

CFC 文档



【版权声明】

版权所有©百度在线网络技术（北京）有限公司、北京百度网讯科技有限公司。未经本公司书面许可，任何单位和个人不得擅自摘抄、复制、传播本文档内容，否则本公司有权依法追究法律责任。

【商标声明】



和其他百度系商标，均为百度在线网络技术（北京）有限公司、北京百度网讯科技有限公司的商标。本文档涉及的第三方商标，依法由相关权利人所有。未经商标权利人书面许可，不得擅自对其商标进行使用、复制、修改、传播等行为。

【免责声明】

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导。如您购买本文档介绍的产品、服务，您的权利与义务将依据百度智能云产品服务合同条款予以具体约定。本文档内容不作任何明示或暗示的保证。

目录	
目录	2
功能发布记录	8
产品描述	14
什么是函数计算CFC	14
产品优势	15
应用场景	16
核心概念	16
使用限制	16
产品定价	17
计费概述	17
资源套餐包	18
快速入门	21
概述	21
控制台使用示例	22
操作指南	29
开通CFC服务	29
创建函数	30
管理函数	39
管理函数	39
配置并查看函数日志	43
网络管理	44
配置VPC功能	44
函数在VPC中访问外网	45
版本与别名管理	47
层依赖管理	51
层操作指南	51
使用CFC层管理依赖包-Node.js	55
使用CFC层管理依赖包-Python	61
死信队列管理	66
配置死信队列	67
使用示例	67
扩展函数最大超时时间	67
配置最大超时时间	67
前提条件	67
操作步骤	67
服务管理	67
异步调用	70
概述	70
预留实例管理	71
自定义域名	74

Baidu 百度智能云文档	目录
开发者指南	78
概述	78
开发语言	79
Golang	79
Java	83
C#	92
Python	94
Lua	97
PHP	99
Node.js	104
创建部署程序包	107
使用BSAM-CLI调试函数	109
BCE-Serverless-Application-Model规范	109
BSAM-CLI使用示例	112
Baidu Serverless VSCode 插件	117
开发注意事项	120
时间戳与时区	120
函数运行环境及安全隔离性	121
无状态函数	122
冷热启动机制	122
触发器	122
触发器概述	122
名词解释	123
BOS触发器	125
CDN触发器	127
百度消息服务触发器	132
BLS触发器	139
定时任务触发器	143
定时任务触发器	143
定时任务计划表达式	145
HTTP触发器	146
HTTP触发器开发参考	146
HTTP触发器使用示例	151
DuerOS触发器	154
DuEdge触发器	155
百度消息服务触发器使用示例	155
CDN触发器使用示例	160
Rocketmq触发器使用示例	165
工作流	170
概述	170

Baidu 百度智能云文档	目录
应用场景	171
基本概念	171
快速入门	172
流程定义语言	176
基本介绍	176
输入和输出	178
传递 (pass) 节点	184
等待 (wait) 节点	184
判断 (switch) 节点	185
任务 (opreation) 节点	186
成功 (succeed) 节点	187
失败 (fail) 节点	188
并行 (parallel) 节点	189
循环 (foreach) 节点	190
错误处理	193
典型实践	195
百度智能小程序使用云开发构建CFC云函数	195
DuerOS开放平台开发者使用CFC构建一个Custom-Skill	200
配置CFC函数以访问百度云中的RDS	208
CFC函数搭建Flask-Web框架	214
概述	214
使用场景	214
环境准备	214
操作部署	214
应用扩展	215
使用定时触发器根据天气信息控制远程设备	216
概述	216
使用 Terraform 快速搭建图像处理应用	228
使用 CFC HTTP 触发器托管 API	232
Java Springboot Web框架迁移到函数计算	237
CFC支持SSE协议	241
CFC支持MCP服务托管	244
AWS lambda 迁移到 CFC 实践	250
API参考	254
概述	254
通用说明	254
服务域名	256
公共请求头与公共响应头	256
错误返回	257
函数调用接口	259
函数操作	261

Baidu 百度智能云文档	目录
创建函数CreateFunction	261
函数列表ListFunctions	265
函数信息GetFunction	267
删除函数DeleteFunction	268
更新函数代码UpdateFunctionCode	269
获取函数配置GetFunctionConfiguration	271
更新函数配置UpdateFunctionConfiguration	272
设置函数预留并发度	275
删除函数预留并发度	276
版本操作	277
获取函数版本列表ListVersionsByFunction	277
发布版本PublishVersion	279
别名操作	281
别名列表ListAliases	281
创建别名CreateAlias	283
获取别名信息GetAlias	284
更新别名信息UpdateAlias	285
删除别名信息DeleteAlias	287
触发器操作	288
获取触发器列表ListTriggers	288
创建触发器CreateTrigger	289
更新触发器UpdateTrigger	291
删除触发器DeleteTrigger	292
工作流操作	293
开始执行StartExecution	293
停止执行StopExecution	294
获取执行列表ListExecutions	296
获取执行信息DescribeExecution	297
获取执行历史GetExecutionHistory	299
工作流列表ListFlow	302
创建工作流CreateFlow	303
更新工作流UpdateFlow	305
获取工作流信息DescribeFlow	307
删除工作流DeleteFlow	308
数据类型	309
SDK	314
Java SDK	314
概述	314
安装SDK工具包	314
初始化	315
CfcClient	316

Baidu 百度智能云文档	目录
函数操作	318
函数调用	321
版本操作	321
别名操作	322
触发器操作	323
异常处理	324
SDK日志	324
版本变更记录	325
GO-SDK	326
概述	326
安装SDK工具包	326
初始化	327
确认Endpoint	327
函数调用	330
函数操作	331
创建函数	331
版本操作	335
获取函数版本列表	335
别名操作	336
获取别名列表	336
触发器操作	338
获取触发器列表	338
错误处理	339
版本变更记录	340
Python SDK	340
概述	340
初始化	340
安装SDK工具包	341
CfcClient	342
函数操作	344
函数调用	345
别名操作	346
版本操作	346
异常处理	347
触发器操作	347
JavaScript-SDK	348
概述	348
安装SDK工具包	348
初始化	348
函数调用	351
函数操作	351

版本操作	354
别名操作	354
触发器操作	355
版本变更记录	358
视频专区	358
技术讲解	358
常见问题	359
账号问题	359
网络问题	359
资源问题	360
函数相关	360
触发器相关	362
日志相关	363
监控相关	364
计费相关	364
API&SDK&工具	365
其他	365
支持	365
服务条款	365
CFC服务等级协议	366

功能发布记录

发布时间	功能概述
2025-4	CFC支持MCP服务托管
2025-3	CFC支持SSE协议
2023-4	CFC新增nodejs14, nodejs16运行时支持
2023-4	CFC新增APIGateway触发器，支持自定义域名，详见 自定义域名
2023-3	CFC新增小额资源套餐包，新客可享受前三月每月100万次免费调用，老客可根据业务需求选择新的月套餐包和之前的年套餐包，详见 资源套餐包

2022-3	CFC新增资源套餐包，资源套餐包是一种资源抵扣包，可以优先抵扣CFC资源使用量。购买资源套餐包后，您可以以更优惠的价格使用CFC的资源，详见 资源套餐包
2021-12	• CFC新增工作流，详见 工作流
2021-10	• CFC新增异步调用配置，详见 异步调用
2021-10	• CFC新增服务错误次数、最大使用内存等多个监控指标，详见 控制台使用示例
2021-09	• 函数执行日志可配置接入日志服务BLS，详见 配置并查看函数日志
2021-09	• Springboot Web框架迁移到CFC，详见 Springboot框架迁移
2021-08	• 函数新增服务管理功能，详见 服务管理

2021-06	• CFC支持单实例多并发功能，详见 单实例多并发管理参考
2021-03	• CFC集成VSCode插件，详见 CFC集成VSCode插件参考
2021-01	• CFC集成Flask框架，详见 集成Flask框架参考
2020-09	• 扩展函数最大超时时间，详见 扩展函数最大超时时间参考
2020-04	• HTTP 触发器支持后端函数简易格式返回，详见 HTTP 触发器开发参考 • 异步调用函数支持死信队列配置
2020-03	• HTTP 触发器支持将二进制请求体传入用户函数

2019-12	• 新增函数层依赖管理功能，可以在层操作指南中进行配置
2019-11	• 新增函数别名的灰度流量功能，可以在别名管理中进行配置 • 新增支持从用户BOS存储空间上传函数代码的功能
2019-10	• 新增语言运行时 .NET Core 2.2 • 新增函数网络配置能力，可在用户 VPC 内运行函数，可参考网络管理获取更多信息
2019-09	• 全新函数模版功能，用户可在创建函数时从模版中选择代码样例，并在此基础上进行开发。
	• 用户可以为新建的函数或已有函数配置CDN触发器或百度消息服务触发器，指定事件来触发函数的执行 • 为开发者用户提供Python SDK和Node.js SDK创建和执行CFC函数的指导

2019-08	• 新增语言Lua 5 • 冷启动时针对大包解压的优化
2019-07	新增Java SDK实现初始化并且新建CfcClient、函数操作、函数调用、版本操作等操作
2019-06	• 新增语言PHP 7、Golang 1.x、Node.js 10 • 支持Java语言的HTTP原生接口开发 • 新增console管理测试事件模板的功能 • 新增定时触发器
2019-04	• CDN触发器上线 • 支持IAM主子用户权限 • 百度智能云消息服务触发器

2019-02	- 支持函数级并发度设置 - 新增Python 3支持
2018-10	- 正式接入计费，取消白名单限制 - 支持HTTP触发器 - 接入BCM报警
2018-09	- 上线苏州区域 - 支持BOS触发器 - 支持BOS日志存储 - 支持控制台查看日志 - 开发语言支持Java

	提供本地开发工具支持
2018-06	• 上线广州区域 • 开发语言支持Python、Node.js • 提供函数运行状态监控指标 • 提供安全数据库存储服务RDS on TLS
2017-11	百度智能云函数计算服务CFC正式发布

产品描述

什么是函数计算CFC

函数计算CFC（Cloud Function Compute）提供基于事件触发、全托管的云端计算能力。由于其“无服务器”特性，您仅需要开发业务代码并上传，无需关注和配置服务器资源，CFC 会托管您的代码并在代码需要执行时自动分配计算资源，使您的业务可靠地运行。

[🔗 1分钟了解CFC](#)

[🔗 功能介绍](#)

[🔗 代码管理](#)

您可以上传代码使用百度云 CFC。CFC 提供多种代码管理方式：

- 控制台编辑代码：适用于没有外部依赖的业务代码，直接在控制台界面上编辑代码。
- 上传代码 ZIP 包：将所有依赖和代码打包成 ZIP 并上传至 CFC，平台将根据函数配置识别入口函数并执行。
- 从 BOS 上传：将打包好的 ZIP 函数代码事先存储在 BOS 中，然后通过 BOS 文件路径上传更新 CFC 函数代码。

[🔗 事件触发](#)

函数计算通过事件触发器连接到其他服务，当对应服务发出请求时，函数即会响应运行，做出相应处理和反馈。例如：通过 BOS触发器将函数连接到百度智能云对象存储服务，设置函数响应BOS的新建对象事件。当用户的对象存储中增加新的文件

时，函数即被触发运行，对新上传的文件进行诸如内容检查等操作。

🔗 支持多开发环境

CFC 支持多种开发环境，您可以选择适合您自身的开发语言，完成函数编写。目前已支持的语言包括：

- Go
- Java 8
- Lua 5.3
- .NET Core 2.2
- Node.js 10
- Node.js 12
- Node.js 8.5
- Node.js 14
- Node.js 16
- PHP 7.2
- PowerShell 6.2
- Python 2.7
- Python 3.6
- python 3.10

🔗 免于运维

利用函数计算，开发者完全无需创建和管理后端服务器，只要开发完业务代码即可快速启用服务，大大提高工作效率。函数计算支持多种编程语言和函数触发器，满足多样化的事件触发场景，同时提供弹性、高可用、扩展性好、极速响应的资源调度能力，保障用户业务的高性能运转。

🔗 按使用付费

用户仅需为函数运行时实际占用的资源付费，当函数不被调用时不会产生任何费用，从而达到成本的最优化。

产品优势

在传统的云服务模式中，用户在部署运行一个业务前，需要关心云资源如何分配和调度，需要关注应用运行环境的搭建和升级，这两个步骤通常要花费大量的时间和精力，并且需要持续投入来进行优化。

CFC提供了“无服务器”的计算能力，并不是真的没有服务器，而是使用者无需关心和服务器等底层资源相关的一切操作。用户可以将精力聚焦和业务相关的核心代码开中，然后将代码托管在CFC中，由CFC负责提供代码运行所需的环境，并且随时调度足够的资源满足代码运行的负载。

🔗 7分钟理解Serverless的优势

通过使用CFC，您可以获得以下优势：

• 运维成本低

您仅需关注业务逻辑开发，函数计算提供运行环境和底层资源的可用性保障，运维成本大幅下降。

• 使用成本低

仅需为代码运行时的资源消耗付费，代码停止运行时不需承担任何开销，真正实现按需计费，使用成本大幅下降。

- **响应速度快**

借助百度智能云容器技术，提供ms级别的事件响应能力，用户对资源调度和函数启动过程基本无感知。

- **扩展能力强**

扩展能力强，根据业务并发量所需快速扩展资源，在超高并发下依然能保障服务的可用性。

应用场景

函数计算适用于承载由一个或者多个特定任务组成的应用或服务，这些任务可以是持续执行，也可以是达到某些触发条件时执行。在一些相对复杂的系统中，用户也可以把其中可以解耦拆分的任务部署到函数计算中，将其余无法解耦的部分部署在百度智能云其它服务中。

例如在图片审查的场景中，后端服务需要执行的任务是在新图片上传时，检查图片内容，将包含不健康内容的图片进行标记或者删除。用户可以将检查图片内容和标记/删除图片的代码编写成函数托管在CFC中，然后使用BOS触发器将用户的对象存储账户设置为事件源。当存储账户中发生新图片上传的事件时，函数即被触发，对图片内容做检查然后根据检查结果执行对应的操作。

函数计算常见的应用场景包括：

- 托管DuerOS交互式应用
- 托管H5或其它响应式小程序
- 为IoT及边缘设备提供云计算能力
- 对象数据流式处理
- 定时执行批处理任务

核心概念

函数（Function）：函数是函数计算中的基本资源单位，每个函数都用来存储和执行用户编写的特定功能的一段代码。用户可以对函数配置触发器、环境变量等高级设置。

事件（Event）：任何能够触发函数执行的行为称之为事件，例如BOS数据处理的场景中，新文件的创建即可被定义为一个事件。

触发器（Trigger）：触发器用于连接函数与外部服务，通过在函数中设置触发器，用户可以定义和管理事件来源，从而指定什么样的外部服务可以产生事件来触发函数的执行。

环境变量（Environment Variables）：是指在函数计算中用来指定函数运行环境的一组键值对参数。环境变量默认不设定，当用户的函数需要指定运行环境的环境变量时可以进行设置。

BRN（Baidu Resource Name）：是百度智能云资源的唯一标识，您可以通过该标识调用或者识别百度智能云资源，比如API调用、DuerOS技能调用和IAM策略识别等。

使用限制

函数运行资源限制

资源	默认值
临时磁盘空间	用户函数只可读写 /tmp 这个临时目录
函数代码路径	用户代码存放在/var/task/下，若代码中包含配置文件，需要采用绝对路径的方式进行读取
函数申请内存区间	128M-1024M(按照128M递增)
函数最大运行时间	300秒
代码部署包大小(压缩为.zip文件)	150 MB
原始代码大小(.zip文件解压后)	500 MB
函数环境变量大小	4 KB
函数同步调用请求正文负载大小	6 MB
函数异步调用请求正文负载大小	256 KB

每个区域账户资源限制

资源	默认值
函数并发执行数	100（如果需要更大并发数，请 工单 联系客服）
可上传的所有程序包总大小	75 GB

产品定价

计费概述

计费方式

函数计算支持资源套餐包和按需付费两种计费方式。

- 资源套餐包，包含三种计费资源类型：资源使用量、函数调用次数和外网出流量。购买资源包后，调用函数时，将优先使用免费额度，当免费额度用完后则使用资源包抵扣。若无有效资源包或资源包已用完，则转为按需计费结算。详细资源套餐价格请参见[CFC价格详情](#)。
- 按需付费，在您创建函数时无需付费，之后根据您使用函数时实际产生的调用次数、函数运行占用资源的时间和公网流量计费，具体的计费价格请参见[CFC价格详情](#)。

计费公式

总费用=函数调用次数费用+函数运行占用资源时间费用+公网流量费用

免费额度

- 取消免费额度的规则自2023年3月10日0时开始生效。
- 生效日前开通CFC且产生首次调用的老用户，将于生效日开始不再享受每月免费额度的申领。
- 生效日后开通CFC且产生首次调用的新用户，可获得购买“0元新客包”资格，并在购买成功后连续3个月（包含购买当月）获赠免费额度，每月获得的免费额度详情如下。
 - 函数调用次数：100万次/月
 - 函数运行占用资源时间：400,000GBs/月
 - 函数公网流量：1GB/月
- 针对超出当月获赠免费额度情况，即当月函数调用次数、资源使用量、外网出流量超出免费额度后，将使用已购有效资源包结算；若无有效资源包，则使用按量计费结算。额度结算优先级为：免费额度>资源包>按量付费。

- 在免费额度体验期间，对于函数资源用量较大或不确定的场景，推荐同时购买对应量级的资源包做补充。

资源套餐包

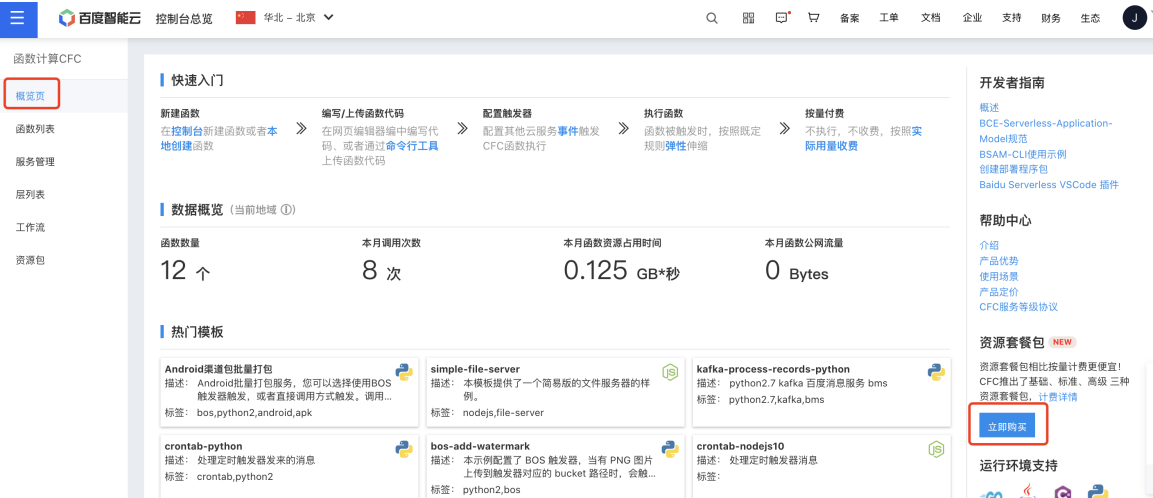
资源套餐包

CFC资源套餐包是一种资源抵扣包，让您以更实惠的价格使用CFC产品。资源套餐包采用预付费方式，分为月包和年包，月包有效期为一月；年包有效期为一年，到期后资源套餐包自动失效。例如您在2023年06月18日00:00:00购买了资源套餐年包，则该套餐包有效期至2024年06月18日00:00:00。请注意，资源套餐包购买后不可退款！

资源套餐包购买和使用

CFC目前支持华北区（北京）、华南区（广州）、华东区（苏州）三个地区的资源套餐包，以华北-北京区域购买基础资源包为例，下面展示如何购买和使用资源包。

- 点击CFC的概览页，并点击资源套餐包的<立即购买>。



- 选择你要购买的"地域"、"套餐类型"和"购买数量"，并点击<下一步>。

选择套餐

选择地域:

华北 - 北京

华南 - 广州

华东 - 苏州

套餐类型:

特惠固定月包

资源使用量: 10 万GBs
函数调用次数: 100 万次
外网出流量: 1 GB
时长: 一月
¥ 9.80
¥13.06

标准固定月包

资源使用量: 50 万GBs
函数调用次数: 500 万次
外网出流量: 5 GB
时长: 一月
¥ 45.71
¥65.30

高级固定月包

资源使用量: 100 万GBs
函数调用次数: 1000 万次
外网出流量: 10 GB
时长: 一月
¥ 84.89
¥130.60

公网流量包

外网出流量: 1 GB
时长: 一月
¥ 0.57
¥0.76

基础套餐包

资源使用量: 1 千万GBs
函数调用次数: 1 千万次
外网出流量: 10 GB
时长: 一年
¥ 728.44
¥1120.60

标准套餐包

资源使用量: 1 亿GBs
函数调用次数: 1 亿次
外网出流量: 100 GB
时长: 一年
¥ 6723.98
¥11206.00

高级套餐包

资源使用量: 10 亿GBs
函数调用次数: 10 亿次
外网出流量: 1000 GB
时长: 一年
¥ 61633.00
¥112060.00

购买数量:

1

每单最多可购买99个资源套餐包, 0元新客套餐包永久限购1次

自动续订:

☒ 自动续订勾选后, CFC每月/年将自动扣除续订费用

下一步

配置费用:
¥9.80

3、点击代金券列别可以查看您当前可用的代金券列表，您也可以选择激活代金券来使用，确认无误后，点击<提交订单>。

函数计算（CFC）资源套餐包订单确认

CFC资源套餐包信息

付费方式: 预付费	套餐类型: 基础套餐包（资源使用量: 1000 万GBs, 函数调用次数: 1000 万次, 外网出流量: 10 GB, 时长: 1 年）
地域: 华北 - 北京	单价: 728.44元
数量: 1	小计: 728.44元

代金券选择

+ 激活代金券

🔗 代金券管理

代金券:

选择代金券

产品金额: ¥728.44 代金券: ¥0.00

实际金额: ¥728.44

[《百度智能云线上订购协议》](#)

上一页 提交订单

4、检查订单的规格和金额是否正确，你可以选择账号余额支付或者第三方支付。

5、支付完成后，您可以在“订单管理”查看刚刚支付的订单。

6、选择“订单查看”，可以看到您的订单详细信息，资源包已经创建。

7、点击CFC控制台的"资源包",可以看到刚刚创建的资源包已经生效。

抵扣顺序

21

顺序扣减。例如，A包1个月后期到，B包3个月后期到，会预先扣减A包的量，扣除为0后再扣除B包用量；如果到期时间相同，则按购买时的下单先后顺序抵扣。

快速入门

概述

函数计算（CFC）让您仅需要管理自己的业务代码，而无需关心运行代码所需的服务器资源，您将代码打包在函数中，托管在CFC中运行即可。

本文将简要介绍CFC的基本使用流程，和通过不同方式快速创建和管理您的函数。

🔗 开通函数计算服务

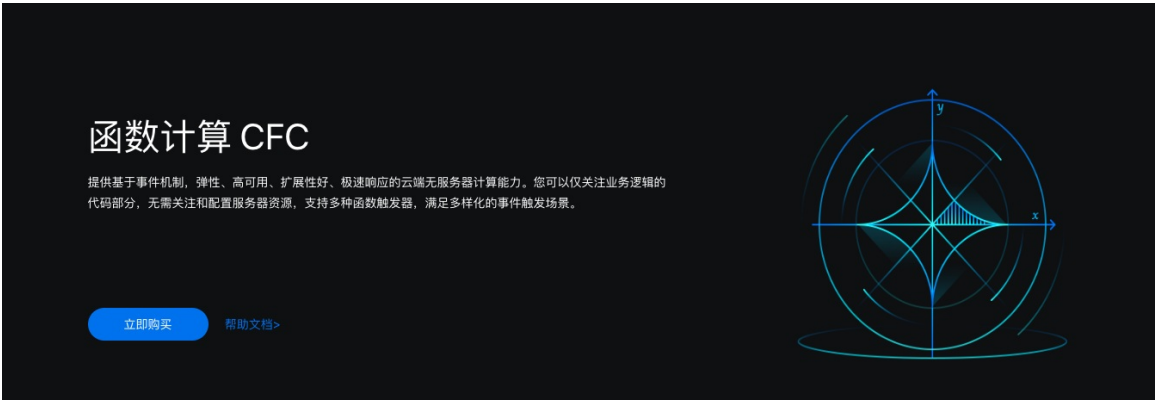
在使用函数计算之前，你需要先开通函数计算服务：

操作步骤

1. 打开[百度智能云官网](#)，通过【产品】-【云基础】-【Serverless】-【函数计算 CFC】进入函数计算主页。



2. 在函数计算主页，点击[立即购买](#)按钮。

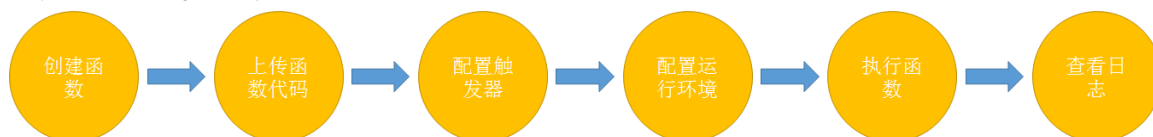


3. 点击“授权并开通服务”，完成函数计算服务开通。



使用流程

函数计算的基本使用流程如下图所示：



1. 创建函数：函数名称、描述和运行语言等基本信息。创建函数具体步骤可参考[创建函数](#)。同时，百度云为您提供了部分函数模版，创建函数时可以从[模板创建函数](#)。
2. 提供代码：可以将您的业务代码zip包上传至函数中，也可直接上传来自百度云对象存储(BOS)中的相关zip代码包；其中，函数运行时为Node.js、Python、Php、Lua、PowerShell的函数可以选择在线编辑函数的代码。
3. （可选）配置触发器：如果您的函数需要被某些特定事件触发执行，则可以为函数配置对应的触发器。
4. （可选）配置运行环境：如果您的函数对环境变量、内存大小等运行环境有要求，则可以进行自定义配置。在函数创建完毕后，您可以通过控制台[修改函数配置](#)。
5. 执行函数：当触发器对应的事件发生时，函数将被调用执行，函数计算自动为其分配资源，无需您手动干预。
6. 查看日志：在函数执行后，您可以在运行日志中查看函数的执行结果和监控数据。

使用方式

目前您可以通过4种不同的方式来创建和管理您的函数，包括：

1. [百度智能云控制台](#)：通过可视化UI管理函数
2. 函数计算Open API：在任何开发语言中快速管理函数
3. 函数计算SDK：在一些常用语言中，更便捷地管理函数；现已上线Python2、Python 3、Node.js、Java等SDK，详情参考[SDK使用示例](#)。
4. 本地开发环境：在您熟悉的IDE环境中管理函数

您可以根据具体情况选择以上的不同方式，来完成函数的创建、编辑和执行流程。关于这些方式的具体操作，请参考[操作指南](#)。

控制台使用示例

开通服务

1. 登录[百度智能云管理控制台](#)。
2. 选择“产品服务->容器->函数计算CFC”进入函数计算。
3. 首次进入，点击“授权并开通服务”，进入函数管理页面。

函数计算服务提供基于事件机制的云端无服务器计算能力。您可以仅关注业务逻辑的代码部分，无需关注和配置服务器资源，支持多种函数触发器，满足多样化的事件触发场景。授权后，我们会为您开通函数计算服务，并支持与IoT规则引擎、DuerOS自定义技能等触发源对接，内测期间函数计算服务完全免费。

若您有函数执行日志归档等合规需求，您可以开启BOS对象存储服务，我们会将您的函数执行日志推送至您的bucket，计费参照BOS计费规则，BOS存储可以保证您的日志持久性存储并方便您进行日志的管理、下载与分析。

授权并开通服务



创建函数

本示例演示从头创建函数，从模版创建函数的更多信息请参阅[CFC从模版创建函数](#)

1.在函数管理页面，点击“创建函数”，完成填写新建函数的信息和创建：

返回

创建函数

选择模版

函数配置

触发器配置

基础信息

函数名称:

testHelloWorld1

只能由数字、字母、-或_组成，长度小于65个字符，且在当前地域内唯一

服务名称:

default

管理服务

事件模式:

event

运行时:

Node.js 16

内存:

128

1285761024

超时时间:

3

最大限额300秒

日志存储:

无

描述:

请输入描述

上一步

下一步

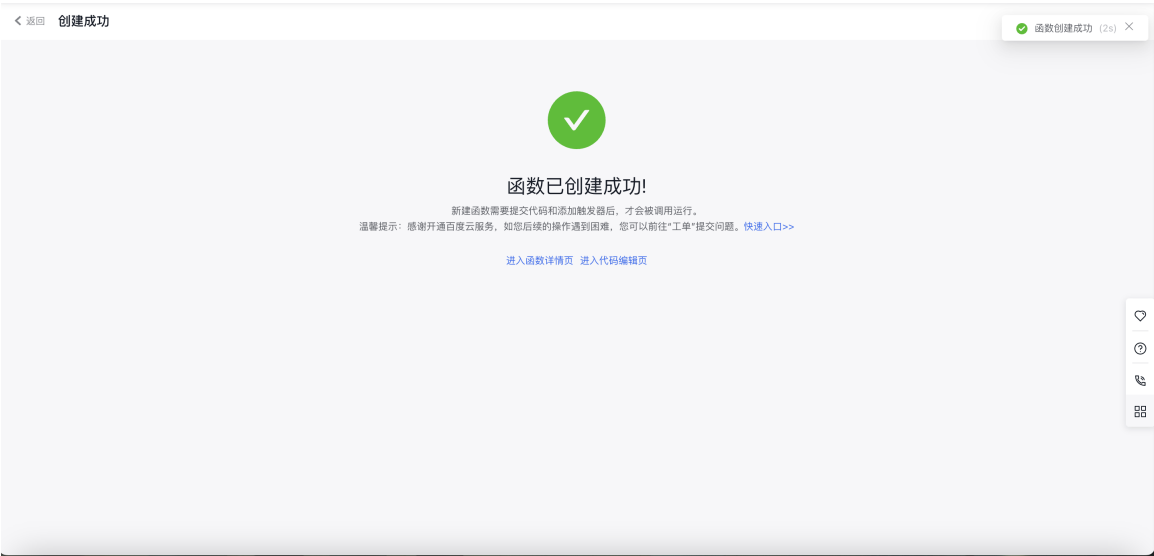
提交

- 函数名称：testHelloWorld1 （您可以指定自己的函数名，由数字、字母、-或_组成，长度小于65个字符）。
- 服务名称：default （您也可指定其他的服务名称）
- 事件模式：保持默认event即可。
- 函数执行内存：允许您为此函数分配内存的大小，默认为 128 M，最大为 1024 M。
- 超时时间：允许您为函数设置运行的最长时间，函数的运行超过这个时间会被强行停止。默认为3秒。
- 运行语言：选择Node.js 16 （在支持的语言范围内，可自行选择熟悉的语言。现支持Python 2.7，Python 3.6，Python 3.10，Node.js 10，Node.js 12，Node.js 14，Node.js 16，Java 8、Go、PHP 等）。

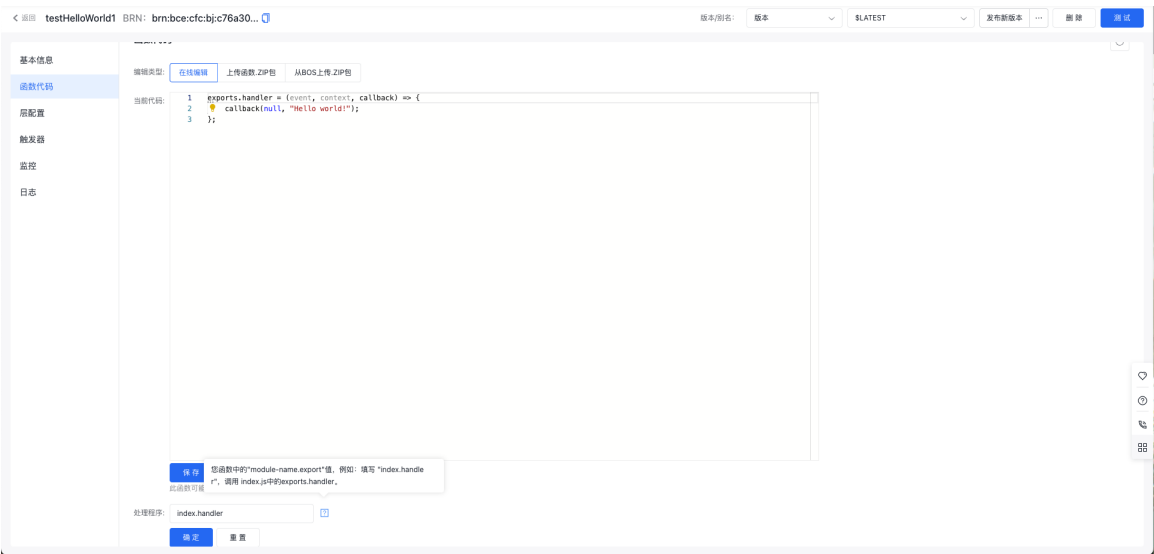
如果您选择的语言是 Java，请参考[Java 函数开发指南](#)文档。

- 日志存储：允许选择将日志存储到对象存储BOS，默认为无。
- 描述：留空（您可以填写对创建函数的详细描述）。

点击创建，完成创建过程。



2.进入代码编辑页面进行其他配置。



配置选项还包括：

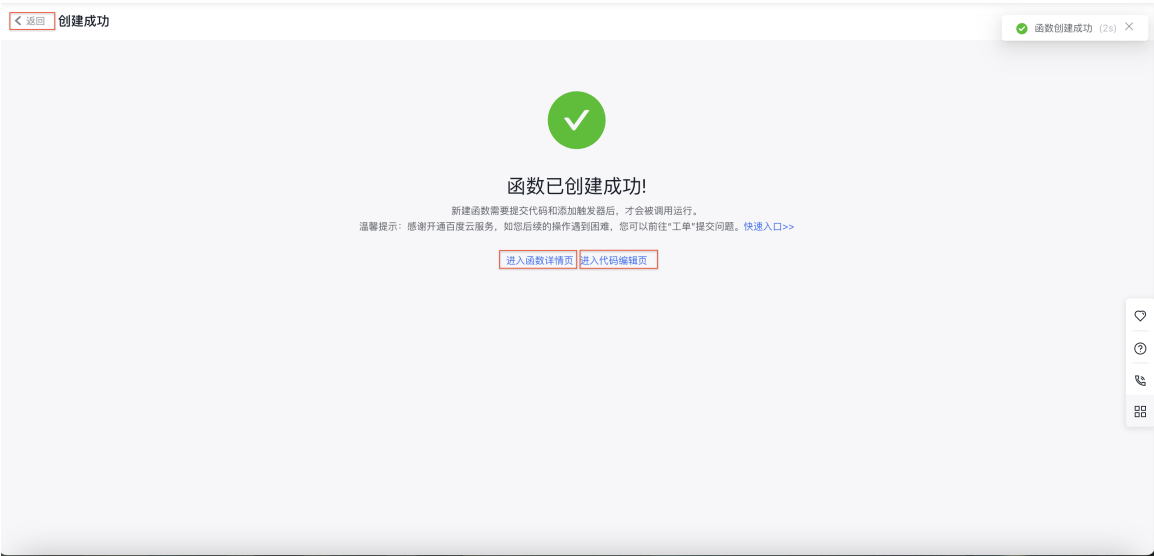
- **编辑类型**：选择**在线编辑**。
- **在线编辑器**：已经写好可输出Hello World的函数，您可以在此处编写自己的函数，本示例此处不需要修改。
- **环境变量**：用于CFC函数，使您可以动态地将设置传递到函数代码和库，而无需更改代码。
- **处理程序显示index.handler值**。它是 filename.handler-function。控制台将示例代码保存在 index.js 文件中，而在该代码中，handler 是调用CFC函数时将事件作为参数接收的函数名称。

调用函数

函数列表

点击返回函数列表，您可以回到函数管理页面，查看到刚才已经创建的函数。点击对应的函数名testHelloWorld 可跳到函数代码页面

函数创建结束后，自动跳转如下页面，用户可自行选择进入“代码编辑”页面，您可以再次编辑之前写的函数，也可以选择上传函数.zip包。本示例在此不作修改。



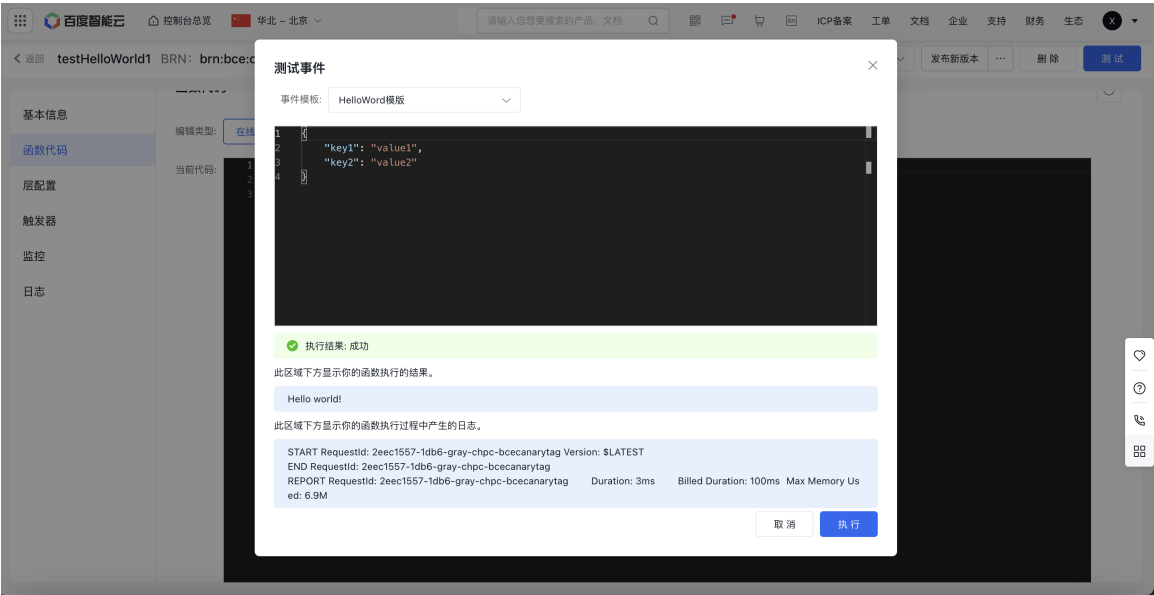
测试运行

- 1. 在函数基本信息页或函数代码页面点击“测试”。
- 2. 选择默认的HelloWorld模版。
- 3. 点击“执行”。

输入事件必须是JSON格式字符串。

查看结果

执行成功后在控制台可以查看本次执行结果和日志。



- **执行结果**：执行状态显示为成功，显示函数执行结果“Hello world！”。
- **执行过程中产生的日志**：显示本次函数执行的的日志，您还可以通过配置日志存储从控制台查看函数的历史执行日志。

查看日志

查看函数的历史执行日志需要为函数配置日志存储，有如下两种方法：

方法一：

- 1. 在创建函数时，设置日志存储项选择对象存储BOS或日志服务BLS。

2. 日志存储路径 选择预先创建的BOS Bucket；使用BLS会默认创建一个日志集，也可选择预先创建的日志集。
3. 点击保存。
4. 点击测试，可多次运行您的CFC函数

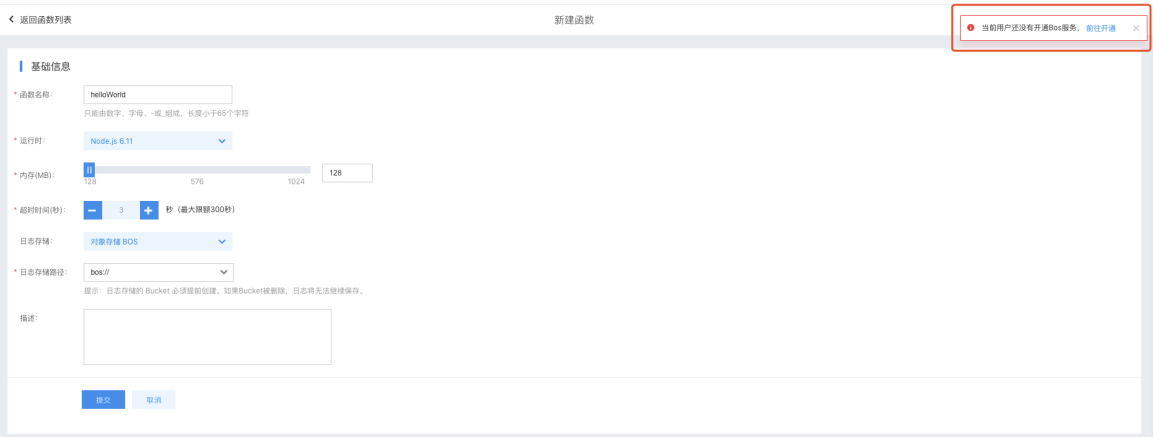
方法二：

1. 在函数基本信息页面选择编辑信息，进入函数信息修改页面。
2. 在设置日志存储 项选择对象存储BOS或日志服务BLS。
3. 日志存储路径 选择预先创建的BOS Bucket；使用BLS会默认创建一个日志集，也可选择预先创建的日志集。
4. 点击保存。
5. 点击测试，可多次运行您的CFC函数

说明：

选择日志存储路径时，如未开通过BOS服务，会出现如下警告信息，点击[前往开通](#)，跳到[开通BOS](#)页面，详情参考[开通对象存储](#)。

服务开通后，可创建一个新的Bucket存放函数计算日志，创建Bucket的详情参考[管理Bucket](#)。



从控制台查看BLS日志

1. 在函数代码页点击左侧的“日志”，可查看日志流列表。在查看日志流中点击某条日志流名称，可查看日志详情。

func-log-bls 版本: \$LATEST 操作

BRN: brn:bce:cfc:bj:1a2cbf55b97ac8a7c760c4177db4e17d:function:func-log-bls:\$LATEST

日志信息

全部日志

近5分钟 近15分钟 近1个小时 近1天 近7天 2021-11-26 15:11:12 - 2021-11-26 15:11:12 查询

输入关键字、requestID搜索

2021-11-26 15:10:39 成功

5bc6da98-25f2-40c5-bb22-1d01708c8474

2021-11-26 15:10:40 成功

035c177c-4693-4a7a-b4b7-681233edfa59

2021-11-26 15:10:42 成功

4356287c-cc9e-4e2f-9a3b-100c3d20e8ab

2021-11-26 15:10:42 成功

e0f13953-65b9-44e2-8e15-e189168aef9

2021-11-26 15:11:08 失败

5cadefa0-0253-4784-a10d-441d8f7e25b2

请求id: 5cadefa0-0253-4784-a10d-441d8f7e25b2

时间: 2021-11-26 15:11:08

日志

2021-11-26T15:11:07+08:00 START RequestId: 5cadefa0-0253-4784-a10d-441d8f7e25b2 Version: \$LATEST
Traceback (most recent call last):
File "/var/task/index.py", line 4, in handler
json.load(event)
NameError: name 'json' is not defined
2021-11-26T15:11:08+08:00 END RequestId: 5cadefa0-0253-4784-a10d-441d8f7e25b2
2021-11-26T15:11:08+08:00 REPORT RequestId: 5cadefa0-0253-4784-a10d-441d8f7e25b2 Duration: 148ms Billed Duration: 200ms Max Memory Used: 7.1M

从控制台查看BOS日志

1. 在函数代码页点击左侧的“日志”，可查看日志流列表。

testHelloWorld
BRN - brn:bce:cfc:bj:640c8817bd1de2928d47256dd0620ce5:function:testHelloWorld:\$LATEST

查看版本创建别名发布新版本测试

查看日志流

日期: < 2018-09-11 >

日志流名称	最后更新时间
\$LATEST_201809111230_4b0335fFeb2c71cf1529a9638f055ced	2018-09-11 12:34:04
\$LATEST_201809111240_9071f2e1d10e102074207938195d7d3f	2018-09-11 12:44:28

共2条 上一页 1 下一页

2. 在查看日志流中点击某条日志流名称，可查看日志详情。

testHelloWorld
BRN - brn:bce:cfc:bj:640c8817bd1de2928d47256dd0620ce5:function:testHelloWorld:\$LATEST

查看版本创建别名发布新版本测试

日志流详情 \$LATEST_201809111230_4b0335fFeb2c71cf1529a9638f055ced

2018-09-11T04:34:00.403Z START RequestId: d565eb32-9612-434f-a34f-c44ced708180 Version: \$LATEST

2018-09-11T04:34:00.411Z END RequestId: d565eb32-9612-434f-a34f-c44ced708180

2018-09-11T04:34:00.411Z REPORT RequestId: d565eb32-9612-434f-a34f-c44ced708180 Duration: 126ms Billed Duration: 200ms Max Memory Used: 4.2M

2018-09-11T04:34:01.979Z START RequestId: 0dc4d1d7-1104-46fc-a1a2-0723057392af Version: \$LATEST

2018-09-11T04:34:01.984Z END RequestId: 0dc4d1d7-1104-46fc-a1a2-0723057392af

2018-09-11T04:34:01.984Z REPORT RequestId: 0dc4d1d7-1104-46fc-a1a2-0723057392af Duration: 6ms Billed Duration: 100ms Max Memory Used: 11.6M

2018-09-11T04:34:03.001Z START RequestId: dd7dea2f-1743-46d0-9b06-388b97568cdd Version: \$LATEST

2018-09-11T04:34:03.002Z END RequestId: dd7dea2f-1743-46d0-9b06-388b97568cdd

2018-09-11T04:34:03.002Z REPORT RequestId: dd7dea2f-1743-46d0-9b06-388b97568cdd Duration: 2ms Billed Duration: 100ms Max Memory Used: 11.5M

2018-09-11T04:34:03.854Z START RequestId: 7b032c4d-ff7c-4981-9657-d846247ce3b6 Version: \$LATEST

2018-09-11T04:34:03.855Z END RequestId: 7b032c4d-ff7c-4981-9657-d846247ce3b6

2018-09-11T04:34:03.856Z REPORT RequestId: 7b032c4d-ff7c-4981-9657-d846247ce3b6 Duration: 1ms Billed Duration: 100ms Max Memory Used: 11.6M

2018-09-11T04:34:04.682Z START RequestId: 1435245b-29fe-48f0-ac8f-7cb0f1fc20fa Version: \$LATEST

2018-09-11T04:34:04.683Z END RequestId: 1435245b-29fe-48f0-ac8f-7cb0f1fc20fa

2018-09-11T04:34:04.683Z REPORT RequestId: 1435245b-29fe-48f0-ac8f-7cb0f1fc20fa Duration: 2ms Billed Duration: 100ms Max Memory Used: 11.5M

从BOS查看日志

- 1. 登录BOS控制台。
- 2. 在Bucket管理页选择存储日志的Bucket。
- 3. 点击名称为testHelloWorld的文件，可在该文件下查看日志列表。

cfc-test-helloworld 存储地区: 华北 - 北京

Object管理Part管理

欢迎使用BOS桌面，文件传输 更便捷! Windows版下载 Mac版下载

+ 上传文件 + 新建文件夹 批量删除

返回上级 cfc-test-helloworld / testHelloWorld

伏支持按前搜索

文件名称	存储类型	大小	更新时间	操作
☐ \$LATEST_201809111230_4b0335fFeb2c71cf1529a9638f055ced	标准	1.53KB	2018-09-11 12:34:04	文件信息 删除 更多
☐ \$LATEST_201809111240_9071f2e1d10e102074207938195d7d3f	标准	3.67KB	2018-09-11 12:44:28	文件信息 删除 更多

加载更多

- 5. 在“操作”中，点击第一条日志的“文件信息”中文件名称后面的下载。
- 6. 将下载的文件用文本编辑器打开，可查看日志详情。

\$LATEST_201809111230_4b0335fFeb2c71cf1529a9638f055ced

2018-09-11T04:34:00.403Z START RequestId: d565eb32-9612-434f-a34f-c44ced708180 Version: \$LATEST

2018-09-11T04:34:00.411Z END RequestId: d565eb32-9612-434f-a34f-c44ced708180

2018-09-11T04:34:00.411Z REPORT RequestId: d565eb32-9612-434f-a34f-c44ced708180 Duration: 126ms Billed Duration: 200ms Max Memory Used: 4.2M

2018-09-11T04:34:01.979Z START RequestId: 0dc4d1d7-1104-46fc-a1a2-0723057392af Version: \$LATEST

2018-09-11T04:34:01.984Z END RequestId: 0dc4d1d7-1104-46fc-a1a2-0723057392af

2018-09-11T04:34:01.984Z REPORT RequestId: 0dc4d1d7-1104-46fc-a1a2-0723057392af Duration: 6ms Billed Duration: 100ms Max Memory Used: 11.6M

2018-09-11T04:34:03.001Z START RequestId: dd7dea2f-1743-46d0-9b06-388b97568cdd Version: \$LATEST

2018-09-11T04:34:03.002Z END RequestId: dd7dea2f-1743-46d0-9b06-388b97568cdd

2018-09-11T04:34:03.002Z REPORT RequestId: dd7dea2f-1743-46d0-9b06-388b97568cdd Duration: 2ms Billed Duration: 100ms Max Memory Used: 11.5M

2018-09-11T04:34:03.854Z START RequestId: 7b032c4d-ff7c-4981-9657-d846247ce3b6 Version: \$LATEST

2018-09-11T04:34:03.855Z END RequestId: 7b032c4d-ff7c-4981-9657-d846247ce3b6

2018-09-11T04:34:03.856Z REPORT RequestId: 7b032c4d-ff7c-4981-9657-d846247ce3b6 Duration: 1ms Billed Duration: 100ms Max Memory Used: 11.6M

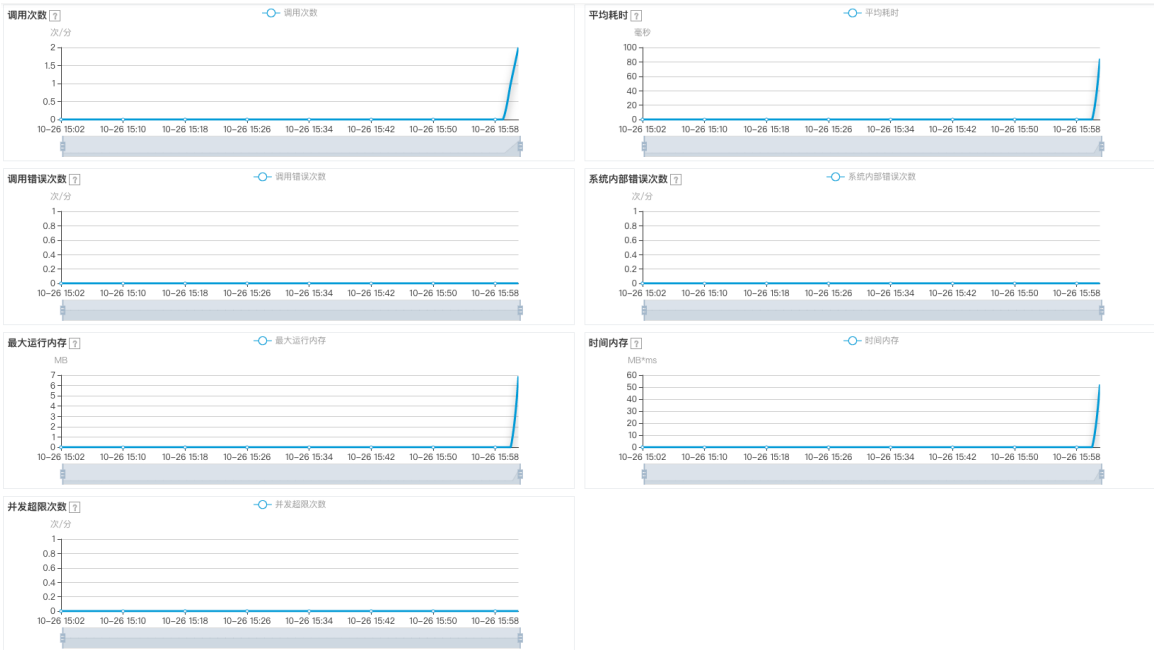
2018-09-11T04:34:04.682Z START RequestId: 1435245b-29fe-48f0-ac8f-7cb0f1fc20fa Version: \$LATEST

2018-09-11T04:34:04.683Z END RequestId: 1435245b-29fe-48f0-ac8f-7cb0f1fc20fa

2018-09-11T04:34:04.683Z REPORT RequestId: 1435245b-29fe-48f0-ac8f-7cb0f1fc20fa Duration: 2ms Billed Duration: 100ms Max Memory Used: 11.5M

查看监控

运行CFC函数后您可以点击导航栏“监控”，可以查看调用次数、平均耗时、调用错误、系统内部错误、最大运行内存、时间内存和并发超限次数等七项指标信息。



说明：

- 所有的监控数据都是在分钟粒度上聚合计算形成的时间序列。
- X轴默认为过去一小时内，可以拖动X轴的下的进度条来更详细的看这一小时内某段时间的指标。
- 调用次数显示此段时间内每分钟函数被调用的次数。
- 平均耗时显示这段时间内您的CFC函数运行所需的平均时间。
- 调用错误显示您的CFC函数内部逻辑调用失败的次数（包括超时、内存超限等用户使用错误）。
- 系统内部错误显示该函数在运行期间遇到的服务方的错误，这部分错误将不会纳入您的计费统计。
- 最大运行内存显示该函数在一段时间内单次最大的内存使用情况。
- 时间内存显示该函数在一段时间内的运行时间与使用内存之积，单位为MB*ms。
- 并发超限次数显示一段时间内该函数超过最大允许并发度上限的次数。

配置报警

监控信息

报警信息：

● 0项状态正常

● 0项状态异常

● 0项数据不足

设置报警

最近：

1小时

调用次数

调用次数

次/分

1

0.8

0.6

0.4

0.2

0

02-11 15:25

02-11 15:35

02-11 15:45

02-11 15:55

02-11 16:05

02-11 16:15

在监控大盘处，点击**设置报警**，即可跳转到报警策略配置界面，按照所需监控项配置监控阈值，同时选定报警动作（短信、邮件）。

策略信息

名称：

test_cfs

监控项：

设置并发超限次数

统计周期：

5分钟

统计方法：

和值

阈值：

>

2

次 仅支持输入整数

重复几次后报警：

1次

报警等级：

重要

次

● 设置并发超限次数

最近12小时

02-11 08:35

02-11 10:15

02-11 11:55

02-11 13:35

02-11 15:15

报警动作

异常

数据不足

恢复正常

+ 添加报警动作

添加成功后请刷新列表

请输入报警动作名称进行搜索

	报警动作名称	用户/用户组	通知方式	定时关闭
☐	only_liuao	liuao	短信、电话、邮件	- (UTC/GMT+08:00)
☐	alert	张天缺日	短信、邮件	- (UTC/GMT+08:00)

注意

- 报警策略配置后，需持续触发一段时间才能够激活生效（10分钟左右）。

操作指南

开通CFC服务

目前函数计算 CFC 服务已正式上线，您可以通过以下两种方式开通。

🔗 官网申请

参考入门指南 [开通函数计算服务](#)

🔗 从DuerOS开放平台申请

DuerOS开放平台开发者用户可以通过下面的步骤完成函数计算CFC服务的申请。

- 使用百度账号登录[DuerOS技能开放平台](#)。
 - 若没有用户名，请先完成注册，操作请参考[注册百度账号](#)。
- 通过如图所示“控制台”按钮，进入DuerOS技能设备平台控制台。



- 点击“创建自定义技能”按钮，在弹窗中技能类型选择“自定义”，然后依次输入**技能名称**和**调用名称**后点击“确定”，完成技能名称的创建步骤，并进入技能基础配置页面。

- 在技能基础配置页面的左侧导航栏选择“配置服务”进入服务部署页面，然后选择“百度智能云CFC”并点击[使用链接](#)，我们将为您开通CFC函数计算服务的使用权限，并且链接在登录百度智能云控制台账号的前提下，可以直接进入[函数计算CFC](#)产品页面，点击创建函数后即可在创建环节，进行您的函数创建。

授权信息配置

服务部署

技能平台SDK提供了Web开发技能所需的各种API接口，查看[PHP SDK 技术文档](#)或查看[Node.js SDK 技术文档](#)。推荐使用平台自带的部署服务，我们已经为您完成了全部的部署工作，您无需任何部署操作，只需在线编辑代码即可，并且这是完全免费的。

☐ 代码编辑 ☐ Web Service ☒ 百度云CFC

* 使用链接 [您可以免费使用百度云CFC服务，点击即可进行服务配置。](#)

* BRN

* 是否回调 ☒ 执行回调 ☐ 不回调

注意：

技能名称：技能名称是技能在小度技能商店中展现的名称，用户通过技能名称，快速辨识理解技能能力。

调用名称：技能在被用户调用时的名称，用户会通过表达“打开+调用名”进行使用技能

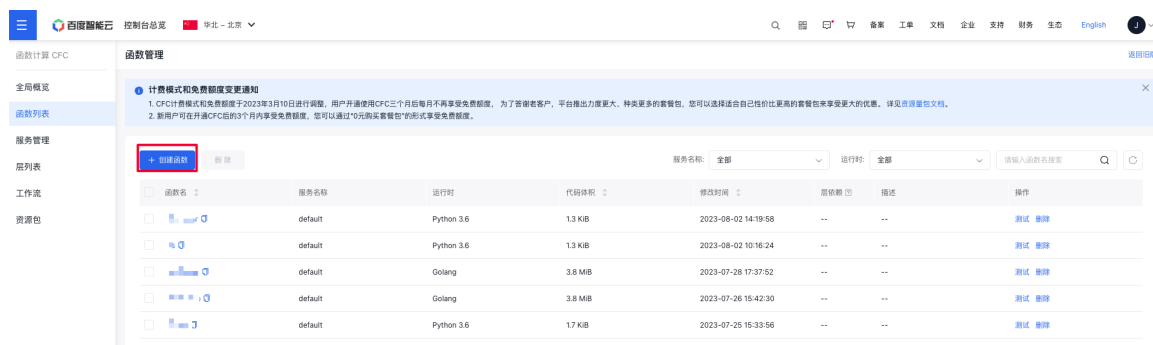
名称唯一：调用名称唯一，不能重复占用。

参考建议：清晰易辨识、避免同音词、避免使用生僻词组、避免复杂自创词汇、避免中文与英文数字混杂、不得带有“小度”或“小度小度”商标占用：非企业认证用户不得使用品牌名称（及同音词）作为调用名，企业级认证用户仅能使用该公司品牌及旗下品牌作为调用名

创建函数

前提条件

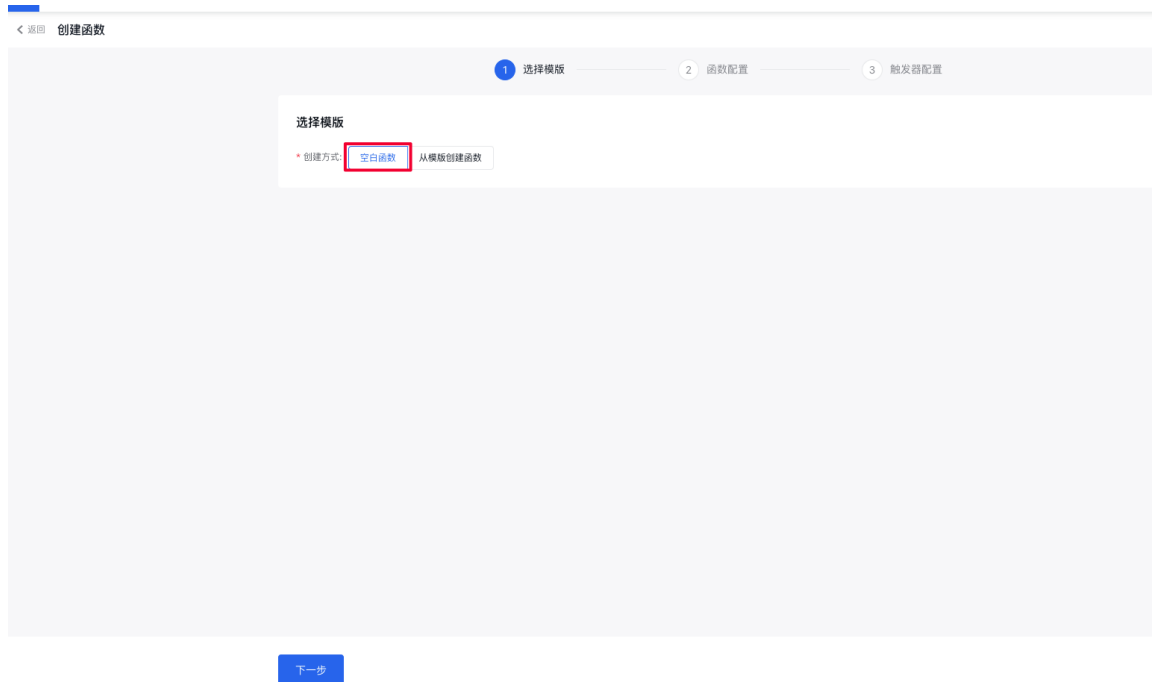
- 登录百度智能云平台[官网](#)。
 - 若没有用户名，请先完成注册，操作请参考[注册百度账号](#)。
 - 若有用户名，登录操作请参考[登录](#)。
- 登录成功后，在控制台页面选择“产品服务>计算>函数计算CFC”，进入[函数列表](#)页面，点击[创建函数](#)，进入创建函数界面。



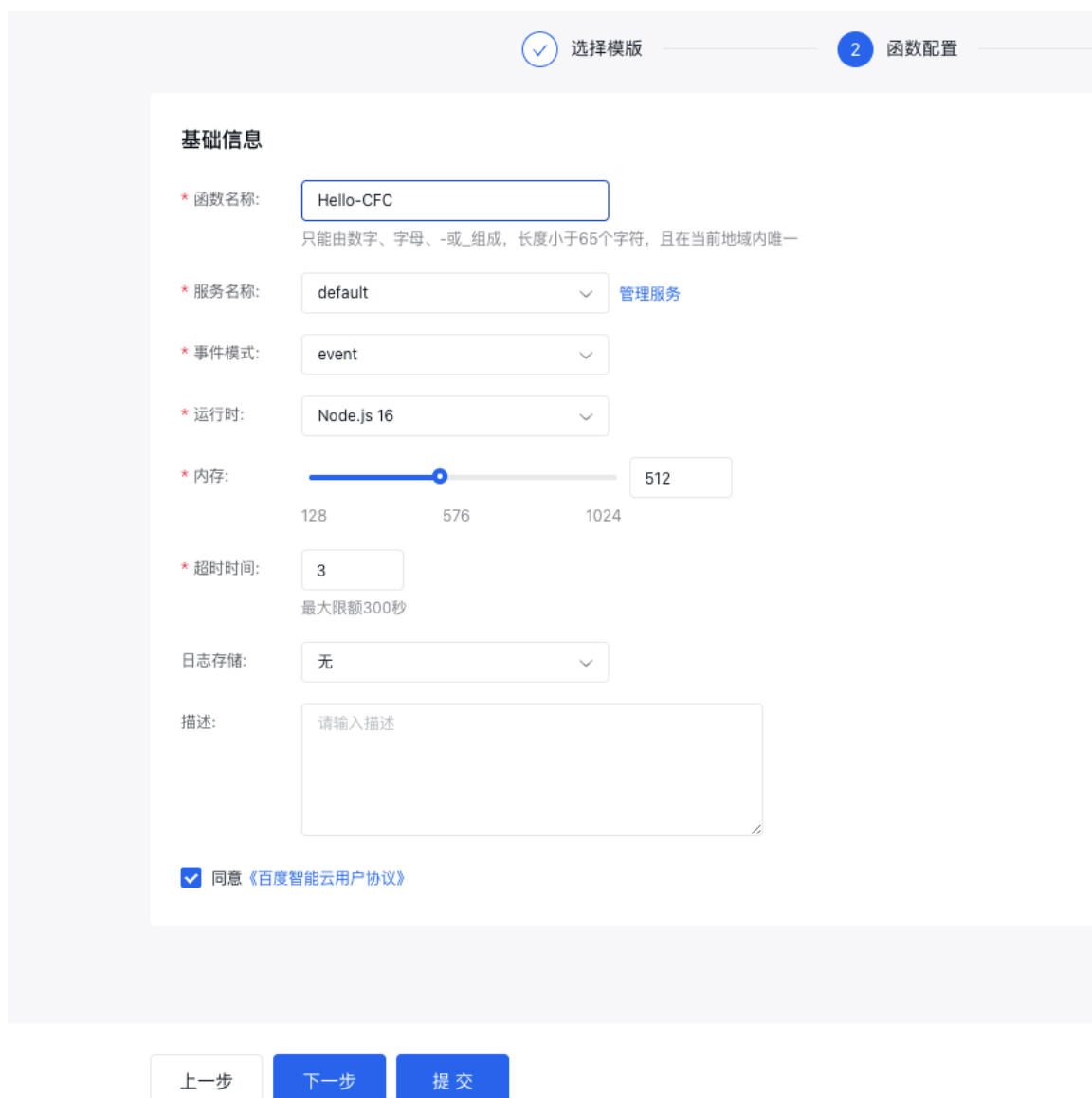
操作步骤

从头创建一个函数

- 在选择模版页面，创建函数选择“空白函数”。点击[下一步](#)进入函数配置页面。



2. 在函数配置页面填写函数的基本信息。此处演示创建一个 **Node.js 12** 运行时的函数。



- 函数名称：命名为 Hello-CFC。

说明：只能由数字、字母、-或_组成，长度小于65个字符。

- 事件模式：选择 event。

说明：支持event和stream两种模式，stream模式只支持java8运行时。

- 运行时：选择 Node.js 12。

说明：支持Node.js 6.11、Node.js 8.5、Node.js 10、Node.js 12、Node.js 14、Node.js 16、Python 2.7、Python 3.6、Python 3.10、Java 8、Golang、PHP 7.2、Lua 5.3、.NET Core 2.2、PowerShell 6.2。

- 内存（MB）：选择 128。

说明：支持128 ~ 1024MB。必须为128的倍数。

- 超时时间（秒）：此处使用默认。

说明：支持3 ~ 300秒。

- 日志存储（可选）：选择 "无"。

说明：可选对象存储BOS。选择 **对象存储BOS** 需指定BOS中预先创建的存储路径。

- 描述（可选）：此处不填写。

说明：0-256字符。

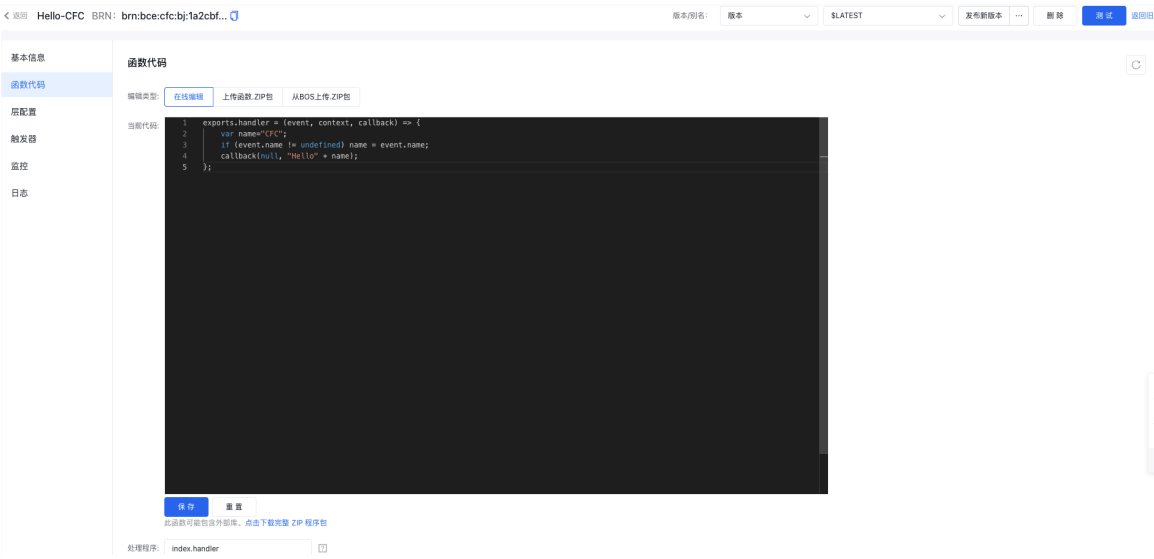
3. 点击 **下一步**，进入触发器配置页面，此处不配置触发器。点击 **提交** 创建函数。

说明：目前支持的触发器有 **DuerOS触发器**，**Duedge触发器**，**BOS触发器**，**HTTP 触发器**，**CDN 触发器**，**百度消息服务(Kafka) 触发器** 和 **定时触发器**等。触发器的详细描述见相关触发器文档。

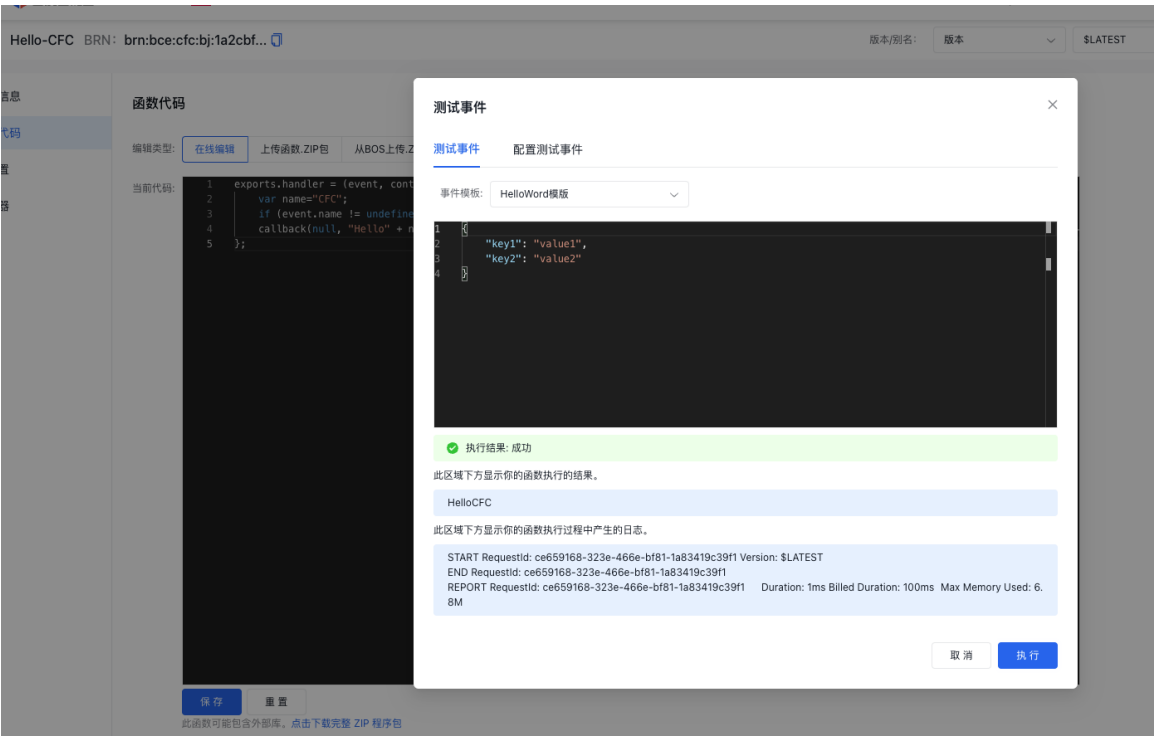
4. 在创建成功页面，点击 **进入代码编辑页** 进入代码编辑页面。



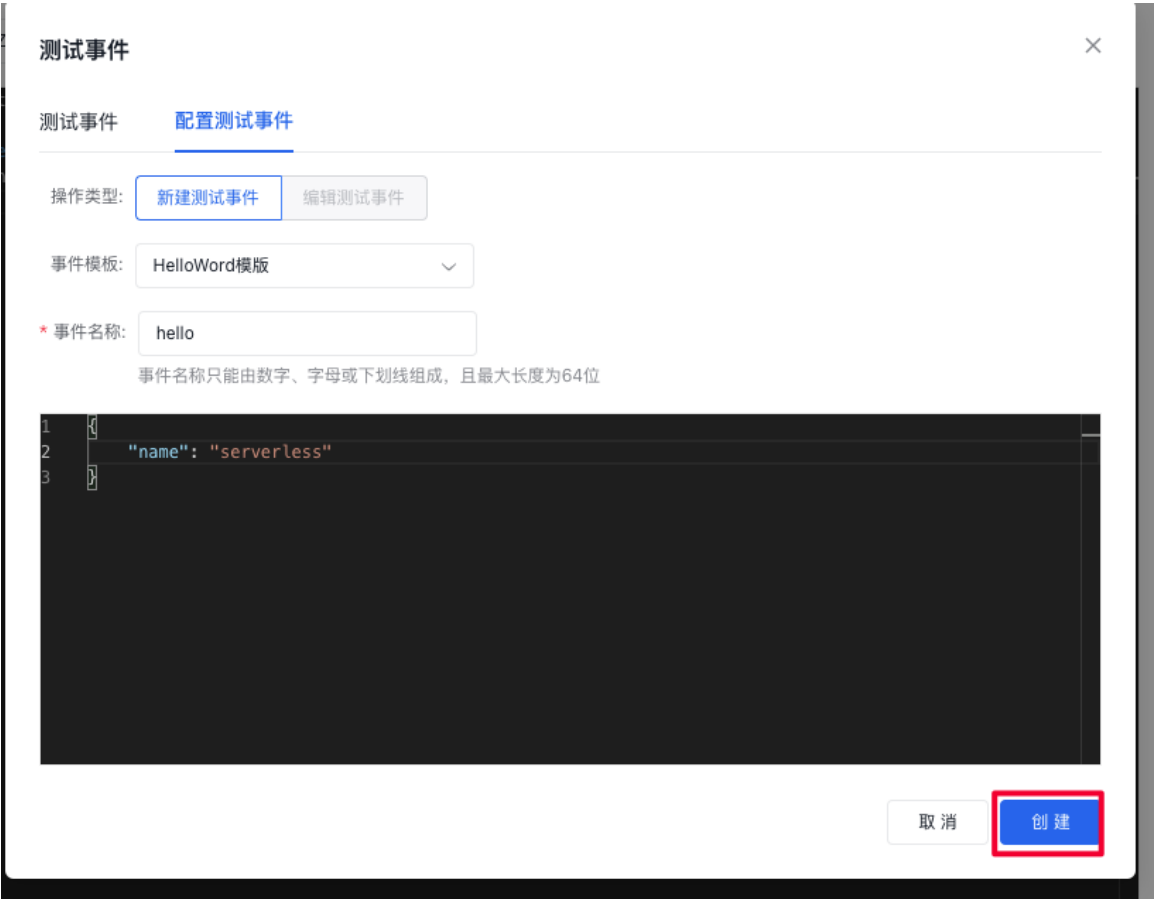
5. 您可以使用多种方式编辑您的函数，此处使用 **在线编辑** 创建一个Hello功能的函数，编辑代码如下，并 **保存** 编辑过的代码。



6. 接着，我们点击 **测试** 弹出函数测试页面。使用默认测试事件执行一次函数。



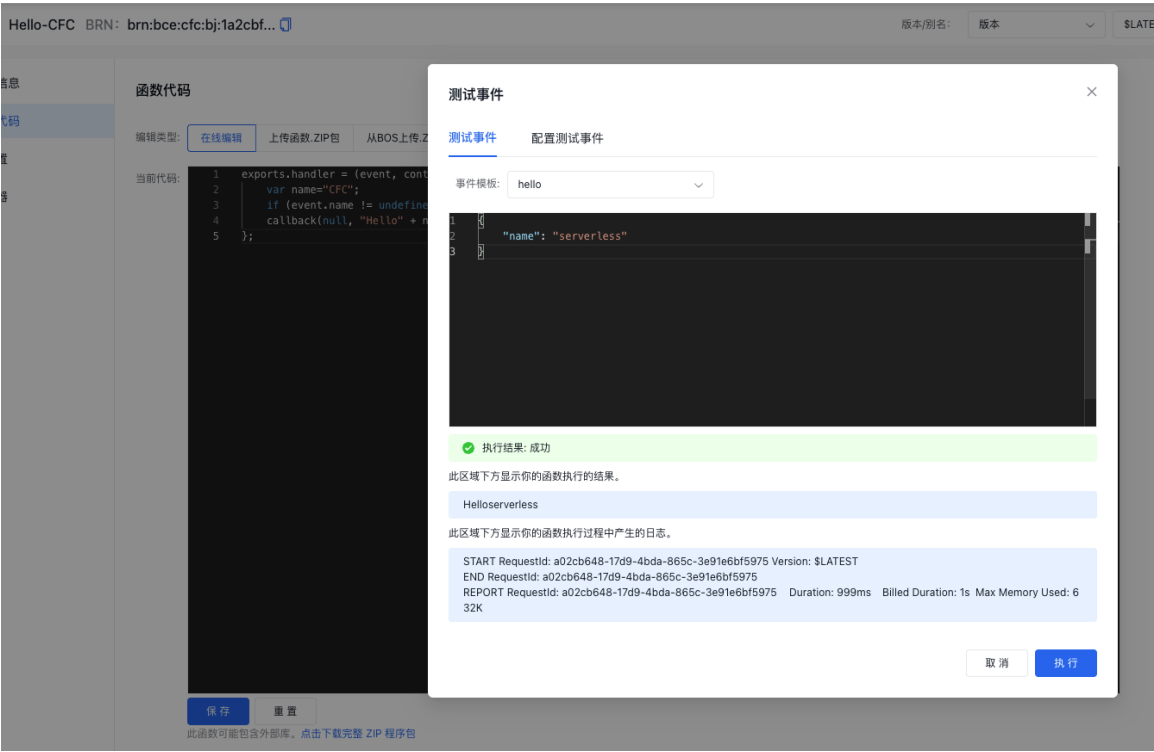
7. 您还可以自定义 **测试事件**，此处我们创建一个 Hello 事件。点击 **创建** 保存您的测试事件。



事件名称：填写hello。

事件结构：填写一个包含 name 属性的json串。

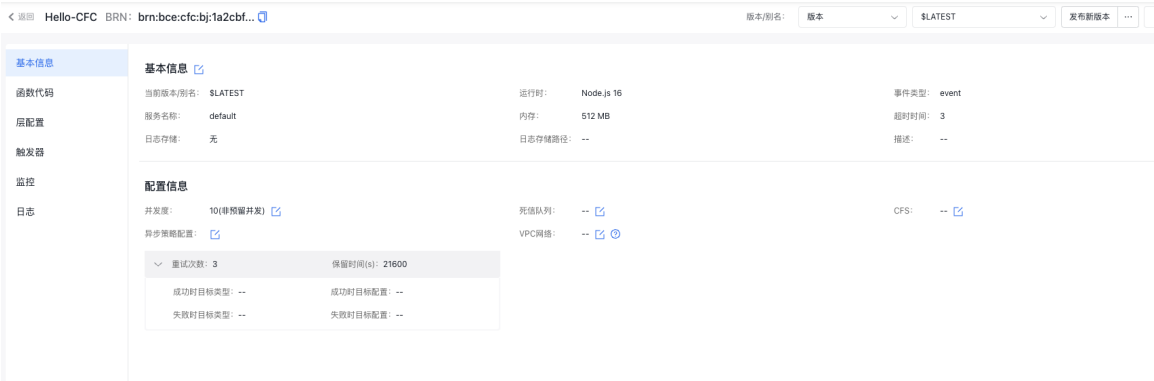
8. 使用自定义的测试事件测试函数。



9. 您还可点击侧边栏的 函数信息 进入函数信息管理页面。此处提供如下功能：

- 查看或修改函数的配置
- 创建别名

- 发布新版本
- 测试
- 删除函数
- 配置函数的高级功能（并发度、死信队列、VPC等）



10. 您还可点击侧边栏的 **触发器** 进入触发器配置页面。此处您可以查看新增和删除触发器。



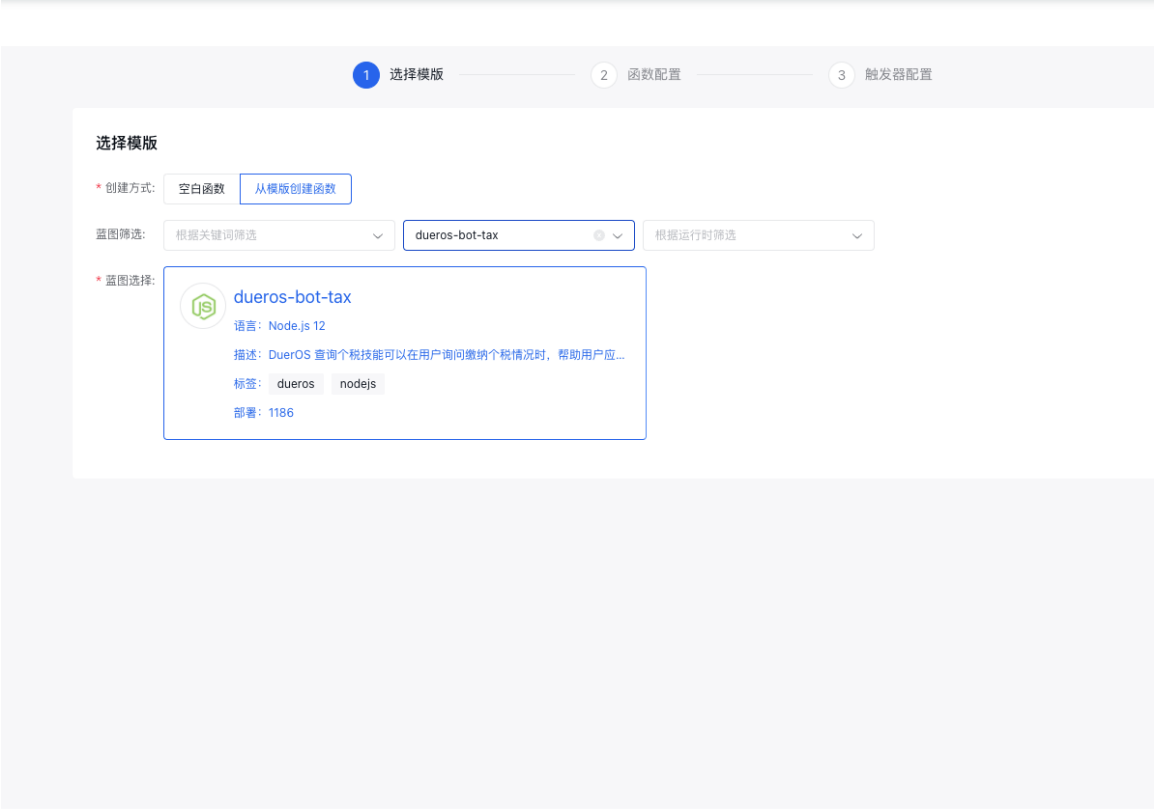
11. 您还可点击侧边栏的 **监控** 进入函数监控信息页面。此处您可以查看函数的调用次数、平均耗时和错误次数等信息。



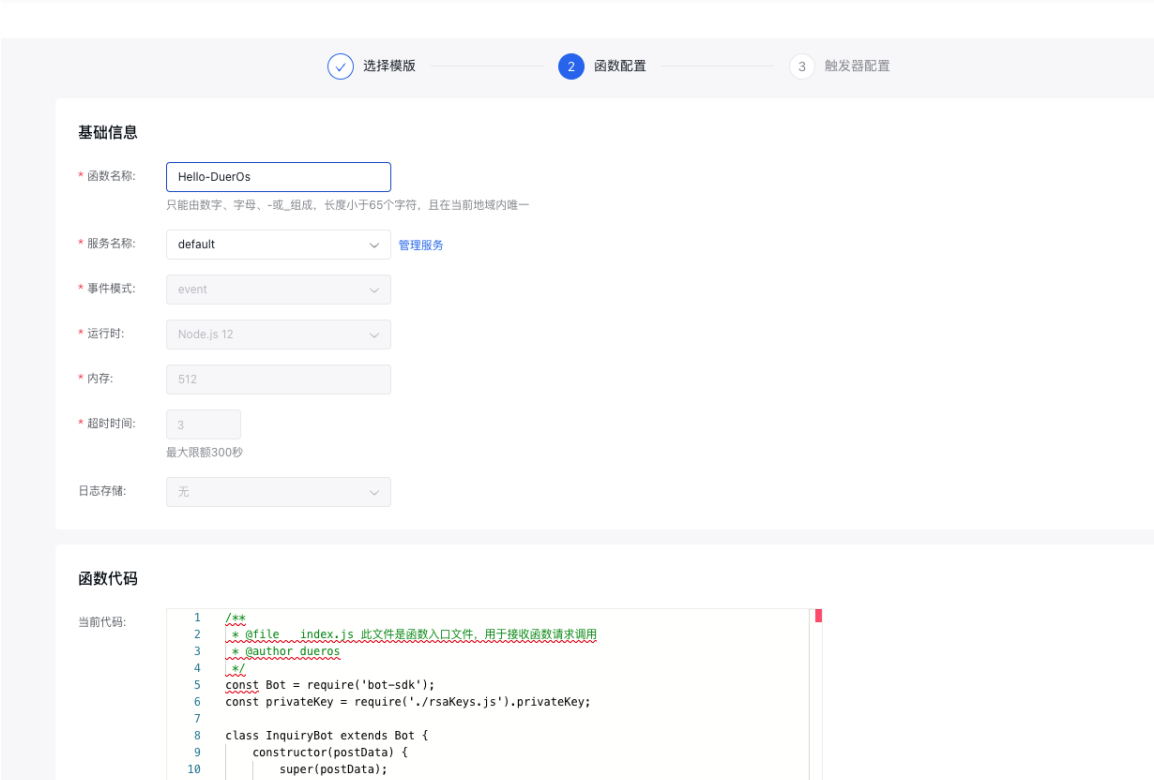
12. 如果您配置了日志存储，您还可以点击侧边栏的 **日志** 进入函数监控信息页面。详见 [函数日志配置](#)。

从模版创建一个函数

1. 在选择模版页面，创建函数选择 **"从模版创建函数"**，选择模版 **dueros-bot-python2**，此模版是一个 DuerOS 的脚手架，函数默认内置了 BOT-SDK。点击 **下一步**。



2. 在函数配置页面填写函数的基本信息。此处可指定函数名称和修改代码。点击 下一步。



3. 点击 下一步，进入触发器配置页面，此处为模板设定的DuerOS触发器，且不可修改。点击 提交。

✓ 选择模版

✓ 函数配置

3 触发器配置

触发器

DuerOS触发器需要在DBP平台指定执行函数，配置操作可参考[文档](#)

* 触发器:

DuerOS触发器

☒ 同意《[百度智能云用户协议](#)》

上一步

提交

4. 在创建成功页面，点击 [进入代码编辑页](#) 进入代码编辑页面。



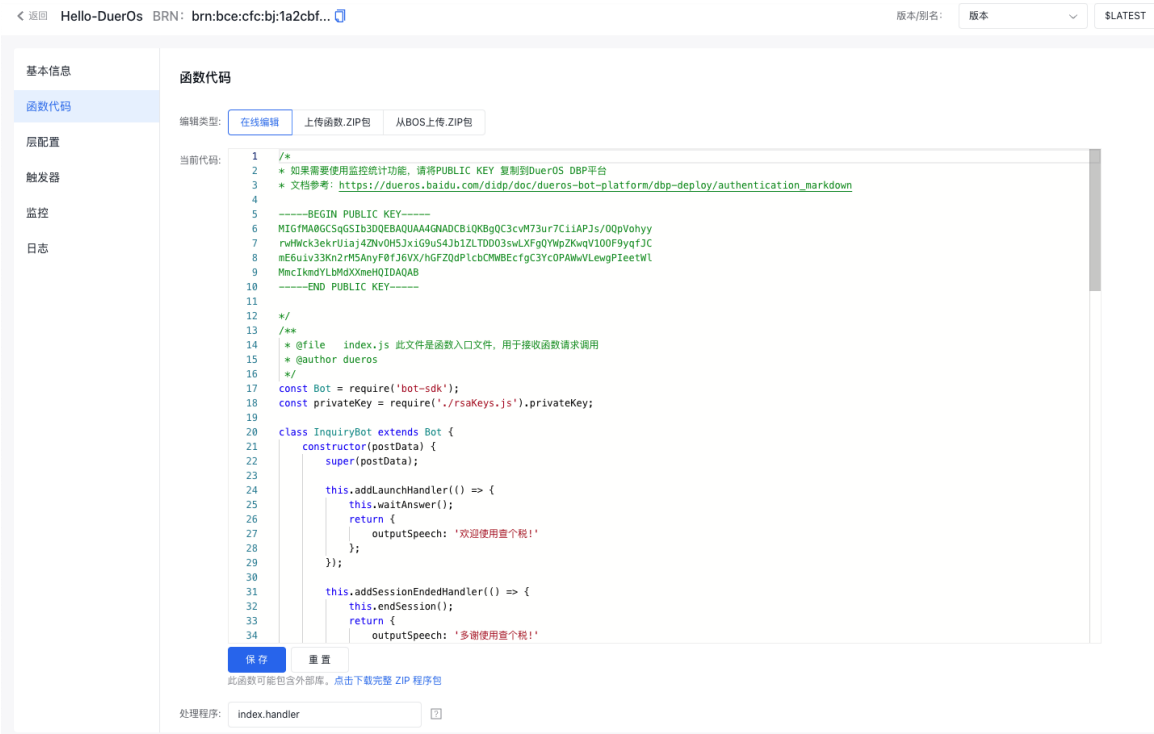
函数已创建成功!

新建函数需要提交代码和添加触发器后，才会被调用运行。

温馨提示：感谢开通百度云服务，如您后续的操作遇到困难，您可以前往“工单”提交问题。[快速入口>>](#)

[进入函数详情页](#) [进入代码编辑页](#)

5. 在函数代码页面，您可以查看和编辑创建的DuerOS技能函数。





管理函数

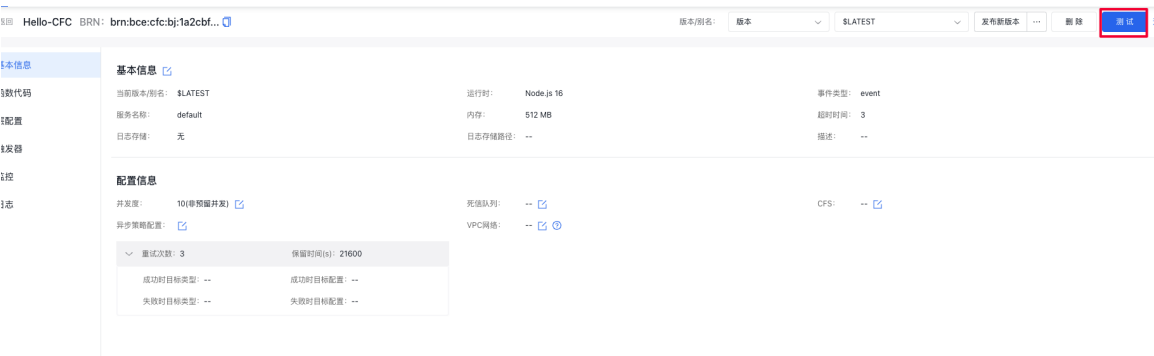
管理函数

测试函数

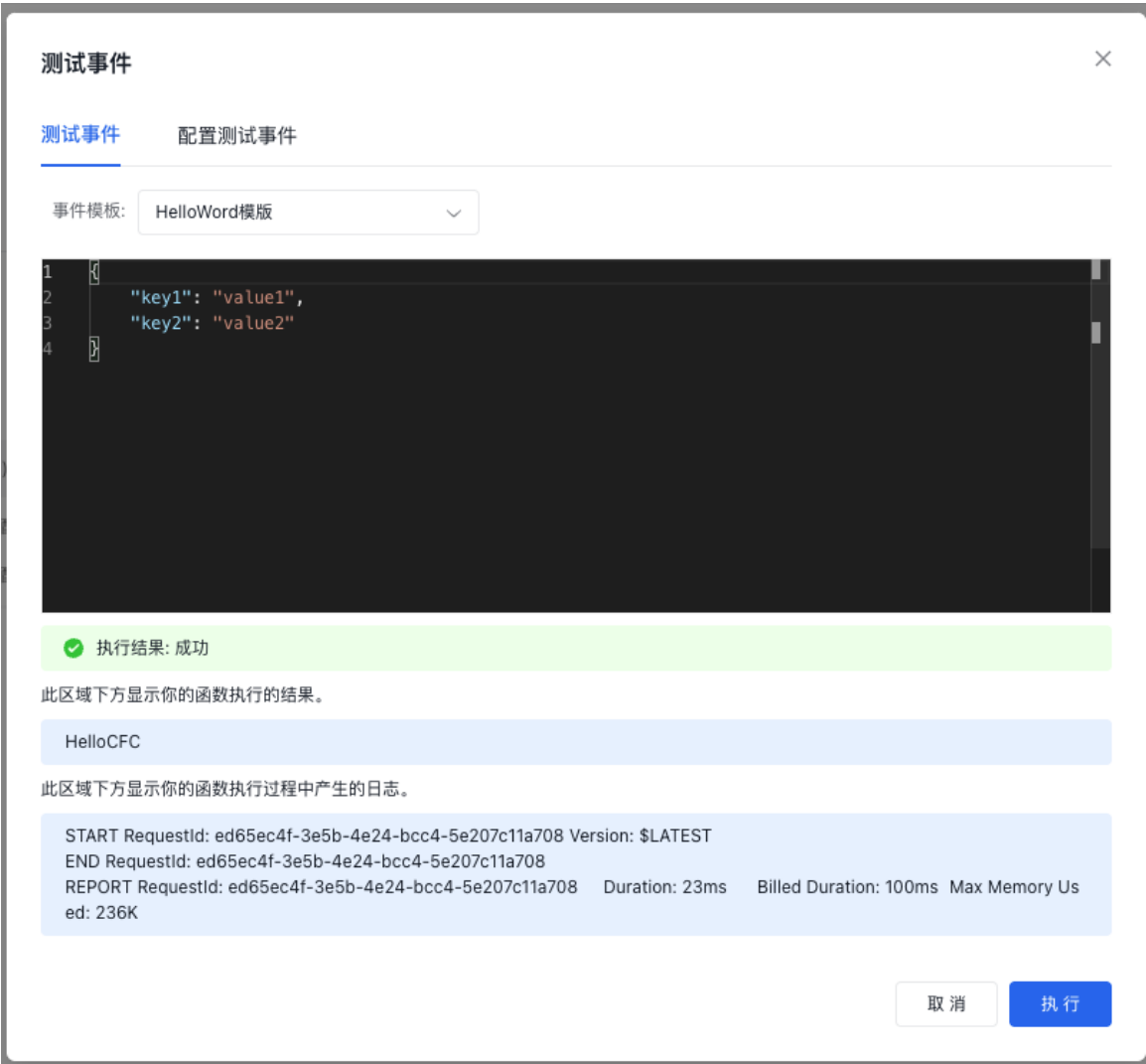
1. 登录[管理控制台](#)，选择“产品服务>云函数计算 CFC”，进入“函数列表”页面。
2. 点击需要测试的函数名称，进入函数详情页面。



3. 点击右上角的测试按钮，运行函数，并在弹出的菜单栏中，输入执行函数所需要的事件。事件是一个 JSON 串，如果这里创建了一个DuerOS的话，事件的 JSON 串需要遵循DuerOS的 API 协议。



4. 最后点击[执行按钮](#)，执行函数的测试。



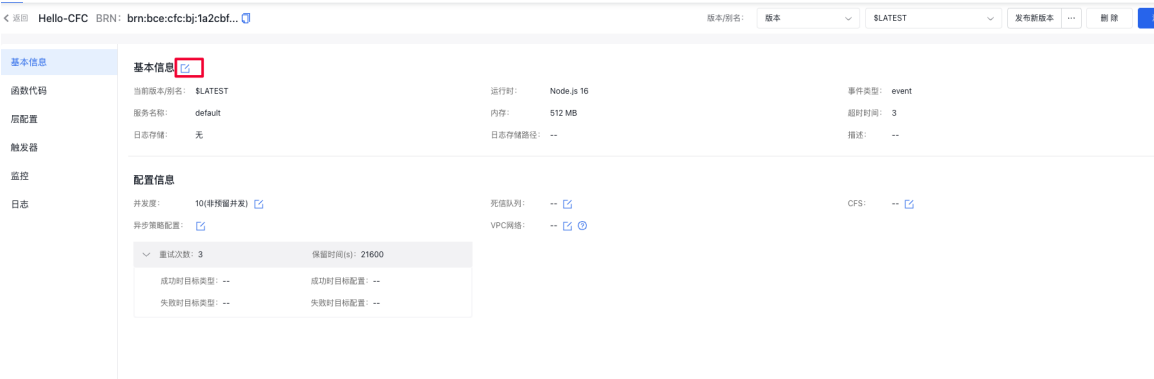
函数修改

函数代码修改

- 1. 登录[管理控制台](#)，选择“产品服务>云函数计算 CFC”，进入“函数列表”页面。
- 2. 点击需要修改的函数名称，进入函数详情页面。
- 3. 在函数详情页中进行函数的修改，之后点击右下角**确定**按钮完成函数的修改操作。

函数配置修改

- 1. 登录[管理控制台](#)，选择“产品服务>云函数计算 CFC”，进入“函数列表”页面。
- 2. 点击需要修改的函数名称，进入函数详情页面。
- 3. 点击函数详情页面左侧的基本信息，点击页面的**编辑信息**按钮。



4. 在函数配置页面进行函数运行时、内存、超时时间、日志存储等的修改，之后点击左下角保存按钮完成函数配置的修改操作。

基础信息

* 函数名称:

* 服务名称: [管理服务](#)

* 事件模式:

* 运行时:

* 内存: 512
128 576 1024

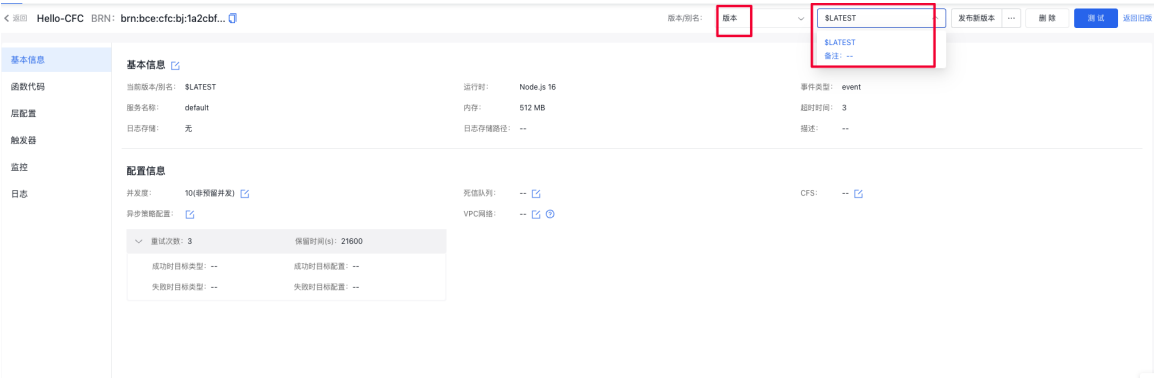
* 超时时间:
最大限额300秒

日志存储:

描述:

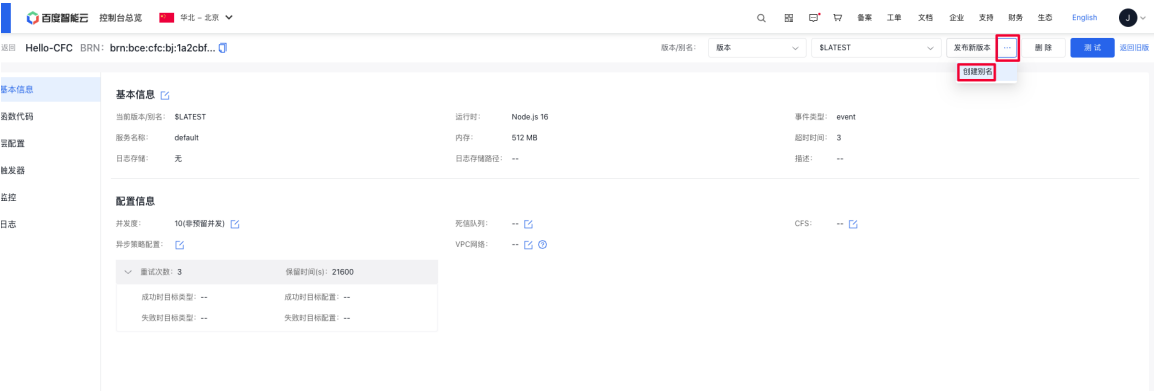
🔗 查看函数版本

1. 登录[管理控制台](#)，选择“产品服务>云函数计算 CFC”，进入“函数列表”页面。
2. 点击需要查看版本的函数名称，进入函数详情页面。
3. 点击函数详情页面左侧的基本信息，点击函数名右侧的版本/别名下拉框，通过查看版本和查看别名，选择需要查看的函数历史版本。



创建别名

1. 登录[管理控制台](#)，选择“产品服务>云函数计算 CFC”，进入“函数列表”页面。
2. 点击需要创建别名的函数名称，进入函数详情页面。
3. 点击函数详情页面左侧的基本信息，点击页面的[创建别名](#)按钮。



4. 可以在弹出的[创建别名](#)菜单栏中，选择需要创建别名的函数版本，并在名称栏中输入别名，完成函数别名的创建。

创建别名

* 别名名称:

请输入别名名称

大小写字母、数字以及-_/特殊字符，必须以字母开头

描述:

请输入描述

0 / 125

* 主版本:

\$LATEST

权重: 100%

添加灰度版本时，主版本不能是\$LATEST

灰度版本:

无

权重: 0 %

根据权重(%)在两个版本之间转移流量

取消

确定

发布新版本

1. 登录[管理控制台](#)，选择“产品服务>云函数计算 CFC”，进入“函数列表”页面。

2. 点击需要发布新版本的函数名称，进入函数详情页面。可以通过点击函数基本信息页的“发布新版本”按钮，或者函数代码页的“发布新版本”按钮。
3. 可以在弹出的发布新版本菜单栏中，输入版本描述，点击**确认**，完成函数新版本的发布。

🔗 查询函数BRN

1. 使用您创建CFC函数的百度智能云账号，登录百度智能云平台[官网](#)。
2. 登录成功后，选择“产品服务>云函数计算 CFC”，进入“函数列表”页面，点击想要获得BRN的函数名称,进入函数详情页面可以查看函数的BRN。

配置并查看函数日志

您可以将函数执行的日志存储至百度云日志服务，再根据日志服务中存储的函数日志来执行代码调试、故障分析、数据分析等操作。本文介绍如何使用控制台来为云函数配置日志存储，并查看函数执行的日志。

背景信息

百度云函数计算提供两种执行日志存储类型：日志服务BLS或对象存储BOS

日志服务BLS是一站式实时日志分析服务，提供日志数据采集传输、检索分析、实时消费与转储等功能，帮助用户轻松应对服务运维管理、商业趋势洞察、安全监控审计等业务场景。

对象存储BOS提供稳定、安全、高效、高可扩展的云存储服务。可以将任意数量和形式的非结构化数据存入BOS，并对数据进行管理和处理。BOS支持标准、低频、冷和归档存储等多种存储类型，满足多场景的存储需求。

操作步骤

百度云函数计算设置函数执行日志存储服务的步骤如下所示：

1. 打开[百度云控制台](#)，进入CFC控制界面。
2. 在左侧导航栏中，点击【函数列表】进入函数管理页面。
3. 点击要设置的函数名，跳转到函数函数基本信息页面。
4. 在基本信息页面中，单击【编辑信息】即可编辑函数日志存储服务。

The screenshot shows the 'Basic Information' (基础信息) page for a Cloud Function. The 'Log Storage' (日志存储) dropdown menu is open, showing three options: 'None' (无), 'Object Storage BOS' (对象存储 BOS), and 'Baidu Log Service BLS' (百度日志服务 BLS). The 'BLS' option is highlighted with a red box. Other fields visible include Function Name (bls-test-lz), Service Name (test_sns), Event Mode (event), Runtime (Python 3.6), Memory (128 MB), Timeout (3 seconds), and Single Instance Concurrency (1).

