百度智能云

BOS 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 29
产品公告 38
安全与合规 38
概述 38
访问控制 39
数据加密 40
监控审计 41
数据容灾 41
数据保留合规 42
数据保护 42
产品描述 42
产品简介 43
核心概念 43
分级存储介绍 44
存储类型介绍 44
为什么会有分级存储 45
产品优势 46
产品功能 47
应用场景 60
使用限制 63
产品定价 65
计费项介绍 65
计费项概览 65
存储费用 65
流量费用 67
请求费用 67
数据处理费用 68
产品价格 70
按需计费方式 70
按需价格清单 70
资源包价格清单 71
资源包计费方式 72
查看消费明细 73
计费案例 76
余额不足和欠费提醒 78
计费常见问题 78
快速入门 80
开始使用BOS 80
快速入门流程 81

创建Bucket 81
上传Object 84
获取Object 85
删除Object 85
删除Bucket 86
控制台操作指南 87
操作准备 87
管理Bucket 87
创建Bucket 87
Bucket概览 89
删除Bucket 90
设置Referer白名单 90
设置服务端加密 91
设置跨域访问 92
设置数据同步 94
设置镜像回源 96
设置访问日志 101
设置静态网站托管 103
设置回收站 104
管理Bucket标签 105
设置存储空间清单 107
设置合规保留 110
查询Bucket 111
请求者付费 112
设置Bucket读写权限 112
Bucket额度管理 116
管理生命周期 117
使用限制 121
注意事项 121
费用说明 121
注意事项 122
设置日志查询 122
设置版本控制 123
数据访问趋势分析 124
管理Object 126
上传文件 126
下载文件 127
复制和移动文件 128
重命名文件 129
删除文件 129
设置Meta信息 130

设置文件权限 131
管理文件夹 132
碎片管理 133
转换文件存储类型 134
Select扫描 136
设置软链接 138
批量删除文件夹 139
文件夹统计 140
分享文件-获取文件url 140
文件夹分享 141
查询Object 144
概述 144
控制台查看全部Object 144
取回归档文件 145
批量管理Object 146
对象标签 148
数据发布 148
数据发布简介 148
查看Bucket官方域名 149
创建Bucket自定义域名 149
开通CDN官方加速域名 152
配置CDN上行加速 152
配置缓存管理 153
全球加速 154
查看并导出用量数据 156
事件通知及数据处理 157
配置BOS多用户访问控制 160
图片审核服务 163
图片服务 164
统计与监控 167
额度管理 169
文件自动解压 170
分层命名空间存储桶操作说明 172
联动百度网盘分享 173
全局设置 175
VPC访问控制 175
收藏路径 176
应用场景 176
开发者指南 177
基本概念 177

开发准备 178
Region与Endpoint 178
Bucket基本操作 178
创建Bucket 178
获取Bucket地域信息 179
查看Bucket列表 179
设置跨域资源共享 180
管理生命周期 180
从规则配置上划分，基础生命周期管理支持简单与复杂模式 181
设置Bucket标签 185
删除Bucket 185
设置访问日志 186
数据发布 187
Object基本操作 187
上传数据 187
下载数据 190
管理文件 191
数据发布 200
绑定自定义域名 200
CDN加速发布 201
数据安全 201
权限控制 201
设置服务端加密 204
设置防盗链 205
设置回收站 205
版本控制 206
最小权限原则 208
数据容灾 209
多AZ存储 209
数据同步 209
事件通知 210
数据迁移上云 211
场景概述 211
本地数据上云 211
从其他云迁移 212
静态网站托管 213
图像审核 213
单链接限速 224
在请求header中指定上传的限速值为819200bit/s，即100KB/s： 224
在请求参数中指定上传的限速值为819200bit/s，即100KB/S： 225
在请求header中指定下载的限速值为819200bit/s，即100KB/s： 225

在请求参数中指定下载的限速值为819200bit/s，即100KB/S： 225
数据湖接入 225
分层命名空间 225
元数据转换服务 226
监控服务操作 227
数据处理 234
数据处理介绍 234
图像通用设置 235
原图保护 235
图像处理指南V1.0 236
使用规则 236
图像缩略 238
图像裁剪 240
图像旋转 242
渐近显示 243
水印 243
质量变换 248
格式转换 249
获取图片信息 249
图像处理指南V2.0 251
使用规则 251
图像缩放 253
图像裁剪 257
图像旋转 259
渐近显示 259
质量变换 260
图像水印 260
获取图片信息 266
格式转换 266
灰度变换 266
图片锐化 267
图片模糊 267
图片亮度 267
去除元信息 268
图片对比度 268
智能处理 269
图像审核 282
图像处理名词解释 283
音视频处理 284
数据处理常见问题 285
视频截帧 285

基本概念 285
文档预览 287
视频即时转码 288
基本概念 288
获取视频元信息 289
数据湖存储 290
分层命名空间 290
服务介绍 290
功能支持说明 291
使用方式 291
普通存储桶转换为分层命名空间存储桶 291
数据迁移和流转 292
使用 DistCp 迁移数据至分层命名空间 Bucket 292
DataX 读写 BOS 293
BOS HDFS 工具 294
概述 294
配置和使用 295
常见问题 298
大数据组件使用指南 298
Flink 使用指南 298
Hive 使用指南 299
Spark 使用指南 301
Presto 使用指南 302
Flume 数据存储到 BOS 303
Impala 使用指南 304
Logstash数据存储到BOS 306
Elasticsearch快照数据存储到BOS 307
工具概述 307
Fluentd收集数据存储到BOS 308
工具概述 308
Kafka数据存储到BOS 309
Hbase使用BOS作为底层存储 310
Druid使用BOS作为Deep storage 312
数据魔方 312
批量处理 312
zip打包压缩 314
智能数据处理 318
视频处理 318
图像处理 323
文档处理 375

内容审核 378
图像审核 378
视频审核 379
API参考 381
简介 381
接口规范 382
错误码 383
访问控制 387
Service相关接口 401
额度管理 401
请求语法 402
请求参数 402
请求头域 402
响应头域 402
响应元素 402
请求示例 402
响应示例 402
请求语法 402
请求参数 402
请求头域 402
响应头域 402
响应参数 402
请求示例 403
响应示例 403
请求语法 403
请求参数 403
请求头域 403
响应头域 403
响应参数 403
请求示例 403
响应示例 403
ListBuckets 403
请求语法 403
请求头域 403
请求参数 403
响应头域 403
响应元素 403
JSON请求示例 404
JSON请求响应示例 404
Bucket相关接口 404
基础操作 404

请求语法 404
请求头域 404
请求参数 404
响应头域 404
响应元素 404
请求示例 405
请求语法 405
请求头域 405
请求参数 405
响应头域 405
响应元素 405
请求示例 405
响应示例 405
请求语法 406
请求头域 406
请求参数 406
响应头域 406
响应元素 406
请求示例 406
响应示例 406
请求语法 406
请求头域 406
请求参数 406
响应头域 406
响应元素 406
请求示例 406
响应示例 407
请求语法 407
请求头域 407
请求参数 407
响应元素 407
响应头域 407
请求示例 407
响应示例 407
请求语法 407
请求头域 408
请求参数 408
响应头域 408
响应元素 408
请求示例 408
响应示例 408

权限控制 408
请求语法 408
请求参数 408
请求头域 408
响应头域 408
响应元素 409
使用CannedAcl的请求示例 409
上传ACL文件的示例 409
响应示例 409
请求语法 409
请求头域 409
请求参数 409
响应元素（以JSON请求为例） 409
响应头域 409
请求示例 410
响应示例 410
生命周期 410
请求语法 410
请求头域 410
请求参数 410
响应元素 411
响应头域 411
请求示例 411
响应示例 412
请求示例 412
响应示例 413
请求示例 413
响应示例 414
请求语法 414
请求头域 414
请求参数 414
请求参数 414
响应元素 415
响应头域 415
请求示例 415
响应示例 415
请求语法 416
响应示例 416
请求语法 417
请求头域 417
请求参数 417

响应元素 417
响应头域 417
请求示例 417
响应示例 417
日志管理 418
请求语法 418
请求头域 418
请求参数 418
响应元素 418
响应头域 418
请求示例 418
响应示例 418
请求语法 419
请求头域 419
请求参数 419
响应元素 419
响应头域 419
请求示例 419
响应示例（已设置Log规则） 419
响应示例（未设置Log规则） 419
请求语法 419
请求头域 419
请求参数 420
响应元素 420
响应头域 420
请求示例 420
响应示例 420
事件通知 420
请求语法 420
请求参数 420
请求头域 421
响应头域 421
响应元素 421
请求示例 421
响应示例 422
请求语法 422
请求头域 422
请求参数 422
响应头域 422
响应元素 422
请求示例 423

响应示例 423
请求语法 424
请求头域 424
请求参数 424
响应头域 424
响应元素 424
请求示例 424
响应示例 424
请求语法 425
请求头域 425
请求参数 425
响应头域 425
响应元素 425
请求示例 426
响应示例 426
请求语法 427
请求参数 427
请求头域 427
响应头域 427
响应元素 427
请求示例 427
响应示例 428
静态网站 428
请求头域 428
请求参数 428
响应元素 428
响应头域 428
请求示例 428
响应示例 429
返回代码 429
请求头域 429
请求语法 429
响应元素 429
响应示例 429
返回代码 429
请求头域 430
请求语法 430
响应元素 430
响应头域 430
响应示例 430

返回代码 430
数据加密 430
请求语法 430
请求参数 430
请求头域 430
请求示例1 431
返回示例1 431
请求示例2 431
返回示例2 431
请求语法 431
请求参数 431
请求头域 431
响应元素 431
响应头域 431
请求示例1 431
返回示例1 431
请求示例2 432
返回示例2 432
请求语法 432
请求参数 432
请求头域 432
响应元素 432
响应头域 432
请求示例 432
返回示例 432
跨域访问 432
请求语法 433
请求参数 433
请求头域 433
请求元素 433
请求示例 434
响应示例 434
请求语法 434
请求参数 434
请求头域 434
请求元素 434
响应头域 434
响应元素 434
请求示例 435
响应示例 435
请求语法 436

请求参数 436
请求头域 436
请求元素 436
响应头域 436
响应元素 436
请求示例 437
响应示例 437
回收站 437
请求语法 437
请求头域 437
请求元素 437
响应头域 437
响应元素 437
请求示例 437
响应示例 437
请求语法 438
请求头域 438
请求元素 438
响应头域 438
响应元素 438
请求示例 438
响应示例 438
请求语法 438
请求头域 438
请求元素 438
响应头域 439
响应元素 439
数据同步 439
请求语法 439
请求头域 440
请求参数 440
响应元素 440
响应头域 440
请求示例 440
响应示例 441
请求语法 441
请求头域 441
请求参数 441
响应元素 441
响应头域 441
请求示例 441

响应示例 442
请求语法 442
请求头域 442
请求参数 442
响应元素 442
响应头域 442
请求示例 442
响应示例 442
请求语法 442
请求头域 443
请求参数 443
响应元素 443
响应头域 443
请求语法 443
请求头域 443
请求参数 443
响应元素 443
响应头域 444
请求示例 444
响应示例 444
合规保留 444
请求语法 444
请求头域 445
请求参数 445
响应头域 445
响应元素 445
请求示例 445
响应示例 445
请求语法 445
请求参数 445
请求头域 445
响应头域 445
响应元素 445
请求示例 445
响应示例 446
请求语法 446
请求参数 446
请求头域 446
响应头域 446
响应元素 446
请求示例 446

响应示例 446
请求语法 446
请求参数 446
请求头域 446
响应头域 446
响应元素 447
请求示例 447
响应示例 447
返回代码 447
请求语法 447
请求头域 447
请求参数 447
响应头域 447
响应元素 447
请求示例 447
响应示例 447
返回代码 447
原图保护 448
存储桶清单 449
请求语法 449
请求头域 449
请求参数 449
响应头域 450
响应元素 450
请求示例 450
响应示例 450
请求语法 450
请求头域 450
请求query参数 450
响应头域 450
响应元素 450
请求示例 451
响应示例 451
请求语法 451
请求头域 451
请求参数 451
响应头域 451
响应元素 451
请求示例 452
响应示例 452

请求语法 452
请求头域 452
请求query参数 452
响应头域 453
响应元素 453
请求示例 453
响应示例 453
请求者付费 453
请求语法 453
请求头域 453
请求参数 453
响应头域 453
响应元素 453
请求示例 453
响应示例 453
请求语法 454
请求头域 454
请求参数 454
响应头域 454
响应元素 454
请求示例 454
响应示例 454
额度管理 454
请求语法 454
请求头域 454
请求参数 454
响应头域 455
响应元素 455
请求示例 455
响应示例 455
请求语法 455
请求头域 455
请求参数 455
响应头域 455
响应元素 455
请求示例 455
响应示例 455
请求语法 456
请求头域 456
请求参数 456
响应头域 456

响应元素 456
请求示例 456
响应示例 456
镜像回源 456
请求语法 456
请求头域 457
请求参数 457
响应示例 457
请求语法 457
请求头域 457
请求参数 457
响应元素 458
响应头域 458
请求示例 458
响应示例 458
请求语法 458
请求头域 459
请求参数 459
响应元素 459
响应头域 459
请求示例 459
响应示例 459
标签管理 459
请求语法 459
请求头域 459
请求参数 459
Tag结构 459
响应头域 459
响应元素 459
请求示例 459
响应示例 460
请求语法 460
请求头域 460
请求参数 460
响应头域 460
响应元素 460
请求示例 460
响应示例 460
请求语法 460
请求头域 461
请求参数 461

响应头域 461
响应元素 461
请求示例 461
响应示例 461
设置多版本状态的示例 461
响应示例 462
请求语法 462
响应元素（以JSON请求为例） 462
响应头域 462
请求示例 462
响应示例 462
Object相关接口 462
基础操作 462
请求语法 462
请求头域 462
请求参数 462
响应头域 463
响应元素 463
请求示例（JSON）1 463
响应示例（JSON）1，递归列举所有object 463
请求示例（JSON）2， 列举根目录下的部分文件和子目录，受maxKeys限制(不包括子目录下的文件) 464
响应示例（JSON）2 464
请求语法 464
请求参数 465
请求头域 465
响应头域 465
响应参数 465
请求示例 465
响应示例 465
请求语法 466
请求参数 466
响应头域 467
响应参数 467
标准存储的请求示例 467
低频/冷存储的请求示例 468
STS鉴权方式请求示例 468
响应示例 469
请求语法 469
请求参数 469
请求头域 469
响应头域 470

响应参数 470
标准存储的请求示例 470
低频/冷存储的请求示例 470
响应示例 470
多版本的请求示例 471
响应示例 471
请求语法 471
多版本请求语法 471
请求头域 471
请求参数 471
响应头域 471
响应元素 472
请求示例 472
响应示例 472
多版本的请求示例 472
响应示例 472
请求语法 473
多版本请求语法 473
请求头域 473
请求参数 473
响应头域 473
响应参数 473
请求示例 474
响应示例 474
归档类型对象已取回时响应示例 474
多版本请求示例 474
响应示例 474
请求语法 474
请求头域 474
请求参数 475
响应头域 475
响应参数 475
请求示例 475
响应示例 475
请求语法 475
请求头域 475
请求参数 476
响应头域 476
响应参数 476
请求示例（同步模式） 476

响应示例（同步模式） 476
请求示例（异步模式） 476
响应示例（异步模式） 476
请求示例（异步模式，通知回调地址） 477
响应示例（异步模式，通知回调地址） 477
请求语法 477
多版本请求语法 477
请求头域 477
请求参数 477
响应头域 477
响应参数 477
请求示例 478
响应示例 478
多版本请求示例 478
响应示例 478
请求语法 478
请求头域 478
请求参数 478
响应头域 478
响应参数 478
请求示例 479
响应示例 479
请求语法 479
请求头域 480
请求参数 480
响应头域 480
响应参数 480
标准存储的请求示例一 481
低频/冷存储的请求示例一 481
响应示例一 481
标准存储的请求示例二 481
低频/冷存储的请求示例二 481
响应示例二 481
请求语法 482
请求参数 482
请求头域 482
请求元素 482
响应头域 482
响应元素 482
请求示例 482
响应示例 482

初始化 516
Bucket管理 519
文件管理 526
数据处理及使用 544
异常处理 548
sdk日志 548
常见问题 549
版本变更记录 550
Python-SDK 551
概述 551
安装SDK工具包 551
初始化 552
Bucket管理 556
文件管理 562
日志控制 581
常见问题 581
异常处理 582
版本变更记录 583
PHP-SDK 584
概述 584
安装SDK工具包 584
初始化 585
Bucket管理 588
文件管理 594
异常处理 604
版本变更记录 604
GO-SDK 605
概述 605
安装SDK工具包 605
初始化 606
Bucket管理 609
文件管理 612
数据处理及使用 625
错误处理 630
sdk日志 630
版本变更记录 631
请求管理 631
Ruby-SDK 635
概述 635
安装SDK工具包 635
初始化 636

Bucket管理 639
文件管理 641
异常处理 648
sdk日志 649
版本变更记录 649
C++-SDK 649
概述 649
安装SDK工具包 649
初始化 651
Bucket管理 653
文件管理 656
获取及更新文件元信息 670
请求管理 673
异常处理 674
sdk日志 674
版本变更记录 675
JavaScript-SDK 675
概述 675
安装SDK 675
初始化 676
Bucket管理 681
文件管理 689
额度管理 706
版本变更记录 707
Android-SDK 708
概述 708
安装SDK工具包 708
初始化 709
Bucket管理 712
文件管理 718
异常处理 741
版本变更记录 742
日志 742
IOS-SDK 743
概述 743
安装SDK工具包 743
初始化 743
Bucket管理 747
文件管理 751
最简单的上传 752
异常处理 767

日志 767
版本变更记录 767
C#-SDK 767
概述 767
安装SDK工具包 767
初始化 768
Bucket管理 773
文件管理 776
异常处理 791
版本变更记录 792
C-SDK 792
概述 792
安装sdk工具包 792
初始化 793
Bucket管理 794
删除Bucket 795
文件管理 795
拷贝文件 804
获取及更新文件元信息 807
异常处理 810
版本变更记录 811
Harmony-SDK 811
概述 811
鸿蒙SDK 811
安装SDK工具包 811
初始化 812
Bucket管理 814
文件管理 816
完整代码 820
日志 829
异常处理 829
版本变更记录 829
周边工具 829
周边工具概述 829
下载专区 830
BOS-API生成工具 832
概述 832
如何调用API 833
Service相关接口 836
Bucket相关接口 837

Bucket数据同步相关接口 861
Object相关接口 864
图像审核服务接口 872
BOS-CMD命令行工具 873
概述 873
安装BOSCMD 873
配置BOSCMD 875
通过CMD使用BOS服务 876
设置 Bucket ACL 888
BOSCMD版本变更记录 896
配置BCECMD命令行补全功能 897
BOS CMD常见问题 897
BOS-FTP工具 897
BOS-Import数据上云工具 899
概述 899
部署方式 900
配置文件 900
任务进度恢复 902
异常处理 903
日志 903
版本记录 904
月光宝盒物理迁移设备 904
月光宝盒介绍 904
月光宝盒操作流程 904
BOS-PC客户端 906
安装及登录 906
使用指南 908
11. 搜索文件 914
常见问题 916
BOS-FS挂载工具 916
概述 916
安装 916
配置 918
操作与实践 919
挂载 919
常见问题 920
版本变更记录 920
BOS-Util批量操作工具 921
概述 921
安装BOS-Util 921
通过BOS-Util使用BOS服务 921

版本变更记录 928

BOS-Probe错误检测工具 928

概述 928

通过BOSProbe检查上传下载 929

安装BOSProbe 931

建议及报告查看 931

BOS-HDFS工具 931

概述 931

配置和使用 932

大数据组件使用指南 934

BOS-AlluxioExtension工具 942

BOS Connector for PyTorch 944

概述 944

功能及使用说明 944

安装 BOS Connector for Pytorch 944

配置 BOS Connector for Pytorch 945

构建 Dataset 945

读写 Checkpoint 946

性能测试 947

fsspec-bosfs 947

BOS-CLI命令行工具 949

概述 949

安装BOSCLI 949

配置BOSCLI 950

通过CLI使用BOS服务 951

版本变更记录 962

第三方工具 963

MinIO 963

rclone 964

Goofys 966

S3 Browser 966

其他工具 968

典型实践 968

数据迁移 968

场景简介 968

本地数据迁移到BOS 968

三方云数据迁移到BOS 969

AWS-lambda同步S3数据到BOS 970

如何解决浏览器跨域CORS问题 973

客户端加密实践 977

防盗链 981
Web数据直传至BOS 985
基础篇：在浏览器中直接上传文件到BOS 985
进阶篇一：STS临时认证 986
进阶篇二：通过PostObject接口处理IE低版本 988
原理篇一：在浏览器中直接上传文件到BOS 988
原理篇二：大文件分块上传 989
原理篇三：STS临时认证 992
手机美图APP实践 994
概述及场景介绍 994
方案实践——搭建美图APP 994
方案实践——使用美图APP 995
代码示例 996
HTTPS传输加密实践 999
混合云中使用专线-VPN访问BOS 1001
CDN动态加速上传数据到BOS 1003
使用CDN加速BOS 1004
Python API请求访问BOS 1005
基础(必看)：鉴权字符串生成 1005
GET请求下载Bucket文件到本地 1007
PUT请求上传文件到Bucket 1008
BOS专线回源IDC最佳实践 1010
通过安卓SDK使用HTTPDNS服务 1013
将 BOS 作为 Jfrog Artifactory 的后端存储服务 1015
常见问题 1016
常见问题总览 1016
一般类问题 1018
计费类问题 1018
性能类问题 1021
安全类问题 1021
认证及权限管理 1022
存储桶域名请求风格 1022
Bucket访问及管理 1022
Object上传下载及管理 1026
API&SDK&工具 1027
数据迁移与备份 1028
图片服务 1029
分析与统计 1029
AWS S3 兼容 1029
概述 1029
服务域名 1029

兼容签名认证 1030
兼容公共头 1031
兼容接口 1031
兼容工具 1032
视频专区 1041
了解BOS 1041
BOS分级存储 1041
Object上传 1041
通过域名访问BOS资源 1041
服务等级协议SLA 1041
BOS服务等级协议SLA(V3.0) 1042
相关协议 1043
百度智能云对象存储联动百度网盘使用协议 1043

功能发布记录

发布时间	功能概述
2025-07	• 存储桶ACL自定义权限中条件配置项支持 UserAgent ，您可灵活配置等于或匹配某些UserAgent禁止或者允许BOS访问资源，可提升数据安全，配置参考设置Bucket读写权限
2025-06	• BOS 分层命名空间类型的存储桶结束邀测，现已正式发布。用户可自助开通使用，详情参见 数据湖存储分层命名空间服务介绍。
2025-05	• BOS桌面客户端版本 1.4.6 发布，进行上传性能，任务管理机制优化，立即安装最新版。
2025-04	• 数据访问趋势分析，可通过分析 Bucket 或前缀粒度数据的访问趋势，帮助用户决策在何时将对应的数据转换为合适的存储类型，进而降低存储整体使用成本。同时，也可以帮助用户从访问变化的角度，分析业务层面的变化，详情参见 配置访问趋势分析 。
2025-03	• fsspec-bosfs 发布，用户可以使用标准的 fsspec 接口操作存储于 BOS 中的数据。详情参见 fsspec-bosfs 。
2025-01	• BOS Connector for Pytorch 发布，是针对 AI 训练场景所提供的高性能数据读写工具。详情参见 BOS Connector for Pytorch 概述 。 • BOS 数据魔方服务支持图像盲水印，详情参见 图像盲水印 。 • 日志查询发布，开通功能后，您可实现基于BOS日志进行操作审计、访问统计、异常事件回溯和问题定位等工作，详情参见 日志查询设置。
2024-10	• 存储桶ACL自定义权限中条件配置支持IP 不等于，可便捷实现拒绝非预期的IP 访问请求，提升数据访问安全，配置参考设置Bucket读写权限 。 • 静态网站托管中错误页配置支持配置 200 响应码 。你可配置响应码为 404 或 200，默认为 404，功能详细介绍参考设置静态网站托管 ，API配置文档请参考PutBucketStaticWebsite 。
2024-9	• BOS控制台新增AI助手，帮助解决产品使用及操作上的问题。
2024-8	• BOS桌面客户端版本 1.4.0 发布，此版本支持归档存储取回功能。可在 BOS-PC客户端 安装最新版本。
2024-7	• BOS控制台文件列表支持筛选、排序、模糊搜索。 • BOS桌面客户端版本 1.3.0 发布，此版本默认支持virtual-host域名。可在 BOS-PC客户端 安装最新版本。
2024-6	• BOS桌面客户端版本 1.2.0 发布，此版本新增碎片管理功能。可在 BOS-PC客户端 安装最新版本。
2024-5	• BOS 内容审核新增视频审核能力，详情请查看 BOS 视频审核服务 。
2024-5	• BOS桌面客户端最新版本 1.1.0 发布，此版本优化数据上传时的功能。新增能力为：数据上传时可以指定 Object 的类型、权限、自定义 meta 信息。可在 BOS-PC客户端 安装最新版本。
2024-4	• BOS桌面客户端版本 1.0.0 发布，此版本更新任务管理模块，支持对数据上传、下载、移动、复制、删除、重命名的操作任务进行查看和管理。可在 BOS-PC客户端 安装最新版本。
2024-3	• BOS桌面客户端版本 0.2.3 发布，此版本提升在 Windows 平台同步大量文件时的流畅性。可在 BOS-PC客户端 安装最新版本。
	• 数据同步支持not include语义，提升数据同步使用体验。

2024-1	• 资源包购买流程及布局全面优化，新增资源包状态提示等。
2023-12	• BOS 支持图片处理持久化，可使用该功能对存量或增量图片进行处理，并将图片处理结果保存在 BOS 中，从而提升访问体验，可以满足数据处理时间较长、计算量较大的场景需求。详情参见 图片处理持久化 。 • 控制台批量处理支持根据 BOS 清单报告（Manifest）进行批量复制、批量取回归档文件。详情参见 批量处理 。 • BOS桌面客户端最新版本 0.2.2 发布，此版本支持后缀搜索并兼容 Windows Server 2012，并对一些问题进行了修复。可在 BOS-PC客户端 安装最新版本。 • BOS图像处理格式转换原图与目标格式支持 AVIF 格式。
2023-11	• STS临时授权访问 支持modify权限。 • BOS日志功能支持设置访问来源、访问类型，您可根据需求进行调整，日志内容更加灵活，详情参见 设置访问日志。 • BOS控制台支持归档文件取回后，单个/批量进行存储类型转换，详情参见 转换存储类型。 • 控制台上传功能全面优化，支持上传时为文件批量设置对象属性，新增上传任务列表实时查询文件状态，并能一键复制已上传文件链接、一键跳转到已上传文件存储桶中位置，详情参见 上传文件。 • BOS桌面客户端 0.2.1 发布，此版本提升了下载并发数限制，并对分享链接登陆后下载的相关问题进行修复。您可在 BOS-PC客户端 安装最新版本。
2023-10	• BOS控制台批量操作全面升级，支持批量转换存储类型、批量取回、批量删除、批量导出文件url，具体批量操作详情可查看 批量管理Object。同时新增任务列表能对不同的批量任务进度进行实时查看。 • BOS控制台支持在线体验文档预览功能，详情参见 文档预览 。
2023-9	• BOS桌面客户端 0.2.0 发布，此版本支持5GB以上文件及文件夹复制。您可在 BOS-PC客户端 安装最新版本。 • BOS桌面客户端 0.1.9 发布，此版本对下载及分享链接登录等进行优化，整体传输更稳定。您可在 BOS-PC客户端 安装最新版本。 • 控制台碎片管理优化，支持批量删除碎片文件，同时新增任务管理可对任务状态进行查看，详情参见 碎片管理。
2023-8	• BOS桌面客户端0.1.8发布，此版本支持大文件重命名并对一些问题进行了优化。您可在 BOS-PC客户端 安装。 • BOS CMD支持单个/批量进行归档文件取回，详情参见 取回文件，支持随机下载详情参见 sync支持随机下载 • Java SDK支持对存储桶标签进行管理，详情参见 Java SDK-Bucket标签管理 • ZIP多文件打包压缩发布，支持为存储桶添加多文件打包压缩规则，并将最终的压缩包投递至存储桶的指定路径下。详情参见 ZIP打包压缩。 • BOS桌面客户端版本0.1.7发布，此版本针对分享链接登陆的批量下载文件夹进行了优化。
	• 存储桶标签管理相关API发布，您可通过API对单个存储桶的标签进行设置、删除以及查看，详情参见 存储桶标签管理相关API 。

2023-7	• BOS桌面客户端0.1.6发布，使用分享链接登录后支持下载文件夹；此版本还对下载及上传进行了优化，下载时将默认选择上一次使用的下载地址。 • M3U8签名API发布，支持对m3u8文件中的ts资源进行批量下载授权，详情参见 M3U8 签名。 • BOS桌面客户端0.1.5发布，新增文件夹下载、文件夹批量下载功能，并对文件列表以及存储桶列表进行优化，支持根据文件名称、文件大小、更新时间、存储类型进行筛选排列文件，存储桶列表新增支持按照Bucket名称排列。 • 智能分层独立入口发布，您可通过一级导航栏进入智能分层管理页面，支持管理账户级别下所有已配置的智能分层规则，也可直接在智能分层管理页面进行配置新增操作，详情参见 智能分层。
2023-5	• 镜像回源设置API发布，您也可使用GO SDK进行设置，API详情参见镜像回源相关API，GO SDK 参见镜像回源设置。 • BOS桌面客户端最新版本0.1.4发布，您可在 BOS-PC客户端 安装。此版本提供控制并发任务数量的能力，从而进一步提升上传下载性能，详情参见 BOS-PC客户端使用指南。
2023-4	• 新版桌面工具发布：新版桌面工具功能更加丰富，支持获取存储桶详情，访问路径登录，批量文件（文件夹）上传与下载等功能，同时交互与界面风格全面升级改版，提供丰富功能与灵活操作的全新体验，详情参见新版BOS桌面工具。 • 软链接支持同地域存储桶：PutSymlink为BOS已有的文件（object）创建软链接（Symlink），该软链接支持在同地域存储桶中生成，详情参见 设置软链接（PutSymlink）。 • Select扫描 - SelectObject 接口 支持Parquet文件：SelectObject 接口在原有支持 CSV、JSON 格式基础上，新增 Parquet 格式，支持用户对 BOS 中 Parquet文件 执行 SQL 语句，详情参见 SelectObject 接口。
2023-3	• 服务端静态加密算法支持SM4：您可通过存储桶服务端加密功能选择SM4加密算法，为存储桶增量文件批量应用静态加密，详情参见服务端加密功能。您也可以通过上传文件时指定服务端静态加密算法为SM4，详情参见上传文件指定服务端加密 • 上传回调：对象存储BOS在完成文件（Object）上传时可以提供同步回调（Callback）给应用服务器。您只需要在发送给BOS的请求中携带相应的Callback参数，即可实现回调，详情参见上传回调 • 镜像回源全面升级： 1.支持设置备源地址，可提升回源成功率，保证用户服务高可用。 2.支持回源BOS私有bucket ，回源的目标文件是同账号下其他Bucket中的私有文件时，可通过回源私有Bucket功能正确获取到目标文件。 3.支持前后缀灵活配置，当BOS在向源站请求时，将文件名前缀的内容替换为指定前缀或裁剪，也可配置文件名为固定后缀。 4.重定向回源方式支持指定固定文件，详情参见设置回源
2022-12	• BOS桌面客户端支持文件路径复制与输入路径直接定位目标路径，方便用户进行文件管理 获取最新版本BOS桌面工具 • BOS CMD发布最新的0.3.5版本，包括： 进度条支持断点续传， 上传下载文件夹时进度条展示整个文件夹的传输进度及速度， 获取最新版本BOS CMD • BOS支持获取视频文件元信息，了解更多获取视频元信息 • 支持实时视频转码，BOS联合音视频处理服务（MCP）可实现实时转换视频码率，实现H264，H265两种码率的互转。了解更多视频实时转码
2022-11	• BOS桌面客户端支持文件夹分享，您可使用秘钥登录后，对文件夹获取分享链接，提升分享效率，立即使用BOS桌面客户端文件夹分享
2022-10	• 当您需要批量分享某一个存储桶（Bucket）中的多个文件（Object）时，可以创建文件夹并将所有待分享的文件存储在文件夹中，通过控制台文件夹分享功能生成分享链接。接收方通过浏览器访问分享链接或使用BOS桌面工具登录，即可下载文件夹中的内容。功能详情请查看文件夹分享 • 发布批量处理功能。您可以通过配置批量处理任务，实现数据批量复制和归档文件批量取回，简化操作提升处理效率。功能详情请查看批量处理

2022-09	• 新增3种图像处理资源包，包括基础图像处理包、图像高级压缩包、图像智能处理包。您通过预付费的方式购买图像处理资源包后，可以用于抵扣后续图像处理过程中产生的费用，详情请查看资源包概览 • 发布文档预览功能。您可以调用API完成文档到图片或文档到HTML的格式转换，从而在线预览文档，详情请查看文档预览 • 网页文件预览方式变更。自2022年9月15日起，BOS所有地域通过浏览器访问BOS官方域名与CDN官方加速域名中的网页类型文件将变更为以附件的形式进行下载。详情请查看BOS通过官方域名访问网页文件预览方式变更公告。如果您需要直接预览网页文件，可以通过创建自定义域名的方式实现，详情请查看创建Bucket自定义域名。 • 自定义域名发布支持验证域名归属。在创建Bucket的自定义域名时，您需要通过验证域名所有权来提升域名安全性。详情请查看创建Bucket自定义域名 • 控制台支持自定义文件访问权限。您可以在设置文件访问权限时选择自定义方式，自定义方式支持您为百度智能云用户授予只读和读写权限。详情请查看设置文件权限 • 控制台文件meta信息支持设置Expires。您可以查看或修改 Object 的缓存失效时间（Expires）。详情请查看设置meta信息
2022-08	• 控制台zip文件解压支持回调通知。您可以在配置文件解压规则时，填写用于接收通知结果的回调 URL 地址，BOS 会将文件解压的结果推送到该地址。详情请查看文件自动解压 • 重磅发布分层命名空间，分层命名空间是百度智能云推出的元数据管理功能。通过开启分层命名空间功能，可以很好的支持目录级别的mv与rename操作，同时优化常见读操作如：List、Head，提升数据处理效率与性能，能很好的满足大数据领域的数据湖计算场景。详情请查看分层命名空间
2022-07	• 全新发布智能分层，提供全自动的冷热分层机制。存储桶开启智能分层后，BOS 将周期性地监测数据访问频率，智能化地判断数据的访问热度，将数据进行自动地沉降或上升至适宜的访问层。智能分层能够有效帮助用户在存储成本和读写性能间寻找平衡点。详情请查看智能分层 • 全新发布单链接限速能力。您可以在调用 BOS 上传、下载相关 API 接口时自定义文件 URL 的访问限速，从而为整个存储桶进行流量控制，避免大流量服务造成带宽挤占。详情请查看单链接限速 • BOS 图像审核功能升级。您可以在 BOS 控制台直接开通内容审核 ICR 后付费服务，从而享受无限的审核调用次数。开通后，您的审核业务将不会再由于触发审核额度后而中断，保障风险内容在第一时间识别。详情请查看 图像审核
2022-06	• 控制台Bucket统计分析-资源用量支持查看 Bucket 使用带宽，包括外网上行、外网下行、内网上行和内网下行， 前往控制台查看使用带宽 • 华北-保定地域支持自定义域名功能绑定 https 证书，详情请查看绑定 https 证书 • fetch 异步抓取操作支持抓取结果回调通知。您可以在提交异步抓取任务时，填写用于接收通知结果的回调 URL 地址，BOS 会将异步抓取的结果推送到该地址。详细请查看 FetchObject 异步模式设置通知回调 • BOS 控制台支持对归档存储文件加急、标准和延缓 3 种取回方式，您可根据数据取回时间要求灵活选择，前往控制台体验 • 存储桶热点数据（包括热点对象，高频 IP，高频 Referer）支持导出为 CSV 文件，方便在本地进行数据分析。详情请查看热点分析数据导出 • VPC 级别访问控制功能上线华北-北京地域，并支持对百度智能云其他账号下的 VPCID 设置访问权限。详情请查看设置 Bucket 读写权限
2022-05	• 归档存储支持分级取回：加急取回、标准取回和延缓取回。加急取回可在 30 分钟内将文件取回，标准取回可在 5 小时内将文件取回，延缓取回可以在 12 小时内将文件取回。更多关于归档存储分级取回的信息，请查看存储类型介绍。关于归档存储分级取回的费用，请查看 BOS 计费文档中关于数据取回和归档文件取回请求次数的说明 • BOS 新推出视频截帧功能。您只需要设置 url 参数，即可对存储在 BOS 中的视频文件进行自动抽帧操作。同时，您可在控制台资源仪表盘查看本月视频截帧用量数据。详情请查看视频截帧 • BOS 命令行工具 CMD 支持文件上传时设置文件标准 Header 与自定义 Meta 信息，方便快速完成文件的上传和管理。详情请查看 CMD 工具文件上传
	• 图片缩放支持设置“背景拉伸”。缩放背景除已有的颜色填充能力外，本次新增图片填充能力，并支持对背景填充图片进行高斯模糊和按比例拉伸。您只需在url中设置相关参数，即可自动化实现缩放+背景填充功能。详情请查看图像缩放

2022-04	• 图片水印支持水印图和背景图等比例缩放。您只需在url中设置缩放比例，即可实现等比例缩放效果。详情请查看添加图像水印 • 支持上传解压功能。该功能基于百度智能云函数计算CFC，您只需在BOS控制台提前为相应存储桶添加文件解压函数规则，上传后的zip压缩文件就会自动解压到指定的存储桶和路径中。当用户待上传的文件数量很多时，可以提前在本地将文件打包压缩，配合BOS上传解压功能即可快速完成大量文件的上传。详情请查看设置文件解压
2022-03	• 控制台交互全新升级，您可以前往控制台体验 • 控制台发布操作引导页与资源仪表盘。操作引导页可以帮您快速了解和使用产品，资源仪表盘帮助您能更加直观的掌握资源使用、存储桶配置和分布等信息，查看服务引导与资源仪表盘
2022-02	• 存储桶日志增加访问者 Ak、镜像回源、CDN 回源和 COPY 源类型字段标识，您可以在日志文件中查看相应访问记录 • 存储桶生成的日志文件支持投递至源桶，并支持多个存储桶日志的集中投递，详细功能参考设置访问日志
2021-12	• 控制台支持文件批量导出文件 URL，您可以前往控制台查看并体验 • 控制台支持批量恢复回收站中文件，详情请查看设置和使用回收站
2021-11	• 图片处理新增缩放背景模糊，设置水印图比例、图片拼图功能 • 资源包支持批量购买且支持加入购物车，便于多类型多规格资源包批量购买，购买资源包
2021-10	• C++ SDK 支持 Bucket/Object ACL • 兼容 S3 GetBucketLocation接口 • 控制台图片处理样式新建与管理功能升级，前往新建样式
2021-09	• BOS 支持在创建 Bucket 时根据您的业务线来绑定不同资源分组。详情请查看创建 Bucket • 存储桶清单功能在华北-北京、华东-苏州和华南-广州全量开放，并支持生成 Manifest 文件报告 • 华东-苏州地域开放售卖标准存储-多AZ和低频存储-多AZ。不同地域的售卖情况请查看地域支持 • BOS FS 工具发布新版本，支持通过内存缓存方式进行挂载后的速写访问，读写性能明显提升，欢迎试用
2021-08	• 单文件上传支持最大 48.8 TB • 分片上传修改限制，分片的最小大小由原有的 1MB 调低到 100KB • 兼容 S3 Options 请求 • 华北-保定区域开放售卖归档存储类型 • PostObject 支持 STS 方式鉴权

	• BOS 新增 18 项监控项，涵盖请求次数，Http Code 统计等
2021-07	• 标准存储、低频存储、冷存储上线1PB、3PB、5PB、10PB大容量资源包
2021-05	• BOS 新增 5 项监控项，涵盖Http Code 2XX、3XX、4XX、5XX 统计等
2021-03	• 全新发布数据流转平台 CloudFlow，支持数据从三方云迁移到百度智能云 BOS。数据流转平台广泛兼容现有的第三方云平台对象存储服务，可通过控制台简单配置启动数据迁移，并可对数据迁移进度进行查看 • BOS 控制台全局概览页交互全新升级，并接入云顾问监控，对云上问题与风险进行系统性检查与报告
2021-02	• BOS 图像处理全新发布图像锐化、图像模糊、图像亮度功能 • BOS 图像处理全新发布图文混合水印功能，支持添加图片和文字组合水印 • BOS 数据同步支持在目的端设置保存路径，将文件保存在目的端的非根目录下
2021-01	• BOS 图像处理新增样式批量导入导出，支持将样式在本地保存，或与其他账号共享 • BOS 桌面支持新建 Bucket 操作，并对交互进行优化
2020-12	• BOS 接入 APIExplorer，支持 BOS 近百个 API 一键生成调用 • BOS 控制台交互全新改版，增加全局导航栏，并优化交互细节，体验更加流畅 • BOS 存储包支持 2 年/ 3 年时长购买，价格更优惠。 • BOS 全新发布智能图像处理，支持"智能场景"和"智能特效" 10 余种 AI 图像处理能力，一键生成图像处理效果 • BOS 兼容 S3 ListObjectsV2 接口 • 新增基础图像处理次数监控，您可以在云监控 BCM 中查看监控次数
	• BOS 全新发布多 AZ 存储类型，支持创建标准存储-多AZ和低频存储-多AZ类型文件，实现冗余高可靠存储 • BOS 发布低频存储包和冷存储包，支持对低频存储和冷存储文件存储容量进行预付费形式购买

2020-11	• BOS 支持文件夹统计，对文件夹内的文件数量及容量进行统计 • BOS 支持在控制台批量删除文件夹，支持对文件夹进行批量删除 • BOS 支持为请求者付费功能。开启后，Bucket 内数据的流量和请求费用将由请求者承担 • 华东-苏州区域 BOS 支持为存储空间设置清单，定期扫描您 Bucket 中指定对象，并输出一份清单报告。该报告会以 CSV 格式文件的形式存储到您指定的 Bucket 中。华北-北京和华南-广州区域白名单开放，请您先提交工单
2020-10	• BOS Util 工具支持 批量设置 ACL • BOS 支持外网下行流量多月包，支持购买 3 个月、半年、一年时长规格的流量包 • 华北-保定区域支持 VPC 级别权限设置，允许或禁止某个 VPC 访问您 Bucket 中的数据 • BOS 调整归档存储文件计费规则。若归档存储类型文件不足 64 KB，则按照 64 KB 大小进行计费；若文件大于等于 64 KB，则按照实际大小进行计费
2020-09	• 图像处理支持图片灰化，支持将彩色图片变为黑白图片 • 控制台支持对文件设置软链接，用于快速访问对象存储空间内的常用文件
2020-08	• 华东-苏州地域支持创建基于访问时间规则的生命周期规则 • 华东-上海地域全量开放，价格与普通地域价格保持一致 • BOS 新增图像处理相关数据监控项，支持对图像基础处理用量、图像压缩调用次数进行查看。您可以在 BOS 全局概览页和云监控 BCM 控制台查看监控数据
2020-07	• BOS 新增软链接功能，用于快速访问对象存储空间内的常用文件 • BOS 支持在全局概览页查看和导出用量数据。数据指标涵盖单个 Bucket 总容量，总流量和总请求次数，并支持对汇总数据和细粒度数据一键下载 • BOS 的 Bucket 概览页交互全新升级，支持以饼状图形式直观展示 Bucket 详细数据，并新增 Bucket 汇总数据展示 • BOS 发布 Alluxio Extension 工具，能够为存储系统（如 BOS、HDFS、S3 等）上层驱动型应用（如 Apache Spark 等）提供统一的客户端 API 和全局命名空间 • BOS 新增跨账号同步功能，支持跨账号同步 Bucket 存储数据。您可以通过控制台、API 和 SDK 进行操作 • BOS 新增多规则同步功能，支持一个源 Bucket 可以同时同步数据到多个目的 Bucket，一个目的 Bucket 可以同时支持多个源 Bucket 同步数据 • BOS 控制台新增 Select 扫描功能，支持对 CSV 和 JSON 文件在控制台进行数据检索和查询 • 支持在控制台配置事件通知规则 ID 和产品 ID

2020-06	• BOS 新增额度管理功能，支持自定义设置 Bucket 总数和存储容量 • 图像处理样式额度提升至每个 Bucket 400 个 • 自定义域名数量提升至每个 Bucket 200 个 • BOS 控制台配置页交互全新升级
2020-04	• BOS 新增 19 项监控项，并升级监控页面 • BOS 智能业务分析功能全量开放，助力成本优化
2020-03	• BOS 支持通过 CDN 进行上传加速，以提升大文件上行场景中的速度需求 • BOS 权限控制增加细粒度权限 GetObjectMeta，满足细粒度权限设置需求 • BOS Select 扫描功能升级，支持更多文件类型
2020-02	• 华北-北京区域支持归档存储，控制台购买通道已打开
2020-01	• 支持 Bucket 合规保留功能。开启后，可以将 Bucket 内文件保留 1 天 - 70 年，以满足金融、保险、医疗、证券等行业的数据保留政策要求 • BOS 控制台支持对事件通知进行加密鉴权 • 图像处理新版协议上线，并新增自适应旋转、内切圆等新处理方式
2019-12	• 生命周期管理模块中，下沉低频存储、冷存储的最低存储时间限制由 7 天调整为 1 天 • 支持子用户创建 Bucket 时关联标签 • 生命周期管理模块中，支持基于文件访问时间记录设置类型下沉或者自动删除 • 冷存储最低存储时间由 90 天调整为 60 天 • BOS 桌面客户端支持"本地同步"功能。您可以在 BOS 桌面客户端中进行设置，将本地文件自动定期同步到 BOS • BOS HDFS 工具全新发布，支持本地自建 Hadoop 集群高效访问 BOS

2019-08	• BOS 提供了业内领先的四级分级存储。用户可以选择标准存储作为主要存储方式，也可以根据不同访问频率选择成本更低、存储时间更长的的低频存储、冷存储和归档存储 • BOS提供标准存储包、外网下行流量包和回源流量包共 3 种资源包用于抵扣流量使用中产生的费用 • 按照北京/广州/苏州/保定、香港、武汉金融和新加坡不同区域，BOS 计费标准不一样
2019-07	• BOS 存储包降价 • Python SDK 工具包可在 Python 2.7 和 Python 3 环境下运行
2019-04	• 新增智能业务分析功能，智能业务分析功能基于用户对数据使用情况的深度分析，科学预测出后续客户业务对数据的持续使用频次，从而给出用户最省钱的生命周期配置策略 • 新增服务器加密方式，BOS 支持 BOS 托管密钥和 KMS 服务托管密钥两种加密方式 • 发布月光宝盒工具，月光宝盒是百度智能云提供的PB级离线数据迁移服务，支持将本地 IDC 中海量数据通过离线设备拷贝的方式迁移上云
2019-02	百度智能云 BOS 提供了业界全新的音视频处理接口，支持上传 mp4 格式的文件，系统自动保存为 hls 格式，用于 m3u8 风格播放器读取使用。充分满足了“ UGC 上传——自动化处理——播放”这条使用流，免去了原本复杂的视频操作处理逻辑，并为业务提速
2018-12	BOS迁移工具BosImport版本 1.0.0 首次发布，支持多种迁移方式，支持迁移不同的源端到目的端，目前支持 AWS S3、阿里云 OOS、腾讯云 COS、七牛、小米生态云、BOS等
2018-11	• 支持对 GIF 格式图片进行裁剪、旋转等操作 • 上线 BOS 回收站功能，支持对 Bucket 内删除的数据进行保留，降低数据被误删或恶意删除的风险 • 新增 BOSFTP 工具，支持将 FTP 协议的操作转化为对 BOS 中存储的对象、目录的操作。支持文件上传、下载、删除以及创建文件夹等操作
2018-09	• 支持事件通知功能。BOS 事件通知功能是基于 BOS 数据处理框架 V2.0 生成的一套智能触发平台。开发者可基于平台提供的功能，定义业务关注的Bucket中发生的数据行为，BOS 会监测 Bucket 中发生的 Put Object、Append Object 等行为，向用户的下游业务系统提供触发通知 • 基于 BOS 事件通知平台，上线了通用文字识别官方应用。系统可自动对用户上传的图片进行文字内容智能识别 • 基于 BOS 事件通知平台，上线了通用物体识别官方应用。系统可自动对用户上传的图片进行物体智能识别 • BOS 对外发布了官方 C++ SDK
	• 百度智能云 BOS 大幅降价，存储费用、流量费用、API 请求费用降至全网最低 • Java SDK 新增功能

2018-08	• 支持设置 Object 的访问权限 • 支持删除文件 • 支持 Object 分块拷贝 • 支持设置生命周期管理 • 支持跨域访问 • 支持设置访问日志

产品公告

【2024升级】

官方默认域名及加速域名apk、ipa 类型文件不支持下载

对象存储（BOS）为确保整体服务的安全与稳定，我们将于北京时间2024年11月25日00:00:00起实施以下服务调整。请仔细阅读以下升级详情，并适时调整您的应用配置以确保业务连续性。

自上述日期起，新创建的Bucket若尝试通过BOS默认官方访问地址（如：bucketname.bj.bcebos.com）访问 apk 和 ipa 文件，采用URL签名验证或者匿名访问，请求将被阻断，并返回错误码DownloadForbidden，HTTP状态码400。

对于所有Bucket，若在上述日期后启用cdn官方加速传输功能，并尝试通过加速域名（如：bucketname.cdn.bcebos.com ）访问apk 和 ipa文件，同样会遇到上述访问限制，错误代码同为DownloadForbidden。

若您希望下载存储桶内的 apk、ipa 类型文件，请使用自定义域名访问，绑定自定义域名详细步骤请参见https://cloud.baidu.com/doc/BOS/s/ckaqihkra。

本次升级给您造成的不便敬请谅解，如有任何问题，您可以通过工单向我们反馈和咨询。

感谢您一直以来对百度智能云的支持与信任！

BOS 存储桶域名请求风格更新

为保证服务的安全和稳定，2024年6月1日起，百度智能云对象存储 BOS (Baidu Object Storage) 对于官方域名的请求风格将作出变更。新创建的存储桶，不支持使用 path-style 的请求风格，默认支持 virtual-hosted 的请求风格。

2024年6月1日之前所创建的存储桶，使用官方域名时，支持继续使用 path-style 的请求风格。对于存量的使用 path-style 请求风格的存储桶，建议可切换至 virtual-hosted 的请求风格。切换过程如有疑问，可参考开发者指南。如需我们的帮助，请与我们联系。

安全与合规

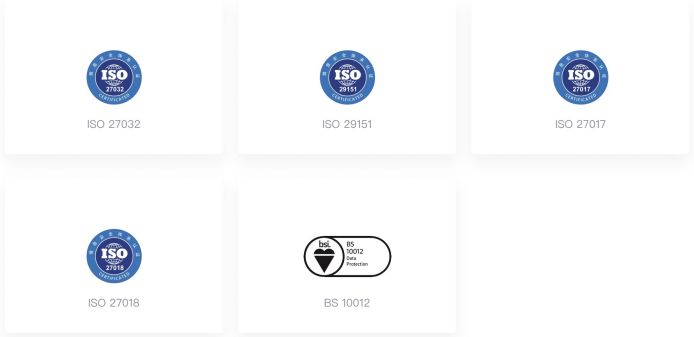
概述

对象存储 BOS（Baidu object Storage）具有丰富的安全防护能力，支持服务器端加密、客户端加密、防盗链白名单、便捷预置与细粒度权限管控、日志审计、合规保留策略（WORM），数据冗余存储到多个可用区（Availability Zone 简称AZ），跨地域备份容灾等特性。BOS 为您的云端数据安全进行全方位的保驾护航，并满足您企业数据的安全与合规要求。

云服务商首家

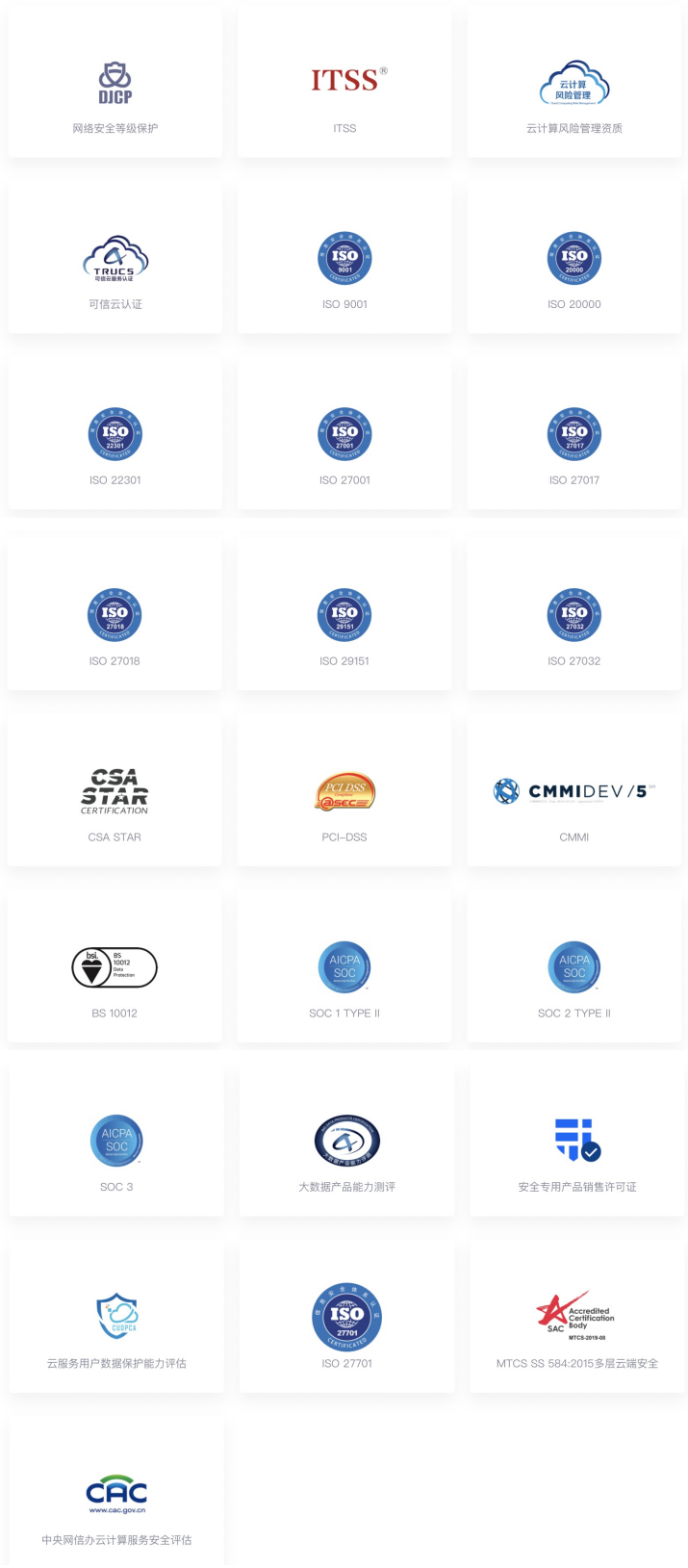
百度智能云始终将合规思路融入到云平台框架体系搭建及云产品服务功能设计之中，立志为客户打造安全可依赖的云生态。首次获得：

- **ISO 27032**：是国际通用的网络空间安全指南，旨在指导控制关键信息基础设施保护必须承担的四大领域的风险，提供控制这些风险的参考措施。百度智能云是国内第一家获得此认证的云服务商;
- **ISO 29151**：是国际通用的个人身份信息保护指南，旨在充分控制风险，确保个人身份信息(PII)生命周期的安全。百度智能云是国内第一家获得此认证的云服务商；
- **ISO 27017**：是专门针对云计算服务的信息安全控制措施实用标准。百度智能云是国内第一家获得此认证的云服务商；
- **ISO 27018**：是第一个要求并指导公有云服务供应商保护云用户个人信息安全的国际标准。百度智能云是国内第一家获得此认证的云服务商；
- **BS 10012**：是世界首部个人信息保护管理体系标准。依据GDPR要求修订的BS 10012，要求企业满足个人信息管理体系规范的同时提供落地的工具。百度智能云是国内首家通过BS 10012认证的云服务商



权威资质概览

对象存储 BOS 连续 5年通过可信云认证，是国内首批通过中国信通院对象存储标准评测的产品。此外，BOS 还获得包括中国开源云联盟颁发的云计算卓越产品奖在内的众多奖项。对象存储 BOS 已经获得如下认证：



BOS安全性能力建设

对象存储 BOS 还从产品能力上提供访问控制、数据加密、监控审计、数据容灾、数据合规保留、数据安全性等众多产品能力，您可以从多个方面对数据进行合规性和安全性存储。

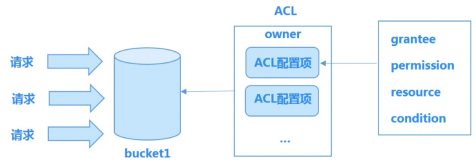
安全能力	说明
访问控制	BOS提供读写权限ACL、授权策略、防盗链白名单等功能，实现存储资源访问的控制和管理。
数据加密	BOS提供服务器端加密和客户端加密，并支持基于SSL/TLS的HTTPS加密传输，有效防止数据在云端的潜在安全风险。
监控审计	BOS提供访问日志的存储和查询功能，可满足您对企业数据的监控审计需求。
数据容灾	BOS提供同城冗余存储和跨区域复制特性，实现同地域和跨地域级别的机房容灾能力。
数据保留合规	BOS支持WORM（Write once read many）特性，允许用户以"不可删除、不可篡改"方式保存和使用数据，满足金融，保险，医疗等行业的合规要求。
数据保护	BOS提供版本控制与回收站功能，开启版本控制功能后，BOS将数据覆盖和删除操作通过历史版本的形式保存下来。在错误覆盖或者删除对象后，能够将 Bucket 中存储的数据恢复至任意时刻的历史版本，可有效的防止数据被篡改和删除；回收站功能可防止数据的误删除，开启回收站后，如您的BOS Bucket中数据被误删除，您还可以通过回收站进行数据恢复。

访问控制

百度智能云对象存储BOS提供读写权限ACL、粗细粒度的自定义授权策略、防盗链白名单等功能，实现存储资源访问的控制和管理。

读写权限

BOS为权限控制提供访问控制列表（ACL）。ACL是授予Bucket和Object访问权限的访问策略。 您可以在创建存储空间（Bucket）或上传对象（Object）时配置ACL，也可以在创建Bucket或上传Object后的任意时间内修改ACL。



- **业界最细粒度权限管理：**
- 支持设置允许或禁止的控制效果
- 支持标准权限、粗粒度、细粒度权限设置，满足不同场景下的控制需求
- 支持对IP、用户ID、VPC、Referer、Https进行API粒度的权限控制

Bucket ACL

Bucket ACL是Bucket级别的权限访问控制。目前有三种访问权限，含义如下：

- Bucket 标准权限：

业界常见的权限设置，分为私有、公共读和公共读写。

权限名称	权限说明
私有（private）	Bucket 创建者独享该 Bucket 的全部权限，其他人不能对该Bucket 写入数据，也不可访问读取该 Bucket。 适用于存储私密文件的场景。
公共读(public-read)	Bucket 创建者独享该 Bucket 的全部权限，其他人可以访问读取该 Bucket 内的内容，但无权写入。 适用于存储需要公众读取的文件的场景，如通知性文件、广告内容等。 **请注意**：若开启该种权限，互联网上的所有用户都可以对该 Bucket 内的文件进行访问读取，请谨慎开启。
公共读写 (public-read-write)	包括 Bucket 创建者在内的任何人都可以访问和写入文件。 **请注意**：若开启该种权限，互联网上的所有用户都可以对该 Bucket 内的文件进行读写操作，请谨慎开启。

但以上简单标准权限无法满足您自定义权限管理的需求时，您可以使用以下自定义权限管理

- 粗粒度自定义权限： 较标准权限更细粒度的权限，可对指定用户设置 READ、LIST、WRITE、FULL_CONTROL 和 MODIFY 权限，并可以指定该权限可访问的资源，以及指定具有该权限的 IP 地址和 Referer 白名单等。更多信息，请参见权限介绍-粗粒度自定义权限定义。
- 细粒度自定义权限： BOS 提供的 API 级别细粒度自定义权限。可设置的 API 类别包括 GetBucket、GetObject、PutObject、DeleteObject 在内的共 18 种。您可以对每种类别自定义相应的访问权限。 更多信息，请参见权限介绍-细粒度自定义权限定义。

Object ACL

Object ACL是Object级别的权限访问控制。目前有2种访问权限，含义如下：

权限名称	权限说明
私有（private）	只有该Object的Owner拥有该Object的读写权限，其他的用户没有权限操作该Object。
公共读(public-read)	所有用户只有该Object的读权限，而Object Owner拥有该Object的读写权限。

细节说明：

- 如果 Object 没有设置权限，则 Object ACL 为空，默认以 Bucket 权限为准。
- Object ACL 优先级高于 Bucket ACL，例如用户设置了 Object 的权限是 public-read，无论 Bucket 是什么权限，该 Object 都可以被访问。
- 目前 Object ACL 鉴权只对 GetObjectMeta、GetObject、CopyObject、UploadPartCopy 四个接口生效。

更多信息，请参见Object权限控制。

IAM主子用户访问控制

在您的百度智能云账号下面，通过 IAM 可以创建具有自己 AccessKey 的子用户。您的百度智能云账号被称为主账号，创建出来的账号被称为子用户，使用子用户的 AccessKey 只能使用主账号授权的操作和资源，BOS已经接入IAM子用户系统，通过控制台多用户访问控制创建子用户，主用户通过关联策略实现授权给子用户，策略文件本质上是一个JSON文件。策略语句中permission和resource用来定义权限和资源，更多细节，请参考IAM主子用户访问控制。

STS临时授权

BOS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。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访问百度智能云资源。更多信息，请参见临时授权访问。



防盗链白名单

对象存储BOS是按使用量收费的服务。为了减少您存储于BOS的数据被其他人盗链而产生额外费用，BOS支持设置基于HTTP和HTTPS header中表头字段Referer的防盗链方法。

您可以设置防盗链白名单，仅允许指定的域名访问BOS资源，或者仅允许HTTP或HTTPS header中包含Referer字段的请求才能访问BOS资源。对于公共读或公共读写的Bucket，防盗链白名单可以有效防止盗链，保护您的合法权益。详情请参见设置Referer白名单。

数据加密

对象存储BOS提供服务器端加密和客户端加密，并支持基于SSL/TLS的HTTPS加密传输，有效防止数据在云端的潜在安全风险。

服务器端加密

BOS支持在服务器端对上传的数据进行加密（Server-Side Encryption）。上传数据时，BOS对收到的用户数据进行加密，然后再将得到的加密数据持久化保存下来；下载数据时，BOS自动对保存的加密数据进行解密

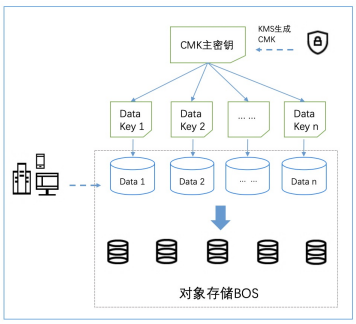
并把原始数据返回给用户，并在返回的HTTP请求Header中，声明该数据进行了服务器端加密。

BOS通过服务器端加密机制，提供静态数据保护。适合于对于文件存储有高安全性或者合规性要求的应用场景。例如，深度学习样本文件的存储、在线协作类文档数据的存储。针对不同的应用场景，BOS有以下两种服务器端加密方式：BOS 托管密钥和 KMS 服务托管密钥。

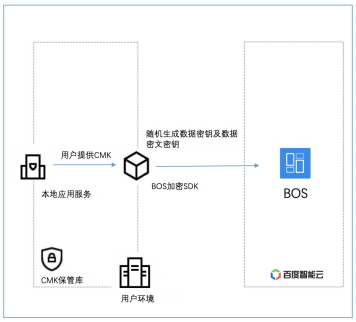
- BOS 托管密钥：当选择BOS托管密钥时，所有上传该Bucket的数据都会使用BOS服务托管的密钥进行统一加密保护，用户获取数据时将由BOS服务进行自动解密，用户不需做其他操作，简单方便。
- KMS 服务托管密钥：您需要开通百度智能云密钥管理服务，并创建由自己管理的 KMS 密钥。KMS 中会为根据您创建的 KMS 主密钥生成对应的 Data Key，数据上传时会根据生成的 Data Key 来对数据进行自动加密。

当文件上传时，您可配置任意一种加密方式，BOS 会通过携带加密参数的方式来对上传数据执行服务端加密设置。同时，为了提升易用性，BOS 在 Bucket维度支持 设置服务端加密开关，开启存储桶加密之后该 Bucket 内新上传的数据都会默认执行加密。

了解更多信息请参见BOS操作指南中的服务器端加密。



服务端加密



客户端加密

客户端加密

客户端加密是指将文件（Object）发送到对象存储BOS之前在本地进行加密。使用客户端加密功能时，百度智能云对象存储不参与加密和解密过程，只负责文件的上传、存储、下载过程，明文密钥由用户自行保管在本地。客户端加密增强了文件安全性，即使文件意外泄露，别人也无法解密得到原始数据。但是由于用户需要自行负责保管密钥，如果明文密钥丢失，用户将无法获得原文件内容，您需要对主密钥的完整性和正确性负责。

使用客户端加密时，会为每个Object生成一个随机数据加密密钥，用该随机数据加密密钥明文对Object的数据进行对称加密。主密钥用于生成随机的数据加密密钥，加密后的内容会当作Object的元信息保存在服务端。解密时先用主密钥将加密后的随机密钥解密出来，再用解密出来的随机数据加密密钥明文解密Object的数据。主密钥只参与客户端本地计算，不会在网络上进行传输或保存在服务端，以保证主密钥的数据安全。我们为您提供了客户端加密典型实践供您参考。

基于SSL/TLS的HTTPS加密传输

BOS支持通过HTTP或HTTPS的方式访问。您也可以在Bucket细粒度访问控制中设置仅允许通过HTTPS（TLS）来访问BOS资源，实现更加安全的数据传输。安全传输层协议（TLS）用于在两个通信应用程序之间提供保密性和数据完整性。了解更多，详情请参见HTTPS传输加密实践。

监控审计

对象存储BOS提供访问日志的存储和查询功能，并支持Bucket操作日志透明化，满足您对企业数据的监控审计需求。

日志转存

访问对象存储BOS的过程中会产生大量的访问日志，您可以通过日志转存功能将这些日志按照固定命名规则，以小时为单位生成日志文件写入您指定的Bucket。对于已存储的日志，您可以通过百度智能云日志服务或搭建Spark集群等方式进行分析。更多信息，请参见设置访问日志。

实时日志查询分析

通过与日志服务 (BLS) 相结合，您可对BOS实时日志进行查询。您可以在BLS控制台完成BOS访问的操作审计、访问统计、异常事件回溯和问题定位等工作。实时日志查询功能能够有效提升您的工作效率，并帮助您基于数据进行决策。更多信息，请参见日志服务。

操作日志审计

百度智能云云审计（BCT，Baidu CloudTrail）提供平台操作日志近实时投递到日志服务，进行相关分析审计服务。CloudTrail可以近实时地记录并存储百度智能云BOS平台操作日志，对云账户进行监管、合规性检查、操作审核和风险审核。借助 CloudTrail，可以记录日志、持续监控并保留与整个云基础设施中的操作相关的账户活动。更多信息，请参见云审计。

CloudTrail 提供通过云管理控制台执行的操作记录查询。这一事件历史记录可以简化安全性分析、资源更改跟踪和故障排除工作。更多信息，请参见平台操作日志简介。

监控服务

BOS深度对接云监控BCM（Baidu Cloud Monitor）为您提供系统基本运行状态、性能以及计量等方面的监控数据指标，并且提供自定义报警服务，帮助您跟踪请求、分析使用情况、统计业务趋势，及时发现以及诊断系统的相关问题。更多信息，请参见云监控操作指南。

图像审核

您在BOS上存储的数据可能包括一些不良内容，例如色情、暴恐、政治敏感、广告等。如果您希望更好的针对非法不合规数据进行识别检测和审核处理，可以通过图片审核服务，对您存储在BOS中的图片进行自动的检测和审核， 判断指定bucket中的图片是否包含色情、暴恐、政治敏感、广告、恶心不适等不良内容，并给出置信度分值。 同时支持智能识别图像质量，包括清晰度和美观度，帮助过滤低质量图片。审核维度支持自定义和灵活组合，模型准确度高、响应速度快，并且可以不断学习提高准确度，可大幅节约审核人力，提升处理效率。更多信息，请参见图片审核服务。

数据容灾

对象存储BOS供多AZ存储（同城冗余存储），以及跨区域与同地域复制特性，实现同地域和跨地域级别的机房容灾能力。

多AZ存储

用户开启多AZ存储后，BOS将用户的数据分散存放在同一地域（Region）的多个可用区。当某个可用区不可用时，仍然能够保障数据的正常访问。BOS多AZ存储提供99.9999999999%（12个9）的数据设计持久性以及99.99%的服务可用性。

BOS的多AZ存储存储能够提供机房级容灾能力。当断网、断电或者发生灾难事件导致某个机房不可用时，仍然能够确保继续提供强一致性的服务能力，整个故障切换过程用户无感知，业务不中断、数据不丢失，可以满足关键业务系统对于“恢复点目标（RPO）”，“恢复时间目标（RTO）”等于0的强需求。

目前BOS的多AZ存储存储支持标准存储类型、低频访问存储类型。这两种存储类型的各项对比指标详情如下：

对比指标	标准存储 -多AZ	标准存储	低频存储 -多AZ	低频存储
可用性	99.995%	99.99%	99.995%	99.95%
最低存储时间	无	无	30天	30 天
数据取回费用	无	无	按量收费（GB）	按量收费（GB）
数据访问特点	实时访问，毫秒延迟。	实时访问，毫秒延迟。	实时访问，毫秒延迟。	实时访问，毫秒延迟。
适用场景	移动应用、网站视频、数据分析等需极高可靠性的场景。	移动应用、网站视频、数据分析等。	网盘存储、监控数据、媒资备份等需极高可靠性的场景。	网盘存储、监控数据、媒资备份等。
场景特点	数据频繁读取场景。	数据频繁读取场景。	月均访问次数小于 1 次场景。	月均访问次数小于 1 次场景。

功能详情请参见BOS开发指南中的多AZ存储存储。

数据同步

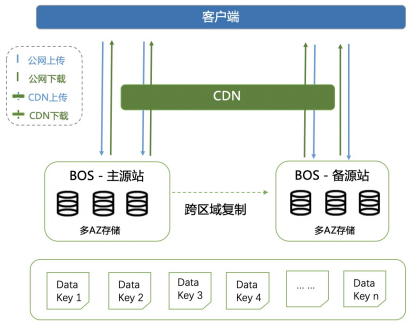
数据同步(Replication) 是 BOS 数据中心 (Region) 的存储空间 (Bucket) 之间自动、异步复制文件 (Object) 的功能，它会将 Object 的创建、更新和删除等操作从源存储空间复制到目标存储空间。

数据同步功能能够很好的提供Bucket同地域、跨区域、跨账号或其他数据复制的需求。目标 Bucket 中的对象是源 Bucket 中对象的精确副本，它们具有相同的对象名、元数据以及内容，例如创建时间、拥有者、用户定义的元数据、Object ACL、对象内容等。

使用场景

- 高效访问，减少延迟： 跨 Region 访问 Bucket 或者 Object，访问速度会因地理上的差异而降低。通过数据同步功能可提前将需要的数据同步至目标 region，提高访问效率。
- 跨地域容灾： 基于金融、政府等行业的合规性要求，在同一 Region 中虽然有数据多副本设置。但为了防止洪水、地震等机房级别灾害，也要求数据跨 Region 保存一份副本，BOS 的数据同步功能提供了这样的能力。
- 跨账号复制： 适用于企业的多账号场景。处于数据安全和多备份考虑，企业希望将数据从 A 账号定期同步到 B 账号，以实现数据的跨账号复制和共享。
- 跨区域数据复用： 由于业务原因例如跨 Region 搭建的计算集群需要使用同一组数据等场景，可以使用 BOS 数据同步功能复制数据副本。
- 数据复制： 由于业务原因，需要将数据从BOS的一个数据中心复制到另一个数据中心。

功能详情请参见BOS开发指南中的数据同步。



- **物理机级别容灾**
  BOS采用分布式存储，实现数据的物理机级别容灾
- **机房级别容灾**
  通过多 AZ 存储类型实现数据在 3 个机房的容灾
- **跨地域容灾：**
  将某个地域下的数据跨地域同步到另一个地域，实现数据的跨地域容灾
- **镜像回源：**
  CDN配置主备回源地址，主bucket会把数据在最长一分钟内同步到备份bucket，主源站故障后会自动切到源站

数据保留合规

BOS支持WORM（Write once read many）特性，允许用户以“不可删除、不可篡改”方式保存和使用数据，数据更加安全，可满足数据合规存储的需求。

BOS提供强合规策略，您可以针对Bucket设置基于时间的合规保留策略。当策略锁定后，用户可以在Bucket中上传和读取Object，但是在Object的保留时间到期之前，任何用户都无法删除或修改Object和策略。Object的保留时间到期后，才可以删除与修改Object。BOS支持的WORM特性适用于金融、保险、证券、医疗等行业，可满足数据安全与合规场景。

了解更多功能详情，请参见BOS控制台操作指南中的设置合规保留。

数据保护

数据保护

当开发者在 BOS 进行数据存储和管理时，基于更高的数据可靠性要求，需要防止数据被误删除，可开启BOS的存储桶（Bucket）版本控制或回收站功能。当数据数据被误删除时，版本控制与回收站可支持您对误删除数据进行恢复，此外，版本控制功能还可基于文件的版本管理机制，将被误覆盖的数据进行恢复，为实现全面的数据保护，开启版本控制功能是最佳选择。

版本控制

版本控制是针对存储空间（Bucket）级别的数据保护功能。开启版本控制后，针对数据的覆盖和删除操作将会以历史版本的形式保存下来。您在错误覆盖或删除对象（Object）后，能够将存储在Bucket中的Object恢复至任意时刻的历史版本。

应用场景

- 数据恢复： 当使用数据时产生误删与误覆盖的情况，希望有途径将数据恢复。
- 数据保护： 当一个 Bucket 通过 访问控制授权给多个用户，并给予他们 Write 权限时，希望能够避免这些用户对 Bucket 中的数据进行恶意删除操作。
- 防止文件被覆盖： 对于在线协作类产品，文件会被频繁修改，针对文件的编辑会产生大量的临时版本。您可以使用版本控制功能找回某个时间点的版本。

了解更多版本控制功能细节，请参考开发者指南-版本控制概述。

回收站

回收站是针对存储空间（Bucket）级别的数据保护功能，开启回收站后，数据删除后会被放入回收站中，可对回收站内数据进行恢复或彻底删除，

应用场景

- 数据恢复： 当使用数据时产生误删情况，希望有途径将数据找回。
- 数据保护： 当一个 Bucket 通过 ACL 授权给多个用户，并给予他们 Write 权限时，希望能够避免这些用户对 Bucket 中的数据进行恶意删除操作。

了解更多回收站功能细节，请参考开发者指南-设置版本控制。

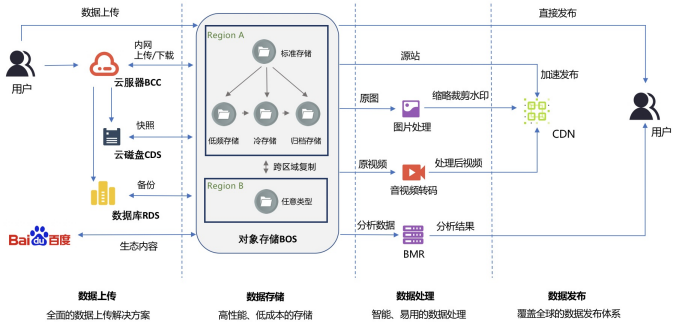
产品描述

产品简介

概述

百度对象存储 BOS (Baidu Object Storage) 提供稳定、安全、高效以及高扩展存储服务，支持单文件最大 48.8 TB 的文本、多媒体、二进制等任何类型的数据存储。

数据多地域跨集群的存储，以实现资源统一利用，降低使用难度，提高工作效率。"存储+计算框架"让您的数据加上"动力"引擎，让数据在传输、存储、处理和发布 4 个环节有机融为一体。



应用场景

BOS 适用于数据分发、数据备份、大数据分析、音视频文件处理等各类场景，详情查看应用场景。

分级存储

不同类型数据的使用方式、访问频率均有不同，同时数据也存在老化曲线。随着数据逐渐老化，其访问频率逐步降低，进入归档状态。分级存储是一种方法论，即将不同类别数据存放到不同存储类型，以降低总存储成本，同时满足服务要求。

BOS 针对不同存储场景，提供了标准存储-多AZ、标准存储、低频存储-多AZ、低频存储、冷存储和归档存储六种存储类型。 您可以在上传文件时指定 Object 的存储类型，此时 Object 的存储类型以上传时指定的为准。此外，您还可以设置 Bucket 的默认存储类型，如果上传文件时未指定存储类型，则文件继承 Bucket 的默认存储类型。关于各类存储类型的详细介绍，请查看分级存储。

计费项

BOS 的计费项包括存储费用、流量费用、请求费用、数据处理费用共 4 部分，且每一项根据您的使用情况，会产生不同的计费价格。详情请查看计费项概览。

管理 BOS

- 通过控制台管理 BOS

  您可以登录 BOS 管理控制台进行 Bucket 管理、Object 管理、数据发布、数据处理、事件通知、监控报警等操作，详情请参见控制台操作指南。

- 通过 API 或 SDK 管理 BOS

  BOS 提供 API 和各种语言的 SDK 开发包，方便您快速进行二次开发。详情请参见 API 参考和 SDK 参考。

- 通过工具管理 BOS

  BOS 提供各类型的管理工具，您可以通过工具管理您的 BOS。详情请参见周边工具。

产品功能

BOS 提供丰富的产品功能，在数据上云、存储桶管理、文件管理、数据发布、数据处理等方面拥有完善的产品能力，详情请查看产品功能。

核心概念

本文将向您介绍对象存储 BOS 产品中涉及的几个核心概念，以便于您更好地理解 BOS 产品。

Object

在BOS中，用户操作的基本数据单元是Object。每个Object包含Key、Meta和Data。其中，Key是Object的名字；Meta是用户对该Object的描述，由一系列Name-Value对组成；Data是Object的数据。

Bucket

Bucket是存放数据的容器，可以当成是存储桶。Bucket具有区域属性，只能位于一个区域。因为Bucket名称创建后不能修改名称和所属区域，所以建议根据业务情况就近存储，方便上传和下载以提高访问速度。Bucket名称具有全局唯一性。每个Object必须都包含在某个Bucket中。一个用户最多可创建100个Bucket，但每个Bucket中存放的Object的数量和大小总和没有限制，用户不需要考虑数据的可扩展性。

Referer白名单

Referer白名单即准入列表，是一种基于HTTP header referer字段的防盗链方法，目的是防止用户存储在BOS上的数据被其他人盗链。用户可以通过BOS控制台设置Referer字段的白名单。设置白名单后，只有Referer字段在白名单内的用户才可以访问Bucket中存储的数据，不在白名单内的请求会被拒绝。但如果用户的Referer为空，默认可以访问，不受白名单限制。

Region

Region代表一个独立地域。百度智能云中的服务除了极少数如账号服务全局有效之外，绝大部分服务都是区域间隔离的。每个区域的服务独立部署互不影响。服务间共享数据需要通过显式拷贝完成。Region 是在创建 Bucket 的时候指定的，一旦指定之后就不允许更改。

百度智能云目前开放了多区域支持，请参考 区域选择说明。

在API中引用区域必须使用其对应区域的服务域名。

区域	服务域名
北京	bj.bcebos.com
保定	bd.bcebos.com
苏州	su.bcebos.com
广州	gz.bcebos.com
阳泉	yq.bcebos.com
香港	hkg.bcebos.com
金融云武汉专区	fwh.bcebos.com
金融云上海专区	fsh.bcebos.com

用户在访问的BOS服务的时候，可以通过URL指定请求落到哪个Region，例如：http(s)://bj.bcebos.com将会把请求落到北京。

Access Key ID / Secret Access Key

用户开通BOS服务后，系统会自动分配一对 Access Key ID (AK)/ Secret Access Key(SK)，该密钥对将在用户向BOS发起请求时用做签名验证。Access Key ID用于标示用户，Secret Access Key 是用户用于加密签名字符串和 BOS 服务用来验证签名字符串的密钥。请参考获取AK/SK。

除系统分配的密钥对外，用户还可自行申请多对AK/SK，总量最多为20对。

权限控制

BOS提供用户签名验证、访问控制列表ACL和对象限时访问相结合的权限控制方式，为用户提供安全可靠的数据保护。其中用户签名验证采用AK/SK非对称加密的方法对URL进行签名来实现用户身份验证；ACL根据签名识别用户身份后，提出请求Bucket的访问权限信息，并根据相应的权限信息对请求做出响应；对象限时访问让用户可以提供自定义时间内有效的URL用于下载等应用场景。

CDN加速

CDN不仅能为包括地图、图片、文档在内的静态网页加载进行加速，也可为音频、下载、游戏等业务提供加速服务，使网站可被高速访问，有效提升网站的用户体验。

分级存储介绍

存储类型介绍

存储类型对比

对比指标	标准存储 -多AZ	标准存储	低频存储 -多AZ	低频存储	冷存储	归档存储
可用性	99.995%	99.99%	99.995%	99.95%	99.95%	99% （数据取回后）
最低存储时间	无	无	30天	30 天	60 天	180 天
数据取回费用	无	无	按量收费（GB）	按量收费（GB）	按量收费（GB）	按量收费（GB），并收取取回接口调用费用（次）
最小计量单位	按实际大小	按实际大小	按实际大小	按实际大小	按实际大小	64 KB
数据取回性能	无需取回	无需取回	自动触发取回，毫秒延迟	自动触发取回，毫秒延迟	自动触发取回，毫秒延迟	需先手动取回后访问 • 加急取回：30分钟内取回 • 标准取回：5小时内取回 • 延缓取回：12小时内取回
适用场景	移动应用、网站视频、数据分析等需极高可靠性的场景	移动应用、网站视频、数据分析等	网盘存储、监控数据、媒资备份等需极高可靠性的场景	网盘存储、监控数据、媒资备份等	医疗影像、业务备份、日志数据、直播转录的备份存储等等	数据长期保存的业务场景。例如基因数据、合规性归档文件等
场景特点	数据频繁读取场景。	数据频繁读取场景。	月均访问次数小于 1 次场景。	月均访问次数小于 1 次场景。	年均访问次数小于 1 次场景。	三年均访问次数小于 1 次场景。

> **说明：**
>
> • 标准存储、标准存储-多AZ、低频存储、低频存储-多AZ、冷存储的数据可靠性都是99.9999999999%（12个9），归档存储的数据可靠性是99.999999999%（11个9）

各存储类型的地域开放情况

BOS 各存储类型在不同地域的开放情况不同，请参考下表进行选购。

地域	标准存储-多AZ	标准存储	低频存储-多AZ	低频存储	冷存储	归档存储
华北-北京	✓	✓	✓	✓	✓	✓
华北-保定	暂无	✓	暂无	✓	✓	✓
华东-上海	暂无	✓	暂无	暂无	暂无	暂无
华东-苏州	✓	✓	✓	✓	✓	✓
华南-广州	✓	✓	✓	✓	✓	✓
西南-成都	暂无	✓	暂无	✓	✓	暂无
华中-武汉	暂无	✓	暂无	✓	✓	✓
中国香港	暂无	✓	暂无	✓	暂无	暂无

注意事项

1. **请根据文件的预期访问频率合理选择存储类型**：
   - **归档存储**适用于**三年均访问次数小于 1 次场景。**比如基因数据、合规性归档文件、归档医疗数据等等；
   - 若为较为频繁访问的冷数据，请选用其他存储类型，如低频存储-多 AZ 和低频存储（适用于月均访问次数小于 1 次）、冷存储（适用于年均访问次数小于 1 次）；

2. **请您注意最小存储周期费用**：

   - 归档存储类型文件的最小存储周期为 180 天。**若归档类型文件的存储时间未满 180 天（如存储时间不满 180 天时删除），BOS 将收取一笔"归档存储最小存储周期"费用，将剩余未存满时间的存储费用一次性收取**；
   - 低频存储-多AZ、低频存储和冷存储类型文件同样有最小存储时间限制。**低频存储-多AZ和低频存储类型文件的最小存储时间为 30 天，冷存储文件的最小存储时间为 60 天**。若未存满相应的时间周期，同样会产生一笔 **"最小存储周期"费用**，该笔费用也将根据剩余未存满时间一次性收取；

3. **请注意数据取回费用**：

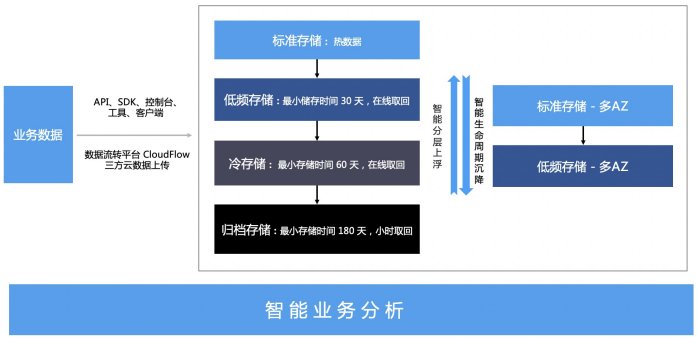
   - BOS 计费项较多，存储类型的选择非常重要。对于不同存储类型而言，标准存储针对频繁访问的热数据，其容量费用最高，但无数据取回费用；归档存储针对几乎不访问的归档备份数据（如三年访问一次的档案数据），其存储的容量费用最低，但偶尔发生数据取回费用最贵。因此，请根据数据的访问频率，选择合适的文件存储类型。更多关于数据取回的费用，请查看 BOS 计费文档；更多关于 BOS 计费项的说明，请查看 BOS 计费项；
   - 在读取低频存储-多AZ、低频存储、冷存储和归档存储类型文件时，均需要**先将文件取回**（标准存储文件不需要取回）；
   - 当您访问低频存储-多AZ、低频存储和冷存储类型文件时，BOS 将根据文件存储类型自动为您触发取回操作，无需手动取回；
   - 当您访问归档存储类型文件时，需要您手动调用 RestoreObject 接口将文件取回，或者在控制台的"文件信息"页面点击"文件取回"；
   - **数据取回是一项收费服务**，相当于 BOS 先将冷冻的文件为您解冻，文件解冻后才可以正常访问和读取。关于数据取回的费用，请查看 BOS 计费文档。对于归档存储类型文件，取回时不仅会产生数据取回（按 GB 计），还会产生数据取回接口调用次数的费用（按取回接口调用次数计）；

4. **归档存储可以分级取回**：

   - 归档存储类型文件属于长期不使用的归档备份文件，访问频率极低。对于归档存储类型文件，您可以根据当前文件对取回效率的要求，触发合适的取回方式：（1）若无特殊需求，归档存储文件在取回时均采用标准取回，一般需要 5 小时左右完成取回；（2）若您遇到紧急情况希望尽快取回文件，您可以触发加急取回，一般需要 30 分钟即可完成取回；（3）若对取回的时间要求不高，如您需要提前一周将归档存储类型文件取回，以应对计划在一周后进行的安全审查，此时您可以使用延缓取回，一般 12 小时左右完成取回操作。关于归档存储文件取回的操作方式，请查看RestoreObject 接口。
   - 归档存储不同级别的取回，取回费用不同：加急取回费用 > 标准取回费用 > 延缓取回费用。更多关于归档存储取回的费用信息，请查看数据取回费用和归档存储取回请求费用。

通过生命周期进行沉降

存储类型不是一成不变的，您可以在文件上传后，通过设置沉降规则来实现存储类型的沉降，如上传 30 天后将某个 Bucket 某一前缀下的文件沉降到低频存储。在您设置了沉降规则后，沉降将自动触发，无需手工修改存储类型。关于生命周期的使用，请查看控制台设置生命周期规则或API-PutBucketLifecycle



计费方式

对于每种存储类型的存储价格、流量费用、请求费用和数据处理费用，请查看计费项概览了解计费项详细信息。关于计费项的价格，请查看按需计费。您也可以提前购买资源包，购买后相应费用将自动从资源包中抵扣。

为什么会有分级存储

为什么会有分级存储？

不同类型数据的使用方式、访问频率均有不同，同时数据也存在老化曲线。随着数据逐渐老化，其访问频率逐步降低，进入归档状态。分级存储是一种方法论，即将不同类别数据存放到不同存储类型，以降低总存储成本，同时满足服务要求。

BOS 针对不同存储场景，提供了标准存储-多AZ、标准存储、低频存储-多AZ、低频存储、冷存储和归档存储共 6 种存储类型。不同存储类型的费用和最小存储时间不同，您可以根据自身需求，选择合适的存储类型。

标准存储-多AZ

- 标准存储-多 AZ 为用户提供高可靠、高可用、高性能的对象存储服务；
- 一份数据在 3 个可用区同步存储，数据可靠性极高，并拥有跨机房的容灾能力；
- 标准存储-多 AZ 适用于需要频繁读取且可靠性要求极高的场景；

标准存储

- 标准存储（Standard）为用户提供高可靠、高可用、高性能的对象存储服务；
- 标准存储适用于需要频繁读取数据的场景；

低频存储-多AZ

- 低频存储-多 AZ 为用户提供高可靠、较低存储费用的对象存储服务；
- 一份数据在 3 个可用区同步存储，数据可靠性极高，并拥有跨机房的容灾能力；
- 低频存储适合需要长期保存且较少访问的数据；
- 相比标准存储，低频存储在在线访问性能、持久性和可用性等方面基本一致，价格却远低于标准存储；

低频存储

- 低频存储（Infrequent Access）为用户提供高可靠、较低存储费用的对象存储服务；
- 低频存储适合需要长期保存且较少访问的数据；
- 相比标准存储，低频存储在在线访问性能、持久性和可用性等方面基本一致，价格却远低于标准存储；

冷存储

- 冷存储（Cold）为用户提供高可靠、低存储费用的对象存储服务；
- 冷存储适合需要长期保存且极少访问的数据；
- 相比标准存储和低频存储，冷存储在在线访问性能、持久性和可用性等方面基本一致，价格却远低于标准存储和低频存储；

归档存储

- 归档存储（Archive）为用户提供高可靠、极低存储费用的对象存储服务；
- 归档存储适合需要长期保存且几乎不访问的数据；
- 相比标准存储、低频存储和冷存储，归档存储在持久性和可用性等方面基本一致，存储价格却远低于标准存储、低频存储和冷存储；

> **如何设置存储类型？**
>
> - 您可以在上传文件时指定文件的存储类型，也可以在创建 Bucket 存储桶时一次性指定存储桶内全部文件的存储类型；
> - 若您上传文件时未指定存储类型，则文件继承 Bucket 的默认存储类型；
> - 若您上传文件时指定的存储类型与 Bucket 默认存储类型不一致，则该文件以文件上传时设置的文件存储类型为准。

存储类型介绍

更多关于存储类型的信息，请查看存储类型介绍。

产品优势

优势分析

稳定

- 服务设计可用性高达 99.995%。
- 数据多副本存储，数据可靠性高达 99.9999999999%。

安全

- 安全的用户认证、授权和传输机制。
- 支持基于签名认证及 ACL 权限设置的资源访问控制。
- 支持白名单功能防止数据被其他人盗链。

高效

- 提供一系列简单易用的 RESTful API、多语言 SDK、管理平台、工具和方案。
- 可与各种云计算产品无缝整合，例如整合 CDN 对 Bucket 进行加速。

弹性

- 系统透明升级和规模自动扩展。
- 支持存储容量弹性扩展。

产品特性

灵活的数据管理

- 创建、查看、删除 Bucket：
  - 用户可以自定义设置需要创建的 Bucket 数量，并支持根据需求修改配额；
  - 用户可以修改、查看 Bucket 的访问权限。
- 上传、查看、删除 Object：
  - 用户可以存储的 Object 没有数量限制；
  - Object 支持文本、多媒体、二进制等任何类型的数据；
  - 每个 Object 可包含从 0 字节到 48.8TB 的数据；
  - 用户可使用通用的和自定义的元信息机制来定义资源属性。

大文件分块上传

- 对于超过 5 GB 的超大文件，用户可以使用分块上传机制；超过 100 KB 但不超过 5 GB 的文件可以由用户选择是否需要分块上传。
- 用户将一个超大文件拆分成子 Object（又称 Part），分别上传这些子 Object，上传完成后 BOS 将自动检查数据，并将子 Object 组合成一个 Object，完成分块上传。
- 分块后的单个 Part 不能超过 5 GB，除最后一个 Part 外，单个 Part 的大小需要大于100 KB。
- 该功能在网络条件较差，或上传文件前无法确定文件大小等场景下具有非常好的表现。

业内领先的分级存储

- BOS 提供了业内领先的多级分层存储。用户可以选择标准存储作为主要存储方式，也可以根据不同访问频率选择成本更低、存储时间更长的的低频存储、冷存储和归档存储。
- BOS 支持生命周期管理。用户可以通过配置生命周期规则，将到期的数据自动转换为相应类型，无需手动操作。

完善的API和开发工具

- BOS 提供标准的 REST 接口，可与任何 Internet 开发工具包搭配使用。
- BOS 基于原生的 REST API，主要提供了以下三类 API：
  - Bucket 操作

- Object 操作
- ACL 操作

- BOS 提供了十多种语言的 SDK，Java、Python、PHP 等。

功能完善的管理平台

- 用户可以通过 Web 管理平台方便的操作 Bucket 和 Object。
- 用户可以通过 Web 管理平台中的 AK / SK 管理功能轻松的创建和管理自己的 Access Key ID / Secret Access Key。
- 用户可以通过 Web 管理平台提供的监控功能，实时监控 Bucket 的读写情况和占用空间等信息。

海量数据存储

- BOS 支持任何类型数据的存储、分享及使用。允许用户将整个存储基础设施转移到云端，利用 BOS 的扩展性和按需付费的优势，以处理不断增长的存储需求。
- BOS 支持最大 48.8TB 的单文件存储，利用断点上传及下载功能，可以存储多媒体视频、软件包等超大文件。

产品功能

概述

本文介绍 BOS 常用功能及使用方式。在您阅读功能概览前，请先查看 BOS 简介和核心概念，以方便您更好的了解 BOS 功能的使用场景和功能含义。

功能列表

BOS 在数据上云、存储桶管理、文件管理、多用户访问控制、域名及发布管理、数据处理、统计与监控服务、数据湖接入、流量控制等方面具有完备且精细的管理能力。在操作方式上，BOS 支持通过控制台、API、SDK、工具等方式进行访问和操作。

分类	产品功能	功能说明	参考链接
数据上云	本地服务器数据上云： • BOS CMD工具 • 月光宝盒离线迁移	数据上云，可以分为"本地服务器数据上云"、"三方云数据迁移"两种场景，每种场景都会面临上传存量和增量数据的问题。BOS 提供多种数据迁移方式，在保证数据一致性的前提下，尽量减少迁移的额外开销。	**相关文档：** • BOS CMD工具 • 月光宝盒离线迁移 • BOS Import工具 • CloudFlow • 本地服务器数据上云 • 三方云数据迁移 • 数据上云方案典型实践
	三方云数据迁移： • BOS Import工具 • CloudFlow		
存储桶管理	控制台查看存储桶概览	可通过控制台查看Bucket 用量数据、监控数据、配置信息和域名发布信息。	**相关文档：** • 控制台查看 Bucket 概览数据
	创建存储桶	在您使用BOS提供的服务之前，需要创建一个Bucket用于存储文件。Bucket具有各种配置属性，包括所属地域、默认存储类型、读写权限等。您可以根据实际需求，创建不同类型的Bucket来存储不同的数据。	**相关文档：** • 控制台创建 Bucket **相关接口：** • 创建Bucket - PutBucket

			• 设置Bucket默认存储类型 - PutBucketStorageclass
	设置存储桶权限	BOS 提供了丰富且精细化的多维度权限管理能力： • Bucket标准权限：私有、公共读和公共读写粒度的配置 • 粗粒度自定义权限：对指定用户设置 READ、LIST、WRITE、FULL_CONTROL 和 MODIFY 权限 • 细粒度自定义权限：BOS 提供的 18 种 API 级别细粒度自定义权限。	**相关文档：** • 控制台设置 Bucket 读写权限 • 开发文档-设置 Bucket 读写权限 **相关接口：** • 设置Bucket访问权限 - PutBucketAcl • 获取Bucket访问权限 - GetBucketAcl
	设置生命周期管理	生命周期管理功能，以帮助用户自动化地完成数据的生命周期管理，实现数据从创建到归档到删除的自动管理流程，从而节约人力和存储费用。	**相关文档：** • 控制台设置生命周期管理 **相关接口：** • 创建生命周期管理规则 - PutBucketLifecycle • 获取定义的生命周期管理规则 - GetBucketLifecycle • 删除定义的生命周期管理规则 - DeleteBucketLifecycle
	控制台设置智能分层	根据业务读写情况，智能进行沉降或回暖，帮助用户在读写性能与成本之间寻找最优平衡点。	**相关文档：** • 配置智能分层
	设置镜像回源	若您为 Bucket 配置了镜像回源，当用户访问该 Bucket 中某一 Object 且该 Object 在 BOS 中并不存在，BOS 会立刻向预先配置好的回源地址请求该 Object，保证访问正常进行。	**相关文档：** • 设置镜像回源 **相关接口：** • 创建Bucket镜像回源规则 - PutBucketMirroring

			• 获取Bucket镜像回源信息 - GetBucketMirroring • 删除Bucket镜像回源配置 - DeleteBuketMirroring
	设置服务端加密	BOS 支持在服务端为数据进行 SSE（Server-Side Encryption）加密编码。当您在配置 SSE 加密后，在上传文件时 BOS 会通过携带加密参数的方式来对上传数据执行服务端加密设置。	**相关文档：** • 控制台设置服务端加密 • 客户端加密典型实践 **相关接口：** • 开启指定Bucket的加密开关 - PutBucketEncryption • 判断服务端加密是否开启 - GetBucketEncryption • 关闭服务端加密功能 - DeleteBucketEncryption
	设置数据同步	数据同步功能能够很好的提供 Bucket 跨区域、 跨账号或其他数据复制的需求。目标 Bucket 中的对象是源 Bucket 中对象的精确副本，它们具有相同的对象名、元数据以及内容，例如创建时间、拥有者、用户定义的元数据、Object ACL、对象内容等。	**相关文档：** • 控制台设置数据同步 **相关接口：** • 创建数据同步 - PutBucketReplication • 获取数据同步信息 - GetBucketReplication • 获取数据同步状态 - GetBucketReplicationProgress • 删除数据同步配置 - DeleteBucketReplication • 获取Bucket所有同步规则 - ListBucketReplication
	设置跨域访问	BOS 支持 HTML5 协议中的跨域资源共享 CORS 设置，帮助用户实现跨域访问。	**相关文档：** • 控制台设置跨域访问 • 如何解决浏览器跨域CORS问题典型实践

			相关接口： • 设定跨域资源共享规则 - PutBucketCors • 获取Bucket的CORS规则 - GetBucketCors • 关闭指定Bucket的CORS功能并清空所有规则 - DeleteBucketCors
	控制台设置 Referer 白名单	BOS 支持基于 HTTP header referer 字段的防盗链方法。您可以在 BOS 控制台设置 Referer 字段的白名单。设置白名单后，只有 Referer 字段在白名单内的用户才可访问 Bucket 中存储的数据，不在白名单内的请求会被拒绝。	**相关文档：** • 控制台设置 Referer 白名单 • 防盗链使用典型实践
	设置合规保留	Bucket 合规保留是数据保护的一种方式，用于规避重要的数据存入 Bucket 后被意外删除的风险，适用于金融、保险、医疗、证券等行业。合规保留是一种基于时间的配置策略，用户开启 Bucket 合规保留后需要设定数据保留的时间期限。	**相关文档：** • 控制台设置合规保留 **相关接口：** • 创建基于时间的合规保留策略 - InitBucketObjectLock • 获取合规保留策略配置信息 - GetBucketObjectLock • 删除合规保留策略 - DeleteBucketObjectLock • 延长合规保留策略保护周期 - ExtendBucketObjectLock
	设置访问日志	BOS 日志功能可以应用在访问统计和安全审核等方面，每条访问日志记录了单个访问请求的详细信息，包括请求者、Bucket 名称、请求时间和请求操作等信息。开启访问日志功能后，会自动将对这个 Bucket 的访问请求，以小时为单位，按照固定的命名规则，生成日志文件写入用户指定的 Bucket。	**相关文档：** • 控制台设置访问日志 **相关接口：** • 开启访问日志 - PutBucketLogging • 获取指定Bucket访问日志配置情况 - GetBucketLogging • 关闭指定Bucket访问日志功能 - DeleteBucketLogging

	设置存储空间清单	存储空间清单是一种帮助用户管理存储桶中对象的功能。BOS 可根据用户的清单任务配置，定期每天或者每周扫描用户 Bucket 中指定对象并输出一份清单报告，以 CSV 格式文件的形式存储到用户指定的 Bucket 中。CSV 文件中会列出对象及其对应的元数据，并根据用户的配置信息，记录用户所需的对象属性信息。	**相关文档：** • 控制台开启存储空间清单 **相关接口：** • 创建清单任务 - PutBucketInventory • 获取指定Bucket清单配置（需id） - GetBucketInventory • 获取指定Bucket清单配置（无需id） - ListBucketInventory • 删除指定Bucket清单配置 - DeleteBucketInventory
	设置请求者付费	请求者付费模式是指由请求者支付读取存储空间（Bucket）内数据时产生的流量和请求费用，而 Bucket 拥有者仅需支付存储费用。当您希望共享数据，但又不希望产生流量费用和请求费用时，您可以开启此功能。	**相关文档：** • 控制台设置请求者付费 **相关接口：** • 创建请求者付费 - PutBucketRequestPayment • 获取指定Bucket请求者计费规则 - GetBucketRequestPayment
	设置回收站	为了提高 BOS 数据的可靠性，用户可以通过配置回收站的功能，保证删除后的数据在回收站内继续保留，用于后续对删除数据的找回。	**相关文档：** • 控制台设置回收站 **相关接口：** • 开通回收站 - PutBucketTrash • 获取指定Bucket回收站状态 - GetBucketTrash • 关闭回收站 - DeleteBucketTrash

	设置静态网站托管	BOS 支持用户将静态网站托管在 Bucket 上，实现网站轻量化运维。设置生效后，用户可通过直接访问该Bucket的域名访问该托管网站。	**相关文档：** • 控制台设置静态网站托管 **相关接口：** • 设置静态网站托管 - PutBucketStaticWebsite • 获取静态网站托管信息 - GetBucketStaticWebsite
	设置 Bucket 标签管理	百度智能云提供标签管理功能，通过给每个云资源添加标签，从而快速分类和识别管理这些资源，并可以按照标签查看账单。	**相关文档：** • 控制台设置 Bucket 标签管理 **相关接口：** • 为单个Bucket设置标签 - PutBucketTagging • 删除单个Bucket的标签 - DeleteBucketTagging • 获取单个Bucket的标签 - GetBucketTagging
	删除Bucket	只有当 Bucket 为空时，即不包含任何 Object 和未完成的三步上传 Part，才可删除 Bucket，否则无法删除并出现相应提示。	**相关文档：** • 控制台删除 Bucket **相关接口：** • 删除Bucket接口 - DeleteBucket
	额度管理	• Bucket级别：针对企业对自身财务和资源的自定义控制场景，BOS 提供 Bucket 级别额度管理功能。该功能支持对单个 Bucket 的文件数量上限和容量上限进行配置，实现对	**相关文档：** • Bucket额度管理

		存储服务弹性能力的自定义控制。 • 账号级别：BOS也支持对您账号中 Bucket 数量上限和总容量上限进行配置，实现对存储服务弹性能力的自定义控制。	• 额度管理 **相关接口：** • 为指定Bucket设置容量及文件上限 - PutBucketQuota • 获取Bucket额度信息 - GetBucketQuota • 删除Bucket的额度配置 - DeleteBucketQuota
	其他常用 Bucket API ： • 获取Bucket所在的区域 - GetBucketLocation • 获取请求者拥有的所有Bucket - ListBuckets • 获取Bucket存储类型 - GetBucketStorageClass • 获取Bucket中Object的信息列表 - ListObjects • 查看Bucket请求者及权限 - HeadBucket		
文件管理	上传文件	您可以通过控制台、API、SDK、BOS 桌面客户端等各类方式进行文件上传，并在上传文件时指定 Object 的 Meta 信息、存储类型等。针对不同的使用场景，BOS 支持简单上传、表单上传、分块上传及断点续传和追加上传，或者参考文件上传典型场景下的典型实践Web 端直传实践。	**相关文档：** • 控制台上传文件 • 简单上传 • 表单上传 • 分块上传及断点续传 • 追加上传 • Web 端直传实践 • 通过各类 SDK 上传文件 • BOS 桌面上传文件 **相关接口：** • 向指定的Bucket上传文件 - PutObject • 使用HTML表单上传文件到指定Bucket - PostObject

			• 以追加写的方式上传文件 - AppendObject • 三步上传/三步Copy - MultipartUpload
	控制台/桌面工具重命名文件	可在控制台以及桌面工具中对已上传的文件进行重命名。	**相关文档：** • 重命名文件 • BOS桌面工具使用指南
	设置 Meta 信息	数据元信息（Object Meta）是文件所携带的元数据信息，用以标识文件数据特征，您可以在控制台修改文件的 Meta 信息。	**相关文档：** • 控制台设置 Meta 信息 • 数据元信息（Object Meta） **相关接口：** • 获取指定Object的Meta信息 - GetObjectMeta
	设置文件权限	您可以单独修改 Object 的权限信息。在不修改文件权限的情况下，该文件的访问权限将和所在的 Bucket 权限保持一致。	**相关文档：** • 控制台设置文件权限 • Object权限设置 **相关接口：** • 设置指定Object的访问权限 - PutObjectAcl • 获取指定Object的访问权限 - GetObjectAcl • 删除指定Object的访问权限 - DeleteObjectAcl
	复制和移动文件	您可以通过控制台、API、SDK 对已上传的文件进行复制和移动操作。	**相关文档：** • 控制台复制和移动文件 • 拷贝数据 **相关接口：** • 拷贝Object - CopyObject

控制台修改文件存储类型		存储在 BOS 中的文件，您可以在任意时间内转换其存储类型。当前，BOS 支持将存储类型在标准存储，低频存储和冷存储之间转换，暂不支持归档存储类型转换。	**相关文档：** • 控制台修改文件存储类型
下载/预览文件		文件上传至Bucket后，您可以将文件下载至本地指定路径；您可对图像、视频等文件进行预览；您可以将文件链接分享给第三方，供其下载或预览。	**相关文档：** • 控制台下载/预览/分享文件 **相关接口：** • 获取指定Object - GetObject • 取回归档存储文件 - RestoreObject
分享文件/文件夹	控制台/桌面工具分享文件	• 文件（Object）上传至存储空间（Bucket）后，您可以通过控制台获取单个或多个文件URL，将文件URL分享给第三方预览或下载。 • 也可通过桌面工具获取单个文件URL。	**相关文档：** • 分享文件-获取文件url • BOS桌面工具使用指南
	控制台/桌面工具分享文件夹	• 当您需要批量分享某一个存储桶（Bucket）中的多个文件（Object）时，可以创建文件夹并将所有待分享的文件存储在文件夹中，通过控制台文件夹分享功能生成分享链接。 • 也可通过桌面工具获取单个文件夹URL。	**相关文档：** • 文件夹分享 • BOS桌面工具使用指南
	控制台联动网盘分享	网盘分享功能是由对象存储（BOS）与百度网盘企业定制平台联合提供的文件分享功能，您可以将BOS中的文件或文件夹批量同步至网盘定制服务专属空间，之后可生成网盘的分享链接进行数据分发交付。	**相关文档：** • 联动百度网盘分享
控制台/桌面工具管理文件夹		为了更加方便的管理数据，BOS 控制台、BOS桌面工具提供文件夹功能，您可以根据需求在 Bucket 中创建文件夹以存放文件，也可以按需删除文件夹。	**相关文档：** • 控制台管理文件夹 • 控制台批量删除文件夹 • BOS桌面工具使用指南
控制台文件夹统计		您可以在控制台统计 Bucket 中某一前缀下全部文件数量和总容量。	**相关文档：**

			• 文件夹统计
	控制台/桌面工具分享文件夹	当您需要批量分享某一个存储桶（Bucket）中的多个文件（Object）时，可以创建文件夹并将所有待分享的文件存储在文件夹中，通过控制台或BOS桌面工具中的文件夹分享功能生成分享链接。接收方通过浏览器访问分享链接或使用BOS桌面工具登录，即可下载文件夹中的内容。	**相关文档：** • 文件夹分享 • BOS桌面工具使用指南
	碎片管理	进行分片上传时，文件（Object）会被切成若干个 Part，这些 Part 被称为碎片。当您将这些 Part 上传到 BOS 后，可以调用 CompleteMultipartUpload 将这些碎片（Part）组合成一个完整的Object。您可在控制台对碎片进行管理。	**相关文档：** • 碎片管理 **相关接口：** • 初始化分片上传 - InitiateMultipartUpload • 上传碎片（Part）- UploadPart • 复制碎片（Part）- UploadPartCopy • 碎片组合成Object - CompleteMultipartUpload • 中断分片上传请求 - AbortMultipartUpload • 列举所有已上传成功的Part - ListParts • 列举所有未完成的分片上传请求 - ListMultipartUploads
	软链接	您可为常用文件设置软链接，便于快速访问。	**相关文档：** • 软链接 **相关接口：** • 为指定Object创建软链接 - PutSymlink • 获取Object的软链接 - GetSymlink
	控制台批量处理	批量处理功能支持用户对BOS存储桶内指定的对象列表执行指定的批量处理操作。您首先需要指定一份待处理对象清单，完成批量处理任务配置后，BOS即可根据清单执行批量处理任务。	**相关文档：** •

			批量处理
	删除文件	您可以通过控制台、API、SDK 删除已上传文件。	**相关文档：** • 控制台删除文件 • 删除数据 **相关接口：** • 删除指定Object - DeleteObject • 删除同一Bucket下多个Object - DeleteMultipleObjects
	其他常用 Object API ： • 从指定URL抓取资源，并将资源存储到指定的Bucket中 - FetchObject • 获取某个Object - GetObject		
多用户访问控制	设置多用户访问控制	多用户访问控制，主要用于帮助用户管理云账户下资源的访问权限，适用于企业内的不同角色，可以对不同的工作人员赋予使用产品的不同权限，当您的企业存在多用户协同操作资源时，推荐您使用多用户访问控制。	**相关文档：** • 控制台设置多用户访问控制 **相关接口：** • 请求临时访问凭证 - GetSessionToken
域名及发布管理	控制台/桌面工具查看Bucket官方域名	BOS 支持为已创建的 Bucket 设置多种域名，从而对存储的数据进行发布。 当您创建一个 Bucket 之后，BOS 会自动为其创建一个官方域名，且该官方域名与 Bucket 名称一一对应且唯一存在。	**相关文档：** • Bucket 官方域名
	控制台创建Bucket自定义域名	BOS 支持为 Bucket 添加自定义域名，将该自定义域名与 Bucket 绑定后，即可利用该自定义域名将 Bucket 中的数据进行发布。您可使用自定义域名直接访问绑定的 Bucket。	**相关文档：** • Bucket 自定义域名
	控制台开通CDN官方加速域名	BOS 支持开启 Bucket 的 CDN 域名，从而实现 CDN 下行加速分发。	**相关文档：**

				• CDN 官方域名
	控制台配置缓存管理		BOS提供缓存管理能力，您可对缓存数据进行自动管理。	**相关文档：** • 缓存管理
	CDN上行加速		开启 CDN 域名后，还可以开启 CDN 上行加速功能，对上行场景进行加速。	**相关文档：** • CDN 上行加速 • 使用 CDN 加速 BOS 典型实践 • 使用 CDN 动态加速上传到 BOS 典型实践
数据处理	图像处理	图片处理	您可以通过控制台、API对图像进行处理。BOS 支持裁剪、旋转、缩放、内切圆、水印等多种图像处理方式。 图像处理协议可以广泛应用于手机 APP 中。关于手机 APP 典型场景下的 BOS 文件直传和图像处理，请参考手机美图 APP 实践。	**相关文档：** • 控制台设置图片处理 • API 调用图像处理 V2.0 • API 设置原图保护 • 手机美图 APP 实践
		图片审核	BOS 提供基于深度学习的智能内容审核方案，准确识别图片和视频中的涉黄、涉暴涉恐、政治敏感、微商广告等内容。	**相关文档：** • 控制台设置图像审核 • API 调用图像审核 • CLI 工具调用图像审核
	视频处理	音视频处理	BOS 提供业界全新的音视频处理接口，支持上传 mp4 格式文件并自动保存为 hls 格式，用于 m3u8 风格播放器读取使用，充分满足“UGC 上传——自动化处理——播放”这条使用流，免去原本复杂的视频操作处理逻辑，为业务提速。	**相关文档：** • 音视频处理 • API 调用音视频处理
		视频即时转码	百度智能云BOS提供了业界全新的视频即时转码接口，支持通过设置url参数的方法即时转换视频的码率，实现h264，h265两种码率的互转。	**相关文档：** • 视频即时转码

	视频截帧	BOS提供了业界全新的视频截帧接口，支持通过设置url参数的方法对存储在bos的视频文件进行截取。	**相关文档：** • 视频截帧
	获取视频元信息	本接口用于获取储存在BOS上的视频文件的元信息。	**相关文档：** • 获取视频元信息
文档处理	文档预览	文档预览功能基于百度智能云文档服务 DOC（Document Service），为开发者提供 Office、WPS 等多种格式的文档存储、管理及在线浏览服务。	**相关文档：** • 文档预览
事件通知	控制台设置事件通知 API调用事件通知	• BOS 事件通知功能，是基于 BOS 数据处理框架 V2.0 生成的一套智能触发平台。开发者可基于平台来 Bucket 中发生的数据行为。如 BOS 会监测 Bucket 中发生的 Put Object、Append Object 等行为，向用户的下游业务系统提供触发通知； • 同时，BOS 结合百度强大的 AI 能力，将监测到的用户行为直接触发各项数据处理能力，如通用物体识别、通用文字识别、音视频转码、函数计算 CFC 等，将处理后的结果直接给到用户下游业务系统。	**相关文档：** • 控制台设置事件通知 • API 调用事件通知 • 通用物体识别 • 通用文字识别 • 音视频转码 • 函数计算 CFC
ZIP打包压缩		ZIP 多文件打包压缩是百度智能云对象存储（Baidu Object Storage，BOS）基于 函数计算（Cloud Function Compute）为用户提供的数据处理解决方案。用户为存储桶添加多文件打包压缩规则后，指定需要打包压缩文件的 URLs，再通过触发 CFC 函数来执行打包压缩的动作，并将最终的压缩包投递至存储桶的指定路径下。	**相关文档：** • zip打包压缩
文件自动解压		文件解压缩功能是对象存储（BOS）基于 百度智能云函数计算（CFC）为用户提供的数据处理解决方案。用户为存储桶添加文件解压缩规则后，当压缩文件上传到对象存储时，将自动触发对象存储为您预配置的云函数，自动将文件解压到指定的存储桶和路径中。	**相关文档：** • 文件自动解压
Select扫描		Select 扫描功能的典型应用场景是大数据场景。您可以将大数据处理操作在云端完成，无需将数据下载到本地。	**相关文档：** • 控制台Select扫描 **相关接口：**

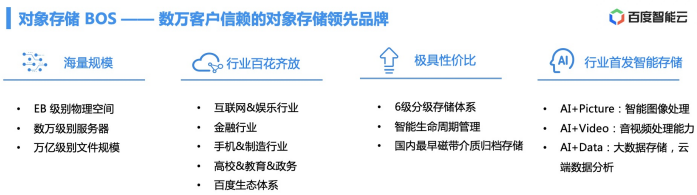
			• 对Bucket中指定object执行SQL语句，选取出指定内容返回 - SelectObject
统计与监控服务	控制台查看、导出统计分析与监控数据	• BOS支持Bucket级别统计：能对 Bucket 中数据进行统计分析，并从资源用量、API统计、智能分层等维度进行分析，以图形或报表形式呈现。 • BOS支持全局统计：能对账户下所有合计数据进行统计分析，并从用量、存储类型、请求、地域等维度进行分析，以图形或报表形式呈现。 • BOS深度对接云监控BCM（Baidu Cloud Monitor）为您提供系统基本运行状态、性能以及计量等方面的监控数据指标，并且提供自定义报警服务，帮助您跟踪请求、分析使用情况、统计业务趋势，及时发现以及诊断系统的相关问题。 • BOS 支持在全局概览页查看和导出用量数据。数据指标涵盖单个 Bucket 总容量，总流量和总请求次数，并支持对汇总数据和细粒度数据一键下载。	**相关文档：** • 统计与监控 • 云监控操作指南 • 用量查看和导出
数据湖接入	分层命名空间	分层命名空间（namespace）是百度智能云推出的元数据管理功能。通过分层命名空间，可以很好的支持目录级别的mv与rename操作，同时优化常见读操作List、Head，提升数据处理效率与性能，能很好的满足大数据领域的数据湖计算场景。	**相关文档：** • 分层命名空间
流量控制	单链接限速	您可以在调用 BOS 上传、下载相关 API 接口时自定义文件 URL 的访问限速，从而对整个存储桶进行流量控制，避免大流量服务造成带宽挤占，从而影响其他应用服务。	**相关文档：** • 单链接限速

常见问题

关于 BOS 功能和使用过程中的常见问题，请参考常见问题概览。

应用场景

随着用户场景的多样化，对象存储 BOS 早已不再是单纯解决读和写问题的单一存储产品。对象存储 BOS 在各个行业都拥有多样化的解决方案，包括存储分发场景方案、冷数据归档存储方案、数据容灾备份方案、高性能大数据计算场景方案、高性能 AI 计算解决方案、多媒体数据处理等等。

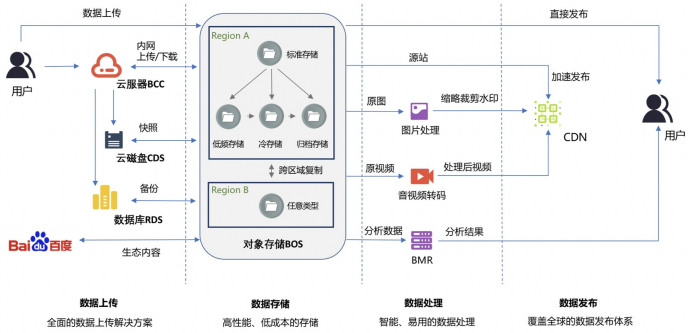


1.存储数据全球分发

随着移动通信技术的不断发展，越来越多的数据被生产、被存储、被分发。以往的数据分发以单点分发场景为主，即从一个数据中心分发到定点终端设备。然而，随着 4G、5G 等技术的发展，数据分发逐渐演变为数据从多个数据中心向全球各地终端进行分发的格局。

对于数据分发而言，热点文件（热门视频、游戏更新包等）的分发场景尤为重要。如互联网行业长视频的电影、电视剧点播、新兴短视频的点播、音频文件全球各地的传输和共享，都需要配合低时延、高并发、大容量的分发系统。对于这种场景，您可以使用对象存储 BOS 和内容分发网络 CDN 来应对大流量和高并发。您可以将数据存储在云端，即对象存储 BOS，利用百度智能云内容分发网络 CDN 遍布全球的边缘节点，将内容分发至全球各地的用户终端。

推荐配置： 对象存储BOS + 内容分发CDN

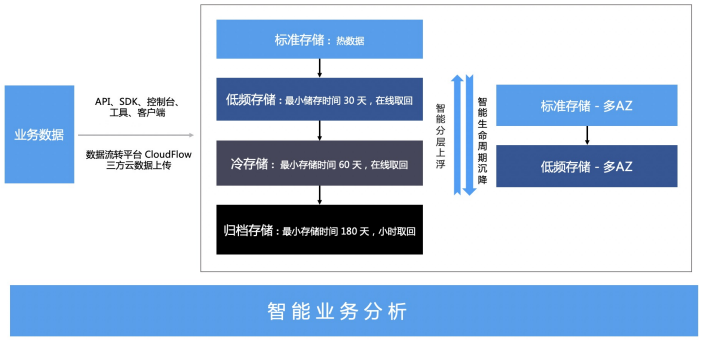


2.冷数据归档存储

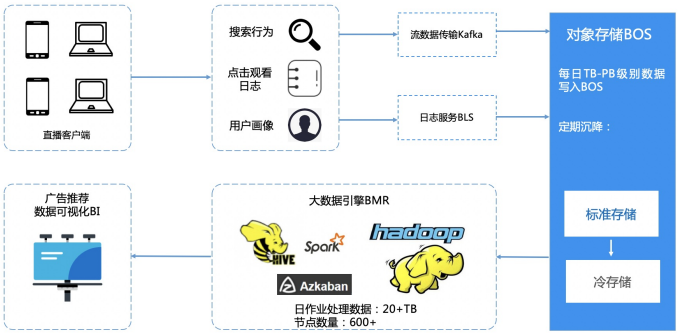
随着时间的推移，数据的访问频率会逐渐降低。如直播场景中，往往需要将直播视频形成录播视频，以方便用户在日后观看。这类视频往往在直播当日，或者直播后的几天内访问频率比较高，在 1 个月、3 个月、半年或几年之后便不会再有人观看，直播当时的"热数据"，随着时间的推移就会变为"冷数据"。

在这类场景中，根据国家法律法规要求，直播转录播视频，尤其是电商直播的录播视频需要保存三年，很多场景下的媒资数据也要保存至少 180 天。因此，对于企业来说，在满足冷数据存储可靠性的同时，降低冷数据的存储成本显得格外重要。

对象存储 BOS 拥有完善的分级存储体系，共有标准存储-多AZ、标准存储、低频存储-多AZ、低频存储、冷存储和归档存储 6 种存储类型，满足客户数据对从热到冷沉降的需求。



推荐配置： 对象存储BOS + 音视频直播LSS

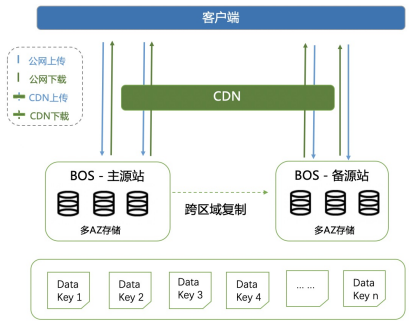


3.数据的容灾备份

随着云计算技术的发展，越来越多的企业将数据从自建数据中心（自建 IDC ）迁移到云端，云存储也随之成为企业的基础设施。通过云端完善的分级存储和数据容灾功能，BOS 可提供高可靠的云端备份方案。

针对数据备份的场景，BOS 提供分级存储的解决方案。对于那些需要长期备份同时访问频率很低的数据，可以存放在低频和冷存储。BOS 还支持生命周期管理，帮助您自动对冷数据进行下沉。此外 BOS 还提供多地域支持，用户可以在地域之间进行数据备份，实现异地容灾。

推荐配置: 低频存储 + 冷存储 + 生命周期管理 + 跨区域复制



- **物理机级别容灾**
  BOS采用分布式存储，实现数据的物理机级别容灾
- **机房级别容灾**
  通过多 AZ 存储类型实现数据在 3 个机房的容灾
- **跨地域容灾：**
  将某个地域下的数据跨地域同步到另一个地域，实现数据的跨地域容灾
- **镜像回源：**
  CDN配置主备回源地址，主bucket会把数据在最长一分钟内同步到备份bucket，主源站故障后会自动切到源站

4.高性能大数据计算场景方案

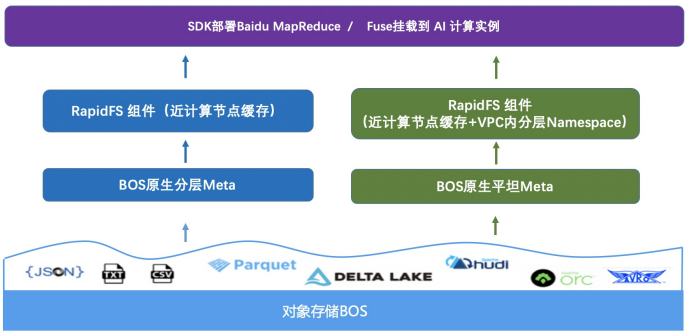
随着大数据时代的到来，越来越多的企业需要对自身大量数据进行大数据分析和大数据计算。对象存储作为云端最佳存储方案，BOS 提供一整套高性能大数据计算解决方案。在大数据场景中，BOS 可以作为您本地自建或者云端 MapReduce 集群的底层存储系统，全方位满足数据处理、建模预测、机器学习及海量数据实时查询任务的需求。

对象存储 BOS 提供原生分层命名空间能力，可以将大数据计算中的大规模批量操作如 Rename、List、Head 等操作的时延降低约 70%，突破传统对象存储支持大数据计算方案的延迟瓶颈。

此外，BOS 还提供完整的 RapidFS 方案，能够在近计算节点进行热数据缓存，从而减少调用 BOS 时所产生的时延。同时，RapidFS 还提供计算节点 VPC 内的分层命名空间功能。通过 RapidFS 的缓存和分层能力，

将大数据计算中的大规模批量操作如 Rename、List、Head 等操作的时延降低一个数量级，达到微秒级别，彻底突破传统对象存储支持大数据计算方案的延迟瓶颈。

推荐配置：对象存储BOS + 百度MapReduce BMR + 百度OLAP引擎Palo + RapidFS



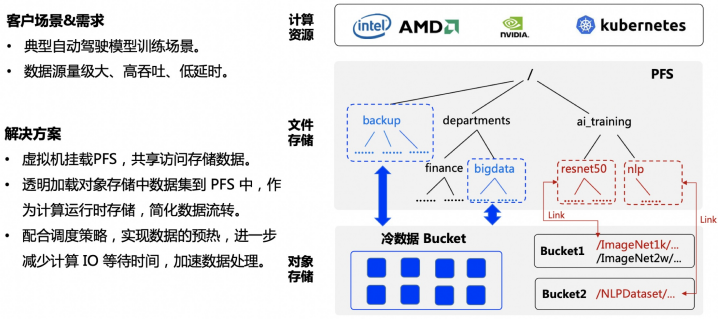
5.高性能 AI 计算场景方案

人工智能作为当下发展趋势，越来越多的企业用户希望通过 AI 的方式改变已有的生产方式，如自动驾驶、传统工业、各类互联网企业等等。然而，AI 计算需要底层大量的资源协同，还需要有大规模的存储系统来支撑数据集、特征数据、训练结果的存放。

对于数据集而言，往往需要企业中很多训练模型同时对数据集进行读取，这无形中对存储系统的吞吐量和延迟形成巨大的挑战。同时，AI 计算场景中往往需要文件系统，而对象存储普遍采用 S3 的数据形态，不符合企业对文件接口的需求。

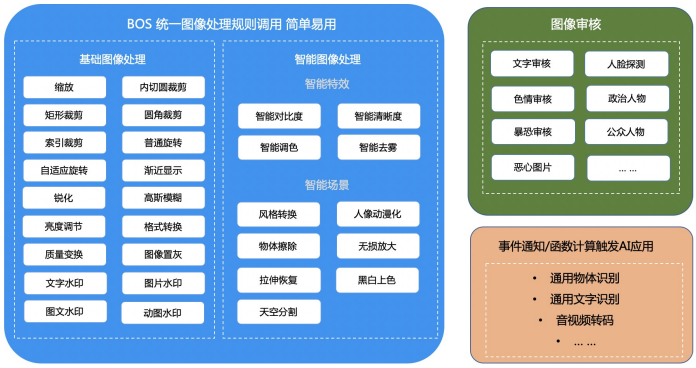
针对以上问题，对象存储 BOS 推出 AI 场景下的存储方案。您可以使用组件 RapidFS 和并行文件系统 PFS 来连接对象存储 BOS，将数据高速读写到计算的 GPU 节点。该方式完全不改变现有的使用方式，并且具有高性能、低时延、大吞吐、简单易用等特点。

推荐配置：对象存储 BOS + RapidFS + 并行文件系统 PFS +云服务器BCC-GPU

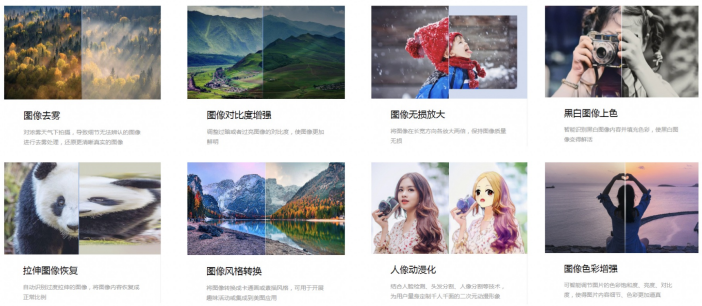


6.多媒体数据处理

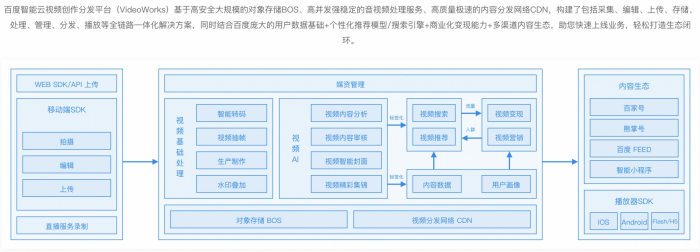
BOS 提供丰富的数据处理功能。您可以对上传到 BOS 的图片进行裁剪、旋转、加水印、缩放、质量变换等图片处理，也可以对音视频文件、文档文本进行转码播放；通过事件通知机制和函数计算 CFC，您还可以实现灵活的自定义事件处理。



同时，针对各类场景，BOS 还提供 AI 图像处理能力，包括智能场景、智能特效等等，能够针对活动运营场景、社交场景、电商场景、海报场景等提供完善的图像处理能力。



对于视频数据，您还可以结合百度智能云视频创作分发平台 VideoWorks 进行音视频编辑。通过自定义工作流的方式，一键制作视频封面、进行视频转码等等。



推荐配置: 对象存储 BOS + 音视频处理 MCP + 文档服务DOC + 视频创作分发平台 VideoWorks。

使用限制

概述

BOS 一些功能在额度和数量上有一定限制，请您在使用前先阅读以下文档。

使用限制

限制类型	具体说明
存储类型	• 不同存储类型所支持的售卖地域也有所不同，请查看 存储类型介绍；
Bucket	• BOS 支持用户自定义账号下 Bucket 总数，但一个主账号下能够创建的 Bucket 数量最多为 100。若想调整 Bucket 总数，请提交工单与我们联系； • 账号下 Bucket 总容量默认无上限；您也可以通过 BOS 额度管理功能对账号总容量进行自定义； • 单个 Bucket 容量无上限； • BOS Bucket 名称全局唯一，创建者必须为新 Bucket 设置唯一的名称，Bucket 名字最长为 64 个字节； • 单个 Bucket 最多支持创建 1000 条生命周期规则； • 单个 Bucket 最多支持创建 20 条数据同步规则；
Object	• Object 名字最长为 1024 字节； • 用户自定义 Meta 最大为 2048 字节； • 支持上传最大为 48.8 TB （50000 GB）的单个文件； • 控制台支持上传最大为 5 GB 的单个文件； • 通过分块方式上传文件，支持的最小分片大小为 100 KB，最大分片为 5 GB，最大文件支持 48.8 TB； • 通过 put、append 等简单上传方式上传文件，最大支持 5 GB 单文件； • 用户删除后的 Object 无法找回，在删除前建议做好确认工作；
	• BOS 对单 Bucket 设置的公网带宽阈值为 10 Gbit/s，内网带宽阈值为 50 Gbit/s。当用户的上传或下载行为达到带宽阈值时，系统会给予 RequestRateLimitExceeded 错误的告警。此限制目前对 BOS 所有 Region 都生效，如您有特殊的业务需求，希望在短时间内上调带宽阈值，请提交工单与我们联系；

数据传输	误的告警，此限制目前对 BOS 所有 Region 都生效。如您有特殊的业务需求，希望在短时间内上调带宽阈值，请提交工单与我们联系； • BOS 对单 Bucket 设置的 QPS 为上传 1500q/s ，单Bucket下载 QPS 不限制，对单个热点Object 下载 QPS 为 800q/s，单个热点Object上传 QPS 为20q/s。如您有特殊的业务需求，希望在短时间内上调阈值，请提交工单与我们联系；
域名	• 单个 Bucket 除官方域名外，最多支持用户绑定 200 个自定义域名；
图片处理	• 支持的原图格式：jpg、png、bmp、webp、heic、gif、avif； • 支持的目标格式：jpg、png、bmp、webp、heic、gif、avif； • 处理文件大小不超过 20 MB； • 原图宽高不超过30000像素且总像素不超过2亿像素。针对动图，原图宽 x 高 x 帧数不超过2亿像。对于图片水印，水印图加原图不超过2亿像素； • 旋转操作中，图片的宽或者高不能超过4096 px； • 处理结果输出图片宽高不超过9999px，包括管道和子命令间的中间输出图片； • 最多可以指定 16 组操作，即 16个 action； • 一个 action 下不可以重复指定同一个参数； • 动图处理结果不超过 20 帧； • 样式名称支持数字，字母和下划线组合，且首位不能为下划线，最长 64 个字符。单张图片可最多设置16张图片水印； • 单个 Bucket 最多支持创建 400 条图像处理样式；
数据同步	• 对于归档型 Object，数据同步暂时不支持，进行数据同步时会忽略归档型 Object。详见设置数据同步；
数据生命周期管理	• 每个 Bucket 可以有至多 1000 条规则； • 基于文件访问时间记录的生效策略，目前仅北京和苏州区域支持； • 单 AZ 类型文件仅能沉降到单 AZ 类型文件，无法沉降到多 AZ 类型文件；标准存储-多AZ 类型文件只能沉降到低频存储-多 AZ 类型文件；
转换文件存储类型	• 通过 Append 上传的文件不支持转为归档类型； • 归档存储类型文件取回完成后才能转为其他存储类型；

产品定价

计费项介绍

计费项概览

概述

BOS 服务的计费项包括存储费用、流量费用、请求费用、数据处理费用四部分。根据您的使用情况，每一项会产生不同的费用。

计费方式

BOS 目前提供按需计费和预付费资源包两种计费形式：

- **按需计费计费方式**： BOS 账单总费用 = 存储费用单价 × 使用时长 + 请求费用单价 x 使用规格 + 流量费用单价 x 使用规格 + 数据处理费用单价 x 使用规格。
- **资源包计费方式**：您先提前购买相应的资源包，BOS 会在每个计费周期内按照资源使用量扣除资源包额度。

> **说明：**
>
> - 如需了解两种计费形式下，BOS 的计费类别和目录价格时，请参考 BOS 计费类别及价格清单。
> - 如需了解按需计费形式下 BOS 的计量方式和账单周期等详情介绍，请参考 BOS 按需计费方式说明。
> - 如需了解 BOS 预付费资源包的购买和使用说明，请参考 BOS 资源包类型和使用方式说明。
> - 如需预估 BOS 的使用费用，请进入 BOS 价格计算器 进行计算。
> - 当您开通 BOS 服务并上传数据后，若不购买预付费资源包，则系统会默认按照按量计费的形式进行计费。若您开通 BOS 服务后未进行任何操作，BOS 不会向您收取费用。
> - BOS 全部的计费项采用按分钟推量计费的方式进行计费，并每小时对相关费用进行出账，您可以在控制台 "财务-账单明细"中查看详细的流水和资源账单。

费用组成

BOS 服务的计费项由4个费用模块组成，各费用模块包含的计费项信息见存储费用、流量费用、请求费用和数据处理费用。



存储费用

存储费用为 BOS 的基本费用。当您在 BOS 中上传文件后会产生存储费用，即容量占用空间产生的费用。低频存储、冷存储和归档存储类型文件具有最低存储周期，若未存满相应的最小存储周期，BOS 一次性收取最小存储周期费用。

类别	子计费项	计费方式	购买方式
容量占用空间	标准存储-多AZ费用	• 当您存储在 BOS 中的文件选择"标准存储-多AZ"类型进行存储时，系统会每小时结算一次该文件所占用的标准存储-多AZ 空间（GB）费用。	按需付费
	标准存储费用	• 当您存储在 BOS 中的文件选择"标准存储类型"进行存储时，系统会每小时结算一次该文件所占用的标准存储空间（GB）费用。	按需付费、资源包计费
	低频存储-多AZ存储费用	• 当您存储在 BOS 中的文件选择"低频存储-多AZ"类型进行存储时，系统会每小时结算一次该文件所占用的低频存储-多AZ 空间（GB）费用。	按需付费

	低频存储费用	• 当您存储在 BOS 中的文件选择"低频存储类型"进行存储时，系统会每小时结算一次该文件所占用的低频存储空间（GB）费用。	按需付费、资源包计费
	冷存储费用	• 当您存储在 BOS 中的文件选择"冷存储类型"进行存储时，系统会每小时结算一次该文件所占用的冷存储空间（GB）费用。	按需付费、资源包计费
	归档存储费用	• 当您存储在 BOS 中的文件选择"归档存储类型"进行存储时，系统会每小时结算一次该文件所占用的归档存储空间（GB）费用。	按需付费、资源包计费
最小存储周期	低频存储-多AZ-最小存储周期费用	• 低频存储-多AZ 类型文件有 30 天最小存储时间要求。若文件在未存满 30 天时被删除/覆盖写，会产生一笔最小存储周期费用，BOS 将一次性收取剩余未存满时间内的容量费用。举例：若您在文件上传 50 小时后将该低频存储-多AZ 类型文件删除，那么最小存储周期的费用为：低频存储-多AZ容量月单价/30/24*（30*24-50）元。 • BOS 对低频存储-多AZ 剩余存储时间的精度会精确到小时级，您在账单中看到该项费用的单位为 Bytes*小时，且会自动向上进位到 GB*小时/TB*小时/PB*小时。剩余时间不足 1 小时的费用将为您免除。 • 因为同时可能有很多不同大小的文件被删除/覆盖写，且每个文件的剩余时间可能不同，因此账单中该项的展示方式为聚合展示，即将所有文件的"容量*小时数"进行累加。举例：客户同时删除了 2 个文件，一个 1GB 且剩余时长为 30 小时，另一个 2GB 且剩余时长为 40 小时。那么这两个文件的最小存储周期费用分别是 1GB*30 小时和 2GB*40 小时，在账单中聚合展示为 1*30+2*40=110GB*小时。	按需一次性付费，系统自动出账
	低频存储-最小存储周期费用	• 低频存储类型文件有 30 天最小存储时间要求。若文件在未存满 30 天时被删除/覆盖写，会产生一笔最小存储周期费用，BOS 将一次性收取剩余未存满时间内的容量费用。举例：若您在文件上传 50 小时后将该低频存储类型文件删除，那么最小存储周期的费用为：低频存储容量月单价/30/24*（30*24-50）元。 • BOS 对低频存储剩余存储时间的精度会精确到小时级别，您在账单中看到该项费用的单位为 Bytes*小时，且会自动向上进位到 GB*小时/TB*小时/PB*小时。剩余时间不足 1 小时的费用将为您免除。 • 因为同时可能有很多不同大小的文件被删除/覆盖写，且每个文件的剩余时间可能不同，因此账单中该项的展示方式为聚合展示，即将所有文件的"容量*小时数"进行累加。举例：客户同时删除了 2 个文件，一个 1GB 且剩余时长为 30 小时，另一个 2GB 且剩余时长为 40 小时。那么这两个文件的最小存储周期费用分别是 1GB*30 小时和 2GB*40 小时，在账单中聚合展示为 1*30+2*40=110GB*小时。	按需一次性付费，系统自动出账
	冷存储-最小存储周期费用	• 冷存储类型文件有 60 天最小存储时间要求。若文件在未存满 60 天时被删除/覆盖写，会产生一笔最小存储周期费用，BOS 将一次性收取剩余未存满时间内的容量费用。举例：若您在文件上传 50 小时后将该低频存储类型文件删除，那么最小存储周期的费用为：冷存储容量月单价/30/24*（60*24-50）元。 • BOS 对冷存储剩余存储时间的精度会精确到小时级别，您在账单中看到该项费用的单位为 Bytes*小时，且会自动向上进位到 GB*小时/TB*小时/PB*小时。剩余时间不足 1 小时的费用将为您免除。 • 因为同时可能有很多不同大小的文件被删除/覆盖写，且每个文件的剩余时间可能不同，因此账单中该项的展示方式为聚合展示，即将所有文件的"容量*小时数"进行累加。举例：客户同时删除了 2 个文件，一个 1GB 且剩余时长为 30 小时，另一个 2GB 且剩余时长为 40 小时。那么这两个文件的最小存储周期费用分别是 1GB*30 小时和 2GB*40 小时，在账单中聚合展示为 1*30+2*40=110GB*小时。	按需一次性付费，系统自动出账
		• 归档存储类型文件有 180 天最小存储时间要求。若文件在未存满 180 天时被删除/覆盖写，会产生一笔最小存储周期费用，BOS 将一次性收取剩余未存满时间内的容量费用。举例：若您在文件上传 3 天后将该归档存储类型文件删除，那么最小存储周期的费用为：归档存储容量月单价/30*（180-3）元。	

	归档存储-最小存储周期费用	• BOS 对归档存储剩余存储时间的精度会精确到天级别，您在账单中看到该项费用的单位为 Bytes*天，且会自动向上进位到 GB*天/TB*天/PB*天。剩余时间不足 1 天的费用将为您免除。 • 因为同时可能有很多不同大小的文件被删除/覆盖写，且每个文件的剩余时间可能不同，因此账单中该项的展示方式为聚合展示，即将所有文件的"容量*天数"进行累加。举例：客户同时删除了 2 个文件，一个 1GB 且剩余时长为 30 天，另一个 2GB 且剩余时长为 40 天。那么这两个文件的最小存储周期费用分别是 1GB*30 天和 2GB*40 天，在账单中聚合展示为 1*30+2*40=110GB*天。	按需一次性付费，系统自动出账

注意：

- 归档存储类型文件若不足 64 KB，则按照 64 KB 进行计费；如文件大小大于等于 64 KB，则按照文件实际大小进行计费。

流量费用

流量费用为 BOS 的基本费用。当您访问 BOS 中存储的文件时，根据不同的使用情况会产生流量费用。

类别	子计费项	计费方式	购买方式
流量费用	外网流出流量	外网数据数据流出，即您在使用 BOS 时，通过互联网（非百度内网） IP 流出的流量（不包含CDN回源产生的流量），如使用互联网将文件下载到本地或分享给他人，系统会每小时结算一次外网数据流出费用。	按需付费、资源包计费
	外网流入流量	免费	-
	内网流出流量	免费	-
	内网流入流量	免费	-
	CDN 回源请求产生的流出流量	指向百度智能云 CDN 回源时产生的费用。其他非百度智能云 CDN 回源无法识别，按照标准外网数据流出收费。系统会每小时结算一次 CDN 回源流出费用。	按需付费、资源包计费
	跨区域流出流量	若您使用文件的跨区域复制，如将位于华北-北京地域的的文件跨区域同步至华南-广州区域，则会产生跨区域流出流量费用。若您使用跨账号同步功能，且源 Bucket 和目的 Bucket 处于两个不同的地域，则跨区域流量费用由源 Bucket 所属账号承担。系统会每小时结算一次跨区域复制数据流出费用。	按需付费
	CDN 上行加速	当您开启 BOS 的 CDN 上行加速功能时，BOS 不会收取费用（即您 BOS 的账单中不包含该项费用），但百度智能云 CDN 服务会收取相应的费用。BOS 的 CDN 上行加速是基于百度智能云 CDN 动态加速功能而研发的专门针对 BOS 上行场景的功能。若您开启该功能，该功能相应的费用将包含在您的 CDN 账单中，详细的计费信息请查看 CDN 动态加速计费文档。	按需付费、CDN 动态加速 PV 包

请求费用

请求费用为 BOS 的基本费用。当您访问 BOS 中存储的文件时，根据不同的使用情况会产生请求费用，包括读请求费用、写请求费用和数据取回费用

类别	子计费项	计费方式	购买方式
请求费用	读请求费用	您所有对 BOS 的操作实际上都是对 BOS API 的调用。当您读取 BOS 中数据时会调用 BOS 读相关的 API 如 GET 等，将产生读请求费用。BOS 控制台操作也是通过 BOS API 调用实现的，因此使用 BOS 控制台也会产生读请求次数。系统会每小时结算一次读请求费用。产生读请求费用的API包括但不限于GetBucketLocation、HeadBucket、GetObject、GetObjectMeta、ListParts、GetSymlink、SelectObject、GetObjectAcl。	按需付费
	写请求费用	您所有对 BOS 的操作实际上都是对 BOS API 的调用。当您向 BOS 中写入数据时会调用 BOS 写相关的 API 如 PUT 等，将产生写请求费用。BOS 控制台操作也是通过 BOS API 调用实现的，因此使用 BOS 控制台也会产生写请求次数。系统会每小时结算一次写请求费用。产生写请求费用的API包括但不限于PutObject、PostObject、InitiateMultipartUpload、UploadPart、CompleteMultipartUpload、AbortMultipartUpload、AppendObject、DeleteObject、DeleteMultipleObjects、FetchObject、Putsymlik。	按需付费
	归档存储取回请求费用	当您通过 API/SDK/BOS官方工具/控制台取回归档存储类型文件时，BOS 均会调用 RetoreObject 接口，此时会产生归档存储取回请求费用。归档存储取回共有加急取回、标准取回、延缓取回 3 种取回模式，每种取回模式对应的请求费用不同。低频存储、低频存储-多 AZ 和冷存储取回不产生取回请求费用。系统会每小时结算一次归档存储取回请求费用。	按需付费
数据取回费用	低频存储-多AZ取回费用	当您在 BOS 中存储的文件选择了低频存储-多 AZ 类型，且当您需要对该文件进行读操作时，需要先将该文件取回，该取回操作会产生数据取回费用。当您读取或者下载低频存储-多 AZ 类型文件时，取回操作自动触发，系统会自动产生取回费用，并生成小时账单。	按需付费
	低频存储取回费用	当您在 BOS 中存储的文件选择了低频存储类型，且当您需要对该文件进行读操作时，需要先将该文件取回，该取回操作会产生数据取回费用。当您读取或者下载低频存储类型文件时，取回操作自动触发，系统会自动产生取回费用，并生成小时账单。	按需付费
	冷存储取回费用	当您在 BOS 中存储的文件选择了冷存储类型，且当您需要对该文件进行读操作时，需要先将该文件取回，该取回操作会产生数据取回费用。当您读取或者下载冷存储类型文件时，取回操作自动触发，系统会自动产生取回费用，并生成小时账单。	按需付费
	归档存储取回费用	当您在 BOS 中存储的文件选择了归档存储类型，且当您需要对该文件进行读操作时，需要先将该文件取回，该取回操作会产生数据取回费用。归档存储文件取回需要您通过 API/SDK/BOS官方工具/控制台手动触发，取回成功后才进行访问或者下载。归档存储取回有三种模式，分别为加急取回、标准取回和延缓取回，每种取回模式费用不同。BOS 每小时生成一次账单，您可以在账单中查看每种取回模式的费用。	按需付费
	归档存储取回缓存费用	当您在 BOS 中存储的文件选择了归档存储类型，且当您需要对该文件进行读操作时，需要先将该文件取回。触发取回操作时，您需要设置文件取回副本的缓存时间，该副本文件将会产生相应缓存费用。如您设置副本缓存 10 天，那么该文件一次性收取 10 天的缓存费用。BOS 每小时生成一次账单，您可以在账单中查看每种取回模式的费用。	按需付费

> **注意：**
>
> - 归档存储取回请求费用、归档存储加急取回、归档存储延缓取回、归档存储取回缓存费用，将在 **2022 年 5 月 20 日 - 2022 年 5 月 25 日间陆续在华南-广州、华东-苏州、华北-保定和华北-北京区域上线，请您及时关注账单费用**。

数据处理费用

BOS 的数据处理费用包含 Select 扫描，图像处理，视频处理-视频截帧，文档预览，清单功能，数据访问趋势分析。

数据处理计费方式详见下表，按需计费请参见“产品定价”>“产品价格”>“按需计费方式”。

类别	子计费项	计量方式	计费方式	
图像处理费用	基础图像处理费用	图像基础处理费用是指对存储在 BOS 中的图像进行缩放、裁剪、旋转、格式转换、渐进显示、质量变换、水印等基础图片处理操作时产生的费用。BOS 会根据处理前的图片大小进行计费。BOS 为每位用户每月免费赠送 10 TB 的图像基础处理额度。当您当月处理的图片量大于 10 TB 时，将按超出部分的使用量进行计费。该项费用会按小时进行出账。	每月（自然月）赠送 10TB 处理量，超出部分按需付费	
	HEIC 图像压缩费用 (800x600 以下规格)	属于图像高级处理服务产生的费用。若您调用 BOS 的 HEIC 图像压缩服务，且处理的原图尺寸为 800x600 以下规格，会产生该项费用。该项费用会按小时进行出账。	按需计费	
	HEIC 图像压缩费用 (800x600 以上规格)	属于图像高级处理服务产生的费用。若您调用 BOS 的 HEIC 图像压缩服务，且处理的原图尺寸为 800x600 以上规格，会产生该项费用。该项费用会按小时进行出账。	按需计费	
	WebP-M6 图像压缩费用(800x600以下规格)	属于图像高级处理服务产生的费用。若您调用 BOS 的 WebP-M6 图像压缩服务，且处理的原图尺寸为 800x600 以下规格，会产生该项费用。该项费用会按小时进行出账。	按需计费	
	WebP-M6 图像压缩费用(800x600以上规格)	属于图像高级处理服务产生的费用。若您调用 BOS 的 WebP-M6 图像压缩服务，且处理的原图尺寸为 800x600 以上规格，会产生该项费用。该项费用会按小时进行出账。	按需计费	
	智能效果-智能去雾	属于图像智能效果服务产生的费用。若您调用智能去雾，可对浓雾天气下拍摄的细节无法辨认的图像进行去雾处理，还原更清晰真实的图像。该计费项按照调用次数进行收费，按小时出具账单。	按需计费	
	智能场景-风格转换	属于图像智能场景服务产生的费用。若您调用风格转换，可将图像转化成卡通画、铅笔画、彩色铅笔画，或者哥特油画、彩色糖块油画、呐喊油画、神奈川冲浪里油画、奇异油画、薰衣草油画等共计 9 种风格，可用于开展趣味活动，或集成到美图应用中对图像进行风格转换。该计费项按照调用次数进行收费，按小时出具账单。	按需计费	
	智能场景-人像动漫画	属于图像智能场景服务产生的费用。人像动漫画技术运用世界领先的对抗生成网络，结合人脸检测、头发分割、人像分割等技术，为用户量身定制千人千面的二次元动漫形象，并且可通过参数设置，生成戴口罩的二次元动漫人像。该计费项按照调用次数进行收费，按小时出具账单。	按需计费	
	智能场景-物体擦除	属于图像智能场景服务产生的费用。物体擦除可以去除图片中不需要的遮挡物，并用背景内容填充，提高图像质量。该计费项按照调用次数进行收费，按小时出具账单。	按需计费	
	智能场景-拉伸修复	属于图像智能场景服务产生的费用。拉伸修复可以自动识别过度拉伸的图像，将图像内容恢复成正常比例。该计费项按照调用次数进行收费，按小时出具账单。	按需计费	
	智能场景-黑白上色	属于图像智能场景服务产生的费用。黑白上色可以智能识别黑白图像内容并填充色彩，使黑白图像变得鲜活。该计费项按照调用次数进行收费，按小时出具账单。	按需计费	
	智能场景-无损放大	属于图像智能场景服务产生的费用。无损放大可对输入的图片在尽量保持图像质量的条件下，将图像在长宽方向各放大两倍。该计费项按照调用次数进行收费，按小时出具账单。	按需计费	
	Select扫描费用	Select扫描费用	Select 扫描是 BOS 提供的一项针对于大数据场景中的增值服务。若您调用了 BOS Select 相关的 API 或通过控制台对文件发起 Select 扫描操作，将产生 Select 扫描费用。系统将根据您扫描文件的大小（GB）对该操作收取费用，并按小时生成账单。	按需付费
视频截帧费用	视频截帧费用	视频截帧是BOS提供的针对视频文件智能截取单帧图片的增值服务。若您调用了BOS视频截帧API，将产生对应费用。系统将根据您成功截取的图片张数收取费用，并按小时生成账单。	按需付费	
文档预览费用	文档转图片费用	指使用文档预览服务时，采用文档转图片功能产生的费用。若您调用了 BOS 文档预览 API ，将产生对应费用。系统将根据您成功转换的图片张数收取费用，并按小时生成账单。	按需付费	
	文档转HTML费用	指使用文档预览服务时，采用文档转 HTML 功能产生的费用。若您调用了 BOS 文档预览 API ，将产生对应费用。系统将根据您成功转换 HTML 的次数收取费用，并按小时生成账单。	按需付费	
存储桶清单	清单功能费用	根据用户的清单任务配置，定期每天或者每周扫描用户 Bucket 中指定对象并输出一份CSV 格式的清单报告文件，其中会列出对象及其对应的元数据信息。系统按照存储桶清单功能列出的对象（文件）的个数计费，并按小时生成账单。	按需付费	
数据访问趋势分析	数据访问趋势分析费用	根据用户的数据访问趋势分析任务的配置，定期分析指定 Bucket 或前缀中的对象，并输出分析结果的图表内容。系统按照数据访问趋势分析功能所分析的对象（文件）个数计费，并按小时生成账单。	按需付费	

说明：

- Select 扫描计费项从 2020 年 3 月 31 日起生效；

- 图像基础处理、图像高级处理中 HEIC 图像压缩及 WebP-M6 图像压缩服务于 2020 年 6 月 1 日开始正式计费。您可以在 BOS 控制台全局概览页中查看图像处理用量数据。
- 视频截帧计费项于 2022年 4 月 1 日开始正式计费。您可以在 BOS 控制台全局概览页中查看单个图像处理用量数据。
- 文档预览计费项于2022年 9 月 20 日开始正式计费。
- 清单功能计费项于2022年 12 月 8日正式计费。

产品价格

按需计费方式

概述

若您未购买 BOS 的预付费资源包，BOS 默认采用后付费方式对您各计费项的使用量进行实时计费。BOS 对部分计费项提供 预付费资源包，若您购买了预付费资源包，对应的服务费用将从预付费资源包中抵扣。若您开通 BOS 服务后未进行操作，BOS 不会产生费用。关于 BOS 的计费项及计费项的含义，请您查看计费项概览。

计费方式

- **计费公式**：BOS 总费用 = 存储费用的单价 x 使用时长 + 请求费用的单价 x 使用规格 + 流量费用单价 x 使用规格 + 数据处理费用单价 x 使用规格。
- **计量周期**：BOS 按需计费采用**分钟级别推量结算**的方式进行计费，如“标准存储容量费用 = 每分钟标准存储容量单价✖每分钟标准存储容量”。若您使用不足 1 分钟，则按 1 分钟计费。官网价格中的容量月单价仅为 30 天的参考价，并非最终的自然月单价。若遇到 31 天、28 天或者 29 天的情况，月单价会有所变化。

> **举例说明**
>
> - 如官网标定的标准存储月容量单价为 0.119 元/GB/月，那么天容量单价为 0.119/30=0.00396667元/GB/天，小时容量单价为 0.119/30/24=0.000165278元/GB/小时，分钟容量单价为 0.119/30/60=0.0000027546元/GB/分钟。若您在该分钟内的容量峰值为 10987654321 Bytes（约10.23GB），那么该容量的分钟级别账单金额为 10987654321/1024/1024/1024✖0.0000027546=0.00002818796 元。若在 30 天的月份，月账单金额为 0.00002818796 ✖ 60 ✖ 24 ✖ 30 = 1.22 元；若在 31 天的月份，月账单金额为 0.00002818796 ✖ 60 ✖ 24 ✖ 31 = 1.26 元；若在 28 天的月份，月账单金额为 0.00002818796 ✖ 60 ✖ 24 ✖ 28 = 1.10 元。因此，BOS 采取分钟级别的精准计费方式进行出账，官网的容量价格仅为 30 天的参考价。

- **账单周期**：BOS **按小时**对您的使用费用进行出账并扣费，即对过去 60 分钟内的分钟级别账单进行加和，出具小时账单。出账单时间是当前计费周期结束后 1 小时左右，如 10:00 - 11:00 的账单会在 12:00 左右生成，具体以系统出账时间为准;
- **停止服务**：按需计费模式下，您需要保证账户无欠款，否则系统会将您的 BOS 服务停服；

按需服务价格

BOS 按需服务的详细价格清单，请查看对象存储 BOS 价格清单 ，或进入 BOS 价格计算器进行计算。

按需价格清单

存储价格

存储价格

BOS 有标准存储-多AZ、标准存储、低频存储-多AZ、低频存储、冷存储和归档存储 6 种存储类型。每种存储类型在不同的地域单价不同，详情请参见"对象存储BOS价格说明"->“按需计费方式”->“存储价格”。

> **说明**
>
> **低频存储、冷存储和归档存储在读取和使用场景上会有一定限制**，如存在最小存储周期、数据取回费用等，请在购买前先查看分级存储，进一步不同存储类型。

请求价格

请求费用

BOS 请求费用包含读写请求费用和数据取回费用。

读写请求费用

当您通过 BOS 控制台/SDK/官方工具/API 调用 BOS 时，均会产生读写请求费用。

读写请求费用的详细价格信息，请参见"对象存储BOS价格说明"->“按需计费方式”->“请求价格”->“读写请求费用”。

> **说明**：
>
> 写请求包括 API 类型为 PUT, COPY, DELETE 等在内的全部写请求，读请求包括 GET 在内的全部读请求。

归档存储取回请求费用

当您取回归档存储类型文件时，会产生归档存储取回请求费用。

归档存储取回请求费用的详细价格信息，请参见"对象存储BOS价格说明"->“按需计费方式”->“请求价格”->“归档存储取回请求费用”。

> **说明**：
>
> - 归档存储取回请求费用**预计将于 2022 年 5 月 1 日起开始收取，当前暂不收取**；
> - 归档存储**当前仅支持标准取回，加急取回和延缓取回即将上线，敬请期待**。关于归档存储不同取回模式的取回性能，请查看分级存储；
> - 归档存储请求指 RestoreObject 请求。

数据取回费用

低频存储-多 AZ、低频存储、冷存储、归档存储文件在被访问前均需要先取回。其中，归档存储取回需要您手动触发取回，而低频存储-多 AZ、低频存储、冷存储在您访问时自动触发取回操作，无需单独调用 API，您直接读取和访问文件即可。标准存储和标准存储-多 AZ 不需要取回，无取回费用。

数据取回费用的详细价格信息，请参见"对象存储BOS价格说明"->“按需计费方式”->“请求价格”->“数据取回费用”。

> **说明**：
>
> - 当前，**归档存储仅支持标准取回，加急取回和延缓取回即将上线，敬请期待**。关于归档存储不同取回模式的取回性能，请查看分级存储；
> - 只有低频存储-多AZ、低频存储、冷存储和归档存储会收取数据取回费用，其他存储类型不收取。该费用不分内外网；
> - 如果您的数据每月访问次数小于 1 建议使用低频存储；每年访问次数小于 1 次建议使用冷存储；每三年访问次数小于 1 次建议使用归档存储。

流量价格

流量费用

不同存储类型的详细价格信息，请参见“对象存储BOS价格说明”->“按需计费方式”->“流量价格”。

> **说明**：
>
> CDN 上行加速是 BOS 基于 CDN 动态加速的一项功能。BOS 对该功能不收取费用，但是 CDN 会产生费用，详情价格请参考 CDN 动态加速费用。

数据处理价格

数据处理费用

BOS 的数据处理费用包含 Select 扫描，图像处理，视频处理-视频截帧，文档预览，清单功能。

数据处理的详细价格，请参见“对象存储BOS价格说明”->“按需计费方式”->“数据处理费用”。 **Select扫描费用**

BOS Select 功能支持在大数据场景中对 CSV/JSON 文件中的内容进行检索扫描，适用于日志处理，数据处理等大数据企业和个人客户。

Select 扫描计费项从 2020 年 3 月 31 日起生效。

图像处理费用

BOS 图像处理包括图像基础处理费用和图像高级处理费用。图像基础处理费用是指对存储在 BOS 中的图像进行缩放、旋转、水印、格式变换、裁剪、智能对比度、智能清晰度等操作时产生的费用，图像高级处理费用是指对存储在 BOS 中的图像进行 HEIC 压缩，WebP-M6 压缩等高阶操作。

图像基础处理、图像高级处理中 HEIC 图像压缩及 WebP-M6 图像压缩服务于 2020 年 6 月 1 日开始正式计费。您可以登录控制台-财务查看用量数据。

视频截帧费用 视频截帧，支持通过设置url参数的方法对存储在BOS中的视频文件进行截取生成图片。

视频截帧计费项于 2022年 4 月 1 日开始正式计费。您可以在 BOS 控制台全局概览页中查看单个存储桶的视频截帧用量数据。

文档预览费用 文档预览，基于百度智能云文档服务 DOC（Document Service），为开发者提供 Office、WPS 等多种格式的文档存储、管理及在线浏览服务。您无需了解文档存储、转码、分发、在线浏览等技术细节，即可快速搭建安全可靠、高可定制的文档处理平台和应用，助力在线教育、企业网盘等业务的转型升级。

文档预览支持您将文档文件转化为HTML或图片可实现文档预览的效果，调用该功能产生的文档预览计费项于2022年 9 月 20 日开始正式计费。

清单功能费用 存储桶清单功能，帮助用户管理存储桶中对象。BOS 可根据用户的清单任务配置，定期每天或者每周扫描用户 Bucket 中指定对象并输出一份清单报告，以 CSV 格式文件的形式存储到用户指定的 Bucket 中。CSV 文件中会列出对象及其对应的元数据，并根据用户的配置信息，记录用户所需的对象属性信息。

该功能产生的清单费用于2022年 12 月 8日正式计费。

资源包价格清单

BOS提供以下6种类型的资源包，详细的价格信息请参见“对象存储BOS价格说明”->“按资源包计费”。

资源包类型	地域	说明
标准存储包	华北-北京 华北-保定 华东-苏州 华南-广州 华中金融-武汉 中国香港	不同地域售卖的资源包，价格会有所不同
低频存储包	华北-北京 华北-保定 华东-苏州 华南-广州 华中金融-武汉 中国香港	不同地域售卖的资源包，价格会有所不同
冷存储包	华北-北京 华北-保定 华东-苏州 华南-广州 华中金融-武汉	不同地域售卖的资源包，价格会有所不同
归档存储包	华北-北京 华北-保定 华南-广州	NA
外网下行流量包	华北-北京 华北-保定 华东-苏州 华南-广州 中国香港 华中金融-武汉	目前提供 1 月包、3 月包、半年包和一年包共4种规格
回源流量包	华北-北京 华南-广州 华东-苏州 华北-保定 中国香港 华中金融-武汉地	1、BOS 流量包目前只有 1 个月一种时长，后续会逐步支持更多时长，敬请期待 2、BOS 流量包的流量额度到期自动清零。

资源包计费方式

资源包概览

概述

BOS 支持包年包月（预付费）形式的资源包购买。购买资源包后，您的 BOS 消费会首先用资源包进行抵扣，超出部分按量付费。您可以直接在百度智能云管理控制台的"BOS->全局概览->资源包购买"页签进行资源包购买，购买时按照提示选择资源包类型、区域、规格、购买时长和生效时间即可。

资源包类型

目前，BOS 提供 9 种类型的资源包：

- **标准存储包**：用于抵扣使用对象存储时产生的容量，**仅**适用于**标准存储类型产生的容量费用**，不能抵扣标准存储-多AZ类型，也不能抵扣其他计费项账单金额；
- **低频存储包**：用于抵扣使用对象存储时产生的容量，**仅**适用于**低频存储类型产生的容量费用**，不能抵扣低频存储-多AZ类型，也不能抵扣其他计费项账单金额；
- **冷存储包**：用于抵扣使用对象存储时产生的容量，**仅**适用于**冷存储类型产生的容量费用**，不能其他类型，也不能抵扣其他计费项账单金额；
- **归档存储包**：用于抵扣使用对象存储时产生的容量，**仅**适用于**归档存储类型产生的容量费**用，不能抵扣其他计费项账单金额；
- **外网下行流量包**：用于抵扣外网流出流量，**仅**适用于所有存储类型产生的**外网下行流量费用**，不能抵扣其他类型计费项账单金额；
- **回源流量包**：用于抵扣使用 CDN 加速域名访问 BOS 资源时产生的回源流量，不区分存储类型。
- **基础图像处理包**：用于抵扣图片裁剪、旋转、缩放、水印等基础图像处理所产生的用量费用，不能抵扣高级压缩处理、智能处理等产生的费用，也不能抵扣其他计费项账单金额；
- **图像高级压缩包**：用于抵扣HEIC和WebP-M6格式图片高级压缩所产生的用量费用，不能抵扣其他图像处理类型产生的费用，也不能抵扣其他计费项账单金额；
- **图像智能处理包**：用于抵扣图像智能处理所产生的用量费用，不能抵扣其他图像处理类型产生的费用，也不能抵扣其他计费项账单金额。

资源包购买说明

（1）资源包为预付费购买方式，需一次性支付资源包费用；

（2）资源包购买后无法转换类型。如您购买了标准存储资源包，该资源包无法变更为低频存储包，请您在购买前务必明确要使用的存储类型或流量类型。

资源包使用说明 BOS各个类型资源包的抵扣规则，请查看资源包抵扣规则。 **资源包价格** BOS资源包的详细价格清单，请查看对象存储 BOS 价格清单。您可通过BOS价格计算器进行费用测算。

资源包抵扣规则

资源包抵扣总体规则

（1）**仅能抵扣相应计费项的账单费用**：

- **若您购买了容量包**：您购买的容量包只能抵扣对应存储类型的容量费用，不能抵扣其他计费项费用，如外网下行流量费用、读写请求次数费用、数据取回费用、数据处理费用等。如您购买了"标准存储容量包"，该资源包只能抵扣标准存储容量费用，不能抵扣标准存储读写请求次数费用；如您购买了"低频存储容量包"，该资源包只能抵扣低频存储容量费用，不能抵扣低频存储读写请求次数费用，也不能抵扣低频存储数据取回费用。
- **若您购买了流量包**：您购买的流量包只能抵扣相应流量类型产生账单金额。BOS 流量费用不区分存储类型。如您购买了"外网下行流量费用"，则只能抵扣外网下行流量费用，无法抵扣其他流量费用如 CDN 回源流量费用、跨区域流量费用等。
- **若您购买了图像处理包**：您购买的图像处理包只能抵扣相应图像处理功能产生的账单金额。图像处理类型包括基础图像处理功能（裁剪、旋转、缩放、水印等）、图像高级压缩功能、图像智能处理功能，各功能分别对应各自的图像处理包，如您购买了"图像智能处理包"，则只能抵扣图像智能处理产生的用量费用，无法抵扣其他图像处理费如基础图像处理功能、图像高级压缩功能等。

> **注意**：BOS 全部计费项默认均为后付费的计费方式，若您只购买了存储容量包，请您务必同时保证账户余额充足，以足以抵扣其他计费项的费用。关于其他计费项，请查看 BOS 计费项介绍;

(2)**资源包具有地域属性**：您购买的容量包、流量包、图像处理包具有地域属性，您需要购买与被抵扣资源处于同一地域的资源包。若购买的资源包与被抵扣资源处于不同地域，则无法进行抵扣。若您在华北-北京区域创建了 Bucket 并进行数据存储，当您希望购买容量包进行预付费抵扣时，您需要购买华北-北京区域的容量包，否则北京区域的容量账单无法产生抵扣；

(3) **退款规则**：您购买资源包在 7 天内且未使用（即设置的生效时间在将来）可全额退款，需提交工单申请退款；若购买资源包在 7 天内且已使用（即资源包已生效），该资源包不支持退款；若您购买时间超出 7 天，该资源包不支持退款；

(4) **到期提醒**：到期资源包立即失效，到期前7天、3天、1天和到期时 BOS 会给您发送提醒消息，请您注意查收短信邮件站内信等通知；

(5) **续费规则**：未到期资源包支持续费，续费时可更改资源包大小和时间，但不可更改原使用区域，也不可指定续费包的生效时间。续费成功后，该资源包可在原生效日期到期后继续使用。已到期和已用尽的资源包不支持续费，若有需要请新购资源包。

> **特殊说明：**
>
> - 资源包仅限本账号使用；
> - 您可以同时购买多个资源包，且同一时间段可以有多个资源包处于生效状态；
> - 资源包仅支持在购买时指定的地域产生对应计费项的抵扣；
> - 若您设置的生效时间早于下单时间，则从下单时间开始生效。

容量包抵扣细节说明

1、什么是容量包 容量包是指可以抵扣存储容量的资源包，如"标准存储容量包"、"低频存储容量包"、"冷存储容量包"和"归档存储容量包"。

2、容量包抵扣规则 在资源包生效期间，您使用相应存储类型所产生的容量费用会优先使用资源包抵扣，超出部分会采用后付费按量付费。为避免账户欠费导致停服，请您及时关注存储容量变化。

> - **以标准存储包为例，抵扣规则为：**
> - **情况1**：用户当前有 2 个 100 GB 标准存储包处于生效状态，且当前"标准存储-占用空间"为 180 GB，由于 180 GB < 200 GB，用量在存储包规格之内，所以该计费项以预先购买的存储包抵扣即可；
> - **情况2**：用户当前有 2 个 100 GB 标准存储包处于生效状态，当前的"标准存储-占用空间"为 250 GB，由于 250 GB > 200 GB，所以 200 GB 费用由资源包抵扣，超出的 50 GB 按量付费。若您此时账户余额不足以抵扣 50 GB 的容量费用，您的账户将变为欠费状态。若您超过 15 天未充值，您的数据都将被释放。

3、容量包生效时间 当选择"支付后立即生效"时，如果容量包是在某一个小时中间购买并为立即生效，那在下个整点账单时，容量包会按照时间比例抵扣，不会完整抵扣。当选择"指定时间生效"时，可选指定时间为从当前时间到最长在未来5年内的任意时间，精确到分钟粒度。

外网下行流量包抵扣细节说明

1、流量包抵扣规则 BOS 提供外网下行流量资源包，您可以根据自身需求购买不同时间规格，如 1 月规格、6 月规格等。若您购买了多月流量包（时间规格非 1 个月），请您注意：

- **多月流量包**是指每个整月均享有一定大小流量，当月用完后用量自动清零，下个整月仍享有相同额度，以此类推。当您当月流量额度用尽后，该流量包不会失效，但当月超出额度的流量不再从流量包中抵扣，会产生外网下行流量后付费账单；从次月（整月非自然月）开始，产生的外网下行流量将继续从流量包中进行抵扣。

> 如您在 2020 年 10 月 8 日购买华北-北京地域的 100 GB 3 月包，且流量包在购买后立即生效，那么：
>
> - 2020 年 10 月 8 日 - 2020 年 11 月 7 日有 100 GB 外网下行流量额度，本月超出 100 GB 部分产生后付费账单；2020 年 11 月 8 日 - 2020 年 12 月 7 日有 100 GB 外网下行流量额度，本月超出 100 GB 部分产生后付费账单；2020 年 12 月 8 日 - 2021 年 1 月 7 日有 100 GB 外网下行流量额度，本月超出 100 GB 部分产生后付费账单；
> - 而不是 2020 年 10 月 8 日 - 2021 年 1 月 7 日共有 100 GB 外网下行流量额度。

2、流量包生效时间 当选择“支付后立即生效”时，如果流量包是在某一个小时中间购买并为立即生效，那在下个整点账单时，流量包会按照时间比例抵扣，不会完整抵扣。当选择“指定时间生效”时，可选指定时间为从当前时间到最长在未来 5 年内的任意时间，精确到分钟粒度。

图像处理包抵扣细节说明 1、什么是图像处理包 图像处理包是指可以抵扣图像处理费用的资源包，如“基础图像处理包”、“图像高级压缩包”和“图像智能处理包”。 **2、基础图像处理包抵扣规则** 基础图像处理包可选购买时长为1个月、6个月、12个月，可选的购买规格为每月 10TB、20TB、50TB 。规格包自生效日期开始，连续 n 个月发放，月底清零（自然月份）。此外，BOS为您每月提供免费10TB用量，当您购买的基础图像处理包用尽后该免费用量生效。

> 例如：您在 2020 年 10 月 25 日购买华北-北京地域的 20TB 6月包，且基础图像处理包在购买后立即生效，那么：2020 年 10 月 25 日 - 2020 年 10 月 31 日有 20TB 基础图像处理用量额度，当月超出 20TB 后开始抵扣免费用量，抵扣完 10TB 的免费用量后，超出的部分产生后付费账单；此后2020 年 11 月 - 2021 年 3 月这 5 个月的每个月均拥有 20TB 的基础图像处理用量额度，当月超出 20TB 后开始抵扣免费用量，抵扣完 10TB 的免费用量后，超出的部分产生后付费账单。

3、图像高级压缩包和图像智能处理包抵扣规则 图像高级压缩包和图像智能处理包以万虚拟次为单位购买，有效时长为一年，即购买后从生效日期算起有效期为一年，逾期无法抵扣。图像高级压缩包的购买规格为每月 10 万次 - 5000 万次之间的 9 个档位，图像智能处理包的购买规格为每月 10万次、20万次、50万次 。每次进行特定的图像处理后会从虚拟资源包中扣除的相应的虚拟次数，进行一次图像处理后扣除的虚拟次数被称为该计费项的抵扣因子。 **4、抵扣因子说明** 抵扣因子的计算规则为：抵扣因子 = 原计费项单价 / 虚拟资源包单价。

> 以图像高级压缩包为例，进行一次 800 x 600 以上规格的 HEIC 图像高级压缩，从图像高级压缩包中扣除 4 次。

图像高级压缩计费项	抵扣因子	抵扣说明
HEIC 图像高级压缩(800x600 以下规格)	1	1 次 800x600 以下规格的 HEIC 图像高级压缩处理从图像高级压缩包中抵扣 1 次
HEIC 图像高级压缩(800x600 以上规格)	4	1 次 800x600 以上规格的 HEIC 图像高级压缩处理从图像高级压缩包中抵扣 4 次
WebP-M6 图像高级压缩(800x600 以下规格)	1	1 次 800x600 以下规格的 WebP-M6 图像高级压缩处理从图像高级压缩包中抵扣 1 次
WebP-M6 图像高级压缩(800x600 以上规格)	4	1 次 800x600 以上规格的 WebP-M6 图像高级压缩处理从图像高级压缩包中抵扣 4 次
AVIF 图像高级压缩(800x600 以下规格)	1	1 次 800x600 以下规格的 AVIF 图像高级压缩处理从图像高级压缩包中抵扣 1 次
AVIF 图像高级压缩(800x600 以上规格)	4	1 次 800x600 以上规格的 AVIF 图像高级压缩处理从图像高级压缩包中抵扣 4 次

部分图像智能处理包采用分段阶梯定价的方式，次数月底清零，次月重新累计次数。

> 以智能去雾处理为例，如果本月进行了 200 万次智能去雾处理，将从图像智能处理包中扣除 1200 万次；如果本月进行了 400 万次智能去雾处理，将从图像智能处理包中扣除 6*300万次 + 3*100万次 = 2100 万次。

图像智能处理计费项	抵扣因子	抵扣说明
智能去雾	<=300万次：6 >300万次：3	1次智能去雾处理从图像智能处理包中抵扣6次 1次智能去雾处理从图像智能处理包中抵扣3次
风格转换	40	1次风格转换处理从图像智能处理包中抵扣40次
人物动漫化	72	1次人物动漫化处理从图像智能处理包中抵扣72次
物体擦除	40	1次物体擦除处理从图像智能处理包中抵扣40次
拉伸修复	<=300万次：7 >300万次：4	1次拉伸修复处理从图像智能处理包中抵扣7次 1次拉伸修复处理从图像智能处理包中抵扣4次
黑白上色	<=300万次：9 >300万次：5	1次黑白上色处理从图像智能处理包中抵扣9次 1次黑白上色处理从图像智能处理包中抵扣5次
无损放大	5	1次无损放大处理从图像智能处理包中抵扣5次

5、图像处理包生效时间 当选择“支付后立即生效”时，如果图像处理包是在某一个小时中间购买并为立即生效，将在下单后的下一个整点自动生效。当选择“指定时间生效”时，基础图像处理包指定生效时间时，仅支持指定为从当前时间的下个月到最长在未来 5 年内的月初（对应月份 1 号零点）生效；图像高级压缩包和图像智能处理包的可选指定时间为从当前时间的下一个整点到最长在未来 5 年内的整点时间，精确到小时粒度；图像智能处理包的可指定为从当前时间的下一个整点到最长在未来 5 年内的任意一天（该日期 0 点）生效，精确到天粒度。

查看消费明细

概述

您可以登录百度智能云控制台，进入财务页面查看消费明细。您可以在财务中心查看 BOS 详细的计量计费信息，包括请求次数、流量、图片处理和存储费用等。关于百度智能云财务使用方法，请查看财务-账单查看。本篇文章在基于百度智能云财务功能的基础上，结合 BOS 产品使用方式，对 BOS 的用量查看、账单金额查看和导出方式进行说明。

概念介绍

- **计费项**：计费项是一个产品的最小计费单元。如对于 BOS 来说，BOS 共有近 60 项左右的计费项，分别对存储容量、流量、请求次数、数据处理等进行计费。一份账单，具体可以体现为产品各个计费项的金额分布，同时对金额对应的用量数据进行展示。若您想查看产品详细账单，请先了解 BOS 全部计费项。
- **汇总账单**：汇总账单是某个时间下某一项计费项的金额汇总。如 2021 年 7 月标准存储费用，便是这一计费项在 7 月这一时间维度的汇总数据。
- **流水账单**：流水账单是按照产品计费的最小单位对金额进行展示。BOS 按照分钟级别进行数据推量，并按照小时级别形成流水账单。

查看时间维度（月级别）BOS计费项汇总账单

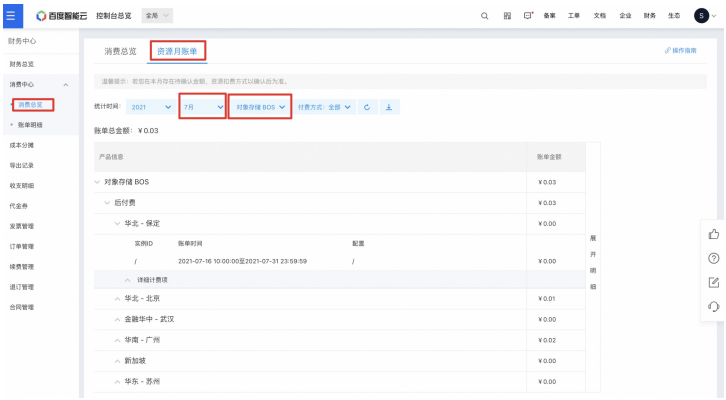
您可以通过查看资源月账单，对您过去每个自然月数据用量和对应计费项金额进行查看。您可以直接在控制台查看，也可以在控制台将资源月账单导出为 Excel 到本地，进行进一步统计、查看和对比。

1.登录： 登录 对象存储 BOS 管理控制台。

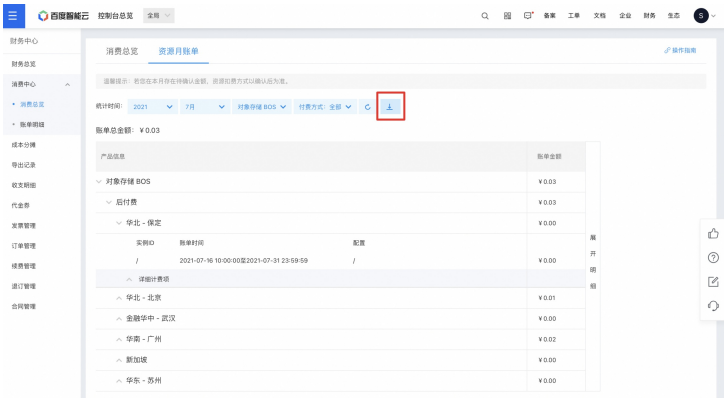
2. 进入财务：在控制台右上角找到"财务"，点击后进入财务页面。



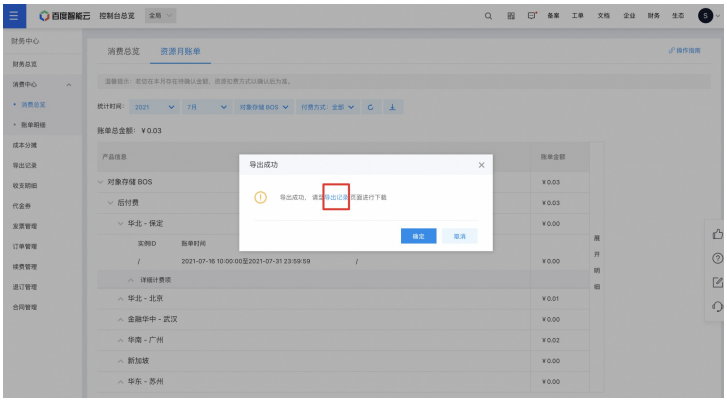
3.进入资源月账单：点击左侧"消费中心-消费总揽"，在页面上方选择"资源月账单"。您可以在上方进行筛选产品"对象存储 BOS"，再选择时间，便可以看到相应月份的 BOS 资源月账单。



4.下载资源月账单：筛选完毕后，您可以点击上方下载按钮，相应月份的 BOS 资源月账单会自动生成 Excel。



点击下载按钮后，在弹窗中点击"导出记录"。待资源月账单生成后，点击后方下载按钮，便可以将资源月账单下载到本地。



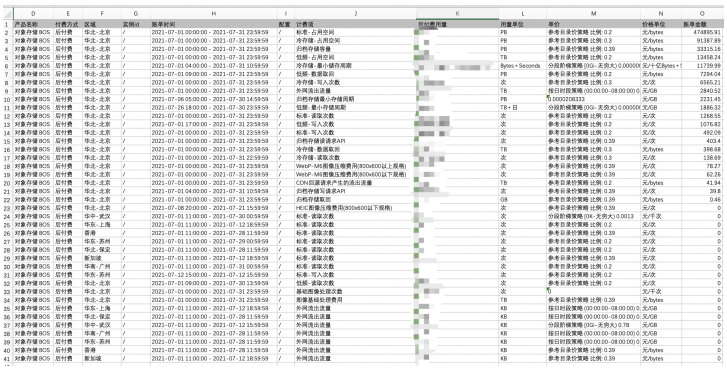
5.使用资源月账单：您可以在资源月账单中查看 BOS 某个月全部计费项的用量数据，并可以查看每个计费项所产生的账单金额，以方便您进行对账。

6.通过资源月账单进行分析：您也可以下载连续两个月，或者连续多月的资源月账单，通过多个月份账单的比对，查看每个计费项资源量的变化情况，对账单金额增长或账单金额下降的原因进行分析。

使用资源月账单进行财务分析案例

下图是某客户在 2021 年 7 月的 BOS 资源月账单。据客户反馈，7 月份的账单金额数据较 6 月预期金额多了近 1.5 万，希望排查账单金额增长原因。从客户 7 月份账单中可以看出：

- 客户使用了 4 种存储类型，分别是标准存储、低频存储、冷存储和归档存储。四种存储类型的存储容量费用总和占到了账单总费用的 88% 左右。因此，BOS 的存储费用占整个账单费用的比例是非常大的。通常来说，该现象为 BOS 的正常使用现象。虽然 BOS 有近 60 个计费项，但是"标准-占用空间"、"低频-占用空间"、"冷-占用空间"和"归档-占用空间"一般都产生最多的费用。若您想了解 BOS 计费项，请查看计费项详情。
- 客户在 7 月份账单中有一项"冷存储-最小存储周期"，该费用有 1.1 万左右。同时，还产生了"低频-最小存储周期"，费用有 1800 左右，"归档存储最小存储周期"产生的费用约 2200 左右。三个存储类型的最小存储周期费用在 7 月份一共有约 1.5 W 左右。客户同时下载了 6 月份的资源月账单，发现 6 月份账单并没有此项金额，于是便找到了金额较预期增加的原因。
- 客户经过查看 BOS 官方帮助文档，又进一步了解到，低频存储、冷存储、归档存储有最小存储时间，低频存储为 30 天，冷存储为 60 天，归档存储为 180 天。若数据在未存满该时间时提前删除，剩余未存满时间的存储费用在删除时将一次性收取，在账单中就体现为"最小存储时间"。根据账单分析，客户根据自身的使用情况和数据删除预期，重新选择了合适的存储类型，以避免后续这类费用的产生。



查看时间维度（小时级别）BOS计费项流水账单

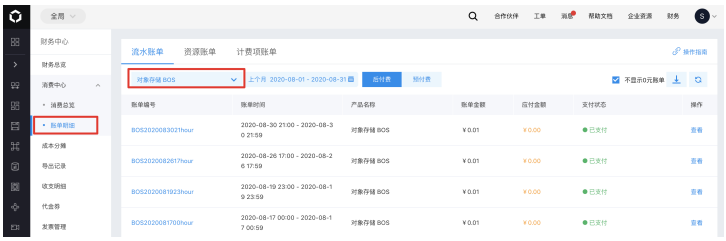
该账单中，系统为您账号下的 BOS 费用按照小时级别进行划分。按照时间维度，为您展示每小时内 BOS 每个计费项所产生的用量数据，并可以查看该小时内的费用明细。

1. 登录： 登录对象存储 BOS 管理控制台。

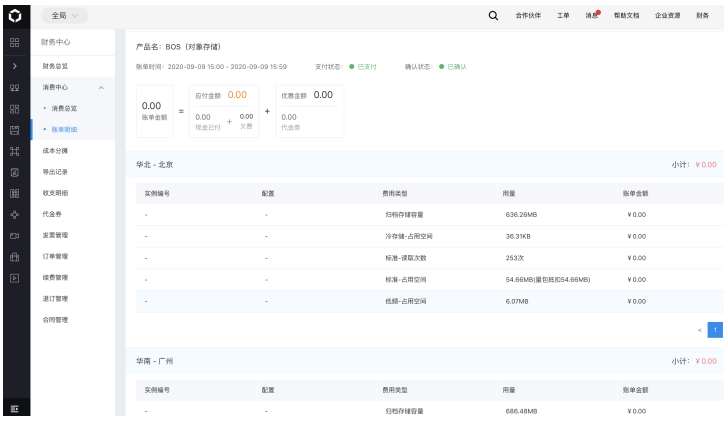
2.进入财务页面： 在控制台右上角找到"财务"，点击后进入财务页面。



3.查看 BOS 小时级别流水账单：点击左侧"消费中心-账单明细"查看账单。在流水账单中筛选产品"对象存储 BOS"，便可以看到 BOS 的流水账单。



4.查看 BOS 小时流水账单详情：后付费账单按小时出账，预付费账单根据您的的预付费订单出账，您可以点击账单编号进入小时账单查看详细计费信息。



查看Bucket维度计费项流水账单

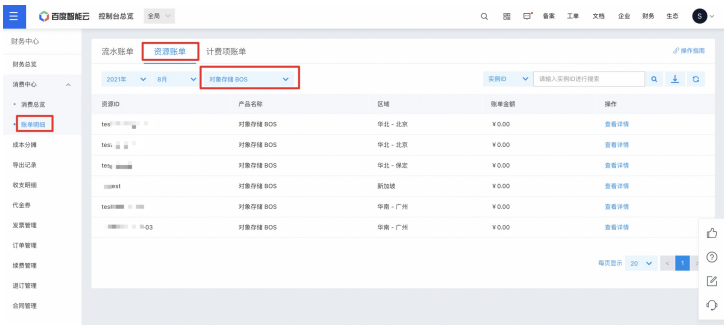
在该账单中，系统为您生成了按照 Bucket 维度划分的账单明细，您可以看到您账号下每个 Bucket 在当月或者以往月所产生的计费项，并可以查看该 Bucket 在每个小时内计费项的用量数据及费用明细。

1. 登录： 登录对象存储 BOS 管理控制台。

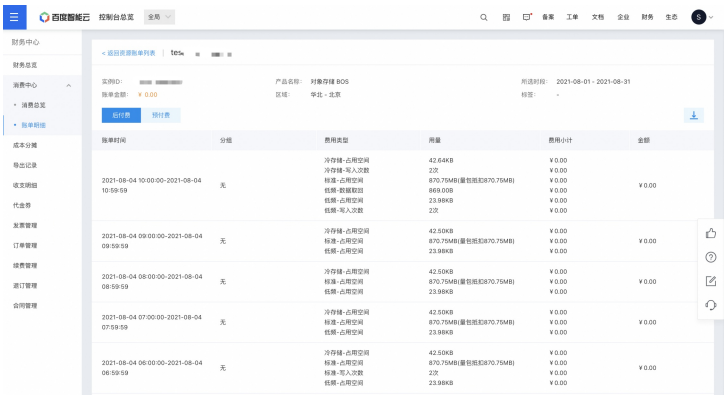
2.进入财务页面： 在控制台右上角找到"财务"，点击后进入财务页面。



3.查看 BOS 资源账单：点击左侧"消费中心-账单明细"。点击上方"资源账单"，筛选"对象存储 BOS "后可以查看 BOS 资源账单。



4.查看 Bucket 小时级别计费项账单：点击每条 Bucket 资源账单后面的"查看详情"，可以查看该 Bucket 的小时级别计费项账单。



查看并导出CSV用量数据

BOS 控制台支持查看并导出您账号下全部 Bucket 当月的统计信息，包括每个 Bucket 的总容量，总流量，总请求次数，以及每个 Bucket 的细粒度统计数据，如标准存储，低频存储等类型文件的容量。该功能目前正在通过白名单方式进行公测，如需使用请提交工单申请。该功能介绍可以查看查看并导出用量数据。

> **说明：**
>
> - 您也可以在 BOS 控制台全局概览页查看资源使用量数据（包括存储量、请求次数、上传下载流量等），但由于查看到的资源使用量数据有延迟，不作为实际计量数据，仅作参考。

查看 BOS 资源包使用情况

您可以通过控制台查看 BOS 资源包的使用和抵扣情况。具体路径方式是：

百度智能云控制台，点击右上角「财务」，进入财务概览页。然后点击左侧菜单中的「资源包管理」，进入资源包管理详情页。即可按需搜索并查看 BOS 相关资源包的总览和使用明细。



计费案例

概述

本文以案例形式介绍对象存储 BOS 费用的计算方法及计费方案的选择。以下案例单价来自 2020 年 9 月 9 日百度智能云官网公布的详细价格信息。单价的变动请以百度智能云官网发布的数据为准。

案例一：标准存储+数据访问

张先生在 BOS 内存储了 505 GB 标准存储类型文件，存储空间所在地域为华北-北京，每小时的请求数均值为 1000 次，每天的外网流出流量约为 5 GB 。李先生可以选择以下两种方案进行付费：

仅按量计费

```
 存储费用 = 505 GB * 0.119 元/GB/月 = 60.095 元
请求费用 = 1000 * 0.01元 / 10000次 * 24小时 * 30天 = 0.72 元
流量费用 = 5 GB * 30天 * 0.49 元/GB = 73.5 元
总费用：约 134.32 元
```

包年包月+按量计费

```
 存储费用 = 53.8 元（500 GB 标准存储包）+ 5 GB * 0.119 元/GB/月 = 54.395 元
请求费用 = 1000 * 0.01 元 / 10000次 * 24 小时 * 30 天 = 0.72 元
流量费用 = 48 元（100 GB 流量包）+ 50 GB * 0.49 元/GB = 72.5 元
总费用：约 127.62 元
```

> **计算说明：**
>
> - BOS 计算流量费用时，分为忙时流量（8:00~24:00，0.49 元/GB）和闲时流量（00:00~08:00，0.25元/GB），此处按忙时流量计算；

- 每月流量总计 5 GB * 30 = 150 GB，超出存储包的 50 GB 数据按量计费，且以忙时流量计算。

案例二：归档存储+数据访问

王先生 1 月 1 日在 BOS 内存储了 1200 GB （1 TB + 176 GB）的归档存储类型文件。存储空间所在地域为华北-北京，每小时的请求数均值为 1000 次。6 月 20 号的时候，王先生下载了一个大小为 10 GB 的文件到本地，并将文件内容更新之后重新上传至 BOS，更新后的文件大小仍为 10 GB。王先生可以选择以下两种方式进行计费：

仅按量计费

```
存储费用 = 1200 GB * 0.015 元/GB/月 = 18 元
外网流出流量费用 = 10 GB * 0.49 元/GB = 4.9 元
请求费用 = 1000 * 0.01元 / 10000次 * 24小时 * 30天 = 0.72 元
数据取回费用 = 10 GB * 0.12 元/GB = 1.2 元
最小存储周期费用 = 10 GB * 0.015 元/GB/月/30天 * 1/3 月 = 0.05 元
总费用：约 24.87 元
```

包年包月+按量计费

```
存储费用 = 14 元（1 TB 归档存储包）+ 176 GB * 0.015 元/GB/月 = 16.64 元
外网流出流量费用 = 10 GB * 0.49 元/GB = 4.9 元
请求费用 = 1000 * 0.01元 / 10000次 * 24小时 * 30天 = 0.72 元
数据取回费用 = 10 GB * 0.12 元/GB = 1.2 元
最小存储周期费用 = 10 GB * 0.015 元/GB/月/30天 * 1/3 月 = 0.05 元
总费用：约 23.51 元
```

计算说明：

- 上传新的同名文件到 BOS 会导致 BOS 内原有文件被删除，并保存新的文件。原文件被提前删除会产生最小存储周期费用。关于不同存储类型的最小存储周期，请查看分级存储。关于最小存储周期的费用计算方式，请查看存储费用，此处王先生距离最小存储周期还差10天（1/3月），需单独支付最小存储周期费用；
- BOS 计算流量费用时，分为忙时流量（8:00~24:00，0.49 元/GB）和闲时流量（00:00~08:00，0.25元/GB），此处按忙时流量计算；
- 外网流出流量参考 10 GB。考虑到文件的包头数据，实际流量大小不会与下载文件大小保持严格相同。

案例三：标准存储+跨区域复制

李女士 10 月 1 日在华北-北京的存储空间 A 内存储了 100 GB 标准存储类型的文件，并且每天新增 3 GB 文件。为保障数据存储安全，李女士设置了跨区域复制，将每天新增的 3 GB 数据和历史的 100 GB 同步至华东-苏州的存储空间 B 内。存储空间 A 和 B 平均每天的请求数共计约 30000 次。李女士可根据情况选择以下付费方式：

按量计费

```
存储费用 =（100 GB * 0.119 元/月/GB + 3 * 0.003967 元 / 天 * 29 天 + 3 * 0.003967 * 28 + 3 * 0.003967 * 27 + ..... + 3 * 0.003967 * 1 ）* 2 = 34.15387 元
跨区域复制流量费用 = 100 GB * 0.49 元/GB + 3 GB * 0.49元/GB * 29 天 = 91.63 元
请求费用 = 30000次 * 0.01元/10000 次 * 30 = 0.9 元
总费用：约 126.68 元
```

计算说明

- 1 GB 文件每天的存储费用为 0.119 元/月/30 天 = 0.003967 元/天;
- 因华东-苏州地域需要同步华北-北京地域的数据，所以总费用需乘 2。

案例四：低频存储+数据访问

李女士在华北-北京的存储空间 A 内存储了 500 GB 低频存储类型的文件，每天的访问请求数均值为 100 次。每天访问请求的数据量为2GB，其中外网流出流量占比为 1 GB，且均在闲时时段流出。李女士可根据情况选择以下付费方式：

仅按量计费

```
存储费用 = 500 GB * 0.08 元/月/GB = 40 元
数据取回费用 = 2 GB * 0.03 元/GB * 31 天 = 1.86 元
请求费用 = 100次 * 0.05元/10000 次 * 31 = 0.0155 元
外网流出费用 = 1 GB * 0.25元/GB *31 = 7.75 元
月度总费用：约 49.63 元
```

包月存储容量包+按量计费

```
存储费用 = 35.6 元（包月的 500 GB 低频存储容量包）
数据取回费用 = 2 GB * 0.03 元/GB * 31 天 = 1.86 元
请求费用 = 100次 * 0.05元/10000 次 * 31 = 0.0155 元
外网流出费用 = 1 GB * 0.25元/GB *31 = 7.75 元
月度总费用：约 45.23 元
```

案例五：标准存储+内网访问

李女士在华北-北京的存储空间 A 内存储了 500 GB 标准存储类型的文件，每天的访问请求数均值为 10000 次。每天访问请求的数据量为 50 GB，且均为华北-北京的内网访问流量。李女士可根据情况选择以下付费方式：

仅按量计费

```
存储费用 = 500 GB * 0.119 元/月/GB = 59.5 元
请求费用 = 10000次 * 0.01元/10000 次 * 31 = 0.31 元
内网流出费用 = 50 GB * 0元/GB *31 = 0 元
月度总费用：约 59.81 元
```

包月存储容量包+按量计费

```
 存储费用 = 53.8 元（包月的 500 GB 标准存储容量包）
请求费用 = 10000次 * 0.01元/10000 次 * 31 = 0.31 元
内网流出费用 = 50 GB * 0元/GB *31 = 0 元
月度总费用：约 54.11 元
```

案例六：冷存储+数据访问

李女士在华北-北京的存储空间 A 内存储了 500 GB 冷存储类型的文件，每月的访问请求数总和约为 10000 次。每月访问请求的数据量为 10 GB，且均为内网访问流量。李女士可根据情况选择以下付费方式：

仅按量计费

```
 年存储费用 = 500 GB * 0.032 元/月/GB * 12 = 192 元
年请求费用 = 10000次 * 0.1元/10000 次 * 12 = 1.2 元
年数据取回费用 = 10 GB * 0.06元/GB * 12 = 7.2 元
年度内网流出费用 = 10 GB * 0元/GB * 12 = 0 元
年度总费用：约 200.4 元
```

包年存储容量包+按量计费

```
 年存储费用 = 144 元（包年的 500 GB 冷存储容量包）
年请求费用 = 10000次 * 0.1元/10000 次 * 12 = 1.2 元
年数据取回费用 = 10 GB * 0.06元/GB * 12 = 7.2 元
年度内网流出费用 = 10 GB * 0元/GB * 12 = 0 元
年度总费用：约 152.4 元
```

案例七：标准存储+开启回收站

李女士在华北-北京的存储桶 A 内存储了 500 GB 标准存储类型的文件，并对存储桶开启了回收站的功能。500 GB 数据中有 100 GB 的数据被删除，并且进入了回收站。存储桶 A 每天的请求数均值为 10000 次。每天访问请求的数据量为 50 GB，均为内网访问流量。李女士可根据情况选择以下付费方式：

仅按量计费

```
 存储费用 = 500 GB * 0.119 元/月/GB = 59.5 元
请求费用 = 10000次 * 0.01元/10000 次 * 31 = 0.31 元
内网流出费用 = 50 GB * 0元/GB * 31 = 0 元
月度总费用：约 59.81 元
```

包月存储容量包+按量计费

```
 存储费用 = 53.8 元（包月的 500 GB 标准存储容量包）
请求费用 = 10000次 * 0.01元/10000 次 * 31 = 0.31 元
内网流出费用 = 50 GB * 0元/GB * 31 = 0 元
月度总费用：约 54.11 元
```

案例八：标准存储+CDN 回源流出流量+外网访问下行流量

李女士在华北-北京的存储桶 A 内存储了 500 GB 标准存储类型的文件。每月 CDN 服务回源 BOS 的流量 100 GB。另外存储桶 A 每天的请求数均值为 10000 次。每月外网访问请求的流出流量为 100 GB，且均为忙时的流出流量。李女士可根据情况选择以下付费方式：

仅按量计费

```
 存储费用 = 500 GB * 0.119 元/月/GB = 59.5 元
请求费用 = 10000次 * 0.01元/10000 次 * 31 = 0.31 元
外网流出费用 = 100 GB * 0.49元/GB = 49 元
CDN 回源流出费用 = 100 GB * 0.14元/GB = 14 元
月度总费用：约 122.81 元
```

包月存储容量包+包月流量包+按量计费

```
 存储费用 = 53.8 元（包月的 500 GB 标准存储容量包）
请求费用 = 10000次 * 0.01元/10000 次 * 31 = 0.31 元
外网流出费用 = 48 元（包月的 100 GB 外网下行流量包）
CDN 回源流出费用 = 13.8 元（包月的 100 GB CDN 回源流量包）
月度总费用：约 115.91 元
```

总结

由上面的案例可以看出：

- 当您使用的资源达到一定数值时，购买资源包（包年包月）的费用更低；
- 存储类型不同，产生的存储费用也不一样，请根据自身的实际情况选择合适的存储类型。**虽然归档存储的存储费用较标准存储更低，但归档存储有最小存储时间限制**。若未存满该时间限制就将其删除，会**产生最小存储周期费用**。若对归档存储类型文件进行读取或者下载，还需先将该文件取回，且取回操作会**产生数据取回费用**。因此，归档存储适合访问频率极低的文件，如基因数据、合规性媒资数据、封存的电子档案数据等等；

余额不足和欠费提醒

余额不足提醒

- 根据您最近3天的账单金额来判断您的账户余额（含可用代金券）是否足够支付未来3天的费用，若不足以支付，系统发送续费提醒。
- 根据您最近1天的账单金额来判断您的账户余额（含可用代金券）是否足以支付未来1天的费用，若不足以支付，系统发送续费提醒。

欠费处理

- 北京时间整点检查您的账户余额是否足以支付本次BOS账单的费用（如北京时间11点整检查账户余额是否足以支付10点至11点的账单费用），若不足以支付，即为欠费，欠费时系统会发送欠费通知。
- 欠费后立即停服，系统会发送欠费停服通知，数据为您保留15天，期间收取存储空间的费用。若15天内仍未充值则清除数据，清除数据的前1天和清除数据时系统都会发送清除数据通知。
- 欠费停服后的15天内，如果用户充值后可用余额大于0，BOS的数据将自动恢复。

计费常见问题

1.账户欠费相关

1.1 欠费后数据会丢失吗？

- 当您的账户余额不足0元无法支付BOS账单，即为欠费，欠费时系统会发送欠费通知。
- 欠费后立即停服，系统会发送欠费停服通知，数据为您保留15天，期间收取存储空间的费用。若15天内仍未充值则清除数据，清除数据的前1天和清除数据时系统都会发送清除数据通知。

1.2 欠费后需要怎么操作才能正常使用BOS？代金券可以用吗？

- 当您的账户欠费后，您只有通过补交欠款才可以消除欠款。补交欠款后，您可以正常使用 BOS；
- 欠费期间，代金券无法使用，请先补交欠款，补交完成后方可进行代金券正常抵扣。

1.3 欠费后BOS资源找不到了怎么办？

- 当 BOS 服务处于欠费状态，且欠费时间超过 24 小时，BOS 服务将自动停止。而用户所占用的存储空间的这部分资源仍会继续按量扣费，因此欠费余额会累计。在欠费后 24 小时内进行充值，用户的服务将不会受到停服影响。如用户欠费时间超过 24 小时，则 BOS 服务将自动停止，用户可以在 15 天内充值补足欠费后，服务会自动开启，可以继续使用。欠费超过 15 天，将视为用户主动放弃 BOS 存储服务，存储空间将被回收，存储空间内的数据会被清理，数据不可恢复。详细描述请参考余额不足和欠费提醒。

1.4 BOS欠费已充值依然无法打开怎么处理？

- BOS 欠费充值后一般是 10-20 分钟左右才可以生效并正常使用，请耐心等待，10 分钟后尝试重新刷新浏览器缓存。

2.产品使用相关

2.1 开启镜像回源后的流量扣费标准是什么？

- 镜像回源是一个Bucket级别的设置，目前镜像回源抓取到的文件会默认存到标准存储里，计费项包含存储、请求和流量三部分费用，具体计费介绍请查看产品定价。

2.2 BOS通过GetObject方法调用获取到的下载文件，会有流量消费吗？

- GetObject 方法调用获取到的下载文件会收取请求费用 +流量费用 ，如果是低频存储和冷存储还会收取数据取回费用 ，具体费用介绍请查看产品定价。

2.3 BOS上重复上传未修改过的文件会消耗请求次数么？

- 每一次的上传都会请求对应的上传接口，会计算请求次数。

2.4 BOS的最小计费时间是什么？

- BOS 每分钟会推送账单用量数据，百度智能云会根据该用量数据和计费项单价，在系统中核算分钟级别费用，并对过去 60 分钟的费用累加形成小时级别账单。具体的计费规则，请查看 BOS计费规则。

2.5 我开启了CDN加速，为什么没有看到该项费用？

- 若您在 BOS 中开启了 CDN 加速，CDN 加速费用将会在另一款产品"内容分发网络 CDN "的账单中进行展示，BOS 账单中不展示，请您前往 CDN 账单进行查看。

2.6 CDN上行加速的费用怎么查看？

- 我们通常所说的 CDN 加速是指下行加速，不包含上行加速功能。若您希望使用 CDN 上行加速，需要在控制台再开启 CDN 上行加速。CDN 上行加速采用 CDN 动态加速功能，该功能的费用将在另一款产品"内容分发网络 CDN "的账单中进行展示，BOS 账单中不展示，请您前往 CDN 账单进行查看。

2.7 我主要通过控制台和SDK进行操作，没有直接用到API，为什么还会产生读写相关的API次数费用？

- BOS 的控制台、各类语言 SDK、各类封装工具均在底层使用 BOS API。因此，若您使用控制台或者 SDK，本质上也是在使用 BOS 的各类 API，因此会产生 API 读写次数相关费用。关于 API 相关计费项信息，请您查看 BOS 请求费用计费项说明。

2.8 我主要使用外网进行数据上传，会产生相关费用吗？

- BOS 仅收取外网（即公网/互联网）下行流量费用，外网上行、内网上行、内网下行费用都免费使用，不产生费用。关于 BOS 流量相关计费项说明，请查看 BOS 流量费用说明。

3.资源包相关

3.1 使用BOS一定要购买资源包吗？

- BOS 默认采用后付费，不一定需要购买资源包。若您未购买资源包便上传文件到 BOS，那么该文件的存储费用将默认按照按需付费的方式进行收取，您只要保证您百度智能云账户中有余额可供进行账单金额抵扣即可。

3.2 BOS资源包会比后付费的单价便宜吗？

- 若您购买的 BOS 资源包的时长为 6 个月、1 年、2 年或 3 年，则该资源包会享受折扣。其中：6 个月时长的资源包为后付费价格的 8.3 折，1 年、2 年和 3 年资源包的购买折扣均为后付费价格的 7.5 折。

3.3 购买了BOS存储包后如何使用？

- BOS 支持包年包月（预付费）形式的存储包购买。购买存储包后，您的BOS消费会首先用存储包进行抵扣（无需特殊设置），超出部分按量付费。您可以直接在管理控制台的"BOS->存储包"页签进行存储包购买，购买时按照提示选择区域、规格、购买时长和生效时间即可。详细说明可以参考如下文档BOS存储包计费。

3.4 资源包到期后可以续费吗？

- 目前资源包暂不支持续费。但您可以重新购买一个新的资源包，并将新资源包的生效时间设置为已有资源包的到期时间。如此，和续费的效果相同，系统将为您的资源包无缝衔接。

4.月初对账相关

4.1 如何查看上个月BOS的消费账单？

- 百度智能云提供可视化的 BOS 资源月账单。您可以登录控制台，点击右上角进入"财务"进行查看。关于 BOS 的各类账单查看方式，请查看 BOS账单查看和分析，已为您整理了各类账单的查看方式。

4.2 如何查看详细的每小时账单？

- 百度智能云提供可视化的小时级别流水账单。您可以登录控制台，点击右上角进入"财务"进行查看。关于 BOS 的各类账单查看方式，请查看 BOS账单查看和分析，已为您整理了各类账单的查看方式。

4.3 如何查看特定Bucket所产生的费用？

- 百度智能云提供Bucket级别详细账单。您可以登录控制台，点击右上角进入"财务"进行查看。关于 BOS 的各类账单查看方式，请查看 BOS账单查看和分析，已为您整理了各类账单的查看方式。

4.4 如何查看账号下标准存储的用量数据？

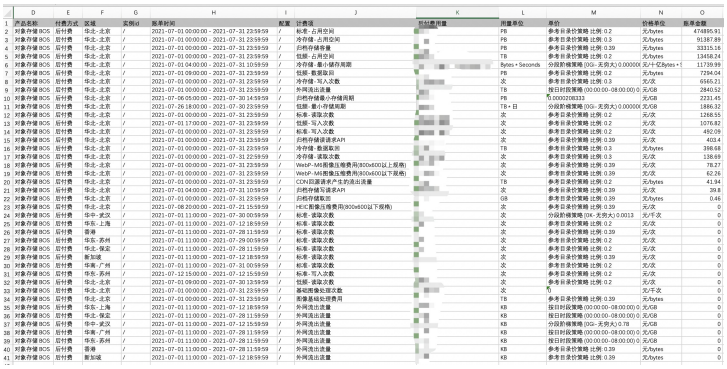
- 您可以通过百度智能云提供的各类账单进行用量数据查看，也可以使用 BOS 提供的控制台用量导出功能。详情请参考BOS用量查看和导出。

4.5 我的数据量没有发生变化，为什么2月的容量消费金额和3月的容量消费金额不一样？难道官网的月单价不准确吗？

- BOS 采用按量付费的方式进行出账，分钟级别推量结算，小时级别出账。官网的容量月单价并非每个自然月的月单价，而是 30 天的参考价。如官网标定的标准存储月容量单价为 0.119 元/GB/月，那么天容量单价为 0.119/30=0.00396667元/GB/天，小时容量单价为 0.000165278元/GB/小时，分钟容量单价为 0.0000027546元/GB/分钟。若您在该分钟内的容量峰值为 10987654321 Bytes，那么该容量的分钟级别账单金额为 10987654321/1024/1024/1024✖0.0000027546=0.00002818796 元。若在 30 天的月份，月账单金额为 0.00002818796 ✖ 60 ✖ 24 ✖ 30 = 1.22 元；若在 31 天的月份，月账单金额为 0.00002818796 ✖ 60 ✖ 24 ✖ 31 = 1.26 元；若在 28 天的月份，月账单金额为 0.00002818796 ✖ 60 ✖ 24 ✖ 28 = 1.10 元。因此，BOS 采用精准的分钟级别结算出账方式，官网的月单价仅为 30 天参考价。

4.6 为什么这个月的费用多了很多，可以怎么排查？

- 您可以在资源月账单中查看 BOS 某个月全部计费项的用量数据，并可以查看每个计费项所产生的账单金额，以方便您进行对账。您也可以下载连续两个月，或者连续多月的资源月账单，通过多个月份账单的比对，查看每个计费项资源量的变化情况，对账单金额增长或账单金额下降的原因进行分析。资源月账单的查看的方式，请参考BOS账单查看和分析。

> **通过资源月账单进行账单分析的客户案例**：
>
> - 下图是某客户在 2021 年 7 月的 BOS 资源月账单。据客户反馈，7 月份的账单金额数据较 6 月预期金额多了近 1.5 万，希望排查账单金额增长原因。从客户 7 月份账单中可以看出：
>
> 
>
> - 客户使用了 4 种存储类型，分别是标准存储、低频存储、冷存储和归档存储。四种存储类型的存储容量费用总和占到了账单总费用的 88% 左右。因此，BOS 的存储费用占整个账单费用的比例是非常大的。通常来说，该现象为 BOS 的正常使用现象。虽然 BOS 有近 60 个计费项，但是"标准-占用空间"、"低频-占用空间"、"冷-占用空间"和"归档-占用空间"一般都产生最多的费用。若您想了解 BOS 计费项，请查看计费项详情。
> - 客户在 7 月份账单中有一项"冷存储-最小存储周期"，该费用有 1.1 万左右。同时，还产生了"低频-最小存储周期"，费用有 1800 左右，"归档存储最小存储周期"产生的费用约 2200 左右。三个存储类型的最小存储周期费用在 7 月份一共有约 1.5 W 左右。客户同时下载了 6 月份的资源月账单，发现 6 月份账单并没有此项金额，于是便找到了金额较预期增加的原因。
> - 客户经过查看 BOS 官方帮助文档，又进一步了解到，低频存储、冷存储、归档存储有最小存储时间，低频存储为 30 天，冷存储为 60 天，归档存储为 180 天。若数据在未存满该时间时提前删除，剩余未存满时间的存储费用在删除时将一次性收取，在账单中就体现为"最小存储时间"。根据账单分析，客户根据自身的使用情况和数据删除预期，重新选择了合适的存储类型，以避免后续这类费用的产生。

4.7 我的冷存储容量只有1.2PB，为什么账单中会有74.90PB的冷存储最小存储周期费用？

- BOS 有最小存储周期的计费项，分别是低频多AZ最小存储周期、低频最小存储周期、冷存储最小存储周期和归档最小存储周期。针对最小存储周期计费项，由于账单中数据为月粒度聚合展示，因此账单中的用量数据为"容量 x 剩余时间"的乘积，单位是bytes✖小时或者bytes✖天（其中bytes自动向上进位为GB/TB/PB）。比如下面这一项的用量数据为74.90PB✖小时，其中 74.90 是容量和小时数的乘积。

- 如果按照剩余 40 天即 960 小时计算，容量数据为 74.9✖1024/40/24=79.89TB，并非产生了 74.90PB 的账单。若您想查看详细的用量，可以查看小时账单。关于最小存储周期计费项的介绍，请查看最小存储周期计费项说明。

更多问题

更多常见问题请查看常见问题。

快速入门

开始使用BOS

注册百度智能云账号

个人用户和企业用户均需先注册百度智能云账号，注册完成后才可使用 BOS。若您已完成账号注册请登录百度智能云官网。点击页面右上方"免费注册"完成账号注册。注册步骤请参考注册账户说明。

实名认证

在您注册百度智能云账号后，需完成统一的初级实名认证。但对于部分产品如对象存储 BOS，因产品特性和风控政策要求，在您进一步使用 BOS 前，您还需完成高级实名认证。

完成统一初级实名认证

用户开通 BOS 服务前必须要经过实名认证，否则会开通失败。每个账户只需认证一次即可，用户可根据情况选择个人认证或企业认证。

用户登录成功之后，点击控制台右上角的用户名的"安全认证"进入认证页面并点击"实名认证"。 用户根据自身情况，进行"企业认证"或者"个人认证"。认证成功后，您可以使用百度智能云各项产品。根据最新风控政策要求，在您使用对象存储 BOS 前，还需完成百度智能云高级实名认证。



完成高级实名认证

根据风控要求，自 2021 年 4 月 19 日起，对象存储 BOS 的用户需完成百度智能云高级实名认证，认证涵盖个人用户和企业用户。

- 在 2021 年 4 月 19 日后开通 BOS 服务的用户，完成百度智能云高级实名认证后方可使用 BOS 服务，请在 BOS 控制台开通服务时完成高级实名认证。
  - **个人用户高级实名认证**：需下载百度智能云 APP，登录后进入"我的-页面上方个人认证-实名认证"完成认证操作，详细认证流程见个人用户高级实名认证。
  - **企业用户高级实名认证**：请登录百度智能云控制台后，点击右上角头像进入用户中心-实名认证模块，点击"添加高级实名认证"按钮进行企业高级实名认证，详细认证流程见企业用户高级实名认证。
- 在 2021 年 4 月 19 日前开通 BOS 服务的用户（个人用户及企业用户），请尽快完成高级实名认证。完成认证前，您的 BOS 服务可正常使用。认证流程参考百度智能云个人用户高级实名认证和企业用户高级实名认证。

费用说明

- **计费项目**：BOS 服务涵盖存储费用、流量费用、请求次数费用和数据处理费用。关于计费项目的计费逻辑，请参考计费项目
- **计费方式**：您可以选择按需计费和资源包计费两种计费方式。对于按需付费，BOS 按分钟实时计费（不足1分钟按1分钟计），按小时在百度智能云-财务为您出具账单。另外一种计费方式——资源包，为包年包月的购买形式。购买资源包后，您所产生的使用费用会先通过资源包进行抵扣，超出部分按量付费。

> **注意：**
>
> - “按需计费”购买前需保证账户无欠款。

快速入门流程

概述

本文档旨在为初级用户介绍如何在控制台上完成创建 Bucket、上传 Object 等基本功能的操作，使用户能快速了解购买和操作对象存储BOS的流程。

基本概念：

- Object：在 BOS 中，用户操作的基本数据单元是 Object。
- Bucket：Bucket 是存放数据的容器，可以当成是存储桶。所有 Object 都必须存储在 Bucket 中。
- 文件夹：Bucket 中为了更好的归类 Object，可以创建文件夹用于存储同类的 Object。

使用流程

基本操作流程如下图所示：



1. 开通 BOS 服务：新用户首次使用 BOS 时，需要开通 BOS 服务。
2. 创建 Bucket ：在 BOS 中创建一个存储空间。
3. 上传 Object：对 Object 进行上传，BOS 支持任意类型文件的存储。
4. 获取 Object：将数据对象存储到 BOS 后，用户可以对 Object 进行下载。
5. 删除 Object/Bucket：当用户不需要使用 BOS 上的存储对象时，请将其删除以免进一步产生费用。

创建Bucket

概述

开通BOS服务后，用户就可以在BOS中进行对象存储了。首先，用户需要先建立一个存储空间，即Bucket。

注意事项

- Bucket 具有区域属性，只能位于一个区域。因为 Bucket 名称创建后不能修改名称和所属区域，所以建议根据业务情况就近存储，方便上传和下载以提高访问速度。
- 每个用户最多创建 100 个 Bucket。
- 如您开通了多区域服务，请根据需要选择 Bucket 所在区域，请参考区域选择说明。
- 创建 Bucket 时可以选择 Bucket 的默认存储类型，如果用户使用 API、CLI 或者SDK 上传的 Object 未指定存储类型，则继承 Bucket 的默认存储类型。在管理控制台上，上传的 Object 默认为标准存储。如果 Object 的存储类型和 Bucket 存储类型不一致时，以 Object 的存储类型为准。存储类型包含标准存储、低频存储、冷存储和归档存储四种，具体使用场景和性能请参见分级存储。
- 每个 Bucket 的名称具有全局唯一性。可使用前缀确保名称的唯一性，如以所在的组织的名称作为 Bucket 的前缀。Bucket 创建后，不能更改名称。
- 在 Bucket 创建时即可设置相应的读写权限，包括：私有、公共读和公共读写三种权限。若需要设置高级权限，则请用户先创建好 Bucket 再去对应的详情页面中进行设置，具体内容请参考 Bucket 权限控制。

使用管理控制台创建 Bucket

1. 登录 对象存储 BOS 管理控制台。
2. 在控制台左侧的导航栏中找到Bucket列表并点击 + 按钮（“新建 Bucket”按钮），在弹出框中按照提示创建Bucket。


对象存储BOS
全局概览
存储管理
输入Bucket名称
概览
创建规则
规则名称
新建Bucket
* Bucket名称：
Bucket名称创建后不可修改，且在所有区域内唯一
只能包含小写字母、数字和"-"，开头结尾为小写字母和数字，长度在3-63之间
所属地域：
华北 - 北京
默认存储类型：
标准存储
低频存储
冷存储
上传文件时如果未指定存储类型，则使用该默认存储类型
标准存储、低频存储、冷存储、归档存储的区别是什么?
读写权限：
私有
公共读
公共读写
私有：只有用户本人可读写该Bucket中的数据
创建完成后可在 Bucket设置-权限设置 里进行更灵活的权限管理，如防盗链等
CDN官方加速：
开
您可使用官方域名对文件进行加速访问
CDN加速域名：
.cdn.bcebos.com
计费方式：
按用量收费
免费创建，使用阶段按照用量收费。
了解计费详情
绑定标签：
标签键：
请选择已有或手动输入
值：
请选择已有或手动输入
添加标签
帮助文档
前往标签管理
标签支持您按各种标准（如用途、所有者或项目）对资源进行分类；
每个标签包含键和值两部分。
取消
确定

配置项	配置描述
Bucket 名称	• 每个 Bucket 的名称具有全局唯一性。可使用前缀确保名称的唯一性，如以所在的组织的名称作为 Bucket 的前缀。Bucket 创建后，其名称不能更改。 • Bucket 名称只能包含小写字母、数字和"-"，开头结尾为小写字母和数字，且长度在4-63之间。
所属地域	• Bucket 具有区域属性，只能位于一个区域。Bucket 名称创建后不能修改所属区域，因此建议根据业务情况就近存储，方便上传和下载以提高访问速度。 • 如您开通了多区域服务，请根据需要选择 Bucket 所在区域，请参考区域选择说明。
默认存储类型	• 创建 Bucket 时可选择 Bucket 默认存储类型。如果用户使用 API、CLI 或者 SDK 上传的 Object 未指定存储类型，则继承所在 Bucket 的默认存储类型。在 Console 控制台上，上传的 Object 默认为标准存储。当 Object 的存储类型和 Bucket 存储类型不一致时，以Object 的存储类型为准。 • 存储类型包含标准存储、低频存储、冷存储和归档存储四种，具体使用场景和性能请参见分级存储。
读写权限	• 私有： Bucket 创建者独享该 Bucket 的全部权限，其他人不能对该Bucket 写入数据，也不可访问读取该 Bucket。适用于存储私密文件的场景。 • 公共读：Bucket 创建者独享该 Bucket 的全部权限，其他人可以访问读取该 Bucket 内的内容，但无权写入。适用于存储需要公众读取的文件的场景，如通知性文件、广告内容等。请注意，若开启该种权限，互联网上的所有用户都可以对该 Bucket 内的文件进行访问读取，请谨慎开启。 • 公共读写：包括 Bucket 创建者在内的任何人都可以访问和写入文件。请注意，若开启该种权限，互联网上的所有用户都可以对该 Bucket 内的文件进行读写操作，请谨慎开启。 • 若需要设置高级权限，则需要在创建完 Bucket 后，修改相应权限设置，具体内容请参考更新 Bucket 权限。
CDN 官方加速	• 开启 Bucket 的 CDN 官方加速后，对应 Bucket 域名的 DNS 解析修改为 CDN 提供的 CNAME 域名，从而接入 CDN 节点，开启 CDN 加速。 • 创建 Bucket 时默认开启 CDN 官方加速。如果需要关闭，可直接在创建时将 " CDN 官方加速 " 设为 OFF 状态。
绑定标签	• 可以根据项目和场景需求，您可以为 Bucket 绑定标签，便于对 Bucket 进行分类和识别管理。在绑定标签前，您需要在控制台上先创建标签。
其他说明	• 每个用户最多创建 100 个 Bucket，请您谨慎创建。若您创建的 Bucket 数量已经满额，请在清除已创建 Bucket 内的文件后将该 Bucket 删除，否则 Bucket 无法删除。清除已有 Bucket 后，才能创建新的 Bucket。 • Bucket 创建后，并不会立马计费。只有当 Bucket 中存入文件后，系统才会开始对 Bucket 内文件占用的存储空间进行计费。

4. 点击 **确认** 后，该 Bucket 创建完成。下一步，您可以在 Bucket 中上传文件。

使用 API 接口创建 Bucket

API 接口：PutBucket。

使用 SDK 创建 Bucket

- JAVA SDK：新建Bucket
- Python SDK：新建Bucket
- PHP SDK：新建Bucket
- C# SDK：新建Bucket
- JavaScript SDK：新建Bucket
- Android SDK：新建Bucket
- iOS SDK：新建Bucket
- Ruby SDK：新建Bucket
- Go SDK：新建Bucket
- C++ SDK：新建Bucket

上传Object

概述

BOS 支持任意类型文件的存储，创建 Bucket 成功后，可通过控制台对该 Bucket 中的 Object 进行上传、下载、复制和删除操作。BOS 文件的存储支持标准存储、低频存储、冷存储和归档存储四种存储类型，用户可以根据数据的冷热情况选择，如何选择可以参考分级存储。

注意事项

- 单个上传文件大小不得超过 5GB。若需要传输大文件，建议使用 SDK 工具。
- 若存储桶不开启版本控制 ，请求者重复上传相同文件名，之前上传的数据将会被覆盖。

使用 SDK 上传 Object

- JAVA SDK：上传文件
- Python SDK：上传文件
- PHP SDK：上传文件
- C# SDK：上传Object
- JavaScript SDK：上传Object
- Android SDK：上传Object
- iOS SDK：上传Object
- Ruby SDK：上传文件
- Go SDK：上传文件
- C++ SDK：上传文件

使用工具上传 Object

您可以使用 BOS 提供的官方工具进行上传。

- 使用命令行工具 BOS cmd 快速上传，详情请参见 BOS cmd-文件上传。
- 使用BOS 桌面，进行上传 Object，详情请参见 BOS 桌面工具-文件上传。

使用管理控制台上传 Object

1. 登录 对象存储 BOS 管理控制台。
2. 在左侧选择创建完成的 Bucket，点击 **上传文件**，打开本地文件选择对话框。



3. 选择文件后，选择 **标准存储**、**低频存储**、**冷存储** 或 **归档存储** 存储类型，并点击 **开始上传** 按钮进行上传。可同时选择多个文件上传。



上传完成后会在对应 Object 的 **存储类型** 列显示。

使用 API 接口上传 Object

API 接口：PutObject。

获取Object

概述

将数据对象存储到 BOS 后，您可以对 Object 进行下载。

前提条件

文件已上传到 Bucket 中。详情请参见 上传 Object。

使用管理控制台获取 Object

1. 登录 对象存储 BOS 管理控制台。
2. 选择需要操作的 Bucket，点击 **文件列表** 页签。
3. 根据您的需求，您可通过以下两种方式对Object进行下载。
   - 点击要下载的 Object 文件右侧的“复制链接”，并在新的浏览器窗口中粘贴已复制的下载地址来进行下载。



   - 点击要下载的 Object 文件右侧的下载按钮进行下载。



使用 API 接口获取 Object

API 接口：GetObject。

使用 SDK 获取 Object

- JAVA SDK：下载文件
- Python SDK：下载文件
- PHP SDK：下载文件
- C# SDK：获取Object
- JavaScript SDK：获取Object
- Android SDK：获取Object
- iOS SDK：获取Object
- Ruby SDK：下载文件
- Go SDK：下载文件
- C++ SDK：下载文件

使用工具获取 Object

您可以使用 BOS 提供的官方工具 BOS 桌面获取 Object，详情请参见 文件下载。

删除Object

概述

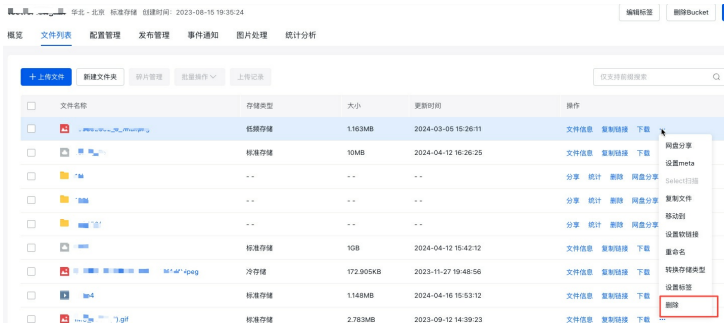
当您 Bucket 内的 Object 不需要再继续存储时，请及时将其删除以免进一步产生费用。

注意事项

Object 删除后 **无法恢复**，请谨慎操作。

使用管理控制台删除 Object

1. 登录 对象存储 BOS 管理控制台。
2. 选择需要操作的 Bucket，点击 **文件列表** 页签。
3. 在 **文件列表** 页面，点击要删除的 Object 右侧的 **更多 -> 删除**，删除指定的 Object。



使用 API 接口删除 Object

API 接口：DeleteObject。

使用 SDK 删除 Object

- JAVA SDK：删除文件
- Python SDK：删除文件
- PHP SDK：删除文件
- C# SDK：删除Object
- JavaScript SDK：删除Object
- Android SDK：删除Object
- iOS SDK：删除Object
- Ruby SDK：删除文件
- Go SDK：删除文件
- C++ SDK：删除文件

使用工具删除 Object

您可以使用 BOS 提供的官方工具 BOS 桌面删除 Object，详情请参见 文件删除。

删除Bucket

概述

当您不需要使用 BOS 上的存储空间时，请及时将其删除以免进一步产生费用。

注意事项

Bucket 删除后 **无法恢复**，请谨慎操作。

使用管理控制台删除 Bucket

1. 登录 对象存储 BOS 管理控制台。
2. 选择需要操作的Bucket。
3. 在 Bucket 页面，点击右上角的 **删除 Bucket**，删除指定的 Bucket。



> **说明：**
>
> - 要删除 Bucket，必须先删除该 Bucket 下的所有 Object。只有当前 Bucket 为空的时候，才可以对此 Bucket 进行删除操作，否则会出现相应提示。
> - 要删除 Bucket下 的文件夹，必须先删除文件夹下的所有 Object，否则会出现相应提示。

使用 API 接口删除 Bucket

API 接口：DeleteBucket。

使用 SDK 删除 Bucket

- JAVA SDK：删除Bucket
- Python SDK：删除Bucket

- PHP SDK：删除Bucket
- C# SDK：删除Bucket
- JavaScript SDK：删除Bucket
- Android SDK：删除Bucket
- iOS SDK：删除Bucket
- Ruby SDK：删除Bucket
- Go SDK：删除Bucket
- C++ SDK：删除Bucket

控制台操作指南

操作准备

开通对象存储

1. 注册并登录百度智能云平台，请参考注册和登录。
2. 如果未进行实名认证，点击控制台右上角的用户名的“安全认证”进入认证页面并点击“实名认证”。请根据实际情况选择“企业认证”或者“个人认证”。



3. 选择“产品服务>对象存储 BOS”，进入“开通对象存储BOS”页面，点击“立即开通”。BOS 开通后，在您账户余额未欠费的情况下，即可创建存储的 Bucket 并上传文件。只有当您上传文件后，BOS 才会产生费用相应的费用。

> **说明**：
>
> - 开通 BOS 需要实名认证，每个账户只需认证一次；
> - 首次使用 BOS 时，需开通 BOS 服务；
> - BOS 支持按需计费的后付费和存储包形式的预付费两种付费形式。

控制台概述

BOS 控制台基于 Bucket 进行管理，将数据传输、数据存储、数据处理和数据分发 4 个环节有机融为一体。

- BOS 支持控制台、SDK 及 API、工具等多种形式的数据传输，关于数据传输方式可参考 BOS 控制台导航“文档与工具”的“数据传输方式”页面。
- 数据管理包含 Object 管理和 Part 管理两部分。数据存储的基本单元为 Object，当单个Object 文件过大时，需要进行分块管理。
- BOS 支持图片处理，音视频数据处理，大数据 Select 扫描 3 种数据处理形式。图片处理支持包括图片缩放、旋转、水印等基础图片处理操作，以及图像审核、图像识别、文字识别等 AI 相关的图像处理服务。音视频处理可以直接开通 MCT 产品对 BOS 中的源音视频文件进行免费转码。大数据 Select 扫描可以针对大数据场景下 CSV 以及 JSON 格式文件的内容进行检索，不需要您将文件下载到本地后再检索。
- 数据分发即将 BOS 中存储的数据对外网发布，用户可以通过官方域名或者自定义域名进行发布。您还可以根据需要开通 CDN 加速，利用百度智能云遍布全球的 CDN 节点来加速文件在下行场景中的传输速度。此外，您还可以为 Bucket 开通 CDN 上行加速服务，满足文件在上行场景中的加速上传需求。
- 此外，您还可以在“Bucket 设置”页面对 Bucket 访问权限、跨域访问CORS、Referer 白名单、IP 白名单等进行设置，并在“监控信息”页面查看 Bucket 的读写量及流量信息。

管理Bucket

创建Bucket

概述

存储桶 Bucket 是用来存储 Object 文件的容器。在使用对象存储 BOS 服务之前，您必须先创建 Bucket。

存储桶 Bucket 具有若干配置属性。在创建 Bucket 时，您需要对部分 Bucket 属性进行配置，如 Bucket 名称、所属地域、默认存储类型和读写权限等，同时可以设置 CDN 官方加速，并对标签进行绑定等。

创建Bucket

1. 登录 对象存储 BOS 管理控制台。
2. 点击左侧导航栏 **Bucket 列表**，点击 **创建存储桶** 按钮，在弹出框中按照提示创建 Bucket。



配置项	配置描述
Bucket 名称	• 每个 Bucket 的名称具有全局唯一性。可使用前缀确保名称的唯一性，如以所在的组织的名称作为 Bucket 的前缀。Bucket 创建后，其名称不能更改。 • Bucket 名称只能包含小写字母、数字和"-"，开头结尾为小写字母和数字，且长度在4-63之间。
所属地域	• Bucket 具有区域属性，只能位于一个区域。Bucket 名称创建后不能修改所属区域，因此建议根据业务情况就近存储，方便上传和下载以提高访问速度。 • 如您开通了多区域服务，请根据需要选择 Bucket 所在区域，请参考区域选择说明。
默认存储类型	• 创建 Bucket 时可选择 Bucket 默认存储类型。如果用户使用 API、CLI 或者 SDK 上传的 Object 未指定存储类型，则继承所在 Bucket 的默认存储类型。在 Console 控制台上，上传的 Object 默认为标准存储。当 Object 的存储类型和 Bucket 存储类型不一致时，以Object 的存储类型为准。 • 存储类型包含标准存储-多AZ、标准存储、低频存储-多AZ、低频存储、冷存储和归档存储六种，具体使用场景和性能请参见分级存储。
读写权限	• 私有： Bucket 创建者独享该 Bucket 的全部权限，其他人不能对该Bucket 写入数据，也不可访问读取该 Bucket。适用于存储私密文件的场景。 • 公共读：Bucket 创建者独享该 Bucket 的全部权限，其他人可以访问读取该 Bucket 内的内容，但无权写入。适用于存储需要公众读取的文件的场景，如通知性文件、广告内容等。请注意，若开启该种权限，互联网上的所有用户都可以对该 Bucket 内的文件进行访问读取，请谨慎开启。 • 公共读写：包括 Bucket 创建者在内的任何人都可以访问和写入文件。请注意，若开启该种权限，互联网上的所有用户都可以对该 Bucket 内的文件进行读写操作，请谨慎开启。 • 若需要设置高级权限，则需要在创建完 Bucket 后，修改相应权限设置，具体内容请参考设置 Bucket 权限。

域名加速（CDN 官方加速）	• 开启 Bucket 的 域名加速（CDN 官方加速）后，对应 Bucket 域名的 DNS 解析修改为 CDN 提供的 CNAME 域名，从而接入 CDN 节点，开启 CDN 加速。 • 创建 Bucket 时默认关闭 CDN 官方加速。如果需要开启，可直接在创建时将 " CDN 官方加速 " 设为 "开" 状态。
版本控制	• 默认存储桶中文件（对象）仅有最新版本，开启 Bucket 版本控制后，针对数据的覆盖和删除操作将会以历史版本的形式保留，有效防止数据的被误删除/误覆盖，功能详细说明参见 存储桶版本控制。
分层命名空间	• 默认存储桶无目录结构，开启分层命名空间配置后，将创建具有层级特性的元数据存储桶，适配大数据分析和大数据计算场景，功能详细说明参见 存分层命名空间。
绑定标签	• 可以根据项目和场景需求，您可以为 Bucket 绑定标签，便于对 Bucket 进行分类和识别管理。在绑定标签前，您需要在控制台上先创建标签。
绑定资源分组	• 可以根据项目和场景需求，您可以为 Bucket 绑定资源分组，便于根据业务对 Bucket 进行分类和识别管理。在绑定资源分组前，您需要在控制台上先创建资源分组，或者选择默认分组。 关于资源分组的介绍和使用信息，请查看资源分组官方文档。
其他说明	• 每个用户最多创建 100 个 Bucket，请您谨慎创建。若您创建的 Bucket 数量已经满额，请在清除已创建 Bucket 内的文件后将该 Bucket 删除，否则 Bucket 无法删除。清除已有 Bucket 后，才能创建新的 Bucket。 • Bucket 创建后，并不会立刻计费。只有当 Bucket 产生数据读/写操作后，该 Bucket 才会产生费用，例如您上传了文件，将产生存储空间占用费用。 • 若创建多 AZ 类型 Bucket，则数据只能同步到多 AZ 类型 Bucket，且生命周期无法沉降到单 AZ 类型。

3. 点击 **确定** 后，该 Bucket 创建完成。下一步，您可以在 Bucket 中上传文件。

相关 API

- PutBucket 接口：使用 PutBucket API 接口创建 Bucket。

Bucket概览

Bucket概览

单击左侧导航 **bucket列表** ，单击 Bucket 名称，可在左侧导航栏选择 **概览** 来查看该 Bucket 的概览数据。

概览数据包括：

- **文件数**：该存储中当前存在的对象个数，不包含碎片。
- **用量数据饼状图**：该 Bucket 的存储用量、本月流量、本月请求、数据取回量均以饼状图的形式展示，可选择不同时间范围展示数据取回量；
- **监控数据趋势图**：该 Bucket 的今日存储容量、今日流量监控数据以及昨日智能分层转化容量；
- **配置信息**：该 Bucket 中已配置的部分信息；
- **域名发布信息**：该 Bucket 的域名信息，包括官方域名和自定义域名。

概览页数据指标

Bucket 预览页详细的数据指标如下表所示。

概览模块	数据指标	指标含义
用量数据饼状图	存储用量	• 该 Bucket 总容量，以及标准存储、低频存储、冷存储和归档存储各自的空间占用大小
	本月流量	• 该 Bucket 本月已使用的总流量，包括外网数据流出流量、CDN 回源流出流量和跨区域数据流出流量
	本月请求	该 Bucket 在本月的总请求数、读请求数和写请求数，其中： • 读取次数为读取该 Bucket 的次数，包括所有发给该 Bucket 的 HTTP GET/HEAD/OPTIONS请求次数; • 写入次数为写入该 Bucket 的次数，包括所有发给该 Bucket 的 HTTP PUT/POST/DELETE请求等
	数据取回量	该 Bucket 在所选时间范围内，不同存储类型数据的取回量和归档存储类型数据在不同取回级别下的取回量，其中： • 不同存储类型数据包括低频存储、冷存储和归档存储; • 归档存储取回可分为三个级别，包括加急取回、标准取回和延缓取回
	文件数	• 该 Bucket 包含的全部文件数。若您的 Bucket 中包含文件夹，那么每一个文件夹将被当成 1 个文件进行计算
监控数据趋势图	今日存储用量	• 该 Bucket 在今日标准存储、低频存储、冷存储和归档存储的累计容量走势信息，数据每 5 分钟采集一次
	今日流量	• 该 Bucket 在今日外网数据流出流量、CDN 回源流出流量和跨区域数据流出流量的走势信息，数据每 5 分钟采集一次
	昨日智能分层转化容量	• 若您对该Bucket开启了智能分层，将展示昨日的沉降与回暖数据量
配置信息	配置信息	• 若您对该 Bucket 进行了配置，那么配置的信息将展示在概览页左下方的配置信息模块。您也可以点击该模块上方的 " 配置更多 " 进行去配置更多配置项
域名发布信息	域名发布信息	• 该 Bucket 的相关域名信息，包括官方域名、CDN 加速域名和自定义域名

> **注意：**
>
> - 用量数据饼状图的数据统计有延迟，不作为计量数据，仅做参考。
> - 存储用量饼状图统计中已包含碎片文件。

删除Bucket

概述

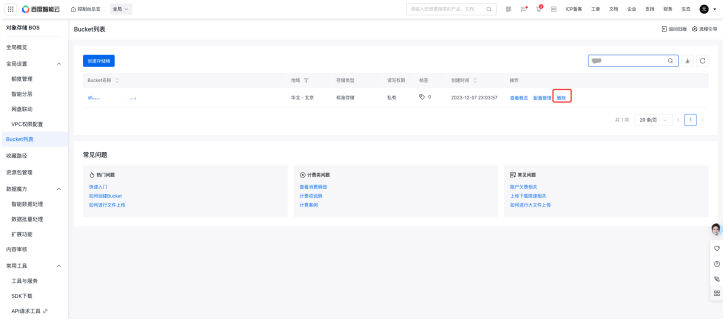
只有当前 Bucket 为空时，即不包含任何 Object 和未完成的碎片（Part），才可删除 Bucket。如果需要直接删除非空 Bucket，可以使用 CLI工具的强制删除命令：`bce bos rb bos:/<bucket-name> --force`，也可以为存储桶配置1天后删除所有的文件以及碎片，等待存储桶为空后可删除存储桶。

注意事项

- 存储空间删除后不可恢复，请谨慎操作；
- 删除存储桶需将其中全部文件、碎片全部删除后方可删除；
- 若您开启了官方加速域名功能需提前关闭；

删除Bucket

1. 登录 对象存储 BOS 管理控制台。
2. 在左侧 **Bucket 列表**页中，单击操作栏中 **删除** 按钮进行删除。



相关 API

- DeleteBucket 接口：使用 DeleteBucket API 接口删除一个 Bucket。

设置Referer白名单

概述

BOS 的 Referer 白名单功能通过设置 Referer 白名单，限制仅白名单中的域名可以访问您 Bucket 内的资源。当您的 Bucket 处于公共读或公共读写的情况下，该设置可以有效防止存储在 BOS 中数据被其他人盗链，保护您的合法权益。

BOS 支持基于 HTTP header referer 字段的防盗链方法。您可以在 BOS 控制台设置 Referer 字段的白名单。设置白名单后，只有 Referer 字段在白名单内的用户才可访问 Bucket 中存储的数据，不在白名单内的请求会被拒绝。但若 Referer 为空时，默认可访问，不受白名单限制。

- 只有在 Bucket 高级权限下才能设置白名单。
- Referer 白名单对访问 BOS 的所有 API 请求均生效。
- Referer 白名单不区分大小写，使用换行符分隔，并支持通配符 (*) 且每个 Referer 只支持 1 个通配符。
- Referer 白名单遵循完全匹配原则。例如： `http://www.baidu.com/abc/` 与`http://www.baidu.com/abc` 在白名单中则被认为是不同的。
- Referer 白名单系统支持检查 http 和 https 两种协议，对于 host 不以 “/” 结尾的，会自动加 “/”。
- 选择 “允许 Referer 为空” 时，HTTP 请求中带白名单中的 Referer 和空 Referer 都允许访问； 选择“不允许 Referer 为空” 时，HTTP 请求只有带白名单的 Referer 可以访问，空 Referer 不允许。

关于 Bucket 权限控制的更多内容请参考 Bucket权限控制。

设置Referer白名单

1. 登录 对象存储 BOS 管理控制台。
2. 单击左侧导航 **Bucket 列表**，点击 Bucket 名称。
3. 在左侧导航栏选择 **基础配置** > **权限配置**。
4. 在 **权限配置** 页面，单击 **修改配置** 。
5. 在 **Bucket 权限配置** 侧边栏中，单击 **自定义** ，添加自定义权限，并对 Referer 白名单进行设置。
   - 每个 Referer 以换行符分隔，每个 Referer 最多支持一个通配符 `*`；
   - 同时可以勾选是否允许 Referer 为空；
   - 选择 “允许 Referer 为空” 时，HTTP 请求中带白名单中的 Referer 和空 Referer 都允许访问；
   - 选择 “不允许 Referer 为空” 时，HTTP 请求只有带白名单的 Referer 可以访问，空 Referer 不允许。



6. 点击 **添加权限** 完成权限配置。
7. 配置完成后，您可以在 **Bucket 权限配置** 中看到已生成的权限记录，并可通过**修改** 和 **删除** 按钮对已有权限进行调整。

设置服务端加密

概述

为了更好地保证数据安全性，BOS 支持在服务端为数据进行 SSE（Server-Side Encryption）加密编码。当您在配置 SSE 加密后，当文件上传时，BOS 会通过携带加密参数的方式来对上传数据执行服务端加密设置。同时，为了提升易用性，BOS 在 Bucket 设置中增设了服务端加密开关，开启之后该 Bucket 内新上传的数据都会默认执行加密，而不影响设置开启前的历史存量数据。

加密方式

目前 BOS 支持两种加密方式：BOS 托管密钥和 KMS 服务托管密钥

- BOS 托管密钥：当选择BOS托管密钥时，所有上传该Bucket的数据都会使用BOS服务托管的密钥进行统一加密保护，用户获取数据时将由BOS服务进行自动解密，用户不需做其他操作，简单方便。

- KMS 服务托管密钥：您需要开通百度智能云密钥管理服务，并创建由自己管理的 KMS 密钥。KMS 中会为根据您创建的 KMS 主密钥生成对应的 Data Key，数据上传时会根据生成的 Data Key 来对数据进行自动加密。

设置服务端加密

1. 登录 对象存储 BOS 管理控制台。
2. 在左侧 Bucket 列表中，单击 Bucket 名称。
3. 在左侧导航栏选择 **基础配置** >**服务端加密配置** 。
4. 在**服务端加密配置** 页面，单击 **修改配置** ，开启加密设置，并选择加密方式与加密算法。

- **加密方式** ：您可以选择 BOS 托管密钥或者 KMS 托管密钥的方式对上传的数据进行服务端加密。
  - 若您选择 BOS 托管密钥，则无需进行其他额外配置，点击确认后即可生效。
  - 若您选择 KMS 托管密钥的方式进行加密，您需要先开通百度智能云密钥管理服务，并创建由自己管理的 KMS 密钥。选择 KMS 服务托管密钥方式，您需要在下拉列表中选择自己在百度智能云 KMS 密钥管理服务中已经创建好的密钥。之后，用户上传该 Bucket 的数据将以用户选择的自定义密钥进行加密保护。同样，当用户读取数据时，BOS 服务会向 KMS 服务申请进行数据解密后，向用户返回明文数据。
- **加密算法** ：您可以配置一种加密算法。
  - AES256
  - SM4（国密）



5. 点击 **确认**，完成服务端加密配置。
6. 您可通过单击**修改配置**，进行加密方式修改或者关闭存储桶服务端加密。

相关 API

- PutBucketEncryption 接口：使用 PutBucketEncryption API 接口开启指定 Bucket 的加密开关。
- GetBucketEncryption 接口：使用 GetBucketEncryption API 接口获取 Bucket 服务端加密是否打开。
- DeleteBucketEncryption 接口：使用 DeleteBucketEncryption API 接口关闭服务端加密功能。

设置跨域访问

概述

跨域资源共享（Cross-Origin Resource Sharing，CORS）是 HTML5 提供的标准跨域解决方案，而跨域访问是指当发起请求的资源所在域不同于该请求所指向的资源所在域时的一种访问形式。出于安全考虑，浏览器会限制这种非同源的访问，但在实际应用中，跨域访问需求是普遍存在的。例如，用户的网站 A （`http://domainA.example`） 后端使用了 BOS 存储，用户想在该网站的 Web 应用程序中引用存储在 BOS 上的资源。而根据 HTML5 协议要求，该引用操作只能向网站域名 `http://domainA.example` 发起请求，向其他网站发送的请求均会被浏览器限制。

BOS 支持 HTML5 协议中的跨域资源共享 CORS 设置，帮助用户实现跨域访问。CORS 通过 HTTP 请求中附带 Origin 的 Header 来表明自己来源域。如上面的例子，Origin 的 Header 就是 `http:// domainA.example`。服务器端接收到这个请求之后，会根据一定的规则判断是否允许该来源域的请求，如果允许的话，服务器在返回的响应中会附带上 Access-Control-Allow-Origin 这个 Header，内容为 `http:// domainA.example` 来表示允许该次跨域访问。如果服务器允许来自任何域名的跨域请求，则可将 Access-Control-Allow-Origin 的Header设置为 * 即可，浏览器根据是否返回了对应的 Header 来决定该跨域请求是否成功，如果没有附加对应的 Header，浏览器将会拦截该返回结果。如果是非简单请求，浏览器会先发送一个 OPTIONS 请求（预请求）来探测这个跨域请求对于服务器端是不是安全可接受的，如果服务器不支持接下来的操作，浏览器会拦截该跨域请求。 更多关于 CORS 的介绍可以参考跨域资源共享。

设置跨域访问

1. 登录 对象存储 BOS 管理控制台。
2. 单击左侧 **Bucket 列表** ，点击 Bucket 名称。
3. 在左方导航栏单击进入 **基础管理** > **跨域访问CORS配置** 。
4. 在 **跨域访问CORS配置** 页面点击 **新增配置** 对该 Bucket 的跨域访问进行配置。

设置参数说明

名称	描述
Origins	指定允许的跨域请求的来源。Origin 可以设置多个，每行 1 个，每行最多 1 个*符号，* 表示允许所有来源的跨域请求，还支持 *作为后缀来表示某一类网站，例如abc * ，表示支持abc开头的网站
Methods	指定允许的跨域请求方法
Headers	指定允许的跨域请求 Header。Header 可以设置多个，每行 1 个，每行最多 1 个 *符号
ExposeHeaders	指定允许用户从应用程序中访问的响应头（例如 1 个 Javascript 的 XMLHttpRequest 对象） ExposeHeader 可以设置多个，每行 1 个，不能出现*符号
maxAgeSeconds	指定浏览器对特定资源的预取（OPTIONS）请求返回结果的缓存时间

注意事项

- CORS 只针对浏览器环境有意义，相关的 Header 附加由浏览器自动完成，不需要用户手动操作。
- CORS 请求与 BOS 的身份验证是完全独立的，即 CORS 规则仅仅是用来决定是否附加 CORS 相关的 Header 的一个规则，是否拦截该请求完全由浏览器决定。
- 使用跨域请求的时候需要关注浏览器是否开启了 Cache 功能。当运行在同一个浏览器上分别来源于不同域 `http://domainA.example` 和 `http://domainB.example` 的两个页面都同时请求同一个跨域资源的时候，如果 `http://domainA.example` 的请求先到达服务器，服务器将资源带上Access-Control-Allow-Origin 的 Header 为 `http://domainA.example` 返回给用户。这个时候 `http://domainB.example` 又发起了请求，浏览器会将 Cache 的上一次请求结果返回给用户，这个时候 Header 的内容和 CORS 的要求不匹配，就会导致后面的请求失败。



5. 点击 **确认**，完成配置。
6. 添加 CORS 规则后，点击列表右侧的 **修改**，可修改 CORS 规则。
7. 添加 CORS 规则后，若您希望删除该条规则，可点击列表右侧的 **删除**，可以删除单条 CORS 规则。

相关 API

- PutBucketCors 接口：使用 PutBucketCors API 接口在指定的 Bucket 上设定一个跨域资源共享（CORS）的规则。
- GetBucketCors 接口：使用 GetBucketCors API 接口获取指定的 Bucket 当前的CORS规则。

- DeleteBucketCors 接口：使用 DeleteBucketCors API 接口关闭指定 Bucket 的 CORS 功能并清空所有规则。

设置数据同步

概述

BOS 提供数据同步功能，您可以在处于不同地域的 2 个 Bucket 之间建立数据的自动同步关系。当您设置好源 Bucket 和目标 Bucket 之后，BOS 将自动执行从源 Bucket 到目标 Bucket 间的数据同步。您还可以指定源 Bucket 内需要同步的数据范围，并设置是否同步历史数据。

数据同步功能能够很好的提供 Bucket 跨区域、跨账号或其他数据复制的需求。目标 Bucket 中的对象是源 Bucket 中对象的精确副本，它们具有相同的对象名、元数据以及内容，例如创建时间、拥有者、用户定义的元数据、Object ACL、对象内容等。

使用场景

您可能基于各种原因对 Bucket 配置数据同步，这些原因包括：

- **高效访问，减少延迟**：跨 Region 访问 Bucket 或者 Object，访问速度会因地理上的差异而降低。通过数据同步功能可提前将需要的数据同步至目标 region，提高访问效率。
- **跨地域容灾**：基于金融、政府等行业的合规性要求，在同一 Region 中虽然有数据多副本设置。但为了防止洪水、地震等机房级别灾害，也要求数据跨 Region 保存一份副本，BOS 的数据同步功能提供了这样的能力。
- **跨账号复制**：适用于企业的多账号场景。处于数据安全和多备份考虑，企业希望将数据从 A 账号定期同步到 B 账号，以实现数据的跨账号复制和共享。
- **跨区域数据复用**：由于业务原因例如跨 Region 搭建的计算集群需要使用同一组数据等场景，可以使用 BOS 数据同步功能复制数据副本。

设置数据同步

1. 登录 对象存储 BOS 管理控制台。
2. 单击左侧 Bucket 列表 ，点击 Bucket 名称进入 Bucket 管理目录。
3. 在左方导航栏单击进入 **高级配置** > **数据同步配置** 。
4. 在 **数据同步配置** 区域点击 **新增配置** 对该 Bucket 的数据同步进行配置。
5. 在数据同步配置操作栏中，选择数据同步策略、数据同步对象、历史数据是否同步、同步目标区域、目标Bucket以及目标存储类型。



配置模块	配置说明
规则基本信息	• 规则名称：该规则的名称，最多20个字符，支持大小写字母，数字及-_特殊字符。 • 规则状态：是否创建时立即开启。若创建时选择关闭，可以后续在规则列表中开启。规则开启后将立即开启数据同步。 • 数据同步策略：若选择全量同步策略，则源 Bucket 文件发生新增/修改/删除操作的文件都将同步到目的 Bucket；若选择写同步，则只有源 Bucket 文件发生新增/修改操作时会同步到目的 Bucket，删除文件不会同步。
数据源设置	• 数据同步范围：可以将 Bucket 内全部文件同步到目标 Bucket，或者指定源 Bucket 文件前缀。一条数据同步规则最多设置 20 个前缀。 • 指定不同步范围：可指定不同的文件范围，支持以下三种匹配样式。样式一：**prefix/***，指定前缀为prefix/；样式二：***.suffix**，指定后缀为.suffix；样式三：***infix***，指定中缀为infix。 • 历史数据：是否开启源 Bucket 历史数据同步。历史数据同步同样根据前缀进行匹配，若选择开启且已指定文件名前缀，那么仅同步符合前缀规则的历史数据；若选择关闭历史数据同步，那么该规则生效后只同步源 Bucket 中的新增数据。
数据目标设置	• 目标区域：BOS 支持跨区域数据同步，您可以选择目标 Bucket 所在的区域。 • 目标 Bucket：选择或输入目标 Bucket 名称。目标 Bucket 仅下拉展示您当前账号下的 Bucket 列表。若您希望实现**跨账号复制**，您可以直接输入其他账号下的目标 Bucket 名称，但 BOS 不会为您获取其他账号的 Bucket 列表。若您输入的目标 Bucket 归属于其他账号，请仔细确认 Bucket 名称准确无误，并确保目标 Bucket 已为源账号开启写权限。权限设置请查看设置Bucket权限文档。 • 目标前缀：您可以选择文件在目标 Bucket 中的存储位置。若输入的前缀不存在，目的端 Bucket 将自动生成新的前缀。前缀不支持连续的/符号，若您输入的前缀中包含连续/如///，则/只保留一个。 • 目标存储类型：您可以设置文件在目的 Bucket 中的存储类型。目标 Bucket 的存储类型可以选择和源对象相同，也可以根据需要选择其它的存储类型。多 AZ 类型 Bucket 仅能同步到多 AZ 类型 Bucket，无法同步到单 AZ 类型 Bucket；单 AZ 类型 Bucket 仅能同步到单 AZ 类型 Bucket，无法同步到多 AZ 类型 Bucket。如果有单 AZ 类型 Bucket 同步到多 AZ 类型 Bucket，或多 AZ 类型 Bucket 同步到单 AZ 类型 Bucket 的需求，可以使用 数据流转平台 CloudFlow。

说明：

- 数据同步的两个 Bucket 可以是**跨区域**的，也可以是**同区域**的。只有跨区域 Bucket 间的数据同步会触发流量费用，同区域 Bucket 间的数据同步，不会收取流量费用。
- 数据同步的两个 Bucket 可以是**跨账号**的，也可以是**同账号**的。请您仔细确认输入的目标 Bucket 名称准确无误。若目标账号中的目标 Bucket 处于其他区域，将产生跨区域流量费用，流量费用由源账号承担，目标账号承担存储和请求费用。
- 支持 Bucket 间的**多规则同步**，即一个源 Bucket 可以有多个目的 Bucket，一个目的 Bucket 可以有多个源 Bucket。假设有三个 Bucket，分别命名为 A、B、C：（1）支持将 A 作为 B 的源 Bucket 的同时，将 B 作为 C 的源 Bucket；（2）支持将 A 作为 B 的源 Bucket 的同时，将 B 作为 A 的源 Bucket；（3）支持将 A 同时作为 B 和 C 的源 Bucket；（4）支持将 C 同时作为 A 和 B 的目标 Bucket。
- 对非标准存储 Object 进行数据复制操作时，会触发数据取回费用。对于归档型 Object，数据同步暂时不支持，进行数据同步时会忽略归档型 Object。
- 对于处于同步状态的两个 Bucket，源 Bucket 复制过去的 Object 可能会覆盖目标 Bucket 中同名的 Object，使用中请注意。
- 由于数据同步采用异步复制，数据复制到目标 Bucket 需要一定的时间，通常几分钟到几小时不等，取决于数据的大小。
- 出于数据安全考虑，BOS 不会获取并返回跨账号同步的启动状态。若您已开启跨账号同步，建议您主动查询目的 Bucket 中是否已经开始数据同步进程。若源数据在 5 分钟内还未同步到目的账号 Bucket 中，请再次检查写权限是否开启。若写权限已开启后仍同步失败，请提交工单。
- 历史数据同步无法保证同步速度，如果着急使用，建议使用bcecmd工具拷贝Object功能进行同步。
- 如果使用数据同步的目的是用来复制迁移数据到其他桶或账号中，且规则中的数据同步策略设定为全量同步时。当数据完成同步之后，如需删除源端的数据，在删除源端数据前请切记关闭数据同步规则，避免目标端的数据被同步误删除。
- 如果数据同步规则已经开启同步历史数据，修改规则会导致已经同步过的历史数据再次同步。

6. 点击 **确认**，完成配置。

查看同步进度

完成数据同步规则配置后，您通过控制台点击该条同步规则【操作】中的【详情】按钮，查看历史数据同步进度与实时同步时间点与详细的数据同步规则配置信息。

- 历史数据同步：您可以查看历史文件同步进度，掌握同步进展。
- 实时数据同步：您可以查看最近增量文件的数据同步时间点信息。



相关 API

- PutBucketReplication
- GetBucketReplication
- DeleteBucketReplication
- GetBucketReplicationProgress
- ListBucketReplication

注意事项

数据跨境合规承诺 您使用该产品或该功能，将使您在云上的业务数据传输至您所选择的区域或产品部署区域，可能会涉及数据跨境。

使用该功能代表您同意并确认，您完全拥有对相关业务数据的合法处置权限，并对您的数据传输行为或指令行为全权负责。您确认并承诺，您的数据传输符合所有适用法律的规定，包括并不限于获得数据主体充分、明示同意，完成数据出境安全评估，与接收方签署个人信息出境标准合同等法定义务，且您承诺您的业务数据不含任何所适用法律限制、禁止传输或披露的内容（具体合规要求请您咨询相关主管部门）。

如您未遵守前述声明与保证，您将承担全部法律后果，并赔偿百度智能云和/或其关联公司遭受的全部损失。

如因法律法规、监管政策规定导致百度智能云须变更或停止提供本产品功能，您理解并同意这并不属于百度智能云违约。百度智能会为您提供服务变更或停止的过渡期协助。

设置镜像回源

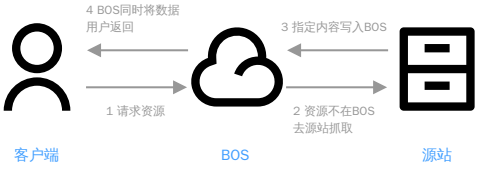
概述

若您为 Bucket 配置了镜像回源，当用户访问该 Bucket 中某一 Object 时，发现该 Object 在 BOS 中并不存在，则 BOS 会立刻向预先配置好的回源地址请求该 Object，保证访问正常进行。在请求过程中，源站返回的数据会自动给到请求用户而用户对该过程无感。同时，该数据也会自动存入 BOS，提升日后该类请求的流畅性。

使用场景

镜像回源主要用于数据无缝迁移到BOS的场景。例如：源站有一批冷数据，同时在不断生成新的热数据。

1. 您可以先通过 BOS CMD工具 或其他迁移工具（如果数据量较大可以提交工单），将数据迁移到 BOS 中，同时为相应的 Bucket 配置镜像回源规则；
2. 直接把业务域名切换成 BOS 的 Bucket 域名。虽然还有新生成的部分热数据尚未迁移到 BOS 中，但用户依然可以正常从 BOS 访问（BOS一边从源站读取数据，一边将数据存入BOS中），并且访问一次后数据就会存入到BOS；
3. 当源站无新的热数据产生时，此时可以扫描一次源站，把残余数据导入 BOS。完成之后，就可以关闭镜像回源设置。



使用限制

- 规则数量：可最多设置**20**条镜像回源规则，多条规则之间不能设置包含关系的前后缀，若设置全部文件回源，则仅能设置一条规则；
- 回源地址：可配置第三方回源地址或者您的私有Bucket。第三方回源地址访问协议支持HTTP/HTTPS 协议，可以使用域名或 IP ，支持加端口。如果回源地址中不带协议，则默认使用 HTTP 协议，私有Bucket仅支持HTTP协议；
- 支持地域：所有地域都支持。

回源规则

触发条件

- 只有当httpcode返回404的情况下，BOS才会执行镜像回源，向源站请求文件；

访问源站

- BOS 在进行镜像回源时，默认不会携带原请求中的 QueryString与Header ，但是您可以主动设置传递QueryString和设置header传递规则；
- BOS 不支持对部分标准的 HTTP header 进行设置，如 Content-Length、Authorization、Range、Date、Content-Md5、Host，以及 x-bce- 的 header；

回源文件命名规则

- 回源返回的文件将以请求源站时使用的文件名存入 BOS 中。若 BOS 向源站请求的URL为http://huiyuan.com/myobject, 则回源到 BOS 的文件名默认为myobject。例如某Bucket设置的回源地址为http://huiyuan.com, 某用户请求的文件myobject.docx不在该Bucket中。则 BOS 会通过http://huiyuan.com/myobject.docx 获取文件，存储到 BOS 的文件名为myobject.docx；

回源请求失败返回规则

- 如果源站返回404状态码，那么BOS也会返回404给用户。

回源文件元信息 BOS会将源站返回的以下HTTP头存为BOS文件的元信息：

```
Content-Type
Content-Encoding
Content-Disposition
Cache-Control
Expires
Content-Language
Access-Control-Allow-Origin
```

设置镜像回源

1. 登录 对象存储 BOS 管理控制台。
2. 单击左侧 Bucket 列表 ，点击 Bucket 名称。
3. 在左方导航栏单击进入 **基础配置** > **镜像回源配置** 。
4. 在 **镜像回源配置** 区域点击 **新增配置** 对该 Bucket 的镜像回源进行配置（详见配置项说明）。
5. 单击确定，完成镜像回源配置；
6. 配置完成后，您可以在 **镜像回源配置** 中看到已生成的回源规则记录，您可查看配置详情，若您希望修改该配置，可以点击 **编辑** 进行修改；若您希望删除该配置，可以点击 **删除，删除后将停止回源，请您谨慎操作**。

配置项说明

1. 基本配置 1.1 回源方式

- **镜像**：镜像回源模式下，您发出的请求在 BOS 中没找到对应文件时，会自动到源站抓取对应文件保存到 BOS，并将内容直接返回给您；
- **重定向**：重定向模式下，满足响应条件的请求会通过 HTTP 重定向的方式返回重定向的地址，浏览器或客户端再到源站获取内容。同时BOS会异步执行到源站抓取对应文件保存到 BOS； **1.2 回源条件**
- **HTTP 状态码 404**：默认当 HTTP 状态码为404时触发回源；
- **文件名前缀**：
  - 当请求的文件名前缀匹配时，可触发回源规则。例如：您输入的前缀为aa，则访问http://mybucket.com/aa/mypic.jpg 且返回的 HTTP 状态码为404时，会触发回源规则；
  - 前缀不能以/结尾；
- **文件名后缀**：
  - 当请求的文件名后缀匹配时，可触发回源规则。例如：您输入的后缀为.jpg，则访问http://mybucket.com/mypic.png 且返回的 HTTP 状态码为404时，会触发回源规则；



2. 主源站设置

2.1 主源站类型

- **BOS私有Bucket**：若您需要回源到私有bucket，请勾选此项，并在下拉框中选择目标Bucket。源站是您账号下的另一个 Bucket，回源可能会产生回源 Bucket 的外网下行流量；
- **替换或裁剪前缀**：您可以灵活的配置BOS访问您配置源站的url，裁剪与替换均是对您的原始访问url的文件路径部分进行处理，若您的访问地址为：mybucket.bj.bcebos.com/abc/def/myobject.docx，配置**裁剪**后，将/abc/def裁剪，回源地址为：您配置的回源域名/myobject.docx；**替换**可对访问地址进行改写，若您配置替换，则必须输入需要替换路径信息，例如：配置替换为123，则回源地址为：您配置的回源域名/123/myobject.docx。为了保证回源地址正确，请谨慎配置并验证控制台中回源地址是否符合预期。 **2.2 回源地址**
- **访问协议**：默认http，若您回源到第三方源站，您可以下拉选择https；
- **回源域名**：需填入回源访问的域名或者ip地址&端口号，若您选择回源到私有Bucket，可以直接进行选择；
- **固定文件名称**：仅重定向方式支持。您可以勾选固定文件名称，并填写文件名，符合规则的文件触发回源规则，统一返回客户端固定文件名；
- **前缀**：填写回源地址的前缀，触发回源规则时，跳转至指定前缀的文件。例如指定前缀为test，访问http://mybucket.com/objectname.doc时触发回源规则，则会跳转至 <回源地址>/test/objectname.doc；
- **后缀**：填写回源地址的后缀，触发回源规则时，跳转至指定后缀的文件。例如指定后缀为.doc，访问http://mybucket.com/objectname.doc时触发回源规则，则会跳转至 <回源地址>/objectname.doc；



3. 备源站设置

- **设置备源站**：
  - BOS支持设置主备源站，以提高镜像回源的可用性和回源的成功率。当您开启备源站后，若主源站回源失败，则BOS会向备源站发起回源请求；
  - 仅镜像回源方式支持配置；
- **回源地址**：* 备源站与主源站回源规则相同，支持域名或 IP 地址&端口号



4. 其他设置 您可以配置BOS是否传递**QueryString**信息和**HTTP header**给源站，和回源文件存储类型。 **4.1 Query参数携带**

- **请求携带Query字符串**：若您勾选请求携带Query字符串，会将 BOS 请求中的 queryString 传递到源站； **4.2 回源文件存储类型**
- **回源文件存储类型**：回源站拉取的文件，将按照您指定的存储类型进行存储至BOS。若您设置回源规则的bucket为单AZ类型，您可以选择回源文件存储类型为标准存储、低频存储、冷存储、归档存储；若您设置回源规则的bucket为多AZ类型，您可以选择回源文件存储类型为标准存储-多AZ、低频存储-多AZ； **4.3 HTTP header**
- **允许传递**：BOS 在进行镜像回源时，默认不会携带原请求中的 header ，但是您可以主动设置 header 传递规则。您可以选择允许：
  - 传递所有 HTTP header 参数
  - 传递指定 HTTP header 参数，单条规则最多10个
- **禁止传递**：您可以禁止传递指定 HTTP header 参数，单条规则最多10个；
- **指定参数设置**：您可以设置指定 HTTP header 参数、参数值，单条规则最多10个；



配置示例

地址示例

假设访问mybucket存储桶中的文件，文件名为myobject.docx，则访问地址默认为：mybucket.bj.bcebos.com/myobject.docx
若访问地址：mybucket.bj.bcebos.com/myobject.docx （初始地址）
回源地址：http://回源域名/myobject.docx（初始地址）

情况1：若回源条件中，您填写了文件名前缀为abc，则访问地址和回源地址为：

若访问地址为：mybucket.bj.bcebos.com/abc/myobject.docx

回源地址：http://回源域名/abc/myobject.docx

情况2:若回源条件中，您填写了文件名前缀为abc/aa，则访问地址和回源地址为：

若访问地址为：mybucket.bj.bcebos.com/abc/aa/myobject.docx

回源地址：http://回源域名/abc/aa/myobject.docx

情况3:若在情况1的基础上，您继续配置替换文件名前缀为tihuan/edf，设置回源协议选择http、回源域名为www.myweb.com，配置文件后缀为.doc，则访问地址和回源地址为：

若访问地址为：mybucket.bj.bcebos.com/abc/myobject.docx

回源地址：http://www.myweb.com/tihuan/edf/myobject.doc

情况4:若在情况2的基础上，您继续设置裁剪文件名前缀，并配置回源协议选择http、回源域名为www.myweb.com，设置前缀为qianzhui/，则访问地址和回源地址为：

若访问地址为：mybucket.bj.bcebos.com/abc/aa/myobject.docx

回源地址：http://www.myweb.com/qianhui/myobject.docx **流程示例一：** 您有名为bucketA的Bucket，需实现如下场景：

- 当请求者访问bucketA中任意不存在的文件时，可以从https://myweb.com 站点的myobject目录中回源文件。 要满足以上场景需求，请参照操作指引配置以下镜像回源规则：

参数	配置
回源方式	• 选择镜像；
回源条件	• 无需配置文件名前后缀，则代表全部文件 ；
主源站设置	• 勾选设置前后缀替换，并填写前缀 myobject ； • 主源站设置，设置为 http ，回源域名设置为 myweb.com ；

以上规则配置完成后访问流程如下：

1. 请求者首次访问bucketA想要获取中任意文件；
2. 如果bucketA的中不存在该文件，则BOS会向 https://myweb.com 站点的myobject目录请求文件；
3. 第三方源站https://myweb.com 会将文件返回BOS，若没有该文件最终会则返回404。 **流程示例二：** 您有名为bucketA的Bucket，需实现如下场景：

- 当请求者访问examplefolder目录中不存在的文件时，可以从https://myweb.com 站点的myobject目录中回源文件。

要满足以上场景需求，请参照操作指引配置以下镜像回源规则：

参数	配置
回源方式	• 选择镜像；
回源条件	• 选中文件名前缀，并设置为 examplefolder ；
替换	• 选中替换，并填写 myobject ；
主回源地址	• 第一列设置为 http ，回源域名设置为 myweb.com 。
备源站地址	• 开启 ，并配置备回源域名设置为 backup.com 。

以上规则配置完成后访问流程如下：

1. 请求者首次访问bucketA想要获取examplefolder目录下文件；

2. 如果bucketA的examplefolder目录中不存在该文件，则BOS会向 https://myweb.com 站点的myobject目录请求文件；

3. 第三方源站https://myweb.com 会将文件返回BOS，若第三方源站中也无该文件，将继续访问备源站backup.com，若均没有该文件最终会则返回404。

流程示例三： 您有名bucketA和bucketB的两个Bucket。其中bucketA为公共读，bucketB为私有，需实现如下场景：

- 当bucketA根目录下folderA目录中不存在的文件时，可以从bucketB的folderB目录中获取目标文件。
- 允许将请求URL中携带的query string传递到源站。
- 允许将请求URL中携带的header1、header2 两个HTTP header传递到源站。 要满足以上场景需求，请参照操作指引配置以下镜像回源规则：

参数	配置
回源方式	• 选择镜像；
回源条件	• 选中文件名前缀，并设置为 folderA ；
主源站类型	• 选中回源BOS私有Bucket，并在回源Bucket下拉列表选择 bucketB ；
替换或裁剪前缀	• 选中替换，并填写 folderB ；
主回源地址	• 第一列设置为 http ，回源域名选择Bucket的endpoint ；
Query参数携带	• 选中请求携带Query字符串；
HTTP header	• 选中允许传递指定HTTP header，并添加header1、header2两个HTTP header。

以上规则配置完成后访问流程如下：

1. 请求者首次访问bucketA想要获取example.docx文件；
2. 如果bucketA中不存在folderA/example.docx文件，则BOS会向bucketB的folderB目录请求文件；
3. bucketB会根据传递的参数统计访问信息，并将example.docx返回BOS（若bucketB中也无example.docx文件，则返回404）；
4. BOS会将获取的文件命名folderA/example.docx，并将命名后的文件存储bucketA；
5. 如果请求中还携带了**header1**、**header2**的HTTP header时，也会被传递到bucketB。

设置访问日志

概述

当用户需要跟踪 BOS 的访问请求时可以开启 BOS 日志功能。日志记录功能可以应用在访问统计和安全审核等方面，每条访问日志记录了单个访问请求的详细信息，包括请求者、Bucket 名称、请求时间和请求操作等信息，访问日志格式描述请参见设置访问日志。当一个 Bucket 开启访问日志功能后，会自动将对这个 Bucket 的访问请求，以小时为单位，按照固定的命名规则，生成日志文件写入用户指定的 Bucket。

说明：

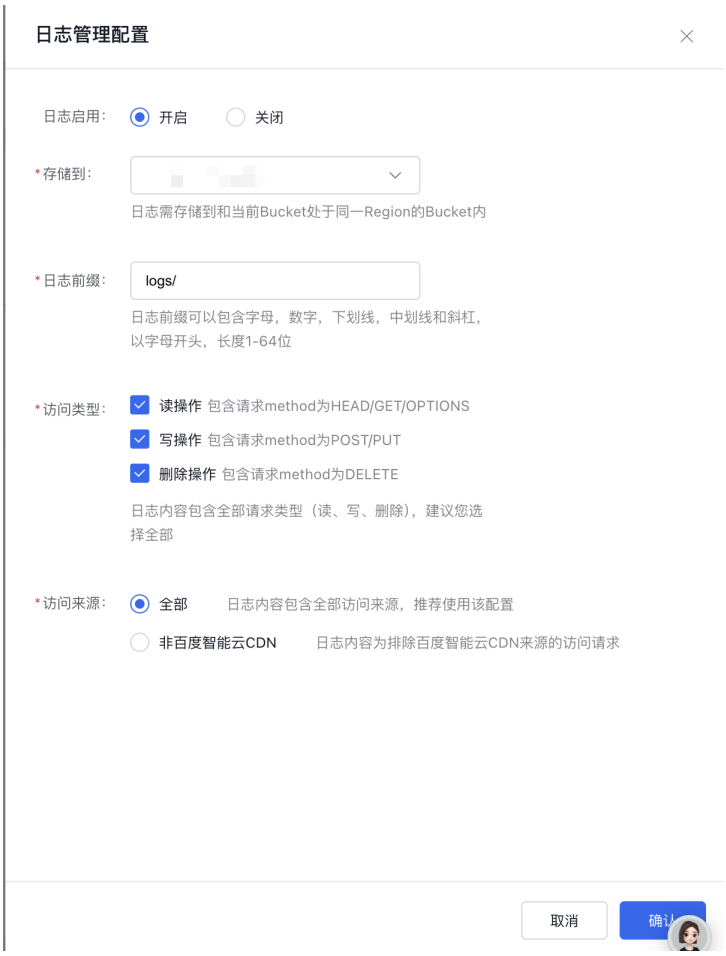
- 访问日志存储投递目标 Bucket存在一定延时。大多数情况下，一个访问请求从发生，到对应日志存储到目标 Bucket 会在 2-3 小时内完成。但受到网络环境等因素影响，有的日志可能会延迟较长时间才能存储到目标 Bucket。
- 日志文件与普通文件性质相同，BOS 无法对其进行清除，只能您主动删除不需要的日志文件。若您一直未删，则日志文件一直保留。如果您有定期删除老旧日志文件的需求，建议配置生命周期规则来删除过期文件。

设置访问日志

1. 登录 对象存储 BOS 管理控制台。
2. 点击 **Bucket 列表** ，进入存储桶列表页，点击需要配置的 Bucket 名称进入 Bucket 详情页，您可通过页面存储桶下拉框进行存储桶便捷切换。
3. 选择 **基础配置** 下 **日志管理配置** ，对该 Bucket 的日志管理进行配置，默认情况下日志功能不开启。 在日志管理配置页面中，点击 **修改配置**，**日志启动** 状态设置为开启，则可进行生成与投递规则进行配置。

配置参数说明

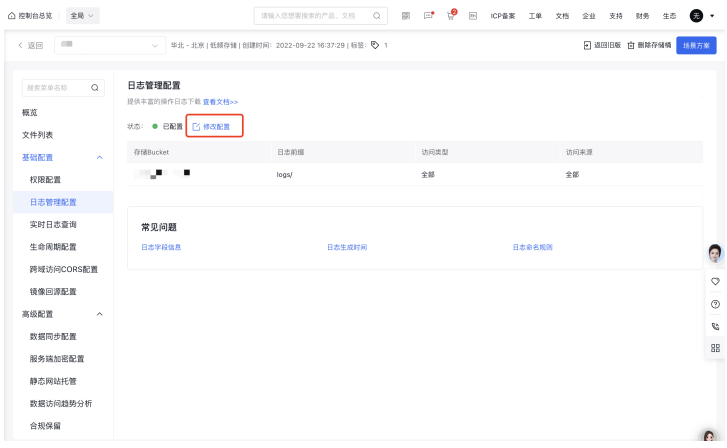
参数	说明
日志投递存储桶	设置日志生成后投递至的目标存储桶信息，需注意目标存储桶需满足与日志生成存储桶相同地域，您可配置与日志源桶或者其他同地域存储桶进行存储日志
日志文件前缀	设置日志生成后投递的路径信息。前缀可以包含字母，数字，下划线，中划线和斜杠，以字母开头，长度1～64字符
访问类型	默认日志将包含全部请求类型，即读/写/删除操作，读操作：包含请求method为HEAD/GET/OPTIONS，写操作：包含请求methodPOST/PUT；删除操作：包含请求method为DELETE，日志内容您可按需生成，**推荐您选择全部操作类型**
访问类型	默认日志将包含全部请求类型，即读/写/删除操作，读操作：包含请求method为HEAD/GET/OPTIONS，写操作：包含请求methodPOST/PUT，批量删除操作请求method为POST，**特别提示** 如需记录批量删除操作，请选择写操作；删除操作：包含请求method为DELETE，日志内容您可按需生成，**推荐您选择全部操作类型**
访问来源	可用于区分请求类型，包含2种配置：全部访问 与 排除百度智能云CDN服务请求，默认为全部来源，你可排除百度智能云CDN的请求日志



注意：

- 日志存储的目标 Bucket 和源 Bucket必须在同地域；
- 日志功能无需额外的收费，只收取日志文件产生的存储费用。日志文件传输不会产生数据传输费，但对已生成的日志文件的访问则与任何其他数据传输同等计费；
- 访问来源默认包含全部来源，建议您保持该配置。
- 访问类型配置，因批量删除操作请求 method 为 POST ，如您需记录批量删除操作，请选择勾选写操作，推荐您采用默认全部访问类型配置

4. 单击 **确定**，保存配置。您可在当前页面查看相应信息并进行修改。



相关 API

- PutBucketLogging 接口：使用 PutBucketLogging API 接口开启 Bucket 的访问日志并指定存放日志的 Bucket 和访问日志的文件前缀。

- GetBucketLogging 接口：使用 GetBucketLogging API 接口获取某个 Bucket 的访问日志配置情况。
- DeleteBucketLogging 接口：使用 DeleteBucketLogging API 接口关闭 Bucket 访问日志记录功能。

设置静态网站托管

概述

BOS 支持用户将静态网站托管在 Bucket 上，实现网站轻量化运维。设置生效后，用户可通过直接访问该Bucket的域名访问该托管网站。

静态网站是指网站内容全部为 HTML、JPG 等静态化资源，不包含诸如 PHP、JSP 或 ASP.NET 等服务器端脚本。同时，静态托管的网站也不支持在服务器端进行脚本编写。

如需部署及管理动态网站，请使用百度智能云服务器BCC或容器服务CCE。

> **说明:**
> 出于安全考虑，对象存储（BOS）所有地域自2022年9月15日起，通过浏览器访问 BOS 官方域名与 CDN 官方加速域名中的网页类型文件（mimetype为text/html，扩展名包括HTM、HTML）将变更为以附件的形式进行下载。具体变更如下：
>
> - 使用 BOS 官方域名访问时，Response Header中会自动加上 `Content-Disposition:'attachment=filename;'`。即从浏览器访问网页类型文件时，不会显示文件内容，而是以附件形式进行下载。查看官方域名信息请见查看Bucket官方域名。
> - 使用 CDN 官方加速域名访问BOS时，Response Header中会自动加上`Content-Disposition:'attachment=filename;'`。即从浏览器访问网页类型文件时，不会显示文件内容，而是以附件形式进行下载。
> - 使用自定义域名访问 BOS 时，Response Header中不会加上`Content-Disposition:'attachment=filename;'`，您可以直接通过浏览器预览网页类型的文件，添加自定义域名的方法详见创建Bucket自定义域名。

业务使用方式

对于待托管的静态网站，您需要先在 Bucket 中上传关键文件资源："索引页面"对应的文件和"404页面"对应的文件。

索引页面

一个标准的网站通常会具备若干索引页面，对应网站的首页及子模块的首页。当用户在浏览器地址栏访问根目录如 `www.example.com`，或以 / 为结尾的目录如`www.example.com/folderA/`时，未请求到特定的页面，BOS则会返回索引页面。

网站管理员可在 Bucket 的根目录及子目录上传静态资源，用于访问索引页面时的效果展现。BOS 当前支持以 html 格式的文件作为索引页面。

404 页面

当静态网站在访问遇到常见的 404 错误时，一般网站都会提供一个经过设计的报错页面，以此给予网站访问者较好的体验。对于在 BOS 托管的静态网站，网站管理员可在 Bucket 根目录下上传 html、jpg、png、bmp、webp 格式的文件作为 404 页面，当网站访问者预期查看的数据找不到时，BOS 将默认展示此 404 页面。

> **注意：**
>
> - 索引页和 404 页配置的文件需为公共读，允许被匿名访问。
> - 如果您对预期做静态网站托管的 Bucket 需设置为公共读权限（不推荐），用于匿名用户的访问。建议不要将机密数据上传至此 Bucket。
> - 请不要为 404 页面设置 meta "Content-Disposition:attachment"。

当管理员在 Bucket 内上传索引页面及 404 页面对应的资源后，可通过 BOS 管理控制台进行索引页面及 404 页面的配置。

设置静态网站托管

1. 登录 对象存储 BOS 管理控制台。
2. 在左侧 Bucket 列表中，单击 Bucket 名称。
3. 在左侧导航栏选择 **基础配置** >**静态网站托管配置** 。
4. 在**静态网站托管配置** 页签，点击 **修改配置** 将 **托管状态** 变更为 **开启**，可对 **索引页面** 和 **404页面** 进行配置。
   - **索引页配置**用于配置访问根目录或者任何一级目录时，默认返回的索引文件
   - **404页面配置**。访问Bucket内文件出现404时，BOS返回的错误页面,文件需放置在存储桶的根目录下， 响应码用于配置返回错误文档时的HTTP响应码为404或200，默认为 404。



> **注意：**
>
> - 索引页面的名称可根据网站管理习惯任意指定，如“index.html”或“admin.html”等，但各级索引页面对应的资源名称需统一为一个相同的名字；
> - 索引页面和404页面至少需填写一个否则配置无法生效；
> - 如果设置的托管页面为归档类型，配置将不会生效；
> - 当管理员填写的与页面匹配的资源在 Bucket 中不存在时，按 404 处理；如 404 页面对应的资源也不存在时，则返回原始的 404 报错页面；
> - 当用户为此 Bucket 同时打开静态网站托管及镜像回源后，当用户预期访问的资源不存在时，BOS 将先尝试进行镜像回源，如仍然无对应资源，则会遵循静态托管的逻辑返回 404 页面。

配置举例

用户在 bj 区域创建了名为 website 的 Bucket，并在其中放置如下文件：

- 在 Bucket 根目录放置 index.html，并设置为公共读
- 在 Bucket 根目录放置 404.html，并设置为公共读
- 在 Bucket 二级目录 website/car 放置 index.html，并设置为公共读
- 在 Bucket 二级目录 website/car 放置 apollo.jpg，并设置为公共读

当用户将该 Bucket 开启静态网站托管功能后，该网站访问者的访问行为定位如下：

- 访问 www.website.bj.bcebos.com 时，页面呈现 index.html
- 访问 www.website.bj.bcebos.com/car/ 时，页面呈现 index.html
- 访问 www.website.bj.bcebos.com/car/apollo.jpg 时，页面呈现 apollo.jpg
- 访问 www.website.bj.bcebos.com/car/dazhong.jpg 时，因不存在此资源，页面呈现 404.html

相关API

静态网站托管功能也支持通过 API 方式设置，具体可参见如下接口说明：

- PutBucketStaticWebsite 接口：使用 PutBucketStaticWebsite API 接口设置静态网站托管。
- GetBucketStaticWebsite 接口：使用 GetBucketStaticWebsite API 接口获取 Bucket 的静态网站托管信息。
- DeleteBucketStaticWebsite 接口：使用 DeleteBucketStaticWebsite API 接口删除 Bucket 设置的静态网站托管信息，并关闭此 Bucket 的静态网站托管。

设置回收站

概述

为了提高 BOS 数据的可靠性，用户可以通过配置回收站的功能，保证删除后的数据在回收站内继续保留，用于后续对删除数据的找回。

> **注意**
>
> - 进入回收站的文件仍然会继续计费，直到您彻底删除。如果您有定期删除老回收站内文件的需求，建议配置生命周期规则来删除过期文件。

设置回收站

1. 登录 对象存储 BOS 管理控制台。
2. 在左侧 Bucket 列表中，单击 Bucket 名称。
3. 在左侧导航栏选择 **高级配置** > **回收站配置** 。

4. 在 **回收站** 区域点击 **修改配置** 对该 Bucket 的回收站进行配置。
5. 在 **回收站配置** 操作栏中，开启回收站。
   开启回收站功能后，在该 Bucket 中删除的文件会在回收站存储路径中继续保留。该存储路径默认为 Bucket/.trash/，用户可以更改此存储路径，但需遵循路径名称规则。仅有该 Bucket 的拥有者及具有 Full Control 权限的用户可进行回收站配置。

> **说明：** 归档类型的文件，无论是否取回，删除后都不会进入回收站。



6. 当 Bucket 中的某文件被删除后，该文件在控制台 **文件列表** 中将不复存在，您可在该存储桶详情列 **回收站** 页签下，查看回收站内 文件名，并进行管理。

回收站内数据管理

对于回收站中的垃圾文件，您可以按需进行管理。您可通过存储桶的左侧导航最下方找到回收站，通过搜索菜单中输入【回收站】后点击进入 **恢复文件** 支持执行单个或者批量 **恢复** ，执行恢复操作成功时，该文件会出现在删除前的文件路径，并从回收站消失； **彻底删除文件** 您可彻底删除文件，当您对文件执行 **彻底删除** 操作时，该文件将被彻底清除，请谨慎操作。

相关 API

- PutBucketTrash 接口：使用 PutBucketTrash API 接口开通回收站功能。
- GetBucketTrash 接口：使用 GetBucketTrash API 接口获取回收站功能开通状态。
- DeleteBucketTrash 接口：使用 DeleteBucketTrash API 接口关闭回收站功能。

管理Bucket标签

概述

百度智能云提供标签管理功能，通过给每个云资源添加标签，从而快速分类和识别管理这些资源。

- 标签：每个标签由键和值两部分组成，标签（键+值）唯一。
- 单个和批量：支持为单个资源设置标签，也可以批量为云资源创建标签。

注意事项

每个用户最多可以创建 200 个标签。

添加标签

用户可以根据项目和场景需求，为 Bucket 添加标签，便于对 Bucket 进行分类和识别管理。

操作步骤

1. 登录 对象存储 BOS 管理控制台。
2. 存储桶标签配置：

方式一：在左侧 Bucket 列表中，对需设置标签的 Bucket ，可通过点击：**标签** 进行编辑。



方式二：进入需要操作的存储桶管理详情页顶部导航处，点击**标签**



> **说明：**
>
> - 存储桶标签可批量编辑，但只能批量添加标签，不能进行已有标签管理。
> - 注意标签键设定时区分大小写。

3. 在弹出的对话框中，输入自定义标签键和标签值，注意每个 Bucket 可以有多个标签键，但每个标签的键必须唯一，值可以为空，不填写。
4. 点击 **确定**，完成标签创建。

> **说明：**
>
> - 批量创建 Bucket 标签时，无法查看已添加过的单个实例标签。
> - 如果修改某个 Bucket 标签，需要先解绑标签确定后再重新设置。

解绑标签

如果 Bucket 不再需要标签，您可以将标签解绑。

操作步骤

1. 登录 对象存储 BOS 管理控制台。
2. 在左侧 Bucket 列表中，选择需要设置标签的 Bucket。
3. 进入该 Bucket 文件列表页后，点击页面右上方 **编辑标签**。
4. 弹出对话框中，点击 × 去掉标签，即可解绑该标签。



查看资源账单

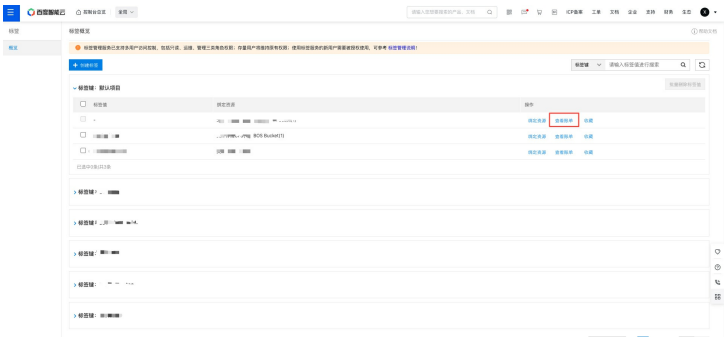
资源账单是记录每个资源的生命周期与消费相关的数据的功能，方便用户以资源为维度查看资源及产品的消费数据。

操作步骤

1. 登录 对象存储 BOS 管理控制台。
2. 在左侧 Bucket 列表中，选择需要设置标签的 Bucket。
3. 进入该 Bucket 文件列表页后，点击页面右上方 **编辑标签**。
4. 在弹框中选择 **前往标签管理>**，进入百度智能云标签管理页面。



5. 在已绑定 BOS Bucket 的操作栏中点击 **查看账单**。



6. 进入资源账单页面，选择需要查看账单的月份，会显示当前标签资源的消费信息。还可以通过组合筛选条件执行标签搜索，按标签键和值进行选择其他标签的消费详情，并下载相关信息。

7. 点击查看详情，可以查看资源ID、产品名称、账单金额。

设置存储空间清单

概述

存储空间清单是一种帮助用户管理存储桶中对象的功能。BOS 可根据用户的清单任务配置，定期每天或者每周扫描用户 Bucket 中指定对象并输出一份清单报告，以 CSV 格式文件的形式存储到用户指定的 Bucket 中。CSV 文件中会列出对象及其对应的元数据，并根据用户的配置信息，记录用户所需的对象属性信息。

> **说明：**
>
> - 存储清单功能于2022年12月8日起正式计费，将根据列出的对象（文件）个数计费，了解价格详情参见BOS价格说明。
> - 当前该功能在华北-北京、华东-苏州和华南-广州区域全量开放。
> - 存储清单导出效率会受存储桶内对象数量影响，针对存储大量文件的存储桶，一至两天的清单生成延迟属于正常情况。

使用场景

清单是一种帮助用户管理存储桶中对象的功能，可以有计划地取代对象存储的 List API 操作。您可以使用清单功能实现包括但不限于以下基本用途：

- 定期审核并输出文件列表及文件相关属性。
- 通过定期扫描简化并加快业务工作流，无需通过脚本实现，简单配置即可定期输出文件报告。

使用前须知

在配置清单前，您需要了解两个概念：

- **源存储桶**：想要开通清单功能的存储桶。
  - 包含清单中所列出的对象。
  - 包含清单的配置。
- **目标存储桶**：存储清单的存储桶。
  - 包含清单列表文件。
  - 包含 Manifest 文件，描述清单列表文件的位置。

设置存储空间清单

1. 登录 对象存储 BOS 管理控制台。
2. 单击**Bucket 列表** ，然后单击 Bucket名称。
3. 在左侧导航栏 选择 **高级配置** > **清单配置**。
4. 在 **清单配置**页签，单击 **新增配置**，在新增清单规则对话框，按以下说明配置各项参数。

配置名称	配置描述
清单状态	• 开启：清单规则创建后立即生效，BOS 根据规则配置定期扫描并输出报告 • 关闭：仅创建清单规则创建但不开启规则，后续可自行开启
清单名称	• 输入清单规则名称，只能包含大小写字母，数字和-_ .；必须以字母开头,长度1-64
清单存储位置	• 输出清单报告的存储位置，需为当前地域下的bucket
报告清单前缀	• 输入清单前缀名称。您可自定义一个名称，作为该策略下生成的清单报告前缀
包含文件范围	• 可设置对整个 Bucket 文件生效或者仅对前缀文件生效
清单导出周期	• 您可以设置每三天或者每周导出一次清单
清单导出格式	• 当前支持导出 CSV 格式清单报告

> **说明：**
>
> • 每个 Bucket 最多同时创建 20 条清单规则。当超过规则数量上限后，您需要删除已有规则后重新创建；



5.编辑或者删除清单规则

您可以对已创建的清单规则进行编辑，如将关闭的清单规则设置为开启状态。您也可以将已创建的清单规则删除。规则删除后，BOS 将不会按照该规则继续为您输入清单报告。

清单报告存储路径

- **清单报告存放位置**：清单报告会发布在目标存储桶中，其中清单报告会发布在以下路径：

```
destination-prefix/[source-bucket]/[config-ID]/[YYYY-MM-DDTHH-MMZ]/xxx.csv
```

- **Manifest 相关文件存放位置**：相关的 Manifest 文件会发布在目标存储桶以下路径：

```
destination-prefix/[source-bucket]/[config-ID]/[YYYY-MM-DDTHH-MMZ]/manifest.json
destination-prefix/[source-bucket]/[config-ID]/[YYYY-MM-DDTHH-MMZ]/manifest.checksum
```

路径字段代表的含义

- **desitination-prefix**：是您在配置清单时设置的“报告清单前缀”，可用于对目标存储桶中的公共位置的所有清单报告进行分组。
- **source-bucket**：是清单报告对应的源存储桶名称，加入这个文件夹是为了避免在多个源存储桶将各自清单报告发送至同一目标存储桶时可能造成的冲突。
- **config-ID**：是您在配置清单时设置的“清单名称”，当同一源存储桶设置多个清单报告并将其发送至同一目标存储桶时，可以用 config-ID 区分不同的清单报告。
- **YYYY-MM-DD-HH-MM**：时间戳，包含生成清单报告时开始存储桶扫描的时间和日期。
- **manifest.json**：指 Manifest 文件。
- **manifest.checksum**：指 manifest.json 文件内容的 MD5。

清单内容

清单的生成内容包括清单 CSV 文件列表和相关的 Manifest 文件。

清单CSV文件列表内容

输出的清单文件列表中包含如下内容：

字段说明	说明
Bucket	Bucket 名称
Key	文件名称
Size	文件数据长度
Etag	对象的哈希。可能是对象数据 base64 编码的 MD5 摘要，这取决于对象的创建方式
StorageClass	文件的存储类型
CreateTime	文件的创建时间。时间格式为：yyyy-mm-ddThh:mm:ssZ 。如：2016-09-07T00:00:00Z（UTC）

Manifest文件内容

Manifest 文件包含如下内容：

字段说明	说明
sourceBucket	清单报告对应的源存储桶名称
destinationBucket	放置导出文件的目标存储桶名称
version	导出的版本。格式为"year-month-day", 如"2021-11-03"
fileFormat	报告的格式
listObjectCount	列出的对象数量，费用按此项计费
listStorageSize	列出的对象总大小，单位是字节。
fileSchema	报告的架构
files	目标存储桶中的导出报告文件的实际列表。包含 key（清单文件的名称）、size（清单文件的大小）、md5Checksum（该文件的 md5 值）

Manifest 报告示例如下：

```
{
  "sourceBucket": "src-bucket-name",
  "destinationBucket":"destination-bucket-name",
  "version":"2020-11-03",
  "fileFormat":"CSV",
  "fileSchema":"Bucket,Key,Size,etag,StorageClass,PutDate,CreateTime",
  "files":[
      {
          "key":"destination-prefix/[source-bucket]-[config-ID]-[YYYY-MM-DDTHH-MMZ]-index.csv",
          "size":1603959,
          "md5Checksum":"2177e775870def72b8d84febe1ad3574"
      }
    ]
}
```

API 参考

- PutBucketInventory：用于为源bucket中创建清单任务，您可以对清单任务命名后，使用该请求创建清单任务；
- GetBucketInventory：使用 GET 操作来获取指定桶的某个清单配置。

设置合规保留

概述

Bucket 合规保留是数据保护的一种方式，用于规避重要的数据存入 Bucket 后被意外删除的风险，适用于金融、保险、医疗、证券等行业。合规保留是一种基于时间的配置策略，用户开启 Bucket 合规保留后需要设定数据保留的时间期限。

注意事项

Bucket 合规保留支持的保留时间为 1 天- 70 年。合规保留一旦开启将无法关闭，也无法缩减保留时间，且 Bucket 中 Object 在保留期内无法删除，请您在开启前仔细确认。

保留期说明

合规保留期是从 Bucket 内每一个 Object 的上传时间进行计算，而不是从配置合规保留策略的时刻开始算。

假设在用户在 2016 年 1 月 1 日创建一个名为 BCEName 的 Bucket，并且在 Bucket 中的不同时间上传了 file1.pdf、file2.pdf、file3.pdf 三个 Object。随后，在 2019 年 1 月 1 日创建了保护周期为 5 年的 Bucket 合规保留策略，有关这三个Object的具体上传时间以及对应的 Object 到期时间如下：

上传文件名称	创建时间	合规保留到期时间
file1.pdf	2017 年 1 月 1 日	2021 年 12 月 31 日
file2.pdf	2018 年 1 月 1 日	2022 年 12 月 31 日
file3.pdf	2018 年 6 月 1 日	2023 年 5 月 31 日

设置合规保留

1. 登录 对象存储 BOS 管理控制台。
2. 在左侧 Bucket 列表中，单击 Bucket 名称。
3. 在左侧导航栏选择 **高级配置** > **合规保留配置** 。
4. 在 **合规保留** 区域点击 **开启配置** ，在 **开启合规保留配置** 操作栏中设定合规保留规则。保留时间设定后，需要锁定后策略才能生效。策略支持 **立即锁定** 和 **稍后锁定** 两种形式。若选择 **稍后锁定** ，该策略会保留



24 小时。若24小时内仍未锁定，该策略将过期，需要将策略删除后重新配置。

策略锁定后，该Bucket内的文件将进入合规保留期。在保留期内，还可以点击 **修改配置**来延长合规保留期限。

> **说明：** 若为一个空 Bucket 设置了合规保留期限，依旧可以删除该 Bucket。删除后该合规保留策略也将同步删除。

5. 点击 **确认** ，完成配置。配置完成并锁定后，合规保留策略立即生效。

相关 API

- InitBucketObjectLock 接口：使用 InitBucketObjectLock API 接口为指定 Bucket 创建基于时间的合规保留策略。
- GetBucketObjectLock 接口：使用 GetBucketObjectLock API 接口获取合规保留策略配置信息。
- DeleteBucketObjectLock 接口：使用 DeleteBucketObjectLock API 接口删除合规保留策略。
- CompleteBucketObjectLock 接口：使用 CompleteBucketObjectLock API 接口将合规保留策略立即锁定。
- ExtendBucketObjectLock 接口：使用 ExtendBucketObjectLock API 接口延长合规保留策略保护周期。

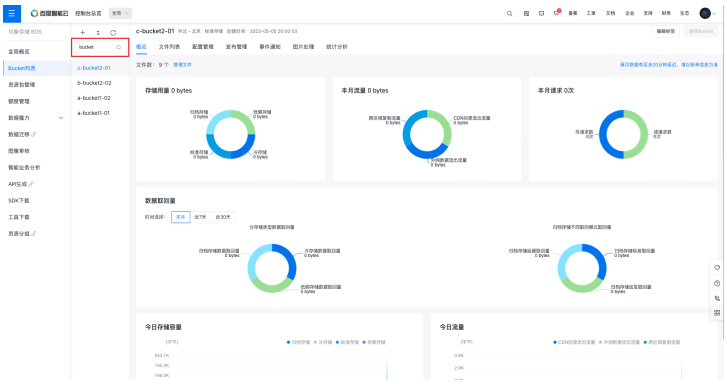
查询Bucket

概述

BOS 支持创建最多 100 个 Bucket 来存放和区分不同数据。您可以对象存储 BOS 控制台查看已创建的全部 Bucket。

控制台查看全部Bucket

1. 登录 对象存储 BOS 管理控制台。
2. 您已创建的 Bucket 会以列表形式展示在 BOS 控制台左侧的导航栏中。您可以对已创建的 Bucket 根据名称进行逆序排列，并可以刷新 Bucket 列表，也可以创建新的 Bucket 。
3. 列表上方有 Bucket 搜索框。搜索框支持模糊搜索，显示所有名称包含当前输入内容的Bucket。
4. 当鼠标指针落在 Bucket 名称上时，会显示该 Bucket 的名称、所属地域和创建时间。



> **说明：**
>
> - 若您使用主账号登录控制台，BOS 会为您列举全部 Bucket。若您使用子账号登录控制台，BOS 只为您列举您有权限查看的 Bucket，其他 Bucket 不会列举。

相关 API

- ListBucket 接口：使用 ListBucket API 接口列举当前全部 Bucket。

请求者付费

概述

请求者付费模式是指由请求者支付读取存储空间（Bucket）内数据时产生的流量和请求费用，而 Bucket 拥有者仅需支付存储费用。当您希望共享数据，但又不希望产生流量费用和请求费用时，您可以开启此功能。

使用场景

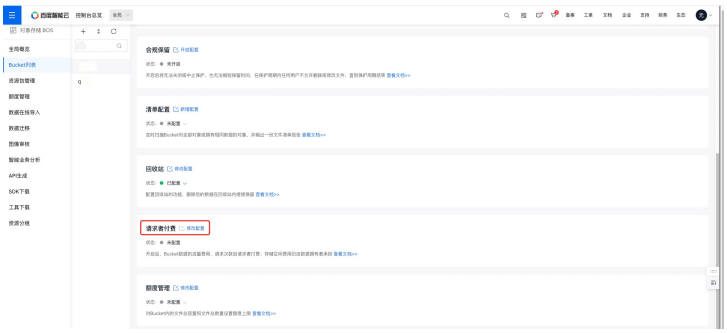
- 大型数据集共享（如计算机视觉图像训练数据集、机器学习训练数据集、遥感探测地理数据等）。如某学术机构需要将深度学习的训练数据集公开，以支持其他学术研究人员访问和下载。此时，该学术机构可以为存储该数据集的 Bucket 开启请求者付费功能。开启后，该研究机构只需要支付数据集的存储费用，而第三方人员承担下载和访问该数据集产生的流量和请求费用；
- 大型数据交付场景。如某公司需要将存储的数据授权并交付到公司的合作伙伴或者客户，此时可以开启相应 Bucket 的请求者付费功能。开启后，数据的下载和访问费用将由合作伙伴或者客户承担。

注意事项

- 开启请求者付费后不允许匿名访问，访问者均需携带身份验证信息；
- 当请求者是通过百度智能云 IAM 角色来进行访问时，请求费用和流量费用将由该角色所属的账户（主用户账户）承担；
- 已开启请求者计费的 Bucket 不支持将权限设置为"公共读"和"公共读写"；
- Bucket 访问权限为"公共读"和"公共读写"时，不支持开启请求者计费。

设置请求者付费

1. 登录 对象存储 BOS 管理控制台。
2. 在左侧 Bucket 列表中，选择需要开启请求者付费的 Bucket，点击 Bucket 名称进入 Bucket 管理目录。
3. 在上方导航栏选择 **配置管理** 页签。
4. 在 **配置管理** 页面中选择 **高级配置**，在 **请求者付费** 区域点击 **修改配置** 对该 Bucket 的请求者付费进行配置。默认情况下请求者付费功能不开启。
5. 在 **请求者付费**操作栏中，将该功能调整为开启状态。



> **注意：**
>
> - 该功能开启后可随时关闭。

6. 设置完成后，点击 **确认** 完成配置。

相关 API

- PutBucketRequestPayment 接口：使用 PutBucketRequestPayment API 接口创建请求者付费功能。
- GetBucketRequestPayment 接口：使用 GetBucketRequestPayment API 接口获取 Bucket 请求者付费规则。

设置Bucket读写权限

权限介绍

权限定义

BOS 权限管理包括如下三类权限，您可以根据自身业务场景进行合适的权限设置。

Bucket标准权限定义

权限名称	权限描述
私有	• Bucket 创建者独享该 Bucket 的全部权限，其他人不能对该Bucket 写入数据，也不可访问读取该 Bucket。 • 适用于存储私密文件的场景。
公共读	• Bucket 创建者独享该 Bucket 的全部权限，其他人可以访问读取该 Bucket 内的内容，但无权写入。 • 适用于存储需要公众读取的文件的场景，如通知性文件、广告内容等。 • 请注意：若开启该种权限，互联网上的所有用户都可以对该 Bucket 内的文件进行访问读取，请谨慎开启。
公共读写	• 包括 Bucket 创建者在内的任何人都可以访问和写入文件。 • 请注意，若开启该种权限，互联网上的所有用户都可以对该 Bucket 内的文件进行读写操作，请谨慎开启。

粗粒度自定义权限定义

如果您认为上面的 Bucket 标准权限不能满足使用需求，您可以选择按需生成粗粒度自定义权限。该权限支持对指定用户设置 READ、LIST、WRITE、MODIFY 和 FULL_CONTROL 权限，并可指定该权限可访问的资源，以及指定具有该权限的 IP 地址和 Referer 白名单等。

BOS 支持的各粗粒度自定义权限说明如下：

权限名称	权限支持的操作
READ	允许读取 Bucket 内的 Object 及其相关信息，但没有列表权限，具体操作权限包含 GetBucketLocation、HeadBucket、GetObject、GetObjectMeta、ListParts、RestoreObject。READ 权限对应的 API 既有 Bucket 级别的 API 如 GetBucketLocation，也有 Object 级别的 API 如 GetObject 和 ListParts。
LIST	列表权限，可以查看指定 Bucket 下的 Object 列表以及获取所有未执行完的 MultipartUpload，具体操作权限包含 ListObjects 和 ListMultipartUploads。LIST 权限对应的 API 只有 Bucket 级别的 API。
WRITE	允许创建，覆盖和删除 Bucket 内的 Object，具体操作权限包含 PutObject、PostObject、InitiateMultipartUpload、UploadPart、CompleteMultipartUpload、AbortMultipartUpload、AppendObject、DeleteObject、DeleteMultipleObjects和FetchObject。WRITE 权限对应的 API 只有 Object 级别。
MODIFY	用户仅可做 PutObject、AppendObject 等相关写入操作，不可做数据新增、删除操作。该权限主要功能是与 Deny 合用防止 Bucket 数据被篡改
FULL_CONTROL	包含以上所有权限。FULL_CONTROL 除了具有 READ、LIST 和 WRITE 的所有操作权限以外，还包括以下操作权限 PutBucketAcl、GetBucketACL、PutBucketCors、GetBucketCors 和 DeleteBucketCors。FULL_CONTROL 权限对应的 API 既有 Bucket 级别的 API 也有 Object 级别的 API。

> **注意：**
>
> • MODIFY **仅包含 PutObject、AppendObject 等对已存在文件的写入操作，不包含文件新增与删除操作**，若您需要禁止子用户或其他用户对已存在BOS文件的修改和删除操作，请配置显式 Deny ，Modify 与 DeleteObject 操作。

细粒度自定义权限定义

如果您认为上述粗粒度自定义权限还不能满足您精细授权的需求，您可以使用 BOS 提供的细粒度自定义权限。BOS 支持的各细粒度自定义权限包括 Bucket 级别和 Object 级别。

Bucket相关权限

Bucket 相关权限	支持的操作
GetBucket	该权限表示允许用户获取 Bucket 内容及其相关信息，例如，列出 Bucket 中 Objects、在三步上传时列出 Bucket中 的所有未执行完成的 Multipart Upload
GetBucketAcl	该权限表示允许用户获取 Bucket Acl 的信息
PutBucketAcl	该权限表示允许用户新增 Bucket Acl
GetBucketCors	该权限表示允许用户获取 Bucket 上的跨域资源共享（CORS）的规则
PutBucketCors	该权限表示允许用户在指定的 Bucket 上设定或者删除一个跨域资源共享（CORS）的规则
GetBucketStyle	该权限表示允许用户获取或者列出 Bucket Style 的规则
PutBucketStyle	该权限表示允许用户新增或者删除 Bucket Style 的规则
GetBucketMirroring	该权限表示允许用户获取 Bucket 镜像回源的相关信息
PutBucketMirroring	该权限表示允许用户新增或者删除 Bucket 镜像回源的相关信息
GetCopyRightProtection	该权限表示允许用户获取 Bucket 的原图保护配置的信息
PutCopyRightProtection	该权限表示允许用户开启或者关闭 Bucket 的原图保护功能

Object 相关权限

Object 相关权限	支持的操作
PutObject	该权限表示允许用户进行上传 Object 的相关操作，例如 PutObject、PostObject、AppendObject、FetchObject、CopyObject、三步上传、三步 Copy
GetObject	仅支持 GetObject 和 GetObjectMeta 操作。GetObject 权限对应的 API 只有 Object 级别。
DeleteObject	该权限表示允许用户进行删除单个 Object 或者批量删除 Object 的相关操作
RenameObject	该权限表示允许用户对 Object 进行重命名（Rename）
ListParts	该权限表示允许用户列出三步上传过程中指定的 UploadId 所有已经上传成功的 Part，用户可以查看三步上传的当前进度
GetObjectMeta	该权限表示允许用户列出 Object 的 Meta 信息
GetObjectAcl	该权限表示允许用户获取 Object Acl
PutObjectAcl	该权限表示允许用户新增 Object Acl 和删除 Object Acl

> **说明：**
>
> - 细粒度和粗粒度权限 READ、LIST、WRITE、FULL_CONTROL、MODIFY 间互不影响；
> - 粗粒度权限的优先级高于细粒度权限。如果既配了粗粒度权限又配了细粒度权限，粗粒度权限会覆盖细粒度权限，以粗粒度为主；
> - Bucket 级别的细粒度是指对 Bucket 进行的相关操作；
> - Object 级别的细粒度是指对 Object 进行的相关操作。

设置Bucket读写权限

概述

为了保障您存储在 BOS 中数据的高安全性，我们为您提供了丰富的多级权限管理能力。BOS的权限体系分为如下三级：

- **Bucket 标准权限**：业界常见的权限设置，分为私有、公共读和公共读写。
- **粗粒度自定义权限**：较标准权限更细粒度的权限，可对指定用户设置 READ、LIST、WRITE、FULL_CONTROL 和 MODIFY 权限，并可以指定该权限可访问的资源，以及指定具有该权限的 IP 地址和 Referer 白名单等。
- **细粒度自定义权限**：BOS 提供的 API 级别细粒度自定义权限。可设置的 API 类别包括 GetBucket、GetObject、PutObject、DeleteObject 在内的共 18 种。您可以对每种类别自定义相应的访问权限。

设置Bucket读写权限

1. 登录 对象存储 BOS 管理控制台。
2. 在左侧 Bucket 列表中，选择需要设置权限的 Bucket，点击 Bucket 名称。
3. 在左侧导航栏选择 **配置管理** > **权限配置** 。
4. 在 **权限配置** 页面 ，单击 **修改配置** 对该 Bucket 的权限进行配置。
5. 可将 Bucket 设置为私有、公共读和公共读写，更多需求可选择自定义权限，点击 **自定义** ，可进行可视化 或 编辑整个ACL 策略文本的方式进行配置。

自定义权限的配置参数参考下表：

配置名称	配置描述

用户授权	• 所有用户：指该配置对所有用户生效 • 自定义：指该配置对指定用户生效。当选择 "自定义" 之后，您需要输入希望指定的用户 ID，用户 ID 可以在用户中心 查看。同时，BOS 支持同时设置多个用户 ID，每个用户 ID 分行填写即可（每行末尾不需要标点符号分隔）。若希望对全部用户生效，可以填写"*"（最多支持填写 1 个"*"）。您也可以填入AuthenticatedUsers，代表授权给所有百度智能云注册用户 • 上述所有用户ID均指主账户ID，如需要对子用户的用户ID进行权限管控，需要通过IAM多用户访问控制进行相应配置： IAM多用户访问控制
授权效果	• 用于设置授权用户的操作效果。若选择 "允许"，那么将配置的权限以 "允许" 的效果授予用户；若选择 "拒绝"，那么将配置的权限以 "拒绝" 的效果授予用户
授权配置	• 粗粒度自定义权限：包括READ、LIST、WRITE、MODIFY、FULL_CONTROL权限 • 细粒度自定义权限：用户可通过点击"高级设置"，展开细粒度权限列表，可按需勾选一个或多个，组合成新的自定义权限 • 细粒度和粗粒度权限 READ、LIST、WRITE、FULL_CONTROL、MODIFY 间互不影响，可同时授权 • 粗粒度权限的优先级高于细粒度权限。如果既配了粗粒度权限又配了细粒度权限，粗粒度权限会覆盖细粒度权限，以粗粒度为主。您可以根据自身需求对粗粒度权限和细粒度权限进行组合
资源	• 指定该权限作用的资源范围。 • 其中 “包含” 的应资源即生效的资源范围。资源必须以 Bucket 名称开始；资源如果只有 1 个斜杠，不能以斜杠结尾，应以通配符 "*" 结尾；资源可以设置多个，每行 1 个且每行以通配符结尾，示例：`myBucket, myBucket/*，myBucket/myfolder/object*`。若资源留空，则等同于 "Bucket名称"。 • 其中 “不包含” 的资源表示对指定范围之外的 Object 设置权限，配置填写方式同 "包含" 相同。如果选择了 “不包含” 但填写设置为空，相当于未配置。此时采用默认配置，即 Bucket 自身和 Bucket 内全部 Object。

访问控制	以下为支持的条件，即满足您配置的条件时，该条访问控制策略生效
	• Referer：设定 Referer 白名单。每个 Referer 以换行符分隔，每个Referer 最多支持一个通配符 `*`。同时可以勾选是否允许Referer 为空。选择 “允许 Referer 为空” 时，HTTP 请求中带白名单中的 Referer 和空 Referer 都允许访问；选择 “不允许 Referer 为空” 时，HTTP 请求只有带白名单的 Referer 可以访问，空 Referer 不允许。
	• IP地址：指定 IP 地址列表，使用 CIDR 方式对 IP 进行标识。IP 地址可以设置多个，每行 1 个，且每行最多 1 个通配符`*`，并以`.*`结尾。示例：`192.168.1.*`或
	`192.168.0.1/24`
	`192.168.0.100`
	`192.168.*`
	支持配置IP 等于和IP 不等于两种条件配置，IP 不等于通常用于配置拒绝非某些来源IP的访问
	• https 协议：白名单仅支持 http 和 https 协议。若您需要使用 https 协议，您需要勾选该选项
	• 访问时间：BOS 支持对该自定义权限设置访问时间，您可以在访问时间中设置最小时间和最大时间
	• VPC：BOS 支持对该自定义权限设置 VPC 级别访问控制，您可以设置允许或禁止某个 VPC 访问您的 Bucket，您可以通过控制台选择本账号VPCID，也可通过【输入其他VPCID】设置其他账号下VPC
	• 签名版本：您可指定BOS兼容S3的签名版本V2版本，设置该签名版本允许或禁止访问资源

	• UserAgent：您可指定UserAgent,设置等于 或 匹配 某些 UserAgent的请求情况下允许或禁止访问资源

说明：

- 若希望根据 IP 地址进行访问控制，您需要使用 Bucket 官方域名或者未开启 CDN 加速的自定义域名进行访问。若您使用 CDN 官方域名，或使用开启 CDN 加速的自定义域名访问 BOS，此时 IP 设置将无效。
- **VPC 级别访问控制目前仅支持华北-北京、华北-保定和华东-苏州区域**，若您需要其他地域支持，请通过您的商务经理或提交工单联系我们。

数据安全提示：

- **公共读写权限**所有人无需身份验证可以直接读写您存储桶中的数据，安全风险极高，建议严格遵循最小权限原则，避免数据安全风险。
- **LIST权限**支持查看存储桶中的Object列表以及获取所有未执行完的分片上传任务，不建议授予所有用户LIST权限，请严格遵循最小权限原则。



6. 单击 **确认** 完成配置。
7. 配置完成后，您可以在 **权限配置** 中看到已生成的权限记录，并可通过 **编辑** 和 **删除** 按钮对已有权限进行调整。

Bucket额度管理

概述

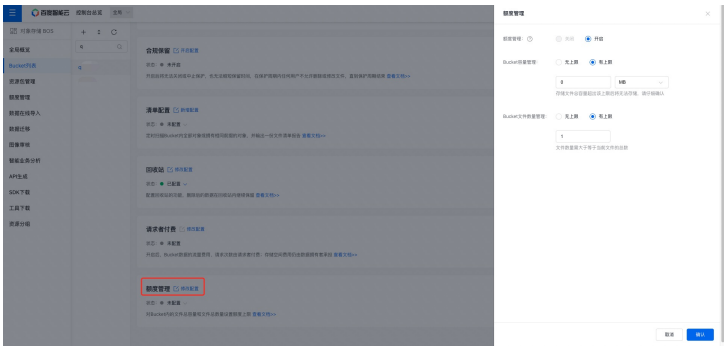
针对企业对自身财务和资源的自定义控制场景，BOS 提供 Bucket 级别额度管理功能。该功能支持对单个 Bucket 的文件数量上限和容量上限进行配置，实现对存储服务弹性能力的自定义控制。

使用限制

- 单个 Bucket 内存储文件数量默认无上限；
- 单个 Bucket 内存储容量默认无上限；
- 若您希望将 Bucket 总容量上限修改为某一数值，该数值需大于等于您当前 Bucket 内的实际存储总量。当 Bucket 内总容量达到您设置的额度时，将不能上传新文件，同时不能对已有文件进行复制操作；
- 若您希望将 Bucket 总文件数量上限修改为某一数值，该数值需大于等于您当前 Bucket 内的实际存储文件总数。当 Bucket 内总文件数量达到您设置的额度时，将不能上传新文件，同时不能对已有文件进行复制操作；

操作指南

1. 登录 对象存储 BOS 管理控制台。
2. 在左侧 Bucket 列表中，选择需要设置权限的 Bucket，点击 Bucket 名称进入 Bucket 管理目录。
3. 在上方导航栏选择 **配置管理** 页签。
4. 在 **配置管理** 页面中选择 **高级配置**，在 **额度管理** 区域点击 **修改配置** 对该 Bucket 的额度进行配置。



相关API

- PutBucketQuota 接口：使用 PutBucketQuota API 接口设置 Bucket 额度。
- GetBucketQuota 接口：使用 GetBucketQuota API 接口获取 Bucket 额度配置信息。
- DeleteBucketQuota 接口：使用 DeleteBucketQuota API 接口删除 Bucket 额度设置。

管理生命周期

概述

存放在 BOS 中的文件通常会发生归档下沉、删除等涉及到文件生命周期的操作。一般情况下，文件在新建后的短期内会被频繁读取访问，随着时间的推移，该文件的读取次数将变少，进而变成"冷文件"，即不再被频繁的访问。到最后，该文件将会被最终删除。用户如果手工维护数据的生命周期，则费时费力；但如果不去维护，则数据始终存放在标准存储里会产生不菲的费用。因此，BOS 提供生命周期管理功能，以帮助用户自动化地完成数据的生命周期管理，实现数据从创建到归档到删除的自动管理流程，从而节约人力和存储费用。

文件生命周期管理方式

- 为了使数据管理更加便捷与智能，我们在保持【基础生命周期管理】（原生命周期管理）的基础上，又支持了【智能分层】，智能分层不仅可根据文件访问频次进行自动沉降，也可根据文件的访问情况进行智能冷热转化，自动升级至标准或低频存储，该功能提供了数据存储的成本与读写性能最优解决方案。
- 在基础生命周期中在基础基于前缀进行细化管理的能力上，又扩展支持设置对象标签功能(Tag)、目录排除(Not)、文件大小过滤(Filesize)、对象多版本的生命周期管理功能，可细化生命周期操作粒度。

智能分层与基础生命周期管理区别

方式	规则配置	适合场景	转化模式	转化存储类型
智能分层	一键开启	访问模式不固定或者无法预估访问模式的数据	支持数据降冷，也可按需回暖	不支持归档与多az存储类型，其余均支持
基础生命周期管理	需自定义管理规则	1.可预估访问模式 2.需要精细管理	单向降冷	全部支持（多版本暂不支持归档）

功能说明

> 存储桶仅支持配置【生命周期管理】或【智能分层】，二者不可同时设置。

智能分层

- 存储桶开启智能分层，系统将周期性地监测数据访问次数，在持续一段时间没有数据访问时，将数据转移至存储成本更低的访问层。如果数据重新被访问，则会被重新转移到高频访问层上，保障数据读取性能。
- 支持文件从标准存储-〉低频存储-〉冷存储 或冷存储-〉低频存储-〉标准存储；
- 可配置删除文件与碎片（分片上传任务中，未完成的文件碎片）定期清除，支持按照指定固定天数后执行删除操作， 规则中计算起始时间以 Object 的创建时间为准，删除优先级高于转化，到期会执行删除；
- 配置规则后您可通过控制台存储桶数据分析查看单个存储桶沉降与回暖数据量。 **智能分层注意事项**
- 不支持转化为归档存储，标准存储-多AZ，低频存储-多AZ；
- 存储桶仅可配置一条规则。

基础生命周期

基础生命周期管理支持如下功能

- 自定义时间换存储类型，从标准存储转低频存储、转冷存储、转归档存储；或从标准存储-多 AZ 到低频存储-多 AZ；
- 定时删除不再需要的数据；
- 清除过期的三步上传数据。 ****从场景上划分，基础生命周期管理支持两种模式****
- 数据达到一定寿命后自动归档：如定义所有创建时间超过30天的数据自动转为存储费用更为低廉的低频存储；
- 数据达到一定寿命后自动删除：如定义所有创建时间超过30天的数据自动删除。

****从规则配置上划分，基础生命周期管理支持简单与复杂模式****

- 简单模式：定义规则中仅还有指定生效文件范围设置仅配置前缀或存储桶全部文件（不包含多版本对象）；
- 复杂模式：定义规则中包含指定生效文件范围设置包含对象标签、排除某些文件、指定文件大小、对象历史版本中的任意一种配置。 ****复杂生命周期规则管理支持功能****

- 支持设置对象标签，支持指定标签沉降删除Object。
- 支持设置目录排除，排除目录外的Object以及指定对象标签。
- 支持设置文件大小，支持对指定文件大小范围内的object进行沉降删除。
- 设置多版本生命周期， 支持对历史版本进行删除。

****特别提示****

存储桶生命周期管理规则复杂和简单模式，表现在对文件的转化与删除的执行规则的区别。

- 简单模式：对冲突的多条简单规则会按照"精细化执行"的逻辑，例如：规则 1：当设置 a/b/ 目录下文件创建满 180 天删除；规则 2： a/b/c 目录下文件创建满365 天删除,执行结果为a/b/c/1.txt 文件会在创建满365 后执行删除，而a/b/2.txt 则会在创建满180 后执行删除。
- 复杂模式：对冲突的多条复杂规则会按照"全局最优执行"的逻辑，且删除优先。例如规则 1：当设置 a/b/ 目录下文件（未配置历史版本）创建满 180 天删除；规则 2： a/b/ 目录下文件配置 365 天删除，并且排除 a/b/test目录,执行结果为a/b/c/1.txt 文件会在创建满180 天后执行删除，a/b/test/2.txt 也会在 180 天后删除。
- **存储桶中若创建一条复杂规则，则存储桶生命周期管理整体应用复杂规则执行**

基础生命周期注意事项

通用注意事项

- 每个 Bucket 可以有至多 1000 条规则;
- BOS 生命周期规则设置后会在一天内生效;
- 规则生效后，BOS 会对符合条件的 Object 进行相应的处理，但处理需要一定的时间，**不一定能马上看到效果**。一般情况下，沉降或删除的时间为几小时，但若沉降数据量较大，则可能会在几天甚至更长的时间完成下沉或者删除；
- 规则中计算的时间（即 Object 的"年龄"）以 Object 的创建时间为准，而不是生命周期规则的创建／修改时间;
- **BOS 只保存文件的最后修改时间，即 last-modified 时间**；如果您不更新 meta 或者覆盖文件，那么 last-modified 就是创建时间。所以生命周期中的"创建时间"其实是 last-modified 时间。
- 基于文件访问时间记录的生效策略，目前仅北京和苏州区域支持。
- 低频存储、冷存储和归档存储的最低存储时间分别为 30 天，60 天和 180 天。您配置的生命周期沉降/删除规则需要满足最低存储时间的要求。若您配置的时间小于最低存储时间时间，控制台将会产生提示，请您根据提示中的要求进行配置。
- **单 AZ 类型文件仅能沉降到单 AZ 类型文件**，无法沉降到多 AZ 类型文件；**标准存储-多AZ 类型文件只能沉降到低频存储-多 AZ 类型文件**。
- 发生生命周期沉降时，原类型若需要取回，则会产生取回费用。同时，沉降完成后，原类型文件会被删除，也会产生请求费用。比如，一个低频存储类型文件沉降到冷存储，那么 BOS 需要先获取原低频存储文件，该操作产生取回费用；当文件成功沉降为冷存储文件后，原低频存储文件会被删除，该操作产生 Delete 请求费用，该请求费用包含在写请求费用中。
- 生命周期暂不支持对通过追加上传所生成的 Appendable Object 进行层级转换。

复杂生命周期规则注意事项

- tag的规则设置: 每个规则最多有10个对象标签，每个key长度小于等于128， value长度小于等于256。
- 每个rule下只有1个排除目录, 开启Not选项时，前缀和标签必须至少存在一项，即同时设置前缀和标签或者只设置前缀或标签，Not中的前缀或标签只有一个；排除前缀必须位于设置前缀之下，并和前缀的结构形式一样，例如前缀是 bucket/prefix，*not prefix可以是bucket/prefix1*
- 指定最小文件指定生命周期规则作用的最小文件大小，最小文件大小要求大于等于0字节，小于5TB。
- 指定最大文件指定生命周期规则作用的最大文件大小，最大文件大小要求大于0字节，小于等于5TB。

基础生命周期简单与复杂生命周期规则在执行层面的区别：

- 简单生命周期语义: 按照的是最长前缀的原则， 如果命中最长前缀，那么就不会去寻找其他规则。
- 复杂生命周期语义会寻找全局最优规则， 寻找符合prefix、 not、 filesize、 tag的规则下最早执行的规则。
- 了解更多功能细节，请参考开发者指南-基础生命周期管理。

配置方式 1.控制台操作

- 基础生命周期，参考配置基础生命周期管理;
- 智能分层，参考配置基础智能分层。

2.相关API操作

- PutBucketLifecycle 接口：使用 PutBucketLifecycle API 接口创建生命周期管理规则。
- GetBucketLifecycle 接口：使用 GetBucketLifecycle API 接口获取定义的生命周期管理规则详细信息。
- DeleteBucketLifecycle 接口：使用 DeleteBucketLifecycle API 接口删除定义的生命周期管理规则。

配置智能分层

智能分层

- 存储桶开启智能分层，系统将周期性地监测数据访问次数，在持续一段时间没有数据访问时，将数据转移至存储成本更低的访问层。如果数据重新被访问，则会被重新转移到高频访问层上，保障数据读取性能。
- 支持文件从高频访问层转化为低频与冷访问层即： **标准存储 - > 低频存储 -> 冷存储**，也可根据数据的访问被重新访问回暖到高频访问层即： **冷存储 -> 低频存储 -> 标准存储**。
- 在文件智能转化存储类型的基本上您也可配置删除文件与碎片（分片上传任务中，未完成的文件碎片）定期清除，支持按照指定固定天数后执行删除操作， 规则中计算起始时间以 Object 的创建时间为准，且删除优先级高于转化，到期会执行删除。
- 配置规则后您可通过控制台存储桶 **统计分析** > **智能分层** 查看单个存储桶沉降与回暖数据量。

智能分层注意事项

- 配置智能分层后，系统会持续对存储桶进行监测分析，**需至少等待30天** 才会根据访问情况将数据转移至存储成本更低的访问层；如果数据重新被多次访问，则会被转移到高频访问层上。
- 不支持转化为归档存储，标准存储-多AZ，低频存储-多AZ。
- 存储桶仅可配置一条智能分层规则。
- 目前仅支持北京、苏州和广州区域。

智能分层计费说明

1. 智能分层功能通过监控分析存储桶中文件的访问情况进行存储类型变化，会额外收取对象监控费用，公测期间该计费项免费，欢迎使用，公测结束正式计费时会提前 7 个工作日通过短信 / 邮件通知您，确保您有充足时间规划使用。
2. 文件按照其对应的存储类型、所处地域，收取对应的存储、流量、请求、功能使用费用等，费用详情参见BOS价格清单。

控制台操作说明

配置智能分层

您可直接在左侧导航列中的【智能分层】进行配置管理，也可在Bucket管理目录中对当前Bucket配置智能分层。

方式一：在智能分层页面设置

1. 登录 对象存储 BOS 管理控制台。
2. 在左侧导航列，选择【智能分层】进入智能分层管理页面，默认将展示当前账户下所有已配置的智能分层规则。
3. 选择【智能分层配置】，可在弹窗中配置一条新的智能分层规则。
4. 您可在智能分层管理页面点击操作列的**编辑**和**删除**按钮，对已配置的智能分层规则进行编辑和删除。
5. 您也可以点击操作列的**统计**按钮，查看当前智能分层规则在不同存储类型下沉降与回暖的数据量。

方式二：在Bucket配置管理中设置

1. 登录 对象存储 BOS 管理控制台。
2. 在左侧 **Bucket 列表** 中，点击 Bucket 名称。
3. 在左侧导航栏选择 **基础配置** > **生命周期配置**。
4. 您可在 **生命周期配置** 页面中点击 **新增配置** ，选择 **智能分层配置**，对该 Bucket 的生命周期进行配置。默认情况下该功能不开启。

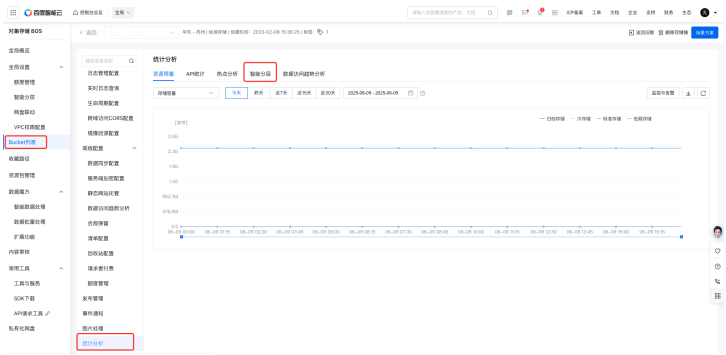


5. 打开智能分层开关，**智能分层规则仅对整个 Bucket全部文件生效**。除此之外，您可根据自身业务特点，可选配置自定义删除规则。 自定义删除支持删除文件与未完成的分片（Part），您可配置文件保留多少天后自动删除。
6. 点击 **确认** ，保存规则。
7. 规则添加成功后，您可以在 **智能分层** 中看到已生成的规则记录。您可以根据自身需求对规则进行重新编辑和删除。



查看智能分层执行

- 配置规则后您可通过控制台对开启智能分层的存储桶，通过 **统计分析** >**智能分层** 页签下查看指定时间范围内存储桶沉降与回暖数据量。



配置基础生命周期管理

概述

存放在 BOS 中的文件通常会发生归档下沉、删除等涉及到文件生命周期的操作。一般情况下，文件在新建后的短期内会被频繁读取访问，随着时间的推移，该文件的读取次数将变少，进而变成"冷文件"，即不再被频繁的访问。到最后，该文件将会被最终删除。用户如果手工维护数据的生命周期，则费时费力；但如果不去维护，则数据始终存放在标准存储里会产生不菲的费用。因此，BOS 提供生命周期管理功能，以帮助用户自动化地完成数据的生命周期管理，实现数据从创建到归档到删除的自动管理流程，从而节约人力和存储费用。

基础生命周期

基础生命周期管理支持如下功能

- 自定义时间换存储类型，从标准存储转低频存储、转冷存储、转归档存储；或从标准存储-多 AZ 到低频存储-多 AZ；

- 定时删除不再需要的数据；
- 清除过期的三步上传数据。 **从场景上划分，基础生命周期管理支持两种模式**
- 数据达到一定寿命后自动归档：如定义所有创建时间超过30天的数据自动转为存储费用更为低廉的低频存储；
- 数据达到一定寿命后自动删除：如定义所有创建时间超过30天的数据自动删除。

从规则配置上划分，基础生命周期管理支持简单与复杂模式

- 简单模式：定义规则中仅还有指定生效文件范围设置仅配置前缀或存储桶全部文件（不包含多版本对象）；
- 复杂模式：定义规则中包含指定生效文件范围设置包含对象标签、排除某些文件、指定文件大小、对象历史版本中的任意一种配置。 **复杂生命周期规则管理支持功能**
- 支持设置对象标签，支持指定标签沉降删除Object。
- 支持设置目录排除，排除目录外的Object以及指定对象标签。
- 支持设置文件大小，支持对指定文件大小范围内的object进行沉降删除。
- 设置多版本生命周期， 支持对历史版本进行删除。

特别提示

存储桶生命周期管理规则复杂和简单模式，表现在对文件的转化与删除的执行规则的区别。

- 简单模式：对冲突的多条简单规则会按照"精细化执行"的逻辑，例如：规则 1：当设置 a/b/ 目录下文件创建满 180 天删除；规则 2： a/b/c 目录下文件创建满365 天删除,执行结果为a/b/c/1.txt 文件会在创建满 365 后执行删除，而a/b/2.txt 则会在创建满180 后执行删除。
- 复杂模式：对冲突的多条复杂规则会按照"全局最优执行"的逻辑，且删除优先。例如规则 1：当设置 a/b/ 目录下文件（未配置历史版本）创建满 180 天删除；规则 2： a/b/ 目录下文件配置 365 天删除，并且排除 a/b/test目录,执行结果为a/b/c/1.txt 文件会在创建满180 天后执行删除，a/b/test/2.txt 也会在 180 天后删除。
- **存储桶中若创建一条复杂规则，则存储桶生命周期管理整体应用复杂规则执行**

基础生命周期注意事项

通用注意事项

- 每个 Bucket 可以有至多 1000 条规则;
- BOS 生命周期规则设置后会在一天内生效;
- 规则生效后，BOS 会对符合条件的 Object 进行相应的处理，但处理需要一定的时间，**不一定能马上看到效果**。一般情况下，沉降或删除的时间为几小时，但若沉降数据量较大，则可能会在几天甚至更长的时间完成下沉或者删除；
- 规则中计算的时间（即 Object 的"年龄"）以 Object 的创建时间为准，而不是生命周期规则的创建／修改时间;
- **BOS 只保存文件的最后修改时间，即 last-modified 时间**；如果您不更新 meta 或者覆盖文件，那么 last-modified 就是创建时间。所以生命周期中的"创建时间"其实是 last-modified 时间。
- 基于文件访问时间记录的生效策略，目前仅北京和苏州区域支持。
- 低频存储、冷存储和归档存储的最低存储时间分别为 30 天，60 天和 180 天。您配置的生命周期沉降/删除规则需要满足最低存储时间的要求。若您配置的时间小于最低存储时间时间，控制台将会产生提示，请您根据提示中的要求进行配置。
- **单 AZ 类型文件仅能沉降到单 AZ 类型文件**，无法沉降到多 AZ 类型文件；**标准存储-多AZ 类型文件只能沉降到低频存储-多 AZ 类型文件**。
- 发生生命周期沉降时，原类型若需要取回，则会产生取回费用。同时，沉降完成后，原类型文件会被删除，也会产生请求费用。比如，一个低频存储类型文件沉降到冷存储，那么 BOS 需要先获取原低频存储文件，该操作产生取回费用；当文件成功沉降为冷存储文件后，原低频存储文件会被删除，该操作产生 Delete 请求费用，该请求费用包含在写请求费用中。
- 生命周期暂不支持对通过追加上传所生成的 Appendable Object 进行层级转换。

复杂生命周期规则注意事项

- tag的规则设置: 每个规则最多有10个对象标签，每个key长度小于等于128， value长度小于等于256。
- 每个rule下只有1个排除目录, 开启Not选项时，前缀和标签*必须至少存在*一项，即同时设置前缀和标签或者只设置前缀或标签，Not中的前缀或标签只有一个；排除前缀必须位于设置前缀之下，并和前缀的结构形式一样，例如前缀是 bucket/prefix，*not prefix可以是bucket/prefix1*
- 指定最小文件指定生命周期规则作用的最小文件大小，最小文件大小要求大于等于0字节，小于5TB。
- 指定最大文件指定生命周期规则作用的最大文件大小，最大文件大小要求大于0字节，小于等于5TB。

基础生命周期简单与复杂生命周期规则在执行层面的区别：

- 简单生命周期语义: 按照的是最长前缀的原则， 如果命中最长前缀，那么就不会去寻找其他规则。
- 复杂生命周期语义会寻找全局最优规则， 寻找符合prefix、 not、 filesize、 tag的规则下最早执行的规则。
- 了解更多功能细节，请参考开发者指南-基础生命周期管理。

控制台配置基础生命周期

1. 登录 对象存储 BOS 管理控制台。
2. 在左侧 **Bucket 列表** 中，点击 Bucket 名称。
3. 在左侧导航栏选择 **基础管理** > **生命周期配置**。
4. 在 **生命周期配置** 页面点击 **新增配置**，选择 **基础生命周期配置** 对该 Bucket 的生命周期进行配置。默认情况下该功能不开启。



配置项说明如下：

配置名称	配置描述
生效范围	本生命周期规则可以作用于整个Bucket，也可以作用于Bucket中某些带有特定前缀的对象，当选择"对前缀生效时"时，则需要填写对象前缀。 **例如：**如果一个 Bucket 中有以下几个 Object： `asset/user.png` `asset/company.png` • 如果前缀是asset/*，本规则就适用于两个以asset/开头的 Object； • 目前暂不支持对单独一个 Object 设置生命周期， 如果前缀是 asset/company.png，则规则无效。
生效策略	生效策略是指生命周期规则的生效方式，您可以选择"基于上传时间记录"和"基于访问时间记录"两种方式。 • 基于上传时间记录：根据文件的上传时间进行操作。如在创建 30 天后将文件删除，那么在文件上传后的第 30 天，该文件将被系统自动删除； • 基于访问时间记录：根据文件的访问时间进行操作。BOS 只记录生命周期规则建立后文件的访问时间，历史的访问时间不在统计范围之内。如文件 A 于 2020 年 1 月 1 日被最后一次访问，用户于 2020 年 2 月 1 日设置了基于访问时间记录的生命周期规则，且规则为最后一次访问时间后的 30 天将该文件下沉。那么这个 30 天从 2 月 1 日开始计算，而不是从 1 月 1 日。

5. 点击确认，完成规则添加。

6. 规则添加成功后，您可以在 **生命周期配置** 中看到已生成的规则记录。您可以根据自身需求对规则进行编辑和删除。

基础生命周期管理使用说明

基于最后一次修改时间的生命周期规则

基础生命周期管理文件生效策略支持两种方式，分别为基于最后一次修改时间(基于上传时间记录)与最近一次访问时间(基于访问时间记录)配置，配置基于文件最后一次修改时间（lastModified）的生命周期规则，定期将Object从热存储类型转为冷存储类型，或者删除Object，以降低存储成本，

基于最后一次修改时间适合业务数据冷热周期明确的场景，例如某医疗机构的医疗数据，上传到BOS后半年内少量访问，半年后基本不在访问，将上传将已上传180天的医疗档案转为冷存储。如果您希望BOS自动监测数据的访问模式并识别冷数据，然后将识别出来的冷数据进行存储类型的转换，从而达到数据的冷热分层存储，最终降低存储成本，您需要配置基于最后一次访问时间的生命周期规则。

使用限制

生命周期规则目前仅支持根据前缀和标签进行匹配，前缀与标签不支持通配符匹配

注意事项

- 采用SDK、API 方式设置需要注意PutBucketLifecycle为覆盖语义。例如，某个Bucket已配置了生命周期规则Rule01，您需要在Rule1基础上继续追加生命周期规则Rule2，您需要执行以下操作：

1. 调用GetBucketLifecycle接口获取当前生命周期规则配置Rule01。

2. 在Rule01基础上叠加Rule02。

3. 调用PutBucketLifecycle接口更新生命周期规则为Rule01+Rule02。

更多注意事项参见基础生命周期管理注意事项说明。

费用说明

> 关于通过生命周期规则转换Object存储类型或者删除Object必会产生转化与删除对应的请求费用，如果规则配置了删除或者转化存储类型且不满足规定的存储最小时长要求，还会产生最小存储周期费用。**规则配置时请特别注意删除或者转化文件时需满足对应存储类型的最小存储周期要求，否则会产生未到达最少存储周期费用**。

