颊遮挡比例，[0-1] ， 1表示完全遮挡
+++right_cheek	否	double	右脸颊遮挡比例，[0-1] ， 1表示完全遮挡
+++chin	否	double	下巴遮挡比例，，[0-1] ， 1表示完全遮挡
++blur	否	double	人脸模糊程度，范围[0~1]，0表示清晰，1表示模糊
++illumination	否	double	取值范围在[0~255], 表示脸部区域的光照程度 越大表示光照越好
++completeness	否	int64	人脸完整度，0或1, 0为人脸溢出图像边界，1为人脸都在图像边界内
+spoofing	否	double	合成图打分 判断图片是否为合成图 face_field包含时返回spoofing

返回示例

```
{
  "error_code": 0,
  "error_msg": "SUCCESS",
  "log_id": 9999201750012,
  "timestamp": 1587021157,
  "cached": 0,
  "result": {
    "thresholds": {
      "frr_1e-4": 0.05,
      "frr_1e-3": 0.3,
      "frr_1e-2": 0.9
    },
    "face_liveness": 1,
    "face_list": [
      {
        "face_token": "6e6880584e9ad6d22e309da37ed72b78",
        "location": {
          "left": 60.21,
          "top": 134.93,
          "width": 162,
          "height": 164,
          "rotation": -4
        },
        "face_probability": 1,
        "angle": {
          "yaw": -0.9,
          "pitch": 12.87,
          "roll": -5.36
        },
        "liveness": {
          "livemapscore": 1
        },
        "spoofing": 0.0002664364583
      }
    ]
  }
}
```


活体阈值参考

请务必在产品侧做好以下条件限制

- 检测的图片为二次采集，即通过相机当场拍摄，确保时间及操作条件的约束；
- SDK输出的多帧情况，只要这些帧中，**任何一张通过了阈值，即可判断为活体，建议可用三帧情况**；
- 推荐分值采用**99.5%**

关于活体检测faceliveness的判断阈值选择，可参考以下数值信息（金融保险等对安全性要求较高的业务可适当调高活体阈值，如：0.5）：

阈值（Threshold）	误拒率（FRR）	通过率（TAR）	攻击拒绝率（TRR））
0.05	0.01%	99.99%	97.75%
0.1	0.05%	99.95%	98.33%
0.3 （推荐）	0.1%	99.9%	98.82%
0.5	0.5%	99.5%	99.67%
0.9	1%	99%	99.77%

关于以上数值的概念介绍：

- **拒绝率（TRR）**：如99%，代表100次作弊假体攻击，会有99次被拒绝。
- **误拒率（FRR）**：如0.5%，指1000次真人请求，会有5次因为活体分数低于阈值被错误拒绝。
- **通过率（TAR）**：如99%，指100次真人请求，会有99次因为活体分数高于阈值而通过。
- **阈值（Threshold）**：高于此数值，则可判断为活体。

合成图阈值参考

> 新推出合成图检测能力，在face_field字段中增加spoofing参数，进行判断，若spoofing分值高于合成图推荐阈值，则可判断为合成图攻击；此参数在face_liveness基础上进行合成图判断。

关于合成图检测spoofing的判断阈值选择，可参考以下数值信息：

阈值	误拒率（FRR）	通过率（TAR）	攻击拒绝率（TRR））
0.00023	5%	95%	94.93%
0.00048（**推荐**）	1%	99%	89.71%
0.00066	0.5%	99.5%	88.02%
0.00109	0.1%	99.9%	84.57%
0.00171	0.05%	99.95%	81.52%
0.00547	0.01%	99.99%	65.52%

- **阈值（Threshold）**：高于此数值，则可判断为是合成图攻击。

质量检测参考

指标	字段与解释	推荐数值界限
遮挡范围	**occlusion**，取值范围[0~1]，0为无遮挡，1是完全遮挡 含有多个具体子字段，表示脸部多个部位 通常用作判断头发、墨镜、口罩等遮挡	left_eye：0.6, #左眼被遮挡的阈值 right_eye：0.6, #右眼被遮挡的阈值 nose：0.7, #鼻子被遮挡的阈值 mouth：0.7, #嘴巴被遮挡的阈值 left_check：0.8, #左脸颊被遮挡的阈值 right_check：0.8, #右脸颊被遮挡的阈值 chin_contour：0.6, #下巴被遮挡阈值
模糊度范围	**blur**，取值范围[0~1]，0是最清晰，1是最模糊	小于0.7
光照范围	**illumination**，取值范围[0~255] 脸部光照的灰度值，0表示光照不好 以及对应客户端SDK中，YUV的Y分量	大于40
姿态角度	**Pitch**：三维旋转之俯仰角度[-90(上), 90(下)] **Roll**：平面内旋转角[-180(逆时针), 180(顺时针)] **Yaw**：三维旋转之左右旋转角[-90(左), 90(右)]	分别小于20度
人脸完整度	**completeness**（0或1），0为人脸溢出图像边界，1为人脸都在图像边界内	视业务逻辑判断
人脸大小	人脸部分的大小 建议长宽像素值范围：80*80~200*200	人脸部分不小于**100*100**像素

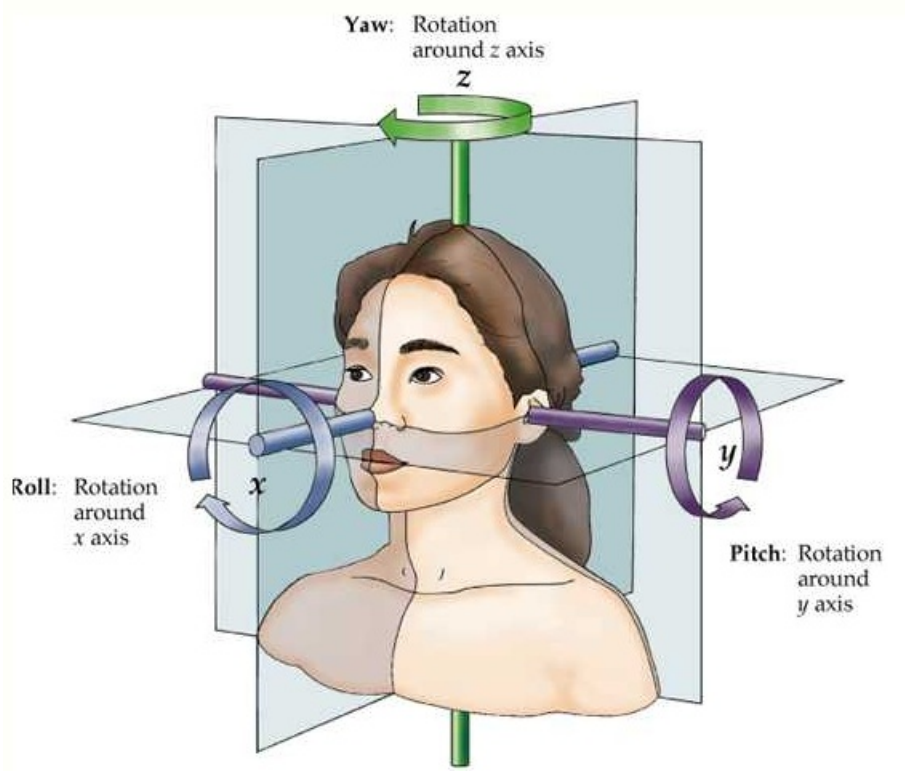
人脸空间姿态角参考

姿态角分为Pitch、Roll、Yaw，用于表示人脸在空间三维坐标系内的角度，常用于判断识别角度的界限值。

各角度阈值如下：

```
Pitch：三维旋转之俯仰角度，范围：[-90（上），90（下）]，推荐俯仰角绝对值不大于20度；
Roll：平面内旋转角，范围：[-180（逆时针），180（顺时针）]，推荐旋转角绝对值不大于20度；
Yaw：三维旋转之左右旋转角，范围：[-90（左），90（右）]，推荐旋转角绝对值不大于20度；
```

各角度范围示意图如下：



从姿态角度来看，**这三个值的绝对值越小越好**，这样代表人脸足够正视前方，最利于实际注册/识别使用。

错误码

请参考人脸识别错误码

公有云接口

人脸比对加密版

人脸识别接口分为V2和V3两个版本，本文档为V3版本接口的说明文档，请确认您在百度云后台获得的是V3版本接口权限，再来阅读本文档。

> 辨别接口版本的方法是：在百度云后台进入【应用列表】，点击【应用名称】，在【API列表】中可以看到【请求地址】，若请求地址中带有【v3】标识，则您具有的是v3权限，可以阅读本文档；若请求地址中带有【v2】标识，则您具有的是v2权限，应该去阅读v2文档。 如果您对文档内容有任何疑问，可以通过以下几种方式联系我们：

- 在百度云控制台内 提交工单，咨询问题类型请选择**人工智能服务**；
- 如有需要讨论的疑问，欢迎进入 AI社区 与其他开发者们一同交流。

人脸比对加密版

能力介绍

业务能力 接口能力

- **两张人脸图片相似度对比**：比对两张图片中人脸的相似度，并返回相似度分值；
- **多种图片类型**：支持**生活照**、**证件照**、**身份证芯片照**、**带网纹照**四种类型的人脸对比；
- **活体检测控制**：基于图片中的破绽分析，判断其中的人脸是否为**二次翻拍**（举例：如用户A用手机拍摄了一张包含人脸的图片一，用户B翻拍了图片一得到了图片二，并用图片二伪造成用户A去进行识别操作，这种情况普遍发生在金融开户、实名认证等环节。）；
- **质量检测控制**：分析图片的中人脸的模糊度、角度、光照强度等特征，判断图片质量；

- **图片加密及风控：**配合采集SDK4.1版本使用，对采集SDK输出的加密图片进行解密（加密传输可以有效避免第三方非法黑产绕过APP模拟请求攻击云端接口的行为）；以及结合百度安全实验室大数据风控能力，对采集SDK的发起端设备进行风控识别，辨别是否为风险设备。

业务应用

用于比对多张图片中的人脸相似度并返回两两比对的得分，可用于判断两张脸是否是同一人的可能性大小。

典型应用场景：如**人证合一验证**，**用户认证**等，可与您现有的人脸库进行比对验证。

调用方式

请求URL数据格式

向API服务地址使用POST发送请求，必须在URL中带上参数`access_token`，可通过后台的API Key和Secret Key生成，具体方式请参考“Access Token获取”。 **请求URL数据格式**

向API服务地址使用POST发送请求，必须在URL中带上参数`access_token`，可通过后台的API Key和Secret Key生成，具体方式请参考“Access Token获取”。

示例代码

Bash

PHP

JAVA

Python

Cpp

C#

Node

```
###### !/bin/bash
curl -i -k 'https://aip.baidubce.com/oauth/2.0/token?grant_type=client_credentials&client_id=【百度云应用的AK】&client_secret=【百度云应用的SK】'
```

> **注意**：`access_token`的有效期为30天，**切记需要每30天进行定期更换，或者每次请求都拉取新token**；

例如此接口，使用HTTPS POST发送：

```
https://aip.baidubce.com/rest/2.0/face/v1/merge?
access_token=24.f9ba9c5341b67688ab4added8bc91dec.2592000.1485570332.282335-8574074
```

POST中Body的参数，按照下方请求参数说明选择即可。

> **提示**：如果您为百度云老用户，正在使用其他非AI的服务，可以参考百度云AKSK鉴权方式发送请求，虽然**请求方式**和**鉴权方法**和本文所介绍的不同，但请求参数和返回结果一致。

请求说明

注意事项：

- **请求体格式化**：Content-Type为`application/json`，通过json格式化请求体。
- **Base64编码**：请求的图片需经过`Base64编码`，图片的base64编码指将图片数据编码成一串字符串，使用该字符串代替图像地址。您可以首先得到图片的二进制，然后用Base64格式编码即可。需要注意的是，图片的base64编码是不包含图片头的，如``
- **图片格式**：现支持PNG、JPG、JPEG、BMP，**不支持GIF图片**
- **人脸识别接口分为V2和V3两个版本，本文档为V3版本接口的说明文档，请确认您在百度云后台获得的是V3版本接口权限，再来阅读本文档。**

> 辨别接口版本的方法是：在百度云后台进入【应用列表】，点击【应用名称】，在【API列表】中可以看到【请求地址】，若请求地址中带有【v3】标识，则您具有的是v3权限，可以阅读本文档；若请求地址中带有【v2】标识，则您具有的是v2权限，应该去阅读v2文档。

请求示例

HTTP方法：`POST`

请求URL：`https://aip.baidubce.com/rest/2.0/face/v3/matchsec`

URL参数：

参数	值
access_token	通过API Key和Secret Key获取的access_token,参考“Access Token获取”

Header如下：

参数	值
Content-Type	application/json

Body中放置请求参数，参数详情如下：

请求参数

参数	必选	类型	说明
image	是	string	采集SDK4.1版本上传的加密图片Base64信息
image_type	是	string	图片类型 **BASE64**:图片的base64值，base64编码后的图片数据，编码后的图片大小不超过2M；
			人脸的类型 **LIVE**：表示生活照：通常为手机、相机拍摄的人像图片、或从网络获取的人像图片等，

face_type	否	string	**IDCARD**：表示身份证芯片照：二代身份证内置芯片中的人像照片， **WATERMARK**：表示带水印证件照：一般为带水印的小图，如公安网小图 **CERT**：表示证件照片：如拍摄的身份证、工卡、护照、学生证等证件图片 **INFRARED** 表示红外照片：使用红外相机拍摄的照片 默认LIVE
quality_control	否	string	图片质量控制 **NONE**: 不进行控制 **LOW**:较低的质量要求 **NORMAL**: 一般的质量要求 **HIGH**: 较高的质量要求 **默认 NONE** 若图片质量不满足要求，则返回结果中会提示质量检测失败
liveness_control	否	string	活体检测控制 **NONE**: 不进行控制 **LOW**:较低的活体要求(高通过率 低攻击拒绝率) **NORMAL**: 一般的活体要求(平衡的攻击拒绝率, 通过率) **HIGH**: 较高的活体要求(高攻击拒绝率 低通过率) **默认 NONE** 若活体检测结果不满足要求，则返回结果中会提示活体检测失败
face_sort_type	否	int	人脸检测排序类型 **0**:代表检测出的人脸按照人脸面积从大到小排列 **1**:代表检测出的人脸按照距离图片中心从近到远排列 **默认为0**
register_image	是	string	图片信息(**总数据大小应小于10M**)，图片上传方式根据image_type来判断。本图片特指客户服务器上传图片，非加密图片Base64值
register_image_type	是	string	图片类型 **BASE64**:图片的base64值，base64编码后的图片数据，编码后的图片大小不超过2M； **URL**:图片的 URL地址(可能由于网络等原因导致下载图片时间过长)； **FACE_TOKEN**: 人脸图片的唯一标识，调用人脸检测接口时，会为每个人脸图片赋予一个唯一的FACE_TOKEN，同一张图片多次检测得到的FACE_TOKEN是同一个。
register_face_type	否	string	人脸的类型 **LIVE**：表示生活照：通常为手机、相机拍摄的人像图片、或从网络获取的人像图片等， **IDCARD**：表示身份证芯片照：二代身份证内置芯片中的人像照片， **WATERMARK**：表示带水印证件照：一般为带水印的小图，如公安网小图 **CERT**：表示证件照片：如拍摄的身份证、工卡、护照、学生证等证件图片 **INFRARED** 表示红外照片：使用红外相机拍摄的照片 默认LIVE
register_quality_control	否	string	图片质量控制 **NONE**: 不进行控制 **LOW**:较低的质量要求 **NORMAL**: 一般的质量要求 **HIGH**: 较高的质量要求 **默认 NONE** 若图片质量不满足要求，则返回结果中会提示质量检测失败
register_liveness_control	否	string	活体检测控制 **NONE**: 不进行控制 **LOW**:较低的活体要求(高通过率 低攻击拒绝率) **NORMAL**: 一般的活体要求(平衡的攻击拒绝率, 通过率) **HIGH**: 较高的活体要求(高攻击拒绝率 低通过率) **默认 NONE** 若活体检测结果不满足要求，则返回结果中会提示活体检测失败
risk_identify	否	boolean	是否进行风险识别 默认False Ture：打开风险识别。（接受SDK端传入的指纹信息，返回识别结果） False：关闭风险识别。

ip	否	string	需要风控判别的ip地址
phone	否	string	需要风控判别的手机号

说明：两张图片的上传使用json格式，如下：

```
{
"image": "image", //采集SDK4.1传入的加密后的图片BASE64信息
"image_type": "BASE64",
"face_type": "LIVE",
"quality_control": "LOW",
"liveness_control": "HIGH",

"register_image": "image", //服务端传入的非加密图片的BASE64
"register_image_type": "BASE64",
"register_face_type": "LIVE",
"register_quality_control": "LOW",
"register_liveness_control": "HIGH",

"risk_identify": "Ture",
"ip": "...ip",
"phone": "...phone"

}
```

返回说明

返回参数

参数名	必选	类型	说明
score	是	float	人脸相似度得分，推荐阈值80分
face_list	是	array	人脸信息列表
+face_token	是	string	人脸的唯一标志
+risk_level	否	string	判断设备是否发生过风险行为来判断风险级别，取值（数值有高到低）： 1 – 高危 2 – 嫌疑 3 – 普通 4 – 正常
+risk_tag	否	string	风险标签，若判断未为有风险，则会有风险标签jason 数组告知风险类型
+risk_warn_code	否	int	风控服务异常，请重试若多次重试后无效，请提交工单咨询

- 返回示例

```
    "result": {
      "score": 21.58333969,
      "face_list": [
        {
          "face_token": "470795667ddb21e38908cf3dc31dcdec"
        },
        {
          "face_token": "605a1fb2a02f4886490db3e4b5f6f0c9"
        }
      ],
      "verify_status": 0
    },
    "risk_level": "1",
    "log_id": 1349004812288524288,
    "risk_tag": [
      "general_inject"
    ]
}
```


- **质量控制参数说明**

> 不同的控制度下所对应的质量控制阈值，如果检测出来的质量信息某一项不符合控制阈值的要求，则会返回错误信息。

遮挡情况的阈值

控制度	left_eye	right_eye	nose	mouth	left_cheek	right_cheek	chin_contour
LOW	0.8	0.8	0.8	0.8	0.8	0.8	0.8
NORMAL	0.6	0.6	0.6	0.6	0.6	0.6	0.6
HIGH	0.2	0.2	0.2	0.2	0.2	0.2	0.2

光照、模糊度、完整度的阈值

控制度	illumination	blurdegree	completeness
LOW	20	0.8	0
NORMAL	40	0.6	0
HIGH	100	0.2	1

活体控制参数说明

> 不同的控制度下所对应的活体控制阈值，如果检测出来的活体分数小于控制阈值，则会返回错误信息。

控制度	阈值	说明
LOW	0.05	活体误拒率：万分之一；拒绝率：97.75%
NORMAL	0.3	活体误拒率：千分之一；拒绝率：98.82%
HIGH	0.9	活体误拒率：百分之一；拒绝率：99.77%

> 1、误拒率: 把**真人识别为假人**的概率. 阈值越高，安全性越高, 要求也就越高, 对应的误识率就越高
> 2、通过率=1-误拒率

关于以上数值的概念介绍：

拒绝率（TRR）：如99%，代表100次作弊假体攻击，会有99次被拒绝。 误拒率（FRR）：如0.5%，指1000次真人请求，会有5次因为活体分数低于阈值被错误拒绝。 通过率（TAR）：如99%，指100次真人请求，会有99次因为活体分数高于阈值而通过。 阈值（Threshold）：高于此数值，则可判断为活体。

错误码

请参考人脸识别错误码

人脸实名认证加密版

人脸识别接口分为V2和V3两个版本，本文档为V3版本接口的说明文档，请确认您在百度云后台获得的是V3版本接口权限，再来阅读本文档。

> 辨别接口版本的方法是：在百度云后台进入【应用列表】，点击【应用名称】，在【API列表】中可以看到【请求地址】，若请求地址中带有【v3】标识，则您具有的是v3权限，可以阅读本文档；若请求地址中带有【v2】标识，则您具有的是v2权限，应该去阅读v2文档。 如果您对文档内容有任何疑问，可以通过以下几种方式联系我们：

- 在百度云控制台内 提交工单，咨询问题类型请选择**人工智能服务**；
- 如有需要讨论的疑问，欢迎进入 AI社区 与其他开发者们一同交流。

人脸实名认证加密版 能力介绍

> 本接口需要配合增强级人脸采集SDK使用，增强级SDK输出的图片会进行加密，在人脸实名认证加密版接口进行解密。以此防止信息传输过程中泄露或遭到第三方非法攻击，此接口必须配合百度人脸采集SDK使用，若您需要不带加密功能的接口，或想使用自己的加密方式，请使用人脸实名认证非加密版接口。

业务能力

- **质量检测（可选）**：判断图片中是否包含人脸，以及人脸在姿态、遮挡、模糊、光照等方面是否符合识别条件；
- **活体检测（可选）**：基于图片中的破绽分析，判断其中的人脸是否为**二次翻拍**（举例：如用户A用手机拍摄了一张包含人脸的图片一，用户B翻拍了图片一得到了图片二，并用图片二伪造成用户A去进行识别操作，这种情况普遍发生在金融开户、实名认证等环节。）；
- **人脸实名认证（必选）**：基于姓名和身份证号，调取公民身份证小图（源自权威数据源），将当前获取的人脸图片，与此证件小图进行对比，得出比对分数，并基于此进行业务判断是否为同一人。由于公民身份证小图，具有最权威的身份证明作用，故对用户本人的验证结果可信度也最为合理。

业务逻辑

- 上述三项能力为顺序串行验证，接口默认返回身份对比分值，质量检测和活体检测为可选项。如选择了这两项，则验证顺序为`人脸质量检测`->`活体检测`->`人脸实名认证`。您也可以根据业务场景，选择这两项中的某一项或都不选择。
- **如选择了前两个环节，则有任意一个条件不通过，则整个请求流程终止，并返回错误码，描述具体的不符合信息**。
- 基于此顺序串行验证逻辑，可以避免大量不符合条件的请求流转到人脸实名认证，节约您的成本。

推荐阈值

- 此接口使用的对比算法，针对带水纹证件照采用了专项的模型处理，可保证水纹信息的影响降到尽可能低。
- 如比对成功，最终返回的有效数据为一个**对比分值**，在0~1之间，您可以设定具体的阈值来判断是否验证通过。
- **人证相似度的推荐阈值为0.8，对应的误识率为万分之一**。

计费逻辑

- 前两个环节图片不符合校验规则，会以error_code形式反馈，属于正向业务判断，这两个环节的请求全部免费。真正请求到人脸实名认证这步并返回结果，才会进行计费。价格文档

调用方式

请求URL数据格式

向API服务地址使用POST发送请求，必须在URL中带上参数access_token，可通过后台的API Key和Secret Key生成，具体方式请参考“Access Token获取”。

示例代码

Bash

PHP

JAVA

Python

Cpp

C#

Node

```
###### !/bin/bash
curl -i -k 'https://aip.baidubce.com/oauth/2.0/token?grant_type=client_credentials&client_id=【百度云应用的AK】&client_secret=【百度云应用的SK】'
```

> **注意**：access_token的有效期为30天，**切记需要每30天进行定期更换，或者每次请求都拉取新token**；

例如此接口，使用HTTPS POST发送：

```
https://aip.baidubce.com/rest/2.0/face/v1/merge?access_token=24.f9ba9c5341b67688ab4added8bc91dec.2592000.1485570332.282335-8574074
```

POST中Body的参数，按照下方请求参数说明选择即可。

> **提示**：如果您为百度云老用户，正在使用其他非AI的服务，可以参考百度云AKSK鉴权方式发送请求，虽然**请求方式**和**鉴权**

> **方法**和本文所介绍的不同，但请求参数和返回结果一致。

请求说明

注意事项：

- **请求体格式化**：Content-Type为`application/json`，通过`json`格式化请求体。
- **Base64编码**：请求的图片需经过`Base64编码`，图片的base64编码指将图片数据编码成一串字符串，使用该字符串代替图像地址。您可以首先得到图片的二进制，然后用Base64格式编码即可。需要注意的是，图片的base64编码是不包含图片头的，如``
- **图片格式**：现支持PNG、JPG、JPEG、BMP，**不支持GIF图片**
- **人脸识别接口分为V2和V3两个版本，本文档为V3版本接口的说明文档，请确认您在百度云后台获得的是V3版本接口权限，再来阅读本文档。**

> 辨别接口版本的方法是：在百度云后台进入【应用列表】，点击【应用名称】，在【API列表】中可以看到【请求地址】，若请求地址中带有【v3】标识，则您具有的是v3权限，可以阅读本文档；若请求地址中带有【v2】标识，则您具有的是v2权限，应该去阅读v2文档。

请求示例

HTTP方法：`POST`

请求URL： `https://aip.baidubce.com/rest/2.0/face/v3/person/verifySec`

URL参数：

参数	值
access_token	通过API Key和Secret Key获取的access_token,参考“Access Token获取”

Header：

参数	值
Content-Type	application/json

Body中放置请求参数，参数详情如下：

请求参数

参数	必选	类型	说明
image	是	string	图片信息(总数据大小应小于10M)，图片上传方式根据image_type来判断
image_type	是	string	图片类型 BASE64:图片的base64值，base64编码后的图片数据，编码后的图片大小不超过2M；图片尺寸不超过1920*1080 URL
id_card_number	是	string	身份证号码
name	是	string	姓名（注：需要是UTF-8编码的中文）
quality_control	否	string	图片质量控制 NONE: 不进行控制 LOW:较低的质量要求 NORMAL: 一般的质量要求 HIGH: 较高的质量要求 默认 NONE
liveness_control	否	string	活体检测控制 NONE: 不进行控制 LOW:较低的活体要求(高通过率 低攻击拒绝率) NORMAL: 一般的活体要求(平衡的攻击拒绝率, 通过率) HIGH: 较高的活体要求(高攻击拒绝率 低通过率) 默认NONE
spoofing_control	否	string	合成图控制 NONE: 不进行控制 LOW:较低的合成图检测要求(高通过率 低攻击拒绝率) NORMAL: 一般的合成图检测要求(平衡的攻击拒绝率, 通过率) HIGH: 较高的活体要求(高攻击拒绝率 低通过率) 默认NONE
risk_identify	否	boolean	是否进行风险识别 默认False Ture：打开风险识别。（接受SDK端传入的指纹信息，返回识别结果） False：关闭风险识别。
ip	否	string	需要风控判别的ip地址
phone	否	string	需要风控判别的手机号

返回说明

- **返回参数**

参数	必须	类型	说明
log_id	是	uint64	日志id
verify_status	是	int	认证状态，取值： 0 姓名与身份证匹配，可进行人脸比对 1 身份证号与姓名不匹配或该身份证号不存在 2 公安网图片不存在或质量过低
score	否	float	与公安小图相似度可能性，用于验证生活照与公安小图是否为同一人，有正常分数时为[0~100]，推荐阈值80，超过即判断为同一人
risk_level	否	string	判断设备是否发生过风险行为来判断风险级别，取值（数值有高到低）： 1 – 高危 2 – 嫌疑 3 – 普通 4 – 正常
risk_tag	否	string	风险标签，若判断未为有风险，则会有风险标签jason 数组告知风险类型
risk_warn_code	否	int	风控服务异常，请重试若多次重试后无效，请提交工单咨询

返回示例

```
{
  "result": {
     "score": 89.97354889,
     "verify_status": 0
  },
  "log_id": 1321833960497479680
}
```

- **质量控制参数说明**

> 不同的控制度下所对应的质量控制阈值，如果检测出来的质量信息某一项不符合控制阈值的要求，则会返回错误信息。

控制度	left_eye	right_eye	nose	mouth	left_cheek	right_cheek	chin_contour	illumination	blurdegree	completeness
LOW	0.8	0.8	0.8	0.8	0.8	0.8	0.8	20	0.8	0
NORMAL	0.6	0.6	0.6	0.6	0.6	0.6	0.6	40	0.6	0
HIGH	0.2	0.2	0.2	0.2	0.2	0.2	0.2	100	0.2	1

活体控制参数说明

> 不同的控制度下所对应的活体控制阈值，如果检测出来的活体分数小于控制阈值，则会返回错误信息。

控制度	阈值	说明
LOW	0.05	活体误拒率：万分之一；拒绝率：97.75%
NORMAL（推荐）	0.3	活体误拒率：千分之一；拒绝率：98.82%
HIGH	0.9	活体误拒率：百分之一；拒绝率：99.77%

> 1、误拒率: 把**真人识别为假人**的概率. 阈值越高，安全性越高, 要求也就越高, 对应的误识率就越高

> 2、通过率=1-误拒率

关于以上数值的概念介绍：

拒绝率（TRR）：如99%，代表100次作弊假体攻击，会有99次被拒绝。 误拒率（FRR）：如0.5%，指1000次真人请求，会有5次因为活体分数低于阈值被错误拒绝。 通过率（TAR）：如99%，指100次真人请求，会有99次因为活体分数高于阈值而通过。 阈值（Threshold）：高于此数值，则可判断为活体。

合成图控制参数说明

> 不同的控制度下所对应的合成图检测（PS、人脸融合等）阈值，如果检测出来的分数大于控制阈值，则会返回错误信息。

控制度	阈值	误拒率（FRR）	通过率	攻击拒绝率（TRR））
LOW	0.00023	5%	95%	94.93%
NORMAL（推荐）	0.00048	1%	99%	89.71%
HIGH	0.00109	0.1%	99.9%	84.57%

> 1、误拒率: 把**正常图片识别为合成图片**的概率. 阈值越高，安全性越高, 要求也就越高, 对应的误识率就越高
> 2、通过率=1-误拒率

关于以上数值的概念介绍：

阈值（Threshold）：高于此数值，则可判断为是合成图攻击。

错误码

请参考人脸识别错误码

炫瞳活体检测

如果您对文档内容有任何疑问，可以通过以下几种方式联系我们：

> 在百度云控制台内 提交工单，咨询问题类型请选择**人工智能服务**；
> 如有需要讨论的疑问，欢迎进入 AI社区 与其他开发者们一同交流。

- **炫瞳活体检测：**基于屏幕颜色打光的方式，通过面部反光和**瞳孔反光**对核验人员进行活体判断。相比于行业内传统的动作活体和视频活体检测方式，通过率大大提升，核验过程更加流畅便捷，有效拦截视频、图片伪造、3D面具、合成图等黑产攻击。通过H5实名认证方案接入时，**炫瞳活体检测过程属于实时校验**，直接在前端完成检测流程，无需用户录制视频上传至后端，提升整体核验流程的流畅度及用户体验。

注： 此能力暂不支持纯服务端接入的方式。如您需要使用此能力，需通过**H5实名认证方案**进行接入。您可根据您的业务环境选择**APP方案接入**或**H5方案接入**。

标准级APP端实名认证方案

方案简介

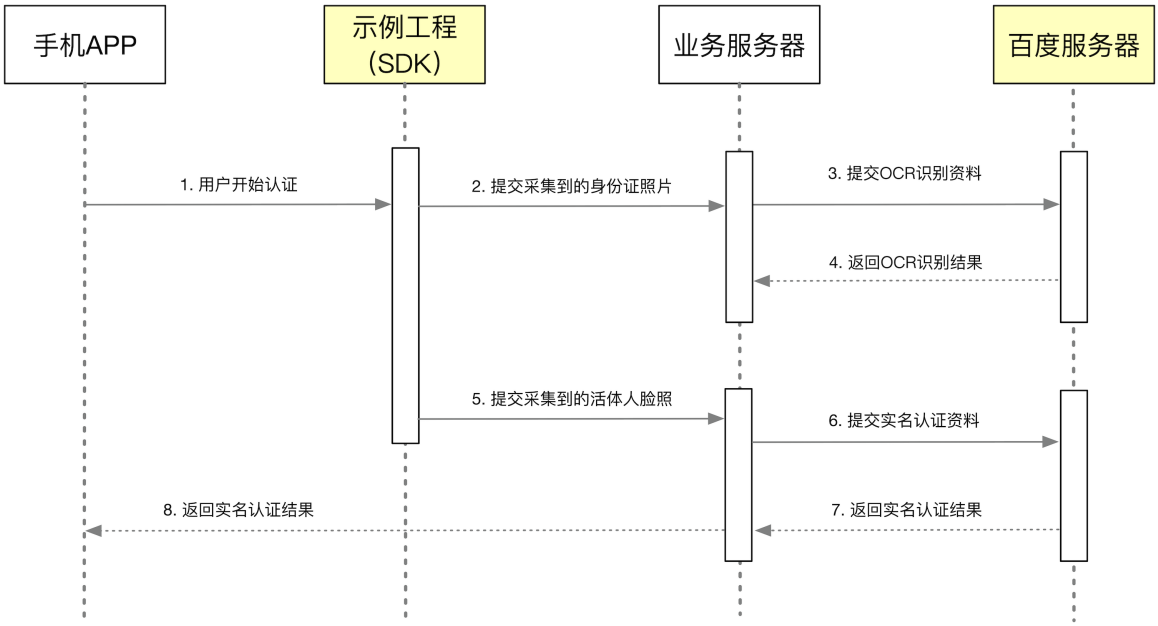
方案简介

- 标准级APP实名认证方案提供标准化的人证核验流程，提供人脸比对、证件识别、活体检测等多项组合能力； 端云配合高防攻击拦截能力，可抵挡屏幕、照片、视频、换脸、面具、3D模型的攻击。 标准级APP实名认证方案适用于**安卓/IOS系统的APP场景**中实现用户实名认证。如果您的业务场景是在微信小程序、公众号、H5等业务场景，推荐采用 **标准级H5实名认证**

方案。

- 【重要】近期有第三方非法黑产模拟用户APP请求攻击云端注册接口的风险行为，针对此类攻击行为，您可配置百度的 增强级APP实名认证方案 进行集成，增强级实名认证方案中增加了**数据传输加密**及**大数据风控能力**，可以规避此类攻击风险。

接入时序图



方案接入步骤

- 标准级APP实名认证方案的具体接入步骤请参考方案接入指南

方案集成指南

本文档介绍了标准级APP实名认证方案配置流程，以及APP集成开发流程。 一、 准备工作

在正式集成前，需要做一些准备工作，完成一些账号、应用及配置，具体如下：

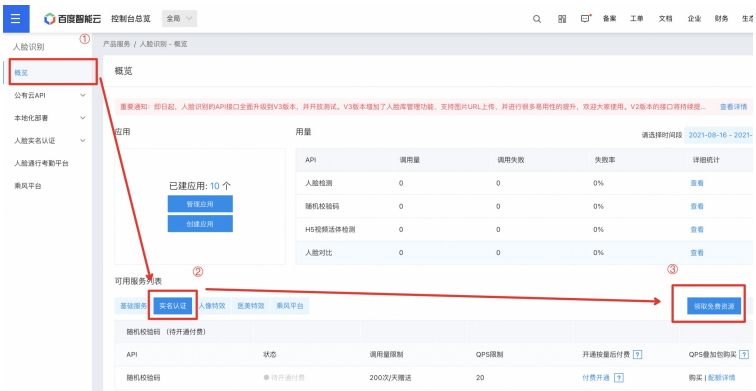
Step1: 注册成为开发者

> 在使用百度人脸实名认证方案之前，首先需注册百度智能云账号，账号注册方式请参考账号注册指南。
>
> 百度智能云账号注册完成以后，为顺利调用百度AI能力，需完成企业认证。具体认证方式请参考企业认证指南。

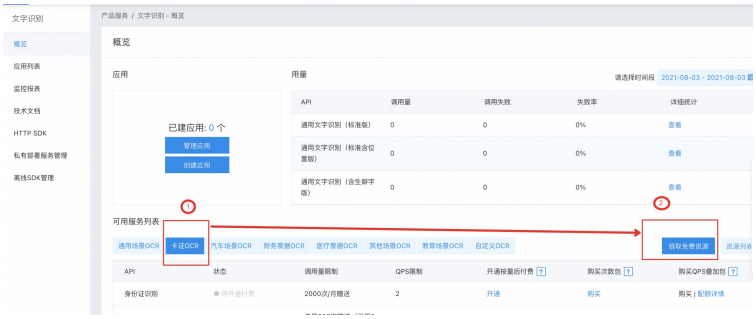
Step2：创建应用

2.1 输入应用名称，领取免费额度

- 创建好账号后，在正式调用AI 能力之前，需首先创建应用，应用是调用服务的基础能力单元。
- 同时领取接口所需的**免费调用额度**，用于接入测试。如下图所示：



- 除人脸服务接口的免费调用额度外，还需领取**身份证识别**接口的**免费调用额度**，用来调用身份证OCR识别功能（必须领取，否则会报错服务异常），点击此处，按下图所示进行领取。



- 如您之前已经领取过免费额度，无需重复领取，请跳至下一步骤。

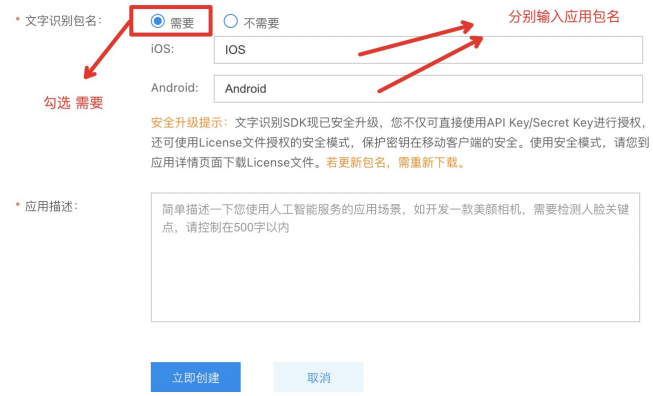
2.2 勾选所需接口

- 人脸识别服务相关接口已默认勾选且不可取消。

- **注：**「接口选择」过程中，还**需勾选「文字识别」中的「身份证识别」接口**，用于实现身份核验流程中的身份证识别功能。如下图所示。

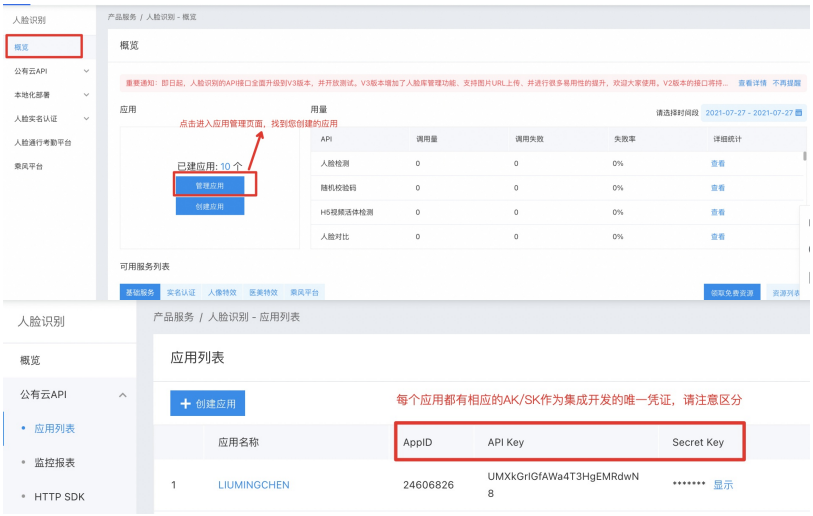
2.3 输入应用包名

- **在「文字识别包名」处选择「需要」**，并根据您的APP应用信息填写包名。此处为必要操作，否则将无法顺利下载集成文件。**至此应用创建完成。**



2.4 获取密钥信息（AK/SK）

完成应用创建后，平台将会分配给您此应用的相关凭证，主要为**AppID、API Key、Secret Key**，以上三个信息是您应用实际开发的主要凭证，每个应用之间各不相同，请您妥善保管。您可在控制台的**应用管理页面**找到以上信息。如下图所示



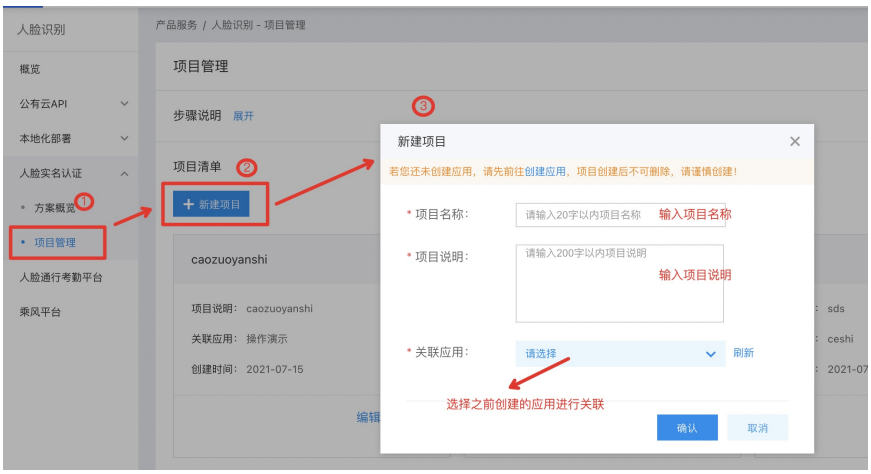
该AK/SK用于调用在线API 如：身份验证。**在之后下载的集成文件（示例工程）中需要填写正确的AK/SK以顺利集成。**

注：开发中请注意区分多份AK/SK（API Key、Secret Key），若填写的AK/SK与开发的应用不对应，会产生鉴权错误。

Step3：创建项目

- 进入控制台-人脸实名认证页面，选择『**项目管理**』页面，点击『**新建项目**』，进行项目创建，如下图所示。

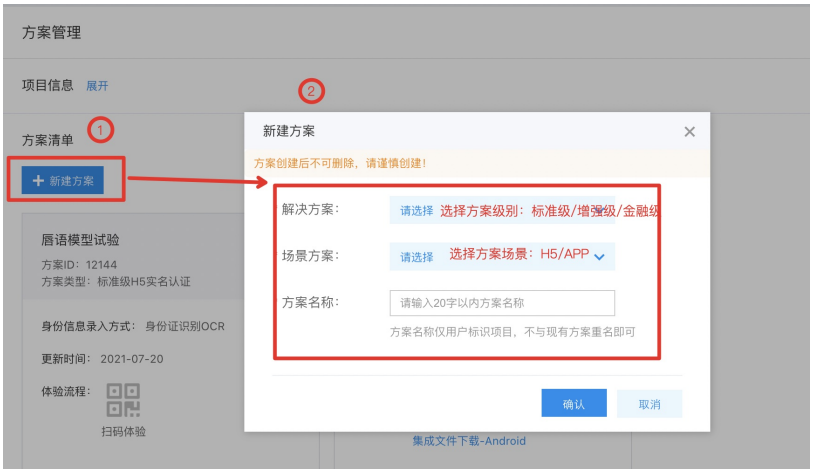
> 创建项目前，请确保您在应用控制台已创建应用，若您未创建应用，请参考Step2创建应用后，再进行项目创建.



Step4：创建方案

- 项目创建完成后，点击「**方案管理**」进入方案管理页面，在这里您可以为您的项目创建不同的方案，如下图所示。

> 若您的场景为APP场景（安卓/IOS系统），『场景方案』请选择APP实名认证方案；
>
> 若您的场景为微信、H5页面，『场景方案』请选择H5实名认证方案。



4.1 身份信息录入

- 身份信息录入支持选择**用户手动输入**或**OCR拍照采集**，如下图所示

> **OCR拍照采集**：会在之后生成的APP集成文件内集成OCR采集SDK，在本地进行身份证质量校验，并判断是否为身份证证件。（**OCR拍照采集**支持实时采集和相册上传两种形式。推荐您使用实时采集，可以在一定程度增加方案的安全性及便捷性。）
>
> **手动输入**：支持用户手动输入姓名+身份证号信息。

4.2 离线采集SDK配置

- **填写授权标识：**选择SDK的授权标识信息。

> 若您还未申请授权标识信息，请点击**『新建授权』，并填写相关信息**进行申请，申请过程如下图所示。
>
> 
>

- **图像质量控制（本地）：**分为严格、正常、宽松三个等级，等级越严格，对采集图片的角度、模糊度、遮挡等信息参数把控越高，推荐使用正常。

> 此项配置为离线采集SDK端对采集图片的质量要求，推荐实名认证场景选择严格或正常模式。图片质量越好，云端接口传输的通过率越高。

- **活体检测设置（本地）：**离线采集 SDK在前端要求用户做出指定动作，并检测动作的完成情况。在该过程，SDK 会随机抓

取几帧图像进行本地活体检测，检测通过后将图片传至后台进行下一步检测。

> 此项配置为离线采集SDK端的活体检测动作，可对动作数量及顺序进行配置。推荐采用随机顺序进行三个以上动作的校验。

附录：

质量控制参数	「宽松」	「正常」	「严格」
光照最小值	30	40	60
光照最大值	240	220	200
遮挡-左眼	0.95	0.8	0.4
遮挡-右眼	0.95	0.8	0.4
遮挡-鼻子	0.95	0.8	0.4
遮挡-嘴巴	0.95	0.8	0.4
遮挡-左脸	0.95	0.8	0.4
遮挡-右脸	0.95	0.8	0.4
遮挡-下巴	0.95	0.8	0.4
姿态-俯仰角	30	20	15
姿态-左右角	18	18	15
姿态-旋转角	30	20	15
模糊度	0.8	0.6	0.4

4.3 人脸实名认证API配置

- **人脸实名认证接口：**标准级实名认证方案默认接入 **人脸实名认证 API接口**。
- **图像质量检测：**分为正常与宽松两个等级，等级设置越严格，对图片角度、模糊度、遮挡等信息参数把控越高，**推荐使用宽松。**
- **活体检测：**分为严格、正常、宽松三个等级，不同等级对应不同的活体检测阈值。等级设置越严格，对活体检测相关参数信息的把控越高。不同等级对应指标可参考下表，**推荐使用正常。**

> **活体检测阈值：**活体检测得分高于此阈值，即判断为活体
>
> **误拒率（FRR）：**指误将活体用户判断为非活体的概率。如误拒率为0.5%，指1000次真人请求，会有5次因为活体分数低于阈值被错误拒绝。

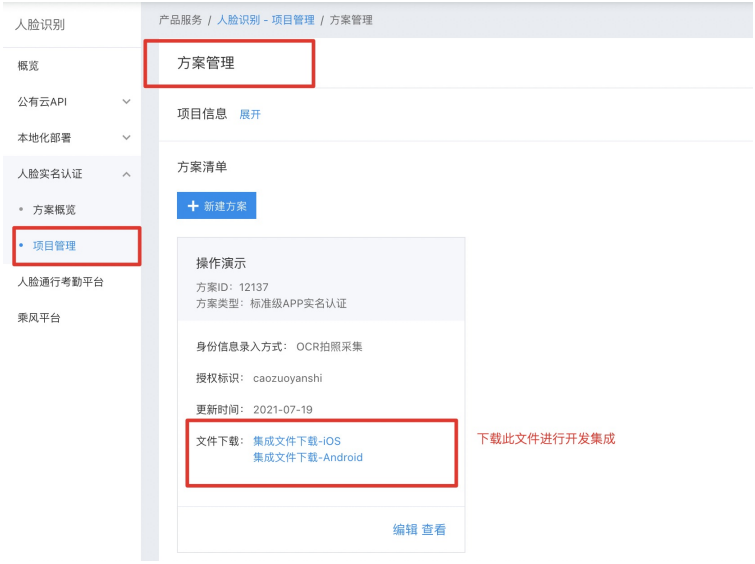
控制度	对应阈值	说明
宽松	0.05	万分之一活体误拒率
正常（推荐）	0.3	千分之一活体误拒率
严格	0.9	百分之一活体误拒率

- **阈值：**用户人脸图片与公安权威数据源中人脸的相似度得分阈值，得分超过此阈值，即被判断为同一人。阈值分数相关指标可参考下表，**推荐阈值为80。**

阈值分数	误识率	识别率
60	0.781615%	99.550128%
70	0.096534%	98.307626%
78	**0.015570%（万分之一）**	**95.672664%**
80 （推荐）	**0.009342%（低于万分之一）**	**94.323051%**

Step5：提交方案，获取示例工程

完成上述方案配置后，点击『提交』，进入方案管理页面，下载**IOS/安卓版集成文件（含示例工程）**进行SDK端集成使用。



> Step4中方案配置的参数会自动生成至集成文件（含示例工程）中，方便开发使用。 注意：请谨慎修改APP方案流程，修改后需要重新下载集成文件进行使用。

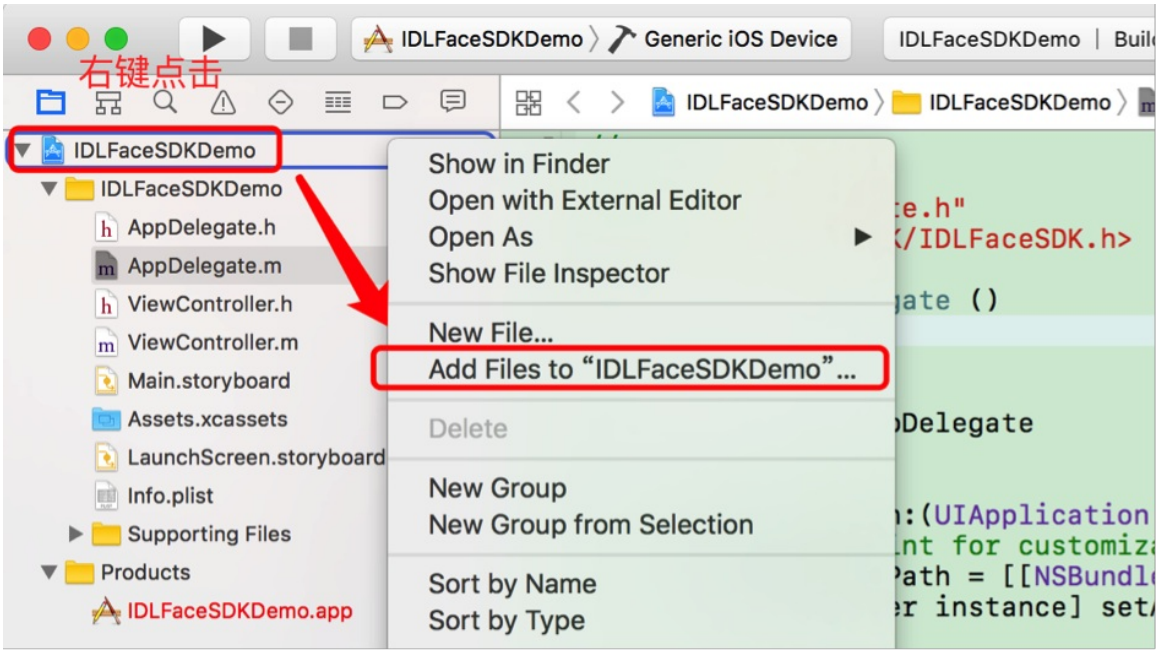
具体开发操作请参考以下步骤。

二、 集成逻辑

2.1 IOS集成

1、打开或者新建一个项目。

2、右键点击项目，会出现一个添加菜单，在菜单中选择『Add Files to“此处是你的项目名字”…… 』,如下图所示：

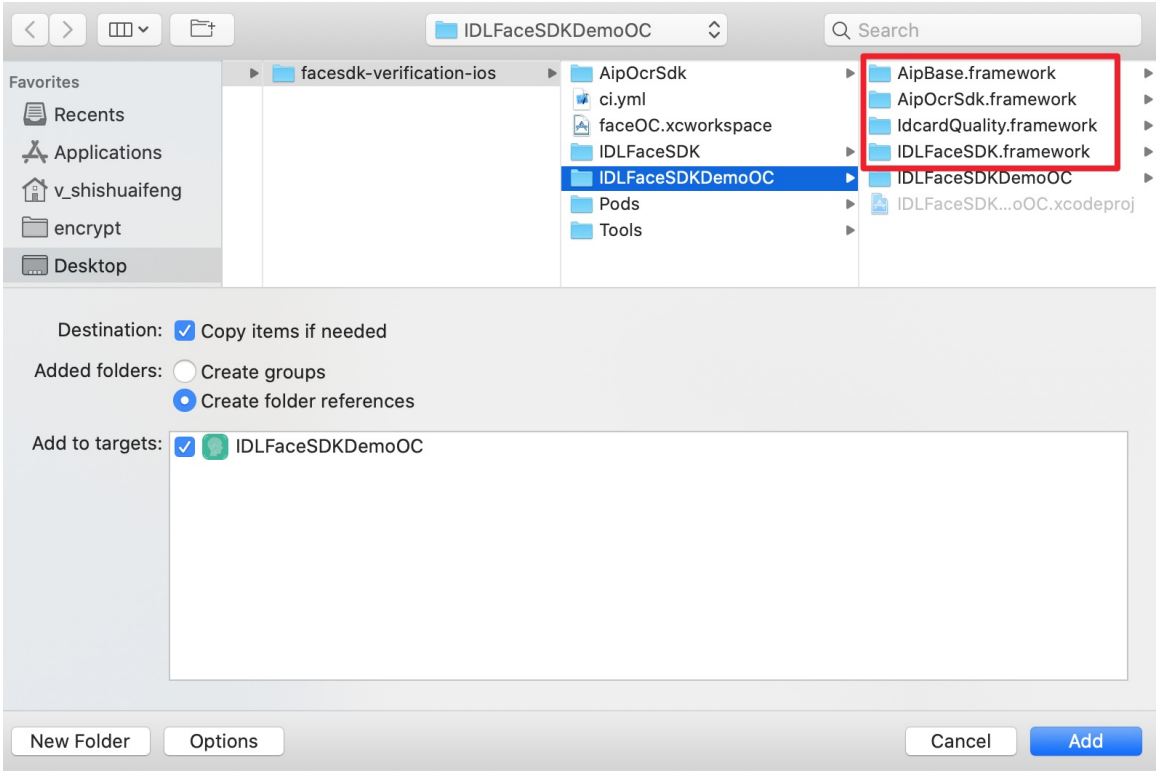


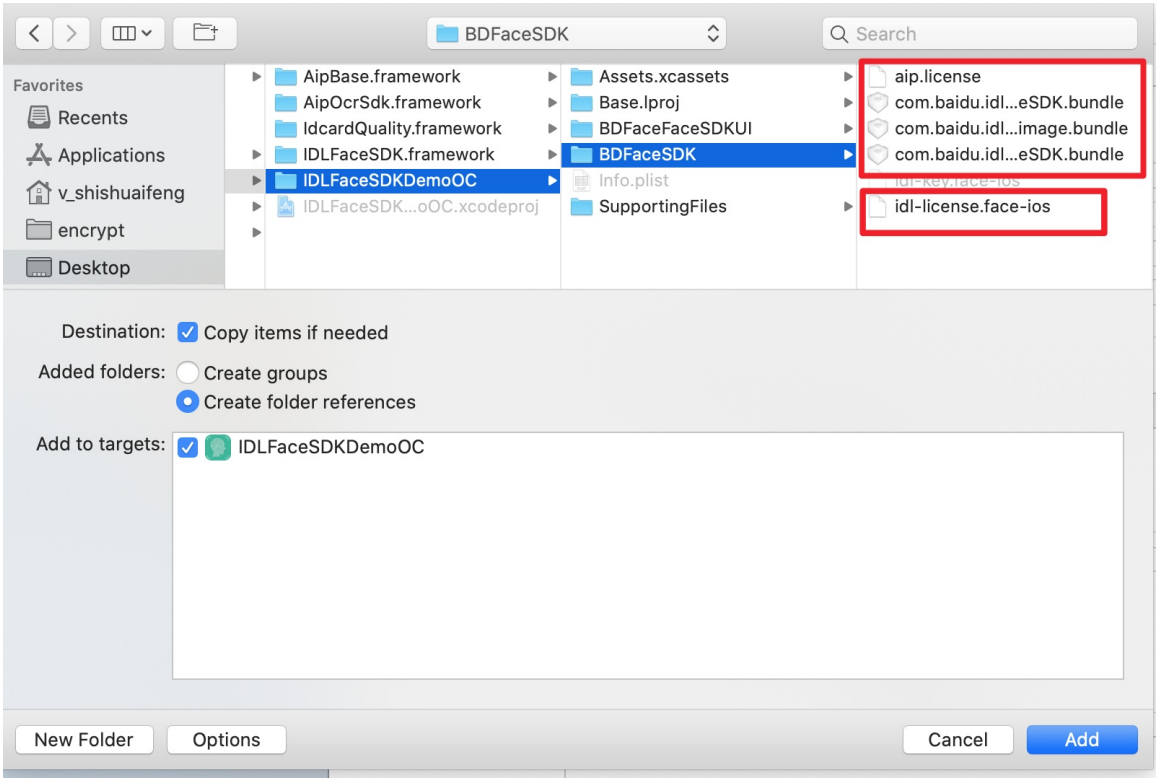
3、在添加文件弹出框里面选择申请到的license和SDK添加进来。如下图：

> 注意：license为百度官方提供，刚才在后台下载的文件（文件名称：idl-license.face-ios）

SDK包含下面几个文件：

- IDLFaceSDK.framework
- com.baidu.idl.face.faceSDK.bundle
- com.baidu.idl.face.model.bundle
- com.baidu.idl.face.live.action.image.bundle
- AipOcrSdk.framework
- AipBase.framework
- IdcardQuality.framework





4、确认下 Bundle Identifier 是否是申请license时填报的那一个.

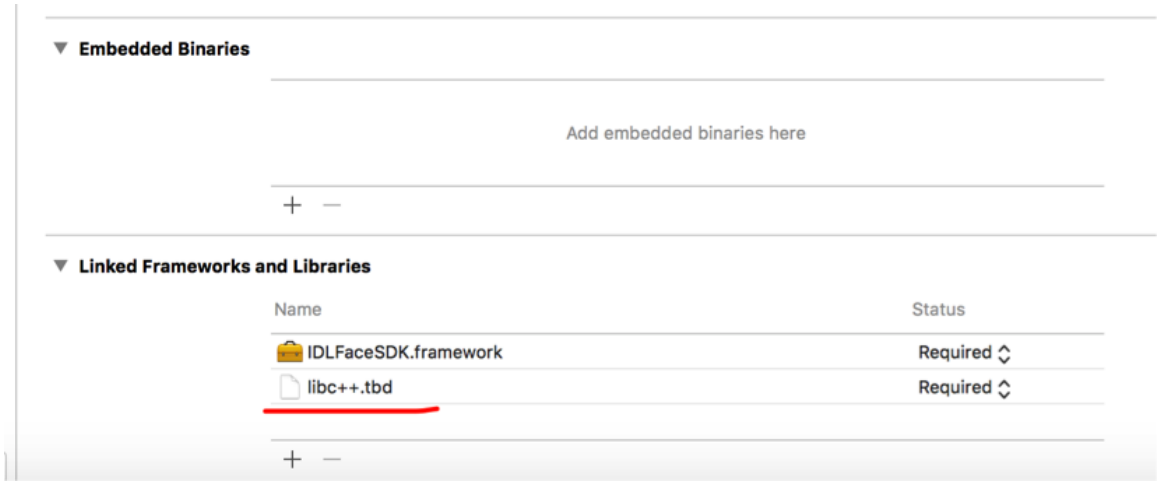
5、注意：license 和 Bundle Identifier 为一一对应关系，填错了会导致SDK不可用.

6、 FACE_LICENSE_ID这个参数填写百度官方给的LicenseID

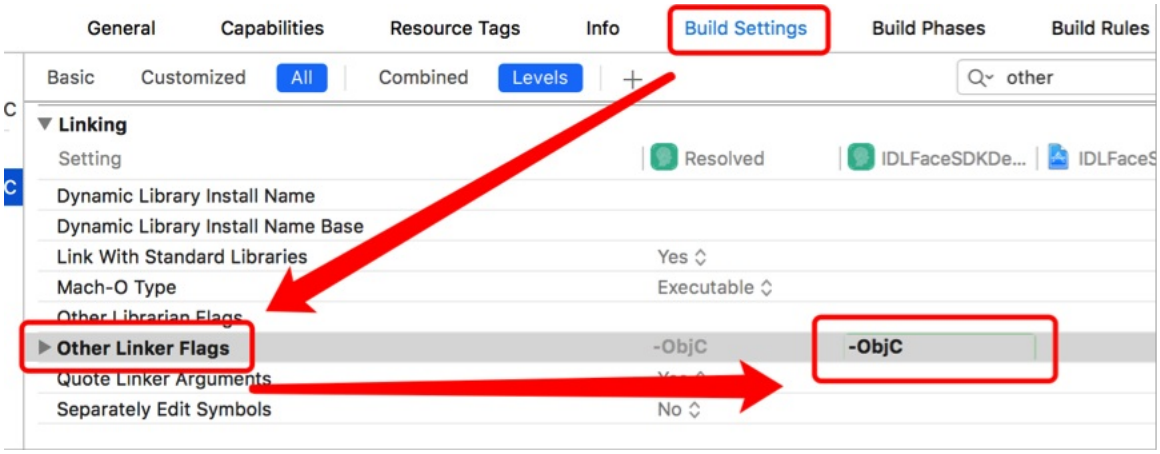
在 FaceParameterConfig.h 文件中填写下面几项。

```
// 人脸license文件名
#define FACE_LICENSE_NAME    @"idl-license"

// 人脸license后缀
#define FACE_LICENSE_SUFFIX  @"face-ios"

//  (您申请的应用名称(appname)+「-face-ios」后缀，如申请的应用名称(appname)为test123，则此处填写test123-face-ios)
// 在后台 -> 产品服务 -> 人脸识别 -> 客户端SDK管理查看，如果没有的话就新建一个
#define FACE_LICENSE_ID        @"vis-var-license-face-ios"

// 配置参数文件名和后缀
#define FACE_CONF_NAME    @"custom"
#define FACE_CONF_SUFFIX  @"conf"

// OCR license文件名
#define FACE_API_ORC_KEY @"aip"
// OCR license后缀
#define FACE_SECRET_OCR_KEY @"license"
```

7、选择链接C++标准库。



8、如果没有使用pod管理第三方库的话，请**在Build Setting Linking Other Linker Flags 上面加入 –ObjC选项。如果用了pod请忽略，因为pod会自动添加上。**



以下为示例工程调用身份验证的代码片段：

2.1.1 OCR身份证识别集成

把下载下来的License文件（文件名：aip.license），添加到项目里面，无需更改文件名称，然后在AppDelegate添加以下代码引用进去。

```
NSString *licenseFile = [[NSBundle mainBundle] pathForResource:FACE_API_ORC_KEY ofType:FACE_SECRET_OCR_KEY];
NSData *licenseFileData = [NSData dataWithContentsOfFile:licenseFile];
[[AipOcrService shardService] authWithLicenseFileData:licenseFileData];
```

在FaceParameterConfig.h里面设置下载下来的License文件的名字和后缀。

```
// OCR license文件名
#define FACE_API_ORC_KEY @"aip"
// OCR license后缀
#define FACE_SECRET_OCR_KEY @"license"
```

通过API调用ViewController进行身份证扫描

```
    // 身份证识别
    UIViewController *vc =
    [AipCaptureCardVC ViewControllerWithCardType:CardTypeLocalIdCardFont
                                andImageHandler:^(UIImage *image) {
        _idCardImage = image;
        [[AipOcrService shardService] detectIdCardFrontFromImage:image
                                                      withOptions:nil
                                                   successHandler:^(id result){
            _successHandler(result);
        }
                                                      failHandler:_failHandler];
    }];
    [weakSelf presentViewController:vc animated:YES completion:nil];
```

详细调用文档，请参考 OCR-iOS-SDK文档

2.1.2 人脸SDK集成

2.1.2.1 授权初始化

把下载下来的License文件（文件名：idl.license.face-ios），添加到项目里面，无需更改文件名称，然后在AppDelegate添加以下代码引用进去。

```
    NSString* licensePath = [NSString stringWithFormat:@"%@.%@", FACE_LICENSE_NAME, FACE_LICENSE_SUFFIX ];
    [[FaceSDKManager sharedInstance] setLicenseID:FACE_LICENSE_ID andLocalLicenceFile:licensePath andRemoteAuthorize:false];
    NSLog(@"canWork = %d",[[FaceSDKManager sharedInstance] canWork]);
    NSLog(@"version = %@",[[FaceSDKManager sharedInstance] getVersion]);
```

在FaceParameterConfig.h里面设置下载下来的License文件的名字和后缀。

2.1.2.2 接口参数及调用说明

```
// 人脸license文件名
#define FACE_LICENSE_NAME    @"idl-license"

// 人脸license后缀
#define FACE_LICENSE_SUFFIX  @"face-ios"

//  (您申请的应用名称(appname)+「-face-ios」后缀，如申请的应用名称(appname)为test123，则此处填写test123-face-ios)
// 在后台 -> 产品服务 -> 人脸识别 -> 客户端SDK管理查看，如果没有的话就新建一个
#define FACE_LICENSE_ID        @"vis-var-license-face-ios"
```

（1）动作采集

调用IDLFaceLivenessManager类的:

```
 /*带黑边的方法-通过图片质量控制
- (void)livenessStratrgyWithImage:(UIImage *)image previewRect:(CGRect)previewRect detectRect:(CGRect)detectRect
completionHandler:(LivenessStrategyCompletion)completion;
/*不带黑边*/
-(void) livenessNormalWithImage:(UIImage *)image previewRect:(CGRect)previewRect detectRect:(CGRect)detectRect
completionHandler:(LivenessNormalCompletion)completion;
```

参数说明

- previewRect 人脸图片大小，类型：Rect
- detectRect 人脸检测区域大小，类型：Rect
- completion 完成后返回照片和状态结果

说明

检测图片中的人脸信息，完成人脸图像采集和人脸活体检测，返回检测状态和结果。

使用方法如下

```
- (void)faceProcesss:(UIImage *)image {
    if (self.hasFinished) {
        return;
    }
    __weak typeof(self) weakSelf = self;
    [[IDLFaceLivenessManager sharedInstance] livenessStratrgyWithImage:image
        previewRect:self.previewRect detectRect:self.detectRect completionHandler:^(NSDictionary
        *images, LivenessRemindCode remindCode) {
        switch (remindCode) {
            case LivenessRemindCodeOK: {
                weakSelf.hasFinished = YES;
                [self warningStatus:CommonStatus warning:@"非常好"];
                if (images[@"bestImage"] != nil && [images[@"bestImage"] count] != 0) {

                    NSData* data = [[NSData alloc] initWithBase64EncodedString:[images[@"bestImage"]
                        lastObject] options:NSDataBase64DecodingIgnoreUnknownCharacters];
                    UIImage* bestImage = [UIImage imageWithData:data];
                    NSLog(@"bestImage = %@",bestImage);
                }
```

图片	key
抠图	bestImage
原图	originalImage
按要求数量的抠图	crop%d（%d为第几张，从0开始）
按要求数量的原图	original%d（%d为第几张，从0开始）

（2）人脸采集 调用IDLFaceDetectionManager类的:

```
 /*带黑边的方法-通过图片质量控制*/
- (void)detectStratrgyWithQualityControlImage:(UIImage *)image previewRect:(CGRect)previewRect detectRect:
(CGRect)detectRect completionHandler:(DetectStrategyCompletion)completion;
/*不带黑边*/
- (void)detectStratrgyWithNormalImage:(UIImage _)image previewRect:(CGRect)previewRect detectRect:(CGRect)detectRect
completionHandler:(DetectStrategyCompletion)completion;
```

参数说明

previewRect与detctRect是为了做距离检测而定义的，为了在上层封装判断脸是否在框内／离太远／离太近。

- image：相机获取的图片
- previewRect：间接定义的最大距离的 maxRect 和最小距离的 minRect。
- detectRect：实际采集区域
- completion：回调

说明

检测图片中的人脸信息，完成人脸图像采集和人脸活体检测，返回检测状态和结果。

使用方法如下

```
- (void)faceProcesss:(UIImage *)image {
    if (self.hasFinished) {
        return;
    }

    __weak typeof(self) weakSelf = self;
    [[IDLFaceDetectionManager sharedInstance] detectStratrgyWithImage:image
        previewRect:self.previewRect detectRect:self.detectRect completionHandler:^(NSDictionary
        *images, DetectRemindCode remindCode) {
        switch (remindCode) {
            case DetectRemindCodeOK: {
                weakSelf.hasFinished = YES;
                [self warningStatus:CommonStatus warning:@"非常好"];
                if (images[@"bestImage"] != nil && [images[@"bestImage"] count] != 0) {
                    NSData* data = [[NSData alloc] initWithBase64EncodedString:[images[@"bestImage"]
                        lastObject] options:NSDataBase64DecodingIgnoreUnknownCharacters];
                    UIImage* bestImage = [UIImage imageWithData:data];
                    NSLog(@"bestImage = %@",bestImage);
                }
                dispatch_async(dispatch_get_main_queue(), ^{
                    [UIView animateWithDuration:0.5 animations:^{
                        weakSelf.animaView.alpha = 1;
```

| 图片 |

key | | ---- | ---------- | | 原图 | originalImage | | 抠图 | bestImage | **（3）设置人脸功能控制参数**

具体方法详见如下：

```
  // 设置鉴权
- (void)setLicenseID:(NSString *)licenseID andLocalLicenceFile:(NSString *)licensePath andRemoteAuthorize:
(BOOL)remoteAuthorize;

// 最小检测人脸阈值，默认40
- (void)setMinFaceSize:(int)width;

// 截取人脸图片大小，默认400
- (void)setCropFaceSizeWidth:(CGFloat)width;

// 人脸检测精度阈值，默认0.5f
- (void)setNotFaceThreshold:(CGFloat)th;

// 人脸遮挡阈值，默认0.5
- (void)setOccluThreshold:(CGFloat)thr;

// 亮度阈值，默认90
- (void)setIllumThreshold:(CGFloat)thr;

// 图像模糊阈值，默认0.5
- (void)setBlurThreshold:(CGFloat)thr;

// 头部姿态角度，默认12，10，10
- (void)setEulurAngleThrPitch:(float)pitch yaw:(float)yaw roll:(float)roll;

//设置无黑边抠图的数量，默认1
- (void)setMaxCropImageNum:(int)imageNum;

//设置最大人脸检测数量，默认1
- (void)setMaxDetectNum:(int)detectNum;

//设置有黑边抠图方式的旋转角度，默认2.0f
- (void)setCropEnlargeRatio:(float)cropEnlargeRatio;

// 检测时间，默认10
- (void)setConditionTimeout:(CGFloat)timeout;
```

调用方法：

```
// 设置最小检测人脸阈值
[[FaceSDKManager sharedInstance] setMinFaceSize:200];

// 设置截取人脸图片大小
[[FaceSDKManager sharedInstance] setCropFaceSizeWidth:400];

// 设置人脸遮挡阀值
[[FaceSDKManager sharedInstance] setOccluThreshold:0.5];

// 设置亮度阀值
[[FaceSDKManager sharedInstance] setIllumThreshold:40];

// 设置图像模糊阀值
[[FaceSDKManager sharedInstance] setBlurThreshold:0.7];

// 设置头部姿态角度
[[FaceSDKManager sharedInstance] setEulurAngleThrPitch:10 yaw:10 roll:10];

// 设置超时时间
[[FaceSDKManager sharedInstance] setConditionTimeout:10];

// 设置人脸检测精度阀值
[[FaceSDKManager sharedInstance] setNotFaceThreshold:0.6];

// 设置照片采集张数
[[FaceSDKManager sharedInstance] setMaxCropImageNum:1];

// 设置抠图的旋转角度
[[FaceSDKManager sharedInstance] setCropEnlargeRatio:2.0];
```

（4）检测功能初始化

```
- (int)initCollect;
```

调用方法：

```
[[FaceSDKManager sharedInstance] initCollect];
```

（5）检测功能释放

```
- (int)uninitCollect
```

参数:

- 无

返回：

- 无

（6）活体动作设置

```
- (void)livenesswithList:(NSArray *)array order:(BOOL)ordernumberOfLiveness:(NSInteger)numberOfLiveness
```

参数:

- array: 活体动作列表

- order: 是否按顺序进行活体动作
- numberOfLiveness: 活体动作数目（array为nil是起作用）

返回：

- 无

说明：

- 活体动作设置

2.2 安卓集成

2.2.1 OCR身份证识别集成

- 把申请的license（aip.license）放到到项目中app module下的assets目录中
- 修改app的build.gradle包名为申请时填入的包名

```
android{}  defaultConfig{}
apply plugin: 'com.android.application'

android {
    compileSdkVersion 25
    buildToolsVersion "25.0.2"
    defaultConfig {
        applicationId "申请时的包名"
        minSdkVersion 15
        targetSdkVersion 25
        versionCode 1000
        versionName "3.2.1.0"

    }
```

- 拷贝ocr-ui到您的工程中

接下来调用具体请看示例工程

1、在首页面初始化OCR SDK（在HomeActivity.java中）

```
private void initOCRSDK() {
    OCR.getInstance().initAccessToken(new OnResultListener<AccessToken>() {
        @Override
        public void onResult(AccessToken result) {
            // 调用成功，返回AccessToken对象
            String token = result.getAccessToken();
        }
        @Override
        public void onError(OCRError error) {
            // 调用失败，返回OCRError子类SDKError对象
        }
    }, getApplicationContext());
}
```