5. 点击日志存储下拉框，选择执行日志存储类型。

注1：对象存储BOS的Bucket必须提前在BOS创建。手动输入Bucket时，应确保Bucket所属地域和当前地域一致。如果Bucket被删除，日志将无法继续保存。

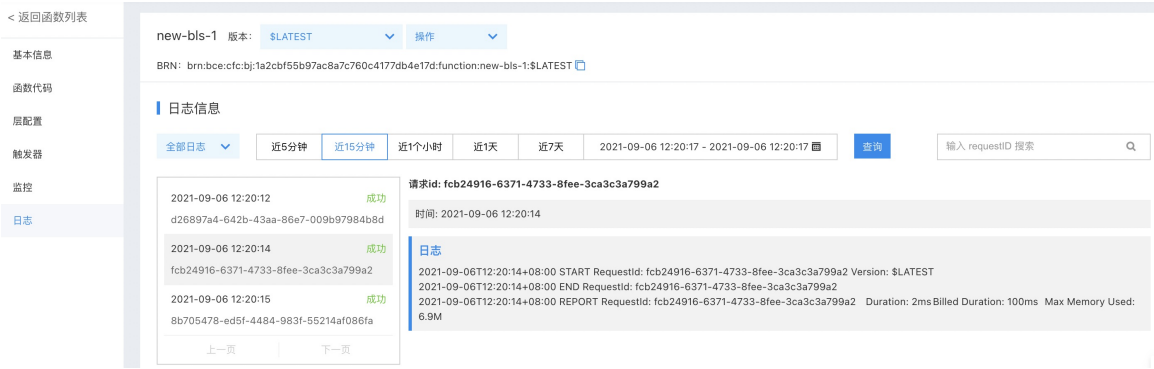
注2：百度日志服务BLS会为用户创建一个默认日志集，日志保存期限是7天

注3：如果您想自己定义日志服务BLS的日志集，可以在BLS新建日志集，选择指定日志集

查看日志

配置了日志存储的函数在运行时，产生的日志都会输出到日志存储服务中。查看日志步骤如下所示：

- 1. 打开[百度云控制台](#)，进入CFC控制界面。
- 2. 在左侧导航栏中，点击【函数列表】进入函数管理页面。
- 3. 点击要设置的函数名，跳转到函数函数基本信息页面。
- 4. 在左侧导航栏中，点击【日志】进入查看日志页面。
- 5. 百度日志服务BLS日志显示如下图：



以下为对象存储BOS日志显示页面：



网络管理

配置VPC功能

私有网络 VPC 是一个用户能够定义的虚拟网络，不同的私有网络之间相互隔离。您可以在自己创建的私有网络内创建和管理云产品实例，比如云服务器 BCC、云数据库 SCS等。在默认情况下，因为 VPC 的网络隔离，函数计算无法访问您在 VPC 中的资源。要让函数计算能够访问，需要您手动为函数配置 VPC 功能，提供包括 VPC 子网 ID 和安全组 ID 在内的配置信息。

VPC 有两个配置字段 `SubnetIds` 和 `SecurityGroupIds`，均不可为空。

```
"VpcConfig":{
  "SubnetIds": ["string", "string"],
  "SecurityGroupIds": ["string", "string"],
}
```

`SubnetIds` 表示函数执行期间进入的子网，您可以同时配置最多16个子网。函数执行期间会随机选择其中一个子网，当这个子网因不存在或者无空闲 IP 时而不可用时，函数会自动尝试进入其它子网。建议您至少配置两个子网，以提高函数的可用性。

`SecurityGroupIds` 表示函数所关联的安全组。您可以同时配置最多5个安全组。安全组是函数在 VPC 中的出入站规则，当有多个安全组时，函数会应用安全组的全部规则。

🔗 注意事项

使用VPC的场景

由于使用 VPC 功能会带来一些额外的耗时，所以建议当且仅当您的函数要访问 VPC 内的资源时，比如访问 BCC、RDS、SCS 等，才给函数配置 VPC。

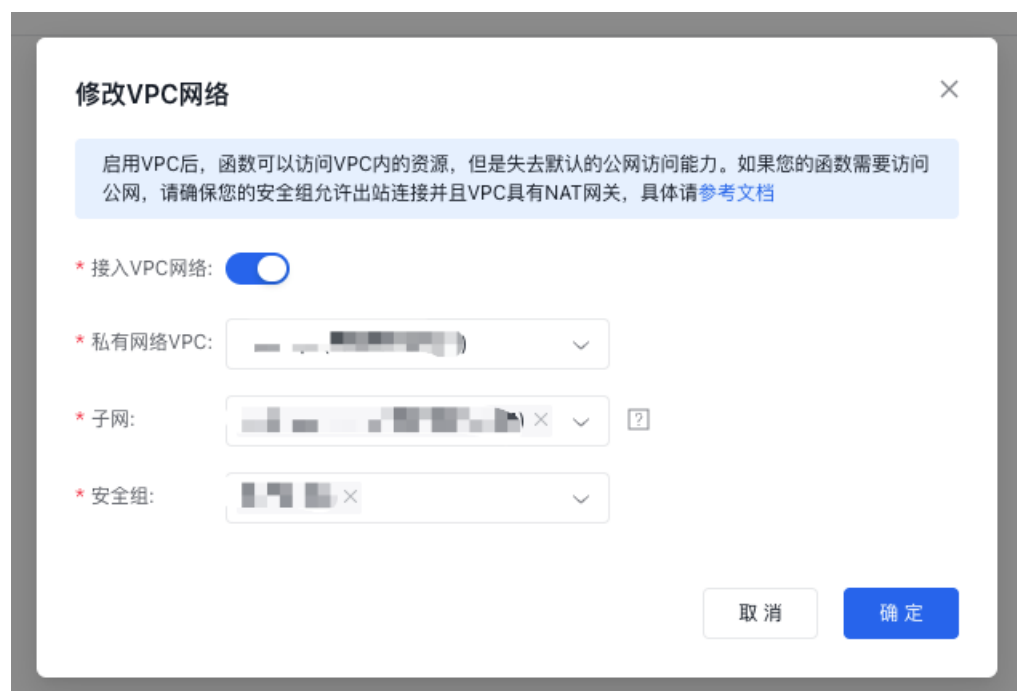
访问公网

函数未配置 VPC 前，函数的出站流量从函数计算服务的公共公网出口流出，因此能够正常访问公网。给函数配置 VPC 后，函数会进入 VPC 内的子网，所有出站流量均进入 VPC。如果函数同时需要访问 VPC 和公网，那么您需要在 VPC 中配置 [NAT 网关](#)，并绑定 [弹性公网 IP](#)。

安全组配置 安全组规则是允许访问的规则集合，表示函数能够访问的 VPC 内的网段和端口，请确保这些规则包含了要访问的资源地址，若未正确设置可能导致函数无法访问您的资源。

🔗 使用示例

使用控制台设置函数 VPC 功能，开启接入 VPC 网络，设置子网与安全组，最后点击“确定”，如下图所示。



函数在VPC中访问外网

默认情况下，函数计算可以访问百度云的服务和公网，给函数配置 VPC 功能后，函数的出口流量都进入 VPC 中。若您的 VPC 没有公网访问能力，那么函数则就无法访问公网。如果您想让函数同时能访问 VPC 资源和公网，您需要给 VPC 添加 NAT 网关。

创建弹性公网IP

登录 [弹性公网 IP 控制台](#)，您可选择“实例”或“共享带宽”。以“实例”为例，点击“创建实例”，根据您的需求配置付费方式、带宽大小等，创建一个公网IP，下图所示。

付费及地域

付费方式：

预付费

后付费

当前地域：

华北 - 北京

弹性公网IP配置信息

名称：

大小写字母、数字以及_/.特殊字符，必须以字母开头，长度1-65

* 公网带宽：

按使用流量计费

按使用带宽计费

* 带宽峰值：

1Mbps

500Mbps

1000Mbps

60Mbps

创建NAT网关

百度智能云 NAT 网关支持 VPC 中的 BCC、BBC 等实例连接 Internet 网络或其他百度智能云服务。NAT 网关将 VPC 中的内网 IP 地址和公网 IP 地址进行转换，为无公网 IP 的实例提供访问互联网的代理服务，详情请参考 [NAT 网关](#)。

登录 [NAT 网关控制台](#)，点击 "创建 NAT 网关"，在创建实例配置页面填写详细信息，如下图所示。

温馨提示：通过NAT网关访问Internet的实例必须位于NAT专属型子网中，详情您可以查看[帮助文档](#)或提交[工单](#)与我们联系。

付费及地域

付费方式：

预付费

后付费

当前地域：

华北 - 北京

配置信息

所在网络：

默认私有网络 (192.168.0.0/16)

* NAT网关名称：

请输入NAT网关名称

大小写字母、数字以及_/.特殊字符，必须以字母开头，长度1-65

类型：

小

中

大

SNAT公网IP：

☒ 弹性公网IP

☐ 共享带宽

180.76.138.79 (100Mbps)

在 SNAT 公网 IP 设置项，选择您刚刚创建出来的弹性 IP 或共享带宽。

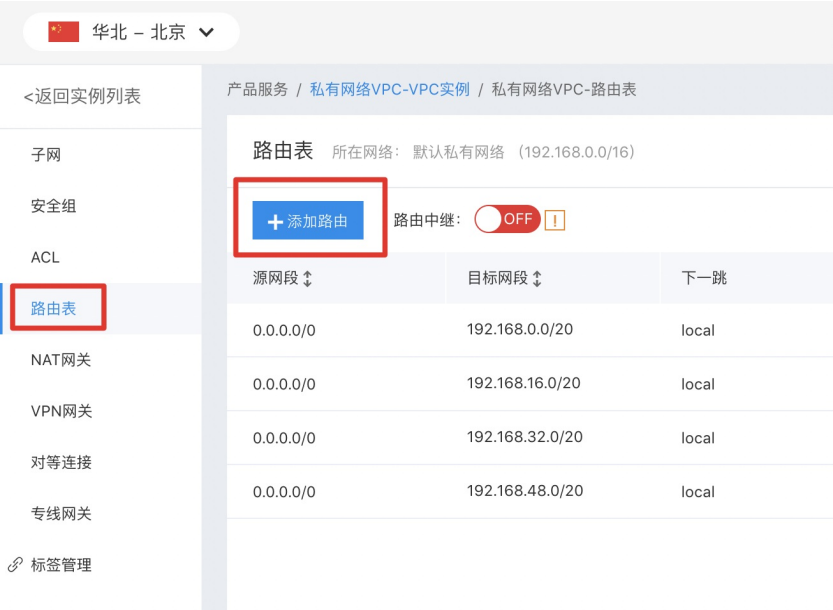
注意：

- 1. NAT 网关要和函数、VPC 部署在同一地域；
- 2. NAT 网关所属的 VPC 必须是函数所配置的 VPC；

47

创建路由策略

在私有网络 VPC 控制台中选择“路由表”，点击添加路由。



在弹出的添加窗口中，填写目标网段，若允许函数访问所有外网，请填写"0.0.0.0/0"，若只想让函数访问部分网段，请填写该网段的 CIDR。路由类型选择 "NAT网关"，下一跳实例选择您刚刚创建的 NAT 实例，点击“确定”即可。

添加路由

源网段:

nat-net (192.168.72.0/21)

* 目标网段:

0.0.0.0/0

路由类型:

NAT网关

* 下一跳实例:

test-nat (180.76.138.79)

描述:

确定

取消

版本与别名管理

版本

您可以使用版本来管理您的百度云CFC函数。每当您发布函数版本时，系统都会创建新的函数版本，新版本是函数未发布版本的副本。函数版本包括：

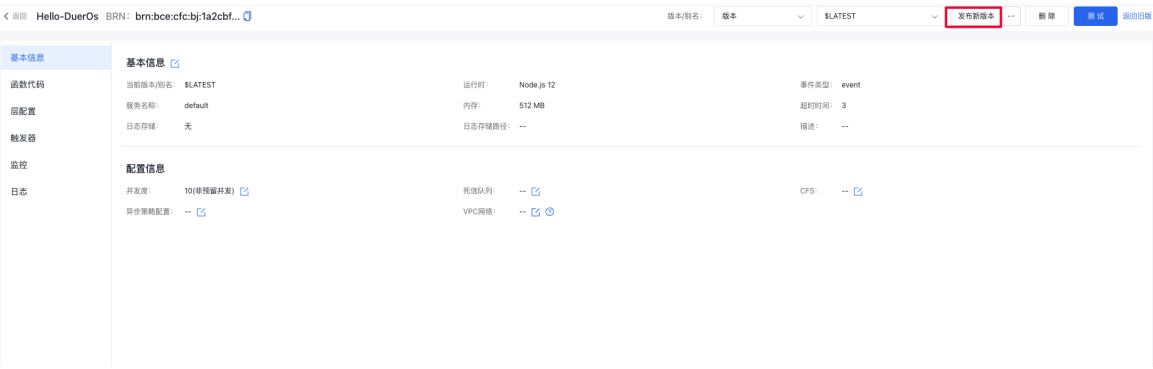
- 函数代码以及相关依赖项。
- 执行该函数的运行时。
- 函数相关设置，包括环境变量。
- 百度云函数唯一资源名（BRN）。

对于任意函数，您只能在函数未发布的版本（即\$LATEST版本）上更新代码以及更改函数设置。当您发布函数版本后，代码和大多数设置都将会被锁定，从而确保为使用该版本的用户提供一致的使用体验。相关函数信息的配置，详情可参考函数配置。

创建函数的新版本

百度云函数计算创建函数新版本步骤如下所示：

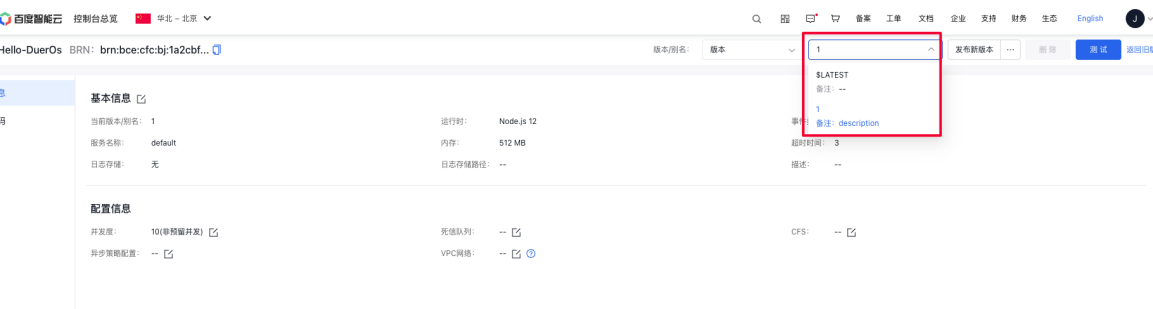
- 1. 打开[百度云控制台](#)，进入CFC控制界面。
- 2. 选择您希望发布新版本的函数。
- 3. 选择【发布新版本】即可创建新的函数版本。如下所示：



在函数版本发布时，您可以填写对应的发布信息，方便日后进行函数版本的管理。如下图所示：



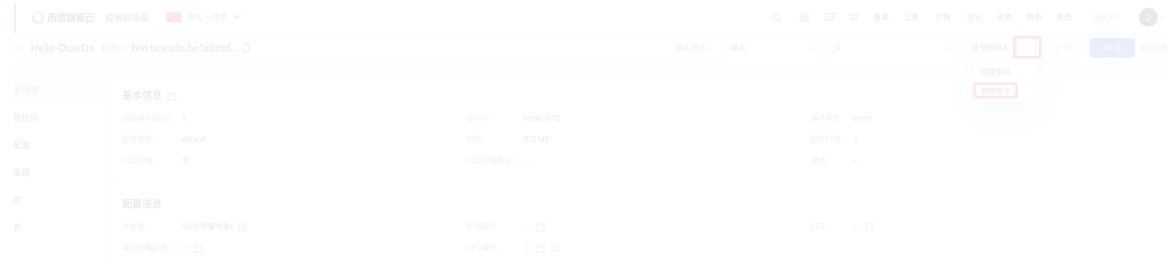
若函数版本发布成功，则可在版本列表中找到对应的函数版本。如下图所示：



删除函数版本

百度云函数计算删除函数版本步骤如下所示：

- 1. 打开[百度云控制台](#)，进入CFC控制界面。
- 2. 选择您希望删除版本的函数
- 3. 在【版本】中选择您希望删除的版本号，随后点击【删除版本】按钮，即可删除对应版本。具体步骤如下图所示：



- 注1：删除版本为不可逆操作，删除时需谨慎操作
- 注2：\$LATEST版本无法删除
- 注3: 若某一版本被别名指向，则在别名删除之前无法删除对应版本。

使用API进行版本控制

同时，您也可以使用百度云函数计算API对函数版本进行操作，详情见[版本操作](#)

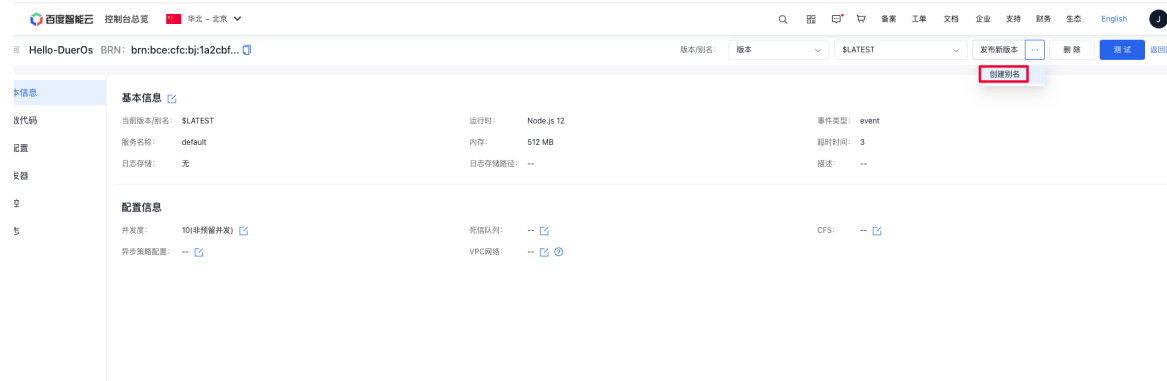
别名

您可以为您的百度云函数创建一个或者多个别名。函数别名类似一个指向特定函数版本的指针，您可以通过别名的百度云函数BRN对函数版本进行访问。

创建别名

百度云函数计算别名创建过程如下图所示：

1. 打开[百度云控制台](#)，进入CFC控制界面。
2. 选择您希望创建别名的函数。
3. 在【基本信息】中选择【创建别名按钮】。如下图所示：



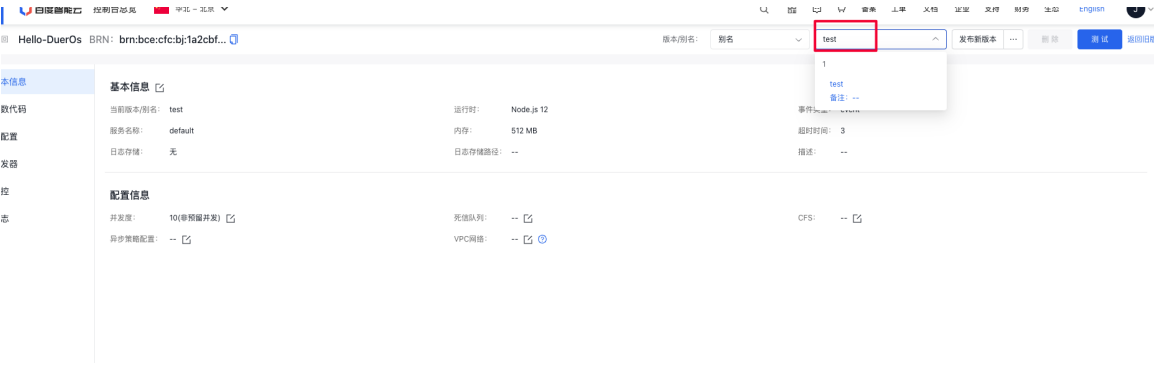
4. 输入别名名称，以及您希望指向的版本号。在百度云函数计算中，别名可以指向一个或者两个版本。若指向两个版本，则函数的调用将在两个版本之间根据权重转移流量。一般情况下，您可以使用一个版本作为函数主版本，另一个版本作为函数的灰度版本。
- 注：若别名指向两个版本，则两个版本均不能为\$LATEST。

删除别名

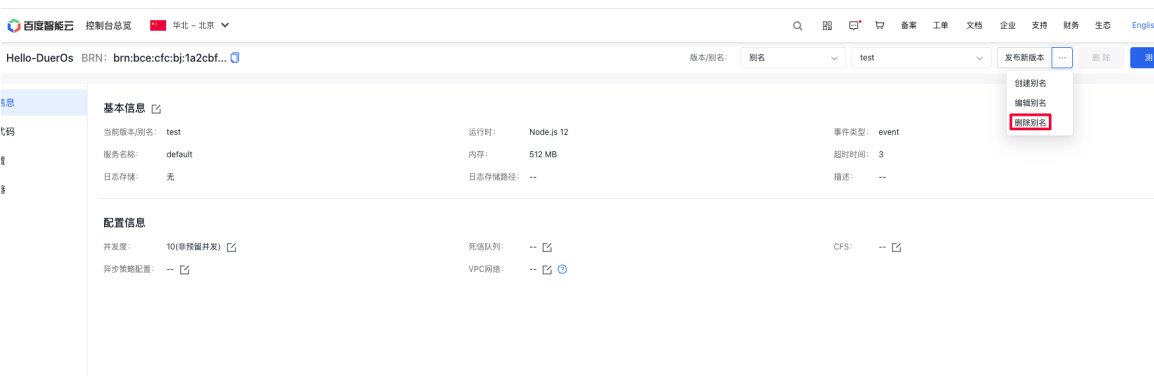
百度云函数计算删除别名过程如下所示：

1. 打开[百度云控制台](#)，进入CFC控制界面。
2. 选择您希望删除别名的函数。

3. 在【基本信息】中选择【别名】中您希望删除的别名。



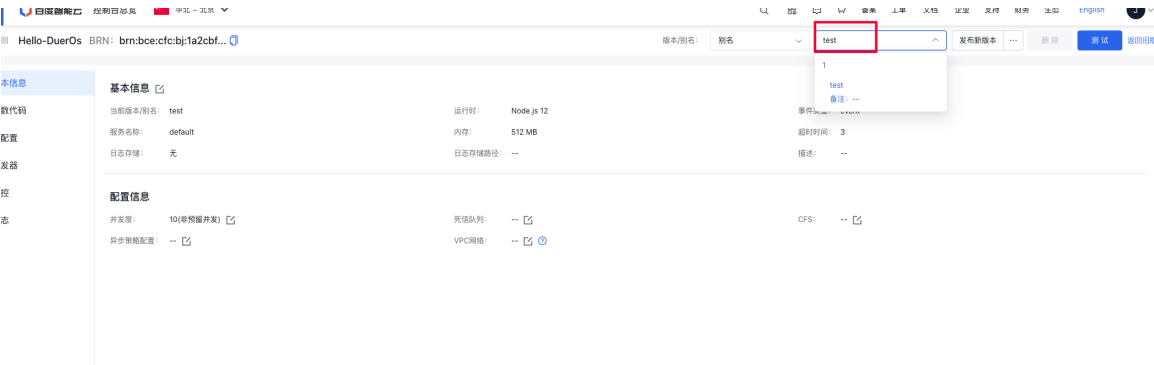
4. 点击【删除别名】按钮即可删除对应别名。



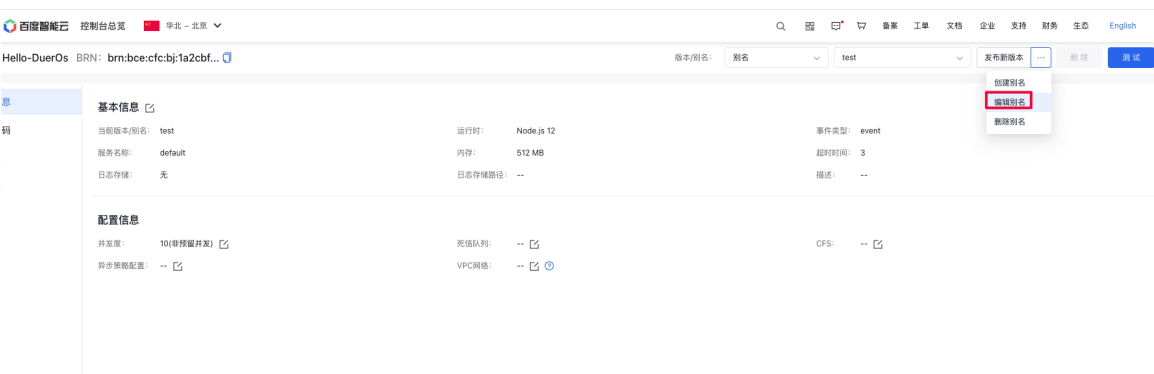
修改别名

部分情况下，您可能需要修改别名指向，或者改变灰度版本的流量等。您可以使用修改别名功能进行修改。其具体步骤如下所示：

1. 打开[百度云控制台](#)，进入CFC控制界面。
2. 选择您希望修改别名等函数。
3. 在【基本信息】中选择【别名】中您希望修改的别名。



4. 点击【编辑别名】按钮即可修改别名信息。



使用API进行别名控制

您也可以使用对应别名API进行别名控制，详情见[别名操作](#)

层依赖管理

层操作指南

层介绍

CFC的层是一种包含库或其他依赖项的zip代码包，用户可以通过本地上传和BOS地址的方式来上传层。CFC的层会将上传的zip代码包固定解压到/opt目录。每个运行时（runtime）将查找位于/opt目录下的库。

您可以配置函数使用CFC层，可以是您自己创建的层或者是CFC提供的公共层，这样您可以在函数中使用库，而不用将库包含在函数程序包之中。

使用层的好处

- 1. 使用层会使您的函数程序包更小，只关注核心代码，使开发变得更加轻松方便。
- 2. 使用层也可以避免在制作函数zip包和依赖项过程中可能出现的错误。

注意

一个函数最多可以添加5个层，函数和层的总解压大小不能超过250M的限制。

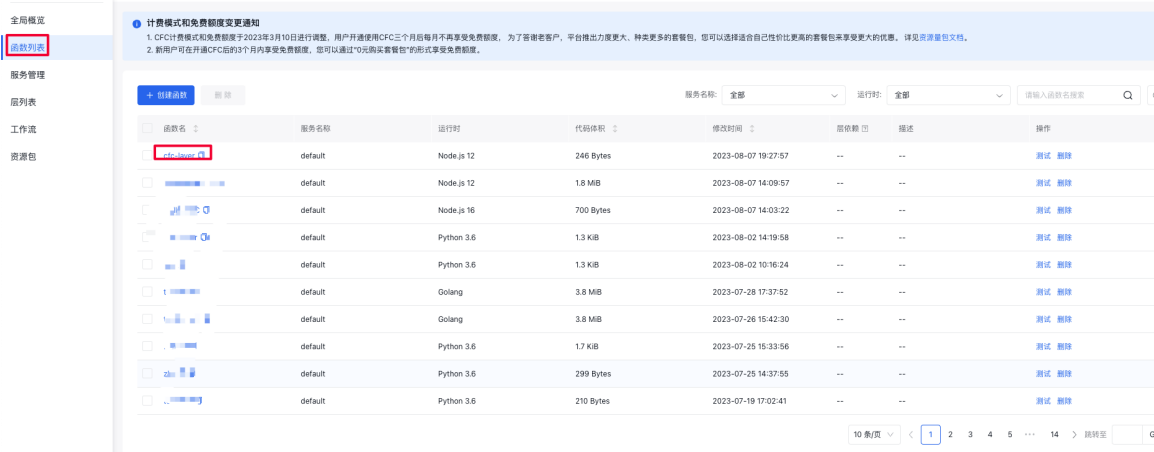
层的使用

您可以使用自己创建的层，也可以使用公共层(公共层为CFC遴选出来的常见依赖包)。
在函数创建时或者更新函数时，您可以在函数配置中指定最多5个层。

在函数中使用层

您可以在函数创建时为函数添加层，或者为已经创建好的函数添加层。要将层添加到函数之中，可以按照如下方式操作。

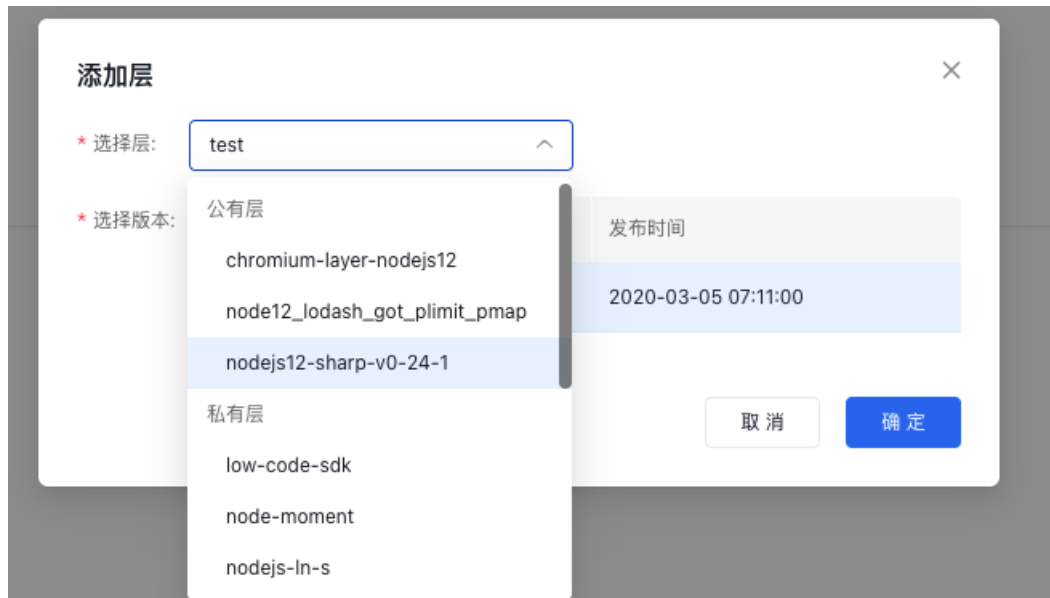
- 1.进入CFC控制台，点击函数列表，选择函数



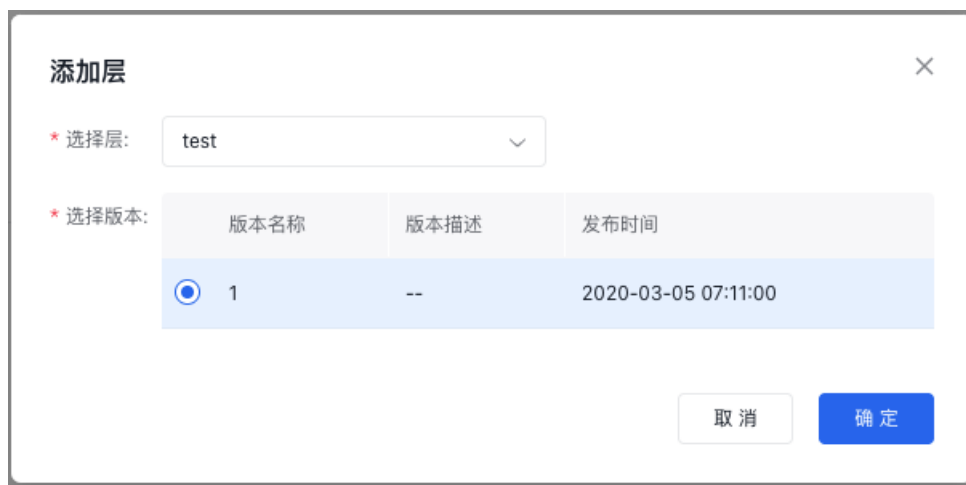
- 2.进入函数详情页面，选择层配置



3. 点击添加层，在弹窗中选择一个层和版本，可以选择公共层(CFC遴选出的基础依赖)，私有层(用户自己创建的层)



4. 选择好层之后，选择层的版本，点击确定



5. 点击确定后会进入层配置列表页，如果有多个层需要添加，那么层会按照指定的顺序合并，如果同一文件出现在多个层中，则使用最后应用的层中的版本，可以使用右侧的上下箭头调整顺序，最后点击确定。



6. 层就已经添加完成。

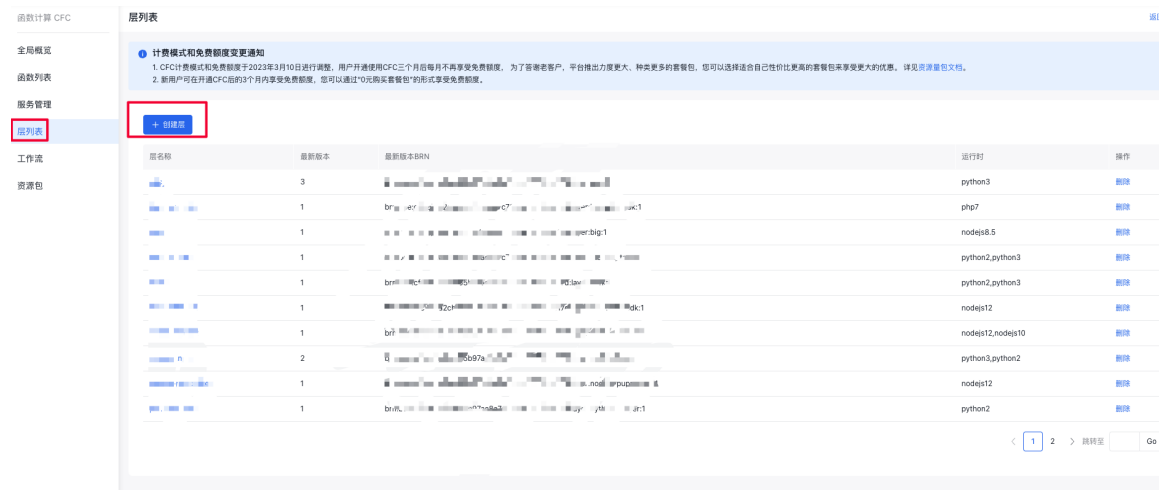
您的函数在 /opt 目录中执行期间可访问层内容。层按指定的顺序应用，这将合并任何具有相同名称的文件夹。如果同一文件出现在多个层中，则使用最后应用的层中的版本。

层的创建者可删除您使用的层的版本。但是您的函数可以继续正常运行，就好像层版本仍然存在一样。但在更新层配置时，必须先删除对已删除版本的引用。

管理层的

创建层，每次创建层时要输入名称，描述，zip压缩包。运行时列表是可选的，在未函数添加层时控制台会根据函数的运行时来选择层。如果想修改层，您必须要创建新的版本，已经创建好的层是无法修改的。

1.您可以进入CFC控制台的层列表，点击创建层来管理您创建的层



2.在弹出的对话框中填入层名称，描述，选择运行时，最多选择5个运行时，在函数中选择配置层时会根据运行时来筛选层，您可以使用zip包上传，也可以从BOS上传zip包。

基本信息

* 层名称:

cfc-layer-python

版本描述:

层

* 兼容运行时:

Node.js 12 × Node.js 14 ×

许可证:

请输入许可证

上传依赖包

* 上传类型:

上传.zip包

从BOS上传.zip包

* 函数依赖包:

+ 选择文件

完成

取消

3.创建好之后会进入层详情页面，我们可以看到版本信息，详情等资料，如果要修改层只能点击发布新版本来创建一个新层，已经发布的层是无法修改的。

< 返回

cfc-layer-python

返回层

版本列表

+ 创建新版本

版本号	描述	兼容运行时	许可证	发布时间	BRN	操作
1	层	Node.js 12, Node.js 14	--	2023-08-08 09:36:52	bm:bce:cfc:bj:1a2c8f55b97ac8a7c760c4177db4e17d:layer:cfc-layer-python:1	删除

<

1

>

4.发布新版本和创建层基本一致

基本信息

* 层名称:

cfc-layer-python

版本描述:

请输入版本描述

* 兼容运行时:

请输入兼容运行时

许可证:

请输入许可证

上传依赖包

* 上传类型:

上传.zip包

从BOS上传.zip包

* 函数依赖包:

+ 选择文件

完成

取消

5.查看层列表，可以点击层名称来查看层的所有版本

版本列表

+ 创建新版本

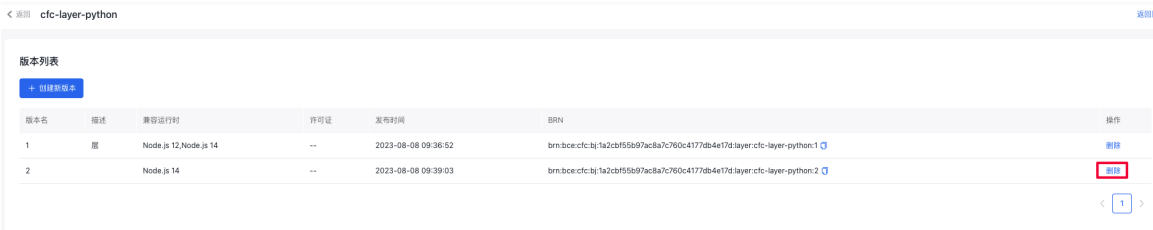
版本号	描述	兼容运行时	许可证	发布时间	BRN	操作
1	层	Node.js 12, Node.js 14	--	2023-08-08 09:36:52	bm:bce:cfc:bj:1a2c8f55b97ac8a7c760c4177db4e17d:layer:cfc-layer-python:1	删除
2		Node.js 14	--	2023-08-08 09:39:03	bm:bce:cfc:bj:1a2c8f55b97ac8a7c760c4177db4e17d:layer:cfc-layer-python:2	删除

<

1

>

6.点击删除按钮来删除层



注意 在删除一个层之后，您无法在函数中新配置该层，但是已有的函数不会受到影响。

如何在层中包含库和依赖

您可以在层中添加各种CFC支持的运行时的依赖项，这样可以把这些依赖项从函数代码中移除。CFC的运行时包括了/opt目录中的路径，您可以在函数中直接访问库和依赖。

将库和依赖放入层中，您需要将库和依赖放入指定的文件夹中。

- nodejs nodejs/node_modules,nodejs/node8/node_modules

```
nodejs_layer.zip
└── nodejs/node_modules/bce-sdk-js
```

- python python,python/lib/python3.8/site-packages

```
python_layer.zip
└── python/bce-sdk-python
```

- 全部语言 bin(PATH) ,lib(LD_LIBRARY_PATH) 您可以把bin和lib放到对应的目录

```
mybin.zip
└── bin/mybin
```

示例

- 使用CFC层管理依赖包-Python
- 使用CFC层管理依赖包-Node.js.md

使用CFC层管理依赖包-Node.js

现在 CFC 新增了“层(Layer)” 功能使得可以单独上传并集中管理CFC的依赖包了，这样开发者开发函数代码的时候只需要关注该函数本身的代码，依赖包只需要引用即可。这样有以下的好处。开发者可以只关注函数代码，不用考虑依赖包的部署。

Node.js依赖包制作

1.打包 比如要将lodash打包在层中，我们先创建一个目录叫my_nodejs_layer，在my_nodejs_layer目录下创建一个nodejs目录，此时目录结构如下

```
→ my_nodejs_layer tree
.
└── nodejs

1 directory, 0 files
```

进入nodejs目录，执行 `npm install lodash` 再查看一下目录结构

```
→ my_nodejs_layer tree -d
```

```
.
├── nodejs
│   └── node_modules
│       └── lodash
│           └── fp
```

4 directories

在my_nodejs_layer目录中将nodejs打包，

```
zip -r nodejs_layer_lodash.zip nodejs
```

这样我们的layer依赖就打好了。

CFC运行时会将层放在/opt 目录下并根据不同运行环境下的目录来调用，需要按照如下的结构来打包

- nodejs nodejs/node_modules,nodejs/node8/node_modules

```
...
my_nodejs_layer.zip
├── nodejs/node_modules/lodash
...
```

2.创建层 到CFC的层控制台，创建一个新的层

输入层，描述，和lodash的LICENSE <https://raw.githubusercontent.com/lodash/lodash/4.17.15-npm/LICENSE> 选择刚才的zip包上传，点击创建

基本信息

* 层名称:

nodejs_lodash

版本描述:

nodejs lodash

* 兼容运行时:

请输入兼容运行时

许可证:

https://raw.githubusercontent.com/lodash/lodash/4.17.15-npm/LICENSE

上传依赖包

* 上传类型:

上传.zip包

从BOS上传.zip包

* 函数依赖包:

+ 选择文件

nodejs_layer_lodash.zip

完成

取消

这样层就创建成功了。以后如果要更新，在层详情里点“创建版本”进行新建一个层。

< 返回

nodejs_lodash

返回

版本列表

+ 创建新版本

版本名	描述	兼容运行时	许可证	发布时间	BRN	操作
2	nodejs lodash	Node.js 12	https://raw.githubusercontent.com/lodash/lodash/4.17.15-npm/LICENSE	2023-08-08 10:58:56	brn:bce-cfc:bj-1a2cbf55b97ac8a7c760c4177db4e17d:layer:nodejs_lodash:2	删除

<

1

>

注意层版本是不可修改的，如果一个版本被删除，已经引用了该版本的CFC函数依然是能工作的，但新建函数中是不能去引用这个被删除的层版本。

3.创建函数 到函数菜单下创建新的函数，选择语言nodejs12

✓ 选择模版

2 函数配置

基础信息

* 函数名称:

只能由数字、字母、-或_组成，长度小于65个字符，且在当前地域内唯一

* 服务名称: [管理服务](#)

* 事件模式:

* 运行时:

* 内存: 512

128 576 1024

* 超时时间:

最大限额300秒

日志存储:

描述:

☒ 同意《百度智能云用户协议》

上一步

下一步

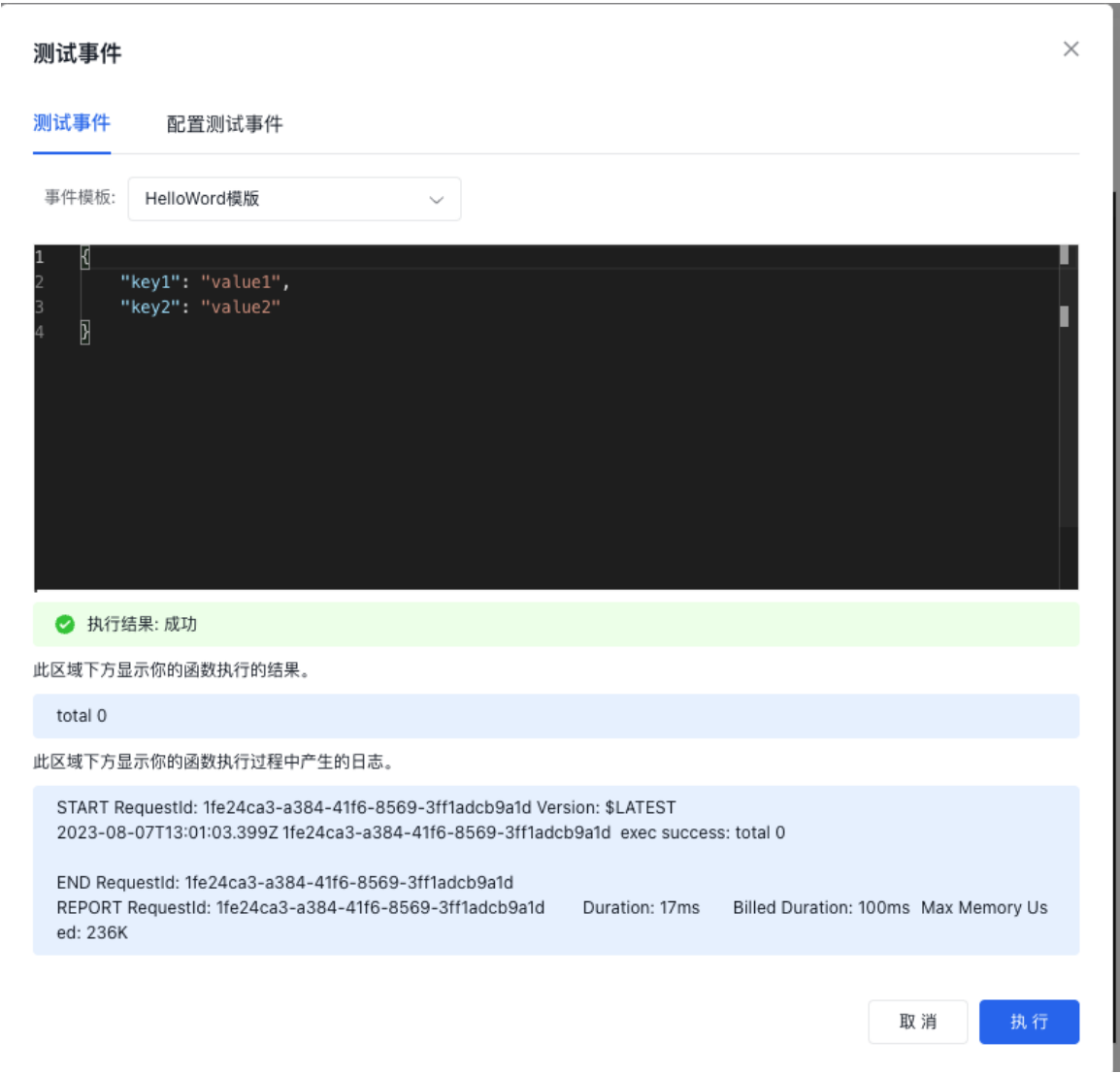
提交

修改代码如下

```
var process = require('child_process');

exports.handler = (event, context, callback) => {
  process.exec('ls -l /opt', function (error, stdout, stderr) {
    if (error !== null) {
      console.log('exec error: ' + error);
      callback(null, error);
    }else{
      console.log('exec success: ' + stdout)
      callback(null, stdout);
    }
  });
};
```

此时层还没有引入函数，先运行一下看看

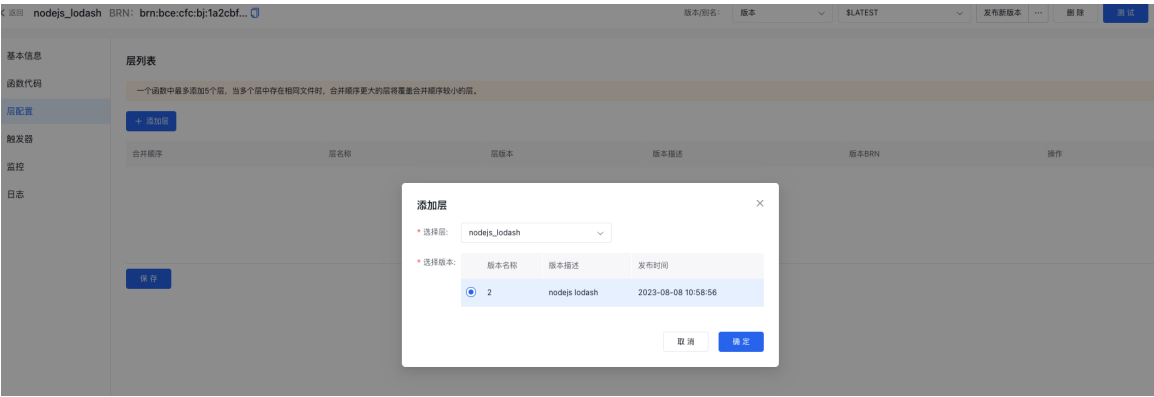


/opt 目录下没有内容

4.引入layer，在函数的层配置里点击添加层



在弹窗中选择刚刚创建的层nodejs-lodash 选择版本，点击确定



最多可以添加5个层，如果有多个层，如果有同名文件存在序号大的会覆盖序号小的。可以点击上下箭头修改顺序

nodejs_lodash

BRN: brn:bce:cf:bj:1a2cbf...

版本/别名: 版本 SLATEST 发布新版本 ... 删除 测试 返回

基本信息

函数代码

层配置

触发器

监控

日志

层列表

一个函数中最多添加5个层。当多个层中存在相同文件时，合并顺序更大的层将覆盖合并顺序较小的层。

合并顺序

层名称

层版本

版本描述

版本BRN

操作

1	nodejs_lodash	2	nodejs_lodash	brn:bce:cf:bj:1a2cbf55b97ac8a7c760c4177db4e17d:layer:nodejs_lodash:2	上移 下移 删除
---	---------------	---	---------------	--	----------

保存

确定层之后，点击保存，保存层配置。

5.测试 此时执行函数的时候，层会被解压到/opt目录下，修改一下函数代码,增加 require lodash

```
var process = require('child_process');
var _ = require('lodash');

exports.handler = (event, context, callback) => {
  process.exec('ls -l /opt', function (error, stdout, stderr) {
    if (error !== null) {
      console.log('exec error: ' + error);
      callback(null, error);
    }else{
      console.log('exec success: ' + stdout)
      console.log(_.chunk(['a', 'b', 'c', 'd'], 2))
      callback(null, stdout);
    }
  });
};
```

nodejs_lodash

BRN: brn:bce:cf:bj:1a2cbf...

版本/别名: 版本 \$

基本信息

函数代码

层配置

触发器

监控

日志

函数代码

编辑类型: 在线编辑 上传函数.ZIP包 从BOS上传.ZIP包

当前代码:

```
1 var process = require('child_process');
2 var _ = require('lodash');
3
4 exports.handler = (event, context, callback) => {
5   process.exec('ls -l /opt', function (error, stdout, stderr) {
6     if (error !== null) {
7       console.log('exec error: ' + error);
8       callback(null, error);
9     }else{
10      console.log('exec success: ' + stdout)
11      console.log(_.chunk(['a', 'b', 'c', 'd'], 2))
12      callback(null, stdout);
13    }
14  });
15 };
```

保存

重置

此函数可能包含外部库。点击下载完整 ZIP 程序包

执行函数

测试事件



测试事件

配置测试事件

事件模板: HelloWord模版

```
1
2  "key1": "value1",
3  "key2": "value2"
4
```

✓ 执行结果: 成功

此区域下方显示你的函数执行的结果。

```
total 0
drwxr-xr-x 3 500 500 60 Aug 7 12:58 nodejs
```

此区域下方显示你的函数执行过程中产生的日志。

```
START RequestId: 052fb57f-2ccb-43a3-a8bf-e893afb0f719 Version: $LATEST
2023-08-07T13:03:20.863Z    052fb57f-2ccb-43a3-a8bf-e893afb0f719 exec success: total 0
drwxr-xr-x 3 500 500 60 Aug 7 12:58 nodejs

2023-08-07T13:03:20.865Z    052fb57f-2ccb-43a3-a8bf-e893afb0f719 [ [ 'a', 'b' ], [ 'c', 'd' ] ]
END RequestId: 052fb57f-2ccb-43a3-a8bf-e893afb0f719
REPORT RequestId: 052fb57f-2ccb-43a3-a8bf-e893afb0f719   Duration: 136ms   Billed Duration: 200ms   Max Memory Us
ed: 232K
```

取消

执行

能看到输出了`_chunk(['a', 'b', 'c', 'd'], 2)`结果，`/opt`下存在`nodejs`目录。这个就是保存依赖包的目录。

使用CFC层管理依赖包-Python

现在 CFC 新增了“层(Layer)”功能使得可以单独上传并集中管理CFC的依赖包了，这样开发者开发函数代码的时候只需要关注该函数本身的代码，依赖包只需要引用即可。这样有以下的好处。开发者可以只关注函数代码，不用考虑依赖包的部署。

Python依赖包制作

1.打包 先把要共享的代码保存在本地电脑的python目录下，本例中创建文件名为`layer_test.py`，代码如下

```
var_test_layer = "this is cfc layer"
```

然后把`layer_test.py`制作成zip压缩包，需要注意目录结构，本例子中打完包的目录结构如下

```
zip -r layer_test.zip *
adding: python/ (stored 0%)
adding: python/layer_test.py (deflated 8%)
```

```
tree
```

```
.
├── layer_test.zip
└── python
    └── layer_test.py
```

1 directory, 2 files

CFC运行时会将层放在/opt 目录下并根据不同运行环境下的目录来调用，需要按照如下的结构来打包

- python python,python/lib/python3.8/site-packages

```
...
python_layer.zip
├── python/bce-sdk-python
...
```

2.创建层 到CFC的层控制台，创建一个新的层

输入层，描述，选择刚才的zip包上传，点击创建

基本信息

* 层名称: python_test

版本描述: python test

* 兼容运行时: Python 2.7 × Python 3.6 × ? v

许可证: 请输入许可证

上传依赖包

* 上传类型: 上传.zip包 从BOS上传.zip包

* 函数依赖包: + 选择文件

📎 layer_test.zip

完成

取消

这样层就创建成功了。以后如果要更新，在层详情里点“创建版本”进行新建一个层。

< 返回python_test返回

版本列表

+ 创建新版本

版本号	描述	兼容运行时	许可证	发布时间	BRN	操作
1	python test	Python 2.7,Python 3.6	--	2023-08-08 09:59:38	bmr:bce:cfrc:bj:1a2cblf55b97ac8a7c760c4177db4e17d:layer:python_test:1	删除

< 1 >

注意层版本是不可修改的，如果一个版本被删除，已经引用了该版本的CFC函数依然是能工作的，但新建函数中是不能去引用这个被删除的层版本。

3.创建函数 到函数菜单下创建新的函数，选择语言python2.7

选择模版

2 函数配置

基础信息

* 函数名称:

layer_python

只能由数字、字母、-或_组成，长度小于65个字符，且在当前地域内唯一

* 服务名称:

default

管理服务

* 事件模式:

event

* 运行时:

Python 2.7

* 内存:

1285761024

512

* 超时时间:

3

最大限额300秒

日志存储:

无

描述:

请输入描述

☒ 同意《百度智能云用户协议》

上一步

下一步

提交

修改代码如下

```
##### -*- coding: utf-8 -*-
import os

def handler(event, context):
    print(os.system('ls -l /opt'))
    return "Hello World"
```

返回

layer_python

BRN: brn:bce:cfc:bj:1a2cbf...

版本/别名: 版本

基本信息

函数代码

层配置

触发器

监控

日志

函数代码

编辑类型:

在线编辑

上传函数.ZIP包

从BOS上传.ZIP包

当前代码:

```

1  # -*- coding: utf-8 -*-
2  import os
3
4  def handler(event, context):
5      print(os.system('ls -l /opt'))
6      return "Hello World"

```

此时层还没有引入函数，先运行以下看看

测试事件

测试事件

配置测试事件

事件模板:

HelloWord模版

```

1  {
2      "key1": "value1",
3      "key2": "value2"
4  }

```

成功

执行结果: 成功

此区域下方显示你的函数执行的结果。

Hello World

此区域下方显示你的函数执行过程中产生的日志。

START RequestId: 55efe4e6-8feb-41c4-b0af-1ec9a1f84b4a Version: \$LATEST

total 0

0

END RequestId: 55efe4e6-8feb-41c4-b0af-1ec9a1f84b4a

REPORT RequestId: 55efe4e6-8feb-41c4-b0af-1ec9a1f84b4a Duration: 23ms Billed Duration: 100ms Max Memory Used: 236K

取消

执行

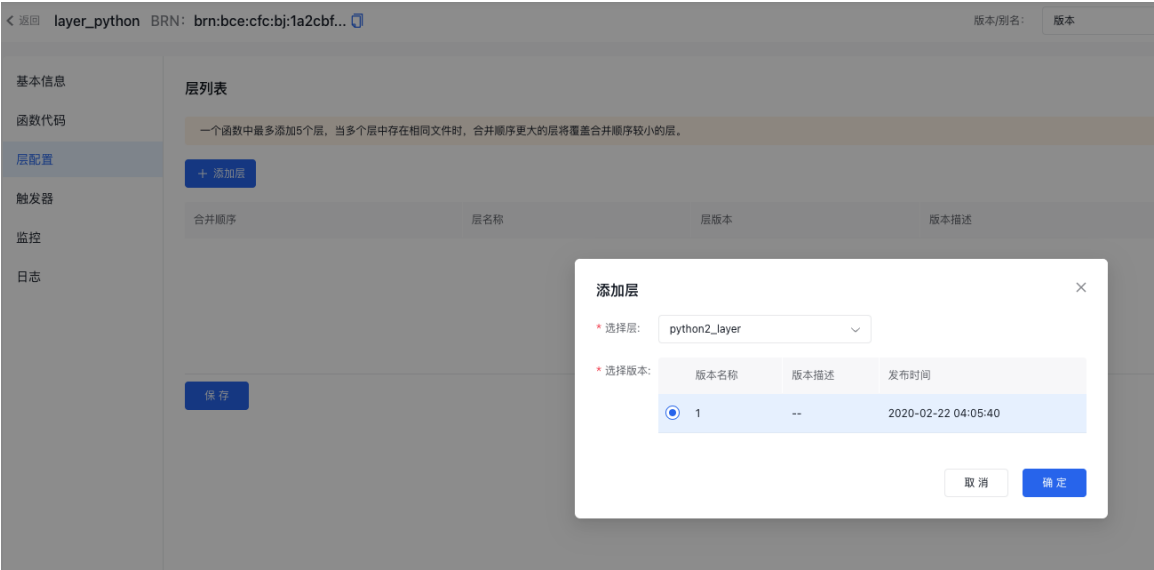
/opt 目录下没有内容

4.引入layer，在函数的层配置里点击添加层

65



在弹窗中选择刚刚创建的层python_test 选择版本1，点击确定



最多可以添加5个层，如果有多个层，如果有同名文件存在序号大的会覆盖序号小的。可以点击上下箭头修改顺序



确定层之后，点击保存，保存层配置。

5.测试 此时执行函数的时候，层会被解压到/opt目录下，修改以下函数代码

```
##### -*- coding: utf-8 -*-
import os
import layer_test

def handler(event, context):
    print(os.system('ls -l /opt'))
    print(layer_test.var_test_layer)
    return "Hello World"
```


[返回](#) layer_python BRN: brn:bce:cfc:bj:1a2cbf...

基本信息

函数代码

层配置

触发器

监控

日志

函数代码

编辑类型:

在线编辑

上传函数.ZIP包

从BOS上传.ZIP包

当前代码:

```
1  # -*- coding: utf-8 -*-
2  import os
3  import layer_test
4
5  def handler(event, context):
6      print(os.system('ls -l /opt'))
7      print(layer_test.var_test_layer)
8      return "Hello World"
```

执行函数

测试事件

测试事件

配置测试事件

事件模板:

HelloWord模版

1 "key1": "value1",
2 "key2": "value2"

✓ 执行结果: 成功

此区域下方显示你的函数执行的结果。

Hello World

此区域下方显示你的函数执行过程中产生的日志。

START RequestId: af48d082-f9dd-4426-a50c-a0bc82a4e1b3 Version: \$LATEST
total 0
drwxr-xr-x 2 500 501 36 Aug 7 12:42 python
0
this is cfc layer
END RequestId: af48d082-f9dd-4426-a50c-a0bc82a4e1b3
REPORT RequestId: af48d082-f9dd-4426-a50c-a0bc82a4e1b3 Duration: 96ms Billed Duration: 100ms Max Memory Used: 240K

取消

执行

能看到输出了层中的变量， /opt下存在python目录。这个就是保存依赖包的目录。

死信队列管理

CFC支持配置死信队列，如果用户设置了死信队列，则异步调用重试3次失败的事件会直接发送到用户的死信队列中，以供用户分析失败原因。

配置死信队列

有以下两种方式为函数配置死信队列：

- 通过API方式：在创建函数或更新函数配置接口中指定函数的DeadLetterTopic
- 通过控制台：在创建函数后，进入函数详情页，为函数添加死信队列

注意：创建死信队列前，首先需要在消息服务中创建Topic，具体请参考[百度消息服务](#)。

使用示例

使用控制台配置死信队列，进入函数详情页，点击高级配置中的死信队列选项，选择现有的topic，然后点击确定即可，如下所示



扩展函数最大超时时间

默认情况下，CFC调用函数超时时间为300s，为了满足长时应用的需求，CFC支持扩展函数最大超时时间(最大可支持1800s)，并以异步调用的方式调用最大超时时间超过300s的函数。
若您需要提高函数最大超时时间配置，您可以提工单来申请。

配置最大超时时间

前提条件

您可以根据实际需要，申请扩展函数最大超时时间。

操作步骤

- 以OpenAPI的方式创建函数[创建函数](#)，并根据实际需要设置函数超时时间Timeout。
- 异步方式调用函数，有两种异步调用方式，普通异步调用方式或为函数创建定时触发器方式

2.1、普通异步调用

在函数请求Query参数中设置InvocationType=Event，参考[函数调用](#)

2.2、以定时触发器方式触发函数异步调用

在函数创建完成后，可以为函数创建定时触发器触发函数异步调用，参考[创建定时触发器](#)

服务管理

服务

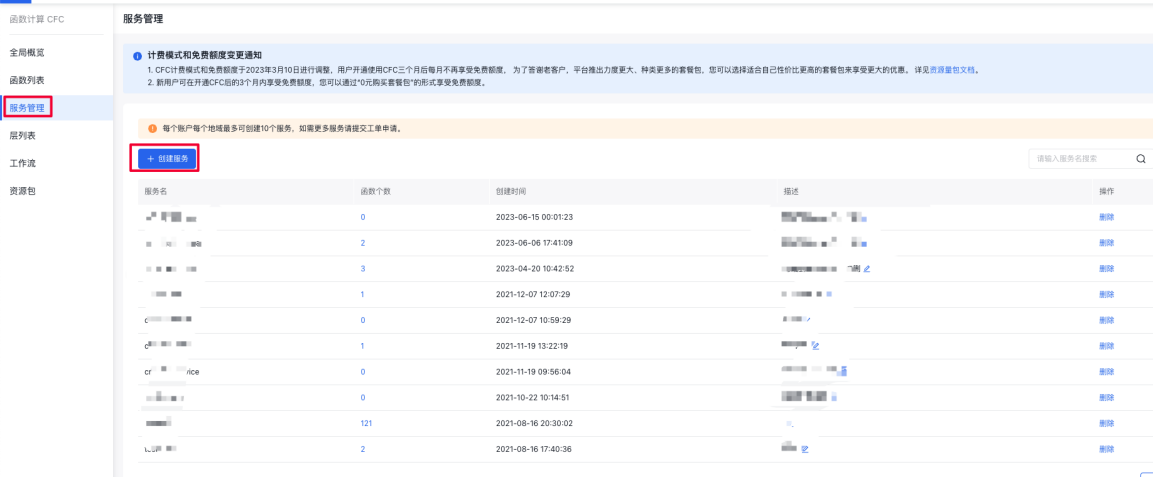
服务是函数计算资源管理的单位。从资源使用维度出发，一个服务可以由多个函数组成。函数按服务维度来拆分管理，易于不

同人员管理维护。在创建云函数时，您可以选择函数所在的服务，从而更有效地管理您的云函数。

创建服务

百度云函数计算创建服务的步骤如下所示：

- 1. 打开[百度云控制台](#)，进入CFC控制界面。
- 2. 在左侧导航栏中，点击【服务管理】进入服务管理页面。
- 3. 在服务管理区域右上角，单击【创建服务】即可创建新的服务。



- 4. 在创建服务页面，设置服务的名称，设置备注信息。备注信息便于区分服务，非必选。点击确定。

创建服务

* 服务名称:

请输入服务名称

只能由数字、字母、-或_组成，长度小于50个字符

备注:

请输入备注

取消

确定

若服务创建成功，则可在服务管理页面查看已创建的服务。每个账户每个地域最多可创建10个服务。

每个账户每个地域最多可创建10个服务。如需更多服务请提交工单申请。				
+ 创建服务				
请输入服务名称	Q			
服务名	函数个数	创建时间	描述	操作
...	0	2023-06-15 00:01:23	...	删除
...	2	2023-06-06 17:41:09	...	删除
...	3	2023-04-20 10:42:52	...	删除
...	1	2021-12-07 12:07:29	...	删除
...	0	2021-12-07 10:59:29	...	删除
...	1	2021-11-19 13:22:19	...	删除
...	0	2021-11-19 09:56:04	...	删除
...	0	2021-10-22 10:14:51	...	删除
...	121	2021-08-16 20:30:02	...	删除
test_sms	2	2021-08-16 17:40:36	测试	删除

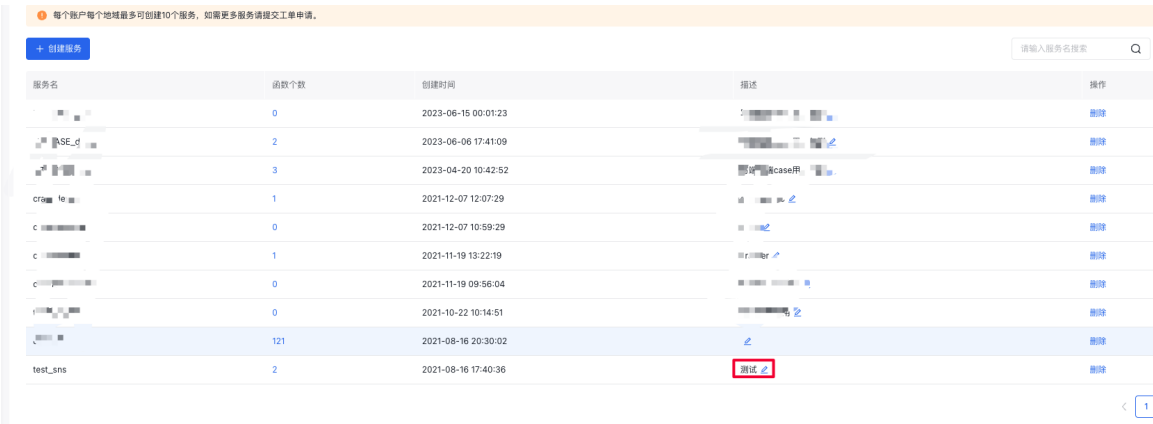
- 注1：服务名称创建后不可修改，只能由数字、字母、-或_组成，长度小于50个字符，在同一地域内唯一

- 注2：用户首次创建函数如果没有选择服务，系统会为用户创建一个名为default的服务
- 注3：服务备注长度限制100，可更新备注描述

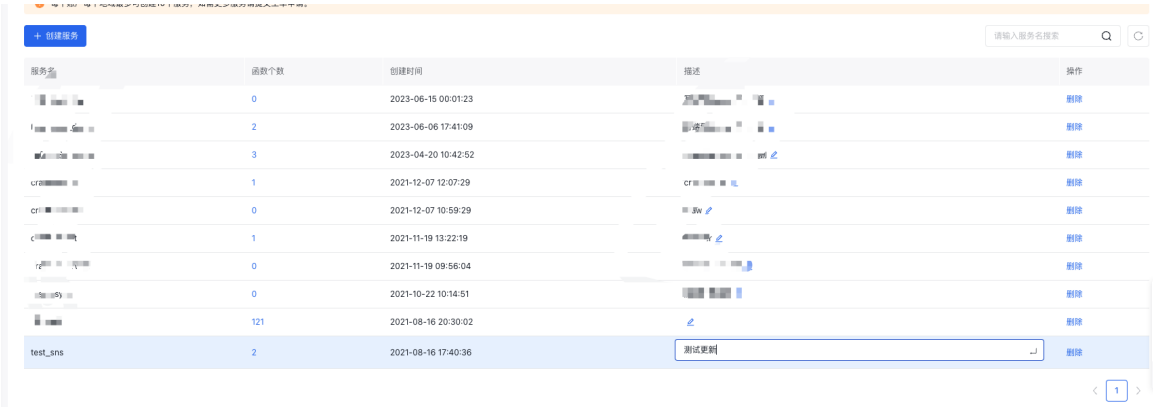
更新服务

百度云函数计算更新服务的步骤如下所示：

1. 打开[百度云控制台](#)，进入CFC控制界面。
2. 在左侧导航栏中，点击【服务管理】进入服务管理页面。
3. 在服务管理列表的目标服务行，点击【修改描述】。



4. 在修改服务描述页面，设置服务备注信息，点击回车确认。



删除服务

百度云函数计算删除函数版本步骤如下所示：

1. 打开[百度云控制台](#)，进入CFC控制界面。
2. 在左侧导航栏中，点击【服务管理】进入服务管理页面。
3. 在服务管理列表的目标服务行，点击【删除】。

+ 创建服务

请输入服务名搜索

Q

C

服务名	函数个数	创建时间	描述	操作
服务名	0	2023-06-15 00:01:23	描述	删除
服务名	2	2023-06-06 17:41:09	描述	删除
服务名	3	2023-04-20 10:42:52	描述	删除
服务名	1	2021-12-07 12:07:29	描述	删除
服务名	0	2021-12-07 10:59:29	描述	删除
服务名	1	2021-11-19 13:22:19	描述	删除
服务名	0	2021-11-19 09:56:04	描述	删除
服务名	0	2021-10-22 10:14:51	描述	删除
服务名	121	2021-08-16 20:30:02	描述	删除
test_sns	2	2021-08-16 17:40:36	描述	删除

<

1

>

4. 在弹出对话框中点击确认。

- 注1：默认服务default不能删除
- 注2：服务下有函数无法删除，需要先删除服务所属的函数

异步调用

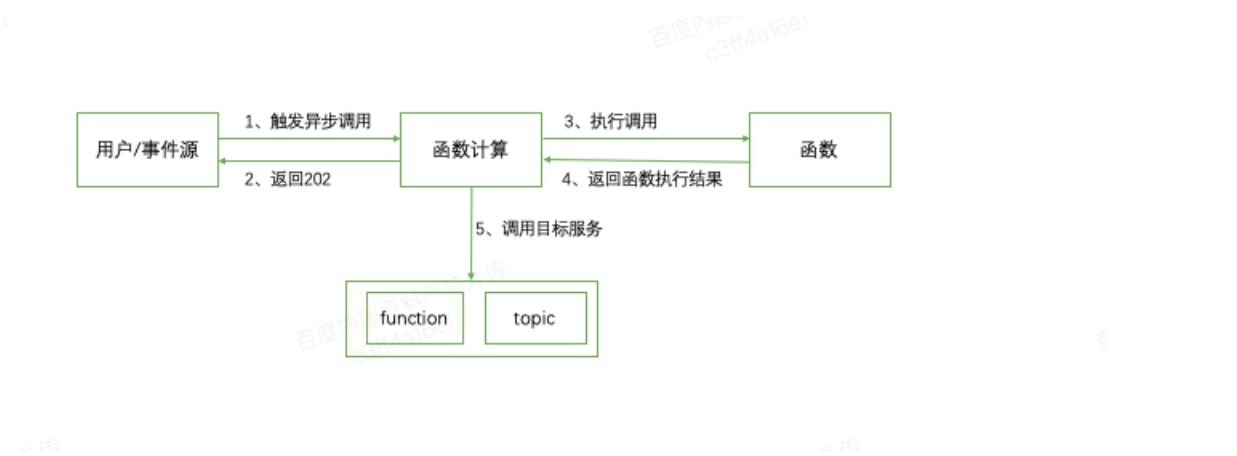
概述

CFC目前支持以下两种调用方式：

- 同步调用：调用请求被函数处理之后会直接返回结果。例如，使用控制台调用、http触发器调用等
- 异步调用：调用事件在写入到函数计算内部队列后返回结果，函数计算系统会保证该消息被可靠地处理。目前，除了API调用外，CFC支持的异步调用方式包括：[BOS触发器](#)和[定时任务触发器](#)两种触发方式。

异步执行过程

当函数被异步调用时，CFC会将用户请求先写入内部的消息队列中，并返回异步调用结果，接下来，CFC会从内部消息队列中读取消息，触发函数调用，如果函数执行失败次数超限或触发事件等待超时，则会触发目标服务（如函数或topic）。



相关概念

- 重试次数** 函数异步调用最大允许重试的次数。
- 最长保留时间** 单次请求最长被允许在内部消息队列保留的时间。
- 目标服务** 异步调用执行完成后，被调用的相关服务。

使用限制

- 最大重试次数限制为0-3。
- 最大超时时间限制为0-24小时。

- 目标服务为函数时，不能绑定函数本身。
- 支持函数执行成功或失败时触发目标服务，目标服务有且只有一个。

注意：目标服务为函数时，应避免循环调用，如函数A配置了成功调用时的异步目标为函数B，为函数B配置了成功调用时的异步目标为函数A，当异步调用触发函数A时且执行成功后，则可能出现A->B->A...循环调用的情况。

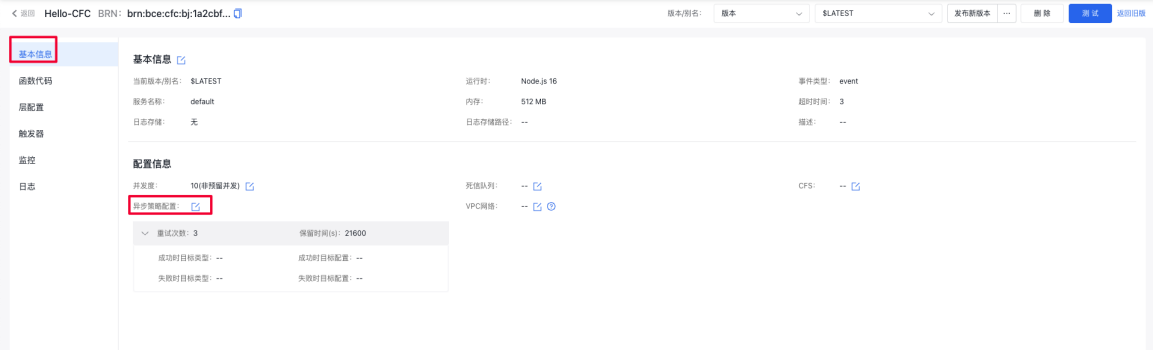
异步调用适用场景

- 不需要经常查看每个执行结果
- 任务量多且比较密集
- 存在服务调用关系

最典型的场景比如：利用Bos触发器接收到日志后需要进行日志批量处理。

异步调用配置

1.进入函数基本信息界面，打开异步策略配置



2.配置重试策略与异步执行目标

异步策略配置

* 重试次数:

3

?

* 保留时间(s):

21600

?

条件: 成功时

目标类型:

无

条件: 失败时

目标类型:

无

取消

确定

目前异步执行目标只支持CFC和kafka消息队列两种服务，当配置为其中的一种服务时

- **CFC** 当该次异步调用执行完成后，会根据满足的配置条件对配置的目标函数进行二次调用
- **Kafka** 当该次异步调用执行完成后，会将此次调用信息写入用户配置的消息队列中去

预留实例管理

预留实例支持函数实例按配置预先启动，并且函数计算平台不会主动回收这些实例，会尽可能地保障有相应数量的函数实例可以处理并发请求。

您可以通过预留实例功能，根据函数预计需要占用的资源量，为函数的指定版本设置预留实例。预留实例的执行环境是常驻的，通过配置预留实例，系统会预先进行计算资源的准备，降低冷启动、代码初始化、环境变量准备等耗时。

概述

预留实例是为了解决函数并发请求开始时的冷启动耗时较大问题，预留实例与函数版本绑定，当您为一个函数版本配置预留实例时，函数计算系统会立即启动预留实例创建，直至达到配置值。函数计算平台不会主动回收预留实例，但是，在预留实例运行的过程中可能会出现进程退出、内存超限等问题，导致实例不可用。一旦有预留实例不可用时，函数计算平台会回收不可用实例，同时准备新的预留实例，以保证系统中预留实例数与您配置数保持一致，期间可能会出现短暂时间的实际预留实例数与配置预留实例不符的情况。

预留实例只能配置在已发布版本的函数上，无法对\$LATEST版本函数进行配置。

预留实例管理

预留实例可以帮助您解决函数初始化冷启动耗时过长的问题，更快地响应请求。其中预留实例个数受函数预留并发的限制，如果所需预留实例数超过函数并发配额，您可以申请工单提高函数预留并发限额。

当您为函数配置预留实例后，函数计算系统收到函数调用请求时，会优先将请求转发给预留实例，当函数请求的峰值超过预留实例的处理能力时，剩余的请求将会转发给普通实例，由函数计算系统自动为您分配执行环境。

预留实例限制

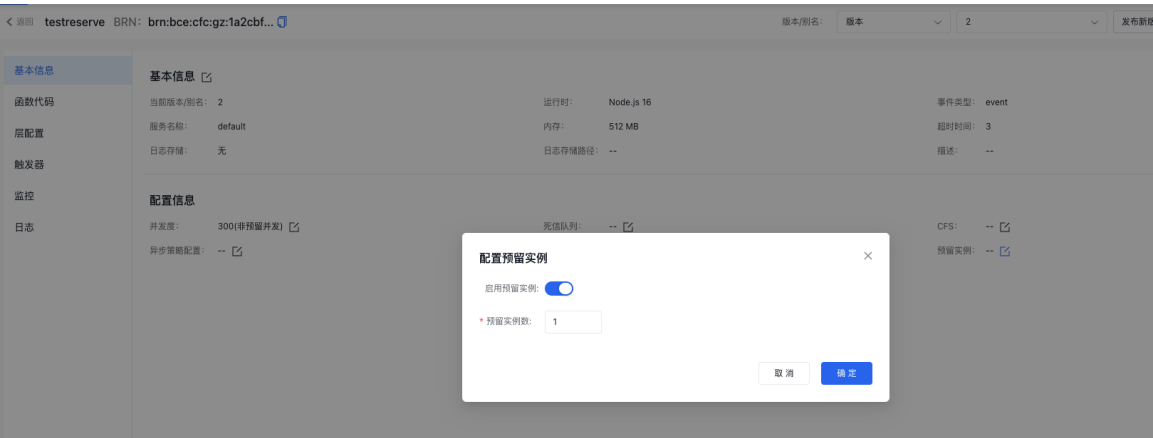
预留实例配置受限于账户维度的并发度，即同一地域的所有函数所有版本的预留实例数小于等于账户维度的并发限额。

操作步骤

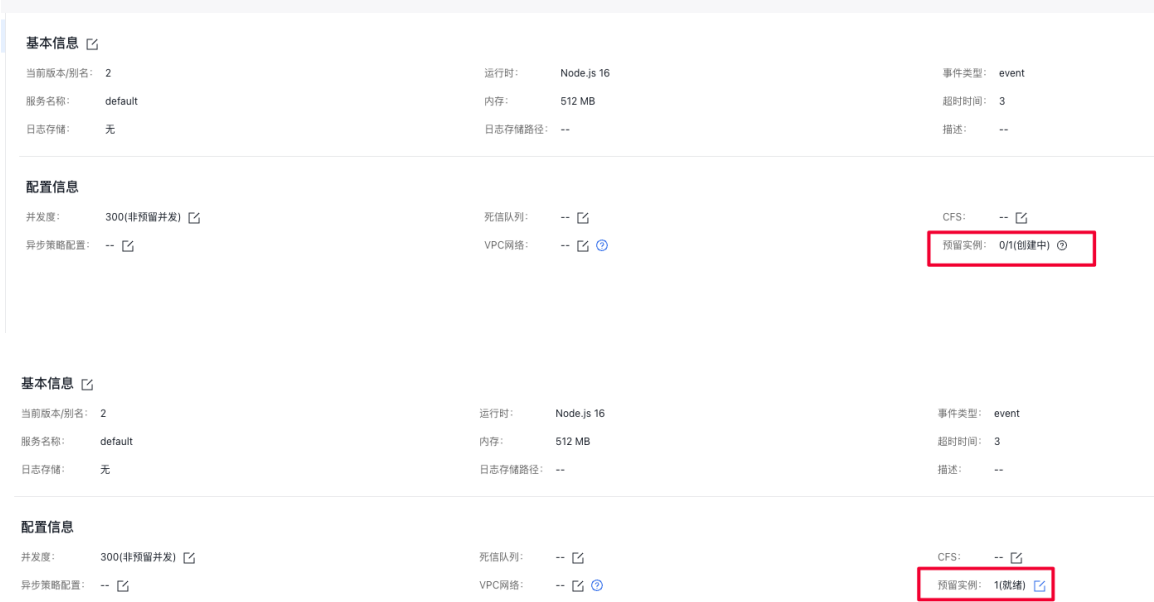
新增预留实例

对已发布版本的函数，可以创建指定数量的预留实例。

- 1、登录[管理控制台](#)，选择产品服务函数计算CFC。
- 2、点击左侧“函数列表”，进入函数列表页面。
- 3、点击“创建函数”新建函数，或选择需要创建预留实例的函数。
- 4、函数发版，或选择已发布的版本，点击进入函数详情页。
- 5、点击“预留实例”，在弹框中选择需要配置的预留实例个数，如下图所示：



- 6、点击确定后，函数计算系统则开始为函数创建预留实例，系统可能会需要一段时间完成创建，期间预留实例状态会由创建中改为就绪。



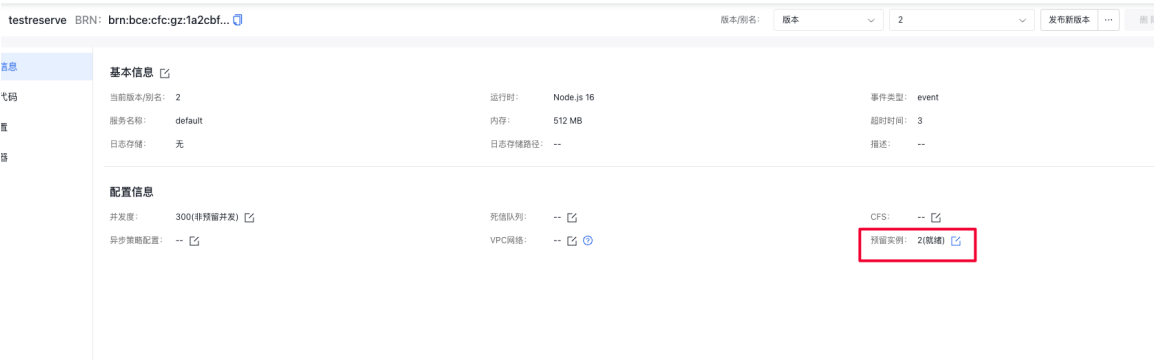
更新预留实例

当函数计算平台完成创建预留实例后，即预留实例状态为就绪时，您可以按需修改预留实例个数。

- 1、登录[管理控制台](#)，选择产品服务函数计算CFC。
- 2、点击左侧“函数列表”，进入函数列表页面。
- 3、选择需要更新预留实例的函数，点击进入函数详情页。
- 4、选择对应函数版本。
- 5、点击“预留实例”，在弹框中选择需要配置预留实例数，点击确定，则函数计算平台开始更新预留实例数。



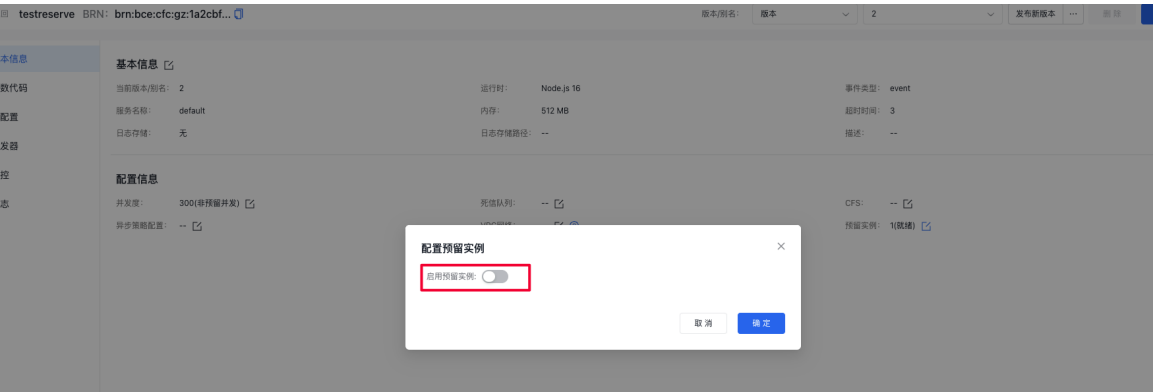
6、一段时间后，可以查看函数预留实例个数已达到预配置值。



删除预留实例

当您不再需要为函数配置预留实例时，可进行删除操作。

- 1、登录[管理控制台](#)，选择产品服务函数计算CFC。
- 2、点击左侧“函数列表”，进入函数列表页面。
- 3、选择需要更新预留实例的函数，点击进入函数详情页。
- 4、选择对应函数版本。
- 5、点击“预留实例”，在弹框中选择配置预留实例数为“无”，点击确定，即可删除预留实例配置。



自定义域名

概述

自定义域名支持用户绑定固定域名，使得服务能以用户固定域名的方式提供。您可以在函数计算创建APIGateway触发器，并在API网关侧绑定您的域名，即可实现自定义域名的功能。本文将详细介绍自定义域名的使用方法。

应用场景

自定义域名的典型应用场景是将您的应用平滑迁移到函数计算，仍然可以通过原来的域名访问该应用

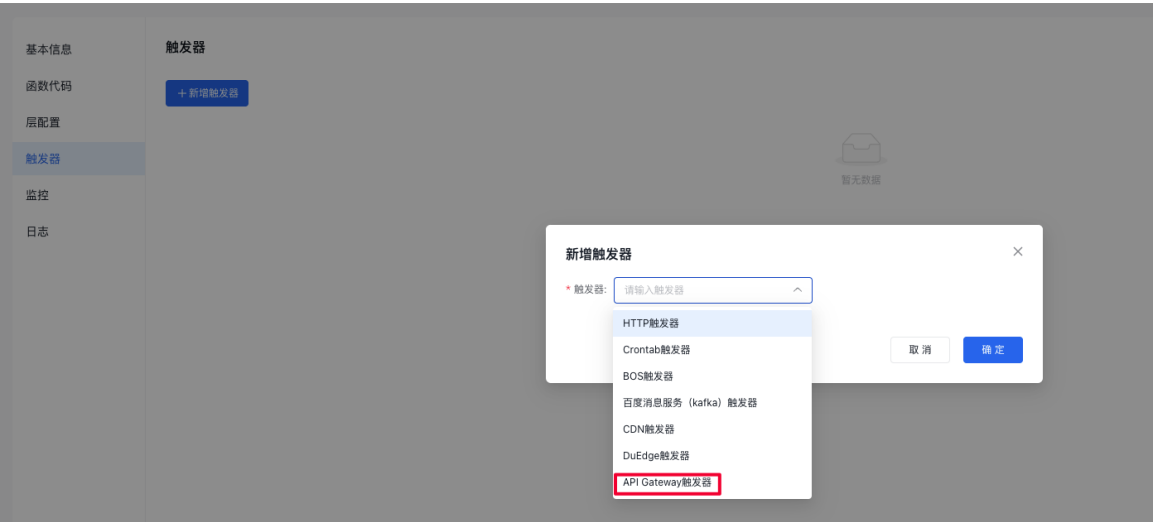
操作步骤

前提条件

- 1、确保您的域名已完成了[ICP备案](#)。这里仅要求您的域名进行ICP备案，已在其他域名注册商进行过ICP备案的域名也可以正常接入。
- 2、请确认您已经开通了[API网关服务](#)，并创建了group分组。

新增APIGateway触发器

- 1、登录[管理控制台](#)，选择产品服务函数计算CFC。
- 2、点击左侧“函数列表”，进入函数列表页面。
- 3、点击“创建函数”新建函数，或选择需要创建APIGateway触发器的函数。
- 4、选择触发器，选择APIGateway触发器，点击确认。



5、输入您创建的APIGateway触发器的相关信息，如下图所示：

新增触发器

* 触发器:

API Gateway触发器

* 触发器名称:

测试API

支持中文、英文、数字和_，不区分大小写，必须以中英文开头，长度4-100位

* 分组:

group

* HTTP方法:

GET

* 安全认证:

无认证

APP AK/SK

* 发布阶段:

☒ 无

☐ 测试

☐ 预发布

☐ 线上

* HTTP协议:

HTTP

HTTPS

取消

确定

6、点击确定，创建APIGateway触发器。未发布的API无法正常提供服务，您需要先发布您的API到线上。

- 点击发布环境中的『发布』，进入到API 列表

APIGateway触发器

删除

分组名称:

group

API名称:

测试API

APIID:

GWAI-pLWHCBmaLDC

发布环境:

无

未发布无法正常提供服务，请先发布

访问路径:

/api/v1/test-api

HTTP方法:

GET

鉴权方式:

无认证

HTTP协议:

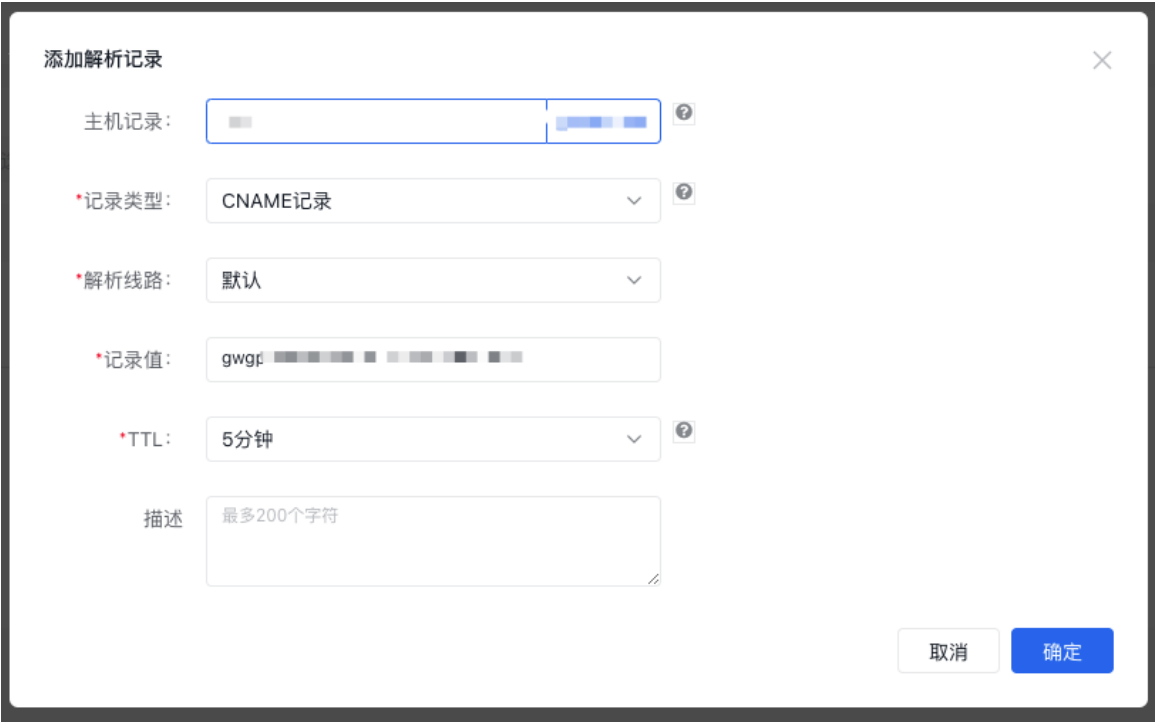
HTTP

- 选择触发器对应的API，进行发布



7、将您的域名cname到分组的二级域名，您可以在[智能云解析DNS](#)中先创建自定义域名，再添加CNAME规则，也可以在域名供应商处增加CNAME解析。

- 在自定义域名中添加解析记录，主机记录填写您的域名，记录值填写group的二级域名。



- group的二级域名，您可以点击[下面链接](#)获取。



- 默认域名为group的二级域名。



8、在APIGateway侧，绑定您的自定义域名。



- 输入您的域名。



- 自定义域名绑定成功。



访问自定义域名服务

APIGateway触发器创建后，默认路由路径为函数名称，通过“自定义域名/函数名称”访问应用。



Hello world APIGateway!

开发者指南

概述

🔗 基本概念

您在使用CFC函数支持的语言编写CFC函数代码时，需要先了解以下基本概念。

🔗 函数入口

调用CFC函数时，首先会寻找**处理程序**作为入口，执行用户代码。该**处理程序**在『函数代码』配置属性页面中『处理程序』属性进行设置，不同语言的函数的处理程序的命名方式稍有不同，您可点击"?"查看详细命名说明。



- 对于node.js函数，处理程序为您函数中的"module-name.export"值，例如，填写"index.handler"则表示调用的是index.js中的export.handler函。
- 对于python函数，处理程序为您函数中的filename.handler-method, 例如，填写index.handler则调用的是index.py文件的handler函数。
- 其他语言函数，您可以新建函数时选定开发语言后在代码编辑页面按照如上所述点击『?』查看具体命名方法。

🔗 函数入参

函数入参，是指函数在被触发调用时所传递给函数的内容。通常情况下，函数入参包括 event 入参和 context 入参两部分，但根据开发语言和环境的不同，入参个数可能有所不同，详见各语言函数开发指南。

- event入参：是一个event对象，它包含来自调用程序的信息，实现代码与触发函数的事件（event）交互，百度云CFC服务会将该event参数传递CFC函数，您编写CFC函数代码时可以从该event参数中获取信息。
- context入参：是一个上下文对象，该对象包含当前请求和运行环境的相关信息，比如当前调用的超时时间，内存限制和请求ID等。

关于函数入参，您还需了解：

- event入参和context入参均使用JSON格式统一封装。
- 不同触发器在触发函数时，传递的数据结构有所不同，详见各个触发器说明文档。
- 当CFC函数不需要任何输入时，您可以在代码中忽略event和context参数。

🔗 函数返回

CFC函数计算平台会获取到函数执行完成后的返回值，并根据调用类型进行处理。目前支持三种类型调用方式，分别是同步调用（RequestResponse）、异步调用（Event）、DryRun，其中RequestResponse方式为默认类型。

- 同步调用：当您同步调用某个CFC函数时，函数计算CFC会运行该函数，等待函数执行完毕并将运行结果等数据作为响应一并返回给调用方。
- 异步调用：当您异步调用某个CFC函数时，函数计算CFC会将调用任务放到任务队列立即返回202返回码及一个requestID，您可以通过该ID查询调用结果。
- DryRun调用: 该调用方式主要用于测试。有关更多调用函数API信息，可参考[链接](#)

异常处理

您可以按照语言特性在代码中进行异常处理，提升应用的健壮性，如显式抛出异常，CFC将捕捉这些异常并提示代码执行失败。

日志

函数调用的所有记录及函数代码的输出会存储在日志中，请使用各编程语言中的打印输出语句或者日志语句生成日志，便于调试和故障排除。如果您使用日志语句生成日志，请将日志文件修改为标准输出或标准错误即可。如果您需要查看日志，详见[查看日志](#)。

开发流程

CFC函数的开发流程主要包括以下过程：

- 编写及调试代码：根据业务需求编写函数代码，CFC提供4种方式创建和管理您的函数，详见[链接](#)
- 打包：将业务代码及其依赖打包成zip格式，打包要求及各语言部署程序包打包说明参见[链接](#)
- 部署：使用API接口创建或更新函数，也可以在CFC控制台使用生成的zip文件创建或更新函数。
- 配置触发器(可选)：如果您的函数需要被某些特定事件触发执行，则可以为函数配置对应的触发器。
- 执行函数：当触发器对应的事件发生时，函数将被调用执行，函数计算自动为其分配资源，无需您手动干预。
- 查看日志：在函数执行后，您可以在运行日志中查看函数的执行结果和监控数据。

开发语言

目前CFC支持以下开发语言，您可以根据不同开发语言特性编写应用代码，点击链接查看各语言的[开发指南示例](#)示例

- Node.js 8.5，Node.js 10，Node.js 12，Node.js 14，Node.js 16
- [Python 2.7](#)，[Python 3.6](#)，[Python 3.10](#)
- [Java8](#)
- [Golang](#)
- [PHP 7.2](#)
- [Lua 5.3](#)
- [C#](#)

注意事项

为了更好地使用CFC函数计算服务，您可能还想了解如下信息。

- 由于CFC函数的特点，您需要以无状态的风格编写函数代码，可在[无状态函数](#)页面了解更多详细信息。
- 如果您对性能优化有较高要求，您还需要了解函数冷热启动机制，可在[冷热启动机制](#)文档了解相关注意事项，以便您能从自身业务代码层面优化达到更好性能表现。
- 有关函数运行环境及安全隔离性的介绍，可见[函数运行环境及安全隔离性](#)文档
- 有关时间戳与时区，详见[时间戳和时区](#)文档

开发语言

Golang

简单示例

本示例创建一个简单的echo函数，该函数将调用事件的内容回显出来。

创建项目

```
$ mkdir echo_demo
$ cd echo_demo
$ go mod init github.com/baidubce/echo_demo
```

添加runtime依赖 go runtime运行时位于github.com/baidubce/bce-cfc-go，可以通过如下命令添加项目依赖：

```
$ go get github.com/baidubce/bce-cfc-go
```

创建Handler

函数的Handler需要实现InvokeHandler接口，该接口原型定义如下：

```
type InvokeHandler interface {
    Handle(input io.Reader, output io.Writer, context InvokeContext) error
}
```

Handle函数有三个参数：

input 用户输入的事件。

output 函数执行的结果。

context 函数执行所需的context。

该示例中定义EchoHandler，它将input事件拷贝至output，代码如下：

```
import (
    "io"
    "log"

    "github.com/baidubce/bce-cfc-go/pkg/cfc"
)

type EchoHandler struct {
}

func (h *EchoHandler) Handle(input io.Reader, output io.Writer, context cfc.InvokeContext) error {
    log.Printf("reqid=%s, brn=%s, name=%s, ver=%s, mem=%d\n",
        context.GetRequestID(), context.GetFunctionBrn(), context.GetFunctionName(),
        context.GetFunctionVersion(), context.GetMemoryLimitMB())
    credential := context.GetCredential()
    if credential != nil {
        log.Printf("ak=%s, sk=%s, token=%s\n",
            credential.AccessKeyId, credential.AccessKeySecret, credential.SessionToken)
    }
    _, err := io.Copy(output, input)
    return err
}
```

向运行时注册Handler

使用init函数，调用运行时中的RegisterNamedHandler来注册Handler，该函数原型如下：

```
func RegisterNamedHandler(name string, h InvokeHandler)
```

其中：

name 为handler名称，要与函数配置界面的处理程序保持一致。

h 实现了InvokeHandler接口的对象

在上述示例中，则需添加如下代码：

```
func init() {  
    cfc.RegisterNamedHandler("echo_handler", &EchoHandler{})  
}
```

启动运行时

需要定义函数的执行入口 **main**，并调用cfc运行时的Main函数。

```
import (  
    "flag"  
  
    "github.com/baidubce/bce-cfc-go/pkg/cfc"  
)  
  
func main() {  
    flag.Parse()  
    // 进入框架，处理函数事件  
    cfc.Main()  
}
```

调用运行时的Main函数后，运行时会自动等待事件请求。在事件请求到达时，运行时会自动调用用户注册的Handler实现功能调用。

编译代码

编译代码时，需要设置GOOS和GOARCH两个环境变量来指定对应的运行时环境。建议您在Linux环境下编译。

```
export GOOS=linux  
export GOARCH=amd64  
go build
```

函数的运行文件必须为可执行文件，通过go build编译，会产生一个二进制可执行文件，如下：

```
$ ls -lah echo_demo  
-rwxr-xr-x 1 work 2.7M 4 3 14:37 echo_demo
```

您也可以使用如下命令为您的二进制文件添加可执行权限：

```
$ chmod a+x echo_demo
```

打包输出

用户可以将生成的二进制执行文件打包到zip文件中，并上传zip包即可。建议您使用Linux的打包工具。

注意：默认情况下，用户的函数zip包中应该只包含一个可执行文件，否则函数会执行失败。如需在函数zip包中包含多个可执行文件，您需要自定义程序启动参数，详见后续说明。

自定义程序启动参数

可以通过在zip包中添加一个bootstrap脚本文件，实现自定义的程序启动参数。该文件中定义一个名为ExecStart的数组。示例

如下：

```
#####!/bin/bash

ExecStart=(
    ./echo_demo
    "--alsologtostderr"
    "-v7"
)
```

上传函数代码

在成功创建Golang的二进制zip包后，您可以将其上传至CFC函数代码。步骤如下：

1. 在百度云管理控制台中选择【产品服务】>【全部产品】>【函数计算 CFC】



2. 点击【创建函数】按钮，创建一个空白函数。函数运行时则选择Golang。

- 1 选择模板 > 2 函数配置 > 3 触发器配置

基本信息

* 函数名称:

GoTest

只能由数字、字母、-或_组成，长度小于65个字符

* 事件模式:

event

* 运行时:

Golang

* 内存(MB):

128

576

1024

128

* 超时时间(秒):

-

3

+

秒（最大限额300秒）

日志存储:

无

描述:

3. 在【函数代码】页面选择【上传函数.ZIP】包按钮，上传您打包好的二进制包。同时选择您的函数执行入口。



4. 上传完毕后，您可以使用默认测试事件测试函数是否部署成功。



示例代码

完整的示例代码[点此下载](#)

Java

函数计算支持使用Java进行编程，支持以下Java运行环境

- OpenJDK 1.8.0

Java语言由于需要编译以后才可以在JVM虚拟机中运行，因此CFC不支持在线编辑代码，仅仅支持通过web页面上上传编译好的zip包和从BOS上传zip包两种方式上传代码。

当前CFC的Java运行时支持两种事件模式，点击下面链接您可以快速查看两种模式的简单开发示例

- [event模式](#)：所有触发器类型均支持该模式
- [stream模式](#)：如果触发器类型是HTTP触发器，还支持stream模式

🔗 简单示例（event事件模式）

环境准备 确保您已经安装了JAVA和Maven。**创建项目** 本示例使用开源Java项目管理工具Maven进行管理, 使用mvn命令行工具创建了一个groupId为com.baidu.demo，artifactId为bce-cfc-java-demo简单函数项目，命令如下。

```
$ mvn archetype:generate \  
-DgroupId=com.baidu.demo \  
-DartifactId=bce-cfc-java-demo \  
-Dversion=1.0 \  
-DarchetypeArtifactId=maven-archetype-quickstart \  
-DinteractiveMode=no
```

运行完成之后，执行 `tree` 命令查看目录结构，应该是这样的

```
.  
├── pom.xml  
└── src  
    ├── main  
    │   ├── java  
    │   │   ├── com  
    │   │   │   ├── baidu  
    │   │   │   │   ├── demo  
    │   │   │   │   │   App.java  
    │   └── test  
    │       ├── java  
    │       │   ├── com  
    │       │   │   ├── baidu  
    │       │   │   │   ├── demo  
    │       │   │   │   │   AppTest.java  
    └── 11 directories, 3 files
```

编写代码

在使用 Java 编程时，您首先需要定义一个**函数入口类**。创建文件 `src/main/java/com/baidu/demo/SimpleHandler.java`，写入以下内容，以下是一个最简单的入口函数：

```
package com.baidu.demo;  
  
import com.baidubce.cfc.core.CfcContext;  
import com.baidubce.cfc.core.StreamHandler;  
  
import java.io.InputStream;  
import java.io.OutputStream;  
  
public class SimpleHandler implements StreamHandler {  
    @Override  
    public void handler(InputStream input, OutputStream output, CfcContext context) throws Exception {  
        System.out.println("console outlog");  
        System.err.println("console errlog");  
        output.write("hello world!".getBytes());  
    }  
}
```

一些说明：

- **函数入口类**需要继承并实现`com.baidubce.cfc.core.StreamHandler`接口，该接口包含一个`handler`函数，`handler`函数有三个参数：
 - `input`表示输入数据
 - `output`表示输出数据
 - `context`表示函数运行时信息,使用`context`需要在代码中导入"`com.baidubce.cfc.core.CfcContext`"
- **日志输出**到`stdout`和`stderr`，函数计算服务会自动收集这些日志。

打开文件`src/main/java/com/baidu/demo/App.java`，写入以下内容：

```
package com.baidu.demo;

import com.baidubce.cfc.core.CfcClient;

public class App
{
    public static void main(String args[])
    {
        CfcClient.main(args);
    }
}
```

在`main`入口函数中调用`CfcClient.main`进入函数运行时执行

maven编译并打包函数代码 在项目文件夹根目录创建`pom.xml`编译文件，可以在通过编辑`pom.xml`增加依赖和`plugin`配置，以下分别给出示例

增加依赖：

函数计算相关接口定义于`cfc-java-core`包中，可以通过在`pom.xml`中增加以下 `pom` 定义引入该包依赖。在 `<dependencies>` 标签下加入这一段：

```
<dependency>
  <groupId>com.baidu.cfc</groupId>
  <artifactId>cfc-java-core</artifactId>
  <version>2.1.3</version>
</dependency>
```

增加 plugin 配置

在 `pom.xml`文件中添加 `maven-assembly-plugin`，构建一个可执行`jar`包。添加`maven-dependency-plugin`，将`jar`包所依赖的其他`jar`包拷贝至`${project.build.directory}/lib`目录。在 `project` 标签下增加如下的两项：

```

<properties>
  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  <maven.compiler.source>1.8</maven.compiler.source>
  <maven.compiler.target>1.8</maven.compiler.target>
</properties>
<build>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-dependency-plugin</artifactId>
      <version>3.1.1</version>
      <executions>
        <execution>
          <id>copy-dependencies</id>
          <phase>package</phase>
          <goals>
            <goal>copy-dependencies</goal>
          </goals>
          <configuration>
            <outputDirectory>${project.build.directory}/lib</outputDirectory>
            <overwriteReleases>false</overwriteReleases>
            <overwriteSnapshots>false</overwriteSnapshots>
            <overwriteIfNewer>true</overwriteIfNewer>
          </configuration>
        </execution>
      </executions>
    </plugin>
    <plugin>
      <!-- Build an executable JAR -->
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-jar-plugin</artifactId>
      <version>3.0.2</version>
      <configuration>
        <archive>
          <manifest>
            <addClasspath>true</addClasspath>
            <classpathPrefix>lib/</classpathPrefix>
            <mainClass>com.baidu.demo.App</mainClass>
            <useUniqueVersions>false</useUniqueVersions>
          </manifest>
        </archive>
      </configuration>
    </plugin>
  </plugins>
</build>

```

发布

```

// 打包
$ mvn clean package

// zip压缩
$ cd target
$ zip -r bce-cfc-java-demo.zip bce-cfc-java-demo-1.0.jar lib/

```

配置函数

创建函数时使用如下配置：

1. 在函数管理页面，点击“创建函数”，进入创建函数页面。

2. 填写您的函数名称等基本信息，运行时选择"java8"，点击 提交。

基础信息

* 函数名称:

只能由数字、字母、-或_组成，长度小于65个字符

* 事件模式: event

* 运行时: Java 8

* 内存(MB): 512 768 1024

* 超时时间(秒): - 3 + 秒 (最大限额300秒)

日志存储: 无

描述:

提交 取消

3. 在创建成功页面，选择进入代码编辑页面

创建成功



函数已创建成功!