组成元素

指定生效范围：

- 按前缀匹配：按指定前缀匹配Object和碎片。可创建多条规则匹配不同的前缀，前缀不能重复。前缀的命名规范与Object命名规范相同。
- 按标签匹配：按指定标签的Key和Value匹配Object。单条规则可配置多个标签，仅当Object包含所有标签时执行生命周期规则。

基础生命周期规则设置的标签与执行说明： 例如基础生命周期规则设置的标签为a:1,b:2

Object携带的标签	生命周期规则是否作用于该Object
a:1	否
a:1,b:3	否
a:1,b:2	是
a;1,b:2,c:3	是

> 说明 标签匹配规则不作用于碎片。

- 按前缀+标签匹配：按指定前缀和标签的筛选条件匹配对象。
- 配置到整个Bucket：匹配整个Bucket内的所有Object和碎片。
- NOT元素：如果您希望按照生命周期规则对与前缀和标签匹配的Object进行相应处理的同时，跳过不需要处理的Object，您可以通过NOT元素对不需要处理的Object指定前缀和标签。

Object的过期时间及操作

- 过期天数：指定一个过期天数为N，为相对时间，在Object最后修改时间N天后过期（到期），并执行对应的操作，Object操作可以是转化存储类型或者删除。
- 过期日期：指定一个过期日期，为绝对时间，最后修改时间在该日期之前的Object全部过期（到期），并执行指定的操作，Object操作可以是转化存储类型或者删除。

注意以上说明适用于未开启版本控制的存储桶

碎片（Part）的过期时间及操作

- 过期天数：可指定一个过期天数N，文件碎片会在其最后修改时间的N天后被删除。

操作方式

1. 使用控制台
2. 使用BOS SDK **相关API** 以上操作方式底层基于API实现，如果您的程序自定义要求较高，您可以直接发起REST API请求。直接发起REST API请求需要手动编写代码计算签名。更多信息，请参见PutBucketLifecycle。

基于最后一次访问时间的生命周期规则

您可以通过基于最后一次访问时间（LastAccessTime）策略的生命周期规则来自动监测数据的访问模式并识别冷数据，然后将识别出来的冷数据进行存储类型的转换，从而达到数据的冷热分层存储，最终降低存储成本。

使用场景

- 相册或网盘场景

对于长时间没有访问的数据，希望设置自定义转储天数，自动将冷数据转为低频访问类型，并确保数据的实时访问。

- 生命科学场景

基因测序生成的大量业务数据，往往需要根据数据的最后访问时间而非最后修改时间来判断数据的冷热。按以往，客户只能手动通过日志分析或其他方式进行数据冷热的分层管理。但如果选用基于最后一次访问时间的生命周期规则，则可实现由服务端根据最后访问时间来自动识别冷热数据并实现数据分层存储。

使用限制

- 仅支持未开启版本控制的存储桶
- 不支持删除文件与碎片
- 基础生命周期规则目前仅支持根据前缀进行匹配，不支持通配符匹配。

注意事项

采用SDK、API 方式设置需要注意PutBucketLifecycle为覆盖语义。例如，某个Bucket已配置了生命周期规则Rule01，您需要在Rule1基础上继续追加生命周期规则Rule2，您需要执行以下操作：

1. 调用GetBucketLifecycle接口获取当前生命周期规则配置Rule01。
2. 在Rule01基础上叠加Rule02。
3. 调用PutBucketLifecycle接口更新生命周期规则为Rule01+Rule02。

更多注意事项参见基础生命周期管理注意事项说明。

转储的Object类型

基于最后一次访问时间的生命周期规则支持将Object从标准存储转为低频、冷或归档存储类型。更多存储类型介绍参见存储类型介绍。

设置日志查询

概述

访问对象存储 BOS 的过程中会产生大量的访问日志。日志查询功能将 BOS 与日志服务（BLS）相结合， 允许您在 BOS 控制台调用 BLS 服务查询 BOS 的访问日志，可实现对 BOS 操作审计、访问统计、异常事件回溯和问题定位等工作，提升工作效率并帮助您更好地基于数据进行决策。该功能需开通百度智能云日志服务（BLS）。

注意：

- 该功能基于百度智能云日志服务（BLS）实现对象存储日志实时查询，您需先开通日志服务（BLS）
- 该功能将产生一定费用，由日志服务单独收费，查看日志服务计费说明，开通服务后您需创建索引对实时日志进行检索与分析。

开通日志查询

1. 登录 对象存储 BOS 管理控制台。

2. 在左侧 Bucket 列表中，选择需要设置日志查询的 Bucket，点击 Bucket 名称进入 Bucket 管理目录。
3. 在【基础配置管理】页面中选择【实时日志查询】，单击【开通服务】进行开通：
   - 若您未开通日志服务（BLS），可点击【立即开通 BLS】前往开通。
   - 若您已开通日志服务（BLS），可点击【开通服务】开通 BOS 日志查询功能。



查询实时日志

方式一：通过BOS控制台

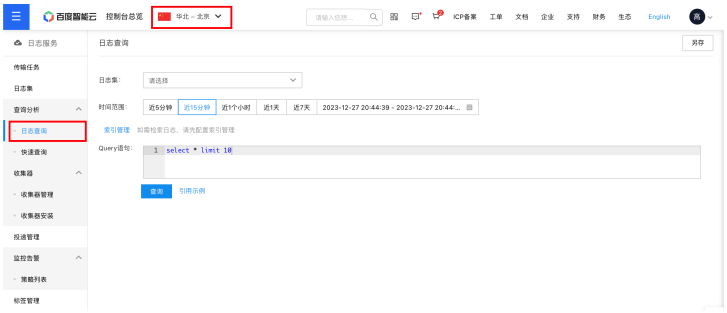
1. 登录 对象存储 BOS 管理控制台。
2. 在左侧 Bucket 列表中，选择已开通 BOS 日志查询功能的 Bucket，点击 Bucket 名称进入 Bucket 管理目录。
3. 在【基础配置管理】页面中选择【实时日志查询】，单击【立即查询日志】跳转至 BLS 页面查询。

注意

北京地域目前需在BLS控制台内在【索引管理】中首先完成索引配置，方可实现对开启后的增量日志进行分析与检索。

方式二：通过BLS控制台

1. 登陆 日志服务 BLS 控制台。
2. 在顶部选择已开通 BOS 日志查询功能的 Bucket 所属地域。
3. 在左侧导航栏中选择【查询分析】->【日志查询】。
4. 选择日志集，点击【查询】。



关闭日志查询

当您确认不再需要保留日志数据，请按如下步骤关闭实时日志查询。

1. 登录 对象存储 BOS 管理控制台。
2. 在左侧 Bucket 列表中，选择需要关闭日志查询的 Bucket，点击 Bucket 名称进入 Bucket 管理目录。
3. 在【基础配置管理】页面中选择【实时日志查询】，页面中点击【关闭服务】。
4. 在二次弹窗中仔细阅读点击【确认】即可。

设置版本控制

概述

版本控制功能为您提供bucket级别的数据保护能力。当您开启版本控制后，对文件的删除及覆盖操作将生成新的版本而不会对原有文件产生影响。当您需要恢复数据时，可将任意历史版本恢复到当前版本。

注意事项

- 当前仅支持广州和北京地域开启版本控制，北京地域需要申请功能白名单。
- **版本控制功能开启后只可切换至暂停状态，无法关闭**。
- 当前未支持归档类型文件的版本控制，默认存储类型为归档类型的bucket不允许开启版本控制。bucket开启版本控制后将无法上传或转换成归档类型的文件，删除操作也将直接删除归档类型文件的数据。
- append类型的文件追加写时不会产生新的版本。
- 功能选项中如果没有历史版本的选项，默认仅对最新版本进行操作。
- 产品功能说明详见版本控制概述 。

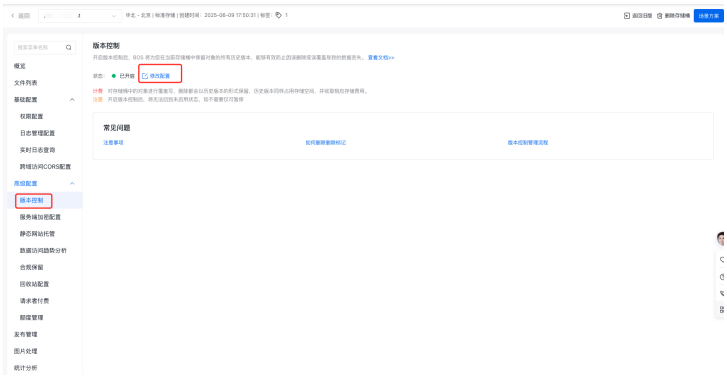
操作步骤

开启版本控制

按以下步骤开启bucket的版本控制功能：

1. 登录 对象存储控制台。
2. 在 **bucket列表** 中，单击需要配置版本控制功能的存储桶名称。

3. 在左侧导航 **高级配置** > **版本控制** 页面中，单击 **修改配置**，在弹出的侧边栏中选择**开启**，即可开启 bucket 版本控制功能。



暂停版本控制

1. 在 **bucket列表** 中，单击需要配置版本控制功能的存储桶名称。

2. 在左侧导航 **高级配置** > **版本控制** 页面中，单击 **修改配置**，在弹出的侧边栏中将状态切换至 **暂停** ，即可暂停bucket的版本控制功能。



计费说明

版本控制功能本身不收取任何费用，但历史版本与当前版本的文件一样都会占用存储空间产生存储费用。此外，若您对历史版本文件进行下载或恢复等操作，还会产生相应的请求费用、流量费用等。计费详情，请参见计费项概览。

数据访问趋势分析

概述

BOS 数据访问趋势分析功能，可通过分析 Bucket 或前缀粒度数据的访问趋势，帮助用户决策在何时将对应的数据转换为合适的存储类型，进而降低存储整体使用成本。同时数据访问趋势分析的结果内容，也可以帮助用户从访问变化的角度，分析业务层面的变化。

对某个 Bucket 或前缀开启数据访问趋势分析规则后，通常需要约 14 天时间来完整分析访问数据，以收集足够信息来获得完整周期的分析结果。在此期间，也可以在控制台查看所生成的部分分析内容。

> **说明：**
>
> - 该功能将根据分析的对象（文件）个数计费，了解价格详情参见 BOS价格说明
> - 当前该功能在华北-北京、华东-苏州和华南-广州区域开放
> - 针对存储大量文件的存储桶，一至两天的全量结果生成延迟属于正常情况
> - 对于已开启多版本的 Bucket，暂时不支持配置数据访问趋势功能

使用场景

- 优化存储使用成本。通过分析数据的访问趋势，帮助用户决策何时将数据沉降或回暖至更匹配的存储类型
- 帮助业务侧根据数据访问变化趋势来分析业务的变化

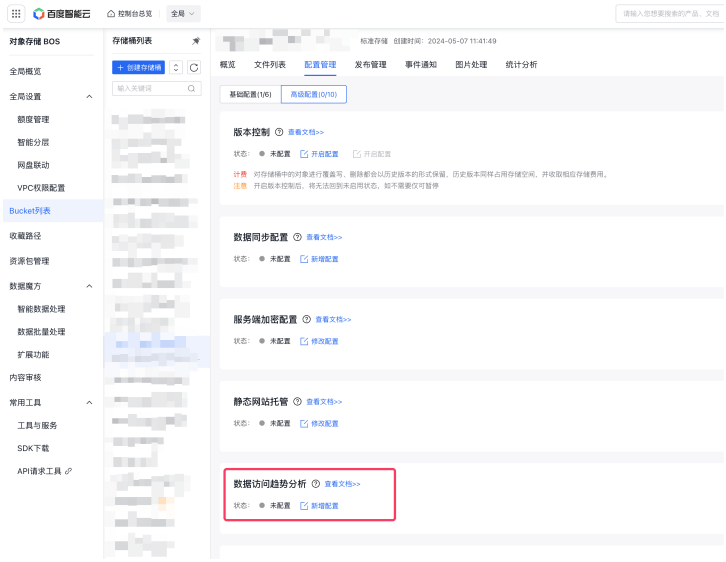
设置数据访问趋势分析规则

1. 登录 对象存储 BOS 管理控制台。

2. 在左侧 Bucket 列表中，选择需要设置权限的 Bucket，点击 Bucket 名称进入 Bucket 管理目录。

3. 在上方导航栏选择 **配置管理** 页签。

4. 在 **配置管理** 页面中选择 **高级配置**，在 **数据访问趋势分析** 区域点击 **新增配置**，创建规则。



5.设置规则



6.编辑或者删除规则

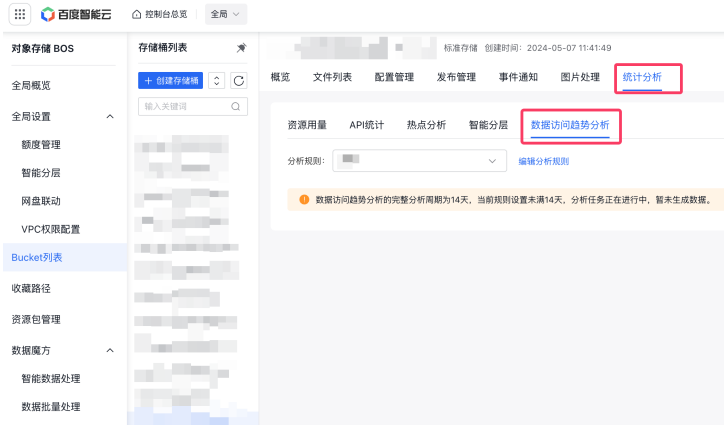
可以对已创建的规则进行编辑，也可以将已创建的规则删除。



7.查看分析结果

可以点击规则操作栏中的**查看数据分析**，或者可以在 **统计分析** 区域点击 **数据访问趋势分析**，进行查看。





管理Object

上传文件

概述

BOS 控制台的文件上传功能，实际上是通过 BOS 的简单上传方式，将文件通过 PutObject 接口上传到您已经创建的 Bucket 中。

前提条件

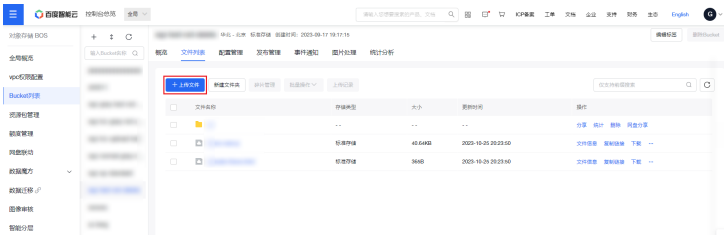
- 已注册百度智能云账号。具体操作请参见注册百度智能云账号。
- 已创建了 Bucket。详情请参见创建 Bucket。

注意事项

- BOS 目前不支持多版本控制。若您重复上传一个文件（文件名称相同），之前上传的数据将被覆盖。
- BOS 控制台的文件上传功能，支持任意类型的文件，单次最多可上传 10000 个文件，且单文件支持最大 512 GB。若您上传的文件大小超过 512 GB，请使用 SDK 工具进行上传。BOS API、SDK 和工具支持上传最大为 48.8 TB的单个文件。
- 对于上传的文件，其存储类型支持标准存储-多AZ、标准存储、低频存储-多AZ、低频存储、冷存储和归档存储共 6 种存储类型，您可以根据数据的使用频率，对文件的存储类型进行设定。关于存储类型的选择，您可以参考如何选择合适的存储类型。不同地域开放售卖的存储类型有所不同，请您提前查看开放地域。
- 若子用户需通过控制台上传文件，则在创建子用户时访问方式需同时设置【编程访问】与【控制台密码访问】，且授权子用户文件上传权限。
- 上传任务创建后将在后台自动上传，您可随时选择文件列表页的【上传记录】进入任务列表查看任务进度。**若您关闭或刷新浏览器，上传任务将会被自动取消，并清除全部记录。在有文件未上传成功之前，请不要关闭或刷新浏览器。**

上传文件

1. 登录 BOS 管理控制台，进入全局概览页面。
2. 在 BOS 控制台左侧的导航栏中选择**目标 Bucket**，点击进入该 Bucket 的文件列表页。
3. 在文件列表页点击页面上方【上传文件】，右侧出现上传文件窗口。



4. 您可选择将对象上传到**当前目录**或者**指定目录**，点击上方进行入口切换。若您需要将文件或文件夹上传到当前Bucket下的指定目录，则**需要在输入栏手动输入目录名称，以"/"结尾**。

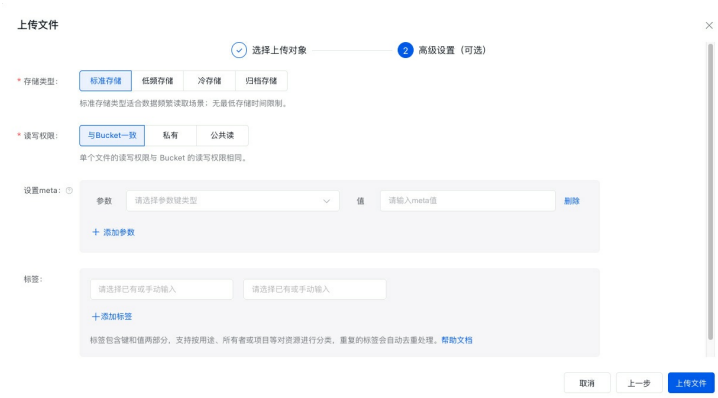


5. 确认上传目录后，点击【上传文件】选择文件，即将上传的文件将会在页面中展示，底部会展示总共的文件数量以及文件大小。若您需要验证MD5，可勾选底部**支持验证MD5**，我们会自动为您验证MD5。

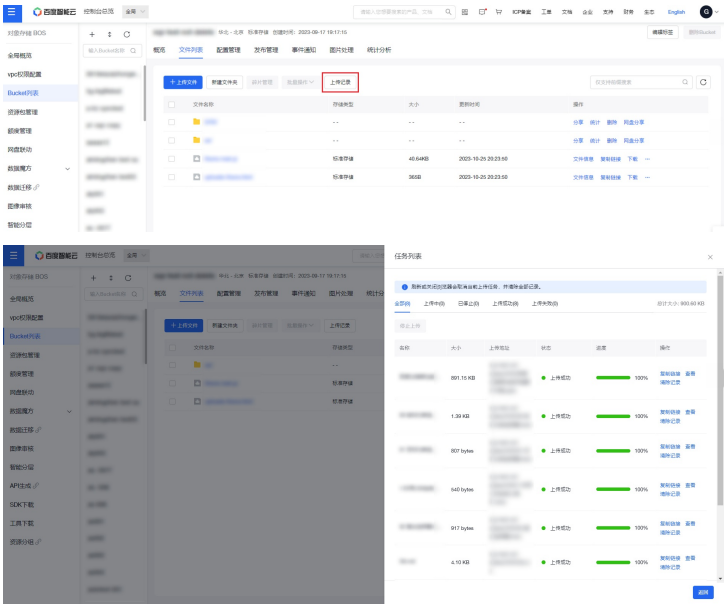


6. 确认上传文件无误后，点击【下一步：高级配置（可选）】设置对象属性，支持为上传文件批量设置存储类型、访问权限、meta信息、标签，您可根据不同需求为上传文件设置不同的属性。确认无误后点击【上

传文件】创建上传任务。



7. 您可点击文件列表页顶部的【上传记录】按钮进入任务列表，查看上传任务进度及状态。在任务列表中，对于已经上传成功的文件，您可以选择操作列中的【复制链接】直接复制文件链接，分享给第三方预览或下载；也可选择【查看】跳转到当前文件路径下。



8. 上传完成的文件会即刻展示在 Bucket 的文件列表中。

常见问题

- BOS 如何上传下载文件或者文件夹？

  答：BOS 支持以下多种方式上传文件：

  1. 控制台直接上传 Object或文件夹。控制台最大支持上传 512G 的数据，批量上传下载或超过 512G 的大文件建议使用SDK或周边工具。
  2. 使用 API 或 SDK 进行上传下载，其中文件夹上传可以参见模拟文件夹功能；
  3. 使用周边工具进行上传下载。

相关 API

- PutObject 接口：使用 PutObject API 接口上传 Object。

下载文件

概述

您可以在 BOS 控制台上下载文件、预览文件。

注意事项

归档类型的文件不支持直接下载，必须先在文件信息抽屉里点击**文件取回**按钮取回Object后，再进行下载操作。

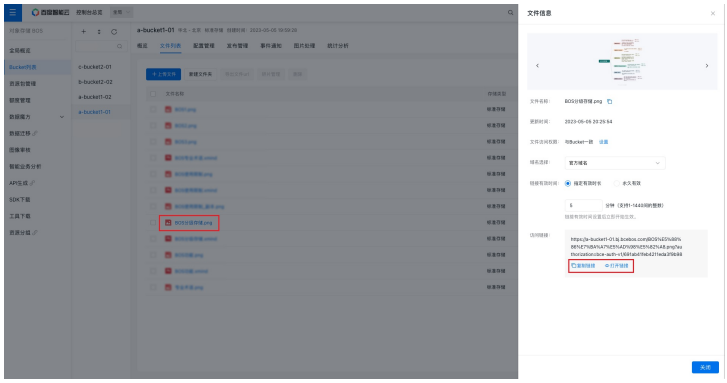
下载文件

1. 登录 BOS 管理控制台，进入全局概览页面。
2. 在 BOS 控制台左侧的导航栏中选择目标文件所在的**Bucket**，点击打开该 Bucket 的文件列表页。
3. 在文件列表中，点击操作栏的**下载**按钮后即可下载。



预览文件

1. 登录 BOS 管理控制台，进入全局概览页面。
2. 在 BOS 控制台左侧的导航栏中选择目标文件所在的**Bucket**，点击打开该 Bucket 的文件列表页。
3. 预览文件： - **控制台预览：**在文件列表中，点击要预览的Object的**文件名称**或者操作栏的**文件信息**，在弹出的文件信息抽屉中，可对图像、视频等文件进行预览。 - **URL预览：**点击要预览文件右侧的**复制链接**并在新的浏览器窗口中粘贴已复制的下载地址；也可在文件信息抽屉中点击**复制链接**复制文件地址或者点击**打开链接**直接打开文件地址，可对图像、视频等文件进行预览。



复制和移动文件

概述

您可以对已上传的文件行复制和移动。文件复制是指将文件复制到其他 Bucket 中且保留源文件，文件移动是指将文件移动到其他 Bucket 中且从原 Bucket 中删除。

注意事项

- 文件支持跨区域进行复制和移动。
- 未取回的归档类型文件不支持复制和移动操作，您需先取回文件后进行操作。

复制文件

1. 登录 BOS 管理控制台，进入全局概览页面。
2. 在 BOS 控制台左侧的导航栏中选择目标文件所在的**Bucket**，点击打开该 Bucket 的文件列表页。
3. 点击目标文件右侧操作栏的**更多->复制文件**按钮。
4. 在弹窗中选择需要复制到的**目标文件夹**，并选择复制文件在目标文件夹中的**存储类型**。文件复制后，原 Bucket 中文件依旧会保留。



> **说明：**您可以通过把文件复制到自身并设置相应存储类型来实现存储类型转换。

移动文件

1. 登录 BOS 管理控制台，进入全局概览页面。
2. 在 BOS 控制台左侧的导航栏中选择目标文件所在的**Bucket**，点击打开该 Bucket 的文件列表页。
3. 点击目标文件右侧操作栏的**更多->移动到**按钮。
4. 在弹窗中选择需要移动到的**目标文件夹**，并选择文件在目标文件夹中的**存储类型**。文件移动后，原 Bucket 中文件将会删除。



相关 API

- CopyObject 接口：使用 CopyObject API 接口把一个已经存在的 Object 拷贝为另外一个 Object。

注意事项

数据跨境合规承诺 您使用该产品或该功能，将使您在云上的业务数据传输至您所选择的区域或产品部署区域，可能会涉及数据跨境。

使用该功能代表您同意并确认，您完全拥有对相关业务数据的合法处置权限，并对您的数据传输行为或指令行为全权负责。您确认并承诺，您的数据传输符合所有适用法律的规定，包括并不限于获得数据主体充分、明示同意，完成数据出境安全评估，与接收方签署个人信息出境标准合同等法定义务，且您承诺您的业务数据不含任何所适用法律限制、禁止传输或披露的内容（具体合规要求请您咨询相关主管部门）。

如您未遵守前述声明与保证，您将承担全部法律后果，并赔偿百度智能云和/或其关联公司遭受的全部损失。

如因法律法规、监管政策规定导致百度智能云须变更或停止提供本产品功能，您理解并同意这并不属于百度智能云违约。百度智能会为您提供服务变更或停止的过渡期协助。

重命名文件

概述

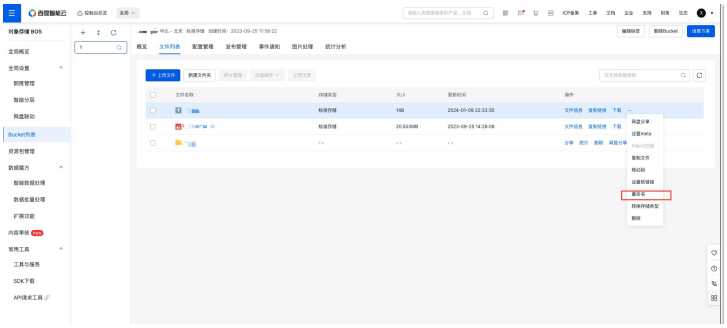
您可以为已上传的文件进行重命名操作。

注意事项

- 归档类型的 Object 不支持重命名操作。

重命名文件

1. 登录 BOS 管理控制台，进入全局概览页面。
2. 在 BOS 控制台左侧的导航栏中选择重命名文件所在的**目标 Bucket**，点击打开该 Bucket 的文件列表页。
3. 点击目标文件右侧操作栏的**更多->重命名**，填写新的文件名，点击**确定**。
4. 重命名成功的文件会即刻展示在 Bucket 中的文件列表中。



删除文件

概述

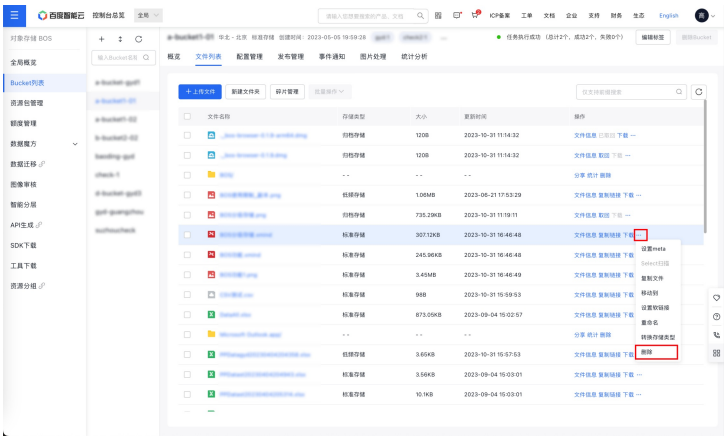
您可以在 BOS 控制台删除已上传文件。

注意事项

- 文件删除后无法恢复，请谨慎操作。
- 存储类型为低频存储、冷存储和归档存储的文件，具有最低存储时间的限制，若您在最低存储时间内删除文件，会收取一定的额外费用。详情参见 BOS计费文档。

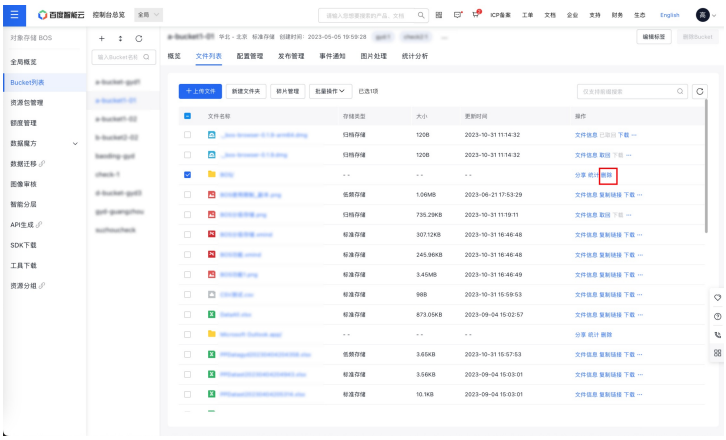
删除文件

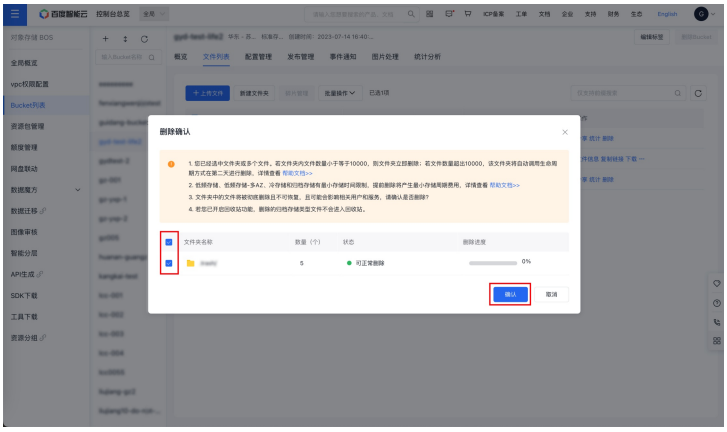
1. 登录 BOS 管理控制台，进入全局概览页面。
2. 在 BOS 控制台左侧的导航栏中选择目标文件所在的**Bucket**，点击打开该 Bucket 的文件列表页。
3. 点击目标文件右侧操作栏的【...】中【删除】按钮。
4. 在删除文件对话框中，单击【确定】，即可删除相应的文件。



删除文件夹

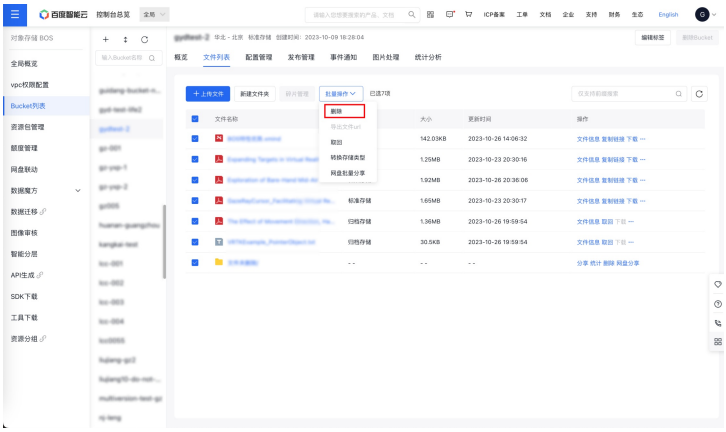
1. 登录 BOS 管理控制台，进入全局概览页面。
2. 在 BOS 控制台左侧的导航栏中选择目标文件夹所在的**Bucket**，点击打开该 Bucket 的文件列表页。
3. 找到待删除的文件夹，点击文件夹操作栏中的【删除】按钮。
4. 在删除确认对话框中，勾选确定要删除的文件夹，单击【确定】，即可删除文件夹。文件夹删除后，文件夹中包含的文件都将被删除，请谨慎操作。





批量删除文件/文件夹

1. 登录 BOS 管理控制台，进入全局概览页面。
2. 在 BOS 控制台左侧的导航栏中选择目标文件所在的**Bucket**，点击打开该 Bucket 的文件列表页。
3. 勾选一个或多个文件/文件夹，点击上方【批量操作】中的【删除】按钮。
4. 在删除文件对话框中，勾选确定要删除的文件/文件夹，单击**确定**，即可批量删除。



相关 API

- DeleteObject 接口：删除指定 Bucket 的一个 Object。
- DeleteMultipleObjects 接口：删除同一个 Bucket 下的多个 Object。

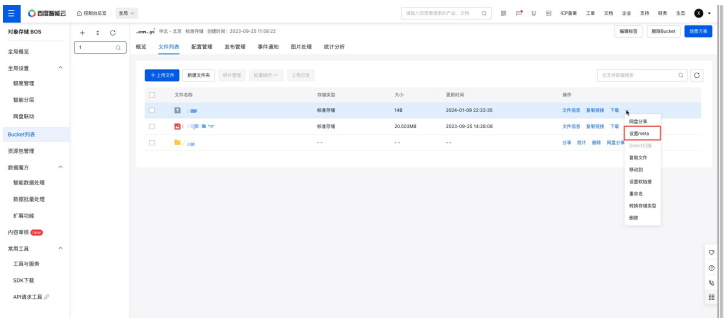
设置Meta信息

概述

数据元信息（Object Meta）是文件所携带的元数据信息，用以标识文件数据特征。对上传到 BOS 的文件属性描述，可以将其分为两种：系统定义的 Meta 和 User Meta（用户自定义元信息）。数据元信息可以在各种方式上传时或者拷贝文件时进行设置。 关于 Meta 的详细描述请参见Metadata。

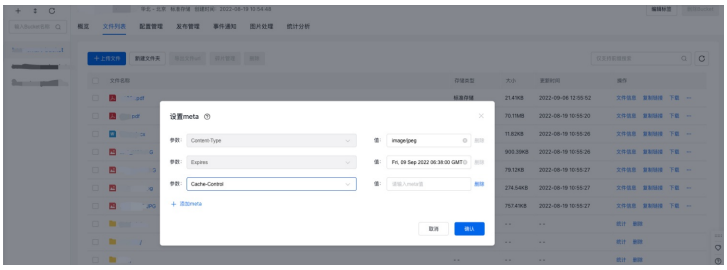
设置 Meta 信息

1. 登录 BOS 管理控制台，进入Bucket列表页面。
2. 在 BOS 控制台左侧的导航栏中选择目标文件所在的**Bucket**，点击打开该 Bucket 的文件列表页。
3. 点击目标文件右侧操作栏的**更多->设置meta**按钮。



4. 在弹框中，您可以 **添加Meta** 或对文件的 Meta 信息进行修改。Metadata的详细描述如下：

参数	说明
Content-Type	Content-Type是HTTP协议header中一个重要的参数，用于标识发送或接收到的数据的类型，浏览器根据该参数来决定数据的打开方式。常见Content-Type有image/jpeg、text/xml、text/html、application/octet-stream等。
Expires	Expires是用于设置文件缓存失效时间的参数，在用户未手动设置的情况下，BOS会自动分配一个动态失效时间，用户可通过修改Expires参数值修改时间。时间格式示例：Fri, 09 Sep 2022 06:37:19 GMT。
Content-Disposition	Content-disposition是MIME协议的扩展，MIME协议用于指示MIME用户代理如何显示附加的文件。例如Content-disposition可以用来将用户请求所得的内容存为一个文件：attachment; filename="image.jpg"。
Cache-Control	Cache-Control用于控制网页的缓存，常见的取值有private、no-cache、max-age、must-revalidate等。
x-bce-meta	用户自定义meta，可以用来保存自定义信息。例如用来保存文件标签，参数用"x-bce-meta-tag"，值用"cats"。



5. 设置完成后，点击**确定**，页面右上角弹出"Meta设置成功"字样，Meta 信息设置成功。

设置文件权限

概述

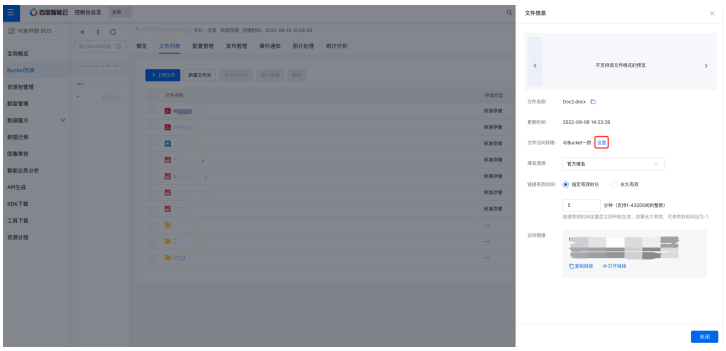
您可以在 BOS 控制台对文件的访问进行设置。BOS 控制台支持粗粒度访问权限设置，可以将其设置为公共读、私有或自定义权限。在不修改文件权限的情况下，该文件的访问权限将和所在的 Bucket 权限保持一致。

注意事项

- 归档存储类型文件不支持设置文件访问权限。

设置文件权限

1. 登录 BOS 管理控制台，进入Bucket列表页面。
2. 在 BOS 控制台左侧的导航栏中选择目标文件所在的**Bucket**，点击打开该 Bucket 的文件列表页。
3. 点击目标**文件名称**或者操作栏中的**文件信息**，页面右侧弹出"文件信息"窗口。
4. 在文件信息页面中，点击"文件访问权限"后的**设置**按钮，对文件的读操作进行权限调整。



5. 文件权限设置中，文件权限默认与所属 Bucket 的权限一致。您可根据需要将文件的访问权限设置为**公共读**、**私有**或**自定义**权限。当文件读操作权限与所属Bucket权限不一致时，以文件权限为准。权限说明如下：
    - **公共读：**任何用户均可读取该object。

- **私有**：除本人外，其余用户都不可读取该object。
- **自定义**：进行对象级别精细权限管理，支持用户通过添加百度智能云账号ID为为指定用户分配权限。
  - 只读：用户仅可读取该object。
  - 读写：用户被授予FULL_CONTROL权限，对该object可进行读写操作。



6. 点击**确定**修改完成，页面弹出“保存成功”，文件信息页面中的**文件访问权限**立即改变。



相关 API

- PutObjectAcl 接口：使用 PutObjectAcl API 设置 Object 的访问权限。
- GetObjectAcl 接口：使用 GetObjectAcl API 获取某个 Object 的访问权限。
- DeleteObjectAcl 接口：使用 DeleteObjectAcl API 删除某个 Object 的访问权限。

管理文件夹

概述

为了更加方便的管理数据，BOS 控制台提供了文件夹功能，您可以根据需求在 Bucket 中创建文件夹用以存放文件，也可以在不需要的时候删除文件夹。

新建文件夹

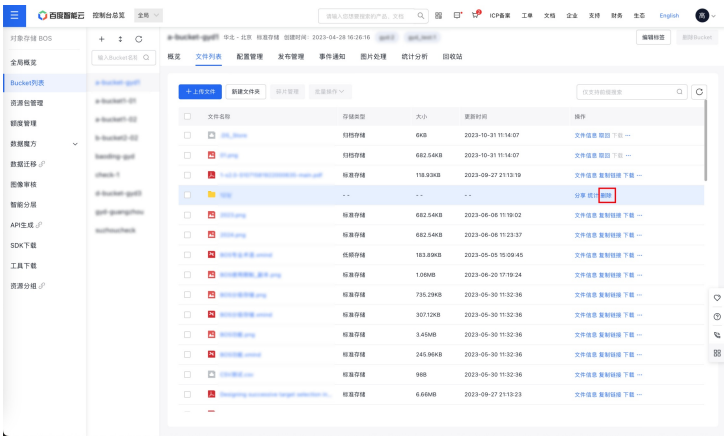
1. 登录 BOS 管理控制台，进入全局概览页面。
2. 在 BOS 控制台左侧的导航栏中选择 **目标Bucket**，点击打开该 Bucket 的文件列表页。
3. 在文件列表页面上方，点击【新建文件夹】，在弹出浮层中按要求创建新文件夹。

删除文件夹

1. 登录 BOS 管理控制台，进入全局概览页面。
2. 在 BOS 控制台左侧的导航栏中选择目标文件所在的 **Bucket**，点击打开该 Bucket 的文件列表页。

删除单个文件夹

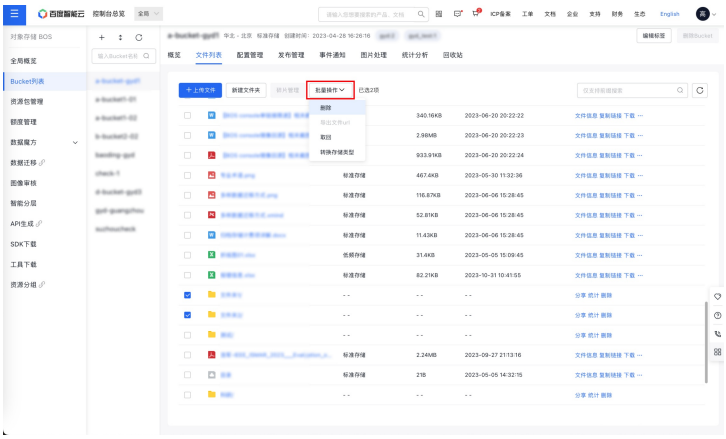
1. 找到待删除的文件夹，点击对象右侧操作列中的【删除】按钮。
2. 在删除文件对话框中，单击【确定】，即可删除文件夹。文件夹删除后，文件夹中包含的文件都将被删除，请谨慎操作。



删除多个文件夹

1. 勾选您需要删除的多个文件夹，点击上方【批量操作】中的【删除】按钮。

2. 在删除文件对话框中，单击【确定】，即可删除多个文件夹。文件夹删除后，文件夹中包含的文件都将被删除，请谨慎操作。



碎片管理

概述

进行分片上传时，文件（Object）会被切成若干个 Part，这些 Part 被称为碎片。当您将这些 Part 上传到 BOS 后，可以调用 CompleteMultipartUpload 将这些碎片（Part）组合成一个完整的Object。Part 是指由 Multipart（分块）上传过程产生，请参考 Object 分块上传。

注意事项

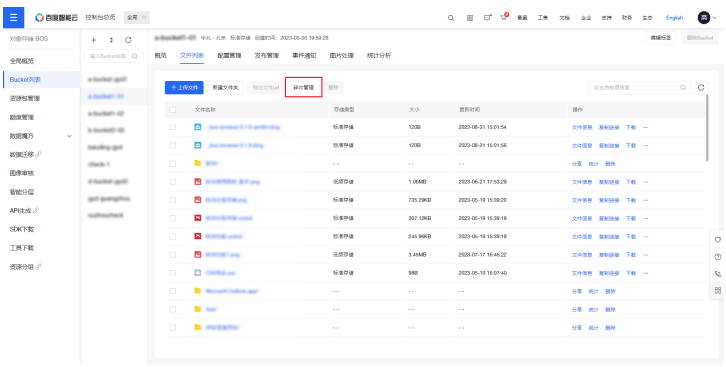
- Part存续期间会计入bucket容量并进行计费，将按照上传时指定的存储类型收费，如您有不再需要使用的碎片文件，请按需删除。对于每种存储类型的存储价格、流量费用、请求费用和数据处理费用，请查看计费项概览了解计费项详细信息。
- 您也可以通过调用 CompleteMultipartUpload，将这些 Part 组合成一个完整的 Object。MultipartUpload使用详情请参考简介。
- 您也可以通过设置生命周期管理来定时清理不需要的 Part，此配置可以用于清理 Bucket 中长期没有完成 CompleteMultipartUpload 的 Part，减少空间占用。具体操作请参见设置生命周期规则。
- Part 在完成 Object 组合前，不能被读取。
- 删除 Bucket 时，必须先删除 Object 和 Part。碎片主要是由分片上传操作产生，具体可参见Object 分块上传。

碎片管理

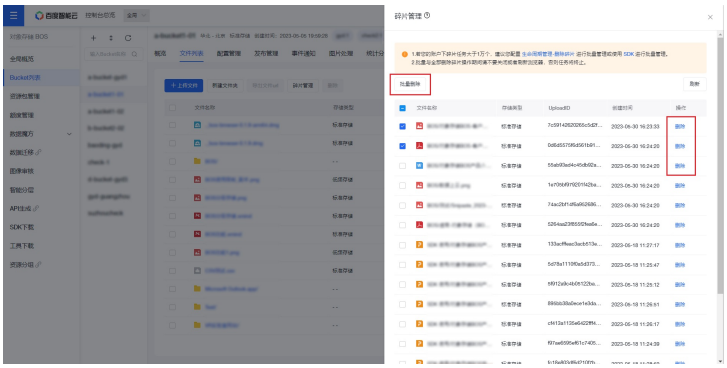
1. 登录 BOS 管理控制台，进入全局概览页面。

2. 在 BOS 控制台左侧的导航栏中选择目标 Bucket，点击文件列表进入该 Bucket 的文件列表页。

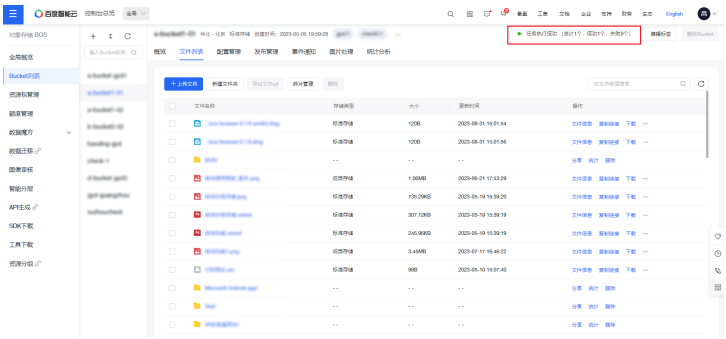
3. 在当前文件列表页顶部点击【碎片管理】，BOS 会列出当前Bucket的所有碎片文件。

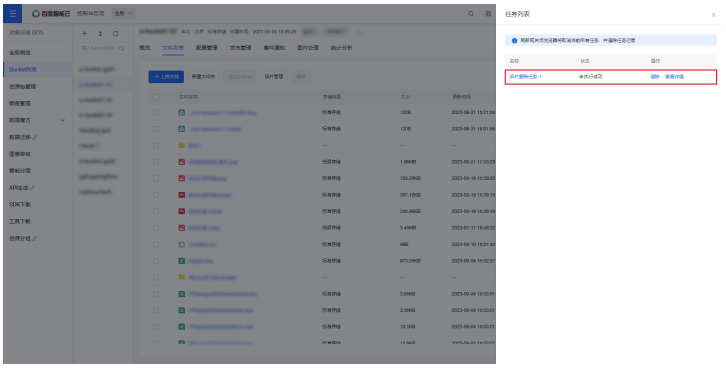


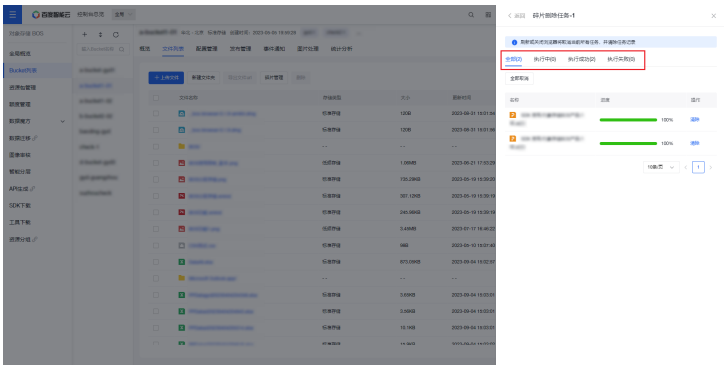
4. 若您需要删除碎片文件，您可以在操作列选择【删除】按钮删除单个碎片文件，也可以勾选多个碎片文件后，点击顶部【批量删除】按钮进行批量删除操作。



5. 批量删除碎片文件任务创建后，将在后台自动进行，您可点击右上角进入任务管理查看批量任务执行状态。







注意：

- 若您的账户下的碎片任务大于1万个，建议您配置 生命周期管理-删除碎片 进行批量管理或使用 SDK 进行批量管理。
- 若您需要对碎片文件进行删除操作，请确保删除的碎片文件中没有需要完成的分片，**删除操作不可逆转，请谨慎操作。**
- 批量删除碎片期间请不要关闭或者刷新浏览器，否则任务将终止。

相关 API

- InitiateMultipartUpload 接口：此命令向BOS请求一个全局唯一的UploadId，用于表示此次MultipartUpload。
- UploadPart 接口：在调用 InitiateMultipartUpload 获取 UploadId 后，我们需要用 UploadPart 命令来上传 Object 拆分后的数据（即 Part）。
- UploadPartCopy 接口：在调用 InitiateMultipartUpload 获取 UploadId 后，我们需要用 UploadPartCopy 命令来复制 Object 拆分后的数据分片（即 Part）。
- CompleteMultipartUpload 接口：使用该API完成整个 MultipartUpload 操作。
- AbortMultipartUpload 接口：可以使用此接口来中断某个MultipartUpload请求，BOS收到此命令后，将会清除已上传的数据，用户可以通过此命令删除碎片文件。
- ListParts 接口：用于列出用户指定UploadId所属的所有已经上传成功的Part，用户可以通过此命令查看当前的进度。
- ListMultipartUploads 接口：用于列出指定Bucket下面的所有未执行完成的Multipart Upload。

转换文件存储类型

概述

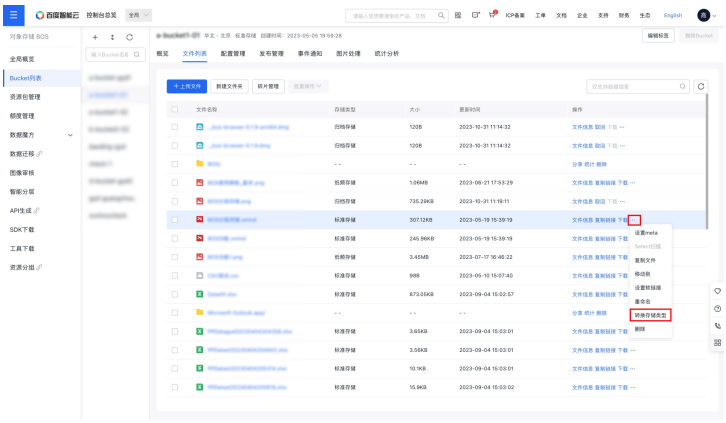
存储在 BOS 中的文件，您可以在任意时间内转换其存储类型。当前，BOS 支持将存储类型在标准存储、低频存储、冷存储和归档存储之间转换，或者在标准存储-多AZ 和低频存储-多AZ 之间转换。

注意事项

- 归档存储类型文件取回完成后才能转为其他存储类型；
- 通过Append上传的文件不支持转为归档类型；
- 冷/低频存储转化存储类型会产生数据取回费用，详情见 请求费用；
- 低频存储、冷存储和归档存储类型文件具有不同最小存储周期，若未存满相应的最小存储周期，BOS一次性收取最小存储周期费用，详情见 存储费用说明。

转换文件存储类型

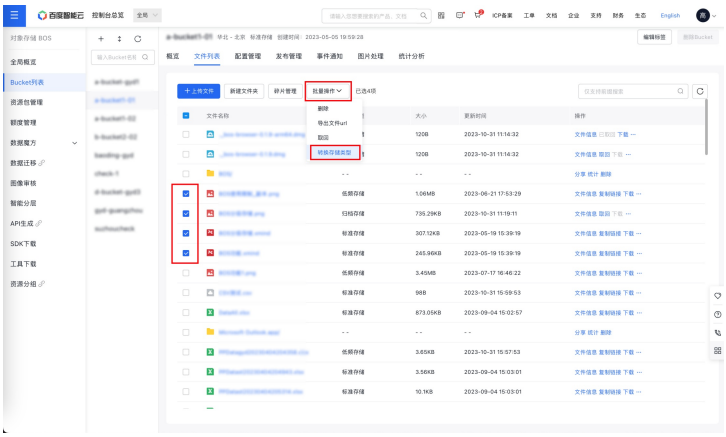
1. 登录 BOS 管理控制台，进入全局概览页面。
2. 在 BOS 控制台左侧的导航栏中选择目标文件所在 Bucket，进入该 Bucket 的文件列表页。
3. 点击目标文件右侧操作栏【...】中【转换存储类型】按钮。

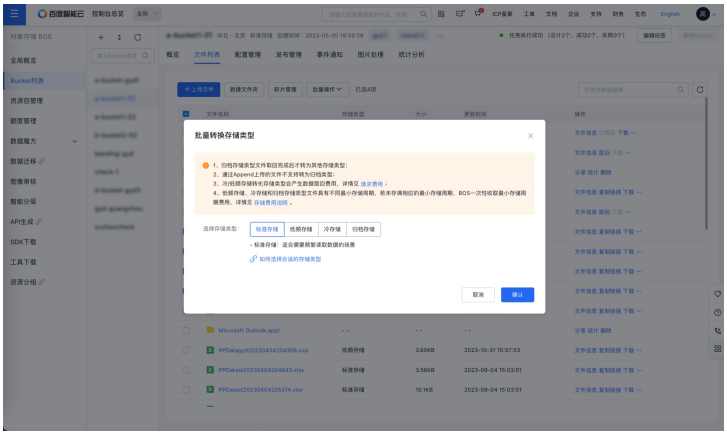


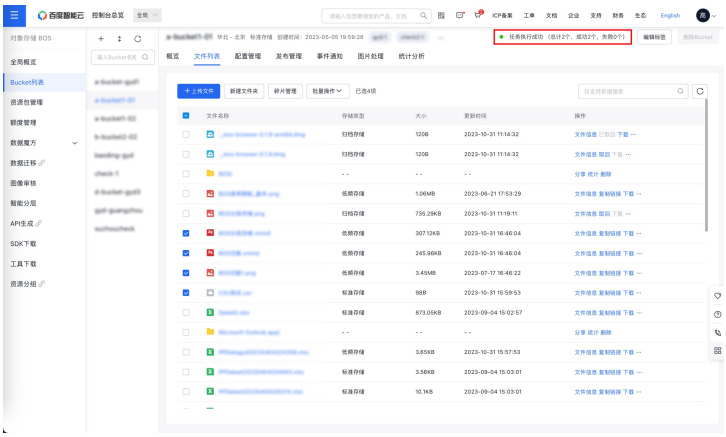
4. 在弹窗中选择需要转换的**目标存储类型**后，点击**确认**即可转换完成。

批量转换存储类型

1. 登录 BOS 管理控制台，进入全局概览页面。
2. 在 BOS 控制台左侧的导航栏中选择目标文件所在 Bucket，进入该 Bucket 的文件列表页。
3. 勾选需要进行批量转换存储类型的文件/文件夹，点击上方【批量操作】中【转换存储类型】
4. 在弹窗中选择需要转换的**目标存储类型**后，点击**确认**即可创建批量存储类型转换任务。
5. 可在右上角的任务状态中查看批量转换任务进度。









Select扫描

概述

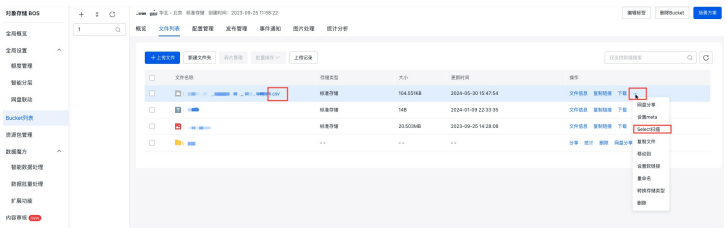
BOS Select 扫描功能，支持通过简单的 SQL 语句对 BOS 单个文件的内容进行检索和查询，从而获取您希望获取的数据。通过 BOS Select 扫描功能，您在控制台即可完成文件内容的检索和查询，不需要将文件下载到本地后进行查询，从而有效节省带宽资源，并提高数据查询的效率。Select 扫描是 BOS 的一款增值服务，会产生相应费用。详情请查看计费概览。

注意事项

- 支持检索的文件类型：
  - CSV 文件：UTF-8 编码的 RFC 4180 标准 CSV 文件，包括 TSV 等类 CSV 文件，且文件的行列分隔符以及 Quote 字符都可自定义；
  - JSON 文件：包括 DOCUMENT 和 LINES 两种 JSON 类型。其中，DOCUMENT 是指整个文件是单一的 JSON 对象， 而 LINES 表示整个文件由多行的 JSON 对象组成，但整个文件本身并不是一个合法的 JSON 对象，行与行之间以换行分隔符隔开；
- 支持检索的存储类型：标准存储、低频存储和冷存储类型文件；
- 支持扫描服务端加密之后的文件；
- 支持扫描通过 GZIP 方式压缩后的文件；
- 支持的 SQL 语法：支持 Select From Where Limit SQL 句型，支持 string、int(64bit)、float(64bit), timestamp，bool 和 decimal(128) 的数据类型，并支持逻辑条件（AND,OR,NOT），算术表达式（+-*/%），比较运算符（>,=, <, >=, <=, !=），匹配运算符（LIKE，BETWEEN+AND，IN），空判断（IS NULL），聚合函数（AVG、COUNT、MAX、MIN、SUM），转换函数 CAST 及别名关键词 AS；
- 控制台目前仅支持扫描 180 MB 以下的文件，且仅提取扫描结果的前 40 MB 数据。如果您需要处理更大的文件或返回更多的记录，请调用 Select Object API 接口 ；
- Select 扫描是 BOS 的增值服务，会按照您扫描的源文件大小（GB）收取一定费用，详细信息请查看计费项信息和使用价格。若因 SQL 语法或系统等错误导致扫描失败，该次扫描不收取费用。

Select扫描

1. 登录 BOS 管理控制台，进入全局概览页面。
2. 在 BOS 控制台左侧的导航栏中选择扫描文件所在的**目标 Bucket**，点击打开该 Bucket 的文件列表页。
3. 点击目标文件右侧操作栏的**更多->Select扫描**。Select 扫描是针对大数据场景的一种云上操作，对文件类型有一定要求。只有 .csv 和 .json 格式的文件支持使用 Select 扫描，其他类型文件不支持。



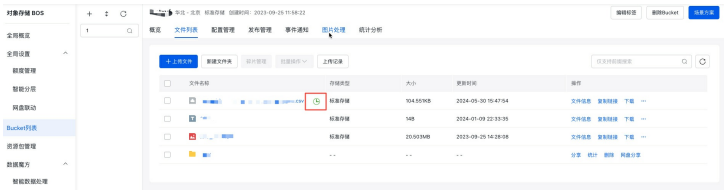
4. 在右侧窗口中，您需要先选取要进行扫描的文件，然后对源文件进行配置。

文件类型	配置名称	配置描述
CSV	标题字段	• 包含：表示扫描的文件中有表头（Header）信息； • 不包含：表示扫描的文件中有表头（Header）信息； • 不论文件是否包含表头信息，控制台均仅支持通过列序号取出某一列，不支持通过表头取出。若您希望用过表头取出某一列，请使用 Select API
	分隔符	• 指定 csv 文件列分隔符，以 Base64 编码，最多为1个字符； • 分隔符默认为逗号，还支持\t、\n、Tab，分号和自定义
	压缩方式	• 指定查询的 Object 是否压缩，可选值为无压缩或 GZIP
	导出格式	当前仅支持导出 CSV 格式的扫描结果
	加密格式	Select扫描支持对加密文件进行扫描，控制台自动识别扫描文件的加密状态
JSON	JSON类型	• 查询的 Json Object格式，支持"DOCUMENT"或"LINES"
	压缩方式	• 指定查询的 Object 是否压缩，可选值为无压缩或 GZIP
	导出格式	当前仅支持导出 CSV 格式的扫描结果
	加密格式	Select扫描支持对加密文件进行扫描，控制台自动识别扫描文件的加密状态



5. 配置完扫描文件之后，您还需要输入 SQL 语句，点击运行即可进行文件扫描。BOS为您提供了若干常用 SQL 语句模板，您可以根据需要选择语句。CSV 文件常用 SQL 语句可以查看 CSV 常用 SQL 语句，JSON 文件常用 SQL 语句可以查看 JSON 常用 SQL 语句。
6. 查看扫描结果。点击立即扫描后，可以查看扫描结果。若扫描失败，BOS 会在结果框中返回相应错误码，此时扫描不收取费用。

> **注意**
>
> - 当您关闭扫描窗口后，本次 Select 扫描结果会自动缓存在您的浏览器中，同时会在文件列表对应文件后方生成一个闹钟提示。当您再次打开扫描窗口，仍可以查看扫描结果。若您对同一文件再次发起扫描，或关闭浏览器，则缓存结果会被自动清除。



设置软链接

概述

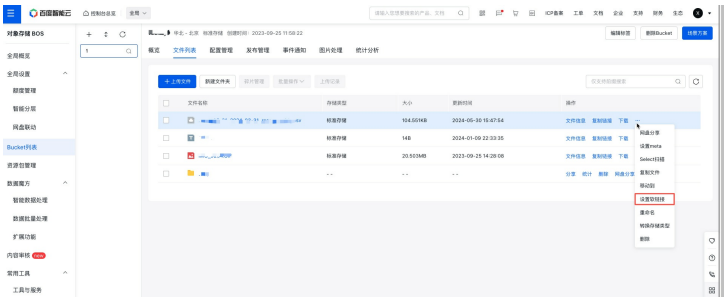
软链接功能用于快速访问对象存储空间内的常用文件。设置软链接后，您可以通过软链接文件快速打开源文件，类似于 Windows 系统中的快捷方式。

注意事项

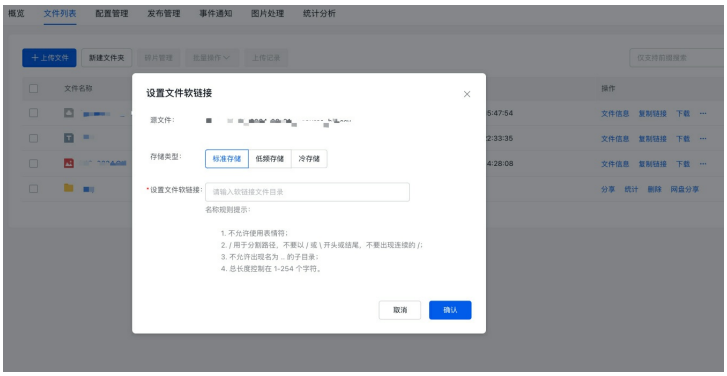
- 软链接文件和源文件可以位于相同region内的不同bucket。
- 目前支持对标准存储、低频存储、冷存储和归档存储类型的源文件设置软链接文件，但软链接文件不支持设置为归档存储类型。
- 创建软链接文件时，BOS 不会检查源文件是否存在，以及创建者是否有权限访问目标文件。创建完成后读取软链接文件时，BOS 会检查软链接文件和源文件是否存在，并检查您对文件的读取权限。
- 创建软链接文件需要有相应写权限。
- 如果不指定"x-bce-forbid-overwrite" Header，且存在与软链接文件同名的文件时，软链接操作时会默认覆盖同名文件，请您仔细确认。
- 不支持为已创建的软链接文件继续创建软链接文件（二级软链）。

设置软链接文件

1. 登录 BOS 管理控制台，进入全局概览页面。
2. 在 BOS 控制台左侧的导航栏中选择扫描文件所在的**目标 Bucket**，点击打开该 Bucket 的文件列表页。
3. 点击目标文件右侧操作栏的**更多->软链接**即可对文件设置软链接。



4.在弹窗中对软链接文件进行命名。若您需要将软链接文件放入 Bucket 中某一个文件夹，请在命名时将文件夹路径一起写入软链接文件名称中。若您在名称中未标明路径，默认统一放入 Bucket 根目录中。如软链接文件名称为 aaa/ruanlianjie1.csv，那么该软链接文件将放入 aaa 文件夹进行保存；若您将其命名为 ruanlianjie.csv，那么该文件将放入 Bucket 根目录进行保存。



> **说明：**
>
> - 若您不希望保存已创建的软链接文件，您可以将其删除。软链接文件删除后，源文件不受影响；
> - 若您对软链接文件重命名，或者转换存储类型，都不会影响源文件的相关属性；
> - 若您将源文件删除，已创建的软链接文件不会随源文件同步删除，此时访问软链接文件将不会成功。若您将源文件删除，请手动同步删除软链接文件；
> - 软链接文件 API 操作方法同普通文件的操作方法相同，如删除，重命名操作等。

相关API

- PutSymlink 接口：使用 PutSymlink API 接口设置文件的软链接。
- GetSymlink 接口：使用 GetSymlink API 接口读取和访问软链接文件。

批量删除文件夹

概述

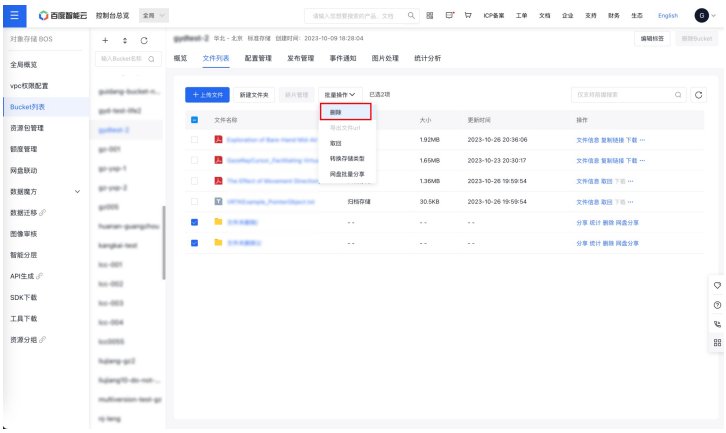
您可以在控制台批量删除文件夹。

注意事项

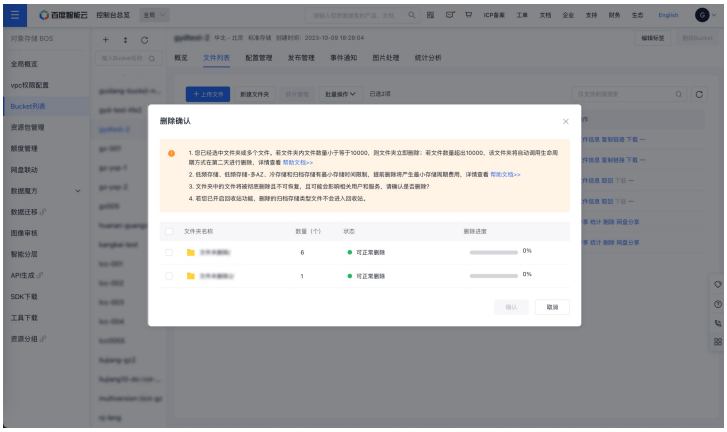
- 若文件夹内文件数量小于等于10000，则文件夹立即删除；若文件数量超出10000，该文件夹将自动调用生命周期方式在第二天进行删除；
- 低频存储、低频存储-多AZ、冷存储和归档存储有最小存储时间限制，提前删除将产生最小存储周期费用，详情查看计费项文档；
- 若您已开启回收站功能，删除的归档存储类型文件不会进入回收站；
- 若选择嵌套文件夹，请仔细检查嵌套文件夹内的文件，避免误删；
- 文件夹中的文件将被彻底删除且不可恢复，且可能会影响相关用户和服务，请在删除前仔细确认。

批量删除文件夹

1. 登录 BOS 管理控制台，进入全局概览页面。
2. 在 BOS 控制台左侧的导航栏中选择需要批量删除的文件夹所在的**目标 Bucket**，点击打开该 Bucket 的文件列表页。
3. 点击需要批量删除的文件夹前的复选按钮，然后点击页面上方【批量操作】中的【删除】。



4. 点击删除后，页面页面会出现弹窗确认，请仔细阅读相关注意事项。



5. 删除完成。

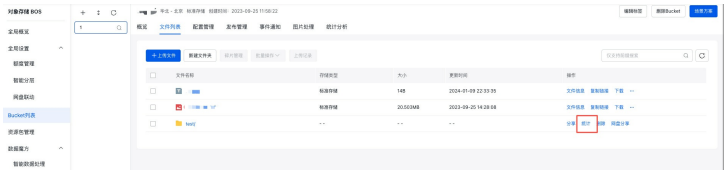
文件夹统计

概述

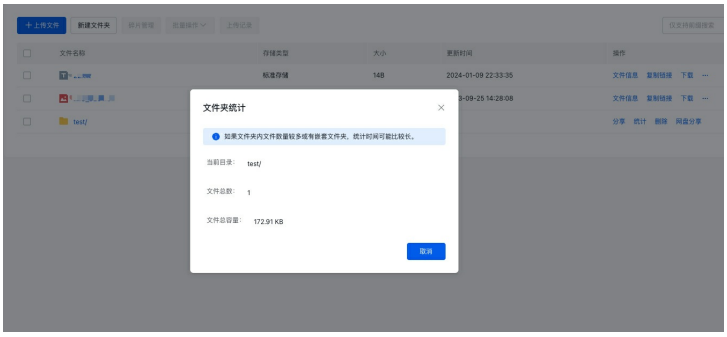
您可以在控制台统计 Bucket 中某一前缀下全部文件数量和总容量。

文件夹统计

1. 登录 BOS 管理控制台，进入全局概览页面。
2. 在 BOS 控制台左侧的导航栏中选择需要统计的文件夹所在的**目标 Bucket**，点击打开该 Bucket 的文件列表页。
3. 点击需要统计的文件夹后的"统计"按钮，对文件夹内文件的数量和容量进行统计。



4. 点击统计后，页面页面会出现统计弹窗，统计当前文件夹内的文件总数及总容量。



5. 统计完成。

分享文件-获取文件url

文件（Object）上传至存储空间（Bucket）后，您可以通过文件 URL 将文件分享给第三方预览或下载。

方式一：使用BOS控制台

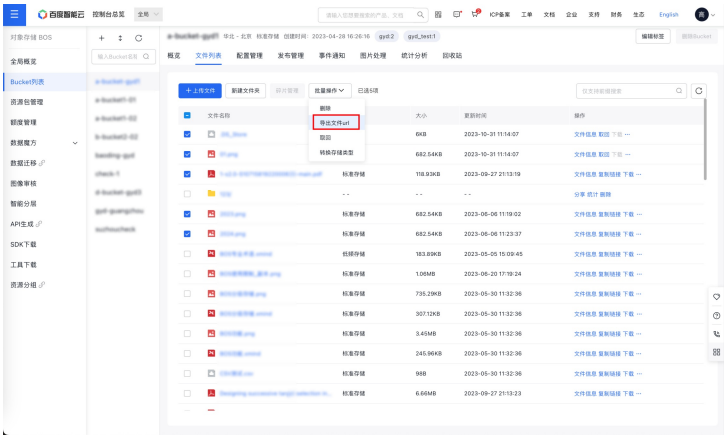
操作步骤

1. 登录 BOS 管理控制台。
2. 在 BOS 控制台左侧的导航栏中选择目标文件所在的 Bucket，点击进入该 Bucket 的文件列表页。
3. 获取文件 URL
   - **获取单个文件 URL**
     - 单击目标文件名称。
     - 在文件信息弹框中配置以下参数，然后单击【复制链接】。

参数	说明
链接有效时间	当目标文件为私有文件时，需设置文件URL的有效时间。取值范围：1-43200分钟，-1 为永久有效
选择域名	默认为官网域名，仅当Bucket绑定自定义域名后可配置此项。更多信息，请参见创建自定义域名。



- **批量获取文件 URL**
  - 在文件列表页中勾选多个目标文件，选择上方【批量操作】中【导出url】。



  - 在导出文件 URL 列表面板配置以下参数

参数	说明
指定协议	默认使用HTTPS协议生成文件URL。如需使用HTTP协议生成文件URL，请选择使用HTTP。
到期时间	当目标文件为私有文件时，需设置文件URL的有效时间。取值范围：1-1440分钟。若您希望获取更长时效的文件URL，建议使用 命令行工具。
自有域名	默认为官网域名，仅当Bucket绑定自定义域名后可配置此项。更多信息，请参见 创建自定义域名。
CDN加速域名	仅当通过BOS为Bucket开启CDN官方加速域名可配置此项。更多信息，请参见 开通CDN官方加速域名。

  - 单击确定，浏览器将自动生成格式为 CSV 的 URL 列表文件，您可保存至本地。

4. 将文件 URL 分享给第三方预览或下载。

方式二：使用图形化桌面工具BOS Browser

- BOS Browser支持分享单个文件，具体操作请参见 BOS桌面工具使用指南。

方式三：使用SDK

- 您可通过对象存储BOS包含的主流语言SDK来获取文件URL，从而分享给第三方预览或下载。

语言	相关文档
Java	Java-SDK 获取文件URL
Python	Python-SDK 获取文件URL
PHP	PHP-SDK 获取文件URL
C#	C#-SDK 获取文件URL
Android	Android-SDK 获取文件URL
iOS	iOS-SDK 获取文件URL
Ruby	Ruby-SDK 获取文件URL
GO	GO-SDK 获取文件URL
C++	C++ SDK 获取文件URL

方式四：使用命令行工具BOS CMD/CLI

- 您可通过BOS CMD/CLI获取文件URL，从而分享给第三方预览或下载。关于使用BOS CMD/CLI的具体操作，请参见通过CMD使用BOS服务、通过CLI使用BOS服务。

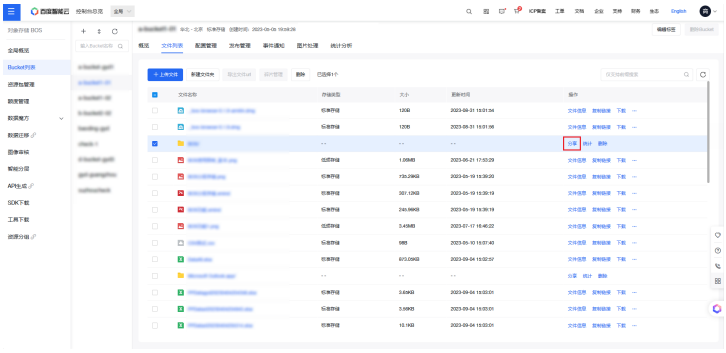
文件夹分享

当您需要批量分享某一个存储桶（Bucket）中的多个文件（Object）时，可以创建文件夹并将所有待分享的文件存储在文件夹中，通过文件夹分享功能生成分享链接。接收方通过浏览器访问分享链接或使用BOS桌面工具登录，即可下载文件夹中的内容。

创建文件夹分享链接

方式一：使用BOS控制台 操作步骤

1. 登录 BOS 管理控制台。
2. 在 BOS 控制台左侧的导航栏中选择目标文件夹所在的 Bucket，点击进入该 Bucket 的文件列表页。
3. 找到需要分享的文件夹，在其右侧操作栏下，单击【分享】。



4. 在弹出的窗口中，根据您的需求，设置文件夹分享链接的有效时间以及提取码，单击【创建分享链接】。

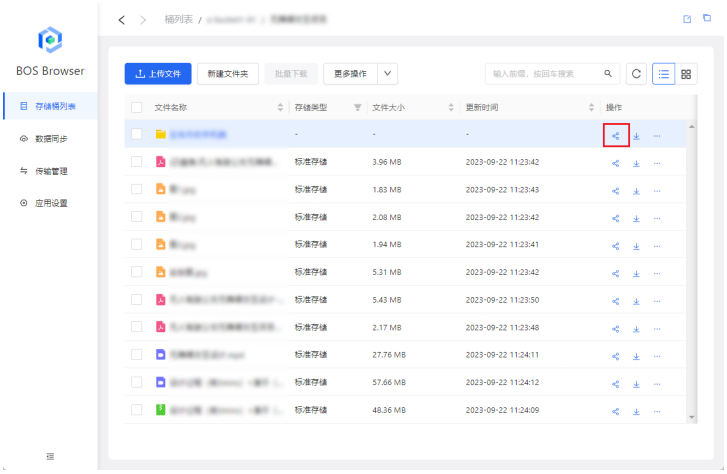


5. 在当前窗口中，即可看到已生成的文件夹URL，您可选择【复制全部】或【复制链接】，快速复制分享链接和提取码并分享给其他人。



方式二：使用图形化桌面工具BOS Browser 操作步骤

1. 使用AK/SK登录BOS桌面工具。若您还未下载BOS桌面工具，可前往 BOS-PC客户端 下载安装最新版。
2. 选择目标 Bucket。
3. 在文件列表页找到待分享的文件夹，点击操作列中【分享】图标。



4. 在文件夹分享弹窗中设置 URL 有效期与6位提取码，点击【生成链接】。



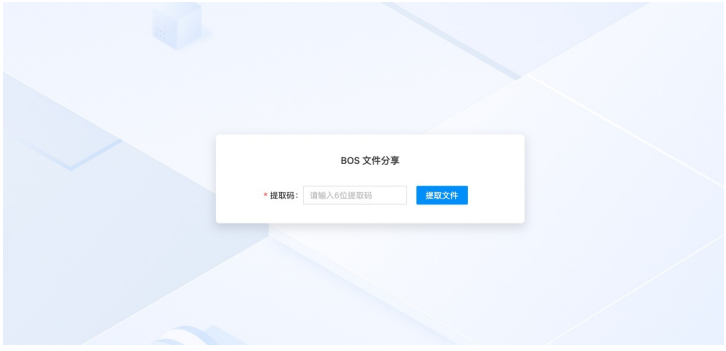
5. 复制文件夹分享链接与提取码。

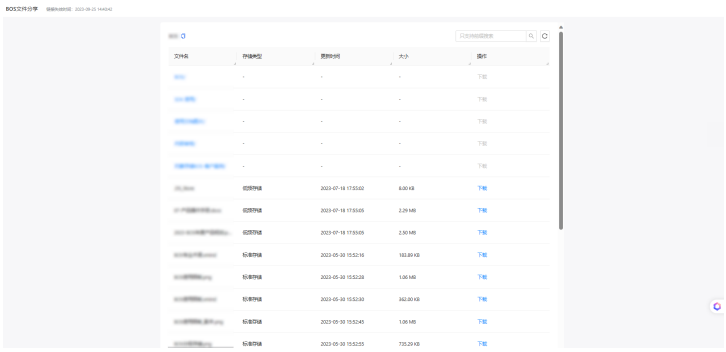


使用文件夹分享链接

当您获取 BOS 文件夹分享链接及提取码后，您可使用分享链接及提取码查看并下载分享链接中的文件/文件夹。 **方式一：使用浏览器打开 操作步骤**

1. 在链接有效期内使用浏览器打开文件夹分享链接，输入提取码后，点击【提取文件】即可查看分享文件夹中的文件列表。





2. 您可直接在文件列表中点击【下载】按钮将文件下载至本地。

> 注意：使用浏览器打开文件夹分享链接暂只支持下载当前文件夹中的文件，若您需要批量下载文件或文件夹，可参考方式二：使用图形化桌面工具BOS Browser。

方式二：使用图形化桌面工具BOS Browser打开 操作步骤

1. 在获取文件夹分享链接及提取码后，使用提取码登录BOS桌面工具。若您还未下载BOS桌面工具，可前往 BOS-PC客户端 下载安装最新版。



2. 您可点击文件列表操作栏中的【下载】按钮下载单个文件/文件夹。
3. 也可勾选需要批量下载的文件或文件夹，点击页面上方【批量下载】，创建批量下载任务。
4. 在 BOS 桌面工具左侧的导航栏中选择【任务管理】，可查看任务状态。
5. 详情可参见 使用BOS桌面工具下载分享文件。

> 注意：请在链接到期前完成文件下载，链接到期后桌面工具将自动退出登录，进行中的下载任务将被强制停止。

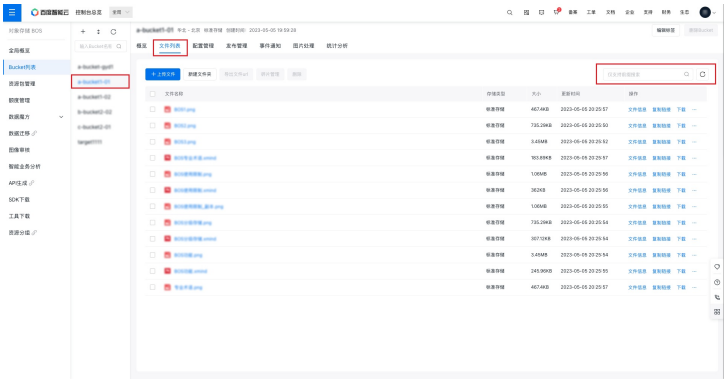
查询Object

概述

BOS中单个Bucket内支持存在无限个文件/文件夹，您可以通过控制台查看指定Bucket中已上传的全部Object

控制台查看全部Object

1. 登陆对象存储BOS管理控制台
2. 选择左侧Bucket列表中的某个Bucket，点击右侧上方导航栏中的**文件列表**，BOS会为您列举当前Bucket中的全部Object
3. 页面右上角有Object搜索框。搜索框仅支持前缀搜索，如输入“BOS”，则会出现所有以“BOS”开头命名的Object
4. 也可通过搜索框右侧的刷新按钮刷新Object列表



相关API

- ListObjects 接口：使用ListObjects API获取指定Bucket的Object信息列表

取回归档文件

概述

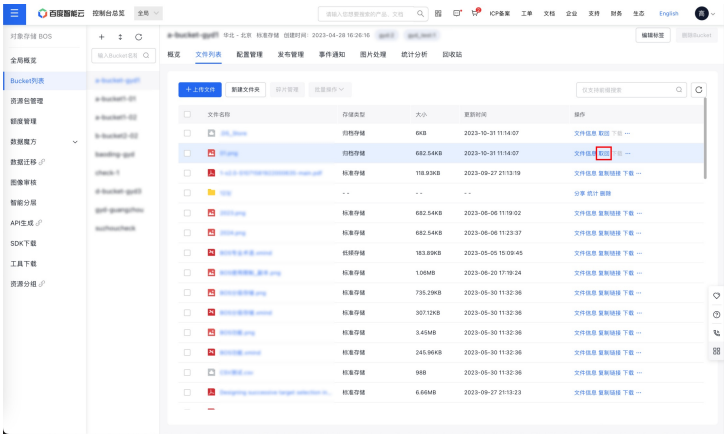
归档存储类型的Object，需要手动取回后才能读取。BOS 为用户提供加急取回、标准取回和延缓取回三种取回模式，可根据您的不同需求进行选择。

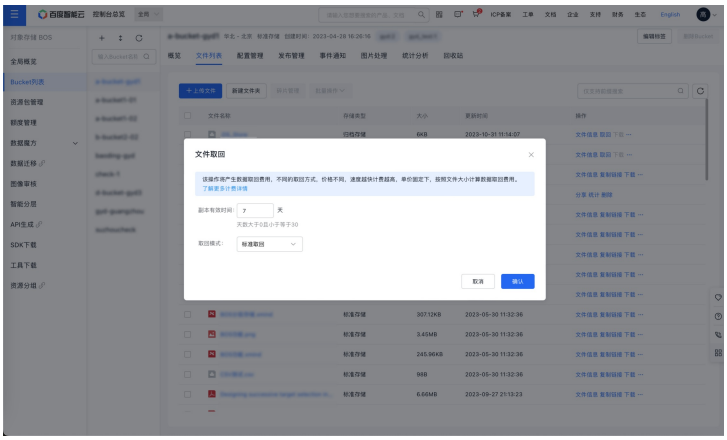
注意事项

- 仅归档类型文件支持取回，不同取回模式可参考 存储类型-归档存储-数据取回性能。
- 取回操作将产生数据取回费用，不同的取回模式价格不同，速度越快计费越高，单价固定下，按照文件大小计算数据取回费用，详情参见 计费详情。

取回单个文件

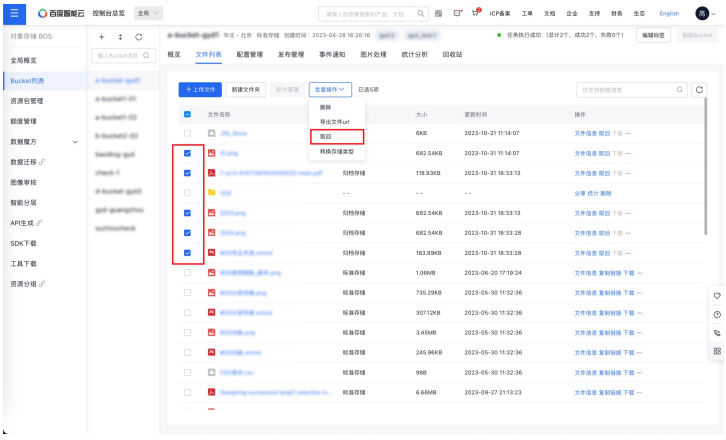
1. 登陆 BOS 控制台，进入概览页面。
2. 选择需要取回的归档文件所在Bucket，进入文件列表页面。
3. 在文件列表页面找到需要取回的归档文件，点击右侧操作列中【取回】按钮。
4. 在取回弹窗中设置取回副本有效时间、取回模式。
5. 点击确认，即可开始取回。

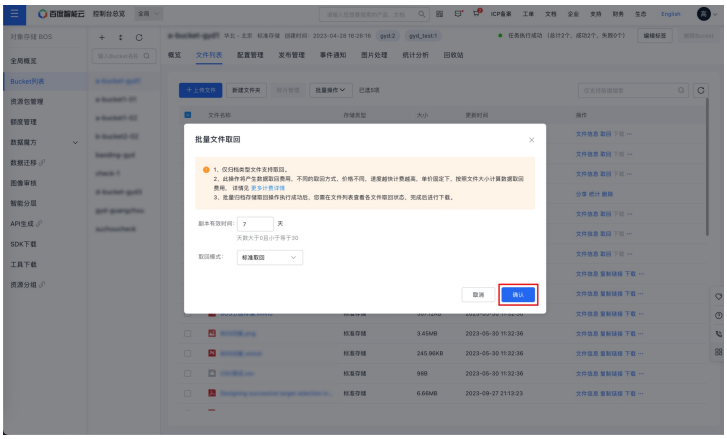


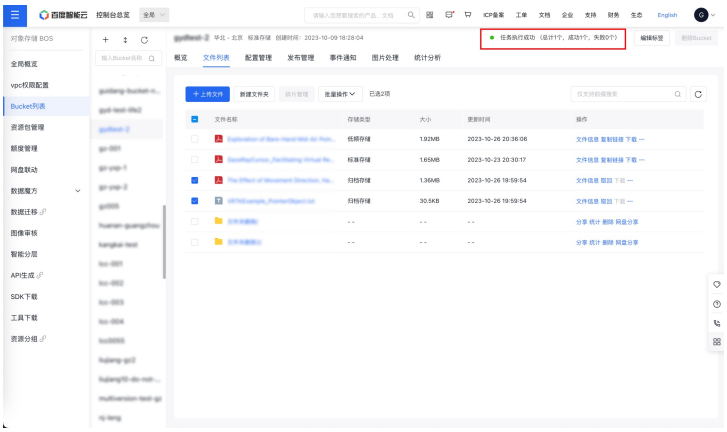


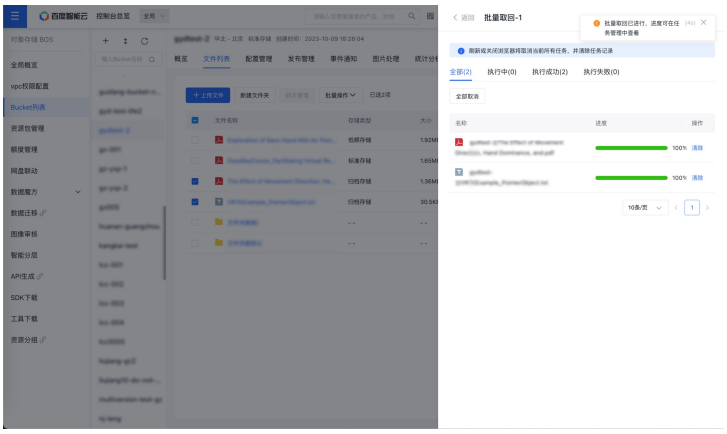
批量取回归档文件

1. 登陆 BOS 控制台，进入概览页面。
2. 选择需要取回的归档文件所在Bucket，进入文件列表页面。
3. 在文件列表页面找到需要取回的归档文件，勾选多个对象后点击上方【批量操作】中【取回】按钮。
4. 在取回弹窗中设置取回副本有效时间、取回模式。
5. 点击【确认】，即可创建批量取回任务。
6. 可在右上角任务状态中查看批量取回任务的进度和结果。









批量管理Object

BOS有哪些批量操作？

对象存储 BOS 提供丰富的访问和管理文件（Object）的方式，本文介绍如何批量管理 Object。

批量上传

您可以使用以下方法，批量上传文件：

方法	说明
BOS 控制台	使用 BOS 管理控制台，批量选中文件后上传到 BOS。详情参见上传文件。
BOS桌面工具	使用 BOS 提供的官方图形化管理工具 BOS Browser 批量上传 Object，支持选中或拖拽上传。详情参见文件/文件夹上传。
BOS CMD/CLI	使用 BOS CMD/CLI 可批量上传 Object，详情参见 BOS CMD-上传Object、BOS CLI-上传Object。

批量下载

您可以使用以下方法，批量下载文件：

方法	说明
BOS桌面工具	使用 BOS Browser 勾选多个文件或文件夹，将文件或文件夹批量下载到本地。详情参见文件/文件夹下载。
BOS CMD/CLI	使用 BOS CMD/CLI 可批量下载 Object，详情参见 BOS CMD-下载Object、BOS CLI-下载Object。

批量分享

您可以使用以下方法，批量分享文件：

方法	说明
BOS 控制台	批量导出文件URL： 使用 BOS 管理控制台，选中多个文件后选择【批量操作】中的【导出文件URL】即可生成包含勾选文件 URL 的 CSV 文件，可以通过文件 URL 将文件分享给第三方预览或下载。详情参见分享文件-获取文件URL。
	网盘批量分享（白名单功能）： 使用 BOS 管理控制台，选中多个文件后选择【批量操作】中的【网盘分享】即可将文件批量同步至网盘，进行相关设置后可生成网盘分享链接。详情参见联动百度网盘分享。

批量复制

您可以使用以下方法，批量复制文件：

方法	说明
数据同步	通过数据同步功能可以满足Bucket跨区域、跨账号或其他数据复制的需求。目前BOS同时支持使用控制台、API、SDK进行跨区域复制，详情参见数据同步。
BOS 桌面工具	使用 BOS Browser 勾选多个文件批量复制文件。详情参见文件复制。
BOS CMD/CLI	使用 BOS CMD/CLI 可实现批量文件拷贝。详情参见 BOS CMD-拷贝Object、BOS CLI-拷贝Object。

批量删除

> **注意：**文件删除后无法恢复，请谨慎操作。

您可以使用以下方法，批量删除文件：

方法	说明
BOS 控制台	使用 BOS 管理控制台，选中多个 Object 后选择【批量操作】中的【删除】即可删除多个 Object。详情参见删除文件。
	使用 BOS 控制台的碎片管理功能，可以批量删除碎片文件。详情参见碎片管理。
	使用生命周期管理可批量自动删除文件。详情参见 管理生命周期。
BOS桌面工具	使用 BOS Browser 勾选多个文件或文件夹批量删除。详情参见文件删除。
BOS API	通过 BOS 的 DeleteMultipleObjects 接口可批量删除文件。详情参见DeleteMultipleObjects。
BOS SDK	使用 BOS SDK 可批量删除文件。更多语言的 SDK 示例请参见SDK 概览。 • Java SDK • Python SDK • Go SDK • C++ SDK
BOS CMD/CLI	使用 BOS CMD/CLI 可批量删除 Object。详情参见 BOS CMD-删除Object、BOS CLI-删除Object。
BOS-Util 批量操作工具	BOS-Util 是以命令行方式管理 BOS 数据的工具，基于 Java 编写，提供对 BOS 端数据的批量操作。使用 BOS-Util 的 DELETE_OBJECTS 命令，可以批量删除 Object。详情参见 DELETE_OBJECTS命令。

批量转换文件存储类型

您可以使用以下方法，批量修改文件的存储类型：

方法	说明
BOS 控制台	使用 BOS 管理控制台，选中多个文件或文件夹后选择【批量操作】中的【转换存储类型】即可进行批量转换文件的存储类型。详情参见转换文件存储类型。

批量修改文件访问权限（ACL）

您可以使用以下方式批量修改文件ACL：

方法	说明
BOS-Util 批量操作工具	使用 BOS-Util 的 SET_OBJECTS_ACL 命令，可以批量设置 Object 的 ACL。详情参见 SET_OBJECTS_ACL。

批量取回归档文件

您可以使用以下方法，批量将归档存储类型的文件取回，取回后文件将由冰冻状态恢复为可读：

方法	说明
BOS 控制台	使用 BOS 管理控制台，选中多个归档类型的 Object 后选择【批量操作】中的【取回】即可批量取回。

批量设置文件Meta信息

您可以使用以下方法，批量修改文件的Meta信息：

方法	说明
BOS-Util 批量操作工具	使用 BOS-Util 的 SET_OBJECTS_META 命令，可以批量修改 Object 的 meta 信息。详情参见 SET_OBJECTS_META命令。

对象标签

概述

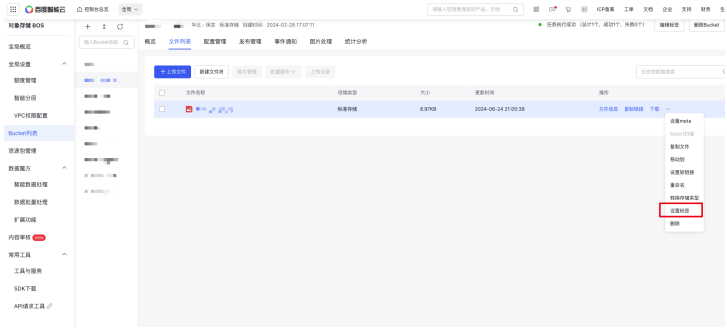
对象标签是键值对形式的对象标识，数据同步、批量操作等功能支持通过指定对象标签的方式选择生效的资源范围，从而简化数据管理操作。

注意事项

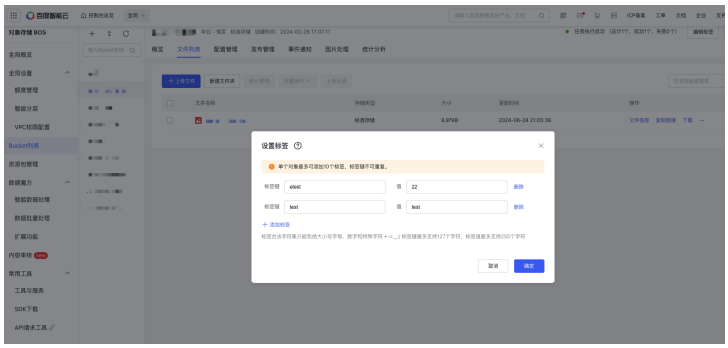
- 每个对象最多配置10个标签。
- 标签键不可重复，长度范围为1-127个字符，支持大小写字母、数字、空格和以下符号： +-=._:/。
- 标签值长度范围为1-125个字符，支持大小写字母、数字、空格和以下符号： +-=._:/。
- 归档类型对象未取回时不支持设置标签，请先取回后再设置。
- 需要确保拥有PutObjectTagging权限才可设置对象标签。

操作步骤

1. 登录 BOS 管理控制台，进入全局概览页面。
2. 在 BOS 控制台左侧的导航栏中选择对象所在的**目标 Bucket**，点击打开该 Bucket 的文件列表页。
3. 点击目标文件右侧操作栏的**更多->设置标签**，添加新的标签键和标签值，或对已有的标签进行修改，点击**确定**提交标签修改。



4. 您可以再次点击**设置标签**查看、编辑或删除已有标签。



数据发布

数据发布简介

概述

BOS 支持为已创建的 Bucket 设置多种域名，从而对存储的数据进行发布。

域名类型及使用场景

BOS 当前支持 Bucket 官方域名、自定义域名、CDN 官方域名。不同的域名对应不同的使用场景，您可以根据自身需求对域名进行配置和使用。

域名类型	使用场景
Bucket官方域名	• 当您创建 Bucket 时候，BOS 会为该 Bucket 自动生成 Bucket 官方域名。 • 关于不同区域的域名 EndPoint，请参考 获取访问域名。
自定义域名	• BOS 支持为 Bucket 添加自定义域名，将该自定义域名与 Bucket 绑定后，即可利用该自定义域名将 Bucket 中的数据进行发布。您可使用自定义域名直接访问绑定的 Bucket。 • 添加的自定义域名需要到域名注册商 DNS 解析管理中，完成自定义域名 CNAME 解析，解析生效后才可以通过自定义域名访问。
CDN 官方加速域名	• BOS 目前支持 Bucket 绑定 CDN 加速域名。绑定后，可利用百度智能云内容分发网络 CDN 遍布全球的边缘节点，对 BOS 中内容进行下行加速分发。在您为 Bukcet 绑定 CDN 域名后，该 Bucket 域名的 DNS 解析将自动修改为 CDN 提供的 CNAME 域名，从而接入 CDN 节点，开启 CDN 加速。CDN 官方加速适用于边缘场景中文件的下行加速。 • 百度智能云 BOS 与 CDN 提供了缓存管理能力，当您同时开通百度智能云 BOS 及 CDN，并将 BOS 作为 CDN 服务的源站时，可进行智能缓存管理操作。 • BOS 支持 CDN 上行加速功能，满足用户在使用 BOS 时对上行场景中的加速需求。上行加速功能需要单独开通，且该上行域名与 CDN 官方加速域名相同。

查看Bucket官方域名

概述

当您创建一个 Bucket 之后，BOS 会自动为其创建一个官方域名，且该官方域名与 Bucket 名称一一对应且唯一存在。

查看 Bucket 官方域名

1. 登录 BOS 管理控制台，进入全局概览页面。
2. 在 BOS 控制台左侧的导航栏中选择目标**Bucket**，点击打开该 Bucket 的文件列表页。
3. 在上方导航栏选择**发布管理**页签，进入发布管理页面。
4. 在**官方域名发布**模块下，可以查看Bucket**官方域名**。官方域名不支持修改，若您需要使用自定义，可创建自定义域名。



创建Bucket自定义域名

概述

BOS 支持为 Bucket 添加自定义域名，将该自定义域名与 Bucket 绑定后，即可利用该自定义域名将 Bucket 中的数据进行发布。您可使用自定义域名直接访问绑定的 Bucket。

注意事项

通过为 Bucket 创建自定义域名，可以满足以下需求场景：

- 通过浏览器访问 Bucket 内的网页类型文件（mimetype为text/html，扩展名包括HTM、HTML）时，需要以预览的方式打开文件，而非下载文件。
- 将 Bucket 配置成静态网站时，需要使用浏览器直接访问静态网站页面，而非下载静态网页文件。

前提条件

- 已完成域名备案。若域名没有备案，请按照备案流程进行备案。

- 已完成域名所有权验证。若没有验证，请按照验证域名归属流程进行验证。
- 已创建 Bucket。创建 Bucket 步骤请参见创建 Bucket。
- 如需开启域名加速，需要先开通 CDN/DRCDN服务。

验证域名归属

1. 在BOS控制台发布管理-自定义域名发布页面，点击添加自定义域名，输入自定义域名地址后，BOS会随机生成域名对应的token并展示在表格中，包括记录类型、主机记录、记录值等，您需要使用这些信息完成TXT域名解析和所有权验证。当输入域名为二级域名时表格中会显示两条信息，上下两条记录中任选一条并完成解析和验证即可。



2. 登录您域名提供商的网站，登录域名管理界面，添加域名，输入BOS控制台表格中的记录类型、主机记录、记录值进行TXT域名解析。若您的域名解析商为百度智能云，详见DNS解析验证操作步骤。
3. TXT解析生效后，在BOS控制台点击域名对应的验证按钮，即可完成域名所有权验证。

域名	记录类型	主机记录	记录值	操作
www.[redacted].com	TXT	bdy-verify.www....	18c67ec3-cfffa2...	验证
[redacted].com	TXT	bdy-verify.[redacted].c...	18c67ec3-f8e68...	验证

注意事项

- 自定义域名中不得包括协议、端口和路径，而应只包含域名本身。
- 每个 Bucket 最多可添加并绑定 200 个不同的自定义域名。
- 一个域名只能绑定在单个bucket上。

创建自定义域名

1. 登录 BOS 管理控制台，进入全局概览页面。
2. 在 BOS 控制台左侧的导航栏的Bucket列表中选择目标 **Bucket名称**，点击打开该 Bucket 的文件列表页。
3. 点击面上方导航栏的**发布管理**页签，进入发布管理页面。
4. 点击“自定义域名发布”页面的**添加自定义域名**。



5. 在弹出的添加自定义域名窗口填写已备案且已完成域名归属验证的**域名地址**，选择是否**开通 CDN 加速**，是否开启 **HTTPS 配置**。



6. 点击**确认**，完成 Bucket 自定义域名的添加和绑定；添加完成后会在下方的自定义域名列表中显示。

> **注意：**
>
> - **添加的自定义域名需要到域名注册商 DNS 解析管理中，完成自定义域名 CNAME 解析，解析生效后才可以通过自定义域名访问**。
> - 如果需要对自定义域名开通 CDN 加速，点击域名对应操作列的**开通 CDN 加速**。
> - 如果希望解除自定义域名的绑定，点击域名对应操作列的**删除**。

CNAME域名解析

完成绑定之后，请登录域名服务商后台，在域名管理中添加 CNAME 域名解析，将需要的域名 CNAME 到 BOS 的域名,BOS域名获取方式：

1.通过控制台： 同样在**发布管理**页签下，将【官方域名】去掉存储桶名称与.，保留二级域名即可。

2.规则拼接
`<region>.bcebos.com`

> 说明：(`<region>`为Bucket所在的区域，请参考[区域选择说明](Reference/区域选择说明/区域.md)，即可完成全部域名绑定配置。例如：
> 您的 Bucket 所在区域为北京，请 CNAME 到`bj.bcebos.com`。
> 您的 Bucekt 所在区域为广州，请 CNAME 到`gz.bcebos.com`。
> 您的 Bucket 所在区域在苏州，请 CNAME 到`su.bcebos.com`。

> **注意：**如绑定的域名原有其他解析记录，CNAME 到`<region>.bcebos.com`后，需要删除原解析记录。

开启域名加速

您可以在添加自定义域名或添加后通过编辑为自定义域名开启域名加速，支持的 静态加速CDN 以及 动态加速 DRCDN两种加速类型。

域名加速开通 CDN/DRCDN服务默认只对国内的域名访问有加速效果，如您有全球访问加速的需求，可在开启域名加速时同时开启 海外CDN加速。开启海外加速可对全球范围内上行和下行的访问进行加速，显著提升访问效率。

> **注意：**当前海外加速功能为白名单功能，如需开通请提交工单。
>
> - 自定义域名选择开启 CDN/DRCDN 其中一种服务后关闭域名，重新开启加速时无法重新选择服务类型。如需更改服务类型请前往 CDN/DRCDN控制台进行服务变更。
> - 海外加速为域名加速的子功能，关闭域名加速时海外加速也将自动关闭。
> - CDN/DRCDN/海外加速功能将产生流量费用，费用由CDN服务收取，您可在CDN相关账单中查看相关费用明细。

绑定https证书

当您做完 CNAME 解析之后，如果您希望以 https 方式访问此自定义域名，可将您持有的 https 证书绑定至此自定义域名。

百度智能云SSL证书服务提供了完备的证书申请与管理一站式服务，用户可在百度智能云上购买、管理、使用https证书，详情可参见证书管理

您在百度智能云 SSL 证书服务中持有可用证书后，在 BOS 自定义域名控制台中，可在首次创建自定义域名时或对已有自定义域名进行修改时，点击“HTTPS 证书”功能项，下拉展示出您在百度智能云SSL证书服务中管理的所有证书，并按需选择对应证书。

当域名已绑定的证书过期或失效时，您同样可以选中该自定义域名，点击“HTTPS 证书”功能项，在下拉菜单中选择可用的证书进行替换。

> **注意：**
>
> - 当您选择的证书所覆盖的域名范围，与您预期绑定的自定义域名不匹配时，界面会给予相应提示，您需要更换其他可用的证书。
> - 建议您在百度智能云SSL证书服务中，注意查看绑定证书的生效时间，避免证书到期后 https 访问不可用。
> - 当前BOS暂时支持为华北-北京（bj）、华北-保定（db）、华东-苏州（su）、华南-广州（gz） 四个 region 下 bucket 的自定义域名配置 https 证书。
> - 选择绑定某自定义证书后，绑定关系的建立通常需要 5-10 分钟，请您耐心等待后尝试使用。

访问地址

完成绑定之后，可使用绑定的域名直接访问 Bucket。

例如：给 Bucket：video绑定的域名为video.51mingxi.com，请在域名服务商后台将域名video.51mingxi.com CNAME 为<region>.bcebos.com。完成后，使用 URL：http://video.51mingxi.com/也可直接访问此 Bucket。

开通CDN官方加速域名

概述

BOS 目前支持为bucket 官方域名开通 CDN/DRCDN 加速服务。开通后，可利用百度智能云内容分发网络 CDN/DRCDN 遍布全球的边缘节点，对 BOS 中内容进行下行加速分发。在您为 Bukcet 开通域名加速后，该 Bucket 域名的 DNS 解析将自动修改为 CDN 提供的 CNAME 域名，从而接入 CDN 节点，开启 CDN 加速。

注意事项

- Bucket 创建时默认不开启域名加速，如需开启该功能请先前往 CDN/DRCDN 控制台开通服务。
- BOS 提供了官方加速域名，即 bucketname.cdn.bcebos.com。此时，您不需要配置 CNAME 解析，该域名不可修改。
- 若没有在 CDN 中配置过期时间，且源站没有显示提供 http 缓存控制头，CDN 不缓存数据。在此情况下，Bukcet 内的数据更新之后无法实现 CDN 加速，需在 CDN 中**手动刷新缓存**，具体操作请参考 刷新缓存。
- 可在 CDN 中配置过期时间，用以自动更新缓存，具体操作请参考 配置缓存过期时间。

开通官方加速域名

1. 登录 BOS 管理控制台，进入全局概览页面。
2. 在 BOS 控制台左侧的导航栏中选择**目标 Bucket**，点击打开该 Bucket 的文件列表页。
3. 展开左侧桶配置中导航栏的 **发布管理** 类别，进入发布管理页面。
4. 点击 官方域名发布 模块下的 **域名加速** 开关，点击即可开始配置官方域名加速。
5. 根据需求选择开通的加速类型，如仅需静态数据加速可选择 CDN服务，如需对静态和动态数据均加速可选择 DRCDN服务,详细信息请参考对应的服务文档。
6. 开启加速成功后，刷新页面可以看到 CDN 加速下的加速域名后出现 **管理加速域名** 按钮，点击可以进入 CDN 的域名管理页面对域名详情进行管理。



配置CDN上行加速

概述

BOS 全新发布 DRCDN 动态加速功能，满足用户在使用 BOS 时对上行场景中的加速需求。如：

- 文件上传加速场景
- 远距离的文件上传场景
- 大文件上传场景
- 动态更新文件上传场景

您也可以查看 DRCDN 上行加速典型实践，以了解更多使用信息。

前提条件

- 账号已开通 DRCDN服务；

当您开通完成后，可以在BOS控制台开启域名的动态加速功能。域名加速开启后，会产生 DRCDN 动态加速费用。

开通 DRCDN 上行加速

1. 登录 BOS 管理控制台，进入全局概览页面。
2. 在 BOS 控制台左侧的导航栏中选择目标 **Bucket**，点击打开该 Bucket 的文件列表页。
3. 点击页面上方导航栏的**发布管理**页签，进入发布管理页面。
4. 您可以为官网域名或者自定义域名开启动态加速。如您需要开启官网域名加速，在**官网域名发布**模块中域名加速部分点击**开启域名加速**，完成选择后点击**确认**即可开通 DRCDN 上行加速。开启后，您可前往 DRCDN控制台对加速规则进行详细配置。



5.自定义域名开启动态加速域官方域名开启动态加速的流程类似，您可以在添加自定义域名或对已经添加的自定义域名进行编辑时开启 DRCDN动态加速。



> **说明：** Bucket 动态加速功能的 DRCDN 域名同该 Bucket 的 CDN 官方域名相同。

6. **关闭上行加速**:当不再需要 CDN 上行加速时，您可以在控制台将该功能关闭。

配置缓存管理

概述

当您开通 CDN 服务时，时常会有如下业务需求：

- 源站中新增了数据，CDN 缓存节点需要第一时间刷新，但此过程希望自动化；
- 源站中产生了文件更改，CDN 缓存节点需要将缓存的旧数据更新，便于用户访问正确的业务信息；
- 源经过审核，源站发现了不合规文件，并已被管理员清除，CDN 缓存节点也需要第一时间清除该缓存。

基于上述场景，百度智能云 BOS 与 CDN 提供了缓存管理能力，当您同时开通百度智能云 BOS 及 CDN，并将 BOS 作为 CDN 服务的源站时，可进行智能缓存管理操作。

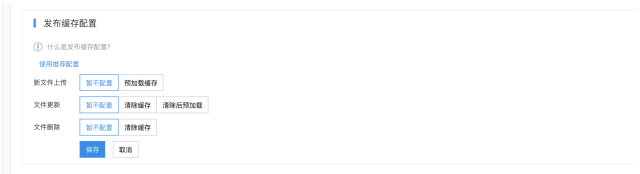
使用CDN缓存管理时有如下注意事项：

- CDN缓存自动刷新管理功能不保证一定能成功提交刷新任务，也不保证刷新任务提交的及时性。
- CDN缓存自动刷新功能仅支持少量文件的更新提交刷新任务。如果有大量文件的更新操作，可能会触发流控丢弃部分刷新任务。

> 注意 对页面或者有大量文件更新需求的用户，建议直接调用CDN刷新缓存，刷新节点上的文件内容,您也可以调用CDN查询限额，查询用户当前的限额(Quota)值。

配置缓存管理

1. 登录 BOS 管理控制台，进入全局概览页面。
2. 在 BOS 控制台左侧的导航栏中选择目标 **Bucket**，点击打开该 Bucket 的文件列表页。
3. 点击页面上方导航栏的**发布管理**页签，进入发布管理页面。
4. 在“发布缓存配置”模块点击**编辑**按钮，配置缓存管理。



5. 对于**新文件上传**，**文件更新**和**文件删除**三种源站中发生的事件，定义 CDN 节点的联动逻辑。

场景	说明
新文件上传	• 预加载缓存：在源站产生新文件上传时，自动预热到各节点。
文件更新	• 清除缓存：在源站文件产生更新后，CDN 节点自动清除原文件对应的缓存。 • 清除后预加载：CDN 节点会在清除原缓存后生成新的缓存。
文件删除	• 清除缓存：在源站产生文件删除时，CDN 节点同步清除对应缓存。



6. 配置缓存节点联动逻辑后，还可指定生效资源。目前支持对整个 Bucket 或对 Bucket 内的部分资源生效。**每条资源以换行符分隔，最多添加 20 条。**



7. 点击**保存**，页面右上角弹出"配置成功"字样，发布缓存配置模块信息也随之更新，缓存管理配置成功。

> **注意：**开启缓存管理后，BOS 会自动按照配置与 CDN 联动执行回源，此时产生的回源流量会产生相应费用，价格标准请参考 BOS 产品定价。

全球加速

概述

不同地域对对象存储BOS中文件访问质量受链路影响较大，当您的业务有海外访问国内存储桶需求时，可通过对象存储BOS提供的全球加速能力提升全球范围访问质量。BOS利用CDN服务遍布全球的边缘节点，客户端就近接入最近的节点并利用节点间高效的路由以及优质的链路，从而优化全球范围内的访问体验。

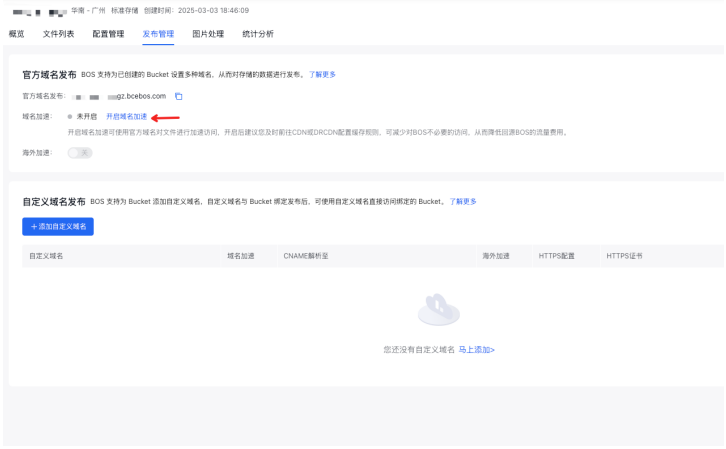
注意事项

> - 开通 CDN/DRCDN 域名加速默认仅对国内范围有访问加速作用，如需全球加速还需打开 海外加速 开关。

- 开通海外加速功能需保证已开通 海外CDN 服务，如需开通请提交工单。
- 海外加速为域名加速的子功能，如您关闭域名加速，海外加速也会同时关闭，请谨慎操作。

官方域名开通全球加速

1. 登录 BOS 管理控制台。
2. 在 BOS 控制台左侧的导航栏中选择**Bucket列表**，点击选择目标bucket进入bucket详情页。
3. 点击页面左侧的配置导航栏中**发布管理**页签，进入发布管理页面。
4. 点击"官方域名发布"模块下的 **开启域名加速** ，在弹窗中选择国内加速的服务类型，详细可参考内容分发网络CDN和动态加速DRCDN产品文档查看服务详情。



5. 国内域名加速开启后，请确保已经开通了海外CDN服务 。点击下方的 **海外加速** 开关 ，即可开启全球加速功能。



自定义域名开通全球加速

1. 登录 BOS 管理控制台。
2. 在 BOS 控制台左侧的导航栏中选择**Bucket列表**，点击选择目标bucket进入bucket详情页。
3. 点击页面左侧的配置导航栏中**发布管理**页签，进入发布管理页面。
4. 您可以在添加自定义域名或对已添加自定义域名进行编辑操作时打开全球加速。现打开 **域名加速** 选项，根据需求选择国内加速的服务类型，详细可参考内容分发网络CDN和动态加速DRCDN产品文档查看服务详情。然后点击 **海外加速** 开关，保存设置后即可开启自定义域名的全球加速。



更多操作

您可独立对 海外加速 功能进行启停，但关闭域名加速功能时海外加速也会被同时关闭。

计费说明

开启全球加速功能将产生流量费用，该费用由CDN服务收取，计费详情请根据开通的服务类型查看对应服务的计费说明内容分发网络CDN计费、动态加速DRCDN计费以及海外加速计费。

注意

> - 海外加速的上行加速功能当前处于公测中，上行流量在公测阶段免费，下行流量计费详见海外加速计费。
> - 通过BOS控制台开通的域名加速功能默认使用按流量计费模式。
> - 开通全球加速功能后，国内访问加速收取 CDN/DRCDN 流量费用，海外访问加速收取 海外CDN 流量费用，两者不会同时收取。

常见问题

原来已开通国内域名加速，再开通海外加速会对原有国内加速有影响吗？ 不会，开启海外加速仅增加了边缘节点的覆盖范围从而实现全球加速的能力，不会对国内加速服务产生影响。

开启全球加速后，海外访问会产生国内CDN/DRCDN流量费用吗？ 不会，国内访问只产生国内CDN/DRCDN费用，海外访问只产生海外CDN费用，不会重复收取。

查看并导出用量数据

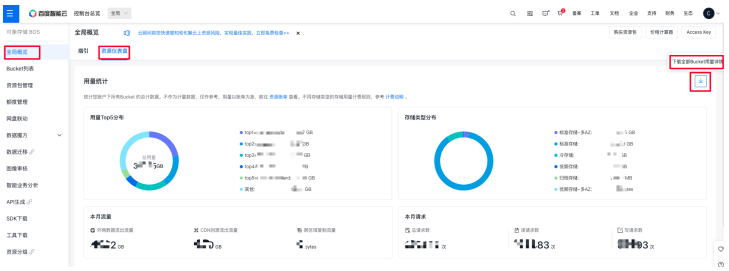
概述

BOS 控制台支持查看并导出您账号下全部 Bucket 当月的统计信息，包括每个 Bucket 的总容量，总流量，总请求次数，以及每个 Bucket 的细粒度统计数据，如标准存储，低频存储等类型文件的容量。

查看并导出用量数据

1. 登录 对象存储 BOS 管理控制台。

2. 在 BOS 控制台左侧的导航栏中找到并点击 **全局概览**，进入全局概览页面，点击 "资源仪表盘" 进入资源仪表盘页面，点击下载按钮"下载全部bucket用量详情"。



3.页面中展示按照 Bucket 维度划分的数据，以饼状图形式直观展示当前各 Bucket 内的存储总容量，并汇总展示您账号下本月已使用的总流量和总请求次数。

饼状容量图展示容量占比最大的 5 个 Bucket 的数据，若您想获取账号下全部 Bucket 的详细数据信息，请点击页面中 "**下载按钮>**"，系统会立即为您生成并下载 CSV 格式数据，您可以在该文件中查看全部用量信息。

> **注意：**
>
> - 若您账号下 Bucket 总数超过 100 个，控制台不会展示当前汇总数据，仅支持下载后查看；
> - 若您账号下 Bucket 数量较多，CSV 文件的生成和下载时间可能花费几分钟，请耐心等待。

导出数据指标

当您点击"立即下载"后，浏览器会自动下载并生成一个名为 **"bosMetricData.csv"** 的文件，该文件包含您账号下每个 Bucket 的总容量，总流量，总请求次数，以及每个 Bucket 的细粒度统计数据，如标准存储，低频存储等类型文件的容量和请求次数等。文件所包含的数据指标如下：

数据指标	指标含义
bucketName	• Bucket 名称
	每个 Bucket 的详细用量指标，包括： • **totalSpaceUsed**：全部存储类型文件的总容量 • **BucketSpaceUsedBytes**：标准存储类型文件总容量 • **BucketStandardIAStorageBytes**：低频存储类型文件总容量 • **BucketColdSpaceUsedBytes**：冷存储类型文件总容量 • **BucketArchiveSpaceUsedBytes**：归档存储类型文件总容量 • **totalBytesUsed**：本月请求总流量

metricName	• **ReadBytes**：外网数据流出流量 • **CDN1ReadBytes**：CDN 回源流出流量 • **XRegionCopyBytes**：跨区域复制流量 • **totalOpCount**：总请求数 • **ReadOpCount**：标准存储文件读请求总数 • **WriteOpCount**：标准存储文件写清求总数 • **StandardIAReadOpCount**：低频存储文件读请求总数 • **StandardIAWriteOpCount**：低频存储文件写请求总数 • **ColdReadOpCount**：冷存储文件读请求总数 • **ColdWriteOpCount**：冷存储文件写请求总数 • **ArchiveReadOpCount**：归档存储文件读请求总数 • **ArchiveWriteOpCount**：归档存储文件写请求总数
value	每项指标的用量数据，数据单位为： • 存储容量数据的 value 单位为 Bytes • 流量数据指标的 value 单位为 Bytes • 请求次数指标的 value 单位为次
startTime	每项数据的统计起始时间： • 容量数据展示近 20 分钟时间段内数据 • 流量和请求数据的起始时间从本月（自然月）第一日零点起
endTime	每项数据的统计结束时间： • 均为文件发起下载的时刻

> **注意：**
>
> - CSV 文件中 startTime 和 endTime 均为格林尼治时间，较北京时间晚 8 小时；
> - 若您希望查看 BOS 各计费项价格，请查看按需计费。

事件通知及数据处理

概述

BOS 事件通知功能，是基于 BOS 数据处理框架 V2.0 生成的一套智能触发平台。开发者可基于平台提供的功能，定义业务关注的 Bucket 中发生的数据行为，BOS 会监测 Bucket 中发生的 Put Object、Append Object

等行为，向用户的下游业务系统提供触发通知。

同时，BOS 结合了百度智能云上强大的 AI 能力，支持基于监测到的用户行为，直接向百度智能云上的各项数据处理能力进行触发，并将处理后的结果直接给到用户下游业务系统。简化用户的开发逻辑，为业务提速的同时也赋予用户更强的数据AI处理能力。

应用场景

BOS 事件通知平台，为广大开发者提供了一个自动化的数据行为监测及通知系统。能够满足类似如下应用场景：

- 某 Bucket 中有新的数据上传后，需要立刻告知上层应用，上层应用立刻将数据拉取使用。
- 在 Bucket 中有新的图片上传后，需要立刻进行内容加工，生成的新图片用于前端展示。而原始图片需要立刻被删除，释放存储空间。
- 当 Bucket 中有新的视频上传后，需要立刻进行内容黄反审核处理，违规的内容需要被立刻清除。
- 当 Bucket 中发生若干操作如 PutObject、DeleteObject 等，需要触发后续的自定义处理流程，可以通过CFC函数计算的BOS触发器来实现。

基于上述众多场景，BOS 的用户仅需在控制台中对该事件通知的触发条件进行简单的配置，如事件名称、是否开启加密鉴权、监测资源、覆盖资源和触发应用等。配置完毕后，当指定Bucket 或者指定前缀下文件的监测资源发生变化后，系统会自动触发相应的应用。

前提条件

- 已注册百度智能云账号。具体操作请参见注册百度智能云账号。
- 事件通知规则中覆盖的资源是某一个 Bucket 或者 Bucket 中某一前缀下的全部文件。因此需要先选择创建事件通知规则的 Bucket，详情请参见创建 Bucket。

支持地域

目前仅支持北京、苏州、广州，更多地域支持请提交工单。

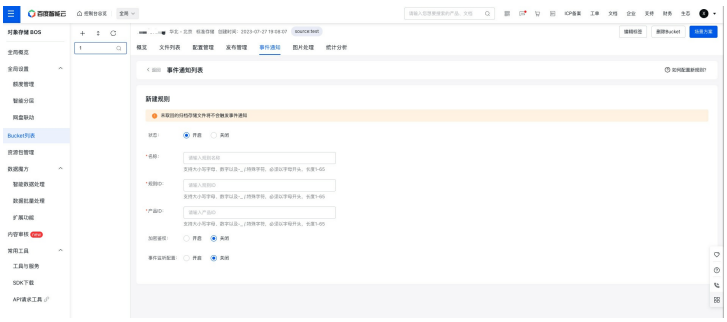
创建规则

> **注意：**未取回的归档类型 Object 将不会触发事件通知，已取回的归档类型 Object 可以正常触发。

1. 登录 BOS 管理控制台，进入全局概览页面。
2. 在 BOS 控制台左侧的导航栏中选择目标 **Bucket**，点击打开该 Bucket 的文件列表页。
3. 点击页面上方导航栏的**事件通知**页签，进入事件通知页面。
4. 点击**创建规则**，进入创建页面，配置以下信息：

参数	说明
状态	选择是否开启新规则的事件通知。
名称	• 支持大小写字母、数字、中文以及 -_ /. 特殊字符。 • 以字母开头，长度 1-65。
规则 ID	• 支持大小写字母、数字以及 -_ /. 特殊字符。 • 以字母开头，长度 1-65。
产品 ID	• 支持大小写字母、数字以及 -_ /. 特殊字符。 • 以字母开头，长度 1-65。
加密鉴权	选择是否对该事件通知规则进行加密。 若开启加密： • 输入长度为 32 位的自定义密码，该密码支持大小写字母和数字，长度为 32 位。 • 或者直接输入 SK。 • 如果只创建一个空事件，则必须设置密钥。 请记得及时保存密码以免丢失，BOS 不保存密码。

事件监听配置	根据需要选择是否开启事件监听配置。若开启需配置：
	• **监测事件**：Bucket 内发生的 API 行为。当前 BOS 支持以下几种监测事件，您可以按需单选或多选若干种事件。
	  • AppendObject，即发生追加上传行为；
	  • CompleteMutipartUpload，即三步上传完成；
	  • CopyObject，即发生数据拷贝行为；
	  • PutObject，即发生简单上传行为；
	  • PostObject，即发生表单上传行为；
	  • FetchObject，即发生从指定 URL 抓取资源并存储行为；
	  • DeleteObject，即发生删除单个 Object 行为；
	  • "BosLifecycle:Copy"，生命周期Lifecycle触发的复制事件，包含CopyObect请求和分块复制第三步CompleteMultipartUpload请求。
	• **覆盖资源**：指该规则监听覆盖的数据范围。覆盖资源除了支持 Bucket 级别资源，还可以通过指定资源前后缀的方式进行配置。同一规则最多可以监听 5 项不同的覆盖资源。
	  • 选择前后缀的方式时，单条前缀中最多只能有1个 * 符号；
	• **触发应用**：指该规则需要告知的下游业务。用户可以根据自有业务的架构，灵活设置接收触发通知的应用。触发应用的类别分为百度官方应用、函数计算CFC、自定义应用三类。触发应用详情请参见触发应用分类。



5. 点击**确定**，保存规则配置信息，成功创建的规则可以在“事件通知”页面的规则列表查看。也可以对已有规则进行修改、删除等管理操作。



触发应用分类

触发应用的类别分为百度官方应用、函数计算 CFC、自定义应用三类。其中，百度官方应用是 BOS 基于百度智能云强大的数据 AI 处理能力，将众多高频、易用的 AI 应用进行了包装，用户仅需在控制台设置一个接收应用处理结果的 URL，即可用最低的开发成本和最高的开发效率享受百度AI能力为业务赋能。

当前，百度官方应用包括通用文字识别、通用物体识别和音视频转码三项。

应用类别	应用名称	应用描述
百度官方应用	通用文字识别	• 百度通用文字识别服务支持多场景下的文字检测识别，多项 ICDAR 指标世界第一，是百度智能云上众多 AI 能力中使用面最广、积累最丰富的能力之一。 • 当触发通用文字识别应用时，BOS 将上传到 Bucket 中的数据自动推送到百度通用文字识别服务。文字识别完成后，BOS 会将识别结果以特定 json 文件的形式，推送到用户设定的接收 URL。
	通用物体识别	• 百度通用物体识别服务为用户提供了图片内容分析、提取、标签化输出的能力，对于有图片聚类、图片识别等场景的上层应用，有非常重要的作用。 • 当触发通用物体识别应用时，BOS 将上传到 Bucket 中的数据自动推送到百度通用物体识别服务。物体识别完成后，BOS 会将识别结果以特定 json 文件的形式，推送到用户设定的接收 URL。
	音视频转码	• 音视频转码针对海量媒资提供了高效、智能、稳定的音视频处理服务，包括：普通转码、倍速转码、智感超清转码、视频编辑、视频加密、智能抽帧等，为您带来多终端流畅高清播放体验。 • 当触发通用音视频转码应用时，BOS 将上传到 Bucket 中的音视频数据自动推送到百度音视频转码服务。转码完成后，BOS 会将结果推送到用户设定的接收 URL 中。
	异常图像检测	• 异常图像检测应用支持通过配置检测率，目标输出路径的方式将image/detect的检测结果以json格式投递到目标存储桶及路径下 • 生成文件格式为${prefix}/${detectDate}/${srcBucket}_${detectTime},其中prefix为配置的目标路径，detectDate为检测的日期，例如20241001代表检测日期为2024年10月1日,srcBucket为配置异常图像检测的bucketName，detectTime为检测时间，例如2024-10-01_12代表检测时间为2024年10月1日12点-13点范围
函数计算 CFC	函数计算 CFC	• 函数计算 CFC (Cloud Function Compute)是百度智能云官方推出的基于事件触发、全托管的云端计算能力。 • 当设置完触发的函数计算公式之后，BOS 将上传到 Bucket 中的数据自动推送到百度函数计算服务。计算完成后，BOS 会将计算结果推送到用户设定的接收 URL 中。
自定义应用	自定义应用	• 自定义应用即用户下游自有的 APP 或其他应用系统，用户仅需填写该业务接收触发通知的 URL，BOS 就会将 Bucket 内监测到的事件推送到该 URL。

相关 API

- PutNotification 接口：使用 PutNotification API 接口创建事件通知规则。
- GetNotification 接口：使用 GetNotification API 获取已创建的事件通知。

配置BOS多用户访问控制

概述

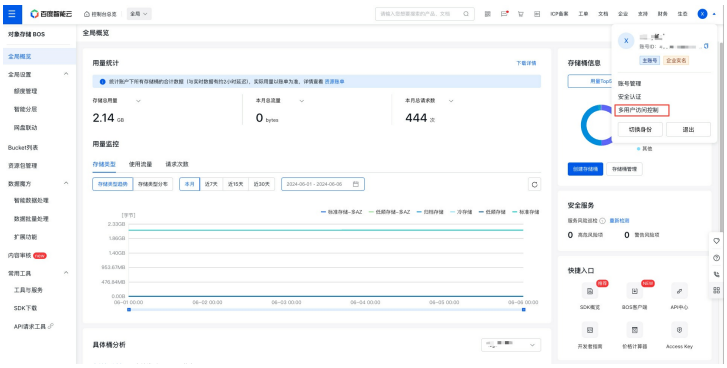
多用户访问控制，主要用于帮助用户管理云账户下资源的访问权限，适用于企业内的不同角色，可以对不同的工作人员赋予使用产品的不同权限，当您的企业存在多用户协同操作资源时，推荐您使用多用户访问控制。

使用场景

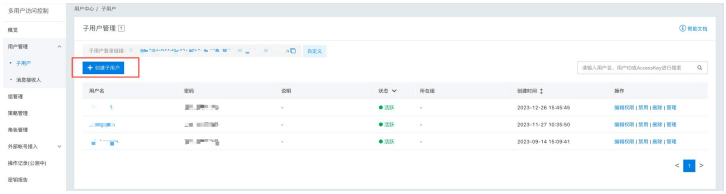
- 中大型企业客户：对公司内多个员工授权管理；
- 偏技术型 vendor 或 SaaS 的平台商：对代理客户进行资源和权限管理；
- 中小开发者或小企业：添加项目成员或协作者，进行资源管理。

创建子用户

1. 主账号用户登录后在控制台选择“多用户访问控制”进入用户管理页面。



2. 在左侧导航栏点击“用户管理”，在“用户管理 > 子用户”列表页，点击“创建子用户”。



3. 在弹出的“创建子用户”对话框中，完成填写“用户名”和确认，返回“子用户管理列表”区可以查看到刚刚创建的子用户。(如果子账号需要上传文件，建议授权“编程访问”，否则上传文件会报错“鉴权失败”)



配置策略

策略类型

BOS 支持系统策略和用户自定义策略两种。

- 系统策略：百度智能云系统为管理资源而预定义的权限集，这类策略可直接为子用户授权，用户只能使用而不能修改。
- 自定义策略：由用户自己创建，更细化的管理资源的权限集，可以针对单个实例配置权限，更加灵活的满足账户对不同用户的差异化权限管理。

系统策略

系统策略包含 BosFullAccess 和 BosListAndReadAccess 两种，权限范围详细如下：

策略名称	权限说明	权限范围
BosFullAccess	BOS 管理权限	管理权限包括 READ、WRITE、LIST对应的API，还包含PutBucketACL, GetBucketACL, PutBucketCors, GetBucketCors, DeleteBucketCors, PutBucketLogging, GetBucketLogging, DeleteBucketLogging。
BosListAndReadAccess	BOS 只读权限	权限包括 ListObjects, ListMultipartUploads, GetBucketLocation, HeadBucket, GetObject, GetObjectMeta, ListParts 接口。

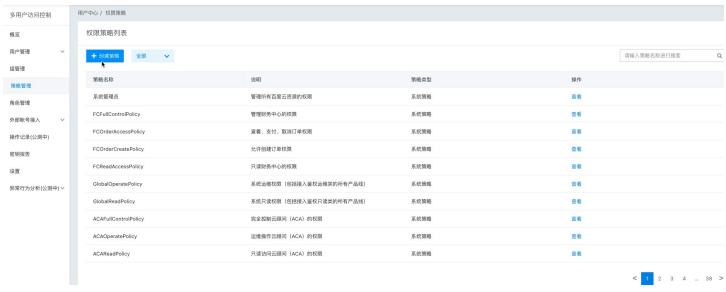
自定义策略

主用户可以通过点击“策略管理>创建策略”添加自定义策略来进行实例级权限控制，自定义策略的添加有"按策略生成器创建"、"按标签创建"和"按策略语法创建"三种方式，用户可以根据具体的权限设置修改策略内容。具体配置方法可以参见创建自定义策略。

策略创建流程

1. 点击创建策略

页面左侧选择"策略管理"，在新的页面中点击"创建策略"。

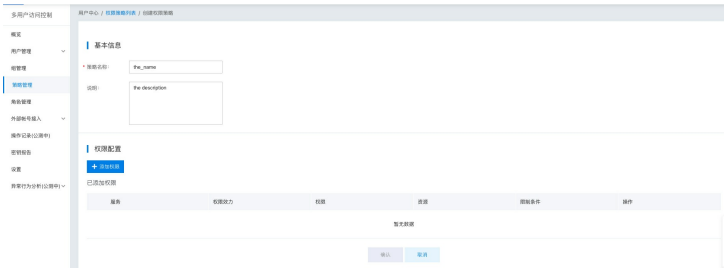


2. 选择创建策略的方式

您可以选择"按策略生成器创建"或者"按标签创建"。若您需要对某个标签下的实例创建策略，请选择按标签创建。若您不需要为标签创建策略，则选择按策略生成方式创建。

3. 填写策略基本信息

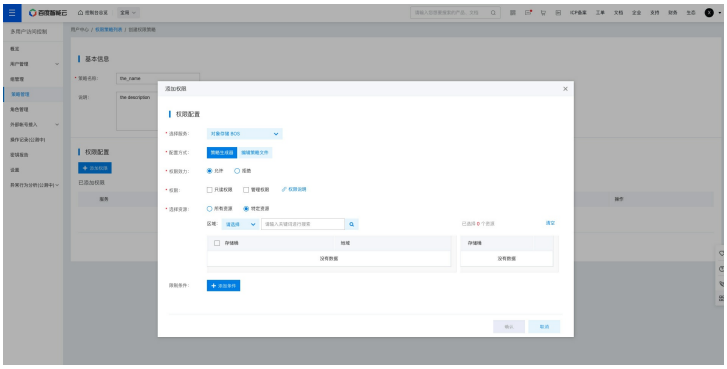
在创建策略的页面中，填写需要创建的策略的名称，并在说明中填写该策略的备注信息，如该策略应用的实例，希望赋予的子用户或者策略的使用场景，以免将该策略和其他策略混淆。



4. 填写策略权限配置

填写完策略的基本信息后，您需要填写该策略中包含的使用权限。

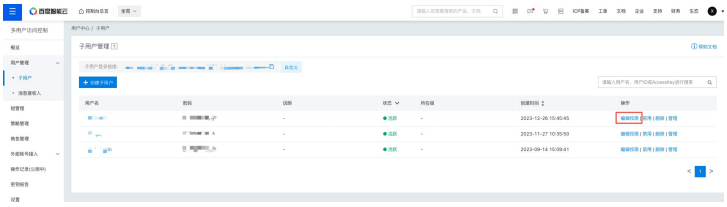
- 首先，您需要填写该策略生效的产品服务。若您希望将该策略生效于 BOS 服务，则在服务中选择"对象存储 BOS"；
- 选择策略的配置方式。其中，策略生成器方式允许您将系统中已生成的权限配置到该策略中；而策略文件需要您编辑 ACL 权限策略，详细的语法可参考文档策略语法。BOS 策略文件的详细解释请参见策略文件说明；您也可以使用BOS ACL编辑工具获取策略模板及自定义策略内容。
- 若您选择策略生成器方式，您需要进一步该权限对应的效力为允许还是拒绝。若您选择允许，则该策略中选择的权限将允许被授权的用户操作；若您选择拒绝，则该策略中选择的权限将禁止被授权用户操作；
- 选择权限，您需要选择该策略中包含的权限为只读权限还是管理权限。关于两种权限的区别和说明，请查看本页面上方"系统策略"中的说明；
- 您还可以选择该策略针对的资源。在 BOS 服务中，您可以指定该策略能够生效的具体 Bucket。若您不选择，那么将对全部 Bucket 生效；
- 最后，您需要选择该策略在生效时间上的限制条件。您可以指定该策略在某一时刻后或者某一时刻前生效。



用户授权

当您创建完子用户并生成策略后，即可以为该子用户授予该策略。授予后，子用户在访问主用户资源时，将受到授予策略中已配置权限的限制。

在"用户管理->子用户"列表页中希望授权的子用户后"操作"栏中选择"添加权限"，为用户授予已生成的策略。



在授权页面中，选择希望授予该子用户的权限对子用户进行访问权限授权。左侧的策略列表中包括全部系统策略和您已经创建的自定义策略。您可以在搜索框中搜索相关的策略名称后，勾选该策略后点击确认完成授权。



> **说明：**如果在不修改已有策略规则的情况下修改某子用户的权限，只能通过删除已有的策略并添加新的策略来实现，不能取消勾选已经添加过的策略权限。 如果创建子用户时没有勾选"编程访问"，子账号上传文件需要添加 IAMManageAccessKeyPolicy 模板，并创建一个AKSK才可以正常上传文件。

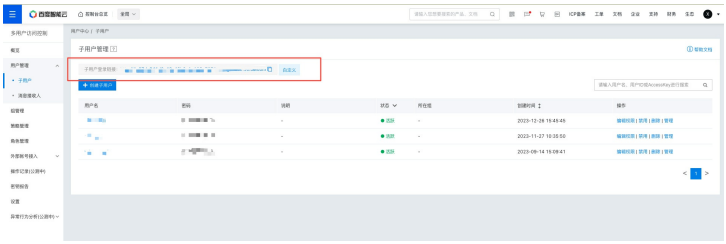
需要说明的是，主子账户中子用户授予权限的优先级低于 BOS 自有 Bucket ACL 鉴权优先级。Bucket ACL 鉴权是根据主用户 UserId 来进行鉴权的。若系统检测到 Bucket ACL 规则中"effect"字段为"Deny"，且"Userid"字段匹配到通配符"*"或子用户所属主账户 UserId 时，那么 BOS 将优先遵循主账号中 Bucket ACL 的权限配置，此时主账号下子账号授予的权限将失效。

Bucket 创建时，Bucket ACL 会包含一条默认规则，表示 Bucket 创建者拥有 FULL_CONTROL 权限。Bucket ACL 设置参考Bucket权限控制

子用户登录

添加完子用户权限后，子用户可以使用以下两种方式使用 BOS 服务。

- 点击子用户名可以查看 IAM 用户详细信息，包括 AKSK 及子用户的权限等。子用户可以使用 AKSK 通过 BOS 的周边工具、SDK、API 来使用BOS服务。
- 将控制台上方的子用户登录链接发送给子用户，子用户可以通过该链接登录控制台。登录后，该子用户将根据被授权的策略对主账户资源进行操作和查看。



相关文档

其他操作请参考多用户访问控制。

图片审核服务

概述

BOS 图像审核服务是百度智能云对外提供的图像智能审核服务，基于深度学习的智能内容审核方案，准确识别图片中的涉黄、涉暴涉恐、政治敏感、微商广告、恶心等内容。为了方便您使用，BOS 为开发者提供 API 或 CLI 调用的同时，也提供了界面化的图审控制台，用户可在控制台查看 BOS 审核结果中的违规图片，并按需进行人工复审。

注意事项

- 为避免违规图片传播，BOS 会将违规图片的权限暂时设为"私有权限"，通过身份识别后当您拥有图片读取权限才可访问。
- 在已配置增量图像自动审核规则的 Bucket 内，如果图片是由数据同步功能复制而来或生命周期策略产生，将不会被自动审核。
- 对于未取回的归档类型 Object，图审策略不会生效；已取回的归档类型 Object 可正常被审核。
- 2022年7月图像审核功能升级，将不再支持设置图像审核上限，且开通功能需要您先开通ICR后付费并请保持开通状态，从而确保您可享受无限的图像审核次数，保证审核业务不中断。
- 因BOS图像审核依赖内容审核平台（ICR）的图片审核功能，该服务条款完全遵循ICR条款，同时会产生图片审核接口调用费用，请参考ICR定价文档。ICR为您提供了免费使用额度，您可前往领取icr免费图像审核调用次数，图片审核产生的调用次数将优先抵扣该免费资源，不足抵扣部分将进行按量付费。

设置审核规则

1. 登录 BOS 管理控制台。
2. 点击左侧导航栏中的【图像审核】，阅读开通图像审核服务指引。

> 您需完成开通ICR服务同时开通其后付费才可设置图像审核规则。



3. 若您已经完成开通ICR并完成后付费，则可通过如下页面，点击【立即使用】，完成审核规则配置。



4. **设置审核规则**。
   - Bucket审核：选择需要审核的目标 Bucket，仅支持选择属于北京、上海、广州地域的Bucket。
   - 审核类型：选择需要审核哪些违规类型，目前提供了色情、暴恐和政治敏感三种违规审核能力。

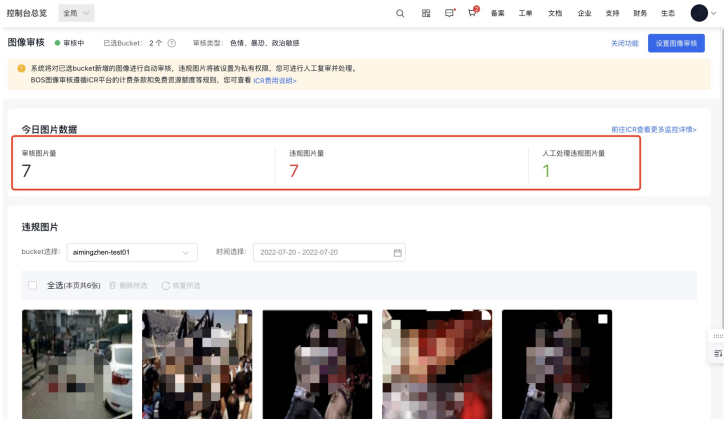


5. 点击**确认**，保存审核规则设置。

使用审核服务

查看审核数据

方式一： BOS 图审服务会监测审核您选择的 Bucket 中的所有新增图片，并将审核信息以按照【今日图像数据】统计概览的方式向您展示，信息一目了然。



- **审核图片量**：代表今天您开启图审的 所有Bucket 中，经过审核的图片总量。
- **违规图片量**：代表审核结果为违规图片的数量。
- **人工处理违规图片量**：代表基于违规图片量，管理员已经人工处理过的图片数量，操作包括：删除，恢复原有权限。

方式二： 您也可前往ICR查看更加具体的图像审核接口调用情况，可自定义时间范围，查询图像审核接口调用成功、失败情况。

违规图像复审

- 您可以通过选择Bucket 与时间维度的，选择展示您希望人工复审的违规图片。
- 您在对违规图片按需人工复审时，可对选中的图片执行查看详细信息、删除或恢复原有权限的操作。
- 您可以批量删除或者恢复违规图片进行，特别提示删除图像将彻底删除从BOS中删除，请谨慎操作。



图片服务

概述

BOS 图片处理是一项基于 BOS 核心存储业务的图片应用类服务，可提供灵活、多样、实时的图片处理服务。您只需要将原始图片上传到 BOS，通过在原始图片的请求中添加处理参数，即可实时地得到处理结果图。BOS 图片处理服务包括图片缩略、裁剪、格式转换、旋转、加文字/图片水印等，处理结果图会实时返回。

您可以通过 BOS 控制台和 API 方式对存储在 BOS 中的图片进行处理。您在 BOS 中创建的每个 Bucket 均默认具有图片处理的能力，无需额外开通操作。

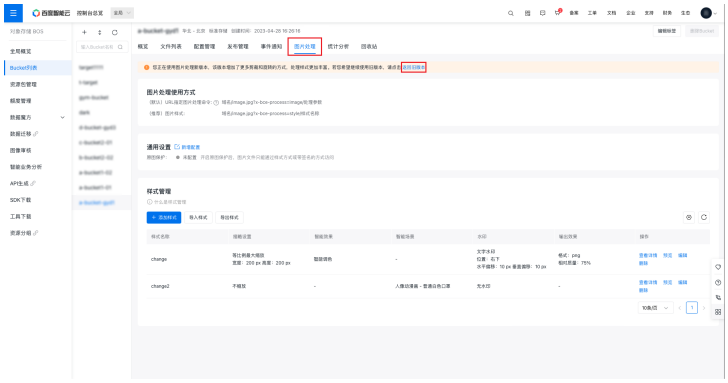
> 图片处理服务的流程：请参考数据处理介绍。
>
> 图片处理服务相关数据处理计费说明请参考数据处理费用。

图像处理协议

BOS 图像处理目前支持 V1.0 和 V2.0 两个版本的图像处理协议。其中，V2.0 版本所支持的图像处理项目更加丰富，且在命令的调用上与 V1.0 有所不同，因此建议您使用 V2.0 版本进行处理。 V1.0 作为老版本，会

在将来停止服务，但当前两个版本均可使用，系统将自动识别并兼容两种协议版本。

在您登录 BOS 控制台后，在 Bucket 的图像处理页签中可以对两个版本切换使用。



通过图片样式访问图片处理服务

图片样式操作说明

1. 登录百度智能云官网。

> 若您未注册账号，须先注册账户。

2. 进入 BOS 控制台

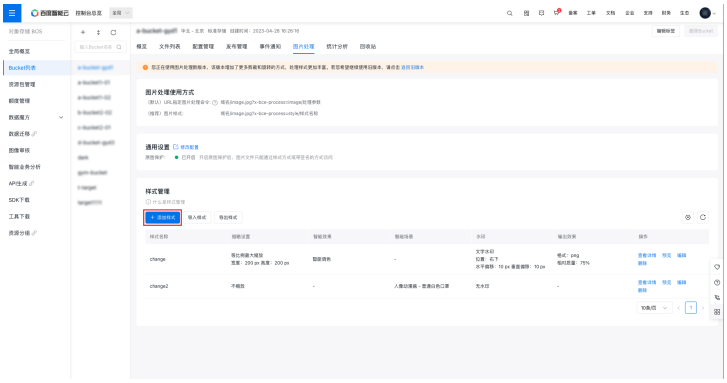
在页面左侧导航栏中选择 "存储和 CDN > 对象存储 BOS" 进入 BOS 控制台。

3. 选择原始图片所在的Bucket 并进入图像处理

在左侧的 Bucket 列表中选择 Bucket 名称并进入图像处理页面。在该页面中，您可以创建新的图像处理样式，也可以在列表中查看已创建的图像处理样式。

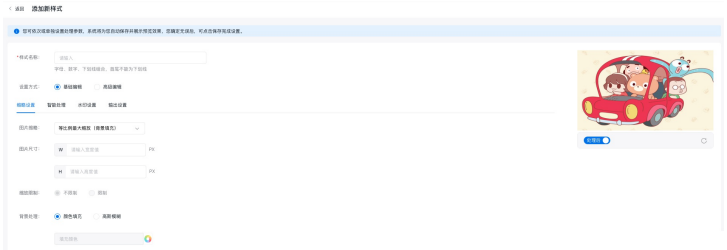
4. 添加样式

点击添加样式按钮，进入添加样式页面。关于图像处理协议 V2.0 版本的样式设置方法，您可以查看图像处理协议 V2.0；关于图像处理协议 V1.0 版本的样式设置方法，您可以查看图像处理协议 V1.0。

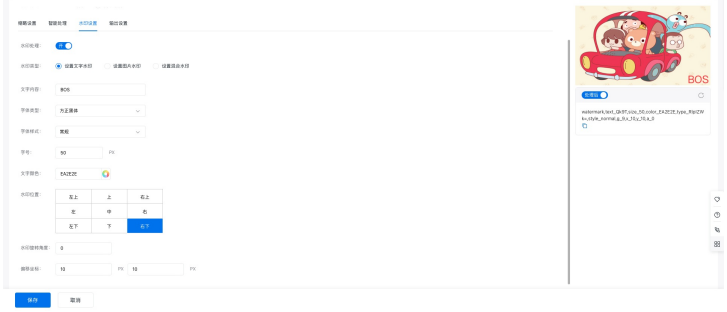


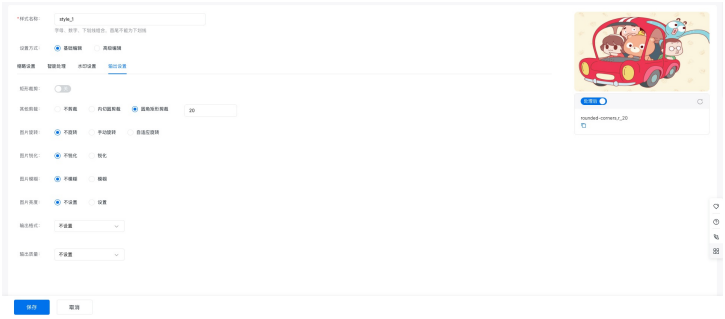
在控制台中，您可以通过基础编辑方式和高级编辑方式对样式进行编辑。其中基础编辑方式是通过图形化界面对样式进行编辑，而高级编辑方式需要您输入相应的命令语句进行设置。

- 在基础编辑中，您可以进行样式名称设置和缩略设置。



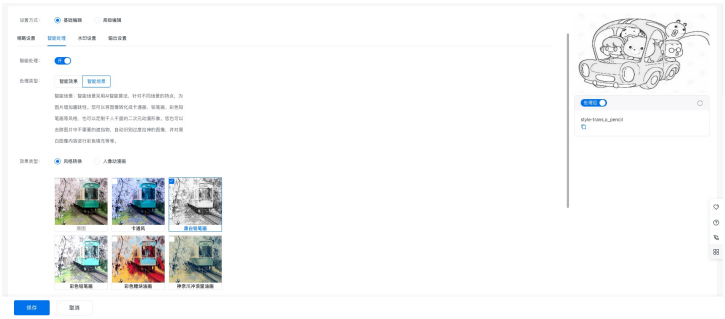
- 也可以对该图像进行水印设置和输出设置。



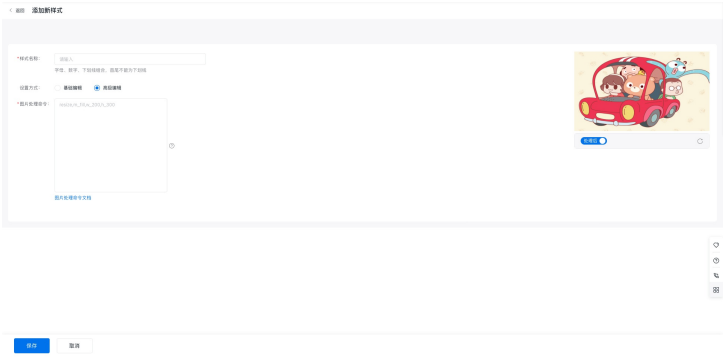


- 您可以进行智能处理设置，处理类型包括**智能效果**和**智能场景**。

> **注意：**智能效果与智能场景无法叠加，以最后一次选择的处理类型为准，您可通过右侧的样例图片预览智能处理后的效果。



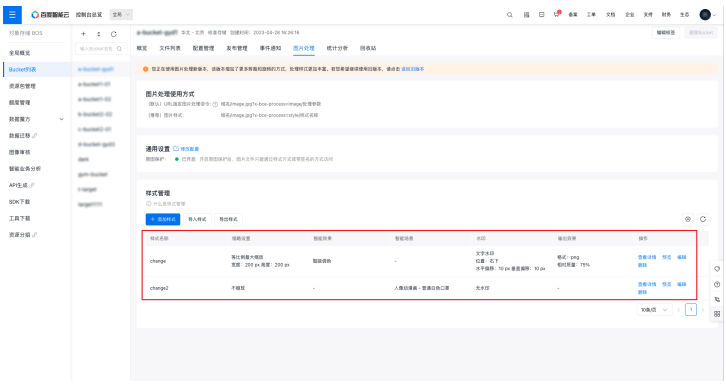
- 在高级编辑中，您可以直接输入命令行设置图片样式。图片服务命令行的具体操作请参见图像处理协议 V2.0和图像处理协议 V1.0。



5. 样式列表

样式添加完成后会在图像处理的样式管理列表中显示，您可以选择操作列的**"查看详情"**查看该样式对应的命令详情，也可以选择**"预览"**在新页面中预览样式，或选择**"编辑"**、**"删除"**对该样式进行编辑或删除操作。

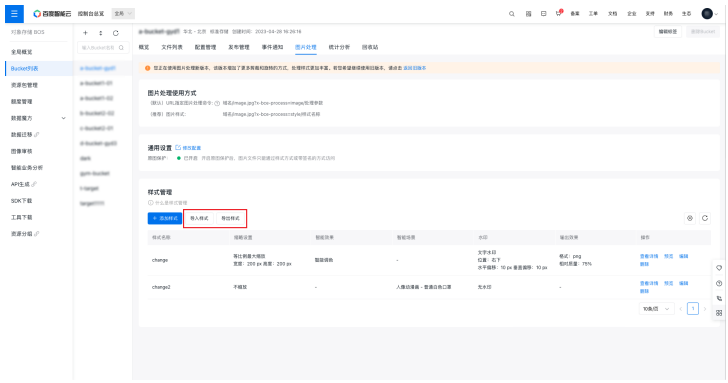
> **注意：**
>
> - 图像处理协议 2.0 生成的样式，仅展示在新版协议的列表中；图像处理协议 1.0 生成的样式，仅展示在旧版协议的列表中，两个协议下的样式不会共同展示，请在页面上方切换版本进行查看。



6. 样式导入导出

样式添加完成后，您可以点击**"导出样式"**批量导出当前全部样式，BOS 会自动生成 .txt 文件，您可以将该样式在本地保存，也可以与其他账号共享。

若您已有样式文件，可点击**"导入样式"**选择.txt 文件，BOS会自动导入样式列表。



原图保护

- 原图保护能对原始图片提供保护服务，能够阻止恶意用户对原始图片的请求。
- 开启原图保护后，将无法通过源文件链接直接访问存储桶中的图片文件，仅能使用携带样式名称的 URL 访问处理后的图像或使用携带签名的 URL 访问源文件。
- 该功能适用于不希望通过图片处理后的链接访问到原图片文件的场景，如原图资源防盗、业务防刷等，针对这些使用场景，您可对指定Bucket开启原图保护，以提升原图片文件的安全性。

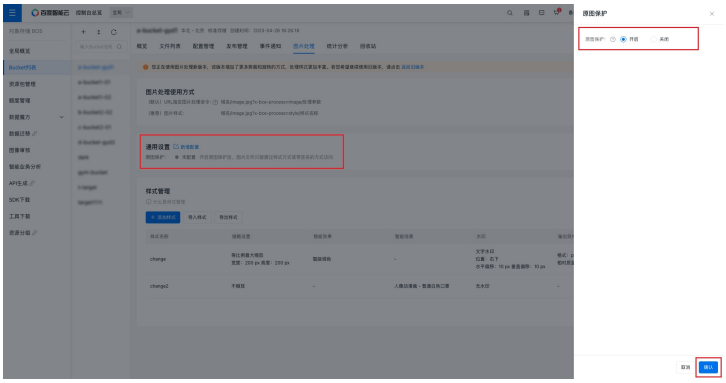
例如，您可设置名称为 stylename 的水印样式，若源文件链接为：`https://a-bucket1-01.bj.bcebos.com/BOS.png`，开启原图保护后，**仅可以通过以下两种方式访问该图片文件**：

1. 访问处理后的图片文件：使用携带样式名称的文件 URL 访问处理后的图片文件，格式为 `https://bucket.bj.bcebos.com/BOS.png?x-bce-process=style/stylename`

2. 访问原图片文件：使用携带签名的文件 URL 访问源文件，格式为 `https://bucket.bj.bcebos.com/BOS.png?authorization=`

> **说明**：如果图片获取者未验证被授权的签名，只能通过"样式方式"访问处理后的图像，直接访问原图或通过"命令方式"访问会被拒绝。

若您希望开启原图保护，可进入**图片处理->通用设置->修改配置**，选择开启并确认即可。



> **注意**：未取回的归档类型 Object 不会触发图片处理，已取回的归档类型 Object 可正常处理。

统计与监控

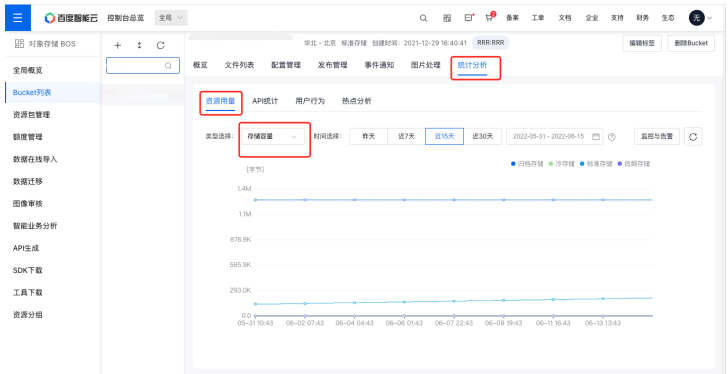
概述

BOS 统计分析主要是以各 Bucket 中的数据为基础，从资源用量、API 统计信息、热点信息等维度进行监控和分析，以图形或报表形式呈现。统计分析报表可以帮助用户了解资源的使用状况和业务特征，方便用户做业务分析。

统计分析分为四个模块，能够查看资源用量、API 统计信息、热点信息、智能分层维度信息。关于监控项的信息，可以查看监控项概览。

资源用量分析

1. 登录 BOS 管理控制台，进入全局概览页面。
2. 在 BOS 控制台左侧的导航栏中选择目标 **Bucket**，点击打开该 Bucket 的文件列表页。
3. 点击页面上方导航栏的**统计分析->资源用量**页签，进入资源用量分析页面。



4. 您可以在资源用量分析页面查看 Bucket 的资源用量统计数据，可查看不同时段不同时间粒度的**存储容量**、**使用带宽**、**使用流量**、**请求次数**、**数据取回量**。

- **存储容量**：展示标准存储、低频存储、冷存储和归档存储四种存储类型的容量数据；
- **使用带宽**：展示每秒外网读数据总量、每秒外网写数据总量、每秒内网读数据总量、每秒内网写数据总量，单位bit/s；
- **使用流量**：展示外网读取量、回源流量和跨区域复制流量数据，单位Byte；
- **请求次数**：按照标准存储、低频存储、冷存储和归档存储的读请求和写请求分别展示请求次数；同时可以查看每秒删除请求总次数、每秒读请求总次数和每秒写请求总次数；错误请求总次数以及数据取回请求数据。
- **数据取回量**：低频存储、冷存储和归档存储的数据取回流量数据。

> 说明： 目前仅支持对华北-北京、华东-苏州、华南-广州地域的存储桶进行热点分析。

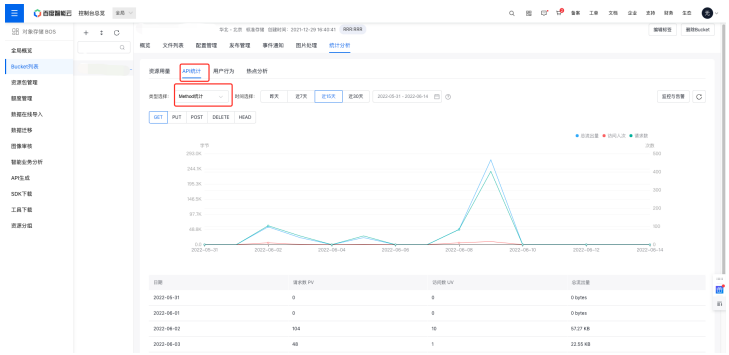
API统计

1. 登录 BOS 管理控制台，进入全局概览页面。
2. 在 BOS 控制台左侧的导航栏中选择目标 **Bucket**，点击打开该 Bucket 的文件列表页。
3. 点击页面上方导航栏的**统计分析->API 统计**页签，进入 API 统计分析页面。
4. 您可以基于选择的 Bucket，统计指定时间范围内 HTTP 的访问行为，及系统返回的 HTTP Code。

- **Method 统计**包含 GET、PUT、POST、DELETE、HEAD 等访问请求，访问分析图表会呈现上述各类请求的总请求数 PV、访问人次 UV 以及总流出数据量。

```
![image.png](http://bjhw-bce-online-public0.bjhw.baidu.com:8109/proxy-external/?url=http://bce-cdn.bj.bcebos.com/doc/bce-doc/BOS/image_117d7b6.png)
```

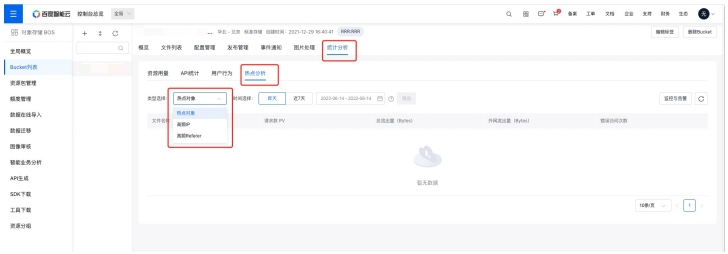
- 访问分析图表还会呈现系统返回的 **HTTP Code**，包括 200、300、403、408、503 等 2XX、3XX、4XX、5XX 系列返回码的统计情况。



热点分析

热点分析查看

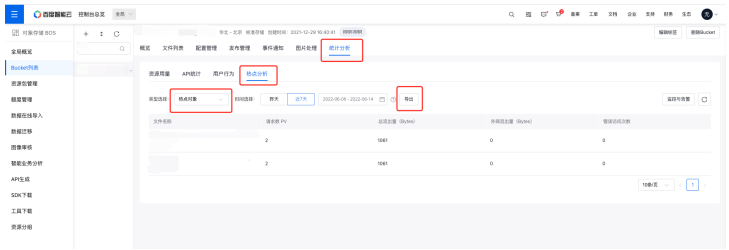
1. 登录 BOS 管理控制台，进入全局概览页面。
2. 在 BOS 控制台左侧的导航栏中选择目标 **Bucket**，点击打开该 Bucket 的文件列表页。
3. 点击页面上方导航栏的**统计分析->热点分析**页签，进入热点分析页面。



4. 您可以根据"请求数、总流出量、错误访问"等维度，对各**热点对象**、**高频 IP**进行统计和排序，方便快速定位热点信息。

热点分析数据导出

热点对象、**高频 IP** 的统计数据，支持在线导出，方便您进一步分析。



智能分层

开启存储桶智能分层后可以，可通过智能分层统计数据，查看指定日期的转化存储容量。

1. 登录 BOS 管理控制台，进入全局概览页面。
2. 在 BOS 控制台左侧的导航栏中选择目标 **Bucket**，点击打开该 Bucket 的文件列表页。
3. 点击页面上方导航栏的**统计分析->智能分层**页签。



额度管理

概述

BOS 提供额度管理功能。该功能支持对您账号中存储桶（Bucket）数量上限，以及账号下存储桶的总容量与文件个数上限进行配置，实现对存储服务弹性能力的自定义控制。

配置管理：

1. **Bucket 数量**： BOS 支持为账号中 Bucket 数量上限进行自定设置。默认情况下，BOS 最多支持同一账号创建最多 100 个 Bucket，当创建的 Bucket 数量达到您设定的上限时，创建新的 Bucket 将不被允许，请合理设置该额度；
2. **存储桶总容量**： BOS 支持自定义设置账号下的存储总容量，即同一账号下全部 Bucket 容量总和。BOS 为您默认设置总容量为无上限，若您希望根据财务预算情况进行修改，可将总容量上限修改为某一数值，但该数值需大于等于您当前的实际存储总量。
3. **文件总数上限**： BOS 支持自定义设置账号下的文件总数上限。BOS 默认设置文件总数无上限，您可通过该功能控制文件总数为某固定数值以下。当您设置的数值小于等于当前文件总数时，bucket将不能上传新的文件。

操作指南

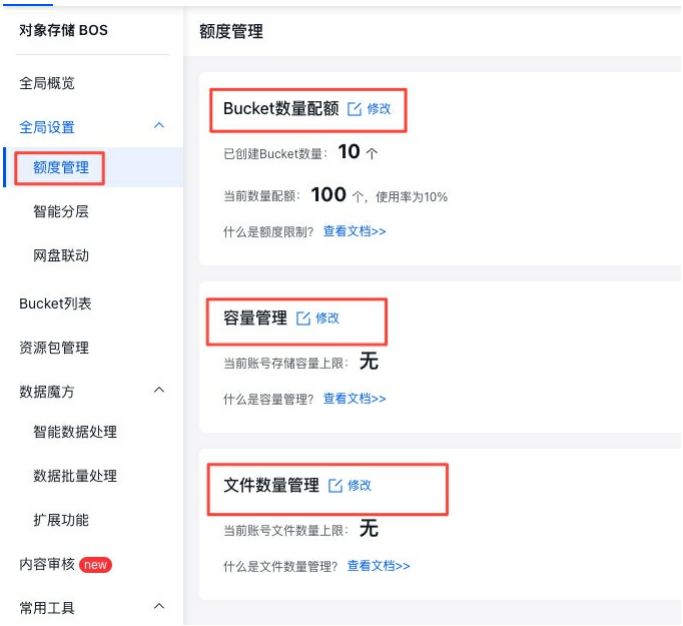
1. 登录百度智能云官网

登录百度智能云官网。

- 若您未注册账号，须先注册账户。
- 若您已注册，可直接登录。

2. 进入 BOS 额度管理

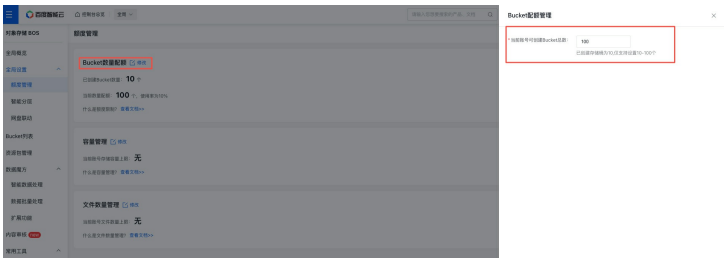
点击左侧导航栏中的"额度管理"，进入额度管理页面。该页面中支持对账号下的 Bucket 总数和账号下总容量与文件总数上限进行设置。



3. 修改 Bucket 数量上限

您可以修改 Bucket 数量上限为当前 Bucket 数量到 100 间的任意数值。如您此时已创建 2 个 Bucket，您可根据使用情况将 Bucket 数量上限修改为 2-100 间的任意数值。

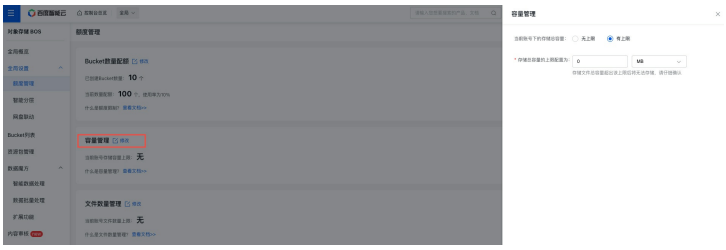
- 当您创建的 Bucket 数量超过设定的上限时，将不能创建新的 Bucket；
- 您可以在 Bucket 总数达到上限时，继续在允许的范围内调整 Bucket 数量上限。



4. 修改总容量上限

您可以修改账号下的存储总容量。BOS 默认存储总容量无上限，若您希望根据财务情况对存储容量进行限制，可以将其修改为有上限，并设定上限数值。

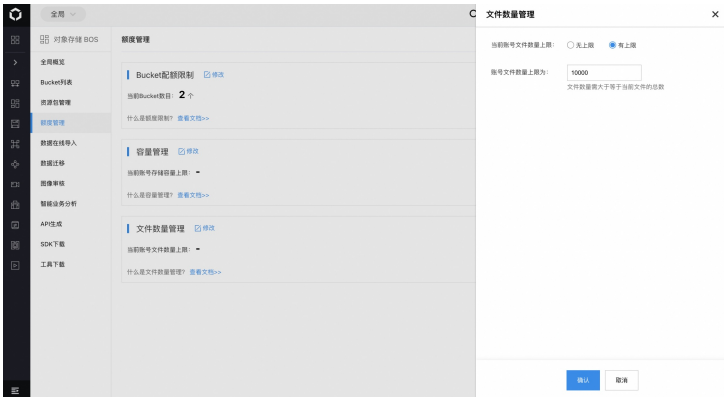
- 存储文件总容量超出该上限后，任何新的文件都将无法存储至 BOS，请您仔细确认；
- 存储总容量超出该上限后，文件将无法被复制，请您仔细确认；
- 设定的存储上限不得低于当前已存储总容量；
- 当您设定存储上限后，若希望继续存储新的文件，可以继续修改总容量上限。



5. 修改文件总数上限

您可以修改账号下的文件总数量。BOS 默认存储文件数量无上限，若您希望根据财务情况对存储文件总数上限进行限制，可以将其修改为有上限，并设定上限数值。

- 存储文件总数量超出该上限后，任何新的文件都将无法存储至 BOS，请您仔细确认；
- 存储文件总数量超出该上限后，文件将无法被复制，请您仔细确认；
- 设定的文件总数上限不得低于当前已存储文件总数；
- 当您设定存储上限后，若希望继续存储新的文件，可以继续修改文件总数上限。

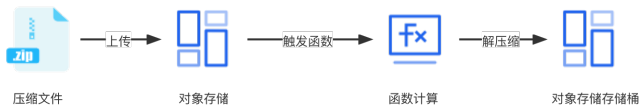


6. 修改完成

文件自动解压

简介

文件解压缩功能是对象存储（BOS）基于 百度智能云函数计算（CFC） 为用户提供的数据处理解决方案。用户为存储桶添加文件解压缩规则后，当压缩文件上传到对象存储时，将自动触发对象存储为您预配置的云函数，自动将文件解压到指定的存储桶和路径中。文件解压缩的流程图如下所示。



您也可利用CFC，在事件通知规则中设置 BOS文件操作自动触发更多自定义函数处理，请查看 函数计算的配置指南。

注意事项

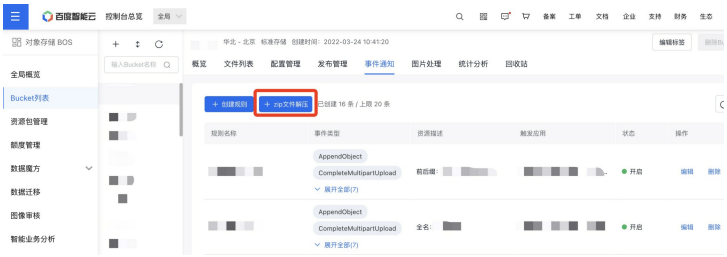
- 文件解压缩仅支持解压 **ZIP** 格式文件。
- 若您在对象存储控制台上为存储桶添加了文件解压缩规则，可以在 云函数计算（CFC）控制台 上看到您所创建的文件解压缩函数，请**不要**删除或随意修改此文件解压缩函数，否则可能导致您的规则不生效。
- 已上线函数计算服务的地域均已支持ZIP包解压缩，包括有北京、广州与苏州，更多支持地域可 提交工单。
- 压缩包中的目录或者文件名请严格使用 UTF-8 编码，否则可能导致解压后的文件名或者目录名出现乱码、解压过程中断等情况；如果出现报错，您可以跳转到云函数控制台查看日志。
- 归档存储类型文件不支持解压缩。

- 使用该功能的前提是您已经开通对象存储与CFC服务，若您尚未开通函数计算（CFC），请前往 函数计算控制台 开通函数计算服务，按照提示完成服务授权即可。
- 解压单个压缩包最大处理时间为1800s，源压缩包过大会导致解压缩失败，推荐上传小于60GB压缩包、zip包中待解压文件数量不超过10000个。对象存储的解压缩功能限制说明基于云函数服务，其他限制请参见 函数计算限制说明。
- 对象存储解压缩功能依赖于云函数服务，函数计算服务为用户提供了 免费额度与定价。当您使用文件解压缩功能时，如果您的压缩包越大，将消耗更多的资源使用量；如果您解压缩的次数越多，则将消耗更多的调用次数。
- 对象存储解压缩功能对上传的文件压缩包进行自动解压，原压缩包和解压后文件都将收取存储空间费用，解压完成后，若您确定压缩包不再需要，需要您手动删除压缩包。

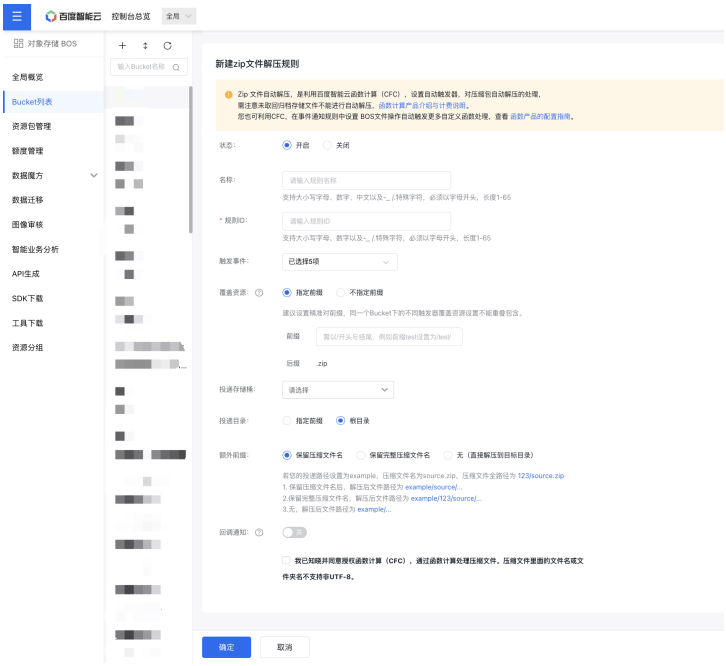
操作步骤

创建

1. 登录 对象存储控制台。
2. 在左侧导航栏中，单击【Bucket列表】，找到需要添加文件自动解压缩规则的存储桶，单击该存储桶名称，进入该存储桶管理页面。
3. 点击，选择【事件通知】，单击【zip文件解压】。



4. 进入ZIP文件解压规则配置页面，配置规则如下信息：



参数	参数说明
状态	默认开启，规则开启状态将对符合规则的压缩包进行自动解压，若想暂停执行该解压任务这可以操作关闭。
规则名称/规则ID/产品ID	解压规则的标识，创建后不可修改。支持大小写字母、数字、中文以及-_ /.特殊字符，必须以字母开头，长度1-65。
触发事件	· 事件是指触发云函数的操作。目前支持5个事件：**PutObject,PostObject,CopyObject,FetchObject,FetchObject,CompleteMultipartUpload(分片上传完成)** · 以上传操作为例，上传的方式可能是调用PutObject接口，也可能是调用POST Object接口，当选择事件为【PutObject】时，只有通过PUT Object接口上传的压缩包会触发解压缩。 · **如果您的文件是通过简单上传、分块上传等多种渠道上传到存储桶中，推荐您选择【全部】事件。**
覆盖资源	· 指压缩包上传到哪个路径时会触发云函数。如果选择指定前缀，则仅当压缩包上传到指定前缀路径下时才会触发云函数；如果选择不指定前缀，则压缩包上传到存储桶任意位置均会触发。 · **为了防止不必要的循环解压，推荐您配置覆盖资源时可精确到目录级别。**
解压格式	指当前可支持的压缩格式，目前仅支持 ZIP 格式压缩包解压。
投递存储桶	选择解压后的文件所存放的存储桶，仅支持与压缩文件上传存储桶相同地域的桶中。
投递目录	· 将匹配到的文件解压到此目标目录下。 · **建议您设置的投递目录与覆盖资源设置的前缀不存在资源重叠。**如果配置的目标文件前缀与触发条件存在包含关系，可能导致循环触发，可能产生不必要的费用，请您尽量避免这种情况。例如投递目录设置为pre/fix/，触发条件为pre/，当上传一个pre/fix/1.zip的压缩包时，将触发循环解压。
额外前缀	· 选择解压后的文件所存放的存储桶，仅支持与压缩文件上传存储桶相同地域的桶中。 * 保留压缩文件名：您可以将压缩包解压到以压缩包为名的前缀下。 * 保留完整压缩文件名：您可以将压缩包解压到以完整压缩包路径为名的前缀下。 * 设置为"无"：压缩包内的文件将直接解压到投递的路径下。 · 举例说明：若您的投递路径设置为example/，原压缩文件名为source.zip，原压缩文件全路径为 123/source.zip，压缩包内文件名为test.txt： * 保留压缩文件名后，解压后文件路径为 example/source/test.txt * 保留完整压缩文件名，解压后文件路径为 example/123/source/test.txt * 无，解压后文件路径为 example/test.txt
回调通知	开启回调通知，配置用于接收解压结果的回调 URL 地址，解压完成后 BOS 会将json格式的解压结果推送到该地址，回调结果详见回调结果示例。
CFC 授权	我已知晓并同意授权函数计算（CFC），通过函数计算处理压缩文件。解压缩需要授权云函数从您的存储桶中读取压缩包，并将解压缩后的文件上传到您指定的位置。因此需要添加此授权，由此您也可能产生CFC一定使用费用，参考CFC计费说明。并且已知压缩文件里面的文件名或文件夹名不支持非UTF-8字符。

5. 确认配置无误后，单击【确定】，即可通过事件通知规则列表看到规则已添加完成。

文件解压规则管理

您可以对新创建的解压规则进行如下操作：



- 单击【编辑】，修改文件解压缩规则。
- 单击【删除】，删除不使用的文件解压缩规则。

回调结果示例

```
POST /<URL>
Content-Type: application/json; charset=utf-8

{
    "domain": "bj.bcebos.com",
    "bucket": "<BUCKET_NAME>",
    "object": "<OBJECT_NAME_ZIP>",
    "target": {
        "bucket": "<BUCKET_NAME>",
        "pathPrefix": "bos/pathPrefix/80m"
    },
    "result": {
        "status": "success",
        "message": "success"
    }
}
```

分层命名空间存储桶操作说明

分层命名空间（namespace）是百度智能云推出的元数据管理功能。通过分层命名空间，可以很好的支持目录级别的mv与rename操作，同时优化常见读操作List、Head，提升数据处理效率与性能，能很好的满足大数据领域的数据湖计算场景。目前分层namespace功能设置粒度为存储桶级别，您需先获取使用权限后创建开启分层namespace的存储桶。更多功能介绍详见分层命名空间。

支持地域：

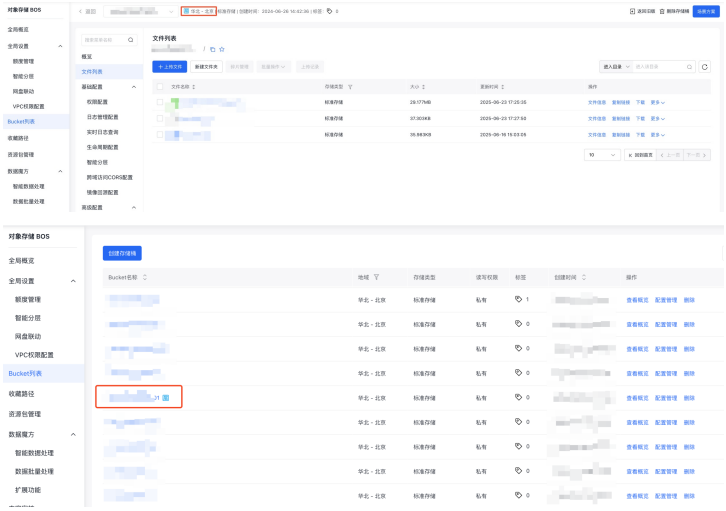
目前支持华北-北京地域、华南-广州地域。

分层namespace控制台操作说明

因开启分层namespace的存储桶与平层存储桶在对象操作与存储桶配置上基本相同，故本文档仅提示与现有平层存储桶相比，开启分层namespace的存储桶的区别之处。

1.存储桶属性为：分层namespace

若您的存储桶成功开启分层namespace，则会在存储桶信息中展示。开启后如下图：



2.文件列表

文件列表搜索功能，输入文件前缀，将进入该目录下，列表展示检索目录的所有文件信息，包括文件与文件夹等信息。



3.文件管理

开启分层namespace的存储桶不支持归档存储类型，即您不能将文件的存储类型设置为归档存储。同时也不支持配置文件软链接功能，若您需要使用归档存储或文件软链接，请使用标准平层namespace。

> **其他说明**： 开启分层命名空间的存储桶，功能支持与限制说明详见 分层命名空间功能支持说明。

联动百度网盘分享

概述

网盘分享功能是由对象存储（BOS）与百度网盘企业定制平台联合提供的文件分享功能，您可以将BOS中的文件或文件夹批量同步至网盘定制服务专属空间，之后可生成网盘的分享链接进行数据分发交付。

说明

目前网盘联动功能处于白名单试用阶段，您可以联系商务经理或者提交工单的方式申请试用，欢迎体验。

注意事项

1. 由于网盘联动功能依赖百度网盘企业定制服务平台的数据分享能力，网盘临时专属空间提供您免费存储7天数据，可获取7天内有效期的分享链接，若您需要更长时间，需付费网盘服务延长存储时间
2. 通过网盘分享链接分享数据，会产生一定百度网盘费用，包括极速下载流量费、百度网盘企业版定制服务专属空间长期（大于7天）存储费用等，以上费用由百度网盘统一收取。为保证服务连续与稳定，建议您购买百度网盘企业定制服务的正式功能。
3. 在网盘联动功能使用过程中，文件通过内网上传至百度网盘定制服务空间，该功能BOS免费提供，但同步过程会产生BOS数据读取请求费用。
4. 归档存储类型的文件，必须取回后才可同步至网盘，若未取回将导致数据上传网盘失败。
5. 目前仅支持华北-北京、华北-保定地域的存储桶中数据同步至网盘，若您需要更多地域支持请联系商务经理或者提交工单的方式联系我们。

操作步骤

授权同意BOS同步文件至网盘

1. 添加功能白名单后，登录 对象存储 BOS 管理控制台，在 BOS 控制台左侧的导航栏中选择**网盘联动**。
2. 勾选产品功能使用协议后，一键授权绑定网盘数据同步服务，即可开通成功。

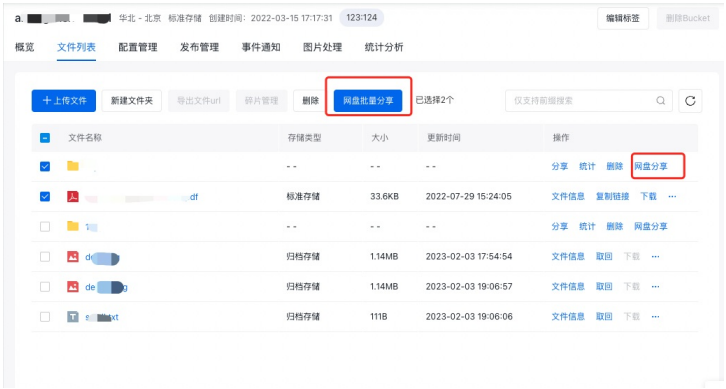


3. 保存成功后，主账户及其开通了网盘联动功能的子账号可直接在bucke的文件列表中选择文件或文件夹进行网盘分享，分享的内容将同步至授权的网盘企业版定制服务专属空间中。

> 注意：功能开通仅支持主用户，子用户在主用户开通权限后可进行数据分享，数据同步至少子用户需存储桶文件列表权限与文件读取权限。

分享至网盘

1. 登录 对象存储 BOS 管理控制台，在 BOS 控制台左侧的导航栏中选择**目标Bucket**，点击打开该 Bucket 的文件列表页。
2. 网盘分享支持分享单个文件、单个文件夹或多个文件（夹）。如需分享单个文件，在待分享的文件的操作栏中，点击【...】-**【网盘分享】**，。如需分享单个文件夹，操作栏中点击【网盘分享】。如需分享多个文件，点击需要批量分享的文件前的复选按钮后，点击**【网盘批量分享】**



> 单次分享请选择少于 5000 个文件，每个文件大小不超过 2TB

3. 配置网盘存储时长后，点击【立即同步】

> 1.网盘存储时长1-7天内免费，大于7天网盘将收取存储费用吗，您需提前网盘处购买存储空间，网盘存储时长决定了本次同步文件的最长分享有效期。例如，您创建30天的分享链接，则可在同步后完成后，可配置最长30天的分享链接有效期。
>
> 2.文件较大或个数较多时，数据同步可能会耗费一定的时间，请耐心等待，请勿在同步期间关闭或者刷新浏览器窗口。



4. 同步完成后，点击**分享**按钮，配置分享链接信息分享有效时长与是否使用极速流量交付，注意百度网盘极速流量使用您需已经完成其流量包购买，确保有足够的资源，然后点击**创建链接**。







5. 点击**复制链接及提取码**，发送给接收方。接收方打开链接，即可转存文件至百度网盘。

全局设置

VPC访问控制

概述

为了进一步保证企业数据的安全性，防止关键数据通过非授权的bucket流出，VPC访问控制提供VPC维度对bucket进行访问授权功能。

注意事项

- 仅支持主账号配置策略。
- 一旦配置权限策略后，所有**非允许效力的bucket访问均会被拒绝**，请合理设置权限策略。
- 该权限配置仅对VPC来源的访问生效，非VPC来源的访问不会生效。
- 配置权限策略后，在VPC内无论是直接访问BOS域名或者通过服务网卡访问BOS服务，VPC访问控制策略均生效。

操作步骤

1. 登录 对象存储 BOS 管理控制台。
2. 在左侧**全局设置**中，选择**VPC权限配置**，进入VPC访问控制配置页。
3. 在上方VPC选择栏选择需要配置的权限规则的VPC。



4. 点击**添加策略**进入VPC访问策略的编辑页面，根据需求选择授权效力类型，然后在下方资源范围选择权限生效的bucket，支持选择账号下自有桶或输入其他账号下的bucket名称，授权操作默认选择全部操作，确认信息后点击确认后提交策略。



收藏路径

BOS控制台通过添加收藏路径，您可以快速访问目标资源，路径支持配置Bucket、目录。

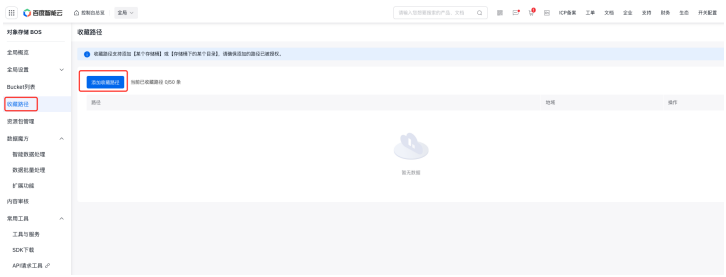
应用场景

通过访问控制功能已经完成授权子账号或其他账号进行某个路径（目录）下的文件读写权限，可通过添加收藏路径的方式，进行直接对授权资源访问。

添加收藏路径

登录BOS管理控制台。

在左侧导航栏，单击【收藏路径】。



点击【添加收藏路径】按钮，弹框中配置以下参数。

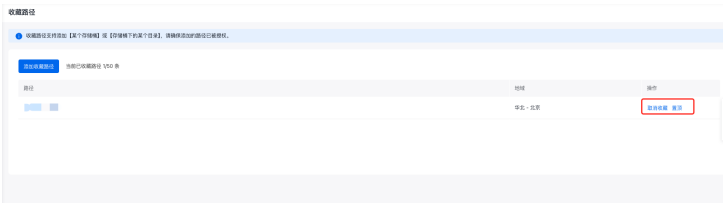
参数	说明
地域	存储桶所在的地域
Bucket	存储桶名称，选择当前账号下的存储桶进行配置
文件路径	选填项，仅支持目录级别的路径配置



管理收藏路径

- 文件路径添加之后，单击对应的路径名称，可直接跳转到对应文件路径。

- 多条路径可单击指定路径后的置顶，可置顶该路径；
- 单击取消收藏，可删除该路径。



开发者指南

基本概念

概述

本文将向您介绍对象存储 BOS 产品中涉及的几个基本概念，以便于您更好地理解 BOS 产品。

存储空间（Bucket）

Bucket 是存放数据的容器，可以当成是存储桶。Bucket 具有区域属性，只能位于一个区域。因为 Bucket 名称创建后不能修改名称和所属区域，所以建议根据业务情况就近存储，方便上传和下载以提高访问速度。Bucket 名称具有全局唯一性。每个 Object 必须都包含在某个 Bucket 中。一个用户最多可创建 100 个 Bucket，但每个 Bucket 中存放的 Object 的数量和大小总和没有限制，用户不需要考虑数据的可扩展性。

文件/对象（Object）

在 BOS 中，用户操作的基本数据单元是 Object。每个 Object 包含 Key、Meta 和 Data。其中，Key 是 Object 的名字；Meta 是用户对该 Object 的描述，由一系列Name-Value 对组成；Data 是 Object 的数据。

访问域名（Endpoint）

百度智能云目前开通了多区域支持，请参考区域选择说明。 BOS 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。 Region 表示 BOS 数据中心所在的区域，EndPoint 表示 BOS 对外服务的访问域名，下面介绍 Region 与 EndPoint 的对应关系,地域信息持续更新，更多信息请参考Region与Endpoint。

区域	访问EndPoint	支持协议
华北-北京	bj.bcebos.com	HTTP,HTTPS
华北-保定	bd.bcebos.com	HTTP,HTTPS
华东-苏州	su.bcebos.com	HTTP,HTTPS
华南-广州	gz.bcebos.com	HTTP,HTTPS
西南-成都	cd.bcebos.com	HTTP,HTTPS
中国香港	hkg.bcebos.com	HTTP,HTTPS
华中-武汉	fwh.bcebos.com	HTTP,HTTPS
华东金融-上海	fsh.bcebos.com	HTTP,HTTPS

> **说明：**
>
> - BOS API 支持 path-style 和 bucket virtual hosting 两种风格的请求方式，推荐使用 bucket virtual hosting 的请求风格，即 Bucket + EndPoint 的形式，以北京区域名为 bos-example 为例，则 bucket virtual hosting 风格的请求为 bos-example.bj.bcebos.com.
> - 2024年6月1日起，新创建的存储桶，使用官方域名时，不支持使用 path-style 域名，默认仅支持 virtual-hosted-style 域名。2024年6月1日之前所创建的存储桶不受影响。详见 BOS 存储桶域名请求风格更新。

以 GetObject 为例，path-style 风格和 bucket virtual hosting 风格的请求语法如下：

风格	请求语法
bucket virtual hosting(三级域名)	GET /object HTTP/1.1 Host: bucket.bj.bcebos.com
Path-style	GET /bucket/object HTTP/1.1 Host: bj.bcebos.com

BOS 支持自定义域名访问，如果用户为 Bucket 绑定了自定义域名，就可以通过自定义的URL 来调用 BOS API。例如用户为 mybucket 绑定了自定义域名 abc.com，则当用户访问 abc.com/mypic.jpg 时，实际上变成访问 mybucket.bj.bcebos.com/mypic.jpg ，对应的 bucket virtual hosting 请求语法为

```
 GET /mypic.jpg HTTP/1.1
Host: abc.com
```

关于如何为 Bucket 绑定自定义域名请参见创建 Bucket 自定义域名。

访问密钥（AK/SK）

用户开通 BOS 服务后，系统会自动分配一对 Access Key ID (AK)/ Secret Access Key(SK)，该密钥对将在用户向BOS发起请求时用做签名验证。Access Key ID用于标示用户，Secret Access Key 是用户用于加密签名字符串和 BOS 服务用来验证签名字符串的密钥。请参考获取AK/SK。

除系统分配的密钥对外，用户还可自行申请多对 AK/SK，总量最多为 20 对。

BOS基本术语表

英文	中文
Bucket	存储空间
Object	对象
Region	地域
AZ(Availability Zone)	可用区
AccessKey	Access Key ID / Secret Access Key 的统称，访问密钥
ACL(Access Control List)	访问控制列表
Standard	标准存储类型
Infrequent Access	低频存储类型
Cold	冷存储类型
Archive	归档存储类型
key	对象的名字，用于唯一标识存储空间中的对象
Metadata	元信息，用来记录一个对象的长度、类型等信息
Part	文件分块上传过程中产生的碎片文件
Prefix	前缀，用来指定文件存放路径
LifeCycle	文件生命周期管理功能
Replication	数据同步功能，用于在两个不同的存储空间之间建立自动同步关系

开发准备

概述

百度智能云对象存储 BOS（Baidu Object Storage），提供稳定、安全、高效以及高扩展存储服务，支持单文件最大 5TB 的文本、多媒体、二进制等任何类型的数据存储。数据多地域跨集群的存储，以实现资源统一利用，降低使用难度，提高工作效率。

BOS 提供标准的 REST 接口，可与任何 Internet 开发工具包搭配使用，并且提供多语言的 SDK。本开发者指南面向开发用户，主要讲解 BOS 的核心概念（如 Bucket 和 Object ）以及如何使用 BOS 的 API 和 SDK 来使用这些资源。

在使用 BCE API 时，需要进行签名认证，请参考认证机制。

资源对照表

在使用百度智能云对象存储 BOS 服务时，您可能会用到以下资源：

资源	描述
百度智能云对象存储BOS开发者指南	本文档从开发者角度为您讲解百度智能云BOS的所有功能以及如何使用百度智能云BOS服务。
百度智能云对象存储BOS控制台操作指南	本文档介绍了在控制台上操作百度智能云BOS的具体流程。
百度智能云对象存储BOS典型实践	本文档介绍了百度智能云BOS的几个代表性的使用场景以及在这些场景下的配置实践。
百度智能云对象存储BOS API参考	本文档介绍了百度智能云BOS支持的RESTful API操作和相关的示例。
百度智能云对象存储 BOS SDK	本文档介绍了主流语言的 SDK 开发操作和参数。
百度智能云对象存储 BOS 周边工具	本文档介绍了几种由百度智能云 BOS 官方推出的管理工具。
百度智能云对象存储 BOS 视频专区	介绍百度智能云BOS的基本概念和功能，并提供基于BOS管理控制台的操作演示。

Region与Endpoint

region 与Endpoint信息

BOS 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。 Region 表示 BOS 数据中心所在的区域，EndPoint 表示 BOS 对外服务的访问域名，更多概念请参考 基本概念。

下面介绍 Region 与 EndPoint 的对应关系：

区域	Region	访问EndPoint	支持协议
华北-北京	bj	bj.bcebos.com	HTTP,HTTPS
华北-保定	bd	bd.bcebos.com	HTTP,HTTPS
华东-苏州	su	su.bcebos.com	HTTP,HTTPS
华南-广州	gz	gz.bcebos.com	HTTP,HTTPS
西南-成都	cd	cd.bcebos.com	HTTP,HTTPS
阳泉	yq	yq.bcebos.com	HTTP,HTTPS
中国香港	hkg	hkg.bcebos.com	HTTP,HTTPS
华中-武汉	fwh	fwh.bcebos.com	HTTP,HTTPS
华东-上海	fsh	fsh.bcebos.com	HTTP,HTTPS

若您在地域选择上存在困惑，请参考 区域选择说明。

Bucket基本操作

创建Bucket

概述

在 BOS 中，用户操作的基本数据单元是 Object，每个 Object 必须都包含在某个存储空间 (Bucket) 中。所以用户要存储数据，必须要创建 Bucket。

操作方式

您可以通过 Console 控制台、 SDK 和 API 来创建 Bucket，具体如下：

- 通过 Console 控制台创建 Bucket请参考创建 Bucket。

- 通过 SDK 创建 Bucket：
  - Java SDK
  - Python SDK
  - PHP SDK
  - C# SDK
  - JavaScript SDK
  - Android SDK
  - iOS SDK
  - Ruby SDK
  - GO SDK
  - C++ SDK
- 通过 API 创建 Bucket：
  - PutBucket

注意事项

- 每个用户最多创建 100 个 Bucket。
- Bucket 具有区域属性，只能位于一个区域。因为 Bucket 名称创建后不能修改名称和所属区域，所以建议根据业务情况就近存储，方便上传和下载以提高访问速度。
- 每个 Bucket 的名称具有全局唯一性。可使用前缀确保名称的唯一性，如以所在的组织的名称作为 Bucket 的前缀。Bucket 创建后，不能更改名称。
- 在 Bucket 创建时即可设置相应的权限，包括：私有、公共读和公共读写三种权限。若需要设置高级权限，则请先创建好 Bucket 再去对应的详情页面中设置，具体内容请参考更新Bucket权限
- 创建 Bucket 时默认开启 CDN 加速，如果需要关闭，可直接在创建时将 “CDN 官方加速”设为 OFF 状态。
- 如您开通了多区域服务，请根据需要选择 Bucket 所在区域，请参考区域选择说明。
- 创建 Bucket 时可以选择 Bucket 的默认存储类型，如果用户使用 API、CLI 或者 SDK 上传的 Object 未指定存储类型，则继承 Bucket 的默认存储类型。在Console 控制台上，上传的 Object 默认为标准存储。如果 Object 的存储类型和Bucket 存储类型不一致时，以 Object 的存储类型为准。存储类型包含标准存储、低频存储、冷存储和归档存储四种，具体使用场景和性能请参见分级存储。

获取Bucket地域信息

概述

您可以通过BOS API的getBucketLocation接口获取存储空间（Bucket）所属的地域，即数据中心的物理位置信息。百度智能云支持的各区域详细信息可参见区域选择说明。

操作方式

用户可以通过控制台查看 Bucket 的区域信息，同时 BOS 也支持使用 API 与使用 SDK 获取 Bucket 的 region 信息，具体如下：

- 通过控制台查看 Bucket 所属的区域：
  1. 登录 对象存储 BOS 管理控制台。
  2. 您可以在控制台页面左侧“存储管理”中点击您想查看的 Bucket，在右侧 Bucket 详细信息的标题栏，即可查看 Bucket 区域信息。或者鼠标悬浮于您想查看的 Bucket，自动弹出区域信息。
- 通过 API 接口查看 Bucket 所属的区域：
  - GetBucketLocation
- 通过 SDK 查看 Bucket 所属的区域：
  - Java SDK
  - Python SDK
  - PHP SDK
  - Ruby SDK
  - GO SDK
  - C++ SDK

示例

下面是使用Java SDK的代码示例：

```
 BosClient client = new BosClient(config);
ListBucketsResponse listBucketsResponse =  client.listBuckets();
List<BucketSummary> bucketSummaryList =  listBucketsResponse.getBuckets();
for(BucketSummary bs : bucketSummaryList) {
    System.out.println( bs.getLocation());
}
System.out.println(client.getBucketLocation("bucket-test").getLocationConstraint());
```

查看Bucket列表

基本介绍

用户创建Bucket之后，用户可以查看所拥有的所有Bucket列表。

操作方式

您可以通过 Console 控制台、 SDK 和 API 查看 Bucket 列表，具体如下：

- 通过控制台查看 Bucket 列表：
  1. 登录 对象存储 BOS 管理控制台。

2. 您可以在控制台页面左侧“存储管理”查看拥有的 Bucket。

- 通过 SDK 查看 Bucket 列表：
  - Java SDK
  - Python SDK
  - PHP SDK
  - C# SDK
  - JavaScript SDK
  - Android SDK
  - iOS SDK
  - Ruby SDK
  - GO SDK
  - C++ SDK
- 通过 API 查看 Bucket 列表：
  - ListBuckets

示例

下面是使用Java SDK的代码示例：

```
public void listBuckets (BosClient client) {
  // 获取用户的Bucket列表
  List<BucketSummary> buckets = client.listBuckets().getBuckets();
  // 遍历Bucket
  for (BucketSummary bucket : buckets) {
     System.out.println(bucket.getName());
  }
}
```

设置跨域资源共享

概述

跨域访问是指发起请求的资源所在域不同于该请求所指向的资源所在域，出于安全考虑，浏览器会限制这种非同源的访问。而在实际应用中，会经常遇到跨域访问的请求，例如，用户的网站A（http://domainA.example）后端使用了BOS存储，用户想在该网站的Web应用程序中引用存储在BOS上的资源，但该页面只能向网站域名http://domainA.example发起请求，向其他网站发送的请求均会被浏览器限制。

跨域资源共享（Cross-Origin Resource Sharing），简称CORS，是HTML5提供的标准跨域解决方案，BOS目前已经支持CORS标准来实现跨域访问。具体的CORS规则可以参考W3C CORS规范。CORS通过HTTP请求中附带Origin的Header来表明自己来源域，比如上面那个例子，Origin的Header就是http://domainA.example。服务器端接收到这个请求之后，会根据一定的规则判断是否允许该来源域的请求，如果允许的话，服务器在返回的响应中会附带上Access-Control-Allow-Origin这个Header，内容为http://domainA.example来表示允许该次跨域访问。如果服务器允许来自任何域名的跨域请求，则可将Access-Control-Allow-Origin的Header设置为 * 即可，浏览器根据是否返回了对应的Header来决定该跨域请求是否成功，如果没有附加对应的Header，浏览器将会拦截该返回结果。如果是非简单请求，浏览器会先发送一个OPTIONS请求（预请求）来探测这个跨域请求对于服务器端是不是安全可接受的，如果服务器不支持接下来的操作，浏览器会拦截该跨域请求。

操作方式

BOS 为开发者提供了两种方式来配置 bucket 资源的跨域访问权限，一种是直接在 BOS 控制台对 bucket 进行 CORS 规则设置；另一种是调用 CORS 相关的 API 接口来控制 bucket 资源的访问权限。

- 通过 Console 控制台配置 bucket 资源的跨域访问权限请参考设置跨域访问。
- 通过 API 配置 bucket 资源的跨域访问权限：
  - PutBucketCors: 用来在指定的 Bucket 上设定一个跨域资源共享（CORS）的规则，如果原规则存在则覆盖原规则。
  - GetBucketCors: 用于获取指定的 Bucket 当前的 CORS 规则。
  - DeleteBucketCors: 用于关闭指定 Bucket 的 CORS 功能并清空所有规则。
  - OPTIONS Object: 浏览器在发送跨域请求之前会发送一个 preflight 请求（OPTIONS）并带上特定的来源域，HTTP 方法和 Header 信息等给 BOS 以决定是否发送真正的请求，此接口即响应这种请求。

> **注意：**
>
> - BOS 中 CORS 配置是在 Bucket 级别的；
> - CORS 请求是否通过和 BOS 的身份验证是完全独立的，因为 CORS 规则仅仅是用来决定是否附加 CORS 相关的 Header 的一个规则，是否拦截该请求完全由浏览器决定。

注意事项

- CORS只针对浏览器环境有意义，相关的Header附加由浏览器自动完成，不需要用户手动操作。
- CORS请求与BOS的身份验证是完全独立的，即CORS规则仅仅是用来决定是否附加CORS相关的Header的一个规则，是否拦截该请求完全由浏览器决定。
- 使用跨域请求的时候需要关注浏览器是否开启了Cache功能。当运行在同一个浏览器上分别来源于不同域 http://domainA.example 和 http://domainB.example 的两个页面都同时请求同一个跨域资源的时候，如果http://domainA.example的请求先到达服务器，服务器将资源带上Access-Control-Allow-Origin的Header为 http://domainA.example 返回给用户。这个时候 http://domainB.example 又发起了请求，浏览器会将Cache的上一次请求结果返回给用户，这个时候Header的内容和CORS的要求不匹配，就会导致后面的请求失败。

管理生命周期

概述

存放在 BOS 中的文件通常会发生归档下沉、删除等涉及到文件生命周期的操作。一般情况下，文件在新建后的短期内会被频繁读取访问，随着时间的推移，该文件的读取次数将变少，进而变成"冷文件"，即不再被频繁的访问。到最后，该文件将会被最终删除。用户如果手工维护数据的生命周期，则费时费力；但如果不去维护，则数据始终存放在标准存储里会产生不菲的费用。因此，BOS 提供生命周期管理功能，以帮助用户自动化地完成数据的生命周期管理，实现数据从创建到归档到删除的自动管理流程，从而节约人力和存储费用。

文件生命周期管理方式

- 为了使数据管理更加便捷与智能，我们在保持【基础生命周期管理】（原生命周期管理）的基础上，又支持了【智能分层】，智能分层不仅可根据文件访问频次进行自动沉降，也可根据文件的访问情况进行智能冷热转化，自动升级至标准或低频存储，该功能提供了数据存储的成本与读写性能最优解决方案。
- 在基础生命周期中在基础基于前缀进行细化管理的能力上，又扩展支持设置对象标签功能(Tag)、目录排除(Not)、文件大小过滤(Filesize)、对象多版本的生命周期管理功能，可细化生命周期操作粒度。

智能分层与基础生命周期管理区别

方式	规则配置	适合场景	转化模式	转化存储类型
智能分层	一键开启	访问模式不固定或者无法预估访问模式的数据	支持数据降冷，也可按需回暖	不支持归档与多az存储类型，其余均支持
基础生命周期管理	需自定义管理规则	1.可预估访问模式 2.需要精细管理	单向降冷	全部支持（多版本暂不支持归档）

功能说明

> 存储桶仅支持配置【基础生命周期管理】或【智能分层】，二者不可同时设置。

智能分层

- 存储桶开启智能分层，系统将周期性地监测数据访问次数，在持续一段时间没有数据访问时，将数据转移至存储成本更低的访问层。如果数据重新被访问，则会被重新转移到高频访问层上，保障数据读取性能。
- 支持文件从标准存储-〉低频存储-〉冷存储 或冷存储-〉低频存储-〉标准存储；
- 可配置删除文件与碎片（分片上传任务中，未完成的文件碎片）定期清除，支持按照指定固定天数后执行删除操作， 规则中计算起始时间以 Object 的创建时间为准，删除优先级高于转化，到期会执行删除；
- 配置规则后您可通过控制台存储桶数据分析查看单个存储桶沉降与回暖数据量。

智能分层注意事项

- 不支持转化为归档存储，标准存储-多AZ，低频存储-多AZ；
- 存储桶仅可配置一条规则。

基础生命周期

基础生命周期管理支持如下功能

- 自定义时间换存储类型，从标准存储转低频存储、转冷存储、转归档存储；或从标准存储-多 AZ 到低频存储-多 AZ；
- 定时删除不再需要的数据；
- 清除过期的三步上传数据。

从场景上划分，基础生命周期管理支持两种模式

- 数据达到一定寿命后自动归档：如定义所有创建时间超过30天的数据自动转为存储费用更为低廉的低频存储；
- 数据达到一定寿命后自动删除：如定义所有创建时间超过30天的数据自动删除。

从规则配置上划分，基础生命周期管理支持简单与复杂模式

- 简单模式：定义规则中仅还有指定生效文件范围设置仅配置前缀或存储桶全部文件（不包含多版本对象）；
- 复杂模式：定义规则中包含指定生效文件范围设置包含对象标签、排除某些文件、指定文件大小、对象历史版本中的任意一种配置。 ****复杂生命周期规则管理支持功能****
- 支持设置对象标签，支持指定标签沉降删除Object。
- 支持设置目录排除，排除目录外的Object以及指定对象标签。
- 支持设置文件大小，支持对指定文件大小范围内的object进行沉降删除。
- 设置多版本生命周期， 支持对历史版本进行删除。

****特别提示****

存储桶生命周期管理规则复杂和简单模式，表现在对文件的转化与删除的执行规则的区别。

- 简单模式：对冲突的多条简单规则会按照"精细化执行"的逻辑，例如：规则 1：当设置 a/b/ 目录下文件创建满 180 天删除；规则 2： a/b/c 目录下文件创建满365 天删除,执行结果为a/b/c/1.txt 文件会在创建满 365 后执行删除，而a/b/2.txt 则会在创建满180 后执行删除。
- 复杂模式：对冲突的多条复杂规则会按照"全局最优执行"的逻辑，且删除优先。例如规则 1：当设置 a/b/ 目录下文件（未配置历史版本）创建满 180 天删除；规则 2： a/b/ 目录下文件配置 365 天删除，并且排除 a/b/test目录,执行结果为a/b/c/1.txt 文件会在创建满180 天后执行删除，a/b/test/2.txt 也会在 180 天后删除。
- **特别提示存储桶中若创建一条复杂规则，则存储桶生命周期管理整体应用复杂规则执行**

基础生命周期注意事项

通用注意事项

- 每个 Bucket 可以有至多 1000 条规则;
- BOS 生命周期规则设置后会在一天内生效;
- 规则生效后，BOS 会对符合条件的 Object 进行相应的处理，但处理需要一定的时间，**不一定能马上看到效果**。一般情况下，沉降或删除的时间为几小时，但若沉降数据量较大，则可能会在几天甚至更长的时间完成下沉或者删除；

- 规则中计算的时间（即 Object 的"年龄"）以 Object 的创建时间为准，而不是生命周期规则的创建／修改时间;
- **BOS 只保存文件的最后修改时间，即 last-modified 时间**；如果您不更新 meta 或者覆盖文件，那么 last-modified 就是创建时间。所以生命周期中的"创建时间"其实是 last-modified 时间。
- 基于文件访问时间记录的生效策略，目前仅北京和苏州区域支持。
- 低频存储、冷存储和归档存储的最低存储时间分别为 30 天，60 天和 180 天。您配置的生命周期沉降/删除规则需要满足最低存储时间的要求。若您配置的时间小于最低存储时间时间，控制台将会产生提示，请您根据提示中的要求进行配置。
- **单 AZ 类型文件仅能沉降到单 AZ 类型文件**，无法沉降到多 AZ 类型文件；**标准存储-多AZ 类型文件只能沉降到低频存储-多 AZ 类型文件**。
- 发生生命周期沉降时，原类型若需要取回，则会产生取回费用。同时，沉降完成后，原类型文件会被删除，也会产生请求费用。比如，一个低频存储类型文件沉降到冷存储，那么 BOS 需要先获取原低频存储文件，该操作产生取回费用；当文件成功沉降为冷存储文件后，原低频存储文件会被删除，该操作产生 Delete 请求费用，该请求费用包含在写请求费用中。
-

包含复杂生命周期规则注意事项

复杂生命周期规则注意事项

- 多版本生命周期的历史版本的判定时间是成为历史版本的时间，而不是其元数据的修改时间。
- 多版本的生命周期不支持归档类型
- tag的规则设置: 每个规则最多有10个对象标签，每个key长度小于等于128， value长度小于等于256。
- 每个rule下只有1个排除目录, 开启Not选项时，前缀和标签必须至少存在一项，即同时设置前缀和标签或者只设置前缀或标签，Not中的前缀或标签只有一个；排除前缀必须位于设置前缀之下，并和前缀的结构形式一样，例如前缀是 bucket/prefix，not prefix可以是bucket/prefix1
- 指定最小文件指定生命周期规则作用的最小文件大小，最小文件大小要求大于等于0字节，小于5TB。
- 指定最大文件指定生命周期规则作用的最大文件大小，最大文件大小要求大于0字节，小于等于5TB。
- 基础生命周期语义和复杂生命周期语义有一定的区别， 对于基础生命周期语义， 按照的是最长前缀的原则， 如果命中最长前缀，那么就不会去寻找其他规则。 复杂生命周期语义会寻找全局最优规则， 寻找符合prefix, not, filesize, tag的规则下最早执行的规则。

配置方式

- 通过 Console 控制台管理生命周期请参考管理生命周期。
- 通过 API 管理生命周期：
  - PutBucketlifecycle
  - GetBucketLifeCycle
  - DeleteBucketLifeCycle

下面重点介绍一下通过 API 管理生命周期时的规则文件语法。

规则定义

BOS提供的生命周期管理服务通过定义规则，并为每条规则设定动作来实现。

> **注意：**
>
> - BOS基础生命周期是天级别执行；
> - BOS基础生命周期规则设置后会在一天内执行，但是执行时规则不一定满足，最坏情况下，需要两天。
> - 规则生效后，BOS会对符合条件的Object进行相应的处理，但处理需要一定的时间(一般情况下为几小时)，所以设置规则后不一定能马上看到效果。
> - 规则中计算的时间（即Object的"年龄"）以Object的创建时间为准，而不是生命周期规则的创建／修改时间。
> - BOS只保存文件的最后修改时间，即last-modified时间；如果用户不更新meta或者覆盖文件，那么last-modified就是创建时间。所以生命周期中的"创建时间"其实是last-modified时间。

规则定义遵循以下原则：

- 规则基于bucket定义；
- 基础生命周期管理每个bucket可以有至多1000条规则，智能分层仅支持配置一条，且二者不可同时配置；

以下是一个基础生命周期规则示例，该规则附属于名为samplebucket的bucket，包含3条规则项：

- 2016年9月7日后删除prefix1下的所有文件；
- 自动将prefix2下创建超过30天的文件转入低频存储；
- 自动清除prefix3下超过5天未完成的三步上传。

```
  {
"rule": [
  {
    "id": "sample-rule-delete-prefix",
    "status": "enabled",
    "resource": [
      "samplebucket/prefix1/*"
    ],
    "condition": {
      "time": {
        "dateGreaterThan": "2016-09-07T00:00:00Z"
      }
    },
    "action": {
      "name": "DeleteObject"
    }
  },
  {
    "id": "sample-rule-transition-prefix",
    "status": "enabled",
    "resource": [
      "samplebucket/prefix2/*"
    ],
    "condition": {
      "time": {
        "dateGreaterThan": "$(lastModified)+P30D"
      }
    },
    "action": {
      "name": "Transition",
      "storageClass": "STANDARD_IA"
    }
  },
  {
    "id": "sample-rule-abort-multiupload-prefix",
    "status": "enabled",
    "resource": [
      "samplebucket/prefix3/*"
    ],
    "condition": {
      "time": {
        "dateGreaterThan": "$(lastModified)+P5D"
      }
    },
    "action": {
      "name": "AbortMultipartUpload"
    }
  }
]
}
```

以下是一个复杂生命周期的配置规则, 包含两个配置

- 最新版本为在prefix/目录下且不在prefix/prefix1/下，tag为'key1':'value1'或者'key2':'value2' 并且tag不包含'key3':'value3'的, 同时满足大小为1024字节到2048字节的object， 并且object在超过3天没有修改的object会被删除，同时这个目录下最新版本的object如果只有删除标记将会在下一轮的生命周期中被清除。
- 历史版本为在整个目录下，tag为'key1':'value1'或者'key2':'value2'的成为历史版本超过5天的object会被删除。

```
{
 "rule": [{
 "id": "rule-id",
 "status": "enabled",
 "ExpiredObjectDeleteMarker" :"true",
 "resource": [
    "bucket/prefix/*"
 ],
 "condition": {
    "time": {
       "dateGreaterThan": "$(lastModified)+P3D"
    },
    "objectSize":{
       "minSize":1024,
       "maxSize":2048
    },
    "tag": {
       "key1":"value1",
       "key2":"value2"
    }
 },
 "action": {
    "name": "DeleteObject"
 },
 "not":{ // 一个规则只有一个not prefix+ not tag 其中prefix和tag 必须存在一项
    "resource": "bucket/prefix/prefix1*",
    "tag": {"key3":"value3"}
 }
},{
 "id": "rule-id2",
 "status": "enabled",
 "resource": [
    "bucket/*"
 ],
 "condition": {
    "time": {
       "dateGreaterThan": "$(lastModified)+P5D"
    },
    "tag": {
       "key1":"value1",
       "key2":"value2"
    }
 },
 "action": {
    "name": "NonCurrentVersionDeleteObject"
 }

 }]
}
```

每条规则由id、status、resource、condition和action组成，详细解释参见下表。

规则项	描述	是否必填	备注
id	规则的标识符。	可选	同一个bucket内规则id必须唯一，不能重复。如果用户不填系统会自动帮用户生成一个。
status	规则的状态。	必填	取值为enabled或disabled，当规则处于disabled时规则不生效。
resource	规则对哪些资源生效。	必填	举例：对samplebucket里以prefix/为前缀的Object生效：samplebucket/prefix/*；对samplebucket里所有Object生效：samplebucket/*。
condition	规则依赖的条件。	必填	目前只支持time形式。
+time	时间限制条件。	必填	通过定义的dateGreaterThan实现。
++dateGreaterThan	描述时间关系。	必填	支持绝对时间date和相对时间days。绝对时间date格式为yyyy-mm-ddThh:mm:ssZ，eg. 2016-09-07T00:00:00Z。绝对时间为UTC时间, 必须以00:00:00(UTC 0点)结尾；相对时间days的描述遵循ISO8601, 支持的最小时间粒度为天，如:$(lastModified)+P7D表示的时间为object的last-modified之后7天。
action	对resource执行的操作动作。	必填	-
+name	执行的操作名称。	必填	取值为Transition、DeleteObject、AbortMultipartU、NonCurrentVersionDeleteObject、NonCurre。
+storageClass	Object的存储类型	可选	action为Transition时可以设定，取值为STANDARD_IA，表示从标准类型转为低频存储类型, 取值为COLD，表示从标准类型转为冷存储类型,取值为ARCHIVE表示从标准类型转为归档存储类型。

action

BOS支持Transition、DeleteObject、AbortMultipartUpload、NonCurrentVersionDeleteObject、NonCurrentVersionTransition五种操作，五种动作的详细解释如下：

操作	描述	备注
Transition	将resource转换存储类型。	- 文件在标准存储中存储时长必须大于7天才会被转为低频存储。因为文件创建后的7天通常为频繁访问期，转换成低频可能会增加成本。文件在低频存储中保存时间大于30天才会被转为冷存储, 因为在低频存储可能还会有低频度的访问需求, 如果转换成冷存储可能会增加成本 - 通过生命周期转成低频时，文件的last-modified不会更改。
DeleteObject	将resource删除。	根据设定的规则删除resource。低频存储有30天的最低保存时间，所以用户如果创建规则去删除一个存储时长不到30天的低频文件，会产生额外花费。
AbortMultipartUpload	清除一段时间内还没有完成的三步上传。	需要设置放弃分块上传的时间days。
NonCurrentVersionTransition	将历史版本resource转换存储类型。	- 文件在标准存储中存储时长必须大于7天才会被转为低频存储。因为文件创建后的7天通常为频繁访问期，转换成低频可能会增加成本。文件在低频存储中保存时间大于30天才会被转为冷存储, 因为在低频存储可能还会有低频度的访问需求, 如果转换成冷存储可能会增加成本 - 通过生命周期转成低频时，文件的last-modified不会更改。
NonCurrentVersionDeleteObject	将历史版本resource删除。	根据设定的规则删除resource。低频存储有30天的最低保存时间，所以用户如果创建规则去删除一个存储时长不到30天的低频文件，会产生额外花费。

dateGreaterThan

BOS支持绝对时间date和相对时间days两种形式的dateGreaterThan设置，注意事项如下：

dateGreaterThan项	描述	支持动作	备注
days	数据达到lastModified+days后执行操作。	支持transition、expiration、abortMultipartUpload三种操作。	- 使用days进行生命周期管理的文件，其操作执行时间是：文件的Last-modified加上days，再取整到下一个UTC午夜零点。例如，一个Object的最后更新时间是UTC的2014年4月12日上午1点，相匹配的规则定义的天数是3天，执行操作的时间是UTC 2014年4月16日0点整。 - 如果之后文件的last-modified发生修改，如meta更新、数据overwrite，那么操作的执行时间也会随之变化。
date	在指定日期执行操作。	支持transition和expiration两种操作。	- 如果指定date，date必须是UTC午夜零点，并且符合形如2014-01-01T00:00:00.000Z的ISO8601格式。 - date规则只会对Last-modified时间早于或等于date的文件生效。 - date规则会在当前时间>=date执行。 - 当date和"标准存储存够7天"冲突时，优先"标准存储存够7天"。

> **说明：**
>
> - 每条规则里只能包含一个action。
> - 每个resource每种action最多只能有一条规则，例如bucket/prefix1/*不能有两条transition规则，但可以有一条transition规则加一条expiration规则。
> - 规则中如果以prefix定义的作用范围有包含关系，对于被包含的prefix以最小的作用范围定义的动作为准。如规则1定义prefix是 p1/p1/ 的Object 10天后delete，规则2定义prefix是 p1/p1/p1/的Object 15天后delete，规则1的prefix包含规则2的prefix，对于prefix为 p1/p1/p1/ 的Object执行15天后delete，对于p1/p1/中除了p1/p1/p1/之外的Object执行10天后delete动作。
> - 如果使用date作为Condition时，如"folder/"为prefix的文件，在30天后转低频，并于2016.8.17号删除。系统会依据规则计算出每个文件执行相应操作的时间，并对时间进行比对。如果文件同一天执行transition和expiration，则直接删除；如果文件删除时间早于转低频时间，也直接删除。
> - 复杂的生命周期规则需要注意的是， 和基础生命周期规则不同，对于被包含的prefix语义，按照时间最小原则执行，如规则1 定义prefix是p1/p1/ 的object 10天后delete， 规则2定义的prefix是p1/p1/p1/的object是13天后删除， 那么会先执行规则1的删除。如果想实现规则2的prefix是13天后删除， 需要在规则1中加入not p1/p1/p1/， 将规则2的目录排除。

设置Bucket标签

概述

百度智能云提供 Bucket 标签管理功能，通过给每个 Bucket 添加标签，从而快速分类和识别管理这些资源。

操作方式

BOS 为开发者提供了在 BOS 控制台对 Bucket 进行标签设置的方法。

- 您可以通过 Console 控制台管理 Bucket 标签请参考管理Bucket标签。

注意事项

- 每个用户最多可以创建200个标签。
- 每个标签由键和值两部分组成，标签（键+值）唯一。

删除Bucket

概述

由于一个用户可创建的 Bucket 有配额限制（目前为100个），当某个 Bucket 不再使用时，用户可通过删除Bucket释放配额。

注意事项

- 删除 Bucket 可能会影响相关用户和服务，请谨慎删除。

- 只有当前 Bucket 为空时，即不包含任何 Object 和未完成的三步上传 Part，才可删除 Bucket，否则会出现相应提示。如果需要直接删除非空 Bucket，可以使用 CLI工具的强制删除命令：`bce bos rb bos:/<bucket-name> --force`。

操作方式

您可以通过 Console 控制台、 SDK 和 API 来删除 Bucket，具体如下：

- 通过 Console 控制台删除 Bucket 请参考 删除 Bucket。
- 通过 API 删除 Bucket：
  - 使用 DeleteBucket API 接口删除 Bucket
- 使用 SDK 删除 Bucket：
  - Java SDK
  - Python SDK
  - PHP SDK
  - C# SDK
  - Android SDK
  - iOS SDK
  - JavaScript SDK
  - GO SDK
  - Ruby SDK

示例

下面是使用 Java SDK 的代码示例：

```
public void deleteBucket (BosClient client, String bucketName) {
  // 删除Bucket
  client.deleteBucket(bucketName);
}
```

设置访问日志

概述

当用户需要跟踪访问 BOS 的请求时可以对 Bucket 开启访问日志功能。每条访问日志记录了单个访问请求的详细信息，包括请求者、Bucket 名称、请求时间和请求操作等。日志记录功能可以应用在访问统计和安全审核等方面。当一个 Bucket 开启访问日志功能后，会自动将对这个 Bucket 的访问请求，以小时为单位，按照固定的命名规则，生成日志文件写入用户指定的 Bucket。

> **说明：**访问日志存储到目标 Bucket 是一个"尽力而为"的过程。大多数情况下，一个访问请求从发生，到对应日志存储到目标 Bucket 会在 2-3 个小时内完成。但受到网络环境等因素影响，有的日志可能会延迟较长时间才能存储到目标 Bucket。

操作方式

您可以通过 Console 控制台和 API 来设置访问日志，具体如下：

- 通过 Console 控制台设置访问日志请参考设置访问日志。
- 通过 API 创建设置访问日志：
  - PutBucketLogging
  - GetBucketLogging
  - DeleteBucketLogging

日志命名规则

存储访问日志记录的Object命名规则为`<TargetPrefix><SourceBucketName>-YYYY-mm-DD-HH-MM-SS-UniqueString`，例如mylog/mybucket-2016-05-17-10-00-00-00000。其中

- TargetPrefix为用户指定的前缀，即示例中的mylog/；
- SourceBucketName为源Bucket名称，即示例中的mybucket；
- YYYY、mm、DD、HH、MM和SS分别表示该日志所记录访问请求中的最小时间戳对应的年、月、日、小时、分钟和秒；
- UniqueString为BOS系统生成的字符串。用于唯一标识这个日志文件，即示例中的00000。

日志格式

输出字段	例子	说明
Time	2016-09-05T09:02:03+08:00	发起请求的时间
Bucket Owner	e13b12d0131b4c8bae959df4969387b8	Bucket的Owner ID
Bucket	samplebucket	请求访问的Bucket名字
Key	dir/sample.jpg	请求访问的Object名字
Reserve	-	保留字段
Remote IP	111.30.20.12	请求者IP地址
Requester	e13b12d0131b4c8bae959df4969387b8	请求者的ID
Request ID	a94dbe12-717e-4e72-bcf2-0aa4a3100360	用于唯一标志该请求的UUID，同Header中的x-bce-request-id
Operation	GetObject	请求对应的API
Request Host	bj.bcebos.com	HTTP请求的Host请求头
Request URI	"GET/v1/tables.xml"	HTTP请求中的URI，需要用双引号引用
HTTP Status	200	BOS返回的HTTP状态码
Error Code	InvalidURI	BOS返回的错误码
Bytes Send	2314	用户从BOS下载的流量
Object Size	1024	Object大小，单位为Bytes
Total Time(ms)	11	从开始接收请求到相应返回的总时长，单位为毫秒
Server Cost Time (ms)	10	从接收完所有请求到开始响应用户请求的总时长，单位为毫秒。目前该字段取值为-
Referer	<https://cloud.baidu.com/product/bos.html>	HTTP请求的Referer请求头
User Agent	"curl/7.33.0"	HTTP请求的User Agent请求头
Storage Class	STANDARD	对象的存储类型
Multiple Delete	"abc%0Adef"	Multiple Delete删除的对象信息,'%0A'为分隔符，用于分隔多个对象
vip	10.0.0.100	用户请求的vip
rename_src	"test"	rename 源object
ak	12342556	用户的ak
source	cdn	cdn标示cdn回源请求 -不是CDN回源
mirror_type	302	302表示是重定向的镜像回源请求。fetch表示是镜像抓取的镜像回源请求。-表示请求不是镜像回源请求。
copy_src	STANDARD	copy的源存储类型

日志样例：

```
2016-09-05T09:02:03+08:00 e13b12d0131b4c8bae959df4969387b8 bos-logging - - 127.0.0.1 e13b12d0131b4c8bae959df4969387b8 98a20094-2002-4950-899d-9347244c71da GetBucket bj.bcebos.com "/bos-logging?delimiter=%2F&maxKeys=1000&prefix=" 200 success 2335 - 10 - "-" "bce-sdk-python/0.8.9/2.7.3 (default, Jun 24 2016, 15:08:42) [GCC 3.4.5 20051201 (Red Hat 3.4.5-2)]/linux2" - - - - 123 cdn 302 STANDARD
```

> **说明：**
>
> - 日志中各个字段由空格分隔。
> - BOS的Log中，某些字段可能存在空格，这些字段（如 “Referrer” ）在生成的时候，外部通过引号包括。
> - Reserve字段为保留字段，Log中用 “-”表示。
> - BOS以小时为单位生成Bucket访问的Log文件，但并不表示这个小时的所有请求都记录在这个小时的Log文件内，也有可能出现在上一个或者下一个Log文件中。
> - BOS生成Log文件会占用目标Bucket的存储空间。日志访问功能是免费的，但日志所占用的存储空间还是照常计费。用户可以像操作普通Object一样操作日志文件。
> - 日志文件在目标Bucket生成后，BOS不会主动清理这些日志文件。 设置日志存储的目标存储桶必须与源桶同region，可以投递至源桶。

数据发布

概述

BOS 支持为已创建的 Bucket 设置多种域名，从而对存储的数据进行发布。

操作方式

您可以通过 Console 控制台、SDK 和 API 来对存储的数据进行发布，具体如下：

- 通过 Console 控制台进行数据发布相关操作：
  - 查看 Bucket 官方域名
  - 创建 Bucket 自定义域名
  - 开通 CDN 官方加速域名
  - 配置缓存管理
  - 配置 CDN 上行加速

注意事项

- 自定义域名中不得包括协议、端口和路径，而应只包含域名本身。
- 每个 Bucket 最多可添加并绑定 20 个不同的自定义域名。

Object基本操作

上传数据

简单上传

概述 用户在创建 Bucket 之后，即可上传文件保存在 Bucket 中。用户可以通过 PutObject 接口上传单个小于 5GB 的文件。

操作方式 用户可以通过使用控制台上传单个文件，BOS 也同时支持使用 API 与使用 SDK 来进行单个 Object 的上传，具体如下：

- 使用控制台上传单个文件

  1. 登录管理控制台，进入"产品->云基础->对象存储 BOS "。
  2. 在"存储管理"区域点击您想把文件上传到的 Bucket 。在右侧文件列表页面，点击上传文件按钮。上传文件窗口如下。



  3. 您可以使用拖拽或直接选择文件的方式上传文件，同时在上传时可以选择验证 MD5，修改文件存储类型。

- 使用 PutObject API 接口上传 Object
- 使用 SDK 上传 Object ：
  - Java SDK
  - Python SDK
  - PHP SDK
  - C# SDK
  - Android SDK
  - iOS SDK
  - Ruby SDK
  - GO SDK
  - C++ SDK

示例

下面是使用 Java SDK 的代码示例：

```
public void PutObject(BosClient client, String bucketName, String objectKey){
  // 获取指定文件
  File file = new File("/path/to/file.zip");

  // 以文件形式上传Object
  PutObjectResponse putObjectFromFileResponse = client.putObject(bucketName, objectKey, file);
```

表单上传

概述

用户可以以 HTML 表单的形式上传 Object ，上传的 Object 不能超过 5GB 。

适用场景

网站可以将表单上传嵌入 HTML 页面，直接把用户上传的文件传到 BOS ，不需要在网站服务器保存。

	表单上传	简单上传
Object 上传流程	用户本地->BOS	用户本地->网站服务器->BOS

操作方式

用户可以使用表单上传的 API 将文件上传文件到指定 bucket 。

使用PostObject API接口上传Object。

分块及断点续传

概述

用户可以使用分块上传API来上传最大 48.8TB 的 Object 数据。

适用场景 分块上传和断点续传适用但不限于以下场景：

- 需要支持断点上传。
- 上传超过 5GB 大小的文件。
- 网络条件较差，和 BOS 的服务器之间的连接经常断开。

- 需要流式地上传文件。
- 上传文件之前，无法确定上传文件的大小。

操作方式

BOS支 持以 SDK 方式进行分块上传和断点续传，具体如下：

- 使用 SDK 进行 Object 分块上传
  - Java SDK
  - Python SDK
  - PHP SDK
  - C# SDK
  - Android SDK
  - iOS SDK
  - Ruby SDK
  - GO SDK
  - C++ SDK
  - JavaScript SDK
- 使用 SDK 进行 Object 断点续传
  - Java SDK
  - Python SDK
  - PHP SDK
  - Ruby SDK
  - C++ SDK

追加上传

概述

用户可以上传 Appendable Object，之后可以以追加的方式写文件，上传时 BOS 只上传新的数据部分；而以 PutObject 上传的 Object 是不可追加写数据的。

适用场景

适合上传实时视频、日志等场景。

操作方式

BOS 同时支持使用 API 与使用 SDK 进行追加写文件，具体如下：

使用AppendObject API接口追加写文件

- 使用 SDK 追加 Object：
  - Java SDK
  - Python SDK
  - PHP SDK
  - Android SDK
  - Ruby SDK
  - GO SDK
  - C++ SDK

注意事项

Appendable Object 大小限制为 0~5G；

AppendObject 接口在进行追加写时要求对该 Object 有写权限；

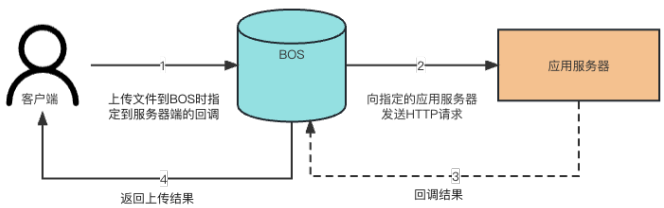
归档类型对象暂时不支持 AppendObject；

上传回调

对象存储BOS在完成文件（Object）上传时可以提供回调（Callback）给应用服务器。您只需要在发送给BOS的请求中携带相应的Callback参数，即可实现回调。

使用场景 上传回调的一种典型应用场景是结合授权第三方上传时使用。适当使用上传回调机制，能有效降低客户端的逻辑复杂度和网络消耗。

限制说明 目前仅普通上传（PutObject）与完成分片上传（CompleteMultipartUpload）操作支持使用上传回调。 **上传回调流程**



1、用户使用客户端上传文件（Object）到BOS时指定到服务器端的回调参数；

2、文件上传成功后，BOS服务器将会向指定的应用服务器发送HTTP请求进行回调，请求中可以包括系统定义的参数（如Bucket名称、Object名称等）和自定义回调参数（例如发起请求的用户ID）信息；

3、应用服务器可以及时得到上传完成的通知，进而完成诸如数据库修改等操作，并向BOS返回上传回调结果；

4、当回调请求接收到服务器端的响应后，BOS服务器再将状态返回给源客户端。

调用方式

使用API

如果您的程序自定义要求较高，您可以直接发起REST API请求。直接发起REST API请求需要手动编写代码计算签名。更多信息，请参见Callback。

下载数据

简单下载

概述

简单下载是通过 BOS API 的 GetObject 接口，下载已经上传的文件（Object），Object 下载是使用 HTTP 的 GET 请求来完成的，该操作需要用户对 Object 所在的 Bucket 具有读权限。

操作方式 BOS 同时支持使用 API 与使用 SDK 工具下载 Object ，具体如下：

- 使用 API 接口获取 Object ：
  - 通过GetObject接口下载Object
- 使用 SDK 获取 Object ：
  - Java SDK
  - Python SDK
  - PHP SDK
  - C# SDK
  - Android SDK

示例

下面是使用 Java SDK 的代码示例，可以简单的下载 Object ：

```
public void getObject(BosClient client, String bucketName, String objectKey)
  throws IOException {

  // 获取Object，返回结果为BosObject对象
  BosObject object = client.getObject(bucketName, objectKey);

  // 获取ObjectMeta
  ObjectMetadata meta = object.getObjectMetadata();

  // 获取Object的输入流
  InputStream objectContent = object.getObjectContent();

  // 处理Object
  ...

  // 关闭流
  objectContent.close();
}
```

断点续传下载

概述

BOS 提供了从 Object 指定的位置开始下载的功能，在下载大的 Object 的时候，可以分多次下载。如果下载中断，重启的时候也可以从上次完成的位置开始继续下载。

和简单上传类似，您也需要对该 Object 有读权限。通过设置参数 Range 来支持断点续传，对于比较大的 Object 建议使用该功能。Range 的定义可参考 HTTP RFC。如果在请求头中使用 Range 参数，则返回消息中会包含整个文件的长度和此次返回的范围。例如：Content-Range: bytes 0–9/44，表示整个文件长度为 44 ，此次返回的范围为 0–9。

注意：

- 对于 Range 读大小为 0 字节的对象，会返回 400 错误， Range 是前闭后闭区间。
- 归档存储类型对象需要先取回才能调用 GetObject 接口。

操作方式

- Java SDK
- Python SDK
- PHP SDK
- C# SDK
- Android SDK

示例

以 Java SDK 为例，通过使用 GetObjectRequest 来指定下载范围，实现更精细化地获取 Object 。如果指定的下载范围是 0 - 100，则返回第 0 到第 100 个字节的数据，包括第 100 个，共 101 字节的数据，即 [0, 100] 。

```
 // 新建GetObjectRequest
GetObjectRequest getObjectRequest = new GetObjectRequest(bucketName, objectKey);

// 获取0~100字节范围内的数据
getObjectRequest.setRange(0, 100);

// 获取Object，返回结果为BosObject对象
BosObject object = client.getObject(getObjectRequest);
```

通过 getObjectRequest 的 setRange 方法可以设置返回 Object 的范围。用户也可以用此功能实现文件的分段下载和断点续传。

管理文件

获取数据元信息

概述

数据元信息（Object Meta）是对上传到 BOS 的文件的属性描述，分为两种：系统定义的 Meta 和 User Meta（用户自定义元信息）。数据元信息可以在各种方式上传时或者拷贝文件时进行设置。

系统定义的 Metadata

每个 Object 都有一组 Metadata 信息，Metadata 用来标识 Object 的属性如日期、内容长度等信息，这些 Metadata 信息用于 Object 管理。

系统定义的 Metadata 也分为用户可修改和用户不可修改两种。其中如 Object 的固有属性如创建日期、内容长度、上次修改时间、MD5 码等为不可修改的 Metadata 。如 Content-Type、Cache-Control、storage-class 等 Metadata，用户在创建 Object 时或者根据需要可以修改这些 Metadata 的值。系统定义的 Metadata 的详细描述参见下表：

名称	类型	描述	是否可被 用户修改
Cache-Control	String	下载 Object 的 Cache 设置，常见的可取值为 private、no-cache、max-age、must-revalidate。	是
Content-Disposition	String	设置浏览器是否下载，可取值为 inline、attachment;"filename=download.txt"。	是
Content-MD5	String	RFC2616 定义的 HTTP 请求内容的 MD5 摘要，可以通过携带该字段来验证保存在 BOS 侧的文件和用户预期的文件是否一致。	否
Content-Length	Long Int	返回 Object 的数据大小。	否
Content-Type	String	Object 的类型及编码方式。	是
Expires	String	设置下载 Object 时的缓存失效时间。	是
ETag	String	Object 的 HTTP 协议实体标签。	否
x-bce-storage-class	String	标准存储返回 STANDARD，低频存储返回 STANDARD_IA, 冷存储返回 COLD ，归档存储返回 ARCHIVE。	否
x-bce-object-type	String	标识 Object 是 Appendable 还是普通的， Appendale 的 Object 可以追加写，普通的不能。归档存储不支持 Appendable。	否
x-bce-tagging-count	Long Int	标识 Object 的对象标签的数量。	否

用户自定义的 Metadata

为了便于用户对 Object 进行更多描述。BOS 中规定所有以 x-bce-meta- 为前缀的参数，则视为用户自定义的 Metadata ，比如 x-bce-meta-tag。一个 Object 可以有若干个用户自定义 Metadata。这些自定义 Metadata 会在下载 GetObject 或者 HeadObject 的时候在 HTTP 头部中返回。

> **说明：**
>
> - PUT 请求头的大小限制为 8KB 。在 PUT 请求标头中，用户定义的 Metadata 的大小限制为 2KB。

操作方式

BOS 支持多种 SDK 获取 Object 的元信息，具体如下：

- Java SDK
- Python SDK
- PHP SDK
- C++ SDK

示例

以 Java SDK 为例

```
 String EndPoint = "bj.bce.bos.com"
public static void GetObjectMeta(BosClient bosClient, String bucketname, String objectkey)
  {
      ObjectMetadata objectMetadata ;
      objectMetadata = bosClient.getObjectMetadata(bucketname, objectkey);
      System.out.println(objectMetadata.getContentType());
      System.out.println(objectMetadata.getUserMetadata());
  }
```

查看文件列表

概述

当创建 Bucket 之后，您可以查看所拥有的 Bucket 列表，还可以从 BOS 中获取指定 Bucket 的 Object 信息列表，BOS 同时支持使用API 与使用 SDK 列出 Bucket 和 Object。

操作方式

- 使用 API 列举 Bucket

ListBucket 接口列举所有 Bucket

- 使用 SDK 列举 Bucket
  - Java SDK
  - Python SDK
  - PHP SDK
  - C# SDK
  - Android SDK
  - iOS SDK
  - JavaScript SDK
  - GO SDK
  - Ruby SDK
- 使用 API 列举 Object

  使用 ListObjects API 接口列出 Object
- 使用 SDK 获取 Object 列表
  - Java SDK
  - Python SDK
  - PHP SDK
  - C# SDK
  - Android SDK

示例

下面是使用 Java SDK 获取 Bucket 列表的代码示例

```
//百度智能云主用户AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建子用户进行API访问或日常运维
String ACCESS_KEY_ID="<yourAccessKeyId>";
String SECRET_ACCESS_KEY= "<yourAccessKeySecret>";
String ENDPOINT="bj.bcebos.com";

//创建BOS实例
BosClientConfiguration config=new BosClientConfiguration();
config.setCredentials(new DefaultBceCredentials(ACCESS_KEY_ID,SECRET_ACCESS_KEY));
config.setEndpoint(ENDPOINT);
BosClient bosClient=new BosClient(config);

// 获取用户的Bucket列表
ListBucketsResponse listing = bosClient.listBuckets();

// 遍历所有Bucekt
for (BucketSummary bucketSummary : listing.getBuckets()) {
    System.out.println("Bucket: " + bucketSummary.getName());
}
```

下面是使用 Java SDK 获取指定 Bucket 中 Object 列表的代码示例

```
// 获取用户指定Bucket的Object列表
ListObjectsResponse listing = bosClient.listObjects(BUCKET_NAME);
// 遍历所有Object
for (BosObjectSummary objectSummary : listing.getContents()) {
    System.out.println("ObjectKey: " + objectSummary.getKey());
}
```

拷贝数据

概述

拷贝数据是指在不改变原有数据内容的情况下，将某个存储空间(Bucket）内的一些数据(Object)复制到另外一个 Bucket 中。其中归档存储类型文件需要先取回，才能进行拷贝操作，并且归档类型不支持更新 meta，您可以通过BOS提供的 CopyObject 接口来实现数据的拷贝。

> **使用场景：**
>
> - 备份 Object：当您需要将指定 Object 备份保存的时候，通过调用 CopyObject 接口，将 Object 拷贝至另外一个 Bucket 中。
> - 重命名 Object：当您需要对Object进行重新命名，通过调用 CopyObject 接口先将原来的 Object 拷贝后重新命名保存，之后删除源Object。
> - 修改 Object 的 Meta 信息：当您仅需要修改某个 Object 的 meta 信息，同样可以通过调用 CopyObject 接口，将 Object 拷贝后保存在同样的位置，即拷贝源数据与目的数据在同一个位置，同时修改 meta 信息，这样 BOS 就会仅更新 Object meta 信息。

操作方式

BOS 同时支持使用 API 与使用 SDK 进行 Object 拷贝，具体如下：

- 使用CopyObject API接口拷贝Object
- 使用SDK拷贝Object
  - Java SDK
  - Python SDK
  - PHP SDK
  - C# SDK

- Android SDK

注意事项

- 您需要有源 Object 的读取权限及目标 Bucket 的读写权限，否则会无法完成操作。
- 不能对追加上传产生的 Object(AppendObject) 进行拷贝。
- 5GB 以下的文件使用简单拷贝，BOS API 接口为 CopyObject。
- 5GB 以上的文件使用分块拷贝，BOS API 接口为 UploadPartCopy。

删除数据

概述

删除数据即删除存储空间(Bucket）中的文件(Object)，BOS 允许您执行如下删除操作：

- 单个删除：删除某个指定的 Object。
- 批量删除：单次请求最多删除 1000 个 Object 。
- 自动删除：如果您需要删除的 Object 数目很多，而且删除的 Object 有一定的规律，例如定期删除某些天之前的 Object ，或者是要清空整个 Bucket，推荐使用生命周期管理来自动删 除 Object 。设置了生命周期规则之后，BOS 会根据规则自动删除到期的 Object，从而极大减少您发送删除请求的次数，提高删除效率。

操作方式

BOS 同时支持使用 API 与 SDK 的方式来删除，具体如下：

- 使用 DeleteObject API 接口删除单个Object
- 使用 DeleteMultipleObjects API 接口批量删除
- 使用 SDK 删除 Object:
  - Java SDK
  - Python SDK
  - PHP SDK
  - C# SDK
  - Android SDK

示例 下面是 Java SDK 删除 Object 的代码示例，具体如下：

删除单个 Object:

```
 public void deleteObject(BosClient client, String bucketName, String objectKey) {
   // 删除Object
   client.deleteObject(bucketName, objectKey);
}
```

批量删除 Object ,可参考如下两种方法删除多个 Object :

```
 // 1、以Json格式的字符串
String jsonObjectKeys = "{\"objects\": [\"+"{\"key\": \"token1.h\"},"+"{\"key\": \"token2.h\"}"+"]}";
DeleteMultipleObjectsRequest request = new DeleteMultipleObjectsRequest();
request.setBucketName("yourBucketName");
request.setJsonDeleteObjects(jsonObjectKeys);
client.deleteMultipleObjects(request);
// 2、用户只需指定指定参数即可
List<String> objectKeys = new ArrayList<String>();
objectKeys.add("object1");
objectKeys.add("object2");
DeleteMultipleObjectsRequest request = new DeleteMultipleObjectsRequest();
request.setBucketName("yourBucketName");
request.setObjectKeys(objectKeys);
DeleteMultipleObjectsResponse response = client.deleteMultipleObjects(request);
```

> **注意：**
>
> - 支持一次请求内最多删除 1000 个 Object。 消息体（body）不超过 2M。 返回的消息体中只包含删除过程中出错的 Object 结果；如果所有 Object 都删除都成功的话，则没有消息体。

Select扫描

概述

SelectObject 接口支持用户对 BOS 中指定格式(CSV/JSON/Parquet)的 object 内容执行 SQL 语句，通过 SQL 这种结构化查询语言对 object 内容进行筛选、分析、过滤之后再返回用户需要的文件内容。

目前用户想对存储在 BOS 的 object 内容进行筛选过滤，只能先通过 GetObject 接口下载单个 object ，然后再在本地对数据分析过滤；而 SelectObject 接口将把筛选过滤工作集成到 BOS 服务层，从而减少客户从 BOS 下载数据的网络带宽和延迟，同时也能节省客户筛选数据消耗的 CPU 和内存等资源，从而降低客户访问BOS中数据所需的应用程序成本。

适用场景

SelectObject 典型的应用场景是和大数据产品结合使用，替换原来处理 BOS 数据的 GetObject 接口，用于提取日志文件指定内容，数据分析筛选等工作。

使用条件

如果想使用 SelectObject 接口筛选 BOS 中的 object，需要满足以下限制条件和细节要求：

1. **支持的文件类型**
   - 仅支持 select UTF-8 编码的 RFC 4180 标准 CSV（包括TSV等类CSV文件） 、Json 文件和Parquet文件；
   - 支持的 CSV 文件的最大行及最大列长度都是 512K；

- 支持 select 的 Json 文件包括 DOCUMENT 和 LINES 两种，DOCUMENT 是指整个文件是单一的 JSON 对象，LINES 表示整个文件由多行的 JSON 对象组成，但整个文件本身并不是一个合法的 JSON 对象，行与行之间以换行分隔符隔开，支持用户指定常见的 \n，\r，\n等行列分隔符；
- 支持 select 标准、低频、冷存储三种存储类别的文件；
- 支持 select SSE-BOS、SSE-KMS、SSE-C三种服务端加密方式加密之后的文件；
- 支持 select 通过 GZIP 方式压缩后的文件，流式解压选取内容返回，GZIP 文件不支持 deflate 格式，支持标准为 RFC1952：Gzip 压缩标准参考。支持检索经过 GZIP 列压缩或 SNAPPY 列压缩的 Parquet文件。

2. **支持的 SQL 语法**

- 目前只支持 SELECT 语法，SQL语句满足`Select field_list From source Where condition Limit number`形式；
- 支持 string、int(64bit)、float(64bit), timestamp，boolean，decimal 数据类型；
- 支持逻辑条件（AND/OR/NOT），算术表达式（+-*/%），比较运算符(>,=,<,>=,<=,!=)，匹配运算符(LIKE，BETWEEN+AND，IN)，空判断(IS NULL/IS NOT NULL)；
- 支持聚合函数(AVG、COUNT、MAX、MIN、SUM)，转换函数 CAST ，别名关键词 AS；
- 仅支持单文件查询，不支持 join、order by、group by、having、offset 等关键词。

3. **SQL 语句限制**

- 单个 SQL 语句最大长度 16K，最大的列数是 1000，列名称最长为 1024，聚合操作(count/avg等)最多 100 个；
- SQL 语句中字符串需要使用单引号包围，标识符使用双引号包围；例如 `SELECT * FROM "ident" = 'str'`，其中 ident 表示数据中的标识符，str 表示具体的字符串值；
- LIMIT 函数优先级高于聚合函数，例如 `Select avg(cast(_1 as int)) from BosObject limit 100` 表示求前 100 个元素的平均值，与 MySQL 语义不同；
- COUNT 函数后只能用 `*`，即 `count(*)`，不允许 count(_1)形式；
- SQL 语句 FROM 之后的 json path 指定的 JSON 节点数据最长为 512K，最大深度为 10 层；
- `[*]`数组通配符只能出现在 SELECT Json 文件时，而且 select 或者 where 后边的表达式中不能有 `[*]` 数组通配符，数组通配符只能出现在 from 后的 json path 中；
- SELECT CSV 文件时，from 关键词之后只能是 BosObject；
- WHERE 语句里不能包含聚合条件，只允许使用逻辑操作符；
- LIKE 语句中，支持最多 5 个 % 通配符，表示 0 或多个任意字符, _表示单个字符；IN 语句中最多支持1024个常量项；
- Select 后的 fields 可以是列名，CSV 列索引(_1, _2等)，或者是聚合函数，例如 AVG(CAST _1 as int),但是不能是单独的CAST _1 as int；field不支持二元表达式；
- select 后边如果有一个 field 是`*`，那就不再允许有其他 field，例如 `select *, _1 from s`这种是不合法的；select 的 field 中聚合函数和单独列名不可单独出现；select 的 field 中所有 alias 别名必须都不一样；
- 如果 json SQL 中存在`key[*]`/`key[1]`这种形式的field或者source，我们会认为这个field是表示select一个数组元素，键是 key；但是如果 SQL field/source 包括`key[a]`这种形式，会被解析成键为`key[a]`，去获取json中对应的value；
- Json 文件和 SQL 中 Key 的匹配是大小写敏感的，比如 select s.Age 和select s.age是不同的；
- `BETWEEN AND` 和 `IN` 关键词用于集合和范围匹配操作时，需要确保 value 属于同一数据类型。

Select 数据容错机制

(一) 处理缺失数据

- csv 文件的某列数据缺失时，如果该列用于 WHERE 之后做条件判断，可以直接认为条件不满足，跳过该行数据；但是如果该缺失列被用于 SELECT 之后做聚合操作时，例如 avg(cast _1 as int)，我们认为聚合一个不存在的列是非法的，应该直接结束并返回相应错误信息；
- json 文件的某个key缺失时，同上；
- csv 文件的列数据缺失或者 json 文件的某个 key 缺失，默认都当做 NULL 处理，也就是可以通过 IS NULL 判断为 true。
- 其它情况：

> 1. 当 json key 或者 csv 列在 WHERE 后边表达式中使用的话，例如…… `where _1 = ''`，…… `where a.b = 'value'`；如果缺失的话，我们都默认当做NULL值处理
> 2. 当 json key 或者 csv 列直接作为 field 出现在 select 之后时，例如`select _1 from……`，`select a.b from……`；如果缺失的话，csv 列应该默认返回空字符串，json key 也是返回空字符串

(二) 处理类型不匹配数据

- csv 文件的列数据类型非法，例如`CAST _1 as INT`，然而 _1 值为非数值字符串导致 cast 失败，如果`CAST _1 as INT`用于WHERE后边做条件判断，可以直接认为条件不满足，跳过该行数据；但是如果该列被用于 SELECT 之后做聚合操作时，例如`avg(cast _1 as int)`，我们认为聚合一个不合法列是非法的，应该直接结束并返回相应错误信息；
- json 文件的某个 key 对应数据类型非法时，同上

注意事项

1. SelectObject 为 CPU 消耗性请求，扫描数据量以 8MB 为单位计算，计费标准详见产品定价-数据处理费用-Select扫描费用。
2. 单个 bucket 限制 SelectObject 请求 QPS 不超过 500。
3. SQL 语句或者文件内容中包含特殊字符串的情况，请使用\转义符进行转义。例如`select "key }\"[" from BosObject.array[*]`表示选取 json 数组 array 中对应路径为`key }"[`的值。
4. 如果涉及到浮点数数值计算或者金钱交易等对数据准确性要求非常高的情况，优先使用 decimal 数据类型，decimal 支持的操作包括算术表达式（+-*/），比较运算符(>,=, <, >=, <=, !=)，匹配运算符(BETWEEN+AND，IN)；同时允许其它数值类型和 decimal 进行运算等操作，默认会将其他类型(例如int，float)提升成 decimal，以便得到准确的结果。
5. CSV 文件中数据都默认 string 类型，Json 不内置 decimal 数据类型；因此如果希望将 csv 的某一列或者 json 的某个 value 作为 decimal 来处理，都必须使用 CAST 函数，例如`cast(_1 as decimal)`、`cast(1.23 as decimal)`、`cast(key as decimal)`。

CSV Object

select 一个 csv object 一般是通过列序号或者列名来选取指定列数据，或者对某些列做聚合操作，例如一个 test.csv 文件内容如下，包含了各种数据类型的列，**csv 默认每一列数据都是字符串**，需要做对应的 CAST 转换操作，列分隔符之间不要留空格：

```
 header1,header2,header3
1,2,3.4
a,b,c
"d","e","f"
true,false,true
2006-01-02 15:04:06,"2006-01-02 16:04:06",2006-01-02 17:04:06
```

常用 SQL 语句

SQL 语句	描述	备注
select * from BosObject limit 100	返回 object 前 100 行数据	-
select header1,header2 from BosObject	返回 object 中名称为 header1，header2 的列	fileHeaderInfo 参数需要是"USE"
select _1,_3 from BosObject where cast(_1 as int) <= cast(_3 as int)	返回 object 第 1 列和第 3 列的整数，满足第 1 列小于或等于第 3 列	需要 _1,_3 表示的列是整型才能被 CAST 转换，否则会因为不满足条件而跳过
select count(*) from BosObject	返回 object 总行数	-
select AVG(cast(_1 AS int)), MAX(cast(_1 AS int)), MIN(cast(_1 AS int)) from BosObject	返回 object 中第一列的平均值，最大值，最小值	每一行的第一列都不能包含非整型字符串，否则会直接失败
select SUM(cast(header1 AS float)) from BosObject WHERE cast(header1 AS float) != 1	返回 object 中所有列名为 header1 且值不等于 1 的和	每一行的 header1 列都不能包含非数值型字符串
select * from BosObject where _1 LIKE '%果_'	返回 object 中_1列形式满足"%果_"的行，例如"苹果树"、"果蔬"满足条件，"苹果"不满足条件	LIKE 操作符后字符串使用单引号
select * from BosObject where cast(_1 AS int) % 3 = 0	返回 object 中 _1 列能被 3 整除的所有行	_1 需要是整形字符串，才能使用 % 操作符
select * from BosObject where cast(_1 AS int) between 1 and 2	返回 object 中 _1 列处于 [1,2] 区间的所有行	_1 需要是整形字符串
select * from BosObject where cast(_1 AS timestamp) NOT IN (cast('2006-01-02 15:04:06' as timestamp), cast('2006-01-03 15:04:06' as timestamp))	返回 object 中 _1 列不在 IN 区间的所有行	_1 需要是日期字符串形式
select * from BosObject where cast(_1 AS int) * cast(_2 AS int) > cast(_3 AS float) + 1	返回 object 中 _1 列形式满足条件表达式计算结果的所有行	_1,_2,_3 需要是满足 CAST 条件的合法字符串形式
SELECT _1,_2 FROM BosObject WHERE cast(_2 as decimal) IN (cast('5.1824349494011866916128' as decimal),cast('5.000000000000001000000000' as decimal))	返回 object 中 _2 列处于 [5.1824349494011866916128,5.000000000000001000000000]区间的所有行的第一列和第二列数据	_2 列被当做 decimal 数据类型进行比较
SELECT MAX(CAST(_3 AS DECIMAL)) FROM BosObject WHERE CAST(_3 AS DECIMAL) >= cast('559062242.92723' as float))	返回 object 中 _3 列数据大于等于 559062242.92723 的最大值	_3 列被当做 decimal 数据类型进行比较

JSON Object select 一个 json object 一般是通过 key 来选取对应的数据，json 文件包括 LINES 和 DOCUMENT 两种，json 文件内容需满足官方标准。

JSON DOCUMENT Object

```
 {"name": "Smith",
"age": 16,
"weight": 65.5,
"org": null,
"projects":
   [
    {"project_name":"project1", "completed":false},
    {"project_name":"project2", "completed":true}
   ]
}
```

JSON LINES Object

```
 {"name": "Smith",
"age": 16,
"org": null,
"projects":
   [
    {"project_name":"project1", "completed":false},
    {"project_name":"project2", "completed":true}
   ]
}
{"name": "charles",
"age": 17,
"org": "baidu",
"weight": 65.5,
"projects":
   [
    {"project_name":"project3", "completed":false},
    {"project_name":"project4", "completed":true}
   ]
}
```

常用 SQL 语句

- **Json path 基础形式**：`field0.field1[n].property1.attributes[*]`表示查找 JSON 文件根节点下的 field 0 节点下 field 1 节点中数组的第 n 个元素，再查找该元素的 property1 中 attributes 数组的全部内容。
- JSON object SQL 同样可以使用聚合函数，逻辑运算，数学运算等；JSON 中 value 自带数据类型，不需要 CAST 转换，除非需要强制解析成 decimal 或者其它类型。

SQL语句	描述	备注
select projects from BosObject where name='Smith'	返回 json 文件中满足 name='Smith' 条件的 projects 元素	
select * from BosObject.projects[*].project_name	返回 json 文件中根节点下的 projects 节点数组的 project_name 字段	
select s.completed from BosObject.projects[1] s where s.project_name='project2'	返回 object 的 projects 数组第一个元素的 completed 字段值，满足 project_name = 'project2'	
select * from BosObject s where s.org IS NULL AND weight is null	返回 json 文件中满足 name 和 weight 都为空的记录	weight 节点不存在也视为 null

Parquet Object

select 一个 Parquet object 一般是通过 key 来选取对应的数据，Parquet是大数据领域流行的一种自解释的列式存储格式，其select使用可以参考json object方式。

错误返回码

1. server 端返回的错误码可能以 http status code 的形式返回，也可能在 End Message 中的 error-code 返回，以何种形式返回ErrorCode 取决于发生具体的错误类型

ErrorCode	描述	HTTP Status Code
AggregateInvalidField	sql 语句聚合函数使用不合法，只能聚合数值列	400
DecompressError	object 解压失败	400
DataOverflowsType	聚合列的结果溢出类型限制	400
FieldNotExist	文件中不存在 SQL 语句中 field 字段对应的内容	400
HeaderNotExist	csv object 中不存在 header 信息	400
InappropriateJson	json object 内容格式不正确	400
InappropriateParquet	parquet object 内容格式不正确	400
InvalidCompressionTypeParameter	SelectObject 请求中 compressionType 参数不合法	400
InvalidExpressionParameter	SelectObject 请求中 expression 参数不合法	400
InvalidExpressionTypeParameter	SelectObject 请求中 expressionType 参数不合法	400
InvalidFileType	select 只支持选取 csv 、json 和 parquet object 内容	400
InvalidJsonTypeParameter	SelectObject 请求中 json type 参数不合法	400
InvalidQuoteFieldsParameter	SelectObject 请求中 quote fields 参数不合法	400
InvalidSelectRequestJsonBody	SelectObject 请求中 json body 不合法	400
InvalidSqlBetweenOperator	sql 语句中 BETWEEN 操作符使用有误，BETWEEN 和 AND 需要同时使用，AND 两侧类型要一致	400
InvalidSqlBinaryExpr	二元操作符使用非法，左右操作数类型不匹配	400
InvalidSqlFields	sql 语句中 SELECT 的 field 不合法，可能存在二元操作符或者其他非法操作	400
InvalidSqlFunction	sql 语句函数使用有误，检查函数参数类型和数目	400
InvalidSqlInOperator	sql 语句中 IN 操作符使用有误，IN 内部值类型要一致	400
InvalidSqlIsOperator	sql 语句中 IS 操作符使用有误，只能和 NULL/NOT NULL 连用	400
InvalidSqlJsonPathDepth	选取的 json object 节点深度不合法，超出 1024 限制或者小于 1	400
InvalidSqlLikeOperator	sql 语句中 LIKE 操作符使用有误	400
InvalidSqlLimitValue	sql 语句中 Limit 字段值不合法，需要为正整数	400
InvalidSqlNotOperator	sql 语句中 NOT 操作符使用有误，只能用于 BETWEEN/IN/LIKE之前，表示否定	400
InvalidSqlSource	sql 语句中 FROM 的 source 不合法，检查 source 格式是否符合要求	400
RecordTooLarge	csv 文件单行记录长度超出 512KB 限制	400
SqlFieldsNumExceedLimit	sql 语句 SELECT 的 field 数目超出限制	400
SqlSourceNumExceedLimit	sql 语句中 FROM 的 source 数目只能有一个	400
SqlSyntaxError	sql 语句不合法，存在语法错误	400

SDK使用示例

目前 BOS Java SDK、BOS GO SDK，还有 BOS Python SDK 都已经支持 SelectObject 接口。

Java SDK示例

```
public void selectCsv(BosClient client, String bucketName, String csvObject) {
    System.out.println("------ select csv object ------");
    SelectObjectRequest request = new SelectObjectRequest(bucketName, csvObject)
        .withSelectType(Constants.SELECT_TYPE_CSV)
        .withExpression("select * from BosObject limit 3")
        .withExpressionType(SelectObjectRequest.ExpressionType.SQL)
        .withInputSerialization(new InputSerialization()
                .withCompressionType("NONE")
                .withFileHeaderInfo("NONE")
                .withRecordDelimiter("\r\n")
                .withFieldDelimiter(",")
                .withQuoteCharacter("\"")
                .withCommentCharacter("#"))
        .withOutputSerialization(new OutputSerialization()
                .withOutputHeader(false)
                .withQuoteFields("ALWAYS")
                .withRecordDelimiter("\n")
                .withFieldDelimiter(",")
                .withQuoteCharacter("\""))
        .withRequestProgress(false);
    SelectObjectResponse response = client.selectObject(request);

    //输出查询结果
    printRecords(reponse.getMessages());
}

public void selectJson(BosClient client, String bucketName, String jsonObject) {
    System.out.println("------ select json object ------");
    SelectObjectRequest request = new SelectObjectRequest(bucketName, jsonkey)
        .withSelectType(Constants.SELECT_TYPE_JSON)
        .withExpression("select * from BosObject where age > 20")
        .withInputSerialization(new InputSerialization()
                .withCompressionType("NONE")
                .withJsonType("LINES"))
        .withOutputSerialization(new OutputSerialization()
                .withRecordDelimiter("\n"))
        .withRequestProgress(false);
    SelectObjectResponse response = client.selectObject(request);

    //输出查询结果
    printRecords(reponse.getMessages());
}

public void selectParquet(BosClient client, String bucketName, String parquetObject) {
    System.out.println("------ select parquet object ------");
    SelectObjectRequest request = new SelectObjectRequest(bucketName, parquetObject)
        .withSelectType(Constants.SELECT_TYPE_PARQUET)
        .withExpression("select * from BosObject where age > 20")
        .withInputSerialization(new InputSerialization()
                .withCompressionType("NONE"))
        .withOutputSerialization(new OutputSerialization()
                .withRecordDelimiter("\n"))
        .withRequestProgress(false);
    SelectObjectResponse response = client.selectObject(request);

    //输出查询结果
    printRecords(reponse.getMessages());
}

public void printRecords(SelectObjectResponse.Messages messages) {
    if (messages == null) {
        return;
    }
    while (messages.hasNext()) {
        SelectObjectResponse.CommonMessage message = messages.next();
        if (message.Type.equals("Records")) {
            for (String record: message.getRecords()) {
                System.out.println(record);
            }
        }
    }
}
```

Golang示例

```
package main
import (
"bufio"
"encoding/binary"
"fmt"
"io"
"strings"
)
import (
"github.com/baidubce/bce-sdk-go/services/bos"
"github.com/baidubce/bce-sdk-go/services/bos/api"
)
func main() {
selectBosObject()
}
func selectBosObject() {
// 初始化BosClient
AK, SK := "ak", "sk"
ENDPOINT := "bj.bcebos.com"
bosClient, _ := bos.NewClient(AK, SK, ENDPOINT)
// 先确保bucket，object已经存在，object满足csv/json文件格式要求
bucket := "select-bucket"
csvObject := "test.csv"
```

```
 fmt.Println("------ select csv object -------")
 csvArgs := &api.SelectObjectArgs{
  SelectType: "csv",
  SelectRequest: &api.SelectObjectRequest{
   Expression:     "c2VsZWN0ICogZnJvbSBCb3NPYmplY3Qgd2hlcmUgY2FzdChfMSBBUyBpbnQpICogY2FzdChfMiBBUyBpbnQpID4gY2FzdChfMyBBUyBmbG9hdCkgKyAx",
   ExpressionType: "SQL",
   InputSerialization: &api.SelectObjectInput{
    CompressionType: "NONE",
    CsvParams: map[string]string{
     "fileHeaderInfo":   "IGNORE",
     "recordDelimiter":  "Cg==",
     "fieldDelimiter":   "LA==",
     "quoteCharacter":   "Ig==",
     "commentCharacter": "Iw==",
    },
   },
   OutputSerialization: &api.SelectObjectOutput{
    OutputHeader: false,
    CsvParams: map[string]string{
     "quoteFields":     "ALWAYS",
     "recordDelimiter": "Cg==",
     "fieldDelimiter":  "LA==",
     "quoteCharacter":  "Ig==",
    },
   },
   RequestProgress: &api.SelectObjectProgress{
    Enabled: true,
   },
  },
 }
 csvRes, err := bosClient.SelectObject(bucket, csvObject, csvArgs)
 if err != nil {
  fmt.Println(err)
  return
 }
 parseMessages(csvRes)
 fmt.Println("------ select json object -------")
 jsonObject := "test.json"
 jsonArgs := &api.SelectObjectArgs{
  SelectType: "json",
  SelectRequest: &api.SelectObjectRequest{
   Expression:     "c2VsZWN0ICogZnJvbSBCb3NPYmplY3QucHJvamVjdHNbKl0ucHJvamVjdF9uYW1l",
   ExpressionType: "SQL",
   InputSerialization: &api.SelectObjectInput{
    CompressionType: "NONE",
    JsonParams: map[string]string{
     "type": "LINES",
    },
   },
   OutputSerialization: &api.SelectObjectOutput{
    JsonParams: map[string]string{
     "recordDelimiter": "Cg==",
    },
   },
   RequestProgress: &api.SelectObjectProgress{
    Enabled: true,
   },
  },
 }
 jsonRes, err := bosClient.SelectObject(bucket, jsonObject, jsonArgs)
 if err != nil {
  fmt.Println(err)
  return
 }
 parseMessages(jsonRes)
}
// 解析所有headers保存到map中
func parseHeaders(headers []byte) map[string]string {
 hm := make(map[string]string)
 index := 0
 for index < len(headers) {
  // headers key length
  keyLen := int(headers[index])
  index += 1
  // headers key
  key := headers[index : index+keyLen]
  index += keyLen

  // headers value length
  valLenByte := headers[index : index+2]
  valLen := int(binary.BigEndian.Uint16(valLenByte))
  index += 2
  // headers value
  val := headers[index : index+valLen]
  index += valLen
  hm[string(key)] = string(val)
 }
 return hm
}

func parseMessages(res *api.SelectObjectResult) {
 defer res.Body.Close()
 reader := bufio.NewReader(res.Body)
 for {
  // total length in prelude, 4 bytes
  p := make([]byte, 4)
  l, err := io.ReadFull(reader, p)
  if err != nil || l < 4 {
   fmt.Printf("read total length err: %+v, len: %d\n", err, l)
   break
```

```
  }
  totalLen := binary.BigEndian.Uint32(p)
  // headers length in prelude, 4 bytes
  l, err = io.ReadFull(reader, p)
  if err != nil || l < 4 {
   fmt.Printf("read headers length err: %+v, len: %d\n", err, l)
   break
  }
  headersLen := binary.BigEndian.Uint32(p)
  // headers part
  headers := make([]byte, headersLen)
  l, err = io.ReadFull(reader, headers)
  if err != nil || uint32(l) < headersLen {
   fmt.Printf("read headers data err: %+v, len: %d\n", err, l)
   break
  }
  // 获取header长度，并解析headers内容，判断具体的msg类型；end msg则结束读取，
  // cont msg则调用回调函数输出进度信息，record msg则输出记录信息
  headersMap := parseHeaders(headers)
  if headersMap["message-type"] == "Records" {
   // payload part
   payloadLen := totalLen - headersLen - 12
   payload := make([]byte, payloadLen)
   if _, err := io.ReadFull(reader, payload); err != nil {
    fmt.Printf("read payload data err: %+v\n", err)
   }
   // 设置你使用的OutputSerialization字段中的换行符做分行处理
   rs := strings.Split(string(payload), "\n")
   _, err = io.ReadFull(reader, p)
   crc := binary.BigEndian.Uint32(p)
   recordsMsg := &api.RecordsMessage{
    CommonMessage: api.CommonMessage{
     Prelude: api.Prelude{
      TotalLen:   totalLen,
      HeadersLen: headersLen,
     },
     Headers: headersMap,
     Crc32:   crc,
    },
    Records: rs,
   }
   fmt.Printf("RecordsMessage: %+v\n", recordsMsg)
   continue
  }
  if headersMap["message-type"] == "Cont" {
   // payload part, progress
   bs := make([]byte, 8)
   _, err = io.ReadFull(reader, bs)
   bytesScanned := binary.BigEndian.Uint64(bs)

   br := make([]byte, 8)
   _, err = io.ReadFull(reader, br)
   bytesReturned := binary.BigEndian.Uint64(br)

   _, err = io.ReadFull(reader, p)
   crc := binary.BigEndian.Uint32(p)

   contMsg := &api.ContinuationMessage{
    CommonMessage: api.CommonMessage{
     Prelude: api.Prelude{
      TotalLen:   totalLen,
      HeadersLen: headersLen,
     },
     Headers: headersMap,
     Crc32:   crc,
    },
    BytesScanned:  bytesScanned,
    BytesReturned: bytesReturned,
   }
   fmt.Printf("ContinuationMessage: %+v\n", contMsg)
   continue
  }
  if headersMap["message-type"] == "End" {
   _, err = io.ReadFull(reader, p)
   crc := binary.BigEndian.Uint32(p)

   endMsg := &api.EndMessage{
    CommonMessage: api.CommonMessage{
     Prelude: api.Prelude{
      TotalLen:   totalLen,
      HeadersLen: headersLen,
     },
     Headers: headersMap,
     Crc32:   crc,
    },
   }
   fmt.Printf("EndMessage: %+v\n", endMsg)
   break
  }
 }
}
```

Python SDK 示例

目前 BOS Python SDK 也支持 SelectObject 接口，具体使用见 "Python-SDK"-" 文件管理 "-"选取文件"章节的相关内容。

对象标签

概述 对象标签(Object Tag)是用于描述Object的类别, BOS支持使用标签对存储空间中的对象进行分类。 **标签规则** Object标签使用一组键值对（Key-Value）标记Object，用户可以在上传Object时添加标签，也可以为

已有Object添加标签。

- 单个Object最多可设置10个标签，Key不可重复。
- 每个Key长度不超过128字符，每个Value长度不超过256字符。
- Key和Value区分大小写。
- 标签合法字符集包括大小写字母、数字、空格和以下符号： +-=._:/

通过HTTP Header的方式设置标签且标签中包含任意字符时，您需要对标签的Key和Value进行URL编码。 **注意事项** 只有Bucket拥有者以及被授予PutObjectTagging权限的用户拥有读写Object标签的权限。 您可以在简单上传、分片上传、追加上传以及拷贝文件过程中为Object设置标签，也可以对已上传Object设置标签。 更改标签信息不会更新Object的Last-Modified时间。 BOS只针对广州集群开放了对象标签，其他地域的Bucket需提交工单开启对象标签功能。

数据发布

绑定自定义域名

基本介绍

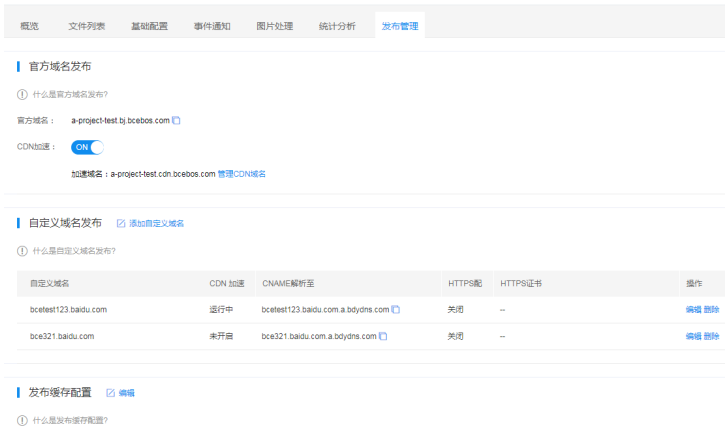
用户可以绑定自定义的Bucket域名，绑定域名即可实现外网直接发布，用户可使用绑定的域名直接访问Bucket。

绑定规则

- 被绑定的域名不得包括协议、端口和路径，而应只包含域名本身。
- 每个Bucket最多可绑定20个不同的域名。
- 一个域名只能绑定在单个bucket上。

操作方式

1. 点击Bucket名称进入Bucket管理页面并选择“发布管理”页签。
2. 点击“添加自定义域名”完成Bucket域名绑定；添加完成后会在下方的自定义域名列表中显示。



CNAME域名解析

完成绑定之后，请登录域名服务商后台，在域名管理中添加CNAME域名解析，将需要的域名CNAME到BOS的域名`<region>.bcebos.com`下(`<region>`为Bucket所在的区域，请参考区域选择说明，即可完成全部域名绑定配置。

例如：

- 如您的Bucket所在区域为北京，请CNAME到`bj.bcebos.com`。
- 如您的Bucekt所在区域为广州，请CNAME到`gz.bcebos.com`。
- 如您的Bucket所在区域在苏州，请CNAME到`su.bcebos.com`。

绑定https证书

当您做完CNAME解析之后，如果您希望以https方式访问此自定义域名，可将您持有的https证书绑定至此自定义域名。

百度智能云SSL证书服务提供了完备的证书申请与管理一站式服务，用户可在百度智能云上购买、管理、使用https证书，详情可参见证书管理

您在百度智能云SSL证书服务中持有可用证书后，在BOS自定义域名控制台中，可在首次创建自定义域名时或对已有自定义域名进行修改时，点击“HTTPS证书”功能项，下拉展示出您在百度智能云SSL证书服务中管理的所有证书，并按需选择对应证书。

当域名已绑定的证书过期或失效时，您同样可以选中该自定义域名，点击“HTTPS证书”功能项，在下拉菜单中选择可用的证书进行替换。

访问地址

完成绑定之后，可使用绑定的域名直接访问Bucket。

例如：给Bucket：`video`绑定的域名为`video.51mingxi.com`，请在域名服务商后台将域名`video.51mingxi.com` CNAME为`<region>.bcebos.com`。完成后，使用URL：`http://video.51mingxi.com/`也可直接访问此Bucket。

注意事项

- 添加的自定义域名需要到域名注册商DNS解析管理中，完成自定义域名CNAME解析，解析生效后才可以通过自定义域名访问。如果需要对自定义域名开通CDN加速，点击域名对应操作列的“开通CDN加速”。如果希望解除自定义域名的绑定，点击域名对应操作列的“删除”。
- 如绑定的域名原有其他解析记录，CNAME到`<region>.bcebos.com`后，需要删除原解析记录。
- 当您选择的证书所覆盖的域名范围，与您预期绑定的自定义域名不匹配时，界面会给予相应提示，您需要更换其他可用的证书。
- 建议您在百度智能云SSL证书服务中，注意查看绑定证书的生效时间，避免证书到期后https访问不可用。
- 当前BOS暂时支持为bj、su、gz三个region下bucket的自定义域名配置https证书。

- 选择绑定某自定义证书后，绑定关系的建立通常需要5-10分钟，请您耐心等待后尝试使用。

CDN加速发布

基本介绍

BOS目前支持使用CDN加速指定的Bucket，选择需要加速的Bukcet后，BOS对应Bucket的域名的DNS解析修改为CDN提供的CNAME域名，从而接入CDN节点，开启CDN加速。

CDN配置规则

- 若没有在CDN中配置过期时间，且源站没有显示提供http缓存控制头，CDN不缓存数据。在此情况下，Bukcet内的数据更新之后无法实现CDN加速，需在CDN中手动刷新缓存，具体操作请参考刷新缓存。
- 可在CDN中配置过期时间，用以自动更新缓存，具体操作请参考配置缓存过期时间。

操作方式

1. 点击Bucket名称进入Bucket管理页面并选择“发布管理”页签。
2. 点击官方加速域名右侧的“开启加速”即可直接开启官方的CDN加速域名。



开启加速成功后，刷新页面可以看到官方域名右侧的按钮变为“管理CDN加速”，点击可以进入CDN的域名管理页面对域名详情进行管理。

3. 如果需要添加自定义加速域名，点击“添加自定义加速域名”，完成添加后会在自定义域名列表中显示。如果需要对加速域名进行管理，点击对应操作列的“CDN加速管理”。

注意事项

- Bucket创建时缺省为不开启CDN加速。
- BOS提供了官方的CDN加速域名，即bucketname.cdn.bcebos.com，官方加速域名为BOS Bucket开启的官方CDN加速域名，不需要配置CNAME解析，该域名不可修改。
- 添加自定义域名前需要到域名注册商DNS解析管理中，将自定义域名CNAME解析至CDN提供的CNAME地址，解析成功后才可以通过CDN加速访问。具体操作请参考绑定CNAME。
- 如果添加的自定义域名被占用，请提交工单处理。

数据安全

权限控制

Bucket权限控制

概述

创建 Bucket 后，可使用 ACL 对 Bucket 权限进行管理。

操作方式

BOS 同时支持使用 API 与使用 SDK 管理 Bucket 权限，具体如下：

- 使用 API 管理 Bucket 权限
  - 上传ACL文件
  - 使用CannedAcl
- 使用 SDK 管理 Bucket 权限
  - Java SDK
  - Python SDK
  - PHP SDK
  - C# SDK
  - Android SDK

例如使用 Java SDK 的示例代码：

```
public void setBucketPrivate (BosClient client, String bucketName) {
  client.setBucketAcl(bucketName, CannedAccessControlList.Private);
}
```

注意事项

在创建 Bucket 的时候，默认权限为 Private。

Object权限控制

概述

目前 BOS 也提供 Object 级别的细粒度访问权限控制，即 Bucket 内不同 Object 对不同用户具有不同的访问控制权限。用户可以通过 CannedACL 和 ACL 文件的方式对单个 Object 的访问权限进行控制，Object 目前支持 READ 和 FULL_CONTROL 两种权限类型。

操作方式

BOS 同时支持使用 API 与使用 SDK 管理 Object 权限，具体如下：

- 使用 API 管理 Object 权限

- Object权限控制

- 使用 SDK 管理 Object 权限
  - Java SDK
  - Python SDK
  - PHP SDK
  - Ruby SDK
  - GO SDK

注意事项

- 新创建的 Object 默认 Object ACL 为空，用户访问时以 Bucket ACL 为准。
- 如果 Object 没有设置权限，则 Object ACL 为空，默认以 Bucket 权限为准。
- Object ACL 优先级高于 Bucket ACL，例如用户设置了 Object 的权限是 public-read，无论 Bucket 是什么权限，该 Object 都可以被访问。
- 目前 Object ACL 鉴权只对 GetObjectMeta、GetObject、CopyObject、UploadPartCopy 四个接口生效。
- 获取、设置和删除 Object Acl 需要拥有 Object 的 FULL_CONTROL 权限；如果 Object ACL 为空，则需要拥有 Bucket 的 FULL_CONTROL 权限或者用户是 Bucket Owner。

IAM主子用户访问控制

概述

在您的百度智能云账号下面，通过 IAM 可以创建具有自己 AccessKey 的子用户。您的百度智能云账号被称为主账号，创建出来的账号被称为子用户，使用子用户的 AccessKey 只能使用主账号授权的操作和资源，BOS已经接入IAM子用户系统，通过控制台多用户访问控制创建IAM子用户，您可以实现：

1. 同账号授权：您可以授权自己账号下的子用户去管理自己账号下的云资源（Bucket和Object）。

2. 跨账号授权：您可以授权自己账号下的子用户去管理他人账号下的云资源（Bucket和Object）。

> 当您在授权的时候，建议严格遵循最小权限原则，限定用户执行受限的操作（如仅授权读操作）并设置访问指定前缀的资源，避免授予过大的权限，导致预期外的越权操作，造成数据安全风险。

应用场景

- 企业子账号管理与分权

  企业A使用百度智能云账号购买了多种云资源（如BCC实例/RDS实例/BLB实例/BOS存储/...），A的员工需要操作这些云资源包括购买、运维、线上应用等。不同岗位员工的工作职责不一样，需要的权限也不一样。出于安全考虑，A不希望将主账号的密钥直接公布给员工，而希望能给不同岗位员工创建相应的用户子账号。用户子账号只能在授权的前提下操作资源，不需要进行独立的计量计费，所有资源费用都归属主账号。A主账号随时可以撤销子账号的权限，也可以随时删除其创建的子账号。

- 企业之间的资源操作与授权管理

  A和B代表不同的企业。A购买了多种云资源（如BCC实例/RDS实例/BLB实例/BOS存储/...）来开展业务。A希望能专注于业务系统，而将云资源运维监控管理等任务委托或授权给企业B。企业B也可以进一步将代运维任务分配给B的员工，即B为其员工创建相应的用户子账号供其使用。B可以精细控制其员工对A的云资源操作权限。如果A和B的这种代运维合同终止，A随时可以撤销对B的授权。

策略文件说明

IAM主用户授权给子用户通过关联策略实现，策略文件本质上是一个JSON文件，文件中permission和resource用来定义权限和资源。策略文件对应的空白模版如下：

```
{
  "accessControlList": [
    {
      "service": "bce:bos",
      "region": "*",
      "effect": "Allow",
      "permission": [],
      "resource": [],
    }
  ]
}
```

策略文件中各字段的含义如下：

字段	数据类型	说明	是否必须	父节点
accessControlList	list	标识acl主体的开始，由一或多组acl配置项组成，其中acl配置项由service+region+effect+permission+resource组合而成。	是	无
+service	string	acl配置项影响的服务组件。	是	accessControlList
+region	string	acl配置项影响的区域，取值范围为**bj**、**gz**和 * 。其中bj代表北京region，gz代表广州region，*代表示所有区域。	是	accessControlList
+effect	string	指定与该条acl配置项匹配的Request能否执行，取值为Allow或Deny。Allow表示可以执行Deny表示拒绝执行。	是	accessControlList
+permission	list	ACL配置项所影响的权限，取值范围为READ、LIST、WRITE、FULL_CONTROL和ListBuckets，不支持配置通配符*。每个permission的具体含义见下表。	是	accessControlList
+resource	list	ACL配置项所影响的资源，支持通配符。如：*，<BucketName>，<BucketName>/<Prefix>；*，<BucketName>/<ObjectKey>。IAM在校验请求时，对resource字段是做严格字符串匹配，因此若该字段被配置为"abc"，则表明该条规则仅对名为"abc"的bucket相关的bucket级别操作生效，而对object级别操作不生效。比如说请求API为操作bucket "abc"下所属某object（如"obj01"），则BOS传给IAM的资源字段为"abc/obj01"，与策略文件中所配置resource（"abc"）会匹配不上，该条规则不生效。	否	accessControlList

每种permission所对应的BOS API如下：

permission	对应的BOS API
ListBuckets	GetService(ListBuckets)
PutBucket	PutBucket
READ	GetBucketLocation, HeadBucket, GetObject, GetObjectMeta, ListParts
DeleteObject	DeleteObject
LIST	ListObjects, ListMultipartUploads
WRITE	PutObject, InitiateMultipartUpload, UploadPart, CompleteMultipartUpload, AbortMultipartUpload, DeleteObject, DeleteMultipleObjects, AppendObject, PostObject
FULL_CONTROL	READ、WRITE、LIST对应的API, 还包含PutBucketACL, GetBucketACL, PutBucketCors, GetBucketCors, DeleteBucketCors, PutBucketStyle，GetBucketStyle，PutBucketMirroring，GetBucketMirroring，PutCopyRightProtection，GetCopyRightProtection，PutBucketLifecycle，GetBucketLifecycle，PutBucketReplication，GetBucketReplication，PutBucketEncryption，GetBucketEncryption，PutBucketStaticWebsite，GetBucketStaticWebsite，PutBucketLogging，GetBucketLogging，PutBucketRequestPayment，GetBucketRequestPayment，PutBucketTagging，GetBucketTagging，PutNotification，GetNotification，PutBucketObjectLock，GetBucketObjectLock，PutBucketInventory，GetBucketInventory，PutBucketStorageAnalysis，GetBucketStorageAnalysis，PutBucketStorageClass，GetBucketStorageClass，PutBucketTrash，GetBucketTrash，PutBucketQuota，GetBucketQuota，GetObjectVersion，DeleteObjectVersion，GetObjectVersionAcl，PutObjectVersionAcl，PutBucketVersioning，GetBucketVersioning，ListObjectVersions

系统策略配置说明

为了方便用户使用，百度智能云内置了两个常用策略作为系统策略：BosFullAccess和BosListAndReadAccess。

- BosFullAccess：管理百度智能云对象存储服务（BOS）的权限。
- BosListAndReadAccess：只读访问百度智能云对象存储服务（BOS）的权限。

> **说明：**
>
> - 系统策略无法修改且不能删除。

在“策略管理”页面点击策略名称对应操作列的“查看”按钮，可以查看两个系统策略对应的JSON文件。

自定义策略配置说明

如果您需要定制更精细的权限控制，可以创建一个自定义策略。自定义策略即用户通过策略文件定义子账号的资源和权限，通过自定义策略用户可以更精准的控制权限和资源。策略本质上为JSON文件，您可以参考策略文件说明，也可以参考以下典型场景示例。 您也可以使用BOS ACL编辑工具获取策略模板及自定义策略配置。 示例中假设bucket名为mybucket。

授权子用户某个Bucket的完全管理权限（使用控制台管理）

```
{
  "accessControlList": [
    {
      "service": "bce:bos",
      "region": "*",
      "effect": "Allow",
      "permission": [
        "FULL_CONTROL"
      ],
      "resource": [
        "mybucket",
        "mybucket/*"
      ]
    },
    {
      "service": "bce:bos",
      "region": "*",
      "effect": "Allow",
      "permission": [
        "ListBuckets"
      ],
      "resource": [
        "*"
      ]
    }
  ]
}
```

> **注意：**
>
> - resource字段需要同时写mybucket和mybucket/*。
> - 如果您希望通过控制台管理某个bucket，那还需要有ListBuckets的权限，不然Bucket列表是打不开的。
> - 子用户支持实现跨账号资源授权。比如这里的mybucket可以是其他账号下的资源。假设这里有两个账号，账号A下有mybucket，账号B下有个子用户。账号A可以首先通过更新Bucket权限来把mybucket授权给账号B来使用。然后账号B可以创建一个自定义策略来进一步把管理mybucket的权限授予子用户。

授权子用户某个Bucket的完全管理权限（不使用控制台管理和BOS桌面）

```
{
  "accessControlList": [
    {
      "service": "bce:bos",
      "region": "*",
      "effect": "Allow",
      "permission": [
        "FULL_CONTROL"
      ],
      "resource": [
        "mybucket",
        "mybucket/*"
      ]
    }
  ]
}
```

> **注意：**
>
> - resource字段需要同时写mybucket和mybucket/*。
> - 如果不通过控制台来管理bucket，则无需开放ListBuckets权限。您可以直接通过BOS周边工具、SDK来使用BOS服务。

授权子用户对某个prefix（目录）的只读权限

假设某bucket用于存放照片，照片按照拍摄地点存放，每个拍摄地点下有年份子目录。现在需要授予子用户读取mybucket/shanghai/2013/目录的只读权限。目录结构如下：

```
 mybucket //bucket
|-- beijing
| |--2010
| |--2011
|-- shanghai
| |--2012
| |--2013 //授予此目录的只读权限
|--shenzhen
 |--2014
 |--2015
```

假设子用户已经知道所有文件的路径，且需要列出文件的权限：

```
{
  "accessControlList": [
    {
      "service": "bce:bos",
      "region": "*",
      "effect": "Allow",
      "permission": [
        "LIST",
        "READ"
      ],
      "resource": [
        "mybucket/shanghai/2013/",
        "mybucket/shanghai/2013/*"
      ]
    },
    {
      "service": "bce:bos",
      "region": "*",
      "effect": "Allow",
      "permission": [
        "READ"
      ],
      "resource": [
        "mybucket"
      ]
    }
  ]
}
```

其他说明：

> - 若子用户需使用BOS控制台上传文件，您不仅需要完成对应操作授权，还需设置子用户访问方式为【编程访问】与【控制台密码访问】，否则将导致文件上传失败。

设置服务端加密

概述

BOS支持用户在上传和复制请求（PutObject、PostObject、InitiateMultipartUpload、AppendObject、FetchObject、CopyObject）中携带x-bce-server-side-encryption的HTTP header并指定用户的加密算法（目前支持AES256与SM4），从而实现数据的有效安全保护。

操作方式

您可以通过 Console 控制台、 SDK 和 API 来设置服务端加密，具体如下：

- 通过 Console 控制台设置服务端加密，目前支持AES256和SM4加密算法，请参考设置服务端加密。

通过 SDK 设置服务端加密，可设置AES256与SM4加密算法：

- Java SDK

- PHP SDK

通过 API 设置服务端加密：

- PutBucketEncryption
- GetBucketEncryption
- DeleteBucketEncryption

注意事项

1. 如果Object的请求中通过携带x-bce-server-side-encryption HTTP header并指定加密，那么不管Bucket加密开关是否打开，该文件都会加密。
2. 如果Object请求中未指定加密，那么如果Bucket加密开关打开，该文件会加密。
3. Bucket加密开关状态变化，不会影响已有文件，仅会对新文件生效。

设置防盗链

概述

为了防止用户存储在 BOS 上的数据被其他人盗链，BOS 支持基于 HTTP header referer 字段的防盗链方法。用户可以通过 BOS 控制台设置 Referer 字段的白名单。设置白名单后，只有 Referer 字段在白名单内的用户才可以访问 Bucket 中存储的数据，不在白名单内的请求会被拒绝。但若用户的 Referer 为空时，默认可以访问，不受白名单限制。

操作方式

设置 Referer 白名单功能目前只支持通过控制台进行操作，具体操作以及一些限制请参考设置 Referer 白名单。

设置回收站

应用场景

当开发者在 BOS 进行数据存储和管理时，基于更高的数据可靠性要求，通常希望如下场景有更好的解决方案：

- 当使用数据时产生误删情况，希望有途径将数据找回。
- 当一个 Bucket 通过 ACL 授权给多个用户，并给予他们 Write 权限时，希望能够避免这些用户对 Bucket 中的数据进行恶意删除操作。

操作方式

基于上述场景为了提升 BOS 的数据可靠性，BOS 提供了回收站功能，Bucket 的 Owner 且对 Bucket 具备 Full Control 权限的用户可对 Bucket 进行回收站相关的配置。配置方法可根据开发习惯，使用 API 或控制台等多种方式：

- 通过 API 管理回收站：
  - PutBucketTrash：开启回收站功能。
  - GetBucketTrash：查看当前回收站状态。
  - DeleteBucketTrash：关闭回收站功能。
- 通过 Console 控制台设置回收站请参考设置回收站。
  1. 当Bucket开启回收站功能后，用户即可在该Bucket的回收站目录中查找到Bucket中被删除的数据。该回收站路径默认为Bucket/.trash/
  2. 如果您需要将回收站内的数据进行恢复，则直接rename该回收站内的文件，将其文件名中的.trash/路径去除即可
  3. 如果您需要将回收站内的数据进行彻底清除，则直接将.trash/路径下的某文件再次delete即可

同时，基于上述应用场景中的第二种情况，您也可以将回收站能力与Bucket ACL做一些组合，例如为某需要在BucketA中有Write权限的用户，配置如下ACL：

```
{
 "id": "policy_3563bce24f2049629f4e0b6bf7a2775d",
 "version": "v1",
 "grantee": [
            {
               "id": "b124deeaf6f641c9ac27700b41a350a8"
            }
         ],
 "accessControlList": [
  {
   "service": "bce:bos",
   "region": "bj",
   "resource": [
    "BucketA"
   ],
   "effect": "Allow",
   "permission": [
    WRITE"
   ]
  },
  {
   "service": "bce:bos",
   "region": "bj",
   "resource": [
    "BucketA/.trash/"
   ],
   "effect": "Deny",
   "permission": [
    "DeleteObject"
   ]
  }
 ]
}
```

通过上述ACL，该用户可对Bucket进行写入和删除行为，但对于回收站内的数据无法进行二次彻底删除。

注意事项

1. 当Bucket已开启回收站功能并设置回收站路径为.trash/，之后将Bucket内的文件A删除（原文件路径为Bucket/test/A），则该文件A的路径变为Bucke/.trash/test/A，之后您关闭该Bucket的回收站，则该文件A在Bucket中的保留路径仍为Bucket/.trash/test/A。因此，提醒您在关闭回收站功能后，同时按需将类似垃圾文件清除。
2. 当Bucket已开启回收站功能并设置回收站路径为.trash/，删除后的文件将进入Bucket/.trash/目录下，之后您将回收站文件路径更改为.trashX/，路径变更前的回收站文件将保留在原位置，即Bucket/.trash/路径下，而之后被删除的文件，将进入Bucket/.trashX/路径下。因此，提醒您在变更回收站路径时，对回收站内原有文件按需进行搬迁。
3. **删除归档类型文件时，归档类型的文件不会进入回收站，会直接删除**。

计费模式

- BOS回收站不另外收费，回收站内的文件还视为Bucket内数据的一部分，按照其存储类型收取容量费用。
- 当回收站内的文件被彻底清除后，将不收取容量费用。

版本控制

版本控制概述

版本控制是针对存储空间（Bucket）级别的数据保护功能。开启版本控制后，针对数据的覆盖和删除操作将会以历史版本的形式保存下来。您在错误覆盖或删除对象（Object）后，能够将存储在Bucket中的Object恢复至任意时刻的历史版本。

使用场景

建议您在以下场景中使用版本控制，为您的数据安全提供更好的保障。

1. 数据误删除
2. 您删除BOS数据后想要找回时，可使用版本控制功能，恢复已删除的数据。
3. 文件被覆盖
4. 对于网盘、在线协作类产品，文件会被频繁修改，针对文件的编辑会产生大量的临时版本。您可以使用版本控制功能找回某个时间点的版本。

注意事项

- 费用说明

版本控制功能本身不收取任何费用，但对当前版本和所有历史版本的文件都会收取存储费用。此外，若您对历史版本文件进行下载或恢复等操作，还会产生相应的请求费用、流量费用等。计费详情，请参见计量项与计费项。

- 权限说明

只有Bucket的拥有者及授予了PutBucketVersioning权限的RAM用户才能配置版本控制。

- 功能互斥

同一Bucket中，版本控制与合规保留策略无法同时配置。

如果Bucket已开启版本控制，上传文件时附加的覆盖同名文件请求头x-bce-forbid-overwrite将不生效。更多信息，请参见请求头。

同一Bucket中，版本控制和回收站无法同时配置。

在多版本初期阶段，存在多个功能互斥，多个az不支持多版本功能，具体详情可以询问相关技术人员。gz集群允许开启多版本。

版本控制状态

Bucket包含三种版本控制状态，分别为未开启、开启或者暂停。

默认情况下，Bucket版本控制状态为“未开启”。如果Bucket处于“开启”版本状态，将无法返回至“未开启”状态。但是，您可以暂停Bucket的版本控制状态。

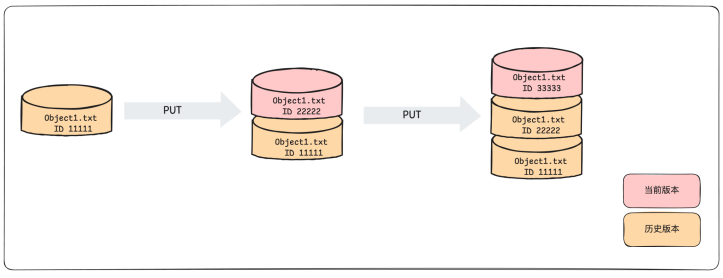
当Bucket版本控制处于“开启”状态时，BOS将为新上传的Object生成全局唯一的随机字符串版本ID。

当Bucket版本控制处于“暂停”状态时，BOS将为新上传的Object生成特殊字符串为“null”的版本ID。

管理启用版本控制状态下的对象

写入对象

在已开启版本控制的Bucket中上传Object时，会为这些上传的Object添加唯一的版本ID。

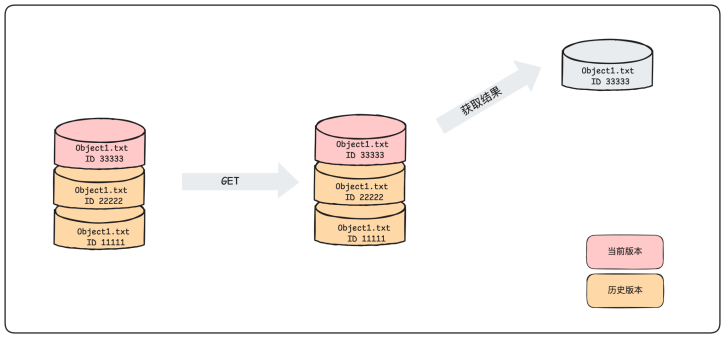


列举对象

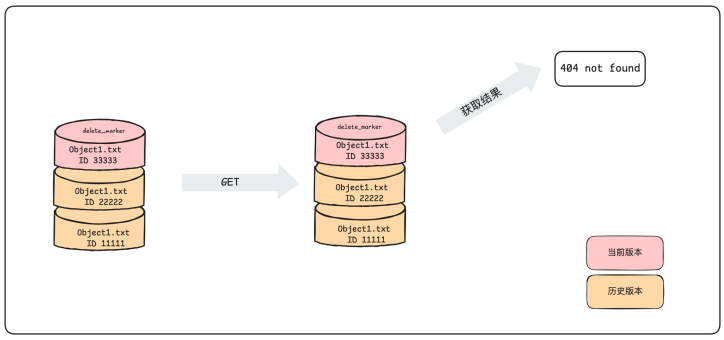
对象存储在与存储桶关联的 versions 参数中存储对象版本信息。 按照存储时间的先后顺序返回对象版Object，同名Object按照最近写入时间顺序。

下载对象

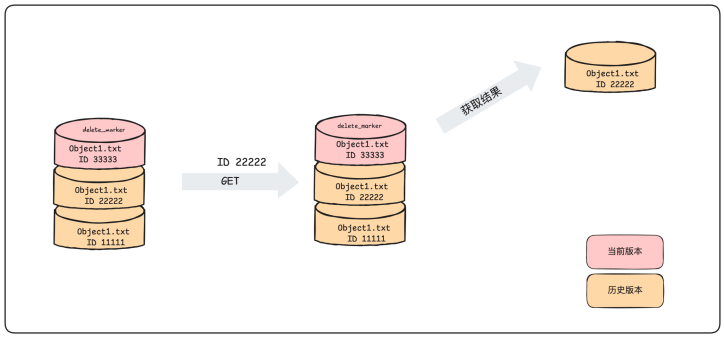
通过GET请求下载Object时，如果未指定Object的版本ID，默认情况下返回Object的最新版本。



如果最新版本为delete marker, 那么返回404

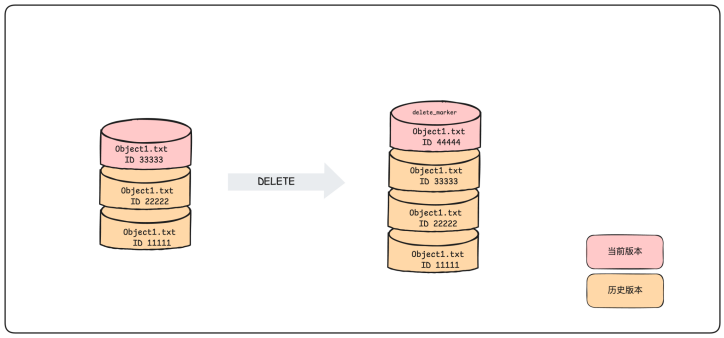


如果指定了Object的版本Id，返回指定版本的内容。

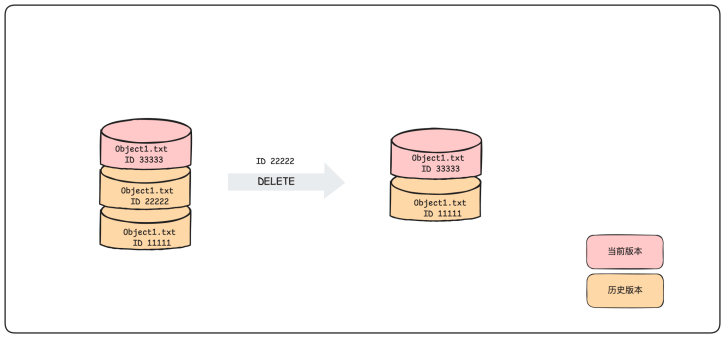


删除文件

如果删除Object时未指定版本ID，则Bucket中将插入一个删除标记（Delete Marker）作为当前版本。



如果删除Object时指定版本Id，那么该object就会被彻底删除。



恢复文件

恢复文件有两种方法可执行：

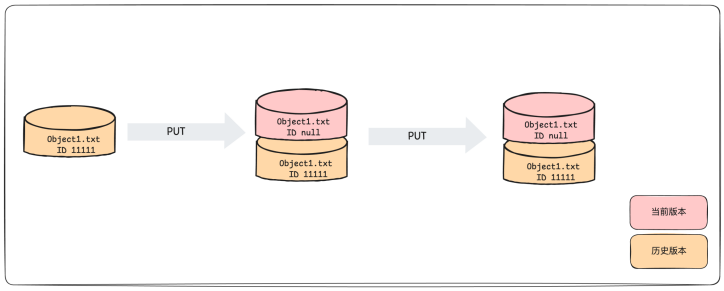
1. 使用Copy接口将对象的早期版本复制到同一存储桶中

2. 指定当前版本Id进行永久删除对象，那么历史版本就会成为最新版本。

管理暂停版本控制状态下的对象

写入对象

在已暂停版本控制的Bucket中上传Object时，Object的版本Id为空。

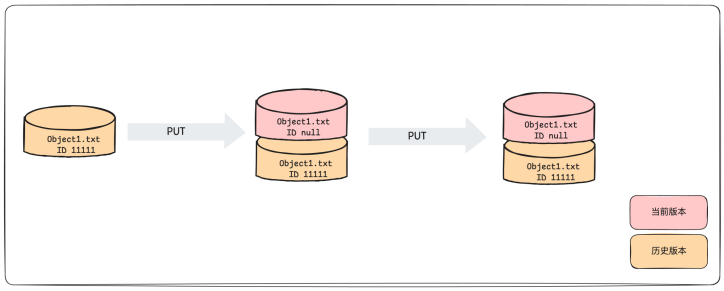


列举对象

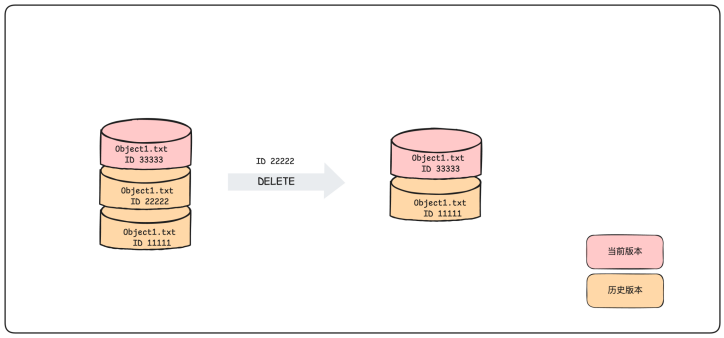
对象存储在与存储桶关联的 versions 参数中存储对象版本信息。 按照存储时间的先后顺序返回对象版Object，同名Object按照最近写入时间顺序。

下载对象 和开启态的表现相同

删除文件 如果删除Object时未指定版本ID，则Bucket中将插入一个删除标记（Delete Marker）作为当前版本。该删除标记的版本Id为"null"



如果删除Object时指定版本Id，那么该object就会被彻底删除。



恢复文件

恢复文件有两种方法可执行：

1. 使用Copy接口将对象的早期版本复制到同一存储桶中

2. 指定当前版本Id进行永久删除对象，那么历史版本就会成为最新版本。

删除标记

删除标记（Delete Marker）可以认为是版本控制状态下，删除Object产生的占位符。“删除标记”与Object同样具有相同地位，都具有Object name和版本ID。区别在于以下几点：

- “删除标记”的内容为空。
- “删除标记”不存在 ACL 值。
- 如果Object为“删除标记”，GET 请求会返回404。
- “删除标记”只能够进行彻底删除，不能对“删除标记”进行多版本控制。

如何删除“删除标记”

- 在DeleteObject请求中，没有指定“删除标记”的版本ID，会生成带有版本ID的“删除标记”，并不会删除“删除标记”。
- 在DeleteObject请求中，指定“删除标记”的版本ID，才能够彻底删除“删除标记”。
- 如果最新版本为“删除标记”，并且没有指定版本ID，返回404。 注意： 只有经过主账号授权DeleteObject操作后才可删除“删除标记”。

最小权限原则

最小权限原则

当您为用户授予权限时，应当尽可能缩小条件的适用范围，包括但不限于减少被授权人的数量、缩小适用资源的范围、减少授权的操作、增加授权生效的限制条件等。

最小权限原则适用于一切通过BOS进行的授权操作，包括设置Bucket读写权限、设置Object读写权限、设置子用户访问权限、申请临时授权访问等。当您在进行授权操作时，请严格遵守最小权限原则，避免授予过大的权限，造成数据安全风险。

常见问题解答

1、AK/SK的作用是什么？如果泄露了会有什么风险？ AK（Access Key ID）/SK（Secret Access Key）用于对用户的调用行为进行鉴权和认证，相当于百度智能云API专用的用户名及密码。在调用API发送请求时，需

要使用AK/SK计算签名并包含在API请求中。AK/SK泄露会导致不具有有效访问权限的未知用户访问BOS，增加数据非法篡改、泄露等风险，请务必妥善保管AK/SK。

2.有哪些场景需要使用临时授权服务STS？ 临时授权服务STS是一种区别于原始AK/SK的鉴权方式，可以在不共享AK/SK的情况下为用户提供BOS服务使用权限，具有高访问安全性，推荐您在为第三方用户授权时使用STS方式。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访问百度智能云资源，STS更多使用说明详见临时授权访问。

3.如何提升多用户共用Bucket的数据安全性？ 可以通过给Bucket划分区域的方式实现多用户隔离，给用户授权时仅开放该Bucket某路径下的权限，例如：自定义bucket权限时，将user1的resource设置为Bucket1/user1_id/*，user2的resource设置为Bucket1/user2_id/*。

4.Bucket粗粒度权限和细粒度权限有什么区别？ 在自定义Bucket权限功能中，BOS支持用户进行粗粒度和细粒度权限的组合配置，您可以通过控制台或API完成此操作。粗粒度权限是BOS预置的权限模板，每个粗粒度权限由一组常用操作的打包而成，方便用户快捷配置。细粒度权限精细到各个API操作，为用户提供严格管理授权的能力。为降低权限控制不严格导致的数据安全风险，建议在配置粗粒度权限的基础上，搭配allow/deny进行细粒度权限配置。粗/细粒度权限详见Bucket权限控制。

数据容灾

多AZ存储

概述

开启多AZ存储后，BOS会将用户的数据分散存放在同一地域下的多个可用区内。

多AZ存储可以提供强大的容灾能力，当某个可用区因物理原因不可用时，BOS仍然能保障数据的正常访问，从而保证业务的连续性。

指标对比

开启多AZ存储与关闭多AZ存储时，标准存储Object的各项指标对比如下：

对比指标	标准存储 -多AZ	标准存储	低频存储 -多AZ	低频存储
可用性	99.995%	99.99%	99.995%	99.95%
最低存储时间	无	无	30天	30 天
数据取回费用	无	无	按量收费（GB）	按量收费（GB）
数据访问特点	实时访问，毫秒延迟。	实时访问，毫秒延迟。	实时访问，毫秒延迟。	实时访问，毫秒延迟。
适用场景	移动应用、网站视频、数据分析等需极高可靠性的场景。	移动应用、网站视频、数据分析等。	网盘存储、监控数据、媒资备份等需极高可靠性的场景。	网盘存储、监控数据、媒资备份等。
场景特点	数据频繁读取场景。	数据频繁读取场景。	月均访问次数小于 1 次场景。	月均访问次数小于 1 次场景。

> **说明：**
>
> - 标准存储、标准存储-多AZ、低频存储、低频存储-多AZ的数据可靠性都是99.9999999999%（12个9）。

操作方式

用户可以通过控制台在创建Bucket时设置多AZ存储，请参见：

- 使用控制台设置多AZ存储。

使用说明

- BOS 只支持在创建 Bucket 时选择多 AZ 类型存储。Bucket创建成功后，您将无法修改多 AZ 存储的启停状态，请您根据业务情况提前规划。
- 目前 BOS 多 AZ 存储支持标准存储-多AZ 类型和低频存储-多AZ 类型。
- 目前 BOS 多 AZ 存储在华北-北京、华东-苏州、华南-广州地域，您可以通过提交工单的方式来获取使用权限。
- 暂不支持存量存储桶，转化为多 AZ 类型存储。

数据同步

概述

数据同步 (Replication) 是 BOS 数据中心 (Region) 的存储空间 (Bucket) 之间自动、异步复制文件 (Object) 的功能，它会将 Object 的创建、更新和删除等操作从源存储空间复制到目标存储空间。

数据同步功能能够很好的提供Bucket跨区域、跨账号或其他数据复制的需求。目标 Bucket 中的对象是源 Bucket 中对象的精确副本，它们具有相同的对象名、元数据以及内容，例如创建时间、拥有者、用户定义的元数据、Object ACL、对象内容等。

使用场景

您可能基于各种原因对 Bucket 配置数据同步，这些原因包括：

- **高效访问，减少延迟**：跨 Region 访问 Bucket 或者 Object，访问速度会因地理上的差异而降低。通过数据同步功能可提前将需要的数据同步至目标 region，提高访问效率。
- **跨地域容灾**：基于金融、政府等行业的合规性要求，在同一 Region 中虽然有数据多副本设置。但为了防止洪水、地震等机房级别灾害，也要求数据跨 Region 保存一份副本，BOS 的数据同步功能提供了这样的能力。
- **跨账号复制**：适用于企业的多账号场景。处于数据安全和多备份考虑，企业希望将数据从 A 账号定期同步到 B 账号，以实现数据的跨账号复制和共享。
- **跨区域数据复用**：由于业务原因例如跨 Region 搭建的计算集群需要使用同一组数据等场景，可以使用 BOS 数据同步功能复制数据副本。

操作方式

BOS同时支持使用控制台与使用SDK进行跨区域复制，具体如下：

- 使用控制台进行数据同步
- 使用API进行数据同步
  - PutBucketReplication
  - GetBucketReplication
  - DeleteBucketReplication
  - GetBucketReplicationProgress
  - ListBucketReplication

- 使用SDK进行数据同步：
  - PHP SDK

使用说明

- 可以通过定义文件名前缀的方式指定源 Bucket 中待同步的数据，也可直接同步源 Bucket 全部数据。
- 目标 Bucket 的存储类型可以选择和源对象相同，也可以根据需要选择其它的存储类型。
- 对非标准 Object 进行数据复制操作时，会触发取回费用。
- 目的 Bucket 文件的存储类型不支持归档存储。
- 规则添加成功后可以在列表中看到当前 Bucket 已有的同步策略，也可以编辑和删除。
- 开启历史文件复制后，历史 Object 将匹配增量数据中的前缀规则，仅同步属于同一前缀的文件；若指定同步 Bucket 内全部数据，则历史也全部同步;如果修改规则，已经同步的历史数据也会重新同步。
- 数据同步的两个 Bucket 可以是**跨区域**的，也可以是**同区域**的。只有跨区域 Bucket 间的数据同步会触发流量费用，同区域 Bucket 间的数据同步，不会收取流量费用。
- 数据同步的两个 Bucket 可以是**跨账号**的，也可以是**同账号**的。请您仔细确认输入的目标 Bucket 名称准确无误。若目标账号中的目标 Bucket 处于其他区域，将产生跨区域流量费用，流量费用由源账号承担，目标账号承担存储和请求费用。
- 支持 Bucket 间的**多规则同步**，即一个源 Bucket 可以有多个目的 Bucket，一个目的 Bucket 可以有多个源 Bucket。假设有三个 Bucket，分别命名为 A、B、C：
  - 支持将 A 作为 B 的源 Bucket 的同时，将 B 作为 C 的源 Bucket；
  - 支持将 A 作为 B 的源 Bucket 的同时，将 B 作为 A 的源 Bucket；
  - 支持将 A 同时作为 B 和 C 的源 Bucket；
  - 支持将 C 同时作为 A 和 B 的目标 Bucket。
- 出于数据安全考虑，BOS 不会获取并返回跨账号同步的启动状态。若您已经设置并开启跨账号同步，请主动查询目的账号中数据同步是否已经开启。若源数据在 5 分钟内还未同步到目的账号 Bucket 中，请再次检查写权限是否开启。若写权限已开启后仍同步失败，请提交工单。

使用限制

- 对于处于同步状态的两个 Bucket，源 Bucket 复制过去的 Object 可能会覆盖目标 Bucket 中同名的 Object，使用中请注意。
- 由于数据同步采用异步复制，数据复制到目标 Bucket 需要一定的时间，通常几分钟到几小时不等，取决于数据的大小。

查看同步进度

完成数据同步规则配置后，您可通过控制台点击该条同步规则【操作】中的【预览】按钮，查看历史数据同步进度与实时同步时间点与详细的数据同步规则配置信息。

- 历史数据同步：您可以查看历史文件同步进度，掌握同步进展。
- 实时数据同步：您可以查看最近增量文件的数据同步时间点信息。

详细操作请参考控制台数据同步进度查询。

注意事项

数据跨境合规承诺 您使用该产品或该功能，将使您在云上的业务数据传输至您所选择的区域或产品部署区域，可能会涉及数据跨境。

使用该功能代表您同意并确认，您完全拥有对相关业务数据的合法处置权限，并对您的数据传输行为或指令行为全权负责。您确认并承诺，您的数据传输符合所有适用法律的规定，包括并不限于获得数据主体充分、明示同意，完成数据出境安全评估，与接收方签署个人信息出境标准合同等法定义务，且您承诺您的业务数据不含任何所适用法律限制、禁止传输或披露的内容（具体合规要求请您咨询相关主管部门）。

如您未遵守前述声明与保证，您将承担全部法律后果，并赔偿百度智能云和/或其关联公司遭受的全部损失。

如因法律法规、监管政策规定导致百度智能云须变更或停止提供本产品功能，您理解并同意这并不属于百度智能云违约。百度智能会为您提供服务变更或停止的过渡期协助。

事件通知

概述

BOS事件通知功能，是基于BOS数据处理框架V2.0生成的一套智能触发平台。开发者可基于平台提供的功能，定义业务关注的Bucket中发生的数据行为，BOS会监测Bucket中发生的Put Object、Append Object等行为，向用户的下游业务系统提供触发通知。

同时，BOS结合了百度智能云上强大的AI能力，支持基于监测到的用户行为，直接向百度智能云上的各项数据处理能力进行触发，并将处理后的结果直接给到用户下游业务系统。简化用户的开发逻辑，为业务提速的同时也赋予用户更强的数据AI处理能力。



支持地域

目前仅支持北京、苏州、广州，更多地域支持请提交工单。

操作方式

您可以通过 Console 控制台使用事件通知功能，详情请参见 事件通知及数据处理。

数据迁移上云

场景概述

数据上云的过程，可以分为“本地数据上云”、“从其他云迁移到BOS”两种场景，每种场景下都有存量和增量数据。百度智能云对象存储BOS提供了多种数据迁移方式，在数据一致性及无缝切换的前提下，尽量减少额外开销。您可以根据自身的业务情况，灵活选择数据迁移方式。

迁移场景	数据类型	迁移方案
本地数据上云	存量数据迁移上云	BOSCMD 工具在线迁移
		月光宝盒物理寄送
		本地数据物理寄送
	增量数据迁移上云	BOSCMD 工具在线迁移
从其他云迁移	存量数据迁移上云	CloudFlow可视化在线迁移
		BOS Import 工具
		控制台在线数据导入功能
	增量数据	镜像回源

本地数据上云迁移方案详见:https://cloud.baidu.com/doc/BOS/s/nk1rl0jnb

从其他云迁移到bos方案详见:https://cloud.baidu.com/doc/BOS/s/Rk1rl2if5

本地数据上云

存量数据迁移上云

存量数据是指您此时已经产生的数据。该数据的特点是数据量一般比较大，迁移需要较长的时间，但数据基本固定，很少再进行修改。本地数据上云时，您可以根据自身网络条件选择在线方式或离线方式进行迁移。

方式1：BOSCMD工具在线迁移

本地数据上云的在线方式适用于客户本地网络条件较好（带宽充裕）的情况。在这种情况下，我们推荐通过 BOS CMD 工具的Object上传功能实现数据上云。

BOS CMD 是BOS提供的命令行工具，它提供了丰富的功能，方便用户对BOS资源进行各种操作和管理。 BOS CMD 支持批量上传、同步上传(sync)、单文件上传三种Object上传方式，使用批量上传和同步上传(sync)可以高效的解决本地存量数据在线上云的问题。在这里先介绍批量上传方式：

您可以在已安装了 BOS CMD 的机器上执行以下命令来使用批量上传：

```
$ bcecmd bos cp <local-path> <bos-path> --recursive [--storage-class [STORAGE_CLASS]] [--restart] [--quiet] [--yes] [--disable-bar]
```

其中 local-path 为本地上传文件的目录。具体操作及注意事项请参见：BOS CMD Object 上传功能

方式2：月光宝盒物理寄送

月光宝盒是百度智能云 BOS 提供的一种 TB 级数据传输解决方案，它使用物理存储设备在百度智能云和客户 IDC 之间通过物流方式来传输大量数据。它解决了大规模数据传输中出现的高昂网络成本、较长传输时间和安全等问题，且具有简单、快速、安全、成本低的优点。

使用场景

- 您的网络环境不好（带宽小、网络不稳定），且带宽成本非常昂贵；
- 希望尽快完成数据的上云，但是专线方式的费用过于昂贵；
- 自己没有硬盘或者无可移动的数据存储和传输设备；

月光宝盒由 BOS 为您提供移动存储设备，该设备具有大容量、可视化操作、军工级别防护加密等特性。单台月光宝盒设备的标定容量为 96 TB，实际使用容量为 83 TB 左右，适用于百 TB 级别的本地 IDC 数据迁移上云场景。

使用方式

您可以通过工单联系我们，也可以通过您的客户经理联系我们。我们将根据当前的设备使用情况，为您匹配合适的使用时间和使用周期。请您告知我们您的预计数据量（如 300TB）、IDC 所在城市、预期使用时间等信息。

产品介绍

月光宝盒详细介绍、操作流程及注意事项请参见月光宝盒介绍和月光宝盒操作流程

方式3：自有硬盘物理寄送

BOS 还提供了一种离线方式————自有硬盘寄送。它和"月光宝盒"的不同之处在于，设备需要由客户自行准备。您可以将数据写入自有硬盘，将硬盘批量寄送到百度智能云机房，我们将为您免费传输上云。该方案适用于百 TB 级别的本地 IDC 数据迁移上云场景。BOS 在机房有多台固定设备用于硬盘方式上云，每台设备最多支持 16 块硬盘同时挂载，同时上云。相比"月光宝盒"，自有硬盘寄送的方式速度更快。

使用场景

自有硬盘寄送的方式适用于客户拥有较多硬盘，且需要迫切上云的场景，如：

- 本地 IDC 机房需要清退，数据需要尽快寻找云上合适的存储位置；
- 本地数据规模较大，希望尽快将数据备份上云。但专线方式花销较大，希望寻求其他性价比更高的方式；

使用限制

- 仅支持国标 3.5 寸或者 2.5 寸移动硬盘；
- 仅支持 USB 接口硬盘。很多大容量 3.5 寸硬盘只有 SATA 接口，您需要自行购买相应的硬盘盒，将硬盘盒和硬盘一同寄送到百度机房。硬盘盒可以通过各大网上商城购买，购买时搜索 "SATA 转 USB 硬盘盒"即可。一块硬盘在上传时需要匹配一个硬盘盒，请根据自身硬盘数量购买相应数量的硬盘盒。若硬盘盒数量小于硬盘数量，那么只能等待当前硬盘盒上硬盘数据上传完成后，再进行下一批次硬盘的数据拷贝上云；
- 仅支持 ext4、ntfs 和 xfs 文件系统。您可以通过 man 5 fs 查看 linux 支持的文件系统详细信息；
- 数据拷贝完成后，您寄来的硬盘和硬盘盒会全部给您寄回。寄回前，**硬盘内全部数据会格式化。若数据较为重要，请提前做好数据备份**；
- 硬盘到百度机房后，技术人员会立即全盘检测硬盘是否有坏道。**若有坏道且导致无法读取，该硬盘会直接格式化后寄回**；

使用费用

- 硬盘寄送的全部运费和保价费用需要客户自行承担，包括硬盘寄送到百度机房的物流费用，以及上传完成后寄回的物流费用。上传完成后，BOS 会选择到付方式寄回硬盘；
- 建议您对硬盘进行保价；
- 数据上传时所产生的机房服务器使用费、网络带宽费用、技术人员支持服务等全部免费；

使用流程

第一步：若您使用自有硬盘寄送方式上云，请先通过工单或客户经理联系我们，并告知您的数据量（如共 30 TB ）、硬盘数量（如 30 块移动硬盘）、是否含硬盘盒（若为 SATA 接口硬盘，请提前自行准备硬盘盒），硬盘预计到达百度机房时间、希望多久内上传完成等信息；

第二步：您需要将数据拷贝到硬盘，并做好数据检查和重要数据的数据备份；

第三步：将硬盘和硬盘盒（若含）寄送到如下地址：

地址	**北京市房山区窦店镇广茂路37号 光环新网数据中心**
电话	18110062722
联系人	BJDD 机房

第四步：通过您的客户经理或者工单告知我们您需要上传的 Bucket 名称、期望的目录结构和您的 AK/SK，我们将为您进行数据上传操作。建议您自行提前创建 Bucket 并创建子目录，并创建新的 AK/SK，不要使用默认 AK/SK；

第五步：BOS 技术人员收到硬盘，检查无误后开始数据上传。上传完成后会通过工单或客户经理与您确认文件数量，确认后**硬盘全部数据会进行格式化**，之后会将硬盘和硬盘盒（若含）为您寄回；

第六步：上传完成，您可以使用 BOS 控制台或者 API、SDK 等方式对数据进行管理和使用，并将 之前提供的 SK/SK 删除；

增量数据迁移上云

增量数据是指您此时正在产生或未来产生的数据。该数据一般是您业务产生的在线数据，具有实时性。

方式1：BOSCMD工具在线迁移

BOSCMD 工具的同步上传(sync)功能可以方便快捷的解决本地数据上云过程中，增量数据的问题。同步上传默认支持批量操作，可以将本地目录同步到 BOS。如果 BOS 端已经有名称相同且修改时间比本地文件新，同步上传会忽略此文件，只同步新文件或有改动的文件。 sync 命令会同时列举本地和 BOS 侧文件, 对于不同情况分别执行不同的动作，以保证增量数据的准确安全上传。

您可以在已安装了 BOS CMD 的机器上执行以下命令来使用同步上传(sync)：

```
 $ bcecmd bos sync <local_dir> bos:/<bucket_name>/[prefix] [--exclude EXCLUDE] [--include INCLUDE] [--delete] [--exclude-delete EXCLUDE-DELETE] [--dryrun] [--yes] [--quiet] [--storage-class STORAGE-CLASS] [--sync-type SYNC-TYPE] [--concurrency CONCURRENCY] [--restart]
```

其中 local_DIR 为本地同步目录。具体操作及注意事项请参见：BOS CMD 同步上传(sync)功能

从其他云迁移

存量数据跨云迁移

针对客户从其他云迁移到BOS的场景，百度智能云对象存储BOS提供了三种迁移方式，分别是 CloudFlow 数据流转平台、BOS Import 工具和控制台在线数据导入功能。

- **数据流转平台 CloudFlow（推荐使用）**：是百度智能云存储推出的存储数据全托管迁移平台，支持对存量和增量数据发起跨云迁移和跨云同步；
- **BOS Import 工具**：可靠、安全、稳定的脚本化迁移工具，需要下载并安装到本地后使用，适用于 1 TB 以上的数据迁移；
- **BOS 镜像回源**：您可以为存储桶 Bucket 配置镜像回源。当用户访问该 Bucket 中某一 Object 时，若发现该 Object 在 BOS 中并不存在，则 BOS 会立刻向预先配置好的回源地址请求该 Object，保证访问正常进行。在请求过程中，源站返回的数据会自动给到请求用户而用户对该过程无感。同时，该数据也会自动存入 BOS，提升日后该类请求的流畅性；

方式1：CloudFlow数据流转平台（推荐使用）

数据流转平台 CloudFlow 是百度智能云存储推出的存储数据全托管迁移平台，通过可视化配置的方式实现三方云到百度智能云的数据迁移。支持的迁移类型为三方对象存储数据到百度智能云对象存储，并支持对迁移任务的进度和结果进行查看。对失败的迁移任务，支持发起迁移重试，并支持对迁移任务发起中断、删除等操作。

数据流转平台支持如下几方面的功能：

- 任务创建：启动一个数据迁移进程前，需要先创建相应的迁移任务；
- 支持源端：数据流转平台支持由阿里云 OSS、腾讯云 COS、金山云 KS3、七牛云 KODO、华为云 OBS 、AWS 中国区、本地自建 S3 集群移到百度智能云对象存储 BOS；
- 任务管理：当迁移任务创建后，您可以对任务进行管理，如中断、删除、下载失败文件、重启等操作；
- 任务进度查看：迁移任务创建后，您可以对任务的迁移进度进行查看，如当前已经完成迁移的文件数、完成中成功迁移的文件数、失败的文件数、待迁移的文件数等等；
- 支持存量数据和增量迁移的迁移；

您可以登录百度智能云 BOS 控制台，点击左侧导航中的数据迁移来创建迁移任务，也可以直接登录CloudFlow控制台创建迁移任务并发起数据的迁移。详细迁移步骤请参考CloudFlow帮助文档。

方式2：BOS Import工具

BOS Import 工具是百度智能云对象存储BOS推出的用于将其它云存储的数据迁移到BOS的工具，具有"可拓展性良好"、"支持迁移任务热启停"等多种优点。工具基于C-S结构实现，client端负责读取文件列表, 任务调度和进度统计等工作, 所有和迁移相关的参数/配置都在client端进行；迁移任务在server端完成, client端通过RPC请求通知server对哪些文件进行迁移。

具体使用方式及注意事项请参见：BOS Import 工具

增量数据跨云同步

您可以通过配置镜像回源规则或 CloudFlow 数据流转平台来完成增量数据的跨云同步。

方式1：CloudFlow数据流转平台

数据流转平台 CloudFlow 是百度智能云存储推出的存储数据全托管迁移平台，通过可视化配置的方式实现三方云到百度智能云的数据迁移，支持对存量数据和增量数据发起跨云迁移和跨云同步。

方式2：镜像回源

镜像回源是一项Bucket级别的配置，主要用于帮助用户在不停服的情况进行数据热迁移，解决从其他云迁移到BOS的增量数据问题。

对一个Bucket配置了镜像回源后，当用户访问（GetObject）一个BOS中的Object，BOS发现该Object在BOS中并不存在时，BOS会向回源地址请求该Object，并将源站所返回数据一边返回给请求用户，一边存入BOS。

具体使用方式及注意事项请参见：BOS镜像回源

静态网站托管

概述

BOS支持用户将静态网站托管在Bucket上，实现网站轻量化运维。设置生效后，用户可通过直接访问该Bucket的域名访问该托管网站。

静态网站是指网站内容全部为HTML、JPG等静态化资源，不包含诸如 PHP、JSP 或 ASP.NET 等服务器端脚本。同时，静态托管的网站也不支持在服务器端进行脚本编写。

如需部署及管理动态网站，请使用 百度智能云服务器 BCC 或 容器服务 CCE。

使用方式

您可以通过 Console 控制台、API 和 SDK 来设置静态网站托管，具体如下：

- 通过 Console 控制台设置静态网站托管，请参考 设置静态网站托管。
- 通过 API 设置静态网站托管：
  - PutBucketStaticWebsite 接口：使用 PutBucketStaticWebsite API 接口设置静态网站托管。
  - GetBucketStaticWebsite 接口：使用 GetBucketStaticWebsite API 接口获取 Bucket 的静态网站托管信息。
  - DeleteBucketStaticWebsite 接口)：使用 DeleteBucketStaticWebsite API 接口删除 Bucket 设置的静态网站托管信息，并关闭此 Bucket 的静态网站托管。
- 使用 SDK 设置静态网站托管：
  - JAVA SDK
  - PHP SDK

图像审核

概述

图像审核服务是百度智能云对外提供的图像智能审核服务，支持对图像的多个维度进行审核，包括色情识别、暴恐识别、政治敏感识别、恶心图识别、广告识别等。图像审核应用场景广泛，例如游戏、社交、论坛、生活服务类、UGC等场景。如游戏厂商需要对玩家上传的自定义头像进行审核，就可以借助图像审核服务对头像进行审核。

图像审核服务和BOS进行了深度集成，您可以通过BOS API、SDK、CLI等方式调用图像审核能力。

计费方式

作为图像审核服务的调用入口，BOS仅收取BOS API的请求费用，详细信息请参考BOS产品定价。

图像审核服务的费用由图像审核产品收取，详细信息请参考图像审核产品文档。

开通图像审核服务

目前百度智能云支持对BOS上的图像进行审核，支持主动调用图像审核API接口与图像自动审核，图像自动审核功能，您需先通过控制台开通图像审核服务，图审规则生效后，您新上传的文件将自动被审核，您可以通过控制台、SDK、CLI等方式上传Object到BOS上。

1. 控制台开通图像审核服务，参考控制台设置图像审核服务。
2. 开通图像审核服务后用户可以通过BOS CLI工具或者BOS API接口调用图像审核功能，下文详细介绍API接口。

API接口

此接口用于请求图像审核服务。

> **说明**：使用本接口需要先在控制台开通图像审核服务。

请求

- 请求语法

```
 POST <ObjectName>?process HTTP/1.1
Host: <BucketName>.bj.bcebos.com
Date: <Date>
Authorization: <AuthorizationString>
Content-Type: application/json; charset=utf-8
Content-Length: <ContentLength>

{
 "action" : {
  "sync" : [{
    "url" : "$(img-censor)",
    "parameters" : "<base64_encode(param)>",

  }
  ]
 }
}
```

- 请求参数
  - url：固定取值为`$(img-censor)`，不需要修改。
  - parameters：此项值是对图像审核功能参数进行base64编码而成。图像审核功能参数依然是个json字符串。

    图像审核功能参数基本结构：

    ```
     {
      "antiporn" : {},
      "terror" : {},
      "ocr" : {
          "detect_direction" : "false",
          "language_type" : "CHN_ENG",
          "recognize_granularity" : "big"
      }
     }
    ```

    可以为多个子服务的按需任意组合，也可以为单个子服务如“鉴黄”，此时参数写成

    ```
     {
       "antiporn" : {}
     }
    ```

    图像审核服务支持的子服务列表有：
    - ocr：通用文字识别
    - face：人脸探测
    - antiporn：色情识别
    - politician：政治人物识别
    - terror：暴恐识别
    - public 公众人物识别
    - disgust 恶心图识别
    - watermark：水印二维码识别
    - quality：图像质量识别 value为各底层服务参数，参照河图OCR通用文字识别服务入参。

    详细的子服务请求参数及返回参数解释参见下文详细解释。
- 请求头域

  无

响应

- 响应头域 无
- 响应元素

  参见各子服务的返回参数解释。

示例

- 请求示例

```
 {
   "antiporn": {},
   "ocr": {
       "detect_direction": "false",
       "language_type": "CHN_ENG",
       "recognize_granularity": "big"
   }
}
```

base64编码后

```
eyJhbnRpcG9ybiI6e30sIm9jciI6eyJkZXRlY3RfZGlyZWN0aW9uIjoiZmFsc2UiLCJsYW5ndWFnZV90eXBlIjoiQ0hOX0VOR1lsInJlY29nbml6ZV9ncmFudWxhcmml0eSI6ImJpZyJ9fQ==
```

填充到parameters中，发送的请求如下：

```
 POST <ObjectName>?process HTTP/1.1
Host: <BucketName>.bj.bcebos.com
Date: <Date>
Authorization: <AuthorizationString>
Content-Type: application/json; charset=utf-8
Content-Length: <ContentLength>

{
 "action" : {
  "sync" : [{
    "url" : "$(img-censor)",
    "parameters" : "eyJhbnRpcG9ybiI6e30sIm9jciI6eyJkZXRlY3RfZGlyZWN0aW9uIjoiZmFsc2UiLCJsYW5ndWFnZV90eXBlIjoiQ0hOX0VORyIsInJlY29nbml6ZV9ncmFudWxhcml0eSI6ImJpZyJ9fQ==",
   }
  ]
 }
}
```

- 成功响应示例

```
  HTTP/1.1 200 OK
Date: Thu, 22 Jun 2017 07:30:56 GMT
Content-Type: application/json; charset=utf-8
Content-Length: 237
Connection: keep-alive
Server: BceBos
x-bce-debug-id: MTAuNzUuNzguNDA6VGh1LCAyMiBKdW4gMjAxNyAxNTozMDo1NiBDU1Q6MTg1MDY5NDg3OQ==
x-bce-request-id: 598f7e18-77fb-424a-bc68-95acb0644076

{
 "result" : {
  "antiporn" : {
   "result" : [{
     "probability" : 0.000071,
     "class_name" : "色情"
    }, {
     "probability" : 0.000291,
     "class_name" : "性感"
    }, {
     "probability" : 0.999638,
     "class_name" : "正常"
    }
   ],
   "log_id" : 1853911322,
   "result_num" : 3
  },
  "ocr" : {
   "log_id" : 2471272194,
   "words_result_num" : 2,
   "words_result" : [{
     "words" : " TSINGTAO"
    }, {
     "words" : "青岛"
    }
   ]
  }
 },
 "log_id" : 149811665151162
}
```

- 失败返回示例

```
  {
    "log_id": 149319909347709,
    "error_code": 216500,
    "error_msg": "unknown error"
}
```

失败error_code解释：

错误码	错误信息	描述
216101	not enough param	参数不足
216102	service not support	输入了不支持的底层服务类型
216200	empty imge	没有图片Url
216500	unknown error	未知错误
282804	download image error	图片下载失败
282000	logic internal error	业务逻辑层错误

ocr

功能

用户向服务请求识别某张图中的所有文字。

请求参数

参数	是否必选	类型	可选值范围	说明
language_type	否	string	CHN_ENG、ENG、POR、FRE、GER、ITA、SPA、RUS、JAP、KOR	识别语言类型，默认为CHN_ENG。可选值包括：- CHN_ENG：中英文混合；- ENG：英文；- POR：葡萄牙语；- FRE：法语；- GER：德语；- ITA：意大利语；- SPA：西班牙语；- RUS：俄语；- JAP：日语；- KOR：韩语
detect_direction	否	boolean	true、false	是否检测图像朝向，默认不检测，即：false。朝向是指输入图像是正常方向、逆时针旋转90/180/270度。可选值包括:- true：检测朝向；- false：不检测朝向
detect_language	否	string	true、false	是否检测语言，默认不检测。当前支持（中文、英语、日语、韩语）
probability	否	string	true、false	是否返回识别结果中每一行的置信度

返回参数

字段	是否必选	类型	说明
direction	否	int32	图像方向，当detect_direction=true时存在。- -1:未定义，- 0:正向，- 1: 逆时针90度，- 2:逆时针180度，- 3:逆时针270度
log_id	是	uint64	唯一的log id，用于问题定位
words_result	是	array()	识别结果数组
words_result_num	是	uint32	识别结果数，表示words_result的元素个数
+words	否	string	识别结果字符串
probability	否	object	识别结果中每一行的置信度值，包含average：行置信度平均值，variance：行置信度方差，min：行置信度最小值

返回示例

```
 {
"log_id": 2471272194,
"words_result_num": 2,
"words_result":
   [
      {"words": " TSINGTAO"},
      {"words": "青岛啤酒"}
   ]
}
```

face

功能

- 检测请求图片中的人脸，返回人脸位置、72个关键点坐标、及人脸相关属性信息。
- 检测响应速度，与图片中人脸数量相关，人脸数量较多时响应时间会有些许延长。
- 典型应用场景：如**人脸属性分析**，**基于人脸关键点的加工分析**，**人脸营销活动**等。
- 五官位置会标记具体坐标；72个关键点坐标也包含具体坐标，但不包含对应位置的详细位置描述。

请求参数

无

返回参数

参数	类型	是否必须	说明
log_id	uint64	是	日志id
result_num	uint32	是	人脸数目
result	object[]	是	人脸属性对象的集合
+age	double	否	年龄。face_fields包含age时返回
+beauty	double	否	美丑打分，范围0-100，越大表示越美。face_fields包含beauty时返回
+location	object	是	人脸在图片中的位置
++left	uint32	是	人脸区域离左边界的距离
++top	uint32	是	人脸区域离上边界的距离
++width	uint32	是	人脸区域的宽度
++height	uint32	是	人脸区域的高度
+face_probability	double	是	人脸置信度，范围0-1
+rotation_angle	int32	是	人脸框相对于竖直方向的顺时针旋转角，[-180,180]
+yaw	double	是	三维旋转之左右旋转角[-90(左), 90(右)]
+pitch	double	是	三维旋转之俯仰角度[-90(上), 90(下)]
+roll	double	是	平面内旋转角[-180(逆时针), 180(顺时针)]
+expression	uint32	否	表情，0，不笑；1，微笑；2，大笑。face_fields包含expression时返回
+expression_probability	double	否	表情置信度，范围0~1。face_fields包含expression时返回
+faceshape	object[]	否	脸型置信度。face_fields包含faceshape时返回
++type	string	是	脸型：square/triangle/oval/heart/round
++probability	double	是	置信度：0~1
+gender	string	否	male、female。face_fields包含gender时返回
+gender_probability	double	否	性别置信度，范围0~1。face_fields包含gender时返回
+glasses	uint32	否	是否带眼镜，0-无眼镜，1-普通眼镜，2-墨镜。face_fields包含glasses时返回
+glasses_probability	double	否	眼镜置信度，范围0~1。face_fields包含glasses时返回
+landmark	object[]	否	4个关键点位置，左眼中心、右眼中心、鼻尖、嘴中心。face_fields包含landmark时返回
++x	uint32	否	x坐标
++y	uint32	否	y坐标
+landmark72	object[]	否	72个特征点位置，示例图 。face_fields包含landmark时返回
++x	uint32	否	x坐标
++y	uint32	否	y坐标
+race	string	否	yellow、white、black、arabs。face_fields包含race时返回
+race_probability	double	否	人种置信度，范围0~1。face_fields包含race时返回
+qualities	object	否	人脸质量信息。face_fields包含qualities时返回
++occlusion	object	是	人脸各部分遮挡的概率， [0, 1] （待上线）
+++left_eye	double	是	左眼
+++right_eye	double	是	右眼
+++nose	double	是	鼻子
+++mouth	double	是	嘴
+++left_cheek	double	是	左脸颊
+++right_cheek	double	是	右脸颊
+++chin	double	是	下巴
++blur	double	是	人脸模糊程度，[0, 1]。0表示清晰，1表示模糊（待上线）
++illumination	-	是	取值范围在[0,255],表示脸部区域的光照程度（待上线）
++completeness	-	是	人脸完整度，[0, 1]。0表示完整，1表示不完整（待上线）
++type	object	是	真实人脸/卡通人脸置信度
+++human	-	是	真实人脸置信度，[0, 1]
+++cartoon	-	是	卡通人脸置信度，[0, 1]

返回示例

```
{
  "result_num": 1,
  "result": [
    {
      "location": {
        "left": 90,
        "top": 92,
        "width": 111,
        "height": 99
      },
      "face_probability": 1,
      "rotation_angle": 6,
      "yaw": 11.61234664917,
      "pitch": -0.30852827429771,
      "roll": 8.8044967651367,
      "landmark": [
        {
          "x": 105,
          "y": 110
        },
        ...
      ],
      "landmark72": [
        {
          "x": 88,
          "y": 109
        },
        ...
      ],
      "gender": "male",
      "gender_probability": 0.99358034133911,
      "glasses": 0,
      "glasses_probability": 0.99991309642792,
      "race": "yellow",
      "race_probability": 0.99960690736771,
      "qualities": {
        "occlusion": {
          "left_eye": 0.000085282314103097,
          "right_eye": 0.00001094374601962,
          "nose": 3.2677664307812e-7,
          "mouth": 2.6582130940866e-10,
          "left_cheek": 8.752236624332e-8,
          "right_cheek": 1.0212766454742e-7,
          "chin": 4.2632994357028e-10
        },
        "blur": 4.5613666312237e-41,
        "illumination": 0,
        "completeness": 0,
        "type": {
          "human": 0.98398965597153,
          "cartoon": 0.016010366380215
        }
      }
    }
  ],
  "log_id": 2418894422
}
```


antiporn

功能

色情识别。

请求参数

无

返回参数

字段	类型	是否必须	说明
confidence_coefficient	string	是	结果是否确定，分为“确定”和“不确定”两种。
result_num	uint32	是	返回结果数目，即：result数组中元素个数。
result	array(array(double))	是	结果数组，每项内容对应一个分类维度的结果。
conclusion	string	是	本张图片最终鉴定的结果，分为“色情”，“性感”，“正常”三种。
log_id	uint64	是	请求标识码，随机数，唯一。

其中元素的每项内容包含以下字段：

字段	类型	是否必须	说明	示例
class_name	string	是	分类结果名称	色情
probability	double	是	分类结果置信度	0.8947165012 3596

返回示例

```
{
"result": [{
 "probability": 0.000301,
 "class_name": "色情"
},
{
 "probability": 0.000054,
 "class_name": " 性感"
},
{
 "probability": 0.999645,
 "class_name": "正常"
}],
"conclusion": "正常",
"log_id": 848999404,
"confidence_coefficient": "确定",
"result_num": 3
}
```


politician

功能

政治人物识别。

请求参数

无

返回参数

参数	子参数	必须	类型	是否必须	说明
log_id	-	-	uint64	是	日志id
result_num	-	-	uint32	是	实际检测到人脸数目(不大于max_face_num)
result	-	-	object[]	是	
-	location	-	object	是	人脸在输入图片中的位置
-	-	left	uint32	是	人脸区域离左边界的距离
-	-	top	uint32	是	人脸区域离上边界的距离
-	-	width	uint32	是	人脸区域的宽度
-	-	height	uint32	是	人脸区域的高度
-	stars	-	object[]	是	公众人物数组
-	-	name	string	是	姓名
-	-	star_id	string	是	人物id，全局唯一
-	-	probability	float	是	相似度，[0,1]