2、跳转至身份证扫描页面进行身份证扫描（在HomeActivity.java中）

```
/**
 * 开始进入身份证认证页面
 */
private void startIdCardVerify() {
    if (mConsoleConfig.getUseOcr() == 1) {
        Intent intent = new Intent(mContext, CameraExpActivity.class);
        // 设置临时存储
        intent.putExtra(CameraActivity.KEY_OUTPUT_FILE_PATH,
                FileUtil.getSaveFile(getApplication()).getAbsolutePath());
        // 调用拍摄身份证正面的activity
        intent.putExtra(CameraActivity.KEY_CONTENT_TYPE, CameraActivity.CONTENT_TYPE_ID_CARD_FRONT);
        intent.putExtra(CameraActivity.KEY_NATIVE_TOKEN, OCR.getInstance(this).getLicense());
        intent.putExtra(CameraActivity.KEY_NATIVE_ENABLE,  value: true);
        startActivity(intent);
    } else {
        Intent intent = new Intent(mContext, IdCardInputActivity.class);
        startActivity(intent);
    }
}
```

3、调用身份证识别请求接口api进行身份证识别（在IdCardRecognizeActivity.java中） 其中请求参数如下：

请求参数	说明
image	身份证扫描的图像
id_card_side	身份证的正反面类型
detect_risk	是否开启旋转识别

```
IDCardParams param = new IDCardParams();
param.setImageFile(new File(filePath));
param.setIdCardSide(idCardSide);
param.setDetectDirection(true);
OCR.getInstance().recognizeIDCard(param, new OnResultListener<IDCardResult>() {
    @Override
    public void onResult(IDCardResult result) {
        if (result != null) {
            Word idnumberWord = result.getIdNumber();
            Word nameWord = result.getName();
            if (idnumberWord != null) {
                idnumber = idnumberWord.getWords();
            }
            if (nameWord != null) {
                username = nameWord.getWords();
            }

            alertText("识别结果", "idnumber->" + idnumber + " name->" + username);
        }
        displayTip("");
    }

    @Override
    public void onError(OCRError error) {
        alertText("识别结果", error.getMessage());
        displayTip("");
    }
});
```

4、获得身份证号码和姓名后进入人脸活体验证步骤

详细调用文档，请参考 OCR-Android-SDK文档

2.2.2 人脸SDK集成

1、授权参数

（1）把申请的license（idl-license.face-android"）放到到项目中app module下的assets目录中

（2）修改 Config.java 类中的参数

> 在人脸实名认证控制台创建完APP方案后，会自动生成LicenseID和LicenseFileName信息，这里您只需要填写apiKey和secretkey信息即可。

```
public class Config {
  // 为了apiKey,secretKey为您调用百度人脸在线接口的，如注册，比对等。
// 为了的安全，建议放在您的服务端，端把人脸传给服务器，在服务端端
// license为调用sdk的人脸检测功能使用，人脸识别 = 人脸检测（SDK功能）+ 人脸比对（服务端API）
  public static String apiKey = "替换为你的apiKey(ak)";
  public static String secretKey = "替换为你的secretKey(sk)";
  public static String licenseID = "替换为你的licenseID,后台SDK管理界面中，已经生成的licenseID,如:test-face-android";
  public static String licenseFileName = "替换为你的license文件";
}
```

（3）配置签名（**申请license时的md5为打包签名的文件，所以必须用申请license的签名文件**）

```
 app->build.gradle->android->signingConfigs
signingConfigs {
      def password = "替换为签名密码"
      def alias = "替换为签名别名"
      def filePath = "替换为签名文件路径" //如 ../facesharp.jks//签名文件路径
      debug {
         keyAlias alias
         keyPassword password
         storeFile file(filePath)
         storePassword(password)
      }
      release {
         keyAlias alias
         keyPassword password
         storeFile file(filePath)
         storePassword(password)
      }
  }
```

（4）修改包名 app->build.gradle->android->defaultConfig ->applicationId您申请license时填的包名

```
android{} defaultConfig{}
apply plugin: 'com.android.application'

android {
    compileSdkVersion 25
    buildToolsVersion "25.0.2"
    defaultConfig {
        applicationId "申请时的包名"
        minSdkVersion 15
        targetSdkVersion 25
        versionCode 1000
        versionName "3.2.1.0"

    }
```

2、**初始化人脸相关SDK（在**HomeActivity.java**中）

```
private void initLicense() {
  setFaceConfig();
  // 为了android和ios 区分授权，appId=appname_face_android ,其中appname为申请sdk时的应用名
  // 应用上下文
  // 申请License取得的APPID
  // assets目录下License文件名
  FaceSDKManager.getInstance().initialize(mContext,
  Config.licenseID, Config.licenseFileName, new IInitCallback() {
        @Override
        public void initSuccess() {
        // 初始化OCR SDK 使用的license是aip.license，名字不能修改
            initOCRSDK();
        }

        @Override
        public void initFailure(final int errCode,
        final String errMsg) {
            runOnUiThread(new Runnable() {
              @Override
              public void run() {
                Log.e(TAG, "初始化失败 = " + errCode + " " + errMsg);
                showToast("初始化失败 = " + errCode + ", " + errMsg);
                mIsInitSuccess = false;
              }
            });
        }
    });
  }
```

3、设置配置参数，如不设置，将使用默认值（在HomeActivity.java中）

```
private void setFaceConfig() {
    FaceConfig config = FaceSDKManager.getInstance().getFaceConfig();
    mConsoleConfig = ConsoleConfigManager.getInstance(mContext).getConfig();
    // -----TODO：以下为通过console平台获取到的配置信息-----
    // 设置模糊度阈值
    config.setBlurnessValue(mConsoleConfig.getBlur());
    // 设置光照阈值（范围0-255）
    config.setBrightnessValue(mConsoleConfig.getIllumination());
    // 设置遮挡阈值
    config.setOcclusionValue(mConsoleConfig.getOcclusion());
    // 设置活体动作，通过设置list，LivenessTypeEunm.Eye, LivenessTypeEunm.Mouth,
    // LivenessTypeEunm.HeadUp, LivenessTypeEunm.HeadDown, LivenessTypeEunm.HeadLeft,
    // LivenessTypeEunm.HeadRight, LivenessTypeEunm.HeadLeftOrRight
    config.setLivenessTypeList(mConsoleConfig.getActions());
    // 设置动作活体是否随机
    config.setLivenessRandom(mConsoleConfig.isFaceVerifyRandom());
    // 风控加密类型，0：普通版；1：加密版
    config.setSecType(mConsoleConfig.getSecType());

    // -----TODO：以下不需要通过console平台配置，需要手动修改-----
    // 设置可检测的最小人脸阈值
    config.setMinFaceSize(FaceEnvironment.VALUE_MIN_FACE_SIZE);
    // 设置可检测到人脸的阈值
    config.setNotFaceValue(FaceEnvironment.VALUE_NOT_FACE_THRESHOLD);
    // 设置人脸姿态角阈值
    config.setHeadPitchValue(FaceEnvironment.VALUE_HEAD_PITCH);
    config.setHeadYawValue(FaceEnvironment.VALUE_HEAD_YAW);
    // 设置闭眼阈值
    config.setEyeClosedValue(FaceEnvironment.VALUE_CLOSE_EYES);
    // 设置图片缓存数量（非动作活体使用）
    config.setCacheImageNum(FaceEnvironment.VALUE_CACHE_IMAGE_NUM);
    // 设置口罩判断开关以及口罩阈值
    config.setOpenMask(FaceEnvironment.VALUE_OPEN_MASK);
    config.setMaskValue(FaceEnvironment.VALUE_MASK_THRESHOLD);
    // 设置开启提示音
    config.setSound(ExampleApplication.isOpenSound);
    // 原图缩放系数
    config.setScale(FaceEnvironment.VALUE_SCALE);
    // 抠图高的设定，为了保证好的抠图效果，我们要求高宽比是4：3，所以会在内部进行计算，只需要传入高即可
    config.setCropHeight(FaceEnvironment.VALUE_CROP_HEIGHT);
    // 抠图人脸框与背景比例
    config.setEnlargeRatio(FaceEnvironment.VALUE_CROP_ENLARGERATIO);
    // 选择针对人脸采集输出图片的类型进行加密，0：原图，1：抠图
    config.setOutputImageType(FaceEnvironment.VALUE_OUTPUT_IMAGE_TYPE);
    FaceSDKManager.getInstance().setFaceConfig(config);
}
```

4、开始进行人脸采集 **非动作活体版：**（在**FaceDetectActivity.java**中） （1）在摄像头的回调方法onPreviewFrame(byte[] data, Camera camera)中调用**FaceSDKManager.getInstance().getDetectStrategyModule()**获得IDetectStrategy对象。（该方法每次调用都会返回一个新对象，建议只调用一次）。

（2）调用**IDetectStrategy.setPreviewDegree(int degree);**设置预览图片的旋转角度。 调用**IDetectStrategy.setDetectStrategySoundEnable(boolean flag);**设置是否开启语音。 调用**IDetectStrategy.setDetectStrategyConfig(Rect previewRect, Rect detectRect, IDetectStrategyCallback callback);**设置预览框的大小，人脸检测框的坐标和回调。

（3）多次调用**detectStrategy(byte[] imageData);**进行人脸图片采集。

（4）实现**IDetectStrategyCallback.onDetectCompletion(FaceStatusNewEnum status, String message, HashMap<String, ImageInfo> base64ImageCropMap, HashMap<String, ImageInfo> base64ImageSrcMap);**回调方法并处理结果。其中

base64ImageCropMap为采集到的指定数量的抠图，base64ImageSrcMap为采集到的指定数量的原图。

（5）获取采集到的最优抠图和原图，即在上述第（4）步的**IDetectStrategyCallback.onDetectCompletion();**回调方法中调用**getBestImage(HashMap<String, ImageInfo> imageCropMap, HashMap<String, ImageInfo> imageSrcMap);**，

其实现如下：

```
private void getBestImage(HashMap<String, ImageInfo> imageCropMap, HashMap<String, ImageInfo> imageSrcMap) {
    String cropBmpStr = null;
    String cropSecBmpStr = null;
    String srcBmpStr = null;
    String srcSecBmpStr = null;
    // 将抠图集合中的图片按照质量降序排序，最终选取质量最优的一张抠图图片
    if (imageCropMap != null && imageCropMap.size() > 0) {
        List<Map.Entry<String, ImageInfo>> list1 = new ArrayList<>(imageCropMap.entrySet());
        Collections.sort(list1, (Comparator) (o1, o2) -> {
                String[] key1 = o1.getKey().split( regex: "_");
                String score1 = key1[2];
                String[] key2 = o2.getKey().split( regex: "_");
                String score2 = key2[2];
                // 降序排序
                return Float.valueOf(score2).compareTo(Float.valueOf(score1));
        });
        cropBmpStr = list1.get(0).getValue().getBase64();
        cropSecBmpStr = list1.get(0).getValue().getSecBase64();
    }
    // 将原图集合中的图片按照质量降序排序，最终选取质量最优的一张原图图片
    if (imageSrcMap != null && imageSrcMap.size() > 0) {
        List<Map.Entry<String, ImageInfo>> list2 = new ArrayList<>(imageSrcMap.entrySet());
        Collections.sort(list2, (Comparator) (o1, o2) -> {
                String[] key1 = o1.getKey().split( regex: "_");
                String score1 = key1[2];
                String[] key2 = o2.getKey().split( regex: "_");
                String score2 = key2[2];
                // 降序排序
                return Float.valueOf(score2).compareTo(Float.valueOf(score1));
        });
        srcBmpStr = list2.get(0).getValue().getBase64();
        srcSecBmpStr = list2.get(0).getValue().getSecBase64();
    }

    intentActivity(cropBmpStr, cropSecBmpStr, srcBmpStr, srcSecBmpStr);

}
```

动作活体版：（在**FaceLivenessActivity.java**中） （1）在摄像头的回调方法onPreviewFrame(byte[] data, Camera camera)中调用**FaceSDKManager.getInstance().getLivenessStrategyModule(ILivenessViewCallback)**获得ILivenessStrategy对象。（该方法每次调用都会返回一个新对象，建议只调用一次）。

（2）调用**ILivenessStrategy.setPreviewDegree(int degree);**设置预览图片的旋转角度。 调用**ILivenessStrategy.setLivenessStrategySoundEnable(boolean flag);**设置是否开启语音。 调用**ILivenessStrategy.setLivenessStrategyConfig(List<LivenessTypeEnum> livenessList, Rect previewRect, Rect detectRect, ILivenessStrategyCallback callback);**设置动作活体类型、预览框的大小，人脸检测框的坐标和回调。

（3）多次调用**livenessStrategy(byte[] imageData);**进行人脸图片采集。

（4）实现**ILivenessStrategyCallback.onDetectCompletion(FaceStatusNewEnum status, String message, HashMap<String, ImageInfo> base64ImageCropMap, HashMap<String, ImageInfo> base64ImageSrcMap, int currentLivenessCount);**回调方法并处理结果。其中base64ImageCropMap为采集到的指定数量的抠图，base64ImageSrcMap为采集到的指定数量的原图。

（5）获取采集到的最优抠图和原图，即在上述第（4）步的**ILivenessStrategyCallback.onDetectCompletion();**回调方法中调

用getBestImage(HashMap<String, ImageInfo> imageCropMap, HashMap<String, ImageInfo> imageSrcMap);，

其实现如下：

```
private void getBestImage(HashMap<String, ImageInfo> imageCropMap, HashMap<String, ImageInfo> imageSrcMap) {
    String cropBmpStr = null;
    String cropSecBmpStr = null;
    String srcBmpStr = null;
    String srcSecBmpStr = null;
    // 将抠图集合中的图片按照质量降序排序，最终选取质量最优的一张抠图图片
    if (imageCropMap != null && imageCropMap.size() > 0) {
        List<Map.Entry<String, ImageInfo>> list1 = new ArrayList<>(imageCropMap.entrySet());
        Collections.sort(list1, (Comparator) (o1, o2) -> {
                String[] key1 = o1.getKey().split( regex: "_");
                String score1 = key1[2];
                String[] key2 = o2.getKey().split( regex: "_");
                String score2 = key2[2];
                // 降序排序
                return Float.valueOf(score2).compareTo(Float.valueOf(score1));
        });
        cropBmpStr = list1.get(0).getValue().getBase64();
        cropSecBmpStr = list1.get(0).getValue().getSecBase64();
    }

    // 将原图集合中的图片按照质量降序排序，最终选取质量最优的一张原图图片
    if (imageSrcMap != null && imageSrcMap.size() > 0) {
        List<Map.Entry<String, ImageInfo>> list2 = new ArrayList<>(imageSrcMap.entrySet());
        Collections.sort(list2, (Comparator) (o1, o2) -> {
                String[] key1 = o1.getKey().split( regex: "_");
                String score1 = key1[2];
                String[] key2 = o2.getKey().split( regex: "_");
                String score2 = key2[2];
                // 降序排序
                return Float.valueOf(score2).compareTo(Float.valueOf(score1));
        });
        srcBmpStr = list2.get(0).getValue().getBase64();
        srcSecBmpStr = list2.get(0).getValue().getSecBase64();
    }

    intentActivity(cropBmpStr, cropSecBmpStr, srcBmpStr, srcSecBmpStr);

}
```

5、跳转至身份核验接口调用页面，需要传入：图片的base64、图片加密类型（加密、不加密）、姓名、身份证号（在FaceDetectExpActivity.java和FaceLivenessExpActivity.java中）

```
/**
 * 进行页面跳转，往新页面需要传入：图片的base64、图片加密类型（加密、不加密）、姓名、身份证号
 * @param cropBmpStr      抠图不加密的base64
 * @param cropSecBmpStr   抠图加密的base64
 * @param srcBmpStr       原图不加密的base64
 * @param srcSecBmpStr    原图加密的base64
 */
private void intentActivity(String cropBmpStr, String cropSecBmpStr, String srcBmpStr, String srcSecBmpStr) {
    Intent intent = new Intent(FaceDetectExpActivity.this, CollectVerifyActivity.class);
    int outputImageType = mFaceConfig.getOutputImageType();
    int secType = mFaceConfig.getSecType();
    // 将采集成功的图片以base64的方式传入到认证页面进行风控请求认证，因图片base64过大，Intent传值
    // 极易造成崩溃，所以采用IntentUtil方式传值

    // 抠图
    if (outputImageType == 1) {
        if (secType == 0) {    // 不加密
            IntentUtil.getInstance().setBase64Img(cropBmpStr);
        } else if (secType == 1) {   // 加密
            IntentUtil.getInstance().setSecBase64Img(cropSecBmpStr);
        }
        // 原图
    } else if (outputImageType == 0) {
        if (secType == 0) {   // 不加密
            IntentUtil.getInstance().setBase64Img(srcBmpStr);
        } else if (secType == 1) {   // 加密
            IntentUtil.getInstance().setSecBase64Img(srcSecBmpStr);
        }
    }
    intent.putExtra("secType", secType);
    intent.putExtra("username", mUsername);
    intent.putExtra("idNumber", mIdCardNum);
    startActivity(intent);
}
```

6、获取accessToken，用于调用接口（在CollectVerifyActivity.java中）

> 注意：为了防止ak、sk泄露，建议把ak、sk放在服务端，移动端通过服务端去拉取token（目前是在工程中的Config.java中）

```
private void getAccessTokenFromServer(final int secType, final String base64Img) {
    APIService.getInstance().init();
    APIService.getInstance().initAccessTokenWithAkSk(new OnResultListener<AccessToken>() {
        @Override
        public void onResult(AccessToken result) {
            if (result != null && result.getAccessToken() != null) {
                // 调用风控接口
                policeVerifyFromServer(base64Img, secType, getApplicationContext());
            }
        }

        @Override
        public void onError(final FaceException error) {
            runOnUiThread(new Runnable() {
                @Override
                public void run() {
                    if (error == null) {
                        return;
                    }

                    // 网络加载失败
                    if (error.getErrorCode() == FaceException.ErrorCode.NETWORK_REQUEST_ERROR) {
                        cancelAnim();
                        mImageAnim.setImageResource(R.mipmap.icon_verify_network);
                        mTextVerifyIng.setVisibility(View.GONE);
                        mTextCheckNet.setVisibility(View.VISIBLE);
                        mTextNetError.setVisibility(View.VISIBLE);
                        mBtnReTry.setVisibility(View.VISIBLE);
                        mBtnReturnHome.setVisibility(View.VISIBLE);
                    }
                }
            });
        }
    }, Config.apiKey, Config.secretKey);
}
```

7、根据离线采集得到的人脸图片（抠图或原图），进行人脸实名认证（在CollectVerifyActivity.java中）

> **提示**：为了安全及维护成本考虑，我们建议您将SDK获取的人脸图像，推送到服务器端，由服务器端进行API调用，并将结果返回给客户端APP
>
> 若业务对安全性要求较高，推荐升级为具备**安全加密**和**大数据风控**功能的增强级人脸实名认证方案或金融级人脸实名认证方案。

其中请求参数如下：

请求参数	含义	说明
image_type	图片类型	BASE64:图片的base64值，base64编码后的图片数据，编码后的图片大小不超过2M；图片尺寸不超过1920*1080
image	图片信息	图片信息(总数据大小应小于10M)，图片上传方式根据image_type来判断
name	姓名	需要是UTF-8编码的中文
id_card_number	身份证号	
quality_control	图片质量控制	NONE: 不进行控制 LOW：较低的质量要求 NORMAL: 一般的质量要求 HIGH: 较高的质量要求 默认 NONE
liveness_control	活体检测控制	NONE: 不进行控制 LOW：较低的活体要求(高通过率 低攻击拒绝率) NORMAL: 一般的活体要求(平衡的攻击拒绝率, 通过率) HIGH: 较高的活体要求(高攻击拒绝率 低通过率) 默认 NONE

```
private void policeVerifyFromServer(String base64Img, int secType, Context context) {
    if (TextUtils.isEmpty(base64Img)) {
        return;
    }

    final ConsoleConfig consoleConfig = ConsoleConfigManager.getInstance(getApplicationContext()).getConfig();
    DynamicParams params = new DynamicParams();
    params.setImgType("BASE64");
    params.setBase64Img(base64Img);
    try {
        params.setUsername(URLDecoder.decode(mUsername, "UTF-8"));
    } catch (UnsupportedEncodingException e) {
        params.setUsername(mUsername);
        e.printStackTrace();
    }
    params.setIdCardNum(mIdCardNum);
    params.setQualityControl(consoleConfig.getOnlineImageQuality());
    params.setLivenessControl(consoleConfig.getOnlineLivenessQuality());
    params.setSpoofingControl("NONE");
    // 以下参数只为加密版接口使用
    if (secType == 1) {
        params.setRiskIdentify(consoleConfig.isOpenRiskIdentify());
        // 开启风控
        if (consoleConfig.isOpenRiskIdentify()) {
            params.setZid(FaceSDKManager.getInstance().getZid(context, 5001));
            params.setIp(IPUtil.getLocalIpAddress(getApplicationContext()));
            // 手机号，需要自己手动修改
            // params.setPhone("");
        }
        params.setImageSec(true);
        params.setApp("Android");
        params.setEv("smrz");
    }
```

```
APIService.getInstance().policeVerify(params, secType, new OnResultListener<LivenessVsIdcardResult>() {
    @Override
    public void onResult(LivenessVsIdcardResult result) {
        if (result == null || consoleConfig == null) {
            return;
        }
        if (result.getScore() > consoleConfig.getRiskScore()) {
            Intent intent = new Intent(CollectVerifyActivity.this,
                    CollectionSuccessExpActivity.class);
            intent.putExtra("destroyType", "CollectVerifyActivity");
            startActivity(intent);
            finish();
        }
    }

    @Override
    public void onError(final FaceException error) {
        runOnUiThread(new Runnable() {
            @Override
            public void run() {
                if (error == null) {
                    return;
                }

                // 失败处理

            }
        });
    }
});
}
```

增强级APP端实名认证方案

方案简介

方案简介

推出背景

- 现在，人脸识别技术被广泛应用在金融支付、用户注册、人脸登录等业务场景中。技术的进步方便用户的同时，黑灰产产业也开始对这些场景产生觊觎。并通过**屏幕攻击、照片、纸张、以及面具、头模**等方式进行非法攻击。随着黑产技术的进步，更是出现了通过**自动化脚本直接攻击云端API、ROM注入、视频劫持替换、批量虚拟机、病毒侵入**等新型攻击手段。使现有的人脸识别方案面临着巨大的安全挑战。
- 为提升人脸识别的安全性，保障客户的业务安全，人脸实名认证产品团队与百度安全实验室联合推出**增强级人脸实名认证方案**，在人脸登录、注册等环境加入层层保障，为您的业务保驾护航。

功能简介

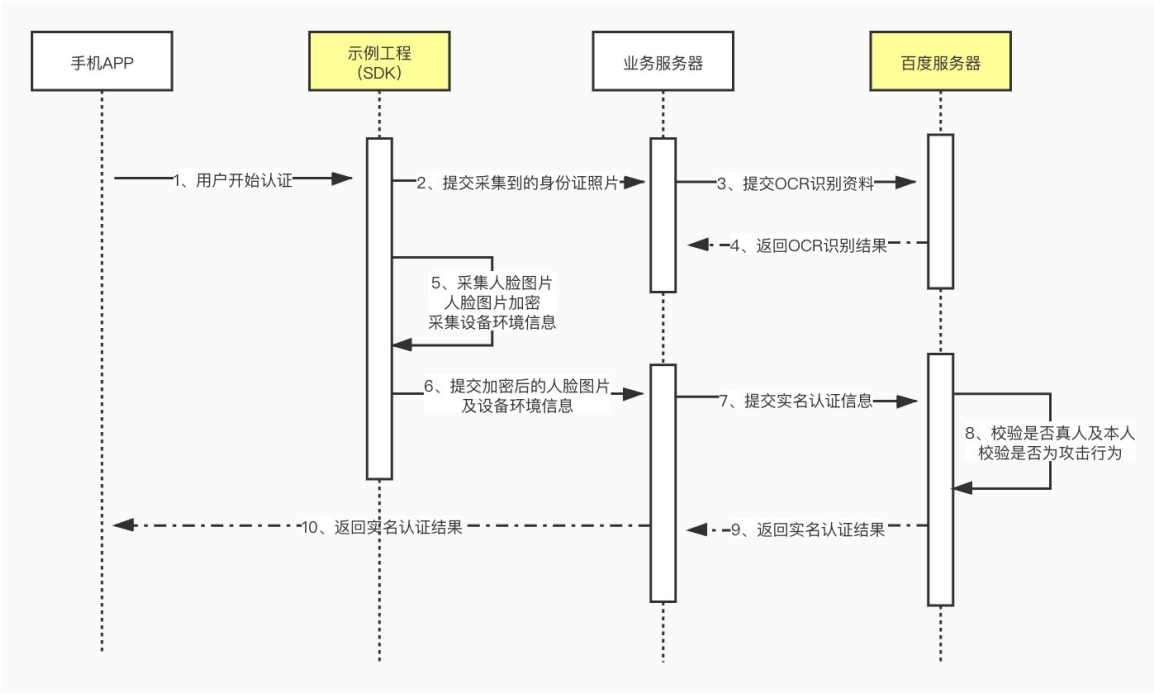
- 增强级APP实名认证方案提供标准化的人身核验流程，具有**人脸比对、证件识别、活体检测**等多项组合能力，以及端云配合高防攻击拦截能力，可抵挡屏幕、照片、视频、换脸、面具、3D模型的攻击。同时，增强级实名认证方案中加入**安全加密能力**以及**大数据风控能力**，针对脚本攻击、ROM注入、视频劫持、批量虚拟机、病毒侵入等新型攻击手段进行强力有效防御。

> **安全加密能力：**（增强级方案中**默认开启**此项功能，无需您自行配置）增强级APP方案集成文件中的采集SDK会对输出的图片进行加密，在云端**增强级实名认证API**进行解密。此端云配合的加密方式是百度专门针对市面黑产绕过采集SDK，攻击云端接口的攻击方式进行的功能升级。
>
> **大数据风控能力：**增强级API接口接受SDK端传入的设备指纹信息，基于百度海量大数据设备因子，对SDK端进行设备风险识别，辨别是否为风险设备，返回识别结果。可有效防御黑产批量虚拟机、病毒侵入等攻击手段，降低第三方黑产攻破概率，提升业务安全性。

适用场景

- 增强级APP实名认证方案适用于**安卓/IOS系统的APP场景**中实现用户实名认证。如果您的业务场景是在微信小程序、公众号、H5等业务场景，推荐采用 增强级H5实名认证方案。

接入时序图



方案接入步骤

- 增强级APP实名认证方案的具体接入步骤请参考方案接入指南

方案集成指南

本文档介绍了增强级APP实名认证方案配置流程，以及APP集成开发流程。 一、 准备工作

在正式集成前，需要做一些准备工作，完成一些账号、应用及配置，具体如下：

Step1: 注册成为开发者

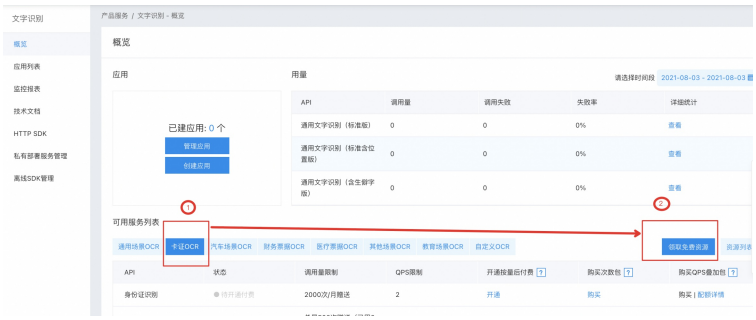
> 在使用百度人脸实名认证方案之前，首先需注册百度云账号，账号注册方式请参考账号注册指南。
>
> 百度云账号注册完成以后，为顺利调用百度AI能力，需完成企业认证。具体认证方式请参考企业认证指南。

Step2：创建应用

2.1 输入应用名称，领取免费额度

- 创建好账号后，在正式调用AI 能力之前，需首先创建应用，应用是调用服务的基础能力单元。 同时领取接口所需的**免费调用额度**，用于接入测试。如下图所示：

- 除人脸服务接口的免费调用额度外，还需领取**身份证识别**接口的**免费调用额度**，用来调用身份证OCR识别功能（必须领取，否则会报错服务异常），点击此处，按下图所示进行领取。



- 如您之前已经领取过免费额度，无需重复领取，请跳至下一步骤。

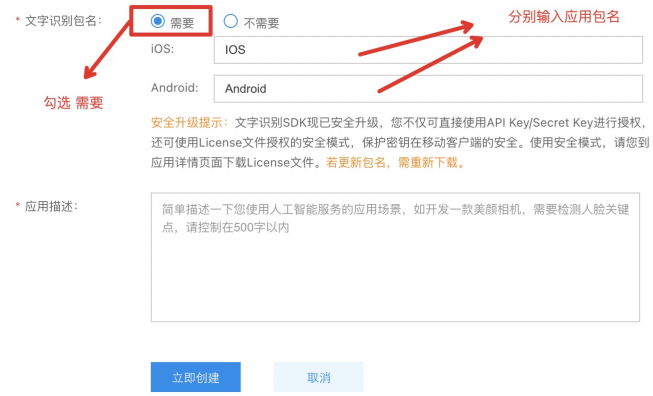
2.2 勾选所需接口

- **人脸识别服务相关接口已默认勾选且不可取消。**

- **注：**「接口选择」过程中，还**需勾选「文字识别」中的「身份证识别」接口**，用于实现身份核验流程中的身份证识别功能。如下图所示。

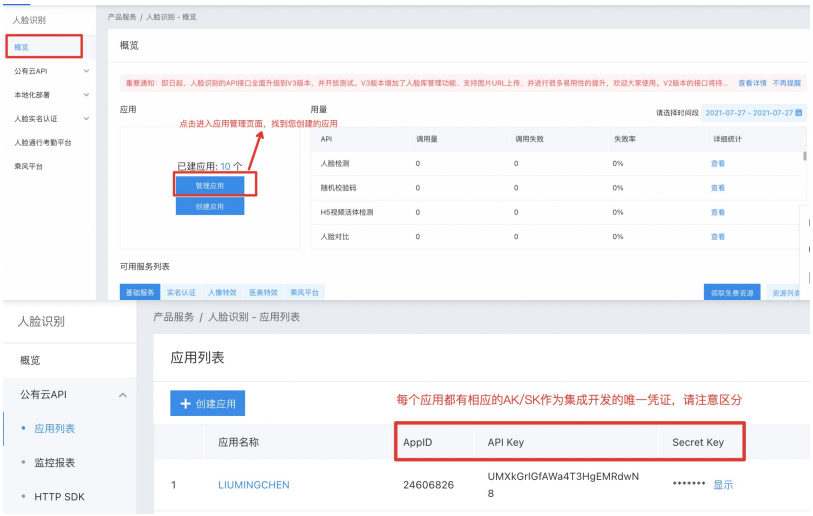
2.3 输入应用包名

- **在「文字识别包名」处选择「需要」**，并根据您的APP应用信息填写包名。此处为必要操作，否则将无法顺利下载集成文件。**至此应用创建完成。**



2.4 获取密钥信息（AK/SK）

完成应用创建后，平台将会分配给您此应用的相关凭证，主要为**AppID、API Key、Secret Key**，以上三个信息是您应用实际开发的主要凭证，每个应用之间各不相同，请您妥善保管。您可在控制台的**应用管理页面**找到以上信息。如下图所示



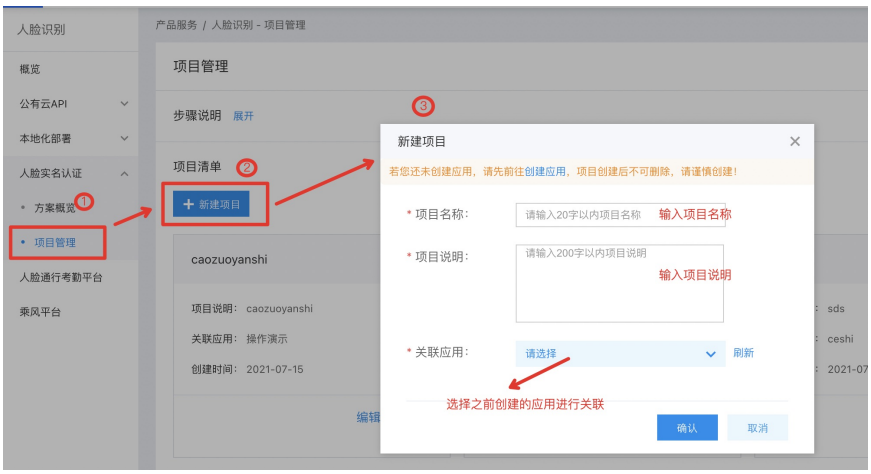
该AK/SK用于调用在线API 如：身份验证。**在之后下载的集成文件（示例工程）中需要填写正确的AK/SK以顺利集成。**

注：开发中请注意区分多份AK/SK（API Key、Secret Key），若填写的AK/SK与开发的应用不对应，会产生鉴权错误。

Step3：创建项目

- 进入控制台-人脸实名认证页面，选择『**项目管理**』页面，点击『**新建项目**』，进行项目创建，如下图所示。

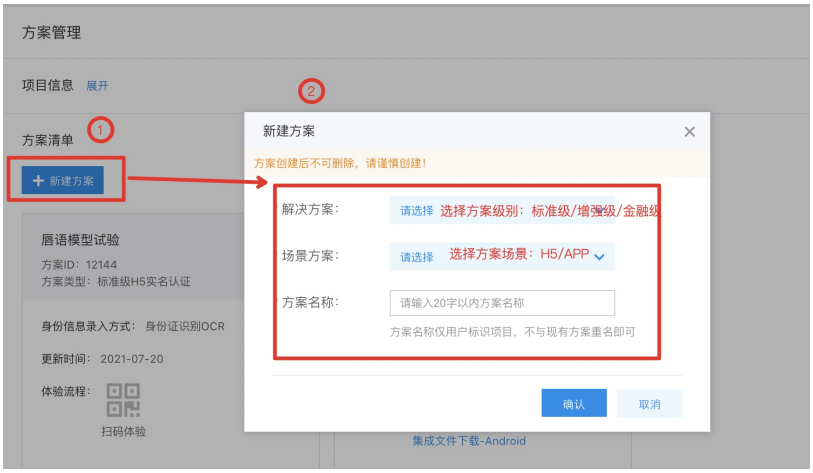
> 创建项目前，请确保您在应用控制台已创建应用，若您未创建应用，请参考STEP2创建应用后，再进行项目创建.



Step4：创建方案

- 项目创建完成后，点击「**方案管理**」进入方案管理页面，在这里您可以为您的项目创建不同的方案，如下图所示。

> 若您的场景为APP场景（安卓/IOS系统），『场景方案』请选择APP实名认证方案；
>
> 若您的场景为微信、H5页面，『场景方案』请选择H5实名认证方案。



4.1 身份信息录入

- 身份信息录入支持选择**用户手动输入**或**OCR拍照采集**，如下图所示

> **OCR拍照采集**：会在之后生成的APP集成文件内集成OCR采集SDK，在本地进行身份证质量校验，并判断是否为身份证证件。（**OCR拍照采集**支持实时采集和相册上传两种形式。推荐您使用实时采集，可以在一定程度增加方案的安全性及便捷性。）
>
> **手动输入**：支持用户手动输入姓名+身份证号信息。

4.2 离线采集SDK配置

- **填写授权标识：**选择SDK的授权标识信息。

> 若您还未申请授权标识信息，请点击**『新建授权』，并填写相关信息**进行申请，申请过程如下图所示。
>
> 
>

- **图像质量控制（本地）：**分为严格、正常、宽松三个等级，等级越严格，对采集图片的角度、模糊度、遮挡等信息参数把控越高，推荐使用正常。

> 此项配置为离线采集SDK端对采集图片的质量要求，推荐实名认证场景选择严格或正常模式。图片质量越好，云端接口传输的通过率越高。

- **活体检测设置（本地）：**离线采集 SDK在前端要求用户做出指定动作，并检测动作的完成情况。在该过程，SDK 会随机抓

取几帧图像进行本地活体检测，检测通过后将图片传至后台进行下一步检测。

> 此项配置为离线采集SDK端的活体检测动作，可对动作数量及顺序进行配置。推荐采用随机顺序进行三个以上动作的校验。

附录：

质量控制参数	「宽松」	「正常」	「严格」
光照最小值	30	40	60
光照最大值	240	220	200
遮挡-左眼	0.95	0.8	0.4
遮挡-右眼	0.95	0.8	0.4
遮挡-鼻子	0.95	0.8	0.4
遮挡-嘴巴	0.95	0.8	0.4
遮挡-左脸	0.95	0.8	0.4
遮挡-右脸	0.95	0.8	0.4
遮挡-下巴	0.95	0.8	0.4
姿态-俯仰角	30	20	15
姿态-左右角	18	18	15
姿态-旋转角	30	20	15
模糊度	0.8	0.6	0.4

4.3 人脸实名认证API配置

- **人脸实名认证接口：**增强级实名认证方案默认接入 **增强级人脸实名认证API**。在增强级APP方案中SDK会自动发起请求，但您需要在百度云控制台**创建人脸识别应用**，勾选增强级人脸实名认证API并**开通付费**。
- **安全加密能力：**（增强级方案中**默认开启**此项功能，无需您自行配置）增强级APP方案集成文件中的采集SDK会对输出的图片进行加密，在云端**增强级实名认证API**进行解密。此端云配合的加密方式是百度专门针对市面黑产绕过采集SDK，攻击云端接口的攻击方式进行的功能升级。
- **大数据风控：**大数据风控功能开启后，接受SDK端传入的设备指纹信息，基于百度海量大数据设备因子，对SDK端进行设备风险识别，辨别是否为风险设备，返回识别结果。可有效防御黑产批量虚拟机、病毒侵入等攻击手段，降低第三方黑产攻破概率，提升业务安全性。
- **图像质量检测：**分为正常与宽松两个等级，等级设置越严格，对图片角度、模糊度、遮挡等信息参数把控越高，**推荐使用宽松。**
- **活体检测：**分为严格、正常、宽松三个等级，不同等级对应不同的活体检测阈值。等级设置越严格，对活体检测相关参数信

息的把控越高。不同等级对应指标可参考下表，**推荐使用正常。**

> **活体检测阈值：**活体检测得分高于此阈值，即判断为活体
>
> **误拒率（FRR）：**指误将活体用户判断为非活体的概率。如误拒率为0.5%，指1000次真人请求，会有5次因为活体分数低于阈值被错误拒绝。

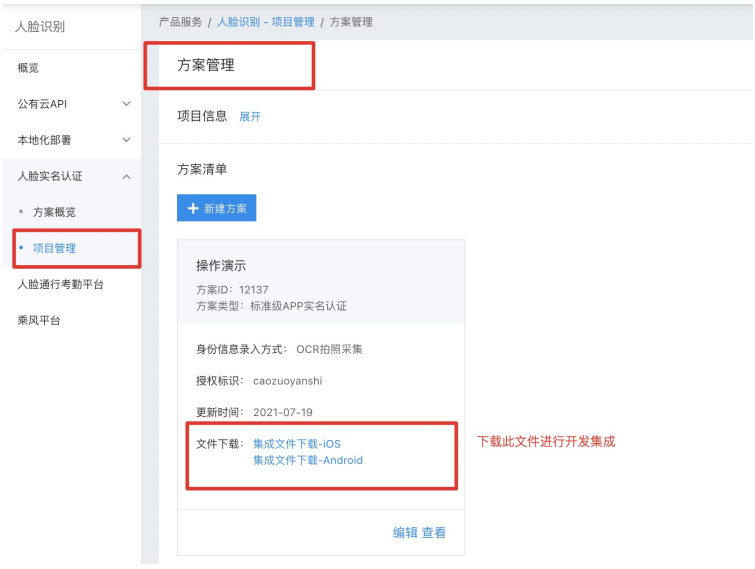
控制度	对应阈值	说明
宽松	0.05	万分之一活体误拒率
正常（推荐）	0.3	千分之一活体误拒率
严格	0.9	百分之一活体误拒率

- **阈值：**用户人脸图片与公安权威数据源中人脸的相似度得分阈值，得分超过此阈值，即被判断为同一人。阈值分数相关指标可参考下表，**推荐阈值为80。**

阈值分数	误识率	识别率
60	0.781615%	99.550128%
70	0.096534%	98.307626%
78	**0.015570%（万分之一）**	**95.672664%**
80 （推荐）	**0.009342%（低于万分之一）**	**94.323051%**

Step5：提交方案，获取示例工程

完成上述方案配置后，点击『提交』，进入方案管理页面，下载**IOS/安卓版集成文件（含示例工程）**进行SDK端集成使用。



> **Step4**中方案配置的参数会自动生成至集成文件（含示例工程）中，方便开发使用。 注意：请谨慎修改APP方案流程，修改后需要重新下载集成文件进行使用。

具体开发操作请参考以下步骤。

二、 集成逻辑

2.1 iOS集成

2.1.1 运行Demo 在真机上运行Demo代码并确认Demo本身可以正确进行人证核验

Step 1： 打开上述步骤中下载的集成文件demo，如下图所示:

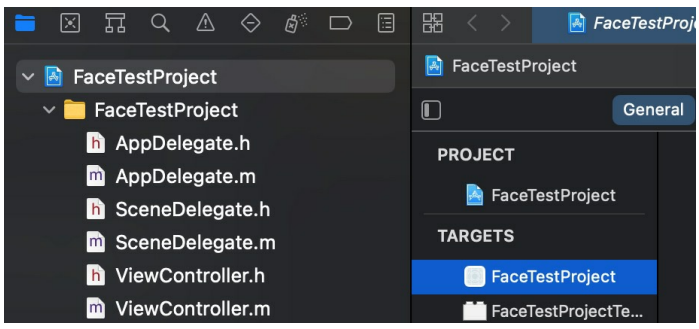
Step 2： 确认demo中使用的bundleIdetifier是否为最终要使用的bundleIdetifier, 如果不对，则需要重新阅读上述文档，保证bundleIdetifier的正确性，也就是建方案的时候的iOS的包名，需要为自己工程的bundleIdetifer.

Step 3： 修改：FaceParameterConfig.h文件中的FACE_API_KEY 和 FACE_SECRET_KEY

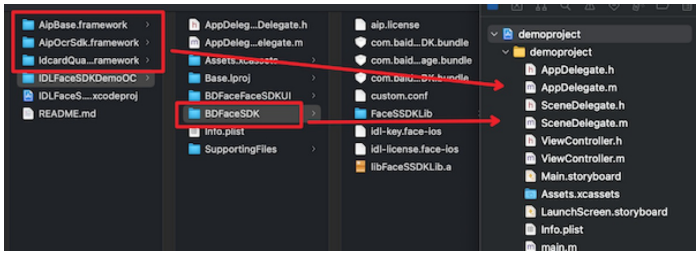
Step 4： 真机运行demo代码，测试Demo

Step 5： 一定要确认demo是可以正常进行人脸检测的，即进行身份证识别和人脸检测之后，真机可以运行，并可以人证核验成功，如下图所示：

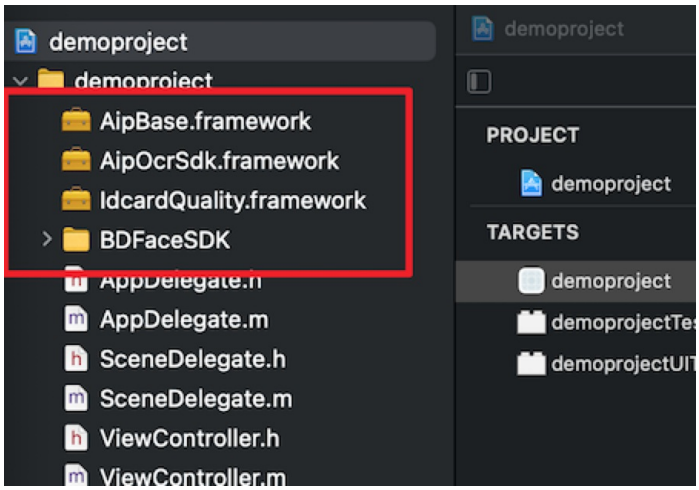
2.1.2 集成至目标工程 Step 1： 新建工程，如下图所示：



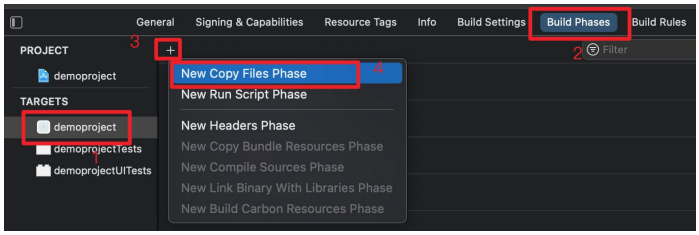
Step 2： 将上述demo中下载的BDFaceSDK文件夹和 AipOcrSdk.framework、AipBase.framework、IdcardQuality.frame三个framework拖动至目标工程中，如下图所示：



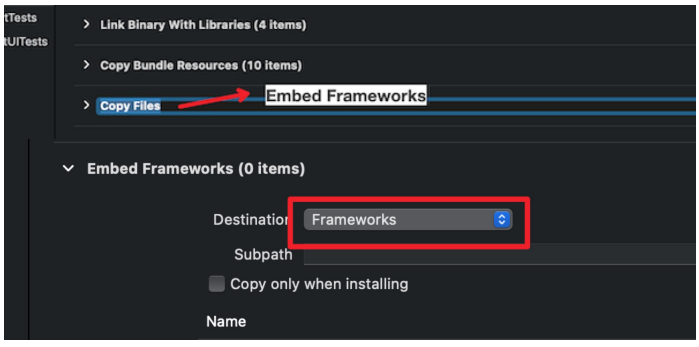
移动完成后，如下所示：



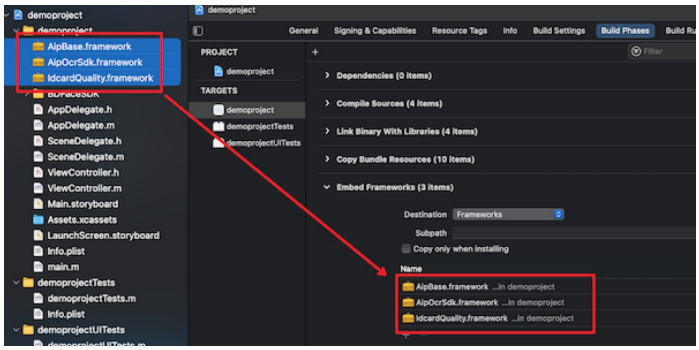
Step 3： 选择targets中的BuildPhases，点击下图所示的+号，然后选择New Copy File Phase，如下图所示



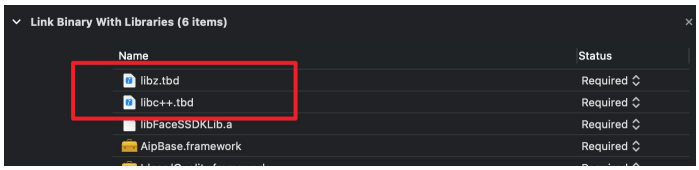
Step 4： 双击Copy Files，将Copy Files改名为Embed Frameworks， 然后点击Copy Files前面的>号打开该项，修改Destination为Frameworks，如下图所示



Step 5： 将工程中的AipOcrSdk.framework，AipBase.framework，IdcardQuality.framework三个framework，拖入Embed Frameworks中，如下图所示：



Step 6： 添加依赖库 libz.tbd 和 libc++.tbd ，如下图所示：



Step 7： 确认目标工程的bundleIdetifier和下载的示例工程的是一致的，同时也和申请授权标识license时填写的iOS包名是一致的。

Step 8： 真机可顺利运行代码。

2.1.3 代码说明 2.1.3.1 OCR鉴权代码

```
 NSString *licenseFile = [[NSBundle mainBundle] pathForResource:FACE_API_ORC_KEY ofType:FACE_SECRET_OCR_KEY];
NSData *licenseFileData = [NSData dataWithContentsOfFile:licenseFile];
[[AipOcrService shardService] authWithLicenseFileData:licenseFileData];`
```

2.1.3.2 人脸SDK使用鉴权

```
 [[SSFaceSDKManager sharedInstance] setBCEClientId:FACE_API_KEY clientSecret:FACE_SECRET_KEY];
NSString* licensePath = [NSString stringWithFormat:@"%@.%@", FACE_LICENSE_NAME, FACE_LICENSE_SUFFIX ];
[[SSFaceSDKManager sharedInstance] setLicenseID:FACE_LICENSE_ID andLocalLicenceFile:licensePath
andRemoteAuthorize:YES];
NSLog(@"canWork = %d",[[SSFaceSDKManager sharedInstance] canWork]);
NSLog(@"version = %@",[[SSFaceSDKManager sharedInstance] getVersion]);`
```

2.1.3.3 获取token代码

```
``[self getAccessTokenWithAK:FACE_API_KEY SK:FACE_SECRET_KEY];``
```

2.1.3.4 初始化函数`[[BDFaceActionLiveConfig sharedInstance] initSDK];`，

函数内容如下:

```
 AppDelegate *appDelegate = (AppDelegate*) [[UIApplication sharedApplication] delegate];
if (![[SSFaceSDKManager sharedInstance] canWork]){
    NSLog(@"授权失败，请检测ID 和 授权文件是否可用");
    return;
}
// 设置人脸遮挡阈值
[[SSFaceSDKManager sharedInstance] setOccluThreshold:0.5];
// 设置亮度阈值
[[SSFaceSDKManager sharedInstance] setMinIllumThreshold:40];
[[SSFaceSDKManager sharedInstance] setMaxIllumThreshold:240];
// 设置图像模糊阈值
[[SSFaceSDKManager sharedInstance] setBlurThreshold:0.3];
// 初始化SDK配置参数，可使用默认配置
// 设置最小检测人脸阈值
[[SSFaceSDKManager sharedInstance] setMinFaceSize:200];
// 设置截取人脸图片高
[[SSFaceSDKManager sharedInstance] setCropFaceSizeWidth:400];
// 设置截取人脸图片宽
[[SSFaceSDKManager sharedInstance] setCropFaceSizeHeight:640];
// 设置头部姿态角度
[[SSFaceSDKManager sharedInstance] setEulurAngleThrPitch:10 yaw:10 roll:10];
// 设置人脸检测精度阈值
[[SSFaceSDKManager sharedInstance] setNotFaceThreshold:0.6];
// 设置抠图的缩放倍数
[[SSFaceSDKManager sharedInstance] setCropEnlargeRatio:2.5];
// 设置照片采集张数
[[SSFaceSDKManager sharedInstance] setMaxCropImageNum:3];
// 设置超时时间
[[SSFaceSDKManager sharedInstance] setConditionTimeout:15];
// 设置开启口罩检测，非动作活体检测可以采集戴口罩图片
[[SSFaceSDKManager sharedInstance] setIsCheckMouthMask:true];
// 设置开启口罩检测情况下，非动作活体检测口罩过滤阈值，默认0.8 不需要修改
[[SSFaceSDKManager sharedInstance] setMouthMaskThreshold:0.8f];
// 设置原始图缩放比例
[[SSFaceSDKManager sharedInstance] setImageWithScale:0.8f];
// 设置图片加密类型，type=0 基于base64 加密；type=1 基于百度安全算法加密
[[SSFaceSDKManager sharedInstance] setImageEncrypteType: 1];
// 初始化SDK功能函数

// 设置人脸过远框比例
[[SSFaceSDKManager sharedInstance] setMinRect:0.4];

// 设置图像阈值
[BDFaceAdjustParamsTool changeConfig:appDelegate.faceImageParams];
[[BDFaceActionLiveConfig sharedInstance] initActionLive];
[[SSFaceSDKManager sharedInstance] initCollect];`
```

2.1.3.5 设置动作函数

```
 [BDFaceLivingConfigModel.sharedInstance.liveActionArray addObject:@(FaceLivenessActionTypeLiveEye)];
[BDFaceLivingConfigModel.sharedInstance.liveActionArray addObject:@(FaceLivenessActionTypeLiveMouth)];
[BDFaceLivingConfigModel.sharedInstance.liveActionArray addObject:@(FaceLivenessActionTypeLiveYawRight)];
BDFaceLivingConfigModel.sharedInstance.isByOrder = NO; // 动作是否是随机顺序
BDFaceLivingConfigModel.sharedInstance.numOfLiveness = 3;`
```

2.1.3.6 初始化SDK函数

```
 [[SSFaceSDKManager sharedInstance] initCollect];`
```

2.1.3.7 活体检测状态的回调函数

```
- (void)livenessActionDidFinishWithCode:(LivenessRemindCode)code;`
```

2.1.3.8 人脸流程回调函数

```
- (void)faceSessionCompletionWithStatus:(BDFaceCompletionStatus)status result:(NSDictionary *)result;`
```

2.1.3.9 BDFaceCompletionStatus的状态

```
BDFaceCompletionStatusSuccess = 1,
BDFaceCompletionStatusNoRisk = 2,
BDFaceCompletionStatusImagesSuccess = 3,
```

- 上述三个都是采集流程正常的回调，**BDFaceCompletionStatusNoRisk**表示设备没有风险。
- **BDFaceCompletionStatusSuccess** 和 **BDFaceCompletionStatusImagesSuccess** 的区别
  是：**BDFaceCompletionStatusSuccess** 代表着整个流程的完成，即人脸离线活体检测和最终人证核验流程的结束；而**BDFaceCompletionStatusImagesSuccess**代表了动作采集流程完成，但最终的人证核验流程还没有完成。

2.1.3.10 重要的开始人脸检测函数

```
[[SSFaceSDKManager sharedInstance] livenesswithList:livenessArray order:order
numberOfLiveness:numberOfLiveness];
```

注意：进行人脸识别之前，需要先调用initCollect方法进行安全SDK初始化； 2.1.3.11 活体状态检测函数

```
- (void)livenessActionDidFinishWithCode:(LivenessRemindCode)code;
```

活体检测状态，如引导用户张嘴，转头都在这个回调里，具体看BDFaceLivenessViewController即可。 2.1.3.12 开始人脸识别函数

```
- (void)startRecognize {
[[BDFaceImageShow sharedInstance] setSuccessImage:nil];
NSDictionary *parameters = [self.videoCapture creatFaceVerifyParameters:self.idCardNumber name:self.name
verifyType:KFaceIdCardTypeDefault nation:nil phoneNumber:nil livenessControl:nil spoofingControl:nil qualityControl:nil];
[self.videoCapture startSessionWithType:BDFaceResultReportTypeVerifySec parameters:parameters
faceFlow:self.faceFlowType viewController:self];
}
```

注意：上述函数中 参数: self.idCardNumber 是身份证号信息，self.name是姓名信息

2.1.4 SDK相关头文件介绍 2.1.4.1 SSFaceSDK.h

包含其他头文件，及人脸扫描、活体验证状态码，同时还有执行业务流程的参数枚举等。此处没有一一列举，具体请参照头文件：

```
 #import "SSFaceSDKManager.h"
#import "SSFaceDetectionManager.h"
// 参数枚举等
typedef NSString *FaceIdCardType NS_STRING_ENUM;
FOUNDATION_EXPORT FaceIdCardType const KFaceIdCardTypeDefault; // 默认 大陆身份证
FOUNDATION_EXPORT FaceIdCardType const KFaceIdCardTypeMTPIDCard; // 港澳居民来往内地通行证
FOUNDATION_EXPORT FaceIdCardType const KFaceIdCardTypeFPRIDCard; // 外国人永久居留身份证
FOUNDATION_EXPORT FaceIdCardType const KFaceIdCardTypePassport; // 定居国外的中国公民护照

// 活体检测返回状态
typedef NS_ENUM(NSUInteger, LivenessRemindCode) {
LivenessRemindCodeOK = 0,  //成功
LivenessRemindCodeBeyondPreviewFrame,  //出框
LivenessRemindCodeNoFaceDetected, //没有检测到人脸
LivenessRemindCodeMuchIllumination,
LivenessRemindCodePoorIllumination,  //光照不足
LivenessRemindCodeImageBlured,  //图像模糊
...
```

2.1.4.2 SSFaceSDKManager.h

图像采集行为和过程中需要的设置，属于全局配置。在原IDL库中FaceSDKManager基础上增加如下两个接口：

```
 /**
 *  SDK云端校验设置
 *  需要云端校验，需提前申请id和secret
 *
 *  @param clientId api key
 *  @param clientSecret api secret
 */
- (void)setBCEClientId:(NSString *)clientId clientSecret:(NSString *)clientSecret;


/**
 * 设置活体动作
 * @param array 包含活体动作种类
 * @param order 是否顺序执行
 * @param numberOfLiveness 活体数量
 */
- (void)livenesswithList:(NSArray *)array order:(BOOL)order numberOfLiveness:(NSInteger)numberOfLiveness;
```

2.1.4.3 SSFaceDetectionManager.h

用于进行人脸识别具体行为动作。

```
 @interface SSFaceDetectionManager : NSObject
// 图像返回帧处理代理
@property (nonatomic, weak) id<SSCaptureDataOutputProtocol> delegate;

// 采集流程运行状态
@property (nonatomic, assign, readonly) BOOL runningStatus;

// 风险检测超时时间：-1 关闭风险检测；大于0 风险检测超时时间；0 使用默认超时时间3秒
@property (nonatomic, assign) NSInteger riskDetectionSetting;

// 设置使用镜头，前置/后置
@property (nonatomic, assign) AVCaptureDevicePosition devicePosition;

// 输出形式，AVCaptureSessionPreset类型
@property (nonatomic, copy) NSString *sessionPresent;
```

```
// 采集图像区域
@property (nonatomic, assign) CGRect previewRect;

// 探测区域
@property (nonatomic, assign) CGRect detectRect;

// 是否开启声音提醒
@property (nonatomic, assign) BOOL enableSound;


/**
 * 创建用于实名认证的参数
 */
- (NSDictionary *)creatFaceVerifyParameters:(NSString *)idCardNumber
                                       name:(NSString *)name
                                 verifyType:(FaceIdCardType)cardType
                                     nation:(NSString *)nation
                                phoneNumber:(NSString *)phoneNumber
                            livenessControl:(FaceLivenessControlType)livenessControl
                            spoofingControl:(FaceSpoofingControlType)spoofingControl
                             qualityControl:(FaceQualityControlType)qualityControl;

/**
 * 创建用于人脸比对的参数
 */
- (NSDictionary *)createFaceMatchParametersWithRegisterImage:(NSString *)registerImageBase64
                                           registerImageType:(FaceRegisterImageType)registerImageType
                                            registerFaceType:(FaceFaceType)registerFaceType
                                                    faceType:(FaceFaceType)faceType
                                                faceSortType:(FaceSortype)faceSortType
                                                 phoneNumber:(NSString *)phoneNumber
                                             livenessControl:(FaceLivenessControlType)livenessControl
                                              qualityControl:(FaceQualityControlType)qualityControl
                                     registerLivenessControl:(FaceLivenessControlType)registerLivenessControl
                                      registerQualityControl:(FaceQualityControlType)registerQualityControl;


/**
 * 开始当前人脸校验流程
 * @param detectionType 业务流程，采集信息用于实名认证、人脸比对
 * @param parameters 流程需要的参数
 * @param flowType 操作流程，人脸采集、人脸活体
 * @param vc 用于进行人脸信息采集的ViewController
 */
- (void)startSessionWithType:(BDFaceResultReportType)detectionType parameters:(NSDictionary *)parameters faceFlow:
(BDFaceFlowType)flowType viewController:(UIViewController *)vc;

/**
 * 取消当前人脸校验流程
 */
- (void)cancel;
```

通过startSession方法开始进行人脸采集之后，会通过SSCaptureDataOutputProtocol类型的delegate进行回调：

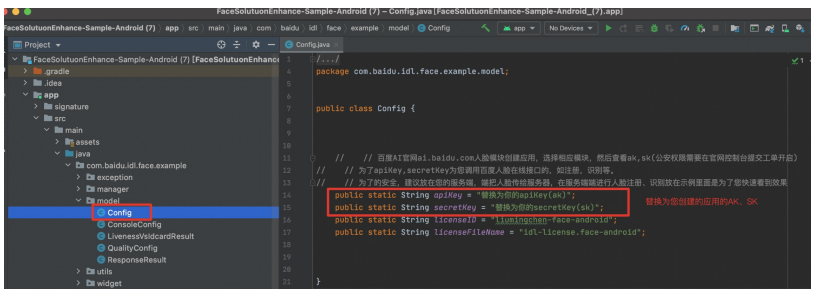
```
/**
 * 流程返回结果类型
 */
typedef NS_ENUM(NSInteger, BDFaceCompletionStatus) {
BDFaceCompletionStatusSuccess = 1,  // 成功
BDFaceCompletionStatusNoRisk = 2,   // 无风险
BDFaceCompletionStatusImagesSuccess = 3, // 图像采集成功

BDFaceCompletionStatusProcessing = 4, // 正在采集图像
```

```
BDFaceCompletionStatusIsRunning = -1,  // 正在采集图像
BDFaceCompletionStatusResultFail = -2,  // 云端服务执行失败
BDFaceCompletionStatusIsRiskDevice = -3,    // 风险设备
BDFaceCompletionStatusCameraError = -5,    // 没有授权镜头
BDFaceCompletionStatusTimeout = -6, // 超时
BDFaceCompletionStatusSDKNotInit = -13, // SDK未初始化
BDFaceCompletionStatusLicenseFail = -15, // 授权错误
BDFaceCompletionStatusNetworkError = -16,    // 网络错误
};

@protocol SSCaptureDataOutputProtocol <NSObject>

// 帧图像回传，用于刷新UI使用
- (void)captureOutputSampleBuffer:(UIImage *)image;

// 本次人脸流程回调
- (void)faceSessionCompletionWithStatus:(BDFaceCompletionStatus)status result:(NSDictionary *)result;

// 活体检测状态
- (void)livenessActionDidFinishWithCode:(LivenessRemindCode)code;

// 人脸识别检测
- (void)detectionActionDidFinishWithCode:(DetectRemindCode)code;

@end
```

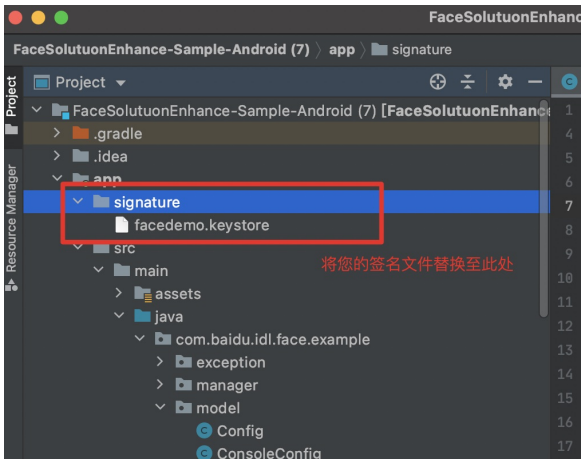
2.2 安卓集成 2.2.0 运行demo

（1）替换AK、SK

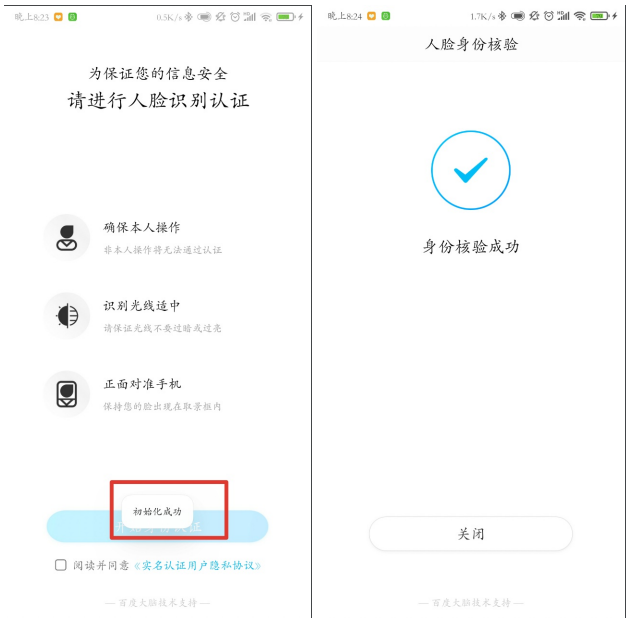
AK/SK获取方式请参考这里



（2）替换签名



运行成功后，如下图所示。



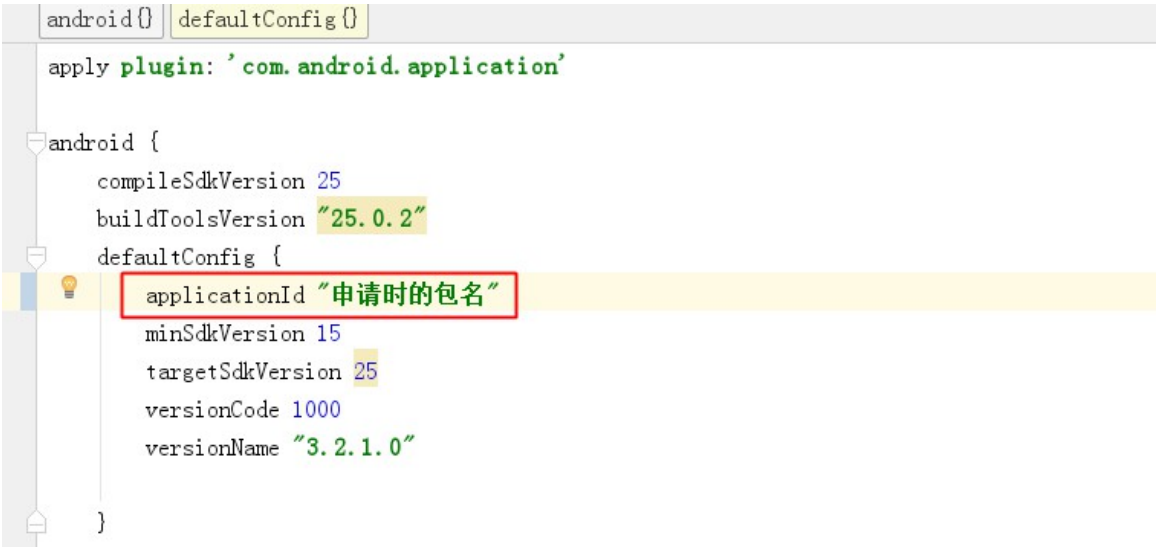
2.2.1 资源准备 将申请得到的以下文件放到app module 下的assets目录中

1. aip.license //OCR 身份证识别的license，不能修改该文件名
2. console_config.json //人脸识别的配置
3. idl-key.face-android //LH 加固版人脸采集的密钥，不能修改该文件名
4. idl-license.face-android //FACESDK的license

备注 ：以下列举几个关键流程，完整流程请参考下载示例代码

2.2.2 OCR身份证识别集成

- 把申请的license（aip.license）放到到项目中app module下的assets目录中
- 修改app的build.gradle包名为申请时填入的包名



- 拷贝ocr-ui到您的工程中

接下来调用具体请看示例工程

Step 1:在首页面初始化OCR SDK（在HomeActivity.java中）

```
private void initOCRSDK() {
    OCR.getInstance(this).initAccessToken(new OnResultListener<AccessToken>() {
        @Override
        public void onResult(final AccessToken result) {
            runOnUiThread(new Runnable() {
                @Override
                public void run() {
                    // 调用成功，返回AccessToken对象
                    String token = result.getAccessToken();
                }
            });
        }

        @Override
        public void onError(final OCRError error) {
            runOnUiThread(new Runnable() {
                @Override
                public void run() {
                    //调用失败
                    Log.d(TAG, "run: erromsg=" + error.getMessage());
                }
            });

        }
    }, getApplicationContext());
}
```

Step 2: 跳转至身份证扫描页面进行身份证扫描（在HomeActivity.java中）

```
/**
 * 开始进入身份证认证页面
 */
private void startIdCardVerify() {
    if (mConsoleConfig.getUseOcr() == 1) {
        Intent intent = new Intent(mContext, CameraExpActivity.class);
        // 设置临时存储
        intent.putExtra(CameraActivity.KEY_OUTPUT_FILE_PATH,
                FileUtil.getSaveFile(getApplication()).getAbsolutePath());
        // 调用拍摄身份证正面的activity
        intent.putExtra(CameraActivity.KEY_CONTENT_TYPE, CameraActivity.CONTENT_TYPE_ID_CARD_FRONT);
        intent.putExtra(CameraActivity.KEY_NATIVE_TOKEN, OCR.getInstance(this).getLicense());
        intent.putExtra(CameraActivity.KEY_NATIVE_ENABLE, true);
        startActivity(intent);
    } else {
        Intent intent = new Intent(mContext, IdCardInputActivity.class);
        startActivity(intent);
    }
}
```

Step 3:调用身份证识别请求接口api进行身份证识别（在 IdCardRecognizeActivity.java**中）** 其中请求参数如下：

请求参数	**说明**
image	身份证扫描的图像
id_card_side	身份证的正反面类型
detect_risk	是否开启旋转识别

```
/**
 * 识别身份证
 */
private void recIDCard(String idCardSide, final String filePath) {
    IDCardParams param = new IDCardParams();
    param.setImageFile(new File(filePath));
    param.setIdCardSide(idCardSide);
    param.setDetectDirection(true);
    OCR.getInstance(this).recognizeIDCard(param, new OnResultListener<IDCardResult>() {
        @Override
        public void onResult(final IDCardResult result) {
            runOnUiThread(new Runnable() {
            @Override
            public void run() {
                if (result == null) {
                    return;
                }
                //识别成功获取值
                Word idNumber = result.getIdNumber();
                Word nameWord = result.getName();
                if (idNumber != null) {
                    mIdCardNum = idNumber.getWords();
                }

                if (nameWord != null) {
                    mUsername = nameWord.getWords();
                }
            }
        });
    }

    @Override
    public void onError(final OCRError error) {
        //识别失败
    }
  });
}
```

Step 4: 获得身份证号码和姓名后进入人脸活体验证步骤

详细调用文档，请参考 OCR-Android-SDK文档

2.2.3 人脸SDK集成

Step 1: 授权参数

（1）导入如下module

- 拷贝faceplatform-ui到您的工程中
- 拷贝lib-liantian到您的工程中

（2）把申请的license（idl-license.face-android","idle-key.face-android")放到到项目中app module下的assets目录中（console_config.json 为示例工程配置，开发者可更换配置方案）

（3）修改 `Config.java` 类中的参数

> 在人脸实名认证控制台创建完APP方案后，会自动生成LicenseID和LicenseFileName信息，这里您只需要填写apiKey和secretkey信息即可。

```
public class Config {
    // 为了apiKey,secretKey为您调用百度人脸在线接口的，如注册，比对等。
    // 为了的安全，建议放在您的服务端，端把人脸传给服务器，在服务端端
    // license为调用sdk的人脸检测功能使用，人脸识别 = 人脸检测（SDK功能）+ 人脸比对（服务端API）
    public static String apiKey = "替换为你的apiKey(ak)";
    public static String secretKey = "替换为你的secretKey(sk)";
    public static String licenseID = "替换为你的licenseID,后台SDK管理界面中，已经生成的licenseID,如:test-face-android";
    public static String licenseFileName = "替换为你的license文件名称（放于assets目录）";
}
```

（4）配置签名（**申请license时的md5为打包签名的文件，所以必须用申请license的签名文件**）

```
app->build.gradle->android->signingConfigs
signingConfigs {
        def password = "替换为签名密码"
        def alias = "替换为签名别名"
        def filePath = "替换为签名文件路径" //如 ../facesharp.jks//签名文件路径
        debug {
            keyAlias alias
            keyPassword password
            storeFile file(filePath)
            storePassword(password)
        }
        release {
            keyAlias alias
            keyPassword password
            storeFile file(filePath)
            storePassword(password)
        }
    }
```

（5）修改包名 app->build.gradle->android->defaultConfig ->applicationId您申请license时填的包名

Step 2:初始化人脸相关SDK

`ExampleApplication`中初始化LH安全加固人脸采集类

```
private void initLH() {
    LH.setAgreePolicy(getApplicationContext(), true);
    LH.init(getApplicationContext(),
            Config.licenseID,
            Config.apiKey,
            Config.secretKey);

}
```

HomeActivity.java中初始化FaceSDK

```
private void initLicense() {
  setFaceConfig();
  // 为了android和ios 区分授权，appId=appname_face_android ,其中appname为申请sdk时的应用名
  // 应用上下文
  // 申请License取得的APPID
  // assets目录下License文件名
  FaceSDKManager.getInstance().initialize(mContext,
  Config.licenseID, Config.licenseFileName, new IInitCallback() {
        @Override
        public void initSuccess() {
        // 初始化OCR SDK 使用的license是aip.license，名字不能修改
            initOCRSDK();
        }

        @Override
        public void initFailure(final int errCode,
        final String errMsg) {
            runOnUiThread(new Runnable() {
              @Override
              public void run() {
                Log.e(TAG, "初始化失败 = " + errCode + " " + errMsg);
                showToast("初始化失败 = " + errCode + ", " + errMsg);
                mIsInitSuccess = false;
              }
            });
        }
    });
}
```