新建函数需要提交代码和添加触发器后，才会被调用运行。

温馨提示：感谢开通百度云服务，如您后续的操作遇到困难，您可以前往“工单”提交问题。 [快速入口>>](#)

[进入函数详情页](#) [进入代码编辑页](#)

4. 选择"上传.zip文件"，将上一步中压缩生成的zip文件上传至百度云。
5. 处理程序中填写函数入口类名称，本例中为com.baidu.demo.SimpleHandler。

TestJava 版本/别名: \$LATEST

BRN: brn:bce:cfc:bj:7a6a347619dbae6fa7e7db132e3acd55:function:TestJava:\$LATEST

函数代码

发布新版本

测试

编辑类型:

在线编辑

上传函数.ZIP包

函数程序包:

+ 选择文件

开始上传

bce-cfc-java-demo.zip

此函数可能包含外部库。 [点击下载完整 ZIP 程序包](#)

处理程序:

com.baidu.demo.SimpleHandler

确定

取消

环境变量:

编辑

🔗 进阶用法 (stream模式)

特别地，对于http trigger请求，除了可以使用普通的event模式进行处理外，还可以使用stream模式进行处理。在stream模式下，将启动一个http server接收并处理http trigger请求。**创建函数** 与普通event模式不同，需要将事件模式设置为stream。

编辑代码 下面显示了一个使用stream模式处理http trigger请求的例子，项目名称为httptrigger-demo。您首先需要定义一个**函数入口类Demo**，需要继承并实现com.baidubce.cfc.core.http.HttpRequestHandler接口，接口包含一个函数handleRequest，该函数包含三个参数 handler函数有三个参数：

- request表示请求
- response表示响应
- context表示函数运行时信息,使用context需要在代码中导入"com.baidubce.cfc.core.CfcContext"

```
package com.baidubce.httptrigger.demo;

import com.baidubce.cfc.core.http.CfcContext;
import com.baidubce.cfc.core.http.HttpRequestHandler;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;
import java.util.logging.Logger;

public class Demo implements HttpRequestHandler {
    Logger logger = Logger.getLogger("java.util.logging.ConsoleHandler");

    public void handleRequest(HttpServletRequest request,
                             HttpServletResponse response,
                             CfcContext context) throws ServletException, IOException {
        logger.info(String.format("requestid=%s", context.getRequestId()));
        logger.info(String.format("appid=%s", context.getAppID()));
        logger.info(String.format("clientip=%s", context.getClientIP()));
        response.getOutputStream().print("hello world!");
    }
}
```

因为该模式使用jetty, java runtime已经集成main函数调用，无需单独写main调用cfcclient.main。

maven打包函数代码 添加依赖。 与event模式不同，stream模式依赖bce-cfc-java-httpcore包。

```
<dependencies>
  <dependency>
    <groupId>com.baidubce.cfc</groupId>
    <artifactId>bce-cfc-java-httpcore</artifactId>
    <version>1.0.0</version>
  </dependency>
</dependencies>
```

添加 plugin配置。

在package阶段增加了copy-dependencies功能，它将函数依赖的所有jar包拷贝到target/lib目录。同时，设置入口点mainClass为com.baidubce.cfc.core.http.CfcClient。


```

<build>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-dependency-plugin</artifactId>
      <version>3.1.1</version>
      <executions>
        <execution>
          <id>copy-dependencies</id>
          <phase>package</phase>
          <goals>
            <goal>copy-dependencies</goal>
          </goals>
          <configuration>
            <outputDirectory>${project.build.directory}/lib</outputDirectory>
            <overWriteReleases>false</overWriteReleases>
            <overWriteSnapshots>false</overWriteSnapshots>
            <overWriteIfNewer>true</overWriteIfNewer>
          </configuration>
        </execution>
      </executions>
    </plugin>
    <plugin>
      <!-- Build an executable JAR -->
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-jar-plugin</artifactId>
      <version>3.0.2</version>
      <configuration>
        <archive>
          <manifest>
            <addClasspath>true</addClasspath>
            <classpathPrefix>lib</classpathPrefix>
            <mainClass>com.baidubce.cfc.core.http.CfcClient</mainClass>
            <useUniqueVersions>false</useUniqueVersions>
          </manifest>
        </archive>
      </configuration>
    </plugin>
  </plugins>
</build>

```

打包

```

$ cd target/
$ zip -r httptrigger-demo.zip httptrigger-demo-1.0.0.jar lib/

```

上传代码 在函数编辑页面上传代码，步骤与普通event模式上传方式相同

🔗 处理POJO

除了StreamHandler外，运行时还支持直接处理POJO，可以通过实现RequestHandler<REQ, RSP>接口处理POJO。 支持的类型有：String，Integer，Boolean和其他可通过json序列化的自定义类型。

🔗 日志

使用Logger输出日志

创建logger配置文件

创建logger配置文件 src/main/resources/logger.properties

```
handlers = java.util.logging.ConsoleHandler

java.util.logging.ConsoleHandler.level=INFO
java.util.logging.ConsoleHandler.formatter=java.util.logging.SimpleFormatter
```

初始化并使用logger

```
package com.baidu.demo;

import com.baidubce.cfc.core.CfcContext;
import com.baidubce.cfc.core.StreamHandler;

import java.io.InputStream;
import java.io.OutputStream;
import java.util.logging.LogManager;
import java.util.logging.Logger;

public class DemoHandler implements StreamHandler {
    static {
        try{
            InputStream input = DemoHandler.class.getResourceAsStream("/logger.properties");
            LogManager.getLogManager().readConfiguration(input);
        }catch (Exception e){
            e.printStackTrace();
        }
    }
    Logger logger = Logger.getLogger(DemoHandler.class);
    public void handler(InputStream input, OutputStream output, CfcContext context) throws Exception {
        logger.info("info log");
        logger.warning("warn log");
        output.write("hello world!".getBytes());
    }
}
```

使用log4j输出日志

增加依赖： 在pom.xml文件中，> 标签下加入这一段：

```
<dependency>
<groupId>log4j</groupId>
<artifactId>log4j</artifactId>
<version>1.2.17</version>
</dependency>
```

配置log4j 创建配置文件src/main/resources/log4j.properties

```
log4j.rootLogger = debug,stdout

**输出信息到控制台 ****
log4j.appender.stdout = org.apache.log4j.ConsoleAppender
log4j.appender.stdout.Target = System.out
log4j.appender.stdout.layout = org.apache.log4j.PatternLayout
log4j.appender.stdout.layout.ConversionPattern = [%-5p] %d{yyyy-MM-dd HH:mm:ss,SSS} method:%l%n%m%n
```

代码中使用log4j

```

package com.baidu.demo.log4j;

import com.baidubce.cfc.core.StreamHandler;
import org.apache.log4j.Logger;
import com.baidubce.cfc.core.CfcContext;
import com.baidubce.cfc.core.StsCredential;
import com.fasterxml.jackson.databind.ObjectMapper;

import java.io.InputStream;
import java.io.OutputStream;
import java.io.PrintStream;

public class DemoHandler implements StreamHandler {
    Logger logger = Logger.getLogger(DemoHandler.class);

    public DemoHandler() {
    }

    public void handler(InputStream input, OutputStream output, CfcContext context) throws Exception {
        ObjectMapper mapper = new ObjectMapper();
        InvokeEvent event = mapper.readValue(input, InvokeEvent.class);
        if (event.getEvent() != null) {
            logger.info(event.getEvent());
        }

        PrintStream stream = new PrintStream(output);
        stream.println("Hello world");
        stream.close();
    }
}

```

🔗 代码示例

- 本章节中举例的Event模式的简单示例，请点击下载 [简单示例](#)
- 本章节中举例的Stream模式示例，请点击下载[使用Stream模式](#)
- 本章节中举例的log4j输出日志示例，请点击下载 [使用log4j输出日志](#)

C#

CFC 支持 .NET Core 2.2 运行环境，支持的函数语言为 C# 和 Powershell，本文主要介绍如何使用 C# 函数。

🔗 函数入口

当创建一个基于 C# 的函数时，您需要从 Nuget 引用 BaiduBce.CFC.Core，该 SDK 中定义了一个接口，定义如下：

```

namespace BaiduBce.CFC.Core {
    public interface StreamHandler {
        void Handler(Stream input, Stream output, InvokeContext context);
    }
}

```

您需要创建一个入口类并实现这个接口，在 Handler 函数中处理您的逻辑。Handler 函数有三个参数：input 表示输入数据，output 表示输出数据，context 表示函数运行时信息。

🔗 函数代码示例

以下是一个较简单的示例，共包含三个文件。

StreamHandler.cs

创建 BaiduBce.CFC.Demo 类，实现 StreamHandler 接口，打印出 input 和 context 中的部分内容。您可以将日志输出到 stdout 和 stderr，函数计算服务会自动收集这些日志。

```
using System;
using System.IO;
using Newtonsoft.Json;
using BaiduBce.CFC.Core;

namespace BaiduBce.CFC.Demo
{
    public class StreamHandlerDemo : StreamHandler
    {
        public void Handler(Stream input, Stream output, InvokeContext context)
        {
            StreamReader reader = new StreamReader(input);
            InvokeEvent invoke = JsonConvert.DeserializeObject<InvokeEvent>(reader.ReadToEnd());
            StreamWriter writer = new StreamWriter(output);
            writer.AutoFlush = true;

            if (invoke.Event != null)
            {
                Console.Out.Write(invoke.Event);
            }

            Console.Out.WriteLine("RequestID = {0}", context.RequestID);
            Console.Out.WriteLine("FunctionBm = {0}", context.FunctionBm);
            StsCredential credential = context.Credential;
            if (credential != null)
            {
                Console.Out.WriteLine("AccessKeyID = {0}", credential.AccessKeyID);
                Console.Out.WriteLine("SecretAccessKey = {0}", credential.SecretAccessKey);
                Console.Out.WriteLine("SessionToken = {0}", credential.SessionToken);
            }

            writer.WriteLine("Hello world!");
        }
    }
}
```

InvokeEvent.cs

该文件定义了一个可以被 JSON 序列化和反序列化的对象，用于反序列化 Handler 函数的 input。您需要根据函数接收的输入 JSON 结构来定义此对象。

```
using Newtonsoft.Json;
namespace BaiduBce.CFC.Demo {
    public class InvokeEvent {
        [JsonProperty(PropertyName = "event")]
        public string Event { get; set; }
    }
}
```

Program.cs

该文件定义了程序的入口，您无需修改此文件。

```
using BaiduBce.CFC.Core;
namespace BaiduBce.CFC.Demo {
    class MainClass {
        public static void Main(string[] args) {
            CfcClient.main();
        }
    }
}
```

函数打包及上传

函数打包

在您的项目根目录下执行 `publish` 命令，dotnet 会编译程序并输出结果：

```
$ dotnet publish -c Release
```

打包输出结果：

```
$ cd bin/Release/netcoreapp2.2/publish/
$ zip -r csharp-demo.zip *
```

然后，您就可以在 CFC 的函数代码页，点击"上传函数.ZIP包"，将 `csharp-demo.zip` 上传。

Handler 命名 在创建函数时，您需要制定一个 handler 方法的字符串，用来告诉函数计算如何调用您的函数。该字符串格式如下：`AssemblyFileName::FullClassName`，其中 `AssemblyFileName` 是函数所在的 `Assembly` 的文件名（省去.dll），`FullClassName` 是该函数所在类的全名，`Namespace.ClassName`。

在上述简单示例中，如果 `Assembly` 文件名为 `csharp_demo`，则 handler 字符串为 `csharp_demo::BaiduBce.CFC.Demo.StreamHandlerDemo`。

至此，您的 `csharp` 函数就已配置完，点击"测试"即可运行函数。

Python

目前支持的Python开发语言包括如下版本：

- Python2.7
- Python3.6
- Python3.10

函数代码

一个简单的Python函数代码如下所示：

```
##### -*- coding: utf-8 -*-
def handler(event, context):
    return "Hello World"
```

执行函数

在创建CFC云函数时，需要指定执行方法。

- 普通方式: 使用Python开发语言时，执行方法格式形如您函数中 `"filename.handler-method"` 的值，例如：填写 `"index.handler"`，调用入口文件 `index.py` 中定义的 `handler` 方法。

- ZIP包上传: 当使用本地上传.ZIP包或从BOS上传.ZIP等方法提交代码ZIP包时，ZIP包的根目录下需要包含指定的入口文件，入口文件中定义入口函数，且文件名和函数名要与执行入口函数名保持一致，否则会无法找到入口文件和入口函数而报错。

🔗 输入

- event: CFC使用此参数将触发事件数据传递到处理函数，此参数形式通常为Python dict类型。
- context: CFC使用此参数向处理函数提供运行时信息，此参数形式通常为Python dict类型。

🔗 输出

在处理函数中您可以使用return来返回值，函数不同调用方式，对返回值的处理方式不同。

- 同步调用(RequestResponse): 使用同步调用时，结果会以序列化的JSON格式返回，调用方可以根据返回值进行后续处理。
- 异步调用(Event): 异步调用时，调用方法触发函数后立即返回，所以，函数返回值会被丢弃。

🔗 异常处理

如果用户函数引发异常，则CFC会将异常信息序列化为JSON格式并返回。如下所示：

```
##### -*- coding: utf-8 -*-

def handler(event, context):
    time.Sleep(5*time.Second)
    print("value1=" + event['key1'])
    return event
```

在调用此函数时，会引发异常，且返回如下错误信息：

```
{
  "stackTrace": [
    " File \" /var/task/index.py\", line 4, in handler\n    time.Sleep(5*time.Second)\n"
  ],
  "errorType": "NameError",
  "errorMessage": "global name 'time' is not defined"
}
```

获取错误信息的方式取决于函数调用方式：

- 同步调用(RequestResponse): 则会将错误信息直接返回。
如在控制台调用(同步调用)，错误信息会直接在执行结果中显示，如下图所示：

✖ 执行结果：失败

此区域下方显示你的函数执行的结果。

```
{
  "stackTrace": [
    " File \"/var/task/index.py\", line 4, in handler\n  time.Sleep(5*time.Second)\n"
  ],
  "errorType": "NameError",
  "errorMessage": "global name 'time' is not defined"
}
```

此区域下方显示你的函数执行过程中产生的日志。

```
START RequestId: 22225c39-5d13-4e40-b8fc-b3afcd5bbf6b Version: $LATEST
Traceback (most recent call last):
  File "/var/task/index.py", line 4, in handler
    time.Sleep(5*time.Second)
NameError: global name 'time' is not defined
END RequestId: 22225c39-5d13-4e40-b8fc-b3afcd5bbf6b
REPORT RequestId: 22225c39-5d13-4e40-b8fc-b3afcd5bbf6b Duration: 62ms    Billed Duration: 100ms    Max
Memory Used: 1M
```

并且在函数监控页面也可以看到调用错误结果，如下图所示：



如果配置了日志存储路径，则可以在日志流详情中查看错误信息，如下图所示：

```
日志流详情$LATEST_201912111530_dd512514de3b7b01f5fdf754b4af51bc

2019-12-11T07:33:54Z START RequestId: 22225c39-5d13-4e40-b8fc-b3afcd5bbf6b Version: $LATEST
2019-12-11T07:33:54Z Traceback (most recent call last):
2019-12-11T07:33:54Z   File "/var/task/index.py", line 4, in handler
2019-12-11T07:33:54Z     time.Sleep(5*time.Second)
2019-12-11T07:33:54Z NameError: global name 'time' is not defined
2019-12-11T07:33:54Z END RequestId: 22225c39-5d13-4e40-b8fc-b3afcd5bbf6b
2019-12-11T07:33:54Z REPORT RequestId: 22225c39-5d13-4e40-b8fc-b3afcd5bbf6b Duration: 62ms Billed
Duration: 100ms Max Memory Used: 1M
```

- 异步调用(Event): 则CFC不会返回任何信息，如果配置了日志存储路径，则错误信息会被记录到日志中，您可以在相应的日志文件中查看错误信息。

🔗 日志

您可以在程序中使用print或logging模块完成日志输出。如下代码所示:

```
##### -*- coding: utf-8 -*-
import logging
logging.basicConfig()
logger = logging.getLogger()
logger.setLevel(logging.INFO)
def handler(event, context):
    logger.info('event: %s', event)
    print(event)
    return "Hello World"
```

输出内容您可以在函数日志中查看。

🔗 依赖的第三方库

使用CFC Python开发语言创建函数时，云端支持的库如下所示：

Python3.6运行时支持的第三方库列表如下所示：

库名称	版本
baidu-aip	2.2.8.0
certifi	2019.9.11
chardet	3.0.4
idna	2.8
pip	9.0.1
requests	2.22.0
setuptools	36.6.0
six	1.11.0
urllib3	1.25.7
websocket-client	0.44.0
wheel	0.30.0

Python2.7运行时支持的第三方库列表如下所示：

库名称	版本
baidu-aip	2.2.8.0
certifi	2019.9.11
chardet	3.0.4
idna	2.8
pip	9.0.1
pycrypto	2.6.1
requests	2.22.0
setuptools	36.6.0
six	1.11.0
urllib3	1.25.7
websocket-client	0.44.0
wheel	0.30.0

Lua

CFC 支持 Lua 编程语言，Lua 函数的运行环境是 Lua5.3，本文主要介绍使用 Lua 函数的基本方法。

函数入口

使用 Lua 运行时，您需要定义一个 Lua 代码编写的函数作为执行入口。以空白函数为例，在 CFC 的页面上创建一个运行时为 Lua5.3 的空白函数后。在“函数代码”页，您可看到如下函数代码，这就是一个最简单的 Lua 函数。

```
function handler(event, context)
    return "hello world"
end
```

如上所示，这个 Lua 代码编写的函数名为 handler，默认情况下 CFC 会将代码存储在一个名为 index.lua 的文件中。CFC 用“不带类型后缀的文件名.函数名”的形式表示 Lua 函数的执行入口，在本例中就是 index.handler。

在“函数代码”页，“处理程序”即表示函数入口，您可以看到，新创建的函数已为您填入正确的入口。若您修改了函数代码和入口函数名，请相应地修改“处理程序”的值，否则函数将无法执行。



event 参数 event 参数是函数的输入参数，也是您调用函数时传入的数据，其数据类型是 table。

context 参数 context 参数是 CFC 向函数提供的运行时信息，数据类型是 table。其包含了一些环境参数，比如 functionBm，或调用函数的客户端传入的 context 参数 clientContext。

从本地或 BOS 上传 ZIP 包 当您从本地或者 BOS 上传一个函数 ZIP 包时，请注意 ZIP 包的层级目录以及函数入口。index.lua 文件应放在 ZIP 包的根目录，而不是根目录下的某个文件夹里。若您修改了文件名称，那么“处理程序”也要做对应的修改，否则 CFC 将无法找到函数入口。示例如下：

```
$ unzip -l luafunc.zip
Archive: luafunc.zip
Length Date Time Name
-----
117 12-27-2019 09:52 index.lua
0 12-27-2019 10:11 mylib/
361 12-27-2019 10:11 mylib/utis.lua
-----
478 3 files
```

🔗 依赖管理

若您的 Lua 函数需要一些依赖文件，您需要把依赖与源文件一起打包上传，函数才能执行。

对于您自己实现的依赖库，把依赖文件或文件夹与 `index.lua` 放置在同级目录，然后一起打包上传即可。

[Luarocks](#) 是一个常用的 Lua 包管理工具，CFC 支持以 Luarocks 的默认组织路径去查找 Lua5.3 的依赖文件。因此您可以在本地使用 Luarocks 安装依赖后，将源文件与依赖直接打包上传，而无需更改依赖文件的路径。示例如下：

```
$ tree
.
├── index.lua
├── mylib
│   └── utils.lua

$ luarocks install --tree `pwd` lpeg # 在当前路径安装依赖
Installing https://luarocks.org/lpeg-1.0.2-1.src.rock
...

$ tree
.
├── index.lua
├── lib
│   ├── lua
│   │   └── 5.3
│   │       └── lpeg.so
│   └── luarocks
│       ├── rocks-5.3
│       │   ├── lpeg
│       │   │   └── 1.0.2-1
│       │   │       ├── lpeg-1.0.2-1.rockspec
│       │   │       └── rock_manifest
│       │       └── manifest
├── luafunc.zip
├── mylib
│   └── utils.lua
├── share
│   └── lua
│       └── 5.3
│           └── re.lua

$ zip -r luafunc.zip * # 将此目录下所有文件打包
```

如上所示，自己实现的依赖位于 `mylib/` 中，Luarocks 安装的依赖分别在 `lib/lua/5.3/` 和 `share/lua/5.3/` 中。把 `luafunc.zip` 上传到 CFC 后，您可以在 `index.lua` 中可以这样引用依赖：

```
require("re")
require("mylib/utils")
```

请注意，Luarocks 安装的一些依赖包中有 `.so` 文件，您在 Windows 或 MacOS 环境中安装的依赖，可能无法在函数的 Linux 执行环境中直接使用。为了避免此类问题，推荐您使用 CFC [本地开发工具 BSAM](#) 在本地调试代码、安装依赖，避免本地与云端不兼容问题。

PHP

🔗 函数代码

以下为一个最基本的 hello world 函数入口示例：

```
<?php
function handler($event, $context) {
    var_dump($event);
    return "hello world!";
}
```

函数入口

在创建CFC云函数时，需要指定执行方法。

- 普通方式：handler需要与函数代码处的 "handler" 字段相对应。例如，函数代码处指定的 "handler" 为 index.handler，函数计算则会加载 index.php 文件中定义的 handler 函数。
- ZIP包上传：当使用本地上传.ZIP包或从BOS上传.ZIP等方法提交代码ZIP包时，ZIP包的根目录下需要包含指定的入口文件，入口文件中定义入口函数，且文件名和函数名要与执行入口函数名保持一致，否则会无法找到入口文件和入口函数而报错。

输入

PHP 环境下的入参包括 \$event 、 \$context。

\$event：使用此参数传递触发事件数据。

\$context：使用此参数向您的处理程序传递运行时信息。

输出

在处理函数中您可以使用return来返回值，函数不同调用方式，对返回值的处理方式不同。

同步调用(RequestResponse): 使用同步调用时，结果会以序列化的JSON格式返回, 调用方可以根据返回值进行后续处理。

异步调用(Event): 异步调用时，调用方法触发函数后立即返回，所以，函数返回值会被丢弃。

日志

您可以在程序中使用如下语句来完成日志输出：

- print_r()
- var_dump()

输出内容您可以在函数日志中查看。

错误处理

PHP 函数在执行过程中发生异常时，函数计算捕获异常并返回异常信息。以下示例代码返回了 panic 的异常信息：

```
<?php
function handler($event, $context) {
    throw new Exception("panic");
}
```

根据以上示例代码，您调用函数时可能收到如下响应信息：

```
panic
Stack Trace:
##### 0 [internal function]: handler(Array, Array)
##### 1 phar:///var/runtime/script/php7-runtime.phar/src/CfcClient.php(89): ReflectionFunction->invoke(Array, Array)
##### 2 phar:///var/runtime/script/php7-runtime.phar/src/CfcClient.php(46): BaiduBce\CFC\CfcClient-
>do_invoke(Array)
##### 3 phar:///var/runtime/script/php7-runtime.phar/bin/php7-runtime(29): BaiduBce\CFC\CfcClient->wait_invoke()
##### 4 /var/runtime/script/php7-runtime.phar(12): require('phar:///var/run...')
##### 5 {main}
```

🔗 已安装扩展

如下列出目前已安装的 PHP 扩展供参考：

- Core
- date
- libxml
- openssl
- cre
- qlite3
- lib
- alendar
- type
- url
- dom
- hash
- fileinfo
- filter
- ftp
- gettext
- SPL
- iconv
- json
- mbstring
- session
- pcntl
- PDO
- standard
- pdo_sqlite

- Phar
- readline
- Reflection
- mysqlnd
- shmop
- SimpleXML
- sockets
- pdo_mysql
- exif
- sysvmsg
- sysvsem
- sysvshm
- tokenizer
- xml
- xmlreader
- xmlwriter
- zip
- mysqli
- redis
- imagick
- Zend OPcache

您也可以随时在函数中通过 `print_r(get_loaded_extensions());` 代码打印查看已安装的扩展。

🔗 代码打包

用户可以将生成的二进制执行文件打包到zip文件中，并上传zip包即可。具体步骤可以参考[创建部署程序包](#)。

下面以一个简单的hello world功能为例。

1. 创建printhello函数的代码文件。

打开文本编辑器，编写helloworld函数，代码如下，文件命名为helloworld.php，保存文件。

```
<?php
function printhello() {
    var_dump('Hello world!');
}
```

2. 创建handler函数的代码文件。

打开文本编辑，定义函数，代码如下，文件命名为index.php，保存文件（与helloworld.php保存在同一文件夹下）。

```
<?php
include_once 'helloworld.php';

function handler($event, $context) {
    $output = json_encode($event);
    printhello();
    return $output;
}
```

3. 打包代码。

函数的代码文件创建以后，可以在目录下看到helloworld.php和index.php两个文件。选中这两个文件，打包命名为helloworld.zip，在【函数代码】页面点击【上传函数.ZIP包】上传。

编辑类型：

[在线编辑](#) [上传函数.ZIP包](#) [从BOS上传.ZIP](#)

当前代码：

```
1 <?php
2 function handler($event, $context) {
3     var_dump($event);
4     return "hello world!";
5 }
```

[保存](#) [重置](#)

此函数可能包含外部库。[点击下载完整 ZIP 程序包](#)

4. 测试函数

在【函数代码】页面点击【测试】，新对话框中事件模板保持默认的"HelloWorld模板"，然后点击执行。

测试事件 ✕

当前测试事件

配置测试事件

事件模板：

HelloWord模版

* 事件名称：

请输入事件名称

只能由数字、字母或下划线组成

1 {

2 "key1": "value1",

3 "key2": "value2"

4 }

创建

执行

取消

在对话框下方将显示如下运行日志：

✓ 执行结果：成功

此区域下方显示你的函数执行的结果。

```
{"key1":"value1","key2":"value2"}
```

此区域下方显示你的函数执行过程中产生的日志。

```
START RequestId: 6bc882d5-a2ea-485f-9373-850d5fac94a9 Version: $LATEST
string(12) "Hello world!"
END RequestId: 6bc882d5-a2ea-485f-9373-850d5fac94a9
REPORT RequestId: 6bc882d5-a2ea-485f-9373-850d5fac94a9  Duration: 95ms    Billed Duration: 100ms
Max Memory Used: 916K
```

Node.js

目前CFC支持的Node.js开发语言包括如下版本：

- Node.js 16.19.1
- Node.js 14.21.3
- Node.js 12.2
- Node.js 10.15
- Node.js 8.5（已下线，目前存量Node.js 8.5函数仍可运行，但无法创建该运行时的新CFC函数）

函数代码

一个简单的Node.js函数示例代码如下所示：

```
exports.handler = (event, context, callback) => {  
  callback(null, "Hello world!");  
};
```

执行函数

在创建CFC函数时，需要指定处理程序以让CFC能够找到入口函数。对于Node.js函数，处理程序为您函数中的"module-name.export"值，例如，填写"index.handler"则表示调用的是index.js中的export.handler函数。

函数代码

[测试](#)

编辑类型：

[在线编辑](#)[上传函数ZIP包](#)[从BOS上传ZIP](#)

当前代码：

```
1 exports.handler = (event, context, callback) => {  
2   callback(null, "hello world")  
3 };  
4
```

[保存](#)[重置](#)此函数可能包含外部库。 [点击下载完整 ZIP 程序包](#)

处理程序：

index.handler

[修改](#) [?](#)

环境变量：

[- 编辑](#)

入参

Node.js函数入参包括event, context和callback,其中callback是可选参数

- event参数：您调用本函数时传入的事件数据，其数据类型是一个object
- context参数：CFC向函数提供的运行时信息，其数据类型是一个object
- callback参数（可选）：该参数用于返回信息。

返回信息

Node.js采用异步编程模型，您的函数需要使用callback入参返回信息。callback的语法如下：

```
callback(Error error, Object result);
```

其中：

- error（可选参数）：在函数执行内部失败时使用此参数返回错误信息，成功的情况下设置为null。
- result（可选参数）：使用此参数返回函数的执行结果。result可以是任意类型。
- 您的函数在callback被调用后立即返回，重复调用 callback 只接受第一次调用的结果。需要确保所有任务在 callback 调用之前完成，否则有些任务可能不会被运行。例如调用下面的函数，将返回 hello world。

```
exports.handler = (event, context, callback) => {  
  callback(null, 'hello world');  
  callback(null, 'done');  
};
```


该函数的返回结果如下：

✓ 执行结果：成功

此区域下方显示你的函数执行的结果。

```
hello world
```

此区域下方显示你的函数执行过程中产生的日志。

```
START RequestId: 145e4e92-edac-4e2e-a41c-817388cc93c4 Version: $LATEST
END RequestId: 145e4e92-edac-4e2e-a41c-817388cc93c4
REPORT RequestId: 145e4e92-edac-4e2e-a41c-817388cc93c4   Duration: 849ms   Billed Duration: 900ms   Max Memory Used: 7.7M
```

- 如果你的函数里未调用callback，则函数执行将会超时。

```
exports.handler = (event, context, callback) => {
  console.log('hello world');
};
```

该函数示例代码执行结果如下：

✗ 执行结果：失败

此区域下方显示你的函数执行的结果。

```
{"errorMessage":"aabcff8b-730d-4c8a-8b3b-8706bb228b62 Task timed out after 3 seconds"}
```

此区域下方显示你的函数执行过程中产生的日志。

```
START RequestId: aabcff8b-730d-4c8a-8b3b-8706bb228b62 Version: $LATEST
2020-07-24T11:15:49.010Z aabcff8b-730d-4c8a-8b3b-8706bb228b62 hello world
END RequestId: aabcff8b-730d-4c8a-8b3b-8706bb228b62
REPORT RequestId: aabcff8b-730d-4c8a-8b3b-8706bb228b62   Duration: 3.001s   Billed Duration: 3.1s   Max Memory Used: 9.8M
```

注意：CFC函数根据您选择的触发器的不同，函数调用方式也不同，对函数的返回结果的处理方式不同。

- 同步调用(RequestResponse): 使用同步调用时，结果会以序列化的JSON格式返回，调用方可以根据返回值进行后续处理。
- 异步调用(Event): 异步调用时，调用方法触发函数后立即返回，函数返回值会被丢弃。

异常处理

如上对callback入参的说明，您可以使用callback的第一个参数返回错误，如：

```
exports.handler = (event, context, callback) => {
  callback(new Error('myError'))
};
```

该函数的返回结果如下：

文件格式要求为 zip 格式。用于压缩或解压的工具，在 Windows 平台下可使用例如 7-Zip 工具，在 Linux 平台下可使用 zip 命令行工具，在 macOS 平台下也可以使用 zip 命令行工具。

🔗 打包方式

打包时，需要对文件打包，而不是对文件外的目录打包；打包完成后，入口函数文件需要位于包内的根目录。

以 Node.js 为例，打包完成的 zip 文件的目录结构应该为

Example.zip	示例工程包
--- lib	业务文件目录
--- node_modules	npm三方件目录
--- index.js	入口js文件（必选）
--- package.json	npm项目管理文件

Windows 平台下，您可以使用图形界面打开函数代码目录，挨个选择需要打包的文件以后，单击鼠标右键，选择“压缩为 zip 包”。用 WinRAR、7-Zip 等工具打开 zip 包时，应该直接能显示入口文件。

Linux 和 macOS 平台下，打开终端程序，例如系统自带的终端或者 iTerm2，cd 进入函数代码目录，调用 zip 命令打包所需文件，以 Node.js 为例，命令应该是：

```
zip -r code.zip index.js node_modules/
```

🔗 各语言创建部署包说明

🔗 Python

- **简单**：如果您的代码只需要 BCE 的 SDK，则无需安装，CFC 内置了 Python 的 BCE SDK。
- **高级**：如果您的代码需要用到其他的第三方库，则需要先在本地创建 zip 压缩包，并使用 CFC 控制台或者命令行工具上传部署代码。

下面是在本地创建 zip 包的过程

1. 创建一个目录，例如 cfc-python-demo
2. 将所有的 Python 源文件 (.py) 保存在此目录的根级
3. 使用 pip 安装所有的库，注意需要指定安装路径。如果您是 Windows 或 Linux，则可以运行命令

```
$ pip install module-name -t /path/to/cfc-python-demo
```

例如，安装requests库

```
$ pip install requests -t /path/to/cfc-python-demo
```

如果使用的是 Mac OS X，并且是通过 Homebrew 安装的 Python，那么前述命令不适用。一个简单的变通方法是使用以下内容将在您的 /path/to/cfc-python-demo 目录中添加 setup.cfg 文件，再执行安装命令。

```
[install]
prefix=
```

4. 压缩 cfc-python-demo 目录的内容，需注意是压缩目录的内容，而不是目录本身。例如此时的目录结构是这样，其中 index.py 定义了您的入口函数

```

...
$ cd cfc-python-demo
$ tree -L 1
.
├── bin
├── certifi
├── certifi-2018.10.15.dist-info
├── chardet
├── chardet-3.0.4.dist-info
├── idna
├── idna-2.7.dist-info
├── index.py
├── requests
├── requests-2.20.0.dist-info
├── setup.cfg
├── urllib3
└── urllib3-1.24.dist-info
...

```

在 `/path/to/cfc-python-demo` 目录下运行压缩命令，`demo.zip` 就是可以上传的 zip 压缩包。

```

...
$ zip -r demo.zip .
...

```

5. 通过控制台上传zip包。注意处理程序需要修改成您的入口文件和处理函数，例如您的入口函数是 `index.py` 里的 `handler` 函数，那么在“处理程序”这里，需要改成 `index.handler`。

函数代码

发布新版本 测试

编辑类型: 在线编辑 上传函数.ZIP包

函数程序包: + 选择文件 开始上传

处理程序: index.handler 修改

环境变量: 编辑

使用BSAM-CLI调试函数

BCE-Serverless-Applcation-Model规范

BCE SAM是用来定义百度函数计算CFC无服务器应用架构的模型，为本地开发工具**bce-sam-cli**提供指引。BCE SAM定义了一组对象，这些对象可以包含在模板定义文件中，以便轻松地描述CFC函数。

当前BCE SAM的版本号为2018-08-30。

🔗 [BCE SAM模板](#)

```
Description:
  bsam-app

Resources:
  HelloWorldFunction:
    Type: BCE::Serverless::Function
    Properties:
      CodeUri: hello_world/
      Handler: app.cfc_handler
      Runtime: python2.7
      Environment:
        Variables:
          PARAM1: VALUE
    Events:
      HelloBos:
        Type: BOS
        Properties:
          Bucket: mybucket
          Prefix: images/
          Suffix: .jpg
          EventTypes: [PutObject, PostObject]
      HelloDuerOS:
        Type: DuerOS
```

说明：

所有的属性字段都是大小写敏感的。

🔗 规范详解

描述符合BCE SAM的CFC应用程序的文件格式为YAML，它包含了Resource、Property和Event Source三个部分，它们共同定义了一个CFC应用。

Resource

BCE::Serverless::Function

表示一个CFC函数，其包含的属性如下：

属性名	类型	描述
Handler	string	必需. 用户代码中的启动函数
Runtime	string	必需. 运行时环境
CodeUri	string	必需. 代码路径
Description	string	函数的描述
MemorySize	integer	每次调用函数分配的内存大小（MB），默认为128
Timeout	integer	函数运行超时时间，默认为3s
Environment	Environment Object	设置函数执行时的环境变量
Events	map of string to Event Source Object	设置函数触发器的键值对

Property

Environment Object

描述函数环境变量属性的对象，属性如下：

属性名	类型	描述
Variables	map of string to string	定义环境变量键值对，其中变量名称是键，变量值是值。变量名称仅限于字母数字字符，第一个字符必须是字母。值仅限于字母数字字符和以下特殊字符_ () { } \$ * + - \ / " # ' , ; . @ ! ? 。

范例:

```
Variables:
TABLE_NAME: my-table
STAGE: prod
```

Event Source Object

描述函数触发器的对象，属性如下：

属性名	类型	描述
Type	string	必需. 触发器类型，目前支持的种类有"BOS"，"HTTP"，"DuerOS"
Properties	map	触发器的详细配置键值对，根据Type类型而定，详细参考Event Source Types

BOS触发器范例:

```
Type: BOS
Properties:
Bucket: myBucket
EventTypes: [PutObject, PostObject]
Prefix: images/
Suffix: .jpg
```

DuerOS触发器范例：

```
Type: DuerOS
```

HTTP触发器范例：

```
Type: HTTP
Properties:
ResourcePath: /hello-world
Method: [GET, POST]
AuthType: anonymous
```

Event Source

BOS

配置BOS触发器，当Bucket上发生您指定的事件时，BOS会向CFC调用您的函数。

属性名	类型	描述
Bucket	string	必需. BOS bucket 名字
EventTypes	string List of string	必需. 触发事件类型
Prefix	string	前缀
Suffix	string	后缀

DuerOS

创建DuerOS触发器，允许DuerOS平台调用您的CFC函数。DuerOS无其它属性。

HTTP

配置HTTP触发器，将函数关联到一个URL上，通过HTTP方法执行函数。

属性名	类型	描述
ResourcePath	string	必需. 访问路径，例如/hello-world
Method	string List of string	必需. 触发器匹配的HTTP方法
AuthType	string	必需. 身份验证方式，可选值为"iam"，"anonymous"

BSAM-CLI使用示例

BSAM CLI是一个基于[BCE SAM规范](#)的命令行工具，它提供了本地开发环境，帮助您在把函数上传到百度智能云CFC之前，在本地进行函数的开发、分析和执行。

🔗 工具安装

1. 安装Python3.6及以上

2. 安装Docker

BSAM CLI依赖Docker，所以您需要提前安装并运行Docker服务，关于各个操作系统Docker的安装可以参考[Docker官网](#)。

为了查看Docker服务是否已运行，您可在安装完成后执行`docker ps`等命令进行验证。

3. 安装 BSAM CLI

执行 `pip3 install bce-sam-cli` 即可完成安装。

安装完成后，执行`bsam`命令，出现下面的提示说明安装成功。

```
Usage: bsam [OPTIONS] COMMAND [ARGS]...
```

```
BCE Serverless Application Model (SAM) CLI
```

```
The BCE Serverless Application Model provide a simplified way of defining
the CFC functions, triggers and other services needed by your serverless
application.You can find more in-depth guide about the SAM specification
here: https://github.com/bcelabs/serverless-application-model.
```

```
... ..
```

🔗 功能概述

- **init** 初始化一个serverless应用，应用文件夹包含源码以及template文件
- **config** 配置连接远端CFC的相关参数

- **package** 打包本地源码
- **deploy** 将本地serverless应用部署到CFC
- **local** 本地运行以及测试serverless应用

1.初始化 主要参数：

- **-l / --location** 生成项目时template的所在路径，可以为git、http、zip以及文件夹
- **-r / --runtime** 选择可支持的运行时环境
- **-o / --output-dir** 项目生成的文件夹路径
- **-n / --name** 项目名称（也是之后云函数的名称）

通过init命令初始化一个工作目录，指定runtime为nodejs10。

```
$ bsam init --runtime nodejs10 --name bsam-app
```

该命令会在当前目录下生成一个bsam-app文件夹，目录结构如下：

```
$ cd bsam-app
$ tree
.
├── README.md
├── src
│   ├── app.js
│   ├── package.json
│   └── tests
│       ├── unit
│       └── test_handler.js
└── template.yaml
```

app.js是函数代码文件。

template.yaml是BCE SAM模型文件，定义了函数的函数名、runtime、handler等元信息。BSAM CLI从模型文件中获取函数配置，从而对正确地执行函数。

2.创建配置文件 主要参数：

- **-l / --location** 配置信息所在路径，默认为~/bce
- **--ak** 以及 **--sk**
- **--region**

函数部署功能需要您在home目录下预先创建好bce config和credentials文件，将access key、secret key和region信息写入其中。

执行**bsam config**命令可帮助您快速创建这两个文件。

3.函数打包与部署

在本地开发和测试完函数后，进入函数项目根目录，可以使用**bsam package**和**bsam deploy**命令部署到CFC。

package命令将压缩您的代码文件，生成一个.zip文件。

deploy命令会将.zip文件上传到CFC，若该函数不存在则创建一个新的函数，若您在模型文件中定义了触发器，那么触发器也会被一并创建。若该函数已存在，则会更新该函数的代码和配置。

deploy支持指定部署到华北-北京、华东-苏州或华南-广州机房。

4.生成事件

为了使您更方便的开发测试CFC函数，您可以创建并自定义触发源事件。

```
bsam local generate-event <service> <event> [options]
```

目前支持的service有 bos、 dueros、 http。每种服务的事件类型不同，以dueros为例，它支持四种event：

event	含义
start-session	Generates a DuerOS Start Session Event
end-session	Generates a DuerOS End Session Event
intent-answer	Generates a DuerOS Intent Answer Event
event-report	Generates a DuerOS Report Event

5.安装依赖/编译

在执行函数之前，需要做一些额外工作来扫清障碍。若您的函数依赖第三方库，BSAM CLI可帮您自动安装这些依赖。

```
bsam local install
```

该命令会在当前目录查找和读取BCE SAM模型文件template.yaml，若模型文件不在当前目录，您可使用-t参数指定文件的路径。

您可使用CodeUri字段指定函数所在目录，BSAM CLI会读取该目录中的package.json（Node.js函数）或requirements.txt（Python函数），根据其中的配置下载所需的模块。您也可以手动在该目录中放置依赖文件。

Java

Java是编译型语言，在执行前需要编译函数(与其类似的有golang)。执行install命令时，BSAM CLI使用maven下载依赖、编译函数，后者会读取本地的pom.xml文件。若您的Java函数依赖本地jar包，可将jar包放在bsam-app目录下，在pom.xml文件中指定jar包的位置，示例如下：

```
<dependency>
<groupId>org.ibm</groupId>
<artifactId>jms</artifactId>
<version>1.0.0</version>
<scope>system</scope>
<systemPath>${project.basedir}/lib/user-local.jar</systemPath>
</dependency>
```

本地jar包只能放在bsam-app目录范围内，放到其它位置不会被挂载到容器中，无法被maven找到。

install命令执行完毕后，bsam-app下会新建一个target目录，其中的 xx-xx-jar-with-dependencies.jar为编译好的包含依赖的jar包。

python

install命令执行完毕后，源码路径src下会新建一个site-packages目录，所有依赖都放置在该文件夹下，其项目路径结构如下：

```

.
├── README.md
├── src
│   ├── __init__.py
│   ├── site-packages    <-- 存放依赖文件
│   │   ├── lib1
│   │   ├── lib2
│   │   └── ...
│   ├── index.py         <-- python 函数源码
│   └── requirements.txt  <-- Python 依赖管理文件
├── template.yaml        <-- BSAM 模型文件
├── tests                 <-- 单元测试
│   └── unit
│       ├── __init__.py
│       └── test_handler.py

```

注意：

- 当执行函数代码打包命令 `bsam package` 时，`bsam` 会将 `site-packages` 下的依赖抽取出来和用户编写的业务代码一起打包
- 这种依赖与源码分离的项目结构，只能利用 `bsam` 命令进行调试与本地运行，如果用户需要利用本地编译器进行运行，可以为其配置环境变量 `$PYTHONPATH`，参考代码如下

```
export PYTHONPATH = .../src/sitepackages:$PYTHONPATH
```

6. 执行函数

在执行函数时，需要传递函数的 Logical ID 和 event json 文件，`bsam local invoke` 也接受 `stdin` 作为 event 输入。

每次执行函数前，BSAM CLI 会自动检查是否有可更新的镜像，您可通过 `--skip-pull-image` 参数取消检查。

示例：

```

# invoke时传入event json文件
$ bsam local invoke bsam-app -e event.json

# 使用stdin传入event
$ echo '{"msg":"hello, cfc"}' | bsam local invoke bsam-app

# 查看更多选项
$ bsam local invoke --help

```

该命令会在当前目录查找和读取 BCE SAM 模型文件 `template.yaml`，若模型文件不在当前目录，您可使用 `-t` 参数指定文件的路径。

若您执行 Java 函数，请用 `CodeUri` 字段指定上一步中编译好的 jar 包文件路径。

7. 函数调试

`bsam local invoke` 命令支持调试功能，您可借助本地 IDE 完成调试，下面以 VS Code 为例介绍如何调试。

调试 Node.js 函数

1. 用 VS Code 打开工作目录 `bsam-app`。
2. 执行命令 `bsam local invoke -e event.json --debug-port 5858 bsam-app`，这里指定容器暴露的调试端口为 5858。
3. 在 VS Code 的菜单栏中，选择 "Debug"，"Open Configurations"，在打开的 `launch.json` 中编辑内容如下：

```
{
  "version": "0.2.0",
  "configurations": [
    {
      "name": "BSAM CLI Node.js Debug",
      "type": "node",
      "request": "attach",
      "address": "localhost",
      "port": 5858,
      "localRoot": "${workspaceRoot}/bsam-app/src",
      "remoteRoot": "/var/task",
      "protocol": "inspector",
      "stopOnEntry": false,
    }
  ]
}
```

其中, "port"字段值要和第二步中指定的端口保持一致, "localRoot"是代码文件所在的目录。 "protocol"表示Node.js使用的调试协议, 请指定为"inspector"。 "localRoot" 表示函数源码所在的本地路径, 其中\${workspaceRoot}是VSCode的一个环境变量, 表示您当前的工作空间路径, <FUNCTION_PATH> 是函数在当前工作空间的相对路径。 关于每个字段的详细解释和VS Code Debugger的更多用法, 您可参考[Node.js debugging in VS Code](#)

4. 在您的代码中打上断点, 然后选择"Debug", "Start Debugging", 开始调试。

调试Python函数

1. 用VS Code打开工作目录bsam-app。
2. 执行命令 `bsam local invoke -e event.json --debug-port 5890 bsam-app`, 这里指定容器暴露的调试端口为5890。
3. 在VS Code的菜单栏中, 选择"Debug", "Open Configurations", 在打开的launch.json中编辑内容如下:

```
{
  "version": "0.2.0",
  "configurations": [
    {
      "name": "BSAM CLI Python Debug",
      "type": "python",
      "request": "attach",
      "port": 5890,
      "host": "localhost",
      "pathMappings": [
        {
          "localRoot": "${workspaceFolder}/bsam-app/src",
          "remoteRoot": "/var/task"
        }
      ]
    }
  ]
}
```

其中, "port"字段值要和第二步中指定的端口保持一致, "localRoot"是代码文件所在的目录。 关于每个字段的详细解释和VS Code Debugger的更多用法, 您可参考[Python debugging in VS Code](#)

4. 在代码中添加断点, 在VS Code菜单栏中选择"Debug", "Start Debugging", 程序会自动停在代码中的断点, 开始调试。

调试Java函数

1. 用VS Code打开工作目录bsam-app。

1. 执行命令 `bsam local invoke -e event.json --debug-port 8890 bsam-app`，这里指定容器暴露的调试端口为8890。
2. 在VS Code的菜单栏中，选择"Debug"，"Open Configurations"，在打开的launch.json中编辑内容如下：

```
{
  "version": "0.2.0",
  "configurations": [
    {
      "type": "java",
      "name": "Debug Java (Attach)",
      "request": "attach",
      "hostName": "localhost",
      "port": 8890
    }
  ]
}
```

其中，"port"字段值要和第二步中指定的端口保持一致。

关于每个字段的详细解释和VS Code Debugger的更多用法，您可参考[Debugging Java in VS Code](#)

4. 在您的代码中打上断点，然后选择"Debug"，"Start Debugging"，开始调试。

Baidu Serverless VSCode 插件

Baidu Serverless VSCode Extension 是百度云 Serverless 产品 CFC 的 VSCode 插件，该插件结合了 CFC 本地命令行工具 BSAM CLI，可以帮您快速创建、管理本地的 CFC 函数，同时借助 Docker 轻松实现函数在本地的执行和 Debug。

🔗 功能概览

- 本地函数快速创建
- 本地函数的编译与依赖安装
- 本地函数执行
- Nodejs、Python、Java 函数本地 Debug
- 本地函数部署到云端
- 自定义测试事件的创建
- 多地域云端函数列表、下载
- 云端函数调用

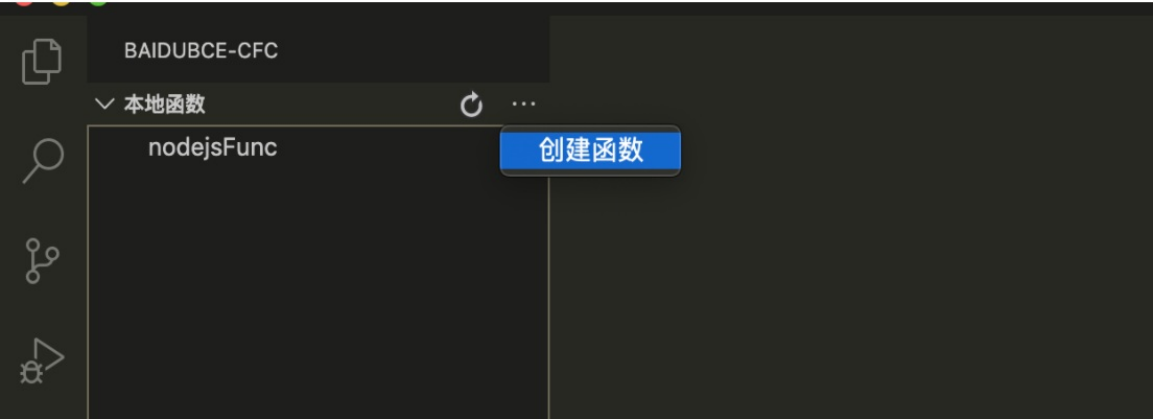
🔗 安装与设置

1. 创建您的[百度云账号](#)，并在个人信息的[安全认证](#)中创建一个 Access Key。
2. 执行 `pip3 install -i https://pypi.tuna.tsinghua.edu.cn/simple bce-sam-cli` 以安装 BSAM CLI。插件在启动时也会检测 BSAM CLI 是否已安装，您可根据插件的提示来操作安装。关于 BSAM CLI 更多信息，您可参考 [BSAM CLI使用示例](#)。
3. 安装 Docker。您可访问 [Docker 官网](#)，根据您的操作系统下载不同的安装包进行安装。
4. 安装 VSCode。您可访问 [VSCode 官网](#) 下载对应的安装包。
5. 在插件市场，搜索 "Baidu Serverless" 并安装。

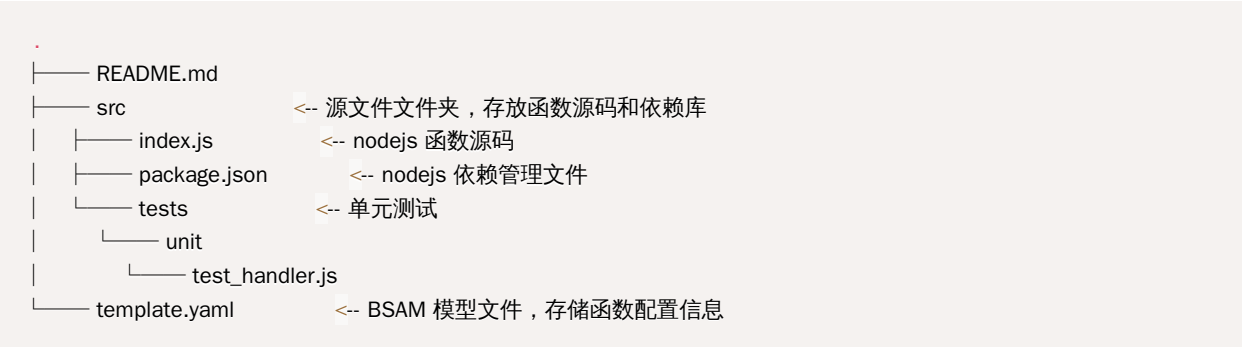
🔗 本地函数功能

🔗 函数创建

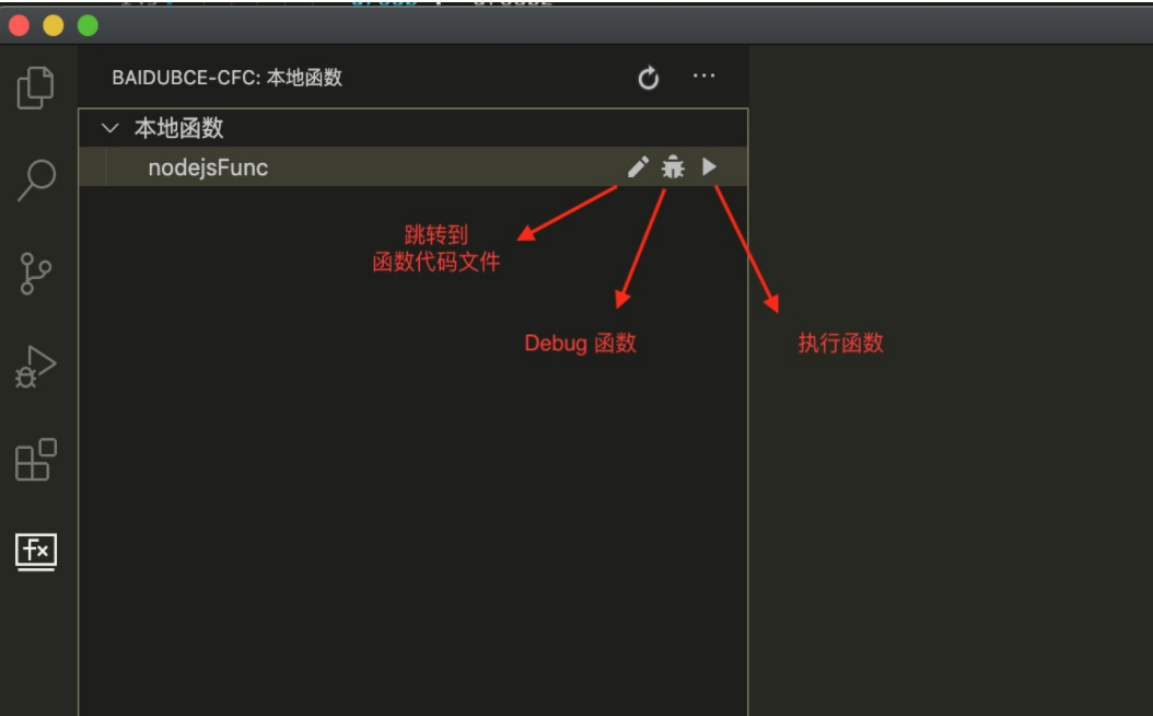
进入插件面板，在『本地函数』栏点击『…』，并选择『创建函数』，根据弹窗的提示步骤即可完成本地函数的创建，函数默认存储路径为当前打开的工作空间的根目录。新函数均使用模板代码，您可在函数创建后修改函数的代码与配置。



以 NodeJS 函数为例，创建后的目录结构如下：



函数创建完成后，您可在『本地函数』栏中看到新创建的函数，每行函数均有快捷功能图标，如下图所示。



依赖安装与编译

函数执行前可能需要下载依赖包和执行编译，CFC 函数的执行环境是 Linux，您在本地下载的依赖和编译结果可能与 CFC 执行环境不兼容。为了避免这一情况，本插件集成了依赖安装与编译功能，您无需手动为函数来安装依赖与编译。在『本地函数』栏对某个函数点击右键，选择『安装依赖与编译』，插件即会根据函数的运行时类型来做相关的操作。

自定义测试事件

插件集成了 CFC 云端所有触发器事件，若您想用自定义测试事件来执行函数，您可以在『本地函数』栏对某个函数点击右键，选择『创建测试事件』，根据弹窗提示在本地创建一个测试文件，创建完成后您可自行修改该文件，在执行函数时选择该事件。

函数执行与 Debug

为了保证函数能够执行与 Debug，请您确定您已安装 Docker 并启动。执行函数有两种方式，一种是通过点击『本地函数』栏函数所在行的执行图标，一种是在函数代码编辑框中，点击 handler 函数上方的 "Local Invoke"。

Debug 函数同样可使用这两种方式启动，您需要首先在函数文件中添加断点，才可启动 Debug。插件目前支持 NodeJS、Python 和 Java 函数的 Debug 功能。

```
Local Invoke | Local Debug
def handler(event, context):
    assert event.get('isBase64Encoded')
    body = base64.standard_b64decode(event.get('body'))

    ocr_result = paddle_ocr(body)
```

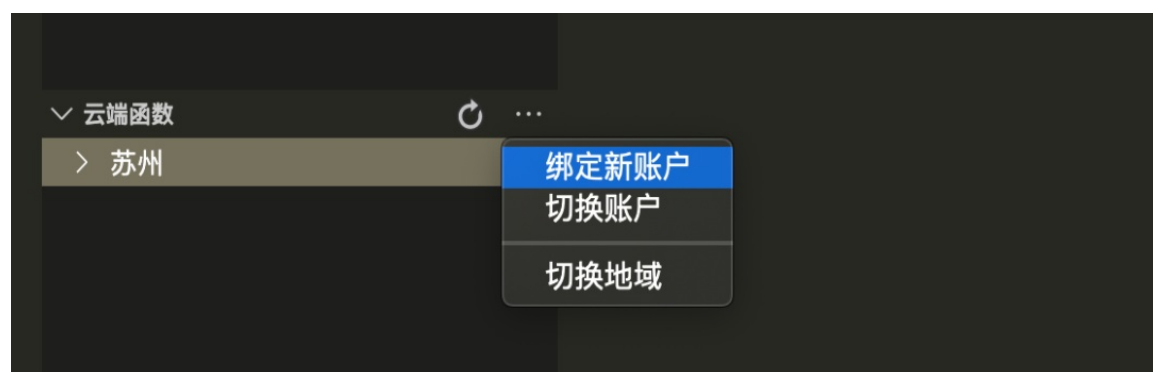
函数部署

若您已登录云端账号，在『本地函数』栏对某个函数点击右键，选择『上传函数』，即可将函数部署到 CFC 云端。若您尚未登录，请您先参考下面的云端账号功能进行登录。

云端函数功能

云端账号

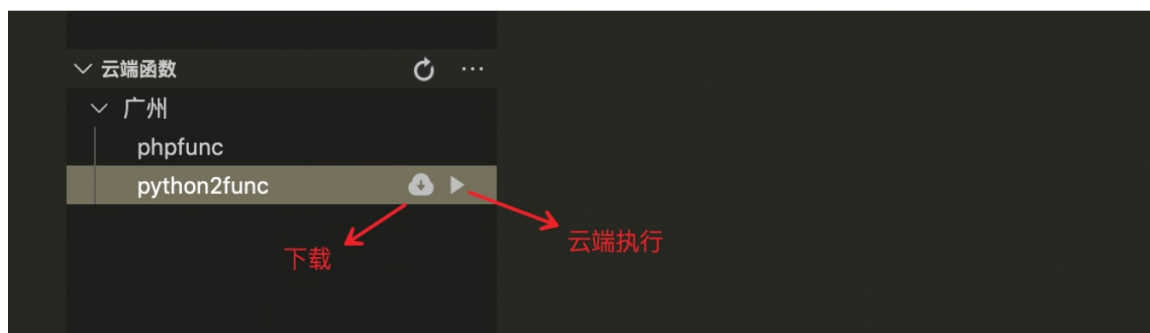
如下图所示，在『云端函数』栏点击『…』，并选择『绑定新账户』，根据提示输入您账号的 AK、SK 及自定义别名。



您可以绑定多个账户，通过『切换账户』来使用不同的账户，通过『切换地域』来显示当前账号下对应区域的云端函数列表。

云端函数执行与下载

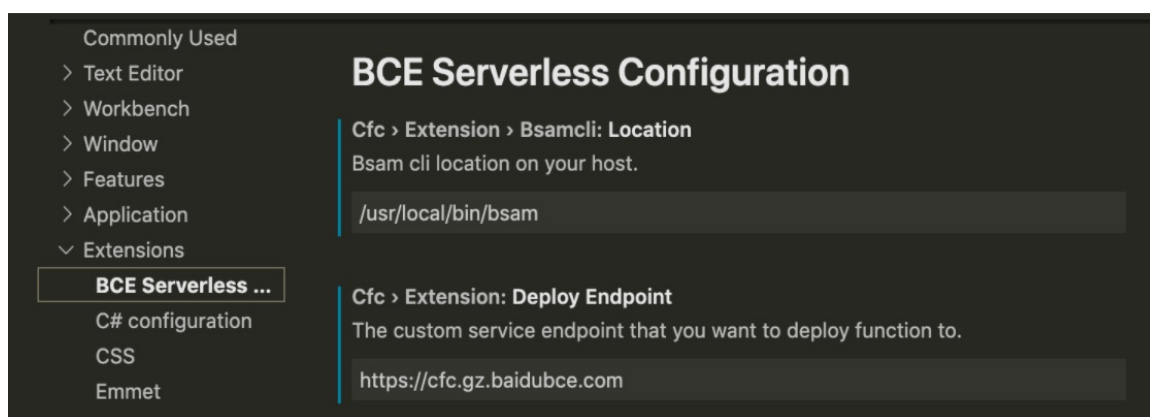
在绑定好百度云账户后，『云端函数』栏会显示您的云端函数列表，如下图所示。



您可点击执行图标来调用云端函数，点击下载图标来把云端函数下载到本地目录。对非编译型语言，比如 NodeJS、Python 等，下载后插件会自动为函数生成 Template.yaml 文件。但是对编译型语言 Java、Golang 等，由于下载结果可能不包含函数源码文件，因此不为其生成 .yaml 文件。

🔗 用户配置项

您可在 VSCode 的 Settings->Extensions->BCE Serverless Configuration 中查看插件的配置项，如图所示。



- **Bsam Cli Location**：默认情况下，插件会在您的 \$PATH 路径下查找 BSAM CLI 的可执行文件，一般情况下您通过 pip3 安装后无需手动配置该项。若您安装在非 \$PATH 路径下，您可修改该配置。
- **插件日志**：在您对插件的操作过程中，相关的日志会输出到 VSCode OUTPUT 的 "BaiduBCE CFC" 中。默认日志输出 Level 为 INFO，您可在 Log Level 项中修改 Level。
- **Deploy Endpoint**：在您配置了云端函数的地域后，该配置会自动切换为对应地域的域名。若您期望把函数部署到私有服务地址，可修改该配置。
- **Max Detect Depth**：插件会在当前工作空间下的路径下查找函数，该项用于配置查找函数的路径深度。
- **Skip Pull Image**：默认情况下，插件每次执行、Debug 函数时会从百度镜像仓库检查镜像是否有更新，若有更新则更新后再继续执行。检查操作会耗费一点时间，若您网络状况不好或期望能快速执行函数，可以勾选此项。

开发注意事项

时间戳与时区

CFC函数内的时间戳为Unix时间戳(Unix timestamp)，时间为UTC世界标准时间。如果您需要在函数内打印格式化时间，需要指定时区，例如 "Asia/Shanghai"。以下提供Node.js和Python两种语言的示例代码，仅供参考：

Node.js相关示例:

```
exports.handler = (event, context, callback) => {
  let d = new Date();
  console.log(d.toUTCString())
  console.log(d.toLocaleString("en-US", {timeZone: "Asia/Shanghai"}))
  console.log(d.toISOString())
  callback(null, "Hello world!");
};
```

Python相关示例：

```
##### -*- coding: utf-8 -*-
from datetime import datetime, timedelta, timezone

def handler(event, context):
    utc_dt = datetime.utcnow().replace(tzinfo=timezone.utc)
    print("utc datetime : ", utc_dt)
    bj_dt = utc_dt.astimezone(timezone(timedelta(hours=8)))
    print("bj datetime : ", bj_dt)
    return "Hello World"
```

函数运行环境及安全隔离性

函数运行时

CFC目前支持的运行环境有Node.js 10, Node.js 12, Node.js 14, Node.js 16, Python 2.7, Python 3.6, Python 3.10, Java 8, Golang, PHP 7.2, Lua 4.3, .Net Core 2.2, PowerShell 6.2。选择以上的运行环境运行CFC函数，各运行环境使用说明文档详见开发语言，如 [Python](#)。

运行环境中已经内置了常用的依赖，可以直接在函数中引用。

函数执行路径

用户的函数会运行在 CFC 定制的容器环境内。函数代码包会被加载到 `/var/task` 路径上。如果需要读取代码包中包含的其他文件，可以使用绝对路径，或是参考以下代码片段，获取当前文件路径后，根据相对位置拼接完整路径。

Node.js：

```
const path = require("path");

exports.handler = (event, context, callback) => {
  const datafile = path.join(__dirname, "data.txt")
  callback(null, datafile);
};
```

Python:

```
##### -*- coding: utf-8 -*-
import os

def handler(event, context):
    currentdir = os.path.dirname(os.path.abspath(__file__))
    datafile = os.path.join(currentdir, "data.txt")
    return datafile
```

函数执行安全

函数执行环境是独立安全的，确保安全的函数计算最重要的目标之一。用户的函数运行环境中的隔离保护和百度云BCC是同一级别的，函数计算是在虚拟机级别隔离运行。CFC还在网络、数据、代码、防DDoS攻击等方面做了完备的保护，确保用户的信

息安全。

函数代码安全

函数计算对代码进行校验并将其存储在CFC私有BOS仓库中。使用代码时，函数计算都执行完整性检查。代码执行与它自己的文件系统和网络命名空间相隔离。

无状态函数

无状态服务

无状态服务即不对用户的数据做持久化的服务。函数计算服务就是典型的无状态服务，一旦用户长时间未进行函数调用，导致函数实例回收，那么之前代码产生的一些中间状态（临时变量、生成的本地文件等等）都会一起丢失。

无状态函数

- 无状态函数即对一个函数而言，以同样的参数作为输入，无论什么时候调用、调用多少次，输出结果始终保持一致。
- 由于CFC属于无状态服务，因此开发者在编写代码的时候，需要考虑将函数代码编写成无状态风格，来保证调用结果的幂等性。

持久化方式

由于函数计算本身是无状态服务，因此可以通过结合其他云服务来对所需的数据进行持久化处理，目前常见的处理方式有：

- 将数据写入对象存储服务，详见[BOS触发器使用](#)
- 将数据写入百度云数据库，详见[配置CFC函数以访问百度云中的RDS](#)

冷热启动机制

冷启动

由于函数在需要响应事件的容器中运行，因此存在一定的延时(启动函数容器实例和语言runtime启动的耗时)。这被称为“冷启动”。当您的函数执行完成后，您的函数容器实例可能会保留一段时间。如果另一个事件在此时被触发，则它的响应速度要快得多，这通常被称为“热启动”。

热启动和冷启动的耗时差异在于容器和语言运行时runtime的启动耗时等初始化的过程。

冷启动的时间不是一个固定的数，它可能取决于使用的运行时（或编程语言）、函数代码包的大小，以及是否需要进入用户自己的VPC等因素。它通常包括实例容器的启动、运行时环境的准备、代码下载和初始化等环节。

优化方法

冷启动的优化涉及开发侧、函数计算平台双方共同的努力。函数计算CFC平台已经对系统侧的冷启动做了大量优化。对于用户侧的冷启动，建议您通常从以下几方面优化：

- 精简代码包：建议您将不必要的依赖剔除，减少代码体积，进而提升下载速度。
- 资源选取：可以根据自身业务情况，选择合适的CPU、内存资源。
- 预热：使用定时触发器预热函数，例如：每隔一定时间(比如1分钟)运行一次函数。
- 其他：选择合适的函数语言可以在一定程度上加快函数的运行，例如：对于简单的函数实现，优先推荐python、nodejs 等；此外，优化函数启动方式等也能在一定程度上提升启动速度。

触发器

触发器概述

函数计算CFC(Cloud Function Compute)提供基于事件触发、全托管的云端计算能力。函数计算通过事件触发器连接到其他服务，当对应服务发出请求时，函数即会响应运行，作出相应处理和反馈。例如：通过BOS触发器将函数连接到百度智能云对象存储服务，设置函数响应BOS的新建对象事件。当用户的对象存储中增加新的文件时，函数即被触发执行，对新上传的文件进行诸如内容检查等操作。

名词解释

- 触发器(Trigger): 触发器是对CFC来说的，表示一种触发关系，包括触发源、目标函数Bm、触发条件等。对触发源所在的服务方来说，触发器是一种事件通知，当触发条件发生时，则通知CFC调用某个函数。

触发器的信息由服务方持久存储，服务方检测到触发条件发生时调用CFC函数。一个函数可以创建多个触发器。

- 触发源: 触发源的粒度是由业务确定，比如Bos触发器的触发源精确到一个bucket；而CDN触发器的粒度则比较大，触发源就是服务本身。
- Policy: Policy是CFC根据触发源为函数设置的允许触发规则，CFC持久存储Policy。没有Policy的话，触发源对函数的调用请求会失败，一条Policy的作用范围是单个函数Bm + 触发源。

🔗 触发器分类

目前CFC支持10种触发器触发函数，各触发器的调用方式如下表所示:

名称	调用方式
DuerOS触发器	同步
Duedge触发器	同步
BOS触发器	异步
HTTP触发器	同步
CDN触发器	同步
百度消息服务(Kafka)触发器	异步
定时触发器	异步
BLS触发器	异步
API Gateway触发器	同步
百度消息服务(专享版Kafka)触发器	异步

🔗 创建触发器

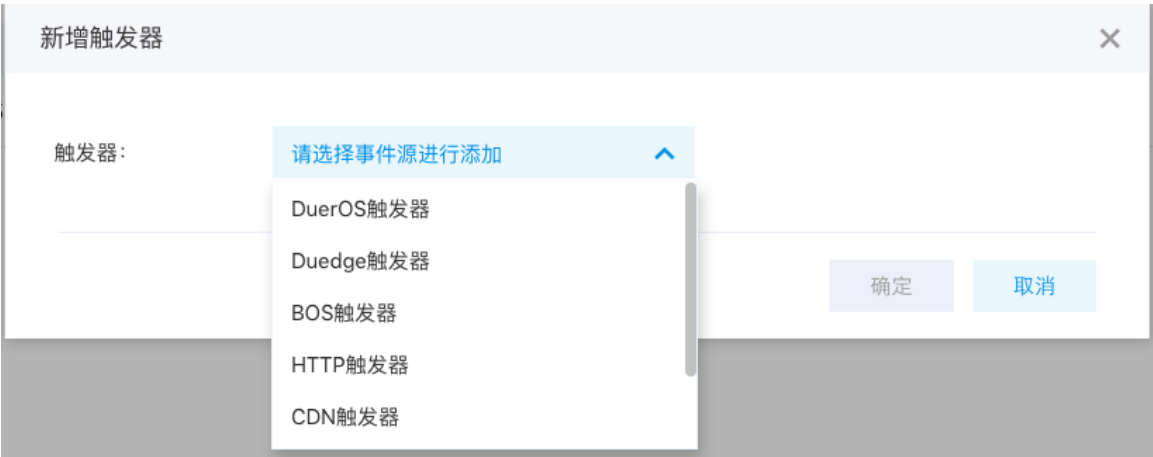
CFC支持两种方式为函数创建触发器：控制台创建和通过API接口创建。

🔗 控制台创建

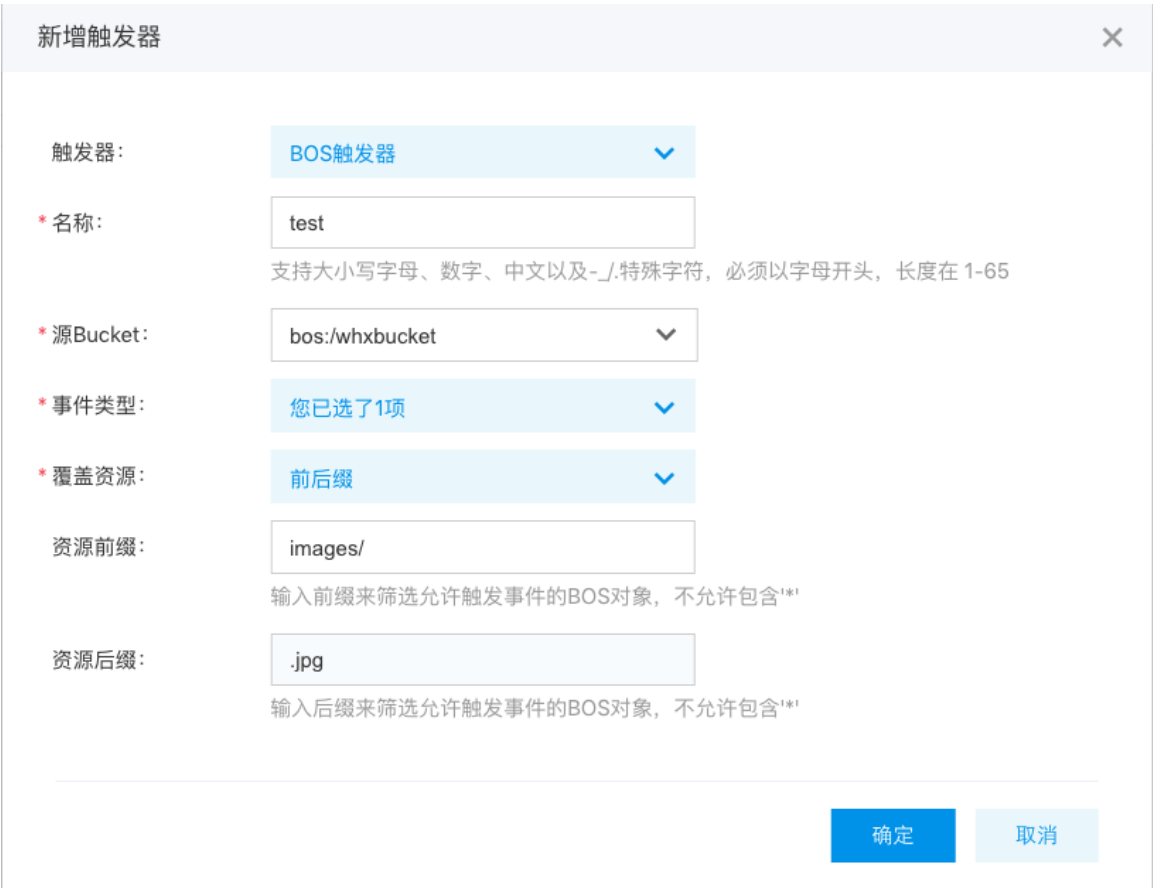
- 1、登录管理控制台，选择“产品服务> 函数计算CFC”，进入“函数列表”页面。
- 2、点击需要添加触发器的函数名称即(test-20191211), 进入函数详情页面。
- 3、点击左侧导航栏中的“触发器”，进入触发器配置页面，如下图所示：



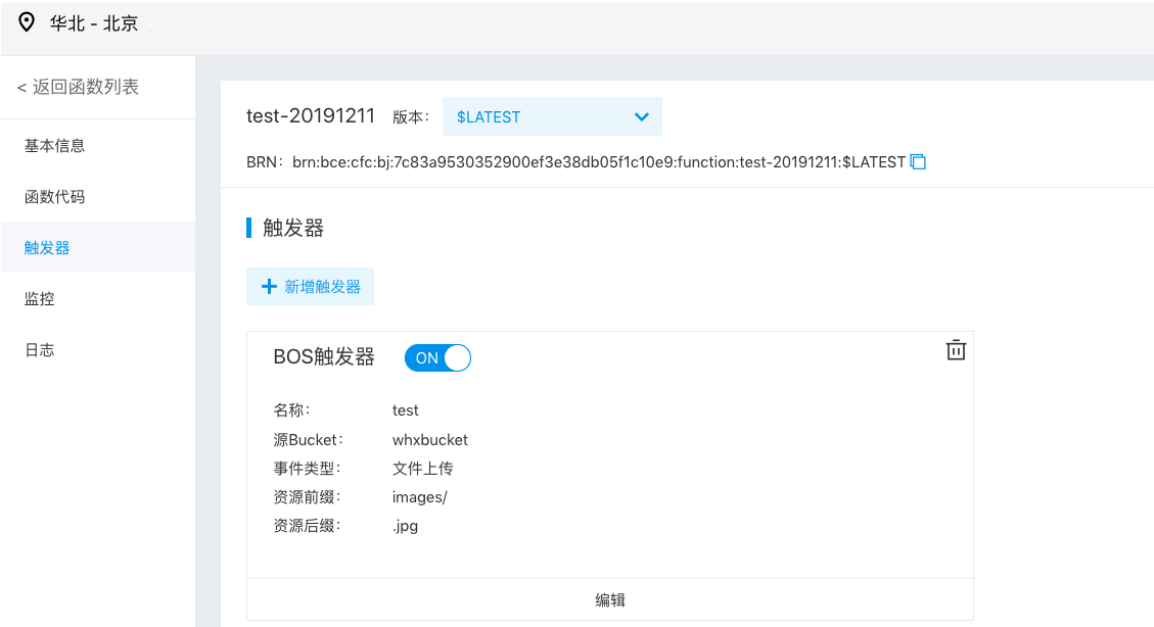
4、在弹出框中，点击下拉框“请选择事件源进行添加”，选择需要添加的触发器



5、之后在弹出框中配置触发器选项，并点击确认，完成触发器的创建。以BOS触发器为例：



6、在函数配置页面“触发器”一栏中，可以看到刚刚创建好的BOS触发器及其信息。



API接口创建

用户可以使用函数计算Open API创建触发器，具体操作方式，请参考[创建触发器](#)。

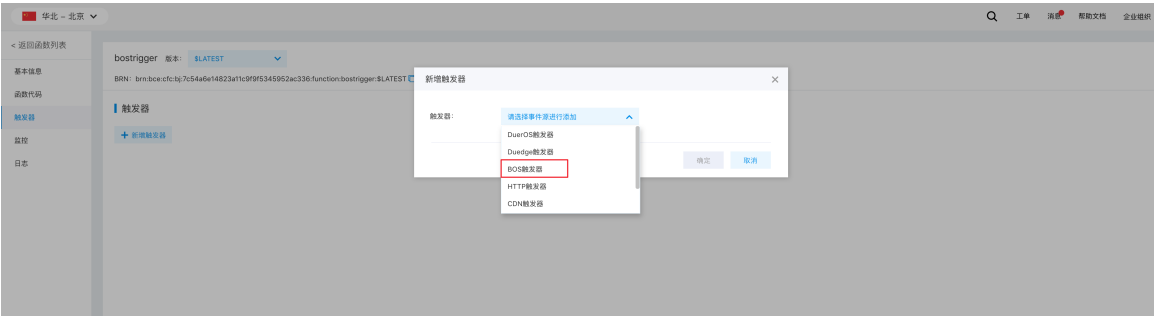
BOS触发器

用户可以为新建的函数或已有函数配置 BOS 触发器，创建函数的流程可以具体参考 [创建函数](#) 。

这里假设用户已完成一个名为 bostrigger 的 Python 函数的创建，以下内容以此为前提，指引用户在 CFC 控制台在函数管理页面中为函数配置触发器。接下来，我们将通过三步来完成一个触发器的设置。

配置 BOS 触发器

- 1. 登录管理控制台，选择“产品服务> 函数计算 CFC”，进入“函数列表”页面。
- 2. 点击需要添加 BOS 触发器的函数名称（即 bostrigger），进入函数详情页面。
- 3. 点击左侧导航栏中的“触发器”，进入函数配置页面，点击“新增触发器”。在弹出框中选择 BOS 触发器。

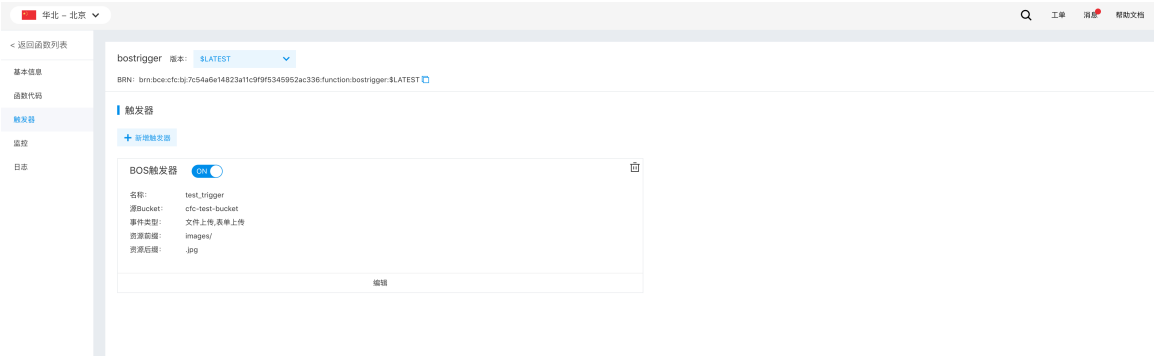


- 4. 之后在弹出框中填入触发器的各个参数、点击确认，触发器即创建完成，各个参数的解释如下：

- 名称：即触发器的名称。
- 源Bucket：每个触发器需要选择一个 BOS Bucket 作为触发源，且 bucket 与函数需位于同一个地域。
- 事件类型：选择一个或多个事件类型，当覆盖资源在源 Bucket 上发生这些事件时，您的函数就会被触发。
- 覆盖资源：表示您想监控的 BOS 资源的文件名，您可以用文件全名来特指某一个文件，或者用指定前后缀的方式，来指定一类文件名。

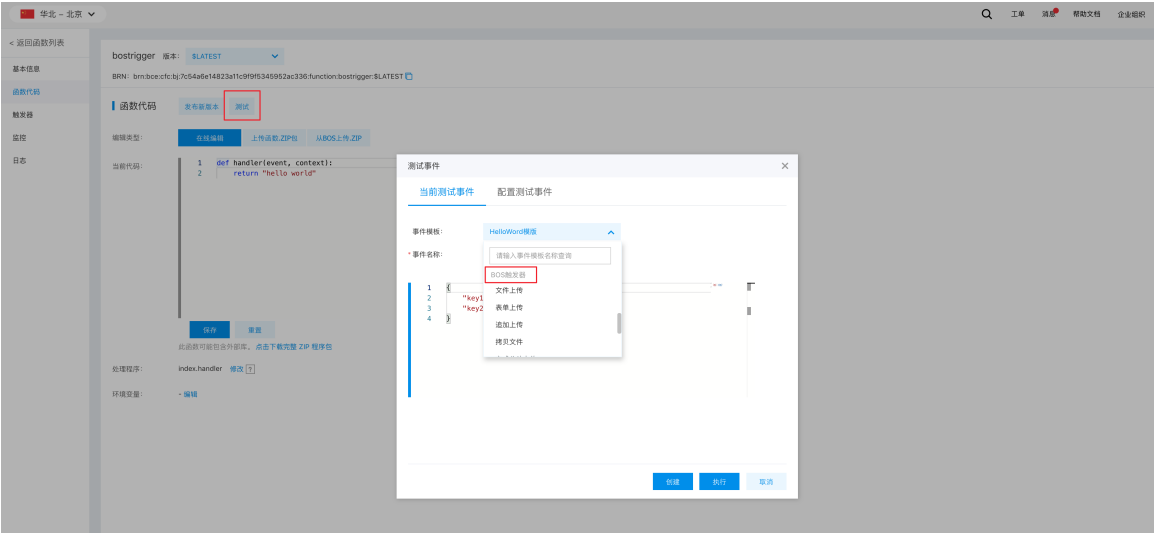


5. 在函数配置页面“触发器”一栏中，可以看到刚刚创建好的 BOS 触发器及其信息。



编写函数代码

使用 BOS 触发器时，一般情况下您需要读取函数的 event，根据 event 中的信息做相应的处理。在“函数代码”页点击“测试”，在“当前测试事件”中选择 BOS 触发器的某个事件。可以看到每种事件类型都有它对应的 event，以及每个 event 的内容示例。



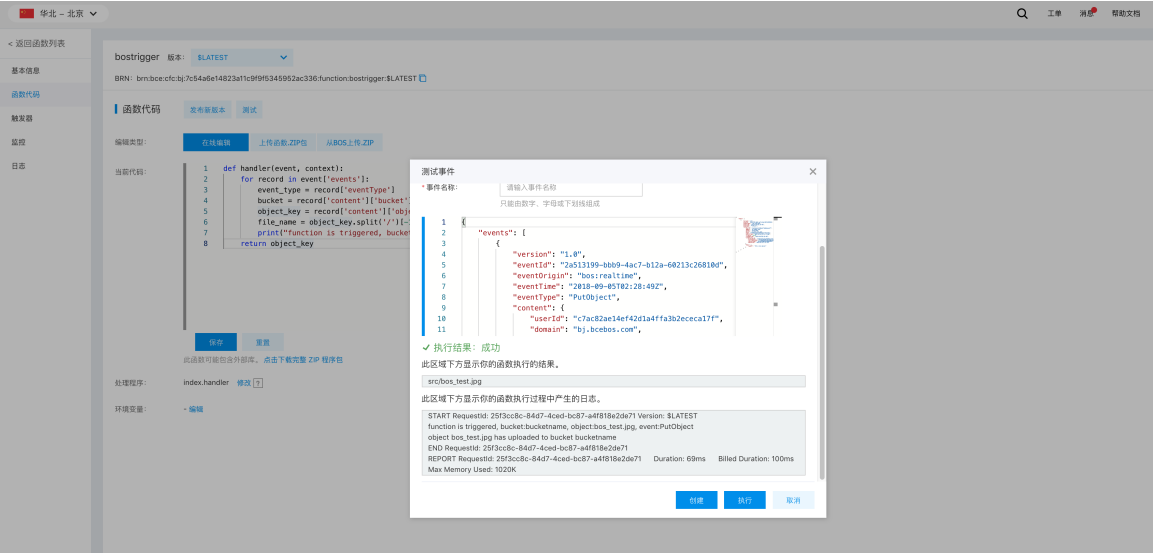
本例选择“文件上传”，根据该 event 的事件格式编写如下代码：

```
def handler(event, context):
    for record in event['events']:
        event_type = record['eventType']
        bucket = record['content']['bucket']
        object_key = record['content']['object']
        file_name = object_key.split('/')[1]
        print("function is triggered, bucket:%s, object:%s, event:%s" % (bucket, file_name, event_type))
    return object_key
```

测试触发器

在“函数代码”页点击“测试”，在“当前测试事件”中选择“文件上传”，点击“执行”。

函数执行结果如图



BOS 事件通知

CFC 的 BOS 触发器与 BOS 的事件通知是互通的，您在 CFC 创建、修改触发器后，在 BOS 对应 bucket 的事件通知页可以看到相应的事件通知，反之亦然。所以，若您需要在 CFC 和 BOS 之间建立触发关系，您在 CFC 或 BOS 的其中一个进行配置即可，不需要重复配置。

关于更多 BOS 事件通知的更多说明，您可以参考 [BOS 事件通知](#)。

更多应用

为了便于您编写代码，CFC 为您提供了 BOS 触发器蓝图，每个蓝图都有自己的功能。创建函数时您可以选择“从模板创建函数”，“关键词”输入 BOS 即可获取相关蓝图，在此蓝图基础上修改代码实现您的逻辑。

CDN触发器

用户可以为新建的函数或已有函数配置CDN触发器，创建函数的流程可以具体参考 [创建函数](#) 这里不再赘述。

这里假设用户已完成一个名为hello-cdn的Python函数的创建，以下内容以此为前提，指引用户在CFC控制台在函数管理页面中为函数配置触发器。接下来，我们将通过三步来完成一个触发器的设置。

编写处理函数

使用CDN触发器需要用户在函数代码中作出相应的调整，按照CDN的事件格式来编写handler，具体可参考 [CDN事件说明](#)。

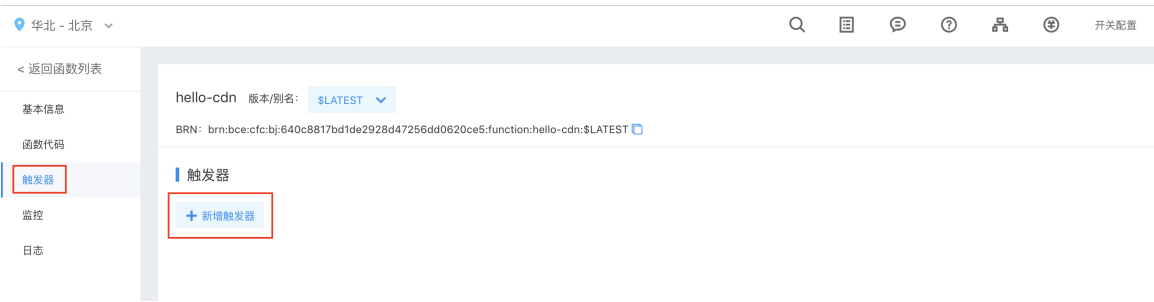
1. 登录管理控制台，选择“产品服务>云函数计算 CFC”，进入“函数列表”页面。
2. 在“函数列表”页面，点击名称为hello-cdn的函数，进入函数详情页面。
3. 在函数详情页中编写函数代码，以对触发操作做出适当的处理，之后点击右下角保存按钮完成函数的修改操作。

```
# -*- coding: utf-8 -*-
def handler(event, context):
    for record in event['events']:
        domain = record['eventDetail']['domain']
        status = record['eventDetail']['status']
        print("current status of domain '%s' is '%s' " % (domain, status))
        return 'Successfully processed {} records.'.format(len(event['events']))
```

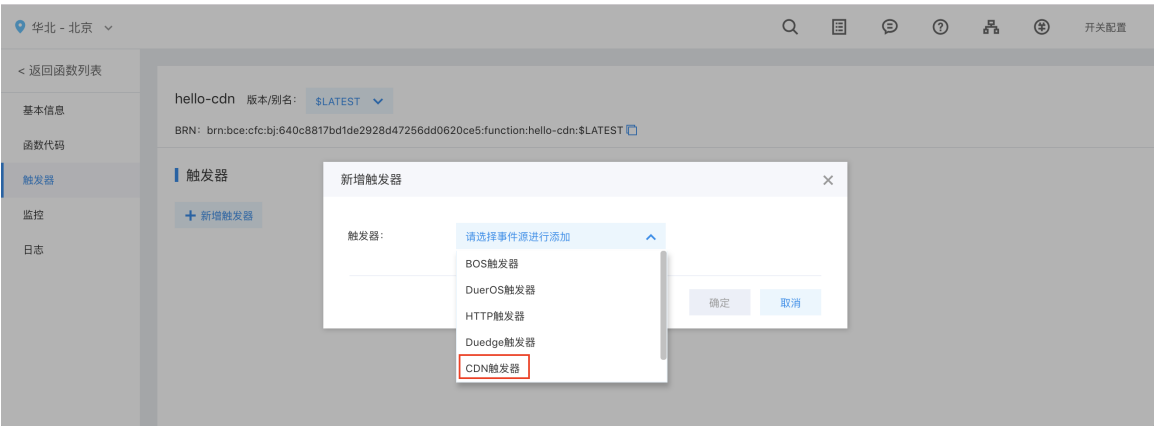
配置CDN触发器

1. 登录管理控制台，选择“产品服务> 函数计算 CFC”，进入“函数列表”页面。
2. 点击需要添加CDN触发器的函数名称（即hello-cdn），进入函数详情页面。

3. 点击左侧导航栏中的“触发器”，进入函数配置页面，点击“新增触发器”。

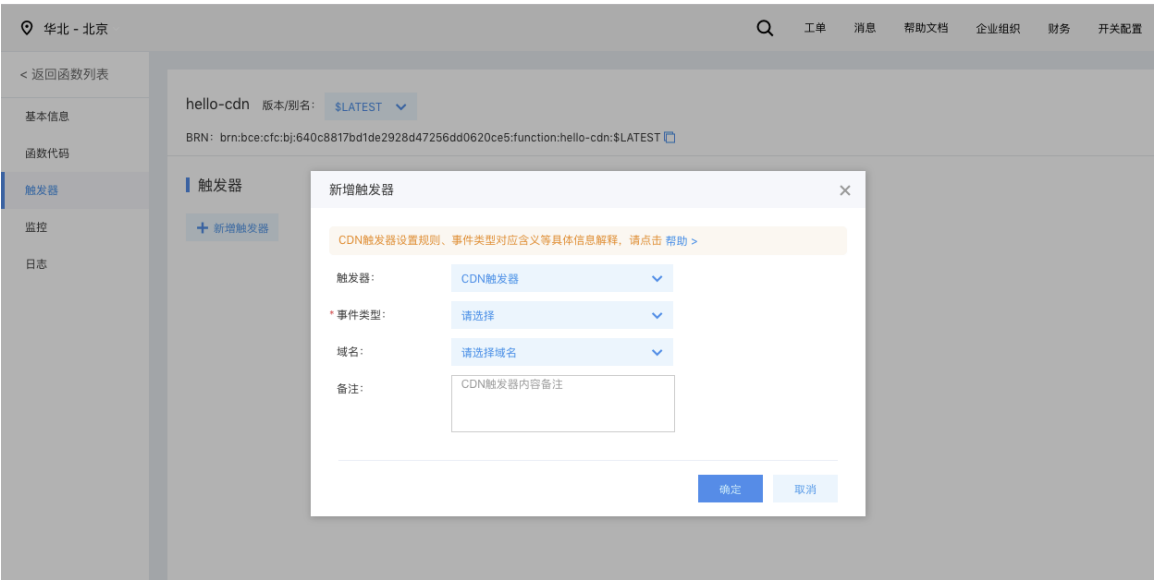


4. 在弹出框中，点击下拉框“选择事件源进行添加”，选择CDN触发器。

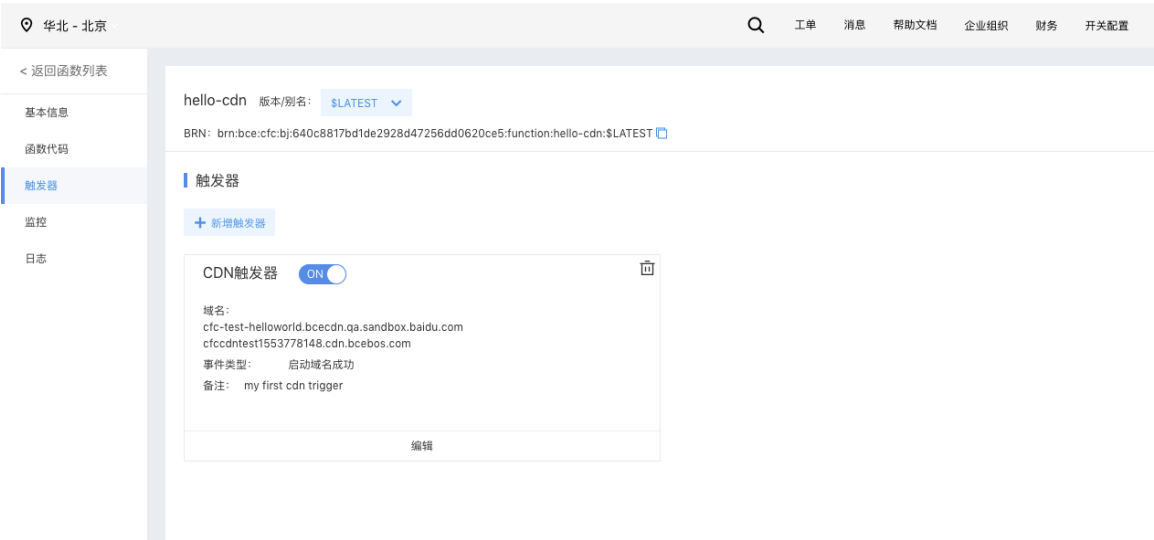


5. 之后在弹出框中配置好域名、事件类型和触发器备注，并点击确认，完成触发器的创建。

- 事件类型：触发器需要匹配的事件类型，可以通过下拉框选择一种类型，这里我们选择“启动域名成功”
- 域名：CDN服务中管理的域名
- 备注：触发器的备注信息

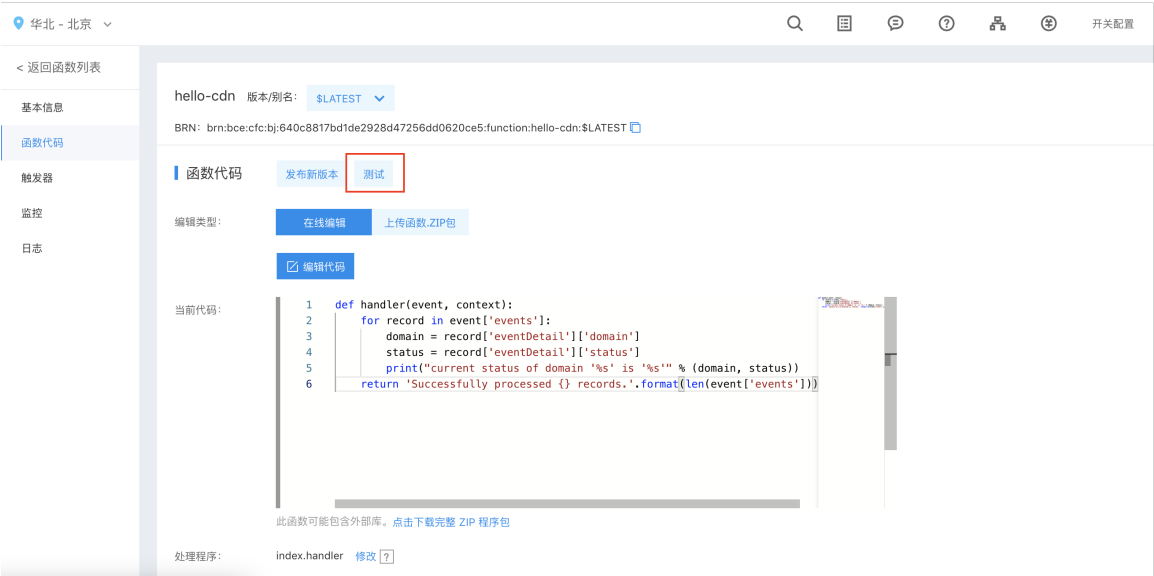


6. 在函数配置页面“触发器”一栏中，可以看到刚刚创建好的CDN触发器及其信息。



测试触发器

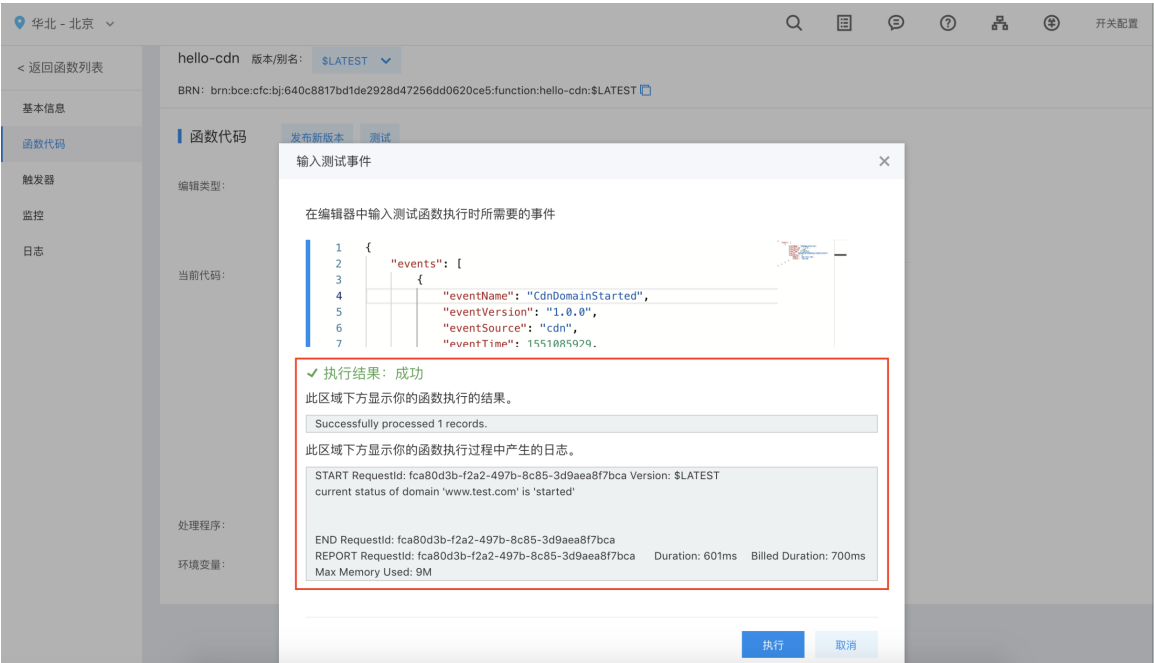
1. 点击测试按钮



2. 输入测试事件，并点击执行，测试内容如下

```
{
  "events": [
    {
      "eventName": "CdnDomainStarted",
      "eventVersion": "1.0.0",
      "eventSource": "cdn",
      "eventTime": 1551085929,
      "userId": "9b2aa05367204d9481ef29bdf5ef8243",
      "eventDetail": {
        "domain": "www.test.com",
        "status": "started"
      }
    }
  ]
}
```

3. 查看返回的结果



CDN事件说明

事件名称	事件说明	事件粒度	事件版本
CachedObjectsBlocked	发现违禁资源	用户粒度	1.0.0
CachedObjectsPushed	预热资源完成	用户粒度	1.0.0
CachedObjectsRefreshed	刷新资源完成	用户粒度	1.0.0
CdnDomainCreated	创建域名成功	用户粒度	1.0.0
CdnDomainDeleted	删除域名成功	用户粒度	1.0.0
LogFileCreated	产生域名日志	域名粒度、用户粒度	1.0.0
CdnDomainStarted	启动域名成功	域名粒度、用户粒度	1.0.0
CdnDomainStopped	停止域名成功	域名粒度、用户粒度	1.0.0

粒度说明

事件的粒度分为用户粒度和域名粒度，其中域名粒度更为精细，用户粒度指代用户全局粒度；针对每个CFC函数触发器的设置，不指定具体域名即为用户粒度触发器，指定具体域名即为域名粒度触发器。

- 用户粒度：用户账户产生指定事件，即可触发相关CFC函数
- 域名粒度：特定域名产生指定事件，即可触发相关CFC函数

事件结构

预热刷新事件 CachedObjectsPushed：用户账户下预热任务完成 CachedObjectsRefreshed：用户账户下刷新任务完成

```
{
  "events": [
    {
      "eventName": "CachedObjectsPushed", // 事件类型
      "eventVersion": "1.0.0", // 事件版本
      "eventSource": "cdn", // 事件源
      "eventTime": 1551085930, // 时间戳
      "userId": "9b2aa05367204d9481ef29bdf5ef8243", // 用户账户ID
      "eventDetail": {
        "taskId": "eJwzMTU3NjQ2MXBIM0tMTTJKNtMkIIMkwwsEo0AAbMLeg==", // 任务ID
        "status": "completed" // 刷新或预热任务完成
      }
    }
  ]
}
```