返回示例

```
{
  "log_id": 3268660173,
  "result_num": 1,
  "result": [
    {
      "location": {
        "left": 132,
        "top": 168,
        "width": 238,
        "height": 223
      },
      "stars": [
        {
          "name": "张三",
          "star_id": "515617",
          "probability": 0.9750030040741
        }
      ]
    }
  ]
}
```


terror

功能

暴恐识别

请求参数

无

返回参数

字段	类型	是否必须	说明
result	array(array(double))	是	暴恐置信度得分
log_id	uint64	是	请求标识码，随机数，唯一
result_coarse	object[]	是	粗粒度得分结果
name	string	是	粗粒度标签，包含两个标签：正常、暴恐
score	float	是	对应标签的置信度得分，越高可信度越高
result_fine	object[ ]	是	细粒度得分结果
name	string	是	细粒度标签，包含9个标签：正常、警察部队、血腥、尸体、爆炸火灾、杀人、暴乱、暴恐人物、军事武器
score	float	是	对应标签的置信度得分，越高可信度越高

返回示例

```
{
  "errno": 0,
  "msg": "success",
  "data": {
    "result": 0.0082325544208288,
    "result_coarse": [
      {
        "name": "正常",
        "score": 0.99176746606827
      },
      {
        "name": "暴恐",
        "score": 0.0082325544208288
      }
    ],
    "result_fine": [
      {
        "name": "正常",
        "score": 0.98908758163452
      },
      {
        "name": "警察部队",
        "score": 0.0062405453063548
      },
      {
        "name": "血腥",
        "score": 0.0009653537417762
      },
      {
        "name": "尸体",
        "score": 0.001054480439052
      },
      {
        "name": "爆炸火灾",
        "score": 0.00011743687355192
      },
      {
        "name": "杀人",
        "score": 0.0011699661845341
      },
      {
        "name": "暴乱",
        "score": 0.0000211903588933 64
      },
      {
        "name": "暴恐人物",
        "score": 0.0010401027975604
      },
      {
        "name": "军事武器",
        "score": 0.00030337597127073
      }
    ]
  }
}
```


public

功能

公众人物识别

请求参数

参数	是否必选	类型	说明
max_face_num	否	uint32	最多处理人脸数目，默认值1，最大值5
max_star_num	否	uint32	单人脸最多相似明星数，默认4

返回参数

参数	子参数	子参数	类型	必须	说明
log_id	-	-	uint64	是	日志id
result_num	-	-	uint32	是	实际检测到人脸数目(不大于max_face_num)
result	-	-	object[]	是	
-	location	-	object	是	人脸在输入图片中的位置
-	-	left	uint32	是	人脸区域离左边界的距离
-	-	top	uint32	是	人脸区域离上边界的距离
-	-	width	uint32	是	人脸区域的宽度
-	-	height	uint32	是	人脸区域的高度
-	stars	-	object[]	是	公众人物数组
-	-	name	string	是	姓名
-	-	star_id	string	是	人物id，全局唯一
-	-	probability	float	是	相似度，[0, 1]

返回示例

```
{
  "log_id": 3268660173,
  "result_num": 1,
  "result": [
    {
      "location": {
        "left": 132,
        "top": 168,
        "width": 238,
        "height": 223
      },
      "stars": [
        {
          "name": "张三",
          "star_id": "515617",
          "probability": 0.9750030040741
        }
      ]
    }
  ]
}
```

disgust

功能

恶心图识别

请求参数

无

返回参数

参数	类型	必须	说明
log_id	uint64	是	请求标识码，随机数，唯一
result	double	是	得分

返回示例

```
{
"result": 9.2708455667889E-7,
"log_id": 2977989308
}
```

watermark

功能

水印检测

请求参数

无

返回值

参数	类型	必须	说明
log_id	uint64	是	请求标识码，随机数，唯一
result_num	uint32	否	返回结果数目，即：result数组中元素个数
result	array(object)	否	返回结果数组，每一项为一个检测出的结果

result中元素的每项内容包含以下字段：

参数	类型	必须	说明
location	object	否	位置信息（左起像素位置、上起像素位置、像素宽、像素高）
probability	double	是	分类结果置信度[0-1.0]
type	string	是	返回结果的类型 (watermark、bar code、QR code)

返回示例

```
 {
"result": [{
 "probability": 0.99872654676437,
 "type": "watermark"
},
{
 "probability": 0.98578763008118,
 "type": "watermark"
}],
"log_id": 686882979,
"result_num": 2
}
```

quality

功能

图片质量

请求参数

无

返回参数

参数	子参数	类型	必须	说明
log_id	-	uint64	是	日志id
result	-	object[]	是	
-	aesthetic	double	是	美观度
-	clarity	double	是	清晰度

返回示例

```
 {
"result": {
 "aesthetic": 0.26410301526388,
 "clarity": 0.28039423624674
},
"log_id": 93316007
}
```

单链接限速

单链接限速

对象存储BOS对单Bucket的公网带宽阈值为10Gbit/s，内网带宽阈值为50Gbit/s，当用户的上传或下载占用带宽达到带宽限制阈值时，会返回RequestRateLimitExceeded的错误码。为保证用户能够正常使用服务，BOS支持在进行上传、下载等行为时进行流量控制，保证大流量服务占用带宽不会对其他应用服务造成影响。

使用方法

用户在使用以下接口时

- 上传类请求接口：PutObject、AppendObject、PostObject、UploadPart、UploadPartCopy；
- 复制类请求接口：CopyObject；
- 下载类请求接口：GetObject。

可以在请求header、请求参数中携带x-bce-traffic-limit参数，并指定限速值，限速值的取值范围为819200~838860800，单位为bit/s，即100KB/s~100MB/s。限速值取值必须为数字，BOS将按照指定的限速值对此次请求进行限速，当限速值不在此范围或不合法时将返回400错误码。

上传请求示例

- 在请求header中指定上传的限速值为819200bit/s，即100KB/s：

```
 PUT /<ObjectName> HTTP/1.1
Host: <BucketName>.bj.bcebos.com
Date: <Date>
Authorization: <AuthorizationString>
Content-Type: text/plain
Content-Length: <Content_Length>
x-bce-storage-class: <Storage_class>
x-bce-traffic-limit: 819200

[11434 bytes of object data]
```

- 在请求参数中指定上传的限速值为819200bit/s，即100KB/S：

```
 PUT /<ObjectName>?x-bce-traffic-limit=819200 HTTP/1.1
Host: <BucketName>.bj.bcebos.com
Date: <Date>
Authorization: <AuthorizationString>
Content-Type: text/plain
Content-Length: <Content_Length>
x-bce-storage-class: <Storage_class>

[11434 bytes of object data]
```

下载请求示例

- 在请求header中指定下载的限速值为819200bit/s，即100KB/s：

```
 GET /<ObjectKey> HTTP/1.1
Host: <BucketName>.bj.bcebos.com
Date: <Date>
Authorization: <AuthorizationString>
Range: <Range_String>
x-bce-traffic-limit: 819200
```

- 在请求参数中指定下载的限速值为819200bit/s，即100KB/S：

```
https://<BucketName>.bj.bcebos.com/<ObjectName>?x-bce-traffic-limit=819200
```

> 注意 对于私有文件，在生成带签名的外链时，需要将限速参数`x-bce-traffic-limit=819200`加入签名中一起计算。

数据湖接入

分层命名空间

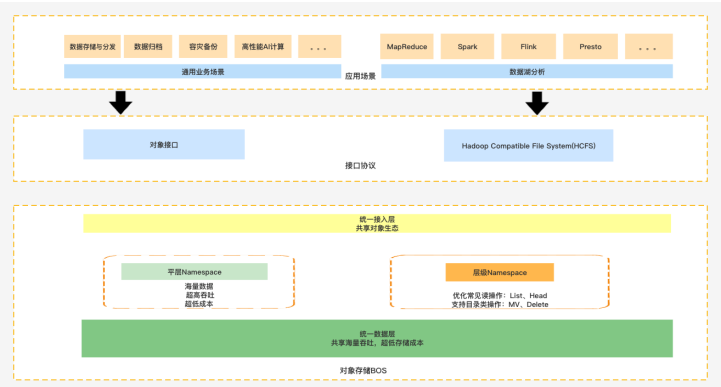
功能概述：

分层命名空间（Namespace）是百度智能云推出的元数据管理功能。通过分层命名空间，可以很好的支持目录级别的 Mv 与 Rename 操作，同时优化常见读操作 List、Head，提升数据处理效率与性能，能很好的满足大数据领域的数据湖计算场景。

功能开通：

- 第一步：可提交工单说明需求，系统将为您开通使用权限。
- 第二步：获取权限后，通过 BOS 工具或登录控制台创建需开启分层命名空间的存储桶。

功能架构：



优势：

1. 大数据场景下也可以直接享受对象存储带来的自动化数据管理，存储低成本，数据强安全优势
2. 原生基于对象的分层 Namespace 带来了大数据计算场景下对于文件语义的高性能表现
   - 支持高性能原子 Rename/Delete Dir/Object
   - 远低于平层对象存储的 List/Head 请求时延
3. 直接基于对象存储的大数据分析（All in Object），节约额外的数据复制与流转、存储以及第三方服务费用
4. 无额外的运维，管理及学习成本，充分享受对象存储的生态及工具

应用场景：

大数据分析与计算

大数据分析和大数据计算。对象存储作为云端最佳存储方案，BOS 提供一整套高性能大数据计算解决方案。在大数据场景中，BOS 可以作为您本地自建或者云端 MapReduce 计算集群 、Hive、Spark 离线数仓集群的底层存储系统，原生分层命名空间能力，可以将大数据计算中的大规模批量操作如 Rename、List、Head 等操作的时延降低约 70%，突破传统对象存储支持大数据计算方案的延迟瓶颈。

使用方式：

目前分层命名空间功能支持华北-北京地域、华南-广州地域的新建存储桶（Bucket）开启分层命名空间功能，开启后不支持关闭该功能。控制台开启分层命名空间的具体操作，详见控制台操作指南-分层命名空间存储桶操作说明。

您可以通过以下方式在开启了分层命名空间的 Bucket 中创建目录、并进行目录的重命名或删除操作。

方式	说明
各语言 SDK	丰富的 Demo
控制台	页面操作直观易用，方便操作
命令行工具（BOS CMD）	简单易用

功能支持与限制

支持

- 支持文件、目录语义和操作
- 目录原子性、毫秒级 Rename
- 优化读性能
- 支持平层对象存储绝大部分功能

限制

开启分层命名空间的存储桶支持功能与平层命名空间存储桶基本相同，均支持对象级别操作基本均支持比如：上传、下载、复制等；存储桶级别的配置功能如生命周期、事件通知、日志管理等也可正常使用，目前相比起平层命名空间存储桶仅有少部分功能暂不支持，若您有需求可提交工单反馈，具体说明如下：

Bucket 相关功能

1. 不支持回收站

Object 相关功能

1. 不支持归档存储类型文件
2. 不支持上传带内容的目录（即上传内容非空但文件名末尾是`/`的文件）
3. 不支持文件软链接功能
4. 不支持对象标签

接口调整

由于分层命名空间特性导致部分 BOS API 会有部分兼容和语义上的调整，如下所示：

ListObjects

对于 平层命名空间存储桶，分层命名空间下的 ListObjects API 的请求参数有所调整，具体说明参见下表，平层命名空间 ListObjects 详情请查看ListObjects接口说明。

字段	类型	是否必需	描述
delimiter	String	否	分隔符; 分层 Namespace 下 ListObjects 必须指定，且只能为`/`
marker	String	否	Object 为字母序排列，从 Marker 之后的第一个开始返回，为对象（文件）的全路径；分层命名空间下，将会返回父目录下的子文件或者子目录按照字典序大于 Marker 的结果
maxKeys	Int	否	返回 Object 列表长度最大为 1000，默认值是 1000，如果指定的值大于 1000，按 1000 操作
prefix	String	否	必须为对象的父目录，限定 ListObjects 接口只能返回以 Prefix 为父目录下的对象；Prefix 可以不以`/`结尾，BOS 会默认为其自动添加目录分隔符

特别说明

> ListObjects 会将父目录下的所有子目录与对象按照字典序排序后返回；
>
> ListObjects 在分层空间存储桶的目录下不再支持列出某个前缀(目录)下的所有文件，只支持根据给出的父目录信息列出子目录和子文件。

元数据转换服务

概述

BOS元数据转换服务支持在未部署任何工具的情况下，直接将BOS普通存储桶转换为分层命名空间存储桶。分层命名空间存储桶在支持普通平层桶绝大部分特性和兼容大部分HDFS文件系统接口的同时，可以支持目录原子性Rename和Delete，同时优化了List和Head请求时延，能很好的满足大数据和AI领域的数据湖计算场景。

前提条件

1. 目前仅北京、苏州、广州、保定地域支持元数据转换服务和分层命名空间存储桶。
2. 需提交工单申请使用BOS元数据转换功能。

使用限制

1. 不支持转换开启回收站或多版本功能的bucket，不支持转换存在归档类型object、软链接Symlink、对象标签的bucket。

2. 元数据转换成功的平层bucket不再支持关闭分层命名空间，转换过程不可逆，转换前后只是元数据组织形式发生变化，数据不发生移动。

3. 元数据转换任务的完成时间取决于bucket中object数量和后端资源配额，暂不提供精确的时效性保证。

风险说明

- 转换任务开始之后，执行结束之前，尽量不进行增量写入和对存量的覆写及删除，以免遗漏。执行过程中遇到归档类型文件会导致任务失败并中止转换。
- 请确保待转换的BOS bucket中不存在同名文件和文件夹，例如同时存在文件dir以及文件夹dir/，转换过程中会产生冲突，且和任务运行时的导入顺序相关，服务将通过重命名操作来避免冲突，具体参考转换规则。
- 请参考转换规则说明，确保BOS bucket中不存在且不写入使用特殊保留文件名的文件，否则可能出现同名文件相互覆盖，造成文件丢失。
- bucket元数据转换开始前可以通过replication等方式先做好备份保存，元数据转换任务完成后请务必再次确认bucket中数据的完整性。

转换规则

- 对于转换前以/结尾的文件（例如a/b/），如果文件是空文件（size为0）则转换后直接创建对应的同名目录（例如a/b），否则其转换后会发生重命名加上特殊后缀.slash和.转换时间戳，例如文件a/b/会变成a/b.slash.1716487553，可以通过ListObjects查看转换后的对应新文件名。
- 分层命名空间存储桶中不允许同名文件和目录共存，因此如果转换前的存储桶中同时存在文件a/b和a/b/，其中a/b/表示一个目录，那么转换过程中会发生冲突，跟两个文件的转换顺序有相关性，第一原则是优先创建和保留目录。
  - 情况一：创建目录a/b时发现同名文件a/b已存在，则将文件重命名加上特殊后缀.file和.转换时间戳，例如文件a/b/会变成a/b.file.1716487553。如果新文件名已经存在，则继续添加次数后缀重试1次，例如a/b.file.1716487553.1。
  - 情况二：创建文件a/b时发现同名目录a/b已存在，尝试创建加上特殊后缀.file和.转换时间戳的新文件，例如a/b.file.1716487553；如果新文件名还是已经存在，则继续添加次数后缀重试3次，例如a/b.file.1716487553.1。

监控服务操作

监控服务概述

BOS监控服务基于 *云监控BCM* 来进行数据统计、展示、报警，提供系统的基础数据、流量数据、请求数据、图像处理数据等方面的监控数据指标。BOS监控指标分为基础数据、流量数据、请求数据、数据处理、服务监控，详见*监控指标*。

基本功能

云监控BCM为对象存储提供以下功能来实现监控与报警功能。

功能	能力	主要功能
监控信息	展示产品当前状态	提供基本信息以及各项指标监控数据
报警策略	提供报警功能与配置策略	支持通过多种通知渠道来对对象存储BOS实施监控和报警，用户可自定义配置报警策略
报警历史	展示历史报警记录	查看指定时间段报警历史以及统计该时间段异常数据

使用监控服务

监控具体信息可从云监控BCM的对象存储BOS bucket列表监控相关信息，包括监控信息、报警策略、报警历史。

1. 登录*云监控BCM控制台*。

2. 在对象存储BOS的bucket列表中单击bucket名称。

3. 进入BOS对象存储监控页面，查看用户监控详情。

- 基础数据

监控维度	监控项说明
标准存储-多AZ	该监控项下包含标准存储-多AZ占用空间大小、标准存储-多AZ文件数量、标准存储-多AZ读请求次数、标准存储-多AZ写请求次数、标准存储-多AZ读错误请求个数、标准存储-多AZ写错误请求个数、帐号下标准存储-多AZ总占用空间、帐号下标准存储-多AZ总文件数量
标准存储	该监控项下包括标准存储占用空间大小、标注存储文件数量、标准存储读请求次数、标准存储写请求次数、标准存储读错误请求次数、标准存储写错误请求次数
低频存储-多AZ	该监控项下包括低频存储-多AZ占用空间大小、低频存储-多AZ文件数量、低频存储-多AZ读请求次数、低频存储-多AZ写请求次数、低频存储-多AZ读错误请求个数、低频存储-多AZ写错误请求个数、低频存储-多AZ数据取回大小、帐号下低频存储-多AZ总占用空间、帐号下低频存储-多AZ总文件数量、低频存储-多AZ最小存储数据
低频存储	该监控项下包括低频存储占用空间大小、低频存储文件数量、低频存储读请求次数、低频存储写请求次数、低频存储读错误请求次数、低频存储写错误请求次数、低频存储数据取回大小、低频存储最小存储数据
冷存储	该监控项下包括冷存储占用空间大小、冷存储文件数量、冷存储读请求次数、冷存储写请求次数、冷存储读错误请求次数、冷存储写错误请求次数、冷存储数据取回大小、冷存储最小存储数据
归档存储	该监控项下包括归档存储占用空间大小、归档存储文件数量、归档存储读请求次数、归档存储写请求次数、归档存储读错误请求次数、归档存储写错误请求次数、归档存储标准取回字节数、归档存储最小存储数据、归档存储加急取回字节数、归档存储延缓取回字节数、归档存储加急取回次数、归档存储标准取回次数、归档存储延缓取回次数
智能存储	该监控项下包括标准转低频存储容量、低频转冷存储容量、冷存储上浮低频存储容量、低频上浮标准存储容量

- 流量数据

监控维度
外网读取量
外网写入量
每秒外网写数据总量
每秒外网读数据总量
每秒内网读数据量
每秒内网写数据量
CDN回源流量
跨区域复制流出流量
跨区域复制流入流量
同区域复制流出流量
同区域复制流入流量
复制总流出流量
复制总流入流量

- 请求数据

监控维度	监控项说明
读/写/删除统计	该监控项下包含每秒删除请求总次数、每秒读请求总次数、每秒写请求总次数
HttpCode统计	该监控项下包含返回状态码为2XX的请求数、返回状态码为3XX的请求数、返回状态码为4XX的请求数、返回状态码为5XX的请求数、返回状态码为408的请求数、返回状态码为499的请求数
请求延迟	该监控项下包含BOS后端平均处理时间、BOS后端平均写处理时间、BOS后端平均读处理时间、后端处理时间过长(1s)的写请求数、后端处理时间过长(1s)的请求总数、后端处理时间过长(1s)的读请求数、后端处理时间过长(5s)的请求总数、后端处理时间过长(5s)的写请求数、后端处理时间过长(5s)的读请求数
api统计	该监控项下包含Get Object请求总次数、List Object请求总次数、Head Object请求总次数、Put Object请求总次数、DeleteObject请求总次数、客户端断链请求占比、客户端超时请求占比
限流监控统计	该监控项下包含触发软限流总次数、触发硬限流总次数、触发限流警告总次数

- 图像处理数据

监控维度
图像基础处理用量
基础图像处理调用次数
Heic压缩（800x600以下尺寸）
Heic压缩（800x600以上尺寸）
Webp压缩（800x600以下尺寸）
Webp压缩（800x600以上尺寸）
视频截帧调用次数
获取视频元信息调用次数

- 服务监控总览

监控维度
可用性

监控指标参考

- 名字空间Scope：BCE_BOS
- 维度Dimension：BucketId;UserId

分类	指标英文名称（Metric name）	指标中文名称	单位	维度	备注
	BucketMAZSpaceUsedBytes	标准存储-多AZ占用空间大小	Bytes	BucketID	
	BucketMAZObjectCount	标准存储-多AZ文件数量	Count	BucketID	
	MAZReadOpCount	标准存储-多AZ读请求次数	Count	BucketID	
	MAZWriteOpCount	标准存储-多AZ写请求次数	Count	BucketID	
	MAZReadErrorOpCount	标准存储-多AZ读错误请求个数	Count	BucketID	
	MAZWriteErr	标准存储-多AZ写错误请	Count	BucketID	

	orOpCount	求个数			
	MAZSpaceUsedBytes	帐号下标准存储-多AZ总占用空间	Bytes	UserID	
	MAZObjectCount	帐号下标准存储-多AZ总文件数量	Count	UserID	
	BucketSpaceUsedBytes	标准存储占用空间大小	Bytes	BucketID	
	BucketObjectCount	标注存储文件数量	Count	BucketID	
	ReadOpCount	标准存储读请求次数	Count	BucketID	
	WriteOpCount	标准存储写请求次数	Count	BucketID	
	ReadErrorOpCount	标准存储读错误请求次数	Count	BucketID	
	WriteErrorOpCount	标准存储写错误请求次数	Count	BucketID	
	BucketMAZStandardIASpaceUsedBytes	低频存储-多AZ占用空间大小	Bytes	BucketID	
	BucketMAZStandardIAObjectCount	低频存储-多AZ文件数量	Count	BucketID	
	MAZStandardIAReadOpCount	低频存储-多AZ读请求次数	Count	BucketID	
	MAZStandardIAWriteOpCount	低频存储-多AZ写请求次数	Count	BucketID	
	MAZStandardIAReadErrorOpCount	低频存储-多AZ读错误请求个数	Count	BucketID	
	MAZStandardIAWriteErrorOpCount	低频存储-多AZ写错误请求个数	Count	BucketID	
	MAZStandardIARetrievalBytes	低频存储-多AZ数据取回大小	Bytes	BucketID	
	MAZStandardIASpaceUsedBytes	帐号下低频存储-多AZ总占用空间	Bytes	UserID	
基础数据	MAZStandardIAObjectCount	帐号下低频存储-多AZ总文件数量	Count	UserID	
	BucketStandardIAStorageBytes	低频存储占用空间大小	Bytes	BucketID	
	BucketStandardIAObjectCount	低频存储文件数量	Count	BucketID	
	StandardIAReadOpCount	低频存储读请求次数	Count	BucketID	
	StandardIAWriteOpCount	低频存储写请求次数	Count	BucketID	
	StandardIAReadErrorOpCount	低频存储读错误请求次数	Count	BucketID	
	StandardIAWriteErrorOpCount	低频存储写错误请求次数	Count	BucketID	
	StandardIARetrievalBytes	低频存储数据取回大小	Bytes	BucketID	
	BucketColdSpaceUsedBytes	冷存储占用空间大小	Bytes	BucketID	
	BucketColdObjectCount	冷存储文件数量	Count	BucketID	
	ColdReadOp	冷存储读请	Count	BucketID	

	Count	求次数			
	ColdWriteOpCount	冷存储写请求次数	Count	BucketID	
	ColdReadErrorOpCount	冷存储读错误请求次数	Count	BucketID	
	BucketArchiveSpaceUsedBytes	归档存储占用空间大小	Bytes	BucketID	
	BucketArchiveObjectCount	归档存储文件数量	Count	BucketID	
	ArchiveReadOpCount	归档存储读请求次数	Count	BucketID	
	ArchiveWriteOpCount	归档存储写请求次数	Count	BucketID	
	ArchiveReadErrorOpCount	归档存储读错误请求次数	Count	BucketID	
	ArchiveWriteErrorOpCount	归档存储写错误请求次数	Count	BucketID	
	ArchiveRetrievalBytes	归档存储取回字节数	Bytes	BucketID	
	ArchiveMinTimedStorage	归档最小存储	Bytes*Second	BucketID	
	StandardTransforToStandardIA	标准转低频存储容量	Byte	BucketID	
	StandardIATransforToCold	低频转冷存储容量	Byte	BucketID	
	ColdTransforToStandardIA	冷存储上浮低频存储容量	Byte	BucketID	
	StandardIATransforToStandard	低频上浮标准存储容量	Byte	BucketID	
流量数据	ReadBytes	外网读取量	Bytes	BucketID	
	WriteBytes	外网写入量	Bytes	BucketID	
	InternetWriteBandwidth	每秒外网写数据总量	bps	BucketID	
	InternetReadBandwidth	每秒外网读数据总量	bps	BucketID	
	IntranetReadBandwidth	每秒内网读数据量	bps	BucketID	
	IntranetWriteBandwidth	每秒内网写数据量	bps	BucketID	
	CDN1ReadBytes	CDN回源流量	Bytes	BucketID	
	XRegionCopyBytes	跨区域复制流出流量	Bytes	BucketID	
	XRegionInCopyBytes	跨区域复制流入流量	Bytes	BucketID	
	ERegionCopyBytes	同区域复制流出流量	Bytes	BucketID	
	ERegionInCopyBytes	同区域复制流入流量	Bytes	BucketID	
	TotalCopyBytes	复制总流出流量	Bytes	BucketID	
	TotalInCopyBytes	复制总流入流量	Bytes	BucketID	
	DeleteRequestsPerSecond	每秒删除请求总次数	rps	BucketID	
	GetRequestsPerSecond	每秒读请求总次数	rps	BucketID	
	PutRequestsPerSecond	每秒写请求总次数	rps	BucketID	
	TotalSuccessOpCount	返回状态码为2XX的请求数	Count	BucketID	
	TotalRedirec	返回状态码			

	tOpCount	为3XX的请求数	Count	BucketID	
	TotalClientErrorOpCount	返回状态码为4XX的请求数	Count	BucketID	
	TotalServerErrorOpCount	返回状态码为5XX的请求数	Count	BucketID	
	TotalRequestTimeoutCount	返回状态码为408的请求数	Count	BucketID	
	ClientDisconnectCount	返回状态码为499的请求数	Count	BucketID	
	ListObjectCount	List Object请求总次数	Count	BucketID	
	GetObjectCount	Get Object请求总次数	Count	BucketID	
	PutObjectCount	Put Object请求总次数	Count	BucketID	
	DeleteObjectCount	DeleteObject请求总次数	Count	BucketID	
	HeadObjectCount	Head Object请求总次数	Count	BucketID	
请求数据	ClientDisconnectErrorRate	客户端断链请求占比	%	BucketID	
	ClientTimeoutErrorRate	客户端超时请求占比	%	BucketID	
	AlertFlowControlCount	触发限流警告总次数	Count	BucketID	
	SoftFlowControlCount	触发软限流总次数	Count	BucketID	
	HardFlowControlCount	触发硬限流总次数	Count	BucketID	
	TurnAroundTime	BOS后端平均处理时间	毫秒	BucketID	
	PutAvgTurnAroundTime	BOS后端平均写处理时间	毫秒	BucketID	
	GetAvgTurnAroundTime	BOS后端平均读处理时间	毫秒	BucketID	
	TurnAroundTimeWarnCount	后端处理时间过长(1s)的请求总数	count	BucketID	
	GetTurnAroundTimeWarnCount	后端处理时间过长(1s)的读请求数	count	BucketID	
	PutTurnAroundTimeWarnCount	后端处理时间过长(1s)的写请求数	count	BucketID	
	TurnAroundTimeWarn5sCount	后端处理时间过长(5s)的请求总数	count	BucketID	
	GetTurnAroundTimeWarn5sCount	后端处理时间过长(5s)的读请求数	count	BucketID	
	PutTurnAroundTimeWarn5sCount	后端处理时间过长(5s的写请求数	count	BucketID	
	ImageBasics1Bytes	图像基础处理用量	Bytes	BucketID	
	ImageBasics1Count	基础图像处理调用次数	Count	BucketID	
	ImageHeic1Count	Heic压缩（800x600以下尺寸）	Count	BucketID	
	ImageHeic2Count	Heic压缩（800x600以上尺寸）	Count	BucketID	
数据处理	ImageWebp1Count	Webp压缩（800x600	Count	BucketID	

	1Count	以下尺寸）			
	ImageWebp2Count	Webp压缩（800x600以上尺寸）	Count	BucketID	
	VideoSnapshotCount	视频截帧调用次数	Count	BucketID	
	VideoInfoCount	获取视频元信息调用次数	Count	BucketID	
服务监控	Availability	可用性	%	BucketID	

监控数据保留策略

监控历史数据将会被保留40天，过期自动清除。

OpenAPI访问监控数据

Java代码示例：

```
 public static void main( String[] args )
   {
       searchBCM();
   }

public static void searchBCM(){
       // params definition
       String endpoint = "http://bcm.bj.baidubce.com";
       String userId = "xxx"; // 百度智能云账号id
       String ak = "xxx";
       String sk = "xxx";
       String scope = "BCE_BOS";
       String metricName = "BucketSpaceUsedBytes"; // 监控字段可以参考：https://ku.baidu-int.com/knowledge/HFVrC7hq1Q/pKzJfZczuc/iEP_IS7I1z/Q0JyPCxC-jqT6i#anchor-fa5ac7f0-2a14-11ee-9203-7fb0cea3ad07
       String dimensions = "BucketId:hj-test";  // 维度信息可以参考bcm文档：https://cloud.baidu.com/doc/BCM/s/ujwvylt2m
       Statistics[] statistics = new Statistics[]{Statistics.average,Statistics.maximum};
       long now = System.currentTimeMillis();
       String startTime = DateUtils.formatAlternateIso8601Date(new Date(now - 60 * 60 * 1000));
       String endTime = DateUtils.formatAlternateIso8601Date(new Date(now));
       int periodInSecond = 60;

       // create a bcm client
       BcmClientConfiguration config = new BcmClientConfiguration();
       config.setCredentials(new DefaultBceCredentials(ak, sk));
       config.setEndpoint(endpoint);
       BcmClient bcmClient = new BcmClient(config);


       // query metric data from bcm interface
       MetricDataRequest request = new MetricDataRequest();
       request.withUserId(userId)
               .withScope(scope)
               .withDimensions(dimensions)
               .withMetricName(metricName)
               .withStatistics(statistics)
               .withStartTime(startTime)
               .withEndTime(endTime)
               .withPeriodInSecond(periodInSecond);
       MetricDataResponse response = bcmClient.getMetricData(request);
       System.out.println(JsonUtils.toJsonString(response));
   }
```

返回结果示例（返回最近一个小时的bucket桶的标准存储占用空间数据变换点）

{"metadata":{"bceRequestId":"904c74a7-8ad2-4317-bb10-fc4c30cadd16","transferEncoding":"chunked","contentLength":-1,"contentType":"application/json;charset=UTF-8","date":1691137068000},"requestId":"904c74a7-8ad2-4317-bb10-fc4c30cadd16","code":"OK","message":"","dataPoints":[{"average":6.95464088E8,"maximum":6.95464088E8,"timestamp":"2023-08-04T07:17:47Z"},
{"average":6.95464088E8,"maximum":6.95464088E8,"timestamp":"2023-08-04T07:18:47Z"},
{"average":6.95464088E8,"maximum":6.95464088E8,"timestamp":"2023-08-04T07:19:47Z"},
{"average":6.95464088E8,"maximum":6.95464088E8,"timestamp":"2023-08-04T07:20:47Z"},
{"average":6.95464088E8,"maximum":6.95464088E8,"timestamp":"2023-08-04T07:21:47Z"},
{"average":6.95464088E8,"maximum":6.95464088E8,"timestamp":"2023-08-04T07:22:47Z"},
{"average":6.95464088E8,"maximum":6.95464088E8,"timestamp":"2023-08-04T07:23:47Z"},
{"average":6.95464088E8,"maximum":6.95464088E8,"timestamp":"2023-08-04T07:24:47Z"},
{"average":6.95464088E8,"maximum":6.95464088E8,"timestamp":"2023-08-04T07:25:47Z"},
{"average":6.95464088E8,"maximum":6.95464088E8,"timestamp":"2023-08-04T07:26:47Z"},
{"average":6.95464088E8,"maximum":6.95464088E8,"timestamp":"2023-08-04T07:27:47Z"},
{"average":6.95464088E8,"maximum":6.95464088E8,"timestamp":"2023-08-04T07:28:47Z"},
{"average":6.95464088E8,"maximum":6.95464088E8,"timestamp":"2023-08-04T07:29:47Z"},
{"average":6.95464088E8,"maximum":6.95464088E8,"timestamp":"2023-08-04T07:30:47Z"},
{"average":6.95464088E8,"maximum":6.95464088E8,"timestamp":"2023-08-04T07:31:47Z"},
{"average":6.95464088E8,"maximum":6.95464088E8,"timestamp":"2023-08-04T07:32:47Z"},
{"average":6.95464088E8,"maximum":6.95464088E8,"timestamp":"2023-08-04T07:33:47Z"},
{"average":6.95464088E8,"maximum":6.95464088E8,"timestamp":"2023-08-04T07:34:47Z"},
{"average":6.95464088E8,"maximum":6.95464088E8,"timestamp":"2023-08-04T07:35:47Z"},
{"average":6.95464088E8,"maximum":6.95464088E8,"timestamp":"2023-08-04T07:36:47Z"},
{"average":6.95464088E8,"maximum":6.95464088E8,"timestamp":"2023-08-04T07:37:47Z"},
{"average":6.95464088E8,"maximum":6.95464088E8,"timestamp":"2023-08-04T07:38:47Z"},
{"average":6.95464088E8,"maximum":6.95464088E8,"timestamp":"2023-08-04T07:39:47Z"},
{"average":6.95464088E8,"maximum":6.95464088E8,"timestamp":"2023-08-04T07:40:47Z"},
{"average":6.95464088E8,"maximum":6.95464088E8,"timestamp":"2023-08-04T07:41:47Z"},
{"average":6.95464088E8,"maximum":6.95464088E8,"timestamp":"2023-08-04T07:42:47Z"},
{"average":6.95464088E8,"maximum":6.95464088E8,"timestamp":"2023-08-04T07:43:47Z"},
{"average":6.95464088E8,"maximum":6.95464088E8,"timestamp":"2023-08-04T07:44:47Z"},
{"average":6.95464088E8,"maximum":6.95464088E8,"timestamp":"2023-08-04T07:45:47Z"},
{"average":6.95464088E8,"maximum":6.95464088E8,"timestamp":"2023-08-04T07:46:47Z"},
{"average":6.95464088E8,"maximum":6.95464088E8,"timestamp":"2023-08-04T07:47:47Z"},
{"average":6.95464088E8,"maximum":6.95464088E8,"timestamp":"2023-08-04T07:48:47Z"},
{"average":6.95464088E8,"maximum":6.95464088E8,"timestamp":"2023-08-04T07:49:47Z"},
{"average":6.95464088E8,"maximum":6.95464088E8,"timestamp":"2023-08-04T07:50:47Z"},
{"average":6.95464088E8,"maximum":6.95464088E8,"timestamp":"2023-08-04T07:51:47Z"},
{"average":6.95464088E8,"maximum":6.95464088E8,"timestamp":"2023-08-04T07:52:47Z"},
{"average":6.95464088E8,"maximum":6.95464088E8,"timestamp":"2023-08-04T07:53:47Z"},
{"average":6.95464088E8,"maximum":6.95464088E8,"timestamp":"2023-08-04T07:54:47Z"},
{"average":6.95464088E8,"maximum":6.95464088E8,"timestamp":"2023-08-04T07:55:47Z"},
{"average":6.95464088E8,"maximum":6.95464088E8,"timestamp":"2023-08-04T07:56:47Z"},
{"average":6.95464088E8,"maximum":6.95464088E8,"timestamp":"2023-08-04T07:57:47Z"},
{"average":6.95464088E8,"maximum":6.95464088E8,"timestamp":"2023-08-04T07:58:47Z"},
{"average":6.95464088E8,"maximum":6.95464088E8,"timestamp":"2023-08-04T07:59:47Z"},
{"average":6.95464088E8,"maximum":6.95464088E8,"timestamp":"2023-08-04T08:00:47Z"},
{"average":6.95464088E8,"maximum":6.95464088E8,"timestamp":"2023-08-04T08:01:47Z"},
{"average":6.95464088E8,"maximum":6.95464088E8,"timestamp":"2023-08-04T08:02:47Z"},
{"average":6.95464088E8,"maximum":6.95464088E8,"timestamp":"2023-08-04T08:03:47Z"},
{"average":6.95464088E8,"maximum":6.95464088E8,"timestamp":"2023-08-04T08:04:47Z"},
{"average":6.95464088E8,"maximum":6.95464088E8,"timestamp":"2023-08-04T08:05:47Z"},
{"average":6.95464088E8,"maximum":6.95464088E8,"timestamp":"2023-08-04T08:06:47Z"},
{"average":6.95464088E8,"maximum":6.95464088E8,"timestamp":"2023-08-04T08:07:47Z"},
{"average":6.95464088E8,"maximum":6.95464088E8,"timestamp":"2023-08-04T08:08:47Z"},
{"average":6.95464088E8,"maximum":6.95464088E8,"timestamp":"2023-08-04T08:09:47Z"},
{"average":6.95464088E8,"maximum":6.95464088E8,"timestamp":"2023-08-04T08:10:47Z"},
{"average":6.95464088E8,"maximum":6.95464088E8,"timestamp":"2023-08-04T08:11:47Z"},
{"average":6.95464088E8,"maximum":6.95464088E8,"timestamp":"2023-08-04T08:12:47Z"},
{"average":6.95464088E8,"maximum":6.95464088E8,"timestamp":"2023-08-04T08:13:47Z"},
{"average":6.95464088E8,"maximum":6.95464088E8,"timestamp":"2023-08-04T08:14:47Z"},
{"average":6.95464088E8,"maximum":6.95464088E8,"timestamp":"2023-08-04T08:15:47Z"},
{"average":6.95464088E8,"maximum":6.95464088E8,"timestamp":"2023-08-04T08:16:47Z"},
{"timestamp":"2023-08-04T08:17:47Z"}]}

使用报警服务

在查看对象存储BOS的使用情况时，可以创建报警规则。如果监控指标达到报警条件，云监控BCM可通过邮件、短信、电话等方式通知用户异常监控数据，并及时处理。

创建报警规则

1. 登录*云监控BCM控制台*。
2. 在左侧导航栏，选择**报警策略**。
3. 在**报警策略**页面，单击**添加策略**。
4. 在创建报警规则面板，设置报警规则相关参数

参数	说明
策略名称	设置报警策略名称。
策略规则	报警级别分为：通知、警告、重要、严重，用户可根据告警对业务的影响程度选择通知、警告、重要、严重类型，用户自定义报警级别后，该字段会在告警通知中展示。
	报警规则可添加且规则以及或规则，在报警规则部分可设置指标类型、阈值、单位、报警次数等。
	用户可配置报警发生后如未恢复正常，是否重复报警。
	用户可配置报警发生后如数据恢复正常，是否恢复报警。
	用户可配置监控数据不足时，是否出发报警动作。
报警通知	云监控BCM的报警通知会发送给用户或用户组内的用户，可以包含一个或多个用户。
	用户通知方式可以包括邮件、短信、电话。
	支持接口回调，支持填写公网可访问到的URL或如流、企业微信、钉钉、飞书、Knock群机器人webhook，详情参见BCM帮助文档。
	用户可设定定时关闭时间以及是否定时关闭。

管理报警规则

您可以对已创建的报警规则执行以下操作：

操作	说明
修改通知状态	可选择是否开启报警通知。
复制报警策略	单击复制，创建相同的策略规则和报警通知。
编辑报警策略	单击编辑，可调整已成功创建的报警策略。
删除报警策略	单击删除，可删除已不再需要的报警规则。

查看报警历史

1. 单击左侧**报警历史**，进入报警历史列表界面。
2. 单击**报警内容**，进入您要查看的报警事件的详情页面。
3. 在报警详情页面，可以查看到该报警事件的基本信息、数据详情、状态变更历史。

数据处理

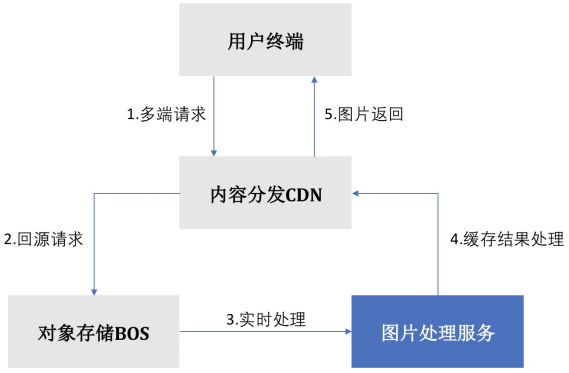
数据处理介绍

概述

BOS 图片处理是一项基于 BOS 核心存储业务的图片应用类服务，可提供灵活、多样、实时的图片处理服务。您只需要将原始图片上传到 BOS，通过在原始图片的请求中添加处理参数，即可实时地得到处理结果图。BOS 图片处理服务包括图片缩略、裁剪、格式转换、旋转、加文字/图片水印等，处理结果图会实时返回。

您可以通过 BOS 控制台和 API 方式对存储在 BOS 中的图片进行处理。您在 BOS 中创建的每个 Bucket 均默认具有图片处理的能力，无需额外开通操作。

图片处理服务的流程如下图所示：



- 用户首次发送请求处理流程：
  1. 用户端首次发送图片处理请求;
  2. 首次请求，CDN未命中，请求会被直接发送到BOS源站进行处理；
  3. 根据用户发送的处理参数对图片进行实时处理；
  4. 处理后的结果图将被缓存到CDN节点上；
  5. 同时，将结果图返回给用户端。
- 用户端非首次访问图片处理流程：
  1. 用户端发送图片处理请求;
  2. 若CDN命中，则直接跳到步骤5；若CDN未命中，则跳到步骤3；
  3. 请求会被回源到BOS，并根据用户发送的处理参数对图片进行实时处理；
  4. 处理后的结果图被缓存到CDN节点上；
  5. 将结果图返回给用户端。

在访问速度上，图片处理服务集成了百度智能云内容分发网络 CDN 功能，可利用遍布全球的高质量 CDN 节点，实现图片内容的加速分发，有效降低源站压力。

计费方式

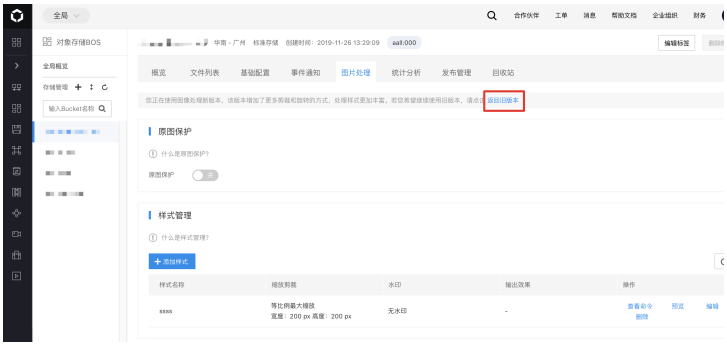
图像基础处理、图像高级处理中 HEIC 图像压缩及 WebP-M6 图像压缩服务将于 2020 年 6 月 1 日 0 点开始正式计费。当前，您可以登录控制台-财务查看用量数据。BOS 费用会按小时进行出账，具体价格请参考数据处理费用。

计费项	计费说明
基础图像处理费用	图像基础处理费用是指对存储在 BOS 中的图像进行缩放、裁剪、旋转、格式转换、渐进显示、质量变换、水印等基础图片处理操作时产生的费用。BOS 会根据处理前的图片大小进行计费。BOS 为每位用户每月免费赠送 10 TB 的图像基础处理额度。当您当月处理的图片量大于 10 TB 时，将按超出部分的使用量进行计费。
HEIC 图像压缩费用 (800x600 以下规格)	属于图像高级处理服务产生的费用。若您调用 BOS 的 HEIC 图像压缩服务，且处理的原图尺寸为 800x600 以下规格，会产生该项费用。该计费项按照调用次数进行计费。
HEIC 图像压缩费用 (800x600 以上规格)	属于图像高级处理服务产生的费用。若您调用 BOS 的 HEIC 图像压缩服务，且处理的原图尺寸为 800x600 以上规格，会产生该项费用。该计费项按照调用次数进行计费。
WebP-M6 图像压缩费用(800x600以下规格)	属于图像高级处理服务产生的费用。若您调用 BOS 的 WebP-M6 图像压缩服务，且处理的原图尺寸为 800x600 以下规格，会产生该项费用。该计费项按照调用次数进行计费。
WebP-M6 图像压缩费用(800x600以上规格)	属于图像高级处理服务产生的费用。若您调用 BOS 的 WebP-M6 图像压缩服务，且处理的原图尺寸为 800x600 以上规格，会产生该项费用。该计费项按照调用次数进行计费。
智能效果-智能去雾	属于图像智能效果服务产生的费用。若您调用智能去雾，可对浓雾天气下拍摄的细节无法辨认的图像进行去雾处理，还原更清晰真实的图像。该计费项按照调用次数进行计费。
智能场景-风格转换	属于图像智能场景服务产生的费用。若您调用风格转换，可将图像转化成卡通画、铅笔画、彩色铅笔画，或者哥特油画、彩色糖块油画、呐喊油画、神奈川冲浪里油画、奇异油画、薰衣草油画等共计 9 种风格，可用于开展趣味活动，或集成到美图应用中对图像进行风格转换。该计费项按照调用次数进行计费。
智能场景-人像动漫画	属于图像智能场景服务产生的费用。人像动漫画技术运用世界领先的对抗生成网络，结合人脸检测、头发分割、人像分割等技术，为用户量身定制千人千面的二次元动漫形象，并且可通过参数设置，生成戴口罩的二次元动漫人像。该计费项按照调用次数进行计费。
智能场景-天空分割	属于图像智能场景服务产生的费用。天空分割可以智能分割出天空边界位置，输出天空和其余背景的灰度图和二值图，可用于图像二次处理，进行天空替换、抠图等图片编辑场景。该计费项按照调用次数进行计费。
智能场景-物体擦除	属于图像智能场景服务产生的费用。物体擦除可以去除图片中不需要的遮挡物，并用背景内容填充，提高图像质量。该计费项按照调用次数进行计费。
智能场景-拉伸修复	属于图像智能场景服务产生的费用。拉伸修复可以自动识别过度拉伸的图像，将图像内容恢复成正常比例。该计费项按照调用次数进行计费。
智能场景-黑白上色	属于图像智能场景服务产生的费用。黑白上色可以智能识别黑白图像内容并填充色彩，使黑白图像变得鲜活。该计费项按照调用次数进行计费。
智能场景-无损放大	属于图像智能场景服务产生的费用。无损放大可对输入的图片在尽量保持图像质量的条件下，将图像在长宽方向各放大两倍。该计费项按照调用次数进行计费。

图像处理协议

BOS 图像处理目前支持 V1.0 和 V2.0 两个版本的图像处理协议。其中，V2.0 版本所支持的图像处理项目更加丰富，且在命令的调用上与 V1.0 有所不同，因此建议您使用 V2.0 版本进行处理。 V1.0 作为老版本，会在将来停止服务，但当前两个版本均可使用，系统将自动识别并兼容两种协议版本。

在您登录 BOS 控制台后，在 Bucket 的图像处理页签中可以对两个版本切换使用。

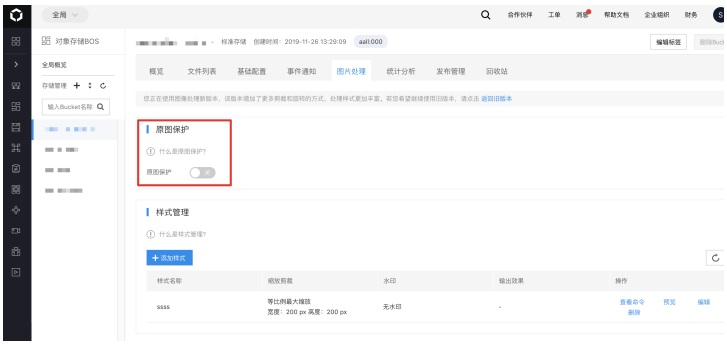


图像通用设置

原图保护

概述

若您希望开启原图保护，图片拥有者在"图片处理"控制台打开"原图保护"开关即可。



> **注意：** 未取回的归档类型 Object 不会触发图片处理，已取回的归档类型 Object 可正常处理。

图像处理指南V1.0

使用规则

概述

图像处理 V1.0 为 BOS 旧版图像处理协议，当前 BOS 新增的功能如自适应格式，图像高级压缩等仅支持通过 V2.0 协议使用。

规则和限制

- 支持的原图格式：jpg、png、bmp、webp、gif、tiff。
- 支持的目标图格式：jpg、png、bmp、webp。
- 支持图片大小为100MB，且图片长和宽不大于4096 px。
- 一个 Bucket 绑定的域名数（包括图片域名）不能超过 200 个，其中图片域名不超过1个。
- 对于同一个域名，只支持一种绑定方式，在普通域名绑定或图片域名绑定中选择一种，不支持两种方式同时绑定。
- 对于同一组处理命令的参数处理采用无序原则，如 @w_250,h_380,q_90 等同于 @h_380,w_250,q_90。
- 如果获取的Object名称本身带有@符号时，其后字符全部按照处理命令对待，用户需要提前替换Object 名称中的@符号。
- 最大支持4个管道。
- Authorization 的签名计算与普通Object处理方式类似，但必须将带参数的图片名称作为一个完整的 URL Path 用于签名。如带水印的图片签名计算的 URL 应该为

```
http://test.gz.bcebos.com/IMG_5974.JPG@!shuiyin1?authorization=...
```

图像访问规则

您可以通三种方式对图像进行访问。BOS 支持直接访问和显示原图，也可以通过命令访问图像，还可以通过创建样式来进行访问。

直接访问和显示原图

直接显示原图与普通的域名绑定访问方式一样，URL形式为：`http://${userdomain}/${objectkey}`

- `userdomain`为用户绑定的域名。
- `objectkey`为存储在BOS上的原图片名称。

样例：http://www.myhost.com/pic.jpg

通过命令访问

URL 形式为：`http://${userdomain}/${objectkey}@${commands}`

- `userdomain`为用户绑定的域名。
- `objectkey`为存储在BOS上的原图片名称。
- `commands`为处理命令。

样例：`http://bucket-A.bj.bcebos.com/sample.jpg@w_200,q_80`

分隔符

图像处理 V1.0 协议包括如下分隔符：

分隔符名称	分隔符	两侧顺序	说明
处理分隔符	@	固定顺序	Object与处理命令之间的分隔符
参数分隔符	_	固定顺序	参数缩略命令与参数值之间的分隔符
命令分隔符	,	无关	多项处理参数项之间的分隔符
管道分隔符	|	有关	两组处理命令之间分隔符，管道的参见下述“管道”

管道

如果一个图片，需要多重处理，如先裁剪、后缩略等，则多组操作命令之间，需要以管道“|”连接，执行顺序按管道指定顺序从左至右执行。

如下示例为一个管道，执行顺序为从左到右，左侧命令处理结果将会作为右侧命令的输入。

```
http://${userdomain}/${objectkey}@${cropcommands}|${scalecommands}
```

命令示例：

```
（1）如对图片先裁剪，再缩放。先以左上角横坐标为10，纵坐标为20进行裁剪，裁剪宽度指定为600，高度为300，再按宽200，高200进行拉伸缩放，并指定目标图片的绝对质量为60，目标图片格式为png。
https://doc.bce.baidu.com/bce-documentation/BOS/image.jpg@c_1,x_10,y_20,w_600,h_300|s_1,w_200,h_200,q_60,f_png
（2）如对图片先缩放、旋转，再裁剪。先按宽200的比例等比缩放并顺时针旋转90度，再以左上角横坐标为50，纵坐标为100为起点进行裁剪，裁剪宽度和高度默认为图片宽度和高度。
https://doc.bce.baidu.com/bce-documentation/BOS/image.jpg@w_200,a_30|c_1,x_50,y_100>
（3）如进行图文混合水印：水印图片为bce-documentation/BOS/bce.jpg，位于原图左下角，距离锚点水平距离为,15，垂直距离为20，透明度为50；文字内容为“hello world”，位于正上方，垂直距离为100，字体颜色为淡紫色。
https://doc.bce.baidu.com/bce-documentation/BOS/image.jpg@wm_1,k_YmNlLWRvY3VtZW50YXRpb24vQk9TL2JjZS5qcGc=,g_7,x_15,y_20,o_50|wm_2,t_aGVsbG8gd29ybGQ=,g_2,y_100,fc_aa88aa
```

- 图文混合水印：水印图片为bce-documentation/BOS/bce.jpg，位于原图左下角，距离锚点水平距离为,15，垂直距离为20，透明度为50；文字内容为“hello world”，位于正上方，垂直距离为100，字体颜色为淡紫色。

  https://doc.bce.baidu.com/bce-documentation/BOS/image.jpg@wm_1,k_YmNlLWRvY3VtZW50YXRpb24vQk9TL2JjZS5qcGc=,g_7,x_15,y_20,o_50|wm_2,t_aGVsbG8gd29ybGQ=,g_2,y_100,fc_aa88aa

通过样式访问

在对图片进行多重处理时，图片访问的 URL 会变得复杂冗长，非常不利于管理和阅读。为了让图片 URL 变得更简洁优雅，或是您想达到隐藏处理参数、保护原图的目的时，您可以使用图片样式功能。

图片样式是指您可以将一些常用的图片处理操作保存为一个别名，即一个样式。这样，一组复杂的操作，利用样式，可以通过一个简洁的URL访问即可达到相同效果。

图像处理 V1.0 的样式形式为：

```
http://${userdomain}/${objectkey}@!${stylename}
```

其中：

- **userdomain**：为用户绑定的域名。
- **objectkey**：为存储在BOS上的原图片名称。
- **stylename**：为样式名称

如您创建了一个名称为 mystylename 的样式，您可以以如下方式进行访问：

```
http://bucket-A.bj.bcebos.com/sample.jpg@!mystylename
```

添加和删除样式

添加样式

您可以通过控制台进行图片样式的定义，同时还可以在定义的同时实时进行效果预览。

1. 登录BOS控制台，进入您已开通图片服务的Bucket。点击左侧导航“图片服务”后，点击右侧页面中的“样式管理”。
2. 点击“添加样式”，进入添加样式表单页面。您可根据需要，设定该样式的图片处理参数，同时您可在右侧的预览区域实时进行效果预览。

   图片样式的设置方式支持“基础设置”和“高级设置”两种。

   - 基础设置，您需要在页面上对下方的每一项处理参数进行手动设定。
   - 高级设置，您可直接在框中直接输入图片处理命令参数。

   创建好样式后，点击“保存”。

   缩放说明

   - 限定宽度，高度等比例。

     您需要输入宽度，缩略时以宽度为基准，将高度等比例缩小。若输入宽度大于原图宽度，则返回原图。

     例如：原图宽高尺寸为400 × 320，限定宽度200，高度等比例，生成缩略图尺寸为200×160；限定宽度500，高度等比例，生成缩略图尺寸为400×320。

   - 限定高度，宽度等比例。

     您需要输入高度，缩略时以高度为基准，将宽度等比例缩小。若输入高度大于原图高度，则返回原图。

     例如：原图宽高尺寸为400 × 320，限定高度160，宽度等比例，生成缩略图尺寸为200×160；限定高度500，宽度等比例，生成缩略图尺寸为400×320。

   - 限定宽度和高度。

     您需要输入宽度和高度，图片会保持原图比例进行缩略，直至宽、高都满足设定值，不裁剪。

     例如：原图宽高尺寸为400 × 320，限定宽度为100，高度为100，生成缩略图尺寸为100 × 80；限定宽度为500，高度为500，生成缩略图尺寸为400×320。

   - 固定宽度和高度。

     您需要输入宽度和高度，图片会保持原图比例先进行缩略，超出设定宽、高部分进行居中裁剪。

     例如：原图宽高尺寸为400 × 320 ，固定宽度100和高度100（缩小较大图片，并裁剪图片），生成缩略图尺寸为100×100；固定宽度500和高度500（放大较小图片，并裁剪图片），生成缩略图尺寸为500×500。

3. 保存样式后，在列表中可以看到您已经添加的样式信息。您可以点击“查看命令”，查看样式对应的参数命令。



删除样式

在样式管理列表中，点击“操作”列的“删除”，可删除样式。



图像缩略

概述

本文介绍如何对图像进行缩略处理。

图像缩略参数

命令参数

参数名称	缩写命令	类型	取值范围	命令描述	缺省值	是否必选
scale	s	unsigned int	0,1,2	指定缩放方式。s_0表示无裁剪，等比例缩放；s_1表示拉伸缩放；s_2表示等比例居中裁剪缩放。	0	否
width	w	unsigned int	1～4096	指定目标缩略图的最大宽度单位为px。	-	s_1, s_2时，必选
height	h	unsigned int	1～4096	指定目标缩略图的最大高度单位为px。	-	s_1, s_2时，必选
limit	l	unsigned int	0,1	指定是否限制图片缩放的尺寸大小：l_0表示不限制图片缩放的尺寸；l_1表示限制图片缩放尺寸（长和宽）不可超过原图大小。	0	否

注意事项：

- 当 s_0 时，即无裁剪等比缩放时：
- w 和 h 可以只指定其中一个，图片按指定的参数比例进行缩放。例如，原图尺寸为 100 × 200，指定参数为 w_50 后，图片处理的尺寸为 50 × 100；若指定 h_50 后，图片处理的尺寸为 25 × 50;
- w 和 h 同时指定时，系统会取“目标宽度/原图宽度”与“目标高度/原图高度”中值小的参数为标准进行等比例缩放。例如：原图尺寸为 400 × 500 ，指定参数为 w_100, h_200 时，由于“100/400”< “200/500”，则系统按照w_100的比例进行缩放，目标尺寸为100 × 125。；
- 当s_1时，必须同时指定w和h的值;
- 当s_2时，必须同时指定w和h的值。指定w和h后，图片会按照“目标宽度/原图宽度”与“目标高度/原图高度”中值大的参数为标准进行等比例缩放，并对比值较大的另一边进行居中裁剪。例如：原图尺寸为500 × 1000 ，指定参数为 s_2，w_250，h_600 时，由于“250/500” < “600/1000”，会以目标高度 600 为基准进行等比缩放，同时将宽度按照居中裁剪为250。

示例

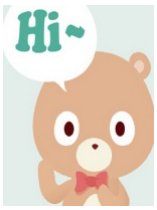
- 将图片缩略成高度为 200，宽度按比例处理。

https://doc.bce.baidu.com/bce-documentation/BOS/image.jpg@h_200



- 将图片缩略成宽度为150，高度为200，目标图片绝对质量为60，目标图片格式为png。

https://doc.bce.baidu.com/bce-documentation/BOS/image.jpg@s_1,w_150,h_200,q_60,f_png



- 将图片进行等比例居中裁剪缩放，缩放宽度为80，高度为100。

https://doc.bce.baidu.com/bce-documentation/BOS/image.jpg@s_2,w_80,h_100



- 将图片缩略成宽度为150，高度为100，逆时针旋转90度，并渐进显示。

https://doc.bce.baidu.com/bce-documentation/BOS/image.jpg@s_1,w_150,h_100,a_-90,d_progressive



图像裁剪

概述

本文介绍图片裁剪操作。

图片裁剪参数

参数名称	缩写命令	类型	取值范围	命令描述	缺省值	是否必选
crop	c	unsigned int	0,1	指定图片是否执行裁剪功能。c_1表示对图片进行裁剪；c_0表示不对图片进行裁剪，其他裁剪参数无效。	-	否
offsetX	x	unsigned int	0～4096	指定图片裁剪左上角起点x坐标。	0	否
offsetY	y	unsigned int	0～4096	指定图片裁剪左上角起点y坐标。	0	否
width	w	unsigned int	1～4096	指定图片裁剪的宽度。如果指定的宽度超过了图片的宽度，则以图片宽度为准裁剪。单位为px。	原始图片宽度	否
height	h	unsigned int	1～4096	指定图片裁剪的高度。如果指定的高度超过了图片的高度，则以图片高度为准裁剪。单位为px。	原始图片高度	否

示例

- 以图片左上角横坐标为100，纵坐标为150为起点进行图片裁剪，裁剪宽度和高度默认为原始图片宽度和高度。

https://doc.bce.baidu.com/bce-documentation/BOS/image.jpg@c_1,x_100,y_150



- 以图片左上角横、纵坐标均为0为起点进行图片裁剪，裁剪宽度指定为600，裁剪高度指定为600。

https://doc.bce.baidu.com/bce-documentation/BOS/image.jpg@c_1,w_600,h_600



- 以图片左上角横坐标为100，纵坐标为150为起点进行图片裁剪，裁剪宽度指定为400，裁剪高度指定为200。

https://doc.bce.baidu.com/bce-documentation/BOS/image.jpg@c_1,x_100,y_150,w_400,h_200



图像旋转

概述

本文介绍图像旋转操作。

图像旋转参数

参数名称	缩写命令	类型	取值范围	命令描述	缺省值	是否必选
angle	a	int	-360～360	指定图片旋转方式。正数表示顺时针旋转，如a_60；负数表示逆时针旋转，如a_-30。	0	否
orientation	o	int	0,1,2	根据exif信息进行自动旋转。o_0：按原图默认方向，不自动旋转；o_1：先旋转，然后再缩略；o_2：先缩略，然后再旋转。		

限制：图片的宽或者高不能超过4096 px

示例

- 将图片顺时针旋转30度。

https://doc.bce.baidu.com/bce-documentation/BOS/image.jpg@a_30



渐近显示

概述

本文介绍渐近显示操作。

渐近显示参数

参数名称	缩写命令	类型	取值范围	命令描述	缺省值	是否必选
display	d	string	baseline progressive	指定图片显示方式。 baseline：标准式，指图片按照从上到下的扫描方式显示； progressive：渐进式，指图片由模糊到清晰渐进显示。 progressive只对jpg、png格式图片有效，若指定其他格式，则默认显示方式仍为baseline。	baseline	否

示例

- 将图片缩略成宽度为150，高度为100，逆时针旋转90度，并渐进显示。

https://doc.bce.baidu.com/bce-documentation/BOS/image.jpg@s_1,w_150,h_100,a_-90,d_progressive



水印

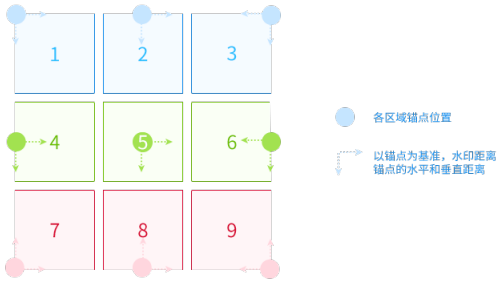
概述

本文介绍如何设置水印。图片水印是指在图片上方插入另一张图片或者文字作为水印，是一种数字信息的保护手段。目前 BOS 图像处理 V1.0 协议支持图片水印和文字水印两种水印方式。BOS 将您的原始图片分成如下图所示 9 个区域，并按照图示位置进行编号，每个区域均包含锚点。

确定水印在原图上的最终位置，您需要确定如下两个因素：

- 锚点位置；
- 基于锚点位置，指定在水平和垂直方向上的距离。

锚点位置如图所示：



图片水印参数

参数名称	缩写命令	类型	取值范围	命令描述	缺省值	是否必选
watermark	wm	unsigned	0，1，2	指定水印方式。wm_0表示关闭水印功能；wm_1表示图片水印；wm_2表示文字水印。	-	是
key	k	string	-	指定水印图片存储在BOS上的Object名称，即ObjectKey。如Object存储在文件夹下，则需要包含文件夹路径。该值要求为Base64编码后的值。	-	是
gravity	g	unsigned int	1～9	指定水印的锚点位置。图片锚点的含义请参考图片水印命令。	9	否
gravityX	x	int	-4096～4096	指定水印基于锚点的水平方向距离。	10	否
				指定水印基于		

gravityY	y	int	-4096～4096	锚点的垂直方向距离。	10	否
angle	a	int	-360～360	指定水印图片的旋转角度。正数表示顺时针旋转，如a_60；负数表示逆时针旋转，如a_-30。	0	否
opacity	o	unsigned int	1～100	指定水印图片的透明度。取值为100时，表示与原始图片透明度一致。	100	否

说明

- 水印图片要与原始图片存储在BOS的同一个Bucket下。

文字水印参数

参数名称	缩写命令	类型	取值范围	命令描述	缺省值	是否必选
watermark	wm	unsigned	0，1，2	指定水印方式。wm_0表示关闭水印功能；wm_1表示图片水印；wm_2表示文字水印。	-	是
text	t	string	-	水印文字内容，需要进行Base64编码，且编码后最多不能超过80个字符。	-	是
gravity	g	unsigned int	1～9	指定水印的锚点位置。图片锚点的含义请参考图片水印命令。	9	否
				指定水印基干		

gravityX	x	int	-4096～4096	锚点的水平方向距离。	10	否
gravityY	y	int	-4096～4096	指定水印基于锚点的垂直方向距离。	10	否
angle	a	int	-360～360	指定水印图片的旋转角度。正数表示顺时针旋转，如a_60；负数表示逆时针旋转，如a_-30。	0	否
fontSize	sz	unsigned int	1～1024	指定水印字体大小，单位是 px。	30	否
fontColor	fc	string	-	指定字体颜色，为一个8位的16进制字符串，其中每2位用来表示RGBA中的一项。前6位分别表示RGB色，如ffffff；最后2位表示Alpha通道，可省略，省略时表示不透明。例如：000000ff表示黑色不透明字体。	000000	否
fontFamily	ff	string	-	指定字体的字体类型，取值要求进行Base64编码。已支持的字体类型请参见下方列表。	宋体	否
			normal（常规）	指定字		

fontStyle	fs	string	italic（斜体） bold（粗体）	体样式。	normal	否

已支持的字体类型：

参数取值	字体类型
MicrosoftYaHei	微软雅黑
FangSong	仿宋
SimHei	黑体
KaiTi	楷体
SimSun	宋体
FZBeiWeiKaiShuJianTi	方正北魏楷书简体
FZBoYaSongJianTi	方正博雅宋简体
FZCuHeiSongJianTi	方正粗黑宋简体
FZLanTingZhongHeiJianTi	方正兰亭中黑简体
FZZhengHei	方正正黑
AvantGarde	AvantGarde
Bookman	Bookman
Courier	Courier
Helvetica	Helvetica
HelveticaNarrow	HelveticaNarrow
NewCenturySchlbk	NewCenturySchlbk
Palatino	Palatino
Times	Times
Symbol	Symbol

> **说明：**
>
> - 只有当fontFamily支持fontStyle效果时，才会渲染文字效果。
> - 若指定的字体样式不在当前已支持的fontFamily列表中，则显示效果为默认字体宋体。

图文混合水印

图文混合水印是指将图片水印和文字水印同时打在原始图片上。由于图片水印和文字水印为两组独立的命令，因此图文混合水印需要用管道实现。

示例

图片水印示例

原图为image.jpg，水印图片为bce-documentation/BOS/bce.jpg(Base64编码后为"YmNlLWRvY3VtZW50YXRpb24vQk9TL2JjZS5qcGc=")。

- 在右下角加上图片水印，指定垂直距离为10，透明度为60。

https://doc.bce.baidu.com/bce-documentation/BOS/image.jpg@wm_1,k_YmNlLWRvY3VtZW50YXRpb24vQk9TL2JjZS5qcGc=,y_10,o_60



- 在左下角加上水印，水平和垂直距离取缺省值，透明度为50，并顺时针旋转90度。

https://doc.bce.baidu.com/bce-documentation/BOS/image.jpg@wm_1,k_YmNlLWRvY3VtZW50YXRpb24vQk9TL2JjZS5qcGc=,g_7,o_50,a_90



文字水印示例

- 最简单的文字水印，文字内容为"hello world"。

https://doc.bce.baidu.com/bce-documentation/BOS/image.jpg@wm_2,t_aGVsbG8gd29ybGQ=



- 文字内容为“hello world”，字体类型为微软雅黑，字体颜色和字体大小取默认值，字体样式为粗体，水印位置在右中，距离锚点水平距离20，垂直距离20，并顺时针旋转30度。

https://doc.bce.baidu.com/bce-documentation/BOS/image.jpg@wm_2,t_aGVsbG8gd29ybGQ=,ff_TWljcm9zb2ZOWWFIZWk=,fs_bold,g_6,x_20,y_20,a_30



图文混合水印

- 最简单的图文混合水印：图片为bce-documentation/BOS/bce.jpg，文字内容为“hello world”，其余参数均取默认值。

https://doc.bce.baidu.com/bce-documentation/BOS/image.jpg@wm_1,k_YmNlLWRvY3VtZW50YXRpb24vQk9TL2JjZS5qcGc=|wm_2,t_aGVsbG8gd29ybGQ=



- 图文混合水印：水印图片为bce-documentation/BOS/bce.jpg，位于原图左下角，距离锚点水平距离为,15，垂直距离为20，透明度为50；文字内容为“hello world”，位于正上方，垂直距离为100，字体颜色为淡紫色。

https://doc.bce.baidu.com/bce-documentation/BOS/image.jpg@wm_1,k_YmNlLWRvY3VtZW50YXRpb24vQk9TL2JjZS5qcGc=,g_7,x_15,y_20,o_50|wm_2,t_aGVsbG8gd29ybGQ=,g_2,y_100,fc_aa88aa



质量变换

概述

本文介绍图像质量变换。

质量变换参数

参数名称	缩写命令	类型	取值范围	命令描述	缺省值	是否必选
quality	q	unsigned int	1～100	指定目标图片的绝对质量，只对jpg/jpeg、png、webp格式有效。当指定q命令的值大于原图quality的值时，则默认使用原图quality的值。	原图quality值。	否

格式转换

概述

本文介绍如何进行格式转换。

格式转换参数

参数名称	缩写命令	类型	取值范围	命令描述	缺省值	是否必选
format	f	string	jpg、 png、 webp、 bmp	指定目标图片格式。响应时对应的Content-Type分别为image/jpeg、image/png、image/webp、image/bmp。	jpg	否

获取图片信息

概述

BOS 支持通过命令用来获取图片信息，如果图片有exif信息则返回包含exif的完整信息，如果图片不包含exif信息，则返回图片的基本信息。

返回的exif信息包含但不限于：

- dateTime、dateTimeOriginal、dateTimeDigitized、format、gpsLatitude、gpsLatitudeRef、gpsLongitude、gpsLongitudeRef、imageHeight、imageWidth、imageSize、make、model、orientation、resolutionX、resolutionY、resolutionUnit

> **说明：**exif信息记录了数码照片的属性信息和拍摄数据，并非每一张图片都包含exif信息。

获取图片信息方式

直接使用 exif 命令即可获取图片信息，无需参数。

示例

- 包含exif信息的图片示例：

https://doc.bce.baidu.com/bce-documentation/BOS/orient.jpg?exif

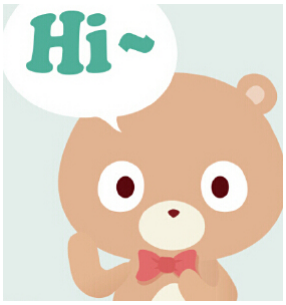


返回示例：


```
{
    "dateTime": {
        "value": "2016-08-31T14:57:07Z"
    },
    "dateTimeDigitized": {
        "value": "2016-08-31T14:57:07Z"
    },
    "dateTimeOriginal": {
        "value": "2016-08-31T14:57:07Z"
    },
    "format": {
        "value": "JPEG"
    },
    "gpsLatitude": {
        "value": "40/1,2/1,3675/100"
    },
    "gpsLatitudeRef": {
        "value": "N"
    },
    "gpsLongitude": {
        "value": "116/1,16/1,257/100"
    },
    "gpsLongitudeRef": {
        "value": "E"
    },
    "imageHeight": {
        "value": "2448"
    },
    "imageSize": {
        "value": "1225883"
    },
    "imageWidth": {
        "value": "3264"
    },
    "make": {
        "value": "Apple"
    },
    "model": {
        "value": "iPhone 6"
    },
    "resolutionUnit": {
        "value": "2"
    },
    "resolutionX": {
        "value": "72/1"
    },
    "resolutionY": {
        "value": "72/1"
    }
}
```


- 不包含exif信息的图片示例：

https://doc.bce.baidu.com/bce-documentation/BOS/image.jpg?exif



返回示例：

```
{
    "format": {
        "value": "JPEG"
    },
    "imageHeight": {
        "value": "296"
    },
    "imageSize": {
        "value": "63506"
    },
    "imageWidth": {
        "value": "279"
    }
}
```

图像处理指南V2.0

使用规则

概述

BOS 图片处理 V2.0 版本，是 BOS 全新的处理协议。相较 V1.0 版本，V2.0 版本支持更多图片处理功能，并采用全新的命令方式。您仍然可以通过 BOS 图片处理 V1.0 版本方式来访问 BOS 图片处理服务，但今后 V1.0 版本功能将不再更新。

在使用 BOS 图像处理功能前，您需要将原始图片上传到 BOS。BOS 的每个 Bucket 都默认开通图片处理的功能，您无需额外开通，也无需使用特定域名访问图片处理服务。当访问图片 URL 时，在 Querystring 中通过**x-bce-process**参数指定图片处理命令，即可触发 BOS 图片处理功能。

> **说明：**
>
> - 为了提升图片的安全性，BOS 也支持对指定 Bucket 内的图片开启原图保护。即开启原图保护后，如果图片获取者未验证被授权的签名，则只能通过样式方式访问处理后的图片，直接访问原图或通过处理参数访问的方式会被拒绝。目前支持在控制台对原图保护功能进行开启或关闭，也支持通过 API 进行相关操作，并可通过 resource 字段指定该功能生效范围。

规则与限制

- 支持的原图格式：jpg、png、bmp、webp、heic、gif；
- 支持的目标格式：jpg、png、bmp、webp、heic、gif；
- 处理结果输出图片宽高不超过4096px，包括管道和子命令间的中间输出图片；
- 最多可以同时指定16组操作，即16个action；
- 一个action下不可以重复指定同一个参数；
- 样式名称支持数字，字母和下划线组合，且首位不能为下划线，最长 64 个字符。

使用方式

BOS 支持通过命令和样式两种方式访问图片服务。

1. 命令方式

图片处理命令形式为：

```
x-bce-process=image/${action},${key}_${value},${key}_${value}/${action},${key}_${value}
```

其中：

- **image**：BOS支持多种数据处理形式，均使用x-bce-process参数触发，当使用图片处理时，需要指定此处名字为image。
- **action**：BOS支持多种图片处理命令，每种图片处理命令称作一种action，比如缩放resize，裁剪crop等。
- **key**：每种action支持多种具体的处理参数，比如缩放宽度w,高度h等。
- **value**：处理参数取值。
- **分隔符**：

分隔符名称	分隔符	两侧顺序	说明
处理分隔符	/	有关	多个action之间的分隔符，前后action顺序执行
参数分隔符	,	无关	多处理参数项之间的分隔符
值分隔符	_	固定顺序	参数名与参数值之间的分隔符

如对图片进行等比缩放图片，且缩放后的图片宽最大200，高最大100，亮度-5，格式转化为webp，则对应命令为：

```
http://bucket-A.bj.bcebos.com/sample.jpg?x-bce-process=image/resize,m_lfit,w_200,h_100/bright,b_-10/format,f_webp
```

2.样式方式

您可以通过控制台将某一项或多项图片处理命令保存为样式，并为其创建创建名称。之后，您可以通过样式达到图像处理的效果，简化原本一系列复杂命令操作和参数，只需通过很简短的 URL 即可实现相同的效果。

通过样式的访问语法：

```
http://${domain}/${objectkey}?x-bce-process=style/${stylename}
```

详细说明：

- **style**：当使用图片样式访问时，需要固定指定为style。

- **stylename**：访问的样式style的名称。

例如：您创建一个名称为 mystylename 的样式，可实现宽度为 200 px，高度为 300 px 的等比例最大缩放，并同步支持自适应旋转（命令为：pxresize,m_lfit,limit_0,w_200,h_300/auto-orient,o_1），您可以直接调用 mystylename 的名称来访问图像处理：

```
http://bucket-A.bj.bcebos.com/sample.jpg?x-bce-process=style/mystylename
```

控制台添加/删除样式

您可以通过 BOS 控制台添加或删除样式。

1. 登录百度智能云官网

登录百度智能云官网。

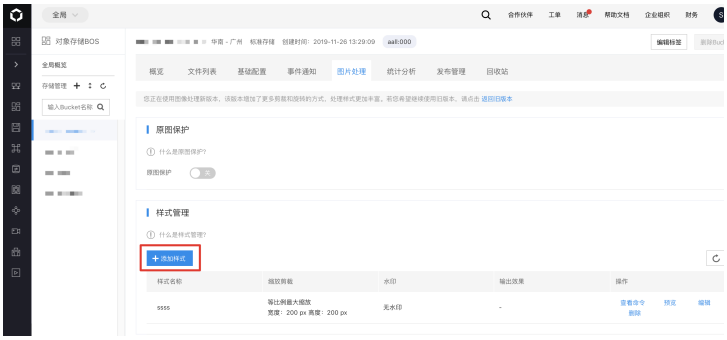
- 若您未注册账号，须先注册账户。
- 若您已注册，可直接登录。

2. 进入 BOS 控制台

在页面左侧导航栏中选择 "存储和 CDN > 对象存储 BOS" 进入 BOS 控制台。

3. 选择 Bucket 并进入图像处理页面

在左侧的 Bucket 列表中选择 Bucket 名称并进入图像处理页面。在该页面中，您可以创建新的图像处理样式，也可以在列表中查看已创建的图像处理样式。



4. 添加样式

点击添加样式按钮，进入添加样式页面。关于图像处理协议 V2.0 版本的样式设置方法，你可以查看图像处理协议 2.0 ；关于图像处理协议 V1.0 版本的样式设置方法，您可以查看图像处理协议 1.0。

在控制台中，您可以通过基础编辑方式和高级编辑方式对样式进行编辑。其中基础编辑方式是通过图形化界面对样式进行编辑，而高级编辑方式需要您输入相应的命令语句进行设置。

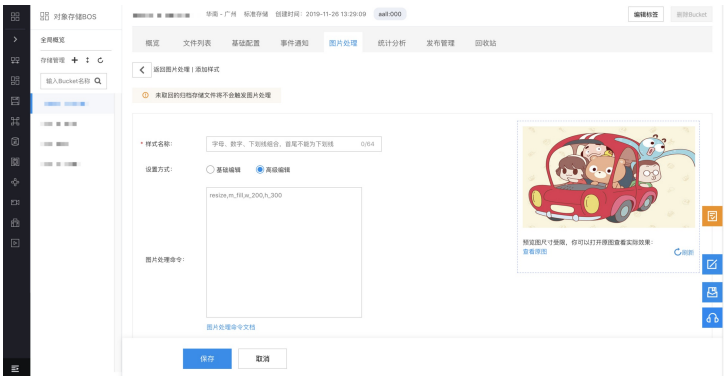
在基础编辑中，您可以进行样式名称设置和缩略设置。



也可以对该图像进行输出设置和水印设置。



在高级编辑中，您可以直接通过命令行设置。图片服务的命令行具体操作请参见图片处理服务 2.0和图像处理协议 1.0。



5. 添加列表

样式添加完成后会在图像处理的样式列表中显示，您可以通过操作列的"查看命令"查看该样式对应的命令，也可以预览样式，或编辑删除该样式。

> **注意：**
>
> - 图像处理协议 2.0 生成的样式，仅展示在新版协议的列表中；图像处理协议 1.0 生成的样式，仅展示在旧版协议的列表中，两个协议下的样式不会共同展示，请在页面上方切换版本进行查看。



> **说明：**
>
> - BOS 图片处理命令分新旧两种版本，因此一个图片处理样式所对应的实际图片处理命令可以是新版或旧版。
> - 本文 `x-bce-process=style/${stylename}`的访问方式只能访问新版命令所对应的样式，旧版本命令所对应的样式仍需通过旧版本@!方式访问。

图片处理结果返回

BOS 会将处理完的图片信息通过 HTTP Header 的方式返回。
除标准 HTTP Header 中已有的信息，其他信息用`x-bce-image-info`头返回，格式为`key1=value;key2=value2;...`

返回内容参考如下：

信息	响应头	值
format	Content-Type: image/jpeg	jpeg
size	Content-Length: 12608	12608
md5	Content-MD5: IsfLjdB8Uu+kwmuFKJ1VZw==	IsfLjdB8Uu+kwmuFKJ1VZw==
width	x-bce-image-info:width=200	200
height	x-bce-image-info:height=113	113

图像缩放

概述

本文介绍如何对图片大小进行缩放处理(`resize`)。

缩放参数

参数	取值	说明
m	lfit，mfit，fill，pad，fixed	• 设置缩略图的宽度，单位px。lfit类型必须设置w或h，fill fixed类型必须同时设置w和h • lfit：等比缩放图片，缩放到能放到以w、h为宽高的矩形内的最大缩放图片 • mfit：等比缩放图片，缩放到能包含住以w、h为宽高的矩形的最小缩放图片 • fill：mfit基础上，对w、h矩形内的图片部分进行裁剪 • pad：lfit基础上，对w、h矩形内的非图片空白部分进行指定颜色填充

		• fixed：强制缩放到指定w宽、h高
w	1-4096	• 设置缩略图的高度，单位px。lfit类型必须设置w或h，fill fixed类型必须同时设置w和h。与l/s/p/wp/hp互斥。
h	1-4096	• 设置缩略图的高度，单位px。lfit类型必须设置w或h，fill fixed类型必须同时设置w和h。与l/s/p/wp/hp互斥。
l	1-4096	• 设置缩略图的最长边。与w/h/p/wp/hp互斥。
s	1-4096	• 设置缩略图的最短边。与w/h/p/wp/hp互斥。
p	1-1000	• 缩放比例，与m/w/h/wp/hp/l/s互斥。
wp	1-1000	• 仅宽度进行缩放比例，与m/w/h/p/l/s互斥。
hp	1-1000	• 仅高度进行缩放比例，与m/w/h/p/l/s互斥。
align	0-100	• 当需要裁剪时，截取图片的偏移。默认50，居中裁剪。
limit	0,1	• 设置是否限制图片缩放的尺寸大小不可超过原图大小，0不限制，1限制。非必选，默认0不限制。
color	8位十六进制颜色码	• 指定pad类型下填充颜色，格式为RRGGBBAA。最后2位Alpha通道可省略，省略时默认为00表示不透明。非必选，默认黑色`000000`。
radius	1-50	• 指定pad类型下背景虚化高斯模糊半径。非必选，默认不模糊，填充颜色。与sigma共同设置时生效。

sigma	1-50	• 指定pad类型下背景虚化高斯模糊标准差。非必选，默认不模糊，填充颜色。与radius共同设置时生效。

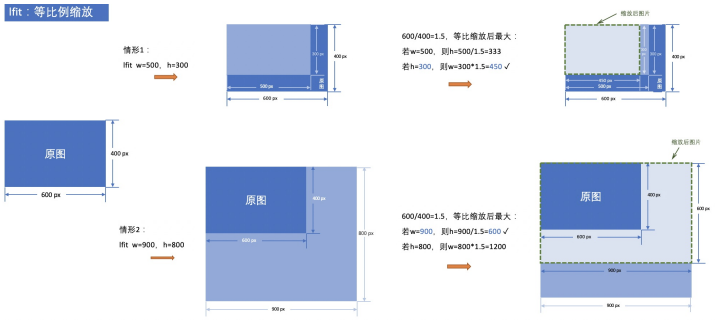
效果示例

- **lfit缩放**

若原图大小为 600 px 乘以 400 px。

情形 1：缩放参数为 w = 500 px，h = 300 px。按照 lift 等比缩放规则，缩放图的 w/h 等于原图的 w/h=1.5。所以，若：w = 500 px，则 h = 500/1.5 = 333 px；h = 300 px，则 w = 300x1.5 = 450 px。若限制在指定 w 与 h 的矩形内的最大图片，即缩放图的 w 乘 h 不能大于 500 px 乘 300 px。通过以上条件得出缩略图大小为 450 px 乘以 300 px。

情形 2：缩放参数为 w = 900 px，h = 800 px。按照 lift 等比缩放规则，缩放图的 w/h 等于原图的 w/h=1.5。所以，若：w = 900 px，则 h = 900/1.5 = 600 px；h = 800 px，则 w = 800x1.5 = 1200 px。若限制在指定 w 与 h 的矩形内的最大图片，即缩放图的 w 乘 h 不能大于 900 px 乘 800 px。通过以上条件得出缩略图大小为 900 px 乘以 600 px。

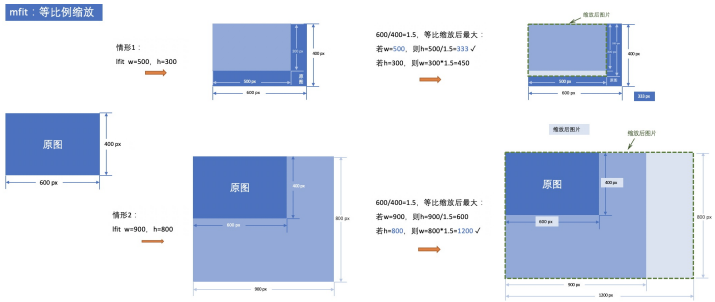


- **mfit缩放**

若原图大小为 600 px 乘以 400 px。

情形 1：缩放参数为 w = 500 px，h = 300 px。按照 mift 等比缩放规则，缩放图的 w/h 等于原图的 w/h=1.5。所以，若：w = 500 px，则 h = 500/1.5 = 333 px；h = 300 px，则 w = 300x1.5 = 450 px。若限制为指定 w 与 h 的矩形框外的最小图片，即缩放图的 w 乘 h 不能小于 500 px 乘 300 px。通过以上条件得出缩略图大小为 500 px 乘以 333 px。

情形 2：缩放参数为 w = 900 px，h = 800 px。按照 lift 等比缩放规则，缩放图的 w/h 等于原图的 w/h=1.5。所以，若：w = 900 px，则 h = 900/1.5 = 600 px；h = 800 px，则 w = 800x1.5 = 1200 px。若限制为指定 w 与 h 的矩形框外的最小图片，即缩放图的 w 乘 h 不能小于 1200 px 乘 800 px。通过以上条件得出缩略图大小为 1200 px 乘以 800 px。

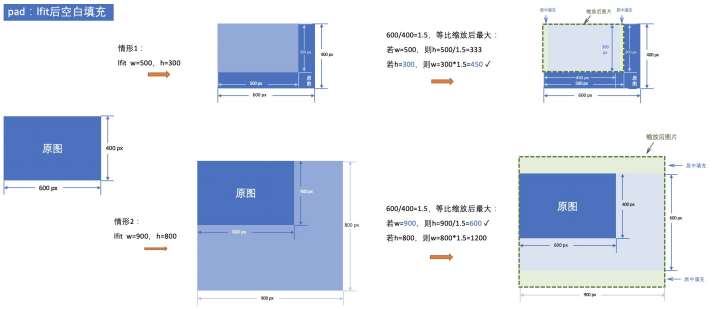


- **pad缩放**

若原图大小为 600 px 乘以 400 px。

情形 1：缩放参数为 w = 500 px，h = 300 px。按照 lift 等比缩放规则，缩放图的 w/h 等于原图的 w/h=1.5。所以，若：w = 500 px，则 h = 500/1.5 = 333 px；h = 300 px，则 w = 300x1.5 = 450 px。若限制在指定 w 与 h 的矩形内的最大图片，即缩放图的 w 乘 h 不能大于 500 px 乘 300 px。通过以上条件得出缩略图大小为 450 px 乘以 300 px。在 pad 缩放规则下，需要对缩略图和 w 乘 h 矩形框中间的部分进行填充。下图示例为填充绿色背景。

情形 2：缩放参数为 w = 900 px，h = 800 px。按照 lift 等比缩放规则，缩放图的 w/h 等于原图的 w/h=1.5。所以，若：w = 900 px，则 h = 900/1.5 = 600 px；h = 800 px，则 w = 800x1.5 = 1200 px。若限制在指定 w 与 h 的矩形内的最大图片，即缩放图的 w 乘 h 不能大于 900 px 乘 800 px。通过以上条件得出缩略图大小为 900 px 乘以 600 px。在 pad 缩放规则下，需要对缩略图和 w 乘 h 矩形框中间的部分进行填充。下图示例为填充绿色背景。



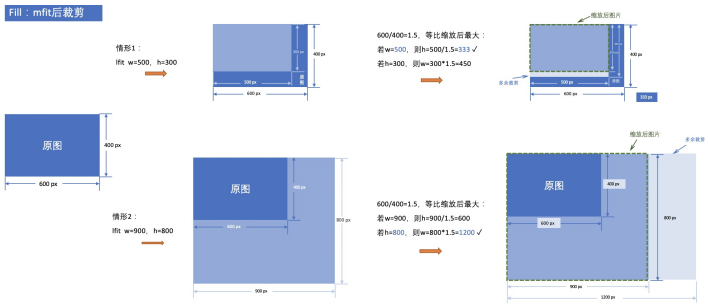
- **fill缩放**

若原图大小为 600 px 乘以 400 px。

情形 1：缩放参数为 w = 500 px，h = 300 px。按照 mift 等比缩放规则，缩放图的 w/h 等于原图的 w/h=1.5。所以，若：w = 500 px，则 h = 500/1.5 = 333 px；h = 300 px，则 w = 300x1.5 = 450 px。若限制为指定 w 与 h 的矩形框外的最小图片，即缩放图的 w 乘 h 不能小于 500 px 乘 300 px。通过以上条件得出缩略图大小为 500 px 乘以 333 px。在 fill 缩放规则下，需要对缩略图和 w 乘 h 矩形框中间的部分进行裁剪。

情形 2：缩放参数为 w = 900 px，h = 800 px。按照 lift 等比缩放规则，缩放图的 w/h 等于原图的 w/h=1.5。所以，若：w = 900 px，则 h = 900/1.5 = 600 px；h = 800 px，则 w = 800x1.5 = 1200 px。若限制为指定 w 与 h 的矩形框外的最小图片，即缩放图的 w 乘 h 不能小于 1200 px 乘 800 px。通过以上条件得出缩略图大小为 1200 px 乘以 800 px。在 fill 缩放规则下，需要对缩略图和 w 乘 h 矩形框中间的部分进行裁

剪。

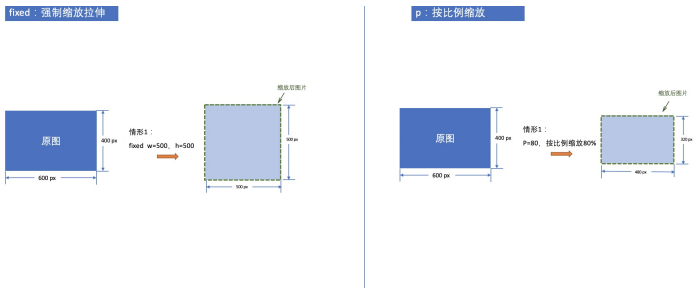


- **fixed缩放**

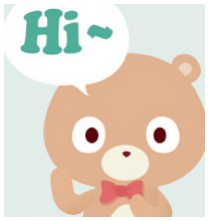
若原图大小为 600 px 乘以 400 px，缩放参数为 w = 500 px，h = 300 px。按照 fixed 缩放规则，需要对原图按照 w 和 h 进行强制拉伸平铺。

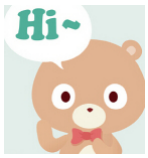
- **按比例缩放**

若原图大小为 600 px 乘以 400 px，缩放参数为 p = 80。按照等比缩放规则，需要对原图按照 80% 的比例进行按比例缩放，缩略图的长和宽都将缩放为原图尺寸的 80%。



使用案例

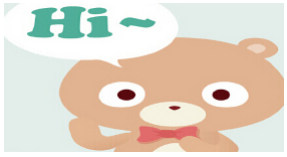
- 等比缩放图片(lfit)，缩放后的图片宽最大200，高最大100。
  https://doc.bce.baidu.com/bce-documentation/BOS/image.jpg?x-bce-process=image/resize,m_lfit,w_200,h_100
  
- 等比缩放图片(mfit)，缩放后的图片宽最小200，高最小100。
  https://doc.bce.baidu.com/bce-documentation/BOS/image.jpg?x-bce-process=image/resize,m_mfit,w_200,h_100
  
- 等比缩放图片(fill)，缩放后的图片宽最小200，高最小100，并对超出200*100部分图片进行居中裁剪(默认居中裁剪)。
  https://doc.bce.baidu.com/bce-documentation/BOS/image.jpg?x-bce-process=image/resize,m_fill,w_200,h_100
  
- 等比缩放图片(fill)，缩放后的图片宽最小200，高最小100，并对超出200*100部分图片从顶部开始裁剪。
  https://doc.bce.baidu.com/bce-documentation/BOS/image.jpg?x-bce-process=image/resize,m_fill,w_200,h_100,align_0
  
- 等比缩放图片(pad)，缩放后的图片宽最大200，高最大100，并对200*100矩形内图片未占满的部分进行填充。
  https://doc.bce.baidu.com/bce-documentation/BOS/image.jpg?x-bce-process=image/resize,m_pad,w_200,h_100,color_FF0000
  
- 强制缩放(fixed)，缩放后的图片宽200，高100。
  https://doc.bce.baidu.com/bce-documentation/BOS/image.jpg?x-bce-process=image/resize,m_fixed,w_200,h_100
  
- 等比缩放图片，按照原图50%的比例缩放。
  https://doc.bce.baidu.com/bce-documentation/BOS/image.jpg?x-bce-process=image/resize,p_50



- 比例缩放宽度图片，宽度按照原图50%的比例缩放。
  https://doc.bce.baidu.com/bce-documentation/BOS/image.jpg?x-bce-process=image/resize,wp_50



- 比例缩放高度图片，高度按照原图50%的比例缩放。
  https://doc.bce.baidu.com/bce-documentation/BOS/image.jpg?x-bce-process=image/resize,hp_50



- 等比缩放图片(pad)，缩放后的图片宽最大200，高最大100，并对200*100矩形内图片未占满的部分进行背景模糊。
  https://doc.bce.baidu.com/bce-documentation/BOS/image.jpg?x-bce-process=image/resize,m_pad,w_200,h_100,radius_2,sigma_50



图像裁剪

普通裁剪

概述

本文介绍对图片进行矩形裁剪操作。

普通裁剪参数

action名称：`crop`

参数	取值	说明
x	0-图片宽度	裁剪左上角起点相对原点的x坐标，非必选，默认0。
y	0-图片高度	裁剪左上角起点相对原点的y坐标，非必选，默认0。
w	1-4096	裁剪后图片的宽度，如果指定的宽度超过了图片的宽度，则以图片宽度为准裁剪。单位为px。非必选，默认值原始图片宽度。
h	1-4096	裁剪后图片的高度，如果指定的高度超过了图片的高度，则以图片高度为准裁剪。单位为px。非必选，默认值原始图片高度。

示例

- 对图片以（10,10）坐标为起点，裁剪200*200.
  https://doc.bce.baidu.com/bce-documentation/BOS/image.jpg?x-bce-process=image/crop,x_10,y_10,w_200,h_200



内切圆裁剪

概述

本文介绍以图片中心为圆心，进行内切圆裁剪的操作。

内切圆裁剪参数

action名称：`circle`

参数	取值	说明
r	0-图片最小边大小的一半	以图片中心为圆心，从图片取出的半径为r的圆形区域，r如果超过最小边大小的一半，默认取原圆的最大内切圆。

示例

- 以100为半径，进行内切圆裁剪.
  https://doc.bce.baidu.com/bce-documentation/BOS/image.jpg?x-bce-process=image/circle,r_100



圆角矩形裁剪

概述

本文介绍图片圆角矩形裁剪。

圆角矩形裁剪

action名称：`rounded-corners`

参数	取值	说明
r	1-4096	设置圆角的半径。如果指定半径r大于原图宽/高，则默认原图宽/高。

示例

- 以50为半径，进行圆角矩形裁剪。

https://doc.bce.baidu.com/bce-documentation/BOS/image.jpg?x-bce-process=image/rounded-corners,r_50



索引裁剪

概述

本文介绍图片索引裁剪操作。

索引裁剪参数

action名称：`indexcrop`

参数	取值	说明
x	1-4096	x与y不能同时指定，且必须指定其一。纵向切割，指定切割宽度。如果超过图片宽度，则默认为图片宽度。
y	1-4096	x与y不能同时指定，且必须指定其一。横向切割，指定切割高度。如果超过图片高度，则默认为图片高度。
i	0-65536	指定切割后的第i块切割区域。如果超过切割区域数，则返回原图。不指定默认为0。

示例

- 将图片以高度50横向切割，选择切割后的第1块和第2块。

https://doc.bce.baidu.com/bce-documentation/BOS/image.jpg?x-bce-process=image/indexcrop,y_50,i_0



https://doc.bce.baidu.com/bce-documentation/BOS/image.jpg?x-bce-process=image/indexcrop,y_50,i_1



- 将图片以宽度100纵向切割，选择切割后的第1块和第2块。

https://doc.bce.baidu.com/bce-documentation/BOS/image.jpg?x-bce-process=image/indexcrop,x_100,i_0



https://doc.bce.baidu.com/bce-documentation/BOS/image.jpg?x-bce-process=image/indexcrop,x_100,i_1



图像旋转

普通旋转

概述 本文介绍对图片进行一定角度的普通旋转。

普通旋转参数

action名称：`rotate`

参数	取值	说明
a	-360-360	图片旋转角度，正数表示顺时针旋转，负数表示逆时针旋转。

限制：图片的宽或者高不能超过4096 px

示例

- 将图片顺时针旋转30度

https://doc.bce.baidu.com/bce-documentation/BOS/image.jpg?x-bce-process=image/rotate,a_30



自适应旋转

概述

图像可以根据 exif 信息进行自适应旋转。

自适应旋转参数

action名称：`auto-orient`

参数	取值	说明
o	0,1	0：按原图默认方向，不自动旋转；1：自适应旋转。

限制：图片的宽或者高不能超过4096 px

渐近显示

概述

BOS 支持指定图片显示方式。

当网络环境较差或图片较大时，图片在网页上有两种显示方式：

- 标准显示：图片会按照从上到下的顺序一行一行地加载显示。
- 渐进显示：先显示整个图片的模糊轮廓，然后逐渐加载直至显示完整的图片。

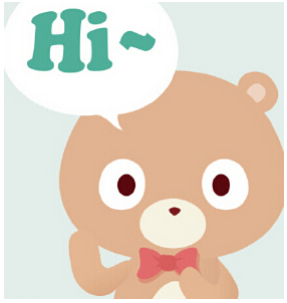
渐近显示参数

action名称：`interlace`

参数	取值	说明
i	baseline，progressive	baseline：标准式，指图片按照从上到下的扫描方式显示；progressive：渐进式 ，指图片由模糊到清晰渐进显示。progressive 只对jpg、png格式图片有效，若指定其他格式，则默认显示方式仍为baseline。

示例

- 图片指定渐进显示。
  https://doc.bce.baidu.com/bce-documentation/BOS/image.jpg?x-bce-process=image/interlace,i_progressive



质量变换

概述

设置图片编码质量。

质量只对有损压缩的图片格式生效，包括JPEG, WebP, Heic等。对于无损压缩的图片格式，包括PNG, TIFF，BMP, GIF等，则不会产生效果。

质量参数

命令名称：`quality`。

参数	取值	说明
q	1-100	指定图片的相对质量，假设原图质量为N，输出质量为N * q%。
Q	1-100	指定图片的绝对质量，假设原图质量为N，输出质量为Q。
c	le, ge, any	指定图片的绝对质量Q的生效条件，假设原图质量为N，如果为le，则Q < N时生效；如果为ge，则Q > N时生效；如果为any，则任意条件生效。默认为le。

注意事项：

1. 不同图片格式的质量没有可对比性，比如JPEG的质量75和WebP的质量75，两者之间无论画质、体积都没有可对比性；
2. 质量参数越高画质越好，但图片体积也会增大，推荐平衡画质和体积的质量：JPEG是75，WebP是75，Heic是45；
3. 参数Q配合c_ge可以提升图片的编码质量，但只有在缩略图的情况下，才会提高图片画质，在没有缩略的情况下，并不会增强图片画质；

示例

- 将图片质量设置为当前质量的80%
  https://doc.bce.baidu.com/bce-documentation/BOS/image.jpg?x-bce-process=image/quality,q_80
  

- 将图片质量设置为80
  https://doc.bce.baidu.com/bce-documentation/BOS/image.jpg?x-bce-process=image/quality,Q_80
  

图像水印

添加文字水印

概述

对一张图片添加文字水印。

文字水印参数

命令名称：`watermark`。

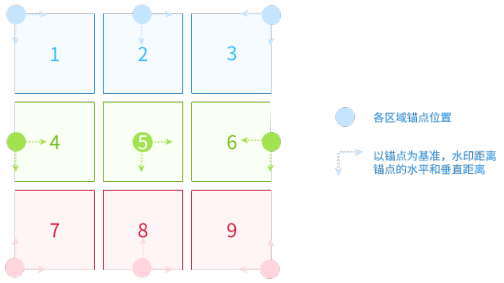
参数	取值	说明
text	-	水印文字内容，需要进行url安全的Base64编码，且编码后最多不能超过200个字符。文字水印必选。
size	1-1024	指定水印字体大小。非必选，默认30。
color	8位十六进制颜色码	指定字体颜色，格式为RRGGBBAA。最后2位Alpha通道可省略，省略时默认为00表示不透明。非必选，默认黑色000000。
type	支持的字体见下表	指定字体的字体类型，取值要求进行url安全的Base64编码。非必选。若指定的字体不支持的字体列表中，则为默认为宋体。
style	normal、italic、bold	指定字体样式。normal（常规）italic（斜体）bold（粗体）。非必选，默认normal。只有当字体支持style效果时，才会渲染文字效果。
g	1-9	指定水印的锚点位置。非必选，默认9。具体含义见下方说明。
x	-4096-4096	指定水印基于锚点的水平方向距离。非必选，默认值10。
y	-4096-4096	指定水印基于锚点的垂直方向距离。非必选，默认值10。
a	-360-360	指定水印旋转角度。正数表示顺时针旋转，负数表示逆时针旋转。非必选，默认值0。
skw	1-1024	指定文字描边宽度。非必选，默认值0。
skc	8位十六进制颜色码	指定文字描边颜色，格式为RRGGBBAA。最后2位Alpha通道可省略，省略时默认为00表示不透明。非必选，默认黑色000000。
blr	1-50	指定描边高斯模糊半径。非必选，默认值0。
bls	1-50	指定描边高斯模糊标准差。非必选，默认值0。
layout	tile	指定水印布局。tile（平铺）。非必选，默认值 无，具体效果见示例。

水印编码 在添加水印操作中，水印的文字内容、文字字体等参数需要进行URL安全的Base64编码。编码步骤如下：

- 将内容编码成Base64。将结果中的部分编码替换。
- 将结果中的加号（+）替换成短划线（-）。
- 将结果中的正斜线（/）替换成下划线（_）。
- 将结果中尾部的所有等号（=）省略。

锚点位置

将原始图片分成如下图所示9个区域，并按照图示位置进行编号，每个区域的锚点位置如图所示：



支持的字体类型

字体	名称	base64编码
FZLanTingHei	方正兰亭黑简体	RlpMYW5UaW5nSGVp
FZLanTingHeiSBGB	方正兰亭粗黑简体	RlpMYW5UaW5nSGVpU0JHQg==
FZLanTingHeiSDB1GB	方正兰亭中粗黑简体	RlpMYW5UaW5nSGVpU0RCMUdC
FZLanTingHeiSDBGB	方正兰亭中黑简体	RlpMYW5UaW5nSGVpU0RCR0I=
FZLanTingHeiSEBGB	方正兰亭大黑简体	RlpMYW5UaW5nSGVpU0VCR0I=
FZLanTingHeiSELGB	方正兰亭纤黑简体	RlpMYW5UaW5nSGVpU0VMR0I=
FZLanTingHeiSHGB	方正兰亭特黑简体	RlpMYW5UaW5nSGVpU0hHQg==
FZLanTingHeiSLGB	方正兰亭细黑简体	RlpMYW5UaW5nSGVpU0xHQg==
FZLanTingHeiSMGB	方正兰亭准黑简体	RlpMYW5UaW5nSGVpU01HQg==
FZLanTingHeiSRGB	方正兰亭黑简体	RlpMYW5UaW5nSGVpU1JHQg==
FZLanTingHeiSULGB	方正兰亭超细黑简体	RlpMYW5UaW5nSGVpU1VMR0I=
FZKaTongM19S	方正卡通简体	RlpLYVRvbmdNMTIT
FZFangSong	方正仿宋简体	RlpGYW5nU29uZw==
FZShuSong	方正书宋简体	RlpTaHVTb25n
FZKai	方正楷体简体	RlpLYWk=
FZHei	方正黑体简体	RlpIZWk=
FZDuHeiS	方正嘟黑简体	RlpEdUhlaVM=

示例

- 添加文字水印，文字为蓝色

  https://doc.bce.baidu.com/bce-documentation/BOS/image.jpg?x-bce-process=image/watermark,text_aGVsbG8=,type_RlpMYW5UaW5nSGVp,g_9,color_0000ff



- 添加文字水印，文字为白色，使用蓝色描边

  https://doc.bce.baidu.com/bce-documentation/BOS/image.jpg?x-bce-process=image/watermark,text_aGVsbG8=,type_RlpMYW5UaW5nSGVp,size_36,g_9,color_ffffff,skw_1,skc_0000ff



- 添加文字水印，使用平铺布局

  https://doc.bce.baidu.com/bce-documentation/BOS/image.jpg?x-bce-process=image/watermark,text_aGVsbG8=,size_30,g_3,x_20,y_20,a_45,layout_tile



添加图片水印

概述

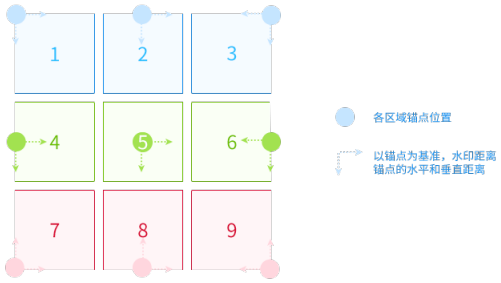
本文介绍对一张图片添加图片水印。

图片水印参数

action名称：`watermark`

图片水印参数	取值	说明
bucket	-	指定水印图片存储在BOS的Bucket名称，非必选，默认当前bucket。非当前bucket则要求该水印图片有公共读权限。
image	-	指定水印图片存储在BOS的Object名称，可以包含预处理命令，需要取url安全的Base64编码后的值。图片水印必选参数。
t	1-100	指定图片水印透明度，取值为100时，表示与原始透明度一致。非必选，默认值100。
g	1-9	指定水印的锚点位置。非必选，默认9。具体含义见下方说明。
x	-4096-4096	指定水印基于锚点的水平方向距离。非必选，默认值10。
y	-4096-4096	指定水印基于锚点的垂直方向距离。非必选，默认值10。
a	-360-360	指定水印旋转角度。正数表示顺时针旋转，负数表示逆时针旋转。非必选，默认值0。
P	1-100	指定水印缩放比例。非必选，默认值20。

> **说明：**
>
> - 目前不支持在一组action中同时指定图片和文字水印，只能指定图片或文字水印一种类型。如果需要指定图片和文字水印，可通过多组action实现。
> - 锚点位置说明，将您的原始图片分成如下图所示9个区域，并按照图示位置进行编号，每个区域的锚点位置如图所示：



水印编码 在添加水印操作中，水印的图片水印的水印图片名称等等参数需要进行URL安全的Base64编码。编码步骤如下：

- 将内容编码成Base64。将结果中的部分编码替换。
- 将结果中的加号（+）替换成短划线（-）。
- 将结果中的正斜线（/）替换成下划线（_）。
- 将结果中尾部的所有等号（=）省略。

示例

- 添加图片水印

https://doc.bce.baidu.com/bce-documentation/BOS/image.jpg?x-bce-process=image/watermark,image_YmNlLWRvY3VtZW50YXRpb24vQk9TL2JjZS5qcGc=,g_9,a_90



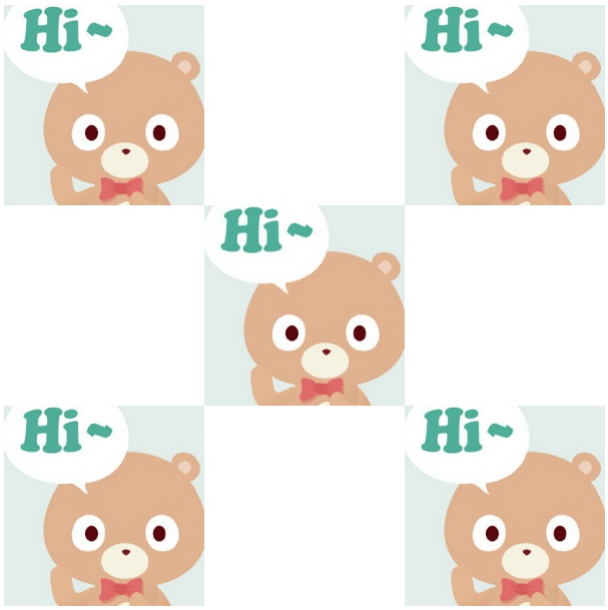
- 设置水印图比例
- https://doc.bce.baidu.com/bce-documentation/BOS/image.jpg?x-bce-process=image/watermark,image_YmNlLWRvY3VtZW50YXRpb24vQk9TL2JjZS5qcGc=,P_15,g_3



- 水印图预处理
- image值为 bce-documentation/BOS/bce.jpg?x-bce-process=image/resize,w_50/rotate,a_90,进行url safe base64处理
- https://doc.bce.baidu.com/bce-documentation/BOS/image.jpg?x-bce-process=image/watermark,image_YmNlLWRvY3VtZW50YXRpb24vQk9TL2JjZS5qcGc_eC1iY2UtcHJvY2Vzcz1pbWFnZS9yZXNpemUsd181MC9yb3RhdGUsYV85MA==



- 图片九宫格
- 可支持九张图片拼接
- https://doc.bce.baidu.com/bce-documentation/BOS/white-600x600.png?x-bce-process=image/watermark,image_YmNlLWRvY3VtZW50YXRpb24vQk9TL2ItYWdlLmpwZz94LWJjZS1wcm9jZXNzPWItYWdlL3Jlc2l6ZSxtX2ZpbGwsd18yMDAsaF8yMDAsYWxpZ25fMzA=,w_200,x_0,y_0,g_1/watermark,image_Yn



添加图文混合水印

概述

本文介绍如何对图片添加图片和文字水印。命令名称：`watermark`。

图片水印参数

图片水印参数

参数	取值	说明
bucket	-	指定水印图片存储在BOS的Bucket名称，非必选，默认当前bucket。非当前bucket则要求该水印图片有公共读权限。
image	-	指定水印图片存储在BOS的Object名称，需要取url安全的Base64编码后的值。
t	1-100	指定图片水印透明度，取值为100时，表示与原始透明度一致。非必选，默认值100。

文字水印参数

参数	取值	说明
text	-	水印文字内容，需要进行url安全的Base64编码，且编码后最多不能超过200个字符。
size	1-1024	指定水印字体大小。非必选，默认30。
color	8位十六进制颜色码	指定字体颜色。非必选，默认黑色000000。
type	支持的字体见下表	指定字体的字体类型，取值要求进行url安全的Base64编码。非必选。若指定的字体不支持的字体列表中，则为默认为宋体。
style	normal, italic, bold	指定字体样式。normal（常规）italic（斜体）bold（粗体）。非必选，默认normal。只有当字体支持style效果时，才会渲染文字效果。
effect	hardshadow, softoutline	文字特效。 hardshadow：文字投影；softoutline：文字投影，投影增加高斯模糊。
shx	0-10	文字特效水平方向的偏移。非必选，默认值0。
shy	0-10	文字特效垂直方向的偏移。非必选，默认值0。
shc	8位十六进制颜色码	指定阴影颜色。
blr	1-50	指定描边高斯模糊半径。只在softoutline中生效。非必选，默认值0。
bls	1-50	指定描边高斯模糊标准差。只在softoutline中生效。非必选，默认值0。

公共参数

参数	取值	说明
g	1-9	指定水印的锚点位置。非必选，默认9。具体含义见下方说明。
x	-4096-4096	指定水印基于锚点的水平方向距离。非必选，默认值10。
y	-4096-4096	指定水印基于锚点的垂直方向距离。非必选，默认值10。
a	-360-360	指定水印旋转角度。正数表示顺时针旋转，负数表示逆时针旋转。非必选，默认值0。
order	0,1	指定文字和图片水印的前后顺序。0（默认值）：表示图片水印在前；1：表示文字水印在前。
align	0,1,2	指定文字水印和图片水印的对齐方式。 0：文字水印和图片水印上对齐；1：文字水印和图片水印居中对齐；2（默认值）：文字水印和图片水印下对齐。
interval	0-1000	指定文字水印和图片水印间的间距。非必选，默认值0。

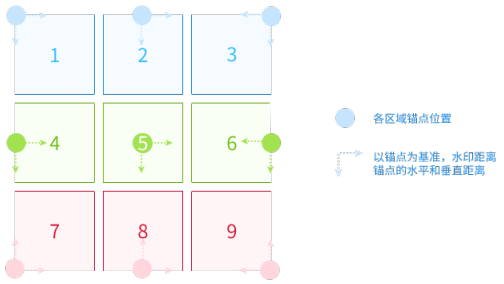
水印编码 在添加水印操作中，水印的文字内容、文字字体、图片水印的水印图片名称等参数需要进行URL安全的Base64编码。编码步骤如下：

- 将内容编码成Base64。将结果中的部分编码替换。
- 将结果中的加号（+）替换成短划线（-）。
- 将结果中的正斜线（/）替换成下划线（_）。

- 将结果中尾部的所有等号（=）省略。

锚点位置

将原始图片分成如下图所示9个区域，并按照图示位置进行编号，每个区域的锚点位置如图所示：



支持的字体类型

字体	名称	base64编码
FZLanTingHei	方正兰亭黑简体	RlpMYW5UaW5nSGVp
FZLanTingHeiSBGB	方正兰亭粗黑简体	RlpMYW5UaW5nSGVpU0JHQg==
FZLanTingHeiSDB1GB	方正兰亭中粗黑简体	RlpMYW5UaW5nSGVpU0RCMUdC
FZLanTingHeiSDBGB	方正兰亭中黑简体	RlpMYW5UaW5nSGVpU0RCR0I=
FZLanTingHeiSEBGB	方正兰亭大黑简体	RlpMYW5UaW5nSGVpU0VCR0I=
FZLanTingHeiSELGB	方正兰亭纤黑简体	RlpMYW5UaW5nSGVpU0VMR0I=
FZLanTingHeiSHGB	方正兰亭特黑简体	RlpMYW5UaW5nSGVpU0hHQg==
FZLanTingHeiSLGB	方正兰亭细黑简体	RlpMYW5UaW5nSGVpU0xHQg==
FZLanTingHeiSMGB	方正兰亭准黑简体	RlpMYW5UaW5nSGVpU01HQg==
FZLanTingHeiSRGB	方正兰亭黑简体	RlpMYW5UaW5nSGVpU1JHQg==
FZLanTingHeiSULGB	方正兰亭超细黑简体	RlpMYW5UaW5nSGVpU1VMR0I=
FZKaTongM19S	方正卡通简体	RlpLYVRvbmdNMTIT
FZFangSong	方正仿宋简体	RlpGYW5nU29uZw==
FZShuSong	方正书宋简体	RlpTaHVTb25n
FZKai	方正楷体简体	RlpLYWk=
FZHei	方正黑体简体	RlpIZWk=
FZDuHeiS	方正嘟黑简体	RlpEdUhIaVM=

示例

- 图文混合水印，居中对齐
  https://doc.bce.baidu.com/bce-documentation/BOS/image.jpg?x-bce-process=image/watermark,image_YmNlLWRvY3VtZW50YXRpb24vQk9TL2JjZS5qcGc=,text_55m-5bqm5LqR,type_ZnpsYW50aW5naGVpc3JnYg==,align_1



- 图文混合水印，文字在前，居中对齐
  https://doc.bce.baidu.com/bce-documentation/BOS/image.jpg?x-bce-process=image/watermark,image_YmNlLWRvY3VtZW50YXRpb24vQk9TL2JjZS5qcGc=,text_55m-5bqm5LqR,type_ZnpsYW50aW5naGVpc3JnYg==,order_1,align_1



- 图文混合水印，文字硬阴影效果
  https://doc.bce.baidu.com/bce-documentation/BOS/image.jpg?x-bce-process=image/watermark,image_YmNlLWRvY3VtZW50YXRpb24vQk9TL2JjZS5qcGc=,text_55m-5bqm5LqR,type_ZnpsYW50aW5naGVpc3JnYg==,size_32,color_FFFFFFFF,effect_hardshadow,shx_2,shy_2,shc_696969,align_1



- 图文混合水印，文字软阴影效果
  https://doc.bce.baidu.com/bce-documentation/BOS/image.jpg?x-bce-process=image/watermark,image_YmNlLWRvY3VtZW50YXRpb24vQk9TL2JjZS5qcGc=,text_55m-5bqm5LqR,type_ZnpsYW50aW5naGVpc3JnYg==,size_32,color_FFFFFFFF,effect_softoutline,shx_2,shy_2,shc_696969,bls_2,align_1



获取图片信息

概述

您可以通过命令来获取图片信息。如果图片有 exif 信息，则返回包含 exif 的完整信息，如果图片不包含 exif 信息，则返回图片的基本信息。

> **说明：**exif信息记录了数码照片的属性信息和拍摄数据，并非每一张图片都包含exif信息。

返回的exif信息包含但不限于：

- dateTime、dateTimeOriginal、dateTimeDigitized、format、gpsLatitude、gpsLatitudeRef、gpsLongitude、gpsLongitudeRef、imageHeight、imageWidth、imageSize、make、model、orientation、resolutionX、resolutionY、resolutionUnit

获取图片信息参数

action名称：`info`

此命令没有参数，以 x-bce-process=image/info 形式访问即可。

格式转换

概述

图片格式转换。

格式转换参数

action名称：`format`

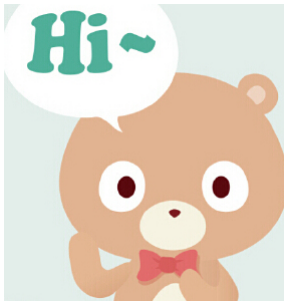
参数	取值	说明
f	jpg、png、bmp、webp、heic、gif、auto	指定目标图片的图片格式

如果值为`auto`则根据请求Header的`Accept`字段自动选择最优格式，当前支持自适应为`webp`，如果无法匹配则保持原图格式。响应头中会按照HTTP规范返回`Vary: Accept`。

需要特别注意的是，使用`auto`时，单个链接会有多个缓存副本，如果使用百度智能云CDN，则需要CDN开启多副本缓存功能，如果没有开启此功能，CDN会缓存第一次请求的结果，当第一次请求返回的是`webp`格式时，会导致safari、ie等浏览器看不到图片。2020/01/01之后创建CDN域名默认都开启了此功能，之前创建的域名需要提工单开启。

示例

- 格式自适应，在Chrome等支持webp的浏览器下返回为webp格式，不支持webp的浏览器中返回为原图格式。
  https://doc.bce.baidu.com/bce-documentation/BOS/image.jpg?x-bce-process=image/format,f_auto



灰度变换

概述

将图片转换为灰图。

灰度变换参数

命令名称：grayscale。

示例

- 将图片转换为灰图
  https://doc.bce.baidu.com/bce-documentation/BOS/image.jpg?x-bce-process=image/grayscale
  

图片锐化

概述

本文介绍图片锐化操作。

锐化参数

action名称：sharpen

参数	取值	说明
r	1-50	锐化的半径
s	1-50	锐化的标准差

示例

- 将图片以半径为2 标准差为50进行锐化处理
  https://doc.bce.baidu.com/bce-documentation/BOS/image.jpg?x-bce-process=image/sharpen,r_2,s_50
  

图片模糊

概述

本文介绍图片模糊操作。

模糊参数

action名称：blur

参数	取值	说明
r	1-50	模糊的半径
s	1-50	模糊的标准差

示例

- 将图片以半径为2 标准差为50进行模糊处理
  https://doc.bce.baidu.com/bce-documentation/BOS/image.jpg?x-bce-process=image/blur,r_2,s_50
  

图片亮度

概述

本文介绍图片亮度操作。

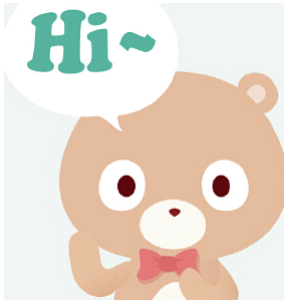
亮度参数

action名称：bright

参数	取值	说明
b	-100 - 100	设置图片亮度。当取值小于0时，图片变暗；当取值等于0时，图片亮度不变；当取值大于0时，图片变亮。

示例

- 将图片以参数-5进行变暗处理
  https://doc.bce.baidu.com/bce-documentation/BOS/image.jpg?x-bce-process=image/bright,b_-5
  

- 将图片以参数5进行变亮处理
  https://doc.bce.baidu.com/bce-documentation/BOS/image.jpg?x-bce-process=image/bright,b_5
  

去除元信息

概述

您可以通过命令来可去除图片元信息，包括 exif 信息。

> **说明：**exif信息记录了数码照片的属性信息和拍摄数据，并非每一张图片都包含exif信息。

返回的exif信息包含但不限于：

- dateTime
- dateTimeOriginal
- dateTimeDigitized
- format
- gpsLatitude
- gpsLatitudeRef
- gpsLongitude
- gpsLongitudeRef
- imageHeight
- imageWidth
- imageSize
- make
- model
- orientation
- resolutionX
- resolutionY
- resolutionUnit

去除图片元信息参数

action名称：strip

此命令没有参数，以 x-bce-process=image/strip 形式访问即可。

图片对比度

概述

对比度是指一幅图像中明暗区域最亮的白和最暗的黑之间不同亮度层级的测量，即指一幅图像灰度反差的大小。

亮度参数

action名称：`contrast`

参数	取值	说明
c	-100 - 100	取值＜0：降低图片对比度。取值=0：维持原图对比度。取值＞0：提高图片对比度。

示例

- 对比度降低50

  https://doc.bce.baidu.com/bce-documentation/BOS/image.jpg?x-bce-process=image/contrast,c_-50

- 对比度提高10

  https://doc.bce.baidu.com/bce-documentation/BOS/image.jpg?x-bce-process=image/contrast,c_10

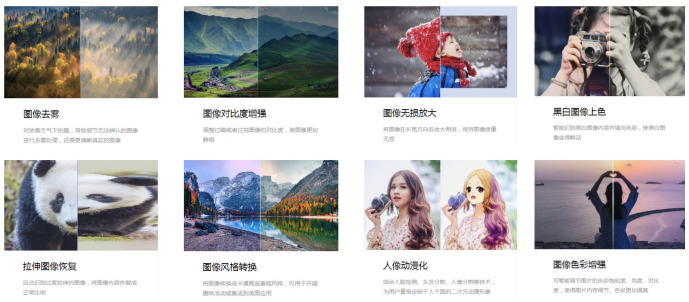
  

智能处理

智能处理使用规则

智能图片处理概述

图片处理服务依托对象存储 BOS 系统对外提供灵活、多样、实时的图片处理服务。用户将原始图片上传到 BOS 上，通过调用简单的 RESTful 接口，实现对图片的实时处理，如缩略、裁剪、格式转换、旋转、加文字/图片水印等，处理结果图会通过CDN实时的返回给用户。智能图片处理则通过 BOS 图片处理平台对外开放百度领先的 AI 图片处理能力。



场景与解决方案

智能图像处理包括智能效果和智能场景两部分，您可以根据自己的业务场景选择合适的智能处理方式。

典型场景	场景需求	解决方案	推荐功能
手机云相册	手机相册需要具备如裁剪、旋转、调节对比度等处理功能	手机 APP 将用户上传的照片存储到 BOS。当用户再次读取时，直接调用 BOS 原生图像处理功能进行处理，并将处理后图像作为副本继续存储在云相册	• BOS 基础图像处理功能（缩放、裁剪、旋转等）； • 智能能力如对比度、清晰度、色彩增强等图像效果能力； • 事件通知触发通用物体/人脸识别；
安防	阴雨天或大雾天导致捕获图像的清晰度较低	安防设备将捕获的视频或者图像存储到 BOS，调用 BOS 原生图像处理能力如图像去雾，以提升文件清晰度	• BOS 基础图像处理功能（缩放、裁剪、旋转等）； • 智能图像处理：图像去雾； • MCT 视频转码，以降低存储费用；
电商	图片失真、水印去除、对图像尺寸进行缩放	将图像存储到 BOS，在读取图像时自动进行无损放大，并添加水印或者其他基础处理样式；	• BOS 基础图像处理功能（缩放、裁剪、旋转、水印等）； • 智能图像处理能力：无损放大、物体擦除；
社交	社交产品需要一定的产品趣味性来拉新留存	社交 APP 将图像存储到 BOS，在读取图像时自动进行人像动漫化或者风格转换	• BOS 基础图像处理功能（缩放、裁剪、旋转等）； • 智能图像处理：人像动漫化、风格转换等； • 事件通知触发通用物体/人脸识别；
促销活动	重大节日或促销场景，商家运营活动需要增添一定趣味性	将图像存储到 BOS，在读取图像时自动进行无损放大，并添加水印或者其他基础处理样式；	• BOS 基础图像处理功能（缩放、裁剪、旋转等）； • 智能图像处理：人像动漫化、风格转换、黑白上色等； • 事件通知触发通用物体/人脸识别；

使用方式

您需要先将原始图片上传到 BOS，每个 BOS Bucket 默认开通图片处理的功能，无需单独开通，也无需使用特定域名访问。当访问图片 URL 时，在 Querystring 中通过 **x-bce-process** 参数指定图片处理命令，即可触发 BOS 图片处理功能。BOS 支持通过命令和样式两种方式访问图片服务。

- 命令方式访问格式：`http://${domain}/${objectkey}?x-bce-process=image/${action},${key}_${value}.`
  样例：`http://bucket-A.bj.bcebos.com/sample.jpg?x-bce-process=image/resize,m_fill,w_200,h_300`。
- 样式方式访问格式：`http://${domain}/${objectkey}?x-bce-process=style/${stylename}`. 样例：`http://bucket-A.bj.bcebos.com/sample.jpg?x-bce-process=style/stylename`。

> **说明：**
>
> - 智能图片处理仅支持图片处理 V2.0 版本，V1.0 版本暂不支持。

规则与限制

- 智能图片处理支持的原图格式：PNG、JPG、JPEG、BMP。
- 支持的目标格式：PNG、JPG、JPEG、BMP。
- 需要处理的图片 base64 编码后大小不超过 4M,最短边至少 64px，最长边最大 4096px，长宽比 3:1 以内。特殊说明的除外。
- 处理结果输出图片宽高不超过 4096 px，包括管道和子命令间的中间输出图片。
- BOS 图片处理 V2.0 最多可以同时指定 4 组操作，即 4 个 action，但只能**同时最多指定一组智能处理有关的 action**。
- 一个 action 下不可以重复指定同一个参数。
- 无损放大需要处理的图片像素乘积不超过 1600p x 1600px。
- 智能去雾需要处理的图片最短边至少 200px，最长边最大 4096px。
- 黑白上色需要处理的图片最短边至少 64px，最长边最大 800px。
- 天空分割需要处理的图片最短边至少 64px，最长边最大 2049px，像素乘积不超过 2049*2049。
- 物体擦除使用设置位置参数方法进行修复时，最短边至少 100px，最长边最大 2000px，长宽比 3：1内。

智能效果

智能去雾

概述 对浓雾天气下拍摄，导致细节无法辨认的图像进行去雾处理，还原更清晰真实的图像。

参数 action名称：dehaze

示例

- 示例原图与智能去雾处理后的效果图:

https://doc.bce.baidu.com/bce-documentation/BOS/src_dehaze.jpg?x-bce-process=image/dehaze





智能对比度

概述 调整过暗或者过亮图像的对比度，使图像更加鲜明。您无需指定对比度参数，BOS 会为您自适应最佳对必须效果。

参数 action名称：contrast-enhance

示例

- 示例原图与智能对比度处理后效果图：

https://doc.bce.baidu.com/bce-documentation/BOS/src_contrast_enhance.jpg?x-bce-process=image/contrast-enhance





智能清晰度

概述 对压缩后的模糊图像实现智能快速去噪，优化图像纹理细节，使画面更加自然清晰。您无需设置清晰度参数，BOS 会为您自适应最佳清晰度。

参数 action名称：definition-enhance

- 示例原图和智能清晰度处理后效果图：

https://doc.bce.baidu.com/bce-documentation/BOS/src_quality_enhance.jpg?x-bce-process=image/definition-enhance





智能调色

概述

可智能调节图片的色彩饱和度、亮度、对比度，使得图片内容细节、色彩更加逼真。智能调色可用于提升网站图片、手机相册图片、视频封面图片的质量。您无需指定色彩参数，BOS 会为您自适应最佳效果。

参数 action名称：color-enhance

示例

- 示例原图和智能调色处理后效果图：

https://doc.bce.baidu.com/bce-documentation/BOS/src_color_enhance.jpg?x-bce-process=image/color-enhance





智能场景

黑白上色

概述 智能识别黑白图像内容并填充色彩，使黑白图像变得鲜活。

参数 action名称：colourize

示例

- 黑白图片上色。如图是黑白图片上色前后的对比效果：

https://doc.bce.baidu.com/bce-documentation/BOS/src_black_and_white.jpg?x-bce-process=image/colourize





风格转换

概述

可将图像转化成卡通画、铅笔画、彩色铅笔画，或者哥特油画、彩色糖块油画、呐喊油画、神奈川冲浪里油画、奇异油画、薰衣草油画等共计 9 种风格。适用于社交，趣味活动等使用场景，或集成到美图应用中对图像进行风格转换。

参数 action名称：style-trans

参数	取值	说明
o	• cartoon：卡通画风格 • pencil：铅笔画风格 • color_pencil：彩色铅笔画风格 • warm：彩色糖块油画风格 • wave：神奈川冲浪里油画风格 • lavender：薰衣草油画风格 • mononoke：奇异油画风格 • scream：呐喊油画风格 • gothic：哥特油画风格	图像风格，必选。

示例

- 风格转换原图：



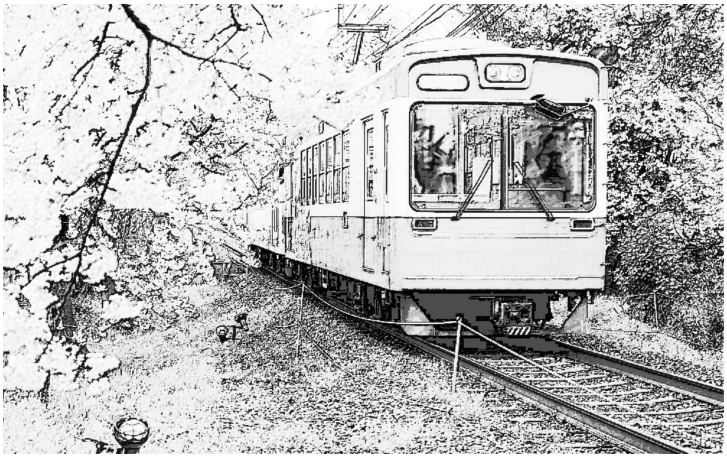
- 转换为卡通画风格：

https://doc.bce.baidu.com/bce-documentation/BOS/src_style_trans.jpg?x-bce-process=image/style-trans,o_cartoon



- 转换为铅笔画风格：

https://doc.bce.baidu.com/bce-documentation/BOS/src_style_trans.jpg?x-bce-process=image/style-trans,o_pencil



- 转换为彩色铅笔画风格：

https://doc.bce.baidu.com/bce-documentation/BOS/src_style_trans.jpg?x-bce-process=image/style-trans,o_color-pencil



- 转换为彩色糖块油画风格：

https://doc.bce.baidu.com/bce-documentation/BOS/src_style_trans.jpg?x-bce-process=image/style-trans,o_warm



- 转换为神奈川冲浪里油画风格：

https://doc.bce.baidu.com/bce-documentation/BOS/src_style_trans.jpg?x-bce-process=image/style-trans,o_wave



- 转换为薰衣草油画风格：

https://doc.bce.baidu.com/bce-documentation/BOS/src_style_trans.jpg?x-bce-process=image/style-trans,o_lavender



- 转换为奇异油画风格：

https://doc.bce.baidu.com/bce-documentation/BOS/src_style_trans.jpg?x-bce-process=image/style-trans,o_mononoke



- 转换为呐喊油画风格：

https://doc.bce.baidu.com/bce-documentation/BOS/src_style_trans.jpg?x-bce-process=image/style-trans,o_scream



- 转换为哥特油画风格：

https://doc.bce.baidu.com/bce-documentation/BOS/src_style_trans.jpg?x-bce-process=image/style-trans,o_gothic



人像动漫化

概述

运用世界领先的对抗生成网络，结合人脸检测、头发分割、人像分割等技术，为用户量身定制千人千面的二次元动漫形象，并且可通过参数设置，生成戴口罩的二次元动漫人像。

参数

action名称：selfie-anime

参数	取值	说明
t	"anime"或者"anime-mask"。	设置二次元动漫画是否戴口罩。前者anime生成二次元动漫图，后者anime-mask生成戴口罩的二次元动漫人像。非必选，默认是anime。
m	1～8之间的整数	在t参数填入anime-mask时生效，m是1～8之间的整数，用于指定所使用的口罩的编码。t参数没有填入anime-mask，或m参数为空时，生成不戴口罩的二次元动漫图。

示例

- 示例原图：



- 转换为二次元动漫形象：

https://doc.bce.baidu.com/bce-documentation/BOS/src_selfie_anime.jpg?x-bce-process=image/selfie-anime,t_anime



- 转换为佩戴 N95 口罩的二次元动漫形象，口罩编号为 1：

https://doc.bce.baidu.com/bce-documentation/BOS/src_selfie_anime.jpg?x-bce-process=image/selfie-anime,t_anime-mask,m_1



- 转换为佩戴普通蓝色口罩的二次元动漫形象，口罩编号为 2：

https://doc.bce.baidu.com/bce-documentation/BOS/src_selfie_anime.jpg?x-bce-process=image/selfie-anime,t_anime-mask,m_2



- 转换为佩戴普通白色口罩的二次元动漫形象，口罩编号为 3：

https://doc.bce.baidu.com/bce-documentation/BOS/src_selfie_anime.jpg?x-bce-process=image/selfie-anime,t_anime-mask,m_3



- 转换为佩戴重型蓝色口罩的二次元动漫形象，口罩编号为 4：

https://doc.bce.baidu.com/bce-documentation/BOS/src_selfie_anime.jpg?x-bce-process=image/selfie-anime,t_anime-mask,m_4



- 转换为佩戴重型棕色口罩的二次元动漫形象，口罩编号为 5：

https://doc.bce.baidu.com/bce-documentation/BOS/src_selfie_anime.jpg?x-bce-process=image/selfie-anime,t_anime-mask,m_5



- 转换为佩戴粉色笑脸口罩的二次元动漫形象，口罩编号为 6：

https://doc.bce.baidu.com/bce-documentation/BOS/src_selfie_anime.jpg?x-bce-process=image/selfie-anime,t_anime-mask,m_6



- 转换为佩戴重型口罩的二次元动漫形象，口罩编号为 7：

https://doc.bce.baidu.com/bce-documentation/BOS/src_selfie_anime.jpg?x-bce-process=image/selfie-anime,t_anime-mask,m_7



- 转换为佩戴粉色纯色口罩的二次元动漫形象，口罩编号为 8：

https://doc.bce.baidu.com/bce-documentation/BOS/src_selfie_anime.jpg?x-bce-process=image/selfie-anime,t_anime-mask,m_8



天空分隔

概述 可智能分割出天空边界位置，输出天空和其余背景的灰度图，可用于图像二次处理，进行天空替换、抠图等图片编辑场景。

参数

action名称：sky-seg
参数：无特殊参数

示例

- 示例原图和天空分隔效果图如下所示：

https://doc.bce.baidu.com/bce-documentation/BOS/src_sky_seg.jpg?x-bce-process=image/sky-seg





无损放大

概述 输入一张图片，可以在尽量保持图像质量的条件下，将图像在长宽方向各放大两倍。

参数 action名称：quality-enhance

示例

- 示例原图与无损放大效果图：

https://doc.bce.baidu.com/bce-documentation/BOS/src_quality_enhance.jpg?x-bce-process=image/quality-enhance





拉伸恢复

概述 自动识别过度拉伸的图像，将图像内容恢复成正常比例。

参数 action名称：stretch-restore

示例

- 示例原图与拉伸修复效果图：

https://doc.bce.baidu.com/bce-documentation/BOS/src_stretch_restore.jpg?x-bce-process=image/stretch-restore



物体擦除

概述 擦除图片中不需要的遮挡物，并用背景内容填充以提高图像质量。擦除位置为规则矩形，需要给出坐标和大小。

参数 action名称：inpainting

参数	取值	说明
x	0-2000	擦除物体左上角起点相对原点的 x 坐标，必选
y	0-2000	擦除物体左上角起点相对原点的 y 坐标，必选
w	1-2000	w，擦除区域的宽度，必选
h	1-2000	h，擦除区域的高度，必选

示例

- 物体擦除前后对比图：

https://doc.bce.baidu.com/bce-documentation/BOS/src_inpainting.jpg?x-bce-process=image/inpainting,x_502,y_158,w_275,h_229





图像审核

概述

BOS 图像审核服务是百度智能云对外提供的图像智能审核服务，基于深度学习的智能内容审核方案，准确识别图片和视频中的涉黄、涉暴涉恐、政治敏感、微商广告、恶心等内容。为了方便您使用，BOS 为开发者提供图像审核 API 或 CLI 调用的同时，也提供了界面化的图审控制台，用户可在控制台查看 BOS 审核结果中的违规图片，并按需进行人工复审。

注意事项

- 为避免违规图片传播，BOS 会将违规图片的权限暂时设为“私有权限”，通过身份识别后当您拥有图片读取权限才可访问。
- 在已配置增量图像自动审核规则的 Bucket 内，如果图片是由数据同步功能复制而来或生命周期策略产生，将不会被自动审核。
- 对于未取回的归档类型 Object，图审策略不会生效；已取回的归档类型 Object 可正常被审核。
- 2022年7月图像审核功能升级，将不再支持设置图像审核上限，且开通功能需要您先开通ICR后付费并请保持开通状态，从而确保您可享受无限的图像审核次数，保证审核业务不中断。
- 因BOS图像审核依赖内容审核平台（ICR）的图片审核功能，该服务条款完全遵循ICR条款，同时会产生图片审核接口调用费用，请参考ICR定价文档。ICR为您提供了免费使用额度，您可前往领取icr免费图像审核调用次数，图片审核产生的调用次数将优先抵扣该免费资源，不足抵扣部分将进行按量付费。

设置审核规则

1. 登录 BOS 管理控制台。
2. 点击左侧导航栏中的【图像审核】，阅读开通图像审核服务指引。

> 您需完成开通ICR服务同时开通其后付费才可设置图像审核规则。



3. 若您已经完成开通ICR并完成后付费，则可通过如下页面，点击【立即使用】，完成审核规则配置。



4. **设置审核规则**。
   - Bucket审核：选择需要审核的目标 Bucket，仅支持选择属于北京、上海、广州地域的Bucket。
   - 审核类型：选择需要审核哪些违规类型，目前提供了色情、暴恐和政治敏感三种违规审核能力。

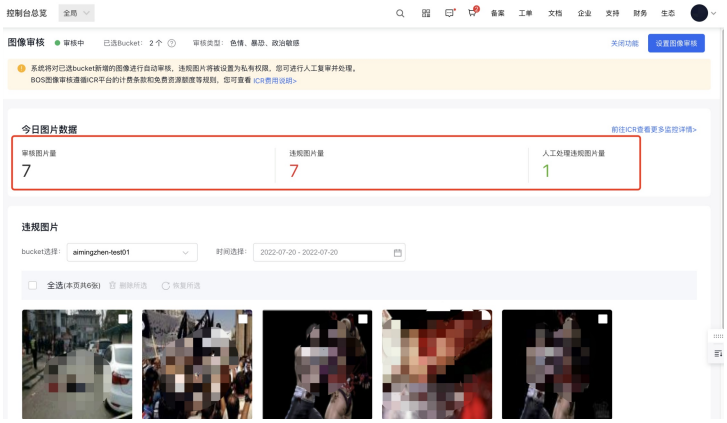


5. 点击**确认**，保存审核规则设置。

使用审核服务

查看审核数据

方式一： BOS 图审服务会监测审核您选择的 Bucket 中的所有新增图片，并将审核信息以按照【今日图像数据】统计概览的方式向您展示，信息一目了然。



- **审核图片量**：代表今天您开启图审的 所有Bucket 中，经过审核的图片总量。
- **违规图片量**：代表审核结果为违规图片的数量。
- **人工处理违规图片量**：代表基于违规图片量，管理员已经人工处理过的图片数量，操作包括：删除，恢复原有权限。

方式二： 您也可前往ICR查看更加具体的图像审核接口调用情况，可自定义时间范围，查询图像审核接口调用成功、失败情况。

违规图像复审

- 您可以通过选择Bucket 与时间维度的，选择展示您希望人工复审的违规图片。
- 您在对违规图片按需人工复审时，可对选中的图片执行查看详细信息、删除或恢复原有权限的操作。
- 您可以批量删除或者恢复违规图片进行，特别提示删除图像将彻底删除从BOS中删除，请谨慎操作。



API使用

开通图像审核功能后，您也可以通过调用图像审核 API 进行图像主动审核，可很好的满足存量图像的审核需求。

图像处理名词解释

概述

在您使用图像处理前，请先了解图像处理涉及到的一些基本概念。

基本概念

图片服务域名

图片服务域名，是指用来访问图片服务的域名。图片服务和普通 BOS 访问使用的是同一个域名，系统会为 Bucket 提供一个外网直接发布的官方域名即`<bucketname>.region.bcebos.com`，用户也可以添加自定义域名。如果 Bucket 开通了 CDN 加速发布，则图片服务同步开通 CDN 加速。如果要开通 Bucket 的 CDN加速发布，可参考 CDN 加速发布。

> **说明：**Console升级后不影响已有域名的图片处理。前期官方提供的bceimg.com域名主要用于客户试用图片服务，升级后将没有存在意义。对于已开通的bceimg.com域名，可继续使用半年，后续会收回。

处理命令

处理命令是指 URL 中用来转换处理图片的一段命令，形式为“缩写命令_参数值”。通过指定的处理命令参数值，BOS 生成并返回另一张转换处理后的图片。如 “w_250”，表示将图片的宽度处理为 250 px。

执行图片某一类处理方式的命令可称为一组命令，如：执行图片缩放功能为一组命令，执行图片裁剪功能为另一组命令。

样式

在对图片进行多重处理时，图片访问的 URL 会变得复杂冗长，非常不利于管理和阅读。为了让图片 URL 变得更简洁优雅，或是您想达到隐藏处理参数、保护原图的目的时，您可以将一组图像处理命令添加为一个图片样式，并对该样式进行自定义命名。创建样式后，您只可以直接调用样式名称来达到预期的图像处理效果，无需调用复杂冗长的命令名称。

分隔符

分隔符是指 URL 中用来区分一些关键字段的分隔符号。

图片信息

图片 exif 信息记录了数码照片的属性信息和拍摄数据，但并非每一张图片都包含 exif 信息。您可以通过命令获取图片的 exif 信息。

管道

管道用来连接实现多条独立命令的执行。如图像处理 V1.0 协议中的管道处理命令格式为`http://${userdomain}/${objectkey}@${commands 1}|${commands 2}`，执行顺序按管道指定顺序从左至右执行。

音视频处理

基本概念

音视频行业用户在将BOS作为后端存储源站时，结合上层业务逻辑往往会有不同的使用场景。 随着业务层对用户体验的不断演进，音视频业务对数据加工、处理、访问的实时性要求越来越高，并且希望提供望更加简化的使用方式。

百度智能云BOS提供了业界全新的音视频处理接口，支持上传mp4格式的文件，系统自动保存为hls格式，用于m3u8风格播放器读取使用。充分满足了“UGC上传——自动化处理——播放”这条使用流，免去了原本复杂的视频操作处理逻辑，并为业务提速。

音视频处理命令

音视频文件远大于图片文件，处理时间也较长，通常在秒级。因此，BOS音视频处理命令，采用了相较于BOS图片处理接口有所区别的写时处理方案，即音视频原文件写入BOS时即开始按需处理，而非基于用户读取时定义的参数处理。

BOS音视频处理接口定义如下：

- 格式转换参数

参数名称	缩写	类型	取值	命令描述	是否必选
format	f	string	[hls]	将视频转换转化为指定格式	是
time	t	int	[6,600]	切片时间，默认10s	否

- 持久化存储参数

参数名称	缩写	类型	取值	命令描述	是否必选
object	o	string	-	经过base64编码的object名称，受object长度限制，即不超过1024个字符	是

请求方法

在BOS的POST接口中，以header形式填写参数，即可实现POST上传的音视频文件，按照header中的命令实现转封装及转存。

```
 POST /video.mp4 HTTP/1.1
Content-Length: ContentLength
Content-Type: ContentType
x-bce-process: video/format,f_hls
```

请求示例

转封装并存储示例

将视频video.mp4转封装为hls，并将结果存储到video/demo.hls。

```
 POST /video.mp4 HTTP/1.1
Content-Length: ContentLength
Content-Type: ContentType
x-bce-process: video/format,f_hls|system/save,o_dmlkZW8vZGVtby5tM3U4
```

规则和限制

- 支持的原音视频格式：mp4
- 支持的目标音视频格式：hls
- 支持处理的视频大小不超过 100 MB
- 目前支持北京，广州，苏州，保定地域使用
- 目前仅支持在bos的post风格上传接口中使用

当前BOS音视频处理仅支持mp4文件向hls文件类型的转换，更多类型的支持、文件码转换等能力，BOS后续将逐步支持。

如您短期内希望采用更广泛的处理能力，同时业务侧也可允许相对长一些的处理延迟，也可参考百度智能云音视频转码MCP、百度智能云音视频点播VOD。

数据处理常见问题

本文主要介绍您在使用 BOS 数据处理时可能遇到的一些常见问题及处理方法。

使用限制

图像处理支持处理哪些格式的图片？

- BOS 图像处理支持主流格式的输入和输出。支持的输入的原图格式包括 jpg、png、bmp、webp、heic、gif；支持的输出目标格式包括 jpg、png、bmp、webp、heic、gif。BOS 提供格式转变，即输出与输入原图格式不同的格式图片。其中，heic 和 webp 为高级压缩格式，会按照调用次数收取一定费用。关于高级压缩定价，请参考 BOS 计费项介绍和 BOS 价格。

图像处理对输入原图有什么限制吗？

- BOS 支持输入的图片大小不超过 20 MB，且图片宽高不超过 4096 px；处理结果输出图片宽高不超过 4096 px，包括管道和子命令间的中间输出图片。

使用方式

图像处理可以以何种方式进行调用？

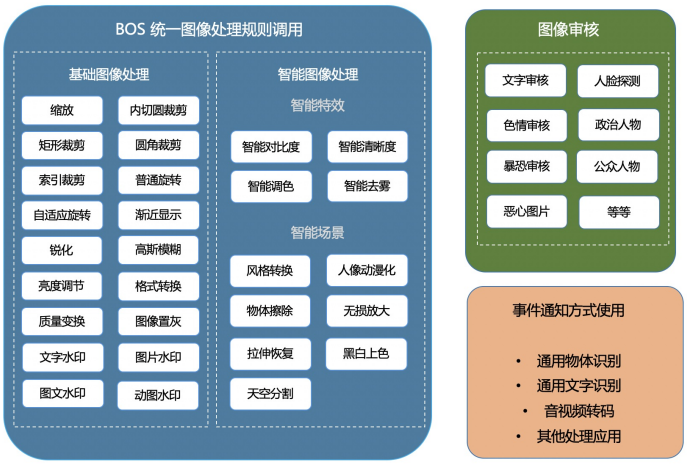
- BOS 支持直接通过 API 接口调用和读取原图，也可以在读写时携带处理命令。若在读文件时携带处理命令，则读取图片为处理后的状态；若在写文件时携带处理命令，则会写入一个新的处理后的图片文件，原图文件依旧保存。同一命令中最多可以指定 4 组处理操作，即 4 个 action，且一个 action 下不可以重复指定同一个参数。为了使用方便，您也可以将一组 action 提前封装为样式进行调用。详细规则请查看图片处理使用规则。

如何创建图像处理样式？

- 您可以通过 BOS 控制台添加或删除样式，单个 Bucket 最多支持创建 400 条图像处理样式。详细操作步骤请查看添加/删除图像样式。

图像处理支持哪些处理内容？

- BOS 图像处理提供丰富的处理方式，整体上包括基础图像处理、智能图像处理、图像审核、OCR识别等能力，但不同能力的使用方式有所不同。基础图像处理和智能图像处理通过 BOS 原生的图片处理使用规则进行使用；图像审核通过单独的界面和接口提供服务，不支持通过 BOS 原生图片处理规则进行使用。若您需要使用通用文字识别或通用物体识别等 OCR 类功能，可以通过事件通知功能触发百度智能云的其他相关产品。



产品价格

图像处理收费吗？

- BOS 基础图像处理、智能图像处理、高级压缩是付费服务，根据调用的图片大小（GB）或调用次数收取费用。图像审核和通过事件通知调用的其他应用，BOS 仅收取 API 的读写费用。关于图像处理的计费，可以查看 BOS 计费项介绍和 BOS 价格。

视频截帧

基本概念

百度智能云BOS提供了业界全新的视频截帧接口，支持通过设置url参数的方法对存储在bos的视频文件进行截取

注意事项

- 使用视频截帧，2022年4月1日正式商业化，计费详情参见BOS价格说明-〉数据处理-〉视频截帧。
- BOS不保存视频截帧的图片，视频截帧的图片需手动下载并保存至本地。

视频截帧命令

视频文件远大于图片文件，处理时间也较长，通常在秒级。

- 操作分类 video
- 操作名称 snapshot

BOS视频处理接口定义如下：

- 参数说明

参数名称	缩写	类型	取值	命令描述	是否必选
time	t	int	视频时长之内	截帧的时间，单位为毫秒	是
format	f	string	[jpg,png,bmp,gif,tif]	输出的图片格式	否
width	w	int	[0,4096]	指定截图宽度	否
height	h	int	[0,4096]	制定截图高度	否
auto-rotate	ar	string	[90,180,270,auto]	指定旋转角度，若设置为auto，则会根据视频文件信息自动旋转	否
mode	m	string	fast	截图类型，若设置为fast，则截取time时间点最近一个关键帧	否

请求示例

- 使用默认模式截取视频7s处的内容，默认情况下输出格式为jpg，输出宽高为原视频宽高

处理的URL为：<原视频URL>?x-bce-process=video/snapshot,t_7000



- 使用关键帧模式截取视频7s处的内容，输出为PNG格式的图片，宽度为800，高度为600,且旋转角度为90度。

处理的URL为：<原视频URL>?x-bce-process=video/snapshot,t_7000,f_png,w_800,h_600,ar_90,m_fast



规则和限制

- 目前仅支持北京、苏州、广州、保定地域使用
- 目前仅支持在页面添加url参数的方式进行截取
- 支持截帧的最大视频大小为10G，支持私有object截取，需在url的参数中增加authorization签名，签名生成可使用IAM工具，链接如下 https://cloud.baidu.com/signature/index.html

文档预览

概述

文档预览功能基于百度智能云文档服务 DOC（Document Service），为开发者提供 Office、WPS 等多种格式的文档存储、管理及在线浏览服务。您无需了解文档存储、转码、分发、在线浏览等技术细节，即可快速搭建安全可靠、高可定制的文档处理平台和应用，助力在线教育、企业网盘等业务的转型升级。

> - 支持多格式
>
> 源文档支持doc、docx、ppt、pptx、xls、xlsx、vsd、pot、pps、rtf、wps、et、dps、pdf、txt、epub多种格式。
>
> - 文件大小限制
>
> 不支持在线预览大于 200 MB的文件。
>
> - 文件页数限制
>
> 不支持在线预览大于 200 页的文件。

计费方式

文档预览服务将于 2022 年 9 月 23 日 0 点开始正式计费，您可以登录控制台-财务查看用量数据。BOS 费用会按小时进行出账，具体价格请参考数据处理费用 。

使用方式

请求参数

操作名称：doc/preview

参数	取值	说明
tt	h5, image	• 设置文档预览转换类型，分为 HTML 和图片，默认为 HTML • h5: 将文档转换为 HTML 格式 • image: 将文档转换为 PNG 格式图片
st	doc、docx、ppt、pptx、xls、xlsx、vsd、pot、pps、rtf、wps、et、dps、pdf、txt、epub	• 指明当前 Object 格式，默认按照拓展名获取
p	1-200	• 指定文档转图片时生成第几页的图片，从 1 开始，最大支持 200 页的文档 • 转图片时此参数必填

请求示例

```
GET https://bj.bcebos.com/bucket/object.pdf?x-bce-process=doc/preview,tt_image,p_1
```

```
GET https://bj.bcebos.com/bucket/object.pdf?x-bce-process=doc/preview,tt_h5
```

```
GET https://bj.bcebos.com/bucket/object?x-bce-process=doc/preview,tt_h5,st_doc
```

请求流程

1. 客户端App或者Web端向业务服务器发起预览请求，并提供要预览的文件名。
2. 业务服务器根据请求文件进行URL签名，将签名完成的URL提供给客户端App或者Web端。
3. 客户端App或者Web端通过签名URL直接预览访问BOS文件。

视频即时转码

基本概念

百度智能云BOS提供了业界全新的视频即时转码接口，支持通过设置url参数的方法即时转换视频的码率，实现h264，h265两种码率的互转。

注意事项

- 使用前需开通百度智能云音视频处理服务（MCP），即时转码能力的计费部分由MCP收取。
- 使用BOS的视频转码时，目前为公测，暂时免费，商用后BOS会按视频即时转码的数量计费。
- BOS不保存转码后的视频，转码后的视频需手动下载并保存至本地。
- 建议的转码视频大小：小于1GB
- 建议的转码分辨率：不超过1080P

视频转码命令

视频文件处理时间较长，通常在秒级。

- 操作分类 video
- 操作名称 codec

BOS视频处理接口定义如下：

- 参数说明

参数名称	缩写	类型	取值	命令描述	是否必选
format	f	string	[h264,h265]	视频转码的目标格式	是
priority	p	int	[0,1]	1：速度优先，质量略差；0：速度落后，质量较高	否

请求示例

- 使用默认模式将h265格式的视频转为h264

处理的URL为：<原视频URL>?x-bce-process=video/codec,f_h264

- 使用高优先级模式将h264码率视频转为h265

处理的URL为：<原视频URL>?x-bce-process=video/codec,f_h265,p_1

规则和限制

- 目前仅支持北京、苏州、广州、保定使用
- x-bce-process仅支持在url参数或header中添加

获取视频元信息

获取视频元信息

接口描述

本接口用于获取储存在BOS上的视频文件的元信息。

权限说明

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。

注意事项

请求结构

```
 GET /<ObjectName>?x-bce-process=video/info HTTP/1.1
Host: <BucketName>.bj.bcebos.com
Authorization: authorization string
```

请求头域

除公共头域外，无其它特殊头域。

请求query参数

参数名称	类型	描述
x-bce-process	string	bos数据处理参数，获取视频元信息为x-bce-process=video/info

响应头域

除公共头域外，无其它特殊头域。

请求示例

```
 GET http://<BucketName>.bj.bcebos.com/<ObjectName>?x-bce-process=video/info
Host: <BucketName>.bj.bcebos.com
X-Bce-Date:2022-11-16T11:54:00Z
Authorization: bce-auth-v1/f81d3b34e48048fbb2634dc7882d7e21/2015-08-11T04:17:29Z/3600/host/74c506f68c65e26c633bfa104c863fffac5190fdec1ec24b7c03eb5d67d2e1de
```

响应示例

```
 HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
Date: Wed, 08 Jul 2015 03:28:11 GMT
x-bce-request-id: d8752367-38e8-45e4-b4c7-e53be3137ce5
Server: BOS
{
    "bucket": "cmdtest2",
    "key": "audio.mp3",
    "fileSizeInByte": 4008612,
    "type": "audio",
    "container": "mp3",
    "durationInSecond": 247,
    "durationInMillisecond": 246890,
    "audio": {
        "codec": "mp3",
        "channels": 2,
        "sampleRateInHz": 44100,
        "bitRateInBps": 128000
    },
    "etag": "575a58dbfec5c808dcd941f927ae1927"
}
```

返回参数

参数名称	描述
bucket	视频所在桶名称
key	视频名称
fileSizeInByte	视频大小
type	视频类型
container	视频格式
durationInSecond	视频时长（秒）
codec	视频格式
channels	视频通道数
sampleRateInHz	视频采样率
etag	文件在BOS上的etag

注意事项

- 当前BOS仅支持北京，广州，苏州，保定集群使用该接口
- 该接口暂时免费公测，后续计费标准待公布

数据湖存储

分层命名空间

服务介绍

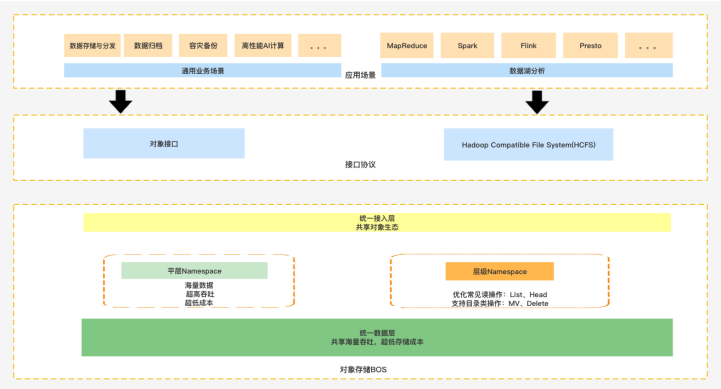
服务概述：

分层命名空间（Namespace）是百度智能云 BOS 对象存储推出的分层元数据管理功能。通过创建并使用分层命名空间的存储桶，可以很好的支持目录级别的 Mv 与 Rename 操作，同时优化常见读操作 List、Head，提升数据处理效率与性能，能很好的满足大数据领域的数据湖计算场景。

注意事项：

- 当前支持华北-北京、华南-广州地域创建并使用分层命名空间的存储桶，其他地域暂不支持。
- 如需将存量数据迁移到分层命名空间的存储桶中，建议使用 Cloudflow 数据迁移复制。

功能架构：



优势：

1. 大数据场景下也可以直接享受对象存储带来的自动化数据管理，存储低成本，数据强安全优势
2. 原生基于对象的分层 Namespace 带来了大数据计算场景下对于文件语义的高性能表现
    - 支持高性能原子 Rename/Delete Dir/Object
    - 远低于平层对象存储的 List/Head 请求时延
3. 直接基于对象存储的大数据分析（All in Object），节约额外的数据复制与流转、存储以及第三方服务费用

4. 无额外的运维，管理及学习成本，充分享受对象存储的生态及工具

应用场景：

大数据分析与计算

大数据分析和大数据计算。对象存储作为云端最佳存储方案，BOS 提供一整套高性能大数据计算解决方案。在大数据场景中，BOS 可以作为您本地自建或者云端 MapReduce 计算集群 、Hive、Spark 离线数仓集群的底层存储系统，原生分层命名空间能力，可以将大数据计算中的大规模批量操作如 Rename、List、Head 等操作的时延降低约 70%，突破传统对象存储支持大数据计算方案的延迟瓶颈。

功能支持说明

功能支持与限制

支持

- 支持文件、目录语义和操作
- 目录原子性、毫秒级 Rename
- 优化读性能
- 支持平层对象存储绝大部分功能

限制

开启分层命名空间的存储桶支持功能与平层命名空间存储桶基本相同，均支持对象级别操作基本均支持比如：上传、下载、复制等；存储桶级别的配置功能如生命周期、事件通知、日志管理等也可正常使用，目前相比起平层命名空间存储桶仅有少部分功能暂不支持，具体说明如下：

Bucket 相关功能

1. 不支持回收站功能。大数据存算分离场景下的回收站能力在 BOS-HDFS 客户端中已支持实现
2. 暂不支持版本控制功能
3. 暂不支持清单功能

Object 相关功能

1. 不支持归档存储类型文件
2. 不支持上传带内容的目录（即上传内容非空但文件名末尾是/的文件）
3. 不支持文件软链接功能
4. 不支持对象标签

接口调整

由于分层命名空间特性导致部分 BOS API 会有部分兼容和语义上的调整，如下所示：

ListObjects

对于 平层命名空间存储桶，分层命名空间下的 ListObjects API 的请求参数有所调整，具体说明参见下表，平层命名空间 ListObjects 详情请查看ListObjects接口说明。

字段	类型	是否必需	描述
delimiter	String	否	分隔符; 分层 Namespace 下 ListObjects 必须指定，且只能为/
marker	String	否	Object 为字母序排列，从 Marker 之后的第一个开始返回，为对象（文件）的全路径；分层命名空间下，将会返回父目录下的子文件或者子目录按照字典序大于 Marker 的结果
maxKeys	Int	否	返回 Object 列表长度最大为 1000，默认值是 1000，如果指定的值大于 1000，按 1000 操作
prefix	String	否	必须为对象的父目录，限定 ListObjects 接口只能返回以 Prefix 为父目录下的对象；Prefix 可以不以/结尾，BOS 会默认为其自动添加目录分隔符

特别说明

> ListObjects 会将父目录下的所有子目录与对象按照字典序排序后返回；
>
> ListObjects 在分层空间存储桶的目录下不再支持列出某个前缀(目录)下的所有文件，只支持根据给出的父目录信息列出子目录和子文件。

使用方式

目前分层命名空间功能支持华北-北京地域、华南-广州地域的新建存储桶（Bucket）开启分层命名空间功能，开启后不支持关闭该功能。控制台开启分层命名空间的具体操作，详见 控制台操作指南-分层命名空间存储桶操作说明。

您可以通过以下方式在开启了分层命名空间的 Bucket 中创建目录、并进行目录的重命名或删除操作。

方式	说明
各语言 SDK	丰富的 Demo
控制台	页面操作直观易用，方便操作
命令行工具（BOS CMD）	简单易用

普通存储桶转换为分层命名空间存储桶

概述

BOS 元数据转换服务支持在未部署任何工具的情况下，直接将 BOS 普通存储桶转换为分层命名空间存储桶。分层命名空间存储桶在支持普通平层桶绝大部分特性和兼容大部分 HDFS 文件系统接口的同时，可以支持目录原子性 Rename 和 Delete，同时优化了 List 和 Head 请求时延，能很好的满足大数据和 AI 领域的数据湖计算场景。

前提条件

1. 目前仅北京、苏州、广州、保定地域支持元数据转换服务和分层命名空间存储桶。
2. 需提交工单申请使用 BOS 元数据转换功能。

使用限制

1. 不支持转换开启回收站或多版本功能的 Bucket，不支持转换存在归档类型 Object、软链接 Symlink、对象标签的 Bucket。
2. 元数据转换成功的平层 Bucket 不再支持关闭分层命名空间，转换过程不可逆，转换前后只是元数据组织形式发生变化，数据不发生移动。
3. 元数据转换任务的完成时间取决于 Bucket 中 Object 数量和后端资源配额，暂不提供精确的时效性保证。

风险说明

- 转换任务开始之后，执行结束之前，尽量不进行增量写入和对存量的覆写及删除，以免遗漏。执行过程中遇到归档类型文件会导致任务失败并中止转换。
- 请确保待转换的 BOS Bucket 中不存在同名文件和文件夹，例如同时存在文件 dir 以及文件夹 dir/，转换过程中会产生冲突，且和任务运行时的导入顺序相关，服务将通过重命名操作来避免冲突，具体参考转换规则。
- 请参考转换规则说明，确保 BOS Bucket 中不存在且不写入使用特殊保留文件名的文件，否则可能出现同名文件相互覆盖，造成文件丢失。
- Bucket 元数据转换开始前可以通过 Replication 等方式先做好备份保存，元数据转换任务完成后请务必再次确认 Bucket 中数据的完整性。

转换规则

- 对于转换前以 / 结尾的文件（例如 a/b/），如果文件是空文件（size 为 0）则转换后直接创建对应的同名目录（例如 a/b），否则其转换后会发生重命名加上特殊后缀 .slash 和 . 转换时间戳，例如文件 a/b/ 会变成 a/b.slash.1716487553，可以通过 ListObjects 查看转换后的对应新文件名。
- 分层命名空间存储桶中不允许同名文件和目录共存，因此如果转换前的存储桶中同时存在文件 a/b 和 a/b/，其中 a/b/ 表示一个目录，那么转换过程中会发生冲突，跟两个文件的转换顺序有相关性，第一原则是优先创建和保留目录。
  - 情况一：创建目录 a/b 时发现同名文件 a/b 已存在，则将文件重命名加上特殊后缀 .file 和 . 转换时间戳，例如文件 a/b/ 会变成a/b.file.1716487553。如果新文件名已经存在，则继续添加次数后缀重试 1 次，例如 a/b.file.1716487553.1。
  - 情况二：创建文件 a/b 时发现同名目录 a/b 已存在，尝试创建加上特殊后缀 .file 和 . 转换时间戳的新文件，例如 a/b.file.1716487553；如果新文件名还是已经存在，则继续添加次数后缀重试 3 次，例如 a/b.file.1716487553.1。

数据迁移和流转

使用 DistCp 迁移数据至分层命名空间 Bucket

Hadoop DistCp 使用

DistCp（分布式拷贝）是 Hadoop 自带的用于大规模集群内部和集群之间拷贝的工具。 它使用 Map/Reduce 实现文件分发，错误处理和恢复，以及报告生成。 它把文件和目录的列表作为 Map 任务的输入，每个任务会完成源列表中部分文件的拷贝。

对于 HDFS 集群和 BOS 间的数据拷贝，可以借助 BOS HDFS 工具，像标准的 Hadoop DistCp 那样。

前置工作

如上，下载 BOS HDFS 相关 jar 包，并做一些必要的配置。

以数据源端是 HDFS 集群，目的端是 BOS 为例，检查源和目的端的读写正常。

```
 $ hadoop fs -get hdfs://host:port/xxx ./
$ hadoop fs -put xxx bos://bucket/xxx
```

开始拷贝

普通拷贝

```
 ##### 从hdfs的src，拷贝到bos指定bucket下的dst路径，默认情况会跳过已经存在的目标文件
$ hadoop distcp hdfs://host:port/src bos://bucket/dst
```

注意：使用 CRC 校验拷贝前后的数据，BOS HDFS 需设置 fs.bos.block.size 和源 HDFS 一致，并开启 fs.bos.crc32c.checksum.enable；仅支持 HDFS 的 dfs.checksum.combine.mode=COMPOSITE_CRC 校验算法。

更新和覆盖

```
 ##### 拷贝，执行覆盖的唯一标准是源文件和目标文件大小是否相同，如果不同，则源文件替换目标文件
$ hadoop distcp -update hdfs://host:port/src bos://bucket/dst

##### 拷贝，并覆盖已经存在的目标文件
$ hadoop distcp -overwrite hdfs://host:port/src bos://bucket/dst
```

拷贝多个源

```
 ##### 指定多个源路径
$ hadoop distcp hdfs://host:port/src1 hdfs://host:port/src2 bos://bucket/dst

##### 从文件里获得多个源
##### srcList的内容是类似hdfs://host:port/src1、hdfs://host:port/src2的多行源路径
$ hadoop distcp hdfs://host:port/srcList bos://bucket/dst
```

注意：多个源的拷贝可能会有冲突，参考标准 Hadoop DistCp 的处理。

更多配置

```
 ##### 指定拷贝数据时map的数目
##### 更多的map数量可能不会提升数据吞吐，反而会带来一些问题，map数量应该根据集群资源和拷贝数据规模综合设定
$ hadoop distcp -m 10 hdfs://host:port/src bos://bucket/dst

##### 忽略失败的map，但会保留失败操作日志
$ hadoop distcp -i hdfs://host:port/src bos://bucket/dst

##### 动态分配任务
##### 默认的分配策略是基于文件大小，在更新拷贝时候，因为有跳过文件，实际拷贝时候有类似"数据倾斜"的问题，耗时最长的map拖慢整体进度
$ hadoop distcp -strategy dynamic -update hdfs://host:port/src bos://bucket/dst
```

更多配置参数，输入 `hadoop distcp` 命令即可获取帮助信息。

DataX 读写 BOS

DataX

DataX 是一个异构数据源离线同步工具，致力于实现包括关系型数据库(MySQL、Oracle等)、HDFS、Hive、ODPS、HBase、FTP等各种异构数据源之间稳定高效的数据同步功能。

配置

1. 下载并解压 DataX；
2. 下载BOS-HDFS 解压缩后，将 jar 包拷贝到 DataX 解压路径plugin/reader/hdfsreader/libs/ 以及 plugin/writer/hdfswriter/libs/ 下；
3. 打开 DataX 解压目录下的 bin/datax.py 脚本，修改脚本中的 CLASS_PATH 变量为如下：

```
CLASS_PATH = ("%s/lib/*:%s/plugin/reader/hdfsreader/libs/*:%s/plugin/writer/hdfswriter/libs/*:.") % (DATAX_HOME, DATAX_HOME, DATAX_HOME)
```

开始

示例

将 {your bucket} 下的 testfile 文件读出并写入到 {your other bucket} 存储桶。

testfile:

```
1 hello
2 bos
3 world
```

bos2bos.json:

```
{
  "job": {
    "setting": {
      "speed": {
        "channel": 1
      },
      "errorLimit": {
        "record": 0,
        "percentage": 0.02
      }
    },
    "content": [{
      "reader": {
        "name": "hdfsreader",
        "parameter": {
          "path": "/testfile",
          "defaultFS": "bos://{your bucket}/",
          "column": [
            {
              "index": 0,
              "type": "long"
            },
            {
              "index": 1,
              "type": "string"
            }
          ],
          "fileType": "text",
          "encoding": "UTF-8",
          "hadoopConfig": {
            "fs.bos.endpoint": "bj.bcebos.com",
            "fs.bos.impl": "org.apache.hadoop.fs.bos.BaiduBosFileSystem",
            "fs.bos.access.key": "{your ak}",
            "fs.bos.secret.access.key": "{your sk}"
          },
          "fieldDelimiter": " "
        }
      },
      "writer": {
        "name": "hdfswriter",
        "parameter": {
          "path": "/testtmp",
          "fileName": "testfile.new",
          "defaultFS": "bos://{your other bucket}/",
          "column": [{
              "name": "col1",
              "type": "string"
            },
            {
              "name": "col2",
              "type": "string"
            }
          ],
          "fileType": "text",
          "encoding": "UTF-8",
          "hadoopConfig": {
            "fs.defaultFS": ""
            "fs.bos.endpoint": "bj.bcebos.com",
            "fs.bos.impl": "org.apache.hadoop.fs.bos.BaiduBosFileSystem",
            "fs.bos.access.key": "{your ak}",
            "fs.bos.secret.access.key": "{your sk}"
          },
          "fieldDelimiter": " ",
          "writeMode": "append"
        }
      }
    }]
  }
}
```

按需替换配置中的 {your bucket}、endpoint、{your sk} 等选项；

支持仅 reader 或 writer 配置为 BOS。

结果

```
python bin/datax.py bos2bos.json
```

执行成功后返回:

```
2023-10-16 14:51:44.907 [job-0] INFO  JobContainer -
任务启动时刻                    : 2023-10-16 14:51:32
任务结束时刻                    : 2023-10-16 14:51:44
任务总计耗时                    :                 12s
任务平均流量                    :               1B/s
记录写入速度                    :              0rec/s
读出记录总数                    :                   3
读写失败总数                    :                   0
```

BOS HDFS 工具

概述

什么是 BOS HDFS 工具

BOS HDFS 工具是百度智能云基于 Hadoop 框架推出的专门用于解决大数据场景下 BOS 中数据的读写和使用问题的便捷工具。

大数据场景下的数据分析已经成为企业关注的核心业务。Hadoop 在分布式数据处理方面具有出色的能力，凭借其可靠、高效、可伸缩、并发处理的特点，已发展为当今最为主流的大数据开源框架之一。Hadoop 实现了一个分布式文件系统（Hadoop Distributed File System），简称为 HDFS。HDFS 具有高容错性的特点，并通过高吞吐量来支持应用程序的数据访问，适合超大数据集的业务场景。HDFS 为海量数据提供了可靠的存储性能，已成为 Hadoop 生态中最重要的一部分。数据量的与日俱增使原生的 Hadoop 面临一些新的问题，HDFS 自建及运维成本非常高，同时本地 HDFS 上海量数据如何存放也是企业面临的巨大挑战。因此，在企业数据上云的趋势下，越来越多的企业选择将数据存储在云端，即存储在对象存储服务当中。但由于对象存储上层数据接口的限制，对象存储中数据和自建 HDFS 之间的访问和读写操作一直是大数据场景下的一个瓶颈，BOS HDFS 很好地解决了这个问题。

BOS HDFS 工具全面兼容 Hadoop 2.7+/3.1+ 系列，支持 HDFS 数据在 BOS 中的海量存储，并在上层数据运算中使用 HDFS 标准接口来对数据进行访问和读写，有效解决自建 HDFS 数据的高运维成本和低可扩展性问题。您可以通过调用该工具充分享受到 BOS 带来的超低价格、超高性能、高可靠和高吞吐的强大优势，满足企业在大数据场景中对数据的读写和使用需求。

BOS HDFS 工具的优势

- 框架兼容：全面兼容 Hadoop 2.7+/3.1+
- 无感调用：实现对 BOS 中数据的无感调用
- 数据存储性价比高：融合对象存储服务 BOS 的超低价格、超高性能、高可靠性、高可用性和高吞吐优势

更新记录

【1.0.6】

- 默认转为virtual-host域名风格（原path-style域名可能触发403错误）

【1.0.5】

- 支持 append/truncate 接口
- 支持多桶隔离配置 ak/sk/endpoint
- 支持 EnvironmentVariableCredentialsProvider
- 优化 getFileStatus/create/delete 接口
- 优化 isFile/isDirectory 接口
- 更新 hadoop-common 依赖到 3.2.2 版本

【1.0.4】

- 支持 CRC32C Checksum 校验
- 优化 create 接口
- 优化 open 接口
- 优化层级 rename 接口
- 默认分块调整 10MB->12MB
- 顺序读策略优化

【1.0.3】

- 支持层级 bucket
- 读 cache 优化，默认开启
- 优化顺序读
- 优化多文件删除
- 已知问题修复

配置和使用

您只需下载相应的 SDK 包，修改部分配置即可使用 BOS HDFS 工具。

下载

- 下载 BOS FS JAR，将解压后的jar包复制到 `$hadoop_dir/share/hadoop/common`中。

使用前准备

- 在 Hadoop 配置路径中修改 `log4j.properties`，调整 BOS SDK 的日志配置：`log4j.logger.com.baidubce.http=WARN`
- 在 `$hadoop_dir/etc/hadoop/core-site.xml` 文件中添加或者修改 BOS HDFS 相关配置。

```
  <property>
  <name>fs.bos.access.key</name>
  <value>{Your AK}</value>
</property>

<property>
  <name>fs.bos.secret.access.key</name>
  <value>{Your SK}</value>
</property>

<property>
  <name>fs.bos.endpoint</name>
  <value>http://bj.bcebos.com</value>
</property>

<property>
  <name>fs.bos.impl</name>
  <value>org.apache.hadoop.fs.bos.BaiduBosFileSystem</value>
</property>

<property>
  <name>fs.AbstractFileSystem.bos.impl</name>
  <value>org.apache.hadoop.fs.bos.BOS</value>
</property>
```

可配置的属性：

名称	描述
fs.bos.access.key	必须，BOS 的访问 accessKey。
fs.bos.secret.access.key	必须，BOS 的访问 secretKey。
fs.bos.endpoint	必须，BOS存储桶所在的endpoint。
fs.bos.session.token.key	可选，临时访问方式，如果配置，fs.bos.access.key 和 fs.bos.secret.access.key 也应该是临时访问密钥。
fs.bos.credentials.provider	可选，credentials 获取方式，可配置 value 值：org.apache.hadoop.fs.bos.credentials.BasicBOSCredentialsProvider、org.apache.hadoop.fs.bos.credentials.EnvironmentVariableCredentialsProvider，默认 org.apache.hadoop.fs.bos.credentials.BasicBOSCredentialsProvider。
fs.bos.bucket.{your bucket}.access.key	可选，指定 {your bucket} 的 BOS 访问 accessKey，访问 {your bucket} 资源优先使用此项。
fs.bos.bucket.{your bucket}.secret.access.key	可选，指定 {your bucket} 的 BOS 访问 secretKey，访问 {your bucket} 资源优先使用此项。
fs.bos.bucket.{your bucket}.session.token.key	可选，指定 {your bucket} 的临时访问 STS 方式，如果配置，fs.bos.bucket.{your bucket}.access.key 和 fs.bos.bucket.{your bucket}.secret.access.key 也应该是临时访问密钥，访问 {your bucket} 资源优先使用此项。
fs.bos.bucket.{your bucket}.endpoint	可选，指定 {your bucket} 的 BOS 存储桶所在的 endpoint，访问 {your bucket} 资源优先使用此项。
fs.bos.block.size	可选，默认128MB， 模拟 HDFS Block Size，在 DistCp 等需要验证 Block 信息的场景使用。
fs.bos.max.connections	可选，客户端支持的最大连接数，默认1000。
fs.bos.multipart.uploads.block.size	可选，客户端使用分块上传时候的分块大小，16777216，单位：byte。
fs.bos.multipart.uploads.concurrent.size	可选，客户端使用分块上传时候并发数，默认10。
fs.bos.object.dir.showtime	可选，list 时候显示目录的修改时间，会额外增加一些交互次数，默认 false。
fs.bos.tail.cache.enable	可选，缓存文件尾部部分数据，可以提升 parquet/orc 等文件的解析性能，默认 true。
fs.bos.crc32c.checksum.enable	可选，开启 crc32c checksum 校验，开启后上传的文件才会计算 CRC32C。
fs.bos.bucket.hierarchy	可选，Bucket 是否是 namespace 类型的，正确设置可节省一次校验开销，错误设置会到导致 Bucket 访问失败。缺省，则自动查询选择。
fs.bos.rename.enable	可选，开启后支持元数据 rename 语义，当前小范围内测阶段，开启前需工单联系是否能使用，否则请求会返回 403 授权失败，默认 false。

如使用 EnvironmentVariableCredentialsProvider，需设置环境变量：

```
 ##### 必须，BOS的访问accessKey
FS_BOS_ACCESS_KEY
##### 必须，BOS的访问secretKey
FS_BOS_SECRET_ACCESS_KEY
##### 可选，临时访问方式
FS_BOS_SESSION_TOKEN_KEY
```

开始使用

使用 BOS HDFS 访问 BOS 服务时路径需要以 `bos://` 开始。如：

```
 $ hdfs dfs -ls bos://{bucket}/
$ hdfs dfs -put ${local_file} bos://{bucket}/a/b/c
```

或者，在 $hadoop_dir/etc/core-site.xml 下配置默认文件目录

```
  <property>
  <name>fs.defaultFS</name>
  <value>bos://{bucket}</value>
</property>
```

> 注意，在配置 fs.defaultFS 为 BosFileSystem 后，如果启动 NameNode 和 DataNode 可能会触发 scheme 检查失败。
>
> 建议在仅使用 BosFileSystem 时配置 fs.defaultFS，且无需启动 NameNode 和 DataNode；否则配置 HDFS 默认地址。

就像使用原生 hdfs 那样：

```
$ hdfs dfs -ls /
```

一次 wordcount 的实践

1. 创建数据目录

用于保存 MapReduce 任务的输入文件

```
$ hdfs dfs -mkdir -p bos://test-bucket/data/wordcount
```

用于保存 MapReduce 任务的输出文件

```
$ hdfs dfs -mkdir bos://test-bucket/output
```

查看这两个新创建的目录

```
$ hdfs dfs -ls bos://test-bucket/
```

```
Found 2 items
drwxrwxrwx   -          0 1970-01-01 08:00 bos://test-bucket/data
drwxrwxrwx   -          0 1970-01-01 08:00 bos://test-bucket/output
```

如果想要显示文件夹的准确创建时间，可以在 `$hadoop_dir/etc/core-site.xml` 下配置

```
<property>
<name>fs.bos.object.dir.showtime</name>
<value>true</value>
</property>
```

2. 写一个单词文件，并上传到 hdfs

单词文件的内容

```
$ cat words.txt
```

```
hello baidu
hello bos
hadoop hdfs
hello hadoop
bos hdfs
```

上传 words.txt 到 hdfs

```
$ hdfs dfs -put words.txt bos://test-bucket/data/wordcount
```

在 hdfs 中查看刚刚上传的文件

```
$ hdfs dfs -cat bos://test-bucket/data/wordcount/words.txt
```

```
hello baidu
hello bos
hadoop hdfs
hello hadoop
bos hdfs
```

3. 运行 wordcount 程序

hadoop自带的wordcount程序在`$hadoop_dir/share/hadoop/mapreduce/`下

```
$ hadoop jar hadoop-mapreduce-examples-2.7.7.jar wordcount bos://test-bucket/data/wordcount bos://test-bucket/output/wordcount
```

4. 查看统计结果

```
$ hdfs dfs -ls bos://test-bucket/output/wordcount/
-rw-rw-rw-   1          0 2020-06-12 16:55 bos://test-bucket/output/wordcount/_SUCCESS
-rw-rw-rw-   1         61 2020-06-12 16:55 bos://test-bucket/output/wordcount/part-r-00000
```

```
$ hdfs dfs -cat bos://test-bucket/output/wordcount/part-r-00000
baidu 1
bos   2
hadoop 2
hdfs 2
hello 3
```

Hadoop DistCp 使用

DistCp（分布式拷贝）是 Hadoop 自带的用于大规模集群内部和集群之间拷贝的工具。 它使用 Map/Reduce 实现文件分发，错误处理和恢复，以及报告生成。 它把文件和目录的列表作为 map 任务的输入，每个任务会完成源列表中部分文件的拷贝。

对于 HDFS 集群和 BOS 间的数据拷贝，可以借助 BOS HDFS 工具，像标准的 Hadoop DistCp 那样。

前置工作

如上，下载 BOS HDFS 相关 jar 包，并做一些必要的配置。

以数据源端是 HDFS 集群，目的端是 BOS 为例，检查源和目的端的读写正常。

```
 $ hadoop fs -get hdfs://host:port/xxx ./
$ hadoop fs -put xxx bos://bucket/xxx
```

开始拷贝

普通拷贝

```
 ##### 从hdfs的src，拷贝到bos指定bucket下的dst路径，默认情况会跳过已经存在的目标文件
$ hadoop distcp hdfs://host:port/src bos://bucket/dst
```

注意：使用 CRC 校验拷贝前后的数据，BOS HDFS 需设置 fs.bos.block.size 和源 HDFS 一致，并开启fs.bos.crc32c.checksum.enable；仅支持 HDFS 的 dfs.checksum.combine.mode=COMPOSITE_CRC 校验算法。

更新和覆盖

```
 ##### 拷贝，执行覆盖的唯一标准是源文件和目标文件大小是否相同，如果不同，则源文件替换目标文件
$ hadoop distcp -update hdfs://host:port/src bos://bucket/dst

##### 拷贝，并覆盖已经存在的目标文件
$ hadoop distcp -overwrite hdfs://host:port/src bos://bucket/dst
```

拷贝多个源

```
 ##### 指定多个源路径
$ hadoop distcp hdfs://host:port/src1 hdfs://host:port/src2 bos://bucket/dst

##### 从文件里获得多个源
##### srcList的内容是类似hdfs://host:port/src1、hdfs://host:port/src2的多行源路径
$ hadoop distcp hdfs://host:port/srcList bos://bucket/dst
```

注意：多个源的拷贝可能会有冲突，参考标准 hadoop distcp 的处理。

更多配置

```
 ##### 指定拷贝数据时map的数目
##### 更多的map数量可能不会提升数据吞吐，反而会带来一些问题，map数量应该根据集群资源和拷贝数据规模综合设定
$ hadoop distcp -m 10 hdfs://host:port/src bos://bucket/dst

##### 忽略失败的map，但会保留失败操作日志
$ hadoop distcp -i hdfs://host:port/src bos://bucket/dst

##### 动态分配任务
##### 默认的分配策略是基于文件大小，在更新拷贝时候，因为有跳过文件，实际拷贝时候有类似"数据倾斜"的问题，耗时最长的map拖慢整体进度
$ hadoop distcp -strategy dynamic -update hdfs://host:port/src bos://bucket/dst
```

更多配置参数，输入 hadoop distcp 命令即可获取帮助信息。

使用进阶

由于自建 Hadoop 集群的可拓展性有限，且需要大量人力对集群进行运维，如果您对性能及安全性有更高要求，推荐使用百度智能云提供的 百度 MapReduce（BMR）。BMR 是全托管的 Hadoop/Spark 集群，您可以按需部署并弹性扩展集群，只需专注于大数据处理、分析、报告，拥有多年大规模分布式计算技术积累的百度运维团队全权负责集群运维，能够在性能，安全性和便捷性上有大幅提升。

常见问题

常见问题

1. 上传报错：Caused by: java.io.IOException: Exception happens during upload:{10001=com.baidubce.BceServiceException: failed to upload part...

问题原因：上传文件体积过大，part数超过10000限制。

解决方法：适当上调fs.bos.multipart.uploads.block.size配置。比如配置16777216，意为分块上传切分后单个part是16MB，最大支持上传文件体积是160GB。

大数据组件使用指南

Flink 使用指南

Flink

flink-bos-hadoop 是百度云对象存储系统 BOS 针对 Flink 的文件系统实现，并且支持了 recoverwriter 接口，Flink 可以基于该文件系统实现读写 BOS 上的数据以及作为流应用的状态后端。

安装

1.Flink 环境准备

以 1.15.0 版本为例。

```
 ##### 下载到一个路径
wget https://archive.apache.org/dist/flink/flink-1.15.0/flink-1.15.0-bin-scala_2.12.tgz

##### 解压缩
tar zxvf flink-1.15.0-bin-scala_2.12.tgz
```

2.添加依赖 jar 包和配置

在启动 Flink 之前，下载 flink-bos-hadoop 插件。

```
##### flink-bos-hadoop插件导入flink下
mkdir ./plugins/bos-fs-hadoop
cp flink-bos-hadoop-1.15.0-0.1.0.jar ./plugins/bos-fs-hadoop/

##### 访问BOS的一些必要配置
vim ./conf/flink-conf.yaml
...
cat ./conf/flink-conf.yaml
fs.bos.impl: org.apache.hadoop.fs.bos.BaiduBosFileSystem
fs.AbstractFileSystem.bos.impl: org.apache.hadoop.fs.bos.BOS
fs.bos.access.key: {your ak}
fs.bos.secret.access.key: {your sk}
fs.bos.endpoint: bj.bcebos.com {your bucket endpoint}
```

使用

启动

```
./bin/start-cluster.sh
```

提交作业

```
##### 通过以下格式指定路径，BOS 对象可类似于普通文件使用：bos://<your-bucket>/{object-name}
./bin/flink run examples/streaming/WordCount.jar --input "bos://my_bucket/students.txt" --output "bos://my_bucket/out"
```

查看运行结果

```
##### 查看wordcount统计结果
$ hadoop fs -ls bos://my_bucket/out
Found 1 items
drwxrwxrwx   -          0 1970-01-01 08:00 bos://my_bucket/out/2023-08-10--15

$ hadoop fs -ls bos://my_bucket/out/2023-08-10--15/
Found 1 items
-rw-rw-rw-   1       1792 2023-08-10 15:52 bos://my_bucket/out/2023-08-10--15/part-3053774f-2d8e-40c5-aa3c-01402ce4b6b4-0

$ hadoop fs -cat bos://my_bucket/out/2023-08-10--15/part-3053774f-2d8e-40c5-aa3c-01402ce4b6b4-0
(name,1)
(studentname,1)
(age,1)
...
```

Hive 使用指南

Hive

Hive 是基于 Hadoop 的一个数据仓库工具，用来进行数据提取、转化、加载，这是一种可以存储、查询和分析存储在 Hadoop 中的大规模数据的机制。Hive 数据仓库工具能将结构化的数据文件映射为一张数据库表，并提供 SQL 查询功能，能将 SQL 语句转变成 MapReduce 任务来执行。

前提条件

首先参考 BOS HDFS 一文安装并配置 BOS HDFS，本机安装的 Hadoop 版本为 hadoop-3.3.2，参考文中"开始使用"一节完成 BOS HDFS 的基本试用，并设置环境变量：

```
export HADOOP_HOME=/opt/hadoop-3.3.2
export HADOOP_CLASSPATH=`$HADOOP_HOME/bin/hadoop classpath
```

安装 mysql

mysql 用来存储 hive 的元数据，可以选择本地安装，也可以直接远程连接已安装的 mysql 或者 RDS。本机安装的版本是：mysql-5.1.61-4.el6.x86_64 安装完成之后，可用 service mysqld status 查看运行状态，并使用

```
/usr/bin/mysqladmin -u root -h ${IP} password ${new-password} #设置新密码
```

可以在 mysql 中创建 hive 专用的用户，并设置密码。

安装 hive

本机安装的版本为 2.3.9。 修改 conf 文件夹下的两个配置：

```
mv hive-env.sh.template hive-env.sh
mv hive-site.xml.template hive-site.xml
```

在 hive-env.sh 中添加：

```
export HIVE_CONF_DIR=/ssd2/apache-hive-2.3.9-bin/conf
```

在 hive-site.xml 中添加：

```
  <property>
    <name>javax.jdo.option.ConnectionURL</name>
    <value>jdbc:mysql://localhost:3306/hive?createDatabaseIfNotExist=true</value>
    <description>MySQL</description>
</property>
<property>
    <name>javax.jdo.option.ConnectionDriverName</name>
    <value>com.mysql.jdbc.Driver</value>
    <description>JDBC</description>
</property>
<property>
    <name>javax.jdo.option.ConnectionUserName</name>
    <value>root</value>
    <description>username</description>
    </property>
<property>
    <name>javax.jdo.option.ConnectionPassword</name>
    <value>new-password</value>
    <description>password</description>
</property>
```

配置中 javax.jdo.option.ConnectionURL 是指 mysql server 的连接地址，javax.jdo.option.ConnectionUserName 就是用于 hive 的 mysql 用户名，javax.jdo.option.ConnectionPassword 是用户名对应的密码。设置完成后，把 mysql 的 JBDC 驱动复制到 lib 文件夹下，本机采用的驱动为 mysql-connector-java-5.1.32-bin.jar.

初始化 msyql

```
./bin/schematool -dbType mysql -initSchema
```

启动 hive

```
./bin/hive
```

hive 测试

创建表

```
 create database hive;     // 创建数据库
create table hive_test (a int, b string) //创建表
ROW FORMAT DELIMITED FIELDS TERMINATED BY ',';
```

1. 新建一个 shell 脚本，名为 gen_data.sh

```
 ##### !/bin/bash
MAXROW=1000000 #指定生成数据行数
for((i = 0; i < $MAXROW; i++))
do
   echo $RANDOM, \"$RANDOM\"
done
```

2.运行脚本，生成测试数据

```
./gen_data.sh > hive_test.data
```

3.把数据加载到表中

```
load data inpath "bos://${bucket_name}/hive_test.data" into table hive.hive_test;
```

4.查询

```
 hive> select count(*) from hive_test;
WARNING: Hive-on-MR is deprecated in Hive 2 and may not be available in the future versions. Consider using a different execution engine (i.e. spark, tez) or using Hive 1.X releases.
Query ID = root_20230528173013_6f5296db-562e-4342-917f-bcf14fc1480d
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks determined at compile time: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Job running in-process (local Hadoop)
2023-05-28 17:30:16,548 Stage-1 map = 0%,  reduce = 0%
2023-05-28 17:30:18,558 Stage-1 map = 100%,  reduce = 100%
Ended Job = job_local238749048_0001
MapReduce Jobs Launched:
Stage-Stage-1:  HDFS Read: 0 HDFS Write: 0 SUCCESS
Total MapReduce CPU Time Spent: 0 msec
OK
1000000
hive> select * from hive_test limit 10;
OK
11027   "11345"
10227   "24281"
32535   "16409"
24286   "24435"
2498    "10969"
16662   "16163"
5345    "26005"
21407   "5365"
30608   "4588"
19686   "11831"
```

Spark 使用指南

Spark

Apache Spark 是专为大规模数据处理而设计的快速通用的计算引擎。Spark 是 UC Berkeley AMP Lab (加州大学伯克利分校的AMP实验室)所开源的类 Hadoop MapReduce 的通用并行框架，拥有 Hadoop MapReduce 所具有的优点；但不同于 MapReduce 的是 --Job 中间输出结果可以保存在内存中，从而不再需要读写 HDFS，因此 Spark 能更好地适用于数据挖掘与机器学习等需要迭代的 MapReduce 的算法。

安装

1.Spark 环境准备

```
 ##### 下载到一个路径
wget https://archive.apache.org/dist/spark/spark-3.1.1/spark-3.1.1-bin-hadoop3.2.tgz

##### 解压
tar -zxvf spark-3.1.1-bin-hadoop3.2.tgz

##### 设置环境变量
SPARK_HOME=/home/hadoop/spark-3.1.1-bin-hadoop3.2
PATH=$PATH:$HADOOP_HOME/bin:$SPARK_HOME/bin
```

2.添加依赖 jar 包

下载 BOS-HDFS

```
 ##### 解压，拷贝到spark依赖路径
unzip bos-hdfs-sdk-1.0.3-community.jar.zip
cp bos-hdfs-sdk-1.0.3-community.jar {SPARK_HOME}/jars

##### 访问BOS的一些必要配置
cp {SPARK_HOME}/conf/spark-defaults.conf.template {SPARK_HOME}/conf/spark-defaults.conf
vim {SPARK_HOME}/conf/spark-defaults.conf
...
cat {SPARK_HOME}/spark-defaults.conf
spark.hadoop.fs.bos.access.key={your ak}
spark.hadoop.fs.bos.secret.access.key={your sk}
spark.hadoop.fs.bos.endpoint=http://bj.bcebos.com {your bucket endpiont}
spark.hadoop.fs.AbstractFileSystem.bos.impl=org.apache.hadoop.fs.bos.BOS
spark.hadoop.fs.bos.impl=org.apache.hadoop.fs.bos.BaiduBosFileSystem
```

使用

编写 demo.py,读取 my-bucket 中 student.parquet，统计 age > 22 的 student。

```
 from pyspark.sql import SparkSession

spark = SparkSession.builder.appName("example_app")\
    .config("spark.driver.bindAddress", "localhost")\
    .getOrCreate()
bosFile = "bos://my-bucket/student"

##### 写入
data = [("abc",22), ("def",17), ("ghi",34)]
df = spark.createDataFrame(data, ["name", "age"])
df.write.parquet(bosFile)

df = spark.read.parquet(bosFile)
df.printSchema()

df.createOrReplaceTempView("students")
sqlDF = spark.sql("SELECT * FROM students WHERE age > 22 LIMIT 10")
sqlDF.show()
```

运行

```
spark-submit demo.py
```

Presto 使用指南

概览

Presto 是 Facebook 开发的数据查询引擎，可对海量数据进行快速地交互式分析，支持 Hive，关系数据库等多种数据源。由于 BOS 在超低价格、超高性能、高可靠和高吞吐的强大存储优势，越来越多企业选择 BOS 作为大数据的存储媒介。因此，本文将对 Presto 在 BOS 上的使用方法作一个简要的介绍。

前提条件

参考 Hive使用指南 一文安装并配置 Hive

安装配置

安装版本为349，可参考 presto部署 一文的过程。其中，如果本机也作为 worker，在 config.properties 中可设置：

```
node-scheduler.include-coordinator=true #需要改为true
```

在 etc/catalog/hive.properties 中，配置为：

```
connector.name=hive-hadoop2
hive.config.resources=/ssd2/hadoop-3.3.2/etc/hadoop/core-site.xml,/ssd2/hadoop-3.3.2/etc/hadoop/hdfs-site.xml #这里的地址一定要正确
hive.metastore.uri=thrift://127.0.0.1:9083
hive.allow-drop-table=false
hive.storage-format=ORC
hive.metastore-cache-ttl=1s
hive.metastore-refresh-interval=1s
hive.metastore-timeout=35m
hive.max-partitions-per-writers=1000
hive.cache.enabled=true
hive.cache.location=/opt/hive-cache
```

把 bos filesystem 的 jar 包复制到 plugin/hive-hadoop2/ 下，之后运行：

```
./bin/launcher start
```

启动 presto-server，

```
./presto-cli --server localhost:8881 --catalog hive --schema default
```

运行

```
presto:default>use hive;
USE
presto:hive>select * from hive_test limit 10;
   a   |    b
-------+----------
 11027 | "11345"
 10227 | "24281"
 32535 | "16409"
 24286 | "24435"
  2498 | "10969"
 16662 | "16163"
  5345 | "26005"
 21407 | "5365"
 30608 | "4588"
 19686 | "11831"
(10 rows)

Query 20230601_084130_00004_dzvjb, FINISHED, 1 node
Splits: 18 total, 18 done (100.00%)
[Latency: client-side: 0:02, server-side: 0:02] [59.4K rows, 831KB] [28.9K rows/s, 404KB/s]
```

在以上示例中，我们就通过 presto 查询到了存储在 bos 中的数据。

基于 S3 的 presto 访问

presto 访问存储在BOS中的数据只能是通过 hive，但是 hive 访问 BOS 中的数据有两种方式，第一种就是通过上述的介绍，基于 bos-hdfs；第二种就是直接通过 S3 协议访问 BOS。

hive 配置

安装 metastore：

```
wget "https://repo1.maven.org/maven2/org/apache/hive/hive-standalone-metastore/3.1.2/hive-standalone-metastore-3.1.2-bin.tar.gz"
tar -zxvf hive-standalone-metastore-3.1.2-bin.tar.gz
sudo mv apache-hive-metastore-3.1.2-bin /usr/local/metastore
sudo chown user:user /usr/local/metastore
```

下载并使用 hive-standalone-metastore，增加必需的 jar 包：

```
rm /usr/local/metastore/lib/guava-19.0.jar
cp /usr/local/hadoop/share/hadoop/common/lib/guava-27.0-jre.jar \
  /usr/local/metastore/lib/
cp /usr/local/hadoop/share/hadoop/tools/lib/hadoop-aws-3.2.1.jar \
  /usr/local/metastore/lib/
cp /usr/local/hadoop/share/hadoop/tools/lib/aws-java-sdk-bundle-1.11.375.jar \
  /usr/local/metastore/lib/
```

之后配置 /usr/local/metastore/conf/metastore-site.xml

```
  <property>
    <name>javax.jdo.option.ConnectionURL</name>
    <value>jdbc:mysql://localhost/metastore?createDatabaseIfNotExist=true</value>
  </property>
  <property>
    <name>javax.jdo.option.ConnectionDriverName</name>
    <value>com.mysql.jdbc.Driver</value>
  </property>
  <property>
    <name>javax.jdo.option.ConnectionUserName</name>
    <value>hive</value>
  </property>
  <property>
    <name>javax.jdo.option.ConnectionPassword</name>
    <value>hive</value>
  </property>
  <property>
    <name>hive.metastore.event.db.notification.api.auth</name>
    <value>false</value>
  </property>
  <property>
    <name>fs.s3a.access.key</name>
    <value>S3_ACCESS_KEY</value>
  </property>
  <property>
    <name>fs.s3a.secret.key</name>
    <value>S3_SECRET_KEY</value>
  </property>
  <property>
    <name>fs.s3a.connection.ssl.enabled</name>
    <value>false</value>
  </property>
  <property>
    <name>fs.s3a.path.style.access</name>
    <value>true</value>
  </property>
  <property>
    <name>fs.s3a.endpoint</name>
    <value>S3_ENDPOINT</value>
  </property>
```

fs.s3a.endpoint 是指 s3 的 endponit，bos 的 s3 endpoint在 BOS S3域名 可查.

启动 hive metastore:

```
  /usr/local/metastore/bin/start-metastore &
```

presto 配置

preto 的版本这里不作改变，hive.properties 的内容改为：

```
  connector.name=hive-hadoop2
hive.metastore.uri=thrift://localhost:9083
hive.s3.path-style-access=true
hive.s3.endpoint=S3_ENDPOINT
hive.s3.aws-access-key=S3_ACCESS_KEY
hive.s3.aws-secret-key=S3_SECRET_KEY
hive.s3.ssl.enabled=false
```

启动 presto

```
  /usr/local/trino/bin/launcher start
```

创建 schema

```
  CREATE SCHEMA IF NOT EXISTS hive.iris
WITH (location = 's3a://my-bos-bucket/'); //注意，这里的location就是以s3a开头，如果bucket为my-bos-bucket，则location应为s3a://my-bos-bucket/开头
##### 创建表
CREATE TABLE IF NOT EXISTS hive.iris.iris_parquet (
  sepal_length DOUBLE,
  sepal_width  DOUBLE,
  petal_length DOUBLE,
  petal_width  DOUBLE,
  class        VARCHAR
)
WITH (
  external_location = 's3a://my-bos-bucket/iris_parquet',
  format = 'PARQUET'
);

SELECT
  sepal_length,
  class
FROM hive.iris.iris_parquet
LIMIT 10;
```

Flume 数据存储到 BOS

Flume

Flume 是一个分布式、可靠性和高可用的海量日志聚合系统，支持在系统中定制各类数据发送方，用于收集数据；同时，Flume 提供对数据进行简单处理，并写到各种数据接收方（可定制）的能力。

Flume 支持多种 Sink 类型，可以借助 HDFS Sink 将收集到的数据存储到 BOS。

开始

1. 下载并安装 apache-flume

略

2. 配置环境

如果已有 hadoop 环境，且已配置过访问 BOS，本环节跳过； 否则

- 将 bos-hdfs 的 jar 包下载到 /opt/apache-flume-1.xx.0-bin/lib 目录下；
- 将 hadoop 下的配置文件 core-site.xml 添加访问 BOS 的相关配置，并复制到 /opt/apache-flume-1.xx.0-bin/conf 目录下。

3. 创建 flume 配置文件

把 flume 的 StressSource 作为 source，使用内存 channel，通过 HDFS 协议写入 BOS。

```
 ##### ss2bos.properties
agent.sources = stress_source
agent.channels = mem_channel
agent.sinks = bos_hdfs_sink

agent.sources.stress_source.type = org.apache.flume.source.StressSource
agent.sources.stress_source.channels = mem_channel
agent.sources.stress_source.size = 1024
agent.sources.stress_source.maxTotalEvents = 1000
agent.sources.stress_source.maxEventsPerSecond = 10
agent.sources.stress_source.batchSize=10

agent.channels.mem_channel.type = memory
agent.channels.mem_channel.capacity = 1000000
agent.channels.mem_channel.transactionCapacity = 100

agent.sinks.bos_hdfs_sink.channel = mem_channel
agent.sinks.bos_hdfs_sink.type = hdfs
agent.sinks.bos_hdfs_sink.hdfs.useLocalTimeStamp = true
agent.sinks.bos_hdfs_sink.hdfs.filePrefix = %{host}_bos_hdfs_sink #host区分文件，避免并发写冲突
agent.sinks.bos_hdfs_sink.hdfs.path = bos://{your bucket}/flume/%Y-%m-%d-%H-%M #替换bucket路径
agent.sinks.bos_hdfs_sink.hdfs.fileType = DataStream
agent.sinks.bos_hdfs_sink.hdfs.writeFormat = Text
agent.sinks.bos_hdfs_sink.hdfs.rollSize = 0
agent.sinks.bos_hdfs_sink.hdfs.rollCount = 100
agent.sinks.bos_hdfs_sink.hdfs.rollInterval = 0
agent.sinks.bos_hdfs_sink.hdfs.batchSize = 100
agent.sinks.bos_hdfs_sink.hdfs.round = true
agent.sinks.bos_hdfs_sink.hdfs.roundValue = 10
agent.sinks.bos_hdfs_sink.hdfs.roundUnit = minute
```

4. 启动 Flume agent

```
./bin/flume-ng agent -n agent -c conf/ -f ss2bos.properties
```

Impala 使用指南

Impala

Impala 是用于处理存储在 Hadoop 集群中的大量数据的 MPP（大规模并行处理）SQL 查询引擎。 它是一个用 C ++ 和 Java 编写的开源软件。 与其他 Hadoop 的 SQL 引擎相比，它提供了高性能和低延迟。

安装步骤

安装 metastore

参考 Presto使用指南 一文中"基于 S3 的 presto 访问"一节安装并配置 metastore

安装 impala

1. 下载 rpm 包，地址http://archive.cloudera.com/cdh5/repo-as-tarball/5.14.0/cdh5.14.0-centos6.tar.gz 使用 `tar -zxvf cdh5.14.0-centos6.tar.gz` 解压之后，cd cdh/5.14.0，创建本地 server，运行

```
python -m SimpleHTTPServer 8092 &
```

2. 配置本地 yum 源

```
 vim /etc/yum.repos.d/localimp.repo
[localimp]
name=localimp
baseurl=http://127.0.0.1:8092/
gpgcheck=0
enabled=1
```

3. 使用如下命令安装

```
yum install -y impala impala-server impala-state-store impala-catalog impala-shell
```

4. 将 hive 的配置文件(metastore-site.xml)复制到 impala 的配置路径下：

```
 ##### 把配置好的conf复制到/etc/impala/conf/路径下
cp metastore/conf/metastore-site.xml  /etc/impala/conf/hive-site.xml
```

5. 增加 s3 配置 vim /etc/impala/conf/core-site.xml，参考 impala-s3 配置

```
 <configuration>
<property>
    <name>fs.s3a.block.size</name>
    <value>134217728 </value>
</property>
<property>
    <name>fs.azure.user.agent.prefix</name>
    <value>User-Agent: APN/1.0 Hortonworks/1.0 HDP/None</value>
</property>
<property>
    <name>fs.s3a.connection.maximum</name>
    <value>1500</value>
</property>
<property>
    <name>fs.defaultFS</name>
      <value>s3a://${bucket}</value>
      </property>
<property>
    <name>fs.s3a.endpoint</name>
      <value>s3.bj.bcebos.com</value>
        <description>endpoint</description>
        </property>
<property>
    <name>fs.s3a.access.key</name>
      <value>${AK}</value>
        <description>AK</description>
        </property>
<property>
    <name>fs.s3a.secret.key</name>
      <value>${SK}</value>
        <description>SK</description>
        </property>
</configuration>
```

6. 修改 bigtop 的配置，设置 JAVA_HOME，并确保 impala 用户也具有访问权限。 修改 bigtop 的 java_home 路径（3台机器）

```
 vim /etc/default/bigtop-utils
export JAVA_HOME=/export/servers/jdk1.8.0_65
```

7. 设置 mysql 驱动的软链接：

```
 ln -s mysql-connector-java-5.1.32.jar /usr/share/java/mysql-connector-java.jar
```

8. 启动 impala

```
 service impala-state-store start
service impala-catalog start
service impala-server start
```

启动后可在 /var/log/impala 文件夹下查看日志 运行 impala-shell 命令：

```
[root@my-node impala]# impala-shell
Starting Impala Shell without Kerberos authentication
Connected to my-node:21000
Server version: impalad version 2.11.0-cdh5.14.0 RELEASE (build d68206561bce6b26762d62c01a78e6cd27aa7690)
***********************************************************************************
Welcome to the Impala shell.
(Impala Shell v2.11.0-cdh5.14.0 (d682065) built on Sat Jan  6 13:27:16 PST 2018)

When pretty-printing is disabled, you can use the '--output_delimiter' flag to set
the delimiter for fields in the same row. The default is ','.
***********************************************************************************
[my-node:21000] > show databases;
Query: show databases
+------------------+----------------------------------------------+
| name             | comment                                      |
+------------------+----------------------------------------------+
| _impala_builtins | System database for Impala builtin functions |
| default          | Default Hive database                        |
+------------------+----------------------------------------------+
Fetched 2 row(s) in 0.16s
[my-node:21000] > CREATE DATABASE db_on_s3 LOCATION 's3a://my-bigdata/impala/s3';
Query: create DATABASE db_on_s3 LOCATION 's3a://my-bigdata/impala/s3'
WARNINGS: Path 's3a://my-bigdata/impala' cannot be reached: Path does not exist.

Fetched 0 row(s) in 2.51s
[my-node:21000] > show databases;
Query: show databases
+------------------+----------------------------------------------+
| name             | comment                                      |
+------------------+----------------------------------------------+
| _impala_builtins | System database for Impala builtin functions |
| db_on_s3         |                                              |
| default          | Default Hive database                        |
+------------------+----------------------------------------------+
Fetched 3 row(s) in 0.01s
[my-node:21000] > use db_on_s3;
Query: use db_on_s3
[my-node:21000] > create table hive_test (a int, b string) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',';
Query: create table hive_test (a int, b string) ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
Fetched 0 row(s) in 2.11s
[my-node:21000] > insert into hive_test(a, b) values(1,'tom');
Query: insert into hive_test(a, b) values(1,'tom')
Query submitted at: 2023-09-13 19:20:26 (Coordinator: http://my-node:25000)
Query progress can be monitored at: http://my-node:25000/query_plan?query_id=ec4463f20d37dfe4:5192e94f00000000
Modified 1 row(s) in 7.57s
[my-node:21000] > insert into hive_test(a, b) values(2,'jerry');
Query: insert into hive_test(a, b) values(2,'jerry')
Query submitted at: 2023-09-13 19:20:42 (Coordinator: http://my-node:25000)
Query progress can be monitored at: http://my-node:25000/query_plan?query_id=694061adf492a154:4a24912d00000000
Modified 1 row(s) in 1.02s
```

在对应路径下可看见新生成的文件：



Logstash数据存储到BOS

工具概述

Logstash是一个开源的数据收集、处理、分析工具，可以从众多来源捕捉事件，流式传输到所需的存储库中，本文将描述Logstash如何把数据存储到BOS bucket。

配置教程

1. 官网下载安装Logstash安装包，选择对应操作系统的最新版本即可；下载后解压安装包并创建配置文件。

```
wget https://artifacts.elastic.co/downloads/logstash/logstash-8.15.0-linux-x86_64.tar.gz
tar zxvf logstash-oss-8.15.0-darwin-aarch64.tar.gz
cd logstash-8.15.0
touch file2bos.conf
```

2. 修改配置文件file2bos.conf，以本地文件为输入源，BOS为输出源，详细配置参考Plugin

```
input {
  file {
    path => "/tmp/access.log"
    start_position => "beginning"
  }
}
output {
  s3 {
    endpoint => "bos endpoint"    # 例如https://s3.bj.bcebos.com
    access_key_id => "bos ak"
    secret_access_key => "bos sk"
    bucket => "bos-bkt"       # 必填项
    size_file => 1024
    time_file => 1
    prefix => "logstash/"
  }
}
```

3. 参数说明

参数名称	参数含义
path	输入源文件所在绝对路径
start_position	决定了从文件的何处开始读取数据，beginning表示从文件的开头开始读取数据
endpoint	BOS Endpoint，参考文档
access_key_id	BOS Access Key ID，参考如何获取AKSK
secret_access_key	BOS Secret Access Key
bucket	BOS桶名称
size_file	设置文件大小，当字节数超过 size_file 值时，将创建一个新的文件。
time_file	设置文件滚动周期（单位为分钟）。当数据写入时间超过周期时，会生成一个新的文件。
prefix	指定文件存储的目录，例如"logstash/"，文件会写入到bucket的logstash/目录下

4. 运行程序：`bin/logstash -f file2bos.conf`
5. 结果验证：可以看到BOS bucket中指定目录下存在新写入的object，名字格式：`ls.s3.ffe6de57-539a-4953-adb1-2807d604d1ec.2023-08-23T14.19.part0.txt`

Elasticsearch快照数据存储到BOS

工具概述

Elasticsearch是一个分布式搜索和数据分析引擎，能够高效地存储、检索、分析、处理庞大的数据集。它支持将快照数据备份到S3等对象存储系统上，本文将详细阐述如何利用ES的S3 Repository插件将快照安全地备份至BOS远程存储。

配置教程

1. 下载Elasticsearch安装包，解压并启动ES。

```
wget https://artifacts.elastic.co/downloads/elasticsearch/elasticsearch-8.15.0-linux-x86_64.tar.gz
tar zxvf elasticsearch-8.15.0-linux-x86_64.tar.gz
cd elasticsearch-8.15.0
bin/elasticsearch
```

2. 最新版本的Elasticsearch已经集成了S3 Repository插件，可以直接使用，启动ES时会看到log显示已加载s3 plugin，如下所示：

```
.....
[2024-08-26T17:36:49,356][INFO ][o.e.p.PluginsService     ] [instance-oaxggkck] loaded module [repository-s3]
[2024-08-26T17:36:49,365][INFO ][o.e.p.PluginsService     ] [instance-oaxggkck] loaded module [repository-azure]
[2024-08-26T17:36:49,365][INFO ][o.e.p.PluginsService     ] [instance-oaxggkck] loaded module [repository-gcs]
.....
```

> 部分老版本ES可以自行安装该插件，请参考官网教程

3. 将BOS使用的Access Key ID和Secret Access Key配置到ES，执行如下命令：

```
bin/elasticsearch-keystore add s3.client.default.access_key
bin/elasticsearch-keystore add s3.client.default.secret_key
```

备份步骤

1. 首先需要创建一个具有创建快照仓库权限的角色用户，例如superuser，执行如下命令：

```
bin/elasticsearch-users useradd my_super_user -p my_pwd -r superuser
```

2. 创建一个名为test的快照仓库，执行如下命令后如果成功会打印`{"acknowledged":true}`：

```
curl -u my_super_user:my_pwd -k -XPUT 'https://localhost:9200/_snapshot/test' -H 'Content-Type: application/json' -d '{ "type": "s3", "settings": { "bucket": "bos-bkt", "endpoint": "s3.bj.bcebos.com"} }'
```

> 访问ES默认需使用https协议；-k参数忽略SSL证书验证（在开发环境中可以使用，在生产环境中不推荐）；BOS endpoint参考官网

3. 使用如下命令来确认仓库创建是否成功：

```
curl -u my_super_user:my_pwd -k -XGET 'https://localhost:9200/_snapshot/test?pretty'
{
  "test" : {
    "type" : "s3",
    "settings" : {
      "bucket" : "bos-bkt",
      "endpoint" : "s3.bj.bcebos.com"
    }
  }
}
```

4. 创建test_index索引，执行如下命令：

```
curl -u my_super_user:my_pwd -k -XPUT "https://localhost:9200/test_index" -H 'Content-Type: application/json' -d'
{
  "settings": {
    "number_of_shards": 1,
    "number_of_replicas": 0
  }
}
'
```

5. 查看目前ES集群的索引信息，执行如下命令：

```
curl -u my_super_user:my_pwd -k -XGET 'https://localhost:9200/_cat/indices?v'
# 输出结果：
health status index      uuid                   pri rep docs.count docs.deleted store.size pri.store.size dataset.size
green  open   test_index kwVZOtQ-RoWvuNxpgeIZ2w   1   0          0            0       227b           227b         227b
```

6. 备份test_index索引到BOS，执行如下命令：

```
curl -u my_super_user:my_pwd -k -XPUT 'https://localhost:9200/_snapshot/test/test_index' -H 'Content-Type: application/json' -d '{ "indices": "test_index" }'
```

7. 然后我们可以从BOS控制台看到备份的结果：



8. 接着就可以利用备份到BOS的快照来恢复索引，先close索引，再从已经备份到BOS的快照restore，执行如下命令：

```
curl -u my_super_user:my_pwd -k -XPOST 'https://localhost:9200/test_index/_close'
# 输出：{"acknowledged":true,"shards_acknowledged":true,"indices":{"test_index":{"closed":true}}}
curl -u my_super_user:my_pwd -k -XPOST 'https://localhost:9200/_snapshot/test/test_index/_restore?pretty'
# 输出：{"accepted" : true}
```

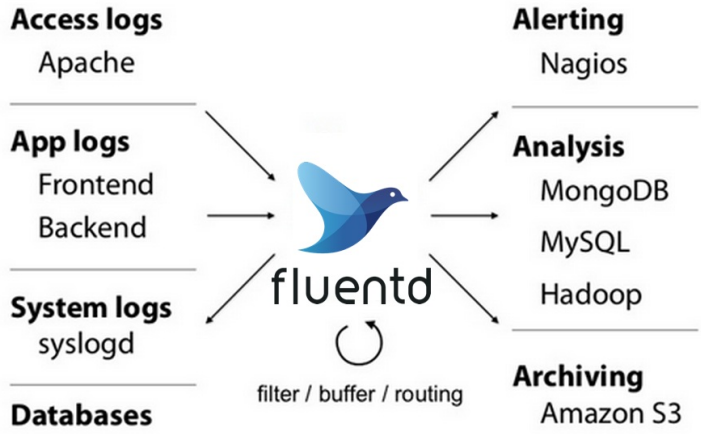
9. 再次查看目前ES集群的索引信息，会发现和备份时的状态一致：

```
curl -u my_super_user:my_pwd -k -XGET 'https://localhost:9200/_cat/indices?v'
```

Fluentd收集数据存储到BOS

工具概述

Fluentd是一个开源数据收集器，可以从各种数据源收集事件，统一收集数据后可以写入文件、RDBMS、NoSQL、Hadoop、S3等存储系统消费，以便更好地使用数据。本文将详细阐述如何利用Fluentd的fluent-plugin-s3插件将日志数据输出到BOS存储桶。



配置教程

1. 前置条件：Ruby 2.7以上版本，可以基于rvm管理多个ruby版本。
2. 基于gem安装Fluentd和fluent-plugin-s3插件，gems默认源可能存在不可用问题，需要通过`gem source`替换成可用镜像源。

```
gem install fluentd
gem install fluent-plugin-s3 --no-document
```

3. Fluentd的输入输出方式主要基于`/etc/fluent/fluent.conf`配置文件，首先修改配置文件中的input source标签，方便起见我们直接使用内置的HTTP input做演示，如下所示：

```
# HTTP input
# http://localhost:8888/<tag>?json=<json>
<source>
  @type http
  @id http_input
  port 8888
</source>
```

4. 修改配置文件中的output match标签，使用BOS作为输出存储，详见Configuration: Output，执行如下命令：

```
<match bos.*.*> # 匹配bos前缀的tag的所有日志
@type s3

aws_key_id bos-ak
aws_sec_key bos-sk
s3_bucket bos-bkt
s3_endpoint bos-endpoint # 例如https://s3.bj.bcebos.com

path logs/${tag}/%Y/%m/%d/
s3_object_key_format %{path}%{time_slice}_%{index}.%{file_extension}
<buffer tag,time>
  @type file
  path /var/log/fluent/bos
  timekey 60 # 1 min partition
  timekey_wait 10s
  timekey_use_utc true # use utc
</buffer>
<format>
  @type json
</format>
</match>
```

5. 重新启动Fluentd进程，执行如下命令：

```
fluentd -c /etc/fluent/fluent.conf
```

6. 构建input日志，测试能否正常写入BOS，执行如下命令：

```
curl -X POST -d 'json={"message":"hello world!"}' http://localhost:8888/bos.tag.0
curl -X POST -d 'json={"message":"hello world!"}' http://localhost:8888/bos.tag.1
```

7. 从BOS控制台可以查看到日志导入结果：

```
bos-bkt/logs/bos.tag.0/2024/08/27/20240827080526_0.gz
bos-bkt/logs/bos.tag.1/2024/08/27/20240827080531_0.gz
```

参数说明

参数名称	参数含义
match tag-pattern	tag-pattern是自定义的用于查找匹配tag的正则表达式，不匹配会报：no patterns matched tag
type	指定output类型为s3类对象存储
s3_endpoint	BOS Endpoint，参考文档
aws_key_id	BOS Access Key ID，参考如何获取AKSK
aws_sec_key	BOS Secret Access Key
s3_bucket	BOS桶名称
path	指定在BOS的存储路径
s3_object_key_format	设置写入BOS的对象命名格式
buffer	指定日志切割规则、时间格式等，详见buffer config
format	指定每行格式

Kafka数据存储到BOS

工具概述

Apache Kafka是一个开源的分布式事件流平台，被广泛用于高性能数据管道、流分析、数据集成和关键任务应用。它支持通过connector方式将事件流数据导出到S3等对象存储系统上，本文将详细阐述如何利用Kafka的S3 Sink Connector插件将数据数据导出至BOS存储桶。

配置教程

1. 下载所需版本的Kafka安装包和confluentinc-kafka-connect-s3安装包，解压并启动Kafka相关服务（注意部分kafka和connector版本存在不兼容问题，需要根据实际情况进行升级，本次以kafka-3.6.2和kafka-connect-s3-10.5.0版本演示）。

```
# 下载安装Kafka，创建plugins目录
tar zxvf kafka_2.13-3.6.2.tgz
cd kafka_2.13-3.6.2
mkdir -p plugins
# 启动Zookeeper
bin/zookeeper-server-start.sh config/zookeeper.properties
# 启动Kafka
bin/kafka-server-start.sh config/server.properties
# 下载安装S3 Sink Connector，放到Kafka安装后的plugins目录下
unzip confluentinc-kafka-connect-s3-10.5.0.zip
cp -r confluentinc-kafka-connect-s3-10.5.0 kafka_2.13-3.6.2/plugins
```

2. 修改Kafka的config目录下的connect-standalone.properties配置文件，添加插件路径和部分配置。

```
# Kafka集群的初始连接地址
bootstrap.servers=localhost:9092
# converter类型配置
key.converter=org.apache.kafka.connect.json.JsonConverter
value.converter=org.apache.kafka.connect.json.JsonConverter
key.converter.schemas.enable=false
value.converter.schemas.enable=false
offset.storage.file.filename=/tmp/connect.offsets
# Kafka Connect将偏移量刷新到存储文件的间隔时间
offset.flush.interval.ms=10000
# Kafka的插件所在具体路径
plugin.path=/path/kafka_2.13-3.6.2/plugins
```

3. 配置S3 Sink Connector插件，修改config/s3-sink.properties配置文件，并启动S3 Sink Connector，详见配置项,如下所示：

```
name=s3-sink
connector.class=io.confluent.connect.s3.S3SinkConnector
# connector可以创建的最大任务数
tasks.max=1
# 指定读取数据的Kafka topic列表
topics=my-topic
# BOS Access Key ID
aws.access.key.id=bos-ak
# BOS Secret Access Key
aws.secret.access.key=bos-sk
# BOS Endpoint，例如https://s3.bj.bcebos.com
store.url=bos-endpoint
# BOS桶名称
s3.bucket.name=bos-bkt
storage.class=io.confluent.connect.s3.storage.S3Storage
format.class=io.confluent.connect.s3.format.json.JsonFormat
partitioner.class=io.confluent.connect.storage.partitioner.DefaultPartitioner
# 将数据刷新到BOS之前缓存的最大记录数
flush.size=3
```

4. 启动S3 Sink Connector，创建kafka topic并写入数据，验证数据是否正常导出到BOS，执行如下命令：

```
# 启动S3 Sink Connector
bin/connect-standalone.sh config/connect-standalone.properties config/s3-sink.properties
# 创建一个名为my-topic的kafka topic
bin/kafka-topics.sh --bootstrap-server localhost:9092 --create --topic my-topic
# 写入数据到my-topic，执行如下命令之后输入数据即可
bin/kafka-console-producer.sh --broker-list localhost:9092 --topic my-topic
# 数据成功导出到BOS后Connector会有如下类似输出
[2024-08-28 12:34:00,186] INFO [s3-sink|task-0] Creating S3 output stream. (io.confluent.connect.s3.storage.S3Storage:200)
[2024-08-28 12:34:00,191] INFO [s3-sink|task-0] Create S3OutputStream for bucket 'bos-bkt' key 'topics/my-topic/partition=0/my-topic+0+0000000000.json'
(io.confluent.connect.s3.storage.S3OutputStream:96)
[2024-08-28 12:34:23,694] INFO [s3-sink|task-0] Starting commit and rotation for topic partition my-topic-0 with start offset {partition=0=0} (io.confluent.connect.s3.TopicPartitionWriter:326)
[2024-08-28 12:34:24,148] INFO [s3-sink|task-0] Files committed to S3. Target commit offset for my-topic-0 is 3 (io.confluent.connect.s3.TopicPartitionWriter:634)
```

5. 然后我们可以从BOS控制台看到导出的topic数据，格式为`bos-bkt/topics/my-topic/partition=0/my-topic+0+0000000000.json`：

文件名称	存储类型	大小	创建时间	操作
my-topic+0+0000000000.json	标准存储	54B	2024-08-28 12:34:24	文件信息 复制链接 下载 更多
my-topic+0+0000000003.json	标准存储	54B	2024-08-28 12:36:10	文件信息 复制链接 下载 更多
my-topic+0+0000000006.json	标准存储	54B	2024-08-28 12:36:25	文件信息 复制链接 下载 更多
my-topic+0+0000000009.json	标准存储	22B	2024-08-28 14:32:47	文件信息 复制链接 下载 更多
my-topic+0+0000000012.json	标准存储	6B	2024-08-28 14:32:48	文件信息 复制链接 下载 更多

Hbase使用BOS作为底层存储

HBase

HBase 是一个面向列式存储的分布式数据库，旨在提供对大量结构化数据的快速随机访问，底层存储一般基于 HDFS 实现。

前提条件

首先参考 BOS HDFS 一文安装并配置 BOS HDFS，本机安装的 Hadoop 版本为 hadoop-3.3.2，参考文中"开始使用"一节完成 BOS HDFS 的基本试用，并设置环境变量：

```
export HADOOP_HOME=/opt/hadoop-3.3.2
export HADOOP_CLASSPATH=`$HADOOP_HOME/bin/hadoop classpath
```

安装

1.HBase 环境准备

```
##### 下载到一个路径
wget https://www.apache.org/dyn/closer.lua/hbase/2.6.0/hbase-2.6.0-bin.tar.gz

##### 解压
tar zxvf hbase-2.6.0-bin.tar.gz
```

2.配置 在conf/hbase-env.sh里配置JAVA_HOME：

```
##### 选择机器已安装的java环境，至少1.8版本
export JAVA_HOME=/usr/java/jdk1.8.0/
```

在conf/hbase-site.xml里配置，使用BOS存储数据：

```
<property>
  <name>hbase.rootdir</name>
  <value>bos://{bucket}/hbase</value>
  <description>此项用于设置持久化存储HBase数据路径，使用对象存储bos时，需设置为带"bos://{bucket}/"前缀的路径</description>
</property>

<property>
   <name>hbase.wal.dir</name>
   <value></value>
   <description>此项用于设置WAL数据路径，要求低延迟，一般使用HDFS存储。如果使用BOS需确保集群的BOS-HDFS版本支持hflush/hfsync接口</description>
</property>

<property>
   <name>hbase.zookeeper.property.dataDir</name>
   <value>/var/zookeeper</value>
   <description>此项用于设置存储ZooKeeper的元数据，如果不设置默认存在/tmp下，重启时数据会丢失。</description>
</property>

<property>
   <name>hbase.cluster.distributed</name>
   <value>false</value>
   <description>此项用于设置分布式集群模式，false为单机或伪分布式模式，true为完全分布式模式</description>
</property>
```

使用

1.启动 HBase

```
./bin/start-hbase.sh
```

2.创建表

```
./bin/hbase shell
>status          # 查看集群状态
1 active master, 0 backup masters, 1 servers, 0 dead, 2.0000 average load
Took 0.7840 seconds

>create 'students','name','age' # 创建students表
2024-09-02 19:23:25,153 INFO  [main] client.HBaseAdmin (HBaseAdmin.java:postOperationResult(3746)) - Operation: CREATE, Table Name: default:students, procId: 9 completed
Created table students
Took 2.4410 seconds
=> Hbase::Table - students
```

3.插入数据

```
>put 'students', 'row1', 'name:lastname', 'zhang'
Took 0.0820 seconds
> put 'students', 'row1', 'name:firstname', 'san'
Took 0.0900 seconds
> put 'students', 'row1', 'age', '23'
Took 0.0990 seconds
> put 'students', 'row2', 'name:lastname', 'li'
Took 0.0710 seconds
> put 'students', 'row2', 'name:firstname', 'si'
Took 0.0520 seconds
> put 'students', 'row2', 'age', '30'
Took 0.0920 seconds
```

查看BOS上存储的数据

文件名称	存储类型	大小	创建时间
.hbck/	-	-	-
.tmp/	-	-	-
MasterData/	-	-	-
archive/	-	-	-
data/	-	-	-
hbase.id	标准存储	42B	2024-09-02 19:20:13
hbase.version	标准存储	7B	2024-09-02 19:20:13
mobdir/	-	-	-
staging/	-	-	-

4.全表扫描

```
>scan 'students'
ROW                                   COLUMN+CELL
 row1                                 column=age:, timestamp=2024-09-02T19:37:56.571, value=23
 row1                                 column=name:firstname, timestamp=2024-09-02T19:37:31.480, value=san
 row1                                 column=name:lastname, timestamp=2024-09-02T19:36:09.318, value=zhang
 row2                                 column=age:, timestamp=2024-09-02T19:38:50.066, value=30
 row2                                 column=name:firstname, timestamp=2024-09-02T19:38:38.772, value=si
 row2                                 column=name:lastname, timestamp=2024-09-02T19:38:24.245, value=li
2 row(s)
Took 0.0350 seconds
```

5.退出

```
>quit
```

Druid使用BOS作为Deep storage

Druid

Apache Druid 是一个高性能的实时分析型数据库。专为需要快速数据查询与摄入的工作流程而设计，在即时数据可见性、即席查询、运营分析以及高并发等方面表现非常出色。

通过BOS-HDFS插件，支持将BOS作为deep storage存储数据。

安装

1.环境准备

下载安装Druid，略。

在BOS-HDFS下载依赖的jar包，并拷贝到Druid安装路径的extensions/druid-hdfs-storage目录下。

2.修改配置

修改conf/druid/_common/common.runtime.properties 文件，使用HDFS方式接入BOS。

```
druid.extensions.loadList=["druid-hdfs-storage"]
druid.storage.type=hdfs
druid.storage.storageDirectory=bos://{bucket}/druid-path
```

在conf/druid/_common/目录下新建一个 hdfs 的配置文件 hdfs-site.xml，配置BOS访问参数。

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<!--
  Licensed under the Apache License, Version 2.0 (the "License");
  you may not use this file except in compliance with the License.
  You may obtain a copy of the License at
    http://www.apache.org/licenses/LICENSE-2.0
  Unless required by applicable law or agreed to in writing, software
  distributed under the License is distributed on an "AS IS" BASIS,
  WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
  See the License for the specific language governing permissions and
  limitations under the License. See accompanying LICENSE file.
-->
<!-- Put site-specific property overrides in this file. -->

<configuration>
    <property>
      <name>fs.bos.access.key</name>
      <value>{Your AK}</value>
    </property>

    <property>
      <name>fs.bos.secret.access.key</name>
      <value>{Your SK}</value>
    </property>

    <property>
      <name>fs.bos.endpoint</name>
      <value>http://bj.bcebos.com</value>
    </property>

    <property>
      <name>fs.bos.impl</name>
      <value>org.apache.hadoop.fs.bos.BaiduBosFileSystem</value>
    </property>

    <property>
      <name>fs.AbstractFileSystem.bos.impl</name>
      <value>org.apache.hadoop.fs.bos.BOS</value>
    </property>
</configuration>
```

更多可参考BOS-HDFS中的可配置属性。

使用

启动Druid相关服务，就可以用BOS作为deep storage存储数据了。

数据魔方

批量处理

概述

批量处理功能支持用户对 BOS 存储桶内指定的对象列表执行指定的批量处理操作。您首先需要指定一份待处理对象清单，完成批量处理任务配置后，BOS 即可根据清单执行批量处理任务。

> 说明： 目前仅支持对华北-北京、华东-苏州、华南-广州地域的存储桶中的文件进行批量处理操作。

批量处理任务类型

目前支持的批量处理任务类型包括：批量数据复制和批量取回归档文件。

1. **批量数据复制：**将指定源 Bucket 中的部分或全部对象复制到指定目标 Bucket 的指定目录下，支持为复制后的对象选择新的存储类型、对象元数据、访问权限等配置信息。如遇数据复制的源 Bucket 和目标 Bucket 所在地域不同，会产生跨区域流出流量费用。
2. **批量取回归档文件：**通过配置一个批量取回任务批量取回归档文件，取回后自动生成一份标准存储类型的临时副本文件，在有效期内该副本文件可以正常读取，不受取回限制。

创建批量处理任务

准备工作

准备清单文件：目前支持的清单格式包括 BOS 清单报告和 CSV 文件。

- **BOS 清单报告：**BOS 可根据用户的清单任务配置，定期扫描用户 Bucket 中指定对象并输出一份清单报告，以 CSV 格式文件的形式存储到指定的 Bucket 中。如何设置清单可参见 设置存储空间清单。
- **CSV 文件：**

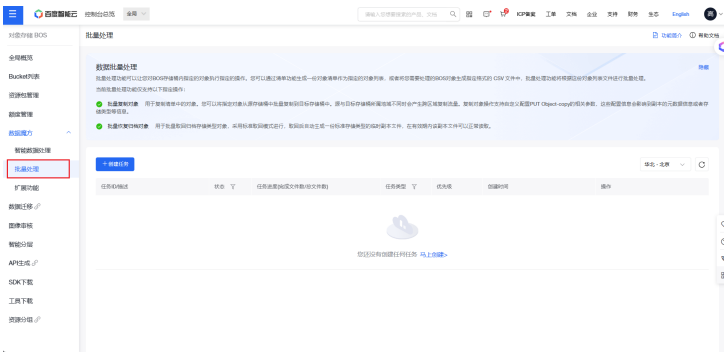
1. 创建 CSV 格式的待处理对象清单列表。清单列表是记录所有待处理对象的列表文件，用于告知BOS待处理的文件。自定义的CSV文件支持指定为，每行两列，并按照文件所属Bucket名称、文件名称（Key值）顺序排列。如下图：

```
bucketName,test.md
bucketName,example.md
```

2. 将自定义 CSV 文件上传至存储桶。

使用控制台创建批量处理任务

1. 登录 对象存储 BOS 管理控制台。
2. 在控制台左侧全局导航栏中选择 **数据魔方-批量处理** 入口，进入批量处理任务管理页面。
3. 点击**创建任务**，开始创建批量处理任务。



4. 配置项说明如下：

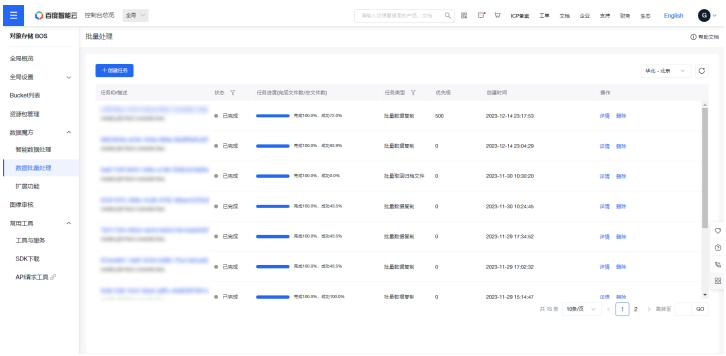
模块	参数	说明
清单文件配置	清单格式	BOS 清单报告 CSV 文件
	清单存储桶	清单文件所在的 Bucket
	清单文件路径	清单文件的具体路径。若您选择的清单格式为： • BOS 清单报告。清单文件路径请选择 Bucket 中的 manifest 文件，文件名和后缀必须为manifest.json • CSV 文件。清单文件路径请选择 Bucket 中的 CSV 文件，后缀必须为 .csv
任务配置	任务类型	批量数据复制 批量取回归档文件
	任务优先级	任务优先级用于标识当前批量处理任务相对于其他批量处理任务的优先状态，数值越大优先级越高。优先级数值范围为0 - 2147483647。
	任务描述	对该任务进行描述，长度100个字符，支持中英文数字及特殊字符。
生成配置	任务报告范围	全部操作记录 仅记录失败操作
	报告投递 Bucket	存储报告的 Bucket。
	报告投递前缀	报告的名称目录前缀。

不同任务类型下的参数配置说明如下：

任务类型	参数	说明
批量数据复制	目标Bucket	复制后的数据存放的 Bucket。
	前缀操作	可选择根目录或设置前缀。对复制后的文件的前缀或目录前缀进行添加、替换或者删除。
	对象元数据	对复制后的对象设置元数据，可选复制所有元数据、替换所有元数据： • 复制所有元数据：即将源文件元数据全部复制到目的端，无需额外配置。 • 替换所有元数据：对源文件的全部header进行替换，自定义meta支持输入key。
	服务端加密	不加密
		BOS 托管密钥
	访问权限	与目标 Bucket 一致
		设置为公共读
		设置为私有
批量取回归档文件	副本有效期	副本在多少天后自动过期删除，范围为 7-30 天。
	取回模式	目前支持标准取回。

5. 确认无误后，单击 **确定** 创建批量处理任务成功。

6. 创建成功后，可在批量处理任务列表中查看任务的状态、进度、详情等。当任务状态为准备中或进行中时，可**取消任务**。当任务状态为等待启动/取消中/已取消/已完成/任务失败时，可**删除任务**。



7. 任务列表中的批量处理任务在任务结束后保留90天，过期将自动删除。

zip打包压缩

简介

ZIP 多文件打包压缩是百度智能云对象存储（Baidu Object Storage，BOS）基于 函数计算（Cloud Function Compute） 为用户提供的数据处理解决方案。用户为存储桶添加多文件打包压缩规则后，指定需要打包压缩文件的 URLs，再通过触发 CFC 函数来执行打包压缩的动作，并将最终的压缩包投递至存储桶的指定路径下。

注意事项

- ZIP 多文件打包压缩最终生成的压缩包不能大于50GB。
- 若您此前在对象存储控制台上为存储桶添加了 ZIP 多文件打包压缩规则，可以在 云函数控制台 上看到您所创建的 ZIP 多文件打包压缩函数，请**不要**删除该函数，否则可能导致您的规则不生效。
- 如果在打包压缩的过程中出现报错，您可以单击所创建的函数右侧的**查看日志**，跳转到云函数控制台查看日志错误详情。
- **归档存储文件需要先解冻取回**后才能打包压缩，若未解冻则打包失败。如您需要打包压缩此类型的对象，请先取回后再进行。
- 仅CFC支持地域可使用该功能。目前暂支持：**华北-北京**、**华南-广州**、**华东-苏州**地域，函数可基于以上地域创建。

操作步骤

> - 在控制台开始使用文件打包压缩功能之前，您需要开通CFC服务，才可创建文件打包压缩规则。
> - 用户也可以在 云函数控制台 上自行创建函数调用使用。

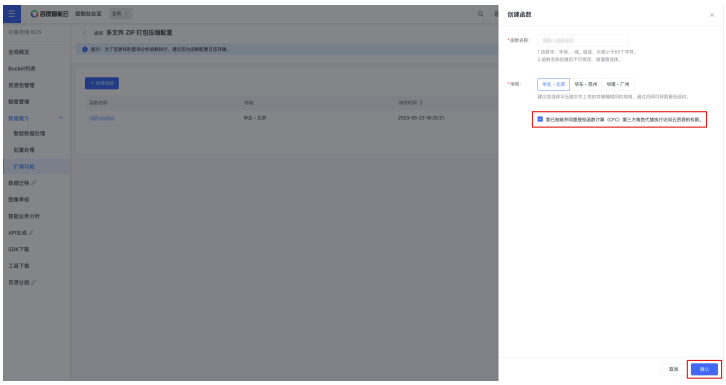
在控制台使用多文件打包压缩功能的步骤如下所示：

1. 打开百度智能云控制台，进入BOS控制界面。

2. 在左侧导航栏中，点击【数据魔方】，选择【扩展功能】，即可使用多文件ZIP打包压缩功能，未开通CFC服务的用户按指引开通后即可使用。

3. 点击【立即配置】进入打包函数配置界面，点击左上角【创建函数】新建打包函数，输入函数名称、选择地域、勾选CFC授权后点击【确认】即可完成新建函数操作。

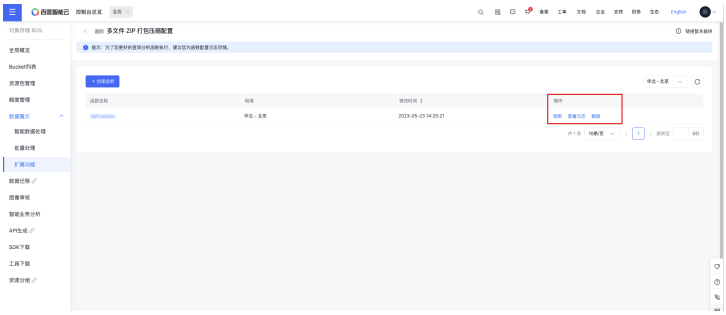


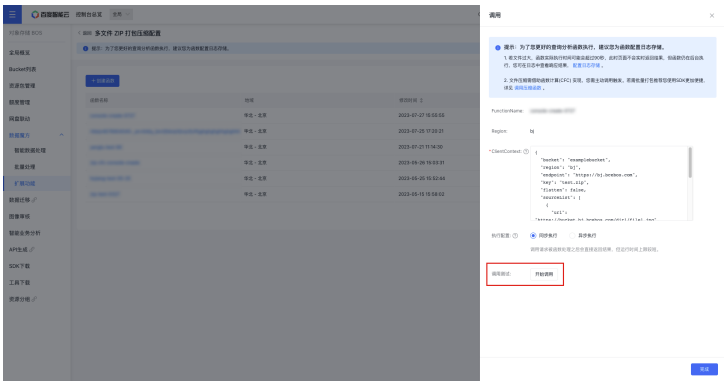


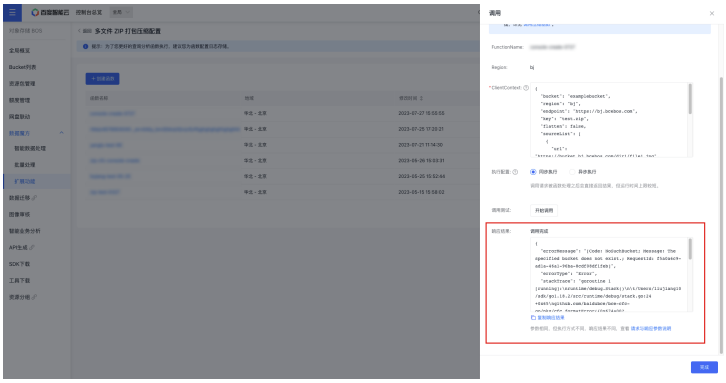
4. 在函数列表中选择你想要调用的函数，点击操作栏中的【调用】进入函数配置界面。各参数解释见下方多文件压缩参数解释。

5. 配置完函数各参数后，点击【开始调用】即可执行该函数，您可在响应结果中查看调用完成后的相关信息。

6. 为了您更好的查询、分析函数执行，您可在函数列表操作栏中的【查看日志】对函数配置日志存储，配置日志存储后可查看函数的日志信息。







多文件压缩参数解释

函数参数配置示例

> 实际使用当中，需将代码中的注释去掉。

```
{
  "bucket": "examplebucket",         // 最终投递 ZIP 文件的存储桶
  "region": "bj",                    // 最终投递 ZIP 文件的存储桶所在地域
  "endpoint":"https://bj.bcebos.com", // 最终投递 ZIP 文件的存储桶对外服务的访问域名
  "key": "test.zip",                 // 最终投递 ZIP 文件的名称
  "flatten": false,                  // 是否需要对源文件路径进行扁平化处理

  /**
   * sourceList 用于指定需打包的源文件列表，格式为 JSON 数组
   * 每一项包含源文件 url，重命名的路径 renamePath 等
   *
   */
  "sourceList": [
    {
      "url": "https://bucket.bj.bcebos.com/dir1/file1.jpg",
      "renamePath": "dir1_rename/file1.jpg"
    },
    {
      "url": "https://bucket.bj.bcebos.com/dir2/file2.mp4",
      "renamePath": "file2.mp4"
    },
    {
      "url": "https://bucket.bj.bcebos.com/file3.md"
    }
  ]
}
```

参数说明如下：

参数名	参数描述	类型	是否必填
bucket	最终投递 ZIP 文件的存储桶，命名格式为 BucketName，例如：examplebucket	String	是
region	最终投递 ZIP 文件的存储桶所在地域，枚举值请参见 区域选择说明	String	是
endpoint	最终投递 ZIP 文件的存储桶的对外访问域名，枚举值请参见 区域选择说明	String	是
key	最终投递 ZIP 文件的名称（Object 的名称），对象在存储桶中的唯一标识，	String	是
flatten	是否需要路径扁平化（去除源目录结构），例如：源文件 URL 为 https://domain/source/test.mp4, 则源文件路径为 source/test.mp4，如果为 true，则 ZIP 包中该文件路径为 test.mp4，否则 ZIP 包中该文件路径为 source/test.mp4。默认为 false	Boolean	否
sourceList	源文件列表，sourceList	Array	是
sourceList[].url	源文件的 URL	String	是
sourceList[].renamePath	带路径的重命名，即源文件在 ZIP 包中的文件路径。例如，将 dir1/file1.jpg 重命名为 dir1_rename/file1.jpg。 注意：renamePath 的优先级高于 flatten，重命名后的路径不受扁平化影响	String	否

函数响应结果示例

同步调用结果示例：

```
{
  "Code": 0,
  "Message": "zip success!",
  "Data": {
    "Bucket": "<bucket_name>",
    "Key": "package1.zip",
    "Location": "http://bj.bcebos.com/<bucket_name>/package1.zip",
    "Etag": "-cf3cac621791214721d845becc3e9906"
  },
  "Error": null
}
```

响应参数说明如下：

参数名	参数描述	类型
Code	业务错误码，如果为 0 则说明执行成功，否则为执行失败	Number
Message	执行结果的文字说明，可能为 null	String
Data	执行成功的信息，如果执行成功，则包含 ZIP 包的 url 信息	Object
Error	执行的错误信息，如执行成功则为 null	Object or String

异步调用结果返回202，具体执行日志见函数配置的日志路径

调用压缩函数

- 文件压缩需借助函数计算(CFC) 实现，您需主动调用触发。
- 若需批量打包推荐您使用SDK更加便捷。

1. 您可前往SDK概览查看各语言安装SDK的指引文档，并根据文档安装对应的SDK。
2. 安装SDK后，可以通过SDK调用CFC的调用函数接口（invocations）从而触发对应的云函数，具体参数见函数调用接口。

实际案例

案例一：简单案例

参数配置

```
{
 "bucket": "<bucket_name>",
 "region": "bj",
 "endpoint":"https://bj.bcebos.com",
 "key": "mypack.zip",
 "flatten": false,
 "sourceList": [
    {
        "url": "https://<bucket_name>.bj.bcebos.com/dir1/file1.jpg"
    },
    {
        "url": "https://<bucket_name>.bj.bcebos.com/dir2/file2.mp4"
    },
    {
        "url": "https://<bucket_name>.bj.bcebos.com/file3.md"
    }
 ]
}
```

最终 ZIP 压缩包结构

```
mypack.zip
    ├──── dir1/file1.jpg
    ├──── dir2/file2.mp4
    └──── file3.md
```

案例二：源文件路径扁平化

参数配置

```
{
 "bucket": "<bucket_name>",
 "region": "bj",
 "endpoint":"https://bj.bcebos.com",
 "key": "mypack.zip",
 "flatten": true,                // flatten 为 true，对源文件路径进行扁平化处理
 "sourceList": [
    {
        "url": "<bucket_name>.bj.bcebos.com/dir1/file1.jpg"
    },
    {
        "url": "<bucket_name>.bj.bcebos.com/dir2/file2.mp4"
    },
    {
        "url": "<bucket_name>.bj.bcebos.com/file3.md"
    }
 ]
}
```

最终 ZIP 压缩包结构

```
mypack.zip
    ├──── file1.jpg
    ├──── file2.mp4
    └──── file3.md
```

案例三：源文件路径重命名

参数配置

```
{
 "bucket": "<bucket_name>",
 "region": "bj",
 "endpoint":"https://bj.bcebos.com",
 "key": "mypack.zip",
 "flatten": false,
 "sourceList": [
    {
        "url": "https://<bucket_name>.bj.bcebos.com/dir1/file1.jpg",
        // 将 dir1/file1.jpg 路径重命名为 dir1_rename/file1.jpg
        "renamePath": "dir1_rename/file1.jpg"
    },
    {
        "url": "https://<bucket_name>.bj.bcebos.com/dir2/file2.mp4",
        // 将 dir2/file2.mp4 重命名为 file2.mp4
        "renamePath": "file2.mp4"
    },
    {
        "url": "https://<bucket_name>.bj.bcebos.com/file3.md"
    }
 ]
}
```

最终 ZIP 压缩包结构

```
mypack.zip
    ├──── dir1_rename/file1.jpg
    ├──── file2.mp4
    └──── file3.md
```

案例四：源文件路径重命名 + 扁平化

参数配置

```
{
 "bucket": "<bucket_name>",
 "region": "bj",
 "endpoint":"https://bj.bcebos.com",
 "key": "mypack.zip",
 "flatten": true,        // flatten 为 true，对源文件路径进行扁平化处理
 "sourceList": [
    {
       "url": "https://<bucket_name>.bj.bcebos.com/dir1/file1.jpg"
    },
    {
       "url": "https://<bucket_name>.bj.bcebos.com/dir2/file2.mp4"
    },
    {
       "url": "https://<bucket_name>.bj.bcebos.com/file3.md",
       // 将 file3.md 重命名为 dir3/file3.md，renamePath 优先级比 flatten 高，因此重命名后的路径不会被扁平化
       "renamePath": "dir3/file3.md"
    }
 ]
}
```

最终 ZIP 压缩包结构

```
mypack.zip
  ├── file1.jpg
  ├── file2.mp4
  └── dir3/file3.md
```

智能数据处理

视频处理

获取视频元信息

获取视频元信息

接口描述

本接口用于获取储存在BOS上的视频文件的元信息。

权限说明

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。

注意事项

请求结构

```
 GET /<ObjectName>?x-bce-process=video/info HTTP/1.1
Host: <BucketName>.bj.bcebos.com
Authorization: authorization string
```

请求头域

除公共头域外，无其它特殊头域。

请求query参数 | 参数名称 | 类型 | 描述 |
| -------- | -------- | ---------- | | x-bce-process | string | bos数据处理参数，获取视频元信息为x-bce-process=video/info|

响应头域

除公共头域外，无其它特殊头域。

请求示例

```
 GET http://<BucketName>.bj.bcebos.com/<ObjectName>?x-bce-process=video/info
Host: <BucketName>.bj.bcebos.com
X-Bce-Date:2022-11-16T11:54:00Z
Authorization: bce-auth-v1/f81d3b34e48048fbb2634dc7882d7e21/2015-08-11T04:17:29Z/3600/host/74c506f68c65e26c633bfa104c863fffac5190fdec1ec24b7c03eb5d67d2e1de
```

响应示例

```
 HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
Date: Wed, 08 Jul 2015 03:28:11 GMT
x-bce-request-id: d8752367-38e8-45e4-b4c7-e53be3137ce5
Server: BOS
{
    "bucket": "cmdtest2",
    "key": "audio.mp3",
    "fileSizeInByte": 4008612,
    "type": "audio",
    "container": "mp3",
    "durationInSecond": 247,
    "audio": {
        "codec": "mp3",
        "channels": 2,
        "sampleRateInHz": 44100,
        "bitRateInBps": 128000
    },
    "etag": "575a58dbfec5c808dcd941f927ae1927"
}
```

返回参数 | 参数名称 | 描述 |
| -------- | --------- | | bucket | 视频所在桶名称| | key | 视频名称| | fileSizeInByte | 视频大小| | type | 视频类型| | container | 视频格式| | durationInSecond | 视频时长（秒）| | codec | 视频格式| | channels | 视频通道数| | sampleRateInHz | 视频采样率| | etag | 文件在BOS上的etag |

注意事项

- 当前BOS仅支持北京，广州，苏州，保定集群使用该接口
- 该接口暂时免费公测，后续计费标准待公布

视频即时转码

基本概念

百度智能云BOS提供了业界全新的视频即时转码接口，支持通过设置url参数的方法即时转换视频的码率，实现h264，h265两种码率的互转。

注意事项

- 使用前需开通百度智能云音视频处理服务（MCP），即时转码能力的计费部分由MCP收取。
- 使用BOS的视频转码时，目前为公测，暂时免费，商用后BOS会按视频即时转码的数量计费。
- BOS不保存转码后的视频，转码后的视频需手动下载并保存至本地。
- 建议的转码视频大小：小于1GB
- 建议的转码分辨率：不超过1080P

视频转码命令

视频文件处理时间较长，通常在秒级。

- 操作分类 video
- 操作名称 codec

BOS视频处理接口定义如下：

- 参数说明

参数名称	缩写	类型	取值	命令描述	是否必选
format	f	string	[h264,h265]	视频转码的目标格式	是
priority	p	int	[0,1]	1：速度优先，质量略差；0：速度落后，质量较高	否

请求示例

- 使用默认模式将h265格式的视频转为h264

处理的URL为：<原视频URL>?x-bce-process=video/codec,f_h264

- 使用高优先级模式将h264码率视频转为h265

处理的URL为：<原视频URL>?x-bce-process=video/codec,f_h265,p_1

规则和限制

- 目前仅支持北京、苏州、广州、保定使用
- x-bce-process仅支持在url参数或header中添加

视频截帧

基本概念

百度智能云BOS提供了业界全新的视频截帧接口，支持通过设置url参数的方法对存储在bos的视频文件进行截取

注意事项

- 使用视频截帧，2022年4月1日正式商业化，计费详情参见BOS价格说明-〉数据处理-〉视频截帧。

视频截帧命令

视频文件远大于图片文件，处理时间也较长，通常在秒级。

- 操作分类 video
- 操作名称 snapshot

BOS视频处理接口定义如下：

- 参数说明

参数名称	缩写	类型	取值	命令描述	是否必选
time	t	int	视频时长之内	截帧的时间，单位为毫秒	是
format	f	string	[jpg,png,bmp,gif,tif]	输出的图片格式	否
width	w	int	[0,4096]	指定截图宽度	否
height	h	int	[0,4096]	制定截图高度	否
auto-rotate	ar	string	[90,180,270,auto]	指定旋转角度，若设置为auto，则会根据视频文件信息自动旋转	否
mode	m	string	fast	截图类型，若设置为fast，则截取time时间点最近一个关键帧	否

视频截帧下载

- 截图下载示例

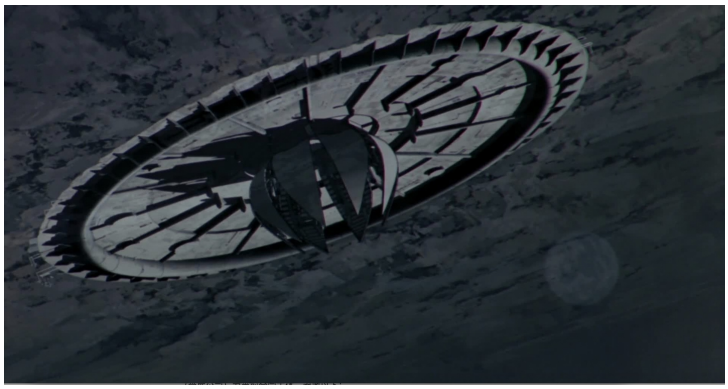
http://${bucketname}.bj.bcebos.com/${objectname}?x-bce-process=video/snapshot,t_5000&responseContentDisposition=attchment **视频截帧转存BOS**

- 转存BOS示例

http://${bucketname}.bj.bcebos.com/${objectname}?x-bce-process=video/snapshot,t_5000|system/save,o_dGVzdC5qcGVn **请求示例**

- 使用默认模式截取视频7s处的内容，默认情况下输出格式为jpg，输出宽高为原视频宽高

处理的URL为：<原视频URL>?x-bce-process=video/snapshot,t_7000



- 使用关键帧模式截取视频7s处的内容，输出为PNG格式的图片，宽度为800，高度为600,且旋转角度为90度。

处理的URL为：<原视频URL>?x-bce-process=video/snapshot,t_7000,f_png,w_800,h_600,ar_90,m_fast



规则和限制

- 目前仅支持北京、苏州、广州、保定地域使用
- 目前仅支持在页面添加url参数的方式进行截取
- 支持截帧的最大视频大小为10G，支持私有object截取，需在url的参数中增加authorization签名，签名生成可使用IAM工具，链接如下 https://cloud.baidu.com/signature/index.html

音视频处理

基本概念

音视频行业用户在将BOS作为后端存储源站时，结合上层业务逻辑往往会有不同的使用场景。 随着业务层对用户体验的不断演进，音视频业务对数据加工、处理、访问的实时性要求越来越高，并且希提供望更加简化的使用方式。

百度智能云BOS提供了业界全新的音视频处理接口，支持上传mp4格式的文件，系统自动保存为hls格式，用于m3u8风格播放器读取使用。充分满足了“UGC上传——自动化处理——播放”这条使用流，免去了原本复杂的视频操作处理逻辑，并为业务提速。

音视频处理命令

音视频文件远大于图片文件，处理时间也较长，通常在秒级。因此，BOS音视频处理命令，采用了相较于BOS图片处理接口有所区别的写时处理方案，即音视频原文件写入BOS时即开始按需处理，而非基于用户读取时定义的参数处理。

BOS音视频处理接口定义如下：

- 格式转换参数

参数名称	缩写	类型	取值	命令描述	是否必选
format	f	string	[hls]	将视频转换转化为指定格式	是
time	t	int	[6,600]	切片时间，默认10s	否

- 持久化存储参数

参数名称	缩写	类型	取值	命令描述	是否必选
object	o	string	-	经过base64编码的object名称，受object长度限制，即不超过1024个字符	是

请求方法

在BOS的POST接口中，以header形式填写参数，即可实现POST上传的音视频文件，按照header中的命令实现转封装及转存。

```
 POST /video.mp4 HTTP/1.1
Content-Length: ContentLength
Content-Type: ContentType
x-bce-process: video/format,f_hls
```

请求示例

转封装并存储示例

将视频video.mp4转封装为hls，并将结果存储到video/demo.hls。

```
 POST /video.mp4 HTTP/1.1
Content-Length: ContentLength
Content-Type: ContentType
x-bce-process: video/format,f_hls|system/save,o_dmlkZW8vZGVtby5tM3U4
```

规则和限制

- 支持的原音视频格式：mp4
- 支持的目标音视频格式：hls
- 支持处理的视频大小不超过 100 MB
- 目前支持北京，广州，苏州，保定地域使用
- 目前仅支持在bos的post风格上传接口中使用

当前BOS音视频处理仅支持mp4文件向hls文件类型的转换，更多类型的支持、文件码转换等能力，BOS后续将逐步支持。

如您短期内希望采用更广泛的处理能力，同时业务侧也可允许相对长一些的处理延迟，也可参考百度智能云音视频转码MCP、百度智能云音视频点播VOD。

M3U8签名

M3U8签名

接口描述

M3U8签名用于私有空间中的 m3u8 文件，作用是对 m3u8文件中的 ts 资源进行批量下载授权。通过将 ts 资源的 url 改写成私有 url，以临时获取访问权限。

权限说明

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。

注意事项

如果请求未携带签名认证认证信息，该接口将返回对象原数据。

请求结构

```
 GET /<ObjectName>?x-bce-process=hls/sign,expires_3600 HTTP/1.1
Host: <BucketName>.bj.bcebos.com
Authorization: authorization string
```

请求头域

除公共头域外，无其它特殊头域。

请求query参数 | 参数名称 | 类型 | 描述 |
| -------- | -------- | ---------- | | x-bce-process | string | bos数据处理参数，获取M3U8签名为x-bce-process=hls/sign,expires_xxx|

- 操作分类 hls
- 操作名称 sign

M3U8签名接口参数定义如下：

- 参数说明

参数名称	类型	取值	命令描述	是否必选
expires	int	[3600,43200]	生成签名过期时间,单位秒	是

使用示例

```
 GET http://databin.bj.bcebos.com/hls/demo.m3u8?x-bce-process=hls/sign,expires_3600
Host: databin.bj.bcebos.com
X-Bce-Date:2022-11-16T11:54:00Z
Authorization: bce-auth-v1/f81d3b34e48048fbb2634dc7882d7e21/2015-08-11T04:17:29Z/3600/host/74c506f68c65e26c633bfa104c863fffac5190fdec1ec24b7c03eb5d67d2e1de
```

原始m3u8链接与内容如下所示：

```
 ###### EXTM3U
###### EXT-X-VERSION:3
###### EXT-X-MEDIA-SEQUENCE:0
###### EXT-X-ALLOW-CACHE:YES
###### EXT-X-TARGETDURATION:62
###### EXTINF:61.120000,
000000.ts
###### EXTINF:59.360000,
000001.ts
###### EXTINF:59.640000,
000002.ts
###### EXTINF:2.480000,
000003.ts
###### EXT-X-ENDLIST
```

对m3u8资源使用m3u8签名接口，可以获取到相应的批量授权下载URL：

```
 ###### EXTM3U
###### EXT-X-VERSION:3
###### EXT-X-MEDIA-SEQUENCE:0
###### EXT-X-TARGETDURATION:62
###### EXTINF:61.120,
000000.ts?x-bce-security-
token=ZjkyZmQ2YmQxZTQ3NDcyNjk0ZTg1ZjYyYjlkZjNjODB8AAAAALECAABEAdI6kNdjSB%2BTmHpMSc0dMXaWHGlooj7S5Db3cCJveP%2BeiUwao6%2FlWgUhpzSWB%2BO2LT3tC%2BZpBv2umQRVUmn3MX%2FwbjjpuC
-auth-v1%2Fd7c4970a293311eeaaccb55a6be64ec1%2F2023-07-23T08%3A35%3A13Z%2F3600%2Fhost%2Ffec8f161023133c69368b58f33eeb88650fc0e9829833c5fb8dd43a823f8eaf0
###### EXTINF:59.360,
000001.ts?x-bce-security-
token=ZjkyZmQ2YmQxZTQ3NDcyNjk0ZTg1ZjYyYjlkZjNjODB8AAAAALECAABEAdI6kNdjSB%2BTmHpMSc0dMXaWHGlooj7S5Db3cCJveP%2BeiUwao6%2FlWgUhpzSWB%2BO2LT3tC%2BZpBv2umQRVUmn3MX%2FwbjjpuC
-auth-v1%2Fd7c4970a293311eeaaccb55a6be64ec1%2F2023-07-23T08%3A35%3A13Z%2F3600%2Fhost%2F3d146f7de5ec5390a364bcf3ab4da8da7d264567c88371cf77c0817b26a22af6
###### EXTINF:59.640,
000002.ts?x-bce-security-
token=ZjkyZmQ2YmQxZTQ3NDcyNjk0ZTg1ZjYyYjlkZjNjODB8AAAAALECAABEAdI6kNdjSB%2BTmHpMSc0dMXaWHGlooj7S5Db3cCJveP%2BeiUwao6%2FlWgUhpzSWB%2BO2LT3tC%2BZpBv2umQRVUmn3MX%2FwbjjpuC
-auth-v1%2Fd7c4970a293311eeaaccb55a6be64ec1%2F2023-07-23T08%3A35%3A13Z%2F3600%2Fhost%2F9bfe3ff75fba9a48fc22b212ba7168810b82540fe656aac3da4f8960d2a58939
###### EXTINF:2.480,
000003.ts?x-bce-security-
token=ZjkyZmQ2YmQxZTQ3NDcyNjk0ZTg1ZjYyYjlkZjNjODB8AAAAALECAABEAdI6kNdjSB%2BTmHpMSc0dMXaWHGlooj7S5Db3cCJveP%2BeiUwao6%2FlWgUhpzSWB%2BO2LT3tC%2BZpBv2umQRVUmn3MX%2FwbjjpuC
-auth-v1%2Fd7c4970a293311eeaaccb55a6be64ec1%2F2023-07-23T08%3A35%3A13Z%2F3600%2Fhost%2F04fdee6283aeb8c9be1a6ea042887986ff9368be1b187adb74351dcc3793b2ce
###### EXT-X-ENDLIST
```

注意:ts资源签名以该对象为 m3u8 文件同级目录 URL 为准.

以上面 m3u8 文件为例，对象 URL 为 `http://databin.bj.bcebos.com/hls/demo.m3u8`, 则内部文件 000000.ts 签名的实际 URL 为 `http://databin.bj.bcebos.com/hls/000000.ts`

图像处理

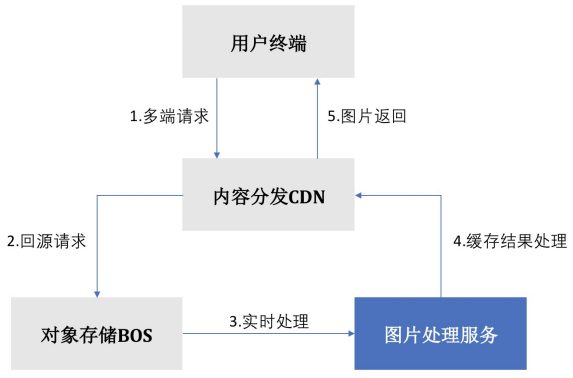
图像处理介绍

概述

BOS 图片处理是一项基于 BOS 核心存储业务的图片应用类服务，可提供灵活、多样、实时的图片处理服务。您只需要将原始图片上传到 BOS，通过在原始图片的请求中添加处理参数，即可实时地得到处理结果图。BOS 图片处理服务包括图片缩略、裁剪、格式转换、旋转、加文字/图片水印等，处理结果图会实时返回。

您可以通过 BOS 控制台和 API 方式对存储在 BOS 中的图片进行处理。您在 BOS 中创建的每个 Bucket 均默认具有图片处理的能力，无需额外开通操作。

图片处理服务的流程如下图所示：



- 用户首次发送请求处理流程：
  1. 用户端首次发送图片处理请求;
  2. 首次请求，CDN未命中，请求会被直接发送到BOS源站进行处理；
  3. 根据用户发送的处理参数对图片进行实时处理；
  4. 处理后的结果图将被缓存到CDN节点上；
  5. 同时，将结果图返回给用户端。
- 用户端非首次访问图片处理流程：

1. 用户端发送图片处理请求;
2. 若CDN命中，则直接跳到步骤5；若CDN未命中，则跳到步骤3；
3. 请求会被回源到BOS，并根据用户发送的处理参数对图片进行实时处理；
4. 处理后的结果图被缓存到CDN节点上；
5. 将结果图返回给用户端。

在访问速度上，图片处理服务集成了百度智能云内容分发网络 CDN 功能，可利用遍布全球的高质量 CDN 节点，实现图片内容的加速分发，有效降低源站压力。

计费方式

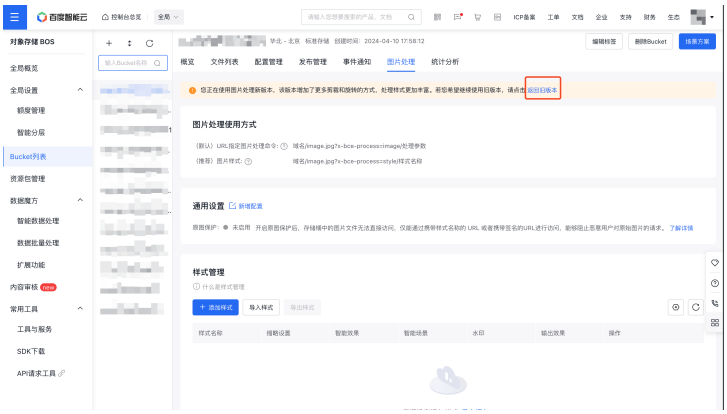
图像基础处理、图像高级处理中 HEIC 图像压缩及 WebP-M6 图像压缩服务将于 2020 年 6 月 1 日 0 点开始正式计费。当前，您可以登录控制台-财务查看用量数据。BOS 费用会按小时进行出账，具体价格请参考数据处理费用。

计费项	计费说明
基础图像处理费用	图像基础处理费用是指对存储在 BOS 中的图像进行缩放、裁剪、旋转、格式转换、渐进显示、质量变换、水印等基础图片处理操作时产生的费用。BOS 会根据处理前的图片大小进行计费。BOS 为每位用户每月免费赠送 10 TB 的图像基础处理额度。当您当月处理的图片量大于 10 TB 时，将按超出部分的使用量进行计费。
HEIC 图像压缩费用 (800x600 以下规格)	属于图像高级处理服务产生的费用。若您调用 BOS 的 HEIC 图像压缩服务，且处理的原图尺寸为 800x600 以下规格，会产生该项费用。该计费项按照调用次数进行计费。
HEIC 图像压缩费用 (800x600 以上规格)	属于图像高级处理服务产生的费用。若您调用 BOS 的 HEIC 图像压缩服务，且处理的原图尺寸为 800x600 以上规格，会产生该项费用。该计费项按照调用次数进行计费。
WebP-M6 图像压缩费用(800x600以下规格)	属于图像高级处理服务产生的费用。若您调用 BOS 的 WebP-M6 图像压缩服务，且处理的原图尺寸为 800x600 以下规格，会产生该项费用。该计费项按照调用次数进行计费。
WebP-M6 图像压缩费用(800x600以上规格)	属于图像高级处理服务产生的费用。若您调用 BOS 的 WebP-M6 图像压缩服务，且处理的原图尺寸为 800x600 以上规格，会产生该项费用。该计费项按照调用次数进行计费。
AVIF 图像压缩费用(800x600以下规格)	属于图像高级处理服务产生的费用。若您调用 BOS 的 AVIF 图像压缩服务，且处理的原图尺寸为 800x600 以下规格，会产生该项费用。该计费项按照调用次数进行计费。
AVIF 图像压缩费用(800x600以上规格)	属于图像高级处理服务产生的费用。若您调用 BOS 的 AVIF 图像压缩服务，且处理的原图尺寸为 800x600 以上规格，会产生该项费用。该计费项按照调用次数进行计费。
智能效果-智能去雾	属于图像智能效果服务产生的费用。若您调用智能去雾，可对浓雾天气下拍摄的细节无法辨认的图像进行去雾处理，还原更清晰真实的图像。该计费项按照调用次数进行计费。
智能场景-风格转换	属于图像智能场景服务产生的费用。若您调用风格转换，可将图像转化成卡通画、铅笔画、彩色铅笔画，或者哥特油画、彩色糖块油画、呐喊油画、神奈川冲浪里油画、奇异油画、薰衣草油画等共计 9 种风格，可用于开展趣味活动，或集成到美图应用中对图像进行风格转换。该计费项按照调用次数进行计费。
智能场景-人像动漫画	属于图像智能场景服务产生的费用。人像动漫画技术运用世界领先的对抗生成网络，结合人脸检测、头发分割、人像分割等技术，为用户量身定制千人千面的二次元动漫形象，并且可通过参数设置，生成戴口罩的二次元动漫人像。该计费项按照调用次数进行计费。
智能场景-天空分割	属于图像智能场景服务产生的费用。天空分割可以智能分割出天空边界位置，输出天空和其余背景的灰度图和二值图，可用于图像二次处理，进行天空替换、抠图等图片编辑场景。该计费项按照调用次数进行计费。
智能场景-物体擦除	属于图像智能场景服务产生的费用。物体擦除可以去除图片中不需要的遮挡物，并用背景内容填充，提高图像质量。该计费项按照调用次数进行计费。
智能场景-拉伸修复	属于图像智能场景服务产生的费用。拉伸修复可以自动识别过度拉伸的图像，将图像内容恢复成正常比例。该计费项按照调用次数进行计费。
智能场景-黑白上色	属于图像智能场景服务产生的费用。黑白上色可以智能识别黑白图像内容并填充色彩，使黑白图像变得鲜活。该计费项按照调用次数进行计费。
智能场景-无损放大	属于图像智能场景服务产生的费用。无损放大可对输入的图片在尽量保持图像质量的条件下，将图像在长宽方向各放大两倍。该计费项按照调用次数进行计费。

图像处理协议

BOS 图像处理目前支持 V1.0 和 V2.0 两个版本的图像处理协议。其中，V2.0 版本所支持的图像处理项目更加丰富，且在命令的调用上与 V1.0 有所不同，因此建议您使用 V2.0 版本进行处理。 V1.0 作为老版本，会在将来停止服务，但当前两个版本均可使用，系统将自动识别并兼容两种协议版本。

在您登录 BOS 控制台后，在 Bucket 的图像处理页签中可以对两个版本切换使用。



图像处理名词解释

概述

在您使用图像处理前，请先了解图像处理涉及到的一些基本概念。

基本概念

图片服务域名

图片服务域名，是指用来访问图片服务的域名。图片服务和普通 BOS 访问使用的是同一个域名，系统会为 Bucket 提供一个外网直接发布的官方域名即`<bucketname>.region.bcebos.com`，用户也可以添加自定义域名。如果 Bucket 开通了 CDN 加速发布，则图片服务同步开通 CDN 加速。如果要开通 Bucket 的 CDN加速发布，可参考 CDN 加速发布。

> **说明：**Console升级后不影响已有域名的图片处理。前期官方提供的bceimg.com域名主要用于客户试用图片服务，升级后将没有存在意义。对于已开通的bceimg.com域名，可继续使用半年，后续会收回。

处理命令

处理命令是指 URL 中用来转换处理图片的一段命令，形式为“缩写命令_参数值”。通过指定的处理命令参数值，BOS 生成并返回另一张转换处理后的图片。如 “w_250”，表示将图片的宽度处理为 250 px。

执行图片某一类处理方式的命令可称为一组命令，如：执行图片缩放功能为一组命令，执行图片裁剪功能为另一组命令。

样式

在对图片进行多重处理时，图片访问的 URL 会变得复杂冗长，非常不利于管理和阅读。为了让图片 URL 变得更简洁优雅，或是您想达到隐藏处理参数、保护原图的目的时，您可以将一组图像处理命令添加为一个图片样式，并对该样式进行自定义命名。创建样式后，您只可以直接调用样式名称来达到预期的图像处理效果，无需调用复杂冗长的命令名称。

分隔符

分隔符是指 URL 中用来区分一些关键字段的分隔符号。

图片信息

图片 exif 信息记录了数码照片的属性信息和拍摄数据，但并非每一张图片都包含 exif 信息。您可以通过命令获取图片的 exif 信息。

管道

管道用来连接实现多条独立命令的执行。如图像处理 V1.0 协议中的管道处理命令格式为`http://${userdomain}/${objectkey}@${commands 1}|${commands 2}`，执行顺序按管道指定顺序从左至右执行。

图像处理常见问题

本文主要介绍您在使用 BOS 数据处理时可能遇到的一些常见问题及处理方法。

使用限制

图像处理支持处理哪些格式的图片？

- BOS 图像处理支持主流格式的输入和输出。支持的输入的原图格式包括 jpg、png、bmp、webp、heic、gif；支持的输出目标格式包括 jpg、png、bmp、webp、heic、gif。BOS 提供格式转变，即输出与输入原图格式不同的格式图片。其中，heic 和 webp 为高级压缩格式，会按照调用次数收取一定费用。关于高级压缩定价，请参考 BOS 计费项介绍和 BOS 价格。

图像处理对输入原图有什么限制吗？

- BOS 支持输入的图片大小不超过 20 MB，原图宽高不超过30000像素且总像素不超过2亿像素，针对动图，原图宽 x 高 x 帧数不超过2亿像素，对于图片水印，水印图加原图不超过2亿像素；处理结果输出图片宽高不超过9999像素，包括管道和子命令间的中间输出图片。

使用方式

图像处理可以以何种方式进行调用？

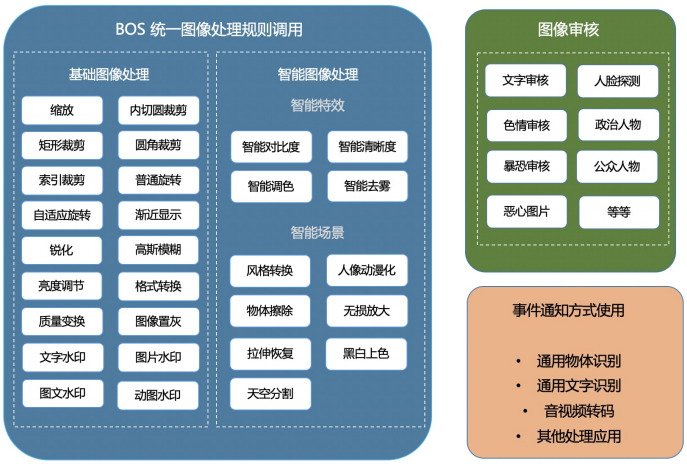
- BOS 支持直接通过 API 接口调用和读取原图，也可以在读写时携带处理命令。若在读文件时携带处理命令，则读取图片为处理后的状态；若在写文件时携带处理命令，则会写入一个新的处理后的图片文件，原图文件依旧保存。同一命令中最多可以指定 4 组处理操作，即 4 个 action，且一个 action 下不可以重复指定同一个参数。为了使用方便，您也可以将一组 action 提前封装为样式进行调用。详细规则请查看图片处理使用规则。

如何创建图像处理样式？

- 您可以通过 BOS 控制台添加或删除样式，单个 Bucket 最多支持创建 400 条图像处理样式。详细操作步骤请查看添加/删除图像样式。

图像处理支持哪些处理内容？

- BOS 图像处理提供丰富的处理方式，整体上包括基础图像处理、智能图像处理、图像审核、OCR识别等能力，但不同能力的使用方式有所不同。基础图像处理和智能图像处理通过 BOS 原生的图片处理使用规则进行使用；图像审核通过单独的界面和接口提供服务，不支持通过 BOS 原生图片处理规则进行使用。若您需要使用通用文字识别或通用物体识别等 OCR 类功能，可以通过事件通知功能触发百度智能云的其他相关产品。



产品价格

图像处理收费吗？

- BOS 基础图像处理、智能图像处理、高级压缩是付费服务，根据调用的图片大小（GB）或调用次数收取费用。图像审核和通过事件通知调用的其他应用，BOS 仅收取 API 的读写费用。关于图像处理的计费，可以查看 BOS 计费项介绍和 BOS 价格。

图像处理指南V1.0

格式转换

概述

本文介绍如何进行格式转换。

格式转换参数

参数名称	缩写命令	类型	取值范围	命令描述	缺省值	是否必选
format	f	string	jpg、png、webp、bmp	指定目标图片格式。响应时对应的Content-Type分别为image/jpeg、image/png、image/webp、image/bmp。	jpg	否

获取图片信息

概述

BOS 支持通过命令用来获取图片信息，如果图片有exif信息则返回包含exif的完整信息，如果图片不包含exif信息，则返回图片的基本信息。

返回的exif信息包含但不限于：

- dateTime、dateTimeOriginal、dateTimeDigitized、format、gpsLatitude、gpsLatitudeRef、gpsLongitude、gpsLongitudeRef、imageHeight、imageWidth、imageSize、make、model、orientation、resolutionX、resolutionY、resolutionUnit

> **说明：**exif信息记录了数码照片的属性信息和拍摄数据，并非每一张图片都包含exif信息。

获取图片信息方式

直接使用 exif 命令即可获取图片信息，无需参数。

示例

- 包含exif信息的图片示例：

https://doc.bce.baidu.com/bce-documentation/BOS/orient.jpg?exif

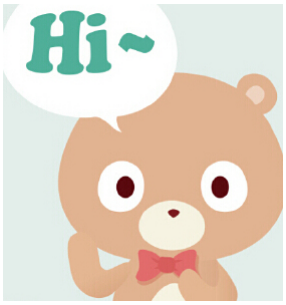


返回示例：


```
{
    "dateTime": {
        "value": "2016-08-31T14:57:07Z"
    },
    "dateTimeDigitized": {
        "value": "2016-08-31T14:57:07Z"
    },
    "dateTimeOriginal": {
        "value": "2016-08-31T14:57:07Z"
    },
    "format": {
        "value": "JPEG"
    },
    "gpsLatitude": {
        "value": "40/1,2/1,3675/100"
    },
    "gpsLatitudeRef": {
        "value": "N"
    },
    "gpsLongitude": {
        "value": "116/1,16/1,257/100"
    },
    "gpsLongitudeRef": {
        "value": "E"
    },
    "imageHeight": {
        "value": "2448"
    },
    "imageSize": {
        "value": "1225883"
    },
    "imageWidth": {
        "value": "3264"
    },
    "make": {
        "value": "Apple"
    },
    "model": {
        "value": "iPhone 6"
    },
    "resolutionUnit": {
        "value": "2"
    },
    "resolutionX": {
        "value": "72/1"
    },
    "resolutionY": {
        "value": "72/1"
    }
}
```


- 不包含exif信息的图片示例：

https://doc.bce.baidu.com/bce-documentation/BOS/image.jpg?exif



返回示例：


```
{
    "format": {
        "value": "JPEG"
    },
    "imageHeight": {
        "value": "296"
    },
    "imageSize": {
        "value": "63506"
    },
    "imageWidth": {
        "value": "279"
    }
}
```


渐近显示

概述

本文介绍渐近显示操作。

渐近显示参数

参数名称	缩写命令	类型	取值范围	命令描述	缺省值	是否必选
display	d	string	baseline progressive	指定图片显示方式。 baseline：标准式，指图片按照从上到下的扫描方式显示； progressive：渐进式，指图片由模糊到清晰渐进显示。 progressive只对jpg、png格式图片有效，若指定其他格式，则默认显示方式仍为baseline。	baseline	否

示例

- 将图片缩略成宽度为150，高度为100，逆时针旋转90度，并渐进显示。

https://doc.bce.baidu.com/bce-documentation/BOS/image.jpg@s_1,w_150,h_100,a_-90,d_progressive



使用规则

概述

图像处理 V1.0 为 BOS 旧版图像处理协议，当前 BOS 新增的功能如自适应格式，图像高级压缩等仅支持通过 V2.0 协议使用。

规则和限制

- 支持的原图格式：jpg、png、bmp、webp、gif、tiff。
- 支持的目标图格式：jpg、png、bmp、webp。
- 支持图片大小为100MB，且图片长和宽不大于4096 px。
- 一个 Bucket 绑定的域名数（包括图片域名）不能超过 200 个，其中图片域名不超过1个。
- 对于同一个域名，只支持一种绑定方式，在普通域名绑定或图片域名绑定中选择一种，不支持两种方式同时绑定。
- 对于同一组处理命令的参数处理采用无序原则，如 @w_250,h_380,q_90 等同于 @h_380,w_250,q_90。
- 如果获取的Object名称本身带有@符号时，其后字符全部按照处理命令对待，用户需要提前替换Object 名称中的@符号。
- 最大支持4个管道。
- Authorization 的签名计算与普通Object处理方式类似，但必须将带参数的图片名称作为一个完整的 URL Path 用于签名。如带水印的图片签名计算的 URL 应该为

```
http://test.gz.bcebos.com/IMG_5974.JPG@!shuiyin1?authorization=...
```

图像访问规则

您可以通三种方式对图像进行访问。BOS 支持直接访问和显示原图，也可以通过命令访问图像，还可以通过创建样式来进行访问。

直接访问和显示原图

直接显示原图与普通的域名绑定访问方式一样，URL形式为：`http://${userdomain}/${objectkey}`

- `userdomain`为用户绑定的域名。
- `objectkey`为存储在BOS上的原图片名称。

样例：`http://www.myhost.com/pic.jpg`

通过命令访问

URL 形式为：`http://${userdomain}/${objectkey}@${commands}`

- `userdomain`为用户绑定的域名。
- `objectkey`为存储在BOS上的原图片名称。
- `commands`为处理命令。

样例：`http://bucket-A.bj.bcebos.com/sample.jpg@w_200,q_80`

分隔符

图像处理 V1.0 协议包括如下分隔符：

分隔符名称	分隔符	两侧顺序	说明
处理分隔符	@	固定顺序	Object与处理命令之间的分隔符
参数分隔符	_	固定顺序	参数缩略命令与参数值之间的分隔符
命令分隔符	,	无关	多项处理参数项之间的分隔符
管道分隔符	|	有关	两组处理命令之间分隔符，管道的参见下述“管道”

管道

如果一个图片，需要多重处理，如先裁剪、后缩略等，则多组操作命令之间，需要以管道"|"连接，执行顺序按管道指定顺序从左至右执行。

如下示例为一个管道，执行顺序为从左到右，左侧命令处理结果将会作为右侧命令的输入。

```
http://${userdomain}/${objectkey}@${cropcommands}|${scalecommands}
```

命令示例：

```
（1）如对图片先裁剪，再缩放。先以左上角横坐标为10，纵坐标为20进行裁剪，裁剪宽度指定为600，高度为300，再按宽200，高200进行拉伸缩放，并指定目标图片的绝对质量为60，目标图片格式为png。
https://doc.bce.baidu.com/bce-documentation/BOS/image.jpg@c_1,x_10,y_20,w_600,h_300|s_1,w_200,h_200,q_60,f_png
（2）如对图片先缩放、旋转，再裁剪。先按宽200的比例等比缩放并顺时针旋转90度，再以左上角横坐标为50，纵坐标为100为起点进行裁剪，裁剪宽度和高度默认为图片宽度和高度。
https://doc.bce.baidu.com/bce-documentation/BOS/image.jpg@w_200,a_30|c_1,x_50,y_100>
（3）如进行图文混合水印：水印图片为bce-documentation/BOS/bce.jpg，位于原图左下角，距离锚点水平距离为,15，垂直距离为20，透明度为50；文字内容为“hello world”，位于正上方，垂直距离为100，字体颜色为淡紫色。
https://doc.bce.baidu.com/bce-documentation/BOS/image.jpg@wm_1,k_YmNlLWRvY3VtZW50YXRpb24vQk9TL2JjZS5qcGc=,g_7,x_15,y_20,o_50|wm_2,t_aGVsbG8gd29ybGQ=,g_2,y_100,fc_aa88aa
```

- 图文混合水印：水印图片为bce-documentation/BOS/bce.jpg，位于原图左下角，距离锚点水平距离为,15，垂直距离为20，透明度为50；文字内容为“hello world”，位于正上方，垂直距离为100，字体颜色为淡紫色。

  https://doc.bce.baidu.com/bce-documentation/BOS/image.jpg@wm_1,k_YmNlLWRvY3VtZW50YXRpb24vQk9TL2JjZS5qcGc=,g_7,x_15,y_20,o_50|wm_2,t_aGVsbG8gd29ybGQ=,g_2,y_100,fc_aa88aa

通过样式访问

在对图片进行多重处理时，图片访问的 URL 会变得复杂冗长，非常不利于管理和阅读。为了让图片 URL 变得更简洁优雅，或是您想达到隐藏处理参数、保护原图的目的时，您可以使用图片样式功能。

图片样式是指您可以将一些常用的图片处理操作保存为一个别名，即一个样式。这样，一组复杂的操作，利用样式，可以通过一个简洁的URL访问即可达到相同效果。

图像处理 V1.0 的样式形式为：

```
http://${userdomain}/${objectkey}@!${stylename}
```

其中：

- **userdomain**：为用户绑定的域名。
- **objectkey**：为存储在BOS上的原图片名称。
- **stylename**：为样式名称

如您创建了一个名称为 mystylename 的样式，您可以以如下方式进行访问：

```
http://bucket-A.bj.bcebos.com/sample.jpg@!mystylename
```

添加和删除样式

添加样式

您可以通过控制台进行图片样式的定义，同时还可以在定义的同时实时进行效果预览。

1. 登录BOS控制台，进入您已开通图片服务的Bucket。点击左侧导航“图片服务”后，点击右侧页面中的“样式管理”。
2. 点击“添加样式”，进入添加样式表单页面。您可根据需要，设定该样式的图片处理参数，同时您可在右侧的预览区域实时进行效果预览。
   图片样式的设置方式支持“基础设置”和“高级设置”两种。
   - 基础设置，您需要在页面上对下方的每一项处理参数进行手动设定。
   - 高级设置，您可直接在框中直接输入图片处理命令参数。

   创建好样式后，点击“保存”。

   缩放说明
   - 限定宽度，高度等比例。
     您需要输入宽度，缩略时以宽度为基准，将高度等比例缩小。若输入宽度大于原图宽度，则返回原图。
     例如：原图宽高尺寸为400 × 320，限定宽度200，高度等比例，生成缩略图尺寸为200×160；限定宽度500，高度等比例，生成缩略图尺寸为400×320。
   - 限定高度，宽度等比例。
     您需要输入高度，缩略时以高度为基准，将宽度等比例缩小。若输入高度大于原图高度，则返回原图。
     例如：原图宽高尺寸为400 × 320，限定高度160，宽度等比例，生成缩略图尺寸为200×160；限定高度500，宽度等比例，生成缩略图尺寸为400×320。
   - 限定宽度和高度。
     您需要输入宽度和高度，图片会保持原图比例进行缩略，直至宽、高都满足设定值，不裁剪。
     例如：原图宽高尺寸为400 × 320，限定宽度为100，高度为100，生成缩略图尺寸为100 × 80；限定宽度为500，高度为500，生成缩略图尺寸为400×320。
   - 固定宽度和高度。
     您需要输入宽度和高度，图片会保持原图比例先进行缩略，超出设定宽、高部分进行居中裁剪。
     例如：原图宽高尺寸为400 × 320 ，固定宽度100和高度100（缩小较大图片，并裁剪图片），生成缩略图尺寸为100×100；固定宽度500和高度500（放大较小图片，并裁剪图片），生成缩略图尺寸为500×500。
3. 保存样式后，在列表中可以看到您已经添加的样式信息。您可以点击“查看命令”，查看样式对应的参数命令。



删除样式

在样式管理列表中，点击“操作”列的“删除”，可删除样式。



水印

概述

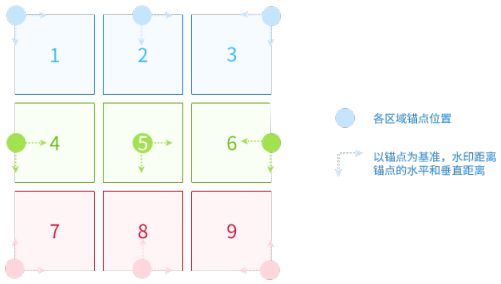
本文介绍如何设置水印。图片水印是指在图片上方插入另一张图片或者文字作为水印，是一种数字信息的保护手段。目前 BOS 图像处理 V1.0 协议支持图片水印和文字水印两种水印方式。BOS 将您的原始图片分成如下图所示 9 个区域，并按照图示位置进行编号，每个区域均包含锚点。

确定水印在原图上的最终位置，您需要确定如下两个因素：

- 锚点位置；

- 基于锚点位置，指定在水平和垂直方向上的距离。

锚点位置如图所示：



图片水印参数

参数名称	缩写命令	类型	取值范围	命令描述	缺省值	是否必选
watermark	wm	unsigned	0，1，2	指定水印方式。wm_0表示关闭水印功能；wm_1表示图片水印；wm_2表示文字水印。	-	是
key	k	string	-	指定水印图片存储在BOS上的Object名称，即ObjectKey。如Object存储在文件夹下，则需要包含文件夹路径。该值要求为Base64编码后的值。	-	是
gravity	g	unsigned int	1～9	指定水印的锚点位置。图片锚点的含义请参考图片水印命令。	9	否
gravityX	x	int	-4096～4096	指定水印基于锚点的水平方向距	10	否

				离。		
gravityY	y	int	-4096～4096	指定水印基于锚点的垂直方向距离。	10	否
angle	a	int	-360～360	指定水印图片的旋转角度。正数表示顺时针旋转，如a_60；负数表示逆时针旋转，如a_-30。	0	否
opacity	o	unsigned int	1～100	指定水印图片的透明度。取值为100时，表示与原始图片透明度一致。	100	否

说明

- 水印图片要与原始图片存储在BOS的同一个Bucket下。

文字水印参数

参数名称	缩写命令	类型	取值范围	命令描述	缺省值	是否必选
watermark	wm	unsigned	0，1，2	指定水印方式。wm_0表示关闭水印功能；wm_1表示图片水印；wm_2表示文字水印。	-	是
text	t	string	-	水印文字内容，需要进行Base64编码，且编码后最多不能超过80个字符。	-	是
gravity	g	unsigned int	1～9	指定水印的锚点位置。图片锚点的含义请参考图片水印	9	否

				命令。		
gravityX	x	int	-4096～4096	指定水印基于锚点的水平方向距离。	10	否
gravityY	y	int	-4096～4096	指定水印基于锚点的垂直方向距离。	10	否
angle	a	int	-360～360	指定水印图片的旋转角度。正数表示顺时针旋转，如a_60；负数表示逆时针旋转，如a_-30。	0	否
fontSize	sz	unsigned int	1～1024	指定水印字体大小，单位是 px。	30	否
fontColor	fc	string	-	指定字体颜色，为一个8位的16进制字符串，其中每2位用来表示RGBA中的一项。前6位分别表示RGB色，如ffffff；最后2位表示Alpha通道，可省略，省略时表示不透明。例如：000000ff表示黑色不透明字体。	000000	否
fontFamily	ff	string	-	指定字体的字体类型，取值要求进行Base64编码。已支持的字体类型请参见	宋体	否

				下方列表。		
fontStyle	fs	string	normal（常规） italic（斜体） bold（粗体）	指定字体样式。	normal	否

已支持的字体类型：

参数取值	字体类型
MicrosoftYaHei	微软雅黑
FangSong	仿宋
SimHei	黑体
KaiTi	楷体
SimSun	宋体
FZBeiWeiKaiShuJianTi	方正北魏楷书简体
FZBoYaSongJianTi	方正博雅宋简体
FZCuHeiSongJianTi	方正粗黑宋简体
FZLanTingZhongHeiJianTi	方正兰亭中黑简体
FZZhengHei	方正正黑
AvantGarde	AvantGarde
Bookman	Bookman
Courier	Courier
Helvetica	Helvetica
HelveticaNarrow	HelveticaNarrow
NewCenturySchlbk	NewCenturySchlbk
Palatino	Palatino
Times	Times
Symbol	Symbol

> **说明：**
>
> - 只有当fontFamily支持fontStyle效果时，才会渲染文字效果。
> - 若指定的字体样式不在当前已支持的fontFamily列表中，则显示效果为默认字体宋体。

图文混合水印

图文混合水印是指将图片水印和文字水印同时打在原始图片上。由于图片水印和文字水印为两组独立的命令，因此图文混合水印需要用管道实现。

示例

图片水印示例

原图为image.jpg，水印图片为bce-documentation/BOS/bce.jpg(Base64编码后为"YmNlLWRvY3VtZW50YXRpb24vQk9TL2JjZS5qcGc=")。

- 在右下角加上图片水印，指定垂直距离为10，透明度为60。

https://doc.bce.baidu.com/bce-documentation/BOS/image.jpg@wm_1,k_YmNlLWRvY3VtZW50YXRpb24vQk9TL2JjZS5qcGc=,y_10,o_60



- 在左下角加上水印，水平和垂直距离取缺省值，透明度为50，并顺时针旋转90度。

https://doc.bce.baidu.com/bce-documentation/BOS/image.jpg@wm_1,k_YmNlLWRvY3VtZW50YXRpb24vQk9TL2JjZS5qcGc=,g_7,o_50,a_90



文字水印示例

- 最简单的文字水印，文字内容为"hello world"。

https://doc.bce.baidu.com/bce-documentation/BOS/image.jpg@wm_2,t_aGVsbG8gd29ybGQ=



- 文字内容为“hello world”，字体类型为微软雅黑，字体颜色和字体大小取默认值，字体样式为粗体，水印位置在右中，距离锚点水平距离20，垂直距离20，并顺时针旋转30度。

https://doc.bce.baidu.com/bce-documentation/BOS/image.jpg@wm_2,t_aGVsbG8gd29ybGQ=,ff_TWljcm9zb2ZOWWFIZWk=,fs_bold,g_6,x_20,y_20,a_30



图文混合水印

- 最简单的图文混合水印：图片为bce-documentation/BOS/bce.jpg，文字内容为“hello world”，其余参数均取默认值。

https://doc.bce.baidu.com/bce-documentation/BOS/image.jpg@wm_1,k_YmNlLWRvY3VtZW50YXRpb24vQk9TL2JjZS5qcGc=|wm_2,t_aGVsbG8gd29ybGQ=



- 图文混合水印：水印图片为bce-documentation/BOS/bce.jpg，位于原图左下角，距离锚点水平距离为,15，垂直距离为20，透明度为50；文字内容为“hello world”，位于正上方，垂直距离为100，字体颜色为淡紫色。

https://doc.bce.baidu.com/bce-documentation/BOS/image.jpg@wm_1,k_YmNlLWRvY3VtZW50YXRpb24vQk9TL2JjZS5qcGc=,g_7,x_15,y_20,o_50|wm_2,t_aGVsbG8gd29ybGQ=,g_2,y_100,fc_aa88aa



图像裁剪

概述

本文介绍图片裁剪操作。

图片裁剪参数

参数名称	缩写命令	类型	取值范围	命令描述	缺省值	是否必选
crop	c	unsigned int	0,1	指定图片是否执行裁剪功能。c_1表示对图片进行裁剪；c_0表示不对图片进行裁剪，其他裁剪参数无效。	-	否
offsetX	x	unsigned int	0～4096	指定图片裁剪左上角起点x坐标。	0	否
offsetY	y	unsigned int	0～4096	指定图片裁剪左上角起点y坐标。	0	否
width	w	unsigned int	1～4096	指定图片裁剪的宽度。如果指定的宽度超过了图片的宽度，则以图片宽度为准裁剪。单位为px。	原始图片宽度	否
height	h	unsigned int	1～4096	指定图片裁剪的高度。如果指定的高度超过了图片的高度，则以图片高度为准裁剪。单位为px。	原始图片高度	否

示例

- 以图片左上角横坐标为100，纵坐标为150为起点进行图片裁剪，裁剪宽度和高度默认为原始图片宽度和高度。

https://doc.bce.baidu.com/bce-documentation/BOS/image.jpg@c_1,x_100,y_150



- 以图片左上角横、纵坐标均为0为起点进行图片裁剪，裁剪宽度指定为600，裁剪高度指定为600。

https://doc.bce.baidu.com/bce-documentation/BOS/image.jpg@c_1,w_600,h_600



- 以图片左上角横坐标为100，纵坐标为150为起点进行图片裁剪，裁剪宽度指定为400，裁剪高度指定为200。

https://doc.bce.baidu.com/bce-documentation/BOS/image.jpg@c_1,x_100,y_150,w_400,h_200



图像缩略

概述

本文介绍如何对图像进行缩略处理。

图像缩略参数

命令参数

参数名称	缩写命令	类型	取值范围	命令描述	缺省值	是否必选
scale	s	unsigned int	0,1,2	指定缩放方式。s_0表示无裁剪，等比例缩放；s_1表示拉伸缩放；s_2表示等比例居中裁剪缩放。	0	否
width	w	unsigned int	1～4096	指定目标缩略图的最大宽度单位为px。	-	s_1,s_2时，必选
height	h	unsigned int	1～4096	指定目标缩略图的最大高度单位为px。	-	s_1,s_2时，必选
limit	l	unsigned int	0,1	指定是否限制图片缩放的尺寸大小：l_0表示不限制图片缩放的尺寸；l_1表示限制图片缩放尺寸（长和宽）不可超过原图大小。	0	否

> **注意事项：**
>
> - 当 s_0 时，即无裁剪等比缩放时：
> - w 和 h 可以只指定其中一个，图片按指定的参数比例进行缩放。例如，原图尺寸为 100 × 200，指定参数为 w_50 后，图片处理的尺寸为 50 × 100；若指定 h_50 后，图片处理的尺寸为 25 × 50;
> - w 和 h 同时指定时，系统会取“目标宽度/原图宽度”与“目标高度/原图高度”中值小的参数为标准进行等比例缩放。例如：原图尺寸为 400 × 500 ，指定参数为 w_100, h_200 时，由于“100/400”< “200/500”，则系统按照w_100的比例进行缩放，目标尺寸为100 × 125。；
> - 当s_1时，必须同时指定w和h的值;
> - 当s_2时，必须同时指定w和h的值。指定w和h后，图片会按照“目标宽度/原图宽度”与“目标高度/原图高度”中值大的参数为标准进行等比例缩放，并对比值较大的另一边进行居中裁剪。例如：原图尺寸为500 × 1000 ，指定参数为 s_2，w_250，h_600 时，由于“250/500” < “600/1000”，会以目标高度 600 为基准进行等比缩放，同时将宽度按照居中裁剪为250。

示例

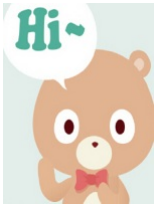
- 将图片缩略成高度为 200，宽度按比例处理。

https://doc.bce.baidu.com/bce-documentation/BOS/image.jpg@h_200



- 将图片缩略成宽度为150，高度为200，目标图片绝对质量为60，目标图片格式为png。

https://doc.bce.baidu.com/bce-documentation/BOS/image.jpg@s_1,w_150,h_200,q_60,f_png



- 将图片进行等比例居中裁剪缩放，缩放宽度为80，高度为100。

https://doc.bce.baidu.com/bce-documentation/BOS/image.jpg@s_2,w_80,h_100



- 将图片缩略成宽度为150，高度为100，逆时针旋转90度，并渐进显示。

https://doc.bce.baidu.com/bce-documentation/BOS/image.jpg@s_1,w_150,h_100,a_-90,d_progressive



图像旋转

概述

本文介绍图像旋转操作。

图像旋转参数

参数名称	缩写命令	类型	取值范围	命令描述	缺省值	是否必选
angle	a	int	-360～360	指定图片旋转方式。正数表示顺时针旋转，如a_60；负数表示逆时针旋转，如a_-30。	0	否
orientation	o	int	0,1,2	根据exif信息进行自动旋转。o_0：按原图默认方向，不自动旋转；o_1：先旋转，然后再缩略；o_2：先缩略，然后再旋转。		

示例

- 将图片顺时针旋转30度。

https://doc.bce.baidu.com/bce-documentation/BOS/image.jpg@a_30



质量变换

概述 本文介绍图像质量变换。

质量变换参数

参数名称	缩写命令	类型	取值范围	命令描述	缺省值	是否必选
quality	q	unsigned int	1～100	指定目标图片的绝对质量，只对jpg/jpeg、png、webp格式有效。当指定q命令的值大于原图quality的值时，则默认使用原图quality的值。	原图quality值。	否

图像处理指南V2.0

格式转换

概述

图片格式转换。

格式转换参数

action名称：`format`

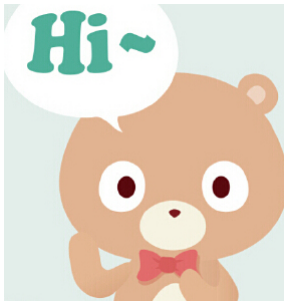
参数	取值	说明
f	jpg、png、bmp、webp、heic、gif、avif、auto	指定目标图片的图片格式

如果值为`auto`则根据请求Header的`Accept`字段自动选择最优格式，当前支持自适应为`webp`，如果无法匹配则保持原图格式。响应头中会按照HTTP规范返回`Vary: Accept`。

需要特别注意的是，使用`auto`时，单个链接会有多个缓存副本，如果使用百度智能云CDN，则需要CDN开启多副本缓存功能，如果没有开启此功能，CDN会缓存第一次请求的结果，当第一次请求返回的是`webp`格式时，会导致safari、ie等浏览器看不到图片。2020/01/01之后创建CDN域名默认都开启了此功能，之前创建的域名需要提工单开启。

示例

- 格式自适应，在Chrome等支持webp的浏览器下返回为webp格式，不支持webp的浏览器中返回为原图格式。
  https://doc.bce.baidu.com/bce-documentation/BOS/image.jpg?x-bce-process=image/format,f_auto



灰度变换

概述

将图片转换为灰图。

灰度变换参数

命令名称：`grayscale`。

示例

- 将图片转换为灰图
  https://doc.bce.baidu.com/bce-documentation/BOS/image.jpg?x-bce-process=image/grayscale



获取图片信息

概述

您可以通过命令来获取图片信息。如果图片有 exif 信息，则返回包含 exif 的完整信息，如果图片不包含 exif 信息，则返回图片的基本信息。

> **说明：**exif信息记录了数码照片的属性信息和拍摄数据，并非每一张图片都包含exif信息。

返回的exif信息包含但不限于：

- dateTime、dateTimeOriginal、dateTimeDigitized、format、gpsLatitude、gpsLatitudeRef、gpsLongitude、gpsLongitudeRef、imageHeight、imageWidth、imageSize、make、model、orientation、resolutionX、resolutionY、resolutionUnit

获取图片信息参数

action名称：`info`

此命令没有参数，以 x-bce-process=image/info 形式访问即可。

渐近显示

概述

BOS 支持指定图片显示方式。

当网络环境较差或图片较大时，图片在网页上有两种显示方式：

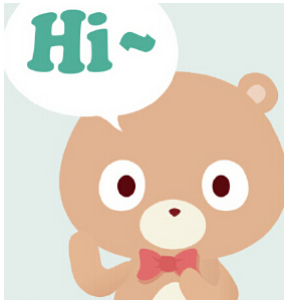
- 标准显示：图片会按照从上到下的顺序一行一行地加载显示。
- 渐进显示：先显示整个图片的模糊轮廓，然后逐渐加载直至显示完整的图片。

渐近显示参数

action名称：`interlace`

参数	取值	说明
i	baseline，progressive	baseline：标准式，指图片按照从上到下的扫描方式显示；progressive：渐进式 ，指图片由模糊到清晰渐进显示。progressive 只对jpg、png格式图片有效，若指定其他格式，则默认显示方式仍为baseline。

示例

- 图片指定渐进显示。
  https://doc.bce.baidu.com/bce-documentation/BOS/image.jpg?x-bce-process=image/interlace,i_progressive
  

去除元信息

概述

您可以通过命令来可去除图片元信息，包括 exif 信息。

> **说明：**exif信息记录了数码照片的属性信息和拍摄数据，并非每一张图片都包含exif信息。

返回的exif信息包含但不限于：

- dateTime
- dateTimeOriginal
- dateTimeDigitized
- format
- gpsLatitude
- gpsLatitudeRef
- gpsLongitude
- gpsLongitudeRef
- imageHeight
- imageWidth
- imageSize
- make
- model
- orientation

- resolutionX
- resolutionY
- resolutionUnit

去除图片元信息参数

action名称：`strip`

此命令没有参数，以 x-bce-process=image/strip 形式访问即可。

使用规则

概述

BOS 图片处理 V2.0 版本，是 BOS 全新的处理协议。相较 V1.0 版本，V2.0 版本支持更多图片处理功能，并采用全新的命令方式。您仍然可以通过 BOS 图片处理 V1.0 版本方式来访问 BOS 图片处理服务，但今后 V1.0 版本功能将不再更新。

在使用 BOS 图像处理功能前，您需要将原始图片上传到 BOS。BOS 的每个 Bucket 都默认开通图片处理的功能，您无需额外开通，也无需使用特定域名访问图片处理服务。当访问图片 URL 时，在 Querystring 中通过**x-bce-process**参数指定图片处理命令，即可触发 BOS 图片处理功能。

> **说明：**
>
> - 为了提升图片的安全性，BOS 也支持对指定 Bucket 内的图片开启原图保护。即开启原图保护后，如果图片获取者未验证被授权的签名，则只能通过样式方式访问处理后的图片，直接访问原图或通过处理参数访问的方式会被拒绝。目前支持在控制台对原图保护功能进行开启或关闭，也支持通过 API 进行相关操作，并可通过 resource 字段指定该功能生效范围。

规则与限制

- 支持的原图格式：jpg、png、bmp、webp、heic、gif、avif；
- 支持的目标格式：jpg、png、bmp、webp、heic、gif、avif；
- 处理文件大小不超过20M；
- 原图宽高不超过30000像素且总像素不超过2亿像素，针对动图，原图宽 x 高 x 帧数不超过2亿像素，对于图片水印，水印图加原图不超过2亿像素；
- 处理结果输出图片宽高不超过9999像素，包括管道和子命令间的中间输出图片；
- 最多可以同时指定16组操作，即16个action；
- 一个action下不可以重复指定同一个参数；
- 样式名称支持数字，字母和下划线组合，且首位不能为下划线，最长 64 个字符；
- 动图处理结果不超过 20 帧；

使用方式

BOS 支持通过命令和样式两种方式访问图片服务。

1. 命令方式

图片处理命令形式为：

```
x-bce-process=image/${action},${key}_${value},${key}_${value}/${action},${key}_${value}
```

其中：

- **image**：BOS支持多种数据处理形式，均使用x-bce-process参数触发，当使用图片处理时，需要指定此处名字为image。
- **action**：BOS支持多种图片处理命令，每种图片处理命令称作一种action，比如缩放resize，裁剪crop等。
- **key**：每种action支持多种具体的处理参数，比如缩放宽度w,高度h等。
- **value**：处理参数取值。
- **分隔符**：

分隔符名称	分隔符	两侧顺序	说明
处理分隔符	/	有关	多个action之间的分隔符，前后action顺序执行
参数分隔符	,	无关	多处理参数项之间的分隔符
值分隔符	_	固定顺序	参数名与参数值之间的分隔符

如对图片进行等比缩放图片，且缩放后的图片宽最大200，高最大100，亮度-5，格式转化为webp，则对应命令为：

```
http://bucket-A.bj.bcebos.com/sample.jpg?x-bce-process=image/resize,m_lfit,w_200,h_100/bright,b_-10/format,f_webp
```

2.样式方式

您可以通过控制台将某一项或多项图片处理命令保存为样式，并为其创建创建名称。之后，您可以通过样式达到图像处理的效果，简化原本一系列复杂命令操作和参数，只需通过很简短的 URL 即可实现相同的效果。

通过样式的访问语法：

```
http://${domain}/${objectkey}?x-bce-process=style/${stylename}
```

详细说明：

- **style**：当使用图片样式访问时，需要固定指定为style。
- **stylename**：访问的样式style的名称。

例如：您创建一个名称为 mystylename 的样式，可实现宽度为 200 px，高度为 300 px 的等比例最大缩放，并同步支持自适应旋转（命令为：`pxresize,m_lfit,limit_0,w_200,h_300/auto-orient,o_1`），您可以直接调用 mystylename 的名称来访问图像处理：

```
http://bucket-A.bj.bcebos.com/sample.jpg?x-bce-process=style/mystylename
```

控制台添加/删除样式

您可以通过 BOS 控制台添加或删除样式。

1. 登录百度智能云官网

登录百度智能云官网。

- 若您未注册账号，须先注册账户。
- 若您已注册，可直接登录。

2. 进入 BOS 控制台

在页面左侧导航栏中选择 "存储和 CDN > 对象存储 BOS" 进入 BOS 控制台。

3. 选择 Bucket 并进入图像处理页面

在左侧的 Bucket 列表中选择 Bucket 名称并进入图像处理页面。在该页面中，您可以创建新的图像处理样式，也可以在列表中查看已创建的图像处理样式。

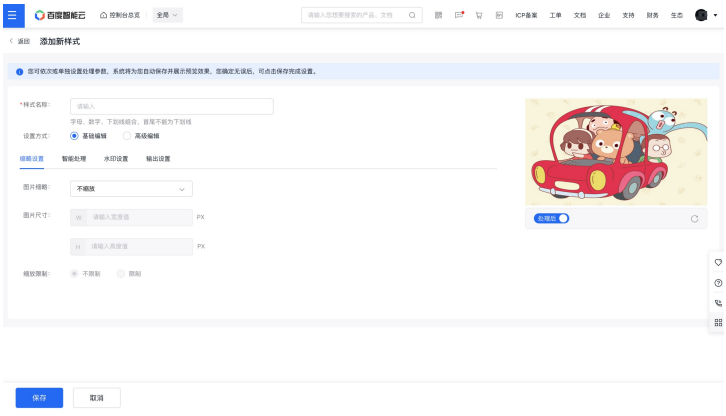


4. 添加样式

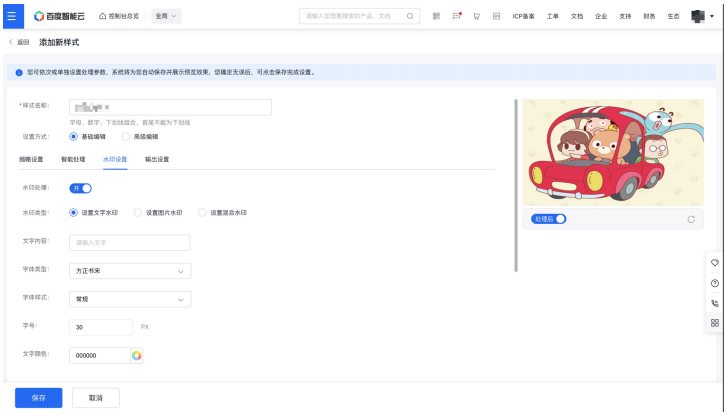
点击添加样式按钮，进入添加样式页面。关于图像处理协议 V2.0 版本的样式设置方法，你可以查看图像处理协议 2.0 ；关于图像处理协议 V1.0 版本的样式设置方法，您可以查看图像处理协议 1.0。

在控制台中，您可以通过基础编辑方式和高级编辑方式对样式进行编辑。其中基础编辑方式是通过图形化界面对样式进行编辑，而高级编辑方式需要您输入相应的命令语句进行设置。

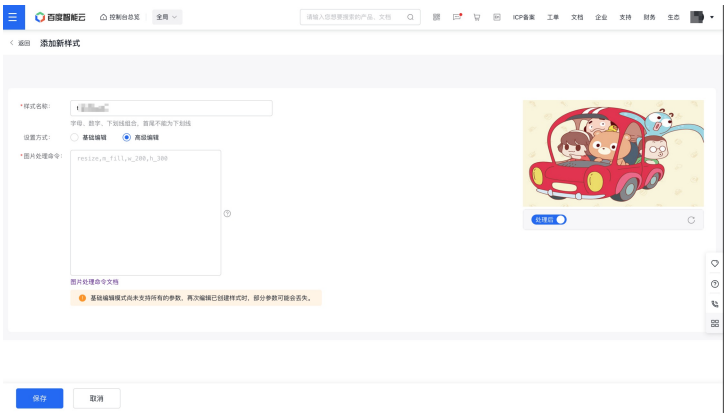
在基础编辑中，您可以进行样式名称设置和缩略设置。



也可以对该图像进行输出设置和水印设置。



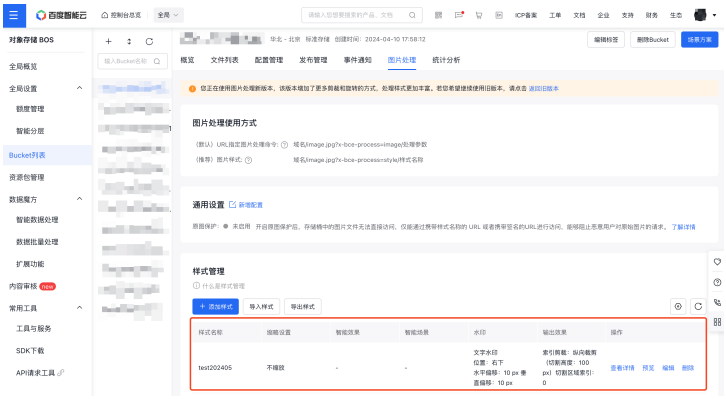
在高级编辑中，您可以直接通过命令行设置。图片服务的命令行具体操作请参见图片处理服务 2.0和图像处理协议 1.0。



5. 添加列表

样式添加完成后会在图像处理的样式列表中显示，您可以通过操作列的"查看命令"查看该样式对应的命令，也可以预览样式，或编辑删除该样式。

> **注意：**
>
> - 图像处理协议 2.0 生成的样式，仅展示在新版协议的列表中；图像处理协议 1.0 生成的样式，仅展示在旧版协议的列表中，两个协议下的样式不会共同展示，请在页面上方切换版本进行查看。



> **说明：**
>
> - BOS 图片处理命令分新旧两种版本，因此一个图片处理样式所对应的实际图片处理命令可以是新版或旧版。
> - 本文 `x-bce-process=style/${stylename}`的访问方式只能访问新版命令所对应的样式，旧版本命令所对应的样式仍需通过旧版本@!方式访问。

图片处理结果返回

BOS 会将处理完的图片信息通过 HTTP Header 的方式返回。
除标准 HTTP Header 中已有的信息，其他信息用`x-bce-image-info`头返回，格式为`key1=value;key2=value2;...`

返回内容参考如下：

信息	响应头	值
format	Content-Type: image/jpeg	jpeg
size	Content-Length: 12608	12608
md5	Content-MD5: IsfLjdB8Uu+kwmuFKJ1VZw==	IsfLjdB8Uu+kwmuFKJ1VZw==
width	x-bce-image-info:width=200	200
height	x-bce-image-info:height=113	113

图片对比度

概述

对比度是指一幅图像中明暗区域最亮的白和最暗的黑之间不同亮度层级的测量，即指一幅图像灰度反差的大小。

亮度参数

action名称：`contrast`

参数	取值	说明
c	-100 - 100	取值＜0：降低图片对比度。取值=0：维持原图对比度。取值＞0：提高图片对比度。

示例

- 对比度降低50
  https://doc.bce.baidu.com/bce-documentation/BOS/image.jpg?x-bce-process=image/contrast,c_-50



- 对比度提高10

  https://doc.bce.baidu.com/bce-documentation/BOS/image.jpg?x-bce-process=image/contrast,c_10



图片亮度

概述

本文介绍图片亮度操作。

亮度参数

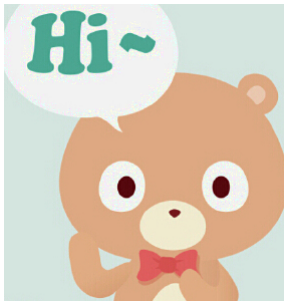
action名称：`bright`

参数	取值	说明
b	-100 - 100	设置图片亮度。当取值小于0时，图片变暗；当取值等于0时，图片亮度不变；当取值大于0时，图片变亮。

示例

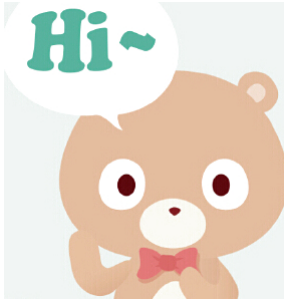
- 将图片以参数-5进行变暗处理

  https://doc.bce.baidu.com/bce-documentation/BOS/image.jpg?x-bce-process=image/bright,b_-5



- 将图片以参数5进行变亮处理

  https://doc.bce.baidu.com/bce-documentation/BOS/image.jpg?x-bce-process=image/bright,b_5



图片模糊

概述

本文介绍图片模糊操作。

模糊参数

action名称：`blur`

参数	取值	说明
r	1-50	模糊的半径
s	1-50	模糊的标准差

示例

- 将图片以半径为2 标准差为50进行模糊处理
  https://doc.bce.baidu.com/bce-documentation/BOS/image.jpg?x-bce-process=image/blur,r_2,s_50



图片锐化

概述

本文介绍图片锐化操作。

锐化参数

action名称：`sharpen`

参数	取值	说明
r	1-50	锐化的半径
s	1-50	锐化的标准差

示例

- 将图片以半径为2 标准差为50进行锐化处理
  https://doc.bce.baidu.com/bce-documentation/BOS/image.jpg?x-bce-process=image/sharpen,r_2,s_50



图像裁剪

内切圆裁剪

概述

本文介绍以图片中心为圆心，进行内切圆裁剪的操作。

内切圆裁剪参数

action名称：`circle`

参数	取值	说明
r	0-图片最小边大小的一半	以图片中心为圆心，从图片取出的半径为r的圆形区域，r如果超过最小边大小的一半，默认取原圆的最大内切圆。

示例

- 以100为半径，进行内切圆裁剪.
  https://doc.bce.baidu.com/bce-documentation/BOS/image.jpg?x-bce-process=image/circle,r_100



普通裁剪

概述

本文介绍对图片进行矩形裁剪操作。

普通裁剪参数

action名称：`crop`

参数	取值	说明
x	0-图片宽度	裁剪左上角起点相对原点的x坐标，非必选，默认0。
y	0-图片高度	裁剪左上角起点相对原点的y坐标，非必选，默认0。
w	1-4096	裁剪后图片的宽度，如果指定的宽度超过了图片的宽度，则以图片宽度为准裁剪。单位为px。非必选，默认值原始图片宽度。
h	1-4096	裁剪后图片的高度，如果指定的高度超过了图片的高度，则以图片高度为准裁剪。单位为px。非必选，默认值原始图片高度。

示例

- 对图片以（10,10）坐标为起点，裁剪200*200.
  https://doc.bce.baidu.com/bce-documentation/BOS/image.jpg?x-bce-process=image/crop,x_10,y_10,w_200,h_200



索引裁剪

概述

本文介绍图片索引裁剪操作。

索引裁剪参数

action名称：`indexcrop`

参数	取值	说明
x	1-4096	x与y不能同时指定，且必须指定其一。纵向切割，指定切割宽度。如果超过图片宽度，则默认为图片宽度。
y	1-4096	x与y不能同时指定，且必须指定其一。横向切割，指定切割高度。如果超过图片高度，则默认为图片高度。
i	0-65536	指定切割后的第i块切割区域。如果超过切割区域数，则返回原图。不指定默认为0。

示例

- 将图片以高度50横向切割，选择切割后的第1块和第2块。
  https://doc.bce.baidu.com/bce-documentation/BOS/image.jpg?x-bce-process=image/indexcrop,y_50,i_0



https://doc.bce.baidu.com/bce-documentation/BOS/image.jpg?x-bce-process=image/indexcrop,y_50,i_1



- 将图片以宽度100纵向切割，选择切割后的第1块和第2块。
  https://doc.bce.baidu.com/bce-documentation/BOS/image.jpg?x-bce-process=image/indexcrop,x_100,i_0



https://doc.bce.baidu.com/bce-documentation/BOS/image.jpg?x-bce-process=image/indexcrop,x_100,i_1



圆角矩形裁剪

概述

本文介绍图片圆角矩形裁剪。

圆角矩形裁剪

action名称：`rounded-corners`

参数	取值	说明
r	1-4096	设置圆角的半径。如果指定半径r大于原图宽/高，则默认原图宽/高。

示例

- 以50为半径，进行圆角矩形裁剪。

https://doc.bce.baidu.com/bce-documentation/BOS/image.jpg?x-bce-process=image/rounded-corners,r_50



图像水印

添加图片水印

概述

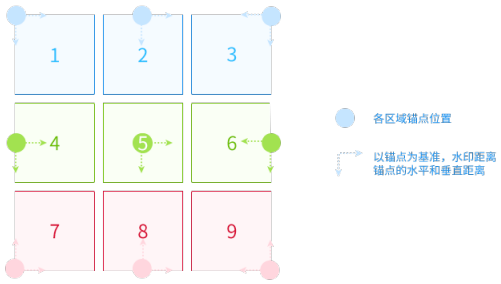
本文介绍对一张图片添加图片水印。

图片水印参数

action名称：`watermark`

图片水印参数	取值	说明
bucket	-	指定水印图片存储在BOS的Bucket名称，非必选，默认当前bucket。非当前bucket则要求该水印图片有公共读权限。
image	-	指定水印图片存储在BOS的Object名称，可以包含预处理命令，需要取url安全的Base64编码后的值。图片水印必选参数。
t	1-100	指定图片水印透明度，取值为100时，表示与原始透明度一致。非必选，默认值100。
g	1-9	指定水印的锚点位置。非必选，默认9。具体含义见下方说明。
x	-4096-4096	指定水印基于锚点的水平方向距离。非必选，默认值10。
y	-4096-4096	指定水印基于锚点的垂直方向距离。非必选，默认值10。
a	-360-360	指定水印旋转角度。正数表示顺时针旋转，负数表示逆时针旋转。非必选，默认值0。
P	1-100	指定水印缩放比例。非必选，默认值20。

> **说明：**
>
> - 目前不支持在一组action中同时指定图片和文字水印，只能指定图片或文字水印一种类型。如果需要指定图片和文字水印，可通过多组action实现。
> - 锚点位置说明，将您的原始图片分成如下图所示9个区域，并按照图示位置进行编号，每个区域的锚点位置如图所示：



水印编码 在添加水印操作中，水印的文字内容、文字字体、图片水印的水印图片名称等参数需要进行URL安全的Base64编码。编码步骤如下：

- 将内容编码成Base64。将结果中的部分编码替换。
- 将结果中的加号（+）替换成短划线（-）。
- 将结果中的正斜线（/）替换成下划线（_）。

示例

- 添加图片水印

https://doc.bce.baidu.com/bce-documentation/BOS/image.jpg?x-bce-process=image/watermark,image_YmNlLWRvY3VtZW50YXRpb24vQk9TL2JjZS5qcGc=,g_9,a_90



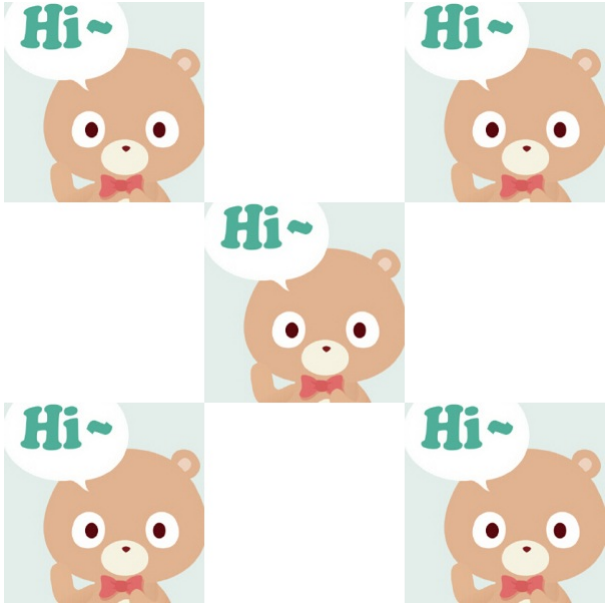
- 设置水印图比例
- https://doc.bce.baidu.com/bce-documentation/BOS/image.jpg?x-bce-process=image/watermark,image_YmNlLWRvY3VtZW50YXRpb24vQk9TL2JjZS5qcGc=,P_15,g_3



- 水印图预处理
- image值为 `bce-documentation/BOS/bce.jpg?x-bce-process=image/resize,w_50/rotate,a_90`,进行url safe base64处理
- https://doc.bce.baidu.com/bce-documentation/BOS/image.jpg?x-bce-process=image/watermark,image_YmNlLWRvY3VtZW50YXRpb24vQk9TL2JjZS5qcGc_eC1iY2UtcHJvY2Vzc1pbWFnZS9yZXNpemUsd181MC9yb3RhdGUsYV85MA==



- 图片九宫格
- 可支持九张图片拼接
- https://doc.bce.baidu.com/bce-documentation/BOS/white-600x600.png?x-bce-process=image/watermark,image_YmNlLWRvY3VtZW50YXRpb24vQk9TL2ItYWdlLmpwZz94LWJjZS1wcm9jZXNzPWltYWdlL3Jlc2l6ZSxtX2ZpbGwsd18yMDAsaF8yMDAsYWxpZ25fMzA=,w_200,x_0,y_0,g_1/watermark,image_Yn



添加图文混合水印

概述

本文介绍如何对图片添加图片和文字水印。命令名称：`watermark`。

图片水印参数

图片水印参数

参数	取值	说明
bucket	-	指定水印图片存储在BOS的Bucket名称，非必选，默认当前bucket。非当前bucket则要求该水印图片有公共读权限。
image	-	指定水印图片存储在BOS的Object名称，需要取url安全的Base64编码后的值。
t	1-100	指定图片水印透明度，取值为100时，表示与原始透明度一致。非必选，默认值100。

文字水印参数

参数	取值	说明
text	-	水印文字内容，需要进行url安全的Base64编码，且编码后最多不能超过200个字符。
size	1-1024	指定水印字体大小。非必选，默认30。
color	8位十六进制颜色码	指定字体颜色。非必选，默认黑色000000。
type	支持的字体见下表	指定字体的字体类型，取值要求进行url安全的Base64编码。非必选。若指定的字体不支持的字体列表中，则为默认为宋体。
style	normal, italic, bold	指定字体样式。normal（常规）italic（斜体）bold（粗体）。非必选，默认normal。只有当字体支持style效果时，才会渲染文字效果。
effect	hardshadow, softoutline	文字特效。 hardshadow：文字投影；softoutline：文字投影，投影增加高斯模糊。
shx	0-10	文字特效水平方向的偏移。非必选，默认值0。
shy	0-10	文字特效垂直方向的偏移。非必选，默认值0。
shc	8位十六进制颜色码	指定阴影颜色。
blr	1-50	指定描边高斯模糊半径。只在softoutline中生效。非必选，默认值0。
bls	1-50	指定描边高斯模糊标准差。只在softoutline中生效。非必选，默认值0。

公共参数

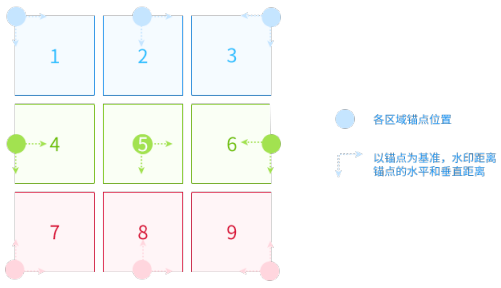
参数	取值	说明
g	1-9	指定水印的锚点位置。非必选，默认9。具体含义见下方说明。
x	-4096-4096	指定水印基于锚点的水平方向距离。非必选，默认值10。
y	-4096-4096	指定水印基于锚点的垂直方向距离。非必选，默认值10。
a	-360-360	指定水印旋转角度。正数表示顺时针旋转，负数表示逆时针旋转。非必选，默认值0。
order	0,1	指定文字和图片水印的前后顺序。0（默认值）：表示图片水印在前；1：表示文字水印在前。
align	0,1,2	指定文字水印和图片水印的对齐方式。 0：文字水印和图片水印上对齐；1：文字水印和图片水印居中对齐；2（默认值）：文字水印和图片水印下对齐。
interval	0-1000	指定文字水印和图片水印间的间距。非必选，默认值0。

水印编码 在添加水印操作中，水印的文字内容、文字字体、图片水印的水印图片名称等参数需要进行URL安全的Base64编码。编码步骤如下：

- 将内容编码成Base64。将结果中的部分编码替换。
- 将结果中的加号（+）替换成短划线（-）。
- 将结果中的正斜线（/）替换成下划线（_）。
- 将结果中尾部的所有等号（=）省略。

锚点位置

将原始图片分成如下图所示9个区域，并按照图示位置进行编号，每个区域的锚点位置如图所示：



支持的字体类型

字体	名称	base64编码
FZLanTingHei	方正兰亭黑简体	RlpMYW5UaW5nSGVp
FZLanTingHeiSBGB	方正兰亭粗黑简体	RlpMYW5UaW5nSGVpU0JHQg==
FZLanTingHeiSDB1GB	方正兰亭中粗黑简体	RlpMYW5UaW5nSGVpU0RCMUdC
FZLanTingHeiSDBGB	方正兰亭中黑简体	RlpMYW5UaW5nSGVpU0RCR0I=
FZLanTingHeiSEBGB	方正兰亭大黑简体	RlpMYW5UaW5nSGVpU0VCR0I=
FZLanTingHeiSELGB	方正兰亭纤黑简体	RlpMYW5UaW5nSGVpU0VMR0I=
FZLanTingHeiSHGB	方正兰亭特黑简体	RlpMYW5UaW5nSGVpU0hHQg==
FZLanTingHeiSLGB	方正兰亭细黑简体	RlpMYW5UaW5nSGVpU0xHQg==
FZLanTingHeiSMGB	方正兰亭准黑简体	RlpMYW5UaW5nSGVpU01HQg==
FZLanTingHeiSRGB	方正兰亭黑简体	RlpMYW5UaW5nSGVpU1JHQg==
FZLanTingHeiSULGB	方正兰亭超细黑简体	RlpMYW5UaW5nSGVpU1VMR0I=
FZKaTongM19S	方正卡通简体	RlpLYVRvbmdNMTIT
FZFangSong	方正仿宋简体	RlpGYW5nU29uZw==
FZShuSong	方正书宋简体	RlpTaHVTb25n
FZKai	方正楷体简体	RlpLYWk=
FZHei	方正黑体简体	RlpIZWk=

示例

- 图文混合水印，居中对齐
  https://doc.bce.baidu.com/bce-documentation/BOS/image.jpg?x-bce-process=image/watermark,image_YmNlLWRvY3VtZW50YXRpb24vQk9TL2JjZS5qcGc=,text_55m-5bqm5LqR,type_ZnpsYW50aW5naGVpc3JnYg==,align_1



- 图文混合水印，文字在前，居中对齐
  https://doc.bce.baidu.com/bce-documentation/BOS/image.jpg?x-bce-process=image/watermark,image_YmNlLWRvY3VtZW50YXRpb24vQk9TL2JjZS5qcGc=,text_55m-5bqm5LqR,type_ZnpsYW50aW5naGVpc3JnYg==,order_1,align_1



- 图文混合水印，文字硬阴影效果
  https://doc.bce.baidu.com/bce-documentation/BOS/image.jpg?x-bce-process=image/watermark,image_YmNlLWRvY3VtZW50YXRpb24vQk9TL2JjZS5qcGc=,text_55m-5bqm5LqR,type_ZnpsYW50aW5naGVpc3JnYg==,size_32,color_FFFFFFFF,effect_hardshadow,shx_2,shy_2,shc_696969,align_1



- 图文混合水印，文字软阴影效果
  https://doc.bce.baidu.com/bce-documentation/BOS/image.jpg?x-bce-process=image/watermark,image_YmNlLWRvY3VtZW50YXRpb24vQk9TL2JjZS5qcGc=,text_55m-5bqm5LqR,type_ZnpsYW50aW5naGVpc3JnYg==,size_32,color_FFFFFFFF,effect_softoutline,shx_2,shy_2,shc_696969,bls_2,align_1



添加文字水印

概述

对一张图片添加文字水印。

文字水印参数

命令名称：watermark。

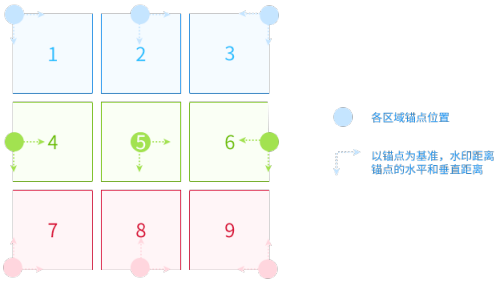
参数	取值	说明
text	-	水印文字内容，需要进行url安全的Base64编码，且编码后最多不能超过200个字符。文字水印必选。
size	1-1024	指定水印字体大小。非必选，默认30。
color	8位十六进制颜色码	指定字体颜色，格式为RRGGBBAA。最后2位Alpha通道可省略，省略时默认为00表示不透明。非必选，默认黑色000000。
type	支持的字体见下表	指定字体的字体类型，取值要求进行url安全的Base64编码。非必选。若指定的字体不支持的字体列表中，则为默认为宋体。
style	normal、italic、bold	指定字体样式。normal（常规）italic（斜体）bold（粗体）。非必选，默认normal。只有当字体支持style效果时，才会渲染文字效果。
g	1-9	指定水印的锚点位置。非必选，默认9。具体含义见下方说明。
x	-4096-4096	指定水印基于锚点的水平方向距离。非必选，默认值10。
y	-4096-4096	指定水印基于锚点的垂直方向距离。非必选，默认值10。
a	-360-360	指定水印旋转角度。正数表示顺时针旋转，负数表示逆时针旋转。非必选，默认值0。
skw	1-1024	指定文字描边宽度。非必选，默认值0。
skc	8位十六进制颜色码	指定文字描边颜色，格式为RRGGBBAA。最后2位Alpha通道可省略，省略时默认为00表示不透明。非必选，默认黑色000000。
blr	1-50	指定描边高斯模糊半径。非必选，默认值0。
bls	1-50	指定描边高斯模糊标准差。非必选，默认值0。
layout	tile	指定水印布局。tile（平铺）。非必选，默认值 无，具体效果见示例。

水印编码 在添加水印操作中，水印的文字内容、文字字体、图片水印的水印图片名称等参数需要进行URL安全的Base64编码。编码步骤如下：

- 将内容编码成Base64。将结果中的部分编码替换。
- 将结果中的加号（+）替换成短划线（-）。
- 将结果中的正斜线（/）替换成下划线（_）。
- 将结果中尾部的所有等号（=）省略。

锚点位置

将原始图片分成如下图所示9个区域，并按照图示位置进行编号，每个区域的锚点位置如图所示：



支持的字体类型

字体	名称	base64编码
FZLanTingHei	方正兰亭黑简体	RlpMYW5UaW5nSGVp
FZLanTingHeiSBGB	方正兰亭粗黑简体	RlpMYW5UaW5nSGVpU0JHQg==
FZLanTingHeiSDB1GB	方正兰亭中粗黑简体	RlpMYW5UaW5nSGVpU0RCMUdC
FZLanTingHeiSDBGB	方正兰亭中黑简体	RlpMYW5UaW5nSGVpU0RCR0I=
FZLanTingHeiSEBGB	方正兰亭大黑简体	RlpMYW5UaW5nSGVpU0VCR0I=
FZLanTingHeiSELGB	方正兰亭纤黑简体	RlpMYW5UaW5nSGVpU0VMR0I=
FZLanTingHeiSHGB	方正兰亭特黑简体	RlpMYW5UaW5nSGVpU0hHQg==
FZLanTingHeiSLGB	方正兰亭细黑简体	RlpMYW5UaW5nSGVpU0xHQg==
FZLanTingHeiSMGB	方正兰亭准黑简体	RlpMYW5UaW5nSGVpU01HQg==
FZLanTingHeiSRGB	方正兰亭黑简体	RlpMYW5UaW5nSGVpU1JHQg==
FZLanTingHeiSULGB	方正兰亭超细黑简体	RlpMYW5UaW5nSGVpU1VMR0I=
FZKaTongM19S	方正卡通简体	RlpLYVRvbmdNMTIT
FZFangSong	方正仿宋简体	RlpGYW5nU29uZw==
FZShuSong	方正书宋简体	RlpTaHVTb25n
FZKai	方正楷体简体	RlpLYWk=
FZHei	方正黑体简体	RlpIZWk=

示例

- 添加文字水印，文字为蓝色

  https://doc.bce.baidu.com/bce-documentation/BOS/image.jpg?x-bce-process=image/watermark,text_aGVsbG8=,type_RlpMYW5UaW5nSGVp,g_9,color_0000ff

- 添加文字水印，文字为白色，使用蓝色描边

  https://doc.bce.baidu.com/bce-documentation/BOS/image.jpg?x-bce-process=image/watermark,text_aGVsbG8=,type_RlpMYW5UaW5nSGVp,size_36,g_9,color_ffffff,skw_1,skc_0000ff

- 添加文字水印，使用平铺布局

  https://doc.bce.baidu.com/bce-documentation/BOS/image.jpg?x-bce-process=image/watermark,text_aGVsbG8=,size_30,g_3,x_20,y_20,a_45,layout_tile

图像缩放

概述

本文介绍如何对图片大小进行缩放处理(resize)。

缩放参数

参数	取值	说明
		• 设置缩略图的宽度，单位px。lfit类型必须设置w或h，fill fixed类型必须同时设置w和h • lfit：等比缩放图片，缩放到能放到以w、h为宽高的矩形内的最大缩放图片

m	lfit，mfit，fill，pad，fixed	• mfit：等比缩放图片，缩放到能包含住以w、h为宽高的矩形的最小缩放图片 • fill：mfit基础上，对w、h矩形内的图片部分进行裁剪 • pad：lfit基础上，对w、h矩形内的非图片空白部分进行指定颜色填充 • fixed：强制缩放到指定w宽、h高
w	1-4096	• 设置缩略图的高度，单位px。lfit类型必须设置w或h，fill fixed类型必须同时设置w和h。与l/s/p/wp/hp互斥。
h	1-4096	• 设置缩略图的高度，单位px。lfit类型必须设置w或h，fill fixed类型必须同时设置w和h。与l/s/p/wp/hp互斥。
l	1-4096	• 设置缩略图的最长边。与w/h/p/wp/hp互斥。
s	1-4096	• 设置缩略图的最短边。与w/h/p/wp/hp互斥。
p	1-1000	• 缩放比例，与m/w/h/wp/hp/l/s互斥。
wp	1-1000	• 仅宽度进行缩放比例，与m/w/h/p/l/s互斥。
hp	1-1000	• 仅高度进行缩放比例，与m/w/h/p/l/s互斥。
align	0-100	• 当需要裁剪时，截取图片的偏移。默认50，居中裁剪。
limit	0,1	• 设置是否限制图片缩放的尺寸大小不可超过原图大小，0不限制，1限制。非必选，默认0不限制。
color	8位十六进制颜色码	• 指定pad类型下填充颜色，格式为RRGGBBAA。最后2位Alpha通道可省略，省略时默认为00表示不透明。非必选，默认黑色`000000`。

radius	1-50	• 指定pad类型下背景虚化高斯模糊半径。非必选，默认不模糊，填充颜色。与sigma共同设置时生效。
sigma	1-50	• 指定pad类型下背景虚化高斯模糊标准差。非必选，默认不模糊，填充颜色。与radius共同设置时生效。

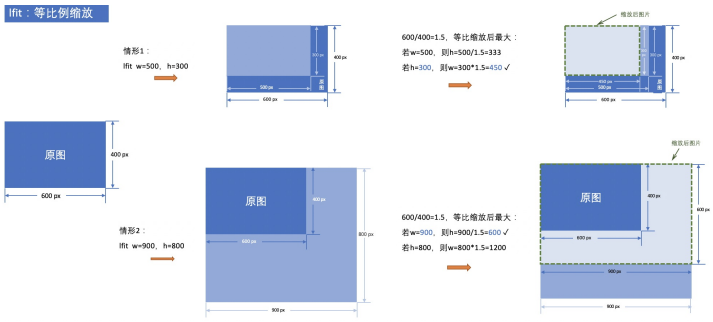
效果示例

- **lfit缩放**

若原图大小为 600 px 乘以 400 px。

情形 1：缩放参数为 w = 500 px，h = 300 px。按照 lift 等比缩放规则，缩放图的 w/h 等于原图的 w/h=1.5。所以，若：w = 500 px，则 h = 500/1.5 = 333 px；h = 300 px，则 w = 300x1.5 = 450 px。若限制在指定 w 与 h 的矩形内的最大图片，即缩放图的 w 乘 h 不能大于 500 px 乘 300 px。通过以上条件得出缩略图大小为 450 px 乘以 300 px。

情形 2：缩放参数为 w = 900 px，h = 800 px。按照 lift 等比缩放规则，缩放图的 w/h 等于原图的 w/h=1.5。所以，若：w = 900 px，则 h = 900/1.5 = 600 px；h = 800 px，则 w = 800x1.5 = 1200 px。若限制在指定 w 与 h 的矩形内的最大图片，即缩放图的 w 乘 h 不能大于 900 px 乘 800 px。通过以上条件得出缩略图大小为 900 px 乘以 600 px。

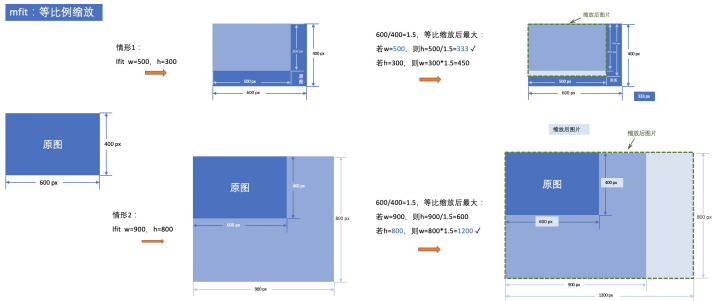


- **mfit缩放**

若原图大小为 600 px 乘以 400 px。

情形 1：缩放参数为 w = 500 px，h = 300 px。按照 mift 等比缩放规则，缩放图的 w/h 等于原图的 w/h=1.5。所以，若：w = 500 px，则 h = 500/1.5 = 333 px；h = 300 px，则 w = 300x1.5 = 450 px。若限制为指定 w 与 h 的矩形框外的最小图片，即缩放图的 w 乘 h 不能小于 500 px 乘 300 px。通过以上条件得出缩略图大小为 500 px 乘以 333 px。

情形 2：缩放参数为 w = 900 px，h = 800 px。按照 lift 等比缩放规则，缩放图的 w/h 等于原图的 w/h=1.5。所以，若：w = 900 px，则 h = 900/1.5 = 600 px；h = 800 px，则 w = 800x1.5 = 1200 px。若限制为指定 w 与 h 的矩形框外的最小图片，即缩放图的 w 乘 h 不能小于 1200 px 乘 800 px。通过以上条件得出缩略图大小为 1200 px 乘以 800 px。

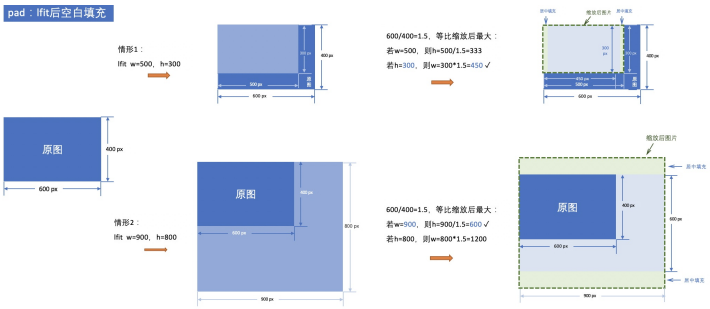


- **pad缩放**

若原图大小为 600 px 乘以 400 px。

情形 1：缩放参数为 w = 500 px，h = 300 px。按照 lift 等比缩放规则，缩放图的 w/h 等于原图的 w/h=1.5。所以，若：w = 500 px，则 h = 500/1.5 = 333 px；h = 300 px，则 w = 300x1.5 = 450 px。若限制在指定 w 与 h 的矩形内的最大图片，即缩放图的 w 乘 h 不能大于 500 px 乘 300 px。通过以上条件得出缩略图大小为 450 px 乘以 300 px。在 pad 缩放规则下，需要对缩略图和 w 乘 h 矩形框中间的部分进行填充。下图示例为填充绿色背景。

情形 2：缩放参数为 w = 900 px，h = 800 px。按照 lift 等比缩放规则，缩放图的 w/h 等于原图的 w/h=1.5。所以，若：w = 900 px，则 h = 900/1.5 = 600 px；h = 800 px，则 w = 800x1.5 = 1200 px。若限制在指定 w 与 h 的矩形内的最大图片，即缩放图的 w 乘 h 不能大于 900 px 乘 800 px。通过以上条件得出缩略图大小为 900 px 乘以 600 px。在 pad 缩放规则下，需要对缩略图和 w 乘 h 矩形框中间的部分进行填充。下图示例为填充绿色背景。

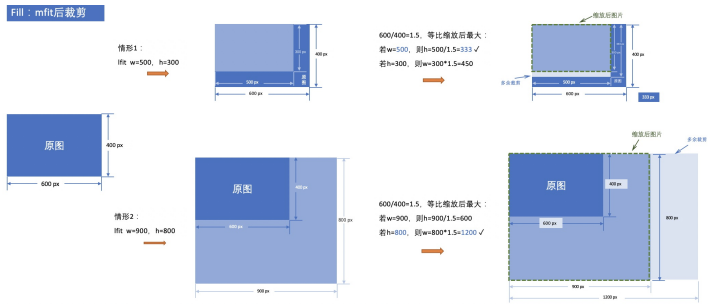


- **fill缩放**

若原图大小为 600 px 乘以 400 px。

情形 1：缩放参数为 w = 500 px，h = 300 px。按照 mift 等比缩放规则，缩放图的 w/h 等于原图的 w/h=1.5。所以，若：w = 500 px，则 h = 500/1.5 = 333 px；h = 300 px，则 w = 300x1.5 = 450 px。若限制为指定 w 与 h 的矩形框外的最小图片，即缩放图的 w 乘 h 不能小于 500 px 乘 300 px。通过以上条件得出缩略图大小为 500 px 乘以 333 px。在 fill 缩放规则下，需要对缩略图和 w 乘 h 矩形框中间的部分进行裁剪。

情形 2：缩放参数为 w = 900 px，h = 800 px。按照 lift 等比缩放规则，缩放图的 w/h 等于原图的 w/h=1.5。所以，若：w = 900 px，则 h = 900/1.5 = 600 px；h = 800 px，则 w = 800x1.5 = 1200 px。若限制为指定 w 与 h 的矩形框外的最小图片，即缩放图的 w 乘 h 不能小于 1200 px 乘 800 px。通过以上条件得出缩略图大小为 1200 px 乘以 800 px。在 fill 缩放规则下，需要对缩略图和 w 乘 h 矩形框中间的部分进行裁剪。

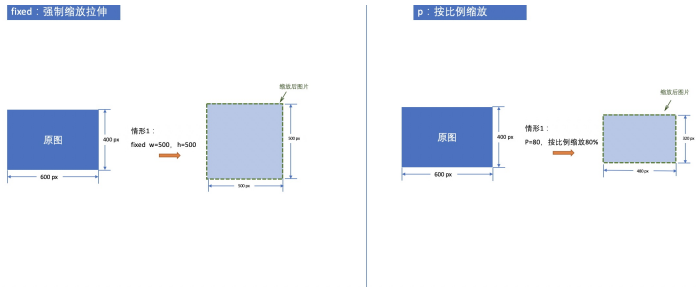


- **fixed缩放**

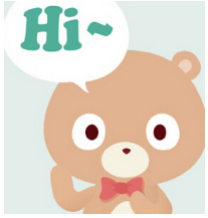
若原图大小为 600 px 乘以 400 px，缩放参数为 w = 500 px，h = 300 px。按照 fixed 缩放规则，需要对原图按照 w 和 h 进行强制拉伸平铺。

- **按比例缩放**

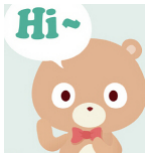
若原图大小为 600 px 乘以 400 px，缩放参数为 p = 80。按照等比缩放规则，需要对原图按照 80% 的比例进行按比例缩放，缩略图的长和宽都将缩放为原图尺寸的 80%。



使用案例

- 等比缩放图片(lfit)，缩放后的图片宽最大200，高最大100。
  https://doc.bce.baidu.com/bce-documentation/BOS/image.jpg?x-bce-process=image/resize,m_lfit,w_200,h_100
  
- 等比缩放图片(mfit)，缩放后的图片宽最小200，高最小100。
  https://doc.bce.baidu.com/bce-documentation/BOS/image.jpg?x-bce-process=image/resize,m_mfit,w_200,h_100
  
- 等比缩放图片(fill)，缩放后的图片宽最小200，高最小100，并对超出200*100部分图片进行居中裁剪(默认居中裁剪)。
  https://doc.bce.baidu.com/bce-documentation/BOS/image.jpg?x-bce-process=image/resize,m_fill,w_200,h_100
  
- 等比缩放图片(fill)，缩放后的图片宽最小200，高最小100，并对超出200*100部分图片从顶部开始裁剪。
  https://doc.bce.baidu.com/bce-documentation/BOS/image.jpg?x-bce-process=image/resize,m_fill,w_200,h_100,align_0



- 等比缩放图片(pad)，缩放后的图片宽最大200，高最大100，并对200*100矩形内图片未占满的部分进行填充。
  https://doc.bce.baidu.com/bce-documentation/BOS/image.jpg?x-bce-process=image/resize,m_pad,w_200,h_100,color_FF0000
  
- 强制缩放(fixed)，缩放后的图片宽200，高100。
  https://doc.bce.baidu.com/bce-documentation/BOS/image.jpg?x-bce-process=image/resize,m_fixed,w_200,h_100
  
- 等比缩放图片，按照原图50%的比例缩放。
  https://doc.bce.baidu.com/bce-documentation/BOS/image.jpg?x-bce-process=image/resize,p_50
  
- 比例缩放宽度图片，宽度按照原图50%的比例缩放。
  https://doc.bce.baidu.com/bce-documentation/BOS/image.jpg?x-bce-process=image/resize,wp_50
  
- 比例缩放高度图片，高度按照原图50%的比例缩放。
  https://doc.bce.baidu.com/bce-documentation/BOS/image.jpg?x-bce-process=image/resize,hp_50
  
- 等比缩放图片(pad)，缩放后的图片宽最大200，高最大100，并对200*100矩形内图片未占满的部分进行背景模糊。
  https://doc.bce.baidu.com/bce-documentation/BOS/image.jpg?x-bce-process=image/resize,m_pad,w_200,h_100,radius_2,sigma_50
  

图像旋转

普通旋转

概述 本文介绍对图片进行一定角度的普通旋转。

普通旋转参数

action名称：`rotate`

参数	取值	说明
a	-360-360	图片旋转角度，正数表示顺时针旋转，负数表示逆时针旋转。

示例

- 将图片顺时针旋转30度

https://doc.bce.baidu.com/bce-documentation/BOS/image.jpg?x-bce-process=image/rotate,a_30



自适应旋转

概述

图像可以根据 exif 信息进行自适应旋转。

自适应旋转参数

action名称：`auto-orient`

参数	取值	说明
o	0,1	0：按原图默认方向，不自动旋转；1：自适应旋转。

质量变换

概述

设置图片编码质量。

质量只对有损压缩的图片格式生效，包括JPEG, WebP, Heic等。对于无损压缩的图片格式，包括PNG, TIFF，BMP, GIF等，则不会产生效果。

质量参数

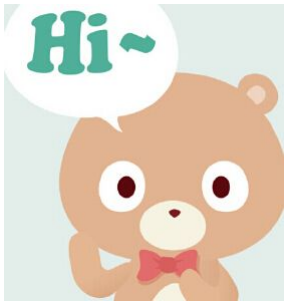
命令名称：`quality`。

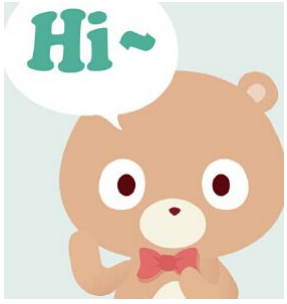
参数	取值	说明
q	1-100	指定图片的相对质量，假设原图质量为N，输出质量为N * q%。
Q	1-100	指定图片的绝对质量，假设原图质量为N，输出质量为Q。
c	le, ge, any	指定图片的绝对质量Q的生效条件，假设原图质量为N，如果为le，则Q < N时生效；如果为ge，则Q > N时生效；如果为any，则任意条件生效。默认为le。

注意事项：

1. 不同图片格式的质量没有可对比性，比如JPEG的质量75和WebP的质量75，两者之间无论画质、体积都没有可对比性；
2. 质量参数越高画质越好，但图片体积也会增大，推荐平衡画质和体积的质量：JPEG是75，WebP是75，Heic是45；
3. 参数Q配合c_ge可以提升图片的编码质量，但只有在缩略图的情况下，才会提高图片画质，在没有缩略的情况下，并不会增强图片画质；

示例

- 将图片质量设置为当前质量的80%
  https://doc.bce.baidu.com/bce-documentation/BOS/image.jpg?x-bce-process=image/quality,q_80
  

- 将图片质量设置为80
  https://doc.bce.baidu.com/bce-documentation/BOS/image.jpg?x-bce-process=image/quality,Q_80
  

智能处理

智能场景

风格转换

概述

可将图像转化成卡通画、铅笔画、彩色铅笔画，或者哥特油画、彩色糖块油画、呐喊油画、神奈川冲浪里油画、奇异油画、薰衣草油画等共计 9 种风格。适用于社交，趣味活动等使用场景，或集成到美图应用中对图像进行风格转换。

参数 action名称：style-trans

参数	取值	说明
o	• cartoon：卡通画风格 • pencil：铅笔画风格 • color_pencil：彩色铅笔画风格 • warm：彩色糖块油画风格 • wave：神奈川冲浪里油画风格 • lavender：薰衣草油画风格 • mononoke：奇异油画风格 • scream：呐喊油画风格 • gothic：哥特油画风格	图像风格，必选。

示例

- 风格转换原图：



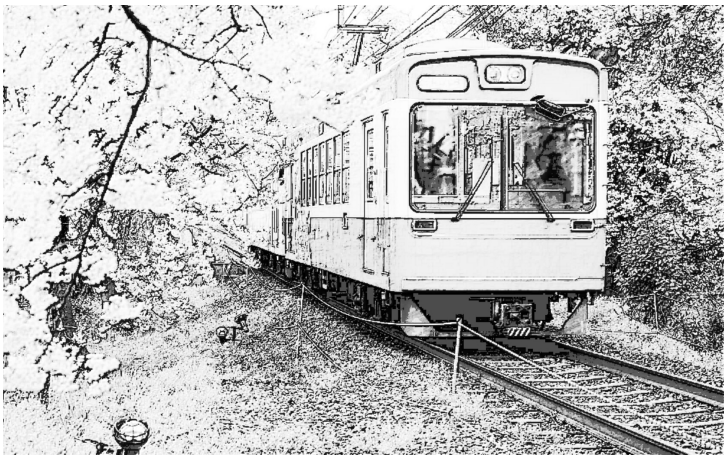
- 转换为卡通画风格：

https://doc.bce.baidu.com/bce-documentation/BOS/src_style_trans.jpg?x-bce-process=image/style-trans,o_cartoon



- 转换为铅笔画风格：

https://doc.bce.baidu.com/bce-documentation/BOS/src_style_trans.jpg?x-bce-process=image/style-trans,o_pencil



- 转换为彩色铅笔画风格：

https://doc.bce.baidu.com/bce-documentation/BOS/src_style_trans.jpg?x-bce-process=image/style-trans,o_color-pencil



- 转换为彩色糖块油画风格：

https://doc.bce.baidu.com/bce-documentation/BOS/src_style_trans.jpg?x-bce-process=image/style-trans,o_warm



- 转换为神奈川冲浪里油画风格：

https://doc.bce.baidu.com/bce-documentation/BOS/src_style_trans.jpg?x-bce-process=image/style-trans,o_wave



- 转换为薰衣草油画风格：

https://doc.bce.baidu.com/bce-documentation/BOS/src_style_trans.jpg?x-bce-process=image/style-trans,o_lavender



- 转换为奇异油画风格：

https://doc.bce.baidu.com/bce-documentation/BOS/src_style_trans.jpg?x-bce-process=image/style-trans,o_mononoke



- 转换为呐喊油画风格：

https://doc.bce.baidu.com/bce-documentation/BOS/src_style_trans.jpg?x-bce-process=image/style-trans,o_scream



- 转换为哥特油画风格：

https://doc.bce.baidu.com/bce-documentation/BOS/src_style_trans.jpg?x-bce-process=image/style-trans,o_gothic



黑白上色

概述 智能识别黑白图像内容并填充色彩，使黑白图像变得鲜活。

参数 action名称：colourize

示例

- 黑白图片上色。如图是黑白图片上色前后的对比效果：

https://doc.bce.baidu.com/bce-documentation/BOS/src_black_and_white.jpg?x-bce-process=image/colourize





拉伸恢复

概述 自动识别过度拉伸的图像，将图像内容恢复成正常比例。

参数 action名称：stretch-restore

示例

- 示例原图与拉伸修复效果图：

https://doc.bce.baidu.com/bce-documentation/BOS/src_stretch_restore.jpg?x-bce-process=image/stretch-restore





人像动漫化

概述

运用世界领先的对抗生成网络，结合人脸检测、头发分割、人像分割等技术，为用户量身定制千人千面的二次元动漫形象，并且可通过参数设置，生成戴口罩的二次元动漫人像。

参数

action名称：selfie-anime

参数	取值	说明
t	"anime"或者"anime-mask"。	设置二次元动漫画是否戴口罩。前者anime生成二次元动漫图，后者anime-mask生成戴口罩的二次元动漫人像。非必选，默认是anime。
m	1～8之间的整数	在t参数填入anime-mask时生效，m是1～8之间的整数，用于指定所使用的口罩的编码。t参数没有填入anime-mask，或m参数为空时，生成不戴口罩的二次元动漫图。

示例

- 示例原图：



- 转换为二次元动漫形象：

https://doc.bce.baidu.com/bce-documentation/BOS/src_selfie_anime.jpg?x-bce-process=image/selfie-anime,t_anime



- 转换为佩戴 N95 口罩的二次元动漫形象，口罩编号为 1：

https://doc.bce.baidu.com/bce-documentation/BOS/src_selfie_anime.jpg?x-bce-process=image/selfie-anime,t_anime-mask,m_1



- 转换为佩戴普通蓝色口罩的二次元动漫形象，口罩编号为 2：

https://doc.bce.baidu.com/bce-documentation/BOS/src_selfie_anime.jpg?x-bce-process=image/selfie-anime,t_anime-mask,m_2



- 转换为佩戴普通白色口罩的二次元动漫形象，口罩编号为 3：

https://doc.bce.baidu.com/bce-documentation/BOS/src_selfie_anime.jpg?x-bce-process=image/selfie-anime,t_anime-mask,m_3



- 转换为佩戴重型蓝色口罩的二次元动漫形象，口罩编号为 4：

https://doc.bce.baidu.com/bce-documentation/BOS/src_selfie_anime.jpg?x-bce-process=image/selfie-anime,t_anime-mask,m_4



- 转换为佩戴重型棕色口罩的二次元动漫形象，口罩编号为 5：

https://doc.bce.baidu.com/bce-documentation/BOS/src_selfie_anime.jpg?x-bce-process=image/selfie-anime,t_anime-mask,m_5



- 转换为佩戴粉色笑脸口罩的二次元动漫形象，口罩编号为 6：

https://doc.bce.baidu.com/bce-documentation/BOS/src_selfie_anime.jpg?x-bce-process=image/selfie-anime,t_anime-mask,m_6



- 转换为佩戴重型口罩的二次元动漫形象，口罩编号为 7：

https://doc.bce.baidu.com/bce-documentation/BOS/src_selfie_anime.jpg?x-bce-process=image/selfie-anime,t_anime-mask,m_7



- 转换为佩戴粉色纯色口罩的二次元动漫形象，口罩编号为 8：

https://doc.bce.baidu.com/bce-documentation/BOS/src_selfie_anime.jpg?x-bce-process=image/selfie-anime,t_anime-mask,m_8



天空分隔

概述 可智能分割出天空边界位置，输出天空和其余背景的灰度图，可用于图像二次处理，进行天空替换、抠图等图片编辑场景。

参数

action名称：sky-seg
参数：无特殊参数

示例

- 示例原图和天空分隔效果图如下所示：

https://doc.bce.baidu.com/bce-documentation/BOS/src_sky_seg.jpg?x-bce-process=image/sky-seg





无损放大

概述 输入一张图片，可以在尽量保持图像质量的条件下，将图像在长宽方向各放大两倍。

参数 action名称：quality-enhance

示例

- 示例原图与无损放大效果图：

https://doc.bce.baidu.com/bce-documentation/BOS/src_quality_enhance.jpg?x-bce-process=image/quality-enhance





物体擦除

概述 擦除图片中不需要的遮挡物，并用背景内容填充以提高图像质量。擦除位置为规则矩形，需要给出坐标和大小。

参数 action名称：inpainting

参数	取值	说明
x	0-2000	擦除物体左上角起点相对原点的 x 坐标，必选
y	0-2000	擦除物体左上角起点相对原点的 y 坐标，必选
w	1-2000	w，擦除区域的宽度，必选
h	1-2000	h，擦除区域的高度，必选

示例

- 物体擦除前后对比图：

https://doc.bce.baidu.com/bce-documentation/BOS/src_inpainting.jpg?x-bce-process=image/inpainting,x_502,y_158,w_275,h_229

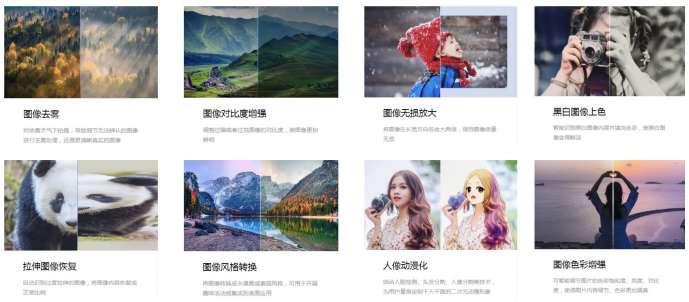




智能处理使用规则

智能图片处理概述

图片处理服务依托对象存储 BOS 系统对外提供灵活、多样、实时的图片处理服务。用户将原始图片上传到 BOS 上，通过调用简单的 RESTful 接口，实现对图片的实时处理，如缩略、裁剪、格式转换、旋转、加文字/图片水印等，处理结果图会通过CDN实时的返回给用户。智能图片处理则通过 BOS 图片处理平台对外开放百度领先的 AI 图片处理能力。



场景与解决方案

智能图像处理包括智能效果和智能场景两部分，您可以根据自己的业务场景选择合适的智能处理方式。

典型场景	场景需求	解决方案	推荐功能
手机云相册	手机相册需要具备如裁剪、旋转、调节对比度等处理功能	手机 APP 将用户上传的照片存储到 BOS。当用户再次读取时，直接调用 BOS 原生图像处理功能进行处理，并将处理后图像作为副本继续存储在云相册	• BOS 基础图像处理功能（缩放、裁剪、旋转等）； • 智能能力如对比度、清晰度、色彩增强等图像效果能力； • 事件通知触发通用物体/人脸识别；
安防	阴雨天或大雾天导致捕获图像的清晰度较低	安防设备将捕获的视频或者图像存储到 BOS，调用 BOS 原生图像处理能力如图像去雾，以提升文件清晰度	• BOS 基础图像处理功能（缩放、裁剪、旋转等）； • 智能图像处理：图像去雾； • MCT 视频转码，以降低存储费用；
电商	图片失真、水印去除、对图像尺寸进行缩放	将图像存储到 BOS，在读取图像时自动进行无损放大，并添加水印或者其他基础处理样式；	• BOS 基础图像处理功能（缩放、裁剪、旋转、水印等）； • 智能图像处理能力：无损放大、物体擦除；
社交	社交产品需要一定的产品趣味性来拉新留存	社交 APP 将图像存储到 BOS，在读取图像时自动进行人像动漫化或者风格转换	• BOS 基础图像处理功能（缩放、裁剪、旋转等）； • 智能图像处理：人像动漫化、风格转换等； • 事件通知触发通用物体/人脸识别；
促销活动	重大节日或促销场景，商家运营活动需要增添一定趣味性	将图像存储到 BOS，在读取图像时自动进行无损放大，并添加水印或者其他基础处理样式；	• BOS 基础图像处理功能（缩放、裁剪、旋转等）； • 智能图像处理：人像动漫化、风格转换、黑白上色等； • 事件通知触发通用物体/人脸识别；

使用方式

您需要先将原始图片上传到 BOS，每个 BOS Bucket 默认开通图片处理的功能，无需单独开通，也无需使用特定域名访问。当访问图片 URL 时，在 Querystring 中通过 **x-bce-process** 参数指定图片处理命令，即可触发 BOS 图片处理功能。BOS 支持通过命令和样式两种方式访问图片服务。

- 命令方式访问格式：`http://${domain}/${objectkey}?x-bce-process=image/${action},${key}_${value}.`
  样例：`http://bucket-A.bj.bcebos.com/sample.jpg?x-bce-process=image/resize,m_fill,w_200,h_300`。
- 样式方式访问格式：`http://${domain}/${objectkey}?x-bce-process=style/${stylename}`. 样例：`http://bucket-A.bj.bcebos.com/sample.jpg?x-bce-process=style/stylename`。

> **说明：**
>
> - 智能图片处理仅支持图片处理 V2.0 版本，V1.0 版本暂不支持。

规则与限制

- 智能图片处理支持的原图格式：PNG、JPG、JPEG、BMP。
- 支持的目标格式：PNG、JPG、JPEG、BMP。
- 需要处理的图片 base64 编码后大小不超过 4M,最短边至少 64px，最长边最大 4096px，长宽比 3:1 以内。特殊说明的除外。
- 处理结果输出图片宽高不超过 4096 px，包括管道和子命令间的中间输出图片。
- BOS 图片处理 V2.0 最多可以同时指定 4 组操作，即 4 个 action，但只能**同时最多指定一组智能处理有关的 action**。
- 一个 action 下不可以重复指定同一个参数。
- 无损放大需要处理的图片像素乘积不超过 1600p x 1600px。
- 智能去雾需要处理的图片最短边至少 200px，最长边最大 4096px。
- 黑白上色需要处理的图片最短边至少 64px，最长边最大 800px。
- 天空分割需要处理的图片最短边至少 64px，最长边最大 2049px，像素乘积不超过 2049*2049。
- 物体擦除使用设置位置参数方法进行修复时，最短边至少 100px，最长边最大 2000px，长宽比 3：1内。

智能效果

智能调色

概述

可智能调节图片的色彩饱和度、亮度、对比度，使得图片内容细节、色彩更加逼真。智能调色可用于提升网站图片、手机相册图片、视频封面图片的质量。您无需指定色彩参数，BOS 会为您自适应最佳效果。

参数 action名称：color-enhance

示例

- 示例原图和智能调色处理后效果图：

https://doc.bce.baidu.com/bce-documentation/BOS/src_color_enhance.jpg?x-bce-process=image/color-enhance





智能对比度

概述 调整过暗或者过亮图像的对比度，使图像更加鲜明。您无需指定对比度参数，BOS 会为您自适应最佳对必须效果。

参数 action名称：contrast-enhance

示例

- 示例原图与智能对比度处理后效果图：

https://doc.bce.baidu.com/bce-documentation/BOS/src_contrast_enhance.jpg?x-bce-process=image/contrast-enhance





智能清晰度

概述 对压缩后的模糊图像实现智能快速去噪，优化图像纹理细节，使画面更加自然清晰。您无需设置清晰度参数，BOS 会为您自适应最佳清晰度。

参数 action名称：definition-enhance

- 示例原图和智能清晰度处理后效果图：

https://doc.bce.baidu.com/bce-documentation/BOS/src_quality_enhance.jpg?x-bce-process=image/definition-enhance





智能去雾

概述 对浓雾天气下拍摄，导致细节无法辨认的图像进行去雾处理，还原更清晰真实的图像。

参数 action名称：dehaze

示例

- 示例原图与智能去雾处理后的效果图:

https://doc.bce.baidu.com/bce-documentation/BOS/src_dehaze.jpg?x-bce-process=image/dehaze





失败返回原图

概述

当前图片处理在遇到图片体积过大，图片格式无法解析时会报错，启用失败返回原图参数可以在遇到这些情况时返回用户上传原始数据。

该参数需要搭配其他处理指令一起使用。

亮度参数

action名称：`ignore-error`

参数	取值	说明
i	0，1	1 表示启用失败返回原图，0表示关闭失败返回原图，默认为0

示例

- 增加文字水印，失败返回原图
  https://doc.bce.baidu.com/bce-documentation/BOS/image.jpg?x-bce-process=image/watermark,text_aGVsbG8/ignore-error,i_1



图片压缩

概述

图片压缩是一种节约存储空间和提高图片访问速度的重要方法。在这方面，BOS的图片压缩功能可以将图片转换为Webp/HEIC/AVIF 压缩图片格式，以减小图片的大小并提高访问速度。

参数说明

action名称：`format`

参数	取值	说明
f	webp、heic 、avif	指定目标图片的图片格式

使用示例

- 原图格式为 JPEG，图片大小为62KB，如下图所示。



- 将JPEG格式的原图转换为WEBP格式。
  https://doc.bce.baidu.com/bce-documentation/BOS/image.jpg?x-bce-process=image/format,f_webp
- 将JPEG格式的原图转换为HEIC格式。
  https://doc.bce.baidu.com/bce-documentation/BOS/image.jpg?x-bce-process=image/format,f_heic
- 压缩率对比

格式	图片大小
jpg	62.02KB
webp	14.77KB(压缩率76.19%)
heic	4.30KB（压缩率93.07%）

图片处理持久化

概述 BOS 支持图片持久化。 持久化有两种方式：

- 存量持久化：将已经存在于BOS的图片进行处理后保存到BOS中。
- 增量持久化：上传图片到BOS，同时处理上传的图片，将原图和处理图都保存到BOS中。

语法 请求（Request）