Step 3:设置配置参数，如不设置，将使用默认值（在 HomeActivity.java中）

```
 /**
 * 参数配置方法
 */
private boolean setFaceConfig() {
    FaceConfig config = FaceSDKManager.getInstance().getFaceConfig();
    mConsoleConfig = ConsoleConfigManager.getInstance(mContext).getConfig();
    if (mConsoleConfig == null) {
        return false;
    }
    // -----TODO：以下为通过console平台获取到的配置信息-----
    // 设置模糊度阈值
    config.setBlurnessValue(mConsoleConfig.getBlur());
    // 设置最小光照阈值（范围0-255）
    config.setBrightnessValue(mConsoleConfig.getIllumination());
    // 设置最大光照阈值（范围0-255）
    config.setBrightnessMaxValue(mConsoleConfig.getMaxIllumination());
    // 设置左眼遮挡阈值
    config.setOcclusionLeftEyeValue(mConsoleConfig.getLeftEyeOcclu());
```

```
        // 设置右眼遮挡阈值
        config.setOcclusionRightEyeValue(mConsoleConfig.getRightEyeOcclu());
        // 设置鼻子遮挡阈值
        config.setOcclusionNoseValue(mConsoleConfig.getNoseOcclu());
        // 设置嘴巴遮挡阈值
        config.setOcclusionMouthValue(mConsoleConfig.getMouthOcclu());
        // 设置左脸颊遮挡阈值
        config.setOcclusionLeftContourValue(mConsoleConfig.getLeftCheekOcclu());
        // 设置右脸颊遮挡阈值
        config.setOcclusionRightContourValue(mConsoleConfig.getRightCheekOcclu());
        // 设置下巴遮挡阈值
        config.setOcclusionChinValue(mConsoleConfig.getChinOcclu());
        // 设置人脸姿态角阈值
        config.setHeadPitchValue(mConsoleConfig.getPitch());
        config.setHeadYawValue(mConsoleConfig.getYaw());
        config.setHeadRollValue(mConsoleConfig.getRoll());
        // 设置活体动作，通过设置list，LivenessTypeEunm.Eye, LivenessTypeEunm.Mouth,
        // LivenessTypeEunm.HeadUp, LivenessTypeEunm.HeadDown, LivenessTypeEunm.HeadLeft,
        // LivenessTypeEunm.HeadRight, LivenessTypeEunm.HeadLeftOrRight
        config.setLivenessTypeList(mConsoleConfig.getActions());
        // 设置动作活体是否随机
        config.setLivenessRandom(mConsoleConfig.isFaceVerifyRandom());
        // 风控加密类型，0：普通版；1：加密版
        config.setSecType(mConsoleConfig.getSecType());

        // -----TODO：以下不需要通过console平台配置，需要手动修改-----
        // 设置可检测的最小人脸阈值
        config.setMinFaceSize(FaceEnvironment.VALUE_MIN_FACE_SIZE);
        // 设置可检测到人脸的阈值
        config.setNotFaceValue(FaceEnvironment.VALUE_NOT_FACE_THRESHOLD);
        // 设置闭眼阈值
        config.setEyeClosedValue(FaceEnvironment.VALUE_CLOSE_EYES);
        // 设置图片缓存数量
        config.setCacheImageNum(FaceEnvironment.VALUE_CACHE_IMAGE_NUM);
        // 设置开启提示音
        config.setSound(ExampleApplication.isOpenSound);
        // 原图缩放系数
        config.setScale(FaceEnvironment.VALUE_SCALE);
        // 抠图宽高的设定，为了保证好的抠图效果，建议高宽比是4：3
        config.setCropHeight(FaceEnvironment.VALUE_CROP_HEIGHT);
        config.setCropWidth(FaceEnvironment.VALUE_CROP_WIDTH);
        // 抠图人脸框与背景比例
        config.setEnlargeRatio(FaceEnvironment.VALUE_CROP_ENLARGERATIO);
        // 选择针对人脸采集输出图片的类型进行加密，0：原图，1：抠图
        config.setOutputImageType(FaceEnvironment.VALUE_OUTPUT_IMAGE_TYPE);
        // 检测超时设置
        config.setTimeDetectModule(FaceEnvironment.TIME_DETECT_MODULE);
        // 检测框远近比率
        config.setFaceFarRatio(FaceEnvironment.VALUE_FAR_RATIO);
        config.setFaceClosedRatio(FaceEnvironment.VALUE_CLOSED_RATIO);
        FaceSDKManager.getInstance().setFaceConfig(config);
        return true;
    }
```

Step 4:开始进行人脸采集

1. `IdCardConfirmActivity.java` 采集到身份证号码和姓名后开始进行人脸活体采集

```
    public static final String EXT_USERNAME = "username";
    public static final String EXT_ID_NUMBER = "idNumber";

    // 调转到动作活体采集界面
    faceIntent.setClass(this, FaceLivenessExpActivity.class);
    faceIntent.putExtra(EXT_USERNAME, mUsername);
    faceIntent.putExtra(EXT_ID_NUMBER, mIdNumber);
    startActivity(faceIntent);
```

2 . 采集过程中设置语音开关

```
  //设置是否开启语音
LH.setSoundEnable(mIsEnableSound);
```

3 .开启预览，示例A为开始实名认证流程，如只进行人脸比对流程则在开启预览时替换为示例B人脸对比方法即可

示例A,开始实名认证流程

```
protected void startPreview() {
        if (mSurfaceView != null && mSurfaceView.getHolder() != null) {
            mSurfaceHolder = mSurfaceView.getHolder();
        }
        Log.d(TAG, "name=" + sName + " ,idNumber=" + sIdNumber);
        LH.startFaceVerify(this, mSurfaceHolder, this, 5, new FaceVerifyInfo
                (sIdNumber, sName, 0, "CHN", FaceEnum.LivenessControl.NONE,
                        FaceEnum.SpoofingControl.NONE,
                        FaceEnum.QualityControl.NONE, ""), true);


    }
```

实名认证参数说明：

> **activity**：执行人脸采集的Activity。
>
> **previewSurfaceHolder**：用于展示摄像头预览的SurfaceView的Holder。
>
> **processCallback**：流程回调，见“FaceProcessCallback接口”。
>
> **deviceCheckTimeout**：安全检测超时时间，传入正数时，若在该时间内（单位：秒）未完成检测，则跳过。传入-1跳过检测，传入0后台检测。
>
> **info**：实名认证信息，包含身份证号，姓名等信息。
>
> **liveness**：是否进行活体检测。

示例B,开始人脸对比流程

```
    String image=Base64Utils.encodeToString(
        getImageFromAssetsFile(getApplicationContext(),"ceshi.jpg"),Base64Utils.NO_WRAP);

LH.startFaceCompare(this, mSurfaceHolder, this, 5, new FaceCompareInfo
        (FaceEnum.QualityControl.NONE, FaceEnum.LivenessControl.NORMAL,
                FaceEnum.FaceType.LIVE,0,
                image,FaceEnum.ImageType.BASE64,FaceEnum.FaceType.LIVE,
                FaceEnum.QualityControl.NORMAL, FaceEnum.LivenessControl.NORMAL,""), true);
//获取assets目录下的图片
public static byte[] getImageFromAssetsFile(Context context, String fileName) {
    byte[] result = null;
    AssetManager am = context.getResources().getAssets();
    try {
        InputStream is = am.open(fileName);
        result=new byte[is.available()];
        is.read(result);
        is.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
    return result;
}
```

人脸对比参数说明：

> **activity**：执行人脸采集的Activity。
>
> **previewSurfaceHolder**：用于展示摄像头预览的SurfaceView的Holder。
>
> **processCallback**：流程回调，见“FaceProcessCallback接口”。
>
> **deviceCheckTimeout**：安全检测超时时间，传入正数时，若在该时间内（单位：秒）未完成检测，则跳过。传入-1跳过检测，传入0后台检测。
>
> **info**：人脸比对信息，包含比对用图片等信息。
>
> **liveness**：是否进行活体检测。

4 .建议在 Activity.onPause()放方法中取消正在进行的流程

```
    @Override
    public void onPause() {
        if (!mIsCompletion) {
            LH.cancelFaceProcess();
        }
        mIsCompletion = false;
        super.onPause();
    }
```

取消正在进行的人脸比对或实名认证流程。安全SDK取消整个流程。请在Activity的onPause中调用此方法，保证界面离开顶端时流程被取消。下次进入界面应当重新开始人脸验证过程。调用此接口后，调用后LH 的onEnd()回调会返回-6错误码。属于正常现象。

5 . 获取校验结果

```
@Override
public void onEnd(final int status, String resultJson) {
    Log.e("FaceLivenessActivity", "onEnd called:" + status + "_" + resultJson);
    if (status == 1) {
        //流程正常结束
    } else {
        //流程异常结束
    }
}
```

获取校验结果API接口参数说明

> status：1代表正常结束， <0的值均表示未正常结束。
>
> resultJson：云端返回的结果。只有status为1时此字段有值。其他情况为空字符串。
>
> **onEnd的部分status值列举如下：**
>
> 1：正常结束，返回云端验证结果。
>
> -1：已经有一个采集验证流程在运行。
>
> -2：云端验证过程异常。
>
> -3：风控验证失败。
>
> -4：更严格情形下的风控验证失败。
>
> -5：摄像头异常。
>
> -6：流程被取消。
>
> -7：线程异常。
>
> -8：筛选图像异常。
>
> -9：采集前流程异常。
>
> -10：活体验证步骤异常。
>
> -11：预览异常。
>
> -12：采集后流程异常。
>
> -13：未初始化Liantian安全SDK。
>
> -14：未同意隐私协议。
>
> -15：未成功加载安全模块。
>
> -18：未检出人脸超时。
>
> -19：网络异常。

更多LH 接口信息点击 安全采集SDKAPI 接口

金融级APP实名认证方案

方案简介

方案简介

推出背景

- 现在，人脸识别技术被广泛应用在金融支付、用户注册、人脸登录等业务场景中。技术的进步方便用户的同时，黑灰产产业也开始对这些场景产生觊觎。并通过**屏幕攻击、照片、纸张、以及面具、头模**等方式进行非法攻击。随着黑产技术的进步，更是出现了通过**自动化脚本直接攻击云端API、ROM注入、视频劫持替换、批量虚拟机、病毒侵入**等新型攻击手段。使现有的人脸识别方案面临着巨大的安全挑战。

- 为提升人脸识别的安全性，保障客户的业务安全，人脸实名认证产品团队与百度安全实验室联合推出**金融级人脸实名认证方案**，在人脸登录、注册等环境加入层层保障，为您的业务保驾护航。

功能简介

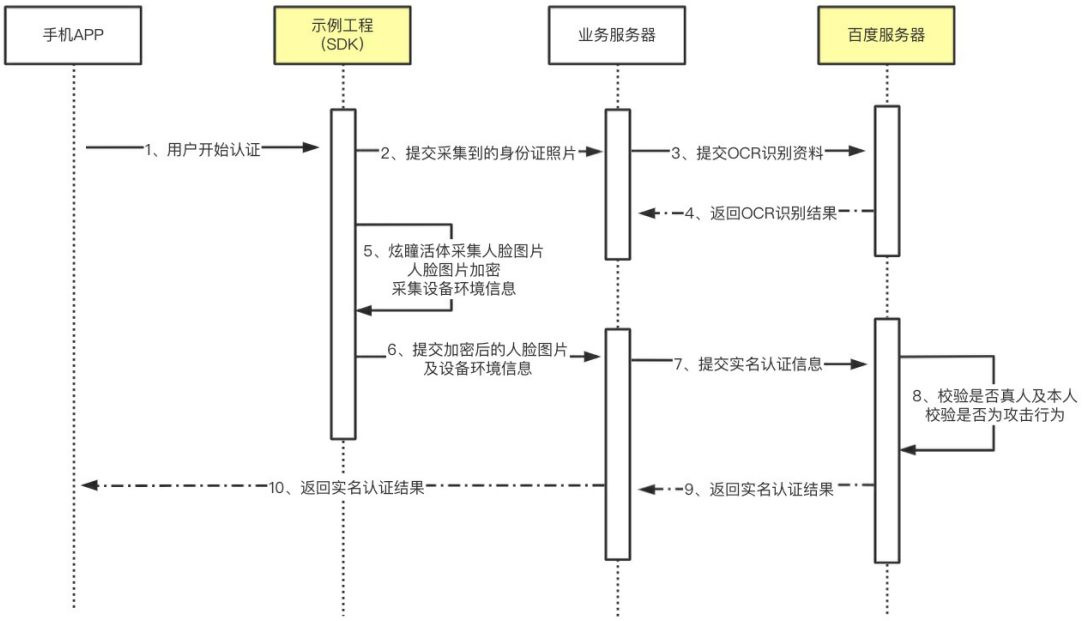
- 金融级APP实名认证方案提供标准化的人身核验流程，具有**人脸比对、证件识别、活体检测**等多项组合能力，以及端云配合高防攻击拦截能力，可抵挡屏幕、照片、视频、换脸、面具、3D模型的攻击。同时，金融级实名认证方案中加入**安全加密能力**以及**大数据风控能力**，同时对活体检测能力进行优化，推出**炫瞳活体检测**功能，针对脚本攻击、ROM注入、视频劫持、批量虚拟机、病毒侵入等新型攻击手段进行强力有效防御。

> **安全加密能力：**（金融级方案中**默认开启**此项功能，无需您自行配置）金融级APP方案集成文件中的采集SDK会对输出的图片进行加密，在云端**金融级实名认证API**进行解密。此端云配合的加密方式是百度专门针对市面黑产绕过采集SDK，攻击云端接口的攻击方式进行的功能升级。
>
> **大数据风控能力：**金融级API接口接受SDK端传入的设备指纹信息，基于百度海量大数据设备因子，对SDK端进行设备风险识别，辨别是否为风险设备，返回识别结果。可有效防御黑产批量虚拟机、病毒侵入等攻击手段，降低第三方黑产攻破概率，提升业务安全性。
>
> **炫瞳活体检测：**基于屏幕颜色打光的方式，通过我们的面部反光和**瞳孔反光**对核验人员进行活体判断。相比于行业内传统的动作活体和视频活体检测方式，通过率大大提升，使用效率更加流畅便捷，有效拦截视频、图片伪造、3D面具、合成图等黑产攻击。

适用场景

- 金融级APP实名认证方案适用于**安卓/IOS系统的APP场景**中实现用户实名认证。如果您的业务场景是在微信小程序、公众号、H5等业务场景，推荐采用金融级H5实名认证方案

接入时序图



方案接入步骤

- 金融级APP实名认证方案的具体接入步骤请参考方案接入指南

方案集成前准备

本文档介绍了金融级APP实名认证方案配置流程，以及APP集成开发前的准备工作。

在正式集成前，需要做一些准备工作，完成一些账号、应用及配置，具体如下：

Step1: 注册成为开发者

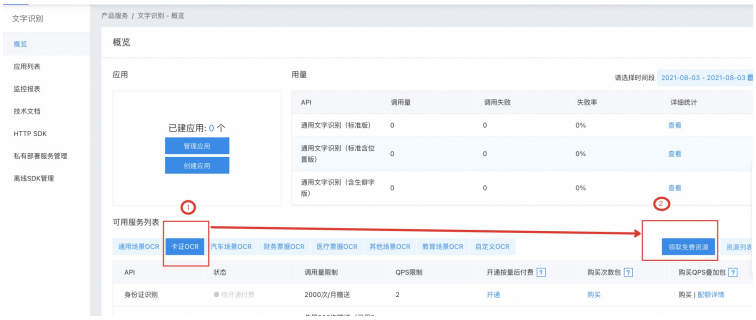
> 在使用百度人脸实名认证方案之前，首先需注册百度云账号，账号注册方式请参考**账号注册指南**。
>
> 百度云账号注册完成以后，为顺利调用百度AI能力，需完成企业认证。具体认证方式请参考**企业认证指南**。

Step2：创建应用

2.1 输入应用名称，领取免费额度

- 创建好账号后，在正式调用AI 能力之前，需首先**创建应用**，应用是调用服务的基础能力单元。 同时领取接口所需的**免费调用额度**，用于接入测试。如下图所示：

- 除人脸服务接口的免费调用额度外，还需领取**身份证识别**接口的**免费调用额度**，用来调用身份证OCR识别功能（必须领取，否则会报错服务异常），点击此处，按下图所示进行领取。



- 如您之前已经领取过免费额度，无需重复领取，请跳至下一步骤。

2.2 勾选所需接口

- **人脸识别服务相关接口已默认勾选且不可取消。**

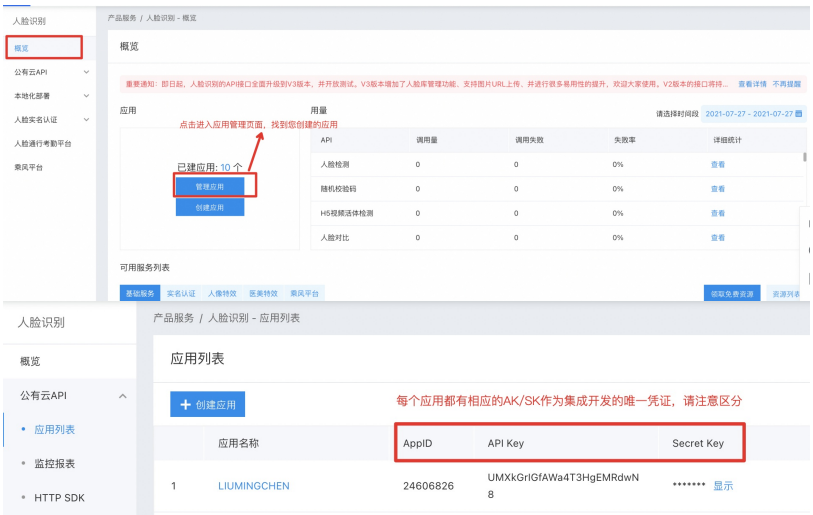
- **注：**「接口选择」过程中，还**需勾选「文字识别」中的「身份证识别」接口**，用于实现身份核验流程中的身份证识别功能。如下图所示。

2.3 输入应用包名

- **在「文字识别包名」处选择「需要」**，并根据您的APP应用信息填写包名。此处为必要操作，否则将无法顺利下载集成文件。**至此应用创建完成。**

2.4 获取密钥信息（AK/SK）

完成应用创建后，平台将会分配给您此应用的相关凭证，主要为**AppID、API Key、Secret Key**，以上三个信息是您应用实际开发的主要凭证，每个应用之间各不相同，请您妥善保管。您可在控制台的**应用管理页面**找到以上信息。如下图所示



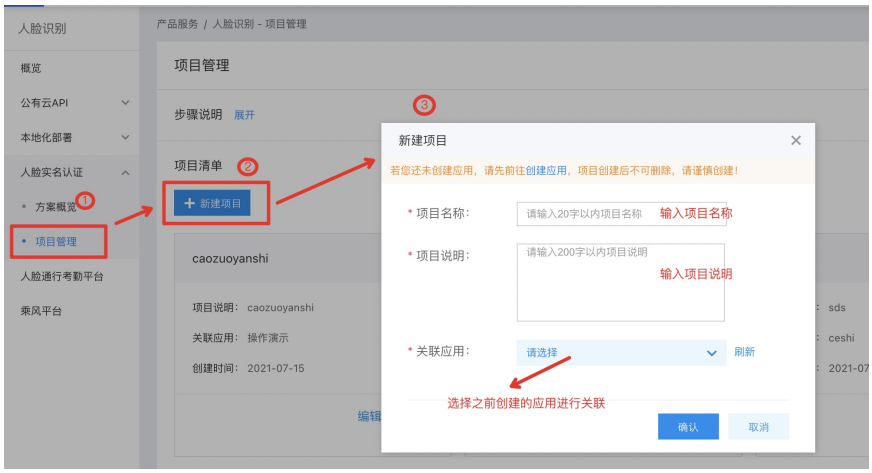
该AK/SK用于调用在线API 如：身份验证。**在之后下载的集成文件（示例工程）中需要填写正确的AK/SK以顺利集成。**

注：开发中请注意区分多份AK/SK（API Key、Secret Key），若填写的AK/SK与开发的应用不对应，会产生鉴权错误。

Step3：创建项目

- 进入控制台-人脸实名认证页面，选择『**项目管理**』页面，点击『**新建项目**』，进行项目创建，如下图所示。

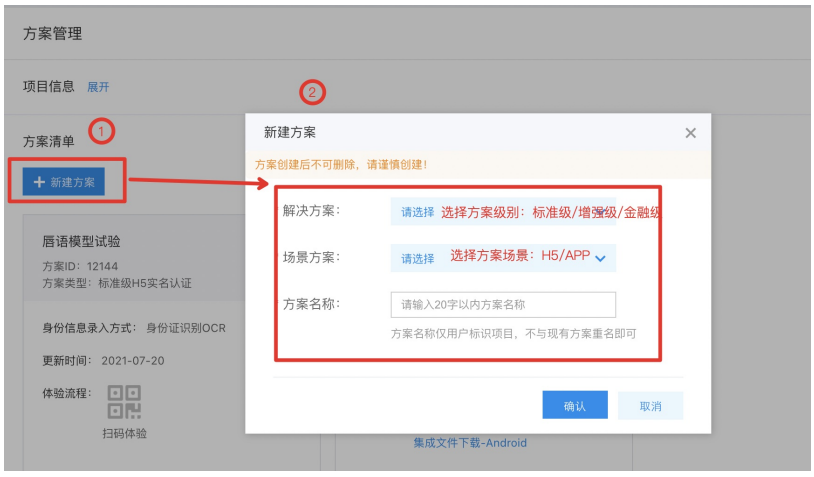
> 创建项目前，请确保您在应用控制台已创建应用，若您未创建应用，请参考STEP2创建应用后，再进行项目创建.



Step4：创建方案

- 项目创建完成后，点击「**方案管理**」进入方案管理页面，在这里您可以为您的项目创建不同的方案，如下图所示。

> 若您的场景为APP场景（安卓/IOS系统），『场景方案』请选择APP实名认证方案；
>
> 若您的场景为微信、H5页面，『场景方案』请选择H5实名认证方案。



4.1 身份信息录入

- 身份信息录入支持选择**用户手动输入**或**OCR拍照采集**，如下图所示

OCR拍照采集：会在之后生成的APP集成文件内集成OCR采集SDK，在本地进行身份证质量校验，并判断是否为身份证证件。 （**OCR拍照采集**支持实时采集和相册上传两种形式。推荐您使用实时采集，可以在一定程度增加方案的安全性及便捷性。）

手动输入：支持用户手动输入姓名+身份证号信息。

注：如您的业务场景无需使用身份信息录入功能，此项可先默认选择，在后续集成指导技术文档中将说明如何去除该部分功能

4.2 采集SDK配置

- **填写授权标识：**选择SDK的授权标识信息。

若您还未申请授权标识信息，请点击**『新建授权』，并填写相关信息**进行申请，申请过程如下图所示。

- **图像质量控制（本地）**：分为严格、正常、宽松三个等级，等级越严格，对采集图片的角度、模糊度、遮挡等信息参数把控越高，推荐使用正常。

> 此项配置为离线采集SDK端对采集图片的质量要求，推荐实名认证场景选择严格或正常模式。图片质量越好，云端接口传输的通过率越高。

- **活体检测设置（本地）**：金融级离线采集 SDK在前端默认采取 **炫瞳活体检测+弱配合动作（眨眼/张嘴）**的活体检测方案。在该过程，SDK 会随机抓取几帧图像进行本地活体检测，检测通过后将图片传至后台进行下一步检测。

附录：

质量控制参数	「宽松」	「正常」	「严格」
光照最小值	30	40	60
光照最大值	240	220	200
遮挡-左眼	0.95	0.8	0.4
遮挡-右眼	0.95	0.8	0.4
遮挡-鼻子	0.95	0.8	0.4
遮挡-嘴巴	0.95	0.8	0.4
遮挡-左脸	0.95	0.8	0.4
遮挡-右脸	0.95	0.8	0.4
遮挡-下巴	0.95	0.8	0.4
姿态-俯仰角	30	20	15
姿态-左右角	18	18	15
姿态-旋转角	30	20	15
模糊度	0.8	0.6	0.4

4.3 金融级人脸实名认证API配置

- **人脸实名认证接口：**金融级实名认证方案默认接入 **金融级人脸实名认证 API接口**。
- **安全加密能力：**（金融级方案中**默认开启**此项功能，无需您自行配置）金融级APP方案集成文件中的采集SDK会对输出的图片进行加密，在云端**金融级实名认证API**进行解密。此端云配合的加密方式是百度专门针对市面黑产绕过采集SDK，攻击云端接口的攻击方式进行的功能升级。
- **大数据风控：**大数据风控功能开启后，接受SDK端传入的设备指纹信息，基于百度海量大数据设备因子，对SDK端进行设备风险识别，辨别是否为风险设备，返回识别结果。可有效防御黑产批量虚拟机、病毒侵入等攻击手段，降低第三方黑产攻破概率，提升业务安全性。
- **图像质量检测：**分为正常与宽松两个等级，等级设置越严格，对图片角度、模糊度、遮挡等信息参数把控越高，**推荐使用宽松。**
- **活体检测：**分为严格、正常、宽松三个等级，不同等级对应不同的活体检测阈值。等级设置越严格，对活体检测相关参数信息的把控越高。不同等级对应指标可参考下表，**推荐使用正常。**

> **活体检测阈值：**活体检测得分高于此阈值，即判断为活体
>
> **误拒率（FRR）：**指误将活体用户判断为非活体的概率。如误拒率为0.5%，指1000次真人请求，会有5次因为活体分数低于阈值被错误拒绝。

控制度	对应阈值	说明
宽松	0.05	万分之一活体误拒率
正常（推荐）	0.3	千分之一活体误拒率
严格	0.9	百分之一活体误拒率

- **阈值：**用户人脸图片与公安权威数据源中人脸的相似度得分阈值，得分超过此阈值，即被判断为同一人。阈值分数相关指标可参考下表，**推荐阈值为80。**

阈值分数	误识率	识别率
60	0.781615%	99.550128%
70	0.096534%	98.307626%
78	**0.015570%（万分之一）**	**95.672664%**
80 （推荐）	**0.009342%（低于万分之一）**	**94.323051%**

Step5：提交方案，获取示例工程

完成上述方案配置后，点击『提交』，进入方案管理页面，下载**IOS/安卓版集成文件（含示例工程）**进行SDK端集成使用。

> **Step4**中方案配置的参数会自动生成至集成文件（含示例工程）中，方便开发使用。 注意：请谨慎修改APP方案流程，修改后需要重新下载集成文件进行使用。

Step6：下载获取服务端示例工程

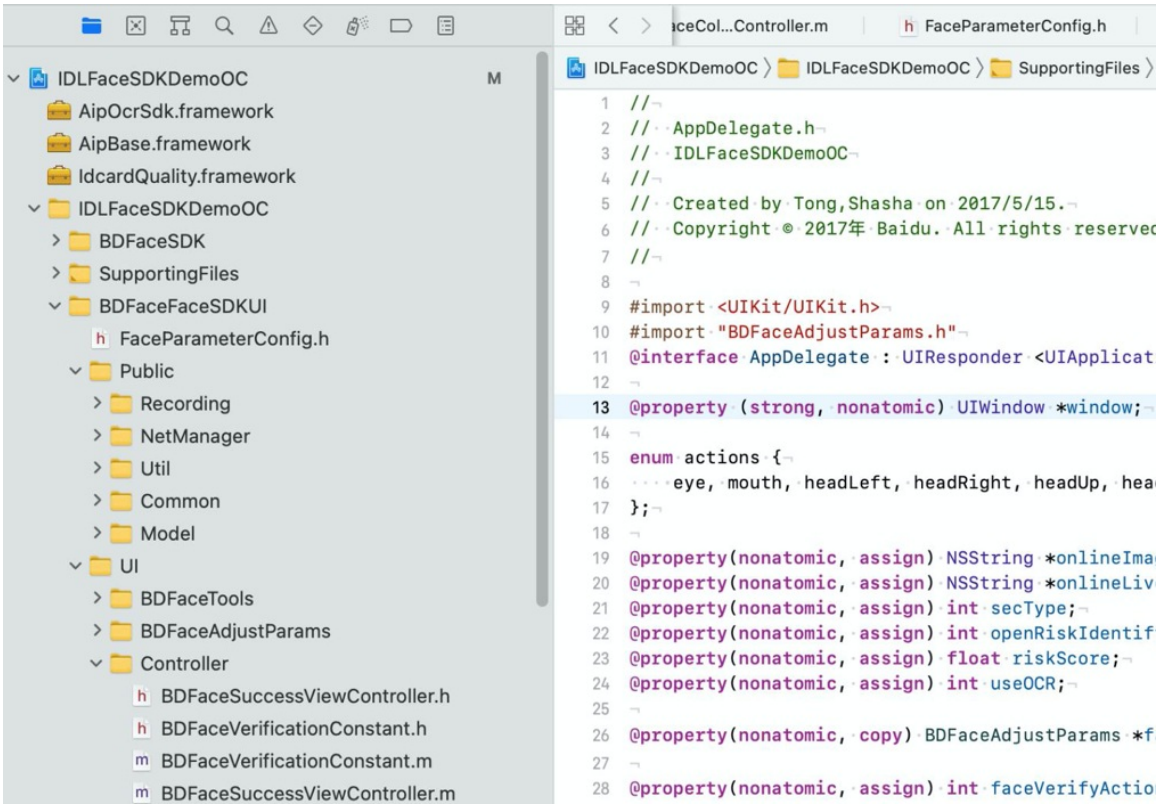
点击此处下载服务端示例工程及技术文档

至此，方案集成前的准备工作已完成，具体集成技术操作请方案集成指南。

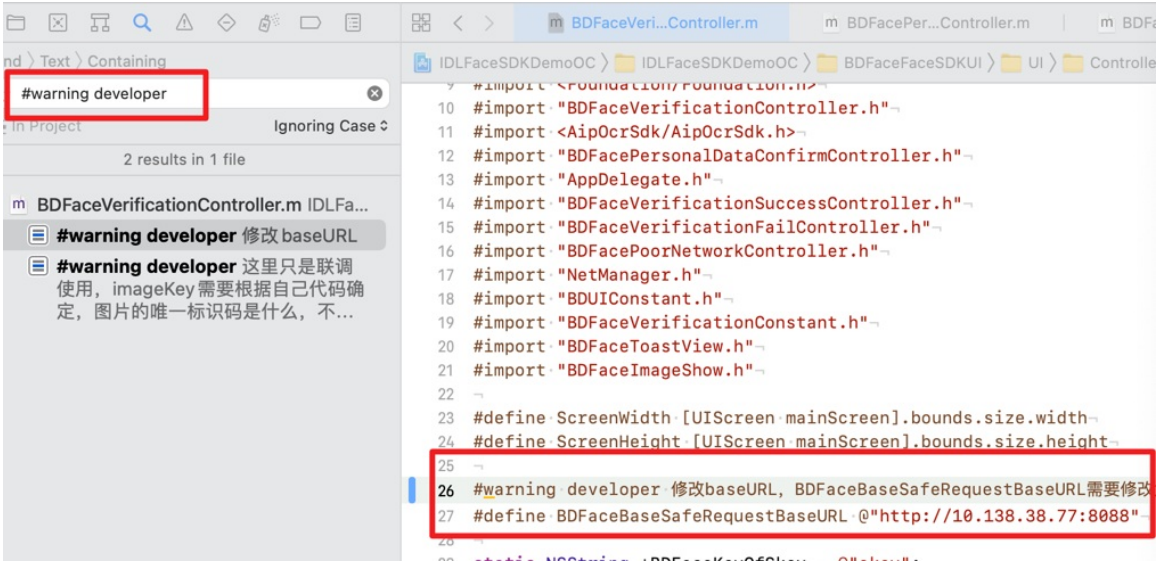
iOS-方案集成指南

本文档介绍了金融级APP实名认证方案iOS端集成开发流程。

一、运行示例工程 1、打开下载的iOS示例工程，如下图所示：

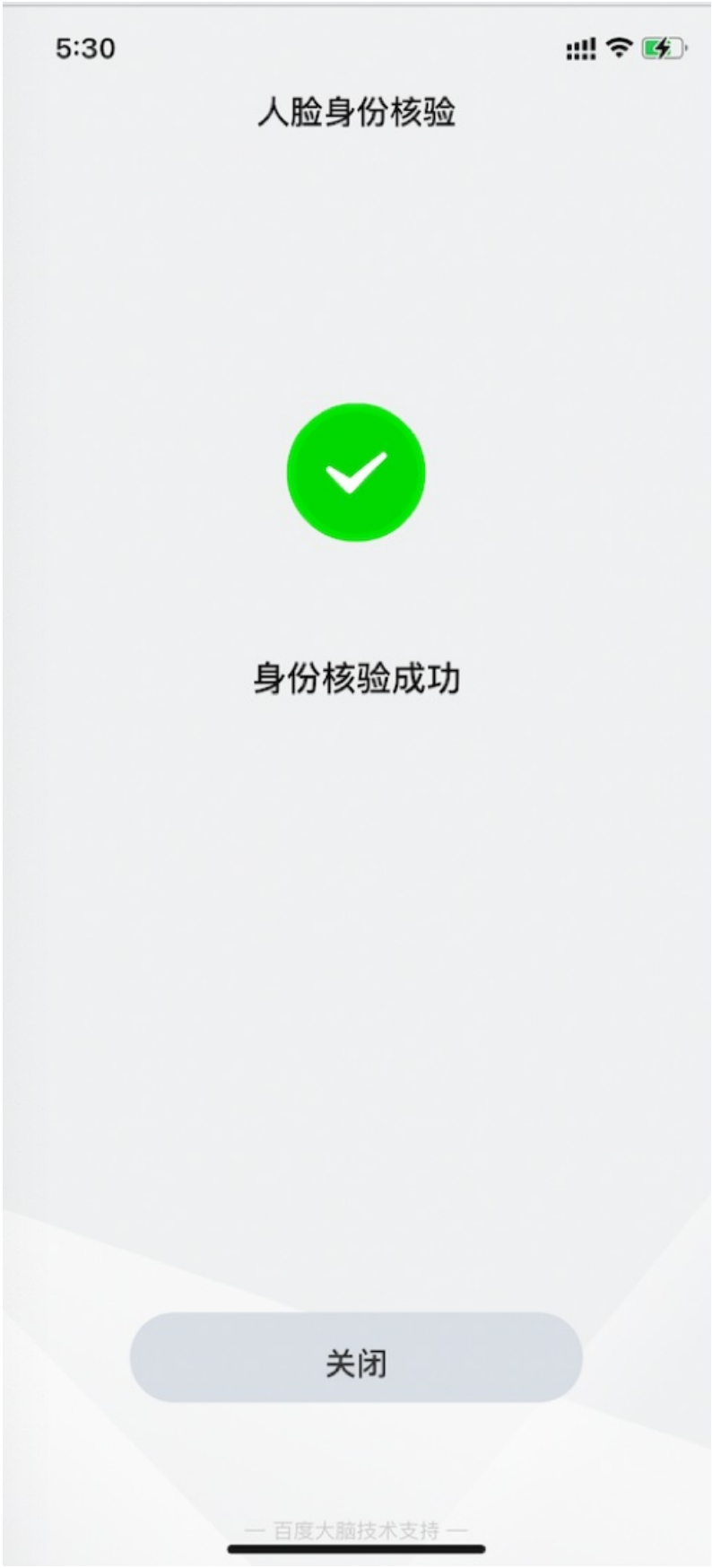


2、全局搜索关键词： #warning developer，找到宏定义BDFaceBaseSafeRequestBaseURL，修改这个值为指定的Server服务地址，具体如下图所示：



3、确认bundleId等信息是否正确，之后连接真机进行运行，之后可以看到如下界面：

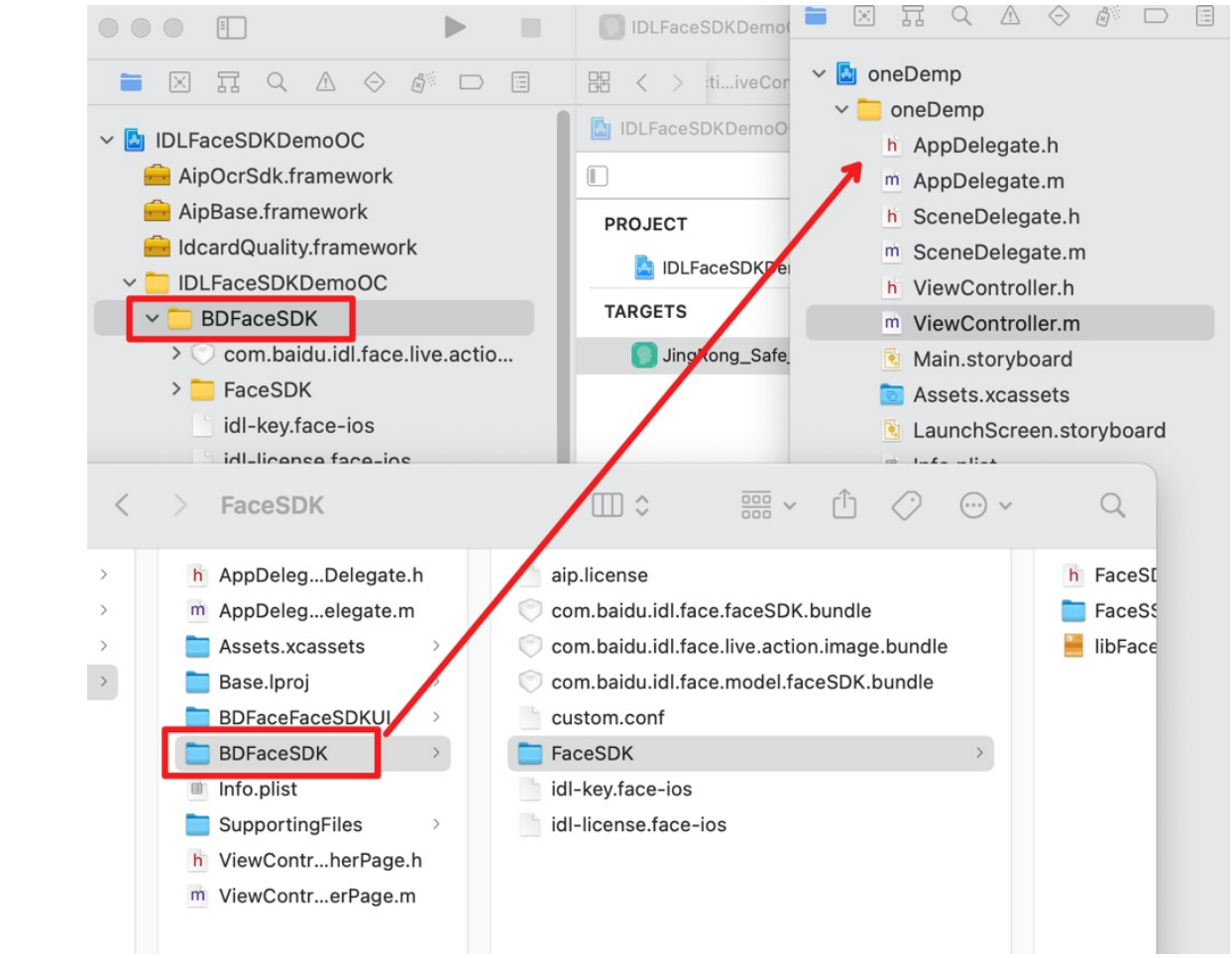
4、点击开始身份认证，测试示例工程是否跑通，之后扫码身份证或输入身份证信息后，点击进行身份核验，验证成功可以看到如下界面：



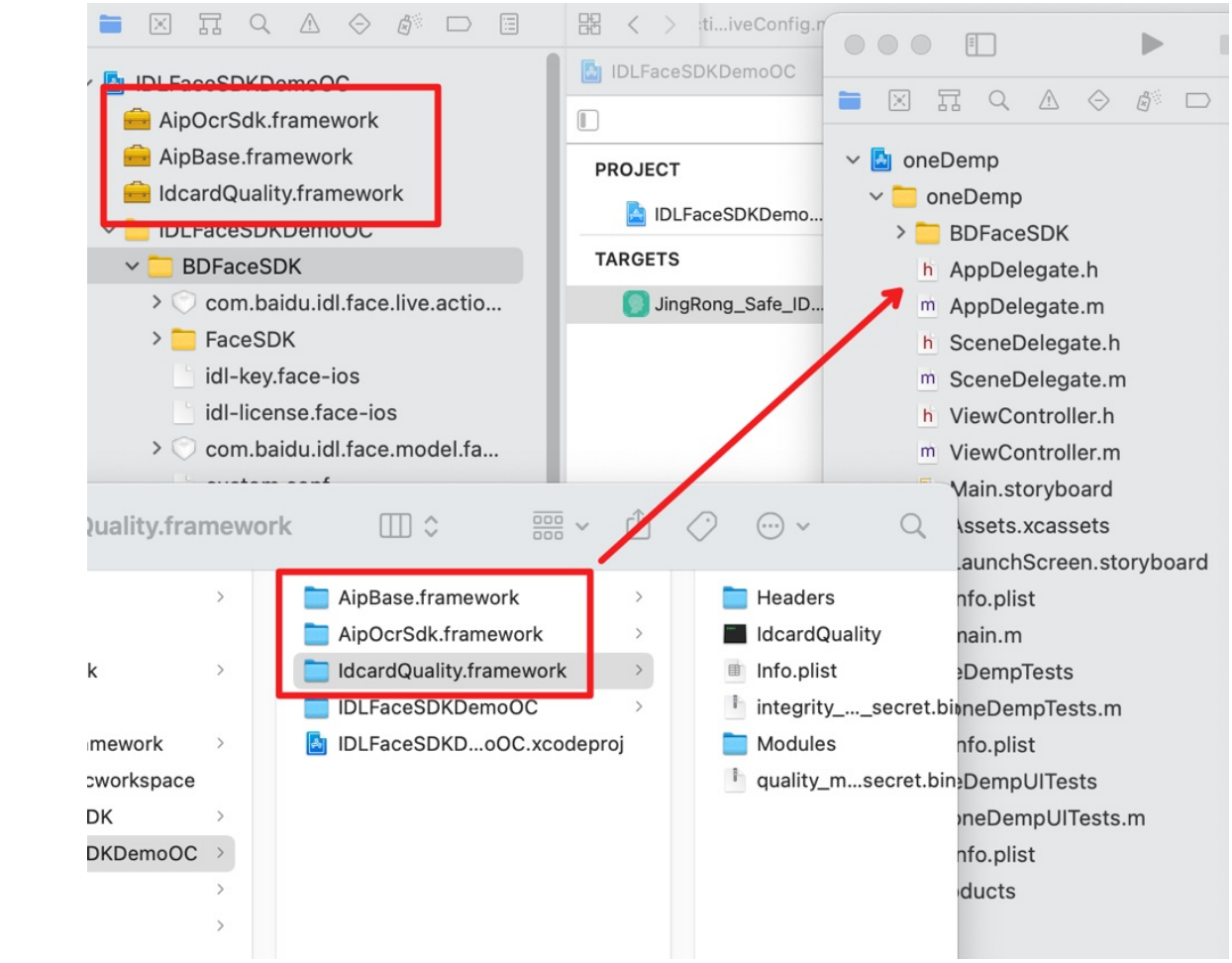
5、示例工程运行成功，之后可以将示例工程代码集成到目标项目中。

二、集成步骤

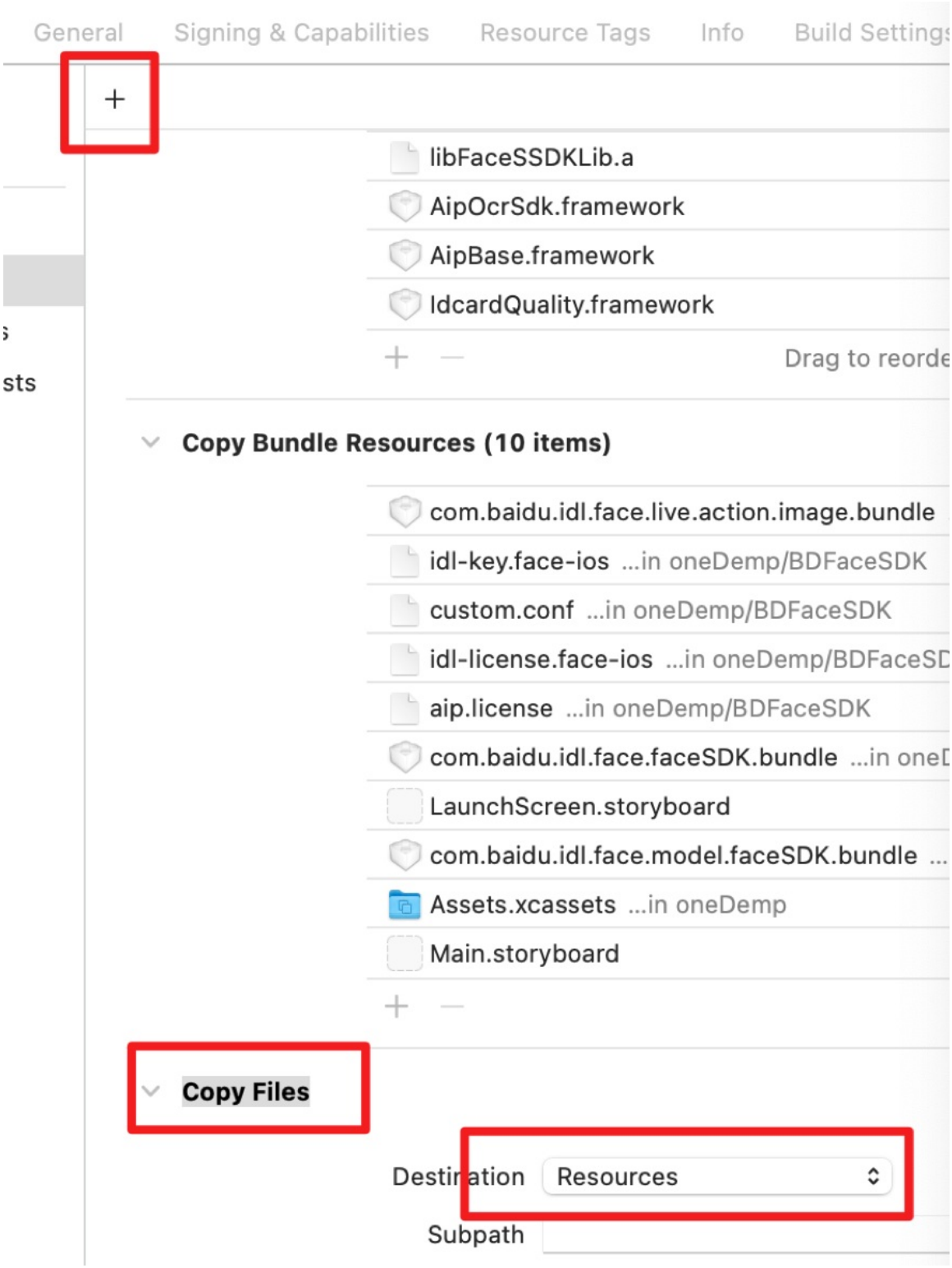
1、将BDFaceSDK文件夹下所有文件拖动到目标项目中，如下图所示



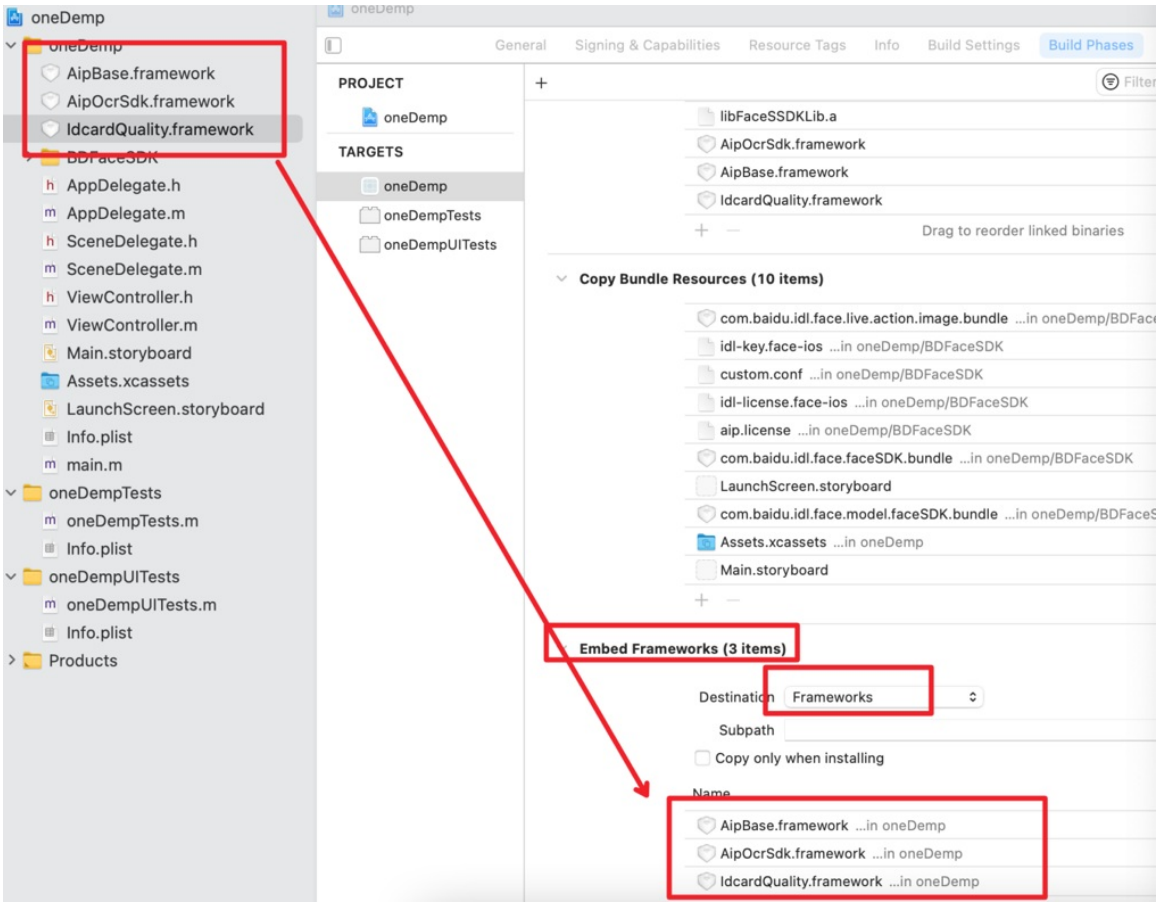
2、将AipBase.framework, AipOcrSdk.framework和IdcardQuality.framework拖动到目标项目，如下所图示



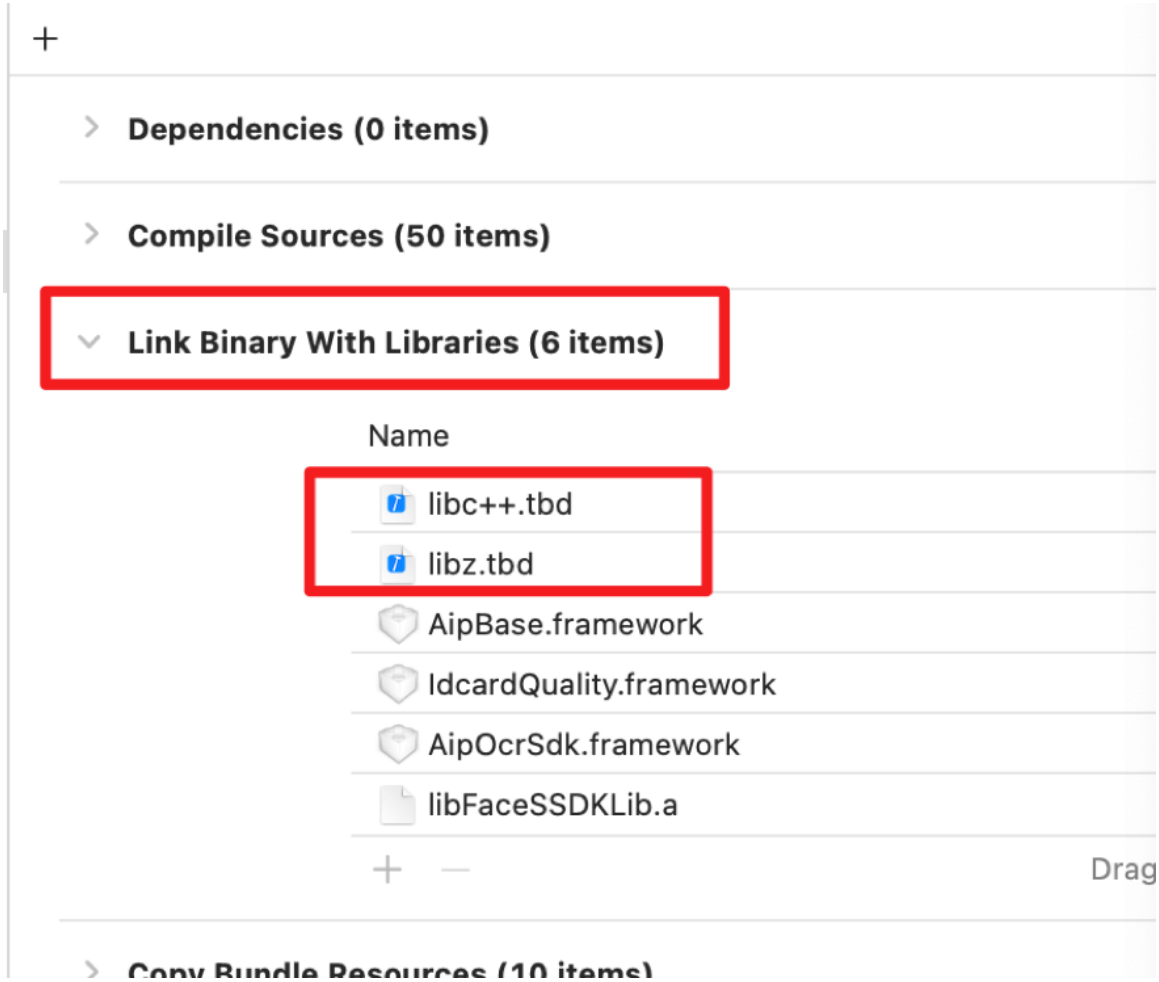
3、在Build Phases中点击下面的+号，选择NewCopyFilePhases,添加一个CopyFiles，如下图所示



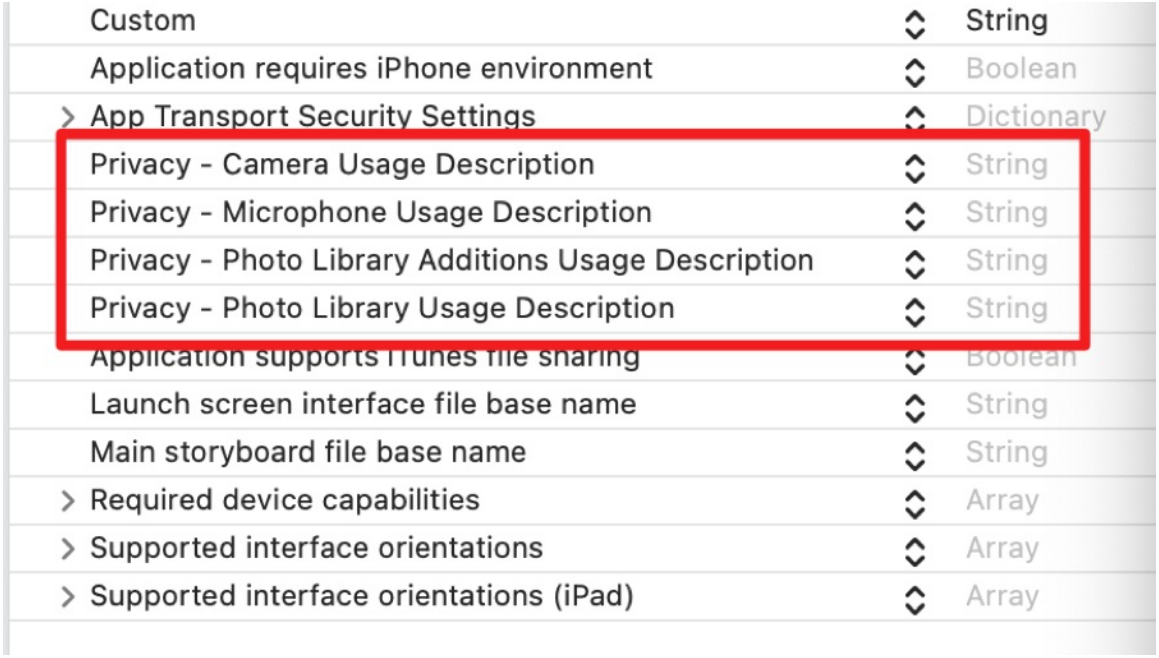
4、双击上图的CopyFiles，改为Embed Frameworks,同时Destination,改为framework,之后将AipBase.framework,AipOcrSdk.framework和IdcardQuality.framework拖动到下面，如下图所示：



5、Link Binary With Libraries中添加libc++.tbd、libz.tbd 如下图所示:



6、info.plist 文件中添加以下key



7、之后可以将人脸核验示例工程代码代码结合自身工程项目，将相关代码集成到目标工程中。

三、**关键代码说明** 1、人脸识别SDK鉴权代码如下,canwork方法返回YES，代表人脸SDK鉴权成功。

```
NSString* licensePath = [NSString stringWithFormat:@"%@.%@", FACE_LICENSE_NAME, FACE_LICENSE_SUFFIX ];
[[FaceSDKManager sharedInstance] setLicenseID:FACE_LICENSE_ID andLocalLicenceFile:licensePath
andRemoteAuthorize:true];
NSLog(@"canWork = %d",[[FaceSDKManager sharedInstance] canWork]);
NSLog(@"version = %@",[[FaceSDKManager sharedInstance] getVersion]);
```

2、OCR SDK鉴权代码如下

```
NSString *licenseFile = [[NSBundle mainBundle] pathForResource:FACE_API_ORC_KEY ofType:FACE_SECRET_OCR_KEY];
NSData *licenseFileData = [NSData dataWithContentsOfFile:licenseFile];
[[AipOcrService shardService] authWithLicenseFileData:licenseFileData];
NSLog(@"ocr sd version: %@", [AipOcrService ocrSdkVersion]);
```

点击开始身份认证，可以成功进入如下界面，没有报错，表示OCR鉴权成功

3、返回拍照页面，并开始离线识别身份证：

```
-(void) startOCRSdk {
  [self configCallback];
  // 身份证识别
  [AipCaptureCardVC clearIdCard];
  _vc =
  [AipCaptureCardVC ViewControllerWithCardType:CardTypeLocalIdCardFont
                              andImageHandler:^(UIImage *image) {
      _idCardImage = image;
      [[AipOcrService shardService] detectIdCardFrontFromImage:image
                                                    withOptions:nil
                                                 successHandler:^(id result){
          _successHandler(result);
      }
                                                    failHandler:_failHandler];
  }];
```

4、调起人脸识别界面，BDFaceColorfulViewController为人脸识别Controller。注意，如果不适用OCR部分，那么这里需要传入自定义的姓名和身份证号字符串。

```
 - (void)faceColor {
    NSArray * colorArr = [[NSArray alloc] initWithObjects:@(FaceLivenessActionTypeLiveEye),
@(FaceLivenessActionTypeLiveMouth), nil];
    int r = arc4random() % [colorArr count];
    BDFaceColorfulViewController* cvc = [[BDFaceColorfulViewController alloc] init];
    BDFaceLivingConfigModel* model = [BDFaceLivingConfigModel sharedInstance];
    [cvc livenesswithList:@[colorArr[r]] order:model.isByOrder numberOfLiveness:model.numOfLiveness];
    cvc.modalPresentationStyle = UIModalPresentationFullScreen;
    cvc.name = _idCardName;
    cvc.idCardNumber = _idCardNumber;
    [self presentViewController:cvc animated:YES completion:nil];
}
```

5、人脸识别完成结果回调函数

```
- (void)faceCallbackWithCode:(BDFaceCompletionStatus)status result:(NSDictionary *)result {

    NSLog(@">>>>>>>>>>>faceCallbackWithCode, %ld  %@", (long)status,[self showCompletionStatusWithType:status]);
    MyLog(@"result = %@",result);
    NSDictionary *dic = [self dictionaryWithJsonString:result[@"data"]];
    [[BDFaceImageShow sharedInstance] setAuraLiveColor:[NSString
stringWithFormat:@"licenseId:%@",dic[@"licenseId"]]];

    if (status == BDFaceCompletionStatusCameraStarted) {
        [self.videoCapture startRecordingVideo];
    } else if (status ==  BDFaceCompletionStatusImagesSuccess) {
        NSLog(@"%@", result);
    } else if (status == BDFaceCompletionStatusSuccess) {
        dispatch_async(dispatch_get_main_queue(), ^{
            [self.remindAnimationView stopActionAnimating];
            BDFaceVerificationController *avc = [[BDFaceVerificationController alloc] init];
            avc.modalPresentationStyle = UIModalPresentationFullScreen;
            avc.name = _name;
            avc.idCardNumber = _idCardNumber;
            avc.receivedData = result;
            if (@available(iOS 13.0, *)) {
                avc.modalPresentationStyle = 0;
            }
            [self closeAction];
            [self.presentingViewController presentViewController:avc animated:YES completion:nil];
        });

    }
```

注意：BDFaceCompletionStatusSuccess为采集结束，同时只有这个时候加密的信息才会在result中返回,BDFaceCompletionStatusImagesSuccess 只是流程上采集结束了，但是result是没有信息的，使用BDFaceCompletionStatusSuccess这个枚举即可。

6、动作活体返回状态回调：

```
- (void)livenessProcesssWithCode:(LivenessRemindCode)remindCode {

    dispatch_async(dispatch_get_main_queue(), ^{
        self.isAnimating = [self.remindAnimationView isActionAnimating];
    });

    switch (remindCode) {
        case LivenessRemindCodeOK:
```

7、炫彩返回状态回调：

```
- (void)colorfulProcesssWithCode:(ColorRemindCode)code imageInfo:(NSDictionary *)images {
    __weak typeof(self) weakSelf = self;

    NSLog(@">>>>>>>>> result code:%ld, info:%@", code, images);

    switch (code) {
      case ColorRemindCodeOK:
```

8、发请求BaseURL,如下图所示，开发者需要配置自己服务器的Host和端口号，进行相关请求；

```
#warning developer
    修改baseURL，BDFaceBaseSafeRequestBaseURL需要修改为指定的Server服务地
#define BDFaceBaseSafeRequestBaseURL @"http://10.138.38.77:8088"
```

9、进行姓名、身份证号和人脸比较，如下：

```
- (void)requestSafelyToCheckIdetifierCardAndPerson:(NSDictionary *)dic {
    NSString *finantialVerificationIdCardPersonImageUrl = [NSString stringWithFormat:@"%@%@",
BDFaceBaseSafeRequestBaseURL, @"/face/biz-demo/verify"];
    NSString *urlString = [NSString stringWithFormat:@"%@", finantialVerificationIdCardPersonImageUrl];
    NSMutableURLRequest *request = [NSMutableURLRequest requestWithURL:[NSURL URLWithString:urlString]];
    NSMutableDictionary *parameters = [NSMutableDictionary dictionary];
```

10、进行图片比对，一般是自身服务器中有图片底库，客户端采集完之后上传一张图片，这两张图片进行比较，注意：imageKey开发者自定义传入（也就是存在服务器中的照片底库标识码）

```
- (void)requestSafelyToCompareImages:(NSDictionary *)dic {
    NSString *finantialVerificationImageCompareUrl = [NSString stringWithFormat:@"%@%@",
BDFaceBaseSafeRequestBaseURL, @"/face/biz-demo/match"];
    NSString *urlString = [NSString stringWithFormat:@"%@", finantialVerificationImageCompareUrl];
    NSMutableURLRequest *request = [NSMutableURLRequest requestWithURL:[NSURL URLWithString:urlString]];
    NSMutableDictionary *parameters = [NSMutableDictionary dictionary];

    NSString *skey = dic[@"skey"];
    NSString *xDeviceId = dic[@"x-device-id"];
    NSString *theDataString = dic[@"data"];
    NSString *appPlatform = @"ios"; // ios 必须是小写，服务端定的
    parameters[BDFaceKeyOfSkey] = skey ?: @"";
    parameters[BDFaceKeyOfXdeviceId] = xDeviceId ?: @"";
    parameters[BDFaceKeyOfApp] = appPlatform ?: @"";

    NSDictionary *tempDataDic = [NSJSONSerialization JSONObjectWithData:[theDataString
dataUsingEncoding:NSUTF8StringEncoding] options:NSJSONReadingMutableLeaves error:nil];
    theDataString = tempDataDic[@"data"];
    parameters[BDFaceKeyOfData] = tempDataDic[@"data"] ?: @"";
```

warning developer 这里只是联调使用，imageKey需要根据自己代码确定，图片的唯一标识码是什么，不能使用如下一行的代码

```
NSString *imageKey = [[NSUserDefaults standardUserDefaults] objectForKey:@"imagekeyfortesting"];
    parameters[BDFaceKeyOfRegisterKey] = imageKey ?: @"";
```

11、活体检测接口调用，如下图所示：

```
- (void)requestSafelyToCheckLive:(NSDictionary *)dic {
    NSString *finantialVerificationCheckLiveUrl =  [NSString stringWithFormat:@"%@%@",
BDFaceBaseSafeRequestBaseURL, @"/face/biz-demo/faceverify"];
    NSString *urlString = [NSString stringWithFormat:@"%@", finantialVerificationCheckLiveUrl];
    NSMutableURLRequest *request = [NSMutableURLRequest requestWithURL:[NSURL URLWithString:urlString]];
    NSMutableDictionary *parameters = [NSMutableDictionary dictionary];
```

四、服务端接口参数说明 4.1 实名认证接口 -url：face/biz-demo/verify

-请求方式：Content-Type 为 application/json，通过 json 格式化请求体

-返回值类型：JSON

-请求参数说明

参数名	必选	类型	说明
skey	是	string	加固版上报的密钥信息 skey
x_device_id	是	string	加固版上报的密钥信息 deviceId
data	是	string	加密数据
id_card_number	是	string	身份证件号
name	是	string	姓名(需要是 utf8 编码)
verify_type	否	string	0:大陆身份证
app	是	string	APP 类型:ios/android/universe

-返回参数说明

参数名	类型	说明
log_id	number	调用的日志id
error_code	number	错误码，0通过，300002低分，300003高风险，300004身份证号与姓名不匹配或该身份证号不存在，300005公安网图片不存在或质量过低
error_msg	string	错误信息
result	jsonObject	认证返回的结果
+ verify_log_id	number	核验日志id，用于查询错误原因
+ dec_image	string	SDK传入的解密的图片

4.2 人脸对比接口 -url：face/biz-demo/match

-请求方式：Content-Type 为 application/json，通过 json 格式化请求体

-返回值类型：JSON

-参数说明

参数名	必选	类型	说明
skey	是	string	加固版上报的密钥信息 skey
x_device_id	是	string	加固版上报的密钥信息 deviceId
data	是	string	加密数据
image_key	是	string	对比图片key ，服务端根据图片key查询对比图片

-返回参数说明

参数名	类型	说明
log_id	number	调用的日志id
error_code	number	错误码，0通过，300002低分，300003高风险
error_msg	string	错误信息
result	jsonObject	认证返回的结果
+ verify_log_id	number	核验日志id，用于查询错误原因

4.3 图片活体接口参数 -url：face/biz-demo/faceverify

-请求方式：Content-Type 为 application/json，通过 json 格式化请求体

-返回值类型：JSON

-参数说明

参数名	必选	类型	说明
skey	是	string	加固版上报的密钥信息 skey
x_device_id	是	string	加固版上报的密钥信息 deviceId
data	是	string	加密数据
zid	否	string	风控参数：唯一标识 ZID
phone	否	string	风控参数：手机号
app	是	string	APP 类型:ios/android/universe

-返回参数说明

参数名	类型	说明
log_id	number	调用的日志id
error_code	number	错误码，0通过，300002低分，300003高风险
error_msg	string	错误信息
result	jsonObject	认证返回的结果
+ verify_log_id	number	核验日志id，用于查询错误原因
+ dec_image	string	SDK传入的解密的图片

五、人脸安全采集SDK接口文档 5.1 SSFaceSDK.h 包含其他头文件，及人脸扫描、活体验证状态码，同时还有执行业务流程的参数枚举等。没有一一列举，具体请参照头文件：

```
 ###### import "FaceSDKManager.h"
###### import "SSFaceDetectionManager.h"

// 参数枚举等
typedef NSString *FaceIdCardType NS_STRING_ENUM;
FOUNDATION_EXPORT FaceIdCardType const KFaceIdCardTypeDefault; // 默认 大陆身份证
FOUNDATION_EXPORT FaceIdCardType const KFaceIdCardTypeMTPIDCard; // 港澳居民来往内地通行证
FOUNDATION_EXPORT FaceIdCardType const KFaceIdCardTypeFPRIDCard; // 外国人永久居留身份证
FOUNDATION_EXPORT FaceIdCardType const KFaceIdCardTypePassport; // 定居国外的中国公民护照

// 活体检测返回状态
typedef NS_ENUM(NSUInteger, LivenessRemindCode) {
    LivenessRemindCodeOK = 0,  //成功
    LivenessRemindCodeBeyondPreviewFrame,  //出框
    LivenessRemindCodeNoFaceDetected, //没有检测到人脸
    LivenessRemindCodeMuchIllumination,
    LivenessRemindCodePoorIllumination,  //光照不足
    LivenessRemindCodeImageBlured,  //图像模糊
    ...
};
```

5.2 FaceSDKManager.h

图像采集行为和过程中需要的设置，属于全局配置。在原IDL库中FaceSDKManager基础上增加如下两个接口：

```
 /**
 *  SDK云端校验设置
 *  需要云端校验，需提前申请id和secret
 *
 *  @param clientId api key
 *  @param clientSecret api secret
 */
- (void)setBCEClientId:(NSString *)clientId clientSecret:(NSString *)clientSecret;


/**
 *  SDK鉴权方法-文件授权
 *  SDK鉴权方法 必须在使用其他方法之前设置，否则会导致SDK不可用
 *
 *  @param licenseID 授权ID
 *  @param licensePath 本地鉴权文件路径
 *  @param remoteAuthorize 是否远程更新过期鉴权文件
 */
- (void)setLicenseID:(NSString *)licenseID andLocalLicenceFile:(NSString *)licensePath andRemoteAuthorize:
(BOOL)remoteAuthorize;
```

5.3 SSFaceDetectionManager.h

用于进行人脸识别具体行为动作。

```
 @interface SSFaceDetectionManager : NSObject
// 图像返回帧处理代理
@property (nonatomic, weak) id<SSCaptureDataOutputProtocol> delegate;

// 采集流程运行状态
@property (nonatomic, assign, readonly) BOOL runningStatus;

// 风险检测超时时间：-1 关闭风险检测；大于0 风险检测超时时间；0 使用默认超时时间3秒
@property (nonatomic, assign) NSInteger riskDetectionSetting;
```

```
// 设置使用镜头，前置/后置
@property (nonatomic, assign) AVCaptureDevicePosition devicePosition;

// 输出形式，AVCaptureSessionPreset类型
@property (nonatomic, copy) NSString *sessionPresent;

// 采集图像区域
@property (nonatomic, assign) CGRect previewRect;

// 探测区域
@property (nonatomic, assign) CGRect detectRect;

// 是否开启声音提醒
@property (nonatomic, assign) BOOL enableSound;

// BDFaceDetectionTypeColorfulLiveness流程中是否开启动作活体，默认开启
@property (nonatomic, assign) BOOL enableLivenessInColorfulFlow;

+ (instancetype)sharedInstance;
- (void)connectPreviewLayer:(AVCaptureVideoPreviewLayer*)pLayer;
/**
 * 创建用于实名认证的参数
 */
- (NSDictionary *)creatFaceVerifyParameters:(NSString *)idCardNumber
                                       name:(NSString *)name
                                 verifyType:(FaceIdCardType)cardType
                                     nation:(NSString *)nation
                                phoneNumber:(NSString *)phoneNumber
                           livenessControl:(FaceLivenessControlType)livenessControl
                           spoofingControl:(FaceSpoofingControlType)spoofingControl
                            qualityControl:(FaceQualityControlType)qualityControl;

/**
 * 创建用于人脸比对的参数
 */
- (NSDictionary *)createFaceMatchParametersWithRegisterImage:(NSString *)registerImageBase64
                                        registerImageType:(FaceRegisterImageType)registerImageType
                                         registerFaceType:(FaceFaceType)registerFaceType
                                                faceType:(FaceFaceType)faceType
                                            faceSortType:(FaceSortype)faceSortType
                                             phoneNumber:(NSString *)phoneNumber
                                          livenessControl:(FaceLivenessControlType)livenessControl
                                          qualityControl:(FaceQualityControlType)qualityControl
                                    registerLivenessControl:(FaceLivenessControlType)registerLivenessControl
                                     registerQualityControl:(FaceQualityControlType)registerQualityControl;

/**
 * 创建用于视频录制的参数
 */
- (NSDictionary *)createVideoRecordParametersWithEnableVideoRecording:(BOOL)enableVideoRecording
                                                enableVideoSound:(BOOL)enableVideoSound
                                                   videoFileName:(NSString *)videoFileName
                                                      imageWidth:(NSUInteger)imageWidth
                                                     imageHeight:(NSUInteger)imageHeight;
/**
 * 开始当前人脸校验流程
 * @param detectionType 业务流程，采集信息用于实名认证、人脸比对
 * @param parameters 流程需要的参数
 * @param flowType 操作流程，人脸采集、人脸活体
 * @param vc 用于进行人脸信息采集的ViewController
 */
- (void)startSessionWithType:(BDFaceResultReportType)detectionType
```

```
                parameters:(NSDictionary *)parameters
                  faceFlow:(BDFaceFlowType)flowType
            viewController:(UIViewController *)vc;


/**
 * 开始当前人脸校验流程，供视频录制时传入参数使用
 * @param detectionType 业务流程，采集信息用于实名认证、人脸比对
 * @param parameters 流程需要的参数
 * @param flowType 操作流程，人脸采集、人脸活体
 * @param vc 用于进行人脸信息采集的ViewController
 * @param videoParameters 需要录制视频的参数
 * @param cameraAutoClose 镜头是否自动关闭
 */
- (void)startSessionWithType:(BDFaceResultReportType)detectionType
                parameters:(NSDictionary *)parameters
                  faceFlow:(BDFaceFlowType)flowType
            viewController:(UIViewController *)vc
           videoParameters:(NSDictionary *)videoParameters
           cameraAutoClose:(BOOL)cameraAutoClose;

/**
 * 取消当前人脸校验流程
 */
- (void)cancel;

/**
 * 开始视频录制，请与BDFaceCompletionStatusCameraStarted回调后调用
 */
- (void)startRecordingVideo;

/**
 * 结束当前录制，活体验证session流程结束时自动结束，不需要主动调用；BDFaceDetectionTypeVideoRecording 时主动调用
 */
- (void)stopRecordingVideo;

/**
 * 取消录制
 */
- (void)cancelRecording;

/**
 * 活体检测过程中，返回活体总数，当前成功个数，当前活体类型
 */
-(void)livenessProcessHandler:(LivenessProcess)process;

/**
 * 返回无黑边的方法
 * @param array 活体动作数组
 * @param order 是否顺序执行
 * @param numberOfLiveness 活体动作个数
 */
- (void)livenesswithList:(NSArray *)array order:(BOOL)order numberOfLiveness:(NSInteger)numberOfLiveness;
```