黄反事件

CachedObjectsBlocked：用户账户下发现黄反URL

```
{
  "events": [
    {
      "eventName": "CachedObjectsBlocked", // 事件类型
      "eventVersion": "1.0.0", // 事件版本
      "eventSource": "cdn", // 事件源
      "eventTime": 1551085930, // 时间戳
      "userId": "9b2aa05367204d9481ef29bdf5ef8243", // 用户账户ID
      "eventDetail": {
        "url": "http://www.test.com/sexy.jpg", // 黄反URL
        "status": "slightPorn" // 取值为slightPorn或severePorn
      }
    }
  ]
}
```

日志事件 LogFileCreated：用户账户下某一域名或特定域名产生最近时间访问日志

```
{
  "events": [
    {
      "eventName": "CachedObjectsBlocked", // 事件类型
      "eventVersion": "1.0.0", // 事件版本
      "eventSource": "cdn", // 事件源
      "eventTime": 1551085930, // 时间戳
      "userId": "9b2aa05367204d9481ef29bdf5ef8243", // 用户账户ID
      "eventDetail": {
        "domain": "www.test.com", // 产生日志的域名
        "logTime": "2019-02-26T12:00:00" // 日志产生时间
      }
    }
  ]
}
```

域名事件 CdnDomainStarted：用户账户下某一域名或特定域名启动成功 CdnDomainStopped：用户账户下某一域名或特定域名停止成功 CdnDomainCreated：用户账户下某一域名创建成功 CdnDomainDeleted：用户账户下某一域名删除成功

```
{
  "events": [
    {
      "eventName": "CdnDomainStarted", // 事件类型
      "eventVersion": "1.0.0", // 事件版本
      "eventSource": "cdn", // 事件源
      "eventTime": 1551085930, // 时间戳
      "userId": "9b2aa05367204d9481ef29bdf5ef8243", // 用户账户ID
      "eventDetail": {
        "domain": "www.test.com", // 触发动作的域名
        "status": "started" // 取值为started、stopped、created、deleted
      }
    }
  ]
}
```

百度消息服务触发器

🔗 百度消息服务触发器说明

百度消息服务是全兼容 Apache Kafka 的分布式、高可扩展、高通量的消息托管服务。

如果在百度消息服务的 Topic 上检测到记录，您可以使用 CFC 函数从此 Topic 读取成批的数据来处理，并且 CFC 会定期轮询（每秒一次）Topic 中的新记录。

百度消息服务目前分为共享版和专享版两类，两者的区别您可参考百度消息服务相关文档 [消息服务 for Kafka](#)，目前推荐使用百度消息服务专享版创建触发器。

如果您使用百度消息服务专享版创建触发器，需要在创建消息服务集群时开启网络配置中的产品间转存储功能，触发器才能正常工作，共享版无相关限制

网络配置

* 网络实例: [创建VPC](#)

公网访问: ☐ 关

产品间转储: ☒ 开

当部分产品转储KAFKA时开启，提供网络打通方案，如流日志、消息中心等，详情见产品协同

🔗 百度消息服务触发器创建

用户可以为新建的函数或已有函数配置百度消息服务触发器，创建函数的流程可以具体参考 [从头创建函数](#)，这里不再赘述。

这里假设您已经使用Python2.7创建了一个名为 kafkatrigger 的函数，以下内容以此为前提，将引导您在 CFC 控制台在函数管理页面中为函数配置百度消息服务触发器。接下来，我们将通过以下步骤来完成一个触发器的设置。

🔗 编写处理函数

1. 登录管理控制台，选择“产品服务>云函数计算 CFC”，进入“函数列表”页面
2. 在“函数列表”页面。点击名称为 kafkatrigger 的函数，进入函数详情页面。

在函数详情页中编写百度消息服务触发器对应的 handler，以对触发操作返回适当的响应，之后点击右下角保存按钮完成函数的修改操作。

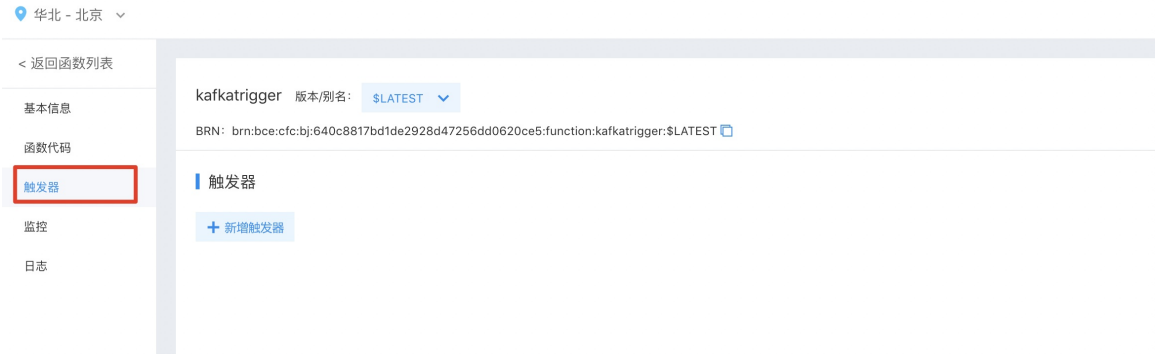
```
# -*- coding: utf-8 -*-
import base64
import json
def handler(event, context):
    for record in event['Records']:
        # kafka value is base64 encoded so decode here
        payload = base64.b64decode(record['Kafka']['Value'])
        print("Decoded payload: " + payload)
    return 'Successfully processed {} records.'.format(len(event['Records']))
```

注：百度消息服务触发器的event消息体

```
{
  "Records": [
    {
      "EventId": "c7ac82ae14ef42d1a4ffa3b2eacea17f__test_only_sl-0-43110",
      "Kafka": {
        "Key": "", //base64后的key
        "Value": "MA==", //base64后的value
        "Topic": "c7ac82ae14ef42d1a4ffa3b2eacea17f__test_only_sl",
        "Partition": 0,
        "Offset": 43110,
        "Timestamp": 1553151704.6529999
      },
      "EventName": "bce:kafka:record",
      "EventSourceBrn": "c7ac82ae14ef42d1a4ffa3b2eacea17f__test_only_sl",
      "EventSource": "bce:kafka"
    },
    {
      "EventId": "c7ac82ae14ef42d1a4ffa3b2eacea17f__test_only_sl-0-43111",
      "Kafka": {
        "Key": "",
        "Value": "MQ==",
        "Topic": "c7ac82ae14ef42d1a4ffa3b2eacea17f__test_only_sl",
        "Partition": 0,
        "Offset": 43111,
        "Timestamp": 1553151704.6529999
      },
      "EventName": "bce:kafka:record",
      "EventSourceBrn": "c7ac82ae14ef42d1a4ffa3b2eacea17f__test_only_sl",
      "EventSource": "bce:kafka"
    }
  ]
}
```

配置百度消息服务触发器（共享版）

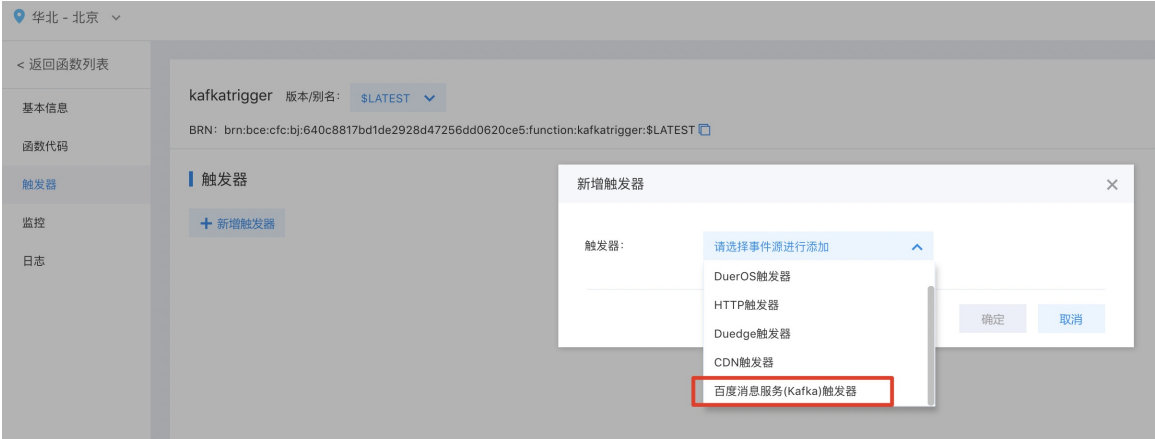
1. 登录管理控制台，选择“产品服务> 函数计算 CFC”，进入“函数列表”页面
2. 点击需要添加百度消息服务触发器的函数名称（即 kafkatrigger），进入函数详情页面。
3. 点击左侧导航栏中的“触发器”，进入函数配置页面。



4. 在函数配置页面中最下方点击“新增触发器”。

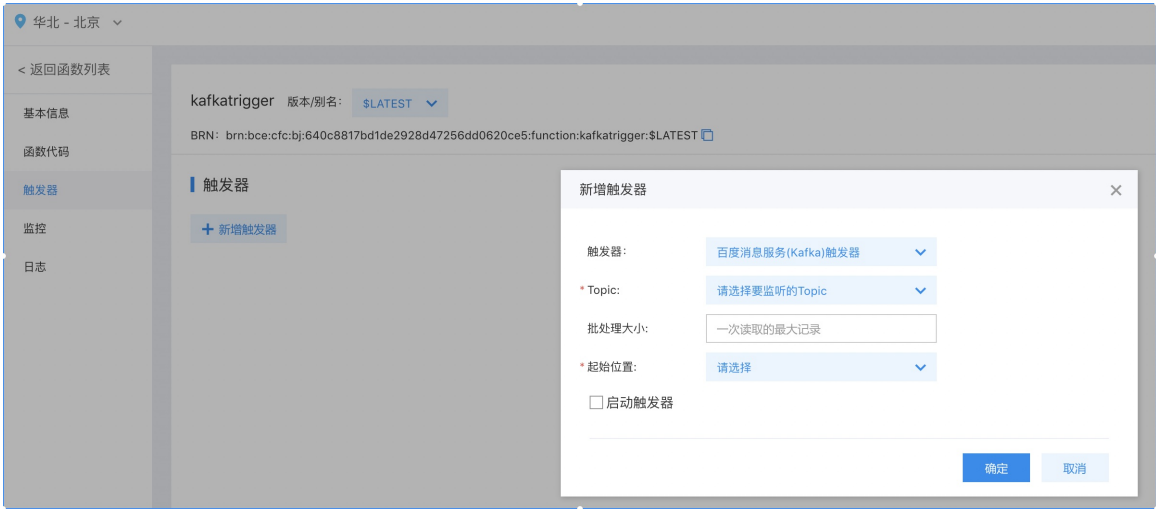


5. 在弹出框中，点击下拉框“选择事件源进行添加”，选择百度消息服务触发器。

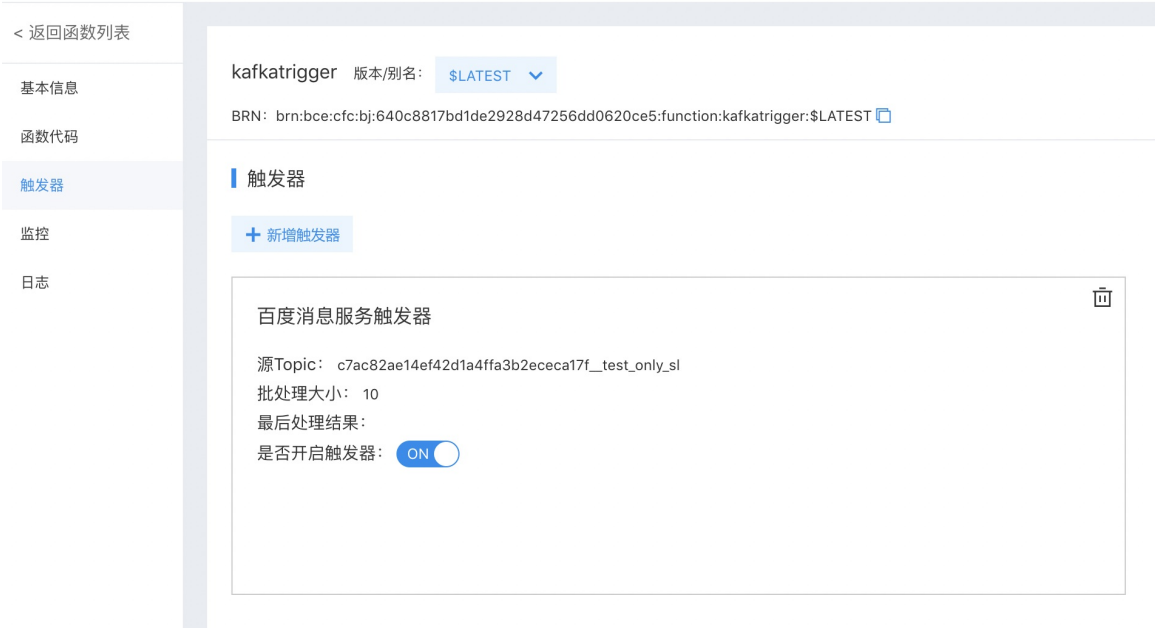


6. 之后在弹出框中配置好选项，并点击确认，完成触发器的创建。

- Topic 选择您将要监听的百度消息服务 Topic
- 批处理大小：从 Topic 中一次读取的最大记录数，1-1000
- 起始位置：在 Topic 中开始读取的位置，最新记录对应 kafka 的 OffsetNewest，最老记录对应 kafka 的 OffsetOldest
- 启用触发器:是否直接启动触发器，建议先关闭触发器以便测试



7. 在函数配置页面“触发器”一栏中，可以看到刚刚创建好的百度消息服务触发器及其信息



配置百度消息服务触发器（专享版）

- 1. 登录管理控制台，选择“产品服务> 函数计算 CFC”，进入“函数列表”页面
- 2. 点击需要添加百度消息服务触发器的函数名称（即 kafkatrigger），进入函数详情页面。
- 3. 点击左侧导航栏中的“触发器”，进入函数配置页面。



4. 在函数配置页面中最下方点击“新增触发器”。



5. 在弹出框中，点击下拉框“选择事件源进行添加”，选择百度消息服务触发器（专享版）。



6. 之后在弹出框中配置好选项，并点击确认，完成触发器的创建。

- Kafka集群: 选择您要使用的百度消息服务专享版集群名
- Topic: 选择您将要监听的百度消息服务 Topic
- 批处理大小：从 Topic 中一次读取的最大记录数，1-1000
- 起始位置：在 Topic 中开始读取的位置，最新记录对应 kafka 的 OffsetNewest，最老记录对应 kafka 的 OffsetOldest
- 启用触发器:是否直接启动触发器，建议先关闭触发器以便测试



7. 在函数配置页面“触发器”一栏中，可以看到刚刚创建好的百度消息服务触发器（专享版）及其信息

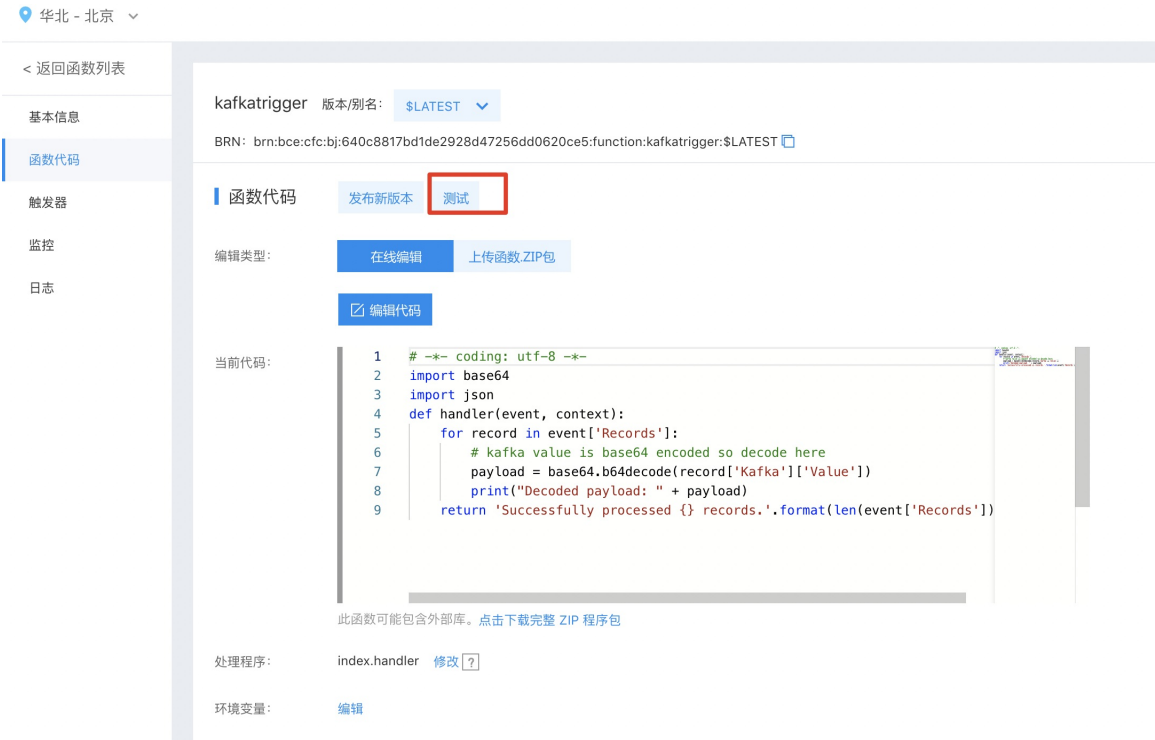


测试触发器

专享版与共享版触发器的使用在CFC侧相同，测试触发器流程一致

模拟测试

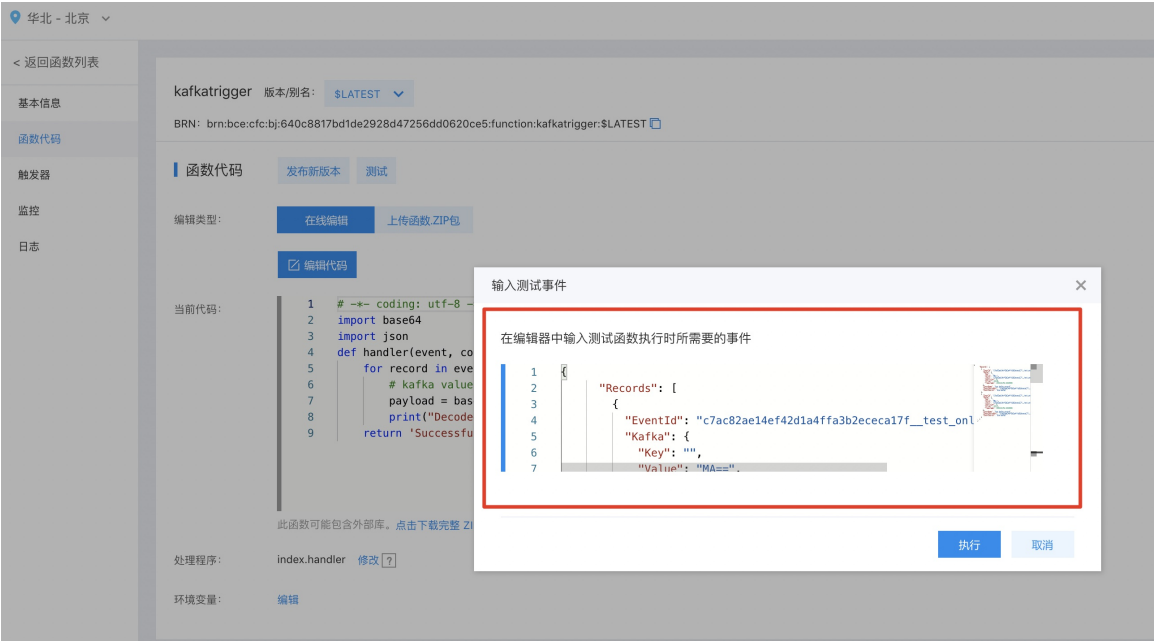
1. 点击测试按钮



2. 输入测试事件，并点击执行

测试内容如下


```
{
  "Records": [
    {
      "EventId": "c7ac82ae14ef42d1a4ffa3b2ececa17f__test_only_sl-0-43110",
      "Kafka": {
        "Key": "",
        "Value": "MA==",
        "Topic": "c7ac82ae14ef42d1a4ffa3b2ececa17f__test_only_sl",
        "Partition": 0,
        "Offset": 43110,
        "Timestamp": 1553151704.6529999
      },
      "EventName": "bce:kafka:record",
      "EventSourceBrn": "c7ac82ae14ef42d1a4ffa3b2ececa17f__test_only_sl",
      "EventSource": "bce:kafka"
    },
    {
      "EventId": "c7ac82ae14ef42d1a4ffa3b2ececa17f__test_only_sl-0-43111",
      "Kafka": {
        "Key": "",
        "Value": "MQ==",
        "Topic": "c7ac82ae14ef42d1a4ffa3b2ececa17f__test_only_sl",
        "Partition": 0,
        "Offset": 43111,
        "Timestamp": 1553151704.6529999
      },
      "EventName": "bce:kafka:record",
      "EventSourceBrn": "c7ac82ae14ef42d1a4ffa3b2ececa17f__test_only_sl",
      "EventSource": "bce:kafka"
    }
  ]
}
```

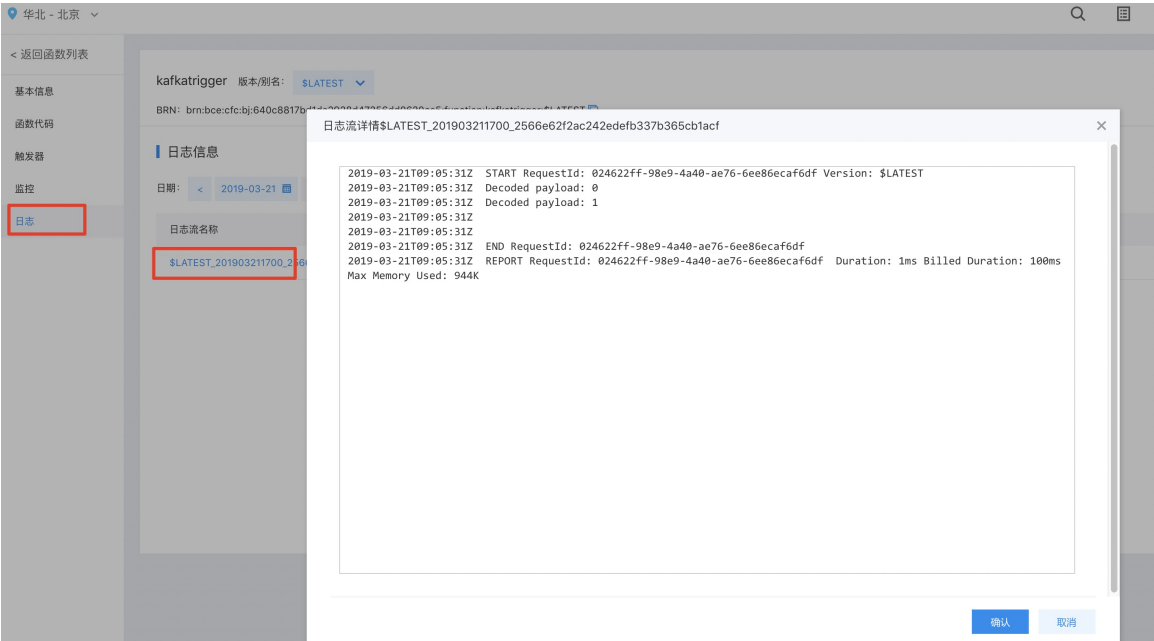


3. 查看

返回的执行结果

真实测试

在添加好触发器之后，需要设置日志存储，之后向 kafka topic 中发送消息，在日志中可以看到执行结果。



并且在触发器的界面您也可以看到最后的执行结果



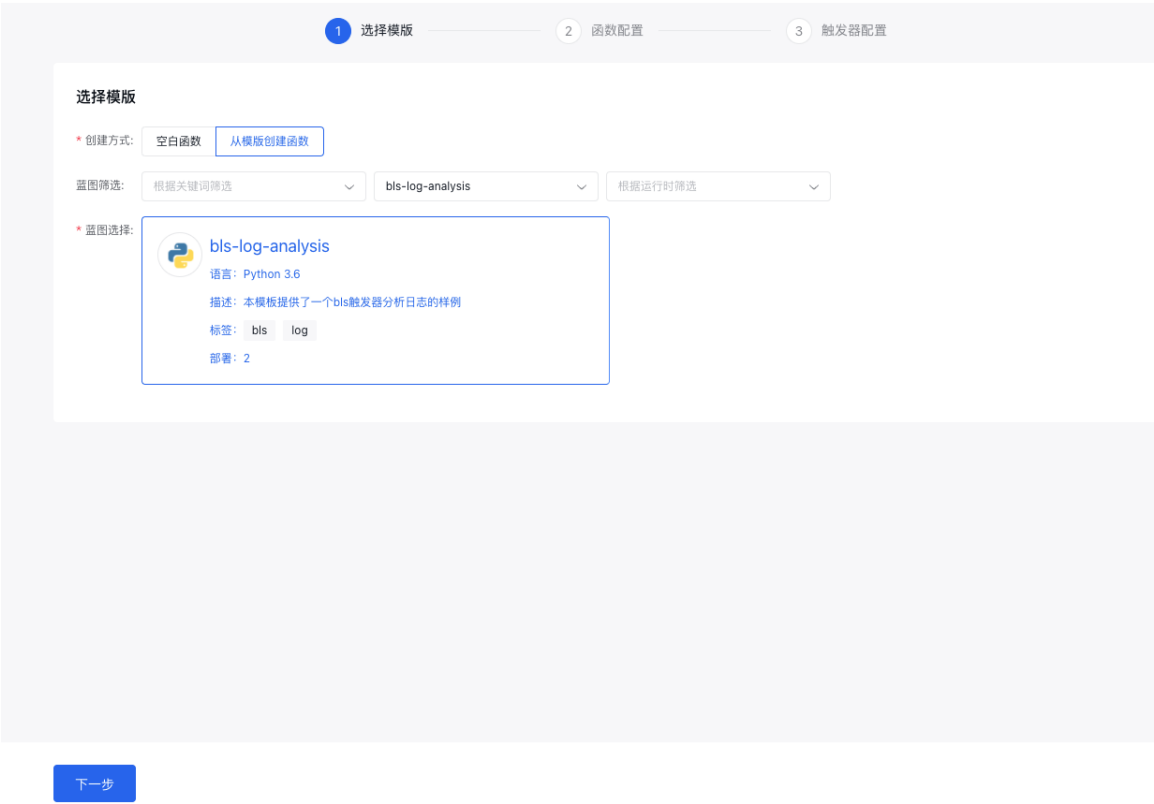
BLS触发器

BLS 触发器概述

日志服务（Baidu Log Service）是百度智能云提供的托管式日志收集与投递服务。函数创建BLS触发器后，CFC会定期下拉处理BLS对应日志集下日志流的消息。有关日志集和日志流的概念，您可以参考[BLS日志集](#)。BLS的openAPI您可以参考[API参考](#)。

配置 BLS 触发器

1. 登录管理控制台，选择“产品服务> 函数计算 CFC”，进入“函数列表”页面。
2. 点击创建函数，选择从模版创建函数，模版选择bls-log-analysis(该模版函数实现了从BLS中拉取日志数据并输出)。



3. 点击左侧导航栏中的“触发器”，进入函数配置页面，点击“新增触发器”。在弹出框中选择 BLS 触发器。



4. 之后在弹出框中填入触发器的各个参数、点击确认，触发器即创建完成，各个参数的解释如下：

- 名称：即触发器的名称。
- 日志集：每个触发器需要选择一个日志集作为触发源，并且日志集与函数需位于同一个地域。
- 日志流：选择该日志集下的日志流。
- 触发器间隔：处理BLS日志的时间间隔，当前仅支持分钟粒度触发，间隔1~10分钟。

新增触发器

* 触发器:

BLS触发器

* 触发器名称:

test

1. 只能包含字母，数字、下划线和中划线

2. 长度限制在1-30之间

* 日志集:

xflow_history

* 日志流:

history_42f6bc2cd374bfc80d9...

* 触发间隔:

1

分钟

当前仅支持分钟粒度触发，间隔1~10分钟

取消

确定

5. 在函数配置页面“触发器”一栏中，可以看到刚刚创建好的 BLS 触发器和一个定时任务触发器bls_default(该触发器负责函数触发执行，您无需修改)。

BLS触发器

编辑 删除

触发器名称: test

日志集: xflow_history

日志流: history_42f6bc2cd374bfc80d9967370fd8ff_xflow111

触发间隔: 1分钟

定时触发器

编辑 删除

规则名称: bls_default

计划表达式: rate(1 minute)

测试触发器

1. 点击测试按钮

blstrigger BRN: brn:bce:cf:bj:1a2cbf...

版本/别名: 版本 SLATEST 发布新版本 删除 测试 返回列表

信息

基本信息

当前版本/别名: SLATEST

运行时: Python 3.6

事件类型: event

服务名称: default

内存: 512 MB

超时时间: 10

日志存储: 无

日志存储路径: --

描述: --

配置信息

并发度: 10(申请留并发)

死信队列: --

CFS: --

异步策略配置: --

VPC网络: --

2. 输入测试事件，并点击执行，测试内容如下

```
{
  "ak": "***此处填写您的账号ak**",
  "sk": "***此处填写您的账号sk**",
  "host": "bls-log.bj.baidubce.com",
  "taskInterval": "1",
  "logrecord": "xflow_history",
  "logStreamName": "history_42f6bc2cd374bfc80d9967370fd8ff_xflow111",
  "token": ""
}
```

测试事件

测试事件

配置测试事件

操作类型:

新建测试事件

编辑测试事件

事件模板:

HelloWord模版

* 事件名称:

请输入事件名称

必填项

事件名称只能由数字、字母或下划线组成，且最大长度为64位

1

2

3

4

5

6

7

8

9

"ak":

"sk":

"host": "bbs-log.bj.baidubce.com",

"taskInterval": "10",

"logrecord":

"logStreamName":

"token": ""

取消

创建

3. 查看返回的结果

测试事件

测试事件

配置测试事件

事件模板:

HelloWord模版

1

2

3

4

5

6

7

8

9

"ak":

"sk":

"host": "bbs-log.bj.baidubce.com",

"taskInterval": "1",

"logrecord": "xflow_history",

"logStreamName": "history_42f6fbc2cd374bfc80d9967370fd8ff_xflow111",

"token": ""

✓ 执行结果: 成功

此区域下方显示你的函数执行的结果。

null

此区域下方显示你的函数执行过程中产生的日志。

START RequestId: c2ffa64f-00a5-4488-a63f-14fa0c3e53f9 Version: \$LATEST

{'isTruncated': False, 'result': []}

END RequestId: c2ffa64f-00a5-4488-a63f-14fa0c3e53f9

REPORT RequestId: c2ffa64f-00a5-4488-a63f-14fa0c3e53f9 Duration: 1.021s Billed Duration: 1.1s Max Memory Used: 6.5M

取消

执行

定时任务触发器

定时任务触发器

定时触发器

函数计算支持定时触发器，即在指定的时间点触发函数的执行。触发的时间使用 UTC 时区，计划的最小精度为 1 分钟。

使用限制

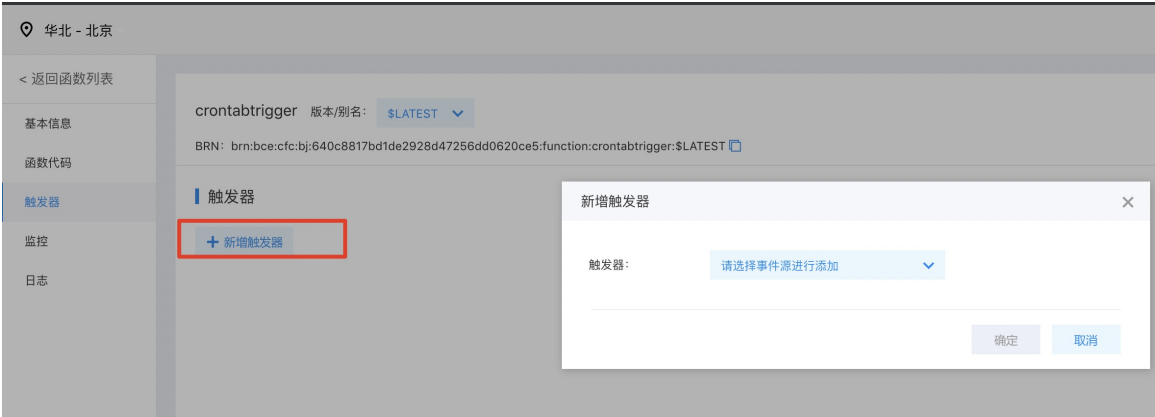
请注意同一函数下最多可创建10个定时触发器。

创建定时触发器

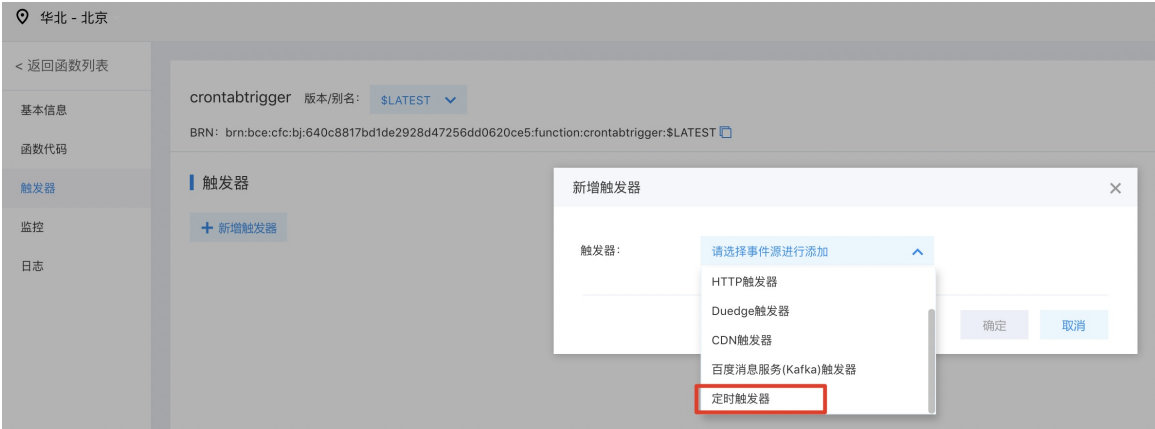
1. 登录管理控制台，选择“产品服务> 函数计算 CFC”，进入“函数列表”页面
2. 点击需要添加定时触发器的函数名称（即 crontabtrigger），进入函数详情页面。
3. 点击左侧导航栏中的“触发器”，进入函数配置页面。



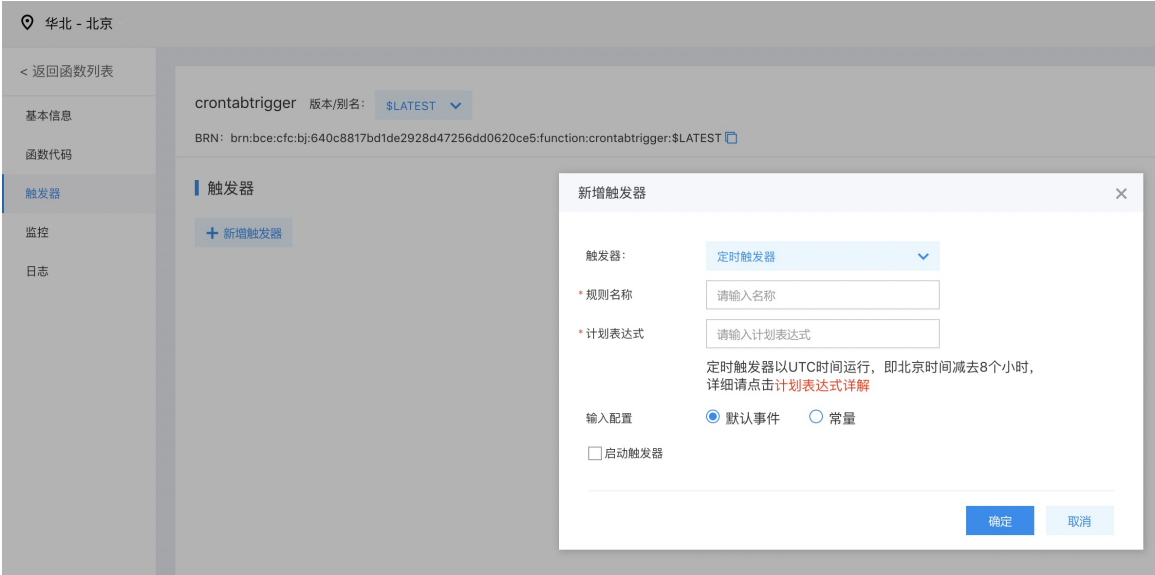
4. 在函数配置页面中最下方点击“新增触发器”。



5. 在弹出框中，点击下拉框“选择事件源进行添加”，选择定时触发器。

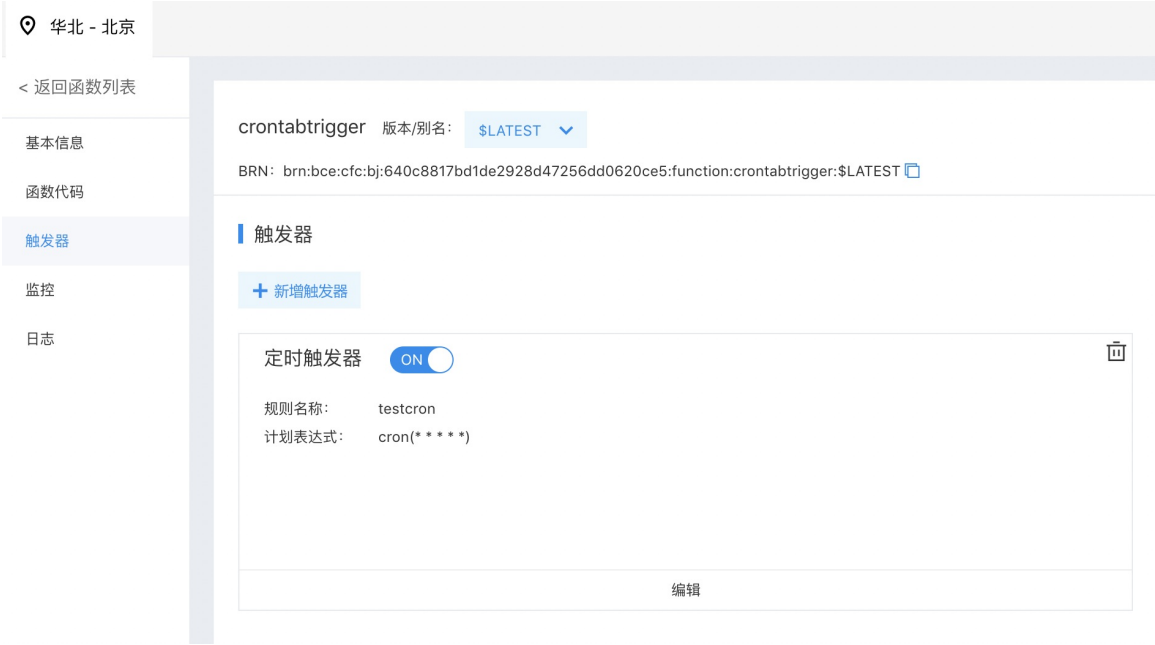


6. 之后在弹出框中配置好选项，并点击确认，完成触发器的创建。



- 规则名称:您要输入的定时触发器的名称testcron
- 计划表达式:详见[计划表达式](#)，此时输入cron(* * * * *)
- 输入配置：
 - 默认事件:定时任务触发器默认的event
 - 常量:您可以自行输入json格式的常量作为触发event
- 启动触发器：勾选之后触发器1-2分钟后即可生效

7. 在函数配置页面“触发器”一栏中，可以看到刚刚创建好的定时触发器及其信息



Event格式

默认事件

```
{
  "account": "c7ac82ae14ef42d1a4ffa3b2ececa17f", //用户account
  "uuid": "d9f671cc-1327-41f3-be33-f1d442f5bf8d", //定时器的uuid
  "function_brn": "brn:bce:cfc:bj:640c8817bd1de2928d47256dd0620ce5:function:crontabtrigger:$LATEST", //触发函数
  "source": "cfc:crontab", //固定值
  "version": "0", // 版本号
  "detail_type": "Scheduled Event", // 固定值
  "time": "2019-05-20T08:03:00Z", // 触发时间
  "id": "4ab17338-b889-4905-b3a5-a5a9cff68677" //事件ID
}
```

定时任务计划表达式

计划表达式

用户可以使用 cron 或 rate 表达式在函数计算中创建按计划自行触发的规则。所有表达式都使用 UTC 时区，计划的最小精度为 1 分钟。CFC提供支持 cron 表达式和 rate 表达式。cron 表达式支持的具体的执行您可以定义具体的那一天的那一分钟。而 Rate 表达式更容易定义，它以一定的间隔触发规则，例如每小时一次或每天一次。

rate表达式

Rate 表达式有两个必需字段。这些字段用空格分隔。

rate(1 minute) rate(value unit)		
字段	说明	取值
value	值	正数
unit	时间单位。如果是1 单位必须是单数 （例如，rate(1 day)），而不是复数（例如，rate(5 days)）	有效值：minute minutes hour hours day days

举个例子：

rate表达式	说明
rate(1 minute)	每隔1分钟
rate(5 minutes)	每隔5分钟
rate(1 hour)	每隔1小时
rate(1 day)	每隔1天
rate(10 hours)	每隔10小时

cron表达式

cron表达式有5个必填字段

字段	值	特殊字符
分钟	0-59	, - * /
小时	0-23	, - * /
日期	1-31	, - * ? /
月	1-12 或 JAN-DEC	, - * /
星期几	0-6 or SUN-SAT	, - * ? /

特殊字符的描述

特殊字符	描述	样例
,	逗号 表示列表	在月份中JAN、FEB 和 MAR 将包含 January、February 和 March
-	破折号 表示范围	在小时字段中使用5-10，表示UTC时间从5到10点
*	星号 包含该字段中的所有值	在分钟字段中， 将包含每分钟，您不能在"日期"和"星期几"字段中同时使用。如果您在一个中使用它，则必须在另一个中使用？
?	问号 表示不确定的值 与其他指定值一起使用	在"日期"字段中，您可以输入7，如果您不介意7日是星期几，则可以在"星期几"字段中输入？
/	斜杠 表示一个值的增量	在"分钟"字段中，您可以输入 1/10 ,它表示从1分开始，每隔10分钟执行

样例（注意：以下样例均使用UTC时区，Asia/Shangha时区比UTC时区多八个小时）

cron表达式(UTC时间)	解释
cron(0 10 * * ?)	每天上午的 10:00 (UTC) 触发；对应Asia/Shanghai为每天下午18:00执行
cron(30 10 * * ?)	每天上午的 10:30 (UTC) 触发 ；对应Asia/Shanghai为每天下午18:30执行
cron(10,11,12 * * * ?)	每小时的10分，11分，12分(UTC) 触发
cron(0 8 1 * ?)	每月第 1 天的上午 8:00 (UTC) 触发；对应Asia/Shanghai为每月第 1 天的下午16:00执行
cron(0/10 * * * ?)	每10分钟 (UTC) 触发
cron(0/10 8-10 ? * MON-FRI)	每星期一到星期五的上午 8:00 和 10:00 (UTC) 之间，每10分钟触发一次；对应Asia/Shanghai为每星期一到星期五的下午16:00和18:00之间执行，每10分钟触发一次

[使用控制台创建定时触发器](#)

HTTP触发器

HTTP触发器开发参考

HTTP触发器实现了将某个函数关联到一个 URL 上（包含相应的 CRUD 操作），它可以接收 HTTP 请求，根据 HTTP 方法、URL，找到匹配的函数将 HTTP 相关信息传入并执行函数，获取执行结果，将函数执行结果包装为 HTTP 返回响应。创建HTTP触发器的核心步骤包括[HTTP触发器配置](#)和[用户代码配置](#)，以下将为您分别介绍。

HTTP触发器参数配置

URL路径规则

URL参数的配置支持以下几种格式：

1. **简单URL**，不包含参数，例如：/user。
2. **带参数的URL**，使用{param}指定路径参数，例如 /user/{id}/posts 中，{id} 即为匹配参数。当请求 /user/123/posts 时，会识别参数id: 123。
3. **带贪婪匹配路径参数的URL**，使用{param+}指定贪婪匹配路径参数，它会匹配当前位置直到 URL 结尾的所有剩余部分，一个URL配置示例，如 /path/{file+},如果请求的URL路径为/file/aaa/bbb/cccdde时，会在HTTP触发器Event事件中，其 pathParameters参数中识别到file字段被设置为aaa/bbb/cccdde。

HTTP方法

HTTP 触发器支持DELETE、GET、HEAD、OPTIONS、PATCH、POST、PUT方式触发函数。HTTP触发器时可以选择多个，之后触发器会匹配相应的请求方法集合。

身份验证

可以选择不验证，或者IAM验证。选择不验证时，会在最终的触发器信息显示中显示 不验证。

二进制请求 body 默认情况下，HTTP 触发器会以纯文本的形式处理 HTTP 请求体，如果请求体内包含非纯文本内容（例如二进制文件上传），并需要在函数中进行处理，可以勾选此选项。勾选后，HTTP 触发器会使用 Base64 编码 HTTP 请求体。用户代码中 event 参数中，body 项会是 Base64 编码后的请求体内容，而 isBase64Encoded 会被设为 true。

🔗 用户代码中的配置

在用户代码中需要设置响应的handler来处理由HTTP触发器转发而来的请求，HTTP触发器可以将整个客户端HTTP请求映射到后端函数的需要的输入 event 参数。参数的说明如下。

event的格式

Event:

```
{
  "resource": "配置的HTTP触发器的资源路径",
  "path": "HTTP请求的URL路径",
  "httpMethod": "本次http触发器请求使用的HTTP方法",
  "headers": {请求头},
  "queryStringParameters": {query string 参数},
  "pathParameters": {代理路径参数},
  "requestContext": {请求上下文},
  "body": "请求体",
  "isBase64Encoded": "请求体是否为Base64编码，这里固定为false，暂不支持二进制格式"
}
```

参数	说明	示例
resource	配置的HTTP触发器的资源路径	/test/{proxy+}
path	HTTP请求的URL路径	/test/aaa/bbb
httpMethod	HTTP请求的方法	GET
headers	该参数为HTTP请求的请求头的列表。以键值对的形式显示，当一个键存在多个值时，值之间使用逗号分隔。	{"header1":"aaa", "header2":"bbb"}
queryStringParameters	HTTP请求的查询参数，例如，如果请求URL为 https://{id}.cfc-execute.{region}.baidubce.com/example?key1=value1 ，则queryStringParameters是URL中?结束的键值对，其键为key1，值为value1。当一个键存在多个值时，值之间会使用逗号分隔。	{"key1": "value1", "key2": "value2,value3"}
queryStringParameters	HTTP请求的Path路径参数，例如，如果请求URL为 https://{id}.cfc-execute.{region}.baidubce.com/test/{id}/get?key1=value1	{"id": "1"}
requestContext	请求上下文，一个包含有关请求的附加信息的对象，例如requestId、请求的时间以及通过授权的调用者身份。	
requestContext.stage	请求上下文字段，表明平台名称，默认是cfc	cfc
requestContext.requestId	请求上下文字段，表明请求id，	"f6cb833e-f818-4b80-b30c- fff437e69886"
requestContext.resourcePath	请求上下文字段，表明HTTP触发器设置的资源路径	/test/{proxy+}
requestContext.httpMethod	请求上下文字段，表明HTTP请求方法	GET
requestContext.apiId	请求上下文字段，表明HTTP请求API ID	"12345"
body	HTTP请求体	"hello world"
isBase64Encoded	请求体是否为Base64编码，这里固定为false，暂不支持二进制格式	false

其中请求头包括一个自定义头来标识请求：

```
{  
  "X-Bce-Request-Id": "6c2d4826-xxxx-4c2b-8039-9ca98ded6954"  
}
```

函数计算HTTP请求映射逻辑

函数计算会将HTTP请求映射成event事件对象传给请求处理程序（Handler），映射逻辑如下：

- HTTP请求头映射为event结构中的headers。
- HTTP的请求参数映射为queryStringParameters。

- HTTP的路径参数映射为pathParameters。
- HTTP请求的上下文信息映射为requestContext。
- POST请求的请求体映射为body。

说明：用户代码可依赖Event中现有参数，但HTTP触发器在将来可能增加参数，用户代码在此处需保证开放性，并对参数数量无硬性依赖。

配置举例

1 在控制台为某个函数配置了参数为如下内容的一个HTTP触发器，拿到http触发器URL。

```
URL路径: "http://7grmt1xfky5jf.cfc-execute.gz.baidubce.com/test/{proxy+}"
HTTP方法: GET
身份验证: 不验证
```

2 使用curl作为客户端向CFC发出HTTP请求。

```
curl http://7grmt1xfky5jf.cfc-execute.gz.baidubce.com/test/aaa/bbb?a=111
```

该请求在CFC侧转换的对应的event的内容如下

```
{
  "resource": "/test/{proxy+}",
  "path": "/test/aaa/bbb",
  "httpMethod": "GET",
  "headers": {
    "Accept": "text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8",
    "Accept-Encoding": "gzip, deflate, br",
    "Accept-Language": "zh-CN,zh;q=0.9,en;q=0.8",
    "Connection": "close",
    "Upgrade-Insecure-Requests": "1",
    "User-Agent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_13_4) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/70.0.3538.67 Safari/537.36",
    "X-Bce-Request-Id": "f6cb833e-f818-4b80-b30c-fff437e69886"
  },
  "queryStringParameters": {
    "a": "111"
  },
  "pathParameters": {
    "proxy": "aaa/bbb."
  },
  "requestContext": {
    "stage": "cfc",
    "requestId": "f6cb833e-f818-4b80-b30c-fff437e69886",
    "resourcePath": "/test/{proxy+}",
    "httpMethod": "GET",
    "apId": "7grmt1xfky5jf",
    "sourceIp": "12.34.56.78",
  },
  "body": "",
  "isBase64Encoded": false
}
```

使用 HTTP 触发器时的函数返回格式

HTTP 触发器支持两种格式的后端函数返回：简易格式和完整格式。

简易格式

HTTP 触发器会获取函数返回，并将其作为纯文本 HTTP 响应发回客户端。此时，响应状态码固定为 200，并包含 Content-Type: text/plain 响应头部。

使用简易格式时，无法在函数中自定义返回状态码和返回头部，也无法返回二进制响应，如果需要自定义这些，请使用下文描述的完整格式返回。

Node.js 范例

```
exports.handler = (event, context, callback) => {  
  callback(null, "Hello world!");  
};
```

完整格式

使用完整格式定义 HTTP 响应，后端处理函数需要按照以下格式返回 JSON：

```
{  
  "isBase64Encoded": true|false,  
  "statusCode": httpStatusCode,  
  "headers": { "headerName": "headerValue", ... },  
  "body": "..."  
}
```

- isBase64Encoded：表示 body 是否进行了 Base64 编码，如果返回纯文本，则设为 false，如果需要返回二进制内容，则需要设为 true，并自行将返回 body 进行 Base64 编码后，设置到 body 字段上。
- statusCode：HTTP 响应状态码，必填项，例如 200 表示 OK，404 表示 Not Found 等。
- headers：以 K-V 的形式定义响应中需要的额外头部，例如 Content-Type 等。不需要额外响应头部时，可以省略。如果需要启用 CORS，必须在 headers 中添加 Access-Control-Allow-Origin:domain-name。其中，domain-name 可以设置成通配符*。
- body：返回给客户端的响应体的内容，必填项。

Node.js 范例

```
exports.handler = (event, context, callback) => {  
  callback(null, {  
    "isBase64Encoded": false,  
    "statusCode": 200,  
    "headers": { "X-Custom-Header": "headerValue" },  
    "body": "hello"  
  });  
};
```

Python 范例

```
import json

def handler(event, context):
    resp = {
        "isBase64Encoded": False,
        "statusCode": 200,
        "headers": {
            "X-Custom-Header": "headerValue"
        },
        "body": "hello"
    }
    return json.dumps(resp)
```

PHP 范例

```
<?php
function handler($event, $context) {
    $resp = [
        "isBase64Encoded" => FALSE,
        "statusCode" => 200,
        "headers" => [
            "X-Custom-Header" => "headerValue",
        ],
        "body" => "hello",
    ];
    return json_encode($resp, JSON_FORCE_OBJECT);
}
```

Lua 范例

```
json = require('cjson')

function handler(event, context)
    respHeaders = {}
    respHeaders["X-Custom-Header"] = "headerValue"
    resp = {
        isBase64Encoded = false,
        statusCode = 200,
        headers = respHeaders,
        body = "hello",
    }
    return json.encode(resp)
end
```

PowerShell 范例

```
$respHeaders = @{}
$respHeaders["X-Custom-Header"] = "headerValue"
$resp = @{}
    isBase64Encoded = $FALSE;
    statusCode = 200;
    headers = $respHeaders;
    body = "hello";
}
$respJson = $resp | ConvertTo-Json
Write-Host $respJson
```

HTTP触发器使用示例

用户可以为新建的函数或已有函数配置HTTP触发器，创建函数的流程可以具体参考 [创建函数](#) 这里不再赘述。

这里假设用户已完成一个名为httptrigger-helloworld的Nodejs函数的创建，以下内容以此为前提，指引用户在CFC控制台在函数管理页面中为函数配置触发器。接下来，我们将通过三步来完成一个触发器的设置。

编写处理函数

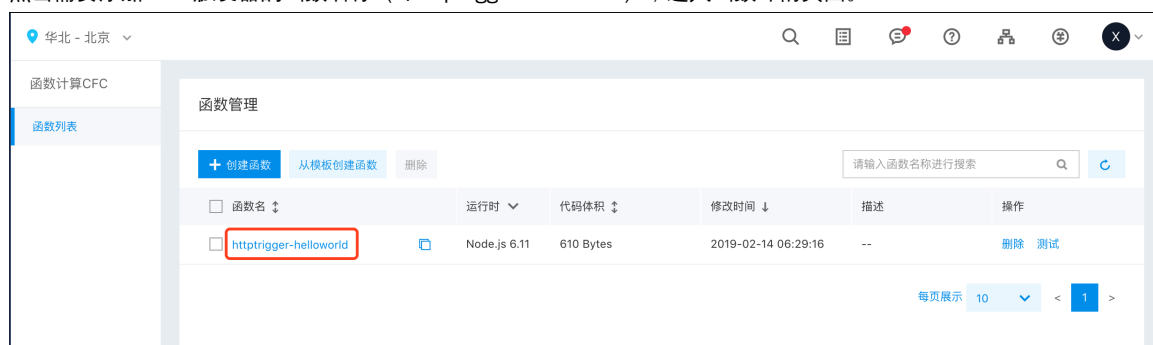
使用HTTP触发器需要用户在函数代码中作出相应的调整，按照响应格式来编写handler函数，具体可参考 [用户代码中的配置](#)。

登录管理控制台，选择“产品服务>云函数计算 CFC”，进入“函数列表”页面；在“函数列表”页面。点击名称为httptrigger-helloworld的函数，进入函数详情页面。在函数详情页中编写HTTP触发器对应的handler，以对触发操作返回适当的响应，之后点击右下角保存按钮完成函数的修改操作。

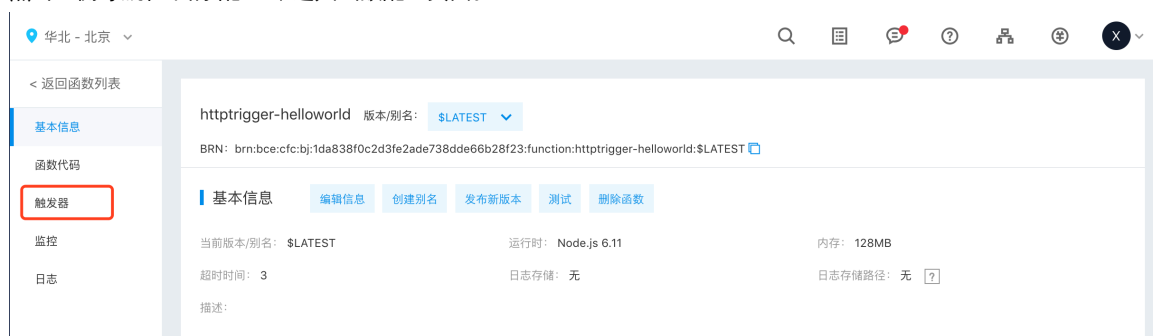
```
exports.handler = (event, context, callback) => {  
  callback(null, {  
    "isBase64Encoded": false,  
    "statusCode": 200,  
    "headers": { "X-Custom-Header": "headerValue" },  
    "body": "hello"  
  });  
};
```

配置HTTP触发器

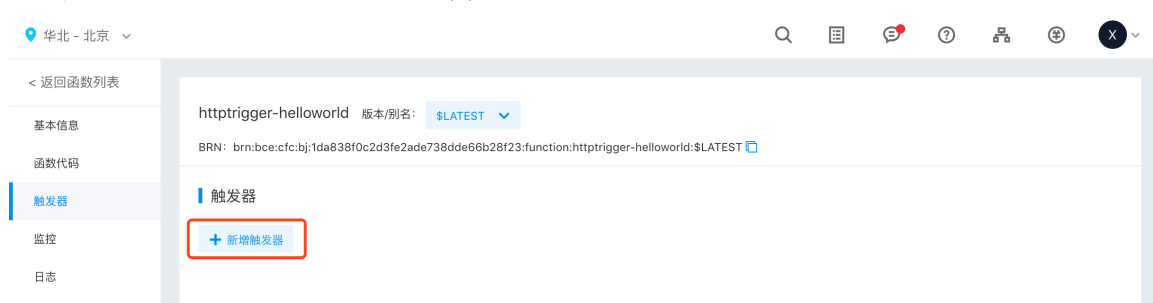
1. 登录管理控制台，选择“产品服务>云函数计算 CFC”，进入“函数列表”页面。
2. 点击需要添加HTTP触发器的函数名称（即httptrigger-helloworld），进入函数详情页面。



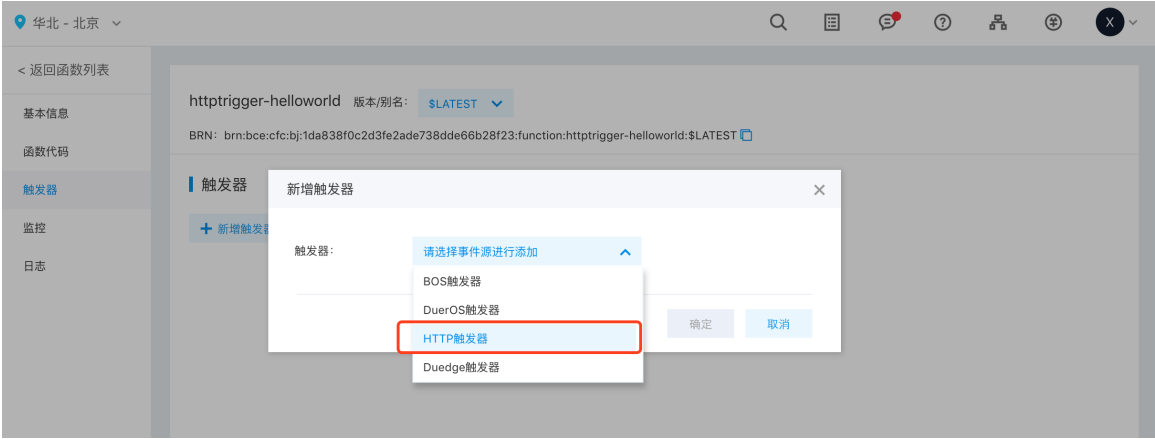
3. 点击左侧导航栏中的“配置”，进入函数配置页面。



4. 在函数配置页面中最下方的“触发器”一栏中，点击“新增触发器”。



5. 在弹出框中，点击下拉框“选择事件源进行添加”，选择HTTP触发器。



6. 之后在弹出框中配置好URL路径，HTTP方法和身份验证选项，并点击确认，完成触发器的创建。

- URL路径:用户定义的访问路径，例如 /helloworld
- HTTP方法:触发器需要匹配的HTTP方法，可以通过下拉框选择一项或多项HTTP方法，这里我们选择“GET”
- 身份验证:触发时是否需要验证身份，这里我们选择“不验证”
- 若您需要使用二进制body，则可以勾选「使用二进制body」选项。

新增触发器

触发器:

HTTP触发器

* URL路径:

/hello/world

填写以 / 开头的匹配路径，完整 URL 在触发器创建完成后可以查看
使用{param}指定路径参数，例如 /user/{id}/posts 中，{id} 即为匹配参数
使用{param+}指定贪婪匹配路径参数，它会匹配当前位置直到 URL 结尾的
所有剩余部分，例如 /path/{filepath+}

* HTTP方法:

请选择HTTP方法

* 身份验证:

☒ 不验证

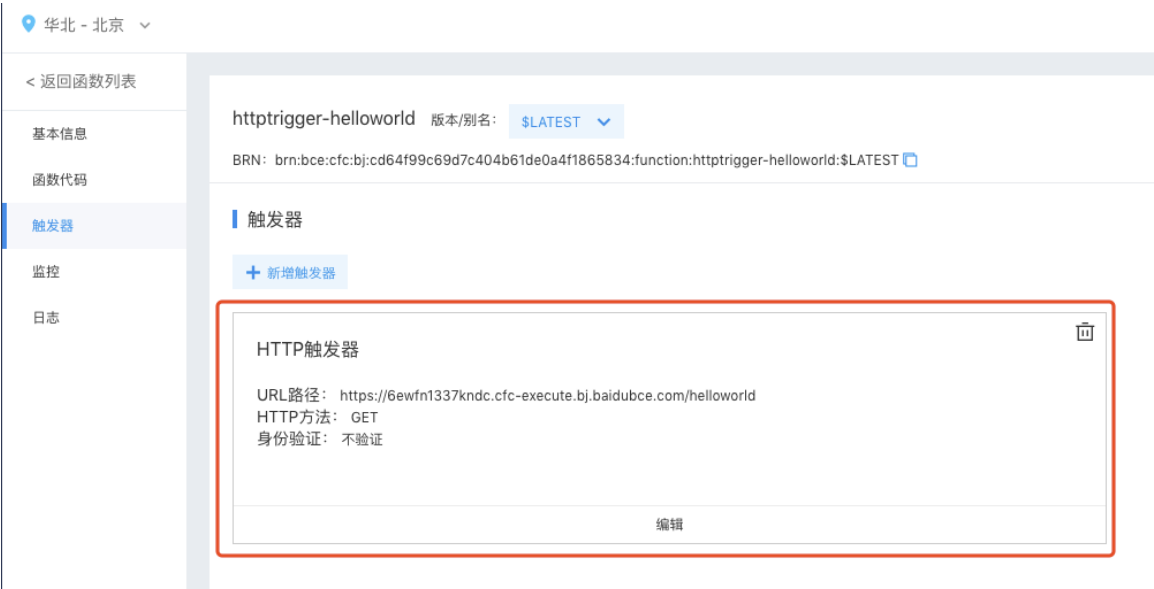
☐ IAM

☐ 使用二进制请求body

确定

取消

7. 在函数配置页面“触发器”一栏中，可以看到刚刚创建好的HTTP触发器及其信息，包括访问的URL路径，HTTP方法和身份验证选项。



🔗 测试触发器

以上两步配置完成后，即可通过客户端向HTTP触发器的URL路径发起请求。例如：

```
curl -i https://6ewfn1337kndc.cfc-execute.bj.baidubce.com/helloworld
```

```
#Response

HTTP/1.1 200 OK
Connection: keep-alive
Content-Type: text/plain; charset=utf-8
Date: Tue, 23 Oct 2018 10:34:30 GMT
Server: nginx/1.9.2
Tracecode: 20708716030189314314102318
Vary: Accept-Encoding
X-Bce-Request-Id: 6f65d518-b782-4b82-950b-afce4b6e285b
X-Custom-Header: headerValue
Content-Length: 5

hello
```

以上内容为一个Nodejs函数httptrigger-helloworld创建了一个HTTP触发器，详细的HTTP触发器配置方法可以参考[HTTP触发器参数配置](#)。

DuerOS触发器

DuerOS触发器是百度云函数计算CFC为[DuerOS技能开放平台](#)开发者用户提供的事件触发器。

百度开放的智能生态有两大开发者支撑平台：一个是“小度技能开放平台”，一个是“百度智能云”。借助两大平台，您可以在搭载DuerOS的智能硬件上开发出丰富的功能，实现AI能力的场景化落地。为了更好地支持开发者，百度云CFC与DuerOS技能开放平台深度整合，提供一站式解决方案，您可以很方便的将在技能开放平台上开发好的技能使用DuerOS触发器与CFC函数绑定。



例如语音响应，您只需要在技能开放平台上完成技能的开发、在百度云函数计算完成函数开发，同时完成相互配置，即可轻松打通业务流。



🔗 配置使用步骤

使用函数计算CFC创建一个自定义技能的步骤如下，详见[DuerOS开放平台开发者使用CFC构建Custom-Skill典型实践](#)，您可以按照典型实践配置并使用该触发器：

- 开通CFC服务
- 创建函数
- 测试函数
- 查询BRN
- 绑定DuerOS 技能 和 CFC 函数
- 测试 DuerOS 技能

🔗 更多例子

关于如何使用DuerOS触发器，您还可以参考更多例子：

- [DuerOS技能开放平台-自定义技能-技能部署-百度云CFC](#)
- [DuerOS技能开放平台-自定义技能-技能部署-CFC部署典型实践 \(Java\)](#)

DuEdge触发器

DuEdge触发器是百度云CFC为DuEdge的用户提供的触发器。DuEdge是百度旗下的边缘计算网络系统，平台详见[官网](#)。

🔗 配置使用步骤

DuEdge触发器配置使用步骤，可见[链接](#)，您可以根据链接中的步骤配置使用该触发器。

百度消息服务触发器使用示例

🔗 百度消息服务触发器说明

百度消息服务是全兼容 Apache Kafka 的分布式、高可扩展、高通量的消息托管服务。

如果在百度消息服务的 Topic 上检测到记录，您可以使用 CFC 函数从此 Topic 读取成批的数据来处理，并且 CFC 会定期轮询（每秒一次）Topic 中的新纪录。

🔗 百度消息服务触发器创建

用户可以为新建的函数或已有函数配置百度消息服务触发器，创建函数的流程可以具体参考 [从头创建函数](#)，这里不再赘述。

这里假设您已经创建了一个名为 kafkatrigger 的函数，以下内容以此为前提，将引导您在 CFC 控制台在函数管理页面中为函数配置百度消息服务触发器。接下来，我们将通过以下步骤来完成一个触发器的设置。

🔗 编写处理函数

1. 登录管理控制台，选择“产品服务>云函数计算 CFC”，进入“函数列表”页面
2. 在“函数列表”页面。点击名称为 kafkatrigger 的函数，进入函数详情页面。

在函数详情页中编写百度消息服务触发器对应的 handler，以对触发操作返回适当的响应，之后点击右下角保存按钮完成函数的修改操作。

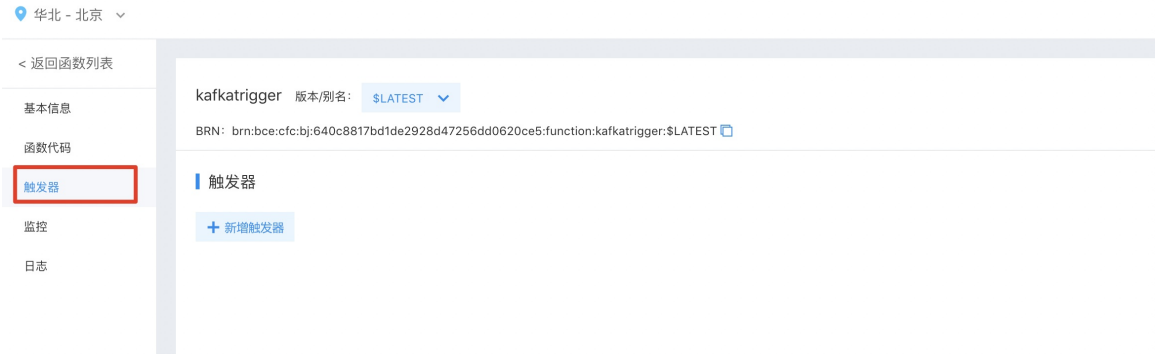
```
# -*- coding: utf-8 -*-
import base64
import json
def handler(event, context):
    for record in event['Records']:
        # kafka value is base64 encoded so decode here
        payload = base64.b64decode(record['Kafka']['Value'])
        print("Decoded payload: " + payload)
    return 'Successfully processed {} records.'.format(len(event['Records']))
```

注：百度消息服务触发器的event消息体

```
{
  "Records": [
    {
      "EventId": "c7ac82ae14ef42d1a4ffa3b2ececa17f__test_only_sl-0-43110",
      "Kafka": {
        "Key": "", //base64后的key
        "Value": "MA==", //base64后的value
        "Topic": "c7ac82ae14ef42d1a4ffa3b2ececa17f__test_only_sl",
        "Partition": 0,
        "Offset": 43110,
        "Timestamp": 1553151704.6529999
      },
      "EventName": "bce:kafka:record",
      "EventSourceBrn": "c7ac82ae14ef42d1a4ffa3b2ececa17f__test_only_sl",
      "EventSource": "bce:kafka"
    },
    {
      "EventId": "c7ac82ae14ef42d1a4ffa3b2ececa17f__test_only_sl-0-43111",
      "Kafka": {
        "Key": "",
        "Value": "MQ==",
        "Topic": "c7ac82ae14ef42d1a4ffa3b2ececa17f__test_only_sl",
        "Partition": 0,
        "Offset": 43111,
        "Timestamp": 1553151704.6529999
      },
      "EventName": "bce:kafka:record",
      "EventSourceBrn": "c7ac82ae14ef42d1a4ffa3b2ececa17f__test_only_sl",
      "EventSource": "bce:kafka"
    }
  ]
}
```

配置百度消息服务触发器

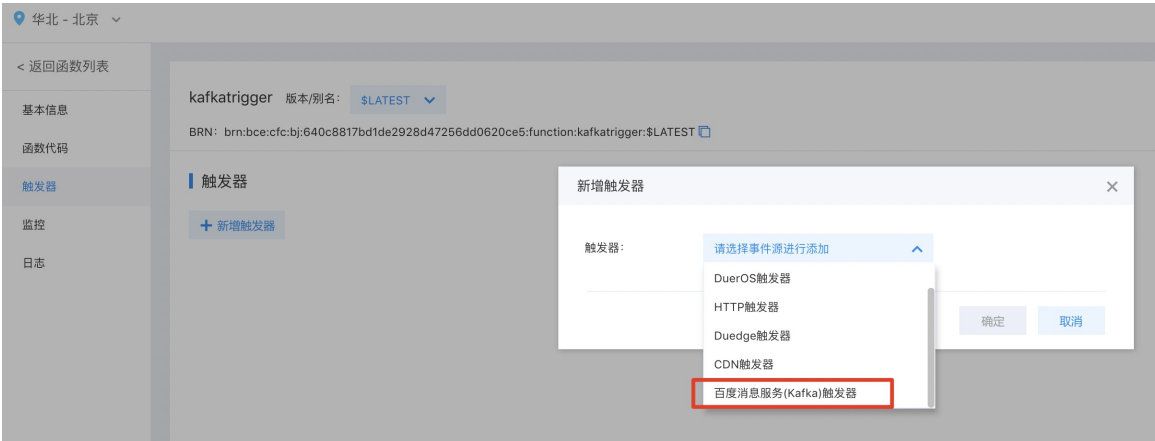
1. 登录管理控制台，选择“产品服务> 函数计算 CFC”，进入“函数列表”页面
2. 点击需要添加百度消息服务触发器的函数名称（即 kafkatrigger），进入函数详情页面。
3. 点击左侧导航栏中的“触发器”，进入函数配置页面。



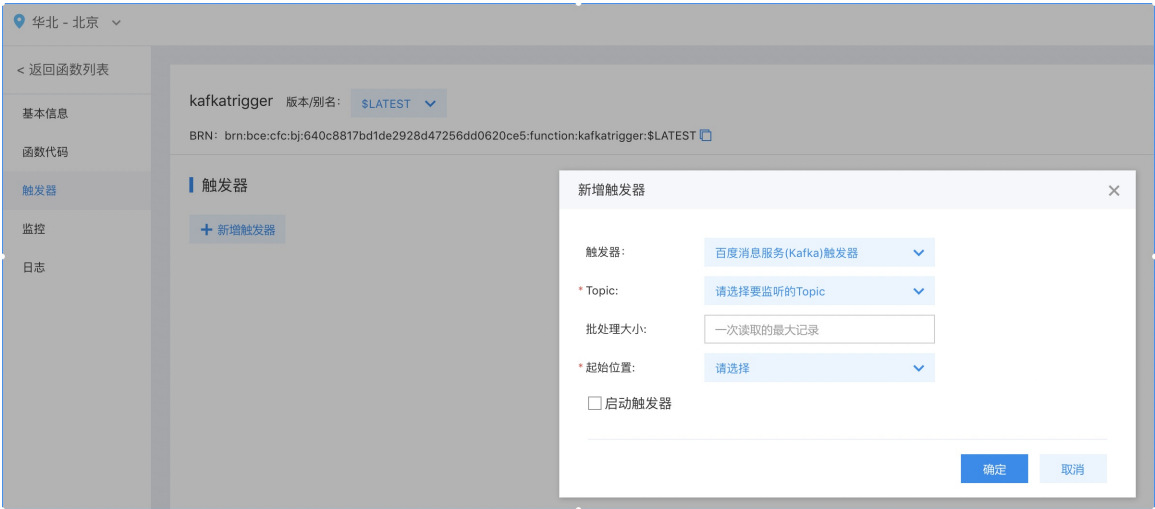
4. 在函数配置页面中最下方点击“新增触发器”。



5. 在弹出框中，点击下拉框“选择事件源进行添加”，选择百度消息服务触发器。



6. 之后在弹出框中配置好选项，并点击确认，完成触发器的创建。 - Topic 选择您将要监听的百度消息服务 Topic - 批处理大小：从 Topic 中一次读取的最大记录数，1-1000 - 起始位置：在 Topic 中开始读取的位置，最新记录对应 kafka 的 OffsetNewest，最老记录对应 kafka 的 OffsetOldest - 启用触发器:是否直接启动触发器，建议先关闭触发器以便测试



7. 在函数配置页面“触发器”一栏中，可以看到刚刚创建好的百度消息服务触发器及其信息

< 返回函数列表

基本信息

函数代码

触发器

监控

日志

kafkatrigger 版本/别名: \$LATEST

BRN: brn:bce:cf:bj:640c8817bd1de2928d47256dd0620ce5:function:kafkatrigger:\$LATEST

触发器

+ 新增触发器

百度消息服务触发器

源Topic: c7ac82ae14ef42d1a4ffa3b2ececa17f__test_only_sl

批处理大小: 10

最后处理结果:

是否开启触发器: ON

测试触发器

模拟测试

1. 点击测试按钮

华北 - 北京

< 返回函数列表

基本信息

函数代码

触发器

监控

日志

kafkatrigger 版本/别名: \$LATEST

BRN: brn:bce:cf:bj:640c8817bd1de2928d47256dd0620ce5:function:kafkatrigger:\$LATEST

函数代码

发布新版本

测试

编辑类型:

在线编辑

上传函数.ZIP包

编辑代码

当前代码:

```
1  -*- coding: utf-8 -*-
2  import base64
3  import json
4  def handler(event, context):
5      for record in event['Records']:
6          # kafka value is base64 encoded so decode here
7          payload = base64.b64decode(record['Kafka']['Value'])
8          print("Decoded payload: " + payload)
9          return 'Successfully processed {} records.'.format(len(event['Records']))
```

此函数可能包含外部库。点击下载完整 ZIP 程序包

处理程序:

index.handler 修改

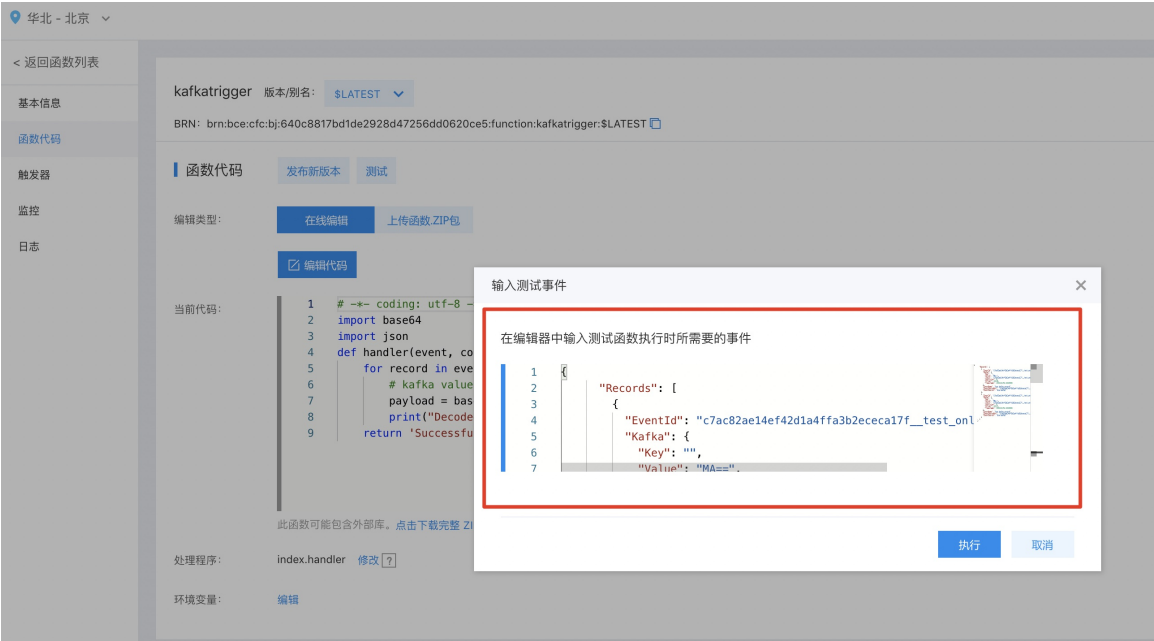
环境变量:

编辑

2. 输入测试事件，并点击执行

测试内容如下

```
{
  "Records": [
    {
      "EventId": "c7ac82ae14ef42d1a4ffa3b2ececa17f__test_only_sl-0-43110",
      "Kafka": {
        "Key": "",
        "Value": "MA==",
        "Topic": "c7ac82ae14ef42d1a4ffa3b2ececa17f__test_only_sl",
        "Partition": 0,
        "Offset": 43110,
        "Timestamp": 1553151704.6529999
      },
      "EventName": "bce:kafka:record",
      "EventSourceBrn": "c7ac82ae14ef42d1a4ffa3b2ececa17f__test_only_sl",
      "EventSource": "bce:kafka"
    },
    {
      "EventId": "c7ac82ae14ef42d1a4ffa3b2ececa17f__test_only_sl-0-43111",
      "Kafka": {
        "Key": "",
        "Value": "MQ==",
        "Topic": "c7ac82ae14ef42d1a4ffa3b2ececa17f__test_only_sl",
        "Partition": 0,
        "Offset": 43111,
        "Timestamp": 1553151704.6529999
      },
      "EventName": "bce:kafka:record",
      "EventSourceBrn": "c7ac82ae14ef42d1a4ffa3b2ececa17f__test_only_sl",
      "EventSource": "bce:kafka"
    }
  ]
}
```

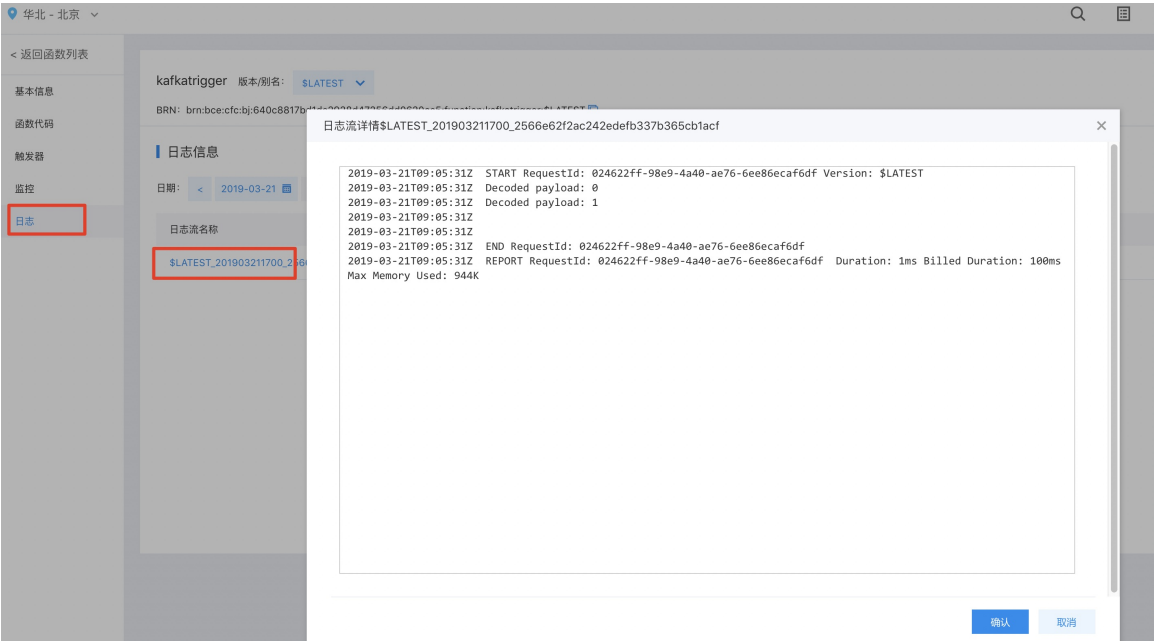


3. 查看

返回的执行结果

🔗 真实测试

在添加好触发器之后，需要设置日志存储，之后向 kafka topic 中发送消息，在日志中可以看到执行结果。



并且在触发器的界面您也可以看到最后的执行结果



CDN触发器使用示例

用户可以为新建的函数或已有函数配置CDN触发器，创建函数的流程可以具体参考 [从头创建函数](#) 这里不再赘述。

这里假设用户已完成一个名为hello-cdn的Python函数的创建，以下内容以此为前提，指引用户在CFC控制台在函数管理页面中为函数配置触发器。接下来，我们将通过三步来完成一个触发器的设置。

编写处理函数

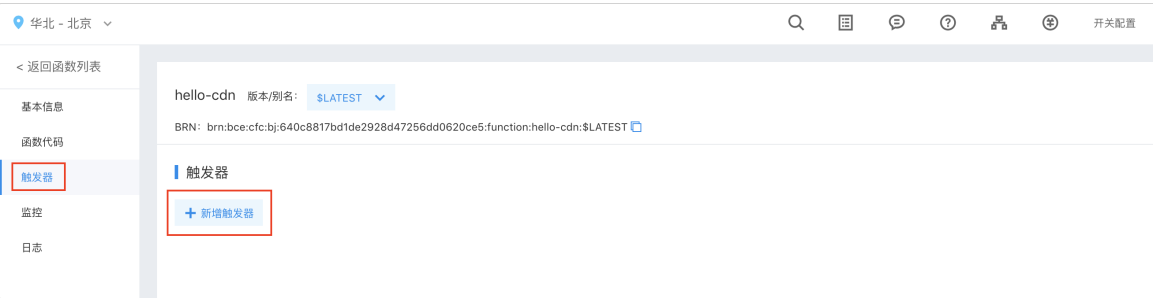
使用CDN触发器需要用户在函数代码中作出相应的调整，按照CDN的事件格式来编写handler，具体可参考 [CDN事件说明](#)。

1. 登录管理控制台，选择“产品服务>云函数计算 CFC”，进入“函数列表”页面。
2. 在“函数列表”页面，点击名称为hello-cdn的函数，进入函数详情页面。
3. 在函数详情页中编写函数代码，以对触发操作做出适当的处理，之后点击右下角保存按钮完成函数的修改操作。

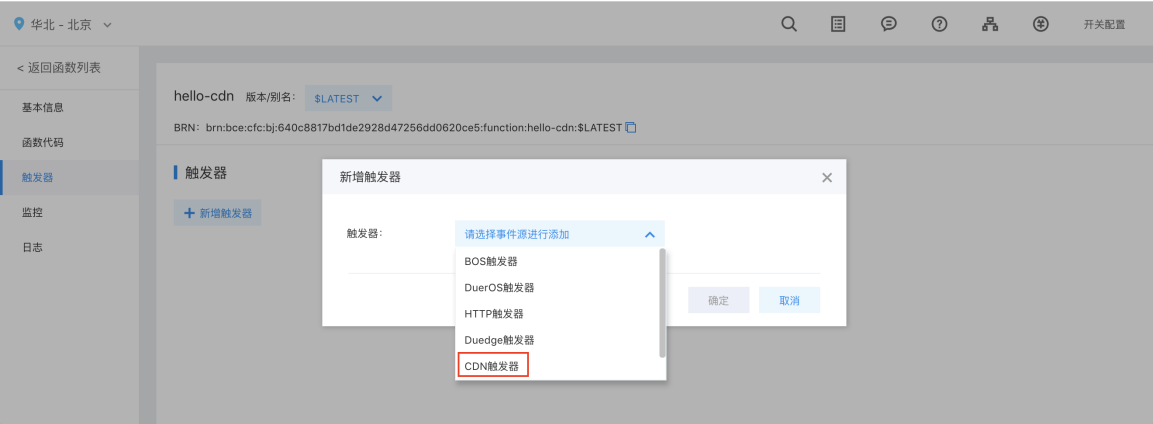
```
# -*- coding: utf-8 -*-
def handler(event, context):
    for record in event['events']:
        domain = record['eventDetail']['domain']
        status = record['eventDetail']['status']
        print("current status of domain '%s' is '%s'" % (domain, status))
    return 'Successfully processed {} records.'.format(len(event['events']))
```

配置CDN触发器

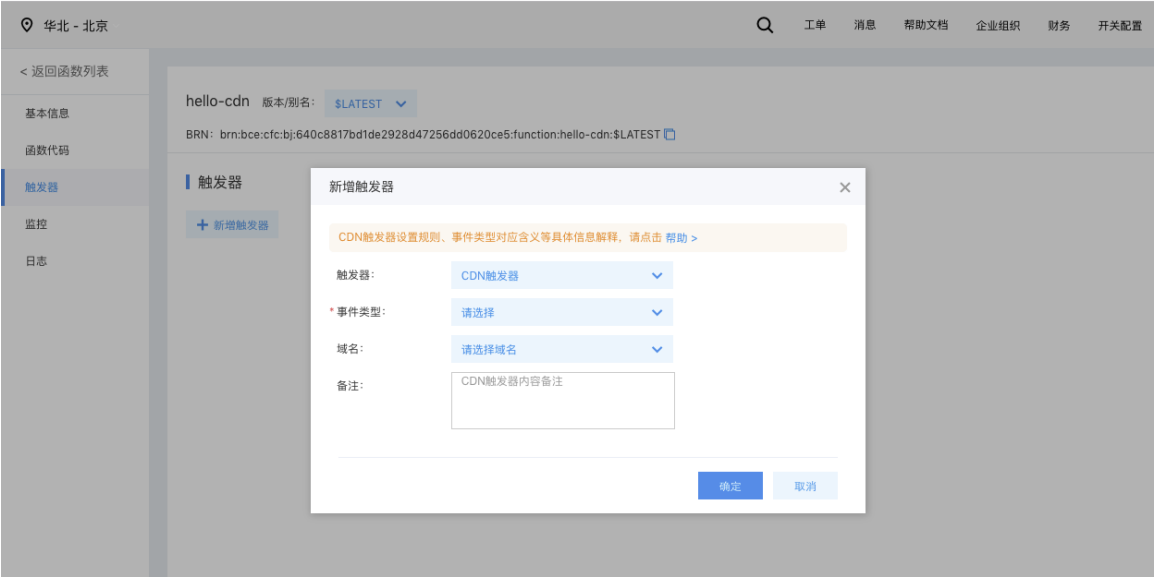
- 1. 登录管理控制台，选择“产品服务> 函数计算 CFC”，进入“函数列表”页面。
- 2. 点击需要添加CDN触发器的函数名称（即hello-cdn），进入函数详情页面。
- 3. 点击左侧导航栏中的“触发器”，进入函数配置页面，点击“新增触发器”。



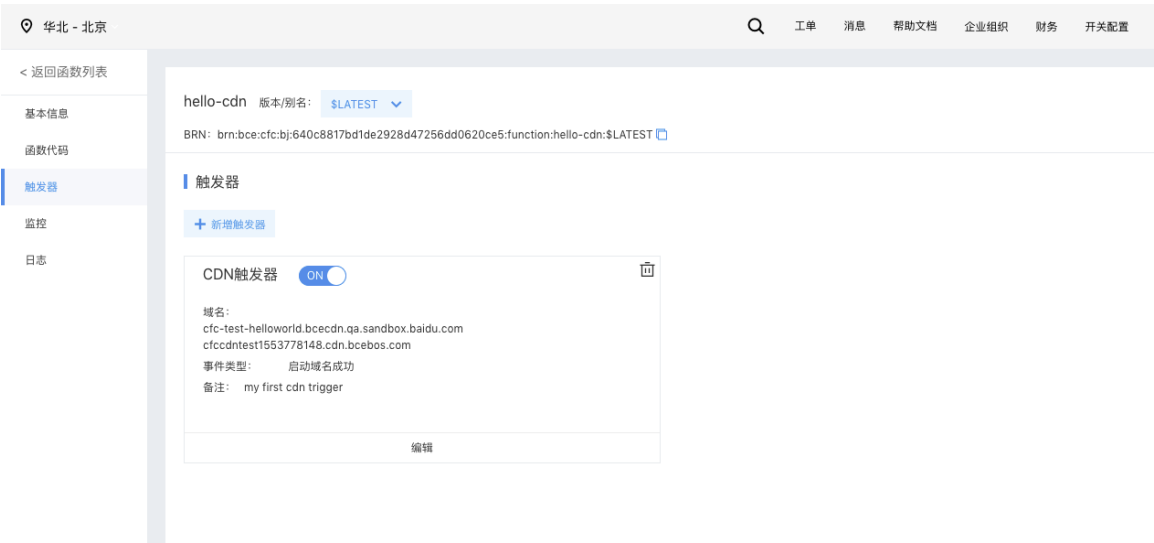
- 4. 在弹出框中，点击下拉框“选择事件源进行添加”，选择CDN触发器。



- 5. 之后在弹出框中配置好域名、事件类型和触发器备注，并点击确认，完成触发器的创建。
 - 事件类型：触发器需要匹配的事件类型，可以通过下拉框选择一种类型，这里我们选择“启动域名成功”
 - 域名：CDN服务中管理的域名
 - 备注：触发器的备注信息

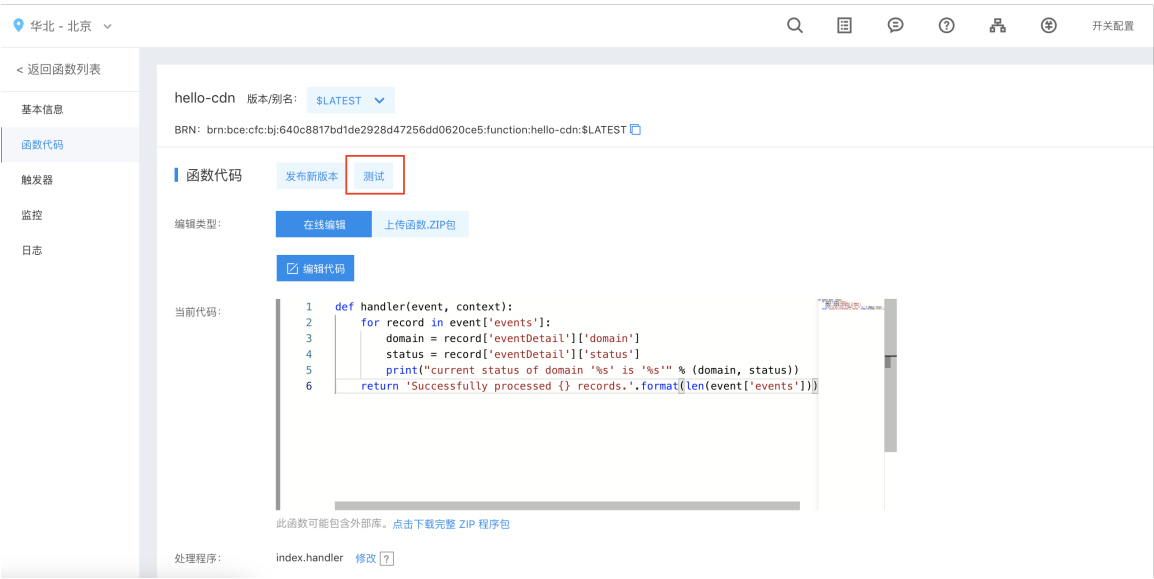


6. 在函数配置页面“触发器”一栏中，可以看到刚刚创建好的CDN触发器及其信息。



测试触发器

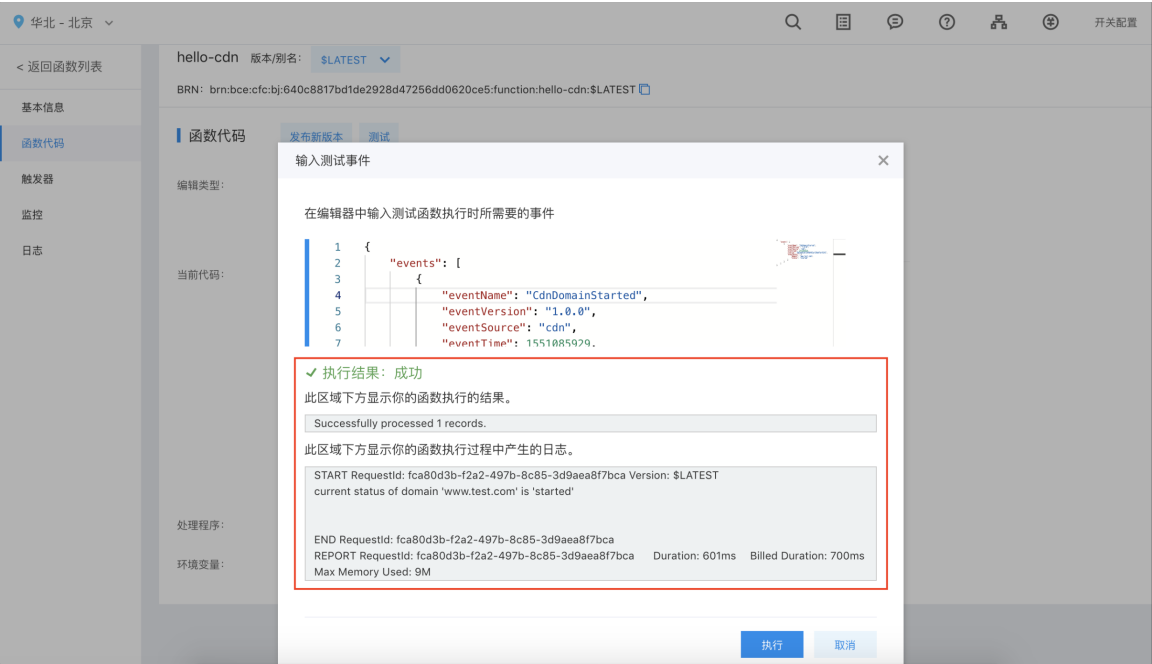
1. 点击测试按钮



2. 输入测试事件，并点击执行，测试内容如下

```
{
  "events": [
    {
      "eventName": "CdnDomainStarted",
      "eventVersion": "1.0.0",
      "eventSource": "cdn",
      "eventTime": 1551085929,
      "userId": "9b2aa05367204d9481ef29bdf5ef8243",
      "eventDetail": {
        "domain": "www.test.com",
        "status": "started"
      }
    }
  ]
}
```

3. 查看返回的结果



CDN事件说明

事件名称	事件说明	事件粒度	事件版本
CachedObjectsBlocked	发现违禁资源	用户粒度	1.0.0
CachedObjectsPushed	预热资源完成	用户粒度	1.0.0
CachedObjectsRefreshed	刷新资源完成	用户粒度	1.0.0
CdnDomainCreated	创建域名成功	用户粒度	1.0.0
CdnDomainDeleted	删除域名成功	用户粒度	1.0.0
LogFileCreated	产生域名日志	域名粒度、用户粒度	1.0.0
CdnDomainStarted	启动域名成功	域名粒度、用户粒度	1.0.0
CdnDomainStopped	停止域名成功	域名粒度、用户粒度	1.0.0

粒度说明

事件的粒度分为用户粒度和域名粒度，其中域名粒度更为精细，用户粒度指代用户全局粒度；针对每个CFC函数触发器的设置，不指定具体域名即为用户粒度触发器，指定具体域名即为域名粒度触发器。

- 用户粒度：用户账户产生指定事件，即可触发相关CFC函数

- 域名粒度：特定域名产生指定事件，即可触发相关CFC函数

🔗 事件结构

预热刷新事件 CachedObjectsPushed：用户账户下预热任务完成 CachedObjectsRefreshed：用户账户下刷新任务完成

```
{
  "events": [
    {
      "eventName": "CachedObjectsPushed", // 事件类型
      "eventVersion": "1.0.0", // 事件版本
      "eventSource": "cdn", // 事件源
      "eventTime": 1551085930, // 时间戳
      "userId": "9b2aa05367204d9481ef29bdf5ef8243", // 用户账户ID
      "eventDetail": {
        "taskId": "eJwzMTU3NjQ2MXBIM0tMTTJKNtMkIIMkwWsEo0AAbMLeg==", // 任务ID
        "status": "completed" // 刷新或预热任务完成
      }
    }
  ]
}
```

黄反事件

CachedObjectsBlocked：用户账户下发现黄反URL

```
{
  "events": [
    {
      "eventName": "CachedObjectsBlocked", // 事件类型
      "eventVersion": "1.0.0", // 事件版本
      "eventSource": "cdn", // 事件源
      "eventTime": 1551085930, // 时间戳
      "userId": "9b2aa05367204d9481ef29bdf5ef8243", // 用户账户ID
      "eventDetail": {
        "url": "http://www.test.com/sexy.jpg", // 黄反URL
        "status": "slightPorn" // 取值为slightPorn或severePorn
      }
    }
  ]
}
```

日志事件 LogFileCreated：用户账户下某一域名或特定域名产生最近时间访问日志

```
{
  "events": [
    {
      "eventName": "CachedObjectsBlocked", // 事件类型
      "eventVersion": "1.0.0", // 事件版本
      "eventSource": "cdn", // 事件源
      "eventTime": 1551085930, // 时间戳
      "userId": "9b2aa05367204d9481ef29bdf5ef8243", // 用户账户ID
      "eventDetail": {
        "domain": "www.test.com", // 产生日志的域名
        "logTime": "2019-02-26T12:00:00" // 日志产生时间
      }
    }
  ]
}
```

域名事件 CdnDomainStarted：用户账户下某一域名或特定域名启动成功 CdnDomainStopped：用户账户下某一域名或特定域名停止成功 CdnDomainCreated：用户账户下某一域名创建成功 CdnDomainDeleted：用户账户下某一域名删除成功

```
{
  "events": [
    {
      "eventName": "CdnDomainStarted", // 事件类型
      "eventVersion": "1.0.0", // 事件版本
      "eventSource": "cdn", // 事件源
      "eventTime": 1551085930, // 时间戳
      "userId": "9b2aa05367204d9481ef29bdf5ef8243", // 用户账户ID
      "eventDetail": {
        "domain": "www.test.com", // 触发动作的域名
        "status": "started" // 取值为started、stopped、created、deleted
      }
    }
  ]
}
```

Rocketmq触发器使用示例

概述

CFC 支持配置 Rocketmq 触发器，配置后可以订阅指定的 Rocketmq topic 进行消息消费并配合函数进行消息处理。以下是创建 Rocketmq 触发器的一般步骤。

创建 Rocketmq 触发器

前提

1. 创建 Rocketmq 集群，参考：<https://cloud.baidu.com/doc/RocketMQ/s/rm4jbrmf4>
2. 创建主题（后续 Rocketmq 触发器需要订阅）
3. 创建消费组（最好为 CFC 单独创建一个消费组，避免和业务消费组混用）
4. 创建用户，如果在 Rocketmq 集群中设置了认证方式为『SASL/PLAIN身份认证』，必须要创建具体的用户并设置好用户的密码以及订阅权限，Rocketmq 触发器中需要使用该用户信息进行订阅处理。

创建函数

用户可以为新建的函数或已有函数配置 Rocketmq 触发器，创建函数的流程可以具体参考[从头创建函数](#)，这里不再赘述。

编写处理函数

1. 登录管理控制台，选择“产品服务>云函数计算 CFC”，进入“函数列表”页面
2. 在“函数列表”页面。选择具体的函数，进入函数详情页面。

在函数详情页中编写百度消息服务触发器对应的 handler，以对触发操作返回适当的响应，之后点击右下角保存按钮完成函数的修改操作。

```
#### -*- coding: utf-8 -*-
import base64
import json

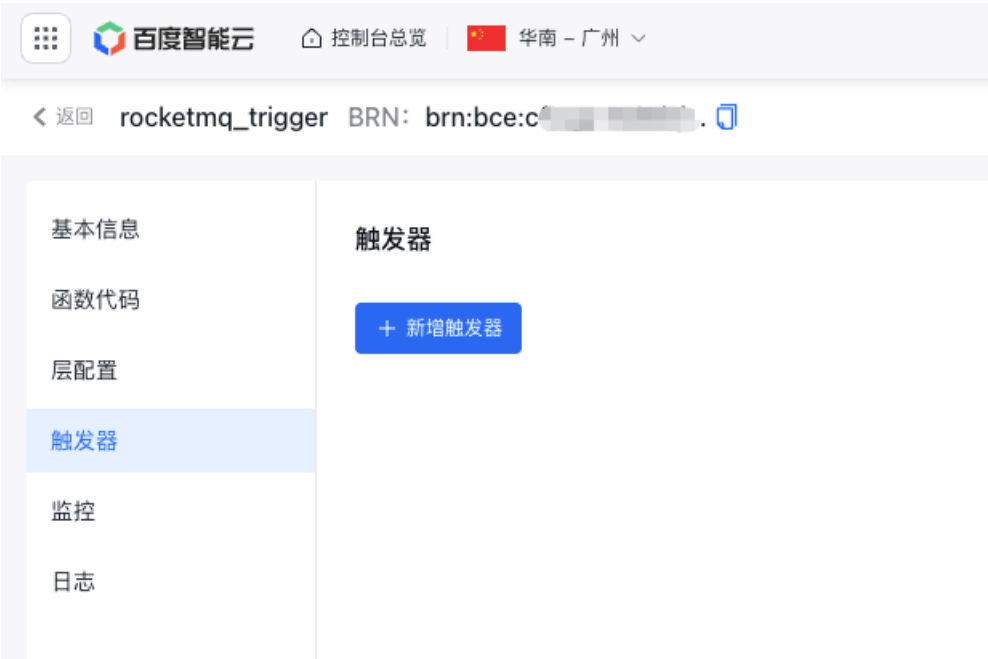
def handler(event, context):
    for record in event['Records']:
        # kafka value is base64 encoded so decode here
        payload = base64.b64decode(record['Rocketmq']['Value'])
        print("Decoded payload: " + payload.decode("utf-8"))
    return 'Successfully processed {} records.'.format(len(event['Records']))
```