```
 PUT /<ObjectKey> HTTP/1.1
Host: <BucketName>.bj.bcebos.com
Date: <Date>
Authorization: <Authorization_String>
Content-Type: <type>
x-bce-process: system/load,b_buc,o_obj|image/<imageprocess>
```

请求头x-bce-process必选，在header和query中填入均可

- image后面使用新版本图片处理参数
- system用来指定保存信息，load表示从指定bucket中取出object；save表示将object保存到指定bucket。具体支持的action及参数如下：

action	参数	取值	说明	是否必须
load	b	-	bucket名，如果省略，则默认原object所在bucket	否
	o	-	经过url安全的Base64编码的object名称	是
save	b	-	bucket名，如果省略，则默认原object所在bucket	否
	o	-	经过url安全的Base64编码的object名称	是

存量持久化 请求（Request） 通过在PutObject请求header中设置x-bce-process头来指定具体的持久化请求。

```
 PUT /<ObjectKey> HTTP/1.1
Host: <BucketName>.bj.bcebos.com
Date: <Date>
Authorization: <Authorization_String>
Content-Type: <type>
x-bce-process: system/load,b_buc,o_obj|image/<imageprocess>
```

响应（Response） 返回处理持久化保存结果

示例 对存储在BOS的mytest bucket的test.jpg进行resize和格式转换，将处理结果图片保存到mytest1 bucket下的test_resize.bmp。

```
 **请求**
PUT /test_resize.bmp HTTP/1.1
Host: mytest1.bj.bcebos.com
Date: <Date>
Authorization: <Authorization_String>
Content-Type: <type>
x-bce-process: system/load,b_mytest,o_dGVzdC5qcGc=|image/resize,m_fill,w_100,h_90/format,f_bmp

 **响应**
HTTP/1.1 200 OK
x-bce-request-id: 4db2b34d-654d-4d8a-b49b-3049ca786409
Date: Wed, 06 Apr 2016 06:34:40 GMT
ETag: "1b2cf535f27731c974343645a3985328"
Content-Length: 0
Connection: close
Server: BceBos
```

> 限制：需要用户对mytest bucket有读权限，对mytest1 bucket有写权限，两者缺一不可

增量持久化 请求（Request） 通过在PutObject请求header中设置x-bce-process头来指定具体的持久化请求。

```
 PUT /<ObjectKey> HTTP/1.1
Host: <BucketName>.bj.bcebos.com
Date: <Date>
Authorization: <Authorization_String>
Content-Type: text/plain
Content-Length: <Content_Length>
x-bce-process: image/<image_process>|system/save,b_buc,o_obj
```

响应（Response） 返回用户上传结果。

示例 上传test.jpg到mytest bucket，同时，对test.jpg进行resize和格式转换，将处理结果图片保存到mytest1 bucket下的test_resize.bmp。

```
 **请求**
PUT /test.jpg HTTP/1.1
Host: mytest.bj.bcebos.com
Date: Wed, 06 Apr 2016 06:34:40 GMT
Authorization: <Authorization_String>
Content-Type: image/jpeg
Content-Length: 102409
x-bce-process：image/resize,m_fill,w_100,h_90/format,f_bmp|system/save,b_mytest1,o_dGVzdF9yZXNpemUuYm1w

[102409 bytes of object data]

**响应**
HTTP/1.1 200 OK
x-bce-request-id: 4db2b34d-654d-4d8a-b49b-3049ca786409
Date: Wed, 06 Apr 2016 06:34:40 GMT
ETag: "1b2cf535f27731c974343645a3985328"
Content-Length: 0
Connection: close
Server: BceBos
```

> 限制：需要用户对mytest bucket有写权限，对mytest1 bucket有写权限，两者缺一不可

图片异常检测

概述

接口检测图片中是否隐含其他类型的可疑文件。

获取图片信息参数

action名称：`detect`

此命令没有参数，以 x-bce-process=image/detect 形式访问即可。

返回参数 | 字段名 | 取值举例 | 说明 | | --- | ---------- | ---------- | | picSuffix | jpg 等 | 图片声明类型 | | picResolution | 1*1 如果是异常图片统一返回0*0 | 图像文件分辨率 | | picSize |12345| 图片大小 | | detectedType | 支持检测的图片类型返回实际图片类型
不支持的图片类型返回"unknow" | 文件头解析类型 | | videoDetected | true：图片中包含未加密的视频流
false：图片中不包含未加密的视频流，可能视频流经过加密，也可能包含其他风险 | 文件是否携带未加密的视频流 | | picRisk | true：图片存在风险
false：图片当前检测安全 | 最终图片的检测结果，图片是否存在风险 |

示例

- 调用图片异常检测
  https://doc.bce.baidu.com/bce-documentation/BOS/image.jpg?x-bce-process=image/detect

```
{
 "picSuffix": "jpg",
 "picResolution": "279*296",
 "picSize": 63506,
 "detectedType": "image/jpeg",
 "videoDetected": false,
 "picRisk": false
}
```

图像盲水印

概述

盲水印是指在不影响原始图片内容的前提下，将不可见水印嵌入到媒体内容中。通常用在版权保护、内容追踪和防伪等场景。盲水印有以下特性：

- **版权保护**。对图片嵌入文字盲水印或者文字盲水印，可通过对图片进行解码提取操作，得到水印图来证明版权信息
- **内容追踪**。可使用盲水印功能，对图片加上特殊标识，如果内容被非授权传播，可根据解码提取出的标识信息，判断泄露源

注意事项

- 因 BOS 图片盲水印功能依赖 MCP 服务，所以使用 BOS 图片盲水印服务过程中产生的接口调用费用，由 MCP 服务进行收取。具体定价请参考 MCP 数字水印服务的定价文档。MCP 服务数字水印计费项
- 使用 BOS 图片盲水印功能前，请确保已经开通音视频 MCP 服务。MCP 服务快速使用流程

使用限制

- BOS 盲水印服务当前仅支持属于北京、苏州、广州地域的 Bucket
- 盲水印图片限制格式为：PNG, JPG, BMP, JPEG 格式
- 使用盲水印功能，原图的宽高都需大于 512
- 文字盲水印当前支持数字 [0 - 9] 及英文大小写 [A - Z,a - z] 及常用特殊字符，长度不大于 100
- 算法 0 可以抵抗缩放、裁剪、遮挡、截屏等攻击，但抗图片压缩能力较弱，水印容量较小。算法 1 可抵抗一定程度的裁剪、遮挡、截屏、图片压缩等攻击，水印容量较大，但不可抵抗缩放攻击
- 输入图片最大 50M；水印图最大 5M
- 嵌入盲水印和提取盲水印使用的算法需要一致，不然提取水印失败。算法 0 会返回没有信息的图片，算法 1 会返回空字符串

算法说明

- 算法 0

> - 将水印以图片的形式嵌入。如果嵌入为文字，文字会被自动转为图片再嵌入。无论嵌入图片还是文字，提取水印结果始终为图片。无论是否含有水印，该算法总会提取出一张图片，需要人工查看图片判断其中是否含有水印内容
> - 该算法能够抵抗一定程度的缩放、裁剪、遮挡、截屏等攻击，但抗图片压缩能力较弱，水印容量较小。相比于未受攻击的图片，从被攻击后的图片提取出的水印图可能有一定的模糊、缺漏，取决于攻击程度

- 算法 1

> - 仅支持嵌入文字，水印以编码形式嵌入。提取结果为字符串
> - 该算法可抵抗一定程度的裁剪、遮挡、截屏、图片压缩等攻击，水印容量较大，但不可抵抗缩放攻击
> - 相比算法 0，该算法在嵌入文字场景更加易用，可端到端提取准确的水印字符，无需人工检查

盲水印参数

action名称：`blind-watermark`

参数	取值	说明
method	extract/embed	区分此次请求为提取水印还是嵌入水印，提取水印 extract，嵌入水印为 embed，必选
mode	0/1	区分此次请求为文字水印还是图片水印，图片水印为 0，文字水印为 1，必选
bucket	-	指定水印图片存储在 BOS 的 Bucket 名称，非必选，默认当前 bucket。非当前 bucket 则要求该水印图片有公共读权限
image	-	指定水印图片存储在 BOS 的 Object 名称，需要取 url 安全的 Base64 编码后的值。图片水印嵌入必选参数。1）原图的宽高都需大于 512。2）水印图片的限制，必须是二值图像，且水印的大小为 64x64，如果不符合条件，会缩放和处理图片到服务要求
text	-	水印文字内容，需要进行 url 安全的 Base64 编码，且编码前最多不能超过 100 个字符。文字水印嵌入必选参数。文字盲水印当前支持数字 [0 - 9] 及英文大小写 [A - Z,a - z]
algorithm	0/1	添加盲水印使用的算法类型，默认为 0

示例

- 嵌入文字盲水印

> - https://doc.bce.baidu.com/bce-documentation/BOS/image.jpg?x-bce-process=image/blind-watermark,mode_1,method_embed,text_aGVsbG8=

- 提取文字盲水印

> - https://doc.bce.baidu.com/bce-documentation/BOS/image.jpg?x-bce-process=image/blind-watermark,mode_1,method_extract

- 嵌入图片盲水印

> - https://doc.bce.baidu.com/bce-documentation/BOS/image.jpg?x-bce-process=image/blind-watermark,image_YmNlLWRvY3VtZW50YXRpb24vQk9TL2ltYWdlLmpwZw==,method_embed,mode_0

- 提取图片盲水印

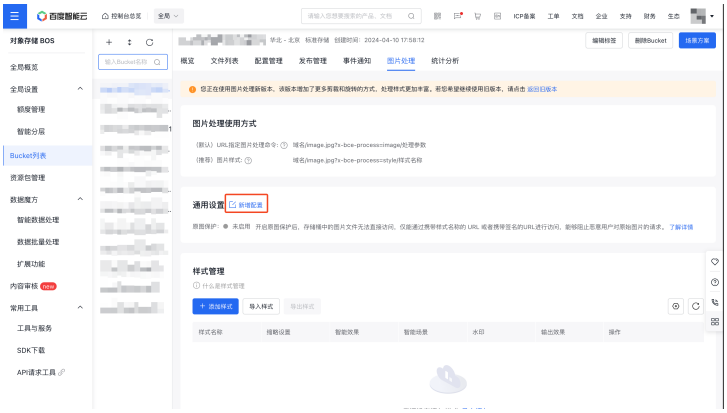
> - https://doc.bce.baidu.com/bce-documentation/BOS/image.jpg?x-bce-process=image/blind-watermark,mode_0,method_extract

图像通用设置

原图保护

概述

若您希望开启原图保护，图片拥有者在“图片处理”控制台的“通用设置”中进行配置即可。



> **注意：** 未取回的归档类型 Object 不会触发图片处理，已取回的归档类型 Object 可正常处理。

文档处理

文档预览

概述 文档预览功能基于百度智能云文档服务 DOC（Document Service），支持对多种格式文档进行数据处理，提供文档转HTML、文档转图片等文档处理服务及在线浏览服务。您无需了解文档存储、转码、分发、在线浏览等技术细节，即可快速搭建安全可靠、高可定制的文档处理平台和应用，助力在线教育、企业网盘等业务的转型升级。

> **说明：**
>
> - 支持多格式
>
> 源文档支持doc、docx、ppt、pptx、xls、xlsx、vsd、pot、pps、rtf、wps、et、dps、pdf、txt、epub多种格式。
>
> - 文件大小限制
>
> 不支持在线预览大于 200 MB的文件。
>
> - 文件页数限制
>
> 不支持在线预览大于 200 页的文件。

计费方式 文档预览服务将于 2022 年 9 月 23 日 0 点开始正式计费，您可以登录控制台-财务查看用量数据。BOS 费用会按小时进行出账，具体价格请参考 数据处理费用 。

使用方式

请求参数 操作名称：doc/preview

参数	取值	说明
tt	h5, image	• 设置文档预览转换类型，分为 HTML 和图片，默认为 HTML • h5: 将文档转换为 HTML 格式 • image: 将文档转换为 PNG 格式图片
st	doc、docx、ppt、pptx、xls、xlsx、vsd、pot、pps、rtf、wps、et、dps、pdf、txt、epub	• 指明当前 Object 格式，默认按照拓展名获取
p	1-200	• 指定文档转图片时生成第几页的图片，从 1 开始，最大支持 200 页的文档 • 转图片时此参数必填

请求示例

```
GET https://bj.bcebos.com/bucket/object.pdf?x-bce-process=doc/preview,tt_image,p_1
```

```
GET https://bj.bcebos.com/bucket/object.pdf?x-bce-process=doc/preview,tt_h5
```

```
GET https://bj.bcebos.com/bucket/object?x-bce-process=doc/preview,tt_h5,st_doc
```

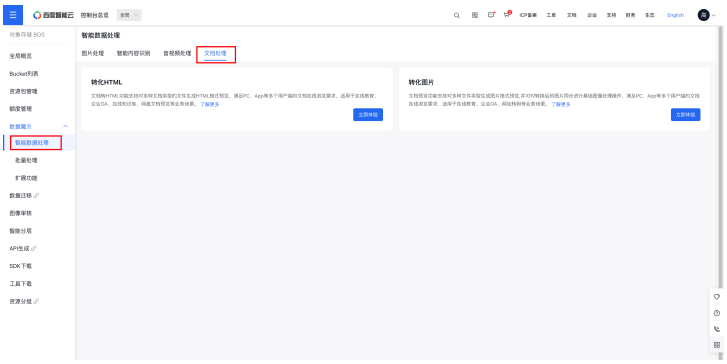
请求流程

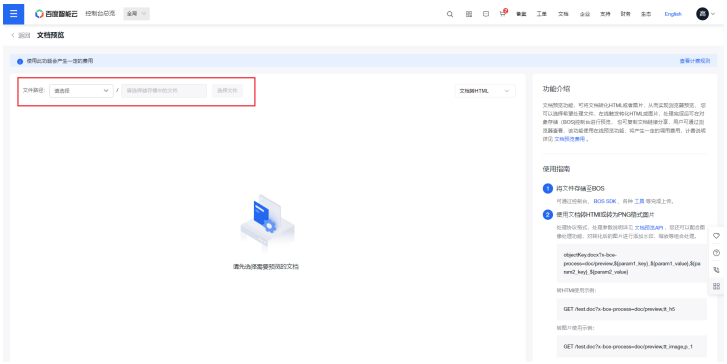
1. 客户端App或者Web端向业务服务器发起预览请求，并提供要预览的文件名。
2. 业务服务器根据请求文件进行URL签名，将签名完成的URL提供给客户端App或者Web端。
3. 客户端App或者Web端通过签名URL直接预览访问BOS文件。

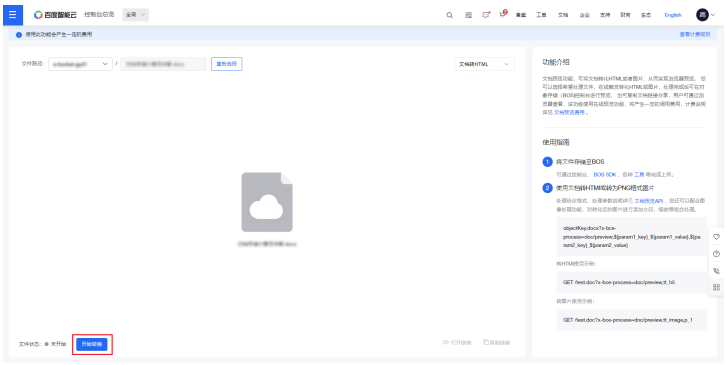
控制台体验说明 您可在BOS控制台体验文档预览功能，具体操作步骤如下。

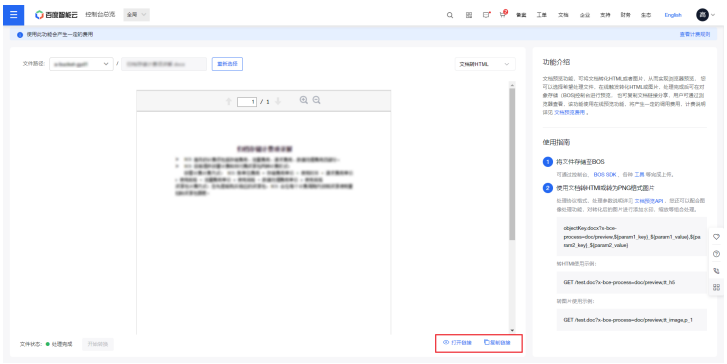
文档转HTML预览

1. 登录 BOS对象存储控制台。
2. 在左侧导航栏选择【数据魔方】->【智能数据处理】进入智能数据处理页面。
3. 选择【文档处理】，在下方 **转化HTML** 模块中点击【立即体验】进入功能页面。
4. 在顶部下拉列表中选择文档所在的存储桶，然后点击【选择文件】选择该存储桶中需要处理的文档文件。
5. 点击【开始转换】即可将文档转化为HTML。
6. 处理完成后，您可点击右下方【复制链接】或【打开链接】进行分享或在浏览器查看，实时在线预览。



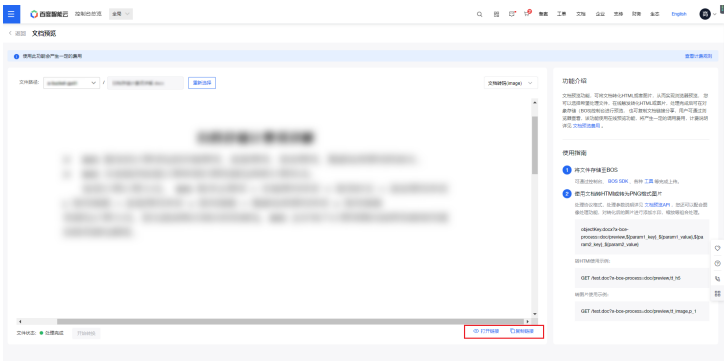






文档转图片预览

1. 登录 BOS对象存储控制台。
2. 在左侧导航栏选择【数据魔方】->【智能数据处理】进入智能数据处理页面。
3. 选择【文档处理】，在下方 **转化图片** 模块中点击【立即体验】进入功能页面。
4. 在顶部下拉列表中选择文档所在的存储桶，然后点击【选择文件】选择该存储桶中需要处理的文档文件。
5. 点击【开始转换】即可将文档转化为图片。
6. 处理完成后，您可点击右下方【复制链接】或【打开链接】进行分享或浏览器查看，实时在线预览。



内容审核

图像审核

概述

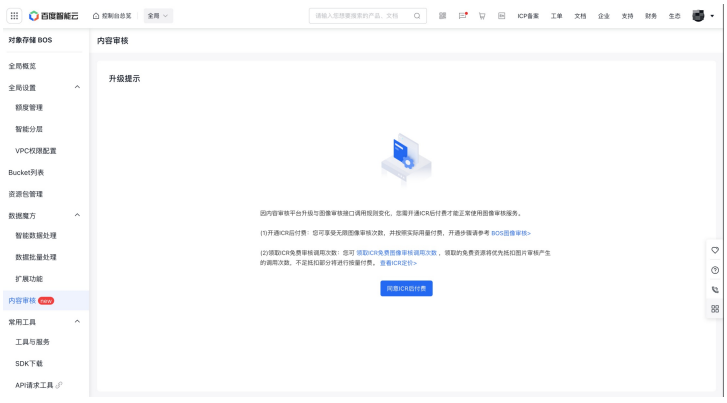
BOS 图像审核服务是百度智能云对外提供的图像智能审核服务，基于深度学习的智能内容审核方案，准确识别图片和视频中的涉黄、涉暴涉恐、政治敏感、微商广告、恶心等内容。为了方便您使用，BOS 为开发者提供 图像审核 API 或 CLI 调用的同时，也提供了界面化的图审控制台，用户可在控制台查看 BOS 审核结果中的违规图片，并按需进行人工复审。

注意事项

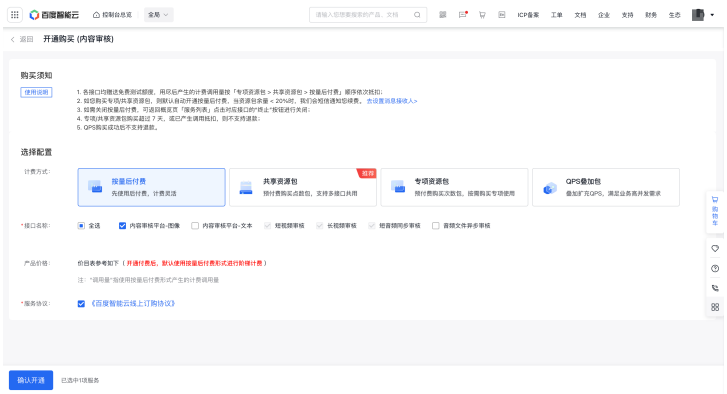
- 暂不支持子用户开启和配置 BOS 图像审核服务
- 为避免违规图片传播，BOS 会将违规图片的权限暂时设为"私有权限"，通过身份识别后当您拥有图片读取权限才可访问。
- 在已配置增量图像自动审核规则的 Bucket 内，如果图片是由生命周期策略产生，将不会被自动审核。
- 对于未取回的归档类型 Object，图审策略不会生效；已取回的归档类型 Object 可正常被审核。
- 2022年7月图像审核功能升级，将不再支持设置图像审核上限，且开通功能需要您先 开通ICR后付费 并请保持开通状态，从而确保您可享受无限的图像审核次数，保证审核业务不中断。
- 因BOS图像审核依赖 内容审核平台（ICR） 的图片审核功能，该服务条款完全遵循 ICR 条款，同时会产生图片审核接口调用费用，请参考ICR定价文档。ICR 为您提供了免费使用额度，您可前往 领取icr免费图像审核调用次数，图片审核产生的调用次数将优先抵扣该免费资源，不足抵扣部分将进行按量付费。

设置审核规则

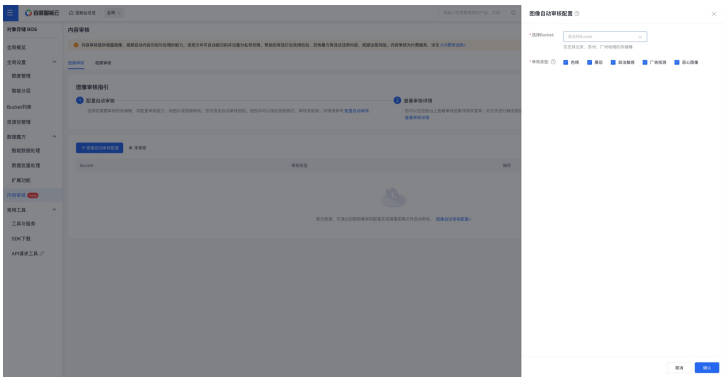
1. 登录 BOS 管理控制台。
2. 点击左侧导航栏中的【内容审核】，阅读开通内容审核服务指引。
   - 您需完成开通ICR服务同时开通其后付费才可设置图像审核规则。



3. 点击【确认开通】，完成开通 ICR 及后付费计费方式。



4. 设置审核规则。
   - Bucket 审核：选择需要审核的目标 Bucket，仅支持选择属于北京、苏州、广州地域的 Bucket。
   - 审核类型：选择需要审核哪些违规类型，目前提供了色情、暴恐、政治敏感、广告检测和恶心图像的审核能力。

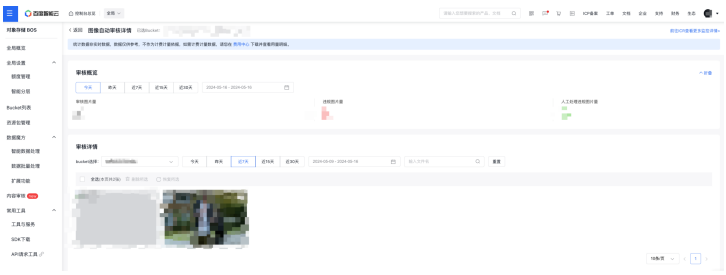


5. 点击**确认**，保存审核规则设置。

使用审核服务

查看审核数据

方式一： BOS 图审服务会监测审核您选择的 Bucket 中的所有新增图片，并将审核信息以按照【审核概览】的方式向您展示。



- **审核图片量**：代表您开启图审的 Bucket 中，经过审核的图片总量。
- **违规图片量**：代表审核结果为违规图片的数量。
- **人工处理违规图片量**：代表基于违规图片量，管理员已经人工处理过的图片数量，操作包括：删除，恢复原有权限。

方式二： 您也 可前往ICR查看更加具体的图像审核接口调用情况，可自定义时间范围，查询图像审核接口调用成功、失败情况。

违规图像复审

- 您可以通过选择 Bucket 与时间维度的，选择展示您希望人工复审的违规图片。
- 您在对违规图片按需人工复审时，可对选中的图片执行查看详细信息、删除或恢复原有权限的操作。
- 您可以批量删除或者恢复违规图片进行，特别提示**删除图像将彻底删除从 BOS 中删除，请谨慎操作**。



API 使用

开通图像审核功能后，您也可以通过调用 图像审核 API 进行图像主动审核，可很好的满足存量图像的审核需求。

视频审核

概述

BOS 视频审核服务是百度智能云对外提供的视频智能审核服务，基于深度学习的智能内容审核方案，准确识别视频中的涉黄、涉暴涉恐、政治敏感、广告、恶心等内容。为了方便您使用，BOS 为用户提供了界面化的视频审核控制台，用户可在控制台查看 BOS 审核结果中的违规内容，并按需进行人工复审。

注意事项

- 暂不支持通过子用户开启和配置 BOS 视频审核服务
- 暂不支持审核 Bucket 中的存量数据，目前支持对创建审核规则之后 Bucket 中的增量数据进行审核
- 设置视频审核规则时，如果审核范围选择前缀方式时，单条前缀中仅允许配置一个 * 字符
- 对于未取回的归档类型 Object，内容审核策略不会生效；已取回的归档类型 Object 可正常被审核
- 对视频审核规则进行编辑修改后，修改后的内容预计需要一小时后开始生效
- 因 BOS 内容审核依赖 内容审核平台（ICR）的内容审核功能，该服务条款完全遵循 ICR 条款，同时会产生内容审核接口调用费用，请参考 ICR 定价文档。ICR 为您提供了免费使用额度，您可前往领取 ICR 免费图像审核调用次数，视频审核产生的调用次数将优先抵扣该免费资源，不足抵扣部分将进行按量付费

设置审核规则

1. 登陆 BOS 管理控制台
2. 点击左侧导航栏中的内容审核，阅读内容审核服务指引。

3. 若您已经完成开通 ICR 并完成后付费，则可通过如下页面，点击【立即使用】，完成审核规则配置。

4. 设置视频审核规则，规则中的配置项包括如下：

配置项	说明
选择 Bucket	选择需要审核的目标 Bucket，目前支持选择属于北京、苏州、广州地域的 Bucket
审核范围	支持选择整个 Bucket，或匹配前缀。其中前缀匹配支持添加100条前缀条目
审核后缀	支持选择多种后缀格式的视频文件。如勾选空后缀的选项，其含义是根据 http 开头的 content-type 判断是否为视频文件
审核内容	支持增加选择对视频声音的审核
审核类型	支持按需选择多种审核类型，包括色情、暴恐、政治敏感、广告检测、恶心图像
审核截帧规则	默认截帧数是5帧/秒，支持用户自定义设置 1～360 之间的整数帧数
文件冻结设置	开启文件冻结后，将机审结果为异常文件的访问权限（ACL）更改为private（私有读）
回调设置	如开启回调设置，预计回调配置会在30分钟后生效

5. 点击确认，提交保存审核规则，审核任务开始生效。



使用审核服务

查看审核数据和结果

通过 BOS 控制台

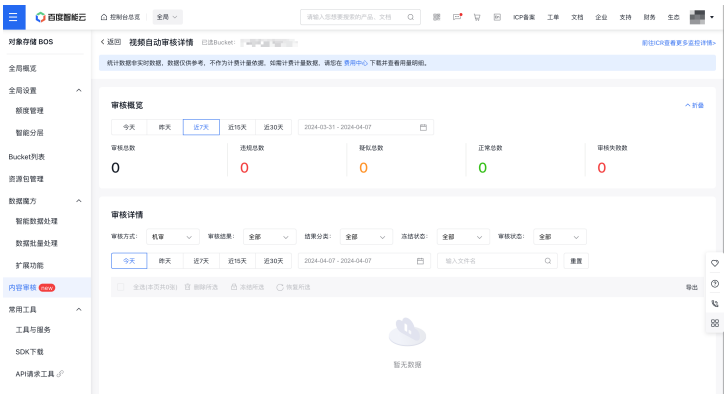
BOS 视频审核服务会审核您选择的 Bucket 中的所有新增上传的视频，并将审核信息以按照【审核概览】和【审核详情】的方式向您展示。同时也支持将审核结果导出为 csv 文件。

审核概览

- **审核总数** ：被审核的视频文件总数
- **违规总数** ：被审核的视频文件中，违规视频的数量
- **疑似总数** ：被审核的视频文件中，疑似违规的视频数量
- **正常总数** ：被审核的视频文件中，不属于违规和疑似范畴内的视频数量
- **审核失败数** ：被审核的视频文件中，无明确审核结果的视频数量

审核详情

- **审核结果** ：可选内容包括全部、正常、违规、疑似、未获取结果
- **结果分类** ：可选内容包括全部、色情、暴恐、政治敏感、广告检测、恶心图像
- **冻结状态** ：可选内容包括全部、冻结、恢复
- **审核状态** ：可选内容包括全部、审核中、审核成功、审核失败



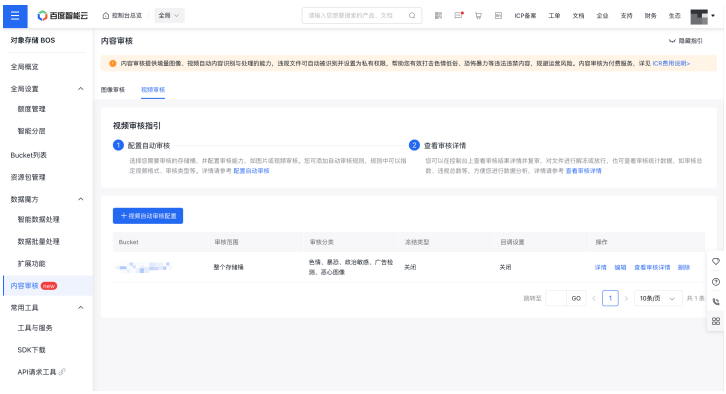
在审核详情中，可以通过点击某个具体的视频信息，进入该视频内容的详情页，在详情页中，用户可以查看该条视频的审核状态、审核结果、异常信息、URL、冻结状态、违规/疑似截图等信息。

通过 ICR 内容审核控制台

也可前往 ICR 内容审核监控报表 查看更具体的视频审核接口调用情况，可自定义时间范围和接口类型，查询视频审核接口调用详情。

编辑修改审核规则

可以通过控制台中视频审核规则列表页，选择对应的审核规则，进行编辑修改。



删除审核规则

可以通过控制台中视频审核规则列表页，选择对应的审核规则，进行删除操作，删除完成后，将停止对增量视频文件的审核。

API参考

简介

百度对象存储BOS(Baidu Object Storage)，是百度智能云对外提供的稳定、安全、高效以及高扩展存储服务，支持文本、多媒体、二进制等任何类型的数据存储。数据多地域跨集群的存储，以实现资源统一利用，降低使用难度，提高工作效率。用户可以通过本文档提供的简单的RESTful API接口，进行资源管理以及数据上传下载等操作。

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的方法。也可以通过可视化API调试工具 - 示例代码中心，进行学习与调用测试。

服务域名

百度智能云开放了多区域，BOS支持的区域及对应域名请查看获取访问域名。

规范化字符串(UrlEncode)

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%20%E6%B5%8B%E8%AF%95。

> **说明：** 下面这段sample代码以java为例演示如何进行UrlEncode。其中input为输入字符串，encodeSlash用来控制是否对"/"进行编码。在有些情况下，做UrlEncode时需要忽略"/"，具体情况可以参见“编码规范”。

```
public static String uri_encode(CharSequence input, boolean encodeSlash) {
  StringBuilder result = new StringBuilder();
  for (int i = 0; i < input.length(); i++) {
  char ch = input.charAt(i);
  if ((ch >= 'A' && ch <= 'Z') || (ch >= 'a' && ch <= 'z') || (ch >= '0' && ch <= '9') || ch == '_' || ch == '-' || ch == '~' || ch == '.') {
     result.append(ch);
     } else if (ch == '/') {
        result.append(encodeSlash ? "%2F" : ch);
     } else {
        result.append(toHexUTF8(ch));
     }
  }
  return result.toString();
}
```

编码规范

- 可解析内容，所有request/response body内容目前均使用UTF-8编码，后续会支持更多encoding类型。
- 在请求时，需要对以下做UrlEncode：
  - Objectname，其中，Resource做UrlEncode的时候需要忽略“/”。
  - Querystring的Value。
  - x-bce-copy-source（忽略“/”）。
  - 自定义Meta:Meta Value只支持可见的ASCII字符，如果需要其它的字符，推荐使用UrlEncode处理。

请求语法风格

BOS API的请求语法支持不带“/v{version}/”的风格，也兼容“/v{version}/”的风格。以GetObject为例，下述两种写法的API调用方式都是正确的。

- 不带/v{version}/的请求语法：GET /object HTTP/1.1
- 带/v{version}/的请求语法：GET /v1/bucket/object HTTP/1.1

资源数目及大小规范

- 用户可有的Bucket数目100。如果需要更多bucket，可以通过工单系统申请。
- Bucket中的Object数不限。
- 支持单个Object最大为48.8TB。
- 用户请求Header大小不得超过8KB。其中用户自定义Meta（x-bce-meta-）不得超过2KB；Meta值不能为空，且只支持ASCII格式。

命名规范

- Bucket格式要求：
  - 只能包括小写字母，数字和连字符“-”。
  - Bucket名开头和结尾必须是小写字母或数字。
  - 长度为3~63位。
- Object格式要求：
  - 长度不能超过1024个字节的UTF-8字符。

日期与时间

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，凡是HTTP标准中规定的表示日期和时间字段用GMT，其他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：

- 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。
- 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。
- 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。

接口规范

请求响应格式标准

- HTTP请求，Querystring中参数的Key，为首字母小写的驼峰方式。如 upLoadId，partNumber等。
- 所有用户自定义Meta，以x-bce-meta-*的形式放Header中，自定义Meta总大小不得超过2K。x-bce-meta-*的Key会被Server端统一按照小写进行处理。

例如：用户使用PutObject接口上传了x-bce-meta-DeMo:value，Server端会统一按照小写x-bce-meta-demo:value进行处理，用户在使用GetObject接口时，Sever端的返回值为x-bce-meta-demo:value。

- 除RFC2616规定的标准Header外，其他Header以x-bce-*的形式定义。
- BOS的RESTful API支持仅支持JSON形式。
- 所有JSON中，Key均为首字母小写的驼峰方式。
- 每个请求响应中均带有x-bce-request-id和x-bce-debug-id这两个Header。
- Header中Date、Content-MD5、Content-Type、Content-Length等相关字段遵守RFC 2616约束。
- 依据HTTP协议的规定，Content-MD5既要做MD5也要进行Base64编码，其计算方法如下：

  Content-MD5 = "Content-MD5" ":" md5-digest md5-digest = <base64 of 128 bit MD5 digest as per RFC 1864>

公共请求头

名字	类型	描述
Authorization	String	用于验证请求合法性的认证信息。更多参见鉴权认证
Content-Length	String	RFC2616中定义的HTTP请求内容的长度。
Content-Type	String	RFC2616中定义的HTTP请求内容的类型。
Content-MD5	String	RFC2616定义的HTTP请求内容的MD5摘要，可以通过携带该字段来验证保存在BOS侧的文件和用户预期的文件是否一致。
Date	String	HTTP 1.1协议中规定的GMT时间，如Wed, 06 Apr 2016 06:34:40 GMT。
Host	String	访问Host值，取值为BucketName.bj.baidubce.com。
x-bce-date	String	当前时间，遵循ISO8601规范，格式如2016-04-06T08:23:49Z。

公共返回头

名字	类型	描述
Content-Length	String	RFC2616中定义的HTTP请求内容长度。
Content-Type	String	RFC2616中定义的HTTP请求内容的类型。
Content-MD5	String	RFC2616定义的HTTP请求内容的MD5摘要，可以通过携带该字段来验证保存在BOS侧的文件和用户预期的文件是否一致。
Connection	String	服务器是否断开连接，取值为close或者keep-alive。
Date	String	HTTP 1.1协议中规定的GMT时间，如Wed, 06 Apr 2016 06:34:40 GMT。
eTag	String	Object的HTTP协议实体标签。eTag (entity tag) 在每个Object生成的时候被创建，用于标识一个Object的内容，eTag值可以用于检查Object内容是否发生变化。
Server	String	服务器的名字，取值为BceBos。
x-bce-request-id	String	由BCE BOS创建，是请求BceBos的唯一标识。
x-bce-debug-id	String	由BCE BOS创建，用于帮助排除故障的标识ID，如果在使用BOS过程中遇到问题，可以在工单中提供该字段便于快速定位问题。

不同存储类型API接口的差异

文件的存储类型是基于object实现的，低频存储、冷存储和归档存储的API实现时通过给object增加一个storage class属性来实现。storage class属性值为STANDARD、STANDARD_IA (infrequent access)、COLD和ARCHIVE，分别代表标准存储、低频存储、冷存储和归档存储。

以下低频/冷存储/归档存储API接口增加storage class属性，其中归档存储类型不支持AppendObject接口：

- PutObject、InitiateMultipartUpload、CopyObject、AppendObject、PostObject接口在请求头域中设置storage class属性参数x-bce-storage-class。

请求头域

名称	类型	描述	是否必需
x-bce-storage-class	String	指定BOS的对象的存储类型，目前支持STANDARD、STANDARD_IA、COLD和ARCHIVE。	否，默认为STANDARD。

- GetObject和GetObjectMeta在响应头域中返回x-bce-storage-class。

  响应头域

名称	类型	描述
x-bce-storage-class	String	BOS的对象的存储类型，目前支持STANDARD、STANDARD_IA、COLD和ARCHIVE。

  另外，当对象为归档存储类型，并且处于正在取回或已经取回状态时，GetObjectMeta会返回头域“x-bce-restore”，表示对象的取回状态。

- ListObjects、ListMultipartUploads和ListParts在响应元素中返回storageClass。

  响应元素

名称	类型	描述
storageClass	String	BOS的对象的存储类型，目前支持STANDARD、STANDARD_IA、COLD和ARCHIVE。

错误码

错误信息格式

当用户访问BOS出现错误时，BOS会返回给用户相应的错误码和错误信息，便于用户定位问题，并做出适当的处理。

系统返回错误信息格式如下：

```
{
  "code":"NoSuchKey",
  "message":"The resource you requested does not exist",
  "requestId":" 4db2b34d-654d-4d8a-b49b-3049ca786409"
}
```

BOS错误码

其中code、message字段定义如下：

错误码（code）	消息（message）	描述	HTTP状态码
success	success	成功	200 OK
success	Both this request and DPF request executed successfully.	请求执行成功	200 OK
success	This request executed successful, but the DPF request is failed.	请求执行成功，但DPF执行失败	203 Non-Authoritative Information
success	This request executed successful, but the third-party platform executed failed.	请求执行成功，第三方执行失败	203 Non-Authoritative Information
success	This request executed successful, but your dpf format is invalid.	请求执行成功，DPF格式无效	203 Non-Authoritative Information
success	Callback failed.	回调失败	203 Non-Authoritative Information
success	Encrypt failed.	加密失败	203 Non-Authoritative Information
success	Payload too large.	有效载荷过大	203 Non-Authoritative Information
success			204 No Content
InvalidEncryptionAlgorithm	The specified encryption algorithm is invalid	加密算法无效	400 Bad Request
InvalidEncryptionCustomerKey	The specified encryption customer-key is invalid, encrypt object by customer key md5 not match	加密密钥无效，密钥md5不匹配。	400 Bad Request
	The specified encryption kms mkid is invalid, please check its		

InvalidEncryptionKmsMkid	The specified encryption KMS mkid is invalid, please check its status	指定的加密KMS MKID无效，请检查其状态	400 Bad Request
InvalidTargetType	The specified object key is symlink, target object of symlink also is symlink.	软链接文件不支持再次设置软链接	400 Bad Request
InvalidEncryptionScheme	The specified encryption scheme is invalid, encrypt object by customer key must use https request	加密Object必须使用HTTPS请求	400 Bad Request
success	This request executed successful, but the dpf executed failed because the assumerole configuration you set is invalid.	请求执行成功，但DPF执行失败,因为处理用户权限失败。	203 Non-Authoritative Information
AccessDenied	Access Denied.	拒绝访问	403 Forbidden
AccountOverdue	Your request is denied because there is an overdue bill of your account.	用户欠费	403 Forbidden
AccessDeniedBySourceUrl	Access Denied By Source Url.	源URL拒绝访问	403 Forbidden
BadDigest	The Content-MD5 or x-bce-content-sha256 or x-bce-content-crc32 you specified did not match what we received.	摘要错误，用户指定的md5或者sha256或者crc32与服务端计算不一致。	400 Bad Request
BadDigestMD5	The Content-MD5 you specified did not match what we received.	错误的Content-MD5字段，与实际上传的数据MD5不符	400 Bad Request
BadDigestSHA256	The x-bce-content-sha256 you specified did not match what we received.	错误的x-bce-content-sha256字段，与实际上传的数据sha256不符	400 Bad Request
BadDigestCRC32	The x-bce-content-crc32 you specified did not match what we received.	错误的x-bce-content-crc32字段，与实际上传的数据crc32不符	400 Bad Request
BucketAlreadyExists	The requested bucket name is not available. The bucket namespace is shared by all users of the system. Please select a different name and try again.	Bucket已经存在	409 Conflict
ObjectLockAlreadyInProgress	The bucket lock status is already InProgress	Bucket已经为锁定状态	409 Conflict
BucketNotEmpty	The bucket you tried to delete is not empty.	试图删除一个不为空的bucket	409 Conflict
ClientSocketError	clinet socket error, maybe client close connection.	客户端连接失败,可能是客户端关闭了连接。	200 OK
CORSConfigurationDoesNotMatch	The CORS Options request is not allowed because the request does not match with the CORS Configuration of the resource.	请求资源与CORS配置不匹配	403 Forbidden
CORSNotEnabled	The CORS is not enabled for this resource.	未配置允许跨域访问	403 Forbidden
ReplicationNotEnabled	The Bucket Replication is not enabled.	跨区域数据同步未开启	404 Not Found
UserQuotaNotConfigured	The user quota does not configured	未配置用户配额	404 Not Found
ReplicationStatusError	The Bucket Replication Status is not correct. Please make sure both the source and dest bucket have no enabled replication conf and both are not the dest replication bucket of other bucket	跨区域复制源Bucket或目标Bucket已经在另一条跨区域复制规则中被指定	409 Conflict
ReplicationStatusNotEmpty	The bucket you tried to delete has an enabled replication conf or is a dest replication bucket of other bucket	待删除的Bucket开启了跨区域复制功能。	409 Conflict
InventoryStatusError	The Bucket Inventory Status is not correct. Please make sure dest bucket is same region!	文件清单状态异常。请确保目的bucket在同地域。	409 Conflict
ReplicationApiUsedInvalid	The replication api must upgrade,because you have used multi rules	跨区域复制API版本需要更新	409 Conflict
ReplicationConfDuplicate	The replication has exist with different id	跨区域复制配置重复	409 Conflict
EmptyHTTPAuthHeader	The HTTP authorization header is empty. We do not allow using Session Token with empty authorization header	sts签名请求中，缺少authorization请求header	400 Bad Request
EntityTooSmall	Your proposed upload is smaller than the minimum allowed object size.	上传的数据小于限制	400 Bad Request
EntityTooLarge	Your proposed upload exceeds the maximum allowed object size.	上传的数据大于限制	400 Bad Request
CapacityExceeded	Your total upload bytes exceeds the maximum allowed in your account.	超出账号容量限制	403 Forbidden
BucketObjectNumExceeded	Your total upload object number exceeds the maxinum in your bucket	超出Bucket文件数量限制	403 Forbidden
BucketCapacityExceeded	Your total upload bytes exceeds the maximum allowed in your bucket	超出Bucket容量限制	403 Forbidden
ObjectNumExceeded	Your total upload object number exceeds the maxinum in your account	超出账号文件数量限制	403 Forbidden
BucketQuotaNotConfigured	The bucket quota does not configured	未配置Bucket配额	404 Not Found
FetchObjectTaskIsRunning	The fetch object task you submit is running.	FetchObject进行中	200 OK
FetchObjectFailed	Fetch object failed, please check argument.	文件抓取失败	400 Bad Request
FileAlreadyExists	The object you specified already exists and can not be overwritten.	指定的文件已存在，无法覆盖	409 Conflict
InappropriateJSON	The JSON you provided was well-formed and valid, but not appropriate for this operation.	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。	400 Bad Request
InappropriateXML	The XML you provided was well-formed and valid, but not appropriate for this operation.	请求中的XML格式正确，但语义上不符合要求。适用场景同InappropriateJSON	400 Bad Request
InvalidAccessKeyId	The BCS Access Key Id you provided does not exist in our records.	提供的Access Key不存在	403 Forbidden
InvalidArgument	Invalid Argument.	无效参数	400 Bad Request
InvalidBucketName	The specified bucket is not valid.	Bucket名称不正确	400 Bad Request
InvalidCopyDestination	The copy destination is not valid	复制的目标对象无效	400 Bad Request
InvalidCORSRequest	The CORS request is not valid	跨域请求错误	400 Bad Request
InvalidDomain	The specified domain is not valid.	域名不正确	400 Bad Request
InvalidHTTPAuthHeader	The HTTP authorization header is invalid. Consult the service documentation for details.	请求头Authorization无效	400 Bad Request

InvalidImageFormat	Your image format is invalid.	图片格式不正确	400 Bad Request
InvalidObjectName	Your object key is too long.	文件名长度超过限制	400 Bad Request
InvalidTrashDirectoryName	Your trash directory name is not valid.	回收站路径不正确	400 Bad Request
InvalidDirectory	Directory file must be empty	目录文件必须为空	400 Bad Request
ObjectLockNotLocked	The object lock is not locked, can not extend retentionday	文件未被锁定，无法延长保留期限	400 Bad Request
InvalidPart	One or more of the specified parts could not be found. The part might not have been uploaded, or the specified entity tag might not have matched the part's entity tag.	找不到指定的分块。该分块可能未上传或指定的etag与分块不匹配。	400 Bad Request
InvalidPartOrder	The list of parts was not in ascending order.Parts list must specified in order by part number.	分块列表未按升序排列	400 Bad Request
InvalidPolicyDocument	The content of the form does not meet the conditions specified in the policy document.	Policy格式错误	400 Bad Request
InvalidRange	The requested range cannot be satisfied.	请求的Range范围无效	416 Range Not Satisfiable
InvalidSessionToken	The Session Token you provided is not valid. Please double check that you are using the correct Session Token obtained from STS (Security Token Service)	STS Token无效	400 Bad Request
InvalidURI	Could not parse the specified URI.	URI形式不正确	400 Bad Request
InvalidStaticWebSiteFormat	The format of index file or 404 file are not allowed.	不允许的index或者404文件名或格式。 比如：文件格式不允许，或者index与404文件同名。	400 Bad Request
InvalidJobStatus	The current job status does not supprot the operation.	当前任务状态不支持操作	400 Bad Request
InvalidStorageClass	The storage class specified is invalid, please check argument and consult the service documentation.	指定的存储类型错误	400 Bad Request
NotSupportStorageClass	The storage class specified is not support, please check argument and consult the service documentation.	指定的存储类型错误	501 Not Implemented
NotAllowShortenRetentionday	Could not shorten retentionday	不允许缩短合规保留期限	400 Bad Request
InvalidVersion	The API version specified was invalid.	API版本错误	404 Not Found
RequestRateLimitExceeded	Your request rate is too high. We have put limits on your bucket.	请求速度太快并触发限速	429 Too Many Requests
MalformedJSON	The JSON you provided was not well-formed.	JSON格式不正确	400 Bad Request
MalformedXML	The XML you provided was not well-formed or did not validate against our published schema.	XML格式不正确	400 Bad Request
MaxMessageLengthExceeded	Your request was too big.	请求体过大	400 Bad Request
MetadataTooLarge	Your metadata headers exceed the maximum allowed metadata size.	超出metadata大小限制	400 Bad Request
MethodNotAllowed	The specified method is not allowed against this resource.	请求方法错误	405 Method Not Allowed
MirrorFailed	Mirroring failed, please check your mirror configuration.	镜像回源失败	424 Failed Dependency
MissingContentLength	You must provide the Content-Length HTTP header.	未提供Content-Length	411 Length Required
MissingDateHeader	Request must have a "date" or "x-bce-date" header.	请求中找不到date和x-bce-date两者之一；如果同时添加了date和x-bce-date，则x-bce-date的优先级高于date，二者的格式请参考：\ndate: Wed, 06 Apr 2016 06:34:40 GMT\nx-bce-date: 2016-04-06T08:23:49Z	400 Bad Request
NoSuchBucket	The specified bucket does not exist.	Bucket不存在	404 Not Found
NoSuchInventoryConfiguration	The specified inventory configuration does not exist.	清单配置不存在	404 Not Found
NoSuchCORSConfiguration	The CORS configuration does not exist.	未配置跨域规则	404 Not Found
NoLifecycleConfiguration	The lifecycle configuration does not exist.	未配置生命周期规则	404 Not Found
NoProcessConfiguration	The process configuration does not exist.	数据处理配置不存在	404 Not Found
NoNotificationConfiguration	The notification configuration does not exist.	未配置事件通知规则	404 Not Found
NoReplicationConfiguration	The Replication configuration does not exist.	未配置跨区域复制	404 Not Found
NoSuchKey	The specified key does not exist.	不存在该Object	404 Not Found
NoSuchStyle	The specified style does not exist.	图片样式不存在	404 Not Found
NoSuchObjectLock	The specified object lock does not exist.	合规保留配置不存在	404 Not Found
NoSuchJob	The job does not exist.	批处理任务不存在	404 Not Found
NoSuchUpload	The specified multipart upload does not exist. The upload ID might be invalid, or the multipart upload might have been aborted or completed.	该uploadId所对应的三步上传不存在。uploadId错误或者三步上传已中止或者已完成。	404 Not Found
NoCopyrightProtectionConfiguration	The copyright protection configuration does not exist.	未配置版权保护	404 Not Found
NoSuchBucketEncryption	The bucket is not encrypted.	该Bucket未加密	404 Not Found
NoSuchBucketTrashDirectory	The bucket does not activate trash, or trash has been turned off	未开启回收站	404 Not Found
NoSuchBucketAlias	The bucket does not activate alias, or bucket alias has been turned off	Bucket别名不存在	404 Not Found
NoSuchBucketDefaultStyle	The bucket does not activate default style, or bucket default style has been turned off	Bucket默认样式不存在	404 Not Found
NoSuchBucketStaticWebSiteConfig	The static web site configuration does not exist.	Bucket 没有开启静态托管	404 Not Found
StaticWebSiteIsDisable	Static web site is disabled or not implemented.	静态网站托管功能被禁止。 如：即BOS不允许此region的bucket 开启静态网站托管功能，或者静态网站托管功能未上线。	501 Not Implemented
NotImplemented	A header you provided implies functionality that is not implemented.	系统未实现	501 Not Implemented
NotSymlink	The specified object key is not symlink.	文件不是软链接类型	400 Bad Request
SymlinkTargetNotExist	The specified object key is symlink, target object of symlink does not exist.	软链接目标文件不存在	404 Not Found

PreconditionFailed	The specified If-Match header doesn't match the header you provided.	指定的If-Match请求头与用户提供的不匹配。	412 Precondition Failed
OptInRequired	You must active bos service.	BOS服务未开启，请开通BOS服务且确保账户未欠费。	403 Forbidden
RequestExpired	Request has expired. Timestamp date is XXX.	请求的时间戳过期。请求超时，XXX要改成x-bce-date的值。如果请求中只有Date，则需要将Date转换为本规范指定的格式。	403 Forbidden
RequestTimeout	Your socket connection to the server was not read from or written to within the timeout period.	请求超时	408 Request Timeout
RequestTimeTooSkewed	The difference between the request time and the server's time is too large.	请求时间与服务器时间差异过大	403 Forbidden
ObjectTooSmallToProcess	Object is too small to process.	文件过小，无法进行处理	400 Bad Request
ObjectTooLargeToProcess	Object is too large to process.	文件过大，无法进行处理	400 Bad Request
VideoSnapshotFail	Handle video snapshot fail	视频截帧失败	400 Bad Request
VideoRealtimeCodecFail	Handle video codec fail	视频转码失败	400 Bad Request
VideoInfoFail	Handle video info fail	获取视频信息失败	400 Bad Request
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.	Authorization头域中附带的签名和服务端验证不一致	403 Forbidden
ServiceUnavailable	Please reduce your request rate.	服务不可用	503 Service Unavailable
SlowDown	Please reduce your request rate.	请求过于频繁	503 Service Unavailable
TooManyBuckets	You have attempted to create more buckets than allowed.	创建的Bucket数量超过限制	400 Bad Request
TooManyConfigurations	You have attempted to create more configurations than allowed.	创建配置超过限制	400 Bad Request
TooManyStyles	You have attempted to create more styles than allowed.	创建样式数量超过限制	400 Bad Request
TooManyJobs	You are attempting to create too many jobs.	创建批处理任务数量超过限制	400 Bad Request
TooManyMultiUploads	You have attempted to do more multipart uploads than allowed.	上传的分块数量超过限制	400 Bad Request
InvalidContentLength	Invalid Content-Length or Transfer-Encoding.	Content-Length不正确	400 Bad Request
ReadBodyError	Read http body error, please try again.	读取请求内容错误，请重试	400 Bad Request
DomainAlreadyExists	The specified domain is bound to %s already.	域名已被绑定到其他bucket	409 Conflict
TooManyDomains	You have attempted to bind to more buckets than allowed.	域名数据超过限制	400 Bad Request
NoSuchDomain	The specified domain does not exist.	域名不存在	404 Not Found
NoSuchMirroring	The specified mirroring configuration does not exist.	未配置镜像回源规则	404 Not Found
OffsetIncorrect	Offset not equal to current object length	追加Appendable Object时，<OffsetSize>值不等于已上传的Object的大小或者<OffsetSize>值不为0但Object不存在	409 Conflict
ObjectUnappendable	The object can not be append	不能对非Appendable的Object做AppendObject操作	403 Forbidden
NotAllowCopy	This object not allow to copy to another one	文件不允许复制	400 Bad Request
NotAllowModify	Permission not match, not allow modify object data	文件不允许修改	403 Forbidden
ObjectIllegal	The Object is illegal	文件不合规	403 Forbidden
OnlyAllowModify	Permission not match, you only have modify permission, so you can not upload a new object	权限不匹配，仅支持修改	403 Forbidden
ObjectAclNotPass	The object acl does not pass.	文件权限不匹配	403 Forbidden
ObjectBanned	Your object contains sensitive information. We have restricted your resources.	文件已被封禁	403 Forbidden
ObjectLocked	Your object has been locked.	文件已被锁定，无法进行操作	423 Locked
ObjectLockConflict	You cannot lock/unlock a banned object.	文件已被封禁	403 Forbidden
ObjectAclNotExists	The object acl does not exist.	未配置文件权限规则	404 Not Found
InternalError	We encountered an internal error. Please try again.	内部错误	500 Internal Server Error
PartialContentError	Errors occurred at storage system and return partial content.	请求时发生错误并返回部分内容	510 Not Extended
NoStorageAnalysisConf	No storage analysis conf is set.	未设置存储分析配置	404 Not Found
StorageAnalysisConfIdConflict	The confId you provide is confict with existing confs.	存储分析配置ID与现有配置冲突	409 Conflict
ObjectImmutable	The object is immutable	已经开启合规保留，文件无法进行删除或修改操作	409 Conflict
ObjectRenameConflict	Object rename conflict occurred, please retry it later or avoid concurrent update.	重命名文件冲突	409 Conflict
InvalidStorageAnalysisPrefix	The storage analysis prefix you provide is invalid.	存储分析配置中的文件前缀无效	400 Bad Request
StorageAnalysisConfigNumExceeded	Storage analysis confs have exceeded the max allow num.	存储分析配置数量超出限制	400 Bad Request
SqlSyntaxError	The sql expression in SelectRequest is invalid, it contains unsupported syntax or incorrect syntax.	SQL语法错误	400 Bad Request
InvalidSqlFields	The field in sql expression is invalid, it contains operators or other literals in SELECT clause.	SQL表达式中字段无效	400 Bad Request
InvalidSqlBinaryExpr	The binary expression in sql is invalid, left operand is incompatible with right operand.	SQL错误	400 Bad Request
SqlFieldsNumExceedLimit	The number of fields in sql expression exceed the maximum allowable number limit.	SQL中的字段数量超过了允许的最大数量限制	400 Bad Request
AggregateInvalidField	The field of aggregate function in the sql is invalid, it is not numeric data type.	SQL中聚合函数错误，使用字段非数值类型。	400 Bad Request
InvalidSqlJsonPathDepth	The depth of json path is invalid, it exceed the maximum limit or less than 1.	SQL错误，json路径的深度无效，超过最大限制或者小于1。	400 Bad Request
SqlSourceNumExceedLimit	Only one source is supported in sql expression.	SQL表达式中仅支持一个数据源	400 Bad Request
RecordTooLarge	The length of record in csv file exceeds limit.	CSV文件长度超过限制	400 Bad Request

FieldNotExist	The field in SQL not exist in file	SQL字段不存在	400 Bad Request
InappropriateJson	The format of json object is inappropriate.	JSON格式错误	400 Bad Request
HeaderNotExist	No valid header info is found in csv file.	CSV文件格式错误	400 Bad Request
DecompressError	Error occurred when decompress object.	文件解压失败	400 Bad Request
InvalidFileType	The type of file is invalid, only JSON and CSV are supported.	文件类型错误，仅支持JSON及CSV	400 Bad Request
DataOverflowsType	The aggregate result of data overflows data type limit.	数据聚合结果超出了数据类型的限制	400 Bad Request
InvalidSqlSource	Invalid data source in sql expression SELECT clause, check the sql source format of csv or json object.	请检查CSV或JSON对象的SQL源格式	400 Bad Request
InvalidSqlLimitValue	The value of LIMIT clause in sql expression is invalid, it must greater than zero.	SQL表达式中LIMIT值无效，必须大于零	400 Bad Request
InvalidSqlNotOperator	The operand of NOT operator is invalid, NOT operator only can be followed by BETWEEN/IN/LIKE.	SQL错误,"NOT"操作需要跟随"BETWEEN/IN/LIKE"之后。	400 Bad Request
InvalidSqlBetweenOperator	The operand of BETWEEN operator is invalid, BETWEEN should with AND.	SQL中"BETWEEN"使用错误	400 Bad Request
InvalidSqlInOperator	The values of IN operator are invalid, check values in (……).	SQL中"IN"操作使用错误	400 Bad Request
InvalidSqlIsOperator	The operand of IS operator is invalid, IS operator can only be followed by NULL or NOT NULL.	SQL中"IS"操作使用错误	400 Bad Request
InvalidSqlLikeOperator	The operand of LIKE operator is invalid, LIKE operator can only be followed by string.	SQL中"LIKE"操作使用错误	400 Bad Request
InvalidSqlFunction	The function in sql expression is invalid, check the function parameters number and type.	SQL表达式中的函数无效，请检查函数参数的数量和类型	400 Bad Request
InvalidExpressionParameter	The value of expression parameter in SelectRequest is invalid, please base64 decode it and do not exceed max length limit.	Select扫描请求expression参数错误，请进行base64编码并不超过长度限制。	400 Bad Request
InvalidExpressionTypeParameter	The value of expressionType parameter in SelectRequest is invalid, only SQL expressions are supported.	Select扫描请求expressionType参数错误	400 Bad Request
InvalidCompressionTypeParameter	The value of compressionType parameter in SelectRequest is invalid, only GZIP and NONE are supported.	Select扫描请求compressionType参数错误	400 Bad Request
InvalidJsonTypeParameter	The value of Json type parameter in SelectRequest is invalid, only DOCUMENT/LINES are supported.	Select扫描请求json type参数错误	400 Bad Request
InvalidQuoteFieldsParameter	The value of quote fields parameter in SelectRequest is invalid, only ALWAYS/ASNEEDED are supported.	Select扫描请求quote fields参数错误	400 Bad Request
InvalidSelectRequestJsonBody	Some parameters in SelectRequest json body is invalid, please check all optional and required parameters.	Select扫描请求参数错误	400 Bad Request
StorageAnalysisConfigNumExceeded	Storage analysis confs have exceeded the max allow num.	存储分析配置超过最大限制	400 Bad Request
NotArchiveObject	Object is not archive storage class.	当前object不是归档类型文件	400 Bad Request
DuplicationRequest	Duplication request.	重复请求	400 Bad Request
RestoreAlreadyInProgress	The object is restoring.Don't restore the object again.	您已经成功调用过RestoreObject接口，BOS服务端正在执行解冻操作。请不要重复执行RestoreObject	409 Conflict
RestoreAlready	The object is restored.	归档文件处于解冻状态，无需重重复调用	409 Conflict
ObjectLockAlreadyLocked	The object lock status is locked.	合规保留已开启	409 Conflict
ObjectLockAlreadyExpired	The object lock status is expired.	合规保留已过期	409 Conflict
CasUpdateConflict	An update conflict occurred, please retry it later or avoid concurrent update.	并发更新冲突，请稍后重试或者避免并发更新。	409 Conflict
InvalidObjectState	The object is freezed or restoring.	归档类型文件状态不合法，归档文件冰冻中或取回中。	403 Forbidden

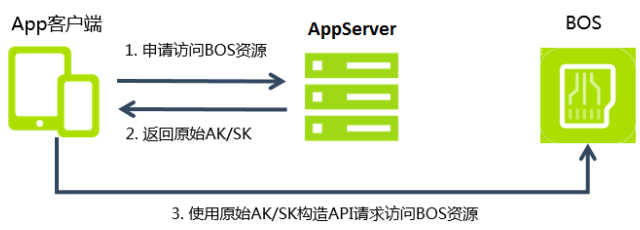
访问控制

签名认证访问

百度智能云采用统一的API鉴权认证机制，详情请见 鉴权认证机制。

原始AK/SK鉴权

- 原始AK/SK是指您在注册BOS时，系统自动分配给您的 AK（Access Key ID）/SK（Secret Access Key），主要用于对用户的调用行为进行鉴权和认证，相当于百度智能云API专用的用户名及密码。您向BOS发送的每个请求，都需要通过鉴权认证通过后，BOS才会处理您的请求。
- 移动端场景使用原始AK/SK鉴权的交互过程如下图所示：



- 将您的原始AK/SK下发到移动端势必会存在一定的安全风险，因此，**原始AK/SK鉴权方式只推荐您在测试过程中使用，正式提供服务时推荐您使用STS方式鉴权。**

> **注意：** 当您在授权的时候，建议严格遵循最小权限原则，限定用户执行受限的操作（如仅授权读操作）并设置访问指定前缀的资源，避免授予过大的权限，导致预期外的越权操作，造成数据安全风险。

临时授权访问

STS简介

BOS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。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访问百度智能云资源。

> **说明：** 这里的第三方用户主要是指您所开发的应用程序的终端用户，例如，若您是移动App开发者，您的App终端用户我们称为第三方用户。

以移动客户端访问BOS服务为例：

如您是移动App开发者，您已开通BOS服务，并且需要为您的App客户端提供访问BOS服务的权限，那么您**有如下两种方式为移动客户端实现鉴权：**

STS鉴权

为了让移动终端能够便捷、安全地享受到BOS服务，百度智能云研发了STS服务，来对移动场景下的安全需求进行支持。您无需开放自己的AK/SK，仅通过STS即可实现临时授权。

使用STS为APP客户端临时授权的流程如下图所示：



1. 申请访问BOS权限。

   App客户端访问BOS资源，需向AppServer申请访问权限。已经开通BOS服务的AppServer可以针对不同的App终端用户进行授权及访问资源限制。

2. 申请临时访问凭证。

   AppServer需要向STS服务申请一个临时访问凭证。AppServer通过调用STS的GetSessionToken()接口进行申请，同时需要指定资源访问权限和过期时间。GetSessionToken()的调用方法请参考GetSessionToken接口，STS服务详情请参考STS相关接口文档。

3. 返回STS凭证给AppServer。

   STS处理申请后，返回给AppServer一份临时的身份凭证（Credential），包括临时访问密钥AK/SK， SessionToken、失效时间和AppServer的用户ID等信息。AppServer可以缓存STS凭证，当多个App客户端需要的权限相同时，可以直接把缓存的凭证颁发给客户端。

4. 返回STS凭证给App客户端。

   AppServer将收到的STS凭证下发给App客户端，App客户端可以缓存STS凭证。当凭证失效时，App客户端需要向AppServer重新申请有效访问凭证。

5. App客户端使用STS凭证访问BOS资源。

   App客户端设备可以直接使用返回的STS凭证构造API请求直接访问BOS资源，BOS会感知STS访问凭证，并会依赖STS服务来验证访问凭证，正确响应用户请求。

使用STS的优点：

- 您不需要向第三方用户透露您的管理账号或AK/SK信息，您只需要向STS申请一个临时访问凭证并颁发给第三方用户即可，且这个访问凭证拥有的权限和有效期均可由您自己去定义。
- 您不需要关注该权限的撤销问题，临时访问凭证在过期后会自动失效。

GetSessionToken接口

接口描述

GetSessionToken接口用于临时授权访问时，请求返回一份临时的访问凭证（Credential）。

权限说明

请求发起人需要具有合法的AccessKeyID和SecretAccessKey才能发起请求。

请求

- 请求语法

```
 POST /v1/sessionToken?durationSeconds=DurationSeconds HTTP/1.1
Host: sts.bj.baidubce.com
Date: <Date>
Authorization: <AuthorizationString>

{
 "id":"userid",
 "accessControlList": [
  {
   "eid":<eid>,
   "service":"bce:bos",
   "region":"bj",
   "effect": "Allow",
   "resource": ["Bucketname/*"],
   "permission": ["READ"]
  }
 ]
}
```

- 请求参数

名称	参数位置	描述	默认值	是否必选
durationSeconds	Query参数	**临时访问凭证的有效时长，即一次请求STS后，在有效时长内可使用此临时访问凭证多次访问BOS资源，不需要重复请求STS。**数据类型：int。单位为秒，最长可指定为129600秒（36小时）。	43200（12小时）	否

- 请求头域

名称	数据类型	描述	默认值	是否必选
Authorization	string	认证字符串，计算方法请参考生成认证字符串。	-	是

- 请求元素

本接口的请求元素是权限控制的主要部分。请求元素由一条或多条acl配置项组成，每条acl配置项间相互独立。

名称	数据类型	描述	默认值	是否必选	父节点
accessControlList	array	标识acl主体的开始，由一或多组acl配置项组成。	-	是	-
effect	string	指定与该条acl配置项匹配的Request能否执行，取值为Allow或Deny。Allow表示可以执行；Deny表示拒绝执行。	-	是	accessControlList
eid	string	标识acl配置项的id 。	-	否	accessControlList
id	string	acl的标识符。	-	否	-
permission	array	acl配置项所影响的权限,可选值为READ、WRITE、LIST、MODIFY、FULL_CONTROL和GetObject等细粒度权限。		是	accessControlList
region	string	acl配置项影响的区域，"*"表示所有区域。	-	是	accessControlList
resource	array	acl配置项所影响的资源，支持通配符。如：<BucketName>/<ObjectKey>或<BucketName>/xxx*	-	是	accessControlList
service	string	acl配置项影响的服务组件，"*"表示所有服务。	-	是	accessControlList

说明：

- 由于GetSessionToken的响应结果会包含STS凭证，强烈建议通过HTTPS协议调用。
- service：以BOS服务为例，取值为"service":"bce:bos"。
- resource：当"service":"bce:bos"时，resource支持Bucket和Object。
  - 当resource为Bucket时，其取值为<BucketName>。例如，"resource"：["sts-bucket-1"]；
  - 当resource为Object时，其取值为<BucketName/ObjectName>。例如，"resource"： ["sts-bucket-1/object1"]；
  - 此外，resource还支持通配符 `*`，表示Bucket下所有Object。例如："resource"： ["sts-bucket-1/*"]。
- permission： BOS服务支持READ/WRITE/LIST/MODIFY/GetObject权限。每个权限对应一组API请求如下：
  - **READ权限对应的操作有：**
    - GetBucketLocation
    - HeadBucket
    - GetObject
    - GetObjectMeta
    - ListParts
  - **WRITE权限对应的操作有：**
    - PutObject
    - PostObjcet
    - InitiateMultipartUpload
    - UploadPart
    - CompleteMultipartUpload
    - AppendObject
    - AbortMultipartUpload
    - DeleteObject
    - DeleteMultipleObjects
    - FetchObject
  - **LIST权限对应的操作有：**
    - ListObjects

    - ListMultipartUploads
- `MODIFY`权限对应的操作有：
    - PutObject
    - PostObjcet
    - AppendObject
    - RenameObject
    - CopyObject
    - FetchObject
    - PutSymlink
- `GetObject`权限对应的操作有：
    - GetObject
    - GetObjectMeta

注意：

- 如在请求元素中不指定acl，则默认返回的临时权限Credential与您当前拥有的READ/WRITE权限相同，授权给第三方用户后，可能会给您的账户资源带来风险，建议您在临时授权时明确指定acl，规避风险。如果请求Body为空，则默认返回的STS凭证与您当前拥有的权限相同。
- 在您指定acl时，建议严格遵循 最小权限原则，限定用户执行受限的操作（permission字段）并设置访问指定前缀的资源（resource字段），避免造成数据安全风险。

响应

- 响应头域

  本接口只用到了公共响应头。

- 响应元素

名称	数据类型	描述
accessKeyId	string	用于STS凭证访问的AK。
expiration	date	访问失效时间。
secretAccessKey	string	用于STS凭证访问的SK。
sessionToken	string	SessionToken，使用STS凭证访问时必须携带。
userId	string	您的用户ID。

示例

- 请求示例

```
 POST /v1/sessionToken?durationSeconds=DurationSeconds HTTP/1.1
Host: sts.bj.baidubce.com
Date: Wed, 06 Apr 2016 06:34:40 GMT
Authorization: AuthorizationString
Content-type:application/json
Content-Length:178

{
 "id":"10eb6f5ff6ff4605bf044313e8f3ffa5",
 "accessControlList": [
  {
   "eid":<eid>,
   "service":"bce:bos",
   "region":"bj",
   "effect": "Allow",
   "resource": ["sts-bucket-1-140683201300192-144784797246238452047 8/*"],
   "permission": ["READ"]
  }
 ]
}
```

- 响应示例

```
 HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Content-Type: application/json;charset=UTF-8
Date: Wed, 06 Apr 2016 06:34:40 GMT

{
 "accessKeyId": "3bdecf4afebd41849628389a20629ecc",
 "secretAccessKey": "3f901ef12d454b9c92ba2dde7c029140",
 "sessionToken":
"ZGZiM2M3MmU4Mjk4NGQ2MGEzYTNhYTAyMDE3NTZmZmV8AAAAAC8CAABf0XcbS9E/leusHxRyZ4DYNb+SH374S+mmkxaOpcwdLPC7JL/aTF5F7x83dQn354VKiTyNGuvQIsv7MidIPN7+0oOehdFlHua4RkwqIi8wOs
",
 "createTime": "2015-11-18T11:48:17Z",
 "expiration": "2015-11-18T23:48:17Z",
 "userId": "2d6f4473c99e4ca7be1ca19ec18beacf"
}
```

- sts配置modify权限使用示例（MODIFY权限定义见下方bucket权限控制）
    - `Deny MODIFY + Allow WRITE`：这种配置下允许新增、删除操作，禁止覆盖操作

```
POST /v1/sessionToken?durationSeconds=DurationSeconds HTTP/1.1
Host: sts.bj.baidubce.com
Date: Wed, 06 Apr 2016 06:34:40 GMT
Authorization: AuthorizationString
Content-type:application/json
Content-Length:178

{
 "id":"10eb6f5ff6ff4605bf044313e8f3ffa5",
 "accessControlList": [
  {
   "eid":<eid>,
   "service":"bce:bos",
   "region":"bj",
   "effect": "Allow",
   "resource": ["sts-bucket-1-140683201300192-144784797246238452O478/*"],
   "permission": ["WRITE"]
  },
  {
   "eid":<eid>,
   "service":"bce:bos",
   "region":"bj",
   "effect": "Deny",
   "resource": ["sts-bucket-1-140683201300192-144784797246238452O478/*"],
   "permission": ["MODIFY"]
  }
 ]
}
```

- `Allow MODIFY`：这种配置下只允许覆盖，不允许新增或删除

```
POST /v1/sessionToken?durationSeconds=DurationSeconds HTTP/1.1
Host: sts.bj.baidubce.com
Date: Wed, 06 Apr 2016 06:34:40 GMT
Authorization: AuthorizationString
Content-type:application/json
Content-Length:178

{
 "id":"10eb6f5ff6ff4605bf044313e8f3ffa5",
 "accessControlList": [
  {
   "eid":<eid>,
   "service":"bce:bos",
   "region":"bj",
   "effect": "Allow",
   "resource": ["sts-bucket-1-140683201300192-144784797246238452O478/*"],
   "permission": ["MODIFY"]
  }
 ]
}
```

STS鉴权过程

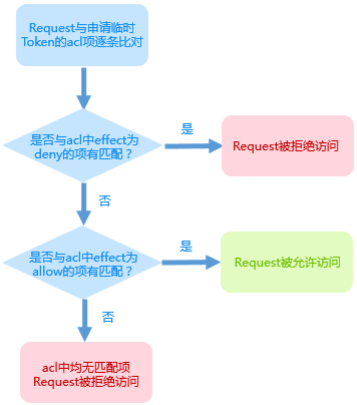
当第三方用户使用返回的临时AK/SK以及SessionToken向百度智能云的服务端发起请求时，服务端是如何完成临时鉴权验证的呢？

- **首先，服务端会验证请求的合法性。**

服务器端会通过请求中的 Access Key ID 找到对应的 Secret Access Key，并提取请求中的签名字符串和signature。如果服务器端计算出来的 signature 和请求中提供的 signature 一样，即认为该请求是合法的；否则认为该请求是非法的，将拒绝处理这次请求。

- **然后，服务端会验证请求者是否对于资源有操作的权限。**

  服务器端会将该次请求与申请临时Token时GetSessionToken携带的acl列表逐条对比，对比后的处理逻辑如下图：



- **举例说明：**

假设在申请临时token时的acl如下：

```
[{
  "effect": "Allow",
  "resource": ["sts-bucket-1"],
  "region": "bj",
  "service": "bce:bos",
  "permission": ["READ"]
}],
```

被授权的第三方想要对sts-bucket-1下的img.jpg进行GetObject操作。 由于GetOject对应READ权限，所以这次请求可以写成：

```
[{
  "resource": ["sts-bucket-1/img.jpg"],
  "region": "bj",
  "service": "bce:bos",
  "permission": ["READ"]
}],
```

和上面的acl进行匹配时会失败，因为resource并不匹配。如果当时申请临时token时resource写成`sts-bucket-1/img.jpg`或`sts-bucket-1/*`则可以成功匹配。

使用STS凭证访问BOS服务

App客户端拿到STSI临时凭证后，通过其中Session Token以及临时访问密钥（AK/SK）构建签名。签名方式同 签名认证访问。需要注意如下关键点：用户使用的签名密钥为STS提供的临时访问密钥（AK/SK）。

与普通的API接口相比，使用STS凭证调用API时，只需要在请求中增加`x-bce-security-token: <session-token>`即可，**以PutObject为例：**

```
 PUT /object HTTP/1.1
Host: <BucketName>.bos.bj.baidubce.com
Date: Wed, 06 Apr 2016 06:34:40 GMT
Authorization: <AuthorizationString>
x-bce-security-token: <SessionToken>
Content-Type: text/plain
Content-Length: 11434
```

> **说明：**
>
> - 目前STS已支持READ、WRITE、LIST、MODIFY、FULL_CONTROL和GetObject等细粒度的API请求。
> - STS具有缓存机制，在有效时间内可使用同一访问凭证多次访问BOS资源，不需要重复发送STS请求。通过修改请求中的durationSeconds参数可调整临时访问凭证的有效时间。

Bucket权限控制

权限控制概述

BOS支持使用ACL对Bucket权限进行管理。Bucket ACL是附属于资源即某个Bucket的权限，其本质上是授权谁（grantee）可以执行哪些操作（permission）。为了方便用户更精细地控制Bucket里的资源，Bucket ACL支持resource和notResource字段。Resource字段用于实现对指定范围的prefix和object粒度的权限控制，notResource字段用户实现对指定范围外的prefix和object粒度的权限控制。 此外，Bucket ACL还支持condition字段，Condition字段可以用来设置访问者的IP、Referer等信息。

为了保障您存储在BOS中数据的高安全性，我们为您提供了丰富的多级权限管理能力。 BOS的权限体系分为如下三级：

- Bucket标准权限（即CannedACL）
- 粗粒度自定义权限
- 细粒度自定义权限

BOS 目前可以通过 **上传ACL文件** 和 **使用CannedAcl** 两种方式来设置 ACL，两种方式都通过 PutBucketAcl接口 实现。上传ACL文件是通过一个JSON文件来描述谁（grantee）在什么条件下（condition）可以对什么资源（resource 或 notResource）执行哪些操作（permission）。直接编辑ACL文件门槛较高，因此 BOS 还支持 CannedAcl 方式。CannedAcl 本质上就是对几种常见的权限控制场景进行了封装，直接在 PutBucketAcl 的头域中的 `x-bce-acl` 字段对资源进行设置。

使用CannedAcl方式的权限控制

CannedAcl是一种方便用户使用的方式，对常见的几种权限情况进行了封装。通过在PutBucketAcl的头域中的 `x-bce-acl` 字段对该资源进行设置的。例：`x-bce-acl:public-read`。字段区分大小写。

当前支持的CannedAcl包括：

ACL	添加的权限
private（私有）	Bucket Owner获得FULL_CONTROL，其他人没有任何权限
public-read（公共读）	Bucket Owner获得FULL_CONTROL，其他人获得READ权限
public-read-write（公共读写）	Bucket Owner获得FULL_CONTROL，其他人获得READ和WRITE权限。注意：该权限安全风险极高。

> **说明：**通过PutBucket创建的新Bucket权限默认是private。

上传ACL文件方式的权限控制

ACL文件格式

PutBucketAcl 可以通过上传一个ACL文件的方式对访问权限进行设置。BOS ACL 使用Json格式的策略描述语言，命名方法为首字母小写的驼峰命名格式，字段区分大小写。

BOS ACL内容可以使用 BOS ACL编辑工具 获取权限配置模板及自定义配置。

- **字段总览：**

字段	数据类型	说明	是否必须	父节点
accessControlList	array	标识acl主体的开始，由一或多组acl配置项组成，其中acl配置项由grantee+permission+resource+condition组合而成。	是	无
+effect	string	指定与该条acl配置项匹配的Request能否执行，取值为Allow或Deny。Allow表示可以执行；Deny表示拒绝执行。	否	accessControlList
+grantee	array	标识被授权人。	是	accessControlList
++id	string	标识被授权人的Account ID，用户的Account ID可以登录控制台点击账户名下的"用户信息->基本信息"查看。	是	grantee
+permission	array	ACL配置项所影响的权限，可选值为 `READ` 、 `LIST` 、 `WRITE` 、和 `GetObject` 。权限详细解释请参见"Bucket ACL支持的permission权限"。	是	accessControlList
+resource	array	ACL配置项所影响的资源，表示对resource指定范围的资源设置访问权限，支持通配符。如：<BucketName>/<ObjectKey>或<BucketName>/xxx*。 resource不填或填Bucket名称，等同于resource字段设为[<bucketName>, <bucketName>/*]，即对Bucket和所有Object设置访问权限。	否	accessControlList
+notResource	array	ACL配置项所影响的资源，表示对notResource指定范围以外的资源设置访问权限，支持通配符。如：<BucketName>/<ObjectKey>或<BucketName>/xxx*，表示对BucketName中ObjectKey之外的Object或者以XXX为前缀的Object之外的其他Object设置访问权限。如果notResource字段不填则等同于没有配置notResource，即采用默认配置，对Bucket及所有Object设置访问权限。	否	accessControlList
+condition	array	ACL配置项所包含的限制条件，支持配置IP地址和Referer名单	否	accessControlList
++ipAddress	array	标识授予访问权限的ip	否	condition
++notIpAddress	array	标识授予访问权限的非ip，表示除这些ip外	否	condition
++referer	object	标识授予访问权限的referer	否	condition
+++stringLike	string[]	标识referer白名单中模糊匹配的地址	否	referer
+++stringEquals	string[]	标识referer白名单中精确匹配的地址	否	referer
++secureTransport	bool	标识是否只允许https方式访问。可选值"true"、"false",不设置时视为"false"。当设置为"true"时，表示只允许https方式访问。	否	condition
++currentTime	object	condition配置项所包含的时间限制条件，支持配置"dateLessThan"，"dateLessThanEquals"，"dateGreaterThan"和"dateGreaterThanEquals"，四个配置项可以任选若干进行设置，匹配有效的条件是所设置配置项均需匹配。	否	condition
+++dateLessThan	string	标识授予访问权限的时间小于指定时间。取值为时间字符串，格式符合ISO 8601规范,如"2018-07-01T12:00:00Z"	否	currentTime
+++dateLessThanEquals	string	标识授予访问权限的时间小于或者等于指定时间。取值为时间字符串，格式符合ISO 8601规范,如"2018-07-01T12:00:00Z"	否	currentTime
+++dateGreaterThan	string	标识授予访问权限的时间大于指定时间。取值为时间字符串，格式符合ISO 8601规范,如"2018-07-01T12:00:00Z"	否	currentTime
+++dateGreaterThanEquals	string	标识授予访问权限的时间大于或者等于指定时间。取值为时间字符串，格式符合ISO 8601规范,如"2018-07-01T12:00:00Z"	否	currentTime
++userAgent	object	标识授予访问权限的UA（user agent）	否	condition
+++stringLike	string[]	标识userAgent中模糊匹配的字符串	否	userAgent
+++stringEquals	string[]	标识userAgent中精确匹配的字符串	否	userAgent

说明：

- 所使用的ACL格式如上所述；在上传ACL文件时，可不带Owner属性(json字段标识Bucket的Owner)；如果带有Owner属性，则其id值必须为Bucket的Owner id
- 上传的ACL文件不大于20KB。
- 若用户使用PutBucketAcl接口的时候，在Http报文的Header(Canned ACL)和Body中同时设置了ACL，则返回错误码400，错误说明为"参数不正确"。
- 设置currentTime子字段值时，设置的是GMT时间，注意与本地时间的差别。
- 在一条ACL规则中，同时只能存在一个resource或一个notResource的设置。

Bucket ACL支持粗粒度自定义权限

permission本质上对应一组BOS API操作。BOS API分为bucket级别API和object级别的API，如ListObjects用来查看一个bucket中的所有Object列表，是一个bucket级别API，PutObject用来上传一个文件，是一个object级别API。 所以在ACL文件里，当设置好permission后，需要设定相应的resource或notResource。缺省情况下resource字段不填或填bucket名称，就可以同时匹配bucket级别和object级别的所有操作。

- **Bucket ACL支持如下粗粒度自定义权限：**

权限名称	权限支持的操作
READ	允许读取Bucket内的Object及其相关信息，但没有列表权限，具体操作权限包含GetBucketLocation、 HeadBucket、 GetObject、GetObjectMeta、 ListParts、 RestoreObject。READ权限对应的API既有bucket级别的API如GetBucketLocation，也有object级别的API如GetObject和ListParts。
LIST	列表权限，可以查看指定Bucket下的Object列表以及获取所有未执行完的MultipartUpload，具体操作权限包含ListObjects和ListMultipartUploads。LIST权限对应的API只有bucket级别的API。
WRITE	允许创建，覆盖和删除Bucket内的Object，具体操作权限包含PutObject、 PostObject、 InitiateMultipartUpload、 UploadPart、CompleteMultipartUpload、 AbortMultipartUpload、 AppendObject、 DeleteObject、DeleteMultipleObjects和FetchObject。WRITE权限对应的API只有object级别。
MODIFY	用户仅可做PutObject、AppendObject等相关写入操作，不可做数据新增、删除操作。该权限主要功能是与Deny合用防止Bucket数据被篡改
FULL_CONTROL	包含以上所有权限。FULL_CONTROL除了具有READ、LIST和WRITE的所有操作权限以外，还包括以下操作权限PutBucketAcl、GetBucketACL、 PutBucketCors、 GetBucketCors等所有细粒度API，具体参考下面细粒度权限表。FULL_CONTROL权限对应的API既有bucket级别的API也有Object级别的API。

Bucket ACL支持细粒度自定义权限

为了保障您存储在BOS中数据的高安全性和满足您对BOS资源更加细粒度的进行权限访问控制，BOS在支持READ、LIST、WRITE、MODIFY、FULL_CONTROL这几种粗细粒度的基础上，现在扩展支持了各类细粒度权限。

- **Bucket级别相关权限说明如下：**

Bucket 细粒度权限	支持的操作	所属粗粒度权限
GetBucket	该权限表示允许用户获取Bucket内容及其相关信息，例如，列出Bucket下面的Objects、在三步上传时列出Bucket下面的所有未执行完成的Multipart Upload	LIST、FULL_CONTROL
GetBucketAcl	该权限表示允许用户获取Bucket Acl的信息	FULL_CONTROL
PutBucketAcl	该权限表示允许用户新增Bucket Acl	FULL_CONTROL
GetBucketCors	该权限表示允许用户获取Bucket上的跨域资源共享（CORS）的规则	FULL_CONTROL
PutBucketCors	该权限表示允许用户在指定的Bucket上设定或者删除一个跨域资源共享（CORS）的规则	FULL_CONTROL
GetBucketStyle	该权限表示允许用户获取或者列出Bucket Style的规则	FULL_CONTROL
PutBucketStyle	该权限表示允许用户新增或者删除Bucket Style的规则	FULL_CONTROL
GetBucketMirroring	该权限表示允许用户获取Bucket镜像回源的相关信息	FULL_CONTROL
PutBucketMirroring	该权限表示允许用户新增或者删除Bucket镜像回源的相关信息	FULL_CONTROL
GetCopyRightProtection	该权限表示允许用户获取Bucket的原图保护配置的信息	FULL_CONTROL
PutCopyRightProtection	该权限表示允许用户开启或者关闭Bucket的原图保护功能	FULL_CONTROL
PutBucketLifecycle	该权限表示允许用户开启或者关闭Bucket的生命周期功能	FULL_CONTROL
GetBucketLifecycle	该权限表示允许用户获取Bucket的生命周期功能	FULL_CONTROL
PutBucketReplication	该权限表示允许用户开启或者关闭Bucket的数据同步功能	FULL_CONTROL
GetBucketReplication	该权限表示允许用户获取Bucket的数据同步功能	FULL_CONTROL
PutBucketEncryption	该权限表示允许用户开启或者关闭Bucket的数据加密功能	FULL_CONTROL
GetBucketEncryption	该权限表示允许用户获取Bucket的数据加密功能	FULL_CONTROL
PutBucketStaticWebsite	该权限表示允许用户开启或者关闭Bucket的静态网站功能	FULL_CONTROL
GetBucketStaticWebsite	该权限表示允许用户获取Bucket的静态网站功能	FULL_CONTROL
PutBucketLogging	该权限表示允许用户开启或者关闭Bucket的日志管理功能	FULL_CONTROL
GetBucketLogging	该权限表示允许用户获取Bucket的日志管理功能	FULL_CONTROL
PutBucketRequestPayment	该权限表示允许用户开启或者关闭Bucket的请求者付费功能	FULL_CONTROL
GetBucketRequestPayment	该权限表示允许用户获取Bucket的请求者付费功能	FULL_CONTROL
PutBucketTagging	该权限表示允许用户开启或者关闭Bucket的标签管理功能	FULL_CONTROL
GetBucketTagging	该权限表示允许用户获取Bucket的标签管理功能	FULL_CONTROL
PutNotification	该权限表示允许用户开启或者关闭Bucket的事件通知功能	FULL_CONTROL
GetNotification	该权限表示允许用户获取Bucket的事件通知功能	FULL_CONTROL
PutBucketObjectLock	该权限表示允许用户开启或者关闭Bucket的合规保留功能	FULL_CONTROL
GetBucketObjectLock	该权限表示允许用户获取Bucket的合规保留功能	FULL_CONTROL
PutBucketInventory	该权限表示允许用户开启或者关闭Bucket的清单设置功能	FULL_CONTROL
GetBucketInventory	该权限表示允许用户获取Bucket的清单设置功能	FULL_CONTROL
PutBucketStorageAnalysis	该权限表示允许用户开启或者关闭Bucket的存储桶分析功能	FULL_CONTROL
GetBucketStorageAnalysis	该权限表示允许用户获取Bucket的存储桶分析功能	FULL_CONTROL
PutBucketStorageClass	该权限表示允许用户开启或者关闭Bucket的存储类型管理功能	FULL_CONTROL
GetBucketStorageClass	该权限表示允许用户获取Bucket的存储类型管理功能	FULL_CONTROL
PutBucketTrash	该权限表示允许用户开启或者关闭Bucket的回收站功能	FULL_CONTROL
GetBucketTrash	该权限表示允许用户获取Bucket的回收站功能	FULL_CONTROL
PutBucketQuota	该权限表示允许用户开启或者关闭Bucket的额度管理功能	FULL_CONTROL
GetBucketQuota	该权限表示允许用户获取Bucket的额度管理功能	FULL_CONTROL
PutBucketVersioning	该权限表示允许用户开启或者关闭Bucket的多版本管理功能	FULL_CONTROL
GetBucketVersioning	该权限表示允许用户获取Bucket的多版本管理功能	FULL_CONTROL
GetObjectVersion	该权限表示允许用户获取对象多版本信息	FULL_CONTROL
DeleteObjectVersion	该权限表示允许用户删除对象多版本信息	FULL_CONTROL
ListObjectVersions	该权限表示允许用户列举对象多版本信息	FULL_CONTROL
PutObjectVersionAcl	该权限表示允许用户设置对象多版本权限	FULL_CONTROL
GetObjectVersionAcl	该权限表示允许用户获取对象多版本权限	FULL_CONTROL

- **Object级别相关权限说明如下：**

Object 细粒度权限	支持的操作	所属粗粒度权限
PutObject	该权限表示允许用户进行上传Object的相关操作，例如PutObject、PostObject、AppendObject、FetchObject、CopyObject、三步上传、三步Copy	WRITE、FULL_CONTROL
GetObject	仅支持GetObject和GetObjectMeta操作。GetObject权限对应的API只有object级别。	READ、FULL_CONTROL
RestoreObject	该权限表示允许用户取回归档类型文件	READ、FULL_CONTROL
DeleteObject	该权限表示允许用户进行删除单个Object或者批量删除Object的相关操作	WRITE、FULL_CONTROL
RenameObject	该权限表示允许用户对Object进行重命名（Rename）	WRITE、FULL_CONTROL
ListParts	该权限表示允许用户列出三步上传过程中指定的UploadId所有已经上传成功的Part，用户可以查看三步上传的当前进度	READ、FULL_CONTROL
GetObjectAcl	该权限表示允许用户获取Object Acl	FULL_CONTROL
PutObjectAcl	该权限表示允许用户新增Object Acl和删除Object Acl	FULL_CONTROL

> **说明：**

- 新增的粗细粒度和以前的READ、LIST、WRITE、FULL_CONTROL、MODIFY、GetObject、RestoreObject这几种粗细粒度互不影响，其中READ，WRITE，LIST，FULL_CONTROL， MODIFY属于粗粒度权限。
- 粗粒度权限高于细粒度权限，如果既配了粗粒度权限又配了细粒度权限，粗粒度权限会覆盖细粒度权限，以粗粒度为主。
- Bucket级别的细粒度是指对Bucket进行的相关操作。
- Object级别的细粒度是指对Object进行的相关操作。

Bucket ACL支持Deny和数据防篡改

Bucket Acl同时支持effect字段，用于设置该条权限的效果，effect支持Allow和Deny这两种配置方式。

说明：

- 在不配置effect的情况下，默认情况下是隐式Allow。在配置effect的情况下，当值为Allow时是显式Allow；在不配置effect和配置effect并且其值为Allow的况下，这两种情况是等价的。
- Deny的语义高于Allow，当用户既配了Allow也配了Deny的情况下，以Deny为准，拒绝通过；Deny了粗粒度权限，不管细粒度权限是否允许，都不允许通过。
- Allow了粗粒度权限，但是Deny了细粒度权限，那么就取粗粒度与细粒度的差集，差集部分的操作是允许通过的。
- 目前READ、LIST、WRITE、FULL_CONTROL、MODIFY权限为粗粒度权限，其余的权限可以归结为细粒度权限。

为了解决由于密钥泄露、权限控制不严格、操作失误等带来的数据删除、篡改风险，BOS推出Bucket数据防篡改的功能，通过**在Bucket Acl设置 MODIFY 粗粒度权限和 Deny 关键字组合**就可以起到Bucket数据防篡改的效果。例如：如果想要设置禁止覆盖与删除Object，可通过同时设置 Deny MODIFY + Deny DeleteObject 达到效果。

目前涉及到写的粗粒度有 WRITE、FULL_CONTROL、MODIFY，细粒度有 PutObject、RenameObject、DeleteObject。

MODIFY的使用方式主要有两大类，Allow 和 Deny 这两种。

说明：

- 写操作，包括新增操作、覆盖操作和删除操作，新增操作是指：新上传一个Object，这个Object以前是不存在的，覆盖操作是指：上传一个Object，这个Object以前是存在的。
- MODIFY粗粒度写权限包含RenameObject细粒度写权限和PutObject细粒度写权限下面的部分操作，如PutObject、PostObject、AppendObject、CopyObject、FetchObject、MultiUploadInitObject等。
- 如果配置允许了PutObject、RenameObject、DeleteObject、WRITE、FULL_CONTROL几种写粒度之中的任意一种，并且没有配置Deny MODIFY，那么允许用户进行正常的写操作。

Allow示例

MODIFY配置为Allow的示例	配置示例描述	备注
Allow MODIFY	只配置了Allow MODIFY，没有配置其它写粒度权限	这种场景下，只允许覆盖，不允许新增或删除
Allow MODIFY + Allow 细粒度写权限	配置了Allow MODIFY和细粒度写权限	这种场景下，在配置细粒度写权限下的操作允许新增，并且允许任何覆盖操作
Allow MODIFY + Allow 粗粒度写权限	配置了Allow MODIFY和粗粒度写权限	这种场景下，允许任何写操作
Allow MODIFY + Allow 粗粒度写权限 + Allow 细粒度写权限	配置了Allow MODIFY和粗细粒度写权限	这种场景下，允许任何写操作，粗细粒度同时配置时以粗粒度为准
Allow MODIFY + Deny 细粒度写权限	配置了Allow MODIFY和Deny细粒度写权限	这种场景下，不允许新增、删除，在配置细粒度写权限之外的操作允许覆盖
Allow MODIFY + Deny 粗粒度写权限	配置了Allow MODIFY和Deny粗粒度写权限	这种场景下，不允许新增、删除，也不允许覆盖
Allow MODIFY + Deny 细粒度写权限 + Allow 粗粒度写权限	配置了Allow MODIFY、Deny细粒度写权限和Allow粗粒度写权限	这种场景下，在配置细粒度写权限下面对应的操作不可以新增、删除，也不可以覆盖；在配置了细粒度写权限之外的操作允许新增并且允许覆盖

Deny示例

MODIFY配置为Deny的示例	配置示例描述	备注
Deny MODIFY	只配置了Deny MODIFY，没有配置其它写粒度权限	这种场景下，不允许覆盖，允许新增或删除
Deny MODIFY + Deny 细粒度写权限	配置了Deny MODIFY和细粒度写权限	这种场景下，不允许覆盖，在配置细粒度写权限之外的操作允许增加、删除操作。例如：如果想要设置禁止覆盖与删除Object，需要同时设置Deny MODIFY+Deny DeleteObject
Deny MODIFY + Deny 粗粒度写权限	配置了Deny MODIFY和粗粒度写权限	这种场景下，不允许任何写操作
Deny MODIFY + Deny 粗细粒度写权限	配置了Deny MODIFY和粗细粒度写权限	这种场景下，不允许任何写操作，粗细粒度同时配置时以粗粒度为准
Deny MODIFY + Allow 细粒度写权限	配置了Deny MODIFY和Allow细粒度写权限	这种场景下，在配置了细粒度写权限下面的操作可以新增、删除，不允许覆盖操作
Deny MODIFY + Allow 粗粒度写权限	配置了Deny MODIFY和Allow粗粒度写权限	这种场景下，允许任何新增、删除操作，不允许覆盖操作
Deny MODIFY + Deny 细粒度写权限 + Allow 粗粒度写权限	配置了Deny MODIFY、Deny细粒度写权限和Allow粗粒度写权限	这种场景下，在配置了细粒度写权限下的操作不允许新增、删除，也不允许覆盖；其余的细粒度写权限下面的操作允许新增、删除，但是不允许覆盖

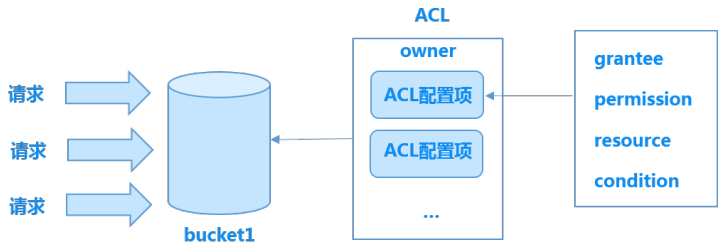
数据安全提示：

在您为用户授权时，建议严格遵循最小权限原则，限定用户执行受限的操作并设置访问指定前缀的资源，避免授予过大的权限，导致预期外的越权操作，造成数据安全风险。包括但不限于以下典型场景：

- 粗粒度自定义权限LIST、WRITE操作的安全风险较高，不建议向所有用户授权该操作，可通过配置grantee字段标识被授权人。
- 授权WRITE操作时，建议避免授权Bucket级别的写权限，建议配置resource字段值精确到BucketName/ObjectName/*。
- 自定义粗粒度权限中，READ权限允许读取Bucket内的Object及其相关信息，但没有列表权限，LIST权限为列表权限，可以查看指定Bucket下的Object列表以及获取所有未执行完的MultipartUpload。建议区分READ权限和LIST权限支持的操作范围，在最小范围内为用户分配权限。
- 多用户共同使用同一个Bucket时，建议通过划分Bucket区域的方式明确用户可访问的资源，授权时精确到BucketName/Userid/*。
- 当时用户同时设置了sts的MODIFY权限和bucket的MODIFY权限时：优先遵循最小权限原则，当两个维度权限无交集时，会直接禁止访问。

请求授权过程

使用ACL对Bucket进行权限管理时，每个Bucket只能有一个ACL文件，但每个ACL里可以有一组或多组ACL配置项用来定义不同用户对不同资源拥有不同操作权限，accessControlList字段用来标识acl主体的开始。ACL文件中的每组ACL配置项由grantee+permission+resource（或notResource）+condition组合而成，如果某个请求能成功授权则需要匹配一组ACL配置项中的所有条件。当一条请求过来时会逐个匹配ACL中的配置项，只要某组ACL配置项中有一个条件没有匹配上则不能通过授权，除非一组ACL配置项中的所有条件匹配上才能完成授权。ACL授权过程如下图：



假定此时ACL文件定义的grantee是*即所有用户，permission是READ，resource是bucket1。

- 如果请求为PutObject即上传文件cat.jpg到bucket1中，该请求会被拒绝，因为PutObject不属于READ permission所包含的API，所以请求不能通过。
- 如果请求为GetObject即从bucket1中下载文件cat.jpg，该请求可以通过授权，因为GetObject对应的permission是READ，匹配ACL中的所有条件，所以通过授权。

> **说明:** CopyObject操作，需要对源Object有READ、GetObject或FULL_CONTROL权限，对目标Object有WRITE或FULL_CONTROL权限。

示例

- 该示例主要讲解grantee和permission的基础用法。

  假设bucket1的owner希望让另一个百度智能云用户（userId=16147f559dd14bb294175a8bab74ff1f）帮助自己管理bucket1，即设置该用户对bucket1拥有FULL_CONTROL权限，对应的acl文件格式如下：

```
{
  "accessControlList": [
    {
      "grantee": [
        {
          "id": "16147f559dd14bb294175a8bab74ff1f"
        }
      ],
      "permission": [
        "FULL_CONTROL"
      ]
    }
  ]
}
```

- 该示例主要讲解多个ACL配置项的使用方法。

  bucket1的owner希望所有人都可以读取bucket的内容，但只有一个百度智能云用户（userid=b124deeaf6f641c9ac27700b41a350a8）能够管理bucket，则设置所有用户为READ权限，而userid=b124deeaf6f641c9ac27700b41a350a8的用户为FULL_CONTROL权限。对应的ACL文件格式如下：

```
{
  "accessControlList": [
    {
      "grantee": [
        {
          "id": "b124deeaf6f641c9ac27700b41a350a8"
        }
      ],
      "permission": [
        "FULL_CONTROL"
      ]
    },
    {
      "grantee": [
        {
          "id": "*"
        }
      ],
      "permission": [
        "READ"
      ]
    }
  ]
}
```

- 该示例主要讲解condition字段的使用方法。

  bucket1的owner允许通过特定IP段的userid=10eb6f5ff6ff4605bf044313e8f3ffa5的用户来管理Bucket，则通过condition定义IP地址段，并授权这些用户FULL_CONTROL权限。对应的ACL文件格式如下：

```
{
 "accessControlList": [
   {
     "grantee": [
       {
         "id":"10eb6f5ff6ff4605bf044313e8f3ffa5"
       }
     ],
     "permission": [
       "FULL_CONTROL"
     ],
     "condition" : {
       "ipAddress": [
         "192.168.0.0/16",
         "192.169.0.*",
         "192.170.0.5"
       ]
     }
   }
 ]
}
```

bucket1的owner允许通过不在特定IP段的userid=10eb6f5ff6ff4605bf044313e8f3ffa5的用户来管理Bucket，则通过condition定义非IP地址段，并授权这些用户FULL_CONTROL权限。对应的ACL文件格式如下：

```
{
 "accessControlList": [
   {
     "grantee": [
       {
         "id":"10eb6f5ff6ff4605bf044313e8f3ffa5"
       }
     ],
     "permission": [
       "FULL_CONTROL"
     ],
     "condition" : {
       "notIpAddress": [
         "192.168.0.0/16",
         "192.169.0.*",
         "192.170.0.5"
       ]
     }
   }
 ]
}
```

- 该示例主要讲解condition的secureTransport和currentTime字段的使用方法。

  bucket1的owner只允许通过https方式在指定时间访问的userid=10eb6f5ff6ff4605bf044313e8f3ffa5的用户来管理Bucket，则通过condition定义secureTransport、currentTime地址段，并授权这些用户FULL_CONTROL权限。对应的ACL文件格式如下：

```
{
 "accessControlList": [
   {
     "grantee": [
       {
         "id": "10eb6f5ff6ff4605bf044313e8f3ffa5"
       }
     ],
     "permission": [
       "FULL_CONTROL"
     ],
     "resource": [
       "bucket1/*"
     ],
     "condition": {
       "currentTime": {
         "dateLessThan": "2020-07-01T12:00:00Z",
         "dateGreaterThan": "2018-03-01T15:00:00Z"
       },
       "secureTransport": true
     }
   }
 ]
}
```

- 该示例主要讲解referer字段的使用方法。

  bucket1的owner允许通过特定IP且referer与配置的白名单匹配的userid=c558855ea8514c299508699b115473ef的用户查看Bucket和Object信息。其中referer字段用来定义允许访问的白名单，stringEquals用于精确匹配，stringLike用于模糊匹配。stringLike中`*`代表0到任意多的字符，最多可以有一个`*`，`*`可以在字符串的任意位置。对应的ACL文件格式如下：

```
{
 "accessControlList": [
    {
       "condition": {
          "referer": {
             "stringLike": [
                "http://www.abc.com/*"
             ],
             "stringEquals": [
                "http://www.abc.com"
             ]
          },
          "ipAddress": [
             "192.168.1.1"
          ]
       },
       "grantee": [
          {
             "id": "c558855ea8514c299508699b115473ef"
          }
       ],
       "permission": [
          "LIST"
       ],
       "resource": [
          "bucket1",
          "bucket1/*"
       ]
    }
 ]
}
```

- 该示例主要讲解userAgent字段的使用方法。

  bucket1的owner允许请求header中携带特定UA的用户c558855ea8514c299508699b115473ef查看Bucket和Object信息。其中userAgent字段用来定义允许访问的白名单，stringEquals用于精确匹配，stringLike用于模糊匹配。stringLike中`*`代表0到任意多的字符，最多可以有一个`*`，`*`可以在字符串的任意位置。对应的ACL文件格式如下：

```
{
 "accessControlList": [
    {
       "condition": {
          "userAgent": {
             "stringLike": [
                "*.test-user-agent"
             ],
          },
       },
       "grantee": [
          {
             "id": "c558855ea8514c299508699b115473ef"
          }
       ],
       "permission": [
          "LIST"
       ],
       "resource": [
          "bucket1",
          "bucket1/*"
       ]
    }
 ]
}
```

- 该示例主要讲解resource字段的使用方法。

  resource字段可以用来对Bucket内的文件和目录（prefix）设置访问权限。 bucket1的owner只允许另一个百度智能云用户（userid=10eb6f5ff6ff4605bf044313e8f3ffa5）对“cook” 为前缀的Object、“edu/” 为前缀的Object和“travel/中国国家地理杂志”这个Object有FULL_CONTROL权限。resource字段的末尾有且只能有一个*。由于resource字段所指定资源是Object，所以只有FULL_CONTROL permission所包含的Object级别的API操作（如PutObject、GetObject、DeleteObject）会被执行。对应的ACL文件格式如下：

```
{
 "accessControlList": [
    {
       "grantee": [
          {
             "id":"10eb6f5ff6ff4605bf044313e8f3ffa5"
          }
       ],
       "permission": [
          "FULL_CONTROL"
       ],
       "resource": [
          "bucket1/cook*",
          "bucket1/edu/*",
          "bucket1/travel/中国国家地理杂志"
       ]
    }
 ]
}
```

- 该示例主要讲解notResource字段的使用方法。

  notResource字段可以用来对Bucket内某些指定文件和目录（prefix）之外的Object设置访问权限。bucket1的owner只允许另一个百度智能云用户（userid=10eb6f5ff6ff4605bf044313e8f3ffa5）对除“cook”为前缀的Object、“edu/”为前缀的Object和“travel/中国国家地理杂志”这个Object之外的其它Object有FULL_CONTROL权限。notResource字段的末尾有且只能有一个*。由于notResource所指定资源是Object，所以只有FULL_CONTROL permission所包含的Object级别的API操作（如PutObject、GetObject、DeleteObject）会被执行。对应的ACL文件格式如下：

```
    {
     "accessControlList": [
       {
          "grantee": [
            {
               "id":"10eb6f5ff6ff4605bf044313e8f3ffa5"
            }
          ],
          "permission": [
            "FULL_CONTROL"
          ],
          "notResource": [
            "bucket1/cook*",
            "bucket1/edu/*",
            "bucket1/travel/中国国家地理杂志"
          ]
       }
     ]
    }
```

- 该示例主要讲解Bucket级别的权限示例的基础用法。 bucket1的owner希望只有一个百度智能云用户（userid=b124deeaf6f641c9ac27700b41a350a8）能够查看bucket的信息或者列出Bucket下面的Object，则设置这个用户为GetBucket权限。对应的ACL文件格式如下：

```
{
  "accessControlList": [
    {
      "grantee": [
        {
          "id": "b124deeaf6f641c9ac27700b41a350a8"
        }
      ],
      "permission": [
        "GetBucket"
      ]
    }
  ]
}
```

- 该示例主要讲解Object级别的权限示例的基础用法。

  bucket1的owner希望所有人都只可以对Bucket里面的Object进行读和写，但只有一个百度智能云用户（userid=b124deeaf6f641c9ac27700b41a350a8）能够管理bucket，则设置所有用户为GetObject，PutObject权限，而userid=b124deeaf6f641c9ac27700b41a350a8的用户为FULL_CONTROL权限。对应的ACL文件格式如下：

```
  {
    "accessControlList": [
      {
        "grantee": [
          {
            "id": "b124deeaf6f641c9ac27700b41a350a8"
          }
        ],
        "permission": [
          "FULL_CONTROL"
        ]
      },
      {
        "grantee": [
          {
            "id": "*"
          }
        ],
        "permission": [
          "GetObject" , "PutObject"
        ]
      }
    ]
  }
```

- 该示例主要讲解Bucket防数据篡改示例的基础用法。

  bucket1的owner不希望一个百度智能云用户（userid=b124deeaf6f641c9ac27700b41a350a8）篡改Bucket的数据，但是允许其新增数据和读取数据，则设置userid=b124deeaf6f641c9ac27700b41a350a8的用户权限为Deny MODIFY、Allow PutObject、Allow READ，对应的ACL文件格式如下：

```
{
"accessControlList": [
{
 "effect" : "Deny",
 "grantee": [
  {
            "id": "b124deeaf6f641c9ac27700b41a350a8"
          }
        ],
        "permission": [
          "MODIFY"
        ]
    },{
  "effect" : "Allow",
  "grantee": [
   {
            "id": "b124deeaf6f641c9ac27700b41a350a8"
          }
        ],
        "permission": [
          "PutObject","READ"
        ]
    }
  ]
}
```

查看访问权限

- 查看某Bucekt的访问权限，详见GetBucketAcl接口。
- 返回的ACL会自动加上Owner属性。

Object权限控制

BOS支持使用ACL对Object权限进行管理。Object ACL是附属于资源即某个Object的权限，其本质上是授权谁（grantee）可以执行哪些操作（permission）。

BOS目前可以通过上传ACL文件和使用CannedAcl两种方式来设置ACL，两种方式都可以通过PutObjectAcl接口实现。

- 上传ACL文件是通过一个JSON文件来描述Object ACL信息。直接编辑ACL文件门槛较高，因此BOS还支持CannedAcl方式。
- CannedAcl本质上就是对几种常见的权限控制场景进行了封装，直接在PutObjectAcl的头域中的"x-bce-acl"字段或者x-bce-grant-read/x-bce-grant-full-control字段对资源进行设置。

Object ACL支持的permission权限

Object Acl支持的权限值	支持的操作
READ	该属性表示允许用户读取Object内容及其相关信息
FULL_CONTROL	该属性表示用户拥有Object的控制权限，等价于READ和put/get/delete object acl的权限

CannedACL支持类型

Acl	添加的权限
private（私有）	Bucket Owner获得FULL_CONTROL，其他人没有任何权限
public-read（公共读）	Bucket Owner获得FULL_CONTROL，其他人获得READ权限

CannedACL支持的三种header

为了方便权限设置，在创建Object、复制Object及设置Object ACL时可以添加CannedACL，通过头域的"x-bce-acl"或者"x-bce-grant-permission"来设置Object访问权限，目前不支持在同一请求中同时设置CannedACL和上传ACL文件。主要有以下header设置方式, 两种类型的header不可以同时在一个请求中出现。

CannedACL Header	有效值	是否必须
x-bce-acl	private/public-read	否
x-bce-grant-read	支持多个id，以逗号分隔	否
x-bce-grant-full-control	支持多个id，以逗号分隔	否

> **说明：**
>
> - 如果object是归档类型的文件，那么该文件在归档文件刚上传和取回过程完成前，不能设置和删除object acl配置。

可通过Header修改Canned ACL的接口

- PutObject接口
- PostObject接口
- AppendObject接口
- CopyObject接口
- InitMultiUpload接口
- PutObjectAcl接口

Service相关接口

额度管理

PutUserQuota

接口描述

此接口用来设置用户的额度，额度包括总的Bucket数及各个Region总的存储容量，当前总的Bucket数最大值不能超过100，注意该接口只能被**主用户**调用。

请求

- 请求语法

```
PUT /?userQuota HTTP/1.1
Host: bj.bcebos.com
Date: <Date>
Authorization: <Authorization_String>
Content-Type: text/plain
Content-Length: <Content_Length>
{
  "maxBucketCount" : 50,
  "maxCapacityMegaBytes"  : 12334424
}
```

- 请求参数

名称	类型	描述	是否必须
maxBucketCount	int64	最大可以创建的Bucket数，若为-1，则表示不设置	是
maxCapacityMegaBytes	int64	单位为MB，表示最大的存储容量，若为-1或者0，表示不设置存储容量额度限制，即无上限	是

- 请求头域

  无特殊请求头域

响应

- 响应头域

  无特殊参数

- 响应元素

  无特殊参数

示例

- 请求示例

```
PUT /?userQuota HTTP/1.1
Host: bj.bcebos.com
Date: <Date>
Authorization: <Authorization_String>
Content-Type: text/plain
Content-Length: <Content_Length>
{
  "maxBucketCount" : 50,
  "maxCapacityMegaBytes"  : 12334424
}
```

- 响应示例

```
HTTP/1.1 200 OK
x-bce-request-id: 4db4b34d-653d-4d9a-b49b-3049ca786409
Date: Wed, 31 May 2019 10:34:40 GMT
Server: BceBos
```

GetUserQuota

接口描述

此接口用来查看用户的额度设置，注意该接口只能被主用户调用。

请求

- 请求语法

```
GET /?userQuota HTTP/1.1
Host: bj.bcebos.com
Date: <Date>
Authorization: <Authorization_String>
```

- 请求参数

  无特殊参数

- 请求头域

  无特殊参数

响应

- 响应头域

  无特殊参数

- 响应参数

参数	描述
maxBucketCount	额度设置中的最大可创建Bucket数，若为-1，表示无限制
maxCapacityMegaBytes	额度设置中的最大存储容量限制，若为-1或0，表示无限制

示例

- 请求示例

```
 GET /?userQuota HTTP/1.1
Host: bj.bcebos.com
Date: <Date>
Authorization: <Authorization_String>
```

- 响应示例

```
 HTTP/1.1 200 OK
x-bce-request-id: 4db2b34d-654d-4d8a-b49b-3049ca786409
Date: Wed, 06 Apr 2016 06:34:40 GMT
Server: BceBos
{
    "maxBucketCount": 50,
    "maxCapacityMegaBytes" : 12334424
}
```

DeleteUserQuota

接口描述 此接口用来删除额度设置，注意该接口只能被主用户调用。

请求

- 请求语法

```
 DELETE /?userQuota HTTP/1.1
Host: bj.bcebos.com
Date: <Date>
Authorization: <Authorization_String>
```

- 请求参数

  无特殊参数

- 请求头域

  无特殊参数

响应

- 响应头域

  无特殊参数

- 响应参数

  无特殊参数

示例

- 请求示例

```
 DELETE /?userQuota HTTP/1.1
Host: bj.bcebos.com
Date: <Date>
Authorization: <Authorization_String>
```

- 响应示例

```
 HTTP/1.1 200 OK
x-bce-request-id: 4db2b34d-654d-4d8a-b49b-3049ca786409
Date: Wed, 06 Apr 2016 06:34:40 GMT
Server: BceBos
```

ListBuckets

接口描述

本接口列举了请求者拥有的所有bucket。

请求（Request）

- 请求语法

```
  GET / HTTP/1.1
Host: bj.bcebos.com
Date: <Date>
Authorization: <AuthorizationString>
```

- 请求头域

  无特殊参数

- 请求参数

  无特殊参数

响应（Response）

- 响应头域

  无特殊Header参数响应

- 响应元素

名称	类型	描述
owner	Object	Bucket owner（拥有者）信息
+id	String	Bucket owner的用户id
+displayName	String	Bucket owner的名称
buckets	Array	存放多个bucket信息的容器
+bucket	Object	存放一个bucket信息的容器
+name	String	Bucket名称
+location	String	Bucket所在区域
+creationDate	Date	Bucket创建时间
+enableMultiAz	Boolean	Bucket数据是否多AZ分布，非多AZ bucket不返回该属性

> **注意:** 如果请求中没有用户验证信息（即匿名访问），返回`403 Forbidden`，错误信息：`AccessDenied`。

示例

- JSON请求示例

  说明：一次请求最多返回100个bucket的信息。

```
GET / HTTP/1.1
Host: bj.bcebos.com
Date: Wed, 06 Apr 2016 06:34:40 GMT
Authorization: AuthorizationString
```

- JSON请求响应示例

```
{
  "owner":{
    "id":"10eb6f5ff6ff4605bf044313e8f3ffa5",
    "displayName":"BosUser"
  },
  "buckets":[
    {
      "name":"bucket1",
      "location":"bj",
      "creationDate":"2016-04-05T10:20:35Z",
      "enableMultiAz":true
    },
    {
      "name":"bucket2",
      "location":"bj",
      "creationDate":"2016-04-05T16:41:58Z"
    }
  ]
}
```

> **说明：** JSON请求响应项的命名规则是首字母小写的驼峰格式。

Bucket相关接口

基础操作

GetBucketLocation

接口描述

本接口用于获取Bucket所在的区域。

请求（Request）

- 请求语法

```
GET /?location HTTP/1.1
Host: <BucketName>.bj.bcebos.com
Date: <Date>
Authorization: <AuthorizationString>
Content-Length: 0
```

- 请求头域

  无特殊参数

- 请求参数

  无特殊参数

响应（Response）

- 响应头域

  无特殊Header参数响应

- 响应元素

名称	类型	描述
location	String	Bucket所在区域

示例

- 请求示例

```
GET /?location HTTP/1.1
Host: BucketName.bj.bcebos.com
Date: Wed, 06 Apr 2016 06:34:40 GMT
Authorization: AuthorizationString
Content-Length: 0
```

- 响应示例

```
HTTP/1.1 200 OK
x-bce-request-id: 4db2b34d-654d-4d8a-b49b-3049ca786409
Date: Wed, 06 Apr 2016 06:34:40 GMT
Content-Length: 35
Connection: close
Server: BceBos

{
    "locationConstraint": "bj"
}
```

PutBucket

接口描述

本接口用于创建Bucket。每一个用户只允许创建100个Bucket。创建的Bucket其权限默认为private，即Bucket Owner获得FULL_CONTROL，其他人没有任何权限。

请求（Request）

- 请求语法

```
PUT / HTTP/1.1
Host: <BucketName>.bj.bcebos.com
Date: <Date>
Authorization: <AuthorizationString>
Content-Length: <ContentLength>
Content-Type: text/plain
```

- 请求头域

名称	类型	描述	是否必须
x-bce-tag-list	string	创建bucket时为bucket绑定tag，格式为tag1=value1&tag2=value2	否

- 请求参数

名称	类型	描述	是否必须
enableMultiAZ	Boolean	指定创建的Bucket中数据是否多AZ分布，达到机房级别的数据容灾，true代表创建多AZ bucket，默认为false代表创建。	否

响应（Response）

- 响应头域

无特殊Header参数返回

- 响应元素

无

> **注意事项**
>
> - 若一个用户创建的Bucket超过100个，服务将返回400 Bad Request，错误码TooManyBuckets。
> - 若请求的Bucket已存在，无论该Bucket是否是请求者创建，都会返回409 Conflict，错误信息：BucketAlreadyExists。
> - 创建数据多AZ分布的bucket的前提是该region已经支持创建多AZ的bucket，目前BOS仅支持北京、广州和苏州三个region，其它region陆续开通中。
> - BOS存储桶名称存在限制，只能包含小写字母、数字和"-"，开头结尾为小写字母和数字，长度在4-63之间。

示例

- 请求示例

```
PUT / HTTP/1.1
Host: <BucketName>.bj.bcebos.com
Date: Wed, 06 Apr 2016 06:34:40 GMT
Authorization: <AuthorizationString>
Content-Type:text/plain
Content-Length: 0

{
    "enableMultiAz":true,
}
```

- 响应示例

```
HTTP/1.1 200 OK
x-bce-request-id: 4db2b34d-654d-4d8a-b49b-3049ca786409
Date: Wed, 06 Apr 2016 06:34:40 GMT
Location: bj
Content-Length: 0
Connection: close
Server: BceBos
```

HeadBucket

接口描述

本接口用于查看Bucket是否存在和请求者是否有权限访问这个Bucket。当请求返还200 OK时，说明Bucket存在且请求者有权限访问。

请求（Request）

- 请求语法

```
HEAD / HTTP/1.1
Host: <BucketName>.bj.bcebos.com
Date: <Date>
Authorization: <AuthorizationString>
```

- 请求头域

无特殊Header参数

- 请求参数

无特殊参数

响应（Response）

- 响应头域

无特殊Header参数返回

- 响应元素

无

示例

- 请求示例

```
HEAD / HTTP/1.1
Host: BucketName.bj.bcebos.com
Date: Wed, 06 Apr 2016 06:34:40 GMT
Authorization: AuthorizationString
```

- 响应示例

```
HTTP/1.1 200 OK
x-bce-request-id: 4db2b34d-654d-4d8a-b49b-3049ca786409
Date: Wed, 06 Apr 2016 06:34:40 GMT
Connection: close
Server: BceBos
```

DeleteBucket

接口描述

本接口用于删除一个Bucket。在删除前需要保证此Bucket下的所有Object和未完成的三步上传Part已经已被删除，否则会删除失败。

> **说明：**删除Bucket之前确认该Bucket没有开通跨区域复制，不是跨区域复制规则中的源Bucket，否则不能删除。

请求（Request）

- 请求语法

```
DELETE / HTTP/1.1
Host: <BucketName>.bj.bcebos.com
Date: <Date>
Authorization: <AuthorizationString>
```

- 请求头域

无特殊Header参数

- 请求参数

无特殊参数

响应（Response）

- 响应头域

无特殊Header参数返回

- 响应元素

无

> **注意事项**
>
> 1. 只有Bucket的拥有者才能删除对应的Bucket，否则返回403 Forbidden，对应错误信息：AccessDenied。
> 2. 为了确保用户数据安全，防止误删除，BOS不允许用户删除一个非空的Bucket。如果用户试图删除一个存在Object的Bucket，返回409 Conflict错误，错误码：BucketNotEmpty。

示例

- 请求示例

```
DELETE / HTTP/1.1
Host: BucketName.bj.bcebos.com
Date: Wed, 06 Apr 2016 06:34:40 GMT
Authorization: AuthorizationString
```

- 响应示例

```
HTTP/1.1 204 No Content
x-bce-request-id: 4db2b34d-654d-4d8a-b49b-3049ca786409
Date: Wed, 06 Apr 2016 06:34:40 GMT
Connection: close
Server: BceBos
```

PutBucketStorageclass

接口描述

本接口用来设置Bucket的默认存储类型。

如果用户使用API、CLI或者SDK上传的Object未指定存储类型，则继承Bucket的默认存储类型。如果上传Object指定的存储类型和Bucket默认存储类型不一致时，以Object的存储类型为准。存储类型包含标准存储、低频存储、冷存储和归档存储四种，具体使用场景和性能请参见分级存储。

请求（Request）

- 请求语法

```
PUT /?storageClass HTTP/1.1
Host: <BcuketName>.bj.bcebos.com
x-bce-date: date
Content-Length: request-body-length
Content-Type: application/json; charset=utf-8
Authorization: AuthorizationString
```

- 请求头域

无特殊参数

- 请求参数

名称	类型	描述	是否必须
storageClass	String	指定Bucket的默认存储类型，STANDARD代表标准存储，STANDARD_IA代表低频存储，COLD代表冷存储，ARCHIVE代表归档类型；对于多AZ类型bucket，只允许设置MAZ_STANDARD和MAZ_STANDARD_IA两种存储类型，MAZ_STANDARD代表多AZ标准存储，MAZ_STANDARD_IA代表多AZ低频存储。	是

响应（Response）

- 响应元素

无特殊元素

- 响应头域

无特殊头域

示例

- 请求示例

```
PUT /?storageClass HTTP/1.1
Host: <BcuketName>.bj.bcebos.com
x-bce-date: date
Content-Length: 38
Content-Type: application/json; charset=utf-8
Authorization: AuthorizationString

{
    "storageClass": "STANDARD_IA"
}
```

- 响应示例

```
HTTP/1.1 200 OK
x-bce-request-id: 4db4b34d-653d-4d9a-b49b-3049ca786409
Date: Wed, 31 May 2017 08:34:40 GMT
Connection: keep-alive
Server: BceBos
```

GetBucketStorageClass

接口描述

本接口用来获取Bucket的默认存储类型。

请求（Request）

- 请求语法

```
GET /?storageClass HTTP/1.1
Host: <BcuketName>.bj.bcebos.com
x-bce-date: date
Content-Type: application/json; charset=utf-8
Authorization: AuthorizationString
```

- 请求头域

  无特殊参数

- 请求参数

  无特殊参数

响应（Response）

- 响应头域

  无特殊头域

- 响应元素

名称	类型	描述
storageClass	String	存储类型。标准存储返回STANDARD，低频存储返回STANDARD_IA，冷存储返回COLD，归档存储返回ARCHIVE。

示例

- 请求示例

```
GET /?storageClass HTTP/1.1
Host: <BcuketName>.bj.bcebos.com
x-bce-date: Wed, 31 May 2017 09:34:40 GMT
Authorization: AuthorizationString
```

- 响应示例

```
HTTP/1.1 200 OK
x-bce-request-id: 2eb2b34d-654d-fd8a-b49b-3049ca786409
Date: Wed, 31 May 2017 09:34:40 GMT
Content-Type: application/json; charset=utf-8
Content-Length: 31
Connection: keep-alive
Server: BceBos

{
    "storageClass": "COLD"
}
```

权限控制

PutBucketAcl

接口描述

本接口用于设置Bucket的访问权限。目前BOS支持两种方式设置ACL。第一种是使用CannedAcl，在PutBucketAcl的时候，通过头域的"x-bce-acl"来设置，当前可设置的权限包括：private, public-read, public-read-write（大小写敏感）。第二种方式是上传一个ACL文件，文件格式参见ACL文件格式。

ACL文件内容可以使用BOS ACL编辑工具生成。

> **注意：**
> BOS系统不支持在同一请求中，同时设置"x-bce-acl"和上传ACL文件。

请求（Request）

- 请求语法

```
PUT /?acl HTTP/1.1
Host: <BucketName>.bj.bcebos.com
x-bce-date: <Date>
Content-Length: <ContentLength>
Content-Type:application/json; charset=utf-8
Authorization: <AuthorizationString>
x-bce-acl: <ACLString>
```

- 请求参数

  无特殊参数

- 请求头域

名称	类型	描述	是否必需
x-bce-acl	String	Bucket设置的ACL权限，支持：private、public-read、public-read-write	否

响应（Response）

- 响应头域

  无特殊Header参数返回

- 响应元素

  无

> **注意事项**
>
> 1. 只有Bucket的拥有者和被授予FULL_CONTROL权限的用户才能设置Bucket的ACL权限。
> 2. 在创建Bucket时，Bucket权限会默认设置为private。

示例

- 使用CannedAcl的请求示例

```
 PUT /?acl HTTP/1.1
Host: BucketName.bj.bcebos.com
x-bce-date: 2016-04-06T08:23:49Z
Authorization: AuthorizationString
x-bce-acl: public-read
Content-Type: application/json; charset=utf-8
Content-length: 0
```

- 上传ACL文件的示例

```
 PUT /?acl HTTP/1.1
Host: BucketName.bj.bcebos.com
x-bce-date: 2016-04-06T08:23:49Z
Content-Length :1324
Content-Type: application/json; charset=utf-8
Authorization: AuthorizationString

{
    "accessControlList":[
        {
            "grantee":[{
              "id":"168bf6fd8fa74d9789f35a283a1f15e2"
            }],
            "permission":["READ"]
        }
    ]
}
```

- 响应示例

```
 HTTPS/1.1 200 OK
x-bce-request-id: 4db2b34d-654d-4d8a-b49b-3049ca786409
Content-Length: 0
Date: Wed, 06 Apr 2016 06:34:40 GMT
Server: BceBos
```

GetBucketAcl

接口描述

本接口用来获取某个Bucket的访问权限。

请求（Request）

- 请求语法

```
 GET /?acl HTTP/1.1
Host: <BucketName>.bj.bcebos.com
x-bce-date: <Date>
Authorization: <AuthorizationString>
```

- 请求头域

  无特殊参数

- 请求参数

  无特殊参数

响应（Response）

- 响应元素（以JSON请求为例）

名称	类型	描述
owner	Objcet	Bucket拥有者信息
+id	String	Owner用户id
accessControlList	Array	保存acl的容器
+grantee	Array	一个被授权人
++id	String	被授权用户id
+permission	Array	授权权限支持：FULL_CONTROL，READ，WRITE、LIST、MODIFY、GetObject、PutObject、DeleteObject、RestoreObject等

- 响应头域

  无

示例

- 请求示例

```
 GET /?acl HTTP/1.1
Host: BucketName.bj.bcebos.com
x-bce-date: 2016-04-06T08:23:49Z
Authorization: AuthorizationString
```

- 响应示例

```
 HTTP/1.1 200 OK
x-bce-acl: public-read
x-bce-request-id: 4db2b34d-654d-4d8a-b49b-3049ca786409
Date: Wed, 06 Apr 2016 06:34:40 GMT
Server: BceBos

{
    "owner":{
        "id":"16df583fe6824d73a5f858f06de0af03"
    }
    "accessControlList":[
        {
            "grantee":[{
                "id":"168bf6fd8fa74d9789f35a283a1f15e2"
            }],
            "permission":["FULL_CONTROL"]
        },
        {
            "grantee":[{
                "id":"10eb6f5ff6ff4605bf044313e8f3ffa5"
            }],
            "permission":["READ"]
        }
    ],
}
```

生命周期

PutBucketLifecycle

接口描述

本接口用来创建生命周期管理规则。

> **注意:**
>
> - 只有bucket的owner且拥有FULL_CONTROL权限才能够进行此请求。
> - PutBucketLifecycle会覆盖原有的生命周期规则， 如果需要在原有规则基础上新增， 需要在请求中携带所有新旧规则。

请求（Request）

- 请求语法

```
 PUT /?lifecycle HTTP/1.1
Host: <BucketName>.bj.bcebos.com
x-bce-date: date
Content-Length: request-body length
Content-Type: application/json; charset=utf-8
Authorization: AuthorizationString
{
    "rule": [{
    "id": "rule-id",
    "status": "enabled",
    "resource": [
        "bucket/prefix/*"
    ],
    "condition": {
        "time": {
            "dateGreaterThan": "2016-09-07T00:00:00Z"
        }
    },
    "action": {
        "name": "DeleteObject"
    }
   }]
}
```

- 请求头域

  无特殊参数

- 请求参数

规则项	描述	是否必须	备注
rule	规则的列表	是	所有规则的列表
+id	规则的标识符。	是	同一个bucket内规则id必须唯一，不能重复。如果用户不填系统会自动帮用户生成一个。
+status	规则的状态。	是	取值为enabled或disabled，当规则处于disabled时规则不生效。
+ExpiredObjectDeleteMarker	删除标记过期标志。	否	取值为true或false，当规则处于false时规则不生效。
+resource	规则对哪些资源生效。	是	举例：对samplebucket里以prefix/为前缀的Object生效：samplebucket/prefix/*；对samplebucket里所有Object生效：samplebucket/*。
+condition	规则依赖的条件。	是	目前只支持time形式，智能分层规则返回值为null
++time	时间限制条件。	否	通过定义的dateGreaterThan实现。
+++dateGreaterThan	描述时间关系。	否	支持绝对时间date和相对时间days。绝对时间date格式为yyyy-mm-ddThh:mm:ssZ，eg. 2016-09-07T00:00:00Z。绝对时间为UTC时间, 必须以00:00:00(UTC 0点)结尾；相对时间days的描述遵循ISO8601, 支持的最小时间粒度为天，如: $(lastModified)+P7D表示的时间为object的last-modified之后7天。
++tag	对象标签限制条件	否	举例: 对这个prefix下的所有满足对象标签规则的Object才生效，如果tag为空，默认应用所有Object。
++objectSize	对象大小限制条件	否	举例: 对这个prefix下的所有满足对象大小规则的Object才生效，如果objectSize为空，默认应用所有Object的大小都符合条件。
+action	对resource执行的操作动作。	是	-
+name	执行的操作名称。	是	取值为Transition、DeleteObject、AbortMultipartU、IntelligentTiering、NonCurrentVersionDelet、NonCurrentVersionTransition。
+storageClass	Object的存储类型	否	action为Transition时可以设定，非多AZ bucket的取值为STANDARD_IA或COLD或ARCHIVE，表示从原存储类型转为低频存储或冷存储或归档存储；对于多AZ的bucket，取值只能为STANDARD_IA，表示从原存储类型转为低频存储，智能分层规则返回null。

响应（Response）

- 响应元素

  无特殊元素

- 响应头域

  无特殊头域

基础生命周期管理请求示例

- 请求示例

```
 PUT /?lifecycle HTTP/1.1
Host: bucket.bj.bcebos.com
x-bce-date: 2016-08-16T08:23:49Z
Content-Length :1324
Content-Type: application/json; charset=utf-8
Authorization: AuthorizationString

{
    "rule": [
        {
            "id": "sample-rule-delete-prefix",
            "status": "enabled",
            "resource": [
                "bucket/prefix/*"
            ],
            "condition": {
                "time": {
                    "dateGreaterThan": "2016-09-07T00:00:00Z"
                }
            },
            "action": {
                "name": "DeleteObject"
            }
        },
        {
            "id": "sample-rule-transition-prefix",
            "status": "enabled",
            "resource": [
                "bucket/prefix/*"
            ],
            "condition": {
                "time": {
                    "dateGreaterThan": "$(lastModified)+P7D"
                }
            },
            "action": {
                "name": "Transition",
                "storageClass": "STANDARD_IA"
            }
        },
        {
            "id": "sample-rule-abort-multiupload-prefix",
            "status": "enabled",
            "resource": [
                "bucket/prefix/*"
            ],
            "condition": {
                "time": {
                    "dateGreaterThan": "$(lastModified)+P7D"
                }
            },
            "action": {
                "name": "AbortMultipartUpload"
            }
        }
    ]
}
```

- 响应示例

```
 HTTP/1.1 200 OK
x-bce-request-id: 0A49CE4060975EAC
Date: Wed, 12 Oct 2016 17:50:00 GMT
Content-Length: 0
Connection: keep-alive
Server: BceBos
```

智能分层管理请求示例

- 请求示例

```
 PUT /?lifecycle HTTP/1.1
Host: bucket.bj.bcebos.com
x-bce-date: 2022-07-26T17:18:55Z
Content-Length: 576
Content-Type: application/json; charset=utf-8
Authorization: AuthorizationString

{
    "rule": [
    {
        "action": {
            "name": "IntelligentTiering"
        },
        "id": "sample-rule-intelligent-tiering-prefix",
        "resource": [
            "bucket/*"
        ],
        "status": "enabled"
    },
    {
        "id": "sample-rule-delete-prefix",
        "status": "enabled",
        "resource": [
            "bucket/prefix/*"
        ],
        "condition": {
            "time": {
                "dateGreaterThan": "2022-07-27T00:00:00Z"
            }
        },
       "action": {
           "name": "DeleteObject"
       }
    },
    {
        "id": "sample-rule-abort-multiupload-prefix",
        "status": "enabled",
        "resource": [
            "bucket/prefix/*"
        ],
        "condition": {
            "time": {
                "dateGreaterThan": "$(lastModified)+P7D"
            }
        },
        "action": {
        "name": "AbortMultipartUpload"
      }
    }
  ]
}
```

- 响应示例

```
  ```
 HTTP/1.1 200 OK
 x-bce-request-id: FSZ89ESW7DS6FA9
 Date: Wed, Tue, 26 Jul 2022 05:01:28 GMT
 Content-Length: 0
 Connection: keep-alive
 Server: BceBos
```
```

复杂生命周期管理请求示例

- 请求示例

```
 PUT /?lifecycle HTTP/1.1
Host: bucket.bj.bcebos.com
x-bce-date: 2025-02-05T08:23:49Z
Content-Length :1345
Content-Type: application/json; charset=utf-8
Authorization: AuthorizationString

{
  "rule": [{
  "id": "rule-id",
  "status": "enabled",
  "ExpiredObjectDeleteMarker" :"true",
  "resource": [
     "bucket/prefix/*"
  ],
  "condition": {
     "time": {
        "dateGreaterThan": "$(lastModified)+P3D"
     },
     "objectSize":{
        "minSize":1024,
        "maxSize":2048
     },
     "tag": {
        "key1":"value1",
        "key2":"value2"
     }
  },
  "action": {
     "name": "DeleteObject"
  },
  "not":{ // 一个规则只有一个not prefix+ not tag 其中prefix和tag 必须存在一项
     "resource": "bucket/prefix/prefix1*",
     "tag": {"key3":"value3"}
  }
 },{
  "id": "rule-id2",
  "status": "enabled",
  "resource": [
     "bucket/*"
  ],
  "condition": {
     "time": {
        "dateGreaterThan": "$(lastModified)+P5D"
     },
     "tag": {
        "key1":"value1",
        "key2":"value2"
     }
  },
  "action": {
     "name": "NonCurrentVersionDeleteObject"
  }

 }]
}
```

- 响应示例

```
 HTTP/1.1 200 OK
x-bce-request-id: 0A49CE4060975EAC
Date: Fri, 5 Feb 2025 08:23:00 GMT
Content-Length: 0
Connection: keep-alive
Server: BceBos
```

GetBucketLifecycle

接口描述

此接口用于获取定义的生命周期管理规则详细信息。

请求（Request）

- 请求语法

```
 GET /?lifecycle HTTP/1.1
Host: <BucketName>.bj.bcebos.com
Date: date
Authorization: AuthorizationString
```

- 请求头域

无特殊参数

- 请求参数

无特殊参数

响应（Response）

- 请求参数

规则项	描述	是否必填	备注
rule	规则的列表	必填	所有规则的列表
+id	规则的标识符。	必填	同一个bucket内规则id必须唯一，不能重复。如果用户不填系统会自动帮用户生成一个。
+status	规则的状态。	必填	取值为enabled或disabled，当规则处于disabled时规则不生效。

| +resource | 规则对哪些资源生效。 | 必填 | 举例：对samplebucket里以prefix/为前缀的Object生效：samplebucket/prefix/*；对samplebucket里所有Object生效：samplebucket/*。 | | +condition | 规则依赖的条件。 | 选填 | 目前只支持time形式，基础生命周期规则下为必填项。 |+ExpiredObjectDeleteMarker | 删除标记过期标志。|可选| 取值为true或者false|| | +time | 时间限制条件。 | 选填 | 通过定义的dateGreaterThan实现，智能分层配置下无填充参数 | | ++dateGreaterThan | 描述时间关系。 | 必填 | 支持绝对时间date和相对时间days。绝对时间date格式为yyyy-mm-ddThh:mm:ssZ，eg. 2016-09-07T00:00:00Z。绝对时间为UTC时间, 必须以00:00:00(UTC 0点)结尾；相对时间days的描述遵循ISO8601, 支持的最小时间粒度为天，如: $(lastModified)+P7D表示的时间为object的last-modified之后7天。 | | action | 对resource执行的操作动作。 | 选填 | 基础生命周期规则下为必填项。 | | +name | 执行的操作名称。 | 必填 | 取值为Transition、DeleteObject、AbortMultipartUpload。 | | +storageClass | Object的存储类型 | 可选 | action为Transition时可以设定，取值为STANDARD_IA或COLD或ARCHIVE，表示从原存储类型转为低频存储或冷存储或归档存储。

- 响应元素

  无特殊元素

- 响应头域

  无特殊头域

注意事项

- 如果请求的源Bucket不存在，返回404错误，错误码为NoSuchBucket。
- 如果请求的源Bucket没有配置lifecycle，返回404错误，错误码为NoLifecycleConfiguration。

基础生命周期请求示例

- 请求示例

```
 GET /?lifecycle HTTP/1.1
Host: <BucketName>.bj.bcebos.com
Date: Thu, 15 Sep 2016 00:16:26 GMT
Authorization: AuthorizationString
```

- 响应示例

```
 HTTP/1.1 200 OK
Date: Thu, 15 Sep 2016 00:17:23 GMT
Server: BceBos
x-bce-request-id: 1a5fd81e-626b-45b3-a885-15fff9cd106c
Connection: keep-alive
Content-Length: 358

{
    "rule": [
        {
            "id": "sample-rule-delete-prefix",
            "status": "enabled",
            "resource": [
                "bucket/prefix/*"
            ],
            "condition": {
                "time": {
                    "dateGreaterThan": "2016-09-07T00:00:00Z"
                }
            },
            "action": {
                "name": "DeleteObject"
            }
        },
        {
            "id": "sample-rule-transition-prefix",
            "status": "enabled",
            "resource": [
                "bucket/prefix/*"
            ],
            "condition": {
                "time": {
                    "dateGreaterThan": "$(lastModified)+P7D"
                }
            },
            "action": {
                "name": "Transition",
                "storageClass": "STANDARD_IA"
            }
        },
        {
            "id": "sample-rule-abort-multiupload-prefix",
            "status": "enabled",
            "resource": [
                "bucket/prefix/*"
            ],
            "condition": {
                "time": {
                    "dateGreaterThan": "$(lastModified)+P7D"
                }
            },
            "action": {
                "name": "AbortMultipartUpload"
            }
        }
    ]
}
```

智能分层请求示例

- 请求语法

```
 GET /?lifecycle HTTP/1.1
Host: <BucketName>.bj.bcebos.com
Date: Tue, 26 Jul 2022 05:01:28 GMT
Authorization: AuthorizationString
```

- 响应示例

```
HTTP/1.1 200 OK
Date: Tue, 26 Jul 2022 09:41:29 GMT
Server: BceBos
x-bce-request-id: ac07037d-f40e-45e0-b126-e38adc238cad
Connection: keep-alive
Content-Length: 740

{
  "rule": [
    {
      "id": "sample-rule-intelligent-tiering-prefix",
      "status": "enabled",
      "resource": [
        "bucket/*"
      ],
      "action": {
        "name": "IntelligentTiering"
      }
    },
    {
      "id": "sample-rule-delete-prefix",
      "status": "enabled",
      "resource": [
        "bucket/*"
      ],
      "condition": {
        "time": {
          "dateGreaterThan": "$(lastModified)+P365D"
        }
      },
      "action": {
        "name": "DeleteObject"
      }
    },
    {
      "id": "sample-rule-abort-multiupload-prefix",
      "status": "enabled",
      "resource": [
        "bucket/*"
      ],
      "condition": {
        "time": {
          "dateGreaterThan": "$(lastModified)+P7D"
        }
      },
      "action": {
        "name": "AbortMultipartUpload"
      }
    }
  ]
}
```

DeleteBucketLifecycle

接口描述

本接口用来删除定义的生命周期管理规则。

请求（Request）

- 请求语法

```
DELETE /?lifecycle HTTP/1.1
Host: <BucketName>.bj.bcebos.com
Date: GMT Date
Authorization: AuthorizationString
```

- 请求头域

  无特殊参数

- 请求参数

  无特殊参数

响应（Response）

- 响应元素

  无特殊元素

- 响应头域

  无特殊头域

示例

- 请求示例

```
DELETE /?lifecycle HTTP/1.1
Host: <BcuketName>.bj.bcebos.com
Date: Tue, 17 Sep 2016 08:36:52 GMT
Authorization: AuthorizationString
```

- 响应示例

```
HTTP/1.1 204 No Content
Date: Wed, 14 Sep 2016 05:37:16 GMT
x-bce-request-id: 1a5fd81e-626b-45b3-a885-15fff9cd106c
Connection: keep-alive
Server: BceBos
```

日志管理

PutBucketLogging

接口描述

本接口用来开启Bucket的访问日志并指定存放日志的Bucket和访问日志的文件前缀。访问日志的规则请参见日志命名规则和日志格式。

请求（Request）

- 请求语法

```
PUT /?logging HTTP/1.1
Host: <BucketName>.bj.bcebos.com
Date: <Date>
Authorization: <AuthorizationString>
Content-Length: <ContentLength>

{
 "targetBucket": "TargetBucketName",
 "targetPrefix": "TargetPrefixName"
}
```

- 请求头域

  无特殊参数

- 请求参数

名称	描述	类型	是否必须
targetBucket	指定存放访问日志的Bucket	字符串	是
targetPrefix	指定最终被保存的访问日志文件前缀	字符串	否（建议填写，用以区分访问日志）

响应（Response）

- 响应元素

  无特殊元素

- 响应头域

  无特殊头域

注意事项

- 用户必须是源Bucket的owner且拥有FULL_CONTROL权限，且是目标Bucket的owner。
- 源Bucket和目标Bucket必须同时存在。
- 源Bucket和目标Bucket必须属于同一个Region。
- 如果HTTP Body中Json不合法，BOS会返回CODE_MALFORMED_JSON错误。
- 如果HTTP Body中Json有无效字段，BOS会返回CODE_INAPPROPRIATE_JSON错误。
- 用户可将不同的源Bucket的Logging都保存在同一个目标Bucket内，建议指定不同的 targetPrefix便于区分。
- 如果源Bucket开通了Logging功能，源Bucket被删除的同时，相应的Logging信息也将被删除。
- 如果Logging的目标Bucket被删除，则源Bucket的Logging功能会被自动关闭。
- 如果需要修改目标Bucket等信息，可再发送一个PutBucketLogging请求，请求中包含需要修改的信息。
- targetPrefix表示存储访问日志记录的Object名字前缀，可以为空。如果不为空时，targetPrefix可以包含字母、数字、中划线、下划线、斜杠，且必须以字母开头，长度不大于32位。
- 重复请求返回结果相同。

示例

- 请求示例

```
PUT /?logging HTTP/1.1
Host: BucketName.bj.bcebos.com
Date: Tue, 17 May 2016 08:36:52 GMT
Authorization: AuthorizationString
Content-Length: 55

{
 "targetBucket": "dscbucket",
 "targetPrefix": "mylog/"
}
```

- 响应示例

```
HTTP/1.1 200 OK
x-bce-request-id: 1a5fd81e-626b-45b3-a885-15fff9cd106c
Date: Tue, 17 May 2016 08:36:52 GMT
Content-Length: 0
Server: BceBos
```

GetBucketLogging

接口描述

本接口用来获取某个Bucket的访问日志配置情况。

请求（Request）

- 请求语法

```
Get /?logging HTTP/1.1
Host: <BucketName>.bj.bcebos.com
Date: <Date>
Authorization: <AuthorizationString>
Content-Length: 0
```

- 请求头域

无特殊参数

- 请求参数

无特殊参数

响应（Response）

- 响应元素

名称	描述
status	Bucket的访问日志功能的开启状态，取值为enabled和disabled。enabled代表已开启，disabled代表未开启。
targetBucket	指定存放访问日志的Bucket，如果未开启Logging功能，响应中无该字段。
targetPrefix	指定最终被保存的访问日志文件前缀，如果未开启Logging功能，响应中无该字段。

- 响应头域

无特殊头域

注意事项

- 如果请求的源Bucket不存在，返回404错误，错误码为NoSuchBucket。
- 请求者只有是源Bucket的owner且拥有FULL_CONTROL权限才允许查看，否则返回403错误，错误码为AccessDenied。

示例

- 请求示例

```
Get /?logging HTTP/1.1
Host: BucketName.bj.bcebos.com
Date: Tue, 17 May 2016 08:36:52 GMT
Authorization: AuthorizationString
Content-Length: 0
```

- 响应示例（已设置Log规则）

```
HTTP/1.1 200 OK
x-bce-request-id: 1a5fd81e-626b-45b3-a885-15fff9cd106c
Date: Tue, 17 May 2016 08:36:52 GMT
Content-Length: 71
Server: BceBos

{
 "status": "enabled",
 "targetBucket": "dscbucket",
 "targetPrefix": "mylog-"
}
```

- 响应示例（未设置Log规则）

```
HTTP/1.1 200 OK
x-bce-request-id: 1a5fd81e-626b-45b3-a885-15fff9cd106c
Date: Tue, 17 May 2016 08:36:52 GMT
Content-Length: 21
Server: BceBos

{
 "status": "disabled"
}
```

DeleteBucketLogging

接口描述

本接口用来关闭Bucket访问日志记录功能。

请求（Request）

- 请求语法

```
DELETE /?logging HTTP/1.1
Host: <BucketName>.bj.bcebos.com
Date: <Date>
Authorization: &lt;AuthorizationString>
```

- 请求头域

无特殊参数

- 请求参数

  无特殊参数

响应（Response）

- 响应元素

  无特殊元素

- 响应头域

  无特殊头域

注意事项

- 如果请求的源Bucket不存在，返回404错误，错误码为NoSuchBucket。
- 请求者只有是源Bucket的owner且拥有FULL_CONTROL权限，才能关闭Bucket访问日志记录功能。否则，BOS返回403错误，错误码为AccessDenied。
- 如果请求的源Bucket没有开通Logging功能，依然返回HTTP状态码204。

示例

- 请求示例

```
DELETE /?logging HTTP/1.1
Host: BucketName.bj.bcebos.com
Date: Tue, 17 May 2016 08:36:52 GMT
Authorization: AuthorizationString
```

- 响应示例

```
HTTP/1.1 204 No Content
x-bce-request-id: 1a5fd81e-626b-45b3-a885-15fff9cd106c
Date: Tue, 17 May 2016 08:36:52 GMT
Connection: close
Server: BceBos
```

事件通知

PutNotification

接口描述

本接口用于指定bucket上增加通知规则。

> **注意：**
>
> - 只有bucket owner或者full control权限才能获取这个bucket的配置。
> - 如果不是bucket owner则返回403，如果对应的文件不存在则返回404。

请求

- 请求语法

```
PUT /?notification HTTP/1.1
Host: <BucketName>.bj.bcebos.com
Date: <Date>
Authorization: <AuthorizationString>
Content-Type: application/json; charset=utf-8
Content-Length: <ContentLength>
```

- 请求参数

字段	类型	必要性	说明
id	String	必选	规则id
name	String	可选	规则名称
appId	String	必选	注册本条规则的产品id
status	String	必选	可选值：{"disabled", "enabled"}
encryption	Object	可选	加密方式
+key	String	可选	加密密钥，如果不为空，则用IAM的算法对通知的请求进行签名，key对应IAM签名中的SecretAccessKey
resources[]	Array	必选	订阅的资源
+resource	String	必选	订阅的资源，`${bucket_name}/path1/*.jpg`或者`/path1/*.jpg`，最多只能有1个`*`
events[]	Array	必选	订阅的事件
+eventType	String	必选	事件类型，当前支持： - PutObject:普通上传Object - PostObject:表单上传Object - AppendObject:追加上传Object - CopyObject:拷贝Object - CompleteMultipartUpload:完成Object分片上传 - FetchObject:抓取Object，包含镜像回源等产生的抓取 - DeleteObject:删除Object - DeleteMultipleObjects:删除多个Object - `"BosLifecycle:Copy"`:生命周期Lifecycle触发的复制事件,包含CopyObect请求和分块复制第三步CompleteMultipartUpload请求。 - `"ObjectCreateGroup:*"`:所有创建新文件操作。 - `"ObjectModifyGroup:*"`:所有修改文件操作，包括覆盖文件、设置文件自定义meta、更改文件存储类型、更改文件加密算法的操作。
apps[]	Array	必选	订阅消息的产品
+id	String	必选	被通知的产品id
+eventUrl	String	必选	处理事件通知的url，可选值：{"http", "https", "brn", "app"}，http/https为自定义应用，brn为cfc的通知，app为官方应用
+xVars	String	可选	透传的自定义参数，对框架透明，用于业务自我回传的值，如果是官方AI应用，则是一个字符串化的json，并包含一个saveUrl的地址，用于接收处理结果

如果仅用于配置通知或者回调的密钥，可以简化为如下参数：

字段	类型	必要性	说明
id	String	必选	规则id
name	String	可选	规则名称
appId	String	必选	注册本条规则的产品id
status	String	必选	可选值：{"disabled", "enabled"}
encryption	Object	可选	加密方式
+key	String	可选	加密密钥，如果不为空，则用IAM的算法对通知的请求进行签名，key对应IAM签名中的SecretAccessKey

- 请求头域

  无

响应

- 响应头域

  无

- 响应元素

  无

示例

- 请求示例

```
 PUT /?notification HTTP/1.1
Host:  <BucketName>.bj.bcebos.com
Date: Wed, 12 Sep 2018 06:34:40 GMT
Authorization: <AthorizationString>
Content-Type: application/json; charset=utf-8
Content-Length: 0

{
   "notifications": [
      {
         "id": "notify-id-1",
         "name": "rule-name",
         "appId": "app-id-1",
         "status": "enabled",
         "encryption": {
            "key": "06a62b70f47dc4a0a7da349609f1a1ac",
         },
         "resources": [
            "bucket-a/path1", "/path2", "/path3/*.jpg", "/path4/*"
         ],
         "events": [
            "PutObject"
         ],
         "apps": [
            {
               "id": "app-id-1",
               "eventUrl": "http://xxx.com/event",
               "xVars": ""
            },
            {
               "id": "app-id-2",
               "eventUrl": "brn:bce:cfc:bj:1f1c3e38xxxxxxx4c44523f0d5b22:function:hello_bos:$LATEST"
            },
            {
               "id": "app-id-3",
               "eventUrl": "app:ImageOcr",
               "xVars": "{\"saveUrl\": \"http://xxx.com/ocr\"}"
            }
         ]
      }
   ]
}
```

- 响应示例

```
 HTTP/1.1 200 OK
x-bce-request-id: 4db2b34d-654d-4d8a-b49b-xxxxxx786409
Date: Wed, 12 Sep 2018 06:34:40 GMT
Content-Length: 0
Connection: close
Server: BceBos
```

GetNotification

接口描述

本接口用于获取指定bucket上的通知规则。

请求

- 请求语法

```
 http
GET /?notification HTTP/1.1
Host: <BucketName>.bj.bcebos.com
Date: <Date>
Authorization: <AuthorizationString>
```

- 请求头域

  无

- 请求参数

  无

响应

- 响应头域

  无

- 响应元素

字段	类型	必要性	说明
id	String	必选	规则id
name	String	可选	规则名称
appId	String	必选	注册本条规则的产品id
status	String	必选	可选值：{"disabled", "enabled"}
encryption	Object	可选	加密方式
+key	String	可选	加密密钥，如果不为空，则用IAM的算法对通知的请求进行签名，key对应IAM签名中的SecretAccessKey
resources[]	Array	必选	订阅的资源
+resource	String	必选	订阅的资源，${bucket_name}/path1/*.jpg或者/path1/*.jpg，最多只能有1个*
events[]	Array	必选	订阅的事件
+eventType	String	必选	事件类型，当前支持： - PutObject:普通上传Object - PostObject:表单上传Object - AppendObject:追加上传Object - CopyObject:拷贝Object - CompleteMultipartUpload:完成Object分片上传 - FetchObject:抓取Object，包含镜像回源等产生的抓取 - DeleteObject:删除Object - DeleteMultipleObjects:删除多个Object。 - "BosLifecycle:Copy":生命周期Lifecycle触发的复制事件,包含CopyObect请求和分块复制第三步CompleteMultipartUpload请求。
apps[]	Array	必选	订阅消息的产品
+id	String	必选	被通知的产品id
+eventUrl	String	必选	处理事件通知的url，可选值：{"http", "https", "brn", "app"}，http/https为自定义应用，brn为cfc的通知，app为官方应用
+xVars	String	可选	透传的自定义参数，对框架透明，用于业务自我回传的值，如果是官方AI应用，则是一个字符串化的json，并包含一个saveUrl的地址，用于接收处理结果

示例

- 请求示例

```
 http
GET /?notification HTTP/1.1
Host: <BucketName>.bj.bcebos.com
Date: Wed, 12 Sep 2018 06:34:40 GMT
Authorization: AuthorizationString
```

- 响应示例

```
 HTTP/1.1 200 OK
x-bce-request-id: 4db2b34d-654d-4d8a-b49b-xxxxxx786409
Date: Wed, 12 Sep 2018 06:34:40 GMT
Content-Length: 0
Connection: close
Server: BceBos

{
   "notifications": [
      {
         "id": "notify-id-1",
         "name": "rule-name",
         "appId": "app-id-1",
         "status": "enabled",
         "resources": [
            "bucket-a/path1", "/path2", "/path3/*.jpg", "/path4/*"
         ],
         "events": [
            "PutObject"
         ],
         "apps": [
            {
               "id": "app-id-1",
               "eventUrl": "http://xxx.com/event",
               "xVars": ""
            },
            {
               "id": "app-id-2",
               "eventUrl": "brn:bce:cfc:bj:1f1c3e383c31e6467c4c44523f0d5b22:function:hello_bos:$LATEST"
            },
            {
               "id": "app-id-3",
               "eventUrl": "app:ImageOcr",
               "xVars": "{\"saveUrl\": \"http://xxx.com/ocr\"}"
            }
         ]
      }
   ]
}
```

DeleteNotification

接口描述

本接口用于删除指定bucket上的通知规则。

请求

- 请求语法

```
 http
DELETE /?notification HTTP/1.1
Host: <BucketName>.bj.bcebos.com
Date: <Date>
Authorization: <AuthorizationString>
```

- 请求头域

  无

- 请求参数

  无

响应

- 响应头域

  无

- 响应元素

  无

示例

- 请求示例

```
 http
DELETE /?notification HTTP/1.1
Host: <BucketName>.bj.bcebos.com
Date: Wed, 12 Sep 2018 06:34:40 GMT
Authorization: AuthorizationString
```

- 响应示例

```
 HTTP/1.1 200 OK
x-bce-request-id: 4db2b34d-654d-4d8a-b49b-xxxxxx786409
Date: Wed, 06 Apr 2016 06:34:40 GMT
Content-Length: 0
Connection: close
Server: BceBos
```

PostEvent

接口描述

将事件消息推送到用户配置的回调url上。回调是发送POST请求，用户的回调服务器需要支持POST请求、

如果规则中配置了encryption字段，那么请求中会包含Authorization的签名，保证消息不会被伪造或篡改。签名是可选的。

> **注意：**
>
> - Status Code: 200 OK才会认为消息推送成功，否则将会重试；
> - 对于过载保护的情况，支持Status Code: 429 Too Many Requests返回值，会间隔一段时间再重试；

请求

- 请求语法

```
 http
POST /?event HTTP/1.1
Host: <User_Host>
Date: <Date>
Content-Type: application/json; charset=utf-8
Content-Length: <Content_Length>
Authorization: <AuthorizationString>
```

- 请求头域

  无

- 请求参数

字段	类型	说明
version	String	事件版本，当前为1.0
eventFrom	String	触发事件的请求来源，表示请求来源于用户、BOS生命周期、BOS数据同步、BOS批处理。 有效值: Client/Lifycycle/Replication/Batch。 默认是"Client"，如果没有设置此字段则表示取值是默认值。
eventId	String	事件唯一标识
eventOrigin	String	事件触发源，当前支持增量(bos:realtime)
eventType	String	事件类型
eventTime	String	事件发生时间(GMT格式)
configurationId	String	事件对应的事件通知规则id。
content	Object	事件具体信息
+domain	String	所属的domain
+bucket	String	所属的bucket
+object	String	object名字
+eTag	String	object的eTag，如果object可能被覆盖，则需要用eTag判断是哪个版本的object触发的事件
+contentType	String	object的Content-Type
+copySourceBucket	string	如果是CopyObject事件或BosLifecycle:Copy事件，表示源object所在bucket名称。
+copySourceObject	string	如果是CopyObject事件或者BosLifecycle:Copy事件，表示源Object名称。
+copySourceStorageClass	string	如果是CopyObject事件或者BosLifecycle:Copy事件，表示源object的存储类型。
+storageClass	string	触发事件的object的存储类型。
+filesize	Number	文件大小
+lastModified	String	文件更新时间
+credentials	Object	相关资源的临时授权，具体使用参考临时授权访问
++accessKeyId	String	临时授权的Access Key ID
++secretAccessKey	String	临时授权的Secret Access Key
++sessionToken	String	临时授权的Session Token
++expiration	String	临时授权的的有效时间
+xVars	String	用户设置通知中的xVars原文

响应

- 响应头域

  无

- 响应元素

  无

回调签名（可选） 用户设置事件通知规则且触发通知规则之后，BOS服务端将会按照用户设置的回调URL发送POST回调请求给用户的应用服务器。应用服务器收到回调请求之后，如果希望验证回调请求确实是由BOS发起的话，可以通过在回调中带上签名来验证BOS的身份。

回调签名使用百度智能云的IAM签名算法来实现，具体可参考IAM签名算法。

签名过程分为三步：

（1）提前创建事件通知规则,并在规则中设置密钥"encryption"。用户可以通过控制台或者sdk创建。

例如创建事件通知规则的id为"callback1"，其中key为用户设置密钥，长度为32字符，可取值为数字[0-9]和小写英文字母。

```
 PUT /?notification HTTP/1.1
Host:  <BucketName>.bj.bcebos.com
Date: Wed, 12 Sep 2018 06:34:40 GMT
Authorization: <AthorizationString>
Content-Type: application/json; charset=utf-8
Content-Length: 0
{
   "notifications": [
      {
         "id": "callback1",
         "name": "rule-name",
         "appId": "app-id-1",
         "status": "enabled",
         "encryption": {
            "key": "06a62b70f47dc4a0a7da349609f1a1ac",
         },
         "resources": [
            "bucket-a/path1/*", "/path2/*.jpg"
         ],
         "events": [
            "PutObject"
         ],
         "apps": [
            {
               "id": "app-id-1",
               "eventUrl": "http://xxx.com/event",
               "xVars": "<User-Define-Raw-Msg>"
            }
         ]
      }
   ]
}
```

（2）用户发起bos请求，并且命中事件通知规则。

（3）BOS服务端向用户的应用服务器发起带签名的回调。其中计算签名的AK为用户本次请求的AK(如果没有带签名，默认使用"-")，SK为用户在第一步中设置的密钥。生成的签名位于请求头"Authorization"中。

示例

- 请求示例

```
 ```
POST /<callback_url> HTTP/1.1
Host: <User_Host>
Date: Wed, 12 Sep 2018 06:34:40 GMT
Authorization: <AthorizationString>
Content-Type: application/json; charset=utf-8
Content-Length: 0

{
   "events": [
      {
         "version": "1.0",
         "eventFrom": "Client",
         "eventId": "2a513199-bbb9-4ac7-b12a-60213c26810d",
         "eventOrigin": "bos:realtime",
         "eventTime": "2018-09-05T02:28:49Z",
         "eventType": "CopyObject",
         "configurationId": "notify-id-1",
         "content": {
            "userId": "c7ac82ae14ef42d1a4ffa3b2ececa17f",
            "domain": "bj.bcebos.com",
            "bucket": "bucket-test",
            "object": "images/test.jpg",
            "eTag": "977b9623a15e520c663ac5ff6647e881",
            "contentType": "application/octet-stream",
            "copySourceBucket": "src-bucket",
            "copySourceObject": "src-object",
            "copySourceStorageClass": "STANDARD",
            "storageClass": "COLD",
            "filesize": 24414,
            "lastModified": "2018-09-05T02:28:49Z",
            "credentials": {
               "accessKeyId": "f3ade7c6b1a911e8bb6821a724cc157d",
               "secretAccessKey": "07381a56d35e4b1193a418362221de81",
               "sessionToken": "ZGZiM2M3MmU4Mjk4NGQ2MGEzYTNhYTAyMDE3NTZmZmV8AAAAAIcCAADaE18IR6jSaVWlMHxZG5wzb/7AQas3Y6V",
               "expiration": "2018-02-22T11:22:33Z"
            },
            "xVars": "<User-Define-Raw-Msg>"
         }
      }
   ]
}
```
```

- 响应示例

```
 HTTP /1.1 200 OK
x-bce-request-id: 4db2b34d-654d-4d8a-b49b-xxxxxx786409
Date: Wed, 12 Sep 2018 06:34:40 GMT
Content-Length: 0
Connection: close
Server: BceBos
```

PostResult

接口描述

将ImageOcr和ImageClassify两种产品处理后的事件消息推送到app设置的url上，内容包含BOS的事件信息和AI处理的结果。

如果规则中配置了encryption字段，那么请求中会包含Authorization的签名，保证消息不会被伪造或篡改。

> **注意：**
>
> - Status Code: 200 OK才会认为消息推送成功，否则将会重试；
> - 对于过载保护的情况，支持Status Code: 429 Too Many Requests返回值，会间隔一段时间再重试；

请求

- 请求语法

```
 http
POST /?result HTTP/1.1
Host: <User_Host>
Date: <Date>
Content-Type: application/json; charset=utf-8
Content-Length: <Content_Length>
Authorization: <AuthorizationString>
```

- 请求参数

字段	类型	说明	
version	String	事件版本，当前为1.0	
eventType	String	事件类型	
eventTime	String	事件发生时间(GMT格式)	
content	Object	事件具体信息	
+domain	String	所属的domain	
+bucket	String	所属的bucket	
+object	String	object名字	
+eTag	String	object的eTag，如果object可能被覆盖，则需要用eTag判断是哪个版本的object触发的事件	
result	String	根据实际的事件为imageOcr和imageClassify，具体内容参见AI通用文字识别和AI通用图像分析	

- 请求头域

  无

响应

- 响应头域

  无

- 响应元素

  无

示例

- 请求示例

```
 http
POST /?result HTTP/1.1
Host: BucketName.bj.bcebos.com
Date: Wed, 12 Sep 2018 06:34:40 GMT
Content-Type: application/json; charset=utf-8
Content-Length: 0

{
    "events": [
        {
            "version": "1.0",
            "eventType": "PutObject",
            "eventTime": "2009-10-28T22:32:00Z",
            "content": {
                "domain": "bj.bcebos.com",
                "bucket": "bucket-test",
                "object": "images/test.jpg",
                "eTag": "977b9623a15e520c663ac5ff6647e881",
            },
            "imageOcr": {
                "log_id": 153673668620866,
                "result": {
                    "ocr": {
                        "log_id": 1078286758436670376,
                        "words_result": [
                            {
                                "words": "MAKE"
                            }
                        ],
                        "words_result_num": 1,
                        "direction": 0
                    }
                }
            },
            "imageClassify": {
                "log_id": 6353842115261500790,
                "result_num": 1,
                "result": [
                    {
                        "score": 0.566877,
                        "root": "人物-人物特写",
                        "keyword": "马尾辫"
                    }
                ]
            }
        }
    ]
}
```

- 响应示例

```
 HTTP/1.1 200 OK
x-bce-request-id: 4db2b34d-654d-4d8a-b49b-xxxxxx786409
Date: Wed, 12 Sep 2018 06:34:40 GMT
Content-Length: 0
Connection: close
Server: BceBos
```

静态网站

PutBucketStaticWebsite

接口描述

本接口用于设置静态网站托管。

用户必须对bucket具有full control 权限。不建议设置归档文件，归档文件没有取回时不可读，StaticWebsite此时不会生效。

请求（Request）

- 请求头域

  无特殊参数

- 请求参数

名称	描述	类型	是否必须
index	Index文件名称	string	否
notFound	404文件名称	string	否
notFoundHttpStatus	返回错误页面时的HTTP状态码。取值："200"、"404"（默认）	string	否

响应（Response）

- 响应元素

  无

- 响应头域

  无

示例

- 请求示例

```
 PUT /?website HTTP/1.1
Host: <BcuketName>.bj.bcebos.com
Date: Tue, 17 May 2016 08:36:52 GMT
Authorization: AuthorizationString
Content-Length: 65

{
    "index": "index.html",
    "notFound": "404.html",
    "notFoundHttpStatus": "200"
}
```

- 响应示例

```
 HTTP/1.1 200 OK
x-bce-request-id: 1a5fd81e-626b-45b3-a885-15fff9cd106c
Date: Tue, 17 May 2016 08:36:52 GMT
Content-Length: 0
Server: BceBos
```

- 返回代码

结果	http status	error code	说明
成功	200	-	-
Json格式错误	400	MalformedJSON	-
Json内容不正确	400	InappropriateJSON	比如index和404文件都为空
Index或者404文件名过长	400	EntityTooLarge	index或者404文件名都不能超过1024个字节
不允许的index或者404文件名	400	InvalidStaticWebSiteFormat	比如文件格式不允许，或者index与404文件同名。
静态网站托管功能被禁止	501	StaticWebSiteIsDisable	即BOS不允许此region的bucket 开启静态网站托管功能
其他错误， 遵循bos之前的错误代码	-	参考错误码	-

GetBucketStaticWebsite

接口描述

本接口用于获取bucket的静态网站托管信息。

请求（Request）

- 请求头域

  无特殊参数

- 请求语法

```
 Get /?website HTTP/1.1
Host: <BucketName>.bj.bcebos.com
Date: <Date>
Authorization: <AuthorizationString>
Content-Length: 0
```

响应（Response）

- 响应元素

名称	描述
index	Index文件名称
notFound	404文件名称
notFoundHttpStatus	返回错误页面时的HTTP状态码

- 响应示例

1. 当配置存在：

```
 HTTP/1.1 200 OK
Date: Thu, 15 May 2017 00:17:23 GMT
Server: BceBos
x-bce-request-id: xxxx-xxxxx-xxxx
Connection: keep-alive
Content-Length: xxx

{
    "index": "index.html",
    "notFound": "404.html",
    "notFoundHttpStatus":"404"
}
```

2. 当未开启静态网站托管

   返回http status 为 404， 错误Code为 NoSuchBucketStaticWebSiteConfig

- 返回代码

结果	http status	error code	说明
成功	200	-	-
没有开启静态托管	404	NoSuchBucketStaticWebSiteConfig	-
静态网站托管功能被禁止	501	StaticWebSiteIsDisable	即BOS不允许此region的bucket 开启静态网站托管功能
其他错误， 遵循bos之前的错误代码	-	参考错误码	-

DeleteBucketStaticWebsite

接口描述

本接口用于删除bucket设置的静态网站托管信息，并关闭此bucket的静态网站托管。

用户必须对bucket具有 FULL_CONTROL 权限。

请求（Request）

- 请求头域

  无特殊参数

- 请求语法

```
DELETE /?website HTTP/1.1
Host: <BucketName>.bj.bcebos.com
Date: <Date>
Authorization: <AuthorizationString>
```

响应（Response）

- 响应元素

  无特殊元素

- 响应头域

  无特殊头域

示例

- 响应示例

```
HTTP/1.1 204 No Content
Date: Wed, 14 Sep 2016 05:37:16 GMT
x-bce-request-id: 1a5fd81e-626b-45b3-a885-15fff9cd106c
Connection: keep-alive
Server: BceBos
```

- 返回代码

结果	http status	error code	说明
成功	204	-	-
静态网站托管功能被禁止	501	StaticWebSiteIsDisable	即BOS不允许此region的bucket 开启静态网站托管功能
其他错误， 遵循bos之前的错误代码	-	参考错误码	-

数据加密

PutBucketEncryption

接口描述

本接口用于开启指定Bucket的加密开关。

请求（Request）

- 请求语法

```
PUT /?encryption HTTP/1.1
Host: <BucketName>.bj.bcebos.com
Date: <Date>
Authorization: <AuthorizationString>
Content-Length: <ContentLength>

{
 "encryptionAlgorithm":"AES256"
}
```

- 请求参数

  无

- 请求头域

名称	类型	描述	是否必须
encryptionAlgorithm	String	指定Bucket的服务器端加密类型，当前支持AES256和SM4加密。	是

响应（Response）

- 响应元素

  无

- 响应头域

  无

示例

- 请求示例1

```
PUT /?encryption HTTP/1.1
Host: bucket.bj.bcebos.com
Date: Wed, 06 Apr 2016 06:34:40 GMT
Authorization: AuthorizationString
Content-Length: 11434

{
 "encryptionAlgorithm":"AES256"
}
```

- 返回示例1

```
HTTP/1.1 200 OK
x-bce-request-id: 4db2b34d-654d-4d8a-b49b-3049ca786409
Date: Wed, 06 Apr 2016 06:34:40 GMT
Content-Length: 11434
Server: BceBos
```

- 请求示例2

```
PUT /?encryption HTTP/1.1
Host: bucket.bj.bcebos.com
Date: Thu, 16 Mar 2023 07:53:05 GMT
Authorization: AuthorizationString
Content-Length: 11431

{
 "encryptionAlgorithm":"SM4"
}
```

- 返回示例2

```
HTTP/1.1 200 OK
x-bce-request-id: 4db2b34d-654d-4d8a-b49b-3049ca786409
Date: Thu, 16 Mar 2023 07:53:06 GMT
Content-Length: 11431
Server: BceBos
```

GetBucketEncryption

接口描述

本接口用于判断Bucket服务端加密是否打开。

请求（Request）

- 请求语法

```
Get /?encryption HTTP/1.1
Host: <BucketName>.bj.bcebos.com
Date: <Date>
Authorization: <AuthorizationString>
Content-Length: 0
```

- 请求参数

  无

- 请求头域

  无

响应（Response）

- 响应元素

名称	类型	描述
encryptionAlgorithm	String	Bucket的服务器端加密类型，当前支持AES256和SM4加密算法。

- 响应头域

  无

示例

- 请求示例1

```
Get /?encryption HTTP/1.1
Host: <BcuketName>.bj.bcebos.com
Date: Wed, 06 Apr 2016 06:34:40 GMT
Authorization: Authorizationstring
Content-Length: 0
```

- 返回示例1

```
 HTTP/1.1 200 OK
x-bce-request-id: 4db2b34d-654d-4d8a-b49b-3049ca786409
Date: Wed, 06 Apr 2016 06:34:40 GMT
Content-Length: 11434
Server: BceBos
x-bce-server-side-encryption: AES256

{
 "encryptionAlgorithm":"AES256"
}
```

- 请求示例2

```
 Get /?encryption HTTP/1.1
Host: <BcuketName>.bj.bcebos.com
Date: Thu, 16 Mar 2023 07:21:10 GMT
Authorization: Authorizationstring
Content-Length: 0
```

- 返回示例2

```
 HTTP/1.1 200 OK
x-bce-request-id: 4db2b34d-654d-4d8a-b49b-3049ca786409
Date: Thu, 16 Mar 2023 07:21:11 GMT
Content-Length: 11431
Server: BceBos
x-bce-server-side-encryption: SM4

{
 "encryptionAlgorithm":"SM4"
}
```

DeleteBucketEncryption

接口描述

本接口用于关闭服务端加密功能。

请求（Request）

- 请求语法

```
 DELETE /?encryption HTTP/1.1
Host: <BucketName>.bj.bcebos.com
Date: <Date>
Authorization: <AuthorizationString>
```

- 请求参数

  无

- 请求头域

  无

响应（Response）

- 响应元素

  无

- 响应头域

  无

示例

- 请求示例

```
 DELETE /?encryption HTTP/1.1
Host: <BcuketName>.bj.bcebos.com
Date: Wed, 06 Apr 2016 06:34:40 GMT
Authorization: Authorizationstring
```

- 返回示例

```
 HTTP/1.1 204 No Content
x-bce-request-id: 1a5fd81e-626b-45b3-a885-15fff9cd106c
Date: Tue, 17 May 2016 08:36:52 GMT
Connection: close
Server: BceBos
```

跨域访问

PutBucketCors

接口描述

本接口用来在指定的Bucket上设定一个跨域资源共享（CORS）的规则，如果原规则存在则覆盖原规则。

权限说明

只有Bucket的所有者和被授予FULL_CONTROL权限的用户才能设置Bucket的CORS。没有权限时，返回403 Forbidden错误，错误码：AccessDenied。

请求

- 请求语法

```
 PUT /?cors HTTP/1.1
Host: <BucketName>.bj.bcebos.com
x-bce-date: date
Content-Length: content_length
Content-Type: application/json; charset=utf-8
Authorization: <AuthorizationString>

{
   Cors json file …
}
```

- 请求参数

无

- 请求头域

无特殊Header参数

- 请求元素

Cors json文件包含如下字段：

名称	描述	是否必需	父节点
corsConfiguration	Bucket的CORS规则容器。每个Bucket最多允许有100条规则。 如果有多条配置，执行顺序为从上到下。	是	无
allowedOrigins	存储允许的跨域请求的来源的容器。	是	corsConfiguration
allowedOrigin	指定允许的跨域请求的来源，允许使用最多一个*通配符。 如果指定为*则表示允许所有的来源的跨域请求。特别的还支持*作为后缀来表示某一类网站比如abc*，则表示支持abc开头的网站。 注意：allowedOrigin匹配时大小写敏感。 类型：字符串	是	allowedOrigins
allowedMethods	存储允许的跨域请求方法的容器。	是	corsConfiguration
allowedMethod	指定允许的跨域请求的方法，不支持通配符*，大小写敏感。 类型：枚举，取值有“GET,PUT,DELETE,POST,HEAD”。	是	allowedMethods
allowedHeaders	存储允许的allowedHeaders容器。控制在OPTIONS预取指令中Access-Control-Request-Headers头中指定的header是否允许。	否	corsConfiguration
allowedHeader	控制在OPTIONS预取指令中Access-Control-Request-Headers头中指定的header是否允许。 在Access-Control-Request-Headers中指定的每个header都必须在allowedHeader中有一条对应的项。每一个header允许使用最多一个*通配符，不区分大小写。 类型：字符串。	否	allowedHeaders
allowedExposeHeaders	存储允许用户从应用程序中访问的响应头的容器。	否	corsConfiguration
allowedExposeHeader	指定允许用户从应用程序中访问的响应头(例如一个Javascript的XMLHttpRequest对象）。不允许使用*通配符。 OPTIONS请求中将根据此定义设置Access-Control-Expose-Headers。 类型：字符串。	否	allowedExposeHeaders
maxAgeSeconds	指定浏览器对特定资源的预取（OPTIONS）请求返回结果的缓存时间，在缓存时间内请求方不必发送重复的preflight 请求，单位为秒。 类型：Int64。	否	corsConfiguration

响应

- 响应头域

无

- 响应元素

无

> **注意事项**
>
> - 通过此接口设置CORS规则之前，Bucket的CORS权限设置为不允许跨域。
> - 每个Bucket最多只允许有一个规则文件，因此新上传的规则文件会覆盖原有的。
> - CORS规则文件大小限制为20KB，因此请求时大于20KB会返回超出大小错误（400 Bad Request: EntityTooLarge）。

示例

- 请求示例

```
 PUT /?cors HTTP/1.1
Host: BucketName.bj.bcebos.com
x-bce-date: 2016-04-06T08:23:49Z
Content-Length: 1024
Content-Type: application/json; charset=utf-8
Authorization: AuthorizationString

{
   "corsConfiguration": [
        {
        "allowedOrigins": [
           "http://www.example.com",
           "www.example2.com"
        ],
        "allowedMethods": [
              "GET",
              "HEAD",
              "DELETE"
           ],
         "allowedHeaders": [
              "Authorization",
              "x-bce-test",
              "x-bce-test2"
           ],
        "allowedExposeHeaders": [
            "user-custom-expose-header"
        ],
        "maxAgeSeconds": 3600
      } ,
      {
        "allowedOrigins": [
           "http://www.baidu.com"
        ],
        "allowedMethods": [
              "GET",
              "HEAD",
              "DELETE"
           ],
        "allowedHeaders": [
              "*",
        ],
        "allowedExposeHeaders": [
            "user-custom-expose-header"
        ],
        "maxAgeSeconds": 1800
      }
   ]
}
```

- 响应示例

```
 HTTP/1.1 200 OK
x-bce-request-id: 4db2b34d-654d-4d8a-b49b-3049ca786409
Content-Length: 0
Date: Wed, 06 Apr 2016 06:34:40 GMT
Server: BceBos
```

GetBucketCors

接口描述

本接口用于获取指定的Bucket当前的CORS规则。

权限说明

只有Bucket的所有者和被授予FULL_CONTROL权限的用户才能设置Bucket的CORS。没有权限时，返回403 Forbidden错误，错误码： AccessDenied。

请求

- 请求语法

```
 GET /?cors HTTP/1.1
Host: <BucketName>.bj.bcebos.com
Date: date
Authorization: <AuthorizationString>
```

- 请求参数

  无

- 请求头域

  无特殊Header参数

- 请求元素

  无

响应

- 响应头域

  无

- 响应元素

名称	描述	是否必需	父节点
corsConfiguration	Bucket的CORS规则容器。每个Bucket最多允许有100条规则。如果有多条配置，执行顺序为从上到下。	是	无
allowedOrigins	存储允许的跨域请求的来源的容器。	是	corsConfiguration
allowedOrigin	指定允许的跨域请求的来源，允许使用最多一个`*`通配符。 如果指定为`*`则表示允许所有的来源的跨域请求。特别的还支持`*`作为后缀来表示某一类网站比如abc`*`，则表示支持abc开头的网站。注意：allowedOrigin匹配时大小写敏感。类型：字符串	是	allowedOrigins
allowedMethods	存储允许的跨域请求方法的容器。	是	corsConfiguration
allowedMethod	指定允许的跨域请求的方法，不支持通配符`*`，大小写敏感。类型：枚举，取值有“GET,PUT,DELETE,POST,HEAD”。	是	allowedMethods
allowedHeaders	存储允许的allowedHeaders容器。控制在OPTIONS预取指令中Access-Control-Request-Headers头中指定的header是否允许。	否	corsConfiguration
allowedHeader	控制在OPTIONS预取指令中Access-Control-Request-Headers头中指定的header是否允许。在Access-Control-Request-Headers中指定的每个header都必须在allowedHeader中有一条对应的项。每一个header允许使用最多一个`*`通配符，不区分大小写。类型：字符串。	否	allowedHeaders
allowedExposeHeaders	存储允许用户从应用程序中访问的响应头的容器。	否	corsConfiguration
allowedExposeHeader	指定允许用户从应用程序中访问的响应头(例如一个Javascript的XMLHttpRequest对象）。不允许使用`*`通配符。OPTIONS请求中将根据此定义设置Access-Control-Expose-Headers。类型：字符串。	否	allowedExposeHeaders
maxAgeSeconds	指定浏览器对特定资源的预取（OPTIONS）请求返回结果的缓存时间，在缓存时间内请求方不必发送重复的preflight请求，单位为秒。类型：Int64。	否	corsConfiguration

注意事项

- 当Bucket不存在时，会返回404 Not Found错误，错误码：NoSuchBucket。
- 如果CORS规则并不存在，会返回404 Not Found错误，错误码：NoSuchCORSConfiguration。

示例

- 请求示例

```
 GET /?cors HTTP/1.1
Host: <BucketName>.bj.bcebos.com
Date: Wed, 06 Apr 2016 06:34:40 GMT
Authorization: AuthorizationString
```

- 响应示例

```
 HTTP/1.1 200 OK
x-bce-request-id: 4db2b34d-654d-4d8a-b49b-3049ca786409
Date: Wed, 06 Apr 2016 06:34:40 GMT
Content-Length: 1324
Content-Type: application/json; charset=utf-8
Server: BceBos

{
   "corsConfiguration": [
        {
        "allowedOrigins": [
           "http://www.example.com",
           "www.example2.com"
        ],
        "allowedMethods": [
              "GET",
              "HEAD",
              "DELETE"
           ],
         "allowedHeaders": [
              "Authorization",
              "x-bce-test",
              "x-bce-test2"
           ],
        "allowedExposeHeaders": [
            "user-custom-expose-header"
        ],
        "maxAgeSeconds": 3600
      } ,
      {
        "allowedOrigins": [
           "http://www.baidu.com"
        ],
        "allowedMethods": [
              "GET",
              "HEAD",
              "DELETE"
           ],
        "allowedHeaders": [
              "*"
        ],

        "allowedExposeHeaders": [
            "user-custom-expose-header"
        ],
        "maxAgeSeconds": 1800
      }
   ]
}
```

DeleteBucketCors

接口描述

DeleteBucketCors用于关闭指定Bucket的CORS功能并清空所有规则。

权限说明

只有Bucket的所有者和被授予FULL_CONTROL权限的用户才有权限删除CORS规则。

请求

- 请求语法

```
 DELETE /?cors HTTP/1.1
Host: <BucketName>.bj.bcebos.com
Date: Wed, 06 Apr 2016 06:34:40 GMT
Authorization: <AuthorizationString>
```

- 请求参数

  无

- 请求头域

  无特殊Header参数

- 请求元素

  无

响应

- 响应头域

  无

- 响应元素

  无

> **注意事项**
>
> - 当所对应的Bucket不存在时，会返回错误404 Not Found错误，错误码：NoSuchBucket。
> - 当没有权限Delete Cors时，会返回错误403 Forbidden，错误码：AccessDenied。
> - 当所对应的Bucket原本并不存在CORS规则时，会返回204 No Content。

示例

- 请求示例

```
DELETE /?cors HTTP/1.1
Host: BucketName.bj.bcebos.com
Date: Wed, 06 Apr 2016 06:34:40 GMT
Authorization: AuthorizationString
```

- 响应示例

```
HTTP/1.1 204 OK
x-bce-request-id: 4db2b34d-654d-4d8a-b49b-3049ca786409
Date: Wed, 06 Apr 2016 06:34:40 GMT
Connection: close
Server: BceBos
```

回收站

PutBucketTrash

接口描述

本接口用于开通Bucket Trash功能，用户必须是源Bucket的owner且拥有FULL_CONTROL权限，且是目标Bucket的owner。

请求

- 请求语法

```
PUT /?trash HTTP/1.1
Host: <BucketName>.bj.bcebos.com
Date: <Date>
Authorization: <AuthorizationString>
Content-Type: application/json
Content-Length: <ContentLength>
```

- 请求头域

无特殊Header参数

- 请求元素

名称	描述
trashDir	回收站的目录名称，有长度限制。可选，不填默认名为.trash

> **说明**：
>
> 1.若已开通trash功能，会覆盖原目录名。
>
> 2."trashDir"只能包含字母，数字，中文，下划线（_）和短横线（-），小数点（.）。包含'/'会报错。
>
> 3.开通了trash功能的bucket， 删除object会到回收站中(回收站中的object的全称是：trashDir + '/' + old object name)。未开通此功能的bucket，或者开通此功能的bucket且删除回收站内的object时，会彻底删除。
>
> 4.归档类型文件不支持Bucket Trash，删除归档类型文件时会直接删除。

响应

- 响应头域

无

- 响应元素

结果	http status	error code	说明
Json格式错误	400	MalformedJSON	-
trash目录名过长	400	EntityTooLarge	不超过1024字节
trash目录包含非法字符	400	InvalidTrashDirectoryName	只能包含字母，数字，中文，下划线（_）和短横线（-），小数点（.）
bucket不存在	404	NoSuchBucket	-
没有权限	403	AccessDenied	只有bucket owner且具有FULL_CONTROL可以操作

示例

- 请求示例

```
PUT /?trash HTTP/1.1
Host: bucket.bj.bcebos.com
x-bce-date: date
Content-Length: request-body-length
Content-Type: application/json; charset=utf-8
Authorization: AuthorizationString
{
"trashDir": "trashDirName"
}
```

- 响应示例

```
HTTP/1.1 200 OK
x-bce-request-id: 4db4b34d-653d-4d9a-b49b-3049ca786409
Date: Wed, 31 May 2017 08:34:40 GMT
Server: BceBos
```

GetBucketTrash

接口描述

本接口用于获取Bucket Trash开通状态，返回当前trash目录名，默认为.trash。用户必须是源Bucket的owner且拥有FULL_CONTROL权限，且是目标Bucket的owner。

请求

- 请求语法

  GET /?trash HTTP/1.1

  Host: .bj.bcebos.com

  Date: Date

  Authorization: AuthorizationString

- 请求头域

  无特殊Header参数

- 请求元素

  无

响应

- 响应头域

  无

- 响应元素

名称	描述
trashDir	回收站的路径名称，有长度限制。可选，不填默认名为.trash

返回结果解释如下：

结果	http status	error code	说明
成功	200	-	-
bucket不存在	404	NoSuchBucket	-
trash功能未开通	404	NoSuchTrashDirectory	-
没有权限	403	AccessDenied	只有bucket owner且具有FULL_CONTROL可以操作

示例

- 请求示例

```
GET /?trash HTTP/1.1
Host: bucket.bj.bcebos.com
x-bce-date: date
Authorization: AuthorizationString
```

- 响应示例

```
HTTP/1.1 200 OK
x-bce-request-id: 4db4b34d-653d-4d9a-b49b-3049ca786409
Date: Wed, 31 May 2017 08:34:40 GMT
Content-Length: request-body-length
Content-Type: application/json; charset=utf-8
Server: BceBos
Connection: keep-alive
{
"trashDir": ".trash/"
}
```

DeleteBucketTrash

接口描述

本接口用于关闭trash功能，用户必须是源Bucket的owner且拥有FULL_CONTROL权限，且是目标Bucket的owner。

请求

- 请求语法

```
DELETE /?trash HTTP/1.1
Host: <BucketName>.bj.bcebos.com
Date: <Date>
Authorization: <AuthorizationString>
```

- 请求头域

  无特殊Header参数

- 请求元素

  无

响应

- 响应头域

  无

- 响应元素

名称	描述
trashDir	回收站的文件夹，有长度限制。可选，不填默认名为.trash

返回结果解释如下：

结果	http status	error code	说明
成功	204	-	-
bucket不存在	404	NoSuchBucket	-
没有权限	403	AccessDenied	只有bucket owner可以操作

> 说明：即使未开通trash功能也返回204，满足delete幂等

示例

- 请求示例

```
DELETE /?trash HTTP/1.1
Host: bucket.bj.bcebos.com
x-bce-date: date
Authorization: AuthorizationString
```

- 响应示例

```
HTTP/1.1 204 OK
x-bce-request-id: 4db4b34d-653d-4d9a-b49b-3049ca786409
Date: Wed, 31 May 2017 08:34:40 GMT
Server: BceBos
Connection: keep-alive
```

数据同步

PutBucketReplication

接口描述

本接口用来创建数据同步。每个id唯一确定一条replication规则，对一个id首次put会认为是创建一条replication规则，对同一个id的再次put认为是覆盖，如果原先status是enable，则不允许覆盖成disable；如果原先是enable，再次put成enable，replication会重新开始执行；不允许两个replication规则除了id一致以外其他项均一致。

> **说明：**
>
> - 用户必须是源Bucket的owner且拥有FULL_CONTROL权限，且有目标Bucket的写权限。
> - 目标Bucket和源Bucket必须存在。
> - 目标Bucket和源Bucket可以是同region下的Bucket，也可以是不同Region下的Bucket。
> - 目标Bucket和源Bucket可以是同账户下的bucket，也可以是不同账户下的bucket，但是源账户需要有目的bucket的写权限
> - 整个配置的大小不能超过128k，当前bucket下所有规则长度不得超过200KB
> - 数据同步暂时不支持归档类型文件的同步，进行数据同步时会忽略归档类型文件。
> - 用户最多配置10条replication规则
> - 单个规则，最多20个resource
> - 单个规则，最多20个notIncludeResource
> - id 由数字字母 - _ 组成，不得超过20个字符

请求（Request）

- 请求语法

```
 PUT /?replication&id=sample-bucket HTTP/1.1
Host: <BucketName>.bj.bcebos.com
x-bce-date: date
Content-Length: request-body length
Content-Type: application/json; charset=utf-8
Authorization:<AuthorizationString>

{
   "status": "enabled",
   "resource": [
       "src-bucket-name/abc",
       "src-bucket-name/cd*"
   ],
   "notIncludeResource": [
       src-bucket-name/prefix/*,
       src-bucket-name/*.jpeg,
       src-bucket-name/*test*
   ],
   "destination": {
       "bucket": "dst-bucket-name",
       "storageClass": "COLD"
   },
   "replicateHistory": {
       "storageClass": "COLD"
   },
   "replicateDeletes": "enabled",
   "id": "sample-bucket"
}
```

- 请求头域

  无特殊参数

- 请求参数

参数	是否必须	描述
id	是	replication规则名，id 由数字字母 - _ 组成，不得超过20个字符
status	是	是否生效
resource	是	replication生效前缀，resource的配置形式为｛$bucket_name/<生效的对象前缀>｝，必须要以$bucket_name+/开头
notIncludeResource	否	replication不生效范围，notIncludeResource的配置形式为｛$bucket_name/<不生效的对象范围>｝，必须要以$bucket_name+/开头, 只接受以下三种形式，${bucketname}/<不生效的对象前缀>+*，${bucketname}/*+<不生效的对象后缀>，${bucketname}/*+不生效的对象中缀+*，参见请求示例
destination	是	复制的目的端配置
+bucket	是	目的Bucket name。同步到不同区域的bucket，指定bucket名称即可自动获取区域信息
+storageClass	否	目的Object的存储类型。如果保持和源Bucket的存储类型一致，则该参数不需要配置；如果需要单独指定存储类型可以为STANDARD，STANDARD_IA，COLD；如果是多AZ类型bucket，可取值为MAZ_STANDARD_IA和MAZ_STANDARD。
+prefix	否	目的存储路径
replicateHistory	否	历史文件复制，有该项则认为是开启。开启历史文件复制后，存量的全部Object都同步复制到目的Bucket，复制范围共用resource。
+storageClass	否	目的Object的存储类型。如果保持和源Bucket的存储类型一致，则该参数不需要配置；如果需要单独指定存储类型可以为STANDARD，STANDARD_IA，COLD；如果是多AZ类型bucket，可取值为MAZ_STANDARD_IA和MAZ_STANDARD。
replicateDeletes	是	是否开启删除同步,可以为enabled,disabled 。

响应（Response）

- 响应元素

  无特殊参数

- 响应头域

  无特殊参数

示例

- 请求示例

```
 PUT /?replication&id=sample-bucket HTTP/1.1
Host: BucketName.bj.bcebos.com
x-bce-date: date
Content-Length: xxx
Content-Type: application/json; charset=utf-8
Authorization: AuthorizationString

 {
    "status": "enabled",
    "resource": [
       "src-bucket-name/abc",
       "src-bucket-name/cd*"
    ],
    "notIncludeResource": [
       src-bucket-name/prefix/*,
       src-bucket-name/*.jpeg,
       src-bucket-name/*test*
    ],
    "destination": {
       "bucket": "dst-bucket-name",
       "storageClass": "COLD",
       "prefix": "dst-file-name/"

    },
    "replicateHistory": {
       "storageClass": "COLD"
    },
    "replicateDeletes": "enabled",
    "id": "sample-bucket-replication-config"
 }
```

- 响应示例

```
 HTTP/1.1 200 OK
x-bce-request-id: xxxx-xxxx-xxxx
Date: Wed, 12 May 2017 17:50:00 GMT
Content-Length: 0
Connection: keep-alive
Server: BceBos
```

GetBucketReplication

接口描述

本接口用来获取bucket指定id的数据同步信息，包括源Bucket名称、目的Bucket名称、存储类型、是否进行历史复制，数据同步策略，目的region等。

请求（Request）

- 请求语法

```
 GET /?replication&id=sample-bucket HTTP/1.1
Host: <BucketName>.bj.bcebos.com
Date: <GMT Date>
Authorization: AuthorizationString
```

- 请求头域

  无特殊参数

- 请求参数

  无特殊参数

响应（Response）

- 响应元素

参数	描述
id	replication规则名
status	是否生效
resource	replication生效前缀
destination	复制的目的端配置
+bucket	目的Bucket name
+storageClass	目的Object的存储类型。如果保持和源Bucket的存储类型一致，则该参数为空；STANDARD表示标准存储类型，STANDARD_IA表示低频存储类型，COLD表示冷存储类型；如果是多AZ类型bucket，可取值为MAZ_STANDARD_IA和MAZ_STANDARD。
replicateHistory	历史文件复制，有该项则认为是开启
+storageClass	目的Object的存储类型，默认为REMAIN,可以为STANDARD,STANDARD_IA,COLD,MAZ_STANDARD_IA,MAZ_STANDARD
replicateDeletes	是否开启删除同步,可以为enabled,disabled
create_time	replication规则创建时间
destRegion	目的region

- 响应头域

  无特殊参数

示例

- 请求示例

```
GET /?replication&id=sample-bucket HTTP/1.1
Host: BucketName.bj.bcebos.com
Date: Thu, 15 May 2017 00:17:23 GMT
Authorization: AuthorizationString
```

- 响应示例

```
HTTP/1.1 200 OK
Date: Thu, 15 May 2017 00:17:23 GMT
Server: BceBos
x-bce-request-id: xxxx-xxxxx-xxxx
Connection: keep-alive
Content-Length: xxx

{
  "status": "enabled",
  "resource": [
     "src-bucket-name/abc",
     "src-bucket-name/cd*"
  ],
  "destination": {
     "bucket": "dst-bucket-name",
     "storageClass": "COLD"
  },
  "replicateHistory": {
     "storageClass": "COLD"
  },
  "replicateDeletes": "enabled",
  "id": "sample-bucket",
  "create_time": 1583060606 ,
  "destRegion":"bj"
}
```

DeleteBucketReplication

接口描述

本接口用来删除对应id的数据同步复制配置。

请求（Request）

- 请求语法

```
DELETE /?replication&id=sample-bucket HTTP/1.1
Host: <BucketName>.bj.bcebos.com
Date: GMT Date
Authorization: AuthorizationString
```

- 请求头域

  无特殊参数

- 请求参数

  无特殊参数

响应（Response）

- 响应元素

  无特殊参数

- 响应头域

  无特殊参数

示例

- 请求示例

```
DELETE /?replication&id=sample-bucket HTTP/1.1
Host: bucket.bj.bcebos.com
Date: Thu, 15 May 2017 00:17:23 GMT
Authorization: AuthorizationString
```

- 响应示例

```
HTTP/1.1 204 No Content
Date: Thu, 15 May 2017 00:17:23 GMT
x-bce-request-id: 1a5fd81e-626b-45b3-a885-15fff9cd106c
Connection: keep-alive
Server: BceBos
```

GetBucketReplicationProgress

接口描述

本接口用来获取指定id的数据同步复制的进程状态。

请求（Request）

- 请求语法

```
Get /?replicationProgress&id=sample-bucket HTTP/1.1
Host: <BucketName>.bj.bcebos.com
Date: GMT Date
Authorization: AuthorizationString
```

- 请求头域

  无特殊参数

- 请求参数

  无特殊参数

响应（Response）

- 响应元素

参数	描述
status	是否开启了数据同步功能，可选值为enabled和disabled
historyReplicationPercent	历史文件数据同步已完成百分比，不再有效返回0
latestReplicationTime	UNIX 时间戳，最新的数据复制的时间

- 响应头域

  无特殊参数

示例

- 请求示例

```
Get /?replicationProgress&id=sample-bucket HTTP/1.1
Host: bucket.bj.bcebos.com
Date: Thu, 15 May 2017 00:17:23 GMT
Authorization: AuthorizationString
```

- 响应示例

```
HTTP/1.1 200 OK
Date: Thu, 15 May 2017 00:17:23 GMT
Server: BceBos
x-bce-request-id: xxxx-xxxxx-xxxx
Connection: keep-alive
Content-Length: xxx

{
  "status":"enabled",
  "historyReplicationPercent":5,
  "latestReplicationTime":1504448315
}
```

ListBucketReplication

接口描述

本接口用来获取bucket所有的replication同步规则

请求（Request）

- 请求语法

```
GET /?replication&list HTTP/1.1
Host: <BucketName>.bj.bcebos.com
Date: <GMT Date>
Authorization: AuthorizationString
```

- 请求头域

  无特殊参数

- 请求参数

  无特殊参数

响应（Response）

- 响应元素

参数	描述
rules	replication规则列表
+id	replication规则名
+status	是否生效
+resource	replication生效前缀
+destination	复制的目的端配置
++bucket	目的Bucket name
++storageClass	目的Object的存储类型。如果保持和源Bucket的存储类型一致，则该参数为空；STANDARD表示标准存储类型，STANDARD_IA表示低频存储类型，COLD表示冷存储类型；如果是多AZ类型bucket，MAZ_STANDARD_IA代表多AZ低频存储，MAZ_STANDARD表示多AZ标准存储类型，不能是其它取值。
+replicateHistory	历史文件复制，有该项则认为是开启
++storageClass	目的Object的存储类型，默认为REMAIN,可以为STANDARD,STANDARD_IA,COLD,MAZ_STANDARD,MAZ_STANDARD_IA
+replicateDeletes	是否开启删除同步,可以为enabled,disabled
+create_time	replication规则创建时间
+destRegion	目的region

- 响应头域

  无特殊参数

示例

- 请求示例

```
 GET /?replication&list HTTP/1.1
Host: BucketName.bj.bcebos.com
Date: Thu, 15 May 2017 00:17:23 GMT
Authorization: AuthorizationString
```

- 响应示例

```
 HTTP/1.1 200 OK
Date: Thu, 15 May 2017 00:17:23 GMT
Server: BceBos
x-bce-request-id: xxxx-xxxxx-xxxx
Connection: keep-alive
Content-Length: xxx

{
  "rules": [
    {
      "status": "enabled",
      "resource": [
        "src-bucket-name/abc",
        "src-bucket-name/cd*"
      ],
      "destination": {
        "bucket": "dst-bucket-name",
        "storageClass": "COLD"
      },
      "replicateHistory": {
        "storageClass": "COLD"
      },
      "replicateDeletes": "enabled",
      "id": "sample-bucket-replication-config",
      "create_time: 1583060606,
      "destRegion":"bj"
    }
  ]
}
```

合规保留

InitBucketObjectLock

接口描述

为指定Bucket创建基于时间的合规保留策略，此时策略状态变成IN_PROGRESS状态。

> **注意：**
>
> - 该状态的有效期为24小时，在有效期24小时内，此策略对应的Bucket资源处于保护状态，其保护规则如下所示： 假设您在2013年6月1日创建一个名为examplebucket的Bucket，并且在不同时间上传了file1.txt、file2.txt、file3.txt三个Object。随后在2014年7月1日创建了保护周期为5年的bucket锁策略。有关这三个Object的具体上传时间以及对应的Object到期时间如下：
>
> | Object名称 | 上传时间 | Object到期时间 |
> | --- | --- | --- |
> | file1.txt | 2013年6月1日 | 2018年5月31日 |
> | file2.txt | 2014年7月1日 | 2019年6月30日 |
> | file3.txt | 2018年9月30日 | 2023年9月29日 |
>
> - Bucket内的数据处于被保护状态时，任何人包括您尝试删除或修改这些数据，BOS API将返回409 FileImmutable的错误信息，请您谨慎配置该合规策略。

请求

- 请求语法

```
POST /?objectlock HTTP/1.1
Host: <BucketName>.bj.bcebos.com
Date: <Date>
Authorization: <AuthorizationString>
Content-Type: application/json; charset=utf-8
Content-Length: <ContentLength>

{
  "retentionDays" : 30
}
```

- 请求头域

  无特殊请求头域

- 请求参数

名称	类型	描述	是否必须
retentionDays	int	保护周期，单位为天	是

响应

- 响应头域

  无特殊参数

- 响应元素

  无特殊参数

示例

- 请求示例

```
POST /?objectlock HTTP/1.1
Host: <BucketName>.bj.bcebos.com
x-bce-date: 2016-04-06T06:34:40Z
Authorization: AuthorizationString

{
   "retentionDays" : 30
}
```

- 响应示例

```
HTTP/1.1 200 OK
x-bce-request-id: 4db2b34d-654d-4d8a-b49b-3049ca786409
Content-Length: 0
Date: Wed, 06 Apr 2016 06:34:40 GMT
Server: BceBos
```

GetBucketObjectLock

接口描述

获取合规保留策略配置信息。

请求

- 请求语法

```
GET /?objectlock HTTP/1.1
Host: <BucketName>.bj.bcebos.com
Date: <Date>
Authorization: <Authorization_String>
```

- 请求参数

  无特殊请求参数

- 请求头域

  无特殊请求头域

响应

- 响应头域

  无特殊参数

- 响应元素

参数	描述
lockStatus	合规保留策略状态
createDate	合规保留策略创建时间
expirationDate	合规保留策略过期时间，只有IN_PROGRESS状态下才可能会过期
retentionDays	合规保留保护周期，以天为单位

示例

- 请求示例

```
GET /?objectlock HTTP/1.1
Host: <BucketName>.bj.bcebos.com
x-bce-date: 2019-12-06T06:34:40Z
Authorization: AuthorizationString
```

- 响应示例

```
HTTP/1.1 200 OK
x-bce-request-id: 4db2b34d-654d-4d8a-b49b-3049ca786409
Content-Length: 0
Date: Wed, 06 Apr 2019 12:06:40 GMT
Server: BceBos

{
"lockStatus": "IN_PROGRESS",
"createDate": 1569317168,
"expirationDate": 1569403568,
"retentionDays": 3
}
```

DeleteBucketObjectLock

接口描述

当合规保留策略处于IN_PROGRESS和EXPIRED状态时，您可以通过该接口进行删除合规保留策略，若合规保留策略处于LOCKED锁定状态时，您将不能进行删除操作，除非删除该Bucket。

请求

- 请求语法

```
DELETE /?objectlock HTTP/1.1
Host: <BucketName>.bj.bcebos.com
Date: <Date>
Authorization: <Authorization_String>
```

- 请求参数

  无特殊请求参数

- 请求头域

  无特殊请求头域

响应

- 响应头域

  无特殊参数

- 响应元素

  无特殊参数

示例

- 请求示例

```
DELETE /?objectlock HTTP/1.1
Host: <BucketName>.bj.bcebos.com
x-bce-date: 2016-04-06T06:34:40Z
Authorization: AuthorizationString
```

- 响应示例

```
HTTP/1.1 200 OK
x-bce-request-id: 4db2b34d-654d-4d8a-b49b-3049ca786409
Content-Length: 0
Date: Wed, 06 Apr 2016 06:34:40 GMT
Server: BceBos
```

CompleteBucketObjectLock

接口描述

将合规保留策略立即锁定，变成LOCKED锁定状态，当合规保留策略处于LOCKED锁定时，任何人不可删除该策略，除非删除该Bucket，请您谨慎配置。

请求

- 请求语法

```
POST /?completeobjectlock HTTP/1.1
Host: <BucketName>.bj.bcebos.com
Date: <Date>
Authorization: <Authorization_String>
```

- 请求参数

  无特殊请求参数

- 请求头域

  无特殊请求头域

响应

- 响应头域

无特殊参数

- 响应元素

无特殊参数

示例

- 请求示例

```
POST /?completeobjectlock HTTP/1.1
Host: <BucketName>.bj.bcebos.com
x-bce-date: 2016-04-06T06:34:40Z
Authorization: AuthorizationString
```

- 响应示例

```
HTTP/1.1 200 OK
x-bce-request-id: 4db2b34d-654d-4d8a-b49b-3049ca786409
Content-Length: 0
Date: Wed, 06 Apr 2016 06:34:40 GMT
Server: BceBos
```

- 返回代码

结果	http status	error code	说明
成功	200	-	-
没有配置合规策略	404	NoSuchObjectLock	没有配置合规策略，无法锁定
合规策略已经锁定	409	ObjectLockAlreadyLocked	-
合规策略过期时不可锁定	409	ObjectLockAlreadyExpired	-

ExtendBucketObjectLock

接口描述

本接口用于延长合规保留策略保护周期。

请求

- 请求语法

```
POST /?extendobjectlock HTTP/1.1
Host: <BucketName>.bj.bcebos.com
Date: <Date>
Authorization: <AuthorizationString>
Content-Type: application/json; charset=utf-8
Content-Length: <ContentLength>

{
  "extendRetentionDays" : 34
}
```

- 请求头域

无特殊请求头域

- 请求参数

名称	类型	描述	是否必须
extendRetentionDays	int	新的保护周期，单位为天，不可缩短保护周期	是

响应

- 响应头域

无特殊参数

- 响应元素

无特殊参数

示例

- 请求示例

```
POST /?extendobjectlock HTTP/1.1
Host: <BucketName>.bj.bcebos.com
x-bce-date: 2016-04-06T06:34:40Z
Authorization: AuthorizationString

{
  "extendRetentionDays" : 34
}
```

- 响应示例

```
HTTP/1.1 200 OK
x-bce-request-id: 4db2b34d-654d-4d8a-b49b-3049ca786409
Content-Length: 0
Date: Wed, 06 Apr 2016 06:34:40 GMT
Server: BceBos
```

- 返回代码

结果	http code	error code	说明
成功	200	-	-
没有配置合规保留策略，无法锁定	404	NoSuchObjectLock	-
合规策略没有锁定，无法延长	400	ObjectLockNotLocked	-
合规保留策略锁定，不可缩短保护周期	400	NotAllowShortenRetentionday	-

原图保护

PutBucketCopyrightProtection

接口描述

本接口用来开启Bucket的原图保护功能，并指定resource字段，表示生效的资源范围。 对于开通原图保护的文件，不允许匿名下载访问该文件，或带自定义图片处理参数访问，只允许使用style样式访问或携带合法签名访问。

请求语法

```
 PUT /?copyrightProtection HTTP/1.1
Host: <BucketName>.bj.bcebos.com
x-bce-date: date
Content-Length: request-body length
Content-Type: application/json; charset=utf-8
Authorization: AuthorizationString

(request body: json string)
```

请求示例

```
 PUT /?copyrightProtection HTTP/1.1
Host: <BucketName>.bj.bcebos.com
x-bce-date: 2016-04-06T06:34:40Z
Content-Length: request-body length
Content-Type: application/json; charset=utf-8
Authorization: AuthorizationString

{
    "resource":[
            "bucket/prefix/*",
            "bucket/*/suffix"
    ]
}
```

响应示例

```
 HTTP/1.1 200 OK
x-bce-request-id: 4db2b34d-654d-4d8a-b49b-3049ca786409
Content-Length: 0
Date: Wed, 06 Apr 2016 06:34:40 GMT
Server: BceBos
```

> **说明**：用户必须拥有FULL_CONTROL权限。

GetCopyrightProtection

接口描述

此命令用来获取某个Bucket的原图保护配置情况。

请求语法

```
 GET /?copyrightProtection HTTP/1.1
Host: <BucketName>.bj.bcebos.com
x-bce-date: date
Authorization: <AuthorizationString>
```

请求示例

```
 GET /?copyrightProtection HTTP/1.1
Host: <BucketName>.bj.bcebos.com
x-bce-date: 2016-04-06T06:34:40Z
Authorization: AuthorizationString
```

响应示例

```
 HTTP/1.1 200 OK
Content-Length: 124
Date: Wed, 06 Apr 2016 06:34:40 GMT
Server: BceBos
x-bce-request-id: 4db2b34d-654d-4d8a-b49b-3049ca786410

{
    "resource": [
        "bucket/prefix/*",
        "bucket/*/suffix"
    ]
}
```

响应示例

```
HTTP/1.1 404
Date: Wed, 06 Apr 2016 06:34:40 GMT
Server: BceBos
x-bce-request-id: 4db2b34d-654d-4d8a-b49b-3049ca786410
```

> **说明**：请求者拥有FULL_CONTROL权限才允许查看，否则返回403错误，错误码为AccessDenied。

DeleteCopyrightProtection

接口描述

本接口用来关闭Bucket原图保护功能。

请求语法

```
DELETE /?copyrightProtection HTTP/1.1
Host: <BucketName>.bj.bcebos.com
x-bce-date: date
Authorization: AuthorizationString
```

****请求示例 ****

```
DELETE /?copyrightProtection HTTP/1.1
Host: <BucketName>.bj.bcebos.com
x-bce-date: 2016-04-06T06:34:40Z
Authorization: AuthorizationString
```

响应示例

```
HTTP/1.1 204 No Content
Date: Wed, 14 Sep 2016 05:37:16 GMT
x-bce-request-id: 1a5fd81e-626b-45b3-a885-15fff9cd106c
Server: BceBos
```

> **说明**：
>
> - 请求者只有拥有FULL_CONTROL权限才允许关闭原图保护功能，否则返回403错误，错误码为AccessDenied。
> - 如果请求的Bucket没有开通原图保护功能，依然返回HTTP状态码204。

存储桶清单

PutBucketInventory

接口描述

PutBucketInventory 用于为源bucket中创建清单任务，您可以对清单任务命名后，使用该请求创建清单任务。

Bos建议用户创建一个生命周期策略，用于删除旧的清单列表文件。

> **注意：**
>
> - 用户必须是源Bucket的owner且拥有FULL_CONTROL权限，且是目标Bucket的owner。
> - 目的Bucket和源Bucket必须存在，目的Bucket和源Bucket必须是同账号下的Bucket，且是同region下的Bucket，不支持跨账号输出。
> - BOS支持在每个存储桶中创建最多20条清单任务。
> - 清单列表提供了新对象和覆盖的 PUT 的最终一致性，并提供了 DELETE 的最终一致性。清单列表是Bucket的滚动快照，内容最终是一致的 ，清单可能不包含最近添加或删除的对象。

请求

- 请求语法

```
PUT /?inventory&id=inventory-configuration-ID HTTP/1.1
Host: <BucketName>.bj.bcebos.com
x-bce-date: date
Content-Length: request-body length
Content-Type: application/json; charset=utf-8
Authorization: AuthorizationString
{
   "id": "inventory-configuration-ID",
   "status": "enabled",
   "resource": [
      "bucket/prefix/*"
   ],
   "schedule": "Weekly",
   "destination": {
      "targetBucket": "destBucketName",
      "targetPrefix": "destination-prefix",
      "format": "CSV"
   }
}
```

- 请求头域

  无

- 请求参数

名称	类型	描述	是否必须
id	string	inventory规则名。规格：最长64字节。合法字符："a-z"、"A-Z"、"0-9"、"-"、"_"和"."。	是
status	string	清单是否生效的标识。如果设置为 “enabled”，清单功能将生效；如果设置为"disabled"，将不生成任何清单	是
resource	list	子元素为string类型。生效前缀，resource的配置形式为bucket_name/<生效的对象前缀>，必须要以bucket_name/开头.对整个bucket生效的时候，使用bucket_name/*。	是
schedule	string	清单任务周期，可选项为每三日、每周、每月，可选值：ThreeDaily、Weekly、Monthly。	是
monthlyDate	int	如果清单任务周期选择了按月，需要明确指定清单产出的日期。支持填入 1-28 范围内的整数。	否
destination	object	描述存放清单结果的信息。	是
+targetBucket	string	指定存放清单文件的Bucket。	是
+targetPrefix	string	指定最终被保存的清单文件前缀。	否
+format	string	清单分析结果的文件形式，当前支持格式：CSV。	是

响应

- 响应头域

  无特殊元素

- 响应元素

  无特殊头域

示例

- 请求示例

```
PUT /?inventory&id=inventory-configuration-ID HTTP/1.1
Host: <BucketName>.bj.bcebos.com
x-bce-date: date
Content-Length: request-body length
Content-Type: application/json; charset=utf-8
Authorization: AuthorizationString
{
    "id": "inventory-configuration-ID",
    "status": "enabled",
    "resource": [
        "bucket/prefix/*"
    ],
    "schedule": "Weekly",
    "destination": {
        "targetBucket": "destBucketName",
        "targetPrefix": "destination-prefix",
        "format": "CSV"
    }
}
```

- 响应示例

```
HTTP/1.1 200 OK
x-bce-request-id: 4db4b34d-653d-4d9a-b49b-3049ca786409
Date: Wed, 31 May 2018 08:34:40 GMT
Server: BceBos
```

GetBucketInventory

接口描述

GetBucketInventory 使用GET操作来获取指定桶的某个清单配置。用户在发起该请求时，需要用户提供清单任务的名称

请求

- 请求语法

```
GET /?inventory&id=inventory-configuration-ID HTTP/1.1
Host: <BucketName>.bj.bcebos.com
x-bce-date: date
Content-Length: 0
```

- 请求头域

  无特殊请求头域

- 请求query参数

名称	类型	描述	是否必须
id	string	inventory规则名	是

响应

- 响应头域

  无特殊响应头域

- 响应元素

名称	类型	描述	是否必须
id	string	inventory规则名。规格：最长64字节。合法字符："a-z"、"A-Z"、"0-9"、"-"、"_"和"."。	是
status	string	清单是否生效的标识。如果设置为 “enabled”，清单功能将生效；如果设置为"disabled"，将不生成任何清单	是
resource	list,子元素为string	生效前缀，resource的配置形式为`bucket_name/<生效的对象前缀>`，必须要以`bucket_name/`开头.对整个bucket生效的时候，使用`bucket_name/*`。	是
schedule	string	清单任务周期，可选项为每三日、每周、每月，可选值：ThreeDaily、Weekly、Monthly。	是
monthlyDate	int	如果清单任务周期选择了按月，指定清单产出的日期。支持1-28范围内的整数。	否
destination	object	描述存放清单结果的信息。	是
+targetBucket	string	指定存放清单文件的Bucket。	是
+targetPrefix	string	指定最终被保存的清单文件前缀。	否
+format	string	清单分析结果的文件形式，当前支持格式：CSV。	是

示例

- 请求示例

```
 GET /?inventory&id=inventory-configuration-ID HTTP/1.1
Host: <BucketName>.bj.bcebos.com
x-bce-date: date
Content-Length: 0
```

- 响应示例

```
 HTTP/1.1 200 OK
x-bce-request-id: 4db4b34d-653d-4d9a-b49b-3049ca786409
Date: Wed, 31 May 2018 08:34:40 GMT
Server: BceBos
Authorization: AuthorizationString
{
    "id": "inventory-configuration-ID",
    "status": "enabled",
    "resource": [
        "bucket/prefix/*"
    ],
    "schedule": "Weekly",
    "destination": {
        "targetBucket": "destBucketName",
        "targetPrefix": "destination-prefix",
        "format": "CSV"
    }
}
```

ListBucketInventory

接口描述

ListBucketInventory 使用不带清单id的GET操作来获取指定桶的所有清单配置，获取到的清单配置一次性返回，不分页。

请求

- 请求语法

```
 GET /?inventory HTTP/1.1
Host: <BucketName>.bj.bcebos.com
x-bce-date: date
Content-Length: 0
```

- 请求头域

  无请求头域

- 请求参数

  无特殊请求参数

响应

- 响应头域

  无特殊头域

- 响应元素

名称	类型	描述	是否必须
inventoryRuleList	list	存储桶中所有清单任务信息的列表	是
+id	string	inventory规则名。规格：最长64字节。合法字符："a-z"、"A-Z"、"0-9"、"-"、"_"和"."。	是
+status	string	清单是否生效的标识。如果设置为 “enabled”，清单功能将生效；如果设置为"disabled"，将不生成任何清单	是
+resource	list,子元素为string	生效前缀，resource的配置形式为bucket_name/<生效的对象前缀>，必须要以bucket_name/开头.对整个bucket生效的时候，使用bucket_name/*。	是
+schedule	string	清单任务周期，可选项为每三日、每周、每月，可选值：ThreeDaily、Weekly、Monthly。	是
+monthlyDate	int	如果清单任务周期选择了按月，指定清单产出的日期。支持1-28范围内的整数。	否
+destination	object	描述存放清单结果的信息。	是
++targetBucket	string	指定存放清单文件的Bucket。	是
++targetPrefix	string	指定最终被保存的清单文件前缀。	否
++format	string	清单分析结果的文件形式，当前支持格式：CSV。	是

示例

- 请求示例

```
GET /?inventory HTTP/1.1
Host:<BucketName>.bj.bcebos.com
x-bce-date: date
Content-Length: 0
```

- 响应示例

```
HTTP/1.1 200 OK
x-bce-request-id: 4db4b34d-653d-4d9a-b49b-3049ca786409
Date: Wed, 31 May 2018 08:34:40 GMT
Server: BceBos
Authorization: AuthorizationString
{
    "inventoryRuleList":[
        {
            "id": "inventory-configuration-ID1",
            "status": "enabled",
            "resource": [
                "bucket/prefix/*"
            ],
            "schedule": "Weekly",
            "destination": {
                "targetBucket": "destBucketName",
                "targetPrefix": "destination-prefix",
                "format": "CSV"
            }
        },
        {
            "id": "inventory-configuration-ID2",
            "status": "enabled",
            "resource": [
                "bucket/prefix2/*"
            ],
            "schedule": "Monthly",
            "monthlyDate": 15,
            "destination": {
                "targetBucket": "destBucketName",
                "targetPrefix": "destination-prefix-another",
                "format": "CSV"
            }
        }
    ]
}
```

DeleteBucketInventory

接口描述

DeleteBucketInventory使用DELETE操作来删除指定bucket的指定清单配置（通过清单id来指定清单配置）。

请求

- 请求语法

```
DELETE /?inventory&id=inventory-configuration-ID HTTP/1.1
Host: <BucketName>.bj.bcebos.com
x-bce-date: date
Content-Length: 0
```

- 请求头域

无特殊元素。

- 请求query参数

名称	类型	描述	是否必须
id	string	inventory规则名称。	是

响应

- 响应头域

  无特殊元素。

- 响应元素

无特殊头域

示例

- 请求示例

```
DELETE /?inventory&id=inventory-configuration-id HTTP/1.1
Host: <BucketName>.bj.bcebos.com
x-bce-date: date
Content-Length: 0
```

- 响应示例

```
HTTP/1.1 200 OK
x-bce-request-id: 4db4b34d-653d-4d9a-b49b-3049ca786409
Date: Wed, 31 May 2018 08:34:40 GMT
Server: BceBos
Authorization: AuthorizationString
```

请求者付费

PutBucketRequestPayment

接口描述

本接口用来创建请求者付费功能

> **注意：** 只有bucket的owner且拥有FULL_CONTROL权限才能够进行此请求。

请求

- 请求语法

```
PUT /?requestPayment HTTP/1.1
Host: <BucketName>.bj.bcebos.com
Date: <Date>
Authorization: <Authorization_String>
```

- 请求头域

  无特殊参数

- 请求参数

名称	类型	描述	是否必须
requestPayment	String	设置请求者付费规则。取值为Requester,表示请求者付费,取值为BucketOwner，表示关闭请求者计费	是

响应

- 响应头域

  无特殊元素

- 响应元素

  无特殊元素

示例

- 请求示例

```
PUT /?requestPayment HTTP/1.1
Host: <BucketName>.bj.bcebos.com
Date: <Date>
Authorization: <Authorization_String>

{
    "requestPayment": "Requester"
}
```

- 响应示例

```
HTTP/1.1 200 OK
x-bce-request-id: 0A49CE4060975EAC
Date: Wed, 12 Oct 2016 17:50:00 GMT
Content-Length: 0
Connection: keep-alive
Server: BceBos
```

GetBucketRequestPayment

接口描述

本接口获取Bucket请求者计费规则。

> **注意：** 只有bucket的owner且拥有FULL_CONTROL权限才能够进行此请求

请求

- 请求语法

```
 GET /?requestPayment HTTP/1.1
Host: <BucketName>.bj.bcebos.com
Date: <Date>
Authorization: <Authorization_String>
```

- 请求头域

无特殊参数

- 请求参数

无特殊参数

响应

- 响应头域

无特殊元素

- 响应元素

名称	类型	描述
requestPayment	string	请求者付费规则。取值为Requester,表示请求者付费,取值为BucketOwner，表示关闭请求者计费

示例

- 请求示例

```
 GET /?requestPayment HTTP/1.1
Host: <BucketName>.bj.bcebos.com
Date: <Date>
Authorization: <Authorization_String>
```

- 响应示例

```
 HTTP/1.1 200 OK
x-bce-request-id: 4db2b34d-654d-4d8a-b49b-3049ca786409
Date: Wed, 06 Apr 2016 06:34:40 GMT
Server: BceBos
{
    "requestPayment": "Requester"
}
```

额度管理

PutBucketQuota

接口描述

本接口用来为单个Bucket设置容量上限及object个数上限

> **注意：** 只有bucket的owner且拥有FULL_CONTROL权限才能够进行此请求。

请求

- 请求语法

```
 PUT /?quota HTTP/1.1
Host: <BucketName>.bj.bcebos.com
Date: <Date>
Authorization: <Authorization_String>
```

- 请求头域

无特殊参数

- 请求参数

名称	类型	描述	是否必须	备注
maxObjectCount	int64	单个Bucket最大个数限制值	是	若值为-1或0，表示无限制
maxCapacityMegaBytes	int64	单个Bucket最大容量限制值，单位为MB	是	若值为-1或0，表示无限制

响应

- 响应头域

  无特殊元素

- 响应元素

  无特殊元素

示例

- 请求示例

```
 PUT /?quota HTTP/1.1
Host: <BucketName>.bj.bcebos.com
Date: <Date>
Authorization: <Authorization_String>

{
  "maxObjectCount" : 50,
  "maxCapacityMegaBytes"  : 12334424
}
```

- 响应示例

```
  HTTP/1.1 200 OK
 x-bce-request-id: 0A49CE4060975EAC
 Date: Wed, 12 Oct 2016 17:50:00 GMT
 Content-Length: 0
 Connection: keep-alive
 Server: BceBos
```

GetBucketQuota

接口描述

本接口用来获取单个bucket的quota信息

> **注意：** 只有bucket的owner且拥有FULL_CONTROL权限才能够进行此请求。

请求

- 请求语法

```
 GET /?quota HTTP/1.1
Host: <BucketName>.bj.bcebos.com
Date: <Date>
Authorization: <Authorization_String>
```

- 请求头域

  无特殊参数

- 请求参数

无特殊参数

响应

- 响应头域

  无特殊元素

- 响应元素

  无特殊元素

示例

- 请求示例

```
 GET /?quota HTTP/1.1
Host: <BucketName>.bj.bcebos.com
Date: <Date>
Authorization: <Authorization_String>
```

- 响应示例

```
HTTP/1.1 200 OK
x-bce-request-id: 0A49CE4060975EAC
Date: Wed, 12 Oct 2016 17:50:00 GMT
Content-Length: 0
Connection: keep-alive
Server: BceBos
{
"maxObjectCount": 50,
"maxCapacityMegaBytes" : 12334424
}
```

DeleteBucketQuota

接口描述

本接口用来删除特定Bucket的quota配置

> **注意：** 只有bucket的owner且拥有FULL_CONTROL权限才能够进行此请求

请求

- 请求语法

```
DELETE /?quota HTTP/1.1
Host: <BucketName>.bj.bcebos.com
Date: <Date>
Authorization: <Authorization_String>
```

- 请求头域

无特殊参数

- 请求参数

无特殊参数

响应

- 响应头域

无特殊元素

- 响应元素

无特殊元素

示例

- 请求示例

```
DELETE /?quota HTTP/1.1
Host: <BucketName>.bj.bcebos.com
Date: <Date>
Authorization: <Authorization_String>
```

- 响应示例

```
HTTP/1.1 200 OK
x-bce-request-id: 4db2b34d-654d-4d8a-b49b-3049ca786409
Date: Wed, 06 Apr 2016 06:34:40 GMT
Server: BceBos
```

镜像回源

PutBucketMirroring

接口描述

本接口用来创建镜像回源管理规则。

> **注意:**
>
> - 只有bucket的owner且拥有FULL_CONTROL权限才能够进行此请求。
> - PutBucketMirroring会覆盖原有的镜像回源规则， 如果需要在原有规则基础上新增， 需要在请求中携带所有新旧规则。

请求（Request）

- 请求语法

```
 PUT /?mirroring HTTP/1.1
Host: <BucketName>.bj.bcebos.com
x-bce-date: date
Content-Length: request-body length
Content-Type: application/json; charset=utf-8
Authorization: AuthorizationString
{
  "bucketMirroringConfiguration":[
      {
          "mode":"fetch",
          "sourceUrl":"http://www.baidu.com",
          "backSourceUrl":"bos://bj.bcebos.com/bucket",
          "resource" : "folder1/folder2*.jpeg",
          "prefix": "testprefix",
          "suffix": ".jpeg",
          "fixedKey":"folder1/404.jpeg",
          "version":"v2",
          "prefixReplace" : "a/b/c",
          "passQuerystring":true,
          "storageClass":"STANDARD",
          "customHeaders":[
             {
                "headerName":"testheader1",
                "headerValue":"name1",
             },
             {
                "headerName":"TestHeaderName",
                "headerValue":"TestHeaderValue",
             }
          ],
          "ignoreHeaders": ["BanHeader1","BanHeader2"],
          "passHeaders":["AllowHeader1","AllowHeader2"], //若设置为["*"],则表示透传所有header
      },
  ]
}
```

- 请求头域

  无特殊参数

- 请求参数

规则项	描述	是否必须	备注	数据类型
bucketMirroringConfiguration	容器	是	镜像回源规则的容器	列表
mode	模式为回源或者重定向	是	可选值 fetch(回源)，redirect（重定向）	字符串
sourceUrl	主源站url	是	主源站回源地址，如果选择私有bucket，则传入以下信息 "bos://endpoint/bucket"， 否则传入用户手动输入的url	字符串
backSourceUrl	备源站url	否	规则与主源站url相同 |字符串|
resource	规则对哪些资源生效	是	回源条件中的前后缀，以*分割, "folder1/folder2"代表回源条件前缀，".jpeg"代表回源条件后缀 。	字符串
prefix	回源地址前缀	否	源站前缀 |字符串|
suffix	回源地址后缀	否	例如当配置为.docx时，回源object名称将被自动扩展为myobject.docx	字符串
fixedKey	回源地址固定名称	否	当fixedKey有传入且不为空时，则回源到固定名称，此时忽略源站前后缀prefix，suffix，前后缀替换等场景，回源地址为sourceUrl+fixedKey	字符串
version	回源前缀替换选项	否	version为空默认v1，代表老版本配置，老版本prefixReplace为""时，代表不对objectname中的路径前缀进行操作，新版本为v2，代表启用object前缀替换，此时传入""代表对路径前缀进行裁剪 |字符串|
prefixReplace	前缀替换裁剪选项	否	仅当version为v2时生效，传入非空字符串则代表进行前缀替换，将"path1/path2/objectname"替换为"a/b/c/objectname"，传入空字符串则代表进行前缀裁剪，"path1/path2/objectname"替换为"objectname" |字符串|
passQuerystring	是否传递请求参数	否	可选值true或false，默认false	bool
storageClass	回源的存储类型	否	fetch模式下回源存储类型，例如STANDARD	字符串
passHeaders	透传的header	否	请求源站时将透传的header,若设置为["*"]则表示透传全部header	列表
ignoreHeaders	忽略的header	否	请求源站时将忽略的header	列表
customHeaders	自定义header	否	请求源站时自定义携带的header	列表

响应（Response）

- 响应示例

```
 HTTP/1.1 200 OK
x-bce-request-id: 0A49CE4060975EAC
Date: Wed, 12 Oct 2016 17:50:00 GMT
Content-Length: 0
Connection: keep-alive
Server: BceBos
```

GetBucketMirroring

接口描述

本接口用来获取bucket的镜像回源信息

请求（Request）

- 请求语法

```
 GET /?mirroring HTTP/1.1
Host: <BucketName>.bj.bcebos.com
Date: <GMT Date>
Authorization: AuthorizationString
```

- 请求头域

  无特殊参数

- 请求参数

无特殊参数

响应（Response）

- 响应元素

参数	描述
mode	模式为拉取或者重定向
sourceUrl	主源站url
backSourceUrl	备源站url
resource	规则对哪些资源生效
prefix	回源地址前缀
suffix	回源地址后缀
fixedKey	回源地址固定名称
storageClass	Object的存储类型
version	回源前缀替换选项
prefixReplace	前缀替换裁剪选项
passQuerystring	是否传递请求参数
passHeaders	透传的header
ignoreHeaders	忽略的header
customHeaders	自定义header

- 响应头域

无特殊参数

示例

- 请求示例

```
GET /?mirroring HTTP/1.1
Host: test-bucket.bj.bcebos.com
Date: Thu, 15 May 2017 00:17:23 GMT
Authorization: AuthorizationString
```

- 响应示例

```
HTTP/1.1 200 OK
Date: Thu, 15 May 2017 00:17:23 GMT
Server: BceBos
x-bce-request-id: xxxx-xxxxx-xxxx
Connection: keep-alive
Content-Length: xxx
{
  "bucketMirroringConfiguration":[
    {
      "mode":"fetch",
      "sourceUrl":"http://www.baidu.com",
      "backSourceUrl":"bos://bj.bcebos.com/bucket",
      "resource" : "folder1/folder2*.jpeg",
      "prefix": "testprefix",
      "suffix": ".jpeg",
      "fixedKey":"folder1/404.jpeg",
      "version":"v2",
      "prefixReplace" : "a/b/c",
      "passQuerystring":true,
      "storageClass":"STANDARD",
      "allHeader":"custom",
      "customHeaders":[
          {
              "headerName":"testheader1",
              "headerValue":"name1",
          },
          {
              "headerName":"TestHeaderName",
              "headerValue":"TestHeaderValue",
          }
      ],
      "ignoreHeaders": ["BanHeader1","BanHeader2"],
      "passHeaders":["AllowHeader1","AllowHeader2"],
    },
  ]
}
```

DeleteBuketMirroring

接口描述

本接口用来删除bucket的镜像回源配置。

请求（Request）

- 请求语法

```
DELETE /?mirroring HTTP/1.1
Host: <BucketName>.bj.bcebos.com
Date: GMT Date
Authorization: AuthorizationString
```

- 请求头域

  无特殊参数

- 请求参数

  无特殊参数

响应（Response）

- 响应元素

  无特殊参数

- 响应头域

  无特殊参数

示例

- 请求示例

```
DELETE /?mirroring HTTP/1.1
Host: bucket.bj.bcebos.com
Date: Thu, 15 May 2017 00:17:23 GMT
Authorization: AuthorizationString
```

- 响应示例

```
HTTP/1.1 204 No Content
Date: Thu, 15 May 2017 00:17:23 GMT
x-bce-request-id: 1a5fd81e-626b-45b3-a885-15fff9cd106c
Connection: keep-alive
Server: BceBos
```

标签管理

PutBucketTagging

接口描述

本接口用来为单个Bucket设置tag

> **注意：** 只有bucket的owner且拥有FULL_CONTROL权限才能够进行此请求。

请求

- 请求语法

```
PUT /?tagging HTTP/1.1
Host: <BucketName>.bj.bcebos.com
Date: <Date>
Authorization: <Authorization_String>
Content-Length: request-body length
Content-Type: application/json; charset=utf-8
```

- 请求头域

  无特殊参数

- 请求参数

名称	类型	描述	是否必须	备注
tags	Tag	需要设置的tag列表	是	列表，最多支持200个tagKey-tagValue健值对

- Tag结构

名称	类型	是否必须
tagKey	String	是
tagValue	String	是

响应

- 响应头域

  无特殊元素

- 响应元素

  无特殊元素

示例

- 请求示例

```
 PUT /?tagging HTTP/1.1
Host: <BucketName>.bj.bcebos.com
x-bce-date: date
Content-Length: request-body length
Content-Type: application/json; charset=utf-8
Authorization: AuthorizationString
{
    "tags":[
        {
            "tagKey":"key1",
            "tagValue":"value123"
        },
        {
            "tagKey":"ttt2",
            "tagValue":"6863gerg"
        }
    ]
}
```

- 响应示例

```
 HTTP/1.1 200 OK
x-bce-request-id: 0A49CE4060975EAC
Date: Wed, 12 Oct 2016 17:50:00 GMT
Content-Length: 0
Connection: keep-alive
Server: BceBos
```

DeleteBucketTagging

接口描述

本接口用来删除特定Bucket的tag配置

> **注意：** 只有bucket的owner且拥有FULL_CONTROL权限才能够进行此请求

请求

- 请求语法

```
 DELETE /?tagging HTTP/1.1
Host: <BucketName>.bj.bcebos.com
Date: <Date>
Authorization: <Authorization_String>
```

- 请求头域

无特殊参数

- 请求参数

无特殊参数

响应

- 响应头域

无特殊元素

- 响应元素

无特殊元素

示例

- 请求示例

```
 DELETE /?tagging HTTP/1.1
Host: <BucketName>.bj.bcebos.com
Date: <Date>
Authorization: <Authorization_String>
```

- 响应示例

```
 HTTP/1.1 200 OK
x-bce-request-id: 4db2b34d-654d-4d8a-b49b-3049ca786409
Date: Wed, 06 Apr 2016 06:34:40 GMT
Server: BceBos
```

GetBucketTagging

接口描述

本接口用来获取单个bucket的tag信息

> **注意：** 只有bucket的owner且拥有FULL_CONTROL权限才能够进行此请求。

请求

- 请求语法

```
 GET /?tagging HTTP/1.1
Host: <BucketName>.bj.bcebos.com
Date: <Date>
Authorization: <Authorization_String>
```

- 请求头域

  无特殊参数

- 请求参数

无特殊参数

响应

- 响应头域

  无特殊元素

- 响应元素

  无特殊元素

示例

- 请求示例

```
 GET /?tagging HTTP/1.1
Host: <BucketName>.bj.bcebos.com
Date: <Date>
Authorization: <Authorization_String>
```

- 响应示例

```
  HTTP/1.1 200 OK
 x-bce-request-id: 0A49CE4060975EAC
 Date: Wed, 12 Oct 2016 17:50:00 GMT
 Connection: keep-alive
 Server: BceBos
 {
    "tag":[
       {
          "tagKey":"key1",
          "tagValue":"value123"
       },
       {
          "tagKey":"ttt2",
          "tagValue":"6863gerg"
       }
    ]
    "group":[]
 }
```

版本控制

PutBucketVersioning

接口描述

调用PutBucketVersioning设置指定存储空间（Bucket）的版本控制状态。

> 注意事项
>* 要配置版本控制，您必须有PutBucketVersioning权限。
>* Bucket包括未开启(notEnabled)、开启（enabled）或者暂停（suspended）三种版本控制状态。默认情况下Bucket处于未开启版本控制状态。
>* 在Bucket处于开启版本控制状态下，所有新添加的文件（Object）都将拥有唯一的版本ID，将累积所添加Object的多个版本。
>* 在Bucket处于暂停版本控制状态下，所有新添加Object的版本ID将为null，将不再为此状态下添加的Object累积更多的版本。

请求（Request）

- ### 请求语法

 ```json
PUT /?versioning HTTP/1.1
Host: <BucketName>.bj.bcebos.com
x-bce-date: <Date>
Content-Length: <ContentLength>
Content-Type:application/json; charset=utf-8
Authorization: <AuthorizationString>
{status:enabled}
```

响应（Response）

- 响应头域无特殊Header参数返回

- 响应元素无

示例

- 设置多版本状态的示例

```
PUT /?versionging HTTP/1.1
Host: BucketName.bj.bcebos.com
x-bce-date: 2024-04-06T08:23:49Z
Content-Length : 16
Content-Type: application/json; charset=utf-8
Authorization: AuthorizationString
{status:enabled}
```

- 响应示例

```
HTTPS/1.1 200 OK
x-bce-request-id: 4db2b34d-654d-4d8a-b49b-3049ca786409
Content-Length: 0
Date: Wed, 06 Apr 2024 06:34:40 GMT
Server: BceBos
```

GetBucketVersioning

接口描述 本接口用来获取某个Bucket的version状态。

请求（Request）

- 请求语法

```
GET /?versioning HTTP/1.1
Host: <BucketName>.bj.bcebos.com
x-bce-date: <Date>
Authorization: <AuthorizationString>
```

- 请求头域无特殊参数
- 请求参数无特殊参数

响应（Response）

- 响应元素（以JSON请求为例）

名称	类型	描述
status	String	bucket的多版本状态

- 响应头域

  无

示例

- 请求示例

```
GET /?versioning HTTP/1.1
Host: BucketName.bj.bcebos.com
x-bce-date: 2024-04-06T08:23:49Z
Authorization: AuthorizationString
```

- 响应示例

```
HTTP/1.1 200 OK
x-bce-request-id: 4db2b34d-654d-4d8a-b49b-3049ca786409
Date: Wed, 06 Apr 2024 06:34:40 GMT
Server: BceBos

{status: enabled}
```

Object相关接口

基础操作

ListObjects

接口描述

本接口用于获得指定Bucket的Object信息列表。

请求（Request）

- 请求语法

```
GET / HTTP/1.1
Host: <BucketName>.bj.bcebos.com
Date: <Date>
Authorization: <AuthorizationString>
```

- 请求头域

  无特殊Header参数

- 请求参数

字段	类型	是否必需	描述
delimiter	String	否	分隔符; 主要用此项实现list文件夹的逻辑。如果在请求的时候指定了delimiter，BOS把匹配到的Object名称按照一定规则（从preifx到第一个delimiter）截取，截取的字符串去重作为CommonPrefixes的数据返回; delimiter长度限制为1
marker	String	否	object为字母序排列，从marker之后的第一个开始返回
maxKeys	Int	否	返回object列表长度最大为1000，默认值是1000，如果指定的值大于1000，按1000操作
prefix	String	否	key前缀，限定返回的object key必须以此为前缀

响应（Response）

- 响应头域

  无特殊Header参数响应

- 响应元素

名称	类型	描述
commonPrefixes	Array	仅当指定delimiter，才会返回此项
+prefix	String	匹配以prefix开始到第一次出现Delimiter字符之间的object作为一组元素返回
contents	Array	返回的一个object的列表
+key	String	Object名称
+lastModified	Date	此Object最后一次被修改的时间
+eTag	String	Object的HTTP协议实体标签
+size	Int	Object的内容的大小（字节数）
+storageClass	String	Object的存储类型，低频存储返回`STANDARD_IA`，冷存储返回`COLD`，归档存储返回`ARCHIVE`，标准存储返回`STANDARD`。
+owner	Container	Object上传者的用户信息
++id	String	Object上传者的用户id
++displayName	String	Object上传者的名称
delimiter	String	查询的结束符
isTruncated	Bool	指明是否查询完整返回了；false-本次已经返回所有结果，true-本次还没有返回所有结果
maxKeys	Int	请求返回的最大数目
marker	String	本次查询的起点
name	String	Bucket名称
nextMarker	String	当IsTruncated true时，才返回此项，作为下次查询marker的值
prefix	String	查询的前缀

> **注意事项**
>
> Delimiter可以用来实现文件夹逻辑：
>
> 1. 如果把prefix设为某个文件夹名，就可以罗列以此prefix开头的文件，即该文件夹下递归的所有的文件和子文件夹内的文件。
> 2. 如果再把delimiter设置为 / 时，返回值就只罗列该文件夹下的文件，该文件夹下的子文件夹名返回在CommonPrefixes部分，子文件夹下递归的文件不被显示。 如一个bucket下存在三个object：fun/test.jpg，fun/movie/001.avi，fun/movie/007.avi。 若设定prefix为“fun/”，则返回三个object；如果增加设定delimiter为“/”，则返回文件“fun/test.jpg”和前缀“fun/movie/”。

示例

- 请求示例（JSON）1

```
GET / HTTP/1.1
Host: <BucketName>.bj.bcebos.com
Date: Wed, 06 Apr 2016 06:34:40 GMT
Authorization: <AuthorizationString>
```

- 响应示例（JSON）1，递归列举所有object

```
HTTP/1.1 200 OK
x-bce-request-id: 4db2b34d-654d-4d8a-b49b-3049ca786409
Date: Wed, 06 Apr 2016 06:34:40 GMT
Server: BceBos

{
    "name":"bucket",
    "prefix":"",
    "delimiter":"/",
    "marker":"",
    "maxKeys":1000,
    "isTruncated":false,
    "contents":[
        {
            "key":"my-image.jpg",
            "lastModified":"2009-10-12T17:50:30Z",
            "eTag":"fba9dede5f27731c9771645a39863328",
            "size":434234,
            "storageClass":"STANDARD",
            "owner":{
                "id":"168bf6fd8fa74d9789f35a283a1f15e2",
                "displayName":"mtd"
            }
        },
        {
            "key":"my-image1.jpg",
            "lastModified":"2009-10-12T17:51:30Z",
            "eTag":"0cce7caecc8309864f663d78d1293f98",
            "size":124231,
            "storageClass":"COLD",
            "owner":{
                "id":"168bf6fd8fa74d9789f35a283a1f15e2",
                "displayName":"mtd"
            }
        }
    ]
}
```

- 请求示例（JSON）2， 列举根目录下的部分文件和子目录，受maxKeys限制(不包括子目录下的文件)

```
GET /?delimiter=/ HTTP/1.1
Host: <BucketName>.bj.bcebos.com
Date: Wed, 06 Apr 2016 06:34:40 GMT
Authorization: <AuthorizationString>
```

- 响应示例（JSON）2

```
HTTP/1.1 200 OK
x-bce-request-id: 4db2b34d-654d-4d8a-b49b-3049ca786409
Date: Wed, 06 Apr 2016 06:34:40 GMT
Server: BceBos

{
    "name":"bucket",
    "prefix":"" ,
    "delimiter":"/",
    "marker":"",
    "maxKeys":1000,
    "isTruncated":false,
    "contents":[
        {
            "key":"my-image.jpg",
            "lastModified":"2009-10-12T17:50:30Z",
            "eTag":"fba9dede5f27731c9771645a39863328",
            "size":434234,
            "storageClass":"STANDARD",
            "owner":{
                "id":"168bf6fd8fa74d9789f35a283a1f15e2",
                "displayName":"mtd"
            }
        }
    ],
    "commonPrefixes":[
        {"prefix":"photos/"},
        {"prefix":"mtd/"}
    ]
}
```

PutObject

接口描述

此接口用于向指定的Bucket上传一个文件，请求者必须具有Write权限。在PutObject前需要确保对应的Bucket已经存在，BOS支持Object文件的长度范围是0Byte-5GB。如果需要上传大于5GB的文件，请参考分块上传指南。

请求（Request）

- 请求语法

```
PUT /<ObjectName> HTTP/1.1
Host: <BucketName>.bj.bcebos.com
Date: <Date>
Authorization: <AuthorizationString>
Content-Type: text/plain
Content-Length: <Content_Length>
x-bce-storage-class: <Storage_class>
```

- 请求参数

  无特殊参数

- 请求头域

名称	类型	描述	是否必需
Cache-Control	String	下载Object的Cache设置，常见的可取值为private、no-cache、max-age、must-revalidate	否
Content-Disposition	String	设置浏览器是否下载，可取值为inline、attachment; filename="download.txt"	否
Content-MD5	String	RFC2616定义的HTTP请求内容的MD5摘要，可以通过携带该字段来验证保存在BOS侧的文件和用户预期的文件是否一致。	否
Expires	String	用于设置下载Object时的缓存失效时间，如果不做时间设置，BOS则会默认设置缓存失效时间为三天。	否
x-bce-meta-*	String	用户自定义的meta	否
x-bce-content-sha256	String	通过携带该字段来验证保存在BOS侧的文件和用户预期的文件是否一致，sha256的校验准确性更高。所传数据的sha256值必须与此匹配，否则PutObject失败	否
x-bce-content-crc32	String	上传object的CRC32值(IEEE算法)。	否
x-bce-content-crc32c	String	上传object的CRC32C值(Castagnoli算法)。	否
x-bce-content-crc32c-flag	String	是否计算CRC32C，仅取值为"true"时有效。	否
x-bce-storage-class	String	指定Object的存储类型，STANDARD_IA代表低频存储，COLD代表冷存储，ARCHIVE代表归档存储，不指定时默认是STANDARD标准存储类型；如果是多AZ类型bucket，MAZ_STANDARD_IA代表多AZ低频存储，不指定时默认是MAZ_STANDARD多AZ标准存储类型，不能是其它取值	否
x-bce-acl	String	CannedACL支持的header，用户设置Object的权限，取值为private和public-read。	否
x-bce-grant-read	String	CannedACL支持的header，用户设置Object的读权限。支持多个id，以逗号分隔	否
x-bce-grant-full-control	String	CannedACL支持的header，用户设置Object的FULL_CONTROL权限。支持多个id，以逗号分隔	否
x-bce-server-side-encryption	String	服务端加密算法，当前支持AES256和SM4加密。	否
x-bce-forbid-overwrite	Boolen	是否禁止被覆盖，默认为false，当该值为true时下次上传同名文件时将返回409错误	否

响应（Response）

- 响应头域

名称	类型	描述
ETag	String	Object的HTTP协议实体标签
x-bce-version-id	String	Object的版本Id

> **注意事项**
>
> 1. Content-Length是必须参数，如果请求者指定的Content-Length比实际请求体（Object的实际数据）长度小，BOS只保存Content-Length指定长度的数据，多的这部分数据直接废弃；相反，如果Content-Length的长度大，BOS将一直等待请求者上传数据，直到超时。
> 2. 上传的Object，如不指定Content-Type，BOS会自动识别设置合适的Content-Type，若无法识别则默认为application/octet-stream
> 3. 由于BOS本身是一个(<Key>,<Value>)的存储系统，所以原则上并不会存在"文件夹"的概念。若需要按照文件夹来划分，可以把 "/" 符号作为分隔符模拟文件夹。例如上传object为 "work/test/123.txt"，控制台显示时会根据"/"自动切分，创建work文件夹下面的test文件夹和test文件夹下的123.txt文件。

- 响应参数

  无

示例

- 请求示例

```
 PUT /ObjectName HTTP/1.1
Host: BucketName.bj.bcebos.com
Date: Wed, 06 Apr 2016 06:34:40 GMT
Authorization: AuthorizationString
Content-Type: text/plain
Content-Length: 11434
x-bce-storage-class: STANDARD_IA
[11434 bytes of object data]
```

- 响应示例

```
 HTTP/1.1 200 OK
x-bce-request-id: 4db2b34d-654d-4d8a-b49b-3049ca786409
Date: Wed, 06 Apr 2016 06:34:40 GMT
ETag: "1b2cf535f27731c974343645a3985328"
x-bce-version-id:AJyQ0XRhboY=
Content-Length: 0
Connection: close
Server: BceBos
```

PostObject

接口描述

此接口使用HTML表单上传文件到指定bucket，用于实现通过浏览器上传文件到bucket。在PutObject操作中通过HTTP请求头传递参数，在PostObject操作中使用消息实体中的表单域传递参数，其中消息实体使用多重表单格式（multipart/form-data）编码。

请求（Request）

- 请求语法

```
 POST / HTTP/1.1
Host: BucketName.bj.bcebos.com
Content-Length：<ContentLength>
Storage-class: <StorageClass>
Date：<date>
Content-Type: multipart/form-data; boundary=<boundary>

--<boundary>
Content-Disposition: form-data; name="accessKey"

499d0610679c4da2a69b64086a4cc3bc
--<boundary>
Content-Disposition: form-data; name="policy"

eyJleHBpcmF0aW9uIjoiMjAxNy0wMS0yOFQxMDo1NjoxOVoiLCJjb25kaXRpb25zIjpbWyJjb250ZW50LWxlbmd0aC1yYW5nZSIsIDMwMDAwMDAwLCA0MDAwMDAwMF0sIHsia2V5IjogImFiKiJ9LCB7ImJ1Y2tldCI6ICJib3MxN
9
--<boundary>
Content-Disposition: form-data; name="signature"

d1a617a725122c203195fe22ed9c4d20406ee259df8552e3f5344c3e1db84afe
--<boundary>
Content-Disposition: form-data; name="key"

test_object_name
--<boundary>
Content-Disposition: form-data; name="Content-Disposition"

attachment;filename="download/object"
--<boundary>
Content-Disposition: form-data; name="x-bce-meta-object-tag"

test1
--<boundary>
Content-Disposition: form-data; name="success-redirect-url"

http://demo.test.com/upload_success?object=test_object&bucket=test_bucket&time=xxx&userid=xxx
--<boundary>
Content-Disposition: form-data; name="file"; filename="upload_file"
Content-Type: text/plain

i'm test file content.
--<boundary>--
```

- 请求参数

名词	类型	描述	是否必须
accessKey	String	用户的AccessKey	可选
Cache-Control、Content-Type、Content-Disposition、Expires	String	上传object支持的Header，上传时设置这些header，下载时会带着这些header返回。	可选
file	-	上传的文本内容，必须是表单中最后一个域，如果file后面有其他域会忽略掉。	是
key	String	上传object的名称，没有这个字段会报错。	是
policy	String	policy描述表单的限制条件，不包含policy的匿名请求，只能访问公共可读写的bucket。policy必须为base64编码格式，最大限制为4096个字符。policy格式参见表格下方说明。	可选
signature	String	signature是根据secret key和policy计算的签名信息，BOS验证signature从而验证Post请求的合法性。	可选
success-action-redirect	String	上传成功之后跳转的URL。	-
success-action-status	Int	支持200,201,204；默认为200，201时Location字段返回object的位置。	-
x-bce-meta-*	String	用户自定义meta。	可选
x-bce-storage-class	String	指定Object的存储类型，`STANDARD_IA`代表低频存储，`COLD`代表冷存储，`ARCHIVE`代表归档存储，不指定时默认是标准存储类型；如果是多AZ类型bucket，`MAZ_STANDARD_IA`代表多AZ低频存储，不指定时默认是`MAZ_STANDARD`多AZ标准存储类型，不能是其它取值 。	可选
x-bce-acl	String	CannedACL支持的header，用户设置Object的权限，取值为private和public-read。	否
x-bce-grant-read	String	CannedACL支持的header，用户设置Object的读权限。支持多个id，以逗号分隔	否
x-bce-grant-full-control	String	CannedACL支持的header，用户设置Object的FULL_CONTROL权限。支持多个id，以逗号分隔。	否
x-bce-security-token	String	临时AccessKey/secretAccessKey和sessionToken鉴权必加字段。	否
x-bce-server-side-encryption	String	服务端加密算法，当前支持AES256和SM4加密。	否
x-bce-content-crc32	String	上传object的CRC值(循环冗余校验码)。	否

policy需要为UTF-8字符，支持过期时间设置和对bucket、key、文件长度的限制，bucket只支持精确匹配，object支持精确匹配和前缀匹配，格式如下：

```
{ "expiration": "2015-03-01T12:00:00Z",
 "conditions": [
     {
        "bucket": "testbucket"
     },
     {
        "key": "testkey"
     }, // 精确匹配
     {
        "key": "testkey*"
     }, // 前缀匹配, 有且只有一个*，且只能放到最后
     ["content-length-range", 0, 4096]
 ]
}
```

注意事项

- PostObject需要对Bucket有写权限，公共可读写的bucket不需要上传签名信息；否则需要验证签名。与PutObject不同，PostObject使用ak对应的sk对base64_encode之后的policy字段进行签名作为signature，BOS会验证signature，从而校验用户的合法性。
- 整个表单域key和file是必选项，如果存在上述参数列表之外的参数，会报错InvalidArgument。
- 如果PostObject里有Authorization字段，BOS不会对其检查。
- PostObject操作提交表单编码必须为“multipart/form-data”，其他格式不支持，即header中Content-Type为multipart/form-data；boundary=xxxxxx这样的形式，boundary为边界字符串。
- PostObject现已支持STS鉴权。
- 如果您的请求大小超过了policy中的content-length-range范围，会返回MaxMessageLengthExceeded错误，您可以修改content-length-range最大值为您的请求实际大小，或者删除该值。

signature的签名方式可以参考Python代码：

```
 import base64
import hashlib
import hmac

sk = bytes('<your secret key>','utf-8')
policy = bytes('your policy','utf-8')
signature = hmac.new(sk, base64.b64encode(policy), hashlib.sha256).hexdigest()
print(signature)
```

响应（Response）

- 响应头域

名称	类型	描述
Content-MD5	String	RFC2616定义的HTTP请求内容的MD5摘要，可以通过携带该字段来验证保存在BOS侧的文件和用户预期的文件是否一致。
ETag	String	Object的HTTP协议实体标签

- 响应参数

  无

示例

- 标准存储的请求示例

```
 POST / HTTP/1.1
Host: BucketName.bj.bcebos.com
Content-Length：11434
Date：Tue, 29 Mar 2016 12:00:00 GMT
Content-Type: multipart/form-data; boundary=341261481596

--341261481596
Content-Disposition: form-data; name="accessKey"

499d0610679c4da2a69b64086a4cc3bc
--341261481596
Content-Disposition: form-data; name="policy"

eyJleHBpcmF0aW9uIjoiMjAxNy0wMS0yOFQxMDo1NjoxOVoiLCJjb25kaXRpb25zIjpbWyJjb250ZW50LWxlbmd0aC1yYW5nZSIsIDMwMDAwMDAwLCA0MDAwMDAwMF0sIHsia2V5IjogImFiKJ9LCB7ImJ1Y2tldCI6ICJib3M
9
--341261481596
Content-Disposition: form-data; name="signature"

d1a617a725122c203195fe22ed9c4d20406ee259df8552e3f5344c3e1db84afe
--341261481596
Content-Disposition: form-data; name="key"

test_object_name
--341261481596
Content-Disposition: form-data; name="Content-Disposition"

attachment;filename="download/object"
--341261481596
Content-Disposition: form-data; name="x-bce-meta-object-tag"

test1
--341261481596
Content-Disposition: form-data; name="success-redirect-url"

http://demo.test.com/upload_success?object=test_object&bucket=test_bucket&time=xxx&userid=xxx
--341261481596
Content-Disposition: form-data; name="file"; filename="upload_file"
Content-Type: text/plain

i'm test file content.
--341261481596--
```

- 低频/冷存储的请求示例

```
 POST / HTTP/1.1
Host: BucketName.bj.bcebos.com
Content-Length：11434
Storage-class: STANDARD_IA
Date：Tue, 29 Mar 2016 12:00:00 GMT
Content-Type: multipart/form-data; boundary=341261481596

--341261481596
Content-Disposition: form-data; name="accessKey"

499d0610679c4da2a69b64086a4cc3bc
--341261481596
Content-Disposition: form-data; name="policy"

eyJleHBpcmF0aW9uIjoiMjAxNy0wMS0yOFQxMDo1NjoxOVoiLCJjb25kaXRpb25zIjpbWyJjb250ZW50LWxlbmd0aC1yYW5nZSIsIDMwMDAwMDAwLCA0MDAwMDAwMF0sIHsia2V5IjogImFiKJ9LCB7ImJ1Y2tldCI6ICJib3M
9
--341261481596
Content-Disposition: form-data; name="signature"

d1a617a725122c203195fe22ed9c4d20406ee259df8552e3f5344c3e1db84afe
--341261481596
Content-Disposition: form-data; name="key"

test_object_name
--341261481596
Content-Disposition: form-data; name="Content-Disposition"

attachment;filename="download/object"
--341261481596
Content-Disposition: form-data; name="x-bce-meta-object-tag"

test1
--341261481596
Content-Disposition: form-data; name="success-redirect-url"

http://demo.test.com/upload_success?object=test_object&bucket=test_bucket&time=xxx&userid=xxx
--341261481596
Content-Disposition: form-data; name="file"; filename="upload_file"
Content-Type: text/plain

i'm test file content.
--341261481596--
```

- STS鉴权方式请求示例

```
 POST / HTTP/1.1
Host: BucketName.bj.bcebos.com
Content-Length：11434
Date：Tue, 29 Mar 2016 12:00:00 GMT
Content-Type: multipart/form-data; boundary=341261481596

--341261481596
Content-Disposition: form-data; name="accessKey"

499d0610679c4da2a69b64086a4cc3bc
--341261481596
Content-Disposition: form-data; name="policy"

eyJleHBpcmF0aW9uIjoiMjAxNy0wMS0yOFQxMDo1NjoxOVoiLCJjb25kaXRpb25zIjpbWyJjb250ZW50LWxlbmd0aC1yYW5nZSIsIDMwMDAwMDAwLCA0MDAwMDAwMF0sIHsia2V5IjogImFiKiJ9LCB7ImJ1Y2tldCI6ICJib3M
9
--341261481596
Content-Disposition: form-data; name="signature"

d1a617a725122c203195fe22ed9c4d20406ee259df8552e3f5344c3e1db84afe
--341261481596
Content-Disposition: form-data; name="x-bce-security-token"

ds3wHsWRSAJFeafjkAji2OfJIOLijRk32IRRAJfMISTRhIAJfSJHkdsa
--341261481596
Content-Disposition: form-data; name="key"

test_object_name
--341261481596
Content-Disposition: form-data; name="Content-Disposition"

attachment;filename="download/object"
--341261481596
Content-Disposition: form-data; name="x-bce-meta-object-tag"

test1
--341261481596
Content-Disposition: form-data; name="success-redirect-url"

http://demo.test.com/upload_success?object=test_object&bucket=test_bucket&time=xxx&userid=xxx
--341261481596
Content-Disposition: form-data; name="file"; filename="upload_file"
Content-Type: text/plain

i'm test file content.
--341261481596--
```

- 响应示例

```
 HTTP/1.1 200 OK
x-bce-request-id: 4db2b34d-654d-4d8a-b49b-3049ca786409
Date：Tue, 29 Mar 2016 12:00:00 GMT
ETag: "1b2cf535f27731c974343645a3985328"
Content-MD5: H2koac2M0YsMxDNte2XJ8A==
Content-Length: 0
Connection: close
Server: BceBOS
```

CopyObject

接口描述

此接口用于把一个已经存在的Object拷贝为另外一个Object，支持Object文件的长度范围是0Byte-5GB。该接口也可以用来实现Meta更新（使用replace模式且源和目标指向同一个文件）。此接口需要请求者在header中指定拷贝源。

CopyObject接口支持跨区域文件复制，即复制文件所在的源Bucket和目标Bucket可以不在同一region(目前只支持从其它Region向本Region复制数据)。当进行跨区域文件复制时，复制产生的流量会收取跨区域流量费。跨区域收费标准参见产品定价。

请求（Request）

- 请求语法

```
 PUT /<ObjectKey> HTTP/1.1
Host: <BucketName>.bj.bcebos.com
Date: <Date>
Authorization: <AuthorizationString>
Content-Length: <ContentLength>
Content-Type:text/plain
x-bce-copy-source: /SourceBucket/SourceObject
x-bce-copy-source-if-match: 3858f62230ac3c915f300c664312c11f
x-bce-metadata-directive: <DirectiveString>
x-bce-storage-class: <StorageClass>
```

- 请求参数

```
 无
```

- 请求头域

名称	类型	描述	是否必需
x-bce-copy-source	String	源Object地址	是
x-bce-copy-source-if-match	String	如果源Object的ETag值和用户提供的ETag相等，则执行拷贝操作，否则拷贝失败。	否
x-bce-metadata-directive	String	目的Object的Meta信息是从源Object拷贝，还是用请求传入的meta。有效值为 `copy` 和 `replace` ，缺省值为 `copy` 。如果设置为 `copy` ，则直接用源Object的meta；如果设置为 `replace` ，则用请求传入的meta。	否
x-bce-meta-*	String	replace模式下启动，可修改用户自定义meta，自定义meta是用来保存对象的自定义信息。例如设置文件的标签，参数用“x-bce-meta-gender”，值设置为“Male”。	否
x-bce-copy-source-if-none-match	String	如果源Object的ETag和用户提供的ETag不相等，则执行拷贝操作，否则拷贝失败。	否
x-bce-copy-source-if-unmodified-since	String	如果源object在x-bce-copy-source-if-unmodified-since之后没被修改，则执行拷贝操作，否则拷贝失败。参数取值为GMT格式，例如：Wed, 06 Apr 2016 06:34:40 GMT。	否
x-bce-copy-source-if-modified-since	String	如果源object在x-bce-copy-source-if-modified-since之后被修改了，则执行拷贝操作，否则拷贝失败。参数取值为GMT格式，例如：Wed, 06 Apr 2016 06:34:40 GMT。	否
x-bce-storage-class	String	指定Object的存储类型， `STANDARD_IA` 代表低频存储， `COLD` 代表冷存储， `ARCHIVE` 代表归档存储，不指定时默认是标准存储类型；如果是多AZ类型bucket， `MAZ_STANDARD_IA` 代表多AZ低频存储，不指定时默认是 `MAZ_STANDARD` 多AZ标准存储类型，不能是其它取值。	否
x-bce-acl	String	CannedACL支持的header，用户设置Object的权限，取值为private和public-read。	否
x-bce-grant-read	String	CannedACL支持的header，用户设置Object的读权限。支持多个id，以逗号分隔	否
x-bce-grant-full-control	String	CannedACL支持的header，用户设置Object的FULL_CONTROL权限。支持多个id，以逗号分隔	否
x-bce-server-side-encryption	String	服务端加密算法，当前支持AES256和SM4加密。	否
x-bce-tagging-directive	String	copy是否携带原Object的对象标签,取值为"Copy"(默认值): 复制源Object的对象标签到目标 Object; "Replace" : 忽略源Object的对象标签	否

响应（Response）

- 响应头域

名称	类型	描述
x-bce-version-id	String	Object的版本Id

- 响应参数

名称	类型	描述
ETag	String	目的Object的ETag
lastModified	DATE	目的Object的最后一次修改时间

> **注意事项**
>
> 1. 请求者必须对源Object有读操作权限。
> 2. 在计算签名之前，用户需要针对x-bce-copy-source字段中为非标准ASCII字符（例如：中文）的内容做一次url-encode。
> 3. 为了保持复制过程中的http连接，CopyObject接口的http结果可能使用Transfer-Encoded: Chunked编码方式。
> 4. CopyObject过程中，如果发生服务器端错误，http status code可能返回2XX但是复制失败，复制结果请根据http body中的json判定。
> 5. CopyObject如果源object是归档类型，需要先取回归档类型才能调用CopyObject接口。
> 6. 归档类型Object不支持通过CopyObject实现Meta更新（使用replace模式且源和目标指向同一个文件）。
> 7. 如果复制软链接文件，并不会复制数据，只会复制软链接本身。如果使用软链接访问该接口，且软链接的目标文件删除了，会返回Http 404，SymlinkTargetNotExist。

示例

- 标准存储的请求示例

```
PUT /ObjectName HTTP/1.1
Host: BucketName.bj.bcebos.com
Date: Wed, 06 Apr 2016 06:34:40 GMT
Authorization: AuthorizationString
Content-Length: 0
Content-Type:text/plain
x-bce-copy-source: /SourceBucket/SourceObject
x-bce-copy-source-if-match: 3858f62230ac3c915f300c664312c11f
x-bce-metadata-directive: replace
x-bce-meta-mykey: myvalue
```

- 低频/冷存储的请求示例

```
PUT /object HTTP/1.1
Host: BucketName.bj.bcebos.com
Date: Wed, 06 Apr 2016 06:34:40 GMT
Authorization: AuthorizationString
Content-Length: 0
Content-Type:text/plain
x-bce-copy-source: /SourceBucket/SourceObject
x-bce-copy-source-if-match: 3858f62230ac3c915f300c664312c11f
x-bce-storage-class: STANDARD_IA
```

- 响应示例

Copy成功

```
 HTTP/1.1 200 OK
x-bce-request-id: 4db2b34d-654d-4d8a-b49b-3049ca786409
Date: Wed, 06 Apr 2016 06:34:40 GMT
Connection: close
Server: BceBos

{
    "lastModified":"2009-10-28T22:32:00Z",
    "ETag":"9b2cf535f27731c974343645a3985328"
}
```

服务端异常， 需要根据返回json判断

```
 HTTP/1.1 200 OK
Date: Thu, 12 May 2016 09:14:32 GMT
Content-Type: application/json; charset=utf-8
Connection: keep-alive
Server: BceBos
x-bce-request-id: bb90cc9c-2b80-462c-87a4-095e610c9a2f
Transfer-Encoding: chunked
{
    "code":"InternalError",
    "message":"We encountered an internal error. Please try again.",
    "requestId":"52454655-5345-4420-4259-204e47494e58"
}
```

多版本的请求示例

```
 PUT /object HTTP/1.1
Host: BucketName.bj.bcebos.com
Date: Wed, 06 Apr 2016 06:34:40 GMT
Authorization: AuthorizationString
Content-Length: 0
Content-Type:text/plain
x-bce-copy-source: /SourceBucket/SourceObject?versionId=AJyQ0XRhboY=
x-bce-copy-source-if-match: 3858f62230ac3c915f300c664312c11f
x-bce-storage-class: STANDARD_IA
```

- 响应示例

Copy成功

```
 HTTP/1.1 200 OK
x-bce-request-id: 41dffrad-654d-4d8a-b49b-304dsd34fadd5df59
Date: Wed, 06 Apr 2024 08:34:40 GMT
Connection: close
x-bce-version-id:AKyQ9DRhhoY=
Server: BceBos

{
    "lastModified":"2024-04-06T08:34:40Z",
    "ETag":"9b2cf535f27731c974343645a3985328"
}
```

GetObject

接口描述

此命令用于从BOS获取某个Object。此操作需要请求者对该Object有读权限。请求者可以在Header中设置Range来指定需要获取的Object数据的范围。

请求（Request）

- 请求语法

```
 GET /<ObjectKey> HTTP/1.1
Host: <BucketName>.bj.bcebos.com
Date: <Date>
Authorization: <AuthorizationString>
Range: <Range_String>
```

- 多版本请求语法

```
 GET /<ObjectKey>?versionId=<VersionId> HTTP/1.1
Host: <BucketName>.bj.bcebos.com
Date: <Date>
Authorization: <AuthorizationString>
Range: <Range_String>
```

- 请求头域

名称	类型	描述	是否必需
Range	String	指定Object返回的文件范围。设定 bytes=0-9，表示传送第0到第9这10个字符。默认返回全部数据。	否

- 请求参数

字段	类型	是否必需	描述
versionId	String	否	获取目标的版本Id。默认为空，获取最新版本；指定为"null"则获取空版本信息。

响应（Response）

- 响应头域

名称	类型	描述
Cache-Control	String	下载Object的Cache设置，常见的可取值为private、no-cache、max-age、must-revalidate
Content-Disposition	String	设置浏览器是否下载，可取值为inline、attachment; filename="download.txt"
Content-Length	Long Int	返回Object的数据大小
Content-Range	String	有range的情况下返回Object的数据范围
Content-Type	String	Object的类型及编码方式
Expires	String	下载Object时的缓存失效时间
ETag	String	Object的HTTP协议实体标签
x-bce-meta-*	String	如果有自定meta，才返回此项
x-bce-storage-class	String	标准存储返回STANDARD，低频存储返回STANDARD_IA，冷存储返回COLD，归档存储返回ARCHIVE；如果是多AZ类型bucket，多AZ低频存储返回MAZ_STANDARD_IA，多AZ标准存储返回MAZ_STANDARD。
x-bce-server-side-encryption	String	Object的服务器端加密类型，当前支持AES256和SM4加密。
x-bce-content-crc32	String	返回Object的crc32。
x-bce-version-id	String	返回Object的版本Id。

> **注意事项**
>
> - GetObject通过Range参数可以支持断点续传，对于比较大的Object建议使用该功能。
> - 如果在请求头中使用Range参数；则返回消息中会包含整个文件的长度和此次返回的范围，例如：Content-Range: bytes 0-9/44，表示整个文件长度为 44，此次返回的范围为0-9。
> - 对于Range读大小为0字节的对象， 会返回400错误， Range是前闭后闭区间。
> - 归档存储类型对象需要先取回才能调用GetObject接口。

如您想在response请求里面获得某些特定的header信息，可通过如下两种方式：

- 在PutObject时增加header信息，则GetObject时会直接返回在response里面，请参考PutObject接口。

  > **注意：** responseContentDisposition的设置在BOS自有域名($region.bcebos.com和*.$region.bcebos.com无效)。

- GetObject时，在queryString里面直接增加"responseXXX=YYY"，返回的header里面会包含"XXX: YYY"。例如：GET /testBucket/testObject?responseContentType= image/jpg，在这个http请求的response里面就会有"Content-Type: image/jpg"。

  具体的形式为**response$header=urlencode(value)**，需要注意，当前$header只支持"ContentDisposition、ContentType、ContentLanguage、Expires、CacheControl、ContentEncoding"。

- 响应元素

```
无
```

示例

- 请求示例

```
GET /ObjectName
Host: BucketName.bj.bcebos.com
Date: Wed, 06 Apr 2016 06:34:40 GMT
Authorization: AuthorizationString
Range: bytes=0-9
```

- 响应示例

```
HTTP/1.1 206 Partial Content
x-bce-request-id: 4db2b34d-654d-4d8a-b49b-3049ca786409
x-bce-storage-class: STANDARD
Date: Wed, 06 Apr 2016 06:34:40 GMT
Last-Modified: Fri, 28 Jan 2011 20:10:32 GMT
ETag: "b2419b1e3fd45d596ee22bdf62aaaa2f"
Accept-Ranges: bytes
Content-Range: bytes 0-9/443
Content-Type: text/plain
Content-Length: 10
Server: BceBos

[10 bytes of object data]
```

- 多版本的请求示例

```
GET /ObjectName?versionId=ADCQMIGFiNo%3D
Host: BucketName.bj.bcebos.com
Date: Wed, 06 Apr 2016 06:34:40 GMT
Authorization: AuthorizationString
Range: bytes=0-9
```

- 响应示例

```
HTTP/1.1 206
x-bce-request-id: ffaa6a28-1d13-44dd-9266-6ae6181cffdc
x-bce-storage-class: STANDARD
Date: Wed, 06 Apr 2016 06:34:40 GMT
Last-Modified: Fri, 28 Jan 2011 20:10:32 GMT
ETag: "b2419b1e3fd45d596ee22bdf62aaaa2f"
Accept-Ranges: bytes
Content-Range: bytes 0-9/443
Content-Type: text/plain
Content-Length: 10
x-bce-version-id:ADCQMlGFiNo=
Server: BceBos

[10 bytes of object data]
```

GetObjectMeta

接口描述

此命令用于获取某个Object的Meta信息，但此时并不返回数据。

注意：

- 如果使用软链接访问GetObjectMeta, 返回响应头中Content-Length、ETag、Content-Md5 均为目标文件的元信息；Last-Modified是目标文件和软链接的最大值；其他均为软链接元信息。
- 如果使用软链接访问该接口且目标文件删除了，会返回Http 404，SymlinkTargetNotExist,调用GetSymlink不会报错。

请求（Request）

- 请求语法

```
HEAD /<ObjectName> HTTP/1.1
Host: <BucketName>.bj.bcebos.com
Date: <Date>
Authorization: <AuthorizationString>
```

- 多版本请求语法

```
HEAD /<ObjectName>?versionId=<VersionId> HTTP/1.1
Host: <BucketName>.bj.bcebos.com
Date: <Date>
Authorization: <AuthorizationString>
```

- 请求头域

无特殊header参数

- 请求参数

字段	类型	是否必需	描述
versionId	String	否	获取对象的版本Id。默认为空，获取最新版本信息；如果为"null"，指定获取空版本的信息。

响应（Response）

- 响应头域

名称	类型	描述
Cache-Control	String	下载Object的Cache设置，常见的可取值为private、no-cache、max-age、must-revalidate
Content-Disposition	String	设置浏览器是否下载，可取值为inline、attachment; filename="download.txt"
Content-Length	Long Int	返回Object的数据大小
Content-Range	String	有range的情况下返回Object的数据范围
Content-Type	String	Object的类型及编码方式
Expires	String	下载Object时的缓存失效时间
ETag	String	Object的HTTP协议实体标签
x-bce-meta-*	String	如果有自定meta，才返回此项
x-bce-storage-class	String	标准存储返回STANDARD，低频存储返回STANDARD_IA，冷存储返回COLD，归档存储返回ARCHIVE；如果是多AZ类型bucket，多AZ低频存储返回MAZ_STANDARD_IA，多AZ标准存储返回MAZ_STANDARD。
x-bce-server-side-encryption	String	Object的服务器端加密类型，当前支持AES256和SM4加密。
x-bce-restore	String	当对象为归档存储类型，并且处于正在取回或已经取回时，会返回头域"x-bce-restore"，表示归档类型对象的取回状态：正在取回时返回ongoing-request="true"；已取回归档对象时，返回ongoing-request="false", expiry-date="Wed, 07 Nov 2019 00:00:00 GMT"。其中expiry-date表示对象取回后的失效时间。
x-bce-object-type	String	当访问的是软链接，则返回值"Symlink"。其他情况暂不返回
x-bce-version-id	String	对象的版本Id

- 响应参数

无

注意事项

无

示例

- 请求示例

```
HEAD /ObjectName HTTP/1.1
Host: BucketName.bj.bcebos.com
Date: Wed, 06 Apr 2016 06:34:40 GMT
Authorization: AuthorizationString
```

- 响应示例

```
HTTP/1.1 200 OK
x-bce-request-id: 4db2b34d-654d-4d8a-b49b-3049ca786409
x-bce-storage-class: STANDARD
Date: Wed, 06 Apr 2016 06:34:40 GMT
Last-Modified: Sun, 1 Jan 2006 12:00:00 GMT
ETag: "fba9dede5f27731c9771645a39863328"
Content-Length: 0
Content-Type: text/plain
Connection: close
Server: BceBos
```

- 归档类型对象已取回时响应示例

```
HTTP/1.1 200 OK
x-bce-request-id: 4db2b34d-654d-4d8a-b49b-3049ca786409
x-bce-storage-class: ARCHIVE
x-bce-restore: ongoing-request="false", expiry-date="Wed, 07 Nov 2019 00:00:00  GMT"
Date: Wed, 27 Feb 2019 06:34:40 GMT
Last-Modified: Wed, 20 Feb 2019 06:34:40 GMT
ETag: "fba9dede5f27731c9771645a39863328"
Content-Length: 15618
Content-Type: text/plain
Connection: close
Server: BceBos
```

- 多版本请求示例

```
HEAD /ObjectName?versionId=ADCQMIGFiNo%3D HTTP/1.1
Host: BucketName.bj.bcebos.com
Date: Wed, 06 Apr 2016 06:34:40 GMT
Authorization: AuthorizationString
```

- 响应示例

```
HTTP/1.1 200 OK
x-bce-request-id: 4db2b34d-654d-4ddd-b49b-3049ca786409
x-bce-storage-class: STANDARD
Date: Wed, 06 Apr 2016 06:34:40 GMT
Last-Modified: Sun, 1 Jan 2022 12:01:00 GMT
ETag: "fba9dede5f27731c9771645a39863328"
Content-Length: 0
Content-Type: text/plain
Connection: close
Server: BceBos
x-bce-version-id:ADCQMIGFiNo=
```

RestoreObject

接口描述

RestoreObject用于取回归档存储文件，请求者必须有归档存储文件的读权限，并且归档存储文件处于冰冻状态。

注意：

- 如果使用软链接访问该接口，会根据软链接的目标文件的存储类型和状态进行处理。
- 如果使用软链接访问该接口且目标文件删除了，会返回Http 404，SymlinkTargetNotExist。
- 多版本环境下，RestoreObject后的文件为空版本

请求（Request）

- 请求语法

```
POST /ObjectName?restore HTTP/1.1
Host: <BucketName>.bj.bcebos.com
Date: <Date>
Content-Length: 0
Authorization: <Authorization_String>
x-bce-restore-tier:Expedited/Standard
x-bce-restore-days:3
```

- 请求头域

名称	类型	描述	是否必需
x-bce-restore-days	int	表示解冻后状态的持续时间，要求是数字，单位是天，取值大于0且不超过30即可。默认解冻持续天数是7天,最长30天。	否
x-bce-restore-tier	string	解冻优先级。可取值范围如下： • Expedited：加急取回，表示30min内完成取回 • Standard：标准取回，表示2~5小时内完成取回 • LowCost：延缓取回，表示12小时内完成取回	否，默认是标准取回

- 请求参数

  无特殊参数

响应（Response）

- 响应头域

  无特殊响应头域。

- 响应参数

  无

注意事项

无

示例

- 请求示例

```
 POST /ObjectName?restore HTTP/1.1
Host: <BucketName>.bj.bcebos.com
x-bce-date: <Date>
Content-Length: 0
x-bce-restore-days: 3
x-bce-restore-tier: Expedited
```

- 响应示例

```
 HTTP/1.1 200 OK
x-bce-request-id: 4db4b34d-653d-4d9a-b49b-3049ca786409
Date: Wed, 31 May 2018 08:34:40 GMT
Server: BceBos
```

FetchObject

接口描述

此接口用于从指定URL抓取资源，并将资源存储到指定的Bucket中。此操作需要请求者对该Bucket有写权限，每次只能抓取一个Object，且用户可以自定义Object的名称。

说明：

- FetchObject接口抓取资源的大小限制为0~48.8TB。

请求（Request）

- 请求语法

```
 POST /<ObjectName>?fetch HTTP/1.1
Host: <BucketName>.bj.bcebos.com
Content-Length: 0
Date: <Date>
x-bce-fetch-source: <Source>
x-bce-fetch-mode: <FetchMode>
x-bce-storage-class: <StorageClass>
Authorization: AuthorizationString
```

- 请求头域

名称	类型	描述	是否必需
x-bce-fetch-source	String	抓取文件的源地址，如http://www.abc.com/img.jpg。该参数可以放到querystring里，当放到querystring时需要做UrlEncode。	是
x-bce-fetch-mode	String	抓取模式，支持异步抓取async和同步抓取sync两种模式。其中异步模式下BOS收到抓取任务后立刻返回成功，同步模式下BOS需要等到抓取Object完成后才会返回成功。对于数据量小或关注抓取结果实时性的建议使用同步模式，异步模式不需要等待，适用不需要实时查看抓取结果的场景，可以后续再对抓取结果进行查询。默认值为sync。 该参数可以放到querystring里。	否
x-bce-storage-class	String	存储类型，支持标准存储STANDARD、低频存储STANDARD_IA、冷存储COLD和归档存储ARCHIVE，默认存储为STANDARD；如果是多AZ类型bucket，支持MAZ_STANDARD_IA代表多AZ低频存储，不指定时默认是MAZ_STANDARD多AZ标准存储类型，不能是其它取值。该参数可以放到querystring里。	否
x-bce-server-side-encryption	String	服务端加密算法，当前支持AES256和SM4加密。	否
x-bce-callback-address	String	fetch结果回调地址，若选择异步抓取，抓取完成后会将结果推送到该地址。	否
referer	string	源站fetch时需要透传的Referer头域	否
user-agent	string	源站fetch时需要透传的User-Agent头域	否

- 请求参数

  无

响应（Response）

- 响应头域

  无

- 响应参数

名称	类型	描述
code	String	返回请求成功或失败的返回码，成功返回success,错误返回错误码。
message	String	返回请求成功或失败的信息，成功返回success,错误返回code对应的错误码。
requestId	String	请求ID。
jobId	String	异步模式才会返回该参数，表示请求任务的ID号，可用于后续任务状态查询。
x-bce-version-id	String	Object的版本Id

示例

- 请求示例（同步模式）

```
 POST /ObjectName?fetch HTTP/1.1
Host: BucketName.bj.bcebos.com
Content-Length: 0
Date: <Date>
x-bce-fetch-source: http://www.abc.com/demo.html
x-bce-fetch-mode: sync
x-bce-storage-class: STANDARD
Authorization: AuthorizationString
```

- 响应示例（同步模式）

```
 HTTP/1.1 200 OK
x-bce-request-id: 4db2b34d-654d-4d8a-b49b-3049ca786409
Transfer-Encoding: chunked
Date: Wed, 25 May 2016 06:34:40 GMT
Server: BceBos

{
    "code": "success",
    "message": "success",
    "requestId": "4db2b34d-654d-4d8a-b49b-3049ca786409",
}
```

- 请求示例（异步模式）

```
 POST /ObjectName?fetch HTTP/1.1
Host: BucketName.bj.bcebos.com
Content-Length: 0
Date: <Date>
x-bce-fetch-source: http://www.abc.com/demo.html
x-bce-fetch-mode: async
x-bce-storage-class: STANDARD_IA
Authorization: AuthorizationString
```

- 响应示例（异步模式）

```
 HTTP/1.1 200 OK
x-bce-request-id: 4db2b34d-654d-4d8a-b49b-3049ca786409
Content-Length: 43
Date: Wed, 25 May 2016 06:34:40 GMT
Server: BceBos

{
    "code": "success",
    "message": "success",
    "requestId": "4db2b34d-654d-4d8a-b49b-3049ca786409",
    "jobId": "b2419b1e3fd45d596ee22bdf62aaaa2f"
}
```

- 请求示例（异步模式，通知回调地址）

```
 POST /ObjectName?fetch HTTP/1.1
Host: BucketName.bj.bcebos.com
Content-Length: 0
Date: <Date>
x-bce-fetch-source: http://www.abc.com/demo.html
x-bce-fetch-mode: async
x-bce-storage-class: STANDARD_IA
x-bce-callback-address:<callback-address>
Authorization: AuthorizationString
```

- 响应示例（异步模式，通知回调地址）

```
 HTTP/1.1 200 OK
x-bce-request-id: 4db2b34d-654d-4d8a-b49b-3049ca786409
Content-Length: 43
Date: Wed, 25 May 2016 06:34:40 GMT
Server: BceBos

{
    "code": "success",
    "message": "success",
    "requestId": "4db2b34d-654d-4d8a-b49b-3049ca786409",
    "jobId": "b2419b1e3fd45d596ee22bdf62aaaa2f"
}
```

回调地址收到的信息示例如下

fetch成功

```
{"code":"success","jobid":"fo-9a98f238d56a33b4eb6664fede20b747","message":"success","requestId":"61c41bdf-672b-49c4-921e-3d6bde8112a6"}
```

fetch失败

```
{"code":"NoSuchKey","jobid":"fo-9a98f238d56a33b4eb6664fede20b747","message":"The specified key does not exist.","requestId":"24ff8d6c-f793-45ef-b55b-973837141c90"}
```

DeleteObject

接口描述

此命令用于删除指定Bucket的一个Object，要求请求者对此Object有写权限。

请求（Request）

- 请求语法

```
 DELETE /<ObjectName> HTTP/1.1
Host: <BucketName>.bj.bcebos.com
Date: <Date>
Authorization: <AuthorizationString>
```

- 多版本请求语法

```
 DELETE /<ObjectName>?versionId=<VersionId> HTTP/1.1
Host: <BucketName>.bj.bcebos.com
Date: <Date>
Authorization: <AuthorizationString>
```

- 请求头域

无特殊Header参数

- 请求参数

字段	类型	是否必需	描述
versionId	String	否	获取目标的版本Id。默认为空，获取最新版本；指定为"null"则获取空版本信息。

响应（Response）

- 响应头域

无特殊Header参数返回

- 响应参数

无

注意事项

无

示例

- 请求示例

```
DELETE /ObjectName HTTP/1.1
Host: BucketName.bj.bcebos.com
Date: Wed, 06 Apr 2016 06:34:40 GMT
Authorization: AuthorizationString
```

- 响应示例

```
HTTP/1.1 204 No Content
x-bce-request-id: 4db2b34d-654d-4d8a-b49b-3049ca786409
Date: Wed, 06 Apr 2016 06:34:40 GMT
Content-Length: 0
Connection: close
Server: BceBos
```

- 多版本请求示例

```
DELETE /ObjectName?versionId=ADCQMlGFiNo%3D HTTP/1.1
Host: BucketName.bj.bcebos.com
Date: Wed, 06 Apr 2016 06:34:40 GMT
Authorization: AuthorizationString
```

- 响应示例

```
HTTP/1.1 204 No Content
x-bce-request-id: 4db2b34d-654d-4d8a-b49b-3049ca786409
Date: Wed, 06 Apr 2016 06:34:40 GMT
Content-Length: 0
Connection: close
Server: BceBos
```

DeleteMultipleObjects

接口描述

该命令可以实现通过一个HTTP请求删除同一个Bucket下的多个Object。

- 支持一次请求内最多删除1000个Object。
- 消息体（body）不超过2M。
- 返回的消息体中只包含删除过程中出错的Object结果；如果所有Object都删除都成功的话，则没有消息体。

请求（Request）

- 请求语法

```
POST /?delete HTTP/1.1
Host: <BucketName>.bj.bcebos.com
Date: <Date>
Authorization: <AuthorizationString>
Content-Length: <ContentLength>
Content-Type: text/plain

{
    "objects": [
        {
            "key": "my-object1"
        },
        {
            "key": "my-object2"
        }
    ]
}
```

- 请求头域

无特殊Header参数

- 请求参数

参数名称	Query参数	描述	父节点
objects	Request Body参数	保存要删除的Object信息的容器，里面包含一个或多个Object元素。	-
+key	Request Body参数	要删除的Object名称。	objects

响应（Response）

- 响应头域

无特殊Header参数返回

- 响应参数

参数名称	描述	父节点
errors	删除过程中出错的Object信息的容器，里面包含一个或多个Object元素。	-
+key	删除出错的Object名称。	errors
+code	错误代码。	errors
+message	错误信息。	errors

示例

- 请求示例

```
 POST /?delete HTTP/1.1
Host: <BucketName>.bj.bcebos.com
Date: Wed, 06 Apr 2016 06:34:40 GMT
Authorization: AuthorizationString
Content-Length: 11434
Content-Type: text/plain

{
   "objects": [
      {
         "key": "my-object1"
      },
      {
         "key": "my-object2"
      }
    ]
}
```

- 响应示例

```
 HTTP/1.1 200 OK
x-bce-request-id：4db2b34d-654d-4d8a-b49b-3049ca786409
Date: Wed, 06 Apr 2016 06:34:40 GMT
Connection: close
Content-Length: 1324
Server: BceBos

{
   "errors": [
      {
         "key": "my-object1",
         "code": "NoSuchKey",
         "message": "The specified key does not exist."
      },
      {
         "key": "my-object2",
         "code": "InvalidArgument",
         "message": "Invalid Argument."
      }
   ]
}
```

AppendObject

接口描述

AppendObject以追加写的方式上传文件。通过AppendObject操作创建的Object类型为Appendable Object，可以对该Object追加数据；而通过PutObject上传的Object是Normal Object，不可进行数据追加写。

说明：

- Appendable Object大小限制为0~48.8T
- AppendObject接口在进行追加写时要求对该Object有写权限
- 归档类型对象暂时不支持AppendObject。
- 软链接不是可追加类型，会返回HTTP 403，ObjectUnappendable错误。
- 当前版本下，多版本的bucket不可进行AppendObject的写入，对于历史写入的AppendObject可以读取。

请求（Request）

- 请求语法

首次上传AppendObject 时，使用如下方法：

```
 POST /<ObjectName>?append HTTP/1.1
Host: <BucketName>.bj.bcebos.com
Date: <GMT Date>
Authorization: <AuthorizationString>
Content-Type: text/plain
Content-Length: <ContentLength>
x-bce-storage-class: <StorageClass>
```

> **说明：**
>
> - 如该Object不存在，则创建一个Appendable的Object。
> - 如已存在同名的Object，则上传的数据会覆盖原有的Object，无论原有Object是否为Appendable，均创建新的Appendable Object。
> - 允许创建或者覆盖一个长度为0的Object。

如已上传了部分AppendObject，需要进行断点续传时，使用如下方法：

```
 POST /<ObjectName>?append&offset=<OffsetSize> HTTP/1.1
Host: <BucketName>.bj.bcebos.com
Date: <GMT Date>
Authorization: <AuthorizationString>
Content-Type: text/plain
Content-Length: <ContentLength>
```

> **说明：**
>
> - 如要续传的Object不存在时，会返回错误码404 NoSuchKey。
> - 如要续传的Object并不是Appendable的，则会返回403 ObjectUnappendable。
> - 如值错误，则返回409 OffsetIncorrect。
> - 如果不指定offset会直接覆盖而不是默认追加在末尾

- 请求头域

名称	类型	描述	是否必须
Cache-Control	String	浏览器cache的一种机制设置	否
Content-Disposition	String	指示浏览器如何显示附加的文件	否
Content-MD5	String	RFC2616定义的HTTP请求内容的MD5摘要，可以通过携带该字段来验证保存在BOS侧的文件和用户预期的文件是否一致。	否
Expires	String	GMT时间，缓存失效时间	否
x-bce-meta-*	String	用户自定义的meta	否
x-bce-content-sha256	String	指本次所传数据的sha256值，必须与此匹配，否则AppendObject失败	否
x-bce-content-crc32	String	本次上传object增量数据的CRC值(循环冗余校验码)。	否
x-bce-storage-class	String	指定Object的存储类型，STANDARD_IA代表低频存储，COLD代表冷存储，ARCHIVE代表归档存储，不指定时默认是标准存储类型；如果是多AZ类型bucket，MAZ_STANDARD_IA代表多AZ低频存储，不指定时默认是MAZ_STANDARD多AZ标准存储类型，不能是其它取值	否
x-bce-acl	String	CannedACL支持的header，用户设置Object的权限，取值为private和public-read。	否
x-bce-grant-read	String	CannedACL支持的header，用户设置Object的读权限。支持多个id，以逗号分隔	否
x-bce-grant-full-control	String	CannedACL支持的header，用户设置Object的FULL_CONTROL权限。支持多个id，以逗号分隔	否
x-bce-server-side-encryption	String	服务端加密算法，当前支持AES256和SM4加密。	否

- 请求参数

名称	类型	参数位置	描述	是否必须
append	-	Query参数	代表对Appendable Object进行追加操作	是
offset	-	Query参数	代表数据断点，即从该点后继续追加，取值为已实际上传的数据大小	否

响应（Response）

- 响应头域

名称	类型	描述	是否必须
Content-MD5	String	当前已成功上传的整个object的MD5值	是
x-bce-next-append-offset	Long Int	指明下次AppendObject请求时传入的OffsetSize的值。如果不使用续传方式，即请求起始行没有offset，则这个值是0	是
x-bce-content-crc32	String	整个object数据的CRC值(循环冗余校验码)。	否
ETag	String	Append后整个Object的ETag值	是

- 响应参数

无

使用细节说明

对于Appendable的Object：

- 支持AppendObject。
- 支持RenameObject。
- 对一个已存在的Appendable的Object执行PutObject，则Appendable的Object会被覆盖变成普通Object。
- 对一个已存在的Appendable的Object，在<OffsetSize>值正确的情况下，追加一个长度为0的内容，不会改变Object的任何状态。
- 不建议并发操作同一个Object，如一个线程执行AppendObject，另一线程执行PutObject/CopyObjcect/DeleteObject等操作，可能会导致操作结果错误或操作失败。
- 如果对同一个Object做多线程或者多进程的并发追加，有可能因为并发过程的不确定性，返回409 OffsetIncorrect失败。

对其他接口的影响

对于GetObjectMeta接口，当一个Object是Appendable时，返回的结果中会多如下两个字段：

名称	类型	描述	是否必须
x-bce-next-append-offset	String	当Object是由AppendObject接口创建的，会返回该字段，指明下次AppendObject时请求传入的**OffsetSize**的值。如果此时Object的大小是5G，offset依然返回5G	是
x-bce-object-type	String	当Object的类型值。当object是由AppendObject创建的，返回的值为Appendable，其他情况暂不返回	是

关于用户自定义的x-bce-meta-*

- 在首次创建AppendableObject（即Http请求起始行不带offset参数）时，携带的x-bce-meta-*Header会被存储，后续追加AppendObject（即Http请求起始行携带offset参数）时的x-bce-meta-*Header则默认被忽略。
- 由于Appendable Object支持CopyObject，因此可以使用CopyObject自我拷贝以覆盖的形式来修改x-bce-meta-* Header的值。

数据校验

- 每次调用AppendObject的时候，如您上传MD5或者Sha256值，BOS会对该次调用上传的数据进行校验。
- 每次数据追加结束，会返回已上传的整个Object的MD5值。

示例

- 标准存储的请求示例一

```
 POST /ObjectName?append  HTTP/1.1
Host: BucketName.bj.bcebos.com
Date: Wed, 06 Apr 2016 06:34:40 GMT
Authorization: AuthorizationString
Content-Type: text/plain
Content-Length: 1134

[1134 bytes of object data]
```

- 低频/冷存储的请求示例一

```
 POST /ObjectName?append  HTTP/1.1
Host: BucketName.bj.bcebos.com
Date: Wed, 06 Apr 2016 06:34:40 GMT
Authorization: AuthorizationString
Content-Type: text/plain
Content-Length: 1134
x-bce-storage-class: STANDARD_IA

[1134 bytes of object data]
```

- 响应示例一

```
 HTTP/1.1 200 OK
x-bce-request-id: c31b374e-048a-41f0-9a9a-31bc4bc57509
Date: Wed, 06 Apr 2016 06:34:40 GMT
ETag: "7c935a3947e3a684333480bd6b58b7c2"
Content-Length: 0
Content-MD5: RJBidEhsrgCKeDjvQjrF8A==
x-bce-next-append-offset: 1134
Connection: close
Server: BceBos
```

- 标准存储的请求示例二

```
 POST /ObjectName?append&offset=1134 HTTP/1.1
Host: BucketName.bj.bcebos.com
Date: Wed, 06 Apr 2016 06:34:40 GMT
Authorization: AuthorizationString
Content-Type: text/plain
Content-Length: 1900

[1900 bytes of object data]
```

- 低频/冷存储的请求示例二

```
 POST /my-object?append&offset=1134 HTTP/1.1
Host: BucketName.bj.bcebos.com
Date: Wed, 06 Apr 2016 06:34:40 GMT
Authorization: AuthorizationString
Content-Type: text/plain
Content-Length: 1900
x-bce-storage-class: STANDARD_IA

[1900 bytes of object data]
```

- 响应示例二

```
 HTTP/1.1 200 OK
x-bce-request-id: 28a99102-d0a5-4252-a4ed-fa6dc801806b
Date: Wed, 06 Apr 2016 06:34:40 GMT
ETag: "11257f81dce31a95f67f6e75018b77e3"
Content-Length: 0
Content-MD5: fJNaOUfjpoQzNIC9a1i3wg==
x-bce-next-append-offset: 3034
Connection: close
Server: BceBos
```

OPTIONSObject

接口描述

浏览器在发送跨域请求之前会发送一个preflight请求（OPTIONS）并带上特定的来源域，HTTP方法和Header信息等给BOS以决定是否发送真正的请求，本接口即响应这种请求。

权限说明

OPTIONS Object操作不需要进行鉴权。

请求

- 请求语法

```
 OPTIONS /<ObjectKey> HTTP/1.1
Host: <BucketName>.bj.bcebos.com
Origin: Origin
Access-Control-Request-Method: HTTPMethod
Access-Control-Request-Headers: RequestHeader
```

- 请求参数

  无

- 请求头域

名称	描述	是否必需
Origin	请求来源域，用来标识跨域请求，只允许一个方法。类型：字符串。默认值：无。	是
Access-Control-Request-Method	在实际请求中将会用到的方法，只允许一个方法。类型：字符串。取值为"PUT/GET/DELETE/POST/HEAD"，无默认值。	是
Access-Control-Request-Headers	在实际请求中会用到的除了简单头部之外的Headers，多个headers用逗号分割。类型：字符串。默认值：无。	否

- 请求元素

  无

响应

- 响应头域

名称	描述
Access-Control-Allow-Credentials	BOS的Server端是否允许客户端在请求中带cookie。需要客户端和Server端同时允许才能生效。BOS在请求通过时会返回允许，即值为`true`，大小写敏感。
Access-Control-Allow-Headers	允许请求携带的Header的列表，如果请求中有不被允许的Header，则CORS规则匹配失败，返回不包含任何Access-Control-开头的头部。多个Headers也用逗号分隔。注意：返回仅仅包括此次请求的Headers列表，而不管配置文件中是否配置了更多被允许的Headers。
Access-Control-Allow-Methods	允许请求的HTTP方法，如果不允许该请求，则不包含该头部以及所有的Access-Control-*相关的头部。类型：字符串。注意：返回的是所有允许的allowedMethods列表，而不仅仅包括此次请求的method。
Access-Control-Allow-Origin	请求中包含的Origin，如果不允许的话将不包含该头部以及所有的Access-Control-*相关的头部。类型：字符串。
Access-Control-Expose-Headers	允许在请求端的JavaScript程序中访问的Headers的列表。与配置文件中的exposeHeaders对应。类型：字符串。
Access-Control-Max-Age	允许浏览器缓存preflight结果的时间，单位为秒。类型：整形。

- 响应元素

  无

> **注意事项**
>
> - 当CORS收到OPTIONS请求后，会读取Bucket对应的CORS规则，然后进行相应的权限检查。整个检查会依次检查每一条规则，使用第一条匹配的规则来允许请求并返回对应的Header。如果所有规则都匹配失败则不附加任何CORS相关的Header。
> - CORS规则匹配成功必须满足三个条件：
>   - 请求的Origin必须匹配一项allowedOrigins中一项。
>   - OPTIONS请求的Access-Control-Request-Method头对应的方法必须匹配一项 allowedMethods中一项。
>   - OPTIONS请求的Access-Control-Request-Headers头包含的每个Header都必须匹配一项 allowedHeader项(只要有一个不符合则整体失败)。

示例

- 请求示例

```
 OPTIONS /object HTTP/1.1
Host: BucketName.bj.bcebos.com
Origin: http://www.example.com
Access-Control-Request-Method: GET
Access-Control-Request-Headers: x-bce-test
```

- 响应示例

```
 HTTP/1.1 200 OK
x-bce-request-id: 4db2b34d-654d-4d8a-b49b-3049ca786409
Date: Wed, 06 Apr 2016 06:34:40 GMT
Access-Control-Allow-Origin: http://www.example.com
Access-Control-Allow-Methods: GET, HEAD, DELETE
Access-Control-Allow-Headers: x-bce-test
Access-Control-Expose-Headers: user-custom-expose-header
Access-Control-Max-Age: 3600
Access-Control-Allow-Credentials: true
Content-Length: 0
Server: BceBos
```

listObjectVersions

接口描述

本接口用于获得指定Bucket的Object多版本信息列表。

请求（Request）

* ### 请求语法

```json
GET /?versions HTTP/1.1
Host: <BucketName>.bj.bcebos.com
Date: <Date>
Authorization: <AuthorizationString>
```

* ### 请求头域

无特殊Header参数

* ### 请求参数

字段	类型	是否必需	描述
delimiter	String	否	分隔符; 主要用此项实现list文件夹的逻辑。如果在请求的时候指定了delimiter，BOS把匹配到的Object名称按照一定规则（从preifx到第一个delimiter）截取，截取的字符串去重作为CommonPrefixes的数据返回; delimiter长度限制为1
marker	String	否	object为字母序排列，从marker之后的第一个开始返回
maxKeys	Int	否	返回object列表长度最大为1000，默认值是1000，如果指定的值大于1000，按1000操作
prefix	String	否	key前缀，限定返回的object key必须以此为前缀
versionIdMarker	String	否	versionId的前缀，从versionIdMarker之后的第一个开始返回

响应（Response）

* ### 响应头域

无特殊Header参数响应

* ### 响应元素

名称	类型	描述
commonPrefixes	Array	仅当指定delimiter，才会返回此项
+prefix	String	匹配以prefix开始到第一次出现Delimiter字符之间的object作为一组元素返回
contents	Array	返回的一个object的列表
+key	String	Object名称
+lastModified	Date	此Object最后一次被修改的时间
+eTag	String	Object的HTTP协议实体标签
+size	Int	Object的内容的大小（字节数）
+versionId	String	Object的版本号
+isLatest	Int	Object是否为最新版本
+storageClass	String	Object的存储类型，低频存储返回`STANDARD_IA`，冷存储返回`COLD`，归档存储返回`ARCHIVE`，标准存储返回`STANDARD`。
+owner	Container	Object上传者的用户信息
++id	String	Object上传者的用户id
++displayName	String	Object上传者的名称
delimiter	String	查询的结束符
isTruncated	Bool	指明是否查询完整返回了；false-本次已经返回所有结果，true-本次还没有返回所有结果
maxKeys	Int	请求返回的最大数目
marker	String	本次查询的起点
name	String	Bucket名称
nextMarker	String	当IsTruncated true时，才返回此项，作为下次查询marker的值
nextVersionIdMarker	String	当IsTruncated true时，才返回此项，作为下次查询versionIdMarker的值
prefix	String	查询的前缀

> **注意事项**
>
> Delimiter可以用来实现文件夹逻辑：
>
> 1. 如果把prefix设为某个文件夹名，就可以罗列以此prefix开头的文件，即该文件夹下递归的所有的文件和子文件夹内的文件。
> 2. 如果再把delimiter设置为 / 时，返回值就只罗列该文件夹下的文件，该文件夹下的子文件夹名返回在CommonPrefixes部分，子文件夹下递归的文件不被显示。 如一个bucket下存在三个object：fun/test.jpg，fun/movie/001.avi，fun/movie/007.avi。 若设定prefix为“fun/”，则返回三个object；如果增加设定delimiter为“/”，则返回文件“fun/test.jpg”和前缀“fun/movie/”。

示例

* ### 请求示例

```json
GET ?maxKeys=1&versions HTTP/1.1
Host: <BucketName>.bj.bcebos.com
Date: Wed, 06 Apr 2016 06:34:40 GMT
Authorization: <AuthorizationString>
```

* ### 响应示例，列举当前bucket下的object

```json
HTTP/1.1 200 OK
x-bce-request-id: 4db2b34d-654d-4d8a-b49b-3049ca786409
Date: Wed, 06 Apr 2016 06:34:40 GMT
Server: BceBos

{
    "name":"bucket",
    "prefix":"",
    "delimiter":"",
    "marker":"",
    "maxKeys":1,
    "isTruncated":false,
    "nextVersionIdMarker":"AKyQMlG4ihY=",
```

```
        "contents":[
          {
            "key":"my-image.jpg",
            "lastModified":"2009-10-12T17:50:30Z",
            "eTag":"fba9dede5f27731c9771645a39863328",
            "size":434234,
            "storageClass":"STANDARD",
            "isLatest":1,
            "versionId":"AKyQMIG4ihY=",
            "owner":{
               "id":"168bf6fd8fa74d9789f35a283a1f15e2",
               "displayName":"mtd"
            }
          }
        ]
    }
```

权限控制

GetObjectAcl

接口描述

此命令用来获取某个Object的访问权限。

请求（Request）

* ### 请求语法

```
GET /<ObjectName>?acl HTTP/1.1
Host: <BucketName>.bj.bcebos.com
x-bce-date: <Date>
Authorization: <AuthorizationString>
```

* ### 请求头域

无特殊参数

* ### 请求参数

无特殊参数

响应（Response）

* ### 响应头域

无特殊Header参数返回

* ### 响应参数

名称	类型	描述
accessControlList	Objcet	ACL访问控制列表
+grantee	Objcet	一个被授权人
++id	String	被授权用户名id
+permission	String	授权权限支持：FULL_CONTROL，READ

示例

* ### 请求示例

```
GET /ObjectKey?acl HTTP/1.1
Host: BucketName.bj.bcebos.com
x-bce-date: 2017-05-01T12:23:49Z
Authorization: AuthorizationString
```

* ### 响应示例

```
HTTP/1.1 200 OK
Date: Wed, 01 Mar 2017 12:25:00 GMT
Server: BceBos
x-bce-request-id: 236A23124128905248E5A01

{
    "accessControlList":[
        {
            "grantee":[{
                "id":"e13b12d0131b4c8bae959df4969387b8"
            }],
            "permission":["FULL_CONTROL"]
        },
        {
            "grantee":[{
                "id":"8c47a952db4444c5a097b41be3f24c94"
            }],
            "permission":["READ"]
        }
    ]
}
```

PutObjectAcl

接口描述

此命令用于设置Object的访问权限。目前BOS支持两种方式设置ACL。第一种是使用Canned Acl，在PutObjectAcl的时候，通过头域的"x-bce-acl"或者"x-bce-grant-permission'来设置object访问权限，当前可设置的权限包括private和public-read，两种类型的header不可以同时在一个请求中出现。第二种方式是上传一个ACL文件，文件格式参见[ACL文件格式](BOS/API参考/访问控制.md#上传ACL文件方式的权限控制)，目前ACL文件只支持accessControlList，grantee，id，permission字段。

目前不支持在同一请求中同时设置canned acl和上传ACL文件。

请求（Request）

* ### ACL文件请求语法

```
PUT /<ObjectKey>?acl HTTP/1.1
Host: <BucketName>.bj.bcebos.com
x-bce-date: <Date>
Authorization: <AuthorizationString>
Content-Type: application/json; charset=utf-8
Content-Length: <ContentLength>
```

* ### Canned ACL文件请求语法（设置x-bce-acl）

```
PUT /<ObjectKey>?acl HTTP/1.1
Host: <BucketName>.bj.bcebos.com
x-bce-date: <Date>
Authorization: <AuthorizationString>
x-bce-acl: <ObjectAcl>
Content-Length: <ContentLength>
Content-Type: application/json; charset=utf-8
```

* ### Canned ACL文件请求语法（设置x-bce-grant-permission）

```
PUT /<ObjectKey>?acl HTTP/1.1
Host: <BucketName>.bj.bcebos.com
x-bce-date: <Date>
Authorization: <AuthorizationString>
x-bce-grant-read: <ObjectGrantRead>
Content-Length: <ContentLength>
Content-Type: application/json; charset=utf-8
```

* ### 请求头域

名称	类型	描述	是否必需
x-bce-acl	String	Object设置的ACL权限，支持：private、public-read	否
x-bce-grant-read	String	授权读的user id，支持多个id，以逗号分隔	否
x-bce-grant-full-control	String	授权控制权限的user id，支持多个id，以逗号分隔	否

>**注意事项**
>
> - 只有Bucket的拥有者和被授予FULL_CONTROL权限的用户才能设置Object的ACL权限。
> - 在上传Object时，Object权限会默认为空，如果没有设置Object的权限，即当Object权限为空时，默认以Bucket权限为准。
> - 当Object权限和Bucket权限不一致时，以Object权限为准。
> - 归档存储类型对象在取回未完成，或者刚上传归档类型文件（时长参考取回时长）时，不能设置Object acl。

* ### 请求参数

无特殊参数

响应（Response）

* ### 响应头域

无特殊参数返回

* ### 响应参数

无

示例

* ### ACL文件请求示例

```
PUT /ObjectName?acl HTTP/1.1
Host: BucketName.bj.bcebos.com
x-bce-date: 2017-05-01T12:23:49Z
Authorization: AuthorizationString
Content-Type: application/json; charset=utf-8
Content-Length: 315

{
    "accessControlList":[
    {
        "grantee":[{
            "id":"e13b12d0131b4c8bae959df4969387b8"
        }],
        "permission":["READ"]
    }
    ]
}
```

* ### Canned ACL请求示例（设置x-bce-acl）

```
PUT /ObjectName?acl HTTP/1.1
Host: BucketName.bj.bcebos.com
x-bce-date: 2017-05-01T12:23:49Z
```

```
x-bce-date: 2017-05-01T12:23:49Z
Authorization: AuthorizationString
x-bce-acl: public-read
Content-Length: 0
Content-Type: application/json; charset=utf-8
```

* ### Canned ACL请求示例（设置x-bce-grant-permission）

```
PUT /ObjectName?acl HTTP/1.1
Host: BucketName.bj.bcebos.com
x-bce-date: 2017-05-01T12:23:49Z
Authorization: AuthorizationString
x-bce-grant-read:id="e13b12d0131b4c8bae959df4969387b8",id="8c47a952db4444c5a097b41be3f24c94"
Content-Length: 0
Content-Type: application/json; charset=utf-8
```

* ### 响应示例

```
HTTP/1.1 200 OK
Date: Wed, 01 Mar 2017 12:25:00 GMT
Content-Length: 0
Server: BceBos
x-bce-request-id:413e34fd-118d-4049-b992-1b1f3a68b1f5
```

DeleteObjectAcl

接口描述

此命令用来删除某个Object的访问权限。

请求（Request）

* ### 请求语法

```
DELETE /<ObjectName>?acl HTTP/1.1
Host: <BucketName>.bj.bcebos.com
x-bce-date: <Date>
Authorization: <AuthorizationString>
```

* ### 请求头域

无特殊Header参数

* ### 请求参数

无特殊参数

响应（Response）

* ### 响应头域

无特殊参数返回

* ### 响应参数

无特殊参数返回

> **注意事项**
> - 归档存储类型对象在取回未完成，或者刚上传归档类型文件（时长参考取回时长）时，不能删除Object acl。

示例

* ### 请求示例

```
DELETE /ObjectName?acl HTTP/1.1
Host: BucketName.bj.bcebos.com
x-bce-date: 2017-05-01T12:23:49Z
Authorization: AuthorizationString
```

* ### 响应示例

```
HTTP/1.1 200 OK
Date: Wed, 01 Mar 2017 12:25:00 GMT
Content-Length: 0
Server: BceBos
x-bce-request-id:413e34fd-118d-4049-b992-1b1f3a68b1f5
```

Select扫描

SelectObject

接口描述

此接口用于向Bucket中指定object执行SQL语句，选取出指定内容返回。请求者必须对选取的object具有read权限。在SelectObject前需要确保对应的Bucket和Object已经存在，详细信息参考 [SelectObject](https://cloud.baidu.com/doc/BOS/s/pk1rmdy5t) 开发者文档。

请求（Request）

* ### 请求语法