5.4 通过startSession方法开始人脸采集后，通过SSCaptureDataOutputProtocol类型的delegate进行回调

```
 /**
 * 流程返回结果类型
 */
typedef NS_ENUM(NSInteger, BDFaceCompletionStatus) {
    BDFaceCompletionStatusSuccess = 1,  // 成功
    BDFaceCompletionStatusNoRisk = 2,   // 无风险
    BDFaceCompletionStatusImagesSuccess = 3, // 图像采集成功
    BDFaceCompletionStatusCameraStarted = 4, // 相机开始

    BDFaceCompletionStatusIsRunning = -1,   // 正在采集图像
    BDFaceCompletionStatusResultFail = -2,  // 云端服务执行失败
    BDFaceCompletionStatusIsRiskDevice = -3,    // 风险设备
    BDFaceCompletionStatusCameraError = -5,    // 没有授权镜头
    BDFaceCompletionStatusTimeout = -6, // 超时
    BDFaceCompletionStatusCancel = -7, // 取消
    BDFaceCompletionStatusVideoRecordingFail = -8, // 视频录制错误
    BDFaceCompletionStatusColorMatchFailed = -9, // 炫彩色彩错误
    BDFaceCompletionStatusVideoColorScoreFailed = -10, // 炫彩分数错误
    BDFaceCompletionStatusSDKNotInit = -13, // SDK未初始化
    BDFaceCompletionStatusLicenseFail = -15, // 授权错误
    BDFaceCompletionStatusNetworkError = -16,    // 网络错误
};

@protocol SSCaptureDataOutputProtocol <NSObject>

// 帧图像回传，用于刷新UI使用
- (void)captureOutputSampleBuffer:(UIImage *)image;

// 本次人脸流程回调
- (void)faceSessionCompletionWithStatus:(BDFaceCompletionStatus)status result:(NSDictionary *)result;

// 活体检测状态
- (void)livenessActionDidFinishWithCode:(LivenessRemindCode)code;

// 人脸识别检测
- (void)detectionActionDidFinishWithCode:(DetectRemindCode)code;

// 炫彩活体状态回调
- (void)colorfulActionDidFinishWithCode:(ColorRemindCode)code imageInfo:(NSDictionary *)imageInfo;

@end
```

5.5 使用流程

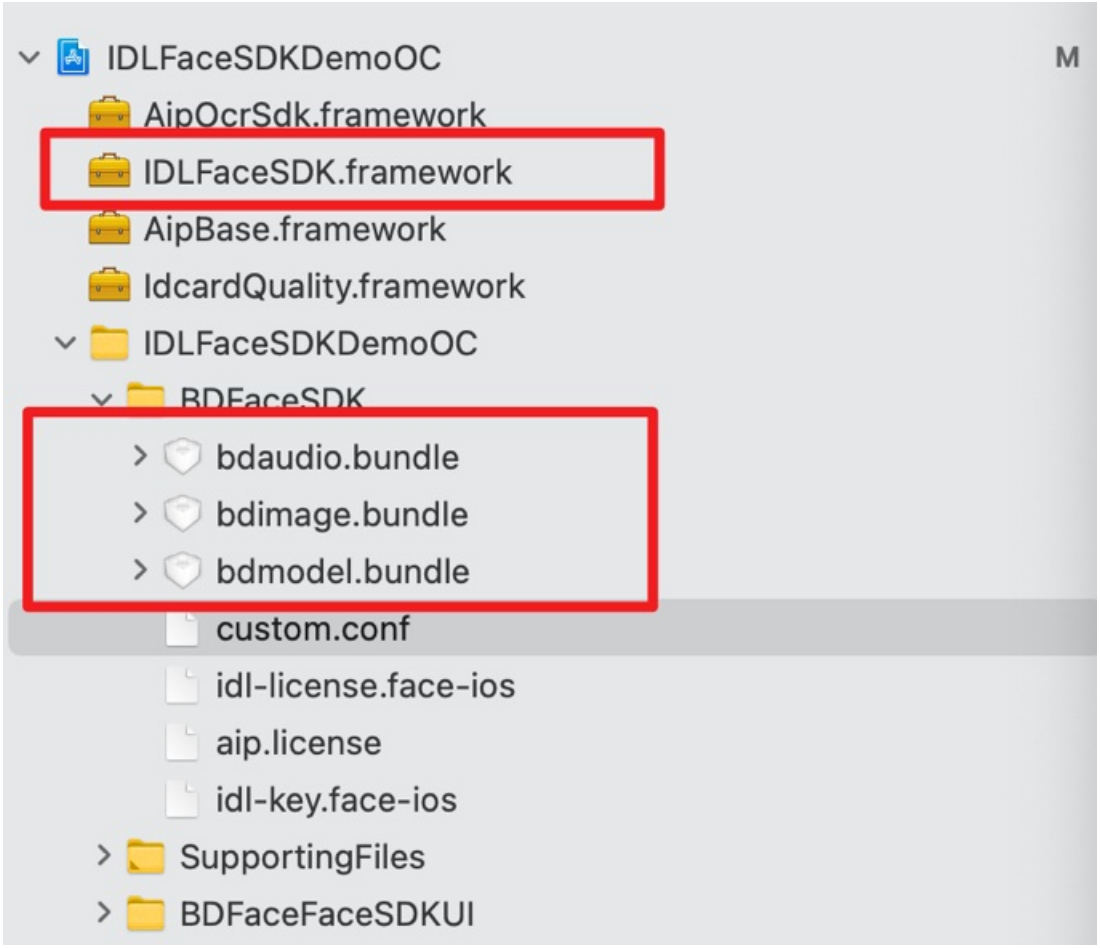
具体代码请参考Demo，以下是流程概述：

```
 // 1. 初始化Face License相关
NSString* licensePath = [NSString stringWithFormat:@"%@.%@", FACE_LICENSE_NAME, FACE_LICENSE_SUFFIX ];
[[FaceSDKManager sharedInstance] setLicenseID:FACE_LICENSE_ID andLocalLicenceFile:licensePath
andRemoteAuthorize:YES];

// 2. 初始化Face相关设置属性
[[FaceSDKManager sharedInstance] setColorJudgeAbility:false]; //炫彩颜色判断能力
[[FaceSDKManager sharedInstance] setMaxCropImageNum:1];  // 设置照片采集张数，【注意：推荐设置成1张】

// 3. 设置所需动作活体动作
[[SSFaceDetectionManager sharedInstance] livenesswithList:_livenessArray order:_order
numberOfLiveness:_numberOfLiveness];

// 4. 开始炫彩活体流程
NSDictionary *parameters =  [self.videoCapture creatFaceVerifyParameters:@"身份证" name:@"姓名"
verifyType:KFaceIdCardTypeDefault nation:nil phoneNumber:@"电话" livenessControl:nil spoofingControl:nil
qualityControl:nil]; // 业务参数

NSDictionary *videoParas = [self.videoCapture createVideoRecordParametersWithEnableVideoRecording:YES
enableVideoSound:YES videoFileName:@"whatmean" imageWidth:480 imageHeight:640]; // 视频录制参数

[self.videoCapture startSessionWithType:BDFaceResultReportTypeVerifySec parameters:parameters
faceFlow:BDFaceDetectionTypeColorfulLiveness viewController:self videoParameters:videoParas cameraAutoClose:YES];
// 开始流程，【注意：faceFlow参数为Colorful】

// 5. 接收回调
- (void)colorfulActionDidFinishWithCode:(ColorRemindCode)code imageInfo:(NSDictionary *)imageInfo {
   // 处理炫彩返回状态提醒
}

- (void)faceSessionCompletionWithStatus:(BDFaceCompletionStatus)status result:(NSDictionary *)result{
// 处理整体流程状态
  if (status == BDFaceCompletionStatusCameraStarted) {
    [self.videoCapture startRecordingVideo]; //  【注意：需要调用方法启动视频录制】
  } else {
    // 其他处理方式请参考demo
  }
}
```

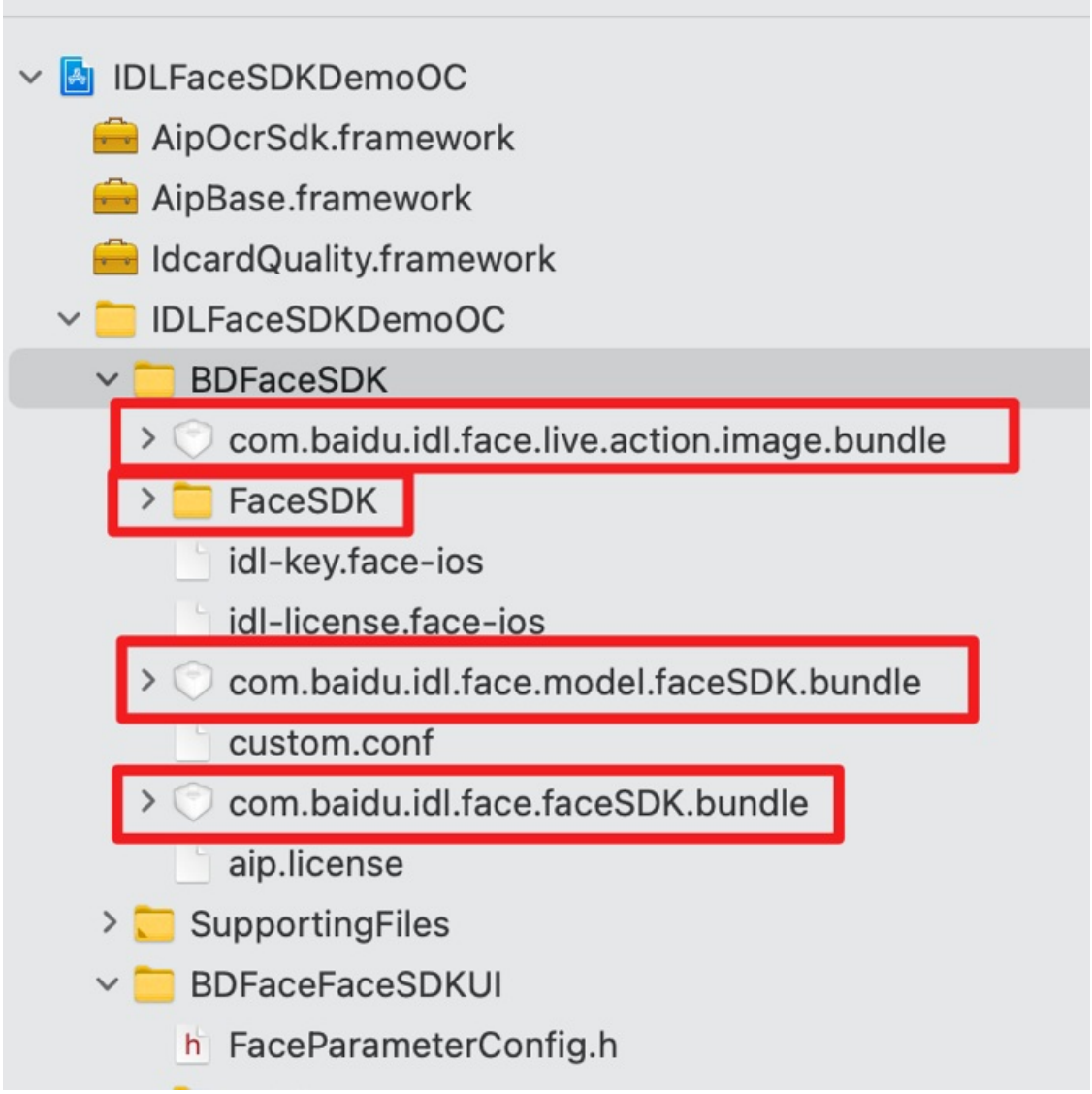
六、原金融级APP方案升级文档

1、从console平台下载新金融版示例工程代码，运行代码；

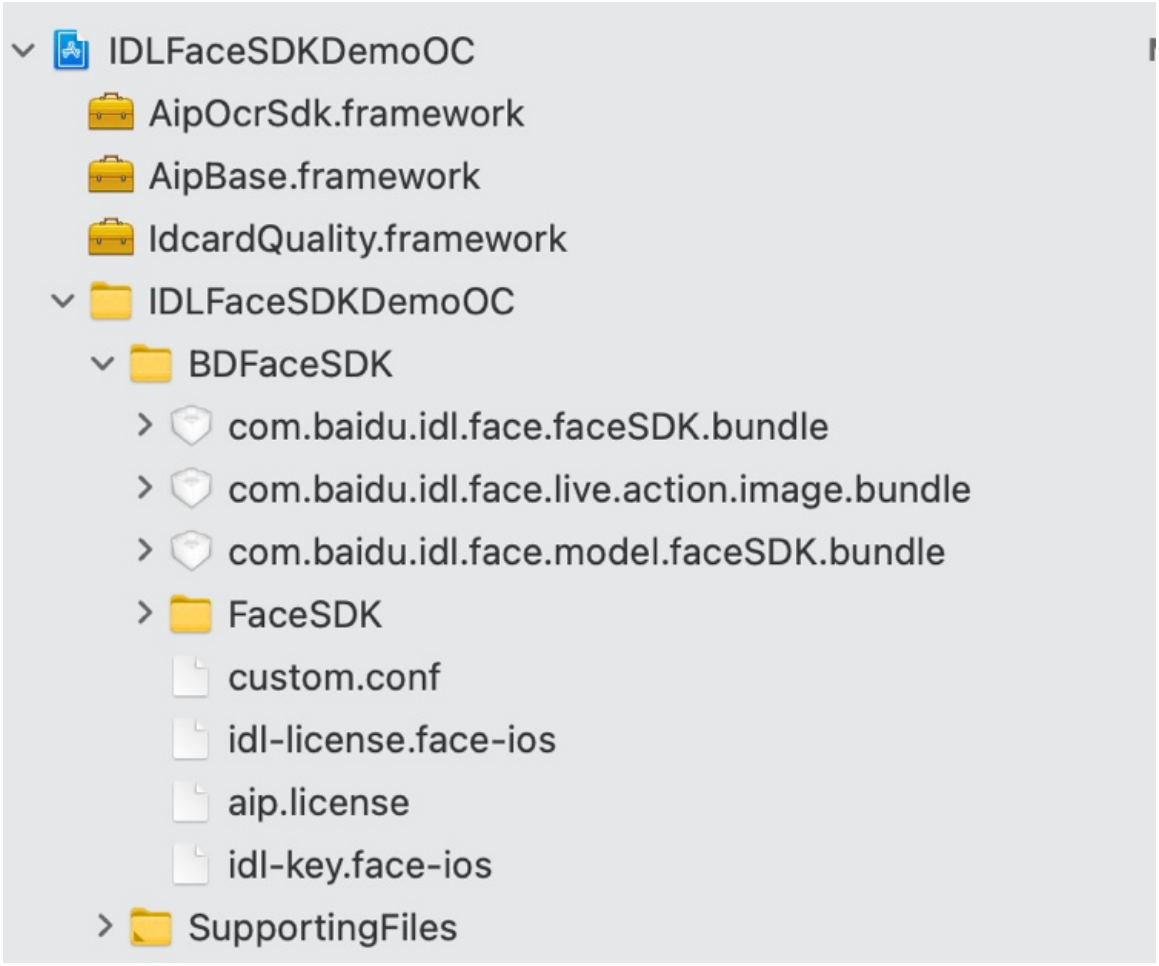
2、删除需要升级的项目中的如下framework和bundle



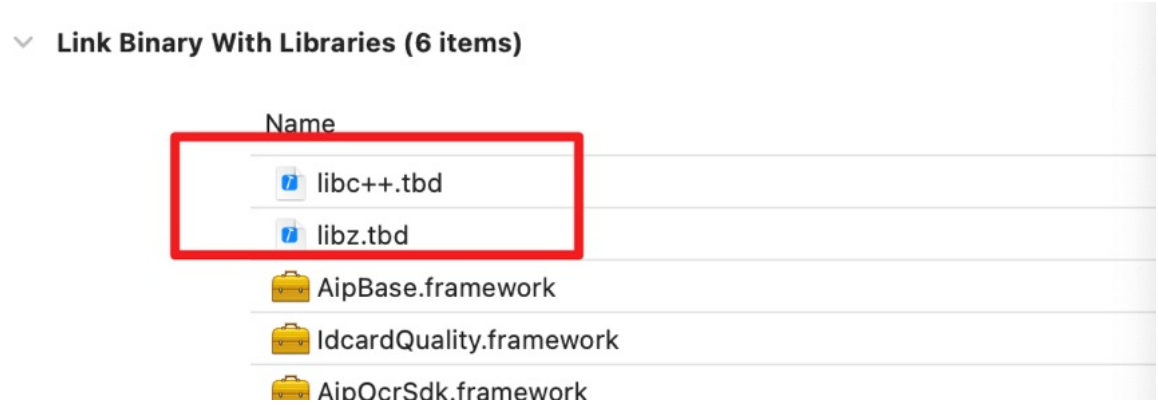
3、将示例工程代码中的如下文件，拷贝到现有工程的BDFaceSDK文件夹目录下；


IDLFaceSDKDemoOC
AipOcrSdk.framework
AipBase.framework
IdcardQuality.framework
IDLFaceSDKDemoOC
BDFaceSDK
com.baidu.idl.face.live.action.image.bundle
FaceSDK
idl-key.face-ios
idl-license.face-ios
com.baidu.idl.face.model.faceSDK.bundle
custom.conf
com.baidu.idl.face.faceSDK.bundle
aip.license
SupportingFiles
BDFaceFaceSDKUI
FaceParameterConfig.h

4、现有工程目录如下：

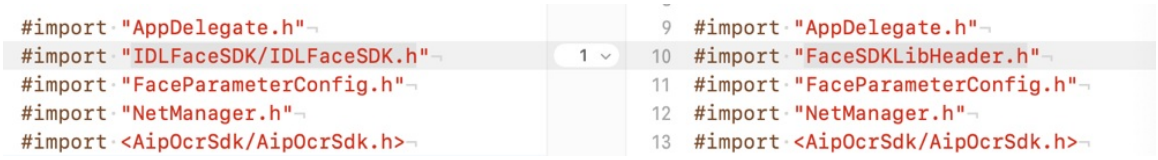


5、Build Phases中添加libc++.tbd和libz.tbd两个库文件，如下图所示：



6、infoplist文件添加key值 Privacy - Microphone Usage Description，如下图所示：

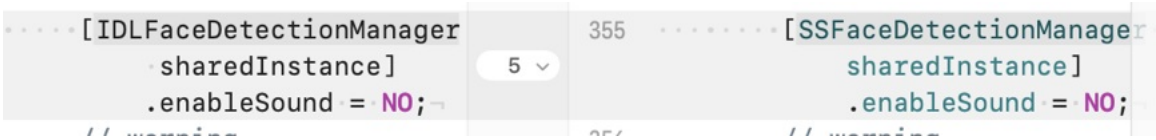
7、搜索#import "IDLFaceSDK/IDLFaceSDK.h" 和 #import <IDLFaceSDK/IDLFaceSDK.h> 替换为 #import "FaceSDKLibHeader.h" 如下图所示：



8、删除BDFaceLivenessViewController和BDFaceDetectionViewController的.h和.m文件,并删除文中的所有代码引用。

9、删除代码 [IDLFaceLivenessManager sharedInstance].enableSound = YES;

10、全局搜索 IDLFaceDetectionManager 替换为SSFaceDetectionManager 如下图所示：



11、因为改动较大，建议整体替换 BDFaceBaseViewController 和 BDFaceColorfulViewController 的.h和.m文件；

12、关键流程介绍：

（1）开始识别代码：

```
- (void)startRecognize {
    NSDictionary *parameters =  [self.videoCapture creatFaceVerifyParameters:self.idCardNumber name:self.name verifyType:KFaceIdCardTypeDefault nation:nil phoneNumber:nil livenessControl:nil spoofingControl:nil qualityControl:nil];
    NSDictionary *videoParas = [self.videoCapture createVideoRecordParametersWithEnableVideoRecording:YES enableVideoSound:YES videoFileName:@"whatmean" imageWidth:480 imageHeight:640];

    [self.videoCapture startSessionWithType:BDFaceResultReportTypeVerifySec parameters:parameters faceFlow:BDFaceDetectionTypeColorfulLiveness viewController:self videoParameters:videoParas cameraAutoClose:YES];
}
```

（2）识别成功回调：

```
- (void)faceCallbackWithCode:(BDFaceCompletionStatus)status result:(NSDictionary *)result
```

注意：result为识别成功后，结果的回调，状态码BDFaceCompletionStatusSuccess为整个采集流程成功结束，此时，result才有数据；BDFaceCompletionStatusImagesSuccess仅仅表示采集动作结束，此时result没有数据。

（3） 动作活体结束：

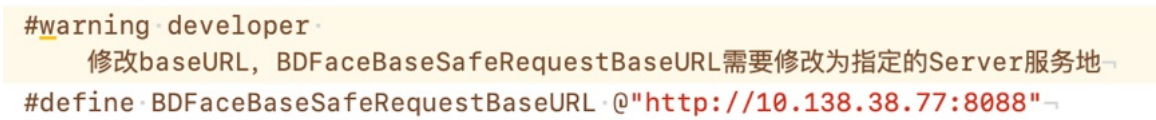
`- (void)livenessProcesssWithCode:(LivenessRemindCode)remindCode`

（4）炫彩活体流程回调：`- (void)colorfulProcesssWithCode:(ColorRemindCode)code imageInfo:(NSDictionary *)images`

13、运行修复剩余error，根据现有项目改动代码，运行代码；

14、修改BDFaceVerificationController.m，根据自身业务使用情况，将人证核验、人脸比对、活体检测请求代码移动到现有项目中；

15、全局搜索#warning developer，将BDFaceBaseSafeRequestBaseURL替换为自身服务器地址，如下图所示的代码：



16、请求接口改动，最新的流程使用了 SDK->自定义的服务器地址->请求百度人脸服务，请求部分代码接口在BDFaceVerificationController.m查看，具体有以下几个接口： （1） 人证核验接口 `- (void)requestSafelyToCheckIdetifierCardAndPerson:(NSDictionary *)dic`

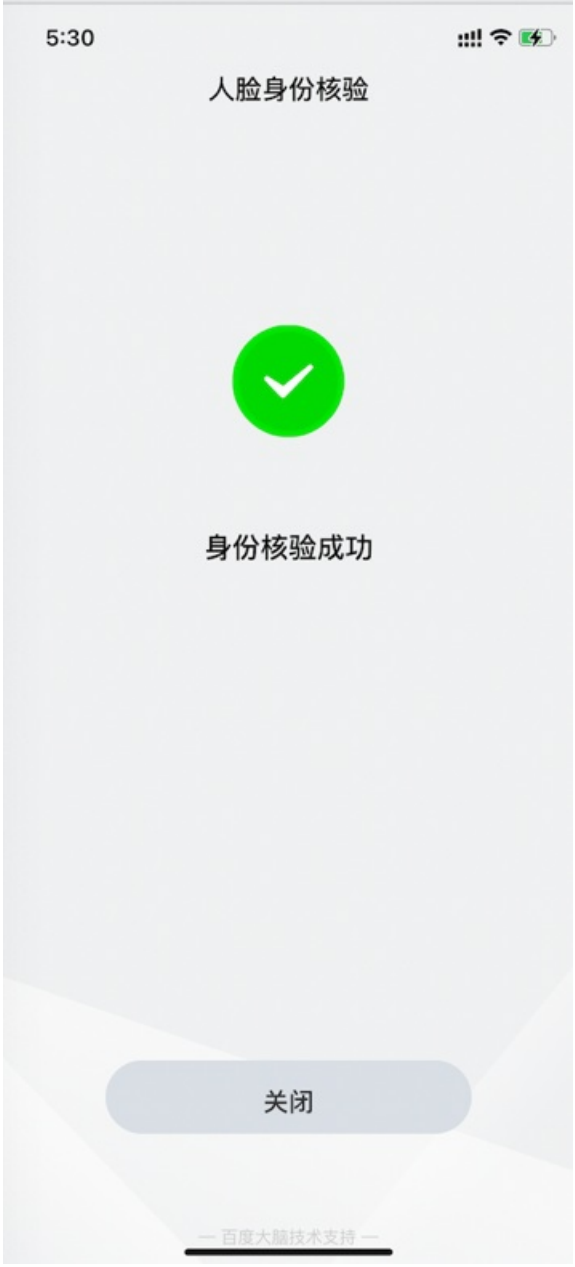
（2） 图片比对接口 `- (void)receiveDataFromServerForCompareImage:(NSDictionary *)dic`

（3）活体检测接口 `- (void)receiveDataFromServerForCheckAlive:(NSDictionary *)dic`

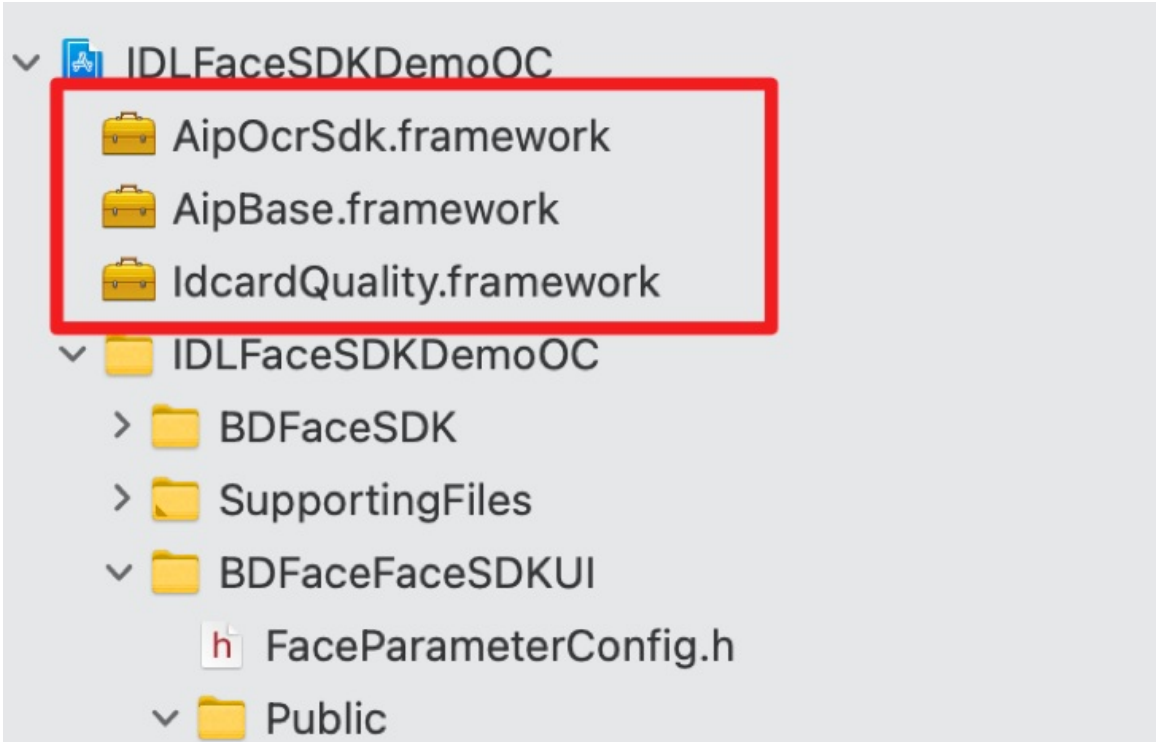
17、根据自身业务的使用，修改剩余代码。

七、不使用OCR及身份信息录入

1、下载完示例工程，配置Server地址，运行示例工程代码；点击开始身份核验，走完所有流程，可以出现如下界面：



2、删除如下图所示的库文件：



3、全局搜索 #import <AipOcrSdk/AipOcrSdk.h>，并删除所有引用；

4、搜索 -(void) startOCRSdk，注释如下函数：

```
-(void) startOCRSdk {
    [self configCallback];
    // 身份证识别
    [AipCaptureCardVC clearIdCard];                                    Use of u
    _vc =
    [AipCaptureCardVC ViewControllerWithCardType:CardTypeLocalIdCardF
                                    andImageHandler:^(UIImage *image) {
        _idCardImage = image;
        [[AipOcrService shardService] detectIdCardFrontFromImage:image
                                                      withOptions:nil
                                                   successHandler:^(id
            _successHandler(result);
        }
                                                      failHandler:_fai
    }];
    if (@available(iOS 13.0, *)) {
        _vc.modalPresentationStyle = 0;
    }
    [self presentViewController:_vc animated:YES completion:nil];
}
```

5、搜索“OCR 能力加载“，注释以下代码:

```
// OCR 能力加载
NSString *licenseFile = [[NSBundle mainBundle] pathForResource:FACE_API_ORC_KE
    ofType:FACE_SECRET_OCR_KEY];
NSData *licenseFileData = [NSData dataWithContentsOfFile:licenseFile];
[[AipOcrService shardService] authWithLicenseFileData:licenseFileData];  U
NSLog(@"ocr sd version: %@", [AipOcrService ocrSdkVersion]);  Use of undeclare
```

6、在- (IBAction)startGatherAction:(UIButton *)sender 里注释以下代码：

```
    AppDelegate *appDelegate = (AppDelegate*) [[UIApplication sharedApplication]delegate];
    if (appDelegate.useOCR == 1) {
        // OCR 身份证识别页面
        [self startOCRSdk];
    } else {
        // 手动输入页面
        BDFaceManualPersonalDataConfirmController * ocr = [[BDFaceManualPersonalDataConfirmController alloc] init];
        if (@available(iOS 13.0, *)) {
            ocr.modalPresentationStyle = 0;
        }
        [self presentViewController:ocr animated:YES completion:nil];
}
```

7、在- (IBAction)startGatherAction:(UIButton *)sender 方法里添加如下代码：

```
 NSArray * colorArr = [[NSArray alloc] initWithObjects:@(FaceLivenessActionTypeLiveEye),
@(FaceLivenessActionTypeLiveMouth), nil];
    int r = arc4random() % [colorArr count];
    BDFaceColorfulViewController* cvc = [[BDFaceColorfulViewController alloc] init];
    BDFaceLivingConfigModel* model = [BDFaceLivingConfigModel sharedInstance];
    [cvc livenesswithList:@[colorArr[r]] order:model.isByOrder numberOfLiveness:model.numOfLiveness];
    cvc.modalPresentationStyle = UIModalPresentationFullScreen;
###### warning developer 这里需要加入要识别的用户的姓名和身份证号
    cvc.name = @"";
    cvc.idCardNumber = @"";
[self presentViewController:cvc animated:YES completion:nil];
```

注意：上面的方法里的cvc.name = @"";和cvc.idCardNumber = @""; 引号里需要写上真真实的姓名和身份证，供之后采集人脸后，身份信息和人脸进行比对。

8、最终的- (IBAction)startGatherAction:(UIButton *)sender 函数如下：

```
#pragma mark - Button Action
- (IBAction)startGatherAction:(UIButton *)sender{
    [BDFaceVerificationConstant sharedInstance].verifyType =
        BDFaceVerificationIdCardAndImageVerification;
    NSNumber *checkAgree = [[NSUserDefaults standardUserDefaults]
        objectForKey:@"checkAgreeBtn"];
    if (!checkAgree.boolValue){
        [[JSToastDialogs shareInstance] makeToast:@"请先同意《人脸验证协议》" duration:1.0];
        return;
    }

    NSArray * colorArr = [[NSArray alloc]
        initWithObjects:@(FaceLivenessActionTypeLiveEye),
        @(FaceLivenessActionTypeLiveMouth), nil];
    int r = arc4random() % [colorArr count];
    BDFaceColorfulViewController* cvc = [[BDFaceColorfulViewController alloc] init];
    BDFaceLivingConfigModel* model = [BDFaceLivingConfigModel sharedInstance];
    [cvc livenesswithList:@[colorArr[r]] order:model.isByOrder
        numberOfLiveness:model.numOfLiveness];
    cvc.modalPresentationStyle = UIModalPresentationFullScreen;
#warning developer 这里需要加入要识别的用户的姓名和身份证号                    Developer
    cvc.name = @"";
    cvc.idCardNumber = @"";
    [self presentViewController:cvc animated:YES completion:nil];
}
```

9、在Viewcontroller.m 文件的头文件中，加入引用：

import "BDFaceColorfulViewController.h"

10、运行工程，点击下面的Button开始身份认证按钮即可开始人证核验过程。

Android-方案集成指南

本文档介绍了金融级APP实名认证方案Android端APP集成开发流程。

一、运行示例工程

1、Demo运行 从控制台下载示例工程包之后，先将AndroidManifest.xml中的com.baidu.idl.face.demo.liantian.ac.provider修改为您的包名，build.gradle中的applicaitonId.liantian.ac.provider。然后在build.gradle中配置好包名和签名。

```
android {
  defaultConfig {
      ...
      applicationId "申请时的包名"
      ...
  }
  signingConfigs {
      def password = "替换为签名密码"
      def alias = "替换为签名别名"
      def filePath = "替换为签名文件路径"
      keyAlias alias
      keyPassword password
      storeFile file(filePath)
      storePassword(password)
  }
  release {
      keyAlias alias
      keyPassword password
      storeFile file(filePath)
      storePassword(password)
  }
}
```

2、SDK集成 首先在app工程中导入demo工程中的faceplatform-ui 模块、lib-liantian模块，ocr_ui模块（如果需要使用OCR身份证识别能力，则需要导入，不需要则不导入）。

在app工程的build.gradle中添加依赖，引入faceplatform-ui和ocr_ui依赖，然后sync。

```
dependencies {
 compile project(': faceplatform-ui)
 compile project(': ocr_ui)
}
```

3、授权文件、加密文件

请将创建App方案时获取的人脸授权文件(idl-license.face-android)、加密文件（idl-key.face-android）放置于Assets目录下。如果使用OCR身份证识别功能，请将OCR身份证识别授权文件（aip.license）也放置于Assets目录下。

二、安全人脸SDK接口 1、隐私协议接口

用户同意隐私协议之后调用此接口，在同意隐私协议前，所有人脸能力的调用均会失败，直到同意了隐私协议。

`void LH.setAgreePolicy(Context context, boolean agree)`

-参数说明

agree：用户是否同意了隐私协议，true为同意，false为不同意。

2、安全能力初始化接口 在开始采集人脸的流程前，调用安全能力初始化接口。请保证在调用此方法之前请确认setAgreePolicy方法已经调用，并且agress为true，否则采集流程将无法启动。

`void LH.init(Context context, String licenseId);`

-参数说明

licenseId：Face为App分配的许可Id，在AI开放平台创建App时获取。

3、人脸能力初始化接口 在开始采集人脸的流程前，调用人脸能力初始化接口。请保证在调用此方法之前LH.init方法已经调用。

```
FaceSDKManager.getInstance().initialize(mContext, Config.licenseID,
        Config.licenseFileName, new IInitCallback() {
            @Override
            public void initSuccess() {}

            @Override
            public void initFailure(final int errCode, final String errMsg) {
                runOnUiThread(new Runnable() {});
            }
        });
```

-参数说明：

licenseId：Face为App分配的许可Id，在AI开放平台创建App时获取。

licenseFileName：Face为App分配的授权文件名称，默认名称为idl-license.face-android。

new IInitCallback()：见“初始化IInitCallback回调接口”。

4、初始化IInitCallback回调接口 人脸能力初始化接口回调，错误码和错误信息如下表。

错误码	说明
SUCCESS（0）	用户取消操作
LICENSE_NOT_INIT_ERROR（1）	license未初始化
LICENSE_DECRYPT_ERROR（2）	license数据解密失败
LICENSE_INFO_FORMAT_ERROR（3）	license数据格式错误
LICENSE_KEY_CHECK_ERROR（4）	license-key校验错误
LICENSE_ALGORITHM_CHECK_ERROR（5）	算法ID校验错误
LICENSE_MD5_CHECK_ERROR（6）	MD5校验错误
LICENSE_DEVICE_ID_CHECK_ERROR（7）	设备ID校验错误
LICENSE_PACKAGE_NAME_CHECK_ERROR（8）	包名(应用名)校验错误
LICENSE_EXPIRED_TIME_CHECK_ERROR（9）	过期时间不正确
LICENSE_FUNCTION_CHECK_ERROR（10）	功能未授权
LICENSE_TIME_EXPIRED（11）	授权已过期
LICENSE_LOCAL_FILE_ERROR（12）	本地文件读取失败
LICENSE_REMOTE_DATA_ERROR（13）	远程数据拉取失败
LICENSE_LOCAL_TIME_ERROR（14）	本地时间校验错误
OTHER_ERROR（other）	其他错误

5、人脸配置 设置配置参数，如不设置，将使用默认值。

人脸配置通过console平台下发，可以参考demo工程assets目录下的console_config.json文件，相关配置代码参考demo的app模块。

```
FaceConfig config = FaceSDKManager.getInstance().getFaceConfig();
FaceSDKManager.getInstance().setFaceConfig(config);
```

6、人脸采集流程（SurfaceView实现） 该方法是异步方法，调用后执行炫彩活体采集流程，该流程需要一个在界面显示的SurfaceView实现。可以参考demo的faceplatform-ui模块。

```
boolean LH.startFaceSurfaceLiveness(Activity activity,SurfaceHolder previewSurfaceHolder, FaceProcessCallback callback, int deviceCheckTimeout)
```

-参数说明：

activity：执行人脸采集的Activity。

previewSurfaceHolder：用于展示摄像头预览的SurfaceView的Holder。

processCallback：流程回调，见“FaceProcessCallback接口”。

deviceCheckTimeout：安全检测超时时间，此处需要传0，开启后台检测，传-1则跳过检测。

-返回值说明：

若本方法返回false，则流程启动失败，此时传入的callback对象的onEnd方法会回调失败原因。

7、人脸采集流程（TextureView实现） 该方法是异步方法，调用后执行炫彩活体采集流程，该流程需要一个在界面显示的TextureView实现。可以参考demo的faceplatform-ui模块。

```
boolean LH. startFaceTextureLiveness(Activity activity, TextureView previewTextureView, FaceProcessCallback callback, int deviceCheckTimeout, IFaceProcessInfo info, boolean record)
```

-参数说明：

activity：执行人脸采集的Activity。

TextureView：用于展示摄像头预览的TextureView。

processCallback：流程回调，见“采集回调FaceProcessCallback接口”。

deviceCheckTimeout：安全检测超时时间，此处需要传0，开启后台检测，传-1则跳过检测。

record：是否进行视频录制，若否，则onEnd返回的RequestInfo中path为null，若是，该path字段为加密视频文件的路径。注意，本SDK不负责删除该文件，请在使用后删除该加密视频文件，以免硬盘被过度占用。

-返回值说明：

若本方法返回false，则流程启动失败，此时传入的callback对象的onEnd方法会回调失败原因。

8、采集回调FaceProcessCallback接口 回调类，回调方法顺序随人脸采集流程自上向下：

（1）onDeviceCheckResult (int status)

回调设备风险检测结果。无风险或未知风险会进入人脸采集流程，若有风险，则不会进行人脸采集流程。此回调方法return后，会直接回调onEnd方法。

status：1代表无风险，-1代表有风险。

（2）回调人脸采集过程的阶段性结果，供APP刷新界面。

status：按FaceStatusNewEnum的枚举值返回的当前状态。

message：需要在界面展示的提示。

livenessScore：活体检测分数。

```
 onCollectCompletion(FaceStatusNewEnum status, String message,                HashMap<String, ImageInfo> base64ImageCropMap, HashMap<String, ImageInfo> base64ImageSrcMap,int currentLivenessCount，float livenessScore);
```

（3）onEnd (int status，RequestInfo info)

必有，包括被Cancel的情况，回调流程结束事件。

status：1代表正常结束， <0的值均表示未正常结束。

info：只有在正常结束时有值，其他情况为null。本对象包含sKey，xDeviceId，data和path共4个字段。其中，sKey， xDeviceId为校验用，data为安全相关数据（该数据包含采集的最终图像，以及解密视频文件的必要信息），path为最终加密后的视频文件（仅在Texture实现的活体采集流程，开启录像功能时有值）。请按需使用这些数据。

onEnd的部分status值列举如下：

错误码	说明
1	正常结束，返回云端验证结果
-1	已经有一个采集验证流程在运行
-2	云端验证过程异常
-3	风控验证失败
-4	更严格情形下的风控验证失败
-5	摄像头异常
-6	流程被取消
-7	线程异常
-8	筛选图像异常
-9	采集前流程异常
-10	活体验证步骤异常
-11	预览异常
-12	采集后流程异常
-13	安全能力未初始化
-14	未同意隐私协议
-15	未成功加载安全模块
-17	录制异常
-18	未检出人脸超时
-19	网络异常
-20	炫彩异常
-21	活体分数异常

9、取消正在进行的流程

取消正在进行的人脸采集流程。调用此接口后，取消整个流程。请在Activity的onPause中调用此方法，保证界面离开顶端时流程被取消。下次进入界面应当重新开始人脸验证过程，可以参考demo的faceplatform-ui模块。

```
void LH.cancelFaceProcess()
```

10、采集过程中设置语音开关 设置是否开启人脸采集过程的语音提示。只有采集过程中可以通过此方法开启或关闭语音提

示。采集前调用此方法不会生效。可以参考demo的faceplatform-ui模块。

`void LH.setSoundEnable (boolean enable)`

11、OCR身份证初始化 如果需要使用OCR身份证能力，需要对OCR能力进行初始化。可以参考demo的app模块。

```
OCR.getInstance(this).initAccessToken(new OnResultListener<AccessToken>() {
    @Override
    public void onResult(final AccessToken result) {
    }

    @Override
    public void onError(final OCRError error) {
    }
}, getApplicationContext());
```

12、OCR身份证在线识别 调用recognizeIDCard进行身份证在线识别，可参考demo的ocr_ui模块。

```
OCR.getInstance(this).recognizeIDCard(param, new OnResultListener<IDCardResult>() {
    @Override
    public void onResult(final IDCardResult result) {}

    @Override
    public void onError(final OCRError error) {}
});
```

三、采集流程

1、正常采集流程 调用startFaceSurfaceLiveness或startFaceTextureLiveness -> onBegin被回调 -> 设备环境扫描 -> onDeviceCheckResult被回调-> onBeginCollectFaceInfo被回调 -> 开启摄像头（需要保证SurfaceView或TextureView为可见状态） -> onConfigCamera被回调，用户配置摄像头 -> 开始预览 -> onCollectCompletion被回调（多次，需要更新界面，最后一次回调status为OK或DetectRemindCodeTimeout）-> onBeginBuildData 被回调 -> onEnd被回调。

2、异常采集流程 onEnd可能在onBegin后的任何时间被回调，其他中间步骤可能会缺失。此种情况下，onEnd会回调错误码。

四、代码混淆 -keep class com.baidu.idl.* {*; }

-dontwarn com.baidu.vis.**

-keep class com.baidu.vis.* {*; }

-dontwarn com.baidu.liantian.**

-keep class com.baidu.sofire.* {*; }

-dontwarn com.baidu.baidusec.**

-keep class com.baidu.baidusec.* {*; }

-dontwarn com.baidu.ocr.**

-keep class com.baidu.ocr.* {*; }

五、注意事项 1、如果之前接入过百度人证核验SDK，如何升级？ 删除faceplatform模块和faceplatform-ui模块，然后重新导入demo工程中的faceplatform-ui模块、lib-liantian模块，ocr_ui模块（如果需要使用OCR身份证识别能力，则需要导入），可以参考新版本demo。

2、demo配置完签名和包名之后，运行报错

```
Manifest merger failed : Attribute provider#com.baidu.liantian.LiantianProvider@authorities value=
(com.baidu.idl.face.demo.liantian.ac.provider) from AndroidManifest.xml
```

将com.baidu.idl.face.demo.liantian.ac.provider修改为您的包名(build.gradle中的applicaitonId).liantian.ac.provider

3、人脸能力初始化的时候，报错初始化失败 = 13 -->local auth failed:6 请确认是否使用了AI开放平台控制台配置的包名、以及签名文件。

4、采集回调FaceProcessCallback接口onEnd方法返回-15，如何处理？ 可以在进入人脸界面之前，通过FH.isInitSuc(1)来判断安全模块是否下载成功，如果没有下载成功，建议用户切换网络后尝试。

5、不使用视频录制，如何操作？ 视频录制会将活体动作视频的留存在本地，如果使用此视频需要上传到服务端解密。如果不使用直接跳转到不到视频录制的界面即可。

6、如果不想使用OCR身份证识别能力，怎么操作？ 不使用OCR身份证识别能力，不需要导入ocr_ui模块，也不需要对OCR能力进行初始化，不需要向工程中放置OCR识别的授权文件。

7、OCR身份证识别初始化的时候，报错初始化失败 = [283502] App identifier unmatch 请确认是否使用了控制台创建应用时配置的文字识别包名、以及签名文件。

8、通过客户端demo如何进行人脸对比/人证核验/在线活体？ 因为客户端demo需要配合服务端demo使用，app客户端通过人脸SDK采集人脸数据后，上传app服务端，app服务端通过ak、sk请求百度服务器，返回结果。人脸对比/人证核验/在线活体接口请前往方案集成前准备下载服务端示例工程及文档。

标准级H5端实名认证方案

方案简介

方案简介

推出背景

- 如今，人脸识别技术被广泛应用在金融支付、用户注册、人脸登录等业务场景中。人脸实名认证产品团队与百度安全实验室联合推出**标准级H5实名认证方案**，在人脸登录、注册等环境加入层层保障，为您的业务保驾护航。

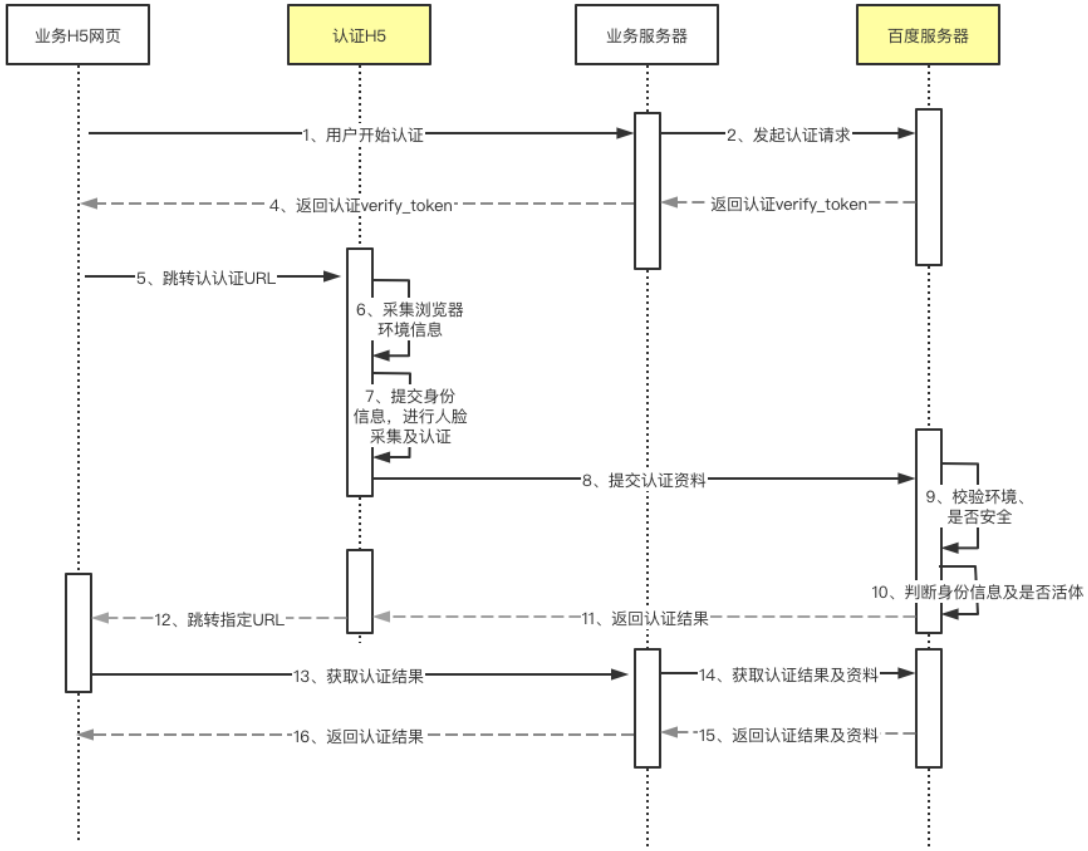
功能简介

- 标准级H5实名认证方案提供标准化的人身核验流程，具有**人脸比对**、**证件识别**、**活体检测**等多项组合能力，可抵挡常见的屏幕、照片、视频、换脸、面具、3D模型的攻击。同时还具有**合成图检测**、**图片质量检测**等优化功能，进一步提高实名认证流程的安全性和易用性。

适用场景

- 标准级H5实名认证方案适用于在微信公众号、H5等业务场景实现用户实名认证，如果您的场景是安卓/IOS系统的app场景，推荐采用 **标准级APP实名认证方案**。

接入时序图



方案接入流程

- 标准级H5实名认证方案具体接入步骤请参考方案接入指南

方案接入指南

本文档将帮助您完成标准级H5实名认证方案的创建及接入全流程。

一、 准备工作

在正式集成前，需要做一些准备工作，完成一些账号、应用及方案配置，具体如下：

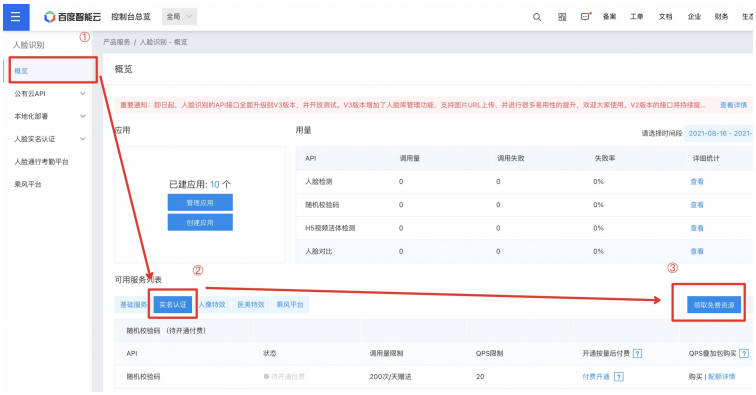
Step1: 注册成为开发者

> 在使用百度人脸实名认证方案之前，首先需注册百度智能云账号，账号注册方式请参考账号注册指南。
>
> 百度智能云账号注册完成以后，为顺利调用百度AI能力，需完成企业认证。具体认证方式请参考企业认证指南。

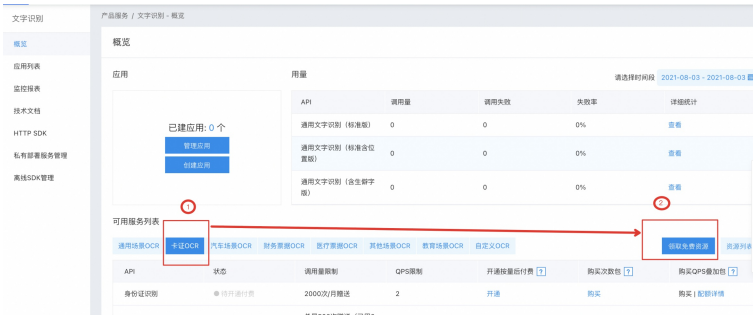
Step2：创建应用

2.1 输入应用名称，领取免费额度

- 创建好账号后，在正式调用AI 能力之前，需首先创建应用，应用是调用服务的基础能力单元。
- 同时领取接口所需的**免费调用额度**，用于接入测试。如下图所示：



- 除人脸服务接口的免费调用额度外，还需领取**身份证识别**接口的**免费调用额度**，用来调用身份证OCR识别功能（必须领取，否则会报错服务异常），点击此处，按下图所示进行领取。



- 如您之前已经领取过免费额度，无需重复领取，请跳至下一步骤。

2.2 勾选所需接口

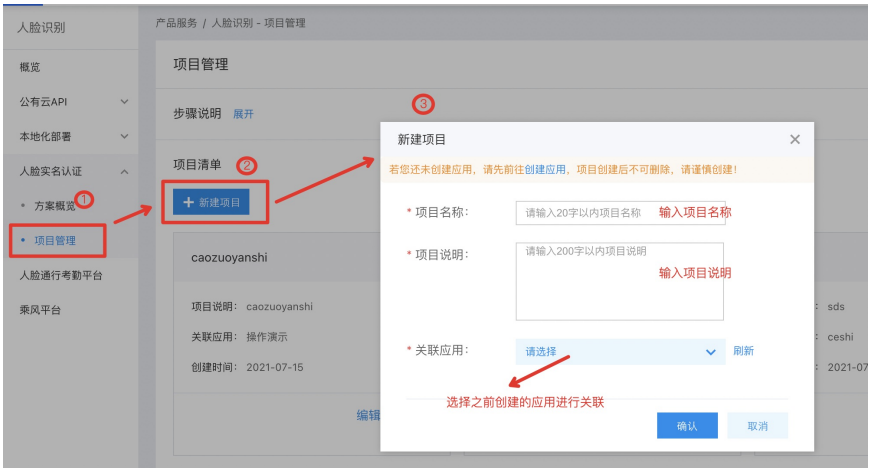
- 人脸识别服务相关接口已默认勾选且不可取消。

- **注：**「接口选择」过程中，还**需勾选「文字识别」中的「身份证识别」接口**，用于实现身份核验流程中的身份证识别功能。如下图所示。

Step3：创建项目

- 进入控制台-人脸实名认证页面，选择『**项目管理**』页面，点击『**新建项目**』，进行项目创建，如下图所示。

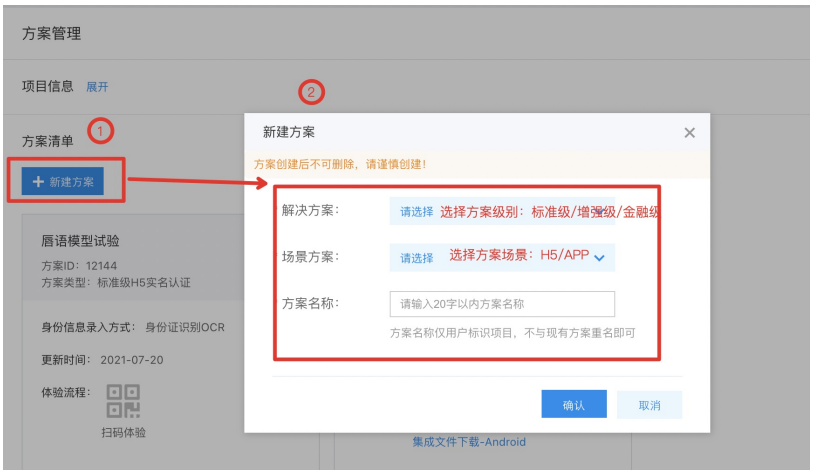
> 创建项目前，请确保您在应用控制台已创建应用，若您未创建应用，请参考Step2创建应用后，再进行项目创建.



Step4：创建方案

- 项目创建完成后，点击「**方案管理**」进入方案管理页面，在这里您可以为您的项目创建不同的方案，如下图所示。

> 若您的场景为APP场景（安卓/IOS系统），『场景方案』请选择APP实名认证方案；
>
> 若您的场景为微信、H5页面，『场景方案』请选择H5实名认证方案。



4.1 标题设置

例：当输入的内容为『标题名称创建演示』时，将于H5实名认证页面的头部进行展示，如下图

4.2 授权声明配置

在这里您可以配置授权声明的信息，您也可以选择跳过此配置。

4.3 身份证信息录入配置

- **非大陆数据源**：默认不使用。配置打开后支持对大陆居民二代身份证、港澳居民往来内陆通行证、外国人永久居住证及定居国外的中国公民护照的验证。若不打开则只支持大陆居民二代身份证的验证。
- **身份信息录入方式**：支持身份证识别OCR、手动输入、指定用户身份核验三种。

> - 当选择**身份证识别OCR**后,支持配置**身份证风控功能**，开启后，对身份证的真伪进行判断，支持识别翻拍、PS伪造的身份信息，但开启后会存在少量误通过和误识别现象。同时支持配置身份证**人像面+国徽面及人像面的选择**，选择后通过查询接口可以查询到接口的返回信息。
> - 当选择**手动输入**后，支持用户自己输入姓名及身份证号信息。
> - 当选择**指定用户身份核验**后，支持指定用户的姓名+身份证号信息，用户侧无需输入，只需进行人脸采集及活体检测即可。**注意：**若需要指定用户身份核验，则需要先请求 指定用户信息上报接口 再请求标准级H5实名认证方案的URL。

4.4 认证过程配置

此步骤对业务场景拍摄的人脸图片的质量信息及活体检测宽松程度进行配置。

- **图像质量检测**：分为严格、正常、宽松三个等级，越严格，图片对角度、模糊度、遮挡等信息参数把控越高，推荐使用宽松。
- **活体检测**：分为严格、正常、宽松三个等级，越严格，对活体的检测等信息把控越高，推荐使用正常。
- **合成图检测**：分为严格、正常、宽松三个等级，越严格，对合成图的检测等信息把控越高，推荐使用正常。
- **活体检测方案**：（推荐使用H5动作活体检测）

> - **数字活体检测**：通过语音验证码的形式（用户朗读屏幕上的随机语音验证码并录制视频）判断拍摄视频的用户是否为活体。
>   - 支持对语音验证码的长度进行限制，支持设置指定3~6位语音验证码
>   - 支持增加唇语的辅助校验，目前唇语识别能力当前为Beta版本，识别准确率较低，仅用于辅助验证。
> - **动作活体检测**：通过用户做指定动作来验证当前拍摄视频的用户是否为活体。
>   - 支持指定动作的个数进行验证，支持设置指定1-3个验证动作，推荐使用1或者2个动作进行验证。
> - **视频活体检测**：通过用户录制一段视频来验证当前拍摄视频的用户是否为活体。
> - **在线图片活体检测**：通过用户拍摄一张图片来验证当前拍摄视频的用户是否为活体。
>
> **附录：**
>
> **活体检测阈值指标** （高于此阈值即判断为活体）
>
> | 控制度 | 对应阈值 | 说明 |
> | --- | --- | --- |
> | 宽松 | 0.05 | 万分之一活体误拒率 |
> | 正常（推荐） | 0.3 | 千分之一活体误拒率 |
> | 严格 | 0.9 | 百分之一活体误拒率 |

4.5 认证结果配置

通过上传「姓名+身份证号+人脸图片」三要素到人脸实名认证接口，通过调取权威数据源判断是否为本人操作。

- **提供的认证结果页面**：您可以选择是否使用百度提供的展示给用户认证结果的页面，若您选择自己开发，可选择不使用。
- **认证未通过时URL是否失效**：当用户认证未通过时，部分用户会出现多次重新请求验证的情况，您可通过此项配置控制用户认证数。
- **阈值**：此阈值设置的是用户上传图片与公安权威数据源图片进行比对后得分的阈值，高于此阈值即判断为用户本人。**阈值设置推荐为80**，您可通过实际业务场景继续调整。

Step5：提交方案，扫码预览 方案配置完成后，点击提交按钮，进入方案管理页面，鼠标移入「扫码体验」即可显示二维码信息，扫码即可体验H5实名认证流程，如下图所示。

注意：请谨慎修改上述H5方案配置，在点击「提交」后会实时对方案配置进行更新。若您需要修改方案，请在不影响线上业务的情况下进行调整。

二、 方案接入

Step1：获取token 通过获取Token接口获取verify_token信息 **Step2：跳转实名认证H5 URL，用户进行操作** 业务H5网页通过获取Token接口返回的verify_token信息请求认证H5页面，进行用户端流程操作。

认证URL： https://brain.baidu.com/face/print/?token=xxx&successUrl=https://xxx&failedUrl=https://xxx

URL中的信息填写如下所示：

（1）token：填写verify_token，verify_token获取参考获取verify_token参考文档。

（2）successUrl：请求成功跳转的网址，网址需要加http/https前缀

（3）failedUrl：请求失败跳转的网址，网址需要加http/https前缀

(4)successUrl和failedUrl推荐使用encodeURIComponent进行转义，不然可能无法正确跳转。转义示例如下

```
encodeURIComponent("https://ai.baidu.com")
```

（5）若需要指定用户进行核验，在方案中身份信息录入-身份信息录入方式-指定用户身份核验，则需要先请求指定用户信息上

报接口再请求url。

操作流程参考如下：

Step3：获取认证结果及资料，返回用户认证信息 可以参考获取认证人脸接口文档、查询认证结果接口文档、查询统计结果接口文档查询用户人脸实名认证流程的相关信息，以进行后续业务集成开发。

接口文档

本文档为您提供了，您可以基于此接口进行业务信息的二次开发及查询操作。

一、方案功能接口 1.获取verify_token接口

本接口为H5实名认证方案的verify_token获取接口，利用所获取的verify_token进行实名认证流程的有效期为2小时。

调用方式

请求URL数据格式

向API服务地址使用POST发送请求，必须在URL中带上参数access_token，可通过后台的API Key和Secret Key生成，具体方式请参考“Access Token获取”。

> **注意**：access_token的有效期为30天，**切记需要每30天进行定期更换，或者每次请求都拉取新token**；

POST中Body的参数，按照下方请求参数说明选择即可。

> **提示**：如果您为百度云老用户，正在使用其他非AI的服务，可以参考百度云AKSK鉴权方式发送请求，虽然**请求方式**和**鉴权方法**和本文所介绍的不同，但请求参数和返回结果一致。

请求说明

注意事项：

- **请求体格式化**：Content-Type为application/json，通过json格式化请求体。

请求示例

HTTP方法：POST

请求URL：https://aip.baidubce.com/rpc/2.0/brain/solution/faceprint/verifyToken/generate

URL参数：

参数	值
access_token	通过API Key和Secret Key获取的access_token,参考“Access Token获取”

Header：

参数	值
Content-Type	application/json

Body中放置请求参数，参数详情如下：

请求参数：

参数	必选	类型	说明
plan_id	是	string	方案的id信息，请在人脸实名认证控制台查看创建的H5方案的方案ID信息

请求示例：

```
 {
 "plan_id"：1
}
```

返回参数

- 返回结果

字段	必选	类型	说明
success	是	boolean	返回请求是否成功信息。 若请求成功返回ture; 请求失败则返回fault
result	是	array	请求结果
+verify_token	是	string	请求获取的verify_token

- 返回示例

```
{
  "success": true,
  "result": {
     "verify_token": "Yz9rWITm4vak16PBAh5x8oG7"
  },
  "log_id": "1814798895"
}
```

2.指定用户信息上报接口 本接口用于，前端在方案中选择身份信息录入-身份信息录入方式-指定用户身份核验时，需要先调用此接口输入指定用户的姓名+身份证号信息，再请求url跳转页面。

调用方式

请求URL数据格式

向API服务地址使用POST发送请求，必须在URL中带上参数`access_token`，可通过后台的API Key和Secret Key生成，具体方式请参考“Access Token获取”。

> **注意**：`access_token`的有效期为30天，**切记需要每30天进行定期更换，或者每次请求都拉取新token**；

POST中Body的参数，按照下方请求参数说明选择即可。

> **提示**：如果您为百度云老用户，正在使用其他非AI的服务，可以参考百度云AKSK鉴权方式发送请求，虽然**请求方式**和**鉴权**

> **方法**和本文所介绍的不同，但请求参数和返回结果一致。

请求说明

注意事项：

- **请求体格式化**：Content-Type为`application/json`，通过`json`格式化请求体。

请求示例

HTTP方法：`POST`

请求URL：https://brain.baidu.com/solution/faceprint/idcard/submit

> 注意这里不需要传access_token

Header：

参数	值
Content-Type	application/json

Body中放置请求参数，参数详情如下：

参数	必选	类型	值
verify_token	是	string	通过access_token获取的verify_token
id_name	是	string	指定输入用户的姓名信息
id_no	是	string	指定输入用户的身份证件号信息
certificate_type	否	Int	证件类型： 0大陆居民二代身份证 1港澳台居民来往内地通行证 2外国人永久居留证 3定居国外的中国公民护照

请求示例：

```
{
  "verify_token": "2sF3nE5mXOHkx2aQwWG4n5WI",
  "id_name": "张三",
  "id_no": "500***********3390",
  "certificate_type": 0          // 证件类型：0大陆居民二代身份证，1港澳台居民来往内地通行证，2外国人永久居留证，3定居国外的中国公民护照
}
```

返回参数

- 返回结果

字段	必选	类型	说明
success	是	boolean	返回请求是否成功信息。 若请求成功返回ture; 请求失败则返回false
result	是	array	请求结果，返回固定结果1，可忽略

- 返回示例

```
{
  "success": true,
  "result": 1,
  "log_id": "1244068892"
}
```

二、验证后查询接口

> 获取Token后，请先按照跳转实名认证H5 URL，用户进行操作后再查询接口，否则生成Token无法生效。

1.获取认证人脸接口

本接口返回进行人脸实名认证过程中进行认证的最终采集的人脸信息（仅在认证成功时返回人脸信息，认证失败返回错误码）。根据Verify_token返回的结果信息会在云端保留两个小时，您可根据需要在此期间进行调取查询。

调用方式

请求URL数据格式

向API服务地址使用POST发送请求，必须在URL中带上参数access_token，可通过后台的API Key和Secret Key生成，具体方式请参考“Access Token获取”。

> **注意**：access_token的有效期为30天，**切记需要每30天进行定期更换，或者每次请求都拉取新token**；

POST中Body的参数，按照下方请求参数说明选择即可。

> **提示**：如果您为百度云老用户，正在使用其他非AI的服务，可以参考百度云AKSK鉴权方式发送请求，虽然**请求方式**和**鉴权方法**和本文所介绍的不同，但请求参数和返回结果一致。

请求说明

注意事项：

- **请求体格式化**：Content-Type为application/json，通过json格式化请求体。

请求示例

HTTP方法：POST

请求URL： https://aip.baidubce.com/rpc/2.0/brain/solution/faceprint/result/simple

URL参数：

参数	值
access_token	通过API Key和Secret Key获取的access_token,参考“Access Token获取”

Header：

参数	值
Content-Type	application/json

Body中放置请求参数，参数详情如下：

参数	值
verify_token	通过access_token获取的verify_token

请求示例：

```
 {
 "verify_token"："clupeyP51sn28XzxGVTfYqoN"
}
```

返回参数

- 返回结果

字段	必选	类型	说明
success	是	boolean	返回请求是否成功信息。 若请求成功返回ture; 请求失败则返回fault
result	是	array	请求结果
+image	是	string	返回采集的用户人脸信息 （仅在认证成功时返回人脸信息，认证失败返回错误码）

- 返回示例

```
{
  "success": true,
  "result": {
     "image":"https://brain.baidu.com/solution/faceprint/image/query?verify_token=xxxxxx"
  },
  "log_id": "1054986003"
}
```

2.查询认证结果接口

本接口为请求返回的认证结果信息查询，包含身份证OCR识别信息、用户二次确认的身份证信息，活体检测信息、及用户对权威数据源图片进行比对的分数信息。（仅在认证成功时返回上述信息，认证失败返回错误码）根据Verify_token返回的结果信息会在云端保留三天，您可根据需要在此期间进行调取查询。

调用方式

请求URL数据格式

向API服务地址使用POST发送请求，必须在URL中带上参数`access_token`，可通过后台的API Key和Secret Key生成，具体方式请参考“Access Token获取”。

> **注意**：`access_token`的有效期为30天，**切记需要每30天进行定期更换，或者每次请求都拉取新token**；

POST中Body的参数，按照下方请求参数说明选择即可。

> **提示**：如果您为百度云老用户，正在使用其他非AI的服务，可以参考百度云AKSK鉴权方式发送请求，虽然**请求方式**和**鉴权方法**和本文所介绍的不同，但请求参数和返回结果一致。

请求说明

注意事项：

- **请求体格式化**：Content-Type为`application/json`，通过`json`格式化请求体。

请求示例

HTTP方法：`POST`

请求URL： `https://aip.baidubce.com/rpc/2.0/brain/solution/faceprint/result/detail`

URL参数：

参数	值
access_token	通过API Key和Secret Key获取的access_token,参考“Access Token获取”

Header：

参数	值
Content-Type	application/json

Body中放置请求参数，参数详情如下：

参数	值
verify_token	通过access_token获取的verify_token

请求示例：

```
 {
 "verify_token"："clupeyP51sn28XzxGVTfYqoN"
}
```

返回参数

- 返回结果

字段	必选	类型	说明
success	是	boolean	返回请求是否成功信息。 若请求成功返回ture; 请求失败则返回fault
result	是	array	请求结果
+idcard_ocr_result	否	array	返回采集的身份证信息 当人脸实名认证控制台设置为使用OCR识别时返回此参数信息
++address	否	string	地址
++birthday	否	string	生日
++name	否	string	姓名
++id_card_number	否	string	身份证号
++gender	否	string	性别
++nation	否	string	民族
++expire_time	否	string	身份证失效日期
++issue_authority	否	string	身份证签发机关
++issue_time	否	string	身份证生效日期
+idcard_images	否	array	返回采集的身份证图片信息 当人脸实名认证控制台设置为使用OCR识别时返回此参数信息
++front_base64	否	string	身份证图片的正面信息
++back_base64	否	string	身份证图片的反面信息 当人脸实名认证控制台设置为使用OCR识别且为国徽面+人像面时返回此参数信息
+verify_result	是	array	认证返还信息
++liveness_score	是	string	**活体检测分数：** 在线图片活体：活体验证通过时返回活体分数，不通过则返回0。 数字/动作/视频活体：活体通过/不通过均会返回活体分数
++score	是	string	人脸实名认证
++spoofing	是	string	合成图分数 若未进行合成图检测，则返回0 若进行活体检测，则返回合成图检测分值
+idcard_confirm	是	array	用户二次确认的身份证信息
++name	是	string	姓名
++idcard_number	是	string	身份证号

- 返回示例

```
{
 "success": true,
 "result": {
    "verify_result": {
        "score": 93.7835,
        "liveness_score": 0.9672966,
        "spoofing": 0.0
    },
    "idcard_ocr_result": {
        "birthday": "19960216",
        "issue_authority": "胶南市公安局",
        "address": "山东省**********",
        "gender": "女",
        "nation": "汉",
        "expire_time": "20221103",
        "name": "柴*",
        "issue_time": "20121103",
        "id_card_number": "370**********5826"
    },
    "idcard_images": {
        "front_base64": "/9j/4AAQSkZJRgAB....",
        "back_base64": "/9j/4AAQSkZJRgAB...."
 },
    "idcard_confirm": {
        "idcard_number": "370**********5826",
        "name": "柴*"
    }
 },
 "log_id": "160931948204246"
}
```


3.查询统计结果

根据Verify_token返回的结果信息会在云端保留三天，您可根据需要在此期间进行调取查询。

调用方式

请求URL数据格式

向API服务地址使用POST发送请求，必须在URL中带上参数`access_token`，可通过后台的API Key和Secret Key生成，具体方式请参考“Access Token获取”。

> **注意**：`access_token`的有效期为30天，**切记需要每30天进行定期更换，或者每次请求都拉取新token**；

POST中Body的参数，按照下方请求参数说明选择即可。

> **提示**：如果您为百度云老用户，正在使用其他非AI的服务，可以参考百度云AKSK鉴权方式发送请求，虽然**请求方式**和**鉴权方法**和本文所介绍的不同，但请求参数和返回结果一致。