注：Rocketmq 触发器的event消息体

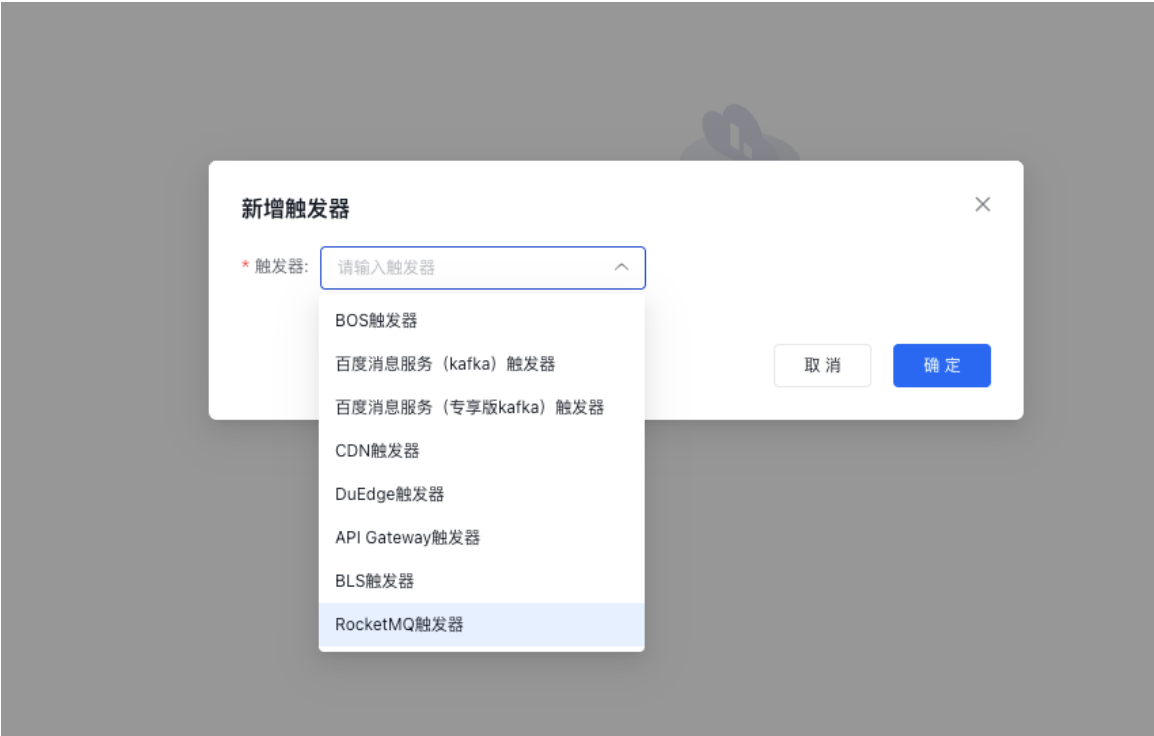
```
{
  "Records": [
    {
      "EventId": "c7ac82ae14ef42d1a4ffa3b2eacea17f__test_only_sl-0-43110",
      "Rocketmq": {
        "Key": "", //base64后的key
        "Value": "MA==", //base64后的value
        "Topic": "c7ac82ae14ef42d1a4ffa3b2eacea17f__test_only_sl",
        "Partition": 0,
        "Offset": 43110,
        "Timestamp": 1553151704.6529999
      },
      "EventName": "bce:rocketmq:record",
      "EventSourceBrn": "c7ac82ae14ef42d1a4ffa3b2eacea17f__test_only_sl",
      "EventSource": "bce:rocketmq"
    },
    {
      "EventId": "c7ac82ae14ef42d1a4ffa3b2eacea17f__test_only_sl-0-43111",
      "Rocketmq": {
        "Key": "",
        "Value": "MQ==",
        "Topic": "c7ac82ae14ef42d1a4ffa3b2eacea17f__test_only_sl",
        "Partition": 0,
        "Offset": 43111,
        "Timestamp": 1553151704.6529999
      },
      "EventName": "bce:rocketmq:record",
      "EventSourceBrn": "c7ac82ae14ef42d1a4ffa3b2eacea17f__test_only_sl",
      "EventSource": "bce:rocketmq"
    }
  ]
}
```

配置 Rocketmq 触发器

1. 登录管理控制台，选择“产品服务> 函数计算 CFC”，进入“函数列表”页面
2. 点击需要添加 Rocketmq 触发器的函数，进入函数详情页面。
3. 点击左侧导航栏中的“触发器”，进入函数配置页面。



- 4. 在函数配置页面中最下方点击“新增触发器”。
- 5. 在弹出框中，点击下拉框“选择事件源进行添加”，选择 Rocketmq 触发器。（目前只支持北京和广州地域）



- 6. 之后在弹出框中配置好选项，并点击确认，完成触发器的创建。
- 集群选择要监听的 Rocketmq 集群
- Topic 选择将要监听的 Rocketmq 服务 Topic
- 消费组选择监听消息使用的消费组
- 认证模式根据 Rocketmq 的认证模式进行
 - 如果Rocketmq 没有配置认证模式，这里选择『无认证』即可
 - 如果Rocketmq 设置了认证方式为『SASL/PLAIN身份认证』，则在触发器中也要选择『ACL 认证』，明天写Rocketmq 用户名和密码（需要个 Rocketmq 中的用户名和密码对应）

新增触发器

* 触发器:

RocketMQ触发器

* RocketMQ集群:

选择RocketMQ集群

* Topic:

选择消息主题

* GroupID:

消费组ID

* 认证模式:

ACL认证

选择认证方式

* 用户名:

请输入用户名

RocketMQ用户名

* 密码:

请输入密码

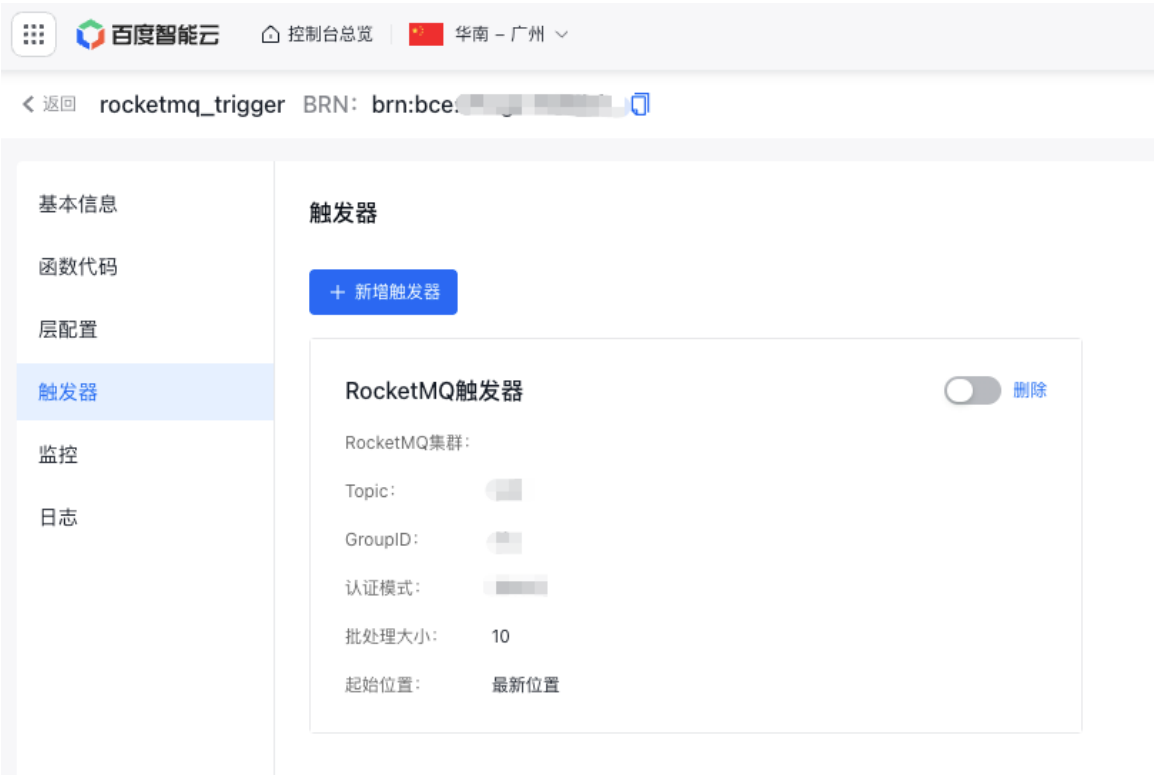
RocketMQ密码

- 批处理大小：从 Topic 中一次读取的最大记录数，范围：1-32
- 起始位置：在 Topic 中开始读取的位置，最新记录对应 Rocketmq 的 OffsetNewest，最老记录对应 Rocketmq 的 OffsetOldest
- 启用触发器：是否直接启动触发器，建议先关闭触发器以便测试

特别注意：

1. Rocketmq 触发器配置的消费组尽量独立，避免与业务消费组重复使用。
2. Rocketmq 一个消费组只能订阅一个 topic，如果同时订阅多个 topic，会出现部分 topic 无法消费的情况，导致出错。所以 CFC 侧对此进行了限制，单一 Rocketmq 集群同一个消费组单地域保持唯一，只可以在一个函数中配置一次，建议使用不同的消费组来订阅不同的 topic。

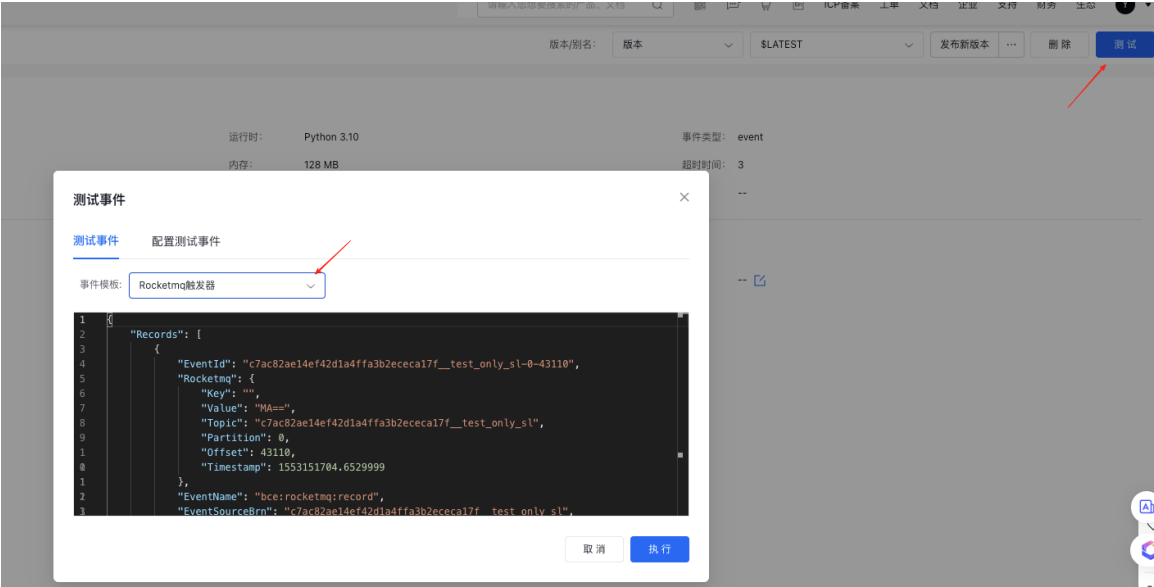
创建完成后，可以在触发器中看到创建好的 Rocketmq 触发器信息。



🔗 测试触发器

🔗 模拟测试

- 1. 点击测试按钮
- 2. 事件模板选择 Rocketmq 触发器

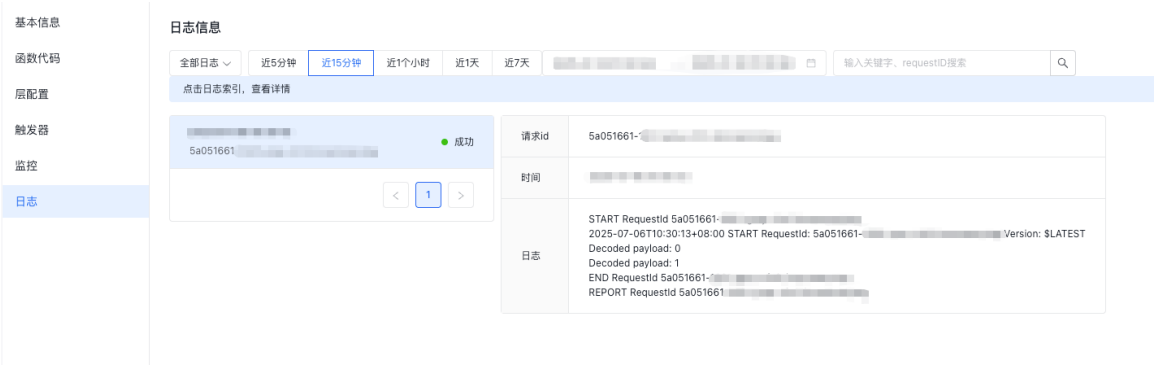


- 3. 点击执行进行测试，可以看到执行结果



真实测试

- 1. 启动 Rocketmq 触发器
- 2. 设置日志存储
- 3. 向 Rocketmq 触发器中发送消息，在日志中可以看到执行结果



工作流

概述

函数计算工作流（XFlow）是一种 Serverless 编排服务，可让您结合 CFC 函数和其它百度云服务，按照您定义的时序逻辑来构建应用。在 XFlow 的图形化控制台，您可以看到应用由一系列事件驱动的步骤组成。

工作流的定义基于状态机和节点，每个节点内设定自身的执行逻辑和下一步要执行的节点，所有节点串联起来形成一个状态机。XFlow 会根据定义逐个执行节点，每个节点定义自己的处理逻辑，以及错误重试和捕获，以避免各类异常导致的失败。可以把多个云函数定义到一个工作流中的各个节点，设定执行顺序、输入输出数据处理等，从而实现各类业务场景。

XFlow 把开发和运行业务流程所需要的任务协调、状态管理和错误处理等工作剥离出来集中管理，使构建应用更为简单高效。目前 XFlow 只支持执行 CFC 函数，在未来将与更多的百度云产品实现打通。

应用场景

工作流主要用来协调分布式应用以构建复杂的、多步骤的、有状态的（Stateful）和长时间运行的业务流程。

ETL数据处理流水线

一些数据处理系统中，常常需要周期性/计划性地处理庞大的数据量，使用工作流可以快速搭建一个高可用的数据处理流水线。例如，专注L4级别无人驾驶技术研发的企业在进行无人驾驶技术研发依赖大量的路测试验数据积累，高效进行路测、快速对路测数据进行处理来指导模型的更新迭代是这类场景的核心诉求。路测过程中会有大量数据产生，而对这些数据的处理流程则复杂多变，即使对于同一批数据，不同的业务小组也会有不同的使用及处理方式。使用工作流可以有效管理不同的数据处理流程，降低人为介入频率，降低成本。

多媒体文件处理

工作流同样适用多任务编排，比如多媒体文件如转码、截帧、人脸识别、语音识别、审核上传等各类处理。用户可以编排成一个完整的处理流程，通过函数计算提交媒体处理任务（或自建处理云函数），产出一个符合业务需求的输出。各任务的执行错误异常可以被可靠的重试，大大提升多媒体任务处理吞吐量。

自动化运维

自动化运维常见有步骤繁琐、时间长短不一、单机脚本可靠性低、依赖复杂、人工效率低等常见的挑战，另外进度无法可视化。工作流可以很好的解决这类问题。例如自动化的软件部署从构建 Docker 容器、上传容器镜像，开始并追踪各节点下拉镜像并启动新版本镜像的容器，每一步云函数产生的日志会被保存到日志服务可供查询分享。基于工作流的自动化运维比单机运维脚本具有高可用、自带的错误处理机制和进度图形化的特点。

机器学习模型训练

机器学习中常常要对模型进行训练，模型训练一般过程为：首先读取原始训练数据；对原始数据进行数据处理如数据清洗等；对数据进行转换；提交数据到机器学习平台进行模型训练；模型训练完毕后得到模型参数数据；应用模型进行预测测试模型效果。整个模型训练的过程可以使用工作流来完成任务编排。

基本概念

工作流（Workflow）

协调多个分布式任务（如云函数）的云服务。通过工作流，您可以用顺序、分支、并行等方式来编排分布式任务，以确保流程按照设定好的顺序可靠地协调任务执行。

流程（Flow）

定义了业务逻辑描述以及流程执行所需要的通用信息。

流程定义语言（Flow Definition Language，简称FDL）

流程定义语言是一种结构化的语言，用来描述和定义业务逻辑，用户通过流程定义语言定义流程（Flow），在执行流程时，工作流服务会根据流程定义依次执行相关节点。

节点（State）

节点是工作流的一个基本单元，表示流程步骤。每个节点接收输入数据，对数据进行操作处理后，将输出数据传递给向下一个节点。节点可以是简单的原子节点，如 pass 节点、operation 节点；也可以是复杂的逻辑控制节点，如 switch 节点、parallel

节点。节点的组合使用构建了复杂的业务逻辑。

🔗 等待节点 (wait)

节点类型之一，用来暂停执行流程，然后再继续执行。

🔗 传递节点 (pass)

节点类型之一，用来输出常量或者将输入转换成期望的输出，通常用于构造和调试工作流，是占位符规划流程基本结构。

🔗 任务节点 (operation)

节点类型之一，用于定义工作流调用的云服务资源信息，执行 operation 节点会调用相应的资源，如调用函数计算服务的函数。

🔗 循环节点 (foreach)

节点类型之一，用于定义并行执行多个相同的分支，功能等同于foreach循环。

🔗 并行节点 (parallel)

节点类型之一，用于定义并行执行多个不同分支。

🔗 选择节点 (switch)

节点类型之一，用于定义不同执行路径，让工作流根据条件执行不同分支。

🔗 成功节点 (succeed)

节点类型之一，用于定义提前结束一系列串行的步骤。成功节点通常和选择节点结合使用，在选择节点条件满足的情况下跳转到一个成功节点，从而不再执行其他节点。

🔗 失败节点 (fail)

节点类型之一，用于定义提前结束一系列串行的节点。当流程执行完失败节点后，定义在失败节点之后的节点不会被继续执行。

快速入门

🔗 创建工作流

本示例演示从头创建工作流，有关工作流的概念请参考[工作流基本概念](#)

1.在工作流管理页面，点击“创建工作流”，完成填写新建工作流的信息和创建。

基本信息

工作流名称：

testWorkflow

只能由数字、字母、-或_组成，必须以字母或_开头，长度小于128个字符

描述：

my first Workflow

工作流定义

1
2 name: demo
3 start: initData
4 states:
5 - type: pass
6 name: initData
7 data:
8 | hello: world
9 next: demoSwitch
10 - type: switch
11 name: demoSwitch
12 description: "using jq expression to decide whether to exec demo"
13 conditions:
14 - condition: .execDemo == true
15 | next: startExec
16 - condition: .execDemo == false
17 | next: stop
18 default:
19 | next: startExec
20 - type: parallel

↺

+

−

↓

Start

InitData

demo@switch

switch

startExec

- 工作流名称：您可以指定自己的工作流，只能由数字、字母、-或_组成，必须以字母或_开头，长度小于64个字符。
- 描述：您可以填写对创建工作流的详细描述。
- 工作流定义：参考[流程定义语言](#)来定义您的工作流，右边是您定义的工作流的流水线。

1
2 name: demo
3 start: initData
4 states:
5 - type: pass
6 name: initData
7 data:
8 | hello: world
9 next: demoSwitch
10 - type: switch
11 name: demoSwitch
12 description: "using jq expression to decide whether to exec demo"
13 conditions:
14 - condition: .execDemo == true
15 | next: startExec
16 - condition: .execDemo == false
17 | next: stop
18 default:
19 | next: startExec
20 - type: parallel
21 name: startExec
22 next: succ
23 branches:
24 - start: wait3s
25 states:
26 - type: wait
27 name: wait3s
28 seconds: 3
29 end: true
30 - start: invokeCfc
31 states:
32 - type: operation
33 name: invokeCfc
34 resource: "brn:bce:cfc:bj:7c54a6e14823a11c:function:test:\$LATEST"
35 end: true
36 catch:
37 - errors:
38 | - XFlow.ALL
39 | next: errorHandler
40 - type: pass

↺

+

−

↓

Start

InitData

demo@switch

switch

startExec

parallel

wait3s

invokeCfc

errorHandler

succ

stop

End

点击提交，完成工作流创建过程。

控制台总览 华北 - 北京 创建工作流

工作流

+ 创建工作流

工作流名称	创建时间	修改时间	备注	操作
testWorkflow	2021-12-27 11:07:00	2021-12-27 11:07:00	my first Workflow	编辑 删除

每页展示 10 <

2.进入工作流信息页面，可查看工作流执行历史和定义，您也可以在"定义"中重新修改您的工作流。

174

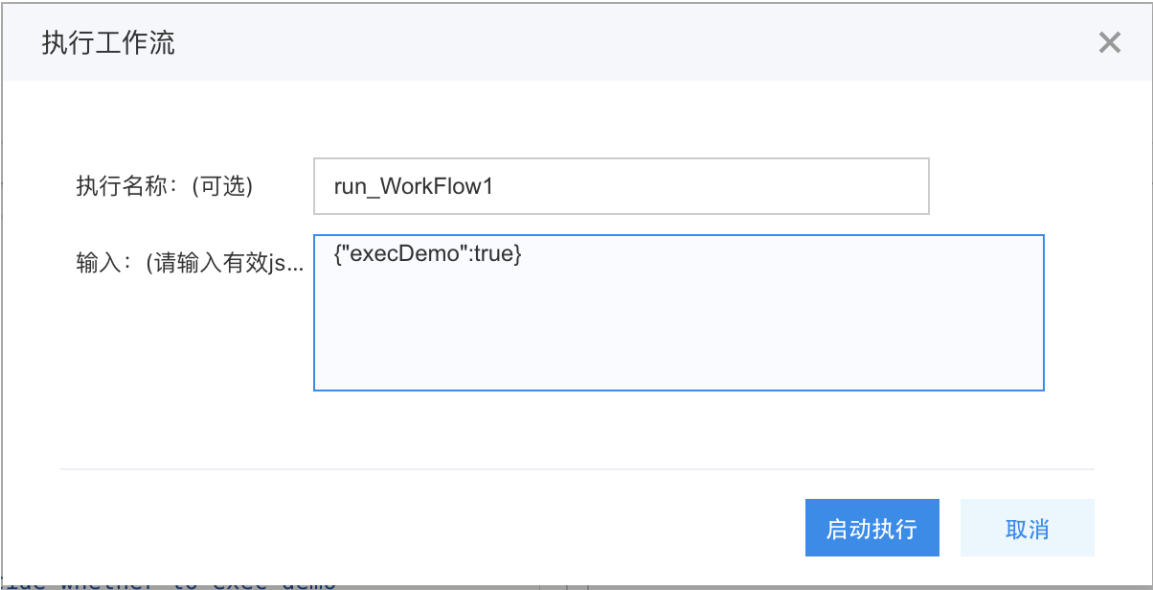


工作流的执行

开始执行

1. 点击“开始执行”。
2. 输入执行名称。
3. 点击“启动运行”。

输入必须是JSON格式字符串，本例中输入：“{“execDemo”:true}”。



查看结果

执行成功后在执行历史可以查看本次执行的状态。



- **执行结果**：执行状态显示为Succeed，执行时间单位为ms。我们可以点击**执行名称**，查看执行的输入和输出。

执行历史记录 输入/输出

输入

```
{ "execDemo": true }
```

输出

```
{ "cause": "The resource specified in the request does not exist", "code": "CFC.ResourceNotFoundException", "execDemo": true, "hello": "world" }
```

查看历史

工作流记录了详细的执行历史记录，我们可以选择某一步骤，查看该步骤详细的输入、执行结果、耗时。

ID	类型	步骤	时间	相对时间 (毫秒)
收起 1	ExecutionStarted		2021-12-27 11:26:41	6
收起 2	StatePassEntered	initData	2021-12-27 11:26:45	4475
<pre>{ "input": { "execDemo": true } }</pre>				
收起 3	StatePassExited	initData	2021-12-27 11:26:50	9955
<pre>{ "output": { "execDemo": true, "hello": "world" } }</pre>				
展开 4	StateSwitchEntered	demoSwitch	2021-12-27 11:26:56	15477
展开 5	StateSwitchExited	demoSwitch	2021-12-27 11:27:02	21049
展开 6	StateParallelEntered	startExec	2021-12-27 11:27:08	27435
展开 7	StateParallelStarted	startExec	2021-12-27 11:27:14	33287
展开 8	StateWaitEntered	wait3s	2021-12-27 11:27:19	38982
展开 9	StateOperationEntered	invokeCfc	2021-12-27 11:27:19	38982

也可以通过点击工作流的流程图标显示具体某个步骤的输入和输出。

工作流 定义

+

-

↓

```
graph TD; Start((Start)) --> initData[initData]; initData --> demoSwitch[demoSwitch]; demoSwitch --> switch{switch}; switch --> startExec[startExec]; startExec --> parallel[parallel]; parallel --> wait3s[wait3s]; parallel --> invokeCfc[invokeCfc]; invokeCfc --> errorHandler[errorHandler]; errorHandler --> succ[succ]; wait3s --> succ; stop[stop] --> succ; succ --> End(( ));
```

```
{ "input": { "execDemo": true }, "output": { "execDemo": true, "hello": "world" } }
```

176

流程定义语言

基本介绍

本文介绍了流程定义语言的基本知识和相关使用示例。

🔗 流程定义语言

流程定义语言是一种基于 `YAML` 的结构化语言，用于定义状态机、节点，包括负责执行的 `operation` 节点，决定下一跳的 `switch` 节点，停止执行并抛出错误的 `fail` 节点等各类节点，这些节点组合起来可以构建复杂的业务逻辑。

下面为一个简单示例，工作流通过输入数据中的 `doExec` 值判断是否进入 `operation` 节点执行 `CFC` 函数。

```
name: demo
start: demoSwitch
version: 1.0.0
timeoutSeconds: 100
description: "XFlow 流程定义语言示例"
states:
  - type: switch
    name: demoSwitch
    description: "whether to exec demo"
    conditions:
      - condition: .doExec == true
        next: invokeFunc
      - condition: .doExec == false
        next: stop
  - type: operation
    name: invokeFunc
    description: "invoke a function"
    resource: "brn:bce:cfc:su:7c54a6e148:function:test:$LATEST"
    end: true
  - type: fail
    name: stop
    cause: "demo stopped"
```

字段	描述
name(必需)	流程定义语言的名字
start(必需)	用于指定流程的开始节点
states(必需)	节点定义的集合
version(可选)	版本信息，需要符合 semver 规则
timeoutSeconds(可选)	工作流的超时时间，单位为秒，最长为 172800，如果流程实际执行超过该时间则会被终止
description(可选)	描述信息

🔗 节点类型

节点是工作流的一个基本单元，所有的节点都定义在工作流的 `states` 参数里。每个节点都有自己的类型，在节点内部使用 `type` 参数来表示。目前工作流支持的节点类型如下：

- 传递节点（pass）
pass 节点可以直接生成常量数据，可用来在初始构建阶段提前占位，或通过修改输出调试工作流。
- 选择节点（switch）
根据节点的输入数据做条件判断，选择执行不同的分支节点。

- 等待节点 (wait)
等待一段时间，不做其它操作。
- 失败节点 (fail)
终止工作流的执行，并把结果执行标记为失败。
- 成功节点 (succeed)
终止工作流的执行，并把结果执行标记为成功，类似编程语言中的 `return 0;`。
- 任务节点 (operation)
负责调用 CFC 函数。
- 并行节点 (parallel)
可在节点内定义多个子分支，并行执行。
- 循环节点 (foreach)
循环处理数组数据，类似编程语言中的 `for` 循环。

🔗 节点拓扑关系

工作流开始运行时，首先进入 `start` 字段对应的节点。在一个节点内部，定义 `next: invoke` 表示节点的下一跳指向为 `invoke` 节点，定义 `end: true` 表示该节点为结束节点，没有下一跳。在节点执行完成后，XFlow 就会根据这些定义来决定接下来的动作。

关于工作流的开始、结束、下一跳指向，在定义时需要遵循如下原则：

- 节点需要明确定义下一跳指向或结束

节点必须定义且只能定义 `next` 或 `end` 其中的一个，流程的最后一个节点需要定义 `end: true`。

`fail/succeed` 节点是一种特殊情况，它们表示工作流的执行已结束，已经内含了 `end` 逻辑，因此不允许再指定 `next` 和 `end` 参数。

- 只能有一个开始节点，可以有多个结束节点

一个工作流内可能会存在多个分支，比如 `parallel` 节点定义了两个同时执行的分支节点，每个分支的最后一个节点，都需要表明自己为结束节点，即定义 `end: true`。

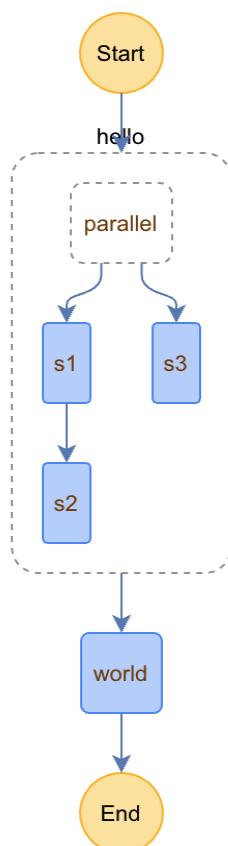
- 节点有层级和分支之分，不同层级、分支的节点是隔离的

以 `parallel` 节点为例，`parallel` 节点内可定义多个节点分支，分支内的节点与 `parallel` 节点是父子关系，并不在一个层级，因此分支内的节点不可以用 `next` 指向上层级的节点，同样上层节点的 `next` 也不可以指向子层级。另外，不同分支的节点之间也是隔离的。

在下面示例中，`s1`、`s2` 节点在 `hello` 节点内部，因此它们的 `next` 不可以设置为外层的 `world` 节点。`s3` 节点与 `s1`、`s2` 虽然在一个节点，但它们属于不同分支，因此 `s3` 也不可以指向 `s1`、`s2`。


```
name: parallel-demo
start: hello
states:
- type: parallel
  name: hello
  branches:
  - start: s1
    states:
    - type: pass
      name: s1
      next: s2
    - type: pass
      name: s2
      end: true
  - start: s3
    states:
    - type: wait
      name: s3
      seconds: 3
      end: true
  next: world
- type: succeed
  name: world
```

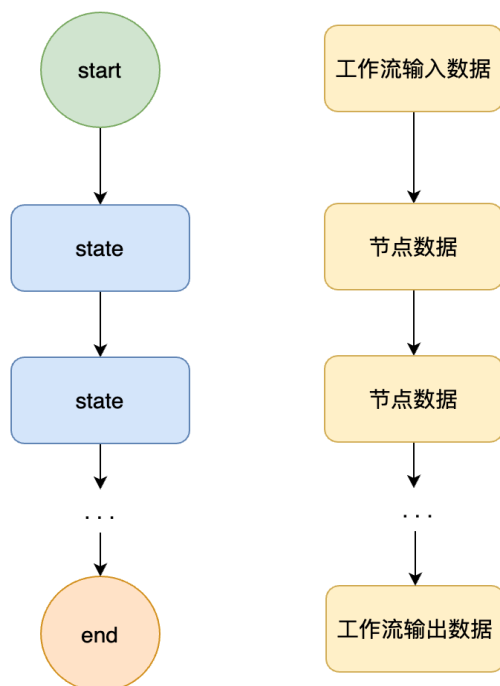
示例图如下所示：



输入和输出

概述

工作流开始运行时，会接收用户指定的输入数据，并将数据传递至工作流的开始节点，即 `start` 所指向的节点。在后续的执行中，每个节点都接收上一个节点的输出作为自己的输入，并在执行结束后，继续将自己的输出传给下一个节点。所有的输入输出数据均为 **JSON 格式**。本节将为您介绍如何处理节点的输入输出数据。



🔗 工作流的输入与输出

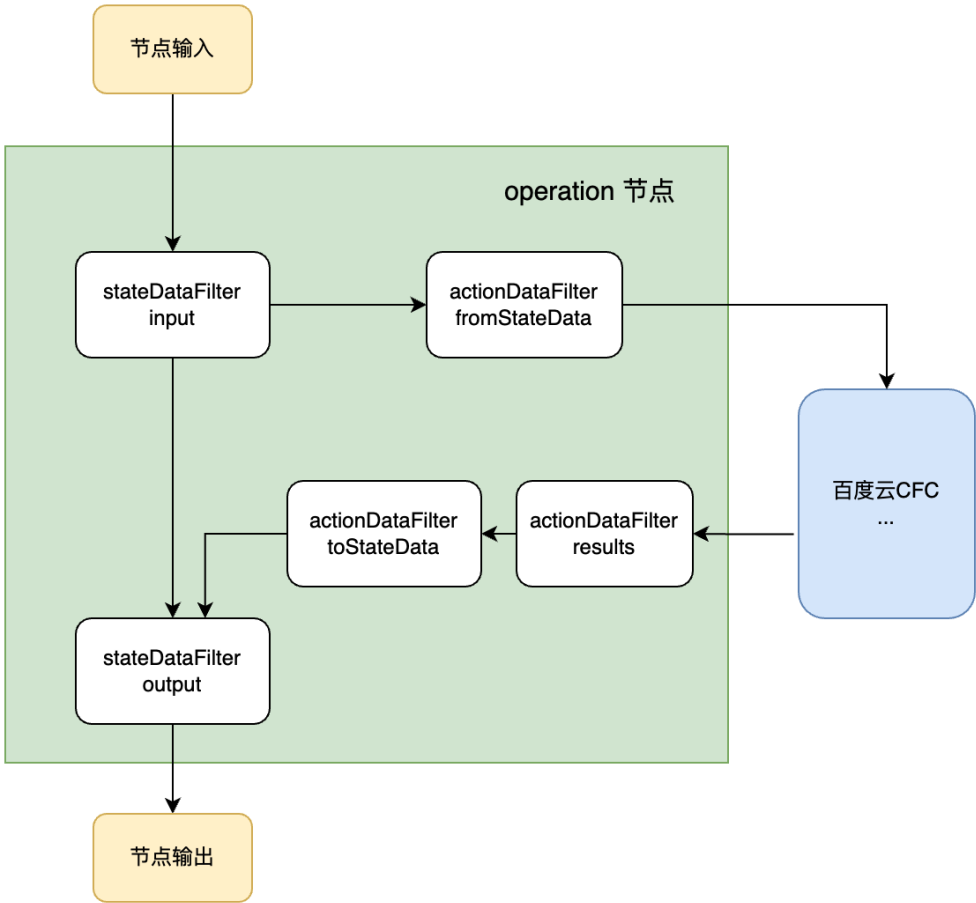
- workflow 的输入由用户提供，如果用户未提供输入，那么使用 "{}" 作为默认输入。
- workflow 的输出是最后一个节点的输出。

🔗 节点的输入与输出

workflow 的输入作为 start 节点的输入，当一个节点执行结束后，它的输出会成为下一个节点的输入。

节点可以通过 `stateDataFilter` 处理上一个节点传入的输入数据，和传给下一个节点的输出数据。operation 节点还可以通过 `actionDataFilter` 定义调用云函数时的输入，和处理云函数的返回结果，比如把节点输入数据的其中一部分作为 payload 去调用云函数，云函数返回结果后只取出其中的一部分放入节点数据里。

`stateDataFilter` 和 `actionDataFilter` 都是节点的可选参数，节点若不定义则不对输入输出做其它处理。



stateDataFilter

参数	类型	描述
input (可选)	string	jq 表达式，在执行 state 前过滤输入
output (可选)	string	jq 表达式，在 state 结束前过滤输出

示例:

工作流的定义如下

```
name: demo
start: hello
description: "从输入数据中取出 apple 对应的值"
states:
- type: wait
  name: hello
  stateDataFilter:
    input: ".fruits"
    output: "{res: .apple}"
  seconds: 2
  end: true
```

假定工作流的输入数据为

```
{
  "fruits": {
    "apple": 7,
    "banana": 10
  },
  "vegetables": ["potato", "tomato", "carrot"]
}
```

工作流的处理流程如下：

- 1、收到执行请求和输入数据，找到开始节点 hello，将数据传递给 hello；
- 2、hello 根据 stateDataFilter 的 input 参数处理输入数据，处理结果作为自己节点数据，结果为

```
{
  "apple": 7,
  "banama" : 10
}
```

- 3、hello 节点执行 wait 逻辑，等待2秒钟；
- 4、hello 节点操作结束，根据 stateDataFilter 的 output 参数处理输出数据，结果为

```
{
  "res": 7
}
```

- 5、由于 hello 节点之后没有其它节点，因次 hello 执行结束后整个工作流就结束了，工作流的输出即上一步中的结果。

actionDataFilter 在 operation 节点中调用函数计算等公有云服务时，可以通过这个 filter 定义传给云服务的参数，也能对服务返回的结果进一步提取筛选，并加入到节点数据中。

参数	类型	描述
fromStateData（可选）	string	jq 表达式，从节点数据中过滤部分数据传给云服务
results（可选）	string	jq 表达式，对调用云服务返回的结果进一步过滤，筛选出想要的结果
toStateData（可选）	string	jq 表达式，把 results 的结果添加/合并到 state data 的某个字段内，若未指定则放入节点数据的顶层
useResults（可选）	bool	是否把云服务返回的结果放入节点数据内，默认为 true

示例：

```
name: demo
start: hello
description: "从输入数据中取出 apple 对应的值"
states:
- type: operation
  name: hello
  actionDataFilter:
    fromStateData: "{ stateInput: .hello }"
    results: "[ .breads[0], .pasta[1] ]"
    toStateData: ".itemsToBuyAtStore"
  resource: "brn:cfc:bj:123456:function:test:$LATEST"
end: true
```

假定本次工作流的输入为{"hello":"world"}，函数的返回为

```
{
  "breads": ["baguette", "brioche", "rye"],
  "pasta": [ "penne", "spaghetti", "ravioli"]
}
```

那么工作流的处理流程大致如下：

- 1、收到执行请求和输入数据，找到开始节点 hello，将输入数据 {"hello":"world"} 传递给 hello；
- 2、hello 节点内没有定义 stateDataFilter，因此不执行输入预处理；
- 3、hello 节点内定义了 actionDataFilter，根据 fromStateData 参数执行数据处理，得到 {"stateInput": "world"}；
- 4、hello 节点将上一步得到的结果作为 payload 去调用函数；
- 5、函数返回结果后，根据 actionDataFilter 的 results 参数处理函数返回，结果为 ["baguette", "spaghetti"]；
- 6、根据 actionDataFilter 的 toStateData 参数继续处理，把 results 的结果放到节点数据的 itemsToBuyAtStore 字段（没有则创建），最终节点数据为

```
{
  "hello": "world",
  "itemsToBuyAtStore": ["baguette", "spaghetti"]
}
```

- 7、hello 节点内没有定义 stateDataFilter，因此不执行输出预处理，工作流执行结束。

jq 表达式

从上文的介绍中可知，节点数据的过滤、处理都依靠 jq 表达式，这里展示出常用的 jq 操作。源数据如下：

```
{
  "store": {
    "book": [
      {
        "category": "reference",
        "author": "Nigel Rees",
        "title": "Sayings of the Century",
        "price": 8.95
      },
      {
        "category": "fiction",
        "author": "Evelyn Waugh",
        "title": "Sword of Honour",
        "price": 12.99
      },
      {
        "category": "fiction",
        "author": "Herman Melville",
        "title": "Moby Dick",
        "isbn": "0-553-21311-3",
        "price": 8.99
      },
      {
        "category": "fiction",
        "author": "J. R. R. Tolkien",
        "title": "The Lord of the Rings",
        "isbn": "0-395-19395-8",
        "price": 22.99
      }
    ],
    "bicycle": {
      "color": "red",
      "price": 19.95
    }
  }
}
```

各类操作结果如下：

表达式	结果
.store	store 内的所有数据
.store.bicycle.price	store 内 bicycle 的价格
{color: .store.bicycle.color}	{"color": "red"}
.store.book[2]	store 内的第3个book对象
.store.book[0:2]	store 内的前两本书
.store.book[] select(.price < 10)	store 内所有价格小于10的book
.store.book[].author	store 内的所有书的作者

jq 是一个功能强大的、用于解析 json 数据的工具，关于 jq 的更多用法，您可参考

[jq 官网](#)

[jqplay \(jq在线调试\)](#)

传递（pass）节点

概述

pass 节点用于数据的生成和传递，可用来在初始构建阶段提前占位，或通过修改输出调试工作流。

参数

以下为 pass 节点所包含的参数字段：

字段	类型	描述
type（必需）	string	节点类型，值为 "pass"
name（必需）	string	节点名称
next（二选一）	string	当前节点运行结束后，下一个要运行的节点
end（二选一）	bool	设定 end 参数为 true，表示该节点运行结束后，没有继续要运行的节点。只可以定义 next 或 end 中的一个
data（可选）	object	要注入到节点数据的 JSON 对象
description（可选）	string	节点描述信息
stateDataFilter（可选）	object	节点输入输出过滤

示例

示例工作流定义如下：

```
name: demo
start: simple_pass
states:
- type: pass
  name: simple_pass
  data:
    person:
      fname: John
      lname: Doe
      address: 1234 SomeStreet
      age: 40
  end: true
```

假定该工作流的输入数据为 {"hello": "world"}，那么 pass 节点的输出为

```
{
  "hello": "world",
  "person": {
    "fname": "John",
    "lname": "Doe",
    "address": "1234 SomeStreet",
    "age": 40
  }
}
```

等待（wait）节点

概述

wait 节点的作用是等待一段时间，可用于节点间并发执行的任务协调。您可以指定一个相对时间，也可以通过时间戳指定一个

绝对结束时间。

参数

以下为 pass 节点所包含的参数字段：

字段	类型	描述
type（必需）	string	节点类型，值为 "wait"
name（必需）	string	节点名称
next（二选一）	string	当前节点运行结束后，下一个要运行的节点
end（二选一）	bool	设定 end 参数为 true，表示该节点运行结束后，没有继续要运行的节点。只可以定义 next 或 end 中的一个
description（可选）	string	节点描述信息
stateDataFilter（可选）	object	节点输入输出过滤
seconds（二选一）	integer	相对等待时间，单位为秒。只可以定义 seconds 或 timestamp 中的一个
timestamp（二选一）	string	RFC3339 格式时间戳，表示等待至该绝对时间

示例

示例工作流定义如下：

- 等待 10 秒

```
name: demo
start: wait10s
states:
- type: wait
  name: wait10s
  seconds: 10
  end: true
```

- 等待至某个绝对时间点

```
name: demo
start: waitForReady
states:
- type: wait
  name: waitForReady
  timestamp: 2022-01-01T01:01:00Z
  end: true
```

- 等待的相对时间值从输入数据中获取

```
name: demo
start: waitForReady
states:
- type: wait
  name: waitForReady
  seconds: ".wait_seconds"
  end: true
```

判断（switch）节点

概述

switch 节点会根据不同条件选择不同的下一个节点，类似于编程语言中的 switch-case。工作流服务会按 switch 节点内 condition 的定义顺序依次匹配，若无匹配到且定义了 default 参数，则会根据 default 参数绝对下一个要执行的节点。

参数

以下为 pass 节点所包含的参数字段：

字段	类型	描述
type（必需）	string	节点类型，值为 "switch"
name（必需）	string	节点名称
conditions（必需）	array of condition	条件匹配数组，数组元素见下表 condition
default（可选）	object	conditions 未匹配到时，使用此字段内指定的节点，object 内只有一对 kv，即 "next: stateName"
description（可选）	string	节点描述信息
stateDataFilter（可选）	object	节点输入输出过滤

条件匹配元素 condition 定义如下：

字段	类型	描述
condition（必需）	string	jq 表达式，结果必须为布尔值
next（必需）	string	下一个节点名称

注：switch 节点不允许额外指定 next 或 end 参数。

示例

示例工作流定义如下，读取节点输入数据中的 .ready 字段值，根据值的不同去执行 fail 或 succeed 节点。

```
name: demo
start: checkReady
states:
  - type: switch
    name: checkReady
    conditions:
      - condition: .ready == "yes"
        next: handleReady
      - condition: .ready == "no"
        next: handleNotReady
    default:
      next: handleReady
  - type: fail
    name: handleNotReady
  - type: succeed
    name: handleReady
```

任务（opreation）节点

概述

operation 节点内定义了 CFC 函数的 BRN，在执行时会调用 CFC 函数，目前只支持同步调用。

参数

以下为 operation 节点所包含的参数字段：

字段	类型	描述
type（必需）	string	节点类型，值为 "operation"
name（必需）	string	节点名称
resource（必需）	string	CFC 函数完整 BRN
next（二选一）	string	当前节点运行结束后，下一个要运行的节点
end（二选一）	bool	设定 end 参数为 true，表示该节点运行结束后，没有继续要运行的节点。只可以定义 next 或 end 中的一个
description（可选）	string	节点描述信息
stateDataFilter（可选）	object	节点输入输出过滤，参考 输入与输出
actionDataFilter（可选）	object	对函数的输入输出过滤，参考 输入与输出
retry（可选）	object	错误重试定义，参考 错误重试
catch（可选）	object	错误捕获定义，参考 错误捕获

示例

示例工作流定义如下，operation 节点内定义了错误重试与捕获。

```
name: demo
start: invokeFunc
states:
  - type: operation
    name: invokeFunc
    resource: "brn:bce:cfc:bj:123456:function:test:$LATEST"
    retry:
      - errors:
          - CFC.ServiceException
        intervalSeconds: 10
        maxAttempts: 2
        multiplier: 2
    catch:
      - errors:
          - CFC.ServiceException
        next: handleFail
    end: true
  - type: pass
    name: handleFail
    end: true
```

成功（succeed）节点

概述

succeed 节点用于标记工作流已执行成功，并结束整个工作流的执行。它通常会跟 switch 节点结合使用，在 switch 节点判断某个条件已满足的情况下跳转到 succeed 节点，结束执行。

参数

以下为 succeed 节点所包含的参数字段：

字段	类型	描述
type（必需）	string	节点类型，值为 "succeed"
name（必需）	string	节点名称
description（可选）	string	节点描述信息

succeed 节点已内含了结束执行，因此不允许再指定 next 或 end 参数。

示例

示例工作流定义如下，读取节点输入数据中的 .ready 字段值，读取 ".ready" 的值，若为 "yes" 则跳到成功节点、结束执行，若为 "no" 则执行其它节点继续等待10秒钟。

```
name: demo
start: checkReady
states:
- type: switch
  name: checkReady
  conditions:
  - condition: .ready == "yes"
    next: handleReady
  - condition: .ready == "no"
    next: handleNotReady
  default:
    next: handleReady
- type: wait
  name: handleNotReady
  seconds: 10
  end: true
- type: succeed
  name: handleReady
```

失败（fail）节点

概述

fail 节点用于标记工作流已执行失败，并结束整个工作流的执行。它通常会跟 switch 节点结合使用，在 switch 节点判断某个条件已满足的情况下跳转到 fail 节点，结束执行。fail 节点内可以自定义错误码和错误信息。

参数

以下为 fail 节点所包含的参数字段：

字段	类型	描述
type（必需）	string	节点类型，值为 "fail"
name（必需）	string	节点名称
error（可选）	string	错误状态码
cause（可选）	string	错误信息
description（可选）	string	节点描述信息

fail 节点已内含了结束执行，因此不允许再指定 next 或 end 参数。

示例

示例工作流定义如下，读取节点输入数据中的 .ready 字段值，读取 ".ready" 的值，若为 "no" 则跳转到失败节点、结束执行。

```
name: demo
start: checkReady
states:
- type: switch
  name: checkReady
  conditions:
  - condition: .ready == "yes"
    next: handleReady
  - condition: .ready == "no"
    next: handleNotReady
  default:
    next: handleReady
- type: fail
  name: handleNotReady
  error: Custom.NotReady
  cause: "it's not ready yet"
- type: succeed
  name: handleReady
```

并行（parallel）节点

概述

parallel 节点用来并行执行多个节点，它定义了多个分支，每个分支内包含一系列串行的节点。

parallel 会把自己经过 stateDataFilter 过滤后的输入数据，拷贝给每个分支的开始节点，作为它们的输入数据。在所有分支执行结束后，parallel 再把分支的输出汇总，放入自己的节点数据内并输出。若其中一个分支执行失败，那么 parallel 节点也会失败。

参数

以下为 parallel 节点所包含的参数字段：

字段	类型	描述
type（必需）	string	节点类型，值为 "parallel"
name（必需）	string	节点名称
branches（必需）	array of branch	分支数组
next（二选一）	string	当前节点运行结束后，下一个要运行的节点
end（二选一）	bool	设定 end 参数为 true，表示该节点运行结束后，没有继续要运行的节点。只可以定义 next 或 end 中的一个
description（可选）	string	节点描述信息
stateDataFilter（可选）	object	节点输入输出过滤，参考 输入与输出
retry（可选）	object	错误重试定义，参考 错误重试
catch（可选）	object	错误捕获定义，参考 错误捕获

分支 branch 的定义：

字段	类型	描述
start（必需）	string	分支的开始节点
states（必需）	array of state	该分支的节点数组

可以看到，分支的定义其实就是一个简化版的完整工作流的定义。

示例

示例工作流定义如下，parallel 节点定义了两个分支：

```
name: demo
start: hello
states:
- type: parallel
  name: hello
  branches:
  - start: func1
    states:
    - type: operation
      name: func1
      resource: "brn:bce:cfc:su:123456:function:test1:$LATEST"
      next: func2
    - type: operation
      name: func2
      resource: "brn:bce:cfc:su:123456:function:test2:$LATEST"
      end: true
  - start: pass1
    states:
    - type: pass
      name: pass1
      data:
        bce: xflow
      end: true
  next: succ
- type: succeed
  name: succ
```

假定工作流的输入是 {"hello":"world"}，那么 parallel 节点及两个分支的输入数据均为 {"hello":"world"}。

假定分支1的输出数据是 {"result":"success"}，分支2从定义可知输出数据是 {"hello":"world", "bce":"xflow"}，

最终 parallel 的输出数据是

```
{
  "hello":"world",
  "bce":"xflow",
  "result":"success"
}
```

循环（foreach）节点

概述

foreach 节点用于遍历数据集，给数据集中的每个元素都执行一次定义好的子流程，类似编程语言中的 for 循环。数据集是一个 JSON 数组对象。

foreach 与其它节点一样，开始执行时仍会先根据 stateDataFilter 过滤输入数据，接下来会根据 inputCollection 参数，从过滤后的输入数据中取出或生成一个数据集。接下来就是从数据集中依次取出元素，结合 iterationParam 参数传给子流程并执行，元素间的执行是并行的。

在所有元素都执行结束后，foreach 把执行结果汇总到一个数组，根据 outputCollection 参数节点数据内并输出。若某个元素的执行失败了，则整个 foreach 节点也会失败。

处理流程可用如下伪码表示：

```
##### stateInput: 节点输入数据
##### stateOutput: 节点输入数据

stateInput = stateDataFilter(stateInput) # 过滤输入

dataSetInput = inputCollection(stateInput) # 生成数据集
dataSetOutput = []

for item in dataSetInput:
    iteratorInput = {"value of iterationParam": item} # 生成传给子流程的数据
    iteratorOutput = iterator(iteratorInput) # 用子流程执行每个元素（并行）
    dataSetOutput.append(iteratorOutput) # 汇总执行结果

if outputCollection is not None: # 如果定义了 outputCollection 参数，则把执行结果添加到节点输出数据
    stateOutput["value of outputCollection"] = dataSetOutput

stateOutput = stateDataFilter(stateOutput) # 过滤输入
```

🔗 参数

以下为 foreach 节点所包含的参数字段：

字段	类型	描述
type（必需）	string	节点类型，值为 "foreach"
name（必需）	string	节点名称
iterator（必需）	object	子流程定义
inputCollection（必需）	string	jq 表达式，用来指定或生成数据集
iterationParam（必需）	string	引用数据集中每个元素的变量名
outputCollection（可选）	string	jq 表达式，用于指定把遍历数据集的返回结果写入节点数据内的某个字段，若不设置则表示忽略返回结果
max（可选）	integer	并行执行元素的最大并发数，默认为100
next（二选一）	string	当前节点运行结束后，下一个要运行的节点
end（二选一）	bool	设定 end 参数为 true，表示该节点运行结束后，没有继续要运行的节点。只可以定义 next 或 end 中的一个
description（可选）	string	节点描述信息
stateDataFilter（可选）	object	节点输入输出过滤，参考 输入与输出
retry（可选）	object	错误重试定义，参考 错误重试
catch（可选）	object	错误捕获定义，参考 错误捕获

iterator 的定义与 parallel 节点中的 branch 类似，如下所示：

字段	类型	描述
start（必需）	string	分支的开始节点
states（必需）	array of state	该分支的节点数组

🔗 示例

示例工作流定义如下，该 foreach 节点的逻辑是，从节点输入数据的 orders 数组中筛选出符合 ".completed == true" 条件的

order，每个 order 传给 iterator 中的函数去执行，最后把数据结果写入节点输出数据的 "confirmationresults" 字段。

```
name: demo
start: processOrder
states:
- name: processOrder
  type: foreach
  inputCollection: "[.orders[] | select(.completed == true)]"
  iterationParam: completedorder
  outputCollection: ".confirmationresults"
  iterator:
    start: sendConfirmation
    states:
    - type: operation
      name: sendConfirmation
      resource: "brn:bce:cfc:bj:123456:function:test:$LATEST"
  end: true
```

假定工作流的输入是：

```
{
  "orders": [
    {
      "orderNumber": "1234",
      "completed": true,
      "email": "firstBuyer@buyer.com"
    },
    {
      "orderNumber": "5678",
      "completed": true,
      "email": "secondBuyer@buyer.com"
    },
    {
      "orderNumber": "9910",
      "completed": false,
      "email": "thirdBuyer@buyer.com"
    }
  ]
}
```

节点开始执行后，通过 `inputCollection` 的 jq 表达式，筛选出 `completed` 为 `true` 的元素，得到数据集：

```
[
  {
    "orderNumber": "1234",
    "completed": true,
    "email": "firstBuyer@buyer.com"
  },
  {
    "orderNumber": "5678",
    "completed": true,
    "email": "secondBuyer@buyer.com"
  }
]
```

接下来对数据集中的每个元素，根据 `iterationParam` 参数，在本例中其值为 `"completedorder"`，构造出传给子流程的数据。

构造出的第一个数据如下所示，然后调用子流程，假如子流程返回结果 `{"1234": "ok"}`。

```
{
  "completedorder": {
    "orderNumber": "1234",
    "completed": true,
    "email": "firstBuyer@buyer.com"
  }
}
```

构造出的第二个数据如下，然后调用子流程，假如子流程返回结果 {"5678": "ok"}。

```
{
  "completedorder": {
    "orderNumber": "5678",
    "completed": true,
    "email": "secondBuyer@buyer.com"
  }
}
```

两次子流程结束后，汇总后得到的结果集为 [{"1234": "ok"}, {"5678": "ok"}]。

本例中定义了 `outputCollection: ".confirmationresults"`，说明需要把结果集放到节点输出数据里的 `"confirmationresults"` 字段里，该字段不存在时会被自动创建，最终节点输出数据为：

```
{
  "orders": [
    {
      "orderNumber": "1234",
      "completed": true,
      "email": "firstBuyer@buyer.com"
    },
    {
      "orderNumber": "5678",
      "completed": true,
      "email": "secondBuyer@buyer.com"
    },
    {
      "orderNumber": "9910",
      "completed": false,
      "email": "thirdBuyer@buyer.com"
    }
  ],
  "confirmationresults": [{"1234": "ok"}, {"5678": "ok"}]
}
```

注：为什么需要设置 `iterationParam`：数据集中的元素可能是一个普通的字符串、数字等非 JSON 对象，为了保证传给子流程的数据为 JSON 对象，所以通过 `iterationParam` 参数进行构造。

错误处理

概述

工作流在执行过程中，可能会因为各种原因出错导致执行失败。目前，`operation`、`foreach`、`parallel` 三类节点支持定义错误重试与错误捕获。

错误重试 (retry)

错误重试，指在节点执行出错时进行多次重试。`retry` 字段负责错误重试的定义，它的值是一个包含若干重试策略的数组，每个重试策略包含了它所匹配的错误码，错误码命中时策略生效，并不再去匹配其它策略。具体字段如下：

字段	类型	描述
errors（必需）	array of string	策略所匹配的错误码集合
intervalSeconds（可选）	integer	第一次重试之前等待的秒数（默认值为1）
maxIntervalSeconds（可选）	integer	最大重试等待时间（默认值为10）
maxAttempts（可选）	integer	重试的最大次数（默认值为2），如果错误发生次数超过指定值，则停止重试
multiplier（可选）	integer	重试时间间隔倍数（默认值为2）

错误捕获（catch）

如果 workflow 发生错误的节点没有定义 `retry`，或者 `retry` 没命中错误码，或者 `retry` 已超过重试最大次数，workflow 会查看该节点是否定义了错误捕获 `catch`。`catch` 与 `retry` 类似，它的值是一个包含若干捕获策略的数组，每个捕获策略同样包含了它所匹配的错误码。如果定义了 `retry` 并且命中捕获策略，那就会按照策略跳转到其它节点执行。具体字段如下：

字段	类型	描述
errors（必需）	array of string	策略所匹配的错误码集合
next（必需）	string	捕获错误后，要执行的下一个节点。下一个节点需要在同一个分支和层级，具体请看 节点拓扑关系

示例

示例如下，`retry` 字段包含了两个重试策略，第一个策略匹配 `CFC.InternalException` 和 `CFC.TooManyRequestsException` 错误，第二个策略匹配任意错误。当两个重试策略未匹配到错误码，或已超过最大重试次数时，则不再重试。如果 `catch` 匹配到了错误码，那么就会执行 `orderFailedHandler` 这个节点。

```
name: demo
start: order
states:
  - type: operation
    name: order
    resource: brn:bce:cfc:bj:123456:function:order:$LATEST
    end: true
    retry:
      - errors:
          - CFC.InternalException
          - CFC.TooManyRequestsException
        intervalSeconds: 5
        maxAttempts: 3
        multiplier: 2
      - errors:
          - XFlow.ALL
        intervalSeconds: 3
        maxAttempts: 2
    catch:
      - errors:
          - XFlow.RuntimeError
        next: orderFailedHandler
  - type: pass
    name: orderFailedHandler
    description: "handle order fail"
    end: true
```

🔗 错误码

retry 和 catch 的策略依赖错误码匹配，相关的错误码如下。

工作流服务错误码：

错误码	描述
XFlow.InternalError	工作流服务内部错误
XFlow.RuntimeError	节点执行过程中发生的错误，比如 switch 判断结果非 bool、输入输出数据非 JSON 等
XFlow.DataLimitExceeded	节点的输入输出数据太大，目前限制为 250 KB
XFlow.OperationFailed	任务节点执行失败，一般为 CFC 函数代码执行时抛出异常或超时
XFlow.ALL	匹配所有错误类型

公共错误码：见 [百度云公共错误码](#)

CFC 错误码：见 [CFC 错误码](#)

典型实践

百度智能小程序使用云开发构建CFC云函数

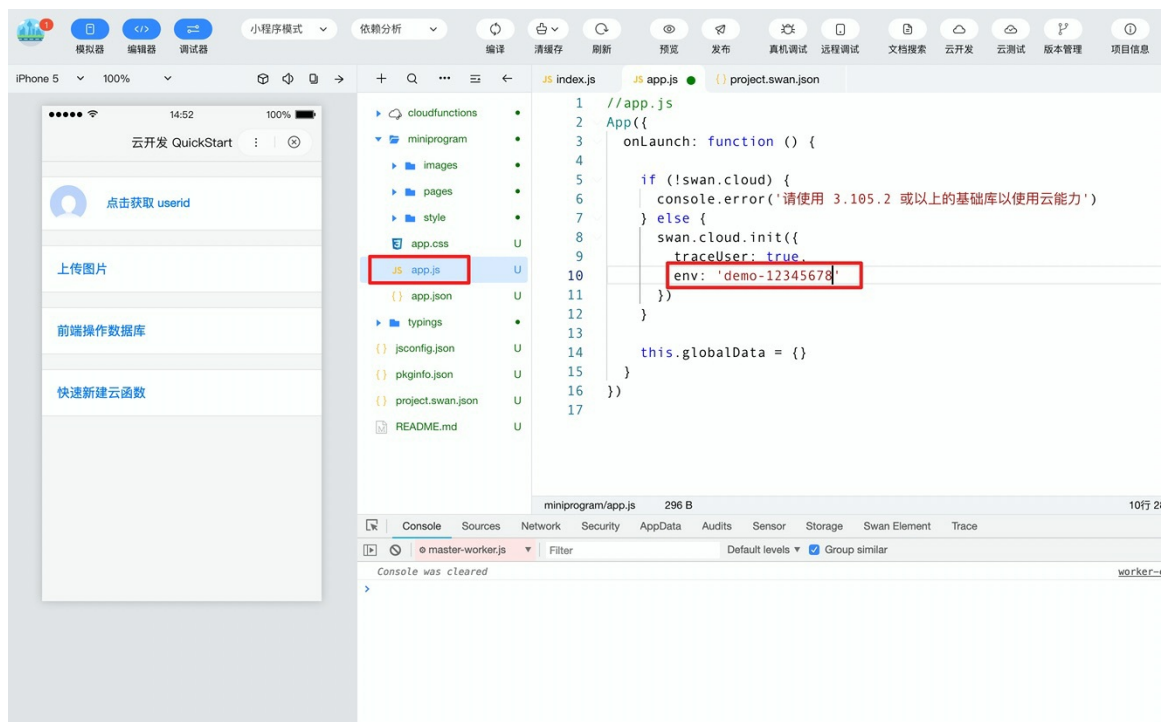
当前，[百度智能小程序](#)已经全面具备了[云开发](#)能力，为开发者提供了一种 Serverless 的小程序开发模式，主要包含了[云函数](#)，[云存储](#)，[云数据库](#)三大部分功能，其中云函数的功能即由[CFC](#)提供了背后支持，本篇文章将向您介绍一下关于云函数的使用建议。

🔗 环境准备

在开发小程序云函数前，您需要做如下准备：

- 申请百度智能小程序 APPID，并开通云开发功能
- 下载并安装百度智能小程序[开发者工具](#)。
- 安装[Nodejs8.5](#)或以上运行环境。
- 参考百度智能小程序[云开发快速上手](#)，创建云开发小程序

当您进入开发者工具，并在如下位置指定了云开发环境ID后，云开发环境就全部准备就绪了



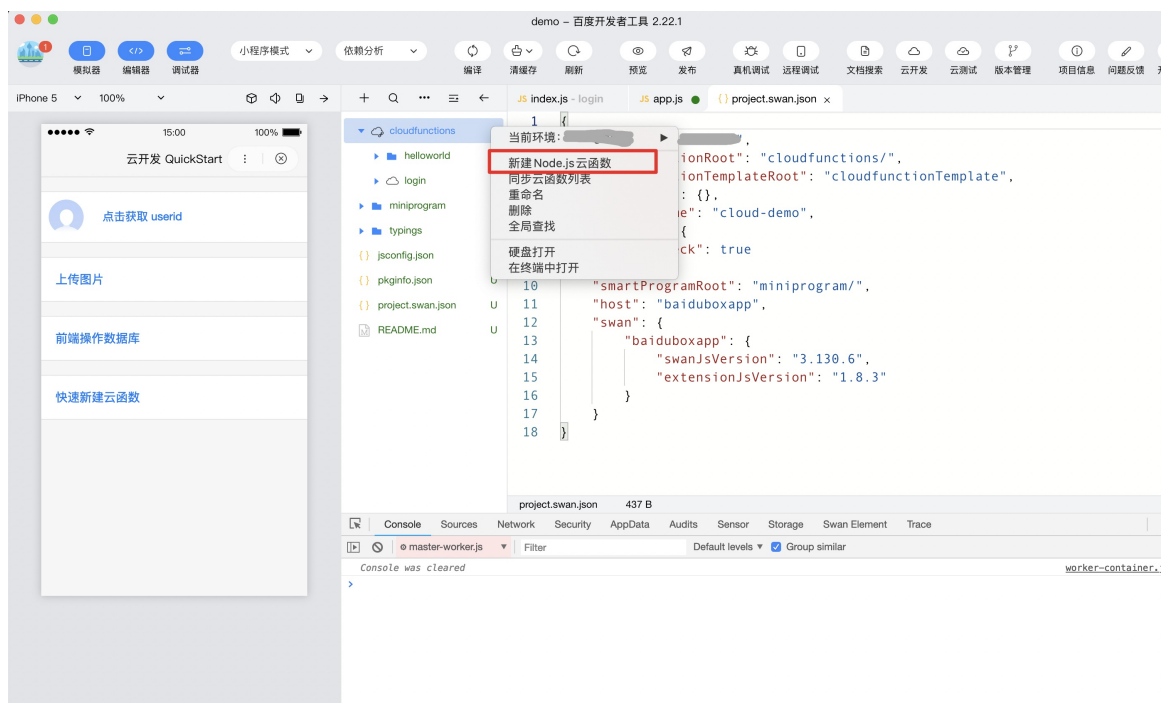
创建云函数

这里我们会演示，如何创建云函数，并在云函数中使用云存储，云数据库，以及调用其他云函数的方式。

helloworld

首先我们创建一个简单的helloworld函数

1. 选择云函数目录，右键选择新建云函数，既可创建一个默认云函数



注：云函数创建完成之后，需要先使用npm安装相关依赖

```
cd /your_cloud_function_dir
npm install
```

2. 修改函数代码，使函数返回helloworld

cloudfunctions/helloworld/index.js

```
// 引用swan-server-sdk, 需要通过npm install安装
const cloud = require('swan-server-sdk');

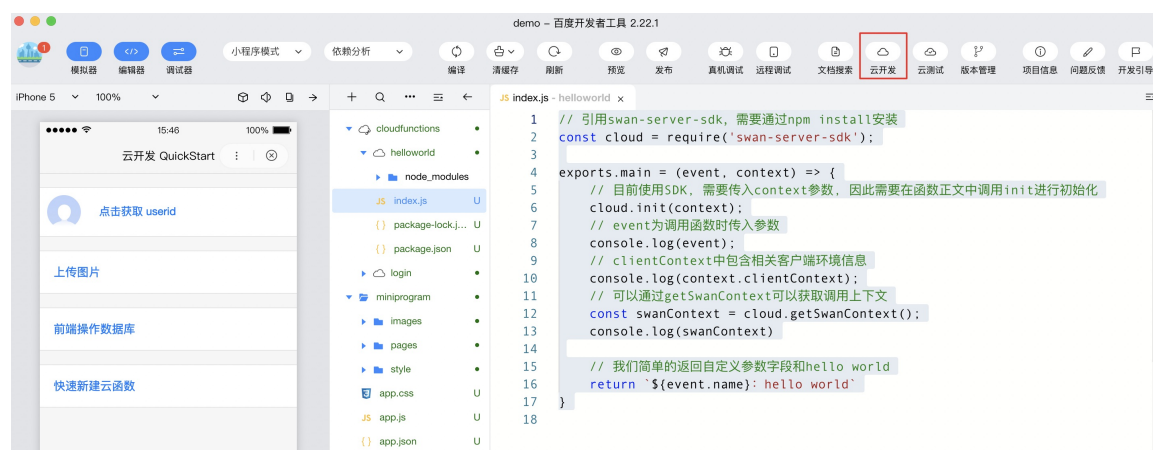
exports.main = (event, context) => {
  // 目前使用SDK, 需要传入context参数, 因此需要在函数正文中调用init进行初始化
  cloud.init(context);
  // event为调用函数时传入参数
  console.log(event);
  // clientContext中包含相关客户端环境信息
  console.log(context.clientContext);
  // 可以通过getSwanContext可以获取调用上下文
  const swanContext = cloud.getSwanContext();
  console.log(swanContext)

  // 我们简单的返回自定义参数字段和hello world
  return `${event.name} : hello world`
}
```

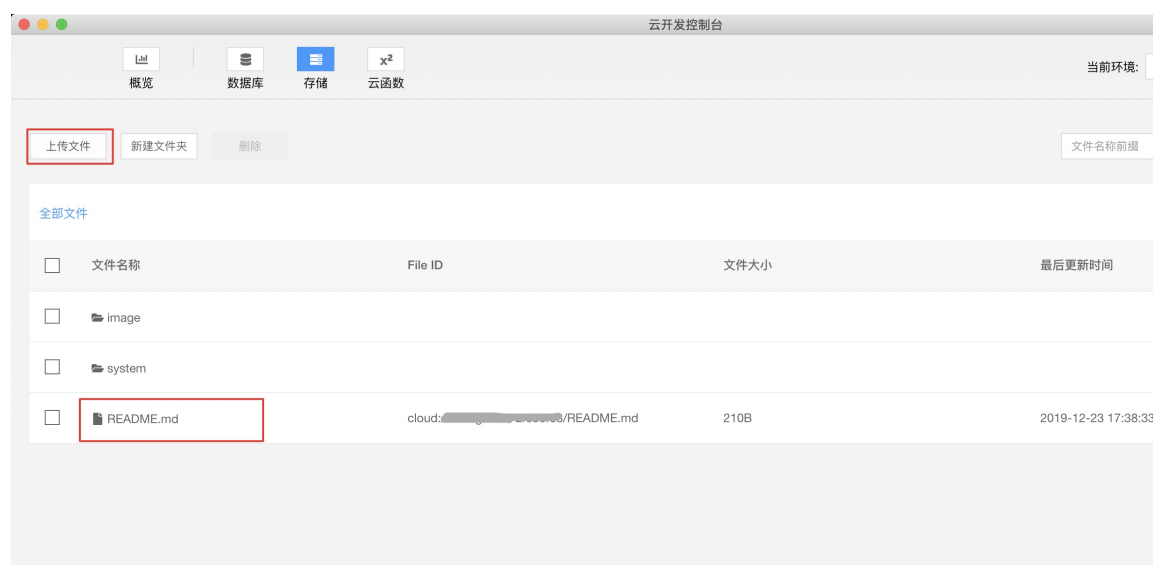
熟悉云开发控制台

在创建新的云函数之前, 我们可以先熟悉一下云开发控制台, 在这里可以通过UI为我们下一个云函数准备一些云存储和云数据库资源

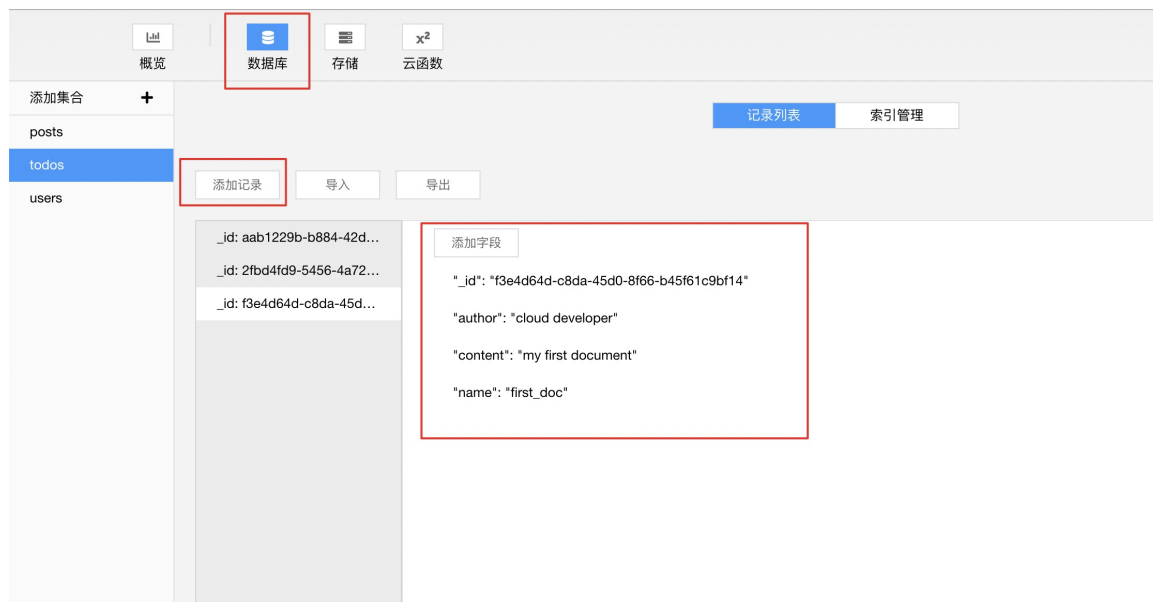
1. 进入云开发控制台



2. 在云存储页面上传文件



3. 在云数据库页面创建文档



注：当然以上这些操作我们也可以通过SDK直接完成。

使用SDK

按照之前的步骤，我们创建一个新的云函数。

编辑函数代码：

cloudfunctions/swan-server-sdk/index.js

```
const cloud = require('swan-server-sdk')

exports.main = async (event, context) => {
  cloud.init(context)

  // 获取云存储文件临时下载地址
  const file = await cloud.getTempFileURL({
    fileID: 'README.md',
  })

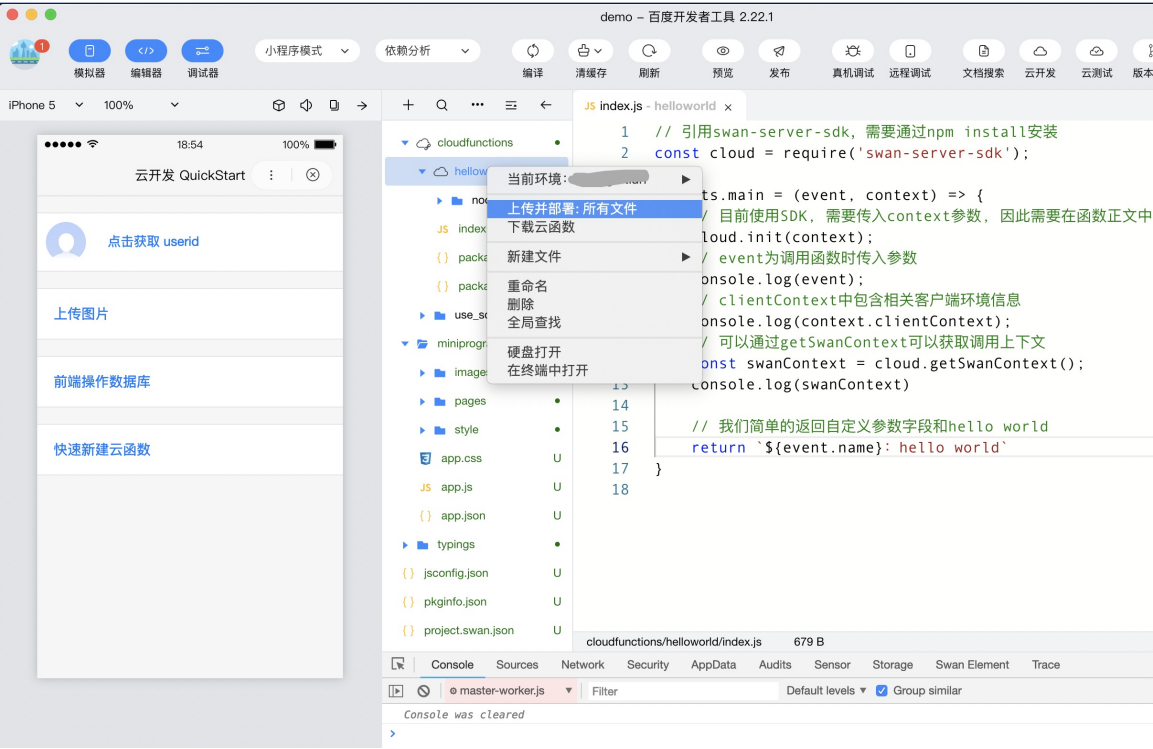
  // 获取云数据库文档内容
  const doc = await cloud.database()
    .collection('todos')
    .doc('f3e4d64d-c8da-45d0-8f66-b45f61c9bf14')
    .get()

  // 调用云函数调
  const res = await cloud.callFunction({
    name: 'helloworld',
    data: {
      name: "cloud developer"
    }
  })

  return {
    file: file.tempFileURL,
    doc: doc,
    result: res.result
  }
}
```

部署云函数

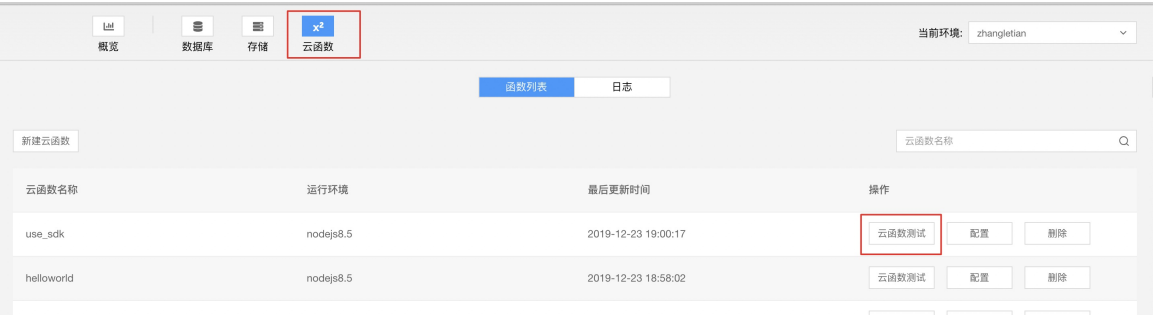
在编写完云函数后，我们需要把本地代码上传到线上，然后才可以真正使用函数计算，右键选择需要部署的云函数，上传并部署：所有文件



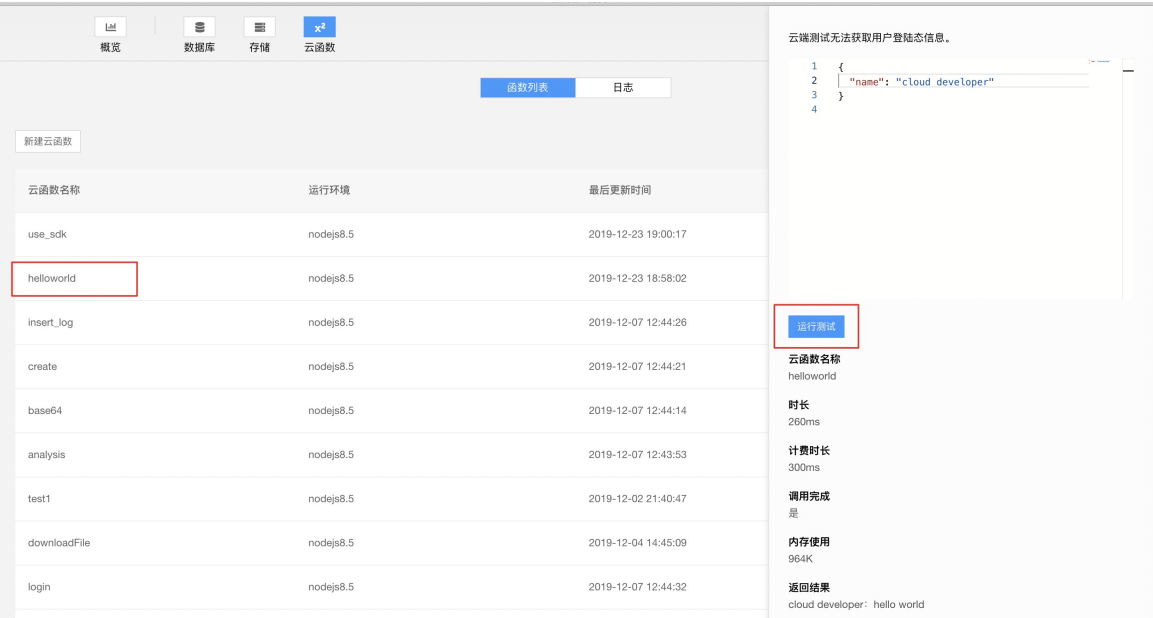
注：在部署之前我们需要确保已经本地npm install所有依赖

测试云函数

函数部署完成后，我们就可以在控制台进行测试了，看一看函数在云端的执行效果。



调用函数：helloworld



调用函数：use_sdk

新建云函数			
云函数名称	运行环境	最后更新时间	
use_sdk	nodejs8.5	2019-12-23 19:00:17	
helloworld	nodejs8.5	2019-12-23 18:58:02	
insert_log	nodejs8.5	2019-12-07 12:44:26	
create	nodejs8.5	2019-12-07 12:44:21	
base64	nodejs8.5	2019-12-07 12:44:14	
analysis	nodejs8.5	2019-12-07 12:43:53	
test1	nodejs8.5	2019-12-02 21:40:47	
downloadFile	nodejs8.5	2019-12-04 14:45:09	

运行测试
云函数名称 use_sdk
时长 520ms
计费时长 600ms
调用完成 是
内存使用 1.1M
返回结果 <pre>{ "file": "https://img.alicdn.com/imgextra/i1/2206666666/User/README.md?authorization=bce-auth-v1%2F125fd0d2ba044da5c0bc916045d316%2F2019-12-23T11%3A00%3A00Z%2F4200%2Fhost%2Fce4bd5b499ac5f8d714adcb["data":{"_id":"13e4d64d-c8da-45d0-8f66-b45f61c9bf14","name":"first_doc","author":"cloud developer"},"content":"my first document"}]","requestID":"067e00bb-fd76-4dd4-839b-a33d2689db06","total":1,"limit":1,"offset":0},"result":{"cloud developer: hello world"}</pre>

🔗 更多参考

微信小程序云开发文档：https://smartprogram.baidu.com/docs/develop/cloud/cloud_info/

微信小程序开发者社区：<https://smartprogram.baidu.com/forum/>

DuerOS开放平台开发者使用CFC构建一个Custom-Skill

DuerOS 技能开放平台，英文全称 DuerOS Bots Platform，是 DuerOS 为第三方开发者提供的一整套技能开发、测试、部署工具的开放平台。第三方开发者可以在平台上，通过可视化界面，简单、高效地开发各类个性化技能。

本典型实践是指导 DuerOS 开放平台开放者用户使用 CFC 创建一个 Custom Skill。

☞ 开通 CFC 服务

1. 使用百度账号登录 [DuerOS技能开放平台](#)。
 - 注：若没有用户名，请先完成注册，操作请参考[注册百度账号](#)。
2. 进入[控制台页面](#)，点击“**创建新技能**”按钮，在弹窗中技能类型选择“自定义”，并选择“**从头开始**“，然后依次输入技能名称和调用名称，选择应用场景和技能付费模式后，点击“确定”完成技能名称的创建步骤，并进入技能基础信息页面。
3. 选择技能配置服务页面，然后选择“**百度智能云CFC**”并点击**使用链接**，我们将为您开通CFC函数计算服务的使用权限，并且链接在登录百度智能云控制台账号的前提下，可以直接进入函数计算CFC产品页面，点击创建函数后即可在创建环节，进行您的函数创建。

授权信息配置

服务部署

技能平台SDK提供了Web开发技能所需的各种API接口， 查看[PHP SDK 技术文档](#)或查看[Node.js SDK 技术文档](#)。推荐使用平台自带的部署服务，我们已经为您完成了全部的部署工作，您无需任何部署操作，只需在线编辑代码即可，并且这是完全免费的。

代码编辑

Web Service

百度云CFC

选择 CFC

* 使用链接

您可以免费使用一定额度的百度云CFC服务， [点击即可进行服务配置](#)。具体计费方式请参考[CFC计费文档](#)。

点击进行 CFC 函数的创建

* BRN

创建后将函数 BRN 复制此处

* 是否回调

执行回调

不回调

使用说明

不回调将无法使用技能数据能力

权限获取配置

保存

授权

注意：

技能名称：技能名称是技能在小度技能商店中展现的名称，用户通过技能名称，快速辨识理解技能能力。

调用名称：技能在被用户调用时的名称，用户会通过表达“打开+调用名”进行使用技能 名称唯一：调用名称唯一，不能重复占用。

参考建议：清晰易辨识、避免同音词、避免使用生僻词组、避免复杂自创词汇、避免中文与英文数字混杂、不得带有“小度”或“小度小度”

商标占用：非企业认证用户不得使用品牌名称（及同音词）作为调用名，企业级认证用户仅能使用该公司品牌及旗下品牌作为调用名。

创建函数

函数的创建有两种方式，您可以根据自己的需求，任选一种方式开始您的函数创建。

从头创建函数

1. 登录百度智能云平台[官网](#)。

2. 登录成功后，选择“产品服务>云函数计算 CFC”，进入“函数列表”页面，点击**创建函数**，进入创建函数界面，选择空白函数创

建。

函数管理

计费模式和免费额度变更通知

1. CFC计费模式和免费额度于2023年3月10日进行调整，用户开通使用CFC三个月后每月不再享受免费额度，为了答谢老客户，平台推出力度更大、种类更多的套餐包，您可以选择适合自己性价比更高的套餐包来享受更大的优惠。 详见[资源量包文档](#)。

2. 新用户可在开通CFC后的3个月内享受免费额度，您可以通过“0元购买套餐包”的形式享受免费额度。

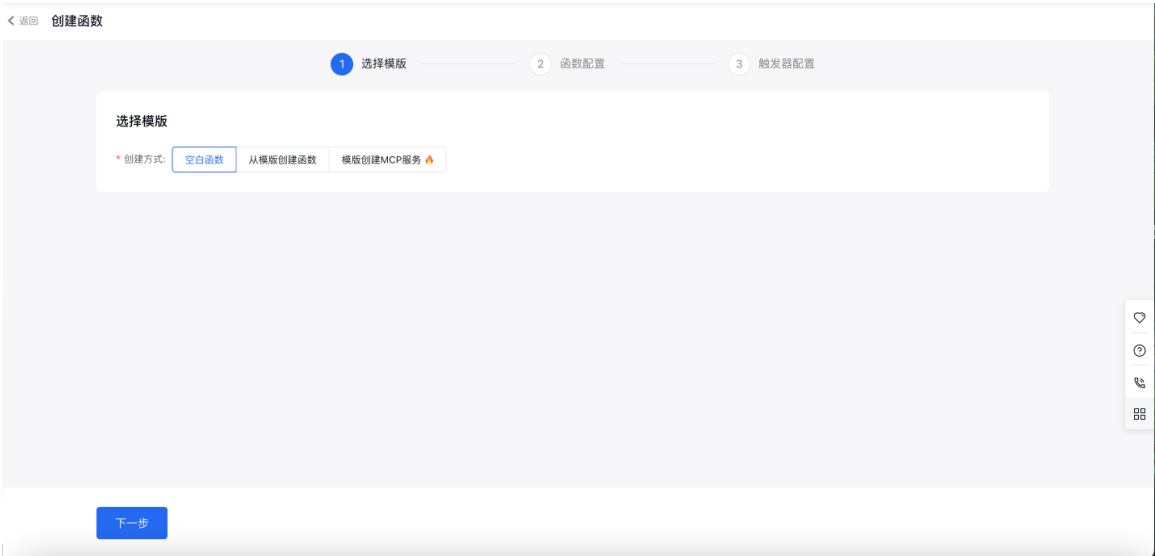
+ 创建函数

删除

服务名称: 全部 运行时: 全部 请输入函数名搜索

函数名	服务名称	运行时	代码体积	修改时间	层依赖	描述	操作
xyf_mcp	default	Python 3.10_mcp	16.4 MiB	2025-05-22 16:52:37	--	--	删除
test-bls	default	Python 3.6	1.3 KiB	2025-05-12 10:23:56	--	--	测试 删除

202



3. 填写函数名称、描述，并根据需要选择运行语言，本典型实践以Nodejs 6.11的运行语言为例。

基础信息

* 函数名称:

hello_event

填写函数名

只能由数字、字母、-或_组成，长度小于65个字符，且在当前地域内唯一

* 服务名称:

default

管理服务 选择服务名

* 事件模式:

event

默认event，无需修改

* 运行时:

Node.js 16

选择你熟悉的语言

* 内存:

128

1285761024

* 超时时间:

3

最大限额300秒

日志存储:

无

描述:

请输入描述

上一步

下一步

提交

点击提交

> **说明：**
> 运行语言目前支持 Python 2.7、Python 3.6、Python 3.10、Node.js 8.5、Node.js 10、Node.js 12、Node.js 14、Node.js 16、Java8、Golang 1.x、PHP 7 等。

4. 点击创建按钮，完成创建。
5. 进入函数代码页面，代码编辑类型分为：在线编辑和上传 .zip 文件两种，本实践以“在线编辑”为例，在“在线编辑器”中输入函数代码，进行您的函数创建。本操作以 DuerOS 技能的实例，输入文本“你好，世界！”，返回一段语音 Response 为例，代码如下。

```
let resp = {
  "version" : "2.0",
  "response" : {
    "outputSpeech": {
      "type" : "PlainText",
      "text" : "你好，世界！",
    },
    "shouldEndSession" : false
  }
}

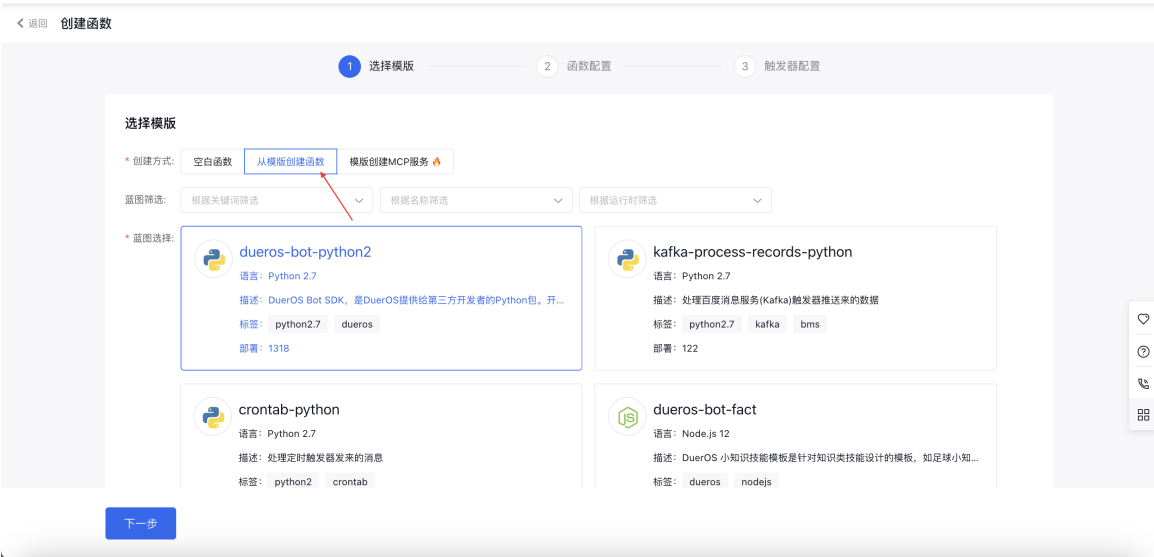
exports.handler = (event, context, callback) => {
  callback(null, resp);
};
```

6. 环境变量，您还可以自定义函数执行时的环境变量。环境变量的总长度不能超过 4KB。

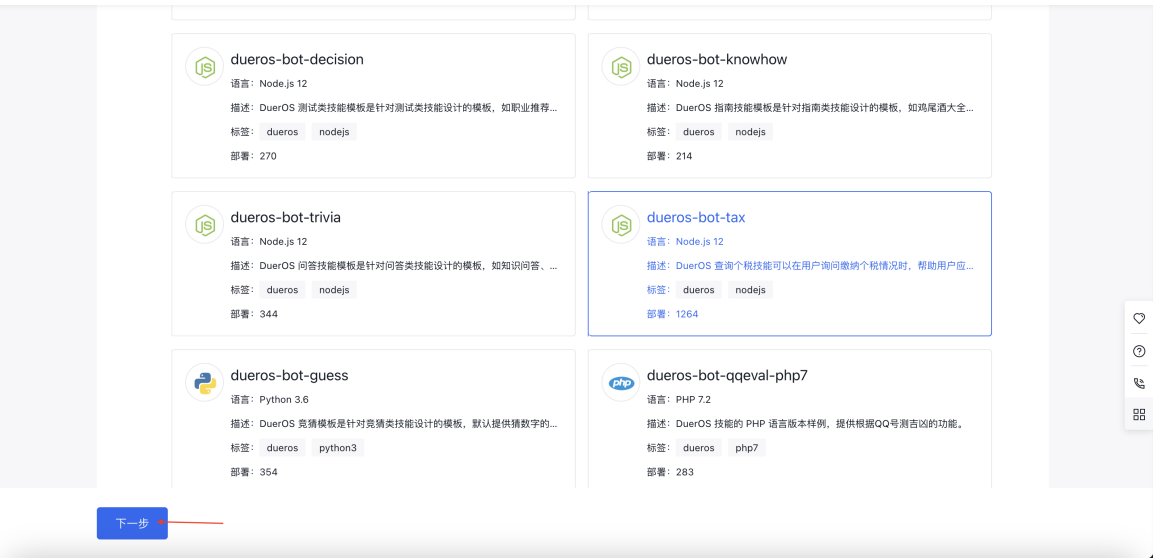
7. 进入函数触发器页面，新增触发器，选择“DuerOS 触发器”。

从模板创建函数

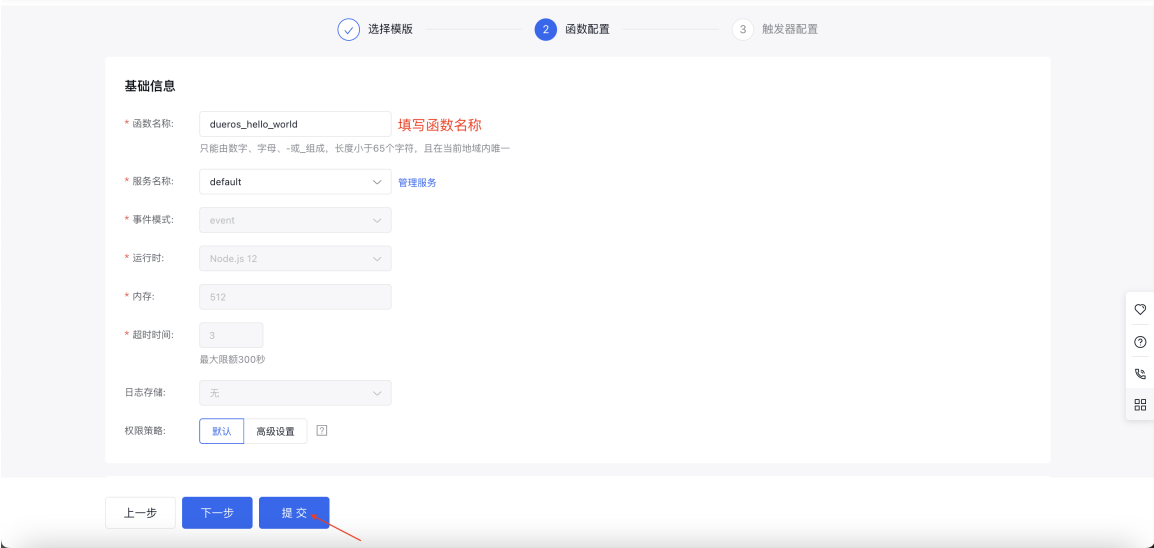
- 1. 登录百度智能云平台[官网](#)。
- 2. 登录成功后，选择“产品服务>云函数计算 CFC”，进入“函数列表”页面，点击**创建函数**，进入创建函数界面，选择**从模版创建函数**。



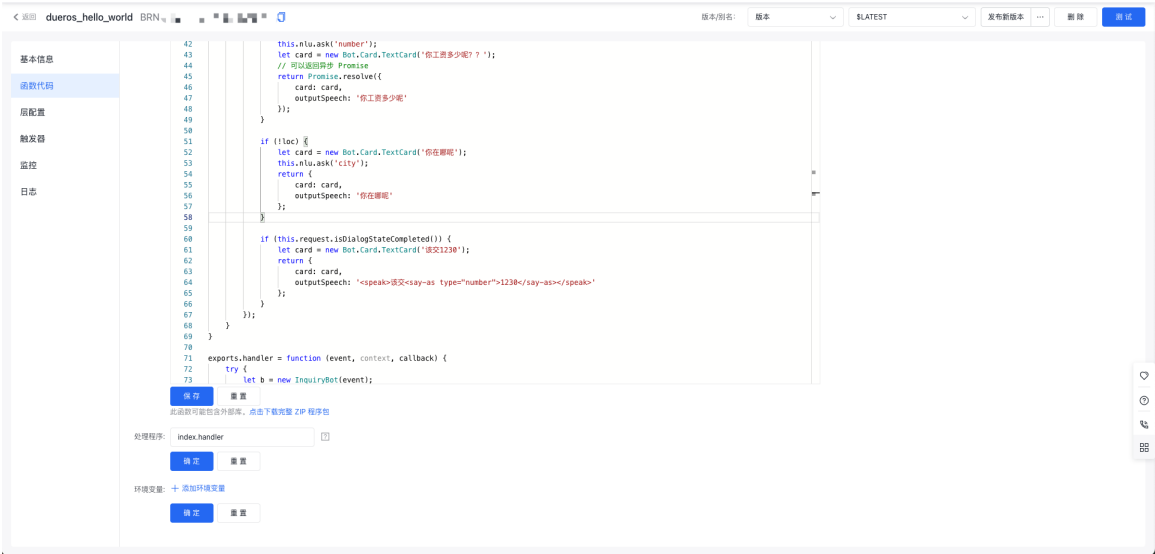
3. 点击后，界面上会显示目前所有的模板，选择一个 DuerOS 相关的模板，点击下面的“下一步”按钮。



4. 在弹框中输入函数名称，点击提交。

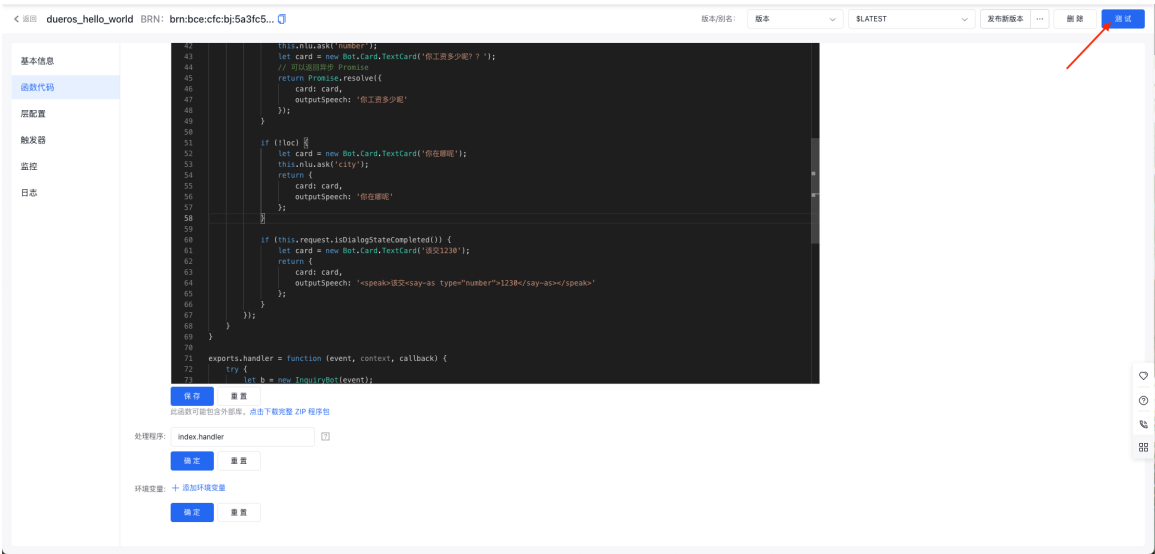


5. 点击提交后，即可创建一个DuerOS 技能的函数，这个函数默认内置了 BOT-SDK，用户可以通过界面上的编辑框修改 intent handler 来让这个函数响应DuerOS 技能 的事件。

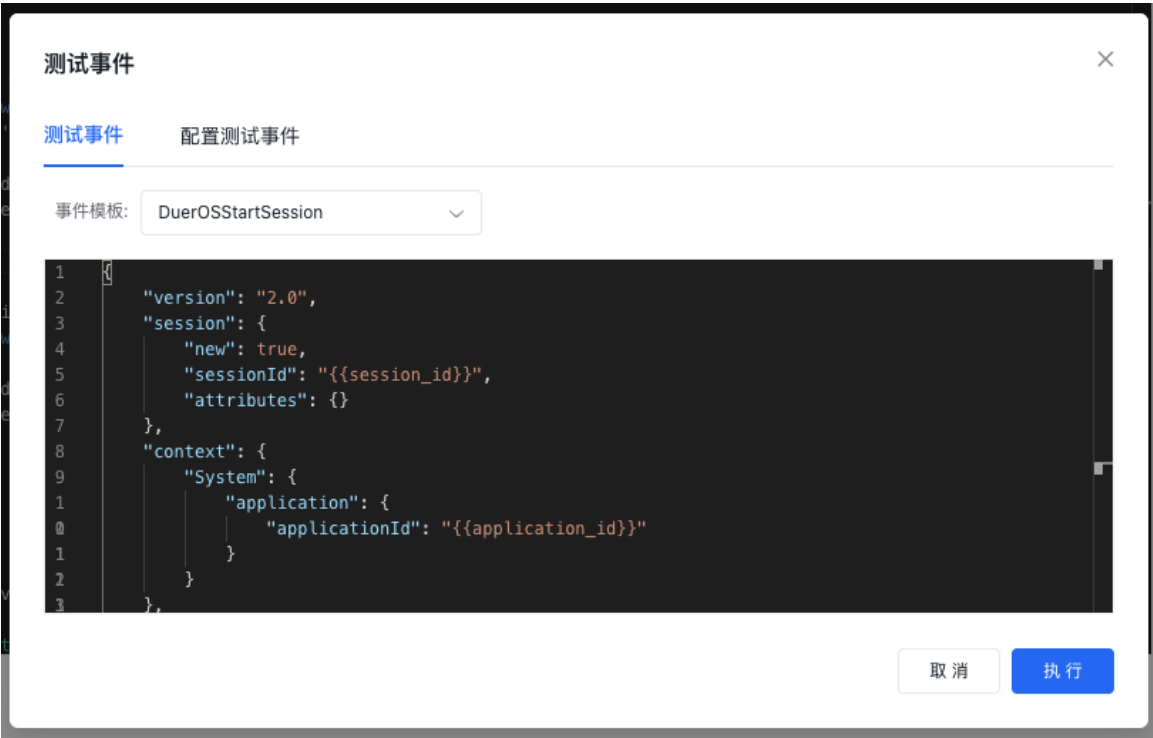


测试你创建的函数

1. 在函数列表页点击函数名右侧的“测试”按钮，或者在函数基本信息页和函数代码页面，都可找到“测试”按钮，进入测试弹框。



2. 输入测试的事件，这里我们创建一个“DuerOSStartSession”类型的事件。



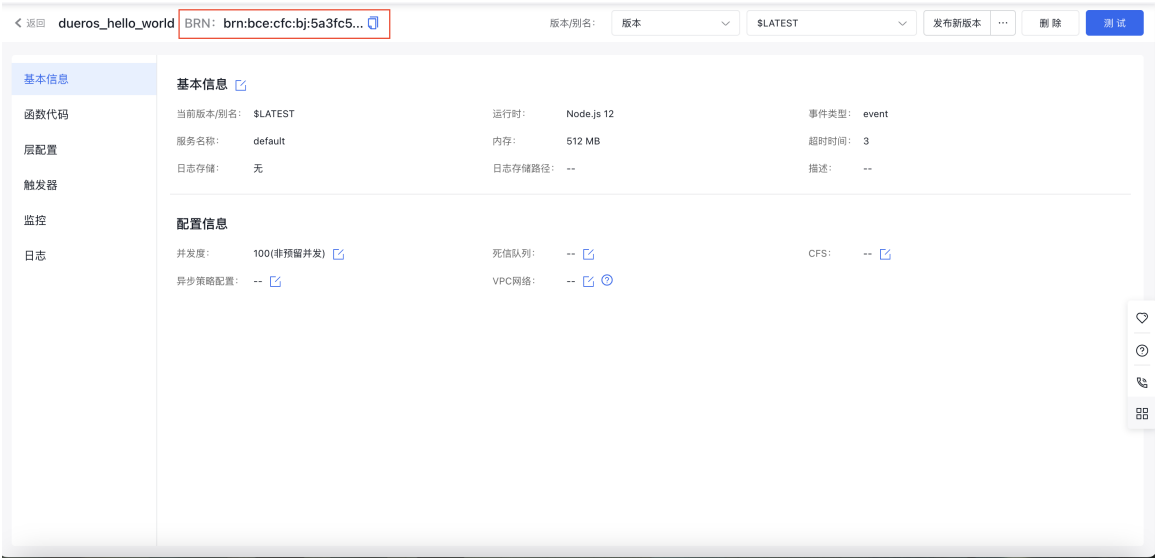
3. 点击“执行”按钮，即可得到返回。



🔗 查询函数BRN

1. 使用您创建CFC函数的百度智能云账号，登录百度智能云平台[官网](#)。

2. 登录成功后，选择“产品服务>云函数计算 CFC”，进入“函数列表”页面，点击想要获得BRN的函数名称,进入函数详情页面可以查看函数的BRN。



绑定 DuerOS 技能 和 CFC 函数

1. 返回 DuerOS 技能的设置界面，接入方式选择 BRN（Baidu Resource Name），在 BRN 的输入框中输入在百度智能云的资源名，即可完成绑定。
2. 在DuerOS技能平台配置服务界面，“是否回调”列，选择执行回调。

注意：如果选择不回调，将无法使用 DBP 平台的技能数据统计的能力。

3. 选择回调后，您需要输入函数的 Public Key,Public Key 获取方式请参考 [DuerOS技能开放平台](#)，输入后点击“保存”完成配置服务项的填写。
4. 创建意图，点击左侧导航栏中的“意图”进入意图详情页面，然后选择“创建意图”进入创建意图界面。
5. 意图信息填写，按照要求填写意图中文名、意图标识名。



注意：意图中文名称：用户调用创建事件意图时使用的意图名称。意图标识名：用户在函数创建时代码中所写的对应调用事件的名称。

6. 词典，词典分为自定义词典和系统词典。自定义词典是指开发者根据需求创建的词典。系统词典是对一些通用领域的词汇进行整理的词典，如城市词典、计量单位词典、等，开发者可以直接使用。自定义词典可以参考 [DuerOS词典](#)，系统词典使用可以参考 [DuerOS系统词典](#)
7. 常用表达，您可以根据创建函数的用处，输入运行函数的常用表达语句，本典型实践为“查询个税”这个程序为例，进行演示。如下图所示：

常用表达 2

回车添加用户常用表达

我在石家庄月收入3600

槽位标识	中文名称	对应词典	值
city	请输入中文名称	@sys.city	石家庄
number	请输入中文名称	@sys.number	3600

我在厦门

月收入60000

我在深圳月收入20000

我在北京月收入50000要交多少税

上一页 下一页

添加词典

* 添加方式

引用系统词典

Q 快速搜索

☐ @sys.country (国家)

☒ @sys.city (中国城市)

☐ @sys.province (中国省份)

☐ @sys.county (区/县/郡)

☐ @sys.village (村庄/乡镇)

☐ @sys.color (颜色)

☐ @sys.animal (动物)

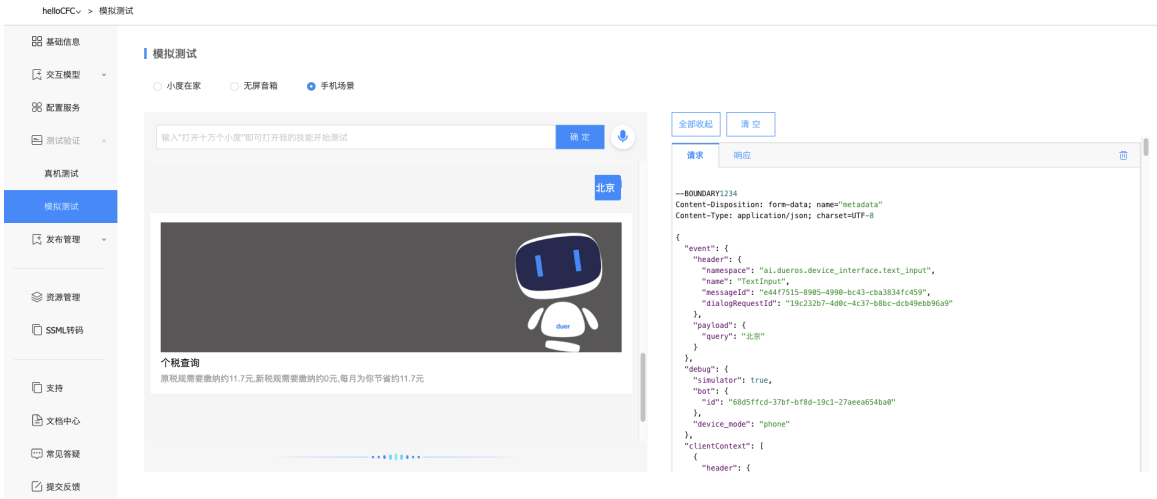
☐ @sys.foreign_city (国外城市)

注意：系统会自动对您输入的文字中的城市名称和收入金额进行槽位标记，如果标记不符合您的业务需求，您可以在输入的文字中选择您想要标记的文字，在弹出的窗口中选择“添加词典”，进行自主标记。

8. 在技能回复中选择“服务配置满足”，然后在页面最下角选择保存完成“查询个税”事件的创建。

测试 DuerOS 技能

1. 使用百度账号登录 [DuerOS技能开放平台](#)。
2. 选择需要测试的自定义技能名称，进入技能详情页面，在左侧导航栏中选择配置服务，进入测试页面。
3. 在输入框中输入“打开+调用名称”打开事件，然后输入想要查询个税的金额或者城市，进行个税查询。



说明：

当用户说“我想查个税”时，DuerOS 会发送 IntentRequest 消息，技能处理消息时，确认意图信息，发现没有工资（sys.number）和城市（sys.city）信息，所以进行槽位追问。

当用户说“我工资 8000”，DuerOS 会发送 IntentRequest 消息，技能记录下工资（sys.number），发现没有城市（sys.city）信息，所以继续追问。

当用户说“我在北京”，DuerOS 会发送 IntentRequest 消息，技能记录下城市（sys.city）信息，并确认已经获得完整的用户请求信息，开始处理用户请求，并向用户返回处理结果。

配置CFC函数以访问百度云中的RDS

关系型数据库RDS，英文全称Relational Database Service是一个专业化的高性能、高可靠云数据库服务，支持Mysql、SQL Server、PostgreSQL引擎，提供简易方便的Web界面管理、可靠的数据备份和恢复、完备的安全管理、完善的监控等功能。在本教程中，您将执行以下操作：

- 创建您的百度云MySQL 数据库引擎实例（仅Mysql5.7以上支持CFC访问）。在 MySQL 实例中，您将创建一个数据库 (exampledb)，其中包含一个示例表 (employee)。关于RDS的更多信息，请参阅[百度云RDS](#)。
- 创建一个 CFC 函数来访问 ExampleDB 数据库，创建一个表 (Employee)，添加几个记录，然后检索表中的记录。
- 手动调用 CFC 函数并验证查询结果。这样您可以验证您的 CFC 函数是否可以访问 VPC 中的 RDS MySQL 实例。

创建RDS

在本教程中，您的CFC函数将创建一个表 (Employee)，插入一些记录，然后检索这些记录。CFC函数创建的表具有以下结构：

Employee(EmpID, Name)

首先，您需要启动RDS Mysql5.7 实例。如果您在百度云上存在RDS Mysql5.7 实例，请跳过此步骤。

- 您可以使用百度云控制台启动百度的RDS MySQL 实例 参见[创建RDS for Mysql主实例](#)
- 创建RDS之后，请[开通公网访问](#)和[设置SSL加密](#)
- 创建数据库exampledb

< 返回实例列表

基本信息

账号管理

数据库管理

参数配置

监控

备份

日志管理

安全

产品服务 / 云数据库 RDS-实例列表 / 数据库管理 / 创建账号

ID: rds-duiNWd5d (名称: mysql57) ● 运行中

+ 创建代理实例 ? + 创建只读实例

数据库名称: *

exampledb

输入数据库名称

由大小写字母、数字、下划线组成、字母开头，字母或数字结尾，最长64个字符

支持字符集: *

utf8

账号授权:

暂无账号，去创建账号

备注:

最多256个字符（一个汉字等于三个字符）

确认

取消

• 创建用户

< 返回实例列表

基本信息

账号管理

数据库管理

参数配置

监控

备份

日志管理

安全

产品服务 / 云数据库 RDS-实例列表 / 账号管理 / 创建账号

ID: rds-duiNWd5d (名称: mysql57) ● 运行中

+ 创建代理实例 ? + 创建只读实例

* 数据库账号:

testuser

输入账号名称

由小写字母、数字、下划线组成、字母开头，字母或数字结尾，最长16个字符

数据库授权:

数据库名称	权限设置
☒ exampledb	☐ 只读 ☒ 读写

选择权限

* 密码:

.....