```
POST /<ObjectKey>?select&type=json/csv/parquet HTTP/1.1
Host: <BucketName>.bj.bcebos.com
```

```
Date: <Date>
Authorization: <Authorization_String>
Content-Type: application/json; charset=utf-8
Content-Length: <Content_Length>
{
    "selectRequest": {
        "expression": "Base64Encode(Select * from BosObject)",
        "expressionType": "SQL",
        "inputSerialization": {
            "compressionType": "GZIP/NONE",
            // JSON or CSV or Parquet
        },
        "outputSerialization": {
            // JSON or CSV or Parquet
        },
        "requestProgress": {
            "enabled": false/true
        }
    }
}
```

* ### 请求参数

名词	类型	描述	是否必须
type	string	select的目标object类型，目前支持json/csv/parquet	是

* ### 请求主体

JSON文件

名称	是否必须	类型	描述
selectRequest	是	-	json body根节点
+ expression	是	string	经过base64编码的sql语句
+ expressionType	是	string	查询语句语法类型，只支持"SQL"
+ inputSerialization	是	-	输入流节点，其子节点描述查询的object格式信息
++compressionType	否	string	指定查询的object是否压缩，可选值为"NONE"或"GZIP"
++ json	是	-	json节点，其子节点描述json文件相关信息
+++ type	是	string	json type，查询的json object格式，可选值为"DOCUMENT"或"LINES"
+ outputSerialization	是	-	输出流节点，其子节点描述查询结果返回格式信息
++ json	是		json节点，其子节点描述返回的json数据相关信息
+++ recordDelimiter	否	string	指定换行符，以Base64编码。默认值为\n（可选）
+ requestProgress	否	-	select进度信息节点，其子节点描述select操作的执行进度，3s定时返回给用户
++ enabled	否	boolean	描述是否需要定时返回进度信息，可选值为false/true；如果数据筛选耗时久可能导致504超时，设置其为true可以维持http连接

CSV文件

名称	是否必须	类型	描述
selectRequest	是	-	json body根节点
+ expression	是	string	经过base64编码的sql语句
+ expressionType	是	string	查询语句语法类型，只支持"SQL"
+ inputSerialization	是	-	输入流节点，其子节点描述查询的object格式信息
++ compressionType	否	string	指定查询的object是否压缩，可选值为"NONE"或"GZIP"
++ csv	是	-	csv节点，其子节点描述csv文件相关信息
+++ fileHeaderInfo	否	string	可选值NONE/IGNORE/USE，指定CSV文件首行头信息，默认值NONE代表无header信息，IGNORE代表有header信息但是忽略，NONE和IGNORE意味着只能用列序号取出某一列；USE表示使用header信息，只能用header名取出某一列
+++ recordDelimiter	否	string	指定csv文件换行符，以Base64编码，默认值为\n（可选），最多为2个字符，例如\r\n
+++ fieldDelimiter	否	string	指定csv文件列分隔符，以Base64编码，默认值为,（可选），最多为1个字符，例如;
+++ quoteCharacter	否	string	指定csv文件引号字符，以Base64编码，在CSV中引号内的换行符，列分隔符将被视作普通字符；默认值为双引号"（可选），最多为1个字符，例如单引号'
+++ commentCharacter	否	string	指定csv文件注释符，以Base64编码，一行以注释符开头则表示该行可以忽略；默认值为#（可选），最多为2个字符，例如//
+ outputSerialization	是	-	输出流节点，其子节点描述查询结果返回格式信息
++ outputHeader	否	boolean	在返回结果开头输出CSV头信息，默认false; 需要fileHeaderInfo字段取值USE，该值为true会在返回结果的每一个Records message第一行加上对应的csv header名称；可选值为false/true
++ csv	是	-	csv节点，其子节点描述返回的csv数据相关信息
+++ quoteFields	否	string	可选值ALWAYS/ASNEEDED，指定返回的csv数据每个字段是否使用""包围; ASNEEDED表示返回文件中原始数据，ALWAYS表示总是用""包围field; 默认是ASNEEDED
+++ recordDelimiter	否	string	指定返回的csv数据的换行符，以Base64编码。默认值为\n（可选），最多为2个字符，例如\r\n
+++ fieldDelimiter	否	string	指定返回的csv数据的列分隔符，以Base64编码。默认值为,（可选），最多为1个字符，例如;
+++ quoteCharacter	否	string	指定返回的csv数据的引号字符，以Base64编码，在CSV中引号内的换行符，列分隔符将被视作普通字符；默认值为双引号"（可选），最多为1个字符，例如单引号'
+ requestProgress	否	-	select进度信息节点，其子节点描述select操作的执行进度，3s定时返回给用户
++ enabled	否	boolean	描述是否需要定时返回进度信息，可选值为false/true；如果数据筛选耗时久可能导致504超时，设置其为true可以维持http连接

Parquet文件

名称	是否必须	类型	描述
selectRequest	是	-	json body根节点
+ expression	是	string	经过base64编码的sql语句
+ expressionType	是	string	查询语句语法类型，只支持"SQL"
+ inputSerialization	是	-	输入流节点，其子节点描述查询的object格式信息
++ compressionType	否	string	指定查询的object是否压缩，可选值为"NONE"
++ parquet	是	-	parquet节点，其子节点描述parquet文件相关信息，为空
+ outputSerialization	是	-	输出流节点，其子节点描述查询结果返回格式信息
++ json	是		json节点，其子节点描述返回的json数据相关信息
+++ recordDelimiter	否	string	指定换行符，以Base64编码。默认值为\n（可选）
+ requestProgress	否	-	select进度信息节点，其子节点描述select操作的执行进度，3s定时返回给用户
++ enabled	否	boolean	描述是否需要定时返回进度信息，可选值为false/true；如果数据筛选耗时久可能导致504超时，设置其为true可以维持http连接

* ### 请求头域

无特殊头域

响应（Response）

* ### 响应头域

名称	类型	描述
Transfer-Encoding	String	值为chunked，表示返回内容以http1.1 chunked编码方式分块返回

* ### 响应参数

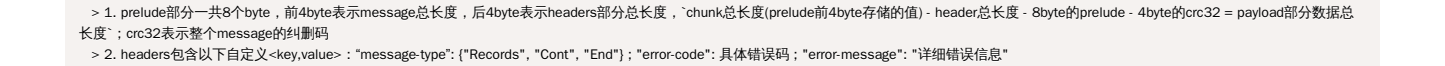
无

* ### 响应主体

SelectObject接口响应以chunk形式返回，包括Records message，Continuation message，End message三种类型：

message	格式	描述
Records message	prelude(8 byte) + n * (header_key_len(1 byte) + header_key + header_val_len(2 byte) + header_val) + payload + crc32(4 byte)	包含Select请求返回的数据，可以是一行，也可以是多行记录
Continuation message	同上，固定格式，只是payload内容不一样	每3秒定时给client返回当前select进展(扫描字节数/返回字节数)以及维持http连接
End message	同上，固定格式，payload内容为空	标志着该次select请求的结束，headers字段包括error-code，error-message，message-type，bytes-scanned等信息

message具体格式详细说明：



> 1. prelude部分一共8个byte，前4byte表示message总长度，后4byte表示headers部分总长度，`chunk总长度(prelude前4byte存储的值) - header总长度 - 8byte的prelude - 4byte的crc32 = payload部分数据总长度`；crc32表示整个message的纠删码
> 2. headers包含以下自定义<key,value>：“message-type": {"Records", "Cont", "End"}；"error-code": 具体错误码；"error-message": "详细错误信息"
> 3. Payload表示返回的真实数据，可以是任意格式；其中Continuation message的Payload里边包括BytesScanned和BytesReturned两个字段表示的select进度信息

示例
* ### CSV文件请求示例

```
POST /object?select&type=csv HTTP/1.1
Host: bucket.bj.bcebos.com
Date: Thu, 15 May 2017 00:17:23 GMT
Authorization: <Authorization_String>
Content-Type: application/json; charset=utf-8
Content-Length: 512
{
    "selectRequest": {
        "expression": "c2VsZWN0IGNvdW50KCopIGZyb20gbxkl2JqZWN0IHdoZXJlIF80ID4gNDU=",
        "expressionType": "SQL",
        "inputSerialization": {
            "compressionType": "NONE",
            "csv": {
                "fileHeaderInfo": "NONE",
                "recordDelimiter": "Cg==",
                "fieldDelimiter": "LA==",
                "quoteCharacter": "Ig==",
                "commentCharacter": "Iw=="
            }
        },
        "outputSerialization": {
            "outputHeader": FALSE,
            "csv": {
                "quoteFields": "ALWAYS",
                "recordDelimiter": "Cg==",
                "fieldDelimiter": "LA==",
                "quoteCharacter": "Ig=="
            }
        },
        "requestProgress": {
            "enabled": false
        }
    }
}
```

* ### CSV文件响应示例
```
HTTP/1.1 200 OK
x-bce-request-id: 4db2b34d-654d-4d8a-b49b-3049ca786409
Date: Wed, 06 Apr 2016 06:34:40 GMT
ETag: "1b2cf535f27731c974343645a3985328"
Transfer-Encoding: chunked
Connection: close
Server: BceBos
----- Body ------
<Records message>
……
<Continuation Message>
……
<Records message>
<Continuation Message>
<End message>
```
* ### JSON文件请求示例

```
POST /object?select&type=json HTTP/1.1
Host: bucket.bj.bcebos.com
Date: Thu, 15 May 2017 00:17:23 GMT
Authorization: <Authorization_String>
Content-Type: application/json; charset=utf-8
Content-Length: 512
{
    "selectRequest": {
```

```
    "selectRequest": {
        "expression": "c2VsZWN0IGNvdW50KCopIGZyb20gbxkI2JqZWN0IHdoZXJIIF80ID4gNDU=",
        "expressionType": "SQL",
        "inputSerialization": {
            "compressionType": "NONE",
            "json": {
                "type": "DOCUMENT"
            }
        },
        "outputSerialization": {
            "json": {
                "recordDelimiter": "Cg=="
            }
        },
        "requestProgress": {
            "enabled": false
        }
    }
}
```

* ### JSON文件响应示例

```
HTTP/1.1 200 OK
x-bce-request-id: 4db2b34d-654d-4d8a-b49b-3049ca786409
Date: Wed, 06 Apr 2016 06:34:40 GMT
ETag: "1b2cf535f27731c974343645a3985328"
Transfer-Encoding: chunked
Connection: close
Server: BceBos
----- Body ------
<Records message>
......
<Continuation Message>
......
<Records message>
<Continuation Message>
<End message>
```

* ### Parquet文件请求示例

```
POST /object?select&type=parquet HTTP/1.1
Host: bucket.bj.bcebos.com
Date: Thu, 15 May 2017 00:17:23 GMT
Authorization: <Authorization_String>
Content-Type: application/json; charset=utf-8
Content-Length: 512
{
    "selectRequest": {
        "expression": "c2VsZWN0IGNvdW50KCopIGZyb20gbxkI2JqZWN0IHdoZXJIIF80ID4gNDU=",
        "expressionType": "SQL",
        "inputSerialization": {
            "compressionType": "NONE",
            "parquet": {}
        },
        "outputSerialization": {
            "json": {
                "recordDelimiter": "Cg=="
            }
        },
        "requestProgress": {
            "enabled": false
        }
    }
}
```

* ### Parquet文件响应示例

```
HTTP/1.1 200 OK
x-bce-request-id: 4db2b34d-654d-4d8a-b49b-3049ca786409
Date: Wed, 06 Apr 2016 06:34:40 GMT
ETag: "1b2cf535f27731c974343645a3985328"
Transfer-Encoding: chunked
Connection: close
Server: BceBos
----- Body ------
<Records message>
......
<Continuation Message>
......
<Records message>
<Continuation Message>
<End message>
```

软链接

PutSymlink

接口描述

PutSymlink接口用于为BOS已有的目的object（本文档称作"target object"）创建软链接（Symlink），您可以通过该软链接访问目的object。

注意：
- 软链接文件和target object可以位于相同region内的不同bucket，若设置x-bce-symlink-bucket，即表示跨bucket创建软连接，x-bce-symlink-bucket指定为target object所在bucket。
- 目前支持对标准存储、低频存储、冷存储、归档存储类型的object文件设置软链接。软链接文件本身不支持ARCHIVE类型。如果用户指定ARCHIVE类型，或者不指定类型且Bucket默认存储类型是ARCHIVE时，返回报错CODE_INVALID_ARGUMENT。
- 调用该接口创建软链接时，不会检查目标文件是否存在、以及是否有权限访问目标文件。使用软链接调用GetObject接口时候，会检查用户是否有软链接和目标文件的读取权限、目标文件是否存在。
- 创建软链接需要有相应写权限。
- 如果不指定"x-bce-forbid-overwrite"这个Header，如果存在与软链接文件同名的object文件时，创建软链接操作时会默认覆盖同名Object。
- 不支持二级软链接。用户创建二级软链时不会进行检查，但是用户无法通过二级软链访问指向的object数据。
- 通过软链接下载目标文件时，需要有软链接和目标文件的读权限。

请求

* ### 请求语法

PUT /?symlink HTTP/1.1 Host: .bj.bcebos.com Date: Authorization: x-bce-symlink-target: x-bce-symlink-bucket: x-bce-forbid-overwrite: false

* ### 请求参数
无特殊参数

* ### 请求头域
名称	类型	描述	是否必选
x-bce-symlink-target	String	指定软链接的目标文件。命名规范同Object。软链接的目标文件不能为软链接文件	是
x-bce-symlink-bucket	String	指定软链接的目标文件所在bucket，若不设置，则表示被创建的软链接与软链接目标文件在同一个bucket下。	否
x-bce-forbid-overwrite	String	指定创建软链接操作时是否覆盖同名Object。如果不指定x-bce-forbid-overwrite时，默认允许覆盖同名Object。指定x-bce-forbid-overwrite为true时，表示禁止覆盖同名Object；指定x-bce-forbid-overwrite为false时，表示允许覆盖同名Object。	否
x-bce-meta-*	String	用户自定义的meta	否
x-bce-storage-class	String	指定软链接文件的存储类型，STANDARD代表标准存储，STANDARD_IA代表低频存储，COLD代表冷存储,不指定时默认是bucket存储类型。软链接文件本身不支持ARCHIVE类型。	否
x-bce-acl	String	CannedACL支持的header，用户设置软链接的权限，取值为private和public-read。	否
x-bce-grant-read	String	CannedACL支持的header，用户设置软链接的读权限。支持多个id，以逗号分隔	否
x-bce-grant-full-control	String	CannedACL支持的header，用户设置软链接的FULL_CONTROL权限。支持多个id，以逗号分隔	否

响应

* ### 响应头域
名称	类型	描述
ETag	String	软链接目标Object名字的HTTP协议实体标签

* ### 响应元素
无特殊元素

示例

* ### 请求示例

PUT /?symlink HTTP/1.1 Host: .bj.bcebos.com Date: Authorization: x-bce-symlink-target: x-bce-forbid-overwrite: false x-bce-grant-read:id="e13bxxxxxxxxxxxxxxxxxxxxxxxx87b8",id="8c47xxxxxxxxxxxxxxxxxxxxxxxx4c94"

* ### 响应示例

HTTP/1.1 200 OK x-bce-request-id: 4db4b34d-653d-4d9a-b49b-3049ca786409 Date: Wed, 31 May 2018 08:34:40 GMT ETag: "1b2cf535f27731c974343645a3985328" Server: BceBos

GetSymlink

接口描述

GetSymlink接口用于获取软链接。此操作需要您对该软链接有读权限。

请求

* ### 请求语法

GET /?symlink HTTP/1.1 Host: .bj.bcebos.com Date: Content-Length: 0 Authorization: AuthorizationString

* ### 请求参数
无特殊参数

* ### 请求头域
无特殊头域

响应

* ### 响应头域
名称	类型	描述
x-bce-symlink-target	String	软链接指向的目标文件

* ### 响应元素
无特殊元素

示例

* ### 请求示例

GET /?symlink HTTP/1.1 Host: .bj.bcebos.com Date: Content-Length: 0 Authorization:

* ### 响应示例

HTTP/1.1 200 OK x-bce-request-id: 4db4b34d-653d-4d9a-b49b-3049ca786409 Date: Server: BceBos Content-Type: text/plain Content-Length: 0 x-bce-symlink-target:

对象标签

PutObjectTagging

接口描述

此接口用于向指定的object上传对象标签，目前BOS支持两种方式设置对象标签。在PutObjectTagging的时候，通过头域的"x-bce-tagging"来设置object对象标签，第二种方式是上传一个TAG文件。

请求（Request）

* ### TAG文件请求语法

```
PUT /<ObjectKey>?tagging HTTP/1.1
Host: <BucketName>.bj.bcebos.com
x-bce-date: <Date>
Authorization: <AuthorizationString>
Content-Type: application/json; charset=utf-8
Content-Length: <ContentLength>
```

* ### Canned TAG请求语法（设置x-bce-tagging）

```
PUT /<ObjectKey>?tagging HTTP/1.1
Host: <BucketName>.bj.bcebos.com
x-bce-date: <Date>
x-bce-tagging: <ObjectTag>
Authorization: <AuthorizationString>
Content-Length: <ContentLength>
Content-Type: application/json; charset=utf-8
```

* ### 请求参数

无特殊参数

* ### 请求头域

名称	类型	描述	是否必需
x-bce-tagging	String	用户自定义的对象标签	否

* ### 请求参数

无特殊参数

响应（Response）

* ### 响应头域

无特殊参数返回

* ### 响应参数

无

示例

* ### 请求示例1

PUT /ObjectName?tagging HTTP/1.1 Host: BucketName.bj.bcebos.com x-bce-date: 2017-05-01T12:23:49Z x-bce-tagging: bos1=value1&bos2=values2 Authorization: AuthorizationString Content-Type: application/json; charset=utf-8 Content-Length: 0

* ### 响应示例

HTTP/1.1 200 OK Date: Wed, 01 Mar 2017 12:25:00 GMT Content-Length: 0 Server: BceBos x-bce-request-id:413e34fd-118d-4049-b992-1b1f3a68b1f5

* ### 请求示例2

PUT /ObjectName?tagging HTTP/1.1 Host: BucketName.bj.bcebos.com x-bce-date: 2017-05-01T12:23:49Z Authorization: AuthorizationString Content-Type: application/json; charset=utf-8 Content-Length: 315 {"tagSet":[{"tagInfo":[{"key":"key1","value":"value1"}, {"key":"key2","value":"value2"},{"key":"key3","value":"value3"}]}]}

* ### 响应示例

HTTP/1.1 200 OK Date: Wed, 01 Mar 2017 12:25:00 GMT Content-Length: 0 Server: BceBos x-bce-request-id:413e34fd-118d-4049-b992-1b1f3a68b1f5

GetObjectTagging

接口描述

此命令用来获取某个Object的对象标签列表。

请求（Request）

* ### 请求语法

```
GET /<ObjectKey>?tagging HTTP/1.1
Host: <BucketName>.bj.bcebos.com
Date: <Date>
Authorization: <AuthorizationString>
```

* ### 请求头域

无特殊参数

* ### 请求参数

无特殊参数

响应（Response）

* ### 响应头域

无特殊Header参数返回

* ### 响应参数

名称	类型	描述
tagSet	Objcet	对象标签集合
+tagInfo	Objcet	对象标签信息

示例

* ### 请求示例

```
GET /objectKey?tagging HTTP/1.1
Host: BucketName.bj.bcebos.com
x-bce-date: 2017-05-01T12:23:49Z
Authorization: AuthorizationString
```

* ### 响应示例

```
HTTP/1.1 200 OK
Date: Wed, 01 Mar 2017 12:25:00 GMT
Server: BceBos
x-bce-request-id: 236A23124128905248E5A01
{
 "tagSet": [{
  "tagInfo": {
   "key9": "value9",
   "key8": "value8",
   "key10": "value10",
   "key3": "value3",
   "key2": "value2",
   "key1": "value1",
   "key0": "value0",
   "key6": "value6",
   "key5": "value5",
   "key4": "value4"
  }
 }]
}
```

DeleteObjectTagging

接口描述

本接口用来删除特定Object的tag配置

请求

* ### 请求语法

```
DELETE /<ObjectName>?tagging HTTP/1.1
Host: <BucketName>.bj.bcebos.com
Date: <Date>
Authorization: <Authorization_String>
```

* ### 请求头域

无特殊参数

* ### 请求参数

无特殊参数

响应

* ### 响应头域

无特殊元素

* ### 响应元素

无特殊元素

示例

* ### 请求示例

```
DELETE /ObjectName?tagging HTTP/1.1
```

```
Host: BucketName.bj.bcebos.com
Date: <Date>
Authorization: <Authorization_String>
```

* ### 响应示例

```
HTTP/1.1 200 OK
x-bce-request-id: 4db2b34d-654d-4d8a-b49b-3049ca786409
Date: Wed, 06 Apr 2016 06:34:40 GMT
Server: BceBos

```

分片上传相关接口

使用指南

概述
MultipartUpload相关接口用于实现BOS的“三步上传”和“三步Copy”功能。

- 三步上传：请求者将一个Object拆分成多个分块（又称Part），然后分别上传这些分块。当所有分块全部上传完成后，BOS将请求者上传的所有分块组合成完整Object。MultipartUpload常使用于流式上传，大文件上传和断点上传。MultipartUpload上传有InitiateMultipartUpload，UploadPart和CompleteMultipartUpload三个步骤，俗称“三步上传”。 三步上传相关的接口包括：
  - [InitMultiUpload](BOS/API参考/分片上传相关接口/InitiateMultipartUpload.md)
  - [UploadPart](BOS/API参考/分片上传相关接口/UploadPart.md)
  - [CompleteMultiUpload](BOS/API参考/分片上传相关接口/CompleteMultipartUpload.md)
  - [ListParts](BOS/API参考/分片上传相关接口/ListParts.md)
  - [AbortMultiUpload](BOS/API参考/分片上传相关接口/AbortMultipartUpload.md)
  - [ListMultipartUploads](BOS/API参考/分片上传相关接口/ListMultipartUploads.md)

- 三步Copy：三步Copy和三步上传实现原理类似，将一个Object分块复制，最后再合并为一个完整的Object。三步Copy通过将大任务切分为小任务，可以并发复制，提高了文件的复制效率和成功率。三步Copy有InitiateMultipartUpload，UploadPartCopy和CompleteMultipartUpload三个步骤，相关接口包括：
  - [InitMultiUpload](BOS/API参考/分片上传相关接口/InitiateMultipartUpload.md)
  - [UploadPartCopy](BOS/API参考/分片上传相关接口/UploadPartCopy.md)
  - [CompleteMultiUpload](BOS/API参考/分片上传相关接口/CompleteMultipartUpload.md)
  - [ListParts](BOS/API参考/分片上传相关接口/ListParts.md)
  - [AbortMultiUpload](BOS/API参考/分片上传相关接口/AbortMultipartUpload.md)
  - [ListMultipartUploads](BOS/API参考/分片上传相关接口/ListMultipartUploads.md)

InitiateMultipartUpload

接口描述

InitiateMultipartUpload是MultipartUpload的第一步，此命令向BOS请求一个全局唯一的UploadId，用于表示此次MultipartUpload，在MultipartUpload后续两个步骤都需要此UploadId，请求者也可以通过UploadId来查询上传的进度或者中断这次上传操作。

请求（Request）

* ### 请求语法

```
POST /<ObjectName>?uploads HTTP/1.1
Host: <BucketName>.bj.bcebos.com
Date: <Date>
Authorization: <AuthorizationString>
Content-Type: text/plain
Content-Length: <ContentLength>
x-bce-storage-class: <StorageClass>
```

* ### 请求头域

<table>
<thead>
<tr>
<th style="width:140px">名称</th>
<th style="width:70px">类型</th>
<th>描述</th>
<th style="width:100px">是否必需</th>
</tr>
</thead>
<tbody>
<tr>
<td>x-bce-storage-class</td>
<td>String</td>
<td>指定BOS的对象的存储类型，目前支持<code class="language-text">STANDARD</code>、<code class="language-text">STANDARD_IA</code>、<code class="language-text">COLD</code>和<code class="language-text">ARCHIVE</code>，默认为<code class="language-text">STANDARD</code>；如果是多AZ类型bucket，<code class="language-text">MAZ_STANDARD_IA</code>代表多AZ低频存储，不指定时默认是<code class="language-text">MAZ_STANDARD</code>多AZ标准存储类型，不能是其它取值。</td>
<td>否</td>
</tr>
<tr>
<td>x-bce-acl</td>
<td>String</td>
<td>CannedACL支持的header，用户设置Object的权限，取值为private或public-read。</td>
<td>否</td>
</tr>
<tr>
<td>x-bce-grant-read</td>
<td>String</td>
<td>CannedACL支持的header，用户设置Object的读权限。支持多个id，以逗号分隔</td>
<td>否</td>
</tr>
<tr>
<td>x-bce-grant-full-control</td>
<td>String</td>
<td>CannedACL支持的header，用户设置Object的FULL_CONTROL权限。支持多个id，以逗号分隔</td>
<td>否</td>

</tr>
<tr>
<td>x-bce-server-side-encryption</td>
<td>String</td>
<td>服务端加密算法，当前支持AES256和SM4加密。</td>
<td>否</td>
</tr>
<tr>
<td>Cache-Control</td>
<td>String</td>
<td>下载Object的Cache设置，常见的可取值为<code class="language-text">private</code>、<code class="language-text">no-cache</code>、<code class="language-text">max-age</code>、<code class="language-text">must-revalidate</code></td>
<td>否</td>
</tr>
<tr>
<td>Content-Disposition</td>
<td>String</td>
<td>设置浏览器是否下载，可取值为<code class="language-text">inline</code>、<code class="language-text">attachment</code>; filename="download.txt"</td>
<td>否</td>
</tr>
<tr>
<td>Expires</td>
<td>String</td>
<td>设置下载Object时的缓存失效时间</td>
<td>否</td>
</tr>
<tr>
<td>x-bce-copy-source</td>
<td>String</td>
<td>设置三步上传的源，bucket/object</td>
<td>否</td>
</tr>
</tbody>
</table>

* ### 请求参数

名称	类型	参数位置	描述	是否必需
uploads	String	Query参数	分块上传的请求	是

响应（Response）

* ### 响应头域

  无

* ### 响应元素

名称	类型	描述
bucket	String	Bucket名称
key	String	Object名称
uploadId	String	全局唯一ID，用于标识此次MultiUpload操作

  > **注意事项**
  >
  > 1. 使用MultipartUpload上传的Object，如不指定Content-Type，BOS会自动识别设置合适的Content-Type，若无法识别则默认为`application/octet-stream`。
  > 2. InitiateMultipartUpload获取的UploadId将用于MultiUpload的后续2步操作，也可以用此UploadId来查询整个MultiUpload的进度和中断此次MultiUpload操作。
  > 3. 获取的UploadId 将用于后续的UploadPartCopy,且copy的源是超过5G的append object，则在init的时候需要设置x-bce-copy-source。copy的源是append object，只能针对超过5G的，如果不足5G不支持。

示例

* ### 标准存储的请求示例

  ```
  POST /ObjectName?uploads
  Host: BucketName.bj.bcebos.com
  Date: Wed, 06 Apr 2016 06:34:40 GMT
  Authorization: AuthorizationString
  Content-Length: 0
  ```

* ### 低频/冷存储的请求示例

  ```
  POST /ObjectName?uploads
  Host: BucketName.bj.bcebos.com
  Date: Wed, 06 Apr 2016 06:34:40 GMT
  Authorization: AuthorizationString
  Content-Length: 0
  x-bce-storage-class: STANDARD_IA
  ```

* ### 响应示例

  ```
  HTTP/1.1 200 OK
  x-bce-request-id: 4db2b34d-654d-4d8a-b49b-3049ca786409
  Date:  Wed, 06 Apr 2016 06:34:40 GMT
  Content-Length: 197
  Connection: keep-alive
  Server: BceBos

  {
      "bucket": "BucketName",
      "key":"ObjectName",
      "uploadId": "a44cc9bab11cbd156984767aad637851"
  }
  ```

```

### UploadPart

###### 接口描述

在调用 InitiateMultipartUpload 获取 UploadId 后，我们需要用 UploadPart 命令来上传 Object 拆分后的数据（Part）。为了标识各个 Part 在 Object 的相对位置，在 UploadPart 需要指定一个 partNumber 参数，partNumber 的取值范围是 1 - 10000。

BOS Part 的大小有一定限制，除最后一个分片外，单个分片最小支持 100 KB，最大支持 5 GB，且整个 Object 大小不超过 48.8 TB。

###### 请求（Request）

* ### 请求语法

  ```
  PUT /<ObjectName>?partNumber=PartNumber&uploadId=UploadId HTTP/1.1
  Host: <BucketName>.bj.bcebos.com
  Date: <Date>
  Authorization: <AuthorizationString>
  Content-Length: <ContentLength>
  ```

* ### 请求头域

  名字 | 类型 | 描述
  --- |--- | ---
  Content-MD5 | String | RFC2616定义的HTTP请求内容的MD5摘要，可以通过携带该字段来验证保存在BOS侧的文件和用户预期的文件是否一致。
  x-bce-content-sha256 | String | 通过携带该字段来验证保存在BOS侧的文件和用户预期的文件是否一致，sha256的校验准确性更高。所传数据的sha256值必须与此匹配，否则UploadPart失败。
  | x-bce-content-crc32 | String | 上传object的CRC32值(IEEE算法)。
  | x-bce-content-crc32c | String | 上传object的CRC32C值(Castagnoli算法)。
  | x-bce-content-crc32c-flag | String | 是否计算CRC32C，仅取值为"true"时有效。

* ### 请求参数

  名称 | 类型 | 参数位置| 描述 | 是否必需
  --- | --- | --- | --- |---
  partNumber | Int | Query参数 | 本次上传的part在object的相对位置 | 是
  uploadId | String | Query参数 | InitiateMultipartUpload获取的UploadId | 是

###### 响应（Response）

* ### 响应元素

  无

* ### 响应头域

  名称 | 类型 | 描述
  --- | --- | ---
  ETag | String | 每个上传分块的ETag
  其他 |-| 其他RFC2616相关的Header字段

  > **注意事项**
  >
  > UploadPart会返回本次Part的eTag，在MultipartUpload的第三步中需要此eTag，也建议用户用eTag验证上传数据的正确性。

###### 示例

* ### 请求示例

  ```
  PUT /ObjectName?partNumber=PartNumber&uploadId=UploadId HTTP/1.1
  Host: BucketName.bj.bcebos.com
  Date: Wed, 06 Apr 2016 06:34:40 GMT
  Authorization: AuthorizationString
  Content-length:2794
  Content-Type:text/plain

  [2794 bytes of object data]
  ```

* ### 响应示例

  ```
  HTTP/1.1 200 OK
  x-bce-request-id: 4db2b34d-654d-4d8a-b49b-3049ca786409
  Date: Wed, 06 Apr 2016 06:34:40 GMT
  ETag: "b54357faf0632cce46e942fa68356b38"
  Content-Length: 0
  Connection: keep-alive
  Server: BceBos

  ```

UploadPartCopy

接口描述

在调用 InitiateMultipartUpload 获取 UploadId 后，我们需要用 UploadPartCopy 命令来复制 Object 拆分后的数据分片（即 Part）。为了标识各个 Part 在 Object 的相对位置，在 UploadPartCopy 需要指定一个 partNumber 参数，partNumber 的取值范围是 1-10000，即每次三步复制最多可以有 10000 个part，最少 1 个 part。

> **说明：**
>
> - 每次 Copy Part 最多可以复制 5G Bytes 的数据。
> - 三步复制得到的目的 Object 的 Meta 信息，不会从源 Object 复制而来，需要用户在 CompleteMultipartUpload 时重新上传，或者完成之后，通过[CopyObject接口](BOS/API参考/Object相关接口/基础操
  ```

作/CopyObject.md)复制自身来更新Meta。
> - UploadPartCopy时，不同的part可能来自不同的Object，系统并不对此进行限制。
> - 目前支持跨区域文件复制，即复制文件所在的源Bucket和目标Bucket可以不在同一region(目前只支持从其它Region向本Region复制数据)。当进行跨区域文件复制时，复制产生的流量会收取跨区域流量费。跨区域收费标准参见[产品定价](BOS/产品定价/产品价格/按需计费方式.md)。
> - 三步复制不适用于Append Object，Append Object可以使用普通复制方法来完成备份和meta更新。
> - 如果复制的源Object是归档存储类型的，需要首先调用[RestoreObject接口](BOS/API参考/Object相关接口/基础操作/RestoreObject.md)取回归档存储类型的Object。

请求（Request）

* ### 请求语法

```
PUT /<ObjectName>?uploadId=UploadId&partNumber=PartNumber HTTP/1.1
Host: <BucketName>.bj.bcebos.com
Date: <Date>
Authorization: <AuthorizationString>
Content-Type: text/plain
Content-Length: <ContentLength>
x-bce-copy-source: /SrcBucket/SrcObject
x-bce-copy-source-range: bytes=0-9
```

* ### 请求头域

<table>
<thead>
<tr>
<th style="width:160px">header</th>
<th style="width:100px">是否必填</th>
<th>描述</th>
</tr>
</thead>
<tbody>
<tr>
<td>x-bce-copy-source</td>
<td>是</td>
<td>复制源文件，比如/SrcBucket/SrcObject</td>
</tr>
<tr>
<td>x-bce-copy-source-range</td>
<td>否</td>
<td>复制源文件的字节范围,必须使用 bytes=first-last 的格式，比如 bytes=0-9，代表复制前 10 个字节。如果不提供此 header，则复制整个文件。除最后一个 Part 外，每个 Part 分片需要大于等于 100 KB，小于等于 5GB，且整个 Object 大小不超过 48.8 TB。</td>
</tr>
<tr>
<td>x-bce-copy-source-if-match</td>
<td>否</td>
<td>如果源文件的etag值与此header提供的相同，则复制，否则不复制</td>
</tr>
<tr>
<td>x-bce-copy-source-if-none-match</td>
<td>否</td>
<td>如果源文件的etag值与此header提供的不相同，则复制，否则不复制</td>
</tr>
<tr>
<td>x-bce-copy-source-if-unmodifed-since</td>
<td>否</td>
<td>如果源文件在此header提供的时间后未曾修改，则复制，否则不复制</td>
</tr>
<tr>
<td>x-bce-copy-source-if-modifed-since</td>
<td>否</td>
<td>如果源文件在此header提供的时间后曾修改过，则复制，否则不复制</td>
</tr>
</tbody>
</table>

> **说明：**
>
> 各字段的格式与[CopyObject接口](BOS/API参考/Object相关接口/基础操作/CopyObject.md)对应字段相同。
> 假设Object内容为"0123456789"，长度为10。x-bce-copy-source-range举例如下：
>
> - range: 0-3，拷贝"0123"；
> - range: 4-，从下标为4的开始拷贝到文件结束即拷贝"456789"；
> - range: -4，拷贝末尾4个即"6789"。

* ### 请求参数

<table>
<thead>
<tr>
<th>名称</th>
<th style="width:70px">类型</th>
<th style="width:100px">参数位置</th>
<th>描述</th>
<th style="width:100px">是否必需</th>
</tr>
</thead>
<tbody>
<tr>
<td>partNumber</td>
<td>Int</td>
<td>Query参数</td>
<td>此part在目的Object中的序号。partNum取值范围 <code class="language-text">1-10000</code>，一次MultiPart的PartNumber要求必须严格有序，比如有3个Part，PartNumber可以是1，3，5。</td>
<td>是</td>
</tr>
<tr>
<td>uploadId</td>

<td>uploadId</td>
<td>String</td>
<td>Query参数</td>
<td>InitiateMultipartUpload获取的UploadId</td>
<td>是</td>
</tr>
</tbody>
</table>

响应（Response）

* ### 响应元素

名称	描述
ETag	复制目的Object的ETag
lastModified	目的Object的修改时间

> **注意事项**
>
> 1. UploadPartCopy会返回本次Part的ETag，在MultipartUpload的第三步中需要此ETag，也建议用户用ETag验证上传数据的正确性。
> 2. 为了保持复制过程中的http连接，UploadPartCopy接口的http结果可能使用Transfer-Encoded: Chunked编码方式。
> 3. UploadPartCopy过程中，如果发生服务器端错误，http status code可能返回2XX但是复制失败，复制结果请根据http body中的json判定。

示例

* ### 请求示例

```
PUT /ObjectName?uploadId=a44cc9bab11cbd156984767aad637851&partNumber=1 HTTP/1.1
Host: BucketName.bj.bcebos.com
Date: <Date>
Authorization: <AuthorizationString>
Content-Type: text/plain
Content-Length: 0
x-bce-copy-source: /SrcBucket/SrcObject
x-bce-copy-source-range: bytes=0-9
```

* ### 响应示例

```
HTTP/1.1 200 OK
Date: Thu, 12 May 2016 09:14:32 GMT
Content-Type: application/json; charset=utf-8
Connection: keep-alive
Server: BceBos
x-bce-debug-id: MTAuNjMuMTIzLjI3OlRodSwgMTIgTWF5IDIwMTYgMTc6MTQ6MzlgQ1NUOjg3MjkzODMwMA==
x-bce-request-id: bb90cc9c-2b80-462c-87a4-095e610c9a2f
Transfer-Encoding: chunked
{
    "lastModified":"2016-05-12T09:14:32Z",
    "eTag":"67b92a7c2a9b9c1809a6ae3295dcc127"
}
```

拷贝失败, 需要根据返回的json判断

```
HTTP/1.1 200 OK
Date: Thu, 12 May 2016 09:14:32 GMT
Content-Type: application/json; charset=utf-8
Connection: keep-alive
Server: BceBos
x-bce-debug-id: MTAuNjMuMTIzLjI3OlRodSwgMTIgTWF5IDIwMTYgMTc6MTQ6MzlgQ1NUOjg3MjkzODMwMA==
x-bce-request-id: bb90cc9c-2b80-462c-87a4-095e610c9a2f
Transfer-Encoding: chunked
{
    "code":"InternalError",
    "message":"We encountered an internal error. Please try again.",
    "requestId":"52454655-5345-4420-4259-204e47494e58"
}
```

CompleteMultipartUpload

接口描述

当请求者用UploadPart将所有的Part都上传完成后，需要用此CompleteMultipartUpload命令完成整个MultipartUpload操作。此命令需要请求提供有效的Part列表，包含part的PartNumber和eTag。BOS收到此命令后会检查数据，然后把所有的Part组合成一个Object。

请求（Request）

* ### 请求语法

```
POST /<ObjectName>?uploadId=UploadId HTTP/1.1
Host: <BucketName>.bj.bcebos.com
Date: <Date>
Authorization: <AuthorizationString>
Content-Length: <ContentLength>
Content-Type: text/plain
```

* ### 请求头域

名称	类型	描述	是否必需
Content-Length	Long Int	头域，JSON数据的长度	是

| x-bce-meta-* | String | 用户自定义的meta | 否 |

* ### 请求主体

名称	类型	描述	是否必需
parts	-	json body根节点	是
+ partNumber	Int	此part在目的Object中的序号。partNum取值范围 `1-10000`，一次MultiPart的PartNumber要求必须严格有序，比如有3个Part，PartNumber可以是1，3，5。	是
+ eTag	String	Object的HTTP协议实体标签	是

响应（Response）

* ### 响应头域

无特殊头域

* ### 响应元素

名称	类型	描述
bucket	String	此Object所属的Bucket
eTag	String	Object的HTTP协议实体标签
key	String	Object名称
location	String	此Object的url

> **注意事项**
>
> 1. CompleteMultipartUpload的请求Body最大为1MB。
> 2. 一次MultiPart的PartNumber可以是不连续的，比如1, 3, 5。

示例

* ### 请求示例

```
POST /ObjectName?uploadId=UploadId HTTP/1.1
Host: BucketName.bj.bcebos.com
Date: Wed, 06 Apr 2016 06:34:40 GMT
Authorization: AuthorizationString
Content-Length: 11434
Content-Type: text/plain

{
 "parts":[
  {
   "partNumber":1,
   "eTag":"a54357aff0632cce46d942af68356b38"
  },
  {
   "partNumber":2,
   "eTag":"0c78aef83f66abc1fa1e8477f296d394"
  },
  {
   "partNumber":3,
   "eTag":"acbd18db4cc2f85cedef654fccc4a4d8"
  }
 ]
}
```

* ### 响应示例

```
HTTP/1.1 200 OK
x-bce-request-id: 4db2b34d-654d-4d8a-b49b-3049ca786409
Date: Wed, 06 Apr 2016 06:34:40 GMT
Connection: close
Server: BceBos

{
 "location":"http://bj.bcebos.com/BucketName/ObjectName",
 "bucket":"BucketName",
 "key":"object",
 "eTag":"3858f62230ac3c915f300c664312c11f"
}
```

AbortMultipartUpload

接口描述

用户可以使用此接口来中断某个MultipartUpload请求，BOS收到此命令后，将会清除已上传的数据。

请求（Request）

- 请求语法：

```
DELETE /<ObjectKey>?uploadId=UploadId HTTP/1.1
Host: <BucketName>.bj.bcebos.com
Date: <Date>
Authorization: <AuthorizationString>
Content-Length: <ContentLength>
```

- 请求头域

无特殊要求头域

- 请求参数

名称	类型	参数位置	描述	是否必需
uploadId	String	Query参数	此次MultipartUpload的ID	是

响应（Response）

- 响应头域

无特殊头域

- 响应元素

无

> **注意事项**
>
> 中止一个Multipart Upload事件时，如果其所属的某些Part仍然在上传，那么这次中止操作将无法删除这些Part。所以如果存在并发访问的情况，为了彻底释放BOS上的空间，需要调用几次Abort Multipart Upload接口。

示例

- 请求示例

```
DELETE /ObjectName?uploadId=a44cc9bab11cbd156984767aad637851 HTTP/1.1
Host: BucketName.bj.bcebos.com
Date: Wed, 06 Apr 2016 06:34:40 GMT
Authorization: AuthorizationString
```

- 响应示例

```
HTTP/1.1 204 No Content
x-bce-request-id: 4db2b34d-654d-4d8a-b49b-3049ca786409
Date:  Wed, 06 Apr 2016 06:34:40 GMT
Content-Length: 0
Connection: keep-alive
Server: BceBos
```

ListParts

接口描述

此命令用于列出用户指定UploadId所属的所有已经上传成功的Part，用户可以通过此命令查看当前的进度。

请求（Request）

* ### 请求语法

GET /?uploadId=UploadId&maxParts=MaxParts&partNumberMarker=PartNumberMarker HTTP/1.1 Host: .bj.bcebos.com Date: Authorization:

* ### 请求头域

无特殊头域

* ### 请求参数

<table>
<thead>
<tr>
<th style="width:100px">名称</th>
<th style="width:70px">类型</th>
<th>参数位置</th>
<th>描述</th>
<th style="width:100px">是否必需</th>
</tr>
</thead>
<tbody>
<tr>
<td>maxParts</td>
<td>Int</td>
<td>Query参数</td>
<td>BOS一次最多返回的part数目，默认1000，最大1000</td>
<td>否</td>
</tr>
<tr>
<td>partNumberMarker</td>
<td>Int</td>
<td>Query参数</td>
<td>按照partNumber排序，本次请求的起始part从此partNumber的下一个开始返回</td>
<td>否</td>
</tr>
<tr>
<td>uploadId</td>
<td>String</td>
<td>Query参数</td>
<td>此次MultipartUpload的ID</td>
<td>是</td>
</tr>
</tbody>
</table>

响应（Response）

* ### 响应头域

 无特殊头域

* ### 响应元素

<table>
<thead>
<tr>
<th style="width:100px">名称</th>
<th style="width:70px">类型</th>
<th>描述</th>
</tr>
</thead>
<tbody>
<tr>
<td>bucket</td>
<td>String</td>
<td>所属Bucket名称</td>
</tr>
<tr>
<td>key</td>
<td>String</td>
<td>Object名称</td>
</tr>
<tr>
<td>uploadId</td>
<td>String</td>
<td>请求指定的UploadId</td>
</tr>
<tr>
<td>initiated</td>
<td>String</td>
<td>multipartUpload的创建时间</td>
</tr>
<tr>
<td>owner</td>
<td>Container</td>
<td>此object所属的用户信息.</td>
</tr>
<tr>
<td>+id</td>
<td>String</td>
<td>用户ID</td>
</tr>
<tr>
<td>+displayName</td>
<td>String</td>
<td>用户名</td>
</tr>
<tr>
<td>storageClass</td>
<td>String</td>
<td>Object的存储类型，低频存储返回<code class="language-text">STANDARD_IA</code>，冷存储返回<code class="language-text">COLD</code>，标准存储返回<code class="language-text">STANDARD</code>，归档存储返回<code class="language-text">ARCHIVE</code>；如果是多AZ类型bucket，多AZ低频存储返回<code class="language-text">MAZ_STANDARD_IA</code>，多AZ标准存储返回<code class="language-text">MAZ_STANDARD</code>。</td>
</tr>
<tr>
<td>partNumberMarker</td>
<td>Int</td>
<td>请求指定的本次part Number起始位置</td>
</tr>
<tr>
<td>nextPartNumberMarker</td>
<td>Int</td>
<td>本次请求返回的最后一条记录的partNumber，可以作为下一次请求的PartNumberMarker</td>
</tr>
<tr>
<td>maxParts</td>
<td>Int</td>
<td>请求指定的本次最多返回的part数量</td>
</tr>
<tr>
<td>isTruncated</td>
<td>Bool</td>
<td>标明是否本次返回的List Part结果列表被截断。 true表示本次没有返回全部结果； false表示本次已经返回了全部结果</td>
</tr>
<tr>
<td>parts</td>
<td>Container</td>
<td>一个part的容器</td>
</tr>
<tr>
<td>+partNumber</td>
<td>Int</td>
<td>该part的标识</td>
</tr>
<tr>
<td>+lastModified</td>
<td>DATE</td>
<td>该part的上传时间</td>
</tr>
<tr>
<td>+ETag</td>
<td>String</td>
<td>每个上传分块的ETag</td>
</tr>
<tr>
<td>+size</td>

<td>partSize</td>
<td>Int</td>
<td>该part大小</td>
</tr>
</tbody>
</table>

> **注意事项**
>
> 1. BOS按照PartNumber升序排序。
> 2. 由于网络传输可能出错，所以不推荐用ListParts出来的结果生成最后CompleteMultipartUpload的Part列表。

示例

* ### 请求示例

```
GET /ObjectName?uploadId=a44cc9bab11cbd156984767aad637851&maxParts=2&partNumberMarker=1
Host: BucketName.bj.bcebos.com
Date: Wed, 06 Apr 2016 06:34:40 GMT
Authorization: AuthorizationString
```

* ### 响应示例

```
HTTP/1.1 200 OK
x-bce-request-id: 4db2b34d-654d-4d8a-b49b-3049ca786409
Date: Wed, 06 Apr 2016 06:34:40 GMT
Content-Length: 985
Connection: keep-alive
Server: BceBos

{
    "bucket":"BucketName",
    "key":"object",
    "uploadId":"a44cc9bab11cbd156984767aad637851",
    "initiated":"2010-11-10T20:48:33Z",
    "owner":{
        "id":"75aa570f8e7faeebf76c078efc7c6caea54ba06a",
        "displayName":"someName"
    },
    "storageClass":"STANDARD",
    "partNumberMarker":1,
    "nextPartNumberMarker":3,
    "maxParts":2,
    "isTruncated":true,
    "parts":[
        {
            "partNumber":2,
            "lastModified":"2010-11-10T20:48:34Z",
            "ETag":"7778aef83f66abc1fa1e8477f296d394",
            "size":10485760
        },
        {
            "partNumber":3,
            "lastModified":"2010-11-10T20:48:33Z",
            "ETag":"aaaa18db4cc2f85cedef654fccc4a4x8",
            "size":10485760
        }
    ]
}
```

ListMultipartUploads

接口描述

此命令用于列出指定Bucket下面的所有未执行完成的Multipart Upload。“未执行完”是指完成了InitMultipartUpload，但是还没有调用CompleteMultipartUpload或AbortMultipartUpload的Multipart Upload。每次BOS最多返回1000个Multipart Upload，BOS支持prefix和delimiter过滤。

请求（Request）

* ### 请求语法

```
GET /?uploads HTTP/1.1
Host: <BucketName>.bj.bcebos.com
Date: <Date>
Authorization: <Authorization_String>
```

* ### 请求头域

无特殊要求头域

* ### 请求参数

<table><thead>
<tr>
<th>名称</th>
<th>类型</th>
<th>参数位置</th>
<th>描述</th>
<th>是否必需</th>
</tr>
</thead>
<tbody>
<tr>

<td>delimiter </td>
<td>String </td>
<td>Query参数 </td>
<td>分隔符;主要应此项实现list文件夹的逻辑 </td>
<td>否 </td>
</tr>
<tr>
<td>keyMarker </td>
<td>String </td>
<td>Query参数 </td>
<td>
与uploadIdMarker参数配合使用，用于指定返回结果的起始位置。

如果未设置uploadIdMarker参数，Object按照字典序排序后，本次从keyMarker的后面的一条开始返回。
如果设置了uploadIdMarker参数，查询结果中包含：

所有Object名称的字典序大于keyMarker参数值的Multipart Upload事件。
Object名称等于keyMarker参数值，但是UploadId比uploadIdMarker参数值大的Multipart Upload事件。

</td>
<td>否 </td>
</tr>
<tr>
<td>uploadIdMarker </td>
<td>String </td>
<td>Query参数 </td>
<td>与keyMarker参数配合使用，用于指定返回结果的起始位置。

如果未设置keyMarker参数，则BOS会忽略uploadIdMarker参数。
如果设置了keyMarker参数，查询结果中包含：

所有Object名称的字典序大于keyMarker参数值的Multipart Upload事件。
Object名称等于keyMarker参数值，但是UploadId比uploadIdMarker参数值大的Multipart Upload事件。

</td>
<td>否 </td>
</tr>
<tr>
<td>maxUploads </td>
<td>Int </td>
<td>Query参数 </td>
<td>本次请求返回Multipart Uploads的最大数目,默认1000，最大1000 </td>
<td>否 </td>
</tr>
<tr>
<td>prefix </td>
<td>String </td>
<td>Query参数 </td>
<td>key前缀，限定返回的object key必须以此为前缀 </td>
<td>否 </td>
</tr>
<tr>
<td>uploads </td>
<td>String </td>
<td>Query参数 </td>
<td>标明请求是 ListMultipartUploads </td>
<td>是 </td>
</tr>
</tbody>
</table>

响应（Response）

* ### 响应头域

无特殊头域

* ### 响应元素

<table>
<thead>
<tr>
<th style="width:120px">名称</th>
<th style="width:70px">类型</th>
<th>描述</th>
</tr>
</thead>
<tbody>
<tr>
<td>bucket</td>
<td>String</td>
<td>所属Bucket名称</td>
</tr>
<tr>
<td>commonPrefixes</td>
<td>-</td>
<td>如果在请求的时候指定了delimiter，将返回此项。BOS把匹配到的Object名称
按照一定规则（从preifx到第一个delimiter）截取，截取的字符串去重作为CommonPrefixes的数据返回</td>
</tr>
<tr>
<td>delimiter</td>
<td>String</td>
<td>返回请求中的delimiter值</td>
</tr>
<tr>
<td>prefix</td>
<td>String</td>

<td>object前缀</td>
</tr>
<tr>
<td>isTruncated</td>
<td>Bool</td>
<td>标明是否本次是否没有返回所有的数据。
<code class="language-text">true</code>表示本次没有返回全部结果； <code class="language-text">false</code>表示本次已经返回了全部结果</td>
</tr>
<tr>
<td>keyMarker</td>
<td>String</td>
<td>本次请求指定的object的起始位置</td>
</tr>
<tr>
<td>uploadIdMarker</td>
<td>String</td>
<td>本次请求指定的uploadId的起始位置。</td>
</tr>
<tr>
<td>maxUploads</td>
<td>Int</td>
<td>请求指定的本次返回的Multipart Uploads的最大数目</td>
</tr>
<tr>
<td>nextKeyMarker</td>
<td>String</td>
<td>如果本次没有返回全部结果，响应请求中将包含nextKeyMarker元素，用于表示接下来请求的keyMarker值。</td>
</tr>
<tr>
<td>nextUploadIdMarker</td>
<td>String</td>
<td>如果本次没有返回全部结果，响应请求中将包含nextUploadIdMarker元素，用于表示接下来请求的uploadIdMarker值</td>
</tr>
<tr>
<td>uploads</td>
<td>Container</td>
<td>保存一个MultipartUpload的容器</td>
</tr>
<tr>
<td>+key</td>
<td>String</td>
<td>Object名称</td>
</tr>
<tr>
<td>+uploadId</td>
<td>String</td>
<td>MultipartUpload的ID</td>
</tr>
<tr>
<td>+owner</td>
<td>Container</td>
<td>Object所属的用户信息</td>
</tr>
<tr>
<td>++displayName</td>
<td>String</td>
<td>用户名</td>
</tr>
<tr>
<td>++id</td>
<td>String</td>
<td>用户ID</td>
</tr>
<tr>
<td>+initiated</td>
<td>Date</td>
<td>本次MultipartUpload开始时间</td>
</tr>
<tr>
<td>+storageClass</td>
<td>String</td>
<td>Object的存储类型，低频存储返回<code class="language-text">STANDARD_IA</code>，冷存储返回<code class="language-text">COLD</code>，标准存储返回<code class="language-text">STANDARD</code>，归档类型返回<code class="language-text">ARCHIVE</code>；如果是多AZ类型bucket，多AZ低频存储返回<code class="language-text">MAZ_STANDARD_IA</code>，多AZ标准存储返回<code class="language-text">MAZ_STANDARD</code>。</td>
</tr>
</tbody>
</table>

> **注意事项**：此处的Delimiter跟ListObjects的类似，可以参考ListObjects的接口说明。

示例

* ### 请求示例

```
GET /?uploads HTTP/1.1
Host: BucketName.bj.bcebos.com
Date: Wed, 06 Apr 2016 06:34:40 GMT
Authorization: AuthorizationString
```

* ### 响应示例（JSON）

```
HTTP/1.1 200 OK
x-bce-request-id: 4db2b34d-654d-4d8a-b49b-3049ca786409
Date: Wed, 06 Apr 2016 06:34:40 GMT
Content-Length: 1330
```

```
Content-Length: 1330
Connection: keep-alive
Server: BceBos

{
    "bucket": "bucket",
    "keyMarker": "",
    "nextKeyMarker": "my-movie.m2ts",
    "nextUploadMarker": "c41cc9aad11cbd637851767bab156984",
    "maxUploads": 3,
    "isTruncated": true,
    "uploads": [
        {
            "key": "my-divisor",
            "uploadId": "a44cc9bab11bdc157676984aad851637",
            "owner": {
                "id": "75aa57f09aa0c8caeab4aeebf76c078efc7c6caea54ba06a",
                "displayName": "OwnerDisplayName"
            },
            "initiated": "2010-11-10T20:48:33Z",
            "storageClass": "STANDARD_IA"
        },
        {
            "key": "my-movie",
            "uploadId": "b44cc9bab11cbd156984767aad637851",
            "owner": {
                "id": "b1d16700c70b0b05597d7acd6a3f92be",
                "displayName": "OwnerDisplayName"
            },
            "initiated": "2010-11-10T20:48:33Z",
            "storageClass": "STANDARD"
        },
        {
            "key": "my-movie.m2ts",
            "uploadId": "b41cc9aad11cbd637851767bab156984",
            "owner": {
                "id": "b1d16700c70b0b05597d7acd6a3f92be",
                "displayName": "OwnerDisplayName"
            },
            "initiated": "2010-11-10T10:49:33Z",
            "storageClass": "STANDARD_IA"
        },
        {
            "key": "my-movie.m2ts",
            "uploadId": "c41cc9aad11cbd637851767bab156984",
            "owner": {
                "id": "b1d16700c70b0b05597d7acd6a3f92be",
                "displayName": "OwnerDisplayName"
            },
            "initiated": "2010-11-10T20:49:33Z",
            "storageClass": "STANDARD_IA"
        }
    ]
}
```

上传回调

上传同步回调

接口描述
此接口用于BOS在上传文件（Object）时提供**同步回调（Callback）**给应用服务器。应用服务器指的是用户提供的回调地址对应的服务器。

支持的上传类操作如下：

* 普通上传（PutObject）
* 完成分片上传（CompleteMultipartUpload）

请求语法（Request）

PUT / HTTP/1.1 Host: .bj.bcebos.com Date: Authorization: Content-Type: text/plain Content-Length: <Content_Length> x-bce-process: <callback_command>

- 请求参数
见"请求头域"部分说明。

- 请求头域

名称	类型	描述	是否必须
x-bce-process	string	使用x-bce-process参数存储回调的命令，命令有两种形式，一是附在querystring中，二是附在header中。具体使用见下文的"x-bce-process使用说明"部分。	否

x-bce-process命令的形式

x-bce-process: callback/callback,${key}*${value},${key}*${value}

x-bce-process参数说明：

参数名称	缩写	类型	取值范围	描述	是否必选
urls	u	string	-	经过url安全的Base64编码后的地址列表，最多3个地址，具体内容为json数组`["http://test1.com/callback", "http://test2.com/callback"]`	是
mode	m	string	`sync`	当前只能为同步。不设置时默认为`sync`	否
vars	v	string	-	任意自定义参数，回调时透传，长度不超过1024字节,建议数据进行base64编码。	否
encrypt	e	string	`config`	回调签名的加密类型。可取值如下：<url> `config`：表示通过创建事件通知规则配置了回调请求的签名密钥。 	否
key	k	string	-	回调签名的key，可取值如下：<ur> 如果encrypt为"config"，表示通过创建事件通知规则配置了回调请求的签名密钥，值为对应的事件通知规则的id。 </url>	否

x-bce-process使用说明:
* x-bce-process回调用的子命令为callback，参数使用缩写;
* 如果用户在一次请求中提供了多条url回调地址，当第一次回调成功则停止并返回，后续url不再做回调。
* 回调的信息放在JSON Body中返回。
* 参数"encrypt"、"key"只在需要设置回调签名的时候使用。具体见下文介绍。

响应（response）
- 响应元素

响应元素同PutObject、CompleteMultipartUpload 接口的响应元素。

如果同步回调成功且用户的应用回调服务器有响应内容，响应元素增加"callback"字段来存储应用服务器响应内容，并返回客户端。

名称	类型	描述
callback	object	包含用户应用服务器返回内容的对象
+result	string	用户应用服务器的返回内容

回调说明:
* 用户应用服务器返回StatusCode: 200响应认为是回调成功并结束，对于其他错误或不响应，则重试下一个url。
* 如果所有url的回调都失败，则回调失败，同时返回错误码Status Code: 203，表示文件已经成功上传到了BOS，但回调失败。回调失败只是表示BOS没有收到预期的回调响应，不代表应用服务器没有收到回调请求。
* 回调成功时，应用服务器的返回值不能超过1MB，否则认为回调失败，错误信息为"PayloadTooLarge"。
* 对于CompleteMultipartUpload原有接口就有响应Body的情况，callbackResult会附加到原有返回的JSON中。
* 回调失败时，code和message中会附带失败的原因。

回调请求格式
上传同步回调应用服务的请求格式与bucket事件通知的回调格式一致。

例如,用户计划回调的地址为`["https://www.test.com/callback"]`，经过base64编码后的取值为"WyJodHRwczovL3d3dy50ZXN0LmNvbS9jYWxsYmFjayJd"。用户计划携带自定义参数内容为`{"key1":"value1"}`,经过base64编码后为"eyJrZXkxIjoidmFsdWUxIn0=".

假设用户上传请求时header中携带了如下回调命令：

x-bce-process: callback/callback,u_WyJodHRwczovL3d3dy50ZXN0LmNvbS9jYWxsYmFjayJd,v_eyJrZXkxIjoidmFsdWUxIn0=

则向用户应用服务器发起url回调请求示例如下所示，其中签名"Authorization"是可选的。

POST /<callback_url> HTTP/1.1 Host: <User_Host> Date: Content-Type: application/json; charset=utf-8 Authorization: Content-Length: <Content_Length> { "events": [{ "version": "1.0", "eventId": "2a513199-bbb9-4ac7-b12a-60213c26810d", "eventOrigin": "bos:callback", "eventTime": "2023-09-05T02:28:49Z", "eventSource": "bos:callback", "eventType": "PutObject", "eventFrom": "Client", "content": { "userId": "abcdefgh12345678a4ffa3b2ececa17f", "ownerId": "abcdefgh12345678a4ffa3b2ececa17f", "accessKeyId": "*", "domain": "bj.bcebos.com", "bucket": "bucket-test", "object": "object-test", "etag": "977b9623a15e520c663ac5ff6647e881", "contentType": "application/octet-stream", "filesize": 24414, "lastModified": "2018-09-05T02:28:49Z", "storageClass": "STANDARD", "xVars": "eyJrZXkxIjoidmFsdWUxIn0=" } }] }

回调签名（可选）
用户设置x-bce-process同步请求回调参数之后，BOS服务端将会按照用户设置的回调URL发送POST回调请求给用户的应用服务器。应用服务器收到回调请求之后，如果希望验证回调请求确实是由BOS发起的话，可以通过在回调中带上签名来验证BOS的身份。

回调安全使用百度智能云的IAM签名算法来实现，具体可参考IAM签名算法。

签名过程分为三步：

（1）提前通过创建事件通知规则设置密钥。用户可以在通过控制台或者sdk创建。

例如创建事件通知规则的id为"callback1"，其中key为用户设置密钥，长度为32字符，可取值为数字[0-9]和小写英文字母。

PUT /?notification HTTP/1.1 Host: .bj.bcebos.com Date: Authorization: <Authorization_String> Content-Type: application/json; charset=utf-8 Content-Length: <Content_Length> { "notifications": [{ "id": "callback1", "encryption": { "key": "12345678abcdefgh9d1aff8bc436a2f6" } }] }

（2）上传文件请求中，携带含加密参数的回调命令。
例如用户计划回调的地址为`["https://www.test.com/callback"]`，经过base64编码后的取值为"WyJodHRwczovL3d3dy50ZXN0LmNvbS9jYWxsYmFjayJd"。

x-bce-process: callback/callback,u_WyJodHRwczovL3d3dy50ZXN0LmNvbS9jYWxsYmFjayJd,e_config,k_callback1,

（3）BOS服务端向用户的应用服务器发起带签名的回调。其中计算签名的AK为用户本次请求的AK(如果没有带签名，默认使用"-")，SK为用户在第一步中设置的密钥。生成的签名位于请求头"Authorization"中。

示例
- 请求示例
例如用户计划回调的地址为`["https://www.test.com/callback"]`，经过base64编码后的取值为"WyJodHRwczovL3d3dy50ZXN0LmNvbS9jYWxsYmFjayJd"。

附在Header中：

PUT / HTTP/1.1 Host: .bj.bcebos.com Date: Authorization: Content-Type: text/plain Content-Length: <Content_Length> x-bce-process: callback/callback,u_WyJodHRwczovL3d3dy50ZXN0LmNvbS9jYWxsYmFjayJd

附在query中：

http://.bj.bcebos.com/image.png?x-bce-process=callback/callback,u_WyJodHRwczovL3d3dy50ZXN0LmNvbS9jYWxsYmFjayJd

- PutObject响应示例

回调完成后，将回调的结果附带返回给客户端，其格式如下所示：

```
HTTP/1.1 200 OK Date: Wed, 06 Apr 2016 06:34:40 GMT ETag: "1b2cf535f27731c974343645a3985328" Content-Length: 22 Connection: close Server: BceBos { "callback": { "result": "${callback_result}" } }
```

- CompleteMultipartUpload响应示例

```
HTTP/1.1 200 OK x-bce-request-id: 4db2b34d-654d-4d8a-b49b-3049ca786409 Date: Wed, 06 Apr 2016 06:34:40 GMT Connection: close Server: BceBos {
"location":"http://bj.bcebos.com/BucketName/ObjectName", "bucket":"BucketName", "key":"object", "eTag":"3858f62230ac3c915f300c664312c11f", "callback": { "result": "${callback_result}" } }
```

图像审核服务接口

接口描述

接口描述

此接口用于请求图像审核服务。图像审核服务是百度智能云对外提供的图像智能审核服务，支持对图像的多个维度进行审核，包括色情识别、暴恐识别、政治敏感识别、恶心图识别、广告识别等。

> **说明：**使用本接口需要先在控制台开通图像审核服务，详情请参见[开通图像审核服务](BOS/控制台操作指南/图片审核服务.md)。

请求

- ### 请求语法

```
POST /<ObjectName>?process HTTP/1.1
Host: <BucketName>.bj.bcebos.com
Date: <Date>
Authorization: <AuthorizationString>
Content-Type: application/json; charset=utf-8
Content-Length: <ContentLength>

{
    "action" : {
        "sync" : [{
                "url" : "$(img-censor)",
                "parameters" : "<base64_encode(param)>",

            }
        ]
    }
}
```

- ### 请求参数

- url：固定取值为 `$(img-censor)`，不需要修改。
- parameters：此项值是对图像审核功能参数进行 base64 编码而成。图像审核功能参数依然是个 json 字符串。

  图像审核功能参数基本结构：

  ```
  {
    "antiporn" : {},
    "terror" : {},
    "ocr" : {
        "detect_direction" : "false",
        "language_type" : "CHN_ENG",
        "recognize_granularity" : "big"
    }
  }
  ```

  可以为多个子服务的按需任意组合，也可以为单个子服务如“鉴黄”，此时参数写成

  ```
  {
      "antiporn" : {}
  }
  ```

  图像审核服务支持的子服务列表有：

  - ocr：通用文字识别
  - face：人脸探测
  - antiporn：色情识别
  - politician：政治人物识别
  - terror：暴恐识别
  - public: 公众人物识别
  - disgust: 恶心图识别
  - watermark：水印二维码识别
  - quality：图像质量识别 value 为各底层服务参数，参照河图 OCR 通用文字识别服务入参。

  详细的子服务请求参数及返回参数解释参见下文详细解释。

- ### 请求头域

  无

响应

- ### 响应头域

  无

- ### 响应元素

  参见各子服务的返回参数解释。

示例

- ### 请求示例

{ "antiporn": {}, "ocr": { "detect_direction": "false", "language_type": "CHN_ENG", "recognize_granularity": "big" } }

base64 编码后

eyJhbnRpcG9ybiI6e30sIm9jciI6eyJkZXRlY3RfZGlyZWN0aW9uIjoiZmFsc2UiLCJsYW5ndWFnZV90eXBlIjoiQ0hOX0VOR1lsInJlY29nbml6ZV9ncmFudWxhcml0eSI6ImJpZyJ9fQ==

填充到 parameters 中，发送的请求如下：

POST /?process HTTP/1.1 Host: .bj.bcebos.com Date: Authorization: Content-Type: application/json; charset=utf-8 Content-Length:

{ "action" : { "sync" : [{ "url" : "$(img-censor)", "parameters" :
"eyJhbnRpcG9ybiI6e30sIm9jciI6eyJkZXRlY3RfZGlyZWN0aW9uIjoiZmFsc2UiLCJsYW5ndWFnZV90eXBlIjoiQ0hOX0VOR1lsInJlY29nbml6ZV9ncmFudWxhcml0eSI6ImJpZyJ9fQ==", }] } }

- 成功响应示例

  ```
  HTTP/1.1 200 OK
  Date: Thu, 22 Jun 2017 07:30:56 GMT
  Content-Type: application/json; charset=utf-8
  Content-Length: 237
  Connection: keep-alive
  Server: BceBos
  x-bce-debug-id: MTAuNzUuNzguNDA6VGh1LCAyMiBKdW4gMjAxNyAxNTozMDo1NiBDU1Q6MTg1MDY5NDg3OQ==
  x-bce-request-id: 598f7e18-77fb-424a-bc68-95acb0644076

  {
   "result" : {
    "antiporn" : {
     "result" : [{
       "probability" : 0.000071,
       "class_name" : "色情"
      }, {
       "probability" : 0.000291,
       "class_name" : "性感"
      }, {
       "probability" : 0.999638,
  ```

```
        "class_name" : "正常"
       }
      ],
      "log_id" : 1853911322,
      "result_num" : 3
     },
     "ocr" : {
      "log_id" : 2471272194,
      "words_result_num" : 2,
      "words_result" : [{
       "words" : " TSINGTAO"
      }, {
       "words" : "青島"
      }
     ]
    }
   },
   "log_id" : 149811665151162
  }
```


- 失败返回示例


```
  {
     "log_id": 149319909347709,
     "error_code": 216500,
     "error_msg": "unknown error"
  }
```


  失败error_code解释：

错误码	错误信息	描述
216101	not enough param	参数不足
216102	service not support	输入了不支持的底层服务类型
216200	empty imge	没有图片 Url
216500	unknown error	未知错误
282804	download image error	图片下载失败
282000	logic internal error	业务逻辑层错误

通用文字识别（ocr）

功能描述

用户向服务请求识别某张图中的所有文字。

请求参数

参数	是否必选	类型	可选值范围	说明
language_type	否	string	CHN_ENG、ENG、POR、FRE、GER、ITA、SPA、RUS、JAP、KOR	识别语言类型，默认为 CHN_ENG。可选值包括： - CHN_ENG：中英文混合； - ENG：英文； - POR：葡萄牙语； - FRE：法语； - GER：德语； - ITA：意大利语； - SPA：西班牙语； - RUS：俄语； - JAP：日语；- KOR：韩语
detect_direction	否	boolean	true、false	是否检测图像朝向，默认不检测，即：false。朝向是指输入图像是正常方向、逆时针旋转90/180/270度。可选值包括: - true：检测朝向； - false：不检测朝向。
detect_language	否	string	true、false	是否检测语言，默认不检测。当前支持（中文、英语、日语、韩语）
probability	否	string	true、false	是否返回识别结果中每一行的置信度

返回参数

字段	是否必选	类型	说明
direction	否	int32	图像方向，当 detect_direction=true 时存在。 - -1:未定义， - 0:正向， - 1: 逆时针90度， - 2:逆时针180度， - 3:逆时针270度
log_id	是	uint64	唯一的 log id，用于问题定位
words_result	是	array()	识别结果数组
words_result_num	是	uint32	识别结果数，表示 words_result 的元素个数
+words	否	string	识别结果字符串
probability	否	object	识别结果中每一行的置信度值，包含 average ：行置信度平均值，variance：行置信度方差，min：行置信度最小值

返回示例


```
  {
   "log_id": 2471272194,
   "words_result_num": 2,
   "words_result":
      [
         {"words": " TSINGTAO"},
         {"words": "青島啤酒"}
      ]
  }
```


人脸探测（face）

功能描述

- 检测请求图片中的人脸，返回人脸位置、72 个关键点坐标、及人脸相关属性信息。
- 检测响应速度，与图片中人脸数量相关，人脸数量较多时响应时间会有些许延长。
- 典型应用场景：如**人脸属性分析**，**基于人脸关键点的加工分析**，**人脸营销活动**等。
- 五官位置会标记具体坐标；72 个关键点坐标也包含具体坐标，但不包含对应位置的详细位置描述。

请求参数

无

返回参数

参数	类型	是否必须	说明
log_id	uint64	是	日志 id
result_num	uint32	是	人脸数目
result	object[]	是	人脸属性对象的集合
+age	double	否	年龄。face_fields 包含 age 时返回
+beauty	double	否	美丑打分，范围 0-100，越大表示越美。face_fields 包含 beauty 时返回
+location	object	是	人脸在图片中的位置
++left	uint32	是	人脸区域离左边界的距离
++top	uint32	是	人脸区域离上边界的距离
++width	uint32	是	人脸区域的宽度
++height	uint32	是	人脸区域的高度
+face_probability	double	是	人脸置信度，范围 0-1
+rotation_angle	int32	是	人脸框相对于竖直方向的顺时针旋转角，[-180,180]
+yaw	double	是	三维旋转之左右旋转角 [-90(左), 90(右)]
+pitch	double	是	三维旋转之俯仰角度 [-90(上), 90(下)]
+roll	double	是	平面内旋转角 [-180(逆时针), 180(顺时针)]
+expression	uint32	否	表情，0：不笑；1：微笑；2：大笑。face_fields包含expression时返回
+expression_probability	double	否	表情置信度，范围 0~1。face_fields 包含expression时返回
+faceshape	object[]	否	脸型置信度。face_fields 包含 faceshape 时返回
++type	string	是	脸型：square/triangle/oval/heart/round
++probability	double	是	置信度：0~1
+gender	string	否	male、female。face_fields 包含 gender时返回
+gender_probability	double	否	性别置信度，范围 0~1。face_fields 包含gender时返回
+glasses	uint32	否	是否带眼镜，0-无眼镜，1-普通眼镜，2-墨镜。face_fields包含glasses时返回
+glasses_probability	double	否	眼镜置信度，范围 0~1。face_fields包含glasses时返回
+landmark	object[]	否	4 个关键点位置，左眼中心、右眼中心、鼻尖、嘴中心。face_fields包含landmark时返回
++x	uint32	否	x坐标
++y	uint32	否	y坐标
+landmark72	object[]	否	72个特征点位置，示例图 。face_fields包含landmark时返回
++x	uint32	否	x坐标
++y	uint32	否	y坐标
+race	string	否	yellow、white、black、arabs。face_fields包含race时返回
+race_probability	double	否	人种置信度，范围 0~1。face_fields包含race时返回
+qualities	object	否	人脸质量信息。face_fields 包含qualities时返回
++occlusion	object	是	人脸各部分遮挡的概率， [0, 1] （待上线）
+++left_eye	double	是	左眼
+++right_eye	double	是	右眼
+++nose	double	是	鼻子
+++mouth	double	是	嘴
+++left_cheek	double	是	左脸颊
+++right_cheek	double	是	右脸颊
+++chin	double	是	下巴
++blur	double	是	人脸模糊程度，[0, 1]。0表示清晰，1表示模糊（待上线）
++illumination	-	是	取值范围在[0,255],表示脸部区域的光照程度（待上线）
++completeness	-	是	人脸完整度，[0, 1]。0表示完整，1表示不完整（待上线）
++type	object	是	真实人脸/卡通人脸置信度
+++human	-	是	真实人脸置信度，[0, 1]
+++cartoon	-	是	卡通人脸置信度，[0, 1]

返回示例


```
{
    "result_num": 1,
    "result": [
      {
        "location": {
          "left": 90,
          "top": 92,
          "width": 111,
          "height": 99
        },
        "face_probability": 1,
        "rotation_angle": 6,
        "yaw": 11.61234664917,
        "pitch": -0.30852827429771,
        "roll": 8.8044967651367,
        "landmark": [
          {
            "x": 105,
            "y": 110
          },
          ...
        ],
        "landmark72": [
          {
            "x": 88,
            "y": 109
          },
          ...
        ],
        "gender": "male",
        "gender_probability": 0.99358034133911,
        "glasses": 0,
        "glasses_probability": 0.99991309642792,
        "race": "yellow",
        "race_probability": 0.99960690736771,
        "qualities": {
          "occlusion": {
            "left_eye": 0.000085282314103097,
            "right_eye": 0.00001094374601962,
            "nose": 3.2677664307812e-7,
            "mouth": 2.6582130940866e-10,
            "left_cheek": 8.752236624332e-8,
            "right_cheek": 1.0212766454742e-7,
            "chin": 4.2632994357028e-10
          },
          "blur": 4.5613666312237e-41,
          "illumination": 0
```

```
                "completeness": 0,
                "type": {
                    "human": 0.98398965597153,
                    "cartoon": 0.016010366380215
                }
            }
        }
    ],
    "log_id": 2418894422
}
```


色情识别（antiporn）

功能描述

色情识别。

请求参数

无

返回参数

字段	类型	是否必须	说明
confidence_coefficient	string	是	结果是否确定，分为“确定”和“不确定”两种。
result_num	uint32	是	返回结果数目，即：result 数组中元素个数。
result	array(array(double))	是	结果数组，每项内容对应一个分类维度的结果。
conclusion	string	是	本张图片最终鉴定的结果，分为“色情”，“性感”，“正常”三种。
log_id	uint64	是	请求标识码，随机数，唯一。

其中元素的每项内容包含以下字段：

字段	类型	是否必须	说明	示例
class_name	string	是	分类结果名称	色情
probability	double	是	分类结果置信度	0.89471650123596

返回示例


```
{
 "result": [{
  "probability": 0.000301,
  "class_name": "色情"
 },
 {
  "probability": 0.000054,
  "class_name": " 性感"
 },
 {
  "probability": 0.999645,
  "class_name": "正常"
 }],
 "conclusion": "正常",
 "log_id": 848999404,
 "confidence_coefficient": "确定",
 "result_num": 3
}
```


政治人物识别（politician）

功能描述

政治人物识别。

请求参数

无

返回参数

参数	子参数	必须	类型	是否必须	说明
log_id	-	-	uint64	是	日志id
result_num	-	-	uint32	是	实际检测到人脸数目(不大于max_face_num)
result	-	-	object[]	是	
-	location	-	object	是	人脸在输入图片中的位置
-	-	left	uint32	是	人脸区域离左边界的距离
-	-	top	uint32	是	人脸区域离上边界的距离
-	-	width	uint32	是	人脸区域的宽度
-	-	height	uint32	是	人脸区域的高度
-	stars	-	object[]	是	公众人物数组
-	-	name	string	是	姓名
-	-	star_id	string	是	人物id，全局唯一
-	-	probability	float	是	相似度，[0, 1]

返回示例


```
{
    "log_id": 3268660173,
    "result_num": 1,
    "result": [
        {
            "location": {
                "left": 132,
                "top": 168,
                "width": 238,
```

```
                "height": 223
            },
            "stars": [
                {
                    "name": "张三",
                    "star_id": "515617",
                    "probability": 0.9750030040741
                }
            ]
        }
    ]
}
```


暴恐识别（terror）

功能描述

暴恐识别

请求参数

无

返回参数

字段	类型	是否必须	说明
result	array(array(double))	是	暴恐置信度得分。
log_id	uint64	是	请求标识码，随机数，唯一。
result_coarse	object[]	是	粗粒度得分结果
name	string	是	粗粒度标签，包含两个标签：正常、暴恐
score	float	是	对应标签的置信度得分，越高可信度越高
result_fine	object[ ]	是	细粒度得分结果
name	string	是	细粒度标签，包含9个标签：正常、警察部队、血腥、尸体、爆炸火灾、杀人、暴乱、暴恐人物、军事武器
score	float	是	对应标签的置信度得分，越高可信度越高

返回示例


```
{
    "errno": 0,
    "msg": "success",
    "data": {
        "result": 0.0082325544208288,
        "result_coarse": [
            {
                "name": "正常",
                "score": 0.99176746606827
            },
            {
                "name": "暴恐",
                "score": 0.0082325544208288
            }
        ],
        "result_fine": [
            {
                "name": "正常",
                "score": 0.98908758163452
            },
            {
                "name": "警察部队",
                "score": 0.0062405453063548
            },
            {
                "name": "血腥",
                "score": 0.0009653537417762
            },
            {
                "name": "尸体",
                "score": 0.001054480439052
            },
            {
                "name": "爆炸火灾",
                "score": 0.00011743687355192
            },
            {
                "name": "杀人",
                "score": 0.0011699661845341
            },
            {
                "name": "暴乱",
                "score": 0.0000211903588933 64
            },
            {
                "name": "暴恐人物",
                "score": 0.0010401027975604
            },
            {
                "name": "军事武器",
                "score": 0.00030337597127073
            }
        ]
    }
}
```


公众人物识别（public）

功能描述

公众人物识别

请求参数

参数	是否必选	类型	说明
max_face_num	否	uint32	最多处理人脸数目，默认值1，最大值5
max_star_num	否	uint32	单人脸最多相似明星数，默认4

返回参数

参数	子参数	子参数	类型	必须	说明
log_id	-	-	uint64	是	日志id
result_num	-	-	uint32	是	实际检测到人脸数目(不大于max_face_num)
result	-	-	object[]	是	
-	location	-	object	是	人脸在输入图片中的位置
-	-	left	uint32	是	人脸区域离左边界的距离
-	-	top	uint32	是	人脸区域离上边界的距离
-	-	width	uint32	是	人脸区域的宽度
-	-	height	uint32	是	人脸区域的高度
-	stars	-	object[]	是	公众人物数组
-	-	name	string	是	姓名
-	-	star_id	string	是	人物id，全局唯一
-	-	probability	float	是	相似度，[0, 1]

返回示例

```
{
  "log_id": 3268660173,
  "result_num": 1,
  "result": [
    {
      "location": {
        "left": 132,
        "top": 168,
        "width": 238,
        "height": 223
      },
      "stars": [
        {
          "name": "张三",
          "star_id": "515617",
          "probability": 0.9750030040741
        }
      ]
    }
  ]
}
```

恶心图识别（disgust）

功能描述

恶心图识别

请求参数

无

返回参数

参数	类型	必须	说明
log_id	uint64	是	请求标识码，随机数，唯一
result	double	是	得分

返回示例

```
  {
   "result": 9.2708455667889E-7,
   "log_id": 2977989308
  }
```

水印二维码识别（watermark）

功能描述

水印检测

请求参数

无

返回值

参数	类型	必须	说明
log_id	uint64	是	请求标识码，随机数，唯一
result_num	uint32	否	返回结果数目，即：result数组中元素个数
result	array(object)	否	返回结果数组，每一项为一个检测出的结果

result中元素的每项内容包含以下字段：

参数	类型	必须	说明
location	object	否	位置信息（左起像素位置、上起像素位置、像素宽、像素高）
probability	double	是	分类结果置信度 [0-1.0]
type	string	是	返回结果的类型 (watermark、barcode、QR code)

| type | string | 是 | 返回结果的类型 (watermark、bar code、QR code) |

返回示例

```
{
 "result": [{
  "probability": 0.99872654676437,
  "type": "watermark"
 },
 {
  "probability": 0.98578763008118,
  "type": "watermark"
 }],
 "log_id": 686882979,
 "result_num": 2
}
```

图像质量识别（quality）

功能描述

图片质量

请求参数

无

返回参数

参数	子参数	类型	必须	说明
log_id	-	uint64	是	日志 id
result	-	object[]	是	
-	aesthetic	double	是	美观度
-	clarity	double	是	清晰度

返回示例

```
{
 "result": {
  "aesthetic": 0.26410301526388,
  "clarity": 0.28039423624674
 },
 "log_id": 93316007
}
```

音视频处理接口

基本概念

音视频行业用户在将BOS作为后端存储源站时，结合上层业务逻辑往往会有不同的使用场景。
随着业务层对用户体验的不断演进，音视频业务对数据加工、处理、访问的实时性要求越来越高，并且希提供望更加简化的使用方式。

百度智能云BOS提供了业界全新的音视频处理接口，支持上传mp4格式的文件，系统自动保存为hls格式，用于m3u8风格播放器读取使用。充分满足了“UGC上传——自动化处理——播放”这条使用流，免去了原本复杂的视频操作处理逻辑，并为业务提速。

音视频处理命令

音视频文件远大于图片文件，处理时间也较长，通常在秒级。因此，BOS音视频处理命令，采用了相较于[BOS图片处理接口](https://cloud.baidu.com/doc/BOS/s/gkbisf3l4)有所区别的写时处理方案，即音视频原文件写入BOS时即开始按需处理，而非基于用户读取时定义的参数处理。

BOS音视频处理接口定义如下：

- 格式转换参数

参数名称	缩写	类型	取值	命令描述	是否必选
format	f	string	[hls]	将视频转换转化为指定格式	是
time	t	int	[6,600]	切片时间，默认10s	否

- 持久化存储参数

参数名称	缩写	类型	取值	命令描述	是否必选
object	o	string	-	经过base64编码的object名称，受object长度限制，即不超过1024个字符(此处为转码后的m3u8文件名称，如xxx.m3u8)	是

请求方法

在BOS的POST接口中，以header形式填写参数，即可实现POST上传的音视频文件，按照header中的命令实现转封装及转存。

```
POST /video.mp4 HTTP/1.1
Content-Length：ContentLength
Content-Type: ContentType
x-bce-process=video/format,f_hls
```

>注：该接口的鉴权生成方式参考[PostObject](https://cloud.baidu.com/doc/BOS/s/akc5orrn5)接口。

请求示例

转封装并存储示例

将视频video.mp4转封装为hls，并将结果存储到video/demo.m3u8(此处文件目录定义与BOS规范一致)。

```
POST /video.mp4 HTTP/1.1
Content-Length：ContentLength
Content-Type: ContentType
x-bce-process=video/format,f_hls|system/save,o_dmlkZW8vZGVtby5tM3U4
```

如果需要指定分片时间转码，可以在参数中加入`t_*`设置分片时长，如指定每个分片时长为20s，则参数为`t_20`。
以下为header的一个示例：

`x-bce-process=video/format,f_hls,t_20|system/save,o_dmlkZW8vZGVtby5tM3U4`

注：`video/demo.m3u8` 经过Base64编码后结果为`dmlkZW8vZGVtby5tM3U4`。

规则和限制

支持的规格：

- 支持的原音视频大小：100MB以下
- 支持的原音视频格式：`mp4`
- 支持的目标音视频格式：`hls`
- 目前支持在北京、保定、苏州、广州等4个region使用
- 目前仅支持在BOS的`HTTP POST`类型上传接口中使用

当前BOS音视频处理仅支持`.mp4`文件向`.hls`文件类型的转换，更多类型的支持、文件码转换等能力，BOS后续将逐步支持。

如您短期内希望采用更广泛的处理能力，同时业务侧也可允许相对长一些的处理延迟，也可参考百度智能云[音视频转码MCT](https://cloud.baidu.com/product/mct.html)、[百度智能云音视频点播VOD](https://cloud.baidu.com/product/vod.html)，
示例：
- 获取BOS中音视频文件的媒体信息，参考[媒体信息获取接口](https://cloud.baidu.com/doc/MCT/s/Sjwvz5g0p)。

SDK

SDK概览

概述

本文介绍对象存储BOS包含的主流语言SDK。

前提条件

已开通BOS服务，详情请参见[开通BOS服务](BOS/控制台操作指南/操作准备.md)。

示例代码

对象存储BOS包含的主流语言SDK的安装和使用情况，如下表所示。

<table>
<thead>
<tr>
<th style="width:110px">
语言
</th>
<th style="width:280px">
参考文档
</th>
</thead>
<tbody>
<tr>
<td>
Java
</td>
<td>
Java SDK参考
</td>
</tr>
<tr>
<td>
Python
</td>
<td>
Python SDK参考
</td>
</tr>
<tr>
<td>
PHP
</td>
<td>
PHP SDK参考
</td>
</tr>
<tr>
<td>
C#
</td>
<td>
C# SDK参考
</td>
</tr>
<tr>
<td>
JavaScript
</td>
<td>
JavaScript SDK参考
</td>
</tr>
<tr>
<td>
Android
</td>
<td>

Android SDK参考
</td>
</tr>
<tr>
<td>
iOS
</td>
<td>
iOS SDK参考
</td>
</tr>
<tr>
<td>
Ruby
</td>
<td>
Ruby SDK参考
</td>
</tr>
<tr>
<td>
GO
</td>
<td>
GO SDK参考
</td>
</tr>
<tr>
<td>
C++
</td>
<td>
C++ SDK参考
</td>
</tr>
<tr>
<td>
Harmony
</td>
<td>
鸿蒙SDK参考
</td>
</tr>
</tbody>
</table>

</br>

Java-SDK

概述

本文档主要介绍BOS Java SDK的安装和使用。在使用本文档前，您需要先了解BOS的一些基本知识，并已经开通了BOS服务。若您还不了解BOS，可以参考[产品描述](BOS/产品描述/产品简介.md)和[快速入门](https://cloud.baidu.com/doc/BOS/s/wk9hxjnr5)。

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（JDK9及以上版本已将tools.jar移除，JDK9及以上版本若需要使用BOS JavaSDK，需要在JDK9的lib下自行添加tools.jar）。

安装SDK

- 对象存储Java SDK源码地址[Java SDK](https://github.com/baidubce/bce-sdk-java/tree/master/src/main/java/com/baidubce/services/bos)

- 对象存储Java SDK示例[Demo](https://github.com/baidubce/bce-sdk-java/tree/master/src/main/java/com/baidubce/services/bos/demo)

方式一：使用Maven安装

在Maven的pom.xml文件中添加bce-java-sdk的依赖：

```java
<dependency>
    <groupId>com.baidubce</groupId>
    <artifactId>bce-java-sdk</artifactId>
    <version>{version}</version>
</dependency>
```

其中，{version}为版本号，可以在maven仓库找到。

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目录结构

```
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
│   └── bos                             //BOS服务相关类
│       ├── model                       //BOS内部model，如Request或Response
│       ├── BosClient.class             //BOS客户端入口类
│       └── BosClientConfiguration.class    //针对BOS特有的HttpClient的配置
├── 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
```

卸载SDK

预期卸载 SDK 时，删除 pom 依赖或源码即可。

初始化

确认Endpoint

在确认您使用SDK时配置的Endpoint时，可先阅读开发人员指南中关于 BOS访问域名 的部分，理解Endpoint相关的概念。 百度智能云目前开放了多区域支持，请参考区域选择说明。

目前支持多个区域，对应信息为：

区域	访问EndPoint	支持协议
北京	bj.bcebos.com	HTTP,HTTPS
保定	bd.bcebos.com	HTTP,HTTPS
苏州	su.bcebos.com	HTTP,HTTPS
广州	gz.bcebos.com	HTTP,HTTPS
成都	cd.bcebos.com	HTTP,HTTPS
香港	hkg.bcebos.com	HTTP,HTTPS
武汉	fwh.bcebos.com	HTTP,HTTPS
金融云上海专区	fsh.bcebos.com	HTTP,HTTPS

获取密钥

要使用百度智能云BOS，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BOS做签名验证。 可以通过如下步骤获得并了解您的AK/SK信息：

注册百度智能云账号

创建AK/SK

新建BosClient

BosClient是BOS服务的客户端，为开发者与BOS服务进行交互提供了一系列的方法。

使用AK/SK新建BosClient

通过AK/SK方式访问BOS，用户可以参考如下代码新建一个BosClient：

```
 public class Sample {
public static void main(String[] args) {
    String ACCESS_KEY_ID = <your-access-key-id>;                // 用户的Access Key ID
    String SECRET_ACCESS_KEY = <your-secret-access-key>;        // 用户的Secret Access Key

    // 初始化一个BosClient
    BosClientConfiguration config = new BosClientConfiguration();
    config.setCredentials(new DefaultBceCredentials(ACCESS_KEY_ID, SECRET_ACCESS_KEY));
    BosClient client = new BosClient(config);
    }
}
```

在上面代码中，`ACCESS_KEY_ID`对应控制台中的“Access Key ID”，`SECRET_ACCESS_KEY`对应控制台中的“Access Key Secret”，获取方式请参考《操作指南 管理ACCESSKEY》。

可以通过传入ENDPOINT参数来指定不同地域的BOS官方域名，不指定时默认为北京区域`http://bj.bcebos.com`。

```
 String ACCESS_KEY_ID = <your-access-key-id>;                // 用户的Access Key ID
String SECRET_ACCESS_KEY = <your-secret-access-key>;        // 用户的Secret Access Key
String ENDPOINT = <domain-name>;                            // 用户自己指定地域的官方域名

BosClientConfiguration config = new BosClientConfiguration();
config.setCredentials(new DefaultBceCredentials(ACCESS_KEY_ID,SECRET_ACCESS_KEY));
config.setEndpoint(ENDPOINT);
BosClient client = new BosClient(config);
```

> **注意：**如需使用自定义域名访问，除了修改`ENDPOINT`参数外还需要打开自定义域名开关，详见下述配置自定义域名访问BOS。

使用STS创建BosClient

申请STS token

BOS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。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BOS，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智能云STS使用介绍。

用STS token新建BOSClient

申请好STS后，可将STStoken配置到BosClient中，用户可以参考如下代码新建一个BosClient：

```
public class StsExample {
    private static final String STS_ENDPOINT = "http://sts.bj.baidubce.com";
    private static final String ENDPOINT = "bj.bcebos.com";
    private static final String ACCESS_KEY_ID = "your accesskey id";
    private static final String SECRET_ACCESS_KEY = "your secret accesskey";

    public static void main(String[] args) {
        BceCredentials credentials = new DefaultBceCredentials(ACCESS_KEY_ID, SECRET_ACCESS_KEY);
        StsClient client = new StsClient(
                new BceClientConfiguration().withEndpoint(STS_ENDPOINT).withCredentials(credentials)
        );
        GetSessionTokenResponse response = client.getSessionToken(new GetSessionTokenRequest());
        // or simply call:
        // GetSessionTokenResponse response = client.getSessionToken();
        // or you can specify limited permissions with ACL:
        // GetSessionTokenResponse response = client.getSessionToken(new GetSessionTokenRequest().withAcl("blabla"));
        // build DefaultBceSessionCredentials object from response:
        BceCredentials bosstsCredentials = new DefaultBceSessionCredentials(
                response.getAccessKeyId(),
                response.getSecretAccessKey(),
                response.getSessionToken());
        System.out.println("===================================");
        System.out.println("GetSessionToken result:");
        System.out.println("    accessKeyId:  " + response.getAccessKeyId());
        System.out.println("    secretAccessKey:  " + response.getSecretAccessKey());
        System.out.println("    securityToken:  " + response.getSessionToken());
        System.out.println("    expiresAt:  " + response.getExpiration().toString());
        System.out.println("===================================");

        // build bos client
        BosClientConfiguration config = new BosClientConfiguration();
        config.setCredentials(bosstsCredentials);
        config.setEndpoint(ENDPOINT);
        BosClient bosClient = new BosClient(config);
    }
}
```

> 注意： 目前使用STS配置client时，无论对应bucket的区域在哪里，endpoint都需配置为`http://sts.bj.baidubce.com`， 但创建BosClient时，仍需使用BOS的endpoint，如`bj.bcebos.com`、`su.bcebos.com`等.

配置自定义域名访问BOS

使用自定义域名 如果希望使用自定义域名作为访问BOS的endpoint，在控制台将自定义域名和BOS某个bucket绑定之后，配置endpoint为自定义域名并打开CnameEnabled开关，例如cdn-test.cdn.bcebos.com，配置代码如下:

```
String ACCESS_KEY_ID = <your-access-key-id>;                // 用户的Access Key ID
String SECRET_ACCESS_KEY = <your-secret-access-key>;        // 用户的Secret Access Key
String ENDPOINT = "https://cdn-test.cdn.bcebos.com";        // 用户自己指定的域名

BosClientConfiguration config = new BosClientConfiguration();
config.setCredentials(new DefaultBceCredentials(ACCESS_KEY_ID,SECRET_ACCESS_KEY));
config.setEndpoint(ENDPOINT);
config.setCnameEnabled(true);
BosClient client = new BosClient(config);
```

配置HTTPS协议访问BOS

BOS支持HTTPS传输协议，您可以通过如下两种方式在BOS Java SDK中使用HTTPS访问BOS服务：

- 在endpoint中指明https:

```
String endpoint = "https://bj.bcebos.com";
String ak = "ak";
String sk = "sk";
BosClientConfiguration config = new BosClientConfiguration();
config.setCredentials(new DefaultBceCredentials(ak, sk));
config.setEndpoint(endpoint);
BosClient client = new BosClient(config);
```

- 通过调用setProtocol方法设置https协议:

```
String endpoint = "bj.bcebos.com"; // endpoint中不包含protocol
String ak = "ak";
String sk = "sk";
BosClientConfiguration config = new BosClientConfiguration();
config.setCredentials(new DefaultBceCredentials(ak, sk));
config.setEndpoint(endpoint);
config.setProtocol(Protocol.HTTPS); // 如果不指明, 则使用http
BosClient client = new BosClient(config);
```

> **注意：**如果在endpoint中指明了protocol, 则endpoint中的生效, 另外单独再调用setProtocol()不起作用。

```
String endpoint = "http://bj.bcebos.com";
String ak = "ak";
String sk = "sk";
BosClientConfiguration config = new BosClientConfiguration();
config.setCredentials(new DefaultBceCredentials(ak, sk));
config.setEndpoint(endpoint);
config.setProtocol(Protocol.HTTPS); // endpoint中已经指明, 此为无效操作, 对http也是如此
BosClient client = new BosClient(config);
```

配置BosClient

如果用户需要配置BosClient的一些细节的参数，可以在构造BosClient的时候传入BosClientConfiguration对象。 BosClientConfiguration是BOS服务的配置类，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BOS服务：

```
String ACCESS_KEY_ID = <your-access-key-id>;                // 用户的Access Key ID
String SECRET_ACCESS_KEY = <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BosClientConfiguration实例
BosClientConfiguration config = new BosClientConfiguration();

// 配置代理为本地8080端口
config.setProxyHost("127.0.0.1");
config.setProxyPort(8080);

// 创建BOS客户端
config.setCredentials(new DefaultBceCredentials(ACCESS_KEY_ID,SECRET_ACCESS_KEY));
config.setEndpoint(ENDPOINT);
BosClient client = new BosClient(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BosClientConfiguration实例
BosClientConfiguration config = new BosClientConfiguration();

// 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BosClientConfiguration对基本网络参数进行设置：

```
BosClientConfiguration config = new BosClientConfiguration();

// 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.setConnectionTimeoutInMillis(5000);

// 设置Socket传输数据超时的时间为2000毫秒
config.setSocketTimeoutInMillis(2000);
```

设置访问域名风格

在0.10.227版本之后，SDK会默认自动开启bucket virtual hosting，可以通过以下配置开启使用PathStyle风格的endpoint：

```
BosClientConfiguration config = new BosClientConfiguration();

// 开启使用PathStyle
config.setPathStyleAccessEnable(true);
```

> 注意：部分非BOS常规域名下，默认bucket virtual hosting风格的可能会有不兼容场景，表现为dns解析失败等，可以开启PathStyle解决。 其他场景建议使用默认风格。

设置重定向 在0.10.330版本之后，出于安全考虑，SDK会默认关闭自动重定向。用户可以通过配置来开启自动重定向和配置参数：

```
BosClientConfiguration config = new BosClientConfiguration();

// 开启自动重定向
config.setRedirectsEnabled(true);
// 设置最大自动重定向次数
config.setMaxRedirects(10);
```

设置同步PUT

对PUT操作，默认使用CloseableHttpAsyncClient，可能会出现用户进程执行完未退出的现象。用户可以通过BosClientConfiguration设置所有BOS请求都通过BosClient同步的方式：

```
BosClientConfiguration config = new BosClientConfiguration();

// 设置PUT操作为同步方式，默认异步
config.setEnableHttpAsyncPut(false);
```

参数说明

通过BosClientConfiguration能指定的所有参数如下表所示：

参数	说明
CnameEnabled	使用cname访问BOS资源
ConnectionTimeoutInMillis	建立连接的超时时间（单位：毫秒）
Credentials	客户端用于签署HTTP请求的BCE凭据
EnableHttpAsyncPut	异步put
Endpoint	访问域名
LocalAddress	本地地址
MaxConnections	允许打开的最大HTTP连接数
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
ProxyPassword	代理服务器验证的密码
ProxyPort	代理服务器端口
ProxyPreemptiveAuthenticationEnabled	是否设置用户代理认证
ProxyUsername	代理服务器验证的用户名
ProxyWorkstation	NTLM代理服务器的Windows工作站名称
Region	地域
RetryPolicy	连接重试策略
SocketBufferSizeInBytes	Socket缓冲区大小
SocketTimeoutInMillis	通过打开的连接传输数据的超时时间（单位：毫秒）
StreamBufferSize	流文件缓冲区大小
UserAgent	用户代理，指HTTP的User-Agent头
RedirectsEnabled	是否开启HTTP重定向。默认关闭
MaxRedirects	HTTP重定向自动重定向最大次数。默认为1次
PathStyleAccessEnable	是否使用PathStyle风格的endpoint，默认关闭，即使用bucket virtual hosting

Bucket管理

概述

Bucket既是BOS上的命名空间，也是计费、权限控制、日志记录等高级功能的管理实体。

- Bucket名称在所有区域中具有全局唯一性，且不能修改。

> **说明：** 百度智能云目前开放了多区域支持，请参考区域选择说明。
>
> 目前支持 “华北-北京”、“华北-保定”、“华东-苏州”、“华南-广州”、“西南-成都”、“中国香港”、“华中-武汉”和“华东-上海”。北京区域： http://bj.bcebos.com ，保定区域:http://bd.bcebos.com,苏州区域:http://su.bcebos.com,广州区域:http://gz.bcebos.com,成都区域:http://cd.bcebos.com,香港区域:http://hkg.bcebos.com,武汉区域：http://fwh.bcebos.com,上海区域：http://fsh.bcebos.com

- 存储在BOS上的每个Object都必须包含在一个Bucket中。
- 一个用户最多可创建100个Bucket，但每个Bucket中存放的Object的数量和大小总和没有限制，用户不需要考虑数据的可扩展性。

新建Bucket

如下代码可以新建一个Bucket：

```
public void createBucket (BosClient client, String bucketName) {
  // 新建一个Bucket
  client.createBucket(<bucketName>);                    //指定Bucket名称
}
```

> **注意：** 由于Bucket的名称在所有区域中是唯一的，所以需要保证bucketName不与其他所有区域上的Bucket名称相同。
>
> Bucket的命名有以下规范：
>
> - 只能包括小写字母，数字，短横线（-）。
> - 必须以小写字母或者数字开头。
> - 长度必须在4-63字节之间。

通过上述代码创建的bucket，权限是私有读写，存储类型是标准类型（Standard）。

- **新建Icc bucket**

java sdk支持新建Icc bucket，用户可以在构造新建bucket的请求中填入IccIocation，CreateBucketRequest具体参数如下：

参数	类型	说明
bucketTags	String	Bucket 标签
IccLocation	String	Icc id，用于创建Icc bucket到特定的Icc集群
enableDedicated	Boolean	用于控制控制台开启Icc识别白名单

如下代码可以新建一个Icc Bucket：

```
public void createBucket (BosClient client, String bucketName) {
  CreateBucketRequest request = new CreateBucketRequest("bucketName");
  // 设置lcc location id
  request.setLccLocation("lcc id");
  request.setEnableDedicated(true);
  client.createBucket(request);
}
```

查看Bucket所属的区域

Bucket Location即Bucket Region，百度智能云支持的各region详细信息可参见区域选择说明。

如下代码可以获取该Bucket的Location信息：

```
BosClient client = new BosClient(config);

ListBucketsResponse listBucketsResponse =  client.listBuckets();

List<BucketSummary> bucketSummaryList =  listBucketsResponse.getBuckets();

for(BucketSummary bs : bucketSummaryList){
   System.out.println( bs.getLocation());
}

System.out.println(client.getBucketLocation("bucket-test").getLocationConstraint());
```

Bucket权限管理

设置Bucket的访问权限

如下代码将Bucket的权限设置为了private。

```
public void setBucketPrivate (BosClient client, String bucketName) {
  client.setBucketAcl(<bucketName>, CannedAccessControlList.Private);
}
```

CannedAccessControlList是枚举类型，包含三个值： `Private` 、 `PublicRead` 、 `PublicReadWrite` ，它们分别对应相关权限。具体内容可以参考BOS API文档 使用CannedAcl方式的权限控制。

设置指定用户对Bucket的访问权限

BOS还可以实现设置指定用户对Bucket的访问权限，参考如下代码实现：

```
List<Grant> accessControlList = new ArrayList<Grant>();
List<Grantee> grantees = new ArrayList<Grantee>();
List<Permission> permissions = new ArrayList<Permission>();
List<String> ipAddress = new ArrayList<String>();
List<String> stringLike = new ArrayList<String>();
List<String> stringEquals = new ArrayList<String>();
List<String> resource = new ArrayList<String>();
List<String> notResource = new ArrayList<String>();
Referer referer = new Referer();
Condition condition = new Condition();

// 授权给特定用户
grantees.add(new Grantee("user_id1"));
grantees.add(new Grantee("user_id2"));
grantees.add(new Grantee("user_id3"));

//授权给Everyone
grantees.add(new Grantee("*"));


//设置权限
permissions.add(Permission.WRITE);
permissions.add(Permission.READ);
permissions.add(Permission.LIST);

// 设置ip
ipAddress.add("ipAddress1");
ipAddress.add("ipAddress2");
ipAddress.add("ipAddress3");
condition.setIpAddress(ipAddress);

//设置 refer stringLike
stringLike.add("http://www.example1.com/");
stringLike.add("http://www.example2.com/");
stringLike.add("http://www.example3.com/");
referer.setStringLike(stringLike);
condition.setReferer(referer);

// 设置 refer stringEquals
stringEquals.add("http://www.baidu.com");
stringEquals.add("http://www.xiaomi.com");
stringEquals.add("http://www.google.com");
referer.setStringEquals(stringEquals);
condition.setReferer(referer);

// 设置 resource
resource.add("yourBucketName");


//设置notResource
List<String> notResouce = new ArrayList<String>();
notResouce.add("yourBucketName");
notResouce.add("yourBucketName/*");

Grant grant = new Grant();
```

```
grant.setGrantee(grantees);
grant.setPermission(permissions);
grant.setCondition(condition);
grant.setResource(resource);

List<Grantee> grantees1 = new ArrayList<Grantee>();
List<Permission> permissions1 = new ArrayList<Permission>();
List<String> ipAddress1 = new ArrayList<String>();
List<String> stringLike1 = new ArrayList<String>();
List<String> stringEquals1 = new ArrayList<String>();
List<String> resource1 = new ArrayList<String>();
List<String> notResource1 = new ArrayList<String>();
Referer referer1 = new Referer();
Condition condition1 = new Condition();

// 授权给特定用户
grantees1.add(new Grantee("user_id4"));
grantees1.add(new Grantee("user_id5"));
grantees1.add(new Grantee("user_id6"));

//授权给Everyone
grantees.add(new Grantee("*"));

//设置权限
permissions.add(Permission.FULL_CONTROL);
permissions1.add(Permission.WRITE);
permissions1.add(Permission.READ);
permissions1.add(Permission.LIST);

// 设置ip
ipAddress1.add("ipAddress4");
ipAddress1.add("ipAddress5");
ipAddress1.add("ipAddress6");
condition1.setIpAddress(ipAddress1);

//设置 refer stringLike
stringLike1.add("http://www.example4.com/");
stringLike1.add("http://www.example5.com/");
stringLike1.add("http://www.example6.com/");
referer1.setStringLike(stringLike1);
condition1.setReferer(referer1);

// 设置 refer stringEquals
stringEquals1.add("http://www.baidu1.com");
stringEquals1.add("http://www.xiaomi1.com");
stringEquals1.add("http://www.google1.com");
referer1.setStringEquals(stringEquals1);
condition1.setReferer(referer1);

// 设置 resource
resource1.add("yourBucketName");

// 设置notResource
List<String> notResouce = new ArrayList<String>();
notResouce.add("yourBucketName");
notResouce.add("yourBucketName/*");

Grant grant1 = new Grant();

grant1.setGrantee(grantees1);
grant1.setPermission(permissions1);
grant1.setCondition(condition1);
grant1.setResource(resource1);

accessControlList.add(grant);
accessControlList.add(grant1);

SetBucketAclRequest request = new SetBucketAclRequest("yourBucketName",accessControlList);
client.setBucketAcl(request);
```

> **注意：** resource和notResource不能同时设置 Permission中的权限设置包含三个值：`READ`、`WRITE`、`FULL_CONTROL`，它们分别对应相关权限。具体内容可以参考BOS API文档 上传ACL文件方式的权限控制。

设置更多Bucket访问权限

1. 通过设置refer白名单方式设置防盗链

```
 String jsonAcl = "";
client.setBucketAcl("bucketName", jsonAcl)
```

其中jsonAcl为{\"accessControlList\":["+ "{\"grantee\":[{\"id\":\"*\"}], "+ "\"permission\":[\"FULL_CONTROL\"], "+ "\"condition\":{\"referer\":{\"stringEquals\":[\"http://test/index\"]}" + "}}}}

2. 限制客户端IP访问，只允许部分客户端IP访问

```
 String jsonAcl = "";
client.setBucketAcl("bucketName", jsonAcl)
```

其中jsonAcl为{\"accessControlList\":["+ "{\"grantee\":[{\"id\":\"*\"}], "+ "\"permission\":[\"FULL_CONTROL\"], "+ "\"condition\":{\"ipAddress\":[\"192.170.0.6\"]" + "}}}}"}

设置STS临时token权限

对于通过STS方式创建的临时访问身份，管理员也可进行专门的权限设定。 STS的简介及设置临时权限的方式可参见临时授权访问。

使用BOS JAVA SDK设置STS临时token权限可参考使用STS创建BosClient

查看Bucket的权限

如下代码可以查看Bucket的权限：

```
 GetBucketAclResponse aclResponse = client.getBucketAcl("bucketName");
System.out.println(aclResponse.getAccessControlList().toString());
```

getBucketAcl方法返回的解析类中可供调用的参数有：

参数	说明
owner	Bucket owner信息
id	Bucket owner的用户ID
acl	标识Bucket的权限列表
grantee	标识被授权人
-id	被授权人ID
permission	标识被授权人的权限

列举Bucket

如下代码可以列出用户所有的Bucket，或参考完整示例。

```
public void listBuckets (BosClient client) {
    // 获取用户的Bucket列表
    List<BucketSummary> buckets = client.listBuckets().getBuckets();

    // 遍历Bucket
    for (BucketSummary bucket : buckets) {
        System.out.println(bucket.getName());
    }
}
```

删除Bucket

如下代码可以删除一个Bucket，或参考完整示例。

```
public void deleteBucket (BosClient client, String bucketName) {
    // 删除Bucket
    client.deleteBucket(<bucketName>);                    //指定Bucket名称
}
```

> **注意：**
>
> - 在删除前需要保证此Bucket下的所有Object和未完成的三步上传Part已经被删除，否则会删除失败。
> - 在删除前确认该Bucket没有开通跨区域复制，不是跨区域复制规则中的源Bucket或目标Bucket，否则不能删除。

判断Bucket是否存在

若用户需要判断某个Bucket是否存在，则如下代码可以做到，或参考完整示例。

```
public void doesBucketExist (BosClient client, String bucketName) {

    // 获取Bucket的存在信息
    // 指定Bucket名称
    boolean exists = client.doesBucketExist(bucketName);

    // 输出结果
    if (exists) {
        System.out.println("Bucket exists");
    } else {
        System.out.println("Bucket not exists");
    }
}
```

> **注意：** 如果Bucket不为空（即Bucket中有Object存在），则Bucket无法被删除，必须清空Bucket后才能成功删除。

设置Bucket存储类型

新建的Bucket默认是标准存储类型，用户也可以通过如下代码设置或获取指定Bucket的存储类型：

参数	说明
storageClass	存储类型，支持"STANDARD"、"STANDARD_IA"、"COLD"、"ARCHIVE"

```
public void PutBucketStorageClass() {
    // 设置"STANDARD_IA"，低频存储
    String storageClass = STORAGE_CLASS_STANDARD_IA;
    this.client.putBucketStorageClass(this.bucketName, storageClass);
}

public String GetBucketStorageClass() {
    GetBucketStorageClassResponse response = this.client.getBucketStorageClass(this.bucketName);
    return response.getStorageClass();
}
```

设置Bucket服务端加密

若用户需要开启Bucket服务端加密，则如下代码可以做到：

参数	说明
encryptionAlgorithm	指定Bucket的服务器端加密类型，当前只支持AES256加密。

```
public void PutBucketEncryptionByEncryption(BosClient client, String bucketName, String encryptionAlgorithm ) {
  SetBucketEncryptionRequest setBucketEncryptionRequest = new SetBucketEncryptionRequest();
  setBucketEncryptionRequest.setBucketName(bucketName);
  BucketEncryption encryption = new BucketEncryption();
  encryption.setEncryptionAlgorithm(encryptionAlgorithm);
  setBucketEncryptionRequest.setBucketEncryption(encryption);
  client.setBucketEncryption(setBucketEncryptionRequest);
}
```

若用户想查看Bucket服务端加密信息，则如下代码可以做到:

```
public GetBucketEncryptionResponse GetBucketEncryption(BosClient client, String bucketName) {
  GetBucketEncryptionRequest getBucketEncryptionRequest = new GetBucketEncryptionRequest();
  getBucketEncryptionRequest.withBucketName(bucketName);
  GetBucketEncryptionResponse resp = new GetBucketEncryptionResponse();
  resp = client.getBucketEncryption(getBucketEncryptionRequest);
  return resp;
}
```

若用户想删除Bucket服务端加密信息，则如下代码可以做到:

```
public void DeleteBucketEncryption(BosClient client, String bucketName) {
  DeleteBucketEncryptionRequest deleteBucketEncryptionRequest = new DeleteBucketEncryptionRequest();
  deleteBucketEncryptionRequest.withBucketName(bucketName);
  client.deleteBucketEncryption(deleteBucketEncryptionRequest);
}
```

Bucket数据同步

若用户需要开启Bucket之间的数据同步，则如下代码可以做到：

参数	是否必须	描述
id	是	replication规则名，id 由数字字母 - _ 组成，不得超过20个字符
status	是	是否生效，生效为enabled
resource	是	replication生效前缀，resource的配置形式为｛$bucket_name/<生效的对象前缀>｝，必须要以$bucket_name+/开头
destination	是	复制的目的端配置
+bucket	是	目的Bucket name
+storageClass	否	目的Object的存储类型。如果保持和源Bucket的存储类型一致，则该参数不需要配置；如果需要单独指定存储类型可以为STANDARD，STANDARD_IA，COLD。
replicateHistory	否	历史文件复制，有该项则认为是开启。开启历史文件复制后，存量的全部Object都同步复制到目的Bucket，复制范围共用resource。
+storageClass	否	目的Object的存储类型。如果保持和源Bucket的存储类型一致，则该参数不需要配置；如果需要单独指定存储类型可以为STANDARD，STANDARD_IA，COLD。
replicateDeletes	是	是否开启删除同步,可以为enabled,disabled 。

```
public void PutBucketReplicationByReplication(BosClient client, String bucketName, String replicationId, String dstBucketName) {
SetBucketReplicationRequest request = new SetBucketReplicationRequest(bucketName);
  request.setId(replicationId);
  request.setStatus("enabled");
  String[] resource = {bucketName + "/abc"};
  request.setResource(resource);
  Destination destination = new Destination();
  destination.setBucket(dstBucketName);
  request.setDestination(destination);
  request.setReplicateDeletes("enabled");
  client.setBucketReplication(request);
 }
```

若用户想获取bucket指定id的数据同步信息，包括源Bucket名称、目的Bucket名称、存储类型、是否进行历史复制，数据同步策略，目的region等。如下代码可以做到:

```
public void GetBucketReplication(BosClient client, String bucketName, String replicationId) {
  GetBucketReplicationRequest grequest = new GetBucketReplicationRequest(bucketName);
  grequest.setId(replicationId);
  GetBucketReplicationResponse response = client.getBucketReplication(grequest);
 }
```

若用户想删除bucket指定id的数据同步，则如下代码可以做到：

```
public void DeleteBucketReplication(BosClient client, String bucketName, String replicationId) {
  DeleteBucketReplicationRequest drequest = new       DeleteBucketReplicationRequest();
  drequest.setBucketName(this.bucketName);
  drequest.setId(this.replicationId);
  client.deleteBucketReplication(drequest);
}
```

若用户想获取bucket所有的replication同步规则，则如下代码可以做到：

```
public void ListBucketReplication(BosClient client, String bucketName) {
  ListBucketReplicationResponse replicationResponse;
  ListBucketReplicationRequest listreq = new ListBucketReplicationRequest(this.bucketName);
  replicationResponse = client.listBucketReplication(listreq);
}
```

若用户想获取指定id的数据同步复制的进程状态，则如下代码可以做到：

```
public void GetBucketReplicationProgress(BosClient client, String bucketName，String replicationId) {
  GetBucketReplicationProgressRequest proreq = new GetBucketReplicationProgressRequest(this.bucketName);
  proreq.setId(this.replicationId);
  BucketReplicationProgress progress = client.getBucketReplicationProgress(proreq);
}
```

Bucket静态网站托管

将网站托管在Bucket上，实现轻量化运维,如下代码可以做到：

参数	说明
index	Index文件名称
notFound	404文件名称

```
public void PutBucketStaticWebsite(BosClient client, String bucketName, String index, String notFound) {
  SetBucketStaticWebsiteRequest setBucketStaticWebsiteRequest = new SetBucketStaticWebsiteRequest();
  setBucketStaticWebsiteRequest.setBucketName(bucketName);
  setBucketStaticWebsiteRequest.setIndex(index);
  setBucketStaticWebsiteRequest.setNotFound(notFound);
  client.setBucketStaticWebSite(setBucketStaticWebsiteRequest);
}
```

若用户想查看静态网站托管信息，则如下代码可以做到:

```
public GetBucketStaticWebsiteResponse GetBucketStaticWebsite(BosClient client, String bucketName) {
  GetBucketStaticWebsiteRequest getBucketStaticWebsiteRequest = new GetBucketStaticWebsiteRequest();
  getBucketStaticWebsiteRequest.withBucketName(bucketName);
  GetBucketStaticWebsiteResponse resp = new GetBucketStaticWebsiteResponse();
  resp = client.getBucketStaticWebsite(getBucketStaticWebsiteRequest);
  return resp;
}
```

若用户想关闭静态网站托管功能，则如下代码可以做到：

```
public void DeleteBucketStaticWebsite(BosClient client, String bucketName) {
  DeleteBucketStaticWebsiteRequest deleteBucketStaticWebsiteRequest = new DeleteBucketStaticWebsiteRequest();
  deleteBucketStaticWebsiteRequest.withBucketName(bucketName);
  client.deleteBucketStaticWebSite(deleteBucketStaticWebsiteRequest);
}
```

原图保护

若用户想开启Bucket的原图保护功能，则如下代码可以做到:

参数	说明
resource	表示生效的资源范围

```
public void PutBucketCopyrightProtection(BosClient client, String bucketName, List<String> resource) {
  SetBucketCopyrightProtectionRequest request = new SetBucketCopyrightProtectionRequest();
  request.setBucketName(bucketName);
  request.setResource(resource);
  client.setBucketCopyrightProtection(request);
}
```

若用户想获取某个Bucket的原图保护配置情况，则如下代码可以做到:

```
public GetBucketCopyrightProtectionResponse GetBucketCopyrightProtection(BosClient client, String bucketName) {
  GetBucketCopyrightProtectionRequest getBucketCopyrightProtectionRequest =
            new GetBucketCopyrightProtectionRequest();
  getBucketCopyrightProtectionRequest.withBucketName(bucketName);
  GetBucketCopyrightProtectionResponse resp = new GetBucketCopyrightProtectionResponse();
  resp = client.getBucketCopyrightProtection(getBucketCopyrightProtectionRequest);
  return resp;
}
```

若用户想关闭原图保护功能，则如下代码可以做到：

```
public void DeleteBucketCopyrightProtection(BosClient client, String bucketName) {
  DeleteBucketCopyrightProtectionRequest deleteBucketCopyrightProtectionRequest =
            new DeleteBucketCopyrightProtectionRequest();
  deleteBucketCopyrightProtectionRequest.withBucketName(bucketName);
  client.deleteBucketCopyrightProtection(deleteBucketCopyrightProtectionRequest);
}
```

镜像回源

若您为 Bucket 配置了镜像回源，当用户访问该 Bucket 中某一 Object 时，发现该 Object 在 BOS 中并不存在，则 BOS 会立刻向预先配置好的回源地址请求该 Object，保证访问正常进行。在请求过程中，源站返回的数据会自动给到请求用户而用户对该过程无感。同时，该数据也会自动存入 BOS，提升日后该类请求的流畅性。参考镜像回源。

设置镜像回源（PutBucketMirroring） | 参数 | 说明 | | ---------- | ---------- | | bucketName | 请求的bucket名称 | | bucketMirroringConfigurations | 镜像回源配置，具体参数与Api接口参数相同 |

如下代码通过构造对象参数来设置镜像回源：

```
// 构造镜像回源配置
BucketMirroringConfiguration config = BucketMirroringConfiguration.builder()
                .mode("fetch")
                .sourceUrl("http://xxx.bj.bcebos.com")
                .resource("*")
                .build();
List<BucketMirroringConfiguration> bucketMirroringConfigurations =
        new ArrayList<BucketMirroringConfiguration>();
bucketMirroringConfigurations.add(config);

// 请求
this.client.putBucketMirroring(bucketName, bucketMirroringConfigurations);
```

如下代码通过直接传入Json来设置镜像回源：

```
// 构造镜像回源配置
String configJson = "{\"bucketMirroringConfiguration\":[{\"mode\":\"fetch\"," +
        "\"sourceUrl\":\"http://xxx.bj.bcebos.com\",\"resource\":\"*\",\"prefix\":\"\"," +
        "\"suffix\":\"\"," +
        "\"fixedKey\":\"\",\"version\":\"v1\",\"customHeaders\":[],\"storageClass\":\"STANDARD\"," +
        "\"ignoreHeaders\":[],\"passHeaders\":[]}]}";

// 请求
this.client.putBucketMirroring(bucketName, configJson);
```

获取镜像回源配置（GetBucketMirroring） | 参数 | 说明 | | ---------- | ---------- | | bucketName | 请求的bucket名称 |

代码示例

```
// 请求
GetBucketMirroringResponse response = this.client.getBucketMirroring(bucketName);
System.out.println(response.getBucketMirroringConfiguration().toString());
```

删除镜像回源配置（DeleteBucketMirroring） | 参数 | 说明 | | ---------- | ---------- | | bucketName | 请求的bucket名称 | 代码示例

```
this.client.deleteBucketMirroring(bucketName);
```

标签管理

百度智能云提供标签管理功能，通过给每个云资源添加标签，从而快速分类和识别管理这些资源。详情参考管理Bucket标签API接口

设置Bucket的标签（PutBucketTagging） | 参数 | 类型 | 说明 | | ---------- |------| ---------- | | bucketName |String| 请求的bucket名称 | | tags | List< Object > | 标签对象列表，每个标签由键和值两部分组成，标签（键+值）唯一 | | +tagKey | String | 标签键 | | +tagValue | String | 标签值 |

> **注意：**设置标签接口可以新增和修改。标签键相同的新标签写入会覆盖旧标签

如下代码通过构造对象参数来设置标签：

```
// 构造标签配置
BucketTag tag = BucketTag.builder()
                .tagKey("key1")
                .tagValue("1")
                .build();
List<BucketTag> tags = new ArrayList<BucketTag>();
tags.add(tag);

// 请求
this.client.putBucketTagging(bucketName, tags);
```

如下代码通过直接传入json字符串来设置标签：

```
// 构造标签配置
String tag = "{\"tags\":[{\"tagKey\":\"key1\"," +
            "\"tagValue\":\"1\"}]}";

// 请求
this.client.putBucketTagging(bucketName, testJson);
```

获取Bucket的标签（GetBucketTagging） | 参数 | 类型|说明 | | ---------- |------| ---------- | | bucketName |String| 请求的bucket名称 |

代码示例：

```
GetBucketTaggingResponse response = this.client.getBucketTagging(bucketName);
```

删除Bucket的标签（DeleteBucketTagging） | 参数 | 类型|说明 | | ---------- |------| ---------- | | bucketName |String| 请求的bucket名称 |

> **注意：**删除标签接口会删除相应bucket的全部标签

代码示例：

```
this.client.deleteBucketTagging(bucketName);
```

回收站

用户可以通过配置回收站(Bucket Trash)的功能，保证删除后的数据在回收站内继续保留，用于后续对删除数据的找回。详情参考回收站功能

> **注意：**
> 1. 进入回收站的文件仍然会继续计费，直到用户彻底删除。对于回收站中的文件，建议用户配置生命周期来删除过期的文件；
> 2. 用户进行覆盖操作时，旧的数据不会进入回收站

开通回收站（PutBucketTrash） | 参数 | 类型 |是否必须| 说明 | | ---------- |-----| -----|---------- | | bucketName |String|是| 请求的bucket名称 | | trashDir | String|否| 指定回收站目录名。不填则回收站默认目录为".trash" |

> **注意：**
> 1. 若之前已开通过回收站功能，再次开通会覆盖原回收站目录名；
> 2. 回收站目录名词只能包含字母、数字、中文、下划线（_）和短横线（-）、小数点（.）。包含'/'会报错；
> 3. 开通了回收站功能的bucket， 删除object会回到回收站中(回收站中的object的全称是：trashDir + '/' + old object name)。未开通此功能的bucket，或者开通此功能的bucket且删除回收站内的object时，会彻底删除
> 4. 归档类型文件不支持Bucket Trash功能，删除归档类型文件时会直接删除

如下代码通过自定义名称来设置回收站：

```
String trashDir="testDir";
this.client.putBucketTrash(bucketName, trashDir);
```

如下代码设置默认回收站：

```
this.client.putBucketTrash(bucketName);
```

获取回收站开通信息（GetBucketTrash） | 参数 | 类型 |是否必须| 说明 | | ---------- |-----| -----|---------- | | bucketName |String|是| 请求的bucket名称 |

代码示例

```
GetBucketTrashResponse response=this.client.getBucketTrash(bucketName);
System.out.println(response.getTrashDir().toString());
```

关闭回收站（DeleteBucketTrash） | 参数 | 类型 |是否必须| 说明 | | ---------- |-----| -----|---------- | | bucketName |String|是| 请求的bucket名称 |

代码示例

```
this.client.deleteBucketTrash(bucketName);
```

文件管理

上传文件

在BOS中，用户操作的基本数据单元是Object。Object包含Key、Meta和Data。其中，Key是Object的名字；Meta是用户对该Object的描述，由一系列Name-Value对组成；Data是Object的数据。

BOS Java SDK提供了丰富的文件上传接口，可以通过以下方式上传文件：

- 简单上传
- 追加上传
- 分片上传
- 断点续传上传

简单上传

BOS在简单上传的场景中，支持以指定文件形式、以数据流方式、以二进制串方式、以字符串方式执行Object上传，请参考如下代码，或简单上传Demo。

```
public void PutObject(BosClient client, String bucketName, String objectKey, byte[] byte1, String string1){
    // 获取指定文件
    File file = new File("/path/to/file.zip");
    // 获取数据流
    InputStream inputStream = new FileInputStream("/path/to/test.zip");

    // 以文件形式上传Object
    PutObjectResponse putObjectFromFileResponse = client.putObject(bucketName, objectKey, file);
    // 以数据流形式上传Object
    PutObjectResponse putObjectResponseFromInputStream = client.putObject(bucketName, objectKey, inputStream);
    // 以二进制串上传Object
    PutObjectResponse putObjectResponseFromByte = client.putObject(bucketName, objectKey, byte1);
    // 以字符串上传Object
    PutObjectResponse putObjectResponseFromString = client.putObject(bucketName, objectKey, string1);
    // 创建空目录，objectKey需要以正斜线结尾，例如test/
    // BOS控制台默认会将以正斜线（/）结尾的对象，作为文件目录的形式展示
    PutObjectResponse putObjectResponseFromString = client.putObject(bucketName, objectKey, "");

    // 打印ETag
    System.out.println(putObjectFromFileResponse.getETag());
}
```

Object以文件的形式上传到BOS中，PutObject函数支持不超过5GB的Object上传。在PutObject请求处理成功后，BOS会在Header中返回Object的ETag作为文件标识。

设置文件元信息

文件元信息(Object Meta)，是对用户在向BOS上传文件时，同时对文件进行的属性描述，主要分为分为两种：设置HTTP标准属性（HTTP Headers）和用户自定义的元信息。

- **设定Object的Http Header**

BOS Java SDK本质上是调用后台的HTTP接口，因此用户可以在上传文件时自定义Object的Http Header。常用的http header说明如下：

名称	描述	默认值
Content-MD5	文件数据校验，设置后BOS会启用文件内容MD5校验，把您提供的MD5与文件的MD5比较，不一致会抛出错误	无
Content-Type	文件的MIME，定义文件的类型及网页编码，决定浏览器将以什么形式、什么编码读取文件。如没有指，BOS则根据文件的扩展名自动生成，如文件没有扩展名则填默认值	application/octet-stream
Content-Disposition	指示MIME用户代理如何显示附加的文件，打开或下载，及文件名称	无
Content-Length	上传的文件的长度，超过流/文件的长度会截断，不足为实际值	流/文件时间长度
Expires	缓存过期时间	无
Cache-Control	指定该Object被下载时的网页的缓存行为	无
x-bce-content-crc32	上传object的CRC值(循环冗余校验码)	无

参考代码如下：

```
 // 初始化上传输入流
ObjectMetadata meta = new ObjectMetadata();

// 设置ContentLength大小
meta.setContentLength(1000);

// 设置ContentType
meta.setContentType("application/json");

// 设置cache-control
meta.setCacheControl("no-cache");

// 设置x-bce-content-crc32，CRC32校验码需自行计算，示例格式如下
meta.setxBceCrc("0x74E947D0");

client.putObject(bucketName, objectKey, content, meta);
```

- **用户自定义元信息**

BOS支持用户自定义元数据来对Object进行描述。如下代码所示：

```
 // 设置自定义元数据name的值为my-data
meta.addUserMetadata("name", "my-data");

// 上传Object
client.putObject(bucketName, objectKey, content, meta);
```

> **提示：**
>
> - 在上面代码中，用户自定义了一个名字为"name"，值为"my-data"的元数据
> - 当用户下载此Object的时候，此元数据也可以一并得到
> - 一个Object可以有多个类似的参数，但所有的User Meta总大小不能超过2KB

上传Object时设置存储类型

BOS支持标准存储, 低频存储和冷存储，上传Object并存储为低频存储类型通过指定StorageClass实现，三种存储类型对应的参数如下：

存储类型	参数
标准存储	STANDARD
低频存储	STANDARD_IA
冷存储	COLD

以低频存储为例，代码如下：

```
public void putObjectStorageClass(){
    PutObjectRequest request = new PutObjectRequest(bucketName, key, file);
    request.withStorageClass(BosClient.STORAGE_CLASS_STANDARD_IA);
    client.putObject(request);
}
```

追加上传

上文介绍的简单上传方式，创建的Object都是Normal类型，用户不可再进行追加写，这在日志、视频监控、视频直播等数据复写较频繁的场景中使用不方便。

正因如此，百度智能云BOS特别支持了AppendObject，即以追加写的方式上传文件。通过AppendObject操作创建的Object类型为Appendable Object，可以对该Object追加数据。AppendObject大小限制为0~5G。

通过AppendObject方式上传示例代码如下，或参考追加上传Demo。

```
public void AppendObject(BosClient client, String bucketName, String objectKey, byte[] byte1, String string1) {
    // 获取指定文件
    File file = new File("/path/to/file.zip");
    // 获取数据流
    InputStream inputStream = new FileInputStream("/path/to/test.zip");

    // 以文件形式上传Object
    AppendObjectResponse appendObjectFromFileResponse = client.appendObject(bucketName, objectKey, file);
    // 以数据流形式上传Object
    AppendObjectResponse appendObjectResponseFromInputStream = client.appendObject(bucketName, objectKey, inputStream);
    // 以二进制串上传Object
    AppendObjectResponse appendObjectResponseFromByte = client.appendObject(bucketName, objectKey, byte1);
    // 以字符串上传Object
    AppendObjectResponse appendObjectResponseFromString = client.appendObject(bucketName, objectKey, string1);

    // 打印ETag
    System.out.println(appendObjectFromFileResponse.getETag());
    // 打印NextAppendOffset
    System.out.println(appendObjectFromFileResponse.getNextAppendOffset());
    // 打印ContentMd5
    System.out.println(appendObjectFromFileResponse.getContentMd5());

    // 追加上传的示例，需要在请求中加上下次追加写的位置
    Long nextAppendOffset = appendObjectFromFileResponse.getNextAppendOffset();
    AppendObjectRequest appendObjectFromFileRequest = new AppendObjectRequest(bucketName,objectKey,file);
    appendObjectFromFileRequest.setOffset(nextAppendOffset);
    AppendObjectResponse appendObjectFromFileResponse = client.appendObject(appendObjectFromFileRequest);
}
```

> **注意：**
>
> 1. 追加上传需要指定追加写的位置，否则已写数据会被覆盖
> 2. 不支持并发，如果确实有多客户端写入的需求，需要自行维护追加写位置，避免冲突

分块上传

除了通过简单上传及追加上传方式将文上传到BOS以外，BOS还提供了另外一种上传模式 —— Multipart Upload。用户可以在如下的应用场景内（但不仅限于此），使用Multipart Upload上传模式，如：

- 需要支持断点上传。
- 上传超过5GB大小的文件。
- 网络条件较差，和BOS的服务器之间的连接经常断开。
- 需要流式地上传文件。
- 上传文件之前，无法确定上传文件的大小。

下面将一步步介绍Multipart Upload的实现。假设有一个文件，本地路径为 `/path/to/file.zip` ，由于文件比较大，将其分块传输到BOS中。或参考分块上传Demo

初始化Multipart Upload

使用 `initiateMultipartUpload` 方法来初始化一个分块上传事件：

```
// 开始Multipart Upload
InitiateMultipartUploadRequest initiateMultipartUploadRequest =
        new InitiateMultipartUploadRequest(bucketName, objectKey);
InitiateMultipartUploadResponse initiateMultipartUploadResponse =
        client.initiateMultipartUpload(initiateMultipartUploadRequest);

// 打印UploadId
System.out.println("UploadId: " + initiateMultipartUploadResponse.getUploadId());
```

`initiateMultipartUpload` 的返回结果中含有 `UploadId` ，它是区分分块上传事件的唯一标识，在后面的操作中，我们将用到它。

- **上传低频存储类型Object的初始化**

初始化低频存储的一个分块上传事件：

```
public void putMultiUploadStorageClass(){
    InitiateMultipartUploadRequest iniReq = new InitiateMultipartUploadRequest(bucketName, key);
    iniReq.withStorageClass(BosClient.STORAGE_CLASS_STANDARD_IA);
    client.initiateMultipartUpload(iniReq);
}
```

- **上传冷存储类型Object的初始化**

初始化低频存储的一个分块上传事件：

```
public void putMultiUploadStorageClass(){
    InitiateMultipartUploadRequest iniReq = new InitiateMultipartUploadRequest(bucketName, key);
    iniReq.withStorageClass(BosClient.STORAGE_CLASS_COLD);
    client.initiateMultipartUpload(iniReq);
}
```

上传分块

接着，把文件分块上传。

```
 // 设置每块为 5MB
final long partSize = 1024 * 1024 * 5L;

File partFile = new File("/path/to/file.zip");

// 计算分块数目
int partCount = (int) (partFile.length() / partSize);
if (partFile.length() % partSize != 0){
    partCount++;
}

// 新建一个List保存每个分块上传后的ETag和PartNumber
List<PartETag> partETags = new ArrayList<PartETag>();

for(int i = 0; i < partCount; i++){
    // 获取文件流
    FileInputStream fis = new FileInputStream(partFile);

    // 跳到每个分块的开头
    long skipBytes = partSize * i;
    fis.skip(skipBytes);

    // 计算每个分块的大小
    long size = partSize < partFile.length() - skipBytes ?
            partSize : partFile.length() - skipBytes;

    // 创建UploadPartRequest，上传分块
    UploadPartRequest uploadPartRequest = new UploadPartRequest();
    uploadPartRequest.setBucketName(bucketName);
    uploadPartRequest.setKey(objectKey);
    uploadPartRequest.setUploadId(initiateMultipartUploadResponse.getUploadId());
    uploadPartRequest.setInputStream(fis);
    uploadPartRequest.setPartSize(size);
    uploadPartRequest.setPartNumber(i + 1);
    UploadPartResponse uploadPartResponse = client.uploadPart(uploadPartRequest);

    // 将返回的PartETag保存到List中。
    partETags.add(uploadPartResponse.getPartETag());

    // 关闭文件
    fis.close();
}
```

上面代码的核心是调用 `UploadPart` 方法来上传每一个分块，但是要注意以下几点：

- UploadPart 方法要求除最后一个Part以外，其他的Part大小都要大于等于**100KB**。但是Upload Part接口并不会立即校验上传Part的大小；只有当Complete Multipart Upload的时候才会校验。
- 为了保证数据在网络传输过程中不出现错误，建议您在`UploadPart`后，使用每个分块BOS返回的Content-MD5值分别验证已上传分块数据的正确性。当所有分块数据合成一个Object后，不再含MD5值。
- Part号码的范围是1~10000。如果超出这个范围，BOS将返回InvalidArgument的错误码。
- 每次上传Part时都要把流定位到此次上传块开头所对应的位置。
- 每次上传Part之后，BOS的返回结果会包含一个 `PartETag` 对象，它是上传块的ETag与块编号（PartNumber）的组合，在后续完成分块上传的步骤中会用到它，因此需要将其保存起来。一般来讲这些 `PartETag` 对象将被保存到List中。

完成分块上传

如下代码所示，完成分块上传：

```
 CompleteMultipartUploadRequest completeMultipartUploadRequest =
        new CompleteMultipartUploadRequest(bucketName, objectKey, initiateMultipartUploadResponse.getUploadId(), partETags);

// 完成分块上传
CompleteMultipartUploadResponse completeMultipartUploadResponse =
        client.completeMultipartUpload(completeMultipartUploadRequest);

// 打印Object的ETag
System.out.println(completeMultipartUploadResponse.getETag());
```

上面代码中的 `partETags` 是第二部中保存的partETag的列表，BOS收到用户提交的Part列表后，会逐一验证每个数据Part的有效性。当所有的数据Part验证通过后，BOS将把这些数据part组合成一个完整的Object。

取消分块上传事件

用户可以使用abortMultipartUpload方法取消分块上传。

```
 AbortMultipartUploadRequest abortMultipartUploadRequest =
        new AbortMultipartUploadRequest(bucketName, objectKey, uploadId);

// 取消分块上传
client.abortMultipartUpload(abortMultipartUploadRequest);
```

获取未完成的分块上传事件

用户可以使用 `listMultipartUploads` 方法获取Bucket内未完成的分块上传事件。

```
 ListMultipartUploadsRequest listMultipartUploadsRequest =
    new ListMultipartUploadsRequest(bucketName);

// 获取Bucket内所有上传事件
ListMultipartUploadsResponse listing = client.listMultipartUploads(listMultipartUploadsRequest);

// 遍历所有上传事件
for (MultipartUploadSummary multipartUpload : listing.getMultipartUploads()) {
    System.out.println("Key: " + multipartUpload.getKey() + " UploadId: " + multipartUpload.getUploadId());
}
```

> **注意：**
>
> 1. 默认情况下，如果Bucket中的分块上传事件的数目大于1000，则只会返回1000个Object，并且返回结果中IsTruncated的值为True，同时返回NextKeyMarker作为下次读取的起点。
> 2. 若想返回更多分块上传事件的数目，可以使用KeyMarker参数分次读取。

获取所有已上传的块信息

用户可以使用 listParts 方法获取某个上传事件中所有已上传的块。

```
ListPartsRequest listPartsRequest = new ListPartsRequest(bucketName, objectKey, uploadId);

// 获取上传的所有Part信息
ListPartsResponse partListing = client.listParts(listPartsRequest);

// 遍历所有Part
for (PartSummary part : partListing.getParts()) {
    System.out.println("PartNumber: " + part.getPartNumber() + " ETag: " + part.getETag());
}
```

如果需要查看Object的存储类型storage class使用以下代码：

```
public void listPartsStorageClass(){
    ListResponse listPartsResponse = client.listParts(bucketName, key, uploadId);
    String storageClass = listPartsResponse.getStorageClass();
}
```

> **注意：**
>
> 1. 默认情况下，如果Bucket中的分块上传事件的数目大于1000，则只会返回1000个Object，并且返回结果中IsTruncated的值为True，同时返回NextPartNumberMarker作为下次读取的起点。
> 2. 若想返回更多分块上传事件的数目，可以使用PartNumberMarker参数分次读取。

封装分块上传

在Java SDK中，Bos为用户提供了putSuperObjectFromFile接口，它对分块上传涉及到的initiateMultipartUpload、UploadPart、completeMultipartUpload三个方法进行封装，用户只需调用该接口即可完成分块上传，

```
File file = new File("/path/to/file.zip");
PutSuperObjectRequest request = new PutSuperObjectRequest(bucketName, objectKey, file);
bosClient.putSuperObjectFromFile(request);
```

其中PutSuperObjectRequest的参数有：

参数	说明
chunkSize	分块大小，默认为5MB，不能小于100KB
nThreads	分块上传中线程池中线程的数量，默认等于CPU的核数
isSuperObjectUploadCanced	是否取消分块上传
File	上传文件

若一个大文件耗时很长，用户想结束分块上传，可调用PutSuperObjectRequest中的cancel()方法设置isSuperObjectUploadCanced为true实现取消分块上传操作。

断点续传上传

当用户向BOS上传大文件时，如果网络不稳定或者遇到程序崩等情况，则整个上传就失败了，失败前已经上传的部分也作废，用户不得不重头再来。这样做不仅浪费资源，在网络不稳定的情况下，往往重试多次还是无法完成上传。 基于上述场景，BOS提供了断点续传上传的能力:

- 当网络情况一般的情况下，建议使用三步上传方式，将object分为5Mb的块，参考分块上传。
- 当您的网络情况非常差，推荐使用appendObject的方式进行断点续传，每次append 较小数据256kb，参考追加上传。

> 提示
>
> - 断点续传是分片上传的封装和加强，是用分片上传实现的；
> - 文件较大或网络环境较差时，推荐使用分片上传；

抓取上传 如下代码用于从指定URL抓取资源，并将资源存储到指定的Bucket中。此操作需要请求者对该Bucket有写权限，每次只能抓取一个Object，且用户可以自定义Object的名称，详情可参考FetchObject接口。

```
String fetchurl = "bj.bcebos.com/sourbucket/sourceobject";
FetchObjectRequest req = new FetchObjectRequest(bucketname, objectname, fetchurl);
req.setReferer("www.referer.com");
bosClient.fetchObject(req);
```

其中FetchObjectRequest的参数有：

参数	说明
source	抓取文件的源地址
mode	抓取模式，支持异步抓取async和同步抓取sync两种模式，可选MODE_SYNC、MODE_ASYNC
storageClass	存储类型，支持标准存储STANDARD、低频存储STANDARD_IA、冷存储COLD和归档存储ARCHIVE，默认存储为STANDARD
callbackAddress	fetch结果回调地址，若选择异步抓取，抓取完成后会将结果推送到该地址
referer	源站fetch时需要透传的Referer头域
userAgent	源站fetch时需要透传的User-Agent头域

获取上传进度

支持在上传过程中实时提供上传进度信息。目前支持PutObject, AppendObject, UploadPart以及PutSuperObjectFromFile四个接口。进度上传接口可以传入自定义数据，非必需。

Java SDK定义的上传进度回调接口如下，您可以在其中定义上传过程中您所需要的操作，如更新界面等等。

```
public interface ProgressCallback<T> {
  void onProgress(long currentSize, long totalSize, T data);
}
```

- PutObject示例代码：

```
BosProgressCallback<Object> callback = new BosProgressCallback<Object>() {
  @Override
  public void onProgress(long currentSize, long totalSize, Object data) {
    // 打印上传进度
    System.out.println("put " + currentSize + "/" + totalSize);
  }
};
// 如果需要在onProgress使用自定义数据
callback.setData(obj);
PutObjectRequest request = new PutObjectRequest(bucketName, objectKey, file);
request.setProgressCallback(callback);
client.putObject(request);
```

- AppendObject示例代码：

```
BosProgressCallback<Object> callback = new BosProgressCallback<Object>() {
  @Override
  public void onProgress(long currentSize, long totalSize, Object data) {
    System.out.println("put " + currentSize + "/" + totalSize);
  }
};
// 如果需要在onProgress使用自定义数据
callback.setData(obj);
AppendObjectRequest request = new AppendObjectRequest(bucketName, objectKey, file);
request.setProgressCallback(callback);
AppendObjectResponse response = client.appendObject(request);
Long nextAppendOffset = response.getNextAppendOffset();
```

- UploadPart示例代码：

```
BosProgressCallback<Object> callback = new BosProgressCallback<Object>() {
  @Override
  public void onProgress(long currentSize, long totalSize, Object data) {
    System.out.println("put " + currentSize + "/" + totalSize);
  }
};
// 如果需要在onProgress使用自定义数据
callback.setData(obj);
UploadPartRequest request = new UploadPartRequest();
request.setBucketName(bucketName);
request.setKey(objectKey);
request.setUploadId(id);
request.setInputStream(inputStream);
request.setPartSize(size);
request.setPartNumber(i);
request.setProgressCallback(callback);
client.uploadPart(request);
```

- PutSuperObjectFromFile示例代码：

```
BosProgressCallback<Object> callback = new BosProgressCallback<Object>() {
  @Override
  public void onProgress(long currentSize, long totalSize, Object data) {
    System.out.println("put " + currentSize + "/" + totalSize);
  }
};
// 如果需要在onProgress使用自定义数据
callback.setData(obj);
PutSuperObjectRequest request = new PutSuperObjectRequest(bucketName, objectKey, file);
request.setProgressCallback(callback);
client.putSuperObjectFromFile(request);
```

单链接限速 对象存储BOS的单Bucket带宽是有限制的，当用户的上传或下载占用带宽达到带宽限制阈值时，会返回RequestRateLimitExceeded的错误码。

为保证用户能够正常使用服务，BOS支持在进行上传、下载等行为时进行流量控制，保证大流量服务占用带宽不会对其他应用服务造成影响。

上传接口支持设置指定限速值，限速值的取值范围为819200~838860800，单位为bit/s，即100KB/s~100MB/s。限速值取值必须为数字，BOS将按照指定的限速值对此次请求进行限速，当限速值不在此范围或不合法时将返回400错误码。

- PutObject示例代码：

```
PutObjectRequest request = new PutObjectRequest(bucketName, objectKey, file);
// 指定上传的限速值为819200bit/s，即100KB/s
request.setTrafficLimitBitPS(819200);
client.putObject(request);
```

- AppendObject示例代码：

```
AppendObjectRequest request = new AppendObjectRequest(bucketName, objectKey, file);
// 指定上传的限速值为819200bit/s，即100KB/s
request.setTrafficLimitBitPS(819200);
AppendObjectResponse response = client.appendObject(request);
Long nextAppendOffset = response.getNextAppendOffset();
```

- PutSuperObjectFromFile示例代码：

```
PutSuperObjectRequest request = new PutSuperObjectRequest(bucketName, objectKey, file);
// 指定上传的限速值为819200bit/s，即100KB/s
request.setTrafficLimitBitPS(819200);
client.putSuperObjectFromFile(request);
```

上传回调 此接口用于BOS在上传文件（Object）时提供同步回调（Callback）给应用服务器。应用服务器指的是用户提供的回调地址对应的服务器。具体描述参考上传回调。Java SDK服务器上传回调使用方式如下：

- PutObject示例代码：

```
PutObjectRequest putObjectRequest = new PutObjectRequest("bucketName","objectKey", file);
String callBackUrl = "[\""+"https://localhost:8080/callback" +"\"]";
String enCodeUrl = "";
try {
    enCodeUrl = Base64.getEncoder().encodeToString(callBackUrl.getBytes(DEFAULT_ENCODING));
}catch (Exception e){
    e.printStackTrace();
}
putObjectRequest.setxBceProcess("callback/callback,u_" + enCodeUrl);
PutObjectResponse putObjectResponse = client.putObject(putObjectRequest);
System.out.println(putObjectResponse.getCallback().getResult());
```

下载文件

BOS Java SDK提供了丰富的文件下载接口，用户可以通过以下方式从BOS中下载文件：

- 简单流式下载
- 下载到本地文件
- 断点续传下载
- 范围下载
- 下载进度条

完整示例代码请参考下载文件Demo

简单流式下载

用户可以通过如下代码将Object读取到一个流中：

```
public void getObject(BosClient client, String bucketName, String objectKey)
  throws IOException {

  // 获取Object，返回结果为BosObject对象
  BosObject object = client.getObject(bucketName, objectKey);

  // 获取ObjectMeta
  ObjectMetadata meta = object.getObjectMetadata();

  // 获取Object的输入流
  InputStream objectContent = object.getObjectContent();

  // 处理Object
  ...

  // 关闭流
  objectContent.close();
}
```

> **注意：**
>
> 1. BosObject中包含了Object的各种信息，包含Object所在的Bucket、Object的名称、MetaData以及一个输入流，用户可以通过操作输入流将Object的内容读取到文件或者内存中。
> 2. ObjectMetadata中包含了Object上传时定义的ETag，Http Header以及自定义的元数据。
> 3. 通过BosObject的getObjectContent方法，还可以获取返回Object的输入流，用户可以读取这个输入流来对Object的内容进行操作。

直接下载Object到文件

用户可以通过如下代码直接将Object下载到指定文件：

```
// 新建GetObjectRequest
GetObjectRequest getObjectRequest = new GetObjectRequest(bucketName, objectKey);

// 下载Object到文件
ObjectMetadata objectMetadata = client.getObject(getObjectRequest, new File("/path/to/file","filename"));
```

当使用上面方法将Object直接下载到文件时，方法返回ObjectMetadata对象。

范围下载

为了实现更多的功能，可以通过使用GetObjectRequest来指定下载范围，实现更精细化地获取Object。如果指定的下载范围是0 - 100，则返回第0到第100个字节的数据，包括第100个，共101字节的数据，即[0, 100]。

```
// 新建GetObjectRequest
GetObjectRequest getObjectRequest = new GetObjectRequest(bucketName, objectKey);

// 获取0~100字节范围内的数据
getObjectRequest.setRange(0, 100);

// 获取Object，返回结果为BosObject对象
BosObject object = client.getObject(getObjectRequest);
```

通过getObjectRequest的setRange方法可以设置返回Object的范围。用户也可以用此功能实现文件的分段下载和断点续传。

下载图像处理后的文件 java sdk支持下载文件的同时进行图像处理。在请求种设置x-bce-process参数即可。具体参数使用参考图像处理。示例代码如下：

```
// 新建GetObjectRequest

GetObjectRequest getObjectRequest = new GetObjectRequest(bucketName, objectKey);

// 设置图像处理为缩放
getObjectRequest.setxBceProcess("image/resize,m_fixed,w_200,h_100");

// 获取Object，返回结果为处理后的BosObject对象
BosObject object = client.getObject(getObjectRequest);
```

其他使用方法

获取Object的存储类型

Object的storage class属性分为STANDARD(标准存储), STANDARD_IA(低频存储)和COLD(冷存储)，通过如下代码可以实现：

```
public void getObjectStorageClass(){
    ObjectMetadata meta = client.getObjectMetadata(bucketName, key);
    String storageClass = meta.getStorageClass();
}
```

只获取ObjectMetadata

通过 getObjectMetadata 方法可以只获取ObjectMetadata而不获取Object的实体。如下代码所示：

```
ObjectMetadata objectMetadata = client.getObjectMetadata(bucketName, objectKey);
```

getObjectMetadata方法返回的解析类中可供调用的参数有：

参数	说明
contentType	Object的类型
contentLength	Object的大小
contentMd5	Object的MD5
etag	Object的HTTP协议实体标签
storageClass	Object的存储类型
userMetadata	如果在PutObject指定了userMetadata自定义meta，则返回此项
xBceCrc	如果在PutObject指定了object的CRC值(循环冗余校验码)，则返回此项

单链接限速

对象存储BOS的单Bucket带宽是有限制的，当用户的上传或下载占用带宽达到带宽限制阈值时，会返回RequestRateLimitExceeded的错误码。

为保证用户能够正常使用服务，BOS支持在进行上传、下载等行为时进行流量控制，保证大流量服务占用带宽不会对其他应用服务造成影响。

下载接口支持设置指定限速值，限速值的取值范围为819200~838860800，单位为bit/s，即100KB/s~100MB/s。限速值取值必须为数字，BOS将按照指定的限速值对此次请求进行限速，当限速值不在此范围或不合法时将返回错误。

- GetObject示例代码：

```
GetObjectRequest getObjectRequest = new GetObjectRequest(bucketName, objectKey);
// 指定下载的限速值为819200bit/s，即100KB/s
getObjectRequest.setTrafficLimitBitPS(819200);
ObjectMetadata objectMetadata = client.getObject(getObjectRequest, new File("/path/to/file","filename"));
```

下载进度条 SDK支持在上传过程中实时提供上传进度信息。Java SDK定义的上传进度回调接口如下，您可以在其中定义上传过程中您所需要的操作，如更新界面等等。

```
public interface ProgressCallback<T> {
    void onProgress(long currentSize, long totalSize, T data);
}
```

- GetObject示例代码：

```
File file = new File("tmp-download");
BosProgressCallback<Object> callback = new BosProgressCallback<Object>() {
    @Override
    public void onProgress(long currentSize, long totalSize, Object data) {
        System.out.println("get " + currentSize + "/" + totalSize);
    }
};
GetObjectRequest getObjectRequest = new GetObjectRequest("bucketName","objectKey");
getObjectRequest.setProgressCallback(callback);
this.client.getObject(getObjectRequest, file);
```

获取文件下载URL

用户可以通过如下代码获取指定Object的URL：

```
public String generatePresignedUrl(BosClient client, String bucketName, String objectKey, int expirationInSeconds) {

  //指定用户需要获取的Object所在的Bucket名称、该Object名称、URL的有效时长
  URL url = client.generatePresignedUrl( < bucketName > , < objectKey > , <expirationInSeconds>);


  return url.toString();
}
```

说明：

- 用户在调用该函数前，需要手动设置endpoint为所属区域域名。百度智能云目前开放了多区域 支持，请参考Region与Endpoint。
- expirationInSeconds为指定的URL有效时长，时间从当前时间算起，为可选参数，不配置时系统默认值为1800秒。如果要设置为永久不失效的时间，可以将expirationInSeconds参数设置为 -1，不可设置为其他负数。
- 如果预期获取的文件时公共可读的，则对应URL链接可通过简单规则快速拼接获取: http://$bucketName.$region.bcebos.com/$objectKey或者http://$region.bcebos.com/$bucketName/$objectKey

支持通过STS方式获取下载URL，配置STS访问BOS见初始化页面，代码示例如下：

```
public String generatePresignedUrl(String bucketName, String objectKey, int expirationInSeconds) {
  // 获取STS
  BceCredentials credentials = new DefaultBceCredentials(BOS_AK, BOS_SK);
  StsClient stsClient = new StsClient(
      new BceClientConfiguration().withEndpoint("http://sts.bj.baidubce.com").withCredentials(credentials)
  );
  GetSessionTokenResponse response = stsClient.getSessionToken(new GetSessionTokenRequest());

  // 生成bosClient
  BceCredentials bosstsCredentials = new DefaultBceSessionCredentials(
      response.getAccessKeyId(),
      response.getSecretAccessKey(),
      response.getSessionToken());
  BosClientConfiguration config = new BosClientConfiguration();
  config.setCredentials(bosstsCredentials);
  config.setEndpoint(ENDPOINT);
  BosClient bosClient = new BosClient(config);

  // 指定用户需要获取的Object所在的Bucket名称、该Object名称、URL的有效时长
  URL url = bosClient.generatePresignedUrl(bucketName, objectKey, expirationInSeconds);

  return url.toString();
}
```

列举存储空间中的文件

BOS SDK支持用户通过以下两种方式列举出object：

- 简单列举
- 通过参数复杂列举

除此之外，用户还可在列出文件的同时模拟文件夹，完整示例代码请参考列举文件Demo

简单列举

当用户希望简单快速列举出所需的文件时，可通过listObjects方法返回ListObjectsResponse对象，ListObjectsResponse对象包含了此次listObject请求的返回结果。用户可以通过ListObjectsResponse中的getContents方法获取所有Object的描述信息。

```
public void listObjects(BosClient client, String bucketName) {

  // 获取指定Bucket下的所有Object信息
  ListObjectsResponse listing = client.listObjects(bucketName);

  // 遍历所有Object
  for (BosObjectSummary objectSummary : listing.getContents()) {
      System.out.println("ObjectKey: " + objectSummary.getKey());
  }

}
```

注意：

1. 默认情况下，如果Bucket中的Object数量大于1000，则只会返回1000个Object，并且返回结果中IsTruncated值为True，并返回NextMarker做为下次读取的起点。
2. 若想增大返回Object的数目，可以使用Marker参数分次读取。

通过参数复杂列举

除上述简单列举外，用户还可通过设置ListObjectsRequest的参数实现各种灵活的查询功能。ListObjectsRequest的可设置的参数如下：

参数	功能	使用方式
Prefix	限定返回的object key必须以prefix作为前缀	setPrefix(String prefix)
Delimiter	是一个用于对Object名字进行分组的字符所有名字包含指定的前缀且第一次出现。Delimiter字符之间的Object作为一组元素: CommonPrefixes	setDelimiter(String delimiter)
Marker	设定结果从marker之后按字母排序的第一个开始返回	setMarker(String marker)
MaxKeys	限定此次返回object的最大数，最大为1000，默认值是1000。如果指定的值大于1000，按1000操作	setMaxKeys(int maxKeys)

> **注意：**
>
> 1. 如果有Object以Prefix命名，当仅使用Prefix查询时，返回的所有Key中仍会包含以Prefix命名的Object。
> 2. 如果有Object以Prefix命名，当使用Prefix和Delimiter组合查询时，返回的所有Key中会有Null，Key的名字不包含Prefix前缀。

下面我们分别以几个案例说明通过参数列举的方法：

指定最大返回条数

```
// 指定最大返回条数为500
ListObjectsRequest listObjectsRequest = new ListObjectsRequest("bucketName");
listObjectsRequest.withMaxKeys(500);
ListObjectsResponse listObjectsResponse = client.listObjects(listObjectsRequest);
for(BosObjectSummary objectSummary :listObjectsResponse.getContents()) {
    System.out.println("ObjectKey:" + objectSummary.getKey());
}
```

返回指定前缀的object

```
// 指定返回前缀为test的object
ListObjectsRequest listObjectsRequest = new ListObjectsRequest("bucketName");
listObjectsRequest.withPrefix("test");
ListObjectsResponse listObjectsResponse = client.listObjects(listObjectsRequest);
for(BosObjectSummary objectSummary :listObjectsResponse.getContents()) {
    System.out.println("ObjectKey:" + objectSummary.getKey());
}
```

从指定Object后返回

```
// 用户可以定义不包括某object，从其之后开始返回
ListObjectsRequest listObjectsRequest = new ListObjectsRequest("bucketName");
listObjectsRequest.withMarker("object");
ListObjectsResponse listObjectsResponse = client.listObjects(listObjectsRequest);
for(BosObjectSummary objectSummary :listObjectsResponse.getContents()) {
    System.out.println("ObjectKey:" + objectSummary.getKey());
}
```

分页获取所有Object

```
// 用户可设置每页最多500条记录
ListObjectsRequest listObjectsRequest = new ListObjectsRequest("bucketName");
listObjectsRequest.withMaxKeys(500);
ListObjectsResponse listObjectsResponse;
boolean isTruncated = true;
while (isTruncated) {
    listObjectsResponse = client.listObjects(listObjectsRequest);
    isTruncated = listObjectsResponse.isTruncated();
    if (listObjectsResponse.getNextMarker() != null) {
        listObjectsRequest.withMarker(listObjectsResponse.getNextMarker());
    }
}
```

分页获取所有特定Object后的结果

```
// 用户可设置每页最多500条记录，并从某特定object之后开始获取
ListObjectsRequest listObjectsRequest = new ListObjectsRequest("bucketName");
listObjectsRequest.withMaxKeys(500);
listObjectsRequest.withMarker("object");
ListObjectsResponse listObjectsResponse;
boolean isTruncated = true;
while (isTruncated) {
    listObjectsResponse = client.listObjects(listObjectsRequest);
    isTruncated = listObjectsResponse.isTruncated();
    if (listObjectsResponse.getNextMarker() != null) {
        listObjectsRequest.withMarker(listObjectsResponse.getNextMarker());
    }
}
```

分页获取所有指定前缀的Object结果

```
// 用户可设置分页获取指定前缀的Object，每页最多500条记录
ListObjectsRequest listObjectsRequest = new ListObjectsRequest("bucketName");
listObjectsRequest.withMaxKeys(500);
listObjectsRequest.withPrefix("object");
ListObjectsResponse listObjectsResponse;
boolean isTruncated = true;
while (isTruncated) {
    listObjectsResponse = client.listObjects(listObjectsRequest);
    isTruncated = listObjectsResponse.isTruncated();
    if (listObjectsResponse.getNextMarker() != null) {
        listObjectsRequest.withMarker(listObjectsResponse.getNextMarker());
    }
}
```

listObject方法返回的解析类中可供调用的参数有：

参数	说明
name	Bucket名称
prefix	匹配以prefix开始到第一次出现Delimiter字符之间的object作为一组元素返回
marker	本次查询的起点
maxKeys	请求返回的最大数目
isTruncated	指明是否所有查询都返回了；false-本次已经返回所有结果，true-本次还没有返回所有结果
contents	返回的一个Object的容器
+key	Object名称
+lastModified	此Object最后一次被修改的时间
+eTag	Object的HTTP协议实体便签
+storageClass	Object的存储形态
+size	Object的内容的大小（字节数）
+owner	Object对应Bucket所属用户信息
++id	Bucket Owner的用户ID
++displayName	Bucket Owner的名称

模拟文件夹功能

在BOS的存储结果中是没有文件夹这个概念的，所有元素都是以Object来存储，但BOS的用户在使用数据时往往需要以文件夹来管理文件。 因此，BOS提供了创建模拟文件夹的能力，其本质上来说是创建了一个size为0的Object。对于这个Object可以上传下载，只是控制台会对以"/"结尾的Object以文件夹的方式展示。

用户可以通过 Delimiter 和 Prefix 参数的配合模拟出文件夹功能。Delimiter 和 Prefix 的组合效果是这样的：

如果把 Prefix 设为某个文件夹名，就可以罗列以此 Prefix 开头的文件，即该文件夹下递归的所有的文件和子文件夹（目录）。文件名在Contents中显示。 如果再把 Delimiter 设置为 "/" 时，返回值就只罗列该文件夹下的文件和子文件夹（目录），该文件夹下的子文件名（目录）返回在 CommonPrefixes 部分，子文件夹下递归的文件和文件夹不被显示。

如下是几个应用方式：

列出Bucket内所有文件

当用户需要获取Bucket下的所有文件时，可以参考分页获取所有Object

递归列出目录下所有文件

可以通过设置 `Prefix` 参数来获取某个目录下所有的文件：

```
 // 构造ListObjectsRequest请求
ListObjectsRequest listObjectsRequest = new ListObjectsRequest(bucketName);

// 递归列出fun目录下的所有文件
listObjectsRequest.setPrefix("fun/");

ListObjectsResponse listing = client.listObjects(listObjectsRequest);

// 遍历所有Object
System.out.println("Objects:");
for (BosObjectSummary objectSummary : listing.getContents()) {
    System.out.println(objectSummary.getKey());
 }
}
```

输出：

```
 Objects:
fun/
fun/movie/001.avi
fun/movie/007.avi
fun/test.jpg
```

查看目录下的文件和子目录

在 `Prefix` 和 `Delimiter` 结合的情况下，可以列出目录下的文件和子目录：

```
 // 构造ListObjectsRequest请求
ListObjectsRequest listObjectsRequest = new ListObjectsRequest(bucketName);

// "/" 为文件夹的分隔符
listObjectsRequest.setDelimiter("/");

// 列出fun目录下的所有文件和文件夹
listObjectsRequest.setPrefix("fun/");

ListObjectsResponse listing = client.listObjects(listObjectsRequest);

// 遍历所有Object
System.out.println("Objects:");
for (BosObjectSummary objectSummary : listing.getContents()) {
    System.out.println(objectSummary.getKey());
}

// 遍历所有CommonPrefix
System.out.println("\nCommonPrefixs:");
for (String commonPrefix : listing.getCommonPrefixes()) {
    System.out.println(commonPrefix);
}
```

输出：

```
 Objects:
fun/
fun/test.jpg

CommonPrefixs:
fun/movie/
```

返回的结果中， ObjectSummaries 的列表中给出的是fun目录下的文件。而 CommonPrefixs 的列表中给出的是fun目录下的所有子文件夹。可以看出 fun/movie/001.avi ， fun/movie/007.avi 两个文件并没有被列出来，因为它们属于 fun 文件夹下的 movie 目录。

列举Bucket中object的存储属性

当用户完成上传后，如果需要查看指定Bucket中的全部Object的storage class属性，可以通过如下代码实现：

```
public void listObjectsStorageClass(){
    ListObjectsResponse listObjectResponse = client.listObjects("bucketName");
    List<BosObjectSummary> objectList = listObjectResponse.getContents();
    for(int i=0; i<objectList.length(); i++) {
        System.out.println(objectList[i].getStorageClass());
    }
}
```

查看文件是否存在

用户可通过如下操作查看某文件是否存在：

```
 // 存在返回true，不存在返回false，其他异常会被抛出
bool exists = client.doesObjectExist("bucketName", "objectKey");
```

获取及更新文件元信息

文件元信息(Object Metadata)，是对用户上传BOS的文件的属性描述，分为两种：HTTP标准属性（HTTP Headers）和User Meta（用户自定义元信息）。

获取文件元信息

通过 getObjectMetadata 方法可以只获取ObjectMetadata而不获取Object的实体，参考Demo。

如下代码所示：

```
 ObjectMetadata objectMetadata = client.getObjectMetadata(bucketName, objectKey);
// 查看object元信息
System.out.println("contentType: " + objectMetadata.getContentType() + "\n" +
        "contentLength: " + objectMetadata.getContentLength() + "\n" +
        "contentMd5: " + objectMetadata.getContentMd5() + "\n" +
        "etag: " + objectMetadata.getETag() + "\n" +
        "storageClass: " + objectMetadata.getStorageClass() + "\n");
```

getObjectMetadata方法返回的解析类中可供调用的参数有：

参数	说明
contentType	Object的类型
contentLength	Object的大小
contentMd5	Object的MD5
etag	Object的HTTP协议实体标签
storageClass	Object的存储类型
userMetadata	如果在PutObject指定了userMetadata自定义meta，则返回此项
xBceCrc	如果在PutObject指定了object的CRC值(循环冗余校验码)，则返回此项

修改文件元信息

BOS修改Object的Metadata通过拷贝Object实现。即拷贝Object的时候，把目的Bucket设置为源Bucket，目的Object设置为源Object，并设置新的Metadata，通过拷贝自身实现修改Metadata的目的。如果不设置新的Metadata，则报错。

```
public void setObjectMeta(BosClient client, String bucketName, String objectKey, ObjectMetadata newObjectMetadata) {

    CopyObjectRequest request = new CopyObjectRequest(bucketName, objectKey, bucketName, objectKey);

    // 设置新的ObjectMetadata
    request.setNewObjectMetadata(newObjectMetadata);

    // 拷贝Object
    CopyObjectResponse copyObjectResponse = client.copyObject(request);

    // 打印结果
    System.out.println("ETag: " + copyObjectResponse.getETag() + " LastModified: " + copyObjectResponse.getLastModified());
}
```

变更文件存储等级

上文中已提到，BOS支持为文件赋予STANDARD(标准存储), STANDARD_IA(低频存储)和COLD(冷存储)三种存储类型。同时，BOS java SDK也支持用户对特定文件执行存储类型变更的操作。 涉及到的参数如下：

参数	说明
x-bce-storage-class	指定Object的存储类型，STANDARD_IA代表低频存储，COLD代表冷存储，不指定时默认是标准存储类型。

示例如下：

```
// 标准存储转为低频存储
CopyObjectRequest copyObjectRequest = new CopyObjectRequest(srcBucketName, srcKey, destBucketName, destKey);
copyObjectRequest.setStorageClass("STANDARD_IA");
client.copyObject(copyObjectRequest);

// 低频存储转为冷存储
CopyObjectRequest copyObjectRequest = new CopyObjectRequest(srcBucketName, srcKey, destBucketName, destKey);
copyObjectRequest.setStorageClass("COLD");
client.copyObject(copyObjectRequest);
```

取回归档存储类型文件

取回归档类型文件

用户上传归档文件后，归档文件是处于冰冻状态的，如果要下载归档文件，必须先取回归档存储文件。请求者必须有归档存储文件的读权限，并且归档存储文件处于冰冻状态。

取回归档文件，示例如下：

```
RestoreObjectRequest request = new RestoreObjectRequest(bucketName, objectKey);
request.setRestoreDays(10);
// 加急取回（RESTORE_TIER_EXPEDITED）、标准取回（RESTORE_TIER_STANDARD）、延缓取回（RESTORE_TIER_LOWCOST）
request.setRestoreTier(RESTORE_TIER_EXPEDITED);
client.restoreObject(request);
```

参数	类型	说明	是否必须
days	int	表示解冻后状态的持续时间，要求是数字，单位是天，取值大于0且不超过30即可。默认解冻持续天数是7天,最长30天。	否
tier	string	解冻优先级。可取值范围如下： • Expedited：加急取回，表示30min内完成取回。 • Standard：标准取回，表示2~5小时内完成取回 • LowCost：延缓取回，表示12小时内完成取回	否

Object权限控制

设置Object的访问权限

目前BOS支持两种方式设置ACL。第一种是使用Canned Acl，在PutObjectAcl的时候，通过头域的"x-bce-acl"或者"x-bce-grant-permission'来设置object访问权限，当前可设置的权限包括private和public-read，两种类型的header不可以同时在一个请求中出现。第二种方式是上传一个ACL文件。

详细信息请参考设置Object权限控制，或查看Demo样例。

1、通过使用头域的"x-bce-acl"或者"x-bce-grant-permission'来设置object访问权限

- set object acl 第一种方式(以请求头方式设置)

```
SetObjectAclRequest setObjectAclRequest = new SetObjectAclRequest("yourBucketName","objectKey",CannedAccessControlList.PublicRead);
client.setObjectAcl(setObjectAclRequest);
```

- set object acl 第一种方式(以请求头方式设置 xBceGrantRead)

```
String xBceGrantRead = "id=\"user_id1\""+",id=\"user_id2\"";
SetObjectAclRequest setObjectAclRequest = new SetObjectAclRequest();
setObjectAclRequest.withBucketName("yourBucketName");
setObjectAclRequest.withKey("objectKey");
setObjectAclRequest.setxBceGrantRead(xBceGrantRead);
client.setObjectAcl(setObjectAclRequest);
```

- set object acl 第一种方式(以请求头方式设置 xBceGrantFullControl)

```
String xBceGrantFullControl = "id=\"user_id1\""+",id=\"user_id2\"";
SetObjectAclRequest setObjectAclRequest = new SetObjectAclRequest();
setObjectAclRequest.withBucketName("yourBucketName");
setObjectAclRequest.withKey("objectKey");
setObjectAclRequest.setxBceGrantFullControl(xBceGrantFullControl);
client.setObjectAcl(setObjectAclRequest);
```

2、通过setObjectAcl设置object访问权限

- set object acl 第二种方式(json字符串)

```
String jsonObjectAcl = "{\"accessControlList\":["+ "{\"grantee\":[{\"id\":\"*\"}], "+ "\"permission\":[\"FULL_CONTROL\"]"+"}]}";

SetObjectAclRequest setObjectAclRequest = new SetObjectAclRequest("yourBucketName","objectKey",jsonObjectAcl);

client.setObjectAcl(setObjectAclRequest);
```

- set object acl 第二种方式，用户只需指定指定参数即可

```
 List<Grant> grants = new ArrayList<Grant>();
List<Grantee> grantees = new ArrayList<Grantee>();
List<Permission> permissions = new ArrayList<Permission>();

// 授权给特定用户
grantees.add(new Grantee("user_id1"));
grantees.add(new Grantee("user_id2"));
grantees.add(new Grantee("user_id3"));

// 设置权限
permissions.add(Permission.READ);
grants.add(new Grant().withGrantee(grantees).withPermission(permissions));

SetObjectAclRequest setObjectAclRequest = new SetObjectAclRequest("yourBucketName","objectKey", grants);
client.setObjectAcl(setObjectAclRequest);
```

查看Object的权限

如下代码可以查Object的权限：

```
 GetObjectAclRequest getObjectRequest = new GetObjectAclRequest();
getObjectRequest.withBucketName("yourBucketName");
getObjectRequest.withKey("objectKey");
GetObjectAclResponse response = client.getObjectAcl(getObjectRequest);
```

`getObjectAcl`方法返回的解析类中可供调用的参数有：

参数	说明
accessControlList	标识Object的权限列表
grantee	标识被授权人
-id	被授权人ID
permission	标识被授权人的权限

删除Object的权限

如下代码可以删除Object的权限：

```
 DeleteObjectAclRequest deleteObjectAclRequest = new DeleteObjectAclRequest("yourBucketName","objectKey");

client.deleteObjectAcl(deleteObjectAclRequest);
```

获取目录容量概览

用户可以通过如下代码获取Bucket或指定前缀下的当前容量、Object数、文件数。 此处文件的含义就是指普通的文件，不包括文件夹，更不包括分隔符逻辑意义上的目录。object就是实际存储在bucket的key数量。即文件数等于object数减去带分隔符结尾的object数，比如，当前目录下有9个object，其中5个是文件夹，4个文件，那么查到的文件数就是4，object数就是9。

```
 public void stateSummary(BosClient client, String bucketName, String prefix) {

   // 获取Bucket下的当前容量、Object数、文件数
   StateSummaryResponse response = client.stateSummary(bucketName);
   System.out.println("currSize : " + response.getTotalSize());
   System.out.println("objectsCount : " + response.getObjectsCount());
   System.out.println("filesCount : " + response.getFilesCount());

}
```

获取带有指定前缀的object容量、Object数、文件数：

```
 public StateSummaryResponse stateSummaryWithPrefix(BosClient client, String bucketName, String prefix) {

   StateSummaryRequest request = new StateSummaryRequest();
   request.setBucketName(this.bucketName);
   request.setPrefix("dir/x");
   StateSummaryResponse response = client.stateSummary(request);

   return response;
}
```

拷贝文件

示例代码可以参考拷贝文件Demo

拷贝一个文件

用户可以通过copyObject方法拷贝一个Object，如下代码所示：

```
 public void copyObject(BosClient client, String srcBucketName, String srcKey, String destBucketName, String destKey) {

   // 拷贝Object
   CopyObjectResponse copyObjectResponse = client.copyObject(srcBucketName, srcKey, destBucketName, destKey);

   // 打印结果
   System.out.println("ETag: " + copyObjectResponse.getETag() + " LastModified: " + copyObjectResponse.getLastModified());
}
```

copyObject 方法返回一个 `CopyObjectResponse` 对象，该对象中包含了新Object的ETag和修改时间。

通过CopyObjectRequest拷贝Object

用户也可以通过 `CopyObjectRequest` 实现Object的拷贝，如下代码所示：

```
 // 初始化BosClient
BosClient client = ...;

// 创建CopyObjectRequest对象
CopyObjectRequest copyObjectRequest = new CopyObjectRequest(srcBucketName, srcKey, destBucketName, destKey);

// 设置新的Metadata
Map<String, String> userMetadata = new HashMap<String, String>();
userMetadata.put('<user-meta-key>','<user-meta-value>');
ObjectMetadata meta = new ObjectMetadata();
meta.setUserMetadata(userMetadata);
copyObjectRequest.setNewObjectMetadata(meta);

// 复制Object
CopyObjectResponse copyObjectResponse = client.copyObject(copyObjectRequest);

System.out.println("ETag: " + copyObjectResponse.getETag() + " LastModified: " + copyObjectResponse.getLastModified());
```

`CopyObjectRequest` 允许用户修改目的Object的ObjectMeta，同时也提供 `MatchingETagConstraints` 参数的设定。

设置Object的Copy属性

BOS同时会提供CopyObject接口用于将一个已经存在的Object拷贝到另外一个Object，拷贝过程中会对源Object的Etag或修改状态进行判断，根据判断结果决定是否执行拷贝。详细的参数解释如下：

名称	类型	描述	是否必需
x-bce-copy-source-if-match	String	如果源Object的ETag值和用户提供的ETag相等，则执行拷贝操作，否则拷贝失败。	否
x-bce-copy-source-if-none-match	String	如果源Object的ETag和用户提供的ETag不相等，则执行拷贝操作，否则拷贝失败。	否
x-bce-copy-source-if-unmodified-since	String	如果源object在x-bce-copy-source-if-unmodified-since之后没被修改，则执行拷贝操作，否则拷贝失败。	否
x-bce-copy-source-if-modified-since	String	如果源object在x-bce-copy-source-if-modified-since之后被修改了，则执行拷贝操作，否则拷贝失败。	否

对应的示例代码：

```
 // 初始化BosClient
BosClient client = ...;

// 创建CopyObjectRequest对象
CopyObjectRequest copyObjectRequest = new CopyObjectRequest(srcBucketName, srcKey, destBucketName, destKey);

// 设置新的Metadata
Map<String, String> userMetadata = new HashMap<String, String>();
userMetadata.put("<user-meta-key>","<user-meta-value>");

meta.setUserMetadata(userMetadata);
copyObjectRequest.setNewObjectMetadata(meta);

//copy-source-if-match
copyObjectRequest.withETag("111111111183bf192b57a4afc76fa632");
//copy-source-if-none-match
copyObjectRequest.withNoMatchingETagConstraint("111111111183bf192b57a4afc76fa632");

Date modifiedSinceConstraint = new Date();
SimpleDateFormat df = new SimpleDateFormat("E, dd MMM yyyy HH:mm:ss z", Locale.UK);
df.setTimeZone(new java.util.SimpleTimeZone(0, "GMT"));
String date = df.format(modifiedSinceConstraint);

//copy-source-if-modified-since
copyObjectRequest.withModifiedSinceConstraint(date);

//copy-source-if-unmodified-since
copyObjectRequest.withUnmodifiedSinceConstraint(date);

// 复制Object
CopyObjectResponse copyObjectResponse = client.copyObject(copyObjectRequest);

System.out.println("ETag: " + copyObjectResponse.getETag() + " LastModified: " + copyObjectResponse.getLastModified());
```

同步Copy功能

当前BOS的CopyObject接口是通过同步方式实现的。同步方式下，BOS端会等待Copy实际完成才返回成功。同步Copy能帮助用户更准确的判断Copy状态，但用户感知的复制时间会变长，且复制时间和文件大小成正比。

同步Copy方式更符合业界常规，提升了与其它平台的兼容性。同步Copy方式还简化了BOS服务端的业务逻辑，提高了服务效率。

如果您使用bce-java-sdk-0.10.8版本之前的SDK，有可能会出现复制请求成功，但文件实际复制失败的情况，所以建议您使用最新版本SDK。

分块拷贝

除了通过CopyObject接口拷贝文件以外，BOS还提供了另外一种拷贝模式——Multipart Upload Copy。用户可以在如下的应用场景内（但不仅限于此），使用Multipart Upload Copy，如：

- 需要支持断点拷贝。
- 拷贝超过5GB大小的文件。
- 网络条件较差，和BOS的服务器之间的连接经常断开。

下面将介绍分步实现三步拷贝。

三步拷贝包含init、“拷贝分块”和complete三步，其中init和complete的操作同分块上传一致。

为了便于理解，下面提供三步拷贝完整代码：

```
 // 第一步 init
InitiateMultipartUploadRequest initiateMultipartUploadRequest =
        new InitiateMultipartUploadRequest("targetBucketName","targetObjectName");
InitiateMultipartUploadResponse initiateMultipartUploadResponse =
        client.initiateMultipartUpload(initiateMultipartUploadRequest);

// 第二步 分块拷贝
long left_size=client.getObjectMetadata("sourceBucketName","sourceObjectName").getContentLength();
long skipBytes = 0;
int partNumber = 1;
List<PartETag> partETags = new ArrayList<PartETag>();

while (left_size > 0) {
    long partSize = 1024 * 1024 * 1L;
    if (left_size < partSize) {
        partSize = left_size;
    }
    UploadPartCopyRequest uploadPartCopyRequest = new UploadPartCopyRequest();
    uploadPartCopyRequest.setBucketName("targetBucketName");
    uploadPartCopyRequest.setKey("targetObjectName");
    uploadPartCopyRequest.setSourceBucketName("sourceBucketName");
    uploadPartCopyRequest.setSourceKey("sourceObjectName");
    uploadPartCopyRequest.setUploadId(initiateMultipartUploadResponse.getUploadId());
    uploadPartCopyRequest.setPartSize(partSize);
    uploadPartCopyRequest.setOffSet(skipBytes);
    uploadPartCopyRequest.setPartNumber(partNumber);
    UploadPartCopyResponse uploadPartCopyResponse = client.uploadPartCopy(uploadPartCopyRequest);
    // 将返回的PartETag保存到List中
    PartETag partETag = new PartETag(partNumber.uploadPartCopyResponse.getETag());
    partETags.add(partETag);
    left_size -= partSize;
    skipBytes += partSize;
    partNumber+=1;
}

// 第三步 complete
CompleteMultipartUploadRequest completeMultipartUploadRequest =
        new CompleteMultipartUploadRequest("targetBucketName", "targetObjectName", initiateMultipartUploadResponse.getUploadId(), partETags);
CompleteMultipartUploadResponse completeMultipartUploadResponse =
        client.completeMultipartUpload(completeMultipartUploadRequest);
```

注意:

1. offset参数以字节为单位，为分块的开始偏移位置。
2. size参数以字节为单位，定义每个分块的大小，除最后一个Part以外，其他的Part大小都要大于 5MB。

选取文件

BOS Java SDK提供了SelectObject接口，用于向Bucket中指定object执行SQL语句，选取出指定内容返回，请参考选取Object。目前支持object类型为CSV（包括TSV等类CSV文件）、JSON文件和Parquet文件：

示例代码可以参考选取文件Demo

- 选取CSV文件
- 选取JSON文件

选取CSV文件

Java SDK选取CSV文件请参考以下代码：

```
 final String csvContent = "header1,header2,header3\r\n" +
                    "1,2,3.4\r\n" +
                    "a,b,c\r\n" +
                    "\"d\",\"e\",\"f\"\r\n" +
                    "true,false,true\r\n" +
                    "2006-01-02 15:04:06,2006-01-02 16:04:06,2006-01-02 17:04:06";
client.putObject("bucketName", "test-csv", new ByteArrayInputStream(csvContent.getBytes()));

SelectObjectRequest request = new SelectObjectRequest("bucketName", "test-csv")
        .withSelectType("csv")
        .withExpression("select * from BosObject limit 3")
        .withInputSerialization(new InputSerialization()
                .withCompressionType("NONE")
                .withFileHeaderInfo("NONE")
                .withRecordDelimiter("\r\n")
                .withFieldDelimiter(",")
                .withQuoteCharacter("\"")
                .withCommentCharacter("#"))
        .withOutputSerialization(new OutputSerialization()
                .withOutputHeader(false)
                .withQuoteFields("ALWAYS")
                .withRecordDelimiter("\n")
                .withFieldDelimiter(",")
                .withQuoteCharacter("\""))
        .withRequestProgress(false);
SelectObjectResponse response = client.selectObject(request);

// 输出返回的记录
SelectObjectResponse.Messages messages = response.getMessages();
while (messages.hasNext()) {
    SelectObjectResponse.CommonMessage message = messages.next();
    if (message.Type.equals("Records")) {
        for (String record: message.getRecords()) {
            System.out.println(record);
        }
    }
}
```

选取CSV文件输出的结果：

```
 "header1","header2","header3"
"1","2","3.4"
"a","b","c"
```

> **注意：**
>
> - Unix/Linux系统里，每行结尾只有"<换行>"，即"\n"；
> - Windows系统里面，每行结尾是"<回车><换行>"，即"\r\n"；
> - Mac系统里，每行结尾是"<换行>"，即"\n"，只有 v9 之前 Mac OS 才是用 '\r'。
> - 根据文件内容，设置合适的recordDelimiter。

选取JSON文件

Java SDK选取JSON文件请参考以下代码：

```
final String jsonContent = "{\n" +
        "\t\"name\": \"Smith\",\n" +
        "\t\"age\": 16,\n" +
        "\t\"org\": null\n" +
        "}\n" +
        "{\n" +
        "\t\"name\": \"charles\",\n" +
        "\t\"age\": 27,\n" +
        "\t\"org\": \"baidu\"\n" +
        "}\n" +
        "{\n" +
        "\t\"name\": \"jack\",\n" +
        "\t\"age\": 35,\n" +
        "\t\"org\": \"bos\"\n" +
        "}";
client.putObject("bucketName", "test-json", new ByteArrayInputStream(jsonContent.getBytes()));

SelectObjectRequest request = new SelectObjectRequest("bucketName", "test-json")
        .withSelectType("json")
        .withExpression("select * from BosObject where age > 20")
        .withInputSerialization(new InputSerialization()
                .withCompressionType("NONE")
                .withJsonType("LINES"))
        .withOutputSerialization(new OutputSerialization()
                .withRecordDelimiter("\n"))
        .withRequestProgress(false);
SelectObjectResponse response = client.selectObject(request);

// 输出返回的记录
SelectObjectResponse.Messages messages = response.getMessages();
while (messages.hasNext()) {
    SelectObjectResponse.CommonMessage message = messages.next();
    if (message.Type.equals("Records")) {
        for (String record: message.getRecords()) {
            System.out.println(record);
        }
    }
}
```

选取JSON文件输出的结果：

```
{"name":"charles","age":27,"org":"baidu"}
{"name":"jack","age":35,"org":"bos"}
```

选取Parquet文件

Java SDK选取Parquet文件请参考以下代码：

```
/*
parquet文件解析内容
{"Name":"StudentName","Age":20,"Id":0,"Weight":50,"Sex":true,"Day":19240,"Scores":{"computer":80,"math":90,"physics":90}}
{"Name":"StudentName","Age":21,"Id":1,"Weight":50.1,"Sex":false,"Day":19240,"Scores":{"computer":81,"math":91,"physics":91}}
{"Name":"StudentName","Age":22,"Id":2,"Weight":50.2,"Sex":true,"Day":19240,"Scores":{"computer":82,"math":92,"physics":92}}
{"Name":"StudentName","Age":23,"Id":3,"Weight":50.3,"Sex":false,"Day":19240,"Scores":{"computer":83,"math":93,"physics":90}}
{"Name":"StudentName","Age":24,"Id":4,"Weight":50.4,"Sex":true,"Day":19240,"Scores":{"computer":84,"math":94,"physics":91}}
{"Name":"StudentName","Age":20,"Id":5,"Weight":50.5,"Sex":false,"Day":19240,"Scores":{"computer":85,"math":90,"physics":92}}
{"Name":"StudentName","Age":21,"Id":6,"Weight":50.6,"Sex":true,"Day":19240,"Scores":{"computer":86,"math":91,"physics":90}}
{"Name":"StudentName","Age":22,"Id":7,"Weight":50.7,"Sex":false,"Day":19240,"Scores":{"computer":87,"math":92,"physics":91}}
{"Name":"StudentName","Age":23,"Id":8,"Weight":50.8,"Sex":true,"Day":19240,"Scores":{"computer":88,"math":93,"physics":92}}
{"Name":"StudentName","Age":24,"Id":9,"Weight":50.9,"Sex":false,"Day":19240,"Scores":{"computer":89,"math":94,"physics":90}}
*/

SelectObjectRequest request = new SelectObjectRequest("bucketName", "test-parquet")
        .withSelectType("parquet")
        .withExpression("select * from BosObject s where s.Scores.computer > 85")
        .withInputSerialization(new InputSerialization()
                .withCompressionType("NONE"))
        .withOutputSerialization(new OutputSerialization()
                .withRecordDelimiter("\n"))
        .withRequestProgress(false);
SelectObjectResponse response = client.selectObject(request);

// 输出返回的记录
SelectObjectResponse.Messages messages = response.getMessages();
while (messages.hasNext()) {
    SelectObjectResponse.CommonMessage message = messages.next();
    if (message.Type.equals("Records")) {
        for (String record: message.getRecords()) {
            System.out.println(record);
        }
    }
}
```

选取Parquet文件输出的结果：

```
{"Name":"StudentName","Age":21,"Id":6,"Weight":50.6,"Sex":true,"Day":19240,"Scores":{"computer":86,"math":91,"physics":90}}
{"Name":"StudentName","Age":22,"Id":7,"Weight":50.7,"Sex":false,"Day":19240,"Scores":{"computer":87,"math":92,"physics":91}}
{"Name":"StudentName","Age":23,"Id":8,"Weight":50.8,"Sex":true,"Day":19240,"Scores":{"computer":88,"math":93,"physics":92}}
{"Name":"StudentName","Age":24,"Id":9,"Weight":50.9,"Sex":false,"Day":19240,"Scores":{"computer":89,"math":94,"physics":90}}
```

注意，对于CSV、JSON和Parquet文件查询时，初始化SelectObjectRequest的参数有很大不同，详细参数设置请参考SelectObject接口。

删除文件

示例代码可以参考删除文件Demo

删除单个文件

可参考如下代码删除了一个Object:

```
public void deleteObject(BosClient client, String bucketName, String objectKey) {

  // 删除Object
  client.deleteObject( < bucketName > , < objectKey > );          //指定要删除的Object所在Bucket名称和该Object名称
}
```

删除多个文件

可参考如下两种方法删除多个Object:

```
// 1、以Json格式的字符串
String jsonObjectKeys = "{\"objects\": ["+"{\"key\": \"token1.h\"},"+"{\"key\": \"token2.h\"}"+"]}";
DeleteMultipleObjectsRequest request = new DeleteMultipleObjectsRequest();
request.setBucketName("yourBucketName");
request.setJsonDeleteObjects(jsonObjectKeys);
client.deleteMultipleObjects(request);
```

```
// 2、用户只需指定指定参数即可
List<String> objectKeys = new ArrayList<String>();
objectKeys.add("object1");
objectKeys.add("object2");
DeleteMultipleObjectsRequest request = new DeleteMultipleObjectsRequest();
request.setBucketName("yourBucketName");
request.setObjectKeys(objectKeys);
DeleteMultipleObjectsResponse response = client.deleteMultipleObjects(request);
```

> 支持一次请求内最多删除1000个Object。 消息体（body）不超过2M。 返回的消息体中只包含删除过程中出错的Object结果；如果所有Object都删除都成功的话，则没有消息体。

对象标签

设置对象标签

目前BOS支持两种方式设置object tagging。第一种是使用Canned方式，在putObjectTagging的时候，通过配置PutObjectTaggingRequest的cannedTag字段设置标签。第二种方式是构造对象标签json。Java sdk已经将两种方式封装。详细信息请参考对象标签。具体demo如下

1、通过Canned方式设置对象标签

```
String objectTagString = "key1=value1&key2=values2";
PutObjectTaggingRequest request = new PutObjectTaggingRequest(bucketName,ObjectName,objectTagString);

client.putObjectTagging(request);
```

2、通过构造结构体设置object tagging

```
List<ObjectTag> objectTags = new ArrayList<>();
Map<String ,String> map = new HashMap<>();
map.put("k1","v1");
map.put("k2","v2");
objectTags.add(ObjectTag.builder().tagInfo(map).build());

this.client.putObjectTagging(bucketName,objectName,objectTags);
```

`ObjectTag`结构体字段如下：

参数	说明
tagInfo	哈希map，标签信息
+key	String，标签key
+value	String，标签value

查看对象标签

如下代码可以查Object的权限：

```
GetObjectTaggingResponse response = this.client.getObjectTagging(bucketName,objectName);
System.out.println(response.toString());
```

删除对象标签

如下代码可以删除Object的权限：

```
DeleteObjectTaggingRequest request = new DeleteObjectTaggingRequest("yourBucketName","objectKey");

this.client.deleteObjectTagging(request);
```

数据处理及使用

生命周期管理

BOS支持用户对Bucket设置生命周期规则，以自动将过期的文件清除，节省存储空间。针对不同前缀的文件，用户可以同时设置多条规则。 详细信息请参考生命周期管理

设置生命周期规则

可以通过如下两种方式设置生命周期规则：

```
// 1、以Json格式的字符串
String jsonBucketLifecylce = "{\"rule\":[{\"id\":\"yourBucketName/*-DeleteObject\""+",\"status\":\"enabled\""+",\"resource\":[\"yourBucketName/*\"]"+",\"condition\":{\"time\":
{\"dateGreaterThan\":\"2018-09-07T00:00:00Z\"}}"+",\"action\":{\"name\":\"DeleteObject\"}}]}";

SetBucketLifecycleRequest request = new SetBucketLifecycleRequest("yourBucketName",jsonBucketLifecylce);
client.setBucketBucketLifecycle(request);
```

```
// 2、用户只需指定指定参数即可
List<Rule> rules = new ArrayList<Rule>();
List<String> resource = new ArrayList<String>();
Action action = new Action();
Condition condition = new Condition();

resource.add("yourBucketName/*");
action.setName("DeleteObject");

Time time = new Time();
time.setDateGreaterThan("2018-09-07T00:00:00Z");
condition.setTime(time);

Rule rule = new Rule();
rule.setId("yourBucketName/*-DeleteObject");
rule.setStatus("enabled");
rule.setResource(resource);
rule.setCondition(condition);
rule.setAction(action);

List<String> resource1 = new ArrayList<String>();
Action action1 = new Action();
Condition condition1 = new Condition();

resource1.add("yourBucketName/*");
action1.setName("Transition");
action1.setStorageClass("STANDARD_IA");

Time time1 = new Time();
time1.setDateGreaterThan("2018-09-08T00:00:00Z");
condition1.setTime(time1);

Rule rule1 = new Rule();
rule1.setId("yourBucketName/*-Transition");
rule1.setStatus("enabled");
rule1.setResource(resource1);
rule1.setCondition(condition1);
rule1.setAction(action1);

rules.add(rule1);
rules.add(rule);

SetBucketLifecycleRequest request = new SetBucketLifecycleRequest();
request.withBucketName("yourBucketName");
request.setRuleList(rules);
client.setBucketBucketLifecycle(request);
```

查看生命周期规则

可通过如下代码查看Bucket内的生命周期规则：

```
GetBucketLifecycleRequest request = new GetBucketLifecycleRequest("yourBucketName");
GetBucketLifecycleResponse response = client.getBucketLifecycle(request);
```

删除生命周期规则

可通过如下代码清空生命周期规则：

```
DeleteBucketLifecycleRequest request = new DeleteBucketLifecycleRequest("yourBucketName");
client.deleteBucketLifecycle(request);
```

跨域资源共享

跨域资源共享（Cross-Origin Resource Sharing），简称CORS，是HTML5提供的标准跨域解决方案，BOS目前已经支持CORS标准来实现跨域访问。关于跨域访问的介绍请参考跨域访问。

设定CORS规则

可以通过如下两种方式设置跨域规则：

```
// 1、以Json格式的字符串
String jsonBucketCors = "{\"corsConfiguration\": [{\"allowedOrigins\":[\"http://*\""+",\"https://*\"]"+",\"allowedMethods\":[\"GET\""+",\"HEAD\""+",\"POST\""+",\"PUT\"]"+",\"allowedHeaders\":
[\"*\"]"+",\"allowedExposeHeaders\":[\"ETag\""+",\"Content-Length\""+",\"x-bce-next-append-offset\""+",\"x-bce-object-type\""+",\"x-bce-request-id\"]"+",\"maxAgeSeconds\":1800}]}";

SetBucketCorsRequest request = new SetBucketCorsRequest();
request.setJsonBucketCors(jsonBucketCors);
request.withBucketName("yourBucketName");
client.setBucketBucketCors(request);
```

```
 // 2、用户只需指定指定参数即可
SetBucketCorsRequest request = new SetBucketCorsRequest();
request.setBucketName("yourBucketName");
List<CorsConfiguration>  corsConfigurations = new ArrayList<CorsConfiguration>();
List<String> allowedOrigins = new ArrayList<String>();
List<AllowedMethods> allowedMethods = new ArrayList<AllowedMethods>();
List<String> allowedHeaders = new ArrayList<String>();
List<String> allowedExposeHeaders = new ArrayList<String>();

CorsConfiguration corsConfiguration = new CorsConfiguration();
CorsConfiguration corsConfiguration1 = new CorsConfiguration();
// AllowedOrigins
allowedOrigins.add("http://www.example.com");
allowedOrigins.add("http://www.example1.com");

allowedMethods.add(AllowedMethods.GET);
allowedMethods.add(AllowedMethods.POST);
allowedMethods.add(AllowedMethods.PUT);
allowedMethods.add(AllowedMethods.DELETE);

// allowedHeaders
allowedHeaders.add("head1");
allowedHeaders.add("head2");

// allowedExposeHeaders
allowedExposeHeaders.add("ETag");
allowedExposeHeaders.add("Content-Length");
allowedExposeHeaders.add("x-bce-next-append-offset");
allowedExposeHeaders.add("x-bce-object-type");

corsConfiguration.setAllowedOrigins(allowedOrigins);
corsConfiguration.setAllowedMethods(allowedMethods);
corsConfiguration.setAllowedHeaders(allowedHeaders);
corsConfiguration.setAllowedExposeHeaders(allowedExposeHeaders);
corsConfiguration.setMaxAgeSeconds(1800);

List<String> allowedOrigins1 = new ArrayList<String>();
List<AllowedMethods> allowedMethods1 = new ArrayList<AllowedMethods>();
List<String> allowedHeaders1 = new ArrayList<String>();
List<String> allowedExposeHeaders1 = new ArrayList<String>();

corsConfigurations.add(corsConfiguration);

// AllowedOrigins
allowedOrigins1.add("allowedOrigin1");
allowedOrigins1.add("allowedOrigin2");

// allowedMethods
allowedMethods1.add(AllowedMethods.GET);
allowedMethods1.add(AllowedMethods.POST);
allowedMethods1.add(AllowedMethods.PUT);
allowedMethods1.add(AllowedMethods.DELETE);

// allowedHeaders
allowedHeaders1.add("head3");
allowedHeaders1.add("head4");

// allowedExposeHeaders
allowedExposeHeaders1.add("ETag");
allowedExposeHeaders1.add("Content-Length");
allowedExposeHeaders1.add("x-bce-next-append-offset");
allowedExposeHeaders1.add("x-bce-object-type");


corsConfiguration1.setAllowedOrigins(allowedOrigins1);
corsConfiguration1.setAllowedMethods(allowedMethods1);
corsConfiguration1.setAllowedHeaders(allowedHeaders1);
corsConfiguration1.setAllowedExposeHeaders(allowedExposeHeaders1);
corsConfiguration1.setMaxAgeSeconds(3600);

corsConfigurations.add(corsConfiguration1);

request.setCorsConfigurationsList(corsConfigurations);
client.setBucketBucketCors(request);
```

注意：

- 通过此接口设置CORS规则之前，Bucket的CORS权限设置为不允许跨域。
- 每个Bucket最多只允许有一个规则文件，因此新上传的规则文件会覆盖原有的。
- CORS规则文件大小限制为20KB，因此请求时大于20KB会返回超出大小错误（400 Bad Request: EntityTooLarge）。

获取CORS规则

可以通过如下方式获取跨域规则：

```
 GetBucketCorsRequest request = new GetBucketCorsRequest();
request.setBucketName("yourBucketName");
GetBucketCorsResponse response = client.getBucketCros(request);
```

删除CORS规则

可以通过如下方式删除跨域规则：

```
DeleteBucketCorsRequest request = new DeleteBucketCorsRequest("yourBucketName");
client.deleteBucketCors(request);
```

设置访问日志

BOS SDK支持将用户访问Bucket时的请求记录记录为日志，会包括请求者、Bucket名称、请求时间和请求操作等。关于Bucket日志的详细功能说明可参见设置访问日志

开启Bucket日志

- 请求参数

名称	描述	类型	是否必须
targetBucket	指定存放访问日志的Bucket	字符串	是
targetPrefix	指定最终被保存的访问日志文件前缀	字符串	否（建议填写，用以区分访问日志）

- 目前提供两种方式供用户使用 1、以Json格式的字符串 2、用户只需指定指定参数即可

可以通过如下两种方式设置日志规则：

```
// 1、以Json格式的字符串
SetBucketLoggingRequest setBucketLoggingRequest = new SetBucketLoggingRequest();
setBucketLoggingRequest.setBucketName("yourBucketName");
String jsonPutBucketLogging = "{\"targetBucket\":\"yourTargetBucketName\""+",\"targetPrefix\":\"mylog1/\"}";
setBucketLoggingRequest.setJsonPutBucketLogging(jsonPutBucketLogging);
client.setBucketLogging(setBucketLoggingRequest);
```

```
// 2、用户只需指定指定参数即可
SetBucketLoggingRequest request = new SetBucketLoggingRequest();
request.setBucketName("yourBucketName");
request.setTargetBucket("yourTargetBucketName");
request.setTargetPrefix("mylog/");
client.setBucketLogging(request);
```

查看Bucket日志设置

可以通过如下方式获取日志规则：

```
GetBucketLoggingRequest request = new GetBucketLoggingRequest();
request.withBucketName("yourBucketName");
GetBucketLoggingResponse response = client.getBucketLogging(request);
```

关闭Bucket日志

可以通过如下方式关闭日志规则：

```
DeleteBucketLoggingRequest request = new DeleteBucketLoggingRequest("yourBucketName");
client.deleteBucketLogging(request);
```

音视频处理

当前支持上传mp4格式的文件，系统自动保存为hls格式，用于m3u8风格播放器读取使用，详细规则及约束见音视频处理接口。

设置音视频处理 BOS音视频处理接口定义如下：

- 格式转换参数

参数名称	缩写	类型	取值	命令描述	是否必选
format	f	string	[hls]	将视频转换转化为指定格式	是
time	t	int	[6,600]	切片时间，默认10s	否

- 持久化存储参数

参数名称	缩写	类型	取值	命令描述	是否必选
object	o	string	-	经过base64编码的object名称，受object长度限制，即不超过1024个字符(此处为转码后的m3u8文件名称，如xxx.m3u8)	是

通过SDK使用如下方式设置：

```
 File file = new File("vedioProcessTest.mp4");
String bucketName = "yourBucketName";
String mp4ObjectKey = "vedioProcessTest.mp4";
String m3u8Path = "video/demo.m3u8";
String encodeToString = Base64.getEncoder().encodeToString(m3u8Path.getBytes(Charset.defaultCharset()));

PutObjectRequest request = new PutObjectRequest(bucketName, mp4ObjectKey, file);
request.setVideoProcess("video/format,f_hls|system/save,o_" + encodeToString);
PutObjectResponse response = client.putObject(request);
```

异常处理

BOS异常提示有如下两种方式：

异常方法	说明
BceClientException	客户端异常
BceServiceException	服务器异常

用户可以使用try获取某个事件所产生的异常，例如：

```
 String objectKey = "testObjectToFile";
bosclient.putObject(bucketName, objectKey, "dataFile");
File file = new File("test");

try {
    bosclient.getObject(this.bucketName, objectKey, file);
} catch (BceServiceException bce){
    System.out.println(bce.getMessage());
} catch ( BceClientException bce){
    System.out.println( bce.getMessage());
} finally {
    file.delete();
}
```

客户端异常

客户端异常表示客户端尝试向BOS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；当上传文件时发生IO异常时，也会抛出ClientException。

服务端异常

当BOS服务端出现异常时，BOS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BOS错误信息格式

sdk日志

SDK日志

BOS 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}/BosUnitTest.%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>
```

常见问题

常见问题

Failed to parse the response result

- 错误原因：该异常表示，java sdk在解析服务端回包解析异常。通常情况下，原因是客户网络层会劫持回包，导致sdk解析不了返回结果
- 解决方法：业务侧可以通过idea调试代码，在返回结果前打断点执行并输出返回结果查看是否有篡改或者劫持。若有，则需要调整网络解析，给bos地址的劫持设置白名单

SignatureDoesNotMatch

- 错误原因：
  - AccessKey ID和AccessKey Secret不一致
  - 版本不匹配。在0.10.227版本之后，SDK会默认自动开启bucket virtual hosting。用户如果报错签名不对，可能是sdk版本过旧导致请求没有正确拼接成四级域名
  - 客户侧网络层修改http请求。客户出网配置可能会将请求header的host修改，导致在服务端获取host验证签名不匹配
- 解决方法：
  - 用户检查自己使用的ak sk
  - 更换sdk到最新版
  - 客户侧检查出网路由，打印请求发出去前的header和body字段。并联系bos值班同学查看后端收到请求的header等字段是否一致

Unable to execute HTTP request

- 错误原因：
  - JVM中存在大量org.apache.http.impl.conn.PoolingHttpClientConnectionManager实例导致连接打满，http请求建立不了连接。客户使用bosclient时每次都new一个新的client会偶发此报错
  - 偶发网络连接抖动或者客户端环境cpu、io打满
- 解决方法：
  - 主动关闭已执行完毕的bosClient或使用单例模式
  - 业务侧进行重试

SocketException

- 错误原因：

- 可能是socket在init阶段就失败了，导致请求没有到达bos

- 解决方法：
  - 检查出现问题时是否出现网络抖动
  - 检查主机的socket连接数是否占满，确认出现问题时连接数是否超过SDK中设置的maxconnection，如果连接数超过maxconnection设置，也会出现socket异常
  - 利用tcpdump或者Wireshark工具进行抓包，问题复现后分析数据包排查

Connection pool shut down

- 错误原因：
  - 调用bosClient.shutdown()接口后，还继续通过bosClient发送请求
- 解决方法：
  - 请检查调用逻辑，确保调用了bosClient.shutdown()接口之后，不再通过bosClient发送请求。需要发请求则重新new一个bosClient

Too Many Requests

- 错误原因：
  - 客户请求并发太高，导致触发了bos服务端的限速，因此部分请求被拒绝执行
- 解决方法：
  - 请检查调用逻辑，并发增高不是异常业务逻辑导致
  - 如果真的需要很大并发，可以联系bos值班同学，看限速还能否上调
  - 业务侧自己降低并发

版本变更记录

- Java SDK开发包[2023-07-18]版本号0.10.272
  - 新增镜像回源处理接口。
- Java SDK开发包[2023-02-10]版本号0.10.244
  - 新增音视频处理接口。
- Java SDK开发包[2022-10-10]版本号0.10.227
  - 新增单链接限速配置，默认使用bucket virtual host域名访问。
- Java SDK开发包[2022-06-10]版本号0.10.212
  - 新增上传进度回调接口；归档取回参数更新。
- Java SDK开发包[2022-04-19]版本号0.10.205
  - 修复三步上传分块最小限制
- Java SDK开发包[2022-04-15]版本号0.10.203
  - 修复STS下设置acl的编码问题
- Java SDK开发包[2022-02-17]版本号0.10.196
  - 调整三步上传分块大小限制
- Java SDK开发包[2021-10-14]版本号0.10.181
  - 修复软链接编码、分块上传默认content-type问题
  - 支持putBucketStorageClass接口
- Java SDK开发包[2021-08-09]版本号0.10.175
  - 修复默认Endpoint，新增object名限制
- Java SDK开发包[2021-06-23]版本号0.10.168
  - 支持sts生成presigned url
- Java SDK开发包[2020-10-20]版本号0.10.135
  - 支持获取Bucket或特定前缀下的容量、object数、文件数
- Java SDK开发包[2020-08-26]版本号0.10.125
  - 支持软链接、多规则replication、取消重定向、归档状态显示等相关接口
- Java SDK开发包[2020-06-11]版本号0.10.111
  - 支持SelectObject接口
- Java SDK开发包[2020-02-14]版本号0.10.94
  - Bucket Acl支持secureTransport
- Java SDK开发包[2019-11-21] 版本号 0.10.82
  - 支持设置、获取、删除bucket服务端加密、静态网站托管、原图访问
  - 支持putSuperObjectFromFile接口封装三步上传大文件
  - 支持更多粗细粒度权限控制
- Java SDK开发包[2018-10-09] 版本号 0.10.36
  新增判断对象是否存在接口
- Java SDK开发包[2018-08-16] 版本号 0.10.34

- 支持设置Object的访问权限。
- 支持删除文件。
- 支持Object分块拷贝。
- 支持设置生命周期管理。
- 支持跨域访问。
- 支持设置访问日志。

- Java SDK开发包[2017-05-25] 版本号 0.10.19
  - 支持冷存储
- Java SDK开发包[2016-12-27] 版本号 0.10.16
  - 支持AppendObject功能。
  - 丰富Object Http header头域。
- Java SDK开发包[2016-08-25] 版本号 0.10.9
  - 支持低频存储功能。
  - 支持cache_control。
- Java SDK开发包[2016-08-02] 版本号 0.10.8
  - 支持同步Copy功能。
  - 内部优化，提升性能以及兼容性。
- Java SDK开发包[2016-05-24] 版本号 0.10.4
  内部优化，提升性能。
- Java SDK开发包[2016-03-03] 版本号 0.9.6
  支持HTTPS。
- Java SDK开发包[2016-01-27] 版本号 0.9.4
  1. “高级权限”中增加“LIST”权限，“LIST”权限包含 “ListObjects” 和 “ListMultipartUploads” API操作；
  2. “高级权限”中缩小“READ”权限，“READ”权限将不支持 “ListObjects” API操作；
  3. SetBucketAcl支持直接传入json格式的acl规则来应用于Bucket。
- Java SDK开发包[2015-10-29] 版本号 0.9.1
  BOS相关功能本次无更新。
- Java SDK开发包[2015-09-06] 版本号 0.9.0
  支持多区域选择。
- Java SDK开发包[2015-07-28] 版本号 0.8.9
  内部优化，提升性能。
- Java SDK开发包[2015-07-21] 版本号 0.8.7
  BOS相关功能本次无更新。
- Java SDK开发包[2015-07-09] 版本号 0.8.6
  BOS相关功能本次无更新。
- Java SDK开发包[2015-06-11] 版本号 0.8.5
  BOS相关功能本次无更新。
- Java SDK开发包[2015-05-28] 版本号 0.8.4
  BOS相关功能本次无更新。
- Java SDK开发包[2015-05-14] 版本号 0.8.3.1
  BOS相关功能本次无更新。
- Java SDK开发包[2015-04-16] 版本号 0.8.2
  更新copyObject中对copy_source的处理方法，当source object为非标准ASCII字符（例如：中文）时使用老版本Java SDK的用户会出现签名无法通过的现象，请及时进行版本更新。
- Java SDK开发包[2015-02-04] 版本号 0.8.1
  首次发布：
  - 支持创建、查看、罗列、删除 Bucket
  - 修改、获取Bucket的访问权限
  - 上传、查看、罗列、删除Object

Python-SDK

概述

本文档主要介绍BOS Python SDK的安装和使用。在使用本文档前，您需要先了解BOS的一些基本知识，并已开通了BOS服务。若您还不了解BOS，可以参考 产品描述和快速入门。

安装SDK工具包

环境准备

1. 运行环境
   Python SDK支持在Python2.7和Python3.x 的环境下运行。

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`

3. 查看当前使用的SDK版本

   ```
   import baidubce
   print(baidubce.SDK_VERSION)
   ```

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。

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
```

来完成SDK的安装。

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
    │      └─── bos                 //BOS目录
    │           ├──bos_client.py      //BOS客户端入口类
    │           ├──bos_handle.py      //HTTP reponse处理函数
    │           ├──canned_acl.py      //权限控制需要的常量
    │           └──storage_class.py   //存储类型定义模块
    ├─── bce_base_client.py          //BCE客户端入口类的基类
    ├─── bce_client_configuration.py    //针对BOS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
```

卸载SDK

预期卸载 SDK 时，使用pip卸载“bce-python-sdk”即可。

初始化

快速入门

1. 初始化一个BosClient。

   BosClient是与BOS服务交互的客户端，BOS Python SDK的BOS操作都是通过BosClient完成的。用户可以参考BosClient。完成初始化客户端的操作。

2. 新建一个Bucket。

   Bucket是BOS上的命名空间，相当于数据的容器，可以存储若干数据实体（Object）。用户可以参考新建Bucket来完成新建一个Bucket的操作。

3. 上传Object。

   Object是BOS中最基本的数据单元，用户可以把Object简单的理解为文件。用户可以参考上传Object完成对Object的上传。

4. 列出指定Bucket中的全部Object。

   当用户完成一系列上传后，可以参考查看Bucket中Object列表来查看指定Bucket下的全部Object。

5. 获取指定Object

   用户可以参考获取Object来实现对一个或者多个Object的获取。

确认Endpoint

在确认您使用SDK时配置的Endpoint时，可先阅读开发人员指南中关于BOS访问域名的部分，理解Endpoint相关的概念。 百度智能云目前开放了多区域支持，请参考区域选择说明。

目前支持多个区域，对应信息为：

区域	访问EndPoint	支持协议
北京	bj.bcebos.com	HTTP,HTTPS
保定	bd.bcebos.com	HTTP,HTTPS
苏州	su.bcebos.com	HTTP,HTTPS
广州	gz.bcebos.com	HTTP,HTTPS
成都	cd.bcebos.com	HTTP,HTTPS
香港	hkg.bcebos.com	HTTP,HTTPS
武汉	fwh.bcebos.com	HTTP,HTTPS
金融云上海专区	fsh.bcebos.com	HTTP,HTTPS

获取密钥

要使用百度智能云BOS，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BOS做签名验证。 可以通过如下步骤获得并了解您的AK/SK信息：

注册百度智能云账号

创建AK/SK

新建BosClient

BosClient是BOS服务的客户端，为开发者与BOS服务进行交互提供了一系列的方法。

通过AK/SK方式访问BOS

BosClient是BOS服务的Python客户端，为调用者与BOS服务进行交互提供了一系列的方法。

1.在新建BosClient之前，需要先创建配置文件对BosClient进行配置，以下将此配置文件命名为`bos_sample_conf.py`，具体配置信息如下所示：

```
 #!/usr/bin/env python
#coding=utf-8

#导入Python标准日志模块
import logging

#从Python SDK导入BOS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BosClient的Host，Access Key ID和Secret Access Key
bos_host = "bj.bcebos.com"
access_key_id = "AK"
secret_access_key = "SK"

#设置日志文件的句柄和日志级别
logger = logging.getLogger('baidubce.http.bce_http_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(access_key_id, secret_access_key), endpoint = bos_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，`access_key_id`对应控制台中的“Access Key ID”，`secret_access_key`对应控制台中的“Access Key Secret”，获取方式请参考《操作指南 管理ACCESSKEY》。

上面的方式用户需要自己指定BOS的服务的域名，可以通过赋值给`bos_host`变量来指定。不指定时无需传入`endpoint`参数，默认为北京区域`http://bos.bj.bcebos.com`。

2.在完成上述配置之后，参考如下代码新建一个BosClient。建议新建一个py文件使用client调用bos功能，和配置文件解耦

```
 #导入BosClient配置文件
import bos_sample_conf

#导入BOS相关模块
from baidubce import exception
from baidubce.services import bos
from baidubce.services.bos import canned_acl
from baidubce.services.bos.bos_client import BosClient

#新建BosClient
bos_client = BosClient(bos_sample_conf.config)
```

通过sts方式访问BOS

BOS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BOS，用户先通过sts-client申请一套AK、SK和token，然后将该套参数配置到BosClient中，用户可以参考如下代码新建一个BosClient：

1. 获取临时token，创建配置文件"sts_sample_conf.py"对BosClient进行配置。

```
import logging
from baidubce.bce_client_configuration import BceClientConfiguration
from baidubce.auth.bce_credentials import BceCredentials

sts_host = "STS_HOST"
access_key_id = "AK"
secret_access_key = "SK"

logger = logging.getLogger('baidubce.services.sts.stsclient')
fh = logging.FileHandler("sample.log")
fh.setLevel(logging.DEBUG)

formatter = logging.Formatter('%(asctime)s - %(name)s - %(levelname)s - %(message)s')
fh.setFormatter(formatter)
logger.setLevel(logging.DEBUG)
logger.addHandler(fh)

config = BceClientConfiguration(credentials=BceCredentials(access_key_id, secret_access_key), endpoint=sts_host)
```

> 注意： 目前使用STS配置client时，无论对应BOS服务的endpoint在哪里，endpoint都需配置为http://sts.bj.baidubce.com。

在上面代码中，access_key_id对应控制台中的“Access Key ID”，secret_access_key对应控制台中的“Access Key Secret”。

2. 新建StsClient，并使用获得的临时token，配置BceClientConfiguration。创建文件"sts_sample.py"通过StsClient申请一套AK、SK和token。

```
import sts_sample_conf
from baidubce.services.sts.sts_client import StsClient
from baidubce.bce_client_configuration import BceClientConfiguration
from baidubce.auth.bce_credentials import BceCredentials

sts_client = StsClient(sts_sample_conf.config)

duration_seconds = 3600
# you can specify limited permissions with ACL
access_dict = {}
access_dict["service"] = "bce:bos"
access_dict["region"] = "bj"
access_dict["effect"] = "Allow"
resource = ["*"]
access_dict["resource"] = resource
permission = ["READ"]
access_dict["permission"] = permission

access_control_list = {"accessControlList": [access_dict]}
# 新建StsClient
response = sts_client.get_session_token(acl=access_control_list, duration_seconds=duration_seconds)

sts_ak = str(response.access_key_id)
sts_sk = str(response.secret_access_key)
token = response.session_token
bos_host = "BOS_HOST"
#配置BceClientConfiguration
config = BceClientConfiguration(credentials=BceCredentials(sts_ak, sts_sk), endpoint = bos_host, security_token=token)
```

3. 在完成上述配置之后，参考如下代码新建一个BosClient。

```
import sts_sample

#导入BOS相关模块
from baidubce import exception
from baidubce.services import bos
from baidubce.services.bos import canned_acl
from baidubce.services.bos.bos_client import BosClient

#新建BosClient
bos_client = BosClient(sts_sample.config)
```

配置HTTPS协议访问BOS

您可以通过如下两种方式在BOS Python SDK中使用HTTPS访问BOS服务：

- 在endpoint中指定HTTPS:

```
from baidubce.bce_client_configuration import BceClientConfiguration
from baidubce.auth.bce_credentials import BceCredentials
config = BceClientConfiguration(
    credentials = BceCredentials('ak','sk'),
    endpoint = 'https://bj.bcebos.com'
)
client = bos_client.BosClient(config)
```

- 通过在protocol中指定https来设置HTTPS协议:

```
from baidubce.bce_client_configuration import BceClientConfiguration
from baidubce.auth.bce_credentials import BceCredentials
config = BceClientConfiguration(
    credentials = bce_credentials.BceCredentials('ak','sk'),
    endpoint = 'bj.bcebos.com',
    protocol = baidubce.protocol.HTTPS
)
client = bos_client.BosClient(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。

```
config = bce_client_configuration.BceClientConfiguration(
  credentials = bce_credentials.BceCredentials(
      access_key_id = 'your-ak',
   secret_access_key = 'your-sk'
  ),
  endpoint = 'http://bj.bcebos.com',
  protocol = baidubce.protocol.HTTPS
)
client = bos_client.BosClient(config)
```

配置BosClient

设置网络参数

用户可以设置一些网络参数：

```
  #设置请求超时时间
  bos_sample_conf.config.connection_timeout_in_mills = TIMEOUT

  #设置接收缓冲区大小
  bos_sample_conf.config.recv_buf_size(BUF_SIZE)

  #设置发送缓冲区大小
  bos_sample_conf.config.send_buf_size(BUF_SIZE)

  #设置连接重试策略
  #三次指数退避重试
  bos_sample_conf.config.retry_policy = BackOffRetryPolicy()
  #不重试
  bos_sample_conf.config.retry_policy = NoRetryPolicy()
```

参数说明

通过bos_client_configuration能指定的所有参数如下表所示：

参数	说明
port	BOS端口号
send_buf_size	发送缓冲区大小
recv_buf_size	接收缓冲区大小
connection_timeout_in_mills	请求超时时间（单位：毫秒）
retry_policy	连接重试策略，初始化Client时默认为三次指数退避

设置代理 下面一段代码可以让客户端使用代理访问BOS服务：

```
config = bce_client_configuration.BceClientConfiguration(
    credentials = bce_credentials.BceCredentials(
        access_key_id = 'your-ak',
       secret_access_key = 'your-sk'
    ),
    endpoint = 'https://bj.bcebos.com',
    proxy_host = 'your-proxy-host',
    proxy_port = your-proxy-port
  )
client = bos_client.BosClient(config)
```

使用自定义域名 政府监管原因，公网上的违法资源的URL，如果是BOS域名，给BOS带来监管风险，鼓励外网用户使用自定义域名 如果希望使用自定义域名作为访问BOS的endpoint，在控制台将自定义域名和BOS某个bucket绑定之后，配置endpoint为自定义域名并打开cname_enabled开关，例如cdn-test.cdn.bcebos.com，配置代码如下：

```
config = bce_client_configuration.BceClientConfiguration(
  credentials = bce_credentials.BceCredentials(
      access_key_id = 'your-ak',
   secret_access_key = 'your-sk'
  ),
  endpoint = 'http://cdn-test.cdn.bcebos.com',
  cname_enabled = True
)
client = bos_client.BosClient(config)
```

使用备用域名 在使用官方域名和自定义域名作为endpoint访问BOS的基础上，SDK进一步支持了使用endpoint主域名访问BOS失败时重试备用域名机制，使用方式如下：

```
config = bce_client_configuration.BceClientConfiguration(
  credentials = bce_credentials.BceCredentials(
      access_key_id = 'your-ak',
   secret_access_key = 'your-sk'
  ),
  endpoint = 'http://cdn-test.cdn.bcebos.com',
  cname_enabled = True,
  # 如果使用endpoint访问BOS失败，SDK会自动重试backup_endpoint域名
  backup_endpoint = 'cdn-test.bj.bcebos.com'
)
client = bos_client.BosClient(config)
```

设置访问官方域名风格 随着BOS的用户规模和吞吐量不断增加， path-style的连接方式逐渐暴露出弊端。所有使用path-style域名的用户，都在使用同一个域名， 使用同样域名解析规则，当DNS解析出现故障，或者遭受攻击导致可用性下降时，影响范围将会很大，而virtual-hosted的域名，可以分别定义域名解析规则，将故障域切分到足够小。

virtual-host风格域名访问url示例:{bucket}.${region}.bcebos.com/${object}

path-style风格域名访问url示例:${region}.bcebos.com/${bucket}/${object}

自v0.9.05版本以后，用户配置endpoint为BOS官方域名时，将默认修改为virtual-host风格域名访问BOS服务。 可以通过以下配置开启使用PathStyle风格的endpoint

```
from baidubce.bce_client_configuration import BceClientConfiguration
from baidubce.auth.bce_credentials import BceCredentials
config = BceClientConfiguration(
    credentials = bce_credentials.BceCredentials('ak','sk'),
    endpoint = 'bj.bcebos.com',
    path_style_enable = True
   )
client = bos_client.BosClient(config)
```

Bucket管理

概述

Bucket既是BOS上的命名空间，也是计费、权限控制、日志记录等高级功能的管理实体。

- Bucket名称在所有区域中具有全局唯一性，且不能修改。

> **说明：**
>
> 百度智能云目前开放了多区域支持，请参考区域选择说明。

- 存储在BOS上的每个Object都必须包含在一个Bucket中。
- 一个用户最多可创建100个Bucket，但每个Bucket中存放的Object的数量和大小总和没有限制，用户不需要考虑数据的可扩展性。

Bucket权限管理

设置Bucket的访问权限

如下代码将Bucket的权限设置为了private：

```
bos_client.set_bucket_canned_acl(bucket_name, canned_acl.PRIVATE)
```

其中canned_acl中包含三个参数：`PRIVATE`、`PUBLIC_READ`、`PUBLIC_READ_WRITE`，它们分别对应的相关权限为：`private`、`public-read`、`public-read-write`。关于权限的具体内容可以参考《BOS API文档 使用CannedAcl方式的权限控制》。

设置指定用户对Bucket的访问权限

BOS提供set_bucket_acl方法来实现指定用户对Bucket的访问权限设置，可以参考如下代码实现：

```
bos_client.set_bucket_acl(
  bucket_name,
  [{'grantee': [{'id': 'b124deeaf6f641c9ac27700b41a350a8'},
                {'id': 'aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa'}],
    'permission': ['FULL_CONTROL'],
    'resource':['your_bucket_name/prefix/*']
   }])
```

> **注意：**
>
> 1. id为用户ID，您可在用户信息中查看。
> 2. permission中的权限设置包含三个值：READ、WRITE、FULL_CONTROL，它们分别对应相关权限。具体内容可以参考《BOS API文档 上传ACL文件方式的权限控制》。
> 3. set_bucket_acl()的第二个参数acl不需要包含"accessControlList"字段，已经封装到接口中。

设置更多Bucket访问权限

1. 通过设置referer白名单方式设置防盗链

```
myAcl ={"grantee":[{"id": "*"}],
 "permission":["FULL_CONTROL"],
 "condition":{
  "referer":{"stringEquals":["http://test/index"]}
  }
 }]
bos_client.set_bucket_acl(bucket_name, myAcl)
```

2. 限制客户端IP访问，只允许部分客户端IP访问

```
myAcl = [{"grantee":[{"id":"*"}],
 "permission":["FULL_CONTROL"],
 "condition":{"ipAddress":["192.170.0.6"]}
 }]
bos_client.set_bucket_acl(bucket_name, myAcl)
```

设置STS临时token权限

对于通过STS方式创建的临时访问身份，管理员也可进行专门的权限设定。 STS的简介及设置临时权限的方式可参见临时授权访问。

使用BOS Python SDK设置STS临时token权限可参考通过sts方式访问BOS

查看Bucket的权限

如下代码可以查看Bucket的权限：

```
response = bos_client.get_bucket_acl(bucket_name)

bos_client.set_bucket_acl(bucket_name, response.access_control_list)
```

`get_bucket_acl`方法返回的解析类中可供调用的参数有：

参数	说明
owner	Bucket owner信息
+id	Bucket owner的用户ID
access_control_list	标识Bucket的权限列表
+grantee	标识被授权人
++id	被授权人ID
+permission	标识被授权人的权限
+resource	ACL配置项所影响的资源

查看Bucket所属的区域

Bucket Location即Bucket Region，百度智能云支持的各region详细信息可参见区域选择说明。

如下代码可以获取该Bucket的Location信息：

```
region = bos_client.get_bucket_location(bucket_name)
print(region)
```

新建Bucket

如下代码可以新建一个Bucket：

```
if not bos_client.does_bucket_exist(bucket_name):
    bos_client.create_bucket(bucket_name)
```

> **注意：** 由于Bucket的名称在所有区域中是唯一的，所以需要保证bucketName不与其他所有区域上的Bucket名称相同。 Bucket的命名有以下规范：
>
> - 只能包括小写字母，数字，短横线（-）。
> - 必须以小写字母或者数字开头。
> - 长度必须在4-63字节之间。

列举Bucket

用如下方式可以列出用户所有的Bucket：

```
response = bos_client.list_buckets()
for bucket in response.buckets:
    print(bucket.name)
```

list_buckets方法返回的解析类中可供调用的参数如下：

参数	说明
owner	Bucket Owner信息
+id	Bucket Owner的用户ID
+display_name	Bucket Owner的名称
buckets	存放多个Bucket信息的容器
bucket	存放一个Bucket信息的容器
+name	Bucket名称
+creation_date	Bucket创建时间
+location	Bucket所属的区域

删除Bucket

如下代码可以删除一个Bucket：

```
bos_client.delete_bucket(bucket_name)
```

> **注意：**
>
> - 如果Bucket不为空（即Bucket中有Object和未完成的三步上传Part存在），则Bucket无法被删除，必须清空Bucket后才能成功删除。
> - 在删除前确认该Bucket没有开通跨区域复制，不是跨区域复制规则中的源Bucket或目标Bucket，否则不能删除。

判断Bucket是否存在

若用户需要判断某个Bucket是否存在，则如下代码可以做到：

```
    # 获取Bucket的存在信息,需要传入bucket名称，返回值为布尔型
    exists = bos_client.does_bucket_exist(bucket_name)
    # 输出结果
    if exists:
        print("Bucket exists")
    else:
        print("Bucket not exists")
```

设置Bucket存储类型

如果用户使用API、CLI或者SDK上传的Object未指定存储类型，则继承Bucket的默认存储类型。如果上传Object指定的存储类型和Bucket默认存储类型不一致时，以Object的存储类型为准。存储类型包含标准存储、低频存储、冷存储和归档存储四种，具体使用场景和性能请参见分级存储。 **put_bucket_storage_class** 本接口用来设置Bucket的默认存储类型。

```
bos_client.set_bucket_storage_class(bucket_name, storage_class=storage_class.COLD)
```

get_bucket_storage_class 本接口用来获取Bucket的默认存储类型。

```
response = bos_client.get_bucket_storage_class(bucket_name)
```

Bucket生命周期管理

应用场景

一个数据是有其生命周期的，从创建到归档到删除可以认为是一个完整的循环。创建之初的数据往往需要频繁访问读取，之后迅速冷却归档，最终被删除。生命周期管理就是对象存储服务帮助用户自动化管理数据的生命周期。 通常可以服务于以下场景：

1. 数据达到一定寿命后自动归档或删除。

2. 指定时间执行操作。

新建Lifecycle配置

如下代码新建了一个Lifecycle配置

```
rule = {}
rule['id'] = 'rule1'
rule['status'] = 'enabled'
rule['action'] = {}
rule['action']['name'] = 'Transition'
rule['action']['storageClass'] = 'STANDARD_IA'
#对baidubosmty2里所有Object生效
rule['resource'] = ['baidubosmty2/*']
rule['condition'] = {}
rule['condition']['time'] = {'dateGreaterThan': 'XXXX-XX-XXTXX:XX:XXZ'}
rules=[]
rules.append(rule)
bos_client.put_bucket_lifecycle(bucket_name, rules)
```

> **注意：**
>
> 1. 只有Bucket的Owner拥有full control才能够进行此操作。
>
> 2. "resource"指明规则对哪些资源生效。例如，对samplebucket里以`prefix/`为前缀的Object生效：`samplebucket/prefix/*`；对samplebucket里所有Object生效：`samplebucket/*`。

生命周期管理功能相关参数的详细解释及配置注意事项请参见PutBucketLifecycle接口。

读取Bucket的Lifecycle配置

如下代码可读取Bucket的Lifecycle配置

```
response = bos_client.get_bucket_lifecycle(bucket_name)
```

删除Bucket Lifecycle

如下代码可以删除Bucket的lifecycle:

```
bos_client.delete_bucket_lifecycle(bucket_name)
```

日志管理

put_bucket_logging 本接口用来开启Bucket的访问日志并指定存放日志的Bucket和访问日志的文件前缀。访问日志的规则请参见日志命名规则和日志格式。

```
bos_client.put_bucket_logging(source_bucket, target_bucket, prefix)
```

get_bucket_logging 本接口用来获取某个Bucket的访问日志配置情况。

```
response = bos_client.get_bucket_logging(bucket_name)
```

delete_bucket_logging

```
bos_client.delete_bucket_logging(bucket_name)
```

Bucket跨域资源访问

应用场景

跨域资源共享(CORS)允许WEB端的应用程序访问不属于本域的资源。BOS提供接口方便开发者控制跨域访问的各种权限。

设定CORS规则

如下代码设定了一个CORS规则：

```
conf = {}
conf['allowedOrigins'] = ['http://www.boluor.com']
conf['allowedMethods'] = ['GET', 'HEAD', 'DELETE']
conf['allowedHeaders'] = ['Authorization', 'x-bce-test', 'x-bce-test2']
conf['allowedExposeHeaders'] = ['user-custom-expose-header']
conf['maxAgeSeconds'] = 3600
confs = []
#每个Bucket最多允许有100条规则。
confs.append(conf)
bos_client.put_bucket_cors(bucket_name, confs)
```

> **注意:**
>
> 1. 如果原规则存在则覆盖原规则。
> 2. 只有Bucket的所有者和被授予FULL_CONTROL权限的用户才能设置Bucket的CORS。没有权限时，返回403 Forbidden错误，错误码：AccessDenied。

CORS规则相关参数的详细解释请参见PutBucketCors接口。

获取Bucket的CORS规则

如下代码可获取Bucket的CORS配置:

```
response = bos_client.get_bucket_cors(bucket_name)
```

关闭Bucket的CORS功能并清空所有规则

如下代码可以关闭Bucket的CORS功能并清空所有规则:

```
bos_client.delete_bucket_cors(bucket_name)
```

数据同步

设置数据同步规则

参数	是否必须	描述
id	是	replication规则名，id 由数字字母 - _ 组成，不得超过20个字符
status	是	是否生效
resource	是	replication生效前缀，resource的配置形式为｛$bucket_name/<生效的对象前缀>｝，必须要以$bucket_name+/开头
destination	是	复制的目的端配置
+bucket	是	目的Bucket name
+storageClass	否	目的Object的存储类型。如果保持和源Bucket的存储类型一致，则该参数不需要配置；如果需要单独指定存储类型可以为STANDARD，STANDARD_IA，COLD；如果是多AZ类型bucket，可取值为MAZ_STANDARD_IA和MAZ_STANDARD。
replicateHistory	否	历史文件复制，有该项则认为是开启。开启历史文件复制后，存量的全部Object都同步复制到目的Bucket，复制范围共用resource。
+storageClass	否	目的Object的存储类型。如果保持和源Bucket的存储类型一致，则该参数不需要配置；如果需要单独指定存储类型可以为STANDARD，STANDARD_IA，COLD；如果是多AZ类型bucket，可取值为MAZ_STANDARD_IA和MAZ_STANDARD。
replicateDeletes	是	是否开启删除同步,可以为enabled,disabled 。

若用户需要开启Bucket之间的数据同步，则如下代码可以做到：

```
 bucket_name = "<YOUR_BUCKET_NAME>"
rule_id = "sample-rep-config"
dst_bucket_name = "<DESTINATION_BUCKET_NAME>"
replication = {
    "status":"enabled",
    "resource":[
        bucket_name + "/*"
        ],
    "destination":
        {
            "bucket": dst_bucket_name,
            "storageClass":"COLD"
        },
    "replicateHistory":
        {
            "bucket": dst_bucket_name,
            "storageClass":"COLD"
        },
    "replicateDeletes":"enabled",
    "id": rule_id
    }
# test put bucket replication
bos_client.put_bucket_replication(bucket_name, replication)
```

获取指定的数据同步规则 若用户想获取bucket指定id的数据同步信息，包括源Bucket名称、目的Bucket名称、存储类型、是否进行历史复制、数据同步策略、目的region等。则如下代码所示：

```
 response = bos_client.get_bucket_replication(bucket_name, id=rule_id)
print("id:{}, status:{}, resource:{}, destination:{}, replicateDeletes:{}, replicateHistory:{}".format(response.id, response.status, response.resource, response.destination, response.replicate_deletes,
response.replicate_history))
```

删除指定的数据同步规则 若用户想删除bucket指定id的数据同步，则如下代码所示：

```
bos_client.delete_bucket_replication(bucket_name ,id=rule_id)
```

列举数据同步规则 若用户想获取bucket所有的replication同步规则，则如下代码所示：

```
 response = bos_client.list_bucket_replication(bucket_name)
print(response.rules)
```

获取指定的数据同步规则的进程状态 若用户想获取指定id的数据同步复制的进程状态，则如下代码所示：

```
response = bos_client.get_bucket_replication_progress(bucket_name, id=rule_id)
print("status:{}, historyReplicationPercent:{}, latestReplicationTime:{}".format(response.status, response.history_replication_percent, response.latest_replication_time))
```

Bucket回收站

应用场景 为了提高 BOS 数据的可靠性，用户可以通过配置回收站(Bucket Trash)的功能，保证删除后的数据在回收站内继续保留，用于后续对删除数据的找回。用户必须是源Bucket的owner且拥有FULL_CONTROL权限，且是目标Bucket的owner。

> **注意：**
> 1.进入回收站的文件仍然会继续计费，直到用户彻底删除。对于回收站中的文件，建议用户配置生命周期来删除过期的文件。
> 2.用户进行覆盖操作时，旧的数据不会进入回收站。

开通回收站 |参数 | 是否必须 | 描述 | | -------- | ------ | ----- | | trash_dir | 否 | 指定回收站目录名，可选。回收站默认目录为".trash"。 |

若用户想开通Bucket回收站并指定回收站目录，则如下代码所示：

```
bos_client.put_bucket_trash(bucket_name, trash_dir=".trash")
```

> **注意：**
> 1.若之前已开通过回收站功能，再次开通会覆盖原回收站目录名。
> 2.回收站目录名词只能包含字母、数字、中文、下划线（_）和短横线（-）、小数点（.）。包含'/'会报错。
> 3.开通了回收站功能的bucket， 删除object会回到回收站中(回收站中的object的全称是：trashDir + '/' + old object name)。未开通此功能的bucket，或者开通此功能的bucket且删除回收站内的object时，会彻底删除。
> 4.归档类型文件不支持Bucket Trash功能，删除归档类型文件时会直接删除。

获取回收站开通信息

若用户想获取Bucket回收站开通信息，获取当前回收站目录名,则如下代码所示：

```
response = bos_client.get_bucket_trash(bucket_name)
print("trash_dir:", response.trash_dir)
```

关闭回收站

若用户想关闭Bucket回收站，则如下代码所示：

```
response = bos_client.delete_bucket_trash(bucket_name)
```

用户额度管理

put_user_quota 此接口用来设置用户的额度，额度包括总的bucket数及各个region总的存储容量，当前总的Bucket数最大值不能超过100，注意该接口只能被主用户调用，

参数	是否必须	描述
max_bucket_count	是	最大可以创建的bucket数，若为-1，则表示不设置
max_capacity_mega_bytes	是	单位为MB，表示最大的存储容量，若为-1或者0，表示不设置存储容量额度限制，即无上限

```
response = bos_client.put_user_quota(max_bucket_count, max_capacity_mega_bytes)
```

get_user_quota 此接口用来查看用户的额度设置，注意该接口只能被主用户调用。

```
response = bos_client.get_user_quota()
print("[Sample] get user quota maxBucketCount:", response.max_bucket_count)
print("[Sample] get user quota maxCapacityMegaBytes:", response.max_capacity_mega_bytes)
```

delete_user_quota 此接口用来删除额度设置，注意该接口只能被主用户调用。

```
response = bos_client.delete_user_quota()
```

事件通知

put_notification 本接口用于指定bucket上增加通知规则。

> **注意：**
> - 只有bucket owner或者full control权限才能获取这个bucket的配置。
> - 如果不是bucket owner则返回403，如果对应的文件不存在则返回404。

相关参数的详细解释请参见PutNotification接口。

```
notifications = list()
notifications.append(
    {
        "resources": [
            "/"
        ],
        "encryption": {
            "key": "06a62b70f47dc4a0a7da349609f1a1ac"
        },
        "status": "enabled",
        "name": "name3",
        "id": "r3",
        "appId": "p3",
        "events": [
            "AppendObject",
            "CompleteMultipartUpload",
            "CopyObject",
            "PutObject",
            "PostObject",
            "FetchObject",
            "DeleteObject"
        ],
        "apps": [
            {
                "eventUrl": "http://www.liujiang.com",
                "id": "ImageCensor",
                "xVars": "{\"saveUrl\": \"http://xxx.com/ocr\"}"
            }
        ]
    })
response = bos_client.put_notification(bucket_name, notifications)
```

get_notification 本接口用于获取指定bucket上的通知规则。

```
response = bos_client.get_notification(bucket_name)
```

delete_notification 本接口用于删除指定bucket上的通知规则。

> **注意：**
> 该接口为一次性全部删除当前bucket下全部规则

```
response = bos_client.delete_notification(bucket_name)
```

原图保护

put_bucket_copyright_protection 本接口用来开启Bucket的原图保护功能，并指定resource字段，表示生效的资源范围。 对于开通原图保护的文件，不允许匿名下载访问该文件，或带自定义图片处理参数访问，只允许使用style样式访问或携带合法签名访问。

```
resource = [bucket_name + "/test/*"]
bos_client.put_bucket_copyright_protection(bucket_name, resource)
```

get_bucket_copyright_protection 此命令用来获取某个Bucket的原图保护配置情况。

```
response = bos_client.get_bucket_copyright_protection(bucket_name)
```

delete_bucket_copyright_protection 本接口用来关闭Bucket原图保护功能。

```
bos_client.delete_bucket_copyright_protection(bucket_name)
```

bucket清单管理

应用场景

存储空间清单（inventory）是一种帮助用户管理存储桶中对象的功能。BOS 可根据用户的清单任务配置，定期每天或者每周扫描用户 Bucket 中指定对象并输出一份清单报告，以 CSV 格式文件的形式存储到用户指定的 Bucket 中。CSV 文件中会列出对象及其对应的元数据，并根据用户的配置信息，记录用户所需的对象属性信息。

说明：

- 当前该功能在华北-北京、华东-苏州和华南-广州区域全量开放。

设定清单规则

如下代码设定了一个清单规则：

```
 # 设置清单规则名称
inventory_conf_id = "test-inventory-rule-01"
# 设置源bucket名称
source_bucket_name = "<SOURCE_BUCKET_NAME>"
# 设置清单输出的目的bucket
target_bucket_name = "<TARGET_BUCKET_NAME>"
# 设置清单输出的目录前缀
target_prefix = "inventory-result/"
# 如需要关闭清单，将"status"设置为"disabled"即可。
my_inventory = {
    "id": inventory_conf_id,
    "status": "enabled",
    "resource": [source_bucket_name + "/*"],
    "schedule": "Weekly",
    "destination":{
        "targetBucket": target_bucket_name,
        "targetPrefix": target_prefix,
        "format": "CSV"
        }
}
# put bucket inventory
bos_client.put_bucket_inventory(bucket_name, my_inventory)
```

> **注意:**
>
> 1.用户必须是源Bucket的owner且拥有FULL_CONTROL权限，且是目标Bucket的owner。
>
> 2.目的Bucket和源Bucket必须存在，目的Bucket和源Bucket必须是同账号下的Bucket，且是同region下的Bucket，不支持跨账号输出。
>
> 3.BOS支持在每个存储桶中创建最多20条清单任务。
>
> 4.清单列表提供了新对象和覆盖的 PUT 的最终一致性，并提供了 DELETE 的最终一致性。清单列表是Bucket的滚动快照，内容最终是一致的 ，清单可能不包含最近添加或删除的对象。
>
> 5.如果原规则存在则覆盖原规则。

获取Bucket的清单规则

如下代码可获取Bucket清单功能指定规则的配置:

```
 # 指定清单规则id
inventory_conf_id = "test-inventory-rule-01"
response = bos_client.get_bucket_inventory(bucket_name, inventory_conf_id)
print(response)
```

列举Bucket的清单规则 如下代码可列举Bucket清单功能的规则:

```
 response = bos_client.list_bucket_inventory(bucket_name)
print(response.inventory_rule_list)
```

删除清单规则

如下代码可删除Bucket清单功能指定规则的配置:

```
 # 指定清单规则id
inventory_conf_id = "test-inventory-rule-01"
response = bos_client.delete_bucket_inventory(bucket_name, inventory_conf_id)
```

镜像回源

若您为 Bucket 配置了镜像回源，当用户访问该 Bucket 中某一 Object 时，发现该 Object 在 BOS 中并不存在，则 BOS 会立刻向预先配置好的回源地址请求该 Object，保证访问正常进行。在请求过程中，源站返回的数据会自动给到请求用户而用户对该过程无感。同时，该数据也会自动存入 BOS，提升日后该类请求的流畅性。参考镜像回源

代码示例

```
    mirror_args = list()
    mirror_args.append({
    "mode": "fetch",
    "sourceUrl": "<url>",
    "backSourceUrl": "<url>",
                "resource": "*.jpeg",
    "version": "v2",

    "passQueryString": False,
    "storageClass": "STANDARD",
    })

    response = bos_client.put_bucket_mirroring(bucket_name, mirror_args=mirror_args)
    __logger.debug("[Sample] put bucket mirroing response :%s", response)

    response = bos_client.get_bucket_mirroring(bucket_name)
    __logger.debug("[Sample] get bucket mirroing response :%s", response)

    response = bos_client.delete_bucket_mirroring(bucket_name)
    __logger.debug("[Sample] delete bucket mirroing response :%s", response)
```

文件管理

上传文件

在BOS中，用户操作的基本数据单元是Object。Bucket中的Object数量不限，但单个Object最大允许存储5TB的数据。Object包含Key、Meta和Data。其中，Key是Object的名字；Meta是用户对该Object的描述，由一系列Name-Value对组成；Data是Object的数据。

BOS Python SDK提供了丰富的文件上传接口，可以通过以下方式上传文件：

- 简单上传

- 追加上传
- 分片上传
- 断点续传上传
- 获取上传进度

Object的命名规范如下：

- 使用UTF-8编码。
- 长度必须在1-1023字节之间。
- 首字母不能为'/'，不能包含'@'字符，'@'用于图片处理接口。

简单上传

BOS在简单上传的场景中，支持以指定文件形式、以数据流方式、以字符串方式执行Object上传，请参考如下代码：

1. 如下代码可以进行Object上传：

```
 data = open(file_name, 'rb')
#以数据流形式上传Object，用户需要自行计算数据长度content_length
#用户需自行计算content_md5.计算方法是对数据执行md5算法获取128位二进制数据，再进行base64编码
bos_client.put_object(bucket_name, object_key, data, content_length,content_md5)

#从字符串中上传的Object
bos_client.put_object_from_string(bucket_name, object_key, string)

#从文件中上传的Object
bos_client.put_object_from_file(bucket_name, object_key, file_name)
```

其中，data为流对象，不同类型的Object采用不同的处理方法，从字符串中的上传使用StringIO的返回，从文件中的上传使用open()的返回，因此BOS提供了封装好的接口方便用户进行快速上传。

Object以文件的形式上传到BOS中，put_object相关接口均支持不超过5GB的Object上传。在put_object、put_object_from_string或者put_object_from_file请求处理成功后，BOS会在Header中返回Object的ETag作为文件标识。

这些接口均有可选参数:

参数	说明
content_type	上传文件或字符串的类型
content_md5	文件数据校验，设置后BOS会启用文件内容MD5校验，把您提供的MD5与文件的MD5比较，不一致会抛出错误
content_length	定义文件长度，put_object_from_string()不包含该参数
content_sha256	用于进行文件校验
user_metadata	用户自定义元数据
storage_class	设置文件存储类型
user_headers	用户定义header

content_md5计算方法是对数据执行md5算法获取128位二进制数据，再进行base64编码。示例如下：

```
 import io
import hashlib
import base64

file_name = "your_file"
buf_size = 8192
with open(file_name, 'rb') as fp:
    md5 = hashlib.md5()
    while True:
        bytes_to_read = buf_size
        buf = fp.read(bytes_to_read)
        if not buf:
            break
        md5.update(buf)
    content_md5 = base64.standard_b64encode(md5.digest())
```

设置文件元信息

文件元信息(Object Meta)，是对用户在向BOS上传文件时，同时对文件进行的属性描述，主要分为分为两种：设置HTTP标准属性（HTTP Headers）和用户自定义的元信息。

设定Object的Http Header BOS Python SDK本质上是调用后台的HTTP接口，因此用户可以在上传文件时自定义Object的Http Header。常用的http header说明如下：

名称	描述	默认值
Cache-Control	指定该Object被下载时的网页的缓存行为	无
Content-Encoding	表示消息主体进行了何种方式的内容编码转换	无
Content-Disposition	指示MINME用户代理如何显示附加的文件，打开或下载，及文件名称	无
Expires	缓存过期时间	无

参考代码如下：

- 从字符串中上传带有特定header的object

```
user_headers = {"header_key":"header_value"}
#从字符串中上传带有特定header的object
bos_client.put_object_from_string(bucket=bucket_name,
                                  key=object_key,
                                  data=string,
                                  user_headers=user_headers)
#从文件中上传带有特定header的object
bos_client.put_object_from_file(bucket=bucket_name,
                                key=object_key,
                                file_name=file,
                                user_headers=user_headers)
```

用户自定义元信息

BOS支持用户自定义元数据来对Object进行描述。如下代码所示：

```
#用户自定义元数据
user_metadata = {"name":"my-data"}
#从字符串中上传带有用户自定义meta的object
bos_client.put_object_from_string(bucket=bucket_name,
                                  key=object_key,
                                  data=string,
                                  user_metadata=user_metadata)
#从文件中上传带有用户自定义meta的object
bos_client.put_object_from_file(bucket=bucket_name,
                                key=object_key,
                                file_name=file,
                                user_metadata=user_metadata)
```

> **提示：**
>
> - 在上面代码中，用户自定义了一个名字为"name"，值为"my-data"的元数据
> - 当用户下载此Object的时候，此元数据也可以一并得到
> - 一个Object可以有多个类似的参数，但所有的User Meta总大小不能超过2KB

设置Object的Copy属性

BOS同时会提供copy_object接口用于将一个已经存在的Object拷贝到另外一个Object，拷贝过程中会对源Object的Etag或修改状态进行判断，根据判断结果决定是否执行拷贝。详细的参数解释如下：

名称	类型	描述	是否必需
x-bce-copy-source-if-match	String	如果源Object的ETag值和用户提供的ETag相等，则执行拷贝操作，否则拷贝失败。	否
x-bce-copy-source-if-none-match	String	如果源Object的ETag和用户提供的ETag不相等，则执行拷贝操作，否则拷贝失败。	否
x-bce-copy-source-if-unmodified-since	String	如果源object在x-bce-copy-source-if-unmodified-since之后没被修改，则执行拷贝操作，否则拷贝失败。	否
x-bce-copy-source-if-modified-since	String	如果源object在x-bce-copy-source-if-modified-since之后被修改了，则执行拷贝操作，否则拷贝失败。	否

对应的示例代码：

```
copy_object_user_headers = {"copy_header_key":"copy_header_value"}

bos_client.copy_object(source_bucket_name = bucket_name,
                       source_key = object_name,
                       target_bucket_name = bucket_name,
                       target_key = object_name,
                       user_metadata = user_metadata,
                       user_headers = user_headers,
                       copy_object_user_headers = copy_object_user_headers)
```

上传Object时设置存储类型

BOS支持标准存储, 低频存储，冷存储和归档存储，上传Object并存储为某种存储类型时通过指定StorageClass实现，默认为标准存储，四种存储类型对应的参数如下：

存储类型	参数
标准存储	STANDARD
低频存储	STANDARD_IA
冷存储	COLD
归档存储	ARCHIVE

以低频存储和归档存储为例，代码如下：

```
from baidubce.services.bos import storage_class
#从文件中上传冷存储类型的Object
bos_client.put_object_from_file(bucket=bucket_name,
                                key=object_key,
                                file_name=file,
                                storage_class=storage_class.COLD)
#从字符串上传冷存储类型的Object
bos_client.put_object_from_string(bucket=bucket_name,
                                  key=object_key,
                                  data=string,
                                  storage_class=storage_class.COLD)
#从文件中上传归档存储类型的Object
bos_client.put_object_from_file(bucket=bucket_name,
                                key=object_key,
                                file_name=file,
                                storage_class=storage_class.ARCHIVE)
```

追加上传

上文介绍的简单上传方式，创建的Object都是Normal类型，用户不可再进行追加写，这在日志、视频监控、视频直播等数据复写较频繁的场景中使用不方便。

正因如此，百度智能云BOS支持AppendObject，即以追加写的方式上传文件。通过AppendObject操作创建的Object类型为Appendable Object，可以对该Object追加数据。AppendObject大小限制为0~5G。归档存储类型不支持追加上传。

通过AppendObject方式上传示例代码如下：

```
#上传appendable object。其中“content_md5(data)”表示需要用户自行计算上传数据的md5值。
#content_md5计算方法是对数据执行md5算法获取128位二进制数据，再进行base64编码。示例见上文“简单上传”部分。
#其中“content_length(data)”表示需要用户自行计算上传数据的长度
response = bos_client.append_object(bucket_name=bucket_name,
                                    key=object_key,
                                    data=data,
                                    content_md5=content_md5(data), content_length=content_length(data))
#获取下次追加写的位置
next_offset = response.metadata.bce_next_append_offset
bos_client.append_object(bucket_name=bucket_name,
                         key=object_key,
                         data=next_data,
                         content_md5=content_md5(next_data), content_length=content_length(next_data),
                         offset=next_offset)

#从字符串上传appendable object
from baidubce.services.bos import storage_class
bos_client.append_object_from_string(bucket_name=bucket_name,
                                     key=object_key,
                                     data=string,
                                     offset=offset,
                                     storage_class=storage_class.STANDARD,
                                     user_headers=user_headers)
```

分块上传

除了通过putObject接口上传文件到BOS以外，BOS还提供了另外一种上传模式 —— Multipart Upload。用户可以在如下的应用场景内（但不仅限于此），使用Multipart Upload上传模式，如：

- 需要支持断点上传。
- 上传超过5GB大小的文件。
- 网络条件较差，和BOS的服务器之间的连接经常断开。
- 需要流式地上传文件。
- 上传文件之前，无法确定上传文件的大小。

下面将介绍分步实现Multipart Upload。

初始化Multipart Upload

BOS使用initiate_multipart_upload方法来初始化一个分块上传事件：

```
upload_id = bos_client.initiate_multipart_upload(bucket_name, object_key).upload_id
```

该方法会返回InitMultipartUploadResponse对象，此对象中包含uploadId参数，用来表示此次的上传事件。

带有特定header的分块上传的初始化

```
bos_client.initiate_multipart_upload(bucket_name=bucket,
                                     key=object_key,
        user_headers=user_headers)
```

其中，header可设置的属性有："Cache-Control"、"Content-Encoding"、"Content-Disposition"、"Expires"，get-object和get-object-meta两个接口会返回设置的这四个header。

低频、冷存储、归档存储存储分块上传的初始化

低频存储分块上传的初始化需要指定storage_class，请参考以下代码(冷存储以此类推)：

```
from baidubce.services.bos import storage_class

bos_client.initiate_multipart_upload(bucket_name=bucket,
                                     key=object_key,
                                     storage_class = storage_class.STANDARD_IA)
```

上传分块

初始化完成后，进行分块上传：

```
 left_size = os.path.getsize(file_name)
# left_size用于设置分块开始位置
# 设置分块的开始偏移位置
offset = 0

part_number = 1
part_list = []

while left_size > 0:
 # 设置每块为5MB
    part_size = 5 * 1024 * 1024
    if left_size < part_size:
        part_size = left_size

    response = bos_client.upload_part_from_file(
        bucket_name, object_key, upload_id, part_number, part_size, file_name, offset)


    left_size -= part_size
    offset += part_size
    part_list.append({
        "partNumber": part_number,
        "eTag": response.metadata.etag
    })

    part_number += 1
```

注意：

1. offset参数以字节为单位，为分块的开始偏移位置。
2. size参数以字节为单位，定义每个分块的大小，除最后一个Part以外，其他的Part大小都要大于5MB。但是Upload Part接口并不会立即校验上传Part的大小；只有当调用complete_multipart_upload()的时候才会校验。
3. 为了保证数据在网络传输过程中不出现错误，建议您在Upload Part后，使用每个分块BOS返回的Content-MD5值分别验证已上传分块数据的正确性。当所有分块数据合成一个Object后，不再含MD5值。
4. Part号码的范围是1~10000。如果超出这个范围，BOS将返回InvalidArguement的错误码。
5. 每次上传Part时都要把流定位到此次上传块开头所对应的位置。
6. 每次上传Part之后，BOS的返回结果会包含一个etag与块编号（partNumber），在后续完成分块上传的步骤中会用到它，因此需要将其保存起来。一般来讲这些 etag和partNumber将被保存到List中。

完成分块上传

```
bos_client.complete_multipart_upload(bucket_name, object_key, upload_id, part_list)
```

其中，part_list类型是list,里面每个元素是个dict，每个dict包含两个关键字，一个是partNumber, 一个是eTag。

示例如下：

```
[{'partNumber': 1, 'eTag': 'f1c9645dbc14efddc7d8a322685f26eb'}, {'partNumber': 2, 'eTag': 'f1c9645dbc14efddc7d8a322685f26eb'}, {'partNumber': 3, 'eTag': '93b885adfe0da089cdf634904fd59f71'}]
```

该方法返回的解析类中可供调用的参数有：

参数	说明
bucket	Bucket名称
key	Object名称
e_tag	每个上传分块的ETag
location	Object的URL

注意：此对象中包含的ETag是上传分块过程中每个Part的ETag，BOS收到用户提交的Part列表后，会逐一验证每个数据Part的有效性。当所有的数据Part验证通过后，BOS将把这些数据part组合成一个完整的Object。

取消分块上传事件

用户可以使用abort_multipart_upload方法取消分块上传：

```
bos_client.abort_multipart_upload(bucket_name, object_key, upload_id = upload_id)
```

获取未完成的分块上传事件

用户可以使用如下两种方法获取Bucket中未完成的分块上传事件：

方法一：

```
 response = bos_client.list_multipart_uploads(bucket_name)
for item in response.uploads:
 print(item.upload_id)
```

list_multipart_uploads每次BOS最多返回1000个Multipart Upload，BOS支持prefix和delimiter过滤。

list_multipart_uploads方法可供调用的参数还有：

名称	类型	描述	是否必需
delimiter	String	分隔符; 主要应此项实现list文件夹的逻辑	否
key_marker	String	Object按照字典序排序后，本次从keyMarker的后面的一条开始返回	否
max_uploads	Int	本次请求返回Multipart Uploads的最大数目,默认1000，最大1000	否
prefix	String	key前缀，限定返回的object key必须以此为前缀	否

`list_multipart_uploads`方法返回的解析类中可供调用的参数有：

参数	说明
bucket	Bucket名称
key_marker	开始上传的分块Object名称
next_key_marker	当指定了delimiter且IsTruncated true时，才返回此项，作为下次查询marker的值
is_truncated	指明是否所有查询都返回了；false-本次已经返回所有结果，true-本次还没有返回所有结果
prefix	匹配以prefix开始到第一次出现Delimiter字符之间的object作为一组元素返回
common_prefixes	仅当指定delimiter，才会返回此项
delimiter	查询的结束符
max_uploads	请求返回的最大数目
uploads	全部未完成的分快上传事件容器
+ owner	对应Bucket所属用户信息
+ id	Bucket Owner的用户id
+ display_name	Bucket Owner的名称
+ key	分块所属Object名称
+ upload_id	分块上传id
+ initiated	分块上传开始时间

`list_all_multipart_uploads`方法返回uploads的生成器（Generator），并且不受单次最大返回1000个结果的限制，会返回所有的结果。

方法二：

```
uploads = bos_client.list_all_multipart_uploads(bucket_name)
for item in uploads:
    print(item.upload_id)
```

获取所有已上传的块信息

用户可以使用如下两种方法获取某个上传事件中所有已上传的块：

方法一：

```
response = bos_client.list_parts(bucket_name, object_key, upload_id)
for item in response.parts:
    print(item.part_number)
```

> **注意:**
>
> 1. BOS按照PartNumber升序排序。
> 2. 由于网络传输可能出错，所以不推荐用ListParts出来的结果生成最后CompleteMultipartUpload的Part列表。

`list_parts`方法可供调用的参数还有：

名称	类型	描述	是否必需
max_parts	Int	BOS一次最多返回的part数目，默认1000，最大1000	否
part_number_marker	Int	按照partNumber排序，本次请求的起始part从此partNumber的下一个开始返回	否

`list_parts`方法返回的解析类中可供调用的参数有：

参数	说明
bucket	Bucket名称
key	Object名称
initiated	本次分块上传开始时间
max_parts	请求返回的最大数目
is_truncated	指明是否所有查询都返回了；false-本次已经返回所有结果，true-本次还没有返回所有结果
storage_class	Object的存储类型，目前分为标准类型STANDARD，低频类型STANDARD_IA、冷存储类型COLD 和归档类型ARCHIVE
part_number_marker	分块开始标记位
parts	分块列表，list类型
+part_number	分块编号
+last_modified	此分块最后一次被修改的时间
+e_tag	每个上传分块的ETag
+size	分块内容的大小（字节数）
upload_id	本次分块上传的id
owner	对应bucket所属用户信息
+id	Bucket owner的用户id
+display_name	Bucket owner的名称
next_part_number_marker	本次请求返回的最后一条记录的partNumber，可以作为下一次请求的**part_number_marker**

方法二：

```
parts = bos_client.list_all_parts(bucket_name, object_key, upload_id = upload_id)
for item in parts:
    print(item.part_number)
```

list_all_parts方法返回parts的生成器（Generator），并且不受单次最大返回1000个结果的限制，会返回所有的结果。

获取分块上传的Object的存储类型

```
response = bos_client.list_parts(bucket_name=bucket,
                key=object_key,
                upload_id=upload_id)

print(response.storage_class)
```

封装分块上传

在Python SDK中，BOS为用户提供了put_super_obejct_from_file()接口，它对分块上传涉及到的initiate_multipart_upload、upload_part_from_file、complete_multipart_upload三个方法进行封装，用户只需调用该接口即可完成分块上传。

```
import multiprocessing

file_name = "/path/to/file.zip"
result = bos_client.put_super_obejct_from_file(bucket_name, key, file_name,
            chunk_size=5, thread_num=multiprocessing.cpu_count())
if result:
    print("Upload success!")
```

方法可供调用的参数还有：

名称	类型	描述	是否必需
chunk_size	int	分块大小，单位MB。默认为5MB	否
thread_num	int	分块上传中线程池中线程的数量，默认等于CPU的核数	否

若一个大文件耗时很长，用户想结束分块上传，可调用UploadTaskHandle中的cancel()方法实现取消分块上传操作。示例如下：

```
import threading
from baidubce.services.bos.bos_client import UploadTaskHandle

file_name = "/path/to/file.zip"
uploadTaskHandle = UploadTaskHandle()
t = threading.Thread(target=bos_client.put_super_obejct_from_file, args=(bucket_name, key, file_name),
        kwargs={
            "chunk_size": 5,
            "thread_num": multiprocessing.cpu_count(),
            "uploadTaskHandle": uploadTaskHandle
            })
t.start()
time.sleep(2)
uploadTaskHandle.cancel()
t.join()
```

断点续传上传

当用户向BOS上传大文件时，如果网络不稳定或者遇到程序崩等情况，则整个上传就失败了，失败前已经上传的部分也作废，用户不得不重头再来。这样做不仅浪费资源，在网络不稳定的情况下，往往重试多次还是无法完成上传。 基于上述场景，BOS提供了断点续传上传的能力:

- 当网络情况一般的情况下，建议使用三步上传方式，将object分为1Mb的块，参考分块上传。
- 当您的网络情况非常差，推荐使用appendObject的方式进行断点续传，每次append 较小数据256kb，参考追加上传。

> 提示

- 断点续传是分片上传的封装和加强，是用分片上传实现的；
- 文件较大或网络环境较差时，推荐使用分片上传；

抓取上传 如下代码用于从指定URL抓取资源，并将资源存储到指定的Bucket中。此操作需要请求者对该Bucket有写权限，每次只能抓取一个Object，且用户可以自定义Object的名称，默认同步抓取。详情可参考FetchObject接口。

```
 from baidubce.services.bos.bos_client import FETCH_MODE_ASYNC
from baidubce.services.bos import storage_class
fetch_url="<YOUR_URL>"
// 默认同步抓取
bos_client.fetch_object(bucket_name, object_key, fetch_url)

// 异步抓取
response = bos_client.fetch_object(bucket_name,
                                   object_key,
                                   fetch_url,
                                   fetch_mode=FETCH_MODE_ASYNC,
                                   storage_class=storage_class.COLD)
print("jobId:{}, return code:{}, return message:{}".format(response.job_id,
                                                           response.code,
                                                           response.message))
```

获取上传进度 Python SDK支持在上传过程中实时提供上传进度信息。目前支持简单上传，追加上传，分块上传。需要在对应接口上增加progress_callback参数，并提供进度条回调函数，也可调用工具类中的默认进度条回调函数。

回调函数示例如下：

```
 def percentage(consumed_bytes, total_bytes):
  """进度条回调函数，计算当前完成的百分比

  :param consumed_bytes: 已经上传/下载的数据量
  :param total_bytes: 总数据量
  """
  if total_bytes:
    rate = int(100 * (float(consumed_bytes) / float(total_bytes)))
    print('\r{0}% '.format(rate))
    sys.stdout.flush()

###### progress_callback为可选参数，用于实现进度条功能.
bos_client.put_object(bucket_name, object_key, data, content_length,content_md5, progress_callback=percentage)
```

推荐使用工具类中的默认进度条回调函数(utils.default_progress_callback)，目前支持百分比和进度条展示,示例如下：

```
 ###### 引入 SDK 工具类包
from baidubce import utils

###### progress_callback为可选参数，用于实现进度条功能.
bos_client.put_object(bucket_name, object_key, data, content_length,content_md5, progress_callback=utils.default_progress_callback)
```

- put_object 示例代码

```
 ###### 引入 SDK 工具类包
from baidubce import utils

###### progress_callback为可选参数，用于实现进度条功能.
data = open(file_name, 'rb')
bos_client.put_object(bucket_name, object_key, data, content_length, content_md5,
progress_callback=utils.default_progress_callback)

###### 从字符串中上传的Object
bos_client.put_object_from_string(bucket_name, object_key, string,
progress_callback=utils.default_progress_callback)

###### 从文件中上传的Object
bos_client.put_object_from_file(bucket_name, object_key, file_name,
progress_callback=utils.default_progress_callback)
```

- append_object 示例代码

```
 ###### 引入 SDK 工具类包
from baidubce import utils

###### progress_callback为可选参数，用于实现进度条功能.
###### 上传appendable object。
response = bos_client.append_object(bucket_name=bucket_name,
                                    key=object_key,
                                    data=data,
                                    content_md5=content_md5(data), content_length=content_length(data),
                                    progress_callback=utils.default_progress_callback)

###### 从字符串上传appendable object
result = bos_client.append_object_from_string(bucket_name=bucket_name,
                                              key=object_key,
                                              data=String, progress_callback=utils.default_progress_callback)
```

- upload_part_from_file 示例代码

```
###### 引入 SDK 工具类包
from baidubce import utils

###### progress_callback为可选参数，用于实现进度条功能.
bos_client.upload_part_from_file(
        bucket_name, key, upload_id, part_number, part_size, file_name, offset, progress_callback=utils.default_progress_callback)~~~~
```

支持单链接限速 对象存储BOS对单Bucket的公网带宽阈值为10Gbit/s，内网带宽阈值为50Gbit/s，当用户的上传或下载占用带宽达到带宽限制阈值时，会返回RequestRateLimitExceeded的错误码。为保证用户能够正常使用服务，BOS支持在进行上传、下载等行为时进行流量控制，保证大流量服务占用带宽不会对其他应用服务造成影响。

上传类请求接口示例 限速值的取值范围为819200~838860800，单位为bit/s，即100KB/s~100MB/s。限速值取值必须为数字，BOS将按照指定的限速值对此次请求进行限速，当限速值不在此范围或不合法时将返回400错误码。

```
traffic_limit_speed = 819200 * 5
###### put a file as object
_create_file(file_name, 5 * 1024 * 1024)
bos_client.put_object_from_file(bucket_name, key, file_name, traffic_limit=traffic_limit_speed)

###### multi-upload operation samples
###### put a super file to object
_create_file(file_name, 10 * 1024 * 1024)

###### SuperFile step 1: init multi-upload
upload_id = bos_client.initiate_multipart_upload(bucket_name, key).upload_id

###### SuperFile step 2: upload file part by part
left_size = os.path.getsize(file_name)
offset = 0
part_number = 1
part_list = []
while left_size > 0:
    part_size = 5 * 1024 * 1024
    if left_size < part_size:
        part_size = left_size

    response = bos_client.upload_part_from_file(bucket_name, key, upload_id, part_number, part_size, file_name, offset, traffic_limit=traffic_limit_speed)
    left_size -= part_size
    offset += part_size
    # your should store every part number and etag to invoke complete multi-upload
    part_list.append({
        "partNumber": part_number,
        "eTag": response.metadata.etag
    })
    part_number += 1

###### copy a object
bos_client.copy_object(source_bucket, source_key, target_bucket, target_key, traffic_limit=traffic_limit_speed)

###### append object
bos_client.append_object(bucket_name, key, traffic_limit=traffic_limit_speed)

###### upload part copy
bos_client.upload_part_copy(source_bucket, source_key, target_bucket, target_key, upload_id, part_number,
 part_size, offset, traffic_limit=traffic_limit_speed)
```

下载文件

BOS Python SDK提供了丰富的文件下载接口，用户可以通过以下方式从BOS中下载文件：

- 简单流式下载
- 下载到本地文件
- 下载为字符串
- 断点续传下载
- 范围下载
- 获取下载进度

简单的读取Object

用户可以通过如下代码将Object读取到一个流中：

```
response = bos_client.get_object(bucket_name, object_key)
s = response.data

###### 处理Object
...

###### 关闭流
response.data.close()
```

直接下载Object到文件或字符串

用户可以参考如下代码将Object下载到指定文件：

```
bos_client.get_object_to_file(bucket_name, object_key, file_name)
```

用户可以参考如下代码将Object下载到字符串：

```
result = bos_client.get_object_as_string(bucket_name, object_key)
print(result)
```

范围下载

为了实现更多的功能，可以通过使用指定range参数来指定下载范围，实现更精细化地获取Object。如果指定的下载范围是0 - 100，则返回第0到第100个字节的数据，包括第100个，共101字节的数据，即[0, 1000]。

```
range = [0,1000]
###### 返回指定范围的Object数据
print(bos_client.get_object_as_string(bucket_name, object_key, range = range))
###### 返回指定范围的Object数据到文件中
bos_client.get_object_to_file(bucket_name, object_key, file_name, range = range)
```

通过get_object_as_string和get_object_to_file的range参数可以设置返回Object的范围。用户也可以用此功能实现文件的分段下载和断点续传。

其他使用方法

获取Object的存储类型

Object的storage class属性分为STANDARD(标准存储), STANDARD_IA(低频存储)，COLD(冷存储)和ARCHIVE(归档存储)，通过如下代码可以代码获得Object的storage class属性：

```
response = bos_client.get_object_meta_data(bucket_name, object_key)
print(response.metadata.bce_storage_class)
```

只获取ObjectMetadata

通过get_object_meta_data方法可以只获取Object的元数据，而非Object实体。如下代码所示：。

```
response = bos_client.get_object_meta_data(bucket_name, object_key)
```

get_object_meta_data方法返回的解析类中可供调用的参数有：

参数	说明
content_length	Object的大小
e_tag	Object的HTTP协议实体标签
bce_meta	如果在PutObject指定了user_metadata自定义meta，则返回此项（）
storageClass	Object的存储类型
bce_restore	归档存储对象处于正在取回或已经取回时返回。对于正在取回的归档对象bce_restore取值为 ongoing-request="true"；对于已取回的归档对象bce_restore取值为 ongoing-request="false", expiry-date="Wed, 07 Nov 2019 00:00:00 GMT"。其中expiry-date表示对象取回后的失效时间，此时间为格林尼治时间。

获取下载进度 Python SDK支持在下载过程中实时提供下载进度信息。目前支持下载object 到文件。需要在对应接口上增加progress_callback参数，并提供进度条回调函数，也可调用工具类中的默认进度条回调函数。

回调函数示例如下：

```
def percentage(consumed_bytes, total_bytes):
    """进度条回调函数，计算当前完成的百分比

    :param consumed_bytes: 已经上传/下载的数据量
    :param total_bytes: 总数据量
    """
    if total_bytes:
        rate = int(100 * (float(consumed_bytes) / float(total_bytes)))
        print('\r{0}% '.format(rate))
        sys.stdout.flush()

###### progress_callback为可选参数，用于实现进度条功能.~~~~
bos_client.get_object_to_file(bucket_name, key, download, progress_callback=percentage)
```

推荐使用工具类中的默认进度条回调函数(utils.default_progress_callback)，目前支持百分比和进度条展示,示例如下：

```
###### 引入 SDK 工具类包
from baidubce import utils

###### progress_callback为可选参数，用于实现进度条功能.
bos_client.get_object_to_file(bucket_name, key, download, progress_callback=utils.default_progress_callback)
```

- get_object_to_file 代码示例

```
###### 引入 SDK 工具类包
from baidubce import utils

###### progress_callback为可选参数，用于实现进度条功能.
bos_client.get_object_to_file(bucket_name, key, download, progress_callback=utils.default_progress_callback)
```

支持单链接限速 对象存储BOS对单Bucket的公网带宽阈值为10Gbit/s，内网带宽阈值为50Gbit/s，当用户的上传或下载占用带宽达到带宽限制阈值时，会返回RequestRateLimitExceeded的错误码。为保证用户能够正常使用服务，BOS支持在进行上传、下载等行为时进行流量控制，保证大流量服务占用带宽不会对其他应用服务造成影响。

下载类请求接口示例 限速值的取值范围为819200~838860800，单位为bit/s，即100KB/s~100MB/s。限速值取值必须为数字，BOS将按照指定的限速值对此次请求进行限速，当限速值不在此范围或不合法时将返回400错误码。

```
traffic_limit_speed = 819200 * 5

###### get obj url by traffic limit
param = {}
param[b'x-bce-traffic-limit'] = traffic_limit_speed
url = bos_client.generate_pre_signed_url(bucket_name, key, timestamp=1649923427,
                                         expiration_in_seconds=1000,
                                         params=param)
__logger.debug("[Sample] get object url is  %s", url)

###### get object into file
bos_client.get_object_to_file(bucket_name, key, download, traffic_limit=traffic_limit_speed)
__logger.debug("[Sample] get object into file, file size:%s", os.path.getsize(download))
```

取回归档存储类型文件

取回归档类型文件 用户上传归档文件后，归档文件是处于冰冻状态的，如果要下载归档文件，必须先取回归档存储文件。请求者必须有归档存储文件的读权限，并且归档存储文件处于冰冻状态。

取回归档文件，示例如下：

```
###### 取回归档文件并设置解冻后持续时间为2天
bos_client.restore_object(bucket_name, target_key)
```

参数	类型	说明	是否必须
days	int	表示解冻后状态的持续时间，要求是数字，单位是天，取值大于0且不超过30即可。默认解冻持续天数是7天,最长30天。	否
tier	string	解冻优先级。可取值范围如下： • Expedited：加急取回，表示30min内完成取回。 • Standard：标准取回，表示2~5小时内完成取回 • LowCost：延缓取回，表示12小时内完成取回	否

判断归档文件是否取回

对于归档存储对象，如果正在取回或已经取回时返回，获取ObjectMetadata时会返回bce_restore。对于正在取回的归档对象bce_restore取值为 `ongoing-request="true"`；对于已取回的归档对象bce_restore取值为 `ongoing-request="false", expiry-date="Wed, 07 Nov 2019 00:00:00 GMT"`。其中expiry-date表示对象取回后的失效时间。

示例如下：

```
response = bos_client.get_object_meta_data(bucket_name, object_key)
if response.metadata.bce_restore is not None:
    if response.metadata.bce_restore.find("expiry-date") >= 0:
        print("archive object is restored.")
    else:
        print("archive object is restoring.")
else:
    print("archvie object is freezed.")
```

变更文件存储等级

上文中已提到，BOS支持为文件赋予STANDARD(标准存储), STANDARD_IA(低频存储)、COLD(冷存储)、ARCHIVE(归档存储)四种存储类型。同时，BOS python SDK也支持用户对特定文件执行存储类型变更的操作。 涉及到的参数如下：

参数	说明
x-bce-storage-class	指定Object的存储类型，STANDARD_IA代表低频存储，COLD代表冷存储，ARCHIVE代表冷存储，不指定时默认是标准存储类型。

> **注意**
>
> • 调用copy_object()接口时如果源object是归档类型，需要先调用restore_object()取回归档文件。

示例如下：

```
###### 标准存储转为低频存储
bos_client.copy_object(source_bucket_name, source_key,
                       target_bucket_name, target_key,
                       storage_class = storage_class.STANDARD_IA)
###### 低频存储转为冷存储
bos_client.copy_object(source_bucket_name, source_key,
                       target_bucket_name, target_key,
                       storage_class = storage_class.COLD)
```

获取文件下载URL

用户可以通过如下示例代码获取指定Object的URL：

```
url = bos_client.generate_pre_signed_url(bucket_name, object_key, timestamp, expiration_in_seconds)
```

> **说明：**
>
> • 用户在调用该函数前，需要手动设置endpoint为所属区域域名。百度智能云目前开放了多区域支持，请参考区域选择说明。目前支持“华北-北京”、“华南-广州”和“华东-苏州”三个区域。北京区域：http://bj.bcebos.com，广州区域：http://gz.bcebos.com，苏州区域：http://su.bcebos.com。
>
> • timestamp为可选参数，不配置时，默认值为当前时间。
>
> • timestamp为时间戳，标识URL有效起始时间，timestamp=int(time.time())，并需要* import time。
>
> • expriation_in_seconds用来设置URL的有效时长，为可选参数，不配置时，默认值为1800秒。如果要设置为永久不失效的时间，可以将expirationInSeconds参数设置为 -1，不可设置为其他负数。

现已支持通过STS方式获取，配置STS访问BOS见初始化页面，代码示例如下：

```
 ###### 引入STS配置文件
import sts_sample
import time

###### create a sts bos client
sts_bos_client = BosClient(sts_sample.config)
timestamp =int(time.time())
url = sts_bos_client.generate_pre_signed_url(bucket_name, key, timestamp,
                                             expiration_in_seconds=3600)
```

列举存储空间中的文件

BOS SDK支持用户通过以下两种方式列举出object：

- 简单列举
- 通过参数复杂列举

除此之外，用户还可在列出文件的同时模拟文件夹

简单列举

当用户完成一系列上传后，可能会需要查看在指定Bucket中的全部Object，可以通过如下代码实现：

```
 response = bos_client.list_objects(bucket_name)
for object in response.contents:
    print(object.key)
```

> **注意：** 1. 默认情况下，如果Bucket中的Object数量大于1000，则只会返回1000个Object，并且返回结果中is_truncated值为True，并返回next_marker做为下次读取的起点。 2. 若想增大返回Object的数目，可以使用Marker参数分次读取。

也可以一次列举当前Bucket的所有的Object。

```
for object in bos_client.list_all_objects(bucket_name):
    print(object.key)
```

通过参数复杂列举

list_objects方法其他可选的参数有：

参数	说明
prefix	限定返回的object key必须以Prefix作为前缀。
delimiter	是一个用于对Object名字进行分组的字符。所有名字包含指定的前缀且第一次出现Delimiter字符之间的object作为一组元素: CommonPrefixes。
max_keys	限定此次返回object的最大数，此数值不能超过1000，如果不设定，默认为1000。
marker	设定结果从Marker之后按字母排序的第一个开始返回。

> **注意：**
>
> 1. 如果有Object以Prefix命名，当仅使用Prefix查询时，返回的所有Key中仍会包含以Prefix命名的Object，详见递归列出目录下所有文件。
> 2. 如果有Object以Prefix命名，当使用Prefix和Delimiter组合查询时，返回的所有Key中会有Null，Key的名字不包含Prefix前缀，详见查看目录下的文件和子目录。

下面我们分别以几个案例说明通过参数列举的方法：

指定最大返回条数

```
  max_keys = 500
# 指定最大返回条数为500
response = bos_client.list_objects(bucket_name, max_keys = max_keys)
for obj in response.contents:
    print(obj.key)
```

返回指定前缀的object

```
  prefix = "test"
# 指定返回前缀为test的object
response = bos_client.list_objects(bucket_name, prefix = prefix)
for obj in response.contents:
    print(obj.key)
```

从指定Object后返回

```
  marker = "object"
# 用户可以定义不包括某object，从其之后开始返回
response = bos_client.list_objects(bucket_name, marker = marker)
for obj in response.contents:
    print(obj.key)
```

分页获取所有Object

```
isTruncated = True
# 用户可设置每页最多500条记录
max_keys = 500
marker = None
while isTruncated:
    response = bos_client.list_objects(bucket_name, max_keys = max_keys, marker=marker)
    for obj in response.contents:
        print(obj.key)
    isTruncated = response.is_truncated
    marker = getattr(response,'next_marker',None)
```

分页获取所有特定Object后的结果

```
# 用户可设置每页最多500条记录，并从某特定object之后开始获取
max_keys = 500
marker = "object"
isTruncated = True
while isTruncated:
    response = bos_client.list_objects(bucket_name, max_keys = max_keys, marker=marker)
    for obj in response.contents:
        print(obj.key)
    isTruncated = response.is_truncated
    marker = getattr(response,'next_marker',None)
```

分页获取所有指定前缀的Object结果

```
# 用户可设置分页获取指定前缀的Object，每页最多500条记录
max_keys = 500
prefix = "object"
isTruncated = True
while isTruncated:
    response = bos_client.list_objects(bucket_name, prefix = prefix)
    for obj in response.contents:
        print(obj.key)
    isTruncated = response.is_truncated
    marker = getattr(response,'next_marker',None)
```

list_objects方法返回的解析类中可供调用的参数有：

参数	说明
name	Bucket名称
prefix	匹配以prefix开始到第一次出现Delimiter字符之间的object作为一组元素返回
marker	本次查询的起点
max_keys	请求返回的最大数目
is_truncated	指明是否所有查询都返回了；false-本次已经返回所有结果，true-本次还没有返回所有结果
contents	返回的一个Object的容器
+key	Object名称
+last_modified	此Object最后一次被修改的时间
+e_tag	Object的HTTP协议实体标签
+size	Object的内容的大小（字节数）
+owner	Object对应Bucket所属用户信息
++id	Bucket Owner的用户ID
++display_name	Bucket Owner的名称
next_marker	只要IsTruncated为true，就会返回next_marker，作为下次查询marker的值
common_prefixes	仅当指定delimiter，才会返回此项

list_all_objects方法返回contents的生成器（Generator），并且不受单次最大返回1000个结果的限制，会返回所有的结果。

模拟文件夹功能

在BOS的存储结果中是没有文件夹这个概念的，所有元素都是以Object来存储，但BOS的用户在使用数据时往往需要以文件夹来管理文件。 因此，BOS提供了创建模拟文件夹的能力，其本质上来说是创建了一个size为0的Object。对于这个Object可以上传下载，只是控制台会对以"/"结尾的Object以文件夹的方式展示。

用户可以通过 Delimiter 和 Prefix 参数的配合模拟出文件夹功能。Delimiter 和 Prefix 的组合效果是这样的：

如果把 Prefix 设为某个文件夹名，就可以罗列以此 Prefix 开头的文件，即该文件夹下递归的所有的文件和子文件夹（目录）。文件名在Contents中显示。 如果再把 Delimiter 设置为 "/" 时，返回值就只罗列该文件夹下的文件和子文件夹（目录），该文件夹下的子文件名（目录）返回在 CommonPrefixes 部分，子文件夹下递归的文件和文件夹不被显示。

假设Bucket中有5个文件：bos.jpg，fun/，fun/test.jpg，fun/movie/001.avi，fun/movie/007.avi，把 "/" 符号作为文件夹的分隔符。

如下是几个应用方式：

列出Bucket内所有文件

当用户需要获取Bucket下的所有文件时，可以参考分页获取所有Object

递归列出目录下所有文件

可以通过设置 Prefix 参数来获取某个目录下所有的文件：

```
prefix = "fun/"
print("Objects:")
# 递归列出fun目录下的所有文件
response = bos_client.list_objects(bucket_name, prefix = prefix)
for obj in response.contents:
    print(obj.key)
```

输出：

```
Objects:
fun/
fun/movie/001.avi
fun/movie/007.avi
fun/test.jpg
```

查看目录下的文件和子目录

在 Prefix 和 Delimiter 结合的情况下，可以列出目录下的文件和子目录：

```
# "/" 为文件夹的分隔符
delimiter = "/"
prefix = "fun/"
# 列出fun目录下的所有文件和文件夹
response = bos_client.list_objects(bucket_name, prefix = prefix, delimiter = delimiter)
print("Objects:")
for obj in response.contents:
    print(obj.key)

# 遍历所有CommonPrefix
print("CommonPrefixs:")
for obj in response.common_prefixes:
    print(obj.prefix)
```

输出：

```
Objects:
fun/
fun/test.jpg

CommonPrefixs:
fun/movie/
```

返回的结果中， Objects 的列表中给出的是fun目录下的文件。而 CommonPrefixs 的列表中给出的是fun目录下的所有子文件夹。可以看出 fun/movie/001.avi ， fun/movie/007.avi 两个文件并没有被列出来，因为它们属于 fun 文件夹下的 movie 目录。

列举Bucket中object的存储属性

用户除了可以查看指定Bucket中的全部Object，还可以查看Object的存储类型，实现代码如下：

```
response = bos_client.list_objects(bucket_name)
for obj in response.contents:
    print('object:{}, storage_class:{}'.format(obj.key, obj.storage_class))
```

Object权限控制

设置Object的访问权限

目前BOS支持两种方式设置ACL。第一种是使用Canned Acl，在PutObjectAcl的时候，通过头域的"x-bce-acl"或者"x-bce-grant-permission'来设置object访问权限，当前可设置的权限包括private和public-read，两种类型的header不可以同时在一个请求中出现。第二种方式是上传一个ACL文件。 详细信息请参考设置Object权限控制

1. 通过使用头域的"x-bce-acl"来设置object访问权限

```
from baidubce.services.bos import canned_acl
# 设置object为private权限
bos_client.set_object_canned_acl(bucket_name, object_key, canned_acl=canned_acl.PRIVATE)
# 设置object为public-read权限
bos_client.set_object_canned_acl(bucket_name, object_key, canned_acl=canned_acl.PUBLIC_READ)
```

2. 通过使用头域的"x-bce-grant-permission'来设置object访问权限

```
# 授予指定用户对Object的可读权限。
bos_client.set_object_canned_acl(bucket_name, object_key, grant_read='id="12345678dfd5487e99f5c85aca5c1234",id="1234567880274ea5a9d50fe94c151234"')
# 授予指定用户对Object的FULL_CONTROL权限。
bos_client.set_object_canned_acl(bucket_name, object_key, grant_full_control='id="12345678dfd5487e99f5c85aca5c1234",id="1234567880274ea5a9d50fe94c151234"')
```

3. 通过set_object_acl()接口来设置object权限。

```
# 授予指定用户对Object的可读权限。
acl = [{
        "grantee":[{
        "id":"12345678dfd5484399f5c85aca5c1234"
        }],
        "permission":["READ"]
    }]
bos_client.set_object_acl(bucket_name, object_key, acl = acl)
```

查看Object的权限

归档存储类型对象在取回未完成，或者刚上传归档类型文件（时长参考取回时长）时，不能设置Object acl

查看Object的权限如下代码所示：

```
response = bos_client.get_object_acl(bucket_name, object_key)
print("object acl:", response.access_control_list)
```

getObjectAcl方法返回的解析类中可供调用的参数有：

参数	说明
accessControlList	标识Object的权限列表
grantee	标识被授权人
-id	被授权人id
permission	标识被授权人的权限

删除Object的权限

归档存储类型对象在取回未完成，或者刚上传归档类型文件（时长参考取回时长）时，不能删除Object acl

如下代码可以删除Object的权限：

```
bos_client.delete_object_acl(bucket_name,object_key)
```

删除文件

删除一个Object

如下代码删除了一个Object：

```
bos_client.delete_object(bucket_name, object_key)
```

批量删除Object

用户可以通过如下代码批量删除object

```
key_list = [object_key1, object_key2, object_key3]
bos_client.delete_multiple_objects(bucket_name, key_list)
```

> 说明:支持一次请求内最多删除1000个Objcet。

查看文件是否存在

用户可通过如下操作查看某文件是否存在：

```
from baidubce import exception

try:
 response = bos_client.get_object_meta_data(bucket_name, object_key)
 print("Get meta:{}",response.metadata)
except exception.BceError as e:
 print(e)
```

获取及更新文件元信息

文件元信息(Object Metadata)，是对用户上传BOS的文件的属性描述，分为两种：HTTP标准属性（HTTP Headers）和User Meta（用户自定义元信息）。

获取文件元信息

参考只获取ObjectMetadata。

修改文件元信息

BOS修改Object的Metadata通过拷贝Object实现。即拷贝Object的时候，把目的Bucket设置为源Bucket，目的Object设置为源Object，并设置新的Metadata，通过拷贝自身实现修改Metadata的目的。如果不设置新的Metadata，则报错。

归档类型文件不支持修改元信息。

```
user_metadata = {'meta_key': 'meta_value'}
bos_client.copy_object(source_bucket_name = bucket_name,
                       source_key = object_name,
                       target_bucket_name = bucket_name,
                       target_key = object_name,
                       user_metadata = user_metadata)
response = bos_client.get_object_meta_data(bucket_name = bucket_name,
                                           key = object_name)
print(response)
```

拷贝Object

用户可以通过copyObject方法拷贝一个Object，如下代码所示：

```
bos_client.copy_object(source_bucket_name, source_object_key, target_bucket_name, target_object_key)
```

同步Copy功能

当前BOS的CopyObject接口是通过同步方式实现的。同步方式下，BOS端会等待Copy实际完成才返回成功。同步Copy能帮助用户更准确的判断Copy状态，但用户感知的复制时间会变长，且复制时间和文件大小成正比。

同步Copy方式更符合业界常规，提升了与其它平台的兼容性。同步Copy方式还简化了BOS服务端的业务逻辑，提高了服务效率。

归档类型文件如果是源Object，需要先取回归档类型文件。

如果您使用bce-python-sdk-0.8.12版本之前的SDK，有可能会出现复制请求成功，但文件实际复制失败的情况，所以建议您使用最新版本SDK。

分块拷贝

除了通过CopyObject接口拷贝文件以外，BOS还提供了另外一种拷贝模式——Multipart Upload Copy。归档类型文件如果是源Object，需要先取回归档类型文件。

用户可以在如下的应用场景内（但不仅限于此），使用Multipart Upload Copy，如：

- 需要支持断点拷贝。
- 拷贝超过5GB大小的文件。
- 网络条件较差，和BOS的服务器之间的连接经常断开。

下面将介绍分步实现三步拷贝。

三步拷贝包含init、“拷贝分块”和complete三步，其中init和complete的操作同分块上传一致，可直接参考初始化Multipart Upload和完成分块上传。

拷贝分块代码参考（参数定义可参考分块上传）：

```
 left_size = int(bos_client.get_object_meta_data(source_bucket,source_key).metadata.content_length)
###### 设置分块开始位置为left_size

###### 设置分块的开始偏移位置
offset = 0
part_number = 1
part_list = []
while left_size > 0:

    #设置每块为5MB
    part_size = 5 * 1024 * 1024
    if left_size < part_size:
        part_size = left_size

    response = bos_client.upload_part_copy(source_bucket, source_key, target_bucket, target_key, upload_id,part_number, part_size, offset)

    left_size -= part_size
    offset += part_size
    part_list.append({
     "partNumber": part_number,
     "eTag": response.etag
    })

    part_number += 1
```

> 注意:
>
> 1. offset参数以字节为单位，为分块的开始偏移位置。
> 2. size参数以字节为单位，定义每个分块的大小，除最后一个Part以外，其他的Part大小都要大于 5MB。

选取文件

SelectObject接口支持用户对BOS中指定格式(CSV/JSON/Parquet)的object内容执行SQL语句，通过SQL这种结构化查询语言对object内容进行筛选、分析、过滤之后再返回用户需要的文件内容。请参考选取Object

使用限制见“开发者指南”-“使用及管理数据”-“选取object”部分。

查询csv文件 示例：

```
 from baidubce import compat
import base64

# 方便理解，我们先上传一个简单的csv文件
csv_content = """
    1,Maurits,2017-09-1216:32:57,685856330,-540265154.48,true
    2,Iago,2018-02-01 12:25:01,-642946677,3781354659.89,false
    3,Dionisio,2018-02-16 09:52:24,-3823711977,79336720.77,false
    4,Aleen,2018-05-17 11:48:45,-3289131518,1499686289.41,false
    5,Herschel,2019-06-04 02:28:37,3456163349,-3810272511.88,true
    """
bos_client.put_object_from_string(bucket_name, key, csv_content)
# 设置select_object()接口的参数
select_object_args = {
    "expressionType": "SQL",
    "inputSerialization": {
        "compressionType": "NONE",
        "csv": {
            "fileHeaderInfo": "NONE",
            "recordDelimiter": "Cg==",
            "fieldDelimiter": "LA==",
            "quoteCharacter": "Ig==",
            "commentCharacter": "Iw=="
        }
    },
    "outputSerialization": {
        "outputHeader": False,
        "csv": {
            "quoteFields": "ALWAYS",
            "recordDelimiter": "Cg==",
            "fieldDelimiter": "LA==",
            "quoteCharacter": "Ig=="
        }
    },
    "requestProgress": {
        "enabled": True
    }
}
# 设置查询的sql语句，需要经过base64编码的sql语句
sql_exp = "SELECT _1, _2, _6 FROM BosObject"
select_object_args["expression"] = compat.convert_to_string(base64.standard_b64encode(compat.convert_to_bytes(sql_exp)))
# 调用select object接口
select_response = bos_client.select_object(bucket_name, key, select_object_args)
# 获取返回结果的生成器
result = select_response.result()
for msg in result:
    print(msg)
    if msg.headers["message-type"] == "Records":
        print("type: {}, heades: {}, payload: {}, crc: {}".format(msg.type, msg.headers, msg.payload, msg.crc))
    elif msg.headers["message-type"] == "Cont":
        print("type: {}, heades: {}, bytes_scanned: {}, bytes_returned: {},  crc: {}".format(msg.type, msg.headers,
            msg.bytes_scanned, msg.bytes_returned, msg.crc))
    else:
        print("type: {}, heades: {}, crc: {}".format(msg.type, msg.headers, msg.crc))
```

查询json文件 示例：

```
 from baidubce import compat
import base64

# 方便理解，我们先上传一个简单的json文件
json_content = """
{
"name": "Smith",
"age": 16,
"weight": 65.5,
"org": null,
"projects":
    [
     {"project_name":"project1", "completed":false},
     {"project_name":"project2", "completed":true}
    ]
}
"""
bos_client.put_object_from_string(bucket_name, key, json_content)
# 设置select_object()接口的参数
select_object_args = {
    "expressionType": "SQL",
    "inputSerialization": {
        "compressionType": "NONE",
        "json": {
            "type": "DOCUMENT"
        }
    },
    "outputSerialization": {
        "json": {
            "recordDelimiter": "Cg=="
        }
    },
    "requestProgress": {
        "enabled": True
    }
}
# 设置查询的sql语句，需要经过base64编码的sql语句
sql_exp = "select projects from BosObject where name='Smith'"
select_object_args["expression"] = compat.convert_to_string(base64.standard_b64encode(compat.convert_to_bytes(sql_exp)))
# 调用select object接口
select_response = bos_client.select_object(bucket_name, key, select_object_args)
# 获取返回结果的生成器
result = select_response.result()
for msg in result:
    print(msg)
    if msg.headers["message-type"] == "Records":
        print("type: {}, heades: {}, payload: {}, crc: {}".format(msg.type, msg.headers, msg.payload, msg.crc))
    elif msg.headers["message-type"] == "Cont":
        print("type: {}, heades: {}, bytes_scanned: {}, bytes_returned: {},  crc: {}".format(msg.type, msg.headers,
            msg.bytes_scanned, msg.bytes_returned, msg.crc))
    else:
        print("type: {}, heades: {}, crc: {}".format(msg.type, msg.headers, msg.crc))
```

查询Parquet文件

```
 from baidubce import compat
import base64

# 我们先上传一个简单的Parquet文件
'''
parquet文件解析内容
{"Name":"StudentName","Age":20,"Id":0,"Weight":50,"Sex":true,"Day":19240,"Scores":{"computer":80,"math":90,"physics":90}}
{"Name":"StudentName","Age":21,"Id":1,"Weight":50.1,"Sex":false,"Day":19240,"Scores":{"computer":81,"math":91,"physics":91}}
{"Name":"StudentName","Age":22,"Id":2,"Weight":50.2,"Sex":true,"Day":19240,"Scores":{"computer":82,"math":92,"physics":92}}
{"Name":"StudentName","Age":23,"Id":3,"Weight":50.3,"Sex":false,"Day":19240,"Scores":{"computer":83,"math":93,"physics":90}}
{"Name":"StudentName","Age":24,"Id":4,"Weight":50.4,"Sex":true,"Day":19240,"Scores":{"computer":84,"math":94,"physics":91}}
{"Name":"StudentName","Age":20,"Id":5,"Weight":50.5,"Sex":false,"Day":19240,"Scores":{"computer":85,"math":90,"physics":92}}
{"Name":"StudentName","Age":21,"Id":6,"Weight":50.6,"Sex":true,"Day":19240,"Scores":{"computer":86,"math":91,"physics":90}}
{"Name":"StudentName","Age":22,"Id":7,"Weight":50.7,"Sex":false,"Day":19240,"Scores":{"computer":87,"math":92,"physics":91}}
{"Name":"StudentName","Age":23,"Id":8,"Weight":50.8,"Sex":true,"Day":19240,"Scores":{"computer":88,"math":93,"physics":92}}
{"Name":"StudentName","Age":24,"Id":9,"Weight":50.9,"Sex":false,"Day":19240,"Scores":{"computer":89,"math":94,"physics":90}}
'''
bos_client.put_object_from_file(bucket_name, key, parquet_file_name)
# 设置select_object()接口的参数
select_object_args = {
    "expressionType": "SQL",
    "inputSerialization": {
        "compressionType": "NONE",
        "parquet": {}
    },
    "outputSerialization": {
        "json": {
            "recordDelimiter": "Cg=="
        }
    },
    "requestProgress": {
        "enabled": false
    }
}
# 设置查询的sql语句，需要经过base64编码的sql语句
sql_exp = "select * from BosObject s where s.Scores.computer > 85"
select_object_args["expression"] = compat.convert_to_string(base64.standard_b64encode(compat.convert_to_bytes(sql_exp)))
# 调用select object接口
select_response = bos_client.select_object(bucket_name, key, select_object_args)
# 获取返回结果的生成器
result = select_response.result()
for msg in result:
    print(msg)
    if msg.headers["message-type"] == "Records":
        print("type: {}, heades: {}, payload: {}, crc: {}".format(msg.type, msg.headers, msg.payload, msg.crc))
    elif msg.headers["message-type"] == "Cont":
        print("type: {}, heades: {}, bytes_scanned: {}, bytes_returned: {},  crc: {}".format(msg.type, msg.headers,
            msg.bytes_scanned, msg.bytes_returned, msg.crc))
    else:
        print("type: {}, heades: {}, crc: {}".format(msg.type, msg.headers, msg.crc))
```

软链接

创建软链接

本接口用于为BOS中相同bucket下已有的目的object（本文档称作"target object"）创建软链接（Symlink），您可以通过该软链接访问目的object。相关API介绍见PutSymlink。

注意：

- 软链接文件和target object文件需要位于同一个bucket下面。
- 目前支持对标准存储、低频存储、冷存储、归档存储类型的object文件设置软链接。软链接文件本身不支持ARCHIVE类型。如果用户指定ARCHIVE类型，或者不指定类型且Bucket默认存储类型是ARCHIVE时，返回报错InvalidArgument。
- 调用该接口创建软链接时，不会检查目标文件是否存在、以及是否有权限访问目标文件。使用软链接调用GetObject接口时候，会检查用户是否有软链接和目标文件的读取权限、目标文件是否存在。
- 创建软链接需要有相应写权限。
- 如果不指定"forbid_overwrite"，且如果存在与软链接文件同名的object文件时，创建软链接操作时会默认覆盖同名Object。
- 不支持二级软链接。用户创建二级软链时不会进行检查，但是用户无法通过二级软链访问指向的object数据。

创建软链接可选参数:

名称	类型	描述	是否必选
forbid_overwrite	bool	指定创建软链接操作时是否覆盖同名Object。如果不指定x-bce-forbid-overwrite时，默认允许覆盖同名Object。指定x-bce-forbid-overwrite为true时，表示禁止覆盖同名Object；指定x-bce-forbid-overwrite为false时，表示允许覆盖同名Object。	否
storage_class	string	设置存储类型,软链接不支持归档类型。不设置时使用当前bucket的存储类型	否
target_bucket	string	指定软链接的目标文件所在bucket，若不设置，则表示被创建的软链接与软链接目标文件在同一个bucket下。	否

示例如下：

```
symlink_key = "mySymlink"
bos_client.put_object_from_string(bucket_name, key, "This is string content.")