请求说明

注意事项：

- **请求体格式化**：Content-Type为`application/json`，通过`json`格式化请求体。

请求示例

HTTP方法：POST

请求URL： https://aip.baidubce.com/rpc/2.0/brain/solution/faceprint/result/stat

URL参数：

参数	值
access_token	通过API Key和Secret Key获取的access_token,参考“Access Token获取”

Header：

参数	值
Content-Type	application/json

Body中放置请求参数，参数详情如下：

参数	值
verify_token	通过access_token获取的verify_token

请求示例：

```
 {
 "verify_token"："clupeyP51sn28XzxGVTfYqoN"
}
```

返回参数

- 返回结果

字段	必选	类型	说明
success	是	boolean	返回请求是否成功信息。 若请求成功返回ture; 请求失败则返回fault
result	是	array	请求结果
+身份证识别	是	array	身份证识别接口请求的统计信息
++error_code	是	string	错误码编号 若为0则表示请求成功
++count	是	int	当前错误码的请求数量
+h5活体视频分析	是	array	h5活体视频分析接口请求的统计信息
++error_code	是	string	错误码编号 若为0则表示请求成功
++count	是	int	当前错误码的请求数量
+人脸实名认证-V3	是	array	人脸实名认证-V3接口请求的统计信息
++error_code	是	string	错误码编号 若为0则表示请求成功
++count	是	string	当前错误码的请求数量

- 返回示例

```
{
  "success": true,
  "result": {
    "身份证识别": [
      {
        "error_code": 0,
        "count": 2
      }
    ],
    "h5活体视频分析": [
      {
        "error_code": 0,
        "count": 1
      }
    ],
    "人脸实名认证-V3": [
      {
        "error_code": 0,
        "count": 1
      }
    ]
  },
  "log_id": "1405335905"
}
```


增强级H5端实名认证方案

方案简介

方案简介

推出背景

- 现在，人脸识别技术被广泛应用在金融支付、用户注册、人脸登录等业务场景中。技术的进步方便用户的同时，黑灰产产业也开始对这些场景产生觊觎。并通过**屏幕攻击、照片、纸张、以及面具、头模**等方式进行非法攻击。随着黑产技术的进步，更是出现了通过**批量虚拟机、病毒侵入**等新型攻击手段。使现有的人脸识别方案面临着巨大的安全挑战。
- 为提升人脸识别的安全性，保障客户的业务安全，人脸实名认证产品团队与百度安全实验室联合推出**增强级人脸实名认证方案**，在人脸登录、注册等环境加入层层保障，为您的业务保驾护航。

功能简介

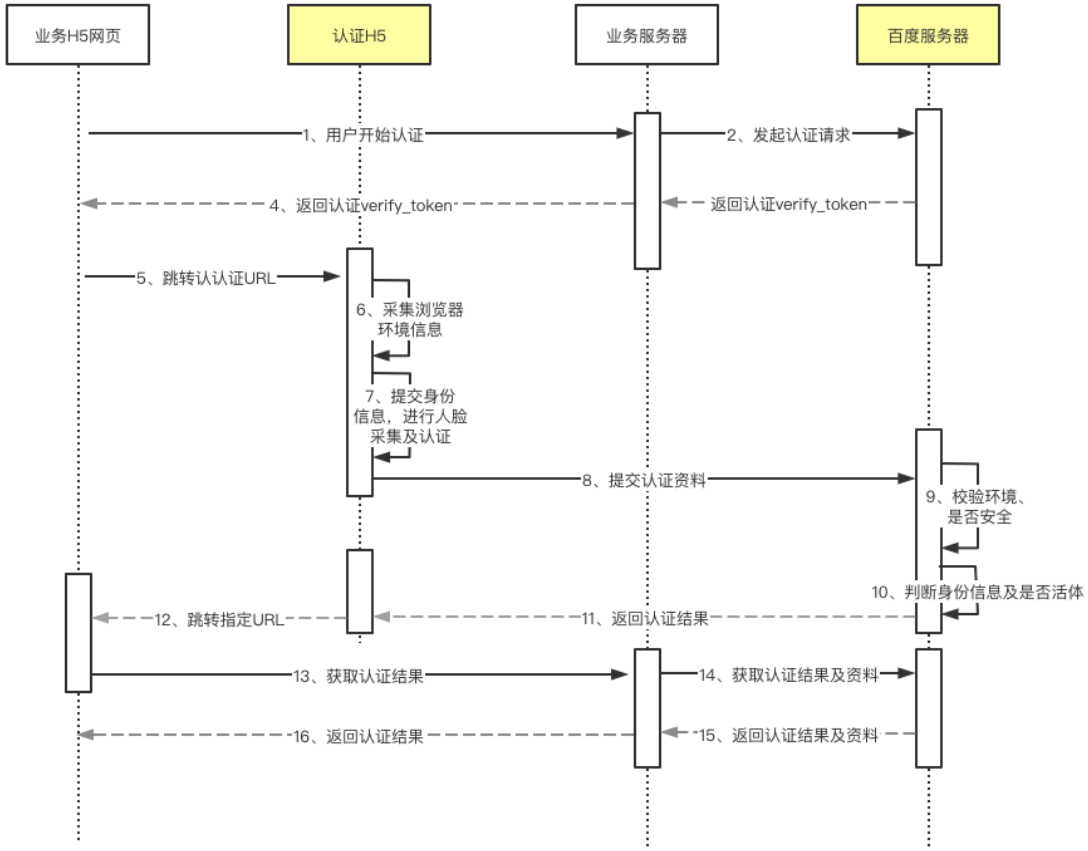
- 增强级H5实名认证方案提供标准化的人身核验流程，具有**人脸比对、证件识别、活体检测**等多项组合能力，可抵挡屏幕、照片、视频、换脸、面具、3D模型的攻击。同时，增强级H5实名认证方案中加入**大数据风控能力**，针对批量虚拟机、病毒侵入等新型攻击手段进行强力有效防御。

> **大数据风控能力：**增强级API接口接受前端传入的设备指纹信息，基于百度海量大数据设备因子，进行设备风险识别，辨别是否为风险设备，返回识别结果。可有效防御黑产批量虚拟机、病毒侵入等攻击手段，降低第三方黑产攻破概率，提升业务安全性。

适用场景

- 增强级H5实名认证方案适用于在微信公众号、H5等业务场景实现用户实名认证，如果您的场景是安卓/IOS系统的app场景，推荐采用 增强级APP实名认证方案。

接入时序图



方案接入流程

- 增强级H5实名认证方案具体接入步骤请参考方案接入指南

方案接入指南

本文档将帮助您完成H5实名认证方案的创建及接入全流程。

一、 准备工作

在正式集成前，需要做一些准备工作，完成一些账号、应用及方案配置，具体如下：

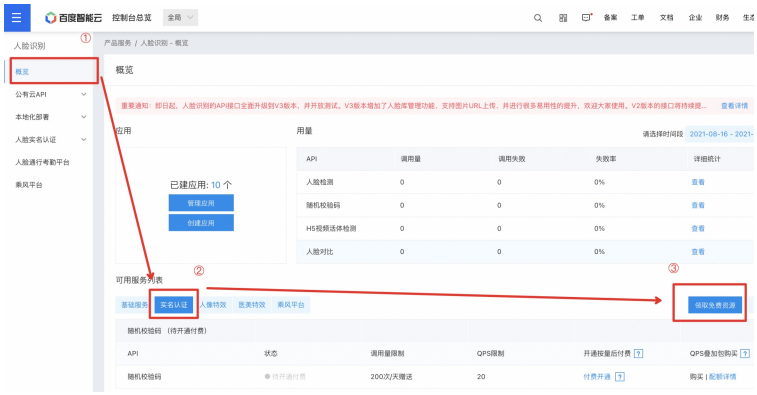
Step1: 注册成为开发者

> 在使用百度人脸实名认证方案之前，首先需注册百度智能云账号，账号注册方式请参考账号注册指南。
>
> 百度智能云账号注册完成以后，为顺利调用百度AI能力，需完成企业认证。具体认证方式请参考企业认证指南。

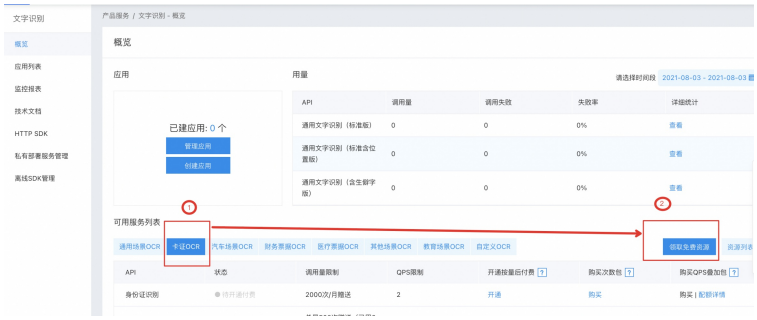
Step2：创建应用

2.1 输入应用名称，领取免费额度

- 创建好账号后，在正式调用AI 能力之前，需首先创建应用，应用是调用服务的基础能力单元。
- 同时领取接口所需的**免费调用额度**，用于接入测试。如下图所示：



- 除人脸服务接口的免费调用额度外，还需领取**身份证识别**接口的**免费调用额度**，用来调用身份证OCR识别功能（必须领取，否则会报错服务异常），点击此处，按下图所示进行领取。

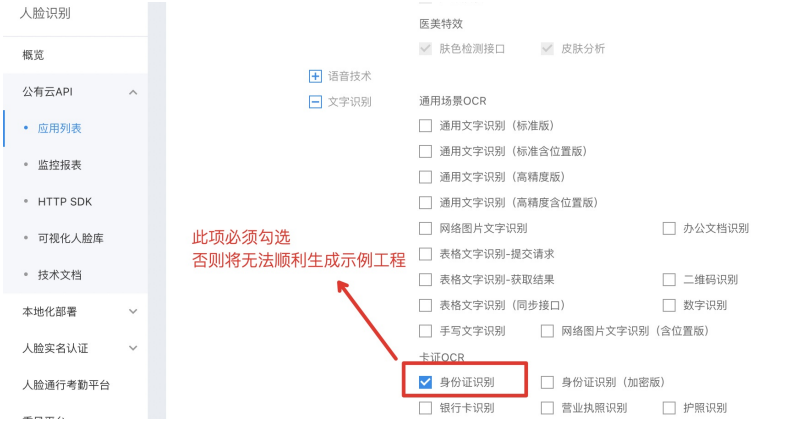


- 如您之前已经领取过免费额度，无需重复领取，请跳至下一步骤。

2.2 勾选所需接口

- 人脸识别服务相关接口已默认勾选且不可取消。

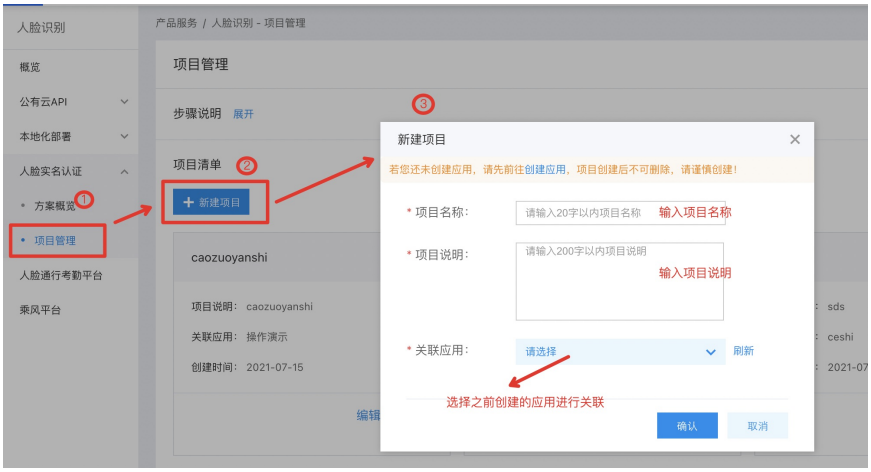
- **注：**「接口选择」过程中，还**需勾选「文字识别」中的「身份证识别」接口**，用于实现身份核验流程中的身份证识别功能。如下图所示。



Step3：创建项目

- 进入控制台-人脸实名认证页面，选择『**项目管理**』页面，点击『**新建项目**』，进行项目创建，如下图所示。

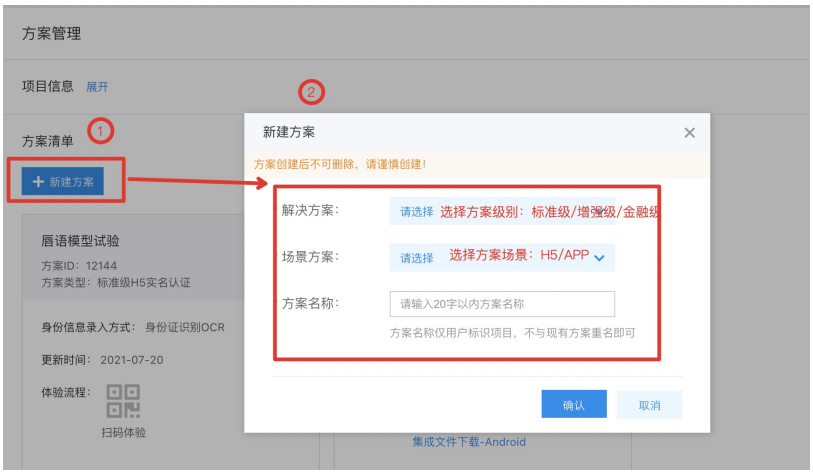
> 创建项目前，请确保您在应用控制台已创建应用，若您未创建应用，请参考Step2创建应用后，再进行项目创建.



Step4：创建方案

- 项目创建完成后，点击「**方案管理**」进入方案管理页面，在这里您可以为您的项目创建不同的方案，如下图所示。

> 若您的场景为APP场景（安卓/IOS系统），『场景方案』请选择APP实名认证方案；
>
> 若您的场景为微信、H5页面，『场景方案』请选择H5实名认证方案。



4.1 标题设置

例：当输入的内容为『标题名称创建演示』时，将于H5实名认证页面的头部进行展示，如下图

4.2 授权声明配置

在这里您可以配置授权声明的信息，您也可以选择跳过此配置。

4.3 身份证信息录入配置

- **非大陆数据源**：默认不使用。配置打开后支持对大陆居民二代身份证、港澳居民往来内陆通行证、外国人永久居住证及定居国外的中国公民护照的验证。若不打开则只支持大陆居民二代身份证的验证。
- **身份信息录入方式**：支持身份证识别OCR、手动输入、指定用户身份核验三种。

> - 当选择**身份证识别OCR**后,支持配置**身份证风控功能**，开启后，对身份证的真伪进行判断，支持识别翻拍、PS伪造的身份信息，但开启后会存在少量误通过和误识别现象。同时支持配置身份证**人像面+国徽面及人像面的选择**，选择后通过查询接口可以查询到接口的返回信息。
> - 当选择**手动输入**后，支持用户自己输入姓名及身份证号信息。
> - 当选择**指定用户身份核验**后，支持指定用户的姓名+身份证号信息，用户侧无需输入，只需进行人脸采集及活体检测即可。**注意：**若需要指定用户身份核验，则需要先请求 指定用户信息上报接口 再请求增强级H5实名认证方案的URL。

4.4 认证过程配置

此步骤对业务场景拍摄的人脸图片的质量信息及活体检测宽松程度进行配置。

- **图像质量检测**：分为严格、正常、宽松三个等级，越严格，图片对角度、模糊度、遮挡等信息参数把控越高，推荐使用宽松。
- **活体检测**：分为严格、正常、宽松三个等级，越严格，对活体的检测等信息把控越高，推荐使用正常。
- **合成图检测**：分为严格、正常、宽松三个等级，越严格，对合成图的检测等信息把控越高，推荐使用正常。
- **活体检测方案**：（推荐使用H5动作活体检测）

> - **数字活体检测**：通过语音验证码的形式（用户朗读屏幕上的随机语音验证码并录制视频）判断拍摄视频的用户是否为活体。
>   - 支持对语音验证码的长度进行限制，支持设置指定3~6位语音验证码
>   - 支持增加唇语的辅助校验，目前唇语识别能力当前为Beta版本，识别准确率较低，仅用于辅助验证。
> - **动作活体检测**：通过用户做指定动作来验证当前拍摄视频的用户是否为活体。
>   - 支持指定动作的个数进行验证，支持设置指定1-3个验证动作，推荐使用1或者2个动作进行验证。
> - **视频活体检测**：通过用户录制一段视频来验证当前拍摄视频的用户是否为活体。
> - **在线图片活体检测**：通过用户拍摄一张图片来验证当前拍摄视频的用户是否为活体。
>
> **附录：**
>
> **活体检测阈值指标** （高于此阈值即判断为活体）
>
> | 控制度 | 对应阈值 | 说明 |
> | --- | --- | --- |
> | 宽松 | 0.05 | 万分之一活体误拒率 |
> | 正常（推荐） | 0.3 | 千分之一活体误拒率 |
> | 严格 | 0.9 | 百分之一活体误拒率 |

4.5 认证结果配置

通过上传「姓名+身份证号+人脸图片」三要素到人脸实名认证接口，通过调取权威数据源判断是否为本人操作。

- **提供的认证结果页面**：您可以选择是否使用百度提供的展示给用户认证结果的页面，若您选择自己开发，可选择不使用。
- **认证未通过时URL是否失效**：当用户认证未通过时，部分用户会出现多次重新请求验证的情况，您可通过此项配置控制用户认证数。
- **阈值**：此阈值设置的是用户上传图片与公安权威数据源图片进行比对后得分的阈值，高于此阈值即判断为用户本人。**阈值设置推荐为80**，您可通过实际业务场景继续调整。

Step5：提交方案，扫码预览 方案配置完成后，点击提交按钮，进入方案管理页面，鼠标移入「扫码体验」即可显示二维码信息，扫码即可体验H5实名认证流程，如下图所示。

注意：请谨慎修改上述H5方案配置，在点击「提交」后会实时对方案配置进行更新。若您需要修改方案，请在不影响线上业务的情况下进行调整。

二、 方案接入

Step1：获取token 通过获取Token接口获取verify_token信息 **Step2：跳转实名认证H5 URL，用户进行操作** 业务H5网页通过获取Token接口返回的verify_token信息请求认证H5页面，进行用户端流程操作。

认证URL： https://brain.baidu.com/face/print/?token=xxx&successUrl=https://xxx&failedUrl=https://xxx

URL中的信息填写如下所示：

（1）**token：**填写verify_token，verify_token获取参考获取verify_token参考文档。

（2）**successUrl：**请求成功跳转的网址，网址需要加http/https前缀

（3）**failedUrl：**请求失败跳转的网址，网址需要加http/https前缀

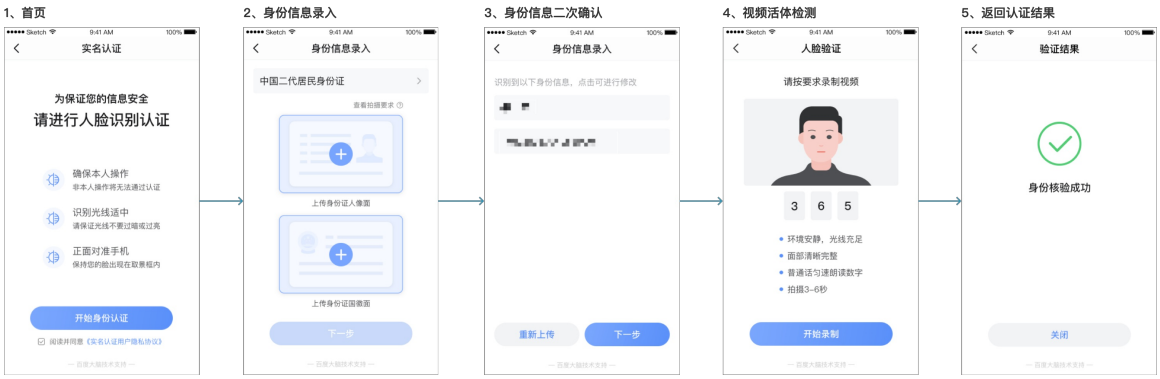
（4）**successUrl和failedUrl推荐使用encodeURIComponent进行转义**，不然可能无法正确跳转，转义示例如下：

```
encodeURIComponent("https://ai.baidu.com")
```

（5）若需要指定用户进行核验，在方案中身份信息录入-身份信息录入方式-指定用户身份核验，则需要先请求指定用户信息上

报接口再请求url。

操作流程参考如下：



Step3：获取认证结果及资料，返回用户认证信息 可以参考获取认证人脸接口文档、查询认证结果接口文档、查询统计结果接口文档查询用户人脸实名认证流程的相关信息，以进行后续业务集成开发。

接口文档

本文档为您提供了，您可以基于此接口进行业务信息的二次开发及查询操作。

一、方案功能接口 1.获取verify_token接口

本接口为H5实名认证方案的verify_token获取接口，利用所获取的verify_token进行实名认证流程的有效期为2小时。

调用方式

请求URL数据格式

向API服务地址使用POST发送请求，必须在URL中带上参数access_token，可通过后台的API Key和Secret Key生成，具体方式请参考“Access Token获取”。

> **注意**：access_token的有效期为30天，**切记需要每30天进行定期更换，或者每次请求都拉取新token**；

POST中Body的参数，按照下方请求参数说明选择即可。

> **提示**：如果您为百度云老用户，正在使用其他非AI的服务，可以参考百度云AKSK鉴权方式发送请求，虽然**请求方式**和**鉴权方法**和本文所介绍的不同，但请求参数和返回结果一致。

请求说明

注意事项：

- **请求体格式化**：Content-Type为application/json，通过json格式化请求体。

请求示例

HTTP方法：POST

请求URL： https://aip.baidubce.com/rpc/2.0/brain/solution/faceprint/verifyToken/generate

URL参数：

参数	值
access_token	通过API Key和Secret Key获取的access_token,参考“Access Token获取”

Header：

参数	值
Content-Type	application/json

Body中放置请求参数，参数详情如下：

请求参数：

参数	必选	类型	说明
plan_id	是	string	方案的id信息，请在人脸实名认证控制台查看创建的H5方案的方案ID信息

请求示例：

```
 {
 "plan_id"：1
}
```

返回参数

- 返回结果

字段	必选	类型	说明
success	是	boolean	返回请求是否成功信息。 若请求成功返回ture; 请求失败则返回fault
result	是	array	请求结果
+verify_token	是	string	请求获取的verify_token

- 返回示例

```
{
  "success": true,
  "result": {
     "verify_token": "Yz9rWITm4vak16PBAh5x8oG7"
  },
  "log_id": "1814798895"
}
```

2.指定用户信息上报接口 本接口用于，前端在方案中选择身份信息录入-身份信息录入方式-指定用户身份核验时，需要先调用此接口输入指定用户的姓名+身份证号信息，再请求url跳转页面。

调用方式

请求URL数据格式

向API服务地址使用POST发送请求，必须在URL中带上参数`access_token`，可通过后台的API Key和Secret Key生成，具体方式请参考“Access Token获取”。

> **注意**：`access_token`的有效期为30天，**切记需要每30天进行定期更换，或者每次请求都拉取新token**；

POST中Body的参数，按照下方请求参数说明选择即可。

> **提示**：如果您为百度云老用户，正在使用其他非AI的服务，可以参考百度云AKSK鉴权方式发送请求，虽然**请求方式**和**鉴权**

> **方法**和本文所介绍的不同，但请求参数和返回结果一致。

请求说明

注意事项：

- **请求体格式化**：Content-Type为`application/json`，通过`json`格式化请求体。

请求示例

HTTP方法：`POST`

请求URL：https://brain.baidu.com/solution/faceprint/idcard/submit

> 注意这里不需要传access_token

Header：

参数	值
Content-Type	application/json

Body中放置请求参数，参数详情如下：

参数	必选	类型	值
verify_token	是	string	通过access_token获取的verify_token
id_name	是	string	指定输入用户的姓名信息
id_no	是	string	指定输入用户的身份证件号信息
certificate_type	否	Int	证件类型： 0大陆居民二代身份证 1港澳台居民来往内地通行证 2外国人永久居留证 3定居国外的中国公民护照

请求示例：

```
 {
   "verify_token": "2sF3nE5mXOHkx2aQwWG4n5WI",
   "id_name": "张三",
   "id_no": "500***********3390",
   "certificate_type": 0          // 证件类型：0大陆居民二代身份证，1港澳台居民来往内地通行证，2外国人永久居留证，3定居国外的中国公民护照
}
```

返回参数

- 返回结果

字段	必选	类型	说明
success	是	boolean	返回请求是否成功信息。 若请求成功返回ture; 请求失败则返回false
result	是	array	请求结果，返回固定结果1，可忽略

- 返回示例

```
{
  "success": true,
  "result": 1,
  "log_id": "1244068892"
}
```

二、验证后查询接口

> 获取Token后，请先按照跳转实名认证H5 URL，用户进行操作后再查询接口，否则生成Token无法生效。

1.获取认证人脸接口

本接口返回进行人脸实名认证过程中进行认证的最终采集的人脸信息（仅在认证成功时返回人脸信息，认证失败返回错误码）。根据Verify_token返回的结果信息会在云端保留两个小时，您可根据需要在此期间进行调取查询。

调用方式

请求URL数据格式

向API服务地址使用POST发送请求，必须在URL中带上参数access_token，可通过后台的API Key和Secret Key生成，具体方式请参考“Access Token获取”。

> **注意**：access_token的有效期为30天，**切记需要每30天进行定期更换，或者每次请求都拉取新token**；

POST中Body的参数，按照下方请求参数说明选择即可。

> **提示**：如果您为百度云老用户，正在使用其他非AI的服务，可以参考百度云AKSK鉴权方式发送请求，虽然**请求方式**和**鉴权方法**和本文所介绍的不同，但请求参数和返回结果一致。

请求说明

注意事项：

- **请求体格式化**：Content-Type为application/json，通过json格式化请求体。

请求示例

HTTP方法：POST

请求URL： https://aip.baidubce.com/rpc/2.0/brain/solution/faceprint/result/simple

URL参数：

参数	值
access_token	通过API Key和Secret Key获取的access_token,参考“Access Token获取”

Header：

参数	值
Content-Type	application/json

Body中放置请求参数，参数详情如下：

参数	值
verify_token	通过access_token获取的verify_token

请求示例：

```
 {
 "verify_token"："clupeyP51sn28XzxGVTfYqoN"
}
```

返回参数

- 返回结果

字段	必选	类型	说明
success	是	boolean	返回请求是否成功信息。 若请求成功返回ture; 请求失败则返回fault
result	是	array	请求结果
+image	是	string	返回采集的用户人脸信息 （仅在认证成功时返回人脸信息，认证失败返回错误码）

- 返回示例

```
{
  "success": true,
  "result": {
     "image":"https://brain.baidu.com/solution/faceprint/image/query?verify_token=xxxxxx"
  },
  "log_id": "1054986003"
}
```

2.查询认证结果接口

本接口为请求返回的认证结果信息查询，包含身份证OCR识别信息、用户二次确认的身份证信息，活体检测信息、及用户对权威数据源图片进行比对的分数信息。（仅在认证成功时返回上述信息，认证失败返回错误码）根据Verify_token返回的结果信息会在云端保留三天，您可根据需要在此期间进行调取查询。

调用方式

请求URL数据格式

向API服务地址使用POST发送请求，必须在URL中带上参数`access_token`，可通过后台的API Key和Secret Key生成，具体方式请参考“Access Token获取”。

> **注意**：`access_token`的有效期为30天，**切记需要每30天进行定期更换，或者每次请求都拉取新token**；

POST中Body的参数，按照下方请求参数说明选择即可。

> **提示**：如果您为百度云老用户，正在使用其他非AI的服务，可以参考百度云AKSK鉴权方式发送请求，虽然**请求方式**和**鉴权方法**和本文所介绍的不同，但请求参数和返回结果一致。

请求说明

注意事项：

- **请求体格式化**：Content-Type为`application/json`，通过`json`格式化请求体。

请求示例

HTTP方法：`POST`

请求URL： `https://aip.baidubce.com/rpc/2.0/brain/solution/faceprint/result/detail`

URL参数：

参数	值
access_token	通过API Key和Secret Key获取的access_token,参考“Access Token获取”

Header：

参数	值
Content-Type	application/json

Body中放置请求参数，参数详情如下：

参数	值
verify_token	通过access_token获取的verify_token

请求示例：

```
 {
 "verify_token"："clupeyP51sn28XzxGVTfYqoN"
}
```

返回参数

- 返回结果

字段	必选	类型	说明
success	是	boolean	返回请求是否成功信息。 若请求成功返回ture; 请求失败则返回fault
result	是	array	请求结果
+idcard_ocr_result	否	array	返回采集的身份证信息 当人脸实名认证控制台设置为使用OCR识别时返回此参数信息
++address	否	string	地址
++birthday	否	string	生日
++name	否	string	姓名
++id_card_number	否	string	身份证号
++gender	否	string	性别
++nation	否	string	民族
++expire_time	否	string	身份证失效日期
++issue_authority	否	string	身份证签发机关
++issue_time	否	string	身份证生效日期
+idcard_images	否	array	返回采集的身份证图片信息 当人脸实名认证控制台设置为使用OCR识别时返回此参数信息
++front_base64	否	string	身份证图片的正面信息
++back_base64	否	string	身份证图片的反面信息 当人脸实名认证控制台设置为使用OCR识别且为国徽面+人像面时返回此参数信息
+verify_result	是	array	认证返还信息
++liveness_score	是	string	**活体检测分数：** 活体检测通过返回活体分数，不通过则返回为0
++score	是	string	人脸实名认证
++spoofing	是	string	合成图分数 若未进行合成图检测，则返回0 若进行活体检测，则返回合成图检测分值
+idcard_confirm	是	array	用户二次确认的身份证信息
++name	是	string	姓名
++idcard_number	是	string	身份证号

- 返回示例

```
{
 "success": true,
 "result": {
    "verify_result": {
        "score": 93.7835,
        "liveness_score": 0.9672966,
        "spoofing": 0.0
    },
    "idcard_ocr_result": {
        "birthday": "19960216",
        "issue_authority": "胶南市公安局",
        "address": "山东省**********",
        "gender": "女",
        "nation": "汉",
        "expire_time": "20221103",
        "name": "柴*",
        "issue_time": "20121103",
        "id_card_number": "370**********5826"
    },
    "idcard_images": {
        "front_base64": "/9j/4AAQSkZJRgAB....",
        "back_base64": "/9j/4AAQSkZJRgAB...."
 },
    "idcard_confirm": {
        "idcard_number": "370**********5826",
        "name": "柴*"
    }
 },
 "log_id": "160931948204246"
}
```


3.查询统计结果

根据Verify_token返回的结果信息会在云端保留三天，您可根据需要在此期间进行调取查询。

调用方式

请求URL数据格式

向API服务地址使用POST发送请求，必须在URL中带上参数`access_token`，可通过后台的API Key和Secret Key生成，具体方式请参考“Access Token获取”。

> **注意**：`access_token`的有效期为30天，**切记需要每30天进行定期更换，或者每次请求都拉取新token**；

POST中Body的参数，按照下方请求参数说明选择即可。

> **提示**：如果您为百度云老用户，正在使用其他非AI的服务，可以参考百度云AKSK鉴权方式发送请求，虽然**请求方式**和**鉴权方法**和本文所介绍的不同，但请求参数和返回结果一致。

请求说明

注意事项：

- **请求体格式化**：Content-Type为`application/json`，通过`json`格式化请求体。

请求示例

HTTP方法：POST

请求URL： https://aip.baidubce.com/rpc/2.0/brain/solution/faceprint/result/stat

URL参数：

参数	值
access_token	通过API Key和Secret Key获取的access_token,参考“Access Token获取”

Header：

参数	值
Content-Type	application/json

Body中放置请求参数，参数详情如下：

参数	值
verify_token	通过access_token获取的verify_token

请求示例：

```
 {
 "verify_token"："clupeyP51sn28XzxGVTfYqoN"
}
```

返回参数

- 返回结果

字段	必选	类型	说明
success	是	boolean	返回请求是否成功信息。 若请求成功返回ture; 请求失败则返回fault
result	是	array	请求结果
+身份证识别	是	array	身份证识别接口请求的统计信息
++error_code	是	string	错误码编号 若为0则表示请求成功
++count	是	int	当前错误码的请求数量
+h5活体视频分析	是	array	h5活体视频分析接口请求的统计信息
++error_code	是	string	错误码编号 若为0则表示请求成功
++count	是	int	当前错误码的请求数量
+人脸实名认证-V3	是	array	人脸实名认证-V3接口请求的统计信息
++error_code	是	string	错误码编号 若为0则表示请求成功
++count	是	string	当前错误码的请求数量

- 返回示例

```
{
  "success": true,
  "result": {
    "身份证识别": [
      {
        "error_code": 0,
        "count": 2
      }
    ],
    "h5活体视频分析": [
      {
        "error_code": 0,
        "count": 1
      }
    ],
    "人脸实名认证-V3": [
      {
        "error_code": 0,
        "count": 1
      }
    ]
  },
  "log_id": "1405335905"
}
```


金融级H5实名认证方案

方案简介

方案简介

推出背景

- 现在，人脸识别技术被广泛应用在金融支付、用户注册、人脸登录等业务场景中。技术的进步方便用户的同时，黑灰产产业也开始对这些场景产生觊觎。并通过**屏幕攻击、照片、纸张、以及面具、头模**等方式进行非法攻击。随着黑产技术的进步，更是出现了通过**自动化脚本直接攻击云端API、ROM注入、视频劫持替换、批量虚拟机、病毒侵入**等新型攻击手段。使现有的人脸识别方案面临着巨大的安全挑战。
- 为提升人脸识别的安全性，保障客户的业务安全，人脸实名认证产品团队与百度安全实验室联合推出**百度H5人脸实名认证方案**，在人脸登录、注册等环境加入层层保障，为您的业务保驾护航。

功能简介

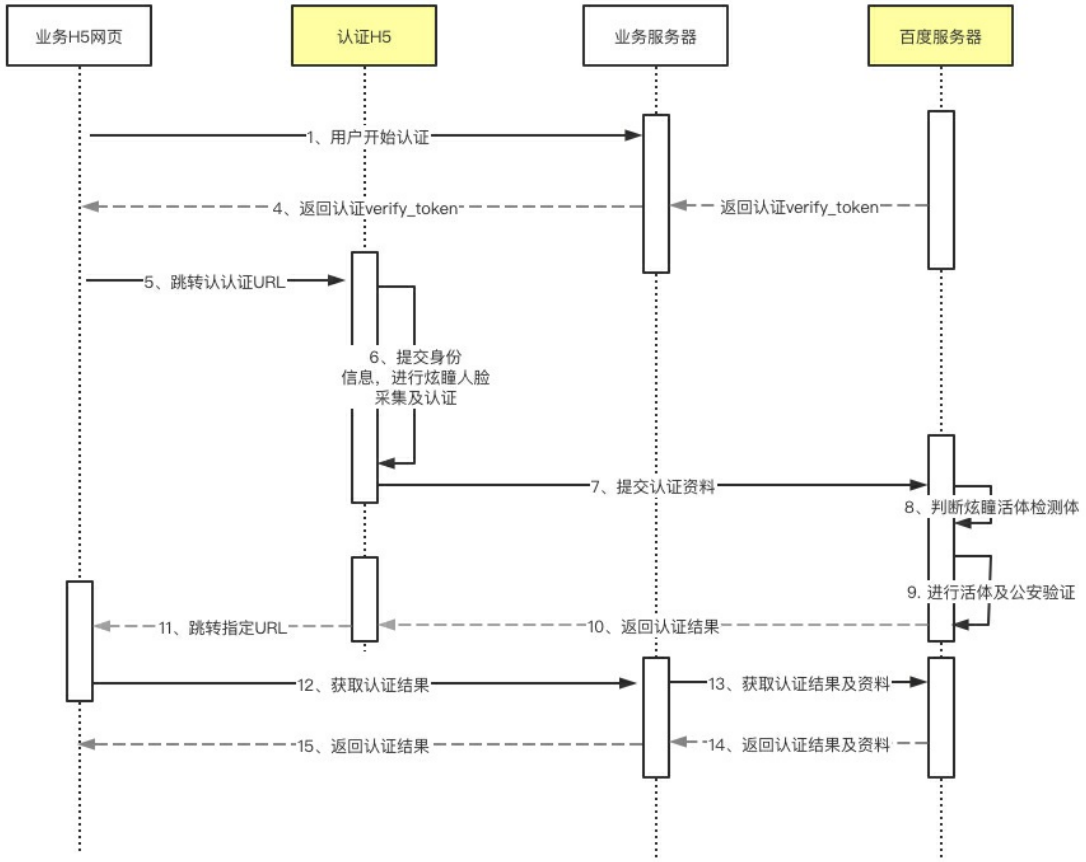
- 百度H5人脸实名认证方案提供标准化的人身核验流程，具有**人脸比对、证件识别、活体检测**等多项组合能力，提供视频、动作、炫瞳等多种活体检测方式供您选择，可抵挡屏幕、照片、视频、换脸、面具、3D模型的攻击。同时，百度H5实名认证方案中加入**大数据风控能力**，针对批量虚拟机、病毒侵入等新型攻击手段进行强力有效防御。

> **大数据风控能力：**人脸实名认证API接口接受SDK端传入的设备指纹信息，基于百度海量大数据设备因子，对JS-SDK端进行设备风险识别，辨别是否为风险设备，返回识别结果。可有效防御黑产批量虚拟机、病毒侵入等攻击手段，降低第三方黑产攻破概率，提升业务安全性。
>
> **炫瞳活体检测（实时检测）：**基于屏幕颜色打光的方式，通过我们的面部反光和**瞳孔反光**对核验人员进行活体判断。相比于行业内传统的动作活体和视频活体检测方式，通过率大大提升，使用效率更加流畅便捷，有效拦截视频、图片伪造、3D面具、合成图等黑产攻击。 并且，百度H5实名认证方案采用**实时活体检测形式，无需用户拍摄上传视频**，直接在前端完成检测流程，提升整体核验流程的流畅度及用户体验。

适用场景

- 百度H5人脸实名认证方案适用于在微信公众号、H5页面等业务场景实现用户实名认证，如果您的场景是安卓/IOS系统的APP场景，推荐采用百度APP人脸实名认证方案。

接入时序图



方案接入流程

- 百度H5人脸实名认证方案具体接入步骤请参考方案接入指南

方案接入指南

本文档将帮助您完成H5实名认证方案的创建及接入全流程。

一、 准备工作

在正式集成前，需要做一些准备工作，完成一些账号、应用及方案配置，具体如下：

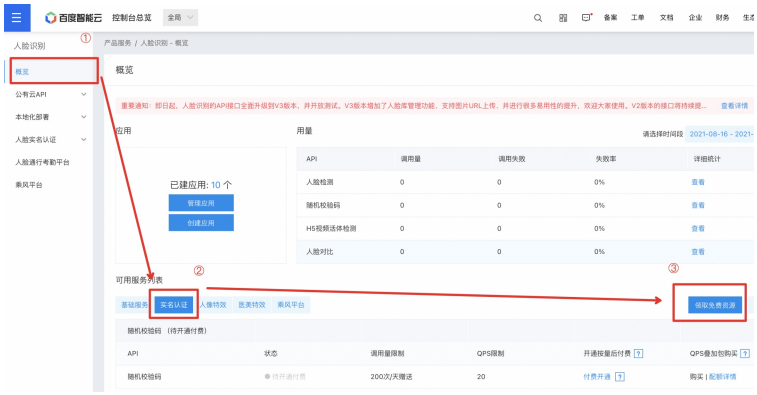
Step1: 注册成为开发者

> 在使用百度人脸实名认证方案之前，首先需注册百度智能云账号，账号注册方式请参考**账号注册指南**。
>
> 百度智能云账号注册完成以后，为顺利调用百度AI能力，需完成企业认证。具体认证方式请参考**企业认证指南**。

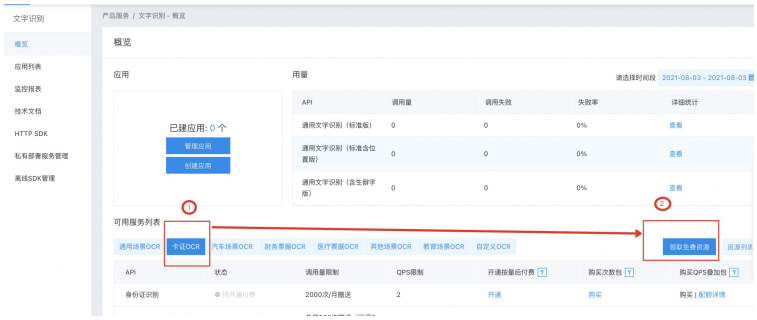
Step2：创建应用

2.1 输入应用名称，领取免费额度

- 创建好账号后，在正式调用AI 能力之前，需首先**创建应用**，应用是调用服务的基础能力单元。
- 同时领取接口所需的**免费调用额度**，用于接入测试。如下图所示：



- 除人脸服务接口的免费调用额度外，还需领取**身份证识别**接口的**免费调用额度**，用来调用身份证OCR识别功能（必须领取，否则会报错服务异常），点击此处，按下图所示进行领取。



- 如您之前已经领取过免费额度，无需重复领取，请跳至下一步骤。

2.2 勾选所需接口

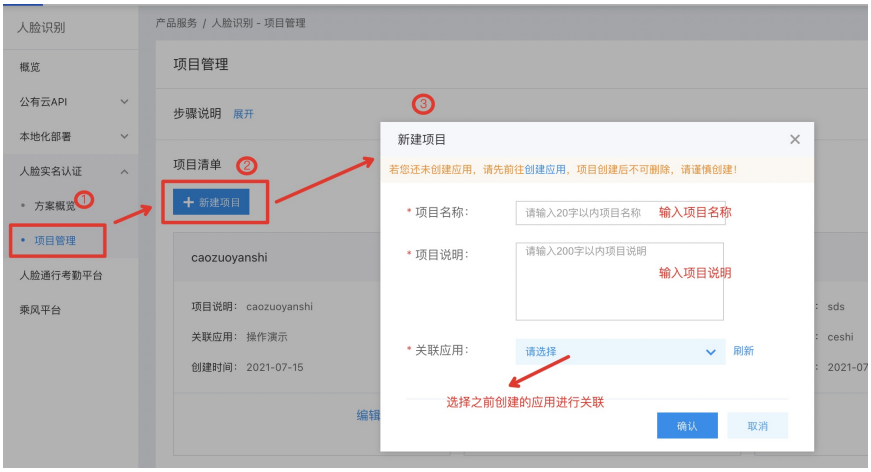
- 人脸识别服务相关接口已默认勾选且不可取消。

- **注：**「接口选择」过程中，还**需勾选「文字识别」中的「身份证识别」接口**，用于实现身份核验流程中的身份证识别功能。如下图所示。

Step3：创建项目

- 进入控制台-人脸实名认证页面，选择『**项目管理**』页面，点击『**新建项目**』，进行项目创建，如下图所示。

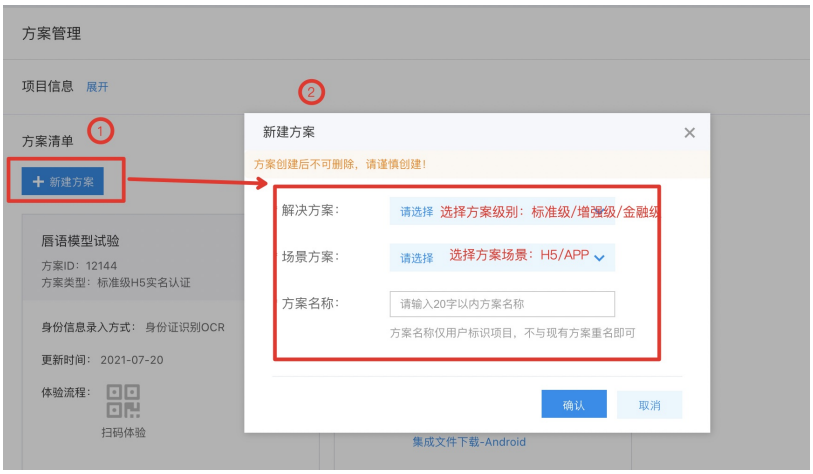
> 创建项目前，请确保您在应用控制台已创建应用，若您未创建应用，请参考Step2创建应用后，再进行项目创建.



Step4：创建方案

- 项目创建完成后，点击「**方案管理**」进入方案管理页面，在这里您可以为您的项目创建不同的方案，如下图所示。

> 若您的场景为APP场景（安卓/IOS系统），『场景方案』请选择APP实名认证方案；
>
> 若您的场景为微信、H5页面，『场景方案』请选择H5实名认证方案。



4.1 标题设置

例：当输入的内容为『标题名称创建演示』时，将于H5实名认证页面的头部进行展示，如下图

4.2 授权声明配置

在这里您可以配置授权声明的信息，您也可以选择跳过此配置。

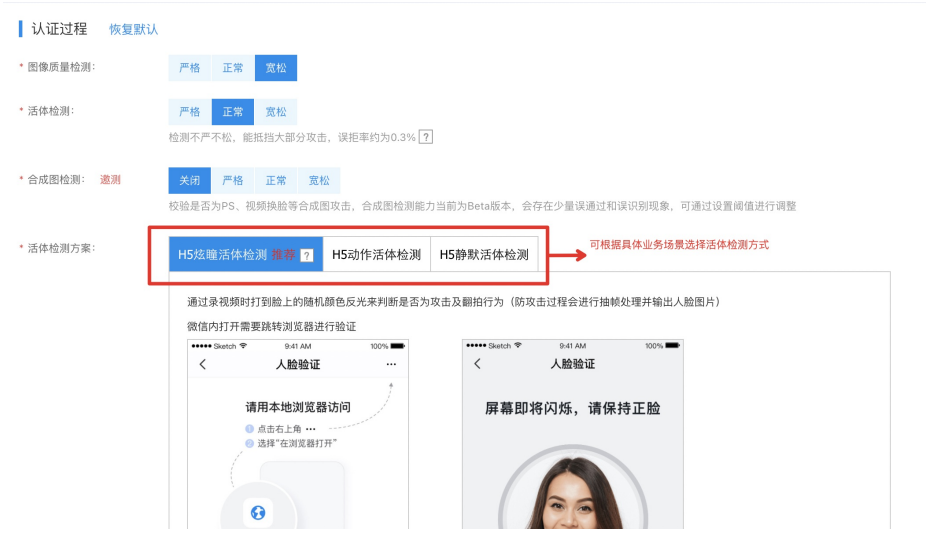
4.3 身份证信息录入配置

- **非大陆数据源**：默认不使用。配置打开后支持对大陆居民二代身份证、港澳居民往来内陆通行证、外国人永久居住证及定居国外的中国公民护照的验证。若不打开则只支持大陆居民二代身份证的验证。
- **身份信息录入方式**：支持身份证识别OCR、手动输入、指定用户身份核验三种。

> - 当选择**身份证识别OCR**后,支持配置**身份证风控功能**，开启后，对身份证的真伪进行判断，支持识别翻拍、PS伪造的身份信息，但开启后会存在少量误通过和误识别现象。同时支持配置身份证**人像面+国徽面及人像面的选择**，选择后通过查询接口可以查询到接口的返回信息。
> - 当选择**手动输入**后，支持用户自己输入姓名及身份证号信息。
> - 当选择**指定用户身份核验**后，支持指定用户的姓名+身份证号信息，用户侧无需输入，只需进行人脸采集及活体检测即可。**注意：**若需要指定用户身份核验，则需要先请求 指定用户信息上报接口 再请求金融级H5实名认证方案的URL。

4.4 认证过程配置

此步骤对业务场景拍摄的人脸图片的质量信息及活体检测宽松程度进行配置。



- **图像质量检测**：分为严格、正常、宽松三个等级，越严格，图片对角度、模糊度、遮挡等信息参数把控越高，推荐使用宽松。
- **活体检测**：分为严格、正常、宽松三个等级，越严格，对活体的检测等信息把控越高，推荐使用正常。
- **合成图检测**：分为严格、正常、宽松三个等级，越严格，对合成图的检测等信息把控越高，推荐使用正常。
- **活体检测方案**：

> - **炫瞳活体检测（实时）**：基于屏幕颜色打光的方式，通过面部反光和**瞳孔反光**对核验人员进行活体判断。相比于行业内传统的动作活体和视频活体检测方式，通过率大大提升，使用效率更加流畅便捷，有效拦截视频、图片伪造、3D面具、合成图等黑产攻击。 并且，金融级H5实名认证方案采用**实时活体检测形式**，**无需用户拍摄上传视频**，直接在前端完成检测流程，提升整体核验流程的流畅度及用户体验。
> - **动作活体检测（实时）**：通过用户做指定动作来验证当前拍摄视频的用户是否为活体。
>   - 支持指定动作的个数进行验证，支持设置指定1-3个验证动作，推荐使用1或者2个动作进行验证。
> - **静默视频活体检测（实时）**：通过用户录制一段视频来验证当前拍摄视频的用户是否为活体。 **附录：**
>
> **活体检测阈值指标** （高于此阈值即判断为活体）
>
> | 控制度 | 对应阈值 | 说明 |
> | --- | --- | --- |
> | 宽松 | 0.05 | 万分之一活体误拒率 |
> | 正常（推荐） | 0.3 | 千分之一活体误拒率 |
> | 严格 | 0.9 | 百分之一活体误拒率 |

4.5 认证结果配置

通过上传「姓名+身份证号+人脸图片」三要素到人脸实名认证接口，通过调取权威数据源判断是否为本人操作。

- **提供的认证结果页面**：您可以选择是否使用百度提供的展示给用户认证结果的页面，若您选择自己开发，可选择不使用。
- **认证未通过时URL是否失效**：当用户认证未通过时，部分用户会出现多次重新请求验证的情况，您可通过此项配置控制用户认证数。
- **阈值**：此阈值设置的是用户上传图片与公安权威数据源图片进行比对后得分的阈值，高于此阈值即判断为用户本人。**阈值设置推荐为80**，您可通过实际业务场景继续调整。

4.6 微信渠道内活体方案配置

当您的业务环境是微信渠道时，需对此项进行配置。

当您的业务环境非微信渠道时，此项配置不生效。

您可点击选项旁的视频button观看核验流程视频作为参考。如下图所示。

- **【不跳出，微信内降级】**：当选择此选项时，整个核验流程于微信内完成，无需跳转至外部环境，活体检测方式由实时检测方式降级至视频录制方式。您需要选择降级后的活体检测方案，包括H5数字活体检测、H5动作活体检测、H5视频活体检测、在线图片检测4种降级方案。如下图所示：

- **【跳出至外部浏览器】**：当选择此选项时，核验流程需由微信内跳转至外部浏览器完成，核验完成后再跳转回微信内，活体检测方式为实时检测，即在4.4认证过程配置中选择的活体检测方案。

Step5：提交方案，扫码预览 方案配置完成后，点击提交按钮，进入方案管理页面，鼠标移入「扫码体验」即可显示二维码信息，扫码即可体验H5实名认证流程，如下图所示。

> 注意：请谨慎修改上述H5方案配置，在点击「提交」后会实时对方案配置进行更新。若您需要修改方案，请在不影响线上业务的情况下进行调整。

二、 方案接入

Step1：获取token 通过获取Token接口获取verify_token信息 **Step2：跳转实名认证H5 URL，用户进行操作** 业务H5网页通过获取Token接口返回的verify_token信息请求认证H5页面，进行用户端流程操作。

认证URL： https://brain.baidu.com/face/print/?token=xxx&successUrl=https://xxx&failedUrl=https://xxx%3CBr%3E

URL中的信息填写如下所示：

（1）**token：**填写verify_token，verify_token获取参考获取verify_token参考文档。

（2）**successUrl：**请求成功跳转的网址，网址需要加http/https前缀

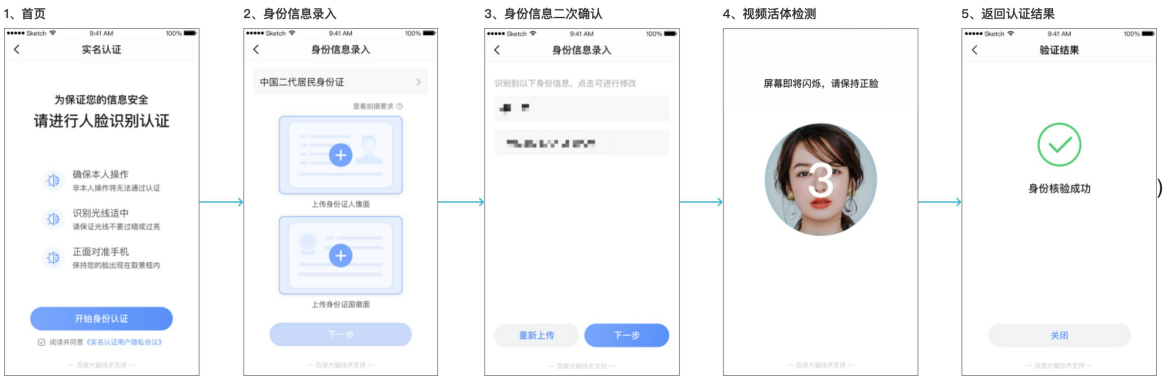
（3）**failedUrl：**请求失败跳转的网址，网址需要加http/https前缀

（4）**successUrl和failedUrl推荐使用encodeURIComponent进行转义**，不然可能无法正确跳转，转义示例如下：

```
encodeURIComponent("https://ai.baidu.com")
```

（5）若需要指定用户进行核验，在方案中身份信息录入-身份信息录入方式-指定用户身份核验，则需要先请求指定用户信息上报接口再请求url。

操作流程参考如下：



Step3：获取认证结果及资料，返回用户认证信息 可以参考获取认证人脸接口文档、查询认证结果接口文档、查询统计结果接口文档查询用户人脸实名认证流程的相关信息，以进行后续业务集成开发。

接口文档

一、方案功能接口

1.获取verify_token接口

本接口为H5实名认证方案的verify_token获取接口，利用所获取的verify_token进行实名认证流程的有效期为2小时。

调用方式

请求URL数据格式

向API服务地址使用POST发送请求，必须在URL中带上参数access_token，可通过后台的API Key和Secret Key生成，具体方式请参考“Access Token获取”。

> **注意**：access_token的有效期为30天，**切记需要每30天进行定期更换，或者每次请求都拉取新token**；

POST中Body的参数，按照下方请求参数说明选择即可。

> **提示**：如果您为百度云老用户，正在使用其他非AI的服务，可以参考百度云AKSK鉴权方式发送请求，虽然**请求方式**和**鉴权方法**和本文所介绍的不同，但请求参数和返回结果一致。

请求说明

注意事项：

- **请求体格式化**：Content-Type为application/json，通过json格式化请求体。

请求示例

HTTP方法：POST

请求URL： https://aip.baidubce.com/rpc/2.0/brain/solution/faceprint/verifyToken/generate

URL参数：

参数	值
access_token	通过API Key和Secret Key获取的access_token,参考“Access Token获取”

Header：

参数	值
Content-Type	application/json

Body中放置请求参数，参数详情如下：

请求参数：

参数	必选	类型	说明
plan_id	是	string	方案的id信息，请在人脸实名认证控制台查看创建的H5方案的方案ID信息

请求示例：

```
{
"plan_id"：1
}
```

返回参数

- 返回结果

字段	必选	类型	说明
success	是	boolean	返回请求是否成功信息。 若请求成功返回ture; 请求失败则返回false
result	是	array	请求结果
+verify_token	是	string	请求获取的verify_token

- 返回示例

```
{
  "success": true,
  "result": {
     "verify_token": "Yz9rWITm4vak16PBAh5x8oG7"
  },
  "log_id": "1814798895"
}
```

2.指定用户信息上报接口 本接口用于，前端在方案中选择身份信息录入-身份信息录入方式-指定用户身份核验时，需要先调用此接口输入指定用户的姓名+身份证号信息，再请求url跳转页面。

调用方式

请求URL数据格式

向API服务地址使用POST发送请求，必须在URL中带上参数`access_token`，可通过后台的API Key和Secret Key生成，具体方式请参考“Access Token获取”。

> **注意**：`access_token`的有效期为30天，**切记需要每30天进行定期更换，或者每次请求都拉取新token**；

POST中Body的参数，按照下方请求参数说明选择即可。

> **提示**：如果您为百度云老用户，正在使用其他非AI的服务，可以参考百度云AKSK鉴权方式发送请求，虽然**请求方式**和**鉴权方法**和本文所介绍的不同，但请求参数和返回结果一致。

请求说明

注意事项：

- **请求体格式化**：Content-Type为`application/json`，通过`json`格式化请求体。

请求示例

HTTP方法：`POST`

请求URL：https://brain.baidu.com/solution/faceprint/idcard/submit

> 注意这里不需要传access_token

Header：

参数	值
Content-Type	application/json

Body中放置请求参数，参数详情如下：

参数	必选	类型	值
verify_token	是	string	通过access_token获取的verify_token
id_name	是	string	指定输入用户的姓名信息
id_no	是	string	指定输入用户的身份证件号信息
certificate_type	否	Int	证件类型： 0大陆居民二代身份证 1港澳台居民来往内地通行证 2外国人永久居留证 3定居国外的中国公民护照

请求示例：

```
{
  "verify_token": "2sF3nE5mXOHkx2aQwWG4n5WI",
  "id_name": "张三",
  "id_no": "500***********3390",
  "certificate_type": 0          // 证件类型：0大陆居民二代身份证，1港澳台居民来往内地通行证，2外国人永久居留证，3定居国外的中国公民护照
}
```

返回参数

- 返回结果

字段	必选	类型	说明
success	是	boolean	返回请求是否成功信息。 若请求成功返回ture; 请求失败则返回false
result	是	array	请求结果，返回固定结果1，可忽略

- 返回示例

```
{
  "success": true,
  "result": 1,
  "log_id": "1244068892"
}
```

二、验证后查询接口

> 获取Token后，请先按照跳转实名认证H5 URL，用户进行操作后再查询接口，否则生成Token无法生效。

1.获取认证人脸接口

本接口返回进行人脸实名认证过程中进行认证的最终采集的人脸信息（仅在认证成功时返回人脸信息，认证失败返回错误码）。根据Verify_token返回的结果信息会在云端保留两个小时，您可根据需要在此期间进行调取查询。

调用方式

请求URL数据格式

向API服务地址使用POST发送请求，必须在URL中带上参数access_token，可通过后台的API Key和Secret Key生成，具体方式请参考“Access Token获取”。

> **注意**：access_token的有效期为30天，**切记需要每30天进行定期更换，或者每次请求都拉取新token**；

POST中Body的参数，按照下方请求参数说明选择即可。

> **提示**：如果您为百度云老用户，正在使用其他非AI的服务，可以参考百度云AKSK鉴权方式发送请求，虽然**请求方式**和**鉴权方法**和本文所介绍的不同，但请求参数和返回结果一致。

请求说明

注意事项：

- **请求体格式化**：Content-Type为application/json，通过json格式化请求体。

请求示例

HTTP方法：POST

请求URL： https://aip.baidubce.com/rpc/2.0/brain/solution/faceprint/result/simple

URL参数：

参数	值
access_token	通过API Key和Secret Key获取的access_token,参考“Access Token获取”

Header：

参数	值
Content-Type	application/json

Body中放置请求参数，参数详情如下：

参数	值
verify_token	通过access_token获取的verify_token

请求示例：

```
 {
 "verify_token"："cIupeyP51sn28XzxGVTfYqoN"
}
```

返回参数

- 返回结果

字段	必选	类型	说明
success	是	boolean	返回请求是否成功信息。 若请求成功返回ture; 请求失败则返回fault
result	是	array	请求结果
+image	是	string	返回采集的用户人脸信息 （仅在认证成功时返回人脸信息，认证失败返回错误码）

- 返回示例

```
{
  "success": true,
  "result": {
     "image":"https://brain.baidu.com/solution/faceprint/image/query?verify_token=xxxxxx"
  },
  "log_id": "1054986003"
}
```

2.查询认证结果接口

本接口为请求返回的认证结果信息查询，包含身份证OCR识别信息、用户二次确认的身份证信息，活体检测信息、及用户对权威数据源图片进行比对的分数信息。（仅在认证成功时返回上述信息，认证失败返回错误码）根据Verify_token返回的结果信息会在云端保留三天，您可根据需要在此期间进行调取查询。

调用方式

请求URL数据格式

向API服务地址使用POST发送请求，必须在URL中带上参数`access_token`，可通过后台的API Key和Secret Key生成，具体方式请参考“Access Token获取”。

> **注意**：`access_token`的有效期为30天，**切记需要每30天进行定期更换，或者每次请求都拉取新token**；

POST中Body的参数，按照下方请求参数说明选择即可。

> **提示**：如果您为百度云老用户，正在使用其他非AI的服务，可以参考百度云AKSK鉴权方式发送请求，虽然**请求方式**和**鉴权方法**和本文所介绍的不同，但请求参数和返回结果一致。

请求说明

注意事项：

- **请求体格式化**：Content-Type为`application/json`，通过json格式化请求体。

请求示例

HTTP方法：`POST`

请求URL： `https://aip.baidubce.com/rpc/2.0/brain/solution/faceprint/result/detail`

URL参数：

参数	值
access_token	通过API Key和Secret Key获取的access_token,参考“Access Token获取”

Header：

参数	值
Content-Type	application/json

Body中放置请求参数，参数详情如下：

参数	值
verify_token	通过access_token获取的verify_token

请求示例：

```
 {
 "verify_token"："clupeyP51sn28XzxGVTfYqoN"
}
```

返回参数

- 返回结果

字段	必选	类型	说明
success	是	boolean	返回请求是否成功信息。 若请求成功返回ture; 请求失败则返回fault
result	是	array	请求结果
+idcard_ocr_result	否	array	返回采集的身份证信息 当人脸实名认证控制台设置为使用OCR识别时返回此参数信息
++address	否	string	地址
++birthday	否	string	生日
++name	否	string	姓名
++id_card_number	否	string	身份证号
++gender	否	string	性别
++nation	否	string	民族
++expire_time	否	string	身份证失效日期
++issue_authority	否	string	身份证签发机关
++issue_time	否	string	身份证生效日期
+idcard_images	否	array	返回采集的身份证图片信息 当人脸实名认证控制台设置为使用OCR识别时返回此参数信息
++front_base64	否	string	身份证图片的正面信息
++back_base64	否	string	身份证图片的反面信息 当人脸实名认证控制台设置为使用OCR识别且为国徽面+人像面时返回此参数信息
+verify_result	是	array	认证返还信息
++liveness_score	是	string	**活体检测分数：** 在线图片/动作活体：活体验证通过时返回活体分数，不通过则返回0。 炫瞳活体：活体通过/不通过均会返回0
++score	是	string	人脸实名认证
++spoofing	是	string	合成图分数 若未进行合成图检测，则返回0 若进行活体检测，则返回合成图检测分值
+idcard_confirm	是	array	用户二次确认的身份证信息
++name	是	string	姓名
++idcard_number	是	string	身份证号

- 返回示例

```
{
 "success": true,
 "result": {
    "verify_result": {
        "score": 93.7835,
        "liveness_score": 0.9672966,
        "spoofing": 0.0
    },
    "idcard_ocr_result": {
        "birthday": "19960216",
        "issue_authority": "胶南市公安局",
        "address": "山东省**********",
        "gender": "女",
        "nation": "汉",
        "expire_time": "20221103",
        "name": "柴*",
        "issue_time": "20121103",
        "id_card_number": "370**********5826"
    },
    "idcard_images": {
        "front_base64": "/9j/4AAQSkZJRgAB....",
        "back_base64": "/9j/4AAQSkZJRgAB...."
 },
    "idcard_confirm": {
        "idcard_number": "370**********5826",
        "name": "柴*"
    }
 },
 "log_id": "160931948204246"
}
```


3.查询统计结果

根据Verify_token返回的结果信息会在云端保留三天，您可根据需要在此期间进行调取查询。

调用方式

请求URL数据格式

向API服务地址使用POST发送请求，必须在URL中带上参数`access_token`，可通过后台的API Key和Secret Key生成，具体方式请参考“Access Token获取”。

> **注意**：`access_token`的有效期为30天，**切记需要每30天进行定期更换，或者每次请求都拉取新token**；

POST中Body的参数，按照下方请求参数说明选择即可。

> **提示**：如果您为百度云老用户，正在使用其他非AI的服务，可以参考百度云AKSK鉴权方式发送请求，虽然**请求方式**和**鉴权方法**和本文所介绍的不同，但请求参数和返回结果一致。

请求说明

注意事项：

- **请求体格式化**：Content-Type为`application/json`，通过`json`格式化请求体。

请求示例

HTTP方法：POST

请求URL： https://aip.baidubce.com/rpc/2.0/brain/solution/faceprint/result/stat

URL参数：

参数	值
access_token	通过API Key和Secret Key获取的access_token,参考“Access Token获取”

Header：

参数	值
Content-Type	application/json

Body中放置请求参数，参数详情如下：

参数	值
verify_token	通过access_token获取的verify_token

请求示例：

```
 {
 "verify_token"："clupeyP51sn28XzxGVTfYqoN"
}
```

返回参数

- 返回结果

字段	必选	类型	说明
success	是	boolean	返回请求是否成功信息。 若请求成功返回ture; 请求失败则返回fault
result	是	array	请求结果
+身份证识别	是	array	身份证识别接口请求的统计信息
++error_code	是	string	错误码编号 若为0则表示请求成功
++count	是	int	当前错误码的请求数量
+h5活体视频分析	是	array	h5活体视频分析接口请求的统计信息
++error_code	是	string	错误码编号 若为0则表示请求成功
++count	是	int	当前错误码的请求数量
+人脸实名认证-V3	是	array	人脸实名认证-V3接口请求的统计信息
++error_code	是	string	错误码编号 若为0则表示请求成功
++count	是	string	当前错误码的请求数量

- 返回示例

```
{
  "success": true,
  "result": {
    "身份证识别": [
      {
        "error_code": 0,
        "count": 2
      }
    ],
    "h5活体视频分析": [
      {
        "error_code": 0,
        "count": 1
      }
    ],
    "人脸实名认证-V3": [
      {
        "error_code": 0,
        "count": 1
      }
    ]
  },
  "log_id": "1405335905"
}
```


人脸应用套件

产品概述

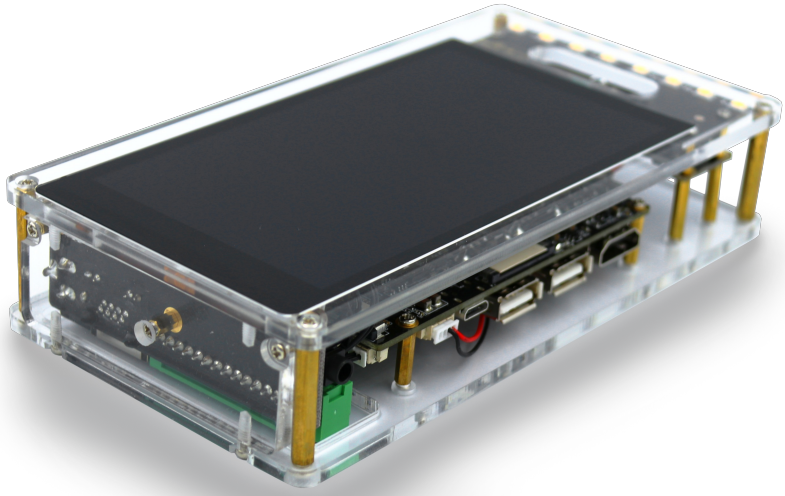
度目人脸应用套件

更新记录

当前在售型号：**H1-Pro**（包含5寸屏幕）

AI市场购买链接

H1-Pro



核心功能

1. **人脸注册**

   动态人脸采集--活体检测--特征提取--人脸和特征值入库

2. **人脸库管理**

   1. 人脸库统计
   2. 查询人脸库记录
   3. 删除特定人脸底库
   4. 清空人脸库

3. **设备管理**

   1. WLAN管理，包括wifi列表查询、热点连接/断开
   2. LAN管理，包括自动IP设置、手动IP设置

4. **IPC Web网页控制**

   1. 视频流预览，包括主码流（RGB）、辅码流（IR）、图像抓拍、实时录像
   2. 设置
      - 视频设置，包括视频质量参数、视频编码、图片参数、字符叠加等
      - 网络设置，包括有线网络参数设置、无线网络参数设置、RTSP参数等
      - 智能分析--门禁控制，包括补光灯控制、LCD屏保控制、开机默认识别模式、流程质量控制（端上视频流人脸注册）、图片质量控制（图片人脸注册）

5. **闸机模式**

   人脸检测--活体检测--特征提取--1:N特征比对（识别）--结果展示（显示+声音）--闸机控制--http上传识别记录

6. **考勤模式**

人脸检测--活体检测--特征提取--1:N特征比对（识别）--签到--http上传识别记录

7. **金融核验模式**

   人脸检测--活体检测--面部姿态判断--面部遮挡判断--1:1特征比对（证件照）--核验成功--人脸信息结果上报

适用场景

- 小区
- 学校
- 商场
- 酒店

常用链接

- 工单支持
- 开发者社区
- 供应商合作

软件说明文档

功能

1. **人脸注册**

   动态人脸采集--活体检测--特征提取--人脸和特征值入库

2. **人脸库管理**

   1. 人脸库统计
   2. 查询人脸库记录
   3. 删除特定人脸底库
   4. 清空人脸库

3. **设备管理**

   1. WLAN管理，包括wifi列表查询、热点连接/断开
   2. LAN管理，包括自动IP设置、手动IP设置

4. **IPC Web网页控制**

   1. 视频流预览，包括主码流（RGB）、辅码流（IR）、图像抓拍、实时录像
   2. 设置
      - 视频设置，包括视频质量参数、视频编码、图片参数、字符叠加等
      - 网络设置，包括有线网络参数设置、无线网络参数设置、RTSP参数等
      - 智能分析--门禁控制，包括补光灯控制、LCD屏保控制、开机默认识别模式、流程质量控制（端上视频流人脸注册）、图片质量控制（图片人脸注册）

5. **闸机模式**

   人脸检测--活体检测--特征提取--1:N特征比对（识别）--结果展示（显示+声音）--闸机控制--http上传识别记录

6. **考勤模式**

人脸检测--活体检测--特征提取--1:N特征比对（识别）--签到--http上传识别记录

7. **金融核验模式**

人脸检测--活体检测--面部姿态判断--面部遮挡判断--1:1特征比对（证件照）--核验成功--人脸信息结果上报

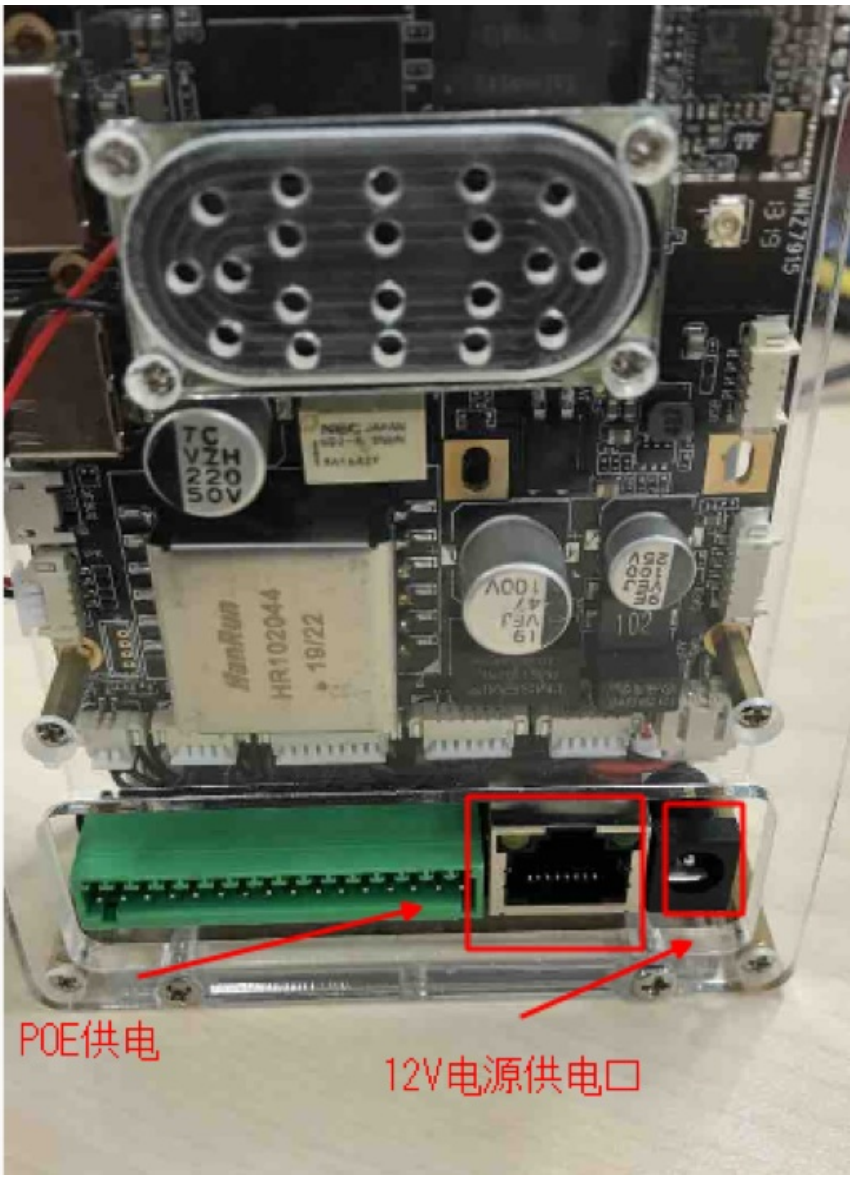
具体参数

名称	功能描述
操作系统	Linux
图像分辨率	支持1080p、720p图像
显示	支持5寸、7寸、8寸屏显示
模态	支持RGB和NIR模态并行采集、检测
活体检测	支持RGB和NIR活体，能抵抗99.99%的攻击
人脸检测速度	25fps
人脸识别性能	150ms内完成端到端人脸识别
检索	支持1:1和1:N的人脸检索，检索速度小于20ms（默认5W）
人脸库管理	最大支持10W的人脸底库的管理，支持人脸底库批量导入，增加，删除，查询
网络	支持RTSP网络视频流、支持HTTP标准网络协议
web	内嵌webserver，支持通过WebServer登录