输入密码

由字母、数字和特殊字符 (!@#\$\$%^_) 中的至少两种组成，长度6-32位

* 确认密码:

.....

确认密码

备注:

最多256个字符（一个汉字等于三个字符）

确认

取消

- 记下数据库名称、用户名称和密码。您还需要数据库实例的公网地址和CA证书，您可以从 RDS 控制台获取该信息（您可能需要等待，直到实例状态变为可用并且终端节点值显示在控制台中）。



创建CFC函数

1.登录百度云平台官网。登录成功后，选择“产品服务>云函数计算 CFC”，进入“函数列表”页面，点击创建函数，进入创建函数界面



2.填写函数名称、描述，并根据需要选择运行语言，本教程以Node.js 16的运行语言为例。

基础信息

* 函数名称:

hello_event

填写函数名

只能由数字、字母、-或_组成，长度小于65个字符，且在当前地域内唯一

* 服务名称:

default

管理服务 选择服务名

* 事件模式:

event

默认event，无需修改

* 运行时:

Node.js 16

选择你熟悉的语言

* 内存:

128

128 576 1024

* 超时时间:

3

最大限额300秒

日志存储:

无

描述:

请输入描述

上一步

下一步

提交

← 点击提交

3.选择“编辑类型”，编辑类型分为：在线编辑和上传.ZIP文件两种，本教程将通过上传.ZIP文件方式创建函数，首先在创建example目录，进入此目录执行以下npm命令

```
npm init
npm install mysql --save
```

打开您的本地文本编辑器，并编写您的代码。保存文件(例如，index.js)。 在创建 cfc 函数时，您将使用此文件名指定处理程序。

```

var fs = require('fs');
var mysql = require('mysql');
var connection = mysql.createConnection({
  host: 'mysql57.rds.cn-hangzhou.mysql.aliyuncs.com', // 上文中的受保护地址或公网域名
  user: 'testuser',
  password: 'password',
  database: 'exampledb',
  ssl: {
    ca: fs.readFileSync(__dirname + '/ca.pem')
  }
});
connection.connect();
console.log('connected!')
exports.handler = (event, context, callback) => {
  var p = new Promise(function(resolve, reject){
    var employeeAddSql = 'INSERT INTO employee(Name) VALUES(?)';
    var employeeAddSqlParams = ['LiLei'];
    connection.query('CREATE TABLE IF NOT EXISTS employee ( EmpID int NOT NULL AUTO_INCREMENT, Name
varchar(255) NOT NULL, PRIMARY KEY (EmpID))', function (error, results) {
      if (error) {
        console.log(error);
        reject(error)
      };
    });
    connection.query(employeeAddSql, employeeAddSqlParams, function (error, results) {
      if (error) {
        console.log(error);
        reject(error)
      };
      console.log("success");
      resolve(results);
    });
  });
  p.then(function (value) {
    callback(null, value);
  }, function (reason) {
    callback(null, reason);
  })
};

```

注意

为了获得更好的性能，connection.connect();放在了处理程序之外执行。

将CA证书放入同一目录,目录结构如下

```

.
├── ca.pem
├── index.js
├── node_modules
├── package-lock.json
└── package.json

```

压缩文件夹的内容，即您的部署程序包(例如，example.zip)。进入目录

```
zip -r example.zip *
```

然后，在创建您的CFC函数时指定此选择上传.ZIP文件，上传此zip包

4.环境变量，您还可以自定义函数执行时的环境变量。环境变量的总长度不能超过 4KB。

测试函数

点击右上角的测试，进行函数测试

< 返回函数列表

基本信息

函数代码

触发器

监控

日志

rds-demo 版本/别名: \$LATEST

BRN: brn:bce:cfc:bj:1a2cbf55b97ac8a7c760c4177db4e17d:function:rds-demo:\$LATEST

函数代码

发布新版本

测试

点击测试

编辑类型:

在线编辑

上传函数.ZIP包

当前代码:

```
1 var fs = require('fs');
2 var mysql = require('mysql');
3 var connection = mysql.createConnection({
4   host: '192.168.1.1', // 上文中的受保护地址或公网域名
5   user: 'root',
6   password: '123456',
7   database: 'test',
8   ssl: {
9     ca: fs.readFileSync(__dirname + '/ca.pem')
10  }
11 });
12 connection.connect();
13 console.log('connect!')
14 exports.handler = (event, context, callback) => {
15   var p = new Promise(function(resolve, reject){
16     var employeeAddSql = 'INSERT INTO employee(Name) VALUES(?)';
```

保存 重置

此函数可能包含外部库。点击下载完整 ZIP 程序包

处理程序: index.handler 修改

环境变量: 编辑

在弹窗中点击执行

测试事件

当前测试事件

配置测试事件

事件模板：

HelloWord模版

* 事件名称：

请输入事件名称

只能由数字、字母或下划线组成

1

{

2

"key1": "value1",

3

"key2": "value2"

4

}

✓ 执行结果：成功

此区域下方显示你的函数执行的结果。

{
 "fieldCount":0,"affectedRows":1,"insertId":12,"serverStatus":2,"warningCount":0,"message":"","protocol41":true,"changedRows":0}

返回的结果

创建

执行

点执行

CFC函数搭建Flask-Web框架

概述

百度云CFC Flask框架模版旨在实现flask-web serverless化，用户可通过该模版的示例代码，使用url去访问函数。

使用场景

Flask-web应用，如Flask url构建。

环境准备

安装Python3及以上运行环境。

操作部署

如果您还没有开通CFC服务，请参考[函数计算概览](#)开通；如果您已开通，则可以跳过该步骤。

使用该模版创建函数时，需要为函数创建Http触发器，具体创建流程如下：

- 1、登录百度智能云平台[官网](#)
- 2、登录成功后，选择“产品服务”->“容器”->“函数计算 CFC”，进入“函数列表”页面，点击从模版创建函数，选择flask框架模版，进入创建函数页面
- 3、点击下一步，在弹框中输入函数名称，点击下一步创建Http触发器
- 4、为函数创建Http触发器，其中URL路径格式为/xxx/{proxy+}，如/test/{param+}或/{param+}，选择HTTP方法，点击提交，即可完成函数和触发器创建。

注意：当为多个函数创建Http触发器时，为了避免URL重复，应保证URL/xxx/{proxy+}中的xxx不同，如为func1配置URL为/test1/{param+}，为func2配置的URL为/test2/{param+}。

- 5、创建成功后，进入函数详情页，可以查看函数代码
- 6、查看触发器，根据触发器URL路径访问
- 7、使用URL访问路由

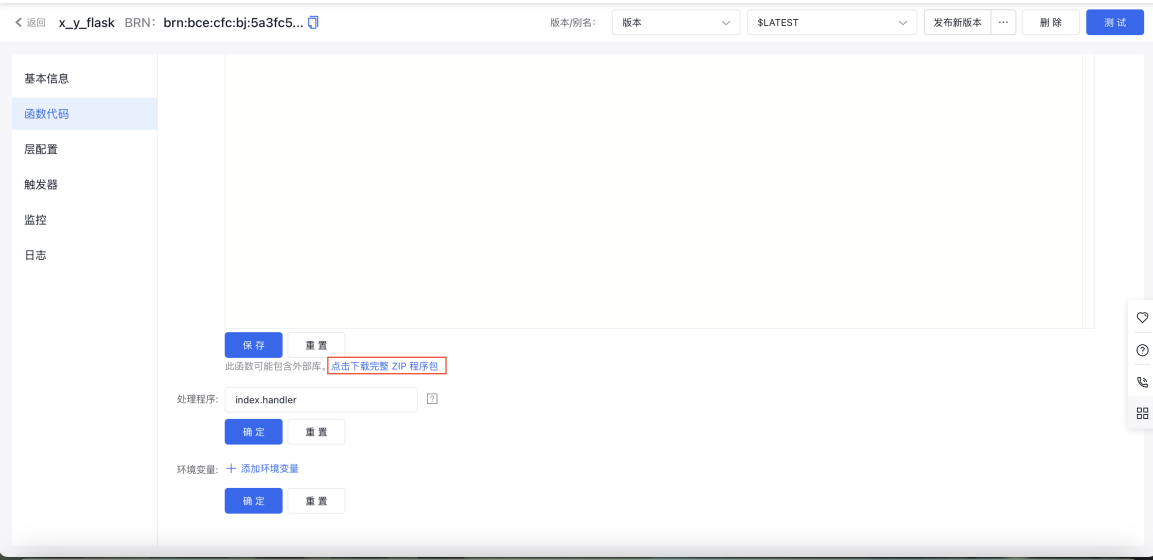
应用扩展

用户可以下载上述创建的函数代码zip，本地开发添加需要的逻辑代码，重新打包上传即可。

1、根据需求更改代码逻辑

下载函数代码包

其中index.py是CFC函数执行入口(如果缺少该文件，则会提示找不到index.py)，app.py为用户逻辑代码，用户可以根据需要更改app.py中的内容，如上述，改变app.route("/users/idx")路由方法，改成根据URL参数提示。



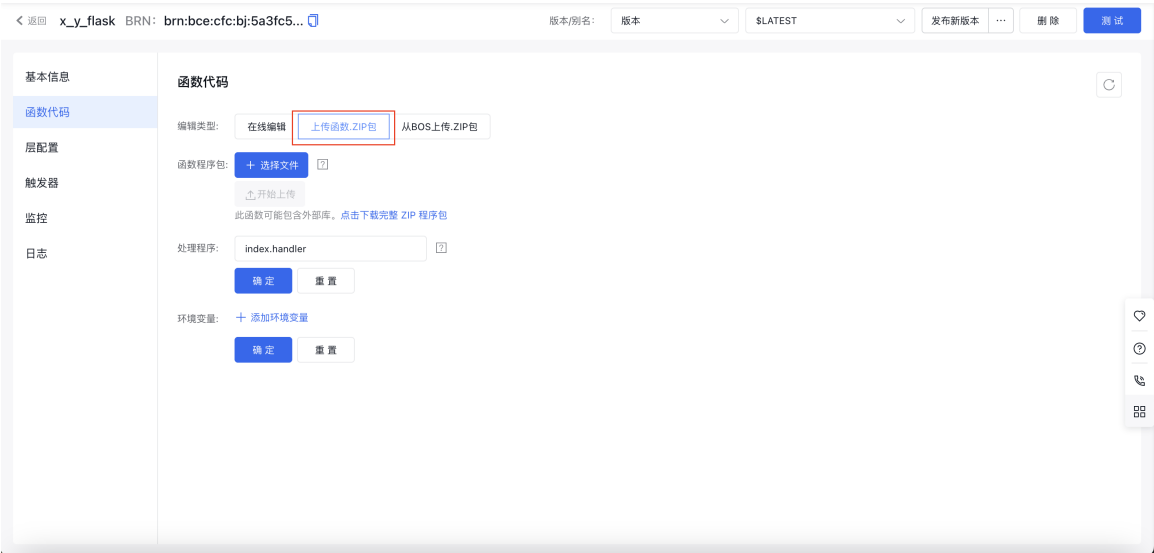
2、打包文件上传

打包文件时，需要在当前目录下打包，即在项目根目录下执行zip -r xxx.zip进行打包，然后在控制台或使用OpenAPI更新函数代码。

```
x_y_flask_dabd1dd6-50c1-475f-b89a-f1a6726a80cf % ls
Flask-1.0.2.dist-info    app.py                  flask                   jinja2
Jinja2-2.11.2.dist-info bin                     index.py               markupsafe
MarkupSafe-1.1.1.dist-info click                   itsdangerous           severless_wsgi.py
Werkzeug-0.16.0.dist-info click-7.1.2.dist-info  itsdangerous-1.1.0.dist-info werkzeug
x_y_flask_dabd1dd6-50c1-475f-b89a-f1a6726a80cf % zip -r ttest.zip ./*
```

2-1、控制台更新方法

进入函数代码详情页，选择"上传函数zip包" -> "选择文件" -> "开始上传"



2-2、使用OpenAPI更新函数代码，需要将函数代码zip包转换成base64码，然后参考[更新函数代码](#)进行更新。

使用定时触发器根据天气信息控制远程设备

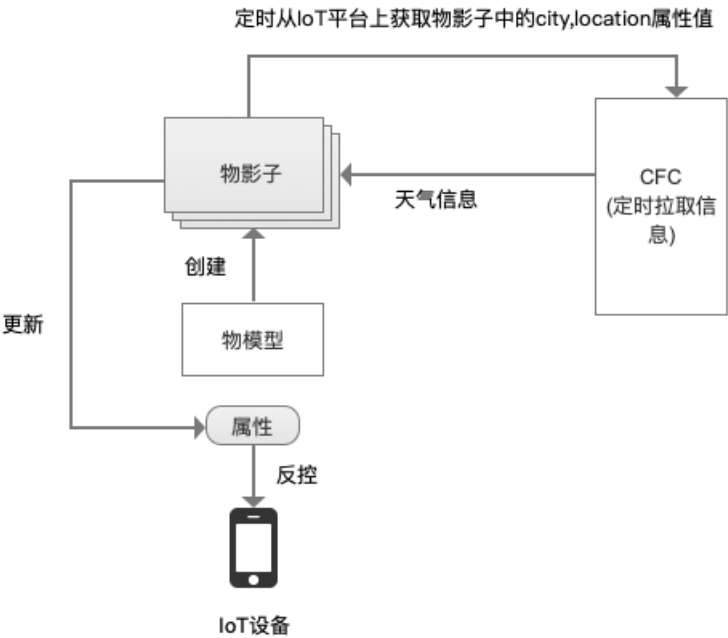
概述

CFC函数计算定时读取物联网智能家电设备上报到IoT平台的最新数据，调用天气API获取指定位置的天气信息、空气质量、紫外线等指数，智能控制远程设备，如根据温度指数，自动开启空调；根据PM2.5指数，自动开启、调节空气净化器。IoT平台结合函数计算，能快速有效地控制远程物联网设备，大大降低开发成本。

适用场景

- 智能家居、定时获取天气/环境信息
- 天气、环境污染预警
- 自动化生产、农业灌溉等

方案架构



- 1、智能家电设备通过控制台或API接口接入IoT物接入平台
- 2、函数计算定时从IoT平台上读取远程设备上报的数据，提取出城市和经纬度信息
- 3、函数计算根据城市和经纬度信息调用云市场的天气API(或其它业务API)
- 4、函数计算将处理后的数据推送给远程设备，实现定时控制远程设备

🔗 方案优势

- IoT物接入平台快速实现端云一体，支持一站式设备管理服务
- IoT物接入支持原生MQTT协议，开放高效，支持设备安全认证，提供权限管理能力以保障设备安全
- 接入函数计算，实现定时根据设备信息智能化控制远程设备

🔗 环境准备

- 配置MQTT的应用客户端，可以快速验证是否可以实现与物接入服务交流发送或者接收消息。登录[MQTT.fx官网](#)，找到合适的版本下载并安装MQTT.fx客户端。
- 安装Python3及以上运行环境

🔗 方案流程

🔗 创建CFC函数

开通CFC服务

如果您还没有开通CFC函数，请参考[函数计算概述](#)进行开通。
如果您已经开通CFC服务，可以跳过此步骤。

创建模版函数

- 1、登录百度智能云平台[官网](#)
- 2、登录成功后，选择"产品服务">"函数计算 CFC"，进入"函数列表"页面，点击从模版创建函数，选择IoT-crontab-cfc模版，进入创建函数界面

选择模板

创建方式：

空白函数

从模板创建函数

蓝图搜索：

按标签和属性筛选，或者按关键字搜索

IoT-crontab-cfc

语言：Python 3.6

描述：CFC根据IoT设备信息，定时根据天气信息智能地控制远程设备的开关，如当温...

标签：IoT,crontab,python3

部署：2

3、在弹框中输入函数名称、环境变量(环境变量包括appcode、broker、port、username、password、topic等信息)，点击确定

基本信息

* 函数名称：

iotl

只能由数字、字母、-或_组成，长度小于65个字符

* 事件模式：

event

* 运行时：

python3

* 内存(MB)：

128

* 超时时间(秒)：

30

秒（最大限额300秒）

日志存储：

无

4、点击确定后，则成功创建一个CFC函数，可以在函数详情页选择函数代码即可查看新建函数代码

函数代码

测试

编辑类型：

在线编辑

上传函数.ZIP包

从BOS上传.ZIP

当前代码：

1'''

2本项目用于模拟定时从百度智能云上获取IoT设备的属性信息，根据该信息中的city和location属性，

3调用云市场上的天气API，然后根据温度智能地控制远程IoT设备的开关（假设远程设备为空调）。

4注意：创建函数后需要为其配置crontab定时触发器

5创建CFC函数后，需要在环境变量中配置以下信息

6appcode:云市场天气API的appcode

7broker,port:物接入默认的连接到百度智能云接入的endpoint和port

8username:物影子连接信息中的username

9password:物影子连接信息中的password

10topic:物影子信息发布/订阅topic

11'''

12

13import paho.mqtt.client as mqtt

14import json

15import ssl

16import urllib.request as urllib2

保存

重置

此函数可能包含外部库。点击下载完整 ZIP 程序包

创建IoT物接入项目

开通物接入IoT Hub服务

如果您还没有开通IoT物接入服务，请参考IoT Core开通服务。如果您已经开通IoT Hub服务，请跳过此步骤。

创建项目

1、IoT Hub服务开通后，选择区域进入项目列表，然后点击创建项目

219

📍 华南 – 广州

项目列表

0 条

本月计费消息条数

0 分钟

本月链接时长数

[+ 创建项目](#) 设备型项目目前最多允许创建1个

2、设置项目名称，选择项目类型(示例中选择设备型)，点击提交

配置信息

当前地域：

华南 – 广州

?

* 项目名称：

iot_weather

描述：

定时根据天气信息控制远程设备

🔔 温馨提示：

请谨慎选择项目类型，选择后暂不支持修改。[如何选择>](#)

项目类型：

设备型 推荐

适用场景：以物影子作为设备在云端的映像，适用基于设备的物联网场景，帮助开发者聚焦业务。

特性概述：提供设备模型构建工具，快速建立以物影子为核心的物联网应用
无需关心协议细节，无缝对接时序数据库TSDB、物可视等产品
支持设备在线状态、权限、反控及OTA远程升级等丰富特性

数据型

适用场景：无设备概念或深度依赖数据流的场景，需使用者有较强的软硬件开发能力

特性概述：支持自定义 Topic，需对协议有较好了解
需开发者搭配规则引擎或自行处理数据流转及存储

提交

取消

3、创建成功后，在项目列表中可以看到物接入默认提供的三类地址，选择不同的地址，意味着您可以通过不同的方式连接到百度智能云物接入，具体请参考[物接入创建项目](#)

[+ 创建项目](#) 设备型项目目前最多允许创建1个

项目名称/Endpoint	类型	描述	地址
iot_weather	设备型	定时根据天气信息控制远程设备 🔗	tcp. ot.gz.baidubce.com:1883 ssl gz.baidubce.com:1884 ws gz.baidubce.com:443

获取连接信息

成功创建物接入项目后，点击项目名称，进入配置物接入设备型项目页面，创建物影子并获取连接信息，具体操作步骤如下：

1、创建物模型：点击项目名称进入后，选择"物模型"，进入物模型列表页面，点击"新建物模型"。填写名称、属性等信息后，点击创建即可。

220

* 名称:

iot_weather

描述:

定时根据天气信息控制远程设备

* OTA远程升级:

OFF

* 属性:

属性名称	显示名称	类型	默认值	单位	操作
city	城市	string			Delete
location	经纬度	string			Delete
temperature	温度	number		摄氏度	Delete
humidity	湿度	number			Delete
wind	风力	number			Delete
pm25	pm2.5	number			Delete
switch	开关	string	off		Delete

添加属性

创建

取消

2、创建物影子：左侧选择"物影子"，进入物影子列表页面，点击"新建物影子"。输入名称，并选择需要的物模型（这里我们选择了刚才创建的iot_weather），点击创建即可完成，此时会弹出连接信息。

名称

创建物影子

* 名称:

test_weather

描述:

根据天气信息控制远程设备

* 选择物模型:

iot_weather

模型详情

存储配置:

OFF

3、获取连接信息：物影子创建完成时，会弹出连接信息，建议下载保存备用。

提示

👌创建成功！请将连接信息配置到SDK中，实现设备与云端连接。🔗[如何连接](#)
为了安全考虑，请合理保管以上密钥，密钥丢失无法找回，只能在配置中重新生成。

```
TCP Address: [redacted]mqtt.iot.gz.baidubce.com:1883
SSL Address: [redacted]mqtt.iot.gz.baidubce.com:1884
WSS Address: [redacted]mqtt.iot.gz.baidubce.com:443
name: [redacted]
key: [redacted]
```

关闭

下载

4、查看(Topic)列表：MQTT 消息的收发是基于主题的。设备型项目的主题是由系统定义的，可点击相应影子的卡片，通过以下页面查看，可根据实际需求选择主题。每个消息均需为 json 格式。

<

test_weather

物影子详情

物详情

交互

HTTP/HTTPS

物接入提供Restful API对物影子的操作。详情请参见[帮助文档](#)

IoT Edge SDK

天工平台提供IoT Edge SDK，帮助用户更简单的在设备端对物影子进行操作。详情请参见[帮助文档](#)

通用 MQTT SDK

物接入兼容通用MQTT的SDK，系统提供默认topic对物影子进行操作。详情请参见[帮助文档](#)

1、\$baidu/iot/shadow/test_weather/update

复制

设备向该主题发布消息，可更新物影子。
订阅主题**\$baidu/iot/shadow/test_weather/update/accepted** 获取物影子更新成功后的结果。
订阅主题**\$baidu/iot/shadow/test_weather/update/rejected** 获取物影子更新失败的信息 名称

2、\$baidu/iot/shadow/test_weather/get

复制

向该主题发布消息，可获取该设备的物影子。
订阅主题**\$baidu/iot/shadow/test_weather/get/accepted** 获取物影子获取成功后的结果。
订阅主题**\$baidu/iot/shadow/test_weather/get/rejected** 获取物影子获取失败的信息。

3、\$baidu/iot/shadow/test_weather/delta

复制

当物影子中的"desired"字段有更新时，订阅主题**\$baidu/iot/shadow/test_weather/delta** 可收到"reported"和"desired"之间的差异。

🔗 修改CFC函数的环境变量

购买天气API

1、访问百度云云市场(<https://cloud.baidu.com/market/>),搜索"天气"



2、本示例中使用"全国天气预报", 找到该记录，点击进入



3、点击购买

规格：

普通版

调用次数：

1000次

10000次

20000次

50000次

100000次

价格：¥0元

温馨提示：0 元商品每账号仅限购买一次

使用期限：自购买日起有效期一年

☒ 同意《云市场服务协议》和《商品服务协议》

立即购买

4、完成开通后，返回云市场，点击管理控制台，在已开通的服务中找到“云市场”，点击进入

已开通服务



5、在"云市场">"已购买的服务列表"中可以看到全国天气预报服务，点击管理查看服务详情

已购服务列表

全部	到期时间: 2019-03-17 - 2040-03-17	请输入商品名或商品补充信息搜索								
订单号	商品	规格	数量	状态	订单金额	现金支付	下单时间 ↓	到期时间 ↓	交付方式	操作
ba529dc870fe4728a	全国天气预报	普通版	1	● 已完成	¥ 0.00	¥ 0.00	2020-03-09 14:09:25	2021-03-09 14:09:25	API交付	管理

6、在"全国天气预报"服务详情中，点击交付信息右侧的"显示密钥"查看AppCode

交付信息

[显示密钥](#)

交付方式：API交付 [API 交付商品使用指南](#)

AccessKey:

AppCode: ***** [重置](#)

7、进入函数详情页，点击环境变量右侧的"编辑"，更新环境变量(环境变量也可以在创建函数时一并添加)，添加内容如下：

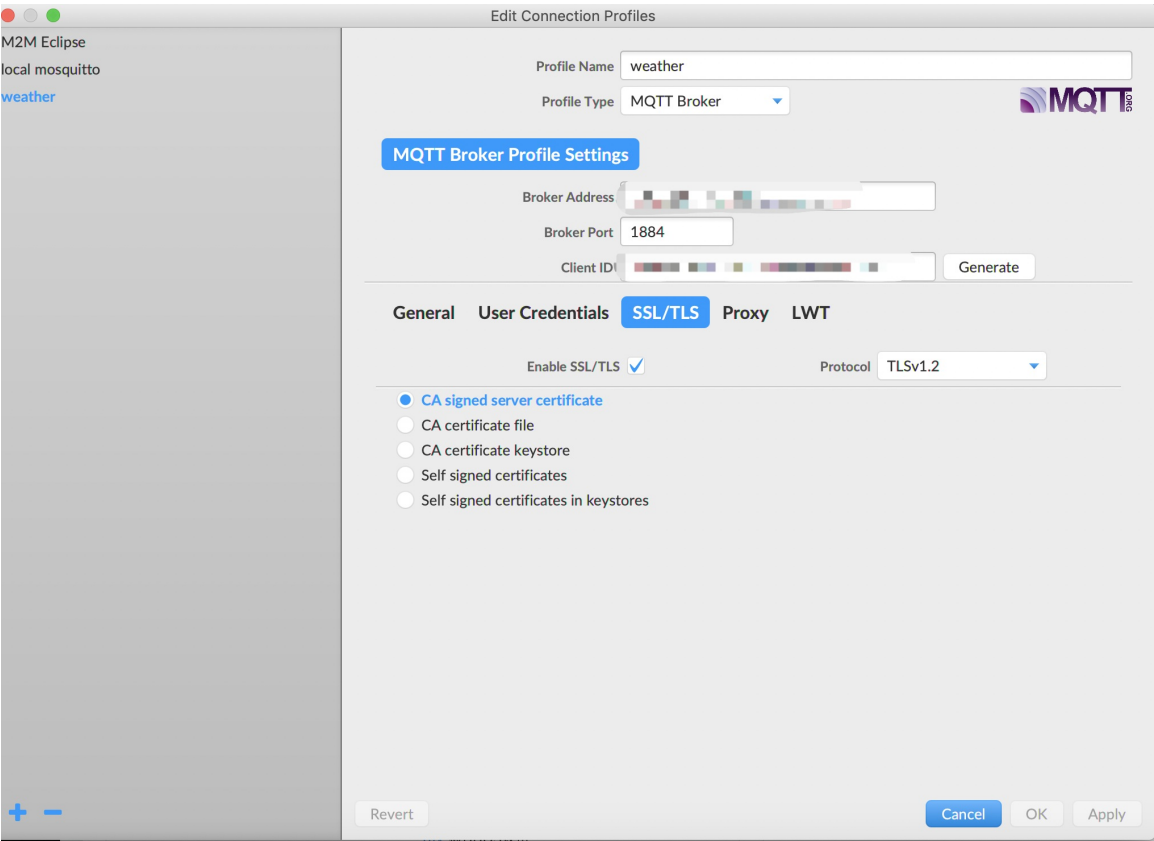
环境变量:

appcode		×
broker		×
port		×
username		×
password		×
topic		×

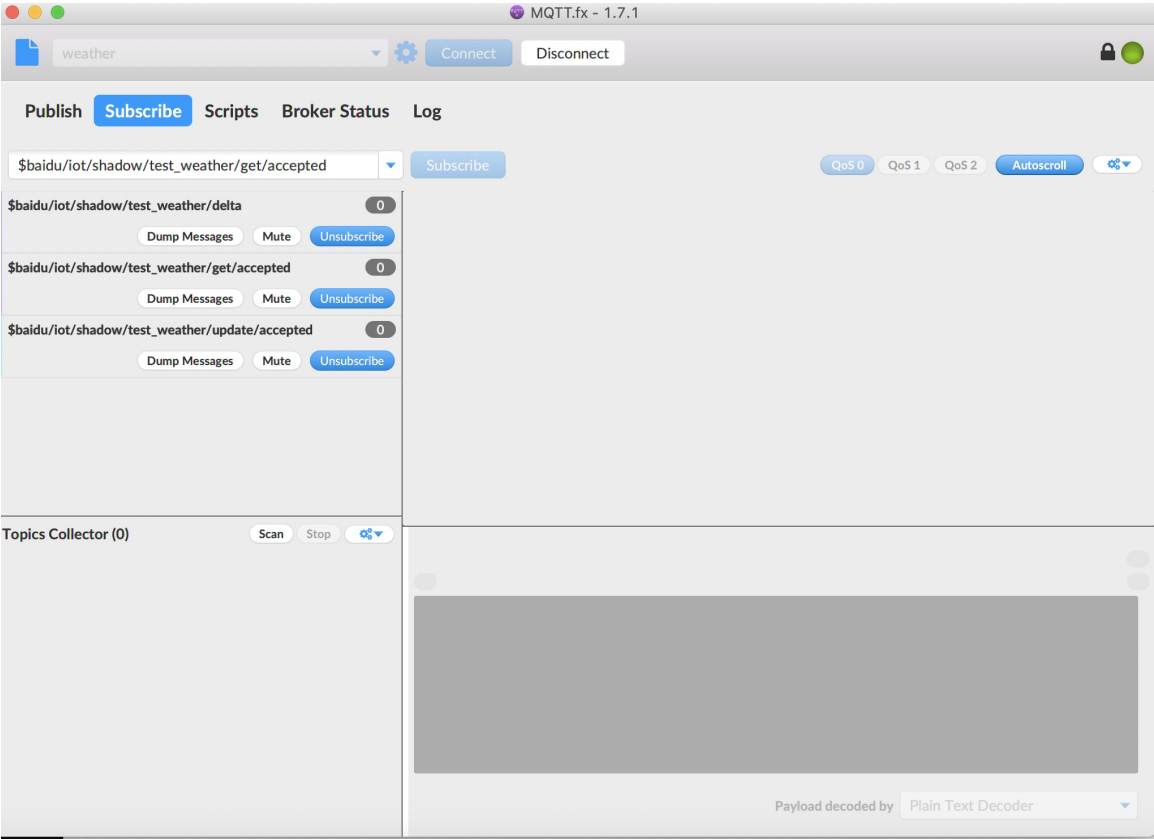
+ 添加环境变量

测试函数

- 1、为了验证CFC函数是否可以实现与物接入服务交流发送或者接收消息，您需要登录[MQTT.fx官网](#)，找到合适的版本下载并安装MQTT.fx客户端。
- 2、打开并配置MQTT客户端，具体请参考[测试MQTT客户端连接](#)



- 3、点击Connect创建连接，在IoT平台物影子交互页面复制\$baidu/iot/shadow/test_weather/update/accepted、\$baidu/iot/shadow/test_weather/delta、\$baidu/iot/shadow/test_weather/get/accepted并添加订阅



4、进入CFC函数详情页，点击测试执行，可以查看函数运行成功

测试事件

✓ 执行结果：成功

此区域下方显示你的函数执行的结果。

null

此区域下方显示你的函数执行过程中产生的日志。

START RequestId: 54716311-1335-4579-b606-36a521b22b62 Version: \$LATEST

Connecting to broker: 5x0yg10.mqtt.iot.bj.baidubce.com

Connected. Client id is: test_weather_9f1296cb-4791-4ec6-814d-630512fc5138

MQTT message received: {"requestId":"test_weather_7676c8a2-b9c0-4ea8-9374-38b623528f83","reported":{"temperature":11,"humidity":35,"city":"北京","wind":2,"pm25":33,"switch":"off","location":"39.983424,116.322987"},"desired":{"wind":3,"humidity":20,"temperature":20,"switch":"on"},"lastUpdatedTime":{"reported":{"temperature":1584356818828,"humidity":1584356818828,"city":1584356818828,"wind":1584356818828,"pm25":1584356818828,"switch":1584326552628,"location":1584342352992},"desired":{"wind":1584013053279,"humidity":158401346689,"temperature":1584326533720,"switch":1584326615316}},"profileVersion":142}

True

END RequestId: 54716311-1335-4579-b606-36a521b22b62

REPORT RequestId: 54716311-1335-4579-b606-36a521b22b62 Duration: 10.938s Billed Duration: 11s Max Memory Used: 13.2M

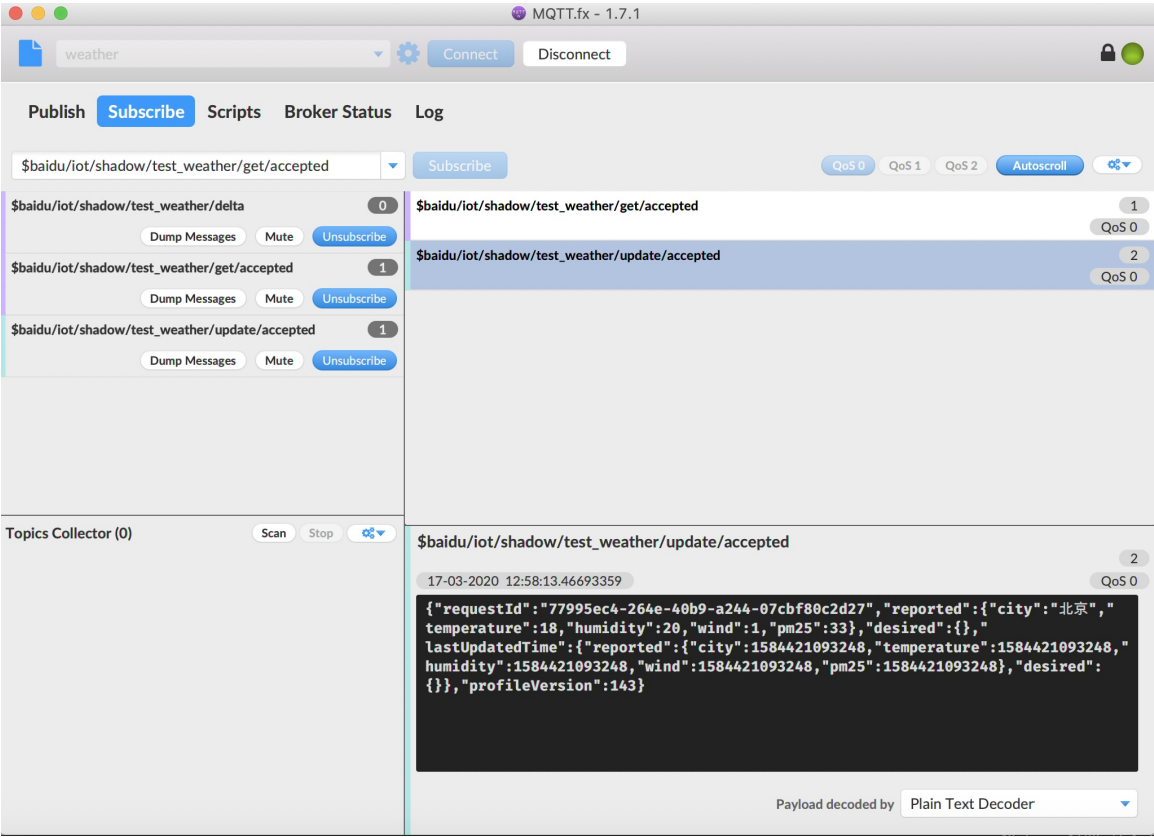
创建

执行

取消

5、查看MQTT客户端，可以看到订阅Topic上已经收到消息

227



5、同时进入IoT物接入平台，查看项目物影子详情，可以看到模型数据已经被更新

影子状态：

模型数据原始数据

编辑刷新清空物影子

属性名称	显示名称	类型	默认值	单位	当前值	修改时间	期望值	发送时间
temperature	温度	number		摄氏度	18	2020-03-17 12:58:13	20	2020-03-16 10:42:13
humidity	湿度	number			20	2020-03-17 12:58:13	20	2020-03-12 19:42:26
wind	风力	number			1	2020-03-17 12:58:13	3	2020-03-12 19:37:33
pm25	pm2.5	number			33	2020-03-17 12:58:13	N/A	
city	城市	string			北京	2020-03-17 12:58:13	N/A	
switch	开关	string	off		off	2020-03-16 10:42:32	on	2020-03-16 10:43:35
location	经纬度	string			39.983424, 116.322987	2020-03-16 15:05:52	N/A	

至此CFC函数能够从IoT平台上读取设备信息，调用天气API，根据天气信息处理数据，最后将数据上报给IoT设备

6、最后需要为CFC函数配置定时触发器触发器，本示例设置每天上午10点触发函数执行，CFC定时触发器的配置请参考[CFC定时触发器](#)

新增触发器

触发器：

定时触发器

* 规则名称

iot_weather

1. 只能包含字母，数字、下划线和中划线
2. 长度限制在1-30之间

* 计划表达式

cron(0 10 * * ?)

定时触发器以UTC时间运行，即北京时间减去8个小时
详细请点击 [表达式详解](#)

输入配置

☒ 默认事件

☐ 常量

☒ 启动触发器

确定

取消

7、点击创建后，可以在函数详情触发器列表中可以看到新创建的定时触发器

触发器

+ 新增触发器

定时触发器

ON

规则名称：iot_weather

计划表达式：cron(0 10 * * ?)

编辑

更多参考

百度函数计算CFC开发文档：<https://cloud.baidu.com/doc/CFC/index.html>

使用 Terraform 快速搭建图像处理应用

概述

本文介绍了使用百度云函数计算，结合 BOS 和百度云图像效果增强能力，实现图像文件的自动处理，并通过 Terraform 实现应用的快速部署。

前提概念

首先介绍在本文出现的几个比较重要的概念：

Terraform：是一种安全有效地构建、更改和版本控制基础设施的工具。它以配置文件为驱动，您在文件中定义各种组件，Terraform 会基于该文件完成所有组件的创建、变更和管理。目前百度云已与 Terraform 集成，您可通过 Terraform 来创

建、管理 CFC、BOS、VPC、BCC 等多种百度云资源，具体请参考 [Terraform BaiduCloud Provider](#)。

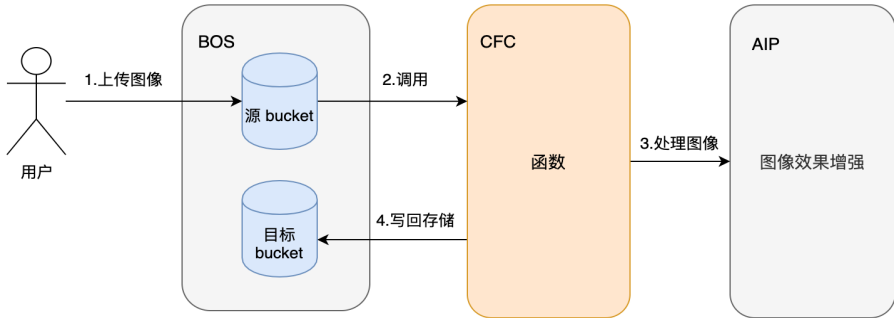
百度云图像效果增强：基于领先的深度学习技术，提供对图片进行去雾、对比度增强、无损放大、拉伸恢复等多种优化处理，具体请参考[图像效果增强](#)。

BOS：百度云提供的对象存储服务，支持任何类型的数据存储。Bucket 是存放数据的容器，可看做是存储桶。

BOS 触发器：BOS 和 CFC 的无缝集成。您可以为各种类型的事件设置处理函数，当 BOS 监控到指定类型的文件、事件后，会自动调用 CFC 函数。

🔗 工作原理

整体方案如图所示：

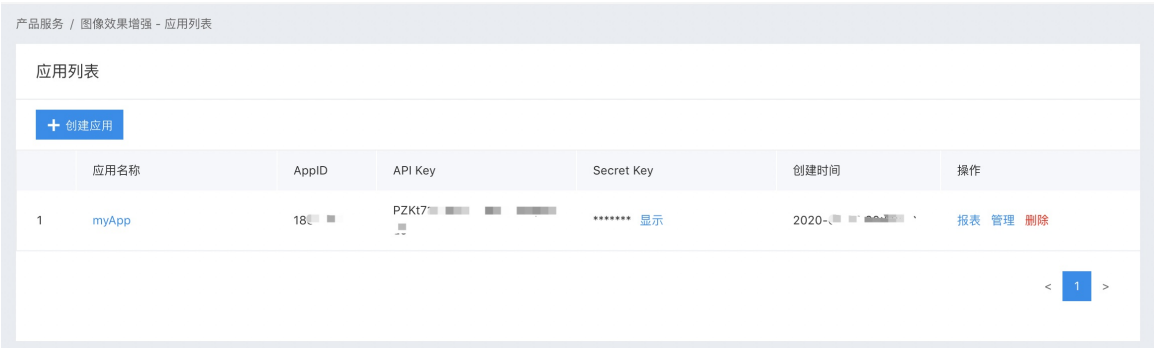


用户上传一个图像文件到 BOS 后，BOS 触发器自动触发函数执行，函数调用 AIP 的图像效果增强服务处理图像，并且将处理后的视频保存回 BOS 的另一个 Bucket。本示例中，函数使用了图像效果增强服务的给黑白图像上色功能。

🔗 操作部署

🔗 创建图像效果增强应用

登录百度云控制台，选择“产品服务”->“人工智能”->“图像效果增强”，点击“创建应用”，输入“应用名称”和“应用描述”后点击“立即创建”，您的第一个应用就创建完成了。返回“应用列表”页，就能看到您创建的应用。如图所示，每个应用都有自己的 AppID、API Key 和 Secret Key，在函数中访问该服务会用到这些信息。



🔗 部署函数和 BOS Bucket

函数和 Bucket 资源的部署有两种方式，下面分别介绍一下。

通过 Terraform 一键部署

使用 Terraform 可以统一管理各个资源，使用 Terraform 可帮助您更快的部署、变更应用，无需在各个产品的控制台上分别操作，降低操作成本。

1. 在本地编写函数代码，实现从 BOS 读取图片、调用图像增强服务的逻辑。本例中，您可直接使用此示例代码包 [aip-image-process.zip](#)。
2. 登录 [Terraform 官网](#) 下载适用于您的操作系统的程序包。
3. 编辑 Terraform 配置文件。新建文件 main.tf，在文件中指定函数、bucket、函数触发器的元信息。参考配置如下：

```

    provider "baiducloud" {
      access_key = "<YOUR_BAIDUCLOUD_ACCESS_KEY>"
      secret_key = "<YOUR_BAIDUCLOUD_SECRET_KEY>"
      region = "bj"
    }

#### 创建源 bucket
resource "baiducloud_bos_bucket" "source" {
  bucket = "source-bucket"
  acl    = "public-read-write"
}

#### 创建目标 bucket
resource "baiducloud_bos_bucket" "dest" {
  bucket = "destination-bucket"
  acl    = "public-read-write"
}

data "baiducloud_zones" "default" {}

#### 创建函数，关于函数的更多信息可查看 https://cloud.baidu.com/doc/CFC/s/xjwvz450q
resource "baiducloud_cfc_function" "default" {
  function_name = "image-process"
  description   = "my function to process images"

  # 指定函数的环境变量，示例函数执行时会读取这些变量以初始化相关 client
  # 将您创建的图像效果增强应用的参数填入
  environment = {
    "APP_ID" : "<YOUR_AIP_ID>"
    "AIP_KEY" : "<YOUR_AIP_KEY>"
    "AIP_SECRET_KEY" : "<YOUR_AIP_SECRET_KEY>"
    "DEST_BUCKET" : "destination-bucket"
  }
  handler      = "index.handler"
  memory_size  = 128
  runtime      = "nodejs12"
  time_out     = 10

  # 函数 zip 包的相对或绝对路径，本示例中请填写您刚下载的 aip-image-process.zip 的路径
  code_file_name = "</PATH/OF/FUNCTION>"

  # 配置函数日志存储地址
  log_type      = "bos"
  log_bos_dir   = baiducloud_bos_bucket.dest.bucket
}

#### 创建 bos 触发器
resource "baiducloud_cfc_trigger" "bos-trigger" {
  source_type = "bos"
  bucket      = baiducloud_bos_bucket.source.bucket
  target      = baiducloud_cfc_function.default.function_brn
  name        = "myBosTrigger"
  status      = "enabled"

  # 当 /images/ 目录下有文件上传时，会触发函数执行
  bos_event_type = ["PutObject", "PostObject"]
  resource       = "/images/*"
}

```

您可再创建一个可选的文件 output.tf，Terraform 成功创建资源后会根据此文件打印资源信息，示例如下：

```
output "cfc_functions" {
  value = "${baiducloud_cfc_function.default}"
}

output "bos_trigger" {
  value = "${baiducloud_cfc_trigger.bos-trigger}"
}
```

4. 执行命令 `terraform init`，Terraform 会根据当前配置文件初始化工作空间。然后执行 `terraform apply`，根据提示输入"yes"，即自动执行资源创建。执行结束后，您可到百度云控制台查看新创建的函数、触发器和 Bucket。

手动部署 您也可以手动分别创建函数、触发器和 Bucket，具体操作如下：

1. CFC 有一个名为 `aip-image-process` 的函数模板，您可使用该模板生成您自己的函数，具体操作请参考 [使用模板创建函数](#)。
2. 创建 2 个 Bucket，`source-bucket` 用于接收上传的图像源文件，`destination-bucket` 用于存储 CFC 函数处理后的图像文件，具体操作请参考 [创建 Bucket](#)。
3. 为函数创建 BOS 触发器，选择刚刚创建的 `source-bucket` 作为触发源 Bucket，并设置触发器的前缀为 `images/`，事件类型选择“文件上传”和“表单上传”两项。设置完成后，当图像文件上传到该 Bucket 的 `images/` 路径下后，BOS 就会自动调用函数。具体操作方法请参考 [创建 BOS 触发器](#)。

功能验证

1. 在 BOS 的控制台，上传黑白图片到 `source-bucket` 的 `images/` 目录下，本例使用下图：



2. 随后进入 `destination-bucket`，在 `images/` 目录下即可看到同名的、已处理好的图像文件。查看该文件，显示如下：



3. 在 CFC 控制台查看函数的日志、监控，可看到函数被调用了一次。

总结

基于函数计算 CFC 搭建图像处理应用，您无需管理服务器等基础设施，只需关注图像处理等业务逻辑代码。借助 Terraform，您可以一键搭建函数运行环境，快速配置各种触发器、对象存储 BOS、虚拟机 BCC 等多种资源，实现应用的快速上线。

使用 CFC HTTP 触发器托管 API

概述

在我们进行 Web API 的相关开发过程中，偶尔需要对系统进行 API 的增加或者迁移，在系统逻辑较为复杂的情况下，进行 API 的

扩充或者迁移往往工作量较大。此时，使用 CFC 以及百度云 TableStorage 则可以简单地实现 API 的托管，从而减少系统的负载，同时增加接口的可移植性。

适用场景

- Web API 托管
- 插件式 API

方案优势

- 托管 API 至 CFC，可节约资源
- 增加 API 的可移植性
- 减少 API 之间的耦合度

方案流程

一、创建 Table

1.1 开通百度云 TableStorage 服务 若您还未开通百度云 TableStorage 服务，则可以通过【百度云控制台】>【数据库】>【TableStorage】开通相关服务。**1.2 创建实例** 开通 TableStorage 服务后，您可以由控制台进入 TableStorage 中，并且创建对应的实例。如图所示。更多实例相关操作详情可参考[TableStorage 实例操作](#)

创建实例

×

实例创建需1分钟左右，创建后请刷新实例列表

实例名称：

test_cfd

- 以字母或者下划线开头，支持数字、字母和下划线
- 不包含baidu、bidu、bce、bts等单词，长度3-255

表格默认存储类型：

高性能型

▼

>>了解存储类型特点及定价

确认

取消

1.3 创建表 创建 TableStorage 实例后，您可以进入对应实例创建对应表。详细如下图所示。详细的创建操作可参考[TableStorage 表操作](#)

创建表格

×

表格数据表创建需1分钟左右，创建后请刷新表格列表

表格名称：

test_cfd

名称需以字母或者下划线开头，支持数字、字母和下划线，长度1-255

高级设置：

开启

默认

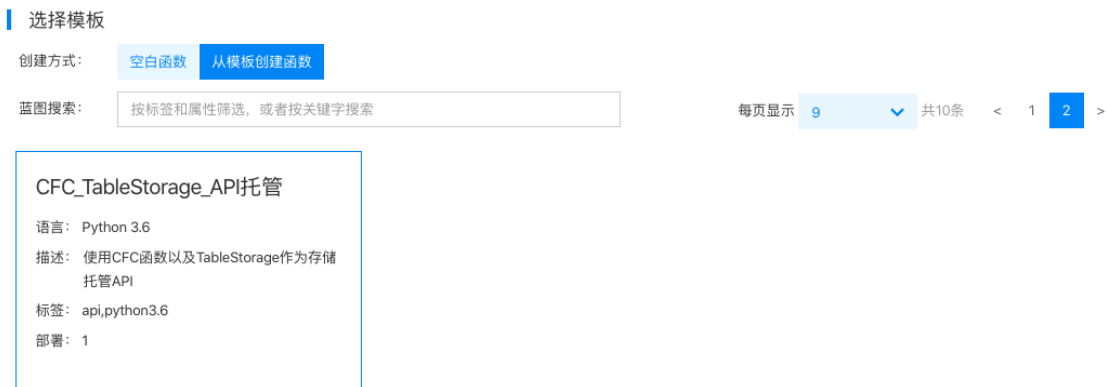
确认

取消

至此，TableStorage 已经准备就绪。

二、创建 CFC 函数

2.1 开通百度云 CFC 服务 若您还未开通百度云 TableStorage 服务，则可以通过【百度云控制台】>【计算】>【函数计算】开通相关服务。**2.2 创建函数** 由于暂时 TableStorage 服务仅支持广州、保定地区。故需要您手动选择 CFC 广州地区，并以 CFC_TableStorag_API为蓝图创建函数。如下所示：



2.3 部署相关代码 根据蓝图中代码相关提示信息补全对应实例名称、表名称以及相关字段名称等。并根据 API 需要修改其返回格式。

三、创建 HTTP 触发器进行 API 托管

3.1 创建 HTTP 触发器 在对应函数中创建 HTTP 触发器，并根据自己需要填写路径以及 HTTP 方法。具体创建流程可参考[HTTP 触发器开发参考](#) **3.2 配置函数路径** 根据蓝图代码注释配置相关的函数route信息。

四、简单实践

4.1 场景 若需要托管一个成绩查询 API 于 CFC 函数，其中学生成绩表格如下所示：

姓名	语文	数学	英语	物理	化学	生物
小明	123	123	123	123	123	123

则对应的 TableStorage 表格如下所示：

☐ 行键	biology	chemistry	chinese	english	math	physics
☐ xiaohong	35	100	99	1	123	2

4.2 API 设计

API 设计如下

```

#### 获取对应学生成绩单
GET /students?instance={instance_name}&table={table_name}&name={name}

#### 上传/修改学生成绩单
POST/PUT /students
{
    "instance": "instance_name",
    "table": "table_name",
    "cells": {
        "chinese": "100",
        "math": "100",
        "english": "100",
        "physics": "100",
        "chemistry": "100",
        "biology": "100"
    }
}

#### 删除学生信息
DELETE /students?instance={instance_name}&table={table_name}&name={name}

```

则对应方法如下所示：

```

@app.route('/students', method='GET')
def get_row():
    """
    @param: request: bottle request
    @return: http return

    本方法用于从 TableStorage 中获取数据
    """
    instance = bytes(request.query.name, encoding='utf8')
    table = bytes(request.query.name, encoding='utf8')
    name = bytes(request.query.name, encoding='utf8')

    bts = init_bts_client()
    query_row_arg = QueryRowArgs()
    query_row_arg.rowkey = name
    try:
        res = bts.get_row(instance, table, query_row_arg)
        if res is None:
            return HTTPResponse(body={'error': 'resource not found'}, status=404)
        else:
            return HTTPResponse(body={'body': str(res.result[0])}, status_code=200)
    except Exception as exception:
        return HTTPResponse(body={'error': exception}, status=400)

@app.route('/students', method=['POST', 'PUT'])
def put_row():
    """
    @param: bottle request
    @return: http response

    本方法用于将输入数据存储于 TableStorage 中
    """
    bts = init_bts_client()
    body = request.json
    instance = bytes(body['instance'], encoding='utf8')
    table = bytes(body['table'], encoding='utf8')
    name = bytes(body['name'], encoding='utf8')
    c = body['cells']
    cells = []

```

```

for key, value in c.items():
    cells.append(Cell(key, value).__dict__)

row = Row()
row.rowkey = name
row.cells = cells
try:
    res = bts.put_row(instance, table, row)
    return HTTPResponse(status_code=200)
except Exception as exception:
    return HTTPResponse(status_code=400, body={'error': exception})

@app.route('/students', method='DELETE')
def delete_row():
    """
    @param: bottle request
    @return: http response

    本方法用于从百度云TableStorage中删除对应数据
    """
    bts = init_bts_client()
    instance = bytes(request.query.instance, encoding="utf8")
    table = bytes(request.query.table, encoding="utf8")
    name = bytes(request.query.name, encoding="utf8")
    query_row_args = QueryRowArgs()
    query_row_args.rowkey = name

    try:
        bts.delete_row(instance, table, query_row_args)
        return HTTPResponse(status_code=204)
    except Exception as e:
        return HTTPResponse(status_code=400, body={'error': e})

```

同时，在函数界面创建 HTTP 触发器，路径为 /students，方法为 GET、POST、PUT、DELETE

4.3 实践 创建完毕触发器，部署完毕方法后，可以简单测试对应 API。

原始数据如下所示：

☐ 行键	biology	chemistry	chinese	english	math	physics
☐ xiaohong	35	100	99	1	123	2

• 上传成绩：使用curl请求上传成绩，如下所示：

```

curl --location --request POST 'https://dn99xpwe8yjgx.cfc-execute.gz.baidubce.com/students' \
--header 'Content-Type: application/json' \
--data-raw '{
  "instance": "test_cfc",
  "table": "test_cfc",
  "name": "xiaoming",
  "cells": {
    "chemistry": "100",
    "chinese": "100",
    "math": "100",
    "english": "100",
    "physics": "100",
    "biology": "100"
  }
}'

```

执行完毕后，TableStorage 表格数据如下所示，如图所示，数据已经被写入表格中。

☐ 行键	biology	chemistry	chinese	english	math	physics
☐ xiaohong	35	100	99	1	123	2
☐ xiaoming	100	100	100	100	100	100

- 查询成绩：使用cURL查询成绩，如下所示：

```
curl --location --request GET 'https://dn99xpwe8yjgx.cfc-execute.gz.baidubce.com/students?instance=test_cfc&table=test_cfc&name=xiaoming' \
--header 'Content-Type: application/json'
```

resonse如下所示：

```
{
  "body": [
    {
      "rowkey": "u:xiaoming",
      "cells": [
        {
          "column": "u:biology",
          "value": "u'100'",
          "timestamp": 1584633381567
        },
        {
          "column": "u:chemistry",
          "value": "u'100'",
          "timestamp": 1584633381567
        },
        {
          "column": "u:chinese",
          "value": "u'100'",
          "timestamp": 1584633381567
        },
        {
          "column": "u:english",
          "value": "u'100'",
          "timestamp": 1584633381567
        },
        {
          "column": "u:math",
          "value": "u'100'",
          "timestamp": 1584633381567
        },
        {
          "column": "u:physics",
          "value": "u'100'",
          "timestamp": 1584633381567
        }
      ]
    }
  ]
}
```

- 删除成绩：使用cURL删除某一学生成绩，如下所示：

```
curl --location --request DELETE 'https://dn99xpwe8yjgx.cfc-execute.gz.baidubce.com/students?instance=test_cfc&table=test_cfc&name=xiaoming' \
--header 'Content-Type: application/json'
```

执行完毕后，相关数据在 TableStorage 中已经被删除。如图所示：

写入数据

查询数据

删除数据

☐ 行键	biology	chemistry	chinese	english	math	physics
☐ xiaohong	35	100	99	1	123	2

每页展示 10 < 1 >

总结

使用 CFC HTTP 触发器以及 TableStorage 数据库，可以轻松地将 API 托管至云端实现插件化的 API，一方面可以节省计算机资源，另一方面则可以为系统升级、迁移等操作提供便利。

Java Springboot Web框架迁移到函数计算

本文将介绍如何将Spring Boot应该迁移部署到函数计算。

环境准备

安装OpenJDK 1.8.0及以上运行环境

迁移应用

如果您还没有开通CFC服务，请参考[函数计算](#)开通；如果您已开通，则可以跳过该步骤。

1. 创建一个spring boot项目
2. 引入百度云函数计算SDK

在项目pom.xml中添加百度云原生函数计算CFC（2.1.0及以上版本）的依赖。

```
<dependency>
  <groupId>com.baidu.cfc</groupId>
  <artifactId>cfc-java-core</artifactId>
  <version>2.1.1</version>
</dependency>
```

3. 代码改造

在项目的根目录src/main/java的任意目录下新增一个执行方法入口，例如在src/main/java/tacos的目录下创建MyHandler类用于实现AbstractSpringBootHandler

```
package tacos;

import com.baidubce.cfc.core.AbstractSpringBootHandler;
public class MyHandler extends AbstractSpringBootHandler{

    @Override
    public void startApp() {
        // 修改为springboot项目的入口主函数，例如: 入口为TacosCloudApplication class下的main函数
        TacosCloudApplication.main(new String[]{""});
    }
}
```

添加mainClass类，如APP类(需要修改pom.xml设置为启动类)

```
package tacos;

import com.baidubce.cfc.core.CfcClient;

public class App {
    public static void main(String[] args){
        CfcClient.main();
    }
}
```

4. maven编译并打包函数代码

如果完成上述对SpringBoot服务的改造工作，还需要对项目的打包构建工作进行一定的修改（主要是对CFC函数代码包的一些限制），主要限制如下：

- 打包上传函数的根目录只有一个mainClass对应的jar包
- mainClass对应的class应为启动CfcClient对应的class
- 打包上传的函数目录中依赖完整，所有依赖jar包都在函数压缩包中
- 在函数中配置的handler类与函数中实现的handler类一致

增加plugin配置

在 pom.xml文件中添加 maven-assembly-plugin，构建一个可执行jar包。添加maven-dependency-plugin，将jar包所依赖的其他jar包拷贝至\${project.build.directory}/lib目录。在 project 标签下增加如下的两项：

```
<build>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-dependency-plugin</artifactId>
      <version>3.1.1</version>
      <executions>
        <execution>
          <id>copy-dependencies</id>
          <phase>package</phase>
          <goals>
            <goal>copy-dependencies</goal>
          </goals>
          <configuration>
            <outputDirectory>${project.build.directory}/lib</outputDirectory>
            <overWriteReleases>false</overWriteReleases>
            <overWriteSnapshots>false</overWriteSnapshots>
            <overWriteIfNewer>true</overWriteIfNewer>
          </configuration>
        </execution>
      </executions>
    </plugin>
    <plugin>
      <!-- Build an executable JAR -->
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-jar-plugin</artifactId>
      <version>3.0.2</version>
      <configuration>
        <archive>
          <manifest>
            <addClasspath>true</addClasspath>
            <classpathPrefix>lib/</classpathPrefix>
            <mainClass>tacos.App</mainClass>
          </manifest>
        </archive>
      </configuration>
    </plugin>
  </plugins>
</build>
```

发布打包

```
#### 打包
$ mvn clean package
#### zip压缩
$ cd target && zip -r springboot.zip xxx-xxx.0.0.1-SNAPSHOT.jar lib/
```

5. 配置函数

- 创建java函数，由于springboot启动时间较长，创建函数时，函数超时时间建议设置长点

创建函数

1 选择模板 > 2 函数配置 > 3 触发器配置

基本信息

函数名称:

springboot

只能由数字、字母、-或_组成, 长度小于65个字符, 且在当前地域内唯一

服务名称:

default

管理服务

事件模式:

event

运行时:

Java 8

内存(MB):

512

768

1024

512

超时时间(秒):

-

3

+

秒 (最大限额300秒)

单实例并发度:

-

100

+

日志存储:

无

描述:

上一步

下一步

提交

- 选择"上传.zip文件", 将上一步中压缩生成的zip文件上传至百度云函数
- 处理程序中填写函数入口类名称, 本例为tacos.MyHandler

函数代码

测试

编辑类型:

在线编辑

上传函数.ZIP包

从BOS上传.ZIP

函数程序包:

+ 选择文件

开始上传

?

此函数可能包含外部库。 [点击下载完整 ZIP 程序包](#)

处理程序:

tacos.MyHandler [修改](#) ?

环境变量:

- [编辑](#)

6. 创建http触发器

新增触发器

触发器：

HTTP触发器

* URL路径：

/tacos

填写以 / 开头的匹配路径，完整 URL 在触发器创建完成后可以查看
使用{param}指定路径参数，例如 /user/{id}/posts 中，{id} 即为匹配参数
使用{param+}指定贪婪匹配路径参数，它会匹配当前位置直到 URL 结尾的
所有剩余部分，例如 /path/{filepath+}

* HTTP方法：

您已选了1项

* 身份验证：

☒ 不验证

☐ IAM

☐ 使用二进制请求body

确定

取消

7. 验证

← → ↺

7grmt1xfky5jf.cfc-execute.bj.baidubce.com/tacos

Welcome to tacos

代码示例

本章节中的代码简单示例，请点击下载[简单示例](#)。

CFC支持SSE协议

概述

CFC支持SSE协议，下文介绍如何在CFC使用SSE协议。

操作步骤

1. 登录 [函数计算控制台](#)。
2. 依次点击“新建函数 > 空白函数 > 事件类型选择web > 填写超时时间（建议10s）> 提交”。

函数计算 CFC

函数管理

计费模式和免费额度变更通知

1. CFC计费模式和免费额度将于2023年3月10日进行调整，用户开通使用CFC三个月后每月不再享受免费额度。为了感谢老客户，平台推出力度更大、种类更多的套餐包，您可以选择适合自己性价比更高的套餐包来享受更大的优惠。详见[资源包文档](#)。
2. 新用户可在开通CFC后的3个月内享受免费额度，您可以通过“0元购买套餐包”的形式享受免费额度。

+ 新建函数

函数名

服务名称

运行时

代码体积

修改时间

最后更新时间

描述

操作

☐

函数名

default

Python 3.10

1.5 KIB

2025-03-26 09:50:53

--

--

测试 删除

☐

函数名

default

Python 3.10

1.5 KIB

2025-03-25 19:00:44

--

--

测试 删除

☐

函数名

default

Python 3.10

1.5 KIB

2025-03-24 14:51:25

--

--

测试 删除

☐

函数名

default

Python 3.10

1.5 KIB

2025-03-24 14:44:57

--

--

测试 删除

☐

函数名

default

Node.js 16

610 Bytes

2025-03-21 16:54:25

--

--

测试 删除

☐

函数名

default

Python 3.10

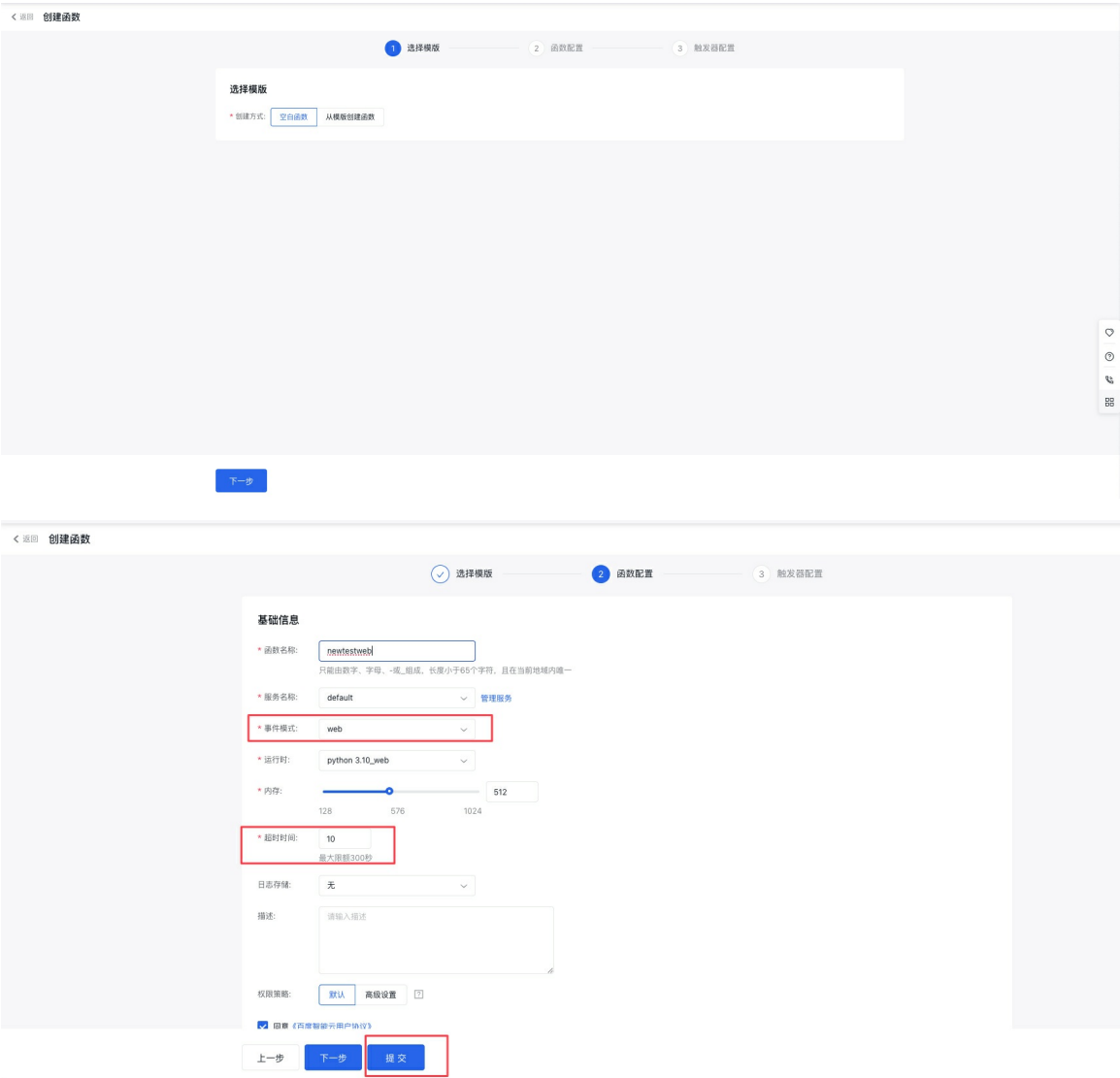
1.6 KIB

2025-03-21 14:17:23

--

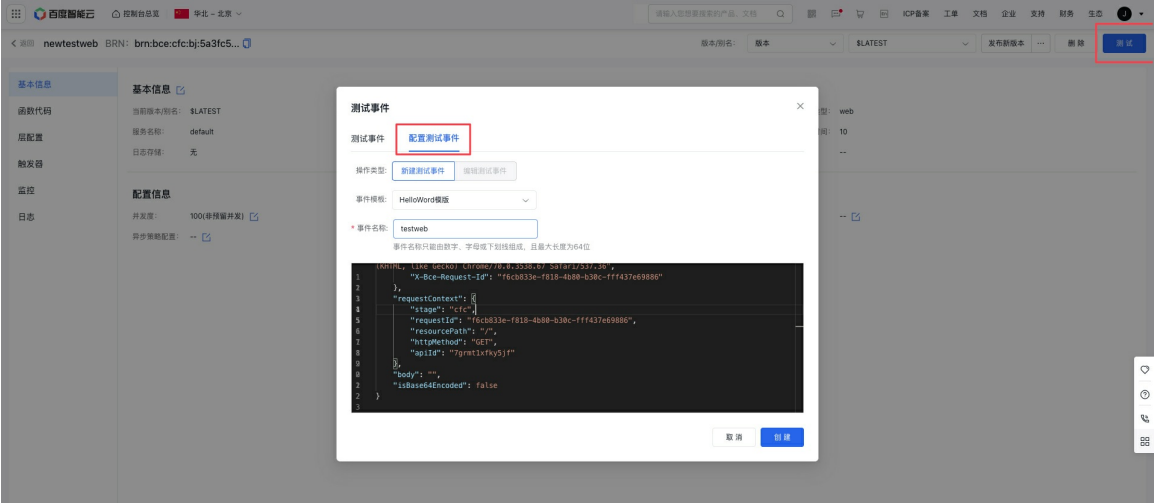
--

测试 删除

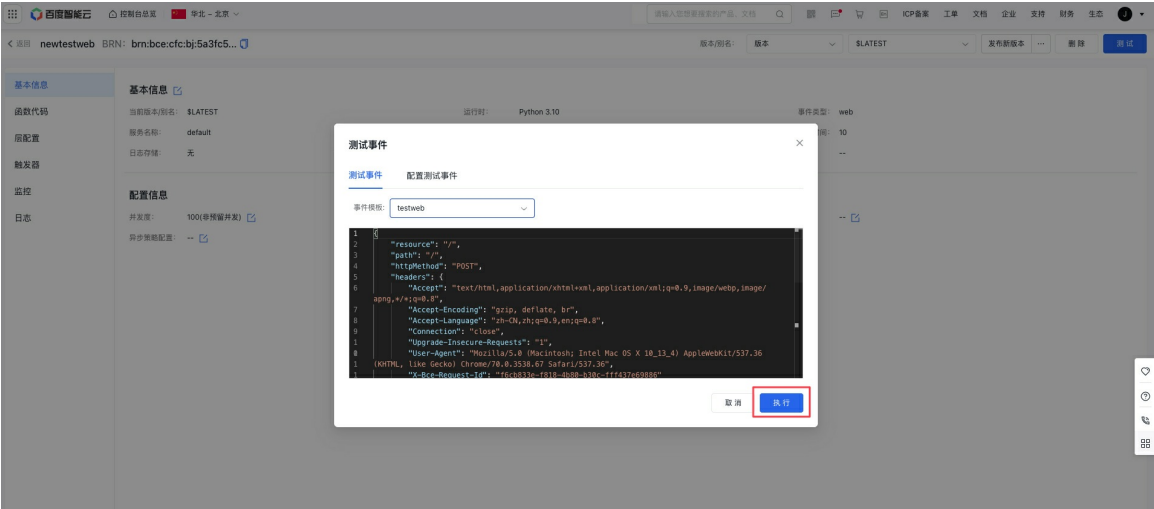


3. 在函数详情页面，点击测试，配置以下事件模版。

```
{
  "resource": "/",
  "path": "/",
  "httpMethod": "POST",
  "headers": {
    "Accept": "text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8",
    "Accept-Encoding": "gzip, deflate, br",
    "Accept-Language": "zh-CN,zh;q=0.9,en;q=0.8",
    "Connection": "close",
    "Upgrade-Insecure-Requests": "1",
    "User-Agent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_13_4) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/70.0.3538.67 Safari/537.36",
    "X-Bce-Request-Id": "f6cb833e-f818-4b80-b30c-fff437e69886"
  },
  "requestContext": {
    "stage": "cfc",
    "requestId": "f6cb833e-f818-4b80-b30c-fff437e69886",
    "resourcePath": "/",
    "httpMethod": "GET",
    "apiId": "7grmt1xfky5jf"
  },
  "body": "",
  "isBase64Encoded": false
}
```



4. 使用刚创建的事件模版，测试函数。





通过API/SDK调用

页面调用当前无法流式返回，只能调试函数(获取全部数据之后返回)。如果您有SSE流式返回的需求，可以使用API/SDK调用函数

```
macosBook:Book-Pro ~ % curl -X POST "http://cfc.bj.baidubce.com/v1/functions/antestmb/invoke?logId=0&url=http://type-tail/invoke?type=RequestResponse" \
-H 'Authorization: ' \
-H 'Content-Type: application/json;charset=UTF-8' \
-d '{
  "httpMethod": "POST",
  "requestContext": {
    "apiId": "79m1bfkfy5jf",
    "requestId": "f6cb833e-f818-4b80-b30c-fff437e69886",
    "resourcePath": "/",
    "httpMethod": "GET",
    "stage": "cfc"
  },
  "resource": "/",
  "isBase64Encoded": false,
  "headers": {
    "Accept": "text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/png,*/*;q=0.8",
    "Accept-Encoding": "gzip, deflate, br",
    "Connection": "close",
    "Upgrade-Insecure-Requests": "1",
    "User-Agent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_13_4) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/70.0.3538.67 Safari/537.36",
    "Content-Language": "zh-CN,zh;q=0.9,en;q=0.8",
    "X-Bce-Request-Id": "f6cb833e-f818-4b80-b30c-fff437e69886"
  },
  "path": "/",
  "body": ""
}
```

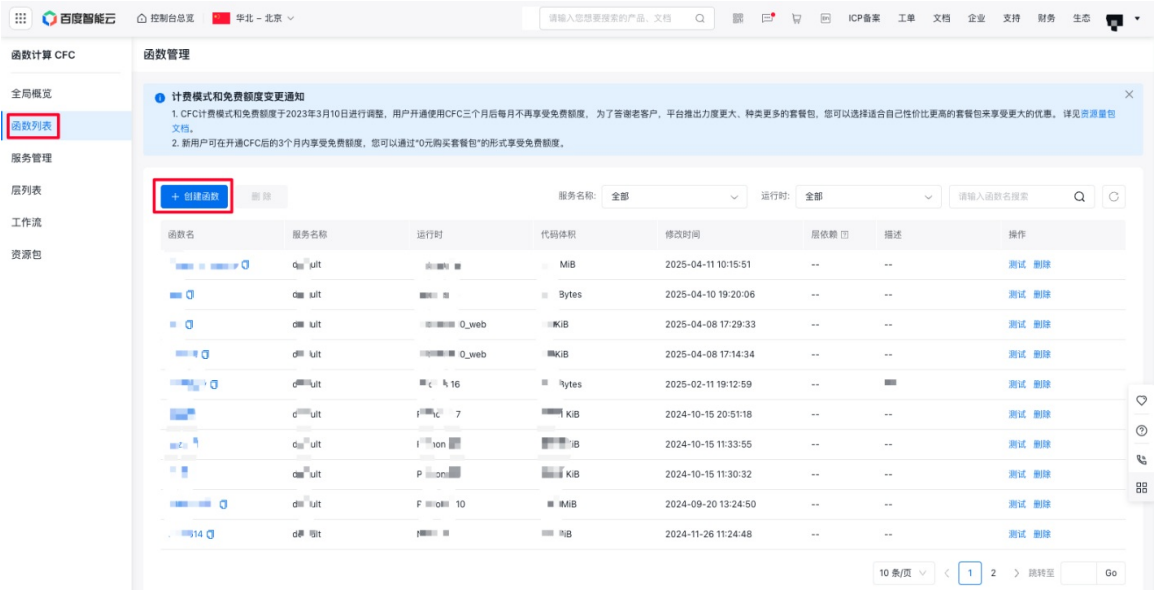
CFC支持MCP服务托管

CFC支持MCP服务托管能力，您可以通过CFC内置的MCP热门模板快速体验部署，或基于基础模版自定义部署MCP服务。当MCP服务部署完成后，平台会生成一个SSE连接地址用于测试。

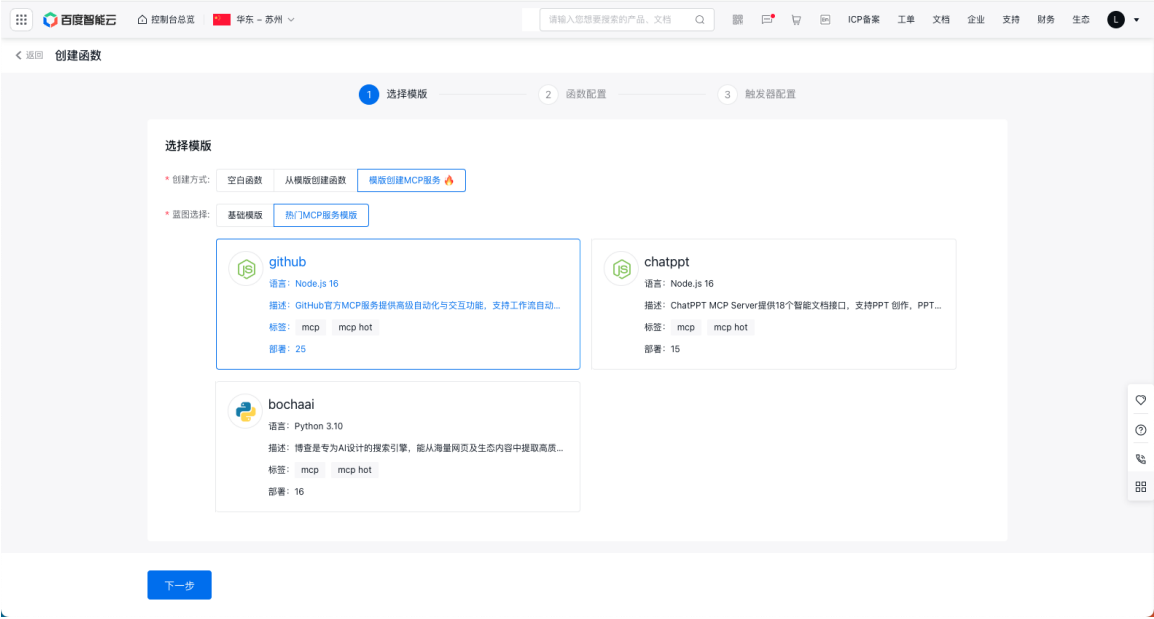
创建MCP服务

方式一：通过热门模版快速体验部署

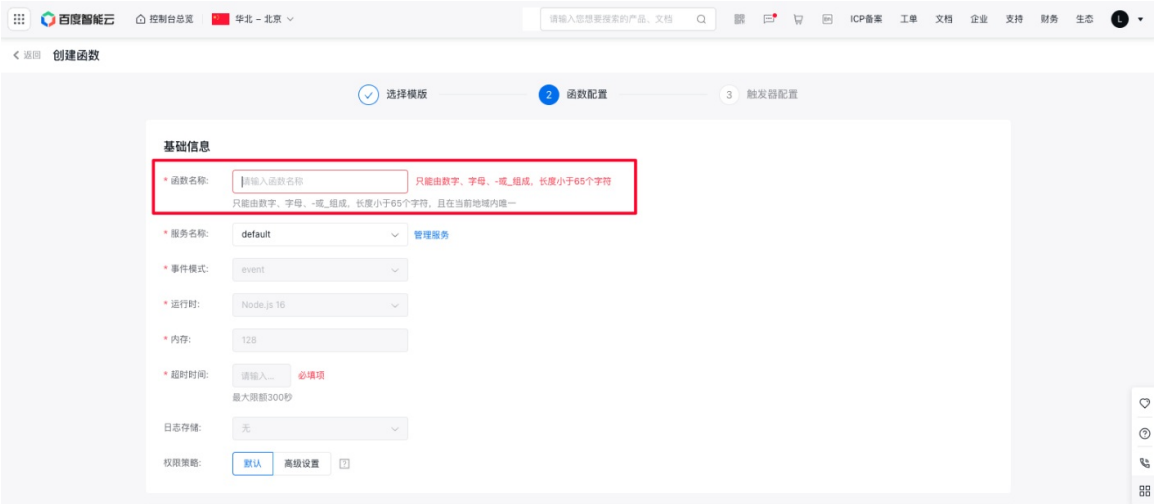
1、登录 [CFC控制台](#)，进入函数列表，点击创建函数。

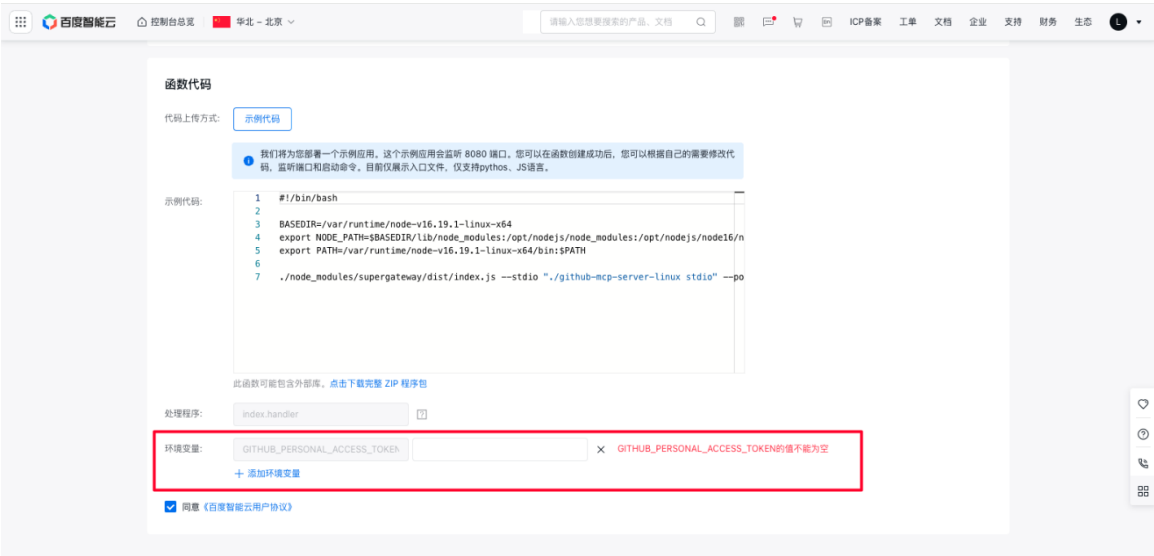


2、第一步，选择 **模版创建MCP服务**，热门MCP服务模版。本文以github为例。

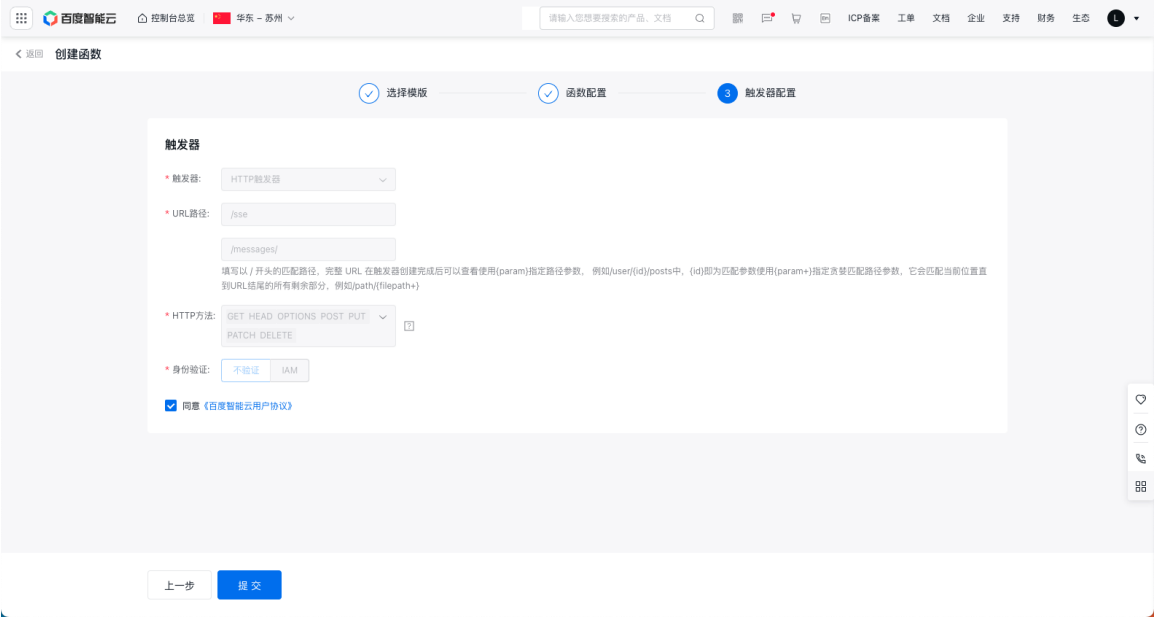


3、第二步，使用热门模版预置的参数和代码，填写您的函数名称、GITHUB_PERSONAL_ACCESS_TOKEN。



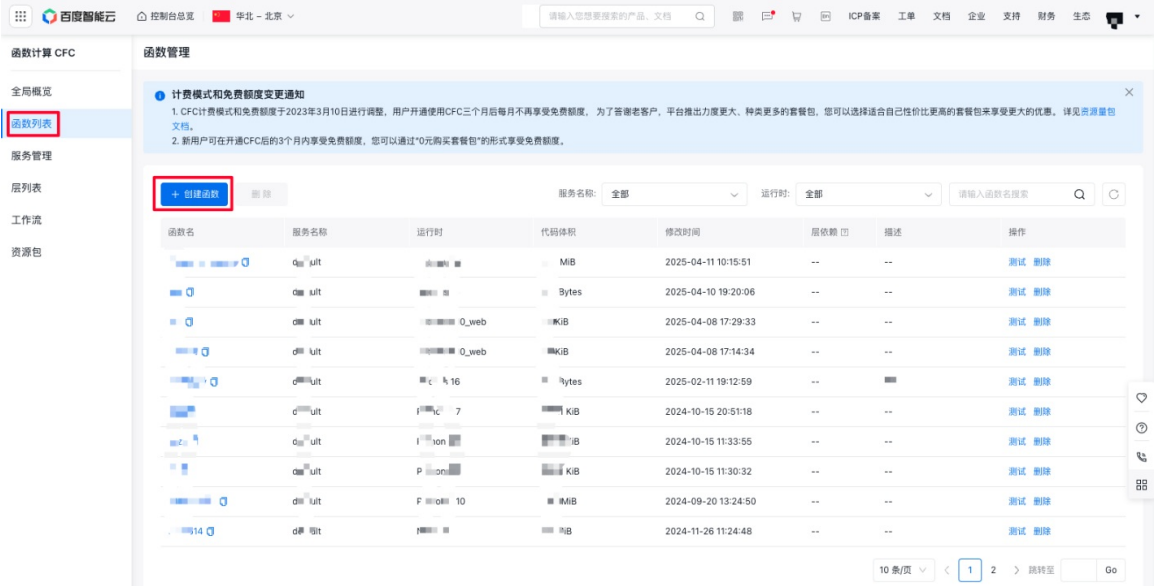


4、第三步，使用默认的触发器配置，完成提交。

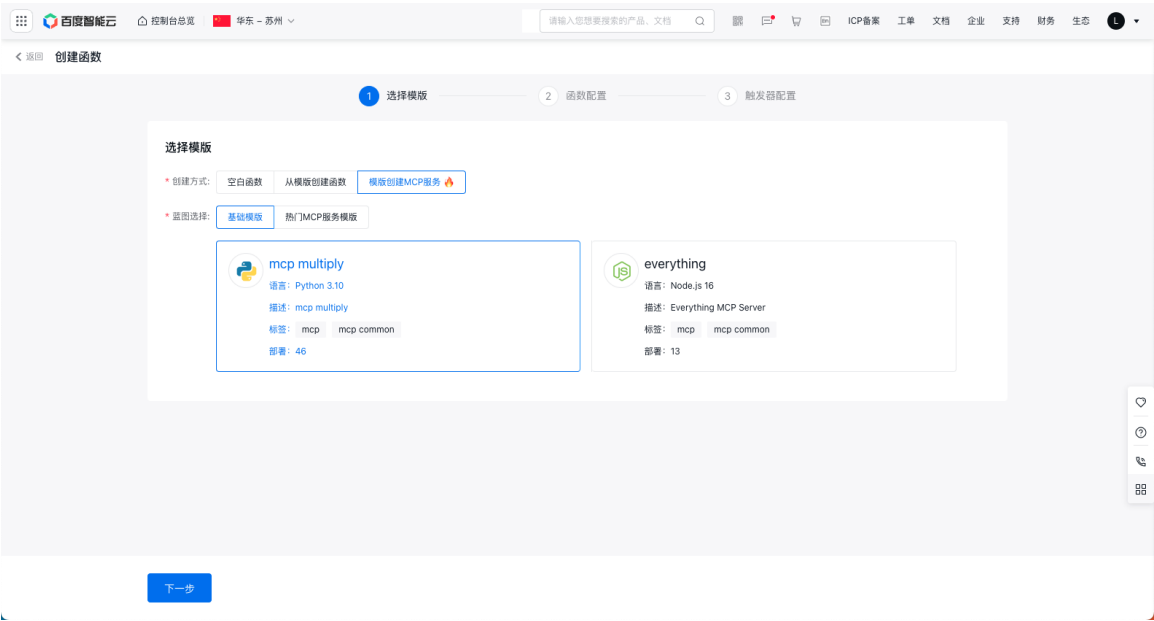


方式二：基于基础模版自定义部署

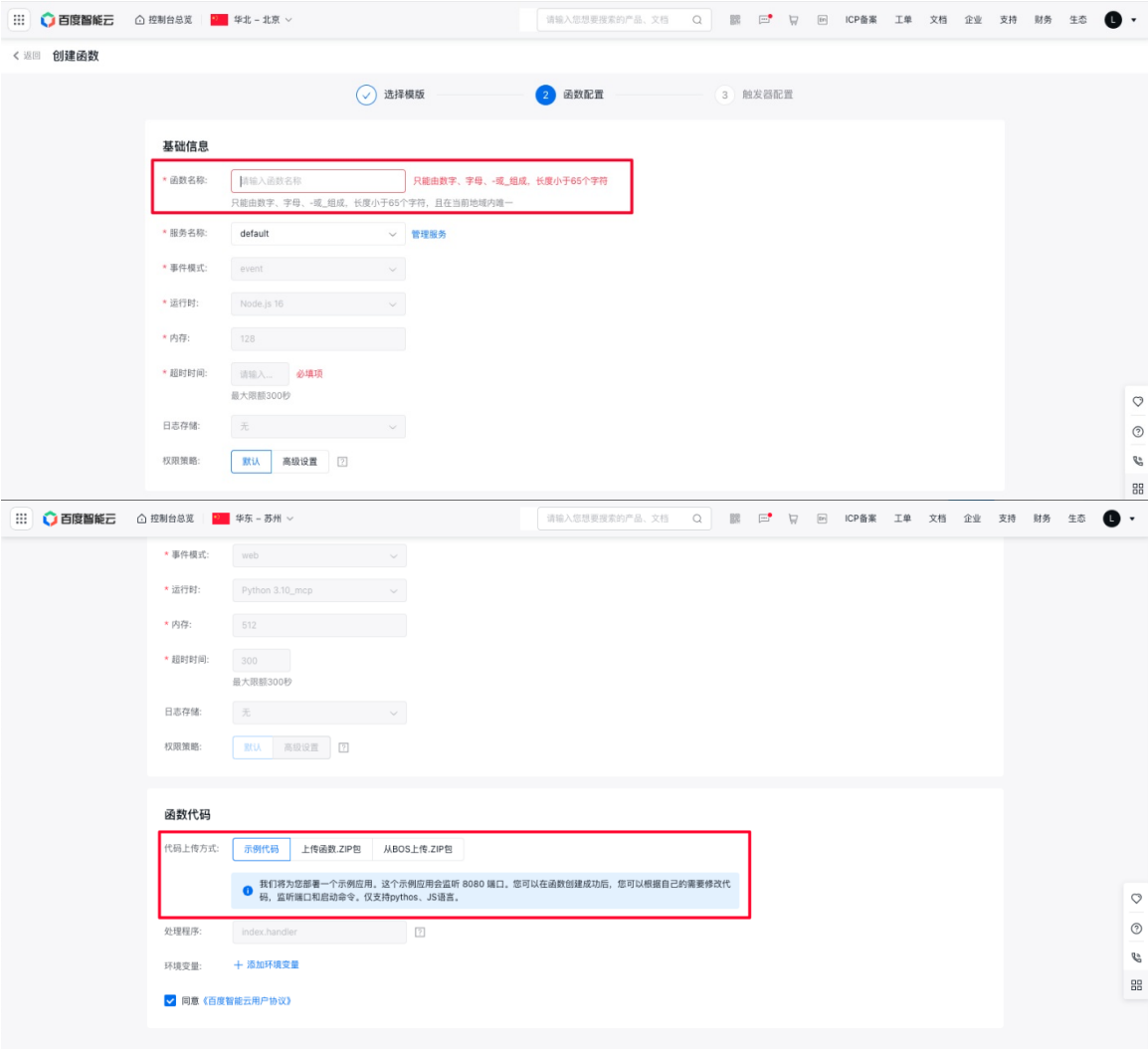
1、登录 CFC控制台，进入函数列表，点击创建函数。



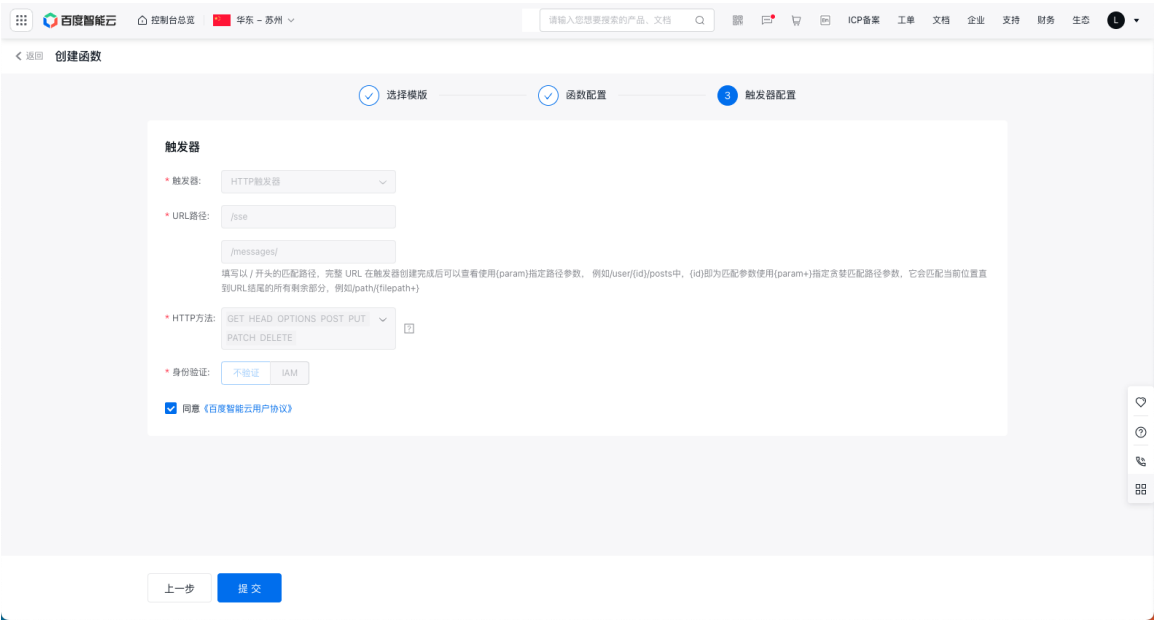
2、第一步，选择 模版创建MCP服务，基础模版。本文以mcp multiply为例。