###### 创建软链接示例1
bos_client.put_object_symlink(bucket_name, key, symlink_key)
###### 创建软链接示例2
bos_client.put_object_symlink(bucket_name, key, symlink_key,
    forbid_overwrite=Fasle, storage_class=storage_class.STANDARD)
###### 创建软链接示例3 指定目标文件的 bucket
bos_client.put_object_from_string(bucket_name, target_key, "This is string content.")
bos_client.put_object_symlink(bucket_name, target_key, symlink_key, target_bucket=target_bucket)
```

获取软链接 接口用于获取软链接。此操作需要您对该软链接有读权限。

示例如下：

```
symlink_key = "mySymlink"

###### 获取软链接，并打印目标文件
response = bos_client.get_object_symlink(bucket_name, symlink_key)
print(response.metadata.bce_symlink_target)

###### 使用软链接获取元信息
response = bos_client.get_object_meta_data(bucket_name, symlink_key)
###### 如果是软链接类型，返回bce_object_type = “Symlink”
print(response.metadata.bce_object_type)

###### 使用软链接下载目的文件
content = bos_client.get_object_as_string(bucket_name, symlink_key)
print(content)
```

日志控制

应用场景

如果用户需要跟踪BOS的访问请求，可以开启BOS日志功能,并指定存放日志的Bucket和访问日志的文件前缀。每条访问日志记录了单个访问请求的详细信 息，包括请求者，Bucket名称，请求时间，请求操作等。日志可以应用在访问统计，安全审核等方面。

开启Bucket日志功能

如下代码开启了Bucket的日志功能，并指定最终被保存的访问日志文件的前缀：

```
bos_client.put_bucket_logging(source_bucket, target_bucket, target_prefix)
```

> **注意：**
>
> - 用户必须是源Bucket的owner且拥有FULL_CONTROL权限，且是目标Bucket的owner。
> - 源Bucket和目标Bucket必须属于同一个Region。
> - 如果源Bucket开通了Logging功能，源Bucket被删除的同时，相应的Logging信息也将被删除。
> - 如果Logging的目标Bucket被删除，则源Bucket的Logging功能会被自动关闭。
> - "targetPrefix"表示存储访问日志记录的Object名字前缀，可以为空。

日志功能相关参数的详细描述及配置注意事项请参见PutBucketLogging接口。

查询Bucket日志功能

如下代码可以查询Bucket日志功能的配置参数:

```
response = bos_client.get_bucket_logging(bucket_name)
```

get_bucket_logging方法返回的解析类中可供调用的参数说明如下:

参数	说明
status	是否开启了Logging功能
targetBucket	指定存放访问日志的Bucket，如果未开启Logging功能，响应中无该字段
targetPrefix	指定最终被保存的访问日志文件前缀，如果未开启Logging功能，响应中无该字段

关闭Bucket日志功能

如下代码可以关闭Bucket日志功能:

```
bos_client.delete_bucket_logging(bucket_name)
```

常见问题

SignatureDoesNotMatch

- 错误原因：
  - AccessKey ID和AccessKey Secret不一致
  - 版本不匹配。在v0.9.05版本之后，SDK会默认自动开启bucket virtual hosting。用户如果报错签名不对，可能是sdk版本过旧导致请求没有正确拼接成四级域名
  - 客户侧网络层修改http请求。客户出网配置可能会将请求header的host修改，导致在服务端获取host验证签名不匹配
- 解决方法：
  - 用户检查自己使用的ak sk

- 更换sdk到最新版
- 客户侧检查出网路由，打印请求发出去前的header和body字段。并联系bos值班同学查看后端收到请求的header等字段是否一致

Unable to execute HTTP request

- 错误原因：
  - 请求 bos 域名网络不通
  - 偶发网络连接抖动或者客户端环境cpu、io打满
- 解决方法：
  - ping 域名查看是否是机器网络不通导致
  - 业务侧进行重试

异常处理

系统异常

BOS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	BOS客户端产生的异常
BceInvalidArgumentError	传递参数产生的异常
BceServerError	BOS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 import bos

try:
    bos_client.delete_object(bucket_name, object_key)
except exception.BceHttpClientError as e:
    print("status code: ", e.status_code)
    print("code: ", e.code)
    print("error message: ", e.last_error)
    print("request id: ", e.request_id)
    print("BceHttpClientError: ", e)
```

返回为：

```
 status code:  404
code:  NoSuchKey
error message:  The specified key does not exist.
request id:  3c22ae79-0aad-42dd-b900-5bb53497a49d
BceHttpClientError: Unable to execute HTTP request. Retried 0 times. All trace backs:
Traceback (most recent call last):
File "/home/work/python-2.7/lib/python2.7/site-packages/baidubce/http/bce_http_client.py", line 184, in send_request
if handler_function(http_response, response):
File "/home/work/python-2.7/lib/python2.7/site-packages/baidubce/http/handler.py", line 71, in parse_error raise bse
BceServerError: The specified key does not exist.
```

客户端异常

客户端异常表示客户端尝试向BOS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；当上传文件时发生IO异常时，也会抛出BceClientError。

服务端异常

当BOS服务端出现异常时，BOS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BOS错误信息格式

参数异常

BOS Python SDK的每个调用都有一些类型固定不可以为空的参数，若该参数传入为空值则返回ValueError，若该参数传入类型错误则返回TypeError。

参数与调用的对应关系如下：

BOS调用	参数	说明	类型
create_bucket, does_bucket_exist, delete_bucket, get_bucket_location, get_bucket_lifecycle, delete_bucket_lifecycle, get_bucket_cors, delete_bucket_cors, get_bucket_logging, delete_bucket_logging, list_objects, list_all_objects, delete_object, get_bucket_acl, list_multipart_uploads, list_all_multipart_uploads	bucket_name	Bucket名称	string, unicode
put_bucket_logging	source_bucket，target_bucket	源Bucket和目标Bucket名称	string, unicode
set_bucket_acl	bucket_name	Bucket名称	string, unicode
	acl	ACL主体包含被授权人及权限	list, dict
set_bucket_canned_acl	bucket_name	Bucket名称	string, unicode
	canned_acl	CannedAcl权限	string
put_bucket_lifecycle	bucket_name	Bucket名称	string, unicode
	rules	生命周期管理规则主体，包含资源及操作动作	list, dict
put_bucket_cors	bucket_name	Bucket名称	string, unicode
	cors_configuration	跨域资源共享（CORS）的规则主体，包含允许的跨域请求的来源及允许的header等	list, dict
get_object, get_object_as_string, get_object_meta_data, initiate_multipart_upload, generate_pre_signed_url	bucket_name	Bucket名称	string, unicode
	key	Object名称	string
get_object_to_file, put_object_from_file	bucket_name	Bucket名称	string, unicode
	key	Object名称	string
	file_name	文件名	string
put_object, append_object	bucket_name	Bucket名称	string, unicode
	key	Object名称	string
	data	流对象	object
	content_length	上传Object的大小	int, long
	content_md5	上传Object的MD5	string
put_object_from_string，append_object_from_string	bucket_name	Bucket名称	string, unicode
	key	Object名称	string
	data	上传字符串	string, unicode
copy_object	source_bucket_name, target_bucket_name	源Bucket和目标Bucket名称	string, unicode
	source_key, target_key	源Object和目标Object名称	string
upload_part	bucket_name, upload_id	Bucket名称，标识MultUpload操作全局ID	string, unicode
	key	Object名称	string
	part_number	分块编号	int
	part_size	上传分块的大小	int, long
	part_fp	上传分块对象	object
upload_part_from_file	bucket_name, upload_id	Bucket名称，标识MultUpload操作全局ID	string, unicode
	key	Object名称	string
	part_number	分块编号	int
	part_size	上传分块的大小	int, long
	file_name	文件名	string
	offset	分块的开始偏移位置	int
complete_multipart_upload	bucket_name, upload_id	Bucket名称，标识MultUpload操作全局ID	string, unicode
	key	Object名称	string
	part_list	分块列表	list
abort_multipart_upload, list_parts, list_all_parts	bucket_name, upload_id	Bucket名称，标识MultUpload操作全局ID	string, unicode
	key	Object名称	string
delete_multiple_objects	bucket_name	Bucket名称	string, unicode
	key_list	object列表	list

版本变更记录

- Python SDK 开发包[2022-09-15] 版本号 0.8.77
  1. BOS PutObject、AppendObject、PostObject、UploadPart、UploadPartCopy、CopyObject、 GetObject等支持单链接限速。
  2. BOS 默认采用 bucket.region.bcebos.com 的域名访问。
- Python SDK开发包[2022-06-01] 版本号 0.8.73

  BOS归档类型文件支持三种不同的取回模式
- Python SDK开发包[2022-05-10] 版本号 0.8.71
  1. 支持用户额度管理的三个api:Put/Get/Delete Quota
  2. 支持事件通知Put/Get/Delete Nofification的三个api

3. 生成url函数支持 sts token方式 相关函数：generate_pre_signed_url()
4. 上传、下载文件支持进度百分比显示

- Python SDK开发包[2020-08-03] 版本号 0.8.62

  BOS下载文件时候，文件名不允许为空或者以"/"开头。

- Python SDK开发包[2020-08-03] 版本号 0.8.44

  新增创建、获取软链接symlink接口；增加封装三步上传接口put_super_obejct_from_file()。

- Python SDK开发包[2020-05-31] 版本号 0.8.41

  增加select_object()接口，支持选取文件进行在线分析并返回结果。

- Python SDK开发包[2019-09-12] 版本号 0.8.31

  支持操作归档存储（ARCHIVE）类型的文件，增加restroe_object()接口。

- Python SDK开发包[2019-05-23] 版本号 0.8.28

  支持在python2和python3下运行bos sdk，支持object acl的设置。

- Python SDK开发包[2017-06-27] 版本号 0.8.16

  支持header丰富，Append Object上传,stsl临时token，CORS跨域设置，生命周期管理，日志管理，三步拷贝和批量删除Object功能。

- Python SDK开发包[2017-05-25] 版本号 0.8.15

  支持冷存储。

- Python SDK开发包[2016-09-02] 版本号 0.8.13

  支持低频存储。

- Python SDK开发包[2016-07-21] 版本号 0.8.12

  支持同步Copy功能。

- Python SDK开发包[2016-03-03] 版本号 0.8.9

  支持HTTPS。

- Python SDK开发包[2015-10-13] 版本号 0.8.7

  内部优化，功能无新增。

- Python SDK开发包[2015-09-06] 版本号 0.8.6

  支持多区域选择。

- Python SDK开发包 [2015-07-23] 版本号 0.8.5 BOS本次无新功能增加。

- Python SDK开发包 [2015-07-09] 版本号 0.8.4

  1. 在访问路径中去掉v1。
  2. 修正了sdk不能操作以斜杠“/”结尾的object的bug。

- Python SDK开发包 [2015-05-28] 版本号 0.8.3

  BOS本次无新功能增加。

- Python SDK开发包 [2015-04-02] 版本号 0.8.2.1

  首次发布：

  - 支持创建、查看、罗列、删除 Bucket
  - 修改、获取Bucket的访问权限
  - 上传、查看、罗列、删除Object

PHP-SDK

概述

本文档主要介绍BOS PHP SDK的安装和使用。在使用本文档前，您需要先了解BOS的一些基本知识，并已开通了BOS服务。若您还不了解BOS，可以参考产品描述和快速入门。

安装SDK工具包

运行环境

PHP SDK 包要求运行环境至少为 PHP 5.3.2 版本，如 5.4、5.5、5.6、7.0、7.1、7.2、7.3、8.0。

> **注意：**
>
> - 0.9.19及以上版本SDK，适用于 PHP 7.3以上版本
> - 如果不需要使用 PHP8，可使用0.9.18及以下版本SDK
> - 如在运行环境为 PHP 8.0或以上环境遇到兼容性问题，请提交工单或联系技术支持，以获得PHP 8.0版本包

安装SDK

1. 在官方网站下载BOS PHP SDK工具包（ZIP包）。
2. 解压ZIP包之后有三个文件，如下：

```
 BaiduBce.phar  //PHP SDK
BosClientSample.php  //示例
SampleConf.php  //参考配置文件，具体内容见下文
```

> **说明：** 用户在配置好AK/SK和Host之后，可以使用`phpunit --debug BosClientSample.php`来运行Sample。

3. 在脚本文件中添加以下代码，即可以使用SDK包：

```
include 'BaiduBce.phar';
require 'YourConf.php';
```

关于配置文件的引用请参考下文中的配置BosClient。

SDK目录结构

```
BaiduBce.phar
├──src
│   └── BaiduBce
│       ├── Auth            //BCE签名相关
│       ├── Exception       //BCE客户端的异常
│       ├── Http            //BCE的Http通信相关
│       ├── Log             //BCE日志
│       ├── Services
│       │   └── Bos             //BOS主目录，此目录必须保留
│       │       ├── BosClient.php    //BOS操作类，所有操作可以通过BosClient类可以完成
│       │       ├── BosOptions.php   //BOS自定义配置
│       │       └── CannedAcl.php    //CannedAcl模块
│       └── Util            //BCE公用工具
└──vendor                   //第三方库
```

卸载SDK

预期卸载 SDK 时，删除脚本中include的phar包即可。

初始化

确认Endpoint

在确认您使用SDK时配置的Endpoint时，可先阅读开发人员指南中关于BOS访问域名的部分，理解Endpoint相关的概念。 百度智能云目前开放了多区域支持，请参考区域选择说明。 region与endpoint参考如下链接 https://cloud.baidu.com/doc/BOS/s/akrqd2wcx 。 目前BOS支持区域与访问域名对应信息为：

访问区域	对应Endpoint
华北-北京	bj.bcebos.com
华北-保定	bd.bcebos.com
华东-苏州	su.bcebos.com
华南-广州	gz.bcebos.com
西南-成都	cd.bcebos.com
中国香港	hkg.bcebos.com
华中-武汉	fwh.bcebos.com
华东金融-上海	fsh.bcebos.com

获取密钥

要使用百度智能云BOS，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BOS做签名验证。 可以通过如下步骤获得并了解您的AK/SK信息：

1. 注册百度智能云账号

2. 创建AK/SK

新建BosClient

BosClient是BOS服务的客户端，为开发者与BOS服务进行交互提供了一系列的方法。

在新建BosClient之前，需要先创建配置文件对BosClient进行配置，以下将此配置文件命名为`YourConf.php`，具体配置信息如下所示：

```
 // 报告所有 PHP 错误
error_reporting(-1);

define('__BOS_CLIENT_ROOT', dirname(__DIR__));

// 设置BosClient的Access Key ID、Secret Access Key和ENDPOINT
$BOS_TEST_CONFIG =
    array(
        'credentials' => array(
            'accessKeyId' => 'your ak',
            'secretAccessKey' => 'your sk',
            'sessionToken' => 'your session token'
        ),
        'endpoint' => 'http://bj.bcebos.com',
        'stsEndpoint' => 'http://sts.bj.baidubce.com',
    );

// 设置log的格式和级别
$__handler = new \Monolog\Handler\StreamHandler(STDERR, \Monolog\Logger::DEBUG);
$__handler->setFormatter(
    new \Monolog\Formatter\LineFormatter(null, null, false, true)
);
\BaiduBce\Log\LogFactory::setInstance(
    new \BaiduBce\Log\MonoLogFactory(array($__handler))
);
\BaiduBce\Log\LogFactory::setLogLevel(\Psr\Log\LogLevel::DEBUG);
```

> **注意：**
>
> 1. 在上面代码中，`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》。

2. 如果用户需要自己指定域名，可以通过传入ENDPOINT参数来指定，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://bj.bcebos.com。

3. 如果不使用STS验证方式，则stsEndpoint和sessionToken可以为空或者删除。

使用AK/SK新建BosClient

通过AK/SK方式访问BOS，用户可以参考如下代码新建一个BosClient：

```
 //使用PHP SDK，并且使用自定义配置文件
include 'BaiduBce.phar';
require 'YourConf.php';

use BaiduBce\BceClientConfigOptions;
use BaiduBce\Util\MimeTypes;
use BaiduBce\Http\HttpHeaders;
use BaiduBce\Services\Bos\BosClient;

//调用配置文件中的参数
global $BOS_TEST_CONFIG;
//新建BosClient
$client = new BosClient($BOS_TEST_CONFIG);
```

使用STS创建BosClient

申请STS token

BOS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。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BOS，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智能云STS使用介绍。

用STS token新建BOSClient

申请好STS后，可将STStoken配置到BosClient中，用户可以参考如下代码新建一个BosClient：

1. 首先进行STS的endpoint配置。STS的配置示例如下：

```
 $BOS_TEST_CONFIG =
  array(
      'credentials' => array(
          'accessKeyId' => 'your ak',
          'secretAccessKey' => 'your sk',
      ),
'stsEndpoint' => 'http://sts.bj.baidubce.com',
  );
```

2. StsClient的示例代码如下：

```
 use BaiduBce\Services\Sts\StsClient;
//新建StsClient
$client = new StsClient($BOS_TEST_CONFIG);
$aclArray =
  array(
    'id' => 'test-acl',
    'accessControlList' => array(
        array(
            'permission' => array('GET'),
            'service' => 'bce:bos',
            'region' => 'bj',
            'effect' => 'Allow',
            'resource' => array('testbucket/*')
        )
    )
  );
$request =
  array(
    'acl' => json_encode($aclArray), //用户定义的acl
    'durationSeconds' => 43200, //STS凭证有效时间
  );
$response = $client->getSessionToken($request);
$accessKeyID= $response->accessKeyId;
$secretAccessKey= $response->secretAccessKey;
$sessionToken = $response->sessionToken;
```

> **注意：**其中acl指用户定义的acl，语法请参照访问控制。

3. 将获取到的accessKeyID/secretAccessKey/sessionToken写入到配置文件YourConf.php中并新建BosClient。

```
 $BOS_TEST_CONFIG =
  array(
      'credentials' => array(
          'accessKeyId' => 'your ak',
          'secretAccessKey' => 'your sk',
          'sessionToken' => 'your session token'
      ),
      'endpoint' => 'http://bj.bcebos.com',
  );

//新建BosClient
$client = new BosClient($BOS_TEST_CONFIG);
```

注意： 目前使用STS配置client时，无论对应BOS服务的endpoint在哪里，endpoint都需配置为http://sts.bj.baidubce.com。

配置HTTPS协议访问BOS

BOS支持HTTPS传输协议，您可以通过如下两种方式在BOS PHP SDK中使用HTTPS访问BOS服务：

- 在endpoint中指定HTTPS:

```
$BOS_CONFIG =
array(
    'credentials' => array(
        'ak' => 'your-ak',
        'sk' => 'your-sk',
    ),
    'endpoint' => 'https://bj.bcebos.com',
);
$client = new BosClient($BOS_CONFIG);
```

- 通过在protocol中指定https来设置HTTPS协议:

```
$BOS_CONFIG =
array(
    'credentials' => array(
        'ak' => 'your-ak',
        'sk' => 'your-sk',
    ),
    'endpoint' => 'bj.bcebos.com',
    'protocol' => 'https',
);
$client = new BosClient($BOS_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。

```
$BOS_CONFIG =
array(
    'credentials' => array(
        'ak' => 'your-ak',
        'sk' => 'your-sk',
    ),
    'endpoint' => 'http://bj.bcebos.com',
    'protocol' => 'https',
);
$client = new BosClient($BOS_CONFIG); //将以http方式访问bos
```

使用自定义域名(CNAME)访问BOS

您可以通过如下方式在BOS PHP SDK中使用自定义域名访问BOS服务：

在endpoint中指定自定义域名，以及custom字段为true，表示使用自定义域名：

```
$BOS_CONFIG =
array(
    'credentials' => array(
        'ak' => 'your-ak',
        'sk' => 'your-sk',
    ),
    'endpoint' => 'http://custom-domain.com',
    'custom' => true,
);
$client = new BosClient($BOS_CONFIG);
$client->putObjectFromFile(null, $objectKey, $fileName);
```

> **注意：**如果您在PHP SDK中使用自定义域名新建BosClient，建议访问API接口时把bucketName字段设置成null；目前PHP SDK使用自定义域名时暂不支持copyObject和uploadPartCopy接口。

配置BosClient

设置自定义参数

PHP SDK在\BaiduBce\Bce.php中默认设置了一些基本参数，若用户想要对参数的值进行修改，可以参考此文件来创建自身的参数配置函数，并在构造BosClient的时候传入，传入代码参考如下：

```
public function CustomizedConfig() {
    $customizedConfig = array(
        BceClientConfigOptions::PROTOCOL => 'http',
        BceClientConfigOptions::REGION => 'bj',
        BceClientConfigOptions::CONNECTION_TIMEOUT_IN_MILLIS => 120 * 1000,
        BceClientConfigOptions::SOCKET_TIMEOUT_IN_MILLIS => 300 * 1000,
        BceClientConfigOptions::SEND_BUF_SIZE => 5 * 1024 * 1024,
        BceClientConfigOptions::RECV_BUF_SIZE => 5 * 1024 * 1024,
        BceClientConfigOptions::CREDENTIALS => array(
            'ak' => 'your-access-key-id',
            'sk' => 'your-secret-access-key',
        ),
        'endpoint' => 'your-endpoint',
    );

    //利用自定义配置创建BOSClient
    $customizedClient = new BosClient($customizedConfig);

    //通过自定义配置调用方法
    $options = array(BosOptions::CONFIG=>$customizedConfig);
    $this->client->listBuckets($options);
}
```

参数说明如下：

参数	说明	默认值
PROTOCOL	协议	http
REGION	区域	bj
CONNECTION_TIMEOUT_IN_MILLIS	请求超时时间（单位：毫秒）	50 * 1000
SOCKET_TIMEOUT_IN_MILLIS	通过打开的连接传输数据的超时时间（单位：毫秒）	0（指的是无限等待，若设置非0数值需要对文件大小和网速进行评估，否则上传大文件时会产生超时）
SEND_BUF_SIZE	发送缓冲区大小	1024 * 1024
RECV_BUF_SIZE	接收缓冲区大小	10 *1024* 1024

设置可选参数

BOSClient将可选的参数封装到$options中，每一个方法具有的可选参数详见具体的接口使用方法介绍，现以putObjectFromFile方法为例，参考如下代码实现设置可选参数：

```
 //利用options在通过文件上传Object的时候传入指定参数
file_put_contents($fileName, "test of put object from string");
$user_meta = array("x-bce-meta-key1" => "value1");

$options = array(
    BosOptions::CONTENT_TYPE=>"text/plain",
    BosOptions::CONTENT_MD5=>base64_encode(hash_file("md5", $fileName, true)),
    BosOptions::CONTENT_LENGTH=>filesize($fileName),
    BosOptions::CONTENT_SHA256=>hash_file("sha256", $fileName),
    BosOptions::USER_METADATA => $user_meta,
);
$client->putObjectFromFile($bucketName, $objectKey, $fileName, $options);
```

> **注意：**不要把null传入$options中，否则调用时会抛出异常。

Bucket管理

Bucket既是BOS上的命名空间，也是计费、权限控制、日志记录等高级功能的管理实体。

- Bucket名称在所有区域中具有全局唯一性，且不能修改。

> **说明：**
>
> - 百度智能云目前开放了多区域支持，请参考区域选择说明。 BOS区域与EndPoint的对应关系，请参考访问域名说明。

- 存储在BOS上的每个Object都必须包含在一个Bucket中。
- 一个用户最多可创建100个Bucket，但每个Bucket中存放的Object的数量和大小总和没有限制，用户不需要考虑数据的可扩展性。

Bucket权限管理

设置Bucket的访问权限

如下代码将Bucket的权限设置为了private：

```
 use BaiduBce\Services\Bos\CannedAcl;
$client->setBucketCannedAcl($bucket, CannedAcl::ACL_PRIVATE);
```

其中CannedAcl.php中封装了三个参数：ACL_PRIVATE、ACL_PUBLIC_READ、ACL_PUBLIC_READ_WRITE，它们分别对应的相关权限为：private、public-read、public-read-write。关于权限的具体内容可以参考《BOS API文档 使用CannedAcl方式的权限控制》。

设置指定用户对Bucket的访问权限

BOS提供setBucketAcl方法来实现指定用户对Bucket的访问权限设置，可以参考如下代码实现：

```
$my_acl = array(
    array(
        'grantee' => array(
            array(
                'id' => '7f34788d02a64a9c98f85600567d98a7',
            ),
            array(
                'id' => 'aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa',
            ),
        ),
        'permission' => array('FULL_CONTROL'),
    ),
);
$client->setBucketAcl($bucket, $my_acl);
```

> **注意：**
>
> 1. permission中的权限设置包含三个值：READ、WRITE、FULL_CONTROL，它们分别对应相关权限。具体内容可以参考《BOS API文档 上传ACL文件方式的权限控制》。
> 2. 设置两个以上（含两个）被授权人时，请参考以上示例的格式，若将数组合并会返回报错。

设置更多Bucket访问权限

1. 通过设置referer白名单方式设置防盗链

```
$my_acl = array(
    array(
        'grantee' => array(
            array(
                'id' => '7f34788d02a64a9c98f85600567d98a7',
            ),
        ),
        'permission' => array('FULL_CONTROL'),
        'condition' => array(
            'referer' => array(
                'stringLike' => array('http://www.abc.com/*'),
                'stringEquals' => array('http://www.abc.com'),
            ),
        ),
    ),
);
$client->setBucketAcl($bucket, $my_acl);
```

2. 限制客户端IP访问，只允许部分客户端IP访问

```
$my_acl = array(
    array(
        'grantee' => array(
            array(
                'id' => '7f34788d02a64a9c98f85600567d98a7',
            ),
        ),
        'permission' => array('FULL_CONTROL'),
        'condition' => array(
          'ipAddress' => array("192.168.0.0/16"),
        ),
    ),
);
$client->setBucketAcl($bucket, $my_acl);
```

设置STS临时token权限

对于通过STS方式创建的临时访问身份，管理员也可进行专门的权限设定。 STS的简介及设置临时权限的方式可参见临时授权访问。

使用BOS PHP SDK设置STS临时token权限可参考使用STS创建BosClient

查看Bucket的权限

如下代码可以查看Bucket的权限：

```
$response = $client->getBucketAcl($bucket);
```

getBucketAcl方法返回的解析类中可供调用的参数有：

参数	说明
owner	Bucket owner信息
id	Bucket owner的用户ID
acl	标识Bucket的权限列表
grantee	标识被授权人
-id	被授权人ID
permission	标识被授权人的权限

查看Bucket所属的区域

Bucket Location即Bucket Region，百度智能云支持的各region详细信息可参见区域选择说明。

如下代码可以获取该Bucket的Location信息：

```
$client->getBucketLocation($bucketName);
```

新建Bucket

如下代码可以新建一个Bucket：

```
$bucketName = "your_bucket";

//Bucket是否存在，若不存在创建Bucket
$exist = $client->doesBucketExist($bucketName);
if(!$exist){
    $client->createBucket($bucketName);
}
```

> **注意：** 由于Bucket的名称在所有区域中是唯一的，所以需要保证bucketName不与其他所有区域上的Bucket名称相同。
>
> Bucket的命名有以下规范：
>
> - 只能包括小写字母，数字，短横线（-）。
> - 必须以小写字母或者数字开头。
> - 长度必须在4-63字节之间。

通过上述代码创建的bucket，权限是私有读写，存储类型是标准类型（Standard）。用户在控制台创建Bucket时可以指定Bucket权限和存储类型。

列举Bucket

如下代码可以列出用户所有的Bucket：

```
 $response = $client->listBuckets();
foreach ($response->buckets as $bucket) {
    print $bucket->name;
}
```

删除Bucket

删除指定Bucket

如下代码可以删除一个Bucket：

```
 $bucketName = "your_bucket";
$client->deleteBucket($bucketName);
```

> **注意：**
>
> - 在删除前需要保证此Bucket下的所有Object和未完成的三步上传Part已经被删除，否则会删除失败。
> - 在删除前确认该Bucket没有开通跨区域复制，不是跨区域复制规则中的源Bucket或目标Bucket，否则不能删除。

删除所有Bucket

将deleteBucket和listBuckets函数结合，可以删除全部Bucket，参考代码如下：

```
  //列出全部Bucket
$response = $client->listBuckets();

//遍历删除全部Bucket
foreach ($response->buckets as $bucket) {
    $marker = null;
    while (true) {
        $options = array();
        if ($marker !== null) {
            $options[BosOptions::MARKER] = $marker;
        }
        $response = $client->listObjects($bucket->name, $options);
        foreach ($response->contents as $object) {
            $client->deleteObject($bucket->name, $object->key);
        }
        if ($response->isTruncated) {
            $marker = $response->nextMarker;
        } else {
            break;
        }
    }
    $client->deleteBucket($bucket->name);
}
```

判断Bucket是否存在

若用户需要判断某个Bucket是否存在，则如下代码可以做到：

```
$client->doesBucketExist($bucketName);
```

Bucket数据同步

Bucket数据同步实现了不同Bucket内数据的自动同步，并且支持通过设置具体规则来定制同步方式。其中同步的数据分为两种，存量数据和增量数据。以下为使用方法：

创建数据同步

```
  $replication_rule = array(
        'status' => 'enabled',
        'replicateDeletes' => 'enabled',
        'id' => 'sample'
        );
$replication_rule['resource'][0] = $raw_bucket . "/*";
$replication_rule['destination']['bucket'] = $target_bucket;
$replication_rule['replicateHistory']['bucket'] = $target_bucket;
$bj_client->putBucketReplication($raw_bucket, $replication_rule);
```

获取数据同步的Bucket信息

```
$response = $bj_client->getBucketReplication($raw_bucket);
```

删除数据同步复制配置

```
$response = $bj_client->deleteBucketReplication($raw_bucket);
```

获取Bucket数据同步的进程状态

```
$response = $bj_client->getBucketReplicationProgress($raw_bucket);
```

Bucket生命周期管理

一个数据是有其生命周期的，从创建到归档到删除可以认为是一个完整的循环。创建之初的数据往往需要频繁访问读取，之后迅速冷却归档，最终被删除。生命周期管理就是对象存储服务帮助用户自动化管理数据的生命周期。 通常可以服务于以下场景：

1. 数据达到一定寿命后自动归档或删除。

2. 指定时间执行操作。

新建Lifecycle配置 如下代码新建了一个Lifecycle配置

```
$lifecycle_rule = array(
  array(
    'id' => 'rule-id0',
    'status' => 'enabled',
    'resource' => array(
      $this->bucket.'/prefix/*',
    ),
    'condition' => array(
      'time' => array(
        'dateGreaterThan' => '2016-09-07T00:00:00Z',
      ),
    ),
    'action' => array(
      'name' => 'DeleteObject',
    )
  ),
  array(
    'id' => 'rule-id1',
    'status' => 'disabled',
    'resource' => array(
      $this->bucket.'/prefix/*',
    ),
    'condition' => array(
      'time' => array(
        'dateGreaterThan' => '2016-09-07T00:00:00Z',
      ),
    ),
    'action' => array(
      'name' => 'Transition',
      'storageClass' => 'COLD',
    ),
  ),
);
// 设置生命周期规则
$client->putBucketLifecycle($bucket, $lifecycle_rule);
```

> **注意：**
>
> 1. 只有Bucket的Owner拥有full control才能够进行此操作。
> 2. "resource"指明规则对哪些资源生效。例如，对samplebucket里以`prefix/`为前缀的Object生效：`samplebucket/prefix/*`；对samplebucket里所有Object生效：`samplebucket/*`。

生命周期管理功能相关参数的详细解释及配置注意事项请参见：PutBucketLifecycle接口

读取Bucket的Lifecycle配置

如下代码可读取Bucket的Lifecycle配置

```
$response = $client->getBucketLifecycle($bucket);
```

删除Bucket Lifecycle

如下代码可以删除Bucket的lifecycle:

```
$client->deleteBucketLifecycle($bucket);
```

Bucket跨域资源访问

跨域资源共享(CORS)允许WEB端的应用程序访问不属于本域的资源。BOS提供接口方便开发者控制跨域访问的各种权限。

设定CORS规则 如下代码设定了一个CORS规则：

```
$cors_rule = array(
  array(
      'allowedOrigins' => array(
          'http://www.example.com',
          'www.example2.com'
      ),
      'allowedMethods' => array(
          'GET',
          'HEAD'
      ),
      'allowedHeaders' => array(
          'Authorization'
      ),
      'allowedExposeHeaders' => array(
          'user-custom-expose-header'
      ),
      'maxAgeSeconds' => 3600
  ),
  array(
      'allowedOrigins' => array(
          'http://www.example3.com'
      ),
      'allowedMethods' => array(
          'GET',
          'PUT'
      ),
      'allowedHeaders' => array(
          'x-bce-test'
      ),
      'allowedExposeHeaders' => array(
          'user-custom-expose-header'
      ),
      'maxAgeSeconds' => 3600
  )
);
$client->putBucketCors($bucket, $cors_rule);
```

> **注意：**
>
> 1. 如果原规则存在则覆盖原规则。
> 2. 只有Bucket的所有者和被授予FULL_CONTROL权限的用户才能设置Bucket的CORS。没有权限时，返回403 Forbidden错误，错误码：AccessDenied。

CORS规则相关参数的详细解释请参见PutBucketCors接口

获取Bucket的CORS规则 如下代码可获取Bucket的CORS配置:

```
$response = $client->getBucketCors($bucket);
```

关闭Bucket的CORS功能并清空所有规则 如下代码可以关闭Bucket的CORS功能并清空所有规则:

```
$client->deleteBucketCors($bucket);
```

Bucket服务端加密

BOS支持用户在上传和复制请求（PutObject、PostObject、InitiateMultipartUpload、AppendObject、FetchObject、CopyObject）中携带x-bce-server-side-encryption的HTTP header并指定用户的加密算法（目前仅支持AES256），从而实现数据的有效安全保护。

开启Bucket的服务端加密 如下代码可以开启bucket服务端加密功能

```
$client->putBucketEncryption($bucket, 'AES256');
```

获取Bucket服务端加密信息 如下代码可以获取bucket服务端加密相关信息，包括加密算法等

```
$response = $client->getBucketEncryption($bucket);
```

关闭Bucket的服务端加密 如下代码可以关闭bucket服务端加密功能

```
$client->deleteBucketEncryption($bucket);
```

Bucket访问日志规则

当用户需要跟踪访问BOS的请求时可以对Bucket开启访问日志功能。每条访问日志记录了单个访问请求的详细信息，包括请求者、Bucket名称、请求时间和请求操作等。日志记录功能可以应用在访问统计和安全审核等方面。当一个Bucket开启访问日志功能后，会自动将对这个Bucket的访问请求，以小时为单位，按照固定的命名规则，生成日志文件写入用户指定的Bucket。

开启Bucket的访问日志 如下代码可以开启Bucket的访问日志并指定存放日志的Bucket和访问日志的文件前缀

```
$logging = array(
      'targetBucket' => $bucket.'logging',
      'targetPrefix' => 'TargetPrefixName'
);
$client->putBucketLogging($bucket, $logging);
```

获取Bucket的访问日志配置情况 如下代码可以获取某个Bucket的访问日志配置情况

```
$response = $client->getBucketLogging($bucket);
```

关闭Bucket访问日志记录功能

```
$client->deleteBucketLogging($bucket);
```

Bucket跨区域复制规则

设置跨区域复制规则 如下代码给bj_bucket设置跨区域复制规则，目的bucket是gz_bucket

```
$replication_rule = array(
        'status' => 'enabled',
        'replicateDeletes' => 'enabled',
        'id' => 'sample'
    );
$replication_rule['resource'][0] = $bj_bucket . "/*";
$replication_rule['destination']['bucket'] = $gz_bucket;
$replication_rule['replicateHistory']['bucket'] = $gz_bucket;
$bj_client->putBucketReplication($bj_bucket, $replication_rule);
```

获取跨区域复制规则 如下代码用来获取数据同步的Bucket信息，包括源Bucket名称、目的Bucket名称、存储类型、是否进行历史复制，数据同步策略等

```
$response = $bj_client->getBucketReplication($bj_bucket);
```

获取跨区域复制进度 如下代码可以用来获取数据同步复制的进程状态

```
$response = $bj_client->getBucketReplicationProgress($bj_bucket);
```

删除跨区域复制规则 如下代码可以删除数据同步复制配置

```
$bj_client->deleteBucketReplication($bj_bucket);
```

Bucket托管静态网站

BOS支持用户将静态网站托管在Bucket上，实现网站轻量化运维。设置生效后，用户可通过直接访问该Bucket的域名访问该托管网站。

设置bucket静态网站托管规则 用户必须对bucket 具有 full control 权限。不建议设置归档文件，归档文件没有取回时不可读，StaticWebsite此时不会生效。

```
$static_website = array(
    'index' => 'index.html',
    'notFound' => '404.html'
);
$client->putBucketStaticWebsite($bucket, $static_website);
```

获取bucket静态网站托管规则

```
$response = $client->getBucketStaticWebsite($bucket);
```

删除bucket静态网站托管规则

```
$client->deleteBucketStaticWebsite($bucket);
```

Bucket回收站规则

为了提升BOS的数据可靠性，BOS提供了回收站功能，Bucket的Owner且对Bucket具备Full Control权限的用户可对Bucket进行回收站相关的配置。

开通Bucket Trash功能 开通Bucket Trash功能，用户必须是源Bucket的owner且拥有FULL_CONTROL权限，且是目标Bucket的owner。

```
$client->putBucketTrash($bucket, '.trashDirName');
```

> **备注：** 若已开通trash功能，会覆盖原目录名。 归档类型文件不支持Bucket Trash，删除归档类型文件时会直接删除。

获取Bucket Trash开通状态 获取Bucket Trash开通状态，返回当前trash目录名，默认为.trash。用户必须是源Bucket的owner且拥有FULL_CONTROL权限，且是目标Bucket的owner。

```
$response = $client->getBucketTrash($bucket);
```

关闭Bucket Trash功能 关闭trash功能，用户必须是源Bucket的owner且拥有FULL_CONTROL权限，且是目标Bucket的owner。

```
$client->deleteBucketTrash($bucket);
```

Bucket原图保护功能

BOS支持通过API实现原图保护功能，并可通过resource字段指定该功能生效范围。 对于开通原图保护的文件，不允许匿名下载访问该文件，或带自定义图片处理参数访问，只允许使用style样式访问或携带合法签名访问。

开启Bucket的原图保护功能 如下代码可以开启Bucket的原图保护功能，并指定resource字段，表示生效的资源范围。 对于开通原图保护的文件，不允许匿名下载访问该文件，或带自定义图片处理参数访问，只允许使用style样式访问或携带合法签名访问。

```
$copyright_protection = array(
    $bucket.'/prefix/*',
    $bucket.'/*/suffix'
);
$client->putBucketCopyrightProtection($bucket, $copyright_protection);
```

获取Bucket的原图保护配置情况

```
$response = $client->getBucketCopyrightProtection($bucket);
```

关闭Bucket原图保护功能

```
$client->deleteBucketCopyrightProtection($bucket);
```

设置Bucket存储类型

设置bucket存储类型

```
// STANDARD代表标准存储，STANDARD_IA代表低频存储，COLD代表冷存储，ARCHIVE代表归档类型；
// 对于多AZ类型bucket，只允许设置MAZ_STANDARD和MAZ_STANDARD_IA两种存储类型
$storageClass = "STANDARD";
$client->putBucketStorageClass($bucketName, $storageClass);
```

获取bucket的存储类型

```
$client->getBucketStorageClass($bucketName)
```

文件管理

上传文件

在BOS中，用户操作的基本数据单元是Object。Object包含Key、Meta和Data。其中，Key是Object的名字；Meta是用户对该Object的描述，由一系列Name-Value对组成；Data是Object的数据。

BOS PHP SDK提供了丰富的文件上传接口，可以通过以下方式上传文件：

- 简单上传
- 追加上传
- 分片上传
- 大文件上传
- 断点续传上传

简单上传

BOS在简单上传的场景中，支持以指定文件形式、以数据流方式、以二进制串方式、以字符串方式执行Object上传，请参考如下代码：

```
// 以数据流形式上传Object
$string = "myobjectcontent";
$length = Strlen($string);
$md5 = base64_encode(md5($string,true));
$data = file_get_contents("myfile","rb");
$client->putObject($bucketName, $objectKey, $data, $length, $md5);

// 从字符串中上传Object
$client->putObjectFromString($bucketName, $objectKey, $string);

// 从文件中直接上传Object
$client->putObjectFromFile($bucketName, $objectKey, $fileName);
```

Object以文件的形式上传到BOS中，putObject相关接口支持不超过5GB的Object上传。在PutObject请求处理成功后，BOS会在Header中返回Object的ETag作为文件标识。

设置文件元信息

文件元信息(Object Meta)，是对用户在向BOS上传文件时，同时对文件进行的属性描述，主要分为分为两种：设置HTTP标准属性（HTTP Headers）和用户自定义的元信息。

设定Object的Http Header

BOS PHP SDK本质上是调用后台的HTTP接口，因此用户可以在上传文件时自定义Object的Http Header。常用的http header说明如下：

名称	描述	默认值
Content-MD5	文件数据校验，设置后BOS会启用文件内容MD5校验，把您提供的MD5与文件的MD5比较，不一致会抛出错误	无
Content-Type	文件的MIME，定义文件的类型及网页编码，决定浏览器将以什么形式、什么编码读取文件。如没有指定，BOS则根据文件的扩展名自动生成，如文件没有扩展名则填默认值	application/octet-stream
Content-Disposition	指示回复的内容该以何种形式展示，是以内联的形式（即网页或者页面的一部分），还是以包含文件名的附件的形式下载并保存到本地	无
Content-Length	上传的文件的长度，超过流/文件的长度会截断，不足为实际值	流/文件时间长度
Expires	缓存过期时间	无
Cache-Control	指定该Object被下载时的网页的缓存行为	指定该Object被下载时网页的缓存行为。取值如下： no-cache：不可直接使用缓存，而是先到服务端验证Object是否已更新。如果Object已更新，表明缓存已过期，需从服务端重新下载Object；如果Object未更新，表明缓存未过期，此时将使用本地缓存。 no-store：所有内容都不会被缓存。 public：所有内容都将被缓存。 private：所有内容只在客户端缓存。 max-age=：缓存内容的相对过期时间，单位为秒。此选项仅在HTTP 1.1中可用。 默认值：无

参考代码如下：

```
 $options = array(
      BosOptions::CONTENT_TYPE=>"text/plain",
      BosOptions::CONTENT_MD5=>"md5",
      BosOptions::CONTENT_LENGTH=>10,
      BosOptions::CACHE_CONTROL=>" max-age=86400",
      );

$client->putObject($bucketName, $objectKey, $data, $options);
```

用户自定义元信息

BOS支持用户自定义元数据来对Object进行描述。如下代码所示：

```
 $user_meta = array("x-bce-meta-key1" => "value1");
$client->putObject($bucketName,
   $objectKey,
   $data,
   array(
      'userMetadata' => $user_meta,
));
```

> **提示：**
>
> - 在上面代码中，用户自定义了一个名字为"x-bce-meta-key1"，值为"value1"的元数据
> - 当用户下载此Object的时候，此元数据也可以一并得到
> - 一个Object可以有多个类似的参数，但所有的User Meta总大小不能超过2KB

上传Object时设置存储类型

BOS支持标准存储, 低频存储和冷存储，上传Object并存储为低频存储类型通过指定StorageClass实现，三种存储类型对应的参数如下：

存储类型	参数
标准存储	STANDARD
低频存储	STANDARD_IA
冷存储	COLD

以低频存储为例，代码如下：

```
 use BaiduBce\Services\Bos\StorageClass;

// 设置object的storage class为`StorageClass::STANDARD_IA`后，将上传一个低频object（默认为标准object）
$options = array(
   BosOptions::STORAGE_CLASS => StorageClass::STANDARD_IA,
);
$response = $bos_client->putObjectFromString($bucket, $object, "test", $options);
```

putObject请求处理成功后，BOS会在Header中返回Object的Content-MD5，用户可以根据这个参数来对文件进行校验。

追加上传

上文介绍的简单上传方式，创建的Object都是Normal类型，用户不可再进行追加写，这在日志、视频监控、视频直播等数据复写较频繁的场景中使用不方便。

正因如此，百度智能云BOS特别支持了AppendObject，即以追加写的方式上传文件。通过AppendObject操作创建的Object类型为Appendable Object，可以对该Object追加数据。AppendObject大小限制为0~5G。

通过AppendObject方式上传示例代码如下：

```
 // 从文件上传appendable object
file_put_contents($filename, "test of put append object");
$response = $client->appendObjectFromFile($bucket, $appendKey, $filename, 0);

// 下次追加写的位置
$nextOffset = $response->metadata[BosOptions::NEXT_APPEND_OFFSET];

// 从字符串上传appendable object
$response = $client->appendObjectFromString($bucket, $appendKey, "appendStr", intval($nextOffset));
```

分块上传

除了通过简单上传几追加上传方式将文上传件到BOS以外，BOS还提供了另外一种上传模式 —— Multipart Upload。用户可以在如下的应用场景内（但不仅限于此），使用Multipart Upload上传模式，如：

- 需要支持断点上传。
- 上传超过5GB大小的文件。
- 网络条件较差，和BOS的服务器之间的连接经常断开。
- 需要流式地上传文件。
- 上传文件之前，无法确定上传文件的大小。

下面将一步步介绍Multipart Upload的实现。假设有一个文件，本地路径为 `/path/to/file.zip` ，由于文件比较大，将其分块传输到BOS中。

初始化Multipart Upload

使用 `initiateMultipartUpload` 方法来初始化一个分块上传事件：

```
 $response = $client->initiateMultipartUpload($bucketName, $objectKey);
$uploadId =$response->uploadId;
```

`initiateMultipartUpload` 的返回结果中含有 `UploadId` ，它是区分分块上传事件的唯一标识，在后面的操作中，我们将用到它。

上传低频存储类型Object的初始化

初始化低频存储的一个分块上传事件：

```
use BaiduBce\Services\Bos\StorageClass;

$options = array(
  BosOptions::STORAGE_CLASS => StorageClass::STANDARD_IA,
);
$bos_client->initiateMultipartUpload($bucketName, $objectName, $options);
```

上传冷存储类型Object的初始化

初始化冷存储的一个分块上传事件：

```
use BaiduBce\Services\Bos\StorageClass;

$options = array(
   BosOptions::STORAGE_CLASS => StorageClass::COLD,
);
$bos_client->initiateMultipartUpload($bucketName, $objectName,$options);
```

上传分块

接着，把文件分块上传。

```
// 设置本地需分块上传的文件
$fileName = "myfilename";
$fileSize = filesize($fileName);
//设置分块的开始偏移位置
$offset = 0;
$partNumber = 1;
//设置每块为5MB
$partSize = 5 * 1024 * 1024;
$length = $partSize;
$partList = array();
$bytesLeft = $fileSize;

//分块上传
while ($bytesLeft > 0) {
   $length = ($length > $bytesLeft) ? $bytesLeft : $length;
   $response = $client->uploadPartFromFile($bucketName, $objectKey, $uploadId,  $partNumber, $fileName, $offset, $length);
   array_push($partList, array("partNumber"=>$partNumber, "eTag"=>$response->metadata["etag"]));
   $offset += $length;
   $partNumber++;
   $bytesLeft -= $length;
}
```

上面代码的核心是调用 `UploadPart` 方法来上传每一个分块，但是要注意以下几点：

- UploadPart 方法要求除最后一个Part以外，其他的Part大小都要大于等于5MB。但是Upload Part接口并不会立即校验上传Part的大小；只有当Complete Multipart Upload的时候才会校验。
- 为了保证数据在网络传输过程中不出现错误，建议您在`UploadPart`后，使用每个分块BOS返回的Content-MD5值分别验证已上传分块数据的正确性。当所有分块数据合成一个Object后，不再含MD5值。
- Part号码的范围是1~10000。如果超出这个范围，BOS将返回InvalidArgument的错误码。
- 每次上传Part时都要把流定位到此次上传块开头所对应的位置。
- 每次上传Part之后，BOS的返回结果会包含`eTag`和`partNumber`，需要保存到$partList中。$partList类型是list里面每个元素是个dict，每个dict包含两个关键字，一个是partNumber，一个是eTag；在后续完成分块上传的步骤中会用到它。

完成分块上传

如下代码所示，完成分块上传：

```
$response = $client->completeMultipartUpload($bucketName, $objectKey, $uploadId, $partList);
print $response->location;
```

上面代码中的 `partList` 是第二步中保存的part列表，BOS收到用户提交的Part列表后，会逐一验证每个数据Part的有效性。当所有的数据Part验证通过后，BOS将把这些数据part组合成一个完整的Object。

取消分块上传

用户可以使用`abortMultipartUpload`方法取消分块上传。

```
$client->abortMultipartUpload($bucketName, $objectKey, $uploadId);
```

获取未完成的分块上传事件

用户可以使用`listMultipartUploads`方法获取Bucket中未完成的分块上传事件。

```
$response = $client->listMultipartUploads($bucketName);
print $response->bucket;
print $response->uploads[0]->key;
```

> **注意：**
>
> 1. 默认情况下，如果Bucket中的分块上传事件的数目大于1000，则只会返回1000个Object，并且返回结果中IsTruncated的值为True，同时返回NextKeyMarker作为下次读取的起点。
> 2. 若想返回更多分块上传事件的数目，可以使用KeyMarker参数分次读取。

大文件上传 BOS PHP SDK 提供针对大文件的分块上传封装接口`putSuperObjectFromFile`，接口内部会对文件进行分块上传，默认分块大小为5MB，可以指定PART_SIZE。

```
$filename = "/home/file.zip";
$userMeta = array("private" => "private data");
$options = array(
    BosOptions::USER_METADATA => $userMeta
);
$client->putSuperObjectFromFile($bucket, $object, $filename, $options);
```

获取所有已上传的块信息

用户可以使用listParts方法获取某个上传事件中所有已上传的块：

```
$response = $client->listParts($bucketName, $objectKey, $uploadId);
print $response->bucket;
print $response->uploads[0]->key;
```

> **注意：**
>
> 1. 默认情况下，如果Bucket中的分块上传事件的数目大于1000，则只会返回1000个Object，并且返回结果中IsTruncated的值为True，同时返回NextPartNumberMarker作为下次读取的起点。
> 2. 若想返回更多分块上传事件的数目，可以使用PartNumberMarker参数分次读取。

断点续传上传

当用户向BOS上传大文件时，如果网络不稳定或者遇到程序崩等情况，则整个上传就失败了，失败前已经上传的部分也作废，用户不得不重头再来。这样做不仅浪费资源，在网络不稳定的情况下，往往重试多次还是无法完成上传。 基于上述场景，BOS提供了断点续传上传的能力:

- 当网络情况一般的情况下，建议使用三步上传方式，将object分为5Mb的块，参考分块上传。
- 当您的网络情况非常差，推荐使用appendObject的方式进行断点续传，每次append 较小数据256kb，参考追加上传。

> 提示
>
> - 断点续传是分片上传的封装和加强，是用分片上传实现的；
> - 文件较大或网络环境较差时，推荐使用分片上传；

下载文件

BOS PHP SDK提供了丰富的文件下载接口，用户可以通过以下方式从BOS中下载文件：

- 简单流式下载
- 下载到本地文件
- 断点续传下载
- 范围下载

简单流式下载

用户可以通过如下代码将Object读取到一个流中：

```
$client->getObjectAsString($bucketName, $objectKey);
```

直接下载Object到文件

用户可以参考如下代码将Object下载到指定文件：

```
$client->getObjectToFile($bucketName, $objectKey, $fileName);
```

范围下载

为了实现更多的功能，可以通过使用options配置RANGE参数来指定下载范围，实现更精细化地获取Object。如果指定的下载范围是0 - 100，则返回第0到第100个字节的数据，包括第100个，共101字节的数据，即[0, 100]。RANGE参数的格式为array($offset, $endset)其中两个变量为长整型，单位为字节。用户也可以用此功能实现文件的分段下载和断点续传。

```
$options = array(
    BosOptions::RANGE=>array(0,100),
);
$client->getObjectToFile($bucketName, $objectKey, $fileName, $options);
```

其他使用方法

获取Object的存储类型

Object的storage class属性分为STANDARD(标准存储), STANDARD_IA(低频存储)和COLD(冷存储)，通过如下代码可以实现：

```
$response = $client->getObjectMetadata($bucketName, $objectKey);
echo $response[BosOptions::STORAGE_CLASS];
```

只获取ObjectMetadata

用户也可通过 getObjectMetadata 方法可以只获取ObjectMetadata而不获取Object的实体。如下代码所示：

```
$response = $client->getObjectMetadata($bucketName, $objectKey);
echo $response[BosOptions::ETAG];
```

getObjectMetadata方法返回的解析类中可供调用的参数有：

参数	说明
contentType	Object的类型
contentLength	Object的大小
contentMd5	Object的MD5
etag	Object的HTTP协议实体标签
storageClass	Object的存储类型
userMetadata	如果在PutObject指定了userMetadata自定义meta，则返回此项

变更文件存储等级

上文中已提到，BOS支持为文件赋予STANDARD(标准存储), STANDARD_IA(低频存储)和COLD(冷存储)三种存储类型。同时，BOS PHP SDK也支持用户对特定文件执行存储类型变更的操作。

涉及到的参数如下：

参数	说明
x-bce-storage-class	指定Object的存储类型，STANDARD_IA代表低频存储，COLD代表冷存储，不指定时默认是标准存储类型。

示例如下：

```
 use BaiduBce\Services\Bos\StorageClass;

$options = array(
    BosOptions::STORAGE_CLASS => StorageClass::STANDARD_IA,
);

// 标准存储转低频存储
$client->copyObject($bucket, $key, $bucket, $key, $options);
$response = $client->getObjectMetadata($bucket, $key);
echo $response[BosOptions::STORAGE_CLASS];

$options = array(
    BosOptions::STORAGE_CLASS => StorageClass::COLD,
);

// 低频存储转冷存储
$client->copyObject($bucket, $key, $bucket, $key, $options);
$response = $client->getObjectMetadata($bucket, $key);
echo $response[BosOptions::STORAGE_CLASS];
```

获取文件下载URL

用户可以参考如下代码获取Object的URL：

```
 use BaiduBce\Services\Bos\BosClient;
use BaiduBce\Services\Bos\BosOptions;
use BaiduBce\Auth\signOptions;

$client = new BoSClient($BOS_CONFIG);

$signOptions = array(
    SignOptions::TIMESTAMP=>new \DateTime(),
    SignOptions::EXPIRATION_IN_SECONDS=>300,
);
$url = $client->generatePreSignedUrl($bucket,
   $key,
   array(BosOptions::SIGN_OPTIONS => $signOptions)
    );
```

说明：

- 用户在调用该函数前，需要手动设置endpoint为所属区域域名。百度智能云目前开放了多区域支持，请参考区域选择说明。目前支持“华北-北京”、“华南-广州”和“华东-苏州”三个区域。北京区域：http://bj.bcebos.com，广州区域：http://gz.bcebos.com，苏州区域：http://su.bcebos.com。
- EXPIRATION_IN_SECONDS为指定的URL有效时长，时间从当前时间算起，为可选参数，不配置时系统默认值为1800秒。如果要设置为永久不失效的时间，可以将expirationInSeconds参数设置为 -1，不可设置为其他负数。
- TIMESTAMP为可选参数，不配置时，系统默认TIMESTAMP为当前时间。
- 如果预期获取的文件时公共可读的，则对应URL链接可通过简单规则快速拼接获取: http://$bucket.$region.bcebos.com/$object或者http://$region.bcebos.com/$bucket/$object。

现已支持通过STS方式获取，配置STS访问BOS见初始化页面，代码示例如下：

```
 $signOptions = array(
    SignOptions::TIMESTAMP=>new \DateTime(),
    SignOptions::EXPIRATION_IN_SECONDS=>1800,

);
$url = $client->generatePreSignedUrl($bucket,
   $key,
   array(BosOptions::SIGN_OPTIONS => $signOptions, BosOptions::PARAMS => array('x-bce-security-token' => $sessionToken))
    );
echo $url;
```

列举存储空间中的文件

BOS SDK支持用户通过以下两种方式列举出object：

- 简单列举
- 通过参数复杂列举

除此之外，用户还可在列出文件的同时模拟文件夹

简单列举

当用户希望简单快速列举出所需的文件时，可通过listObjects方法获取Bucket中的Object列表。

```
$client->listObjects($bucketName);
```

> **注意：**
>
> 1. 默认情况下，如果Bucket中的Object数量大于1000，则只会返回1000个Object。
> 2. 若想增大返回Object的数目，可以使用Marker参数分次读取。

通过参数复杂列举

除上述简单列举外，用户还可通过options配置可选参数来实现各种灵活的查询功能。可设置的参数如下：

参数	功能	
PREFIX	限定返回的object key必须以prefix作为前缀	setPrefix(String prefix)
DELIMITER	是一个用于对Object名字进行分组的字符所有名字包含指定的前缀且第一次出现。Delimiter字符之间的Object作为一组元素: CommonPrefixes	
MARKER	设定结果从marker之后按字母排序的第一个开始返回	
MAX_KEYS	限定此次返回object的最大数，最大为1000，默认值是1000。如果指定的值大于1000，按1000操作	

> **注意：**
>
> 1. 如果有Object以Prefix命名，当仅使用Prefix查询时，返回的所有Key中仍会包含以Prefix命名的Object，详见递归列出目录下所有文件。
> 2. 如果有Object以Prefix命名，当使用Prefix和Delimiter组合查询时，返回的所有Key中会有Null，Key的名字不包含Prefix前缀，详见查看目录下的文件和子目录。

下面我们分别以几个案例说明通过参数列举的方法：

指定最大返回条数

```
 // 指定最大返回条数为500
$options = array(
        BosOptions::MAX_KEYS=>500,
);

$response = $client->listObjects($bucket, $options);
```

返回指定前缀的object

```
 // 指定返回前缀为usr的object
$options = array(
        BosOptions::PREFIX=>"usr",
);

$response = $client->listObjects($bucket, $options);
```

从指定Object后返回

```
 // 用户可以定义不包括某object，从其之后开始返回
$options = array(
        BosOptions::MARKER=>"object",
);

$response = $client->listObjects($bucket, $options);
```

分页获取所有Object

用户可设置每页最多500条记录

```
 $options = array(
        BosOptions::MAX_KEYS => 500,
);

$isTruncated = true;
while($isTruncated) {
    $res = $client->listObjects($bucket, $options);
    $isTruncated = $res->isTruncated;
    if (property_exists($res, 'nextMarker')) {
        $options[BosOptions::MARKER] = $res->nextMarker;
    }
}
```

分页获取所有特定Object后的结果

用户可设置每页最多500条记录，并从某特定object之后开始获取

```
$options = array(
    BosOptions::MAX_KEYS => 500,
    BosOptions::MARKER => "object",
);

$isTruncated = true;
while($isTruncated) {
    $res = $client->listObjects($bucket, $options);
    $isTruncated = $res->isTruncated;
    if (property_exists($res, 'nextMarker')) {
        $options[BosOptions::MARKER] = $res->nextMarker;
    }
}
```

listObjects方法返回的解析类中可供调用的参数有：

参数	说明
name	Bucket名称
prefix	匹配以prefix开始到第一次出现Delimiter字符之间的object作为一组元素返回
marker	本次查询的起点
maxKeys	请求返回的最大数目
isTruncated	指明是否所有查询都返回了；false-本次已经返回所有结果，true-本次还没有返回所有结果
contents	返回的一个Object的容器
+key	Object名称
+lastModified	此Object最后一次被修改的时间
+eTag	Object的HTTP协议实体标签
+storageClass	Object的存储形态
+size	Object的内容的大小（字节数）
+owner	Object对应Bucket所属用户信息
++id	Bucket Owner的用户ID
++displayName	Bucket Owner的名称

模拟文件夹功能

在BOS的存储结果中是没有文件夹这个概念的，所有元素都是以Object来存储，但BOS的用户在使用数据时往往需要以文件夹来管理文件。

因此，BOS提供了创建模拟文件夹的能力，其本质上来说是创建了一个size为0的Object。对于这个Object可以上传下载，只是控制台会对以"/"结尾的Object以文件夹的方式展示。

用户可以通过 Delimiter 和 Prefix 参数的配合模拟出文件夹功能。Delimiter 和 Prefix 的组合效果是这样的：

如果把 Prefix 设为某个文件夹名，就可以罗列以此 Prefix 开头的文件，即该文件夹下递归的所有的文件和子文件夹（目录）。文件名在Contents中显示。

如果再把 Delimiter 设置为 "/" 时，返回值就只罗列该文件夹下的文件和子文件夹（目录），该文件夹下的子文件名（目录）返回在 CommonPrefixes 部分，子文件夹下递归的文件和文件夹不被显示。

如下是几个应用方式：

列出Bucket内所有文件

当用户需要获取Bucket下的所有文件时，可以参考分页获取所有Object

递归列出目录下所有文件

可以通过设置 Prefix 参数来获取dir目录下所有的文件：

```
$options = array(
    BosOptions::PREFIX => "dir/",
);

$isTruncated = true;
while($isTruncated) {
    $res = $client->listObjects($bucket, $options);
    $isTruncated = $res->isTruncated;
    if (property_exists($res, 'nextMarker')) {
        $list_options[BosOptions::MARKER] = $res->nextMarker;
    }
}
```

查看目录下的文件和子目录

在 Prefix 和 Delimiter 结合的情况下，可以列出dir目录下的文件和子目录：

```
$options = array(
    BosOptions::PREFIX => "dir/",
    BosOptions::DELIMITER => "/",
);

$isTruncated = true;
while($isTruncated) {
    $res = $client->listObjects($bucket, $options);
    $isTruncated = $res->isTruncated;
    if (property_exists($res, 'nextMarker')) {
        $list_options[BosOptions::MARKER] = $res->nextMarker;
    }
}
```

列举Bucket中object的存储属性

当用户完成上传后，如果需要查看指定Bucket中的全部Object的storage class属性，可以通过如下代码实现：

```
use BaiduBce\Services\Bos\StorageClass;

$response = $bos_client->listObjects($bucketName);
$objectList = $response->contents;
for ($i= 0;$i< count($objectList); $i++){
    echo $objectList[$i]->storageClass;
}
```

取回object

用户将文件归档后，可以将文件取回，要求文件处于冰冻状态才能取回

```
$client->restoreObject($bucket, $archiveKey, $archiveDay);
```

Object权限控制

设置Object的访问权限

如下代码将Object的权限设置为了private：

```
$canned_acl = array("x-bce-acl" => "private");
$client->setObjectCannedAcl($bucket, $key, $canned_acl);
```

其中CannedAcl.php中封装了三个参数：ACL_PRIVATE、ACL_PUBLIC_READ、ACL_PUBLIC_READ_WRITE，它们分别对应的相关权限为：private、public-read、public-read-write，Object权限控制目前支持前两种。关于权限的具体内容可以参考《BOS API文档 Object权限控制》。

设置指定用户对Object的访问权限

BOS提供setObjectAcl方法和setObjectCannedAcl方法来实现指定用户Object的访问权限设置，可以参考如下代码实现：

1. 通过setObjectCannedAcl的x-bce-grant-read和x-bce-grant-full-control设置指定用户的访问权限

```
$canned_acl = array("x-bce-grant-read" => "id=\"7f34788d02a64a9c98f85600567d98a7\",id=\"7f34788d02a64a9c98f85600567d98a8\"");
$client->setObjectCannedAcl($bucket, $object, $canned_acl);

$canned_acl = array("x-bce-grant-full-control" => "id=\"7f34788d02a64a9c98f85600567d98a7\",id=\"7f34788d02a64a9c98f85600567d98a8\"");
$client->setObjectCannedAcl($bucket, $object, $canned_acl);
```

2. 通过setObjectAcl设置object访问权限

```
$my_acl = array(
    array(
        'grantee' => array(
            array(
                'id' => '7f34788d02a64a9c98f85600567d98a7',
            ),
            array(
                'id' => 'aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa',
            ),
        ),
        'permission' => array('FULL_CONTROL'),
    ),
);
$client->setObjectAcl($bucket, $object, $my_acl);
```

> **注意：**
>
> 1. permission中的权限设置包含两个值：READ、FULL_CONTROL，它们分别对应相关权限。
> 2. 设置两个以上（含两个）被授权人时，请参考以上示例的格式，若将数组合并会返回报错。

查看Object的权限

如下代码可以查Object的权限：

```
$response = $client->getObjectAcl($bucket, $object);
```

getObjectAcl方法返回的解析类中可供调用的参数有：

参数	说明
accessControlList	标识Object的权限列表
grantee	标识被授权人
-id	被授权人ID
permission	标识被授权人的权限

删除Object的权限

如下代码可以删除Object的权限：

```
$response = $client->deleteObjectAcl($bucket, $object);
```

删除文件

删除单个文件

可参考如下代码删除了一个Object:

```
$client->deleteObject($bucket, $object);
```

删除多个文件

可参考如下代码删除多个object:

```
$deleteattay = array(
        array(
            "key"=>$diping_key
        ),
    );
$client->deleteMultipleObjects($bucket, $deleteattay);
```

查看文件是否存在

用户可通过如下操作查看某文件是否存在：

```
try {
    $client->getObjectMetadata($bucket, $object);
} catch (\BaiduBce\Exception\BceBaseException $e) {
    if ($e->getStatusCode() == 404) {
        echo $object . " not exist!";
    }
}
```

获取及更新文件元信息

文件元信息(Object Metadata)，是对用户上传BOS的文件的属性描述，分为两种：HTTP标准属性（HTTP Headers）和User Meta（用户自定义元信息）。

获取文件元信息

参考只获取ObjectMetadata。

修改文件元信息

BOS修改Object的Metadata通过拷贝Object实现。即拷贝Object的时候，把目的Bucket设置为源Bucket，目的Object设置为源Object，并设置新的Metadata，通过拷贝自身实现修改Metadata的目的。如果不设置新的Metadata，则报错。

```
$userMeta = array("private" => "newvalue");
$contentType ="text/plain"
$options = array(
 BosOptions::USER_METADATA => $userMeta,
    BosOptions::CONTENT_TYPE=>$contentType
);

$client->copyObject($bucket, $key, $bucket, $key,$options);
$response = $client->getObjectMetadata($bucket, $key);
echo $response['userMetadata']['private']
```

拷贝Object

拷贝一个文件

用户可以通过copyObject方法拷贝一个Object，如下代码所示：

```
$client->copyObject($sourceBucketName, $sourceObjectKey, $targetBucketName, $targetObjectKey);
```

copyObject方法可以通过options配置可选参数，参数列表参考如下:

参数	说明
USER_METADATA	用户自定义Meta，包含Key-Value对
ETAG	Source Object的Etag，若选择上传，则会对Target Object和Source Object的Etag进行比对，若不相同，则返回错误。

```
use BaiduBce\Services\Bos\BosOptions;

$userMeta = array("private" => "private data");
$options = array(
    BosOptions::USER_METADATA => $userMeta
);

$client->copyObject($sourceBucketName, $sourceObjectKey, $targetBucketName, $targetObjectKey, $options);
```

同步Copy

当前BOS的CopyObject接口是通过同步方式实现的。同步方式下，BOS端会等待Copy实际完成才返回成功。同步Copy能帮助用户更准确的判断Copy状态，但用户感知的复制时间会变长，且复制时间和文件大小成正比。

同步Copy方式更符合业界常规，提升了与其它平台的兼容性。同步Copy方式还简化了BOS服务端的业务逻辑，提高了服务效率。

如果您使用bce-php-sdk-0.8.17版本之前的SDK，有可能会出现复制请求成功，但文件实际复制失败的情况，所以建议您使用最新版本SDK。

分块拷贝

除了通过CopyObject接口拷贝文件以外，BOS还提供了另外一种拷贝模式——Multipart Upload Copy。用户可以在如下的应用场景内（但不仅限于此），使用Multipart Upload Copy，如：

- 需要支持断点拷贝。
- 拷贝超过5GB大小的文件。
- 网络条件较差，和BOS的服务器之间的连接经常断开。

下面将介绍分步实现三步拷贝。

三步拷贝包含init、“拷贝分块”和complete三步，其中init和complete的操作同分块上传一致，可直接参考初始化Multipart Upload和完成分块上传。

```
 $partNumber = 1;
$offSet = 0;

//设置每块为5MB
$partSize = 5 * 1024 * 1024;
$length = $partSize;
$partList = array();
$bytesLeft = $client->getObjectMetadata($sourceBucketName, $sourceKey)["contentLength"];

//分块上传
while ($bytesLeft > 0) {
    $length = ($length > $bytesLeft) ? $bytesLeft : $length;
    $options = array(
        BosOptions::RANGE => array($offSet, $offSet + $length - 1)
    );
    $response = $client->uploadPartCopy($sourceBucketName, $sourceKey, $targetBucketName, $targetKey, $uploadId, $partNumber, $options);
    array_push($partList, array("partNumber"=>$partNumber, "eTag"=>$response->eTag));
    $partNumber++;
    $bytesLeft -= $length;
    $offSet += $length;
}
```

> 注意: size参数以字节为单位，定义每个分块的大小，除最后一个Part以外，其他的Part大小都要大于5MB。

抓取object

如下代码用于从指定URL抓取资源，并将资源存储到指定的Bucket中。此操作需要请求者对该Bucket有写权限，每次只能抓取一个Object，且用户可以自定义Object的名称，详情可参考FetchObject接口。

```
 use BaiduBce\Services\Bos\BosOptions;

$options = array(
    BosOptions::STORAGE_CLASS => "STANDARD",
    BosOptions::BCE_FETCH_MODE => "async"
);
$fetchSource = "dowload_url";
$client->fetchObject(
        $bucketName,
        $objectName,
        $fetchSource,
        $options
    );
```

用户额度

设置用户额度

如下代码用于设置用户额度

```
$client->putUserQuota($maxBucketCount, $maxCapacityMegaBytes, $maxObjectCount);
```

获取用户额度

如下代码用于获取用户额度

```
$client->getUserQuota();
```

删除用户额度

如下代码用于删除用户额度

```
$client->deleteUserQuota();
```

合规保留

初始化合规保留

```
$client->initBucketObjectLock($BUCKET_NAME, $retentDays);
```

获取合规保留

```
$client->getBucketObjectLock($BUCKET_NAME);
```

删除合规保留

```
$client->deleteBucketObjectLock($BUCKET_NAME);
```

延长合规保留

如下代码可以延长保留时间

```
$client->extendBucketObjectLock($BUCKET_NAME);
```

事件通知

创建事件通知

如下代码用于创建事件通知

```
$mynotification =
        array(
        array(
            'id'=>        "water-test-1",
            'name'=>      "water-rule-1",
            'appId'=>     "water-app-id-1",
            'status'=>    "enabled",
            'resources'=> array("/*.jpg", "/*.png"),
            'events'=>    array("PutObject") ,
            'apps'=> array(
                array(
                    'id' =>       "app-id-1",
                    'eventUrl' => "https://op-log-app-service.chehejia.com/op-log-app-service/v1-0/log/bos/event",
                    'xVars' =>    "xvars",
                ),
            ),
        ));
$client->putNotification($bucket, $mynotification);
```

****获取事件****

如下代码用于获取事件通知

```
$getRes = $client->getNotification($bucket);
```

****删除事件通知****

如下代码用于删除事件

```
$client->deleteNotification($bucket);
```

异常处理

BOS异常提示有如下四种方式：

异常方法	说明
BceBaseException	异常总集
BceClientException	客户端异常
BceServerException	服务器异常
InvalidArgumentException	系统自带异常，参数错误

用户可以使用try获取某个事件所产生的异常：

```
try {
    $client->deleteObject($bucketName, $objectKey);
} catch (\BaiduBce\Exception\BceBaseException $e) {
    print $e->getMessage();
    if (strcmp(get_class($e), "BaiduBce\Exception\BceClientException") == 0) {
        print "Catch a client exception";
    }
    if (strcmp(get_class($e), "BaiduBce\Exception\BceServiceException") == 0) {
        print "Catch a server exception";
    }
    if (strcmp(get_class($e), "BaiduBce\Exception\BceBaseException") == 0) {
        print "Catch a base exception";
    }
    if (strcmp(get_class($e), "BaiduBce\Exception\InvalidArgumentException") == 0) {
        print "Catch a invalid argument exception";
    }
};
```

客户端异常

客户端异常表示客户端尝试向BOS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；当上传文件时发生IO异常时，也会抛出ClientException。

服务端异常

当BOS服务端出现异常时，BOS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BOS错误信息格式

版本变更记录

- PHP SDK开发包[2019-10-30] 版本号 0.9.7
    - 支持设置、获取、删除bucket生命周期、日志规则、回收站、静态网站托管、服务端加密、跨域访问、原图访问
    - 支持fetchObject接口抓取文件到BOS
    - 支持putSuperObjectFromFile接口封装三步上传大文件
- PHP SDK开发包[2019-01-11] 版本号 0.9.2

  修复bug。
- PHP SDK开发包[2018-02-21] 版本号 0.9.1
    - 支持设置、获取、删除Object ACL
    - 支持AppendObject追加写方式上传文件
    - 支持UploadPartCopy分块拷贝文件
    - 支持Replication跨区域复制功能
    - 上传文件接口支持CRC32字段

    - 修复带指定签名header的签名bug

- PHP SDK开发包[2018-02-01] 版本号 0.9

  PHP SDK支持Object ACL、AppendObject、分块拷贝、跨区域复制、CRC32。

- PHP SDK开发包[2017-05-25] 版本号 0.8.24

  PHP SDK支持冷存储

- PHP SDK开发包[2017-02-28] 版本号 0.8.22

  PHP SDK支持通过STS验证方式访问BOS。

- PHP SDK开发包[2016-08-25] 版本号 0.8.19

  BOS支持上传Object的存储类型为低频存储，并可以获取Object的storage class属性。

- PHP SDK开发包[2016-07-21] 版本号 0.8.17

  完善HttpResponse处理流程并支持同步copy功能。

- PHP SDK开发包[2016-04-19] 版本号 0.8.13

    - 支持php7。
    - 优化缩减了phar文件的大小。

- PHP SDK开发包[2016-03-03] 版本号 0.8.9

  支持HTTPS。

- PHP SDK开发包[2015-09-06] 版本号 0.8.6

  支持多区域选择。

- PHP SDK开发包 [2015-07-28] 版本号 0.8.5

  内部优化，提升性能。

- PHP SDK开发包 [2015-07-23] 版本号 0.8.4

    - 访问路径中去掉“v1”。
    - sampleconf支持webserver环境。

- PHP SDK开发包 [2015-07-07] 版本号 0.8.3

  BOS本次无新功能增加。

- PHP SDK开发包 [2015-05-28] 版本号 0.8.2

  BOS本次无新功能增加。

- PHP SDK开发包 [2015-05-14] 版本号 0.8.1

  首次发布：

    - 支持创建、查看、罗列、删除 Bucket
    - 修改、获取Bucket的访问权限
    - 上传、查看、罗列、删除Object

GO-SDK

概述

本文档主要介绍BOS GO SDK的安装和使用。在使用本文档前，您需要先了解BOS的一些基本知识，并已开通了BOS服务。若您还不了解BOS，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
|  |--bos               //BOS服务目录
|  |  |--bos_client.go    //BOS客户端入口
|  |  |--api             //BOS相关API目录
|  |     |--bucket.go     //BOS的Bucket相关API实现
|  |     |--object.go     //BOS的Object相关API实现
|  |     |--multipart.go  //BOS的Multipart相关API实现
|  |     |--module.go     //BOS相关API的数据模型
|  |     |--util.go       //BOS相关API实现使用的工具
|  |--sts               //STS服务目录
|--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

在确认您使用SDK时配置的Endpoint时，可先阅读开发人员指南中关于BOS访问域名的部分，理解Endpoint相关的概念。百度智能云目前开放了多区域支持，请参考区域选择说明。

region与endpoint参考如下链接

https://cloud.baidu.com/doc/BOS/s/akrqd2wcx

常见示例如下:

访问区域	对应Endpoint
BJ	bj.bcebos.com
BD	bd.bcebos.com
SU	su.bcebos.com
GZ	gz.bcebos.com
CD	cd.bcebos.com
HKG	hkg.bcebos.com
FWH	fwh.bcebos.com
FSH	fsh.bcebos.com

获取密钥

要使用百度智能云BOS，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BOS做签名验证。

可以通过如下步骤获得并了解您的AK/SK信息：

注册百度智能云账号

创建AK/SK

新建BOS Client

BOS Client是BOS服务的客户端，为开发者与BOS服务进行交互提供了一系列的方法。

使用AK/SK新建BOS Client

通过AK/SK方式访问BOS，用户可以参考如下代码新建一个BOS Client：

```
 import (
"github.com/baidubce/bce-sdk-go/services/bos"
)

func main() {
 // 用户的Access Key ID和Secret Access Key
 AK, SK := <ak>, <sk>

 // 用户指定的Endpoint
 ENDPOINT := <domain-name>

   clientConfig := bos.BosClientConfiguration{
 Ak:               AK,
 Sk:               SK,
 Endpoint:         ENDPOINT,
 RedirectDisabled: false,
 }

 // 初始化一个BosClient
 bosClient, err := bos.NewClientWithConfig(&clientConfig)
}
```

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》。第三个参数ENDPOINT支持用户自己指定域名，如果设置为空字符串，会使用默认域名作为BOS的服务地址。RedirectDisabled 是否关闭重定向，true关闭。

> **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://bj.bcebos.com。

使用STS创建BOS Client

申请STS token

BOS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。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BOS，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智能云STS使用介绍。

用STS token新建BOS Client

申请好STS后，可将STS Token配置到BOS Client中，从而实现通过STS Token创建BOS 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BOS 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/bos" //导入BOS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
sts, err := stsClient.GetSessionToken(60, "")
if err != nil {
 fmt.Println("get session token failed:", err)
 return
  }

  //若需指定特殊的权限，示例代码如下
  /*
  aclstr := `{
        "accessControlList":[
          {
            "region":"*",
            "effect":"Allow",
            "resource":["*"],
            "service":"bce:bos",
            "permission":["READ", "WRITE"]
          }
        ]
      }`
  sts, err := stsClient.GetSessionToken(60, aclstr)
  /*
fmt.Println("GetSessionToken result:")
fmt.Println("  accessKeyId:", sts.AccessKeyId)
fmt.Println("  secretAccessKey:", sts.SecretAccessKey)
fmt.Println("  sessionToken:", sts.SessionToken)
fmt.Println("  createTime:", sts.CreateTime)
fmt.Println("  expiration:", sts.Expiration)
fmt.Println("  userId:", sts.UserId)

// 使用申请的临时STS创建BOS服务的Client对象，Endpoint使用默认值
bosClient, err := bos.NewClient(sts.AccessKeyId, sts.SecretAccessKey, "")
if err != nil {
 fmt.Println("create bos client failed:", err)
 return
}
stsCredential, err := auth.NewSessionBceCredentials(
 sts.AccessKeyId,
 sts.SecretAccessKey,
 sts.SessionToken)
if err != nil {
 fmt.Println("create sts credential object failed:", err)
 return
}
bosClient.Config.Credentials = stsCredential
}
```

注意： 目前使用STS配置BOS Client时，无论对应BOS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BOS

BOS支持HTTPS传输协议，您可以通过在创建BOS Client对象时指定的Endpoint中指明HTTPS的方式，在BOS GO SDK中使用HTTPS访问BOS服务：

```
 // import "github.com/baidubce/bce-sdk-go/services/bos"

ENDPOINT := "https://bj.bcebos.com" //指明使用HTTPS协议
AK, SK := <your-access-key-id>, <your-secret-access-key>
bosClient, _ := bos.NewClient(AK, SK, ENDPOINT)
```

配置自定义域名/备用域名访问BOS

使用自定义域名 如果希望使用自定义域名作为访问BOS的ENDPOINT，在控制台将自定义域名和BOS某个bucket绑定之后，配置ENDPOINT为自定义域名并打开`CnameEnabled`开关，例如`cdn-test.cdn.bcebos.com`，配置代码如下：

```
 // import "github.com/baidubce/bce-sdk-go/services/bos"

AK, SK := "xxx", "yyy"
ENDPOINT := "cdn-test.cdn.bcebos.com"
bosClient, _ := bos.NewClient(AK, SK, ENDPOINT)
bosClient.Config.CnameEnabled = true
```

使用备用域名 在使用官方域名和自定义域名作为ENDPOINT访问BOS的基础上，SDK进一步支持了使用ENDPOINT主域名访问BOS失败时重试备用域名机制，使用方式如下：

```
 // 如果使用ENDPOINT访问BOS失败，SDK会自动重试BackupEndpoint域名
bosClient.Config.BackupEndpoint = "cdn-test.bj.bcebos.com"
```

配置BOS Client

如果用户需要配置BOS Client的一些细节的参数，可以在创建BOS 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BOS服务：

```
// import "github.com/baidubce/bce-sdk-go/services/bos"

//创建BOS Client对象
AK, SK := <your-access-key-id>, <your-secret-access-key>
ENDPOINT := "bj.bcebos.com"
client, _ := bos.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/bos"

AK, SK := <your-access-key-id>, <your-secret-access-key>
ENDPOINT := "bj.bcebos.com"
client, _ := bos.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/bos"

AK, SK := <your-access-key-id>, <your-secret-access-key>
ENDPOINT := "bj.bcebos.com"
client, _ := bos.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
```

配置网络超时参数

用户可以通过如下示例代码的方式初始化BOS Client并配置各类网络超时参数。

```
// import "github.com/baidubce/bce-sdk-go/services/bos"
AK, SK := <your-access-key-id>, <your-secret-access-key>
ENDPOINT := "bj.bcebos.com"

//生成config，配置网络超时参数
config := bos.NewBosClientConfig(ak, sk, endpoint).
    WithDialTimeout(3 * time.Second).
    WithReadTimeout(10 * time.Second).
    WithWriteTimeout(10 * time.Second).
    WithHttpClientTimeout(600 * time.Second).
    WithResponseHeaderTimeout(5 * time.Second).
    WithTLSHandshakeTimeout(10 * time.Second).
    WithIdleConnectionTimeout(90 * time.Second)
//用上面带有各类网络超时参数config构造 bos client
client, _ := bos.NewClientWithConfig(config)
```

参数说明

用户使用GO SDK访问BOS时，创建的BOS 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
DialTimeout	*time.Duration	建立连接的超时时间，默认30*time.Second
KeepAlive	*time.Duration	活跃连接上keepalive探帧的发送周期，默认30*time.Second
ReadTimeout	*time.Duration	每次读调用的超时时间，默认0，不设置超时时间
WriteTimeout	*time.Duration	每次写调用的超时时间，默认0，不设置超时时间
TLSHandshakeTimeout	*time.Duration	TLS握手的超时时间，默认10 * time.Second
IdleConnectionTimeout	*time.Duration	空闲连接最长保留时间，默认90 * time.Second
ResponseHeaderTimeout	*time.Duration	等待服务端应答headers的超时时间，默认60 * time.Second
HTTPClientTimeout	*time.Duration	从建立连接到读完response body的超时时间，默认1200 * time.Second

> 说明：
>
> 1. 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BOS Client”小节。
> 2. 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。
>
> 3. 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，直到达到最大重试次数或者最长重试时间为止。

Bucket管理

Bucket既是BOS上的命名空间，也是计费、权限控制、日志记录等高级功能的管理实体。

- Bucket名称在所有区域中具有全局唯一性，且不能修改。

> **说明：**
>
> 百度智能云目前开放了多区域支持，请参考区域选择说明 。 目前支持“华北-北京”、“华南-广州”和“华东-苏州”三个区域。北京区域：http://bj.bcebos.com，广州区域：http://gz.bcebos.com，苏州区域：http://su.bcebos.com。

- 存储在BOS上的每个Object都必须包含在一个Bucket中。
- 一个用户最多可创建100个Bucket，但每个Bucket中存放的Object的数量和大小总和没有限制，用户不需要考虑数据的可扩展性。

Bucket权限管理

设置Bucket的访问权限

如下代码将Bucket的权限设置为了private。

```
err := bosClient.PutBucketAclFromCanned(bucketName, "private")
```

用户可设置的CannedACL包含三个值：private 、public-read 、public-read-write，它们分别对应相关权限。具体内容可以参考BOS API文档 使用CannedAcl方式的权限控制。

设置指定用户对Bucket的访问权限

BOS还可以实现设置指定用户对Bucket的访问权限，参考如下代码实现：

```
 // import "github.com/baidubce/bce-sdk-go/bce"
// import "github.com/baidubce/bce-sdk-go/services/bos/api"

// 1. 直接上传ACL文件流
aclBodyStream := bce.NewBodyFromFile("<path-to-acl-file>")
err := bosClient.PutBucketAcl(bucket, aclBodyStream)

// 2. 直接使用ACL json字符串
aclString := `{
    "accessControlList":[
        {
            "grantee":[{
                "id":"e13b12d0131b4c8bae959df4969387b8" //指定用户ID
            }],
            "permission":["FULL_CONTROL"] //指定用户权限
        }
    ]
}`
err := bosClient.PutBucketAclFromString(bucket, aclString)

// 3. 使用ACL文件
err := bosClient.PutBucketAclFromFile(bucket, "<acl-file-name>")

// 4. 使用ACL struct对象设置
grantUser1 := api.GranteeType{"<user-id-1>"}
grantUser2 := api.GranteeType{"<user-id-2>"}
grant1 := api.GrantType{
 Grantee: []api.GranteeType{grantUser1},
 Permission: []string{"FULL_CONTROL"},
}
grant2 := api.GrantType{
 Grantee: []api.GranteeType{granteUser2},
 Permission: []string{"READ"},
}
grantArr := make([]api.GranteeType, 0)
grantArr = append(grantArr, grant1)
grantArr = append(grantArr, grant2)
args := &api.PutBucketAclArgs{grantArr}
err := bosClient.PutBucketAclFromStruct(bucketName, args)
```

注意： Permission中的权限设置包含三个值：`READ`、`WRITE`、`FULL_CONTROL`，它们分别对应相关权限。具体内容可以参考BOS API文档 上传ACL文件方式的权限控制。 ACL规则比较复杂，直接编辑ACL的文件或JSON字符串比较困难，因此提供了第四种方式方便使用代码创建ACL规则。

设置更多Bucket访问权限

1. 通过设置referer白名单方式设置防盗链

```
 aclString := `{
   "accessControlList":[
      {
         "grantee":[{"id":"*"}],       //指定用户ID为全部用户
         "permission":["FULL_CONTROL"], //指定用户权限
         "condition":[{"referer": {"stringEquals": "http://allowed-domain/"}}]
      }
   ]
}`
err := bosClient.PutBucketAclFromString(bucket, aclString)
```

2. 限制客户端IP访问，只允许部分客户端IP访问

```
 aclString := `{
   "accessControlList":[
      {
         "grantee":[{"id":"*"}], //指定用户ID为全部用户
         "permission":["READ"],  //指定用户权限
         "condition":[{"ipAddress": ["ip-1", "ip-2"]}]
      }
   ]
}`
err := bosClient.PutBucketAclFromString(bucket, aclString)
```

设置STS临时token权限

对于通过STS方式创建的临时访问身份，管理员也可进行专门的权限设定。

STS的简介及设置临时权限的方式可参见临时授权访问。

使用BOS GO SDK设置STS临时token权限可参考如下示例：

```
 // import "github.com/baidubce/bce-sdk-go/services/sts"

AK, SK := <your-access-key-id>, <your-secret-access-key>
stsClient, err := sts.NewClient(AK, SK)
aclString := `{
    "accessControlList":[
        {
            "grantee":[{"id":"*"}],       //指定用户ID为全部用户
            "permission":["FULL_CONTROL"], //指定用户权限
            "condition":[{"referer": {"stringEquals": "http://allowed-domain/"}}]
        }
    ]
}`
//使用有效期为300秒且指定ACL的方式获取临时STS token
sts, err := stsClient.GetSessionToken(300, aclString)
```

查看Bucket的访问权限

用户可以通过如下接口查看Bucket的访问权限，注意：Bucket的访问权限不能删除，默认为私有。

```
result, err := bosClient.GetBucketAcl(bucketName)
```

返回的结果对象的字段包含了访问权限的详细内容，具体定义如下：

```
type GetBucketAclResult struct {
AccessControlList []struct{
 Grantee []struct {
  Id string
 }
 Permission []string
}
Owner struct {
 Id string
}
}
```

查看Bucket所属的区域

Bucket Location即Bucket Region，百度智能云支持的各region详细信息可参见区域选择说明。

如下代码可以获取该Bucket的Location信息：

```
location, err := bosClient.GetBucketLocation(bucketName)
```

新建Bucket

如下代码可以新建一个Bucket：

```
 // 新建Bucket的接口为PutBucket，需指定Bucket名称
if loc, err := bosClient.PutBucket(<your-bucket-name>); err != nil {
 fmt.Println("create bucket failed:", err)
} else {
 fmt.Println("create bucket success at location:", loc)
}
```

> **注意：** 由于Bucket的名称在所有区域中是唯一的，所以需要保证bucketName不与其他所有区域上的Bucket名称相同。
>
> Bucket的命名有以下规范：
>
> - 只能包括小写字母，数字，短横线（-）。
> - 必须以小写字母或者数字开头。
> - 长度必须在4-63字节之间。

- **新建lcc bucket**

go sdk支持新建lcc bucket，用户可以在构造api.PutBucketArgs结构体，填入LccLocation，调用PutBucketWithArgs接口创建lcc bucket，api.PutBucketArgs具体参数如下：

参数	类型	说明
TagList	string	Bucket标签
EnableMultiAz	bool	是否开启多AZ
LccLocation	string	lcc id，用于创建lcc bucket到特定的lcc集群
EnableDedicated	bool	用于控制控制台开启lcc识别白名单

如下代码可以新建一个lcc Bucket：

```
 //调用PutBucketWithArgs创建lcc bucket，需指定Bucket名称和lcc id
putBucketArgs := &api.PutBucketArgs{}
putBucketArgs.LccLocation = "<lcc id>"
putBucketArgs.EnableDedicated = true
if loc, err := bosClient.PutBucketWithArgs(<your-bucket-name>, putBucketArgs); err != nil {
 fmt.Println("create lcc bucket failed:", err)
} else {
 fmt.Println("create lcc bucket success at location:", loc)
}
```

列举Bucket

如下代码可以列出用户所有的Bucket：

```
 if res, err := bosClient.ListBuckets(); err != nil {
 fmt.Println("list buckets failed:", err)
} else {
 fmt.Println("owner:", res.Owner)
 for i, b := range res.Buckets {
  fmt.Println("bucket", i)
  fmt.Println("    Name:", b.Name)
  fmt.Println("    Location:", b.Location)
  fmt.Println("    CreationDate:", b.CreationDate)
 }
}
```

删除Bucket

如下代码可以删除一个Bucket：

```
err := bosClient.DeleteBucket(bucketName)
```

> **注意：**
>
> - 在删除前需要保证此Bucket下的所有Object已经已被删除，否则会删除失败。
> - 在删除前确认该Bucket没有开通跨区域复制，不是跨区域复制规则中的源Bucket或目标Bucket，否则不能删除。

判断Bucket是否存在

若用户需要判断某个Bucket是否存在，则如下代码可以做到：

```
 exists, err := bosClient.DoesBucketExist(bucketName)
if err == nil && exists {
 fmt.Println("Bucket exists")
} else {
 fmt.Println("Bucket not exists")
}
```

> **注意：** 如果Bucket不为空（即Bucket中有Object存在），则Bucket无法被删除，必须清空Bucket后才能成功删除。

文件管理

上传文件

在BOS中，用户操作的基本数据单元是Object。Object包含Key、Meta和Data。其中，Key是Object的名字；Meta是用户对该Object的描述，由一系列Name-Value对组成；Data是Object的数据。

BOS GO SDK提供了丰富的文件上传接口，可以通过以下方式上传文件：

- 简单上传
- 追加上传
- 抓取上传
- 分块上传
- 自动三步上传

简单上传

BOS在简单上传的场景中，支持以指定文件形式、以数据流方式、以二进制串方式、以字符串方式执行Object上传，请参考如下代码：

```
 // import "github.com/baidubce/bce-sdk-go/bce"

// 从本地文件上传
etag, err := bosClient.PutObjectFromFile(bucketName, objectName, fileName, nil)

// 从字符串上传
str := "test put object"
etag, err := bosClient.PutObjectFromString(bucketName, objectName, str, nil)

// 从字节数组上传
byteArr := []byte("test put object")
etag, err := bosClient.PutObjectFromBytes(bucketName, objectName, byteArr, nil)

// 从数据流上传
bodyStream, err := bce.NewBodyFromFile(fileName)
etag, err := bosClient.PutObject(bucketName, objectName, bodyStream, nil)

// 使用基本接口，提供必需参数从数据流上传
bodyStream, err := bce.NewBodyFromFile(fileName)
etag, err := bosClient.BasicPutObject(bucketName, objectName, bodyStream)
```

Object以文件的形式上传到BOS中，上述简单上传的接口支持不超过5GB的Object上传。在请求处理成功后，BOS会在Header中返回Object的ETag作为文件标识。

设置文件元信息

文件元信息(Object Meta)，是对用户在向BOS上传文件时，同时对文件进行的属性描述，主要分为分为两种：设置HTTP标准属性（HTTP Headers）和用户自定义的元信息。

设定Object的Http Header

BOS GO SDK本质上是调用后台的HTTP接口，因此用户可以在上传文件时自定义Object的Http Header。常用的http header说明如下：

名称	描述	默认值
Content-MD5	文件数据校验，设置后BOS会启用文件内容MD5校验，把您提供的MD5与文件的MD5比较，不一致会抛出错误	有
Content-Type	文件的MIME，定义文件的类型及网页编码，决定浏览器将以什么形式、什么编码读取文件。如没有指定，BOS则根据文件的扩展名自动生成，如文件没有扩展名则填默认值	application/octet-stream
Content-Disposition	指示MIME用户代理如何显示附加的文件，打开或下载，及文件名称	无
Content-Length	上传的文件的长度，超过流/文件的长度会截断，不足为实际值	流/文件的长度
Expires	缓存过期时间	无
Cache-Control	指定该Object被下载时的网页的缓存行为	无
x-bce-content-crc32c	本次上传数据的crc32c值，本次上传BOS服务端需计算crc32c，并比较客户端和服务端的计算值是否一致，不一致则上传失败。保存到meta，本次response返回crc32c。	空字符串
x-bce-content-crc32c-flag	仅在取true时，本次上传bos服务端需计算crc32c，保存到meta，本次response返回crc32c。	false

参考代码如下：

```
// import "github.com/baidubce/bce-sdk-go/services/bos/api"

args := new(api.PutObjectArgs)

// 设置上传内容的MIME类型
args.ContentType = "text/javascript"

// 设置上传内容的长度
args.ContentLength = 1024

// 设置缓存过期时间
args.Expires = "Mon, 19 Mar 2018 11:55:32 GMT"

// 设置缓存行为
args.CacheControl = "max-age=3600"

// 设置本次上传数据的crc32c值
args.ContentCrc32c = "the crc32c of Your Data"

// 设置本次上传是否需要BOS服务端计算crc32c
args.ContentCrc32cFlag = true

etag, err := bosClient.PutObject(bucketName, objectName, bodyStream, args)
```

> 注意：用户上传对象时SDK会自动设置ContentLength和ContentMD5，用来保证数据的正确性。如果用户自行设定ContentLength，必须为大于等于0且小于等于实际对象大小的数值，从而上传截断部分的内容，为负数或大于实际大小均报错。

用户自定义元信息

BOS支持用户自定义元数据来对Object进行描述。如下代码所示：

```
// import "github.com/baidubce/bce-sdk-go/services/bos/api"

args := new(api.PutObjectArgs)

// 设置用户自定义元数据
args.UserMeta = map[string]string{
 "name1": "my-metadata1",
 "name2": "my-metadata2",
}

etag, err := bosClient.PutObject(bucketName, objectName, bodyStream, args)
```

> **提示：**
>
> - 在上面代码中，用户自定义了一个名字为“name1”和“name2”，值分别为“my-metadata1”和“my-metadata2”的元数据
> - 当用户下载此Object的时候，此元数据也可以一并得到
> - 一个Object可以有多个类似的参数，但所有的User Meta总大小不能超过2KB

上传Object时设置存储类型

BOS支持标准存储、低频存储和冷存储，上传Object并存储为低频存储类型通过指定StorageClass实现，三种存储类型对应的参数如下：

存储类型	参数
标准存储	STANDARD
低频存储	STANDARD_IA
冷存储	COLD

以低频存储为例，代码如下：

```
// import "github.com/baidubce/bce-sdk-go/services/bos/api"

args := new(api.PutObjectArgs)
args.StorageClass = api.STORAGE_CLASS_STANDARD_IA
etag, err := bosClient.PutObject(bucketName, objectName, bodyStream, args)
```

追加上传

上文介绍的简单上传方式，创建的Object都是Normal类型，用户不可再进行追加写，这在日志、视频监控、视频直播等数据复写较频繁的场景中使用不方便。

正因如此，百度智能云BOS特别支持了AppendObject，即以追加写的方式上传文件。通过AppendObject操作创建的Object类型为Appendable Object，可以对该Object追加数据。AppendObject大小限制为0~5G。当您的网络情况较差时，推荐使用AppendObject的方式进行上传，每次追加较小数据（如256kb）。

通过AppendObject方式上传示例代码如下：

```
 // import "github.com/baidubce/bce-sdk-go/services/bos/api"

args := new(api.AppendObjectArgs)

// 1. 原始接口上传，设置为低频存储，设置追加的偏移位置
args.StorageClass = api.STORAGE_CLASS_STANDARD_IA
// offset为0或不设置表示上传新文件，该文件支持追加上传。追加续传时offset参数需要设置为上一次上传完成时文件大小
args.Offset = 1024
res, err := bosClient.AppendObject(bucketName, objectName, bodyStream, args)

// 2. 封装的简单接口，仅支持设置offset
res, err := bosClient.SimpleAppendObject(bucketName, objectName, bodyStream, offset)

// 3. 封装的从字符串上传接口，仅支持设置offset
res, err := bosClient.SimpleAppendObjectFromString(bucketName, objectName, "abc", offset)

// 4. 封装的从给出的文件名上传文件的接口，仅支持设置offset
res, err := bosClient.SimpleAppendObjectFromFile(bucketName, objectName, "<path-to-local-file>", offset)

fmt.Println(res.ETag)            // 打印ETag
fmt.Println(res.ContentMD5)      // 打印ContentMD5
fmt.Println(res.NextAppendOffset) // 打印NextAppendOffset
```

设置文件元信息

文件元信息(Object Meta)，是对用户在向BOS上传文件时，同时对文件进行的属性描述。

设定Object的Http Header

BOS GO SDK本质上是调用后台的HTTP接口，因此用户可以在上传文件时自定义Object的Http Header。常用的http header说明如下：

名称	描述	默认值
x-bce-content-crc32c	本次上传数据的crc32c值，本次上传BOS服务端需计算crc32c，并比较客户端和服务端的计算值是否一致，不一致则上传失败。保存到meta，本次response返回crc32c。	空字符串
x-bce-content-crc32c-flag	仅在取true时，本次上传bos服务端需计算crc32c，保存到meta，本次response返回crc32c。	false

参考代码如下：

```
 // import "github.com/baidubce/bce-sdk-go/services/bos/api"

args := new(api.AppendObjectArgs)

// 设置本次上传数据的crc32c值
args.ContentCrc32c = "the crc32c of this append data"

// 设置本次上传是否需要BOS服务端计算crc32c
args.ContentCrc32cFlag = true

etag, err := bosClient.AppendObject(bucketName, objectName, bodyStream, args, nil)
```

抓取上传

BOS支持用户提供的url自动抓取相关内容并保存为指定Bucket的指定名称的Object。

```
 // import "github.com/baidubce/bce-sdk-go/services/bos/api"

args := new(api.FetchObjectArgs)

// 1. 原始接口抓取，设置为异步抓取模式
args.FetchMode = api.FETCH_MODE_ASYNC
res, err := bosClient.FetchObject(bucketName, objectName, url, args)

// 2. 基本抓取接口，默认为同步抓取模式
res, err := bosClient.BasicFetchObject(bucketName, objectName, url)

// 3. 易用接口，直接指定可选参数
res, err := bosClient.SimpleFetchObject(bucketName, objectName, url,
 api.FETCH_MODE_ASYNC, api.STORAGE_CLASS_STANDARD_IA)

fmt.Printf("res: %v", res) // 打印抓取结果
```

分块上传

除了通过简单上传及追加上传方式将文件上传到BOS以外，BOS还提供了另外一种上传模式 —— Multipart Upload。用户可以在如下的应用场景内（但不仅限于此），使用Multipart Upload上传模式，如：

- 需要支持断点上传。
- 上传超过5GB大小的文件。
- 网络条件较差，和BOS的服务器之间的连接经常断开。
- 需要流式地上传文件。
- 上传文件之前，无法确定上传文件的大小。

BOS GO SDK提供了分块操作的控制参数：

- MultipartSize：每个分块的大小，默认为10MB，最小不得低于5MB
- MaxParallel：分块操作的并发数，默认为10

下面的示例代码设置了分块的大小为20MB，并发数为100：

```
// import "github.com/baidubce/bce-sdk-go/services/bos"

client := bos.NewClient(<your-ak>, <your-sk>, <endpoint>)
client.MultipartSize = 20 * (1 << 20)
client.MaxParallel = 100
```

除了上述参数外，还会对设置的每个分块数进行1MB对齐，同时限制是最大分块数目不得超过10000，如果分块较小导致分块数超过这个上限会自动调整分块大小。

下面将一步步介绍Multipart Upload的实现。假设有一个文件，本地路径为 /path/to/file.zip，由于文件比较大，将其分块传输到BOS中。

初始化Multipart Upload

使用BasicInitiateMultipartUpload方法来初始化一个基本的分块上传事件：

```
res, err := bosClient.BasicInitiateMultipartUpload(bucketName, objectKey)
fmt.Println(res.UploadId) // 打印初始化分块上传后获取的UploadId
```

返回结果中含有 UploadId ，它是区分分块上传事件的唯一标识，在后面的操作中，我们将用到它。

上传低频存储类型Object的初始化

BOS GO SDK提供的InitiateMultipartUpload接口可以设置其他分块上传的相关参数，下面的代码初始化了低频存储的一个分块上传事件：

```
// import "github.com/baidubce/bce-sdk-go/services/bos/api"

args := new(api.InitiateMultipartUploadArgs)
args.StorageClass = api.STORAGE_CLASS_STANDARD_IA
res, err := bosClient.InitiateMultipartUpload(bucketName, objectKey, contentType, args)
fmt.Println(res.UploadId) // 打印初始化分块上传后获取的UploadId
```

上传冷存储类型Object的初始化

初始化低频存储的一个分块上传事件：

```
// import "github.com/baidubce/bce-sdk-go/services/bos/api"

args := new(api.InitiateMultipartUploadArgs)
args.StorageClass = api.STORAGE_CLASS_COLD
res, err := bosClient.InitiateMultipartUpload(bucketName, objectKey, contentType, args)
fmt.Println(res.UploadId) // 打印初始化分块上传后获取的UploadId
```

上传分块

接着，把文件分块上传。

```
// import "github.com/baidubce/bce-sdk-go/bce"
// import "github.com/baidubce/bce-sdk-go/services/bos"
// import "github.com/baidubce/bce-sdk-go/services/bos/api"

file, _ := os.Open("/path/to/file.zip")

// 分块大小按MULTIPART_ALIGN=1MB对齐
partSize := (bosClient.MultipartSize +
 bos.MULTIPART_ALIGN - 1) / bos.MULTIPART_ALIGN * bos.MULTIPART_ALIGN

// 获取文件大小，并计算分块数目，最大分块数MAX_PART_NUMBER=10000
fileInfo, _ := file.Stat()
fileSize := fileInfo.Size()
partNum := (fileSize + partSize - 1) / partSize
if partNum > bos.MAX_PART_NUMBER { // 超过最大分块数，需调整分块大小
 partSize = (fileSize + bos.MAX_PART_NUMBER + 1) / bos.MAX_PART_NUMBER
 partSize = (partSize + bos.MULTIPART_ALIGN - 1) / bos.MULTIPART_ALIGN * bos.MULTIPART_ALIGN
 partNum = (fileSize + partSize - 1) / partSize
}

// 创建保存每个分块上传后的ETag和PartNumber信息的列表
partEtags := make([]api.UploadInfoType, 0)

// 逐个分块上传
for i := int64(1); i <= partNum; i++ {
 // 计算偏移offset和本次上传的大小uploadSize
 uploadSize := partSize
 offset := partSize * (i - 1)
 left := fileSize - offset
 if left < partSize {
  uploadSize = left
 }

 // 创建指定偏移、指定大小的文件流
 partBody, _ := bce.NewBodyFromSectionFile(file, offset, uploadSize)

 // 上传当前分块
 etag, err := bosClient.BasicUploadPart(bucketName, objectKey, uploadId, int(i), partBody)

 // 保存当前分块上传成功后返回的序号和ETag
 partEtags = append(partEtags, api.UploadInfoType{int(i), etag})
}
```

上面代码的核心是调用 BasicUploadPart 方法来上传每一个分块，但是要注意以下几点：

- BasicUploadPart 方法要求除最后一个Part以外，其他的Part大小都要大于等于5MB。但是该接口并不会立即校验上传Part的大小；只有当Complete Multipart Upload的时候才会校验。
- Part号码的范围是1~10000。如果超出这个范围，BOS将返回InvalidArgument的错误码。
- 每次上传Part之后，BOS的返回结果会包含一个 PartETag对象，它是上传块的ETag与块编号（PartNumber）的组合，在后续完成分块上传的步骤中会用到它，因此需要将其保存起来。一般来讲这些PartETag 对象将被保存到List中。

带参数的上传分块

上传分块时，可以使用 UploadPart 设置文件元信息(Object Meta)。文件元信息是对用户在向BOS上传文件时，同时对文件进行的属性描述。用户可以在上传文件时自定义Object的Http Header，达到设置文件元信息的目的。常用的http header说明如下：

名称	描述	默认值
x-bce-content-crc32c	本次上传数据的crc32c值，本次上传BOS服务端需计算crc32c，并比较客户端和服务端的计算值是否一致，不一致则上传失败。保存到meta，本次response返回crc32c。	空字符串
x-bce-content-crc32c-flag	仅在取true时，本次上传bos服务端需计算crc32c，保存到meta，本次response返回crc32c。	false

参考代码如下：

```
 // import "github.com/baidubce/bce-sdk-go/bce"
// import "github.com/baidubce/bce-sdk-go/services/bos"
// import "github.com/baidubce/bce-sdk-go/services/bos/api"

file, _ := os.Open("/path/to/file.zip")

// 分块大小按MULTIPART_ALIGN=1MB对齐
partSize := (bosClient.MultipartSize +
 bos.MULTIPART_ALIGN - 1) / bos.MULTIPART_ALIGN * bos.MULTIPART_ALIGN

// 获取文件大小，并计算分块数目，最大分块数MAX_PART_NUMBER=10000
fileInfo, _ := file.Stat()
fileSize := fileInfo.Size()
partNum := (fileSize + partSize - 1) / partSize
if partNum > bos.MAX_PART_NUMBER { // 超过最大分块数，需调整分块大小
 partSize = (fileSize + bos.MAX_PART_NUMBER + 1) / bos.MAX_PART_NUMBER
 partSize = (partSize + bos.MULTIPART_ALIGN - 1) / bos.MULTIPART_ALIGN * bos.MULTIPART_ALIGN
 partNum = (fileSize + partSize - 1) / partSize
}

// 创建保存每个分块上传后的ETag和PartNumber信息的列表
partEtags := make([]api.UploadInfoType, 0)

// 逐个分块上传
for i := int64(1); i <= partNum; i++ {
 // 计算偏移offset和本次上传的大小uploadSize
 uploadSize := partSize
 offset := partSize * (i - 1)
 left := fileSize - offset
 if left < partSize {
  uploadSize = left
 }

 // 创建指定偏移、指定大小的文件流
 partBody, _ := bce.NewBodyFromSectionFile(file, offset, uploadSize)

    args := new(api.UploadPartArgs)

    // 设置本次上传数据的crc32c值
    args.ContentCrc32c = "the crc32c of this part"

    // 设置本次上传是否需要BOS服务端计算crc32c
    args.ContentCrc32cFlag = true

 // 上传当前分块
 etag, err := bosClient.UploadPart(bucketName, objectKey, uploadId, int(i), partBody)

 // 保存当前分块上传成功后返回的序号和ETag
 partEtags = append(partEtags, api.UploadInfoType{int(i), etag})
}
```

完成分块上传

如下代码所示，完成分块上传：

```
 // import "github.com/baidubce/bce-sdk-go/services/bos/api"

completeArgs := api.CompleteMultipartUploadArgs{Parts: partEtags}
res, _ := bosClient.CompleteMultipartUploadFromStruct(
 bucketName, objectKey, uploadId,&completeArgs)

// 输出结果对象的内容
fmt.Println(res.Location)
fmt.Println(res.Bucket)
fmt.Println(res.Key)
fmt.Println(res.ETag)
```

上面代码中的 partETags是第二部中保存的partETag的列表，BOS收到用户提交的Part列表后，会逐一验证每个数据Part的有效性。当所有的数据Part验证通过后，BOS将把这些数据part组合成一个完整的Object。

取消分块上传

用户可以使用abortMultipartUpload方法取消分块上传。

```
 bosClient.AbortMultipartUpload(bucketName, objectKey, uploadId)
```

获取未完成的分块上传

用户可以使用 ListMultipartUploads 方法获取Bucket内未完成的分块上传事件。

```
 // 列出给定bucket下所有未完成的分块信息
res, err := bosClient.BasicListMultipartUploads(bucketName)

// 输出返回结果状态信息
fmt.Println(res.Bucket)
fmt.Println(res.Delimiter)
fmt.Println(res.Prefix)
fmt.Println(res.IsTruncated)
fmt.Println(res.KeyMarker)
fmt.Println(res.NextKeyMarker)
fmt.Println(res.MaxUploads)

// 遍历所有未完成分块信息列表
for _, multipartUpload := range res.Uploads {
 fmt.Println("Key:", multipartUpload.Key, ", UploadId:", multipartUpload.UploadId)
}
```

> **注意：** 1. 默认情况下，如果Bucket中的分块上传事件的数目大于1000，则只会返回1000个Object，并且返回结果中IsTruncated的值为True，同时返回NextKeyMarker作为下次读取的起点。 2. 若想返回更多分块上传事件的数目，可以使用KeyMarker参数分次读取。

获取所有已上传的块信息

用户可以使用 ListParts 方法获取某个上传事件中所有已上传的块。

```
 // 使用基本接口列出当前上传成功的分块
res, err := bosClient.BasicListParts(bucketName, objectKey, uploadId)

// 使用原始接口提供参数，列出当前上传成功的最多100个分块
args := new(api.ListPartsArgs)
args.MaxParts = 100
res, err := bosClient.ListParts(bucketName, objectKey, uploadId, args)

// 打印返回的状态结果
fmt.Println(res.Bucket)
fmt.Println(res.Key)
fmt.Println(res.UploadId)
fmt.Println(res.Initiated)
fmt.Println(res.StorageClass)
fmt.Println(res.PartNumberMarker)
fmt.Println(res.NextPartNumberMarker)
fmt.Println(res.MaxParts)
fmt.Println(res.IsTruncated)

// 打印分块信息
for _, part := range res.Parts {
 fmt.Println("PartNumber:", part.PartNumber, ", Size:", part.Size,
  ", ETag:", part.ETag, ", LastModified:", part.LastModified)
}
```

> **注意：** 1. 默认情况下，如果Bucket中的分块上传事件的数目大于1000，则只会返回1000个Object，并且返回结果中IsTruncated的值为True，同时返回NextPartNumberMarker作为下次读取的起点。 2. 若想返回更多分块上传事件的数目，可以使用PartNumberMarker参数分次读取。

上述示例是使用API依次实现，没有并发执行，如果需要加快速度需要用户实现并发上传的部分。为了方便用户使用，BOS Client特封装了分块上传的并发接口UploadSuperFile：

- 接口：UploadSuperFile(bucketName, objectName, fileName, storageClass string) error
- 参数:
  - bucketName: 上传对象的bucket的名称
  - objectName: 上传对象的名称
  - fileName: 本地文件名称
  - storageClass: 上传对象的存储类型，默认标准存储
- 返回值:
  - error: 上传过程中的错误，成功则为空

用户只需给出bucket、object、filename即可并发的进行分块上传，同时也可指定上传对象的storageClass。

自动三步上传

对于超过5G的object，提供的封装好的三步上传，请参考如下代码：

```
 // import "github.com/baidubce/bce-sdk-go/bce"

// 从本地文件上传
res, err := bosClient.ParallelUpload(bucketName, objectName, fileName, "", nil)
```

返回请求对应的response和错误信息，args参数信息同上面简单上传。

下载文件

BOS GO SDK提供了丰富的文件下载接口，用户可以通过以下方式从BOS中下载文件：

- 简单流式下载
- 下载到本地文件

- 范围下载

简单流式下载

用户可以通过如下代码将Object读取到一个流中：

```
 // 提供Bucket和Object，直接获取一个对象
res, err := bosClient.BasicGetObject(bucketName, objectName)

// 获取ObjectMeta
meta := res.ObjectMeta

// 获取Object的读取流（io.ReadCloser）
stream := res.Body

// 确保关闭Object读取流
defer stream.Close()

// 调用stream对象的Read方法处理Object
...
```

> **注意：** 1. 上述接口的返回结果对象中包含了Object的各种信息，包含Object所在的Bucket、Object的名称、MetaData以及一个读取流。 2. 可通过结果对象的ObjectMeta字段获取对象的元数据，它包含了Object上传时定义的ETag，Http Header以及自定义的元数据。 3. 可通过结果对象的Body字段获取返回Object的读取流，通过操作读取流将Object的内容读取到文件或者内存中或进行其他操作。

下载到本地文件

用户可以通过如下代码直接将Object下载到指定文件：

```
 err := bosClient.BasicGetObjectToFile(bucketName, objectName, "path-to-local-file")
```

范围下载

为了实现更多的功能，可以指定下载范围、返回header来实现更精细化地获取Object。如果指定的下载范围是0 - 100，则返回第0到第100个字节的数据，包括第100个，共101字节的数据，即[0, 100]。

```
 // 指定范围起始位置和返回header
responseHeaders := map[string]string{"ContentType": "image/gif"}
rangeStart := 1024
rangeEnd := 2048
res, err := bosClient.GetObject(bucketName, objectName, responseHeaders, rangeStart, rangeEnd)

// 只指定起始位置start
res, err := bosClient.GetObject(bucketName, objectName, responseHeaders, rangeStart)

// 不指定range
res, err := bosClient.GetObject(bucketName, objectName, responseHeaders)

// 不指定返回可选头部
res, err := bosClient.GetObject(bucketName, objectName, nil)
```

基于范围下载接口，用户可以据此实现文件的分段下载和断点续传。为了方便用户使用，BOS GO SDK封装了并发下载的接口`DownloadSuperFile`：

- 接口：`DownloadSuperFile(bucketName, objectName, fileName string) error`
- 参数:
  - bucketName: 下载对象所在bucket的名称
  - objectName: 下载对象的名称
  - fileName: 该对象保存到本地的文件名称
- 返回值:
  - error: 下载过程中的错误，成功则为空

该接口利用并发控制参数执行并发范围下载，直接下载到用户指定的文件中。

其他使用方法

获取Object的存储类型

Object的storage class属性分为`STANDARD`(标准存储)、`STANDARD_IA`(低频存储)和`COLD`(冷存储)，通过如下代码可以实现：

```
 res, err := bosClient.GetObjectMeta(bucketName, objectName)
fmt.Println(res.StorageClass)
```

只获取Object Metadata

通过GetObjectMeta方法可以只获取Object Metadata而不获取Object的实体。如下代码所示：

```
 res, err := bosClient.GetObjectMeta(bucketName, objectName)
fmt.Printf("Metadata: %+v\n", res)
```

获取文件下载URL

用户可以通过如下代码获取指定Object的URL：

```
 // 1. 原始接口，可设置bucket、object名称，过期时间、请求方法、请求头和请求参数
url := bosClient.GeneratePresignedUrl(bucketName, objectName,
 expirationInSeconds, method, headers, params)

// 2. 基本接口，默认为`GET`方法，仅需设置过期时间
url := bosClient.BasicGeneratePresignedUrl(bucketName, objectName, expirationInSeconds)
```

> **说明：**
>
> - expirationInSeconds为指定的URL有效时长，时间从当前时间算起，为可选参数，不配置时系统默认值为1800秒。如果要设置为永久不失效的时间，可以将expirationInSeconds参数设置为-1，不可设置为其他负数。
> - 如果预期获取的文件时公共可读的，则对应URL链接可通过简单规则快速拼接获取: http://{$bucketName}.{$region}.bcebos.com/{$objectName}。

列举存储空间中的文件

BOS GO SDK支持用户通过以下两种方式列举出object：

- 简单列举
- 通过参数复杂列举

除此之外，用户还可在列出文件的同时模拟文件夹。

简单列举

当用户希望简单快速列举出所需的文件时，可通过ListObjects方法返回ListObjectsResult对象，ListObjectsResult对象包含了此次请求的返回结果。用户可以从ListObjectsResult对象的Contents字段获取Object的所有描述信息。

```
listObjectResult, err := bosClient.ListObjects(bucketName, nil)

// 打印当前ListObjects请求的状态结果
fmt.Println("Name:", listObjectResult.Name)
fmt.Println("Prefix:", listObjectResult.Prefix)
fmt.Println("Delimiter:", listObjectResult.Delimiter)
fmt.Println("Marker:", listObjectResult.Marker)
fmt.Println("NextMarker:", listObjectResult.NextMarker)
fmt.Println("MaxKeys:", listObjectResult.MaxKeys)
fmt.Println("IsTruncated:", listObjectResult.IsTruncated)

// 打印Contents字段的具体结果
for _, obj := range listObjectResult.Contents {
 fmt.Println("Key:", obj.Key, ", ETag:", obj.ETag, ", Size:", obj.Size,
  ", LastModified:", obj.LastModified, ", StorageClass:", obj.StorageClass)
}
```

> **注意：** 1. 默认情况下，如果Bucket中的Object数量大于1000，则只会返回1000个Object，并且返回结果中IsTruncated值为True，并返回NextMarker做为下次读取的起点。 2. 若想增大返回Object的数目，可以使用Marker参数分次读取。

通过参数复杂列举

除上述简单列举外，用户还可通过设置ListObjectsArgs参数实现各种灵活的查询功能。ListObjectsArgs可设置的参数如下：

参数	功能
Prefix	限定返回的object key必须以prefix作为前缀
Delimiter	分隔符，是一个用于对Object名字进行分组的字符所有名字包含指定的前缀且第一次出现。Delimiter字符之间的Object作为一组元素
Marker	设定结果从marker之后按字母排序的第一个开始返回
MaxKeys	限定此次返回object的最大数，如果不设定，默认为1000，max-keys取值不能大于1000

> **注意：** 1. 如果有Object以Prefix命名，当仅使用Prefix查询时，返回的所有Key中仍会包含以Prefix命名的Object，详见递归列出目录下所有文件。 2. 如果有Object以Prefix命名，当使用Prefix和Delimiter组合查询时，返回的所有Key中会有Null，Key的名字不包含Prefix前缀，详见查看目录下的文件和子目录。

下面我们分别以几个案例说明通过参数列举的方法：

```
// import "github.com/baidubce/bce-sdk-go/services/bos/api"

args := new(api.ListObjectsArgs)

// 指定最大返回参数为500
args.MaxKeys = 500

// 指定满足特定前缀
args.Prefix = "my-prefix/"

// 指定分隔符，实现类似文件夹的功能
args.Delimiter = "/"

// 设置特定Object之后的排序结果
args.Marker = "bucket/object-0"

listObjectResult, err := bosClient.ListObjects(bucketName, args)
```

模拟文件夹功能

在BOS的存储结果中是没有文件夹这个概念的，所有元素都是以Object来存储，但BOS的用户在使用数据时往往需要以文件夹来管理文件。因此，BOS提供了创建模拟文件夹的能力，其本质上来说是创建了一个size为0的Object。对于这个Object可以上传下载，只是控制台会对以“/”结尾的Object以文件夹的方式展示。

用户可以通过Delimiter和Prefix参数的配合模拟出文件夹功能。Delimiter和Prefix的组合效果是这样的：

如果把Prefix设为某个文件夹名，就可以罗列以此Prefix开头的文件，即该文件夹下递归的所有的文件和子文件夹（目录）。文件名在Contents中显示。 如果再把 Delimiter 设置为“/”时，返回值就只罗列该文件夹下的文件和子文件夹（目录），该文件夹下的子文件名（目录）返回在CommonPrefixes 部分，子文件夹下递归的文件和文件夹不被显示。

如下是几个应用方式：

列出Bucket内所有文件

当用户需要获取Bucket下的所有文件时，可以参考如下代码：

```
// import "github.com/baidubce/bce-sdk-go/services/bos/api"

args := new(api.ListObjectsArgs)
args.Delimiter = "/"
listObjectResult, err := bosClient.ListObjects(bucketName, args)
```

递归列出目录下所有文件

可以通过设置 `Prefix` 参数来获取某个目录下所有的文件：

```
// import "github.com/baidubce/bce-sdk-go/services/bos/api"

args := new(api.ListObjectsArgs)
args.Prefix = "fun/"
args.MaxKeys = 1000
listObjectResult, err := bosClient.ListObjects(bucketName, args)
fmt.Println("Objects:")
for _, obj := range listObjectResult.Contents {
 fmt.Println(obj.Key)
}
```

输出：

```
Objects:
fun/
fun/movie/001.avi
fun/movie/007.avi
fun/test.jpg
```

查看目录下的文件和子目录

在 `Prefix` 和 `Delimiter` 结合的情况下，可以列出目录下的文件和子目录：

```
// import "github.com/baidubce/bce-sdk-go/services/bos/api"

args := new(api.ListObjectsArgs)
args.Delimiter = "/"
args.Prefix = "fun/"
listObjectResult, err := bosClient.ListObjects(bucketName, args)

// 遍历所有的Objects（当前目录和直接子文件）
fmt.Println("Objects:")
for _, obj := range listObjectResult.Contents {
 fmt.Println(obj.Key)
}

// 遍历所有的CommonPrefix（子目录）
fmt.Println("CommonPrefixs:")
for _, obj := range listObjectResult.CommonPrefixes {
 fmt.Println(obj.Prefix)
}
```

输出：

```
Objects:
fun/
fun/test.jpg

CommonPrefixs:
fun/movie/
```

返回的结果中，`ObjectSummaries` 的列表中给出的是fun目录下的文件。而`CommonPrefixs`的列表中给出的是fun目录下的所有子文件夹。可以看出`fun/movie/001.avi` ，`fun/movie/007.avi`两个文件并没有被列出来，因为它们属于 `fun` 文件夹下的 `movie` 目录。

列举Bucket中object的存储属性

当用户完成上传后，如果需要查看指定Bucket中的全部Object的storage class属性，可以通过如下代码实现：

```
listObjectResult, err := bosClient.ListObjects(bucketName, args)
for _, obj := range listObjectResult.Contents {
 fmt.Println("Key:", obj.Key)
 fmt.Println("LastModified:", obj.LastModified)
 fmt.Println("ETag:", obj.ETag)
 fmt.Println("Size:", obj.Size)
 fmt.Println("StorageClass:", obj.StorageClass)
 fmt.Println("Owner:", obj.Owner.Id, obj.Owner.DisplayName)
}
```

权限控制

设置对象的访问权限

目前BOS支持两种方式设置ACL。第一种是使用Canned Acl，在PutObjectAcl的时候，通过头域的"x-bce-acl"或者"x-bce-grant-permission"来设置对象的访问权限，当前可设置的权限包括private和public-read，两种类型的header不可以同时在一个请求中出现。第二种方式是上传一个ACL文件。详细信息请参考设置Object权限控制

设置Canned ACL

Canned ACL是预定义的访问权限，用户可选择对某个对象进行设置，支持三种接口：

```
// 1. 使用x-bce-acl Header设置
err := bosClient.PutObjectAclFromCanned(bucketName, objectName, cannedAcl) //cannedAcl可取值为：private、public-read

// 2. 使用x-bce-grant-{permission} Header设置
err1 := bosClient.PutObjectAclGrantRead(bucketName, objectName, userId)
err2 := bosClient.PutObjectAclGrantFullControl(bucketName, objectName, userId)
// userId为授权的用户，支持可变参数，传入多个用户ID
```

设置自定义ACL

用户可参考如下代码设置Bucket内的对象的自定义访问权限，支持四种不同参数：

```
// import "github.com/baidubce/bce-sdk-go/bce"
// import "github.com/baidubce/bce-sdk-go/services/bos/api"

// 1. 直接上传ACL文件流
aclBodyStream := bce.NewBodyFromFile("<path-to-acl-file>")
err := bosClient.PutObjectAcl(bucketName, objectName, aclBodyStream)

// 2. 直接使用ACL json字符串
//id 用户id,permission 用户权限
aclString := `{
    "accessControlList":[
        {
            "grantee":[{
                "id":"e13b12d0131b4c8bae959df4969387b8"
            }],
            "permission":["FULL_CONTROL"]
        }
    ]
}`
err := bosClient.PutObjectAclFromString(bucketName, objectName, aclString)

// 3. 使用ACL文件
err := bosClient.PutObjectAclFromFile(bucketName, objectName, "<acl-file-name>")

// 4. 使用ACL struct对象设置
grantUser1 := api.GranteeType{Id: "<user-id-1>"}
grantUser2 := api.GranteeType{Id: "<user-id-2>"}
grant1 := api.GrantType{
 Grantee: []api.GranteeType{grantUser1},
 Permission: []string{"FULL_CONTROL"},
}
grant2 := api.GrantType{
 Grantee: []api.GranteeType{grantUser2},
 Permission: []string{"READ"},
}
grantArr := make([]api.GrantType, 0)
grantArr = append(grantArr, grant1)
grantArr = append(grantArr, grant2)
args := &api.PutObjectAclArgs{AccessControlList: grantArr}
err := bosClient.PutObjectAclFromStruct(bucketName, objectName, args)
```

获取对象的访问权限

如下代码可获取一个对象的访问权限：

```
result, err := bosClient.GetObjectAcl(bucketName, objectName)
```

返回结果对象的字段包含了访问权限的详细内容，具体定义如下：

```
type GetObjectAclResult struct {
AccessControlList []struct{
 Grantee []struct{
  Id string
 }
 Permission []string
}
}
```

删除对象的访问权限

对设置过访问权限的对象，可以调用此接口进行删除：

```
err := bosClient.DeleteObjectAcl(bucketName, objectName)
```

删除文件

删除单个文件

可参考如下代码删除了一个Object:

```
// 指定要删除Object名称和所在的Bucket名称
err := bosClient.DeleteObject(bucketName, objectName)
```

删除多个文件

用户也可通过一次调用删除同一个Bucket下的多个文件，有如下参数：

参数名称	描述	父节点
objects	保存要删除的Object信息的容器，包含一个或多个Object元素	-
+key	要删除的Object的名称	objects

具体示例如下：

```
 // import "github.com/baidubce/bce-sdk-go/services/bos/api"
// import "github.com/baidubce/bce-sdk-go/bce"

objectList := `{
 "objects":[
  {"key": "aaa"},
  {"key": "bbb"}
 ]
}`
// 1. 原始接口，提供多个Object的List Stream
objectListStream, err := bce.NewBodyFromString(objectList)
res, err := bosClient.DeleteMultipleObjects(bucketName, objectListStream)

// 2. 提供json字符串删除
res, err := bosClient.DeleteMultipleObjectsFromString(bucketName, objectList)

// 3. 提供删除Object的List对象
deleteObjectList := make([]api.DeleteObjectArgs, 0)
deleteObjectList = append(deleteObjectList, api.DeleteObjectArgs{"aaa"})
deleteObjectList = append(deleteObjectList, api.DeleteObjectArgs{"bbb"})
multiDeleteObj := &api.DeleteMultipleObjectsArgs{deleteObjectList}
res, err := bosClient.DeleteMultipleObjectsFromStruct(bucketName, multiDeleteObj)

// 4. 直接提供待删除Object的名称列表
deleteObjects := []string{"aaa", "bbb"}
res, err := bosClient.DeleteMultipleObjectsFromKeyList(bucketName, deleteObjects)
```

> **说明：**
>
> 一次删除多个Object的时候，返回的结果里包含了未删除成功的Object名称列表。删除部分对象成功时res里包含了未删除成功的名称列表。 删除部分对象成功时err为nil且res不为nil，判断全部删除成功：err为nil且res为&{[]}。

查看文件是否存在

用户可通过如下操作查看某文件是否存在：

```
 // import "github.com/baidubce/bce-sdk-go/bce"

_, err := bosClient.GetObjectMeta(bucketName, objectName)
if realErr, ok := err.(*bce.BceServiceError); ok {
 if realErr.StatusCode == 404 {
  fmt.Println("object not exists")
 }
}
fmt.Println("object exists")
```

获取及更新文件元信息

文件元信息(Object Metadata)，是对用户上传BOS的文件的属性描述，分为两种：HTTP标准属性（HTTP Headers）和User Meta（用户自定义元信息）。

获取文件元信息

用户通过GetObjectMeta方法可以只获取Object Metadata而不获取Object的实体。如下代码所示：

```
 res, err := bosClient.GetObjectMeta(bucketName, objectName)
fmt.Printf("Metadata: %+v\n", res)
```

修改文件元信息

BOS修改Object的Metadata通过拷贝Object实现。即拷贝Object的时候，把目的Bucket设置为源Bucket，目的Object设置为源Object，并设置新的Metadata，通过拷贝自身实现修改Metadata的目的。如果不设置新的Metadata，则报错。这种方式下必须使用拷贝模式为"replace"（默认情况为"copy"）。示例如下：

```
 // import "github.com/baidubce/bce-sdk-go/bce"

args := new(api.CopyObjectArgs)

// 必须设置拷贝模式为"replace"，默认为"copy"是不能执行Metadata修改的
args.MetadataDirective="replace"

// 设置Metadata参数值，具体字段请参考官网说明
args.LastModified = "Wed, 29 Nov 2017 13:18:08 GMT"
args.ContentType = "text/json"

// 使用CopyObject接口修改Metadata，源对象和目的对象相同
res, err := bosClient.CopyObject(bucketName, objectName, bucketName, objectName, args)
```

拷贝文件

拷贝一个文件

用户可以通过CopyObject方法拷贝一个Object，如下代码所示：

```
 // 1. 原始接口，可设置拷贝参数
res, err := bosClient.CopyObject(bucketName, objectName, srcBucketName, srcObjectName, nil)

// 2. 忽略拷贝参数，使用默认
res, err := bosClient.BasicCopyObject(bucketName, objectName, srcBucketName, srcObjectName)

fmt.Println("ETag:", res.ETag, "LastModified:", res.LastModified)
```

上述接口返回的结果对象中包含了新Object的ETag和修改时间LastModified。

设置拷贝参数拷贝Object

示例代码：

```
// import "github.com/baidubce/bce-sdk-go/services/bos/api"

args := new(api.CopyObjectArgs)

// 设置用户自定义Metadata
args.UserMeta = map[string]string{"<user-meta-key>": "<user-meta-value>"}

res, err := bosClient.CopyObject(bucketName, objectName, srcBucketName, srcObjectName, args)
fmt.Println("ETag:", res.ETag, "LastModified:", res.LastModified)
```

用户在执行拷贝的过程中，可以对源Object的Etag或修改状态进行判断，根据判断结果决定是否执行拷贝。详细的参数解释如下：

名称	类型	描述	是否必需
x-bce-copy-source-if-match	String	如果源Object的ETag值和用户提供的ETag相等，则执行拷贝操作，否则拷贝失败。	否
x-bce-copy-source-if-none-match	String	如果源Object的ETag和用户提供的ETag不相等，则执行拷贝操作，否则拷贝失败。	否
x-bce-copy-source-if-unmodified-since	String	如果源object在x-bce-copy-source-if-unmodified-since之后没被修改，则执行拷贝操作，否则拷贝失败。	否
x-bce-copy-source-if-modified-since	String	如果源object在x-bce-copy-source-if-modified-since之后被修改了，则执行拷贝操作，否则拷贝失败。	否
x-bce-acl	String	CannedACL支持的header，用户设置Object的权限，取值为private和public-read。	否
x-bce-content-crc32c	String	本次上传数据的crc32c值，本次上传BOS服务端需计算crc32c，并比较客户端和服务端的计算值是否一致，不一致则上传失败。保存到meta，本次response返回crc32c。	否
x-bce-content-crc32c-flag	Bool	仅在取true时，本次上传bos服务端需计算crc32c，保存到meta，本次response返回crc32c。	否

对应的示例代码：

```
// import "github.com/baidubce/bce-sdk-go/services/bos/api"

args := new(api.CopyObjectArgs)

// 设置用户自定义Metadata
args.UserMeta = map[string]string{"<user-meta-key>": "<user-meta-value>"}

// 设置copy-source-if-match
args.IfMatch = "111111111183bf192b57a4afc76fa632"

// 设置copy-source-if-none-match
args.IfNoneMatch = "111111111183bf192b57a4afc76fa632"

// 设置copy-source-if-modified-since
args.IfModifiedSince = "Fri, 16 Mar 2018 17:07:21 GMT"

// 设置copy-source-if-unmodified-since
args.IfUnmodifiedSince = "Fri, 16 Mar 2018 17:07:21 GMT"

// 设置x-bce-acl
args.CannedAcl = "private"

// 设置本次上传数据的crc32c值
args.ContentCrc32c = "the crc32c of Your Data"

// 设置本次上传是否需要BOS服务端计算crc32c
args.ContentCrc32cFlag = true

res, err := bosClient.CopyObject(bucketName, objectName, srcBucketName, srcObjectName, args)
fmt.Println("ETag:", res.ETag, "LastModified:", res.LastModified)
```

分块拷贝

除了通过CopyObject接口拷贝文件以外，BOS还提供了另外一种拷贝模式——ParallelCopy。用户可以在如下的应用场景内（但不仅限于此），使用ParallelCopy，如：

- 需要支持断点拷贝。
- 拷贝超过5GB大小的文件。
- 网络条件较差，和BOS的服务器之间的连接经常断开。

示例代码：

```
// 自动三步copy，可设置拷贝参数
res, err := bosClient.ParallelCopy(srcBucketName, srcObjectName, bucketName, objectName, nil, nil)

fmt.Println("ETag:", res.ETag)
```

同步Copy功能

当前BOS的CopyObject接口是通过同步方式实现的。同步方式下，BOS端会等待Copy实际完成才返回成功。同步Copy能帮助用户更准确的判断Copy状态，但用户感知的复制时间会变长，且复制时间和文件大小成正比。

同步Copy方式更符合业界常规，提升了与其它平台的兼容性。同步Copy方式还简化了BOS服务端的业务逻辑，提高了服务效率。

归档类型

归档上传

在上传的时候设置StorageClass为 api.STORAGE_CLASS_ARCHIVE

```
 // import "github.com/baidubce/bce-sdk-go/services/bos/api"
args := api.PutObjectArgs{
    StorageClass:api.STORAGE_CLASS_ARCHIVE,
}
resCommon, err := bosClient.PutObjectFromFile(bucketName, objectName, filename, &args)
```

解冻

使用新封装的api RestoreObject

```
 // import "github.com/baidubce/bce-sdk-go/services/bos/api"
// restoreDays 表示解冻后状态的持续时间，要求是数字，单位是天，取值大于0且不超过30即可。默认解冻持续天数是7天,最长30天。
// RESTORE_TIER_STANDARD  = "Standard"  //标准取回对象 - 默认
// RESTORE_TIER_EXPEDITED = "Expedited" //快速取回对象
err := bosClient.RestoreObject(bucketName, objectName, restoreDays, api.RESTORE_TIER_STANDARD)
```

选取文件

SelectObject接口支持对BOS中指定格式(CSV/JSON)的object内容执行SQL语句，通过SQL这种结构化查询语言对object内容进行筛选、分析、过滤之后再返回用户需要的文件内容。示例代码：

选取CSV

```
 // import  "encoding/base64"

// CSV文件选取参数
csvObject := "test.csv"
originSQL := "select * from BosObject" // 注意此处的 BosObject是固定写法，表示csvObject
base64SQL := base64.StdEncoding.EncodeToString([]byte(originSQL))
csvArgs := &api.SelectObjectArgs{
    SelectType: "csv",
    SelectRequest: &api.SelectObjectRequest{
        Expression: base64SQL,
        ExpressionType: "SQL",
        InputSerialization: &api.SelectObjectInput{
            CompressionType: "NONE",
            CsvParams: map[string]string{
                "fileHeaderInfo":    "IGNORE",
                "recordDelimiter":  "Cg==",
                "fieldDelimiter":    "LA==",
                "quoteCharacter":    "Ig==",
                "commentCharacter": "Iw==",
            },
        },
        OutputSerialization: &api.SelectObjectOutput{
            OutputHeader: false,
            CsvParams: map[string]string{
                "quoteFields":     "ALWAYS",
                "recordDelimiter": "Cg==",
                "fieldDelimiter":  "LA==",
                "quoteCharacter":  "Ig==",
            },
        },
        RequestProgress: &api.SelectObjectProgress{
            Enabled: true,
        },
    },
}
csvRes, err := bosClient.SelectObject(bucketName, csvObject, csvArgs)
if err != nil {
    fmt.Println(err)
    return
}
```

选取JSON

```
 // import  "encoding/base64"

// JSON文件选取参数
jsonObject := "test.json"
originSQL := "select * from BosObject" // 注意此处的 BosObject是固定写法，表示jsonObject
base64SQL := base64.StdEncoding.EncodeToString([]byte(originSQL))
jsonArgs := &api.SelectObjectArgs{
    SelectType: "json",
    SelectRequest: &api.SelectObjectRequest{
        Expression:  base64SQL,
        ExpressionType: "SQL",
        InputSerialization: &api.SelectObjectInput{
            CompressionType: "NONE",
            JsonParams: map[string]string{
                "type": "LINES",
            },
        },
        OutputSerialization: &api.SelectObjectOutput{
            JsonParams: map[string]string{
                "recordDelimiter": "Cg==",
            },
        },
        RequestProgress: &api.SelectObjectProgress{
            Enabled: true,
        },
    },
}
jsonRes, err := bosClient.SelectObject(bucketName, jsonObject, jsonArgs)
if err != nil {
    fmt.Println(err)
    return
}
```

解析结果 SelectObject接口返回的response采用固定结构的编码方式，具体解析方法参考SelectObject代码示例

object tagging相关操作

- PutObjectTag使用示例

```
 args := &api.PutObjectTagArgs{
   ObjectTags: []api.ObjectTags{
      {
         []api.ObjectTag{
            {
               Key:   "key1",
               Value: "value1",
            },
            {
               Key:   "key2",
               Value: "value2",
            },
            {
               Key:   "key3",
               Value: "value3",
            },
         },
      },
   },
}
err = bosClient.PutObjectTag(bucketName, objectName, args)
```

- GetObjectTag使用示例

```
 res, err := bosClient.GetObjectTag(bucketName, objectName)
```

- DeleteObjectTag使用示例

```
 err := bosClient.DeleteObjectTag(bucketName, objectName)
```

数据处理及使用

生命周期管理

BOS支持用户对Bucket设置生命周期规则，以自动将过期的文件清除，节省存储空间。针对不同前缀的文件，用户可以同时设置多条规则。 在为Bucket设置一条生命周期规则时，需注意如下参数的使用方式：

规则项	描述	是否必填	备注
id	规则的标识符	必填	同一个bucket内规则id必须唯一，不能重复。如果用户不填系统会自动帮用户生成一个
status	规则的状态	必填	取值为enabled或disabled，当规则处于disabled时规则不生效
resource	规则对哪些资源生效	必填	举例：对samplebucket里以prefix/为前缀的Object生效：`samplebucket/prefix/*`
condition	规则依赖的条件	必填	目前只支持time形式
+time	时间限制条件	必填	通过定义的dateGreaterThan实现
++dateGreaterThan	描述时间关系	必填	支持绝对时间date和相对时间days。绝对时间date格式为yyyy-mm-ddThh:mm:ssZ，eg. 2016-09-07T00:00:00Z。绝对时间为UTC时间,必须以00:00:00(UTC 0点)结尾；相对时间days的描述遵循ISO8601,支持的最小时间粒度为天，如:$(lastModified)+P7D表示的时间为object的last-modified之后7天。
action	对resource执行的操作动作	必填	-
+name	执行的操作名称	必填	取值为Transition、DeleteObject、AbortMultipartUpload
+storageClass	Object的存储类型	可选	action为Transition时可以设定，取值为STANDARD_IA或COLD，表示从原存储类型转为低频存储或冷存储

设置生命周期规则

可通过如下代码设置一条生命周期规则：

```
// import "github.com/baidubce/bce-sdk-go/bce"

ruleStr := `{
    "rule": [
        {
            "id": "delete-rule-1",
            "status": "enabled",
            "resource": ["my-bucket/abc*"],
            "condition": {
                "time": {
                    "dateGreaterThan": "2018-01-01T00:00:00Z"
                }
            },
            "action": {
                "name": "DeleteObject"
            }
        }
    ]
}`

// 1. 通过stream调用接口进行设置
body, _ := bce.NewBodyFromString(ruleStr)
err := bosClient.PutBucketLifecycle(bucketName, body)

// 2. 直接传入字符串
err := bosClient.PutBucketLifecycleFromString(bucketName, ruleStr)
```

查看生命周期规则

可通过如下代码查看Bucket内的生命周期规则：

```
res, err := bosClient.GetBucketLifecycle(bucketName)
fmt.Printf("%+v\n", res.Rule)
```

删除生命周期规则

可通过如下代码清空生命周期规则：

```
err := bosClient.DeleteBucketLifecycle(bucketName)
```

跨域资源共享

跨域资源共享（Cross-Origin Resource Sharing），简称CORS，是HTML5提供的标准跨域解决方案，BOS目前已经支持CORS标准来实现跨域访问。关于跨域访问的介绍请参考跨域访问。

设置CORS规则

用户可针对Bucket设置CORS规则，支持通过json字符串、文件、流、对象方式设置：

```
// import "github.com/baidubce/bce-sdk-go/service/bos/api"

// 1. 通过流式调用接口进行设置
err := bosClient.PutBucketCors(bucketName, body)

// 2. 直接传入字符串
err := bosClient.PutBucketCorsFromString(bucketName, corsString)

// 3. 传入CORS文件名
err := bosClient.PutBucketCorsFromFile(bucketName, corsFile)

// 4. 传入对象
corsObj := api.BucketCORSType{
    AllowedOrigins: []string{"example.com"},
    AllowedMethods: []string{"HEAD", "GET"},
    AllowedHeaders: []string{"*"},
    AllowedExposeHeaders: []string{"user-custom-expose-header"},
    MaxAgeSeconds: 3600,
}
err := bosClient.PutBucketCorsFromStruct(src, &api.PutBucketCorsArgs{CorsConfiguration:[]api.BucketCORSType{corsObj}})
```

获取CORS规则

用户可获取指定Bucket的CORS规则：

```
result, err := bosClient.GetBucketCors(bucketName)
```

结果对象的定义与`PutBucketCorsFromStruct`接口的请求参数相同。

删除CORS规则

可参考如下代码删除Bucket的CORS规则，删除后的Bucket将无法进行跨域访问。

```
err := bosClient.DeleteBucketCors(bucketName)
```

管理存储类型

每个Bucket会有自身的存储类型，如果该Bucket下的Object上传时未指定存储类型则会默认继承该Bucket的存储类型。

设置Bucket存储类型

Bucket默认的存储类型为标准模式，用户可以通过下面的代码进行设置：

```
storageClass := "STANDARD_IA"
err := bosClient.PutBucketStorageclass(bucketName, storageClass)
```

获取Bucket存储类型

下面的代码可以查看一个Bucket的默认存储类型：

```
storageClass, err := bosClient.GetBucketStorageclass(bucketName)
```

设置访问日志

BOS GO SDK支持将用户访问Bucket时的请求记录记录为日志，用户可以指定访问Bucket的日志存放的位置。日志会包括请求者、Bucket名称、请求时间和请求操作等。关于Bucket日志的详细功能说明可参见设置访问日志。

开启Bucket日志

用户通过设置用于放置日志的Bucket和日志文件前缀来开启Bucket日志功能。下面的示例代码可以设置访问日志的位置和前缀：

```
// import "github.com/baidubce/bce-sdk-go/bce"

// 1. 从JSON字符串设置
loggingStr := `{"targetBucket": "logging-bucket", "targetPrefix": "my-log/"}`
err := bosClient.PutBucketLoggingFromString(bucketName, loggingStr)

// 2. 从参数对象设置
args := new(api.PutBucketLoggingArgs)
args.TargetBucket = "logging-bucket"
args.TargetPrefix = "my-log/"
err := bosClient.PutBucketLoggingFromStruct(bucketName, args)

// 3. 读取json格式的文件进行设置
loggingStrem := bce.NewBodyFromFile("<path-to-logging-setting-file>")
err := bosClient.PutBucketLogging(bucketName, loggingStream)
```

查看Bucket日志设置

下面的代码分别给出了如何获取给定Bucket的日志配置信息：

```
res, err := bosClient.GetBucketLogging(bucketName)
fmt.Println(res.Status)
fmt.Println(res.TargetBucket)
fmt.Println(res.TargetPrefix)
```

关闭Bucket日志

需要关闭Bucket的日志功能时，只需调用删除接口即可实现：

```
err := bosClient.DeleteBucketLogging(bucketName)
```

服务端加密

设置服务端加密功能

用户可针对一个Bucket设置开启服务端加密的功能，所有存储到该Bucket的数据将会进行加密存储，保证数据安全性。

```
err := bosClient.PutBucketEncryption(bucketName, algorithm)
```

`algorithm`参数为加密算法，当前只支持"AES256"加密算法。

获取服务端加密

用户可调用如下接口获取Bucket的服务端加密功能：

```
algorithm, err := bosClient.GetBucketEncryption(bucketName)
```

删除服务端加密功能

用户可调用如下接口删除指定Bucket的服务端加密功能：

```
err := bosClient.DeleteBucketEncryption(bucketName)
```

原图保护功能

用户可针对Bucket下存储的图片设置原图保护功能，用户需指定待保护的资源。

开启原图保护功能

通过如下代码开启原图保护功能：

```
resources := []stirng{
  "bucket/prefix/*",
  "bucket/*/suffix",
  }
err := bosClient.PutBucketCopyrightProtection(bucketName, resources...)
```

`resources`参数表示图片保护的范围，可指定前缀、后缀，`"bucket/*"`表示整个bucket。

获取原图保护设置

用户通过如下示例代码获取Bucket设置的原图保护的资源：

```
resources, err := bosClient.GetBucketCopyrightProtection(bucketName)
```

删除原图保护配置

```
err := bosClient.DeleteBucketCopyrightProtection(bucketName)
```

静态网站托管

开启静态网站托管

用户可通过如下示例代码设置一个Bucket开启静态网站托管的功能：

```
// import "github.com/baidubce/bce-sdk-go/service/bos/api"

// 1. 通过流式调用接口进行设置
err := bosClient.PutBucketStaticWebsite(bucketName, body)

// 2. 直接传入json字符串
jsonStr := `{"index": "index.html", "notFound": "404.html"}`
err := bosClient.PutBucketStaticWebsiteFromString(bucketName, jsonStr)

// 3. 传入对象
args := &api.BucketStaticWebsiteType{
 Index: "index.html",
 NotFound: "404.html",
}
err := bosClient.PutBucketStaticWebsiteFromStruct(bucketName, (*api.PutBucketStaticWebsiteArgs)(args))

// 4. 简单接口设置
err := bosClient.SimplePutBucketStaticWebsite(bucketName, "index.html", "404.html")
```

获取静态网站托管的设置

用户通过如下代码获取指定Bucket的静态网站托管的设置情况：

```
result, err := bosClient.GetBucketStaticWebsite(bucketName)
fmt.Println(result.Index)
fmt.Println(result.NotFound)
```

删除静态网站托管的设置

通过如下示例代码删除指定Bucket的静态网站托管功能：

```
err := bosClient.DeleteBucketStaticWebsite(bucketName)
```

跨区域复制

BOS提供了跨区域复制功能，针对用户在某个物理区域创建的Bucket，为了数据安全或其他目的，可配置将整个Bucket的数据复制到物理上的一个或多个地域

开启Bucket跨区域复制功能

用户可通过如下代码开启Bucket的跨区域复制功能：

```
 // 1. json字符串
jsonStr := `{
  "status":"enabled",
  "resource":[
    "bucket/abc",
    "bucket/cd*"
  ],
  "destination": {
    "bucket":"bucket-name",
    "storageClass":"COLD"
  },
  "replicateHistory": {
    "storageClass":"COLD"
  },
  "replicateDeletes":"enabled",
  "id":"sample-bucket"
}`
err := bosClient.PutBucketReplicationFromString(bucketName, jsonStr, "sample-bucket")

// 2. 使用配置文件名
err := bosClient.PutBucketReplicationFromFile(bucketName, configFile, "sample-bucket")

// 3. 使用参数对象
argsObj := &api.PutBucketReplicationArgs{
  Id: "sample-bucket",
  Status: "enabled",
  Resource: []string{"bucket/abc"},
  ReplicateDeletes: "enabled",
  Destination: &api.BucketReplicationDescriptor{"bucket-abc", "COLD"},
  ReplicateHistory: &api.BucketReplicationDescriptor{"bucket-abc", "COLD"},
}
err := bosClient.PutBucketReplicationFromStruct(bucketName, argsObj, argsObj.Id)

// 4. 使用流
err := bosClient.PutBucketReplication(bucketName, bodyStream, "sample-bucket")
```

获取Bucket跨区域复制的配置

用户可使用如下示例代码获取Bucket的跨区域复制的配置，返回的结果与Put接口字段相同。

```
result, err := bosClient.GetBucketReplication(bucketName, "sample-bucket")
```

删除Bucket跨区域复制配置

用户可使用如下示例代码删除Bucket跨区域复制功能：

```
err := bosClient.DeleteBucketReplication(bucketName, "sample-bucket")
```

获取跨区域复制的进度

由于跨区域复制需要后台进行异步复制操作，用户可通过如下接口查询当前复制到进度：

```
result, err := bosClient.GetBucketReplicationProgress(bucketName, "sample-bucket")
```

返回的结果对象包含了如下字段：

- `Status`(string): 当前的状态
- `HistoryReplicationPercent`(float64): 当前复制到进度
- `LatestReplicationTime`(string): 最近一次执行复制的时间

获取replication规则列表 用户可使用如下示例代码获取bucket replication规则列表

```
list, err := bosClient.ListBucketReplication(srcBucket)
```

镜像回源

开启镜像回源

用户可通过如下示例代码设置一个Bucket开启镜像回源的功能

```
 mirrorRule := &api.MirrorConfigurationRule{
  Prefix:          "prefix",
  SourceUrl:       "http://bj.bcebos.com",
  Mode:            "fetch",
  StorageClass:    "STANDARD",
  PassQueryString: false,
  PassHeaders:     []string{"header3", "header4"},
  IgnoreHeaders:   []string{"header3", "header4"},
  Resource:        "*",
  PrefixReplace:   "test1",
  Version:         "v2",
 }
 mirrorArgs := &api.PutBucketMirrorArgs{
  BucketMirroringConfiguration: []api.MirrorConfigurationRule{*mirrorRule},
 }
 err = bosClient.PutBucketMirror(bucketName, mirrorArgs)
```

获取镜像回源

```
 result, err := bosClient.GetBucketMirror(bucketName)
```

删除镜像回源

```
err = bosClient.DeleteBucketMirror(bucketName)
```

标签管理

为bucket添加标签

用户可通过如下示例代码为bucket绑定标签

```
    tag1 := api.Tag{
        Tagkey: "tagKey1",
        TagValue: "tagValue1",
    }
    tag2 := api.Tag{
        Tagkey: "tagKey2",
        TagValue: "tagValue2",
    }
    tags := []api.Tag{tag1, tag2}
  putBucketTagArgs := &api.PutBucketTagArgs{
        Tags: tags,
  }
  err = bosClient.PutBucketTag(bucketName, putBucketTagArgs)
```

获取bucket标签

```
  result, err := bosClient.GetBucketTag(bucketName)
```

删除Bucket标签

```
  err = bosClient.DeleteBucketTag(bucketName)
```

错误处理

GO语言以error类型标识错误，BOS支持两种错误见下表：

错误类型	说明
BceClientError	用户操作产生的错误
BceServiceError	BOS服务返回的错误

用户使用SDK调用BOS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：

```
  // bosClient 为已创建的BOS Client对象
  bucketLocation, err := bosClient.PutBucket("test-bucket")
  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("create bucket success, bucket location:", bucketLocation)
  }
```

客户端异常

客户端异常表示客户端尝试向BOS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BOS服务端出现异常时，BOS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BOS错误信息格式

sdk日志

SDK日志

BOS GO SDK自行实现了支持六个级别、三种输出（标准输出、标准错误、文件）、基本格式设置的日志模块，导入路径为github.com/baidubce/bce-sdk-go/util/log。输出为文件时支持设置五种日志滚动方式（不滚动、按天、按小时、按分钟、按大小），此时还需设置输出日志文件的目录。详见示例代码。

默认日志

BOS GO SDK自身使用包级别的全局日志对象，该对象默认情况下不记录日志，如果需要输出SDK相关日志需要用户自定指定输出方式和级别，详见如下示例：

```
  // import "github.com/baidubce/bce-sdk-go/util/log"

  // 指定输出到标准错误，输出INFO及以上级别
  log.SetLogHandler(log.STDERR)
  log.SetLogLevel(log.INFO)

  // 指定输出到标准错误和文件，DEBUG及以上级别，以1GB文件大小进行滚动
  log.SetLogHandler(log.STDERR | log.FILE)
  log.SetLogDir("/tmp/gosdk-log")
  log.SetRotateType(log.ROTATE_SIZE)
  log.SetRotateSize(1 << 30)

  // 输出到标准输出，仅输出级别和日志消息
  log.SetLogHandler(log.STDOUT)
  log.SetLogFormat([]string{log.FMT_LEVEL, log.FMT_MSG})
```

说明： 1. 日志默认输出级别为DEBUG 2. 如果设置为输出到文件，默认日志输出目录为/tmp，默认按小时滚动 3. 如果设置为输出到文件且按大小滚动，默认滚动大小为1GB 4. 默认的日志输出格式为：FMT_LEVEL, FMT_LTIME, FMT_LOCATION, FMT_MSG

项目使用

该日志模块无任何外部依赖，用户使用GO SDK开发项目，可以直接引用该日志模块自行在项目中使用，用户可以继续使用GO SDK使用的包级别的日志对象，也可创建新的日志对象，详见如下示例：

```
 // 直接使用包级别全局日志对象（会和GO SDK自身日志一并输出）
log.SetLogHandler(log.STDERR)
log.Debugf("%s", "logging message using the log package in the BOS go sdk")

// 创建新的日志对象（依据自定义设置输出日志，与GO SDK日志输出分离）
myLogger := log.NewLogger()
myLogger.SetLogHandler(log.FILE)
myLogger.SetLogDir("/home/log")
myLogger.SetRotateType(log.ROTATE_SIZE)
myLogger.Info("this is my own logger from the BOS go sdk")
```

版本变更记录

v0.9.3 [2018-09-20]

- 支持Bucket跨区域复制、服务端加密、原图保护
- 支持设置Bucket的CORS规则和静态网站托管
- 支持Object级别的ACL设置
- 修复GeneratePresignedUrl问题

v0.9.2 [2018-3-16]

- 修复go get下载时的错误提示信息
- 修复重试请求时请求的body流丢失的问题
- 完善UploadSuperFile返回的错误提示信息
- 将GeneratePresignedUrl接口统一调整为bucket virtual hosting模式

v0.9.1 [2018-1-4]

首次发布：

- 创建、查看、罗列、删除Bucket，获取位置和判断是否存在
- 支持管理Bucket的生命周期、日志、ACL、存储类型
- 上传、下载、删除、罗列Object，支持分块上传、分块拷贝
- 提供AppendObject功能和FetchObject功能
- 封装并发的下载和分块上传接口

请求管理

请求管理

Golang提供了context库允许调用者主动控制请求行为。在0.9.228版本之后，Go SDK新增了带有context参数的接口，实现对BOS请求的控制。本次主要对处理耗时可能较长的接口支持context参数控制请求。

本文主要介绍如何使用这些接口，支持通过context进行请求控制的接口包括：

- 列举桶/列举对象
- 上传对象
- 拷贝对象
- 下载对象
- 追加上传
- 抓取上传
- 分块上传/分块拷贝

带context参数的上传对象

context.WithCancel返回一个context对象及其相应的cancel方法，何时执行cancel方法由调用者决定，示例代码如下：

```
func PutObjectWithContext(bosClient *bos.Client, ctx context.Context, wg *sync.WaitGroup) {
bucketName := "bucket-test"
objectName := "prefix/object1"
fileName := "/path/to/localfile"
  bodyStream, err := bce.NewBodyFromFile(fileName)
etag, err := bosClient.PutObjectFromFileWithContext(ctx, bucketName, objectName, fileName, &api.PutObjectArgs{})
if err == nil {
 fmt.Printf("put object with context success: %s\n", etag)
} else {
 fmt.Printf("put object with context failed: %s\n", err.Error())
}
wg.Done()
}

func main() {
AK, SK := "Your Access Key", "Your Secret Key"
ENDPOINT := "bj.bcebos.com"
bosClient, _ := bos.NewClient(AK, SK, ENDPOINT)

  // 构造context对象及其cancel方法
  ctx, cancel := context.WithCancel(context.Background())
var wg sync.WaitGroup
wg.Add(1)
  // 启动协程执行上传对象
  go PutObjectWithContext(bosClient, ctx)

  // 等待3s后调用cancel方法，取消上传对象请求
  time.Sleep(3 * time.Second)
  cancel()
  wg.Wait()
}
```

context还提供了 WithTimeout 和 WithDeadline 方法设置请求的控制行为，示例代码如下：

```
// import "github.com/baidubce/bce-sdk-go/bce"

ctx, cancel := context.WithTimeout(context.Background(), 3*time.Second)
//ctx, cancel := context.WithDeadline(context.Background(), time.Now().Add(10*time.Millisecond))
defer cancel()

// 从本地文件上传
etag, err := bosClient.PutObjectFromFileWithContext(ctx, bucketName, objectName, fileName, nil)

// 从字符串上传
str := "test put object"
etag, err := bosClient.PutObjectFromStringWithContext(ctx, bucketName, objectName, str, nil)

// 从字节数组上传
byteArr := []byte("test put object")
etag, err := bosClient.PutObjectFromBytesWithContext(ctx, bucketName, objectName, byteArr, nil)

// 从数据流上传
bodyStream, err := bce.NewBodyFromFile(fileName)
etag, err := bosClient.PutObjectWithContext(ctx, bucketName, objectName, bodyStream, nil)
```

带context参数的拷贝对象

示例代码如下：

```
// 1. 使用context.WithCancel方法控制请求行为
ctx, cancel := context.WithCancel(context.Background())
go bosClient.CopyObjectWithContext(ctx, bucketName, objectName, srcBucketName, srcObjectName, nil)
time.Sleep(3 * time.Second)
cancel()

// 2.使用context.WithTimeout和context.WithDeadline控制请求行为
ctx, cancel := context.WithTimeout(context.Background(), 3*time.Second)
//ctx, cancel := context.WithDeadline(context.Background(), time.Now().Add(10*time.Millisecond))
defer cancel()

res, err := bosClient.CopyObjectWithContext(ctx, bucketName, objectName, srcBucketName, srcObjectName, nil)

fmt.Println("ETag:", res.ETag, "LastModified:", res.LastModified)
```

带context参数的下载对象

示例代码如下：

```
 // 1. 使用context.WithCancel方法控制请求行为
ctx, cancel := context.WithCancel(context.Background())

// 提供Bucket和Object，直接获取一个对象
res, err := bosClient.GetObjectWithContext(ctx, bucketName, objectName, nil)

// 获取ObjectMeta
meta := res.ObjectMeta

// 获取Object的读取流（io.ReadCloser）
stream := res.Body

// 确保关闭Object读取流
defer stream.Close()

// 调用stream对象的Read方法处理Object
...

time.Sleep(3 * time.Second)
cancel()

// 2.使用context.WithTimeout和context.WithDeadline控制请求行为
ctx, cancel := context.WithTimeout(context.Background(), 3*time.Second)
//ctx, cancel := context.WithDeadline(context.Background(), time.Now().Add(10*time.Millisecond))
defer cancel()

res, err := bosClient.GetObjectWithContext(ctx, bucketName, objectName, nil)

// 获取Object的读取流（io.ReadCloser）
stream := res.Body

// 确保关闭Object读取流
defer stream.Close()

// 调用stream对象的Read方法处理Object
...
```

带context参数的追加上传

示例代码如下：

```
 // import "github.com/baidubce/bce-sdk-go/services/bos/api"

args := new(api.AppendObjectArgs)

// 1.使用context.WithCancel方法控制请求行为
ctx, cancel := context.WithCancel(context.Background())
go bosClient.AppendObjectWithContext(ctx, bucketName, objectName, bodyStream, args)
time.Sleep(3 * time.Second)
cancel()

// 2.使用context.WithTimeout和context.WithDeadline控制请求行为
ctx, cancel := context.WithTimeout(context.Background(), 3*time.Second)
//ctx, cancel := context.WithDeadline(context.Background(), time.Now().Add(10*time.Millisecond))
defer cancel()

res, err := bosClient.AppendObjectWithContext(ctx, bucketName, objectName, bodyStream, args)

fmt.Println(res.ETag)             // 打印ETag
fmt.Println(res.ContentMD5)       // 打印ContentMD5
fmt.Println(res.NextAppendOffset) // 打印NextAppendOffset
```

带context参数的抓取上传

示例代码如下：

```
 // import "github.com/baidubce/bce-sdk-go/services/bos/api"

args := new(api.FetchObjectArgs)

// 1.使用context.WithCancel方法控制请求行为
ctx, cancel := context.WithCancel(context.Background())
args.FetchMode = api.FETCH_MODE_SYNC
go bosClient.FetchObject(bucketName, objectName, url, args)
time.Sleep(3 * time.Second)
cancel()

// 2.使用context.WithTimeout和context.WithDeadline控制请求行为
ctx, cancel := context.WithTimeout(context.Background(), 3*time.Second)
//ctx, cancel := context.WithDeadline(context.Background(), time.Now().Add(10*time.Millisecond))
defer cancel()

res, err := bosClient.FetchObject(bucketName, objectName, url, args)

fmt.Printf("res: %v", res) // 打印抓取结果
```

带context参数的分块上传

分块上传包括：初始化分块上传，上传分块，完成分块上传，其中初始化分块上传和完成分块上传一般不通过context控制请求行为，调用方式可参考Go sdk文件管理-分块上传中介绍的初始化分块上传和完成分块上传的调用方式。这里介绍通过context控制上传分块的请求，示例代码如下：

```
 // import "github.com/baidubce/bce-sdk-go/bce"
// import "github.com/baidubce/bce-sdk-go/services/bos"
// import "github.com/baidubce/bce-sdk-go/services/bos/api"

file, _ := os.Open("/path/to/file.zip")

// 分块大小按MULTIPART_ALIGN=1MB对齐
partSize := (bosClient.MultipartSize +
 bos.MULTIPART_ALIGN - 1) / bos.MULTIPART_ALIGN * bos.MULTIPART_ALIGN

// 获取文件大小，并计算分块数目，最大分块数MAX_PART_NUMBER=10000
fileInfo, _ := file.Stat()
fileSize := fileInfo.Size()
partNum := (fileSize + partSize - 1) / partSize
if partNum > bos.MAX_PART_NUMBER { // 超过最大分块数，需调整分块大小
 partSize = (fileSize + bos.MAX_PART_NUMBER + 1) / bos.MAX_PART_NUMBER
 partSize = (partSize + bos.MULTIPART_ALIGN - 1) / bos.MULTIPART_ALIGN * bos.MULTIPART_ALIGN
 partNum = (fileSize + partSize - 1) / partSize
}

// 创建保存每个分块上传后的ETag和PartNumber信息的列表
partEtags := make([]api.UploadInfoType, 0)

// 逐个分块上传
for i := int64(1); i <= partNum; i++ {
 // 计算偏移offset和本次上传的大小uploadSize
 uploadSize := partSize
 offset := partSize * (i - 1)
 left := fileSize - offset
 if left < partSize {
  uploadSize = left
 }

 // 创建指定偏移、指定大小的文件流
 partBody, _ := bce.NewBodyFromSectionFile(file, offset, uploadSize)

   // 使用context.WithTimeout和context.WithDeadline控制请求行为
   ctx, cancel := context.WithTimeout(context.Background(), 3*time.Second)
   //ctx, cancel := context.WithDeadline(context.Background(), time.Now().Add(10*time.Millisecond))
   defer cancel()

 // 上传当前分块
 etag, err := bosClient.UploadPartWithContext(ctx, bucketName, objectKey, uploadId, int(i), partBody, nil)

 // 保存当前分块上传成功后返回的序号和ETag
 partEtags = append(partEtags, api.UploadInfoType{int(i), etag})
}
```

带context参数的分块拷贝

分块拷贝的基本流程与分块上传类似，包括三步，分别是：初始化分块拷贝，分块拷贝和完成分块拷贝，其中第一步与第三步与分块上传相同，第二步分块拷贝使用 `UploadPartCopy` 方法完成，带context参数的分块拷贝方法则为 `UploadPartCopyWithContext`。

初始化Multipart Upload

分块拷贝初始化参数`InitiateMultipartUploadArgs`支持字段如下：

名称	描述	默认值
CacheControl	指定该Object被下载时的网页的缓存行为	无
ContentDisposition	指示MIME用户代理如何显示附加的文件，打开或下载，及文件名称	无
Expires	缓存过期时间	无
StorageClass	指定拷贝目的对象的存储类型	标准存储
ObjectTagging	指定拷贝目的对象的标签	无

上述初始化参数的值可以设置为拷贝源对象的元信息，也可以根据实际使用需求自行设置。

分块拷贝初始化的示例代码如下：

```
 // import "github.com/baidubce/bce-sdk-go/bce"
// import "github.com/baidubce/bce-sdk-go/services/bos"
// import "github.com/baidubce/bce-sdk-go/services/bos/api"

// 获取拷贝源对象的元信息
srcObjectMeta, err := srcClient.GetObjectMeta(srcBucketName, srcObjectName)

// 将拷贝源对象的元信息设置到拷贝目的对象的初始化参数中，具体参数按需设置
initArgs := api.InitiateMultipartUploadArgs{
    CacheControl:       srcObjectMeta.CacheControl,
    ContentDisposition: srcObjectMeta.ContentDisposition,
    Expires:            srcObjectMeta.Expires,
    StorageClass:       srcObjectMeta.StorageClass,
}

initRes, err := api.InitiateMultipartUpload(c, destBucketName, destObjectName, srcObjectMeta.ContentType, &initArgs, c.BosContext)
```

分块拷贝

完成分块拷贝初始化后，执行分块拷贝，分块拷贝可以通过context进行控制，示例代码如下：

```
 // import "github.com/baidubce/bce-sdk-go/bce"
// import "github.com/baidubce/bce-sdk-go/services/bos"
// import "github.com/baidubce/bce-sdk-go/services/bos/api"

size := srcObjectMeta.ContentLength

// 分块大小按MULTIPART_ALIGN=1MB对齐
partSize := (bosClient.MultipartSize + bos.MULTIPART_ALIGN - 1) / bos.MULTIPART_ALIGN * bos.MULTIPART_ALIGN

// 计算分块数目，最大分块数MAX_PART_NUMBER=10000
dstObjectSize := srcObjectMeta.ContentLength
partNum := (dstObjectSize + partSize - 1) / partSize
if partNum > bos.MAX_PART_NUMBER { // 超过最大分块数，需调整分块大小
 partSize = (dstObjectSize + bos.MAX_PART_NUMBER + 1) / bos.MAX_PART_NUMBER
 partSize = (partSize + bos.MULTIPART_ALIGN - 1) / bos.MULTIPART_ALIGN * bos.MULTIPART_ALIGN
 partNum = (dstObjectSize + partSize - 1) / partSize
}

// 创建保存每个分块上传后的ETag和PartNumber信息的列表
partEtags := make([]api.UploadInfoType, 0)

// 逐个分块上传
for i := 1; i <= partNum; i++ {
 // 计算偏移offset和本次上传的大小uploadSize
 uploadSize := partSize
 offset := partSize * (i - 1)
 left := fileSize - offset
 if left < partSize {
  uploadSize = left
 }

   partCopyArgs := api.UploadPartCopyArgs{
      SourceRange: fmt.Sprintf("bytes=%d-%d", (i-1)*partSize, (i-1)*partSize+uploadSize-1),
      IfMatch:     srcMeta.ETag,
   }

   source := fmt.Sprintf("/%s/%s", srcBucketName, srcObjectName)
   copyObjectResult, err := bosClient.UploadPartCopyWithContext(ctx, destBucketName, destObjectName, source, uploadId, i, args)

 // 保存当前分块上传成功后返回的序号和ETag
 partEtags = append(partEtags, api.UploadInfoType{i, copyObjectResult.ETag})
}
```

完成Multipart Upload

所有分块拷贝成功后，最后执行完成分块拷贝的步骤，示例代码如下：

```
 // import "github.com/baidubce/bce-sdk-go/services/bos/api"

completeArgs := api.CompleteMultipartUploadArgs{Parts: partEtags}
res, err := bosClient.CompleteMultipartUploadFromStruct(destBucketName, destObjectName, initRes.UploadId, &completeArgs)

// 输出结果对象的内容
fmt.Println(res.Location)
fmt.Println(res.Bucket)
fmt.Println(res.Key)
fmt.Println(res.ETag)
```

Ruby-SDK

概述

本文档主要介绍BOS Ruby SDK的安装和使用。在使用本文档前，您需要先了解BOS的一些基本知识，并已开通了BOS服务。若您还不了解BOS，可以参考 产品描述和快速入门。

安装SDK工具包

运行环境

Ruby SDK包要求运行环境至少为Ruby 2.0 版本。

安装SDK

方式一：通过gem安装

`gem install baidubce-sdk`

方式二：通过bundler安装

1. 首先得确认安装了bundler，安装命令：`gem install bundler`
2. 在你的应用程序的Gemfile中添加：`gem 'baidubce-sdk', '~> 0.9.0'`，再运行`bundle install`

> **说明：** 用户在安装好gem之后，可以输入irb进入Ruby交互式命令行，输入`require 'baidubce/services/bos/bos_client'`，如果显示"true"则SDK已经顺利安装。github源码链接

SDK目录结构

```
 |——baidubce-sdk.gemspec        //依赖的第三方gem
├——lib
|   └—— baidubce
|       ├—— auth              //BCE签名相关
|       ├—— http              //BCE的http通信相关
|       ├—— services
|       |   └—— bos                 //BOS主目录
|       |       ├—— bos_client.rb    //BOS操作类，所有操作可以通过BosClient类可以完成
|       |       ├—— bos_constants.rb  //BOS常量
|       |   └—— sts                 //STS主目录
|       |       ├—— sts_client.rb    //STS操作类
|       ├—— exception.rb            //BCE客户端的异常
|       ├—— retry_policy.rb         //BCE客户端重试机制
|       ├—— bce_base_client.rb       //BCE公用客户端
|       └—— utils               //BCE公用工具
└——samples                    //使用示例
```

卸载SDK

卸载SDK时，执行`gem uninstall baidubce-sdk`，删除脚本中require和include语句即可。

初始化

确认Endpoint

- 在确认您使用SDK时配置的Endpoint时，可先阅读开发人员指南中关于BOS访问域名的部分，理解Endpoint相关的概念。
- 百度智能云目前开放了多区域支持，请参考区域选择说明。
- 目前支持 “华北-北京”、“华北-保定”、“华东-苏州”、“华南-广州”、“西南-成都”、“中国香港”、“华中-武汉”和“华东-上海”。北京区域： `http://bj.bcebos.com`，保定区域:http://bd.bcebos.com,苏州区域:http://su.bcebos.com,广州区域:http://gz.bcebos.com,成都区域:http://cd.bcebos.com,香港区域:http://hkg.bcebos.com,武汉区域：http://fwh.bcebos.com,上海区域：http://fsh.bcebos.com。

对应信息为：

```
 访问区域 | 对应Endpoint
---|---
BJ | bj.bcebos.com
BD | bd.bcebos.com
SU | su.bcebos.com
GZ | gz.bcebos.com
CD | cd.bcebos.com
HKG| hkg.bcebos.com
FWH | fwh.bcebos.com
FSH | fsh.bcebos.com
```

获取密钥

要使用百度智能云BOS，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BOS做签名验证。

可以通过如下步骤获得并了解您的AK/SK信息：

1. 注册百度智能云账号
2. 创建AK/SK

新建BosClient

BosClient是BOS服务的客户端，为开发者与BOS服务进行交互提供了一系列的方法。

使用AK/SK新建BosClient

通过AK/SK方式访问BOS，用户可以参考如下代码新建一个BosClient：

```
 ##### 使用Ruby SDK，引入bos_client和Baidubce模块
require 'baidubce/services/bos/bos_client'
include Baidubce

##### 配置client参数
credentials = Auth::BceCredentials.new(
    "accessKeyId",
    "secretAccessKey"
)

conf = BceClientConfiguration.new(
    credentials,
    "ENDPOINT"
)
##### 新建BosClient
client = Services::BosClient.new(conf)
```

> **注意：**
>
> 1. 在上面代码中，`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》。
> 2. 如果用户需要自己指定域名，可以通过传入ENDPOINT参数来指定，`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://bj.bcebos.com`。

使用STS创建BosClient

申请STS token

BOS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。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BOS，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智能云STS使用介绍。

用STS token新建BOSClient

申请好STS后，可将STStoken配置到BosClient中，用户可以参考如下代码新建一个BosClient：

1. 首先进行STS的endpoint配置。STS的配置示例如下：

```
 require 'baidubce/services/sts/sts_client'
require 'baidubce/services/bos/bos_client'

credentials = Baidubce::Auth::BceCredentials.new(
    "your ak",
    "your sk"
)

sts_conf = Baidubce::BceClientConfiguration.new(
    credentials,
    "http://sts.bj.baidubce.com"
)
```

2. StsClient的示例代码如下：

```
 # 新建StsClient
sts_client = Baidubce::Services::StsClient.new(sts_conf)
acl = {
            id: '8c47a952db4444c5a097b41be3f24c94',
            accessControlList: [
                {
                    eid: 'shj',
                    service: 'bce:bos',
                    region: 'bj',
                    effect: 'Allow',
                    resource: ["bos-demo"],
                    permission: ["READ"]
                }
            ]
}

# durationSeconds为失效时间，如果为非int值或者不设置该参数，会使用默认的12小时作为失效时间
# sts_client.get_session_token(acl, "test")
# sts_client.get_session_token(acl, 1024)
sts_response = sts_client.get_session_token(acl)

sts_ak = sts_response["accessKeyId"]
sts_sk = sts_response['secretAccessKey']
token = sts_response['sessionToken']
```

> **注意：**其中acl指用户定义的acl，语法请参照访问控制。

3. 将获取到的accessKeyID/secretAccessKey/sessionToken用于新建BosClient。

```
  # 使用获取到的ak, sk, token新建BosClient访问BOS
sts_credentials = Baidubce::Auth::BceCredentials.new(
    sts_ak,
    sts_sk,
    token
)

conf = Baidubce::BceClientConfiguration.new(
    sts_credentials,
    "http://bj.bcebos.com",
)

client = Baidubce::Services::BosClient.new(conf)
```

> **注意：**目前使用STS配置client时，无论对应BOS服务的endpoint在哪里，endpoint都需配置为`http://sts.bj.baidubce.com`。

配置HTTPS协议访问BOS

BOS支持HTTPS传输协议，您可以通过如下两种方式在BOS Ruby SDK中使用HTTPS访问BOS服务：

- 在`endpoint`中指定HTTPS:

```
 # 配置client参数
credentials = Auth::BceCredentials.new(
    "accessKeyId",
    "secretAccessKey"
)

conf = BceClientConfiguration.new(
    credentials,
    "https://bj.bcebos.com"
)
# 新建BosClient
client = Services::BosClient.new(conf)
```

- 通过在`protocol`中指定`https`来设置HTTPS协议:

```
 # 配置client参数
credentials = Auth::BceCredentials.new(
    "accessKeyId",
    "secretAccessKey"
)

options = {
    'protocol' => 'https'
}

conf = BceClientConfiguration.new(
    credentials,
    "bj.bcebos.com",
    options
)

# 新建BosClient
client = Services::BosClient.new(conf)
```

> **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。

配置自定义域名/备用域名访问BOS

使用自定义域名 如果希望使用自定义域名作为访问BOS的ENDPOINT，在控制台将自定义域名和BOS某个bucket绑定之后，配置ENDPOINT为自定义域名并打开cname_enabled开关，例如cdn-test.cdn.bcebos.com，配置代码如下：

```
 require 'baidubce/services/bos/bos_client'
include Baidubce

##### 配置client参数
credentials = Auth::BceCredentials.new(
    "accessKeyId",
    "secretAccessKey"
)
options = {
    'cname_enabled' => true
}
conf = BceClientConfiguration.new(
    credentials,
    "cdn-test.cdn.bcebos.com",
    options
)

##### 新建BosClient
client = Services::BosClient.new(conf)
```

使用备用域名 在使用官方域名和自定义域名作为ENDPOINT访问BOS的基础上，SDK进一步支持了使用ENDPOINT主域名访问BOS失败时重试备用域名机制，使用方式如下：

```
 // 如果使用ENDPOINT访问BOS失败，SDK会自动重试backup_endpoint域名
options = {
    'backup_endpoint' => "bj.bcebos.com"
}
conf = BceClientConfiguration.new(
    credentials,
    "cdn.bcebos.com",
    options
)

##### 新建BosClient
client = Services::BosClient.new(conf)
```

配置BosClient

设置自定义参数

Ruby SDK默认设置了一些基本参数，若用户想要对参数的值进行修改，可以创建自身的参数配置，并在构造BosClient的时候传入，传入代码参考如下：

```
 ##### 配置自定义参数
options = {
    'protocol' => 'https',
    'read_timeout_in_millis' => 1000 * 60,
    'region' => 'bj'
}

conf = BceClientConfiguration.new(
    credentials,
    "http://bj.bcebos.com",
    options
)

##### 新建BosClient
client = Services::BosClient.new(conf)
```

参数说明如下：

参数	说明	默认值
protocol	协议	http
region	区域	bj
open_timeout_in_millis	请求超时时间（单位：毫秒）	50 * 1000
read_timeout_in_millis	通过打开的连接传输数据的超时时间（单位：毫秒）	10 *60* 1000（设置时需要对文件大小和网速进行评估，否则上传大文件时会产生超时）
send_buf_size	发送缓冲区大小	1024 * 1024
recv_buf_size	接收缓冲区大小	10 *1024* 1024

设置可选参数

BosClient将可选的参数封装到options中，每一个方法具有的可选参数详见具体的接口使用方法介绍，现以put_object_from_string方法为例，参考如下代码实现设置可选参数：

```
 ##### 利用options在上传Object的时候传入指定参数
user_metadata = { "key1" => "value1" }
options = { Http::CONTENT_TYPE => 'string',
            "key2" => "value2",
            'Content-Disposition' => 'inline',
            'user-metadata' => user_metadata
}

client.put_object_from_string(bucket_name, object_name, "obj_str", options)
```

Bucket管理

概述

Bucket既是BOS上的命名空间，也是计费、权限控制、日志记录等高级功能的管理实体。

- Bucket名称在所有区域中具有全局唯一性，且不能修改。

> **说明：**
>
> - 百度智能云目前开放了多区域支持，请参考区域选择说明。
> - 目前支持"华北-北京"、"华南-广州"和"华东-苏州"三个区域。北京区域：http://bj.bcebos.com，广州区域：http://gz.bcebos.com，苏州区域：http://su.bcebos.com。

- 存储在BOS上的每个Object都必须包含在一个Bucket中。
- 一个用户最多可创建100个Bucket，但每个Bucket中存放的Object的数量和大小总和没有限制，用户不需要考虑数据的可扩展性。

新建Bucket

如下代码可以新建一个Bucket：

```
 bucketName = "your_bucket";

###### Bucket是否存在，若不存在创建Bucket
client.create_bucket(bucket_name) unless client.does_bucket_exist(bucket_name)
```

> **注意：** 由于Bucket的名称在所有区域中是唯一的，所以需要保证bucketName不与其他所有区域上的Bucket名称相同。
>
> Bucket的命名有以下规范：
>
> - 只能包括小写字母，数字，短横线（-）。
> - 必须以小写字母或者数字开头。
> - 长度必须在4-63字节之间。

通过上述代码创建的bucket，权限是私有读写，存储类型是标准类型（Standard）。用户在控制台创建Bucket时可以指定Bucket权限和存储类型。

Bucket权限管理

设置Bucket的访问权限

如下代码将Bucket的权限设置为了private：

```
 client.set_bucket_canned_acl(bucket_name, "private")
```

canned acl支持三种权限，分别为：private、public-read、public-read-write。关于权限的具体内容可以参考《BOS API文档 使用CannedAcl方式的权限控制》。

设置指定用户对Bucket的访问权限

BOS提供set_bucket_acl方法来实现指定用户对Bucket的访问权限设置，可以参考如下代码实现：

```
 acl = [{'grantee' => [{'id' => 'b124deeaf6f641c9ac27700b41a350a8'},
                       {'id' => 'aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa'}],
        'permission' => ['FULL_CONTROL']
}]

client.set_bucket_acl(bucket_name, acl)
```

> **注意：**
>
> 1. permission中的权限设置包含三个值：READ、WRITE、FULL_CONTROL，它们分别对应相关权限。具体内容可以参考《BOS API文档 上传ACL文件方式的权限控制》。
> 2. 设置两个以上（含两个）被授权人时，请参考以上示例的格式，若将数组合并会返回报错。

设置更多Bucket访问权限

1. 通过设置referer白名单方式设置防盗链

```
acl = [{'grantee' => [{'id' => 'b124deeaf6f641c9ac27700b41a350a8'},
                      {'id' => 'aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa'}],
        'permission' => ['FULL_CONTROL'],
        'condition' => {
           'referer' => {
              'stringLike' => ['http://www.abc.com/*'],
              'stringEquals' => ['http://www.abc.com']
           }
        }
}]

client.set_bucket_acl(bucket_name, acl)
```

2. 限制客户端IP访问，只允许部分客户端IP访问

```
acl = [{'grantee' => [{'id' => 'b124deeaf6f641c9ac27700b41a350a8'},
                      {'id' => 'aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa'}],
        'permission' => ['FULL_CONTROL'],
        'condition' => {
           "ipAddress" => [
                '192.168.0.0/16',
                '192.169.0.*',
                '192.170.0.5'
           ]
        }
}]

client.set_bucket_acl(bucket_name, acl)
```

设置STS临时token权限

对于通过STS方式创建的临时访问身份，管理员也可进行专门的权限设定。 STS的简介及设置临时权限的方式可参见临时授权访问。

使用BOS Ruby SDK设置STS临时token权限可参考使用STS创建BosClient

查看Bucket的权限

如下代码可以查看Bucket的权限：

```
client.get_bucket_acl(bucket_name)
```

`get_bucket_acl`方法返回的解析类中可供调用的参数有：

参数	说明
owner	Bucket owner信息
id	Bucket owner的用户ID
acl	标识Bucket的权限列表
grantee	标识被授权人
-id	被授权人ID
permission	标识被授权人的权限

查看Bucket所属的区域

Bucket Location即Bucket Region，百度智能云支持的各region详细信息可参见区域选择说明 。

如下代码可以获取该Bucket的Location信息：

```
client.get_bucket_location(bucket_name)
```

列举Bucket

如下代码可以列出用户所有的Bucket：

```
buckets = client.list_buckets()
```

判断Bucket是否存在

若用户需要判断某个Bucket是否存在，则如下代码可以做到：

```
client.does_bucket_exist(bucketName)
```

删除Bucket

删除指定Bucket

如下代码可以删除一个Bucket：

```
bucketName = "your_bucket";
client.delete_bucket(bucketName)
```

> **注意：**
>
> - 在删除前需要保证此Bucket下的所有Object和未完成的三步上传Part已经被删除，否则会删除失败。
> - 在删除前确认该Bucket没有开通跨区域复制，不是跨区域复制规则中的源Bucket或目标Bucket，否则不能删除。

删除所有Bucket

将delete_bucket和list_buckets函数结合，可以删除全部Bucket，参考代码如下：

```
###### 列出全部Bucket
buckets = client.list_buckets()['buckets']

###### 遍历删除全部Bucket
buckets.each do |bucket|
  while true
    options = {}
    res = client.list_objects(bucket['name'], options)
    res['contents'].each do |object|
      client.delete_object(bucket['name'], object['key'])
    end
    if res['isTruncated']
      options[:marker] = res['nextMarker']
    else
      break
    end
  end
  client.delete_bucket(bucket['name'])
end
```

文件管理

上传文件

在BOS中，用户操作的基本数据单元是Object。Object包含Key、Meta和Data。其中，Key是Object的名字；Meta是用户对该Object的描述，由一系列Name-Value对组成；Data是Object的数据。

BOS Ruby SDK提供了丰富的文件上传接口，可以通过以下方式上传文件：

- 简单上传
- 追加上传
- 分片上传
- 断点续传上传

简单上传

BOS在简单上传的场景中，支持以指定文件形式、以数据流方式、以二进制串方式、以字符串方式执行Object上传，请参考如下代码：

```
###### 以数据流形式上传Object
client.put_object(bucket_name, object_name, data)

###### 从字符串中上传Object
client.put_object_from_string(bucket_name, object_name, "string")

###### 从文件中直接上传Object
client.put_object_from_file(bucket_name, object_name, file_path)
```

Object以文件的形式上传到BOS中，putObject相关接口支持不超过5GB的Object上传。在PutObject请求处理成功后，BOS会在Header中返回Object的ETag作为文件标识。

设置文件元信息

文件元信息(Object Meta)，是对用户在向BOS上传文件时，同时对文件进行的属性描述，主要分为分为两种：设置HTTP标准属性（HTTP Headers）和用户自定义的元信息。

设定Object的Http Header

BOS Ruby SDK本质上是调用后台的HTTP接口，因此用户可以在上传文件时自定义Object的Http Header。常用的http header说明如下：

名称	描述	默认值
Content-MD5	文件数据校验，设置后BOS会启用文件内容MD5校验，把您提供的MD5与文件的MD5比较，不一致会抛出错误	无
Content-Type	文件的MIME，定义文件的类型及网页编码，决定浏览器将以什么形式、什么编码读取文件。如没有指，BOS则根据文件的扩展名自动生成，如文件没有扩展名则填默认值	application/octet-stream
Content-Disposition	指示MINME用户代理如何显示附加的文件，打开或下载，及文件名称	无
Content-Length	上传的文件的长度，超过流/文件的长度会截断，不足为实际值	流/文件时间长度
Expires	缓存过期时间	无
Cache-Control	指定该Object被下载时的网页的缓存行为	无

参考代码如下：

```
options = { Http::CONTENT_TYPE => 'string',
          Http::CONTENT_MD5 => 'md5',
          Http::CONTENT_DISPOSITION => 'inline',
          'key1' => 'value1'
}

client.put_object_from_string(bucket_name, object_name, "string", options)
```

用户自定义元信息

BOS支持用户自定义元数据来对Object进行描述。如下代码所示：

```
options = {
        'user-metadata' => { "key1" => "value1" }
}

client.put_object_from_string(bucket_name, object_name, "string", options)
```

> **提示：**
>
> - 在上面代码中，用户自定义了一个名字为"key1"，值为"value1"的元数据
> - 当用户下载此Object的时候，此元数据也可以一并得到
> - 一个Object可以有多个类似的参数，但所有的User Meta总大小不能超过2KB

上传Object时设置存储类型

BOS支持标准存储, 低频存储和冷存储，上传Object并存储为低频存储类型通过指定StorageClass实现，三种存储类型对应的参数如下：

存储类型	参数
标准存储	STANDARD
低频存储	STANDARD_IA
冷存储	COLD

以低频存储为例，代码如下：

```
###### 上传一个低频object（默认为标准object）
client.put_object_from_file(bucket_name, object_name, file_path, Http::BOS_STORAGE_CLASS => 'STANDARD_IA')
```

putObject请求处理成功后，BOS会在Header中返回Object的Content-MD5，用户可以根据这个参数来对文件进行校验。

追加上传

上文介绍的简单上传方式，创建的Object都是Normal类型，用户不可再进行追加写，这在日志、视频监控、视频直播等数据复写较频繁的场景中使用不方便。

正因如此，百度智能云BOS特别支持了AppendObject，即以追加写的方式上传文件。通过AppendObject操作创建的Object类型为Appendable Object，可以对该Object追加数据。AppendObject大小限制为0~5G。

通过AppendObject方式上传示例代码如下：

```
###### 从字符串上传一个appendable object
client.append_object_from_string(bucket_name, object_name, "string")

###### 从offset处开始追加写
client.append_object_from_string(bucket_name, object_name, "append_str", 'offset' => 6)
```

分块上传

除了通过简单上传几追加上传方式将文上传件到BOS以外，BOS还提供了另外一种上传模式 —— Multipart Upload。用户可以在如下的应用场景内（但不仅限于此），使用Multipart Upload上传模式，如：

- 需要支持断点上传。
- 上传超过5GB大小的文件。
- 网络条件较差，和BOS的服务器之间的连接经常断开。
- 需要流式地上传文件。
- 上传文件之前，无法确定上传文件的大小。

下面将一步步介绍Multipart Upload的实现。假设有一个文件，本地路径为 `/path/to/file.zip` ，由于文件比较大，将其分块传输到BOS中。

初始化Multipart Upload

使用`initiate_multipart_upload` 方法来初始化一个分块上传事件：

```
upload_id = client.initiate_multipart_upload(bucket_name, object_name)["uploadId"]
```

`initiate_multipart_upload`的返回结果中含有 `uploadId` ，它是区分分块上传事件的唯一标识，在后面的操作中，我们将用到它。

上传低频存储类型Object的初始化

初始化低频存储的一个分块上传事件：

```
options = {
        Http::BOS_STORAGE_CLASS => 'STANDARD_IA'
}
client.initiate_multipart_upload(bucket_name, object_name, options)
```

上传冷存储类型Object的初始化

初始化冷存储的一个分块上传事件：

```
options = {
        Http::BOS_STORAGE_CLASS => 'COLD'
}
client.initiate_multipart_upload(bucket_name, object_name, options)
```

上传分块

接着，把文件分块上传。

```
  ###### 设置分块的开始偏移位置
left_size = File.open(multi_file, "r").size()
offset = 0
part_number = 1
part_list = []

while left_size > 0 do
    part_size = 5 * 1024 * 1024
    if left_size < part_size
        part_size = left_size
    end

    response = client.upload_part_from_file(
        bucket_name, object_name, upload_id, part_number, part_size, multi_file, offset)
    left_size -= part_size
    offset += part_size
    # your should store every part number and etag to invoke complete multi-upload
    part_list << {
        "partNumber" => part_number,
        "eTag" => response['etag']
    }
    part_number += 1
end
```

上面代码的核心是调用 UploadPart 方法来上传每一个分块，但是要注意以下几点：

- UploadPart 方法要求除最后一个Part以外，其他的Part大小都要大于等于5MB。但是Upload Part接口并不会立即校验上传Part的大小；只有当Complete Multipart Upload的时候才会校验。
- 为了保证数据在网络传输过程中不出现错误，建议您在UploadPart后，使用每个分块BOS返回的Content-MD5值分别验证已上传分块数据的正确性。当所有分块数据合成一个Object后，不再含MD5值。
- Part号码的范围是1~10000。如果超出这个范围，BOS将返回InvalidArgument的错误码。
- 每次上传Part时都要把流定位到此次上传块开头所对应的位置。
- 每次上传Part之后，BOS的返回结果会包含eTag和partNumber，需要保存到part_list中。part_list类型是array，里面每个元素是个hash，每个hash包含两个关键字，一个是partNumber, 一个是eTag；在后续完成分块上传的步骤中会用到它。

完成分块上传

如下代码所示，完成分块上传：

```
client.complete_multipart_upload(bucket_name, object_name, upload_id, part_list)
```

上面代码中的 part_list 是第二步中保存的part列表，BOS收到用户提交的Part列表后，会逐一验证每个数据Part的有效性。当所有的数据Part验证通过后，BOS将把这些数据part组合成一个完整的Object。

取消分块上传

用户可以使用abort_multipart_upload方法取消分块上传。

```
client.abort_multipart_upload(bucket_name, object_name, upload_id)
```

获取未完成的分块上传事件

用户可以使用list_multipart_uploads方法获取Bucket中未完成的分块上传事件。

```
response = client.list_multipart_uploads(bucket_name)
puts response['bucket']
puts response['uploads'][0]['key']
```

> **注意：**
>
> 1. 默认情况下，如果Bucket中的分块上传事件的数目大于1000，则只会返回1000个Object，并且返回结果中IsTruncated的值为True，同时返回NextKeyMarker作为下次读取的起点。
> 2. 若想返回更多分块上传事件的数目，可以使用KeyMarker参数分次读取。

获取所有已上传的块信息

用户可以使用list_parts方法获取某个上传事件中所有已上传的块：

```
response = client.list_parts(bucket_name, object_name, upload_id)
puts response['bucket']
puts response['uploads'][0]['key']
```

> **注意：**
>
> 1. 默认情况下，如果Bucket中的分块上传事件的数目大于1000，则只会返回1000个Object，并且返回结果中IsTruncated的值为True，同时返回NextPartNumberMarker作为下次读取的起点。
> 2. 若想返回更多分块上传事件的数目，可以使用PartNumberMarker参数分次读取。

断点续传上传

当用户向BOS上传大文件时，如果网络不稳定或者遇到程序崩等情况，则整个上传就失败了，失败前已经上传的部分也作废，用户不得不重头再来。这样做不仅浪费资源，在网络不稳定的情况下，往往重试多次还是无法完成上传。 基于上述场景，BOS提供了断点续传上传的能力:

- 当网络情况一般的情况下，建议使用三步上传方式，将object分为1Mb的块，参考分块上传。
- 当您的网络情况非常差，推荐使用appendObject的方式进行断点续传，每次append 较小数据256kb，参考追加上传。

> **提示**

> - 断点续传是分片上传的封装和加强，是用分片上传实现的；
> - 文件较大或网络环境较差时，推荐使用分片上传；

下载文件

BOS Ruby SDK提供了丰富的文件下载接口，用户可以通过以下方式从BOS中下载文件：

- 简单流式下载
- 下载到本地文件
- 断点续传下载
- 范围下载

简单流式下载

用户可以通过如下代码将Object读取到一个流中：

```
client.get_object_as_string(bucket_name, object_name)
```

直接下载Object到文件

用户可以参考如下代码将Object下载到指定文件：

```
client.get_object_to_file(bucket_name, object_name, file_name)
```

范围下载

为了实现更多的功能，可以通过配置RANGE参数来指定下载范围，实现更精细化地获取Object。如果指定的下载范围是0 - 100，则返回第0到第100个字节的数据，包括第100个，共101字节的数据，即[0, 100]。RANGE参数的格式为array(offset, endset)其中两个变量为长整型，单位为字节。用户也可以用此功能实现文件的分段下载和断点续传。

```
range = [0,100]
client.get_object_as_string(bucket_name, object_name, range)
```

其他使用方法

获取Object的存储类型

Object的storage class属性分为STANDARD(标准存储), STANDARD_IA(低频存储)和COLD(冷存储)，通过如下代码可以实现：

```
response = client.get_object_meta_data(bucket_name, object_name)
puts response[Http::BOS_STORAGE_CLASS];
```

只获取ObjectMetadata

用户也可通过get_object_meta_data方法可以只获取ObjectMetadata而不获取Object的实体。如下代码所示：

```
response = client.get_object_meta_data(bucket_name, object_name)
puts response['etag'];
```

get_object_meta_data方法返回的解析类中可供调用的参数有：

参数	说明
content-type	Object的类型
content-length	Object的大小
content-md5	Object的MD5
etag	Object的HTTP协议实体标签
x-bce-storage-class	Object的存储类型
user-metadata	如果在PutObject指定了userMetadata自定义meta，则返回此项

变更文件存储等级

上文中已提到，BOS支持为文件赋予STANDARD(标准存储), STANDARD_IA(低频存储)和COLD(冷存储)三种存储类型。同时，BOS Ruby SDK也支持用户对特定文件执行存储类型变更的操作。

涉及到的参数如下：

参数	说明
x-bce-storage-class	指定Object的存储类型，STANDARD_IA代表低频存储，COLD代表冷存储，不指定时默认是标准存储类型。

示例如下：

```
options = {
        Http::BOS_STORAGE_CLASS => 'STANDARD_IA'
}

###### 标准存储转低频存储
client.copy_object(bucket_name, object_name, bucket_name, object_name, options)
puts client.get_object_meta_data(bucket_name, object_name)[Http::BOS_STORAGE_CLASS]

options = {
        Http::BOS_STORAGE_CLASS => 'COLD'
}

###### 低频存储转冷存储
client.copy_object(bucket_name, object_name, bucket_name, object_name, options)
puts client.get_object_meta_data(bucket_name, object_name)[Http::BOS_STORAGE_CLASS]
```

获取文件下载URL

用户可以参考如下代码获取Object的URL：

```
 options = { 'expiration_in_seconds' => 360,
          'timestamp' => Time.now.to_i
}

puts client.generate_pre_signed_url(bucket_name, object_name, options)
###### 如果使用STS方式申请到的临时AK、SK、Token生成访问URL，需自行在http请求中携带指定header：<x-bce-security-token: sessionToken值>
```

> **说明：**
>
> - 用户在调用该函数前，需要手动设置endpoint为所属区域域名。百度智能云目前开放了多区域支持，请参考区域选择说明。目前支持“华北-北京”、“华南-广州”和“华东-苏州”三个区域。北京区域：`http://bj.bcebos.com`，广州区域：`http://gz.bcebos.com`，苏州区域：`http://su.bcebos.com`。
> - `EXPIRATION_IN_SECONDS`为指定的URL有效时长，时间从当前时间算起，为可选参数，不配置时系统默认值为1800秒。如果要设置为永久不失效的时间，可以将`expiration_in_seconds`参数设置为 -1，不可设置为其他负数。
> - `TIMESTAMP`为可选参数，不配置时，系统默认TIMESTAMP为当前时间。
> - 如果预期获取的文件时公共可读的，则对应URL链接可通过简单规则快速拼接获取：`http://bucketName.$region.bcebos.com/$bucket/$object`

列举存储空间中的文件

BOS SDK支持用户通过以下两种方式列举出object：

- 简单列举
- 通过参数复杂列举

除此之外，用户还可在列出文件的同时模拟文件夹

简单列举

当用户希望简单快速列举出所需的文件时，可通过listObjects方法获取Bucket中的Object列表。

```
client.list_objects(bucket_name)
```

> **注意：**
>
> 1. 默认情况下，如果Bucket中的Object数量大于1000，则只会返回1000个Object。
> 2. 若想增大返回Object的数目，可以使用Marker参数分次读取。

通过参数复杂列举

除上述简单列举外，用户还可通过`options`配置可选参数来实现各种灵活的查询功能。可设置的参数如下：

参数	功能	
PREFIX	限定返回的object key必须以prefix作为前缀	setPrefix(String prefix)
DELIMITER	是一个用于对Object名字进行分组的字符所有名字包含指定的前缀且第一次出现。Delimiter字符之间的Object作为一组元素: CommonPrefixes	
MARKER	设定结果从marker之后按字母排序的第一个开始返回	
MAX_KEYS	限定此次返回object的最大数，如果不设定，默认为100，max-keys取值不能大于1000	

> **注意：**
>
> 1. 如果有Object以Prefix命名，当仅使用Prefix查询时，返回的所有Key中仍会包含以Prefix命名的Object，详见递归列出目录下所有文件。
> 2. 如果有Object以Prefix命名，当使用Prefix和Delimiter组合查询时，返回的所有Key中会有Null，Key的名字不包含Prefix前缀，详见查看目录下的文件和子目录。

下面我们分别以几个案例说明通过参数列举的方法：

指定最大返回条数

```
 ###### 指定最大返回条数为500
options = {
          maxKeys: 500
}
puts client.list_objects(bucket_name, options)
```

返回指定前缀的object

```
 ###### 指定返回前缀为usr的object
options = {
          prefix: 'usr'
}
puts client.list_objects(bucket_name, options)
```

从指定Object后返回

```
###### 用户可以定义不包括某object，从其之后开始返回
options = {
        marker: 'object'
}
puts client.list_objects(bucket_name, options)
```

分页获取所有Object

用户可设置每页最多500条记录

```
options = {
        maxKeys: 500
}

is_truncated = true
while is_truncated
    res = client.list_objects(bucket_name, options)
    is_truncated = res['isTruncated']
    options[:marker] = res['nextMarker'] unless res['nextMarker'].nil?
end
```

分页获取所有特定Object后的结果

用户可设置每页最多500条记录，并从某特定object之后开始获取

```
options = {
        maxKeys: 5,
        marker: 'object'
}

is_truncated = true
while is_truncated
    res = client.list_objects(bucket_name, options)
    is_truncated = res['isTruncated']
    options[:marker] = res['nextMarker'] unless res['nextMarker'].nil?
end
```

listObjects方法返回的解析类中可供调用的参数有：

参数	说明
name	Bucket名称
prefix	匹配以prefix开始到第一次出现Delimiter字符之间的object作为一组元素返回
marker	本次查询的起点
maxKeys	请求返回的最大数目
isTruncated	指明是否所有查询都返回了；false-本次已经返回所有结果，true-本次还没有返回所有结果
contents	返回的一个Object的容器
+key	Object名称
+lastModified	此Object最后一次被修改的时间
+eTag	Object的HTTP协议实体标签
+storageClass	Object的存储形态
+size	Object的内容的大小（字节数）
+owner	Object对应Bucket所属用户信息
++id	Bucket Owner的用户ID
++displayName	Bucket Owner的名称

模拟文件夹功能

在BOS的存储结果中是没有文件夹这个概念的，所有元素都是以Object来存储，但BOS的用户在使用数据时往往需要以文件夹来管理文件。

因此，BOS提供了创建模拟文件夹的能力，其本质上来说是创建了一个size为0的Object。对于这个Object可以上传下载，只是控制台会对以"/"结尾的Object以文件夹的方式展示。

用户可以通过 Delimiter 和 Prefix 参数的配合模拟出文件夹功能。Delimiter 和 Prefix 的组合效果是这样的：

如果把 Prefix 设为某个文件夹名，就可以罗列以此 Prefix 开头的文件，即该文件夹下递归的所有的文件和子文件夹（目录）。文件名在Contents中显示。

如果再把 Delimiter 设置为 "/" 时，返回值就只罗列该文件夹下的文件和子文件夹（目录），该文件夹下的子文件名（目录）返回在 CommonPrefixes 部分，子文件夹下递归的文件和文件夹不被显示。

如下是几个应用方式：

列出Bucket内所有文件

当用户需要获取Bucket下的所有文件时，可以参考分页获取所有Object

递归列出目录下所有文件

可以通过设置 Prefix 参数来获取dir目录下所有的文件：

```
options = {
        prefix: 'dir/'
}

is_truncated = true
while is_truncated
    res = client.list_objects(bucket_name, options)
    is_truncated = res['isTruncated']
    options[:marker] = res['nextMarker'] unless res['nextMarker'].nil?
end
```

查看目录下的文件和子目录

在 Prefix 和 Delimiter 结合的情况下，可以列出dir目录下的文件和子目录：

```
 options = {
          prefix: 'dir/',
          delimiter: '/'
}

is_truncated = true
while is_truncated
   res = client.list_objects(bucket_name, options)
   is_truncated = res['isTruncated']
   options[:marker] = res['nextMarker'] unless res['nextMarker'].nil?
end
```

列举Bucket中Object的存储属性

当用户完成上传后，如果需要查看指定Bucket中的全部Object的storage class属性，可以通过如下代码实现：

```
 res = client.list_objects(bucket_name)

res['contents'].each { |obj| puts obj['storageClass'] }
```

Object权限控制

设置Object的访问权限

如下代码将Object的权限设置为了private：

```
client.set_object_canned_acl(bucket_name, object_name, Http::BCE_ACL => 'private')
```

关于权限的具体内容可以参考《BOS API文档 Object权限控制》。

设置指定用户对Object的访问权限

BOS提供set_object_acl方法和set_object_canned_acl方法来实现指定用户Object的访问权限设置，可以参考如下代码实现：

1. 通过set_object_canned_acl的x-bce-grant-read和x-bce-grant-full-control设置指定用户的访问权限

```
 id_permission = "id=\"8c47a952db4444c5a097b41be3f24c94\",id=\"bbbbbbbbbbbbbbbbbbbbbbbbbbbbbb\""
client.set_object_canned_acl(bucket_name, object_name, 'x-bce-grant-read' => id_permission)

id_permission = "id=\"8c47a952db4444c5a097b41be3f24c94\",id=\"bbbbbbbbbbbbbbbbbbbbbbbbbbbbbb\""
client.set_object_canned_acl(bucket_name, object_name, 'x-bce-grant-full-control' => id_permission)
```

2. 通过set_object_acl设置object访问权限

```
 acl = [{'grantee' => [{'id' => 'b124deeaf6f641c9ac27700b41a350a8'},
                       {'id' => 'aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa'}],
        'permission' => ['FULL_CONTROL']
}]
client.set_object_acl(bucket_name, object_name, acl)
```

> **注意：**
>
> 1. permission中的权限设置包含两个值：READ、FULL_CONTROL，它们分别对应相关权限。
> 2. 设置两个以上（含两个）被授权人时，请参考以上示例的格式，若将数组合并会返回报错。

查看Object的权限

如下代码可以查Object的权限：

```
client.get_object_acl(bucket_name, object_name)
```

get_object_acl方法返回的解析类中可供调用的参数有：

参数	说明
accessControlList	标识Object的权限列表
grantee	标识被授权人
-id	被授权人ID
permission	标识被授权人的权限

删除Object的权限

如下代码可以删除Object的权限：

```
client.delete_object_acl(bucket_name, object_name)
```

删除文件

可参考如下代码删除单个Object:

```
client.delete_object(bucket_name, object_name)
```

查看文件是否存在

用户可通过如下操作查看某文件是否存在：

```
begin
  client.get_object_meta_data(bucket_name, object_name)
rescue BceServerException => e
  puts "#{object_name} not exist!" if e.status_code == 404
end
```

获取及更新文件元信息

文件元信息(Object Metadata)，是对用户上传BOS的文件的属性描述，分为两种：HTTP标准属性（HTTP Headers）和User Meta（用户自定义元信息）。

获取文件元信息

参考只获取ObjectMetadata。

修改文件元信息

BOS修改Object的Metadata通过拷贝Object实现。即拷贝Object的时候，把目的Bucket设置为源Bucket，目的Object设置为源Object，并设置新的Metadata，通过拷贝自身实现修改Metadata的目的。如果不设置新的Metadata，则报错。

```
user_metadata = { "key1" => "value1" }
options = {
        'user-metadata' => user_metadata
}

client.copy_object(bucket_name, object_name, bucket_name, object_name, options)
puts client.get_object_meta_data(bucket_name, object_name)['user-metadata']['key1']
```

拷贝Object

拷贝一个文件

用户可以通过copyObject方法拷贝一个Object，如下代码所示：

```
client.copy_object(source_bucket_name, source_object_key, target_bucket_name, target_object_key)
```

`copy_object`方法可以通过`options`配置可选参数，参数列表参考如下:

参数	说明
user-metadata	用户自定义Meta，包含Key-Value对
eTag	Source Object的eTag，若选择上传，则会对Target Object和Source Object的eTag进行比对，若不相同，则返回错误。

同步Copy

当前BOS的CopyObject接口是通过同步方式实现的。同步方式下，BOS端会等待Copy实际完成才返回成功。同步Copy能帮助用户更准确的判断Copy状态，但用户感知的复制时间会变长，且复制时间和文件大小成正比。

同步Copy方式更符合业界常规，提升了与其它平台的兼容性。同步Copy方式还简化了BOS服务端的业务逻辑，提高了服务效率。

分块拷贝

除了通过CopyObject接口拷贝文件以外，BOS还提供了另外一种拷贝模式——Multipart Upload Copy。用户可以在如下的应用场景内（但不仅限于此），使用Multipart Upload Copy，如：

- 需要支持断点拷贝。
- 拷贝超过5GB大小的文件。
- 网络条件较差，和BOS的服务器之间的连接经常断开。

下面将介绍分步实现三步拷贝。

三步拷贝包含init、“拷贝分块”和complete三步，其中init和complete的操作同分块上传一致，可直接参考初始化Multipart Upload和完成分块上传。

```
left_size = client.get_object_meta_data(source_bucket_name, source_object_key)['content-length']
offset = 0
part_number = 1
part_list = []

while left_size > 0 do
  part_size = 5 * 1024 * 1024
  if left_size < part_size
    part_size = left_size
  end

  response = client.upload_part_copy(
    source_bucket_name, source_object_key, target_bucket_name, target_object_key, upload_id, part_number, part_size, offset)
  left_size -= part_size
  offset += part_size
  # your should store every part number and etag to invoke complete multi-upload
  part_list << {
    "partNumber" => part_number,
    "eTag" => response["eTag"]
  }
  part_number += 1
end
```

> **注意:** size参数以字节为单位，定义每个分块的大小，除最后一个Part以外，其他的Part大小都要大于5MB。

异常处理

BOS异常提示有如下四种方式：

异常方法	说明
BceHttpException	客户端异常
BceServerException	服务器异常
BceHttpException	net::http相关异常

用户可以使用rescue获取某个事件所产生的异常：

```
begin
  client.get_object_meta_data(bucket_name, object_name)
rescue Exception => e
  puts "Catch a http exception" if e.is_a?(BceHttpException)
  puts "Catch a client exception" if e.is_a?(BceClientException)
  puts "Catch a server exception" if e.is_a?(BceServerException)
end
```

客户端异常

客户端异常表示客户端尝试向BOS发送请求以及数据传输时遇到的异常。

服务端异常

当BOS服务端出现异常时，BOS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BOS错误信息格式

sdk日志

SDK日志

BOS Ruby SDK支持四个级别的日志(默认为Logger::INFO级别)，支持设置输出日志文件的目录，详细可以参考Log模块。示例代码：

```
##### 默认日志路径：DEFAULT_LOG_FILE = "./baidubce-sdk.log"
Log.set_log_file(file_path)

##### 四个日志级别：Logger::DEBUG | Logger::INFO | Logger::ERROR | Logger::FATAL
Log.set_log_level(Logger::DEBUG)
```

版本变更记录

- Ruby SDK开发包[2018-05-21] 版本号 0.9.0
  - 创建、查看、罗列、删除Bucket，获取位置和判断是否存在
  - 支持管理Bucket的生命周期、日志、ACL、存储类型
  - 上传、下载、删除、罗列Object，支持追加上传、分块上传、分块拷贝

C++-SDK

概述

本文档主要介绍BOS C++ SDK的安装和使用。在使用本文档前，您需要先了解BOS的一些基本知识，并已经开通了BOS服务。若您还不了解BOS，可以参考产品描述。

安装SDK工具包

运行环境

依赖开源的第三方库，包括curl、openssl、uuid、jsoncpp，其中jsoncpp可在thirdlib文件夹下编译并安装。

SDK目录结构

```
├──example // 使用示例
├──include // 头文件
│  └──bcesdk
│     ├──auth       // BCE签名相关类
│     ├──bos        // SDK服务相关类
│     │  ├──model    // BCE公用model类
│     │  ├──request  // SDK请求类
│     │  └──response // SDK响应类
│     ├──common     // 日志相关和文件工具类，如文件输入输出流
│     ├──http       // BCE的HTTP通信相关类
│     ├──model      // BOS内部model，如request或response
│     └──util       // BCE公用工具类
├──msvc             // windows viusal studio相关
├──src              // 源码路径
│  ├──auth
│  ├──bench
│  ├──bos
│  │  ├──model
│  │  ├──request
│  │  └──response
│  ├──http
│  ├──model
│  └──util
├──thirdlib // 第三方库路径
   └──json  // linux下json依赖库路径
```

安装SDK

在官网下载BOS C++ SDK工具包。

Linux Centos用户安装指南

依赖公共库安装:

```
 sudo yum install gcc-c++
sudo yum install autoconf
sudo yum install automake
sudo yum install libuuid-devel
sudo yum install openssl-devel
sudo yum install libcurl-devel
sudo yum install cmake
```

执行以下命令即可在thirdlib中生成jsoncpp库:

```
 cd thirdlib/json
cmake ./jsoncpp-1.9.0/
make
make install
```

基于autotools环境构建

1. 执行`sh autogen.sh`生成configure；
2. 执行`configure`，添加必要的选项，比如`--prefix`或者`CXXFLAGS`等；
3. 执行`make & make install`，将在对应路径生成include和lib。

编译引用

1. 如果没有安装到gcc默认路径下，则在CXXFLAGS里使用-L指定lib路径，使用-I指定include路径；
2. 在LDFLAGS里指定-lbossdk。
3. cpp需要std-11及其以上环境, 可使用CC=或CXX=来指定高版本gcc或g++编译器。

如果直接写编译命令的，可也以直接指定`-L -I -l`。

Linux Ubuntu & Debian用户安装指南

依赖公共库安装:

```
 sudo apt-get -y install build-essential
sudo apt-get install autoconf
sudo apt-get install uuid-dev
sudo apt-get install libssl-dev
sudo apt-get install libcurl4-openssl-dev
sudo apt-get install cmake
```

执行以下命令即可在thirdlib中生成jsoncpp库:

```
 cd thirdlib/json/
cmake ./jsoncpp-1.9.0/
make
make install
```

基于autotools环境构建

1. 执行`sh autogen.sh`生成configure文件；
2. 执行`configure`，添加必要的选项，比如`--prefix`或者`CXXFLAGS`等；
3. 执行`make & make install`，将在对应路径生成include和lib。

编译引用

1. 如果没有安装到gcc默认路径下，则在CXXFLAGS里使用-L指定lib路径，使用-I指定include路径；
2. 在LDFLAGS里指定-lbossdk。
3. cpp需要std-11及其以上环境, 可使用`CC=`或`CXX=`来指定高版本gcc或g++编译器。

如果直接写编译命令的，可也以直接指定`-L -I -l`。

Windows用户安装指南

环境准备

1. 安装Visual studio 2012或更高版本。
2. 安装CMake3.1或以上。
3. thirdlib已经包含所需要三方库（curl、jsoncpp），如需要其它版本，请到官方网站下载。

快速体验

1. 解压thirdlib目录下的`windows_dependency.zip`到`thirdlib`下。
2. 双击`example\\example.sln`

编译使用

1. 解压`thirdlib`目录下的`windows_dependency.zip`到`thirdlib`下
2. 创建build目录

```
  mkdir build
cd build
```

3. 生成目标`sln cmake -G "\\<type>" ..`这里要根据所安装的Visual studio 版本和编译平台，而选择不同的`\<type>`。

   生成Visual Studio 2015的sln： `cmake -G "Visual Studio 14 2015" ..` 生成Visual Studio 2013编译目标是64位的sln： `cmake -G "Visual Studio 12 2013 Win64"`

参数详见CMake帮助文档。

4. 编译： 直接使用cmake 编译 cmake --build .或手动打开bossdk.sln。

5. 结果生成： 编译结果将生成在代码根目录的output文件夹下。结构如下：

```
output
├──── include
│    └──── bcesdk
│        ├──── auth
│        ├──── bos
│        │    ├──── model
│        │    ├──── request
│        │    └──── response
│        ├──── common
│        ├──── http
│        ├──── model
│        └──── util
└──── lib
    ├──── x64
    │    ├──── Debug
    │    └──── Release
    └──── x86
        ├──── Debug
        └──── Release
```

6. 使用SDK

编译：把output/include添加include目录下

链接：主动链接bossdk.lib，jsoncpp.lib，LIBCURL.LIB 执行：把LIBCURL.DLL放到目标exe同一个目录下

低于Visual studio 2012 的编译问题

1. 低于Visual studio 2005将存在大量问题，强烈不建议使用。

2. 必须使用windows SDK 7.1或以上版本。

3. 需要解决stdint.h不存在问题和替换少量的CRT函数如：strtoull，strtoll等。

卸载SDK

预期卸载SDK时，删除output路径。

初始化

确定Endpoint

在确认您使用SDK时配置的Endpoint时，可先阅读开发人员指南中关于BOS访问域名的部分，理解Endpoint相关的概念。百度智能云目前开放了多区域支持，请参考区域选择说明。

region与endpoint参考如下链接

https://cloud.baidu.com/doc/BOS/s/akrqd2wcx

常见示例如下:

访问区域	对应Endpoint
BJ	bj.bcebos.com
BD	bd.bcebos.com
SU	su.bcebos.com
GZ	gz.bcebos.com
CD	cd.bcebos.com
HKG	hkg.bcebos.com
FWH	fwh.bcebos.com
FSH	fsh.bcebos.com

获取密钥

要使用百度智能云BOS，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BOS做签名验证。 可以通过如下步骤获得并了解您的AK/SK信息：

注册百度智能云账号

创建AK/SK

新建BosClient

BosClient是BOS服务的客户端，为开发者与BOS服务进行交互提供了一系列的方法。

使用AK/SK新建BosClient

通过AK/SK方式访问BOS，用户可以参考如下代码新建一个BosClient：

```
#### include <bcesdk/bos/client.h>
using namespace bcesdk_ns;
int main() {
    std::string ACCESS_KEY_ID = <your-access-key-id>;  // 用户的Access Key ID
    std::string SECRET_ACCESS_KEY = <your-secret-access-key>; // 用户的Secret Access Key

    /* 初始化一个Client*/

    //设置日志
    FILE *logfp = fopen("sdk.log", "w");
    sdk_set_log_stream(logfp);
    sdk_set_log_level(SDK_LOG_DEBUG);
    //设置客户端
    Client client(ACCESS_KEY_ID, SECRET_ACCESS_KEY);
}
```

在上面代码中，ACCESS_KEY_ID对应控制台中的“Access Key ID”，SECRET_ACCESS_KEY对应控制台中的“Access Key Secret”，获取方式请参考《操作指南 管理ACCESSKEY》。

上面的方式使用默认域名作为BOS的服务地址，如果用户需要自己指定域名，可以通过传入ENDPOINT参数来指定。

```
std::string ACCESS_KEY_ID = <your-access-key-id>;  // 用户的Access Key ID
std::string SECRET_ACCESS_KEY = <your-secret-access-key>; // 用户的Secret Access Key
std::string ENDPOINT = <domain-name>;  // 用户自己指定的域名


config.endpoint = ENDPOINT;
Client client(ACCESS_KEY_ID, SECRET_ACCESS_KEY, config);
```

> **注意：**ENDPOINT参数只能用指定的包含区域的域名来进行定义，不指定时默认为北京区域http://bj.bcebos.com。

使用STS创建BOS Client

申请STS token BOS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。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BOS，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智能云STS使用介绍。

用STS token新建BOS Client 申请好STS后，可将STS Token配置到BOS Client中，从而实现通过STS Token创建BOS Client。 **代码示例** 使用申请好的STS Token创建BOS Client对象。

```
#### include <bcesdk/bos/client.h>
using namespace bcesdk_ns;
int main() {
  //设置日志
   FILE *logfp = fopen("sdk.log", "w");
   sdk_set_log_stream(logfp);
   sdk_set_log_level(SDK_LOG_DEBUG);

   std::string ak = <access-key-id>;  // 用于STS凭证访问的AK
   std::string sk = <secret-access-key>; // 用于STS凭证访问的SK
   std::string sts_token = <session_token>;//SessionToken,使用STS凭证访问时必须携带
  //使用自定义选项
   ClientOptions option;
  option.endpoint = <domain-name>;
  //配置STS客户端
   Client client(Credential(ak, sk, sts_token), option);
}
```

配置HTTPS协议访问BOS

BOS支持HTTPS传输协议，您可以通过如下两种方式在BOS C++ SDK中使用HTTPS访问BOS服务：

在endpoint中指明https:

```
std::string endpoint = "https://bj.bcebos.com";
std::string ak = "ak";
std::string sk = "sk";
ClientOptions config;
config.endpoint = endpoint;
Client client(ak, sk, config);
```

配置BosClient

如果用户需要配置Client的一些细节的参数，可以在构造Client的时候传入ClientOptions对象。 ClientOptions是BOS服务的配置类，可以为客户端配置超时、重试次数等。

设置网络参数

用户可以用对ClientOptions基本网络参数进行设置：

```
ClientOptions config;

// 设置HTTP请求超时为120秒
config.timeout = 120;

// 设置额外重试次数
config.retry = 2;

// 设置并发请求数
config.max_parallel = 10;
```

参数说明

通过ClientOptions能指定的所有参数如下表所示：

参数	默认值	说明
endpoint	http://bj.bcebos.com	HTTP请求终端
retry	2	HTTP请求额外的重试次数
timeout	120	是秒为单位的超时，这个要根据请求类型慎重选取，比如下载一个大文件可能用到比较大的超时
connect_timeout_ms	10000	是毫秒为单位的http连接超时，默认值为10秒
user_agent	cppsdk相关	可以指定请求发送的时候的UA
max_parallel	10	并发请求接口的默认并发数（也可以直接在调用接口时参数指定）
multi_part_size	10MB	功能性函数里封装三步上传或者并发下载的块大小
sign_expire_seconds	1200s	指定签名超时的时间
calculate_md5_on	false	指定发送请求时是否需要计算md5
cname_enabled	false	指定发送请求时是否使用自定义域名，开启前需先申请开通bucket的自定义域名
auto_cname_enabled	true	默认开启，使用virtual host的四级域名来访问BOS否则GET请求会遇到403报错
proxy_protocol	1200s	指定签名超时的时间
client_enable_cancel	false	指定BosClient发送的请求能否被其他线程主动取消，开启后同时需要在希望被取消的request里设置取消

Bucket管理

概述

Bucket既是BOS上的命名空间，也是计费、权限控制、日志记录等高级功能的管理实体。

- Bucket名称在所有区域中具有全局唯一性，且不能修改。

> **说明：** 百度智能云目前开放了多区域支持，请参考区域选择说明。
> 目前支持“华北-北京”、“华南-广州”和“华东-苏州”三个区域。北京区域：`http://bj.bcebos.com`，广州区域：`http://gz.bcebos.com`，苏州区域：`http://su.bcebos.com`。

- 存储在BOS上的每个Object都必须包含在一个Bucket中。
- 一个用户最多可创建100个Bucket，但每个Bucket中存放的Object的数量和大小总和没有限制，用户不需要考虑数据的可扩展性。

新建Bucket

新建Bucket

如下代码可以新建一个Bucket：

```
int createBucket (Client& client, const std::string& bucketName) {
    // 新建一个Bucket
    PutBucketRequest request(bucketName);
    PutBucketResponse response;
    int ret = client.put_bucket(request, &response);
    if (ret !=0) {
        return ret;
    }
    if (response.is_fail()) {
        printf("error-message:%s\n", response.error().message().c_str());
    }
    return ret;
}
int main() {
    std::sting ak = "ak";
    std::string sk = "sk";
    Client client(ak, sk);
    int ret = createBucket(client, "bucketName");
    std::string ret_info = stringfy_ret_code(ret);
    std::cout << ret_info << std::endl;
}
```

> **注意：** 由于Bucket的名称在所有区域中是唯一的，所以需要保证bucketName不与其他所有区域上的Bucket名称相同。
> Bucket的命名有以下规范：
>
> - 只能包括小写字母，数字，短横线（-）。
> - 必须以小写字母或者数字开头。
> - 长度必须在4-63字节之间。

通过上述代码创建的bucket，权限是私有读写，存储类型是标准类型（Standard）。

列举Bucket

列举Bucket

如下代码可以列出用户所有的Bucket：

```
int listBuckets (Client& client) {
  // 获取用户的Bucket列表
  ListBucketsRequest listBucketsRequest;
  ListBucketsResponse listBucketsResponse;
  int ret = client.list_buckets(listBucketsRequest, &listBucketsResponse);
  std::vector<BucketSummary> bucketSummaryList = listBucketsResponse.buckets();
  if (listBucketsResponse.is_fail()) {
      printf("error-message:%s\n", listBucketsResponse.error().message().c_str());
      return ret;
  }
  // 遍历Bucket
  for(const BucketSummary& bs : bucketSummaryList){
      std::cout << "bucketName: " << bs.name <<  std::endl;
  }
  return ret;
}
```

查看Bucket所属的区域

查看Bucket所属的区域

Bucket Location即Bucket Region，百度智能云支持的各region详细信息可参见区域选择说明。

如下代码可以获取该Bucket的Location信息：

```
 Client client (ak, sk, config);

ListBucketsRequest listBucketsRequest;
ListBucketsResponse listBucketsResponse;
int ret = client.list_buckets(listBucketsRequest, &listBucketsResponse)
std::vector<BucketSummary> bucketSummaryList =  listBucketsResponse.buckets();
if (listBucketsResponse.is_fail()) {
      printf("error-message:%s\n", listBucketsResponse.error().message().c_str());
      return ;
}
for(const BucketSummary& bs : bucketSummaryList){
    std::cout << "name: " << bs.name <<
      " location: " << bs.location << std::endl;
}

GetBucketLocationRequest getBucketLocationRequest("bucketName");
GetBucketLocationResponse getBucketLocationResponse;
ret = client.get_bucket_location(getBucketLocationRequest,
                  &getBucketLocationResponse);
if (getBucketLocationResponse.is_fail()) {
      printf("error-message:%s\n", response.error().message().c_str());
      return;
}
std::cout << "location: " << getBucketLocationResponse.location() << std::endl;
```

判断Bucket是否存在

判断Bucket是否存在

若用户需要判断某个Bucket是否存在，则如下代码可以做到：

```
 //0 表示check成功，bucket存在
//-1不存在
//other其他错误，具体error-message给出
int checkBucketExist (Client& client, const std::string& bucketName) {
    bool exist = false;
    // 获取Bucket的存在信息
    HeadBucketRequest request(bucketName);
    HeadBucketResponse response;
    int ret = client.head_bucket(request, &response);
    if (response.is_fail() && response.status_code() != 404){
        printf("error-message:%s\n", response.error().message().c_str());
        return ret;
    } else if (response.status_code() == 404){
        return -1;
    }
    //200 ok
    return 0;
}
```

> **注意：** 如果Bucket不为空（即Bucket中有Object存在），则Bucket无法被删除，必须清空Bucket后才能成功删除。

Bucket权限管理

Bucket权限管理 目前BOS支持两种方式设置ACL. 具体可参考权限控制

第一种是使用Canned Acl，在put_object_acl的时候，通过头域的"x-bce-acl", "x-bce-grant-read", 或者 "x-bce-grant-permission"来设置对象的访问权限，当前可设置的权限包括private和public-read，三种类型的header不可以同时在一个请求中出现.

第二种方式是以自定义Acl样式, 具体可通过上传其json字符串, 设置access_control_list结构体, 或者直接上传ACL文件. 具体可参考权限控制概述

查看Bucket的权限

如下代码可以查看Bucket的权限：

```
 GetBucketAclRequest getBucketAclRequest("bucketName");
GetBucketAclResponse getBucketAclResponse;
int ret = client.get_bucket_acl(getBucketAclRequest, &getBucketAclResponse);
if (ret) {
    LOGF(WARN, "client err: %d", ret);
}
if (getBucketAclResponse.is_fail()) {
    LOGF(WARN,"get_bucket_acl: [status_code = %d], [message = %s], [requestid = %s]",
        getBucketAclResponse.status_code(),
        getBucketAclResponse.error().message().c_str(),
        getBucketAclResponse.error().request_id().c_str());
}

//获取ownerid
std::string owner_id = getBucketAclResponse.owner().id;

//获取acl结构数据
std::vector<Grant> aclData = getBucketAclResponse.access_control_list();
std::string jsonStr = getBucketAclResponse.json_access_control_list();
```

get_bucket_acl方法返回的response中可供调用的参数有：

参数	类型	说明
owner	Owner(string)	Bucket owner id
access_control_list	vector<Grant>	保存acl的容器

Grant所有字段含义，具体可参考权限控制概述

> **注意：**
>
> - 若您使用STS进行访问控制，C++ SDK仅支持传入STS， 不支持获取。

设置Bucket的访问权限

设置Canned ACL 如下代码将Bucket的权限设置为了private.

```
 PutBucketAclRequest putBucketAclRequest(bucketName);
PutBucketAclResponse putBucketAclResponse;
putBucketAclRequest.set_canned_acl("private");

int ret = client.put_bucket_acl(putBucketAclRequest, &putBucketAclResponse);
if (ret) {
    LOGF(WARN, "client err: %d", ret);
}
if (putBucketAclResponse.is_fail()) {
    LOGF(WARN,"put_bucket_acl: [status_code = %d], [message = %s], [requestid = %s]",
        putBucketAclResponse.status_code(),
        putBucketAclResponse.error().message().c_str(),
        putBucketAclResponse.error().request_id().c_str());
}
```

CannedACL包含三个值： `private` 、 `public-read` 、 `public-read-write` ，它们分别对应相关权限。
具体内容可以参考BOS API文档 使用CannedAcl方式的权限控制。

设置自定义ACL 用户可参考如下代码设置Bucket的自定义访问权限，支持三种不同参数：

```
 PutBucketAclRequest putBucketAclRequest(bucketName);
PutBucketAclResponse putBucketAclResponse;

// 1. 通过上传acl json串
std::string jsonAcl =
            "{\"accessControlList\":[{\"grantee\":[{\"id\":\"*\"}],\"permission\":[\"READ\"]},{"
            "\"grantee\":[{\"id\":\"cb5f8xxxxxxxxx82bbc\"}],\"permission\":["
            "\"FULL_CONTROL\"]}]}";
std::string cannedAcl="public-read";
putBucketAclRequest.set_json_acl(jsonAcl);


// 2. 上传acl文件
std::string aclFilePath = "/tmp/acl.json"
int setRet = putBucketAclRequest.set_acl_file(aclFilePath);
if (ret) {
    LOGF(WARN, "client set_acl_file: %d", ret);
}

// 3. 通过设置access_control_list数据
std::vector<Grant> grants;
Grant grant;
grantee.id = "77fxxxxxxxxxx5fa406";
grant.grantee.push_back(grantee);
grant.permission.push_back("READ");
grants.push_back(grant);
putBucketAclRequest.set_access_control_list(grants);

int ret = client.put_bucket_acl(putObjectAclRequest, &putBucketAclRequest);
if (ret) {
    LOGF(WARN, "client err: %d", ret);
}
if (putBucketAclRequest.is_fail()) {
    LOGF(WARN,"put_bucket_acl: [status_code = %d], [message = %s], [requestid = %s]",
        putBucketAclRequest.status_code(),
        putBucketAclRequest.error().message().c_str(),
        putBucketAclRequest.error().request_id().c_str());
}
```

删除Bucket

删除Bucket

如下代码可以删除一个Bucket：

```
int deleteBucket (Client& client, const std::string& bucketName) {
    // 删除Bucket
    DeleteBucketRequest request(bucketName);
    DeleteBucketResponse response;
    int ret = client.delete_bucket(request, &response);
    if (response.is_fail()) {
        printf("error-message:%s\n", listBucketsResponse.error().message().c_str());
    }
    return ret;
}
```

> **注意：**
>
> - 在删除前需要保证此Bucket下的所有Object和未完成的三步上传Part已经被删除，否则会删除失败。
> - 在删除前确认该Bucket没有开通跨区域复制，不是跨区域复制规则中的源Bucket或目标Bucket，否则不能删除。

文件管理

上传文件

上传文件

在BOS中，用户操作的基本数据单元是Object。Object包含Key、Meta和Data。其中，Key是Object的名字；Meta是用户对该Object的描述，由一系列Name-Value对组成；Data是Object的数据。

BOS C++ SDK提供了丰富的文件上传接口，可以通过以下方式上传文件：

- 简单上传
- 追加上传
- 分块上传
- 断点续传上传

简单上传

BOS在简单上传的场景中，支持以指定文件形式、以数据流方式、以文件描述符方式、以字符串方式执行Object上传，请参考如下代码：

```
int PutObjectDemo(Client& client,const std::string& bucketName, const std::string objectKey){
  // 获取文件数据流
  FileInputStream inputStream("/path/to/test.zip"); // bcesdk/util/util.h
int ret = 0;
  // 以文件名作为参数上传Object
  ret = client.upload_file(bucketName, objectKey, "/path/to/test.zip");
  // 以数据流形式上传Object
  ret = client.upload_file(bucketName, objectKey, inputStream);
  // 以文件描述符形式上传Object
  fd_t fd = open("/path/to/test.zip", O_RDWR, 0666);//linux下，fd_t在common.h中定义
  ret = client.upload_file(bucketName, objectKey, fd);
  // 以字符串形式上传Object
  std::string data = "this is data";
ret = client.put_object(bucketName, objectKey, data);
  return ret;
}
```

Object以文件的形式上传到BOS中，put_object、upload_file函数支持不超过5GB的Object上传。若要支持大文件（大于5G的文件）上传请使用upload_super_file，参考代码如下:

```
int PutLargeObjectDemo(Client& client,const std::string& bucketName, const std::string objectKey){
  std::string fileName = "/path/to/test.zip"
  return client.upload_super_file(bucketName, objectKey, fileName);//第三个参数用fd_t形式也可行
}
```

设置文件元信息

文件元信息(Object Meta)，是对用户在向BOS上传文件时，同时对文件进行的属性描述，主要分为分为两种：设置HTTP标准属性（HTTP Headers）和用户自定义的元信息。

- **设定Object的Http Header**

BOS C++ SDK本质上是调用后台的HTTP接口，因此用户可以在上传文件时自定义Object的Http Header。常用的http header说明如下：

名称	描述	默认值
Content-MD5	文件数据校验，设置后BOS会启用文件内容MD5校验，把您提供的MD5与文件的MD5比较，不一致会抛出错误	无
Content-Type	文件的MIME，定义文件的类型及网页编码，决定浏览器将以什么形式、什么编码读取文件。如没有指，BOS则根据文件的扩展名自动生成，如文件没有扩展名则填默认值	application/octet-stream
Content-Disposition	指示MIME用户代理如何显示附加的文件，打开或下载，及文件名称	无
Content-Length	上传的文件的长度，超过流/文件的长度会截断，不足为实际值	流/文件时间长度
Expires	缓存过期时间	无
Cache-Control	指定该Object被下载时的网页的缓存行为	无

参考代码如下：

```
...
// 初始化meta
ObjectMetaData meta;

// 设置ContentType
meta.set_content_type("application/json");

// 设置cache-control
meta.set_cache_control("no-cache");

// 设置x-bce-storage-class
meta.set_storage_class("STANDARD");

ret = client.upload_file(bucketName, objectKey, content, meta);
...
```

- **用户自定义元信息**

BOS支持用户自定义元数据来对Object进行描述。如下代码所示：

```
// 设置自定义元数据name的值为my-data
meta.set_user_meta("name", "my-data");

// 上传Object
ret = client.upload_file(bucketName, objectKey, file_name, meta);
```

> **提示：**
>
> - 在上面代码中，用户自定义了一个名字为`name`，值为`my-data`的元数据
> - 当用户下载此Object的时候，此元数据也可以一并得到
> - 一个Object可以有多个类似的参数，但所有的User Meta总大小不能超过2KB

设置Object的Copy属性

BOS同时会提供CopyObject接口用于将一个已经存在的Object拷贝到另外一个Object，拷贝过程中会对源Object的Etag或修改状态进行判断，根据判断结果决定是否执行拷贝。详细的参数解释如下：

名称	类型	描述	是否必需
x-bce-copy-source-if-match	std::string	如果源Object的ETag值和用户提供的ETag相等，则执行拷贝操作，否则拷贝失败。	否
x-bce-copy-source-if-none-match	std::string	如果源Object的ETag和用户提供的ETag不相等，则执行拷贝操作，否则拷贝失败。	否
x-bce-copy-source-if-unmodified-since	std::string	如果源object在x-bce-copy-source-if-unmodified-since之后没被修改，则执行拷贝操作，否则拷贝失败。	否
x-bce-copy-source-if-modified-since	std::string	如果源object在x-bce-copy-source-if-modified-since之后被修改了，则执行拷贝操作，否则拷贝失败。	否

对应的示例代码：

```
 // 初始化BosClient
Client client = ...;

// 创建CopyObjectRequest对象
CopyObjectRequest copyObjectRequest(destBucketName, destKey, srcBucketName, srcKey);
CopyObjectResponse copyObjectResponse；

// 设置新的Metadata
StringMap& userMetadata = *(meta.mutable_user_meta());//StringMap == map<string, string>
userMetadata.clear();
userMetadata["<user-meta-key>"] = "<user-meta-value>";

copyObjectRequest.set_meta(&meta, false);//第二个参数(is_own)若为true则由copyObjectRequest析构时delete meta
//copy-source-if-match
copyObjectRequest.set_if_match("111111111183bf192b57a4afc76fa632");
//copy-source-if-none-match
copyObjectRequest.set_if_none_match("111111111183bf192b57a4afc76fa632");

std::string gmtDate = TimeUtil::now_gmttime();//当前GMT格式时间

//copy-source-if-modified-since
copyObjectRequest.set_if_modified_since(gmtDate);

//copy-source-if-unmodified-since
copyObjectRequest.set_if_unmodified_since(gmtDate);

// 复制Object
client.copy_object(copyObjectRequest， copyObjectResponse；);

std::cout << "ETag: " << copyObjectResponse.etag() << " LastModified: " <<      copyObjectResponse.last_modified() << std::endl;
```

上传Object时设置存储类型

BOS支持标准存储, 低频存储和冷存储，上传Object并存储为低频存储类型通过指定StorageClass实现，四种存储类型对应的参数如下：

存储类型	参数
标准存储	STANDARD
低频存储	STANDARD_IA
冷存储	COLD
归档存储	ARCHIVE

以低频存储为例，代码如下：

```
 void print_common_response(BceResponse &result) {
    printf("status:%d\n", result.status_code());
    if (result.is_ok()) {
        printf("request-id:%s\n", result.request_id().c_str());
        printf("debug-id:%s\n", result.debug_id().c_str());
    }
    if (result.is_fail()) {
        printf("error-message:%s\n", result.error().message().c_str());
    }
}
int putObjectStorageClass(){
    std::string filename = "file.txt";
    FileInputStream file(filename);
    PutObjectRequest request(bucket, object, &file);
    request.mutable_meta()->set_storage_class("STANDARD_IA");
    PutObjectResponse result;
    client.put_object(request, &result);
    print_common_response(result);
    printf("etag: %s\n", result.etag().c_str());
}
```

使用上传进度条

```
 // 上传进度回调函数
// 注意该回调函数中不得出现耗时较长/阻塞操作, 会影响数据上传性能.
// increment: 当次上传的数据量
// transfered: 已上传数据量
// total: 需上传的数据总量
// userData: 用户自定义数据, 例如object bucket+key等.
void progress_callback(int64_t increment, int64_t transfered, int64_t total, void* user_data) {
    std::cout << "progress_callback[" << user_data << "] => " <<
    increment <<" ," << transfered << "," << total << std::endl;
}

//待上传文件
std::string filename = "/tmp/put_file_test";
FileInputStream file(filename);

PutObjectRequest req(BUCKET, "transfer_progress_t1", &file);
PutObjectResponse rsp;

// 设置上传进度相关数据
// TransferProgress结构在头文件: "bcesdk/common/common.h"
TransferProgress progress;
progress.transfer_progress_cb = progress_callback;
req.set_progress(progress);

//将filename文件中数据上传
int ret = client()->put_object(req, &rsp);
if (ret) {
    LOGF(WARN, "client err: %d", ret);
}
if (rsp.is_fail()) {
    LOGF(WARN,
        "put_object: [status_code = %d], [message = %s], [requestid = %s]",
        rsp.status_code(),
        rsp.error().message().c_str(),
        rsp.error().request_id().c_str());
}
```

追加上传

上文介绍的简单上传方式，创建的Object都是Normal类型，用户不可再进行追加写，这在日志、视频监控、视频直播等数据复写较频繁的场景中使用不方便。

正因如此，百度智能云BOS特别支持了AppendObject，即以追加写的方式上传文件。通过AppendObject操作创建的Object类型为Appendable Object，可以对该Object追加数据。AppendObject大小限制为0~5G。

通过AppendObject方式上传示例代码如下：

```
int AppendObjectDemo(Client& client,const std::string& bucketName, const std::string& objectKey) {

    // 获取数据流
    FileInputStream inputStream("/path/to/test.zip");

    // 以数据流形式上传Object
    AppendObjectRequest appendObjectFromInputStreamRequest(bucketName, objectKey, &inputStream);
    AppendObjectResponse appendObjectFromInputStreamResponse;
    int ret = client.append_object(appendObjectFromInputStreamRequest, &appendObjectFromInputStreamResponse);

    // 以字符串上传Object
    std::string data = "this is data";
    AppendObjectRequest appendObjecFromStringtRequest(bucketName, objectKey, data);
    AppendObjectResponse appendObjectFromStringResponse;

    ret = client.append_object(appendObjecFromStringtRequest, &appendObjectFromStringResponse);

    // 打印ETag
    std::cout << appendObjectFromInputStreamResponse.etag() << std::endl;
    // 打印NextAppendOffset
    std::cout << appendObjectFromInputStreamResponse.next_append_offset() << std::endl;


    // 追加上传的示例，需要在请求中加上下次追加写的位置
    long long nextAppendOffset = appendObjectFromInputStreamResponse.next_append_offset();
    AppendObjectRequest appendObjectFromStringRequest(bucketName,objectKey,data);
    appendObjectFromStringRequest.set_offset(nextAppendOffset);
    AppendObjectResponse appendObjectFromStringResponse;
    ret = client.append_object(appendObjectFromStringRequest, &appendObjectFromStringResponse);
    return ret;
}
```

分块上传

除了通过简单上传及追加上传方式将文上传件到BOS以外，BOS还提供了另外一种上传模式 —— Multipart Upload。用户可以在如下的应用场景内（但不仅限于此），使用Multipart Upload上传模式，如：

- 需要支持断点上传。
- 上传超过5GB大小的文件。
- 网络条件较差，和BOS的服务器之间的连接经常断开。
- 需要流式地上传文件。
- 上传文件之前，无法确定上传文件的大小。

下面将一步步介绍Multipart Upload的实现。假设有一个文件，本地路径为 `/path/to/file.zip` ，由于文件比较大，将其分块传输到BOS中。

初始化Multipart Upload

使用 `initiateMultipartUpload` 方法来初始化一个分块上传事件：

```
 // 开始Multipart Upload
InitMultiUploadRequest initMultiUploadRequest(bucketName, objectKey);
InitMultiUploadResponse initMultiUploadResponse;
int ret = client.init_multipart_upload(initMultiUploadRequest, &initMultiUploadResponse);
//异常处理
...
// 打印UploadId
std::cout << "UploadId: " << initMultiUploadResponse.upload_id() << std::endl;
```

initMultiUploadResponse 的返回结果中含有 UploadId ，它是区分分块上传事件的唯一标识，在后面的操作中，我们将用到它。

- **上传低频存储类型Object的初始化**

初始化低频存储的一个分块上传事件：

```
 void putMultiUploadStorageClass(){
    ObjectMetaData meta;
    meta.set_storage_class("STANDARD_IA");
    InitMultiUploadRequest initMultiUploadRequest(bucketName, objectKey);
    InitMultiUploadResponse initMultiUploadResponse;
    initMultiUploadRequest.set_meta(&meta);
    client.init_multipart_upload(initMultiUploadRequest, &initMultiUploadResponse);
}
```

- **上传冷存储类型Object的初始化**

初始化低频存储的一个分块上传事件：

```
 void putMultiUploadStorageClass(){
    ObjectMetaData meta;
    meta.set_storage_class("COLD)");
    InitMultiUploadRequest initMultiUploadRequest(bucketName, objectKey);
    InitMultiUploadResponse initMultiUploadResponse;
    initMultiUploadRequest.set_meta(&meta);
    client.init_multipart_upload(initMultiUploadRequest, &initMultiUploadResponse);
}
```

上传分块

接着，把文件分块上传。

```
 // 设置每块为 5MB
// [注意] 除了最后一个分块, 其余分块需满足size>=100kb
long partSize = 1024 * 1024 * 5L;

// 注意: 当分块上传的数据为string/内存数据, UploadPartRequest的构造函数如下:
// UploadPartRequest(const std::string &bucket_name, const std::string &object_name,  const std::string &data, int part_number, const std::string &upload_id)
// 其中data字段为std::string, 不得传入C风格的char*串, 会导致计算数据size错误.

//待分块上传文件
std::string partFileName = "/path/to/file.zip";
FileInputStream file(partFileName);

// 计算分块数目
int partCount = static_cast<int>(file.get_size() / partSize);
if (file.get_size() % partSize != 0){
    partCount++;
}

int64_t size = file.size();
int64_t off = 0;
std::vector<part_t> partEtags;
for (int i = 0; off < file.size(); ++i) {
    if (off + partSize > size) {
        partSize = size - off;
    }
    FileInputStream partFile(file.fd(), off, partSize);
    UploadPartRequest uploadPartRequest(bucketName, objectName, partFile, i + 1, initMultiUploadResponse.upload_id());
    UploadPartResponse uploadPartResponse;

    int ret = client.upload_part(uploadPartRequest, &uploadPartResponse);
    //校验返回值

    // 将返回的PartETag保存到List中。
    part_t partInfo;
    partInfo.part_number = i+1;
    partInfo.etag = uploadPartResponse.etag();
    partEtags.push_back(partInfo);

    off += partSize;
}
```

上面代码的核心是调用 upload_part 方法来并发的上传每一个分块，但是要注意以下几点：

- upload_part 要求除最后一个Part以外，其他的Part大小都要大于等于100KB。但是Upload Part接口并不会立即校验上传Part的大小；只有当Complete Multipart Upload的时候才会校验，若upload_part流程中的块大小不符合预期，则complete_multipart_upload接口会报错。
- 为了保证数据在网络传输过程中不出现错误，建议您在upload_part后，使用每个分块BOS返回的Content-MD5值分别验证已上传分块数据的正确性。当所有分块数据合成一个Object后，不再含MD5值。
- Part号码的范围是1~10000。如果超出这个范围，BOS将返回InvalidArgument的错误码。
- 每次上传Part时都要把流定位到此次上传块开头所对应的位置。
- 每次上传Part之后，BOS的返回结果会包含一个 ETag 对象，它是上传块的ETag与块编号（PartNumber）的组合，在后续完成分块上传的步骤中会用到它，因此需要将其保存起来。一般来讲这些 ETag 对象将被保存到vector中。

完成分块上传

如下代码所示，完成分块上传：

```
 CompleteMultipartUploadRequest completeMultipartUploadRequest(bucketName, objectKey, initMultiUploadResponse.upload_id());

//添加part信息，即part合并顺序
for (part_t partInfo : partEtags) {
    completeMultipartUploadRequest.add_part(partInfo.part_number, partInfo.etag);
}

// 完成分块上传
CompleteMultipartUploadResponse completeMultipartUploadResponse;
int ret = client.complete_multipart_upload(completeMultipartUploadRequest, &completeMultipartUploadResponse);

// 打印Object的ETag
std::cout << completeMultipartUploadResponse.etag() << std::endl;
```

上面代码中的 `partETags` 是第二部中保存的`part_t`的列表，BOS收到用户提交的Part列表后，会逐一验证每个数据Part的有效性。当所有的数据Part验证通过后，BOS将把这些数据part组合成一个完整的Object。

取消分块上传事件

用户可以使用abortMultipartUpload方法取消分块上传。

```
 AbortMultipartUploadRequest abortMultipartUploadRequest(bucketName, objectKey, uploadId);
AbortMultipartUploadResponse abortMultipartUploadResponse;
// 取消分块上传
int ret = client.abort_multipart_upload(abortMultipartUploadRequest, &abortMultipartUploadResponse);
```

获取未完成的分块上传事件

用户可以使用 `list_multipart_uploads` 方法获取Bucket内未完成的分块上传事件。

```
 ListMultipartUploadsRequest listMultipartUploadsRequest(bucketName);
ListMultipartUploadsResponse listMultipartUploadsResponse;
// 获取Bucket内所有上传事件
int ret = client.list_multipart_uploads(listMultipartUploadsRequest, &
listMultipartUploadsResponse);

if (ret != 0) {
    return ret;
}

// 遍历所有上传事件
for (const MultipartUploadSummary& multipartUpload : listMultipartUploadsResponse.uploads()) {
    std::cout << "Key: " << multipartUpload.key <<
        " UploadId: " << multipartUpload.upload_id << std::endl;
}
```

> **注意：**
>
> 1. 默认情况下，如果Bucket中的分块上传事件的数目大于1000，则只会返回1000个Object，并且返回结果中`is_truncated`的值为`True`，同时返回`next_marker`作为下次读取的起点。
> 2. 若想返回更多分块上传事件的数目，可以使用`set_marker`函数设置`marker`分次读取。

获取所有已上传的块信息

用户可以使用 `listParts` 方法获取某个上传事件中所有已上传的块。

```
 ListPartsRequest listPartsRequest(bucketName, objectKey, uploadId);

// 获取上传的所有Part信息
ListPartsResponse listPartsResponse;
int ret = client.list_parts(listPartsRequest, &listPartsResponse);

if (ret != 0) {
    return ret;
}

// 遍历所有Part
for (consr PartSummary& part : listPartsResponse.parts()) {
    std::cout << "PartNumber: " << part.part_number << " ETag: " << part.etag;
}
```

如果需要查看Object的存储类型storage class使用以下代码：

```
 public void listPartsStorageClass(){
    ListPartsRequest listPartsRequest(bucketName, objectKey, uploadId);

    // 获取上传的所有Part信息
    ListPartsResponse listPartsResponse;
    int ret = client.list_parts(listPartsRequest, &listPartsResponse);

    if (ret != 0) {
        return ret;
    }
    std::string storageClass = listPartsResponse.storage_class();
}
```

断点续传上传

当用户向BOS上传大文件时，如果网络不稳定或者遇到程序崩等情况，则整个上传就失败了，失败前已经上传的部分也作废，用户不得不重头再来。这样做不仅浪费资源，在网络不稳定的情况下，往往重试多次还是无法完成上传。 基于上述场景，BOS提供了断点续传上传的能力:

- 当网络情况一般的情况下，建议使用三步上传方式，将object分为5Mb的块，参考分块上传。
- 当您的网络情况非常差，推荐使用append_object的方式进行断点续传，每次 append 较小数据256kb，参考追加上传。

> 提示
>
> - 断点续传是分片上传的封装和加强，是用分片上传实现的；
> - 文件较大或网络环境较差时，推荐使用分片上传；

下载文件

下载文件

BOS C++ SDK提供了丰富的文件下载接口，用户可以通过以下方式从BOS中下载文件：

- 简单流式下载
- 下载到本地文件
- 下载到内存中的字符串
- 断点续传下载
- 范围下载
- 下载进度条

简单流式下载

用户可以通过如下代码将Object输出到一个文件流中：

```
void getObject(Client& client, const std::string& bucketName, const std::string& objectKey) {

  //获取文件输出流
  FileOutputStream outStream("test.txt");

  //方法一，通过request、response
  //初始化request
  GetObjectRequest getObjectRequest(bucketName, objectKey);
  //将outStream作为输出对象
  GetObjectResponse getObjectResponse(&outStream);
  int ret = client.get_object(getObjectRequest, &getObjectResponse);

  //方法二，通过便利接口,将Object写到outStream文件流中
  ret = client.download_file(bucketName, objectKey, outStream);
  ...
}
```

直接下载Object到文件

用户可以通过如下代码直接将Object下载到指定文件：

```
 // 下载Object到文件
std::string localFileName = "test.txt";
ObjectMetaData ObjectMetaData;

//设置Object读取范围
int64_t start = 0;
int64_t length = 100;
//下载Object
int ret = client.download_file(bucketName, objectKey, localFileName, start, length, &ObjectMetaData); //存到test.txt中,ObjectMetaData
...
```

当使用上面方法将Object直接下载到文件时，方法返回ObjectMetaData对象。

使用下载进度条

```
 // 上传进度回调函数
// 注意该回调函数中不得出现耗时较长/阻塞操作, 会影响数据下载性能.
// increment: 当次下载的数据量
// transfered: 已下载数据量
// total: 需下载的数据总量
// userData: 用户自定义数据, 例如object bucket+key等.
void progress_callback(int64_t increment, int64_t transfered, int64_t total, void* user_data) {
    std::cout << "progress_callback[" << user_data << "] => " <<
    increment <<" ," << transfered << "," << total << std::endl;
}

// 本地待接收数据文件
std::string filename = "/tmp/get_file_test";
FileOutputStream outStream(filename);

GetObjectRequest req(BUCKET, "transfer_progress_2");
GetObjectResponse rsp(&outStream);

// 设置下载进度条相关数据
TransferProgress progress;
progress.transfer_progress_cb = progress_callback;
rsp.set_progress(progress);

// 下载数据到本地filename文件中
int ret =  client()->get_object(req, &rsp);
if (ret) {
    LOGF(WARN, "client err: %d", ret);
}
if (rsp.is_fail()) {
    LOGF(WARN,
        "get_object: [status_code = %d], [message = %s], [requestid = %s]",
        rsp.status_code(),
        rsp.error().message().c_str(),
        rsp.error().request_id().c_str());
}
```

使用单链接限速

```
 // 本地待接收数据文件
std::string filename = "/tmp/get_file_test";
FileOutputStream outStream(filename);

GetObjectRequest req(BUCKET, "traffic_limit_file");
// 设置下载限速819200 bit/s(100KB/s)
// 限速值范围为: [819200 ~ 838860800] 即 [100KB - 100MB/s]
req.set_traffic_limit(819200);
GetObjectResponse rsp(&outStream);

// 下载数据到本地filename文件中
int ret =  client()->get_object(req, &rsp);
if (ret) {
    LOGF(WARN, "client err: %d", ret);
}
if (rsp.is_fail()) {
    LOGF(WARN,
        "get_object: [status_code = %d], [message = %s], [requestid = %s]",
        rsp.status_code(),
        rsp.error().message().c_str(),
        rsp.error().request_id().c_str());
}
```

下载到内存中的字符串

此方法直接将Object存到`string`当中，不会写入磁盘

```
 std::string inMemoryData;
//方法一，通过request、response
GetObjectRequest getObjectRequest(bucketName, objectKey);
//将inMemoryData作为输出对象
GetObjectResponse getObjectResponse(&inMemoryData);
int ret = client.get_object(getObjectRequest, &getObjectResponse);

if (ret != 0) {
    return ret;
}

//方法二，通过便利接口
ret = client.get_object(bucketName, objectKey, &inMemoryData);
//打印出来
std::cout << inMemoryData << std::endl;
```

范围下载

为了实现更多的功能，可以通过使用`GetObjectRequest`来指定下载范围，实现更精细化地获取Object。如果指定的下载范围是0 - 100，则返回第0到第100个字节的数据，包括第100个，共101字节的数据，即[0, 100]。

```
 GetObjectRequest getObjectRequest(bucketName, objectKey);
//初始化文件输出流
FileOutputStream outStream("test.txt");
GetObjectResponse getObjectResponse(&outStream);
//设置下载范围[0，100]字节数据
int start = 0;
int end = 100;
getObjectRequest.set_range(start, end);
int ret = client.get_object(getObjectRequest, &getObjectResponse);

if (ret != 0) {
    return ret;
}
return 0;
```

通过getObjectRequest的set_range方法可以设置返回Object的范围。用户也可以用此功能实现文件的分段下载和断点续传。 指定范围end为较大值, 可下载从start开始的Object数据。

获取文件下载URL

获取文件下载URL

用户可以通过如下代码获取指定Object的URL：

```
std::string generatePresignedUrl(Client& client, const std::string& bucketName, const std::string& objectKey, int expirationInSeconds) {

  //指定用户需要获取的Object所在的Bucket名称、该Object名称、URL的有效时长
  return  client.generate_url(bucketName, objectKey, expirationInSeconds);
}
```

> **说明：**
>
> - 用户在调用该函数前，需要手动设置endpoint为所属区域域名。百度智能云目前开放了多区域 支持，请参考区域选择说明。目前支持“华北-北京”、“华南-广州”和“华东-苏州”三个区域。北京区域：http://bj.bcebos.com，广州区域：http://gz.bcebos.com，苏州区域：http://su.bcebos.com。
> - expirationInSeconds为指定的URL有效时长，时间从当前时间算起，为可选参数，不配置时系统默认值为1800秒。如果要设置为永久不失效的时间，可以将expirationInSeconds参数设置为 -1，不可设置为其他负数。
> - 如果预期获取的文件时公共可读的，则对应URL链接可通过简单规则快速拼接获取: http://$bucket.$region.bcebos.com/$object

列举存储空间中的文件

列举存储空间中的文件

BOS SDK支持用户通过以下两种方式列举出object：

- 简单列举
- 通过参数复杂列举

除此之外，用户还可在列出文件的同时模拟文件夹

简单列举

当用户希望简单快速列举出所需的文件时，可通过listObjects方法返回ListObjectsResponse对象，ListObjectsResponse对象包含了此次listObject请求的返回结果。用户可以通过ListObjectsResponse中的getContents方法获取所有Object的描述信息。

```
void listObjects(Client& client, const std::string bucketName) {
  ListObjectsRequest listObjectsRequest(bucketName);
  // 获取指定Bucket下的所有Object信息
  ListObjectsResponse listObjectsResponse;
  int ret = client.list_objects(listObjectsRequest, &listObjectsResponse);

  if (ret != 0) return;
  if (listObjectResponse.is_fail()) {
      return;
  }
  // 遍历所有Object
  for (const ObjectSummary& objectSummary : listObjectsResponse.contents()) {
      std::cout << "ObjectKey: " << objectSummary.key << std::endl;
  }
}
```

> **注意：**
>
> 1. 默认情况下，如果Bucket中的Object数量大于1000，则只会返回1000个Object，并且返回结果中is_truncated值为True，并返回next_marker做为下次读取的起点。
> 2. 若想增大返回Object的数目，可以使用marker参数分次读取。

通过参数复杂列举

除上述简单列举外，用户还可通过设置ListObjectsRequest的参数实现各种灵活的查询功能。ListObjectsRequest的可设置的参数如下：

参数	功能	使用方式
prefix	限定返回的object key必须以prefix作为前缀	set_prefix(const std::string& prefix)
delimiter	是一个用于对Object名字进行分组的字符所有名字包含指定的前缀且第一次出现。Delimiter字符之间的object作为一组元素: CommonPrefixes	set_delimiter(const std::string& delimiter)
marker	设定结果从marker之后按字母排序的第一个开始返回	set_marker(const std::string& marker)
max_keys	限定此次返回object的最大数，最大为1000，默认值是1000。如果指定的值大于1000，按1000操作	set_max_keys(int maxKeys)

> **注意：**
>
> 1. 如果有Object以Prefix命名，当仅使用Prefix查询时，返回的所有Key中仍会包含以Prefix命名的Object，详见递归列出目录下所有文件。
> 2. 如果有Object以Prefix命名，当使用Prefix和Delimiter组合查询时，返回的所有Key中会有Null，Key的名字不包含Prefix前缀，详见查看目录下的文件和子目录。

下面我们分别以几个案例说明通过参数列举的方法：

指定最大返回条数

```
 // 指定最大返回条数为500
ListObjectsRequest listObjectsRequest("bucketName");
listObjectsRequest.set_max_keys(500);
ListObjectsResponse listObjectsResponse;
int ret = client.list_objects(listObjectsRequest, &listObjectsResponse);
if (ret != 0) return;
...//response异常处理
for(const ObjectSummary& objectSummary :listObjectsResponse.contents()) {
    std::cout << "ObjectKey: " << objectSummary.key << std::endl;
}
```

返回指定前缀的object

```
 // 指定返回前缀为test的object
ListObjectsRequest listObjectsRequest("bucketName");
listObjectsRequest.set_prefix("test");
ListObjectsResponse listObjectsResponse;
int ret = client.list_objects(listObjectsRequest, &listObjectsResponse);
if (ret != 0) return;
...//response异常处理
for(const ObjectSummary& objectSummary :listObjectsResponse.contents()) {
    std::cout << "ObjectKey: " << objectSummary.key << std::endl;
}
```

从指定Object后返回

```
 // 用户可以定义不包括某object，从其之后开始返回
ListObjectsRequest listObjectsRequest("bucketName");
listObjectsRequest.set_marker("object");
ListObjectsResponse listObjectsResponse;
int ret = client.list_objects(listObjectsRequest, &listObjectsResponse);
if (ret != 0) return;
...//response异常处理
for(const ObjectSummary& objectSummary :listObjectsResponse.contents()) {
    std::cout << "ObjectKey: " << objectSummary.key << std::endl;
}
```

分页获取所有Object

```
 // 用户可设置每页最多500条记录
ListObjectsRequest listObjectsRequest("bucketName");
listObjectsRequest.set_max_keys(500);
ListObjectsResponse listObjectsResponse;
int ret = client.list_objects(listObjectsRequest, &listObjectsResponse);
if (ret != 0) return;
...//response异常处理

bool isTruncated = true;
while (isTruncated) {
    ListObjectsResponse listObjectsResponse;
    client.list_objects(listObjectsRequest, &listObjectsResponse);
    isTruncated = listObjectsResponse.is_truncated();
    if (listObjectsResponse.next_marker() != "") {
        listObjectsRequest.set_marker(listObjectsResponse.next_marker());
    }
}
```

分页获取所有特定Object后的结果

```
 // 用户可设置每页最多500条记录，并从某特定object之后开始获取
ListObjectsRequest listObjectsRequest("bucketName");
listObjectsRequest.set_max_keys(500);
listObjectsRequest.set_marker("object");
ListObjectsResponse listObjectsResponse;
int ret = client.list_objects(listObjectsRequest, &listObjectsResponse);
if (ret != 0) return;

bool isTruncated = true;
while (isTruncated) {
    ListObjectsResponse listObjectsResponse;
    client.list_objects(listObjectsRequest, &listObjectsResponse);
    isTruncated = listObjectsResponse.is_truncated();
    if (listObjectsResponse.next_marker() != "") {
        listObjectsRequest.set_marker(listObjectsResponse.next_marker());
    }
}
```

分页获取所有指定前缀的Object结果

```
 // 用户可设置分页获取指定前缀的Object，每页最多500条记录
ListObjectsRequest listObjectsRequest("bucketName");
listObjectsRequest.set_max_keys(500);
listObjectsRequest.set_prefix("object");
ListObjectsResponse listObjectsResponse;
int ret = client.list_objects(listObjectsRequest, &listObjectsResponse);
if (ret != 0) return;

bool isTruncated = true;
while (isTruncated) {
    ListObjectsResponse listObjectsResponse;
    client.list_objects(listObjectsRequest, &listObjectsResponse);
    isTruncated = listObjectsResponse.is_truncated();
    if (listObjectsResponse.next_marker() != "") {
        listObjectsRequest.set_marker(listObjectsResponse.next_marker());
    }
}
```

list_object方法返回的ListObjectsResponse解析类中可供调用的参数有：

参数	说明
name	Bucket名称
prefix	匹配以prefix开始到第一次出现Delimiter字符之间的object作为一组元素返回
marker	本次查询的起点
max_keys	请求返回的最大数目
is_truncated	指明是否所有查询都返回了；false-本次已经返回所有结果，true-本次还没有返回所有结果
contents	返回的一个Object的容器
+key	Object名称
+last_modified	此Object最后一次被修改的时间
+etag	Object的HTTP协议实体便签
+storage_class	Object的存储形态
+size	Object的内容的大小（字节数）
+owner_id	Bucket Owner的用户ID
+owner_display_name	Bucket Owner的名称

模拟文件夹功能

在BOS的存储结果中是没有文件夹这个概念的，所有元素都是以Object来存储，但BOS的用户在使用数据时往往需要以文件夹来管理文件。 因此，BOS提供了创建模拟文件夹的能力，其本质上来说是创建了一个size为0的Object。对于这个Object可以上传下载，只是控制台会对以"/"结尾的Object以文件夹的方式展示。

用户可以通过 Delimiter 和 Prefix 参数的配合模拟出文件夹功能。Delimiter 和 Prefix 的组合效果是这样的：

如果把 Prefix 设为某个文件夹名，就可以罗列以此 Prefix 开头的文件，即该文件夹下递归的所有的文件和子文件夹（目录）。文件名在Contents中显示。 如果再把 Delimiter 设置为 "/" 时，返回值就只罗列该文件夹下的文件和子文件夹（目录），该文件夹下的子文件名（目录）返回在 CommonPrefixes 部分，子文件夹下递归的文件和文件夹不被显示。

如下是几个应用方式：

列出Bucket内所有文件

当用户需要获取Bucket下的所有文件时，可以参考分页获取所有Object

递归列出目录下所有文件

可以通过设置 Prefix 参数来获取某个目录下所有的文件：

```
 // 构造ListObjectsRequest请求
ListObjectsRequest listObjectsRequest("bucketName");
// 递归列出fun目录下的所有文件
listObjectsRequest.set_prefix("fun/");
ListObjectsResponse listObjectsResponse;
int ret = client.list_objects(listObjectsRequest, &listObjectsResponse);
if (ret != 0) return;
...//response异常处理
std::cout << "ObjectKeys:" << std::endl;
// 遍历所有Object
for(const ObjectSummary& objectSummary :listObjectsResponse.contents()) {
    std::cout << objectSummary.key << std::endl;
}
```

输出：

```
 ObjectKeys:
fun/
fun/movie/001.avi
fun/movie/007.avi
fun/test.jpg
```

查看目录下的文件和子目录

在 Prefix 和 Delimiter 结合的情况下，可以列出目录下的文件和子目录：

```
 // 构造ListObjectsRequest请求
ListObjectsRequest listObjectsRequest("bucketName");

// "/" 为文件夹的分隔符
listObjectsRequest.set_delimiter("/");

// 列出fun目录下的所有文件和文件夹
listObjectsRequest.set_prefix("fun/");

ListObjectsResponse listObjectsResponse;
int ret = client.list_objects(listObjectsRequest, &listObjectsResponse);
if (ret != 0) return;
// 遍历所有Object
std::cout << "\nObjectKeys:" << std::endl;
for(const ObjectSummary& objectSummary :listObjectsResponse.contents()) {
    std::cout << objectSummary.key << std::endl;
}

// 遍历所有CommonPrefix
std::cout << "CommonPrefixs:" << std::endl;
for (const std::string& commonPrefix : listObjectsResponse.common_prefixes()) {
    std::cout << commonPrefix << std::endl;
}
```

输出：

```
 ObjectKeys:
fun/
fun/test.jpg

CommonPrefixs:
fun/movie/
```

返回的结果中， `ObjectSummary` 的列表contents中给出的是fun目录下的文件。而 `common_prefixes` 的列表中给出的是fun目录下的所有子文件夹。可以看出 `fun/movie/001.avi` ， `fun/movie/007.avi` 两个文件并没有被列出来，因为它们属于 `fun` 文件夹下的 `movie` 目录。

列举Bucket中object的存储属性

当用户完成上传后，如果需要查看指定Bucket中的全部Object的storage class属性，可以通过如下代码实现：

```
 public void listObjectsStorageClass(){
    ListObjectsRequest listObjectsRequest("bucketName");
    ListObjectsResponse listObjectsResponse;
    int ret = client.list_objects(listObjectsRequest, &listObjectsResponse);
    std::vector<ObjectSummary> objectList = listObjectResponse.contents();
    for(int i = 0; i<objectList.size(); i++) {
        std::cout << objectList[i].storage_class() << std::endl;
    }
}
```

Object权限控制

权限控制

设置对象的访问权限

目前BOS支持两种方式设置ACL. 具体可参考权限控制

第一种是使用Canned Acl，在put_object_acl的时候，通过头域的"x-bce-acl", "x-bce-grant-read", 或者 "x-bce-grant-permission"来设置对象的访问权限，当前可设置的权限包括private和public-read，三种类型的header不可以同时在一个请求中出现.

第二种方式是以自定义Acl样式, 具体可通过上传其json字符串, 设置access_control_list结构体, 或者直接上传ACL文件. 具体可参考权限控制概述

设置Canned ACL

Canned ACL是预定义的访问权限，用户可选择对某个对象进行设置，支持三种接口：

```
 PutObjectAclRequest putObjectAclRequest(bucketName, objectKey);
PutObjectAclResponse putObjectAclResponse;

// 1. 使用x-bce-acl Header设置
// cannedAcl支持:private、public-read
std::string cannedAcl="public-read";
putObjectAclRequest.set_canned_acl(cannedAcl);

// 2. 使用x-bce-grant-read / x-bce-grant-read-permission Header设置
// idStrings为id集合, 可一次传入多个id, 用逗号隔开, 字符串固定格式为:"id=/"xxxxx/", id=/"xxxxx/"";
std::string idStrings="id=\"77f47fbbc29d41xxxxxxxxx6\"";
putObjectAclRequest.set_xbce_grant_read(idStrings);
putObjectAclRequest.set_xbce_grant_full_control(idStrings);

int ret = client.put_object_acl(putObjectAclRequest, &putObjectAclResponse);
if (ret) {
    LOGF(WARN, "client err: %d", ret);
}
if (putObjectAclResponse.is_fail()) {
    LOGF(WARN,"put_object_acl: [status_code = %d], [message = %s], [requestid = %s]",
        putObjectAclResponse.status_code(),
        putObjectAclResponse.error().message().c_str(),
        putObjectAclResponse.error().request_id().c_str());
}
```

> 注意: 三种方式set_canned_acl(),set_xbce_grant_read(),set_xbce_grant_full_control()
> 一次put_object_acl()调用只能设置上述三种接口其中之一.

设置自定义ACL

用户可参考如下代码设置Bucket内的对象的自定义访问权限，支持三种不同参数：

```
 PutObjectAclRequest putObjectAclRequest(bucketName, objectKey);
PutObjectAclResponse putObjectAclResponse;

// 1. 通过上传acl json串
std::string jsonAcl =
                "{\"accessControlList\":[{\"grantee\":[{\"id\":\"*\"}],\"permission\":[\"READ\"]},{"
                "\"grantee\":[{\"id\":\"cb5f8xxxxxxxxx82bbc\"}],\"permission\":["
                "\"FULL_CONTROL\"]}]}";
std::string cannedAcl="public-read";
putObjectAclRequest.set_json_acl(jsonAcl);


// 2. 上传acl文件
std::string aclFilePath = "/tmp/acl.json"
int setRet = putObjectAclRequest.set_acl_file(aclFilePath);
if (ret) {
    LOGF(WARN, "client set_acl_file: %d", ret);
}

// 3. 通过设置access_control_list数据
std::vector<Grant> grants;
Grant grant;
grantee.id = "77fxxxxxxxxxx5fa406";
grant.grantee.push_back(grantee);
grant.permission.push_back("READ");
grants.push_back(grant);
putObjectAclRequest.set_access_control_list(grants);

int ret = client.put_object_acl(putObjectAclRequest, &putObjectAclResponse);
if (ret) {
    LOGF(WARN, "client err: %d", ret);
}
if (putObjectAclResponse.is_fail()) {
    LOGF(WARN,"put_object_acl: [status_code = %d], [message = %s], [requestid = %s]",
        putObjectAclResponse.status_code(),
        putObjectAclResponse.error().message().c_str(),
        putObjectAclResponse.error().request_id().c_str());
}
```

获取对象的访问权限

如下代码可获取一个对象的访问权限：

```
 GetObjectAclRequest getObjectAclRequest(bucketName, objectKey);
GetObjectAclResponse getObjectAclResponse;

int ret = client()->get_object_acl(getObjectAclRequest, &getObjectAclResponse);
if (ret) {
    LOGF(WARN, "get_object_acl err: %d", ret);
}

if (getObjectAclResponse.is_fail()) {
    LOGF(WARN,
        "get_object_acl: [status_code = %d], [message = %s], [requestid = %s]",
        getObjectAclResponse.status_code(),
        getObjectAclResponse.error().message().c_str(),
        getObjectAclResponse.error().request_id().c_str());
}

//获取具体权限(两种方式)
std::vector<Grant> objectAcl = getObjectAclResponse.access_control_list();
std::string objectAclJsonStr = getObjectAclResponse.json_access_control_list();
```

```
 //acl具体结构
struct Grantee {
    std::string id;
};
struct Grant {
    std::vector<Grantee> grantee;
    std::vector<std::string> permission;
    //std::vector<std::string> resource;
    //std::vector<std::string> notResource;
    //Condition condition;
    //std::string effect;
}
```

> 注意: acl涉及到的具体结构体Grant, 在bucket acl和object acl体系中共用
> 目前object acl体系中只用到其中grantee, permission两个字段.
> 其余注释的字段均为bucket acl体系独有.

删除对象的访问权限

对设置过访问权限的对象，可以调用此接口进行删除：

```
 DeleteObjectAclRequest deleteObjectAclRequest(BUCKET_NAME, OBJECT_NAME);
DeleteObjectAclResponse deleteObjectAclResponse;

int ret = client.delete_object_acl(deleteObjectAclRequest, &deleteObjectAclResponse);
if (ret) {
    LOGF(WARN, "client err: %d", ret);
}
if (deleteObjectAclResponse.is_fail()) {
    LOGF(WARN, "put_object_acl: [status_code = %d], [message = %s], [requestid = %s]",
        deleteObjectAclResponse.status_code(),
        deleteObjectAclResponse.error().message().c_str(),
        deleteObjectAclResponse.error().request_id().c_str());
}
```

删除文件

删除文件

删除单个文件

可参考如下代码删除了一个Object:

```
 int deleteObject(Client& client,const std::string bucketName, const std::string objectKey) {

    // 删除Object
    return client.delete_object(bucketName, objectKey);          //指定要删除的Object所在Bucket名称和该Object名称
}
```

删除多个文件

删除多个文件`deleteMultiObject`使用并发接口，提高请求数的吞吐，代码如下：

```
int deleteMultiObject(Client& client, const& std::string bucketName, const std::vector<std::string>& objects) {
  //批量删除object
  std::vector<BceRequestContext> ctx(objects.size());
  int i = 0;
  for (const std::string& objectKey : objects) {
     //构造上下文信息
     ctx[i].request = new DeleteObjectRequest(bucketName, objectKey);
     ctx[i].response = new DeleteObjectResponse;
     ctx[i].is_own = true;//自动析构request和response
     ++i;
  }
  //并发请求
  int ret = client.send_request((int)ctx.size(), &ctx.front());
  if (ret != 0) return ret;
  //处理response
  for (size_t i = 0; i < ctx.size(); ++i) {
     DeleteObjectRequest* request = (DeleteObjectRequest*)ctx[i].request;
     DeleteObjectResponse* response = (DeleteObjectResponse*)ctx[i].response;
     if (response->is_fail()) {//有请求删除失败，处理这种情况
        std::cout << "delete objectKey=" << request->object_name() << " failed"
                  << " reason: " << response->error().message();
        ret = RET_SERVICE_ERROR;
     }
  }
  return ret;
}
```

查看文件是否存在

查看文件是否存在

用户可通过如下操作查看某文件是否存在：

```
 HeadObjectRequest request(bucketName, objectKey);
HeadObjectResponse response;
int ret = client.head_object(request, &response);
if (response.status_code() == 404) {
    std::cout << "ObjectKey not exist!" << std::endl;
}
```

获取及更新文件元信息

文件元信息

设置文件元信息

文件元信息(Object Meta)，是对用户在向BOS上传文件时，同时对文件进行的属性描述，主要分为分为两种：设置HTTP标准属性（HTTP Headers）和用户自定义的元信息。

- **设定Object的Http Header**

BOS C++ SDK本质上是调用后台的HTTP接口，因此用户可以在上传文件时自定义Object的Http Header。常用的http header说明如下：

名称	描述	默认值
Content-MD5	文件数据校验，设置后BOS会启用文件内容MD5校验，把您提供的MD5与文件的MD5比较，不一致会抛出错误	无
Content-Type	文件的MIME，定义文件的类型及网页编码，决定浏览器将以什么形式、什么编码读取文件。如没有指，BOS则根据文件的扩展名自动生成，如文件没有扩展名则填默认值	application/octet-stream
Content-Disposition	指示MIME用户代理如何显示附加的文件，打开或下载，及文件名称	无
Content-Length	上传的文件的长度，超过流/文件的长度会截断，不足为实际值	流/文件时间长度
Expires	缓存过期时间	无
Cache-Control	指定该Object被下载时的网页的缓存行为	无

参考代码如下：

```
 ...
// 初始化meta
ObjectMetaData meta;

// 设置ContentType
meta.set_content_type("application/json");

// 设置cache-control
meta.set_cache_control("no-cache");

// 设置x-bce-storage-class
meta.set_storage_class("STANDARD");

ret = client.upload_file(bucketName, objectKey, content, meta);
...
```

- **用户自定义元信息**

BOS支持用户自定义元数据来对Object进行描述。如下代码所示：

```
 // 设置自定义元数据name的值为my-data
meta.set_user_meta("name", "my-data");

// 上传Object
ret = client.upload_file(bucketName, objectKey, file_name, meta);
```

> **提示：**
>
> - 在上面代码中，用户自定义了一个名字为"name"，值为"my-data"的元数据
> - 当用户下载此Object的时候，此元数据也可以一并得到
> - 一个Object可以有多个类似的参数，但所有的User Meta总大小不能超过2KB

获取Object的存储类型

Object的storage class属性分为STANDARD(标准存储), STANDARD_IA(低频存储)和COLD(冷存储)，通过如下代码可以实现：

```
int getObjectStorageClass(){
    HeadObjectRequest request(bucketName, objectKey);
    HeadObjectResponse response;
    int ret = client.head_object(request, &response);
    if (ret != 0) {
        return ret;
    }
    ObjectMetaData& meta = response.meta();
    std::string storageClass = meta.storage_class();
    std::cout << storageClass << std::endl;
    return 0;
}
```

只获取ObjectMetaData

通过 head_object 方法可以只获取ObjectMetaData而不获取Object的实体。

`ObjectMetaData`解析类中可供调用的参数有：

参数	说明
content_type	Object的类型
content_length	Object的大小
content_md5	Object的MD5
etag	Object的HTTP协议实体标签
storage_class	Object的存储类型
user_meta	如果在PutObject指定了userMetadata自定义meta，则返回此项
expires	下载Object时的缓存失效时间
content_disposition	设置浏览器是否下载，可取值为inline、attachment; filename="download.txt"
cache_control	下载Object的Cache设置，常见的可取值为private、no-cache、max-age、must-revalidate
content_range	有range的情况下返回Object的数据范围

获取及更新文件元信息

文件元信息(Object Metadata)，是对用户上传BOS的文件的属性描述，分为两种：HTTP标准属性（HTTP Headers）和User Meta（用户自定义元信息）。

```
void getObjectMeta(Client& client, const std::string& bucketName, const std::string& objectKey, ObjectMetaData* objectMetaData) {

  HeadObjectRequest request(bucketName, objectKey);
  HeadObjectResponse rsponse;

  request.set_meta(newObjectMetaData);


  int ret = client.head_object(request, &rsponse);
  if (ret != 0) {
      return;
  }
  objectMetaData = &rsponse.meta();

}
```

修改文件元信息

BOS修改Object的Metadata通过拷贝Object实现。即拷贝Object的时候，把目的Bucket设置为源Bucket，目的Object设置为源Object，并设置新的Metadata，通过拷贝自身实现修改Metadata的目的。如果不设置新的Metadata，则报错。

```
void setObjectMeta(Client& client, const std::string& bucketName, const std::string& objectKey, ObjectMetaData* newObjectMetaData) {

  CopyObjectRequest request(bucketName, objectKey, bucketName, objectKey);

  // 设置新的ObjectMetaData
  request.set_meta(newObjectMetaData);

  // 拷贝Object
  CopyObjectResponse copyObjectResponse;
  int ret = client.copy_object(request, &copyObjectResponse);
  if (ret != 0) {
      return;
  }
  // 打印结果
  std::cout << "ETag: " << copyObjectResponse.etag() << " LastModified: " << copyObjectResponse.last_modified() << std::endl;
}
```

拷贝文件

拷贝文件

拷贝一个文件

用户可以通过copy_object函数拷贝一个Object，如下代码所示：

```
void copyObject(Client& client, std::string destBucketName, std::string destKey, const std::string& srcBucketName, const std::string& srcKey) {

  // 拷贝Object
  int ret = client.copy_object(srcBucketName, srcKey, destBucketName, destKey, "STANDARD");//目标Object为标准存储
}
```

`CopyObjectResponse` 对象中包含了新Object的ETag和修改时间。

用户也可以通过 `CopyObjectRequest` 实现Object的拷贝，如下代码所示：

```
 // 初始化Client
Client client = ...;

// 创建CopyObjectRequest对象
CopyObjectRequest copyObjectRequest(destBucketName, destKey, srcBucketName, srcKey);

// 设置Metadata
ObjectMetaData* userMetadata = copyObjectRequest.mutable_meta();
userMetadata->set_user_meta("userMetaKey", "userMetaValue");

// 复制Object
CopyObjectResponse copyObjectResponse;
int ret = client.copy_object(copyObjectRequest, &copyObjectResponse);

std::cout << "ETag: " + copyObjectResponse.etag() <<
 " LastModified: " << copyObjectResponse.last_modified());
```

同步Copy功能

当前BOS的CopyObject接口是通过同步方式实现的。同步方式下，BOS端会等待Copy实际完成才返回成功。同步Copy能帮助用户更准确的判断Copy状态，但用户感知的复制时间会变长，且复制时间和文件大小成正比。

同步Copy方式更符合业界常规，提升了与其它平台的兼容性。同步Copy方式还简化了BOS服务端的业务逻辑，提高了服务效率。

分块拷贝

除了通过CopyObject接口拷贝文件以外，BOS还提供了另外一种拷贝模式——Multipart Upload Copy。用户可以在如下的应用场景内（但不仅限于此），使用Multipart Upload Copy，如：

- 需要支持断点拷贝。
- 拷贝超过5GB大小的文件。
- 网络条件较差，和BOS的服务器之间的连接经常断开。

下面将介绍分步实现三步拷贝。

三步拷贝包含init、“拷贝分块”和complete三步，其中init和complete的操作同分块上传一致。

为了便于理解，下面提供三步拷贝完整代码：

```
 // 第一步 init
InitMultiUploadRequest initMultiUploadRequest("targetBucketName","targetObjectName");
InitMultiUploadResponse initMultiUploadResponse;
client.init_multipart_upload(initMultiUploadRequest, &initMultiUploadResponse);
//获取Object大小
HeadObjectRequest request("targetBucketName", "targetObjectName");
HeadObjectResponse response;
client.head_object(request, &response);
if (response.is_fail()) {
    return RET_SERVICE_ERROR;
}
long left_size = response.meta().content_length();

// 第二步 分块拷贝
long skipBytes = 0;
int partNumber = 1;
std::vector<part_t> partETags;

while (left_size > 0) {
    long partSize = 1024 * 1024 * 5L;
    if (left_size < partSize) {
        partSize = left_size;
    }
    CopyPartRequest copyPartRequest;
    copyPartRequest.set_upload_id(initMultiUploadResponse.upload_id());
    copyPartRequest.set_bucket_name("targetBucketName");
    copyPartRequest.set_object_name("targetObjectName");
    copyPartRequest.set_source_bucket_name("sourceBucketName");
    copyPartRequest.set_source_object_name("sourceObjectName");
    copyPartRequest.set_range(skipBytes, skipBytes + partSize - 1);
    copyPartRequest.set_part_number(partNumber);

    CopyPartResponse copyPartResponse;
    client.copy_part(copyPartRequest, &copyPartResponse);
    // 将返回的partNumber和ETag保存到vector
    part_t partInfo;
    partInfo.etag = copyPartResponse.etag();
    partInfo.part_number = partNumber;
    partETags.push_back(partInfo);
    left_size -= partSize;
    skipBytes += partSize;
    partNumber+=1;
}

// 第三步 complete
CompleteMultipartUploadRequest completeMultipartUploadRequest("targetBucketName", "targetObjectName", initMultiUploadResponse.upload_id());

//添加part信息，即part合并顺序
for (part_t partInfo : partETags) {
    completeMultipartUploadRequest.add_part(partInfo.part_number, partInfo.etag);
}

// 完成分块上传
CompleteMultipartUploadResponse completeMultipartUploadResponse;
int ret = client.complete_multipart_upload(completeMultipartUploadRequest, &completeMultipartUploadResponse);

// 打印Object的ETag
std::cout << completeMultipartUploadResponse.etag() << std::endl;
```

注意:

1. offset参数以字节为单位，为分块的开始偏移位置。
2. size参数以字节为单位，定义每个分块的大小，除最后一个Part以外，其他的Part大小都要大于 5MB。

使用便利接口

`parallel_copy`使用并发接口，提高请求吞吐，实现分块分块拷贝

```
client.parallel_copy(srcBucketName, sourceObjectName, destBucketName, destObjectName);
```

归档存储

归档存储 上传

```
 std::string filename = "test_archive";
FileInputStream file(filename);
PutObjectRequest request(bucket, "test_archive", &file);
request.mutable_meta()->set_storage_class("ARCHIVE");
PutObjectResponse result;
client.put_object(request, &result);
```

取回

```
 RestoreObjectRequest request(bucket, ""test_archive"");
request.set_restore_tier("Expedited"); //加急
request.set_restore_tier("Standard"); //标准
request.set_restore_tier("LowCost"); //延迟
RestoreObjectResponse response;
client.restore_object(request, &response);
print_common_response(response);
```

变更文件存储等级

变更文件存储等级

上文中已提到，BOS支持为文件赋予STANDARD(标准存储), STANDARD_IA(低频存储)和COLD(冷存储)三种存储类型。同时，BOS C++ SDK也支持用户对特定文件执行存储类型变更的操作。 涉及到的参数如下：

参数	说明
x-bce-storage-class	指定Object的存储类型，STANDARD_IA代表低频存储，COLD代表冷存储，不指定时默认是标准存储类型。

示例如下：

```
 // 标准存储转为低频存储
CopyObjectRequest copyObjectRequest(destBucketName, destKey, srcBucketName, srcKey);
copyObjectRequest.mutable_meta()->set_storage_class("STANDARD_IA");
CopyObjectResponse copyObjectResponse;
int ret = client.copy_object(copyObjectRequest, &copyObjectResponse);

// 低频存储转为冷存储
CopyObjectRequest copyObjectRequest(destBucketName, destKey, srcBucketName, srcKey);
copyObjectRequest.mutable_meta()->set_storage_class("COLD");
CopyObjectResponse copyObjectResponse;
ret = client.copy_object(copyObjectRequest, &copyObjectResponse);
```

Object标签管理

Object Tagging

BOS目前支持直接设置object标签, 以及上传object时指定object标签

设置文件的标签

```
 PutObjectTaggingRequest req("bucket", "object");
PutObjectTaggingResponse rsp;

req.add_object_tagging("key", "valaue");
req.add_object_tagging("key1", "valaue1");

int ret = client()->put_object_tagging(req, &rsp);
if (ret) {
    LOGF(WARN, "client err: %d", ret);
}
if (rsp.is_fail()) {
    LOGF(WARN,
        "put_ns_object: [status_code = %d], [message = %s], [requestid = %s]",
        rsp.status_code(),
        rsp.error().message().c_str(),
        rsp.error().request_id().c_str());
}
```

上传文件, 同时设置标签

```
 PutObjectRequest req("bucket", "object", "data");
PutObjectResponse rsp;

#设置标签
req.add_object_tagging("key", "valaue");
req.add_object_tagging("key1", "value1");

int ret = client()->put_object(req, &rsp);
if (ret) {
    LOGF(WARN, "client err: %d", ret);
}
if (rsp.is_fail()) {
    LOGF(WARN,
        "put_ns_object: [status_code = %d], [message = %s], [requestid = %s]",
        rsp.status_code(),
        rsp.error().message().c_str(),
        rsp.error().request_id().c_str());
}
```

获取文件的标签

```
 GetObjectTaggingRequest req("bucket", "object");
GetObjectTaggingResponse rsp;

int ret = client()->get_object_tagging(req, &rsp);
if (ret) {
    LOGF(WARN, "client err: %d", ret);
}
if (rsp.is_fail()) {
    LOGF(WARN,
        "put_ns_object: [status_code = %d], [message = %s], [requestid = %s]",
        rsp.status_code(),
        rsp.error().message().c_str(),
        rsp.error().request_id().c_str());
}

// 遍历获取到的标签
for (const auto& tag : rsp.tag_set()) {
    LOGF(INFO, "key: %s, value: %s", tag.key.c_str(), tag.value.c_str());
}
```

请求管理

中断取消请求

中断取消请求

在0.8.7.4版本中，C++ SDK新增了支持请求的中断取消，允许用户另起线程来主动取消请求。本文主要介绍如何使用该功能来取消请求：

前置配置

要能够实现取消请求，除了在请求在被处理时候的操作以外，C++ SDK还需要用户请求被处理前对于相应的Client和Request在创建时配置相关变量，以防误操作取消了本不希望取消的请求。其主要配置如下：

- `ClientOption`：在client被创建时，如果你希望从这个client发出的请求能够支持请求取消，请在配置client的clientOption下添加如下代码：`clientOption.client_enable_cancel = true;`(此处clientOption仅为举例，具体请按照自己代码中的变量名进行相应的更改)
- `Request`：在创建请求后，请手动添加如下代码修改该请求的属性， 从而使得该请求能够支持取消：`request.set_request_cancelable(true);`(此处request仅为举例，具体请按照自己代码中的变量名进行相应的更改)

请求取消

在可被取消的请求被发出后，并在处理完之前，用户可以在其他线程调用`client.cancel(*request)`来进行请求取消。

简单示例

Client创建的前置配置：

```
 ClientOptions option;
option.max_parallel = 10;
option.endpoint = "bj.bcebos.com";
option.client_enable_cancel = true;

std::string ak = "";
std::string sk = "";
Client client(ak, sk, option);

test_cancel();
```

发送可取消的请求并在一秒后取消：

```
 void threadPutObject(Client* client, PutObjectRequest* request, PutObjectResponse* response){
    client->put_object(*request, response);
}

void threadChangeCancel(Client* client, PutObjectRequest* request){
    std::this_thread::sleep_for(std::chrono::milliseconds(1000));
    client->cancel_request(*request);
}

void test_cancel(Client& client){
    FileInputStream file(g_object);
    PutObjectRequest request(g_bucket, g_object, &file);
    PutObjectResponse response;
    request.set_request_cancelable(true);
    std::thread t1(threadPutObject, &client, &request, &response);
    std::thread t2(threadChangeCancel, &client, &request);

    t1.join();
    t2.join();
}
```

异常处理

客户端异常

客户端异常表示客户端尝试向BOS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，client操作会返回非零；当上传文件时发生IO异常时，返回非零。

客户端执行put_object、delete_bucket等操作返回非零，具体返回错误码信息见bcesdk/common/common.h。也可调用bcesdk/util/util.h当中的stringfy_ret_code函数将返回的错误码转为字符串进行打印。

```
 int ret = client.upload_file("bucketName", "objectName", "fileName");
if (ret != 0) {
    std::cout << "client error occurs: " << stringfy_ret_code(ret) << std::endl;
    return ret;
}
return RET_OK;//RET_OK=0代表成功执行

//建议使用request&response标准请求接口, 请求更加规范
```

服务端异常

当BOS服务端出现异常时，BOS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BOS错误信息格式

服务端出现异常，会将错误信息写到response中，通过response进行异常处理

```
 int ret = client.put_object(request, &response);
if (response.is_fail()) {
    //状态码，0表示成功，400~599代表HTTP错误码，1000~代表客户端错误
    //建议打印requestid, 有助于快速定位异常请求
    int status = response.status_code();
    std::string msg = response.error().message();
    std::string request_id = response.error().request_id();
    printf("put object fail, [status_code = %d], [message = %s], [requestid = %s]", status, msg.c_str(), request_id.c_str());
    return status;
}
return RET_OK;
```

sdk日志

SDK日志

```
 //设置日志
FILE *logfp = fopen("sdk.log", "w");//设置日志输出文件
sdk_set_log_stream(logfp);//设置输出流
sdk_set_log_level(SDK_LOG_DEBUG);//设置日志级别
```

日志一共五个级别：

1. SDK_LOG_DEBUG
2. SDK_LOG_INFO
3. SDK_LOG_WARN
4. SDK_LOG_ERROR
5. SDK_LOG_FATAL

如您在使用过程中出现了预期之外的问题需要定位，建议把日志级别调整至SDK_LOG_DEBUG，进行初步判断

版本变更记录

- C++ SDK开发包[2024-3-18] 版本号 0.8.7.8
  - 底层jsoncpp库升级至1.9.0
- C++ SDK开发包[2024-3-18] 版本号 0.8.7.7
  - 支持层级NameSpace相关功能
- C++ SDK开发包[2024-12-13] 版本号 0.8.7.6
  - 支持BosClient使用连接池长连接发送请求
- C++ SDK开发包[2024-11-22] 版本号 0.8.7.5
  - 支持自定义配置proxy
- C++ SDK开发包[2024-11-08] 版本号 0.8.7.4
  - 修复已知bug
  - 支持多线程取消已发送的正在进行中的请求

JavaScript-SDK

概述

本文档主要介绍BOS JavaScript SDK的安装和使用。在使用本文档前，您需要先了解BOS的一些基本知识，并已经开通了BOS服务。若您还不了解BOS，可以参考 产品描述和开发者指南。

BOS 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BOS存储服务。

安装SDK

环境差异

您可以在Node.js环境和浏览器环境中使用JavaScript SDK，但受限于浏览器环境本身的功能限制和安全性限制，在浏览器环境中，SDK的一些功能将无法使用。具体有如下几种情况：

- 本地文件读写：浏览器中无法直接操作本地文件，所以BOS服务的PutObjectFromFile和GetObject等相关API将无法使用。类似的，Node.js环境也没有Blob对象，所以BOS服务的PutObjectFromBlob是专门为浏览器环境而开发的。
- Bucket相关操作：由于浏览器跨域访问的限制，需要在服务端设置相应的CORS配置才能进行跨域访问。百度智能云已支持对特定Bucket内的对象进行CORS配置，但不支持对Bucket级别API的跨域访问，所以ListBuckets等API在浏览器端无法使用，Bucket级别的API请参考Bucket。

Node.js环境

支持的Node.js版本

- Node 4.x 以及更高版本

安装步骤

1. JavaScript包已经上传npm管理器，直接使用npm安装SDK的开发包：

   ```
   npm install @baiducloud/sdk
   ```

2. 然后在你的程序中使用：

```
  var sdk = require('@baiducloud/sdk');
var BosClient = sdk.BosClient;

const config = {
    endpoint: <EndPoint>,         //传入Bucket所在区域域名
    credentials: {
        ak: <AccessKeyID>,         //您的AccessKey
        sk: <SecretAccessKey>       //您的SecretAccessKey
    }
};

let bucket = 'my-bucket';
let key = 'hello.js';
let client = new BosClient(config);

client.putObjectFromFile(bucket, key, __filename)
    .then(response => console.log(response))    // 成功
    .catch(error => console.error(error));      // 失败
```

接口会返回一个promise对象。

浏览器环境

支持的浏览器版本

浏览器类型	最低版本
Google Chrome	28.0+
Microsoft Internet Explorer	10.0
Mozilla Firefox	23.0+
Apple Safari	5.1+
Opera	17.0+
Android Browser	4.3+

安装步骤

1. 推荐使用 npm 来安装：

   `npm install @baiducloud/sdk`

   或者通过CDN的方式直接引用

   ```
   <script src="http://bce-cdn.bj.bcebos.com/lib/@baiducloud/sdk/1.0.0-rc.40/baidubce-sdk.bundle.min.js"></script>
   ```

2. 如果通过 npm 来安装，需要使用<script>来加载 `baidubce-sdk.bundle.min.js`到页面：

   ```
   <script src="node_modules/@baiducloud/sdk/dist/baidubce-sdk.bundle.min.js"></script>
   ```

   当 js 加载到页面之后，可以用过全局变量 `baidubce.sdk`来使用 sdk 的功能。

3. 然后在你的程序中使用：

```
const config = {
  endpoint: <EndPoint>,        //传入Bucket所在区域域名
  credentials: {
    ak: <AccessKeyID>,         //您的AccessKey
    sk: <SecretAccessKey>      //您的SecretAccessKey
  }
};

let bucket = 'my-bucket';
let key = 'hello.js';
let client = new baidubce.sdk.BosClient(config);

client.putObjectFromString(bucket, key, 'hello world')
    .then(response => console.log(response))   // 成功
    .catch(error => console.error(error));     // 失败
```

初始化

确定Endpoint

在确认您使用SDK时配置的Endpoint时，可先阅读开发人员指南中关于BOS访问域名的部分，理解Endpoint相关的概念。百度智能云目前开放了多区域支持，请参考区域选择说明。

region与endpoint参考如下链接

https://cloud.baidu.com/doc/BOS/s/akrqd2wcx

常见示例如下:

区域	访问EndPoint	支持协议
bj	bj.bcebos.com	HTTP,HTTPS
bd	bd.bcebos.com	HTTP,HTTPS
su	su.bcebos.com	HTTP,HTTPS
gz	gz.bcebos.com	HTTP,HTTPS
cd	cd.bcebos.com	HTTP,HTTPS
hkg	hkg.bcebos.com	HTTP,HTTPS
fwh	fwh.bcebos.com	HTTP,HTTPS
fsh	fsh.bcebos.com	HTTP,HTTPS

Node.js端快速入门

1. 初始化一个BosClient。

BosClient是与BOS服务交互的客户端，BOS JavaScript SDK的BOS操作都是通过BosClient完成的。

> **说明：**考虑到安全风险，推荐您初始化一个带有STS验证的BosClient。

示例代码（带有STS验证）：

```
var sdk = require('@baiducloud/sdk');
var BosClient = sdk.BosClient;
var config = {
  credentials: {
    ak: '{accessKeyId}',    // STS服务器下发的临时ak
    sk: '{secretAccessKey}' // STS服务器下发的临时sk
  },
  sessionToken: '{sessionToken}',     // STS服务器下发的sessionToken
  endpoint: 'https://bj.bcebos.com'  //需要根据bucket所属局域进行修改
};
```

示例代码（不带STS验证）：

```
var sdk = require('@baiducloud/sdk');
var BosClient = sdk.BosClient;
var config = {
credentials: {
    ak: 'ak',  //您的AK
    sk: 'sk'   //您的SK
  },
endpoint: 'https://bj.bcebos.com'  //需要根据bucket所属局域进行修改
};
```

2. 新建一个Bucket。

   Bucket是BOS上的命名空间，相当于数据的容器，可以存储若干数据实体（Object）。在您上传数据前，必须先创建一个Bucket。

   示例代码：

```
let newBucketName = <BucketName>;    // 新建一个Bucket，指定Bucket名称
client.createBucket(newBucketName)
  .then(function() {
      // 创建完成，添加您自己的代码；
   })
   .catch(function(error) {
      // 创建失败，添加您自己的代码，处理异常
   });
```

3. 上传Object。

   Object是BOS中最基本的数据单元，您可以把Object简单的理解为文件。对于一个简单的Object的上传，BOS为您提供了四种方式：文件形式上传、数据流形式上传、二进制串上传和字符串上传。

   示例代码：

```
function done(response) {
  // 上传完成
}
function fail(fail) {
  // 上传失败
}

// 以字符串形式上传
client.putObjectFromString(bucket, object, 'hello world')
  .then(done)
  .catch(fail);

// 以buffer形式上传
var buffer = new Buffer('hello world'); client.putObject(bucket, object, buffer)
  .then(done)
  .catch(fail);

// 以文件形式上传，仅支持Node.js环境
client.putObjectFromFile(bucket, object, <path-to-file>)
  .then(done)
  .catch(fail);

// 以blob对象形式上传，仅支持浏览器环境
client.putObjectFromBlob(bucket, object, <blob对象>)
  .then(done)
  .catch(fail);
```

4. 查看Bucket中的Object

   您可以参考如下代码来查看Bucket中的Object列表：

```
client.listObjects(<bucketName>)
.then(function (response) {
    var contents = response.body.contents;
    for (var i = 0, l = contents.length; i < l; i++) {
        console.log(contents[i].key);
    }
})
.catch(function (error) {
    // 查询失败
});
```

新建BosClient

BosClient是BOS服务的JavaScript客户端，为开发者与BOS服务进行交互提供了一系列的方法。在使用SDK发起对BOS的请求前，您需要初始化一个BosClient实例，并对它进行一些必要设置。

- 浏览器端： `let BosClient = baidubce.sdk.BosClient`
- Node.js端： `import {BosClient} from '@baiducloud/sdk'`

通过STS方式访问BOS

> **说明：**移动终端是一个不受信任的环境，把 AccessKeyId 和 SecretAccessKey 直接保存在终端用来加签请求，存在极高的风险。故推荐您使用STS鉴权模式，通过STS方式访问BOS。

BOS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BOS，用户先通过sts-client申请一套AK、SK和token，然后将该套参数配置到BosClient中，用户可以参考如下代码新建一个BosClient：

- Node.js端：

```
const {BosClient} = require('@baiducloud/sdk');

var bosConfig = {
    credentials: {
        ak: '{accessKeyId}', // STS服务器下发的临时ak
        sk: '{secretAccessKey}' // STS服务器下发的临时sk
    },
    sessionToken: '{sessionToken}',  // STS服务器下发的sessionToken
    endpoint: 'http://bj.bcebos.com' //需要根据bucket所属局域进行修改
};

var client = new BosClient(bosConfig);
```

- 浏览器端：

```
var bosConfig = {
    credentials: {
        ak: '{accessKeyId}', // STS服务器下发的临时ak
        sk: '{secretAccessKey}' // STS服务器下发的临时sk
    },
    sessionToken: '{sessionToken}',  // STS服务器下发的sessionToken
    endpoint: 'http://bj.bcebos.com'
};

var client = new baidubce.sdk.BosClient(bosConfig);
```

通过AK/SK方式访问BOS

1. 确定EndPoint。EndPoint是指BOS服务在各个区域的域名地址，默认域名为北京：`http://bj.bcebos.com`。

2. 传入您的AK/SK。

3. 将配置好的config传入BosClient。

示例代码

用户可以参考如下代码新建一个BosClient：

```
let config = {
    endpoint: 'https://bj.bcebos.com',
    credentials: {
        ak: <AccessKeyID>,          //您的AK
        sk: <SecretAccessKey>       //您的SK
    }
}
let client = new BosClient(config);
```

> **注意：** EndPoint参数即BOS访问域名，只能用指定的包含区域的域名来进行定义，不指定时默认为北京区域`http://bj.bcebos.com`。BOS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，建议使用`https://bj.bcebos.com`。百度智能云目前开放了多区域支持，请参考区域选择说明。

配置自定义域名访问BOS

使用自定义域名 如果希望使用自定义域名作为访问BOS的endpoint，在控制台将自定义域名和BOS某个bucket绑定之后，配置endpoint为自定义域名并打开cname_enabled开关，例如cdn-test.cdn.bcebos.com，配置代码如下：

```
var sdk = require('@baiducloud/sdk');
var BosClient = sdk.BosClient;
var config = {
    credentials: {
        ak: 'ak',       //您的AK
        sk: 'sk'        //您的SK
    },
    endpoint: 'http://cdn-test.cdn.bcebos.com', // 自定义域名
    cname_enabled: true // 使用自定义域名标示
};
```

配置访问域名风格

> @baiducloud/sdk@1.0.1-beta.9及以上版本支持

JavaScript SDK 在1.0.1-beta.9 及以上访问域名默认使用virtual hosting风格，若希望继续使用PathStyle风格的endpoint，可以通过以下设置 pathStyleEnable 为true开启（不建议），具体域名风格规定请参考存储桶域名请求风格

```
var sdk = require('@baiducloud/sdk');
var BosClient = sdk.BosClient;
var config = {
    credentials: {
        ak: 'ak',       //您的AK
        sk: 'sk'        //您的SK
    },
    endpoint: 'https://bj.bcebos.com', // 域名
    pathStyleEnable: true // 使用pathStyle标志
};
```

配置自定义签名函数

> @baiducloud/sdk@1.0.1-beta.4及以上版本支持

JavaScript SDK 支持 STS（Security Token Service）临时授权的方式进行鉴权。服务端生成一组具体特定操作权限、具有一定时效性的临时AK/SK，这组临时的 AK/SK 可以暴露给浏览器端直接使用。用户只需要通过createSignature签名函数，使用临时的AK、SK和sessionToken计算出最新签名，即可访问BOS资源。关于 STS 方面的介绍请参考临时授权访问。

createSignature函数签名：

```
import type {HttpClient} from '@baiducloud/sdk';

// 自定义签名函数
interface SignatureFunction {
  (
    /* 当前的全局鉴权密钥 */
    credentials: {
        ak: string;
        sk: string
    },
    /* 当前HTTP请求的方法, GET,POST,PUT,DELETE,HEAD */
    httpMethod: string,
    /* 当前HTTP请求的path */
    path: string,
    /* 当前HTTP请求的查询字符串 */
    params: Record<string, any>,
    /* 当前HTTP请求的请求头 */
    headers: Record<string, any>,
    /* 当前函数的上下文，通常为HttpClient实例 */
    context: HttpClient
  ): Promise<string>;
}
```

HttpClient.updateConfigByPath可用于更新全局BceConfig，可以在SignatureFunction中通过context.updateConfigByPath发起调用。注意该函数会对当前服务实例上的config直接进行修改，不要破坏原对象引用，避免SignatureFunction失效，函数签名如下：

```
interface UpdateGlobalConfigFunction {
  (
    /* 要设置的属性的key路径，比如"sessionToken"、"credentials.ak"等 */
    path: string;
    /* 更新后的值 */
    value: string,
  ): void;
}
```

操作步骤如下：

1. 使用STS服务的getSessionToken方法生成临时密钥，这一步通常在服务端完成，这里仅作为演示。
2. 声明自定义签名函数createSignature，主要用于判断当前的密钥是否过期，如果过期，则需要再次获取最新的临时密钥和sessionToken，并更新以下信息：
   - 使用context.updateConfigByPath方法将全局的ak、sk和sessionToken更新。
   - 更新请求头x-bce-security-token字段值为最新的sessionToken。
   - 更新请求头x-bce-date为最新的当前时间，因为服务端计算签名可能会带来时间差。

```
import {BosClient, STS, Auth, Q} from '@baiducloud/sdk';

(async function () {
    const ak = '<Your Access Key>';
    const sk = '<Your Secret Access Key>';
    const bucket = '<Your Bucket Name>';
    // 临时密钥有效时间（秒）
    const durationInSeconds = 3600;

    /* 获取临时鉴权密钥 */
    async function generateTempAuth() {
        const stsClient = new STS({
            protocol: 'https',
            region: 'bj',
            endpoint: 'https://sts.bj.baidubce.com',
            credentials: {ak, sk}
        });

        const res = await stsClient.getSessionToken(durationInSeconds, {
            accessControlList: [
                {
                    // 限定服务范围为BOS
                    service: 'bce:bos',
                    // bucket 代表操作对象是bucket
                    // bucket/* 代表操作对象是存储桶内所有object
                    resource: [bucket, `${bucket}/*`],
                    region: 'bj',
                    effect: 'Allow',
                    permission: ['READ', 'WRITE', 'LIST']
                }
            ]
        });

        return res.body;
    }

    let {
        accessKeyId,
        secretAccessKey,
        sessionToken,
        expiration: expiredUTCTime
    } = await generateTempAuth();

    /**
     * 签名计算函数，这里仅作为演示
     */
```

```
    async function createSignature(
        credentials: {ak: string; sk: string},
        httpMethod: string,
        path: string,
        params: Record<string, any>,
        headers: Record<string, any>,
        context: any
    ) {
        let upatedAK = credentials.ak;
        let upatedSK = credentials.sk;
        const now = new Date();

        // 将当前的秒数向上取整，因为getSessionToken接口返回的expiration不带毫秒数
        if (now.getUTCMilliseconds() > 0) {
            // 如果有毫秒，则增加一秒并设置毫秒为0
            now.setUTCSeconds(now.getUTCSeconds() + 1, 0);
        }

        // 如果当前时间大于过期时间，则重新获取临时密钥
        if (now >= new Date(expiredUTCTime)) {
            const tempRes = await generateTempAuth();

            // 更新临时密、过期时间和sessionToken
            upatedAK = tempRes.ak;
            upatedSK = tempRes.sk;
            expiredUTCTime = tempRes.expiration;
            sessionToken = tempRes.sessionToken;

            // 更新全局config信息
            context.updateConfigByPath('credentials.ak', upatedAK);
            context.updateConfigByPath('credentials.sk', upatedSK);
            context.updateConfigByPath('sessionToken', sessionToken);
        }

        // 更新session Token
        headers['x-bce-security-token'] = sessionToken;

        // 更新当前时间，遵循ISO8601规范，格式如2016-04-06T08:23:49Z。
        const revisionTimestamp = Date.now() + (context?.timeOffset || 0);
        headers['x-bce-date'] = new Date(revisionTimestamp).toISOString().replace(/\.\d+Z$/, 'Z');

        // 计算最新签名并返回，格式为string
        return Q.fcall(function () {
            var auth = new Auth(upatedAK, upatedSK);
            return auth.generateAuthorization(httpMethod, path, params, headers, revisionTimestamp / 1000);
        });
    }

    // 生成BOS SDK服务实例
    const bosClient = new BosClient({
        protocol: 'https',
        endpoint: 'https://bj.bcebos.com',
        credentials: {
          ak: accessKeyId,
          sk: secretAccessKey
        },
        sessionToken: sessionToken,
        createSignature
    });
})();
```

动态切换endpoint

> `@baiducloud/sdk@1.0.1-beta.9`及以上版本支持

JavaScript SDK 支持 调用API时动态切换endpoint，有两种方案可供用户选择：

1. 提供属性`region`作为options的config参数的属性，进行endpoint切换

2. 提供返回请求endpoint的自定义函数`customGenerateUrl`作为options的config参数的属性，其中`customGenerateUrl`可使用参数`bucketName`、`region`

`createSignature`函数签名：

```
import {BosClient} from '@baiducloud/sdk';

// 生成BOS SDK服务实例
const bosClient = new BosClient({
    protocol: 'https',
    endpoint: 'https://bj.bcebos.com',
    credentials: {
      ak: accessKeyId,
      sk: secretAccessKey
    },
});

// 提供region，更新endpoint
bosClient.getBucketStorageclass(CDBucket, {config: {region: 'cd'}}).then(res => {
    // do something
}).catch(err => {})

// 自定义生成url函数
const customGenerateUrl = function (bucketName, region) {
    const protocol = 'https';
    return `${protocol}://${bucketName}.${region}.bcebos.com`;
}
bosClient.getBucketStorageclass(CDBucket, {config: {customGenerateUrl: customGenerateUrl}}).then(res => {
    // do something
}).catch(err => {})
```

Bucket管理

概述

Bucket既是BOS上的命名空间，也是计费、权限控制、日志记录等高级功能的管理实体。

- Bucket名称在所有区域中具有全局唯一性，且不能修改。

> **说明：**
>
> 百度智能云目前开放了多区域支持，请参考区域选择说明。

- 存储在BOS上的每个Object都必须包含在一个Bucket中。
- 一个用户最多可创建100个Bucket，但每个Bucket中存放的Object的数量和大小总和没有限制，用户不需要考虑数据的可扩展性。

新建Bucket

新建Bucket

- 基本流程

1. 创建BosClient。
2. 执行createBucket()方法，你需要提供Bucket的名字。

- 示例代码

```
let newBucketName = <BucketName>;    // 新建一个Bucket，指定Bucket名称
client.createBucket(newBucketName)
  .then(function() {
        // 创建完成，添加您自己的代码；
  })
  .catch(function(error) {
       // 创建失败，添加您自己的代码，处理异常
  });
```

> **注意：**
>
> 1. 由于Bucket的名称在所有区域中是唯一的，所以需要保证BucketName不与其他所有区域上的BucketName相同。
> 2. Bucket的命名有以下规范：
>
> - 只能包括小写字母，数字，短横线（-）。
> - 必须以小写字母或者数字开头。
> - 长度必须在4-63字节之间。
>
> 3. 通过上述代码创建的bucket，权限是私有读写，存储类型是标准类型（Standard）

设置Bucket存储类型

设置Bucket存储类型 新建的Bucket默认是标准存储类型，可通过如下代码设置或获取指定Bucket的存储类型：

- 基本流程

1. 创建BosClient
2. 调用putBucketStorageclass()方法

- 示例代码

```
client.putBucketStorageclass(<BucketName>，<storageClass>)
.then(function() {
    // 设置完成
})
.catch(function(error) {
    // 设置失败
});
```

- 请求参数

参数	说明
BucketName	存储桶名称
storageClass	存储类型，支持： 'STANDARD'（标准存储类型）、'STANDARD_IA'（低频存储）、'ARCHIVE'（归档存储）、'COLD'（冷存储）、'MAZ_STANDARD'（标准存储-多AZ）、'MAZ_STANDARD_IA'（低频存储-多AZ）

查看Bucket列表

查看Bucket列表

- 基本流程

  1. 创建BosClient。
  2. 执行listBuckets()方法。

- 示例代码

  如下代码可以列出用户所有的Bucket：

```
client.listBuckets()
.then(function(response) {
    (response.body.buckets || []).forEach(function (bucket) { console.log(bucket.name) })
    })
.catch(function() {
    // 查询失败，添加您自己的代码，处理异常
});
```

- 返回参数

listBuckets方法返回的解析类中可供调用的参数如下：

参数	说明
owner	Bucket Owner信息
+id	Bucket Owner的用户ID
+displayName	Bucket Owner的名称
buckets	存放多个Bucket信息的容器
bucket	存放一个Bucket信息的容器
+name	Bucket名称
+createDate	Bucket创建时间
+location	Bucket所属的区域

判断Bucket是否存在

判断Bucket是否存在

- 基本流程

  1. 创建BosClient。
  2. 执行doesBucketExist()方法。

- 示例代码

```
client.doesBucketExist(<BucketName>)          //指定Bucket名称
.then(function(response) {
    if(response) {
        console.log('Bucket exists');
    }
    else {
        console.log('Bucket not exists');
    }
})
.catch(function() {
    // 查询失败，添加您自己的代码，处理异常
});
```

Bucket权限管理

Bucket权限控制

设置Bucket的访问权限

- 基本流程

  1. 创建BosClient。
  2. 执行setBucketAcl()方法。

- 示例代码 js

```
// 设置Bucket访问权限为private
client.setBucketCannedAcl(<BucketName>, 'private')
    .then(function() {
```

```
    // 设置完成
  })
  .catch(function(error) {
    // 设置失败
  });
```

setBucketCannedAcl方法请求参数：

参数	说明
BucketName	Bucket名字
CannedAcl	Bucket访问权限

> **注意**
>
> cannedAcl包含三个值： private 、 public-read 、 public-read-write ，它们分别对应相关权限。具体内容可以参考《BOS API文档 使用CannedAcl方式的权限控制》。

设置指定用户对Bucket的访问权限

- 基本流程
  1. 创建BosClient。
  2. 执行setBucketAcl()方法。
- 示例代码

```
let grant_list = [
  {
    'grantee': [
      {'id': <UserID1>},        // 授权给特定用户1
      {'id': <UserID2>},        // 授权给特定用户2
    ],
    'permission': ['FULL_CONTROL']  // 设置权限为FULL_CONTROL
  },
  {
    'grantee': [
      {'id': <UserID3>}         // 授权给特定用户3
    ],
    'permission': ['READ']      // 设置权限为READ
  }
];
client.setBucketAcl(<BucketName>, grant_list)
  .then(function() {
    // 设置完成
  })
  .catch(function(error) {
    // 设置失败
  });
```

执行setBucketAcl方法请求参数：

参数	说明
BucketName	Bucket名字
accessControlList	Bucket权限列表
+grantee	设置权限的用户id列表
+permission	设置的权限

> **注意**
>
> permission中的权限设置包含三个值：READ、WRITE、FULL_CONTROL，它们分别对应相关权限。具体内容可以参考《BOS API文档 上传ACL文件方式的权限控制》。

设置STS临时token权限

对于通过STS方式创建的临时访问身份，管理员也可进行专门的权限设定。 STS的简介及设置临时权限的方式可参见临时授权访问。

使用BOS JavaScript SDK设置STS临时token权限可参考通过sts方式访问BOS

查看Bucket的权限

- 基本流程
  1. 创建BosClient。
  2. 执行getBucketAcl()方法。
- 示例代码

```
client.getBucketAcl(<BucketName>)
  .then(function() {
    // 设置完成
  })
  .catch(function(error) {
    // 设置失败
  });
```

执行getBucketAcl方法返回的参数有：

参数	说明
owner	Bucket owner信息
+id	Bucket owner的用户ID
access_control_list	标识Bucket的权限列表
+grantee	标识被授权人
++id	被授权人ID
+permission	标识被授权人的权限
+resource	ACL配置项所影响的资源

查看Bucket所属的区域

Bucket Location即Bucket Region，百度智能云支持的各region详细信息可参见区域选择说明。

- 基本流程

1. 创建BosClient。
2. 执行listBuckets()方法。

- 示例代码

如下代码可以获取该Bucket的Location信息：

```
BosClient.getBucketLocation(<BucketNname>)
  .then(function(response) {
        if(response) {
            console.log(response);
        }
    })
  .catch(err){}
```

Bucket生命周期管理

应用场景

一个数据是有其生命周期的，从创建到归档到删除可以认为是一个完整的循环。创建之初的数据往往需要频繁访问读取，之后迅速冷却归档，最终被删除。生命周期管理就是对象存储服务帮助用户自动化管理数据的生命周期。 通常可以服务于以下场景：

1. 数据达到一定寿命后自动归档或删除。
2. 指定时间执行操作。

新建Lifecycle配置

- 基本流程

1. 创建BosClient。
2. 调用putBucketLifecycle()方法

- 示例代码

```
  const rules = {
   id: 'rules1',
   status: 'enabled',
   action: {
     name: 'Transition',
     storageClass: 'STANDARD_IA',
     resource: ['baidubosmty2/*'],
     condition: {
       time: {
         dateGreaterThan: 'XXXX-XX-XXTXX:XX:XXZ'
       }
     }
   }
}
client.putBucketLifecycle(<bucketName>,<rules>)
 .then(function(response){
    // 操作
})
.catch(err){}
```

> **注意：**
>
> 1. 只有Bucket的Owner拥有full control才能够进行此操作。
> 2. "resource"指明规则对哪些资源生效。例如，对samplebucket里以prefix/为前缀的Object生效：`samplebucket/prefix/*`；对samplebucket里所有Object生效：`samplebucket/*`。 生命周期管理功能相关参数的详细解释及配置注意事项请参见PutBucketLifecycle接口。

读取Bucket的Lifecycle配置

- 基本流程

1. 创建BosClient。
2. 调用getBucketLifecycle()方法

- 示例代码

```
client.getBucketLifecycle(<bucketName>)
.then(function(response){
  if (resnpose) {
    console.log('lifecycle rules', response)
  }

})
.catch(err){}
```

删除Bucket Lifecycle

- 基本流程

1. 创建BosClient。

2. 调用deleteBucketLifecycle()方法

- 示例代码

```
client.deleteBucketLifecycle(<bucketName>)
.then(function(response){
  // 操作
})
.catch(err){}
```

Bucket日志管理

应用场景 1. 创建BosClient。 2. 执行doesBucketExist()方法。

期望跟踪访问对 BOS 中某个Bucket的请求时可以对该 Bucket 开启访问日志功能

设置访问日志

本接口用来开启Bucket的访问日志并指定存放日志的Bucket和访问日志的文件前缀。访问日志的规则请参见日志命名规则和日志格式。

- 基本流程

1. 创建BosClient。

2. 执行 putBucketLogging()方法。

- 示例代码

```
  const logInfo = {
    targetBucket: 'targetBucket',
    targetPrefix: 'targetPrefix',
  };
  client.putBucketLogging(<BucketName>,<logInfo>)
    .then(function(response) {
      if(response) {
        console.log('Bucket exists');
      }
      else {
        console.log('Bucket not exists');
      }
    })
    .catch(function() {});
```

- 请求参数

参数	类型	含义	是否必填
BucketName	string	存储桶名称	是
logInfo	Record<{targetBuckett: string, targetPrefix: string}>	日志配置信息	是
+targetBucket	string	指定存放访问日志的Bucket	是
+targetPrefix	string	指定存放访问日志的Bucket	否

获取访问日志配置

本接口用来获取某个Bucket的访问日志配置情况。

- 基本流程

1. 创建BosClient。

2. 执行 getBucketLogging()方法。

- 示例代码

```
  client.getBucketLogging(<BucketName>)
  .then(function(response) {
    if(response) {
      console.log('Bucket exists');
    }
    else {
      console.log('Bucket not exists');
    }
  })
  .catch(function() {});
```

删除访问日志

- 基本流程

1. 创建BosClient。

2. 执行 deleteBucketLogging()方法。

- 示例代码

```
client.deleteBucketLogging(<BucketName>)
.then(function(response) {
    if(response) {
        console.log('Bucket exists');
    }
    else {
        console.log('Bucket not exists');
    }
})
.catch(function() {});
```

数据同步配置

Bucket数据同步配置 设置数据同步规则

- 基本流程

1. 创建BosClient。

2. 执行putBucketReplication()方法。

- 示例代码

```
const ReplicationInfo = {
status:"enabled",
resource:[
    bucketName + "/*"
  ],
destination:
    {
        bucket: <dstBucketName>,
        storageClass: "COLD"
    },
replicateHistory:
    {
        bucket: <dstBucketName>,
        storageClass:"COLD"
    },
replicateDeletes:"enabled",
id: <ruleId>
}

client.putBucketReplication(<BucketName>,<ReplicationInfo>)
.then(function(response) {
    if(response) {
      // TODO
    }
})
.catch(function() {});
```

- 参数 ReplicationInfo 结构

参数	是否必须	描述
id	是	replication规则名，id 由数字字母 - _ 组成，不得超过20个字符
status	是	是否生效
resource	是	replication生效前缀，resource的配置形式为｛$BucketName/<生效的对象前缀>｝，必须要以$BucketName+/开头
destination	是	复制的目的端配置
+bucket	是	目的Bucket name
+storageClass	否	目的Object的存储类型。如果保持和源Bucket的存储类型一致，则该参数不需要配置；如果需要单独指定存储类型可以为STANDARD，STANDARD_IA，COLD；如果是多AZ类型bucket，可取值为MAZ_STANDARD_IA和MAZ_STANDARD。
replicateHistory	否	历史文件复制，有该项则认为是开启。开启历史文件复制后，存量的全部Object都同步复制到目的Bucket，复制范围共用resource。
+storageClass	否	目的Object的存储类型。如果保持和源Bucket的存储类型一致，则该参数不需要配置；如果需要单独指定存储类型可以为STANDARD，STANDARD_IA，COLD；如果是多AZ类型bucket，可取值为MAZ_STANDARD_IA和MAZ_STANDARD。
replicateDeletes	是	是否开启删除同步,可以为enabled,disabled 。

注意

- 用户必须是源Bucket的owner且拥有FULL_CONTROL权限，且有目标Bucket的写权限。
- 目标Bucket和源Bucket必须存在。
- 目标Bucket和源Bucket可以是同region下的Bucket，也可以是不同Region下的Bucket。
- 目标Bucket和源Bucket可以是同账户下的bucket，也可以是不同账户下的bucket，但是源账户需要有目的bucket的写权限
- 整个配置的大小不能超过128k，当前bucket下所有规则长度不得超过200KB
- 数据同步暂时不支持归档类型文件的同步，进行数据同步时会忽略归档类型文件。
- 用户最多配置10条replication规则
- 单个规则，最多20个resource

> - id 由数字字母 - _ 组成，不得超过20个字符

获取指定的数据同步规则

用户可通过该方法获取bucket指定id的数据同步信息，包括源Bucket名称、目的Bucket名称、存储类型、是否进行历史复制、数据同步策略、目的region等

- 基本流程

```
1. 创建BosClient。
2. 执行getBucketReplication()方法。
```

- 示例代码

```
```js
client.getBucketReplication(<BucketName>,<RuleId>)
  .then(function(response) {
      if(response) {
          // TODO
      }
      else {}
  })
  .catch(function() {});
```
```

删除指定的数据同步规则

用户可通过该方法删除bucket指定id的数据同步规则

- 基本流程

```
1. 创建BosClient。
2. 执行deleteBucketReplication()方法。
```

- 示例代码

```
```js
client.deleteBucketReplication(<BucketName>,<RuleId>)
  .then(function(response) {
      if(response) {
          // TODO
      }
      else {}
  })
  .catch(function() {});
```
```

列举所有数据同步规则

用户可通过该方法获取bucket所有的replication同步规则

- 基本流程

```
1. 创建BosClient。
2. 执行listBucketReplication()方法。
```

- 示例代码

```
```js
client.listBucketReplication(<BucketName>)
  .then(function(response) {
      if(response) {
          // TODO
      }
      else {}
  })
  .catch(function() {});
```
```

获取指定的数据同步规则的进程状态

用户可通过该方法获取bucket指定id的数据同步复制的进程状态

- 基本流程

```
1. 创建BosClient。
2. 执行getBucketReplicationProgress()方法。
```

- 示例代码

```
```js
client.getBucketReplicationProgress(<BucketName>,<RuleId>)
  .then(function(response) {
      if(response) {
          // TODO
      }
      else {}
  })
  .catch(function() {});
```
```

删除Bucket

删除Bucket

- 基本流程

  1. 创建BosClient。
  2. 执行deleteBucket()方法。

- 示例代码

```
client.deleteBucket(<BucketName>)
.then(function() {
    // 删除完成
})
.catch(function(error) {
    // 删除失败
});
```

> **注意：**
>
> - 如果Bucket不为空（即Bucket中有Object和未完成的三步上传Part存在），则Bucket无法被删除，必须清空Bucket后才能成功删除。
> - 在删除前确认该Bucket没有开通跨区域复制，不是跨区域复制规则中的源Bucket或目标Bucket，否则不能删除。

JavaSDK

版本控制

getBucketVersioning

获取某个Bucket的version权限。

权限要求：需拥有该Bucket的**READ**权限

请求参数：

参数	类型	必填	描述
bucketName	string	是	Bucket名称

请求示例：

```
import {BosClient} from '@baiducloud/sdk';

const client = new BosClient({
  endpoint: process.env.BOS_ENDPOINT,
  credentials: {
    ak: process.env.BOS_AK,
    sk: process.env.BOS_SK
    }
  }
);
const res = await client.getBucketVersioning('multi-test');
```

响应示例：

```
{
  http_headers: {
    date: 'Mon, 23 Jun 2025 02:26:34 GMT',
    'content-type': 'application/json; charset=utf-8',
    'content-length': '20',
    connection: 'close',
    server: 'BceBos',
    'x-bce-debug-id': 'rBQQRQ1IbInIIFkDMuZg7spX+c27z0/4xWnW2NZw4k28Q+3QRHtenJQwKON/oCvHaAB6RoNAeT/sZnUXFajb5g==',
    'x-bce-flow-control-type': '-1',
    'x-bce-is-transition': 'false',
    'x-bce-request-id': 'd12c988c-37f9-4c6c-a6f2-743c6ba405c5',
    'x-bce-version-id': 'null'
  },
  body: {
    status: 'enabled'
  }
}
```

putBucketVersioning 设置指定存储空间（Bucket）的版本控制状态。

> - 在Bucket处于开启版本控制状态下，所有新添加的文件（Object）都将拥有唯一的版本ID，将累积所添加Object的多个版本。
> - 在Bucket处于暂停版本控制状态下，所有新添加Object的版本ID将为null，将不再为此状态下添加的Object累积更多的版本。

权限要求：需拥有该Bucket的**PutBucketVersioning**权限

请求参数：

参数	类型	必填	描述
bucketName	string	是	Bucket名称
status	string	是	目标状态：enabled、suspended或notEnabled

请求示例：

```
import {BosClient} from '@baiducloud/sdk';

const client = new BosClient({
  endpoint: process.env.BOS_ENDPOINT,
  credentials: {
    ak: process.env.BOS_AK,
    sk: process.env.BOS_SK
    }
  }
);
const res = await client.putBucketVersioning('multi-test', 'suspended');
```

响应示例：

```
{
  http_headers: {
    date: 'Mon, 23 Jun 2025 02:34:15 GMT',
    'content-length': '0',
    connection: 'close',
    server: 'BceBos',
    'x-bce-debug-id': 'k41Dyyj/W0JV5KU32pCrl14O2/P5lTt9OfDg5l/6bOl7IE1l8ATz0DH7LQoBu+kKr9K7SH7C1YVZab84Y9x7Zg==',
    'x-bce-flow-control-type': '-1',
    'x-bce-is-transition': 'false',
    'x-bce-request-id': 'eea42f0b-15b1-44c7-9d52-7b08111fca96',
    'x-bce-version-id': 'null'
  },
  body: {}
}
```

文件管理

上传Object

在BOS中，用户操作的基本数据单元是Object。Bucket中的Object数量不限，但单个Object最大允许存储5TB的数据。

Object包含Key、Meta和Data。其中：

- Key是Object的名字；
- Meta是用户对该Object的描述，由一系列Name-Value对组成；
- Data是Object的数据。

BOS JavaScript SDK提供了丰富的文件上传接口，可以通过以下方式上传文件：

- 简单上传
- 追加上传
- 分块上传
- 断点续传上传

Object的命名规范如下：

- 使用UTF-8编码。
- 长度必须在1-1023字节之间。
- 首字母不能为'/'，不能包含'@'字符，'@'用于图片处理接口。

简单上传 在简单上传的场景中，JS SDK 支持以数据流方式、以字符串方式、以指定文件形式（仅支持Node.js环境）、以blob对象形式（仅支持浏览器环境）执行Object上传。 分别对应 **putObject** 、**putObjectFromString** 、**putObjectFromFile** 、**putObjectFromBlob** 方法。

- 基本流程
  1. 创建BosClient。
  2. 调用putObject()相关方法
- 示例代码

```
function done(response) {
  // 上传完成
}
function fail(fail) {
  // 上传失败
}
// 以buffer形式上传
var buffer = new Buffer('hello world');
client.putObject(bucket, object, buffer)
  .then(done)
  .catch(fail);

// 以字符串形式上传
client.putObjectFromString(bucket, object, 'hello world')
  .then(done)
  .catch(fail);

// 以文件形式上传，仅支持Node.js环境
client.putObjectFromFile(bucket, object, <path-to-file>)
  .then(done)
  .catch(fail);

// 以blob对象形式上传，仅支持浏览器环境
client.putObjectFromBlob(bucket, object, <blob对象>)
  .then(done)
  .catch(fail);
```

> **说明**：Object以文件的形式上传到BOS中，putObject函数支持不超过5GB的Object上传。在putObject请求处理成功后，BOS会在Header中返回Object的ETag作为文件标识。

追加上传 上文介绍的简单上传方式，创建的Object都是Normal类型，用户不可再进行追加写，这在日志、视频监控、视频直播等数据复写较频繁的场景中使用不方便。

正因如此，百度智能云BOS特别支持了AppendObject，即以追加写的方式上传文件。通过AppendObject操作创建的Object类型为Appendable Object，可以对该Object追加数据。AppendObject大小限制为0~5G。

```
 let bucketName = "yourbucket";
let appendKey = "appendObjectKey";

// 首次上传时，offset设置为null
client.appendObjectFromString(bucketName,appendKey,"firstContent",null)
.then(function(response){
 // 从响应头中获取offset偏移量
  var offset = + response.http_headers['x-bce-next-append-offset'];
  // 第二次追加上传时，指定上面获取到的offset传入
  client.appendObjectFromString(bucketName, appendKey, "appendContent", offset);
});
```

分块上传 除了通过putObject()方法上传文件到BOS以外，BOS还提供了另外一种上传模式：分块上传（Multipart Upload）。用户可以在如下的应用场景内（但不仅限于此），使用分块上传模式，如：

- 需要支持断点上传。
- 上传超过5GB大小的文件。
- 网络条件较差，和BOS的服务器之间的连接经常断开。
- 需要流式地上传文件。
- 上传文件之前，无法确定上传文件的大小。

分块上传比直接上传稍微复杂一点，分块上传需要分为三个阶段：

- 开始上传（initiateMultipartUpload）
- 上传分块（uploadPartFromBlob）
- 上传完成（completeMultipartUpload）

浏览器端代码示例

对文件进行分块

```
  let options = {
  'Content-Type': 'application/json', // 添加http header
  'Cache-Control': 'public, max-age=31536000', // 指定缓存指令
  'Content-Disposition': 'attachment; filename="example.jpg"', // 指示回复的内容该以何种形式展示

  'x-bce-meta-foo1': 'bar1', // 添加自定义meta信息
  'x-bce-meta-foo2': 'bar2', // 添加自定义meta信息
  'x-bce-meta-foo3': 'bar3' // 添加自定义meta信息
};
let PART_SIZE = 5 * 1024 * 1024; // 指定分块大小

function getTasks(file, uploadId, bucketName, key) {
    let leftSize = file.size;
    let offset = 0;
    let partNumber = 1;

    let tasks = [];

    while (leftSize > 0) {
        let partSize = Math.min(leftSize, PART_SIZE);
        tasks.push({
            file: file,
            uploadId: uploadId,
            bucketName: bucketName,
            key: key,
            partNumber: partNumber,
            partSize: partSize,
            start: offset,
            stop: offset + partSize - 1
        });

        leftSize -= partSize;
        offset += partSize;
        partNumber += 1;
    }
    return tasks;
}
```

处理每个分块的上传逻辑

```
  function uploadPartFile(state, client) {
    return function(task, callback) {
        let blob = task.file.slice(task.start, task.stop + 1);
        client.uploadPartFromBlob(task.bucketName, task.key, task.uploadId, task.partNumber, task.partSize, blob)
            .then(function(res) {
                ++state.loaded;
                callback(null, res);
            })
            .catch(function(err) {
                callback(err);
            });
    };
}
```

初始化uploadID，开始上传分块，并完成上传

```
 let uploadId;
client.initiateMultipartUpload(bucket, key, options)
    .then(function(response) {
        uploadId = response.body.uploadId; // 开始上传，获取服务器生成的uploadId

        let deferred = sdk.Q.defer();
        let tasks = getTasks(blob, uploadId, bucket, key);
        let state = {
            lengthComputable: true,
            loaded: 0,
            total: tasks.length
        };

        // 为了管理分块上传，使用了async（https://github.com/caolan/async）库来进行异步处理
        let THREADS = 2; // 同时上传的分块数量
        async.mapLimit(tasks, THREADS, uploadPartFile(state, client), function(err, results) {
            if (err) {
                deferred.reject(err);
            } else {
                deferred.resolve(results);
            }
        });
        return deferred.promise;
    })
    .then(function(allResponse) {
        let partList = [];
        allResponse.forEach(function(response, index) {
            // 生成分块清单
            partList.push({
                partNumber: index + 1,
                eTag: response.http_headers.etag
            });
        });
        return client.completeMultipartUpload(bucket, key, uploadId, partList); // 完成上传
    })
    .then(function (res) {
        // 上传完成
    })
    .catch(function (err) {
        // 上传失败，添加您的代码
        console.error(err);
    });
```

Node.js端代码示例

对文件进行分块，并初始化UploadID，上传分块

```
  let options = {
  'Content-Type': 'application/json', // 添加http header
  'Cache-Control': 'public, max-age=31536000', // 指定缓存指令
  'Content-Disposition': 'attachment; filename="example.jpg"', // 指示回复的内容该以何种形式展示

  'x-bce-meta-foo1': 'bar1', // 添加自定义meta信息
  'x-bce-meta-foo2': 'bar2', // 添加自定义meta信息
  'x-bce-meta-foo3': 'bar3' // 添加自定义meta信息
};
let PART_SIZE = 5 * 1024 * 1024;          // 指定分块大小
let uploadId;
client.initiateMultipartUpload(bucket, key, options)
    .then(function(response) {
        uploadId = response.body.uploadId; // 开始上传，获取服务器生成的uploadId
        let deferred = sdk.Q.defer();
        let blob = {
            // 使用fs文件库获取文件大小
            size: fs.statSync(localFileName).size,
            filename: localFileName
        }
        let tasks = getTasks(blob, uploadId, bucket, key);
        let state = {
            lengthComputable: true,
            loaded: 0,
            total: tasks.length
        };
        // 为了管理分块上传，使用了async（https://github.com/caolan/async）库来进行异步处理
        let THREADS = 2; // 同时上传的分块数量
        async.mapLimit(tasks, THREADS, uploadPartFile(state, client), function(err, results) {
            if (err) {
                deferred.reject(err);
            } else {
                deferred.resolve(results);
            }
        });
        return deferred.promise;
    })
    .then(function(allResponse) {
        let partList = [];
        allResponse.forEach(function(response, index) {
            // 生成分块清单
            partList.push({
                partNumber: index + 1,
                eTag: response.http_headers.etag
            });
        });
        return client.completeMultipartUpload(bucket, key, uploadId, partList); // 完成上传
    })
    .then(function (res) {
        // 上传完成
    })
    .catch(function (err) {
        // 上传失败，添加您的代码
        console.error(err);
    });

function getTasks(file, uploadId, bucketName, key) {
    let leftSize = file.size;
    let offset = 0;
    let partNumber = 1;
    let tasks = [];
    while (leftSize > 0) {
        let partSize = Math.min(leftSize, PART_SIZE);
        tasks.push({
            file: file.filename,
            uploadId: uploadId,
            bucketName: bucketName,
            key: key,
            partNumber: partNumber,
            partSize: partSize,
            start: offset,
            stop: offset + partSize - 1
        });

        leftSize -= partSize;
        offset += partSize;
        partNumber += 1;
    }
    return tasks;
}


function uploadPartFile(state, client) {
    return function(task, callback) {
        console.log("task: ", task)
        return client.uploadPartFromFile(task.bucketName, task.key, task.uploadId, task.partNumber, task.partSize, task.file , task.start)
            .then(function(res) {
                ++state.loaded;
                console.log("ok")
                callback(null, res);
            })
            .catch(function(err) {
                console.log("bad")
                callback(err);
            });
    };
}
```

取消分块上传事件

用户可以使用abortMultipartUpload方法取消分块上传。

```
client.abortMultipartUpload(<BucketName>, <Objectkey>, <UploadID>);
```

获取未完成的分块上传事件

用户可以使用listMultipartUploads方法获取Bucket内未完成的分块上传事件。

```
client.listMultipartUploads(<bucketName>)
  .then(function (response) {
      // 遍历所有上传事件
      for (var i = 0; i < response.body.multipartUploads.length; i++) {
          console.log(response.body.multipartUploads[i].uploadId);
      }
  });
```

获取所有已上传的块信息

用户可以使用listParts方法获取某个上传事件中所有已上传的块。

```
client.listParts(<bucketName>, <key>, <uploadId>)
  .then(function (response) {
      // 遍历所有上传事件
      for (var i = 0; i < response.body.parts.length; i++) {
          console.log(response.body.parts[i].partNumber);
      }
  });
```

断点续传上传

当用户向BOS上传大文件时，如果网络不稳定或者遇到程序崩等情况，则整个上传就失败了，失败前已经上传的部分也作废，用户不得不重头再来。这样做不仅浪费资源，在网络不稳定的情况下，往往重试多次还是无法完成上传。基于上述场景，BOS提供了断点续传上传的能力，主要利用了分块上传的能力，将待上传的文件拆分成多个分块（Part），然后分别上传这些分块，当所有分块全部上传完成后，BOS将请求者上传的所有分块组合成完整的Object。

putSuperObject

> 1.0.1-beta.2及以上版本支持

JavaScript SDK提供了putSuperObject方法，对分块上传相关的API做了高级封装，支持分上传任务暂停、恢复、取消、分片并发数设置，失败重试等功能。

请求头 除公共头域外，无其他特殊请求头

初始化参数

参数名称	说明	类型	必填	默认值	示例值
bucketName	存储桶名称	string	是	-	"bucket001"
objectName	上传后对象名称	string	是	-	"file001"
data	上传数据，类型为string时表示文件路径，还支持Buffer和Blob对象	string | Buffer | Blob	是	-	-
StorageClass	文件存储类型	"STANDARD" | "STANDARD_IA" | "COLD" | "ARCHIVE" | "MAZ_STANDARD" | "MAZ_STANDARD_IA"	否	"STANDARD"	"STANDARD"
chunkSize	上传文件分片默认体积，单位为bytes	number	否	5 * 1024 ** 2（5MB）	1048576
partConcurrency	分片并发数量	number	否	5	5
ContentLength	文件大小，单位为bytes，非必填，内部会基于自动计算	string	否	-	1048576
ContentType	文件媒体类型，非必填，内部会基于objectName字段自动生成	string	否	-	"application/x-gtar"
uploadId	分片上传任务ID，非必填，当需要从断点恢复任务时，需要传入该参数	string	否	-	"a44cc9bab11cbd156984767aad637851"
onProgress	上传进度回调函数	ProgressCallback	否	-	-
onStateChange	任务状态变化回调函数	StateChangeCallback	否	-	-

ProgressCallback

```
type ProgressCallback = (
  /* 当前上传速度 */
  speed: string,
  /* 上传进度，保留4位小数 */
  progress: number,
  /* 上传进度-百分比 */
  percent: string,
  /* 已上传字节数 */
  uploadedBytes: number,
  /* 文件总字节数 */
  totalBytes
) => void
```

StateChangeCallback

```
type StateChangeCallback = (
  /* 状态 */
  state: string,
  /* 如果上传失败会传递失败的message，上传成功会返回上传后的地址 */
  options: {
    message: string;
    data: Record<string, any> | null
  }
) => void;
```

状态

- `"inited"`：任务初始化完成
- `"running"`：任务队列运行中
- `"paused"`：任务队列暂停
- `"completed"`：任务完成，上传结束
- `"cancelled"`：任务取消，上传结束
- `"failed"`：任务异常，上传结束

Node.js端代码示例

```
const sdk = require('@baiducloud/sdk');
const client = new sdk.BosClient({
  endpoint: 'http://bj.bcebos.com',
  credentials: {
    ak: "<Your Access Key>",
    sk: "<Your Secret Key>"
  }
});
// 存储桶名称
const bucketName = "<Your Bucket Name>";
// 上传后文件名称
const objectName = 'demo.tgz';
// 本地文件路径
const data = '/Mock/path/to/local/file/demo.tgz';

// 初始化上传任务
const SuperUploadTask = client.putSuperObject({
    // 桶名称
    bucketName,
    // 上传后对象名称
    objectName,
    // 上传数据, 类型为string时表示文件路径
    data,
    // 分片并发数
    partConcurrency: 2,
    // 上传进度回调函数
    onProgress: (options) => {
        const {speed, progress, percent, uploadedBytes, totalBytes} = options;
        console.log(options);
    },
    // 状态变化回调函数
    onStateChange: (state, data) => {
        if (state === 'completed') {
            console.log('上传成功');
        } else if (state === 'failed') {
            console.error('上传失败，失败原因：' + data.message);
        } else if (state === 'cancelled') {
            console.log('上传任务取消');
        } else if (state === 'inited') {
            console.log('上传任务初始化完成');
        } else if (state === 'running') {
            console.log('上传任务开始运行...');
        } else if (state === 'paused') {
            console.log('上传任务已暂停');
        }
    }
});

// 启动上传任务
const tasks = SuperUploadTask.start();
console.log('切分任务: ', tasks);

// 暂停上传任务
setTimeout(() => {
    SuperUploadTask.pause();
}, 5000);

// 恢复上传任务
setTimeout(() => {
    SuperUploadTask.resume();
}, 15000);

// 取消上传任务
setTimeout(async () => {
    const result = SuperUploadTask.cancel();
    console.log(result ? '任务取消成功' : '任务取消失败');
}, 25000);
```

浏览器端代码示例

```
<html>
<head>
```

```
    <meta charset="utf-8" />
    <title>SuperUpload测试</title>
    <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/amis/6.2.2/sdk.min.css" integrity="sha512-
9yikVhRqNeq1ryplzAFKR8CA2uG8V5gYppoDKK4xx+7eoLXJsEm+f9QN0++xqHvr0xxvb93uiipgYk9uEW7RIA==" crossorigin="anonymous" referrerpolicy="no-referrer" />
    <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/amis/6.2.2/cxd.min.css" integrity="sha512-
9yikVhRqNeq1ryplzAFKR8CA2uG8V5gYppoDKK4xx+7eoLXJsEm+f9QN0++xqHvr0xxvb93uiipgYk9uEW7RIA==" crossorigin="anonymous" referrerpolicy="no-referrer" />
    <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/simple-notify@1.0.4/dist/simple-notify.css" />
  </head>
  <body>
    <div id="root"></div>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/jquery/3.7.1/jquery.min.js" integrity="sha512-
v2CJ7UaYy4JwqLDIrZUI/4hqeoQieOmAZNXBeQyjo21dadnwR+8ZaIJVT8EE2iyI610V8e6M8PP2/4hpQINQ/g==" crossorigin="anonymous" referrerpolicy="no-referrer"></script>
    <script src="https://cdn.jsdelivr.net/npm/simple-notify@1.0.4/dist/simple-notify.min.js"></script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/amis/6.2.2/sdk.min.js" integrity="sha512-
BpMIHWCtAUDARuH/qnGH6eBxoOv4lOi4Y9/9f6vnGVDNnPqbWCZoPgk3/KvY6NuOb6cAPFmqVN1R2sXz7Z3VcQ==" crossorigin="anonymous" referrerpolicy="no-referrer"></script>
    <script src="http://bce-cdn.bj.bcebos.com/lib/@baiducloud/sdk/1.0.1-beta.2/baidubce-sdk.bundle.min.js" ></script>
    <script>
      (async function () {
        // 实例化BOS SDK
        const client = new window.baidubce.sdk.BosClient({
          endpoint: 'http://bj.bcebos.com',
          credentials: {
            ak: "<Your Access Key>",
            sk: "<Your Secret Key>"
          }
        });
        // 实例化BOS SDK
        const bucketName = '<Your Bucket Name>';
        // 上传任务实例
        let SuperUpload;
        // 进度信息
        let p1 = 0;

        // 消息弹窗
        function pushNotify({status, title, text}) {
          new Notify({
            status: status || 'success',
            title: title,
            text: text,
            distance: 200,
            position: 'x-center'
          });
        }

        // 获取任务实例
        function getInstance() {
          if (SuperUpload) {
            pushNotify({title: '获取实例信息', text: '实例信息获取成功'});
          } else {
            pushNotify({status: 'error', title: '获取实例信息', text: '实例信息获取失败'});
          }
        }

        async function start() {
          const files = $('#file')?.[0]?.files;

          if (!files || files.length === 0) {
            pushNotify({status: 'error', title: '上传', text: '请先选择文件'});
            return;
          }

          const file = files[0];
          let reader = new FileReader();
          let rs = reader.readAsArrayBuffer(file);
          let blob = null;

          // 将文件读取为Blob对象
          reader.onload = async e => {
            if (typeof e.target.result === 'object') {
              blob = new Blob([e.target.result]);
            } else {
              blob = e.target.result;
            }

            SuperUpload = client.putSuperObject({
              bucketName,
              objectName: file.name,
              ContentLength: file.size,
              ContentType: file.type,
              data: blob,
              partConcurrency: 2,
              onProgress: options => {
                const {speed, progress, uploadedBytes, totalBytes} = options;

                console.log(options);
                amisScoped.updateProps({
                  data: {p1: progress * 100}
                });
              },
              onStateChange: (state, data) => {
                if (state === 'completed') {
                  pushNotify({title: '上传', text: '上传成功'});
                  console.log(data);
                } else if (state === 'failed') {
                  pushNotify({status: 'error', title: '上传失败', text: data.message});
                } else if (state === 'cancelled') {
                  pushNotify({title: '上传', text: '上传任务取消'});
                } else if (state === 'inited') {
                  pushNotify({title: '上传', text: '上传任务初始化完成'});
                } else if (state === 'running') {
```

```
        } else if (state === 'running') {
          pushNotify({title: '上传', text: '上传任务开始运行...'});
        } else if (state === 'paused') {
          pushNotify({title: '上传', text: '上传任务已暂停'});
        }
      }
    });

    const tasks = await SuperUpload.start();
    console.log('切片列表: ', tasks);
  };
}

function pause() {
  if (SuperUpload) {
    SuperUpload.pause();
  } else {
    pushNotify({status: 'error', title: '暂停', text: '实例不存在'});
  }
}

function resume() {
  if (SuperUpload) {
    SuperUpload.resume();
  } else {
    pushNotify({status: 'error', title: '恢复', text: '实例不存在'});
  }
}

async function cancel() {
  if (SuperUpload) {
    const result = await SuperUpload.cancel();

    if (result) {
      SuperUpload = undefined;
      p1 = 0;

      amisScoped.updateProps({
        data: {p1: 0}
      });
    }
  } else {
    pushNotify({status: 'error', title: '取消任务', text: '实例不存在'});
  }
}

const amis = amisRequire('amis/embed');
const amisJSON = {
  type: 'page',
  body: [
    {
      type: 'alert',
      title: 'CORS配置',
      body: '如果接口报CORS错误，请开启对应存储桶的<a href="https://cloud.baidu.com/doc/BOS/s/Dk6kqw1g8", target="__blank">CORS配置</a>',
      level: 'warning',
      showIcon: true,
      className: 'mb-3'
    },
    {
      type: 'custom',
      name: 'file',
      onMount: (dom, value, onChange, props) => {
        const input = document.createElement('input');
        input.setAttribute('type', 'file');
        input.setAttribute('id', 'file');

        dom.appendChild(input);
      }
    },
    {
      type: 'progress',
      value: '${p1}',
      style: {
        marginTop: '20px',
        width: '500px'
      }
    },
    {
      type: 'flex',
      justify: 'flex-start',
      style: {
        marginTop: '20px'
      },
      items: [
        {
          type: 'button',
          level: 'primary',
          label: '获取实例',
          style: {
            marginRight: '10px'
          },
          onClick: getInstance
        },
        {
          type: 'button',
          level: 'primary',
          label: '上传',
          style: {
            marginRight: '10px'
          },
```

```
                onClick: start
              },
              {
                type: 'button',
                level: 'primary',
                label: '暂停',
                style: {
                  marginRight: '10px'
                },
                onClick: pause
              },
              {
                type: 'button',
                level: 'primary',
                label: '恢复',
                style: {
                  marginRight: '10px'
                },
                onClick: resume
              },
              {
                type: 'button',
                level: 'danger',
                label: '取消',
                onClick: cancel
              }
            ]
          }
        ]
      };
      const amisScoped = amis.embed('#root', amisJSON, {data: {p1}});
    })()
  </script>
 </body>
</html>
```

设定Object的Http Header和自定义Meta数据

SDK本质上是调用后台的HTTP接口，因此BOS服务允许用户自定义Http Header。同时也允许用户对要上传的Object添加自定义Meta信息。以putObjectFromFile()函数为例，可以用以下代码来处理：

- 示例代码

```
let options = {
 'Content-Type': 'application/json', // 添加http header
 'Cache-Control': 'public, max-age=31536000', // 指定缓存指令
 'Content-Disposition': 'attachment; filename="example.jpg"', // 指示回复的内容该以何种形式展示

 'x-bce-meta-foo1': 'bar1', // 添加自定义meta信息
 'x-bce-meta-foo2': 'bar2', // 添加自定义meta信息
 'x-bce-meta-foo3': 'bar3' // 添加自定义meta信息
}
client.putObjectFromFile(bucket, object, <path-to-file>, options)
 .then(done)
 .catch(fail);
```

> **注意：**自定义Meta信息的key需要以x-bce-meta-开头。

获取上传进度 JavaScript SDK支持在上传过程中实时提供上传进度信息，可通过监听process事件获取相关信息，所有上传相关接口均支持该功能。

- 示例代码：以putObjectFromBlob接口为例

```
 // 以blob对象形式上传，仅支持浏览器环境
client.putObjectFromBlob(bucket, object, <blob对象>)
 .then(done)
 .catch(fail);

client.on('progress', function() {
  // do something
})
```

同步回调 JavaScript SDK支持BOS服务端同步回调接口，通过请求头中添加`x-bce-process`或者在url的query中添加`x-bce-process`，指定回调的服务器地址和相关参数配置，就可以在上传完成后，让BOS服务端主动调用回调接口来达到通知客户目的。

> 目前仅支持普通上传（PutObject）和 完成分片上传（CompleteMultipartUpload）

- 方法一:

> 使用`callback`参数，SDK会帮助你处理参数并添加到请求头中

```
try {
 const res = await client.putObjectFromString('bucketName', 'fileName', 'demo-string', {
     callback: {
       urls: ["https://www.test.com/callback"],
       vars: {name: 'baidu'},
       encrypt: 'config',
       key: 'callback1'
     }
 });

 /* callback result */
 console.log(res.body.callback.result);
} catch (e) {
 /* callback error code */
 console.error(res.body.callback.code);
 /* callback error message */
 console.error(res.body.callback.message);
}
```

- 方法二:

> 自行处理参数，将"x-bce-process"参数和值添加到请求头中

```
try {
 const res = await client.putObjectFromString('bucketName', 'fileName', 'demo-string', {
     'x-bce-process': 'callback/callback,u_WyJodHRwczovL3d3dy50ZXN0LmNvbS9jYWxsYmFjayJd,m_sync,v_eyJuYW1lIjoiYmFpZHUifQ'
 });

 /* callback result */
 console.log(res.body.callback.result);
} catch (e) {
 /* callback error code */
 console.error(res.body.callback.code);
 /* callback error message */
 console.error(res.body.callback.message);
}
```

获取Object

获取Object(仅支持Node.js)

简单的获取Object

用户可以通过如下代码将Object读取到一个流中。

- 基本流程

  1. 创建BosClient。
  2. 执行getObject()。

- 示例代码

```
  let range = '0-100';
client.getObject(<BucketName>, <Key>, range)
    .then(function(response) {
        let buffer = response.body;
    });
```

> **说明：** 设置range为0-100表示只获取0到100字节的数据，您可以用此功能实现文件的分段下载和断点续传。如果不设置range，则获取整个Object。

下载Object到指定路径

用户可以通过如下代码直接将Object下载到指定路径。

- 基本流程

  1. 创建BosClient。
  2. 执行client.getObject( )操作。
  3. Object可以直接下载到指定路径。

- 示例代码

```
  let range = '0-100';
client.getObjectToFile(<bucketName>, <key>, <filePath>, range)
    .then(function() {
        // 下载完成
    });
```

> **说明：** 设置range为0-100表示只获取0到100字节的数据，您可以用此功能实现文件的分段下载和断点续传。如果不设置range，则获取整个Object。filePath为文件完整路径，包含文件名称及类型。

只获取ObjectMetadata

通过 getObjectMetadata() 方法可以只获取ObjectMetadata而不获取Object的实体。

- 示例代码

```
client.getObjectMetadata(<BucketName>, <ObjectKey>)
.then(function(response) {
    console.dir(response.http_headers);
});
```

获取文件URL

用户可以通过如下示例代码获取指定指定Object的URL：

```
url = client.generatePresignedUrl(bucketName, key, timestamp, expirationInSeconds, headers, params, headersToSign, config)
```

说明：

- 用户在调用该函数前，需要手动设置endpoint为所属区域域名。百度智能云目前开放了多区域支持，请参考区域选择说明。目前支持“华北-北京”、“华南-广州”和“华东-苏州”三个区域。北京区域：http://bj.bcebos.com，广州区域：http://gz.bcebos.com，苏州区域：http://su.bcebos.com。
- timestamp为可选参数时间戳，标识URL有效起始时间，缺省值为当前时间。
- expirationInSeconds用来设置URL的有效时长，为可选参数，缺省值为1800秒。如果要设置为永久不失效的时间，可以将expirationInSeconds参数设置为 -1，不可设置为其他负数。
- headersToSign用来设置计算至前面的header list

代码示例如下：

```
 // 获取访问链接
let url = client.generatePresignedUrl(
            bucketName,
            key,
            Math.floor(Date.now()/1000),
            -1
        );

// 获取下载链接
let donwloadUrl = client.generatePresignedUrl(
            bucketName,
            key,
            Math.floor(Date.now()/1000),
            -1,
            {},
            {responseContentDisposition: 'attachment'}
        );
```

查看Bucket中的Object

查看Bucket中的Object

简单查询

查看Bucket中Object列表。

- 基本流程
    1. 创建BosClient。
    2. 执行listObjects()方法。
- 示例代码

```
client.listObjects(<bucketName>)
.then(function (response) {
    var contents = response.body.contents;
    for (var i = 0, l = contents.length; i < l; i++) {
        console.log(contents[i].key);
    }
})
.catch(function (error) {
    // 查询失败
});
```

注意：

- 默认情况下，如果Bucket中的Object数量大于1000，则只会返回1000个Object，并且返回结果中IsTruncated值为True，并返回NextMarker做为下次读取的起点。
- 若想获取更多的Object，可以使用Marker参数分次读取，请参考扩展查询。

扩展查询

用户可以通过设置listObjects参数来完成更多扩展查询操作设置。可以设置的扩展参数如下：

参数名称	说明	默认值
maxKeys	设定此次返回Object的最大个数，不可超过1000。	1000
prefix	设定objectKey的前缀，前缀是指objectKey包含并以prefix的值作为开始。 通常与delimiter配合在查询模拟文件夹中使用。	-
delimiter	是一个分隔符，用来对objectKey进行分层。 通常与prefix配合在查询模拟文件夹中使用。 从prefix开始到第一次出现delimiter字符之间的objectKey称为：commonPrefixes。	-
marker	是一个字符串，用来设定返回结果的起始位置。 设定marker值之后，返回的Object会从marker值之后按字母排序开始返回。	-

- 示例代码 `js // 设置参数 var options = { delimiter: '/', marker: '123' };

```
client.listObjects(<bucketName>, options)
  .then(function (response) {
    var contents = response.body.contents;
    for (var i = 0, l = contents.length; i < l; i++) {
      console.log(contents[i].key);
    }
  })
  .catch(function (error) {
    // 查询失败
  });
```

查询模拟文件夹

由于BOS本身是一个(<Key>,<Value>)的存储系统，所以原则上并不会存在“文件夹”的概念，但您可以通过 delimiter 和 prefix 参数的配合进行文件夹功能模拟。

假设Bucket中有5个文件：bos.jpg，fun/，fun/test.jpg，fun/movie/001.avi，fun/movie/007.avi，可以把 “/” 符号作为分隔符模拟文件夹。

递归列出模拟文件夹下所有文件

可以通过设置 Prefix 参数来获取某个模拟文件夹下所有的文件：

```
// 设置参数
let options = {
prefix: 'fun/' // 递归列出fun目录下的所有文件
};


client.listObjects(<bucketName>, options)
.then(function (response) {
  console.log('Objects:');
  var contents = response.body.contents;
  for (var i = 0, l = contents.length; i < l; i++) {
    console.log(contents[i].key);
  }
})
.catch(function (error) {
  // 查询失败
});
```

输出：

```
Objects:
fun/
fun/movie/001.avi
fun/movie/007.avi
fun/test.jpg
```

查看模拟文件夹下的文件和子文件夹

在 Prefix 和 Delimiter 结合的情况下，可以列出模拟文件夹下的文件和子文件夹：

```
// 设置参数
let options = {
  prefix: 'fun/', // 列出fun目录下的所有文件和文件夹
  delimiter: '/' // "/" 为文件夹的分隔符
};

client.listObjects(<bucketName>, options)
.then(function (response) {
  console.log('Objects:');
  var contents = response.body.contents;
  for (var i = 0, l = contents.length; i < l; i++) {
    console.log(contents[i].key);
  }
  console.log('CommonPrefixs:');
  var commonPrefixes = response.body.commonPrefixes;
  for (i = 0, l = commonPrefixes.length; i < l; i++) {
    console.log(commonPrefixes[i]);
  }
})
.catch(function (error) {
  // 查询失败
});
```

输出：

```
Objects:
fun/
fun/test.jpg

CommonPrefixs:
fun/movie/
```

> **说明：** 返回的结果中， Objects 的列表中给出的是fun文件夹下的文件。而 CommonPrefixs 的列表中给出的是fun文件夹下的所有子文件夹。可以看出 fun/movie/001.avi ， fun/movie/007.avi 两个文件并没有被列出来，因为它们属于 fun 文件夹下的 movie 子文件夹下的文件。

拷贝Object

拷贝Object

简单拷贝Object

- 基本流程

  1. 创建BosClient。
  2. 执行copyObject( )方法。

- 示例代码

```
let options = {
'Content-Length': <file size>, // 添加http header
'Content-Type': 'application/json', // 添加http header
'Cache-Control': 'public, max-age=31536000', // 指定缓存指令
'Content-Disposition': 'attachment; filename="example.jpg"', // 指示回复的内容该以何种形式展示
'x-bce-meta-foo1': 'bar1', // 覆盖自定义meta信息
'x-bce-meta-foo2': 'bar2', // 覆盖自定义meta信息
'x-bce-meta-foo3': 'bar3' // 覆盖自定义meta信息
}
//SrcBucketName, SrcKey为Bucket/Object，DestBucketName, DestKey为拷贝到的目的Bucket/Object
client.copyObject(<SrcBucketName>, <SrcKey>, <DestBucketName>, <DestKey>, options);
```

同步Copy功能

当前BOS的CopyObject接口是通过同步方式实现的。同步方式下，BOS端会等待Copy实际完成才返回成功。同步Copy能帮助用户更准确的判断Copy状态，但用户感知的复制时间会变长，且复制时间和文件大小成正比。

同步Copy方式更符合业界常规，提升了与其它平台的兼容性。同步Copy方式还简化了BOS服务端的业务逻辑，提高了服务效率。

Object设置meta信息 可以通过拷贝Object的方式设置meta信息。

BOS修改Object的Metadata通过拷贝Object实现。即拷贝Object的时候，把目的Bucket设置为源Bucket，目的Object设置为源Object，并设置新的Metadata，通过拷贝自身实现修改Metadata的目的。如果不设置新的Metadata，则报错。

- 示例代码

```
let options = {<meta_key>: <meta_value>}

// SrcBucketName, SrcKey为Bucket/Object
// DestBucketName = SrcBucketName, DestKey = SrcKey 目标与原Bucket/Object相同
client.copyObject(<SrcBucketName>, <SrcKey>, <DestBucketName>, <DestKey>, options);
```

分块拷贝

除了通过CopyObject接口拷贝文件以外，BOS还提供了另外一种拷贝模式——Multipart Upload Copy。用户可以在如下的应用场景内（但不仅限于此），使用Multipart Upload Copy，如：

- 需要支持断点拷贝。
- 拷贝超过5GB大小的文件。
- 网络条件较差，和BOS的服务器之间的连接经常断开。

下面将介绍分步实现三步拷贝。

三步拷贝包含init、“拷贝分块”和complete三步，其中init和complete的操作同分块上传一致。

为了便于理解，下面提供三步拷贝完整代码：

```
 // 准备
let options = {
  'Content-Length': <file.size>, // 添加http header
  'Content-Type': 'application/json', // 添加http header
  'Cache-Control': 'public, max-age=31536000', // 指定缓存指令
  'Content-Disposition': 'attachment; filename="example.jpg"', // 指示回复的内容该以何种形式展示

  'x-bce-meta-foo1': 'bar1', // 添加自定义meta信息
  'x-bce-meta-foo2': 'bar2', // 添加自定义meta信息
  'x-bce-meta-foo3': 'bar3' // 添加自定义meta信息
};
let PART_SIZE = 5 * 1024 * 1024; // 指定分块大小

/** 第一步 init **/
// SrcBucketName, SrcKey为原Bucket/Object
// DestBucketName, DestKey为拷贝到的目的Bucket/Object
client.initiateMultipartUpload(srcBucketName, SrcKey)
    .then(function(response) {
        uploadId = response.body.uploadId; // 获取服务器生成的uploadId标识任务
    })


/** 第二步 分块拷贝 **/
// 获取文件大小
let leftSize = 0;
client.getObjectMetadata(<SrcBucketName>, <SrcKey>).then(response => {
    leftSize = response.http_headers?.['content-length']
});

let tasks = [];
let offset = 0;
let partNumber = 1;

while (leftSize > 0) {
    let partSize = Math.min(leftSize, PART_SIZE);
    tasks.push({
        file: <SrcKey>,
        bucketName: <SrcBucketName>,
        uploadId: uploadId,
        partNumber: partNumber,
        partSize: partSize,
        start: offset,
        stop: offset + partSize - 1
    })
    leftSize -= partSize;
    offset += partSize;
    partNumber+=1;
}
    let partList = [];
    Promise.all(tasks.map(async (task, index) => {
         return client.uploadPartCopy(<SrcBucketName>, <SrcKey>, <DestBucketName>, <DestKey>,uploadId, partNumber, range, <options>).then(response => {
            // do something
        })
    })).then(response => {
        partList = response
            .map(res => ({partNumber: res.partNumber, eTag: res.http_headers.eTag}));
    });


/** 第三步 complete **/
client.completeMultipartUpload(<DestBucketName>, <DestKey>, uploadId, partList)
```

注意:

1. offset参数以字节为单位，为分块的开始偏移位置。
2. size参数以字节为单位，定义每个分块的大小，除最后一个Part以外，其他的Part大小都要大于 5MB。

Object的分块上传

Object的分块上传 除了通过putObject()方法上传文件到BOS以外，BOS还提供了另外一种上传模式：分块上传（Multipart Upload）。用户可以在如下的应用场景内（但不仅限于此），使用分块上传模式，如：

- 需要支持断点上传。
- 上传超过5GB大小的文件。
- 网络条件较差，和BOS的服务器之间的连接经常断开。
- 需要流式地上传文件。
- 上传文件之前，无法确定上传文件的大小。

分块上传比直接上传稍微复杂一点，分块上传需要分为三个阶段：

- 开始上传（initiateMultipartUpload）
- 上传分块（uploadPartFromBlob）
- 上传完成（completeMultipartUpload）

浏览器端代码示例

对文件进行分块

```
  let options = {
  'Content-Length': <file.size>, // 添加http header
  'Content-Type': 'application/json', // 添加http header
  'Cache-Control': 'public, max-age=31536000', // 指定缓存指令
  'Content-Disposition': 'attachment; filename="example.jpg"', // 指示回复的内容该以何种形式展示

  'x-bce-meta-foo1': 'bar1', // 添加自定义meta信息
  'x-bce-meta-foo2': 'bar2', // 添加自定义meta信息
  'x-bce-meta-foo3': 'bar3' // 添加自定义meta信息
};
let PART_SIZE = 5 * 1024 * 1024; // 指定分块大小

function getTasks(file, uploadId, bucketName, key) {
    let leftSize = file.size;
    let offset = 0;
    let partNumber = 1;

    let tasks = [];

    while (leftSize > 0) {
        let partSize = Math.min(leftSize, PART_SIZE);
        tasks.push({
            file: file,
            uploadId: uploadId,
            bucketName: bucketName,
            key: key,
            partNumber: partNumber,
            partSize: partSize,
            start: offset,
            stop: offset + partSize - 1
        });

        leftSize -= partSize;
        offset += partSize;
        partNumber += 1;
    }
    return tasks;
}
```

处理每个分块的上传逻辑

```
  function uploadPartFile(state, client) {
    return function(task, callback) {
        let blob = task.file.slice(task.start, task.stop + 1);
        client.uploadPartFromBlob(task.bucketName, task.key, task.uploadId, task.partNumber, task.partSize, blob)
            .then(function(res) {
                ++state.loaded;
                callback(null, res);
            })
            .catch(function(err) {
                callback(err);
            });
    };
}
```

初始化uploadID，开始上传分块，并完成上传

```
let uploadId;
client.initiateMultipartUpload(bucket, key, options)
    .then(function(response) {
        uploadId = response.body.uploadId; // 开始上传，获取服务器生成的uploadId

        let deferred = sdk.Q.defer();
        let tasks = getTasks(blob, uploadId, bucket, key);
        let state = {
            lengthComputable: true,
            loaded: 0,
            total: tasks.length
        };

        // 为了管理分块上传，使用了async（https://github.com/caolan/async）库来进行异步处理
        let THREADS = 2; // 同时上传的分块数量
        async.mapLimit(tasks, THREADS, uploadPartFile(state, client), function(err, results) {
            if (err) {
                deferred.reject(err);
            } else {
                deferred.resolve(results);
            }
        });
        return deferred.promise;
    })
    .then(function(allResponse) {
        let partList = [];
        allResponse.forEach(function(response, index) {
            // 生成分块清单
            partList.push({
                partNumber: index + 1,
                eTag: response.http_headers.etag
            });
        });
        return client.completeMultipartUpload(bucket, key, uploadId, partList); // 完成上传
    })
    .then(function (res) {
        // 上传完成
    })
    .catch(function (err) {
        // 上传失败，添加您的代码
        console.error(err);
    });
```

Node.js端分块上传

对文件进行分块，并初始化UploadID，上传分块

```
  let options = {
  'Content-Length': <file.size>, // 添加http header
  'Content-Type': 'application/json', // 添加http header
  'Cache-Control': 'public, max-age=31536000', // 指定缓存指令
  'Content-Disposition': 'attachment; filename="example.jpg"', // 指示回复的内容该以何种形式展示

  'x-bce-meta-foo1': 'bar1', // 添加自定义meta信息
  'x-bce-meta-foo2': 'bar2', // 添加自定义meta信息
  'x-bce-meta-foo3': 'bar3' // 添加自定义meta信息
};
let PART_SIZE = 5 * 1024 * 1024;            // 指定分块大小
let uploadId;
client.initiateMultipartUpload(bucket, key, options)
    .then(function(response) {
        uploadId = response.body.uploadId; // 开始上传，获取服务器生成的uploadId
        let deferred = sdk.Q.defer();
        let blob = {
            // 使用fs文件库获取文件大小
            size: fs.statSync(localFileName).size,
            filename: localFileName
        }
        let tasks = getTasks(blob, uploadId, bucket, key);
        let state = {
            lengthComputable: true,
            loaded: 0,
            total: tasks.length
        };
        // 为了管理分块上传，使用了async（https://github.com/caolan/async）库来进行异步处理
        let THREADS = 2; // 同时上传的分块数量
        async.mapLimit(tasks, THREADS, uploadPartFile(state, client), function(err, results) {
            if (err) {
                deferred.reject(err);
            } else {
                deferred.resolve(results);
            }
        });
        return deferred.promise;
    })
    .then(function(allResponse) {
        let partList = [];
        allResponse.forEach(function(response, index) {
            // 生成分块清单
            partList.push({
                partNumber: index + 1,
                eTag: response.http_headers.etag
            });
        });
        return client.completeMultipartUpload(bucket, key, uploadId, partList); // 完成上传
    })
    .then(function (res) {
        // 上传完成
    })
    .catch(function (err) {
        // 上传失败，添加您的代码
        console.error(err);
    });

function getTasks(file, uploadId, bucketName, key) {
    let leftSize = file.size;
    let offset = 0;
    let partNumber = 1;
    let tasks = [];
    while (leftSize > 0) {
        let partSize = Math.min(leftSize, PART_SIZE);
        tasks.push({
            file: file.filename,
            uploadId: uploadId,
            bucketName: bucketName,
            key: key,
            partNumber: partNumber,
            partSize: partSize,
            start: offset,
            stop: offset + partSize - 1
        });

        leftSize -= partSize;
        offset += partSize;
        partNumber += 1;
    }
    return tasks;
}


function uploadPartFile(state, client) {
    return function(task, callback) {
        console.log("task: ", task)
        return client.uploadPartFromFile(task.bucketName, task.key, task.uploadId, task.partNumber, task.partSize, task.file , task.start)
            .then(function(res) {
                ++state.loaded;
                console.log("ok")
                callback(null, res);
            })
            .catch(function(err) {
                console.log("bad")
                callback(err);
            });
    };
}
```

取消分块上传事件

用户可以使用abortMultipartUpload方法取消分块上传。

```
client.abortMultipartUpload(<BucketName>, <Objectkey>, <UploadID>);
```

获取未完成的分块上传事件

用户可以使用listMultipartUploads方法获取Bucket内未完成的分块上传事件。

```
client.listMultipartUploads(<bucketName>)
  .then(function (response) {
      // 遍历所有上传事件
      for (var i = 0; i < response.body.multipartUploads.length; i++) {
          console.log(response.body.multipartUploads[i].uploadId);
      }
  });
```

获取所有已上传的块信息

用户可以使用listParts方法获取某个上传事件中所有已上传的块。

```
client.listParts(<bucketName>, <key>, <uploadId>)
  .then(function (response) {
      // 遍历所有上传事件
      for (var i = 0; i < response.body.parts.length; i++) {
          console.log(response.body.parts[i].partNumber);
      }
  });
```

删除Object

删除Object

- 基本流程
  1. 创建BosClient。
  2. 执行deleteObject()方法。
- 示例代码

```
 // 删除Object
//指定要删除的Object所在Bucket名称和该Object名称
client.deleteObject( < BucketName > , < ObjectKey > );
```

- 完整示例
  - 浏览器端： `let BosClient = baidubce.sdk.BosClient`
  - Node.js： `import {BosClient} from '@baiducloud/sdk'`

```
 let config = {
    endpoint: 'https://bj.bcebos.com'  //需要根据bucket所属区域进行修改
    credentials: {
        ak: <AccessKeyID>,          //您的AK
        sk: <SecretAccessKey>       //您的SK
    }
};

let client = new BosClient(config);

client.deleteObject( < BucketName > , < ObjectKey > );
```

额度管理

额度管理

设置存储桶额度

调用该方法可设置用户的额度，额度包括总的Bucket数及各个Region总的存储容量，当前总的Bucket数最大值不能超过100

- 基本流程

1. 初始化BosClient

2. 调用putUserQuota()方法

- 示例代码

```
 const quotaInfo = {
    maxBucketCount: <maxBucketCount>
    maxCapacityMegaBytes: <maxCapacityMegaBytes>
}
client.putUserQuota(quotaInfo)
```

- 参数 quotaInfo 结构

参数	是否必须	描述
max_bucket_count	是	最大可以创建的bucket数，若为-1，则表示不设置
max_capacity_mega_bytes	是	单位为MB，表示最大的存储容量，若为-1或者0，表示不设置存储容量额度限制，即无上限

> **注意:**
>
> 只有bucket的owner且拥有FULL_CONTROL权限才能够进行此请求

查询额度设置

此接口用来查看用户的额度设置，注意该接口只能被主用户调用。

```
client.getUserQuota()
```

删除额度设置

此接口用来删除额度设置，注意该接口只能被主用户调用。

```
client.deleteUserQuota()
```

版本变更记录

CHANGELOG

1.0.0-rc.42

published on 2023-10-26

- BosClient: support callback parameter in options;

Approach 1:

> Use the callback parameter, the SDK will help you process the parameter and add it to the request header.

```
try {
 const res = await client.putObjectFromString('bucketName', 'fileName', 'demo-string', {
    callback: {
      urls: ["https://www.test.com/callback"],
      vars: {name: 'baidu'},
      encrypt: 'config',
      key: 'callback1'
    }
 });

 /* callback result */
 console.log(res.body.callback.result);
} catch (e) {
 /* callback error code */
 console.error(res.body.callback.code);
 /* callback error message */
 console.error(res.body.callback.message);
}
```

Approach 2:

> Directly pass the "x-bce-process" parameter to headers.

```
try {
 const res = await client.putObjectFromString('bucketName', 'fileName', 'demo-string', {
    'x-bce-process': 'callback/callback,u_WyJodHRwczovL3d3dy50ZXN0LmNvbS9jYWxsYmFjayJd,m_sync,v_eyJuYW1lIjoiYmFpZHUifQ'
 });

 /* callback result */
 console.log(res.body.callback.result);
} catch (e) {
 /* callback error code */
 console.error(res.body.callback.code);
 /* callback error message */
 console.error(res.body.callback.message);
}
```

1.0.0-rc.41

published on 2023-10-26

- BosClient: support 'x-bce-process' in headers;
- BosClient: add createFolderShareUrl method for sharing links;

1.0.0-rc.40

published on 2023-06-19

- BosClient: 'x-bce-security-token' considered in auth token.

1.0.0-rc.39

published on 2023-06-16

- BosClient: Add 'x-bce-security-token' when using generatePresignedUrl with sessionToken;

1.0.0-rc.38

published on 2023-02-16

- BosClient: support symlink;

1.0.0-rc.37

published on 2023-01-09

- BosClient: support 'x-bce-server-side-encryption', 'x-bce-restore-days', 'x-bce-restore-tier' headers;

1.0.0-rc.36

published on 2022-05-06

- BOSClient: fix getObject stream.store

1.0.0-rc.35

published on 2022-05-05

- BOSClient: Just keep an array of all of buffers and concat at the end.

1.0.0-rc.34

published on 2021-12-15

- BOSClient: Provide cname_enabled field when using a custom domain name to delete bucketName path.

1.0.0-rc.33

published on 2021-11-26

- TsdbDataClient: support database parameter.
- BOSClient: fix the browser environment reference entry.

1.0.0-rc.32

published on 2021-10-22

- BOSClient: fix the browser environment reference entry.

1.0.0-rc.31

published on 2021-08-13

- BOSClient: add key valiation in getObject() & getObjectToFile() method;
  - empty key is not allowed
  - consecutive forward slashes (/) are not allowed in key
  - forward slash (/) and a backslash (\) are not allowed at head or tail
  - consecutive periods (..) are not allowed in sub-path

1.0.0-rc.30

published on 2021-08-03

- BOSClient: fix issue of lack of '/' prefix of object url;

1.0.0-rc.29

published on 2021-08-03

- BOSClient: support "config.removeVersionPrefix"(boolean) parameter to dynamic control 'v1' prefix of resource in generatePresignedUrl, generateUrl;

1.0.0-rc.28

published on 2021-03-26

- Fix entry file path for browser environment

1.0.0-rc.27

published on 2021-03-03

- CFCClient supports trigger API

1.0.0-rc.26

published on 2020-12-23

- BOSClient supports putBucketStorageclass API
- BOSClient supports putBucketAcl API

1.0.0-rc.25

published on 2020-09-01

- BOSClient supports `x-bce-storage-class` header

Android-SDK

概述

本文档主要介绍BOS Android SDK的安装和使用。在使用本文档前，您需要先了解BOS的一些基本知识，并已经开通了BOS服务。若您还不了解BOS，可以参考 产品描述和开发者指南。

BOS Android SDK帮助移动开发者实现在Android端简单便捷地使用BOS存储服务，通过该SDK，开发者开发的APP可以直接从终端向BOS发送请求并与BOS进行交互操作。

安装SDK工具包

安装步骤

1. 在官方网站下载SDK压缩包。

2. 解压得到jar包。目前jar包应该包含三个文件，分别是：bos-android-sdk-1.0.X.jar、okhttp-3.x.x.jar 和 okio-1.x.x.jar。

3. 将jar包复制到您的Android工程的lib(libs)目录下。

4. 在Eclipse右键工程->Properties->Java Build Path->Add JARs，导入您刚才复制的jar包。

   在Android Studio右键项目->Open Module Settings->选择项目模块->Dependencies->+，选择File dependency，在lib(libs)目录下选择复制的jar包就可以完成导入。

> 注意：从1.0.3版本开始，BOS android SDK将不再使用Apache HttpClient，而是采用更为轻便高效的OKHttp。具体内容可以参考官网：http://square.github.io/okhttp/

配置权限

请确保您的工程中AndroidManifest.xml文件中已经配置了如下权限，否则，SDK将无法正常运行。

```
<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission android:name="android.permission.READ_PHONE_STATE"/>
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE"/>
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
```

配置混淆协议

```
-dontwarn com.baidubce.**
-keep class com.baidubce.**{*;}
-keep interface com.baidubce.**{*;}
```

SDK目录结构

```
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
    |      └─── bos                    //BOS服务相关类
    |          ├─── model              //BOS内部model，如Request或Response
    |          ├─── BosClient.class    //BOS客户端入口类
    |          └─── BosClientConfiguration.class    //针对BOS特有的HttpClient的配置
    |      └─── sts                    //STS服务相关类
    |          └─── model              //STS的内部model
    ├─── 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

快速入门

1. 初始化一个带有STS验证的BosClient。

   BosClient是与BOS服务交互的客户端，BOS Android SDK的BOS操作都是通过BosClient完成的。

   示例代码：

```
BosClientConfiguration config = new BosClientConfiguration(); //初始化一个带有STS验证的BosClient
config.setCredentials(new DefaultBceSessionCredentials(<AccessKeyID>, <SecretAccessKey>, <Token>); //STS返回的临时AK/SK及Token
config.setEndpoint(<EndPoint>); //传入Bucket所在区域域名
BosClient client = new BosClient(config);
```

2. 新建一个Bucket。

   Bucket是BOS上的命名空间，相当于数据的容器，可以存储若干数据实体（Object）。在您上传数据前，必须先创建一个Bucket。

   示例代码：

```
BosClient client = new BosClient(config);    //创建BOSClient实例
client.createBucket(<BucketName>);    //新建一个Bucket，指定Bucket名称
```

3. 上传Object。

   Object是BOS中最基本的数据单元，您可以把Object简单的理解为文件。对于一个简单的Object的上传，BOS为您提供了四种方式：文件形式上传、数据流形式上传、二进制串上传和字符串上传。

   示例代码：

```
// 获取指定文件
File file = new File(<FilePath>);    //指定文件路径

// 以文件形式上传Object
PutObjectResponse putObjectFromFileResponse = client.putObject(<BucketName>, <ObjectKey>, file);

// 获取数据流
InputStream inputStream = new FileInputStream(<FilePath>);

// 以数据流形式上传Object
PutObjectResponse putObjectResponseFromInputStream = client.putObject(<BucketName>, <ObjectKey>, inputStream);

// 以二进制串上传Object
PutObjectResponse putObjectResponseFromByte = client.putObject(<BucketName>, <ObjectKey>, <byte>);

// 以字符串上传Object
PutObjectResponse putObjectResponseFromString = client.putObject(<BucketName>, <ObjectKey>, <string>);

// 打印ETag
System.out.println(putObjectFromFileResponse.getETag());
```

4. 查看Bucket下的Object列表。

   当您完成一系列上传后，可以参考如下代码来查看Bucket下的全部Object。

   示例代码：

```
 // 获取指定Bucket下的所有Object信息
ListObjectsResponse listing = client.listObjects(<BucketName>);

// 遍历所有Object
for (BosObjectSummary objectSummary : listing.getContents()) {
  System.out.println("ObjectKey: " + objectSummary.getKey());
  }
```

5. 获取指定Object

   用户可以参考如下代码来实现对一个或者多个Object的获取。

   示例代码：

```
 // 获取Object，返回结果为BosObject对象
BosObject object = client.getObject(<BucketName>, <ObjectKey>);

// 获取ObjectMeta
ObjectMetadata meta = object.getObjectMetadata();

// 获取Object的输入流
InputStream objectContent = object.getObjectContent();

// 处理Object
...