操作说明

接通电源

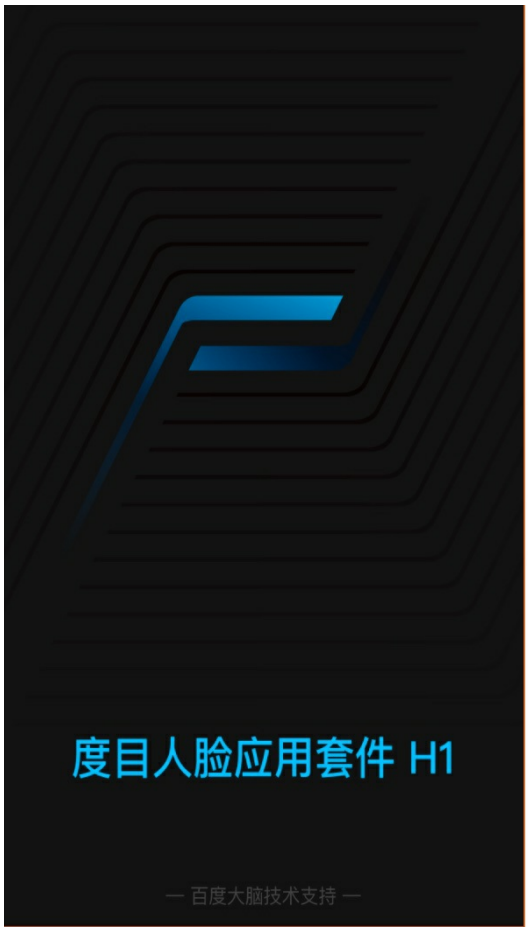
度目人脸应用套件可使用12V电源适配器独立供电，或者POE网线供电，用户可根据需求自行选择



接通电源后，系统进入开机画面，开始初始化，这个过程时间较长（大约30s），需要加载AI算法和人脸库，人脸库规模越大，时间越长，用户请耐心等待

系统初始化完毕，进入开机默认界面，该界面展示了度目人脸应用套件的所有功能：**设备管理**、**闸机模式**、**考勤模式**、**金融核验模式**、**人脸注册**、**人脸库管理**

初始化后的默认界面可通过web页面或者http协议设置，设置方法参考web页面设置和API



设备管理

点击图上图中，右上角  设置按钮，进入设备管理界面，该界面具备3种设备设置能力：

- WiFi设置，点击wifi列表中的相应热点名，可进行WLAN wifi热点列表展示、热点连接
- 有线网络设置，可进行LAN相关设置，包括手动获取IP、自动获取IP，IP地址、子网掩码、网关和DNS

设备管理同样可以在web页面和http协议中进行设置，参考web页面设置和API

人脸注册

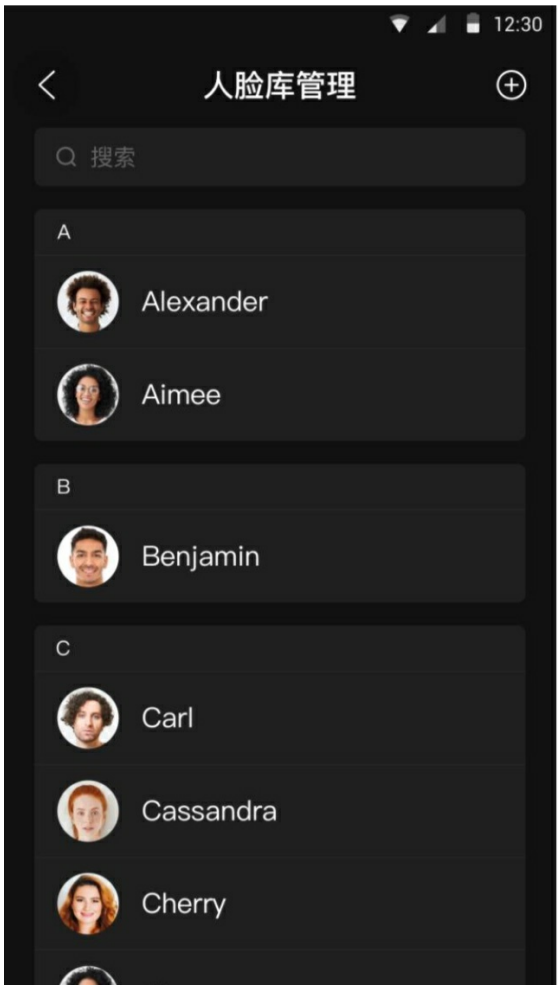
在开机默认界面，点击**人脸注册**按钮，进入人脸注册界面，请根据界面提示，将人脸放置于圆圈内，当人脸四周出现红色框时，表示人脸质量不符合要求，请适当调整面部姿态、距离，直到红色框变成绿色，若长时间未检测到绿色框，系统会给出提

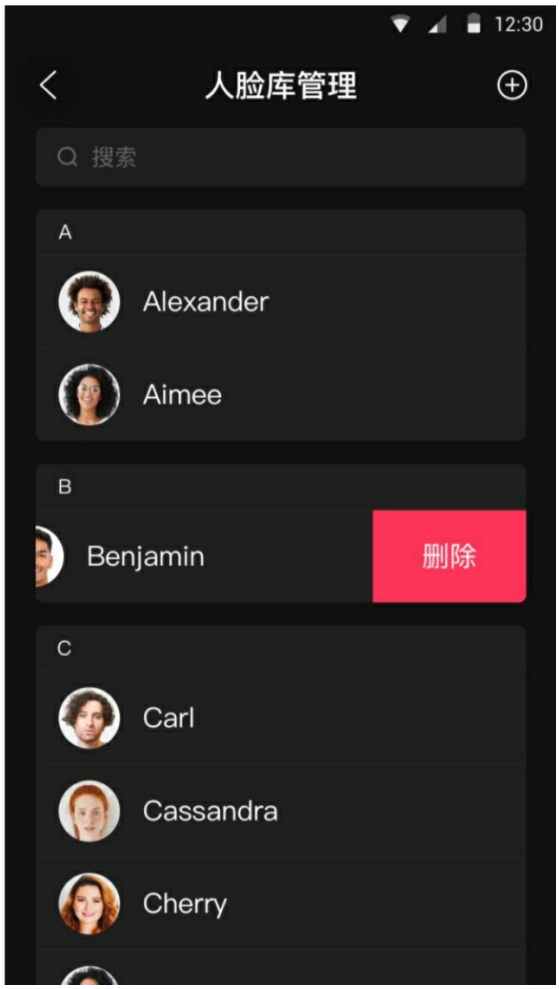
示，并且要求用户选择是否继续注册人脸

当人脸质量符合要求时（绿色人脸框），界面中心圆圈将开始顺时针旋转，请注意保持姿势不变，待旋转一圈时，完成人脸注册，输入注册用户名

人脸库管理

在开机默认界面，点击**人脸库管理**按钮，进入人脸库管理界面，该界面展示了已注册人脸信息，最下方显示总数以及“清空数据库”按钮，点击该按钮可一键清除所有人脸库，也可以选中某个人脸信息，点击右侧出现的“删除”按钮





闸机模式

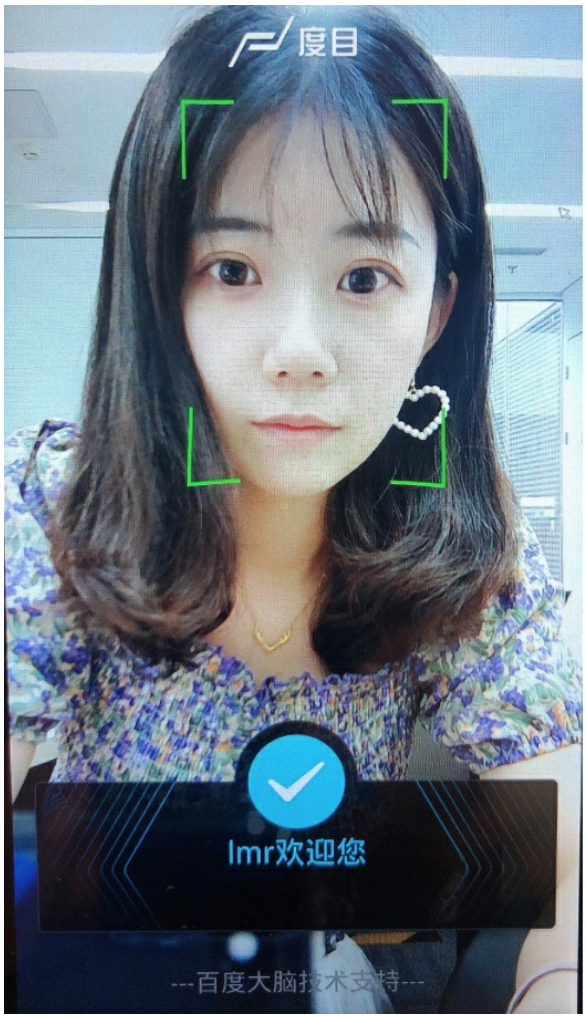
在开机默认界面，点击**闸机模式**按钮，进入闸机模式界面（如下图左图），该模式下系统实时检测人脸、活体检测、特征提取、1:N比对识别，识别成功会由声音、画面提示，如果连接闸机端子，同时还会输出控制信号。系统可同时检测多张人脸，但只有显示绿色框的人脸被送去做活检和识别。

考勤模式

在开机默认界面，点击**考勤模式**按钮，进入考勤模式界面（如下图左图），该界面下显示当前时间，当有人脸被检测到，并完成活体检测、特征提取、1:N识别成功后，记录签到时间，并通过http协议上传到控制中心（控制中心需要单独开发），协议请参考API

金融核验模式

在开机默认界面，点击**金融核验模式**按钮，进入金融核验模式界面，该模式专门为金融核验场景设计，它对注册环节采取更严格的条件限制，确保确保采集人脸质量、角度、遮挡、活体等条件均满足金融场景要求，当前该模式的使用方式类似人脸注册

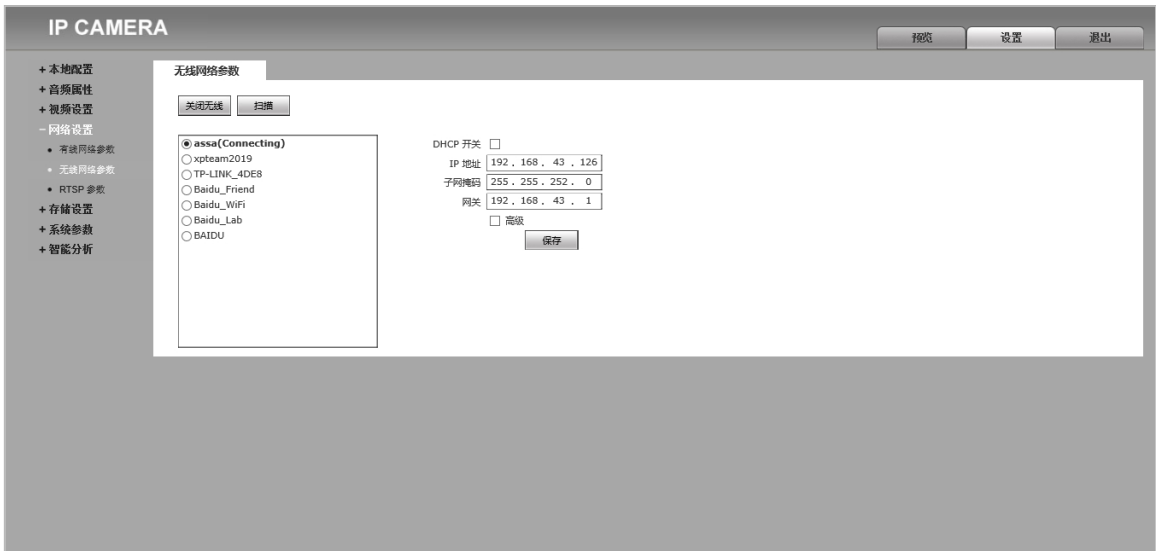


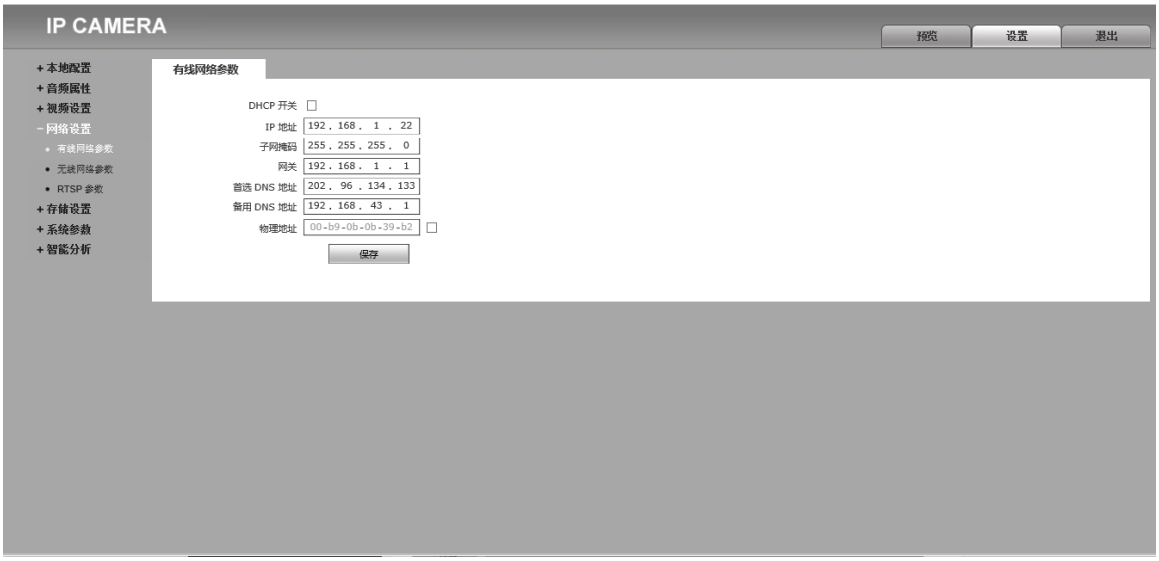
web页面设置

准备一台windows电脑，将电脑和套件接入同一局域网内，并确保电脑与套件在同一网段内，例如：套件默认IP地址是192.168.1.88，则电脑可将IP地址设置为192.168.1.100，打开IE浏览器，在网址处输入192.168.1.88，则登录套件的web页面，首次登录会强制设置用户名和密码

> **用户名：admin，密码：123456**

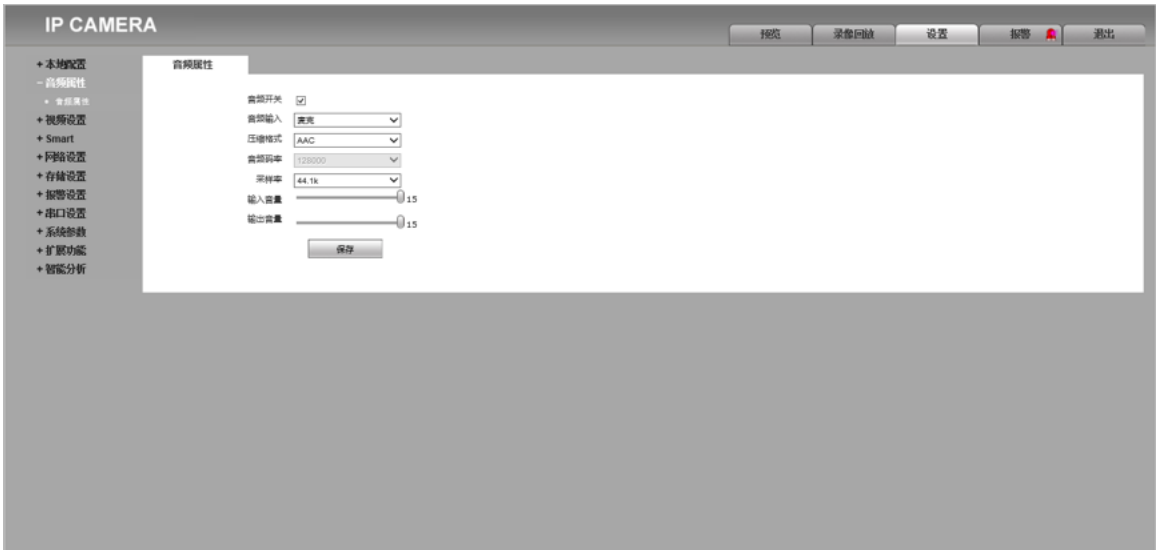
- **网络设置**：
  - 有线网络参数，可以进行IP的相关测试修改，例如：IP、子网掩码、网关等
  - 无线网络配置，可以进行搜索到无线网络进行连接





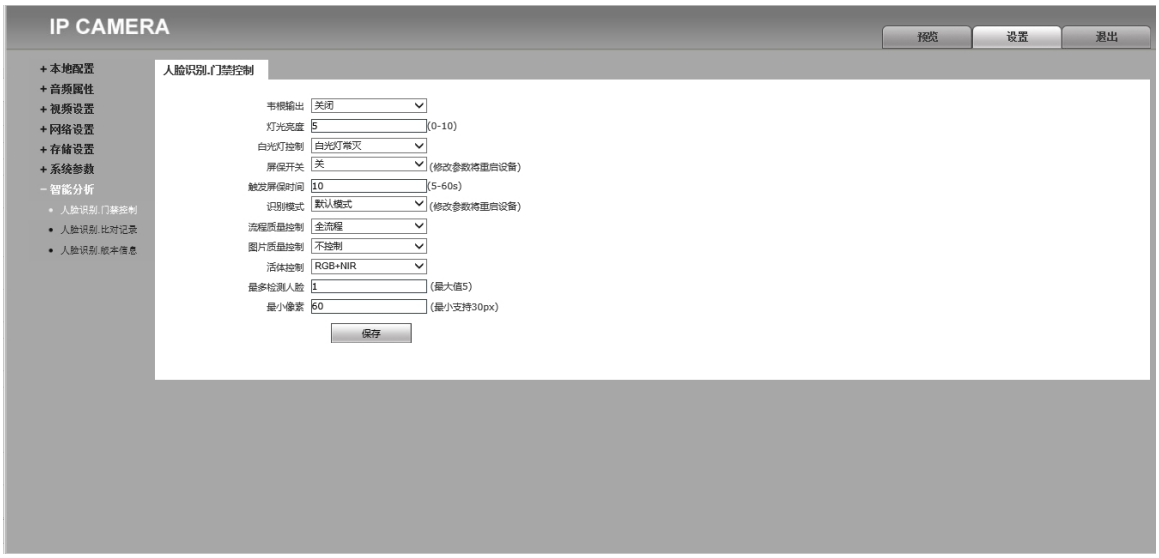
- **音频属性**：

音频属性，设置输出音量控制（0~15），喇叭的声音大小



- **智能分析**

人脸识别门禁控制



1. 灯光亮度：0~10s
2. 屏保开关：开/关（设置完必须重启生效）
3. 触发屏保时间：0~60s（设置完必须重启生效）
4. 识别模式：（设置完必须重启生效）
    - 门禁闸机模式：参考闸机模式
    - 考勤打开模式：参考考勤模式
    - 金融核验模式：参考金融核验模式
5. 流程质量控制：

    该设置项用于选择某个流程（环节），与“图片质量控制”相结合，共同确定一个质量控制等级，该设置项包括3类流程（环节），参考HTTP协议设置章节：

    - 全流程：设置所有流程（环节），包括图片注册流程+活体检测识别流程
    - 图片注册流程：仅设置图片注册流程（环节）
    - 识别流程：仅设置识别流程（环节）
6. 图片质量控制：

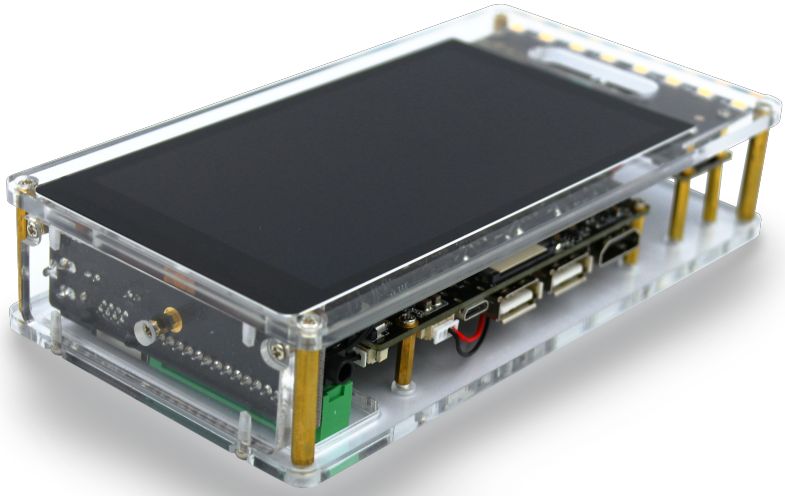
    该设置项对图片的质量等级进行设置，主要有4个等级，参考API

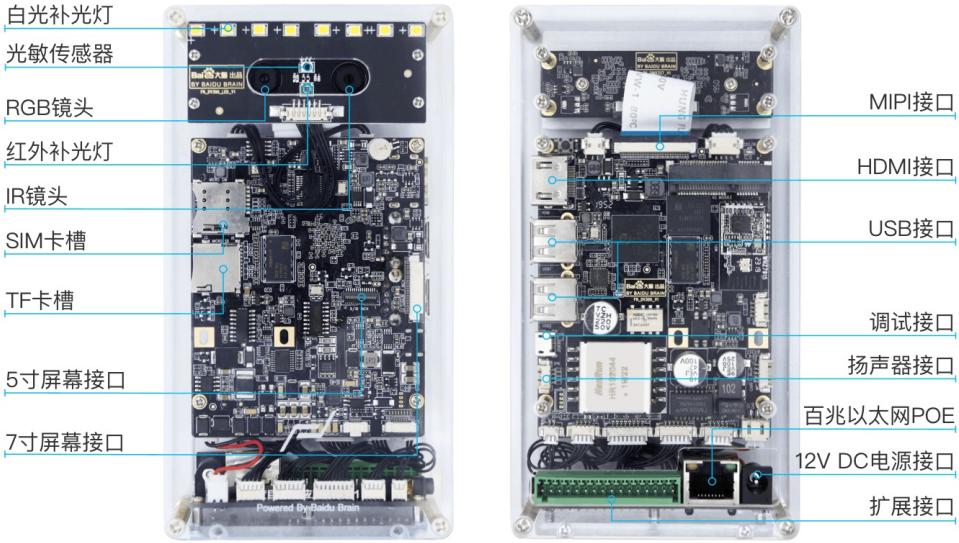
    - 不控制：不对图片质量进行判断，低质量图片也会被允许
    - 较低：对图片提出较低的质量要求，例如：遮挡0.8、角度30°、光照不要求、模糊度 0.8等
    - 一般：对图片提出一般的质量要求，例如：遮挡0.6、角度20°、光照在40-255、模糊度 0.6
    - 较高：对图片提出较高的质量要求，例如：遮挡0.3、角度15°、光照在60-120、模糊度 0.3

硬件说明文档

外观说明

H1-Pro



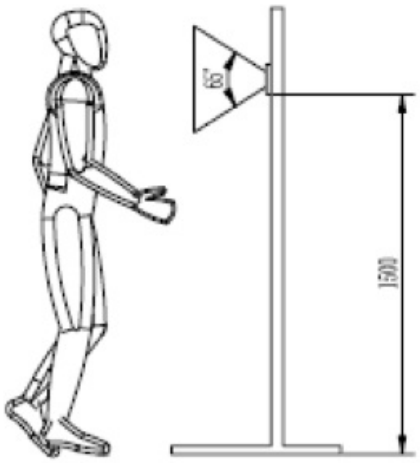
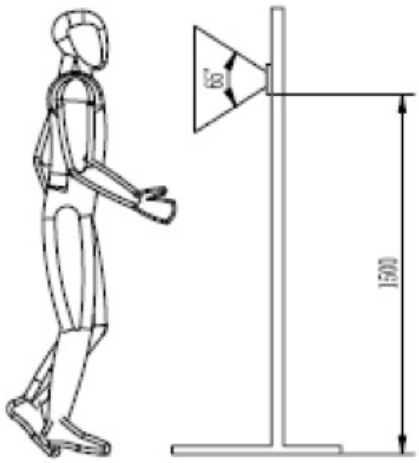
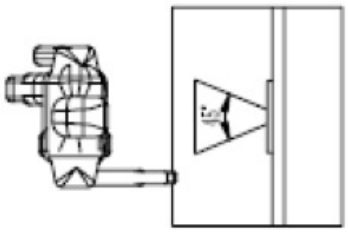
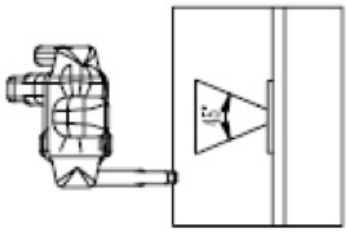

白光补光灯
光敏传感器
RGB镜头
红外补光灯
IR镜头
SIM卡槽
TF卡槽
5寸屏幕接口
7寸屏幕接口
MIPI接口
HDMI接口
USB接口
调试接口
扬声器接口
百兆以太网POE
12V DC电源接口
扩展接口

参数

硬件参数

名称	功能描述
CPU	HI3516DV300
内存	DDR4L 1GB
存储容量	8GB Emmc Mirco SD扩展，最大64GB
网络	100M/10M自适应有线网络 2.4G/5.8G 双频 WIFI，支持802.11a/b/g/n
图传传感器	双SONY 1/2.8 sensor，200W分辨率，支持RGB + IR
补光功能	自动或手动控制白光和红外补光，可以控制补光强度
显示	HDMI或者TFT液晶屏显示，可支持5、7、8三种TFT液晶屏
触摸屏	配套的5、7、8寸触摸屏进行交互输入
外设	1路RS485，一路RS232 2路USB2.0 外设接口 1路韦根协议 2路继电器输出（第二路驱动电流100Ma），1路报警输入 1路薄膜键盘接入 1路身份证读卡器输入接口 3W@4R喇叭，支持麦克风输入
电源输入	DC12V 5.5mm/2.1MM的DC头输入，最高电压支持24V POE供电,最大25W输出能力
功耗	5W@12V(不包含补光灯)
工作温度	-20~+60度@90%RH
套件尺寸	155mm*84mm*32mm（长、宽、高）

镜头参数

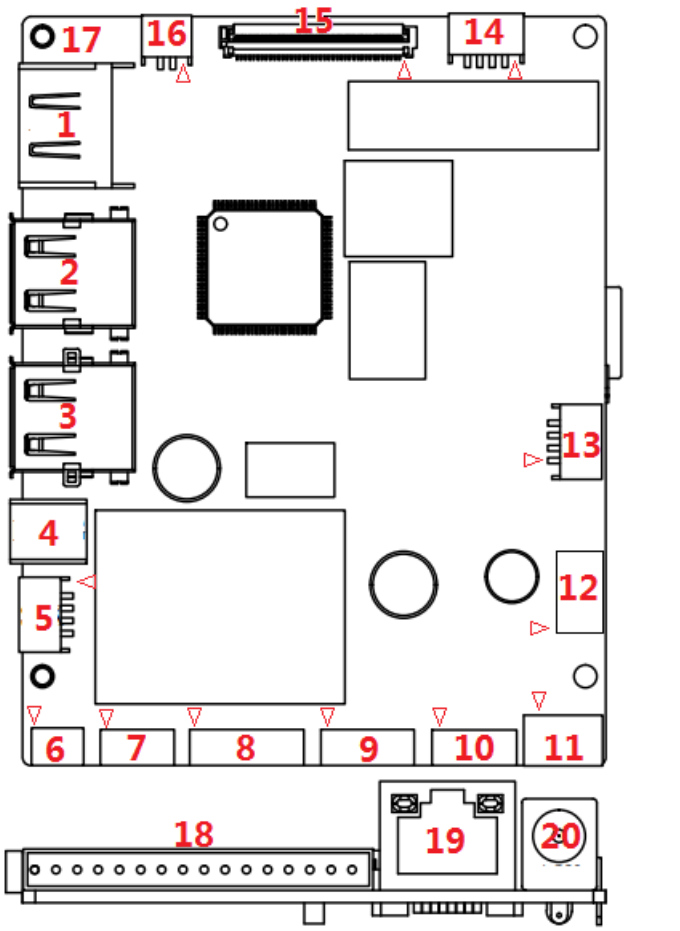
名称	参数	参数
类型	RGB摄像头	IR红外摄像头
焦距	EFL 4.3mm	EFL 4.3mm
光学总长（TTL）	7.8mm	7.8mm
畸变率	<5%	<5%
视场角(对角/水平/垂直)	73 / 65 / 40	73 / 65 / 40
分辨率	200万	200万
最低照度	0.01LUX @ F2.0	0.01LUX @ F2.0
帧率	30FPS	30FPS
信噪比	50DB	50DB
动态范围	105DB	105DB
输出格式	RAW12	RAW12
垂直视角		
水平视角		

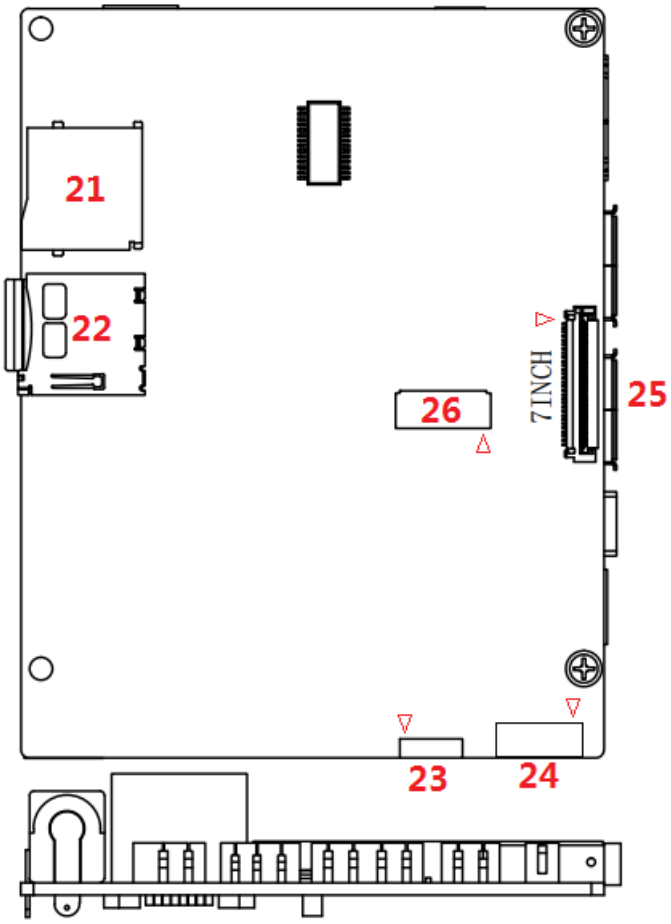
硬件接口

本节描述模块的外部连接器及有关信号的定义

位置接口结构图

板上的接口连接器和配置跳线的位置，如下图所示，**三角形引脚为第一脚**





连接器综述

连接器	功能	尺寸
1	显示接口	Type A
2	USB接口	Type A
3	USB接口	Type A
4	调试接口	Micro USB
5	音频接口	1.25mm-1×4Pin
6	MIC	1.25mm-1×2Pin
7	WIG	1.25mm-1×4Pin
8	ALM多路GPIO接口	1.25mm-1×8Pin
9	网络接口	RJ45
10	串口	1.25mm-1×5Pin
11	12V电源接口	2.0mm-1×2Pin
12	CARD	1.25mm-1×5Pin
13	USB	1.25mm-1×4Pin
14	红外补光、光感接口	1.5mm-1×4Pin
15	MIPI	0.5mm-FPC_40Pin
16	白光补光接口	1.5mm-1×2Pin
17	复位	-
18	多功能接口	凤凰端子2.54mm_16P
19	网口	RJ45
20	12V电源	-
21	SIM	-
22	TF接口	-
23	触摸屏接口	0.5mm-FPC_6Pin
24	扩展接口	0.5mm-FPC_10Pin
25	7英寸屏接口	0.5mm-FPC_30Pin
26	5、8英寸屏接口	0.5mm-FPC_31Pin

音频接口（5）

引脚	信号定义
1	SPK1+
2	SPK1-
3	SPK2+
4	SPK2-

MIC接口（6）

引脚	信号定义
1	MIC
2	GND

WIG接口（7）

引脚	信号定义
1	WIG_D0
2	WIG_D1
3	GND
4	12V

ALM多路GPIO接口（8）

引脚	信号定义
1	GPIO0_4_SYS_RESET
2	SW_SYS_PWR_EN
3	ALM_IN1
4	GND
5	ALM_OUT1_A
6	ALM_OUT1_B
7	ALM_OUT2_A
8	ALM_OUT2_B

串口接口（10）

引脚	信号定义
1	RS485+
2	RS485-
3	GND
4	RS232_TXD
5	RS232_RXD

12V电源接口（11）

引脚	信号定义
1	12V
2	GND

card接口（12）

引脚	信号定义
1	GND
2	GPIO0_1/I2C3_SDA
3	GPIO0_2/I2C3_SCL
4	ID_INT下拉
5	5V

USB接口（13）

引脚	信号定义
1	GND
2	DP
3	DM
4	5V

红外补光、光感接口（14）

引脚	信号定义
1	IR_LED+
2	IR_LED-
3	CDS+
4	GND

MIPI接口（15）

引脚	信号定义	引脚	信号定义
1	Vin_12V	2	GND
3	GPIO8_0_ICR_CTL1	4	GPIO8_1_ICR_CTL2
5	UART4_RXD	6	UART4_TXD
7	GND	8	MIPI_RX0_CK0P/VI_DATA12
9	MIPI_RX0_CK0N/VI_DATA13	10	GND
11	MIPI_RX0_D0P	12	MIPI_RX0_D0N
13	GND	14	MIPI_RX0_D1P
15	MIPI_RX0_D1N	16	GND
17	NLMIPI0RX00D2P0VI0DATA14	18	NLMIPI0RX00D2N0VI0DATA15
19	GND	20	MIPI_RX0_D3P/VI_DATA11
21	MIPI_RX0_D3N/VI_DATA10	22	GND
23	MIPI_RX0_D3N/VI_DATA9	24	MIPI_RX0_D3N/VI_DATA8
25	GND	26	SENSOR0_CLK
27	VI_HS/SENSOR_HS/SENSOR1_RSTN	28	VI_VS/SENSOR_VS/SENSOR1_CLK
29	GND	30	SENSOR0_RSTN/BOOT_SEL1
31	SPI0_SCLK/I2C0_SCL	32	SPI0_SDO/I2C0_SDA
33	SPI0_SDI/I2C1_SDA	34	NLSPI00CSN0I2C10SCL
35	GND	36	DC_IRIS_PWM4
37	NC	38	GND
39	3V3	40	5V

白光补光接口（16）

引脚	信号定义
1	W_LED+
2	W_LED-

多功能接口（18）

引脚	信号定义	引脚	信号定义
1	WGI_D0	2	WGI_D1
3	GND	4	NC
5	NC	6	ALM_IN
7	ALM_OUT1_A	8	ALM_OUT1_B
9	ALM_OUT2_A	10	ALM_OUT2_B
11	RS485+	12	RS485-
13	RS232_TXD	14	RS232_RXD
15	GND	16	12V

触摸屏接口（23）

引脚	信号定义
1	GPIO3_1/TP_INT
2	GPIO10_7/I2C7_SDA
3	GPIO10_6/I2C7_SCL
4	GPIO0_3/TP_RST
5	GND
6	3V3

扩展接口（24）

引脚	信号定义	引脚	信号定义
1	5V	2	GND
3	KEY1	4	KEY2
5	KEY3	6	KEY4
7	KEY5	8	KEY6
9	KEY7	10	KEY8

7寸屏接口（25）

引脚	信号定义	引脚	信号定义
1	LED+	2	LED+
3	VGH	4	VGL
5	UPDN	6	SHLR
7	LED-	8	LED-
9	AVDD	10	GND
11	DSI_D3P	12	DSI_D3N
13	GND	14	DSI_D2P
15	DSI_D2N	16	GND
17	DSI_CKP	18	DSI_CKN
19	GND	20	DSI_D1P
21	DSI_D1N	22	GND
23	DSI_D0P	24	DSI_D0N
25	GND	26	STBYB
27	LCD_RESET	28	LCD_1V8
29	LCD_1V8	30	LCD_VCOM

5、8寸屏接口（26）

引脚	信号定义	引脚	信号定义
1	LED+	2	LED+
3	LED+	4	NC
5	LED-	6	LED-
7	LED-	8	LED-
9	GND	10	GND
11	DSI_D2P	12	DSI_D2N
13	GND	14	DSI_D1P
15	DSI_D1N	16	GND
17	DSI_CKP	18	DSI_CKN
19	GND	20	DSI_D0P
21	DSI_D0N	22	GND
23	DSI_D3P	24	DSI_D3N
25	GND	26	LCD_1V8
27	LCD_RESET	28	NC
29	LCD_1V8	30	LCD_3V3
31	LCD_3V3		

资料下载

您好：

度目已为您提供关于人脸应用套件H1系列相关文档及使用工具包，请您下载使用哦

度目人脸应用套件H1系列HTTP接口文档.pdf

度目人脸应用套件H1系列补充文档.pdf

度目人脸应用套件H1系列升级工具包.zip

备注：

度目为您提供的度目人脸应用套件H1系列升级工具包.zip 内包含人脸应用套件H1系列产品升级工具保障您使用度目最新程序，祝您使用愉快！

公有云采集SDK 4.1.5

概览

快速导览

如果您的需求是与人脸比对、人脸实名认证、在线图片活体检测API搭配使用，请创建APP方案获取**最新版本SDK及示例工程**。

- 人脸实名认证方案-iOS版本
- 人脸实名认证方案-Android

核心概念

人脸识别的应用场景，核心可以分为三大类：

1. **权威数据源身份核验**：即人脸实名认证，基于姓名、身份证号码、现场采集的人脸图片，判断三个要素信息的一致性，判断`你是你`，通常用于需要验证用户身份真实性的场景，如人脸实名认证。
2. **身份识别**：即1:1识别，判断两张脸的相似度，判断`你是你`，通常用于需要验证用户身份真实性的场景。
3. **活体检测**：即在线图片活体，基于单张图片，判断图片中的人脸是否为二次翻拍。

而以上场景的几乎所有业务过程，核心可以分为两个步骤：

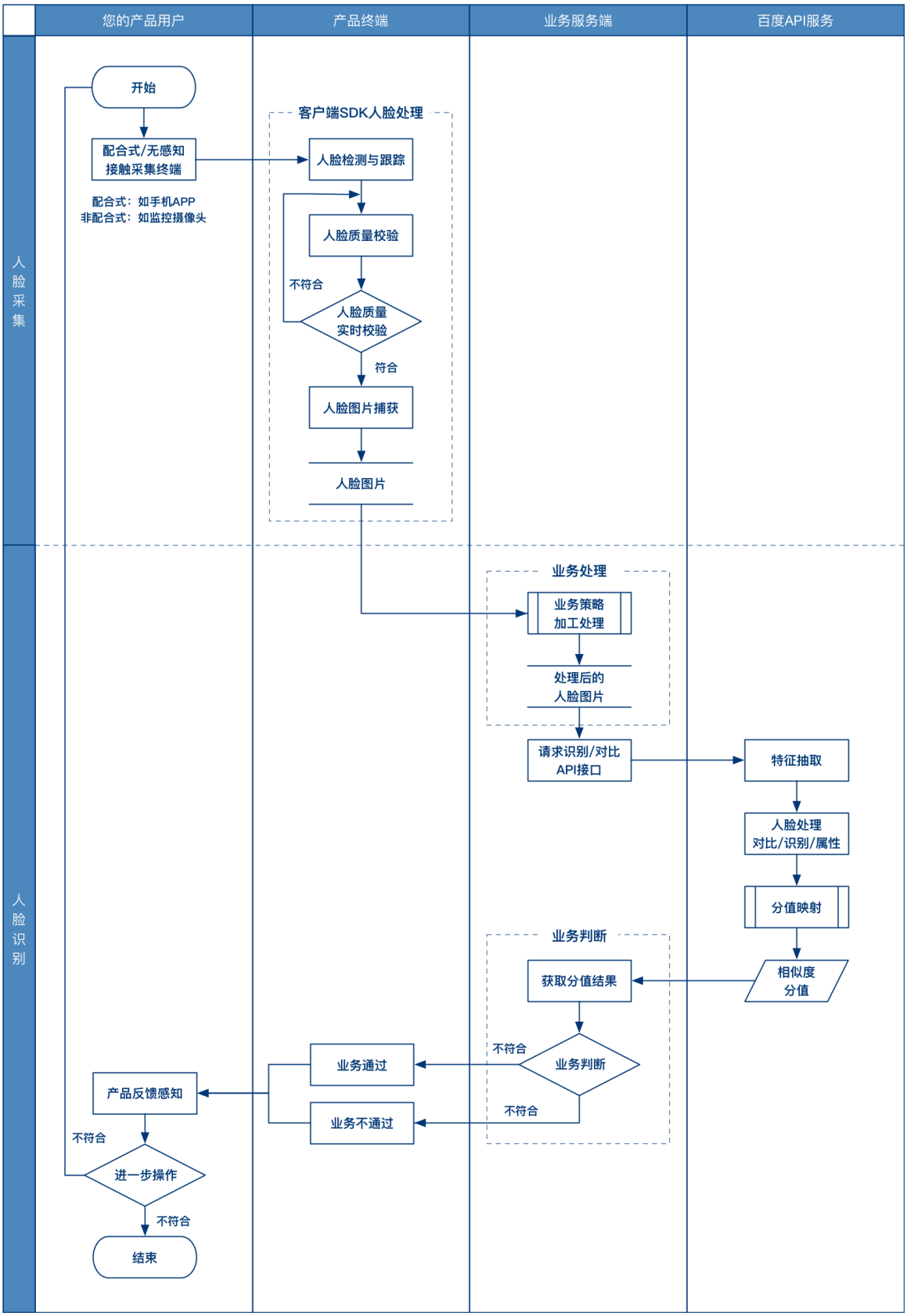
1. **人脸采集**：人脸识别的前置步骤，即获取到人脸图片，用于对比、识别、属性分析等操作。
2. **人脸分析**：包括人脸图片的加工处理，特征抽取与对比，结果返回等一系列操作，也是通常理解为的人脸识别操作。

要想确保人脸识别的应用效果得到保障，最为核心的一个环节即人脸的获取，即**人脸采集**。目前市面上所有人脸识别应用落地，面临的主要问题就是**应用环境复杂**，包括光照、遮挡、作弊攻击等一系列环境因素干扰，涉及产品策略、硬件选型、施工方案等多个维度地综合作用，才能不断提升最终效果。

为了更好地帮助您完成人脸采集步骤，下文将详细介绍采集过程中的一些SDK、API、硬件相关内容，希望对您的人脸识别集成有所帮助。

> **P.S.**
>
> 采集SDK本身仅对动作等行为进行活体校验，验证用户所做动作是否符合要求。针对手机翻拍、PS照片，头模攻击等行为，需配合云端活体检测算法模型共同使用，才可达到最优拦截效果。百度公有云的接口人脸比对、在线图片活体检测、人脸实名认证等都包含云端活体检测模型，请在云端接口调用时，选择liveness_control的阈值（建议值为NORMAL）进行控制。

流程概览



目录导览

目录导览如下，文档内容较长，但前后具备相关性，建议顺序阅读。

```
准备工作
- 账号及应用
- SDK
- 采集设备
- API接口
采集指标
- 质量控制
  - 概述
  - 遮挡
  - 模糊度
  - 光照
  - 姿态
  - 其他
- 活体检测
  - 概述
  - 有动作活体检测
  - 静默图片活体检测
  - 视频活体检测
SDK采集
- 概述
- 功能介绍
- 支持平台
- 规格参数
- 场景版本
- 离线质量检测
- 离线有动作活体检测
- 开放参数配置
- 应用方案建议
采集设备
- 概述
- 手机
- 近距离人脸采集开发组件
- 中远距离人脸采集开发组件
- 远距离人脸采集开发组件
```

获取帮助

如果在阅读文档后，对内容有任何疑问，可以通过以下几种方式联系我们：

- 在百度云控制台内 提交工单 ，咨询问题类型请选择人工智能服务。
- 如有问题讨论，可以进入 AI社区 随时提问，和其他开发者一同交流。

准备工作

磨刀不误砍柴工，在正式讲解人脸采集细节前，让我们先梳理下必要的前置准备工作。

1、账号及应用

在进入实际开发前，请先阅读 新手指南，指南中详细地讲解了开放平台账号及应用的创建及管理操作。

新手指南中主要讲解三件事情：

1. **准备账号**：可以为百度账户或者推广账户，如果首次使用，需要先做以下开发者认证（验证下手机号，很快的）。这是您使用API接口、SDK管理、付费等操作的主体。
2. **创建应用**：在人脸识别后台的**应用列表**中操作，创建一个应用，会对应生成`AppID`、`API-Key`、`Secret-Key`，这是应用的唯一标识，是您调用API接口的基本操作单元（主要用于权限配置、应用高级配置等）。
3. **生成签名**：也称为`Access_token`，是调用API时的鉴权凭证，具体可参考 Token获取方法。

以上三项事宜是使用公有云服务的必备步骤，也是后续很高频使用的内容。

2、SDK

客户端SDK

客户端SDK将**人脸采集**这个步骤放到前端解决。内部封装了离线人脸检测、跟踪、质量校验、图片捕捉功能，但权威数据源核验、1：1、1：N识别还需调用在线API接口。

目前对外开放`iOS`、`Android`两个版本，客户端SDK需要 创建APP方案，流程分为以下几步：

1. 完成企业认证（营业执照信息认证）
2. 新建项目
3. 方案管理-新建APP方案
4. 配置方案，下载示例工程

完成后，您即可下载SDK及官方提供的示例工程源码。

SDK默认配备4个产品线授权数量（产品线指：基于应用维度，如一款APP即为一条产品线，如手机百度 iOS/Android APP，依据为Bundle ID或者Package name），4个授权基本可以满足大部分的产品业务需求，如果需要更多的产品线授权，请申请增加授权数量。

SDK的使用，**涉及有效期**，有效期过后，SDK本地功能将会不可用，SDK会自动向百度服务器请求拉取最新的授权license，如果对这个产品线的授权在服务端操作了延期，则SDK会在过期后的第一次初始化时，自动更新本地license。

P.S.

搭配人脸识别公有云API使用时，采集SDK为免费产品，在上线前请将产品授权替换为正式授权再发布上线，否则，C端用户在使用过程中，若直接上测试授权，授予过期后，需要在有网的条件下拉取线上的鉴权进行更新，会出现客户APP的网络出现异常的情况导致服务不可用。

3、采集设备

实际使用中，方案的最终使用，都需要配合前端采集的硬件设备，目前对于常规人脸需求，几种选型方案如下：

- **手机**：APP场景下的绝大部分使用场景，摄像头、屏幕、数据处理一体化，除了APP产品，也可以用于快速构建demo，验证业务流程的首选。

P.S. 人脸采集SDK为移动端人脸采集场景使用，支持iOS&Android手机系统，若您的场景为开发板以及一体机场景，建议申请人脸离线识别SDK，采集SDK不保证在非手机设备的适配性，不支持横屏，不涵盖后置摄像头场景。

以上仅为简单列举，后文会对硬件选型进行详细介绍。

4、API接口

人脸采集后，仅是在前端设备上拿到人脸图片，实际的权威数据源人脸实名认证、1：1人脸识别、1：N人脸搜索识别仍要通过API调用实现。

目前平台提供的接口分为以下几种：

- **在线图片活体**：基于图片中的破绽分析，判断其中的人脸是否为二次翻拍（举例：如用户A用手机拍摄了一张包含人脸的图片一，用户B翻拍了图片一得到了图片二，并用图片二伪造成用户A去进行识别操作，这种情况普遍发生在金融开户、实名认证等环节）。；查看详细介绍>>
- **人脸对比**：比对两张图片中人脸的相似度，并返回相似度分值；支持生活照、证件照、带水印照等多种人脸图片类型；查看详细介绍>>
- **人脸实名认证**：可通过姓名+身份证号码+人脸图片请求权威数据源，判断用户是否为本人且为真人；查看详细介绍>>

采集指标

质量控制

写在最前，采集SDK在本地会做质量校验，输出图片后，调用云端接口时，建议将云端接口的质量控制参数调整为NORMAL，不建议使用HIGH，避免采集SDK通过后，由于云端质量严格导致用户访问失败，重新录制，而降低用户体验。

概述

人脸识别或对比的最终效果，取决于人脸在采集过程中，采集到的人脸**是否符合标准质量要求**。从业务使用角度，主要影响两个核心业务步骤：

1. **人脸注册环节**：如果注册的人脸质量不佳，则会影响注册环节的特征抽取，导致原始注册的人脸信息较差，后面的识别/对比都会受到直接的影响，往往得到的相似度分值，将不会特别准确。
2. **人脸识别/对比环节**：因为注册人脸质量不佳，每次的识别/对比都会存在一定的分值误差，往往造成明明是本人却过不去的情况。

人脸的质量检测，概括起来包括以下几点：

- **遮挡**：指人脸各部位的遮挡比例；
- **模糊度**：指人脸的清晰程度；
- **光照**：指人脸的光照强度；
- **完整性**：指图片中的人脸是否完整；

另外在实际采集过程中，也会将姿态作为一个重要控制指标：

- **姿态**：指人脸在三维空间的角度分布；

人脸采集步骤，需要做好以上5项的条件判断，从而确保最终识别效果。

下面让我们一起详细看看这5个质量检测项：

遮挡

Occlusion：

人脸中各个部位的遮挡程度判断，区域可以分为：`左眼`、`右眼`、`鼻子`、`左脸颊`、`右脸颊`、`嘴巴`、`下巴`，共7个区域。通常某一个或者多个区域遮挡面积过大，会影响最终的识别效果，可以通过区域的遮挡值，在产品侧给用户比较明确的产品反馈提示，供用户参考调节。

每个区域的建议阈值选择如下：

```
 取值范围：[0~1]
0：无遮挡
1：完全遮挡

left_eye : 0.6, //左眼被遮挡的阈值
right_eye : 0.6, //右眼被遮挡的阈值
nose : 0.7, //鼻子被遮挡的阈值
mouth : 0.7, //嘴巴被遮挡的阈值
left_check : 0.8, //左脸颊被遮挡的阈值
right_check : 0.8, //右脸颊被遮挡的阈值
chin_contour : 0.6, //下巴被遮挡阈值
```

> 说明：**左右位置的表示**，是以当前检测图片的方向确定。如实际场景中为人物的左脸颊，但是在分析时，则认为是右脸

> 颊。

下面展示三个示例图片，用于理解**遮挡**的概念：

【示例一】

分析：脸颊有一定头发遮挡，但都在阈值范围内，整体问题不大。**不过为了更好的识别效果**，如果可以向用户反馈提示文案的话，可提示用户重新整理右脸颊侧的头发，并重新录入人脸。而文案触发机制可以设定一个更严格的产品自定义阈值。

```
 "occlusion": {
"left_eye": 0,
"right_eye": 0.12301587313414,
"nose": 0.016585364937782,
"mouth": 0,
"left_cheek": 0.013826940208673,
"right_cheek": 0.66922038793564, //右脸颊遮挡较大
"chin": 0.13753361999989
}
```

【示例二】

分析：脸部各区域都些许遮挡，都未到达阈值，大小相对适中，不影响正常使用。**对于识别精准度比较严格的场景，倒也可以**

提示重新录入。


```
 "occlusion": {
"left_eye": 0.055045872926712,
"right_eye": 0,
"nose": 0.1026463508606,
"mouth": 0.33571428060532,
"left_cheek": 0.18918919563293,
"right_cheek": 0.38289964199066,
"chin": 0.38516879081726
}
```


【示例四】

分析：右眼明显遮挡严重，随是艺术照示意，但是在实际应用场景中，主要由于发型原因，**存在很多眼镜部分被头发遮挡较严重的情况**，产品应用时建议提示用户整理下头发再重试。


```
 "occlusion": {
"left_eye": 0.032388664782047,
"right_eye": 0.90909093618393,
"nose": 0.030821917578578,
"mouth": 0,
"left_cheek": 0.22132650017738,
"right_cheek": 0.23717948794365,
"chin": 0.0049975011497736
}
```


【示例四】

分析：因为戴墨镜，导致左右眼遮挡很大，这种情况可直接提示用户摘下眼镜，并重新录入一张。同理，对于**口罩**、**长发**等也可以如此操作。

```
 "occlusion": {
"left_eye": 0.71283352375031,
"right_eye": 0.93929713964462,
"nose": 0.22417153418064,
"mouth": 0.37142857909203,
"left_cheek": 0.089686095714569,
"right_cheek": 0.64788734912872,
"chin": 0.66417443752289
}
```

模糊度

Blur：

人脸的模糊程度，取值范围：0~1，0是最清晰，1是最模糊，通常 **小于0.7** 即可认为是符合条件。

下面展示三个示例图片，用于理解**模糊度**的概念：

【示例一：原清晰图】

```
 "blur": 6.6545192967116e-11； //科学计数法，实际数值非常小
```

分析：模糊度极接近0，十分清晰，可放心使用。

【示例二：人脸些许模糊】

```
"blur": 0.00000165249139172 37;
```

分析：模糊度接近于0，远小于0.7（建议的阈值），可放心使用。

【示例三：人脸十分模糊】

分析：模糊度接近于1，不可使用。这种情况通常为摄像头未对好焦距，如手机没有拿稳，或摄像头焦点主体错误，通常可提示用户保持身体稳定不动，正视镜头即可。

```
"blur": 0.99997198581696；
```

光照

illumination：

人脸部分光照的灰度值，反映脸部的光照情况，以及客户端SDK中，`YUV的Y分量`；**取值范围[0~255]，0表示光照不好**。人脸过暗对于识别会有显著影响，所以通常在所有质量校验中，要优先保持人脸的光照充足。

人脸采集过程中，常遇到的光线问题，一般分为以下几种:

- 对比度过低
- 过曝光
- 欠曝光
- 过亮
- 过暗
- 阴阳脸

以上问题，主要原因主要分为以下几种：

- **镜头对焦**：因为镜头对焦问题，导致曝光不正常，往往人脸的光线过暗或过亮。
- **光源不足**：往往导致脸部光线较暗。
- **光源角度**：导致阴阳脸等光线分布严重不均匀的情况。

所有问题中，**逆光**往往是最严重的问题，大部分的人脸采集设备是在白天作业，摄像头在逆光情况下，采集到的脸部光照会很暗，导致识别不到人脸或者识别效果差。这种情况，通常可以有以下几种应对措施：

1. **补光灯**：在摄像头旁边添加一个补光灯，用于提高脸部光线强度，此方案也可以同时保证夜间门禁等场景的光线问题；
2. **调整设备角度**：在满足识别体验的角度情况下，避免镜头对着逆光位置（如阳光、室内大灯等）；
3. **添置遮罩物件**：

从接口返回值来看，推荐阈值如下：

```
"illumination"：40 //大于40通常认为光照满足，推荐40~100之间
```

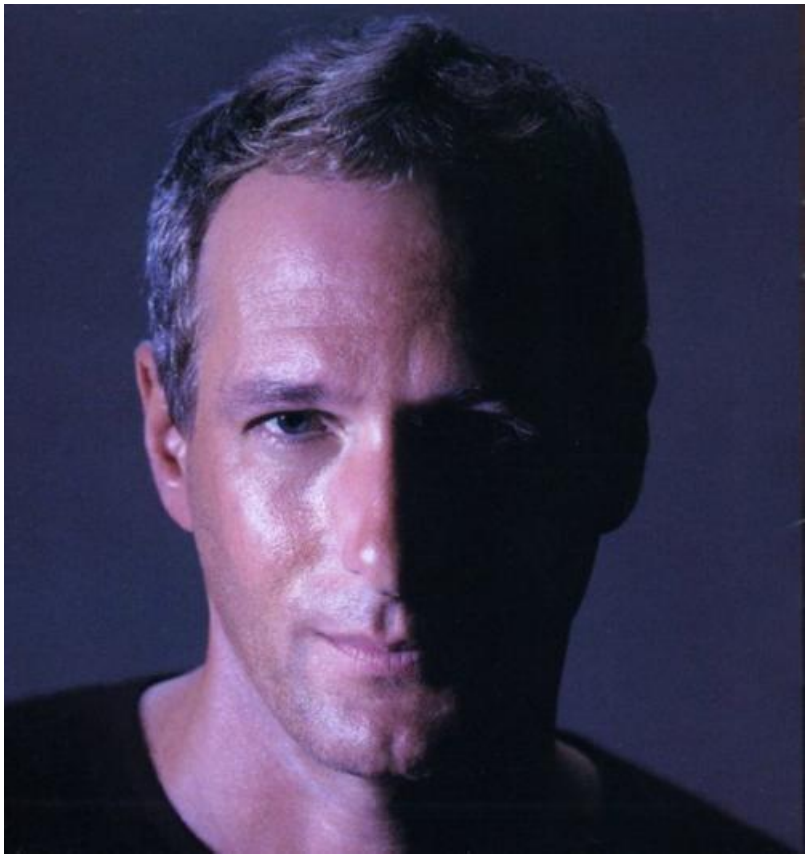
下面展示一个例子，用于理解光照的概念：

【示例一】

分析：过暗的情况，人脸的识别往往存在误差，甚至识别不到，这种情况需要在采集设备周围增加补光设备。

```
"illumination": 20；
```

【示例二】



分析：阴阳脸。往往是光源位置角度问题，导致某一边人脸光线明显不足，需要确保实际应用过程中，光线分布均匀。

```
"illumination": ;
```

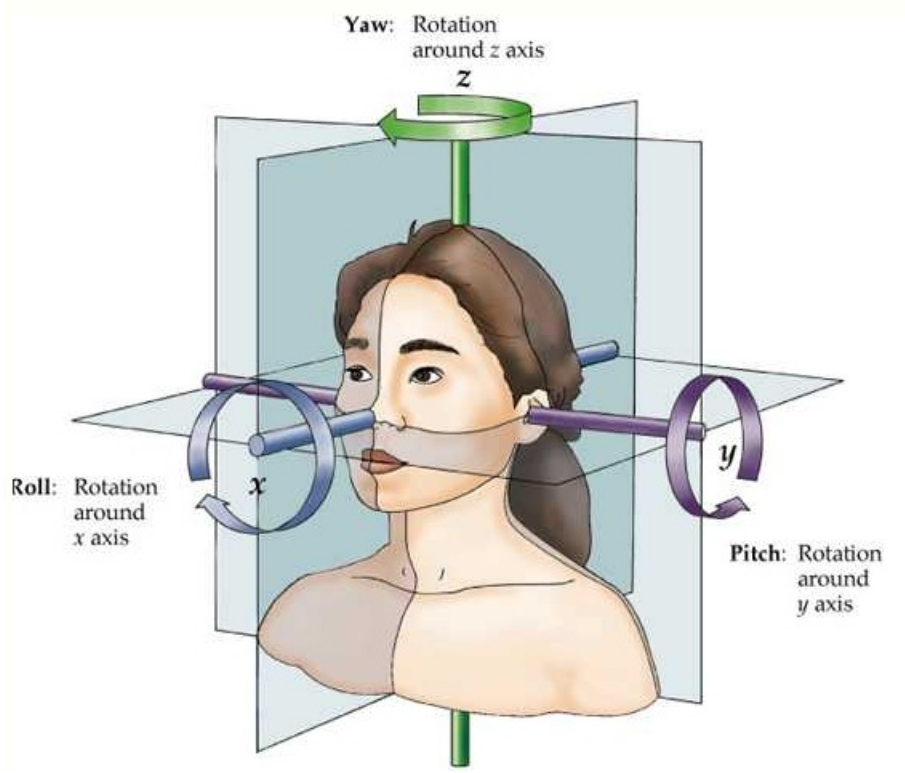
姿态

姿态角分为Pitch、Roll、Yaw，用于表示人脸在空间三维坐标系内的角度，常用于判断识别角度的界限值。

各角度阈值如下：

```
Pitch：三维旋转之俯仰角度，范围：[-90（上），90（下）]，推荐俯仰角绝对值不大于20度；
Roll：平面内旋转角，范围：[-180（逆时针），180（顺时针）]，推荐旋转角绝对值不大于20度；
Yaw：三维旋转之左右旋转角，范围：[-90（左），90（右）]，推荐旋转角绝对值不大于20度；
```

各角度范围示意图如下：



从姿态角度来看，**这三个值的绝对值越小越好**，这样代表人脸足够正视前方，最利于实际注册/识别使用。

但在实际应用场景中，由于摄像头的布设位置，往往无法拿到正对人脸的图片，主要分为以下几种情况：

1. **监控摄像头**：此类摄像头一般置于室内棚顶/室外架杆顶端，垂直向下倾斜一定角度，水平方向有一定的拍摄广度，一般Pitch和Yaw角度变化范围较大，所以采集到的人脸往往存在大量的俯角过大、侧脸等，导致识别效果不佳。这种情况通常是调整摄像头角度（摄像头与水平面夹角减小）、调整最小检测人脸（在行人靠近摄像头时尽可能早些拿到人脸）、增加摄像头数量（不同角度互相补充，避免采集死角），实际项目实施中，还是要通过实地考察，基于现场环境一点点调节摄像头角度；

2. **闸机/门禁**：闸机方面，摄像头往往置于机器顶部，摄像头向上仰一定角度，通常低于平均成人身高，所以行人路过时，一般需要微微低头看闸机上的摄像头/屏幕，如果摆放不当，会造成采集到的人脸Pitch角度过大；门禁方面，摄像头往往置于成人身高平视高度，但多为门框侧面，如果行人朝向正门，侧面对视摄像头，如摆放不当，会造成采集的人脸Yaw角度过大。以上两种主要场景，需要基于实际场景情况动态调整安装位置，尽量避免角度过大的同时，避免用户动作不要太大。

3. **USB摄像头**：线下场景中，USB摄像头常用于近场场景录入人脸（如柜台、大屏、自助柜机等），这种情况下为了拿到角度最好的图片（用户正视屏幕），在应用UI方面做一个前置页面提示，往往能最小成本地提高操作标准度。

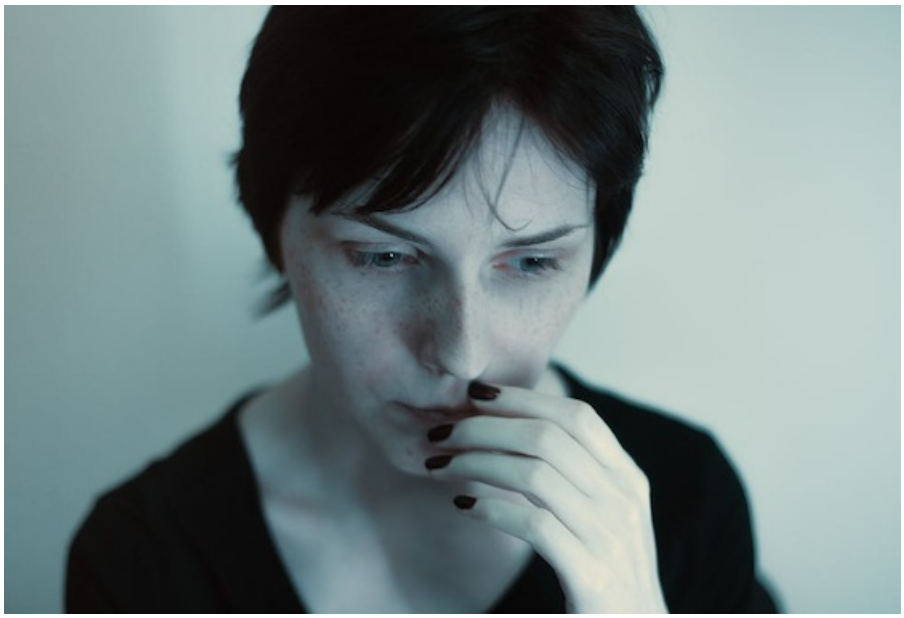
下面展示两个示例图片，用于理解**姿态角**的概念：

【示例一】

分析：通常线下应用过程中，难以获得相对正面的人脸图像，姿态判断需要作为一个标准的校验项。

```
  "yaw": 16.319696426392,
"pitch": -4.8351268768311,
"roll": 2.0234982967377,
```

【示例二】



分析：如果是将摄像头置于棚顶或室外高处，往往获取到的人脸图片，`pitch`值比较大，需要特别关注。

```
  "yaw": -10.153998374939,
"pitch": 19.434478759766,
"roll": 1.6366827487946,
```

其他

完整性

可以使用`v3/detect`接口中的`qualities->completeness`参数判断。此参数用于判断人脸完整度，返回只有0或1两种结果，0为人脸溢出图像边界，1为人脸都在图像边界内。通过此参数可以简单判断人脸是否全部置于画面内，做一定的条件校验。

真人/卡通

可以使用`v3/detect`接口中的`qualities->type`参数判断，此参数分为`human`和`cartoon`两个值，取值范围为[0~1]，置信度大于0.5即可判断为真人或者卡通。通过此参数可以简单判断画面中是否存在真实人脸，避免一些动画、素描、油画等人像的仿冒。

活体检测

概述

根据应用场景及安全性要求的不同，目前提供不同的活体检测算法和策略，具体包括：

- **实时炫瞳活体检测**：通过屏幕上闪烁不同颜色的光线，判断当前用户是否真人操作。通过颜色活体进行面部反光鉴别的同时，百度特加入独有的瞳孔反光识别，提升整体的攻击拒绝率指标，此方式往往结合人脸实名认证、人脸比对、在线图片活体一起使用。
- **实时动作活体检测**：通过要求用户配合做出一些面部动作，验证是否为身份伪造攻击，此方式往往结合人脸实名认证、人脸比对、在线图片活体一起使用。
- **静默图片活体检测**：通过实时摄像头采集或提交图片，检测图片中的屏幕边框、反光、摩尔纹、成像畸变等线索，来区分是否为**二次翻拍攻击**（举例：如用户A用手机拍摄了一张包含人脸的图片一，用户B翻拍了图片一得到了图片二，并用图片二伪造成用户A去进行识别操作，这种情况普遍发生在金融开户、实名认证等环节），此方式往往结合人脸实名认证、人脸比对、在线图片活体一起使用。
- **视频活体检测**：通过上传一段视频，进行视频随机抽帧分析，对随机图片进行静默图片活体检测，得出综合攻击情况分析结果。为确保视频唯一性，可以结合随机校验码接口使用，实现生成随机数字或随机动作，用户通过配合完成指令进行活体检测。

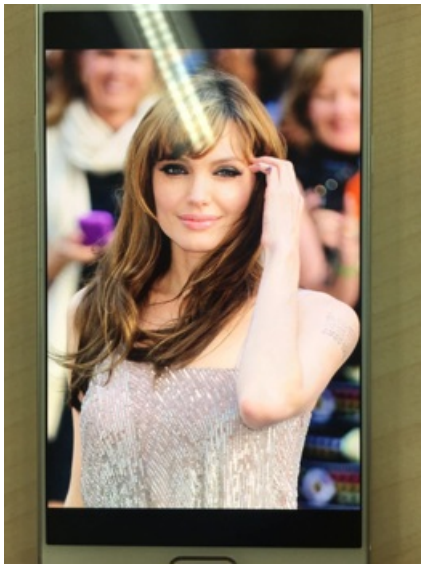
实际采集过程中，为了确保业务流程的合理性，需要确保采集的图片为**真人操作**，避免其他用户恶意伪造作弊。而以上几种活体检测策略，需要根据业务场景选择最合适的一个或组合方式。下面简单举例一些常见的活体攻击类型：

打印照片

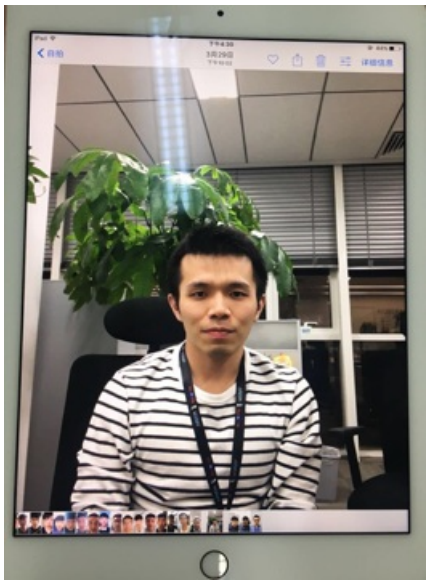
照片抠图

照片变形

手机照片翻拍



Pad/PC照片、视频



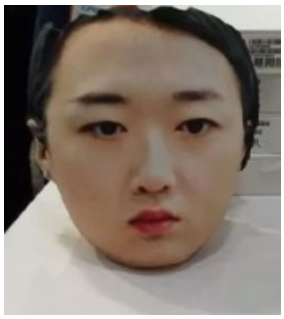
照片背景替换

换脸

2.5D动画

3D动画

高仿面具



炫瞳活体检测

客户端采集SDK在前端离线随机通过屏幕上闪烁不同颜色的光线，判断当前用户是否真人操作。通过颜色活体进行面部反光鉴别的同时，百度特加入独有的瞳孔反光识别，提升整体的攻击拒绝率指标。需要强调的是，在此策略下，前端仅仅检测面部反光的情况，**并不返回活体分数，或对视频和真人做出区分**。在炫瞳活体检测的过程中，SDK会随机抓取图像，并在动作通过后将抓取的图像上传到后端服务器进行活体判断。

动作活体检测

客户端采集SDK在前端离线随机要求用户配合做出眨眼、张嘴、点头、抬头、左摇头、右摇头动作，同时检测用户完成情况。需要强调的是，在此策略下，前端仅仅检测动作的通过情况，**并不返回活体分数，或对视频和真人做出区分**。在做动作的过程中，SDK会随机抓取图像，并在动作通过后将抓取的图像上传到后端服务器进行活体判断。

该方案的优势在于：随机动作可以增加攻击的成本，并且大大降低打印照片等攻击通过的概率，而这些正好对于后端检测算法来说属于较难案例。

> 有动作活体检测，不推荐单独使用，需要结合静默图片活体检测一起使用。对于APP场景，一般推荐都可以使用客户端SDK，增加有动作活体检测步骤，提升人脸验证过程的安全性。

静默活体检测

该方案为接入成本最低的使用方式，只需提交一张人脸图片即可。但是活体检测的效果不是十分理想，主要是由于：区别于客户端随机抓拍的方式，H5拍照为主动有意识拍照，因此攻击者可以更加方便的设法减少出现后端算法判断的线索，并且在这个过程中用户可以调用后置摄像头进行拍照，前置摄像头和后置摄像头由于焦距的不同，会使后置摄像头拍摄的攻击图片更难分辨真假。另外，H5拍照只上传一张图片，增加了误判的可能性。

活体阈值

阈值（Threshold）	误拒率（FRR）	通过率（TAR）	攻击拒绝率（TRR））
0.05	0.01%	99.99%	97.75%
0.1	0.05%	99.95%	98.33%
0.3 （推荐）	0.1%	99.9%	98.82%
0.5	0.5%	99.5%	99.67%
0.9	1%	99%	99.77%

关于以上数值的概念介绍**：

- **拒绝率（TRR）**：如99%，代表100次作弊假体攻击，会有99次被拒绝。
- **误拒率（FRR）**：如0.5%，指1000次真人请求，会有5次因为活体分数低于阈值被错误拒绝。
- **通过率（TAR）**：如99%，指100次真人请求，会有99次因为活体分数高于阈值而通过。
- **阈值（Threshold）**：高于此数值，则可判断为活体。