3、第二步，使用热门模版预置参数，填写您的函数名称、您可以使用示例代码，或者上传您的自定义代码。



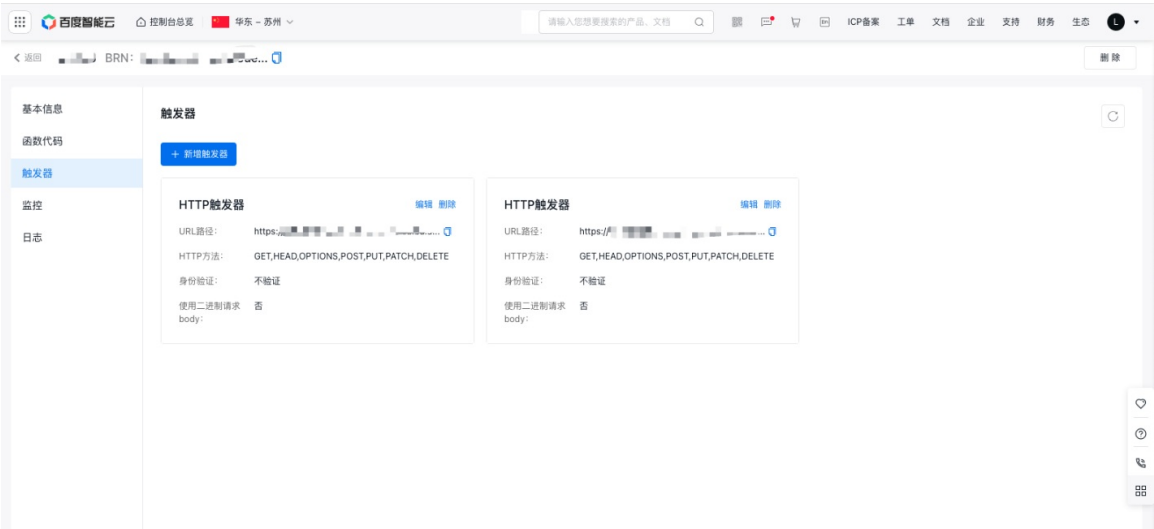
4、第三步，使用默认的触发器配置，完成提交。



测试MCP服务

您可以进入函数详情，获得SSE连接地址用于测试

- 1、进入函数列表，点击函数名称，进入函数详情。
- 2、点击 触发器，查看SSE连接地址。

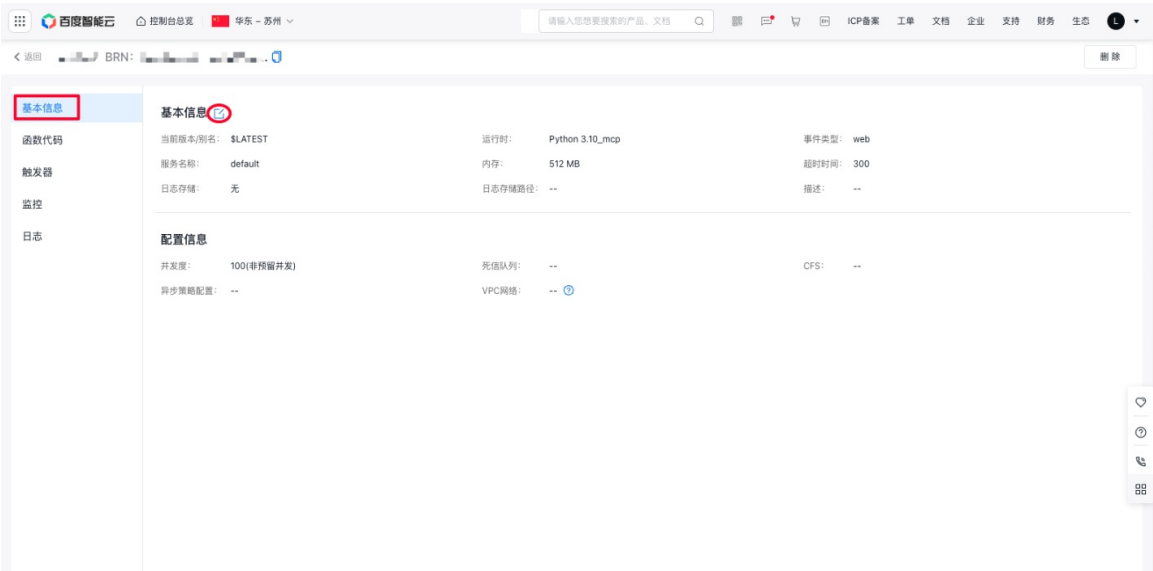


变更管理

您可以进入函数详情，更新函数基本信息，以及函数代码。

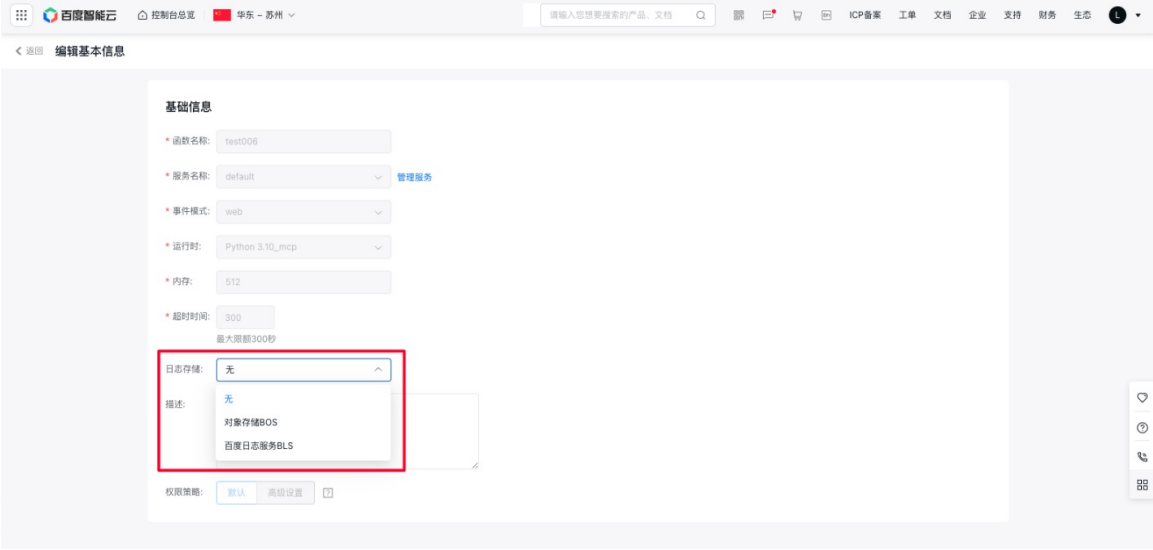
设置日志存储

- 1、进入函数列表，点击函数名称，进入函数详情。
- 2、点击基本信息，进入编辑。



3、设置

日志存储方式，默认是“无”，您可以选择使用BOS或者BLS，来保存您的函数运行过程中的日志。



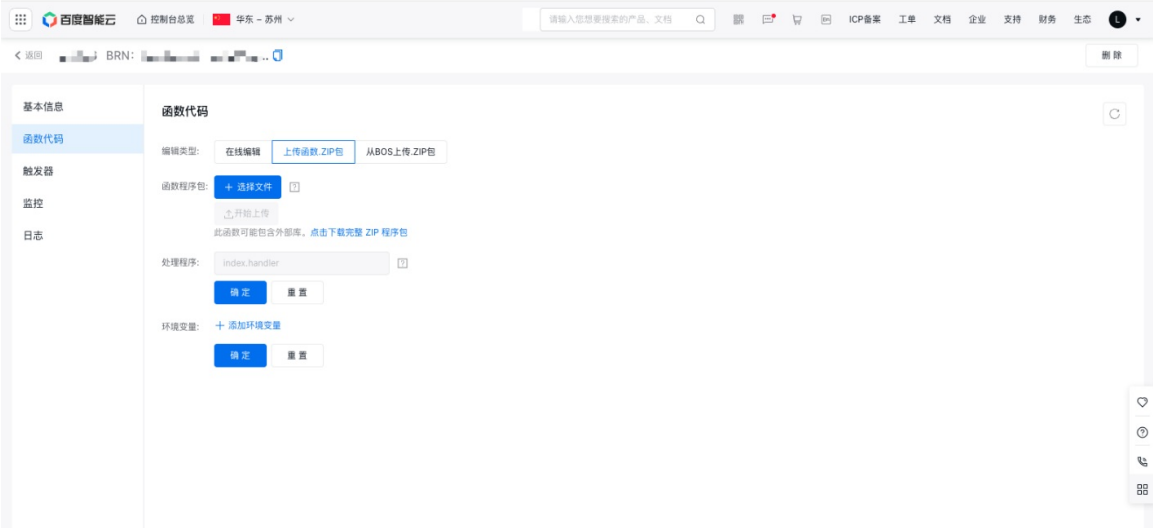
请注意：如果您选择BOS或者BLS方式存储日志，因此产生的费用将由BOS或者BLS产品收取，详情请参考：[BOS价格详情](#)、[BLS价格详情](#)。

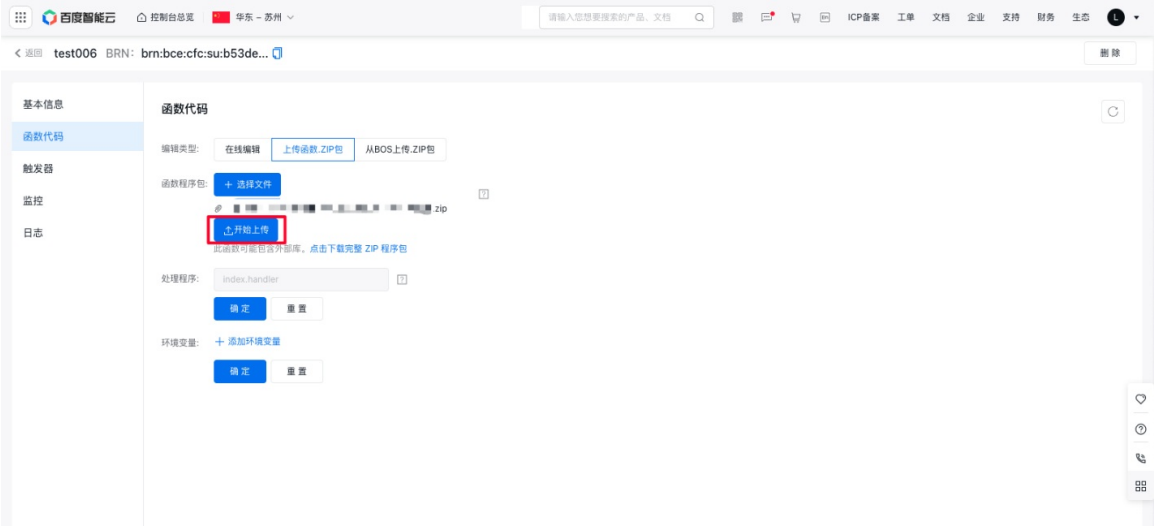
更新函数代码

1、进入函数列表，点击函数名称，进入函数详情。

2、点击函数代码，进行更新。

您可将ZIP包下载到本地进行更新，更新完成后重新上传，完成更新。





AWS Iamdba 迁移到 CFC 实践

本文将介绍如何从 AWS Lambda 函数迁移到百度云 CFC 的实践过程。

☞ AWS Iamdba 函数功能

☞ 创建函数

☞ 直接创建函数

1. 登录 AWS 官网，进入 [Lambda 产品页](#)



2. 点击创建函数进入函数创建页面，输入函数参数，创建函数



☞ 使用蓝图创建函数

1. 也可以选择『使用蓝图』来创建函数，选择具体的蓝图名称就可以创建函数



☞ 函数代码展示与编辑

☞ 函数代码展示

1. 进入详情页后可以查看函数代码



☞ 函数代码编辑

1. 编辑函数代码后，可以点击『Deploy』重新部署函数



☞ 配置函数层

1. 在『代码』选项最后，可以添加或者编辑层



2. 选择要添加的层类型和具体的层版本



3. 添加后可以看到具体的层信息，层代码会在函数执行时挂载到函数容器中



🔗 测试函数

1. 在『测试』页面中，点击测试，可以测试函数是否能正常运行，以及查看函数输出



🔗 配置触发器

1. 选择『配置』-『触发器』-『添加触发器』
2. 选择触发器类型为 APIGateway，API 类型选择 HTTP API，添加触发器



1. 添加后可以查看到触发器的具体 HTTP URL



访问该 url 可以调用函数获取结果

🔗 配置函数环境变量

1. 选择『配置』-『环境变量』，点击编辑，添加具体的环境变量



2. 添加具体的函数变量



3. 添加后可以查看具体的『环境变量』



🔗 函数版本与别名

🔗 创建函数版本

1. 点击『版本』选项，创建新的版本



2. 创建后进入版本详情页，可以为该函数版本配置具体的触发器等



3. 返回函数详情页，在『版本』选项可以查看具体创建的版本



🔗 创建函数别名

1. 点击『别名』，可以为函数创建别名



2. 选择具体的函数版本（需要已经创建函数版本，或者选择\$LATEST），还可以配置加权别名（选择配置别名时，不能选\$LATEST），并设置不同版本的权重



该功能可以用于发布函数新版本后，对版本进行灰度流量测试

3. 创建别名后，可以在别名详情页，为函数别名配置具体的触发器等



🔗 函数监控与日志

在『监控』选项，可以查看函数具体的调用数据与日志



🔗 迁移到 CFC

🔗 函数迁移

1. 在 lambda 中选择要迁移的函数，在函数详情页点击『下载』，选择下载函数代码 zip 文件



2. 进入 [CFC 首页](#)，点击创建函数，选择空白函数



下载后解压 zip 包，查看函数文件结构，主要关注函数入口文件。例子中的函数 zip 包解压后入口文件为 lambda_function.py，****这里要记住入口函数文件名称 和入口函数名称****。

3. 配置函数名称、运行时等参数，注意运行时需要选择和 lambda 函数一样或者类似版本的函数



4. 直接提交，创建函数。创建后，进入函数详情页，查看函数信息



5. 在『函数代码』中查看函数具体的例子



CFC 函数入口形式与 lambda 基本一致，但是具体的入口文件和函数需要根据 lambda 函数包实际进行修改。

6. 点击上传 zip 包，将 lambda 下载的函数包上传



7. zip 包上传后，需要修改处理程序入口，改为上面 lambda 函数的入口文件（注意入口文件与入口函数都需要修改）



8. 修改后，就能查看到 lambda 函数代码，**这里能看到函数代码，才可以认为 lambda 函数部署完成。**



🔗 函数功能使用

🔗 层配置

在『层配置』选项中，可以添加具体的函数层，添加后，层代码会自动加载到函数中



🔗 测试函数

在函数详情页，点击最右侧的测试按钮，执行测试



测试成功，说明函数可用，如果失败，可能是函数文件和入口函数不对



🔗 配置触发器

点击『触发器』选项，新增触发器，选择 HTTP 触发器，配置 URL 路径和 HTTP 方法



配置后的 HTTP 触发器会暴露一个可以公网访问的 URL



🔗 设置环境变量

在『函数代码』中，可以添加和更改环境变量



🔗 函数版本与别名

在函数详情页顶端，选择『发布新版本』可以发布函数版本



发布后可以查看具体的函数版本，具体的函数版本不能够修改代码，但是可以配置触发器



点击该按钮，选择『创建别名』可以创建函数别名



可以基于特定的版本发布别名，也可以发布灰度版本（对应 lambda 的加权别名）



🔗 函数监控与日志

在『监控』页面可以查看具体的监控项



在『日志』页面可以查看每次函数调用的日志信息，但是需要先配置函数的日志选项



这里可以配置日志存到到 BOS 还是 BLS



配置完成后，再次调用函数，可以在日志中查看具体的函数调用



API参考

概述

CFC提供基于事件机制，弹性、高可用、扩展性好、极速响应的云端无服务器计算能力。您可以仅关注业务逻辑的代码部分，无需关注和配置服务器资源，支持多种函数触发器，满足多样化的事件触发场景。

CFC API对于每个HTTP请求，认证签名放在Authorization头域中，后端统一认证。且后端在Response头域中会添加x-bce-request-id，作为请求唯一标识，方便追踪定位问题。

如果您是初次调用百度智能云产品的API，可以观看[API入门视频指南](#)，快速掌握调用API的能力。

通用说明

API调用遵循HTTP协议，各Region采用不同的域名，具体域名为cfc.{region}.baidubce.com。数据交换格式为JSON，所有request/response body内容均采用UTF-8编码。

实名认证

使用CFC API的用户需要实名认证，没有通过实名认证的可以前往[百度开放云官网控制台](#)中的安全认证下的实名认证中进行认证。没有通过实名认证的用户请求将会得到以下错误提示码：

错误码	错误描述	HTTP状态码	中文解释
QualifyNotPass	The User has not pass qualify.	403	账号没有通过实名认证

API认证机制

所有API的安全认证一律采用Access Key与请求签名机制。Access Key由Access Key ID和Secret Access Key组成，均为字符串。对于每个HTTP请求，使用下面所描述的算法生成一个认证字符串。提交认证字符串放在Authorization头域里。服务端根据生成算法验证认证字符串的正确性。认证字符串的格式为bce-auth-v{version}/{accessKeyId}/{timestamp}/{expirationPeriodInSeconds}/{signedHeaders}/{signature}。

- version是正整数。
- timestamp是生成签名时的UTC时间。
- expirationPeriodInSeconds表示签名有效期限。
- signedHeaders是签名算法中涉及到的头域列表。头域名之间用分号（;）分隔，如host;x-bce-date。列表按照字典序排列。（本API签名仅使用host和x-bce-date两个header）
- signature是256位签名的十六进制表示，由64个小写字母组成。

当百度智能云接收到用户的请求后，系统将使用相同的SK和同样的认证机制生成认证字符串，并与用户请求中包含的认证字符串进行比对。如果认证字符串相同，系统认为用户拥有指定的操作权限，并执行相关操作；如果认证字符串不同，系统将忽略该操作并返回错误码。

鉴权认证机制的详细内容请参见[鉴权认证机制](#)。

通信协议

API支持HTTP和HTTPS两种调用方式。为了提升数据的安全性，建议通过HTTPS调用。

请求结构说明

数据交换格式为JSON，所有request/response body内容均采用UTF-8编码。

请求参数包括如下4种：

参数类型	说明
URI	通常用于指明操作实体,如:POST /{version}/functions/{FunctionName}
Query参数	URL中携带的请求参数
HEADER	通过HTTP头域传入,如:x-bce-date
RequestBody	通过JSON格式组织的请求数据体

🔗 响应结构说明

响应值分为两种形式：

返回内容	说明
HTTP STATUS CODE	如200,400,403,404等
ResponseBody	JSON格式组织的响应数据体

🔗 API版本号

参数	类型	参数位置	描述	是否必须
version	String	URL参数	API版本号，当前取值为1	是

🔗 密码加密传输规范

所有涉及密码的接口参数都需要加密，禁止明文传输。密码一律采用AES 128位加密算法进行加密，用SK的前16位作为密钥，加密后生成的二进制字节流需要转成十六进制，并以字符串的形式传到服务端。具体步骤如下：

- byte[] bCiphertext= AES(明文,SK)
- String strHex = HexStr(bCiphertext)

🔗 日期与时间规范

日期与时间的表示有多种方式。为统一起见，除非是约定俗成或者有相应规范的，凡需要日期时间表示的地方一律采用UTC时间，遵循ISO 8601，并做以下约束：

1. 表示日期一律采用YYYY-MM-DD方式，例如2014-06-01表示2014年6月1日
2. 表示时间一律采用hh:mm:ss方式，并在最后加一个大写字母Z表示UTC时间。例如23:00:10Z表示UTC时间23点0分10秒。
3. 凡涉及日期和时间合并表示时，在两者中间加大写字母T，例如2014-06-01T23:00:10Z表示UTC时间2014年6月1日23点0分10秒。

🔗 规范化字符串

通常一个字符串中可以包含任何Unicode字符。在编程中这种灵活性会带来不少困扰。因此引入“规范字符串”的概念。一个规范字符串只包含百分号编码字符以及URI（Uniform Resource Identifier）非保留字符（Unreserved Characters）。

RFC 3986规定URI非保留字符包括以下字符：字母（A-Z，a-z）、数字（0-9）、连字号（-）、点号（.）、下划线（_）、波浪线（~）。

将任意一个字符串转换为规范字符串的方式是：

- 将字符串转换成UTF-8编码的字节流。
- 保留所有URI非保留字符原样不变。

- 对其余字节做一次RFC 3986中规定的百分号编码（Percent-Encoding），即一个%后面跟着两个表示该字节值的十六进制字母。字母一律采用大写形式。

示例：

原字符串：this is an example for 测试, 对应的规范字符

串：this%20is%20an%20example%20for%20E6%B5%8B%E8%AF%95。

🔗 排版约定

排版格式	含义
< >	变量
[]	可选项
{ }	必选项
	互斥关系
等宽字体 Courier New	屏幕输出

服务域名

百度智能云目前开通了多区域支持，请参考[区域选择说明](#)。

目前支持"华北-北京"、"华南-广州"、"华东-苏州"三个区域。

CFC和XFLOW访问域名支持HTTP和HTTPS两种调用方式。为了提升数据的安全性，建议通过HTTPS调用。更多关于HTTPS保障传输安全的使用请参考[HTTPS传输加密实践](#)。

CFC访问域名为：

区域	访问Endpoint	协议
北京	cfc.bj.baidubce.com	HTTP, HTTPS
广州	cfc.gz.baidubce.com	HTTP, HTTPS
苏州	cfc.su.baidubce.com	HTTP, HTTPS

XFLOW访问域名为：

区域	访问Endpoint	协议
北京	xflow.bj.baidubce.com	HTTP, HTTPS
广州	xflow.gz.baidubce.com	HTTP, HTTPS
苏州	xflow.su.baidubce.com	HTTP, HTTPS

公共请求头与公共响应头

🔗 公共请求头

下表列出了所有CFC API所携带的公共头域。HTTP协议的标准头域不在这里列出。

头域 (Header)	是否必须	说明
Authorization	必须	包含Access Key与请求签名，签名获取请阅读 百度云鉴权认证机制文档
Content-Type	可选	application/json; charset=utf-8
x-bce-date	可选	表示日期的字符串，符合百度智能云API规范
x-bce-content-sha256	可选	表示内容部分的SHA256签名的十六进制字符串。这里内容指HTTP Request Payload Body。即Content部分在被HTTP encode之前的原始数据

公共头域将在每个CFC API中出现，是必需的头域，其中x-bce-content-sha256头域只出现在POST和PUT请求中。

POST、PUT、DELETE等请求数据放在request body中。

🔗 公共响应头

下表列出了所有CFC API的公共响应头域。HTTP协议的标准响应头域不在这里列出。

头域 (Header)	说明
Content-Type	只支持JSON格式，application/json; charset=utf-8
X-Bce-Request-Id	CFC后端生成，并自动设置到响应头域中

其中，X-Bce-Request-Id使用UUID version4由CFC服务生成。

错误返回

🔗 通用错误返回

CFC错误响应符合BCE规范，统一为如下格式。后续各接口不再单独列出。

```
{
  "requestId": "47e0ef1a-9bf2-11e1-9279-0100e8cf109a",
  "code": "NoSuchKey",
  "message": "The resource you requested does not exist"
}
```

其中，code为错误码，所有错误码取值来源BCE公共错误码和CFC专有错误码（参考错误码部分内容）。

🔗 公共错误码

Code	Message	HTTP状态码	说明
AccessDenied	Access denied.	403 Forbidden	无权限访问对应的资源
InappropriateJSON	The JSON you provided was well-formed and valid, but not appropriate for this operation.	400 Bad Request	请求中的JSON格式正确，但语义上不符合要求。如缺少某个必需项，或者值类型不匹配等。出于兼容性考虑，对于所有无法识别的项应直接忽略，不应该返回这个错误。
InternalServerError	We encountered an internal error. Please try again.	500 Internal Server Error	所有未定义的其他错误。在有明确对应的其他类型的错误时（包括通用的和服务自定义的）不应该使用。
InvalidAccessKeyId	The Access Key ID you provided does not exist in our records.	403 Forbidden	Access Key ID不存在
InvalidHTTPAuthHeader	The HTTP authorization header is invalid. Consult the service documentation for details.	400 Bad Request	Authorization头域格式错误
InvalidHTTPRequest	There was an error in the body of your HTTP request.	400 Bad Request	HTTP body格式错误。例如不符合指定的Encoding等
InvalidURI	Could not parse the specified URI.	400 Bad Request	URI形式不正确。例如一些服务定义的关键词不匹配等。对于ID不匹配等问题，应定义更加具体的错误码，例如NoSuchKey。
MalformedJSON	The JSON you provided was not well-formed.	400 Bad Request	JSON格式不合法
InvalidVersion	The API version specified was invalid.	404 Not Found	URI的版本号不合法
OptInRequired	A subscription for the service is required.	403 Forbidden	没有开通对应的服务
IdempotentParameterMismatch	The request uses the same client token as a previous, but non-identical request.	412 Precondition Failed	ClientToken对应的API参数不一样。
SignatureDoesNotMatch	The request signature we calculated does not match the signature you provided. Check your Secret Access Key and signing method. Consult the service documentation for details.	400 Bad Request	Authorization头域中附带的签名和服务端验证不一致

|

Code	Message	HTTP状态码	说明
ENILimitReachedException	Unable to create an Elastic Network Interface (ENI) in the VPC	400 BadRequest	无法在VPC上创建一个弹性网络接口
InvalidParameterValueException	One of the parameters in the request is invalid	400 BadRequest	存在非法参数
InvalidRequestContentException	The request body could not be parsed as JSON	400 BadRequest	请求主体不能被解析为JSON
InvalidSecurityGroupIDException	The Security Group ID provided in the function VPC configuration is invalid	400 BadRequest	在函数VPC配置中提供的安全组ID是无效的
InvalidSubnetIDException	The Subnet ID provided in the function VPC configuration is invalid	400 BadRequest	在函数VPC配置中提供的子网ID是无效的
KMSAccessDeniedException	Unable to decrypt the environment variables because KMS access was denied	400 BadRequest	无密钥管理服务KMS访问权限，无法解析环境变量
KMSDisabledException	Unable to decrypt the environment variables because the KMS key used is disabled	400 BadRequest	无法对环境变量进行解密，因为使用的km是禁用的
KMSInvalidStateException	Unable to decrypt the environment variables because the KMS key used is a invalid state for Decrypt	400 BadRequest	无法解密环境变量，因为使用的公里密钥是一个无效的解密状态
PolicyLengthExceededException	Function access policy is limited to 20 KB	400 BadRequest	函数访问策略限制为20 KB
ServiceException	Service encountered an internal error	500 Internal Server Error	服务遇到内部错误
SubnetIPAddressLimitReachedException	Unable to set up VPC access because one or more configured subnets has n availab IP addresses"	400 BadRequest	无法设置VPC访问，因为一个或多个已配置的子网有n个可用的IP地址
UnsupportedMediaTypeException	The content type of the Invoke request body is not JSON	400 BadRequest	调用请求主体的内容类型不是JSON
ValidationException	Validation exception	400 BadRequest	校验出错

函数调用接口

调用函数 invocations

接口描述

- 本接口用于调用函数。

请求结构

```
POST /v1/functions/{FunctionName}/invocations HTTP/1.1
Host: cfc.bj.baidubce.com
Authorization: authorization string
```

请求头域

除公共头域外，无其它特殊头域。

请求参数

参数名称	类型	是否必需	参数位置	描述
FunctionName	String	是	Path参数	您可以指定一个函数名(例如, Thumbnail), 或者您可以指定函数的BRN资源名(例如, brn:bce:cfc:bj:account-id:function:thumbnail:\$LATEST)。注意, BRN长度限制1-140。如果只指定函数名, 则长度限制为64个字符。
InvocationType	String	否	Query参数	Event(异步调用)返回202/RequestResponse(同步返回)/DryRun(测试函数)。默认RequestResponse。
LogType	String	否	Query参数	日志类型 Tail / None 您可以将这个可选参数设置为Tail, 前提是InvocationType参数必须为RequestResponse。在本例中, CFC在x-bce-log-result头中返回最后4KB的base64编码的日志数据。
Qualifier	String	否	Query参数	您可以使用这个可选参数来指定CFC函数版本或别名。如果指定一个函数版本, 那么API将使用限定的函数BRN来调用特定的CFC函数。如果指定别名, 则API使用别名BRN来调用别名指向的CFC函数版本。如果您不提供此参数, 那么API将调用\$LATEST。

自定义参数

自定义函数所需的输入参数, 在Body参数以json数据格式传递。会作为input参数调用函数。

响应头域

HTTP/1.1 StatusCode
X-BCE-Function-Error: FunctionError
X-BCE-Log-Result: LogResult

Payload

响应参数

无指定响应参数。

请求示例

```
POST /v1/functions/testHelloworld/invocations HTTP/1.1
Authorization: Authorization string
Content-Type: application/json; charset=utf-8
Host: cfc.bj.baidubce.com
Content-Length: 49

{
  "key1": "value1",
  "key2": "value2",
  "key3": "value3"
}
```

返回示例

```
HTTP/1.1 200 OK
Content-Type: text/plain; charset=utf-8
Date: Fri, 15 Mar 2019 05:41:50 GMT
Server: nginx/1.9.2
Tracecode: 25104155220692565258031513
Vary: Accept-Encoding
X-Bce-Request-Id: 9bccb5e4-94c4-418c-b6da-e52f6c15ab1e
Content-Length: 18
Connection: close
```

Hello world!

函数操作

创建函数CreateFunction

接口描述

- 本接口用于创建函数。

请求结构

```
POST /v1/functions HTTP/1.1
Host: cfc.bj.baidubce.com
Authorization: authorization string
```

```
{
  "Code": {
    "ZipFile": "string",
    "Publish": false,
    "DryRun": true
  },
  "Description": "string",
  "Timeout": 3,
  "FunctionName": "string",
  "Handler": "string",
  "Runtime": "string",
  "MemorySize": 128,
  "Environment": {
    "Variables": {
      "additionalProp1": "string",
      "additionalProp2": "string",
      "additionalProp3": "string"
    }
  }
}
```

请求头域

除公共头域外，无其它特殊头域。

请求参数

参数名称	类型	是否必需	参数位置	描述
Code	Code	是	Body参数	CFC函数(zip格式最大长度限制为50MB, unzip格式的最大长度限制为250MB)
Description	String	否	Body参数	一个简短的说明 0-256字符
Environment	Environment	否	Body参数	环境变量(最大长度限制为4KB)
Function Name	String	是	Body参数	您想要分配给您正在上传的函数的名称。注意，长度限制只适用于BRN。如果只指定函数名，则长度限制为64个字符。
Handler	String	是	Body参数	cfc调用的入口函数，对于node为module-name.export eg. index.handler 最大长度128
Memory Size	int	否	Body参数	内存的大小，以MB为单位，CFC使用此内存大小来推断分配给您的函数的CPU和内存数量。默认值是128mb，必须是128MB的倍数。(现在CFC提供128至1024M的内存)
Runtime	String	是	Body参数	运行语言，支持 nodejs6.11/nodejs8.4/nodejs8.5/nodejs10/nodejs12/python2/python3/java8/java8_stream/golang/php7/lua5.3/dotnetcore2.2/powershell6.2
Timeout	int	是	Body参数	超时时间 1-300 最大300
LogType	string	否	Body参数	日志类型 bos（对象存储） / none。
LogBucket	string	否	Body参数	日志存储的 Bucket 路径。
VpcConfig	VpcConfig	否	Body参数	函数绑定的Vpc配置
DeadLetterTopic	string	否	Body参数	函数死信队列配置

响应头域

除公共头域，无其它特殊头域。

响应参数

参数名称	类型	描述
Uid	string	用户ID（由数字、字母、下划线组成），长度限制128字符。
Description	string	一个简短的说明，0-256字符。
FunctionBrn	string	函数的唯一性资源标志符，1-170字符。
Region	string	区域，现有 bj（北京）、su（苏州）、gz（广州）。
Timeout	int	超时时间 1-300，最大300。
VersionDesc	string	版本描述，0-256字符。
UpdatedAt	string	函数最新更新时间。 ISO-8601 格式 (YYYY-MM-DDThh:mm:ss.sTZD)。
LastModified	string	同UpdatedAt，兼容AWS Lambda。
CodeSha256	string	函数代码包的SHA256散列。
CodeSize	int	函数包的大小，单位bytes。
FunctionArn	string	同FunctionBrn，兼容AWS Lambda。
FunctionName	string	函数的名称，由数字、字母、-或_组成。则长度限制为64个字符。
Handler	string	CFC调用的入口函数，对于node为module-name.export eg. index.handler 最大长度为128字符。
Version	string	版本。LATEST表示最新，否则由数字组成。1-32字符。
Runtime	string	运行语言。
MemorySize	int	内存的大小，以MB为单位，CFC使用此内存大小来推断分配给您的函数的CPU和内存数量。默认值是128MB，必须是128MB的倍数。(现在CFC提供128至1024M的内存)。如果是Java8运行环境，内存大小为512至1024M。
Environment		Environment
CommitId	string	单次提交的唯一标志。
Role	string	函数的执行者角色。
LogType	string	日志类型 bos（对象存储） / none。
LogBosDir	string	日志存储的 Bucket 路径。
DeadLetterTopic	string	死信队列Topic。

请求示例

```
POST /v1/functions HTTP/1.1
Authorization: Authorization string
Content-Type: application/json; charset=utf-8
Host: cfc.bj.baidubce.com
Content-Length: 1042

{
  "Code": {
    "ZipFile":
"UESDBBQACAAIAAjjX00AAAAAAAAAAAAAAAAIABAAaW5kZXguanNVWAwAsJ/ZW/ie2Vv6Z7qeS60oyC8qKdbLSMxLyUktUrBVOE
",
  },
  "Description": "test api",
  "Environment": {
    "Variables": {
      "additionalProp1": "string"
    }
  },
  "FunctionName": "fly-hello-001",
  "Handler": "index.handler",
  "MemorySize": 256,
  "Publish": true,
  "Runtime": "python2",
  "Timeout": 5
}
```

返回示例

```
HTTP/1.1 201 Created
Content-Type: application/json
X-Bce-Request-Id: 2a9de674-03f6-42ff-b006-f0729e108dd5

{
  "Uid": "df391b08c64c426a81645468c75163a5",
  "Description": "test api",
  "FunctionBrn": "brn:bce:cfc:bj:cd64f99c69d7c404b61de0a4f1865834:function:fly-hello-001:1",
  "Region": "bj",
  "Timeout": 5,
  "VersionDesc": "",
  "UpdatedAt": "2019-03-15T15:07:06.040245209+08:00",
  "LastModified": "2019-03-15T15:07:06.040245209+08:00",
  "CodeSha256": "40FxEke82hUugwLdGb/BxnQdSUTsPAYcSU9PNVdFIU=",
  "CodeSize": 610,
  "FunctionArn": "brn:bce:cfc:bj:cd64f99c69d7c404b61de0a4f1865834:function:fly-hello-001:1",
  "FunctionName": "fly-hello-001",
  "Handler": "index.handler",
  "Version": "1",
  "Runtime": "python2",
  "MemorySize": 256,
  "Environment": {
    "Variables": {
      "additionalProp1": "string"
    }
  },
  "CommitId": "Ofeb0174-4288-4669-bfb6-a8c8c80df112",
  "Role": "",
  "LogType": "none"
}
```

函数列表ListFunctions

接口描述

- 本接口用于查询用户函数列表。

请求结构

```
GET /v1/functions HTTP/1.1
Host: cfc.bj.baidubce.com
Authorization: authorization string
```

请求头域

除公共头域外，无其它特殊头域。

请求参数

参数名称	类型	是否必需	参数位置	描述
FunctionVersion	String	否	Query参数	指定函数版本，如果没有指定返回所有函数\$LATEST版本，可选有效值 ALL将返回所有版本，包括\$LATEST
Marker	int	否	Query参数	标记
MaxItems	int	否	Query参数	返回最大条数，范围1-10000

响应头域

除公共头域，无其它特殊头域。

响应参数

参数名称	类型	描述
Functions	FunctionConfiguration数组	FunctionConfiguration数组

请求示例

```
GET /v1/functions?Marker=1&MaxItems=2 HTTP/1.1
Authorization: Authorization string
Host: cfc.bj.baidubce.com
```

响应示例

```
HTTP/1.1 200
Content-type: application/json

{
  "Functions": [
    {
      "Uid": "df391b08c64c426a81645468c75163a5",
      "Description": "",
      "FunctionBrn": "brn:bce:cfc:bj:cd64f99c69d7c404b61de0a4f1865834:function:httptrigger-helloworld:$LATEST",
      "Region": "bj",
      "Timeout": 3,
      "VersionDesc": "",
      "UpdatedAt": "2019-03-14T15:58:23+08:00",
      "LastModified": "2019-03-14T15:58:23+08:00",
      "CodeSha256": "vIqmTrOvKwuL94XG2HIdGBjqNOLRtZRAird21tOY2u0=",
      "CodeSize": 322291,
      "FunctionArn": "brn:bce:cfc:bj:cd64f99c69d7c404b61de0a4f1865834:function:httptrigger-helloworld:$LATEST",
      "FunctionName": "httptrigger-helloworld",
      "Handler": "index.handler",
      "Version": "$LATEST",
      "Runtime": "nodejs6.11",
      "MemorySize": 128,
      "Environment": {
        "Variables": {
          "helloworldsdssdsdsdsdsdsdsdsdsdsdsdsdsdsdsdsdsdsdsdsdsdsdsdsdsdsdsdsdsdsdsdsdsdsdsdsdsdsdsdsdsdsdsdsdsdsdsdsdsdsdsdsdsdsdsdsdsdsdsdsdsdsdsdsdsdsshellc
          :
          helloworldsdssdsdsdsdsdsdsdsdsdsdsdsdsdsdsdsdsdsdsdsdsdsdsdsdsdsdsdsdsdsdsdsdsdsdsdsdsdsdsdsdsdsdsdsdsdsdsdsdsdsdsdsdsdsdsdsdsdsdsdsdsdsdsdsdsdsshellc
          "
        }
      },
      "CommitId": "dcfbd208-796e-4a22-83e7-0c1c0698338d",
      "Role": "",
      "LogType": "bos",
      "LogBosDir": "bos://test-cfc/"
    },
    {
      "Uid": "df391b08c64c426a81645468c75163a5",
      "Description": "test api",
      "FunctionBrn": "brn:bce:cfc:bj:cd64f99c69d7c404b61de0a4f1865834:function:test:$LATEST",
      "Region": "bj",
      "Timeout": 5,
      "VersionDesc": "",

```

```
"UpdatedAt": "2019-01-21T17:10:22+08:00",
"LastModified": "2019-01-21T17:10:22+08:00",
"CodeSha256": "JTp+7f2JCxK41CGvec30Ctr1zc2kHNvWcgfOgF6VQSM=",
"CodeSize": 931858,
"FunctionArn": "brn:bce:cfc:bj:cd64f99c69d7c404b61de0a4f1865834:function:test:$LATEST",
"FunctionName": "test",
"Handler": "index.handler",
"Version": "$LATEST",
"Runtime": "python2",
"MemorySize": 256,
"Environment": {
  "Variables": {}
},
"CommitId": "6a648ed6-8f54-437f-af6b-6bfcf7d1e18f",
"Role": "",
"LogType": "none"
}
],
"NextMarker": "3"
}
```

函数信息GetFunction

接口描述

- 本接口用于查询用户单个函数。

请求结构

```
GET /v1/functions/{FunctionName} HTTP/1.1
Host: cfc.bj.baidubce.com
Authorization: authorization string
```

请求头域

除公共头域外，无其它特殊头域。

请求参数

参数名称	类型	是否必需	参数位置	描述
FunctionName	String	是	Path参数	函数名称 您可以指定一个函数名(例如，Thumbnail)，或者您可以指定函数的BRN资源名(例如，brn:bce:cfc:bj:account-id:function:thumbnail:\$LATEST)。CFC也允许您指定一个部分的BRN(例如，account-id:Thumbnail)。注意，BRN长度限制为1-170。如果只指定函数名，则长度限制为64个字符。
Qualifier	String	否	Query参数	使用此可选参数指定函数版本或别名。如果您指定函数版本，那么API将使用限定的函数BRN来请求并返回关于特定CFC函数版本的信息。如果指定别名，API会返回有关别名所指向的函数版本的信息。如果您不提供此参数，则API返回有关CFC函数\$LATEST的信息。

响应头域

除公共头域，无其它特殊头域。

响应参数

参数名称	类型	描述
Code	CodeStorage	代码信息相关
Configuration	FunctionConfiguration	函数信息相关

请求示例

```
GET /v1/functions/testHelloWorld HTTP/1.1
Host: cfc.bj.baidubce.com
Authorization: authorization string
```

响应示例

```
HTTP/1.1 200
Content-type: application/json

{
  "Code": {
    "Location":
"https://cfcbj.bj.bcebos.com/42f6fbc2cd374bfc80d9967370fd8ff/sltest_3ec6ae9a1aa36374b2efac5d03d8fe499270d5
?authorization=bce-auth-v1%2F4746faeb37854542b225fef7cc87fc36%2F2018-01-
23T07%3A17%3A03Z%2F600%2Fhost%2Fdea74e91c35c3dede418b8f9cf71b5269c42cce5c9d464af3d9e015ac8f4609"
  ,
    "RepositoryType": ""
  },
  "Configuration": {

    "Uid": "42f6fbc2cd374bfc80d9967370fd8ff",
    "Description": "测试",
    "FunctionBm": "bm:bce:cfc:bj:1a2cbf55b97ac8a7c760c4177db4e17d:function:sltest:1",
    "Region": "bj",
    "Timeout": 3,
    "VersionDesc": "测试",
    "UpdatedAt": "2018-01-23T15:16:34+08:00",
    "LastModified": "2018-01-23T15:16:34+08:00",
    "CodeSha256": "PsaumhqjY3Sy76xdA9j+SZJw1XjFBT5zUq0rIUUZUrw=",
    "CodeSize": 191,
    "FunctionArn": "bm:bce:cfc:bj:1a2cbf55b97ac8a7c760c4177db4e17d:function:sltest:1",
    "FunctionName": "sltest",
    "Handler": "index.handler",
    "Version": "1",
    "Runtime": "python2",
    "MemorySize": 128,
    "Environment": {
      "Variables": {
        "": ""
      }
    },
    "CommitId": "349dd8a4-db7d-4fc3-a4cd-8c8b482e00c3"
  }
}
```

删除函数DeleteFunction

接口描述

- 本接口用于删除函数。

请求结构

```
DELETE /v1/functions/{FunctionName} HTTP/1.1
Host: cfc.bj.baidubce.com
Authorization: authorization string
```

请求头域

除公共头域外，无其它特殊头域。

请求参数

参数名称	类型	是否必需	参数位置	描述
FunctionName	String	是	Path参数	函数名称 您可以指定一个函数名(例如，Thumbnail)，或者您可以指定函数的BRN资源名(例如，brn:bce:cfc:bj:account-id:function:thumbnail: \$LATEST)。CFC也允许您指定一个部分的BRN(例如，account-id:Thumbnail)。注意，BRN长度限制为1-170。如果只指定函数名，则长度限制为64个字符。
Qualifier	String	否	Query参数	使用这个可选参数，您可以指定一个函数版本(但不是\$LATEST)让CFC删除一个特定的函数版本。如果函数版本有一个或多个别名指向它，您将会得到一个错误，因为您不能有指向它的别名。您可以删除任何函数版本，但不能删除\$LATEST，也就是说，您不能指定\$LATEST作为该参数的值。只有当您想删除所有的函数版本和别名时，才可以删除\$LATEST。 您只能使用此参数指定函数版本，而不是别名。不能使用别名删除函数版本。 如果没有指定这个参数，CFC将删除该函数，包括所有的版本和别名。

响应头域

除公共头域，无其它特殊头域。

响应参数

无响应参数。

请求示例

```
DELETE /v1/functions/hello-001 HTTP/1.1
Host: cfc.bj.baidubce.com
Authorization: authorization string
```

响应示例

```
HTTP/1.1 204
Content-type: application/json
```

更新函数代码UpdateFunctionCode

接口描述

- 本接口用于更新指定function代码,可以选择ZipFile直接更新或者将函数代码Zip包放置到BOS中更新，两种方式不能同时使用，否则返回400。若选择ZIP包直接更新，则需设置ZipFile参数；若选择放置到BOS中更新，则需设置BosBucket，BosObject参数。其中函数代码包打包格式参考[文档](#)。

请求结构

```
PUT /v1/functions/{FunctionName}/code HTTP/1.1
Host: cfc.bj.baidubce.com
Authorization: authorization string
```

请求头域

除公共头域外，无其它特殊头域。

请求参数

参数名称	类型	是否必需	参数位置	描述
FunctionName	String	是	Path参数	函数名称 您可以指定一个函数名(例如，Thumbnail)，或者您可以指定函数的BRN资源名(例如，brn:bce:cfc:bj:account-id:function:thumbnail)。注意，BRN长度限制为1-140。如果只指定函数名，则长度限制为64个字符。
ZipFile	String	否	Body参数	您要发布的zip包的 base64-encoded 注意zip包压缩目录的内容，而不是目录本身,你可以通过使用base64命令行工具获得base64编码后的代码包。
BosBucket	String	否	Body参数	如果选择从BOS更新代码，BosBucket名为Zip包所在的Bucket
BosObject	String	否	Body参数	如果选择从BOS更新代码，BosObject名为Zip包的名字
Publish	Boolean	否	Body参数	是否直接发布
DryRun	Boolean	否	Body参数	这个布尔参数可用于测试您对CFC的请求，以更新CFC函数，并将一个版本作为原子操作发布。它将对你的代码进行所有必要的计算和验证，但是不会上传它或者发布一个版本。每次调用该操作时，所提供代码的CodeSha256散列值也将在响应中计算并返回。 暂未支持

响应头域

除公共头域，无其它特殊头域。

响应参数

见FunctionConfiguration

请求示例

```
PUT /v1/functions/fly-hello-001/code HTTP/1.1
Host: cfc.bj.baidubce.com
Authorization: authorization string

{
  "DryRun": true,
  "Publish": true,
  "ZipFile":
  "UESDBBQACAAIAAjjX00AAAAAAAAAAAAAAAAAIBAAaW5kZXguanNVWAwAsJ/ZW/ie2Vv6Z7qeS60oyC8qKdbLSMxLyUktUrBVOE
  "
}
```

响应示例

```
HTTP/1.1 200
Content-type: application/json

{
  "Uid": "df391b08c64c426a81645468c75163a5",
  "Description": "test api",
  "FunctionBrn": "brn:bce:cfc:bj:cd64f99c69d7c404b61de0a4f1865834:function:fly-hello-001:3",
  "Region": "bj",
  "Timeout": 5,
  "VersionDesc": "",
  "UpdatedAt": "2019-03-15T15:59:50+08:00",
  "LastModified": "2019-03-15T15:59:50+08:00",
  "CodeSha256": "40FxEke82hUugwLdGb/BxnQdSUTsPAYcSU9PNVdFIU=",
  "CodeSize": 610,
  "FunctionArn": "brn:bce:cfc:bj:cd64f99c69d7c404b61de0a4f1865834:function:fly-hello-001:3",
  "FunctionName": "fly-hello-001",
  "Handler": "index.handler",
  "Version": "3",
  "Runtime": "python2",
  "MemorySize": 256,
  "Environment": {
    "Variables": {
      "additionalProp1": "string"
    }
  },
  "CommitId": "dc901628-44b9-4b29-91e2-feacc3f5099",
  "Role": "",
  "LogType": "none"
}
```

获取函数配置GetFunctionConfiguration

接口描述

- 本接口用于获取指定函数配置。

请求结构

```
GET /v1/functions/{FunctionName}/configuration HTTP/1.1
Host: cfc.bj.baidubce.com
Authorization: authorization string
```

请求头域

除公共头域外，无其它特殊头域。

请求参数

参数名称	类型	是否必需	参数位置	描述
FunctionName	String	是	Path参数	函数名称 您可以指定一个函数名(例如，Thumbnail)，或者您可以指定函数的BRN资源名(例如，brn:bce:cfc:bj:account-id:function:thumbnail:\$LATEST)。CFC也允许您指定一个部分的BRN(例如，account-id:Thumbnail)。注意，BRN长度限制为1-170。如果只指定函数名，则长度限制为64个字符。
Qualifier	String	否	Query参数	使用此可选参数指定函数版本或别名。如果您指定函数版本，那么API将使用限定的函数BRN来请求并返回关于特定CFC函数版本的信息。如果指定别名，API会返回有关别名所指向的函数版本的信息。如果您不提供此参数，则API返回有关CFC函数\$LATEST的信息。

响应头域

除公共头域，无其它特殊头域。

响应参数

见[FunctionConfiguration](#)

请求示例

```
GET /v1/functions/testHelloWorld/configuration HTTP/1.1
Host: cfc.bj.baidubce.com
Authorization: authorization string
```

响应示例

```
HTTP/1.1 200
Content-type: application/json

{
  "Uid": "df391b08c64c426a81645468c75163a5",
  "Description": "",
  "FunctionBrn": "brn:bce:cfc:bj:cd64f99c69d7c404b61de0a4f1865834:function:testHelloWorld:$LATEST",
  "Region": "bj",
  "Timeout": 3,
  "VersionDesc": "",
  "UpdatedAt": "2019-03-14T14:23:51+08:00",
  "LastModified": "2019-03-14T14:23:51+08:00",
  "CodeSha256": "Zpf3bcfy2h7roMY9BmbmVElqlyapn3Er7LUmDp+qck=",
  "CodeSize": 221,
  "FunctionArn": "brn:bce:cfc:bj:cd64f99c69d7c404b61de0a4f1865834:function:testHelloWorld:$LATEST",
  "FunctionName": "testHelloWorld",
  "Handler": "index.handler",
  "Version": "$LATEST",
  "Runtime": "nodejs6.11",
  "MemorySize": 128,
  "Environment": {
    "Variables": {}
  },
  "CommitId": "d5cfc35b-a109-44f6-a6c7-7b82e6fca830",
  "Role": "",
  "LogType": "bos",
  "LogBosDir": "bos://hello-dev-docs/"
}
```

更新函数配置UpdateFunctionConfiguration

接口描述

- 本接口用于修改函数配置。

请求结构

```
PUT /v1/functions/{FunctionName}/configuration HTTP/1.1
Host: cfc.bj.baidubce.com
Authorization: authorization string

{
  "Description": "string",
  "Timeout": 3,
  "Handler": "string",
  "Runtime": "string",
  "Environment": {
    "Variables": {
      "additionalProp1": "string",
      "additionalProp2": "string",
      "additionalProp3": "string"
    }
  }
}
```

请求头域

除公共头域外，无其它特殊头域。

请求参数

参数名称	类型	是否必需	参数位置	描述
FunctionName	String	是	Path参数	函数名称 您可以指定一个函数名(例如, Thumbnail), 或者您可以指定函数的BRN资源名(例如:brn:bce:cfc:bj:account-id:function:thumbnail)。CFC也允许您指定一个部分的BRN(例如, account-id:Thumbnail)。注意, BRN长度限制为1-140。如果只指定函数名, 则长度限制为64个字符。
Timeout	int	否	Body参数	超时时间。
Description	String	否	Body参数	一个简短的说明。
Handler	String	否	Body参数	入口。
Runtime	String	否	Body参数	运行语言, 支持 nodejs6.11/nodejs8.4/nodejs8.5/nodejs10/nodejs12/python2/python3/java8/java8_stream/golang/php7/lua5.3/dotnetcore2.2/powershell6.2 。
Environment	Environment	否	Body参数	环境变量。
MemorySize	int	否	Body参数	内存大小。
LogType	string	否	Body参数	日志类型 bos (对象存储) / none。
LogBosDir	string	否	Body参数	日志存储的 Bucket 路径。
VpcConfig	VpcConfig	否	Body参数	函数绑定的Vpc配置
DeadLetterTopic	string	否	Body参数	函数死信队列配置

- 注意：上述请求参数只能在\$LATEST版本中更改，非\$LATEST版本只能更改LogType和LogBosDir两个字段

响应头域

除公共头域, 无其它特殊头域。

响应参数

见[FunctionConfiguration](#)

请求示例

```
PUT /v1/functions/fly-hello-001/configuration HTTP/1.1
Host: cfc.bj.baidubce.com
Authorization: authorization string

{
  "Code": {
    "ZipFile":
"UESDBBQACAAIAAyiX00AAAAAAAAAAAAAAAAAIBAAaW5kZXguanNVWAwAsJ/ZW/ie2Vv6Z7qeS60oyC8qKdbLSMxLyUktUrBVOE
",
  },
  "Description": "test api",
  "Environment": {
    "Variables": {
      "additionalProp1": "string"
    }
  },
  "FunctionName": "fly-hello-001",
  "Handler": "index.handler",
  "MemorySize": 256,
  "Publish": true,
  "Runtime": "python2",
  "Timeout": 5
}
```

响应示例

```
HTTP/1.1 200
Content-type: application/json

{
  "Uid": "df391b08c64c426a81645468c75163a5",
  "Description": "test api",
  "FunctionBrn": "brn:bce:cfc:bj:cd64f99c69d7c404b61de0a4f1865834:function:fly-hello-001:$LATEST",
  "Region": "bj",
  "Timeout": 5,
  "VersionDesc": "",
  "UpdatedAt": "2019-03-15T16:07:24+08:00",
  "LastModified": "2019-03-15T16:07:24+08:00",
  "CodeSha256": "40FxEke82hUugwLdGb/BxnQdSUTsPAYcSU9PNVdFIU=",
  "CodeSize": 610,
  "FunctionArn": "brn:bce:cfc:bj:cd64f99c69d7c404b61de0a4f1865834:function:fly-hello-001:$LATEST",
  "FunctionName": "fly-hello-001",
  "Handler": "index.handler",
  "Version": "$LATEST",
  "Runtime": "python2",
  "MemorySize": 256,
  "Environment": {
    "Variables": {
      "additionalProp1": "string"
    }
  },
  "CommitId": "4e478a27-8f4a-4b7f-96e1-dc7923512f7b",
  "Role": "",
  "LogType": "none"
}
```

设置函数预留并发度

接口描述

- 本接口用于设置函数预留并发度。

请求结构

```
PUT /v1/functions/{FunctionName}/concurrency HTTP/1.1
Host: cfc.bj.baidubce.com
Authorization: authorization string

{
  "ReservedConcurrentExecutions": 10
}
```

请求头域

除公共头域外，无其它特殊头域。

请求参数

参数名称	类型	是否必需	参数位置	描述
FunctionName	String	是	Path参数	函数名称 您可以指定一个函数名(例如，Thumbnail)，或者您可以指定函数的BRN资源名(例如:brn:bce:cfc:bj:account-id:function:thumbnail)。CFC也允许您指定一个部分的BRN(例如，account-id:Thumbnail)。注意，BRN长度限制为1-140。如果只指定函数名，则长度限制为64个字符。
ReservedConcurrentExecutions	int	是	Body参数	函数预留并发度，并发度为函数可以并发执行的数量上限。当前账户的总并发度限额为100，已预留给函数的额度为0，剩余额度由非预留并发函数共享。 如需提高并发度限额，请提交工单申请。

响应头域

除公共头域，无其它特殊头域。

响应参数

无响应参数。

请求示例

```
PUT /v1/functions/fly-hello-001/concurrency HTTP/1.1
Host: cfc.bj.baidubce.com
Authorization: authorization string

{
  "ReservedConcurrentExecutions": 10
}
```

响应示例

```
HTTP/1.1 200
```

删除函数预留并发度

接口描述

- 本接口用于删除函数预留的并发度。

请求结构

```
DELETE /v1/functions/{FunctionName}/concurrency HTTP/1.1
Host: cfc.bj.baidubce.com
Authorization: authorization string
```

请求头域

除公共头域外，无其它特殊头域。

请求参数

参数名称	类型	是否必需	参数位置	描述
FunctionName	String	是	Path参数	函数名称 您可以指定一个函数名(例如，Thumbnail)，或者您可以指定函数的BRN资源名(例如:brn:bce:cfc:bj:account-id:function:thumbnail)。CFC也允许您指定一个部分的BRN(例如，account-id:Thumbnail)。注意，BRN长度限制为1-140。如果只指定函数名，则长度限制为64个字符。

响应头域

除公共头域，无其它特殊头域。

响应参数

无响应参数。

请求示例

```
DELETE /v1/functions/fly-hello-001/concurrency HTTP/1.1
Host: cfc.bj.baidubce.com
Authorization: authorization string
```

响应示例

```
HTTP/1.1 204
```

版本操作

获取函数版本列表ListVersionsByFunction

接口描述

- 本接口用于查询函数所有版本。

请求结构

```
GET /v1/functions/{FunctionName}/versions HTTP/1.1
Host: cfc.bj.baidubce.com
Authorization: authorization string
```

请求头域

除公共头域外，无其它特殊头域。

请求参数

参数名称	类型	是否必需	参数位置	描述
FunctionName	String	是	Path参数	函数名称 您可以指定一个函数名(例如，Thumbnail)，或者您可以指定函数的BRN资源名(例如:brn:bce:cfc:bj:account-id:function:thumbnail)。CFC也允许您指定一个部分的BRN(例如，account-id:Thumbnail)。注意，BRN长度限制为1-140。如果只指定函数名，则长度限制为64个字符。
Marker	int	否	Query参数	标记
MaxItems	int	否	Query参数	最大数目 1-10000

响应头域

除公共头域，无其它特殊头域。

响应参数

参数名称	类型	描述	
Versions	FunctionConfiguration数组	FunctionConfiguration数组	

请求示例

```
GET /v1/functions/testHelloWorld/versions HTTP/1.1
Host: cfc.bj.baidubce.com
Authorization: authorization string
```

响应示例

```
HTTP/1.1 200
Content-type: application/json

{
  "Versions": [
    {
      "Uid": "df391b08c64c426a81645468c75163a5",
      "Description": "",
      "FunctionBrn": "brn:bce:cfc:bj:cd64f99c69d7c404b61de0a4f1865834:function:testHelloWorld:$LATEST",
      "Region": "bj",
      "Timeout": 3,
      "VersionDesc": "",
      "UpdatedAt": "2019-03-14T14:23:51+08:00",
      "LastModified": "2019-03-14T14:23:51+08:00",
      "CodeSha256": "Zpf3bcfyy2h7roMY9BmbmVElqlyapn3Er7LUmDp+qck=",
      "CodeSize": 221,
      "FunctionArn": "brn:bce:cfc:bj:cd64f99c69d7c404b61de0a4f1865834:function:testHelloWorld:$LATEST",
      "FunctionName": "testHelloWorld",
      "Handler": "index.handler",
      "Version": "$LATEST",
      "Runtime": "nodejs6.11",
      "MemorySize": 128,
      "Environment": {
        "Variables": {}
      },
      "CommitId": "d5cfc35b-a109-44f6-a6c7-7b82e6fca830",
      "Role": "",
      "LogType": "bos",
      "LogBosDir": "bos://hello-dev-docs/"
    },
    {
      "Uid": "df391b08c64c426a81645468c75163a5",
      "Description": "",
      "FunctionBrn": "brn:bce:cfc:bj:cd64f99c69d7c404b61de0a4f1865834:function:testHelloWorld:1",
      "Region": "bj",
      "Timeout": 3,
      "VersionDesc": "xxxxxx",
      "UpdatedAt": "2018-12-13T19:39:13+08:00",
      "LastModified": "2018-12-13T19:39:13+08:00",
      "CodeSha256": "rvQxaATysJUHWDHQZu5c/Kw7H/HtmDmmSnVtXBSFCcl=",
      "CodeSize": 215,
      "FunctionArn": "brn:bce:cfc:bj:cd64f99c69d7c404b61de0a4f1865834:function:testHelloWorld:1",
      "FunctionName": "testHelloWorld",
      "Handler": "index.handler",
      "Version": "1",
      "Runtime": "nodejs6.11",
      "MemorySize": 128,
      "Environment": {
        "Variables": {}
      },
      "CommitId": "b8cb6304-63cd-42ee-854d-d5ac408643e8",
      "Role": "",
      "LogType": "none"
    }
  ]
}
```

发布版本PublishVersion

接口描述

- 本接口用于发布函数版本。

请求结构

```
POST /v1/functions/{FunctionName}/versions HTTP/1.1
Host: cfc.bj.baidubce.com
Authorization: authorization string

{
  "CodeSha256": "string",
  "Description": "string"
}
```

请求头域

除公共头域外，无其它特殊头域。

请求参数

参数名称	类型	是否必需	参数位置	描述
FunctionName	String	是	Path参数	函数名称 您可以指定一个函数名(例如，Thumbnail)，或者您可以指定函数的BRN资源名(例如:brn:bce:cfc:bj:account-id:function:thumbnail)。注意，BRN长度限制为1-140。如果只指定函数名，则长度限制为64个字符。
Description	string	否	Body参数	版本描述 0-125
CodeSha256	string	否	Body参数	您想要发布的部署包的SHA256散列。这将为发布的代码提供验证。如果您提供此参数，则该值必须与发布成功的Latest版本的SHA256匹配

响应头域

除公共头域，无其它特殊头域。

响应参数

见[FunctionConfiguration](#)

请求示例

```
POST /v1/functions/{FunctionName}/versions HTTP/1.1
Host: cfc.bj.baidubce.com
Authorization: authorization string

{
  "Description": "xxxxxxx",
  "FunctionVersion": "$LATEST",
  "Name": "helloFly001"
}
```

响应示例

```
HTTP/1.1 201
Content-type: application/json

{
  "Uid": "df391b08c64c426a81645468c75163a5",
  "Description": "",
  "FunctionBrn": "brn:bce:cfc:bj:cd64f99c69d7c404b61de0a4f1865834:function:testHelloWorld:2",
  "Region": "bj",
  "Timeout": 3,
  "VersionDesc": "xxxxxxx",
  "UpdatedAt": "2019-03-14T14:23:51+08:00",
  "LastModified": "2019-03-14T14:23:51+08:00",
  "CodeSha256": "Zpf3bcfy2h7roMY9BmbmVElqlyapn3Er7LUmDp+qck=",
  "CodeSize": 221,
  "FunctionArn": "brn:bce:cfc:bj:cd64f99c69d7c404b61de0a4f1865834:function:testHelloWorld:2",
  "FunctionName": "testHelloWorld",
  "Handler": "index.handler",
  "Version": "2",
  "Runtime": "nodejs6.11",
  "MemorySize": 128,
  "Environment": {
    "Variables": {}
  },
  "CommitId": "d5cfc35b-a109-44f6-a6c7-7b82e6fca830",
  "Role": "",
  "LogType": "bos",
  "LogBosDir": "bos://hello-dev-docs/"
}
```

别名操作

别名列表ListAliases

接口描述

- 本接口用于查询函数所有别名。

请求结构

```
GET /v1/functions/{FunctionName}/aliases HTTP/1.1
Host: cfc.bj.baidubce.com
Authorization: authorization string
```

请求头域

除公共头域外，无其它特殊头域。

请求参数

参数名称	类型	是否必需	参数位置	描述
FunctionName	String	是	Path参数	函数名称 创建别名的函数名。注意，长度限制只适用于BRN。如果只指定函数名，则长度限制为64个字符。
FunctionVersion	String	否	Query参数	函数版本 如果您指定了这个可选参数，那么API只返回指向特定CFC函数版本的别名，否则API将返回为CFC函数创建的所有别名。
Marker	int	否	Query参数	标记
MaxItems	int	否	Query参数	最大数目 1-10000

响应头域

除公共头域，无其它特殊头域。

响应参数

参数名称	类型	描述
Aliases	AliasConfiguration数组	AliasConfiguration数组

请求示例

```
GET /v1/functions/testHelloWorld/aliases HTTP/1.1
Host: cfc.bj.baidubce.com
Authorization: authorization string
```

响应示例

```
HTTP/1.1 200 OK
Content-type: application/json

{
  "Aliases": [
    {
      "AliasBrn": "brn:bce:cfc:bj:1a2cbf55b97ac8a7c760c4177db4e17d:function:111:ceshi",
      "AliasArn": "brn:bce:cfc:bj:1a2cbf55b97ac8a7c760c4177db4e17d:function:111:ceshi",
      "FunctionName": "111",
      "FunctionVersion": "$LATEST",
      "Name": "ceshi",
      "Description": "ceshi",
      "Uid": "42f6fbc2cd374bfc80d9967370fd8ff",
      "UpdatedAt": "2018-01-23T17:32:11+08:00",
      "CreatedAt": "2018-01-23T17:32:11+08:00"
    },
    {
      "AliasBrn": "brn:bce:cfc:bj:1a2cbf55b97ac8a7c760c4177db4e17d:function:111:ceshi1",
      "AliasArn": "brn:bce:cfc:bj:1a2cbf55b97ac8a7c760c4177db4e17d:function:111:ceshi1",
      "FunctionName": "111",
      "FunctionVersion": "$LATEST",
      "Name": "ceshi1",
      "Description": "ceshi",
      "Uid": "42f6fbc2cd374bfc80d9967370fd8ff",
      "UpdatedAt": "2018-01-23T17:42:02+08:00",
      "CreatedAt": "2018-01-23T17:42:02+08:00"
    }
  ]
}
```

创建别名CreateAlias

接口描述

- 本接口用于创建别名。

请求结构

```
POST /v1/functions/{FunctionName}/aliases HTTP/1.1
Host: cfc.bj.baidubce.com
Authorization: authorization string

{
  "FunctionVersion": "string",
  "Name": "string",
  "Description": "string"
}
```

请求头域

除公共头域外，无其它特殊头域。

请求参数

参数名称	类型	是否必需	参数位置	描述
FunctionName	String	是	path参数	要创建的CFC函数的别名。注意，长度限制只适用于BRN。如果只指定函数名，则长度限制为64个字符。
FunctionVersion	String	是	body参数	别名指向函数版本 (\$LATEST [0-9]+)
Name	String	是	body参数	别名名称 (?![0-9]+\$)([a-zA-Z0-9- _]+)
Description	string	否	body参数	别名描述

响应头域

除公共头域，无其它特殊头域。

响应参数

见[AliasConfiguration](#)

请求示例

```
POST /v1/functions/testHelloWorld/aliases HTTP/1.1
Host: cfc.bj.baidubce.com
Authorization: authorization string

{
  "Description": "xxxxxxx",
  "FunctionVersion": "$LATEST",
  "Name": "helloFly001"
}
```

响应示例

```
HTTP/1.1 200 OK
Content-type: application/json

{
  "AliasBrn": "brn:bce:cfc:bj:cd64f99c69d7c404b61de0a4f1865834:function:testHelloWorld:helloFly001",
  "AliasArn": "brn:bce:cfc:bj:cd64f99c69d7c404b61de0a4f1865834:function:testHelloWorld:helloFly001",
  "FunctionName": "testHelloWorld",
  "FunctionVersion": "$LATEST",
  "Name": "helloFly001",
  "Description": "xxxxxxx",
  "Uid": "df391b08c64c426a81645468c75163a5",
  "UpdatedAt": "2019-03-15T16:38:09.85018412+08:00",
  "CreatedAt": "2019-03-15T16:38:09.85018412+08:00"
}
```

获取别名信息GetAlias

接口描述

- 本接口用于查询别名详情。

请求结构

```
GET /v1/functions/{FunctionName}/aliases/{AliasName} HTTP/1.1
Host: cfc.bj.baidubce.com
Authorization: authorization string
```

请求头域

除公共头域外，无其它特殊头域。

请求参数

参数名称	类型	是否必需	参数位置	描述
AliasName	String	是	path参数	别名名称
FunctionName	String	是	path参数	函数名称

响应头域

除公共头域，无其它特殊头域。

响应参数

见[AliasConfiguration](#)

请求示例

```
GET /v1/functions/testHelloWorld/aliases HTTP/1.1
Host: cfc.bj.baidubce.com
Authorization: authorization string
```

响应示例

```
HTTP/1.1 200 OK
Content-type: application/json

{
  "AliasBrn": "brn:bce:cfc:bj:1a2cbf55b97ac8a7c760c4177db4e17d:function:111:ceshi",
  "AliasArn": "brn:bce:cfc:bj:1a2cbf55b97ac8a7c760c4177db4e17d:function:111:ceshi",
  "FunctionName": "111",
  "FunctionVersion": "$LATEST",
  "Name": "ceshi",
  "Description": "ceshiIIIIII",
  "Uid": "42f6fbc2cd374bfc80d9967370fd8ff",
  "UpdatedAt": "2018-01-23T17:32:10.826183381+08:00",
  "CreatedAt": "2018-01-23T17:32:10.826183381+08:00"
}
```

更新别名信息UpdateAlias

接口描述

- 本接口用于修改别名。

请求结构

```
PUT /v1/functions/{FunctionName}/aliases/{AliasName}/ HTTP/1.1
Host: cfc.bj.baidubce.com
Authorization: authorization string

{
  "FunctionVersion": "string",
  "Description": "string"
}
```

请求头域

除公共头域外，无其它特殊头域。

请求参数

参数名称	类型	是否必需	参数位置	描述
FunctionName	String	是	path参数	函数名称
AliasName	String	是	path参数	别名名称
FunctionVersion	String	是	body参数	别名指向函数版本((\$LATEST [0-9]+)
Description	string	否	body参数	别名描述

响应头域

除公共头域，无其它特殊头域。

响应参数

见[AliasConfiguration](#)

请求示例

```
PUT /v1/functions/testHelloWorld/aliases/alias-2019-01-17T174730 HTTP/1.1
Host: cfc.bj.baidubce.com
Authorization: authorization string

{
  "Description": "HHxxHHxx",
  "FunctionVersion": "$LATEST"
}
```

响应示例

```
HTTP/1.1 200 OK
Content-type: application/json

{
  "AliasBrn": "brn:bce:cfc:bj:cd64f99c69d7c404b61de0a4f1865834:function:testHelloWorld:alias-2019-01-17T174730",
  "AliasArn": "brn:bce:cfc:bj:cd64f99c69d7c404b61de0a4f1865834:function:testHelloWorld:alias-2019-01-17T174730",
  "FunctionName": "testHelloWorld",
  "FunctionVersion": "$LATEST",
  "Name": "alias-2019-01-17T174730",
  "Description": "HHxxHHxx",
  "Uid": "df391b08c64c426a81645468c75163a5",
  "UpdatedAt": "2019-03-15T16:45:40+08:00",
  "CreatedAt": "2019-01-17T17:47:30+08:00"
}
```

删除别名信息DeleteAlias

接口描述

- 本接口用于删除别名。

请求结构

```
DELETE /v1/functions/{FunctionName}/aliases/{AliasName} HTTP/1.1
Host: cfc.bj.baidubce.com
Authorization: authorization string
```

请求头域

除公共头域外，无其它特殊头域。

请求参数

参数名称	类型	是否必需	参数位置	描述
FunctionName	String	是	Path参数	函数名称
AliasName	String	是	Path参数	别名名称

响应头域

除公共头域，无其它特殊头域。

响应参数

无响应参数。

请求示例

```
DELETE /v1/functions/testHelloWorld/aliases/alias-2019-01-17T174730 HTTP/1.1
Host: cfc.bj.baidubce.com
Authorization: authorization string
```

响应示例

```
HTTP/1.1 204
Content-type: application/json
```

触发器操作

获取触发器列表ListTriggers

接口描述

- 本接口用于获取触发器列表。

请求结构

```
GET /v1/relation HTTP/1.1
Host: cfc.bj.baidubce.com
Authorization: authorization string
```

请求头域

除公共头域外，无其它特殊头域。

请求参数

参数名称	类型	是否必需	参数位置	描述
FunctionBrn	String	是	Query参数	函数BRN

响应头域

除公共头域，无其它特殊头域。

响应参数

参数名称	类型	描述
Relation	Relation 数组	Relation 数组

请求示例

```
GET /v1/relation?FunctionBrn=brn:bce:cfc:bj:cd64f99c69d7c404b61de0a4f1865834:function:helloCFCDocs:$LATEST HTTP/1.1
Host: cfc.bj.baidubce.com
Authorization: authorization string
```

响应示例

```
HTTP/1.1 200 OK
Content-type: application/json

{
  "Relation": [
    {
      "RelationId": "brn:bce:cfc-http-
trigger:bj:cd64f99c69d7c404b61de0a4f1865834:b8542048977633ad0a867aefc33fd32a/cfc/GET/cfc/docs",
      "Sid": "cfc-dfbd5359-1afb-49e7-81d7-16554056c79d",
      "Source": "cfc-http-trigger/v1/CFCAPI",
      "Target": "brn:bce:cfc:bj:cd64f99c69d7c404b61de0a4f1865834:function:helloCFCDocs:$LATEST",
      "Data": {
        "AuthType": "anonymous",
        "Brn": "brn:bce:cfc-http-
trigger:bj:cd64f99c69d7c404b61de0a4f1865834:b8542048977633ad0a867aefc33fd32a/cfc/GET/cfc/docs",
        "EndpointPrefix": "https://6ewfn1337kndc.cfc-execute.bj.baidubce.com",
        "Method": "GET",
        "ResourcePath": "/cfc/docs"
      }
    },
    {
      "RelationId": "brn:bce:cfc-http-
trigger:bj:cd64f99c69d7c404b61de0a4f1865834:b8542048977633ad0a867aefc33fd32a/cfc/POST,PUT/cfc/docs/edit"
      ,
      "Sid": "cfc-dfbd5359-1afb-49e7-81d7-16554056c79d",
      "Source": "cfc-http-trigger/v1/CFCAPI",
      "Target": "brn:bce:cfc:bj:cd64f99c69d7c404b61de0a4f1865834:function:helloCFCDocs:$LATEST",
      "Data": {
        "AuthType": "anonymous",
        "Brn": "brn:bce:cfc-http-
trigger:bj:cd64f99c69d7c404b61de0a4f1865834:b8542048977633ad0a867aefc33fd32a/cfc/POST,PUT/cfc/docs/edit"
        ,
        "EndpointPrefix": "https://6ewfn1337kndc.cfc-execute.bj.baidubce.com",
        "Method": "POST,PUT",
        "ResourcePath": "/cfc/docs/edit"
      }
    }
  ]
}
```

创建触发器CreateTrigger

接口描述

- 本接口用于创建触发器。

请求结构


```
POST /v1/relation HTTP/1.1
Host: cfc.bj.baidubce.com
Authorization: authorization string

{
  "Target": "string",
  "Source": "string",
  "Data": {
    "config1": "string",
    "config2": "string"
  }
}
```

请求头域

除公共头域外，无其它特殊头域。

请求参数

参数名称	类型	是否必需	参数位置	描述
Target	String	是	Body参数	函数BRN
Source	SourceType	是	Body参数	触发源
Data	RelationConfiguration	否	Body参数	触发器参数配置

响应头域

除公共头域，无其它特殊头域。

响应参数

参数名称	类型	描述
Relation	Relation	Relation

请求示例

```
POST /v1/relation HTTP/1.1
Host: cfc.bj.baidubce.com
Authorization: authorization string

{
  "Data": {
    "AuthType": "anonymous",
    "Method": "GET",
    "ResourcePath": "helloFly003"
  },
  "Source": "cfc-http-trigger/v1/CFCAPI",
  "Target": "brn:bce:cfc:bj:cd64f99c69d7c404b61de0a4f1865834:function:helloCFCDocs:$LATEST"
}
```

响应示例

```
HTTP/1.1 201 OK
Content-type: application/json

{
  "Relation": {
    "Sid": "cfc-c53bef4e-2cac-4bc6-84c1-2d5ec2f8bec2",
    "RelationId" : "00457f0b-20d8-4f3d-8555-c2u121f38313",
    "Source": "string",
    "Target": "brn:bce:cfc:bj:640c8817bd1de2928d47256dd0620ce5:function:test:$LATEST",
    "Data": {
      "EventType":["PutObject", "PostObject"],
      "Prefix":"images/",
      "Suffix": ".jpg",
      "Status":"enabled"
    }
  }
}
```

更新触发器UpdateTrigger

接口描述

- 本接口用于更新触发器。

请求结构

```
PUT /v1/relation HTTP/1.1
Host: cfc.bj.baidubce.com
Authorization: authorization string

{
  "RelationId": "string",
  "Target": "string",
  "Source": "string",
  "Data": {
    "config1": "string",
    "config2": "string"
  }
}
```

请求头域

除公共头域外，无其它特殊头域。

请求参数

参数名称	类型	是否必需	参数位置	描述
RelationId	String	是	Body参数	触发器Id
Target	String	是	Body参数	函数BRN
Source	SourceType	是	Body参数	触发源
Data	RelationConfiguration	否	Body参数	触发器参数配置

响应头域

除公共头域，无其它特殊头域。

响应参数

参数名称	类型	描述
Relation	Relation	Relation

请求示例

```
PUT /v1/relation HTTP/1.1
Host: cfc.bj.baidubce.com
Authorization: authorization string

{
  "Data": {
    "AuthType": "anonymous",
    "Method": "GET",
    "ResourcePath": "helloFly003"
  },
  "RelationId": "brn:bce:cfc-http-trigger:bj:cd64f99c69d7c404b61de0a4f1865834:b8542048977633ad0a867aefc33fd32a/cfc/GET/helloFly003",
  "Source": "cfc-http-trigger/v1/CFCAPI",
  "Target": "brn:bce:cfc:bj:cd64f99c69d7c404b61de0a4f1865834:function:helloCFCDocs:$LATEST"
}
```

响应示例

```
HTTP/1.1 200
Content-type: application/json

{
  "Relation": {
    "RelationId": "brn:bce:cfc-http-trigger:bj:cd64f99c69d7c404b61de0a4f1865834:b8542048977633ad0a867aefc33fd32a/cfc/GET/helloFly003",
    "Sid": "",
    "Source": "cfc-http-trigger/v1/CFCAPI",
    "Target": "brn:bce:cfc:bj:cd64f99c69d7c404b61de0a4f1865834:function:helloCFCDocs:$LATEST",
    "Data": {
      "AuthType": "anonymous",
      "Method": "GET",
      "ResourcePath": "helloFly003"
    }
  }
}
```

删除触发器DeleteTrigger

接口描述

- 本接口用于删除触发器。

请求结构

```
DELETE /v1/relation HTTP/1.1
Host: cfc.bj.baidubce.com
Authorization: authorization string
```

请求头域

除公共头域外，无其它特殊头域。

请求参数

参数名称	类型	是否必需	参数位置	描述
Target	String	是	Query参数	函数BRN
Source	SourceType	是	Query参数	触发源
RelationId	String	是	Query参数	触发器Id

响应头域

除公共头域，无其它特殊头域。

响应参数

无响应参数。

请求示例

```
DELETE /v1/relation?
Target=brn:bce:cfc:bj:cd64f99c69d7c404b61de0a4f1865834:function:helloCFCDocs:$LATEST&Source=cfc-http-
trigger%2Fv1%2FCFCAPI&RelationId=brn:bce:cfc-http-
trigger:bj:cd64f99c69d7c404b61de0a4f1865834:b8542048977633ad0a867aefc33fd32a%2Ffcdc%2FGET%2FhelloFly00
3 HTTP/1.1
Host: cfc.bj.baidubce.com
Authorization: authorization string
```

响应示例

```
HTTP/1.1 204
Content-type: application/json
```

workflows操作

开始执行StartExecution

接口描述

- 本接口用于用户工作流开始执行。

请求结构

```
POST /v1/execution/start HTTP/1.1
Host: xflow.bj.baidubce.com
Authorization: authorization string
```

请求头域

除公共头域外，无其它特殊头域。

请求参数

参数名称	类型	是否必须	参数位置	描述
flowName	string	是	Body参数	工作流名称
executionName	string	否	Body参数	执行工作流的名称
input	string	是	Body参数	输入数据

响应头域

除公共头域，无其它特殊头域。

参数名称	类型	描述
input	string	执行的输入数据
output	string	执行的输出信息，若还未执行完则为空字符串
startedTime	int	执行的开始时间戳
stoppedTime	int	执行的结束时间戳，若未执行完则为0
flowName	string	执行所属的工作流的名称
name	string	执行的名称
status	string	执行的状态，可能的值有Fail/Succeed/Running/Stopped/Timeout
flowDefinition	string	执行的工作流的定义

请求示例

```
POST /v1/execution/start HTTP/1.1
Authorization: Authorization string
Host: xflow.bj.baidubce.com

{
  "flowName": "comb-test",
  "executionName": "run_WorkFlow",
  "input": "{ \"execDemo\": true }"
}
```

响应示例

```
HTTP/1.1 200
Content-type: application/json

{
  "flowDefinition": "\nname: demo\nstart: makeData\nstates:\n  - type: pass\n    name: makeData\n    data:\n      hello: world\n    next: demoSwitch\n  - type: switch\n    name: demoSwitch\n    description: \"using jq expression to decide whether to exec demo\"\n    conditions:\n      - condition: .execDemo == true\n        next: startExec\n      - condition: .execDemo == false\n        next: stop\n    default:\n      next: startExec\n  - type: parallel\n    name: startExec\n    next: succ\n  branches:\n    - start: wait1s\n      states:\n        - type: wait\n          name: wait1s\n          seconds: 1\n      next: throwSucceed\n    - type: succeed\n      name: throwSucceed\n      - start: wait3s\n        states:\n          - type: wait\n            name: wait3s\n            seconds: 3\n        next: invokeCfc\n      - type: operation\n        name: invokeCfc\n        resource: \"brn:bce:cfc:bj:7c54a6e14823a11c:function:test:$LATEST\"\n        end: true\n      retry:\n        - errors:\n            - XFlow.ALL\n          intervalSeconds: 2\n          maxAttempts: 1\n        catch:\n          - errors:\n              - XFlow.ALL\n            next: errorHandler\n          - type: pass\n            name: errorHandler\n            end: true\n          - type: fail\n            name: stop\n            cause: \"stop demo\"\n          - type: succeed\n            name: succ",
  "flowName": "comb-test",
  "input": "{ \"execDemo\": true }",
  "name": "run_WorkFlow",
  "output": "",
  "startedTime": 1640608416307,
  "status": "Running",
  "stoppedTime": 0
}
```

停止执行StopExecution

接口描述

- 本接口用于用户停止执行工作流。

请求结构

```
POST /v1/execution/stop HTTP/1.1
Host: xflow.bj.baidubce.com
Authorization: authorization string
Content-Type: application/json
```

请求头域

除公共头域外，无其它特殊头域。

请求参数

参数名称	类型	是否必须	参数位置	描述
flowName	string	是	Body参数	工作流名称
executionName	string	是	Body参数	执行工作流的名称

响应头域

除公共头域，无其它特殊头域。

响应参数

参数名称	类型	描述
input	string	执行的输入数据
output	string	执行的输出信息，若还未执行完则为空字符串
startedTime	int	执行的开始时间戳
stoppedTime	int	执行的结束时间戳，若未执行完则为0
flowName	string	执行所属的工作流的名称
name	string	执行的名称
status	string	执行的状态，可能的值有Fail/Succeed/Running/Stopped/Timeout
flowDefinition	string	执行的工作流的定义

请求示例

```
POST /v1/execution/stop HTTP/1.1
Authorization: Authorization string
Host: xflow.bj.baidubce.com

{
  "flowName": "comb-test",
  "executionName": "run_WorkFlow"
}
```

响应示例

```
HTTP/1.1 200
Content-type: application/json

{
  "flowDefinition": "\nname: demo\nstart: makeData\nstates:\n - type: pass\n  name: makeData\n  data:\n    hello: world\n  next: demoSwitch\n - type: switch\n  name: demoSwitch\n  description: \"using jq expression to decide whether to exec demo\"\n  conditions:\n    - condition: .execDemo == true\n    next: startExec\n  - condition: .execDemo == false\n    next: stop\n  default:\n    next: startExec\n - type: parallel\n  name: startExec\n  next: succ\n  branches:\n    - start: wait1s\n    states:\n      - type: wait\n        name: wait1s\n        seconds: 1\n    next: throwSucceed\n  - type: succeed\n    name: throwSucceed\n  - start: wait3s\n    states:\n      - type: wait\n        name: wait3s\n        seconds: 3\n    next: invokeCfc\n  - type: operation\n    name: invokeCfc\n    resource: \"brn:bce:cfc:bj:7c54a6e14823a11c:function:test:$LATEST\"\n    end: true\n  retry:\n    - errors:\n      - XFlow.ALL\n    intervalSeconds: 2\n    maxAttempts: 1\n  catch:\n    - errors:\n      - XFlow.ALL\n    next: errorHandler\n  - type: pass\n    name: errorHandler\n    end: true\n  - type: fail\n    name: stop\n    cause: \"stop demo\"\n  - type: succeed\n    name: succ",
  "flowName": "comb-test",
  "input": "{\n  \"execDemo\": true\n}",
  "name": "test",
  "output": "",
  "startedTime": 1640608416307,
  "status": "Running",
  "stoppedTime": 0
}
```

获取执行列表ListExecutions

接口描述

- 本接口用于查询用户 workflow 执行列表。

请求结构

```
GET /v1/executions HTTP/1.1
Host: xflow.bj.baidubce.com
Authorization: authorization string
```

请求头域

除公共头域外，无其它特殊头域。

请求参数

参数名称	类型	是否必须	参数位置	描述
flowName	string	是	Query参数	工作流名称

响应头域

除公共头域，无其它特殊头域。

响应参数

参数名称	类型	描述
executions	Execution数组	所有执行的信息
total	number	总数

请求示例

```
GET /v1/executions?flowName=testWorkFlow HTTP/1.1
Authorization: Authorization string
Host: xflow.bj.baidubce.com
```

响应示例

```
HTTP/1.1 200
Content-type: application/json

{
  "total":10,
  "executions":[
    {
      "output":"",
      "input":"exercitation irure est voluptate anim",
      "startedTime":1679389309000,
      "stoppedTime":1679389312000,
      "flowName":"comb-test",
      "name":"test",
      "status":"Running",
      "flowDefinition": "\nname: demo\nstart: makeData\nstates:\n - type: pass\n  name: makeData\n  data:\nhello: world\n  next: demoSwitch\n - type: switch\n  name: demoSwitch\n  description: \"using jq expression to\n  decide whether to exec demo\"\n  conditions:\n    - condition: .execDemo == true\n      next: startExec\n    -\ncondition: .execDemo == false\n      next: stop\n  default:\n    next: startExec\n - type: parallel\n  name:\nstartExec\n  next: succ\n  branches:\n    - start: wait1s\n      states:\n        - type: wait\n          name: wait1s\nseconds: 1\n      next: throwSucceed\n    - type: succeed\n      name: throwSucceed\n    - start: wait3s\n      states:\n        - type: wait\n          name: wait3s\nseconds: 3\n      next: invokeCfc\n    - type:\noperation\n      name: invokeCfc\n      resource: \"brn:bce:cfc:bj:7c54a6e14823a11c:function:test:$LATEST\"\nend: true\n      retry:\n        - errors:\n          - XFlow.ALL\n          intervalSeconds: 2\nmaxAttempts: 1\n      catch:\n        - errors:\n          - XFlow.ALL\n      next: errorHandler\n    - type: pass\n      name: errorHandler\n      end: true\n    - type: fail\n      name: stop\n      cause: \"stop demo\"\n    - type: succeed\n      name: succ"
    }
  ]
}
```

获取执行信息DescribeExecution

接口描述

- 本接口用于查询用户工作流下某个执行的情况。

请求结构

```
GET /v1/execution HTTP/1.1
Host: xflow.bj.baidubce.com
Authorization: authorization string
```

请求头域

除公共头域外，无其它特殊头域。

请求参数

参数名称	类型	是否必须	参数位置	描述
flowName	string	是	Query参数	工作流名称
executionName	string	是	Query参数	某次执行的名字

响应头域

除公共头域，无其它特殊头域。

响应参数

参数名称	类型	描述
input	string	执行的输入数据
output	string	执行的输出信息，若还未执行完则为空字符串
startedTime	int	执行的开始时间戳
stoppedTime	int	执行的结束时间戳，若未执行完则为0
flowName	string	执行所属的工作流的名称
name	string	执行的名称
status	string	执行的状态，可能的值有Fail/Succeed/Running/Stopped/Timeout
flowDefinition	string	执行的工作流的定义

请求示例

```
GET /v1/execution?flowName=testWorkFlow&executionName=testExecutionName HTTP/1.1
Authorization: Authorization string
Host: xflow.bj.baidubce.com
```

响应示例

```
HTTP/1.1 200
Content-type: application/json

{
  "output": "",
  "input": "exercitation irure est voluptate anim",
  "startedTime": 1679389309000,
  "stoppedTime": 1679389312000,
  "flowName": "comb-test",
  "name": "test",
  "status": "Running",
  "flowDefinition": "\nname: demo\nstart: makeData\nstates:\n  - type: pass\n    name: makeData\n    data:\n      hello: world\n    next: demoSwitch\n  - type: switch\n    name: demoSwitch\n    description: \"using jq expression to decide whether to exec demo\"\n    conditions:\n      - condition: .execDemo == true\n        next: startExec\n      - condition: .execDemo == false\n        next: stop\n    default:\n      next: startExec\n  - type: parallel\n    name: startExec\n    next: succ\n  branches:\n    - start: wait1s\n      states:\n        - type: wait\n          name: wait1s\n          seconds: 1\n        next: throwSucceed\n      - type: succeed\n        name: throwSucceed\n      - start: wait3s\n        states:\n          - type: wait\n            name: wait3s\n            seconds: 3\n        next: invokeCfc\n      - type: operation\n        name: invokeCfc\n        resource: \"brn:bce:cfc:bj:7c54a6e14823a11c:function:test:$LATEST\"\n        end: true\n      retry:\n        - errors:\n            - XFlow.ALL\n          intervalSeconds: 2\n          maxAttempts: 1\n        catch:\n          - errors:\n              - XFlow.ALL\n            next: errorHandler\n          - type: pass\n            name: errorHandler\n            end: true\n          - type: fail\n            name: stop\n            cause: \"stop demo\"\n          - type: succeed\n            name: succ"
}
```

获取执行历史GetExecutionHistory

接口描述

- 本接口用于查询用户 workflow 执行历史。

请求结构

```
GET /v1/execution/history HTTP/1.1
Host: xflow.bj.baidubce.com
Authorization: authorization string
```

请求头域

除公共头域外，无其它特殊头域。

请求参数

参数名称	类型	是否必须	参数位置	描述
flowName	string	是	Query参数	工作流名称
executionName	string	是	Query参数	执行的名字
limit	int	是	Query参数	读取的历史记录的条数

响应头域

除公共头域，无其它特殊头域。

响应参数

参数名称	类型	描述
events	Event 数组	执行历史列表

请求示例

```
GET /v1/execution/history?flowName=testWorkFlow&executionName=testExecutionName&limit=50 HTTP/1.1
Authorization: Authorization string
Host: xflow.bj.baidubce.com
```

响应示例

```
HTTP/1.1 200
Content-type: application/json

{
  "events":[
    {
      "costTime":86,
      "eventDetail":{"input\":"{}"},
      "eventId":"1",
      "executionId":"c7ac82ae14ef42d1a4ffa3b2eacea17f:comb-test:2a51a4d7-d843-4c0b-a945-d322f359c954",
      "stateName":"",
      "time":1640608954782,
      "type":"ExecutionStarted"
    },
    {
      "costTime":102,
      "eventDetail":{"input\":"{}"},
      "eventId":"2",
      "executionId":"c7ac82ae14ef42d1a4ffa3b2eacea17f:comb-test:2a51a4d7-d843-4c0b-a945-d322f359c954",
      "stateName":"makeData",
      "time":1640608954798,
      "type":"StatePassEntered"
    },
    {
      "costTime":116,
      "eventDetail":{"output\":"hello\":"world\"}},
      "eventId":"3",
      "executionId":"c7ac82ae14ef42d1a4ffa3b2eacea17f:comb-test:2a51a4d7-d843-4c0b-a945-d322f359c954",
      "stateName":"makeData",
      "time":1640608954812,
      "type":"StatePassExited"
    },
    {
      "costTime":127,
      "eventDetail":{"input\":"hello\":"world\"}},
      "eventId":"4",
      "executionId":"c7ac82ae14ef42d1a4ffa3b2eacea17f:comb-test:2a51a4d7-d843-4c0b-a945-d322f359c954",
      "stateName":"demoSwitch",
      "time":1640608954823,
      "type":"StateSwitchEntered"
    },
    {
      "costTime":140,
      "eventDetail":{"output\":"hello\":"world\"}},
      "eventId":"5",
      "executionId":"c7ac82ae14ef42d1a4ffa3b2eacea17f:comb-test:2a51a4d7-d843-4c0b-a945-d322f359c954",
      "stateName":"demoSwitch",
      "time":1640608954836,
      "type":"StateSwitchExited"
    },
    {
      "costTime":154,
      "eventDetail":{"input\":"hello\":"world\"}},
```

```
"eventId":"6",
"executionId":"c7ac82ae14ef42d1a4ffa3b2ececa17f:comb-test:2a51a4d7-d843-4c0b-a945-d322f359c954",
"stateName":"startExec",
"time":1640608954850,
"type":"StateParallelEntered"
},
{
  "costTime":165,
  "eventDetail":{"input":{"hello":"world"}},
  "eventId":"7",
  "executionId":"c7ac82ae14ef42d1a4ffa3b2ececa17f:comb-test:2a51a4d7-d843-4c0b-a945-d322f359c954",
  "stateName":"startExec",
  "time":1640608954861,
  "type":"StateParallelStarted"
},
{
  "costTime":181,
  "eventDetail":{"input":{"hello":"world"}},
  "eventId":"8",
  "executionId":"c7ac82ae14ef42d1a4ffa3b2ececa17f:comb-test:2a51a4d7-d843-4c0b-a945-d322f359c954",
  "stateName":"wait1s",
  "time":1640608954877,
  "type":"StateWaitEntered"
},
{
  "costTime":181,
  "eventDetail":{"input":{"hello":"world"}},
  "eventId":"9",
  "executionId":"c7ac82ae14ef42d1a4ffa3b2ececa17f:comb-test:2a51a4d7-d843-4c0b-a945-d322f359c954",
  "stateName":"wait3s",
  "time":1640608954877,
  "type":"StateWaitEntered"
},
{
  "costTime":1196,
  "eventDetail":{"output":{"hello":"world"}},
  "eventId":"10",
  "executionId":"c7ac82ae14ef42d1a4ffa3b2ececa17f:comb-test:2a51a4d7-d843-4c0b-a945-d322f359c954",
  "stateName":"wait1s",
  "time":1640608955892,
  "type":"StateWaitExited"
},
{
  "costTime":1215,
  "eventDetail":{"input":{"hello":"world"}},
  "eventId":"11",
  "executionId":"c7ac82ae14ef42d1a4ffa3b2ececa17f:comb-test:2a51a4d7-d843-4c0b-a945-d322f359c954",
  "stateName":"throwSucceed",
  "time":1640608955911,
  "type":"StateSucceedEntered"
},
{
  "costTime":1231,
  "eventDetail":{"output":{"hello":"world"}},
  "eventId":"12",
  "executionId":"c7ac82ae14ef42d1a4ffa3b2ececa17f:comb-test:2a51a4d7-d843-4c0b-a945-d322f359c954",
  "stateName":"throwSucceed",
  "time":1640608955927,
  "type":"StateSucceedExited"
},
{
  "costTime":3192,
```

```
{
  "eventDetail": "{\\\"output\\\":{\\\"hello\\\":\\\"world\\\"}}",
  "eventId": "13",
  "executionId": "c7ac82ae14ef42d1a4ffa3b2ececa17f:comb-test:2a51a4d7-d843-4c0b-a945-d322f359c954",
  "stateName": "wait3s",
  "time": 1640608957888,
  "type": "StateWaitExited"
},
{
  "costTime": 3207,
  "eventDetail": "{\\\"output\\\":{\\\"hello\\\":\\\"world\\\"}}",
  "eventId": "14",
  "executionId": "c7ac82ae14ef42d1a4ffa3b2ececa17f:comb-test:2a51a4d7-d843-4c0b-a945-d322f359c954",
  "stateName": "startExec",
  "time": 1640608957903,
  "type": "StateParallelSucceed"
},
{
  "costTime": 3220,
  "eventDetail": "{\\\"output\\\":{\\\"hello\\\":\\\"world\\\"}}",
  "eventId": "15",
  "executionId": "c7ac82ae14ef42d1a4ffa3b2ececa17f:comb-test:2a51a4d7-d843-4c0b-a945-d322f359c954",
  "stateName": "startExec",
  "time": 1640608957916,
  "type": "StateParallelExited"
},
{
  "costTime": 3232,
  "eventDetail": "{\\\"output\\\":{}}",
  "eventId": "16",
  "executionId": "c7ac82ae14ef42d1a4ffa3b2ececa17f:comb-test:2a51a4d7-d843-4c0b-a945-d322f359c954",
  "stateName": "",
  "time": 1640608957928,
  "type": "ExecutionSucceed"
}
],
"executionName": "2a51a4d7-d843-4c0b-a945-d322f359c954"
}
```

工作流列表ListFlow

接口描述

- 本接口用于查询用户工作流列表。

请求结构

```
GET /v1/flows HTTP/1.1
Host: xflow.bj.baidubce.com
Authorization: authorization string
```

请求头域

除公共头域外，无其它特殊头域。

请求参数

无

响应头域

除公共头域，无其它特殊头域。

响应参数

参数名称	类型	描述
flows	Flow数组	flow列表

请求示例

```
GET /v1/flows HTTP/1.1
Authorization: Authorization string
Host: xflow.bj.baidubce.com
```

响应示例

```
HTTP/1.1 200
Content-type: application/json

{
  "flows":[
    {
      "createdTime":"2021-12-16T12:43:11+08:00",
      "definition":"name: demo\nstart: initData\nstates:\n - type: pass\n  name: initData\n  data:\n    hello: world\nnext: demoSwitch\n - type: switch\n  name: demoSwitch\n  description: \"using jq expression to decide whether to\nexec demo\"\n  conditions:\n    - condition: .execDemo == true\n    next: startExec\n    - condition: .execDemo ==\nfalse\n    next: stop\n  default:\n    next: startExec\n - type: parallel\n  name: startExec\n  next: succ\nbranches:\n  - start: wait3s\n  states:\n    - type: wait\n    name: wait3s\n    seconds: 3\nend: true\n  - start: invokeCfc\n  states:\n    - type: operation\n    name: invokeCfc\n    resource:\n      \"bm:bce:cfc:bj:7c54a6e14823a11c:function:test:$LATEST\"\n    end: true\n    catch:\n      - errors:\n        - XFlow.ALL\n    next: errorHandler\n    - type: pass\n    name: errorHandler\n    end: true\n - type:\nfail\n  name: stop\n  cause: \"stop demo\"\n - type: succeed\n  name: succ",
      "description":"this is a xflow demo 1",
      "lastModifiedTime":"2021-12-24T13:42:16+08:00",
      "name":"complex-demo",
      "type":"FDL"
    },
    {
      "createdTime":"2021-12-17T15:45:57+08:00",
      "definition":"name: example\nstart: s1\nstates:\n - name: s1\n  type: foreach\n  inputCollection: \"\n[.orders[]]\"\n  iterationParam: order\n  outputCollection: \".results\"\n  iterator:\n  start: s2\n  states:\n - type: operation\n  name: s2\n  resource:\n    \"bm:bce:cfc:su:7c54a6e14823a11c9f9f5345952ac336:function:test-c:$LATEST\"\n    end: true\n    end: true",
      "lastModifiedTime":"2021-12-17T15:45:57+08:00",
      "name":"foreach-example",
      "type":"FDL"
    }
  ]
}
```

创建工作流CreateFlow

接口描述

- 本接口用于创建工作流。

请求结构

```
POST /v1/flow HTTP/1.1
Host: xflow.bj.baidubce.com
Authorization: authorization string
Content-Type: application/json
```

这里注意 host 必须指定为每个地域的 xflow 域名，例如：xflow.bj.baidubce.com

请求头域

除公共头域外，无其它特殊头域。

请求参数

参数名称	类型	是否必需	参数位置	描述
name	string	是	Body参数	工作流的名字，最大长度: 128，最小长度: 1
type	string	是	Body参数	类型，当前为固定值 "FDL"
definition	string	是	Body参数	自定义的 yaml，最大长度10000
description	string	否	Body参数	一个简短的说明 0-128字符

响应头域

除公共头域，无其它特殊头域。

响应参数

参数名称	类型	描述
name	string	工作流的名称，只能由数字、字母、-或_组成，必须以字母或_开头，长度小于64个字符
type	string	类型，当前为固定值 "FDL"
definition	string	满足工作流语法规范的 yaml 定义
description	string	工作流描述，0-128字符
created_at	string	工作流创建时间。 ISO-8601 格式 (YYYY-MM-DDThh:mm:ss.sTZD)
updated_at	string	工作流最新更新时间。 ISO-8601 格式 (YYYY-MM-DDThh:mm:ss.sTZD)

请求示例

```

POST /v1/flow HTTP/1.1
Host: xflow.bj.baidubce.com
Authorization: authorization string
Content-Type: application/json

{
  "name": "testWorkflow",
  "description": "first Workflow",
  "definition": "name: demo\nstart: initData\nstates:\n - type: pass\n  name: initData\n  data:\n    hello: world\nnext: demoSwitch\n - type: switch\n  name: demoSwitch\n  description: \"using jq expression to decide whether to\nexec demo\"\n  conditions:\n    - condition: .execDemo == true\n    next: startExec\n    - condition: .execDemo ==\nfalse\n    next: stop\n  default:\n    next: startExec\n - type: parallel\n  name: startExec\n  next: succ\nbranches:\n  - start: wait3s\n  states:\n    - type: wait\n    name: wait3s\n    seconds: 3\nend: true\n - start: invokeCfc\n  states:\n    - type: operation\n    name: invokeCfc\n    resource:\n      \"brn:bce:cfc:bj:7c54a6e14823a11c:function:test:$LATEST\"\n    end: true\n    catch:\n      - errors:\n        - XFlow.ALL\n    next: errorHandler\n    - type: pass\n    name: errorHandler\n    end: true\n - type:\nfail\n  name: stop\n  cause: \"stop demo\"\n - type: succeed\n  name: succ",
  "type": "FDL"
}

```

返回示例

```

HTTP/1.1 201 Created
Content-Type: application/json

{
  "description": "first Workflow",
  "created_at": "2021-12-27 16:20:08",
  "name": "testWorkflow",
  "updated_at": "2021-12-27 16:23:37",
  "type": "FDL",
  "definition": "name: demo\nstart: initData\nstates:\n - type: pass\n  name: initData\n  data:\n    hello: world\nnext: demoSwitch\n - type: switch\n  name: demoSwitch\n  description: \"using jq expression to decide whether to\nexec demo\"\n  conditions:\n    - condition: .execDemo == true\n    next: startExec\n    - condition: .execDemo ==\nfalse\n    next: stop\n  default:\n    next: startExec\n - type: parallel\n  name: startExec\n  next: succ\nbranches:\n  - start: wait3s\n  states:\n    - type: wait\n    name: wait3s\n    seconds: 3\nend: true\n - start: invokeCfc\n  states:\n    - type: operation\n    name: invokeCfc\n    resource:\n      \"brn:bce:cfc:bj:7c54a6e14823a11c:function:test:$LATEST\"\n    end: true\n    catch:\n      - errors:\n        - XFlow.ALL\n    next: errorHandler\n    - type: pass\n    name: errorHandler\n    end: true\n - type:\nfail\n  name: stop\n  cause: \"stop demo\"\n - type: succeed\n  name: succ"
}

```

更新工作流UpdateFlow

接口描述

- 本接口用于更新指定工作流。

请求结构

```

PUT /v1/flow HTTP/1.1
Host: xflow.bj.baidubce.com
Authorization: authorization string
Content-Type: application/json

```

请求头域

除公共头域外，无其它特殊头域。

请求参数

参数名称	类型	是否必需	参数位置	描述
name	string	是	Body参数	要更新的工作流的名字，最大长度: 128，最小长度: 1
type	string	是	Body参数	类型，当前为固定值"FDL"
definition	string	是	Body参数	满足工作流语法规范的YAML定义，最大长度10000
description	string	否	Body参数	工作流描述信息，最大长度128

响应头域

除公共头域，无其它特殊头域。

响应参数

参数名称	类型	描述
name	string	工作流的名称
workspaceId	string	工作空间id
type	string	类型，当前为固定值"FDL"
definition	string	工作流YAML定义
description	string	工作流描述信息
created_at	string	工作流创建时间。 ISO-8601 格式 (YYYY-MM-DDThh:mm:ss.sTZD)
updated_at	string	工作流更新时间。 ISO-8601 格式 (YYYY-MM-DDThh:mm:ss.sTZD)

请求示例

```
PUT /v1/flow HTTP/1.1
Host: xflow.bj.baidubce.com
Authorization: authorization string

{
  "name": "testWorkFlow",
  "description": "first WorkFlow",
  "definition": "name: demo\nstart: initData\nstates:\n - type: pass\n  name: initData\n  data:\n    hello: world\nnext: demoSwitch\n - type: switch\n  name: demoSwitch\n  description: \"using jq expression to decide whether to\nexec demo\"\n  conditions:\n    - condition: .execDemo == true\n      next: startExec\n    - condition: .execDemo ==\nfalse\n  next: stop\n  default:\n    next: startExec\n - type: parallel\n  name: startExec\n  next: succ\nbranches:\n - start: wait3s\n  states:\n    - type: wait\n      name: wait3s\n      seconds: 3\nend: true\n - start: invokeCfc\n  states:\n    - type: operation\n      name: invokeCfc\n      resource:\n        \"bm:bce:cfc:bj:7c54a6e14823a11c:function:test:$LATEST\"\n      end: true\n      catch:\n        - errors:\n          - XFlow.ALL\n      next: errorHandler\n    - type: pass\n      name: errorHandler\n      end: true\n - type:\nfail\n  name: stop\n  cause: \"stop demo\"\n - type: succeed\n  name: succ",
  "type": "FDL"
}
```