// 关闭流
objectContent.close();
```

完整示例

```
 import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;

import android.os.Bundle;

import com.baidubce.BceClientException;
import com.baidubce.BceServiceException;
import com.baidubce.auth.DefaultBceCredentials;
import com.baidubce.development.AppSettings;
import com.baidubce.development.BaseActivity;
import com.baidubce.development.R;
import com.baidubce.services.bos.BosClient;
import com.baidubce.services.bos.BosClientConfiguration;
import com.baidubce.services.bos.model.BosObject;
import com.baidubce.services.bos.model.BosObjectSummary;
import com.baidubce.services.bos.model.CreateBucketResponse;
import com.baidubce.services.bos.model.ListObjectsResponse;
import com.baidubce.services.bos.model.ObjectMetadata;
import com.baidubce.services.bos.model.PutObjectResponse;
import com.baidubce.util.BLog;

public class ExampleActivity extends BaseActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        //打开bossdk运行时log
        BLog.enableLog();

        BosClientConfiguration config = new BosClientConfiguration(); //初始化一个带有STS验证的BosClient
        config.setCredentials(new DefaultBceSessionCredentials(<AccessKeyID>, <SecretAccessKey>, <Token>); //STS返回的临时AK/SK及Token
        config.setEndpoint(<EndPoint>); //Bucket所在区域
        final BosClient client = new BosClient(config);

        new Thread(new Runnable() {

            @Override
            public void run() {

                try {
                    //创建Bucket
                    CreateBucketResponse response = client.createBucket(<BucketName>); //新建一个Bucket并指定Bucket名称
                    System.out.println(response.getLocation());
                    System.out.println(response.getName());

                    //上传Object
                    File file = new File(<Path>);//上传文件的目录
                    PutObjectResponse putObjectFromFileResponse = client.putObject(<BucketName>, <ObjectKey>, file);
                    System.out.println(putObjectFromFileResponse.getETag());

                    //查看Object
                    ListObjectsResponse list = client.listObjects(<BucketName>);
                    for (BosObjectSummary objectSummary : list.getContents()) {
                        System.out.println("ObjectKey: " + objectSummary.getKey());
                    }

                    // 获取Object
                    BosObject object = client.getObject(<BucketName>, <ObjectKey>);
                    // 获取ObjectMeta
                    ObjectMetadata meta = object.getObjectMetadata();
```

```
                // 获取Object的输入流
                InputStream objectContent = object.getObjectContent();
                // 处理Object
                FileOutputStream fos=new FileOutputStream(<Path>);//下载文件的目录/文件名
                byte[] buffer=new byte[2048];
                int count=0;
                while ((count=objectContent.read(buffer))>=0) {
                    fos.write(buffer,0,count);
                }

                // 关闭流
                objectContent.close();
                fos.close();
                System.out.println(meta.getETag());
                System.out.println(meta.getContentLength());

            }catch (BceServiceException e) {
                System.out.println("Error ErrorCode: " + e.getErrorCode());
                System.out.println("Error RequestId: " + e.getRequestId());
                System.out.println("Error StatusCode: " + e.getStatusCode());
                System.out.println("Error Message: " + e.getMessage());
                System.out.println("Error ErrorType: " + e.getErrorType());
            } catch (BceClientException e) {
                System.out.println("Error Message: " + e.getMessage());
            } catch (IOException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }



        }
    }).start();

  }
}
```

新建带有STS验证的BosClient

BosClient是BOS服务的客户端，为开发者与BOS服务进行交互提供了一系列的方法。在使用SDK发起对BOS的请求前，您需要初始化一个BosClient实例，并对它进行一些必要设置。

> **说明：**Android SDK主要用于移动端开发的场景，移动端是一个不受信任的环境，把AccessKeyId和SecretAccessKey直接保存在终端用来加签请求，存在极高的风险。故推荐您使用STS鉴权模式。STS的详细介绍请参考 临时授权访问。

使用STS方式需要您在调用API时首先创建带有STS的BosClient来执行，目前BOS的STS只支持如下方法的API调用：

- DeleteObject
- PutObject
- ListParts
- GetObject
- DoesBucketExist
- CompleteMultipartUpload
- AbortMultipartUpload
- InitiateMultipartUpload
- UploadPart

示例代码：用户可以参考如下代码新建一个带有STS验证的BosClient：

```
BosClientConfiguration config = new BosClientConfiguration(); //初始化一个带有STS验证的BosClient
config.setCredentials(new DefaultBceSessionCredentials(<AccessKeyID>, <SecretAccessKey>, <Token>); //STS返回的临时AK/SK及Token
config.setEndpoint(<EndPoint>); //传入Bucket所在区域域名。EndPoint是指BOS服务在各个区域的域名地址，例如北京域名为bj.bcebos.com
BosClient client = new BosClient(config);
```

> **注意：**EndPoint参数只能用指定的包含区域的域名来进行定义。百度智能云目前开放了多区域支持，请参考 区域选择说明。
>
> 目前支持 “华北-北京”、“华南-广州” 和 “华东-苏州” 三个区域。北京区域：`http://bj.bcebos.com`，广州区域：`http://gz.bcebos.com`，苏州区域：`http://su.bcebos.com`。

配置HTTPS协议访问BOS

BOS支持HTTPS传输协议，您可以通过如下两种方式在BOS Android SDK中使用HTTPS访问BOS服务：

- 在endpoint中指明https:

```
String endpoint = "https://bj.bcebos.com";
String ak = "ak";
String sk = "sk";
BosClientConfiguration config = new BosClientConfigration();
config.setEndpoint(endpoint);
config.setCredentials(new DefaultBceCredentials(ak, sk));
BosClient client = new BosClient(config);
```

- 通过调用setProtocol方法设置https协议:

```
String endpoint = "bj.bcebos.com"; // endpoint中不包含protocol
String ak = "ak";
String sk = "sk";
BosClientConfiguration config = new BosClientConfigration();
config.setEndpoint(endpoint);
config.setCredentials(new DefaultBceCredentials(ak, sk));
config.setProtocol(Protocol.HTTPS); // 如果不指明, 则使用http
BosClient client = new BosClient(config);
```

> **注意：** 如果在endpoint中指明了protocol, 则endpoint中的生效, 另外单独再调用setProtocol()不起作用。

```
String endpoint = "http://bj.bcebos.com";
String ak = "ak";
String sk = "sk";
BosClientConfiguration config = new BosClientConfigration();
config.setEndpoint(endpoint);
config.setCredentials(new DefaultBceCredentials(ak, sk));
config.setProtocol(Protocol.HTTPS); // endpoint中已经指明, 此为无效操作, 对http也是如此
BosClient client = new BosClient(config);
```

配置BosClient

如果您需要配置BosClient的一些细节的参数，可以在构造BosClient的时候传入BosClientConfiguration对象。 BosClientConfiguration是BOS服务的配置类，可以通过BosClientConfiguration对基本网络参数进行设置。

设置网络参数

您可以使用BosClientConfiguration对基本网络参数进行设置。

- 示例代码

```
BosClientConfiguration config = new BosClientConfiguration();

// 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传输数据超时的时间为2000毫秒
config.setSocketTimeout(2000);
```

- 参数说明

  通过BosClientConfiguration能指定的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，缺省值HTTP协议
LocalAddress	本地地址
ConnectionTimeoutInMillis	建立连接的超时时间（单位：毫秒），缺省值为30000
SocketTimeoutInMillis	通过打开的连接传输数据的超时时间（单位：毫秒），缺省值为30000
MaxConnections	允许打开的最大HTTP连接数，缺省值为5
RetryPolicy	连接重试策略
SocketBufferSizeInBytes	Socket缓冲区大小
StreamBufferSize	流文件缓冲区大小
Ipv4Priority	Ipv4优先，缺省值false

Bucket管理

新建Bucket

基本流程

1. 创建BOSClient 类的实例。
2. 执行BOSClient.createBucket()方法，你需要提供Bucket的名字。
3. 对返回的对象可以调用执行getLocation()/getName()获取Bucket所在区域和Bucket名字。

示例代码

```
BosClient client = new BosClient(config);    //创建BOSClient实例
client.createBucket(<BucketName>);    //新建一个Bucket，指定Bucket名称
```

完整示例

```
import android.app.Activity;
import android.os.Bundle;

import com.baidubce.BceClientException;
import com.baidubce.BceServiceException;
import com.baidubce.auth.DefaultBceCredentials;
import com.baidubce.development.R;
import com.baidubce.services.bos.BosClient;
import com.baidubce.services.bos.BosClientConfiguration;
import com.baidubce.services.bos.model.CreateBucketResponse;

public class ExampleActivity extends Activity {

 private static String bucketName = <BucketName>;

 @Override
 protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    new Thread(new Runnable() {
       @Override-9
       public void run() {
          try {
             BosClientConfiguration config = new BosClientConfiguration();
             config.setCredentials(new DefaultBceCredentials(<AccessKeyID>, <SecretAccessKey>));  //您的原始AK/SK
             config.setEndpoint(<EndPoint>);   //Bucket所在区域
             BosClient client = new BosClient(config);

             CreateBucketResponse response = client.createBucket(<BucketName>);   //新建一个Bucket并指定Bucket名称

             System.out.println(response.getLocation());
             System.out.println(response.getName());

          } catch (BceServiceException e) {
             System.out.println("Error ErrorCode: " + e.getErrorCode());
             System.out.println("Error RequestId: " + e.getRequestId());
             System.out.println("Error StatusCode: " + e.getStatusCode());
             System.out.println("Error Message: " + e.getMessage());
             System.out.println("Error ErrorType: " + e.getErrorType());
          } catch (BceClientException e) {
             System.out.println("Error Message: " + e.getMessage());
          }
       }
    }).start();
    }
}
```

> **注意：**由于Bucket的名称在所有区域中是唯一的，所以需要保证BucketName不与其他所有区域上的BucketName相同。

查看Bucket列表

基本流程

1. 创建BOSClient类的实例。
2. 执行BOSClient.listBuckets()方法，会返回ListBucketsResponse类的实例。
3. 对ListBucketsResponse类型实例可以进行getBuckets()/getOwner()/getMetadata()操作。

示例代码

如下代码可以列出用户所有的Bucket：

```
List<BucketSummary> buckets = client.listBuckets().getBuckets();
// 遍历Bucket
for (BucketSummary bucket : buckets) {
  System.out.println(bucket.getName());
 }
```

如下代码可以列出Bucket的Owner：

```
User usr = client.listBuckets().getOwner();
System.out.println(usr.getDisplayName());
System.out.println(usr.getId());
```

如下代码可以列出Bucket的Metadata：

```
BosResponseMetadata metaData = client.listBuckets().getMetadata();
System.out.println(metaData.getBceRequestId());
System.out.println(metaData.getBceContentSha256());
System.out.println(metaData.getContentLength());
System.out.println(metaData.getEtag());
System.out.println(metaData.getExpires());
System.out.println(metaData.getContentMd5());
System.out.println(metaData.getContentDisposition());
System.out.println(metaData.getLastModified());
```

> **注意**：Bucket如果没有设置这些参数，返回时可能为空。

完整示例

```
import java.util.List;
import android.app.Activity;
import android.os.Bundle;
import com.baidubce.BceClientException;
import com.baidubce.BceServiceException;
import com.baidubce.auth.DefaultBceCredentials;
import com.baidubce.development.R;
import com.baidubce.model.User;
import com.baidubce.services.bos.BosClient;
import com.baidubce.services.bos.BosClientConfiguration;
import com.baidubce.services.bos.model.BosResponseMetadata;
import com.baidubce.services.bos.model.BucketSummary;
import com.baidubce.services.bos.model.CreateBucketResponse;
import com.baidubce.services.bos.model.ListBucketsResponse;

public class ExampleActivity extends Activity {

 @Override
 protected void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);
   setContentView(R.layout.activity_main);
   new Thread(new Runnable() {
       @Override
       public void run() {
           try {
               BosClientConfiguration config = new BosClientConfiguration();
               config.setCredentials(new DefaultBceCredentials(<AccessKeyID>, <SecretAccessKey>));
               config.setEndpoint(<EndPoint>);
               BosClient client = new BosClient(config);

               ListBucketsResponse bucketsResponse = client.listBuckets() ;
               // 获得Bucket列表
               List<BucketSummary> buckets = bucketsResponse.getBuckets();
               for (BucketSummary bucket : buckets) {
                   System.out.println(bucket.getName());
               }

               // 获得Bucket的Owner
               User user = bucketsResponse.getOwner();
               System.out.println(user.getDisplayName());
               System.out.println(user.getId());

           } catch (BceServiceException e) {
               System.out.println("Error ErrorCode: " + e.getErrorCode());
               System.out.println("Error RequestId: " + e.getRequestId());
               System.out.println("Error StatusCode: " + e.getStatusCode());
               System.out.println("Error Message: " + e.getMessage());
               System.out.println("Error ErrorType: " + e.getErrorType());
           } catch (BceClientException e) {
               System.out.println("Error Message: " + e.getMessage());
           }
       }
   }).start();
}
}
```

判断Bucket是否存在

基本流程

1. 创建BOSClient类的实例。

2. 执行BOSClient.doesBucketExist()方法。

3. doesBucketExist()方法会返回布尔类型，判断bucket是否存在。

示例代码

```
 // 获取Bucket的存在信息
boolean exists = client.doesBucketExist(<BucketName>); //指定Bucket名称

// 输出结果
if (exists) {
    System.out.println("Bucket exists");
} else {
    System.out.println("Bucket not exists");
}
```

完整示例

```
 import android.app.Activity;
import android.os.Bundle;
import com.baidubce.BceClientException;
import com.baidubce.BceServiceException;
import com.baidubce.auth.DefaultBceCredentials;
import com.baidubce.development.R;
import com.baidubce.services.bos.BosClient;
import com.baidubce.services.bos.BosClientConfiguration;

public class ExampleActivity extends Activity {

private String bucketName = <BucketName>;  //您的Bucket名称

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    new Thread(new Runnable() {
        @Override
        public void run() {
            try {
                BosClientConfiguration config = new BosClientConfiguration();
                config.setCredentials(new DefaultBceCredentials(<AccessKeyID>, <SecretAccessKey>));
                config.setEndpoint(<EndPoint>);
                BosClient client = new BosClient(config);

                // 获取Bucket的存在信息
                boolean exists = client.doesBucketExist(<BucketName>); //指定Bucket名称
                // 输出结果
                if (exists) {
                    System.out.println("Bucket exists");
                } else {
                    System.out.println("Bucket not exists");
                }

            } catch (BceServiceException e) {
                System.out.println("Error ErrorCode: " + e.getErrorCode());
                System.out.println("Error RequestId: " + e.getRequestId());
                System.out.println("Error StatusCode: " + e.getStatusCode());
                System.out.println("Error Message: " + e.getMessage());
                System.out.println("Error ErrorType: " + e.getErrorType());
            } catch (BceClientException e) {
                System.out.println("Error Message: " + e.getMessage());
            }
        }
    }).start();

}
}
```

删除Bucket

基本流程

1. 创建BOSClient类的实例。
2. 执行BOSClient.deleteBucket()方法。
3. deleteBucket()无返回值，删除失败时会抛出异常。

示例代码

```
client.deleteBucket(<BucketName>);    //指定Bucket名称
```

> **注意：**如果Bucket不为空（即Bucket中有Object和未完成的三步上传Part存在），则Bucket无法被删除，必须清空Bucket后才能成功删除。

完整示例

```
import android.app.Activity;
import android.os.Bundle;
import com.baidubce.BceClientException;
import com.baidubce.BceServiceException;
import com.baidubce.auth.DefaultBceCredentials;
import com.baidubce.development.R;
import com.baidubce.services.bos.BosClient;
import com.baidubce.services.bos.BosClientConfiguration;

public class ExampleActivity extends Activity {

 private String bucketName = <BucketName>;

 @Override
 protected void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.activity_main);
  new Thread(new Runnable() {
      @Override
      public void run() {
          try {
              BosClientConfiguration config = new BosClientConfiguration();
              config.setCredentials(new DefaultBceCredentials(<AccessKeyID>, <SecretAccessKey>));
              config.setEndpoint(<EndPoint>);
              BosClient client = new BosClient(config);

              // 删除Bucket
              client.deleteBucket(<BucketName>); //指定Bucket名称

          } catch (BceServiceException e) {
              System.out.println("Error ErrorCode: " + e.getErrorCode());
              System.out.println("Error RequestId: " + e.getRequestId());
              System.out.println("Error StatusCode: " + e.getStatusCode());
              System.out.println("Error Message: " + e.getMessage());
              System.out.println("Error ErrorType: " + e.getErrorType());
          } catch (BceClientException e) {
              System.out.println("Error Message: " + e.getMessage());
          }
      }
  }).start();
}
}
```

Bucket权限控制

设置Bucket的访问权限

基本流程

1. 创建BOSClient类的实例。
2. 执行BOSClient.setBucketAcl()方法。
3. setBucketAcl()无返回值，设置失败时会抛出异常。

示例代码

```
// 设置Bucket访问权限
client.setBucketAcl(<BucketName>, CannedAccessControlList.Private);
```

> **说明：**CannedAccessControlList是枚举类型，包含三个值： `Private` 、 `PublicRead` 、 `PublicReadWrite` ，它们分别对应相关权限。具体内容可以参考《BOS API文档 使用CannedAcl方式的权限控制》。

完整示例

```
import org.json.JSONException;
import android.app.Activity;
import android.os.Bundle;
import com.baidubce.BceClientException;
import com.baidubce.BceServiceException;
import com.baidubce.auth.DefaultBceCredentials;
import com.baidubce.development.R;
import com.baidubce.services.bos.BosClient;
import com.baidubce.services.bos.BosClientConfiguration;
import com.baidubce.services.bos.model.CannedAccessControlList;

public class ExampleActivity extends Activity {

 private String bucketName = <BucketName>;

 @Override
 protected void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.activity_main);
  new Thread(new Runnable() {
     @Override
     public void run() {
        try {
           BosClientConfiguration config = new BosClientConfiguration();
           config.setCredentials(new DefaultBceCredentials(<AccessKeyID>, <SecretAccessKey>));
           config.setEndpoint(<EndPoint>);
           BosClient client = new BosClient(config);

           // 设置Bucket访问权限
           client.setBucketAcl(<BucketName>, CannedAccessControlList.Private);   //指定Bucket名称，并设置Bucket权限为Private

        } catch (BceServiceException e) {
           System.out.println("Error ErrorCode: " + e.getErrorCode());
           System.out.println("Error RequestId: " + e.getRequestId());
           System.out.println("Error StatusCode: " + e.getStatusCode());
           System.out.println("Error Message: " + e.getMessage());
           System.out.println("Error ErrorType: " + e.getErrorType());
        } catch (BceClientException e) {
           System.out.println("Error Message: " + e.getMessage());
        } catch (JSONException e) {
           // TODO Auto-generated catch block
           e.printStackTrace();
        }
     }
  }).start();

}
}
```

设置指定用户对Bucket的访问权限

基本流程

1. 创建BOSClient类的实例。

2. 执行BOSClient.setBucketAcl()方法，您需要创建一个SetBucketAclRequest的实例来提供上述信息。

3. setBucketAcl()无返回值，设置失败时会抛出异常。

示例代码

```
List<Grant> grants = new ArrayList<Grant>();
List<Grantee> grantee = new ArrayList<Grantee>();
List<Permission> permission = new ArrayList<Permission>();

//授权给用户
grantee.add(new Grantee(<UserID>);

//设置权限
permission.add(Permission.READ);
grants.add(new Grant().withGrantee(grantee).withPermission(permission));

//封装成request
SetBucketAclRequest request = new SetBucketAclRequest(<BucketName>, grants);

//设置bucket的访问权限
client.setBucketAcl(request);
```

> **注意**：Permission中的权限设置包含三个值：`READ`、`WRITE`、`FULL_CONTROL`，它们分别对应相关权限。具体内容可以参考《BOS API文档 上传ACL文件方式的权限控制》。

完整示例

```
import java.util.ArrayList;
import java.util.List;
import org.json.JSONException;
import android.app.Activity;
import android.os.Bundle;
import com.baidubce.BceClientException;
import com.baidubce.BceServiceException;
import com.baidubce.auth.DefaultBceCredentials;
import com.baidubce.development.R;
import com.baidubce.services.bos.BosClient;
import com.baidubce.services.bos.BosClientConfiguration;
import com.baidubce.services.bos.model.Grant;
import com.baidubce.services.bos.model.Grantee;
import com.baidubce.services.bos.model.Permission;
import com.baidubce.services.bos.model.SetBucketAclRequest;

public class ExampleActivity extends Activity {

 private String bucketName = <BucektName>;
 List<Grant> grants = new ArrayList<Grant>();
 List<Grantee> grantee = new ArrayList<Grantee>();
 List<Permission> permission = new ArrayList<Permission>();

 @Override
 protected void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.activity_main);
  new Thread(new Runnable() {
     @Override
     public void run() {
        try {
           BosClientConfiguration config = new BosClientConfiguration();
           config.setCredentials(new DefaultBceCredentials(<AccessKeyID>, <SecretAccessKey>));
           config.setEndpoint(<EndPoint>);
           BosClient client = new BosClient(config);

           //授权给特定用户
           grantee.add(new Grantee("UserId_1"));
           grantee.add(new Grantee("UserId_2"));

           //授权给Everyone
           grantee.add(new Grantee("*"));

           //设置权限
           permission.add(Permission.READ);
           permission.add(Permission.WRITE);

           grants.add(new Grant().withGrantee(grantee).withPermission(permission));

           //封装request
           SetBucketAclRequest request = new SetBucketAclRequest(<BucketName>, grants);

           //设置bucket的访问权限
           client.setBucketAcl(request);

        } catch (BceServiceException e) {
           System.out.println("Error ErrorCode: " + e.getErrorCode());
           System.out.println("Error RequestId: " + e.getRequestId());
           System.out.println("Error StatusCode: " + e.getStatusCode());
           System.out.println("Error Message: " + e.getMessage());
           System.out.println("Error ErrorType: " + e.getErrorType());
        } catch (BceClientException e) {
           System.out.println("Error Message: " + e.getMessage());
        } catch (JSONException e) {
           // TODO Auto-generated catch block
           e.printStackTrace();
        }
     }
  }).start();

}
}
```

文件管理

上传Object

最简单的上传

基本流程

1. 创建BOSClient 类的实例。

2. 调用BOSClient.putObject()方法，可以通过如下四种方式上传Object：文件、数据流、二进制串和字符串的形式。

3. 对返回的PutObjectResponse类型实例，执行getETag()获得上传成功的ETag。

示例代码

```
 // 获取指定文件
File file = new File(<FilePath>);     //指定文件路径

// 以文件形式上传Object
PutObjectResponse putObjectFromFileResponse =
    client.putObject(<BucketName>, <ObjectKey>, file);

// 获取数据流
InputStream inputStream = new FileInputStream(<FilePath>);

// 以数据流形式上传Object
PutObjectResponse putObjectResponseFromInputStream =
    client.putObject(<BucketName>, <ObjectKey>, inputStream);

// 以二进制串上传Object
PutObjectResponse putObjectResponseFromByte =
    client.putObject(<BucketName>, <ObjectKey>, <byte>);

// 以字符串上传Object
PutObjectResponse putObjectResponseFromString =
    client.putObject(<BucketName>, <ObjectKey>, <string>);

// 打印ETag
System.out.println(putObjectFromFileResponse.getETag());
```

> **说明**：Object以文件的形式上传到BOS中，putObject函数支持不超过5GB的Object上传。在putObject请求处理成功后，BOS会在Header中返回Object的ETag作为文件标识。

完整示例

```
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.InputStream;
import android.app.Activity;
import android.os.Bundle;
import com.baidubce.BceClientException;
import com.baidubce.BceServiceException;
import com.baidubce.auth.DefaultBceCredentials;
import com.baidubce.development.R;
import com.baidubce.services.bos.BosClient;
import com.baidubce.services.bos.BosClientConfiguration;
import com.baidubce.services.bos.model.PutObjectResponse;

public class ExampleActivity extends Activity {

 private String bucketName = <BucketName>;
 private String objectKey = <ObjectKey>;
 byte[] b = null;
 String str = <PutString>;

 @Override
 protected void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.activity_main);
  new Thread(new Runnable() {
     @Override
     public void run() {
        try {
           BosClientConfiguration config = new BosClientConfiguration();
           config.setCredentials(new DefaultBceCredentials(<AccessKeyID>, <SecretAccessKey>));
           config.setEndpoint(<EndPoint>);
           BosClient client = new BosClient(config);

           // 获取指定文件
           File file = new File("/path/to/file.zip");

           // 以文件形式上传Object
           PutObjectResponse putObjectFromFileResponse = client.putObject(<BucketName>, <ObjectKey>, file);

           // 获取数据流
           InputStream inputStream = new FileInputStream("/path/to/test.zip");

           // 以数据流形式上传Object
           PutObjectResponse putObjectResponseFromInputStream = client.putObject(<BucketName>, <ObjectKey>, inputStream);

           // 以二进制串上传Object
           PutObjectResponse putObjectResponseFromByte = client.putObject(<BucketName>, <ObjectKey>, b);

           // 以字符串上传Object
           PutObjectResponse putObjectResponseFromString = client.putObject(<BucketName>, <ObjectKey>, str);

           // 打印ETag
           System.out.println(putObjectFromFileResponse.getETag());

        } catch (BceServiceException e) {
           System.out.println("Error ErrorCode: " + e.getErrorCode());
           System.out.println("Error RequestId: " + e.getRequestId());
           System.out.println("Error StatusCode: " + e.getStatusCode());
           System.out.println("Error Message: " + e.getMessage());
           System.out.println("Error ErrorType: " + e.getErrorType());
        } catch (BceClientException e) {
           System.out.println("Error Message: " + e.getMessage());
        } catch (FileNotFoundException e) {
           // TODO Auto-generated catch block
           e.printStackTrace();
        }
     }
  }).start();
}}
```

设定Object的Copy属性

CopyObject接口用于将一个已经存在的Object拷贝到另外一个Object，拷贝过程中会对源Object的Etag或修改状态进行判断，根据判断结果决定是否执行拷贝。详细的参数解释如下：

名称	类型	描述	是否必需
x-bce-copy-source-if-match	String	如果源Object的ETag值和用户提供的ETag相等，则执行拷贝操作，否则拷贝失败。	否
x-bce-copy-source-if-none-match	String	如果源Object的ETag和用户提供的ETag不相等，则执行拷贝操作，否则拷贝失败。	否
x-bce-copy-source-if-unmodified-since	String	如果源object在x-bce-copy-source-if-unmodified-since之后没被修改，则执行拷贝操作，否则拷贝失败。	否
x-bce-copy-source-if-modified-since	String	如果源object在x-bce-copy-source-if-modified-since之后被修改了，则执行拷贝操作，否则拷贝失败。	否

示例代码

```
 // 初始化BosClient
BosClient client = ...;

// 创建CopyObjectRequest对象
CopyObjectRequest copyObjectRequest = new CopyObjectRequest(srcBucketName, srcKey, destBucketName, destKey);

// 设置新的Metadata
Map<String, String> userMetadata = new HashMap<String, String>();
userMetadata.put("<user-meta-key>","<user-meta-value>");
ObjectMetadata meta = new ObjectMetadata();
meta.setUserMetadata(userMetadata);
copyObjectRequest.setNewObjectMetadata(meta);

//copy-source-if-match
copyObjectRequest.withETag("111111111183bf192b57a4afc76fa632");
//copy-source-if-none-match
copyObjectRequest.withNoMatchingETagConstraint("111111111183bf192b57a4afc76fa632");

Date modifiedSinceConstraint = new Date();
SimpleDateFormat df = new SimpleDateFormat("E, dd MMM yyyy HH:mm:ss z", Locale.UK);
df.setTimeZone(new java.util.SimpleTimeZone(0, "GMT"));
String date = df.format(modifiedSinceConstraint);

//copy-source-if-modified-since
copyObjectRequest.withModifiedSinceConstraint(date);

//copy-source-if-unmodified-since
copyObjectRequest.withUnmodifiedSinceConstraint(date);

// 复制Object
CopyObjectResponse copyObjectResponse = client.copyObject(copyObjectRequest);
```

设定Object的Http Header

BOS支持您在上传object时设定Http Header。

基本流程

1. 创建ObjectMetadata类的实例。

2. 执行setContentLength()/setContentType()等方法对Http Header进行设定。

3. 将设定后的meta作为参数加入client.putObject()中。

示例代码

```
 // 创建ObjectMetadata类的实例
ObjectMetadata meta = new ObjectMetadata();

// 设置ContentLength大小
meta.setContentLength(<Length>);

// 设置ContentType
meta.setContentType("application/json");

client.putObject(<BucketName>, <ObjectKey>, content, meta);
```

> **说明：**header可设置的属性有："Cache-Control"、"Content-Encoding"、"Content-Disposition"、"Expires"

完整示例

```
 import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.InputStream;
import android.app.Activity;
import android.os.Bundle;
import com.baidubce.BceClientException;
import com.baidubce.BceServiceException;
import com.baidubce.auth.DefaultBceCredentials;
import com.baidubce.demo.R;
import com.baidubce.services.bos.BosClient;
import com.baidubce.services.bos.BosClientConfiguration;
import com.baidubce.services.bos.model.ObjectMetadata;
import com.baidubce.services.bos.model.PutObjectResponse;

public class ExampleActivity extends Activity {

private String bucketName = <BucketName>;
private String objectKey = <ObjectKey>;

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    new Thread(new Runnable() {
        @Override
        public void run() {
            try {
                BosClientConfiguration config = new BosClientConfiguration();
                config.setCredentials(new DefaultBceCredentials(<AccessKeyID>, <SecretAccessKey>));
                config.setEndpoint(<EndPoint>);
                BosClient client = new BosClient(config);

                // 获取数据流
                InputStream inputStream = new FileInputStream("/path/to/test.zip");

                // 创建ObjectMetadata类的实例
                ObjectMetadata meta = new ObjectMetadata();

                // 设置ContentLength大小
                meta.setContentLength(1000);

                // 设置ContentType
                meta.setContentType("application/json");

                PutObjectResponse putObjectResponseFromInputStream = client.putObject(bucketName, objectKey, inputStream, meta);

                 // 打印ETag
                System.out.println(putObjectResponseFromInputStream.getETag());

            } catch (BceServiceException e) {
                System.out.println("Error ErrorCode: " + e.getErrorCode());
                System.out.println("Error RequestId: " + e.getRequestId());
                System.out.println("Error StatusCode: " + e.getStatusCode());
                System.out.println("Error Message: " + e.getMessage());
                System.out.println("Error ErrorType: " + e.getErrorType());
            } catch (BceClientException e) {
                System.out.println("Error Message: " + e.getMessage());
            } catch (FileNotFoundException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
        }
    }).start();
}}
```

用户自定义元数据

BOS支持用户自定义元数据来对Object进行描述。

基本流程

1. 创建ObjectMetadata类的实例。

2. 执行addUserMetadata()方法对Http Header进行自定义元数据。

3. 将设定后的meta作为参数加入client.putObject()中。

示例代码

```
 // 设置自定义元数据name的值为my-data
meta.addUserMetadata("name", "my-data");

// 上传Object
client.putObject(<BucketName>, <ObjectKey>, <Content>, meta);
```

> **说明：**在上面代码中，用户自定义了一个名字为"name"，值为"my-data"的元数据。当用户下载此Object的时候，此元数据也可以一并得到。一个Object可以有多个类似的参数，但所有的User Meta总大小不能超过2KB。

完整示例

```
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.InputStream;
import android.app.Activity;
import android.os.Bundle;
import com.baidubce.BceClientException;
import com.baidubce.BceServiceException;
import com.baidubce.auth.DefaultBceCredentials;
import com.baidubce.demo.R;
import com.baidubce.services.bos.BosClient;
import com.baidubce.services.bos.BosClientConfiguration;
import com.baidubce.services.bos.model.ObjectMetadata;
import com.baidubce.services.bos.model.PutObjectResponse;

public class ExampleActivity extends Activity {

private String bucketName = <BucketName>;
private String objectKey = <ObjectKey>;

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    new Thread(new Runnable() {
        @Override
        public void run() {
            try {
                BosClientConfiguration config = new BosClientConfiguration();
                config.setCredentials(new DefaultBceCredentials(<AccessKeyID>, <SecretAccessKey>));
                config.setEndpoint(<EndPoint>);
                BosClient client = new BosClient(config);

                // 获取数据流
                InputStream inputStream = new FileInputStream("/path/to/test.zip");

                // 创建ObjectMetadata类的实例
                ObjectMetadata meta = new ObjectMetadata();

                // 自定义元数据
                meta.addUserMetadata("name", "my-data");

                PutObjectResponse putObjectResponseFromInputStream = client.putObject(<BucketName>, <ObjectKey>, inputStream, meta);

                 // 打印ETag
                System.out.println(putObjectResponseFromInputStream.getETag());

            } catch (BceServiceException e) {
                System.out.println("Error ErrorCode: " + e.getErrorCode());
                System.out.println("Error RequestId: " + e.getRequestId());
                System.out.println("Error StatusCode: " + e.getStatusCode());
                System.out.println("Error Message: " + e.getMessage());
                System.out.println("Error ErrorType: " + e.getErrorType());
            } catch (BceClientException e) {
                System.out.println("Error Message: " + e.getMessage());
            } catch (FileNotFoundException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
        }
    }).start();
}}
```

Append方式上传Object

BOS支持AppendObject，即以追加写的方式上传文件，适用场景如日志追加及直播等实时视频文件上传。通过AppendObject操作创建的Object类型为Appendable Object，可以对该Object追加数据；而通过PutObject上传的Object是Normal Object，不可进行数据追加写。AppendObject大小限制为0~5G。

示例代码

```
public void AppendObject(BosClient client, String bucketName, String objectKey, byte[] byte1, String string1) {
    // 获取指定文件
    File file = new File("/path/to/file.zip");
    // 获取数据流
    InputStream inputStream = new FileInputStream("/path/to/test.zip");

    // 以文件形式上传Object
    AppendObjectResponse appendObjectFromFileResponse = client.appendObject(bucketName, objectKey, file);
    // 以数据流形式上传Object
    AppendObjectResponse appendObjectResponseFromInputStream = client.appendObject(bucketName, objectKey, inputStream);
    // 以二进制串上传Object
    AppendObjectResponse appendObjectResponseFromByte = client.appendObject(bucketName, objectKey, byte1);
    // 以字符串上传Object
    AppendObjectResponse appendObjectResponseFromString = client.appendObject(bucketName, objectKey, string1);

    // 向AppendFile追加内容
    Long nextOffset = appendObjectFromFileResponse.getNextAppendOffset();
    AppendObjectRequest request =new AppendObjectRequest(bucketName, objectKey,
    RestartableInputStream.wrap(string1.getBytes()));
    request.withOffset(nextOffset);
    AppendObjectResponse appendResponse = client.appendObject(request);
}
```

获取上传进度

Android SDK支持在上传过程中实时提供上传进度信息。目前支持PutObject, AppendObject, UploadPart以及PutSuperObjectFromFile四个接口。进度上传接口的使用必须构造对应的Request（PutObjectRequest，AppendObjectRequest，UploadPartRequest以及PutSuperObjectRequest）。

SDK提供的上传进度回调接口如下，您可以在其中定义上传过程中您所需要的操作，如更新界面等等。

```
public interface BceProgressCallback<T extends AbstractBceRequest> {
    // request为上传的请求
    // currentSize为当前上传的大小(单位:byte)
    // totalSize为本次请求需要上传的总大小(单位:byte)
    void onProgress(T request, long currentSize, long totalSize);
}
```

PutObject示例代码

```
PutObjectRequest request = new PutObjectRequest(this.bucketName, "test", file);
ObjectMetadata objectMetadata = new ObjectMetadata();
objectMetadata.setContentType("text/plain");
request.setObjectMetadata(objectMetadata);
request.setProgressCallback(new BosProgressCallback<PutObjectRequest>() {
    @Override
    public void onProgress(PutObjectRequest request, long currentSize, long totalSize) {
        Log.e(currentSize + "", totalSize + "");
    }
});
String eTag = this.client.putObject(request).getETag();
```

AppendObject示例代码

```
ObjectMetadata objectMetadata = new ObjectMetadata();
objectMetadata.setContentType("text/plain");
AppendObjectRequest request = new AppendObjectRequest(this.bucketName, "test", file);
request.setObjectMetadata(objectMetadata);
AppendObjectResponse response = this.client.appendObject(request);
Long nextOffset = response.getNextAppendOffset();
request.withOffset(nextOffset);
request.setProgressCallback(new BosProgressCallback<AppendObjectRequest>() {
    @Override
    public void onProgress(AppendObjectRequest request, long currentSize, long totalSize) {
        Log.e(currentSize + "", totalSize + "");
    }
});
response = this.client.appendObject(request);
```

UploadPart示例代码

```
UploadPartRequest request = new UploadPartRequest().withBucketName(this.bucketName)
        .withKey("test").withUploadId(uploadId).withPartNumber(1).withPartSize(8000)
        .withInputStream(fis);
request.setProgressCallback(new BosProgressCallback<UploadPartRequest>() {
    @Override
    public void onProgress(UploadPartRequest request, long currentSize, long totalSize) {
        Log.e(currentSize + "", totalSize + "");
    }
});
UploadPartResponse response = this.client.uploadPart(request);
```

PutSuperObjectFromFile示例代码

```
PutSuperObjectRequest request = new PutSuperObjectRequest().withBucketName(this.bucketName)
        .withKey("test").withPartSize(1024*1024*2L).withFile(file);
request.setProgressCallback(new BosProgressCallback<PutSuperObjectRequest>() {
    @Override
    public void onProgress(PutSuperObjectRequest request, long currentSize, long totalSize) {
        Log.e(currentSize + "", totalSize + "");
    }
});
PutSuperObjectResponse response = this.client.putSuperObjectFromFile(request);
```

目前，SDK默认上传进度的回调粒度为2048个字节。您可以通过如下方法设置一个更适合您App应用的回调粒度：

BosClientConfiguration config=new BosClientConfiguration(); config.setUploadSegmentPart(1024);

> **注意：** 该值必须在1-8192之间，如不在该范围内，Android SDK会强制设置为2048.

取消接口

Android SDK支持在请求过程中取消本次请求，其中上传请求中剩余的数据将不再会上传至BOS，下载请求中取消操作会关闭连接，执行中的操作会抛出`Request is canceled!`异常

> 所有的BOS操作都可以使用取消接口，但是已经完成的请求无法被取消，已经完成的操作也不会抛出异常

```
 // 以上传接口取消为例
final PutObjectRequest req = new PutObjectRequest(<BucketName>, <ObjectName>,
        new FileInputStream(<FilePath>));
Runnable cancelTask = new Runnable() {
    @Override
    public void run() {
        try {
            Thread.sleep(100);
        } catch (InterruptedException e) {
            e.printStackTrace()
        }
        // 取消本次上传请求
        req.cancel();
    }
};
new Thread(cancelTask).start();
client.putObject(req);
```

同步回调 Android SDK支持BOS服务端同步回调接口，通过在PutObjectRequest中，设置process参数，就可以在上传完成后，让BOS服务端主动调用回调接口来达到通知客户目的。

```
 PutObjectRequest request = new PutObjectRequest({bucket}, {object}, {inputStream});
// 设置x-bce-process参数
request.setProcess({x-bce-process});
PutObjectResponse response = client.putObject(request);
// 获取返回的http status code
int statusCode = response.getHttpResponse().getStatusCode();
// 获取返回的回调接口返回的数据
string callbackString = response.getServerCallbackReturnBody();
```

单链接限速 对象存储BOS的单Bucket带宽是有限制的，当用户的上传或下载占用带宽达到带宽限制阈值时，会返回RequestRateLimitExceeded的错误码。

为保证用户能够正常使用服务，BOS支持在进行上传、下载等行为时进行流量控制，保证大流量服务占用带宽不会对其他应用服务造成影响。

读写接口支持设置指定限速值，限速值的取值范围为819200~838860800，单位为bit/s，即100KB/s~100MB/s。限速值取值必须为数字，BOS将按照指定的限速值对此次请求进行限速，当限速值不在此范围或不合法时将返回400错误码。

当前PutObjectRequest、PutSuperObjectRequest、UploadPartRequest、CopyObjectRequest、GetObjectRequest均支持设置单链接限速。 以PutObjectRequest为例：

```
 PutObjectRequest request = new PutObjectRequest({bucket}, {object}, {inputStream});
// 设置单链接限速
request.setTrafficLimitBitPS(819200);
```

查看Bucket中的Object

简单查询

查看Bucket中Object列表。

基本流程

1. 创建BOSClient类的实例。
2. 执行BOSClient.listObjects(bucketName)方法，会返回ListObjectsResponse类的实例。
3. 对ListObjectsResponse类型可以进行getBuckets()/getOwner()/getMetadata()操作。

示例代码

```
 // 获取指定Bucket下的所有Object信息
ListObjectsResponse listing = client.listObjects(<BucketName>);

// 遍历所有Object
for (BosObjectSummary objectSummary : listing.getContents()) {
    System.out.println("ObjectKey: " + objectSummary.getKey());
}
```

> **说明：** listObjects( )方法返回ListObjectsResponse对象，ListObjectsResponse对象包含了此次listObject请求的返回结果。用户可以通过ListObjectsResponse中的getContents方法获取所有Object的描述信息。
>
> - 默认情况下，如果Bucket中的Object数量大于1000，则只会返回1000个Object，并且返回结果中IsTruncated值为True，并返回NextMarker做为下次读取的起点。
> - 若想获取更多的Object，可以使用Marker参数分次读取，请参考扩展查询。

完整示例

```
import android.app.Activity;
import android.os.Bundle;
import com.baidubce.BceClientException;
import com.baidubce.BceServiceException;
import com.baidubce.auth.DefaultBceCredentials;
import com.baidubce.demo.R;
import com.baidubce.services.bos.BosClient;
import com.baidubce.services.bos.BosClientConfiguration;
import com.baidubce.services.bos.model.BosObjectSummary;
import com.baidubce.services.bos.model.ListObjectsResponse;

public class ExampleActivity extends Activity {

private String bucketName = <BucketName>;

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    new Thread(new Runnable() {
        @Override
        public void run() {
            try {
                BosClientConfiguration config = new BosClientConfiguration();
                config.setCredentials(new DefaultBceCredentials(<AccessKeyId>, <SecretAccessKey>));
                config.setEndpoint(<EndPoint>);
                BosClient client = new BosClient(config);

                // 获取指定Bucket下的所有Object信息
                ListObjectsResponse listing = client.listObjects(<BucketName>);

                // 遍历所有Object
                for (BosObjectSummary objectSummary : listing.getContents()) {
                    System.out.println("ObjectKey: " + objectSummary.getKey());
                }

            } catch (BceServiceException e) {
                System.out.println("Error ErrorCode: " + e.getErrorCode());
                System.out.println("Error RequestId: " + e.getRequestId());
                System.out.println("Error StatusCode: " + e.getStatusCode());
                System.out.println("Error Message: " + e.getMessage());
                System.out.println("Error ErrorType: " + e.getErrorType());
            } catch (BceClientException e) {
                System.out.println("Error Message: " + e.getMessage());
            }
        }
    }).start();
}}
```

扩展查询

用户可以通过设置ListObjectsRequest参数来完成更多扩展查询操作设置。ListObjectsRequest中可以设置的扩展参数如下：

参数名称	说明	默认值
MaxKeys	设定此次返回Object的最大个数，不可超过1000。	1000
Prefix	设定objectKey的前缀，前缀是指objectKey包含并以Prefix的值作为开始。 通常与Delimiter配合在查询模拟文件夹中使用。	-
Delimiter	是一个分隔符，用来对objectKey进行分层。 通常与Prefix配合在查询模拟文件夹中使用。 从Prefix开始到第一次出现Delimiter字符之间的objectKey称为：CommonPrefixes。	-
Marker	是一个字符串，用来设定返回结果的起始位置。 设定Marker值之后，返回的Object会从Marker值之后按字母排序开始返回。	-

基本流程

1. 创建ListObjectsRequest类的实例。
2. 在ListObjectsRequest中执行setDelimiter/setMarker()/setPrefix()等方法，实现更多的扩展查询操作。
3. 创建BOSClient类的实例，执行listObjects(listObjectsRequest)。

示例代码

```
// 构造ListObjectsRequest请求
ListObjectsRequest listObjectsRequest = new ListObjectsRequest(<BucketName>);

// 设置扩展查询参数
listObjectsRequest.setDelimiter(<Delimiter>);
listObjectsRequest.setMarker(<Marker>);
...
ListObjectsResponse listing = client.listObjects(listObjectsRequest);
```

> **说明：**上面代码中调用了 `listObjects` 中的一个重载方法，通过传入 `ListObjectsRequest` 来完成请求。

完整示例

示例一：

```
 import android.app.Activity;
import android.os.Bundle;
import com.baidubce.BceClientException;
import com.baidubce.BceServiceException;
import com.baidubce.auth.DefaultBceCredentials;
import com.baidubce.demo.R;
import com.baidubce.services.bos.BosClient;
import com.baidubce.services.bos.BosClientConfiguration;
import com.baidubce.services.bos.model.BosObjectSummary;
import com.baidubce.services.bos.model.ListObjectsRequest;
import com.baidubce.services.bos.model.ListObjectsResponse;

public class ExampleActivity extends Activity {

private String bucketName = <BucketName>;

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    new Thread(new Runnable() {
        @Override
        public void run() {
            try {
                BosClientConfiguration config = new BosClientConfiguration();
                config.setCredentials(new DefaultBceCredentials(<AccessKeyID>, <SecretAccessKey>));
                config.setEndpoint(<EndPoint>);
                BosClient client = new BosClient(config);

                // 构造ListObjectsRequest请求
                ListObjectsRequest listObjectsRequest = new ListObjectsRequest(<BucketName>);

                // 设置参数
                listObjectsRequest.setDelimiter("/");
                listObjectsRequest.setMarker("123");
                listObjectsRequest.setMaxKeys(100);
                listObjectsRequest.setPrefix("fun");

                // 获取指定Bucket下符合上述条件的所有Object信息
                ListObjectsResponse listing = client.listObjects(listObjectsRequest);

                // 遍历所有Object
                for (BosObjectSummary objectSummary : listing.getContents()) {
                    System.out.println("ObjectKey: " + objectSummary.getKey());
                }

            } catch (BceServiceException e) {
                System.out.println("Error ErrorCode: " + e.getErrorCode());
                System.out.println("Error RequestId: " + e.getRequestId());
                System.out.println("Error StatusCode: " + e.getStatusCode());
                System.out.println("Error Message: " + e.getMessage());
                System.out.println("Error ErrorType: " + e.getErrorType());
            } catch (BceClientException e) {
                System.out.println("Error Message: " + e.getMessage());
            }
        }
    }).start();
}}
```

示例二：使用Nextmarker完整示例。

```
import android.app.Activity;
import android.os.Bundle;
import com.baidubce.BceClientException;
import com.baidubce.BceServiceException;
import com.baidubce.auth.DefaultBceCredentials;
import com.baidubce.demo.R;
import com.baidubce.services.bos.BosClient;
import com.baidubce.services.bos.BosClientConfiguration;
import com.baidubce.services.bos.model.BosObjectSummary;
import com.baidubce.services.bos.model.ListObjectsRequest;
import com.baidubce.services.bos.model.ListObjectsResponse;

public class ExampleActivity extends Activity {

private String bucketName = <BucketName>;

@Override
protected void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.activity_main);
  new Thread(new Runnable() {
      @Override
      public void run() {
        try {
          BosClientConfiguration config = new BosClientConfiguration();
          config.setCredentials(new DefaultBceCredentials(<AccessKeyId>, <SecretAccessKey>));
          config.setEndpoint(<EndPoint>);
          BosClient client = new BosClient(config);
          ListObjectsRequest listObjectsRequest = new ListObjectsRequest(<BucketName>);
          listObjectsRequest.setMarker("");
          listObjectsRequest.setMaxKeys(20);
          listObjectsRequest.setDelimiter(BceConfig.BOS_DELIMITER);
          ListObjectsResponse listObjectsResponse = client.listObjects(listObjectsRequest);
          String nextMark = "";
          if(listObjectsResponse.isTruncated()){
            nextMark = listObjectsResponse.getNextMarker();
          }

          ListObjectsRequest listObjectsRequest2 = new ListObjectsRequest(<BucketName>);
          listObjectsRequest2.setMarker(nextMark);
          listObjectsRequest2.setMaxKeys(20);
          listObjectsRequest2.setDelimiter(BceConfig.BOS_DELIMITER);
          ListObjectsResponse listing = client.listObjects(listObjectsRequest2);

          for (BosObjectSummary objectSummary : listing.getContents()) {
            System.out.println("ObjectKey: " + objectSummary.getKey());
          }

        } catch (BceServiceException e) {
          System.out.println("Error ErrorCode: " + e.getErrorCode());
          System.out.println("Error RequestId: " + e.getRequestId());
          System.out.println("Error StatusCode: " + e.getStatusCode());
          System.out.println("Error Message: " + e.getMessage());
          System.out.println("Error ErrorType: " + e.getErrorType());
        } catch (BceClientException e) {
          System.out.println("Error Message: " + e.getMessage());
        }
      }
    }).start();
}}
```

查询模拟文件夹

由于BOS本身是一个(<Key>,<Value>)的存储系统，所以原则上并不会存在“文件夹”的概念，但您可以通过 Delimiter 和 Prefix 参数的配合进行文件夹功能模拟。

假设Bucket中有5个文件：bos.jpg，fun/，fun/test.jpg，fun/movie/001.avi，fun/movie/007.avi，可以把 “/” 符号作为分隔符模拟文件夹。

递归列出模拟文件夹下所有文件

可以通过设置 Prefix 参数来获取某个模拟文件夹下所有的文件：

```
// 构造ListObjectsRequest请求
ListObjectsRequest listObjectsRequest = new ListObjectsRequest(<BucketName>);

// 递归列出fun文件夹下的所有文件
listObjectsRequest.setPrefix("fun/");

ListObjectsResponse listing = client.listObjects(listObjectsRequest);

// 遍历所有Object
System.out.println("Objects:");
for (BosObjectSummary objectSummary : listing.getContents()) {
  System.out.println(objectSummary.getKey());
}
```

输出：

```
Objects:
fun/
fun/movie/001.avi
fun/movie/007.avi
fun/test.jpg
```

查看模拟文件夹下的文件和子文件夹

在 Prefix 和 Delimiter 结合的情况下，可以列出模拟文件夹下的文件和子文件夹：

```
// 构造ListObjectsRequest请求
ListObjectsRequest listObjectsRequest = new ListObjectsRequest(<BucketName>);

// 指定"/" 为模拟文件夹的分隔符
listObjectsRequest.setDelimiter("/");

// 列出fun文件夹下的所有文件和子文件夹
listObjectsRequest.setPrefix("fun/");

ListObjectsResponse listing = client.listObjects(listObjectsRequest);

// 遍历所有Object
System.out.println("Objects:");
for (BosObjectSummary objectSummary : listing.getContents()) {
    System.out.println(objectSummary.getKey());
}

// 遍历所有CommonPrefix
System.out.println("\nCommonPrefixs:");
for (String commonPrefix : listing.getCommonPrefixes()) {
    System.out.println(commonPrefix);
}
```

输出：

```
Objects:
fun/
fun/test.jpg

CommonPrefixs:
fun/movie/
```

> **说明：**
>
> 返回的结果中， `Objects` 的列表中给出的是fun文件夹下的文件。而 `CommonPrefixs` 的列表中给出的是fun文件夹下的所有子文件夹。可以看出 `fun/movie/001.avi` ， `fun/movie/007.avi` 两个文件并没有被列出来，因为它们属于 `fun` 文件夹下的 `movie` 子文件夹下的文件。

获取Object

简单的获取Object

用户可以通过如下代码将Object读取到一个流中。

基本流程

1. 创建BOSClient类的实例。
2. 执行BOSClient.getObject()方法，会返回指定的BosObject。
3. 将BosObject读到流中，然后对流进行操作。

示例代码

```
// 获取Object，返回结果为BosObject对象
BosObject object = client.getObject(<BucketName>, <ObjectKey>);

// 获取ObjectMeta
ObjectMetadata meta = object.getObjectMetadata();

// 获取Object的输入流
InputStream objectContent = object.getObjectContent();

// 处理Object
...

// 关闭流
objectContent.close();
```

> **注意：**
>
> - BosObject中包含了Object的各种信息，包含Object所在的Bucket、Object的名称、MetaData以及一个输入流，用户可以通过操作输入流将Object的内容读取到文件或者内存中。
> - ObjectMetadata中包含了Object上传时定义的ETag，Http Header以及自定义的元数据。
> - 通过BosObject的getObjectContent()方法，还可以获取返回Object的输入流，用户可以读取这个输入流来对Object的内容进行操作。

完整示例

```
 import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import android.app.Activity;
import android.os.Bundle;
import com.baidubce.BceClientException;
import com.baidubce.BceServiceException;
import com.baidubce.auth.DefaultBceCredentials;
import com.baidubce.demo.R;
import com.baidubce.services.bos.BosClient;
import com.baidubce.services.bos.BosClientConfiguration;
import com.baidubce.services.bos.model.BosObject;
import com.baidubce.services.bos.model.ObjectMetadata;

public class ExampleActivity extends Activity {

private String bucketName = <BucketName>;
private String objectKey = <ObjectKey>;

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    new Thread(new Runnable() {
        @Override
        public void run() {
            try {
                BosClientConfiguration config = new BosClientConfiguration();
                config.setCredentials(new DefaultBceCredentials(<AccessKeyID>, <SecretAccessKey>));
                config.setEndpoint(<EndPoint>);
                BosClient client = new BosClient(config);

                // 获取Object，返回结果为BosObject对象
                BosObject object = client.getObject(<BucketName>, <ObjectKey>);

                // 获取ObjectMeta
                ObjectMetadata meta = object.getObjectMetadata();

                // 获取Object的输入流
                InputStream objectContent = object.getObjectContent();

                // 处理Object
                FileOutputStream fos=new FileOutputStream(android.os.Environment.getExternalStorageDirectory()+"/1/file");
                byte[] buffer=new byte[<bufferSize>];
                int count=0;
                while ((count=objectContent.read(buffer))>=0) {
                    fos.write(buffer,0,count);
                }

                System.out.println(meta.getETag());
                System.out.println(meta.getContentLength());

                // 关闭流
                objectContent.close();
                fos.close();

            } catch (BceServiceException e) {
                System.out.println("Error ErrorCode: " + e.getErrorCode());
                System.out.println("Error RequestId: " + e.getRequestId());
                System.out.println("Error StatusCode: " + e.getStatusCode());
                System.out.println("Error Message: " + e.getMessage());
                System.out.println("Error ErrorType: " + e.getErrorType());
            } catch (BceClientException e) {
                System.out.println("Error Message: " + e.getMessage());
            } catch (IOException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
        }
    }).start();
}}
```

通过GetObjectRequest获取Object

为了实现更多的功能，可以通过使用GetObjectRequest来获取Object。

基本流程

1. 创建GetObjectRequest类的实例。

2. 对GetObjectRequest执行setRange( )，获取Object中所需的字节数据。

3. 执行client.getObject( )操作。

示例代码

```
 // 新建GetObjectRequest
GetObjectRequest getObjectRequest = new GetObjectRequest(<BucketName>, <ObjectKey>);

// 获取0~100字节范围内的数据
getObjectRequest.setRange(0, 100);

// 获取Object，返回结果为BosObject对象
BosObject object = client.getObject(getObjectRequest);
```

说明：通过getObjectRequest的setRange( )方法可以设置返回Object的范围。用户可以用此功能实现文件的分段下载和断点续传。

获取下载进度

用户可以通过回调函数获取到下载的进度信息。

SDK提供的下载进度回调接口如下，您可以在其中定义在上传过程中您所需要的操作，如更新界面等等。

```
public interface BceProgressCallback<T extends AbstractBceRequest> {
        // request为下载的请求
        // currentSize为当前下载的大小(单位:byte)
        // totalSize为本次请求需要下载的总大小(单位:byte)
        void onProgress(T request, long currentSize, long totalSize);
    }
```

示例代码

```
GetObjectRequest request = new GetObjectRequest()
        .withBucketName(this.bucketName)
        .withKey(objectKey)
        .withRange(0, 5);
request.setProgressCallback(new BosProgressCallback<GetObjectRequest>() {
    @Override
    public void onProgress(GetObjectRequest request, long currentSize, long totalSize) {
        Log.e(currentSize + "", totalSize + "");
    }
});
InputStream input = this.client.getObject(request).getObjectContent();
```

> **注意：** 下载进度接口与上传进度接口有所不同，无法指定每次回调时，下载量增长的片段大小。

下载Object到指定路径

用户可以通过如下代码直接将Object下载到指定路径。

基本流程

1. 创建GetObjectRequest类的实例。
2. 执行client.getObject( )操作。
3. Object可以直接下载到指定路径。

示例代码

```
// 新建GetObjectRequest
GetObjectRequest getObjectRequest = new GetObjectRequest(<BucketName>, <ObjectKey>);

// 下载Object到文件
ObjectMetadata objectMetadata = client.getObject(getObjectRequest, new File("/path/to/file","w+"));
```

> **说明：**当使用上面方法将Object直接下载到指定路径时，方法会返回ObjectMetadata对象。

获取Object的storageClass

Object的storage class属性分为STANDARD(标准存储)、STANDARD_IA(低频存储)、COLD(冷存储)和ARCHIVE（归档存储）。

示例代码

```
public void getObjectStorageClass(){
//...
ObjectMetadata meta = client.getObjectMetadata(bucketName, key);
String storageClass = meta.getStorageClass();
}
```

只获取ObjectMetadata

通过 getObjectMetadata() 方法可以只获取ObjectMetadata而不获取Object的实体。

示例代码

```
ObjectMetadata objectMetadata = client.getObjectMetadata(<BucketName>, <ObjectKey>);
```

获取Object的URL

您可以通过如下代码获取指定Object的URL，该功能通常用于您将Object的URL临时分享给其他用户的场景。

基本流程

1. 创建BOSClient类的实例。
2. 执行BOSClient.generatePresignedUrl( )方法。
3. 返回一个Object的URL。

示例代码

```
URL url = client.generatePresignedUrl(<BucketName>, <ObjectKey>, <ExpirationInSeconds>);
```

> **说明：**`ExpirationInSeconds`为指定的URL有效时长，时间从当前时间算起，为可选参数，不配置时系统默认值为1800秒。如果要设置为永久不失效的时间，可以将`ExpirationInSeconds`参数设置为 -1，不可设置为其他负数。

完整示例

```
import java.net.URL;
import android.app.Activity;
import android.content.Intent;
import android.net.Uri;
import android.os.Bundle;
import com.baidubce.BceClientException;
import com.baidubce.BceServiceException;
import com.baidubce.auth.DefaultBceCredentials;
import com.baidubce.demo.R;
import com.baidubce.services.bos.BosClient;
import com.baidubce.services.bos.BosClientConfiguration;

public class ExampleActivity extends Activity {
private String bucketName = <BucketName>;
private String objectKey = <ObjectKey>;

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    new Thread(new Runnable() {
        @Override
        public void run() {
            try {
                BosClientConfiguration config = new BosClientConfiguration();
                config.setCredentials(new DefaultBceCredentials(<AccessKeyID>, <SecretAccessKey>));
                config.setEndpoint(<EndPoint>);
                BosClient client = new BosClient(config);

                //获取Object的URL
                URL url = client.generatePresignedUrl(<BucketName>, <ObjectKey>, 2000);
                Uri uri = Uri.parse(url.toString());
                Intent intent = new Intent(Intent.ACTION_VIEW, uri);

                //执行URL
                startActivity(intent);

            } catch (BceServiceException e) {
                System.out.println("Error ErrorCode: " + e.getErrorCode());
                System.out.println("Error RequestId: " + e.getRequestId());
                System.out.println("Error StatusCode: " + e.getStatusCode());
                System.out.println("Error Message: " + e.getMessage());
                System.out.println("Error ErrorType: " + e.getErrorType());
            } catch (BceClientException e) {
                System.out.println("Error Message: " + e.getMessage());
            }
        }
    }).start();
}}
```

Object权限控制

设置Object的访问权限

目前BOS支持两种方式设置ACL。第一种是使用Canned Acl，在PutObjectAcl的时候，通过头域的"x-bce-acl"或者"x-bce-grant-permission'来设置object访问权限，当前可设置的权限包括private和public-read，两种类型的header不可以同时在一个请求中出现。第二种方式是上传一个ACL文件。

详细信息请参考设置Object权限控制。

1、通过使用头域的"x-bce-acl"或者"x-bce-grant-permission'来设置object访问权限

- set object acl 第一种方式(以请求头方式设置)

```
SetObjectAclRequest setObjectAclRequest = new SetObjectAclRequest("yourBucketName","objectKey",CannedAccessControlList.PublicRead);
client.setObjectAcl(setObjectAclRequest);
```

- set object acl 第一种方式(以请求头方式设置 xBceGrantRead)

```
String xBceGrantRead = "id=\"user_id1\""+",id=\"user_id2\"";
SetObjectAclRequest setObjectAclRequest = new SetObjectAclRequest();
setObjectAclRequest.withBucketName("yourBucketName");
setObjectAclRequest.withKey("objectKey");
setObjectAclRequest.setxBceGrantRead(xBceGrantRead);
client.setObjectAcl(setObjectAclRequest);
```

- set object acl 第一种方式(已请求头方式设置 xBceGrantFullControl)

```
String xBceGrantFullControl = "id=\"user_id1\""+",id=\"user_id2\"";
SetObjectAclRequest setObjectAclRequest = new SetObjectAclRequest();
setObjectAclRequest.withBucketName("yourBucketName");
setObjectAclRequest.withKey("objectKey");
setObjectAclRequest.setxBceGrantFullControl(xBceGrantFullControl);
client.setObjectAcl(setObjectAclRequest);
```

2、通过setObjectAcl设置object访问权限

- set object acl 第二种方式(json字符串)

```
String jsonObjectAcl = "{\"accessControlList\":["+ "{\"grantee\":[{\"id\":\"*\"}], "+ "\"permission\":[\"FULL_CONTROL\"]"+"}]}";

SetObjectAclRequest setObjectAclRequest = new SetObjectAclRequest("yourBucketName","objectKey",jsonObjectAcl);

client.setObjectAcl(setObjectAclRequest);
```

- set object acl 第二种方式，用户只需指定指定参数即可

```
List<Grant> grants = new ArrayList<Grant>();
List<Grantee> grantees = new ArrayList<Grantee>();
List<Permission> permissions = new ArrayList<Permission>();

// 授权给特定用户
grantees.add(new Grantee("user_id1"));
grantees.add(new Grantee("user_id2"));
grantees.add(new Grantee("user_id3"));

// 设置权限
permissions.add(Permission.READ);
grants.add(new Grant().withGrantee(grantees).withPermission(permissions));

SetObjectAclRequest setObjectAclRequest = new SetObjectAclRequest("yourBucketName","objectKey", grants);
client.setObjectAcl(setObjectAclRequest);
```

查看Object的权限

如下代码可以查Object的权限：

```
GetObjectAclRequest getObjectRequest = new GetObjectAclRequest();
getObjectRequest.withBucketName("yourBucketName");
getObjectRequest.withKey("objectKey");
GetObjectAclResponse response = client.getObjectAcl(getObjectRequest);
```

getObjectAcl方法返回的解析类中可供调用的参数有：

参数	说明
accessControlList	标识Object的权限列表
grantee	标识被授权人
-id	被授权人ID
permission	标识被授权人的权限

删除Object的权限

如下代码可以删除Object的权限：

```
DeleteObjectAclRequest deleteObjectAclRequest = new DeleteObjectAclRequest("yourBucketName","objectKey");

client.deleteObjectAcl(deleteObjectAclRequest);
```

删除Object

基本流程

1. 创建BOSClient类的实例。
2. 执行BOSClient.deleteObject()方法。
3. 若操作失败后抛出异常，若成功无返回值。

示例代码

```
// 删除Object
client.deleteObject(<BucketName>, <ObjectKey>);  //指定要删除的Object所在Bucket名称和该Object名称
```

完整示例

```
 import android.app.Activity;
import android.os.Bundle;
import com.baidubce.BceClientException;
import com.baidubce.BceServiceException;
import com.baidubce.auth.DefaultBceCredentials;
import com.baidubce.demo.R;
import com.baidubce.services.bos.BosClient;
import com.baidubce.services.bos.BosClientConfiguration;

public class ExampleActivity extends Activity {

private String bucketName = <BucketName>;
private String objectKey = <ObjectKey>;

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    new Thread(new Runnable() {
        @Override
        public void run() {
            try {
                BosClientConfiguration config = new BosClientConfiguration();
                config.setCredentials(new DefaultBceCredentials(<AccessKeyID>, <SecretAccessKey>));
                config.setEndpoint(<EndPoint>);
                BosClient client = new BosClient(config);

                // 删除Object
                client.deleteObject(<BucketName>, <ObjectKey>);   //指定要删除的Object所在Bucket名称和该Object名称

            } catch (BceServiceException e) {
                System.out.println("Error ErrorCode: " + e.getErrorCode());
                System.out.println("Error RequestId: " + e.getRequestId());
                System.out.println("Error StatusCode: " + e.getStatusCode());
                System.out.println("Error Message: " + e.getMessage());
                System.out.println("Error ErrorType: " + e.getErrorType());
            } catch (BceClientException e) {
                System.out.println("Error Message: " + e.getMessage());
            }
        }
    }).start();
}}
```

拷贝Object

简单拷贝Object

基本流程

1. 创建BOSClient 类的实例。

2. 执行BOSClient.copyObject( )方法。

3. 返回CopyObjectResponse类实例，可通过getETag()/getLastModified()获取eTag和最后修改时间。

示例代码

```
 // 拷贝Object
CopyObjectResponse copyObjectResponse = client.copyObject(<SrcBucketName>, <SrcKey>, <DestBucketName>, <DestKey>);  //SrcBucketName, SrcKey为原地址，DestBucketName, DestKey为拷贝到的目的地址

// 打印结果
System.out.println("ETag: " + copyObjectResponse.getETag() + " LastModified: " + copyObjectResponse.getLastModified());
```

完整示例

```
 import android.app.Activity;
import android.os.Bundle;
import com.baidubce.BceClientException;
import com.baidubce.BceServiceException;
import com.baidubce.auth.DefaultBceCredentials;
import com.baidubce.demo.R;
import com.baidubce.services.bos.BosClient;
import com.baidubce.services.bos.BosClientConfiguration;
import com.baidubce.services.bos.model.CopyObjectResponse;

public class ExampleActivity extends Activity {

private String srcBucketName = <SrcBucketName>;
private String srcKey = <SrcKey>;

private String destBucketName = <DestBucketName>;
private String destKey = <DestKey>;

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    new Thread(new Runnable() {
        @Override
        public void run() {
            try {
                BosClientConfiguration config = new BosClientConfiguration();
                config.setCredentials(new DefaultBceCredentials(<AccessKeyID>, <SecretAccessKey>));
                config.setEndpoint(<EndPoint>);
                BosClient client = new BosClient(config);

                // 拷贝Object
                CopyObjectResponse copyObjectResponse = client.copyObject(<SrcBucketName>, <SrcKey>, <DestBucketName>, <DestKey>);

                // 打印结果
                System.out.println("ETag: " + copyObjectResponse.getETag() + " LastModified: " + copyObjectResponse.getLastModified());

            } catch (BceServiceException e) {
                System.out.println("Error ErrorCode: " + e.getErrorCode());
                System.out.println("Error RequestId: " + e.getRequestId());
                System.out.println("Error StatusCode: " + e.getStatusCode());
                System.out.println("Error Message: " + e.getMessage());
                System.out.println("Error ErrorType: " + e.getErrorType());
            } catch (BceClientException e) {
                System.out.println("Error Message: " + e.getMessage());
            }
        }
    }).start();
}}
```

> **说明：**copyObject 方法返回一个 `CopyObjectResponse` 对象，该对象中包含了新Object的ETag和修改时间。

通过CopyObjectRequest拷贝Object

您也可以通过 `CopyObjectRequest` 实现Object的拷贝。该功能一般用于如下场景：

- Copy一个Object但重新设置meta。
- 重置某个现有Object的meta（把Src和Des设置为同一个Object）。

基本流程

1. 创建CopyObjectRequest类的实例，传入`<SrcBucketName>`，`<SrcKey>`，`<DestBucketName>`，`<DestKey>`参数。
2. 创建ObjectMetadata类的实例。
3. 返回CopyObjectResponse类实例，可通过getETag()/getLastModified()获取eTag和最后修改时间。

示例代码

```
 // 创建CopyObjectRequest对象
CopyObjectRequest copyObjectRequest = new CopyObjectRequest(<SrcBucketName>, <SrcKey>, <DestBucketName>, <DestKey>);

// 设置新的Metadata
Map<String, String> userMetadata = new HashMap<String, String>();
userMetadata.put(<UserMetaKey>,<UserMetaValue>);
meta.setUserMetadata(userMetadata);
copyObjectRequest.setNewObjectMetadata(meta);

// 复制Object
CopyObjectResponse copyObjectResponse = client.copyObject(copyObjectRequest);

System.out.println("ETag: " + copyObjectResponse.getETag() + " LastModified: " + copyObjectResponse.getLastModified());
```

> **说明：**`CopyObjectRequest` 允许用户修改目的Object的ObjectMeta，同时也提供 `MatchingETagConstraints` 参数的设定。

Object的分块上传

分块上传的场景

除了通过putObject()方法上传文件到BOS以外，BOS还提供了另外一种上传模式：分块上传（Multipart Upload）。用户可以在如下的应用场景内（但不仅限于此），使用分块上传模式，如：

- 需要支持断点上传。
- 上传超过5GB大小的文件。

- 网络条件较差，和BOS的服务器之间的连接经常断开。
- 需要流式地上传文件。
- 上传文件之前，无法确定上传文件的大小。

Multipart Upload分块上传流程

假设有一个文件，本地路径为 `/path/to/file.zip` ，由于文件比较大，使用分块上传其传输到BOS中。

基本流程：

1. 初始化Multipart Upload。
2. 上传分块。
3. 完成分块上传。

初始化Multipart Upload

使用 `initiateMultipartUpload` 方法来初始化一个分块上传事件：

示例代码：

```
 // 开始Multipart Upload
InitiateMultipartUploadRequest initiateMultipartUploadRequest =
      new InitiateMultipartUploadRequest(<BucketName>, <ObjectKey>);
InitiateMultipartUploadResponse initiateMultipartUploadResponse =
      client.initiateMultipartUpload(initiateMultipartUploadRequest);

// 打印UploadId
System.out.println("UploadId: " + initiateMultipartUploadResponse.getUploadId());
```

> **说明：** `initiateMultipartUpload` 的返回结果中含有 `UploadId` ，它是区分分块上传事件的唯一标识，在后面的操作中，我们将用到它。

上传分块

将文件分块上传。

示例代码：

```
 // 设置每块为 5MB
final long partSize = 1024 * 1024 * 5L;

File partFile = new File("/path/to/file.zip");

// 计算分块数目
int partCount = (int) (partFile.length() / partSize);
if (partFile.length() % partSize != 0) {
    partCount++;
}
// 新建一个List保存每个分块上传后的ETag和PartNumber
List<PartETag> partETags = new ArrayList<PartETag>();
// 获取文件流
FileInputStream fis = new FileInputStream(partFile);
for (int i = 0; i < partCount; i++) {

    // 计算每个分块的大小
    long skipBytes = partSize * i;
    long size = partSize < partFile.length() - skipBytes ?
            partSize : partFile.length() - skipBytes;

    byte[] buf = new byte[(int)size];
    int offset = 0;
    while (true) {
        int byteRead = fis.read(buf, offset, (int)size);
        offset += byteRead;
        if (byteRead < 0 || offset >= size) {
            break;
        }
    }
    ByteArrayInputStream bufStream = new ByteArrayInputStream(buf);

    // 创建UploadPartRequest，上传分块
    UploadPartRequest uploadPartRequest = new UploadPartRequest();
    uploadPartRequest.setBucketName(bucketName);
    uploadPartRequest.setKey(objectkey);
    uploadPartRequest.setUploadId(initiateMultipartUploadResponse.getUploadId());
    uploadPartRequest.setInputStream(bufStream);
    uploadPartRequest.setPartSize(size);
    uploadPartRequest.setPartNumber(i + 1);
    // 上传进度回调
    uploadPartRequest.setProgressCallback(new BosProgressCallback<UploadPartRequest>() {
        @Override
        public void onProgress(UploadPartRequest request, long currentSize, long totalSize) {
            Log.e(currentSize + "", totalSize + "");
        }
    });

    UploadPartResponse uploadPartResponse = client.uploadPart(uploadPartRequest);

    // 将返回的PartETag保存到List中。
    partETags.add(uploadPartResponse.getPartETag());

    System.out.println(uploadPartResponse.getPartETag());
}
// 关闭文件
fis.close();
```

注意：上面代码的核心是调用 `UploadPart` 方法来上传每一个分块，但是要注意以下几点：

- UploadPart 方法要求Part大小是1MB的整数倍或大于5MB。但是Upload Part接口并不会立即校验上传Part的大小；只有当Complete Multipart Upload的时候才会校验。
- 为了保证数据在网络传输过程中不出现错误，建议您在UploadPart后，使用每个分块BOS返回的Content-MD5值分别验证已上传分块数据的正确性。当所有分块数据合成一个Object后，不再含MD5值。
- Part号码的范围是1~10000。如果超出这个范围，BOS将返回InvalidArgument的错误码。
- 每次上传Part时都要把流定位到此次上传块开头所对应的位置。
- 每次上传Part之后，BOS的返回结果会包含一个 `PartETag` 对象，它是上传块的ETag与块编号（PartNumber）的组合，在后续完成分块上传的步骤中会用到它，因此需要将其保存起来。一般来讲这些 `PartETag` 对象将被保存到List中。
- 进度回调接口的使用可以参考"获取上传进度"一章。

完成分块上传

示例代码

```
 CompleteMultipartUploadRequest completeMultipartUploadRequest =
    new CompleteMultipartUploadRequest(<BucketName>, <ObjectKey>,
        initiateMultipartUploadResponse.getUploadId(), partETags);

// 完成分块上传
CompleteMultipartUploadResponse completeMultipartUploadResponse =
    client.completeMultipartUpload(completeMultipartUploadRequest);

// 打印Object的ETag
System.out.println(completeMultipartUploadResponse.getETag());
```

说明：上面代码中的 `partETags` 是第二步中保存的partETag的列表，BOS收到用户提交的Part列表后，会逐一验证每个数据Part的有效性。当所有的数据Part验证通过后，BOS将把这些数据part组合成一个完整的Object。

完整示例

```
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;
import org.json.JSONException;
import android.app.Activity;
import android.os.Bundle;
import com.baidubce.BceClientException;
import com.baidubce.BceServiceException;
import com.baidubce.auth.DefaultBceCredentials;
import com.baidubce.demo.R;
import com.baidubce.services.bos.BosClient;
import com.baidubce.services.bos.BosClientConfiguration;
import com.baidubce.services.bos.model.CompleteMultipartUploadRequest;
import com.baidubce.services.bos.model.CompleteMultipartUploadResponse;
import com.baidubce.services.bos.model.InitiateMultipartUploadRequest;
import com.baidubce.services.bos.model.InitiateMultipartUploadResponse;
import com.baidubce.services.bos.model.PartETag;
import com.baidubce.services.bos.model.UploadPartRequest;
import com.baidubce.services.bos.model.UploadPartResponse;

public class ExampleActivity extends Activity {

private String bucketName = <BucketName>;
private String objectKey = <ObjectKey>;

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    new Thread(new Runnable() {
        @Override
        public void run() {
            try {
                BosClientConfiguration config = new BosClientConfiguration();
                config.setCredentials(new DefaultBceCredentials(<AccessKeyID>, <SecretAccessKey>));
                config.setEndpoint(<EndPoint>);
                BosClient client = new BosClient(config);

                // 开始Multipart Upload
                InitiateMultipartUploadRequest initiateMultipartUploadRequest = new InitiateMultipartUploadRequest(<BucketName>, <ObjectKey>);
                InitiateMultipartUploadResponse initiateMultipartUploadResponse =
                        client.initiateMultipartUpload(initiateMultipartUploadRequest);

                // 打印UploadId
                System.out.println("UploadId: " + initiateMultipartUploadResponse.getUploadId());

                // 设置每块为 5MB
                final long partSize = 1024 * 1024 * 5L;

                File partFile = new File("/path/to/file.zip");

                // 计算分块数目
                int partCount = (int) (partFile.length() / partSize);
                if (partFile.length() % partSize != 0) {
                    partCount++;
                }

                // 新建一个List保存每个分块上传后的ETag和PartNumber
                List<PartETag> partETags = new ArrayList<PartETag>();

                // 获取文件流
                FileInputStream fis = new FileInputStream(partFile);
                for (int i = 0; i < partCount; i++) {

                    // 计算每个分块的大小
                    long skipBytes = partSize * i;
                    long size = partSize < partFile.length() - skipBytes ?
                            partSize : partFile.length() - skipBytes;

                    byte[] buf = new byte[(int)size];
                    int offset = 0;
                    while (true) {
                        int byteRead = fis.read(buf, offset, (int)size);
                        offset += byteRead;
                        if (byteRead < 0 || offset >= size) {
                            break;
                        }
                    }
                    ByteArrayInputStream bufStream = new ByteArrayInputStream(buf);

                    // 创建UploadPartRequest，上传分块
                    UploadPartRequest uploadPartRequest = new UploadPartRequest();
                    uploadPartRequest.setBucketName(<BucketName>);
                    uploadPartRequest.setKey(<ObjectKey>);
                    uploadPartRequest.setUploadId(initiateMultipartUploadResponse.getUploadId());
                    uploadPartRequest.setInputStream(fis);
                    uploadPartRequest.setPartSize(size);
                    uploadPartRequest.setPartNumber(i + 1);
                    UploadPartResponse uploadPartResponse = client.uploadPart(uploadPartRequest);

                    // 将返回的PartETag保存到List中。
                    partETags.add(uploadPartResponse.getPartETag());

                    System.out.println(uploadPartResponse.getPartETag());
                }
                // 关闭文件
                fis.close();
```

```
            CompleteMultipartUploadRequest completeMultipartUploadRequest = new CompleteMultipartUploadRequest(<BucketName>, <ObjectKey>, initiateMultipartUploadResponse.getUploadId(), partETags);

            // 完成分块上传
            CompleteMultipartUploadResponse completeMultipartUploadResponse =
                    client.completeMultipartUpload(completeMultipartUploadRequest);

            // 打印Object的ETag
            System.out.println(completeMultipartUploadResponse.getETag());

          } catch (BceServiceException e) {
              System.out.println("Error ErrorCode: " + e.getErrorCode());
              System.out.println("Error RequestId: " + e.getRequestId());
              System.out.println("Error StatusCode: " + e.getStatusCode());
              System.out.println("Error Message: " + e.getMessage());
              System.out.println("Error ErrorType: " + e.getErrorType());
          } catch (BceClientException e) {
              System.out.println("Error Message: " + e.getMessage());
          } catch (IOException e) {
              // TODO Auto-generated catch block
              e.printStackTrace();
          } catch (JSONException e) {
              // TODO Auto-generated catch block
              e.printStackTrace();
          }
        }
    }).start();
}}
```

取消分块上传

用户可以使用abortMultipartUpload方法取消分块上传。

示例代码：

```
 AbortMultipartUploadRequest abortMultipartUploadRequest =
   new AbortMultipartUploadRequest(<BucketName>, <ObjectKey>, <UploadId>);

// 取消分块上传
client.abortMultipartUpload(abortMultipartUploadRequest);
```

获取未完成的分块上传

用户可以使用 listMultipartUploads 方法获取Bucket内未完成的分块上传事件。

基本流程

1. 创建listMultipartUploadsRequest类的实例,传入<BucketName>参数。

2. 创建BOSClient类的实例,执行BOSClient.listMultipartUploads( )方法。

3. listMultipartUploads()返回所有已上传part的信息。

示例代码

```
 ListMultipartUploadsRequest listMultipartUploadsRequest =
   new ListMultipartUploadsRequest(<BucketName>);

// 获取Bucket内所有上传事件
ListMultipartUploadsResponse list = client.listMultipartUploads(listMultipartUploadsRequest);

// 遍历所有上传事件
for (MultipartUploadSummary multipartUpload : list.getMultipartUploads()) {
    System.out.println("Key: " + multipartUpload.getKey() + " UploadId: " + multipartUpload.getUploadId());
}
```

> **注意：**
>
> - 默认情况下，如果Bucket中的分块上传事件的数目大于1000，则只会返回1000个Object，并且返回结果中IsTruncated的值为True，同时返回NextKeyMarker作为下次读取的起点。
> - 若想获取更多分块上传事件，可以使用KeyMarker参数分次读取。

完整示例

```
import android.app.Activity;
import android.os.Bundle;
import com.baidubce.BceClientException;
import com.baidubce.BceServiceException;
import com.baidubce.auth.DefaultBceCredentials;
import com.baidubce.demo.R;
import com.baidubce.services.bos.BosClient;
import com.baidubce.services.bos.BosClientConfiguration;
import com.baidubce.services.bos.model.ListMultipartUploadsRequest;
import com.baidubce.services.bos.model.ListMultipartUploadsResponse;
import com.baidubce.services.bos.model.MultipartUploadSummary;

public class ExampleActivity extends Activity {

private String bucketName = <BucketName>;

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    new Thread(new Runnable() {
        @Override
        public void run() {
            try {
                BosClientConfiguration config = new BosClientConfiguration();
                config.setCredentials(new DefaultBceCredentials(<AccessKeyID>, <SecretAccessKey>));
                config.setEndpoint(<EndPoint>);
                BosClient client = new BosClient(config);

                ListMultipartUploadsRequest listMultipartUploadsRequest =
                        new ListMultipartUploadsRequest(<BucketName>);

                   // 获取Bucket内所有上传事件
                ListMultipartUploadsResponse listing = client.listMultipartUploads(listMultipartUploadsRequest);

                 // 遍历所有上传事件
                 for (MultipartUploadSummary multipartUpload : listing.getMultipartUploads()) {
                     System.out.println("Key: " + multipartUpload.getKey() + " UploadId: " + multipartUpload.getUploadId());
                 }

            } catch (BceServiceException e) {
                System.out.println("Error ErrorCode: " + e.getErrorCode());
                System.out.println("Error RequestId: " + e.getRequestId());
                System.out.println("Error StatusCode: " + e.getStatusCode());
                System.out.println("Error ErrorType: " + e.getErrorType());
                System.out.println("Error Message: " + e.getMessage());
            } catch (BceClientException e) {
                System.out.println("Error Message: " + e.getMessage());
            }
        }
    }).start();
}}
```

获取所有已上传的分块信息

用户可以使用 listParts 方法获取某个上传事件中所有已上传的块。

基本流程

1. 创建ListPartsRequest类的实例,传入<BucketName>,<ObjectKey>, <UploadId>参数。

2. 创建BOSClient类的实例,执行BOSClient.listParts( )方法。

3. listParts()返回所有已上传part的信息。

示例代码

```
ListPartsRequest listPartsRequest = new ListPartsRequest(<BucketName>, <ObjectKey>, <UploadId>);

// 获取上传的所有Part信息
ListPartsResponse partListing = client.listParts(listPartsRequest);

// 遍历所有Part
for (PartSummary part : partListing.getParts()) {
    System.out.println("PartNumber: " + part.getPartNumber() + " ETag: " + part.getETag());
}
```

> **注意：**
>
> - 默认情况下，如果Bucket中的分块上传事件的数目大于1000，则只会返回1000个Object，并且返回结果中IsTruncated的值为True，同时返回NextPartNumberMarker作为下次读取的起点。
> - 若想获取更多已上传的分块信息，可以使用PartNumberMarker参数分次读取。

完整示例

```
import android.app.Activity;
import android.os.Bundle;
import com.baidubce.BceClientException;
import com.baidubce.BceServiceException;
import com.baidubce.auth.DefaultBceCredentials;
import com.baidubce.demo.R;
import com.baidubce.services.bos.BosClient;
import com.baidubce.services.bos.BosClientConfiguration;
import com.baidubce.services.bos.model.ListPartsRequest;
import com.baidubce.services.bos.model.ListPartsResponse;
import com.baidubce.services.bos.model.PartSummary;

public class ExampleActivity extends Activity {

private String bucketName = <BucketName>;
private String objectKey = <ObjectKey>;
private String uploadId = <UploadId>;

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    new Thread(new Runnable() {
        @Override
        public void run() {
            try {
                BosClientConfiguration config = new BosClientConfiguration();
                config.setCredentials(new DefaultBceCredentials(<AccessKeyID>, <SecretAccessKey>));
                config.setEndpoint(<EndPoint>);
                BosClient client = new BosClient(config);

                ListPartsRequest listPartsRequest = new ListPartsRequest(<BucketName>, <ObjectKey>, <UploadId>);
                listPartsRequest.setMaxParts(100);
                listPartsRequest.setPartNumberMarker(50);

                // 获取上传的所有Part信息
                ListPartsResponse partListing = client.listParts(listPartsRequest);

                // 遍历所有Part
                for (PartSummary part : partListing.getParts()) {
                    System.out.println("PartNumber: " + part.getPartNumber() + " ETag: " + part.getETag());
                }

            } catch (BceServiceException e) {
                System.out.println("Error ErrorCode: " + e.getErrorCode());
                System.out.println("Error RequestId: " + e.getRequestId());
                System.out.println("Error StatusCode: " + e.getStatusCode());
                System.out.println("Error ErrorType: " + e.getErrorType());
                System.out.println("Error Message: " + e.getMessage());
            } catch (BceClientException e) {
                System.out.println("Error Message: " + e.getMessage());
            }
        }
    }).start();
}}
```

封装分块上传

在Android SDK中，Bos为用户提供了putSuperObjectFromFile接口，它对分块上传涉及到的initiateMultipartUpload、UploadPart、completeMultipartUpload三个方法进行封装，用户只需调用该接口即可完成分块上传，同时，支持进度同步回调。

- 简单示例:

```
File file = new File("/path/to/file.zip");
PutSuperObjectRequest request = new PutSuperObjectRequest(bucketName, objectKey, file);
bosClient.putSuperObjectFromFile(request);
```

- 示例：设置分块大小为2MB，线程数为4，并且每1024个byte同步进度回调

```
BosClientConfiguration config=new BosClientConfiguration();
config.setUploadSegmentPart(1024);
File file = new File("/path/to/file.zip");
PutSuperObjectRequest request = new PutSuperObjectRequest(bucketName, objectKey,
                                        file, 1024 * 1024 * 2L, 4);
request.setProgressCallback(new BosProgressCallback<PutSuperObjectRequest>() {
    @Override
    public void onProgress(PutSuperObjectRequest request, long currentSize, long totalSize) {
        Log.e(currentSize + "", totalSize + "");
    }
});
bosClient.putSuperObjectFromFile(request);
```

> **注意：**
>
> - 默认情况下，分块大小是5MB，线程数是5，进度回调周期是每2048个byte。
> - 若一个大文件耗时很长，用户想结束分块上传，可调用PutSuperObjectRequest中的cancel()方法。

异常处理

BOS异常提示有如下两种方式：

异常方法	说明
BceClientException	客户端异常
BceServiceException	服务器异常

用户可以使用try获取某个事件所产生的异常。

- 示例代码

```
  String objectKey = "testObjectToFile";
bosclient.putObject(<BucketName>, <ObjectKey>, <DataFile>);
File file = new File("test");

try {
   bosclient.getObject(this.bucketName, objectKey, file);
  } catch (BceServiceException bce){
    System.out.println("Error ErrorCode: " + e.getErrorCode());
    System.out.println("Error RequestId: " + e.getRequestId());
    System.out.println("Error StatusCode: " + e.getStatusCode());
    System.out.println("Error ErrorType: " + e.getErrorType());
    System.out.println("Error Message: " + e.getMessage());
  } catch ( BceClientException bce){
    System.out.println( bce.getMessage());
  } finally {
    file.delete();
  }
```

版本变更记录

V1.0.10 [2020-05-09]

- 封装分块上传
- 支持HTTPDNS

V1.0.9 [2019-11-15]

- 支持BOS服务端同步回调接口
- 优化上传逻辑

V1.0.7 [2019-02-19]

- 支持自定义域名和CDN加速域名

V1.0.6 [2019-01-25]

- 支持取消接口
- 优化上传框架

V1.0.5 [2018-09-13]

- 解决客户端时间偏差导致上传问题

V1.0.4 [2017-12-27]

- 支持上传下载进度回调接口
- 修复不同时区用户使用导致请求失败的问题

V1.0.3 [2017-07-13]

- SDK内核替换及结构调整
- 支持header丰富、STS、Appendfile以及低频、冷存储

V1.0.2 [2016-04-07]

- 支持https
- 修改`BosClientConfiguration`设置`Protocol`策略，以设置的url中的为准

V1.0.1 [2015-12-10]

首次发布：

- 支持创建、查看、罗列、删除 Bucket
- 修改、获取Bucket的访问权限
- 上传、查看、罗列、删除Object
- 生成下载Object的URL
- 支持STS 验证
- 支持多服务

日志

在调试时使用如下代码打开/关闭SDK运行时的Log。

- 示例代码

```
  ......
import com.baidubce.util.BLog;
......
BLog.disableLog();
BLog.enableLog();
```

IOS-SDK

概述

本文档主要介绍BOS iOS SDK的安装和使用。在使用本文档前，您需要先了解BOS的一些基本知识，并已经开通了BOS服务。若您还不了解BOS，可以参考产品描述和开发者指南。

BOS iOS SDK帮助移动开发者实现在iOS端简单便捷地使用BOS存储服务。通过该SDK，开发者开发的APP可以直接从终端向BOS发送请求并与BOS进行交互操作。

安装SDK工具包

安装步骤

1. 在官方网站下载SDK压缩包。
2. 解压得到frameworks。frameworks包含Basic、BOS和STS三部分，其中Basic和BOS为必须部分，STS根据是否进行临时授权可选。
3. 根据需要将frameworks复制到您的Xcode工程目录下，例如lib目录。
4. 在Xcode右键工程->Add files to "your project"，导入您刚才复制的frameworks。
5. 在Build Setttings中的Other Linker Flags选项里需要加上-ObjC。

配置 App Transport Security (ATS)

> **说明：**如果BOS使用https endpoint则不需要进行ATS相关配置，只有使用http endpoint时需要配置info.plist文件。

从 iOS 9.0 和 OS X v10.11 开始，App Transport Security (ATS)功能默认开启。ATS强制使用HTTPS访问Web服务，并且必须使用TLS v1.2以上版本保证通信的安全性。ATS的详细需求请参考：Requirements for Connecting

如果App支持iOS 9.0而且需要访问不满足上述要求的Web服务，必须在info.plist文件中使用 `NSAppTransportSecurity`元素添加例外处理规则。`NSAppTransportSecurity`的使用方法请参考：App Transport Security

SDK目录结构

```
 Baidu-BOS-SDK-iOS-v1.0.0
|_BaiduBCEBasic.framework                // 基础模块
| |____Headers
|  |____BCEClient.h                    // BCE服务基类
|  |____BCEClientConfiguration.h       // BCE服务配置基类
|  |____BCECredentials.h               // BCE AK、SK签名类
|  |____BCEError.h                     // BCE通用的错误码
|  |____BCERegion.h                    // BCE提供服务的区域
|  |____BCERequest.h                   // BCE网络请求类
|  |____BCEResponse.h                  // BCE网络响应类
|  |____BCEResponseMetadata.h          // BCE网络响应元数据类
|  |____BCESTSCredentials.h            // BCE临时授权签名类
|  |____BCETask.h                      // BCE通用的任务类
|
|_BaiduBCEBOS.framework                 // BOS模块
| |____Headers
|  |____BOSClient.h                    // BOS客户端入口类
|  |____BOSClientConfiguration.h       // BOS特有的Client配置
|  |____BOSListBucketModel.h           // 罗列Bucket相关类
|  |____BOSGetOBjectModel.h            // 下载Object相关类
|  |____ .
|  |____ .                             // ...
|  |____ .
|  |____BOSPutObjectModel.h            // 上传Object相关类
|
|_BaiduBCESTS.framework                 // STS授权模块
| |____Headers
|  |____STSClient.h                    // STS服务入口类
|  |____STSClientConfiguration.h       // STS特有的Client配置
|  |____STSGetSessionTokenModel.h      // 获取token相关类
```

初始化

快速入门

1. 初始化BOSClient。

   BOSClient是与BOS服务交互的客户端，BOS iOS SDK的BOS操作都是通过BOSClient完成的。

   新建带有STS验证方式的BOSClient：

   示例代码：

```
 BCESTSCredentials* credentials = [[BCESTSCredentials alloc] init];
 // access key和secret key分别代表AKSK。
credentials.accessKey = @"<access key>"; // 临时返回的AK
credentials.secretKey = @"<secret key>"; // 临时返回的SK
credentials.sessionToken = @"<session token>"; // 临时返回的SessionToken

BOSClientConfiguration* configuration = [[BOSClientConfiguration alloc] init];
configuration.credentials = credentials;

BOSClient* client = [[BOSClient alloc] initWithConfiguration:configuration];
```

2. 新建一个Bucket。

   Bucket是BOS上的命名空间，相当于数据的容器，可以存储若干数据实体（Object）。在您上传数据前，必须先创建一个Bucket。

   示例代码：

```
BCETask* task = [client putBucket:@"<bucketName>"];

// 任务可以异步执行。
task.then(^(BCEOutput* output) {
  // 可以在block内判断任务执行的结果。
  if (output.response) {
      // 任务执行成功。
  }

  if (output.error) {
      // 任务执行失败。
  }

  if (output.progress) {
      // 任务执行进度。
  }
});

// 可以同步方式，等待任务执行完毕。
[task waitUtilFinished];
```

3. 上传Object。

   Object是BOS中最基本的数据单元，您可以把Object简单的理解为文件。

   示例代码：

```
BOSObjectContent* content = [[BOSObjectContent alloc] init];
// 以下两行设置一种。
content.objectData.file = @"<FilePath>";
content.objectData.data = <NSData object>;

BOSPutObjectRequest* request = [[BOSPutObjectRequest alloc] init];
request.bucket = @"<bucketname>";
request.key = @"<ObjectName>";
request.objectContent = content;

__block BOSPutObjectResponse* response;
BCETask* task = [client putObject:request];
task.then(^(BCEOutput* output) {
    if (output.progress) {
        // 打印进度
        NSLog(@"put object progress is %@", output.progress);
    }

    if (output.response) {
        response = (BOSPutObjectResponse*)output.response;
        // 打印eTag
        NSLog(@"The eTag is %@", response.metadata.eTag);
    }

    if (output.error) {
        NSLog(@"put object failure with %@", output.error);
    }
});

[task waitUtilFinished];
```

4. 查看Bucket下的Object列表。

   当您完成一系列上传后，可以参考如下代码来查看Bucket下的全部Object。

   示例代码：

```
  // 获取指定Bucket下的所有Object信息
BOSListObjectsRequest* request = [[BOSListObjectsRequest alloc] init];
request.bucket = @"<bucketname>";

__block BOSListObjectsResponse* response = nil;
BCETask* task = [client listObjects:request];
task.then(^(BCEOutput* output) {
  if (output.response) {
      response = (BOSListObjectsResponse*)output.response;
      for (BOSObjectInfo* object in response.contents) {
          NSLog(@"The object key is %@", object.key);
      }
  }

  if (output.error) {
      NSLog(@"list objects failure with %@", output.error);
  }
});
```

5. 获取指定Object

   用户可以参考如下代码来实现对一个或者多个Object的获取。

   示例代码：

```
BOSGetObjectRequest* request = [[BOSGetObjectRequest alloc] init];
request.bucket = @"<bucketname>";
request.key = @"<Objectname>";
request.file = @"<filename>";

__block BOSGetObjectResponse* response = nil;
BCETask* task = [self.client getObject:request];
task.then(^(BCEOutput* output) {
    // ...
});
```

完整示例

```
 ##### import <BaiduBCEBasic/BaiduBCEBasic.h>
##### import <BaiduBCEBOS/BaiduBCEBOS.h>

void example(void) {
// 初始化带有STS验证的BOSClient
 BOSClient* client = [[BOSClient alloc] initWithConfiguration:configuration];
    BCESTSCredentials* credentials = [[BCESTSCredentials alloc] init];
     // access key和secret key分别代表AKSK。
    credentials.accessKey = @"<access key>"; // 临时返回的AK
    credentials.secretKey = @"<secret key>"; // 临时返回的SK
    credentials.sessionToken = @"<session token>"; // 临时返回的SessionToken

    BOSClientConfiguration* configuration = [[BOSClientConfiguration alloc] init];
    configuration.credentials = credentials;

    BOSClient* client = [[BOSClient alloc] initWithConfiguration:configuration];

// 1. 新建bucket
BCETask* task = [client putBucket:@"<bucketName>"];
task.then(^(BCEOutput* output) { // 任务可以异步执行。
    if (output.response) {
        // 任务执行成功。
    }

    if (output.error) {
        // 任务执行失败。
    }

    if (output.progress) {
        // 任务执行进度。
    }
});
[task waitUtilFinished]; // 可以同步方式，等待任务执行完毕。

// 2. 上传Object
BOSObjectContent* content = [[BOSObjectContent alloc] init];
content.objectData.file = @"<FilePath>";

BOSPutObjectRequest* request = [[BOSPutObjectRequest alloc] init];
request.bucket = @"<bucketname>";
request.key = @"<Objectname>";
request.objectContent = content;

__block BOSPutObjectResponse* response = nil;
task = [client putObject:request];
task.then(^(BCEOutput* output) {
    if (output.progress) {
        // 打印进度
        NSLog(@"put object progress is %@", output.progress);
    }

    if (output.response) {
        response = (BOSPutObjectResponse*)output.response;
        // 打印eTag
        NSLog(@"The eTag is %@", response.metadata.eTag);
    }

    if (output.error) {
        NSLog(@"put object failure with %@", output.error);
    }
});
[task waitUtilFinished];

// 3. 查看Object
BOSListObjectsRequest* listRequest = [[BOSListObjectsRequest alloc] init];
listRequest.bucket = @"<bucketname>";

__block BOSListObjectsResponse* listRepsponse = nil;
task = [client listObjects:listRequest];
task.then(^(BCEOutput* output) {
    if (output.response) {
        listRepsponse = (BOSListObjectsResponse*)output.response;
        for (BOSObjectInfo* object in listRepsponse.contents) {
            NSLog(@"The object key is %@", object.key);
        }
    }

    if (output.error) {
        NSLog(@"list objects failure with %@", output.error);
    }
});
[task waitUtilFinished];

// 4. 下载Object
BOSGetObjectRequest* getRequest = [[BOSGetObjectRequest alloc] init];
getRequest.bucket = @"<bucketname>";
getRequest.key = @"<Objectname>";
getRequest.file = @"<file>";

__block BOSGetObjectResponse* getResponse = nil;
task = [client getObject:getRequest];
task.then(^(BCEOutput* output) {
    if (output.response) {
        getResponse = (BOSGetObjectResponse*)output.response;
    }

    if (output.error) {
```

```
        NSLog(@"get object failure with %@", output.error);
    }
});
[task waitUtilFinished];

}
```

新建带有STS验证的BOSClient

BOSClient是BOS服务的客户端，为开发者与BOS服务进行交互提供了一系列的方法。在使用SDK发起对BOS的请求前，您需要初始化一个BOSClient实例，并对它进行一些必要设置。

> **说明：**iOS SDK主要用于移动端开发的场景，移动端是一个不受信任的环境，把AccessKey和SecretKey直接保存在终端用来加签请求，存在极高的风险。推荐您使用STS鉴权模式。STS的详细介绍请参考临时授权访问。

使用STS方式需要您在调用API时首先创建带有STS的BOSClient来执行，目前支持通过STS方式调用的BOS API接口请参见STS 服务接口。

- 示例代码：

```
BCESTSCredentials* credentials = [[BCESTSCredentials alloc] init];
credentials.accessKey = @"<access key>"; // 临时返回的AK
credentials.secretKey = @"<secret key>"; // 临时返回的SK
credentials.sessionToken = @"<session token>"; // 临时返回的SessionToken

BOSClientConfiguration* configuration = [[BOSClientConfiguration alloc] init];
configuration.region = [BCERegion region:BCERegionBJ];//服务区域，北京是BCERegionBJ，广州是BCERegionGZ，苏州是BCERegionSU，不指定默认为北京
configuration.credentials = credentials;

BOSClient* client = [[BOSClient alloc] initWithConfiguration:configuration];
```

> **注意：**
>
> - region参数只能定义为规定的枚举值，不指定时默认为北京区域。百度智能云目前开放了多区域支持，请参考区域选择说明。
> - 目前支持 BOS 官网开放的区域。具体域名参考BOS region

新建带有AKSK验证的BOSClient

新建AKSK鉴权方式的bosclient与上文提到的sts方式类似。

- 示例代码：

```
BCECredentials* credentials = [[BCECredentials alloc] init];
credentials.accessKey = @"AK";
credentials.secretKey = @"SK";

BOSClientConfiguration* configuration = [[BOSClientConfiguration alloc] init];
configuration.credentials = credentials;
configuration.endpoint = @"http://bj.bcebos.com";

BOSClient* client = [[BOSClient alloc] initWithConfiguration:configuration];
```

配置HTTPS协议访问BOS

BOS支持HTTPS传输协议，您可以通过如下方式在BOS iOS SDK中使用HTTPS访问BOS服务：

```
configuration.scheme = @"https";
```

配置BOSClient

如果您需要配置BOSClient的一些细节的参数，可以在构造BOSClient的时候传入BOSClientConfiguration对象。 BOSClientConfiguration是BOS服务的配置类，可以通过BOSClientConfiguration对基本网络参数进行设置。

设置网络参数

您可以使用BOSClientConfiguration对基本网络参数进行设置。

- 示例代码

```
BOSClientConfiguration config = [[BOSClientConfiguration alloc] init];

// 设置HTTP最大连接数为5
config.maxConurrentConnection = 5;

// 设置TCP连接超时为5秒
config.connectionTimeout = 5;

// 设置请求最长传输数据时间为60秒
config.totalTransferTimeout = 60;
```

- 参数说明

通过BOSClientConfiguration能指定的所有参数如下表所示：

参数	说明
authVersion	使用的鉴权算法版本，目前默认值 1
APIVersion	使用BOS API版本，目前默认值 1
scheme	使用的HTTP协议，可取http和https，默认值 http
region	使用BOS服务的区域
endpoint	如果知道明确的服务地址，可以不设置scheme和region等，直接设置endpoint
allowsCellularAccess	是否允许使用蜂窝网络，默认值 NO
connectionTimeout	设置TCP连接超时，默认值 60s
totalTransferTimeout	设置请求最长传输数据时间, 单位秒，默认值7天
maxConurrentConnection	允许打开的最大HTTP连接数。

配置访问域名风格

在v1.3.1版本之后，SDK会默认自动开启bucket virtual hosting。

> path-style: ${region}.bcebos.com/${bucket}/${object}
>
> virtual-hosted: {bucket}.${region}.bcebos.com/${object}

可以通过以下配置开启使用PathStyle风格的endpoint：

```
BOSClientConfiguration* configuration = [[BOSClientConfiguration alloc] init];
configuration.isPathStyleAccessEnable = YES;
```

注意：部分非BOS常规域名下，默认bucket virtual hosting风格的可能会有不兼容场景，表现为dns解析失败等，可以开启PathStyle解决。 其他场景建议使用默认风格。

Bucket管理

新建Bukcet

基本流程

1. 创建BOSClient类的实例。
2. 执行BOSClient putBucket方法，您需要提供Bucket的名字。

示例代码

```
BCETask* task = [client putBucket:@"<bucketname>"]; //新建一个Bucket，指定Bucket名称
```

完整示例

```
###### import <BaiduBCEBasic/BaiduBCEBasic.h>
###### import <BaiduBCEBOS/BaiduBCEBOS.h>

void example(void) {
// 初始化
BCECredentials* credentials = [[BCECredentials alloc] init];
credentials.accessKey = @"<access key>";
credentials.secretKey = @"<secret key>";
BOSClientConfiguration* configuration = [[BOSClientConfiguration alloc] init];
configuration.credentials = credentials;

BOSClient* client = [[BOSClient alloc] initWithConfiguration:configuration];

BCETask* task = [client putBucket:@"<bucketName>"];
task.then(^(BCEOutput* output) { // 任务可以异步执行。
    if (output.response) {
        // 任务执行成功。
    }

    if (output.error) {
        // 任务执行失败。
    }

    if (output.progress) {
        // 任务执行进度。
    }
});
[task waitUtilFinished]; // 可以同步方式，等待任务执行完毕。
}
```

> **注意：**由于Bucket的名称必须在所有区域中是唯一的，所以需要保证BucketName不与其他所有区域上的BucketName相同。

查看Bucket列表

基本流程

1. 创建BOSClient类的实例。
2. 执行BOSClient listBuckets方法，会返回BOSListBucketResponse类的实例。
3. 对BOSListBucketResponse类型实例可以进行获取buckets/owner操作。

示例代码

如下代码可以列出用户所有的Bucket：

```
__block BOSListBucketResponse* response = nil;
BCETask* task = [client listBuckets];
task.then(^(BCEOutput* output) {
    if (output.response) {
        response = (BOSListBucketResponse*)output.response;
    }

    if (output.error) {
    }
});
[task waitUtilFinished];
```

如下代码可以列出Bucket的Owner：

```
BOSBucketOwner* owner = response.owner;
```

如下代码可以列出Bucket的Metadata：

```
NSArray<BOSBucketSummary*>* buckets = response.buckets;
for (BOSBucketSummary* bucket in buckets) {
    NSLog(@"bucket name: %@", bucket.name);
    NSLog(@"bucket location: %@", bucket.location);
    NSLog(@"bucket create date: %@", bucket.createDate);
}
```

完整示例

```
###### import <BaiduBCEBasic/BaiduBCEBasic.h>
###### import <BaiduBCEBOS/BaiduBCEBOS.h>

void example(void) {
    // 初始化
    BCECredentials* credentials = [[BCECredentials alloc] init];
    credentials.accessKey = @"<access key>";
    credentials.secretKey = @"<secret key>";
    BOSClientConfiguration* configuration = [[BOSClientConfiguration alloc] init];
    configuration.credentials = credentials;

    BOSClient* client = [[BOSClient alloc] initWithConfiguration:configuration];

    // 列举Buckets
    __block BOSListBucketResponse* response = nil;
    BCETask* task = [client listBuckets];
    task.then(^(BCEOutput* output) {
        if (output.response) {
            response = (BOSListBucketResponse*)output.response;
        }

        if (output.error) {
        }
    });
    [task waitUtilFinished];

    // 获取Owner
    BOSBucketOwner* owner = response.owner;
    NSLog(@"the buckets owner is %@", owner.ownerID);

    // 获取Bucket信息
    NSArray<BOSBucketSummary*>* buckets = response.buckets;
    for (BOSBucketSummary* bucket in buckets) {
        NSLog(@"bucket name: %@", bucket.name);
        NSLog(@"bucket location: %@", bucket.location);
        NSLog(@"bucket create date: %@", bucket.createDate);
    }
}
```

判断Bucket是否存在，以及是否有权限访问

基本流程

1. 创建BOSClient类的实例；

2. 执行BOSClient headBucket方法；

3. 当请求无错误发生时，说明Bucket存在且请求者有权限访问。

示例代码

```
__block BOSHeadBucketResponse* response = nil;
BCETask* task = [client headBucket:@"<bucketname>"];
task.then(^(BCEOutput* output) {
    if (output.response) {
        response = (BOSHeadBucketResponse*)output.response;
        NSLog(@"head bucket success!");
    }

    if (output.error) {
        NSLog(@"Bucket not exist or no permission!");
    }
});
[task waitUtilFinished];
```

完整示例

```
 ###### import <BaiduBCEBasic/BaiduBCEBasic.h>
###### import <BaiduBCEBOS/BaiduBCEBOS.h>

void example(void) {
// 初始化
BCECredentials* credentials = [[BCECredentials alloc] init];
credentials.accessKey = @"<access key>";
credentials.secretKey = @"<secret key>";
BOSClientConfiguration* configuration = [[BOSClientConfiguration alloc] init];
configuration.credentials = credentials;

BOSClient* client = [[BOSClient alloc] initWithConfiguration:configuration];

__block BOSHeadBucketResponse* response = nil;
BCETask* task = [client headBucket:@"<bucketname>"];
task.then(^(BCEOutput* output) {
    if (output.response) {
        response = (BOSHeadBucketResponse*)output.response;
        NSLog(@"head bucket success!");
    }

    if (output.error) {
        NSLog(@"Bucket not exist or no permission!");
    }
});
[task waitUtilFinished];
}
```

删除Bucket

基本流程

1. 创建BOSClient类的实例；

2. 执行BOSClient deleteBucket方法；

3. 删除失败时会产生错误。

示例代码

```
 __block BOSDeleteBucketResponse* response = nil;
BCETask* task = [client deleteBucket:@"<bucketname>"];
task.then(^(BCEOutput* output) {
  if (output.response) {
      response = (BOSDeleteBucketResponse*)output.response;
      NSLog(@"delete bucket success!");
  }

  if (output.error) {
      NSLog(@"delete bucket failure");
  }
});
[task waitUtilFinished];
```

> **注意：**如果Bucket不为空（即Bucket中有Object和未完成的三步上传Part存在），则Bucket无法被删除，必须清空Bucket后才能成功删除。

完整示例

```
 ###### import <BaiduBCEBasic/BaiduBCEBasic.h>
###### import <BaiduBCEBOS/BaiduBCEBOS.h>

void example(void) {
// 初始化
BCECredentials* credentials = [[BCECredentials alloc] init];
credentials.accessKey = @"<access key>";
credentials.secretKey = @"<secret key>";
BOSClientConfiguration* configuration = [[BOSClientConfiguration alloc] init];
configuration.credentials = credentials;

BOSClient* client = [[BOSClient alloc] initWithConfiguration:configuration];

__block BOSDeleteBucketResponse* response = nil;
BCETask* task = [client deleteBucket:@"<bucketname>"];
task.then(^(BCEOutput* output) {
    if (output.response) {
        response = (BOSDeleteBucketResponse*)output.response;
        NSLog(@"delete bucket success!");
    }

    if (output.error) {
        NSLog(@"delete bucket failure");
    }
});
[task waitUtilFinished];
}
```

Bucket权限控制

设置Bucket的访问权限

基本流程

1. 创建BOSClient类的实例；

2. 执行BOSClient putBucketACL方法；

3. 设置失败时会产生错误。

示例代码

```
 BOSPutBucketAclRequest* request = [[BOSPutBucketAclRequest alloc] init];
request.cannedAcl = BOS_ACL_PUBLIC_READ;
request.bucket = @"<bucketname>";

__block BOSPutBucketAclResponse* response = nil;
BCETask* task = [client putBucketACL:request];
task.then(^(BCEOutput* output) {
  if (output.response) {
      response = (BOSPutBucketAclResponse*)output.response;
      NSLog(@"pub bucket acl success!");
  }

  if (output.error) {
      NSLog(@"pub bucket acl failure with %@", output.error);
  }
});
[task waitUtilFinished];
```

> **说明：**cannedAcl 字段可取3个值：`Private`、`PublicRead`和`PublicReadWrite`，它们分别对应相关权限，具体内容可以参考《BOS API文档 使用CannedAcl方式的权限控制》。

完整示例

```
 ###### import <BaiduBCEBasic/BaiduBCEBasic.h>
###### import <BaiduBCEBOS/BaiduBCEBOS.h>

void example(void) {
// 初始化
BCECredentials* credentials = [[BCECredentials alloc] init];
credentials.accessKey = @"<access key>";
credentials.secretKey = @"<secret key>";
BOSClientConfiguration* configuration = [[BOSClientConfiguration alloc] init];
configuration.credentials = credentials;

BOSClient* client = [[BOSClient alloc] initWithConfiguration:configuration];

BOSPutBucketAclRequest* request = [[BOSPutBucketAclRequest alloc] init];
request.cannedAcl = BOS_ACL_PUBLIC_READ;
request.bucket = @"<bucketname>";

__block BOSPutBucketAclResponse* response = nil;
BCETask* task = [client putBucketACL:request];
task.then(^(BCEOutput* output) {
    if (output.response) {
        response = (BOSPutBucketAclResponse*)output.response;
        NSLog(@"pub bucket acl success!");
    }

    if (output.error) {
        NSLog(@"pub bucket acl failure with %@", output.error);
    }
});
[task waitUtilFinished];
}
```

设置指定用户对Bucket的访问权限

基本流程

1. 创建BOSClient类的实例。

2. 执行putBucketACL方法，您需要创建一个BOSPutBucketAclRequest的实例来提供授权用户信息。

3. 设置失败时会抛出异常。

示例代码

```
 ###### import <BaiduBCEBasic/BaiduBCEBasic.h>
###### import <BaiduBCEBOS/BaiduBCEBOS.h>

void example(void) {
    // 初始化
    BCECredentials* credentials = [[BCECredentials alloc] init];
    credentials.accessKey = @"<access key>";
        credentials.secretKey = @"<secret key>";
    BOSClientConfiguration* configuration = [[BOSClientConfiguration alloc] init];
    configuration.credentials = credentials;

    BOSClient* client = [[BOSClient alloc] initWithConfiguration:configuration];
    NSArray<NSString*>* grantee = @[
                                @"<grantee1>",
                                @"<grantee2>"
                                ];

    NSArray<NSString*>* permission = @[
                                    [BOSGrant permissionToString:BOSBucketGranteePermissionRead],
                                    [BOSGrant permissionToString:BOSBucketGranteePermissionList],
                                    ];

    BOSGrant* grant = [[BOSGrant alloc] init];
    grant.granteeIDArray = grantee;
    grant.permission = permission;
    BOSPutBucketAclRequest* request = [[BOSPutBucketAclRequest alloc] init];
    request.acl = [[BOSACL alloc] init];
    request.acl.grantees = @[grant];
    request.bucket = @"<bucketname>";

    __block BOSPutBucketAclResponse* response = nil;
    BCETask* task = [client putBucketACL:request];
    task.then(^(BCEOutput* output) {
        if (output.response) {
            response = (BOSPutBucketAclResponse*)output.response;
            NSLog(@"pub bucket acl success!");
        }

        if (output.error) {
            NSLog(@"pub bucket acl failure with %@", output.error);
        }
    });
    [task waitUtilFinished];
}
```

注意：Permission中的权限设置包含三个值：`READ`、`WRITE`、`LIST`、`GetObject`和`FULL_CONTROL`，它们分别对应相关权限，具体内容可以参考《BOS API文档 上传ACL文件方式的权限控制》。

完整示例

```
 ###### import <BaiduBCEBasic/BaiduBCEBasic.h>
###### import <BaiduBCEBOS/BaiduBCEBOS.h>

void example(void) {
// 初始化
BCECredentials* credentials = [[BCECredentials alloc] init];
credentials.accessKey = @"<access key>";
credentials.secretKey = @"<secret key>";
BOSClientConfiguration* configuration = [[BOSClientConfiguration alloc] init];
configuration.credentials = credentials;

BOSClient* client = [[BOSClient alloc] initWithConfiguration:configuration];
    NSArray<NSString*>* grantee = @[
    @"<grantee1>",
    @"<grantee2>"
];

NSArray<NSString*>* permission = @[
    [BOSGrant permissionToString:BOSBucketGranteePermissionRead],
    [BOSGrant permissionToString:BOSBucketGranteePermissionList],
];

BOSGrant* grant = [[BOSGrant alloc] init];
grant.granteeIDArray = grantee;
grant.permission = permission;
BOSPutBucketAclRequest* request = [[BOSPutBucketAclRequest alloc] init];
request.acl = @[grant];
request.bucket = @"<bucketname>";

__block BOSPutBucketAclResponse* response = nil;
BCETask* task = [client putBucketACL:request];
task.then(^(BCEOutput* output) {
    if (output.response) {
        response = (BOSPutBucketAclResponse*)output.response;
        NSLog(@"pub bucket acl success!");
    }

    if (output.error) {
        NSLog(@"pub bucket acl failure with %@", output.error);
    }
});
[task waitUtilFinished];
}
```

文件管理

上传Object

最简单的上传

基本流程

1. 创建BOSClient类的实例。

2. 调用BOSClient putObject方法，可以通过如下二种方式上传Object：文件、二进制数据的形式。

3. 对返回的BOSPutObjectResponse类型实例，可以执行获取eTag操作。

示例代码

```
 BOSObjectContent* content = [[BOSObjectContent alloc] init];
// 以文件方式
content.objectData.file = @"<file path>";

// 或者以二进制数据方式
NSData* data = [[NSData alloc] init];
content.objectData.data = data;

BOSPutObjectRequest* request = [[BOSPutObjectRequest alloc] init];
request.bucket = @"<bucketname>";
request.key = @"<objectname>";
request.objectContent = content;

__block BOSPutObjectResponse* response = nil;
BCETask* task = [client putObject:request];
task.then(^(BCEOutput* output) {
  if (output.progress) {
     NSLog(@"put object progress is %@", output.progress);
  }

  if (output.response) {
     response = (BOSPutObjectResponse*)output.response;
     NSLog(@"put object success!");
  }

  if (output.error) {
     NSLog(@"put object failure");
  }
});
[task waitUtilFinished];
```

> **说明**：Object以文件的形式上传到BOS中，putObject函数支持不超过5GB的Object上传。在putObject请求处理成功后，BOS会在Header中返回Object的ETag作为文件标识。

完整示例

```
 ###### import <BaiduBCEBasic/BaiduBCEBasic.h>
###### import <BaiduBCEBOS/BaiduBCEBOS.h>

void example(void) {
    // 初始化
    BCECredentials* credentials = [[BCECredentials alloc] init];
    credentials.accessKey = @"<access key>";
    credentials.secretKey = @"<secret key>";
    BOSClientConfiguration* configuration = [[BOSClientConfiguration alloc] init];
    configuration.credentials = credentials;

    BOSClient* client = [[BOSClient alloc] initWithConfiguration:configuration];

    BOSObjectContent* content = [[BOSObjectContent alloc] init];
    // 以文件方式
    content.objectData.file = @"<file path>";

    // 或者以二进制数据方式
    NSData* data = [[NSData alloc] init];
    content.objectData.data = data;

    BOSPutObjectRequest* request = [[BOSPutObjectRequest alloc] init];
    request.bucket = @"<bucketname>";
    request.key = @"<objectname>";
    request.objectContent = content;

    __block BOSPutObjectResponse* response = nil;
    BCETask* task = [client putObject:request];
    task.then(^(BCEOutput* output) {
        if (output.progress) {
            NSLog(@"put object progress is %@", output.progress);
        }

        if (output.response) {
            response = (BOSPutObjectResponse*)output.response;
            NSLog(@"put object success!");
        }

        if (output.error) {
            NSLog(@"put object failure");
        }
    });
    [task waitUtilFinished];
}
```

设定Object的Http Header

BOS支持您在上传object时设定Http Header。

基本流程

1. 创建BOSObjectMetadata类的实例。

2. 设定BOSObjectMetadata实例的contentEncoding/contentType/contentDisposition等字段。

3. 将BOSObjectMetadata实例设置到BOSPutObjectRequest的objectContent.metadata字段上。

示例代码

```
 BOSObjectMetadata* metadata = [[BOSObjectMetadata alloc] init];
metadata.contentEncoding = @"<encoding>";
metadata.contentDisposition = @"<content disposition>";
content.metadata = metadata;
```

完整示例

```
 ###### import <BaiduBCEBasic/BaiduBCEBasic.h>
###### import <BaiduBCEBOS/BaiduBCEBOS.h>

void example(void) {
    // 初始化
    BCECredentials* credentials = [[BCECredentials alloc] init];
    credentials.accessKey = @"<access key>";
    credentials.secretKey = @"<secret key>";
    BOSClientConfiguration* configuration = [[BOSClientConfiguration alloc] init];
    configuration.credentials = credentials;

    BOSClient* client = [[BOSClient alloc] initWithConfiguration:configuration];

    BOSObjectContent* content = [[BOSObjectContent alloc] init];
    // 以文件方式
    content.objectData.file = @"<file path>";

    // 或者以二进制数据方式
    NSData* data = [[NSData alloc] init];
    content.objectData.data = data;

    BOSObjectMetadata* metadata = [[BOSObjectMetadata alloc] init];
    metadata.contentEncoding = @"<encoding>";
    metadata.contentDisposition = @"<content disposition>";
    content.metadata = metadata;

    BOSPutObjectRequest* request = [[BOSPutObjectRequest alloc] init];
    request.bucket = @"<bucketname>";
    request.key = @"<Objectname>";
    request.objectContent = content;

    __block BOSPutObjectResponse* response = nil;
    BCETask* task = [client putObject:request];
    task.then(^(BCEOutput* output) {
        if (output.progress) {
            NSLog(@"put object progress is %@", output.progress);
        }

        if (output.response) {
            response = (BOSPutObjectResponse*)output.response;
            NSLog(@"put object success!");
        }

        if (output.error) {
            NSLog(@"put object failure");
        }
    });
    [task waitUtilFinished];
}
```

用户自定义元数据

BOS支持用户自定义元数据来对Object进行描述。

基本流程

1. 创建BOSObjectMetadata类的实例。

2. 将自定义元数据字典设置到BOSObjectMetadata的userMetadata字段。

示例代码

```
 BOSObjectMetadata* metadata = [[BOSObjectMetadata alloc] init];
metadata.contentEncoding = @"<encoding>";
metadata.contentDisposition = @"<content disposition>";
content.metadata = metadata;

NSDictionary* customMetadata = @{
  @"name" : @"my-data"
};
content.metadata.userMetadata = customMetadata;
```

> **说明：**在上面代码中，用户自定义了一个名字为"name"，值为"my-data"的元数据。当用户下载此Object的时候，此元数据也可以一并得到。一个Object可以有多个类似的参数，但所有的User Meta总大小不能超过2KB。

完整示例

```
###### import <BaiduBCEBasic/BaiduBCEBasic.h>
###### import <BaiduBCEBOS/BaiduBCEBOS.h>

void example(void) {
    // 初始化
    BCECredentials* credentials = [[BCECredentials alloc] init];
    credentials.accessKey = @"<access key>";
    credentials.secretKey = @"<secret key>";
    BOSClientConfiguration* configuration = [[BOSClientConfiguration alloc] init];
    configuration.credentials = credentials;

    BOSClient* client = [[BOSClient alloc] initWithConfiguration:configuration];

    BOSObjectContent* content = [[BOSObjectContent alloc] init];
    // 以文件方式
    content.objectData.file = @"<file path>";

    // 或者以二进制数据方式
    NSData* data = [[NSData alloc] init];
    content.objectData.data = data;

    BOSObjectMetadata* metadata = [[BOSObjectMetadata alloc] init];
    metadata.contentEncoding = @"<encoding>";
    metadata.contentDisposition = @"<content disposition>";
    content.metadata = metadata;

    NSDictionary* customMetadata = @{
        @"name" : @"my-data"
    };
    content.metadata.userMetadata = customMetadata;

    BOSPutObjectRequest* request = [[BOSPutObjectRequest alloc] init];
    request.bucket = @"<bucketname>";
    request.key = @"<objectname>";
    request.objectContent = content;

    __block BOSPutObjectResponse* response = nil;
    BCETask* task = [client putObject:request];
    task.then(^(BCEOutput* output) {
        if (output.progress) {
            NSLog(@"put object progress is %@", output.progress);
        }

        if (output.response) {
            response = (BOSPutObjectResponse*)output.response;
            NSLog(@"put object success!");
        }

        if (output.error) {
            NSLog(@"put object failure");
        }
    });
    [task waitUtilFinished];
}
```

查看Bucket中的Object

简单查询

查看Bucket中Object列表。

基本流程

1. 创建BOSClient类的实例。

2. 执行BOSClient listObjects方法，会返回BOSListObjectsResponse类的实例。

3. 可以对BOSListObjectsResponse类型的contents字段进行枚举，获取key/lastModified/eTag/size/owner。

示例代码

```
BOSListObjectsRequest* listObjRequest = [[BOSListObjectsRequest alloc] init];
listObjRequest.bucket = @"<bucketname>";

__block BOSListObjectsResponse* listObjResponse = nil;
BCETask* task = [client listObjects:listObjRequest];
task.then(^(BCEOutput* output) {
    if (output.response) {
        listObjResponse = (BOSListObjectsResponse*)output.response;

        for (BOSObjectInfo* object in listObjResponse.contents) {
            NSLog(@"the object key is %@", object.key);
            NSLog(@"the object lastModified is %@", object.lastModified);
            NSLog(@"the object eTag is %@", object.eTag);
            NSLog(@"the object size is %llu", object.size);
            NSLog(@"the object owner id is %@", object.owner.ownerID);
        }
    }

    if (output.error) {
        NSLog(@"list objects failure");
    }
});
[task waitUtilFinished];
```

> **说明：** listObjects方法返回BOSListObjectsResponse对象，BOSListObjectsResponse对象包含了此次listObject请求的返回结果。用户可以通过BOSListObjectsResponse中的contents属性获取所有Object的描述信息。

- 默认情况下，如果Bucket中的Object数量大于1000，则只会返回1000个Object，并且返回结果中isTruncated值为YES，并返回nextMarker做为下次读取的起点。
- 若想获取更多的Object，可以使用marker参数分次读取，请参考扩展查询。

完整示例

```
 ###### import <BaiduBCEBasic/BaiduBCEBasic.h>
###### import <BaiduBCEBOS/BaiduBCEBOS.h>

void example(void) {
    // 初始化
    BCECredentials* credentials = [[BCECredentials alloc] init];
    credentials.accessKey = @"<access key>";
    credentials.secretKey = @"<secret key>";
    BOSClientConfiguration* configuration = [[BOSClientConfiguration alloc] init];
    configuration.credentials = credentials;

    BOSClient* client = [[BOSClient alloc] initWithConfiguration:configuration];

    BOSListObjectsRequest* listObjRequest = [[BOSListObjectsRequest alloc] init];
    listObjRequest.bucket = @"<bucketname>";

    __block BOSListObjectsResponse* listObjResponse = nil;
    BCETask* task = [client listObjects:listObjRequest];
    task.then(^(BCEOutput* output) {
        if (output.response) {
            listObjResponse = (BOSListObjectsResponse*)output.response;

            for (BOSObjectInfo* object in listObjResponse.contents) {
                NSLog(@"the object key is %@", object.key);
                NSLog(@"the object lastModified is %@", object.lastModified);
                NSLog(@"the object eTag is %@", object.eTag);
                NSLog(@"the object size is %llu", object.size);
                NSLog(@"the object owner id is %@", object.owner.ownerID);
            }
        }

        if (output.error) {
            NSLog(@"list objects failure");
        }
    });
    [task waitUtilFinished];
}
```

扩展查询

用户可以通过设置BOSListObjectsResquest参数来完成更多扩展查询操作设置。BOSListObjectsResquest中可以设置的扩展参数如下：

参数名称	说明	默认值
maxKeys	设定此次返回Object的最大个数，不可超过1000。	1000
prefix	设定objectKey的前缀，前缀是指objectKey包含并以prefix的值作为开始。通常与delimiter配合在查询模拟文件夹中使用。	-
delimiter	是一个分隔符，用来对objectKey进行分层。通常与prefix配合在查询模拟文件夹中使用。从prefix开始到第一次出现delimiter字符之间的objectKey称为：commonPrefixes。	-
marker	是一个字符串，用来设定返回结果的起始位置。设定marker值之后，返回的Object会从marker值之后按字母排序开始返回。	-

基本流程

1. 创建BOSListObjectsResquest类的实例。

2. 设置BOSListObjectsResquest的delimiter/marker/prefix/maxKeys等字段，实现更多的扩展查询操作。

3. 创建BOSClient类的实例，执行listObjects。

示例代码

```
 BOSListObjectsRequest* listObjRequest = [[BOSListObjectsRequest alloc] init];
listObjRequest.bucket = @"<bucketname>";
listObjRequest.marker = @"<marker>";

__block BOSListObjectsResponse* listObjResponse = nil;
BCETask* task = [client listObjects:listObjRequest];
task.then(^(BCEOutput* output) {
  if (output.response) {
      listObjResponse = (BOSListObjectsResponse*)output.response;

      for (BOSObjectInfo* object in listObjResponse.contents) {
          NSLog(@"the object key is %@", object.key);
          NSLog(@"the object lastModified is %@", object.lastModified);
          NSLog(@"the object eTag is %@", object.eTag);
          NSLog(@"the object size is %llu", object.size);
          NSLog(@"the object owner id is %@", object.owner.ownerID);
      }
  }

  if (output.error) {
      NSLog(@"list objects failure");
  }
});
[task waitUtilFinished];
```

完整示例

示例一：

```
 ###### import <BaiduBCEBasic/BaiduBCEBasic.h>
###### import <BaiduBCEBOS/BaiduBCEBOS.h>

void example(void) {
    // 初始化
    BCECredentials* credentials = [[BCECredentials alloc] init];
    credentials.accessKey = @"<access key>";
    credentials.secretKey = @"<secret key>";
    BOSClientConfiguration* configuration = [[BOSClientConfiguration alloc] init];
    configuration.credentials = credentials;

    BOSClient* client = [[BOSClient alloc] initWithConfiguration:configuration];

    BOSListObjectsRequest* listObjRequest = [[BOSListObjectsRequest alloc] init];
    listObjRequest.bucket = @"<bucketname>";
    listObjRequest.marker = @"<marker>";

    __block BOSListObjectsResponse* listObjResponse = nil;
    BCETask* task = [client listObjects:listObjRequest];
    task.then(^(BCEOutput* output) {
        if (output.response) {
            listObjResponse = (BOSListObjectsResponse*)output.response;

            for (BOSObjectInfo* object in listObjResponse.contents) {
                NSLog(@"the object key is %@", object.key);
                NSLog(@"the object lastModified is %@", object.lastModified);
                NSLog(@"the object eTag is %@", object.eTag);
                NSLog(@"the object size is %llu", object.size);
                NSLog(@"the object owner id is %@", object.owner.ownerID);
            }
        }

        if (output.error) {
            NSLog(@"list objects failure");
        }
    });
    [task waitUtilFinished];
}
```

示例二：使用nextMarker的完整示例。

```
 ###### import <BaiduBCEBasic/BaiduBCEBasic.h>
###### import <BaiduBCEBOS/BaiduBCEBOS.h>

void example(void) {
    // 初始化
    BCECredentials* credentials = [[BCECredentials alloc] init];
    credentials.accessKey = @"<access key>";
    credentials.secretKey = @"<secret key>";
    BOSClientConfiguration* configuration = [[BOSClientConfiguration alloc] init];
    configuration.credentials = credentials;

    BOSClient* client = [[BOSClient alloc] initWithConfiguration:configuration];

    BOSListObjectsRequest* listObjRequest = [[BOSListObjectsRequest alloc] init];
    listObjRequest.bucket = @"<bucketname>";
    listObjRequest.marker = @"<marker>";

    __block BOSListObjectsResponse* listObjResponse = nil;
    BCETask* task = [client listObjects:listObjRequest];
    task.then(^(BCEOutput* output) {
        if (output.response) {
            listObjResponse = (BOSListObjectsResponse*)output.response;
        }

        if (output.error) {
            NSLog(@"list objects failure");
        }
    });
    [task waitUtilFinished];

    if (listObjResponse != nil) {
        listObjRequest.marker = listObjResponse.nextMarker;
        task = [client listObjects:listObjRequest];
        task.then(^(BCEOutput* output) {
            if (output.response) {
                listObjResponse = (BOSListObjectsResponse*)output.response;

                for (BOSObjectInfo* object in listObjResponse.contents) {
                    NSLog(@"the object key is %@", object.key);
                    NSLog(@"the object lastModified is %@", object.lastModified);
                    NSLog(@"the object eTag is %@", object.eTag);
                    NSLog(@"the object size is %llu", object.size);
                    NSLog(@"the object owner id is %@", object.owner.ownerID);
                }
            }
        });
    }
}
```

查询模拟文件夹

由于BOS本身是一个(<Key>,<Value>)的存储系统，所以原则上并不会存在“文件夹”的概念，但您可以通过delimiter和prefix参数的配合进行文件夹功能模拟。

假设Bucket中有5个文件：bos.jpg，fun/，fun/test.jpg，fun/movie/001.avi，fun/movie/007.avi，可以把 "/" 符号作为分隔符模拟文件夹。

递归列出模拟文件夹下所有文件

可以通过设置prefix参数来获取某个模拟文件夹下所有的文件：

```
 BOSListObjectsRequest* listObjRequest = [[BOSListObjectsRequest alloc] init];
listObjRequest.bucket = @"<bucketname>";
 listObjRequest.prefix = @"fun/";

__block BOSListObjectsResponse* listObjResponse = nil;
BCETask* task = [client listObjects:listObjRequest];
task.then(^(BCEOutput* output) {
    if (output.response) {
        listObjResponse = (BOSListObjectsResponse*)output.response;

        for (BOSObjectInfo* object in listObjResponse.contents) {
            NSLog(@"%@", object.key);
        }
    }
});
[task waitUtilFinished];
```

输出：

```
 Objects:
fun/
fun/movie/001.avi
fun/movie/007.avi
fun/test.jpg
```

查看模拟文件夹下的文件和子文件夹

在prefix和delimiter结合的情况下，可以列出模拟文件夹下的文件和子文件夹：

```
 BOSListObjectsRequest* listObjRequest = [[BOSListObjectsRequest alloc] init];
listObjRequest.bucket = @"<bucketname>";

// 指定"/"为模拟文件夹的分隔符
listObjRequest.delimiter = @"/";

// 列出fun文件夹下的所有文件和子文件夹
listObjRequest.prefix = @"fun/";

__block BOSListObjectsResponse* listObjResponse = nil;
BCETask* task = [client listObjects:listObjRequest];
task.then(^(BCEOutput* output) {
    if (output.response) {
        listObjResponse = (BOSListObjectsResponse*)output.response;
    }

    if (output.error) {
        NSLog(@"list objects failure");
    }
});
[task waitUtilFinished];

// 遍历所有Object
NSLog(@"Objects:");
for (BOSObjectInfo* object in listObjResponse.contents) {
    NSLog(@"%@", object.key);
}

// 遍历所有CommonPrefix
NSLog(@"CommonPrefixs:");
for (NSString* commonPrefix in listObjResponse.commonPrefixes) {
    NSLog(@"%@", commonPrefix);
}
```

输出：

```
 Objects:
fun/
fun/test.jpg

CommonPrefixs:
fun/movie/
```

> **说明：** 返回的结果中，contents的列表中给出的是fun文件夹下的文件。而commonPrefixs的列表中给出的是fun文件夹下的所有子文件夹。可以看出fun/movie/001.avi，fun/movie/007.avi两个文件并没有被列出来，因为它们属于fun文件夹下的movie子文件夹下的文件。

获取Object

简单的获取Object

用户可以通过如下代码将Object读取到内存中。

基本流程

1. 创建BOSClient类的实例。
2. 执行BOSClient getObject方法，会返回BOSGetObjectResponse实例。
3. 访问BOSGetObjectResponse示例的objectContent.objectData.data属性，获取数据。

示例代码

```
 __block BOSGetObjectResponse* getObjResponse = nil;
BOSGetObjectRequest* getObjRequest = [[BOSGetObjectRequest alloc] init];
getObjRequest.bucket = @"<bucketname>";
getObjRequest.key = @"<objectname>";
BCETask* task = [client getObject:getObjRequest];
task.then(^(BCEOutput* output) {
  if (output.response) {
     getObjResponse = (BOSGetObjectResponse*)output.response;
     NSLog(@"get object success!");
  }

  if (output.error) {
     NSLog(@"get object failure with %@", outpu****t.error);
  }

  if (output.progress) {
     NSLog(@"the get object progress is %@", output.progress);
  }
});
[task waitUtilFinished];

// 获取内存中的数据
NSData* data = getObjResponse.objectContent.objectData.data;
```

注意：

- BOSObjectContent中包含了Object的各种信息，包含Object所在的Bucket、Object的名称、MetaData以及数据存储。
- BOSObjectMetadata中包含了Object上传时定义的ETag，Http Header以及自定义的元数据。
- 通过BOSObjectContent的objectData属性，获取到Object的数据。

完整示例

```
 ###### import <BaiduBCEBasic/BaiduBCEBasic.h>
###### import <BaiduBCEBOS/BaiduBCEBOS.h>

void example(void) {
    // 初始化
    BCECredentials* credentials = [[BCECredentials alloc] init];
    credentials.accessKey = @"<access key>";
    credentials.secretKey = @"<secret key>";
    BOSClientConfiguration* configuration = [[BOSClientConfiguration alloc] init];
    configuration.credentials = credentials;

    BOSClient* client = [[BOSClient alloc] initWithConfiguration:configuration];

    __block BOSGetObjectResponse* getObjResponse = nil;
    BOSGetObjectRequest* getObjRequest = [[BOSGetObjectRequest alloc] init];
    getObjRequest.bucket = @"<bucketname>";
    getObjRequest.key = @"<objectname>";
    BCETask* task = [client getObject:getObjRequest];
    task.then(^(BCEOutput* output) {
        if (output.response) {
            getObjResponse = (BOSGetObjectResponse*)output.response;
            NSLog(@"get object success!");
        }

        if (output.error) {
            NSLog(@"get object failure with %@", output.error);
        }

        if (output.progress) {
            NSLog(@"the get object progress is %@", output.progress);
        }
    });
    [task waitUtilFinished];

    NSData* data = getObjResponse.objectContent.objectData.data;
}
```

下载Object的一部分内容

基本流程

1. 创建BOSGetObjectRequest类的实例。

2. BOSGetObjectRequestt实例的rangeStart和/或rangeEnd字段。

3. 执行client getObject操作。

示例代码

```
__block BOSGetObjectResponse* getObjResponse = nil;
BOSGetObjectRequest* getObjRequest = [[BOSGetObjectRequest alloc] init];
getObjRequest.bucket = @"<bucketname>";
getObjRequest.key = @"<objectname>";

// 获取前100个字节
getObjRequest.rangeStart = @"0";
getObjRequest.rangeEnd = @"99";

BCETask* task = [client getObject:getObjRequest];
task.then(^(BCEOutput* output) {
    if (output.response) {
        getObjResponse = (BOSGetObjectResponse*)output.response;
        NSLog(@"get object success!");
    }

    if (output.error) {
        NSLog(@"get object failure with %@", output.error);
    }

    if (output.progress) {
        NSLog(@"the get object progress is %@", output.progress);
    }
});
[task waitUtilFinished];

NSData* data = getObjResponse.objectContent.objectData.data;
```

> **说明：**用户可以用此功能实现文件的分段下载和断点续传。

下载Object到指定路径

用户可以通过如下代码直接将Object下载到指定路径。

基本流程

1. 创建BOSGetObjectRequest类的实例。
2. 设置要保存的文件名到BOSGetObjectRequest实例的file字段。
3. 执行client getObject操作。
4. Object可以直接下载到指定路径。

示例代码

```
__block BOSGetObjectResponse* getObjResponse = nil;
BOSGetObjectRequest* getObjRequest = [[BOSGetObjectRequest alloc] init];
getObjRequest.bucket = @"<bucketname>";
getObjRequest.key = @"<objectname>";

// 设置保存到的文件路径
getObjRequest.file = @"<file>";

BCETask* task = [client getObject:getObjRequest];
task.then(^(BCEOutput* output) {
    if (output.response) {
        getObjResponse = (BOSGetObjectResponse*)output.response;
        NSLog(@"get object success!");
    }

    if (output.error) {
        NSLog(@"get object failure with %@", output.error);
    }

    if (output.progress) {
        NSLog(@"the get object progress is %@", output.progress);
    }
});
[task waitUtilFinished];
```

获取Object的storageClass

Object的storage class属性分为STANDARD(标准存储), STANDARD_IA(低频存储)和COLD(冷存储)。

示例代码：

```
__block BOSGetObjectMetadataResponse* getObjMetaResponse = nil;
BCETask* task = [client getObjectMetadata:@"<bucketname>" objectKey:@"<objectname>"];
task.then(^(BCEOutput* output) {
    if (output.response) {
        getObjMetaResponse = (BOSGetObjectMetadataResponse*)output.response;
        NSString storageClass = getObjMetaResponse.storageClass;
        NSLog(@"get object storageClass success!");
    }

    if (output.error) {
        NSLog(@"get object storageClass failure");
    }
});
[task waitUtilFinished];
```

只获取ObjectMetadata

通过getObjectMetadata 方法可以只获取 Object metadata 而不获取Object的实体。

示例代码：

```
 __block BOSGetObjectMetadataResponse* getObjMetaResponse = nil;
BCETask* task = [client getObjectMetadata:@"<bucketname>" objectKey:@"<objectname>"];
task.then(^(BCEOutput* output) {
    if (output.response) {
        getObjMetaResponse = (BOSGetObjectMetadataResponse*)output.response;
        NSLog(@"get object metadata success!");
    }

    if (output.error) {
        NSLog(@"get object metadata failure");
    }
});
[task waitUtilFinished];
```

获取Object的URL

您可以通过如下代码获取指定Object的URL，该功能通常用于您将Object的URL临时分享给其他用户的场景。

基本流程

1. 创建BOSClient类的实例。
2. 执行BOSClient generatePresignedUrl方法。
3. 返回一个Object的URL。

示例代码

```
 __block BOSGeneratePresignedUrlResponse *generateObjetUrlRes = nil;
 BCEOutput_ output = [client generatePresignedUrl:@"<bucketname>" objectKey:@"<objectname>" expirationInSeconds:<ExpirationInSeconds>];
 if (output.response) {
     generateObjetUrlRes = (BOSGeneratePresignedUrlResponse_)output.response;
     NSLog(@"get url success, the usrlstting is : %@", [generateObjetUrlRes.objectUrl absoluteString]);
 }
 if (output.error) {
     NSLog(@"get url failure, error : %@:", output.error);
 }
```

> **说明：** ExpirationInSeconds为指定的URL有效时长，时间从当前时间算起，为可选参数，不配置时系统默认值为1800秒。如果要设置为永久不失效的时间，可以将ExpirationInSeconds参数设置为 -1，不可设置为其他负数。

完整示例

```
 ###### import <BaiduBCEBasic/BaiduBCEBasic.h>
###### import <BaiduBCEBOS/BaiduBCEBOS.h>

void example(void) {
    // 初始化
    BCECredentials* credentials = [[BCECredentials alloc] init];
    credentials.accessKey = @"<access key>";
    credentials.secretKey = @"<secret key>";
    BOSClientConfiguration* configuration = [[BOSClientConfiguration alloc] init];
    configuration.credentials = credentials;

    BOSClient* client = [[BOSClient alloc] initWithConfiguration:configuration];

    __block BOSGeneratePresignedUrlResponse *generateObjetUrlRes = nil;
      BCEOutput* output = [client generatePresignedUrl:@"<bucketname>" objectKey:@"<objectname>" expirationInSeconds:<ExpirationInSeconds>];
    if (output.response) {
    generateObjetUrlRes =  (BOSGeneratePresignedUrlResponse*)output.response;
    NSLog(@"get url success, the usrlstting is : %@", [generateObjetUrlRes.objectUrl absoluteString]);
    }
    if (output.error) {
    NSLog(@"get url failure, error : %@:", output.error);
    }
}
```

删除Object

基本流程

1. 创建BOSClient类的实例。
2. 执行BOSClient deleteObject方法。
3. 若操作失败后产生错误。

示例代码

```
 __block BOSDeleteObjectResponse* response = nil;
BCETask* task = [client deleteObject:@"<bucketname>" objectKey:@"<objectname>"];
task.then(^(BCEOutput* output) {
    if (output.response) {
        response = (BOSDeleteObjectResponse*)output.response;
        NSLog(@"delete obj success!");
    }

    if (output.error) {
        NSLog(@"delete obj failure");
    }
});
[task waitUtilFinished];
```

完整示例

```
 ###### import <BaiduBCEBasic/BaiduBCEBasic.h>
###### import <BaiduBCEBOS/BaiduBCEBOS.h>

void example(void) {
    // 初始化
    BCECredentials* credentials = [[BCECredentials alloc] init];
    credentials.accessKey = @"<access key>";
    credentials.secretKey = @"<secret key>";
    BOSClientConfiguration* configuration = [[BOSClientConfiguration alloc] init];
    configuration.credentials = credentials;

    BOSClient* client = [[BOSClient alloc] initWithConfiguration:configuration];

    __block BOSDeleteObjectResponse* response = nil;
    BCETask* task = [client deleteObject:@"<bucketname>" objectKey:@"<objectname>"];
    task.then(^(BCEOutput* output) {
      if (output.response) {
          response = (BOSDeleteObjectResponse*)output.response;
          NSLog(@"delete obj success!");
      }

      if (output.error) {
          NSLog(@"delete obj failure");
      }
    });
    [task waitUtilFinished];
}
```

拷贝Object

简单拷贝Object

基本流程

1. 创建BOSClient类的实例。

2. 执行BOSClient copyObject方法。

3. 返回BOSCopyObjectResponse类实例，可通过eTag/lastModified等属性获取eTag和最后修改时间。

示例代码

```
 BOSCopyObjectRequest* request = [[BOSCopyObjectRequest alloc] init];
request.bucket = @"<bucketname>";
request.key = @"<objectname>";
request.source = @"<sourceBucket>/<sourceObject";

__block BOSCopyObjectResponse* response = nil;
BCETask* task = [client copyObject:request];
task.then(^(BCEOutput* output) {
    if (output.response) {
        response = (BOSCopyObjectResponse*)output.response;
        NSLog(@"copy obj success!");
    }

    if (output.error) {
        NSLog(@"copy obj failure");
    }
});
[task waitUtilFinished];
```

完整示例

```
 ###### import <BaiduBCEBasic/BaiduBCEBasic.h>
###### import <BaiduBCEBOS/BaiduBCEBOS.h>

void example(void) {
    // 初始化
    BCECredentials* credentials = [[BCECredentials alloc] init];
    credentials.accessKey = @"<access key>";
    credentials.secretKey = @"<secret key>";
    BOSClientConfiguration* configuration = [[BOSClientConfiguration alloc] init];
    configuration.credentials = credentials;

    BOSClient* client = [[BOSClient alloc] initWithConfiguration:configuration];

    BOSCopyObjectRequest* request = [[BOSCopyObjectRequest alloc] init];
    request.bucket = @"<bucketname>";
    request.key = @"<objectname>";
    request.source = @"<sourceBucket>/<sourceObject";

    __block BOSCopyObjectResponse* response = nil;
    BCETask* task = [client copyObject:request];
    task.then(^(BCEOutput* output) {
      if (output.response) {
          response = (BOSCopyObjectResponse*)output.response;
          NSLog(@"copy obj success!");
      }

      if (output.error) {
          NSLog(@"copy obj failure");
      }
    });
    [task waitUtilFinished];
}
```

> **说明：**copyObject 方法返回一个 `BOSCopyObjectResponse` 对象，该对象中包含了新Object的ETag和修改时间。

指定条件拷贝Object

您也可以通过指定条件来实现Object的拷贝。该功能一般用于如下场景：

- Copy一个Object但重新设置meta。
- 重置某个现有Object的meta（把源和目标设置为同一个Object）。
- 当源Object的eTag与指定的eTag相同时复制；
- 当源Object的eTag与指定的eTag不相同时复制；
- 当源Object的在指定的时间后没有被修改时复制；
- 当源Object的在指定的时间后被修改过时复制；

具体内容如下：

基本流程

1. 创建BOSCopyObjectRequest类的实例，传入`<source>`,`<ifMatchEtag>`, `<ifNotMatchEtag>`,`<ifModifiedSince>`,`<ifUnmodifiedSince>`, `<metadataDirective>`参数。

2. 返回BOSCopyObjectResponse类实例，可通过eTag/lastModified属性获取eTag和最后修改时间。

示例代码

```
 BOSCopyObjectRequest* request = [[BOSCopyObjectRequest alloc] init];
request.bucket = @"<bucketname>";
request.key = @"<objectname>";
request.source = @"<sourceBucket>/<sourceObject";
request.metadataDirective = @"replace";
request.ifModifiedSince = @"Wed, 01 Mar 2006 12:00:00 GMT";

__block BOSCopyObjectResponse* response = nil;
BCETask* task = [client copyObject:request];
task.then(^(BCEOutput* output) {
  if (output.response) {
      response = (BOSCopyObjectResponse*)output.response;
      NSLog(@"copy obj success!");
  }

  if (output.error) {
      NSLog(@"copy obj failure");
  }
});
[task waitUtilFinished];
```

> **说明：**`BOSCopyObjectRequest` 允许用户修改目的Object的ObjectMeta，同时也提供 `MatchingETagConstraints` 参数的设定。

Object的分块上传

除了通过putObject()方法上传文件到BOS以外，BOS还提供了另外一种上传模式：分块上传（Multipart Upload）。用户可以在如下的应用场景内（但不仅限于此），使用分块上传模式，如：

- 需要支持断点上传。
- 上传超过5GB大小的文件。
- 网络条件较差，和BOS的服务器之间的连接经常断开。
- 需要流式地上传文件。
- 上传文件之前，无法确定上传文件的大小。

分块完成Multipart Upload

假设有一个文件，本地路径为/path/to/file.zip，由于文件比较大，使用分块上传其传输到BOS中。基本流程：

1. 初始化Multipart Upload。

2. 上传分块。

3. 完成分块上传。

初始化Multipart Upload

使用initiateMultipartUpload方法来初始化一个分块上传事件：

示例代码：

```
 BOSInitiateMultipartUploadRequest* initMPRequest = [[BOSInitiateMultipartUploadRequest alloc] init];
initMPRequest.bucket = @"<bucketname>";
initMPRequest.key = @"<objectname>";
initMPRequest.contentType = @"<content type>";

__block BOSInitiateMultipartUploadResponse* initMPResponse = nil;
BCETask* task = [client initiateMultipartUpload:initMPRequest];
task.then(^(BCEOutput* output) {
  if (output.response) {
      initMPResponse = (BOSInitiateMultipartUploadResponse*)output.response;
      NSLog(@"initiate multipart upload success!");
  }

  if (output.error) {
      NSLog(@"initiate multipart upload failure");
  }
});
[task waitUtilFinished];

NSString* uploadID = initMPResponse.uploadId;
```

说明：initiateMultipartUpload的返回结果中含有uploadId，它是区分分块上传事件的唯一标识，在后面的操作中，我们将用到它。

上传分块

将文件分块上传。

示例代码

```
 // 计算分块个数
NSString* file = @"/path/to/file.zip";
NSDictionary<NSString*, id>* attr = [[NSFileManager defaultManager] attributesOfItemAtPath:file error:nil];
uint64_t fileSize = attr.fileSize;
uint64_t partSize = 1024 * 1024 * 5L;
uint64_t partCount = fileSize / partSize;
if (fileSize % partSize != 0) {
  ++partCount;
}

NSMutableArray<BOSPart*>* parts = [NSMutableArray array];
NSFileHandle* handle = [NSFileHandle fileHandleForReadingAtPath:@"/path/to/file.zip"];
for (uint64_t i = 0; i < partCount; ++i) {
  // seek
  uint64_t skip = partSize * i;
  [handle seekToFileOffset:skip];
  uint64_t size = (partSize < fileSize - skip) ? partSize : fileSize - skip;

  // data
  NSData* data = [handle readDataOfLength:size];

  // request
  BOSUploadPartRequest* uploadPartRequest = [[BOSUploadPartRequest alloc] init];
  uploadPartRequest.bucket = @"<bucketname>";
  uploadPartRequest.key = @"<objectname>";
  uploadPartRequest.objectData.data = data;
  uploadPartRequest.partNumber = i + 1;
  uploadPartRequest.uploadId = uploadID;

  __block BOSUploadPartResponse* uploadPartResponse = nil;
  task = [client uploadPart:uploadPartRequest];
  task.then(^(BCEOutput* output) {
      if (output.response) {
          uploadPartResponse = (BOSUploadPartResponse*)output.response;
          BOSPart* part = [[BOSPart alloc] init];
          part.partNumber = i + 1;
          part.eTag = uploadPartResponse.eTag;
                [parts addObject:part];
      }
  });
  [task waitUtilFinished];
}
```

注意：上面代码的核心是调用uploadPart方法来上传每一个分块，但是要注意以下几点：

- uploadPart方法要求除最后一个Part以外，其他的Part大小都要大于等于5MB。但是Upload Part接口并不会立即校验上传Part的大小；只有当Complete Multipart Upload的时候才会校验。
- 为了保证数据在网络传输过程中不出现错误，建议您在uploadPart后，使用每个分块BOS返回的Content-MD5值分别验证已上传分块数据的正确性。当所有分块数据合成一个Object后，不再含MD5值。
- Part号码的范围是1~10000。如果超出这个范围，BOS将返回InvalidArgument的错误码。
- 每次上传Part时都要把流定位到此次上传块开头所对应的位置。
- 每次上传Part之后，BOS的返回结果会包含一个BOSPart对象，它是上传块的ETag与块编号（PartNumber）的组合，在后续完成分块上传的步骤中会用到它，因此需要将其保存起来。一般来讲这些BOSPart对象将被保存到数组中。

完成分块上传

示例代码

```
 BOSCompleteMultipartUploadRequest* compMultipartRequest = [[BOSCompleteMultipartUploadRequest alloc] init];
compMultipartRequest.bucket = @"<bucketname>";
compMultipartRequest.key = @"<objectname>";
compMultipartRequest.uploadId = uploadID;
compMultipartRequest.parts = parts;

__block BOSCompleteMultipartUploadResponse* complResponse = nil;
task = [client completeMultipartUpload:compMultipartRequest];
task.then(^(BCEOutput* output) {
  if (output.response) {
      complResponse = (BOSCompleteMultipartUploadResponse*)output.response;
      NSLog(@"complte multiparts success!");
  }

  if (output.error) {
      NSLog(@"complte multiparts failure %@", output.error);
  }
});
[task waitUtilFinished];
```

说明：上面代码中的 parts 是第二步中保存的parts的列表，BOS收到用户提交的Part列表后，会逐一验证每个数据Part的有效性。当所有的数据Part验证通过后，BOS将把这些数据part组合成一个完整的Object。

完整示例

```
 ###### import <BaiduBCEBasic/BaiduBCEBasic.h>
###### import <BaiduBCEBOS/BaiduBCEBOS.h>

void example(void) {
// 初始化
BCECredentials* credentials = [[BCECredentials alloc] init];
credentials.accessKey = @"<access key>";
credentials.secretKey = @"<secret key>";
BOSClientConfiguration* configuration = [[BOSClientConfiguration alloc] init];
configuration.credentials = credentials;

BOSClient* client = [[BOSClient alloc] initWithConfiguration:configuration];

// 初始化分块上传
BOSInitiateMultipartUploadRequest* initMPRequest = [[BOSInitiateMultipartUploadRequest alloc] init];
initMPRequest.bucket = @"<bucketname>";
initMPRequest.key = @"<objectname>";
initMPRequest.contentType = @"<content type>";

 __block BOSInitiateMultipartUploadResponse* initMPResponse = nil;
BCETask* task = [client initiateMultipartUpload:initMPRequest];
task.then(^(BCEOutput* output) {
    if (output.response) {
        initMPResponse = (BOSInitiateMultipartUploadResponse*)output.response;
        NSLog(@"initiate multipart upload success!");
    }

    if (output.error) {
        NSLog(@"initiate multipart upload failure");
    }
});
[task waitUtilFinished];

NSString* uploadID = initMPResponse.uploadId;

// 计算分块个数
NSString* file = @"/path/to/file.zip";
NSDictionary<NSString*, id>* attr = [[NSFileManager defaultManager] attributesOfItemAtPath:file error:nil];
uint64_t fileSize = attr.fileSize;
uint64_t partSize = 1024 * 1024 * 5L;
uint64_t partCount = fileSize / partSize;
if (fileSize % partSize != 0) {
    ++partCount;
}

NSMutableArray<BOSPart*>* parts = [NSMutableArray array];

NSFileHandle* handle = [NSFileHandle fileHandleForReadingAtPath:@"/path/to/file.zip"];
for (uint64_t i = 0; i < partCount; ++i) {
    // seek
    uint64_t skip = partSize * i;
    [handle seekToFileOffset:skip];
    uint64_t size = (partSize < fileSize - skip) ? partSize : fileSize - skip;

    // data
    NSData* data = [handle readDataOfLength:size];

    // request
    BOSUploadPartRequest* uploadPartRequest = [[BOSUploadPartRequest alloc] init];
    uploadPartRequest.bucket = @"<bucketname>";
    uploadPartRequest.key = @"<objectname>";
    uploadPartRequest.objectData.data = data;
    uploadPartRequest.partNumber = i + 1;
    uploadPartRequest.uploadId = uploadID;

    __block BOSUploadPartResponse* uploadPartResponse = nil;
    task = [client uploadPart:uploadPartRequest];
    task.then(^(BCEOutput* output) {
        if (output.response) {
            uploadPartResponse = (BOSUploadPartResponse*)output.response;
            BOSPart* part = [[BOSPart alloc] init];
            part.partNumber = i + 1;
```

```
            part.eTag = uploadPartResponse.eTag;
                [parts addObject:part];
        }
    });
    [task waitUtilFinished];
}

BOSCompleteMultipartUploadRequest* compMultipartRequest = [[BOSCompleteMultipartUploadRequest alloc] init];
compMultipartRequest.bucket = @"<bucketname>";
compMultipartRequest.key = @"<objectname>";
compMultipartRequest.uploadId = uploadID;
compMultipartRequest.parts = parts;

__block BOSCompleteMultipartUploadResponse* complResponse = nil;
task = [client completeMultipartUpload:compMultipartRequest];
task.then(^(BCEOutput* output) {
    if (output.response) {
        complResponse = (BOSCompleteMultipartUploadResponse*)output.response;
        NSLog(@"complte multiparts success!");
    }

    if (output.error) {
        NSLog(@"complte multiparts failure %@", output.error);
    }
});
[task waitUtilFinished];
}
```

取消分块上传

用户可以使用abortMultipartUpload方法取消分块上传。

示例代码：

```
 BOSAbortMultipartUploadRequest* abortRequest = [[BOSAbortMultipartUploadRequest alloc] init];
abortRequest.bucket = @"bucket";
abortRequest.key = @"<objectname>";
abortRequest.uploadId = uploadID;

__block BOSAbortMultipartUploadResponse* abortResponse = nil;
task = [client abortMultipartUpload:abortRequest];
task.then(^(BCEOutput* output) {
  if (output.response) {
      abortResponse = (BOSAbortMultipartUploadResponse*)output.response;
      NSLog(@"abort multiparts success!");
  }

  if (output.error) {
      NSLog(@"abort multiparts failure %@", output.error);
  }
});
[task waitUtilFinished];
```

获取未完成的分块上传

用户可以使用listMultipartUploads方法获取Bucket内未完成的分块上传事件。

基本流程

1. 创建BOSListMultipartUploadsRequest类的实例,传入<BucketName>参数。

2. 创建BOSClient类的实例,执行BOSClient listMultipartUploads方法。

3. listMultipartUploads返回所有未完成的分块上传信息。

示例代码

```
 BOSListMultipartUploadsRequest* listMultipartRequest = [[BOSListMultipartUploadsRequest alloc] init];
listMultipartRequest.bucket = @"<bucketname>";

__block BOSListMultipartUploadsResponse* listMultipartResponse = nil;
task = [client listMultipartUploads:listMultipartRequest];
task.then(^(BCEOutput* output) {
  if (output.response) {
      listMultipartResponse = (BOSListMultipartUploadsResponse*)output.response;
      NSLog(@"list multipart success");
  }

  if (output.error) {
      NSLog(@"list multipart failure %@", output.error);
  }
});
[task waitUtilFinished];
```

> **注意：**
>
> - 默认情况下，如果Bucket中的分块上传事件的数目大于1000，则只会返回1000个Object，并且返回结果中IsTruncated的值为True，同时返回nextKeyMarker作为下次读取的起点。
> - 若想获取更多分块上传事件，可以使用keyMarker参数分次读取。

完整示例

```
 ###### import <BaiduBCEBasic/BaiduBCEBasic.h>
###### import <BaiduBCEBOS/BaiduBCEBOS.h>

void example(void) {
    // 初始化
    BCECredentials* credentials = [[BCECredentials alloc] init];
    credentials.accessKey = @"<access key>";
    credentials.secretKey = @"<secret key>";
    BOSClientConfiguration* configuration = [[BOSClientConfiguration alloc] init];
    configuration.credentials = credentials;

    BOSClient* client = [[BOSClient alloc] initWithConfiguration:configuration];

    BOSListMultipartUploadsRequest* listMultipartRequest = [[BOSListMultipartUploadsRequest alloc] init];
    listMultipartRequest.bucket = @"<bucketname>";

    __block BOSListMultipartUploadsResponse* listMultipartResponse = nil;
    BCETask* task = [client listMultipartUploads:listMultipartRequest];
    task.then(^(BCEOutput* output) {
      if (output.response) {
          listMultipartResponse = (BOSListMultipartUploadsResponse*)output.response;
          NSLog(@"list multipart success");
      }

      if (output.error) {
          NSLog(@"list multipart failure %@", output.error);
      }
    });
    [task waitUtilFinished];

    for (BOSMultipartUpload* upload in listMultipartResponse.uploads) {
        NSLog(@"upload id : %@", upload.uploadId);
    }
}
```

获取所有已上传的分块信息

用户可以使用listParts方法获取某个上传事件中所有已上传的块。

基本流程

1. 创建BOSListPartsRequest类的实例,传入<BucketName>,<ObjectKey>, <UploadId>参数。

2. 创建BOSClient类的实例,执行BOSClient listParts方法。

3. listParts返回所有已上传part的信息。

示例代码

```
 BOSListPartsRequest* listPartsRequest = [[BOSListPartsRequest alloc] init];
listPartsRequest.bucket = @"<bucketname>";
listPartsRequest.key = @"<objectname>";
listPartsRequest.uploadId = @"<upload id>";;

__block BOSListPartsResponse* listPartsResponse = nil;
BCETask* task = [client listParts:listPartsRequest];
task.then(^(BCEOutput* output) {
  if (output.response) {
      listPartsResponse = (BOSListPartsResponse*)output.response;
      NSLog(@"list parts success!");
  }

  if (output.error) {
      NSLog(@"list part failure %@", output.error);
  }
});
[task waitUtilFinished];

for (BOSPart* part in listPartsResponse.parts) {
    NSLog(@"part etag %@", part.eTag);
}
```

> **注意：**
>
> - 默认情况下，如果Bucket中的分块上传事件的数目大于1000，则只会返回1000个Object，并且返回结果中IsTruncated的值为True，同时返回NextPartNumberMarker作为下次读取的起点。
> - 若想获取更多已上传的分块信息，可以使用PartNumberMarker参数分次读取。

完整示例

```
 ###### import <BaiduBCEBasic/BaiduBCEBasic.h>
###### import <BaiduBCEBOS/BaiduBCEBOS.h>

void example(void) {
    // 初始化
    BCECredentials* credentials = [[BCECredentials alloc] init];
    credentials.accessKey = @"<access key>";
    credentials.secretKey = @"<secret key>";
    BOSClientConfiguration* configuration = [[BOSClientConfiguration alloc] init];
    configuration.credentials = credentials;

    BOSClient* client = [[BOSClient alloc] initWithConfiguration:configuration];

    BOSListPartsRequest* listPartsRequest = [[BOSListPartsRequest alloc] init];
    listPartsRequest.bucket = @"<bucketname>";
    listPartsRequest.key = @"<objectname>";
    listPartsRequest.uploadId = @"<upload id>";;

    __block BOSListPartsResponse* listPartsResponse = nil;
    BCETask* task = [client listParts:listPartsRequest];
    task.then(^(BCEOutput* output) {
    if (output.response) {
        listPartsResponse = (BOSListPartsResponse*)output.response;
        NSLog(@"list parts success!");
    }

    if (output.error) {
        NSLog(@"list part failure %@", output.error);
    }
    });
    [task waitUtilFinished];

    for (BOSPart* part in listPartsResponse.parts) {
        NSLog(@"part etag %@", part.eTag);
    }
}
```

异常处理

在发生错误时，可以查看BCEOutput的error字段,error字段的详细解释请参见错误信息格式。

日志

暂未支持输出日志。

版本变更记录

V1.3.1 [2024-03-05]

- iOS开启默认virtual-host域名方式访问BOS

V1.2.1 [2023-03-27]

- iOS端支持使用STS生成presigned URL

V1.1.1 [2018-12-05]

- 解决客户端时间偏差导致上传问题
- 支持生成Object URL
- 支持获取Object的storageClass

V1.0.0 [2016-08-04]

首次发布：

- 支持http和https协议
- 支持创建、查看、罗列、删除Bucket
- 支持修改、获取Bucket的访问权限
- 支持上传、查看、罗列、删除Object
- 支持分块上传方式
- 支持STS验证
- 支持HTML5 CORS(跨域访问)相关接口

C#-SDK

概述

本文档主要介绍BOS C# SDK的安装和使用。在使用本文档前，您需要先了解BOS的一些基本知识，并已经开通了BOS服务。若您还不了解BOS，可以参考产品描述和开发者指南。

安装SDK工具包

运行环境

C# SDK工具包可在Microsoft .NET Framework v4.6环境下运行。

安装步骤

1. 在官方网站下载C# SDK压缩工具包。
2. 将下载的bce-dotnet-sdk-version.zip解压后，复制到工程文件夹中。
3. 在Visual Studio项目中“添加引用 -> 浏览”。

4. 添加SDK工具包BceSdkDotNet.dll和第三方依赖工具包log4net.dll和Newtonsoft.Json.dll。

SDK目录结构

```
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
    |       └──── Bos                   //BOS服务相关类
    |           ├──── Model             //BOS内部model，如Request或Response
    |           ├──── BosClient.cs      //BOS客户端入口类
    |           └──── BosConstants.cs   //BOS特有的常量定义，如权限常量等
    |       └──── Sts                   //Sts认证相关类
    |           ├──── Model             //BOS内部model，如Request或Response
    |           └────  StsClient.cs     //STS客户端入口类
    ├──── Util                          //BCE公用工具类
    |
    ├──── BceBaseException.cs           //BCE基础的异常类
    ├──── BceClientBase.cs              //BCE客户端的基础类
    ├──── BceClientConfiguration.cs     //对BCE的HttpClient的配置
    ├──── BceClientException.cs         //BCE客户端的异常类
    ├──── BceServiceException.cs        //与BCE服务端交互后的异常类
    ├──── BceConstants.cs               //BCE的通用常量(区域，请求头，错误码等)
    ├──── DefaultRetryPolicy.cs         //缺省的重试策略
    └──── IRetryPolicy.cs               //重试策略
```

初始化

快速入门

1. 初始化一个BosClient。

   BosClient是与BOS服务交互的客户端，BOS C# SDK的BOS操作都是通过BosClient完成的。

   示例代码：

```
class BosClientSample
{
  static void Main(string[] args)
  {
     const string accessKeyId = <AccessKeyID>; // 您的Access Key ID
     const string secretAccessKey = <SecretAccessKey>; // 您的Secret Access Key
     const string endpoint = "https://bj.bcebos.com";        //传入Bucket所在区域域名

     // 初始化一个BosClient
     BceClientConfiguration config = new BceClientConfiguration();
     config.Credentials = new DefaultBceCredentials(accessKeyId, secretAccessKey);
     config.Endpoint = endpoint;
     BosClient client = new BosClient(config);
  }
}
```

2. 新建一个Bucket。

   Bucket是BOS上的命名空间，相当于数据的容器，可以存储若干数据实体（Object）。在您上传数据前，必须先创建一个Bucket。

   示例代码：

```
public void CreateBucket(BosClient client, string bucketName)
{
  // 新建一个Bucket
  client.CreateBucket(bucketName);
}
```

3. 上传Object。

   Object是BOS中最基本的数据单元，您可以把Object简单的理解为文件。对于一个简单的Object的上传，BOS为您提供了四种方式：文件形式上传、数据流形式上传、二进制串上传和字符串上传。

   示例代码：

```
public void PutObject(BosClient client, String bucketName, String objectKey, byte[] byte1, String string1)
{
  // 获取指定文件
  FileInfo file = new FileInfo(<FilePath>);   //指定文件路径

  // 以文件形式上传Object
  PutObjectResponse putObjectFromFileResponse = client.PutObject(bucketName, objectKey, file);

  // 获取数据流
  Stream inputStream = file.OpenRead();

  // 以数据流形式上传Object
  PutObjectResponse putObjectResponseFromInputStream = client.PutObject(bucketName, objectKey, inputStream);

  // 以二进制串上传Object
  PutObjectResponse putObjectResponseFromByte = client.PutObject(bucketName, objectKey, Encoding.Default.GetBytes("sampledata"));

  // 以字符串上传Object
  PutObjectResponse putObjectResponseFromString = client.PutObject(bucketName, objectKey, "sampledata");

  // 打印ETag
  Console.WriteLine(putObjectFromFileResponse.ETAG);
}
```

4. 查看Bucket下的Object列表。

   当您完成一系列上传后，可以参考如下代码来查看Bucket下的全部Object。

   示例代码：

```
public void ListObjects(BosClient client, string bucketName)
{
    // 获取指定Bucket下的所有Object信息
    ListObjectsResponse listObjectsResponse = client.ListObjects(bucketName);

    // 遍历所有Object
    foreach (BosObjectSummary objectSummary in listObjectsResponse.Contents)
    {
        Console.WriteLine("ObjectKey: " + objectSummary.Key);
    }
}
```

5. 获取指定Object

用户可以参考如下代码来实现对一个或者多个Object的获取。

示例代码：

```
public void GetObject(BosClient client, String bucketName, String objectKey)
{
    // 获取Object，返回结果为BosObject对象
    BosObject bosObject = client.GetObject(bucketName, objectKey);

    // 获取ObjectMeta
    ObjectMetadata meta = bosObject.ObjectMetadata;

    // 获取Object的输入流
    Stream objectContent = bosObject.ObjectContent;

    // 处理Object
    ...

    // 关闭流
    objectContent.Close();
}
```

完整示例

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Text;
using BaiduBce;
using BaiduBce.Auth;
using BaiduBce.Services.Bos;
using BaiduBce.Services.Bos.Model;

namespace DotnetSample
{
    internal class BaseSample
    {
        private static void Main(string[] args)
        {
            BosClient client = GenerateBosClient();
            const string bucketName = <BucektName>;    //指定Bucket名称
            const string objectKey = <ObjectKey>;    //指定object名称

            //创建Bucket
            client.CreateBucket(bucketName);

            //上传Object
            FileInfo file = new FileInfo("d:\\lzb\\sample.txt");    //指定上传文件的路径
            PutObjectResponse putObjectFromFileResponse = client.PutObject(bucketName, objectKey, file);
            Console.WriteLine(putObjectFromFileResponse.ETAG);

            //查看Object
            ListObjectsResponse listObjectsResponse = client.ListObjects(bucketName);
            foreach (BosObjectSummary objectSummary in listObjectsResponse.Contents)
            {
                Console.WriteLine("ObjectKey: " + objectSummary.Key);
            }

            // 获取Object
            BosObject bosObject = client.GetObject(bucketName, objectKey);
            // 获取ObjectMeta
            ObjectMetadata meta = bosObject.ObjectMetadata;
            // 获取Object的输入流
            Stream objectContent = bosObject.ObjectContent;
            // 处理Object
            FileStream fileStream = new FileInfo("d:\\lzb\\sampleout.txt").OpenWrite();    //指定下载文件的目录/文件名
            byte[] buffer = new byte[2048];
            int count = 0;
            while ((count = objectContent.Read(buffer, 0, buffer.Length)) > 0)
            {
                fileStream.Write(buffer, 0, count);
            }

            // 关闭流
            objectContent.Close();
            fileStream.Close();
            Console.WriteLine(meta.ETag);
            Console.WriteLine(meta.ContentLength);
        }

        private static BosClient GenerateBosClient()
        {
            const string accessKeyId = <AccessKeyID>; // 您的Access Key ID
            const string secretAccessKey = <SecretAccessKey>; // 您的Secret Access Key
            const string endpoint = "https://bj.bcebos.com";        //指定Bucket所在区域域名

            // 初始化一个BosClient
            BceClientConfiguration config = new BceClientConfiguration();
            config.Credentials = new DefaultBceCredentials(accessKeyId, secretAccessKey);
            config.Endpoint = endpoint;

            return new BosClient(config);
        }
    }
}
```

新建BosClient

BosClient是BOS服务的客户端，为开发者与BOS服务进行交互提供了一系列的方法。

使用AK/SK新建BosClient

- 基本流程
  1. 确定EndPoint。EndPoint是指BOS服务在各个区域的域名地址，默认域名为北京http://bj.bcebos.com。
  2. 创建一个BceClientConfiguration实例。
  3. 使用您的AK/SK创建DefaultBceCredentials并赋值给BceClientConfiguration的Credentials属性。
  4. 用配置好的BceClientConfiguration创建BosClient实例。
- 示例代码

```
class BosClientSample
{
    static void Main(string[] args)
    {
        const string accessKeyId = <AccessKeyID>;          // 您的Access Key ID
        const string secretAccessKey = <SecretAccessKey>; // 您的Secret Access Key
        const string endpoint = "https://bj.bcebos.com";  // 传入Bucket所在区域域名

        // 初始化一个BosClient
        BceClientConfiguration config = new BceClientConfiguration();
        config.Credentials = new DefaultBceCredentials(accessKeyId, secretAccessKey);
        config.Endpoint = endpoint;
        BosClient client = new BosClient(config);
    }
}
```

EndPoint参数只能用指定的包含区域的域名来进行定义，不指定时默认为北京区域http://bj.bcebos.com。百度智能云目前开放了多区域支持，请参考区域选择说明。 目前支持"华北-北京"、"华南-广州"和"华东-苏州"三个区域。北京区域：http://bj.bcebos.com，广州区域：http://gz.bcebos.com，苏州区域：http://su.bcebos.com。

使用STS新建BosClient

- 基本流程

1. 确定StsEndPoint 。StsEndPoint 是指授权服务的域名地址，默认域名为http://sts.bj.baidubce.com。
2. 创建一个BceClientConfiguration实例。
3. 使用您的AK/SK创建DefaultBceCredentials并赋值给BceClientConfiguration的Credentials属性。
4. 用配置好的BceClientConfiguration创建StsClient实例。
5. 通过StsClient获取临时AK/SK、token用于临时授权。
6. 确定EndPoint。EndPoint是指BOS服务在各个区域的域名地址，默认域名为北京http://bj.bcebos.com。
7. 再次创建一个BceClientConfiguration实例。
8. 使用临时AK/SK、token创建DefaultBceCredentials并赋值给BceClientConfiguration的Credentials属性。
9. 用配置好的BceClientConfiguration创建BosClient实例。

- 示例代码

```
class BosClientSample
{
    static void Main(string[] args)
    {
        const string accessKeyId = <AccessKeyID>;              // 您的Access Key ID
        const string secretAccessKey = <SecretAccessKey>;      // 您的Secret Access Key
        const string endpoint = "https://bj.bcebos.com";       // 指定Bucket所在区域域名
        const string stsEndpoint = "http://sts.bj.baidubce.com";// 指定授权服务域名

        // 获取临时授权Token
        BceClientConfiguration stsConfig = new BceClientConfiguration();
        stsConfig.Credentials = new DefaultBceCredentials(accessKeyId, secretAccessKey);
        stsConfig.Endpoint = stsEndpoint;
        StsClient stsClient = new StsClient(stsConfig);
        GetSessionTokenResponse response = stsClient.GetSessionToken();

        // 使用Token初始化一个BosClient
        BceClientConfiguration config = new BceClientConfiguration();
        config.Credentials = new DefaultBceSessionCredentials(
                response.AccessKeyId,
                response.SecretAccessKey,
                response.SessionToken);
        config.Endpoint = endpoint;
        BosClient client = new BosClient(config);
    }
}
```

注意： 目前使用STS配置client时，无论对应bucket的区域在哪里，stsEndpoint都需配置为http://sts.bj.baidubce.com， 但创建BosClient时，仍需使用BOS的endpoint，如bj.bcebos.com、su.bcebos.com等.

配置HTTPS协议访问BOS

BOS支持HTTPS传输协议，您可以将EndPoint直接设置为https即可，不设置EndPoint时，默认为http协议。

- 示例代码

```
class BosClientSample
{
    static void Main(string[] args)
    {
        const string accessKeyId = <AccessKeyID>; // 您的Access Key ID
        const string secretAccessKey = <SecretAccessKey>; // 您的Secret Access Key
        const string endpoint =  "https://bj.bcebos.com" ;        //传入Bucket所在区域域名，并直接设置为https协议

        // 初始化一个BosClient
        BceClientConfiguration config = new BceClientConfiguration();
        config.Credentials = new DefaultBceCredentials(accessKeyId, secretAccessKey);
        config.Endpoint = endpoint;
        BosClient client = new BosClient(config);
    }
}
```

配置自定义域名访问BOS

如果希望使用自定义域名作为访问BOS的endpoint，在控制台将自定义域名和BOS某个bucket绑定之后，配置endpoint为自定义域名并打开CnameEnabled开关，例如cdn-test.cdn.bcebos.com，配置代码如下：

```
 class BosClientSample
{
    static void Main(string[] args)
    {
        const string accessKeyId = <AccessKeyID>;          // 您的Access Key ID
        const string secretAccessKey = <SecretAccessKey>; // 您的Secret Access Key
        const string endpoint = <EndPoint>;  // 传入自定义域名

        // 初始化一个BosClient
        BceClientConfiguration config = new BceClientConfiguration();
        config.Credentials = new DefaultBceCredentials(accessKeyId, secretAccessKey);
        config.Endpoint = endpoint;
       config.CnameEnabled = true;
        BosClient client = new BosClient(config);
    }
}
```

配置BosClient

- 基本流程

  1. 创建一个BceClientConfiguration实例。
  2. 使用您的AK/SK创建DefaultBceCredentials并赋值给BceClientConfiguration的Credentials属性。
  3. 对BceClientConfiguratio实例的各种属性进行赋值。
  4. 用配置好的BceClientConfiguration创建BosClient实例。

如果用户需要配置BosClient的一些细节的参数，可以在构造BosClient的时候传入BosClientConfiguration对象。 BosClientConfiguration是BOS服务的配置类，可以为客户端配置超时时间，最大连接数等参数。

设置网络参数

- 示例代码

```
 BceClientConfiguration config = new BceClientConfiguration();

// 设置HTTP最大连接数为10
config.ConnectionLimit = 10;

// 设置TCP连接超时为5000毫秒
config.TimeoutInMillis = 5000;

// 设置读写数据超时的时间为50000毫秒
config.ReadWriteTimeoutInMillis = 50000;
```

- 参数说明

  通过BceClientConfiguration能指定的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，缺省值为HTTP协议
TimeoutInMillis	建立连接的超时时间（单位：毫秒），缺省值为30000
ReadWriteTimeoutInMillis	通过打开的连接传输数据的超时时间（单位：毫秒），缺省值为30000
ConnectionLimit	允许打开的最大HTTP连接数，缺省值为5
RetryPolicy	连接重试策略
SocketBufferSizeInBytes	Socket缓冲区大小
CnameEnabled	是否使用自定义域名，默认为false，开启前需确认已经申请自定义域名
PathStyleEnabled	是否使用三级域名方式请求，默认为false使用四级域名，如有服务需要使用三级域名时开启，一般不建议设为true

- 完整示例

  下面示例代码演示了BosClient的创建和配置。

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using BaiduBce;
using BaiduBce.Auth;
using BaiduBce.Services.Bos;

namespace DotnetSample
{
    internal class BosClientSample
    {
        private static void Main(string[] args)
        {
            const string accessKeyId = <AccessKeyID>; // 您的Access Key ID
            const string secretAccessKey = <SecretAccessKey>; // 您的Secret Access Key
            const string endpoint = "https://bj.bcebos.com";

            // 初始化一个BosClient
            BceClientConfiguration config = new BceClientConfiguration();
            config.Credentials = new DefaultBceCredentials(accessKeyId, secretAccessKey);
            config.Endpoint = endpoint;

            // 设置HTTP最大连接数为10
            config.ConnectionLimit = 10;

            // 设置TCP连接超时为5000毫秒
            config.TimeoutInMillis = 5000;

            // 设置读写数据超时的时间为50000毫秒
            config.ReadWriteTimeoutInMillis = 50000;

            BosClient client = new BosClient(config);
        }
    }
}
```

Bucket管理

新建Bucket

- 基本流程
  1. 创建一个BosClient类的实例。
  2. 调用BosClient.CreateBucket()方法。您需要提供Bucket的名字，即<Bucketname> 。
- 示例代码

```
public void CreateBucket(BosClient client, string bucketName)
{
    // 新建一个Bucket
    client.CreateBucket(bucketName);
}
```

> **注意：**由于Bucket的名称在所有区域中是唯一的，所以需要保证bucketName不与其他所有区域上的Bucket名称相同。

- 完整示例

  请参考完整示例。

查看Bucket列表

- 基本流程
  1. 创建一个BosClient类的实例。
  2. 调用BosClient.ListBuckets()方法获取Bucket列表。
  3. 遍历Bucket列表并查看相关信息。
- 示例代码

```
public void ListBuckets(BosClient client)
{

    // 获取用户的Bucket列表
    List<BucketSummary> buckets = client.ListBuckets().Buckets;

    // 遍历Bucket
    foreach (BucketSummary bucket in buckets)
    {
        Console.WriteLine(bucket.Name);
    }

}
```

请参考完整示例。

判断Bucket是否存在

- 基本流程
  1. 创建一个BosClient类的实例。
  2. 调用BosClient.DoesBucketExist()方法获取结果。

- 示例代码

  若用户需要判断某个Bucket是否存在，则如下代码可以做到：

```
public void DoesBucketExist(BosClient client, string bucketName)
{

    // 获取Bucket的存在信息
    bool exists = client.DoesBucketExist(bucketName); //指定Bucket名称

    // 输出结果
    if (exists)
    {
        Console.WriteLine("Bucket exists");
    }
    else
    {
        Console.WriteLine("Bucket not exists");
    }
}
```

- 完整示例

  请参考完整示例。

删除Bucket

- 基本流程

  1. 创建一个BosClient类的实例。
  2. 调用BosClient.DeleteBucket()方法删除Bucket。

- 示例代码

```
public void DeleteBucket(BosClient client, string bucketName)
{
    // 删除Bucket
    client.DeleteBucket(bucketName);
}
```

> **注意：**如果Bucket不为空（即Bucket中有Object和未完成的三步上传Part存在），则Bucket无法被删除，必须清空Bucket后才能成功删除。

- 完整示例

  请参考完整示例。

Bucket权限控制

- 基本流程

  1. 创建一个BosClient类的实例。
  2. 调用BosClient.SetBucketAcl()方法。调用该方法时，您需要提供Bucket名称、权限字符串等内容。
  3. 您也可以构造一个SetBucketAclRequest来调用SetBucketAcl方法，SetBucketAclRequest可以设置指定用户对Bucket的访问权限。

设置Bucket的访问权限

- 示例代码

  如下代码将Bucket的权限设置为了private。

```
public void SetBucketPrivate(BosClient client, string bucketName)
{
    client.SetBucketAcl(bucketName, BosConstants.CannedAcl.Private);
}
```

> **说明：**BosConstants.CannedAcl包含三个常量定义： `Private` 、 `PublicRead` 、 `PublicReadWrite` ，它们分别对应相关权限。具体内容可以参考《BOS API文档 使用CannedAcl方式的权限控制》。

设置指定用户对Bucket的访问权限

- 示例代码

  BOS还可以实现设置指定用户对Bucket的访问权限，参考如下代码实现：

```
public void SetBucketAclFromBody(BosClient client, string bucketName)
{
    List<Grant> grants = new List<Grant>();
    List<Grantee> grantee = new List<Grantee>();
    List<string> permission = new List<string>();

    //授权给特定用户
    grantee.Add(new Grantee() {Id = "userid1"});
    grantee.Add(new Grantee() {Id = "userid2"});

    //授权给Everyone
    grantee.Add(new Grantee() {Id = "*"});

    //设置权限
    permission.Add(BosConstants.Permission.Read);
    permission.Add(BosConstants.Permission.Write);

    grants.Add(new Grant() {Grantee = grantee, Permission = permission});
    client.SetBucketAcl(new SetBucketAclRequest() {BucketName = bucketName, AccessControlList = grants});
}
```

注意：

- BosConstants.Permission中包含三个常量定义：`READ`、`WRITE`、`FULL_CONTROL`，它们分别对应相关权限。具体内容可以参考《BOS API文档 上传ACL文件方式的权限控制》；
- C# SDK不支持通过STS方式进行鉴权。

完整示例

下面示例代码演示了Bucket权限控制的完整过程：

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using BaiduBce;
using BaiduBce.Auth;
using BaiduBce.Services.Bos;
using BaiduBce.Services.Bos.Model;

namespace DotnetSample
{
    internal class BucketAclSample
    {
        private static void Main(string[] args)
        {
            BosClient client = GenerateBosClient();
            const string bucketName = <BucketName>; //Bucket名称

            // 新建一个Bucket
            client.CreateBucket(bucketName); //指定Bucket名称


            // 设置Bucket为Private
            client.SetBucketAcl(bucketName, BosConstants.CannedAcl.Private);

            // 设置Bucket为PublicRead
            client.SetBucketAcl(bucketName, BosConstants.CannedAcl.PublicRead);

            List<Grant> grants = new List<Grant>();
            List<Grantee> grantee = new List<Grantee>();
            List<string> permission = new List<string>();

            //授权给特定用户
            grantee.Add(new Grantee() { Id = "userid1" });
            grantee.Add(new Grantee() { Id = "userid2" });
            //授权给Everyone
            grantee.Add(new Grantee() { Id = "*" });

            //设置权限
            permission.Add(BosConstants.Permission.Read);
            permission.Add(BosConstants.Permission.Write);

            grants.Add(new Grant() { Grantee = grantee, Permission = permission });
            client.SetBucketAcl(new SetBucketAclRequest() { BucketName = bucketName, AccessControlList = grants });

        }

        private static BosClient GenerateBosClient()
        {
            const string accessKeyId = <AccessKeyID>;        // 您的Access Key ID
            const string secretAccessKey = <SecretAccessKey>; // 您的Secret Access Key
            const string endpoint = "https://bj.bcebos.com";       // 指定BOS服务域名

            // 初始化一个BosClient
            BceClientConfiguration config = new BceClientConfiguration();
            config.Credentials = new DefaultBceCredentials(accessKeyId, secretAccessKey);
            config.Endpoint = endpoint;

            return new BosClient(config);
        }
    }
}
```

完整示例

下面示例代码演示了创建Bucket，查看Bucket，判断Bucket是否存在，删除Bucket的完整过程。

```
 using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using BaiduBce;
using BaiduBce.Auth;
using BaiduBce.Services.Bos;
using BaiduBce.Services.Bos.Model;

namespace DotnetSample
{
    internal class BucketSample
    {
        private static void Main(string[] args)
        {
            BosClient client = GenerateBosClient();
            const string bucketName = <BucketName>;    //指定Bucket名称

            // 新建一个Bucket
            client.CreateBucket(bucketName);

            // 获取用户的Bucket列表
            List<BucketSummary> buckets = client.ListBuckets().Buckets;

            // 遍历Bucket
            foreach (BucketSummary bucket in buckets)
            {
                Console.WriteLine(bucket.Name);
            }

            // 获取Bucket的存在信息
            bool exists = client.DoesBucketExist(bucketName);

            // 输出结果
            if (exists)
            {
                Console.WriteLine("Bucket exists");
            }
            else
            {
                Console.WriteLine("Bucket not exists");
            }

            // 删除Bucket
            client.DeleteBucket(bucketName);

            // 再次获取Bucket的存在信息
            exists = client.DoesBucketExist(bucketName);

            // 输出结果
            if (exists)
            {
                Console.WriteLine("Bucket exists");
            }
            else
            {
                Console.WriteLine("Bucket not exists");
            }
            Console.ReadKey();
        }

        private static BosClient GenerateBosClient()
        {
            const string accessKeyId = <AccessKeyID>; // 您的Access Key ID
            const string secretAccessKey = <SecretAccessKey>; // 您的Secret Access Key
            const string endpoint = "https://bj.bcebos.com"; // 指定BOS服务域名

            // 初始化一个BosClient
            BceClientConfiguration config = new BceClientConfiguration();
            config.Credentials = new DefaultBceCredentials(accessKeyId, secretAccessKey);
            config.Endpoint = endpoint;

            return new BosClient(config);
        }
    }
}
```

文件管理

上传Object

最简单的上传

- 基本流程
    1. 创建一个BosClient类的实例。
    2. 调用BosClient.PutObject()方法。调用该方法时您需要提供Bucket名称、ObjectKey和Object的内容。可以通过如下四种方式上传Object：文件、数据流、二进制串和字符串的形式。
- 示例代码

```
public void PutObject(BosClient client, String bucketName, String objectKey, byte[] byte1, String string1)
{

    // 获取指定文件
    FileInfo file = new FileInfo(<FilePath>);    //指定文件路径

    // 以文件形式上传Object
    PutObjectResponse putObjectFromFileResponse = client.PutObject(bucketName, objectKey, file);

    // 获取数据流
    Stream inputStream = file.OpenRead();

    // 以数据流形式上传Object
    PutObjectResponse putObjectResponseFromInputStream = client.PutObject(bucketName, objectKey, inputStream);

    // 以二进制串上传Object
    PutObjectResponse putObjectResponseFromByte = client.PutObject(bucketName, objectKey,Encoding.Default.GetBytes("sampledata"));

    // 以字符串上传Object
    PutObjectResponse putObjectResponseFromString = client.PutObject(bucketName, objectKey, "sampledata");

    // 打印ETag
    Console.WriteLine(putObjectFromFileResponse.ETAG);

}
```

> **说明：**Object以文件的形式上传到BOS中，putObject函数支持不超过5GB的Object上传。在putObject请求处理成功后，BOS会在Header中返回Object的ETag作为文件标识。

- 完整示例

  请参考完整示例。

设定Object的Http Header

BOS支持您在上传object时设定Http Header。

- 基本流程

  1. 创建一个BosClient类的实例。
  2. 调用BosClient.PutObject()时候，您还可以传入一个ObjectMetadata对象，该对象可以设置自定义的Http Header。

- 示例代码

```
// 初始化上传输入流
ObjectMetadata meta = new ObjectMetadata();

// 设置ContentLength大小
meta.ContentLength = <Length>;

// 设置ContentType
meta.ContentType = "application/json";

client.PutObject(bucketName, objectKey, <SampleData>, meta);
```

- 完整示例

  请参考完整示例。

用户自定义元数据

- 基本流程

  1. 创建一个BosClient类的实例。
  2. 调用BosClient.PutObject()时候，您还可以传入一个ObjectMetadata对象，该对象支持用户自定义元数据来对Object进行描述。

- 示例代码

```
// 设置自定义元数据name的值为my-data
meta.UserMetadata["name"] = "my-data";

// 上传Object
client.PutObject(bucketName, objectKey, <SampleData>, meta);
```

> **说明：**在上面代码中，用户自定义了一个名字为"name"，值为"my-data"的元数据。当用户下载此Object的时候，此元数据也可以一并得到。一个Object可以有多个类似的参数，但所有的User Meta总大小不能超过2KB。

- 完整示例

  请参考完整示例。

完整示例

下面示例代码演示了简单上传Object、设定Object的Http Header和用户自定义元数据的完整过程：

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Text;
using BaiduBce;
using BaiduBce.Auth;
using BaiduBce.Services.Bos;
using BaiduBce.Services.Bos.Model;

namespace DotnetSample
{
    internal class PutObjectSample
    {
        private static void Main(string[] args)
        {
            BosClient client = GenerateBosClient();
            const string bucketName = <BucketName>;        //您的Bucket名称
            const string objectNameFile = <ObjectNameFile>;  //文件形式上传的Object名称
            const string objectNameStream = <ObjectNameStream>; //数据流形式上传的Object名称
            const string objectNameString = <ObjectNameString>; //字符串形式上传的Object名称
            const string objectNameByte = <ObjectNameByte>; //二进制形式上传的Object名称

            // 新建一个Bucket
            client.CreateBucket(bucketName); //指定Bucket名称

            // 设置待上传的文件名
            const string fileName = "d:\\sample.txt";

            // 以文件形式上传Object
            PutObjectResponse putObjectFromFileResponse = client.PutObject(bucketName, objectNameFile,
                new FileInfo(fileName));

            // 以数据流形式上传Object
            PutObjectResponse putObjectResponseFromInputStream = client.PutObject(bucketName, objectNameStream,
                new FileInfo(fileName).OpenRead());

            // 以二进制串上传Object
            PutObjectResponse putObjectResponseFromByte = client.PutObject(bucketName, objectNameByte,
                Encoding.Default.GetBytes("sampledata"));

            // 以字符串上传Object
            PutObjectResponse putObjectResponseFromString = client.PutObject(bucketName, objectNameString,
                "sampledata");

            // 打印四种方式的ETag。示例中，文件方式和stream方式的ETag相等，string方式和byte方式的ETag相等
            Console.WriteLine(putObjectFromFileResponse.ETAG);
            Console.WriteLine(putObjectResponseFromInputStream.ETAG);
            Console.WriteLine(putObjectResponseFromByte.ETAG);
            Console.WriteLine(putObjectResponseFromString.ETAG);

            // 上传Object并设置自定义参数
            ObjectMetadata meta = new ObjectMetadata();
            // 设置ContentLength大小
            meta.ContentLength = 10;
            // 设置ContentType
            meta.ContentType = "application/json";
            // 设置自定义元数据name的值为my-data
            meta.UserMetadata["name"] = "my-data";
            // 上传Object并打印ETag
            putObjectResponseFromString = client.PutObject(bucketName, objectNameString, "sampledata", meta);
            Console.WriteLine(putObjectResponseFromString.ETAG);
        }

        private static BosClient GenerateBosClient()
        {
            const string accessKeyId = <AccessKeyID>; // 您的Access Key ID
            const string secretAccessKey = <SecretAccessKey>; // 您的Secret Access Key
            const string endpoint = "https://bj.bcebos.com"; // 指定BOS服务域名

            // 初始化一个BosClient
            BceClientConfiguration config = new BceClientConfiguration();
            config.Credentials = new DefaultBceCredentials(accessKeyId, secretAccessKey);
            config.Endpoint = endpoint;

            return new BosClient(config);
        }
    }
}
```

查看Bucket中的Object

简单查询

查看Bucket中Object列表。

- 基本流程
    1. 创建一个BOSClient类的实例。
    2. 调用BOSClient.ListObjects()方法。调用该方法时你需要提供Bucket名称。
- 示例代码

```
public void ListObjects(BosClient client, string bucketName)
{

    // 获取指定Bucket下的所有Object信息
    ListObjectsResponse listObjectsResponse = client.ListObjects(bucketName);

    // 遍历所有Object
    foreach (BosObjectSummary objectSummary in listObjectsResponse.Contents)
    {
        Console.WriteLine("ObjectKey: " + objectSummary.Key);
    }

}
```

> **说明：** listObjects( )方法返回ListObjectsResponse对象，ListObjectsResponse对象包含了此次listObject请求的返回结果。用户可以通过ListObjectsResponse中的getContents方法获取所有Object的描述信息。
>
> - 默认情况下，如果Bucket中的Object数量大于1000，则只会返回1000个Object，并且返回结果中IsTruncated值为True，并返回NextMarker做为下次读取的起点。
> - 若想获取更多的Object，可以使用Marker参数分次读取，请参考扩展查询。

- 完整示例

  请参考完整示例。

扩展查询

用户可以通过设置ListObjectsRequest参数来完成更多扩展查询操作设置。ListObjectsRequest中可以设置的扩展参数如下：

参数名称	说明	默认值
MaxKeys	设定此次返回Object的最大个数，不可超过1000。	1000
Prefix	设定objectKey的前缀，前缀是指objectKey包含并以Prefix的值作为开始。 通常与Delimiter配合在查询模拟文件夹中使用。	-
Delimiter	是一个分隔符，用来对objectKey进行分层。 通常与Prefix配合在查询模拟文件夹中使用。 从Prefix开始到第一次出现Delimiter字符之间的objectKey称为：CommonPrefixes。	-
Marker	是一个字符串，用来设定返回结果的起始位置。 设定Marker值之后，返回的Object会从Marker值之后按字母排序开始返回。	-

- 基本流程

  1. 创建一个BOSClient类的实例。
  2. 您也可以构造一个ListObjectsRequest来调用ListObjects()，ListObjectsRequest等方法，实现更多的扩展查询操作。

- 示例代码

```
// 构造ListObjectsRequest请求
ListObjectsRequest listObjectsRequest = new ListObjectsRequest() {BucketName = bucketName};

// 设置参数
listObjectsRequest.Delimiter=<Delimiter>;
listObjectsRequest.Marker=<Marker>;

ListObjectsResponse listObjectsResponse = client.ListObjects(listObjectsRequest);
```

> **说明：**
>
> 上面代码中调用了 `listObjects` 中的一个重载方法，通过传入 `ListObjectsRequest` 来完成请求。

- 完整示例

  请参考完整示例。

查询模拟文件夹

由于BOS本身是一个(<Key>,<Value>)的存储系统，所以原则上并不会存在“文件夹”的概念，但您可以通过 Delimiter 和 Prefix 参数的配合进行文件夹功能模拟。

假设Bucket中有5个文件：bos.jpg，fun/，fun/test.jpg，fun/movie/001.avi，fun/movie/007.avi，可以把 “/” 符号作为分隔符模拟文件夹。

递归列出模拟文件夹下所有文件

可以通过设置 `Prefix` 参数来获取某个目录下所有的文件：

```
// 构造ListObjectsRequest请求
ListObjectsRequest listObjectsRequest = new ListObjectsRequest() {BucketName = bucketName};

// 递归列出fun目录下的所有文件
listObjectsRequest.Prefix = "fun/";

// List Objects
ListObjectsResponse listObjectsResponse = client.ListObjects(listObjectsRequest);

// 遍历所有Object
Console.WriteLine("Objects:");
foreach (BosObjectSummary objectSummary in listObjectsResponse.Contents)
{
    Console.WriteLine("ObjectKey: " + objectSummary.Key);
}
```

输出：

```
 Objects:
fun/
fun/movie/001.avi
fun/movie/007.avi
fun/test.jpg
```

查看模拟文件夹下的文件和子文件夹

在 `Prefix` 和 `Delimiter` 结合的情况下，可以列出模拟文件夹下的文件和子文件夹：

```
 // 构造ListObjectsRequest请求
ListObjectsRequest listObjectsRequest = new ListObjectsRequest() {BucketName = bucketName};

// "/" 为文件夹的分隔符
listObjectsRequest.Delimiter = "/";

// 列出fun目录下的所有文件和文件夹
listObjectsRequest.Prefix = "fun/";

// List Objects
ListObjectsResponse listObjectsResponse = client.ListObjects(listObjectsRequest);

// 遍历所有Object，相当于获取fun目录下的所有文件
Console.WriteLine("Objects:");
foreach (BosObjectSummary objectSummary in listObjectsResponse.Contents)
{
    Console.WriteLine("ObjectKey: " + objectSummary.Key);
}

// 遍历所有CommonPrefix，相当于获取fun目录下的所有文件夹
Console.WriteLine("\nCommonPrefixs:");
foreach (ObjectPrefix objectPrefix in listObjectsResponse.CommonPrefixes)
{
    Console.WriteLine(objectPrefix.Prefix);
}
```

输出：

```
 Objects:
fun/
fun/test.jpg

CommonPrefixs:
fun/movie/
```

> **说明：**
>
> 返回的结果中， `Objects` 的列表中给出的是fun文件夹下的文件。而 `CommonPrefixs` 的列表中给出的是fun文件夹下的所有子文件夹。可以看出 `fun/movie/001.avi` ， `fun/movie/007.avi` 两个文件并没有被列出来，因为它们属于 `fun` 文件夹下的 `movie` 子文件夹下的文件。

- 完整示例

  请参考完整示例。

完整示例

下面示例代码演示了Object的简单查询、使用nextmarker分次查询、扩展查询、查询模拟文件夹的完整过程：

```
 using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using BaiduBce;
using BaiduBce.Auth;
using BaiduBce.Services.Bos;
using BaiduBce.Services.Bos.Model;

namespace DotnetSample
{
    internal class ListObjectsSample
    {
        private static void Main(string[] args)
        {
            BosClient client = GenerateBosClient();
            const string bucketName = <BucketName>;     //您的Bucket名称

            // 新建一个Bucket
            client.CreateBucket(bucketName);

            // 创建bos.jpg,fun/,fun/test.jpg,fun/movie/001.avi,fun/movie/007.avi五个文件
            client.PutObject(bucketName, "bos.jpg", "sampledata");
            client.PutObject(bucketName, "fun/", "sampledata");
            client.PutObject(bucketName, "fun/test.jpg", "sampledata");
            client.PutObject(bucketName, "fun/movie/001.avi", "sampledata");
            client.PutObject(bucketName, "fun/movie/007.avi", "sampledata");

            // 构造ListObjectsRequest请求
            ListObjectsRequest listObjectsRequest = new ListObjectsRequest() { BucketName = bucketName };

            // 1. 简单查询，列出Bucket下所有文件
            ListObjectsResponse listObjectsResponse = client.ListObjects(listObjectsRequest);

            // 输出：
            // Objects:
            // bos.jpg
            // fun/
```

```
            // fun/
            // fun/movie/001.avi
            // fun/movie/007.avi
            // fun/test.jpg

            Console.WriteLine("Objects:");
            foreach (BosObjectSummary objectSummary in listObjectsResponse.Contents)
            {
                Console.WriteLine("ObjectKey: " + objectSummary.Key);
            }

            // 2. 使用NextMarker分次列出所有文件
            listObjectsRequest.MaxKeys = 2;
            listObjectsResponse = client.ListObjects(listObjectsRequest);

            // 输出：
            // Objects:
            // bos.jpg
            // fun/
            // fun/movie/001.avi
            // fun/movie/007.avi
            // fun/test.jpg
            Console.WriteLine("Objects:");
            while (listObjectsResponse.IsTruncated)
            {
                foreach (BosObjectSummary objectSummary in listObjectsResponse.Contents)
                {
                    Console.WriteLine("ObjectKey: " + objectSummary.Key);
                }
                listObjectsResponse = client.ListNextBatchOfObjects(listObjectsResponse);
            }
            foreach (BosObjectSummary objectSummary in listObjectsResponse.Contents)
            {
                Console.WriteLine("ObjectKey: " + objectSummary.Key);
            }

            // 3. 递归列出“fun/”下所有文件和子文件夹
            listObjectsRequest.Prefix = "fun/";
            listObjectsResponse = client.ListObjects(listObjectsRequest);

            // 输出：
            // Objects:
            // fun/
            // fun/movie/001.avi
            // fun/movie/007.avi
            // fun/test.jpg

            Console.WriteLine("Objects:");
            foreach (BosObjectSummary objectSummary in listObjectsResponse.Contents)
            {
                Console.WriteLine("ObjectKey: " + objectSummary.Key);
            }

            // 4. 列出“fun”下的文件和子文件夹
            listObjectsRequest.Delimiter = "/";
            listObjectsResponse = client.ListObjects(listObjectsRequest);

            // 输出：
            // Objects:
            // fun/
            // fun/test.jpg

            Console.WriteLine("Objects:");
            foreach (BosObjectSummary objectSummary in listObjectsResponse.Contents)
            {
                Console.WriteLine("ObjectKey: " + objectSummary.Key);
            }

            // 遍历所有CommonPrefix，相当于获取fun目录下的所有子文件夹
            // 输出：
            // CommonPrefixs:
            // fun/movie

            Console.WriteLine("\nCommonPrefixs:");
            foreach (ObjectPrefix objectPrefix in listObjectsResponse.CommonPrefixes)
            {
                Console.WriteLine(objectPrefix.Prefix);
            }
        }

        private static BosClient GenerateBosClient()
        {
            const string accessKeyId = <AccessKeyID>; // 您的Access Key ID
            const string secretAccessKey = <SecretAccessKey>; // 您的Secret Access Key
            const string endpoint = "https://bj.bcebos.com"; // 指定BOS服务域名

            // 初始化一个BosClient
            BceClientConfiguration config = new BceClientConfiguration();
            config.Credentials = new DefaultBceCredentials(accessKeyId, secretAccessKey);
            config.Endpoint = endpoint;

            return new BosClient(config);
        }
    }
}
```

获取Object

简单的获取Object

- 基本流程

  1. 创建BosClient类的实例。
  2. 执行BosClient.GetObject()方法，会返回指定的BosObject。
  3. 将BosObject读到流中，然后对流进行操作。

- 示例代码

```
public void GetObject(BosClient client, String bucketName, String objectKey)
{

    // 获取Object，返回结果为BosObject对象
    BosObject bosObject = client.GetObject(bucketName, objectKey);

    // 获取ObjectMeta
    ObjectMetadata meta = bosObject.ObjectMetadata;

    // 获取Object的输入流
    Stream objectContent = bosObject.ObjectContent;

    // 处理Object
    ...

    // 关闭流
    objectContent.Close();
}
```

> **注意：**
>
> - BosObject中包含了Object的各种信息，包含Object所在的Bucket、Object的名称、MetaData以及一个输入流，
> - 用户可以通过操作输入流将Object的内容读取到文件或者内存中。
> - ObjectMetadata中包含了Object上传时定义的ETag，Http Header以及自定义的元数据。
> - 通过BosObject的getObjectContent()方法，还可以获取返回Object的输入流，用户可以读取这个输入流来对Object的内容进行操作。

- 完整示例

  请参考完整示例。

通过GetObjectRequest获取Object

为了实现更多的功能，可以通过使用GetObjectRequest来获取Object。

- 基本流程

  1. 创建GetObjectRequest类的实例。
  2. 对GetObjectRequest执行setRange( )，获取Object中所需的字节数据。
  3. 执行GetObject( )操作。

- 示例代码

```
// 新建GetObjectRequest
GetObjectRequest getObjectRequest = new GetObjectRequest() {BucketName = bucketName, Key = objectKey};

// 获取0~100字节范围内的数据
getObjectRequest.SetRange(0, 100);

// 获取Object，返回结果为BosObject对象
BosObject bosObject = client.GetObject(getObjectRequest);
```

> **说明：**通过getObjectRequest的setRange( )方法可以设置返回Object的范围。用户可以用此功能实现文件的分段下载和断点续传。

- 完整示例

  请参考完整示例。

直接下载Object到指定路径

用户可以通过如下代码直接将Object下载到指定路径。

- 基本流程

  1. 创建GetObjectRequest类的实例。
  2. 执行client.getObject( )操作。
  3. Object可以直接下载到指定路径。

- 示例代码

```
// 新建GetObjectRequest
GetObjectRequest getObjectRequest = new GetObjectRequest() {BucketName = bucketName, Key = objectKey};

// 下载Object到文件
ObjectMetadata objectMetadata = client.GetObject(getObjectRequest, new FileInfo("d:\\sample.txt"));
```

> **说明：**当使用上面方法将Object直接下载到指定路径时，方法会返回ObjectMetadata对象。

- 完整示例

请参考完整示例。

只获取ObjectMetadata

通过 getObjectMetadata() 方法可以只获取ObjectMetadata而不获取Object的实体。

- 示例代码

```
ObjectMetadata objectMetadata = client.GetObjectMetadata(bucketName, objectKey);
```

- 完整示例

请参考完整示例。

获取Object的URL

您可以通过如下代码获取指定Object的URL，该功能通常用于您将Object的URL临时分享给其他用户的场景。

- 基本流程

1. 创建一个BosClient类的实例。
2. 调用BosClient.GeneratePresignedUrl()方法，在调用该方法时您需要提供Bucket名称，object名称，过期时间等参数。
3. 返回一个Object的URL。

- 示例代码

```
public string GeneratePresignedUrl(BosClient client, string bucketName, string objectKey,
        int expirationInSeconds)
{
  //指定用户需要获取的Object所在的Bucket名称、该Object名称、时间戳、URL的有效时长
  Uri url = client.GeneratePresignedUrl(bucketName, objectKey, expirationInSeconds);
  return url.AbsoluteUri;
}
```

> **说明：**ExpirationInSeconds为指定的URL有效时长，时间从当前时间算起，为可选参数，不配置时系统默认值为1800秒。如果要设置为永久不失效的时间，可以将ExpirationInSeconds参数设置为 -1，不可设置为其他负数。

- 完整示例

请参考完整示例。

完整示例

下面示例代码演示了简单获取Object、通过GetObjectRequest获取Object、直接下载Object到指定路径、只获取ObjectMetadata以及获取Object的URL的完整过程：

```
 using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Text;
using BaiduBce;
using BaiduBce.Auth;
using BaiduBce.Services.Bos;
using BaiduBce.Services.Bos.Model;

namespace DotnetSample
{
    internal class GetObjectsSample
    {
        private static void Main(string[] args)
        {
            BosClient client = GenerateBosClient();
            const string bucketName = <BucketName>; //您的Bucket名称

            // 初始化，创建Bucket和Object
            client.CreateBucket(bucketName);
            string objectName = <ObjectKey>;
            client.PutObject(bucketName, objectName, <SampleData>);

            // 获取BosObject对象并通过BosObject的输入流获取内容
            BosObject bosObject = client.GetObject(bucketName, objectName);
            Stream objectContent = bosObject.ObjectContent;
            string content = new StreamReader(objectContent).ReadToEnd();
            Console.WriteLine(content); // 您传入的<SampleData>

            // 通过GetObjectRequest只获取部分数据
            GetObjectRequest getObjectRequest = new GetObjectRequest() {BucketName = bucketName, Key = objectName};
            getObjectRequest.SetRange(0, 5);
            bosObject = client.GetObject(getObjectRequest);
            objectContent = bosObject.ObjectContent;
            content = new StreamReader(objectContent).ReadToEnd();
            Console.WriteLine(content); // 您传入的<SampleData>

            // 直接通过GetObjectContent获取byte[]内容
            byte[] bytes = client.GetObjectContent(bucketName, objectName);
            content = Encoding.Default.GetString(bytes);
            Console.WriteLine(content); // 您传入的<SampleData>

            // 将Object内容下载到文件
            FileInfo fileInfo = new FileInfo("my file path and name");
            client.GetObject(bucketName, objectName,fileInfo );
            content = File.ReadAllText(fileInfo.FullName);
            Console.WriteLine(content); // 您传入的<SampleData>

            // 只获取object的meta，不获取内容
            ObjectMetadata objectMetadata = client.GetObjectMetadata(bucketName, objectName);
            Console.WriteLine(objectMetadata.ContentLength);

            Console.ReadKey();

            // 生成url，并通过该url直接下载和打印对象内容
            string url = client.GeneratePresignedUrl(bucketName, objectName, 60).AbsoluteUri;
            using (WebClient webClient = new WebClient())
            {
                using (Stream stream = webClient.OpenRead(url))
                using (StreamReader streamReader = new StreamReader(stream))
                {
                    string response = streamReader.ReadToEnd();
                    Console.WriteLine(response);  // 您传入的<SampleData>
                }
            }
        }

        private static BosClient GenerateBosClient()
        {
            const string accessKeyId = <AccessKeyID>; // 您的Access Key ID
            const string secretAccessKey = <SecretAccessKey>; // 您的Secret Access Key
            const string endpoint = "https://bj.bcebos.com"; // 指定BOS服务域名

            // 初始化一个BosClient
            BceClientConfiguration config = new BceClientConfiguration();
            config.Credentials = new DefaultBceCredentials(accessKeyId, secretAccessKey);
            config.Endpoint = endpoint;

            return new BosClient(config);
        }
    }
}
```

删除Object

- 基本流程
    1. 创建一个BosClient类的实例。
    2. 调用BosClient.DeleteObject()方法。在调用该方法时你需要提供Bucket名称，Object名称等参数。
- 示例代码

```
public void DeleteObject(BosClient client, string bucketName, string objectKey)
{
// 删除Object
client.DeleteObject(bucketName, objectKey);
}
```

- 完整示例

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Net;
using System.Text;
using BaiduBce;
using BaiduBce.Auth;
using BaiduBce.Services.Bos;
using BaiduBce.Services.Bos.Model;

namespace DotnetSample
{
    internal class DeleteObjectSample
    {
        private static void Main(string[] args)
        {
            BosClient client = GenerateBosClient();
            const string bucketName = <BucketName>;     //指定Bucket名称

            // 初始化：创建示例Bucket和Object
            client.CreateBucket(bucketName);
            string objectName = <ObjectKey>;
            client.PutObject(bucketName, objectName, <Sampledata>);

            // 删除Object
            client.DeleteObject(bucketName, objectName);
        }

        private static BosClient GenerateBosClient()
        {
            const string accessKeyId = <AccessKeyID>; // 您的Access Key ID
            const string secretAccessKey = <SecretAccessKey>; // 您的Secret Access Key
            const string endpoint = "https://bj.bcebos.com"; // 指定BOS服务域名

            // 初始化一个BosClient
            BceClientConfiguration config = new BceClientConfiguration();
            config.Credentials = new DefaultBceCredentials(accessKeyId, secretAccessKey);
            config.Endpoint = endpoint;

            return new BosClient(config);
        }
    }
}
```

拷贝Object

简单拷贝Object

- 基本流程

  1. 创建BosClient 类的实例。
  2. 执行BosClient.copyObject( )方法。

- 示例代码

```
public void CopyObject(BosClient client, String srcBucketName, String srcKey, String destBucketName,
String destKey)
{
  // 拷贝Object
  CopyObjectResponse copyObjectResponse = client.CopyObject(srcBucketName, srcKey, destBucketName, destKey);

  // 打印结果
  Console.WriteLine("ETag: " + copyObjectResponse.ETag + " LastModified: " + copyObjectResponse.LastModified);
}
```

> **说明：**copyObject 方法返回一个 `CopyObjectResponse` 对象，该对象中包含了新Object的ETag和修改时间。

- 完整示例

  请参考完整示例。

通过CopyObjectRequest拷贝Object

您也可以通过 `CopyObjectRequest` 实现Object的拷贝。该功能一般用于如下场景：

- Copy一个Object但重新设置meta。
- 重置某个现有Object的meta（把Src和Des设置为同一个Object）。
- 基本流程

  1. 创建CopyObjectRequest类的实例，传入`<SrcBucketName>`，`<SrcKey>`，`<DestBucketName>`，`<DestKey>`参数。
  2. 创建ObjectMetadata类的实例。

- 示例代码

```
 // 初始化BosClient
BosClient client = ...;

// 创建CopyObjectRequest对象
CopyObjectRequest copyObjectRequest = new CopyObjectRequest()
{
    SourceBucketName = srcBucketName,
    SourceKey = srcKey,
    BucketName = destBucketName,
    Key = destKey
};

// 设置新的Metadata
Dictionary<String, String> userMetadata = new Dictionary<String, String>();
userMetadata["usermetakey"] = "usermetavalue";
ObjectMetadata objectMetadata = new ObjectMetadata()
{
    UserMetadata = userMetadata
};
copyObjectRequest.NewObjectMetadata = objectMetadata;

// 复制Object并打印新的ETag
CopyObjectResponse copyObjectResponse = client.CopyObject(copyObjectRequest);
Console.WriteLine("ETag: " + copyObjectResponse.ETag + " LastModified: " + copyObjectResponse.LastModified);
```

> **说明：** CopyObjectRequest 允许用户修改目的Object的ObjectMeta，同时也提供 MatchingETagConstraints 参数的设定。

- 完整示例

  请参考完整示例。

完整示例

下面示例代码演示了简单拷贝Object和通过CopyObjectRequest拷贝Object的完整过程：

```
 using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Net;
using System.Text;
using BaiduBce;
using BaiduBce.Auth;
using BaiduBce.Services.Bos;
using BaiduBce.Services.Bos.Model;

namespace DotnetSample
{
    internal class CopyObjectSample
    {
        private static void Main(string[] args)
        {
            BosClient client = GenerateBosClient();
            const string bucketName = <SrcBucketName>;

            // 初始化，创建示例Bucket和Object
            client.CreateBucket(bucketName);
            string objectName = <SrcObjectKey>;
            client.PutObject(bucketName, objectName, <SrcSampleData>);

            // 普通拷贝并打印结果
            string newObjectName = <DesObjectKey>;
            CopyObjectResponse copyObjectResponse = client.CopyObject(bucketName, objectName, bucketName,
                newObjectName);

            Console.WriteLine(Encoding.Default.GetString(client.GetObjectContent(bucketName, newObjectName)));

            // 拷贝并设置新的meta
            newObjectName = <DesObjectKey>;
            CopyObjectRequest copyObjectRequest = new CopyObjectRequest()
            {
                SourceBucketName = bucketName,
                SourceKey = objectName,
                BucketName = bucketName,
                Key = newObjectName
            };
            Dictionary<String, String> userMetadata = new Dictionary<String, String>();
            userMetadata["usermetakey"] = "usermetavalue";
            ObjectMetadata objectMetadata = new ObjectMetadata()
            {
                UserMetadata = userMetadata
            };
            copyObjectRequest.NewObjectMetadata = objectMetadata;
            client.CopyObject(copyObjectRequest);
            // usermetavalue
            Console.WriteLine(client.GetObjectMetadata(bucketName, newObjectName).UserMetadata["usermetakey"]);
        }

        private static BosClient GenerateBosClient()
        {
            const string accessKeyId = <AccessKeyID>; // 您的Access Key ID
            const string secretAccessKey = <SecretAccessKey>; // 您的Secret Access Key
            const string endpoint = "https://bj.bcebos.com"; // 指定BOS服务域名

            // 初始化一个BosClient
            BceClientConfiguration config = new BceClientConfiguration();
            config.Credentials = new DefaultBceCredentials(accessKeyId, secretAccessKey);
            config.Endpoint = endpoint;

            return new BosClient(config);
        }
    }
}
```

Object的分块上传

除了通过putObject()方法上传文件到BOS以外，BOS还提供了另外一种上传模式：分块上传（Multipart Upload）。用户可以在如下的应用场景内（但不仅限于此），使用分块上传模式，如：

- 需要支持断点上传。
- 上传超过5GB大小的文件。
- 网络条件较差，和BOS的服务器之间的连接经常断开。
- 需要流式地上传文件。
- 上传文件之前，无法确定上传文件的大小。

分块完成Multipart Upload

假设有一个文件，本地路径为 `d:\\sample.txt` ，由于文件比较大，将其分块传输到BOS中。

- 基本流程

  1. 创建一个BosClient类的实例。
  2. 调用BosClient.InitiateMultipartUpload()方法。在调用该方法时您需要提供Bucket名称和Object名称，该方法会返回一个UploadId，在后面的步骤中需要用到该Id。
  3. 多次调用调用BosClient.UploadPart方法将文件分块多次上传。调用该方法时您需要提供Bucket名称，Object名称，UploadId，分块序号，分块大小，分块内容等参数，每次调用该方法都会返回本次分块的序号和ETag，在后面的步骤中会用到。
  4. 调用BosClient.CompleteMultipartUpload()方法完成本次分块上传。调用该方法您需要提供Bucket名称，Object名称，UploadId以及每个分块的序号和ETag。
  5. 上传过程中可以通过BosClient.ListParts()方法获取指定UploadId中已经上传的所有块；还可以通过BosClient.ListMultipartUploads()方法获取指定Bucket中所有未完成的UploadId。

初始化Multipart Upload

使用 InitiateMultipartUpload 方法来初始化一个分块上传事件：

- 示例代码

```
 // 开始Multipart Upload
InitiateMultipartUploadRequest initiateMultipartUploadRequest =
        new InitiateMultipartUploadRequest() {BucketName = bucketName, Key = objectKey};
InitiateMultipartUploadResponse initiateMultipartUploadResponse =
        client.InitiateMultipartUpload(initiateMultipartUploadRequest);

// 打印UploadId
Console.WriteLine("UploadId: " + initiateMultipartUploadResponse.UploadId);
```

> **说明：** initiateMultipartUpload 的返回结果中含有 UploadId ，它是区分分块上传事件的唯一标识，在后面的操作中，我们将用到它。

上传分块

把文件分块上传。

- 示例代码

```
 // 设置每块为 5Mb
long partSize = 1024 * 1024 * 5L;

FileInfo partFile = new FileInfo("my file");

// 计算分块数目
int partCount = (int) (partFile.Length / partSize);
if (partFile.Length % partSize != 0)
{
  partCount++;
}

// 新建一个List保存每个分块上传后的ETag和PartNumber
List<PartETag> partETags = new List<PartETag>();

for (int i = 0; i < partCount; i++)
{
  // 获取文件流
  Stream stream = partFile.OpenRead();

  // 跳到每个分块的开头
  long skipBytes = partSize * i;
  stream.Seek(skipBytes, SeekOrigin.Begin);

  // 计算每个分块的大小
  long size = Math.Min(partSize, partFile.Length - skipBytes);

  // 创建UploadPartRequest，上传分块
  uploadPartRequest uploadPartRequest = new UploadPartRequest();
  uploadPartRequest.BucketName = bucketName;
  uploadPartRequest.Key = objectKey;
  uploadPartRequest.UploadId = initiateMultipartUploadResponse.UploadId;
  uploadPartRequest.InputStream = stream;
  uploadPartRequest.PartSize = size;
  uploadPartRequest.PartNumber = i + 1;
  UploadPartResponse uploadPartResponse = client.UploadPart(uploadPartRequest);

  // 将返回的PartETag保存到List中。
  partETags.Add(new PartETag()
  {
    ETag = uploadPartResponse.ETag,
    PartNumber = uploadPartResponse.PartNumber
   });

  // 关闭文件
  stream.Close();
  }
```

> **注意：**
>
> 上面代码的核心是调用 UploadPart 方法来上传每一个分块，但是要注意以下几点：
>
> - 分块上传方法要求除最后一个Part以外，其他的Part大小都要大于等于5MB。但是Upload Part接口并不会检查上传Part的大小；只有当Complete Multipart Upload的时候才会校验。
> - 为了保证数据在网络传输过程中不出现错误，建议您在UploadPart后，使用每个分块BOS返回的Content-MD5值分别验证已上传分块数据的正确性。当所有分块数据合成一个Object后，不再含MD5值。
> - Part号码的范围是1~10000。如果超出这个范围，BOS将返回InvalidArgument的错误码。
> - 每次上传Part时都要把流定位到此次上传块开头所对应的位置。
> - 每次上传Part之后，BOS的返回结果会包含一个 PartETag 对象，它是上传块的ETag与块编号（PartNumber）的组合，在后续完成分块上传的步骤中会用到它，因此需要将其保存起来。一般来讲这些 PartETag 对象将被保存到List中。

完成分块上传

- 示例代码

```
CompleteMultipartUploadRequest completeMultipartUploadRequest =
new CompleteMultipartUploadRequest()
{
  BucketName = bucketName,
  Key = objectKey,
  UploadId = initiateMultipartUploadResponse.UploadId,
  PartETags = partETags
};

// 完成分块上传
CompleteMultipartUploadResponse completeMultipartUploadResponse =
client.CompleteMultipartUpload(completeMultipartUploadRequest);

// 打印Object的ETag
Console.WriteLine(completeMultipartUploadResponse.ETag);
```

> **说明：**上面代码中的 `partETags` 是第二部中保存的partETag的列表，BOS收到用户提交的Part列表后，会逐一验证每个数据Part的有效性。当所有的数据Part验证通过后，BOS将把这些数据part组合成一个完整的Object。

取消分块上传事件

用户可以使用abortMultipartUpload方法取消分块上传。

- 示例代码

```
AbortMultipartUploadRequest abortMultipartUploadRequest = new AbortMultipartUploadRequest()
{
  BucketName = bucketName,
  Key = objectKey,
  UploadId = initiateMultipartUploadResponse.UploadId,
};

// 取消分块上传
client.AbortMultipartUpload(abortMultipartUploadRequest);
```

获取未完成的分块上传事件

用户可以使用 `ListMultipartUploads` 方法获取Bucket内未完成的分块上传事件。

- 示例代码

```
ListMultipartUploadsRequest listMultipartUploadsRequest =
  new ListMultipartUploadsRequest() {BucketName = bucketName};

// 获取Bucket内所有上传事件
ListMultipartUploadsResponse listMultipartUploadsResponse =
  client.ListMultipartUploads(listMultipartUploadsRequest);

// 遍历所有上传事件
foreach (MultipartUploadSummary multipartUpload in listMultipartUploadsResponse.Uploads)
{
  Console.WriteLine("Key: " + multipartUpload.Key + " UploadId: " + multipartUpload.UploadId);
}
```

> **注意：**
>
> - 默认情况下，如果Bucket中的分块上传事件的数目大于1000，则只会返回1000个，并且返回结果中IsTruncated的值为True，同时返回NextKeyMarker作为下次读取的起点。
> - 若想获取更多分块上传事件，可以使用KeyMarker参数分次读取。

获取所有已上传的块信息

用户可以使用 `ListParts` 方法获取某个上传事件中所有已上传的块。

- 示例代码

```
ListPartsRequest listPartsRequest = new ListPartsRequest()
{
  BucketName = bucketName,
  Key = objectKey,
  UploadId = initiateMultipartUploadResponse.UploadId,
};

// 获取上传的所有Part信息
ListPartsResponse listPartsResponse = client.ListParts(listPartsRequest);

// 遍历所有Part
foreach (PartSummary part in listPartsResponse.Parts)
{
  Console.WriteLine("PartNumber: " + part.PartNumber + " ETag: " + part.ETag);
}
```

> **注意：**
>
> - 默认情况下，如果Bucket中的分块上传事件的数目大于1000，则只会返回1000个part，并且返回结果中IsTruncated的值为True，同时返回NextPartNumberMarker作为下次读取的起点。
> - 若想获取更多已上传的分块信息，可以使用PartNumberMarker参数分次读取。

完整示例

下面示例代码演示了分块上传的完整过程：

```
 using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Text;
using BaiduBce;
using BaiduBce.Auth;
using BaiduBce.Services.Bos;
using BaiduBce.Services.Bos.Model;

namespace DotnetSample
{
    internal class MultiUploadSample
    {
        private static void Main(string[] args)
        {
            BosClient client = GenerateBosClient();
            const string bucketName = <BucketName>;    //Bucket名称

            // 初始化：创建示例Bucket
            client.CreateBucket(bucketName);
            string objectKey = <ObjectKey>;

            // 开始Multipart Upload
            InitiateMultipartUploadRequest initiateMultipartUploadRequest =
                new InitiateMultipartUploadRequest() { BucketName = bucketName, Key = objectKey };
            InitiateMultipartUploadResponse initiateMultipartUploadResponse =
                client.InitiateMultipartUpload(initiateMultipartUploadRequest);

            // 获取Bucket内的Multipart Upload
            ListMultipartUploadsRequest listMultipartUploadsRequest =
                new ListMultipartUploadsRequest() { BucketName = bucketName };
            ListMultipartUploadsResponse listMultipartUploadsResponse =
                client.ListMultipartUploads(listMultipartUploadsRequest);
            foreach (MultipartUploadSummary multipartUpload in listMultipartUploadsResponse.Uploads)
            {
                Console.WriteLine("Key: " + multipartUpload.Key + " UploadId: " + multipartUpload.UploadId);
            }

            // 分块上传，首先设置每块为 5Mb
            long partSize = 1024 * 1024 * 5L;
            FileInfo partFile = new FileInfo("d:\\lzb\\sample");

            // 计算分块数目
            int partCount = (int)(partFile.Length / partSize);
            if (partFile.Length % partSize != 0)
            {
                partCount++;
            }

            // 新建一个List保存每个分块上传后的ETag和PartNumber
            List<PartETag> partETags = new List<PartETag>();
            for (int i = 0; i < partCount; i++)
            {
                // 获取文件流
                Stream stream = partFile.OpenRead();

                // 跳到每个分块的开头
                long skipBytes = partSize * i;
                stream.Seek(skipBytes, SeekOrigin.Begin);

                // 计算每个分块的大小
                long size = Math.Min(partSize, partFile.Length - skipBytes);

                // 创建UploadPartRequest，上传分块
                UploadPartRequest uploadPartRequest = new UploadPartRequest();
                uploadPartRequest.BucketName = bucketName;
                uploadPartRequest.Key = objectKey;
                uploadPartRequest.UploadId = initiateMultipartUploadResponse.UploadId;
                uploadPartRequest.InputStream = stream;
                uploadPartRequest.PartSize = size;
                uploadPartRequest.PartNumber = i + 1;
                UploadPartResponse uploadPartResponse = client.UploadPart(uploadPartRequest);

                // 将返回的PartETag保存到List中。
                partETags.Add(new PartETag()
                {
                    ETag = uploadPartResponse.ETag,
                    PartNumber = uploadPartResponse.PartNumber
                });

                // 关闭文件
                stream.Close();
            }

            // 获取UploadId的所有Upload Part
            ListPartsRequest listPartsRequest = new ListPartsRequest()
            {
                BucketName = bucketName,
                Key = objectKey,
                UploadId = initiateMultipartUploadResponse.UploadId,
            };

            // 获取上传的所有Part信息
            ListPartsResponse listPartsResponse = client.ListParts(listPartsRequest);

            // 遍历所有Part
            foreach (PartSummary part in listPartsResponse.Parts)
            {
```

```
            Console.WriteLine("PartNumber: " + part.PartNumber + " ETag: " + part.ETag);
        }

        // 完成分块上传
        CompleteMultipartUploadRequest completeMultipartUploadRequest =
            new CompleteMultipartUploadRequest()
            {
                BucketName = bucketName,
                Key = objectKey,
                UploadId = initiateMultipartUploadResponse.UploadId,
                PartETags = partETags
            };
        CompleteMultipartUploadResponse completeMultipartUploadResponse =
            client.CompleteMultipartUpload(completeMultipartUploadRequest);
        Console.WriteLine(completeMultipartUploadResponse.ETag);
        Console.ReadKey();
    }

    private static BosClient GenerateBosClient()
    {
        const string accessKeyId = <AccessKeyID>; // 您的Access Key ID
        const string secretAccessKey = <SecretAccessKey>; // 您的Secret Access Key
        const string endpoint = "https://bj.bcebos.com"; // 指定BOS服务域名

        // 初始化一个BosClient
        BceClientConfiguration config = new BceClientConfiguration();
        config.Credentials = new DefaultBceCredentials(accessKeyId, secretAccessKey);
        config.Endpoint = endpoint;

        return new BosClient(config);
    }
  }
}
```

支持单链接限速

对象存储BOS对单Bucket的公网带宽阈值为10Gbit/s，内网带宽阈值为50Gbit/s，当用户的上传或下载占用带宽达到带宽限制阈值时，会返回RequestRateLimitExceeded的错误码。为保证用户能够正常使用服务，BOS支持在进行上传、下载等行为时进行流量控制，保证大流量服务占用带宽不会对其他应用服务造成影响。

单链接限速上传和下载接口示例

限速值的取值范围为819200~838860800，单位为bit/s，即100KB/s~100MB/s。限速值取值必须为数字，BOS将按照指定的限速值对此次请求进行限速，当限速值不在此范围或不合法时将返回400错误码。

```
 public void ObjectTrafficLimit(BosClient client, String bucketName, String objectKey, byte[] byte1, String string1)
{
    // 具体的限速值
    long trafficLimit = 819200;

    // 获取指定文件
    FileInfo file = new FileInfo(<FilePath>);    //指定文件路径

    // 生成put请求
    PutObjectRequest putRequest = new PutObjectRequest()
    {
        BucketName = bucketName,
        Key = objectKey,
        FileInfo = file,
        TrafficLimit = trafficLimit
    };

    // 以文件形式上传Object
    PutObjectResponse putObjectFromFileResponse = client.PutObject(putRequest);

    // 打印ETag
    Console.WriteLine(putObjectFromFileResponse.ETAG);

    // 生成get请求
    GetObjectRequest getRequest = new GetObjectRequest()
    {
        BucketName = bucketName,
        Key = objectKey,
        TrafficLimit = trafficLimit
    };

    // 获取object
    BosObject bosObject = client.GetObject(getRequest);

    // 打印ETag
    Console.WriteLine(bosObject.ObjectMetadata.ETag);
}
```

异常处理

BOS异常提示有如下两种方式：

异常方法	说明
BceClientException	客户端异常
BceServiceException	服务器异常

用户可以使用try获取某个事件所产生的异常，例如：

```
 String objectKey = "testObjectToFile";
bosclient.putObject(bucketName, objectKey, "dataFile");
File file = new File("test");

try{
    bosclient.getObject(this.bucketName, objectKey, file);
  } catch (BceServiceException bce){
    System.out.println(bce.getMessage());
  } catch ( BceClientException bce){
    System.out.println( bce.getMessage());
  } finally {
    file.delete();
  }
```

版本变更记录

- C# SDK开发包[2024-03-01] 版本号 1.0.6
  - 支持Virtual Host四级域名访问BOS
  - 支持单链接限速
- C# SDK开发包[2019-04-23] 版本号 1.0.4
  - 升级Newtonsoft.Json到11.0.0
- C# SDK开发包[2018-02-08] 版本号 1.0.3
  - 修复STS鉴权失败
- C# SDK开发包[2016-07-28] 版本号 1.0.2
  - 修复GeneratePresignedUrl当路径有中文时不能下载的问题
  - 优化CopyObject的默认设置
- C# SDK开发包[2015-12-22] 版本号 1.0.1

  首次发布：
  - 支持创建、查看、罗列、删除 Bucket
  - 修改、获取Bucket的访问权限
  - 上传、查看、罗列、删除Object

C-SDK

概述

本文档主要介绍BOS C SDK的安装和使用。在使用本文档前，您需要先了解BOS的一些基本知识，并已经开通了BOS服务。若您还不了解BOS，可以参考产品描述。

安装sdk工具包

运行环境

依赖开源的第三方库，包括libcurl-devel、apr、jansson，其中jansson可在thirdlib文件夹下编译并安装。

SDK目录结构

```
 ├──MakeList.txt
├──bos_c_sdk.sln          // windows viusal studio相关
├──util
├──bos_c_sdk              // 源码路径
|   ├──auth
|   ├──bce
|   ├──http
|   ├──sts
|   ├──api
├──third_part // 第三方库路径
├──bos_c_sdk_test
├──bos_c_sdk_ut
```

安装SDK

用户可提交工单联系工作人员获取安装包。

Linux Centos用户安装指南

依赖公共库安装:

```
 yum -y install libcurl-devel
yum install apr-util
yum install apr-devel
yum install apr-util-devel
```

安装jasson库

a. 下载源码访问 Jansson 的官方网站，下载最新的稳定版本的源代码包。

b. 解压文件使用 tar 命令解压下载的压缩文件：其中version是下载的版本号。

```
 tar -xzf jansson-<version>.tar.gz
```

c. 在解压后的目录中执行以下命令，这将配置、编译并安装 Jansson 库到系统中。

```
 ./configure
make
sudo make install
```

环境构建

```
 tar zxf bos-c-sdk-1.0.0.tar.gz
cd bos-c-sdk-1.0.0
cmake .
make
sudo make install
```

如果需要指定安装目录

```
 cmake -f CMakeLists.txt
// 编译类型为Release。常用的编译类型为：Debug、Release、RelWithDebInfo和MinSizeRel，默认使用Debug。
-DCMAKE_BUILD_TYPE=Release
// 自定义安装目录。
-DCMAKE_INSTALL_PREFIX=/usr/local/
// 指定curl、apr、apr-util和xml第三方库头文件和库文件的所在目录。
-DCURL_INCLUDE_DIR=/usr/include/curl
-DCURL_LIBRARY=/usr/lib64/libcurl.so
-DAPR_INCLUDE_DIR=/usr/include/apr-1
-DAPR_LIBRARY=/usr/lib64/libapr-1.so
-DAPR_UTIL_INCLUDE_DIR=/usr/include/apr-1
-DAPR_UTIL_LIBRARY=/usr/lib64/libaprutil-1.so
-DMINIXML_INCLUDE_DIR=/usr/include
-DMINIXML_LIBRARY=/usr/lib64/libmxml.so
// 编译时报错Could not find apr-config/apr-1-config，原因是在默认路径里面找不到apr-1-config文件，请添加该选项。
-DAPR_CONFIG_BIN=/path/to/bin/apr-1-config
// 编译时报错Could not find apu-config/apu-1-config，原因是在默认路径里面找不到apu-1-config文件，请添加该选项。
-DAPU_CONFIG_BIN=/path/to/bin/apu-1-config
```

编译引用

1. 如果没有安装到gcc默认路径下，则在CXXFLAGS里使用-L指定lib路径，使用-I指定include路径；

2. 在LDFLAGS里指定-lbossdk。

对于编写好的程序进行编译，也可以直接指定-L -I -l。

```
 gcc -Wall -O -g -I. -I /usr/local/include/bos_c_sdk -I /usr/include/apr-1 -c main.c -o main.o
gcc main.o -o main -lpthread -L/usr/local/lib -lbos_c_sdk -lcurl -lmxml -L/usr/local/apr/lib -lapr-1 -laprutil-1 -ljansson
```

卸载SDK

预期卸载SDK时，删除output路径。

初始化

确定Endpoint

在确认您使用SDK时配置的Endpoint时，可先阅读开发人员指南中关于BOS访问域名的部分，理解Endpoint相关的概念。百度智能云目前开放了多区域支持，请参考区域选择说明。

region与endpoint参考如下链接

https://cloud.baidu.com/doc/BOS/s/akrqd2wcx

常见示例如下:

访问区域	对应Endpoint
BJ	bj.bcebos.com
BD	bd.bcebos.com
SU	su.bcebos.com
GZ	gz.bcebos.com
CD	cd.bcebos.com
HKG	hkg.bcebos.com
FWH	fwh.bcebos.com
FSH	fsh.bcebos.com

获取密钥

要使用百度智能云BOS，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BOS做签名验证。 可以通过如下步骤获得并了解您的AK/SK信息：

注册百度智能云账号

创建AK/SK

使用BOS C SDK的一般流程

下面为您介绍使用 C SDK 的一般流程。

1. 初始化 SDK。

2. 设置请求选项参数。

3. 设置 API 接口必需的参数。

4. 调用 SDK API 发起请求并获得请求响应结果。

初始化

```
int main(int argc, char *argv[])
{
    /* 程序入口处调用 bos_http_io_initialize 方法，这个方法内部会做一些全局资源的初始化，涉及网络，内存等部分 */
    if (bos_http_io_initialize(NULL, 0) != BOSE_OK) {
        exit(1);
    }


    /* 调用 BOS SDK 的接口上传或下载文件 */
    /* ... 用户逻辑代码，这里省略 */


    /* 程序结束前，调用 bos_http_io_deinitialize 方法释放之前分配的全局资源 */
    bos_http_io_deinitialize();
    return 0;
}
```

初始化请求选项

用户可以参考如下代码初始化请求选项：

```
    bos_pool_t *pool;
    bos_request_options_t *options;
    bos_pool_create(&pool, NULL);
    bos_string_t bucket;
    bos_string_t object;
    bos_string_t file;
    bos_acl_e bos_acl = 3;
    bos_table_t *resp_headers = NULL;
    bos_table_t *headers = NULL;
    options = bos_request_options_create(pool);
    options->config = bos_config_create(options->pool);
    bos_str_set(&options->config->endpoint, "bj.bcebos.com");
    bos_str_set(&options->config->access_key_id, "xxxxxxxxxx");
    bos_str_set(&options->config->access_key_secret, "xxxxxxxxxxxx");
  bos_str_set(&options->config->sts_token, "<用户的StsToken>");            //临时密钥的 Token
    bos_str_set(&bucket, TEST_BUCKET_NAME);
    options->config->is_cname = 0;
```

在上面代码中，`access_key_id`对应控制台中的“Access Key ID”，`access_key_secret`对应控制台中的“Access Key Secret”，获取方式请参考《操作指南 管理ACCESSKEY》。

上面的方式使用默认域名作为BOS的服务地址，如果用户需要自己指定域名，可以通过传入ENDPOINT参数来指定。

> **注意：**`ENDPOINT`参数只能用指定的包含区域的域名来进行定义，不指定时默认为北京区域`http://bj.bcebos.com`。

Bucket管理

概述

Bucket既是BOS上的命名空间，也是计费、权限控制、日志记录等高级功能的管理实体。

- Bucket名称在所有区域中具有全局唯一性，且不能修改。

> **说明：** 百度智能云目前开放了多区域支持，请参考区域选择说明。
> 目前支持“华北-北京”、“华南-广州”和“华东-苏州”三个区域。北京区域：`http://bj.bcebos.com`，广州区域：`http://gz.bcebos.com`，苏州区域：`http://su.bcebos.com`。

- 存储在BOS上的每个Object都必须包含在一个Bucket中。
- 一个用户最多可创建100个Bucket，但每个Bucket中存放的Object的数量和大小总和没有限制，用户不需要考虑数据的可扩展性。

新建bucket

新建Bucket

如下代码可以新建一个Bucket：

```
void test_put_bucket(bos_request_options_t *options, bos_string_t bucket,bos_acl_e bos_acl, bos_table_t *resp_headers ) {
    bos_status_t *s = NULL;
    s = bos_create_bucket(options, &bucket, bos_acl, &resp_headers);
    print_headers(resp_headers);
    log_status(s);
    if (bos_status_is_ok(s)) {
        printf("create bucket succeeded\n");
    } else {
        printf("create bucket failed\n");
    }
}
```

> **注意：** 由于Bucket的名称在所有区域中是唯一的，所以需要保证bucketName不与其他所有区域上的Bucket名称相同。
> Bucket的命名有以下规范：
>
> - 只能包括小写字母，数字，短横线（-）。
> - 必须以小写字母或者数字开头。
> - 长度必须在4-63字节之间。

通过上述代码创建的bucket，权限是私有读写，存储类型是标准类型（Standard）。

列举bucket

列举Bucket

如下代码可以列出用户所有的Bucket：

```
void test_get_service(bos_pool_t *p, bos_request_options_t *options, bos_string_t bucket)
{
    bos_status_t *s = NULL;
    bos_get_service_params_t *list_params = NULL;
    bos_table_t *resp_headers = NULL;
    json_t *root;
    list_params = bos_create_get_service_params(options->pool);
    s = bos_get_service(options, &root,list_params, &resp_headers);
    log_status(s);
    bos_pool_destroy(p);
    char* jsonString = json_dumps(root, JSON_INDENT(2));
    printf("\n%s\n", jsonString);
}
```

判断bucket 是否存在

判断Bucket是否存在

若用户需要判断某个Bucket是否存在，则如下代码可以做到：

```
void test_head_bucket(bos_request_options_t *options, bos_string_t bucket, bos_table_t *resp_headers) {
    bos_status_t *status = NULL;
    status = bos_head_bucket(options, &bucket, &resp_headers);
    print_headers(resp_headers);
    if (bos_status_is_ok(status)) {
        printf("head bucket succeeded\n");
    } else {
        printf("head bucket failed\n");
    }
    bos_bucket_exist_status_e bucket_exist;
    status = bos_check_bucket_exist(options, &bucket, &bucket_exist, &resp_headers);
    if (bucket_exist == BOS_BUCKET_NON_EXIST) {
        printf("bucket: %.*s non exist.\n", bucket.len, bucket.data);
    } else if (bucket_exist == BOS_BUCKET_EXIST) {
        printf("bucket: %.*s exist.\n", bucket.len, bucket.data);
    } else {
        printf("bucket: %.*s unknown status.\n", bucket.len, bucket.data);
        log_status(status);
    }
}
```

> **注意：** 如果Bucket不为空（即Bucket中有Object存在），则Bucket无法被删除，必须清空Bucket后才能成功删除。

删除bucket

删除Bucket

如下代码可以删除一个Bucket：

```
    bos_pool_create(&p, NULL);
    options = bos_request_options_create(p);
    init_test_request_options(options, is_cname);
    bos_str_set(&bucket, TEST_BUCKET_NAME);
    s = bos_delete_bucket(options, &bucket, &resp_headers);
```

> **注意：**
>
> - 在删除前需要保证此Bucket下的所有Object和未完成的三步上传Part已经被删除，否则会删除失败。
> - 在删除前确认该Bucket没有开通跨区域复制，不是跨区域复制规则中的源Bucket或目标Bucket，否则不能删除。

文件管理

上传文件

上传文件

在BOS中，用户操作的基本数据单元是Object。Object包含Key、Meta和Data。其中，Key是Object的名字；Meta是用户对该Object的描述，由一系列Name-Value对组成；Data是Object的数据。

BOS C SDK提供了丰富的文件上传接口，可以通过以下方式上传文件：

- 简单上传
- 追加上传
- 分块上传
- 断点续传上传

简单上传

BOS在简单上传的场景中，支持以指定文件形式、以数据流方式、以文件描述符方式、以字符串方式执行Object上传，请参考如下代码：

```
bos_pool_t *p = NULL;
char *object_name = "bos_test_put_object.ts";
char *str = "test bos c sdk";
bos_status_t *s = NULL;
int is_cname = 0;
bos_string_t bucket;
bos_string_t object;
bos_table_t *headers = NULL;
bos_table_t *head_headers = NULL;
bos_table_t *head_resp_headers = NULL;
char *content_type = NULL;
bos_request_options_t *options = NULL;

/* test put object */
bos_pool_create(&p, NULL);
options = bos_request_options_create(p);
init_test_request_options(options, is_cname);
headers = bos_table_make(p, 1);
apr_table_set(headers, "x-bos-meta-author", "bos");
s = create_test_object(options, TEST_BUCKET_NAME, object_name, str, headers);
```

Object以文件的形式上传到BOS中，put_object、upload_file函数支持不超过5GB的Object上传。若要支持大文件（大于5G的文件）上传请使用upload_super_file，参考代码如下:

```
bos_pool_t *p = NULL;
int is_cname = 0;
bos_status_t *s = NULL;
bos_request_options_t *options = NULL;
bos_acl_e bos_acl = BOS_ACL_PRIVATE;
bos_string_t bucket;
bos_table_t *resp_headers;
bos_string_t object;
bos_string_t file_path;

/* create test bucket */
bos_pool_create(&p, NULL);
options = bos_request_options_create(p);
init_test_request_options(options, is_cname);
s = create_test_bucket(options, TEST_BUCKET_NAME, bos_acl);

bos_str_set(&bucket, TEST_BUCKET_NAME);
bos_str_set(&object, "test.mp4");
bos_str_set(&file_path, "../../../test.mp4");
s = bos_put_object_from_file(options, &bucket, &object, &file_path, NULL, &resp_headers);
```

设置文件元信息

文件元信息(Object Meta)，是对用户在向BOS上传文件时，同时对文件进行的属性描述，主要分为分为两种：设置HTTP标准属性（HTTP Headers）和用户自定义的元信息。

- **设定Object的Http Header**

BOS C SDK本质上是调用后台的HTTP接口，因此用户可以在上传文件时自定义Object的Http Header。常用的http header说明如下：

名称	描述	默认值
Content-MD5	文件数据校验，设置后BOS会启用文件内容MD5校验，把您提供的MD5与文件的MD5比较，不一致会抛出错误	无
Content-Type	文件的MIME，定义文件的类型及网页编码，决定浏览器将以什么形式、什么编码读取文件。如没有指，BOS则根据文件的扩展名自动生成，如文件没有扩展名则填默认值	application/octet-stream
Content-Disposition	指示MIME用户代理如何显示附加的文件，打开或下载，及文件名称	无
Content-Length	上传的文件的长度，超过流/文件的长度会截断，不足为实际值	流/文件时间长度
Expires	缓存过期时间	无
Cache-Control	指定该Object被下载时的网页的缓存行为	无

参考代码如下：

```
...
headers = bos_table_make(p, 1);
apr_table_set(headers, BOS_CONTENT_TYPE, "image/jpeg");
...
```

- **用户自定义元信息**

BOS支持用户自定义元数据来对Object进行描述。如下代码所示：

```
apr_table_set(headers, "x-bce-meta-author", "bos");
```

> **提示：**
>
> - 在上面代码中，用户自定义了一个名字为"x-bce-meta-author"，值为"bos"的元数据
> - 当用户下载此Object的时候，此元数据也可以一并得到
> - 一个Object可以有多个类似的参数，但所有的User Meta总大小不能超过2KB

设置Object的Copy属性

BOS同时会提供CopyObject接口用于将一个已经存在的Object拷贝到另外一个Object，拷贝过程中会对源Object的Etag或修改状态进行判断，根据判断结果决定是否执行拷贝。详细的参数解释如下：

名称	类型	描述	是否必需
x-bce-copy-source-if-match	std::string	如果源Object的ETag值和用户提供的ETag相等，则执行拷贝操作，否则拷贝失败。	否
x-bce-copy-source-if-none-match	std::string	如果源Object的ETag和用户提供的ETag不相等，则执行拷贝操作，否则拷贝失败。	否
x-bce-copy-source-if-unmodified-since	std::string	如果源object在x-bce-copy-source-if-unmodified-since之后没被修改，则执行拷贝操作，否则拷贝失败。	否
x-bce-copy-source-if-modified-since	std::string	如果源object在x-bce-copy-source-if-modified-since之后被修改了，则执行拷贝操作，否则拷贝失败。	否

对应的示例代码：

```
bos_pool_t *p = NULL;
  int is_cname = 0;
  bos_status_t *s = NULL;
  bos_request_options_t *options = NULL;
  bos_string_t bucket;
  bos_string_t object;
  bos_string_t src_bucket;
  bos_string_t src_object;
  bos_string_t src_endpoint;
  bos_table_t *resp_headers = NULL;

  bos_pool_create(&p, NULL);
  options = bos_request_options_create(p);
  init_test_request_options(options, is_cname);
  bos_str_set(&bucket, TEST_BUCKET_NAME);
  bos_str_set(&object, "test_copy.txt");
  bos_str_set(&src_bucket, TEST_BUCKET_NAME);
  bos_str_set(&src_object, "bos_test_put_object.ts");
  bos_str_set(&src_endpoint, options->config->endpoint.data);

  bos_copy_object_params_t *params = NULL;
  params = bos_create_copy_object_params(p);
  bos_table_t *headers = bos_table_make(p, 2);
  apr_table_add(headers, "x-bce-metadata-directive", "replace");
  apr_table_add(headers, "x-bce-storage-class", "STANDARD_IA");
  json_t *root;
  s = bos_copy_object(options, &src_bucket, &src_object, &bucket, &object, headers, &root, &resp_headers);
```

上传Object时设置存储类型

BOS支持标准存储, 低频存储和冷存储，上传Object并存储为低频存储类型通过指定StorageClass实现，三种存储类型对应的参数如下：

存储类型	参数
标准存储	STANDARD
低频存储	STANDARD_IA
冷存储	COLD
归档存储	ARCHIVE

追加上传

上文介绍的简单上传方式，创建的Object都是Normal类型，用户不可再进行追加写，这在日志、视频监控、视频直播等数据复写较频繁的场景中使用不方便。

正因如此，百度智能云BOS特别支持了AppendObject，即以追加写的方式上传文件。通过AppendObject操作创建的Object类型为Appendable Object，可以对该Object追加数据。AppendObject大小限制为0~5G。

通过AppendObject方式上传示例代码如下：

```
bos_pool_t *p = NULL;
  char *object_name = "bos_test_append_object_from_file";
  bos_string_t bucket;
  bos_string_t object;
  char *filename = __FILE__;
  bos_string_t append_file;
  bos_status_t *s = NULL;
  int is_cname = 0;
  int64_t position = 0;
  bos_table_t *headers = NULL;
  bos_table_t *resp_headers = NULL;
  bos_request_options_t *options = NULL;

  /* test append object */
  bos_pool_create(&p, NULL);
  options = bos_request_options_create(p);
  init_test_request_options(options, is_cname);
  headers = bos_table_make(p, 0);
  bos_str_set(&bucket, TEST_BUCKET_NAME);
  bos_str_set(&object, object_name);
  bos_str_set(&append_file, filename);

  s = bos_append_object_from_file(options, &bucket, &object, position,
                                  &append_file, headers, &resp_headers);
```

分块上传

除了通过简单上传及追加上传方式将文上传件到BOS以外，BOS还提供了另外一种上传模式 —— Multipart Upload。用户可以在如下的应用场景内（但不仅限于此），使用Multipart Upload上传模式，如：

- 需要支持断点上传。
- 上传超过5GB大小的文件。
- 网络条件较差，和BOS的服务器之间的连接经常断开。

- 需要流式地上传文件。
- 上传文件之前，无法确定上传文件的大小。

下面将一步步介绍Multipart Upload的实现。假设有一个文件，本地路径为 `/path/to/file.zip` ，由于文件比较大，将其分块传输到BOS中。

初始化Multipart Upload

使用 `init_test_multipart_upload` 方法来初始化一个分块上传事件：

```
bos_status_t *s = NULL;
    bos_table_t *resp_headers = NULL;
    bos_string_t object;
    bos_table_t *headers = NULL;
    bos_table_t *complete_headers = NULL;
    bos_string_t upload_id;
    bos_upload_file_t *upload_file = NULL;
    bos_list_upload_part_params_t *params = NULL;
    bos_list_t complete_part_list;
    bos_list_part_content_t *part_content = NULL;
    bos_complete_part_content_t *complete_part_content = NULL;
    int part_num = 1;
    int64_t pos = 0;
    int64_t file_length = 0;
    bos_str_set(&object, "test1");
    s = bos_init_multipart_upload(options, &bucket, &object,
                                  &upload_id, headers, &resp_headers);

    if (bos_status_is_ok(s)) {
        printf("Init multipart upload succeeded, upload_id:%.*s\n",
                        upload_id.len, upload_id.data);
    } else {
        printf("Init multipart upload failed\n");
        return;
    }
```

`initMultiUploadResponse` 的返回结果中含有 `UploadId` ，它是区分分块上传事件的唯一标识，在后面的操作中，我们将用到它。

上传分块

接着，把文件分块上传。

```
int res = BOSE_OK;
    bos_file_buf_t *fb = bos_create_file_buf(p);
    res = bos_open_file_for_all_read(p, TEST_MULTIPART_FILE, fb);
    if (res != BOSE_OK) {
        bos_error_log("Open read file fail, filename:%s\n", TEST_MULTIPART_FILE);
        return;
    }
    file_length = fb->file_last;
    apr_file_close(fb->file);
    while(pos < file_length) {
        upload_file = bos_create_upload_file(p);
        bos_str_set(&upload_file->filename, TEST_MULTIPART_FILE);
        upload_file->file_pos = pos;
        pos += 2 * 1024 * 1024;
        upload_file->file_last = pos < file_length ? pos : file_length; //2MB
        s = bos_upload_part_from_file(options, &bucket, &object, &upload_id,
                                      part_num++, upload_file, &resp_headers);

        if (bos_status_is_ok(s)) {
            printf("Multipart upload part from file succeeded\n");
        } else {
            printf("Multipart upload part from file failed\n");
        }
    }
```

上面代码的核心是调用 `bos_upload_part_from_file` 方法来并发的上传每一个分块，但是要注意以下几点：

- `bos_upload_part_from_file` 要求除最后一个Part以外，其他的Part大小都要大于等于100KB。但是Upload Part接口并不会立即校验上传Part的大小；只有当Complete Multipart Upload的时候才会校验，若upload_part流程中的块大小不符合预期，则complete_multipart_upload接口会报错。
- 为了保证数据在网络传输过程中不出现错误，建议您在upload_part后，使用每个分块BOS返回的Content-MD5值分别验证已上传分块数据的正确性。当所有分块数据合成一个Object后，不再含MD5值。
- Part号码的范围是1~10000。如果超出这个范围，BOS将返回InvalidArgument的错误码。
- 每次上传Part时都要把流定位到此次上传块开头所对应的位置。
- 每次上传Part之后，BOS的返回结果会包含一个 `ETag` 对象，它是上传块的ETag与块编号（PartNumber）的组合，在后续完成分块上传的步骤中会用到它，因此需要将其保存起来。一般来讲这些 `ETag` 对象将被保存到vector中。

完成分块上传

如下代码所示，完成分块上传：

```
s = bos_list_upload_part(options, &bucket, &object, &upload_id,
                         params, &list_part_resp_headers);
CuAssertIntEquals(tc, 200, s->code);
CuAssertIntEquals(tc, 1, params->truncated);
CuAssertStrEquals(tc, expect_part_num_marker,
                  params->next_part_number_marker.data);
CuAssertPtrNotNull(tc, list_part_resp_headers);

bos_list_for_each_entry(bos_list_part_content_t, part_content1, &params->part_list, node) {
    complete_content1 = bos_create_complete_part_content(p);
    bos_str_set(&complete_content1->part_number, part_content1->part_number.data);
    bos_str_set(&complete_content1->etag, part_content1->etag.data);
    bos_list_add_tail(&complete_content1->node, &complete_part_list);
}
bos_list_init(&params->part_list);
if (params->next_part_number_marker.data) {
    bos_str_set(&params->part_number_marker, params->next_part_number_marker.data);
}
s = bos_list_upload_part(options, &bucket, &object, &upload_id, params, &list_part_resp_headers);
CuAssertIntEquals(tc, 200, s->code);
CuAssertIntEquals(tc, 0, params->truncated);
CuAssertPtrNotNull(tc, list_part_resp_headers);
bos_list_for_each_entry(bos_list_part_content_t, part_content2, &params->part_list, node) {
    complete_content2 = bos_create_complete_part_content(p);
    bos_str_set(&complete_content2->part_number, part_content2->part_number.data);
    bos_str_set(&complete_content2->etag, part_content2->etag.data);
    bos_list_add_tail(&complete_content2->node, &complete_part_list);
}

bos_complete_part_content_t *content_test;
bos_list_for_each_entry(bos_complete_part_content_t, content_test, &complete_part_list, node) {
}
s = bos_complete_multipart_upload(options, &bucket, &object, &upload_id,
                                  &complete_part_list, complete_headers, &resp_headers);
```

上面代码中的 `complete_part_list` 是第二部中保存的part_t的列表，BOS收到用户提交的Part列表后，会逐一验证每个数据Part的有效性。当所有的数据Part验证通过后，BOS将把这些数据part组合成一个完整的Object。

取消分块上传事件

用户可以使用abortMultipartUpload方法取消分块上传。

```
bos_status_t *s = NULL;
bos_table_t *resp_headers = NULL;
bos_string_t object;
bos_table_t *headers = NULL;
bos_string_t upload_id;
bos_str_set(&object, "test1");
s = bos_abort_multipart_upload(options, &bucket, &object, &upload_id,
                               &resp_headers);
```

获取未完成的分块上传事件

用户可以使用 `bos_list_multipart_upload` 方法获取Bucket内未完成的分块上传事件。

```
bos_pool_t *p = NULL;
bos_string_t bucket;
char *object_name1 = "bos_test_abort_multipart_upload1";
char *object_name2 = "bos_test_abort_multipart_upload2";
int is_cname = 0;
bos_request_options_t *options = NULL;
bos_string_t upload_id1;
bos_string_t upload_id2;
bos_status_t *s = NULL;
bos_table_t *resp_headers;
bos_list_multipart_upload_params_t *params = NULL;
char *expect_next_key_marker = "bos_test_abort_multipart_upload1";

bos_pool_create(&p, NULL);
options = bos_request_options_create(p);
init_test_request_options(options, is_cname);
s = init_test_multipart_upload(options, TEST_BUCKET_NAME, object_name1, &upload_id1);
CuAssertIntEquals(tc, 200, s->code);

s = init_test_multipart_upload(options, TEST_BUCKET_NAME, object_name2, &upload_id2);
CuAssertIntEquals(tc, 200, s->code);

params = bos_create_list_multipart_upload_params(p);
params->max_ret = 1;
bos_str_set(&bucket, TEST_BUCKET_NAME);
s = bos_list_multipart_upload(options, &bucket, params, &resp_headers);
```

> **注意：**
>
> 1. 默认情况下，如果Bucket中的分块上传事件的数目大于1000，则只会返回1000个Object，并且返回结果中is_truncated的值为True，同时返回next_marker作为下次读取的起点。
> 2. 若想返回更多分块上传事件的数目，可以使用set_marker函数设置marker分次读取。

获取所有已上传的块信息

用户可以使用 `listParts` 方法获取某个上传事件中所有已上传的块。

```
params = bos_create_list_upload_part_params(p);
params->max_ret = 1;
bos_list_init(&complete_part_list);
s = bos_list_upload_part(options, &bucket, &object, &upload_id,
                         params, &list_part_resp_headers);
```

断点续传上传

当用户向BOS上传大文件时，如果网络不稳定或者遇到程序崩等情况，则整个上传就失败了，失败前已经上传的部分也作废，用户不得不重头再来。这样做不仅浪费资源，在网络不稳定的情况下，往往重试多次还是无法完成上传。 基于上述场景，BOS提供了断点续传上传的能力:

- 当网络情况一般的情况下，建议使用三步上传方式，将object分为5Mb的块，参考分块上传。
- 当您的网络情况非常差，推荐使用append_object的方式进行断点续传，每次append 较小数据256kb，参考追加上传。

> 提示
>
> - 断点续传是分片上传的封装和加强，是用分片上传实现的；
> - 文件较大或网络环境较差时，推荐使用分片上传；

下载文件

下载文件

BOS C SDK提供了丰富的文件下载接口，用户可以通过以下方式从BOS中下载文件：

- 下载到本地文件
- 下载到内存中的字符串
- 范围下载

下载到内存中的字符串

用户可以通过如下代码将Object输出到一个文件流中：

```
bos_pool_t *p = NULL;
bos_string_t bucket;
char *object_name = "bos_test_put_object.ts";
bos_string_t object;
int is_cname = 0;
bos_request_options_t *options = NULL;
bos_table_t *headers = NULL;
bos_table_t *params = NULL;
bos_table_t *resp_headers = NULL;
bos_status_t *s = NULL;
bos_list_t buffer;
bos_buf_t *content = NULL;
char *expect_content = "test bos c sdk";
char *buf = NULL;
int64_t len = 0;
int64_t size = 0;
int64_t pos = 0;
char *content_type = NULL;

bos_pool_create(&p, NULL);
options = bos_request_options_create(p);
init_test_request_options(options, is_cname);
bos_str_set(&bucket, TEST_BUCKET_NAME);
bos_str_set(&object, object_name);
bos_list_init(&buffer);

/* test get object to buffer */
s = bos_get_object_to_buffer(options, &bucket, &object, headers,
                             params, &buffer, &resp_headers);
```

直接下载Object到文件

用户可以通过如下代码直接将Object下载到指定文件：

```
bos_pool_t *p = NULL;
bos_string_t bucket;
char *object_name = "bos_test_put_object_from_file2.txt";
bos_string_t object;
char *filename = "bos_test_get_object_to_file";
char *source_filename = __FILE__;
bos_string_t file;
bos_request_options_t *options = NULL;
int is_cname = 0;
bos_table_t *headers = NULL;
bos_table_t *params = NULL;
bos_table_t *resp_headers = NULL;
bos_status_t *s = NULL;
char *content_type = NULL;

bos_pool_create(&p, NULL);
options = bos_request_options_create(p);
init_test_request_options(options, is_cname);
bos_str_set(&bucket, TEST_BUCKET_NAME);
bos_str_set(&object, object_name);
bos_str_set(&file, filename);

/* test get object to file */
s = bos_get_object_to_file(options, &bucket, &object, headers,
                           params, &file, &resp_headers);
```

范围下载

为了实现更多的功能，可以通过使用apr_table_set(headers, "Range", " bytes=5-13");来指定下载范围，实现更精细化地获取Object。如果指定的下载范围是0 - 100，则返回第0到第100个字节的数据，包括第100个，共101字节的数据。

```
void test_get_object_to_buffer_with_range(CuTest *tc)
{
    bos_pool_t *p = NULL;
    bos_string_t bucket;
    char *object_name = "bos_test_put_object.ts";
    bos_string_t object;
    int is_cname = 0;
    bos_request_options_t *options = NULL;
    bos_table_t *headers = NULL;
    bos_table_t *params = NULL;
    bos_table_t *resp_headers = NULL;
    bos_status_t *s = NULL;
    bos_list_t buffer;
    bos_buf_t *content = NULL;
    char *expect_content = "bos c sdk";
    char *buf = NULL;
    int64_t len = 0;
    int64_t size = 0;
    int64_t pos = 0;

    bos_pool_create(&p, NULL);
    options = bos_request_options_create(p);
    init_test_request_options(options, is_cname);
    bos_str_set(&bucket, TEST_BUCKET_NAME);
    bos_str_set(&object, object_name);
    headers = bos_table_make(p, 1);
    apr_table_set(headers, "Range", " bytes=5-13");
    bos_list_init(&buffer);
/* test get object to buffer */
    s = bos_get_object_to_buffer(options, &bucket, &object, headers,
                                 params, &buffer, &resp_headers);

    /* get buffer len */
    len = bos_buf_list_len(&buffer);

    buf = bos_pcalloc(p, (apr_size_t)(len + 1));
    buf[len] = '\0';

    /* copy buffer content to memory */
    bos_list_for_each_entry(bos_buf_t, content, &buffer, node) {
        size = bos_buf_size(content);
        memcpy(buf + pos, content->pos, (size_t)size);
        pos += size;
    }
```

通过apr_table_set(headers, "Range", " bytes=5-13")的方法可以设置返回Object的范围。用户也可以用此功能实现文件的分段下载和断点续传。

获取文件下载链接

获取文件下载URL

用户可以通过如下代码获取指定Object的URL：

```
void test_gen_presinurl(bos_pool_t *p, bos_request_options_t *options, bos_string_t bucket) {
    bos_string_t object;

    bos_str_set(&object, "test1");

    bos_string_t presigned_url;
    int64_t expire = 300;


    bos_gen_presigned_url(options, &bucket, &object, expire, HTTP_GET, &presigned_url);
    printf("presigned_url: %s\n", presigned_url.data);

    bos_pool_destroy(p);
}
```

说明：

- 用户在调用该函数前，需要手动设置endpoint为所属区域域名。百度智能云目前开放了多区域 支持，请参考区域选择说明。目前支持"华北-北京"、"华南-广州"和"华东-苏州"三个区域。北京区域：`http://bj.bcebos.com`，广州区域：`http://gz.bcebos.com`，苏州区域：`http://su.bcebos.com`。
- `expire`为指定的URL有效时长，时间从当前时间算起，为可选参数，不配置时系统默认值为1800秒。如果要设置为永久不失效的时间，可以将`expire`参数设置为 -1，不可设置为其他负数。
- 如果预期获取的文件时公共可读的，则对应URL链接可通过简单规则快速拼接获取: `http://$region.bcebos.com/$bucket/$object`

列举存储空间中的文件

列举存储空间中的文件

BOS SDK支持用户通过以下两种方式列举出object：

- 简单列举
- 通过参数复杂列举

除此之外，用户还可在列出文件的同时模拟文件夹

简单列举

当用户希望简单快速列举出所需的文件时，可通过bos_list_object方法修改bos_list_object_params_t结构体，bos_list_object_params_t对象包含了此次listObject请求的返回结果。用户可以通过bos_list_object_params_t中的getContents方法获取所有object_list的描述信息。

```
    bos_pool_t *p = NULL;
    bos_string_t bucket;
    bos_request_options_t *options = NULL;
    int is_cname = 0;
    bos_table_t *resp_headers = NULL;
    bos_status_t *s = NULL;
    bos_list_object_params_t *params = NULL;
    bos_list_object_content_t *content = NULL;
    int size = 0;
    char *key = NULL;

    bos_pool_create(&p, NULL);
    options = bos_request_options_create(p);
    init_test_request_options(options, is_cname);
    params = bos_create_list_object_params(p);
    params->max_ret = 1;
    params->truncated = 0;
    bos_str_set(&params->prefix, "bos_test");
    bos_str_set(&bucket, TEST_BUCKET_NAME);
    s = bos_list_object(options, &bucket, params, &resp_headers);
//bos_list_for_each_entry(bos_list_object_content_t, content, &params->object_list, node) {
    //    ++size;
    //    key = apr_psprintf(p, "%.*s", content->key.len, content->key.data);
    //}
```

注意：

1. 默认情况下，如果Bucket中的Object数量大于1000，则只会返回1000个Object，并且返回结果中is_truncated值为True，并返回next_marker做为下次读取的起点。
2. 若想增大返回Object的数目，可以使用marker参数分次读取。

通过参数复杂列举

除上述简单列举外，用户还可通过设置params的参数实现各种灵活的查询功能。params的可设置的参数如下：

参数	功能	
prefix	限定返回的object key必须以prefix作为前缀	
delimiter	是一个用于对Object名字进行分组的字符所有名字包含指定的前缀且第一次出现。Delimiter字符之间的object作为一组元素: CommonPrefixes	
marker	设定结果从marker之后按字母排序的第一个开始返回	
max_ret	限定此次返回object的最大数，最大为1000，默认值是1000。如果指定的值大于1000，按1000操作	

注意：

1. 如果有Object以Prefix命名，当仅使用Prefix查询时，返回的所有Key中仍会包含以Prefix命名的Object，详见递归列出目录下所有文件。
2. 如果有Object以Prefix命名，当使用Prefix和Delimiter组合查询时，返回的所有Key中会有Null，Key的名字不包含Prefix前缀，详见查看目录下的文件和子目录。

下面我们分别以几个案例说明通过参数列举的方法：

指定最大返回条数

```
params->max_ret = 100;
```

返回指定前缀的object

```
bos_str_set(&params->prefix, "bos_test");
```

从指定Object后返回

```
bos_str_set(&params->marker, params->next_marker.data);
```

bos_list_object_params_t解析类中可供调用的参数有：

参数	说明
name	Bucket名称
prefix	匹配以prefix开始到第一次出现Delimiter字符之间的object作为一组元素返回
marker	本次查询的起点
max_ret	请求返回的最大数目
is_truncated	指明是否所有查询都返回了；false-本次已经返回所有结果，true-本次还没有返回所有结果
object_list	返回的一个Object的列表
+key	Object名称
+last_modified	此Object最后一次被修改的时间
+etag	Object的HTTP协议实体便签
+storage_class	Object的存储形态
+size	Object的内容的大小（字节数）
+owner_id	Bucket Owner的用户ID
+owner_display_name	Bucket Owner的名称
common_prefix_list	返回的common prefix的列表

模拟文件夹功能

在BOS的存储结果中是没有文件夹这个概念的，所有元素都是以Object来存储，但BOS的用户在使用数据时往往需要以文件夹来管理文件。 因此，BOS提供了创建模拟文件夹的能力，其本质上来说是创建了一个size为0的Object。对于这个Object可以上传下载，只是控制台会对以"/"结尾的Object以文件夹的方式展示。

用户可以通过 Delimiter 和 Prefix 参数的配合模拟出文件夹功能。Delimiter 和 Prefix 的组合效果是这样的：

如果把 Prefix 设为某个文件夹名，就可以罗列以此 Prefix 开头的文件，即该文件夹下递归的所有的文件和子文件夹（目录）。文件名在Contents中显示。 如果再把 Delimiter 设置为 "/" 时，返回值就只罗列该文件夹下的文件和子文件夹（目录），该文件夹下的子文件名（目录）返回在 CommonPrefixes 部分，子文件夹下递归的文件和文件夹不被显示。

如下是几个应用方式：

列出Bucket内所有文件

当用户需要获取Bucket下的所有文件时，可以参考分页获取所有Object

递归列出目录下所有文件

可以通过设置 Prefix 参数来获取某个目录下所有的文件：

```
void test_list_object_with_delimiter(CuTest *tc)
{
    bos_pool_t *p = NULL;
    bos_string_t bucket;
    bos_request_options_t *options = NULL;
    int is_cname = 0;
    bos_table_t *resp_headers = NULL;
    bos_status_t *s = NULL;
    bos_list_object_params_t *params = NULL;
    bos_list_object_common_prefix_t *common_prefix = NULL;
    int size = 0;
    char *prefix = NULL;

    bos_pool_create(&p, NULL);
    options = bos_request_options_create(p);
    init_test_request_options(options, is_cname);
    params = bos_create_list_object_params(p);
    params->max_ret = 5;
    params->truncated = 0;
    bos_str_set(&params->prefix, "bos_tmp");
    bos_str_set(&params->delimiter, "/");
    bos_str_set(&bucket, TEST_BUCKET_NAME);
    s = bos_list_object(options, &bucket, params, &resp_headers);
    bos_list_for_each_entry(bos_list_object_common_prefix_t, common_prefix, &params->common_prefix_list, node) {
        ++size;
        prefix = apr_psprintf(p, "%.*s", common_prefix->prefix.len,
                              common_prefix->prefix.data) ;
    }
    bos_pool_destroy(p);
    printf("test_list_object_with_delimiter ok\n");
}
```

Object权限控制

权限控制

设置对象的访问权限

目前BOS支持两种方式设置ACL. 具体可参考权限控制

第一种是使用Canned Acl，在bos_put_object_acl的时候，通过头域的"x-bce-acl", "x-bce-grant-read", 或者 "x-bce-grant-permission"来设置对象的访问权限，当前可设置的权限包括private和public-read，三种类型的header不可以同时在一个请求中出现.

第二种方式是以自定义Acl样式, 具体可通过上传其json字符串, 设置accessControlList, 或者直接上传ACL文件. 具体可参考权限控制概述

设置Canned ACL

Canned ACL是预定义的访问权限，用户可选择对某个对象进行设置，支持三种接口：

```
bos_acl_e bos_acl;
 bos_acl = BOS_ACL_PRIVATE;
 bos_status_t *s = NULL;
 bos_table_t *resp_headers = NULL;
 bos_string_t object;
 s = bos_put_object_acl(options, &bucket, &object, bos_acl, NULL, NULL, NULL, NULL, &resp_headers);
 print_headers(resp_headers);
 if (bos_status_is_ok(s)) {
    printf("put bucket acl succeeded\n");
 } else {
    printf("put bucket acl failed\n");
 }
```

设置自定义ACL

用户可参考如下代码设置Bucket内的对象的自定义访问权限，支持三种不同参数：

```
bos_acl_e bos_acl;
 bos_acl = 6;
 bos_status_t *s = NULL;
 bos_table_t *resp_headers = NULL;
 bos_string_t object;
 char * acl_json = "{ \"accessControlList\": [ { \"grantee\":[{ \"id\":\"168bf6fd8fa74d9789f35a283a1f15e2\
" }], \"permission\":[\"WRITE\"] } ] }";
 s = bos_put_object_acl(options, &bucket, &object, NULL, acl_json, NULL, NULL, NULL, &resp_headers);
 print_headers(resp_headers);
 if (bos_status_is_ok(s)) {
    printf("put object acl succeeded\n");
 } else {
    printf("put object acl failed\n");
 }
```

获取对象的访问权限

如下代码可获取一个对象的访问权限：

```
s = bos_get_object_acl(options, &bucket, &object, &root, &resp_headers);
  *jsonString = json_dumps(root, JSON_INDENT(2));
 printf("\n%s\n", jsonString);
```

> 注意:
>
> acl涉及到的具体结构体Grant, 在bucket acl和object acl体系中共用
> 目前object acl体系中只用到其中grantee, permission两个字段.
> 其余注释的字段均为bucket acl体系独有.

拷贝文件

拷贝文件

拷贝一个文件

用户可以通过bos_copy_object函数拷贝一个Object，如下代码所示：

```
bos_pool_t *p = NULL;
int is_cname = 0;
bos_status_t *s = NULL;
bos_request_options_t *options = NULL;
bos_string_t bucket;
bos_string_t object;
bos_string_t src_bucket;
bos_string_t src_object;
bos_string_t src_endpoint;
bos_table_t *resp_headers = NULL;

bos_pool_create(&p, NULL);
options = bos_request_options_create(p);
init_test_request_options(options, is_cname);
bos_str_set(&bucket, TEST_BUCKET_NAME);
bos_str_set(&object, "test_copy.txt");
bos_str_set(&src_bucket, TEST_BUCKET_NAME);
bos_str_set(&src_object, "bos_test_put_object.ts");
bos_str_set(&src_endpoint, options->config->endpoint.data);

bos_copy_object_params_t *params = NULL;
params = bos_create_copy_object_params(p);
bos_table_t *headers = bos_table_make(p, 2);
apr_table_add(headers, "x-bce-metadata-directive", "replace");
apr_table_add(headers, "x-bce-storage-class", "STANDARD_IA");
json_t *root;
s = bos_copy_object(options, &src_bucket, &src_object, &bucket, &object, headers, &root, &resp_headers);
```

同步Copy功能

当前BOS的CopyObject接口是通过同步方式实现的。同步方式下，BOS端会等待Copy实际完成才返回成功。同步Copy能帮助用户更准确的判断Copy状态，但用户感知的复制时间会变长，且复制时间和文件大小成正比。

同步Copy方式更符合业界常规，提升了与其它平台的兼容性。同步Copy方式还简化了BOS服务端的业务逻辑，提高了服务效率。

分块拷贝

除了通过CopyObject接口拷贝文件以外，BOS还提供了另外一种拷贝模式——Multipart Upload Copy。用户可以在如下的应用场景内（但不仅限于此），使用Multipart Upload Copy，如：

- 需要支持断点拷贝。
- 拷贝超过5GB大小的文件。
- 网络条件较差，和BOS的服务器之间的连接经常断开。

下面将介绍分步实现三步拷贝。

三步拷贝包含init、“拷贝分块”和complete三步，其中init和complete的操作同分块上传一致。

为了便于理解，下面提供三步拷贝完整代码：

```
void test_upload_part_copy(CuTest *tc)
{
    bos_pool_t *p = NULL;
    bos_request_options_t *options = NULL;
    int is_cname = 0;
    bos_string_t upload_id;
    bos_list_upload_part_params_t *list_upload_part_params = NULL;
    bos_upload_part_copy_params_t *upload_part_copy_params1 = NULL;
    bos_upload_part_copy_params_t *upload_part_copy_params2 = NULL;
    bos_table_t *headers = NULL;
    bos_table_t *query_params = NULL;
    bos_table_t *resp_headers = NULL;
    bos_table_t *list_part_resp_headers = NULL;
    bos_list_t complete_part_list;
    bos_list_part_content_t *part_content = NULL;
    bos_complete_part_content_t *complete_content = NULL;
    bos_table_t *complete_resp_headers = NULL;
    bos_status_t *s = NULL;
    int part1 = 1;
    int part2 = 2;
    char *local_filename = "test_upload_part_copy.file";
    char *download_filename = "test_upload_part_copy.file.download";
    char *source_object_name = "bos_test_upload_part_copy_source_object";
    char *dest_object_name = "bos_test_upload_part_copy_dest_object";
    FILE *fd = NULL;
    bos_string_t download_file;
    bos_string_t dest_bucket;
    bos_string_t dest_object;
    int64_t range_start1 = 0;
   int64_t range_end1 = 6000000;
    int64_t range_start2 = 6000001;
    int64_t range_end2;
    bos_string_t data;

    bos_pool_create(&p, NULL);
    options = bos_request_options_create(p);

    // create multipart upload local file
    make_rand_string(p, 10 * 1024 * 1024, &data);
    fd = fopen(local_filename, "w");
    CuAssertTrue(tc, fd != NULL);
    fwrite(data.data, sizeof(data.data[0]), data.len, fd);
    fclose(fd);

    init_test_request_options(options, is_cname);
    headers = bos_table_make(p, 0);
    s = create_test_object_from_file(options, TEST_BUCKET_NAME, source_object_name,
        local_filename, headers);
```

```
    //init mulitipart
    s = init_test_multipart_upload(options, TEST_BUCKET_NAME, dest_object_name, &upload_id);
    //upload part copy 1
    upload_part_copy_params1 = bos_create_upload_part_copy_params(p);
    bos_str_set(&upload_part_copy_params1->copy_source, "mybucket.bcebos.com/bos_test_upload_part_copy_source_object");
    bos_str_set(&upload_part_copy_params1->dest_bucket, TEST_BUCKET_NAME);
    bos_str_set(&upload_part_copy_params1->dest_object, dest_object_name);
    bos_str_set(&upload_part_copy_params1->upload_id, upload_id.data);
    upload_part_copy_params1->part_num = part1;
    upload_part_copy_params1->range_start = range_start1;
    upload_part_copy_params1->range_end = range_end1;

    headers = bos_table_make(p, 0);
    s = bos_upload_part_copy(options, upload_part_copy_params1, headers, &resp_headers);
    //upload part copy 2
    resp_headers = NULL;
    range_end2 = get_file_size(local_filename) - 1;
    upload_part_copy_params2 = bos_create_upload_part_copy_params(p);
    bos_str_set(&upload_part_copy_params2->copy_source, "mybucket.bcebos.com/bos_test_upload_part_copy_source_object");
    bos_str_set(&upload_part_copy_params2->dest_bucket, TEST_BUCKET_NAME);
    bos_str_set(&upload_part_copy_params2->dest_object, dest_object_name);
    bos_str_set(&upload_part_copy_params2->upload_id, upload_id.data);
    upload_part_copy_params2->part_num = part2;
    upload_part_copy_params2->range_start = range_start2;
    upload_part_copy_params2->range_end = range_end2;

    headers = bos_table_make(p, 0);
    s = bos_upload_part_copy(options, upload_part_copy_params2, headers, &resp_headers);
    //list part
    list_upload_part_params = bos_create_list_upload_part_params(p);
    list_upload_part_params->max_ret = 10;
    bos_list_init(&complete_part_list);

    bos_str_set(&dest_bucket, TEST_BUCKET_NAME);
    bos_str_set(&dest_object, dest_object_name);
    s = bos_list_upload_part(options, &dest_bucket, &dest_object, &upload_id,
                             list_upload_part_params, &list_part_resp_headers);

    bos_list_for_each_entry(bos_list_part_content_t, part_content, &list_upload_part_params->part_list, node) {
        complete_content = bos_create_complete_part_content(p);
        bos_str_set(&complete_content->part_number, part_content->part_number.data);
        bos_str_set(&complete_content->etag, part_content->etag.data);
        bos_list_add_tail(&complete_content->node, &complete_part_list);
    }
    s = bos_complete_multipart_upload(options, &dest_bucket, &dest_object,
            &upload_id, &complete_part_list, headers, &complete_resp_headers);
    //check upload copy part content equal to local file
    headers = bos_table_make(p, 0);
    bos_str_set(&download_file, download_filename);
    s = bos_get_object_to_file(options, &dest_bucket, &dest_object, headers,
                               query_params, &download_file, &resp_headers);
    remove(download_filename);
    remove(local_filename);
    bos_pool_destroy(p);

    printf("test_upload_part_copy ok\n");
}
```

使用自动切换函数

```
 void test_upload_file(CuTest *tc)
{
    bos_pool_t *p = NULL;
    bos_string_t bucket;
    char *object_name = "bos_test_multipart_upload_from_file";
    bos_string_t object;
    int is_cname = 0;
    bos_request_options_t *options = NULL;
    bos_status_t *s = NULL;
    int part_size = 100 * 1024;
    bos_string_t upload_id;
    bos_string_t filepath;

    bos_pool_create(&p, NULL);
    options = bos_request_options_create(p);
    init_test_request_options(options, is_cname);
    bos_str_set(&bucket, TEST_BUCKET_NAME);
    bos_str_set(&object, object_name);
    bos_str_null(&upload_id);
    bos_str_set(&filepath, __FILE__);
    s = bos_upload_file(options, &bucket, &object, &upload_id, &filepath,
                        part_size, NULL);
    bos_pool_destroy(p);
    printf("test_upload_file ok\n");
}
```

获取及更新元数据信息

文件元信息

设置文件元信息

文件元信息(Object Meta)，是对用户在向BOS上传文件时，同时对文件进行的属性描述，主要分为分为两种：设置HTTP标准属性（HTTP Headers）和用户自定义的元信息。

- **设定Object的Http Header**

BOS C SDK本质上是调用后台的HTTP接口，因此用户可以在上传文件时自定义Object的Http Header。常用的http header说明如下：

名称	描述	默认值
Content-MD5	文件数据校验，设置后BOS会启用文件内容MD5校验，把您提供的MD5与文件的MD5比较，不一致会抛出错误	无
Content-Type	文件的MIME，定义文件的类型及网页编码，决定浏览器将以什么形式、什么编码读取文件。如没有指，BOS则根据文件的扩展名自动生成，如文件没有扩展名则填默认值	application/octet-stream
Content-Disposition	指示MIME用户代理如何显示附加的文件，打开或下载，及文件名称	无
Content-Length	上传的文件的长度，超过流/文件的长度会截断，不足为实际值	流/文件时间长度
Expires	缓存过期时间	无
Cache-Control	指定该Object被下载时的网页的缓存行为	无

参考代码如下：

```
...
bos_table_t *headers = bos_table_make(p, 2);
  apr_table_add(headers, "x-bce-metadata-directive", "replace");
  apr_table_add(headers, "x-bce-storage-class", "STANDARD_IA");
...
```

- **用户自定义元信息**

BOS支持用户自定义元数据来对Object进行描述。如下代码所示：

```
// 设置自定义元数据name的值为my-data
apr_table_add(headers, "name", "my-data");
```

> **提示：**
>
> - 在上面代码中，用户自定义了一个名字为"name"，值为"my-data"的元数据
> - 当用户下载此Object的时候，此元数据也可以一并得到
> - 一个Object可以有多个类似的参数，但所有的User Meta总大小不能超过2KB

只获取ObjectMetaData

通过 head_object 方法可以只获取ObjectMetaData而不获取Object的实体。

`ObjectMetaData`解析类中可供调用的参数有：

参数	说明
content_type	Object的类型
content_length	Object的大小
content_md5	Object的MD5
etag	Object的HTTP协议实体标签
storage_class	Object的存储类型
user_meta	如果在PutObject指定了userMetadata自定义meta，则返回此项
expires	下载Object时的缓存失效时间
content_disposition	设置浏览器是否下载，可取值为inline、attachment; filename="download.txt"
cache_control	下载Object的Cache设置，常见的可取值为private、no-cache、max-age、must-revalidate
content_range	有range的情况下返回Object的数据范围

获取及更新文件元信息

文件元信息(Object Metadata)，是对用户上传BOS的文件的属性描述，分为两种：HTTP标准属性（HTTP Headers）和User Meta（用户自定义元信息）。

```
bos_pool_t *p = NULL;
int is_cname = 0;
bos_status_t *s = NULL;
bos_request_options_t *options = NULL;
bos_string_t bucket;
bos_string_t object;
bos_table_t *resp_headers = NULL;


//创建内存池
bos_pool_create(&p, NULL);


//初始化请求选项
options = bos_request_options_create(p);
init_test_request_options(options, is_cname);
bos_str_set(&bucket, TEST_BUCKET_NAME);


//获取对象元数据
bos_str_set(&object, TEST_OBJECT_NAME1);
s = bos_head_object(options, &bucket, &object, NULL, &resp_headers);
print_headers(resp_headers);
if (bos_status_is_ok(s)) {
    printf("head object succeeded\n");
} else {
    printf("head object failed\n");
}


//销毁内存池
bos_pool_destroy(p);
```

修改文件元信息

BOS修改Object的Metadata通过拷贝Object实现。即拷贝Object的时候，把目的Bucket设置为源Bucket，目的Object设置为源Object，并设置新的Metadata，通过拷贝自身实现修改Metadata的目的。如果不设置新的Metadata，则报错。

```
bos_pool_t *p = NULL;
int is_cname = 0;
bos_status_t *s = NULL;
bos_request_options_t *options = NULL;
bos_string_t bucket;
bos_string_t object;
bos_string_t src_bucket;
bos_string_t src_object;
bos_string_t src_endpoint;
bos_table_t *resp_headers = NULL;


//创建内存池
bos_pool_create(&p, NULL);

//初始化请求选项
options = bos_request_options_create(p);
init_test_request_options(options, is_cname);
bos_str_set(&bucket, TEST_BUCKET_NAME);


//设置对象复制
bos_str_set(&object, TEST_OBJECT_NAME2);
bos_str_set(&src_bucket, TEST_BUCKET_NAME);
bos_str_set(&src_endpoint, TEST_BOS_ENDPOINT);
bos_str_set(&src_object, TEST_OBJECT_NAME1);
bos_table_t *headers = bos_table_make(p, 2);

bos_copy_object_params_t *params = NULL;
params = bos_create_copy_object_params(p);
s = bos_copy_object(options, &src_bucket, &src_object, &bucket, &object, headers, params, &resp_headers);
if (bos_status_is_ok(s)) {
    printf("put object copy succeeded\n");
} else {
    printf("put object copy failed\n");
}


//销毁内存池
bos_pool_destroy(p);
```

变更文件存储等级

变更文件存储等级

上文中已提到，BOS支持为文件赋予STANDARD(标准存储), STANDARD_IA(低频存储)和COLD(冷存储)三种存储类型。同时，BOS C SDK也支持用户对特定文件执行存储类型变更的操作。 涉及到的参数如下：

参数	说明
x-bce-storage-class	指定Object的存储类型，STANDARD_IA代表低频存储，COLD代表冷存储，不指定时默认是标准存储类型。

示例如下：

```
bos_str_set(&object, TEST_OBJECT_NAME1);
bos_copy_object_params_t *params = NULL;
params = bos_create_copy_object_params(p);
bos_table_t *headers = bos_table_make(p, 2);
apr_table_add(headers, "x-bce-metadata-directive", "replace");
//apr_table_add(headers, "x-bce-storage-class", "ARCHIVE");
apr_table_add(headers, "x-bce-storage-class", "STANDARD_IA");
s = bos_copy_object(options, &bucket, &object, &bucket, &object, headers, &root, &resp_headers);
if (bos_status_is_ok(s)) {
    printf("put object copy succeeded\n");
} else {
    printf("put object copy failed\n");
}
```

归档存储

归档存储 取回

```
bos_pool_t *p = NULL;
bos_string_t bucket;
bos_string_t object;
int is_cname = 0;
bos_table_t *resp_headers = NULL;
bos_request_options_t *options = NULL;
bos_status_t *s = NULL;

bos_pool_create(&p, NULL);
options = bos_request_options_create(p);
init_test_request_options(options, is_cname);
bos_str_set(&bucket, TEST_BUCKET_NAME);
bos_str_set(&object, "test_restore.dat");
bos_object_restore_params_t *restore_params = bos_create_object_restore_params(p);
restore_params->days = 30;
bos_str_set(&restore_params->tier, "Standard");
s = bos_post_object_restore(options, &bucket, &object, restore_params, NULL, NULL, &resp_headers);
bos_pool_destroy(p);

printf("test_object_restore ok\n");
```

查看文件是否存在

查看文件是否存在

用户可通过如下操作查看某文件是否存在：

```
options = bos_request_options_create(p);
init_test_request_options(options, is_cname);
bos_str_set(&bucket, TEST_BUCKET_NAME);

//获取对象元数据
bos_str_set(&object, TEST_OBJECT_NAME1);
s = bos_head_object(options, &bucket, &object, NULL, &resp_headers);
print_headers(resp_headers);
if (bos_status_is_ok(s)) {
    printf("ObjectKey exist!\n");
} else {
    printf("ObjectKey not exist!\n");
}
```

删除文件

删除文件

删除单个文件

可参考如下代码删除了一个Object:

```
void test_delete_object(CuTest *tc)
{
    bos_pool_t *p = NULL;
    bos_string_t bucket;
    char *object_name = "bos_test_put_object";
    bos_string_t object;
    int is_cname = 0;
    bos_request_options_t *options = NULL;
    bos_table_t *resp_headers = NULL;
    bos_status_t *s = NULL;

    bos_pool_create(&p, NULL);
    options = bos_request_options_create(p);
    init_test_request_options(options, is_cname);
    bos_str_set(&bucket, TEST_BUCKET_NAME);
    bos_str_set(&object, object_name);

    /* test delete object */
    s = bos_delete_object(options, &bucket, &object, &resp_headers);
    bos_pool_destroy(p);

    printf("test_delete_object ok\n");
}
```

删除多个文件

删除多个文件bos_delete_objects使用并发接口，提高请求数的吞吐，代码如下：

```
bos_pool_t *p = NULL;
int is_cname = 0;
bos_string_t bucket;
bos_status_t *s = NULL;
bos_table_t *resp_headers = NULL;
bos_request_options_t *options = NULL;
char *object_name1 = "bos_tmp1/";
char *object_name2 = "bos_tmp2/";
bos_object_key_t *content1 = NULL;
bos_object_key_t *content2 = NULL;
bos_list_t object_list;
bos_list_t deleted_object_list;
int is_quiet = 0;

bos_pool_create(&p, NULL);
options = bos_request_options_create(p);
init_test_request_options(options, is_cname);
bos_str_set(&bucket, TEST_BUCKET_NAME);
bos_list_init(&object_list);
bos_list_init(&deleted_object_list);
content1 = bos_create_bos_object_key(p);
bos_str_set(&content1->key, object_name1);
bos_list_add_tail(&content1->node, &object_list);
content2 = bos_create_bos_object_key(p);
bos_str_set(&content2->key, object_name2);
bos_list_add_tail(&content2->node, &object_list);
s = bos_delete_objects(options, &bucket, &object_list, is_quiet,
    &resp_headers, &deleted_object_list);
```

按照某个prefix删除objects

```
void test_delete_objects_by_prefix(CuTest *tc)
{
    bos_pool_t *p = NULL;
    bos_request_options_t *options = NULL;
    int is_cname = 0;
    bos_string_t bucket;
    bos_status_t *s = NULL;
    bos_string_t prefix;
    char *prefix_str = "bos_tmp3/";

    bos_pool_create(&p, NULL);
    options = bos_request_options_create(p);
    init_test_request_options(options, is_cname);
    bos_str_set(&bucket, TEST_BUCKET_NAME);
    bos_str_set(&prefix, prefix_str);

    s = bos_delete_objects_by_prefix(options, &bucket, &prefix);
    bos_pool_destroy(p);

}
```

异常处理

SDK 调用失败时，结果信息包含在 API 返回的 bos_status_t 结构中。

SDK 中使用每一个 API 的正确做法如下所示，为了简要，文档中范例不再给出具体异常的处理，仅给出 API 的使用范例。

```
bos_status_t *s = NULL;
s = bos_put_object_from_file(options, &bucket, &object, &file, &headers, &resp_headers);
if (!s && !bos_status_is_ok(s)) {
 // 按需要进行异常场景的日志输出和处理
    bos_warn_log("failed to put object from file", buf);
    if (s->error_code) bos_warn_log("status->error_code: %s", s->error_code);
    if (s->error_msg) bos_warn_log("status->error_msg: %s", s->error_msg);
    if (s->req_id) bos_warn_log("status->req_id: %s", s->req_id);
}
```

客户端异常

客户端异常表示客户端尝试向BOS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，client操作会返回非零；当上传文件时发生IO异常时，返回非零。

当 bos_status_t 结构中 code 成员值小于0时，表明发生 SDK 本地客户端错误，错误码信息参考枚举 bos_error_code_e 定义。

```
typedef enum {
  BOSE_OK = 0,
  BOSE_OUT_MEMORY = -1000,
  BOSE_OVER_MEMORY = -999,
  BOSE_FAILED_CONNECT = -998,
  BOSE_ABORT_CALLBACK = -997,
  BOSE_INTERNAL_ERROR = -996,
  BOSE_REQUEST_TIMEOUT = -995,
  BOSE_INVALID_ARGUMENT = -994,
  BOSE_INVALID_OPERATION = -993,
  BOSE_CONNECTION_FAILED = -992,
  BOSE_FAILED_INITIALIZE = -991,
  BOSE_NAME_LOOKUP_ERROR = -990,
  BOSE_FAILED_VERIFICATION = -989,
  BOSE_WRITE_BODY_ERROR = -988,
  BOSE_READ_BODY_ERROR = -987,
  BOSE_SERVICE_ERROR = -986,
  BOSE_OPEN_FILE_ERROR = -985,
  BOSE_FILE_SEEK_ERROR = -984,
  BOSE_FILE_INFO_ERROR = -983,
  BOSE_FILE_READ_ERROR = -982,
  BOSE_FILE_WRITE_ERROR = -981,
  BOSE_JSON_PARSE_ERROR = -975,
  BOSE_UTF8_ENCODE_ERROR = -979,
  BOSE_CRC_INCONSISTENT_ERROR = -978,
  BOSE_FILE_FLUSH_ERROR = -977,
  BOSE_FILE_TRUNC_ERROR = -976,
  BOSE_UNKNOWN_ERROR = -100
} bos_error_code_e;
```

服务端异常

当 bos_status_t 结构中 code 成员值大于0时，表明发生网络侧错误。

以下是 bos_status_t 结构的描述：

bos_status_t 成员	描述	类型
code	response 的 status 状态码， 4xx 是指请求因客户端而失败， 5xx 是服务端异常导致的失败	Int
error_code	请求失败时 body 返回的 Error Code	String
error_msg	请求失败时 body 返回的 Error Message	String
req_id	请求 ID，用于标识用户唯一请求	String

版本变更记录

- C SDK开发包[2024-03-06] 版本号 1.0.0

  首次发布：

  - 支持创建、查看、罗列、删除 Bucket
  - 修改、获取Bucket的访问权限
  - 上传、查看、罗列、删除、拷贝、追加写Object
  - 三步上传，断点续传
  - 生命周期，replication

Harmony-SDK

概述

鸿蒙SDK

本文档主要介绍BOS鸿蒙SDK的安装和使用。在使用本文档前，您需要先了解BOS的一些基本知识，并已开通了BOS服务。若您还不了解BOS，可以参考 产品描述和快速入门。

安装SDK工具包

运行环境

BOS鸿蒙SDK可以在API 5.0.0（12）及以上环境下运行。

安装SDK

1. 在官网下载BOS 鸿蒙SDK工具包。
2. 将其置于您的工程目录下，例如`./lib/sdk.har`（假设当前目录为工程的根目录）
3. 修改entry目录下的oh-package.json5文件中的dependencies项，以添加依赖。

```
  "dependencies": {
   "@bdbase/bos": "file:../lib/sdk.har"
  }
```

4. File -> Sync And Refresh Project 以更新生效依赖

常见问题

1. 如果编译时遇到如下报错：

```
 hvigor ERROR: ArkTS:ERROR Failed to execute es2abc.
Error Message: Error: The input abc file
‘/Users/user/DevEcoStudioProjects/test_har/oh_modules/.ohpm/@bdbase+bos@gu1lksm7jphrvy+88nlwwgx1ljzjciyeijghnl5gdfq=/oh_modules/@bdbase/bos/ets/modules.abc’ owns a higher api version or a higher sdkReleaseType compared to current compilation process
```

可以尝试在build-profile.json5中products项下添加compatibleSdkVersionStage: "beta6",

初始化

快速入门

1. 初始化一个带有STS验证的BosClient。

BosClient是与BOS服务交互的客户端，BOS Harmony SDK的BOS操作都是通过BosClient完成的。示例代码：

```
let credential = new Credential(<AccessKeyID>, <SecretAccessKey>, <Token>); //STS返回的临时AK/SK及Token
let clientOptions = new ClientOptions();
clientOptions.endpoint = "bj.bcebos.com";  //传入Bucket所在区域域名
let bosClient = new BosClient(credential, clientOptions); // 创建BosClient
```

> **说明**：Harmony SDK主要用于移动端开发的场景，移动端是一个不受信任的环境，把AccessKeyId和SecretAccessKey直接保存在终端用来加签请求，存在极高的风险。故推荐您使用STS鉴权模式。STS的详细介绍请参考临时授权访问。

2. 新建一个Bucket.

Bucket是BOS上的命名空间，相当于数据的容器，可以存储若干数据实体（Object）。在您上传数据前，必须先创建一个Bucket。示例代码：

```
let location: string;
let bucketName = "test-bucket-harmony";
try {
    location = await bosClient.putBucket(<bucketName>);
    logger.info(`create bucket success, location:${location}`);
} catch (bosResponse) { // 创建bucket失败
    printErrorInfo(bosResponse, "put bucket error"); // 打印错误信息
}
```

3. 上传Object。

Object是BOS中最基本的数据单元，您可以把Object简单的理解为文件。对于一个简单的Object的上传，BOS提供了二进制串上传的方式。示例代码：

```
let cacheDir = getContext().cacheDir;
let path = cacheDir + '/test.txt';
let file = fs.openSync(path, fs.OpenMode.CREATE | fs.OpenMode.READ_WRITE | fs.OpenMode.TRUNC)
fs.writeSync(file.fd, "Hello BOS");
fs.fsyncSync(file.fd);
fs.closeSync(file.fd);


file = fs.openSync(path, fs.OpenMode.READ_ONLY); // 使用同步的方式打开文件
let stat = fs.lstatSync(path); //获取文件的信息
let buf = new ArrayBuffer(stat.size); //构造ArrayBuffer
fs.readSync(file.fd, buf); // 读取文件内容
fs.fsyncSync(file.fd);
fs.closeSync(file.fd);
let etag: string;

try {
    etag = await bosClient.putObject(<bucketName>, <objectName>, buf); //上传object
    logger.info(`put object success, etag:${etag}`); // 打印etag
} catch (bosResponse) {
    printErrorInfo(bosResponse, "put object fail");
}
```

4. 查看Bucket下的Object列表。

当您完成一系列上传后，可以参考如下代码来查看Bucket下的全部Object。示例代码：

```
let listObjectsResult: ListObjectsResult;
try {
    listObjectsResult = await bosClient.listObjects(<bucketName>);
// 遍历所有Object
for (let objectInfo of listObjectsResult.contents as ObjectSummaryType[]) {
    logger.info(`name: ${objectInfo.key} size: ${objectInfo.size} storageClass: ${objectInfo.storageClass}`);
}
```

5. 获取指定的Object.

用户可以参考如下代码来实现对一个或者多个Object的获取。示例代码：

```
let getObjectResult: GetObjectResult;

try { // 注意: 当前接口仅支持下载不超过5MB的Object
    getObjectResult = await bosClient.getObject(, );
    logger.info(`get object success, etag:${JSON.stringify(getObjectResult.etag)}`);
} catch (bosResponse) {
    printErrorInfo(bosResponse, "get object fail");
}
```

完整示例

```
import fs from '@ohos.file.fs';
import { logger, Credential, BosClient, ClientOptions} from "bos"
import { BosResponse } from "bos/src/main/ets/bos/model/BosResponse"
// 下文中所有返回结果的类都在这个路径下
import {  GetObjectResult, ObjectSummaryType, ListObjectsResult, ListObjectsArgs } from "bos/src/main/ets/bos/api/DataType

function printErrorInfo(bosResponse: BosResponse, prefix: string) {
  logger.error(`${prefix}, info: ${JSON.stringify(bosResponse.error)}`);
}

export async function fastStart() {
  let AccessKeyID = ""
  let SecretAccessKey = ""
  let Token = ""
  let objectName = "test-object"
  let credential = new Credential(AccessKeyID, SecretAccessKey, Token); //STS返回的临时AK/SK及Token
  let clientOptions = new ClientOptions();
  clientOptions.endpoint = "bj.bcebos.com";  //传入Bucket所在区域域名
  let bosClient = new BosClient(credential, clientOptions); // 创建BosClient
  let bucketName = "test-bucket-harmony";
  let location: string;

  try {
    location = await bosClient.putBucket(bucketName);
    logger.info(`create bucket success, location:${location}`);
  } catch (bosResponse) { // 创建bucket失败
    printErrorInfo(bosResponse, "put bucket error"); // 打印错误信息
  }

  // 创建一个临时文件

  let cacheDir = getContext().cacheDir;
  let path = cacheDir + '/test.txt';
  let file = fs.openSync(path, fs.OpenMode.CREATE | fs.OpenMode.READ_WRITE | fs.OpenMode.TRUNC)
  fs.writeSync(file.fd, "Hello BOS");
  fs.fsyncSync(file.fd);
  fs.closeSync(file.fd);

  file = fs.openSync(path, fs.OpenMode.READ_ONLY); // 使用同步的方式打开文件
  let stat = fs.lstatSync(path); //获取文件的信息
  let buf = new ArrayBuffer(stat.size); //构造ArrayBuffer
  fs.readSync(file.fd, buf); // 读取文件内容
  fs.fsyncSync(file.fd);
  fs.closeSync(file.fd);

  let etag: string;
  try {
    etag = await bosClient.putObject(bucketName, objectName, buf); //上传object
    logger.info(`put object success, etag:${etag}`); // 打印etag
  } catch (bosResponse) {
    printErrorInfo(bosResponse, "put object fail");
  }

  let args = new ListObjectsArgs();
  let listObjectsResult: ListObjectsResult;

  try {
    listObjectsResult = await bosClient.listObjects(bucketName, args);
    // 遍历所有Object
    for (let objectInfo of listObjectsResult.contents as ObjectSummaryType[]) {
      logger.info(`name: ${objectInfo.key} size: ${objectInfo.size} storageClass: ${objectInfo.storageClass}`);
    }
  } catch (bosResponse) {
    printErrorInfo(bosResponse, "list objects error");
  }

  let getObjectResult: GetObjectResult;

  try {
    getObjectResult = await bosClient.getObject(bucketName, objectName);
    logger.info(`get object success, etag:${JSON.stringify(getObjectResult.etag)}`);
  } catch (bosResponse) {
    printErrorInfo(bosResponse, "get object fail");
  }
}
```

配置BosClient

如果您需要配置BosClient的一些细节的参数，可以在构造BosClient的时候传入ClientOptions对象。 ClientOptions是BOS服务的配置类，可以通过ClientOptions对基本网络参数进行设置。

- 示例代码

  let clientOptions = new ClientOptions(); //创建ClientOptions对象 clientOptions.calcMd5On = true; //上传object的时候本地计算MD5 clientOptions.endpoint = "bj.bcebos.com"; // 设置域名 clientOptions.multiPartSize = 6164480; //分块上传的时候每一块的大小

- 参数说明

参数	说明
endpoint	BOS服务在各个区域的域名地址
userAgent	用户代理，指HTTP的User-Agent头
signExpireSeconds	签名有效时长，单位是s
connectTimeout	建立连接的超时时间（单位：毫秒），缺省值为10000
multiPartSize	分块大小，默认是5242880
calcMd5On	上传object的时候是否在本地计算MD5值，默认是false。注意：设置为true时将会增加上传的时间消耗

配置HTTPS协议访问BOS

endpoint设置为

https://开头即可

- 示例代码

let clientOptions = new ClientOptions(); //创建ClientOptions对象 clientOptions.endpoint = "https://bj.bcebos.com"; // 设置域名

Bucket管理

新建Bucket

基本流程

1. 创建BosClient类的实例。
2. 执行BosClient.putBucket方法，你需要提供Bucket的名字。

示例代码

```
 let bosClient = new BosClient(credential, clientOptions); // 创建BosClient实例
let location: string;
try {
    location = await bosClient.putBucket(bucketName);
} catch (bosResponse) {
    logger.error(`errCode: ${bosResponse.error.code}`)
    logger.error(`requestId: ${bosResponse.error.requestId}`)
    logger.error(`errMessage: ${bosResponse.error.message}`)
    logger.error(`statusCode: ${bosResponse.statusCode}`)
}
```

注意

1. 由于Bucket的名称在所有区域中是唯一的，所以需要保证BucketName不与其他所有区域上的BucketName相同。
2. Bucket的命名有以下规范：
    - 只能包括小写字母，数字，短横线（-）。
    - 必须以小写字母或者数字开头。
    - 长度必须在4-63字节之间。
3. 通过上述代码创建的bucket，权限是私有读写，存储类型是标准类型（Standard）

查看Bucket列表

基本流程

1. 创建BosClient类的实例。
2. 执行BosClient.listBuckets()方法，会返回ListBucketsResult类的实例。 **示例代码** 如下代码可以列出用户所有的Bucket：

```
 let listBucketResult: ListBucketsResult;
try {
    listBucketResult = await bosClient.listBuckets();
    for (let bucketInfo of listBucketResult.buckets as BucketSummaryType[]) {
        logger.info(`bucketName: ${bucketInfo.name}, location: ${bucketInfo.location}`)
    }
} catch (bosResponse) {
    logger.error(`errCode: ${bosResponse.error.code}`)
    logger.error(`requestId: ${bosResponse.error.requestId}`)
    logger.error(`errMessage: ${bosResponse.error.message}`)
    logger.error(`statusCode: ${bosResponse.statusCode}`)
}
```

返回参数 listBuckets方法返回的解析类中可供调用的参数如下：

参数	说明
owner	Bucket Owner信息
+id	Bucket Owner的用户ID
+displayName	Bucket Owner的名称
buckets	存放多个Bucket信息的容器
bucket	存放一个Bucket信息的容器
+name	Bucket名称
+createDate	Bucket创建时间
+location	Bucket所属的区域

查看Bucket所属的区域

基本流程

1. 创建BosClient类的实例。
2. 执行BosClient.testGetBucketLocation方法。

示例代码

```
  let location: string;
try {
    location = await bosClient.getBucketLocation(bucketName);
    logger.info(`location: ${location}`);
} catch (bosResponse) {
    logger.error(`errCode: ${bosResponse.error.code}`)
    logger.error(`requestId: ${bosResponse.error.requestId}`)
    logger.error(`errMessage: ${bosResponse.error.message}`)
    logger.error(`statusCode: ${bosResponse.statusCode}`)
}
```

判断Bucket是否存在

基本流程

1. 创建BosClient类的实例。

2. 执行BosClient.headBucket方法。

示例代码

```
  try {
    await bosClient.headBucket(bucketName);
} catch (bosResponse) {
    // bucket不存在的话返回404，需要在异常中判断
    if (bosResponse.statusCode = 404) {
        logger.info("this bucket is not exist");
    } else {
        logger.error(`errCode: ${bosResponse.error.code}`)
        logger.error(`requestId: ${bosResponse.error.requestId}`)
        logger.error(`errMessage: ${bosResponse.error.message}`)
        logger.error(`statusCode: ${bosResponse.statusCode}`)
    }
}
```

Bucket存储类型设置与查询

基本流程

1. 创建BosClient类的实例。

2. 执行BosClient.putBucketStorageClass方法可以设置存储类型，执行BosClient.getBucketStorageClass方法可以查询存储类型。

示例代码

```
  // 设置存储类型为COLD
let storageClass = "COLD";
try {
    await bosClient.putBucketStorageClass(<bucketName>, <storageClass>);
    logger.info(`put bucket storage class success`);
} catch (bosResponse) {
    logger.error(`errCode: ${bosResponse.error.code}`)
    logger.error(`requestId: ${bosResponse.error.requestId}`)
    logger.error(`errMessage: ${bosResponse.error.message}`)
    logger.error(`statusCode: ${bosResponse.statusCode}`)
}
// 查询存储类型
try {
    storageClass = await bosClient.getBucketStorageClass(<bucketName>);
    logger.info(`get bucket storage class success, ${storageClass}`);
} catch (bosResponse) {
    logger.error(`errCode: ${bosResponse.error.code}`)
    logger.error(`requestId: ${bosResponse.error.requestId}`)
    logger.error(`errMessage: ${bosResponse.error.message}`)
    logger.error(`statusCode: ${bosResponse.statusCode}`)
}
```

请求参数

参数说明

`bucketName`：存储桶名称

`storageClass`: 存储类型，支持： `STANDARD`（标准存储类型）、`STANDARD_IA`（低频存储）、`ARCHIVE`（归档存储）、`COLD`（冷存储）

Bucket权限控制

Bucket权限控制

设置Bucket的访问权限

- 基本流程

    1. 创建BosClient。

    2. 执行putBucketAcl方法，putBucketAcl方法无返回值，不抛出异常即设置成功，`putBucketAcl`方法请求参数：

参数	说明
BucketName	存储桶名称
CannedAcl	Bucket访问权限，可取private，public-read或public-read-write
aclStr	acl文件内容，详见上传acl文件方式的权限控制

- 示例代码

```
  // 使用cannedAcl的方式设置Bucket访问权限为private
try {
    await bosClient.putBucketAcl(bucketName, "private", undefined)
    logger.info(`put bucket acl success`);
} catch (bosResponse) {
    logger.error(`errCode: ${bosResponse.error.code}`)
    logger.error(`requestId: ${bosResponse.error.requestId}`)
    logger.error(`errMessage: ${bosResponse.error.message}`)
    logger.error(`statusCode: ${bosResponse.statusCode}`)
}

//使用acl文件内容的方式设置访问权限
let aclStr = `{
        "accessControlList":[
          {
            "grantee":[{
              "id":"123456"
            }],
            "permission":["FULL_CONTROL"]
          }
        ]
      }`;
  try {
    await bosClient.putBucketAcl(bucketName, undefined, aclStr)
    logger.info(`put bucket acl success`);
  } catch (bosResponse) {
    logger.error(`errCode: ${bosResponse.error.code}`)
    logger.error(`requestId: ${bosResponse.error.requestId}`)
    logger.error(`errMessage: ${bosResponse.error.message}`)
    logger.error(`statusCode: ${bosResponse.statusCode}`)
  }
```

查看Bucket的权限

- 基本流程
  1. 创建BosClient。
  2. 执行getBucketAcl()方法。
- 示例代码

```
  let bucketAcl: GetBucketAclResult;
try {
    bucketAcl = await bosClient.getBucketAcl(bucketName)
    logger.info(`get bucket acl success, content: ${JSON.stringify(bucketAcl)}`)
} catch (bosResponse) {
    logger.error(`errCode: ${bosResponse.error.code}`)
    logger.error(`requestId: ${bosResponse.error.requestId}`)
    logger.error(`errMessage: ${bosResponse.error.message}`)
    logger.error(`statusCode: ${bosResponse.statusCode}`)
}
```

删除Bucket

基本流程

1. 创建BosClient类的实例。
2. 执行BosClient.deleteBucket方法。

示例代码

```
 try {
   await bosClient.deleteBucket(bucketName);
   logger.info(`delete ${bucketName} success`);
} catch (bosResponse) {
   logger.error(`errCode: ${bosResponse.error.code}`)
   logger.error(`requestId: ${bosResponse.error.requestId}`)
   logger.error(`errMessage: ${bosResponse.error.message}`)
   logger.error(`statusCode: ${bosResponse.statusCode}`)
}
```

注意

- 如果Bucket不为空（即Bucket中有Object和未完成的三步上传Part存在），则Bucket无法被删除，必须清空Bucket后才能成功删除。
- 在删除前确认该Bucket没有开通跨区域复制，不是跨区域复制规则中的源Bucket或目标Bucket，否则不能删除。

文件管理

上传Object

最简单的上传

基本流程

1. 创建BosClient 类的实例。
2. 调用BOSClient.putObject()方法，支持将二进制串上传Object.
3. 上传成功后即返回ETag.

示例代码

```
  let etag: string;
try {
    etag = await bosClient.putObject(<bucketName>, <objectName>, <buf>);
    logger.info(`put object success, etag:${etag}`);
} catch (bosResponse) {
    logger.error(`errCode: ${bosResponse.error.code}`)
    logger.error(`requestId: ${bosResponse.error.requestId}`)
    logger.error(`errMessage: ${bosResponse.error.message}`)
    logger.error(`statusCode: ${bosResponse.statusCode}`)
}
```

参数说明

参数	说明
bucketName	存储桶名称
objectName	object名称
buf	二进制流，类型为ArrayBuffer

设定Object的Http Header或者标签

BOS支持您在上传object时设定Http Header。

基本流程

1. 设置BasicMeta类以及TagConf类的实例.

2. 根据需要设置不同的Header以及标签.

3. 将设定后的meta作为参数加入putObject()中。

示例代码

```
  let args = new BasicMeta();
let userMeta = new TreeMap<string, string>();
userMeta.set("region", "bj");
userMeta.set("type", "test");
args.userMeta = userMeta; // 用户自定义的元数据，但所有的User Meta总大小不能超过2KB。
args.cacheControl = "private"; // 下载Object的Cache设置，常见的可取值为private、no-cache、max-age、must-revalidate
args.contentDisposition = "inline"; // 设置浏览器是否下载，可取值为inline、attachment; filename="download.txt"
args.storageClass = "COLD"; // 存储类型
args.expires = "300";  // 用于设置下载Object时的缓存失效时间，如果不做时间设置，BOS则会默认设置缓存失效时间为三天。
args.trafficLimit = 8192000; // 单链接限速，取值范围为819200 ~ 838860800，单位为bit/s，即100KB/s~100MB/s
let tagConf = new TagConf();
let tag1 = new Tag(); // 标签设置
tag1.tagKey = "a";
tag1.tagValue = "b";
let tag2 = new Tag();
tag1.tagKey = "c";
tag1.tagValue = "d";
tagConf.tags = [tag1, tag2];
try {
    await bosClient.putObject(<bucketName>, <objectName>, <buf>, <args>, <tagConf>);
} catch (bosResponse) {
    logger.error(`errCode: ${bosResponse.error.code}`)
    logger.error(`requestId: ${bosResponse.error.requestId}`)
    logger.error(`errMessage: ${bosResponse.error.message}`)
    logger.error(`statusCode: ${bosResponse.statusCode}`)
}
```

Append方式上传Object

BOS支持AppendObject，即以追加写的方式上传文件，适用场景如日志追加及直播等实时视频文件上传。通过AppendObject操作创建的Object类型为Appendable Object，可以对该Object追加数据；而通过PutObject上传的Object是Normal Object，不可进行数据追加写。AppendObject大小限制为0~5GB。

示例代码

```
  // 从文件中读取数据
// 第一次append
let appendObjectResult: AppendObjectResult;
try {
    // 首次append
    appendObjectResult = await bosClient.appendObject(<bucketName>, <objectName>, <buf>);
    logger.info(`create append object success`);
    let args = new AppendObjectArgs(); //下一次追加写的参数
    for (let i = 0; i < 100; ++i) {
      args.offset = appendObjectResult.nextAppendOffset as number; // 本次追加写的偏移量
      try {
        appendObjectResult = await bosClient.appendObject(<bucketName>, <objectName>, <buf>, <args>);
        logger.info(`append object success, ${i} times`);
      } catch (bosResponse) {
        logger.error(`errCode: ${bosResponse.error.code}`)
        logger.error(`requestId: ${bosResponse.error.requestId}`)
        logger.error(`errMessage: ${bosResponse.error.message}`)
        logger.error(`statusCode: ${bosResponse.statusCode}`)
      }
    }
} catch (bosResponse) {
    logger.error(`errCode: ${bosResponse.error.code}`)
    logger.error(`requestId: ${bosResponse.error.requestId}`)
    logger.error(`errMessage: ${bosResponse.error.message}`)
    logger.error(`statusCode: ${bosResponse.statusCode}`)
}
```

参数说明

参数	说明
bucketName	存储桶名称
objectName	object名称
buf	二进制流，类型为ArrayBuffer
args	追加写的参数， AppendObjectArgs类的实例

完整代码

```
 import fs from '@ohos.file.fs';
import TreeMap from '@ohos.util.TreeMap';
import { logger, Credential, BosClient, ClientOptions} from "bos"
import {  BasicMeta, TagConf, Tag, AppendObjectResult, AppendObjectArgs } from "bos/src/main/ets/bos/api/DataType"

let credential = new Credential(AccessKeyID, SecretAccessKey, Token); //STS返回的临时AK/SK及Token
let clientOptions = new ClientOptions();
clientOptions.endpoint = "bj.bcebos.com";  //传入Bucket所在区域域名
let bosClient = new BosClient(credential, clientOptions); // 创建BosClient
let bucketName = "test-bucket-harmony";
let objectName = "test-put-object";
// 创建一个临时文件
let cacheDir = getContext().cacheDir;
let path = cacheDir + '/test.txt';
let file = fs.openSync(path, fs.OpenMode.CREATE | fs.OpenMode.READ_WRITE | fs.OpenMode.TRUNC)
fs.writeSync(file.fd, "Hello BOS");
fs.fsyncSync(file.fd);
fs.closeSync(file.fd);

file = fs.openSync(path, fs.OpenMode.READ_ONLY); // 使用同步的方式打开文件
let stat = fs.lstatSync(path); //获取文件的信息
let buf = new ArrayBuffer(stat.size); //构造ArrayBuffer
fs.readSync(file.fd, buf); // 读取文件内容
fs.fsyncSync(file.fd);
fs.closeSync(file.fd);
let etag: string;
try {
    etag = await bosClient.putObject(bucketName, objectName, buf); //上传object
    logger.info(`put object success, etag:${etag}`); // 打印etag
} catch (bosResponse) {
    logger.error(`errCode: ${bosResponse.error.code}`)
    logger.error(`requestId: ${bosResponse.error.requestId}`)
    logger.error(`errMessage: ${bosResponse.error.message}`)
    logger.error(`statusCode: ${bosResponse.statusCode}`)
}

let args = new BasicMeta();
let userMeta = new TreeMap<string, string>();
userMeta.set("region", "bj");
userMeta.set("type", "test");
args.userMeta = userMeta; // 用户自定义的元数据，但所有的User Meta总大小不能超过2KB。
args.cacheControl = "private"; // 下载Object的Cache设置，常见的可取值为private、no-cache、max-age、must-revalidate
args.contentDisposition = "inline"; // 设置浏览器是否下载，可取值为inline、attachment; filename="download.txt"
args.storageClass = "COLD"; // 存储类型
args.expires = "300";  // 用于设置下载Object时的缓存失效时间，如果不做时间设置，BOS则会默认设置缓存失效时间为三天。
args.trafficLimit = 8192000; // 单链接限速，取值范围为819200 ~ 838860800，单位为bit/s，即100KB/s~100MB/s
let tagConf = new TagConf();
let tag1 = new Tag(); // 标签设置
tag1.tagKey = "a";
tag1.tagValue = "b";
let tag2 = new Tag();
tag1.tagKey = "c";
tag1.tagValue = "d";
tagConf.tags = [tag1, tag2];
try {
    await bosClient.putObject(bucketName, objectName, buf, args, tagConf);
} catch (bosResponse) {
    logger.error(`errCode: ${bosResponse.error.code}`)
    logger.error(`requestId: ${bosResponse.error.requestId}`)
    logger.error(`errMessage: ${bosResponse.error.message}`)
    logger.error(`statusCode: ${bosResponse.statusCode}`)
}

// 追加写的方式上传object
let appendObjectResult: AppendObjectResult;
try {
    // 首次append
    appendObjectResult = await bosClient.appendObject(bucketName, objectName, buf);
    logger.info(`create append object success`);
    let args = new AppendObjectArgs(); //下一次追加写的参数
    // 测试多次append
    for (let i = 0; i < 100; ++i) {
        args.offset = appendObjectResult.nextAppendOffset as number; // 本次追加写的偏移量
        try {
            appendObjectResult = await bosClient.appendObject(bucketName, objectName, buf, args);
            logger.info(`append object success, ${i} times`);
        } catch (bosResponse) {
            logger.error(`errCode: ${bosResponse.error.code}`)
            logger.error(`requestId: ${bosResponse.error.requestId}`)
            logger.error(`errMessage: ${bosResponse.error.message}`)
            logger.error(`statusCode: ${bosResponse.statusCode}`)
        }
    }
} catch (bosResponse) {
    logger.error(`errCode: ${bosResponse.error.code}`)
    logger.error(`requestId: ${bosResponse.error.requestId}`)
    logger.error(`errMessage: ${bosResponse.error.message}`)
    logger.error(`statusCode: ${bosResponse.statusCode}`)
}
```

查看Bucket中的Object

简单查询

查看Bucket中Object列表。

基本流程

1. 创建BosClient类的实例。

2. 执行BosClient.listObjects(bucketName)方法，会返回ListObjectsResult类的实例。

3. 返回信息见ListObjectsResult对象的各个成员。

示例代码

```
 let listObjectsResult: ListObjectsResult;
try {
    listObjectsResult = await bosClient.listObjects(bucketName);
    for (let objectInfo of listObjectsResult.contents as ObjectSummaryType[]) {
        logger.info(`name: ${objectInfo.key} size: ${objectInfo.size} storageClass: ${objectInfo.storageClass}`);
    }
} catch (bosResponse) {
    logger.error(`errCode: ${bosResponse.error.code}`)
    logger.error(`requestId: ${bosResponse.error.requestId}`)
    logger.error(`errMessage: ${bosResponse.error.message}`)
    logger.error(`statusCode: ${bosResponse.statusCode}`)
}
```

说明：

- 默认情况下，如果Bucket中的Object数量大于1000，则只会返回1000个Object，并且返回结果中IsTruncated值为True，并返回NextMarker做为下次读取的起点。
- 若想获取更多的Object，可以使用Marker参数分次读取。

完整代码

```
 import { logger, Credential, BosClient, ClientOptions} from "bos"
import { ListObjectsResult, ObjectSummaryType } from "bos/src/main/ets/bos/api/DataType"

let credential = new Credential(AccessKeyID, SecretAccessKey, Token); //STS返回的临时AK/SK及Token
let clientOptions = new ClientOptions();
clientOptions.endpoint = "bj.bcebos.com";  //传入Bucket所在区域域名
let bosClient = new BosClient(credential, clientOptions); // 创建BosClient
let bucketName = "test-bucket-harmony";

let listObjectsResult: ListObjectsResult;
try {
    listObjectsResult = await bosClient.listObjects(bucketName);
    for (let objectInfo of listObjectsResult.contents as ObjectSummaryType[]) {
        logger.info(`name: ${objectInfo.key} size: ${objectInfo.size} storageClass: ${objectInfo.storageClass}`);
    }
} catch (bosResponse) {
    logger.error(`errCode: ${bosResponse.error.code}`)
    logger.error(`requestId: ${bosResponse.error.requestId}`)
    logger.error(`errMessage: ${bosResponse.error.message}`)
    logger.error(`statusCode: ${bosResponse.statusCode}`)
}
```

扩展查询

用户可以通过设置ListObjectsArgs参数来完成更多扩展查询操作设置。ListObjectsArgs中可以设置的扩展参数如下：

参数名称	说明	默认值
delimiter	是一个分隔符，用来对objectKey进行分层。通常与prefix配合在查询模拟文件夹中使用。从prefix开始到第一次出现delimiter字符之间的objectKey称为：commonPrefixes	-
marker	是一个字符串，用来设定返回结果的起始位置。设定Marker值之后，返回的Object会从Marker值之后按字母排序开始返回。	-
maxKeys	设定此次返回Object的最大个数，不可超过1000。	-
prefix	设定objectKey的前缀，前缀是指objectKey包含并以Prefix的值作为开始。通常与Delimiter配合在查询模拟文件夹中使用。	-

示例代码

示例一：

```
 import { logger, Credential, BosClient, ClientOptions} from "bos"
import { ListObjectsResult, ObjectSummaryType, ListObjectsArgs  } from "bos/src/main/ets/bos/api/DataType"

let credential = new Credential(AccessKeyID, SecretAccessKey, Token); //STS返回的临时AK/SK及Token
let clientOptions = new ClientOptions();
clientOptions.endpoint = "bj.bcebos.com";  //传入Bucket所在区域域名
let bosClient = new BosClient(credential, clientOptions); // 创建BosClient
let bucketName = "test-bucket-harmony";

let args = new ListObjectsArgs();
args.delimiter = "/";
args.maxKeys = 3;
args.prefix = "fun";
args.marker = "123";
let listObjectsResult: ListObjectsResult;
try {
    listObjectsResult = await bosClient.listObjects(bucketName, args);
    for (let objectInfo of listObjectsResult.contents as ObjectSummaryType[]) {
        logger.info(`name: ${objectInfo.key} size: ${objectInfo.size} storageClass: ${objectInfo.storageClass}`);
    }
} catch (bosResponse) {
    logger.error(`errCode: ${bosResponse.error.code}`)
    logger.error(`requestId: ${bosResponse.error.requestId}`)
    logger.error(`errMessage: ${bosResponse.error.message}`)
    logger.error(`statusCode: ${bosResponse.statusCode}`)
}
```

示例二：使用nextmarker完整示例。

```
 import { logger, Credential, BosClient, ClientOptions} from "bos"
import { ListObjectsResult, ObjectSummaryType, ListObjectsArgs } from "bos/src/main/ets/bos/api/DataType"


let credential = new Credential(AccessKeyID, SecretAccessKey, Token); //STS返回的临时AK/SK及Token
let clientOptions = new ClientOptions();
clientOptions.endpoint = "bj.bcebos.com";  //传入Bucket所在区域域名
let bosClient = new BosClient(credential, clientOptions); // 创建BosClient
let bucketName = "test-bucket-harmony";

let args = new ListObjectsArgs();
args.delimiter = "/";
args.maxKeys = 3;
args.prefix = "test";
let listObjectsResult: ListObjectsResult;
try {
    listObjectsResult = await bosClient.listObjects(bucketName, args);
    for (let objectInfo of listObjectsResult.contents as ObjectSummaryType[]) {
        logger.info(`name: ${objectInfo.key} size: ${objectInfo.size} storageClass: ${objectInfo.storageClass}`);
    }
    if (listObjectsResult.isTruncated) {
        args.marker = listObjectsResult.nextMarker as string;
    }
    listObjectsResult = await bosClient.listObjects(bucketName, args);
    for (let objectInfo of listObjectsResult.contents as ObjectSummaryType[]) {
        logger.info(`name: ${objectInfo.key} size: ${objectInfo.size} storageClass: ${objectInfo.storageClass}`);
    }
} catch (bosResponse) {
    logger.error(`errCode: ${bosResponse.error.code}`)
    logger.error(`requestId: ${bosResponse.error.requestId}`)
    logger.error(`errMessage: ${bosResponse.error.message}`)
    logger.error(`statusCode: ${bosResponse.statusCode}`)
}
```

查询prefix

由于BOS本身是一个KV的存储系统，所以原则上并不会存在"文件夹"的概念，但您可以通过 `delimiter` 和 `prefix` 参数的配合进行文件夹功能模拟。

假设Bucket中有5个文件：bos.jpg，fun/，fun/test.jpg，fun/movie/001.avi，fun/movie/007.avi，可以把 "/" 符号作为分隔符模拟文件夹。

递归列出prefix下所有文件

可以通过设置 `prefix` 参数来获取某个模拟文件夹下所有的文件：

```
 let args = new ListObjectsArgs();
args.prefix = "fun/";

let listObjectsResult: ListObjectsResult;
try {
    listObjectsResult = await bosClient.listObjects(bucketName, args);
    for (let objectInfo of listObjectsResult.contents as ObjectSummaryType[]) {
        logger.info(`name: ${objectInfo.key} size: ${objectInfo.size} storageClass: ${objectInfo.storageClass}`);
    }
} catch (bosResponse) {
    logger.error(`errCode: ${bosResponse.error.code}`)
    logger.error(`requestId: ${bosResponse.error.requestId}`)
    logger.error(`errMessage: ${bosResponse.error.message}`)
    logger.error(`statusCode: ${bosResponse.statusCode}`)
}
```

输出：

```
 name: fun/ size: 0 storageClass: COLD
name: fun/movie/001.avi size: 0 storageClass: COLD
name: fun/movie/007.avi size: 0 storageClass: COLD
name: fun/test.jpg size: 0 storageClass: COLD
```

查看prefix下的文件和子文件夹

在 prefix 和 delimiter 结合的情况下，可以列出模拟文件夹下的文件和子文件夹：

```
 import { logger, Credential, BosClient, ClientOptions} from "bos"
import { ListObjectsResult, ObjectSummaryType, ListObjectsArgs, PrefixType } from "bos/src/main/ets/bos/api/DataType"

let credential = new Credential(AccessKeyID, SecretAccessKey, Token); //STS返回的临时AK/SK及Token
let clientOptions = new ClientOptions();
clientOptions.endpoint = "bj.bcebos.com";  //传入Bucket所在区域域名
let bosClient = new BosClient(credential, clientOptions); // 创建BosClient
let bucketName = "test-bucket-harmony";

let args = new ListObjectsArgs();
args.delimiter = "/";
args.prefix = "fun/";
let listObjectsResult: ListObjectsResult;
try {
    listObjectsResult = await bosClient.listObjects(bucketName, args);
    for (let objectInfo of listObjectsResult.contents as ObjectSummaryType[]) {
        logger.info("objects:");
        logger.info(`name: ${objectInfo.key} size: ${objectInfo.size} storageClass: ${objectInfo.storageClass}`);
    }
    logger.info("commonPrefix:");
    // 遍历所有CommonPrefix
    for(let prefix of listObjectsResult.commonPrefixes as PrefixType[]) {
      logger.info(`commonPrefix: ${prefix.prefix}`);
    }
} catch (bosResponse) {
    logger.error(`errCode: ${bosResponse.error.code}`)
    logger.error(`requestId: ${bosResponse.error.requestId}`)
    logger.error(`errMessage: ${bosResponse.error.message}`)
    logger.error(`statusCode: ${bosResponse.statusCode}`)
}
```

输出：

```
 objects:
name: fun/ size: 0 storageClass: COLD
name: fun/test.jpg size: 0 storageClass: COLD
commonPrefix:
commonPrefix: fun/movie/
```

> **说明**：返回的结果中， objects 的列表中给出的是fun文件夹下的文件。而 commonPrefixs 的列表中给出的是fun文件夹下的所有子文件夹。可以看出 fun/movie/001.avi ， fun/movie/007.avi 两个文件并没有被列出来，因为它们属于 fun 文件夹下的 movie 子文件夹下的文件。

获取Object

简单的获取Object

用户可以通过如下代码将Object读取到内存中。

基本流程

1. 创建BosClient类的实例。
2. 执行BosClient getObject方法，会返回GetObjectResult实例。
3. 访问GetObjectResult实例的data属性，获取数据。

示例代码

```
 let getObjectResult: GetObjectResult;
try {
    getObjectResult = await bosClient.getObject(bucketName, objectName);
    logger.info(`get object success, etag:${JSON.stringify(getObjectResult.etag)}`);
    // 获取实际的数据
    let data = getObjectResult.data as ArrayBuffer;
} catch (bosResponse) {
    logger.error(`errCode: ${bosResponse.error.code}`)
    logger.error(`requestId: ${bosResponse.error.requestId}`)
    logger.error(`errMessage: ${bosResponse.error.message}`)
    logger.error(`statusCode: ${bosResponse.statusCode}`)
}
```

> **注意：**
>
> - GetObjectResult中存储了object的数据信息(data)以及元数据信息(userMeta、contentType等)
> - 该操作为下载整个object，由于底层使用API的限制，如果object总长度需小于5MB

完整代码

```
 import { logger, Credential, BosClient, ClientOptions} from "bos"
import { GetObjectResult } from "bos/src/main/ets/bos/api/DataType"

let credential = new Credential(AccessKeyID, SecretAccessKey, Token); //STS返回的临时AK/SK及Token
let clientOptions = new ClientOptions();
clientOptions.endpoint = "bj.bcebos.com";  //传入Bucket所在区域域名
let bosClient = new BosClient(credential, clientOptions); // 创建BosClient
let bucketName = "test-harmony-bucket";
let objectName = "test-object";

let getObjectResult: GetObjectResult;
try {
    getObjectResult = await bosClient.getObject(bucketName, objectName);
    logger.info(`get object success, etag:${JSON.stringify(getObjectResult.etag)}`);
    // 获取实际的数据处理
    let data = getObjectResult.data as ArrayBuffer;
} catch (bosResponse) {
    logger.error(`errCode: ${bosResponse.error.code}`)
    logger.error(`requestId: ${bosResponse.error.requestId}`)
    logger.error(`errMessage: ${bosResponse.error.message}`)
    logger.error(`statusCode: ${bosResponse.statusCode}`)
}
```

下载Object的一部分内容

基本流程

1. 创建BosClient类的实例。

2. 执行BosClient getObject方法，并指定ranges参数

3. 访问GetObjectResult实例的data属性，获取数据。

示例代码

```
 //返回第0到第100个字节的数据，包括第100个
let ranges = [0, 100];
try {
    getObjectResult = await bosClient.getObject(bucketName, objectName, ranges);
    let data = getObjectResult.data as ArrayBuffer
    logger.info(`get object success, ranges : 0 - 100 , return length:${JSON.stringify(data.byteLength)} `);
} catch (bosResponse) {
    logger.error(`errCode: ${bosResponse.error.code}`)
    logger.error(`requestId: ${bosResponse.error.requestId}`)
    logger.error(`errMessage: ${bosResponse.error.message}`)
    logger.error(`statusCode: ${bosResponse.statusCode}`)
}
```

> **说明**：用户可以用此功能实现文件的分段下载和断点续传，当object总大小超过5MB时必须指定ranges分多次下载才能成功。

获取object的元数据信息

基本流程

1. 创建BosClient类的实例。

2. 执行BosClient.getObjectMeta方法，获取ObjectMeta实例。

3. 访问ObjectMeta实例中的contentLength、userMeta、storageClass等方法获取不同的元数据信息

示例代码

```
 let objectMeta: ObjectMeta;
try {
    objectMeta = await bosClient.getObjectMeta(bucketName, objectName);
    logger.info(`get object meta success, info : ${JSON.stringify(objectMeta)}`);
} catch (bosResponse) {
    logger.error(`errCode: ${bosResponse.error.code}`)
    logger.error(`requestId: ${bosResponse.error.requestId}`)
    logger.error(`errMessage: ${bosResponse.error.message}`)
    logger.error(`statusCode: ${bosResponse.statusCode}`)
}
```

获取文件的下载链接

您可以通过如下代码获取指定Object的URL，该功能通常用于您将Object的URL临时分享给其他用户的场景。

基本流程

1. 创建BosClient类的实例。

2. 执行BosClient generatePresignedUrl方法。

3. 返回一个object的URL。

示例代码

```
 let expire = 1800;
try {
    let url = await bosClient.generatePresignedUrl(bucketName, objectName, expire);
    logger.info(`url: ${url}`);
} catch (bosResponse) {
    logger.error(`errCode: ${bosResponse.error.code}`)
    logger.error(`requestId: ${bosResponse.error.requestId}`)
    logger.error(`errMessage: ${bosResponse.error.message}`)
    logger.error(`statusCode: ${bosResponse.statusCode}`)
}
```

> **说明**： expire为指定的URL有效时长，时间从当前时间算起，为可选参数，不配置时系统默认值为1800秒。

下载大文件到指定路径

基本流程

1. 创建BosClient类的实例。
2. 执行BosClient downloadSuperFile方法。

完整代码

```
 import fs from '@ohos.file.fs';
import { logger, Credential, BosClient, ClientOptions } from "bos"

let credential = new Credential(AccessKeyID, SecretAccessKey, Token); //STS返回的临时AK/SK及Token
let clientOptions = new ClientOptions();
clientOptions.endpoint = "bj.bcebos.com";  //传入Bucket所在区域域名
let bosClient = new BosClient(credential, clientOptions); // 创建BosClient
let bucketName = "test-harmony-bucket";
let objectName = "superfile.txt";


let cacheDir = getContext().cacheDir;
let path = cacheDir + '/test.txt';

// download
try {
    await bosClient.downloadSuperFile(bucketName, objectName, path + ".copy");
    let stat = fs.lstatSync(path + ".copy");
    logger.info(`download super file success, filesize : ${stat.size}`);
} catch (bosResponse) {
    logger.error(`errCode: ${bosResponse.error.code}`)
    logger.error(`requestId: ${bosResponse.error.requestId}`)
    logger.error(`errMessage: ${bosResponse.error.message}`)
    logger.error(`statusCode: ${bosResponse.statusCode}`)
}
```

> **说明**： 该方法使用单线程通过多次http请求，每次下载5MB，把大文件下载到本地。

Object权限控制

设置Object的访问权限

目前BOS支持两种方式设置ACL。第一种是使用Canned Acl，在PutObjectAcl的时候，通过头域的"x-bce-acl"或者"x-bce-grant-permission'来设置object访问权限，当前可设置的权限包括private和public-read，两种类型的header不可以同时在一个请求中出现。第二种方式是上传一个ACL文件。 详细信息请参考设置Object权限控制。

1. 通过使用头域的"x-bce-acl"来设置object访问权限

```
  // 只能为private，public-read和public-read-write中的一个
 let cannedAcl = "private";
 // cannel acl
 try {
     await bosClient.putObjectAcl(bucketName, objectName, cannedAcl);
     logger.info("put object acl success");
 } catch (bosResponse) {
     logger.error(`errCode: ${bosResponse.error.code}`)
     logger.error(`requestId: ${bosResponse.error.requestId}`)
     logger.error(`errMessage: ${bosResponse.error.message}`)
     logger.error(`statusCode: ${bosResponse.statusCode}`)
 }
```

2. 设置细粒度的权限

```
import { ObjectAclConf } from "bos/src/main/ets/bos/api/DataType"
// 给用户e13b12d0131b4c8bae959df4969387b8授予读权限 let acl = {
  "accessControlList":[
    {
      "grantee":[{
        "id":"e13b12d0131b4c8bae959df4969387b8"
      }],
      "permission":["READ"]
    }
  ]
} let objectAclConf = JSON.parse(acl) as ObjectAclConf;
try { await bosClient.putObjectAcl(bucketName, objectName, undefined, undefined, undefined, objectAclConf); logger.info("put object acl success"); } catch (bosResponse) { logger.error(errCode: ${bosResponse.error.code}) logger.error(requestId: ${bosResponse.error.requestId}) logger.error(errMessage: ${bosResponse.error.message}) logger.error(statusCode: ${bosResponse.statusCode}) }
```

查看Object的权限

示例代码

```
 let objectAclConf: ObjectAclConf;

try {
    objectAclConf = await bosClient.getObjectAcl(bucketName, objectName);
    logger.info(`get object acl success, info : ${JSON.stringify(objectAclConf)}`)
} catch (bosResponse) {
    logger.error(`errCode: ${bosResponse.error.code}`)
    logger.error(`requestId: ${bosResponse.error.requestId}`)
    logger.error(`errMessage: ${bosResponse.error.message}`)
    logger.error(`statusCode: ${bosResponse.statusCode}`)
}
```

getObjectAcl方法返回的解析类中可供调用的参数有：

参数	说明
accessControlList	标识object的权限列表
grantee	标识被授权人
id	被授权人ID
permission	标识被授权人的权限

删除Object的权限

示例代码

```
 try {
    await bosClient.deleteObjectAcl(bucketName, objectName);
    logger.info(`delete object acl success`)
} catch (bosResponse) {
    logger.error(`errCode: ${bosResponse.error.code}`)
    logger.error(`requestId: ${bosResponse.error.requestId}`)
    logger.error(`errMessage: ${bosResponse.error.message}`)
    logger.error(`statusCode: ${bosResponse.statusCode}`)
}
```

拷贝Object

简单拷贝Object

基本流程

1. 创建BosClient 类的实例。

2. 执行BosClient.copyObject()方法。

3. 返回CopyObjectResult类实例，可通过lastModifie和eTag成员查看最后修改时间和eTag.