SDK采集

概述

客户端SDK，具备SDK本地的众多人脸采集相关功能，可以更高效准确地，实时采集到符合质量的人脸，配合摄像头等设备，能够快速构建**实时人脸采集能力**。

客户端SDK申请地址

功能介绍

人脸检测

设备端离线实时监测视频流中的人脸，同时支持处理静态图片或者视频流。

人脸跟踪

对当前检测到的人脸持续跟踪，动态定位人脸轮廓，稳定贴合人脸。

人脸关键点

对当前检测到的人脸持续跟踪，并动态实时展现人脸上的核心关键点。

人脸图片采集

在人脸检测及追踪过程中，完成人脸图片采集，并输出预设条件的人脸图片

人脸质量控制

在人脸检测及追踪过程中，实时校验人脸的姿态角度、遮挡、清晰度、光照条件，符合质量条件的才会被采集。

炫瞳活体检测

通过屏幕上闪烁不同颜色的光线，判断当前用户是否真人操作。通过颜色活体进行面部反光鉴别的同时，百度特加入独有的瞳孔反光识别，提升整体的攻击拒绝率指标。

有动作活体检测

实时反馈，人脸中眼睛，嘴巴，头部姿态的状态，通过给用户设定完成相关动作，判断是否是活体。支持指定生效的动作及应用顺序。

开放参数设置

SDK内部支持高度可定制化参数，对人脸检测、追踪、采集、质量模块进行个性化调整。

UI自定义修改

SDK内部所有UI层代码、音频文件全部开源，可根据实际业务需求任意调整。

多种场景版本

提供iOS、Android端，适应各种应用场景及设备类型需求。

支持平台

- iOS版本：iOS 9+
- 安卓版本：Android 5.1.1+

规格参数

- SDK大小：~ 5M
- 最小人脸检测大小：30px * 30px（实际应用推荐80px~200px之间）
- 可识别人脸角度：yaw ≤ ±30°, pitch ≤ ±30°
- 检测速度：100ms 1080p*
- 追踪速度：10ms 1080p*

场景版本

- 支持纯人脸采集及做动作的采集场景。建议开启动作校验，可以提高人脸采集的安全程度。

离线质量检测

客户端SDK中，内置了人脸质量检测模块。在人脸检测及追踪过程中，实时校验人脸的姿态角度、遮挡、清晰度、光照条件，符合质量条件的才会被采集，此质量校验为本地离线校验。

离线炫瞳活体检测

通过屏幕上闪烁不同颜色的光线，判断当前用户是否真人操作。通过颜色活体进行面部反光鉴别的同时，百度特加入独有的瞳孔反光识别，提升整体的攻击拒绝率指标。

离线有动作活体检测

客户端SDK，离线本地实时反馈，人脸中眼睛，嘴巴，头部姿态的状态，通过给用户设定完成相关动作，判断是否是活体。支持指定生效的动作及应用顺序。

开放参数配置

最小检测人脸

在检测人脸过程中，可设置检测到的最小人脸尺寸，小于该尺寸或比例的人脸不会被检测到。

采集姿态角控制

在检测人脸过程中，可设置采集图片时的人脸姿态角度阈值，阈值范围内才会采集人脸。

采集人脸质量控制

在检测人脸过程中，可设置人脸关于光照、清晰度、各部位遮挡的阈值，符合条件才会采集图片。

采集图片设置

可设置采集人脸图片的数量、大小、以及人脸与图片的大小比例，超过图片边界将用黑色填充。

活体检测动作设置时

支持眨眼、张嘴、摇摇头、左右转头、上下点头六个指定动作，可设定具体生效的动作，以及校验顺序。

应用方案建议

数据传输

SDK本地采集人脸的过程，完全无需联网。但人脸对比、人脸搜索、人脸属性分析能力需要调用API使用。产品策略方面，因API使用需要使用在线鉴权token，生成token的API Key和Secret Key，考虑到数据安全和维护成本，建议都置于Server端，并下发token到客户端产品，实际的API调用，由Server端做中转控制。

图片压缩

SDK支持采集图片的大小设置，以及人脸和采集图片的比例设置，可基于业务需要，对采集图片大小进行适当地进一步加工。**如剪裁（人脸不小于100px）、分辨率压缩（最小宽度200px左右）、质量压缩（控制在0.8以上），以上三种处理方式也可以组合叠加使用。**

数据安全

对安全有进一步需求的话，为防止人脸传输过程中被篡改，可对SDK本地输出的人脸图像做加密处理，在server端进行相应解密操作，进一步增强安全性。

话术提示

手机把握姿态：

采集设备

概述

可基于Phone形式，快速搭建人脸采集前端设备。摄像头实时输入视频流，对视频流中的人脸实时处理，并输出符合条件的人脸图片。

手机

应用场景

对于远程开户、刷脸登录、移动考勤等手机端场景，可将iOS/Android SDK集成到现有APP应用中，在设备端离线完成人脸采集，确保采集图片质量的同时，提供更加流畅的交互体验。也可以基于微信小程序，或者H5形式，完成业务快速落地。

适用采集人数

1人。

产品架构

手机+客户端SDK

产品形态

APP

核心性能需求

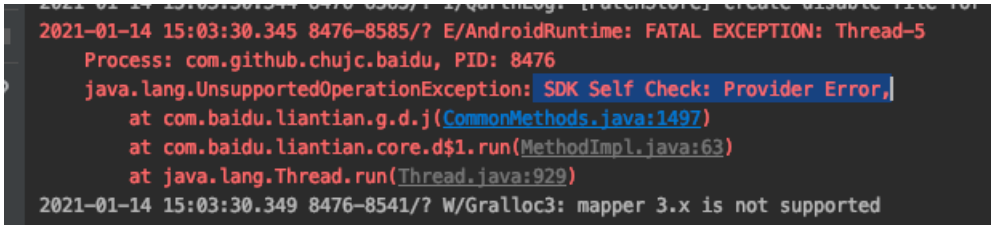
暂无要求。需要注意的是需要根据业务特点，做好系统版本或者浏览器版本的兼容工作。

常见问题

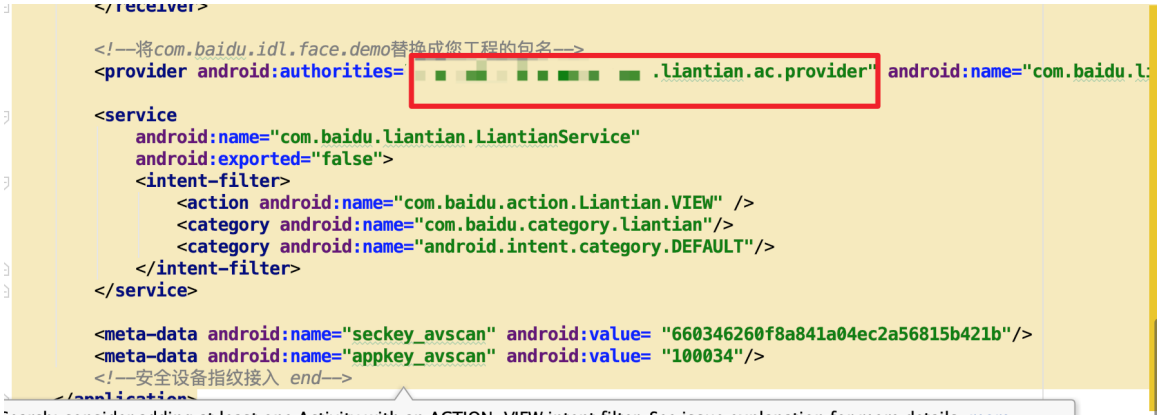
人脸采集SDK4.1 常见问题

Android端人脸采集SDK4.1常见问题

1、如果运行demo或集成时出现以下崩溃信息：



则需要将app module下的AndroidManifest.xml中的liantian库的注册包名改成您工程中的包名



2、license文件有什么用，该放在什么地方？

license文件需要申请，目的是作为sdk校验开发者的使用合法性，license文件放置位置不对或未放置license文件会导致没法使用sdk，一般应先申请license文件，并把申请得到的license文件，放置在assets目录下面。

3、如果应用启动时，在首页弹出以下toast信息：

或者log是这样的：

```
2021-01-26 20:48:07.822 21237-21237/com.baidu.idl.face.demo I/PressGestureDetector: onAttached end
2021-01-26 20:48:07.824 21237-21438/com.baidu.idl.face.demo I/PressGestureDetector: enabledInPad = false,isPcCastMode = false
2021-01-26 20:48:07.851 21237-21285/com.baidu.idl.face.demo D/mali_winsys: EGLint new_window_surface(egl_winsys_display *, void *, EGLSurface,
  EGLConfig, egl_winsys_surface **, egl_color_buffer_format *, EGLBoolean) returns 0x3000
2021-01-26 20:48:07.857 21237-21237/com.baidu.idl.face.demo W/InputMethodManager: startInputReason = 1
2021-01-26 20:48:07.864 21237-21280/com.baidu.idl.face.demo I/FaceSDK: licenseKey:FACEEXAMPLE-FACE-ANDROID,algorithmId:3;packageName:com.baidu.idl
  .face.demo,md5:663B4668271FC91341CB67563DE1992A;deviceId:11140C1F12FEEB2C52DFBEF8EE0AAC81D9,fingerVersion:3.5.7.5;licenseSdkVersion:5.5.0
  .1-mobile-collection
2021-01-26 20:48:07.894 21237-21237/com.baidu.idl.face.demo E/HomeActivity: 初始化失败 = 4  -->local auth failed:4
2021-01-26 20:48:07.896 21237-21237/com.baidu.idl.face.demo D/HwRTBlurUtils: check blur style for HwToast-Toast, themeResId : 0x7f0a0002, context :
  com.baidu.idl.face.example.HomeActivity@616489, Nhwext : 6, get Blur : disable with , android.graphics.drawable.NinePatchDrawable@c4b99f0
2021-01-26 20:48:07.903 21237-21237/com.baidu.idl.face.demo W/VRSystemServiceManager: vr service is not alive
2021-01-26 20:48:07.906 21237-21285/com.baidu.idl.face.demo D/OpenGLRenderer:   HWUI Binary is  enabled
```

则说明鉴权初始化失败了。 鉴权初始化的错误码对应的信息如下：

ErrorCode	常量值	说明
SUCCESS	0	成功
LICENSE_NOT_INIT_ERROR	1	license未初始化
LICENSE_DECRYPT_ERROR	2	license数据解密失败
LICENSE_INFO_FORMAT_ERROR	3	license数据格式错误
LICENSE_KEY_CHECK_ERROR	4	license-key(api-key)校验错误
LICENSE_ALGORITHM_CHECK_ERROR	5	算法ID校验错误
LICENSE_MD5_CHECK_ERROR	6	MD5校验错误
LICENSE_DEVICE_ID_CHECK_ERROR	7	设备ID校验错误
LICENSE_PACKAGE_NAME_CHECK_ERROR	8	包名(应用名)校验错误
LICENSE_EXPIRED_TIME_CHECK_ERROR	9	过期时间不正确
LICENSE_FUNCTION_CHECK_ERROR	10	功能未授权
LICENSE_TIME_EXPIRED	11	授权已过期
LICENSE_LOCAL_FILE_ERROR	12	本地文件读取失败
LICENSE_REMOTE_DATA_ERROR	13	远程数据拉取失败
LICENSE_LOCAL_TIME_ERROR	14	本地时间校验错误
OTHER_ERROR	0xff	其他错误

4、**FaceSDK-LicenseLICENSE_INFO_CHECK_ERROR =license id**

licenseID为您申请时填appname+“_face_android”。如下图demo-turnstile-face-android为license里面的licenseID, **demo-turnstile-face-android1**为app运行时Config.licenseID，两者必须一致

```
E/FaceSDK: FaceSDK-License LICENSE_INFO_CHECK_ERROR =license id  demo-turnstile-face-android  demo-turnstile-face-android1
```

5、**FaceSDK-LicenseLICENSE_INFO_CHECK_ERROR =signature md5**

md5不一致错误，签名的为license里面的md5，后面的为app运行时获取的签名文件的md5，这两个md5必须一致且区分大小写。`E/FaceSDK: FaceSDK-License LICENSE_INFO_CHECK_ERROR =signature md5  F5846C60804CC6042D55D09F1AB82364  4357A3EDBC0CA02EA8B5E0578E58D`

6、**FaceSDK-License LICENSE_INFO_CHECK_ERROR =package name**

PackageName不一致错误。License里面的packagename为申请license时填的，需要保证和app里面的packagename一致。

7、活体检测常见有那些动作？是否可配置？

常见有6个动作，眨眼，张大嘴，向上抬头，向下低头，向左摇头，向右摇头等。 sdk提供FaceConfig参数设置类，如活体检测角度、光线，检测动作，检测动作数量等设置。

8、使用sdk一般会用到活体检测拍照等功能，有什么需要注意？

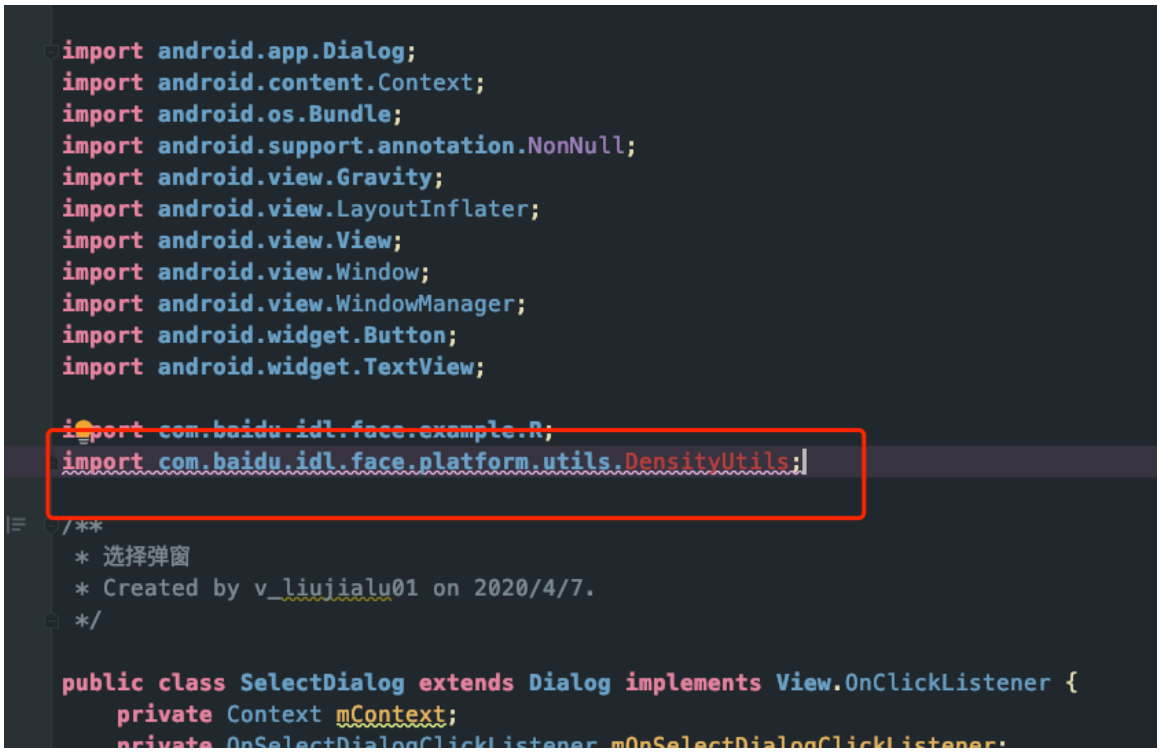
Android 6.0+，需要注意相机拍摄权限问题。如没申请权限，可能导致没法调起相机。

9、license 文件失效了，不能用了怎么办？

license文件申请时候有期限，如过期会导致校验失效，需要在后台申请延期。

10、离线采集安卓端示例工程提示缺少DensityUtils工具类

请您先进行编译，此工具在编译后会产生。DensityUtils工具是用于UI适配用，主要是像素与Android单位dp转换使用。



```
import android.app.Dialog;
import android.content.Context;
import android.os.Bundle;
import android.support.annotation.NonNull;
import android.view.Gravity;
import android.view.LayoutInflater;
import android.view.View;
import android.view.Window;
import android.view.WindowManager;
import android.widget.Button;
import android.widget.TextView;

import com.baidu.idl.face.example.R;
import com.baidu.idl.face.platform.utils.DensityUtils;

/**
 * 选择弹窗
 * Created by v_liujialu01 on 2020/4/7.
 */

public class SelectDialog extends Dialog implements View.OnClickListener {
    private Context mContext;
    private OnSelectDialogClickListener mOnSelectDialogClickListener;
```

IOS端常见问题 1、鉴权问题。提示「验证失败」

A：先确定网络情况是否正常，本地鉴权文件失效了才走网络鉴权。定位错误码，排查鉴权失败的原因。一般是 licenseID 和 bundleID 配置不一致导致的鉴权失败。请注意上线前授权文件一定要更新。

2、license 文件失效了，不能用了怎么办？

A：License 文件申请时候有期限，如过期会导致校验失效，需要在后台进行申请延期。

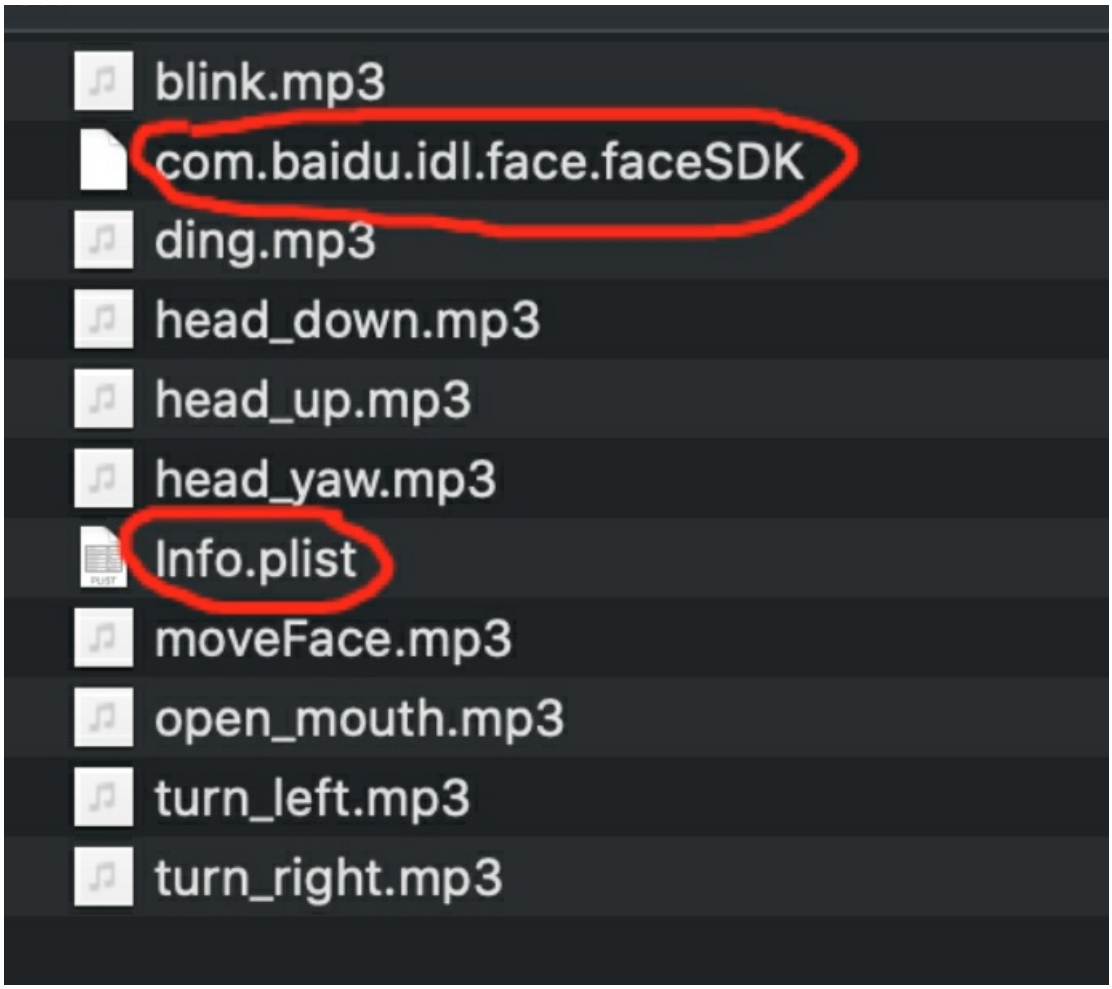
3、使用 iOS 采集端，采集到的图片是斜着的，这个正常吗，会影响识别吗？

A：不会影响识别。有黑边和倾斜是因为图片质量算法造成的，我们是按 1:3 对图像进行背景填充使人脸居中，为的是更好的识别图像。这个版本提供了`detectStratrgyWithQualityControlImage`和`detectStratrgyWithNormalImage`两种方法供选择。

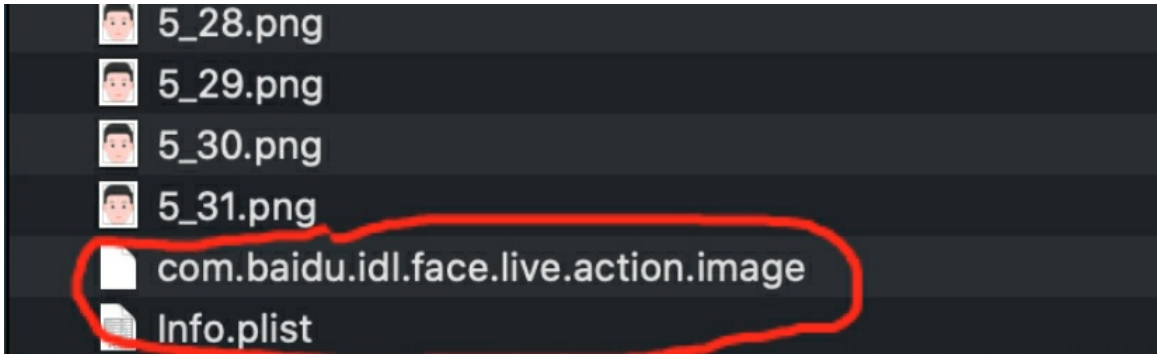
更多问题请点击 常见问题

4、线上审核Bundle文件通不过问题修复

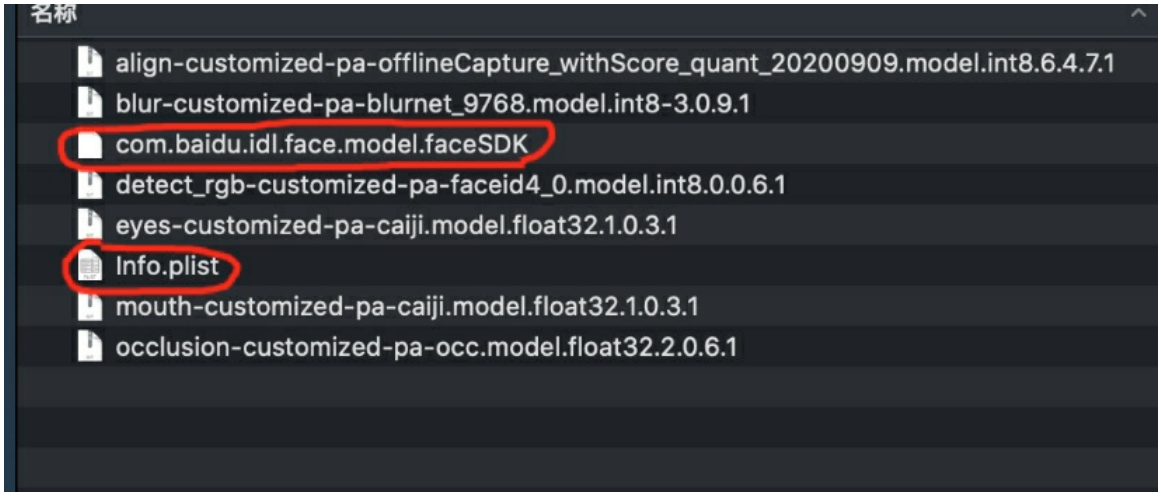
（1） 删除com.baidu.idl.face.faceSDK.bundle中的文件 A：com.baidu.idl.face.faceSDK B：Info.plist 如下图所示



（2）删除com.baidu.idl.face.live.action.image.bundle中的 文件A：com.baidu.idl.face.live.action.image 文件B：Info.plist 如下图所示：



（3）删除com.baidu.idl.face.model.faceSDK.bundle中的 文件A：com.baidu.idl.face.model.faceSDK 文件B：Info.plist 如下图所示：

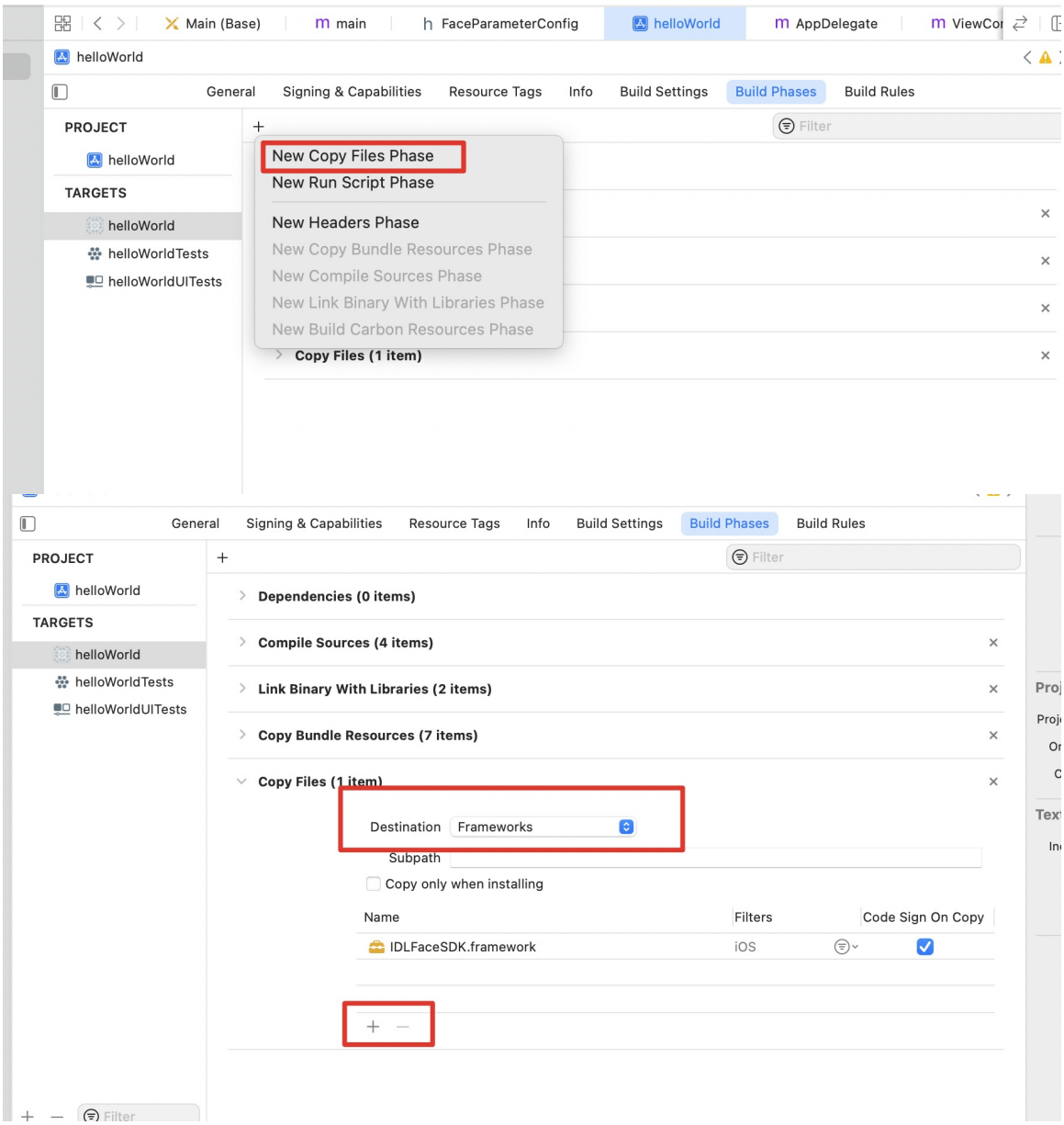


5、MD5无法获取问题修复

您可尝试切换使用jdk1.8版本获取MD5，避免因jdk高版本导致md5获取失败情况

6、集成时候出现library not loaded报错

该问题是由于SDK库资源路径引用查找不到导致的，按照以下步骤引入IDLFaceSDK.framework可解决：



Android-SDK4.1.5

如果您的需求是与人脸比对、人脸实名认证、在线图片活体检测API搭配使用，请**创建APP方案**获取最新5.2版本实名认证方案SDK及示例工程。

人脸实名认证方案-iOS版本
人脸实名认证方案-Android

1.简介

1.1 功能介绍

百度人脸离线采集SDK Android 版是一种面向 Android 移动设备人脸技术开发包，此版SDK包含人脸检测、活体识别等功能，以aar包+动态链接库的形式发布。基于该方案，开发者可以轻松的构建包含人脸检测、采集和活体识别的应用。在您使用SDK之前，我们首先为您介绍以下SDK本身及相关人脸能力，以便您能更方便使用人脸采集服务。

此版SDK所包含的能力如下：

- **离线动作活体检测**：通过让用户做出指定人脸配合式的交互动作，识别当前操作者是否为活体，此功能为离线使用，可设定指定动作是否使用及应用顺序。动作包含：眨眨眼、张闭嘴、向左摇头、向右摇头、向上抬头，向下低头6个。可有效抵御高清图片、3D建模、视频等攻击。

- **离线人脸质量检测**：判断视频流中的图片帧中，哪些图片质量较佳，即人脸图像特征清晰（满足姿态角、光照、模糊度、遮挡等校验）。
- **离线人脸图像采集**：通过本地SDK能力，采集人脸图像，同时经过人脸质量检测，确保采集到的人脸图像符合各条件校验（满足姿态角、光照、模糊度、遮挡等校验），为设备前端获取有效可分析人脸的主要功能。
- **离线授权**：SDK的授权判断，授权介质也称为license，在SDK使用中，优先验证本地离线鉴权文件，验证通过可离线使用；在本地鉴权失败情况下需要向授权服务器发起请求，远程拉取授权进行验证。

> 此版SDK**全部功能为离线版本，所有功能均本地化使用**，主要用于在客户端（Android）获取人脸，实际业务使用中，可以按照业务需要，配合在线API完成全流程的业务集成。

为了方便您的开发，我们已经为您准备了【人脸实名认证】场景的示例工程，您可以根据业务需要，创建APP方案后进行直接下载。

1.2 兼容性

系统：支持 Android 5.1(API Level 22)及以上系统。需要开发者通过 minSdkVersion来保证支持系统的检测。

机型：手机（不支持横屏或后置摄像头）

构架：支持 CPU架构平台【armeabi-v7a、arm64-v8a】

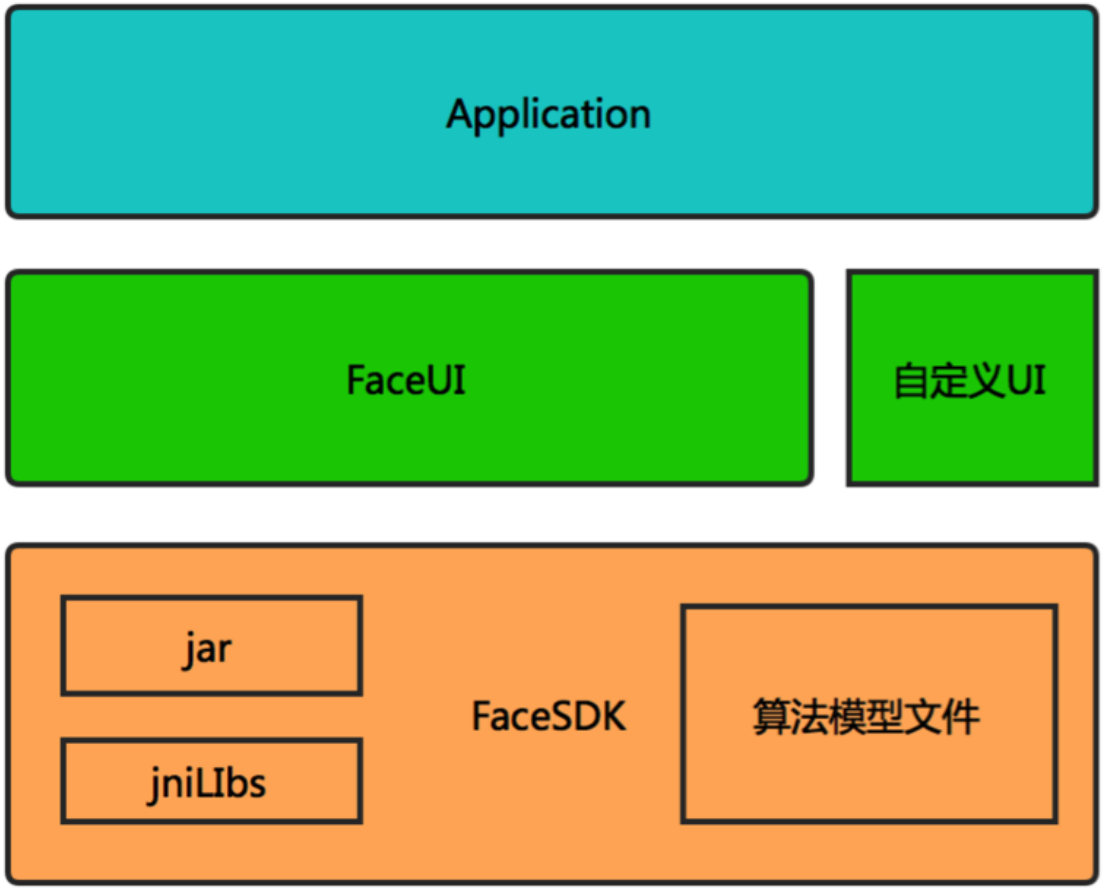
网络：支持 WIFI 及移动网络。

1.3 开发包说明

文件/文件夹名	说明
/faceplatform-release	SDK lib库、模型文件以及采集逻辑代码打包的aar
/faceplatform-ui	SDK的UI库，封装采集和动作活体UI等功能，以及各平台的so库。so包含以下几个平台如果关注包大小，请自行删减。【armeabi-v7a、arm64-v8a】
/app	DEMO工程（包含首页面、采集成功失败页面、设置页面等）

2.集成指南

> 本章将进行 Step-By-Step的讲解,如何快速的集成 人脸Sdk到现有应用中。一个完整的Demo 请参考开发包中的示例程序 FacePlatform。方案架构参考下图：

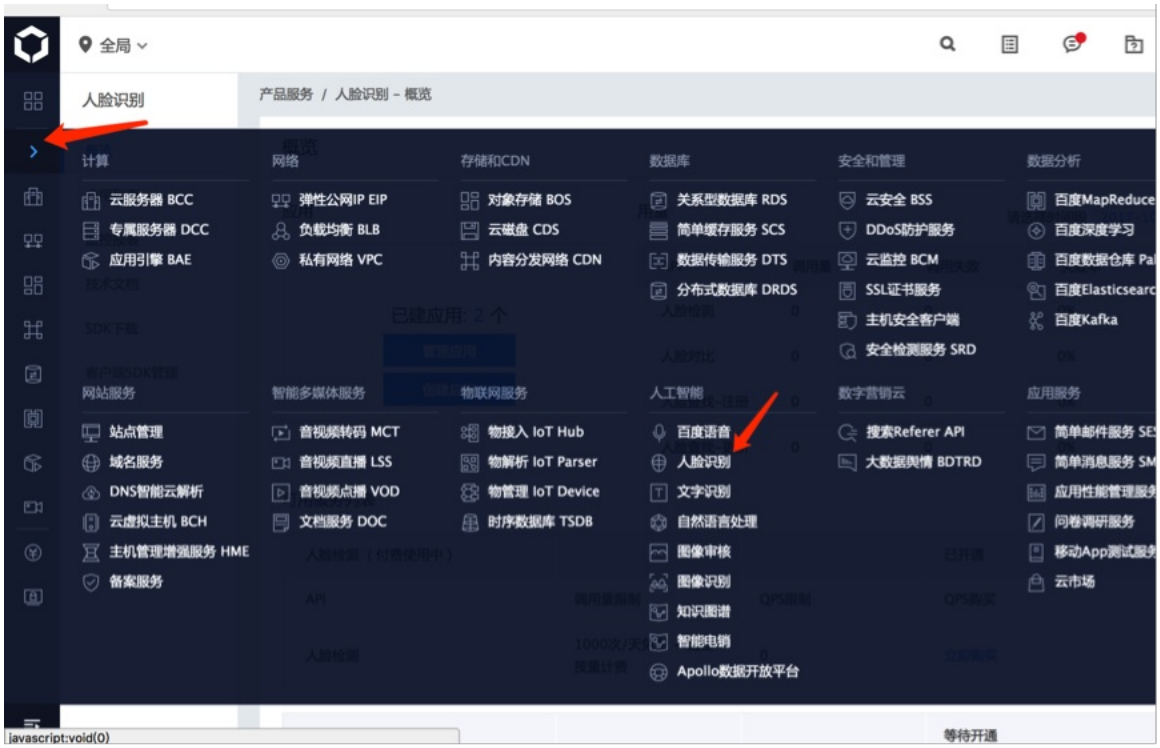


2.1 准备工作

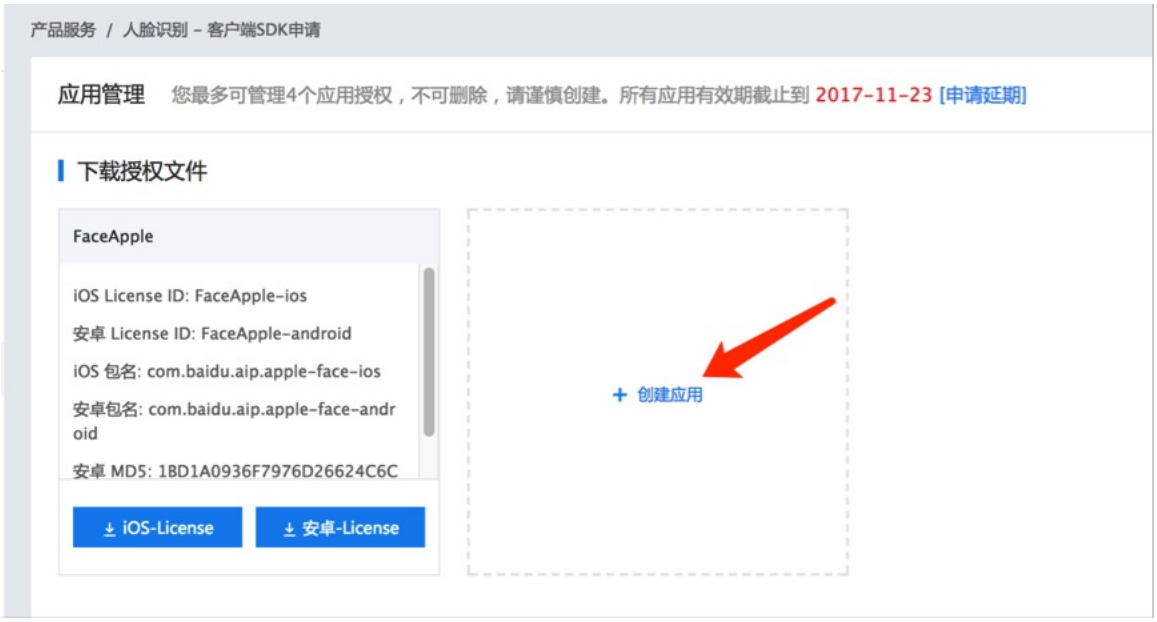
2.1.1 申请license

人脸SDK License：此license用于SDK离线功能使用，在您的申请人脸SDK的后台页面，全局->产品服务->人脸识别->客户端SDK申请

人脸控制台路径如下：

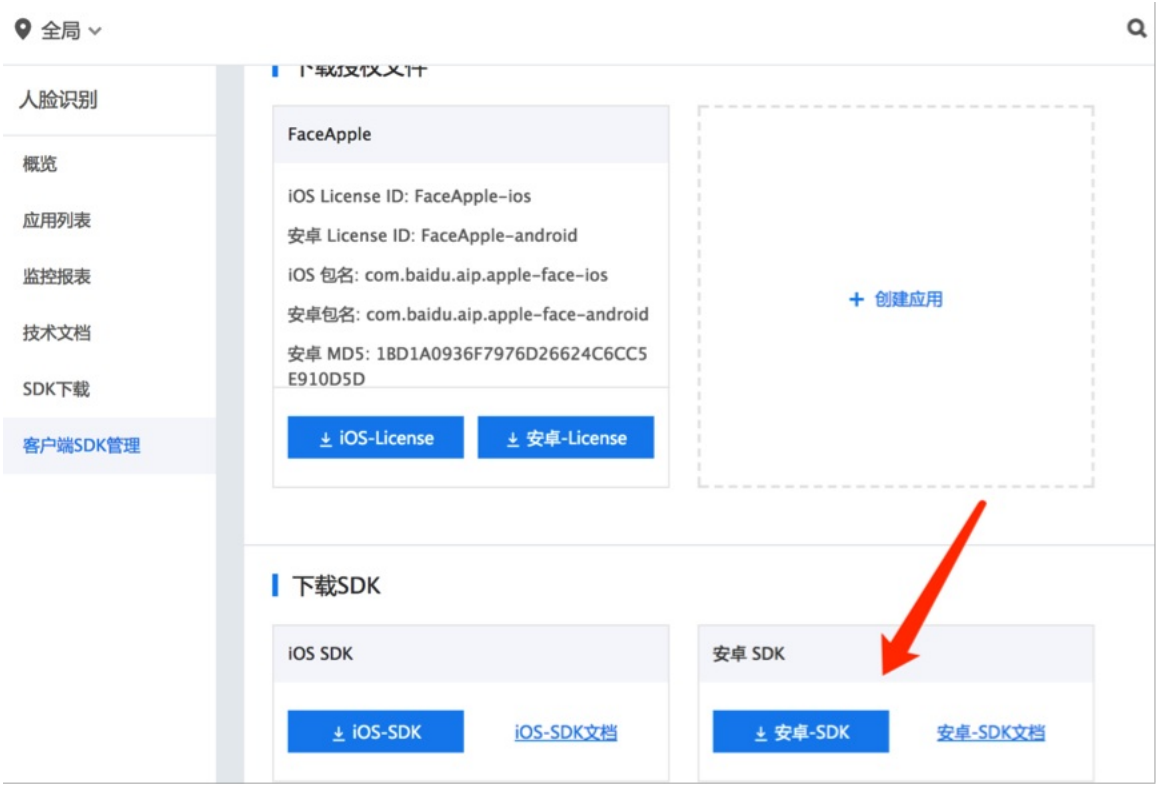


点击客户端SDK管理，弹出如下图：创建应用（这里创建应用是为了使用离线SDK，上面创建应用为了使用人脸在线接口，如注册、识别等）

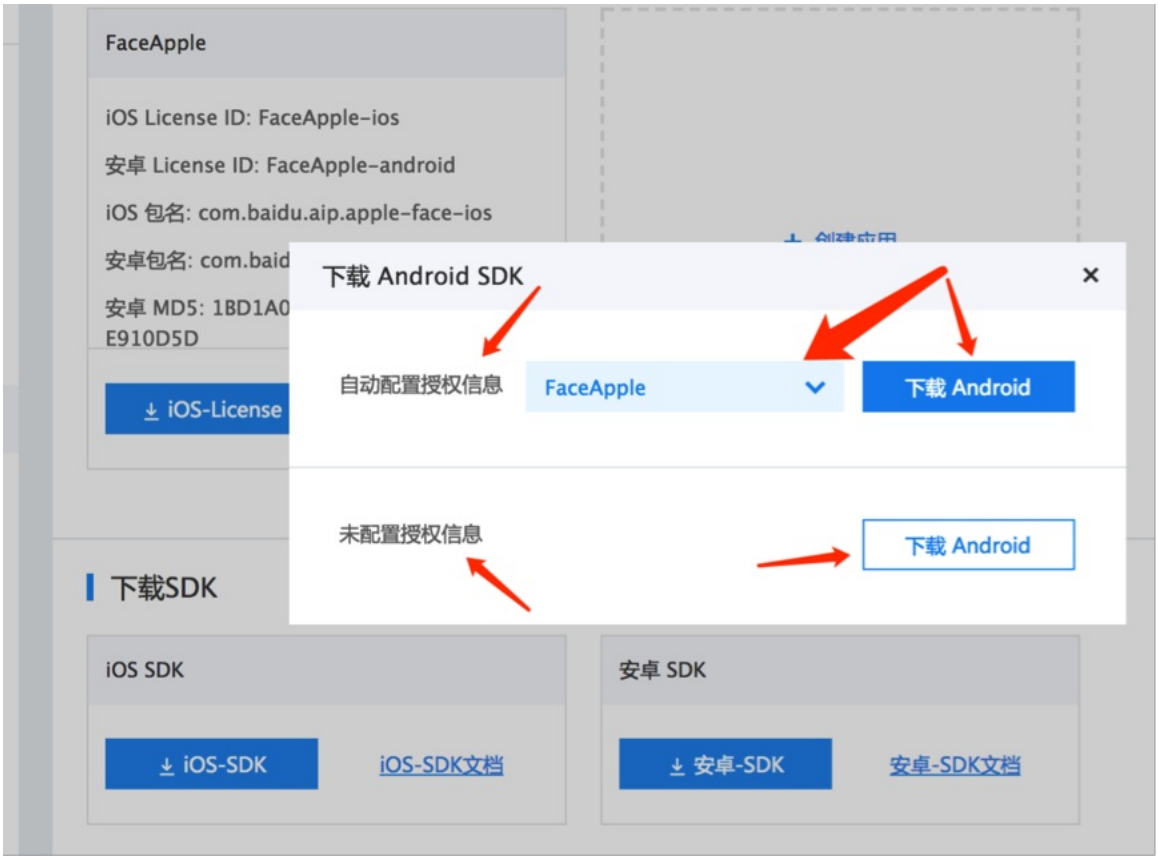


在弹出的框中输入授权标识，选择应用类型，应用系统，以及包名、MD5签名，详情请查看输入框右边提示

2.1.2 下载SDK



下载SDK分为自动配置授权信息（创建license后就可以选择为该应用，下载后SDK自动帮您配置授权，不用下载license拷贝到工程中，初始化参数licenseID,包名也帮您配置好了）和未配置授权信息两种方式：



2.2 运行示例工程

2.2.1 运行自动配置授权信息的示例工程

该下载的示例工程，已经帮你改好了license和包名

- Android Studio导入下载的示例工程
- 配置打包签名文件，由于SDK运行时会校验签名文件的MD5是否和申请时填入一样。为了便于debug能正常使用SDK，需要

配置打包签名文件。

- 运行示例工程


```
    }
    signingConfigs {
        debug {
            storeFile file("[redacted]")
            storePassword '[redacted]'
            keyAlias '[redacted]'
            keyPassword '[redacted]'
        }
        release {
            storeFile file("[redacted]")
            storePassword '[redacted]'
            keyAlias '[redacted]'
            keyPassword '[redacted]'
        }
    }
    buildTypes {
```

签名文件的路径

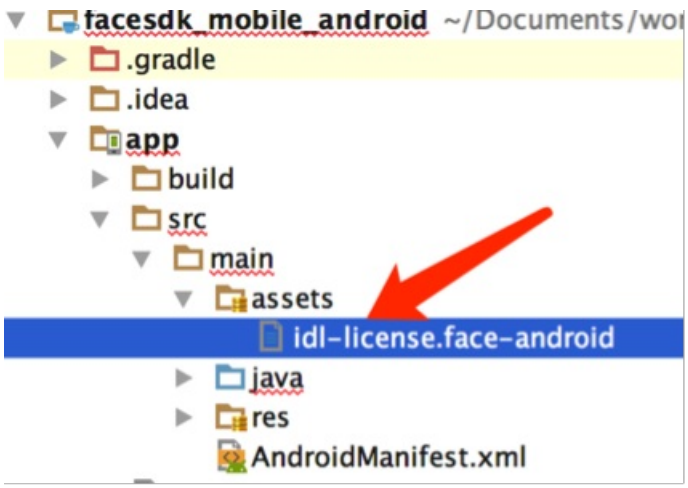
签名文件的别名

签名文件的密码

2.2.2 运行未配置授权信息的示例工程

（1）Android Studio导入下载的示例工程

（2）下载license拷贝到工程的assets目录

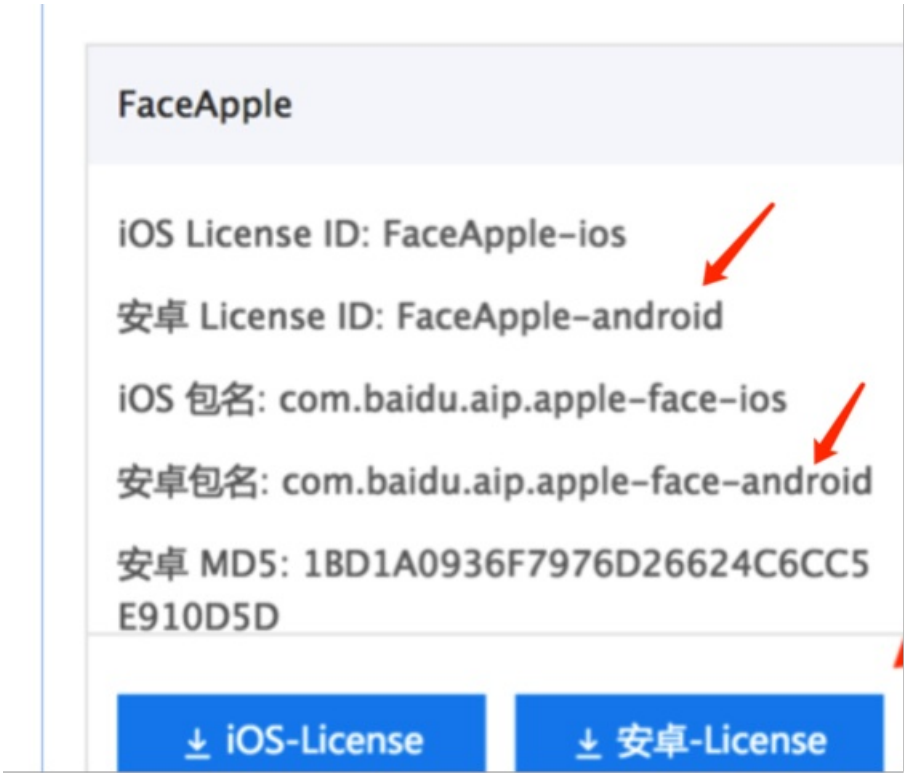


（3）修改HomeActivity.java的initialize方法参数

```
// 为了android和ios 区分授权，appId=appname_face_android ,其中appname为申请sdk时的应用名
// 应用上下文
// 申请License取得的APPID
// assets目录下License文件名
FaceSDKManager.getInstance().initialize(mContext,  licenseID: "[redacted]",
        licenseFileName: "[redacted]", new IInitCallback() {
            @Override
            public void initSuccess() {
                runOnUiThread(() -> {
                        Log.e(TAG,  msg: "初始化成功");
                        showToast( msg: "初始化成功");
                        mIsInitSuccess = true;
                });
            }

            @Override
            public void initFailure(final int errCode, final String errMsg) {
                runOnUiThread(() -> {
                        Log.e(TAG,  msg: "初始化失败 = " + errCode + " " + errMsg);
                        showToast( msg: "初始化失败 = " + errCode + ", " + errMsg);
                        mIsInitSuccess = false;
                });
            }
        });
```

查看申请license信息，里面包含licenseId

填入放到assets目录下的授权文件名称



（4）修改app->build.gradle里面的包名为申请license填入的包名，如上图安卓包名。

（5）配置打包签名文件，由于SDK运行时会校验签名文件的MD5是否和申请时填入一样。为了便于debug能正常使用SDK，需要配置打包签名文件。

（6）运行示例工程。如果无法正常体验，请查看logcat日志。是否有 **FaceSDK-License LICENSE_INFO_CHECK_ERROR**日志。如果有说明授权没有成功，可以查看本文档最后的常见问题进行解决。

2.3 添加SDK到工程

FaceSdk以androidstudio开发方式提供，以下介绍在android studio开发工具导入FaceSdk

（1）将开发包中的faceplatform-release库Copy 到工程根目录。

（2）将开发包中的faceplatform-ui库Copy 到工程根目录。

（3）SDK提供的了开源的faceplatform-ui库，把活体检测和人脸图像采集功能等功能进行了封装，适配了主流机型机型。如果需要使用，请添加faceplatform-ui模块到的工程中。faceplatform-ui目录结构如下图

```
    ![img](http://bjhw-bce-online-public0.bjhw.baidu.com:8109/proxy-external/?
url=https://ai.bdstatic.com/file/A22B3618EB134243ADD6A54691D28791)
```

（4）在build.gradle使用compile project引人faceplatform-ui库工程。

```
dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    compile "com.android.support:recyclerview-v7:+"
    compile project(path: ':faceplatform-ui')
}
```

（5）Setting.gradle中include faceplatfrom-ui和facepaltfrom-release

```
include ':app', ':faceplatform-release'
include ':faceplatform-ui'
```

（6）从官网下载授权文件license，复制到app/src/main/assets目录下。

（7）申请的license已经和打包签名key进行了绑定（申请时用到了签名的md5，为了便于debug模式也能调用SDK的功能，需要把debug的key改成申请license的key。

- 把key拷贝到项目根目录下
- 主appbuild.gradle android 下面添加(修改)signingConfigs相关的配置。如下图。

```
    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'
        }
    }

    signingConfigs {
        debug {
            keyAlias 'facesharp'
            keyPassword [redacted]
            storeFile file('../facesharp.jks')
            storePassword([redacted])
        }
        release {
            keyAlias [redacted]
            keyPassword [redacted]
            storeFile file('../facesharp.jks')
            storePassword([redacted])
        }
    }
```

2.4 权限声明

名称	用途
需要动态申请的权限	
android.permission.READ_EXTERNAL_STORAGE	读取手机外部存储权限
android.permission.WRITE_EXTERNAL_STORAGE	写入手机外部存储权限
android.permission.CAMERA	拍照权限
不需要动态申请的权限	
android.permission.READ_PHONE_STATE	获取用户手机的 IMEI,用来唯一的标识用户
android.hardware.camera.autofocus	允许相机对焦
android.permission.INTERNET	允许访问网络
android.permission.WRITE_SETTINGS	允许修改系统设置
android.permission.WAKE_LOCK	屏幕常亮权限

2.5 混淆设置

如果工程需要做混淆处理的话，则在工程的proguard-rules.pro文件中添加如下配置：

```
-keep class com.baidu.vis.unified.license.** {*;}

-keep class com.baidu.liantian.** {*;}

-keep class com.baidu.baidusec.** {*;}

-keep class com.baidu.idl.main.facesdk.** {*;}
```

3.功能使用

3.1 人脸采集（包含动作活体）

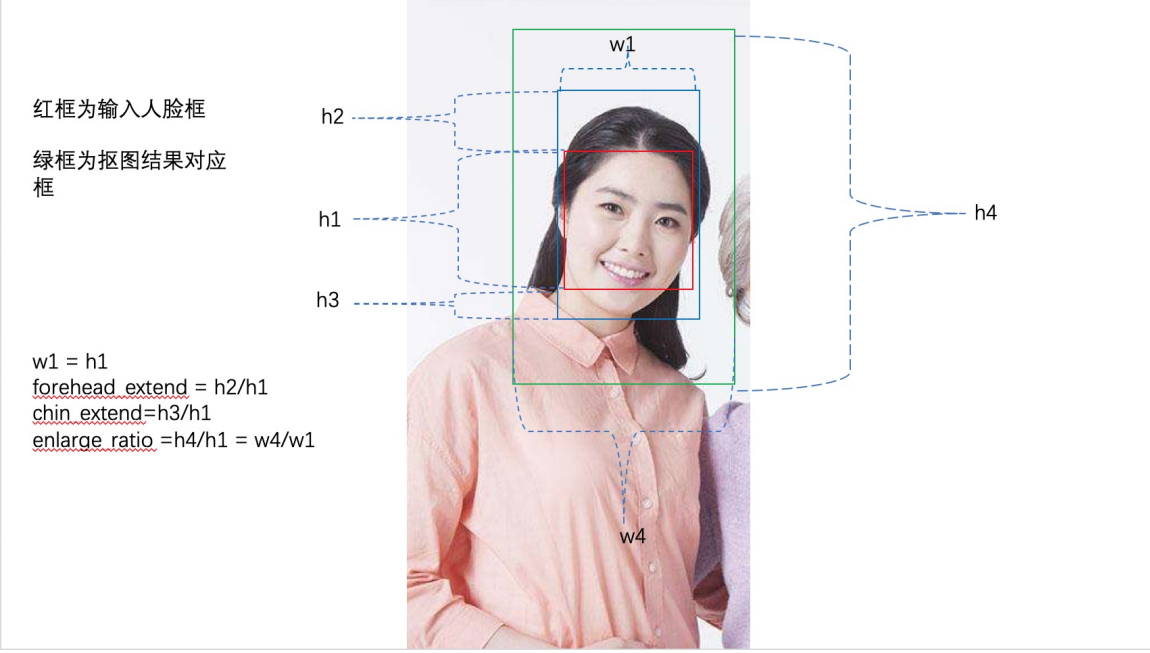
（1）初始化SDK **参数分别表示：当前上下文、鉴权key、鉴权名称、回调参数** 调用 FaceSDKManager.getIntance().initialize(Context context, String licenseID, String licenseFileName, IInitCallback callback); **Demo中此段代码在HomeActivity中。**

（2） 初始化参数设置（以下参数的配置文件在qpp->src->main->assets->quality_config.json）

```
 FaceConfig config = FaceSDKManager.getInstance().getFaceConfig();
// SDK初始化已经设置完默认参数（推荐参数），也可以根据实际需求进行数值调整
// 质量等级（0：正常、1：宽松、2：严格、3：自定义）
// 获取保存的质量等级
SharedPreferencesUtil util = new SharedPreferencesUtil(mContext);
int qualityLevel = (int) util.getSharedPreference(Const.KEY_QUALITY_LEVEL_SAVE, -1);
if (qualityLevel == -1) {
    qualityLevel = ExampleApplication.qualityLevel;
}
// 根据质量等级获取相应的质量值（注：第二个参数要与质量等级的set方法参数一致）
QualityConfigManager manager = QualityConfigManager.getInstance();
manager.readQualityFile(mContext.getApplicationContext(), qualityLevel);
QualityConfig qualityConfig = manager.getConfig();
if (qualityConfig == null) {
    return false;
}
// 设置模糊度阈值
config.setBlurnessValue(qualityConfig.getBlur());
// 设置最小光照阈值（范围0-255）
config.setBrightnessValue(qualityConfig.getMinIllum());
// 设置最大光照阈值（范围0-255）
config.setBrightnessMaxValue(qualityConfig.getMaxIllum());
// 设置左眼遮挡阈值
config.setOcclusionLeftEyeValue(qualityConfig.getLeftEyeOcclusion());
// 设置右眼遮挡阈值
config.setOcclusionRightEyeValue(qualityConfig.getRightEyeOcclusion());
// 设置鼻子遮挡阈值
config.setOcclusionNoseValue(qualityConfig.getNoseOcclusion());
// 设置嘴巴遮挡阈值
config.setOcclusionMouthValue(qualityConfig.getMouseOcclusion());
// 设置左脸颊遮挡阈值
config.setOcclusionLeftContourValue(qualityConfig.getLeftContourOcclusion());
// 设置右脸颊遮挡阈值
config.setOcclusionRightContourValue(qualityConfig.getRightContourOcclusion());
// 设置下巴遮挡阈值
config.setOcclusionChinValue(qualityConfig.getChinOcclusion());
// 设置人脸姿态角阈值
config.setHeadPitchValue(qualityConfig.getPitch());
config.setHeadYawValue(qualityConfig.getYaw());
config.setHeadRollValue(qualityConfig.getRoll());
// 设置可检测的最小人脸阈值
config.setMinFaceSize(FaceEnvironment.VALUE_MIN_FACE_SIZE);
// 设置可检测到人脸的阈值
config.setNotFaceValue(FaceEnvironment.VALUE_NOT_FACE_THRESHOLD);
```

```
// 设置闭眼阈值
config.setEyeClosedValue(FaceEnvironment.VALUE_CLOSE_EYES);
// 设置图片缓存数量
config.setCacheImageNum(FaceEnvironment.VALUE_CACHE_IMAGE_NUM);
// 设置活体动作，通过设置list，LivenessTypeEunm.Eye, LivenessTypeEunm.Mouth,
// LivenessTypeEunm.HeadUp, LivenessTypeEunm.HeadDown, LivenessTypeEunm.HeadLeft,
// LivenessTypeEunm.HeadRight
config.setLivenessTypeList(ExampleApplication.livenessList);
// 设置动作活体是否随机
config.setLivenessRandom(ExampleApplication.isLivenessRandom);
// 设置开启提示音
config.setSound(ExampleApplication.isOpenSound);
// 原图缩放系数
config.setScale(FaceEnvironment.VALUE_SCALE);
// 抠图宽高的设定，为了保证好的抠图效果，建议高宽比是4：3
config.setCropHeight(FaceEnvironment.VALUE_CROP_HEIGHT);
config.setCropWidth(FaceEnvironment.VALUE_CROP_WIDTH);
// 抠图人脸框与背景比例
config.setEnlargeRatio(FaceEnvironment.VALUE_CROP_ENLARGERATIO);
// 加密类型，0：Base64加密，上传时image_sec传false；1：百度加密文件加密，上传时image_sec传true
config.setSecType(FaceEnvironment.VALUE_SEC_TYPE);
// 检测超时设置
config.setTimeDetectModule(FaceEnvironment.TIME_DETECT_MODULE);
// 检测框远近比率
config.setFaceFarRatio(FaceEnvironment.VALUE_FAR_RATIO);
config.setFaceClosedRatio(FaceEnvironment.VALUE_CLOSED_RATIO);
FaceSDKManager.getInstance().setFaceConfig(config);
```

关于抠图内部实现方案如下图所示：



（3） startActivity(new Intent(this, FaceLivenessExpActivity.class))，开启预览。

（4）调用FaceSDKManager.getInstance().getLivenessStrategyModule()获得ILivenessStrategy对象。(该方法每次调用都会返回一个新对象).

（5）调用ILivenessStrategy.setPreviewDegree();设置预览图片的旋转角度。
调用setDetectStrategySoundEnable设置是否开启语音。
调用setDetectStrategyConfig设置，预览图的大小，人脸检测框的坐标和回调。

（6）多次调用livenessStrategy进行人脸图片采集，人脸跟踪、质量检测、动作活体检测。

（7）实现ILivenessStrategyCallback的onLivenessCompletion并处理结果。**其中base64ImageCropMap为存放人脸抠图图片**

集，base64ImageSrcMap为存放人脸原图图片集并通过调用getBestImage()方法，通过从优到差的质量排序，获取最优的bitmap，代码如下图所示：

```
main 〉 java 〉 com 〉 baidu 〉 idl 〉 face 〉 example 〉 FaceLivenessExpActivity
ategyExtModule.java × | SoundPoolHelper.java × | SoundPlayer.java × | FaceLivenessExpActivity.java × | FileUtils.java ×

@Override
public void onLivenessCompletion(FaceStatusNewEnum status, String message,
                                 HashMap<String, ImageInfo> base64ImageCropMap,
                                 HashMap<String, ImageInfo> base64ImageSrcMap, int currentLivenessCount) {
    super.onLivenessCompletion(status, message, base64ImageCropMap, base64ImageSrcMap, currentLivenessCount);
    if (status == FaceStatusNewEnum.OK && mIsCompletion) {
        // 获取最优图片
        getBestImage(base64ImageCropMap, base64ImageSrcMap);
    } else if (status == FaceStatusNewEnum.DetectRemindCodeTimeout) {
        if (mViewBg != null) {
            mViewBg.setVisibility(View.VISIBLE);
        }
        showMessageDialog();
    }
}


    /**
     * 获取最优图片
     * @param imageCropMap 抠图集合
     * @param imageSrcMap  原图集合
     */
    private void getBestImage(HashMap<String, ImageInfo> imageCropMap, HashMap<String, ImageInfo> imageSrcMap) {
        String bmpStr = null;
        // 将抠图集合中的图片按照质量降序排序，最终选取质量最优的一张抠图图片
        if (imageCropMap != null && imageCropMap.size() > 0) {
            List<Map.Entry<String, ImageInfo>> list1 = new ArrayList<>(imageCropMap.entrySet());
            Collections.sort(list1, new Comparator<Map.Entry<String, ImageInfo>>() {

                @Override
                public int compare(Map.Entry<String, ImageInfo> o1,
                                   Map.Entry<String, ImageInfo> o2) {
                    String[] key1 = o1.getKey().split("_");
                    String score1 = key1[2];
                    String[] key2 = o2.getKey().split("_");
                    String score2 = key2[2];
                    // 降序排序
                    return Float.valueOf(score2).compareTo(Float.valueOf(score1));
                }
            });

            // 获取抠图中的加密或非加密的base64
//            int secType = mFaceConfig.getSecType();
//            String base64;
//            if (secType == 0) {
//                base64 = list1.get(0).getValue().getBase64();
//            } else {
//                base64 = list1.get(0).getValue().getSecBase64();
//            }
        }


        // 将原图集合中的图片按照质量降序排序，最终选取质量最优的一张原图图片
        if (imageSrcMap != null && imageSrcMap.size() > 0) {
            List<Map.Entry<String, ImageInfo>> list2 = new ArrayList<>(imageSrcMap.entrySet());
            Collections.sort(list2, new Comparator<Map.Entry<String, ImageInfo>>() {

                @Override
                public int compare(Map.Entry<String, ImageInfo> o1,
                                   Map.Entry<String, ImageInfo> o2) {
                    String[] key1 = o1.getKey().split("_");
                    String score1 = key1[2];
                    String[] key2 = o2.getKey().split("_");
                    String score2 = key2[2];
                    // 降序排序
                    return Float.valueOf(score2).compareTo(Float.valueOf(score1));
                }
            });
            bmpStr = list2.get(0).getValue().getBase64();

            // 获取原图中的加密或非加密的base64
//            int secType = mFaceConfig.getSecType();
//            String base64;
//            if (secType == 0) {
//                base64 = mBmpStr;
//            } else {
//                base64 = list2.get(0).getValue().getBase64();
//            }
        }

        // 页面跳转
        IntentUtils.getInstance().setBitmap(bmpStr);
        Intent intent = new Intent(FaceLivenessExpActivity.this,
                CollectionSuccessActivity.class);
        intent.putExtra("destroyType", "FaceLivenessExpActivity");
        startActivity(intent);
    }
```

（8）Demo中的最优抠图或者最优原图，可以调用SecRequest类下的sendMessage(Context context, String secBase64, int secType);将加密后的base64发送到服务端，如需风控，需要把SecRequest.java下的risk_identify参数置为true。

3.2 人脸采集（不包含动作活体）

（1）初始化SDK **参数分别表示：当前上下文、鉴权key、鉴权名称、回调参数** 调用 FaceSDKManager.getIntance().initialize(Context context, String licenseID, String licenseFileName, IInitCallback callback); **Demo中此段代码在HomeActivity中。**

（2） 初始化参数设置

```
 FaceConfig config = FaceSDKManager.getInstance().getFaceConfig();
// SDK初始化已经设置完默认参数（推荐参数），也可以根据实际需求进行数值调整
// 质量等级（0：正常、1：宽松、2：严格、3：自定义）
// 获取保存的质量等级
SharedPreferencesUtil util = new SharedPreferencesUtil(mContext);
int qualityLevel = (int) util.getSharedPreference(Const.KEY_QUALITY_LEVEL_SAVE, -1);
if (qualityLevel == -1) {
    qualityLevel = ExampleApplication.qualityLevel;
}
// 根据质量等级获取相应的质量值（注：第二个参数要与质量等级的set方法参数一致）
QualityConfigManager manager = QualityConfigManager.getInstance();
manager.readQualityFile(mContext.getApplicationContext(), qualityLevel);
QualityConfig qualityConfig = manager.getConfig();
if (qualityConfig == null) {
    return false;
}
// 设置模糊度阈值
config.setBlurnessValue(qualityConfig.getBlur());
// 设置最小光照阈值（范围0-255）
config.setBrightnessValue(qualityConfig.getMinIllum());
// 设置最大光照阈值（范围0-255）
config.setBrightnessMaxValue(qualityConfig.getMaxIllum());
// 设置左眼遮挡阈值
config.setOcclusionLeftEyeValue(qualityConfig.getLeftEyeOcclusion());
// 设置右眼遮挡阈值
config.setOcclusionRightEyeValue(qualityConfig.getRightEyeOcclusion());
// 设置鼻子遮挡阈值
config.setOcclusionNoseValue(qualityConfig.getNoseOcclusion());
// 设置嘴巴遮挡阈值
config.setOcclusionMouthValue(qualityConfig.getMouseOcclusion());
// 设置左脸颊遮挡阈值
config.setOcclusionLeftContourValue(qualityConfig.getLeftContourOcclusion());
// 设置右脸颊遮挡阈值
config.setOcclusionRightContourValue(qualityConfig.getRightContourOcclusion());
// 设置下巴遮挡阈值
config.setOcclusionChinValue(qualityConfig.getChinOcclusion());
// 设置人脸姿态角阈值
config.setHeadPitchValue(qualityConfig.getPitch());
config.setHeadYawValue(qualityConfig.getYaw());
config.setHeadRollValue(qualityConfig.getRoll());
// 设置可检测的最小人脸阈值
config.setMinFaceSize(FaceEnvironment.VALUE_MIN_FACE_SIZE);
// 设置可检测到人脸的阈值
config.setNotFaceValue(FaceEnvironment.VALUE_NOT_FACE_THRESHOLD);
// 设置闭眼阈值
config.setEyeClosedValue(FaceEnvironment.VALUE_CLOSE_EYES);
// 设置图片缓存数量
config.setCacheImageNum(FaceEnvironment.VALUE_CACHE_IMAGE_NUM);
// 设置活体动作，通过设置list，LivenessTypeEunm.Eye, LivenessTypeEunm.Mouth,
// LivenessTypeEunm.HeadUp, LivenessTypeEunm.HeadDown, LivenessTypeEunm.HeadLeft,
```

```
// LivenessTypeEunm.HeadRight
config.setLivenessTypeList(ExampleApplication.livenessList);
// 设置动作活体是否随机
config.setLivenessRandom(ExampleApplication.isLivenessRandom);
// 设置开启提示音
config.setSound(ExampleApplication.isOpenSound);
// 原图缩放系数
config.setScale(FaceEnvironment.VALUE_SCALE);
// 抠图宽高的设定，为了保证好的抠图效果，建议高宽比是4：3
config.setCropHeight(FaceEnvironment.VALUE_CROP_HEIGHT);
config.setCropWidth(FaceEnvironment.VALUE_CROP_WIDTH);
// 抠图人脸框与背景比例
config.setEnlargeRatio(FaceEnvironment.VALUE_CROP_ENLARGERATIO);
// 加密类型，0：Base64加密，上传时image_sec传false；1：百度加密文件加密，上传时image_sec传true
config.setSecType(FaceEnvironment.VALUE_SEC_TYPE);
// 检测超时设置
config.setTimeDetectModule(FaceEnvironment.TIME_DETECT_MODULE);
// 检测框远近比率
config.setFaceFarRatio(FaceEnvironment.VALUE_FAR_RATIO);
config.setFaceClosedRatio(FaceEnvironment.VALUE_CLOSED_RATIO);
FaceSDKManager.getInstance().setFaceConfig(config);
```

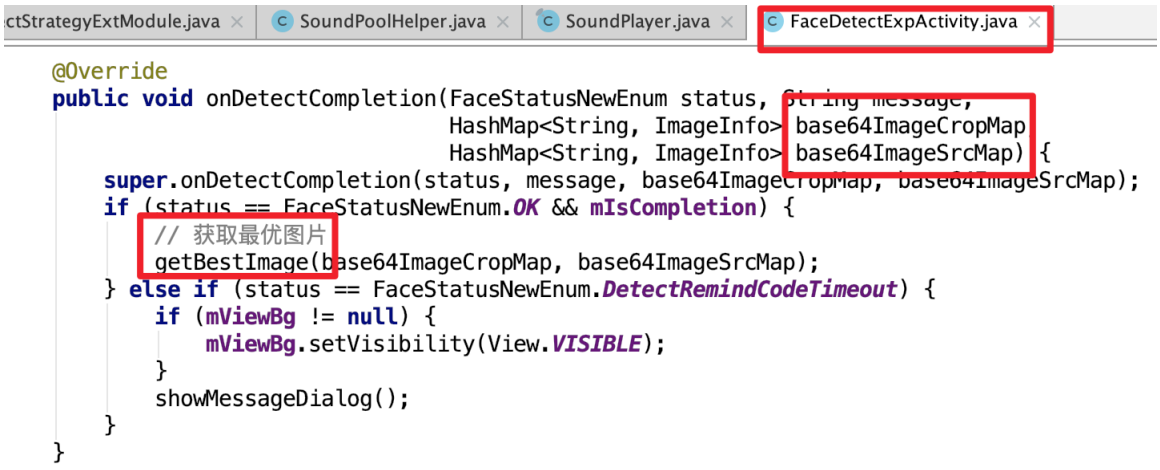
（3）startActivity(new Intent(this, FaceDetectExpActivity.class))，开启预览。

（4）调用FaceSDKManager.getInstance().getDetectStrategyModule()获得IDetectStrategy对象。(该方法每次调用都会返回一个新对象).