响应示例

```
HTTP/1.1 200
Content-type: application/json

{
  "name": "testWorkflow",
  "updated_at": "2021-12-27 22:44:52",
  "description": "first Workflow",
  "definition": "name: demo\nstart: initData\nstates:\n - type: pass\n  name: initData\n  data:\n    hello: world\nnext: demoSwitch\n - type: switch\n  name: demoSwitch\n  description: \"using jq expression to decide whether to\nexec demo\"\n  conditions:\n    - condition: .execDemo == true\n      next: startExec\n    - condition: .execDemo ==\nfalse\n  next: stop\n  default:\n    next: startExec\n - type: parallel\n  name: startExec\n  next: succ\nbranches:\n  - start: wait3s\n    states:\n      - type: wait\n        name: wait3s\n        seconds: 3\nend: true\n  - start: invokeCfc\n    states:\n      - type: operation\n        name: invokeCfc\n        resource:\n          \"brn:bce:cfc:bj:7c54a6e14823a11c:function:test:$LATEST\"\n        end: true\n        catch:\n          - errors:\n            - XFlow.ALL\n            next: errorHandler\n          - type: pass\n            name: errorHandler\n            end: true\n          - type:\nfail\n  name: stop\n  cause: \"stop demo\"\n - type: succeed\n  name: succ",
  "created_at": "2021-12-27 12:20:27",
  "workspaceId": "default",
  "type": "FDL"
}
```

获取 workflow 信息DescribeFlow

接口描述

- 本接口用于查询用户单个工作流信息。

请求结构

```
GET /v1/flow HTTP/1.1
Host: xflow.bj.baidubce.com
Authorization: authorization string
```

请求头域

除公共头域外，无其它特殊头域。

请求参数

参数名称	类型	是否必需	参数位置	描述
flowName	String	是	Query参数	工作流名称

响应头域

除公共头域，无其它特殊头域。

响应参数

参数名称	类型	描述
name	string	工作流名称
workspaceId	string	工作空间id
type	string	类型，当前为固定值"FDL"
definition	string	工作流的YAML定义
created_at	string	工作流创建时间。 ISO-8601 格式 (YYYY-MM-DDThh:mm:ss.sTZD)
updated_at	string	工作流最新更新时间。 ISO-8601 格式 (YYYY-MM-DDThh:mm:ss.sTZD)
description	string	工作流描述

请求示例

```
GET /v1/flow?flowName=1testWorkFlow HTTP/1.1
Host: xflow.bj.baidubce.com
Authorization: authorization string
```

响应示例

```
HTTP/1.1 200
Content-type: application/json

{
  "name": "testWorkFlow",
  "updated_at": "2021-12-27 22:44:52",
  "description": "first WorkFlow",
  "definition": "name: demo\nstart: initData\nstates:\n - type: pass\n  name: initData\n  data:\n    hello: world\nnext: demoSwitch\n - type: switch\n  name: demoSwitch\n  description: \"using jq expression to decide whether to\nexec demo\"\n  conditions:\n    - condition: .execDemo == true\n      next: startExec\n    - condition: .execDemo ==\nfalse\n      next: stop\n  default:\n    next: startExec\n - type: parallel\n  name: startExec\n  next: succ\nbranches:\n  - start: wait3s\n    states:\n      - type: wait\n        name: wait3s\n        seconds: 3\n    end: true\n  - start: invokeCfc\n    states:\n      - type: operation\n        name: invokeCfc\n        resource:\n          \"brn:bce:cfc:bj:7c54a6e14823a11c:function:test:$LATEST\"\n        end: true\n        catch:\n          - errors:\n            - XFlow.ALL\n            next: errorHandler\n          - type: pass\n            name: errorHandler\n            end: true\n - type:\nfail\n  name: stop\n  cause: \"stop demo\"\n - type: succeed\n  name: succ",
  "created_at": "2021-12-27 12:20:27",
  "workspaceId": "default",
  "type": "FDL"
}
```

删除工作流DeleteFlow

接口描述

- 本接口用于删除工作流。

请求结构

```
DELETE /v1/flow HTTP/1.1
Host: xflow.bj.baidubce.com
Authorization: authorization string
Content-Type: application/x-www-form-urlencoded
```

请求头域

除公共头域外，无其它特殊头域。

请求参数

参数名称	类型	是否必需	参数位置	描述
flowName	String	是	Query参数	工作流名称

响应头域

除公共头域，无其它特殊头域。

响应参数

无响应参数。

请求示例

```
DELETE /v1/flow?flowName=testWorkFlow HTTP/1.1
Host: xflow.bj.baidubce.com
Authorization: authorization string
Content-Type: application/x-www-form-urlencoded
```

响应示例

```
HTTP/1.1 204
Content-type: application/json
```

数据类型

🔗 参数定义

🔗 Code

状态	类型	描述
ZipFile	blob	您要发布的zip包的 base64-encoded 注意zip包压缩目录的内容，而不是目录本身
Publish	Boolean	是否直接发布版本
BosBucket	string	若要从bos上传代码，代码存储的Bos bucket
BosObject	string	若要从bos上传代码，代码在bos中的object

🔗 SourceType

SourceType 可选值	描述
dueros	指定DuerOS触发器
bos/your-bucket-name	指定BOS触发器和bucket
duedge	指定Duedge触发器
cfc-http-trigger/v1/CFCAPI	指定HTTP触发器
cfc-crontab-trigger/v1/	指定Crontab触发器

🔗 RelationConfiguration

RelationConfiguration是以JSON格式表示的触发器配置，每种触发器的配置不同。

BOS

```
"Data": {
  "Resource": "string",
  "Status": "string",
  "EventType": ["string", "string"],
  "Name": "string"
}
```

参数名称	类型	是否必须	描述
Resource	string	是	订阅的资源
Status	string	是	可赋值为"enabled"或"disabled"
EventType	list of string	是	BOS事件类型
Name	string	是	触发器名称

EventType可选值	描述
PutObject	创建/覆盖文件：简单上传
PostObject	创建/覆盖文件：表单上传
AppendObject	创建/覆盖文件：追加上传
CopyObject	创建/覆盖文件：拷贝上传
CompleteMultipartObject	创建/覆盖文件：完成分片上传

DuerOS

无需指定"Data"参数。

Duedge

无需指定"Data"参数。

HTTP

```
"Data": {
  "ResourcePath": "string",
  "Method": "string",
  "AuthType": "string"
}
```

参数名称	类型	是否必须	描述
ResourcePath	string	是	URL路径
Method	string	是	HTTP方法，如"GET,HEAD"
AuthType	string	是	身份验证类型，可选值为"anonymous"或"iam"

CDN

```
"Data": {
  "Domains": ["string", "string"],
  "EventType": "string",
  "Remark": "string",
  "Status": "string"
}
```

参数名称	类型	是否必须	描述
EventType	string	是	CDN事件类型
Domains	list of string	否	域名列表，事件类型为域名粒度时可添加此参数
Remark	string	否	备注信息
Status	string	是	是否启用触发器，可选值为"enabled"或"disabled"

EventType可选值	描述	事件粒度
CachedObjectsBlocked	发现违禁资源	用户粒度
CachedObjectsPushed	预热资源完成	用户粒度
CachedObjectsRefreshed	刷新资源完成	用户粒度
CdnDomainCreated	创建域名成功	用户粒度
CdnDomainDeleted	删除域名成功	用户粒度
LogFileCreated	产生域名日志	域名粒度、用户粒度
CdnDomainStarted	启动域名成功	域名粒度、用户粒度
CdnDomainStopped	停止域名成功	域名粒度、用户粒度

Crontab

参数名称	类型	是否必须	描述
Brn	string	是	函数Brn
Enabled	string	否	是否启用，Enabled/Disabled
Input	strng	否	输入配置
Name	string	是	触发器名称
ScheduleExpression	string	是	计划表达式

🔗 Model对象定义

🔗 FunctionConfiguration

参数名称	类型	描述
Uid	string	用户ID（由数字、字母、下划线组成），长度限制128字符。
Description	string	一个简短的说明，0-256字符。
FunctionBrn	string	函数的唯一性资源标志符，1-170字符。
Region	string	区域，现有 bj（北京）、su（苏州）、gz（广州）。
Timeout	int	超时时间 1-300，最大300。
VersionDesc	string	版本描述，0-256字符。
UpdatedAt	string	函数最新更新时间。 ISO-8601 格式 (YYYY-MM-DDThh:mm:ss.sTZD)。
LastModified	string	同UpdatedAt，兼容AWS Lambda。
CodeSha256	string	函数代码包的SHA256散列。
CodeSize	int	函数包的大小，单位bytes。
FunctionArn	string	同FunctionBrn，兼容AWS Lambda。
FunctionName	string	函数的名称，由数字、字母、-或_组成。长度限制为64个字符。
Handler	string	CFC调用的入口函数，对于node为module-name.export eg. index.handler 最大长度为128字符。
Version	string	版本。\$LATEST表示最新，否则由数字组成。1-32字符。
Runtime	string	运行语言。nodejs6.11、nodejs8.5、python2、java8。
MemorySize	int	内存的大小，以MB为单位，CFC使用此内存大小来推断分配给您的函数的CPU和内存数量。默认值是128MB，必须是128MB的倍数。(现在CFC提供128至1024M的内存)。如果是Java8运行环境，内存大小为512至1024M。
Environment		Environment
CommitId	string	单次提交的唯一标志。
Role	string	函数的执行者角色。
LogType	string	日志类型 bos（对象存储） / none。
LogBosDir	string	日志存储的 Bucket 路径。

 CodeStorage

参数名称	类型	描述
Location	string	函数代码存储的地址。
RepositoryType	string	函数代码类型。

Environment

状态	类型	描述
Variables	json	环境变量参数。

AliasConfiguration

参数名称	类型	描述
AliasBrn	string	别名的唯一性资源标志符。
AliasArn	string	同AliasBrn，兼容AWS Lambda。
FunctionName	string	函数的名称，由数字、字母、-或_组成。长度限制为64个字符。
FunctionVersion	string	函数版本。\$LATEST表示最新，否则由数字组成。1-32字符。
Name	string	别名名称。1-128字符。
Description	string	一个简短的说明，0-256字符。
Uid	string	用户ID（由数字、字母、下划线组成），长度限制128字符。
UpdatedAt	string	别名最新更新时间。 ISO-8601 格式 (YYYY-MM-DDThh:mm:ss.sTZD)。
CreatedAt	string	别名创建时间。 ISO-8601 格式 (YYYY-MM-DDThh:mm:ss.sTZD)。

Relation

参数名称	类型	描述
RelationId	string	触发器ID。
Sid	string	关联的策略的statement-id。
Source	string	触发源。
Target	string	触发函数BRN。
Data	RelationConfiguration	

VpcConfig

参数名称	类型	是否必须	描述
VpcId	string	是	要绑定的VpcID
SubnetIds	list of string	是	绑定的子网列表
SecurityGroupIds	list of string	是	绑定的安全组列表

Execution

参数名称	类型	描述
name	string	执行的名称
flowName	string	工作流的名字
input	string	输入
startedTime	number	开始时间
output	string	输出
status	string	状态
stoppedTime	number	停止时间
flowDefinition	string	Yaml定义的字符串

🔗 Event

参数名称	类型	描述
eventId	string	事件的名称
type	string	事件类型
stateName	string	状态名字
eventDetail	string	详细事件信息
time	integer	时间
costTime	integer	花费的时间

🔗 Flow

参数名称	类型	描述
name	string	事件的名称
type	string	事件类型
definition	string	Yaml定义的字符串
description	string	描述
createdTime	integer	创建时间
lastModifiedTime	integer	最后修改时间

SDK

Java SDK

概述

本文档主要介绍CFC Java SDK的安装和使用。在使用本文档前，您需要先了解CFC的一些基本知识，并且已经开通了CFC服务。若您还不了解CFC，可以参考[产品描述](#)和[快速入门](#)。

安装SDK工具包

🔗 运行环境

Java SDK 工具包可在JDK1.7、JDK1.8环境下运行。

🔗 安装SDK

方式一：使用Maven安装

在Maven的pom.xml文件中追加bce-java-sdk的依赖：

```
<dependency>
  <groupId>com.baidubce</groupId>
  <artifactId>bce-java-sdk</artifactId>
  <version>{version}</version>
</dependency>
```

其中，{version}为版本号，可以在[SDK下载页面找到](#)。

方式二：直接使用JAR包安装

1. 下载最新版[Java SDK](#)压缩工具包。
2. 将下载的 bce-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 lib/bce-java-sdk-version.jar 和第三方依赖工具包 third-party/*.jar。其中version为版本号。

SDK目录

```
com.baidubce
├── auth                //BCE签名相关类
├── http                //BCE的Http通信相关类
├── internal            //SDK内部类
├── model               //BCE公用model类
├── services
│   └── cfc             //CFC服务相关类
│       ├── model       //CFC内部model，如Request或Response
│       ├── CfcClient.class //CFC客户端入口类
│       ├── CfcInputStream.class //CFC解析HTTP请求响应
│       └── CfcResponseHandler.class //CFC特有的Response Handler
├── util                //BCE公用工具类
├── BceClientConfiguration.class //对BCE的HttpClient的配置
├── BceClientException.class //BCE客户端的异常类
├── BceServiceException.class //与BCE服务端交互后的异常类
├── ErrorCode.class     //BCE通用的错误码
└── Region.class        //BCE提供服务的区域
```

初始化

确认Endpoint

在确认您使用SDK时配置的Endpoint时，可先阅读API参考中关于[CFC域名服务](#)的部分，理解Endpoint相关概念。

目前CFC支持“华北-北京”：<http://cfc.bj.baidubce.com>、“华南-广州”：<http://cfc.gz.baidubce.com>、“华东-苏州”：<http://cfc.su.baidubce.com>三个区域。对应信息为：

访问区域	对应Endpoint
BJ	cfc.bj.baidubce.com
GZ	cfc.gz.baidubce.com
SU	cfc.su.baidubce.com

CFC访问域名支持HTTP和HTTPS两种调用方式，为了提升数据的安全性，建议通过HTTPS调用。更多关于HTTPS保障传输安全的使用请参考[HTTPS传输加密实践](#)。

获取密钥

要使用百度智能云CFC，您需要拥有一个有效的AK (Access Key ID)和SK (Secret Access Key)用来进行签名认证。AK/SK是由系统分配给用户的，均为字符串，用于标识用户，为访问CFC做签名验证。

可以通过如下步骤了解并获得您的AK/SK信息：

[注册百度智能云账号](#)

[创建AK/SK](#)

CfcClient

CfcClient是CFC服务的客户端，为开发者与CFC服务进行交互提供了一系列的方法，包括函数调用、函数创建、获取函数列表等。

新建CfcClient

通过AK/SK方式访问CFC

通过AK/SK方式访问CFC，用户可以参考如下代码新建一个CfcClient:

```
public class Sample {
public static void main(String[] args){
    String endPoint = "https://cfc.bj.baidubce.com";    //CFC服务端接口地址
    String accessKeyId = "your-access-key-id";        //用户的Access Key ID
    String secretAccessKey = "your-secret-access-key"; //用户的Secret Access Key

    BceClientConfiguration config = new BceClientConfiguration()
        .withCredentials(new DefaultBceCredentials(accessKeyId, secretAccessKey))
        .withEndpoint(endPoint);
    CfcClient cfcClient = new CfcClient(config);
}
}
```

在上面的代码中，变量accessKeyId与secretAccessKey是由系统分配给用户的，均为字符串，用于标识用户，为访问CFC做签名验证。其中accessKeyId对应控制台中的"Access Key ID"，secretAccessKey对应控制台中的"Access Key Secret"，获取方式请参考《操作指南[管理ACCESSKEY](#)》。

用户可以根据需要指定域名，通过endPoint参数来制定。

配置HTTPS协议访问CFC

CFC支持HTTPS传输协议，您可以通过如下两种方式在CFC Java SDK中使用HTTPS访问CFC服务：

- 在endpoint中指明 https:

```
String endPoint = "https://cfc.bj.baidubce.com";
String ak = "ak";
String sk = "sk";
BceClientConfiguration config = new BceClientConfiguration()
    .withCredentials(new DefaultBceCredentials(ak, sk))
    .withEndpoint(endPoint);
CfcClient cfcClient = new CfcClient(config);
```

- 通过调用setProtocol方法设置https协议

```
String endPoint = "bce.bj.baidubce.com"; //endPoint中不包含protocol
String ak = "ak";
String sk = "sk";
BceClientConfiguration config = new BceClientConfiguration();
config.setCredentials(new DefaultBceCredentials(ak, sk));
config.setEndpoint(endPoint);
config.setProtocol(Protocol.HTTPS); //如果不指明，则使用http
CfcClient cfcClient = new CfcClient(config);
```

注意：如果在endPoint中指明了protocol，则endPoint中的生效，另外单独调用setProtocol()不起作用。

```
String endPoint = "https://bce.bj.baidubce.com"; //endPoint中不包含protocol
String ak = "ak";
String sk = "sk";
BceClientConfiguration config = new BceClientConfiguration();
config.setCredentials(new DefaultBceCredentials(ak, sk));
config.setEndpoint(endPoint);
config.setProtocol(Protocol.HTTPS); //endPoint中已经声明，此为无效操作，对http也是如此
CfcClient cfcClient = new CfcClient(config);
```

配置CfcClient

如果用户需要配置CfcClient的一些细节参数，可以在构造CfcClient的时候传入BceClientConfiguration对象。

使用代理 下面代码可以让客户端使用代理访问CFC服务：

```
String accessKeyID = <your-access-key-id>; // 用户的Access Key ID
String secretAccessKey = <your-secret-access-key>; // 用户的Secret Access Key
String endPoint = <domain-name>; // 用户自己指定的域名

//创建BceClientConfiguration实例
BceClientConfiguration config = new BceClientConfiguration();

//配置代理为本地8080端口
config.setProxyHost("127.0.0.1");
config.setProxyPort(8080);

//创建CFC客户端
config.setCredentials(new DefaultBceCredentials(accessKeyID, secretAccessKey));
config.setEndpoint(endPoint);
CfcClient cfcClient = new CfcClient(config);
```

使用上面的代码段，客户端的所有操作都会通过127.0.0.1地址的8080端口做代理执行。对于有用户验证的代理，可以通过下面的代码段配置用户名和密码：

```
//创建BceClientConfiguration实例
BceClientConfiguration config = new BceClientConfiguration();

//配置代理为本地8080端口
config.setProxyHost("127.0.0.1");
config.setProxyPort(8080);

//设置用户名和密码
config.setProxyUsername(<username>); //用户名
config.setProxyPassword(<password>); //密码
```

设置网络参数

用户可以用BceClientConfiguration对基本网络参数进行设置：

```
BceClientConfiguration config = new BceClientConfiguration();

//设置HTTP最大连接数为10
config.setMaxConnections(10);

//设置TCP连接超时为5000毫秒
config.setConnectionTimeout(5000);

//设置Socket传输数据超时时间为2000毫秒
config.setSocketTimeout(2000);
```

参数说明

通过BceClientConfiguration能指定所有参数如下表所示：

参数	说明
UserAgent	用户代理，指HTTP的User-Agent头
Protocol	连接协议类型
ProxyDomain	访问NTLM验证的代理服务器的Windows域名
ProxyHost	代理服务器主机地址
ProxyPort	代理服务器端口
ProxyUsername	代理服务器验证的用户名
ProxyPassword	代理服务器验证的密码
ProxyPreemptiveAuthenticationEnabled	是否设置用户代理认证
ProxyWorkstation	NTLM代理服务器的Windows工作站名称
LocalAddress	本地地址
ConnectionTimeoutInMillis	建立连接的超时时间（单位：毫秒）
SocketTimeoutInMillis	通过打开的连接传输数据的超时时间（单位：毫秒）
MaxConnections	允许打开的最大HTTP连接数
RetryPolicy	连接重试策略
SocketBufferSizeInBytes	Socket缓冲区大小
StreamBufferSize	流文件缓冲区大小

函数操作

各接口的请求参数和响应参数说明请参考[函数操作](#)。

创建函数

如下代码可以创建一个CFC函数：

```

public void testCreateFunction(CfcClient cfcClient) {
    String zipFile =
"UEsDBBQACAgIAgumvE4AAAAAAAAAAAAAAAAIAAAAw5kZXgucHkVykEKgCAQQNG94B2m3FQk0i66TeWUgYwxjVG3z/7y8U3l8
;
    Code code = new Code();    //CFC函数代码
    code.setZipFile(zipFile);
    code.setPublish(false);
    code.setDryRun(true);
    String description = "test api";
    HashMap<String, String> variables = new HashMap<String, String>();
    variables.put("additional", "aaa");
    Environment environment = new Environment();
    environment.setVariables(variables);
    long time = System.currentTimeMillis();
    String functionName = "test-" + String.valueOf(time); //函数名称
    String handler = "index.handler";
    Integer memorySize = 256;
    String runtime = "python2";
    Integer timeOut = 5;

    CreateFunctionRequest request = new CreateFunctionRequest(functionName, null)
        .withCode(code)
        .withEnvironment(environment)
        .withHandler(handler)
        .withMemorySize(memorySize)
        .withRuntime(runtime)
        .withTimeout(timeOut);
    CreateFunctionResponse response = cfcClient.createFunction(request);
    System.out.println(response); //创建成功，获取新建函数信息
}

```

函数列表

如下代码可列出用户所在区域的所有函数：

```

public void testListFunctions(CfcClient cfcClient) {
    ListFunctionsResponse response = cfcClient.listFunctions("ALL", null, null);
    System.out.println(response);
}

```

获取函数信息

如下代码可以查询用户单个函数：

```

public void testGetFunction(CfcClient cfcClient) {
    String functionName = "test-1559096559957"; //已经存在的函数名称
    GetFunctionResponse response = cfcClient.getFunction(functionName, "$LATEST");
    System.out.println(response);
}

```

删除函数 如下代码用于删除用户函数：

```

public void testDeleteFunction(CfcClient cfcClient) {
    String functionName = "test-1557893401910"; //已经存在的函数名称
    cfcClient.deleteFunction(functionName, null);
}

```

更新函数代码

如下代码用于更新指定函数代码：

```
public void testUpdateFunctionCode(CfcClient cfcClient) {
    String zipFile =
"UEsDBBQACAgIAIumvE4AAAAAAAAAAAAAAAAIAAAaW5kZXgucHkVjMsOgjAQA09N+g8rXNTYVEHk8TMG6EJJmi2pW5C/t85xM
";
    Code code = new Code();
    code.setZipFile(zipFile);
    Boolean publish = true;
    Boolean dryRun = false;
    String functionName = "test-1559048290101"; //已经存在的函数名称

    UpdateFunctionCodeRequest request = new UpdateFunctionCodeRequest()
        .withFunctionName(functionName)
        .withZipFile(zipFile)
        .withPublish(publish)
        .withDryRun(dryRun);
    UpdateFunctionCodeResponse response = cfcClient.updateFunctionCode(request);
    System.out.println(response);
}
```

获取函数配置

如下代码用于获取指定函数的配置信息：

```
public void testGetFunctionConfiguration(CfcClient cfcClient) {
    String functionName = "test-1557387367523"; //已经存在的函数名称
    GetFunctionConfigurationResponse response = cfcClient.getFunctionConfiguration(functionName, null);
    System.out.println(response);
}
```

更新函数配置

如下代码用于修改函数配置：

```

public void testUpdateFunctionConfiguration(CfcClient cfcClient) {
    HashMap<String, String> variables = new HashMap<String, String>();
    variables.put("additional1", "bbbb");
    Environment environment = new Environment();
    environment.setVariables(variables);
    String functionName = "test-1559096559957"; //已经存在的函数名
    String desc = "123";
    Integer timeout = 3;
    String zipFile =
"UEsDBBQACAgIAIumvE4AAAAAAAAAAAAAAAAAIAAAAw5kZXgucHkVjMsOgjAQAO9N+g8rXNTYVEHk8TMG6EJjmi2pW5C/t85xM
";
    Code code = new Code();
    code.setZipFile(zipFile);
    code.setPublish(false);
    code.setDryRun(true);

    UpdateFunctionConfigurationRequest request = new UpdateFunctionConfigurationRequest(functionName, null)
        .withEnvironment(environment)
        .withTimeout(timeout)
        .withCode(code)
        .withFunctionName(functionName);
    UpdateFunctionConfigurationResponse response = cfcClient.updateFunctionConfiguration(request);
    System.out.println(response);
}

```

函数调用

请求参数和响应参数请参考[函数调用接口](#)。

调用函数

如下代码用于调用用户函数：

```

public void testInvoke() {
    HashMap<String, Object> payload = new HashMap<String, Object>();
    HashMap<String, Object> payload1 = new HashMap<String, Object>();
    payload1.put("key1", "value1");
    payload.put("key2", payload1);
    payload.put("key3", "value3");
    byte[] payloadBytes = JsonUtils.toJsonString(payload).getBytes();
    String functionName = "test-1603961009154"; // 已经存在的函数名称
    InvokeResponse response = cfcClient.invoke(functionName, "RequestResponse", "Tail", null, payloadBytes);
    String res = new String(response.getPayload());
    System.out.println(res);
}

```

注意：CFC调用执行用户函数返回结果在Payload字段中。

版本操作

各接口的请求参数和响应参数说明请参考[版本操作](#)。

获取函数版本列表

如下代码用于查询用户函数所有版本：


```
public void testListVersionsByFunction(CfcClient cfcClient) {  
    ListVersionsByFunctionResponse response = cfcClient.listVersionsByFunction("test-1557387367523", null, null);  
    System.out.println(response);  
}
```

🔗 发布版本

如下代码用于发布函数版本：

```
public void testPublishVersion(CfcClient cfcClient) {  
    PublishVersionResponse response = cfcClient.publishVersion("test-1557387367523", "1123", null);  
    System.out.println(response);  
}
```

别名操作

各接口的请求参数和响应参数说明请参考[别名操作](#)。

🔗 获取别名列表

如下代码用于查询特定函数的所有别名：

```
public void testListAliases(CfcClient cfcClient) {  
    String functionName = "test-1557387367523"; //已经存在的函数名称  
    ListAliasesResponse response = cfcClient.listAliases(functionName, null, null, null);  
    System.out.println(response);  
}
```

🔗 创建别名

如下代码用于创建函数别名：

```
public void testCreateAlias(CfcClient cfcClient) {  
    String functionName = "test-1557894532251"; //已经存在的函数名称  
    CreateAliasResponse response = cfcClient.createAlias(functionName, "$LATEST", "test2", null);  
    System.out.println(response);  
}
```

🔗 获取别名信息

如下代码用于查询特定别名详情：

```
public void testGetAlias(CfcClient cfcClient) {  
    String functionName = "test-1557387367523"; //已经存在的函数名称  
    String aliasName = "test1";  
    GetAliasResponse response = cfcClient.getAlias(aliasName, functionName);  
    System.out.println(response);  
}
```

🔗 更新别名信息

如下代码用于修改函数别名：

```
public void testUpdateAlias(CfcClient cfcClient) {  
    String functionName = "test-1559096559957"; //已经存在的函数名称  
    UpdateAliasResponse response = cfcClient.updateAlias(functionName, "$LATEST", "test1", "更新别名1");  
    System.out.println(response);  
}
```

删除别名

如下代码用于删除别名：

```
public void testDeleteAlias(CfcClient cfcClient) {  
    cfcClient.deleteAlias("test-1557387367523", "test1"); //参数分别为functionName, aliasName  
}
```

触发器操作

各接口的请求参数和响应参数说明请参考[触发器操作](#)。

获取触发器列表

如下代码用于获取触发器列表：

```
public void testListTrigger(CfcClient cfcClient) {  
    String functionBrn = "brn:bce:cfc:bj:7c83a9530352900ef3e38db05f1c10e9:function:test-1557900680300:$LATEST";  
    ListTriggersResponse response = cfcClient.listTriggers(functionBrn);  
    System.out.println(response);  
}
```

创建触发器

如下代码用于创建触发器：

```
public void testCreateTrigger(CfcClient cfcClient) {  
    String target = "brn:bce:cfc:bj:7c83a9530352900ef3e38db05f1c10e9:function:test-1557900680300:$LATEST";  
    String source = "cfc-http-trigger/v1/CFCAPI";  
    HashMap<String, String> data = new HashMap<String, String>();  
    data.put("AuthType", "anonymous");  
    data.put("Method", "GET");  
    data.put("ResourcePath", "hello002");  
  
    CreateTriggerResponse response = cfcClient.createTrigger(target, source, data);  
    System.out.println(response);  
}
```

更新触发器

如下代码用于更新触发器：

```
public void testUpdateTrigger(CfcClient cfcClient) {
    String relationId = "brn:bce:cfc-http-
trigger:bj:7c83a9530352900ef3e38db05f1c10e9:ece4c25ee836060f84d60fcc4a5477fd/cfc/GET/hello002";
    String source = "cfc-http-trigger/v1/CFCAPI";
    String target = "brn:bce:cfc:bj:7c83a9530352900ef3e38db05f1c10e9:function:test-1557900680300:$LATEST";
    HashMap<String, String> data = new HashMap<String, String>();
    data.put("AuthType", "anonymous");
    data.put("Method", "GET");
    data.put("ResourcePath", "hello1");

    UpdateTriggerResponse response = cfcClient.updateTrigger(relationId, target, source, data);
    System.out.println(response);
}
```

🔗 删除触发器

如下代码用于删除触发器：

```
public void testDeleteTrigger(CfcClient cfcClient) {
    String relationId = "brn:bce:cfc-http-
trigger:bj:7c83a9530352900ef3e38db05f1c10e9:ece4c25ee836060f84d60fcc4a5477fd/cfc/GET/hello1";
    String source = "cfc-http-trigger/v1/CFCAPI";
    String target = "brn:bce:cfc:bj:7c83a9530352900ef3e38db05f1c10e9:function:test-1557387367523:$LATEST";
    cfcClient.deleteTrigger(target, source, relationId);
}
```

异常处理

CFC异常提示有如下两种方式：

异常方法	说明
BceClientException	客户端异常
BceServerException	服务端异常

用户可以使用try获取某个事件所产生的异常，例如：

```
String functionName = "test";
String qualifier = "$LATEST";
try {
    cfcClient.getFunction(functionName, qualifier);
} catch (BceServiceException bce) {
    System.out.println(bce.getMessage());
} catch ( BceClientException bce){
    System.out.println( bce.getMessage());
}
```

🔗 客户端异常

客户端异常表示客户端尝试向CFC发送请求以及数据传输时遇到的异常。
例如，当发送请求时网络连接不可用时，则会抛出ClientException。

🔗 服务端异常

当CFC服务端出现异常时，CFC服务端会返回给用户响应的错误信息，以便定位问题。常见的服务端异常可参考[CFC错误返回](#)。

SDK日志

Java SDK发布版本中增加了logback作为slf4j的实现，如果用户没有自己的实现可以直接使用，如果工程中有其他的如log4j则可

以替代。

默认日志

如果用户使用默认的logback，则需要配置logback.xml到classpath中。如果没有这个配置文件，日志级别默认为DEBUG。

```
<configuration>
  <property name="LOG_HOME" value="./log/" />
  <appender name="STDOUT" class="ch.qos.logback.core.ConsoleAppender">
    <!-- encoders are assigned the type
         ch.qos.logback.classic.encoder.PatternLayoutEncoder by default -->
    <encoder>
      <pattern>%d{HH:mm:ss.SSS} [%thread] %-5level %logger{36} - %msg%n</pattern>
    </encoder>
  </appender>
  <appender name="FILE" class="ch.qos.logback.core.rolling.RollingFileAppender">
    <rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">
      <FileNamePattern>${LOG_HOME}/CdnUnitTest.%d{yyyy-MM-dd}.log</FileNamePattern>
      <MaxHistory>30</MaxHistory>
    </rollingPolicy>
    <encoder class="ch.qos.logback.classic.encoder.PatternLayoutEncoder">
      <pattern>%d{yyyy-MM-dd HH:mm:ss.SSS} [%thread] %-5level %logger{50} - %msg%n</pattern>
    </encoder>
    <triggeringPolicy class="ch.qos.logback.core.rolling.SizeBasedTriggeringPolicy">
      <MaxFileSize>10MB</MaxFileSize>
    </triggeringPolicy>
  </appender>
  <root level="info">
    <appender-ref ref="STDOUT" />
    <appender-ref ref="FILE" />
  </root>
</configuration>
```

自有日志模块

若用户使用自己的日志实现模块，例如项目依赖于Maven，则可以类似下面的配置到pom.xml中来去除logback。

```
<?xml version="1.0" encoding="utf-8"?>
<dependency>
  <groupId>com.baidubce</groupId>
  <artifactId>bce-java-sdk</artifactId>
  <version>${bce.sdk.version}</version>
  <exclusions>
    <exclusion>
      <groupId>ch.qos.logback</groupId>
      <artifactId>logback-classic</artifactId>
    </exclusion>
    <exclusion>
      <groupId>ch.qos.logback</groupId>
      <artifactId>logback-core</artifactId>
    </exclusion>
    <exclusion>
      <groupId>org.slf4j</groupId>
      <artifactId>jcl-over-slf4j</artifactId>
    </exclusion>
  </exclusions>
</dependency>
```

版本变更记录

- Java SDK开发包[2020-11-02]版本号 0.10.137 CFC 修改invoke函数输入参数payload格式，改为byte[]。

- Java SDK开发包[2019-07-05]版本号 0.10.64 CFC相关功能本次无更新
- Java SDK开发包[2019-07-01]版本号 0.10.63 CFC相关功能本次无更新
- Java SDK开发包[2019-06-26]版本号 0.10.62 CFC相关功能本次无更新
- Java SDK开发包[2019-06-25] 版本号 0.10.61

首次发布：

- 支持创建函数、查看函数列表、查看函数详情、删除函数、更新函数代码、获取函数配置、更新函数配置、函数调用
- 获取函数版本列表、发布新版本
- 获取别名列表、创建别名、获取别名信息、删除别名
- 获取触发器列表、创建触发器、更新触发器、删除触发器

GO-SDK

概述

本文档主要介绍CFC GO SDK的安装和使用。在使用本文档前，您需要先了解CFC的一些基本知识，并且已经开通了CFC服务。若您还不了解CFC，可以参考[产品描述](#)和[快速入门](#)。

安装SDK工具包

运行环境

GO SDK可以在go1.3及以上环境下运行。

安装SDK

直接从github下载

使用go get工具从github进行下载：

```
go get github.com/baidubce/bce-sdk-go
```

SDK目录结构

```
bce-sdk-go
|--auth          //BCE签名和权限认证
|--bce           //BCE公用基础组件
|--http          //BCE的http通信模块
|--services      //BCE相关服务目录
| |--cfc         //CFC服务目录
| | |--client.go //CFC客户端入口
| | |--api       //CFC相关API目录
| | |--cfc.go    //CFC的相关API实现
| | |--errors.go //CFC的错误信息定义
| | |--model.go  //CFC的API的数据模型
| | |--request.go //CFC相请求模块定义
| | |--util.go   //CFC相关API实现使用的工具
| | |--validator.go //CFC相关API参数检验工具
|--util          //BCE公用的工具实现
```

卸载SDK

预期卸载SDK时，删除下载的源码即可。

初始化

确认Endpoint

目前支持“华北-北京”、“华南-广州”、“华东-苏州”三个区域。北京区域：<http://cfc.bj.baidubce.com>，广州区域：<http://cfc.gz.baidubce.com>，苏州区域：<http://cfc.su.baidubce.com> 对应信息为：

访问区域	对应Endpoint
BJ	cfc.bj.baidubce.com
GZ	cfc.gz.baidubce.com
SU	cfc.su.baidubce.com

🔗 获取密钥

要使用百度云CFC，您需要拥有一个有效的AK(Access Key ID)和SK(Secret Access Key)用来进行签名认证。AK/SK是由账户管理员创建并分配给用户的，均为字符串，用于标识用户，为访问CFC做签名验证。

可以通过如下步骤获得并了解您的AK/SK信息：

[注册百度云账号](#)

[创建AK/SK](#)

🔗 CFC Client

CFC Client是CFC服务的客户端，为开发者与CFC服务进行交互提供了一系列的方法。

使用AK/SK新建CFC Client

通过AK/SK方式访问CFC，用户可以参考如下代码新建一个CFC Client：

```
import (
    "github.com/baidubce/bce-sdk-go/services/cfc"
)

func main() {
    // 用户的Access Key ID和Secret Access Key
    ACCESS_KEY_ID, SECRET_ACCESS_KEY := <your-access-key-id>, <your-secret-access-key>

    // 用户指定的Endpoint
    ENDPOINT := <domain-name>

    // 初始化一个cfcClient
    cfcClient, err := cfc.NewClient(AK, SK, ENDPOINT)
}
```

注意：ENDPOINT参数需要用指定区域的域名来进行定义，如服务所在区域为北京，则为<http://cfc.bj.baidubce.com>。

使用STS创建CFC Client

申请STS token

CFC可以通过STS机制实现第三方的临时授权访问。STS (Security Token Service) 是百度云提供的临时授权服务。通过STS，您可以为第三方用户颁发一个自定义时效和权限的访问凭证。第三方用户可以使用该访问凭证直接调用百度云的API或SDK访问百度云资源。

通过STS方式访问CFC，用户需要先通过STS的client申请一个认证字符串，申请方式可参见[百度云STS使用介绍](#)。

用STS token新建CFC Client

申请好STS后，可将STS Token配置到CFC Client中，从而实现通过STS Token创建CFC Client。

代码示例

GO SDK实现了STS服务的接口，用户可以参考如下完整代码，实现申请STS Token和创建CFC Client对象：

```
import (
    "fmt"

    "github.com/baidubce/bce-sdk-go/auth"    //导入认证模块
    "github.com/baidubce/bce-sdk-go/services/cfc" //导入CFC服务模块
    "github.com/baidubce/bce-sdk-go/services/sts" //导入STS服务模块
)

func main() {
    // 创建STS服务的Client对象，Endpoint使用默认值
    AK, SK := <your-access-key-id>, <your-secret-access-key>
    stsClient, err := sts.NewClient(AK, SK)
    if err != nil {
        fmt.Println("create sts client object :", err)
        return
    }

    // 获取临时认证token，有效期为60秒，ACL为空
    stsObj, err := stsClient.GetSessionToken(60, "")
    if err != nil {
        fmt.Println("get session token failed:", err)
        return
    }

    fmt.Println("GetSessionToken result:")
    fmt.Println("  accessKeyId:", stsObj.AccessKeyId)
    fmt.Println("  secretAccessKey:", stsObj.SecretAccessKey)
    fmt.Println("  sessionToken:", stsObj.SessionToken)
    fmt.Println("  createTime:", stsObj.CreateTime)
    fmt.Println("  expiration:", stsObj.Expiration)
    fmt.Println("  userId:", stsObj.UserId)

    // 使用申请的临时STS创建CFC服务的Client对象，Endpoint使用默认值
    cfcClient, err := cfc.NewClient(stsObj.AccessKeyId, stsObj.SecretAccessKey, "")
    if err != nil {
        fmt.Println("create cfc client failed:", err)
        return
    }

    stsCredential, err := auth.NewSessionBceCredentials(
        stsObj.AccessKeyId,
        stsObj.SecretAccessKey,
        stsObj.SessionToken)
    if err != nil {
        fmt.Println("create sts credential object failed:", err)
        return
    }

    cfcClient.Config.Credentials = stsCredential
}
```

注意：目前使用STS配置CFC Client时，无论对应CFC服务的Endpoint在哪里，STS的Endpoint都需配置为<http://sts.bj.baidubce.com>。上述代码中创建STS对象时使用此默认值。

🔗 配置HTTPS协议访问CFC

CFC支持HTTPS传输协议，您可以通过在创建CFC Client对象时指定的Endpoint中指明HTTPS的方式，在CFC GO SDK中使用HTTPS访问CFC服务：

```
// import "github.com/baidubce/bce-sdk-go/services/cfc"

ENDPOINT := "https://cfc.bj.baidubce.com" //指明使用HTTPS协议
AK, SK := <your-access-key-id>, <your-secret-access-key>
cfcClient, _ := cfc.NewClient(AK, SK, ENDPOINT)
```

🔗 配置cfc Client

如果用户需要配置CFC Client的一些细节的参数，可以在创建CFC Client对象之后，使用该对象的导出字段Config进行自定义配置，可以为客户端配置代理，最大连接数等参数。

使用代理

下面一段代码可以让客户端使用代理访问CFC服务：

```
// import "github.com/baidubce/bce-sdk-go/services/cfc"

//创建CFC Client对象
AK, SK := <your-access-key-id>, <your-secret-access-key>
ENDPOINT := "cfc.bj.baidubce.com"
client, _ := cfc.NewClient(AK, SK, ENDPOINT)

//代理使用本地的8080端口
client.Config.ProxyUrl = "127.0.0.1:8080"
```

设置网络参数

用户可以通过如下的示例代码进行网络参数的设置：

```
// import "github.com/baidubce/bce-sdk-go/services/cfc"

AK, SK := <your-access-key-id>, <your-secret-access-key>
ENDPOINT := "cfc.bj.baidubce.com"
client, _ := cfc.NewClient(AK, SK, ENDPOINT)

// 配置不进行重试，默认为Back Off重试
client.Config.Retry = bce.NewNoRetryPolicy()

// 配置连接超时时间为30秒
client.Config.ConnectionTimeoutInMillis = 30 * 1000
```

配置生成签名字符串选项


```
// import "github.com/baidubce/bce-sdk-go/services/cfc"

AK, SK := <your-access-key-id>, <your-secret-access-key>
ENDPOINT := "cfc.bj.baidubce.com"
client, _ := cfc.NewClient(AK, SK, ENDPOINT)

// 配置签名使用的HTTP请求头为`Host`
headersToSign := map[string]struct{"Host": struct{}}
client.Config.SignOption.HeadersToSign = HeadersToSign

// 配置签名的有效期为30秒
client.Config.SignOption.ExpireSeconds = 30
```

参数说明

用户使用GO SDK访问CFC时，创建的CFC Client对象的Config字段支持的所有参数如下表所示：

配置项名称	类型	含义
Endpoint	string	请求服务的域名
ProxyUrl	string	客户端请求的代理地址
Region	string	请求资源的区域
UserAgent	string	用户名称，HTTP请求的User-Agent头
Credentials	*auth.BceCredentials	请求的鉴权对象，分为普通AK/SK与STS两种
SignOption	*auth.SignOptions	认证字符串签名选项
Retry	RetryPolicy	连接重试策略
ConnectionTimeoutInMillis	int	连接超时时间，单位毫秒，默认20分钟

说明：

- Credentials字段使用auth.NewBceCredentials与auth.NewSessionBceCredentials函数创建，默认使用前者，后者为使用STS鉴权时使用，详见“使用STS创建CFC Client”小节。
- SignOption字段为生成签名字符串时的选项，详见下表说明：

名称	类型	含义
HeadersToSign	map[string]struct{}	生成签名字符串时使用的HTTP头
Timestamp	int64	生成的签名字符串中使用的时间戳，默认使用请求发送时的值
ExpireSeconds	int	签名字符串的有效期

其中，HeadersToSign默认为`Host`，`Content-Type`，`Content-Length`，`Content-MD5`；TimeStamp一般为零值，表示使用调用生成认证字符串时的时间戳，用户一般不应该明确指定该字段的值；ExpireSeconds默认为1800秒即30分钟。

- Retry字段指定重试策略，目前支持两种：NoRetryPolicy和BackOffRetryPolicy。默认使用后者，该重试策略是指定最大重试次数、最长重试时间和重试基数，按照重试基数乘以2的指数级增长的方式进行重试，直到达到最大重试测试或者最长重试时间为止。

函数调用

使用以下代码可以调用执行一个指定的CFC函数

```
args := &api.InvocationsArgs{
    FunctionName: "sdk-create",
    InvocationType: api.InvocationTypeRequestResponse,
    Payload:      nil,
}

// 若想执行特定版本的函数，可以设置
args.Qualifier = "1"

result, err := client.Invocations(args)
if err != nil {
    fmt.Println("invocation function failed:", err)
} else {
    fmt.Println("invocation function success: ", result)
}
```

函数操作

创建函数

使用以下代码可以创建一个CFC函数

```

    arge := &api.CreateFunctionArgs{
        // 配置函数的代码，需要上传代码的zip压缩包
        Code:      &api.CodeFile{ZipFile: zipFile},
        // 函数名称，每个用户的函数名称不可重复，不可修改
        FunctionName: "sdk-create",
        // 函数调用的入口函数
        Handler:     "index.handler",
        // 函数的runtime
        Runtime:     "nodejs8.5",
        // 函数运行的内存大小，单位mb，必须是128的整数倍，最大可选1024
        MemorySize: 256,
        // 函数执行超时时间，可选1-300s
        Timeout:     3,
        // 函数描述信息
        Description: "sdk create",
        // 函数日志存放方式，可选bos，表示函数执行日志存放在bos中
        LogType:     "bos",
        // 若LogType配置为bos，此参数设置函数执行日志在bos中的存储地址
        LogBosDir:   "bos://ashjfdklsfhlk/",
    })

    // 若要配置从bos bucket中上传函数代码，可以如下设置
    // 这两个参数不能和args.Code.ZipFile同时设置
    args.Code.BosBucket = "bucketName"
    args.Code.BosObject = "objectKey"

    // 若要直接发布函数，可以设置
    args.Code.Publish = true

    // 若要配置函数访问VPC网络，可以如下设置
    args.VpcConfig = &api.VpcConfig{
        SubnetIds:      []string{"subnet_id1"},
        SecurityGroupIds: []string{"security_group_id1"},
    }

    // 若要配置环境变量，可以如下设置
    args.Environment = &api.Environment{
        Variables: map[string]string{
            "key": "value",
        },
    },

    result, err := client.CreateFunction(args)
    if err != nil {
        fmt.Println("create function failed:", err)
    } else {
        fmt.Println("create function success: ", result)
    }

```

函数列表

使用以下代码可以获取CFC函数的列表

```
args := &api.ListFunctionArgs{}

// 若想查询指定版本1的函数，可以如下设置
args.FunctionVersion = "1"

result, err := client.ListFunctions(args)
if err != nil {
    fmt.Println("list function failed:", err)
} else {
    fmt.Println("list function success: ", result)
}
```

🔗 函数信息

使用以下代码可以获取特定函数的信息

```
args := &api.GetFunctionArgs{
    FunctionName: "functionName"
}
result, err := client.GetFunction(args)
if err != nil {
    fmt.Println("get function failed:", err)
} else {
    fmt.Println("get function success: ", result)
}
```

🔗 删除函数

使用以下代码可以删除一个特定的CFC函数

```
args := &api.DeleteFunctionArgs{
    FunctionName: "sdk-create",
}

// 若想删除函数的某个版本，可以设置
args.Qualifier = "1"

err := client.DeleteFunction(args)
if err != nil {
    fmt.Println("delete function failed:", err)
}
```

🔗 更新函数代码

使用以下代码可以更新特定CFC函数的代码

```
args := &api.UpdateFunctionCodeArgs{
    FunctionName: "sdk-creat"
    ZipFile:      []byte(functionZipCode)
}

// 若要配置从bos bucket中上传函数代码，可以如下设置
// 这两个参数不能和args.ZipFile同时设置
args.BosBucket = "bucketName"
args.BosObject = "objectKey"

// 若要直接发布函数，可以设置
args.Publish = true

result, err := client.UpdateFunctionCode(args)
if err != nil {
    fmt.Println("update function code failed:", err)
} else {
    fmt.Println("update function code success: ", result)
}
```

🔗 获取函数配置

使用以下代码可以获取特定CFC函数的配置

```
args := &api.GetFunctionConfigurationArgs{
    FunctionName: "sdk-create",
}

// 若想查询特定版本的函数的配置，可以设置
args.Qualifier = functionBrn

if err != nil {
    fmt.Println("get function configure failed:", err)
} else {
    fmt.Println("get function configure success: ", result)
}
```

🔗 更新函数配置

使用以下代码可以更新特定CFC函数的配置

```

args := &api.UpdateFunctionConfigurationArgs{
    FunctionName: "sdk-create",
    Timeout:      20,
    Description:  "sdk update",
    Runtime:      "nodejs8.5",
    MemorySize:   &memorySize,
    Environment:  &api.Environment{
        Variables: map[string]string{
            "name": "Test",
        },
    },
}

result, err := client.UpdateFunctionConfiguration(args)
if err != nil {
    fmt.Println("update function configure failed:", err)
} else {
    fmt.Println("update function configure success: ", result)
}

```

🔗 设置函数预留并发度

使用以下代码可以设置和更新特定CFC函数的预留并发度

```

args := &api.ReservedConcurrentExecutionsArgs{
    FunctionName: "sdk-create",
    // 预留并发度会由本函数的所有版本共享，最高能设置90
    ReservedConcurrentExecutions: 10,
}

err := client.SetReservedConcurrentExecutions(args)
if err != nil {
    fmt.Println("set function reserved concurrent executions failed:", err)
}

```

🔗 删除函数预留并发度设置

使用以下代码可以删除特定CFC函数的预留并发度设置

```

args := &api.DeleteReservedConcurrentExecutionsArgs{
    FunctionName: "sdk-create",
}

err := client.DeleteReservedConcurrentExecutions(args)
if err != nil {
    fmt.Println("delete function reserved concurrent executions failed:", err)
}

```

版本操作

获取函数版本列表

使用以下代码可以获取函数版本列表

```
args := &api.ListVersionsByFunctionArgs{
    FunctionName: "sdk-create",
}

result, err := client.ListVersionsByFunction(args)
if err != nil {
    fmt.Println("get function version failed:", err)
} else {
    fmt.Println("get function version success: ", result)
}
```

发布版本

使用以下代码可以为函数发布一个版本

```
args := &api.PublishVersionArgs{
    FunctionName: "sdk-create",
}

// 若想添加版本描述，可以设置
args.Description = "publish description"

// 若想对版本的部署包进行sha256验证，可以设置
args.CodeSha256 = "codeSha256"

result, err := client.PublishVersion(args)
if err != nil {
    fmt.Println("publish function version failed:", err)
} else {
    fmt.Println("publish function version success: ", result)
}
```

别名操作

获取别名列表

使用以下代码可以获取函数的别名列表

```
args := &api.ListAliasesArgs{
    FunctionName: "sdk-create",
}

// 若想获取特定函数版本的别名，可以设置
args.FunctionVersion = "1"

result, err := client.ListAliases(args)
if err != nil {
    fmt.Println("list function alias failed:", err)
} else {
    fmt.Println("list function alias success: ", result)
}
```

创建别名

使用以下代码可以为特定函数版本创建一个别名

```
args := &api.CreateAliasArgs{
    FunctionName: "sdk-create",
    Name:         "alias-create",
}

// 若要将别名绑定到特定函数版本, 可以设置
args.FunctionVersion = "1"

// 若要设置别名标书, 可以设置
args.Description = "alias description"

result, err := client.CreateAlias(args)
if err != nil {
    fmt.Println("create function alias failed:", err)
} else {
    fmt.Println("create function alias success: ", result)
}
```

🔗 获取别名信息

使用以下代码可以获取一个特定函数的别名的信息

```
args := &api.GetAliasArgs{
    FunctionName: "sdk-create",
    AliasName:    "alias-create",
}

result, err := client.GetAlias(args)
if err != nil {
    fmt.Println("get function alias failed:", err)
} else {
    fmt.Println("get function alias success: ", result)
}
```

更新别名

使用以下代码可以更新一个函数的别名

```
args := &api.UpdateAliasArgs{
    FunctionName: "sdk-create",
    AliasName:    "alias-create",
    Description:  "test alias",
}

// 若要修改别名绑定的函数版本, 可以设置
args.FunctionVersion = "$LATEST"

result, err := client.UpdateAlias(args)
if err != nil {
    fmt.Println("update function alias failed:", err)
} else {
    fmt.Println("update function alias success: ", result)
}
```

🔗 删除别名

使用以下代码可以删除一个函数的别名


```
args := &api.DeleteAliasArgs{
    FunctionName: "sdk-create",
    AliasName:    "alias-create",
}

err := client.DeleteAlias(args)
if err != nil {
    fmt.Println("delete function alias failed:", err)
}
```

触发器操作

获取触发器列表

使用以下代码可以获取一个触发器的列表

```
args := &api.ListTriggersArgs{
    FunctionBrn: "functionBrn",
}

result, err := client.ListTriggers(args)
if err != nil {
    fmt.Println("get function trigger failed:", err)
} else {
    fmt.Println("get function trigger success: ", result)
}
```

🔗 创建触发器

使用以下代码可以创建一个特定的触发器并绑定

```
args := &api.CreateTriggerArgs{
    Target: "functionBrn",
    Source: api.SourceTypeCrontab,
    // 创建crontab触发器所需的数据
    Data: &api.CrontabTriggerData{
        Name:        "sdkName",
        Brn:         "functionBrn",
        ScheduleExpression: "cron(0 10 * * * ?)",
        Enabled:      "Disabled",
    },
}

result, err := client.CreateTrigger(args)
if err != nil {
    fmt.Println("create function trigger failed:", err)
} else {
    fmt.Println("create function trigger success: ", result)
}
```

提示：

1. 不同类型的触发器，其Data字段所需内容不同，具体可以参考文档[触发器配置](#)

🔗 更新触发器

使用以下代码可以更新一个函数的触发器

```
args := &api.UpdateTriggerArgs{
    RelationId: RelationId,
    Target:     functionBRN,
    Source:     api.SourceTypeHTTP,
    Data: &api.HttpTriggerData{
        ResourcePath: fmt.Sprintf("tr99-%s", time.Now().Format("2006-01-02T150405")),
        Method:       "GET",
        AuthType:     "anonymous",
    },
}

result, err := client.UpdateTrigger(args)
if err != nil {
    fmt.Println("update function trigger failed:", err)
} else {
    fmt.Println("update function trigger success: ", result)
}
```

删除触发器

使用以下代码可以删除一个触发器

```
args := &api.DeleteTriggerArgs{
    RelationId: RelationId,
    Target:     functionBRN,
    Source:     api.SourceTypeHTTP,
}

err := client.DeleteTrigger(args)
if err != nil {
    fmt.Println("delete function trigger failed:", err)
}
```

错误处理

GO语言以error类型标识错误，CFC支持两种错误见下表：

错误类型	说明
CFCClientError	用户操作产生的错误
BceServiceError	CFC服务返回的错误

用户使用SDK调用CFC相关接口，除了返回所需的结果之外还会返回错误，用户可以获取相关错误进行处理。实例如下：

```
// cfcClient 为已创建的CFC Client对象
args := &api.GetFunctionArgs{
    FunctionName: "functionName"
}
result, err := cfcClient.GetFunction(args)
if err != nil {
    switch realErr := err.(type) {
    case *bce.BceClientError:
        fmt.Println("client occurs error:", realErr.Error())
    case *bce.BceServiceError:
        fmt.Println("service occurs error:", realErr.Error())
    default:
        fmt.Println("unknown error:", err)
    }
} else {
    fmt.Println("get function detail success: ", result)
}
```

🔗 客户端异常

客户端异常表示客户端尝试向CFC发送请求以及数据传输时遇到的异常。例如，当发送请求时网络连接不可用时，则会返回BceClientError；当上传文件时发生IO异常时，也会抛出BceClientError。

🔗 服务端异常

当CFC服务端出现异常时，CFC服务端会返回给用户相应的错误信息，以便定位问题。常见服务端异常可参见[CFC错误返回](#)

版本变更记录

🔗 v0.9.1 [2019-09-26]

首次发布：

- 执行函数
- 创建、查看、列表、删除函数，更新函数代码，更新、获取函数配置
- 设置、删除函数预留并发度
- 列表、创建函数版本
- 列表、创建、获取、更新、删除别名
- 获取、创建、更新、删除触发器

Python SDK

概述

本文档主要介绍CFC Python SDK的安装和使用。在使用本文档前，您需要先了解CFC的一些基本知识，并且已经开通了CFC服务。若您还不了解CFC，可以参考[产品描述](#)和[快速入门](#)。

初始化

🔗 确认Endpoint

在确认您使用SDK时配置的Endpoint时，可先阅读API参考中关于[CFC域名服务](#)的部分，理解Endpoint相关概念。

目前CFC支持“华北-北京”：<http://cfc.bj.baidubce.com>、“华南-广州”：<http://cfc.gz.baidubce.com>、“华东-苏州”：<http://cfc.su.baidubce.com>三个区域。对应信息为：

访问区域	对应Endpoint
BJ	cfc.bj.baidubce.com
GZ	cfc.gz.baidubce.com
SU	cfc.su.baidubce.com

CFC访问域名支持HTTP和HTTPS两种调用方式，为了提升数据的安全性，建议通过HTTPS调用。更多关于HTTPS保障传输安全的使用请参考[HTTPS传输加密实践](#)。

🔗 获取密钥

要使用百度智能云CFC，您需要拥有一个有效的AK (Access Key ID)和SK (Secret Access Key)用来进行签名认证。AK/SK是由系统分配给用户的，均为字符串，用于标识用户，为访问CFC做签名验证。

可以通过如下步骤了解并获得您的AK/SK信息：

[注册百度智能云账号](#)

[创建AK/SK](#)

安装SDK工具包

🔗 环境准备

1. 运行环境

Python SDK工具包支持在Python 2.7 和Python 3 的环境运行。

2. 安装pycrypto依赖

安装SDK之前，需要先执行命令 `pip install pycrypto` 安装pycrypto依赖。

如果安装失败，请执行 `pip install pycryptodome`。

🔗 下载和安装

方式一：通过pip安装

您可以通过pip安装的方式将百度智能云Python SDK安装到您的环境中。联网状态下，在命令行中执行如下命令：

```
pip install bce-python-sdk
```

即可将Python SDK安装到本地。

方式二：将源码包下载到本地后进行安装

1. 在[开发者资源中心](#)下载Python SDK压缩工具包。

2. 命令行移动到压缩包所在路径，执行如下命令（version替换为包名称中的版本号）：

```
pip install bce-python-sdk-version.zip
```

即可将Python SDK安装到本地。

方式三：使用setup.py install文件进行安装

您也可以解压压缩包后执行如下命令（version替换为包名称中的版本号）

```
cd bce-python-sdk-version
```

```
python setup.py install
```

卸载SDK

卸载 SDK 时，执行命令 `pip uninstall bce-python-sdk` 即可完成卸载。

SDK目录结构

```
baidubce
├── auth                //公共权限目录
├── http                //Http请求模块
├── services            //服务公共目录
│   └── cfc            //CFC目录
│       ├── cfc_client.py //CFC客户端入口类
│       └── cfc_handle.py //HTTP reponse处理函数
├── bce_base_client.py //BCE客户端入口类的基类
├── bce_client_configuration.py //针对CFC特有的HttpClient的配置类
├── bce_response.py     //BCE客户端的请求类
├── exception.py        //BCE客户端的异常类
├── protocol.py         //网络协议定义
├── region.py           //区域处理模块
├── retry_policy.py     //BCE服务公共配置类
└── utils.py            //BCE公用工具类
```

CfcClient

CfcClient是CFC服务的客户端，为开发者与CFC服务进行交互提供了一系列的方法，包括函数调用、函数创建、获取函数列表等。

新建CfcClient

通过AK/SK方式访问CFC

1、 在新建CfcClient之前，需要先创建配置文件对CfcClient进行配置，以下将此配置文件命名为 `cfc_sample_conf.py`，具体配置信息如下：

```

"""
Configuration for cfc samples.
"""

##### 导入Python标准日志模块
import logging

##### 从Python SDK导入CFC配置管理模块以及安全认证模块
from baidubce.bce_client_configuration import BceClientConfiguration
from baidubce.auth.bce_credentials import BceCredentials

##### 设置CfcClient的Host，Access Key ID和Secret Access Key
HOST = b'http://cfc.bj.baidubce.com'
AK = b'ak'
SK = b'sk'

##### 设置日志文件的句柄和日志级别
logger = logging.getLogger('baidubce.services.cfc.client')
fh = logging.FileHandler('sample.log')
fh.setLevel(logging.DEBUG)

##### 设置日志文件输出的顺序、结构和内容
formatter = logging.Formatter('%(asctime)s - %(name)s - %(levelname)s - %(message)s')
fh.setFormatter(formatter)
logger.setLevel(logging.DEBUG)
logger.addHandler(fh)

##### 创建BceClientConfiguration
config = BceClientConfiguration(credentials=BceCredentials(AK, SK), endpoint=HOST)

```

注意：针对日志文件，Logging有如下级别：DEBUG，INFO，WARNING，ERROR，CRITICAL。

在上面代码中，AK对应控制台中的“Access Key ID”，SK对应控制台中的“Access Key Secret”，获取方式请参考《操作指南[管理ACCESSKEY](#)》。

使用AK/SK方式创建CfcClient，用户需要自己指定CFC的服务域名，可以通过赋值给HOST变量来指定。不指定时无需传入endpoint参数，默认为北京区域https://cfc.bj.baidubce.com。

2、在配置完上述配置之后，参考如下代码新建一个CfcClient。

```

"""
Samples for cfc client.
"""

##### 导入CfcClient配置文件
import cfc_sample_conf

##### 导入CFC相关模块
from baidubce.services.cfc.cfc_client import CfcClient

##### 新建CfcClient
cfc_client = CfcClient(cfc_sample_conf.config)

```

配置HTTPS协议访问CFC

您可以通过如下两种方式在CFC Python SDK中使用HTTPS访问CFC服务：

- 在endpoint中指定HTTPS：

```

config = bce_client_configuration.BceClientConfiguration(
    credentials = bce_credentials.BceCredentials(
        access_key_id = 'your-ak',
        secret_access_key = 'your-sk'
    ),
    endpoint = 'https://cfc.bj.baidubce.com'
)
client = cfc_client.CfcClient(config)

```

- 通过在protocol中指定https来设置HTTPS协议：

```

config = bce_client_configuration.BceClientConfiguration(
    credentials = bce_credentials.BceCredentials(
        access_key_id = 'your-ak',
        secret_access_key = 'your-sk'
    ),
    endpoint = 'cfc.bj.baidubce.com',
    protocol = baidubce.protocol.HTTPS
)
client = cfc_client.CfcClient(config)

```

注意：如果您在指定了endpoint的scheme的同时指定了protocol参数，则以endpoint为准。

函数操作

各接口的请求参数和响应参数说明请参考[函数操作](#)。

🔗 创建函数

如下代码可以创建一个CFC函数：

```

function_name = "testHelloWorld" #您可以自定义您的函数名
# 创建一个cfc客户端
#有三个endpoint可以选择:cfc.bj.baidubce.com,cfc.gz.baidubce.com,cfc.su.baidubce.com
host = '<Your Endpoint>'
AK = '<Your AccessKeyID>'
SK = '<Your AccessKeySecret>'

# 也可以通过环境变量的形式获取ak,sk
# AK = os.environ["BCE_ACCESS_KEY_ID"]
# SK = os.environ["BCE_ACCESS_KEY_SECRET"]

config = BceClientConfiguration(credentials=BceCredentials(AK, SK), endpoint=host)
cfc_client = CfcClient(config)

##### 下面是您要发布的zip包的 base64-encoded， 注意zip包压缩时不能包含顶层文件夹目录
base64_file = '<Your base64-encoded Code>'

#创建函数
create_response = cfc_client.create_function(function_name,
                                             description="CFC SDK Demo",
                                             handler="<Your index>.handler",
                                             memory_size=128,
                                             region='bj',
                                             zip_file=base64_file,
                                             publish=False,
                                             run_time='python2',
                                             timeout=3,
                                             dry_run=False)

```

🔗 函数列表

如下代码可列出用户所在区域的所有函数：

```
##### list functions
list_functions_response = cfc_client.list_functions()
```

🔗 获取函数信息

如下代码可以查询用户单个函数：

```
get_function_response = cfc_client.get_function(FunctionName)
```

🔗 更新函数代码

如下代码用于更新指定函数代码：

```
brn = create_response.FunctionBrn
update_function_code_response = cfc_client.update_function_code(brn,
                                                                zip_file=base64_file_with_args,
                                                                publish=True)
```

🔗 获取函数配置

如下代码用于获取指定函数的配置信息：

```
// 根据函数BRN获取配置信息，也可以根据函数名称获取配置信息
brn = create_response.FunctionBrn
get_function_configuration_response = cfc_client.get_function_configuration(brn)
```

🔗 更新函数配置

如下代码用于修改函数配置：

```
brn = create_response.FunctionBrn
// 更新函数描述信息和环境变量
response = cfc_client.update_function_configuration(brn, description="update config",
                                                    environment={"e1": "1"})
```

🔗 删除函数

如下代码用于删除用户函数：

```
delete_function_response = cfc_client.delete_function(function_name)
```

函数调用

请求参数和响应参数请参考[函数调用接口](#)。

🔗 调用函数

如下代码用于调用用户函数：

```
invocations_response = cfc_client.invocations(functionName, log_type="Tail")
```


别名操作

各接口的请求参数和响应参数说明请参考[别名操作](#)。

🔗 获取别名列表

如下代码用于查询特定函数的所有别名：

```
// 获取别名列表，参数为函数名称
response = cfc_client.list_aliases(function_name)
```

🔗 创建别名

如下代码用于创建函数别名：

```
// 创建别名
response = cfc_client.create_alias(function_name, function_version="1", name="my_alias",
                                   description="this is first alias")
```

🔗 获取别名信息

如下代码用于查询特定别名详情：

```
// 获取函数别名信息
response = cfc_client.get_alias(function_name, alias_name="my_alias")
```

🔗 更新别名信息

如下代码用于修改函数别名：

```
// 更新别名信息，其中参数brn可以换成function_name
response = cfc_client.update_alias(brn, function_version="$LATEST",
                                   alias_name="my_alias", description="change")
```

🔗 删除别名

如下代码用于删除指定别名：

```
// 删除别名， 其中参数brn可以换成function_name
response = cfc_client.delete_alias(brn, alias_name="my_alias")
```

版本操作

各接口的请求参数和响应参数说明请参考[版本操作](#)。

🔗 获取函数版本列表

如下代码用于查询用户函数所有版本：

```
##### 获取函数所有版本
response = cfc_client.list_versions_by_function(functionName)
```

🔗 发布新版本

如下代码用于发布新版本：

```
##### 发布新版本
response = cfc_client.publish_version(functionName, description="this is test publish")
```

异常处理

🔗 系统异常

CFC系统异常提示有如下三种方式：

异常方法	说明
BceHttpClientError	重试时抛出的异常
last_error	最后一次重试时抛出的异常
BceClientError	CFC客户端产生的异常
BceInvalidArgumentError	传递参数产生的异常
BceServerError	CFC服务器产生的异常

用户可以使用try获取某个事件所产生的异常：

```
from baidubce import exception
from baidubce.services.cfc.cfc_client import CfcClient

try:
    cfc_client.delete_invocations(brn, log_type="Tail")
except exception.BceHttpError as e:
    print e.message
```

🔗 客户端异常

客户端异常表示客户端尝试向CFC发送请求以及数据传输时遇到的异常。例如，当发送请求时网络连接不可用时，则会抛出BceHttpClientError。

🔗 服务端异常

当CFC服务端出现异常时，CFC服务端会返回给用户相应的错误信息，以便定位问题，常见服务端异常可参考[CFC错误返回](#)。

触发器操作

各接口的请求参数和响应参数说明请参考[触发器操作](#)。

🔗 获取触发器列表

如下代码用于获取触发器列表：

```
##### 获取触发器列表，brn也可以换成functionName
list_triggers_response = cfc_client.list_triggers(brn)
__logger.debug("[Sample CFC] list_triggers response:%s", list_triggers_response)
```

🔗 创建触发器

如下代码用于创建触发器：

```
##### 创建触发器
##### 下面代码用于创建http触发器
data = {}
data["AuthType"] = 'anonymous'
data["Method"] = 'GET'
data["ResourcePath"] = 'hello'
source = 'cfc-http-trigger/v1/CFCAPI' target = create_response.FunctionBmcreate_trigger_response =
cfc_client.create_trigger(target, source, data)
__logger.debug("[Sample CFC] create_trigger response:%s", create_trigger_response)
```

更新触发器

如下代码用于更新触发器：

```
##### 更新触发器
data["ResourcePath"] = 'hello11'
update_trigger_response = cfc_client.update_trigger(relationId, target, source,data)
__logger.debug("[Sample CFC] update_trigger response:%s", update_trigger_response)
```

删除触发器

如下代码用于删除触发器：

```
##### 删除触发器
delete_trigger_response = cfc_client.delete_trigger(target, source, relationId)
__logger.debug("[Sample CFC] delete_trigger response:%s", delete_trigger_response)
```

JavaScript-SDK

概述

本文档主要介绍CFC Javascript SDK的安装和使用。在使用本文档前，您需要先了解CFC的一些基本知识，并且已经开通了CFC服务。若您还不了解CFC，可以参考[产品描述](#)和[快速入门](#)。

CFC JavaScript SDK可以运行在浏览器或者Node.js环境中，开发者通过调用API简单便捷地使用CFC服务。

安装SDK工具包

安装

SDK 的安装较为简单，推荐使用 npm 来安装。在终端输入如下命令：

```
npm install @baiducloud/sdk --save
```

更新

```
npm update @baiducloud/sdk
```

初始化

确认Endpoint

在确认您使用SDK时配置的Endpoint时，可先阅读API参考中关于[CFC域名服务](#)的部分，理解Endpoint相关概念。

目前CFC支持“华北-北京”：<http://cfc.bj.baidubce.com>、“华南-广州”：<http://cfc.gz.baidubce.com>、“华东-苏州”：<http://cfc.su.baidubce.com>三个区域。对应信息为：

访问区域	对应Endpoint
BJ	cfc.bj.baidubce.com
GZ	cfc.gz.baidubce.com
SU	cfc.su.baidubce.com

CFC访问域名支持HTTP和HTTPS两种调用方式，为了提升数据的安全性，建议通过HTTPS调用。更多关于HTTPS保障传输安全的使用请参考[HTTPS传输加密实践](#)。

🔗 获取密钥

要使用百度智能云CFC，您需要拥有一个有效的AK (Access Key ID)和SK (Secret Access Key)用来进行签名认证。AK/SK是由系统分配给用户的，均为字符串，用于标识用户，为访问CFC做签名验证。

可以通过如下步骤了解并获得您的AK/SK信息：

[注册百度智能云账号](#)

[创建AK/SK](#)

🔗 CFC Client

CfcClient是CFC服务的客户端，为开发者与CFC服务进行交互提供了一系列的方法，包括函数调用、函数创建、获取函数列表等。

通过AK/SK方式访问CFC

通过AK/SK方式访问CFC，用户可以参考如下代码新建一个CFC Client：

```
var CfcClient = require('@baiducloud/sdk').CfcClient;

config = {
  endpoint: '<CFC Endpoint>',
  credentials: {
    ak: '<Your Access Key ID>',
    sk: '<Your Secret Access Key>'
  }
};

// 创建cfc客户端
var client = new CfcClient(config);
```

在上面代码中，Access Key ID和Secret Access Key的获取方式请参考《操作指南[管理ACCESSKEY](#)》。

Endpoint参数需要用指定区域的域名来进行定义，如服务所在区域为北京，则为http://cfc.bj.baidubce.com。

使用STS访问CFC

CFC可以通过STS机制实现第三方的临时授权访问。STS（Security Token Service）是百度智能云提供的临时授权服务，详情可参见[百度智能云STS使用介绍](#)。通过STS，您可以为第三方用户颁发一个自定义时效和权限的访问凭证。第三方用户可以使用该访问凭证直接调用百度智能云的API或SDK访问百度智能云资源。

通过STS方式访问CFC，用户先通过StsClient申请一套AK、SK和token，然后将该套参数配置到CfcClient中，用户可以参考如下代码新建一个CfcClient：

```
var CfcClient = require('@baiducloud/sdk').CfcClient;
var STS = require('@baiducloud/sdk').STS;

var stsConfig = {
  endpoint: 'http://sts.bj.baidubce.com',
  credentials: {
    ak: '<Your Access Key ID>',
    sk: '<Your Secret Access Key>'
  }
}
var stsClient = new STS(stsConfig);

stsClient.getSessionToken(60, '').then(function (response) {
  var stsObj = response.body

  cfcConfig = {
    endpoint: '<CFC Endpoint>',
    credentials: {
      ak: stsObj.accessKeyId,
      sk: stsObj.secretAccessKey
    },
    sessionToken: stsObj.sessionToken
  };

  var cfcClient = new CfcClient(cfcConfig);
  cfcClient.listFunctions().then(function (response) {
    console.log('list function:', response.body);
    return response
  }).catch(function (err) {
    console.error('error', err)
  })

}).catch(function (err) {
  console.error('error', err)
})
```

在上面代码中，Access Key ID和Secret Access Key的获取方式请参考《操作指南[管理ACCESSKEY](#)》。

Endpoint参数需要用指定区域的域名来进行定义，如服务所在区域为北京，则为<http://cfc.bj.baidubce.com>。

注意：目前使用STS配置CFC Client时，无论对应CFC服务的Endpoint在哪里，STS的Endpoint都需配置为<http://sts.bj.baidubce.com>。上述代码中创建STS对象时使用此默认值。

🔗 配置HTTPS协议访问CFC

CFC支持HTTPS传输协议，您可以通过在创建CFC Client对象时指定的Endpoint中指明HTTPS的方式，在CFC Javascript SDK中使用HTTPS访问CFC服务：

```
var CfcClient = require('@baiducloud/sdk').CfcClient;

var config = {
  endpoint: 'https://cfc.bj.baidubce.com',
  credentials: {
    ak: '<Your Access Key ID>',
    sk: '<Your Secret Access Key>'
  }
};

// 创建cfc客户端
var client = new CfcClient(config);
```

函数调用

请求参数和响应参数请参考[函数调用接口](#)。

使用以下代码可以调用执行一个指定的CFC函数，默认会调用 \$LATEST 版本的函数。

```
const payload = '{"foo": "bar"}'
const functionName = 'myfunc'

cfcClient.invocations(functionName, payload).then(response => {
  const funcResult = response.body.toString('Utf-8')
  console.log(funcResult)
}).catch(error => {
  console.error(error)
})
```

您可以在调用执行时设置参数，指定函数的别名或版本，或指定函数返回的信息和格式。

```
const payload = '{"foo": "bar"}'
const functionName = 'myfunc'
const options = {
  Qualifier: 'prod',           // 指定函数别名
  logToBody: true,             // 函数执行结果放在http body的Payload字段中
  logType: 'Tail',             // 返回函数日志
  invocationType: 'RequestResponse' // 同步调用
}

cfcClient.invocations(functionName, payload, options).then(response => {
  console.log(response.body.LogResult)
  console.log(response.body.Payload)
}).catch(error => {
  console.error(error)
})
```

函数操作

各接口的请求参数和响应参数说明请参考[函数操作](#)。

🔗 创建函数

如下代码可以创建一个CFC函数：

```
// 下面是您要发布的zip包的 base64-encoded，注意zip包压缩时不能包含顶层文件夹目录
var base64_file = '<Your base64-encoded Code>'

var body = {
  'Code': {
    'ZipFile': base64_file,
    'Publish': false,
  },
  'Description': 'CFC SDK Demo',
  'Region': 'bj',
  'Timeout': 3,
  'FunctionName': 'testHelloWorld',
  'Handler': '<your index>.handler',
  'Runtime': 'nodejs12',
  'MemorySize': 128,
  'Environment': {
    'Variables': {
      'a': 'b',
    }
  },
},
'LogType': 'bls'
};

client.createFunction(body).then(response => {
  console.log('create function:', response.body);
  return response;
}).catch(err => {
  console.error('error:', err);
});
```

函数列表

如下代码可列出用户在某个地域的所有函数列表：

```
client.listFunctions().then(response => {
  console.log(response.body)
})
```

获取函数信息

如下代码可以查询用户单个函数，指定函数名称或函数BRN均可：

```
cfcClient.getFunction(functionName).then(response => {
  console.log(response.body)
  console.log(response.body.Configuration.FunctionBmn) // 输出函数 brn
})
```

更新函数代码

如下代码用于更新指定函数代码：

```
var updateOptions = {
  'ZipFile': '<Your base64-encoded Code>',
  'Publish': false
}

cfcClient.updateFunctionCode(functionName, updateOptions).then(response => {
  console.log(response.body.FunctionBn)
}).catch(err => {
  console.error(err)
})
```

🔗 获取函数配置

如下代码用于获取指定函数的配置信息：

```
var options = {
  'Qualifier': '$LATEST'
}

client.getFunctionConfiguration(functionName, options).then(response => {
  console.log(response.body.FunctionBn)
})
```

🔗 更新函数配置

如下代码用于修改函数配置：

```
// 更新函数环境变量、超时时间和描述信息
var updateOptions = {
  'Environment': {
    'Variables': {
      'additionalProp1': 'string',
      'additionalProp2': 'string',
      'additionalProp3': 'string'
    }
  },
  'Timeout': 10,
  'Description': 'update config desc'
}

client.updateFunctionConfiguration(functionName, updateOptions).then(response => {
  console.log(response.body.FunctionBn)
}).catch(err => {
  console.error(err)
})
```

🔗 删除函数

如下代码用于删除用户函数的某个版本，不传deleteOptions参数则删除整个函数：

```
// 删除版本2
var deleteOptions = {
  'Qualifier': '2'
}

cfcClient.deleteFunction(functionName, deleteOptions).catch(err => {
  console.error(err)
})
```


版本操作

各接口的请求参数和响应参数说明请参考[版本操作](#)。

🔗 获取函数版本列表

如下代码用于获取某个函数所有的版本：

```
cfcClient.listVersionsByFunction(functionName).then(response => {  
  console.log(response.body)  
})
```

🔗 发布新版本

如下代码用于发布新版本，版本号会自动生成：

```
cfcClient.publishVersion(functionName, "new version desc").then(response => {  
  console.log(response.body)  
})
```

别名操作

各接口的请求参数和响应参数说明请参考[别名操作](#)。

🔗 获取别名列表

如下代码用于查询函数的别名列表，可查看某个版本的别名列表，若不指定版本则列出该函数所有别名：

```
var options = {  
  'FunctionVersion': '1' // 查看版本1关联的别名列表  
}  
  
cfcClient.listAliases(functionName, options).then(response => {  
  console.log(response.body)  
})
```

🔗 创建别名

如下代码用于创建函数别名：

```
var body = {  
  'FunctionVersion': '1', // 别名指向的版本  
  'Name': 'prod',  
  'Description': 'point to the prod version'  
}  
  
client.createAlias(functionName, body).then(response => {  
  console.log(response.body)  
}).catch(err => {  
  console.error(err)  
})
```

🔗 获取别名信息

如下代码用于查询特定别名详情：

```
cfcClient.getAlias(functionName, 'prod').then(response => {
  console.log(response.body)
})
```

更新别名信息

如下代码用于修改函数别名的指向及描述信息：

```
var body = {
  'FunctionVersion': '2',
  'Description': 'update to version 2'
}

client.updateAlias(functionName, 'prod', body).then(response => {
  console.log(response.body)
}).catch(err => {
  console.log(err)
})
```

删除别名

如下代码用于删除指定别名：

```
cfcClient.deleteAlias(functionName, 'prod').catch(err => {
  console.error(err)
})
```

触发器操作

各接口的请求参数和响应参数说明请参考[触发器操作](#)。

触发器为函数BRN级别，函数不同的版本、别名函数均可关联触发器。因此在以下的增删改查示例中，函数相关的标识符均使用函数BRN而非函数名。

根据后端实现的不同，函数计算触发器可以分成relation和eventSourceMapping两种类型，对应两种API和SDK调用方法，具体分类如下表格所示。在使用SDK时需要根据触发器的类型调用不同的方法。

类型	触发器种类
relation	DuerOS、BOS、HTTP、定时任务、CDN、DuEdge
eventSourceMapping	百度消息服务

获取触发器列表

如下代码用于获取函数relation触发器列表：

```
var options = {
  'FunctionBrn': 'brn:bce:cfc:su:7c54a6e14823a11c9f9f5345952ac336:function:test:$LATEST'
}

cfcClient.listRelations(options).then(response => {
  console.log(response.body)
})
```

如下代码用于获取函数eventSourceMapping触发器列表：

```
var options = {
  'FunctionName': 'brn:bce:cfc:su:7c54a6e14823a11c9f9f5345952ac336:function:test:$LATEST'
}

cfcClient.listEventSourceMappings(options).then(response => {
  console.log(response.data)
})
```

创建触发器

如下代码用于创建HTTP触发器：

```
// 不同触发器的Source和Data字段不同，具体请参考接口文档。
var body = {
  'Target': 'brn:bce:cfc:su:7c54a6e14823a11c9f9f5345952ac336:function:test:$LATEST', // 函数 BRN
  'Source': 'cfc-http-trigger/v1/CFCAPI', // 触发源类型为http
  'Data': { // http触发器的配置参数
    'AuthType': 'anonymous',
    'Method': 'GET,POST',
    'ResourcePath': '/helloworld'
  }
}

cfcClient.createRelation(body).then(response => {
  console.log(response.data)
}).catch(err => {
  console.error(err)
})
```

如下代码用于创建百度消息服务触发器：

```
var body = {
  'BatchSize': 100,
  'Enabled': true,
  'EventSourceBrn': '7c54a6e14823a11c9f9f5345952ac336__test_topic',
  'FunctionName': 'brn:bce:cfc:bj:7c54a6e14823a11c9f9f5345952ac336:function:test:$LATEST',
  'Source': 'kafka',
  'StartingPosition': 'TRIM_HORIZON'
}

cfcClient.createEventSourceMapping(body).then(response => {
  console.log(response.body)
}).catch(err => {
  console.error(err)
})
```

更新触发器

如下代码用于更新HTTP触发器：

```

var updateBody = {
  // RelationId 触发器的唯一标识，从 listRelations 的返回结果中获取
  'RelationId': 'brn:bce:cfc-http-
trigger:su:7c54a6e14823a11c9f9f5345952ac336:8b1ddba7fec6f3e1506cad8e5a5512fb/cfc/GET,POST/test/helloworld',

  'Target': 'brn:bce:cfc:su:7c54a6e14823a11c9f9f5345952ac336:function:test:$LATEST', // 函数 BRN
  'Source': 'cfc-http-trigger/v1/CFCAPI', // 触发源类型为http
  'Data': { // http触发器的配置参数
    'AuthType': 'anonymous',
    'Method': 'GET,POST,PUT',
    'ResourcePath': '/hellocfc'
  }
};

cfcClient.updateRelation(updateBody).then(response => {
  console.log(response.body)
}).catch(err => {
  console.log(err)
})

```

如下代码用于更新百度消息服务触发器：

```

var updateBody = {
  'BatchSize': 150,
  'FunctionName': 'brn:bce:cfc:su:7c54a6e14823a11c9f9f5345952ac336:function:test:$LATEST'
}

// esmId 为触发器唯一标识，从listEventSourceMappings的返回结果中获取
cfcClient.updateEventSourceMapping(esmId, updateBody).then(response => {
  console.log(response.body)
}).catch(err => {
  console.error(err)
})

```

删除触发器

如下代码用于删除HTTP触发器：

```

var deleteOptions = {
  'Target': 'brn:bce:cfc:su:7c54a6e14823a11c9f9f5345952ac336:function:test:$LATEST',
  'Source': 'cfc-http-trigger/v1/CFCAPI',
  'RelationId': 'brn:bce:cfc-http-
trigger:su:7c54a6e14823a11c9f9f5345952ac336:8b1ddba7fec6f3e1506cad8e5a5512fb/cfc/GET,POST/test/helloworld'
}

cfcClient.deleteRelation(deleteOptions).catch(err => {
  console.error(err)
})

```

如下代码用于删除百度消息服务触发器：

```

// esmId 为列表接口返回的触发器 id
cfcClient.deleteEventSourceMapping(esmId).catch(err => {
  console.error(err)
})

```

版本变更记录

🔗 1.0.0-rc.27 (2021-03-03)

Feature

- CFCCClient 支持触发器 API
- 增加 cfc-client.d.ts 类型定义文件

视频专区

技术讲解

提示：点击图片跳转到具体的技术讲解视频教程

🔗 Docker入门实战

- 讲解什么是Docker，如何使用Docker。



🔗 Kubernetes工作原理及使用

- 介绍Kubernetes架构剖析，教你快速搭建Kubernetes集群。



🔗 Kubernetes应用部署

- 介绍Kubernetes应用的组成、描述、部署与访问。



🔗 Kubernetes实现应用的高可用

- 介绍Kubernetes探针与Deployment的使用。



🔗 Kubernetes日志采集与监控告警

- 介绍容器日志/容器监控指标的采集与管理。



常见问题

账号问题

🔗 CFC对应的AK/SK如何确定？

您可以在控制台页面右上角的“账户信息-->安全认证--Accesskey”里面创建或获取AK/SK。每个账户最多可以创建20对AK/SK，详细参考：[如何获取AKSK](#)。

🔗 创建函数提示没有权限？

请检查您的账号是否欠费，如有欠费，请先完成缴费。如果没有欠费，请您检查是否开通了CFC服务。如果以上都没有问题，您可以发起[工单](#)，我们会马上安排专业工程师为您解答。

🔗 使用CFC必须企业/个人实名认证吗？

由于工信部要求，CFC产品都需要做企业/个人高级实名认证后方可进行使用。

🔗 子用户创建函数失败，显示无权限怎么处理？

目前子用户针对CFC服务进行操作，需要授予对应的操作权限，请联系主账号管理员。主账号开通请参考：[开通权限](#)

用户中心 / 子用户

子用户管理

子用户登录链接

自定义

+ 创建子用户

请输入用户名或AccessKey进行搜索

用户名	密码	说明	状态	所在组	创建时间	操作
	未设置		活跃	-	2022-04-06 22:06:49	编辑权限 禁用 删除 管理
	已绑	-	活跃	-	2022-03-24 13:57:07	编辑权限 禁用 删除 管理
	已绑	-	活跃	-	2021-12-24 11:30:14	编辑权限 禁用 删除 管理
	已绑	-	活跃	-	2021-11-25 11:07:27	编辑权限 禁用 删除 管理
	已绑定	-	活跃	-	2021-08-24 11:09:36	编辑权限 禁用 删除 管理

网络问题

☞ CFC是否支持IPv6网络？

CFC暂时还不支持IPv6网络。

☞ DDoS攻击对CFC有影响么？

DDoS流量攻击一般针对是固定IP，CFC服务提供的访问接口是百度智能云的域名，真实IP是隐藏的，因此一般无法被DoS攻击。

资源问题

☞ 通过BOS上传代码不成功？

CFC对上传代码的压缩包以及代码解压后的大小均有限制，请确认您的资源是否超过限制：[使用限制](#)。

☞ 账号下是否有限制函数创建个数？

CFC对创建函数的个数没有限制，目前只对函数资源和账户资源有限制。

函数相关

☞ 如何开始使用CFC？

您可以通过[快速入门](#)，创建并运行您的第一个函数。

☞ CFC函数的开发流程？

CFC函数的开发流程主要包括以下过程：

1. 编写及调试代码：根据业务需求编写函数代码，您可以通过参考[创建函数](#)文档创建函数，并参考各语言运行时开发指南进行函数代码开发，如[Golang函数开发指南](#)。
2. 打包：将业务代码及其依赖打包成zip格式，打包要求及各语言部署程序包打包说明参见[创建部署程序包](#)。
3. 部署：使用API接口创建或更新函数，也可以在CFC控制台使用生成的zip文件创建或更新函数。
4. 配置触发器(可选)：如果您的函数需要被某些特定事件触发执行，则可以为函数配置对应的触发器。
5. 执行函数：当触发器对应的事件发生时，函数将被调用执行，函数计算自动为其分配资源，无需您手动干预。
6. 查看日志：在函数执行后，您可以在运行日志中查看函数的执行结果和监控数据。

☞ CFC支持哪些语言？

CFC目前支持的运行环境有Node.js 10、Node.js 12、Node.js 14、Node.js 16、Python 2.7、Python 3.6、Python 3.10、Java 8、Golang、PHP 7.2、Lua 5.3、.Net Core 2.2、PowerShell 6.2。

☞ CFC的函数之间能否互相调用？

CFC支持函数之间的互相调用，您可以在CFC函数中使用SDK调用CFC的接口，也可以通过异步调用策略，配置并调用CFC的函

异步策略配置

异步策略配置

重试次数

3

?

最长保留时间(s)

21600

?

异步目标配置

条件: 成功时

目标类型

cfc

?

目标配置

服务: default

函数: test

版本/别名: \$LATEST

?

目标

brn:bce:cfc:bj: est:\$LATEST

条件: 失败时

目标类型

无

?

确定

取消

☞ CFC支持哪些代码在线编辑？

解释型语言如PHP、Python、Node.js、Lua、PowerShell可在线编辑，其他类型语言如Java、.NET、Golang等暂时不支持在线编辑。

☞ 读取同一路径的配置文件时出错？

请检查您是否使用了绝对路径，建议您将绝对路径改为相对路径的方式。

☞ CFC上传代码有哪几种方式？

上传代码压缩包到CFC有4种方式，请注意CFC对代码压缩包以及代码解压后的大小均有限制，详细请参考[资源使用限制](#)。

- 您可以通过Console端直接上传您的代码压缩包。
- 您可以在Console端使用BOS上传您的代码压缩包，需要您开通BOS服务，并上传时候需要确定“BOS Bucket”和“对象文件”。
- 您可以使用CFC的[SDK](#)，调用上传代码接口完成上传，CFC目前支持Java、Python、Golang、Javascript四种SDK。
- 您可以调用CFC的更新代码的[API](#)接口，完成上传。

☞ 函数内输出时间与当前不对应？

CFC函数内的时间戳为Unix时间戳(Unix timestamp)，时间为UTC世界标准时间。如果您需要在函数内打印格式化时间，需要指定时区，例如“Asia/Shanghai”。CFC提供了Node.js和Python两种语言的示例代码，[CFC时间戳与时区](#)。

☞ 第一次执行函数，等待时间过长？

第一次执行函数时间较长的原因是函数在需要响应事件的容器中运行，存在一定的延时(启动容器和runtime的耗时)。这被称为“冷启动”。当你的函数执行完成后，你的容器可能会保留一段时间。如果另一个事件在此时被触发，则它的响应速度要快得多，这通常被称为“热启动”。如果函数的执行时长已经超过了设置的超时时间，请检查您的函数是否存在死循环。如果是调用接口等待时间过长，您可以调大超时时间。有关冷热启动的概念您可参考[冷启动优化](#)。

☞ CFC函数的并发度限制？

并发度为函数可以并发执行的数量上限。当前账户的总并发度限额为300。在某个地域的某个时刻，您的账户下最多可以有300个请求在同时执行。如果您想要提高并发度限制，您可以发起[工单](#)联系CFC工程师。

362

🔗 CFC函数名称能否重复创建？

同一账号的一个地域下，只能有一个同名函数，函数的bm会作为函数的唯一标识。

触发器相关

🔗 什么是触发器，CFC目前支持哪些触发器？

可参考[CFC触发器相关概念和使用](#)

🔗 触发器Event格式报错？

CFC目前只支持json类型的Event，请检查您的Event是否是正确的json数据。

🔗 添加DuerOS触发器失败？

CFC目前只支持北京地域DuerOS触发器，其他地域暂不支持，请确认您是否创建了其他地域的DuerOS触发器。

🔗 创建HTTP 触发器报错？

请您查看URL路径的命名规则，并仔细检查输入的url路径是否符合要求。如果URL没有问题，仍然创建失败，请您发起[工单](#)，我们会立马安排专业的工程师为您解答问题。

新增触发器 ×

触发器：

HTTP触发器

* URL路径：

/hello/world

填写以 / 开头的匹配路径，完整 URL 在触发器创建完成后可以查看
使用{param}指定路径参数，例如 /user/{id}/posts 中，{id} 即为匹配参数
使用{param+}指定贪婪匹配路径参数，它会匹配当前位置直到 URL 结尾的
所有剩余部分，例如 /path/{filepath+}

* HTTP方法：

请选择HTTP方法

?

* 身份验证：

☒ 不验证

☐ IAM

☐ 使用二进制请求body

确定

取消

🔗 HTTP触发器如何使用iam？

请参考[iam鉴权](#)

🔗 HTTP触发器如何使用二进制请求body？

请确保您在创建HTTP触发器的时候，勾选了"使用二进制请求body"。CFC系统会自动用base64转一下二进制body，因此需要您在代码里使用base64解码才能拿到原始二进制数据。

新增触发器

触发器：

HTTP触发器

* URL路径：

/hello/world

填写以 / 开头的匹配路径，完整 URL 在触发器创建完成后可以查看
使用{param}指定路径参数，例如 /user/{id}/posts 中，{id} 即为匹配参数
使用{param+}指定贪婪匹配路径参数，它会匹配当前位置直到 URL 结尾的
所有剩余部分，例如 /path/{filepath+}

* HTTP方法：

请选择HTTP方法

?

* 身份验证：

☒ 不验证

☐ IAM

☒ 使用二进制请求body

确定

取消

🔗 使用定时任务触发器，时间相差八个小时？

CFC的crontab所有表达式都使用UTC时区，计划的最小精度为1分钟，Asia/Shanghai时区比UTC时区多八个小时。例如每天上午的10:00(UTC)触发，对应Asia/Shanghai为每天下午18:00执行。

🔗 HTTP 触发器如何跨域调用？

HTTP触发器使用可参考<https://cloud.baidu.com/doc/CFC/s/kjzmgugty>，并在函数的response中设置 允许跨域的响应头 Access-Control-Allow-Origin：，示例如下

```
1  # -*- coding: utf-8 -*-
2
3  import json
4
5  def handler(event, context):
6      resp = {
7          "isBase64Encoded": False,
8          "statusCode": 200,
9          "headers": {
10             "Access-Control-Allow-Origin": "*"
11         },
12         "body": "hello"
13     }
14     return json.dumps(resp)
```

🔗 dueros触发器的使用？

CFC和dueros触发器的使用，您可以参考视频完成配置。 <http://duerstatic.bj.bcebos.com/didp-course/dbp/5%E9%85%8D%E7%BD%AE%E9%83%A8%E7%BD%B2%E2%80%9494CFC%E4%BD%BF%E7%94%A8.mp4>

日志相关

🔗 如何查看CFC函数日志？

点击函数执行后，您可以看到当条函数的执行信息。如果您想查看之前执行的函数日志，您可以点击CFC导航页的日志，配置百度日志服务BLS。您可以选择百度对象存储BOS或者百度日志服务BLS来存储日志。详细请参考：[配置并查看CFC函数日志](#)。

监控相关

🔗 CFC监控服务都有哪些功能和指标？

- CFC支持7*24不间断的监控，监控粒度为1分钟，最长支持40天的监控数据。
- 监控信息包括调用次数、平均耗时、调用错误次数、系统内部错误次数、最大运行内存、时间内存、并发超限次数。
- 您也可以函数配置报警信息，当函数执行出现异常时及时通知您。您可以设置“监控项”、“统计周期”、“统计方法”、“阈值”、“重复几次后报警”、“报警等级”等指标。

计费相关

🔗 CFC免费额度有多少？

CFC提供的免费资源包括：100万次/每个月的调用次数、40万GB秒的资源占用时间、1GB的公网流量。

🔗 CFC是如何计费的？

函数计算提供了一定的免费额度，同时支持资源套餐包、按需付费两种计费方式。详细计费数据您可参考[CFC价格详情](#)。

- 资源套餐包，包含三种类型的资源包：资源使用量、函数调用次数和外网出流量。购买资源包后，调用函数时，将优先使用资源包抵扣。若无有效资源包或资源包已用完，您仍可使用免费额度，免费额度用完后则转为按需计费结算。
- 按需付费，在您创建函数时无需付费，之后根据您使用函数时实际产生的调用次数、函数运行占用资源的时间和公网流量计费。总费用=函数调用次数费用+函数运行占用资源时间费用+公网流量费用。

🔗 CFC的扣费顺序？

CFC包含一定的免费额度和资源套餐包、按需付费两种计费方式。当同时包含上述三种方式时候，优先使用资源套餐包，当资源套餐包使用完毕或者到期后，您仍可使用免费额度。若免费额度用完，则进行按需计费结算。每个主账号最多可购买99个资源包，在有多个资源包的情况下，按到期时长的从近到远按顺序扣减。例如，A包1个月后到期，B包3个月后到期，会预先扣减A包用量，扣除为0后再扣除B包用量；如果到期时间相同，则按购买时的下单先后顺序抵扣。

🔗 CFC欠费已充值依然无法使用怎么处理？

CFC欠费充值后需要一段时间才能生效并正常使用，请耐心等待，10分钟后尝试重新刷新。

🔗 如何索要CFC发票？

您可以走线上流程，在“财务中心>发票管理”进行申请。

🔗 资源包购买后，如何使用？

资源套餐包购买后立即生效，直接使用即可。

🔗 使用资源套餐包后，查看已使用量包数据没有立即生效？

用量包是在每个小时的第20分钟，统一计算上个小时的用量数据。比如您在上午9点40分使用了CFC资源，会在10点20分计算剩余量，此时你可以查看套餐包的剩余量。如果您在10点到10点20分之间使用了CFC资源，那么将会在11点20分钟，统一计算剩余量。

🔗 如何停止cfc计费？

CFC在函数调用时候才会产生费用，您不再调用CFC函数就不会产生费用。如果您不再使用CFC，可以删除CFC下的所有函数资源。

🔗 购买资源包失败？

请检查您的账单是否欠费，如果欠费请补齐欠款，等待10分钟左右后重试。如果没有欠费，请发工单联系CFC工程师为您解答。

API&SDK&工具

🔗 调用CFC的API返回接口参数的含义？

您可使用HTTP或者HTTPS请求调用CFC API，包括“函数调用”、“函数操作”、“版本操作”、“别名操作”、“触发器操作”、“工作流操作”，有关数据类型您可参考[数据类型](#)。

🔗 如何根据CFC的错误响应定位原因？

当用户访问CFC的API出现错误时，CFC会返回用户错误码和错误信息，便于用户定位问题。详细的错误码和错误信息解释请参见[错误码](#)。

🔗 CFC支持哪些语言的SDK？

CFC目前支持Java、Python、Golang、Javascript四种语言的SDK。

🔗 调用CFC的API接口出现403和QualifyNotPass？

请检查您是否已经完成实名认证，没有通过实名认证的可以前往[百度智能云官网控制台](#)中的安全认证下的实名认证中进行认证。

🔗 CFC开发工具有哪些？

CFC为开发者提供了VSCode 插件和命令行工具

- [Baidu Serverless VSCode Extension](#)是百度智能云Serverless产品CFC的VSCode插件，该插件结合了CFC本地命令行工具BSAM CLI，可以帮您快速创建、管理本地的CFC函数，同时借助Docker轻松实现函数在本地的执行和Debug。
- [BSAM CLI](#)是一个基于BCE SAM规范的命令行工具，它提供了本地开发环境，帮助您在把函数上传到百度智能云CFC之前，在本地进行函数的开发、分析和执行。

🔗 如何安装函数依赖？

python安装依赖，您可以参考<https://cloud.baidu.com/doc/CFC/s/9jzqubw7f> golang安装依赖，可以使用vendor将依赖下载到本地，后面可以参考<https://cloud.baidu.com/doc/CFC/s/Ujxtzqqg7> 构建二进制。

🔗 使用API/SDK/http触发器执行函数，一直提示报错“function response err”？

请检查您的账单是否欠费，如果欠费请补齐欠款，等待10分钟左右后重试。如果没有欠费，请发工单联系CFC工程师为您解答。

其他

🔗 CFC能否支持主流web开发框架迁移？

CFC目前支持[Java Springboot Web](#)框架迁移到函数计算，并支持搭建[Flask-Web](#)框架。其他框架暂时不支持，如果您有此需求，可发起[工单](#)询问CFC工程师。

支持

如果您遇到无法解决的问题，您可以申请[工单](#)，我们会立马安排专业的工程师为您解答问题。

工单服务时间：工作日9：30 - 19：00

服务条款

CFC服务等级协议

协议生效时间：2021年05月18日

本服务等级协议（Service Level Agreement，以下简称 "SLA"）规定了百度智能云向客户提供的函数计算服务CFC（简称"CFC"）的服务可用性等级指标及赔偿方案。

在您使用百度智能云函数服务之前，请您仔细阅读本服务协议的全部内容。您使用本服务的任何行为，即视为您已阅读并同意下列协议、规则等的约束。

1. 相关定义

1.1 百度智能云函数计算服务（Cloud Function Compute，CFC）

是指百度智能云为客户提供函数计算的在线托管运行服务。

1.2 服务周期

一个服务周期为一个自然月，总分钟数 = 服务周期内的总天数 x 24（小时）x 60（分钟）计算。

1.3 服务不可用时长

服务在每五分钟的统计频次内错误率超过5%时，视为该单位时间内的服务不可用。

1.4 响应失败请求

是指服务内部产生的错误，即响应状态码为5xx的请求。

1.5 错误率定义

错误率 = 五分钟内的响应失败请求次数 / 该五分钟内的CFC正常收到的有效请求总数

2 服务可用性

2.1 服务可用性计算公式

服务可用性 = (1-服务周期内总的响应失败请求数 / 服务周期内收到的有效请求总数) * 100%

2.2 服务可用性

百度智能云CFC承诺可用性不低于99.95%，如果未达到上述标准，可以根据本协议第3条约定获得赔偿。

2.3 除外情形

赔偿范围不包括以下原因所导致的不可用：

- (1) 百度智能云预先通知用户后进行系统维护所引起的，包括割接、维修、升级和模拟故障演练；
- (2) 任何百度智能云所属设备以外的网络、设备故障或配置调整引起的；
- (3) 用户的应用程序或数据信息受到黑客攻击而引起的；
- (4) 用户维护不当或保密不当致使数据、口令、密码等丢失或泄漏所引起的；
- (5) 用户的疏忽或由用户授权的操作所引起的；
- (6) 不可抗力以及意外事件引起的；
- (7) CFC服务只赔付CFC自身，不赔付CFC实例所关联的其他云服务。
- (8) 其他非百度智能云原因所造成的CFC服务无法正常使用。

3 赔偿条款

3.1 赔偿方式

CFC将以百度智能云代金券的形式对用户进行赔偿，其中代金券的使用规则需以百度智能云官方发布的代金券相关规则为准。所发代金券仅能通过当前账户购买本服务，不能购买其他服务，也不能对代金券进行交易、转让、赠予等。

3.2 赔偿标准

赔偿将以上述的代金券形式进行发放，当某服务月度并未达到服务可用性标准，赔偿额度则需要单独进行计算，赔偿的总额度不超过相应未达标月度内用户所支付的服务费（所支付的服务费不包含代金券、优惠券以及折扣减免等各种抵扣方式）总额。

服务可用性	补偿金额（代金券）
低于99.95%但等于或高于99.00%	月度服务费用的10%
低于99.00%但等于或高于95.00%	月度服务费用的25%
低于95.00%	月度服务费用的100%

3.3 赔偿申请时限

用户可以通过相应用户的工单系统在次月的第五个工作日后，以及服务结束的60个自然日之前（包括第60天）提出相应的赔偿申请。提出申请后，将会由百度智能云进行相应核实，核实完成后，将会把代金券发放至对应账户。若双方出现争议，双方需以百度智能云的后台记录为准。

4. 其他说明

- (1) 在法律法规允许的范围内，百度智能云负责对本协议进行解释说明。
- (2) 本协议一经公布立即生效，百度智能云有权对本SLA条款作出修改。如本SLA条款有任何修改，百度智能云将以网站公示或发送邮件的方式通知您。如您不同意百度智能云对SLA所做的修改，您有权停止使用CFC服务，如您继续使用CFC服务，则视为您接受修改后的SLA。
- (3) 协议项下百度智能云对于用户所有的通知均可通过网页公告、站内信、电子邮件、手机短信或其他形式等方式进行；该通知于发送之日视为已送达收件人。因用户未及时获知百度智能云的服务变更或终止条款遭受损失的，百度智能云不承担任何责任。
- (4) 本协议的订立、执行和解释及争议的解决均应适用中国法律并受中国法院管辖。如双方就本协议内容或其执行发生任何争议，双方应尽量友好协商解决；协商不成时，任何一方均可向北京市海淀区人民法院提起诉讼。
- (5) 本协议构成双方对本协议之约定事项及其他有关事宜的完整协议，除本协议规定的之外，未赋予本协议各方其他权利。
- (6) 如本协议中的任何协议无论因何种原因完全或部分无效或不具有执行力，本协议的其余协议仍应有效并且有约束力。
- (7) 关于用户约束条款，详见《[百度智能云用户服务协议](#)》中的"用户的权利与义务"相关条款内容。
- (8) 关于服务商免责条款，详见《[百度智能云用户服务协议](#)》中的"免责声明"相关条款内容。