（5）调用IDetectStrategy.setPreviewDegree();设置预览图片的旋转角度。调用setDetectStrategySoundEnable设置是否开启语音。调用setDetectStrategyConfig设置，预览图的大小，人脸检测框的坐标和回调。

（6） 多次调用detectStrategy进行人脸图片采集。

（7） 实现IDetectStrategyCallback的onDetectCompletion并处理结果。**其中base64ImageCropMap为存放人脸抠图图片集，base64ImageSrcMap为存放人脸原图图片集**并通过调用getBestImage()方法，通过从优到差的质量排序，获取最优的bitmap，代码如下图所示：



```
    /**
     * 获取最优图片
     * @param imageCropMap 抠图集合
     * @param imageSrcMap  原图集合
     */
    private void getBestImage(HashMap<String, ImageInfo> imageCropMap, HashMap<String, ImageInfo> imageSrcMap) {
        String bmpStr = null;
        // 将抠图集合中的图片按照质量降序排序，最终选取质量最优的一张抠图图片
        if (imageCropMap != null && imageCropMap.size() > 0) {
            List<Map.Entry<String, ImageInfo>> list1 = new ArrayList<>(imageCropMap.entrySet());
            Collections.sort(list1, new Comparator<Map.Entry<String, ImageInfo>>() {

                @Override
                public int compare(Map.Entry<String, ImageInfo> o1,
                                   Map.Entry<String, ImageInfo> o2) {
                    String[] key1 = o1.getKey().split("_");
                    String score1 = key1[2];
                    String[] key2 = o2.getKey().split("_");
                    String score2 = key2[2];
                    // 降序排序
                    return Float.valueOf(score2).compareTo(Float.valueOf(score1));
                }
            });

            // 获取抠图中的加密或非加密的base64
//            int secType = mFaceConfig.getSecType();
//            String base64;
//            if (secType == 0) {
//                base64 = list1.get(0).getValue().getBase64();
//            } else {
//                base64 = list1.get(0).getValue().getSecBase64();
//            }
        }


        // 将原图集合中的图片按照质量降序排序，最终选取质量最优的一张原图图片
        if (imageSrcMap != null && imageSrcMap.size() > 0) {
            List<Map.Entry<String, ImageInfo>> list2 = new ArrayList<>(imageSrcMap.entrySet());
            Collections.sort(list2, new Comparator<Map.Entry<String, ImageInfo>>() {

                @Override
                public int compare(Map.Entry<String, ImageInfo> o1,
                                   Map.Entry<String, ImageInfo> o2) {
                    String[] key1 = o1.getKey().split("_");
                    String score1 = key1[2];
                    String[] key2 = o2.getKey().split("_");
                    String score2 = key2[2];
                    // 降序排序
                    return Float.valueOf(score2).compareTo(Float.valueOf(score1));
                }
            });
            bmpStr = list2.get(0).getValue().getBase64();

            // 获取原图中的加密或非加密的base64
//            int secType = mFaceConfig.getSecType();
//            String base64;
//            if (secType == 0) {
//                base64 = mBmpStr;
//            } else {
//                base64 = list2.get(0).getValue().getBase64();
//            }
        }

        // 页面跳转
        IntentUtils.getInstance().setBitmap(bmpStr);
        Intent intent = new Intent(FaceLivenessExpActivity.this,
                CollectionSuccessActivity.class);
        intent.putExtra("destroyType", "FaceLivenessExpActivity");
        startActivity(intent);
    }
```

（8）Demo中的最优抠图或者最优原图，可以调用SecRequest类下的sendMessage(Context context, String secBase64, int secType);将加密后的base64发送到服务端，如需风控，需要把SecRequest.java下的risk_identify参数置为true。

3.3 质量校验设置

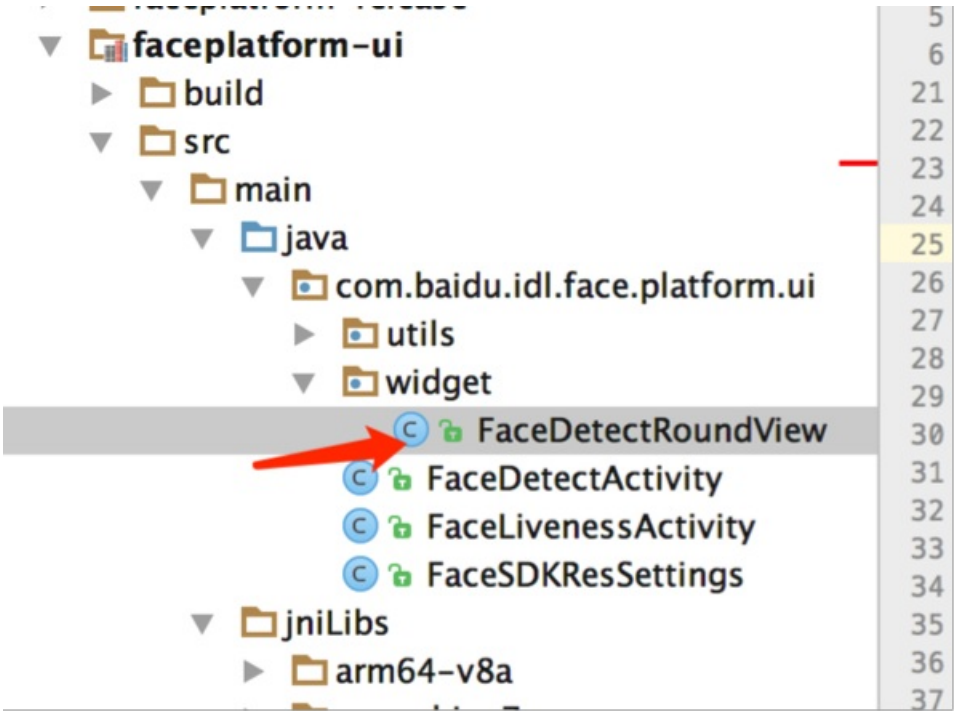
com.baidu.idl.face.platform.FaceConfig类用于人脸检测参数设置。

参数	名称	默认值	取值范围
minFaceSize	最小人脸阈值	200	
notFaceValue	非人脸阈值	0.6f	0~1.0f
brightnessValue	图片最小光照阈值	宽松：30、正常：40、严格：60	0-255f
brightnessMaxValue	图片最大光照阈值	宽松：240、正常：220、严格：200	0-255f
blurnessValue	图像模糊阈值	宽松：0.8f、正常：0.6f、严格：0.4f	0~1.0f
occlusionLeftEyeValue	左眼遮挡阈值	宽松：0.95f、正常：0.8f、严格：0.4f	0~1.0f
occlusionRightEyeValue	右眼遮挡阈值	宽松：0.95f、正常：0.8f、严格：0.4f	0~1.0f
occlusionNoseValue	鼻子遮挡阈值	宽松：0.95f、正常：0.8f、严格：0.4f	0~1.0f
occlusionMouthValue	嘴巴遮挡阈值	宽松：0.95f、正常：0.8f、严格：0.4f	0~1.0f
occlusionLeftContourValue	左脸颊遮挡阈值	宽松：0.95f、正常：0.8f、严格：0.4f	0~1.0f
occlusionRightContourValue	右脸颊遮挡阈值	宽松：0.95f、正常：0.8f、严格：0.4f	0~1.0f
occlusionChinValue	下巴遮挡阈值	宽松：0.95f、正常：0.8f、严格：0.4f	0~1.0f
headPitchValue	低头抬头角度	宽松：30、正常：20、严格：15	0~45
headYawValue	左右摇头角度	宽松：30、正常：20、严格：15	0~45
headRollValue	偏头角度	宽松：30、正常：20、严格：15	0~45
eyeClosedValue	闭眼阈值	0.7f	0~1.0f
cacheImageNum	图片缓存数量	3	建议3~6

3.4 界面定制说明

3.4.1 修改faceplatform_ui界面

（1）如果SDK自带的UI和您的APP不统一，您可以自行修改FaceDetectRoundView。



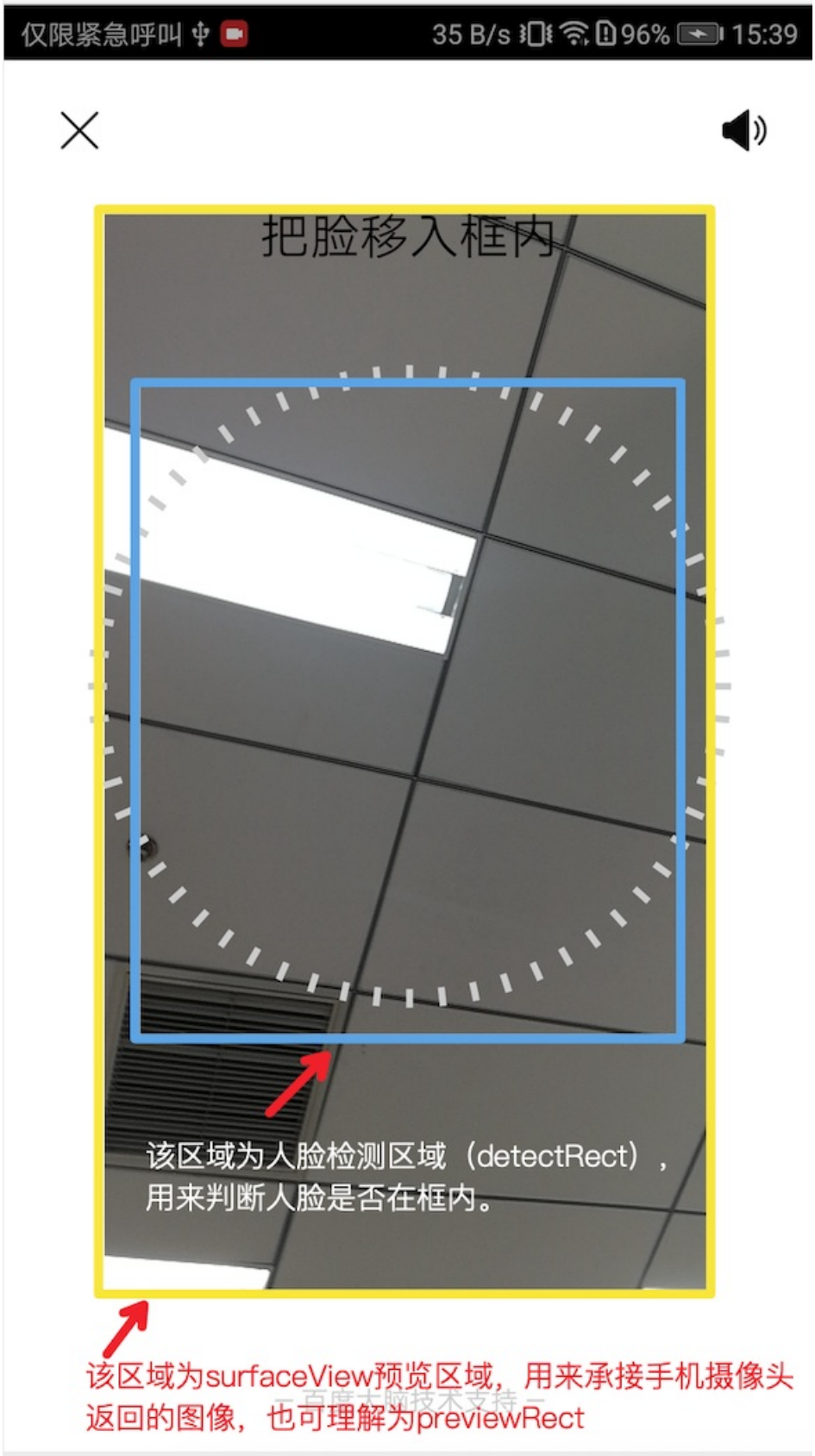
（2）修改提示语音音频文件，有两种方式。

a、直接替换FaceUI工程raw下的mp3文件(faceplatform-ui->src->main->res->raw->.mp3)和string.xml(faceplatform-ui->src->main->res->values->strings.xml)。

b、FaceEnvironment 提供了setSoundId(FaceStatusNewEnum status, int soundId); 设置提示音资源。 FaceStatusNewEnum为不同的状态。soundId为资源文件所对应的resource id。

3.5 采集中的预览区域和检测区域说明

如下图所示（图中背景设置为透明了，可以看出这两个区域）：



其中： surfaceView宽高为屏幕的宽高； previewRect的宽高为手机摄像头的宽高； detectRect的宽高是以previewRect为基准计算出来的。 计算方式参照FaceDetectRoundView.java的Rect getPreviewDetectRect(int w, int pw, int ph)方法

4.接口设计说明

4.1 人脸功能管理器

4.1.1创建实例

- 方法

```
FaceSDKManager getInstance()
```

- 参数

```
无
```

- 返回

```
人脸功能管理器
```

- 说明

```
创建人脸功能管理器
```

4.1.2 人脸功能管理器初始化

- 方法

```
public void initialize(final Context context, String licenseID, String licenseFileName, IInitCallback callback)
```

- 参数

```
context 上下文环境，licenseID 传入申请License时获取的应用名称+_face_android后缀，licenseFileName 鉴权文件名称， callback 鉴权成功与否回调
```

- 返回

```
无
```

- 说明

```
初始化人脸检测功能。进行人脸检测功能License鉴权验证。
```

4.1.3 设置人脸功能控制参数

- 方法

```
void setFaceConfig(FaceConfig config)
```

- 参数

```
config 人脸功能控制参数对象
```

- 返回

```
无
```

- 说明

  设置人脸功能控制参数对象。

  FaceConfig对象参数：

  最小光照阈值

  最大光照阈值

  模糊阈值

  遮挡阈值

  头部姿态角度阈值

  最小人脸检测阈值

  非人脸阈值

  人脸抠图宽高设置

  进行活体检测的动作类型列表

  超时时间设置

  检测框远近比率设置等

4.1.4 取得人脸图像采集功能接口

- 方法

```
IDetectStrategy getDetectStrategyModule()
```

- 参数

  无

- 返回

  人脸图像采集功能接口

- 说明

  取得人脸图像采集功能接口。人脸图像采集接口完成，解析图片人脸信息，返回检测结果。

4.1.5 取得活体检测功能接口

- 方法

```
ILivenessStrategy getLivenessStrategyModule()
```

- 参数

  无

- 返回

活体检测功能接口

- 说明

取得活体检测功能接口。活体检测功能接口完成，解析图片人脸信息，返回活体检测结果。

4.1.6 内存释放接口

- 方法

```
void release()
```

- 参数

无

- 返回

无

- 说明

主要针对模型的释放，以减少内存

4.2 人脸图像采集器

人脸图像采集器IDetectStrategy，检测图片中人脸信息，返回人脸检测状态，完成人脸图像采集。

4.2.1 设置人脸图像采集功能参数

- 方法

```
void setDetectStrategyConfig(Rect previewRect, Rect detectRect,IDetectStrategyCallback callback)
```

- 参数

previewRect 人脸图片大小，类型：Rect

detectRect 人脸检测区域大小，类型：Rect

callback 人脸图像采集功能状态监听器

- 返回

无

- 说明

设置人脸功能控制参数对象。

4.2.2 人脸图像采集

- 方法

```
void detectStrategy(byte[] imageData)
```

- 参数

  imageData 图片信息

- 返回

  无

- 说明

  检测图片中的人脸信息，完成人脸图像采集，返回检测状态和结果。

4.3 活体检测器

活体检测器LivenessStrategy，检测图片人脸信息，活体检测结果状态。

4.3.1 设置人脸功能控制参数

- 方法

```
void setLivenessStrategyConfig(
        List<LivenessTypeEnum> livenessList,
        Rect previewRect,
        Rect detectRect,
        ILivenessStrategyCallback callback);
```

- 参数

  livenessList 活体动作列表

  previewRect 人脸图片大小，类型：Rect

  detectRect 人脸检测区域大小，类型：Rect

  callback 人脸图像采集功能状态监听器

- 返回

  无

- 说明

  设置活体检测功能参数对象。

4.3.2 活体检测

- 方法

  ```
  void livenessStrategy(byte[] imageData);
  ```

- 参数

  imageData 图片信息

- 返回

  无

- 说明

  检测图片中的人脸信息，完成人脸图像采集和人脸活体检测，返回检测状态和结果。

4.4 人脸图像采集界面

人脸图像采集器界面FaceDetectActivity，包括UI界面，系统相机控制，使用人脸图像采集器IDetectStrategy**处理相机采集到的图像。**

4.5 活体检测界面

活体检测界面FaceLivenessActivity，包括UI界面，系统相机控制，使用活体检测器ILivenessStrategy**处理相机采集到的图像，完成活体检测功能。**

4.6 图片加密请求类

图片加密请求类SecRequest，里面的sendMessage方法是为了测试图片加密之后secBase64是否可以通过云端解密的一个示例代码。

5.常见问题

（1）**license**文件有什么用，该放在什么地方？

license文件需要申请，目的是作为sdk校验开发者的使用合法性，license文件放置位置不对或未放置license文件会导致没法使用sdk，一般应先申请license文件，并把申请得到的license文件，放置在assets目录下面。 授权延期后要重新替换到SDK中，否则会因为授权过期导致无法使用。

（2）鉴权初始化的错误码对应的信息

ErrorCode	常量值	说明
SUCCESS	0	成功
LICENSE_NOT_INIT_ERROR	1	license未初始化
LICENSE_DECRYPT_ERROR	2	license数据解密失败
LICENSE_INFO_FORMAT_ERROR	3	license数据格式错误
LICENSE_KEY_CHECK_ERROR	4	license-key(api-key)校验错误
LICENSE_ALGORITHM_CHECK_ERROR	5	算法ID校验错误
LICENSE_MD5_CHECK_ERROR	6	MD5校验错误
LICENSE_DEVICE_ID_CHECK_ERROR	7	设备ID校验错误
LICENSE_PACKAGE_NAME_CHECK_ERROR	8	包名(应用名)校验错误
LICENSE_EXPIRED_TIME_CHECK_ERROR	9	过期时间不正确
LICENSE_FUNCTION_CHECK_ERROR	10	功能未授权
LICENSE_TIME_EXPIRED	11	授权已过期
LICENSE_LOCAL_FILE_ERROR	12	本地文件读取失败
LICENSE_REMOTE_DATA_ERROR	13	远程数据拉取失败
LICENSE_LOCAL_TIME_ERROR	14	本地时间校验错误
OTHER_ERROR	0xff	其他错误

（3）**FaceSDK-LicenseLICENSE_INFO_CHECK_ERROR =license id**

licenseID为您申请时填appname+“_face_android”。如下图demo-turnstile-face-android为license里面的licenseID, **demo-turnstile-face-android1**为app运行时Config.licenseID，两者必须一致

```
E/FaceSDK: FaceSDK-License LICENSE_INFO_CHECK_ERROR =license id  demo-turnstile-face-android  demo-turnstile-face-android1
```

（4）**FaceSDK-LicenseLICENSE_INFO_CHECK_ERROR =signature md5**

md5不一致错误，签名的为license里面的md5，后面的为app运行时获取的签名文件的md5，这两个md5必须一致且区分大小写。 `E/FaceSDK: FaceSDK-License LICENSE_INFO_CHECK_ERROR =signature md5  F5846C60804CC6042D55D09F1AB82364  4357A3EDBC0CA02EA8B5E0578E58D`

（5）**FaceSDK-License LICENSE_INFO_CHECK_ERROR =package name**

PackageName不一致错误。License里面的packagename为申请license时填的，需要保证和app里面的packagename一致。

（6）活体检测常见有那些动作？是否可配置？

常见有6个动作，眨眼，张大嘴，向上抬头，向下低头，向左摇头，向右摇头等。 sdk提供FaceConfig参数设置类，如活体检测角度、光线，检测动作，检测动作数量等设置。

（7）使用sdk一般会用到活体检测拍照等功能，有什么需要注意？

Android 6.0+，需要注意相机拍摄权限问题。如没申请权限，可能导致没法调起相机。

（8）在有些机型上出现特别卡或出现无响应？

SDK在**armeabi**上性能非常差，建议删掉其他so只留下armeabi-v7a，包括使用的其他第三方so。因为如果其他so有armeabi，根据android系统查找so的逻辑，在armeabi的机型上只会去该目录下查找so，而人脸SDK没有，就会出现找不到so。

（9）**license** 文件失效了，不能用了怎么办？

license文件申请时候有期限，如过期会导致校验失效，需要在后台申请延期。

IOS-SDK4.1.5

如果您的需求是与人脸比对、人脸实名认证、在线图片活体检测API搭配使用，请**创建APP方案**获取最新5.2版本SDK及示例工程。

人脸实名认证方案-iOS版本
人脸实名认证方案-Android

1.简介

1.1 功能介绍

百度人脸离线采集SDK IOS 版是一种面向 IOS 移动设备人脸技术开发包，此版SDK包含人脸检测、活体识别等功能。基于该方案，开发者可以轻松的构建包含人脸检测、采集和活体识别的应用。在您使用SDK之前，我们首先为您介绍以下SDK本身及相关人脸能力，以便您能更方便使用人脸服务。

此版SDK所包含的能力如下：

- **离线动作活体检测**：通过让用户做出指定人脸配合式的交互动作，识别当前操作者是否为活体，此功能为离线使用，可设定指定动作是否使用及应用顺序。动作包含：眨眼、张嘴、左摇头，右摇头，向上抬头，向下低头六个。可有效抵御高清图片、3D建模、视频等攻击。
- **离线人脸质量检测**：判断视频流中的图片帧中，哪些图片质量较佳，即人脸图像特征清晰（满足遮挡、姿态、光照、模糊度等校验）。
- **离线人脸图像采集**：通过本地SDK能力，采集人脸图像，同时经过人脸质量检测，确保采集到的人脸图像符合各条件校验（遮挡、姿态、光照、模糊度等），为设备前端获取有效可分析人脸的主要功能。
- **离线授权**：SDK的授权判断，授权介质也称为license，在SDK使用中，优先验证本地离线鉴权文件，验证通过可离线使用；在本地鉴权失败情况下需要向授权服务器发起请求，远程拉取授权进行验证。

1.2 兼容性

- **系统**：支持iOS11以上系统。需要开发者通过Deployment Target 来保证支持系统的检测。
- **机型**：手机和平板皆可（暂不支持横屏）
- **架构**：arm64、armv7
- **网络**：支持 WIFI 及移动网络,移动网络支持使用 NET 网关及WAP 网关(CMWAP、CTWAP、UNIWAP、3GWAP)。

1.3 开发包说明

文件/文件夹名	说明
FaceSDK	FaceSDK 包、bundle 资源文件、bundle 模型文件、鉴权文件
Public/Common	视频流处理类
Public/Utils	图像处理类
UI/Controller	DEMO工程
UI/View	View控制类
FaceParameterConfig.h	配置信息

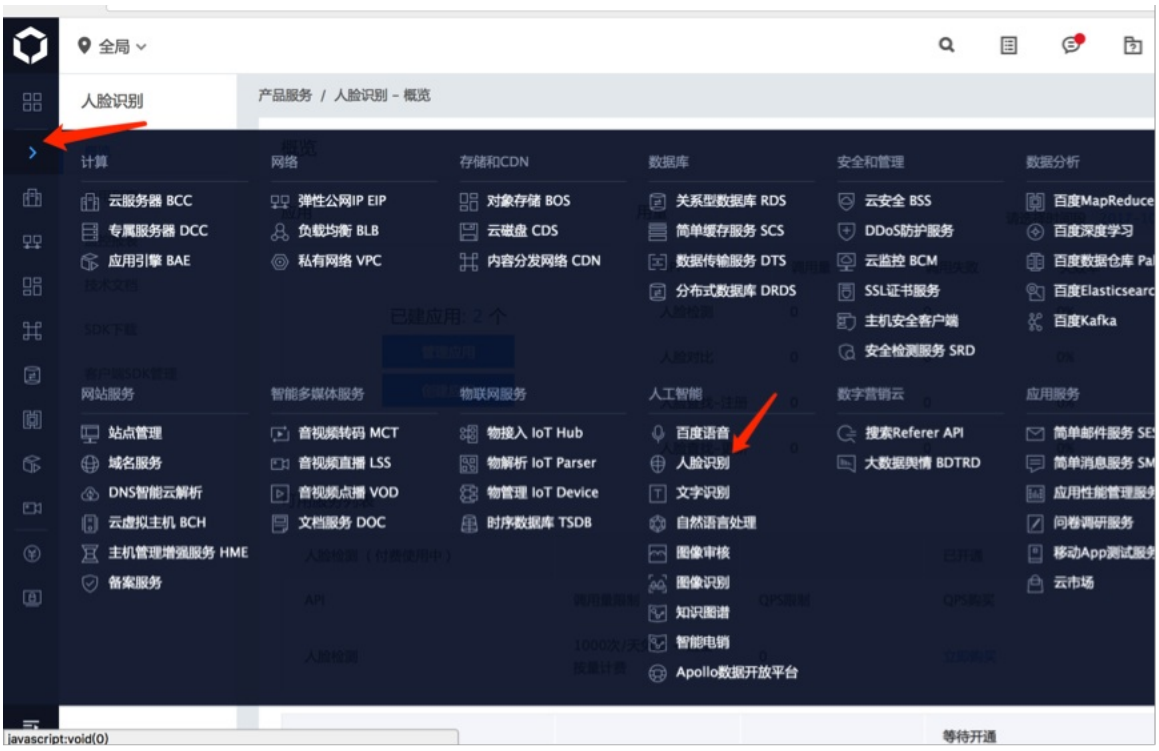
2.集成指南

2.1 准备工作

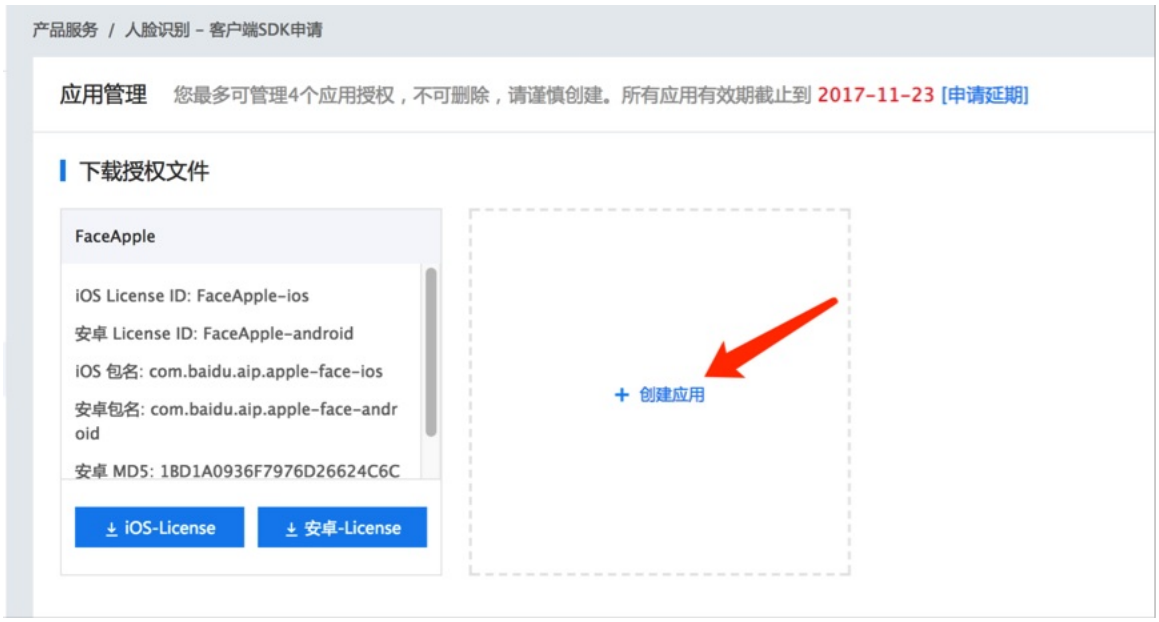
2.1.1 申请license

人脸SDK License：此license用于SDK离线功能使用，在您的申请人脸SDK的后台页面，全局->产品服务->人脸识别->客户端SDK申请

人脸控制台路径如下：



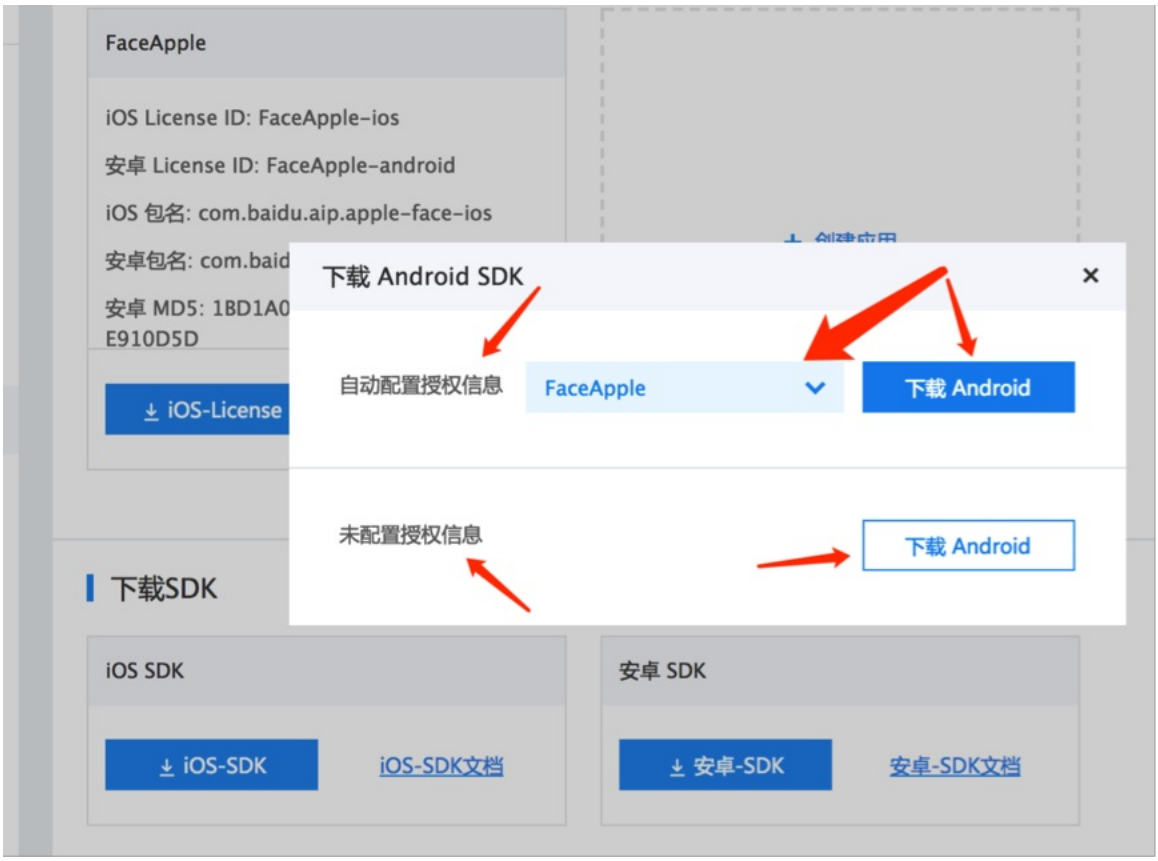
点击客户端SDK管理，弹出如下图：创建应用（这里创建应用是为了使用离线SDK，上面创建应用为了使用人脸在线接口，如注册、识别等）



在弹出的框中输入授权标识，选择应用类型，应用系统，以及包名，详情请查看输入框右边提示

2.1.2 下载SDK

下载SDK分为自动配置授权信息（创建license后就可以选择为该应用，下载后SDK自动帮您配置授权，不用下载license拷贝到工程中，初始化参数licenseID,包名也帮您配置好了）和未配置授权信息两种方式：



2.2 运行示例工程

2.2.1 自动配置授权信息集成

如果您是通过自动配置授权信息下载的示例工程，只需配置好证书即可。查看下项目中的FaceParameterConfig.h文件，已经自动配置

```
11
12 // 人脸license文件名
13 #define FACE_LICENSE_NAME    @"idl-license"
14
15 // 人脸license后缀
16 #define FACE_LICENSE_SUFFIX  @"face-ios"
17
18 //  (您申请的应用名称(appname)+「-face-ios」后缀，如申请的应用名称(appname)为test123，则此处填写test123-face-ios)
19 #define FACE_LICENSE_ID        @"BaiduFaceDemo-face-ios"
20
```

示例图

配置好证书，即可运行。注意已经设置好的bundle id不要随意改动。

2.2.2 未使用自动配置授权信息的集成

手动导入授权信息。 需要手动导入license文件，配置好bundleID、licenseID等信息:

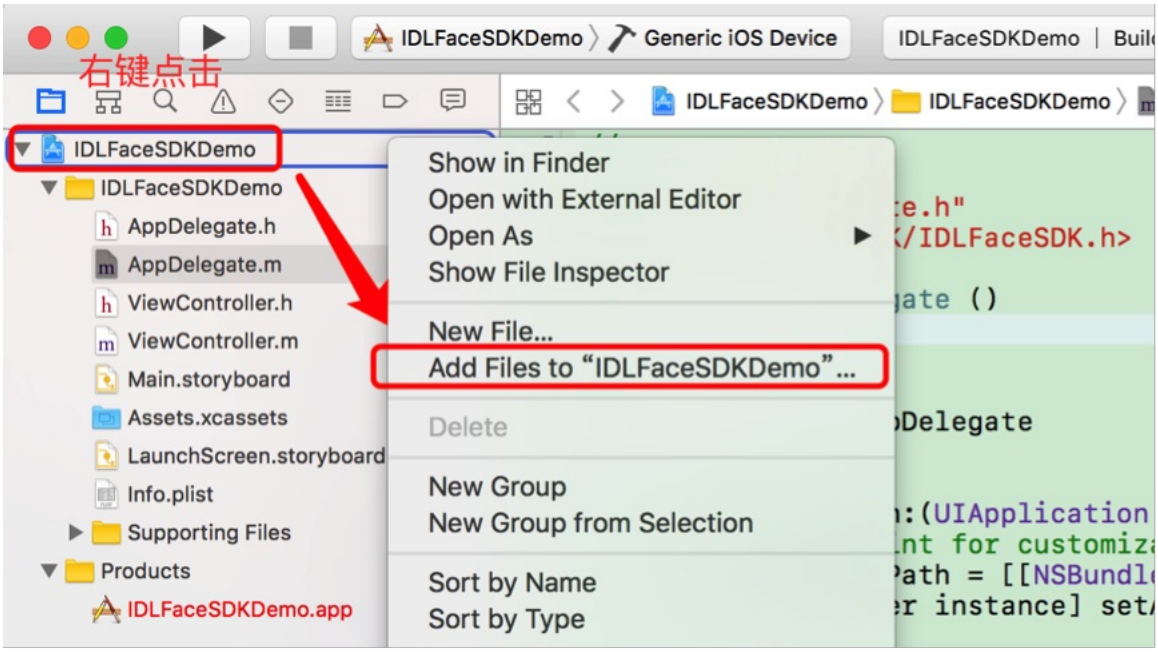
```
11
12 // 人脸license文件名
13 #define FACE_LICENSE_NAME    @"idl-license"
14
15 // 人脸license后缀
16 #define FACE_LICENSE_SUFFIX  @"face-ios"
17
18 //  (您申请的应用名称(appname)+「-face-ios」后缀，如申请的应用名称(appname)为test123，则此处填写test123-face-ios)
19 #define FACE_LICENSE_ID        @"BaiduFaceDemo-face-ios"
20
```

示例图

2.3 添加SDK到工程

1. 打开或者新建一个项目。

2. 右键点击项目，会出现一个添加菜单，在菜单中选择『Add Files to“此处是你的项目名字”…… 』,如下图所示：

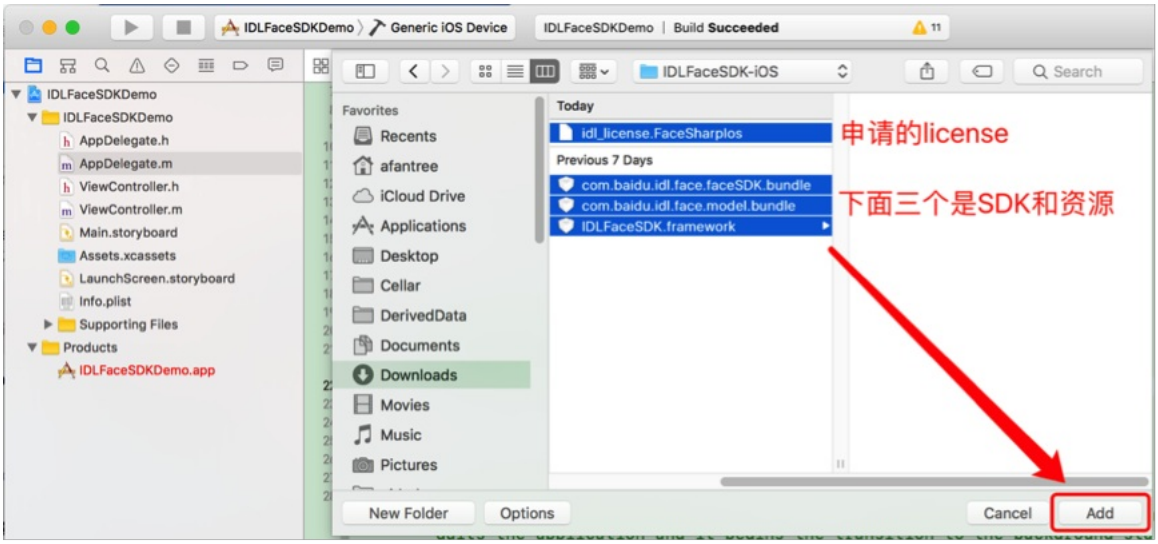


3. 在添加文件弹出框里面选择申请到的license和SDK添加进来。如下图：

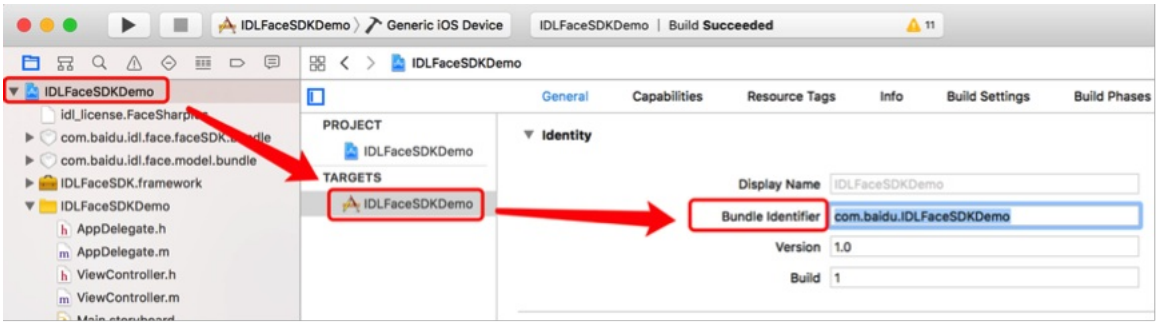
   注意：license为百度官方提供的。

   SDK包含下面三个文件:

   - IDLFaceSDK.framework
   - com.baidu.idl.face.faceSDK.bundle
   - com.baidu.idl.face.model.faceSDK.bundle



4. 确认下Bundle Identifier 是否是申请license时填报的那一个，注意：license和Bundle Identifier是一一对应关系，填错了会导致SDK不能用。



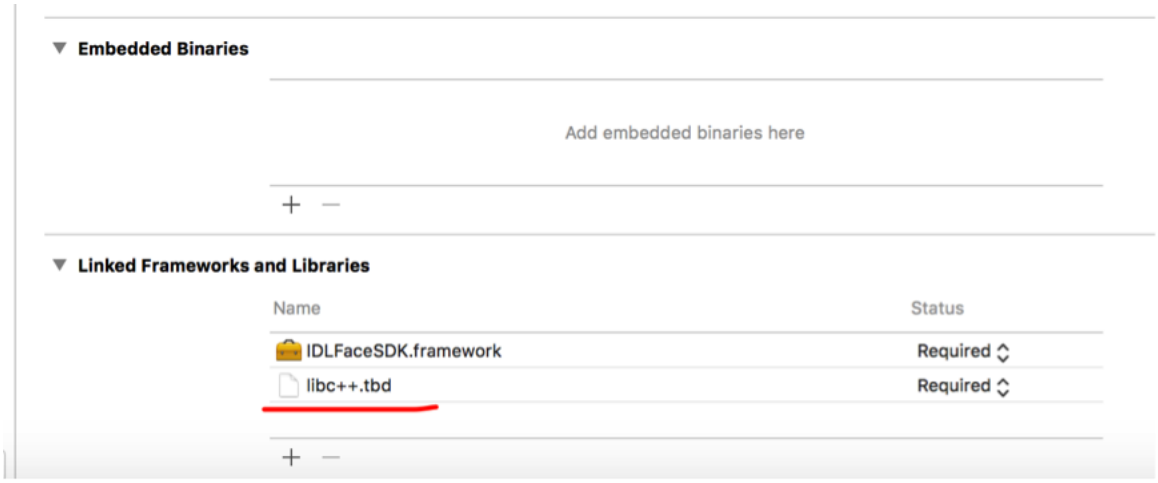
5. 填写正确的FACE_LICENSE_ID。

（即后台展示的LicenseID）

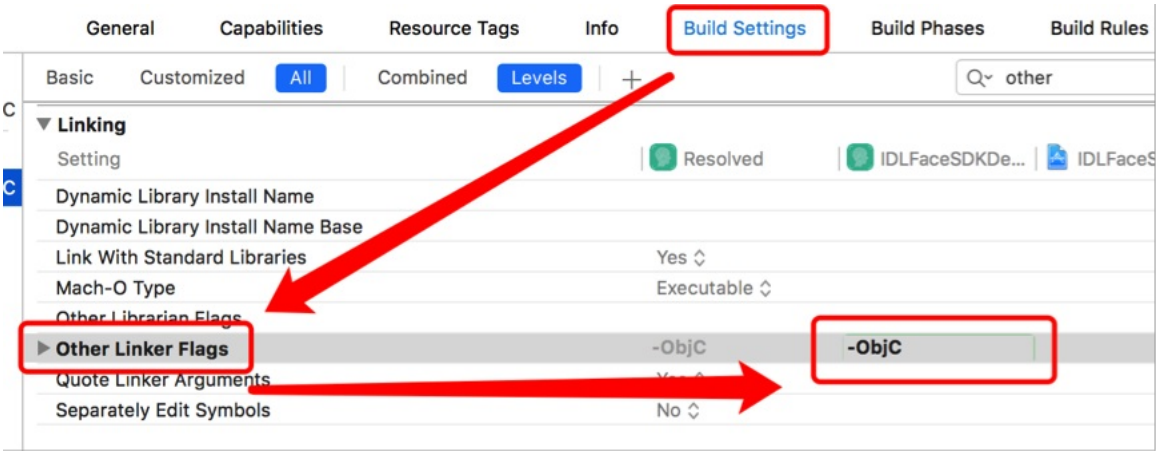
在FaceParameterConfig.h文件里面填写拼接好的FACE_LICENSE_ID。

```
//  (您申请的应用名称(appname)+「-face-ios」后缀，如申请的应用名称(appname)为test123，则此处填写test123-face-ios)
#define FACE_LICENSE_ID        @"BaiduFaceDemo-face-ios"
```

6. 选择链接C++标准库。



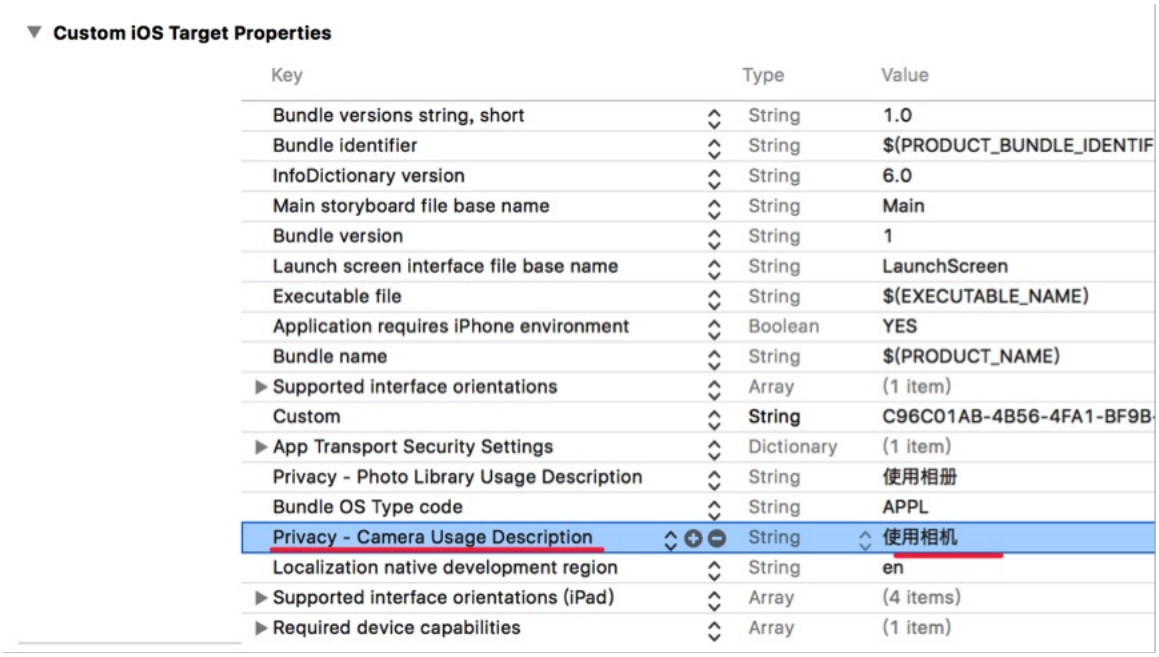
7. 如果没有使用pod管理第三方库的话，请在Build Setting >Linking > Other Linker Flags 上面加入 –ObjC 选项。如果用了pod请忽略，因为pod会自动添加上。



2.4 权限声明

需要使用相机权限：编辑Info.plist文件，添加

Privacy- Camera Usage Description 的Key值，Value为使用相机时候的提示语，可以填写：『使用相机』。



3.接口说明

3.1 FaceSDK 鉴权初始化

3.1.1 设置鉴权功能

```
 - (void)setLicenseID:(NSString *)licenseID andLocalLicenceFile:(NSString *)licensePath andRemoteAuthorize:(BOOL)remoteAuthorize;
```

参数：

- licenseId：平台申请的 licenseID
- localLicencePath：鉴权文件路径
- remoteAuthorize：是否开启网络鉴权

返回：

- 无

参考AppDelegate.m 实现，使用方法如下：

```
    NSString* licensePath = [[NSBundle mainBundle] pathForResource:FACE_LICENSE_NAME ofType:FACE_LICENSE_SUFFIX];
    NSAssert([[NSFileManager defaultManager] fileExistsAtPath:licensePath], @"license文件路径不对，请仔细查看文档");
    [[FaceSDKManager sharedInstance] setLicenseID:FACE_LICENSE_ID andLocalLicenceFile:licensePath andRemoteAuthorize:true];
```

3.1.2 鉴权成功的凭证

```
 - (BOOL)canWork
```

参数：

- 无

返回：

- True代表成功，false代表失败

参考ViewController.m 实现，判断鉴权是否通过：

```
 if (![[FaceSDKManager sharedInstance] canWork]){
  NSLog(@"授权失败，请检测ID 和 授权文件是否可用");
  return;
}
```

3.2 FaceSDK 功能初始化

3.2.1 FaceSDK 参数配置

具体方法详见如下：

```
 /**
 * 设置预测库耗能模式
 * 默认 LITE_POWER_NO_BIND
 */
- (void)setLitePower:(int)litePower;

/**
 * 需要检测的最大人脸数目
 * 默认1
 */
- (void)setMaxDetectNum:(int)detectNum；

/**
 * 需要检测的最小人脸大小
 * 默认40
 */
- (void)setMinFaceSize:(int)width;

/**
 * 人脸置信度阈值（检测分值大于该阈值认为是人脸
 * RGB
 * 默认 0.5f
 */
- (void)setNotFaceThreshold:(CGFloat)thr；

/**
 * 质量检测遮挡阈值
 * 默认0.5
 */
- (void)setOccluThreshold:(CGFloat)thr；

/**
 * 质量检测光照阈值
 * 默认100
 */
- (void)setIllumThreshold:(CGFloat)thr；

/**
 * 质量检测模糊阈值
 * 默认0.5
 */
- (void)setBlurThreshold:(CGFloat)thr；

/**
 * 姿态检测阈值
 * 默认pitch=12，yaw=12，row=10
 */
- (void)setEulurAngleThrPitch:(float)pitch yaw:(float)yaw roll:(float)roll；
```

```
/**
 * 输出图像个数
 * 默认3
 */
- (void)setMaxCropImageNum:(int)imageNum；

/**
 * 输出图像宽，设置为有效值(大于0)则对图像进行缩放，否则输出原图抠图结果
 * 默认 480
 */
- (void)setCropFaceSizeWidth:(CGFloat)width；

/**
 * 输出图像高，设置为有效值(大于0)则对图像进行缩放，否则输出原图抠图结果
 * 默认 680
 */
- (void)setCropFaceSizeHeight:(CGFloat)height；

/**
 * 输出图像，下巴扩展，大于等于0，0：不进行扩展
 * 默认0.1
 */
- (void)setCropChinExtend:(CGFloat)chinExtend；

/**
 * 输出图像，额头扩展，大于等于0，0：不进行扩展
 * 默认0.2
 */
- (void)setCropForeheadExtend:(CGFloat)foreheadExtend；

/**
 * 输出图像，人脸框与背景比例，大于等于1，1：不进行扩展
 * 默认1.5f
 */
- (void)setCropEnlargeRatio:(float)cropEnlargeRatio;

/**
 * 动作超时配置
 */
- (void)setConditionTimeout:(CGFloat)timeout；

/**
 * 语音间隔提醒配置
 */
- (void)setIntervalOfVoiceRemind:(CGFloat)timeout;

/**
 * 是否开启静默活体，默认false
 */
- (void)setIsCheckSilentLive:(BOOL)isCheck;

/**
 * 设置原始图片缩放比例，默认1不缩放，scale 阈值0~1
 */
- (void)setImageWithScale:(CGFloat)scale;

/**
 * 设置图片加密类型，type=0 基于base64 加密；type=1 基于百度安全算法加密
 */
- (void)setImageEncrypteWithType:(int) type;
```

```
/**
 *  人脸过远框比例 默认：0.4
 */
- (void)setMinRect:(float) minRectScale;
```

3.2.2 FaceSDK 初始化

```
 - (int)initCollect;
```

参数:

- 无

返回：

- 无

参考ViewController.m initSDK 方法实现：

```
 - (void) initSDK {

    if (![[FaceSDKManager sharedInstance] canWork]){
        NSLog(@"授权失败，请检测ID 和 授权文件是否可用");
        return;
    }

    // 初始化SDK配置参数，可使用默认配置
    // 设置最小检测人脸阈值
    [[FaceSDKManager sharedInstance] setMinFaceSize:200];
    // 设置截取人脸图片高
    [[FaceSDKManager sharedInstance] setCropFaceSizeWidth:480];
    // 设置截取人脸图片宽
    [[FaceSDKManager sharedInstance] setCropFaceSizeHeight:640];
    // 设置人脸遮挡阈值
    [[FaceSDKManager sharedInstance] setOccluThreshold:0.5];
    // 设置亮度阈值
    [[FaceSDKManager sharedInstance] setIllumThreshold:40];
    // 设置图像模糊阈值
    [[FaceSDKManager sharedInstance] setBlurThreshold:0.3];
    // 设置头部姿态角度
    [[FaceSDKManager sharedInstance] setEulurAngleThrPitch:10 yaw:10 roll:10];
    // 设置人脸检测精度阈值
    [[FaceSDKManager sharedInstance] setNotFaceThreshold:0.6];
    // 设置抠图的缩放倍数
    [[FaceSDKManager sharedInstance] setCropEnlargeRatio:2.5];
    // 设置照片采集张数
    [[FaceSDKManager sharedInstance] setMaxCropImageNum:3];
    // 设置超时时间
    [[FaceSDKManager sharedInstance] setConditionTimeout:1500];
    // 设置原始图缩放比例
    [[FaceSDKManager sharedInstance] setImageWithScale:0.8f];
    // 设置图片加密类型，type=0 基于base64 加密；type=1 基于百度安全算法加密
    [[FaceSDKManager sharedInstance] setImageEncrypteType:0];
    // 设置人脸过远框比例
    [[FaceSDKManager sharedInstance] setMinRect:0.4];
    // 初始化SDK功能函数
    [[FaceSDKManager sharedInstance] initCollect];
}
```

3.2.3 FaceSDK 释放

```
- (int)uninitCollect
```

参数:

- 无

返回：

- 无

3.3 人脸采集

调用IDLFaceDetectionManager类的:

```
/**
* 人脸采集，成功之后返回扣图图片，原始图片
* @param image 镜头拿到的图片
* @param previewRect 预览的Rect
* @param detectRect 检测的Rect
* return completion 回调信息
*/
- (void)detectStratrgyWithNormalImage:(UIImage *)image previewRect:(CGRect)previewRect detectRect:(CGRect)detectRect completionHandler:(DetectStrategyCompletion)completion;

typedef void (^DetectStrategyCompletion) (FaceInfo * faceinfo,NSDictionary * images, DetectRemindCode remindCode);
```

温馨提示：NSDictionary * images 返回FaceCropImageInfo 对象包含带黑边的图片，主要是为了配合在线 API 方式的活体检测使用，本地算法会对图片做一定的预处理操作；而不带黑边的图片，如果无需使用活体，则可使用不带黑边的图片采集方式。

参数说明：

`previewRect`与`detctRect`是为了做距离检测而定义的，为了在上层封装判断脸是否在框内／离太远／离太近。

- image：相机获取的图片
- previewRect： 人脸图片大小，间接定义的最大距离的 maxRect 和最小距离的 minRect。
- detectRect：人脸检测区域大小，实际采集区域
- completion：完成后返回照片和状态结果

参考BDFaceDetectionViewController.m 实现，使用方法如下：

```
        __weak typeof(self) weakSelf = self;
    [[IDLFaceDetectionManager sharedInstance] detectStratrgyWithNormalImage:image previewRect:self.previewRect
detectRect:self.detectRect completionHandler:^(FaceInfo *faceinfo, NSDictionary *images, DetectRemindCode
remindCode) {
        switch (remindCode) {
          case DetectRemindCodeOK: {
              weakSelf.hasFinished = YES;
              [self warningStatus:CommonStatus warning:@"非常好"];
              if (images[@"image"] != nil && [images[@"image"] count] != 0) {

                  NSArray *imageArr = images[@"image"];
                  for (FaceCropImageInfo * image in imageArr) {
                      NSLog(@"cropImageWithBlack %f %f", image.cropImageWithBlack.size.height,
image.cropImageWithBlack.size.width);
                      NSLog(@"originalImage %f %f", image.originalImage.size.height, image.originalImage.size.width);
                  }

                  FaceCropImageInfo * bestImage = imageArr[0];
                  [[BDFaceImageShow sharedInstance] setSuccessImage:bestImage.originalImage];
                  [[BDFaceImageShow sharedInstance] setSilentliveScore:bestImage.silentliveScore];
                  // 公安验证接口测试
                  [self request:bestImage.cropImageWithBlackEncryptStr];
                  dispatch_async(dispatch_get_main_queue(), ^{
                      UIViewController* fatherViewController = weakSelf.presentingViewController;
                      [weakSelf dismissViewControllerAnimated:YES completion:^{
                          BDFaceSuccessViewController *avc = [[BDFaceSuccessViewController alloc] init];

                          [fatherViewController presentViewController:avc animated:YES completion:nil];
                      }];

                  });
              }
              [self singleActionSuccess:true];
              break;
```

3.4 动作采集

调用IDLFaceLivenessManager类的:

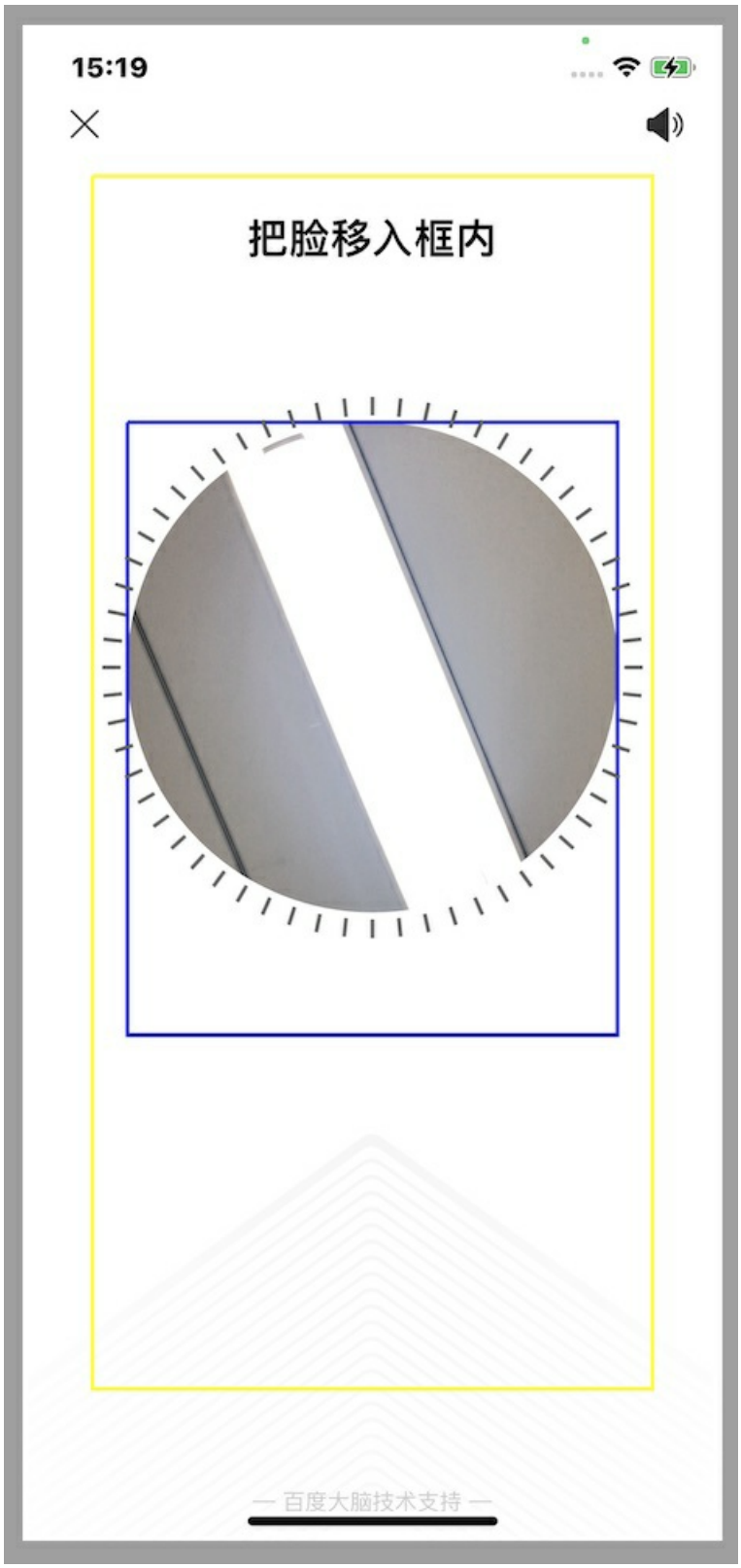
```
 /**
 *  人脸活体验证，成功之后扣图图片，原始图片三种
 * @param image 镜头拿到的图片
 * @param previewRect 预览的Rect
 * @param detectRect 检测的Rect
 * return completion 回调信息
 */
-(void) livenessNormalWithImage:(UIImage *)image previewRect:(CGRect)previewRect detectRect:(CGRect)detectRect
completionHandler:(LivenessNormalCompletion)completion;

typedef void (^LivenessNormalCompletion) (NSDictionary * images, FaceInfo *faceInfo, LivenessRemindCode
remindCode);
```

温馨提示：NSDictionary * images 返回FaceCropImageInfo 对象包含带黑边的图片，主要是为了配合在线 API 方式的活体检测使用，本地算法会对图片做一定的预处理操作；而不带黑边的图片，如果无需使用活体，则可使用不带黑边的图片采集方式。

参数说明：

`previewRect`与`detctRect`是为了做距离检测而定义的，为了在上层封装判断脸是否在框内／离太远／离太近。

- image：相机获取的图片
- previewRect： 人脸图片大小(下图黄色框)
- detectRect：人脸检测区域大小，实际采集区域，间接定义的最大距离的 maxRect 和最小距离的 minRect。（下图蓝色框）
- completion：完成后返回照片和状态结果



说明：

- 检测图片中的人脸信息，完成人脸图像采集和人脸活体检测，返回检测状态和结果。

```
    [[IDLFaceLivenessManager sharedInstance] livenessNormalWithImage:image previewRect:self.previewRect
detectRect:self.detectRect completionHandler:^(NSDictionary *images, FaceInfo *faceInfo, LivenessRemindCode
remindCode) {

        switch (remindCode) {
            case LivenessRemindCodeOK: {
                weakSelf.hasFinished = YES;
                [self warningStatus:CommonStatus warning:@"非常好"];
                if (images[@"image"] != nil && [images[@"image"] count] != 0) {

                    NSArray *imageArr = images[@"image"];
                    for (FaceCropImageInfo * image in imageArr) {
                        NSLog(@"cropImageWithBlack %f %f", image.cropImageWithBlack.size.height,
image.cropImageWithBlack.size.width);
                        NSLog(@"originalImage %f %f", image.originalImage.size.height, image.originalImage.size.width);
                    }

                    FaceCropImageInfo * bestImage = imageArr[0];
```

3.5 活体动作设置

调用IDLFaceLivenessManager类的:

```
- (void)livenesswithList:(NSArray *)array order:(BOOL)ordernumberOfLiveness:(NSInteger)numberOfLiveness
```

参数：

- array: 活体动作列表
- order: 是否按顺序进行活体动作
- numberOfLiveness: 活体动作数目（array为nil是起作用）

返回：

- 无

说明：

- 活体动作设置

4.常见问题

Q：鉴权问题。提示「验证失败」 A：先确定网络情况是否正常，本地鉴权文件失效了才走网络鉴权。定位错误码，排查鉴权失败的原因。一般是 licenseID 和 bundleID 配置不一致导致的鉴权失败。请注意上线前授权文件一定要更新。

Q：license 文件失效了，不能用了怎么办？ A：License 文件申请时候有期限，如过期会导致校验失效，需要在后台进行申请延期。

Q：使用 iOS 采集端，采集到的图片是斜着的，这个正常吗，会影响识别吗？ A：不会影响识别。有黑边和倾斜是因为图片质量算法造成的，我们是按 1:3 对图像进行背景填充使人脸居中，为的是更好的识别图像。这个版本提供了`detectStratrgyWithQualityControlImage`和`detectStratrgyWithNormalImage`两种方法供选择。

SDK合规使用指南

前言

2023年2月工信部发布的**《工业和信息化部关于进一步提升移动互联网应用服务能力的通知》**对SDK、个人信息保护、服务体验等方面提出了具体要求，同时为了帮助开发者向最终用户提供相应功能及服务，特此起草本SDK合规使用指导。 您作为开发者为最终用户提供服务，需知悉并确认将遵守适用的法律法规和相关的标准规范，履行个人信息保护义务，并遵循合法、正

当、必要和诚信的原则处理用户个人信息，包括但不限于《中华人民共和国个人信息保护法》、《中华人民共和国网络安全法》、《中华人民共和国数据安全法》以及其他适用的法律法规和相关的标准规范。 此文档用于帮助您更好地了解百度人脸离线采集SDK并合规使用本SDK服务，仅适用于开发者的业务区域为中国大陆地区的场景。

特别提示

本**《SDK合规指南》**是对本 SDK 的合规性和安全性描述与要求仅为我们向您提供的服务说明及使用建议，不构成也不应被视为我们对于任何法律法规及政策文件的及时的、完整的、全面的、甚至完全准确的分析，亦不构成我们对百度人脸离线采集SDK产品和/或服务的承诺和保证。本《指南》不能作为判断您与您所开发、运营的 App 是否满足合规与安全要求的可依赖的标准，亦不构成我们对前述事宜作出的任何担保或保证，您应当自行、独立地对前述 App 承担合规与安全责任。

目录

- SDK收集个人信息的频次、精度、使用目的、场景及对应选择的配置方式、示例
- SDK所需系统权限与功能关系，以及权限申请时机
- SDK扩展业务功能介绍及对应关闭的配置方式、示例
- SDK初始化及各项业务功能接口合规调用时机
- 联系我们

1. SDK收集个人信息不同频次、精度使用目的、场景及对应选择的配置方式、示例

为了帮助开发者向最终用户提供相应功能及服务，当最终用户使用相应功能及服务时，我们会通过开发者应用向系统申请最终用户设备的相应权限。开发者应确保最终用户可以随时通过取消系统授权开发者应用获取相应设备权限或其他开发者应用提供的授权设置，停止我们对最终用户个人信息的收集，之后最终用户可能将无法使用基于相应个人信息而提供的相关服务或功能，或者无法达到基于相应个人信息提供的相关服务拟达到的效果，但不会影响最终用户正常使用人脸离线采集SDK的其他不基于相应个人信息即可实现的业务功能。各项功能及服务涉及的个人信息包括：

序号	功能服务	个人信息类型	收集方式	适用系统版本
1.1	为实现设备风险环境检测、恶意应用检测、DNS劫持检测、网络安全检测等安全检测与防护功能，以及识别是否为真实设备及真人使用人脸离线采集SDK	设备信息（包括IMEI、运营商信息、WiFi状态、WiFi参数、WiFi列表、系统设置、系统属性、设备型号、操作系统、设备应用安装列表、正在运行的程序列表）、位置信息（通过IP地址解析获取粗略位置信息）、日志信息（包括设备信息和IP地址）	SDK直接采集	仅Android
1.2	为实现设备风险环境检测、恶意应用检测、DNS劫持检测、网络安全检测等安全检测与防护功能，以及识别是否为真实设备及真人使用人脸离线采集SDK	设备信息（包括IDFA、IDFV、运营商信息、WiFi状态、WiFi参数、系统设置、系统属性、设备型号、操作系统）、****位置信息（通过IP地址解析获取粗略位置信息）、日志信息（包括设备信息和IP地址）	SDK直接采集	仅iOS
2	为帮助开发者向最终用户提供最佳质量人脸图片采集，以进行活体校验及身份核验的功能	人脸图片	SDK直接采集	iOS及Android
3	为帮助开发者向最终用户提供视频录制功能，以便作为人脸图片活体校验的二次复核数据，实现活体校验二次复核功能	人脸视频信息（含语音）	SDK直接采集	iOS及Android

人脸图片、人脸视频信息（含语音）为敏感个人信息，最终用户可以随时通过**【取消系统授权开发者应用获取相应设备权限或其他开发者应用提供的授权设置或手动输入】**停止我们对上述敏感个人信息的收集，之后最终用户可能将无法使用基于上述敏感个人信息而提供的相关服务或功能，或者无法达到基于上述敏感个人信息提供的相关服务拟达到的效果，但不会影响最终用

户正常使用人脸离线采集SDK的其他不基于上述敏感个人信息即可实现的业务功能。 特别提示开发者及最终用户注意：

1. 仅在开发者集成人脸离线采集SDK且调用我们提供的公有云服务时，序号1.1及序号1.2所述信息才会上报至我们的服务器，否则我们不会也无法获取到前述信息。
2. 序号2所述信息将上报至开发者所部署的配合人脸离线采集SDK使用的后端人脸及安全私有化服务的服务器地址，我们不会也无法获取到前述信息。序号3所述信息将在本地加密处理后存储在应用指定路径下，我们不会也无法获取到前述信息，仅开发者可根据存储路径获取。若开发者在调用我们提供的公有云服务过程中委托我们处理最终用户的个人信息，我们将可能会获取相关个人信息，具体的收集、使用规则请参见公有云服务相关服务条款。

注：更多个人信息收集、使用、处理的细节请参考 人脸离线采集SDK隐私政策

2. SDK所需系统权限与功能关系，以及权限申请时机

为了保证最终用户能正常使用人脸离线采集SDK相应功能及服务，我们会通过开发者应用向系统申请最终用户设备的以下系统设备权限，申请前我们会征询最终用户的同意，最终用户可以选择“允许”或“禁止”权限申请。经过最终用户的授权后我们会开启相关权限，最终用户可以随时在系统中取消授权，最终用户取消授权会导致最终用户无法使用相关的业务功能，但不会导致最终用户无法使用其他业务功能。各项功能及功能对设备权限的调用情况如下：

Android系统版本

设备权限	功能及服务	权限授权方式
读取/写入外部存储卡	用于存储人脸离线采集SDK的模型信息、配置文件、以及保存安全风险诊断信息；从相册中读取身份证照片，以便识别身份证号和姓名，以实现权威数据源身份核验功能	授权方式由设备系统开发方以及开发者应用决定；当最终用户同意向开发者应用授予该权限时开启
获取粗略位置	用于采集设备IP地址检验设备环境及网络风险	授权方式由设备系统开发方以及开发者移动应用决定；当最终用户同意向开发者移动应用授予该权限时开启
访问网络	用于连接网络，进行数据传输，用作活体及身份核验	授权方式由设备系统开发方以及开发者产品决定；当最终用户同意向开发者产品授予该权限时开启
获取网络状态	用于获取当前网络信息，进行安全风险诊断，并保护网络传输	授权方式由设备系统开发方以及开发者产品决定；当最终用户同意向开发者产品授予该权限时开启
打开相机/摄像头	实现人脸图片采集、身份证照片采集和人脸视频录制功能，以便进行活体及身份校验	授权方式由设备系统开发方以及开发者产品决定；当最终用户同意向开发者产品授予该权限时开启
麦克风权限	实现人脸视频录制功能（当开启视频录制功能时，需要记录语音），以便进行活体及身份校验	授权方式由设备系统开发方以及开发者产品决定；当最终用户同意向开发者产品授予该权限时开启

iOS系统版本

设备权限	功能服务	权限授权方式
打开相机/摄像头	实现人脸图片采集、身份证照片采集和人脸视频录制功能，以便进行活体及身份校验	授权方式由设备系统开发方以及开发者应用决定；当最终用户同意向开发者应用授予该权限时开启
麦克风权限	实现人脸视频录制功能（当开启视频录制功能时，需要记录语音），以便进行活体及身份校验	授权方式由设备系统开发方以及开发者移动应用决定；当最终用户同意向开发者移动应用授予该权限时开启

注：在不同设备中，权限显示方式及关闭方式可能有所不同，具体请最终用户参考设备及系统开发方说明或指引。

3. 告知最终用户的文案示例

为帮助您明确地告知最终用户百度人脸离线采集SDK个人信息保护规则相关事宜，我们为您提供了以下告知方文案示例，供您参考执行：

文案示例A

为向您提供【人脸采集及核验】功能/服务，我们集成了【百度人脸离线采集】SDK。在为您提供【人脸采集及核验】功能/服务时，【百度人脸离线采集】SDK需要收集、使用您必要的个人信息。关于【百度人脸离线采集】SDK收集、使用的个人信息类型、目的及用途，以及【百度人脸离线采集】SDK将如何保护所收集、使用的个人信息，请您仔细阅读 人脸离线采集SDK隐私政策 了解。

文案示例B

为保障App相关功能的实现与应用安全稳定的运行，我们可能会接入由第三方提供的SDK实现相关目的，具体接入的相关第三方SDK列明如下：

第三方SDK名称	使用目的	官网链接/隐私政策链接
百度人脸离线采集	实现人脸采集及核验	https://ai.baidu.com/ai-doc/REFERENCE/Ul028s5hk

文案示例C

为向您提供【人脸采集及核验】功能/服务，我们集成了【百度人脸离线采集】SDK。在为您提供【人脸采集及核验】功能/服务时，【百度人脸离线采集】SDK收集、使用您的【设备信息如操作系统版本/设备型号、日志信息】信息，用于【人脸采集及核验以及设备风险检测】目的/用途。请您仔细阅读 人脸离线采集SDK隐私政策 了解。

文案示例D

为保障App相关功能的实现与应用安全稳定的运行，我们可能会接入由第三方提供的SDK实现相关目的，具体接入的相关第三方SDK列明如下：

第三方SDK名称	业务功能	个人信息类型	具体目的/用途
百度人脸离线采集	实现人脸采集及核验	设备信息如操作系统版本、设备型号、日志信息	人脸采集及核验以及设备风险检测

4. SDK初始化及各项业务功能接口合规调用时机

为了避免您的应用在未获取用户的同意前SDK提前处理用户的个人信息，我们提供了SDK初始化的接口，请保证您的应用获取用户同意后才能调用此接口初始化SDK。

- iOS：initCollect
- Android：initialize

5. 联系我们

人脸离线采集SDK的成长离不开各方开发者及最终用户的共同努力，我们非常感谢开发者及最终用户对人脸离线采集SDK数据更新、使用反馈方面的贡献。 开发者及最终用户可以通过百度云工单反馈开发者及最终用户对人脸离线采集SDK产品和服务的建议以及在使用过程中遇到的问题，以帮助我们优化产品功能及服务，使更多用户更加便捷的使用我们的产品和服务。