示例代码

```
 let copyObjectResult: CopyObjectResult;
try {
    let destObjectName = objectName + ".cp"; // copy目的object名称
    copyObjectResult = await bosClient.copyObject(bucketName, destObjectName, bucketName, objectName);
    logger.info(`Copy object success, return : ${JSON.stringify(copyObjectResult)}`);
} catch (bosResponse) {
    logger.error(`errCode: ${bosResponse.error.code}`)
    logger.error(`requestId: ${bosResponse.error.requestId}`)
    logger.error(`errMessage: ${bosResponse.error.message}`)
    logger.error(`statusCode: ${bosResponse.statusCode}`)
}
```

完整示例

```
 import { logger, Credential, BosClient, ClientOptions } from "bos"
import { CopyObjectResult  } from "bos/src/main/ets/bos/api/DataType"

let credential = new Credential(AccessKeyID, SecretAccessKey, Token); //STS返回的临时AK/SK及Token
let clientOptions = new ClientOptions();
clientOptions.endpoint = "bj.bcebos.com";  //传入Bucket所在区域域名
let bosClient = new BosClient(credential, clientOptions); // 创建BosClient
let bucketName = "test-harmony-bucket";
let objectName = "test-object"; // copy源object名称

let copyObjectResult: CopyObjectResult;
try {
    let destObjectName = objectName + ".cp"; // copy目的object名称
    copyObjectResult = await bosClient.copyObject(bucketName, destObjectName, bucketName, objectName);
    logger.info(`Copy object success, return : ${JSON.stringify(copyObjectResult)}`);
} catch (bosResponse) {
    logger.error(`errCode: ${bosResponse.error.code}`)
    logger.error(`requestId: ${bosResponse.error.requestId}`)
    logger.error(`errMessage: ${bosResponse.error.message}`)
    logger.error(`statusCode: ${bosResponse.statusCode}`)
}
```

通过CopyObjectArgs拷贝Object

您也可以通过 CopyObjectArgs 实现object的拷贝。该功能一般用于如下场景：

- 拷贝一个Object但重新设置meta。
- 重置某个现有Object的meta（把源Object和目的Object设置为同一个Object）。

基本流程

1. 创建CopyObjectArgs类的实例，设置metadataDirective/ifMatch/userMeta等参数，含义可见copyObject参数设置。

2. 调用copyObject方法，传入CopyObjectArgs类的实例

示例代码

```
 import TreeMap from '@ohos.util.TreeMap';
import { logger, Credential, BosClient, ClientOptions } from "bos"
import { CopyObjectResult, CopyObjectArgs  } from "bos/src/main/ets/bos/api/DataType"

let credential = new Credential(AccessKeyID, SecretAccessKey, Token); //STS返回的临时AK/SK及Token
let clientOptions = new ClientOptions();
clientOptions.endpoint = "bj.bcebos.com";  //传入Bucket所在区域域名
let bosClient = new BosClient(credential, clientOptions); // 创建BosClient

//拷贝的目的bucket和目的object
let destBucketName = "dest-bucket";
let destObjectName = "dest-object";

//拷贝的源bucket和源object
let srcBucketName = "src-bucket"
let srcObjectName = "src-object";

//设置拷贝的参数
let args:CopyObjectArgs = new CopyObjectArgs();
//如果源object的eTag是设定的值，则发起拷贝
args.ifMatch = "123456";
//设置为replace才可对目的bucket设置userMeta
args.metadataDirective = "replace";
//设置目的object的userMeta
let userMeta = new TreeMap<string, string>();
userMeta.set("region", "bj");
userMeta.set("type", "test");
args.userMeta = userMeta;

let copyObjectResult: CopyObjectResult;
try {
    copyObjectResult = await bosClient.copyObject(destBucketName, destObjectName, srcBucketName, srcObjectName, args);
    logger.info(`Copy object success, return : ${JSON.stringify(copyObjectResult)}`);
} catch (bosResponse) {
    logger.error(`errCode: ${bosResponse.error.code}`)
    logger.error(`requestId: ${bosResponse.error.requestId}`)
    logger.error(`errMessage: ${bosResponse.error.message}`)
    logger.error(`statusCode: ${bosResponse.statusCode}`)
}
```

Object的分块上传

分块上传的场景

除了通过putObject()方法上传文件到BOS以外，BOS还提供了另外一种上传模式：分块上传（Multipart Upload）。用户可以在如下的应用场景内（但不仅限于此），使用分块上传模式，如：

1. 需要支持断点上传。

2. 上传超过5GB大小的文件。

3. 网络条件较差，和BOS的服务器之间的连接经常断开。

4. 需要流式地上传文件。

5. 上传文件之前，无法确定上传文件的大小。

Multipart Upload分块上传流程

假设有一个文件，本地路径为/path/to/file.zip，由于文件比较大，使用分块上传其传输到BOS中。 基本流程：

1. 初始化Multipart Upload。

2. 上传分块。

3. 完成分块上传。

初始化Multipart Upload

使用initiateMultipartUpload方法来初始化一个分块上传事件：

```
 let bucketName = "test-harmony-bucket"
let objectName = "test-multi-upload-object"
let args = new InitiateMultipartUploadArgs();
// 设置上传的object为标准存储类型
args.storageClass = "STANDARD";
// initialization
let initMultiUploadResult: InitiateMultipartUploadResult;
try {
  // contentType参数设置为undefined占位，使用默认值
  initMultiUploadResult = await bosClient.initiateMultipartUpload(bucketName, objectName, undefined, args);
  logger.info(`init multi upload success, info: ${JSON.stringify(initMultiUploadResult)}`)
} catch(bosResponse) {
  logger.error(`errCode: ${bosResponse.error.code}`)
  logger.error(`requestId: ${bosResponse.error.requestId}`)
  logger.error(`errMessage: ${bosResponse.error.message}`)
  logger.error(`statusCode: ${bosResponse.statusCode}`)
}
```

> **说明**：initiateMultipartUpload的返回结果中含有UploadId，它是区分分块上传事件的唯一标识，在后面的操作中，我们将用到它。

上传分块

```
 let stat = fs.lstatSync("/path/to/file.zip");
let file = fs.openSync(filePath, fs.OpenMode.READ_ONLY);
if (stat.size < MAX_SINGLE_OBJECT_SIZE) { //BOS最大文件大小为48.8TB
  return Promise.reject(fillBceError(`The file size exceeds ${MAX_SINGLE_OBJECT_SIZE}`));
}
// 如果文件小于一个块的大小，就不需要使用分块上传
if (stat.size < MIN_MULTIPART_SIZE || this._clientImpl.clientOptions.multiPartSize < MIN_MULTIPART_SIZE) {
  return Promise.reject(fillBceError(`multipart size should not be less than ${MIN_MULTIPART_SIZE}`));
}
// 计算每块的大小的尺寸
let partSize = (this._clientImpl.clientOptions.multiPartSize + MULTIPART_ALIGN - 1) / MULTIPART_ALIGN * MULTIPART_ALIGN;
let partNum = Math.floor((stat.size + partSize - 1) / partSize);
// 如果块的数量超过10000，将重新计算每块的大小，不再使用默认的块大小
if (partNum > MAX_PART_NUMBER) {
  partSize = (stat.size + MAX_PART_NUMBER - 1) / MAX_PART_NUMBER;
  partSize = (partSize + MULTIPART_ALIGN - 1) / MULTIPART_ALIGN * MULTIPART_ALIGN;
  partNum = Math.floor((stat.size + partSize - 1) / partSize);
}
logger.debug(`start to upload super file, total parts: ${partNum}, part size: ${partSize}`);
let args = new InitiateMultipartUploadArgs();
if (storageClass && isValidStorageClass(storageClass)) {
  args.storageClass = storageClass;
}
// 使用初始化分块上传的结果返回的uploadId
let uploadId = initMultiUploadResult.uploadId as string;
// group task by partNum
let completeArgs = new CompleteMultipartUploadArgs();
completeArgs.partInfo = new PartInfo();
completeArgs.partInfo.parts = [];
let partIndex = 1;
// 依次上传每一块
while (partIndex <= partNum) {
    logger.debug(`upload part: ${partIndex}`);
    let uploadSize = partSize;
    let offset = (partIndex - 1) * partSize;
    if (uploadSize > stat.size - offset) {
        uploadSize = stat.size - offset;
    }
    let data = new ArrayBuffer(uploadSize);
    fs.readSync(file.fd, data, {offset: offset, length: uploadSize});
    try {
        let etag = await bosClient.uploadPart(bucketName, objectName, uploadId, partIndex, data);
        // 每块上传成功后都需要记录eTag信息和partNumber信息
        let uploadInfo = new UploadInfoType();
        uploadInfo.partNumber = partIndex;
        uploadInfo.eTag = etag;
        completeArgs.partInfo.parts.push(uploadInfo);
    } catch (bosResponse) { // 如果上传过程有失败的情况
        logger.error(`upload error, info: ${JSON.stringify(bosResponse.error)}`)
        await bosClient.abortMultipartUpload(bucketName, objectName, uploadId); //放弃本次分块上传
        fs.closeSync(file);
        return Promise.reject(bosResponse);
    }
    partIndex++;
}
fs.closeSync(file.fd); // 关闭本地文件
```

注意：

- UploadPart 方法要求Part大小是1MB的整数倍或大于5MB。但是Upload Part接口并不会立即校验上传Part的大小；只有当Complete Multipart Upload的时候才会校验。
- 为了保证数据在网络传输过程中不出现错误，建议您在UploadPart后，使用每个分块BOS返回的Content-MD5值分别验证已上传分块数据的正确性。当所有分块数据合成一个Object后，不再含MD5值。
- Part号码的范围是1~10000。如果超出这个范围，BOS将返回InvalidArgument的错误码。
- 每次上传Part之后，BOS的返回结果会包含一个 PartETag 对象，它是上传块的ETag与块编号（PartNumber）的组合，在后续完成分块上传的步骤中会用到它，因此需要将其保存起来。一般来讲这些 PartETag 对象将被保存到List中。

完成分块上传

```
 let completeMultiUploadResult:CompleteMultipartUploadResult;
try {
    completeMultiUploadResult = await bosClient.completeMultipartUpload(bucketName, objectName, uploadId, completeArgs);
} catch(bosResponse) {
    logger.error(`complete multipart fail, info: ${JSON.stringify(bosResponse.error)}`)
    await bosClient.abortMultipartUpload(bucketName, objectName, uploadId);
}
```

完整示例

该部分已经封装到BosClient.uploadSuperFile方法中，用户可以直接调用即可：

```
 import { logger, Credential, BosClient, ClientOptions } from "bos"
import { CompleteMultipartUploadResult  } from "bos/src/main/ets/bos/api/DataType"

let credential = new Credential(AccessKeyID, SecretAccessKey, Token); //STS返回的临时AK/SK及Token
let clientOptions = new ClientOptions();
clientOptions.endpoint = "bj.bcebos.com";  //传入Bucket所在区域域名
let bosClient = new BosClient(credential, clientOptions); // 创建BosClient

let bucketName = "test-harmony-bucket"
let objectName = "multi_upload_object.txt"
let path = "/path/to/file.zip"

let result: CompleteMultipartUploadResult;

try {
    result = await bosClient.uploadSuperFile(bucketName, objectName, path);
    logger.info(`upload super file success, result : ${JSON.stringify(result)}`);
} catch (bosResponse) {
    logger.error(`errCode: ${bosResponse.error.code}`)
    logger.error(`requestId: ${bosResponse.error.requestId}`)
    logger.error(`errMessage: ${bosResponse.error.message}`)
    logger.error(`statusCode: ${bosResponse.statusCode}`)
}
```

取消分块上传

用户可以使用abortMultipartUpload方法取消分块上传。

```
 try {
    await bosClient.abortMultipartUpload(bucketName, objectName, uploadId);
} catch(bosResponse) {
    logger.error(`errCode: ${bosResponse.error.code}`)
    logger.error(`requestId: ${bosResponse.error.requestId}`)
    logger.error(`errMessage: ${bosResponse.error.message}`)
    logger.error(`statusCode: ${bosResponse.statusCode}`)
}
```

获取未完成的分块上传

用户可以使用`listMultipartUploads`方法获取Bucket内未完成的分块上传事件。

示例代码

```
 try {
let listMultipartUploadsResult = await bosClient.listMultipartUploads(bucketName);
    for (let multiUpload of listMultipartUploadsResult.uploads as ListMultipartUploadsType[]) {
        logger.info(`key: ${multiUpload.key}, uploadId: ${multiUpload.uploadId}`);
    }
} catch(bosResponse) {
    logger.error(`errCode: ${bosResponse.error.code}`)
    logger.error(`requestId: ${bosResponse.error.requestId}`)
    logger.error(`errMessage: ${bosResponse.error.message}`)
    logger.error(`statusCode: ${bosResponse.statusCode}`)
}
```

> **注意**：
>
> - 默认情况下，如果Bucket中的分块上传事件的数目大于1000，则只会返回1000个Object，并且返回结果中IsTruncated的值为True，同时返回NextKeyMarker作为下次读取的起点。
> - 若想获取更多分块上传事件，可以访问ListMultipartUploadsResult实例中的keyMarker参数分次读取。

获取所有已上传的分块信息

用户可以使用`listParts`方法获取某个上传事件中所有已上传的块。

示例代码

```
 try {
    let listPartsResult = await bosClient.listParts(bucketName, objectName, "e8583fb592ec699d7ecca085fc46fdd8");
    for (let part of listPartsResult.parts as ListPartType[]) {
        logger.info(`partNumber: ${part.partNumber}, eTag: ${part.eTag}`);
    }
} catch(bosResponse) {
    logger.error(`errCode: ${bosResponse.error.code}`)
    logger.error(`requestId: ${bosResponse.error.requestId}`)
    logger.error(`errMessage: ${bosResponse.error.message}`)
    logger.error(`statusCode: ${bosResponse.statusCode}`)
}
```

> **注意**：
>
> - 默认情况下，如果Bucket中的分块上传事件的数目大于1000，则只会返回1000个Object，并且返回结果中IsTruncated的值为True，同时返回NextPartNumberMarker作为下次读取的起点。
> - 若想获取更多已上传的分块信息，可以访问ListPartsResult中的partNumberMarker参数分次读取。

删除Object

基本流程

1. 创建BosClient类的实例。
2. 执行BosClient.deleteObject()方法。

3. 若操作失败后抛出异常，若成功无返回值。

示例代码

```
 // delete single object
try {
    await bosClient.deleteObject(bucketName, objectName);
    logger.info(`delete object success`);
} catch (bosResponse) {
    logger.error(`errCode: ${bosResponse.error.code}`)
    logger.error(`requestId: ${bosResponse.error.requestId}`)
    logger.error(`errMessage: ${bosResponse.error.message}`)
    logger.error(`statusCode: ${bosResponse.statusCode}`)
}
```

完整示例

```
 import { logger, Credential, BosClient, ClientOptions } from "bos"

let credential = new Credential(AccessKeyID, SecretAccessKey, Token); //STS返回的临时AK/SK及Token
let clientOptions = new ClientOptions();
clientOptions.endpoint = "bj.bcebos.com";  //传入Bucket所在区域域名
let bosClient = new BosClient(credential, clientOptions); // 创建BosClient
let bucketName = "test-harmony-bucket";
let objectName = "test-object";

// delete single object
try {
    await bosClient.deleteObject(bucketName, objectName);
    logger.info(`delete object success`);
} catch (bosResponse) {
    logger.error(`errCode: ${bosResponse.error.code}`)
    logger.error(`requestId: ${bosResponse.error.requestId}`)
    logger.error(`errMessage: ${bosResponse.error.message}`)
    logger.error(`statusCode: ${bosResponse.statusCode}`)
}
```

日志

SDK的log基于`hilog`实现，tag为`BOSSDK`，一般情况下错误信息会通过异常抛出，涉及多个API的方法，运行出现错误，会打印error日志。

详情请见华为开发者指南

异常处理

调用sdk的方法是，异常信息会通过`BosResponse`实例抛出，用户需要读取BosResponse.error信息查看原因，或者访问BosResponse.statusCode查看http状态码。通常的使用方式如下：

```
 try {
  ....
} catch(bosResponse) {
  logger.error(`errCode: ${bosResponse.error.code}`)
  logger.error(`requestId: ${bosResponse.error.requestId}`)
  logger.error(`errMessage: ${bosResponse.error.message}`)
  logger.error(`statusCode: ${bosResponse.statusCode}`)
}
```

> **注意**：非200到300之间的错误码都会抛出异常，如果判断一个object是否存在，如果不存在会返回404并触发异常，这个时候需要判断http状态码是否为404确定是文件不存在还是请求BOS不成功。

版本变更记录

- 鸿蒙SDK开发包[2024-3-31] 版本号 1.0.4
  - 初版鸿蒙SDK发布

周边工具

周边工具概述

百度对象存储 BOS 支持十余款官方工具，在数据上云、数据迁移、数据批量处理等方面为您提供便捷管理方式。同时，BOS 还支持第三方工具，满足您在多种场景下的使用需求。

BOS工具概览

BOS官方工具

BOS 提供 11 款官方工具，详细介绍可参见下表。

工具名称	工具描述	适用场景	使用限制
BOS CMD 命令行工具	不依赖系统环境的命令行工具，支持Bucket和Object级别操作，性能强大且功能完备	适用于本地中小量级数据上云，自动化脚本任务集成的业务场景	-
API 生成工具	提供可视化的 API 调试界面，并能自动生成调用接口所需的代码，帮助开发者更高效地开展研发测试工作。API 生成工具可以在控制台在线生成，前往 API 生成控制台	适用于API调试工作	-
BOS FS 挂载工具	能够将BOS远程服务挂载到本地文件系统中，实现用户对远程数据的便捷调用和高效管理	将BOS远程服务挂载为本地文件系统，多用于办公场景	BOS FS工具不适生产环境对性能要求较高的场景，高并发场景能力较弱
BOS-Util 批量处理工具	批量处理工具，用以解决BOS中海量数据的处理问题，如批量设置Metadata、批量删除文件、批量重命名文件、修改目录、计算给定目录下文件总容量等	批量处理数据场景	推荐使用功能集成更为全面的BOS CMD工具
BOS Probe 错误检测工具	上传和下载的自我检测工具，能够根据用户当前网络和命令执行的相关信息，形成问题报告和处理建议，帮助用户进行错误的快速排查、定位和解决	错误的快速排查、定位	仅作为探测状态使用工具
BOS FTP 工具	基于 Python 实现的 FTP server工具，能够将 FTP 协议的操作转化为对 BOS 中存储对象和目录的操作	FTP 协议的操作转化场景	-
BOS Import 数据迁移工具	数据迁移工具，需要命令行操作，支持将第三方云或者本地自建 S3 数据迁移到 BOS	数据迁移场景，适用于第三方云数据迁移	暂无可视化界面，更推荐使用CloudFlow数据流转平台来进行数据迁移
BOS 桌面 PC 客户端	Windows 和 Mac 操作系统环境下的图形化客户端。您可以通过 BOS 桌面对存储资源进行管理和操作，例如文件或文件夹的上传、下载和删除，并支持本地文件的云端自动同步	在本地操作系统中通过客户端对存储资源进行可视化管理和操作	上传下载速度可能会收PC设备的硬件条件影响
月光宝盒物理设备	月光宝盒是 BOS 提供的 TB - PB 级本地数据迁移上云的硬件设备。支持将本地 IDC 中海量数据通过物理设备拷贝后物流寄送的方式迁移上云	本地IDC数据上云场景	时效性可能会收物流影响
BOS HDFS 工具	支持 HDFS 数据在 BOS 中的海量存储，并在上层数据运算中使用 HDFS 标准接口对数据进行访问和读写，有效解决自建 HDFS 数据的高运维成本和低可扩展性问题	对象存储中数据和自建 HDFS 之间的访问和读写场景	暂不完全支持所有hdfs dfs指令
Alluxio Extension 工具	支持为存储系统（如 BOS、HDFS、S3 等）上层驱动型应用（如 Apache Spark 等）提供统一的客户端 API 和全局命名空间	需要利用内存存储数据，更快地访问BOS	-
BOS CLI 命令行工具	基于BCE Python SDK的命令行工具，支持Bucket的创建和删除，Object的上传、下载、删除和拷贝等功能。	适用于本地中小量级数据上云，自动化脚本任务集成的业务场景	该工具已不再更新和维护，请您安装使用CMD 工具。

第三方工具

BOS 支持四款第三方工具。该工具来源于第三方论坛或者第三方开发者，您可以根据自身需求选择合适的工具进行使用。

工具名称	工具描述
Wordpress插件	Wordpress是主流的博客建站工具之一。博客里的图片、音频、视频内容通常会占用大量空间,该插件可以把博客里的图片自动存储到BOS云端，一方面解决本地磁盘占用，一方面提高图片等多媒体浏览速度。
Discuz插件	Discuz是主流的论坛建站工具之一，人们可以在论坛里发帖并上传附件。附件越来越多就会占用本地磁盘大量空间，所以附件的存储一直是个问题。该插件就是解决该问题的，它可以把论坛附件存储在BOS云端，一方面解决本地磁盘占用，一方面提高附件下载速度。
UCenter插件	UCenter是Comsenz旗下的一个插件。借助UCenter站长可以无缝整合Comsenz系列产品，实现用户的一站式注册、登录、退出以及社区其他数据的交互的php框架。该插件的描述详细介绍了如何改写UCenter的部分代码把用户的头像图片上传到BOS，从而节约本地存储占用并提高头像的显示速度。
Python flask	Flask是一个基于Python的web服务框架，使用Flask可以用几行代码实现复杂的web服务，该工具帮助用户在Flask框架里基于BOS Python SDK实现简单的文件上传、下载、删除操作。

下载专区

BOS CMD下载地址

文件名及下载地址	操作系统	架构	大小	MD5
windows-bcecmd-0.5.6.zip	Windows	x86-64	7.8MB	zip：3b785df5704b653830d39740feef3d4b bcecmd.exe:c367010fcbbae088d6c59289294d06cd
linux-bcecmd-0.5.6.zip	Linux	x86-64	7.6MB	zip:0053f6e2dc1e0c7920887dd3b2138cf4 bcecmd:d7370f81e283afa73908a4e1f823bb47
mac-bcecmd-0.5.6.zip	Mac	x86-64	7.4MB	zip:ac53688676c9447ae8d7d89966e450ed bcecmd:a1d2e0dec12c1fae1c614c1a48cb0c03
arm64-linux-bcecmd-0.5.6.zip	Linux	arm64	7.1MB	zip:cc0488895df97a5c0e6136113b9846936 bcecmd:1c3815ec321ba0727d456ade0f74b004
arm64-mac-bcecmd-0.5.6.zip	Mac	arm64	7.1MB	zip:2667d4b2f80219fec84e3c1640134c6a bcecmd:eb9c22658affa1c79fa7c1667cdfca50

> **说明：** BOSProbe是bcecmd的子命令，如果您已经安装了bcecmd就无需再安装BOSProbe， 如果您还没有安装 bcecmd您需要先安装bcecmd。

BOS CLI下载地址

文件名及下载地址	操作系统	MD5
bce-cli-0.10.10.zip	Linux	640d89ee52ef4bb2c94f53a0cd132c37
BCE CLI工具包	Windows	378b2c18d11f95fb650673b3053593d0
BCE CLI工具包（BCC Centos6.5镜像专用版）	Linux	5ec85c00ae261f81997c19ffb238f391

> **说明：**
>
> - BCE CLI的Windows版本只更新到0.9.1，后续不再更新。Linux版本会持续更新，请按照操作系统获取对应的版本进行安装。
> - BCC Centos6.5镜像专用版CLI封装了Centos6.5环境下缺失的python2.7.5安装依赖包。该版本为定制版本，解压后即可使用，解压后使用./bce运行或者自己设置PATH路径。

BOS Util下载地址

文件名及下载地址	MD5
bosutil-0.1.5.zip	91740e7bfd3e2c31f3ab6ccf92a51736

BOS桌面下载地址

请参考BOSPC客户端安装

BOSFTP下载地址

文件名及下载地址	MD5
bosftp-1.0.1.tar.gz	98a2bec89f974b33d7e85ebce8550360

BosImport下载地址

文件名及下载地址	MD5
bosimport-1.0.1.tar	9d68656bfe3d70bfbc018ec6f8341ee6

BOS-API生成工具

概述

百度对象存储BOS(Baidu Object Storage)，是百度智能云对外提供的稳定、安全、高效以及高扩展存储服务，支持文本、多媒体、二进制等任何类型的数据存储。数据多地域跨集群的存储，以实现资源统一利用，降低使用难度，提高工作效率。用户可以通过本文档提供的简单的RESTful API接口，进行资源管理以及数据上传下载等操作。

API3.0介绍

百度智能云API升级至API3.0，提供可视化的API调试界面，并能自动生成调用接口所需的代码，帮助开发者更高效地开展研发测试工作。 API3.0现已支持对象存储（BOS）、弹性公网IP（EIP）、数据传输服务（DTS）等产品，更多产品接入中，敬请期待。

API3.0目前处于公测阶段，如果您对API3.0的文档、调试界面、代码样例等方面有任何建议，都可以通过文档反馈入口进行反馈。

服务域名

百度智能云开放了多区域支持，BOS支持的区域及对应域名请查看获取访问域名。

规范化字符串(UrlEncode)

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%20%E6%B5%8B%E8%AF%95`。

> **说明：** 下面这段sample代码以java为例演示如何进行UrlEncode。其中input为输入字符串，encodeSlash用来控制是否对"/"进行编码。在有些情况下，做UrlEncode时需要忽略"/"，具体情况可以参见“编码规范”。

```
public static String uri-encode(CharSequence input, boolean encodeSlash) {
  StringBuilder result = new StringBuilder();
  for (int i = 0; i < input.length(); i++) {
  char ch = input.charAt(i);
  if ((ch >= 'A' && ch <= 'Z') || (ch >= 'a' && ch <= 'z') || (ch >= '0' && ch <= '9') || ch == '_' || ch == '-' || ch == '~' || ch == '.') {
     result.append(ch);
     } else if (ch == '/') {
        result.append(encodeSlash ? "%2F" : ch);
     } else {
        result.append(toHexUTF8(ch));
     }
  }
  return result.toString();
}
```

编码规范

- 可解析内容，所有request/response body内容目前均使用UTF-8编码，后续会支持更多encoding类型。
- 在请求时，需要对以下做UrlEncode：
  - Objectname，其中，Resource做UrlEncode的时候需要忽略“/”。
  - Querystring的Value。
  - x-bce-copy-source（忽略“/”）。
  - 自定义Meta:Meta Value只支持可见的ASCII字符，如果需要其它的字符，推荐使用UrlEncode处理。

请求语法风格

BOS API的请求语法支持不带“/v{version}/”的风格，也兼容“/v{version}/”的风格。以GetObject为例，下述两种写法的API调用方式都是正确的。

- 不带/v{version}/的请求语法：GET /object HTTP/1.1
- 带/v{version}/的请求语法：GET /v1/bucket/object HTTP/1.1

资源数目及大小规范

- 用户可有的Bucket数目100。如果需要更多bucket，可以通过工单系统申请。
- Bucket中的Object数不限。
- 支持单个Object最大为5TB。
- 用户请求Header大小不得超过8KB。其中用户自定义Meta（x-bce-meta-）不得超过2KB；Meta值不能为空，且只支持ASCII格式。

命名规范

Bucket格式要求：

- 只能包括小写字母，数字和连字符“-”。
- Bucket名开头和结尾必须是小写字母或数字。
- 长度为3~63位。

Object格式要求：长度不能超过1024个字节的UTF-8字符。

日期与时间

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，凡是HTTP标准中规定的表示日期和时间字段用GMT，其他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：

- 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。
- 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。
- 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。

如何调用API

构造请求

本篇文档介绍了百度智能云REST API请求的构造方法，文档中介绍的信息均可以从API3.0的接口文档中获取。您也可以通过视频指南来进一步了解如何构造一个API请求。

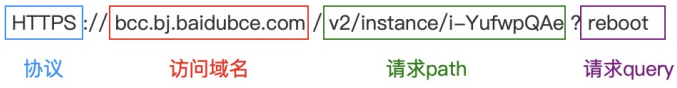
请求URI

请求URI的结构如下：

{协议}://{访问域名}/{请求path}?{请求query}

参数	说明	举例
协议	用于传输请求的协议类型。支持HTTP、HTTPS两种协议，推荐使用HTTPS	HTTPS
访问域名	指提供云产品API服务的域名或IP，不同产品的Endpoint有所不同。请参考BOS访问域名	bj.bcebos.com
请求path	API访问路径，一般用于指向资源所在路径。	v2/instance
请求query	查询参数，跟在?后面。并非每个接口都有请求query，需要根据接口文档判断。	reboot

举例来说，如果您想调用云服务器BCC的重启实例接口，则请求URI为：



> 在具体的接口文档中，由于协议和访问域名相对固定，出于简洁起见将这二者省略，并将接口的请求方法放在请求path之前，如下所示：
>
> 
>

请求消息头

请求消息头中承载一些公共的配置信息，例如消息体的数据类型、发起调用的时间、鉴权信息等。所有接口都可以使用的请求消息头被称为公共请求消息头，列表如下。

名字	类型	描述	是否必选
Authorization	String	用于验证请求合法性的认证信息。更多参见鉴权认证	是
Content-Length	String	RFC2616中定义的HTTP请求内容的长度	否
Content-Type	String	RFC2616中定义的HTTP请求内容的类型，通常情况下为application/json; charset=utf-8	否
Content-MD5	String	RFC2616定义的HTTP请求内容的MD5摘要，可以通过携带该字段来验证保存在BOS侧的文件和用户预期的文件是否一致。	否
Date	String	HTTP 1.1协议中规定的GMT时间，如`Wed, 06 Apr 2016 06:34:40 GMT`	否
Host	String	访问Host值，取值为`BucketName.bj.baidubce.com`	是
x-bce-date	String	当前时间，遵循ISO8601规范，格式如`2016-04-06T08:23:49Z`	否

如果某接口有自己专属的请求消息头，会在接口文档中写明。

请求消息体

请求消息体中的数据结构由请求消息头中的Content-Type定义，在绝大多数情况下要求为json格式。

每个接口的请求消息体内容不同，也并不是每个接口都需要有请求消息体（或者说消息体为空），GET、DELETE操作类型的接口一般来说不需要消息体，消息体具体内容需要您参阅接口文档。

在接口文档中，**请求体字段数据结构说明**是请求消息体的json最外一层的字段信息，如果该接口没有请求消息体，则该部分内容为空。

如果请求消息体json内含有嵌套的数据结构，会在**请求体字段数据结构说明**的表格下面用**某某字段数据结构说明**进行展示。

鉴权认证

签名认证访问

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。您也可以参考如何调用API接口视频进行了解。

临时授权访问

BOS可以通过STS机制实现第三方的临时授权访问，请参考临时授权访问文档。

返回响应

发送API请求后即可收到响应，包含响应状态码、响应消息头、响应消息体三部分。

响应状态码

状态码是一串三位数字代码，代表了API调用的状态。
当状态码为200时，代表调用成功。当调用出现问题时会返回错误码。请见错误码

响应消息头

名字	类型	描述
Content-Length	String	RFC2616中定义的HTTP请求内容长度。
Content-Type	String	RFC2616中定义的HTTP请求内容的类型。
Content-MD5	String	RFC2616定义的HTTP请求内容的MD5摘要，可以通过携带该字段来验证保存在BOS侧的文件和用户预期的文件是否一致。
Connection	String	服务器是否断开连接，取值为close或者keep-alive。
Date	String	HTTP 1.1协议中规定的GMT时间，如Wed, 06 Apr 2016 06:34:40 GMT。
eTag	String	Object的HTTP协议实体标签。eTag (entity tag) 在每个Object生成的时候被创建，用于标识一个Object的内容，eTag值可以用于检查Object内容是否发生变化。
Server	String	服务器的名字，取值为BceBos。
x-bce-request-id	String	由BCE BOS创建，是请求BceBos的唯一标识，
x-bce-debug-id	String	由BCE BOS创建，用于帮助排除故障的标识ID，如果在使用BOS过程中遇到问题，可以在工单中提供该字段便于快速定位问题。

响应消息体

响应消息体通常以json结构化格式返回，传递除响应消息头之外的内容。对于信息获取类请求，会用响应消息体来承载获取到的信息。

错误码

当用户调用API出现错误时，服务端会返回给用户相应的错误码和错误信息，便于用户定位问题，并做出适当的处理。

系统返回错误信息格式如下：

```
{
  "code":"NoSuchKey",
  "message":"The resource you requested does not exist",
  "requestId":" 4db2b34d-654d-4d8a-b49b-3049ca786409"
}
```

其中Code字段定义如下：

错误码（code）	消息（message）	描述	HTTP状态码
AccessDenied	Access denied.	拒绝访问	403 Forbidden
AccountOverdue	Your request is denied because there is an overdue bill of your account.	用户欠费	403 Forbidden
BadDigest	The Content-MD5 you specified did not match what we received.	错误的Content-MD5字段，与实际上传的数据MD5不符	400 Bad Request
BucketAlreadyExists	The requested bucket name is not available. The bucket namespace is shared by all users of the system. Please select a different name and try again.	Bucket已经存在	409 Conflict
BucketNotEmpty	The bucket you tried to delete is not empty.	试图删除一个不为空的bucket	409 Conflict
EntityTooLarge	Your proposed upload exceeds the maximum allowed object size.	上传的数据大于限制	400 Bad Request
EntityTooSmall	Your proposed upload is smaller than the minimum allowed object size.	上传的数据小于限制	400 Bad Request
InappropriateJSON	The JSON you provided was well-formed and valid, but not appropriate for this operation.	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。	400 Bad Request

InappropriateXML	The XML you provided was well-formed and valid, but not appropriate for this operation.	适用场景同InappropriateJSON	400 Bad Request
InternalError	We encountered an internal error. Please try again.	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。	500 Internal Server Error
InvalidArgument	Invalid Argument.	无效参数	400 Bad Request
InvalidBucketName	The specified bucket is not valid.	BucketName不合法	400 Bad Request
InvalidEncryptionAlgorithm	The specified encryption algorithm is invalid.	指定的加密算法无效。	400 Bad Request
InvalidHTTPAuthHeader	The HTTP authorization header is invalid. Consult the service documentation for details.	Authorization头域格式错误	400 Bad Request
InvalidHTTPRequest	There was an error in the body of your HTTP request.	HTTP body格式错误。例如不符合指定的Encoding等	400 Bad Request
InvalidObjectName	Your object key is too long.	Object Key过长	400 Bad Request
InvalidPart	One or more of the specified parts could not be found. The part might not have been uploaded, or the specified entity tag might not have matched the part's entity tag.	无效的Part，在三步上传的第三步，发现有一些part不存在，或者part eTag不匹配	400 Bad Request
InvalidPartOrder	The list of parts was not in ascending order.Parts list must specified in order by part number.	上传的Part必须按照PartNumber升序排列进行上传的第三步	400 Bad Request
InvalidPolicyDocument	The content of the form does not meet the conditions specified in the policy document.	Policy格式错误	400 Bad Request
InvalidRange	The requested range cannot be satisfied.	请求的Range不合法	416 Requested Range Not Satisfiable
InvalidURI	Could not parse the specified URI.	URI形式不正确	400 Bad Request
MalformedJSON	The JSON you provided was not well-formed.	JSON格式不合法	400 Bad Request
MalformedXML	The XML you provided was not well-formed or did not validate against our published schema.	XML格式不合法	400 Bad Request
MaxMessageLengthExceeded	Your request was too big.	超出消息长度的限制	400 Bad Request
MetadataTooLarge	Your metadata headers exceed the maximum allowed metadata size.	Meta数据超过限制	400 Bad Request
MethodNotAllowed	he specified method is not allowed against this resource.	请求的方法不允许	405 Method Not Allowed
MissingContentLength	You must provide the Content-Length HTTP header.	缺少Content-Length字段	411 Length Required
MissingDateHeader	Request must have a "date" or "x-bce-date" header.	请求中找不到date和x-bce-date两者之一；如果同时添加了date和x-bce-date，则x-bce-date的优先级高于date，二者的格式请参考： date: Wed, 06 Apr 2016 06:34:40 GMT x-bce-date: 2016-04-06T08:23:49Z	400 Bad Request
NoReplicationConfiguration	The Replication configuration does not exist.	未配置跨区域复制	404 Not Found
NoSuchBucket	The specified bucket does not exist.	不存在该Bucket	404 Not Found
NoSuchBucketEncryption	The bucket is not encrypted.	该Bucket未加密。	404 Not Found
NoSuchKey	The specified key does not exist.	不存在该Object	404 Not Found
NoSuchUpload	The specified multipart upload does not exist. The upload ID might be invalid, or the multipart upload might have been aborted or completed.	该uploadId所对应的三步上传不存在	404 Not Found
NotImplemented	A header you provided implies functionality that is not implemented.	系统未实现	501 Not Implemented
ObjectUnappendable	The object can not be append	对非Appendable的Object做AppendObject操作	403 Forbidden
OffsetIncorrect	Offset not equal to current object length	追加Appendable Object时，<OffsetSize>值不等于已上传的Object的大小或者<OffsetSize>值不为0但Object不存在	409 Conflict
PreconditionFailed	The specified If-Match header doesn't match the eTag header.	预处理错误	412 Precondition Failed
ReplicationNotEnabled	The Bucket Replication is not enabled.	跨区域同步未开启错误	404 Not Found
ReplicationStatusError	The Bucket Replication Status is not correct. Please make sure both the source and dest bucket have no enabled replication conf and both are not the dest replication bucket of other bucket.	跨区域复制源Bucket或目标Bucket已经在另一条跨区域复制规则中被指定。	409 Conflict
ReplicationStatusNotEmpty	The bucket you tried to delete has an enabled replication conf or is a dest replication bucket of other bucket.	待删除的Bucket开启了跨区域复制功能。	409 Conflict
RequestExpired	Request has expired. Timestamp date is XXX.	请求的时间戳过期。请求超时，XXX要改成x-bce-date的值。如果请求中只有Date，则需要将Date转换为本规范指定的格式。	403 Forbidden
RequestTimeout	Your socket connection to the server was not read from or written to within the timeout period.	请求超时。	408 Request Timeout
ServiceUnavailable	Please reduce your request rate.	服务不可用	503 Service Unavailable
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.	Authorization头域中附带的签名和服务端验证不一致	403 Forbidden
SlowDown	Please reduce your request rate.	请求过于频繁	503 Slow Down
TooManyBuckets	You have attempted to create more buckets than allowed.	创建的Bucket数目超过了限制	400 Bad Request
InvalidStaticWebSiteFormat	The format of index file or 404 file are not allowed.	不允许的index或者404文件名或格式。 比如：文件格式不允许，或者index与404文件同名。	400
NoSuchBucketStaticWebSiteConfig	The static web site configuration does not exist.	Bucket 没有开启静态托管。	404
StaticWebSiteIsDisable	Static web site is disabled or not implemented.	静态网站托管功能被禁止。 如：即BOS不允许此region的bucket 开启静态网站托管功能，或者静态网站托管功能未上线。	501
InvalidObjectState	The object is freezed or restoring.	归档类型文件状态不合法，归档文件冰冻中或取回中。	403
NotArchiveObject	Object is not archive storage class.	当前object不是归档类型文件。	400
RestoreAlreadyInProgress	The object is restoring.Don't restore the object again.	您已经成功调用过RestoreObject接口，BOS服务端正在执行解冻操作，请不要重复执行RestoreObject。	409

		作。请不要重复执行RestoreObject。	
RestoreAlready	The object is restored.	处于解冻状态，无需重重复调用。	409

Service相关接口

列举存储桶

接口说明 本接口列举了请求者拥有的所有bucket

> **注意**
>
> 1. 如果请求中没有用户验证信息（即匿名访问），返回`403 Forbidden`，错误信息：`AccessDenied`。
> 2. JSON请求响应项的命名规则是首字母小写的驼峰格式。
> 3. 一次请求最多返回100个bucket的信息。

请求URI GET /v1

请求体参数

请求体字段数据结构说明

响应体参数

响应体字段数据结构说明 参数名称 | 参数类型 | 描述 | 示例值 ------ | ------- | ---- | ---- **owner** | **Owner** | Bucket owner（拥有者）信息 | **buckets** | **List<Buckets>** | 存放多个bucket信息的容器 |

Owner字段数据结构说明 参数名称 | 参数类型 | 描述 | 示例值 ------ | ------- | ---- | ---- **id** | **String** | Bucket owner的用户id | 10eb6f5ff6ff4605bf044313e8f3ffa5 **displayName** | **String** | Bucket owner的名称 | BosUser

Buckets字段数据结构说明 参数名称 | 参数类型 | 描述 | 示例值 ------ | ------- | ---- | ---- **name** | **String** | Bucket名称 | bucket2 **location** | **String** | Bucket所在区域 | bj **creationDate** | **String** | Bucket创建时间 | 2016-04-05T16:41:58Z **enableMultiAz** | **Boolean** | Bucket数据是否多AZ分布，非多AZ bucket不返回该属性 | true

请求示例

```
 GET /v1
<公共请求头>


{}
```

响应示例

```
 HTTP/1.1 200 OK
<公共响应头>

{
  "owner" : {
    "displayName" : "BosUser",
    "id" : "10eb6f5ff6ff4605bf044313e8f3ffa5"
  },
  "buckets" : [ {
    "name" : "bucket2",
    "location" : "bj",
    "creationDate" : "2016-04-05T16:41:58Z",
    "enableMultiAz" : true
  }, {
    "name" : "bucket2",
    "location" : "bj",
    "creationDate" : "2016-04-05T16:41:58Z",
    "enableMultiAz" : true
  } ]
}
```

错误码 请参考通用错误码

额度管理

获取用户的额度信息

接口说明 此接口用来查看用户的额度设置，注意该接口只能被主用户调用。

请求URI GET /v1/?userQuota

请求体参数

请求体字段数据结构说明

响应体参数

响应体字段数据结构说明 参数名称 | 参数类型 | 描述 | 示例值 ------ | ------- | ---- | ---- **maxBucketCount** | **Long** | 额度设置中的最大可创建Bucket数，若为-1，表示无限制 | 50 **maxCapacityMegaBytes** | **Long** | 额度设置中的最大存储容量限制，若为-1或0，表示无限制 | 12334424

请求示例

```
 GET /v1/?userQuota
<公共请求头>


{}
```

响应示例

```
 HTTP/1.1 200 OK
<公共响应头>

{
  "maxBucketCount" : 50,
  "maxCapacityMegaBytes" : 12334424
}
```

错误码 请参考通用错误码

设置用户的额度

接口说明 此接口用来设置用户的额度，额度包括总的bucket数及各个region总的存储容量，当前总的Bucket数最大值不能超过100，注意该接口只能被主用户调用，

请求URI PUT /v1/?userQuota

请求体参数

请求体字段数据结构说明 参数名称 | 参数类型 | 是否必须 | 描述 | 示例值 ------- | ------ | ------- | ---- | ---- **maxBucketCount** | **Long** | 是|最大可以创建的bucket数，若为-1，则表示不设置 | 50 **maxCapacityMegaBytes** | **Long** | 是|单位为MB，表示最大的存储容量，若为-1或者0，表示不设置存储容量额度限制，即无上限 | 12334424

响应体参数 无

请求示例

```
 PUT /v1/?userQuota
<公共请求头>


{
  "maxBucketCount" : 50,
  "maxCapacityMegaBytes" : 12334424
}
```

响应示例 无

错误码 请参考通用错误码

删除用户的额度设置

接口说明

此接口用来删除额度设置，注意该接口只能被主用户调用。

请求URI DELETE /v1/?userQuota

请求体参数

请求体字段数据结构说明

响应体参数

响应体字段数据结构说明

请求示例

```
 DELETE /v1/?userQuota
<公共请求头>


{}
```

响应示例

```
 HTTP/1.1 200 OK
<公共响应头>

{}
```

错误码 请参考通用错误码

Bucket相关接口

基础操作

创建存储桶

接口说明 本接口用于创建Bucket。每一个用户只允许创建100个Bucket。创建的Bucket其权限默认为private，即Bucket Owner获得FULL_CONTROL，其他人没有任何权限。

> **注意事项**

- 若一个用户创建的Bucket超过100个，服务将返回400 Bad Request，错误码TooManyBuckets。
- 若请求的Bucket已存在，无论该Bucket是否是请求者创建，都会返回409 Conflict，错误信息：BucketAlreadyExists。
- 创建数据多AZ分布的bucket的前提是该region已经支持创建多AZ的bucket，目前BOS仅支持北京和广州两个region，其它region待开通中。

在线调试

https://console.bce.baidu.com/support/api-center.html?product=BOS&project=BOS&parent=基础操作&api=v1/{bucketname}&method=put

请求URI PUT /v1/{bucketname}

参数名称	参数类型	是否必须	描述	示例值	参数位置
bucketname	String	是	存储桶(Bucket)的名称	"bucketname_example"	Path

请求体参数

请求体字段数据结构说明 参数名称 | 参数类型 | 是否必须 | 描述 | 示例值 ------- | ------ | ------- | ---- | ---- **enableMultiAz** | **Boolean** | 否|指定创建的Bucket中数据是否多AZ分布，达到机房级别的数据容灾，true代表创建多AZ bucket，默认为false代表创建。 | true

响应体参数

响应体字段数据结构说明

请求示例

```
 PUT /v1/bucketname_example
<公共请求头>


{
  "enableMultiAz" : true
}
```

响应示例

```
 HTTP/1.1 200 OK
<公共响应头>

{}
```

错误码 请参考通用错误码

列举存储桶下的Object

接口说明 本接口用于获得指定Bucket的Object信息列表。

> **注意事项**
>
> Delimiter可以用来实现文件夹逻辑：
>
> 1. 如果把prefix设为某个文件夹名，就可以罗列以此prefix开头的文件，即该文件夹下递归的所有的文件和子文件夹内的文件。
> 2. 如果再把delimiter设置为 / 时，返回值就只罗列该文件夹下的文件，该文件夹下的子文件夹名返回在CommonPrefixes部分，子文件夹下递归的文件不被显示。 如一个bucket下存在三个object：fun/test.jpg，fun/movie/001.avi，fun/movie/007.avi。 若设定prefix为“fun/”，则返回三个object；如果增加设定delimiter为“/”，则返回文件“fun/test.jpg”和前缀“fun/movie/”。

在线调试

https://console.bce.baidu.com/support/api-center.html?product=BOS&project=BOS&parent=基础操作&api=v1/{bucketName}&method=get

请求URI GET /v1/{bucketName}

参数名称	参数类型	是否必须	描述	示例值	参数位置
bucketName	String	是	BucketName	"bucketName_example"	Path
delimiter	String	否	分隔符; 主要用此项实现list文件夹的逻辑。如果在请求的时候指定了delimiter，BOS把匹配到的Object名称按照一定规则（从preifx到第一个delimiter）截取，截取的字符串去重作为CommonPrefixes的数据返回; delimiter长度限制为1	"delimiter_example"	Query
marker	String	否	object为字母序排列，从marker之后的第一个开始返回	"marker_example"	Query
maxKeys	Integer	否	返回object列表长度最大为1000，默认值是1000，如果指定的值大于1000，按1000操作	56	Query
prefix	String	否	key前缀，限定返回的object key必须以此为前缀	"prefix_example"	Query

请求体参数

请求体字段数据结构说明

响应体参数

响应体字段数据结构说明 参数名称 | 参数类型 | 描述 | 示例值 ------ | ------- | --- | --- **name** | **String** | Bucket名称 | bucket **prefix** | **String** | 查询的前缀 |
delimiter | **String** | 查询的结束符 | / **marker** | **String** | 本次查询的起点 |
maxKeys | **Integer** | 请求返回的最大数目 | 1000 **isTruncated** | **Boolean** | 指明是否查询完整返回了；false-本次已经返回所有结果，true-本次还没有返回所有结果 | false **contents** | **List<Contents>** | 返回的一个object的列表 |

Contents字段数据结构说明 参数名称 | 参数类型 | 描述 | 示例值 ------ | ------- | --- | --- **key** | **String** | Object名称 | my-image1.jpg **lastModified** | **String** | 此Object最后一次被修改的时间 | 2009-10-12T17:51:30Z **eTag** |

String | Object的HTTP协议实体标签 | 0cce7caecc8309864f663d78d1293f98 **size** | **Integer** | Object的内容的大小（字节数） | 124231 **storageClass** | **String** | Object的存储类型，低频存储返回STANDARD_IA，冷存储返回COLD，归档存储返回ARCHIVE，标准存储返回STANDARD。 | COLD **owner** | **Owner** | Object上传者的用户信息 |

Owner字段数据结构说明 参数名称 | 参数类型 | 描述 | 示例值 ------ | ------- | ---- | ---- **id** | **String** | Object上传者的用户id | 168bf6fd8fa74d9789f35a283a1f15e2 **displayName** | **String** | Object上传者的名称 | mtd

请求示例

```
GET /v1/bucketName_example?delimiter=delimiter_example&marker=marker_example&maxKeys=56&prefix=prefix_example
<公共请求头>


{}
```

响应示例

```
HTTP/1.1 200 OK
<公共响应头>

{
  "isTruncated" : false,
  "maxKeys" : 1000,
  "contents" : [ {
    "owner" : {
      "displayName" : "mtd",
      "id" : "168bf6fd8fa74d9789f35a283a1f15e2"
    },
    "storageClass" : "COLD",
    "size" : 124231,
    "eTag" : "0cce7caecc8309864f663d78d1293f98",
    "lastModified" : "2009-10-12T17:51:30Z",
    "key" : "my-image1.jpg"
  }, {
    "owner" : {
      "displayName" : "mtd",
      "id" : "168bf6fd8fa74d9789f35a283a1f15e2"
    },
    "storageClass" : "COLD",
    "size" : 124231,
    "eTag" : "0cce7caecc8309864f663d78d1293f98",
    "lastModified" : "2009-10-12T17:51:30Z",
    "key" : "my-image1.jpg"
  } ],
  "prefix" : "prefix",
  "delimiter" : "/",
  "marker" : "marker",
  "name" : "bucket"
}
```

错误码 请参考通用错误码

查看Bucket是否存在和请求者是否有权限访问这个Bucket

接口说明 本接口用于查看Bucket是否存在和请求者是否有权限访问这个Bucket。当请求返还200 OK时，说明Bucket存在且请求者有权限访问。

在线调试

https://console.bce.baidu.com/support/api-center.html?product=BOS&project=BOS&parent=基础操作&api=v1/{bucketName}&method=head

请求URI HEAD /v1/{bucketName}

参数名称	参数类型	是否必须	描述	示例值	参数位置
bucketName	String	是	bucket名称	"bucketName_example"	Path

请求体参数

请求体字段数据结构说明

响应体参数

响应体字段数据结构说明

请求示例

```
HEAD /v1/bucketName_example
<公共请求头>


{}
```

响应示例

```
HTTP/1.1 200 OK
<公共响应头>

{}
```

错误码 请参考通用错误码

删除存储桶

接口说明 本接口用于删除一个Bucket。在删除前需要保证此Bucket下的所有Object和未完成的三步上传Part已经已被删除，否则会删除失败。

> **注意事项**

1. 删除Bucket之前确认该Bucket没有开通跨区域复制，不是跨区域复制规则中的源Bucket或目标Bucket，否则不能删除。

2. 只有Bucket的拥有者才能删除对应的Bucket，否则返回403 Forbidden，对应错误信息：AccessDenied。

3. 为了确保用户数据安全，防止误删除，BOS不允许用户删除一个非空的Bucket。如果用户试图删除一个存在Object的Bucket，返回409 Conflict错误，错误码：BucketNotEmpty。

在线调试

https://console.bce.baidu.com/support/api-center.html?product=BOS&project=BOS&parent=基础操作&api=v1/{bucketName}&method=delete

请求URI DELETE /v1/{bucketName}

参数名称	参数类型	是否必须	描述	示例值	参数位置
bucketName	String	是	待删除bucket的名称	"bucketName_example"	Path

请求体参数

请求体字段数据结构说明

响应体参数

响应体字段数据结构说明

请求示例

```
 DELETE /v1/bucketName_example
<公共请求头>


{}
```

响应示例

```
 HTTP/1.1 200 OK
<公共响应头>

{}
```

错误码 请参考通用错误码

列举存储桶

接口说明 本接口列举了请求者拥有的所有bucket

> **注意**
>
> 1. 如果请求中没有用户验证信息（即匿名访问），返回403 Forbidden，错误信息：AccessDenied。
>
> 2. JSON请求响应项的命名规则是首字母小写的驼峰格式。
>
> 3. 一次请求最多返回100个bucket的信息。

请求URI GET /v1

请求体参数

请求体字段数据结构说明

响应体参数

响应体字段数据结构说明 参数名称 | 参数类型 | 描述 | 示例值 ------ | ------- | ---- | ---- **owner** | **Owner** | Bucket owner（拥有者）信息 | **buckets** | **List<Buckets>** | 存放多个bucket信息的容器 |

Owner字段数据结构说明 参数名称 | 参数类型 | 描述 | 示例值 ------ | ------- | ---- | ---- **id** | **String** | Bucket owner的用户id | 10eb6f5ff6ff4605bf044313e8f3ffa5 **displayName** | **String** | Bucket owner的名称 | BosUser

Buckets字段数据结构说明 参数名称 | 参数类型 | 描述 | 示例值 ------ | ------- | ---- | ---- **name** | **String** | Bucket名称 | bucket2 **location** | **String** | Bucket所在区域 | bj **creationDate** | **String** | Bucket创建时间 | 2016-04-05T16:41:58Z **enableMultiAz** | **Boolean** | Bucket数据是否多AZ分布，非多AZ bucket不返回该属性 | true

请求示例

```
 GET /v1
<公共请求头>


{}
```

响应示例

```
HTTP/1.1 200 OK
<公共响应头>

{
  "owner" : {
    "displayName" : "BosUser",
    "id" : "10eb6f5ff6ff4605bf044313e8f3ffa5"
  },
  "buckets" : [ {
    "name" : "bucket2",
    "location" : "bj",
    "creationDate" : "2016-04-05T16:41:58Z",
    "enableMultiAz" : true
  }, {
    "name" : "bucket2",
    "location" : "bj",
    "creationDate" : "2016-04-05T16:41:58Z",
    "enableMultiAz" : true
  } ]
}
```

错误码 请参考通用错误码

获取bucket所在的区域

接口说明 本接口用于获取Bucket所在的区域。

请求URI GET /v1/{bucketName}/?location

参数名称	参数类型	是否必须	描述	示例值	参数位置
bucketName	String	是	bucket名称	"bucketName_example"	Path

请求体参数

请求体字段数据结构说明

响应体参数

响应体字段数据结构说明 参数名称 | 参数类型 | 描述 | 示例值 ------ | ------- | ---- | ---- **locationConstraint** | **String** | Bucket所在区域 | bj

请求示例

```
GET /v1/bucketName_example/?location
<公共请求头>


{ }
```

响应示例

```
HTTP/1.1 200 OK
<公共响应头>

{
  "locationConstraint" : "bj"
}
```

错误码 请参考通用错误码

获取Bucket的默认存储类型

接口说明 本接口用来获取Bucket的默认存储类型。

在线调试

https://console.bce.baidu.com/support/api-center.html?product=BOS&project=BOS&parent=基础操作&api=v1/{bucketName}/?storageClass&method=get

请求URI GET /v1/{bucketName}/?storageClass

参数名称	参数类型	是否必须	描述	示例值	参数位置
bucketName	String	是	bucket名称	"bucketName_example"	Path

请求体参数

请求体字段数据结构说明

响应体参数

响应体字段数据结构说明 参数名称 | 参数类型 | 描述 | 示例值 ------ | ------- | ---- | ---- **storageClass** | **String** | 存储类型。标准存储返回STANDARD，低频存储返回STANDARD_IA，冷存储返回COLD，归档存储返回ARCHIVE。 | COLD

请求示例

```
GET /v1/bucketName_example/?storageClass
<公共请求头>


{ }
```

响应示例

```
 HTTP/1.1 200 OK
<公共响应头>

{
  "storageClass" : "COLD"
}
```

错误码 请参考通用错误码

设置Bucket的默认存储类型

接口说明 本接口用来设置Bucket的默认存储类型。

如果用户使用API、CLI或者SDK上传的Object未指定存储类型，则继承Bucket的默认存储类型。如果上传Object指定的存储类型和Bucket默认存储类型不一致时，以Object的存储类型为准。存储类型包含标准存储、低频存储、冷存储和归档存储四种，具体使用场景和性能请参见分级存储。

请求URI PUT /v1/{bucketName}/?storageClass

参数名称	参数类型	是否必须	描述	示例值	参数位置
bucketName	String	是	Bucket名称	"bucketName_example"	Path

请求体参数

请求体字段数据结构说明 参数名称 | 参数类型 | 是否必须 | 描述 | 示例值 ------- | ------ | ------- | ---- | ---- **storageClass** | **String** | 是|指定Bucket的默认存储类型，STANDARD代表标准存储，STANDARD_IA代表低频存储，COLD代表冷存储，ARCHIVE代表归档类型；对于多AZ类型bucket，只允许设置MAZ_STANDARD和MAZ_STANDARD_IA两种存储类型，MAZ_STANDARD代表多AZ标准存储，MAZ_STANDARD_IA代表多AZ低频存储。 | STANDARD_IA

响应体参数

响应体字段数据结构说明

请求示例

```
 PUT /v1/bucketName_example/?storageClass
<公共请求头>


{
  "storageClass" : "STANDARD_IA"
}
```

响应示例

```
 HTTP/1.1 200 OK
<公共响应头>

{}
```

错误码 请参考通用错误码

跨域查询

接口说明 浏览器在发送跨域请求之前会发送一个preflight请求（OPTIONS）并带上特定的来源域，HTTP方法和Header信息等给BOS以决定是否发送真正的请求，本接口即响应这种请求。

请求URI OPTIONS /v1/{bucketName}/{object}

请求体参数 无

响应头参数

参数名称	参数类型	描述	示例值
Access-Control-Allow-Credentials	String	BOS的Server端是否允许客户端在请求中带cookie。需要客户端和Server端同时允许才能生效。BOS在请求通过时会返回允许，即值为true，大小写敏感。	
Access-Control-Allow-Headers	String	允许请求携带的Header的列表，如果请求中有不被允许的Header，则CORS规则匹配失败，返回不包含任何Access-Control-开头的头部。多个Headers也用逗号分隔。注意：返回仅仅包括此次请求的Headers列表，而不管配置文件中是否配置了更多被允许的Headers。	
Access-Control-Allow-Methods	String	允许请求的HTTP方法，如果不允许该请求，则不包含该头部以及所有的Access-Control-*相关的头部。类型：字符串。注意：返回的是所有允许的allowedMethods列表，而不仅仅包括此次请求的method。	
Access-Control-Allow-Origin	String	请求中包含的Origin，如果不允许的话将不包含该头部以及所有的Access-Control-*相关的头部。类型：字符串。	
Access-Control-Expose-Headers	String	允许在请求端的JavaScript程序中访问的Headers的列表。与配置文件中的exposeHeaders对应。类型：字符串。	
Access-Control-Max-Age	String	允许浏览器缓存preflight结果的时间，单位为秒。类型：整形。	

响应体参数

无

请求示例 无

响应示例 无

错误码 请参考通用错误码

权限控制

获取Bucket的访问权限

接口说明 本接口用来获取某个Bucket的访问权限。

在线调试

https://console.bce.baidu.com/support/api-center.html?product=BOS&project=BOS&parent=权限控制&api=v1/{bucketName}/?acl&method=get

请求URI GET /v1/{bucketName}/?acl

参数名称	参数类型	是否必须	描述	示例值	参数位置
bucketName	String	是	bucket名称	"bucketName_example"	Path

请求体参数

请求体字段数据结构说明

响应体参数

响应体字段数据结构说明 参数名称 | 参数类型 | 描述 | 示例值 ------ | ------- | ---- | ---- **owner** | **Owner** | Bucket拥有者信息 | **accessControlList** | **List<AccessControlList>** | 保存acl的容器 |

Owner字段数据结构说明 参数名称 | 参数类型 | 描述 | 示例值 ------ | ------- | ---- | ---- **id** | **String** | Owner用户id | 16df583fe6824d73a5f858f06de0af03

AccessControlList字段数据结构说明 参数名称 | 参数类型 | 描述 | 示例值 ------ | ------- | ---- | ---- **grantee** | **List<Grantee>** | 一个被授权人 | **permission** | **List<String>** | 授权权限支持：FULL_CONTROL，READ，WRITE、LIST、MODIFY、GetObject、PutObject、DeleteObject、RestoreObject等 |

Grantee字段数据结构说明 参数名称 | 参数类型 | 描述 | 示例值 ------ | ------- | ---- | ---- **id** | **String** | 被授权用户id | 10eb6f5ff6ff4605bf044313e8f3ffa5

请求示例

```
 GET /v1/bucketName_example/?acl
<公共请求头>


{}
```

响应示例

```
 HTTP/1.1 200 OK
<公共响应头>

{
  "owner" : {
    "id" : "16df583fe6824d73a5f858f06de0af03"
  },
  "accessControlList" : [ {
    "grantee" : [ {
      "id" : "10eb6f5ff6ff4605bf044313e8f3ffa5"
    }, {
      "id" : "10eb6f5ff6ff4605bf044313e8f3ffa5"
    } ],
    "permission" : [ "FULL_CONTROL", "FULL_CONTROL" ]
  }, {
    "grantee" : [ {
      "id" : "10eb6f5ff6ff4605bf044313e8f3ffa5"
    }, {
      "id" : "10eb6f5ff6ff4605bf044313e8f3ffa5"
    } ],
    "permission" : [ "FULL_CONTROL", "FULL_CONTROL" ]
  } ]
}
```

错误码 请参考通用错误码

设置Bucket的访问权限

接口说明 本接口用于设置Bucket的访问权限。目前BOS支持两种方式设置ACL。第一种是使用CannedAcl，在PutBucketAcl的时候，通过头域的"x-bce-acl"来设置，当前可设置的权限包括：private, public-read, public-read-write（大小写敏感）。第二种方式是上传一个ACL文件，文件格式参见ACL文件格式。

> **注意事项**
>
> 1. BOS系统不支持在同一请求中，同时设置"x-bce-acl"和上传ACL文件。
> 2. 只有Bucket的拥有者和被授予FULL_CONTROL权限的用户才能设置Bucket的ACL权限。
> 3. 在创建Bucket时，Bucket权限会默认设置为private。

请求URI PUT /v1/{bucketName}/?acl

参数名称	参数类型	是否必须	描述	示例值	参数位置
bucketName	String	是	bucket名称	"bucketName_example"	Path

请求头参数 参数名称 | 参数类型 | 是否必须 | 描述 | 示例值 ------- | ------ | ------ | ---- | ---- | xBceAcl | String | 否 | Bucket设置的ACL权限，支持：private、public-read、public-read-write | "xBceAcl_example"

请求体参数

请求体字段数据结构说明 参数名称 | 参数类型 | 是否必须 | 描述 | 示例值 ------- | ------ | ------- | ---- | ---- **accessControlList** | **List<AccessControlList>** | 否|标识acl主体的开始，由一或多组acl配置项组成，其中acl配置项由grantee+permission+resource+condition组合而成。 |

AccessControlList字段数据结构说明 参数名称 | 参数类型 | 是否必须 | 描述 | 示例值 ------- | ------ | ------- | ---- | ---- **effect** | **String** | 否|指定与该条acl配置项匹配的Request能否执行，取值为Allow或Deny。Allow表示可以执行；Deny表示拒绝执行。 | Allow **grantee** | **List<Grantee>** | 是|标识被授权人。 | **permission** | **List<String>** | 是|ACL配置项所影响的权限，可选值为READ、LIST、WRITE、和GetObject。权限详细解释请参见"Bucket ACL支持的permission权限"。 | **resource** | **List<String>** | 否|ACL配置项所影响的资源，表示对resource指定范围的资源设置访问权限，支持通配符。如：/或/xxx。 *resource不填或填Bucket名称，等同于resource字段设为/, /_*]，即对Bucket和所有Object设置访问权限。 | **notResouce** | **String** | 否|ACL配置项所影响的资源，表示对notResource指定范围以外的资源设置访问权限，支持通配符。如：/或/xxx*，表示对BucketName中ObjectKey之外的Object或者以XXX为前缀的Object之外的其他Object设置访问权限。如果notResource字段不填则等同于没有配置notResource，即采用默认配置，对Bucket及所有Object设置访问权限。 | **condition** | **Condition** | 否|ACL配置项所包含的限制条件，支持配置IP地址和Referer名单 |

Grantee字段数据结构说明 参数名称 | 参数类型 | 是否必须 | 描述 | 示例值 ------- | ------ | ------- | ---- | ---- **id** | **String** | 否|标识被授权人的Account ID，用户的Account ID可以登录控制台点击账户名下的"用户信息->基本信息"查看。 | 10eb6f5ff6ff4605bf044313e8f3ffa5

Condition字段数据结构说明 参数名称 | 参数类型 | 是否必须 | 描述 | 示例值 ------- | ------ | ------- | ---- | ---- **currentTime** | **CurrentTime** | 否|condition配置项所包含的时间限制条件，支持配置"dateLessThan"，"dateLessThanEquals"， "dateGreaterThan"和"dateGreaterThanEquals"，四个配置项可以任选若干进行设置，匹配有效的条件是所设置配置项均需匹配。 | **secureTransport** | **Boolean** | 否|标识是否只允许https方式访问。可选值"true"、"false",不设置时视为"false"。当设置为"true"时，表示只允许https方式访问。 | true **ipAddress** | **List<String>** | 否|标识授予访问权限的ip | **referer** | **Referer** | 否|标识授予访问权限的referer |

CurrentTime字段数据结构说明 参数名称 | 参数类型 | 是否必须 | 描述 | 示例值 ------- | ------ | ------- | ---- | ---- **dateLessThan** | **String** | 否| | 2020-07-01T12:00:00Z **dateGreaterThan** | **String** | 否| | 2018-03-01T15:00:00Z

Referer字段数据结构说明 参数名称 | 参数类型 | 是否必须 | 描述 | 示例值 ------- | ------ | ------- | ---- | ---- **stringLike** | **List<String>** | 否| | **stringEquals** | **List<String>** | 否| |

响应体参数 无

请求示例

```
 PUT /v1/bucketName_example/?acl
<公共请求头>
x-bce-acl: xBceAcl_example


{
  "accessControlList" : [ {
    "notResouce" : "notResouce",
    "condition" : {
      "currentTime" : {
        "dateLessThan" : "2020-07-01T12:00:00Z",
        "dateGreaterThan" : "2018-03-01T15:00:00Z"
      },
      "secureTransport" : true,
      "referer" : {
        "stringLike" : [ "http://www.abc.com/*" ],
        "stringEquals" : [ "http://www.abc.com" ]
      },
      "ipAddress" : [ "192.168.1.1" ]
    },
    "resource" : [ "bucket1/*" ],
    "effect" : "Allow",
    "grantee" : [ {
      "id" : "10eb6f5ff6ff4605bf044313e8f3ffa5"
    } ],
    "permission" : [ "FULL_CONTROL" ]
  } ]
}
```

响应示例 无

错误码 请参考通用错误码

生命周期

获取Bucket的生命周期管理规则

接口说明 此接口用于获取定义的生命周期管理规则详细信息。

> **注意事项**
>
> - 如果请求的源Bucket不存在，返回404错误，错误码为NoSuchBucket。
> - 如果请求的源Bucket没有配置lifecycle，返回404错误，错误码为NoLifecycleConfiguration。

请求URI GET /v1/{bucketName}/?lifecycle

参数名称	参数类型	是否必须	描述	示例值	参数位置
bucketName	String	是	bucket的名称	"bucketName_example"	Path

请求体参数

请求体字段数据结构说明

响应体参数

响应体字段数据结构说明 参数名称 | 参数类型 | 描述 | 示例值 ------ | ------- | ---- | ---- **rule** | **List<Rule>** | 所有规则的列表 |

Rule字段数据结构说明 参数名称 | 参数类型 | 描述 | 示例值 ------ | ------- | ---- | ---- **id** | **String** | 规则的标识符 | sample-rule-abort-multiupload-prefix **status** | **String** | 规则的状态。 | enabled **resource** | **List<String>** | 规则对哪些资源生效。 |
condition | **Condition** | 规则依赖的条件。 |
action | **Action** | 对resource执行的操作动作。 |

Condition字段数据结构说明 参数名称 | 参数类型 | 描述 | 示例值 ------ | ------- | ---- | ---- **time** | **Time** | 时间限制条件。 |

Time字段数据结构说明 参数名称 | 参数类型 | 描述 | 示例值 ------ | ------- | ---- | ---- **dateGreaterThan** | **String** | 描述时间关系。 | $(lastModified)+P7D

Action字段数据结构说明 参数名称 | 参数类型 | 描述 | 示例值 ------ | ------- | ---- | ---- **name** | **String** | | AbortMultipartUpload **storageClass** | **String** | | STANDARD_IA

请求示例

```
 GET /v1/bucketName_example/?lifecycle
<公共请求头>


{ }
```

响应示例

```
 HTTP/1.1 200 OK
<公共响应头>

{
  "rule" : [ {
    "condition" : {
      "time" : {
        "dateGreaterThan" : "$(lastModified)+P7D"
      }
    },
    "resource" : [ "bucket/prefix/*", "bucket/prefix/*" ],
    "action" : {
      "storageClass" : "STANDARD_IA",
      "name" : "AbortMultipartUpload"
    },
    "id" : "sample-rule-abort-multiupload-prefix",
    "status" : "enabled"
  }, {
    "condition" : {
      "time" : {
        "dateGreaterThan" : "$(lastModified)+P7D"
      }
    },
    "resource" : [ "bucket/prefix/*", "bucket/prefix/*" ],
    "action" : {
      "storageClass" : "STANDARD_IA",
      "name" : "AbortMultipartUpload"
    },
    "id" : "sample-rule-abort-multiupload-prefix",
    "status" : "enabled"
  } ]
}
```

错误码 请参考通用错误码

设置Bucket的生命周期

接口说明 本接口用来创建生命周期管理规则。

> **说明**
>
> 只有bucket的owner且拥有**FULL_CONTROL**权限才能够进行此请求。

请求URI PUT /v1/{bucketName}/?lifecycle

参数名称	参数类型	是否必须	描述	示例值	参数位置
bucketName	String	是	bucket的名称	"bucketName_example"	Path

请求体参数

请求体字段数据结构说明 参数名称 | 参数类型 | 是否必须 | 描述 | 示例值 ------- | ------ | ------- | ---- | ---- **rule** | **List<Rule>** | 是|规则的列表 |

Rule字段数据结构说明 参数名称 | 参数类型 | 是否必须 | 描述 | 示例值 ------- | ------ | ------- | ---- | ---- **id** | **String** | 是|规则的标识符 | rule-id **status** | **String** | 是|规则的状态。 | enabled **resource** | **List<String>** | 是|规则对哪些资源生效。 | **condition** | **Condition** | 是|规则依赖的条件 | **action** | **Action** | 是|对resource执行的操作动作。 |

Condition字段数据结构说明 参数名称 | 参数类型 | 是否必须 | 描述 | 示例值 ------- | ------ | ------- | ---- | ---- **time** | **Time** | 否|时间限制条件。 |

Time字段数据结构说明 参数名称 | 参数类型 | 是否必须 | 描述 | 示例值 ------- | ------ | ------- | ---- | ---- **dateGreaterThan** | **String** | 否|描述时间关系。 | 2016-09-07T00:00:00Z

Action字段数据结构说明 参数名称 | 参数类型 | 是否必须 | 描述 | 示例值 ------- | ------ | ------- | ---- | ---- **name** | **String** | 是|执行的操作名称。 | DeleteObject **storageClass** | **String** | 否|Object的存储类型 |

响应体参数

响应体字段数据结构说明

请求示例

```
PUT /v1/bucketName_example/?lifecycle
<公共请求头>


{
  "rule" : [ {
    "condition" : {
      "time" : {
        "dateGreaterThan" : "2016-09-07T00:00:00Z"
      }
    },
    "resource" : [ "bucket/prefix/*" ],
    "action" : {
      "storageClass" : "storageClass",
      "name" : "DeleteObject"
    },
    "id" : "rule-id",
    "status" : "enabled"
  } ]
}
```

响应示例

```
HTTP/1.1 200 OK
<公共响应头>

{}
```

错误码 请参考通用错误码

删除Bucket的生命周期管理规则

接口说明 本接口用来删除定义的生命周期管理规则。

请求URI DELETE /v1/{bucketName}/?lifecycle

参数名称	参数类型	是否必须	描述	示例值	参数位置
bucketName	String	是	bucket名称	"bucketName_example"	Path

请求体参数

请求体字段数据结构说明

响应体参数

响应体字段数据结构说明

请求示例

```
DELETE /v1/bucketName_example/?lifecycle
<公共请求头>


{}
```

响应示例

```
HTTP/1.1 200 OK
<公共响应头>

{}
```

错误码 请参考通用错误码

日志管理

获取Bucket的访问日志配置

接口说明 本接口用来获取某个Bucket的访问日志配置情况。

> **注意事项**
>
> 如果请求的源Bucket不存在，返回404错误，错误码为NoSuchBucket。 请求者只有是源Bucket的owner且拥有FULL_CONTROL权限才允许查看，否则返回403错误，错误码为AccessDenied。

请求URI GET /v1/{bucketName}/?logging

参数名称	参数类型	是否必须	描述	示例值	参数位置
bucketName	String	是	bucket名称	"bucketName_example"	Path

请求体参数

请求体字段数据结构说明

响应体参数

响应体字段数据结构说明 参数名称 | 参数类型 | 描述 | 示例值 ------ | ------ | ---- | ---- **status** | **String** | Bucket的访问日志功能的开启状态，取值为enabled和disabled。enabled代表已开启，disabled代表未开启。 |

enabled **targetBucket** | **String** | 指定存放访问日志的Bucket，如果未开启Logging功能，响应中无该字段。 | dscbucket **targetPrefix** | **String** | 指定最终被保存的访问日志文件前缀，如果未开启Logging功能，响应中无该字段。 | mylog-

请求示例

```
 GET /v1/bucketName_example/?logging
<公共请求头>


{}
```

响应示例

```
 HTTP/1.1 200 OK
<公共响应头>

{
  "targetPrefix" : "mylog-",
  "targetBucket" : "dscbucket",
  "status" : "enabled"
}
```

错误码 请参考通用错误码

开启Bucket的访问日志

接口说明 本接口用来开启Bucket的访问日志并指定存放日志的Bucket和访本接口用来开启Bucket的访问日志并指定存放日志的Bucket和访问日志的文件前缀。访问日志的规则请参见日志命名规则和日志格式。

注意事项

- 用户必须是源Bucket的owner且拥有FULL_CONTROL权限，且是目标Bucket的owner。
- 源Bucket和目标Bucket必须同时存在。
- 源Bucket和目标Bucket必须属于同一个Region。
- 如果HTTP Body中Json不合法，BOS会返回CODE_MALFORMED_JSON错误。
- 如果HTTP Body中Json有无效字段，BOS会返回CODE_INAPPROPRIATE_JSON错误。
- 用户可将不同的源Bucket的Logging都保存在同一个目标Bucket内，建议指定不同的 targetPrefix便于区分。
- 如果源Bucket开通了Logging功能，源Bucket被删除的同时，相应的Logging信息也将被删除。
- 如果Logging的目标Bucket被删除，则源Bucket的Logging功能会被自动关闭。
- 如果需要修改目标Bucket等信息，可再发送一个PutBucketLogging请求，请求中包含需要修改的信息。
- targetPrefix表示存储访问日志记录的Object名字前缀，可以为空。如果不为空时，targetPrefix可以包含字母、数字、中划线、下划线、斜杠，且必须以字母开头，长度不大于32位。
- 重复请求返回结果相同。

请求URI PUT /v1/{bucketName}/?logging

参数名称	参数类型	是否必须	描述	示例值	参数位置
bucketName	String	是	bucket的名称xxx	"bucketName_example"	Path

请求体参数

请求体字段数据结构说明 参数名称 | 参数类型 | 是否必须 | 描述 | 示例值 ------ | ------ | ------- | ---- | ---- **targetBucket** | **String** | 是|指定存放访问日志的Bucket | TargetBucketName **targetPrefix** | **String** | 否|指定最终被保存的访问日志文件前缀 | TargetPrefixName

响应体参数 无

请求示例

```
 PUT /v1/bucketName_example/?logging
<公共请求头>


{
  "targetPrefix" : "TargetPrefixName",
  "targetBucket" : "TargetBucketName"
}
```

响应示例 无

错误码 请参考通用错误码

关闭Bucket的访问日志配置

接口说明 本接口用来关闭Bucket访问日志记录功能。

注意事项

- 如果请求的源Bucket不存在，返回404错误，错误码为NoSuchBucket。
- 请求者只有是源Bucket的owner且拥有FULL_CONTROL权限，才能关闭Bucket访问日志记录功能。否则，BOS返回403错误，错误码为AccessDenied。
- 如果请求的源Bucket没有开通Logging功能，依然返回HTTP状态码204。

请求URI DELETE /v1/{bucketName}/?logging

参数名称	参数类型	是否必须	描述	示例值	参数位置
bucketName	String	是	bucket的名称	"bucketName_example"	Path

请求体参数

请求体字段数据结构说明

响应体参数

响应体字段数据结构说明

请求示例

```
DELETE /v1/bucketName_example/?logging
<公共请求头>


{}
```

响应示例

```
HTTP/1.1 200 OK
<公共响应头>

{}
```

错误码 请参考通用错误码

静态网站

获取bucket的静态网站托管信息

接口说明 本接口用于获取bucket的静态网站托管信息。

请求URI GET /v1/{bucketName}/?website

参数名称	参数类型	是否必须	描述	示例值	参数位置
bucketName	String	是	bucket的名称	"bucketName_example"	Path

请求体参数

请求体字段数据结构说明

响应体参数

响应体字段数据结构说明 参数名称 | 参数类型 | 描述 | 示例值 ------ | ------- | ---- | ---- **index** | **String** | Index文件名称 | index.html **notFound** | **String** | 404文件名称 | 404.html **notFoundHttpStatus** | **String** | 返回错误页面时的HTTP状态码 | 404

请求示例

```
GET /v1/bucketName_example/?website
<公共请求头>


{}
```

响应示例

```
HTTP/1.1 200 OK
<公共响应头>

{
  "index" : "index.html",
  "notFound" : "404.html"
}
```

错误码 请参考通用错误码

设置静态网站托管

接口说明 本接口用于设置静态网站托管。

用户必须对bucket具有full control 权限。不建议设置归档文件，归档文件没有取回时不可读，StaticWebsite此时不会生效。

请求URI PUT /v1/{bucketName}/?website

参数名称	参数类型	是否必须	描述	示例值	参数位置
bucketName	String	是	bucket的名称	"bucketName_example"	Path

请求体参数

请求体字段数据结构说明 参数名称 | 参数类型 | 是否必须 | 描述 | 示例值 ------ | ------ | ------- | ---- | ---- **index** | **String** | 否|Index文件名称 | index.html **notFound** | **String** | 否|404文件名称 | 404.html **notFoundHttpStatus** | **String** | 否|返回错误页面时的HTTP状态码。取值：200、404（默认） | 404

index 与 notFound 必须至少设置一个

响应体参数

响应体字段数据结构说明

请求示例

```
 PUT /v1/bucketName_example/?website
<公共请求头>


PUT /?website HTTP/1.1
Host: <BcuketName>.bj.bcebos.com
Date: Tue, 17 May 2016 08:36:52 GMT
Authorization: AuthorizationString
Content-Length: 65

{
    "index": "index.html",
    "notFound": "404.html"
}
```

响应示例

```
 HTTP/1.1 200 OK
<公共响应头>

HTTP/1.1 200 OK
x-bce-request-id: 1a5fd81e-626b-45b3-a885-15fff9cd106c
Date: Tue, 17 May 2016 08:36:52 GMT
Content-Length: 0
Server: BceBos
```

错误码 请参考通用错误码

删除bucket的静态网站托管信息

接口说明 本接口用于删除bucket设置的静态网站托管信息，并关闭此bucket的静态网站托管。

用户必须对bucket具有 FULL_CONTROL 权限。

请求URI DELETE /v1/{bucketName}/?website

参数名称	参数类型	是否必须	描述	示例值	参数位置
bucketName	String	是	bucket的名称	"bucketName_example"	Path

请求体参数

请求体字段数据结构说明

响应体参数

响应体字段数据结构说明

请求示例

```
 DELETE /v1/bucketName_example/?website
<公共请求头>


{}
```

响应示例

```
 HTTP/1.1 200 OK
<公共响应头>

{}
```

错误码 请参考通用错误码

数据加密

判断bucket的加密是否打开

接口说明 本接口用于判断Bucket服务端加密是否打开。

请求URI GET /v1/{bucketName}/?encryption

参数名称	参数类型	是否必须	描述	示例值	参数位置
bucketName	String	是	bucket的名称	"bucketName_example"	Path

请求体参数

请求体字段数据结构说明

响应体参数

响应体字段数据结构说明 参数名称 | 参数类型 | 描述 | 示例值 ------ | ------- | --- | --- **encryptionAlgorithm** | **String** | Bucket的服务器端加密类型，当前支持AES256和SM4加密算法。 | AES256

请求示例

```
 GET /v1/bucketName_example/?encryption
<公共请求头>


{}
```

响应示例

```
 HTTP/1.1 200 OK
<公共响应头>

{
  "encryptionAlgorithm" : "AES256"
}
```

错误码 请参考通用错误码

开启bucket的加密开关

接口说明 本接口用于开启指定Bucket的加密开关。

请求URI PUT /v1/{bucketName}/?encryption

参数名称	参数类型	是否必须	描述	示例值	参数位置
bucketName	String	是	bucket的名称	"bucketName_example"	Path

请求体参数

请求体字段数据结构说明 参数名称 | 参数类型 | 是否必须 | 描述 | 示例值 ------- | ------ | ------- | --- | --- **encryptionAlgorithm** | **String** | 是|指定Bucket的服务器端加密类型，当前支持AES256和SM4加密算法。 | AES256

响应体参数

响应体字段数据结构说明

请求示例

```
 PUT /v1/bucketName_example/?encryption
<公共请求头>


{
  "encryptionAlgorithm" : "AES256"
}
```

响应示例

```
 HTTP/1.1 200 OK
<公共响应头>

{}
```

错误码 请参考通用错误码

关闭bucket的服务端加密功能

接口说明 本接口用于关闭服务端加密功能。

请求URI DELETE /v1/{bucketName}/?encryption

参数名称	参数类型	是否必须	描述	示例值	参数位置
bucketName	String	是	bucket的名称	"bucketName_example"	Path

请求体参数

请求体字段数据结构说明

响应体参数

响应体字段数据结构说明

请求示例

```
 DELETE /v1/bucketName_example/?encryption
<公共请求头>


{}
```

响应示例

```
 HTTP/1.1 200 OK
<公共响应头>

{ }
```

错误码 请参考通用错误码

跨域访问

获取指定Bucket上设定一个跨域资源共享规则

接口说明 本接口用于获取指定的Bucket当前的CORS规则。

权限说明： 只有Bucket的所有者和被授予FULL_CONTROL权限的用户才能设置Bucket的CORS。没有权限时，返回403 Forbidden错误，错误码： AccessDenied。

请求URI GET /v1/{bucketName}/?cors

参数名称	参数类型	是否必须	描述	示例值	参数位置
bucketName	String	是	bucket的名称	"bucketName_example"	Path

请求体参数

请求体字段数据结构说明 参数名称 | 参数类型 | 是否必须 | 描述 | 示例值 ------- | ------ | ------- | ---- | ---- **corsConfiguration** | **List<CorsConfiguration>** | 是|Bucket的CORS规则容器。每个Bucket 最多允许有100条规则。 如果有多条配置，执行顺序为从上到下。 |

CorsConfiguration字段数据结构说明 参数名称 | 参数类型 | 是否必须 | 描述 | 示例值 ------- | ------ | ------- | ---- | ---- **allowedOrigins** | **List<String>** | 是|存储允许的跨域请求的来源的容器。 | **allowedMethods** | **List<String>** | 是|存储允许的跨域请求方法的容器。 | **allowedHeaders** | **List<String>** | 是|存储允许的allowedHeaders容器。控制 在OPTIONS预取指令中Access-Control- Request-Headers头中指定的header是 否允许。 | **allowedExposeHeaders** | **List<String>** | 否|存储允许用户从应用程序中访问的响应头的容器。 | **maxAgeSeconds** | **Integer** | 否|指定浏览器对特定资源的预取（OPTIONS）请求返 回结果的缓存时间，在缓存时间内请求方不必发送重 复的preflight 请求，单位为秒。 类型：Int64。 | 1800

响应体参数

响应体字段数据结构说明 参数名称 | 参数类型 | 描述 | 示例值 ------ | ------- | ---- | ---- **corsConfiguration** | **List<CorsConfiguration>** | Bucket的CORS规则容器。每个Bucket最多允许有100条规则。如果有多条配置，执行顺序为从上到下。 |

CorsConfiguration字段数据结构说明 参数名称 | 参数类型 | 描述 | 示例值 ------ | ------- | ---- | ---- **allowedOrigins** | **List<String>** | 存储允许的跨域请求的来源的容器。 |
allowedMethods | **List<String>** | 存储允许的跨域请求方法的容器。 |
allowedHeaders | **List<String>** | 存储允许的allowedHeaders容器。控制在OPTIONS预取指令中Access-Control-Request-Headers头中指定的header是否允许。 |
allowedExposeHeaders | **List<String>** | 存储允许用户从应用程序中访问的响应头的容器。 |
maxAgeSeconds | **Long** | 指定浏览器对特定资源的预取（OPTIONS）请求返回结果的缓存时间，在缓存时间内请求方不必发送重复的preflight请求，单位为秒。类型：Int64。 | 1800

请求示例

```
 GET /v1/bucketName_example/?cors
<公共请求头>


{
  "corsConfiguration" : [ {
    "allowedExposeHeaders" : [ "user-custom-expose-header" ],
    "allowedOrigins" : [ "http://www.example.com" ],
    "allowedHeaders" : [ "Authorization" ],
    "allowedMethods" : [ "GET" ],
    "maxAgeSeconds" : 1800
  } ]
}
```

响应示例

```
 HTTP/1.1 200 OK
<公共响应头>

{
  "corsConfiguration" : [ {
    "allowedExposeHeaders" : [ "user-custom-expose-header", "user-custom-expose-header" ],
    "allowedOrigins" : [ "http://www.example.com", "http://www.example.com" ],
    "allowedHeaders" : [ "Authorization", "Authorization" ],
    "allowedMethods" : [ "GET", "GET" ],
    "maxAgeSeconds" : 1800
  }, {
    "allowedExposeHeaders" : [ "user-custom-expose-header", "user-custom-expose-header" ],
    "allowedOrigins" : [ "http://www.example.com", "http://www.example.com" ],
    "allowedHeaders" : [ "Authorization", "Authorization" ],
    "allowedMethods" : [ "GET", "GET" ],
    "maxAgeSeconds" : 1800
  } ]
}
```

错误码 请参考通用错误码

在指定Bucket上设定一个跨域资源共享规则

接口说明 本接口用来在指定的Bucket上设定一个跨域资源共享（CORS）的规则，如果原规则存在则覆盖原规则。

权限说明 只有Bucket的所有者和被授予FULL_CONTROL权限的用户才能设置Bucket的CORS。没有权限时，返回403 Forbidden错误，错误码：AccessDenied。

注意事项

- 通过此接口设置CORS规则之前，Bucket的CORS权限设置为不允许跨域。
- 每个Bucket最多只允许有一个规则文件，因此新上传的规则文件会覆盖原有的。

- CORS规则文件大小限制为20KB，因此请求时大于20KB会返回超出大小错误（400 Bad Request: EntityTooLarge）。

请求URI PUT /v1/{bucketName}/?cors

参数名称	参数类型	是否必须	描述	示例值	参数位置
bucketName	String	是	bucket的名称	"bucketName_example"	Path

请求体参数

请求体字段数据结构说明 参数名称 | 参数类型 | 是否必须 | 描述 | 示例值 ------- | ------ | ------- | ---- | ---- **corsConfiguration** | **List<CorsConfiguration>** | 是|Bucket的CORS规则容器。每个Bucket 最多允许有100条规则。 如果有多条配置，执行顺序为从上到下。 |

CorsConfiguration字段数据结构说明 参数名称 | 参数类型 | 是否必须 | 描述 | 示例值 ------- | ------ | ------- | ---- | ---- **allowedOrigins** | **List<String>** | 是|存储允许的跨域请求的来源的容器。 | **allowedMethods** | **List<String>** | 是|存储允许的跨域请求方法的容器。 | **allowedHeaders** | **List<String>** | 是|存储允许的allowedHeaders容器。控制 在OPTIONS预取指令中Access-Control- Request-Headers头中指定的header是 否允许。 | **allowedExposeHeaders** | **List<String>** | 否|存储允许用户从应用程序中访问的响应头的容器。 | **maxAgeSeconds** | **Integer** | 否|指定浏览器对特定资源的预取（OPTIONS）请求返 回结果的缓存时间，在缓存时间内请求方不必发送重 复的preflight 请求，单位为秒。 类型：Int64。 | 1800

响应体参数

响应体字段数据结构说明

请求示例

```
 PUT /v1/bucketName_example/?cors
<公共请求头>


{
  "corsConfiguration" : [ {
    "allowedExposeHeaders" : [ "user-custom-expose-header" ],
    "allowedOrigins" : [ "http://www.example.com" ],
    "allowedHeaders" : [ "Authorization" ],
    "allowedMethods" : [ "GET" ],
    "maxAgeSeconds" : 1800
  } ]
}
```

响应示例

```
 HTTP/1.1 200 OK
<公共响应头>

{ }
```

错误码 请参考通用错误码

关闭指定Bucket上设定一个跨域资源共享规则

接口说明 本接口用于关闭指定Bucket的CORS功能并清空所有规则。

权限说明

只有Bucket的所有者和被授予FULL_CONTROL权限的用户才有权限删除CORS规则。

注意事项

- 当所对应的Bucket不存在时，会返回错误404 Not Found错误，错误码：NoSuchBucket。
- 当没有权限Delete Cors时，会返回错误403 Forbidden，错误码：AccessDenied。
- 当所对应的Bucket原本并不存在CORS规则时，会返回204 No Content。

请求URI DELETE /v1/{bucketName}/?cors

请求体参数 无

响应体参数

无

请求示例 无

响应示例

```
 HTTP/1.1 200 OK
<公共响应头>

{ }
```

错误码 请参考通用错误码

回收站

获取Bucket回收站的开通状态

接口说明 本接口用于获取Bucket Trash开通状态，返回当前trash目录名，默认为.trash。用户必须是源Bucket的owner且拥有FULL_CONTROL权限，且是目标Bucket的owner。

请求URI GET /v1/{bucketName}/?trash

请求体参数 无

响应体参数

无

请求示例 无

响应示例

```
 HTTP/1.1 200 OK
<公共响应头>

{
  "trashDir" : ".trash/"
}
```

错误码 请参考通用错误码

开通Bucket回收站功能

接口说明

本接口用于开通Bucket Trash功能，用户必须是源Bucket的owner且拥有FULL_CONTROL权限，且是目标Bucket的owner。

请求URI PUT /v1/{bucketName}/?trash

参数名称	参数类型	是否必须	描述	示例值	参数位置
bucketName	String	是	bucket的名称	"bucketName_example"	Path

请求体参数

请求体字段数据结构说明 参数名称 | 参数类型 | 是否必须 | 描述 | 示例值 ------- | ------ | ------- | ---- | ---- **trashDir** | **String** | 否|回收站的目录名称，有长度限制。可选，不填默认名为.trash | trashDirName

响应体参数 无

请求示例

```
 PUT /v1/bucketName_example/?trash
<公共请求头>


{
  "trashDir" : "trashDirName"
}
```

响应示例 无

错误码

错误码	错误描述	HTTP状态码	中文解释
MalformedJSON	Json格式错误	400	Json格式错误
EntityTooLarge	trash目录名过长	400	不超过1024字节
InvalidTrashDirectoryName	trash目录包含非法字符	400	只能包含字母，数字，中文，下划线（_）和短横线（-），小数点（.）
NoSuchBucket	bucket不存在	404	bucket不存在
AccessDenied	没有权限	403	只有bucket owner且具有FULL_CONTROL可以操作

关闭Bucket回收站功能

接口说明 本接口用于关闭trash功能，用户必须是源Bucket的owner且拥有FULL_CONTROL权限，且是目标Bucket的owner。

请求URI DELETE /v1/{bucketName}/?trash

参数名称	参数类型	是否必须	描述	示例值	参数位置
bucketName	String	是	bucket的名称	"bucketName_example"	Path

请求体参数

请求体字段数据结构说明

响应体参数

响应体字段数据结构说明

请求示例

```
 DELETE /v1/bucketName_example/?trash
<公共请求头>


{}
```

响应示例

```
 HTTP/1.1 200 OK
<公共响应头>

{}
```

错误码 请参考通用错误码

合规保留

获取Bucket合规保留策略

接口说明 本接口获取合规保留策略配置信息。

请求URI GET /v1/{bucketName}/?objectlock

参数名称	参数类型	是否必须	描述	示例值	参数位置
bucketName	String	是	bucket的名称	"bucketName_example"	Path

请求体参数

请求体字段数据结构说明

响应体参数

响应体字段数据结构说明 参数名称 | 参数类型 | 描述 | 示例值 ------ | ------- | ---- | ---- **lockStatus** | **String** | 合规保留策略状态 | IN_PROGRESS **createDate** | **Integer** | 合规保留策略创建时间 | 1569317168 **expirationDate** | **Integer** | 合规保留策略过期时间，只有IN_PROGRESS状态下才可能会过期 | 1569403568 **retentionDays** | **Integer** | 合规保留保护周期，以天为单位 | 3

请求示例

```
 GET /v1/bucketName_example/?objectlock
<公共请求头>


{}
```

响应示例

```
 HTTP/1.1 200 OK
<公共响应头>

{
  "lockStatus" : "IN_PROGRESS",
  "retentionDays" : 3,
  "createDate" : 1569317168,
  "expirationDate" : 1569403568
}
```

错误码 请参考通用错误码

初始化合规保留策略

接口说明 本接口为指定Bucket创建基于时间的合规保留策略，此时策略状态变成IN_PROGRESS状态。

注意：

- 该状态的有效期为24小时，在有效期24小时内，此策略对应的Bucket资源处于保护状态，其保护规则如下所示： 假设您在2013年6月1日创建一个名为examplebucket的Bucket，并且在不同时间上传了file1.txt、file2.txt、file3.txt三个Object。随后在2014年7月1日创建了保护周期为5年的bucket锁策略。有关这三个Object的具体上传时间以及对应的Object到期时间如下：

Object名称	上传时间	Object到期时间
file1.txt	2013年6月1日	2018年5月31日
file2.txt	2014年7月1日	2019年6月30日
file3.txt	2018年9月30日	2023年9月29日

- Bucket内的数据处于被保护状态时，任何人包括您尝试删除或修改这些数据，BOS API将返回409 FileImmutable的错误信息，请您谨慎配置该合规策略。

请求URI POST /v1/{bucketName}/?objectlock

参数名称	参数类型	是否必须	描述	示例值	参数位置
bucketName	String	是	bucket的名称	"bucketName_example"	Path

请求体参数

请求体字段数据结构说明 参数名称 | 参数类型 | 是否必须 | 描述 | 示例值 ------- | ------ | ------- | ---- | ---- **retentionDays** | **Integer** | 是|保护周期，单位为天 | 30

响应体参数 无

请求示例

```
 POST /v1/bucketName_example/?objectlock
<公共请求头>


{
  "retentionDays" : 30
}
```

响应示例 无

错误码 请参考通用错误码

删除合规保留策略

接口说明 当合规保留策略处于IN_PROGRESS和EXPIRED状态时，您可以通过该接口进行删除合规保留策略，若合规保留策略处于LOCKED锁定状态时，您将不能进行删除操作，除非删除该Bucket。

请求URI DELETE /v1/{bucketName}/?objectlock

参数名称	参数类型	是否必须	描述	示例值	参数位置
bucketName	String	是	bucket的名称	"bucketName_example"	Path

请求体参数

请求体字段数据结构说明

响应体参数

响应体字段数据结构说明

请求示例

```
 DELETE /v1/bucketName_example/?objectlock
<公共请求头>


{}
```

响应示例

```
 HTTP/1.1 200 OK
<公共响应头>

{}
```

错误码 请参考通用错误码

锁定Bucket合规保留策略

接口说明 将合规保留策略立即锁定，变成LOCKED锁定状态，当合规保留策略处于LOCKED锁定时，任何人不可删除该策略，除非删除该Bucket，请您谨慎配置。

请求URI POST /v1/{bucketName}/?completeobjectlock

参数名称	参数类型	是否必须	描述	示例值	参数位置
bucketName	String	是	bucket的名称	"bucketName_example"	Path

请求体参数

请求体字段数据结构说明

响应体参数

响应体字段数据结构说明

请求示例

```
 POST /v1/bucketName_example/?completeobjectlock
<公共请求头>


{}
```

响应示例

```
 HTTP/1.1 200 OK
<公共响应头>

{}
```

错误码 请参考通用错误码

延迟Bucket的合规保留策略保护周期

接口说明 本接口用于延长合规保留策略保护周期

请求URI POST /v1/{bucketName}/?extendobjectlock

参数名称	参数类型	是否必须	描述	示例值	参数位置
bucketName	String	是	bucket的名称	"bucketName_example"	Path

请求体参数

请求体字段数据结构说明 参数名称 | 参数类型 | 是否必须 | 描述 | 示例值 ------- | ------ | ------- | ---- | ---- **extendRetentionDays** | **Integer** | 是|新的保护周期，单位为天，不可缩短保护周期 | 2

响应体参数

响应体字段数据结构说明

请求示例

```
 POST /v1/bucketName_example/?extendobjectlock
<公共请求头>


{
  "extendRetentionDays" : 2
}
```

响应示例

```
 HTTP/1.1 200 OK
<公共响应头>

{}
```

错误码 请参考通用错误码

原图保护

获取Bucket的原图保护配置情况

接口说明 本接口用来获取某个Bucket的原图保护配置情况。

说明： 请求者拥有FULL_CONTROL权限才允许查看，否则返回403错误，错误码为AccessDenied。

请求URI GET /v1/{bucketName}/?copyrightProtection

参数名称	参数类型	是否必须	描述	示例值	参数位置
bucketName	String	是	bucket的名称	"bucketName_example"	Path

请求体参数

请求体字段数据结构说明

响应体参数

响应体字段数据结构说明

请求示例

```
 GET /v1/bucketName_example/?copyrightProtection
<公共请求头>


{}
```

响应示例

```
 HTTP/1.1 200 OK
<公共响应头>

{}
```

错误码 请参考通用错误码

开启Bucket的原图保护功能

接口说明 本接口用来开启Bucket的原图保护功能，并指定resource字段，表示生效的资源范围。 对于开通原图保护的文件，不允许匿名下载访问该文件，或带自定义图片处理参数访问，只允许使用style样式访问或携带合法签名访问。

说明： 用户必须拥有FULL_CONTROL权限。

请求URI PUT /v1/{bucketName}/?copyrightProtection

参数名称	参数类型	是否必须	描述	示例值	参数位置
bucketName	String	是	bucket的名称	"bucketName_example"	Path

请求体参数

请求体字段数据结构说明 参数名称 | 参数类型 | 是否必须 | 描述 | 示例值 ------- | ------ | ------- | ---- | ---- **resource** | **List<String>** | 否|表示生效的资源范围 |

响应体参数

响应体字段数据结构说明

请求示例

```
PUT /v1/bucketName_example/?copyrightProtection
<公共请求头>


{
  "resource" : [ "bucket/prefix/*" ]
}
```

响应示例

```
HTTP/1.1 200 OK
<公共响应头>

{}
```

错误码 请参考通用错误码

关闭Bucket的原图保护配置情况

接口说明 本接口用来关闭Bucket原图保护功能。

说明：

- 请求者只有拥有FULL_CONTROL权限才允许关闭原图保护功能，否则返回403错误，错误码为AccessDenied。
- 如果请求的Bucket没有开通原图保护功能，依然返回HTTP状态码204。

请求URI DELETE /v1/{bucketName}/?copyrightProtection

参数名称	参数类型	是否必须	描述	示例值	参数位置
bucketName	String	是	bucket的名称	"bucketName_example"	Path

请求体参数

请求体字段数据结构说明

响应体参数

响应体字段数据结构说明

请求示例

```
DELETE /v1/bucketName_example/?copyrightProtection
<公共请求头>


{}
```

响应示例

```
HTTP/1.1 200 OK
<公共响应头>

{}
```

错误码 请参考通用错误码

事件通知

获取Bucket的通知规则

接口说明 本接口用于获取指定bucket上的通知规则。

请求URI GET /v1/{bucketName}/?notification

参数名称	参数类型	是否必须	描述	示例值	参数位置
bucketName	String	是	bucket的名称	"bucketName_example"	Path

请求体参数

请求体字段数据结构说明

响应体参数

响应体字段数据结构说明 参数名称 | 参数类型 | 描述 | 示例值 ------ | ------- | ---- | ---- **notifications** | **List<Notifications>** | |

Notifications字段数据结构说明 参数名称 | 参数类型 | 描述 | 示例值 ------ | ------- | ---- | ---- **id** | **String** | 规则id | notify-id-1 **name** | **String** | 规则名称 | rule-name **appId** | **String** | 注册本条规则的产品id | app-id-1 **status** | **String** | 可选值：{"disabled", "enabled"} | enabled **resources** | **List<String>** | 订阅的资源 |
events | **List<String>** | 订阅的事件 |
apps | **List<Apps>** | 订阅消息的产品 |

Apps字段数据结构说明 参数名称 | 参数类型 | 描述 | 示例值 ------ | ------- | ---- | ---- **id** | **String** | | app-id-3 **eventUrl** | **String** | | app:ImageOcr **xVars** | **String** | | {"saveUrl": "http://xxx.com/ocr\"}

请求示例

```
 GET /v1/bucketName_example/?notification
<公共请求头>


{ }
```

响应示例

```
 HTTP/1.1 200 OK
<公共响应头>

{
  "notifications" : [ {
    "appId" : "app-id-1",
    "name" : "rule-name",
    "resources" : [ "bucket-a/path1", "bucket-a/path1" ],
    "id" : "notify-id-1",
    "events" : [ "PutObject", "PutObject" ],
    "status" : "enabled",
    "apps" : [ {
      "eventUrl" : "app:ImageOcr",
      "xVars" : "{\"saveUrl\": \"http://xxx.com/ocr\"}",
      "id" : "app-id-3"
    }, {
      "eventUrl" : "app:ImageOcr",
      "xVars" : "{\"saveUrl\": \"http://xxx.com/ocr\"}",
      "id" : "app-id-3"
    } ]
  }, {
    "appId" : "app-id-1",
    "name" : "rule-name",
    "resources" : [ "bucket-a/path1", "bucket-a/path1" ],
    "id" : "notify-id-1",
    "events" : [ "PutObject", "PutObject" ],
    "status" : "enabled",
    "apps" : [ {
      "eventUrl" : "app:ImageOcr",
      "xVars" : "{\"saveUrl\": \"http://xxx.com/ocr\"}",
      "id" : "app-id-3"
    }, {
      "eventUrl" : "app:ImageOcr",
      "xVars" : "{\"saveUrl\": \"http://xxx.com/ocr\"}",
      "id" : "app-id-3"
    } ]
  } ]
}
```

错误码 请参考通用错误码

设置Bucket通知规则

接口说明 本接口用于指定bucket上增加通知规则。

注意：

- 只有bucket owner或者full control权限才能获取这个bucket的配置。
- 如果不是bucket owner则返回403，如果对应的文件不存在则返回404。

请求URI PUT /v1/{bucketName}/?notification

参数名称	参数类型	是否必须	描述	示例值	参数位置
bucketName	String	是	bucket的名称	"bucketName_example"	Path

请求体参数

请求体字段数据结构说明 参数名称 | 参数类型 | 是否必须 | 描述 | 示例值 ------- | ------ | ------- | ---- | ---- **notifications** | **List<Notifications>** | 否| |

Notifications字段数据结构说明 参数名称 | 参数类型 | 是否必须 | 描述 | 示例值 ------- | ------ | ------- | ---- | ---- **id** | **String** | 是|规则id | notify-id-1 **name** | **String** | 否|规则名称 | rule-name **appId** | **String** | 是|注册本条规则的产品id | app-id-1 **status** | **String** | 是|可选值：{"disabled", "enabled"} | enabled **encryption** | **Encryption** | 否|加密方式 | **resources** | **List<String>** | 是|订阅的资源，${bucket*name}/path1/.jpg或者/path1/*.jpg，最多只能有1个* | **events** | **List<String>** | 是|订阅的事件 | **apps** | **List<Apps>** | 是|订阅消息的产品 |

Encryption字段数据结构说明 参数名称 | 参数类型 | 是否必须 | 描述 | 示例值 ------- | ------ | ------- | ---- | ---- **key** | **String** | 否|加密密钥，如果不为空，则用IAM的算法对通知的请求进行签名，key对应IAM签名中的SecretAccessKey | 06a62b70f47dc4a0a7da349609f1a1ac

Apps字段数据结构说明 参数名称 | 参数类型 | 是否必须 | 描述 | 示例值 ------- | ------ | ------- | ---- | ---- **id** | **String** | 否|被通知的产品id | app-id-3 **eventUrl** | **String** | 否|处理事件通知的url，可选值：{"http", "https", "brn", "app"}，http/https为自定义应用，brn为cfc的通知，app为官方应用 | app:ImageOcr **xVars** | **String** | 否|透传的自定义参数，对框架透明，用于业务自我回传的值，如果是官方AI应用，则是一个字符串化的json，并包含一个saveUrl的地址，用于接收处理结果 | {"saveUrl": "http://xxx.com/ocr\"}

响应体参数

响应体字段数据结构说明

请求示例

```
 PUT /v1/bucketName_example/?notification
<公共请求头>


{
  "notifications" : [ {
    "encryption" : {
      "key" : "06a62b70f47dc4a0a7da349609f1a1ac"
    },
    "appId" : "app-id-1",
    "name" : "rule-name",
    "resources" : [ "bucket-a/path1" ],
    "id" : "notify-id-1",
    "events" : [ "PutObject" ],
    "status" : "enabled",
    "apps" : [ {
      "eventUrl" : "app:ImageOcr",
      "xVars" : "{\"saveUrl\": \"http://xxx.com/ocr\"}",
      "id" : "app-id-3"
    } ]
  } ]
}
```

响应示例

```
 HTTP/1.1 200 OK
<公共响应头>

{ }
```

错误码 请参考通用错误码

删除Bucket的通知规则

接口说明 本接口用于删除指定bucket上的通知规则。

请求URI DELETE /v1/{bucketName}/?notification

参数名称	参数类型	是否必须	描述	示例值	参数位置
bucketName	String	是	bucket的名称	"bucketName_example"	Path

请求体参数

请求体字段数据结构说明

响应体参数

响应体字段数据结构说明

请求示例

```
 DELETE /v1/bucketName_example/?notification
<公共请求头>


{ }
```

响应示例

```
 HTTP/1.1 200 OK
<公共响应头>

{ }
```

错误码 请参考通用错误码

推送ImageOcr和ImageClassify消息

接口说明 将ImageOcr和ImageClassify两种产品处理后的事件消息推送到app设置的url上，内容包含BOS的事件信息和AI处理的结果。

如果规则中配置了encryption字段，那么请求中会包含Authorization的签名，保证消息不会被伪造或篡改。

注意：

- Status Code: 200 OK才会认为消息推送成功，否则将会重试；
- 对于过载保护的情况，支持Status Code: 429 Too Many Requests返回值，会间隔一段时间再重试；

请求URI POST /v1/{bucketName}/?result

参数名称	参数类型	是否必须	描述	示例值	参数位置
bucketName	String	是		"bucketName_example"	Path

请求体参数

请求体字段数据结构说明 参数名称 | 参数类型 | 是否必须 | 描述 | 示例值 ------- | ------ | ------- | ---- | ---- **events** | **List<Events>** | 是| |

Events字段数据结构说明 参数名称 | 参数类型 | 是否必须 | 描述 | 示例值 ------- | ------ | ------- | ---- | ---- **version** | **String** | 否|事件版本，当前为1.0 | 1.0 **eventType** | **String** | 否|事件类型 | PutObject **eventTime** | **String** | 是|

事件发生时间(GMT格式) | 2009-10-28T22:32:00Z **content** | **Content** | 是|事件具体信息 | **imageOcr** | **ImageOcr** | 是| | **imageClassify** | **ImageClassify** | 是| |

Content字段数据结构说明 参数名称 | 参数类型 | 是否必须 | 描述 | 示例值 ------- | ------ | ------- | ---- | ---- **domain** | **String** | 是|所属的domain | bj.bcebos.com **bucket** | **String** | 是|所属的bucket | bucket-test **object** | **String** | 是|object名字 | images/test.jpg **eTag** | **String** | 是|object的eTag，如果object可能被覆盖，则需要用eTag判断是哪个版本的object触发的事件 | 977b9623a15e520c663ac5ff6647e881

ImageOcr字段数据结构说明 参数名称 | 参数类型 | 是否必须 | 描述 | 示例值 ------- | ------ | ------- | ---- | ---- **log_id** | **Integer** | 是| | 153673668620866 **result** | **String** | 是|根据实际的事件为imageOcr和imageClassify，具体内容参见AI通用文字识别和AI通用图像分析 |

ImageClassify字段数据结构说明 参数名称 | 参数类型 | 是否必须 | 描述 | 示例值 ------- | ------ | ------- | ---- | ---- **log_id** | **Integer** | 是| | 6353842115261500000 **result_num** | **Integer** | 是| | 1 **result** | **List<Result>** | 是| |

响应体参数

响应体字段数据结构说明

请求示例

```
 POST /v1/bucketName_example/?result
<公共请求头>


{
  "events" : [ {
    "imageOcr" : {
      "result" : "result",
      "log_id" : 153673668620866
    },
    "eventTime" : "2009-10-28T22:32:00Z",
    "imageClassify" : {
      "result" : [ {
        "score" : 0.566877,
        "root" : "人物-人物特写",
        "keyword" : "马尾辫"
      } ],
      "log_id" : 6353842115261500000,
      "result_num" : 1
    },
    "eventType" : "PutObject",
    "version" : "1.0",
    "content" : {
      "bucket" : "bucket-test",
      "domain" : "bj.bcebos.com",
      "eTag" : "977b9623a15e520c663ac5ff6647e881",
      "object" : "images/test.jpg"
    }
  } ]
}
```

响应示例

```
 HTTP/1.1 200 OK
<公共响应头>

{ }
```

错误码 请参考通用错误码

推送事件消息

接口说明 将事件消息推送到配置的url上。

如果规则中配置了encryption字段，那么请求中会包含Authorization的签名，保证消息不会被伪造或篡改。

注意：

- Status Code: 200 OK才会认为消息推送成功，否则将会重试；
- 对于过载保护的情况，支持Status Code: 429 Too Many Requests返回值，会间隔一段时间再重试；

请求URI POST /v1/{bucketName}/?event

参数名称	参数类型	是否必须	描述	示例值	参数位置
bucketName	String	是	bucket的名称	"bucket Name_e xample"	Path

请求体参数

请求体字段数据结构说明 参数名称 | 参数类型 | 是否必须 | 描述 | 示例值 ------- | ------ | ------- | ---- | ---- **events** | **List<Events>** | 否| |

Events字段数据结构说明 参数名称 | 参数类型 | 是否必须 | 描述 | 示例值 ------- | ------ | ------- | ---- | ---- **version** | **String** | 否|事件版本，当前为1.0 | 1.0 **eventId** | **String** | 否|事件唯一标识 | 2a513199-bbb9-4ac7-b12a-60213c26810d **eventOrigin** | **String** | 否|事件触发源，当前支持增量(bos:realtime) | bos:realtime **eventTime** | **String** | 否|事件发生时间(GMT格式) | 2018-09-05T02:28:49Z **eventType** | **String** | 否|事件类型 | PutObject **content** | **Content** | 否|事件具体信息 |

Content字段数据结构说明 参数名称 | 参数类型 | 是否必须 | 描述 | 示例值 ------- | ------ | ------- | ---- | ---- **userId** | **String** | 否|用户id | c7ac82ae14ef42d1a4ffa3b2ececa17f **domain** | **String** | 否|所属的domain | bj.bcebos.com **bucket** | **String** | 否|所属的bucket | bucket-test **object** | **String** | 否|object名字 | images/test.jpg **eTag** | **String** | 否| object的eTag，如果object可能被覆盖，则需要用eTag判断是哪个版本的object触发的事件 | 977b9623a15e520c663ac5ff6647e881 **contentType** | **String** | 否|object的Content-Type | application/octet-stream **filesize** | **Integer** | 否|文件大小 | 24414 **lastModified** | **String** | 否|文件更新时间 | 2018-09-05T02:28:49Z **credentials** | **Credentials** | 否|相关资源的临时授权，具体使用参考临时授权访问 | **xVars** | **String** | 否| 用户设置通知中的xVars原文 | {"saveUrl": "http://xxx.com/ocr\"}

Credentials字段数据结构说明 参数名称 | 参数类型 | 是否必须 | 描述 | 示例值 ------- | ------ | ------- | ---- | ---- **accessKeyId** | **String** | 否|临时授权的Access Key ID | f3ade7c6b1a911e8bb6821a724cc157d **secretAccessKey** | **String** | 否|临时授权的Secret Access Key | 07381a56d35e4b1193a418362221de81 **sessionToken** | **String** | 否|临时授权的Session Token | ZGZiM2M3MmU4Mjk4NGQ2MGEzYTNhYTAyMDE3NTZmZmV8AAAAAIcCAADaE18IR6jSaVWlMHxZG5wzb/7AQas3Y6V **expiration** | **String** | 否|临时授权的的有效时间 | 2018-02-22T11:22:33Z

响应体参数

响应体字段数据结构说明

请求示例

```
 POST /v1/bucketName_example/?event
<公共请求头>


{
  "events" : [ {
    "eventId" : "2a513199-bbb9-4ac7-b12a-60213c26810d",
    "eventTime" : "2018-09-05T02:28:49Z",
    "eventType" : "PutObject",
    "version" : "1.0",
    "content" : {
      "bucket" : "bucket-test",
      "credentials" : {
        "accessKeyId" : "f3ade7c6b1a911e8bb6821a724cc157d",
        "secretAccessKey" : "07381a56d35e4b1193a418362221de81",
        "sessionToken" : "ZGZiM2M3MmU4Mjk4NGQ2MGEzYTNhYTAyMDE3NTZmZmV8AAAAAIcCAADaE18lR6jSaVWlMHxZG5wzb/7AQas3Y6V",
        "expiration" : "2018-02-22T11:22:33Z"
      },
      "domain" : "bj.bcebos.com",
      "xVars" : "{\"saveUrl\": \"http://xxx.com/ocr\"}",
      "eTag" : "977b9623a15e520c663ac5ff6647e881",
      "filesize" : 24414,
      "lastModified" : "2018-09-05T02:28:49Z",
      "userId" : "c7ac82ae14ef42d1a4ffa3b2ececa17f",
      "contentType" : "application/octet-stream",
      "object" : "images/test.jpg"
    },
    "eventOrigin" : "bos:realtime"
  } ]
}
```

响应示例

```
 HTTP/1.1 200 OK
<公共响应头>

{ }
```

错误码 请参考通用错误码

Bucket数据同步相关接口

获取Bucket数据同步信息

接口说明 本接口用来获取bucket指定id的数据同步信息，包括源Bucket名称、目的Bucket名称、存储类型、是否进行历史复制，数据同步策略，目的region等。

请求URI GET /v1/{bucketName}/?replication

参数名称	参数类型	是否必须	描述	示例值	参数位置
bucketName	String	是	bucket的名称	"bucketName_example"	Path
id	String	是	replication规则名	"id_example"	Query

请求体参数

请求体字段数据结构说明

响应体参数

响应体字段数据结构说明 参数名称 | 参数类型 | 描述 | 示例值 ------ | ------- | ---- | ---- **status** | **String** | 是否生效 | enabled **resource** | **List<String>** | replication生效前缀 | **destination** | **Destination** | 复制的目的端配置 | **replicateHistory** | **ReplicateHistory** | 历史文件复制，有该项则认为是开启 | **replicateDeletes** | **String** | 是否开启删除同步,可以为enabled,disabled | enabled **id** | **String** | replication规则名 | sample-bucket **create_time** | **Integer** | replication规则创建时间 | 1583060606 **destRegion** | **String** | 目的region | bj

Destination字段数据结构说明 参数名称 | 参数类型 | 描述 | 示例值 ------ | ------- | ---- | ---- **bucket** | **String** | 目的Bucket name | dst-bucket-name **storageClass** | **String** | 目的Object的存储类型。如果保持和源Bucket的存储类型一致，则该参数为空；STANDARD表示标准存储类型，STANDARD_IA表示低频存储类型，COLD表示冷存储类型；如果是多AZ类型bucket，可取值为MAZ_STANDARD_IA和MAZ_STANDARD。 | COLD

ReplicateHistory字段数据结构说明 参数名称 | 参数类型 | 描述 | 示例值 ------ | ------- | ---- | ---- **storageClass** | **String** | 目的Object的存储类型，默认为REMAIN,可以为STANDARD,STANDARD_IA,COLD,MAZ_STANDARD_IA,MAZ_STANDARD | COLD

请求示例

```
 GET /v1/bucketName_example/?replication&id=id_example
<公共请求头>


{ }
```

响应示例

```
HTTP/1.1 200 OK
<公共响应头>

{
  "destRegion" : "bj",
  "create_time" : 1583060606,
  "resource" : [ "src-bucket-name/abc", "src-bucket-name/abc" ],
  "destination" : {
    "bucket" : "dst-bucket-name",
    "storageClass" : "COLD"
  },
  "id" : "sample-bucket",
  "replicateHistory" : {
    "storageClass" : "COLD"
  },
  "replicateDeletes" : "enabled",
  "status" : "enabled"
}
```

错误码 请参考通用错误码

设置Bucket数据同步

接口说明 本接口用来创建数据同步。每个id唯一确定一条replication规则，对一个id首次put会认为是创建一条replication规则，对同一个id的再次put认为是覆盖，如果原先status是enable，则不允许覆盖成disable；如果原先是enable，再次put成enable，replication会重新开始执行；不允许两个replication规则除了id一致以外其他项均一致。

说明：

- 用户必须是源Bucket的owner且拥有FULL_CONTROL权限，且有目标Bucket的写权限。
- 目标Bucket和源Bucket必须存在。
- 目标Bucket和源Bucket可以是同region下的Bucket，也可以是不同Region下的Bucket。
- 目标Bucket和源Bucket可以是同账户下的bucket，也可以是不同账户下的bucket，但是源账户需要有目的bucket的写权限
- 整个配置的大小不能超过128k，当前bucket下所有规则长度不得超过200KB 数据同步暂时不支持归档类型文件的同步，进行数据同步时会忽略归档类型文件。
- 用户最多配置10条replication规则
- 单个规则，最多20个resource
- id 由数字字母 - _ 组成，不得超过20个字符

请求URI PUT /v1/{bucketName}/?replication

参数名称	参数类型	是否必须	描述	示例值	参数位置
bucketName	String	是	bucket的名称	"bucketName_example"	Path
id	String	是	replication规则名，id 由数字字母 - _ 组成，不得超过20个字符	"id_example"	Query

请求体参数

请求体字段数据结构说明 参数名称 | 参数类型 | 是否必须 | 描述 | 示例值 ------- | ------ | ------- | ---- | ---- **status** | **String** | 是|是否生效 | enabled **resource** | **List<String>** | 是|replication生效前缀，resource的配置形式为｛$bucket*name/<生效的对象前缀>｝，必须要以$bucket_name+/开头 | **destination** | **Destination** | 是|复制的目的端配置 | **replicateHistory** | **ReplicateHistory** | 否|历史文件复制，有该项则认为是开启。开启历史文件复制后，存量的全部Object都同步复制到目的Bucket，复制范围共用resource。 | **replicateDeletes** | **String** | 是|是否开启删除同步,可以为enabled,disabled 。 | enabled **id** | **String** | 是|replication规则名，id 由数字字母* - 组成，不得超过20个字符 | sample-bucket

Destination字段数据结构说明 参数名称 | 参数类型 | 是否必须 | 描述 | 示例值 ------- | ------ | ------- | ---- | ---- **bucket** | **String** | 是|目的Bucket name | dst-bucket-name **storageClass** | **String** | 否|目的Object的存储类型。如果保持和源Bucket的存储类型一致，则该参数不需要配置；如果需要单独指定存储类型可以为STANDARD，STANDARD_IA，COLD；如果是多AZ类型bucket，可取值为MAZ_STANDARD_IA和MAZ_STANDARD。 | COLD

ReplicateHistory字段数据结构说明 参数名称 | 参数类型 | 是否必须 | 描述 | 示例值 ------- | ------ | ------- | ---- | ---- **storageClass** | **String** | 否|目的Object的存储类型。如果保持和源Bucket的存储类型一致，则该参数不需要配置；如果需要单独指定存储类型可以为STANDARD，STANDARD_IA，COLD；如果是多AZ类型bucket，可取值为MAZ_STANDARD_IA和MAZ_STANDARD。 | COLD

响应体参数

响应体字段数据结构说明

请求示例

```
PUT /v1/bucketName_example/?replication&id=id_example
<公共请求头>


{
  "resource" : [ "src-bucket-name/abc" ],
  "destination" : {
    "bucket" : "dst-bucket-name",
    "storageClass" : "COLD"
  },
  "id" : "sample-bucket",
  "replicateHistory" : {
    "storageClass" : "COLD"
  },
  "replicateDeletes" : "enabled",
  "status" : "enabled"
}
```

响应示例

```
 HTTP/1.1 200 OK
<公共响应头>

{}
```

错误码 请参考通用错误码

删除Bucket的指定id的同步复制配置

接口说明 本接口用来删除对应id的数据同步复制配置。

请求URI DELETE /v1/{bucketName}/?replication

参数名称	参数类型	是否必须	描述	示例值	参数位置
bucketName	String	是	bucket的名称	"bucketName_example"	Path
id	String	是	replication规则名	"id_example"	Query

请求体参数

请求体字段数据结构说明

响应体参数

响应体字段数据结构说明

请求示例

```
 DELETE /v1/bucketName_example/?replication&id=id_example
<公共请求头>


{}
```

响应示例

```
 HTTP/1.1 200 OK
<公共响应头>

{}
```

错误码 请参考通用错误码

获取Bucket指定id数据同步的进程状态

接口说明 本接口用来获取指定id的数据同步复制的进程状态。

请求URI GET /v1/{bucketName}/?replicationProgress

参数名称	参数类型	是否必须	描述	示例值	参数位置
bucketName	String	是	bucket的名称	"bucketName_example"	Path
id	String	是	规则id	"id_example"	Query

请求体参数

请求体字段数据结构说明

响应体参数

响应体字段数据结构说明 参数名称 | 参数类型 | 描述 | 示例值 ------ | ------- | --- | --- **status** | **String** | 是否开启了数据同步功能，可选值为enabled和disabled | enabled **historyReplicationPercent** | **Integer** | 历史文件数据同步已完成百分比，不再有效返回0 | 5 **latestReplicationTime** | **Integer** | UNIX 时间戳，最新的数据复制的时间 | 1504448315

请求示例

```
 GET /v1/bucketName_example/?replicationProgress&id=id_example
<公共请求头>


{}
```

响应示例

```
 HTTP/1.1 200 OK
<公共响应头>

{
  "latestReplicationTime" : 1504448315,
  "historyReplicationPercent" : 5,
  "status" : "enabled"
}
```

错误码 请参考通用错误码

列举Bucket的所有replication同步规则

接口说明 本接口用来获取bucket所有的replication同步规则。

请求URI GET /v1/{bucketName}/?replication&list

参数名称	参数类型	是否必须	描述	示例值	参数位置
bucketName	String	是	bucket的名称	"bucketName_example"	Path

请求体参数

请求体字段数据结构说明

响应体参数

响应体字段数据结构说明 参数名称 | 参数类型 | 描述 | 示例值 ------ | ------- | ---- | ---- **rules** | **List<Rules>** | replication规则列表 |

Rules字段数据结构说明 参数名称 | 参数类型 | 描述 | 示例值 ------ | ------- | ---- | ---- **status** | **String** | 是否生效 | enabled **resource** | **List<String>** | replication生效前缀 |
destination | **Destination** | 复制的目的端配置 |
replicateHistory | **ReplicateHistory** | 历史文件复制，有该项则认为是开启 |
replicateDeletes | **String** | 是否开启删除同步,可以为enabled,disabled | enabled **id** | **String** | replication规则名 | sample-bucket-replication-config **create_time** | **Integer** | replication规则创建时间 | 1583060606
destRegion | **String** | 目的region | bj

Destination字段数据结构说明 参数名称 | 参数类型 | 描述 | 示例值 ------ | ------- | ---- | ---- **bucket** | **String** | 目的Bucket name | dst-bucket-name **storageClass** | **String** | 目的Object的存储类型。如果保持和源Bucket的存储类型一致，则该参数为空；STANDARD表示标准存储类型，STANDARD_IA表示低频存储类型，COLD表示冷存储类型；如果是多AZ类型bucket，MAZ_STANDARD_IA代表多AZ低频存储，MAZ_STANDARD表示多AZ标准存储类型，不能是其它取值。 | COLD

ReplicateHistory字段数据结构说明 参数名称 | 参数类型 | 描述 | 示例值 ------ | ------- | ---- | ---- **storageClass** | **String** | 目的Object的存储类型，默认为REMAIN,可以为STANDARD,STANDARD_IA,COLD,MAZ_STANDARD,MAZ_STANDARD_IA | COLD

请求示例

```
 GET /v1/bucketName_example/?replication&list
<公共请求头>


{ }
```

响应示例

```
 HTTP/1.1 200 OK
<公共响应头>

{
  "rules" : [ {
    "destRegion" : "bj",
    "create_time" : 1583060606,
    "resource" : [ "src-bucket-name/abc", "src-bucket-name/abc" ],
    "destination" : {
      "bucket" : "dst-bucket-name",
      "storageClass" : "COLD"
    },
    "id" : "sample-bucket-replication-config",
    "replicateHistory" : {
      "storageClass" : "COLD"
    },
    "replicateDeletes" : "enabled",
    "status" : "enabled"
  }, {
    "destRegion" : "bj",
    "create_time" : 1583060606,
    "resource" : [ "src-bucket-name/abc", "src-bucket-name/abc" ],
    "destination" : {
      "bucket" : "dst-bucket-name",
      "storageClass" : "COLD"
    },
    "id" : "sample-bucket-replication-config",
    "replicateHistory" : {
      "storageClass" : "COLD"
    },
    "replicateDeletes" : "enabled",
    "status" : "enabled"
  } ]
}
```

错误码 请参考通用错误码

Object相关接口

基础操作

获取对象

接口说明 此命令用于从BOS获取某个Object。此操作需要请求者对该Object有读权限。请求者可以在Header中设置Range来指定需要获取的Object数据的范围。

> **注意：**
>
> - GetObject通过Range参数可以支持断点续传，对于比较大的Object建议使用该功能。
> - 如果在请求头中使用Range参数；则返回消息中会包含整个文件的长度和此次返回的范围，例如：Content-Range: bytes 0-9/44，表示整个文件长度为 44，此次返回的范围为0-9。
> - 对于Range读大小为0字节的对象， 会返回400错误， Range是前闭后闭区间。
> - 归档存储类型对象需要先取回才能调用GetObject接口。

如您想在response请求里面获得某些特定的header信息，可通过如下两种方式：

- 在PutObject时增加header信息，则GetObject时会直接返回在response里面，请参考PutObject接口。
- GetObject时，在queryString里面直接增加“responseXXX=YYY”，返回的header里面会包含“XXX: YYY”。例如：GET /testBucket/testObject?responseContentType= image/jpg，在这个http请求的response里面就会有“Content-Type: image/jpg”。 具体的形式为response$header=urlencode(value)，需要注意，当前$header只支持“ContentDisposition、ContentType、ContentLanguage、Expires、CacheControl、ContentEncoding”。

> **注意：** responseContentDisposition的设置在BOS自有域名($region.bcebos.com和*.$region.bcebos.com无效)。

请求URI GET /v1/{bucketName}/{objectKey}

参数名称	参数类型	是否必须	描述	示例值	参数位置
bucketName	String	是	bucket的名称	"bucketName_example"	Path
objectKey	String	是	object的名称	"objectKey_example"	Path

请求头参数 参数名称 | 参数类型 | 是否必须 | 描述 | 示例值 ------- | ------ | ------ | ---- | ---- | range | String | 否 | 指定Object返回的文件范围。设定 bytes=0-9，表示传送第0到第9这10个字符。默认返回全部数据 | "range_example"

请求体参数

请求体字段数据结构说明

响应头参数

参数名称	参数类型	描述	示例值
Cache-Control	String	下载Object的Cache设置，常见的可取值为private、no-cache、max-age、must-revalidate	
Content-Disposition	String	设置浏览器是否下载，可取值为inline、attachment;filename="download.txt"	
Content-Length	String	返回Object的数据大小	
Content-Range	String	有range的情况下返回Object的数据范围	
Content-Type	String	Object的类型及编码方式	
Expires	String	下载Object时的缓存失效时间	
ETag	String	Object的HTTP协议实体标签	
x-bce-meta-*	String	如果有自定meta，才返回此项	
x-bce-storage-class	String	标准存储返回STANDARD，低频存储返回STANDARD_IA，冷存储返回COLD，归档存储返回ARCHIVE；如果是多AZ类型bucket，多AZ低频存储返回MAZ_STANDARD_IA，多AZ标准存储返回MAZ_STANDARD。	
x-bce-server-side-encryption	String	Object的服务器端加密类型，当前只支持AES256加密。	

响应体参数

响应体字段数据结构说明

请求示例

```
 GET /v1/bucketName_example/objectKey_example
<公共请求头>
Range: range_example


{}
```

响应示例

```
 HTTP/1.1 200 OK
<公共响应头>

{}
```

错误码 请参考通用错误码

获取对象的meta信息

接口说明 此命令用于获取某个Object的Meta信息，但此时并不返回数据。 **注意**：

- 如果使用软链接访问GetObjectMeta, 返回响应头中Content-Length、ETag、Content-Md5 均为目标文件的元信息；Last-Modified是目标文件和软链接的最大值；其他均为软链接元信息。
- 如果使用软链接访问该接口且目标文件删除了，会返回Http 404，SymlinkTargetNotExist,调用GetSymlink不会报错。

请求URI HEAD /v1/{bucketName}/{objectKey}

参数名称	参数类型	是否必须	描述	示例值	参数位置
bucketName	String	是	bucket的名称	"bucketName_example"	Path
objectKey	String	是	object的名称	"objectKey_example"	Path

请求体参数

请求体字段数据结构说明

响应头参数

参数名称	参数类型	描述	示例值
Cache-Control	String	下载Object的Cache设置，常见的可取值为private、no-cache、max-age、must-revalidate	
Content-Disposition	String	设置浏览器是否下载，可取值为inline、attachment;filename="download.txt"	
Content-Length	String	返回Object的数据大小	
Content-Range	String	有range的情况下返回Object的数据范围	
Content-Type	String	Object的类型及编码方式	
Expires	String	下载Object时的缓存失效时间	
ETag	String	Object的HTTP协议实体标签	
x-bce-meta-*	String	果有自定meta，才返回此项	
x-bce-storage-class	String	标准存储返回STANDARD，低频存储返回STANDARD_IA，冷存储返回COLD，归档存储返回ARCHIVE；如果是多AZ类型bucket，多AZ低频存储返回MAZ_STANDARD_IA，多AZ标准存储返回MAZ_STANDARD。	
x-bce-server-side-encryption	String	Object的服务器端加密类型，当前只支持AES256加密。	
x-bce-restore	String	当对象为归档存储类型，并且处于正在取回或已经取回时，会返回头域"x-bce-restore"，表示归档类型对象的取回状态：正在取回时返回ongoing-request="true"；已取回归档对象时，返回ongoing-request="false", expiry-date="Wed, 07 Nov 2019 00:00:00 GMT"。其中expiry-date表示对象取回后的失效时间。	
x-bce-object-type	String	当访问的是软链接，则返回值"Symlink"。其他情况暂不返回	

响应体参数

响应体字段数据结构说明

请求示例

```
 HEAD /v1/bucketName_example/objectKey_example
<公共请求头>


{}
```

响应示例

```
 HTTP/1.1 200 OK
<公共响应头>

{}
```

错误码 请参考通用错误码

删除Object

接口说明 此命令用于删除指定Bucket的一个Object，要求请求者对此Object有写权限。

请求URI DELETE /v1/{bucketName}/{objectKey}

参数名称	参数类型	是否必须	描述	示例值	参数位置
bucketName	String	是	bucket的名称	"bucketName_example"	Path
objectKey	String	是	object的名称	"objectKey_example"	Path

请求体参数

请求体字段数据结构说明

响应体参数

响应体字段数据结构说明

请求示例

```
 DELETE /v1/bucketName_example/objectKey_example
<公共请求头>


{}
```

响应示例

```
 HTTP/1.1 200 OK
<公共响应头>

{}
```

错误码 请参考通用错误码

取回归档文件

接口说明 RestoreObject用于取回归档存储文件，请求者必须有归档存储文件的读权限，并且归档存储文件处于冰冻状态。

注意：

- 如果使用软链接访问该接口,会根据软链接的目标文件的存储类型和状态进行处理。
- 如果使用软链接访问该接口且目标文件删除了，会返回Http 404，SymlinkTargetNotExist。

请求URI POST /v1/{bucketName}/{objectKey}?restore

参数名称	参数类型	是否必须	描述	示例值	参数位置
bucketName	String	是	bucket的名称	"bucketName_example"	Path
objectKey	String	是	object的名称	"objectKey_example"	Path

请求头参数 参数名称 | 参数类型 | 是否必须 | 描述 | 示例值 ------- | ------ | ------ | ---- | ---- | xBceRestoreDays | int | 否 | 示解冻后状态的持续时间，要求是数字，单位是天，取值大于0且不超过30即可。默认解冻持续天数是7天，最长30天。 | 7| x-bce-restore-tier | string | 否 | 解冻优先级。可取值范围如下：
Expedited：加急取回，表示30min内完成取回
Standard：标准取回，表示2~5小时内完成取回
LowCost：延缓取回，表示12小时内完成取回| Expedited|

请求体参数

请求体字段数据结构说明

响应体参数

响应体字段数据结构说明

请求示例

```
 POST /v1/bucketName_example/objectKey_example?restore
<公共请求头>
x-bce-restore-days: 7


{}
```

响应示例

```
 HTTP/1.1 200 OK
<公共响应头>

{}
```

错误码 请参考通用错误码

抓取Object并存储到Bucket中

接口说明 此接口用于从指定URL抓取资源，并将资源存储到指定的Bucket中。此操作需要请求者对该Bucket有写权限，每次只能抓取一个Object，且用户可以自定义Object的名称。

说明：

- FetchObject接口抓取资源的大小限制为0~10GB。

请求URI POST /v1/{bucketName}/{objectKey}?fetch

参数名称	参数类型	是否必须	描述	示例值	参数位置
bucketName	String	是	bucket的名称	"bucketName_example"	Path
objectKey	String	是	object的名称	"objectKey_example"	Path

请求头参数 参数名称 | 参数类型 | 是否必须 | 描述 | 示例值 ------- | ------ | ------ | ---- | ---- | xBceFetchSource | String | 是 | 抓取文件的源地址，如http://www.abc.com/img.jpg。该参数可以放到querystring里，当放到querystring时需要做UrlEncode。 | "xBceFetchSource_example" xBceFetchMode | String | 否 | 抓取模式，支持异步抓取async和同步抓取sync两种模式。其中异步模式下BOS收到抓取任务后立刻返回成功，同步模式下BOS需要等到抓取Object完成后才会返回成功。对于数据量小或关注抓取结果实时性的建议使用同步模式，异步模式不需要等待，适用不需要实时查看抓取结果的场景，可以后续再对抓取结果进行查询。默认值为

sync。 该参数可以放到querystring里。 | "xBceFetchMode_example" xBceStorageClass | String | 否 | 存储类型，支持标准存储STANDARD、低频存储STANDARD_IA、冷存储COLD和归档存储ARCHIVE。默认存储为STANDARD。 该参数可以放到querystring里。 | "xBceStorageClass_example" xBceServerSideEncryption | String | 否 | 服务端加密算法，当前仅支持AES256。 | "xBceServerSideEncryption_example"

请求体参数

请求体字段数据结构说明

响应体参数

响应体字段数据结构说明 参数名称 | 参数类型 | 描述 | 示例值 ------ | ------- | ---- | ---- **errors** | **List<Errors>** | 抓取过程中出错的Object信息的容器，里面包含一个或多个Object元素。 |

Errors字段数据结构说明 参数名称 | 参数类型 | 描述 | 示例值 ------ | ------- | ---- | ---- **key** | **String** | 抓取出错的Object名称。 | my-object2 **code** | **String** | 错误代码。 | InvalidArgument **message** | **String** | 错误信息。 | Invalid Argument.

请求示例

```
 POST /v1/bucketName_example/objectKey_example?fetch
<公共请求头>
x-bce-fetch-source: xBceFetchSource_example
x-bce-fetch-mode: xBceFetchMode_example
x-bce-storage-class: xBceStorageClass_example
x-bce-server-side-encryption: xBceServerSideEncryption_example


{ }
```

响应示例

```
 HTTP/1.1 200 OK
<公共响应头>

{
  "errors" : [ {
    "code" : "InvalidArgument",
    "message" : "Invalid Argument.",
    "key" : "my-object2"
  }, {
    "code" : "InvalidArgument",
    "message" : "Invalid Argument.",
    "key" : "my-object2"
  } ]
}
```

错误码 请参考通用错误码

删除多个Object

接口说明 该命令可以实现通过一个HTTP请求删除同一个Bucket下的多个Object。

- 支持一次请求内最多删除1000个Object。
- 消息体（body）不超过2M。
- 返回的消息体中只包含删除过程中出错的Object结果；如果所有Object都删除都成功的话，则没有消息体。

请求URI POST /v1/{bucketName}/?delete

参数名称	参数类型	是否必须	描述	示例值	参数位置
bucketName	String	是	bucket的名称	"bucketName_example"	Path

请求体参数

请求体字段数据结构说明 参数名称 | 参数类型 | 是否必须 | 描述 | 示例值 ------- | ------ | ------- | ---- | ---- **objects** | **List<Objects>** | 是|保存要删除的Object信息的容器，里面包含一个或多个Object元素 |

Objects字段数据结构说明 参数名称 | 参数类型 | 是否必须 | 描述 | 示例值 ------- | ------ | ------- | ---- | ---- **key** | **String** | 否| | my-object2

响应体参数

响应体字段数据结构说明 参数名称 | 参数类型 | 描述 | 示例值 ------ | ------- | ---- | ---- **errors** | **List<Errors>** | 删除过程中出错的Object信息的容器，里面包含一个或多个Object元素。 |

Errors字段数据结构说明 参数名称 | 参数类型 | 描述 | 示例值 ------ | ------- | ---- | ---- **key** | **String** | 删除出错的Object名称。 | my-object2 **code** | **String** | 错误代码。 | InvalidArgument **message** | **String** | 错误信息。 | Invalid Argument.

请求示例

```
 POST /v1/bucketName_example/?delete
<公共请求头>


{
  "objects" : [ {
    "key" : "my-object2"
  } ]
}
```

响应示例

```
 HTTP/1.1 200 OK
<公共响应头>

{
  "errors" : [ {
    "code" : "InvalidArgument",
    "message" : "Invalid Argument.",
    "key" : "my-object2"
  }, {
    "code" : "InvalidArgument",
    "message" : "Invalid Argument.",
    "key" : "my-object2"
  } ]
}
```

错误码 请参考通用错误码

权限控制

获取Object的访问权限

接口说明 此命令用来获取某个Object的访问权限。

请求URI GET /v1/{bucketName}/{objectKey}?acl

参数名称	参数类型	是否必须	描述	示例值	参数位置
bucketName	String	是	bucket的名称	"bucketName_example"	Path
objectKey	String	是	object的名称	"objectKey_example"	Path

请求体参数

请求体字段数据结构说明

响应体参数

响应体字段数据结构说明 参数名称 | 参数类型 | 描述 | 示例值 ------ | ------- | ---- | ---- **accessControlList** | **List<AccessControlList>** | ACL访问控制列表 |

AccessControlList字段数据结构说明 参数名称 | 参数类型 | 描述 | 示例值 ------ | ------- | ---- | ---- **grantee** | **List<Grantee>** | 一个被授权人 | **permission** | **List<String>** | 授权权限支持：FULL_CONTROL，READ |

Grantee字段数据结构说明 参数名称 | 参数类型 | 描述 | 示例值 ------ | ------- | ---- | ---- **id** | **String** | 被授权用户名id | 8c47a952db4444c5a097b41be3f24c94

请求示例

```
 GET /v1/bucketName_example/objectKey_example?acl
<公共请求头>


{ }
```

响应示例

```
 HTTP/1.1 200 OK
<公共响应头>

{
  "accessControlList" : [ {
    "grantee" : [ {
      "id" : "8c47a952db4444c5a097b41be3f24c94"
    }, {
      "id" : "8c47a952db4444c5a097b41be3f24c94"
    } ],
    "permission" : [ "FULL_CONTROL", "FULL_CONTROL" ]
  }, {
    "grantee" : [ {
      "id" : "8c47a952db4444c5a097b41be3f24c94"
    }, {
      "id" : "8c47a952db4444c5a097b41be3f24c94"
    } ],
    "permission" : [ "FULL_CONTROL", "FULL_CONTROL" ]
  } ]
}
```

错误码 请参考通用错误码

设置Object的访问权限

接口说明 此命令用于设置Object的访问权限。目前BOS支持两种方式设置ACL。第一种是使用Canned Acl，在PutObjectAcl的时候，通过头域的"x-bce-acl"或者"x-bce-grant-permission'来设置object访问权限，当前可设置的权限包括private和public-read，两种类型的header不可以同时在一个请求中出现。第二种方式是上传一个ACL文件，文件格式参见ACL文件格式，目前ACL文件只支持accessControlList，grantee，id，permission字段。

目前不支持在同一请求中同时设置canned acl和上传ACL文件。

注意事项：

- 只有Bucket的拥有者和被授予FULL_CONTROL权限的用户才能设置Object的ACL权限。
- 在上传Object时，Object权限会默认为空，如果没有设置Object的权限，即当Object权限为空时，默认以Bucket权限为准。
- 当Object权限和Bucket权限不一致时，以Object权限为准。

- 归档存储类型对象在取回未完成，或者刚上传归档类型文件（时长参考取回时长）时，不能设置Object acl。

请求URI PUT /v1/{bucketName}/{objectKey}?acl

请求头参数 参数名称 | 参数类型 | 是否必须 | 描述 | 示例值 ------- | ------ | ------ | ---- | ---- | xBceAcl | String | 否 | Object设置的ACL权限，支持：private、public-read | "xBceAcl_example" xBceGrantRead | String | 否 | 授权读的Object id，支持多个id，以逗号分隔 | "xBceGrantRead_example" xBceGrantFullControl | String | 否 | 授权控制权限的Object id，支持多个id，以逗号分隔 | "xBceGrantFullControl_example"

请求体参数 无

响应体参数 无

请求示例 无

响应示例 无

错误码 请参考通用错误码

删除Object的访问权限

接口说明 本接口用来删除某个Object的访问权限。

> **注意事项：**
>
> - 归档存储类型对象在取回未完成，或者刚上传归档类型文件（时长参考取回时长）时，不能删除Object acl。

请求URI DELETE /v1/{bucketName}/{objectKey}?acl

参数名称	参数类型	是否必须	描述	示例值	参数位置
bucketName	String	是	bucket的名称	"bucketName_example"	Path
objectKey	String	是	object的名称	"objectKey_example"	Path

请求体参数

请求体字段数据结构说明

响应体参数

响应体字段数据结构说明

请求示例

```
 DELETE /v1/bucketName_example/objectKey_example?acl
<公共请求头>


{}
```

响应示例

```
 HTTP/1.1 200 OK
<公共响应头>

{}
```

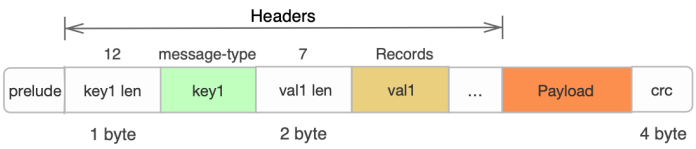
错误码 请参考通用错误码

Select扫描

向Object执行Sql语句

接口说明 此接口用于向Bucket中指定object执行SQL语句，选取出指定内容返回。请求者必须对选取的object具有read权限。在SelectObject前需要确保对应的Bucket和Object已经存在，详细信息参考 SelectObject 开发者文档。

message具体格式详细说明：



1. prelude部分一共8个byte，前4byte表示message总长度，后4byte表示headers部分总长度，chunk总长度(prelude前4byte存储的值) - header总长度 - 8byte的prelude - 4byte的crc32 = payload部分数据总长度；crc32表示整个message的纠删码
2. headers包含以下自定义<key,value>："message-type": {"Records", "Cont", "End"}；"error-code": 具体错误码；"error-message": "详细错误信息"
3. Payload表示返回的真实数据，可以是任意格式；其中Continuation message的Payload里边包括BytesScanned和BytesReturned两个字段表示的select进度信息

请求URI POST /v1/{bucketName}/{objectKey}?select

参数名称	参数类型	是否必须	描述	示例值	参数位置
bucketName	String	是	bucket的名称	"bucketName_example"	Path
objectKey	String	是	object的名称	"objectKey_example"	Path
type	String	是	select的目标object类型，目前支持json/csv	"type_example"	Query

请求体参数

请求体字段数据结构说明 参数名称 | 参数类型 | 是否必须 | 描述 | 示例值 ------- | ------ | ------- | ---- | ---- **selectRequest** | **SelectRequest** | 是|json body根节点 |

SelectRequest字段数据结构说明 参数名称 | 参数类型 | 是否必须 | 描述 | 示例值 ------- | ------ | ------- | ---- | ---- **expression** | **String** | 是|经过base64编码的sql语句 | c2VsZWN0IGNvdW50KCopIGZyb20gbxkl2JqZWN0IHdoZXJlIF80ID4gNDU= **expressionType** | **String** | 是|查询语句语法类型，只支持"SQL" | SQL **inputSerialization** | **InputSerialization** | 是|输入流节点，其子节点描述查询的object格式信息 | **outputSerialization** | **OutputSerialization** | 是|输出流节点，其子节点描述查询结果返回格式信息 | **requestProgress** | **RequestProgress** | 否| + requestProgress 否 - select进度信息节点，其子节点描述select操作的执行进度，3s定时返回给用户 |

InputSerialization字段数据结构说明 参数名称 | 参数类型 | 是否必须 | 描述 | 示例值 ------- | ------ | ------- | ---- | ---- **compressionType** | **String** | 否|指定查询的object是否压缩，可选值为"NONE"或"GZIP" | NONE **csv** | **Csv** | 否| csv节点，其子节点描述csv文件相关信息 |

Csv字段数据结构说明 参数名称 | 参数类型 | 是否必须 | 描述 | 示例值 ------- | ------ | ------- | ---- | ---- **fileHeaderInfo** | **String** | 否|可选值NONE/IGNORE/USE，指定CSV文件首行头信息，默认值NONE代表无header信息，IGNORE代表有header信息但是忽略，NONE和IGNORE意味着只能用列序号取出某一列；USE表示使用header信息，只能用header名取出某一列 | NONE **recordDelimiter** | **String** | 否|指定csv文件换行符，以Base64编码，默认值为\n（可选），最多为2个字符，例如\r\n | \r\n **fieldDelimiter** | **String** | 否| 指定csv文件列分隔符，以Base64编码，默认值为,（可选），最多为1个字符，例如; | ; **quoteCharacter** | **String** | 否|指定csv文件引号字符，以Base64编码，在CSV中引号内的换行符，列分隔符将被视作普通字符；默认值为双引号"（可选），最多为1个字符，例如单引号' | ' **commentCharacter** | **String** | 否|指定csv文件注释符，以Base64编码，一行以注释符开头则表示改行可以忽略；默认值为#（可选），最多为2个字符，例如// | //

OutputSerialization字段数据结构说明 参数名称 | 参数类型 | 是否必须 | 描述 | 示例值 ------- | ------ | ------- | ---- | ---- **outputHeader** | **Boolean** | 否|在返回结果开头输出CSV头信息，默认false; 需要fileHeaderInfo字段取值USE，该值为true会在返回结果的每一个Records message第一行加上对应的csv header名称；可选值为false/true | false **csv** | **Csv** | 否|csv节点，其子节点描述返回的csv数据相关信息 |

RequestProgress字段数据结构说明 参数名称 | 参数类型 | 是否必须 | 描述 | 示例值 ------- | ------ | ------- | ---- | ---- **enabled** | **Boolean** | 否|描述是否需要定时返回进度信息，可选值为false/true；如果数据筛选耗时久可能导致504超时，设置其为true可以维持http连接 | false

响应体参数

响应体字段数据结构说明

请求示例

```
POST /v1/bucketName_example/objectKey_example?select&type=type_example
<公共请求头>


{
  "selectRequest" : {
    "outputSerialization" : {
      "outputHeader" : false,
      "csv" : {
        "quoteFields" : "ALWAYS",
        "recordDelimiter" : "Cg==",
        "quoteCharacter" : "Ig==",
        "fieldDelimiter" : "LA=="
      }
    },
    "expression" : "c2VsZWN0IGNvdW50KCopIGZyb20gbxkl2JqZWN0IHdoZXJlIF80ID4gNDU=",
    "expressionType" : "SQL",
    "requestProgress" : {
      "enabled" : false
    },
    "inputSerialization" : {
      "csv" : {
        "fileHeaderInfo" : "NONE",
        "recordDelimiter" : "\\r\\n",
        "quoteCharacter" : "'",
        "fieldDelimiter" : ";",
        "commentCharacter" : "//"
      },
      "compressionType" : "NONE"
    }
  }
}
```

响应示例

```
HTTP/1.1 200 OK
<公共响应头>

{}
```

错误码 请参考通用错误码

软链接

获取软链接

接口说明 GetSymlink接口用于获取软链接。此操作需要您对该软链接有读权限。

请求URI GET /v1/{bucketName}/{objectKey}?symlink

参数名称	参数类型	是否必须	描述	示例值	参数位置
bucketName	String	是	bucket的名称	"bucketName_example"	Path
objectKey	String	是	object的名称	"objectKey_example"	Path

请求体参数

请求体字段数据结构说明

响应头参数

参数名称	参数类型	描述	示例值
x-bce-symlink-target	String	软链接指向的目标文件	

响应体参数

响应体字段数据结构说明

请求示例

```
 GET /v1/bucketName_example/objectKey_example?symlink
<公共请求头>


{}
```

响应示例

```
 HTTP/1.1 200 OK
<公共响应头>

{}
```

错误码 请参考通用错误码

创建软链接

接口说明 PutSymlink接口用于为BOS的相同bucket下已有的目的object（本文档称作"target object"）创建软链接（Symlink），您可以通过该软链接访问目的object。

注意：

- 软链接文件和target object文件需要位于同一个bucket下面。
- 目前支持对标准存储、低频存储、冷存储、归档存储类型的object文件设置软链接。软链接文件本身不支持ARCHIVE类型。如果用户指定ARCHIVE类型，或者不指定类型且Bucket默认存储类型是ARCHIVE时，返回报错CODE_INVALID_ARGUMENT。
- 调用该接口创建软链接时，不会检查目标文件是否存在、以及是否有权限访问目标文件。使用软链接调用GetObject接口时候，会检查用户是否有软链接和目标文件的读取权限、目标文件是否存在。
- 创建软链接需要有相应写权限。
- 如果不指定"x-bce-forbid-overwrite"这个Header，如果存在与软链接文件同名的object文件时，创建软链接操作时会默认覆盖同名Object。
- 不支持二级软链接。用户创建二级软链时不会进行检查，但是用户无法通过二级软链访问指向的object数据。

请求URI PUT /v1/{bucketName}/{objectKey}?symlink

请求头参数 参数名称 | 参数类型 | 是否必须 | 描述 | 示例值 ------- | ------ | ------ | ---- | ---- | xBceSymlinkTarget | String | 是 | 指定软链接的目标文件。命名规范同Object。软链接的目标文件不能为软链接文件 | "xBceSymlinkTarget*example" xBceForbidOverwrite | String | 否 | 指定创建软链接操作时是否覆盖同名Object。如果不指定x-bce-forbid-overwrite时，默认允许覆盖同名Object。指定x-bce-forbid-overwrite为true时，表示禁止覆盖同名Object；指定x-bce-forbid-overwrite为false时，表示允许覆盖同名Object。 | "xBceForbidOverwrite_example" xBceMeta* | String | 否 | 用户自定义的meta | "xBceMeta__example" xBceStorageClass | String | 否 | 指定软链接文件的存储类型，STANDARD代表标准存储，STANDARD_IA代表低频存储，COLD代表冷存储,不指定时默认是bucket存储类型。软链接文件本身不支持ARCHIVE类型。 | "xBceStorageClass_example"

请求体参数 无

响应头参数

参数名称	参数类型	描述	示例值
ETag	String	软链接目标Object名字的HTTP协议实体标签	

响应体参数

无

请求示例 无

响应示例

```
 HTTP/1.1 200 OK
<公共响应头>

{}
```

错误码 请参考通用错误码

图像审核服务接口

图像审核接口

接口说明 此接口用于请求图像审核服务。图像审核服务是百度智能云对外提供的图像智能审核服务，支持对图像的多个维度进行审核，包括色情识别、暴恐识别、政治敏感识别、恶心图识别、广告识别等。

> **说明:** 使用本接口需要先在控制台开通图像审核服务。 url：固定取值为 $(img-censor)，不需要修改。 parameters：此项值是对图像审核功能参数进行 base64 编码而成。图像审核功能参数依然是个 json 字符串。

图像审核服务支持的子服务列表有：

- ocr：通用文字识别
- face：人脸探测
- antiporn：色情识别
- politician：政治人物识别
- terror：暴恐识别
- public: 公众人物识别
- disgust: 恶心图识别
- watermark：水印二维码识别
- quality：图像质量识别 value 为各底层服务参数，参照河图 OCR 通用文字识别服务入参。

详细的子服务请求参数及返回参数解释参见下文详细解释。

请求URI POST /v1/{bucketName}/{objectKey}?process

参数名称	参数类型	是否必须	描述	示例值	参数位置
bucketName	String	是	bucket的名称	"bucketName_example"	Path
objectKey	String	是	object的名称	"objectKey_example"	Path

请求体参数

请求体字段数据结构说明 参数名称 | 参数类型 | 是否必须 | 描述 | 示例值 ------- | ------ | ------- | ---- | ---- **action** | **Action** | 否| |

Action字段数据结构说明 参数名称 | 参数类型 | 是否必须 | 描述 | 示例值 ------- | ------ | ------- | ---- | ---- **sync** | **List<Sync>** | 否| |

Sync字段数据结构说明 参数名称 | 参数类型 | 是否必须 | 描述 | 示例值 ------- | ------ | ------- | ---- | ---- **url** | **String** | 否|固定取值为 $(img-censor)，不需要修改 | $(img-censor) **parameters** | **String** | 否|此项值是对图像审核功能参数进行 base64 编码而成。图像审核功能参数依然是个 json 字符串 | <base64_encode(param)>

响应体参数

响应体字段数据结构说明

请求示例

```
 POST /v1/bucketName_example/objectKey_example?process
<公共请求头>


{
  "action" : {
    "sync" : [ {
      "parameters" : "<base64_encode(param)>",
      "url" : "$(img-censor)"
    } ]
  }
}
```

响应示例

```
 HTTP/1.1 200 OK
<公共响应头>

{ }
```

错误码 请参考通用错误码

BOS-CMD命令行工具

概述

BCE CMD是百度智能云对象存储（BOS）提供的客户端命令行工具。BCE CMD当前只包含了BOS的功能，所有下文都用BOS CMD代指BCE CMD。BOS CMD提供了丰富的功能，您不仅可以通过BOS CMD完成Bucket的创建和删除，Object的批量上传、下载、删除和拷贝等，当您从BOS下载文件或者上传文件到BOS遇到问题时，还可以使用BCE CMD的子命令bosprobe进行错误检查。

安装BOSCMD

BOS CMD下载地址

文件名及下载地址	操作系统	架构	大小	MD5
windows-bcecmd-0.5.6.zip	Windows	x86-64	7.8MB	zip：3b785df5704b653830d39740feef3d4b bcecmd.exe:c367010fcbbae088d6c59289294d06cd
linux-bcecmd-0.5.6.zip	Linux	x86-64	7.6MB	zip:0053f6e2dc1e0c7920887dd3b2138cf4 bcecmd:d7370f81e283afa73908a4e1f823bb47
mac-bcecmd-0.5.6.zip	Mac	x86-64	7.4MB	zip:ac53688676c9447ae8d7d89966e450ed bcecmd:a1d2e0dec12c1fae1c614c1a48cb0c03
arm64-linux-bcecmd-0.5.6.zip	Linux	arm64	7.1MB	zip:cc0488895df97a5c0e6136139b9846936 bcecmd:1c3815ec321ba0727d456ade0f74b004
arm64-mac-bcecmd-0.5.6.zip	Mac	arm64	7.1MB	zip:2667d4b2f80219fec84e3c1640134c6a bcecmd:eb9c22658affa1c79fa7c1667cdfca50

安装

请根据自己的操作系统类型，下载对应版本的bcecmd，下载完成后只需解压Zip包到您需要执行程序的目录，无需安装，您可以直接在终端中使用。

说明：

- Linux 用户解压命令为 `$ unzip linux-bcecmd-0.5.6.zip`；
- Mac 用户解压命令为 `$ unzip mac-bcecmd-0.5.6.zip`；
- 建议您将 bcecmd 所在目录加入操作系统的环境变量中，这样您在任何目录下都能够运行 bcecmd；~~~~
- Windows用户请在Windows cmd中运行程序， Linux和Mac用户请在终端中运行程序。
- bcecmd解压后命令只能在解压文件路径中调用，若需要全局使用，需要生成链接到/usr/sbin目录，命令为`$ ln linux-bcecmd-0.5.6/bcecmd /usr/sbin/ `；

配置BOSCMD

通用配置

配置支持中文显示

如需要配置支持中文显示的CMD环境，请执行如下命令配置：

```
$ export LANG=en_US.UTF-8
```

> **说明：**可先执行locale命令查看当前环境的语言支持情况，若不是LANG=en_US.UTF-8，可执行上述命令进行修改。

配置AK/SK、Region、Host信息

使用BOS CMD工具之前，推荐先设置Access Key、Secure Key、Region和Host。可以通过-c/--configure来设置AK、SK、Region和Host信息。

```
 $ ./bcecmd -c  [<conf-path>]
$ Your Access Key ID [None]: Enter Your AK
$ Your Security Token []: Enter Your STS token
$ Your Secret Access Key ID [None]: Enter Your SK
$ Default Region Name [bj]: Enter Your region
$ Default Domain [bj.bcebos.com]: Enter Your host
$ Default Proxy Host [none]: Enter Your proxy host
$ Default breakpoint_file_expiration [7] days:
$ Default use path style [yes]:
```

> **说明：**
>
> - <conf-path>为本地文件目录，当用户有多个百度智能云账户时，通过指定<conf-path>将不同账号的配置文件写到不同目录下，执行命令时可以指定目录选择对应账户的配置。
> - 不指定<conf-path>时，CMD在配置完成后将配置文件自动写到当前用户主目录~/.go-bcecli；指定<conf-path>时，将配置文件自动写到指定目录。
> - 即使不设置AK/SK，您也可以通过BOS CMD访问public权限的bucket。
> - 指定Region之后您可以不指定Host，BOS CMD可自动拼接Host为Region.bcebos.com。
> - 如果您指定了Host，那么BOS CMD不再自动拼接Host，而是访问您指定的Host。
> - 可以填写None将AK/SK/Region以及Domain设置为空。当Region设置为空时默认值为bj。当Domain设置为空时默认使用Region拼接，当两个同时为空时默认访问**bj.bcebos.com**。
> - 指定proxy host后CMD客户端将使用代理访问BOS服务
> - 在使用临时STS token的时候可以设置Token,在设置Token的情况下，优先使用Token
> - breakpoint_file_expiration 断点文件的有效期，超过有效期之后，之前的断点不生效
> - use path style指定发送BOS请求时是否使用path style方式，默认使用path style，及请求中host为3级域名，url的path中携带bucket名字

BOS CMD相关配置

bcecmd -c命令除了可以进行通用设置，还支持设置自动切换域名，另还支持设置BOS上传的相关属性，如是否使用HTTPS协议及分块上传的并发数等。

设置自动切换域名

BOS拥有多个Region，如果您需要在不同Region下面进行操作，BOS CMD的"自动切换域名"功能可以根据您当前操作的Bucket所在Region，来自动切换域名。为了方便您的使用，CMD会默认开启此功能。

如果您想让您所有的操作请求都发送到您指定的域名或者Host（通过bcecmd -c配置的domain），你可以通过bcecmd -c命令关闭"自动切换域名"功能, 设置为no表示关闭此功能。

```
 $ ./bcecmd -c [<conf-path>]
$ Default use auto switch domain [yes]:
```

> **说明：**取值可以是yes和no，yes代表开启根据Bucket所在Region自动切换域名，no表示关闭。缺省情况下为开启。

设置HTTPS协议上传

为了保证用户数据的安全性，建议用户使用HTTPS协议上传。是否使用HTTPS可以通过bcecmd -c命令进行设置，设置为no代表使用HTTP协议上传。

```
 $ ./bcecmd -c [<conf-path>]
$ Default use https protocol [no]:
```

> **说明：**取值可以是yes和no，缺省情况下使用HTTP协议上传。

设置大文件分块上传的并发数

使用BOS CMD上传大于32MB的文件时，BOS CMD会自动进行分块上传，您可以通过bcecmd -c命令设置单个文件分块上传的并发数。

```
$ ./bcecmd -c [<conf-path>]

$ Default multi upload thread num [10]:
```

> **说明：**分块上传的并发数应设置为大于等于1的整数，缺省为10。

设置BOS CMD sync命令并发上传下载的文件数

BOS CMD 通过sync命令执行上传或下载时，会并发操作多文件。您可以通过`bcecmd -c`命令设置 sync 命令并发数。

```
$ ./bcecmd -c [<conf-path>]

$ Default sync processing num [10]:
```

> **说明：**并发数应设置为大于等于1的整数，缺省为10。
>
> **注意：**使用sync命令时，sync会并发地处理多个文件，如果有文件大小超过分块上传的阈值时，这个文件也会被分块后并发上传， 所以最坏的情况下将有 “sync并发数 x 单文件分块上传并发数” 个协程同时运行， 但是并不是并发数越高上传效率越高，所以您需要根据自己电脑配置来配置合适的sync并发数和分块上传并发数。

设置分块上传的单个分块大小

BCE CMD默认的单个分块大小为10MB，如果您网络环境良好（使用默认分块大小上传文件未出现超时），建议您就使用默认的分块大小。 当上传出现超时时（当BOS返回的错误码为RequestTimeout），建议你调低单个分块的大小来避免上传超时。您可以通过如下的命令配置分块大小：

```
$ ./bcecmd -c [<conf-path>]

$ Default multi upload part size [10] MB (Must be positive integer and equal or greater than 1) :
```

> **说明：**
>
> - 分块大小应设置为大于等于1的整数，缺省为10 （单位为MB），BOS分块上传时Part数目范围是1-10000，当采用缺省值时可以上传的单个Object最大为100GB，当上传大文件时需要调整改参数，Part数目在1000以内较佳。
> - 动态调整：如果分片大小设置较小，例如设置为 1MB 那么当文件大小超过10GB（但不超过BOS支持最大文件48.8TB）Part数目会超过10000，此时BCE CMD工具会自动调整分片，降低Part数目保证文件正常上传。
> - 如果您网络环境很差， 建议您也调低[sync命令并发上传下载的文件数](#设置BOS CMD sync命令并发上传下载的文件数)和 大文件分块上传的并发数。

通过CMD使用BOS服务

BOS 级命令

Object管理

Object上传

描述： BOS CMD支持同步上传、单文件上传、批量上传三种方式，上传过程中可以指定Object的存储类型是标准存储、低频存储或冷存储。此外BOS CMD还支持：

- 对于大于32M的文件自动分块并发上传，并设置上传的并发连接数等。

当您需要在本地和BOS之间进行数据上传，同步上传（sync）是最方便的上传命令。同步上传默认支持批量操作，可以将本地目录同步到BOS。如果BOS端已经有名称相同且修改时间比本地文件新，同步上传会忽略此文件，只同步新文件或有改动的文件。 sync命令会同时列举本地和BOS侧文件, 对于不同情况分别执行以下动作：

- 对于新增的文件（本地的某个文件BOS侧没有）：上传
- 对于已上传但是有改动的文件（本地文件的更新时间晚于BOS侧文件的创建时间, 或者本地文件的大小跟BOS侧不一致）：覆盖上传
- 对于已上传且没改动的文件：跳过不处理
- 对于已上传但是在本地没有的文件：根据--delete参数来做不同的处理。
- 对于sync命令来说需要配置粗粒度LIST权限 或 细粒度列表LIST权限以及GetBucketLocation才能生效
- 上传进度信息保存在 `--conf-path` 指定目录下的 `task_progress` 子目录下的文件，文件名为 `PID-<UNIX TimeStamp>`，如： `10516-1752479760`

命令格式：

- 同步上传：$ bcecmd bos sync <local_dir> bos:/<bucket_name>/[prefix] [--exclude EXCLUDE] [--include INCLUDE] [--delete] [--exclude-delete EXCLUDE-DELETE] [--dryrun] [--yes] [--quiet] [--storage-class STORAGE-CLASS] [--sync-type SYNC-TYPE] [--concurrency CONCURRENCY] [--restart] [--cache-control cacheControl] [--content-type CONTENT-TYPE] [--expires EXPIRES] [--content-disposition CONTENT-DISPOSITION] [--user-meta USER-META]

  > **参数说明：**
  >
  > - `local_dir`必须为本地目录，不能为单个文件。也不支持通配符匹配的语法，如`<local_dir>/*` 以及`<local_dir>/prefix*`等。
  > - 在windows和linux系统下, 硬链接文件和普通文件的处理方式一样，软链接文件夹和普通文件夹的处理方式一样,但软链接文件(指向文件的软链接)和普通文件的处理方式不一样，sync会操作软链接文件指向的文件而不是软链接文件本身，但是软链接本身被删除, 也会触发sync。
  > - --exclude：执行本地到bos的sync的时候, 需要跳过的文件, 支持通配符*和多项配置,通配符*可能会被shell解析，需要加上引号 如要跳过tmp文件和当前目录所有子目录下的svn目录:`--exclude './*.tmp' --exclude '*/.svn'`
  > - --include：执行本地到bos的sync的时候, 只同步指定文件, 支持通配符*和多项配置, 如只同步当前目录所有子目录下的log文件和html文件:`--include './*.log' --include './*.html' --include "./*.log"` (windows只能双引号)
  > - --delete：删除BOS有而本地没有的文件，使BOS和本地的数据完全一致。执行后默认会提示用户是否要执行删除。
  > - --exclude-delete：此选项需要同 `--delete` 一起使用， 用于在删除本地文件时排除特定文件或目录， 具体用法请参考 `--exclude`的使用规则。
  > - --dryrun：列出会执行的操作但不实际执行。

- --yes：执行删除动作时不需要确认直接删除。
- --quiet：同步上传但不输出任何中间过程。
- --storage-class [STORAGE_CLASS]：通过storage-class参数指定同步上传的Object的存储类型。指定storage-class为STANDARD_IA、COLD时表示存储为低频存储、冷存储,指定的storage-class为STANDARD或者不指定storage-class时为标准存储。
- --sync-type: 同步类型可以为 time-size， time-size-crc32 ， only-crc32 和 force-overwrite， 默认为 time-size。 bcecmd 执行 sync 时首先会判断要同步的文件在目的端是否存在，如果不存在就会上传或者下载文件， 当源端和目的端都存在时， 就会根据同步类型来决定是否需要同步文件。
  - time-size: 比较源端和目的端文件的修改日期和大小来判断是否同步，当源端文件的修改日期早于目的端文件时，不会同步文件；当源端文件的修改日期等于目的端文件的修改日期，且它们的大小相同时不同步文件， 否则同步;
  - time-size-crc32: bcecmd 先使用 time-size 模式判断是否需要同步文件， 当time-size 输出结果为需要同步文件时, bcecmd 将比较源端和目的端文件的 crc32， 当crc32的值不同时才会同步文件;
  - only-crc32: bcecmd 只会比较源端和目的端文件的crc32， 如果不同就会同步文件。
  - force-overwrite: 当遇到源端与目的端同名文件，bcecmd 将强制用源端文件覆盖目的端文件。
- --concurrency: 用于设置 sync 命令并发上传下载的文件数，默认为10。
- --restart: 忽略断点续传，重头开始上传文件。
- --cache-control: 指定文件缓存策略，常见值为private,public,no-cache,no-store。
- --expires: 定义http页面的缓存过期时间，默认为3天。
- --content-type:定义网络文件的类型和网页的编码。
- --content-disposition:如何显示附加的文件,常见值为inline,attachment。
- --user-meta:用户自定义meta值，格式为name1:value1,name2:value2,name3:value3。
- --traffic-limit:单链接限速，限速值的取值范围为819200~838860800，单位为bit/s，即100KB/s~100MB/s

注意

- sync命令需要您的aksk拥有fullcontrol权限
- 当--delete 与过滤（include 或exclude）同时使用时，如果一个文件被过滤，尽管它同时存在于源端和目的端， 它也会被删除。因为bcecmd只会比较未被过滤的文件。
- 需要注意 BOS 在2018年1月之后才全面支持CRC32, 所以当 bcecmd sync-type 选择为time-size-crc32或only-crc32时，如果bcecmd 未能从BOS获取到 object 的 CRC32信息时， bcecmd 将总是会同步这些文件。

- 单文件上传：$ bcecmd bos cp <local-path> <bos-path> [--storage-class [STORAGE_CLASS]] [--restart] [--quiet] [--yes] [--disable-bar]
- 批量上传：$ bcecmd bos cp <local-path> <bos-path> --recursive [--storage-class [STORAGE_CLASS]] [--restart] [--quiet] [--yes] [--disable-bar]

说明：

- 执行bcecmd bos cp /tmp/file1 bos:/bucket/file2命令时，file2可以与file1相同，如果file2在bucket中已存在则会被覆盖。
- 执行bcecmd bos cp /tmp/file bos:/bucket/命令时，自动使用本地文件名作为bos端文件名，如果已存在重名文件会被覆盖。
- 执行bcecmd bos cp /tmp/file bos:/bucket/prefix[/] -r命令时，本地为文件时会退化为单文件上传，bos路径结尾为"/"时prefix会当做目录，新文件保存在prefix下一级；结尾不为"/"时会当做object名直接上传。
- 执行bcecmd bos cp /tmp/dir/ bos:/bucket/prefix[/] -r命令时，本地为目录时bos路径结尾不论是否有"/"都会以prefix为目录名在bos端新建目录； 并将dir中Object全部拷贝到prefix目录下。
- 执行批量上传时，local-path只能是本地目录且以"/"结尾，bos-path必须为bucket/prefix。
- 上传Object过程中可以通过storage-class参数指定Object的存储类型，storage-class默认为STANDARD即标准存储，storage-class为STANDARD_IA、COLD时表示存储为低频存储、冷存储。如果storage-class输入参数错误依旧会按照STANDARD进行存储，不会提示错误信息。
- --disable-bar：禁止使用进度条。

示例：

- **同步当前目录下的sample目录到bos:/bce-test/test下**

  操作示例： bcecmd bos sync ./sample/ bos:/bce-test/test

  返回示例：

```
 Upload: sample/BOS_Batch_Delete.png to bos:/bce-test/test/BOS_Batch_Delete.png
Upload: sample/sample.log to bos:/bce-test/test/sample.log
```

- **列出会执行的同步操作但不执行**

  操作示例： bcecmd bos sync ./sample/ bos:/bce-test/test --dryrun

  返回示例：

```
 Upload: sample/BOS_DataImport.png to bos:/bce-test/test/BOS_DataImport.png
```

- **同步上传但不输出任何中间过程**

  操作示例：bcecmd bos sync ./sample/ bos:/bce-test/test --quiet

  返回示例： 无返回

- **同步sample目录到bos:/bce-test/test上且删除BOS有而本地没有的文件**

  操作示例：bcecmd bos sync ./sample/ bos:/bce-test/test --delete

  返回示例：

```
 Delete object: bos:/bce-test/test/sample.log
```

- **同步sample目录到bos:/bce-test/test上并存储为低频存储类型**

  操作示例： `bcecmd bos sync ./sample/ bos:/bce-test/test --storage-class STANDARD_IA`

  返回示例：

  ```
  Upload: sample/BOS_DataImport.png to bos:/bce-test/test/BOS_DataImport.png
  ```

- **同步sample目录到bos:/bce-test/test上并存储为冷存储类型**

  操作示例： `bcecmd bos sync ./sample/ bos:/bce-test/test --storage-class COLD`

  返回示例：

  ```
  Upload: sample/BOS_DataImport.png to bos:/bce-test/test/BOS_DataImport.png
  ```

- **单文件上传**

  操作示例：

  ```
  $ bcecmd bos cp text.txt bos:/mybucket/test.txt
  ```

  返回示例：

  ```
  Upload: test.txt to bos:/mybucket/test.txt
  ```

- **批量上传**

  操作示例：

  ```
  $ bcecmd bos cp ./temp/ bos:/bce-test -r
  ```

  返回示例：

  ```
  Upload: /home/users/username/temp/BAE.png to bos:/bce-test/BAE.png
  Upload: /home/users/username/tem/CreatePolicy.png to bos:/bce-test/CreatePolicy.png
  Upload: /home/users/username/tem/BAE_Pro.png to bos:/bce-test/BAE_Pro.png
  Upload: /home/users/username/tem/BCD.png to bos:/bce-test/BCD.png
  Upload: /home/users/username/tem/04Eng.png to bos:/bce-test/04Eng.png
  [5] object uploaded.
  ```

- **上传Object并存储为低频存储类型**

  操作示例：

  ```
  $ bcecmd bos cp text.txt bos:/mybucket/test.txt --storage-class STANDARD_IA
  ```

  返回示例：

  ```
  Upload: test.txt to bos:/mybucket/test.txt
  ```

- **上传Object并存储为冷存储类型**

  操作示例：

  ```
  $ bcecmd bos cp text.txt bos:/mybucket/test.txt --storage-class COLD
  ```

  返回示例：

  ```
  Upload: test.txt to bos:/mybucket/test.txt
  ```

Object下载

描述： BOS CMD支持同步下载、单个文件下载和批量文件下载。

同步下载默认支持批量操作，可以将BOS同步到本地目录。如果本地已经有同名文件，且本地文件修改时间晚于BOS上的同名文件，同步下载会忽略这些文件，只同步新文件或有改动的文件。

sync命令会同时列举本地和BOS侧文件, 对于不同情况分别执行以下动作：

- 对于新增的文件（BOS的某个文件本地没有）：下载
- 对于已下载但是有改动的文件（BOS侧文件的创建时间晚于本地文件的更新时间, 或者BOS侧文件的大小跟本地不一致）：覆盖下载
- 对于已下载且没改动的文件：跳过不处理
- 对于已下载但是BOS侧没有的文件：根据--delete参数来做不同的处理。
- 下载进度信息保存在 `--conf-path` 指定目录下的 `task_progress` 子目录下的文件中，文件名为 PID-<UNIX TimeStamp>，如： `10516-1752479760`

命令格式：

- 同步下载Object: $ bcecmd bos sync <bos-dir> <local-dir> [--exclude EXCLUDE] [--include INCLUDE] [--delete] [--exclude-delete EXCLUDE-DELETE] [--dryrun] [--yes] [--quiet] [--sync-type SYNC-TYPE] [--download-tmp-path DOWNLOAD-TMP-PATH] [--concurrency CONCURRENCY] [--restart] [--rand]

  > **说明：**
  >
  > - 同步下载中，local-dir和bos-dir必须为目录，不能为单个文件。
  > - --exclude：执行bos到本地的sync的时候, 需要跳过的文件, 支持通配符*和多项配置, 如要跳过bucket sample下tmp文件和所有目录下的svn目录:`--exclude 'bos:/sample/*.tmp' --exclude 'bos:/sample/*/.svn/*'`
  > - --include：执行bos到本地的sync的时候, 只同步指定文件, 支持通配符*和多项配置, 如只同步bucket sample下的log文件和html文件： `--include 'bos:/sample/*.log' --include 'bos:/sample/*.html'`
  > - --delete：删除本地有而BOS没有的文件，使本地和BOS的数据完全一致。执行后默认会提示用户是否要执行删除。
  > - --exclude-delete：此选项需要同 `--delete` 一起使用，用于在删除在目的端存在而在源端不存在的文件时排除目的端特定文件或目录，具体用法请参考 `--exclude`的使用规则。
  > - --dryrun：列出会执行的操作但不实际执行。

- --yes：执行删除动作时不需要确认直接删除。
- --quiet：同步下载但不输出任何中间过程。
- --sync-type: 同步类型可以为 time-size， time-size-crc32， only-crc32 和 force-overwrite， 默认为 time-size。 bcecmd 执行 sync 时首先会判断要同步的文件在目的端是否存在，如果不存在就会上传或者下载文件， 当源端和目的端都存在时， 就会根据同步类型来决定是否需要同步文件。
  - time-size: 比较源端和目的端文件的修改日期和大小来判断是否同步，当源端文件的修改日期早于目的端文件时，不会同步文件；当源端文件的修改日期等于目的端文件的修改日期，且它们的大小相同时不同步文件， 否则同步;
  - time-size-crc32: bcecmd 先使用 time-size 模式判断是否需要同步文件， 当time-size 输出结果为需要同步文件时, bcecmd 将比较源端和目的端文件的 crc32， 当crc32的值不同时才会同步文件;
  - only-crc32: bcecmd 只会比较源端和目的端文件的crc32， 如果不同就会同步文件。
  - force-overwrite: 当遇到源端与目的端同名文件，bcecmd 将强制用源端文件覆盖目的端文件。
- --download-tmp-path：用于手动指定断点下载时用于保存临时文件的目录， 如果不指定， 默认使用执行命令时所在目录。
- --concurrency: 用于设置 sync 命令并发上传下载的文件数，默认为10。
- --restart: 忽略断点续传，重头开始上传文件。
- --rand: 随机下载bos文件到本地。

注意

- 当--delete 与过滤（include 或exclude）同时使用时，如果一个文件被过滤，尽管它同时存在于源端和目的端， 它也会被删除。因为bcecmd只会比较未被过滤的文件。
- 需要注意 BOS 在2018年1月之后才全面支持CRC32, 所以当 bcecmd sync-type 选择为time-size-crc32或only-crc32时，如果bcecmd 未能从BOS获取到 object 的 CRC32信息时， bcecmd 将总是会同步这些文件。

- 下载单个Object：$ bcecmd bos cp <bos-path> <local-path> [--restart] [--storage-class STORAGE-CLASS] [--download-tmp-path DOWNLOAD-TMP-PATH] [--quiet] [--yes] [--disable-bar]
- 批量下载Object：$ bcecmd bos cp <bos-path> <local-path> --recursive [--restart] [--storage-class STORAGE-CLASS] [--download-tmp-path DOWNLOAD-TMP-PATH] [--quiet] [--yes] [--disable-bar]

说明：

- 执行bcecmd bos cp bos:/bucket/file /tmp/file命令时，本地文件名可以任意指定，若本地文件存在会提示是否覆盖。
- 执行bcecmd bos cp bos:/bucket/file /tmp/命令时，本地文件名会自动存为bos端文件名，若存在会提示是否覆盖。
- 执行bcecmd bos cp bos:/bucket/file /tmp/ -r命令时，bos端为文件时会退化为单个文件下载，若存在会提示是否覆盖。
- 执行bcecmd bos cp bos:/bucket/prefix/ /tmp/ -r命令时，本地路径如果为一个已存在的文件会报错，本地目录不存在时会创建。
- 执行批量下载时，local-path只能是本地目录，bos-path必须为bucket/prefix。如果bos路径为Object，则为单个Object下载。
- --restart: 忽略断点续传，重头开始下载文件。
- --storage-class：下载时不管是否指定参数storage-class，都不影响下载，该参数会被自动忽略，不起作用。
- --download-tmp-path：用于手动指定断点下载时用于保存临时文件的目录， 如果不指定， 默认使用执行命令时所在目录。
- --quiet：同步下载但不输出任何中间过程。
- --yes：执行删除动作时不需要确认直接删除。
- --disable-bar： 不显示进度条。

示例：

- **同步下载bos:/mybucket/pre/下文件到当前目录下的temp目录**

  操作示例：

  ```
  $ bcecmd bos sync bos:/mybucket/pre/ ./temp/
  ```

  返回示例：

  ```
  Download: bos:/mybucket/pre/sts1.png to /home/users/username/temp/sts1.png
  Download: bos:/mybucket/pre/sts2.png to /home/users/username/temp/sts2.png
  Sync done: bos:/mybucket/pre/ to ./temp/, [2] success [0] failure
  ```

- **下载Object**

  操作示例：

  ```
  $ bcecmd bos cp bos:/mybucket/test.txt text.txt
  ```

  返回示例：

  ```
  Download: bos:/mybucket/test.txt to test.txt
  ```

- **批量下载Object**

  操作示例：

  ```
  $ bcecmd bos cp bos:/mybucket/pre/ temp -r
  ```

  返回示例：

  ```
  Download: bos:/mybucket/pre/sts1.png to /home/users/username/tmp/sts1.png
  Download: bos:/mybucket/pre/sts2.png to /home/users/username/tmp/sts2.png
  [2] objects downloaded.
  ```

拷贝Object

描述： BOS CMD支持在bucket间拷贝文件，支持同步拷贝、单个文件拷贝和批量文件拷贝。

同步拷贝默认支持批量操作，如果目的Bucket已经有同名文件，同步拷贝会忽略这些文件，只同步新文件或有改动的文件。

sync命令会同时列举源Bucket和目的Bucket的文件, 对于不同情况分别执行以下动作：

- 对于新增的文件（源Bucket的某个文件目的Bucket没有）：拷贝
- 对于已拷贝但是有改动的文件（源Bucket文件的创建时间晚于目的Bucket文件的更新时间, 或者源Bucket侧文件的大小跟目的Bucket不一致）：覆盖拷贝
- 对于已拷贝且没改动的文件：跳过不处理
- 对于已拷贝但是源Bucket没有的文件：根据--delete参数来做不同的处理。
- 拷贝进度信息保存在 `--conf-path` 指定目录下的 `task_progress` 子目录下的文件中，文件名为 `PID-<UNIX TimeStamp>`，如： `10516-1752479760`

命令格式：

- 同步拷贝Object：`$ bcecmd bos sync <bos-dir> <bos-dir> [--exclude EXCLUDE] [--include INCLUDE] [--delete] [--exclude-delete EXCLUDE-DELETE] [--dryrun] [--yes] [--quiet] [--storage-class STORAGE-CLASS] [--sync-type SYNC-TYPE] [--concurrency CONCURRENCY] [--restart]`

  说明：

  - 同步拷贝中，bos-dir必须为目录，不能为单个文件。
  - --exclude：执行bos到bos的sync的时候, 需要跳过的文件, 支持通配符*和多项配置, 如要跳过bucket source下tmp文件和所有目录下的svn目录: `--exclude 'bos:/source/*.tmp' --exclude 'bos:/source/*/.svn/*'`
  - --include：执行bos到bos的sync的时候, 只同步指定文件, 支持通配符*和多项配置, 如只同步bucket source下的log文件和html文件： `--include 'bos:/source/*.log' --include 'bos:/source/*.html'`
  - --delete：删除源Bucket有而目的Bucket没有的文件，使源和目的Bucket的数据完全一致。执行后默认会提示用户是否要执行删除。
  - --exclude-delete：此选项需要同 `--delete` 一起使用，用于在删除在目的端存在而在源端不存在的文件时排除目的端特定文件或目录，具体用法请参考 `--exclude`的使用规则。
  - --dryrun：列出会执行的操作但不实际执行。
  - --yes：执行删除动作时不需要确认直接删除。
  - --quiet：同步上传但不输出任何中间过程。
  - --storage-class [STORAGE_CLASS]：通过storage-class参数指定同步上传的Object 的存储类型。指定storage-class为`STANDARD_IA`、`COLD`时表示存储为低频存储、冷存储,指定的storage-class为`STANDARD`或者不指定storage-class时为标准存储。
  - --sync-type: 同步类型可以为 `time-size`， `time-size-crc32`, `only-crc32` 和 `force-overwrite`， 默认为 `time-size`。 bcecmd 执行 sync 时首先会判断要同步的文件在目的端是否存在，如果不存在就会同步文件，当源端和目的端都存在时， 就会根据同步类型来决定是否需要同步文件。
    - `time-size`: 比较源端和目的端文件的修改日期和大小来判断是否同步，当源端文件的修改日期早于目的端文件时，不会同步文件；当源端文件的修改日期等于目的端文件的修改日期，且它们的大小相同时不同步文件， 否则同步;
    - `time-size-crc32`: bcecmd 先使用 `time-size` 模式判断是否需要同步文件， 当`time-size` 输出结果为需要同步文件时, bcecmd 将比较源端和目的端文件的 crc32， 当crc32的值不同时才会同步文件;
    - `only-crc32`: bcecmd 只会比较源端和目的端文件的crc32， 如果不同就会同步文件。
    - `force-overwrite`: 当遇到源端与目的端同名文件，bcecmd 将强制用源端文件覆盖目的端文件。
  - --concurrency: 用于设置 sync 命令并发拷贝的文件数。
  - --restart: 忽略断点续传，重头开始拷贝文件。

  注意

  - 当--delete 与过滤（include 或exclude）同时使用时，如果一个文件被过滤，尽管它同时存在于源端和目的端，它也会被删除。因为bcecmd只会比较未被过滤的文件。
  - 需要注意 BOS 在2018年1月之后才全面支持CRC32, 所以当 bcecmd sync-type 选择为`time-size-crc32`或`only-crc32`时，如果bcecmd 未能从BOS获取到 object 的 CRC32信息时， bcecmd 将总是会同步这些文件。

- 拷贝单个Object：`$ bcecmd bos cp <bos-path> <bos-path> [--restart] [--storage-class STORAGE-CLASS] [--quiet] [--yes] [--disable-bar]`
- 批量拷贝Object：`$ bcecmd bos cp <bos-path> <bos-path> --recursive [--restart] [--storage-class STORAGE-CLASS] [--quiet] [--yes] [--disable-bar]`

  说明：

  - 执行bcecmd bos cp bos:/bucket/obj1 bos:/bucket/obj2命令时，复制的两个路径相同时会报错，目的路径若已存在会被覆盖。
  - 执行bcecmd bos cp bos:/bucket/obj1 bos:/bucket/prefix/命令时，obj1会被拷贝到prefix目录下，目的路径必须以“/”结尾，否则会当做一个新文件名新建一个文件。
  - 执行bcecmd bos cp bos:/bucket/obj1 bos:/bucket/prefix/ -r命令时，源路径为单个文件时退化为单个文件拷贝，obj1会被拷贝到prefix目录下。
  - 执行bcecmd bos cp bos:/bucket/prefix1/ bos:/bucket/prefix2/ -r命令，实现目录到目录的拷贝，两个路径都必须以“/”的结尾；若目的路径结尾不以“/”结尾会当做单个文件从而报错。
  - 执行批量拷贝时，bos-path必须为bucket/prefix。
  - 用户必须具备对源bucket的读权限和目标bucket的写权限。

示例：

- **同步拷贝bos:/mybucket/pre/下文件到bos:/mybucket2/pre/**

  操作示例：

  ```
  $ bcecmd bos sync bos:/mybucket/pre/ bos:/mybucket2/pre/
  ```

  返回示例：

  ```
  [----------------------------------------------] 100.00% 2.52 MiB p/s 3s
  Sync done: bos:/mybucket/pre/ to bos:/mybucket2/pre/, [2] success [0] failure
  ```

- **拷贝单个Object**

  操作示例：

  ```
  $ bcecmd bos cp bos:/mybucket/test.txt bos:/mybucket2/test.txt
  ```

  返回示例：

  ```
  Copy: bos:/mybucket/test.txt to bos:/mybucket2/test.txt
  ```

- **批量拷贝Object**

  操作示例：

  ```
  $ bcecmd bos cp bos:/mybucket/pre1/ bos:/bce-test/pre/ -r
  ```

  返回示例：

  ```
  [----------------------------------------------------] 100.00% 45.86 MiB p/s 0s
  [2] objects remote copied.
  ```

罗列Object

描述： ls命令可以列出一个Bucket内的不超过1000条Objects，可以通过指定Prefix来过滤结果。ls会自动以/作为delimiter来达到模拟文件夹的效果。

命令格式： $ bcecmd bos ls bos:/<bucket-name>/[prefix]

操作示例： bcecmd bos ls bos:/mybucket/web

返回示例：

```
                           PRE  pre/
2016-04-01 16:43:01       8698    STANDARD  web_bos.png
2016-04-01 16:41:49      17188    STANDARD  web_server_bos.png
2016-04-01 16:43:53      24620  STANDARD_IA  webtobos_cors.png
2016-04-01 16:45:53      24623        COLD  cold_abc.png
```

> **说明**
>
> - -a, --all：超过1000个Object时会全部显示出来。
> - -r, --recursive：不显示目录，直接显示下面的Object。
> - -s, --summerize：显示个数以及总大小等统计信息。
> - 示例输出第一列代表Object的创建日期，第二列代表创建时间，第三列代表Object的大小（以Byte为单位），第四列表示Object的存储类型，第五列表示Object的名称。
> - pre/表示为文件夹，不会显示创建时间及大小信息。

获取Object的URL

描述： gen_signed_url命令可以获取指定Object的URL，根据提供的URL可以下载Object。

命令格式 $ bcecmd bos gen_signed_url bos:/<bucket-name>/<object-key> [-eTIME]

> **说明：**
>
> - 获取的URL地址输入浏览器中可以直接下载Object。生成URL前，需要确认配置的AK/SK与Bucket对应账户的AK/SK一致，否则生成的URL无效不能正常下载Object。
> - -e: 指定URL过期时间（单位秒），值必须大于等于-1，其中-1表示URL永久有效。
> - 不指定过期时间时默认过期时间为1800秒。
>
> **注意：** 参数-e 与过期时间之前不能有空格。

示例：

- **不指定过期时间**

  操作示例：

  ```
  $ bcecmd bos gen_signed_url bos:/bce-test/test/BOS_DataImport.png
  ```

  返回示例：

  ```
  http://bj.bcebos.com/bce-test/test/BOS_DataImport.png?authorization=bce-auth-v1%2Fd35a0b8750114e49b3567e0b0542c2f1%2F2016-08-25T02%3A03%3A57Z%2F1800%2F%2F96d77180ec2cb7e8de522c15ca42bf67c30bae0edb47bad495fe8e7953794a4f
  ```

- **指定过期时间为100秒**

  操作示例：

  ```
  $ bcecmd bos gen_signed_url bos:/bce-test/test/BOS_DataImport.png -e100
  ```

  返回示例：

  ```
  http://bj.bcebos.com/bce-test/test/BOS_DataImport.png?authorization=bce-auth-v1%2F637986f41b0046248e3a333817371502%2F2017-08-31T03%3A19%3A42Z%2F100%2F%2F3e77accca7f0206460c662b0a9596b7ec353599214d42b124bb797a48da7359a
  ```

- **指定URL永久有效**

  操作示例：

  ```
  $ bcecmd bos gen_signed_url bos:/bce-test/test/BOS_DataImport.png -e-1
  ```

返回示例：

```
http://bj.bcebos.com/bce-test/test/BOS_DataImport.png?authorization=bce-auth-v1%2F637986f41b0046248e3a333817371502%2F2017-08-31T03%3A19%3A42Z%2F-12F%2F3e77accca7f0206460c662b0a9596b7ec353599214d42b124bb797a48da7359a
```

删除Object

命令格式：

- 删除单个Object：`$ bcecmd bos rm [-y, --yes] bos:/<bucket-name>/<object-key>`
- 批量删除Object：`$ bcecmd bos rm [-y, --yes] [-r, --recursive] bos:/<bucket-name>/<prefix>`

> **说明：**
>
> - -y, --yes: 使用该选项时跳过确认步骤。
> - -r, --recursive：批量删除object。

示例：

- **删除单个Object**

  操作示例：

  ```
  $ bcecmd bos rm bos:/bce-temp/s1.png
  ```

  返回示例：

  ```
  Do you really want to REMOVE object bos:/bce-temp/s1.png? (Y/N) **Y**
  Delete object: bos:/bce-temp/s1.png
  ```

- **批量删除Object**

  操作示例：

  ```
  $ bcecmd bos rm bos:/bce-test/pre/ -r
  ```

  返回示例：

  ```
  Delete object: bos:/bce-test/pre/s1.png
  Delete object: bos:/bce-test/pre/s2.png
  [2] objects removed on remote.
  ```

取回object 描述： `restore`命令用于取回指定object，支持批量取回。

命令格式：

- `$ bcecmd bos restore bos://${bucket-name}/{object-key} --restore-tier STANDARD --restore-days 7 -r`

> **说明：**
>
> - --bucket-name: bucket名称
> - --object-key：要取回的object名称
> - --restore-tier：解冻优先级。Expedited：加急取回，表示30min内完成取回。
>   Standard：标准取回，表示2~5小时内完成取回。LowCost：延缓取回，表示12小时内完成取回。
> - --restore-days：表示解冻后状态的持续时间，要求是数字，单位是天，取值大于0且不超过30即可。默认解冻持续天数是7天,最长30天。
> - -r: 当object-key为文件夹时，需添加-r参数，批量取回该前缀下的所有object

示例：

操作示例：

```
$ bcecmd bos restore bos://testbucket/testobject
```

Bucket管理

创建Bucket

描述： 创建一个空的Bucket。

命令格式： `$ bcecmd bos mb bos:/<bucket-name> [--region REGION]`

> **说明：**
>
> - 仅在“自动切换域名”功能开启后，您才能为Bucket指定所属地域。
> - 如果您没有指定Bucket所属地域，我们将使用您在CMD中配置的默认地域（即使用`bcecmd -c`配置的默认地域，如果您没有配置默认地域，CMD的默认地域将为bj）。

示例：

- **不指定创建Bucket所属地域**

  操作示例：

  `$ bcecmd bos mb bos:/mybucket`

  返回示例：

  `Make bucket: mybucket in region **`

- **指定创建Bucket所属地域**

  操作示例：

  `$ bcecmd bos mb bos:/mybucket -r bj`

  返回示例：

  `Make bucket: mybucket in region bj`

Bucket删除

描述： 删除一个Bucket，要求Bucket内无Object和未完成的三步上传Part。

命令格式： `$ bcecmd bos rb bos:/<bucket-name> [-y, --yes] [-f, --force]`

> **说明：**
>
> - -y, --yes：使用该选项时，默认跳过确认步骤。
> - -f, --force：如果需要删除非空Bucket，可以使用-f/--force选项进行强制删除。使用该选项时，Bucket不为空也可以将该Bucket及其内所有Objects一起删除。

操作示例： `$ bcecmd bos rb bos:/mybucket --force`

返回示例：

```
 Do you really want to REMOVE bucket bos:/bce-test and all objects in it? (Y/N) **Y**
Delete object: bos:/mybucket/test1.txt
Delete object: bos:/mybucket/test2.txt
Remove bucket: mybucket
```

罗列Bucket

描述： 罗列Bucket列表。

命令格式： `$ bcecmd bos ls [bos:/]`

操作示例： `$ bcecmd bos ls`

返回示例：

```
 2015-08-12 10:20:58      bj  bce-doc
2015-08-12 10:21:09      bj  bce-sdk
2016-07-19 11:11:41      bj  bce-temp
2016-07-19 10:53:37      bj  bce-test
```

> **说明：**示例输出第一列代表bucket的创建日期，第二列代表创建时间，第三列代表bucket所在区域，第四列表示Bucket的名称。

CMD通用参数

用户可以通过bos命令来访问BOS服务，管理、操作Bucket和Object。如果BOS服务余额不足时进行操作会提示Access Denied。

CMD使用多层命令结构，所有的命令以bcecmd开头，[options]表示BCE CMD支持的选项，<service>表示CMD所支持的服务，如BOS等，每个服务拥有多个特定的子命令。

```
$ bcecmd [options] [<service> <command> [parameters [options]]]
```

参数解释

参数项	取值	说明
options	BCE通用的Option，取值为**-c**、**-v**、**-h**、**-d**和**--conf-path**。	-
service	取值为**bos**、**bosprobe**。	bosprobe的使用方法请查看bosprobe
command	取值为：**mb**、**rb**、**ls**、**cp**、**rm**。	各命令的作用请参见命令功能描述。
parameters	取值为**<local-path>**或**<bos-path>**。	<bos-path>包含以下几种形式： - BOS根目录：bos:/ - BOS Bucket根目录：bos:/<bucket-name> - BOS Bucket下文件夹：bos:/<bucket-name>/<folder-name>/ - BOS Bucket下的Object：bos:/<bucket-name>/<folder-name>/<object-key> 注：在<bos-path>中Bucket后面如果出现连续多个"/"会处理成一个"/"。 参数的值可以是简单的字符串或数字，如`$ bcecmd bos cp localfile.txt bos:/mybucket/remotefile.txt`。如果字符串中包含空格及特殊字符，在Linux中需要使用单引号，在Windows中需要使用双引号包围。如`$ bcecmd bos cp 'my object 1' 'bos:/mybucket/my object 1'`
+options	BOS命令的Option，取值为**-r**、**-a**或**-s**等。	BOS各命令可配置的option不一样，详细的使用方法请参见各命令行解释。

CMD通用Option

- 查看帮助：可以在任意命令后面添加`-h`或`--help`来查看该命令的帮助信息。

  ```
  $ bcecmd -h
  $ bcecmd bos -h
  $ bcecmd bos ls --help
  ```

- 查看、设置AK、SK等配置信息：`-c/--configure`

  ```
  $ bcecmd -c [<conf-path>]
  ```

- 查看版本信息：`-v/--version`

```
$ bcecmd -v
```

- 查看debug信息：-d/--debug

```
$ bcecmd -d bos ls bos:/bce-test/pre/
```

> **注意：** -d/--debug命令不能单独使用，必须与bos命令结合使用。

- 使用指定配置文件：--conf-path

```
$ bcecmd --conf-path ./conf-sample/ bos ls bos:/bce-test/pre/
```

> **注意：** 不使用--conf-path命令时，默认使用用户主目录~/.go-bcecli下的配置。

ObjectAcl管理

描述： Object ACL管理命令允许用户对指定Object或符合指定Prefix条件的Objects执行设置、追加、获取或删除ACL规则。

命令格式：

```
bcecmd bos object-acl <BOS_PATH> --method METHOD [--acl-config-file ACL-CONFIG-FILE] [--canned CANNED] [--grant-read GRANT-READ] [--grant-full-control GRANT-FULL-CONTROL] [--recursive]
```

> 说明：
>
> - <BOS_PATH> 指定BOS路径，格式为：bos:/BUCKET_NAME/OBJECT_KEY。
> - --method 指定Object ACL操作的类型，包括以下4种：
>   - put：设置新的ACL配置，已有的ACL配置将被覆盖。
>   - append：追加新的ACL配置，并保留已有的ACL配置，只支持 --acl-config-file 方式提供的ACL配置。
>   - get：获取指定object或前缀的ACL配置。
>   - delete：删除指定object或前缀的所有标签配置。
> - acl-config-file 指定ACL配置文件，配置文件格式参考 ACL配置文件。
> - canned 设置 CannedACL，当前支持的CannedAcl包括 private（私有）和 public-read（公共读）。
> - grant-read 为指定用户授予指定Object的读权限，传参示例：'user_id_A,user_id_B'。
> - grant-full-control 为指定用户授予指定Object的FULL_CONTROL权限，传参示例：'user_id_A,user_id_B'。
> - --recursive 使用该参数时，命令将为所有匹配<BOS_PATH>前缀的Object设置标签。否则只为指定Object设置标签。**
> - acl-config-file，canned，grant-read和 grant-full-control同时只支持一种提供ACL配置的方式。

设置Object ACL

示例：

- **通过配置文件上传对象ACL配置** 操作示例：

```
$ bcecmd bos object-acl bos:/bucket1/prefix1/object1 --method put --acl-config-file acl-config.json
```

返回示例：

```
  success put object acl: prefix1/object1.
[1] objects of put-ACL on remote.
```

ACL配置文件示例1：采用json array的格式指定一条或多条ACL规则。

```
[
  {
    "grantee": [
      {
        "id": "b13a56d27878788766e4989989212"
      }
    ],
    "permission": [
       "READ"
    ]
  },
  {
    "grantee": [
      {
        "id": "c89d235665666655f478823567981"
      }
    ],
    "permission": [
       "READ"
    ]
  },
  ......
]
```

ACL配置文件示例2：采用一般json格式指定需要追加的ACL规则。

```
{
  "accessControlList": [
    {
      "grantee": [
        {
          "id": "e13b12d2132132132323f492323"
        }
      ],
      "permission": [
        "FULL_CONTROL"
      ]
    },
    {
      "grantee": [
        {
          "id": "e56d577879621c56a577879677765"
        }
      ],
      "permission": [
        "READ"
      ]
    }
  ]
}
```

- **指定用户ID授予指定Object的读权限**

  操作示例：

  ```
  $ bcecmd bos object-acl bos:/bucket1/prefix1/object1 --method put --grant-read 'e56d577879621c56a577879677765,c13b12d2132132132323f492323'
  ```

  返回示例：

  ```
  success put object acl: prefix1/object1.
  [1] objects of put-ACL on remote.
  ```

- **指定用户ID授予指定Object的FULL_CONTROL权限**

  操作示例：

  ```
  $ bcecmd bos object-acl bos:/bucket1/prefix1/object1 --method put --grant-full-control 'e56d577879621c56a577879677765,c13b12d2132132132323f492323'
  ```

  返回示例：

  ```
  success put object acl: prefix1/object1.
  [1] objects of put-ACL on remote.
  ```

- **批量设置ACL配置**

  操作示例：

  ```
  $ bcecmd bos object-acl bos:/bucket1/prefix1 --method put --acl-config-file acl-config.json --recursive
  ```

  返回示例：

  ```
  success put object acl: prefix1/object1.
  success put object acl: prefix1/object2.
  success put object acl: prefix1/object3.
  [3] objects of put-ACL on remote.
  ```

追加Object ACL

示例：

- **通过配置文件上传对象ACL配置** 操作示例：

  ```
  $ bcecmd bos object-ACL bos:/bucket1/prefix1/object1 --method append --acl-config-file acl-config.json
  ```

  返回示例：

  ```
  success append object acl: prefix1/object1.
  [1] objects of append-ACL on remote.
  ```

  ACL配置文件示例与设置Object ACL中的ACL配置文件示例相同。

- **批量设置ACL配置**

  操作示例：

  ```
  $ bcecmd bos object-acl bos:/bucket1/prefix1 --method put --acl-config-file acl-config.json --recursive
  ```

  返回示例：

  ```
  success append object acl: prefix1/object1.
  success append object acl: prefix1/object2.
  success append object acl: prefix1/object3.
  [3] objects of put-ACL on remote.
  ```

获取Object ACl

示例：

- **获取指定Object的ACL配置** 操作示例：

  ```
  $ bcecmd bos object-acl bos:/bucket1/prefix1/object1 --method get
  ```

  返回示例：

```
  prefix1/object1    &{AccessControlList:[{Grantee:[{Id:3a39e5e346bb451287df8a4cf950c1b5}] Permission:[FULL_CONTROL] Resource:[] NotResource:[] Condition:{IpAddress:[] Referer:{StringLike:[]
StringEquals:[]} VpcId:[]} Effect:}]}
[1] objects of get-ACL on remote.
```

- **批量获取对象ACL配置**

  操作示例：

```
  $ bcecmd bos object-acl bos:/bucket1/prefix1 --method get --recursive
```

  返回示例：

```
  prefix1/object1    &{AccessControlList:[{Grantee:[{Id:3a39e5e346bb451287df8a4cf950c1b5}] Permission:[FULL_CONTROL] Resource:[] NotResource:[] Condition:{IpAddress:[] Referer:{StringLike:[]
StringEquals:[]} VpcId:[]} Effect:}]}
prefix1/object2    &{AccessControlList:[{Grantee:[{Id:3a39e5e346bb451287df8a4cf950c1b5}] Permission:[FULL_CONTROL] Resource:[] NotResource:[] Condition:{IpAddress:[] Referer:{StringLike:[]
StringEquals:[]} VpcId:[]} Effect:}]}
prefix1/object3    &{AccessControlList:[{Grantee:[{Id:3a39e5e346bb451287df8a4cf950c1b5}] Permission:[FULL_CONTROL] Resource:[] NotResource:[] Condition:{IpAddress:[] Referer:{StringLike:[]
StringEquals:[]} VpcId:[]} Effect:}]}
[3] objects of get-ACL on remote.
```

删除Object ACL

示例：

- **删除指定object的ACL配置** 操作示例：

```
  $ bcecmd bos object-acl bos:/bucket1/prefix1/object1 --method delete
```

  返回示例：

```
  success delete object acl: prefix1/object1.
[1] objects of delete-ACL on remote.
```

- **批量获取对象标签**

  操作示例：

```
  $ bcecmd bos object-acl bos:/bucket1/prefix1 --method delete --recursive
```

  返回示例：

```
  success delete object acl: prefix1/object1.
success delete object acl: prefix1/object2.
success delete object acl: prefix1/object3.
[3] objects of delete-ACL on remote.
```

ObjectTagging管理

描述： Object Tagging管理命令允许用户对指定Object或指定Prefix的Objects执行设置、追加、获取和删除自定义标签的操作，方便用户在object级别对数据进行分类和管理，标签配置以 `key=value` 形式传递和保存。

命令格式：

```
bcecmd bos object-tagging <BOS_PATH> --method METHOD [--tagging TAGGING] [--keys KEYS] [--recursive]
```

> **说明：**
>
> - `<BOS_PATH>` 指定BOS路径，格式为：bos:/BUCKET_NAME/OBJECT_KEY。
> - `--method` 指定标签操作的类型，包括以下4种：
>   - `put`：用 --tagging 指定的新标签完全覆盖已有的标签。
>   - `append`：追加--tagging指定的新标签，并保留已有的标签。
>   - `get`：获取指定object或前缀的所有标签配置，可以通过--keys指定标签key值获取相应的标签value。
>   - `delete`：删除指定object或前缀的所有标签配置，可以通过--keys删除指定的标签值。
> - `--recursive` 使用该参数时，命令将为所有匹配<BOS_PATH>前缀的Object设置标签。否则只为指定Object设置标签。

设置Object Tagging

示例：

- **通过配置文件上传对象标签** 操作示例：

```
  $ bcecmd bos object-tagging bos:/bucket1/prefix1/object1 --method put --tagging file://tag-config.json
```

  返回示例：

```
  success put object tagging: prefix1/object1.
[1] objects of put-tagging on remote.
```

  标签配置文件示例：

```
{
  "tagInfo": [
    { "key": "key001", "value": "value001"},
    { "key": "key002", "value": "value002"},
    { "key": "key003", "value": "value003"}
  ]
}
```

- **固定格式字符串上传对象标签**

  操作示例：

  ```
  $ bcecmd bos object-tagging bos:/bucket1/prefix1/object1 --method put --tagging 'key01=value01&key02=value02'
  ```

  返回示例：

  ```
  success put object tagging: prefix1/object1.
  [1] objects of put-tagging on remote.
  ```

- **批量上传对象标签**

  操作示例：

  ```
  $ bcecmd bos object-tagging bos:/bucket1/prefix1 --method put --tagging 'key01=value01&key02=value02' --recursive
  ```

  返回示例：

  ```
  success put object tagging: prefix1/object1.
  success put object tagging: prefix1/object2.
  success put object tagging: prefix1/object3.
  [3] objects of put-tagging on remote.
  ```

追加Object Tagging

示例：

- **通过配置文件追加对象标签** 操作示例：

  ```
  $ bcecmd bos object-tagging bos:/bucket1/prefix1/object1 --method append --tagging file://tag-config.json
  ```

  返回示例：

  ```
  success append object tagging: prefix1/object1.
  [1] objects of append-tagging on remote.
  ```

  标签配置文件示例：

  ```
  {
    "tagInfo": [
        { "key": "key001", "value": "value001"},
        { "key": "key002", "value": "value002"},
        { "key": "key003", "value": "value003"}
    ]
  }
  ```

- **固定格式字符串追加对象标签**

  操作示例：

  ```
  $ bcecmd bos object-tagging bos:/bucket1/prefix1/object1 --method append --tagging 'key01=value01&key02=value02'
  ```

  返回示例：

  ```
  success append object tagging: prefix1/object1.
  [1] objects of append-tagging on remote.
  ```

- **批量上传对象标签**

  操作示例：

  ```
  $ bcecmd bos object-tagging bos:/bucket1/prefix1 --method append --tagging 'key01=value01&key02=value02' --recursive
  ```

  返回示例：

  ```
  success append object tagging: prefix1/object1.
  success append object tagging: prefix1/object2.
  success append object tagging: prefix1/object3.
  [3] objects of append-tagging on remote.
  ```

获取Object Tagging

示例：

- **获取指定object的所有标签** 操作示例：

  ```
  $ bcecmd bos object-tagging bos:/bucket1/prefix1/object1 --method get
  ```

  返回示例：

  ```
  prefix1/object1    map[key004:value004 key005:value005 key006:value006 key01:value01 key02:value02]
  [1] objects of get-tagging on remote.
  ```

- **获取object的指定标签**

  操作示例：

  ```
  $ bcecmd bos object-tagging bos:/bucket1/prefix1/object1 --method get --keys 'key01&key02'
  ```

  返回示例：

  ```
  prefix1/object1    map[key01:value01 key02:value02]
  [1] objects of append-tagging on remote.
  ```

- **批量获取对象标签**

操作示例：

```
$ bcecmd bos object-tagging bos:/bucket1/prefix1 --method get --recursive
```

返回示例：

```
 prefix1/object1    map[key01:value01 key02:value02]
prefix1/object2    map[key01:value01 key02:value02]
prefix1/object3    map[key01:value01 key02:value02]
[3] objects of append-tagging on remote.
```

删除Object Tagging

示例：

- **删除指定object的所有标签** 操作示例：

```
$ bcecmd bos object-tagging bos:/bucket1/prefix1/object1 --method delete
```

返回示例：

```
 success delete object tagging: prefix1/object1.
[1] objects of delete-tagging on remote.
```

- **获取object的指定标签**

操作示例：

```
$ bcecmd bos object-tagging bos:/bucket1/prefix1/object1 --method delete --keys 'key01&key02'
```

返回示例：

```
 success delete object tagging: prefix1/object1.
[1] objects of delete-tagging on remote.
```

- **批量获取对象标签**

操作示例：

```
$ bcecmd bos object-tagging bos:/bucket1/prefix1 --method delete --recursive
```

返回示例：

```
 success delete object tagging: prefix1/object1.
success delete object tagging: prefix1/object2.
success delete object tagging: prefix1/object3.
[3] objects of delete-tagging on remote.
```

BOS API 级命令

Bucket ACL管理

设置 Bucket ACL

描述： 配置Bucket ACL有两种方法: 1. 使用CannedAcl； 2. 通过上传ACL配置文件设置Bucket ACL。

命令格式：

```
bcecmd bosapi put-bucket-acl --bucket-name BUCKET-NAME [--acl-config-file ACL-CONFIG-FILE] [--canned CANNED]
```

> **说明：**
>
> - 使用`--acl-config-file` 指定ACL配置文件，ACL配置文件格式请参考 Bucket权限控制；
> - 使用 `--canned` 设置 CannedACL，当前支持的CannedAcl包括 private（私有）、public-read（公共读）、public-read-write（公共读写）；
> - `--acl-config-file` 和`--canned` 不能同时使用；
> - 配置CannedACL时，如果配置失败则会打印错误信息，如果配置成功则不会输出任何消息；
> - 通过上传ACL配置文件设置Bucket ACL时，如果设置成功会打印ACL配置信息，失败则会打印ACL配置文件的内容和错误信息。

示例：

- **使用CannedAcl方式的权限控制**

操作示例：

```
$ bcecmd bosapi put-bucket-acl --bucket-name bucket1 --canned private
```

返回示例： 无

- **通过上传ACL配置文件设置Bucket ACL**

操作示例：

```
$ bcecmd bosapi put-bucket-acl --bucket-name bucket1 --acl-config-file acl-config.json
```

返回示例：

```
{
  "accessControlList": [
    {
      "grantee": [
        {
          "id": "e13b12d2132132132323f492323"
        }
      ],
      "permission": [
          "READ"
      ]
    }
  ]
}
```

追加 Bucket ACL

描述： 追加Bucket ACL的功能是：在指定Bucket原有的ACL配置的基础上，追加新的ACL规则，ACL规则同样由ACL配置文件指定。

命令格式：

```
bcecmd bosapi append-bucket-acl --bucket-name BUCKET-NAME [--acl-config-file ACL-CONFIG-FILE]
```

说明：

- 使用--acl-config-file 指定ACL配置文件，ACL配置文件格式请参考 Bucket权限控制；
- 如果ACL配置文件中指定的ACL规则与已有ACL规则相同，则会自动过滤掉，并输出提示信息。
- 通过上传ACL配置文件追加Bucket ACL时，如果设置成功会打印ACL配置信息，失败则会打印ACL配置文件的内容和错误信息。

示例：

- **通过上传ACL配置文件追加Bucket ACL**

  操作示例：

  ```
  $ bcecmd bosapi append-bucket-acl --bucket-name bucket1 --acl-config-file acl-config.json
  ```

  返回示例：

  ```
  {
    "accessControlList": [
      {
        "grantee": [
          {
            "id": "e13b12d2132132132323f492323"
          }
        ],
        "permission": [
            "READ"
        ]
      },
      {
        "grantee": [
          {
            "id": "b13a56d27878788766e4989989212"
          }
        ],
        "permission": [
            "READ"
        ]
      }
    ]
  }
  ```

- **ACL配置文件示例**

  ACL配置文件中可包含单条或多条ACL规则，示例如下：

  示例1:

  采用json array的格式指定一条或多条ACL规则。

```
[
  {
    "grantee": [
      {
        "id": "b13a56d27878788766e4989989212"
      }
    ],
    "permission": [
      "READ"
    ]
  },
  {
    "grantee": [
      {
        "id": "c89d2356656666655f478823567981"
      }
    ],
    "permission": [
      "READ"
    ]
  },
  ......
]
```

示例2:

采用一般json格式指定需要追加的ACL规则。

```
{
  "accessControlList": [
    {
      "grantee": [
        {
          "id": "e13b12d2132132132323f492323"
        }
      ],
      "permission": [
        "FULL_CONTROL"
      ]
    },
    {
      "grantee": [
        {
          "id": "e56d577879621c56a577879677765"
        }
      ],
      "permission": [
        "READ"
      ]
    }
  ]
}
```

获取Bucket ACL

描述： 获取Bucket 的ACL配置。

命令格式：

```
bcecmd bosapi get-bucket-acl --bucket-name BUCKET-NAME
```

操作示例：

```
bcecmd bosapi get-bucket-acl --bucket-name bucket1
```

返回示例：

```
{
 "accessControlList": [
  {
    "grantee": [
      {
        "id": "e13b12d2132132132323f492323"
      }
    ],
    "permission": [
      "FULL_CONTROL"
    ],
    "condition": {
      "ipAddress": null,
      "referer": {
        "stringLike": null,
        "stringEquals": null
      }
    }
  }
 ],
 "owner": {
   "id": "e13b12d2132132132323f492323"
 }
}
```

Bucket Tagging管理

描述： Bucket Tagging管理命令允许用户对指定bucket执行设置、追加、获取和删除自定义标签的操作。bucket标签方便用户在bucket级别对数据进行分类和管理，标签配置以 `key=value` 形式传递和保存。

命令格式：

```
bcecmd bosapi bucket-tagging --bucket-name BUCKET-NAME --method METHOD [--tagging TAGGING] [--keys KEYS]
```

> **说明：**
>
> - `--bucket-name` 指定需要进行标签操作的bucket名称
> - `--method` 指定标签操作的类型，包括以下4种：
>   - `put`：用 `--tagging` 指定的新标签完全覆盖已有的标签。
>   - `append`：追加`--tagging`指定的新标签，并保留已有的标签。
>   - `get`：获取指定bucket的所有标签配置，可以通过`--keys`指定标签key值获取相应的标签value。
>   - `delete`：删除指定bucket的所有标签配置，可以通过`--keys`删除指定的标签值。
> - 使用`--tagging` 指定bucket标签配置，用于put和append两种method，支持两种传参方式：
>   - 参数值前缀为 `file://` 时，表示从文件中读取配置，配置文件必须为json格式。
>   - 固定格式字符串指定标签配置，如：`'key001=value001&key002=value002'`。
> - `--keys`指定想要获取或删除的标签key值，用于get和delete两种method，如果不使用该`flag`，获取或删除对象将是指定bucket的全部标签。传参格式：`'key001&key002&key003'`。

设置 Bucket Tagging

示例：

- **使用配置文件上传bucket标签配置**

  操作示例：

  ```
  $ bcecmd bosapi bucket-tagging --bucket-name bucket1 --method put --tagging file://tag-config.json
  ```

  返回示例： 无

  标签配置文件示例：

  ```
  {
    "tags":[
      {
        "tagKey":"key-222",
        "tagValue":"value-xxxx"
      },
      {
         "tagKey":"key-333",
        "tagValue":"value-333"
      }
    ]
  }
  ```

- **通过固定格式字符串指定bucket标签配置**

  操作示例：

  ```
  $ bcecmd bosapi bucket-tagging --bucket-name bucket1 --method put --tagging 'key001=value001&key3=value3'
  ```

  返回示例： 无

追加 Bucket Tagging

示例：

- **使用配置文件上传bucket标签配置**

  操作示例：

  ```
  $ bcecmd bosapi bucket-tagging --bucket-name bucket1 --method append --tagging file://tag-config.json
  ```

  返回示例： 无

  标签配置文件示例：

  ```
  {
    "tags":[
      {
        "tagKey":"key-222",
        "tagValue":"value-xxxx"
      },
      {
         "tagKey":"key-333",
        "tagValue":"value-333"
      }
    ]
  }
  ```

- **通过固定格式字符串指定bucket标签配置**

  操作示例：

  ```
  $ bcecmd bosapi bucket-tagging --bucket-name bucket1 --method append --tagging 'key001=value001&key3=value3'
  ```

  返回示例： 无

获取Bucket Tagging

示例：

- **获取指定bucket的全部标签**

  操作示例：

  ```
  $ bcecmd bosapi bucket-tagging --bucket-name bucket1 --method get
  ```

  返回示例：

  ```
  {
    "tag": [
      {
        "tag_key": "ttt-222",
        "tag_value": "0898dsfijidf"
      },
      {
        "tag_key": "key-333",
        "tag_value": "value-333"
      },
      {
        "tag_key": "key-222",
        "tag_value": "value-xxxx"
      }
    ]
  }
  ```

- **获取指定标签key的值**

  操作示例：

  ```
  $ bcecmd bosapi bucket-tagging --bucket-name bucket1 --method get --keys 'ttt-222&key-333'
  ```

  返回示例：

  ```
  {
    "tag": [
      {
        "tag_key": "ttt-222",
        "tag_value": "0898dsfijidf"
      },
      {
        "tag_key": "key-333",
        "tag_value": "value-333"
      }
    ]
  }
  ```

删除Bucket Tagging

示例：

- **删除指定bucket的全部标签**

  操作示例：

  ```
  $ bcecmd bosapi bucket-tagging --bucket-name bucket1 --method delete
  ```

  返回示例： 无

- **删除指定标签key的值**

  操作示例：

  ```
  $ bcecmd bosapi bucket-tagging --bucket-name bucket1 --method get --keys 'ttt-222&key-333'
  ```

  返回示例： 无

FetchObject

描述： 此命令用于从指定URL抓取资源，并将资源存储到指定的Bucket中。此操作需要请求者对该Bucket有写权限，每次只能抓取一个Object，Object的名称可自定义。

命令格式：

```
bcecmd bosapi fetch-object --bucket-name BUCKET-NAME --object-name OBJECT-NAME --source SOURCE [--mode MODE] [--storage-class STORAGE-CLASS] [--callback-address CALLBACK-ADDRESS]
```

> **说明：**
>
> - `--bucket-name` 指定抓取资源将要存储的bucket名称。
> - `--object-name` 指定抓取资源存储到bucket中的object名称。
> - `--source` 指定抓取资源的URL。
> - `--mode` 指定抓取模式，支持 `sync` 和 `async`两种模式，默认为 `sync` 模式。
> - `--storage-class` 指定object的存储类型，可选值：`STANDARD`，`STANDARD_IA`，`COLD` 和 `ARCHIVE`，默认为 `STANDARD`。
> - `--callback-address` 当指定抓取模式为 `async` 时，可指定一个回调地址接收抓取的结果。

示例：

- **抓取指定URL资源存储到指定bucket**

  操作示例：

  ```
  $ bcecmd bosapi fetch-object --bucket-name bucket1 --object-name object1 --source http://www.abc.com/demo.html
  ```

  返回示例：

```
{"code":"success","message":"success","requestId":"90a38877-42bc-40ce-b84b-e05ec70ad6ea","jobId":""}
```

- **异步模式抓取指定URL资源**

  操作示例：

  ```
  $ bcecmd bosapi fetch-object --bucket-name bucket1 --object-name object1 --source http://www.abc.com/demo.html --mode async
  ```

  返回示例：

  ```
  {"code":"success","message":"success","requestId":"db5defbe-0bc0-4824-a273-1419289af7d6","jobId":"fo-cbb19a08a82bcf3f06cb68c174991247"}
  ```

Bucket Lifecycle管理

设置生命周期

描述： 按照提供的生命周期管理文件为指定的Bucket设置生命周期管理规则，或生成生命管理文件模板。

命令格式：

- 生成Transition的配置模板：`bcecmd bosapi put-lifecycle --template`
- 按照指定的配置文件为Bucket设置生命周期管理规则： `bcecmd bosapi put-lifecycle --lifecycle-config-file LIFECYCLE_CONFIG_FILE --bucket-name BUCKET_NAME`

示例：

- **生成Transition的配置模板**

  操作示例：

  ```
  $ bcecmd bosapi put-lifecycle --template
  ```

  返回示例：

  ```
  {
   "rule": [
    {
     "action": {
      "name": "Transition",
      "storageClass": "STANDARD_IA"
     },
     "status": "enabled",
     "resource": [
      "${bucket_name}/${prefix}/*"
     ],
     "id": "sample-id",
     "condition": {
      "time:": {
       "dateGreaterThan": "$(lastModified)+P30D"
      }
     }
    }
   ]
  }
  ```

- **按照配置文件1.txt为Bucket1设置生命周期管理规则**

  操作示例：

  ```
  bcecmd bosapi put-lifecycle --lifecycle-config-file lifecycle_bj.json --bucket-name bucket1
  ```

  返回示例：

  ```
  {
   "rule": [
    {
     "status": "enabled",
     "action": {
      "name": "Transition",
      "storageClass": "STANDARD_IA"
     },
     "resource": [
      "bucket1/*"
     ],
     "id": "sample-rule-transition",
     "condition": {
      "time": {
       "dateGreaterThan": "$(lastModified)+P180D"
      }
     }
    }
   ]
  }
  ```

> **说明：**设置成功后会返回生命周期配置文件，否则报错。

获取生命周期配置

描述： 获取某个指定Bucket的生命周期配置。

命令格式： `bcecmd bosapi get-lifecycle --bucket-name BUCKET_NAME`

操作示例： `bcecmd bosapi get-lifecycle --bucket-name bucket1`

返回示例：

```
{
 "rule": [
  {
    "action": {
      "name": "Transition",
      "storageClass": "STANDARD_IA"
    },
    "status": "enabled",
    "resource": [
      "bucket1/prefix1/*"
    ],
    "id": "sample-id",
    "condition": {
      "time:": {
        "dateGreaterThan": "$(lastModified)+P30D"
      }
    }
  }
 ]
}
```

删除生命周期配置

描述： 删除指定Bucket的生命周期管理配置。

命令格式： `bcecmd bosapi delete-lifecycle --bucket-name BUCKET_NAME`

操作示例： `bcecmd bosapi delete-lifecycle --bucket-name bucket1`

返回示例： 无

Bucket Logging管理

设置日志规则

描述： 为Bucket开启访问日志功能，并保存到指定Bucket的prefix中。

命令格式： `bcecmd bosapi put-logging --target-bucket TARGET_BUCKET --target-prefix TARGET_PREFIX --bucket-name BUCKET_NAME`

操作示例： `bcecmd bosapi put-logging --target-bucket bucket2 --target-prefix log --bucket-name bucket1`

返回示例： 无

获取日志规则

描述： 获取指定Bucket的访问日志配置情况。

命令格式： `bcecmd bosapi get-logging --bucket-name BUCKET_NAME`

操作示例： `bcecmd bosapi get-logging --bucket-name bucket1`

返回示例：

```
 $ bcecmd bosapi get-logging --bucket-name bucket1
{
  "status": "enabled",
  "targetPrefix": "loggin/",
  "targetBucket": "bucket1"
}
```

删除日志规则

描述： 删除指定Bucket的访问日志设置。

命令格式： `bcecmd bosapi delete-logging --bucket-name BUCKET_NAME`

操作示例： `bcecmd bosapi delete-logging --bucket-name b1`

返回示例： 无

GetObjectMeta

描述： `get-object-meta`命令用于获取指定Bucket中的具体Object的meta信息。

命令格式：

- $ bcecmd bosapi get-object-meta --bucket-name BUCKET-NAME --object-name OBJECT-NAME

> **说明：**
>
> - --bucket-name: bucket名称
> - --object-name：要查询的object名称

示例：

操作示例：

```
$ bcecmd bosapi get-object-meta --bucket-name=100kperf-xitaotao --object-name=water
```

返回示例：

```
2019 / 10 / 10 11: 06: 49 {
 "CacheControl": "",
 "ContentDisposition": "",
 "ContentEncoding": "",
 "ContentLength": 0,
 "ContentRange": "",
 "ContentType": "application/json",
 "ContentMD5": "1B2M2Y8AsgTpgAmY7PhCfg==",
 "ContentSha256": "",
 "Expires": "Sun, 13 Oct 2019 03:06:49 GMT",
 "LastModified": "Thu, 26 Sep 2019 08:55:32 GMT",
 "ETag": "d41d8cd98f00b204e9800998ecf8427e",
 "UserMeta": null,
 "StorageClass": "COLD",
 "NextAppendOffset": "",
 "ObjectType": "",
 "Crc32": "0"
}
```

Bucket Versioning管理

Bucket Versioning管理

描述： Bucket Versioning管理命令允许用户设置bucket版本控制状态。

> **注意事项：**
>
> - 要配置版本控制，您必须有PutBucketVersioning权限。
> - Bucket包括未开启(notEnabled)、开启（enabled）或者暂停（suspended）三种版本控制状态。默认情况下Bucket处于未开启版本控制状态。
> - 在Bucket处于开启版本控制状态下，所有新添加的文件（Object）都将拥有唯一的版本ID，将累积所添加Object的多个版本。
> - 在Bucket处于暂停版本控制状态下，所有新添加Object的版本ID将为null，将不再为此状态下添加的Object累积更多的版本。

设置 Bucket Versioning 状态

描述： `put-bucket-versioning` 命令允许用户设置bucket版本控制状态，包括`enabled`和`suspended`。

命令格式：

```
bcecmd bosapi put-bucket-versioning --bucket-name BUCKET-NAME --status STATUS
```

> **说明：**
>
> - `--bucket-name` 指定需要设置versioning状态的bucket名称。
> - `--status` 指定versioning的状态值，可选值：`enabled`和`suspended`。

示例：

- **开启bucket的versioning功能**

操作示例：

```
$ bcecmd bosapi put-bucket-versioning --bucket-name bucket1 --status enabled
```

返回示例： 无

- **暂停bucket的versioning功能**

操作示例：

```
$ bcecmd bosapi put-bucket-versioning --bucket-name bucket1 --status suspended
```

返回示例： 无

获取 Bucket Versioning 状态

描述： `get-bucket-versioning` 管理命令允许用户获取bucket版本控制状态。

命令格式：

```
bcecmd bosapi get-bucket-versioning --bucket-name BUCKET-NAME
```

> **说明：**
>
> - `--bucket-name` 指定需要获取versioning状态的bucket名称。

示例：

操作示例：

```
$ bcecmd bosapi put-bucket-versioning --bucket-name bucket1
```

开启状态返回示例：

```
 {
   "status": "enabled"
}
```

暂停状态返回示例：

```
 {
   "status": "suspended"
}
```

BOSCMD版本变更记录

- BCE CMD工具包 [2025-07-09] 版本号0.5.6
  - 修复已知问题
- BCE CMD工具包 [2025-06-30] 版本号0.5.5 变更记录：
  - 支持bucket versioning开启和暂停
  - 支持bucket tagging
  - 支持object tagging
  - 支持fetch object
  - 支持object acl
  - bugfix: --quiet和--disable-bar没有取消进度条打印
- BCE CMD工具包 [2025-06-17] 版本号0.5.4 变更记录：
  - sync 命令增加强制覆盖模式
  - 支持多任务进度查看功能
  - 分片上传下载错误日志完善
- BCE CMD工具包 [2025-06-09] 版本号0.5.3 变更记录：
  - 追加bucekt acl规则方式调整: 增加append-bucket-acl命令
- BCE CMD工具包 [2025-06-05] 版本号0.5.2 变更记录：
  - 支持配置use path style和追加bucket acl规则
- BCE CMD工具包 [2024-05-21] 版本号0.4.6 变更记录：
  - 支持命令行自动补全功能
- BCE CMD工具包 [2024-03-22] 版本号0.4.5 变更记录：
  - sync时间过滤bug修复
  - 修改大文件上传下载时的默认分片大小
  - 支持大文件下载并发度配置
- BCE CMD工具包 [2023-12-04] 版本号0.4.3 变更记录：
  - 支持rand sync
  - 上传或copy文件时支持设置文件meta信息
  - 支持arm架构
  - 修复sync路径Bug
- BCE CMD工具包 [2022-04-29] 版本号0.3.3 变更记录：
  - 修复Bug。
- BCE CMD工具包 [2019-11-05] 版本号0.3.0 变更记录：
  - 修复Bug。
- BCE CMD工具包 [2019-10-08] 版本号0.2.9 变更记录：
  - 支持STS
  - 支持get-object-meta
- BCE CMD工具包 [2019-09-05] 版本号0.2.8 变更记录：
  - 修复Bug。
- BCE CMD工具包 [2019-08-22] 版本号0.2.7 变更记录：
  - 修复Bug。
- BCE CMD工具包 [2019-06-25] 版本号0.2.6 变更记录：
  - 修复Bug。
- BCE CMD工具包 [2019-05-14] 版本号0.2.5 变更记录：
  - 修改bcecmd返回值：上传、下载和 sync时，部分文件出错bcecmd也以1退出。
  - 修复Bug。
- BCE CMD工具包 [2019-03-19] 版本号0.2.4

  变更记录：
  - 修复Bug和性能优化。

- 为 bcecmd sync 添加三种同步模式(time-size, time-size-crc32 和 only-crc32）。
- 为 bcecmd bos sync 添加 `--exclude-delete` 和 `--download-tmp-path` 选项。

- BCE CMD工具包 [2018-11-01] 版本号0.2.3

  变更记录：

  - 支持三步COPY。
  - 支持以断点续传方式上传、下载和拷贝大文件。
  - 为上传、下载和拷贝大文件添加进度条。
  - 支持bos前缀 `bos://`。

- BCE CMD工具包 [2018-05-29] 版本号0.2.2

  变更记录：

  - 支持用户手动配置分块上传的分块大小。
  - 优化CMD上传下载性能。
  - 修复Bug。

- BCE CMD工具包 [2018-04-04] 版本号0.2.1

  变更记录：

  - 优化CMD性能。
  - 修复Bug。

- BCE CMD工具包 [2018-02-01] 版本号0.2.0

  变更记录：

  - 支持BOSAPI接口，包含Bucket ACL、生命周期、日志和默认存储类型等管理接口。
  - 为sync 添加 exclude 和 include 过滤。
  - 修复Bug。

- BCE CMD工具包 [2017-12-27] 版本号0.1

  变更记录：

  - Bucket创建和删除，Object的上传、下载、删除和拷贝等功能。
  - 支持sync操作。
  - 支持根据操作的Bucket自动切换域名。

配置BCECMD命令行补全功能

功能介绍

bcecmd目前在0.4.6版本新支持了命令行补全功能，可以帮助用户节省打字时间以及提示各种式样的Flag和子命令。当前版本支持四种命令行的自动补全功能：bash, zsh, fish 以及PowerShell。以下将介绍该如何配置各种终端该如何配置。（在开始配置前请确保已经将bcecmd的路径加入系统路径，以下命令皆以此为前提）

Bash

bcecmd可以使用命令 `bcecmd completion bash` 输出bash补全脚本，source该脚本可激活命令行补全功能，但在此之前，需要进行一些前置bash配置。

- 请先确保你的bash版本在4.1及以上
- 如果是Mac用户，Mac至多支持到3.27版本的Bash,需使用补全请使用zsh
- 如果是Linux用户请使用命令 `apt-get install bash-completion`或者`yum install bash-completion` 安装bash-completion

做完以上操作，系统应自动完成创建上述命令将创建文件 `/usr/share/bash-completion/bash_completion`, 之后请打开~/.bashrc文件，加入以下命令并且重启命令行:

- `source /usr/share/bash-completion/bash_completion`
- `source <(bcecmd completion bash)`

Zsh bcecmd可以使用命令 `bcecmd completion bash` 输出bash补全脚本, 在/.zshrc文件中保存下列指令并重启终端即可使用命令行补全功能：

- `source <(bcecmd completion zsh)`

如果遇到`command not found: compdef`报错，则在./zshrc文件头部添加如下命令：

- `autoload -U compinit; compinit`

Fish 启用Fish补全，可以对~/.config/fish/config.fish文件修改达成永久补全功能激活，在该文件中输入以下指令并且重启命令行：

- `bcecmd completion fish | source`

PowerShell PowerShell的bcecmd命令行补全功能较为残缺，无法补全所有的Flag选项，仅可以提示所有的命令和子命令，但PowerShell能识别输入flag格式的正确与否。 要启用PowerShell的命令行补全功能，需在命令行输入以下语句并重启命令行：

- `bcecmd completion powershell >> $PROFILE`

BOS CMD常见问题

本文介绍在使用 CMD 工具时可能出现的问题及处理方法。

常见报错

错误码：

BOS-FTP工具

简介

BOSFTP是一个基于Python实现的BOS的FTP server工具，将FTP协议的操作转化为对BOS中存储的对象、目录的操作。支持文件上传、下载、删除以及创建文件夹等操作。

主要的功能

BOSFTP支持的主要功能如下：

- 文件流式上传，大文件分块并行上传
- 文件流式下载
- 支持大部分FTP命令，满足FTP需求
- 服务配置化，满足性能要求

系统环境

- 操作系统：LINUX
- Python版本：Python 2
- 依赖：pyftpdlib>=1.5.0 bce-python-sdk>=0.8.23

安装

1. 使用pip安装pyftpdlib

```
pip install pyftpdlib
```

2. 安装python sdk及其依赖。

```
pip install pycryptodome
pip install future
pip install futures

pip install bce-python-sdk
```

3. 下载BOSFTP, 并解压。

4. 验证是否安装好依赖

```
cd ..
python setup.py install
```

启动FTP server

执行命令启动一个FTP server， 默认监听端口为2121。

```
bash start.sh
> running ftp server at port 2121...
```

关闭

-执行命令ctrl + c关闭FTP Server服务。 -可以指定在后台运行sh start.sh&将服务启动，运行sh stop.sh 关闭服务。

配置文件

```
[BOS_ACCOUNT]
# 设置ak, sk, bucket
access_key_id = <your access_key_id>
secret_access_key = <your secret_access_key>
bucket = <BucketName>
# FTP Server 的工作目录。
bos_user_home_dir = /home/users/xxx

[FTP_ACCOUNT]
# FTP 账户配置。配置格式为<用户名:密码:读写权限>，多个账户用分号分割。
login_users = usr1:psw1:RW;usr2:psw2:R

[NETWORK]
# 设置endpoint,默认为http://bj.bcebos.com
#endpoint = http://xx.bcebos.com
# FTP监听端口号,默认为2121
listen_port = 8081
# HTTP连接超时时间，默认为3600
connection_timeout_in_mills = 10000
# 并行上传最大并行数，即线程池线程数， 默认为cpu_count * 4
parallel_upload_num = 9
# FTP server最大连接数，默认为1024
max_connection_num = 1000
# 三步上传将文件切割成每个multi_part_size后进行并行上传， 默认为10485760（10M）
multi_part_size = 10485760
masquerade_address = 127.0.0.1
# passive_port可以设置passive模式下，端口的选择范围，默认在(60000, 65535)区间上选择
passive_port = 60000,65535

[LOG]
# 设置日志级别
log_level = INFO
# 设置日志目录, 日志名为bosftp.log
log_dir = ./log/
```

配置文件说明

- bos_user_home_dir: 本地文件夹路径，上传或是下载都在这个目录进行
- login_users: 为用户账户信息，启动服务时会验证用户名和密码是否存在于login_users1当中，例如: open 127.0.0.1 2121连接到server，提示输入用户名和密码，这里输入的信息必须存在login_users当中才会通过权限认证

- parallel_upload_num: 大文件上传会将文件分块（默认10M），然后并发的将这些文件块上传，这个参数用来控制并发度
- masquerade_address: 使用masquerade_address选项来唯一绑定一个passive模式下用于reply的IP
- log_dir: BOSFTP的运行日志文件所在目录。

测试

```
 sh start.sh& "启动后台服务
cd test
sh run_test.sh
sh stop.sh
```

> 注意：配置AK/SK时，Bucket要实际存在，才会再后台启动服务。

常见问题

- 创建目录删除后导致父级目录被删除怎么办？

例如，执行如下操作：

```
 mkdir dir1/dir2
cd dir1
rm dir2
cd ..
```

由于创建目录的实现方式是创建名为dir1/dir2的空Object，所以上述操作会导致dir1也被删除。建议创建目录时使用如下方式：

```
 mkdir dir1
cd dir1
mkdir dir2
```

- 执行 `size file_name`时， 显示'550 SIZE not allowed in ASCII mode'

执行`type binary`命令修改传输方式。

- 当`connection_timeout_in_mills`配置参数比较小的时候会导致rename一个大的文件对象超时失败

建议`connection_timeout_in_mills`配置参数值为10000，即10s。

- put一个大小为0的object显示成功，但是执行list操作缺显示不出。

pyftpdlib库本身不会去读取0大小的文件，所以无法上传0大小的文件。

- 启动ftp服务后，用其他机器连接，显示Connection refused怎么办？

显示如下

```
 227 Entering passive mode (127,0,0,1,97,183).
ftp: connect: Connection refused
```

执行`passive`命令即可。

版本记录

- BOSFTP工具包 [2024-01-24] 版本号1.3.1
  - 兼容支持pyftpdlib v1.5.9+的版本
- BOSFTP工具包 [2018-03-08] 版本号1.2.0
  - 修复安装时的python sdk版本问题，优化日志目录配置
- BOSFTP工具包 [2018-03-08] 版本号1.0.1
  - 支持APPE接口实现断点续传功能
- BOSFTP工具包 [2018-11-09] 版本号1.0.0
  - 支持文件上传，下载，删除， 支持文件夹创建
  - 支持put、get、delete、mkdir、ls、cd、quite、size、rename命令
  - 自定义ftp配置
  - 大文件分块上传

BOS-Import数据上云工具

概述

BosImport工具用于将其它云存储的数据迁移到BOS。 工具基于C-S结构实现，client端负责读取文件列表, 任务调度和进度统计等工作, 所有和迁移相关的参数/配置都在client端进行；迁移任务在server端完成, client端通过RPC请求通知server对哪些文件进行迁移。

BosImport工具具有以下特点：

- 支持多种迁移方式, 可扩展性良好, 支持迁移不同的源端目的端，目前支持AWS S3、阿里云OSS、腾讯云COS、七牛、小米生态云、BOS、金山云KS3等；
- 支持实时迁移并发度/压力调节，server"热升级"；
- 支持迁移任务热启停，迁移进度实时显示；
- 数据传输过程不落磁盘, 减少磁盘IO, 传输过程中将大文件切块传输, 块大小可以实时配置, 最大效率地利用迁移机器的带宽资源；
- 负载均衡, client会选择没有压力的server进行迁移任务；
- 支持自定义迁移的任务列表和文件数目；

- 支持迁移任务QPS和流量限速；
- 支持多种迁移策略: 1. FORCE_OVERWRITE: 强制覆盖目的端文件, 可能会破坏目的端文件, 危险, 慎用, 在确定需求的前提下使用； 2. KEEP_DESTITNATION: 如果目的端有同名文件文件存在则跳过； 3. KEEP_NEWER: 如果目的端有同名文件存在, 且新于源端文件, 则跳过, 否则覆盖之; 如果目的端没有"同名"文件(有可能迁移时文件需要改名)存在, 则按照迁移规则直接复制到目的端

下载地址

错误码（code）	解决方式
Receive error: [Code: AccessDenied; Message: Access Denied.; RequestId: xxxxxxxxxxxxx]]	在使用CMD工具时通过主账户使用，同时配置fullcontrol权限

文件名及下载地址	MD5
bosimport-1.0.2.tar	cc3fa596f917383278463aba1ebc3e65

运行环境

Java 1.7及以上

目录结构

```
├── client
│   ├── client.jar   # client jar，负责读取文件列表, 任务调度和进度统计等工作
│   ├── resources    # client配置文件夹
│   │   ├── client.properties   # client迁移任务配置文件
│   │   └── client_logback.xml   # client日志配置文件
│   ├── start_client.sh   # 启动client进程
│   └── stop_client.sh   # 停止client进程
├── properties_examples   # 迁移任务client.properties不同源示例配置
│   ├── client.properties.bos   # BOS迁移到BOS
│   ├── client.properties.bos-check   # BOS到BOS校验
│   ├── client.properties.cos   # COS迁移到BOS
│   ├── client.properties.fds   # 小米生态云迁移到BOS
│   ├── client.properties.http   # HTTP迁移到BOS
│   ├── client.properties.oss   # OOS迁移到BOS
│   ├── client.properties.qiniu   # 七牛迁移到BOS
│   ├── client.properties.ks3   # 金山云ks3迁移到BOS
│   └── client.properties.s3   # AWS S3迁移到BOS
├── server
│   ├── resources   # server配置文件夹
│   │   ├── server.properties   # server配置文件
│   │   └── server_logback.xml   # server日志配置文件
│   ├── server.jar   # server.jar，负责读取数据源并迁移到BOS
│   ├── supervise.sh   # supervisor脚本，保持server进程退出自动重启
│   ├── start_server.sh   # 启动server进程
│   └── stop_server.sh   # 停止server进程
├── start_migration.sh   # 一键启动迁移任务，包括server和client
└── stop_migration.sh   # 一键停止迁移任务，包括server和client
```

部署方式

BosImport工具主要包括server和client两部分。

- 对于小于30TB的小规模数据迁移，client和server可以运行在同一个机器上，修改client.properties文件，然后直接sh start_migration.sh启动迁移即可。
- 对于大规模的数据迁移，或者需要提高迁移速度，可以部署多个server到多台机器上，client通过serverList配置多个server，并根据负载选择一个合适的server进行迁移，从而提高迁移效率。

注意事项

1. 单机可以开多个server进程，但是需要注意server/logs文件夹占用空间大小；
2. server进程线程数需要根据机器CPU核数调整到合适数目，maxConnectionsPerServer建议设置为1，否则fail出现次数容易比较多。

部署server

1. 修改server的配置文件 server.properties, 主要修改监听端口号；
2. 如果需要多个server并发迁移，可以批量复制server文件夹到用于迁移的多台机器上；
3. 执行sh start_server.sh启动server，通过检查server进程是否监听了指定端口来判断是否已经成功启动server。

部署client

修改client.properties文件对client进行配置, 主要配置项包括

1. 迁移类型：type，迁移名称：migrationName
2. server响应单次请求所使用的并发数:serverThreadNum
3. 单次请求文件数量: filesPerRequest
4. 任务列表: taskList
5. 文件列表：listDelimiter， 指定文件列表的表示形式: listFormat
6. 源端目的端ak, sk, endpoint等信息
7. 迁移策略：migrationStrategy
8. 切块大小: blockSize, 默认5MB，因为BOS分块上传时Part数目范围是1-10000，blockSize=5MB时可以上传的单个Object最大为50GB，迁移大文件时需要调大该参数，Part数目在1000以内较佳。

启动迁移

server和client单机部署时，执行sh start_migration.sh脚本启动server和client，系统根据client.properties文件中配置的文件列表迁移数据；如果server部署到了多台机器上，需要在每台机器上分别执行sh start_server.sh启动server进程。

停止迁移

执行sh stop_migration.sh脚本停止server和client。如果server部署到了多台机器上，需要在每台机器上分别执行sh stop_server.sh停止server进程。

配置文件

server.properties

server.properties文件用于配置端口、流量限制以及RPC服务线程数，在没有特殊要求的情况下，一般只需要修改监听端口号即可，默认监听8090端口。

```
 ##### server进程监听的端口号
serverPort=8090

##### 每个client请求对应的处理线程数目
fileWorkerNum=50
```

client.properties

client.properties文件（**必须是UTF8编码的**）：该文件是**迁移任务的核心配置文件**，在项目的client/resources目录下，通过修改该文件对client和server进行调节。

文件主要包含迁移任务的列表信息和并发度等，配置文件中有些配置被修改后会自动重新加载，如serverThreadNum(单次迁移请求期望server使用的并发线程数量)；有些是不能重新加载的，如taskList(迁移任务列表)，具体配置可以参考文件内的注释说明。

以下为不同类型迁移任务的配置文件示例，可以参考下表中示例修改client.properties文件

配置示例

迁移类型	示例文件	说明
从http迁移到BOS	properties_examples/client.properties.http	给定url(每个URL代表一个可以直接使用GET请求下载的文件)列表, 迁移到BOS
从OSS迁移到BOS	properties_examples/client.properties.oss	给定OSS的Object列表, 配置源端和目的端ak/sk等信息
从COS迁移到BOS	properties_examples/client.properties.cos	给定COS的Object列表, 配置源端和目的端ak/sk等信息；srcEndpoint参数需要填cos region
从七牛迁移到BOS	properties_examples/client.properties.qiniu	给定七牛的Object列表, 配置源端和目的端ak/sk等信息
从AWS S3迁移到BOS	properties_examples/client.properties.s3	给定AWS S3的Object列表, 配置源端和目的端ak/sk等信息
从FDS迁移到BOS	properties_examples/client.properties.fds	给定源小米生态云的Object列表, 配置源端和目的端ak/sk 等信息
从KS3迁移到BOS	properties_examples/client.properties.ks3	给定源金山云ks3的Object列表, 配置源端和目的端ak/sk等信息
从BOS迁移到BOS	properties_examples/client.properties.bos	给定源BOS的Object列表, 配置源端和目的端ak/sk等信息
从BOS到BOS的check校验	properties_examples/client.properties.bos-check	给定源BOS的Object列表和相关信息, 校验目的端是否和源端匹配

参数说明

- serverThreadNumber：此配置项设置单个请求发送到server后，server使用多少个线程来处理该请求，在源端和目的端以及server资源条件允许的情况下，一般以打满server带宽为标准来设置该数值，需要调整几次。**server占用内存 = serverThreadNumber * blockSize * maxConnectionsPerServer**
- serverList：迁移server列表，每个server由ip和端口号组成，server之间使用;进行分隔，示例如下：

```
 # serverList这个选项可以置空，置空没有可用的迁移server，迁移会暂停
serverList=192.168.1.1 8090;192.168.1.2 8090
```

- filesPerRequest：每次请求分发到迁移server的文件数量。建议该值配置完后每次迁移请求的完成时间大于30秒。因为一次请求使用一个tcp连接，需要保证在下个请求开始时上一次tcp连接可以正常断开，避免tcp fd过多，可以在开始迁移时尝试由小到大增大这个值，直到每次迁移请求完成时间大于30秒为止。
- taskList：该参数需要填写**任务列表文件的绝对路径**。**任务列表文件**中包含了一个或多个任务，这些任务是预先处理好的，每个任务对应一个**object_list文件的绝对路径**，每个object_list文件中的每一行对应一个最小粒度的数据迁移。配置示例如下：

```
 # 如下表示任务列表文件绝对路径为/home/work/list/task_list
taskList=/home/work/list/task_list
```

任务列表文件内容示例：

```
 /home/work/list/object_list1
/home/work/list/object_list2
# /home/work/list/object_list3
```

1. 以上内容表示taskList包括3个object_list，对应着3个object_list文件，每个object_list文件中又包含多行，每一行是一个最小粒度的数据迁移。
2. 当以#开头时表示该行被注释掉, 不会被当做迁移任务处理。

- listDelimiter和listFormat

listDelimiter表示object_list文件中每行数据的每一列之间的分隔符，listFormat表示object_list文件中每一行应该包括哪几列，列名以;分割，字段值大小写敏感。配置示例如下：

```
 listDelimiter=" "
listFormat=bucketName;objectName;dstObjectName
```

object_list文件内容示例：

```
 bucketA prefixA/aaa/object1 prefixB/bbb/object1
bucketA prefixA/aaa/object2 prefixB/bbb/object2
# bucketA prefixA/aaa/object3 prefixB/bbb/object3
bucketA prefixA/aaa/object4 prefixB/bbb/object4
```

1. 以上内容和listDelimiter和listFormat参数一一对应，每一行包括3列，分别是bucketName、objectName和dstObjectName，每一列之间用空格分隔。
2. 当以#开头时表示该行被注释掉, 不会被当做迁移任务处理。

目前文件列表格式中支持的的各个fields包括:

field	意义	示例
srcAk	源端ak	
srcSk	源端sk	
dstAk	目的端(bos)ak	
dstSk	目的端(bos)sk	
srcBucketName	源端bucket名称	
bucketName	源端bucket名称	
objectName	源端object名	
dstBucketName	目的端bucket名称	
dstObjectName	迁移之后的名称, 可以通过这一项来实现改名	
contentType	如果源端没有, 迁移之后存放到bos的contentType	text/plain
contentDisposition	如果源端没有, 迁移之后存放到bos的contentDisposition	attachment; filename="abc.mp4"
toBosIa	是否迁移到bos ia	toBosIa

> srcBucketName/bucketName是或的关系，两者取其一，优先取srcBucketName

- migrationStrategy：迁移策略
  1. FORCE_OVERWRITE: 强制覆盖目的端文件，可能会破坏目的端文件，危险，慎用，在确定需求的前提下使用
  2. KEEP_DESTITNATION: 如果目的端有同名文件文件存在则跳过
  3. KEEP_NEWER: 如果目的端有同名文件存在，且新于源端文件，则跳过，否则覆盖之；如果目的端没有"同名"文件(有可能迁移时文件需要改名)存在，则按照迁移规则直接复制到目的端

> - KEEP_DESTITNATION和 KEEP_NEWER 这两个迁移策略比较适合用于业务平稳切换时，在源端和目的端进行双写时使用，两者的效率相当，根据具体需求确定使用。
> - 迁移策略设置成三种策略之外的未知策略，会默认设置成KEEP_DESTINATION。

- headQps、putQps和bandwidth：QPS和带宽限速
  1. headQps表示从数据源迁移到BOS或者与BOS数据check对比时getObjectMeta请求的QPS上限，最大不超过5000次/秒；
  2. putQps表示从数据源迁移到BOS时putObject请求的QPS上限，最大不超过1500次/秒；
  3. bandwidth表示从数据源迁移到BOS时putObject请求的带宽上限，最大不超过3000MBps。

```
 headQps=5000
putQps=1500
bandwidth=3000
```

任务进度恢复

从 status.log 里可以看到相应的文件列表具体的迁移进度，如果client崩溃了可以从日志中找到相应的**断点**恢复迁移，进度恢复是根据任务列表(list)的粒度来进行的。

示例:

client.properties文件配置：

```
taskList=/home/work/list/task_list
```

task_list文件内容：

```
/home/work/list/object_list1
/home/work/list/object_list2
##### /home/work/list/object_list3
```

object_list1文件内容：

```
bucket1 obj1
bucket1 obj2
bucket1 obj3
##### bucket1 obj4
bucket1 obj5
```

object_list2文件内容：

```
bucket1 obj21
##### bucket1 obj22
bucket1 obj23
bucket1 obj24
bucket1 obj25
bucket1 obj26
```

status.log文件内容：

```
2015-12-31 17:13:42.013 /home/work/list/object_list1 2/-1
2015-12-31 17:13:45.014 /home/work/list/object_list2 3/-1
2015-12-31 17:13:48.014 /home/work/list/object_list2 4/6
2015-12-31 17:13:51.014 /home/work/list/object_list1 5/5 finished
2015-12-31 17:13:53.014 /home/work/list/object_list2 4/6
```

1. object_list1 2/-1 表示已经处理完了 object_list1 这个列表的第2行，-1 表示这个文件还没有读完不知道有多少行，如果这个时候崩溃了，由于采用了并发，所以并不知道具体是完成了列表中的哪几个文件的迁移。

2. 如果需要从某个进度位置重新开始任务，需要建立新的list文件，使用新的配置重新启动client。首先要根据status.log文件内容找出还有哪些object_list文件的数据没有迁移完，例如object_list2还没有迁移完，就将其放到新的task_list_continue文件里，并修改配置文件，示例如下：

修改client.properties文件：

```
taskList=/home/work/list/task_list_continue
```

修改task_list_continue 文件：

```
/home/work/list/object_list2
```

redo重传

当所有需要迁移的列表处理完之后，需要对迁移失败或者迁移过程当中出现问题的文件进行重传。 从迁移日志中整理出重传的列表：redo和unsent日志文件中的文件可以使用KEEP_DESTITNATION的迁移策略,但是对于failed的文件列表，需要考虑使用FORCE_OVERWRITE策略进行迁移，因为可能是因为迁移出现了数据错误。

异常处理

server异常处理

- 多步上传到BOS异常

  上传到BOS的multiupload在完成最后一个part的时候会调用 finishUpload，由于是多线程, 同个文件的多个part可能在最后一part完成之前未完成, 在调用finishUpload 的时候可能会失败, 处理这个问题的策略是：重试5次, 每次间隔20秒，重试5次失败后abort掉之前已经完成的part。

  > 建议：当出现5次重试依旧失败的情况，可以把blockSize尽量调大，使得part总数较少。

client 异常处理

- 当server不可用时, client在第一次尝试将迁移请求发送给该server时可以立即发现其不可用, 将该请求记录为unsent, 并标记该server不可用, 之后的请求不会再发往该server。如果确定该server修复之后, 可以touch一下client的配置文件client.properties, 触发client配置重新加载；建议设置一个crontab定时去touch配置文件client.properties。
- client达到系统分配的fd(socket fd)数量极限，检查是否filesPerrequet是否过小, 如果确定是这个原因, 则需要适当调大, 杀掉所有server(或者让client于所有server"失联"), 等待已经占有的大部分socket fd释放之后重新touch client的配置文件。

日志

1. src_not_found.log

   源端未找到，在尝试迁移所给的文件列表中的文件时，发现源文件找不到。

   示例：

   ```
   2015-12-31 17:13:48.014 bucket1 obj24
   2015-12-31 17:13:48.015 bucket1 obj25
   2015-12-31 17:13:48.016 bucket1 obj26
   ```

2. migrate_failed.log

   迁移过程发生异常(包括迁移过程中写目的端中途失败，迁移后校验时发现读目的端文件信息异常或者目的端和源端不一致等情况)，发生概率低，需要以FORCE_OVERWRITE策略重做。

   示例：

   ```
   2015-12-31 17:13:48.014 bucket1 obj24
   2015-12-31 17:13:48.015 bucket1 obj25
   2015-12-31 17:13:48.016 bucket1 obj26
   ```

3. redo_list.log

   迁移任务失败(包括迁移过程中读取源端文件数据或信息异常，迁移后校验时发现目的端文件不存在等情况)，需要重做。

   示例：

   ```
   2015-12-31 17:13:48.014 bucket1 obj24
   2015-12-31 17:13:48.015 bucket1 obj25
   2015-12-31 17:13:48.016 bucket1 obj26
   ```

4. dst_not_match.log

   只有check任务会生成，表示目的端和源端不一致，只用来迁移则无需理会。

   示例：

   ```
   2015-12-31 17:13:48.014 bucket1 obj24
   2015-12-31 17:13:48.015 bucket1 obj25
   2015-12-31 17:13:48.016 bucket1 obj26
   ```

5. status.log

   当前进度，每3秒更新一次，x/y 表示处理进度，x为已经处理的数量，y表示总量；y为-1表示未读完该文件列表，完成迁移的列表后会标记一个finished。

   示例：

   ```
   2015-12-31 17:13:42.013 object_list1.txt 2/2 finished
   2015-12-31 17:13:45.014 object_list2.txt 3/-1
   2015-12-31 17:13:48.014 object_list3.txt 4/6
   ```

6. unsent.log

   由于网络或者server不可用等问题在发送迁移请求给server时报错，告知有哪些文件在尝试迁移的时候server拒绝了。

   示例：

   ```
   2015-12-31 17:13:48.014 bucket1 obj24
   2015-12-31 17:13:48.015 bucket1 obj25
   2015-12-31 17:13:48.016 bucket1 obj26
   ```

7. client.log

client端日志，记录着各个迁移任务分发的情况以及当前client的情况，可以从这个日志中分析出文件的平均大小，和单个文件迁移所用时间，从而调整迁移参数，实现高效迁移。

8. server.log

迁移server端日志，主要包含文件迁移的过程(head get put verify)，根据request-id可以跟踪某个迁移任务的处理过程。

> 第1-6种log的格式除了每行开头加上了产生该行的时间之外，和所提供的文件列表格式一样。

版本记录

- BosImport工具包 [2020-10-22] 版本号1.0.2
  - 修复Bug。
- BosImport工具包 [2020-08-17] 版本号1.0.1
  - 新增支持金山云KS3数据迁移到BOS；
- BosImport工具包 [2018-12-05] 版本号1.0.0
  - 支持多种迁移方式, 支持迁移不同的源端到目的端，目前支持AWS S3、阿里云OOS、腾讯云COS、七牛、小米生态云、BOS等；
  - 支持实时迁移并发度/压力调节，server"热升级"；支持迁移任务热启停，迁移进度实时显示；
  - 大文件切块传输, 块大小支持实时配置, 最大效率地利用迁移机器的带宽资源；
  - 负载均衡, client会选择没有压力的server进行迁移任务；
  - 支持自定义迁移的任务列表和文件数目；支持迁移任务QPS和流量限速；支持多种迁移策略。

月光宝盒物理迁移设备

月光宝盒介绍

产品介绍

月光宝盒是 BOS 推出的一种通过物理寄送上云方式，能够实现本地 IDC 数据的迁移上云。

月光宝盒适用于百 TB 级数据的传输和迁移上云，它使用特定存储设备在百度智能云和客户之间传输大量数据。它解决了大规模数据传输中出现的高昂网络成本、较长传输时间和安全等问题，且具有简单、快速、安全、成本低的优点。

月光宝盒由 BOS 为您提供移动存储设备，该设备具有大容量、可视化操作、军工级别防护加密等特性。单台月光宝盒设备的标定容量为 96 TB，实际使用容量为 83 TB 左右/每台，适用于百 TB 级别的本地 IDC 数据迁移上云场景。

使用场景

- 您的网络环境不好（带宽小、网络不稳定），且带宽成本非常昂贵；
- 希望尽快完成数据的上云，但是专线方式的费用过于昂贵；
- 自己没有硬盘或者无可移动的数据存储和传输设备。

优势对比

相较于其他集中迁移方式，月光宝盒的优势对比如下：

迁移方式	迁移成本	迁移时间	安全性
通过公网上传	较高，会占用 IDC 大量出口带宽	取决于公网出口带宽大小	公网传输部分不安全
拉专线上传	专线建设成本较高	专线建设时间通常较久	非常安全
月光宝盒上传	免费	设备往返物流时间+数据上传时间，较快	安全，数据全程服务端加密

产品特性

- 传输效率高：支持万兆光口，最高可达到 10MB/s 数据吞吐能力。单台设备可在几天之内完成几十 TB 数据的迁移上云；
- 安全保密：数据传输全程采用 AES256 加密，即使设备丢失被盗数据也无法被解密读取；
- 稳定可靠：外层采用军工级抗震防摔保护，从若干米高摔下可保护内部磁盘不被损坏。内置页部署了RAID5阵列，进一步提高数据可靠性；
- 简单易用：提供界面化操作界面，非 IT 人员可直接上手操作，像 U 盘拷贝一样简单；

产品规格

单台月光宝盒的规格信息如下：

规格项	规格信息
重量	约30KG左右
尺寸	宽度 深度 高度 = 194mm x 460mm x 420mm
标定容量	96 TB（实际可用容量约83TB左右/台）
网络支持	默认1GbE电口*2 + 10GbE光口*2*
电源功率	850W 110V/220V AC输入

月光宝盒操作流程

整体操作流程概述

步骤	用户的任务	百度智能云的任务
第一步	用户审核公测申请	审核公测申请
第二步	告知IDC机房地址，所需上云数据量，联系人信息等	向用户寄送合适数量的月光宝盒设备
第三步	注意查收设备，并支付运费	无
第四步	参照下文说明，将本地数据拷贝至月光宝盒	关注用户迁移过程，响应用户疑问
第五步	将装满数据的月光宝盒寄送回百度智能云机房（机房地址可与百度智能云接口人获取），并支付邮费	关注用户迁移过程，响应用户疑问
第六步	无	接收用户寄回的设备，将数据拷贝至用户预期云上位置

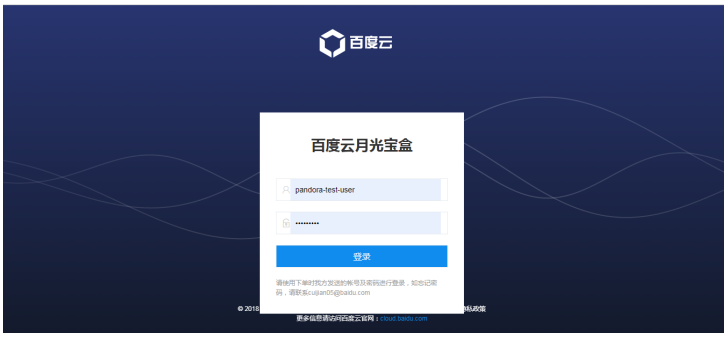
详细操作流程

1. 公测申请

- 目前月光宝盒处于公测阶段，请点击公测申请或与您的客户经理联系；
- 在进行简单筛选后，我们会通过邮件或电话方式与您进行联系，获取您的 IDC 机房地址，用于将月光宝盒设备寄送到您机房；
- 设备寄送费用需要由您承担。

2. 环境配置

- 将月光宝盒设备与您 IDC 的网络环境联通。每个月光宝盒设备具有 2 个 1 GbE 电口 和 2 个 10GbE 光口。电口需要的网线为普通电脑标准网线插口，光口网线类型为 SFP+AOC 类型通用国标光缆接口；
- 月光宝盒内置 Ubuntu 操作系统，可拿任意显示器与月光宝盒相连，在浏览器打开月光宝盒在机房内分配的ip地址，或直接输入127.0.0.1:8080，即可打开月光宝盒登录页面。
- 月光宝盒登录密码，可通过向百度智能云同事咨询获取。



3. 创建任务

登录月光宝盒后，点击左上角 “新建上传任务”，在展开的新页面中创建及描述数据上传任务。



- **任务名称**：为该迁移任务进行命名，便于后续管理
- **源地址类型**：即待上云数据在本地 IDC 的存储管理方式。 如是可通过网线直联的块设备或带有 USB 插头的设备，可与月光宝盒直接连接对拷，则选择“本地 USB 设备”。这时，需要同时填写待上传数据在本地的绝对路径。 如待上传数据存储在本地文件系统中，则选需选择本地文件系统的协议或操作系统，同时需填写挂载点 IP 及待上传数据在文件系统的绝对路径。 目前月光宝盒支持本地 Linux 文件系统的迁移，暂时不支持 Windows 文件系统。
- **目的地址**：即数据上云后预期存储的地址。 目前月光宝盒支持向百度智能云对象存储 BOS 迁移数据，因此用户可在“目的地址”中填写 BOS 产品的存储路径。 如填写“Bucket/A/B/C”,则月光宝盒迁移的数据后续会在相应 BOS 路径进行上传。
- **自定义加密**：为了保证设备传输过程中的意外情况（设备被盗等），我们提供了自定义加密能力。您需要为当前任务定义一个自定义密钥，该任务上传的数据都默认使用该密钥加密。 密钥需由字母和数字组成，长度 6-64 字符。

4. 管理任务

用户创建完成的任务会展现在月光宝盒任务列表中。



您可对列表中的任务进行启动/停止操作，也可基于该任务的传输情况打印日志。 如果用户认为创建得该任务有误，希望将此任务已经拷贝到月光宝盒中的数据清除掉，点击“清除数据”按钮即可。已经传输完成的任务，可以在“已完成任务”列表查看，同样可对希望撤回的拷贝任务进行数据清除。



5. 设备寄回

您确认完成本地数据向月光宝盒的拷贝任务后，可通过电话或邮件联系百度智能云同事，并通过物流将设备寄回百度智能云机房。下一步将由百度智能云机房同事完成数据向云上 BOS 的拷贝。数据完成最后的上云步骤后，百度智能云会及时告知您。

> **说明：**
>
> - 月光宝盒也支持作为 NFS 客户端使用。如您需要该种使用方式，请提前与我们联系。

BOS-PC客户端

安装及登录

什么是 BOS 桌面（BOS Browser）

BOS 桌面是由 BOS 官方发布的图形化桌面客户端，支持 Windows 和 Mac 操作系统，提供类似资源管理器的功能，例如文件/文件夹的上传、下载、删除和重命名等操作。本文介绍如何快速安装并登录 BOS Browser，具体功能及使用可查看 BOS 桌面使用指南。

> **注意：**
>
> - 您在 BOS 桌面对资源做的所有修改，都将同步至控制台，包括上传新文件、重命名文件、删除文件和新建文件夹等。 BOS 桌面还支持本地实时同步功能，支持将本地文件夹中的内容自动同步到关联的云端文件夹中，当您本地文件夹中新增文件时，云端文件夹会自动同步新增文件。
> - 2023 年 4 月 7 日后 BOS 桌面升级为功能更加丰富的新版本，新旧版本兼容。**旧版桌面工具将停止迭代，欢迎使用新版本。**

BOS 新版桌面工具下载

支持平台	操作系统	版本（推荐）	下载地址
Windows	• Windows7、8、10，x86/x64位操作系统 • Windows Server 2008、2012、2016、2019 x86/x64位操作系统	1.4.9	1.4.9-windows.exe
macOS	macOS（适用于 Intel 芯片）	1.4.9	1.4.9-mac.dmg
	macOS（适用于 Apple Silicon 芯片）	1.4.9	1.4.9-arm64.dmg

BOS 新版桌面工具历史版本下载

支持平台	操作系统	版本	下载地址
Windows	• Windows7、8、10，x86/x64位操作系统 • Windows Server 2008、2012、2016、2019 x86/x64位操作系统	1.4.8	1.4.8-windows.exe
		1.4.7	1.4.7-windows.exe
		1.4.6	1.4.6-windows.exe
macOS	macOS（适用于 Intel 芯片）	1.4.8	1.4.8-mac.dmg
		1.4.7	1.4.7-mac.dmg
		1.4.6	1.4.6-mac.dmg
	macOS（适用于 Apple Silicon 芯片）	1.4.8	1.4.8-arm64.dmg
		1.4.7	1.4.7-arm64.dmg
		1.4.6	1.4.6-arm64.dmg

如何登录

> **说明：**以下操作步骤以新版桌面工具为例。

BOS 桌面客户端可以采用两种方式进行登录：

1. AK/SK 方式 。适合用户能获取 AK/SK 的场景，该登录方式可选配置精确的存储桶或者访问文件路径，支持的功能广泛。

2. 文件夹分享链接与提取码方式。适合仅能获取文件夹分享 URL 与提取码的场景，主要支持文件 / 文件夹的批量下载功能。

AK/SK 登录

- AK（Access Key ID）/SK（Secret Access Key）相当于百度智能云为用户分配的用户名和密码，用于对用户的调用行为进行鉴权和认证。在您注册百度智能云账号并成功登录 Console 控制台后， 您可通控制台申请并管理自己的 AK/SK。详细的获取方式，您可以查看获取 AK/SK。

> **注意：**为了你的让您使用更加便捷，提供了保持会话的能力，选择后将为您保留登录的 AK/SK 信息 你无需重复输入密钥信息。**为了您的数据安全，请您不要在临时电脑上勾选此功能。**

- 访问路径格式为：bos:/bucketname/path/



文件夹分享链接登录

- 当您获取到文件夹分享链接与提取码后可使用文件夹分享链接与提取码完成登录。
- 登录后仅可获取该文件夹下文件列表，并可进行批量文件、文件夹下载。

> **注意：**登录前请注意链接和提取码的有效性，若过期，请**重新生成链接及提取码**后登录。



退出登录

- 当您需要切换登录密钥或者登录方式时，可以先退出登录后，切换登录方式，或者更新登录信息。
- 您可在应用设置->关于中点击【退出登录】退出当前账号。



常见问题

- MacOS 打开应用程序报错：bos-browser已经损坏

  修复： 在终端中执行 sudo spctl --master-disable，随后输入管理员密码即可。如依然提示损坏可在终端中执行sudo xattr -r -d com.apple.quarantine /Applications/bos-browser.app 来移除应用程序的保护标志，请确认将/Applications/bos-browser.app更改为当前桌面工具的安装路径。

- 子用户登录时被系统拒绝访问？

  修复：需要在多用户管理中，给子用户创建授权策略，给子用户授予"ListBuckets"权限。策略内容 demo 如下：

```
{
 "id": "policy_af6c3c62c1da432497b1fdc0d82bca12",
 "version": "v1",
 "accessControlList": [
  {
   "service": "bce:bos",
   "region": "*",
   "resource": [
    "bucket-name",
    "bucket-name/*"
   ],
   "effect": "Allow",
   "permission": [
    "*"
   ]
  },
  {
   "service": "bce:bos",
   "region": "*",
   "resource": [
    "*"
   ],
   "effect": "Allow",
   "permission": [
    "ListBuckets"
   ]
  }
 ]
}
```

意见反馈

如果您对 BOS 桌面有任何使用上的意见和建议，欢迎您通过以下方式联系我们。

- 工单
- Github Issue

使用指南

BOS 桌面是百度对象存储 BOS 提供的图形化管理工具，支持在 Windows 和 Mac 系统中安装后使用。关于 BOS 桌面的下载及安装，请查看安装及登录。

BOS桌面功能概览

BOS 桌面支持 Object 级别的操作和管理，包括文件的上传和下载、删除、重命名、创建文件夹、查看文件ACL等操作。您通过 BOS 桌面对 Bucket 中文件做出的修改（包括上传、删除、重命名等）会自动和控制台同步。

功能名称	功能描述
新建存储桶（Bucket）	支持在 BOS 桌面新建 Bucket
查看Bucket详情	支持查看存储桶基本信息
搜索存储桶	支持搜索已创建的存储桶
删除Bucket	支持在 BOS 桌面删除非空 Bucket，需要确定您的存储桶不存在任何文件和未完成的分片（Part）
文件（文件夹）上传	支持通过拖拽或者点击的形式将本地文件上传到 BOS 的 Bucket 中
文件（文件夹）下载	支持将 Bucket 中文件、文件夹下载至本地
文件（文件夹）批量下载	支持将 Bucket 中文件、文件夹批量下载至本地
查看文件详情	支持查看存储桶中的文件基本信息
文件重命名	支持对 Bucket 中的文件进行重命名
文件复制	支持通过下拉选择的方式将 Bucket 中文件复制到 BOS 中其他 Bucket 中
文件/文件夹分享	支持分享文件和文件夹，支持设置分享的有效时间
文件删除	支持单个和批量删除 Bucket 中已有文件
搜索文件	支持以前缀搜索的方式对存储桶中的文件进行搜索
新建文件夹	支持在 Bucket 中新建文件夹，用以添加新的存储目录
本地文件自动同步上传	支持通过同步盘的方式，将本地数据自动同步到 BOS，无需手动上传
传输管理	（功能适用于 v0.2.4 及更低版本）单个或批量文件的上传，下载等操作均可在任务管理中进行查看，暂停，重试等操作
任务管理	（功能适用于 v1.0.0 及更高版本）单个或批量文件的上传，下载，移动，复制，删除，重命名操作任务均可在任务管理中进行查看和管理
指定路径登录	用户登录时，在配置AK/SK登录的基础上，可选配置访问路径，路径访问方式支持单个存储桶或者具体存储桶下某个文件路径，该功能可实现通过桌面工具进行细粒度文件管理。
本地日志	支持全部日志（包含info、warm、error等全部信息）与仅含info信息，默认开启，您可在应用设置中修改配置
上传/下载性能优化	为了进一步提升上传下载性能，BOS Browser提供控制并发任务数量的能力，用户可根据自身配置和网络环境等因素对相关参数进行灵活调整，从而获得最大传输效率。

BOS桌面使用指南

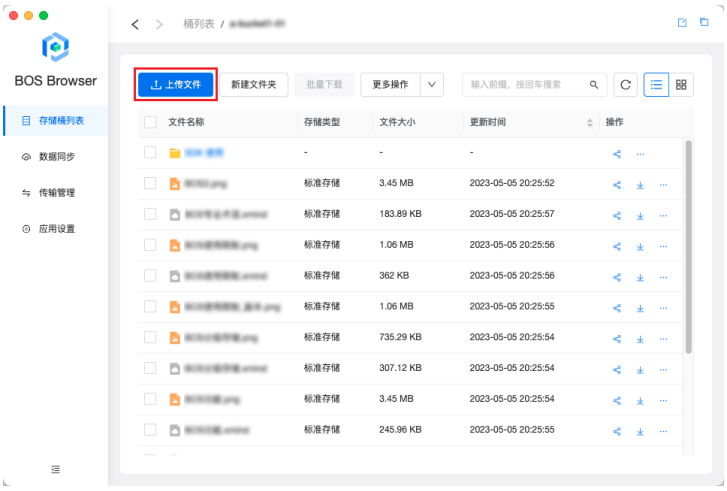
下面以 Mac 操作系统中 v 0.1.0 相关操作为例进行说明。若您使用的是 Windows 系统，也可以参考该使用指南。

1. 创建/删除/查看存储桶

功能	说明	如何操作
创建存储桶	可以通过桌面工具客户端直接创建存储桶	1. 在存储桶列表，单击左上角【新建Bucket】 2. 正确填写存储桶名称，并选择地域、默认存储桶类型、访问权限 3. 单击确定，即可完成创建
删除存储桶	删除存储桶前，请确认存储桶中文件与碎片已清空	1. 在存储桶列表，单击对应存储桶操作栏中【删除】 2. 确认存储桶中的数据已全部清空，单击确定进行删除
查看存储桶详情	存储桶详细信息包含存储桶名称、地域、访问权限，官方存储桶域名。	您可通过单击存储桶列表操作栏中【查看详情】

2. 文件/文件夹上传

点击 BOS 桌面中的上传按钮选择单个或者批量文件与文件夹进行上传，或者也可以直接通过拖拽的方式，将本地文件（文件夹）直接拖拽到目标 Bucket 中。多个文件上传任务可以在左侧目录【传输管理】-【上传】中进行查看与管理。



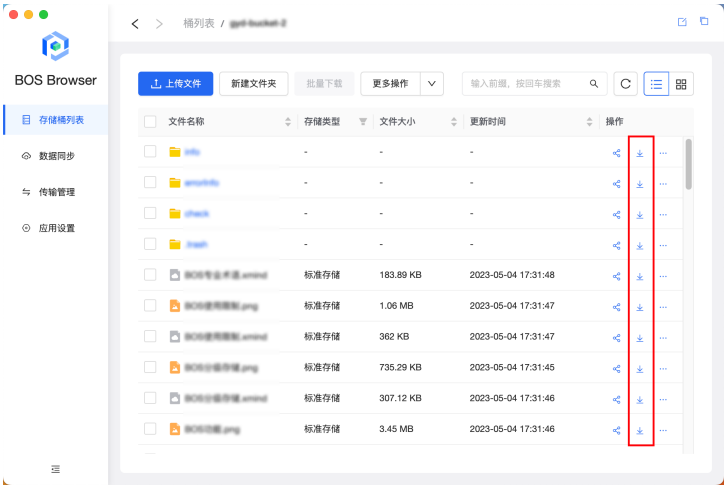
说明：

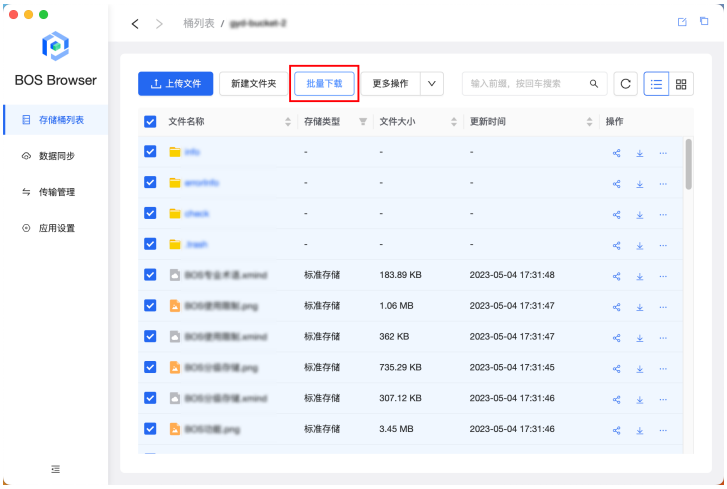
- 如果上传文件时提示“上传失败”，可能是由于您在登录 BOS 桌面后一直未进行操作而导致的连接超时。此时需要关闭 BOS 桌面后重新启动并登录，然后在上传列表中重新上传即可。

3. 文件/文件夹下载

可直接下载单个文件/文件夹，也可对多个文件/文件夹进行批量下载。

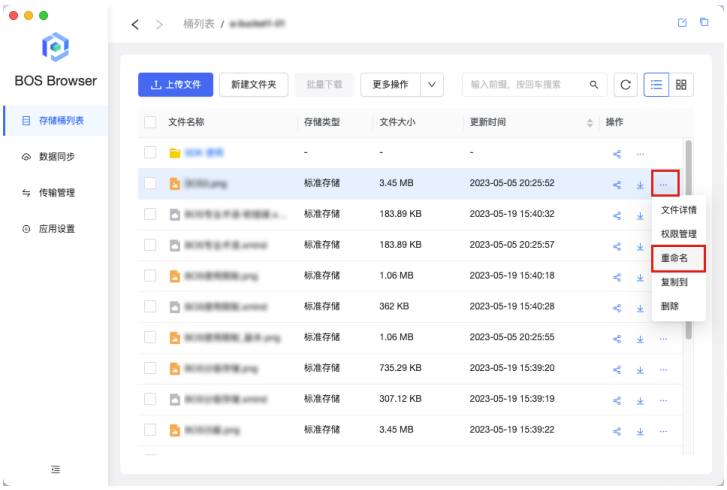
- 点击 BOS 桌面中文件（Object）列表操作栏中的【下载】图标，可对单个文件/文件夹进行下载。 也可选择多个文件/文件夹后，点击顶部【批量下载】按钮，对多个文件/文件夹批量下载，下载任务在左侧导航栏【传输管理】-【下载】中进行查看与管理。





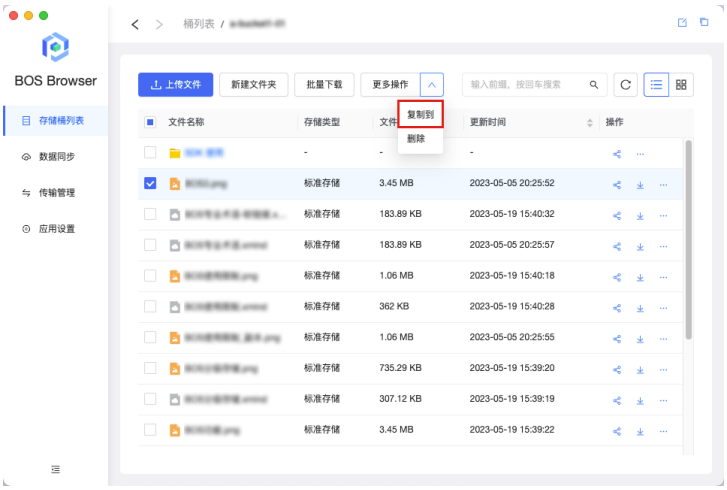
4. 文件重命名

点击文件（Object）列表操作栏中的【...】图标，下拉列表中选择【重命名】。点击之后，在弹框中输入文件的新名称，点击确认后保存。



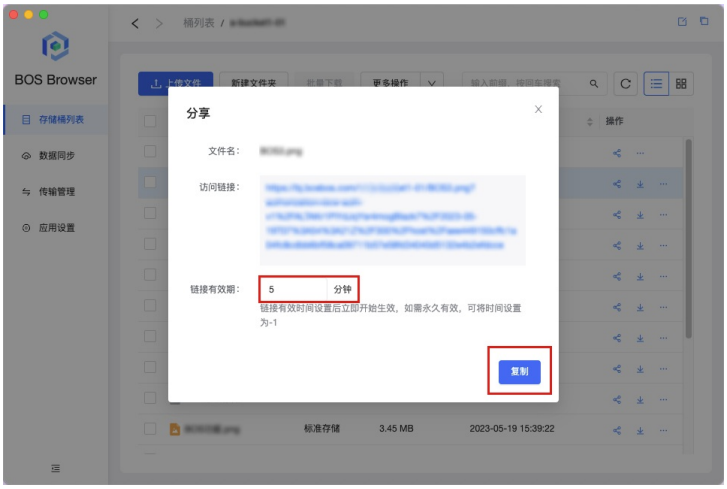
5. 文件复制

点击文件（Object）列表操作栏中的【...】，下拉列表中选择【复制到】，将文件复制到其他 Bucket，需在复制到弹框中选择或者直接输入希望复制到的目标 Bucket或更细粒度的目标文件夹，存储路径需包含存储桶，可选配置到具体目录。批量复制文件，可选择多个文件后，点击顶部【更多操作】-下拉选择【复制到】。

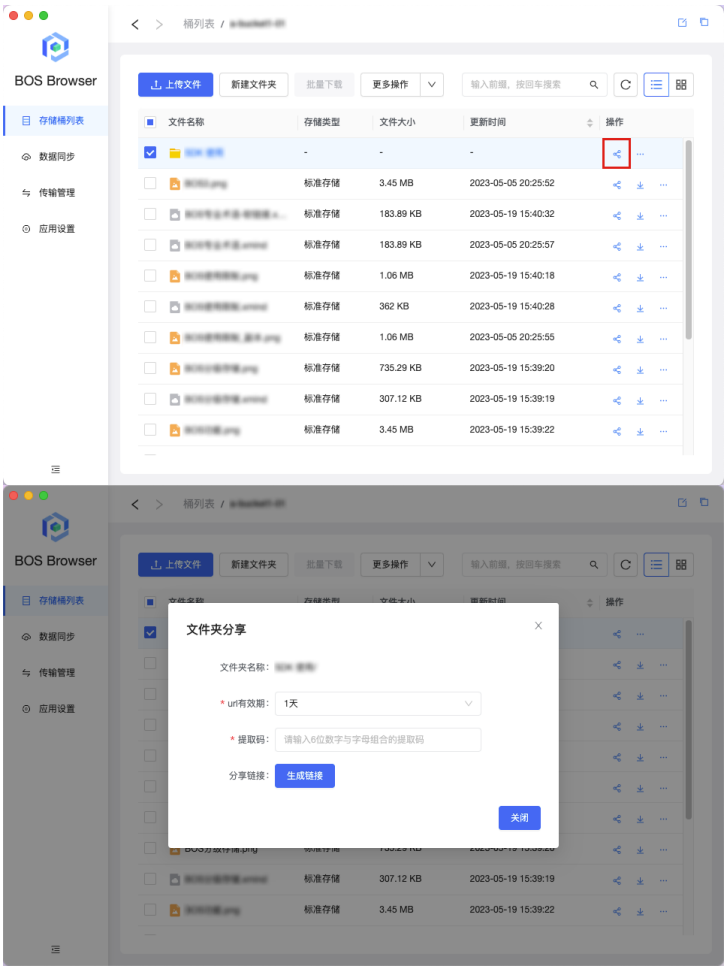


6. 文件与文件夹分享

6.1 文件分享 当需要分享单个文件时，可以点击 BOS 桌面中 Object 列表后的【分享】图标，设置分享链接有效期后，点击复制，即可进行文件分享。



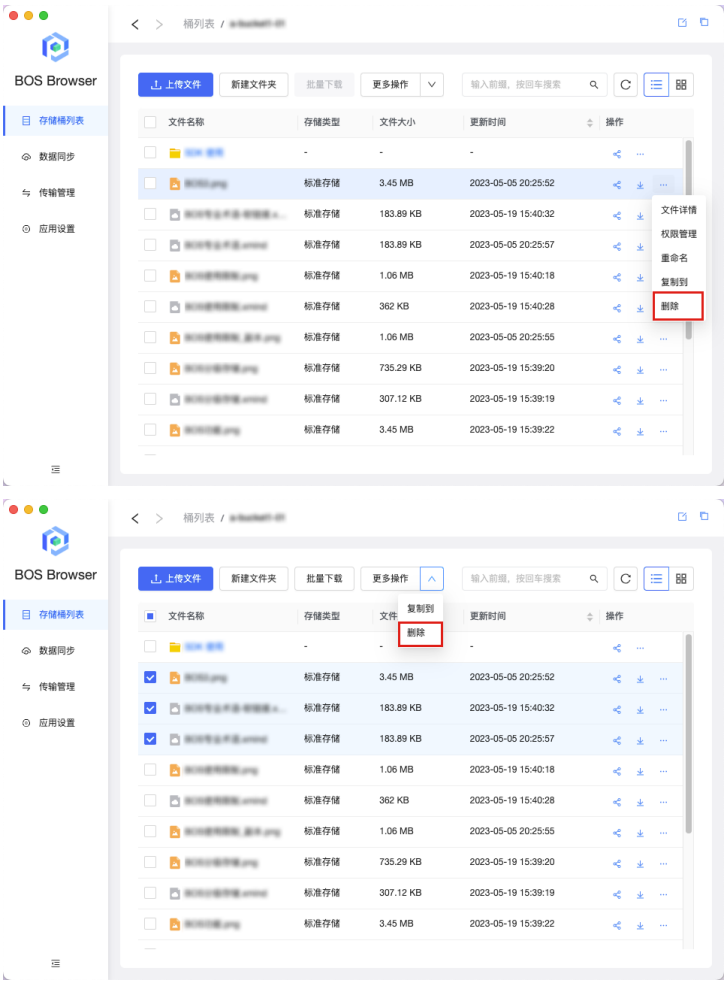
6.2 文件夹分享 为了提升分享的效率，您可以为直接分享某个目录（文件夹）下的所有文件，操作方式为点击选中待分享的文件夹，并点击操作栏中【分享】图标，设置分享链接有效期与6位提取码，点击生成链接，复制进行文件夹分享链接与提取码，即可进行文件夹分享



7. 文件删除

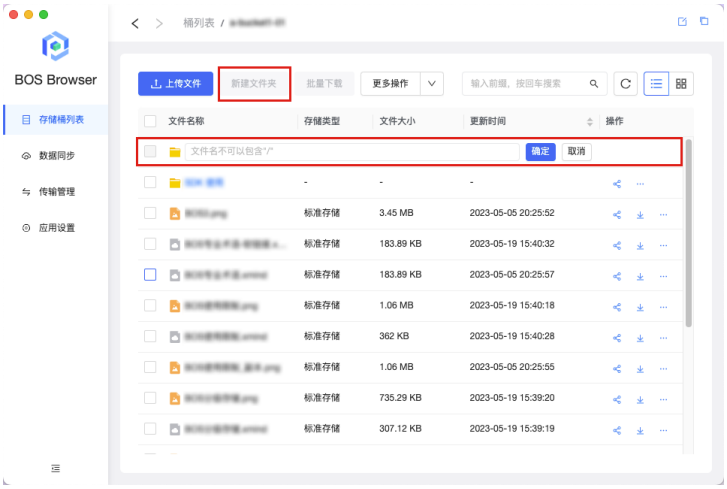
点击文件（Object）列表操作栏中的【...】，下拉列表中选择【删除】，将文件进行删除。也可选择多个文件后，点击【更多操作】-【删除】进行文件的批量删除。

文件一旦在 BOS 桌面中删除，将在 BOS 中彻底删除，请谨慎操作。



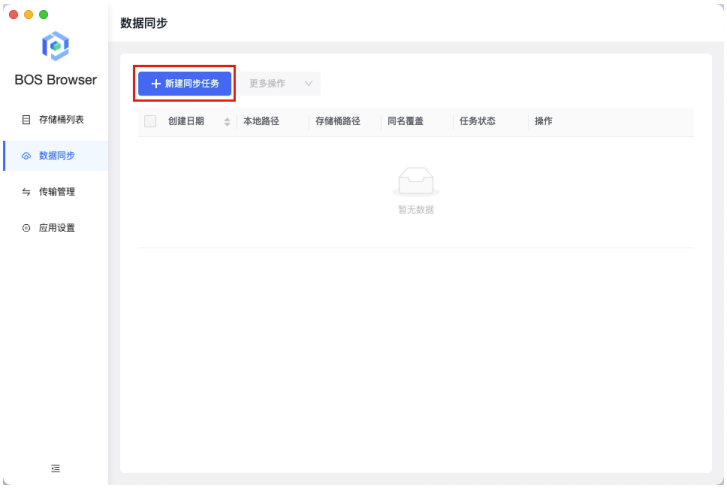
8. 新建文件夹

点击【新建文件夹】按钮可为 Bucket 新建目录。新建成功后，可以在新的文件夹中对文件进行上传下载等管理操作。



9. 本地文件自动上传（数据同步）

BOS 桌面支持本地文件自动上传，到达自动将本地目录与云端数据同步的效果。当在 BOS 桌面中设置数据同步后（旧版本中【同步盘】后，已选本地目录中的文件将会实时自动同步到 BOS 的 Bucket 中。当有新的文件被添加到本地目录中，BOS Bucket 中的文件列表也会自动完成同步更新。可以对已经建立的同步任务进行暂停、查看日志、删除操作，通过【更多操作】可批量对同步任务进行以上管理。



> **说明：**
>
> - 用户需要选择一个本地文件夹和一个云端文件夹，确认后在这两个文件夹间建立映射关系；
> - BOS 文件目录可选择 Bucket，也可选择 Bucket 中的一个文件夹；本地文件夹只能选择本地的文件夹。两者都不能选择单个文件；
> - BOS 对映射关系的数量有限制，同一账户最多只能同时存在 20 个映射关系。
> - BOS中的数据变更不会影响本地的文件夹数据，本地文件自动上传操作仅会单向的将本地文件同步至BOS，不会将BOS的改变同步至本地。

对于不同的情况，BOS 桌面会分别执行不同的操作：

- 一个文件在本地目录中存在，但是在存在映射关系的 Bucket 中不存在，那么该文件会被自动同步上传到该 Bucket。
- 一个文件在本地目录和存在映射关系的 BOS Bucket 中均已存在，但是该文件在本地目录中发生改动，那么该文件会被覆盖上传到该 Bucket。
- 一个文件在本地目录和存在映射关系的 BOS Bucket 中均已存在，但该文件在本地目录中被删除，那么该文件将会继续保存在BOS 的指定Bucket中，BOS不会同步删除该文件。
- 一个文件在 Bucket 中存在，但是在存在映射关系的本地目录中不存在，那么该文件不会自动同步到本地目录中。若您希望将该文件保存在本地目录中，请从 Bucket 中下载该文件。

> **说明：**
>
> - 本地目录中同名但内容不同的文件会上传覆盖到 Bucket。若您不希望覆盖上传，请区别文件名。

10. 下载分享文件夹

当您获取BOS文件分享链接及提取码后，您可使用BOS桌面工具查看并批量下载BOS分享链接中的文件。

前期准备 获取BOS文件分享链接及提取码。使用网页版BOS控制台或者桌面工具生成文件分享链接的过程请见文件夹分享。

操作步骤

1. 打开BOS桌面工具登录页，使用文件分享链接登录BOS桌面工具。



2.在 BOS 桌面工具左侧的导航栏中选择收到的分享，可查看分享的文件列表信息。

3. 点击需要批量下载的文件前的复选按钮，然后点击页面上方的批量下载，开始下载。

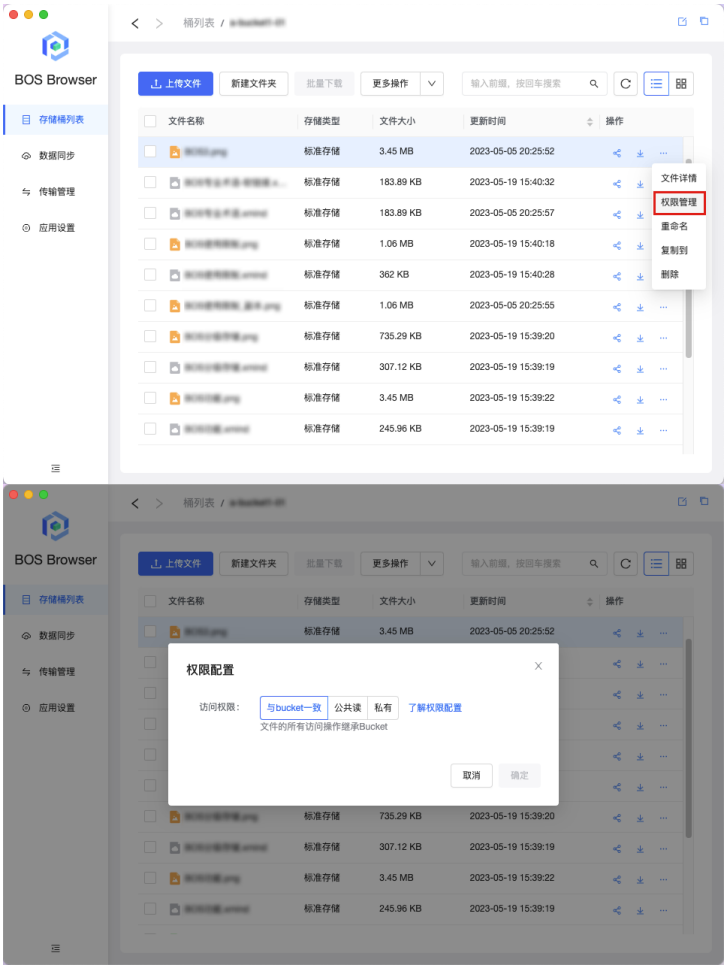
4. 在 BOS 桌面工具左侧的导航栏中选择任务管理，可查看任务状态。

5. 点击桌面工具右下角【注销】按钮可退出文件分享链接登录。

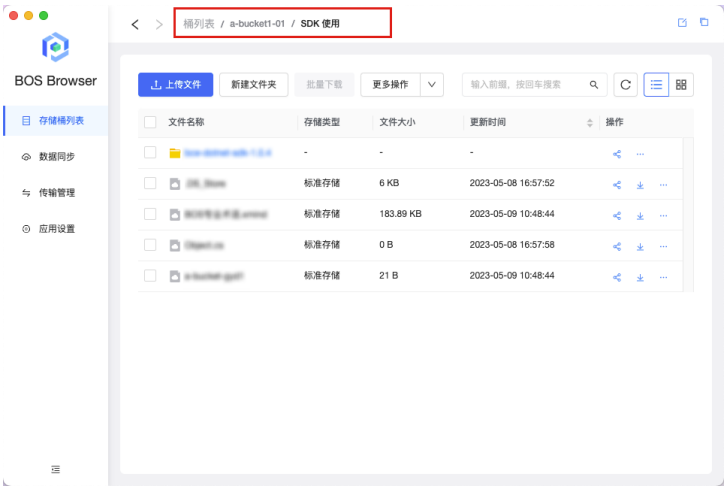
> **说明：**请在链接到期前完成文件下载，链接到期后桌面工具将自动退出登录，进行中的下载任务将强制停止。

11. 搜索文件

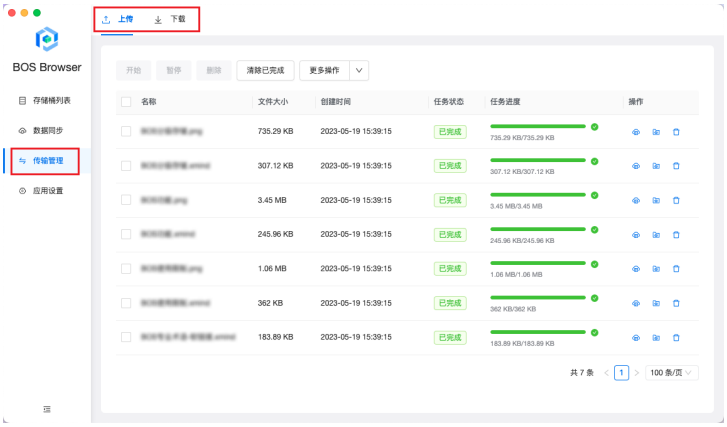
可通过存储桶内右上方的搜索框，输入文件名进行搜索。支持文件名前缀搜索。 **12. 查看文件详情** 可通过单击文件名或右键单击操作栏中的详情来查看文件详情，文件详细信息包含文件名、修改时间、访问权限、ETag、Headers。 **13. 修改文件访问权限** 对象权限管理，可通过单击文件右侧操作栏中【...】图标，下拉选择【权限管理】进行查看与修改。



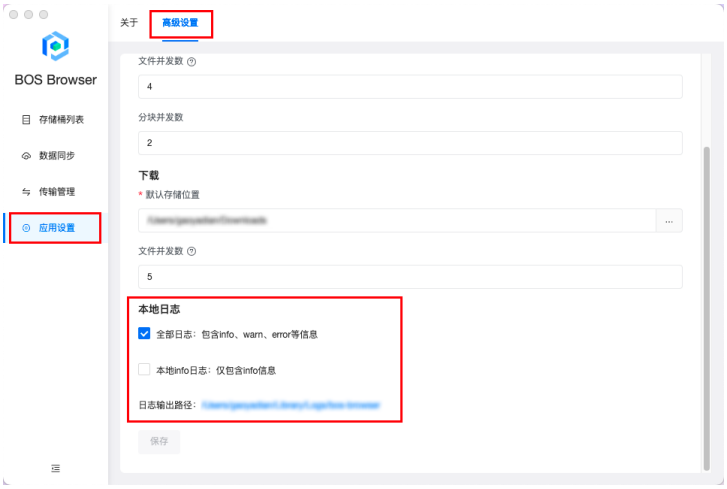
14. 直接输入路径与路径复制 可在工具地址导航栏直接输入路径，点击回车将自动定位到指定的路径，需注意路径需要包含完整的存储桶或存储桶/文件夹名称/，例如：bos:/a-bucket1-01/SDK使用/。也可以点击路径右侧的复制图标复制访问路径，分享给他人，可配合ak/sk密钥登录方式的，BOS访问路径一同使用，实现文件列表精准定位。



15. 搜索存储桶 可通过左侧存储桶列表上方的搜索框，输入存储桶名称快速定位存储桶。 **16. 传输管理（适用于 v0.2.4 及更早期版本）** 传输管理支持对历史的文件上传与下载任务进行管理，可批量进行开始，暂停，删除等操作。



17. 任务管理（适用于 v1.0.0 及后续版本） 任务管理支持对单个或批量文件的上传，下载，移动，复制，删除，重命名的任务进行查看和管理。 **18. 本地日志** 您通过桌面工具进行的各种操作将生成本地日志，供您查看与审计等场景，支持全部日志（包含info、warn、error等全部信息）与仅含info信息，日志按天生成。您可在【应用设置】-【高级设置】中修改本地日志配置，若二者都不勾选则默认默认不开启本地日志，不建议您关闭本地日志。点击日志输出路径，可打开本地日志所在文件夹，方便您查看。



19. 上传/下载性能优化 为了进一步提升上传下载性能，BOS Browser提供控制并发任务数量的能力，用户可根据自身配置和网络环境等因素对相关参数进行灵活调整，从而获得最大传输效率。 用户可在【应用设置】->【高级设置】中对并发任务数量进行动态调整。



****相关参数说明****

- 上传

参数	描述
文件并发数	同时运行上传任务的任务数量，默认设置为5，用户可进行调整，合法范围为1-1000000
分块并发数	文件分开上传时同时执行的分片数量，默认设置为5，用户可进行调整，合法范围为1-10000

- 下载

参数	描述
文件并发数	同时运行下载任务的任务数量，默认设置为5，用户可进行调整，合法范围为1-6

> **说明：**
>
> - 并发数的设置不宜过大。如果并发数配置过大，可能会因线程间资源切换、抢夺等原因，导致上传/下载的性能下降，因此需要根据实际的文件大小和网络情况调整；
> - 在客户端网络情况不理想的情况下，为了减少受网络波动影响导致任务失败的情况，可以适当减小上传/下载的段大小和整体的并发数量；
> - 目前暂只支持最多设置6个下载任务同时进行。

常见问题

文件下载速度慢

解决方案：

- 重新启动应用，重新尝试下载文件
- 通过分享文件，分享链接的方式，将链接复制到其他下载工具
- 若以上方法无效，可提交工单并提供log日志、磁盘大小等信息

出现闪退问题

若有很多上传&下载记录，请尝试删除db文件

删除db文件方式：

- window：C:\Users${用户名}\AppData\Roaming\bos-browser 下面的bos-brower-db.json文件。
- mac：/Users/${用户名}/Library/Application\ Support/Electron/ 下面的bos-brower-db.json文件。

BOS-FS挂载工具

概述

概述

BOS FS用于实现将远程bos服务挂载到本地文件系统中，实现类似于网络文件系统的功能，目的是便于用户使用系统命令或者类POSIX接口访问BOS上的文件，就像使用本地磁盘一样方便，但不适用于生产化环境性能要求较高的应用，必须依赖挂载或文件系统语义的业务推荐使用BOS FS，其余场景推荐使用BOS CMD。

BOS FS基于FUSE实现，因此依赖于支持FUSE的操作系统。目前支持基本的POSIX语义调用，暂时不支持硬链接、扩展属性，文件ACL和文件锁等。

安装包	名称及下载地址	MD5	说明
源码包	bosfs-1.0.0.13.4.tar.gz	d8049ed01c23ed850e6f67081c019b5a	安装源码包， 如果您需要自定义一些功能可以直接进行修改
libfuse	fuse-2.9.4.tar.gz	ecb712b5ffc6dffd54f4a405c9b372d8	安装libfuse用到的源码包

使用限制

- bosfs工具不适生产环境对性能要求较高的场景。
- 元数据相关请求(尤其是列举目录list dir)因需要远程访问bos服务，性能较差。
- 重命名(rename)相关请求不保证原子性，本地与bos后端数据可能不一致。
- 不完全支持文件系统语义，例如文件权限以及硬连接等支持不完善。
- bosfs内文件操作将映射到本地磁盘空间，需保证本地磁盘空间容量足够。
- 多个客户端访问同一个或多个挂载点时，数据一致性由使用侧维护，可能出现多端数据不一致情况。
- 机器重启后需要进行重新挂载。

安装

fuse模块安装

用户安装bosfs前需要先确保目标系统已经安装FUSE内核模块，可以通过`ls /dev/fuse`或`modprobe fuse`命令判断是否安装。如果没有安装，执行以下步骤：

- centos系统，使用root权限 执行`yum -y install fuse fuse-devel`
- ubuntu系统，使用root权限 执行`apt install libfuse2 libfuse-dev`;
- 若无法安装，或者系统已安装的fuse版本低于2.9.4，则需要**进行源码安装**:
  下载libfuse源码包，自行编译安装(`./configure && make && sudo make insatll`)；
- 由于低版本的Linux系统内核版本比较低，bosfs进程在运行过程中容易出现掉线或者其他问题。
  因此建议用户将操作系统升级到CentOS 7.0或者Ubuntu 14.04以及以上版本，同时bosfs升级到1.0.0.12及其以上版本。

Ubuntu系统安装

1. 安装bosfs所需要的依赖

```
 sudo apt-get -y install build-essential
sudo apt-get install autoconf
sudo apt-get install uuid-dev
sudo apt-get install libssl-dev
sudo apt-get install libcurl4-openssl-dev

如果系统fuse安装不成功, 或系统版本低于2.9.4
则需要源码安装fuse, 下载libfuse源码包后进行安装
tar -xzvf fuse-2.9.4.tar.gz
cd fuse-2.9.4
./configure
sudo make && sudo make install
```

> **注意**:如果您的libstdc++版本过低, bosfs安装成功后会报错如下：
>
> ```
>  bosfs: /usr/lib64/libstdc++.so.6: version `GLIBCXX_3.4.14' not found (required by bosfs)
> bosfs: /usr/lib64/libstdc++.so.6: version `GLIBCXX_3.4.15' not found (required by bosfs)
> ```
>
> 建议手动升级相关libstdc++, 或更新gcc/g++版本, 需支持-std=c++11

2. 下载bosfs源码包

3. 编译并安装

```
 tar -xzvf bosfs-1.0.0.13.1.tar.gz
cd bosfs-1.0.0.13.1
sudo sh build.sh
```

> **注意：**
> (1) 如果编译的时候显示'rpl_malloc' was not declared in this scope,这是因为您的系统malloc与bosfs冲突， 解决办法将编译目录下configure.ac文件的AC_FUNC_MALLOC注释掉。
> (2) 若编译相关依赖有问题, 可sh -x查看具体编译流程, 应修改build.sh进行调整

4. 安装完成后，可以使用`bosfs -v`命令查看版本

> **注意**:
> (1) 如果您的fuse版本过低， 建议使用官网提供的libfuse。
> (2) 挂载的时候显示libfuse.so.2 ：can not open Shared object file：No such file or directory
> 原因是libfuse.so生成路径不在动态链接路径, 解决办法：
>
> 1. echo /usr/local/lib >> /etc/ld.so.conf, 将/usr/local/lib添加到链接路径
>
> 2. ldconfig, 重新载入系统

Centos系统安装

由于BOS FS对环境有依赖，所以安装源代码包前需要安装依赖。以百度智能云虚机CentOS 6.5为例：

1. 安装bosfs所需要的依赖

```
 sudo yum install gcc-c++
sudo yum install autoconf
sudo yum install automake
sudo yum install libuuid-devel
sudo yum install openssl-devel
sudo yum install libcurl-devel

如果系统fuse安装不成功, 或系统版本低于2.9.4
则需要源码安装fuse, 下载libfuse源码包后进行安装
tar -xzvf fuse-2.9.4.tar.gz
cd fuse-2.9.4
./configure
sudo make && sudo make install
```

> **注意**:如果您的libstdc++版本过低, bosfs安装成功后会报错如下：
>
> ```
>  bosfs: /usr/lib64/libstdc++.so.6: version `GLIBCXX_3.4.14' not found (required by bosfs)
> bosfs: /usr/lib64/libstdc++.so.6: version `GLIBCXX_3.4.15' not found (required by bosfs)
> ```
>
> 建议手动升级相关libstdc++, 或更新gcc/g++版本, 需支持-std=c++11

2. 下载bosfs源码包

3. 编译并安装

(1) 解压BOS FS源码包
(2) 编译源代码：

- 如果选择默认编译：直接执行`build.sh`；
- 如果选择自定义编译：先进入cppsdk目录，按自定义参数执行`configure`，`make`，`make install`；然后再进入bosfs目录，按自定义参数执行`configure`，`make`，`make install`。
- 最后获取二进制可执行文件bosfs

> **注意：**
> (1) 如果编译的时候显示'rpl_malloc' was not declared in this scope,这是因为您的系统malloc与bosfs冲突， 解决办法将编译目录下configure.ac文件的AC_FUNC_MALLOC注释掉。
> (2) 若编译相关依赖有问题, 可sh -x查看具体编译流程, 应修改build.sh进行调整

4. 安装完成后，可以使用`bosfs -v`命令查看版本

> **注意**:
> (1) 如果您的fuse版本过低， 建议使用官网提供的libfuse。
> (2) 挂载的时候显示libfuse.so.2 ：can not open Shared object file：No such file or directory
> 原因是libfuse.so生成路径不在动态链接路径, 解决办法：
>
> 1. echo /usr/local/lib >> /etc/ld.so.conf, 将/usr/local/lib添加到链接路径
> 2. ldconfig, 重新载入系统

配置

参数设置

bosfs支持设定参数来配置用户权限、存储类型及缓存等。执行`bosfs –h`可以获得bosfs参数设置的详细说明：

bosfs参数	配置作用	注意事项
-o use_cache, -o bos.fs.cache.base=< your_cache_dir_path >	指定文件缓存模式下cache file位置	该位置不能在挂载点内，且必须是绝对路径, 被缓存的文件会长期存储在该目录下
-o allow_other -o mount_umask=0000	允许其他用户访问挂载点,并通过mount_umask禁止某些权限,mount_umask与linux umask含义一致	默认情况下，bosfs只具有挂载用户的读、写、执行权限即S_IRWXU
-o meta_expires=< integer seconds>	指定meta缓存过期时间	默认为不过期, 多端同时使用/挂载同一bucket时,建议配置-o meta_expires=0, 保证元数据和bos服务端一致, 详情见下文中 配置实践
-o meta_capacity=< integer number>	指定内存中meta缓存容量，超过容量后过期的缓存会被以lru方式清除	默认 100000, 注意使用meta cache后, 多挂载点读写同一个bucket数据时, bosfs中文件元信息可能出现与bos服务端不一致情况
-o meta_strategy=<remote,merge,local>	指定meta读取策略 remote:从远端读 merge:从远端读且于本地合并 local:只读本地缓存	该参数需于meta_expires配置使用
-o storage_class=< standard or standard_ia>	指定object存储类型，包括标准存储和低频存储	默认为标准存储
-o tmpdir=<your_tmp_dir>	指定file模式下临时文件位置	该位置不能在挂载点内，且必须是绝对路径，临时文件非长期存储在磁盘，close file后临时文件会被自动清除, 但需保证该目录磁盘空间充足, bosfs默认使用该模式
-o multipart_size=<integer KB,MB>	指定并发传输的分片大小，可用KB或MB单位	默认为10MB
-o multipart_threshold=<integer KB,MB>	指定启用三步上传的文件大小阈值, 超过该阈值则开启	默认为100MB
-o multipart_parallel=<integer number'>	限制大文件上传并发度	可根据网络带宽等因素合理配置, 一般配置为10
-o preload_blocks=<integer number,MB>	指定预读blocks数, 单次预读数据为: preload_blocks*1MB	默认0, 如进行大文件下载(顺序读), 建议配置, 若进行随机读, 则不建议配置
-o connect_timeout=< milliseconds>	指定连接Bos server的超时时间	默认1000ms
-o client_timeout=< milliseconds>	指定数据传输的超时时间	默认1200000s, 处于弱网/公网状态建议进行重新配置, 超时后会进行重试, 根据网络环境配置.
-o rw_strategy=<mem_sequence,mem_random>	指定读写策略, mem_sequence:使用内存模式, 只进行顺序读写(适用于只上传下载场景); mem_random:使用内存模式, 进行随机读写	mem_sequence:只能进行顺序读写, 使用于上传/下载大文件(size>10GB), 可提升性能且减少/tmp目录下临时文件的空间使用.
-o mem_cache_per_size=< integer MB>	配置rw_strategy=mem_sequence/mem_random前提下, 指定单个文件的内存cache大小	配置rw_strategy=mem_random时, 需注意, mem_cache_per_size值必须大于所有可能打开文件大小最大值.
-o mem_cache_limit_size=< integer MB>	配置rw_strategy=mem_sequence/mem_random前提下, 指定bosfs进程所能使用当做文件缓存的总内存阈值, 超过该阈值, 则自动转换成普通file模式	bosfs进程所能使用当做文件缓存的总内存阈值=同时open文件数据 *mem_cache_per_size, 默认值为启动bosfs时系统free内存大小 * 80%
-o logfile=</tmp/bosfs.log> -o loglevel=debug	开启bosfs日志,可配置日志位置在/tmp/bosfs.log下	默认1GB进行切割, 测试环境下建议开启debug日志
-o credentials=/path/credentials	配置通过/path/credentials文件读取ak/sk等信息, 使用sts权限时必须使用此方式	具体格式: [defaults] bce_access_key_id = xxx bce_secret_access_key = xxx bce_security_token = xxx reload_interval_s=60

fuse参数	配置作用	注意事项
-o ro	设置为只读模式, 用于保护bucket不被写入	该选项是fuse的标准选项
-o direct_io	设置为direct_io, 不使用操作系统page cache	配置rw_strategy=mem_sequence/mem_random前提下可设置为direct_io, 该选项是fuse的标准选项

日志

日志默认输出到stderr，可以用`-o logfile=xxxx/xxxx.log`来指定名称。使用命令`-d`打开debug模式的情况下，可以重定向到本地文件。

> **注意：**
>
> - 切勿将logfile文件路径或者重定向的文件路径指定到mountpoint里面，这会导致持续地将日志刷入BOS，造成不必要的开销。
> - 如果没有使用`-d`或者`-f`指定前台运行模式，启动之后bosfs进程会变成daemon，同时会将工作目录切到/之下，因此logfile需要指定绝对路径。bosfs的日志文件不会自动切割，需要用户自行使用脚本管理。
> - BOS FS挂载后，如果使用rsync功能，建议设置`-o loglevel=ERROR`和`--partitial-dir=本地路径`，因为rsync会产生临时文件，默认会存在目的文件所在位置（bosfs中），产生不必要的bosfs中的上传开销。

只读

可以使用`-o ro`选项来设置只读模式，用于保护bucket不被写入，该选项是fuse的标准选项。

配置实践

0. 一般场景下(仅用作上传或下载文件场景, 无或很少随机读写)的推荐挂载参数:

```
 bosfs <bos-bucket> <local-mount-dir> -o endpoint=bj.bcebos.com -o ak=xxx -o sk=xxx -o logfile=/tmp/bosfs.log -o loglevel=debug -o meta_expires=0 -o preload_blocks=100 -o
bos.sdk.multipart_threshold=100MB -o bos.sdk.multipart_size=10MB -o bos.fs.multipart_parallel=10
```

1. 元数据缓存 meta_expires 配置项
   该配置项为bosfs元数据信息(文件size, last_modify_time, acl等)缓存时间, 使用元数据缓存可获得更高的性能, 但无法保证与bos服务端数据一致性, 一般有以下情况:
   不配置-o meta_expires该项, 默认为使用元数据缓存, 且无过期时间, 当缓存条数达到meta_capacity后, 进行lru淘汰
   -o meta_expires=0 表示不使用元数据缓存, 即每次都从bos服务端拉取元数据信息, 相应的会降低性能
   -o meta_expires=3600 表示元数据缓存为3600秒, 该过期时间内使用本地缓存的元数据信息, 过期后从bos服务端重新拉取元数据信息
   (1) 在有多个挂载点/sdk/其他途径对同一bucket中数据进行更新, 当前挂载点读数据时, 配置-o meta_expires=0可保证此挂载点始终能拉取到bos服务端的最新数据.
   (2) 在只使用此挂载点来读写bucket中数据, 无其他挂载点读写同一bucket时, 可不配置该项, 以获得更高性能.
   (3) 在只使用bosfs挂载点进行上传/写数据, 不进行下载/读数据时, 可不配置该项, 以获得更高性能.

2. 下载/上传文件(顺序读/写)场景下, 可参考以下配置, 提升性能.
   (1)开启100MB预读, 当文件size超过100MB开启分片上传, 分片size为10MB, 并行度为10.
   (2)下载场景下, 当文件size>10MB, bosfs会默认开启并行下载.
   (3) 注意, 随机读写下开启预读会导致性能下降.
   -o meta_expires=0 -o preload_blocks=100 -o bos.sdk.multipart_threshold=100MB -o bos.sdk.multipart_size=10MB -o bos.fs.multipart_parallel=10

3. 在一次下载, 多次读取的场景下, 可开启file缓存模式, 参考以下配置, 提升性能.
   开启file缓存模式后(默认为不开启file缓存模式), bosfs会在本机器的bos.fs.cache.base目录下缓存通过bosfs读到的文件, 下次读取相同文件, 会直接读取本地文件, 而非从bos服务端重新拉取, 以此提升性能.
   (1) 开启file缓存模式, 并配置bosfs缓存目录为/tmp/bosfs-cache
   (2) 开启100MB预读, 当文件size超过100MB开启分片上传, 分片size为10MB, 并行度为10.
   (3) 下载场景下, 当文件size>10MB, bosfs会默认开启并行下载.
   -o meta_expires=0 -o use_cache -o bos.fs.cache.base=/tmp/bosfs-cache -o preload_blocks=100 -o bos.sdk.multipart_threshold=100MB -o bos.sdk.multipart_size=10MB -o bos.fs.multipart_parallel=10

4. 只用做上传/下载大文件(文件size>10GB)场景下, 可开启内存模式, 参考以下配置, 提升整性能.
   (1) 开启内存模式, 配置单个文件内存cache为20MB, bosfs进程所能使用总的cache size为1000MB
   ① 顺序读写场景下, 即配置rw_strategy=mem_sequence, 当bosfs进程使用cache size内存超过配置阈值时, 自动退化为locla file模式
   (2) 开启20MB预读, 指定上传文件size超过100MB开启分片并发上传, 分片size为10MB, 并行度为10.
   (3) 下载场景下, 当文件size>10MB, bosfs会默认开启并行下载.
   -o meta_expires=0 -o preload_blocks=20 -o bos.sdk.multipart_threshold=100MB -o bos.sdk.multipart_size=10MB -o bos.fs.multipart_parallel=10 -o mem_cache_per_size=20 -o rw_strategy=mem_sequence -o
   mem_cache_limit_size=1000 -o direct_io

操作与实践

挂载

BOS FS挂载bucket到本地目录执行命令：

```
bosfs mybucket my_local_directory -o endpoint=http://bj.bcebos.com -o ak=xxxxxxxxxxxxxxx -o sk=xxxxxxxxxxxxxxxx -o logfile=xx/xx.log
```

执行之后后台会启动一个名为bosfs的daemon进程，如果挂载成功，可以用df命令看挂载点。此外可以查看/etc/fstab，来了解具体使用到的挂载选项。

同时bosfs支持挂载子目录，使用方式将bucket名变成bucket/subdir即可实现挂载子目录,即可实现将子目录作为挂载点的根目录。

> **说明：** 默认只有root用户有权限执行fusermount,如果需要特定用户执行fusermount，则要通过命令usermod -a -G fuse YourAccount把这个用户加入到安装fuse后获得的用户组中。

卸载

1. 使用umount命令可以直接卸载。
2. 对于非root用户且没有卸载权限的，如果加入到了fuse用户组中，可以使用fusermount -u my_local_directory进行卸载。
3. 还可以通过杀掉后台的bosfs进程来卸载，不过一般不建议这么使用，会造成未FLUSH文件的损坏。

操作示例

以同步两个Bucket的数据为例，讲解BOS FS的使用。

假设有bucket-a和bucket-b两个bucket，且这两个bucket可能位于不同的Region（比如北京，广州）。业务上需要将bucket-a的数据备份到bucket-b，且只同步有变更的文件，以减小开销。可以按照如下步骤实现：

1. 找到一台可用的Centos镜像的虚机（没有则可以购买一台）并开通EIP服务；
2. 安装bosfs；
3. 使用bosfs挂载命令将bucket-a和bucket-b挂载到本地的bucket-a，bucket-b两个目录上；
4. 使用命令rsync -av --partial-dir=/tmp bucket-a bucket-b开始同步这两个bucket的数据。
5. 可以考虑将rsync命令加入crontab，用来定期执行。

> **注意事项:**
>
> - 同步一个文件的时候，会在/tmp目录或者指定的cache目录占用这个文件大小的空间，同步完这个文件的时候将释放。因此需要保证本地有足够的磁盘空间。
> - 使用rsync命令时加上--partial-dir=/tmp选项, 把临时文件放到tmp目录下, 避免临时文件在挂载点内产生不必要的数据传输。
> - 如果使用一个rsync进程同步速度可能会比较慢，如果使用另外一台机器同步速度也会受单机网卡的影响。所以如果同步的bucket本身目录结构可以进行切分，则可以考虑多个rsync进程并行，或者多部署几套虚机并行同步不同的子目录。
> - 如果同时在多个mount点上对同一个文件进行操作，BOS无法保证一个可预期的结果，可能会造成文件截断、写入丢失，或刚写入的无法在另一个挂载点上读出等问题，所以需要避免。
> - 本示例因为两个Bucket位于不同Region，所以需要开通EIP。只要两个Bucket及BCC有任意一个跨区域则需要开通EIP服务，只有三者均处于同一Region才不需开通EIP。

实践

1. **关于bosfs性能与注意事项:**
   - bosfs基于fuse开发和进行挂载, 由于其框架接口的语义限制, 对挂载点内的本地文件的操作可能需要与BOS对象存储服务进行多次网络交互, 因此bosfs挂载点的实例与BOS endpoint之间的网络环境会影响bosfs的性能.
   - bosfs提供了大量参数, 可根据自身使用的场景进行参数调优, 可参考文档
   - bosfs操作并非原子的, 尤其是如使用多个bosfs进程挂载同一个BOS bubket, 多个挂载点中对该bucket的并发操作可能出现未定义的行为.
   - 若bosfs无法满足业务需求时, 建议使用boscmd, 非线上敏感业务可以使用bosfs，若要求可靠性、稳定性, 以及可控性, 建议使用API接入.
2. **docker/kubernetes环境下使用bosfs:**
   - 在k8s环境中, 可使用静态PV/PVC的方式通过bosfs挂载来访问BOS.
   - 建议将业务pod和bosfs pod分开管理, 可使用保活探针机制, 当bosfs异常后重新拉起.
   - 当单个业务pod中使用大量bosfs挂载点或者k8s中拉起多个业务pod时, 如果多个bosfs容器部署在同一node, 可能会占用大量内存和网络带宽, 建议根据实际情况进行调整.
   - 百度智能云容器引擎CCE 已集成了容器中通过bosfs挂载的方式访问BOS的组件, 可参考文档.

常见问题

常见问题排查

1. **使用 bosfs 过程中，突然显示 "unable to access MOUNTPOINT /path/to/mountpoint: Transport endpoint is not connected"，并且无法再访问.**
   - 首先, 您可以使用 ps ax | grep bosfs 命令查看 bosfs 进程是否存在, 如果 bosfs 进程是由于误操作而挂掉，您可以执行如下命令进行重新挂载：

```
fusermount -uz /path/to/mnt_dir
bosfs bucket /mnt/bosfs xxxx
```

   - 如果 bosfs 进程不是由于误操作挂掉，可以检查机器上的 fuse 版本是否低于 2.9.4，libfuse 在低于 2.9.4 版本的情况下可能会导致 bosfs 进程异常退出。此时，建议您按照本文 编译和安装 bosfs 部分更新 fuse 版本或安装最新版本的 bosfs。
   - bosfs 在读写时会占用磁盘写大量的 temp cache(默认在/tmp目录下, 可使用-o tmpdir=/mnt/tmp-path来指定文件目录)，可能会导致磁盘可用空间不足，需要经常清理
   - 可能的原因是listbucket、listobject 申请内存过多，触发了系统的 oom
   - 仅用作数据下载/上传场景, 建议使用 boscmd 替代 bosfs ，非线上敏感业务可以使用 bosfs ，要求可靠性、稳定性的建议使用 bcecmd
2. **在使用过程中出现长时间卡住的情况.**
   - 检查是否是磁盘io负载较高引起, 建议降低并发度， 挂载时指定-o multipart_parallel=5（默认为10）
   - 检查机器到bos endpoint的网络环境, 若处于公网或网络不稳定场景下, 可配置-o client_timeout=5000, 使得网络传输达到配置时间后超时, 进行重试, 超时时间可根据具体网络环境设定.
   - 可开启日志查看其他具体原因.
3. **使用多个bosfs进程挂载相同bucket, 文件元信息不一致问题**
   - 在使用meta cache的情况下可能导致不同机器的元数据不一致(长度、mod权限等)， 建议挂载时指定-o meta_expires=0来避免从cache读取meta数据, 但可能会增加ls/stat等指令的耗时
4. **上传大文件时出现"there is no enough disk space for used as cache(or temporary)"报错**
   - bosfs在上传,下载中会使用/tmp目录存放临时文件, 若该目录下磁盘空间不足, 会报错
   - /tmp目录下磁盘可用空间大小 < 文件大小, 需增加/tmp下空间大小.
   - 检查multipart_size和multipart_parallel配置, 是否配错, 确保/tmp/下空间大小 > multipart_size * multipart_parallel
   - 如果是只有上传,下载(顺序读写)的场景, 也可使用-o rw_strategy=mem_sequence -o mem_cache_per_size=20MB 配置开启memory cache模式, 将不再使用/tmp/目录.
5. **使用过程中出现403报错**
   - 403报错一般是权限相关问题引起, 建议查看ak和权限配置.

版本变更记录

- BOS FS工具包 [2023-07-27] 版本号1.0.0.13.3
  - 修复已知问题
- BOS FS工具包 [2022-12-16] 版本号1.0.0.13.2
  - 支持层级meta特性
- BOS FS工具包 [2022-04-06] 版本号1.0.0.12
  - 元数据缓存过期时间支持毫秒级, 优化gc策略以及同名更新流程.
  - 修复bug若干
- BOS FS工具包 [2021-09-03] 版本号1.0.0.11
  - 支持配置memory cache模式读写策略, 提升读写性能
  - 修复bug若干
- BOS FS工具包 [2019-12-03] 版本号1.0.0.10
  - 修复开机自动挂载失败
  - 支持挂载子目录跳过bucket权限校验
- BOS FS工具包 [2019-11-11] 版本号1.0.0.9
  - 支持文件名大于256字节
  - 更新SDK工具包
  - 发布新的libfuse安装包

- BOS FS工具包 [2018-08-27] 版本号1.0.0.8
  - 支持子目录挂载
  - 限制大文件并发度
- BOS FS工具包 [2018-01-30] 版本号1.0.0.7
  - 元信息缓存优化
  - 目录元信息获取优化
  - 依赖SDK版本更新
- BOS FS工具包 [2017-08-01] 版本号1.0.0.6
  - 判断单个文件存在的多级探测改为并发执行；
  - curl开启DNS cache；
  - 删减部分日志；
  - 优化多步上传消耗的fd数量；
  - 配置更新，支持三步上传大小，临时文件目录；
  - 日志滚动切割。
- BOS FS工具包 [2017-05-18] 版本号1.0.0.5

  修复自动加载credential出现的coredump或者failure的问题。
- BOS FS工具包 [2017-04-13] 版本号1.0.0.4
  - 支持mv目录；
  - 支持扩展属性；
  - 上传文件的性能优化；
  - user_meta调整为bosfs-的前缀；
  - 修正ubuntu上的编译链接问题；
  - 支持sts、credential配置；
  - 支持credential定时reload和信号。
- BOS FS工具包 [2017-01-03] 版本号1.0.0.3

  第一次发布。

BOS-Util批量操作工具

概述

概述

BOS-Util是以命令行方式管理BOS数据的工具，基于Java编写，提供对BOS端数据的批量操作，主要包含批量设置metadata、批量删除文件、批量重命名文件、批量设置object访问权限、修改prefix(目录)、计算给定prefix(目录)下所有文件大小的总和等功能。

安装BOS-Util

前提条件

该工具以jar包形式发布，需要有java运行环境，且版本在1.8及以上，可以使用命令javac -version检测，若显示java版本为1.8以上即可，否则请安装或升级java环境。

下载地址

下载地址: bosutil-0.1.7.zip

MD5码： 38dfb09f6ff3e6ff1854392d2cc21b0a

安装BOS-Util

该工具以jar包形式发布，作为java命令行的"-jar"参数使用，不需要任何安装过程也无任何外部依赖。

通过BOS-Util使用BOS服务

命令格式

该工具的通用格式如下:

```
java -jar bosutil.jar -endpoint {endpoint} -ak {access_key} -sk {secret_key}
  -cmd {command} [-bucketName {bucket_name}] [-other_options {option_value},...]
```

上述格式中使用"{}"的字段为必需参数，对应参数值应使用双引号包含，如:`-endpoint "http://bj.bcebos.com"`。 使用"[]"的字段为可选参数，具体由cmd参数决定。对于"-cmd"这个选项，只能从以下支持的命令中选取一个执行：

命令	作用
LIST_OBJECTS	列举存在bos上的object
SET_CONTENT_TYPE_BY_FILE_EXTENSION	根据object的后缀名设置content-type
GET_OBJECTS_META	批量获取object的meta
DELETE_OBJECTS	批量删除object
SET_OBJECTS_META	设置object的meta
SET_OBJECTS_ACL	设置object的acl
TEST_CONTENT_TYPE	查看某种类型在本工具中对应的content-type
RENAME_OBJECTS	批量重命名object
CHANGE_PREFIX	批量修改object的prefix
COMPUTE_SIZE	计算某个bucket和prefix的标准存储容量，不包含碎片容量
TEST_SPEED	测试上传下载速度

> 说明：以下示例均为Linux/Unix环境下，如果在windows下使用需要注意:
>
> - 命令折行使用^
> - 对"的使用: windows下不能使用'括起参数，必须使用"，如果参数中包含"，需要使用"进行转义。

公共选项

下表列举了命令行支持的公共选项，这四个选项为命令格式中给出的必需选项，其它选项依据指定cmd选择设置。

选项	描述	是否有参数值	默认值	示例
-ak	access key，如何获取AK/SK，请参见如何获取AK/SK	yes	-	-
-sk	secret key，如何获取AK/SK，请参见如何获取AK/SK	yes	-	-
-endpoint	BOS endpoint，关于BOS endpoint的详细描述请参见BOS访问域名	yes	-	http://bj.bcebos.com
-cmd	当前将要执行的命令	yes	-	SET_OBJECTS_META

LIST_OBJECTS命令

功能

递归列举用户给定bucket下的所有objects信息到文件。

语法

```
java -jar bosutil.jar -endpoint {endpoint} -ak {access_key} -sk {secret_key}
  -cmd LIST_OBJECTS -bucketName {bucket_name}
```

选项

必需的ak、sk、endpoint、cmd参数，以及列举的bucket的名称参数bucketName。

选项	描述	示例	是否必选
bucketName	bucket 名称	"bucket1"	是

其他额外参数若指定会忽略。

详细说明

该命令会列举指定bucket下所有objects列表到一个本地文件中，该文件保存在java运行时的同一目录下，以给出的bucketName与"_object_list"组合作为文件名称。如给出的bucketName为"mybucekt"，当前java运行目录为"/home/user1"，则会生成一个"/home/user1/mybucket_object_list"的文件，该文件每一行为一个object的信息，分别为bucket名称、object全称（包含prefix）、object大小、object的最后修改时间以及obejct的etag，输出文件各列按照\t(tab)分隔，输出文件的格式示例如下：

```
 BucketName\tObjectKey\tSize\tLastModified\tEtag\n
... ... ... ...
```

使用示例

```
java -jar bosutil.jar -endpoint "http://bj.bcebos.com" -ak "your_access_key" -sk "your_secret_key" \
  -cmd LIST_OBJECTS -bucketName "bucket"
```

SET_CONTENT_TYPE_BY_FILE_EXTENSION命令

功能

该命令设置给定object的contentType信息。

语法

```
  java -jar bosutil.jar -endpoint {endpoint} -ak {access_key} -sk {secret_key}
    -cmd SET_CONTENT_TYPE_BY_FILE_EXTENSION
    [-objectList {objectListFilePath}] |
    [-bucketName {bucket_name}
    -objectName {object_name}
    -prefix {prefix}
    -marker {marker}
    -contentType {content-type}
    -defaultContentType {default content-type}
```

选项

除了必需的ak、sk、endpoint、cmd参数外，其它参数参见下表：

选项	描述	示例	是否必选
bucketName	bucket名称	"bucket1"	是
objectName	object名称	"prefix1/object1"	否
prefix	待设置object的prefix	"prefix1"	否
marker	待设置object的marker	""	否
objectList	包含bucket与object名称列表的文件路径，文件中具体内容依赖使用的命令，文件编码必需为utf8，每行的格式为"BucketName\tObjectName"（其中\t代表tab制表符）。	"list-file-name"	否
contentType	用户指定的Content-Type metadata信息	".txt:text/plain;.flv:video/flv"	否
defaultContentType	依据后缀查找的Content-Type在失败时使用的默认值	"application/otect-stream"	否

详细说明

该命令依据AK/SK信息，在指定的endpoint下，为**选项**中指定的Object设置content-type信息。

该工具内置了一个常见后缀名与Content-Type的字典。对于待设置Content-Type的object，Util工具会根据用户指定的contentType参数、内置字典和defaultContentType参数，依据如下规则决定其Content-Type：

1. 先获得该object的后缀名，和命令行中的contentType参数进行匹配，若精确指定了就使用指定的contentType参数。如某个object的后缀名为".js"，而contentType参数中存在一项为".js:text/javascript"则精确指定了。
2. 若命令行中没有精确指定contentType参数，就判断contentType中是否有模糊指定".*:text/plain"，若有则使用之；若模糊指定没有，或者没有给出contentType参数，就前往内置字典获取该object的后缀名对应的Content-Type。若获取成功就使用内置的Content-Type进行设置；
3. 若内置字典也没有匹配到，则按照命令行中defaultContentType参数指定的值进行设置。若命令行中没有设置defaultContentType参数，则直接返回，对该object不作任何设置，并将该object打印到命令行。

该命令提供两种方式指定待设置Content-Type的object。

- -objectList方式: 即提供一个object列表文件，各列使用\t(tab)分隔，每行的格式为"BucketName\tObjectName"（其中\t代表tab制表符）。
- -bucketName -objectName -prefix -marker方式: 即使用bucketName、prefix、marker、objectName进行匹配。该方式必须提供bucketName参数指定bucket，其余参数支持以objectName进行精确指定的方式进行，同时支持使用prefix、marker的方式进行模糊匹配。其中marker的使用方式是:用户初次执行只需要输入prefix, 如果执行到一半失败了, 不必重新开始执行, 只需要找到当前目录下last_marker+${bucket_name}.txt这个文件, 拿到最后一行(格式为<object名字 已完成任务数量>), 将object名字作为marker的值再次执行命令则可继续上次未完成的批量操作任务。

使用示例

```
java -jar bosutil.jar -endpoint "http://bj.bcebos.com" -ak "your_access_key" -sk "your_secret_key" \
  -cmd SET_CONTENT_TYPE_BY_FILE_EXTENSION \
  -defaultContentType "text/plain" \
  -contentType ".js:text/javascript;.flv:video/flv" \
  -objectList "./a/filelist"
```

```
java -jar bosutil.jar -endpoint "http://bj.bcebos.com" -ak "your_access_key" -sk "your_secret_key" \
  -cmd SET_CONTENT_TYPE_BY_FILE_EXTENSION \
  -defaultContentType "text/plain" \
  -contentType ".js:text/javascript;.flv:video/flv" \
  -bucketName "abc" \
  -objectName "test-object"
```

SET_OBJECTS_ACL

功能

该命令设置给定object的acl信息

语法

```
java -jar bosutil.jar -endpoint "http://bj.bcebos.com" -ak "your_access_key" -sk "your_secret_key"
  -cmd SET_OBJECTS_ACL
  -bucketName {bucket_name}
  [-objectName {object_name}] |
  [-prefix {prefix}]
  [-marker {marker}]
  [-canned {canned acl}] |
  [-aclConfigFile {acl_config_file}]
  [-threadNumber {concurrent number}]
```

选项

除了必须的ak、sk、endpoint、cmd参数外，其他参数见下表：

选项	描述	示例	是否必选
bucketName	bucket名称	"bucket1"	是
objectName	object名称	"prefix1/object1"	否
prefix	待设置object的prefix	"prefix1"	否
marker	待设置object的market	""	否
canned	通过使用头域的"x-bce-acl"字段来设置object访问权限，支持：PRIVATE、PUBLIC-READ	"PRIVATE"	否
aclConfigFile	提供一个ACL文件，文件格式参见ACL文件格式，目前ACL文件只支持accessControlList，grantee，permission字段	"example.txt"	否
threadNumber	并发数，默认值是1	10	否

详细说明

该命令依据AK/SK信息，在指定的endpoint下，为选项中指定的Object设置acl信息。

该工具支持两种方式为object设置acl信息：

1. 通过canned参数，指定object的访问权限，支持PRIVATE和PUBLIC-READ；

2. 通过aclConfigFile参数，提供一个ACL文件，文件内容为JSON格式，具体如下：

```
{
  "accessControlList": [
    {
      "grantee": [
        {
          "id": "e13b12d0131b4c8bae959df4969387b8"
        }
      ],
      "permission": [
        "READ"
      ]
    }
  ]
}
```

该命令提供指定待设置acl的object方式如下：

- -bucketName -objectName -prefix -marker方式: 即使用bucketName、prefix、marker、objectName进行匹配。该方式必须提供bucketName参数指定bucket，其余参数支持以objectName进行精确指定的方式进行，同时支持使用prefix、marker的方式进行模糊匹配。其中marker的使用方式是:用户初次执行只需要输入prefix, 如果执行到一半失败了，不必重新开始执行，只需要找到当前目录下last_marker+${bucket_name}.txt这个文件，拿到最后一行(格式为<object名字 已完成任务数量>)，将object名字作为marker的值再次执行命令则可继续上次未完成的批量操作任务。

使用示例

```
java -jar bosutil.jar -endpoint "http://bj.bcebos.com" -ak "your_access_key" -sk "your_secret_key" \
  -cmd SET_OBJECTS_ACL \
  -bucketName "bucket1" \
  -objectNmae "prefix1/object1" \
  -canned "PRIVATE"
```

```
java -jar bosutil.jar -endpoint "http://bj.bcebos.com" -ak "your_access_key" -sk "your_secret_key" \
  -cmd SET_OBJECTS_ACL \
  -bucketName "bucket1" \
  -prefix "prefix" \
  -marker "" \
  -canned "PRIVATE"
```

```
java -jar bosutil.jar -endpoint "http://bj.bcebos.com" -ak "your_access_key" -sk "your_secret_key" \
  -cmd SET_OBJECTS_ACL \
  -bucketName "bucket1" \
  -prefix "prefix" \
  -marker "" \
  -aclConfigFile "example.txt"
```

TEST_CONTENT_TYPE命令

功能

Util工具内置了一个常见后缀名与Content-Type的字典，可以通过TEST_CONTENT_TYPE命令测试字典里所预置的Content-Type。

语法

```
java -jar bosutil.jar \
  -cmd TEST_CONTENT_TYPE \
  -extension {extension-value}
```

选项

必需的参数为cmd以及extension参数。

选项	描述	示例	是否必选
extension	文件扩展名	.txt	是

其他额外参数若指定会忽略。

详细说明

该命令仅用来进行获取指定后缀名的Content-Type信息，因此不需要与bos端交互，无需指定AK/SK以及endpoint信息，只需提供待测试的后缀名参数，然后根据内部支持的Content-Type设置规则返回相应信息到命令行，对于不支持的后缀名返回不支持的提示信息。

使用示例

```
java -jar bosutil.jar -cmd TEST_CONTENT_TYPE -extension ".mp4"
```

GET_OBJECTS_META命令

功能

依据用户指定的object获取方式获取这些object的metadata信息。

语法

```
java -jar bosutil.jar -endpoint {endpoint} -ak {access_key} -sk {secret_key}
  -cmd GET_OBJECTS_META
  [-objectList {objectListFilePath}] |
  [-bucketName {bucket_name}
  -objectName {object_name}
  -prefix {prefix}
  -marker {marker}
```

选项

除了必需的ak、sk、endpoint、cmd参数外，其它参数参见下表：

选项	描述	示例	是否必选
bucketName	bucket名称	"bucket1"	是
objectName	object名称	"prefix1/object1"	否
prefix	object的prefix	"prefix1"	否
marker	object的marker	""	否
objectList	包含bucket与object名称列表的文件路径，文件中具体内容依赖使用的命令，文件编码必需为utf8，每行的格式为"BucketName\tObjectName"（其中\t代表tab制表符）。	"list-file-name"	否

详细说明

该命令依据AK/SK信息，在指定的endpoint下，依据**选项**中说明的两种方式获取Object并获取这些object的metadata信息，每个object的metadata信息为一行输出到命令行显示。

支持如下两种方式获取object，从而进行metadata信息的获取：

- -objectList方式: 即提供一个object列表文件，各列使用\t(tab)分隔，各列使用\t(tab)分隔，每行的格式为"BucketName\ObjectName"（其中\t代表tab制表符）。
- -bucketName -objectName -prefix -marker方式: 即使用bucketName、prefix、marker、objectName进行匹配。该方式必须提供bucketName参数指定bucket，其余参数支持以objectName进行精确指定的方式进行，同时支持使用prefix、marker的方式进行模糊匹配。其中marker的使用方式是:用户初次执行只需要输入prefix, 如果执行到一半失败了, 不必重新开始执行, 只需要找到当前目录下last_marker+${bucket_name}.txt这个文件, 拿到最后一行(格式为<object名字 已完成任务数量>), 将object名字作为marker的值再次执行命令则可继续上次未完成的批量操作任务。

上述两种方式由用户选取一种即可，若同时指定，则会选取第一种从objectList参数中读取的方式，忽略第二种方式指定的参数。其他额外参数若指定会忽略。

使用示例

```
java -jar bosutil.jar -endpoint "http://bj.bcebos.com" -ak "your_access_key" -sk "your_secret_key" \
  -cmd GET_OBJECTS_META \
  -objectList "./a/filelist"
```

```
java -jar bosutil.jar -endpoint "http://bj.bcebos.com" -ak "your_access_key" -sk "your_secret_key" \
  -cmd GET_OBJECTS_META \
  -bucketName "abc" \
  -objectName "test-object"
```

SET_OBJECTS_META命令

功能

依据用户指定的object获取方式设置这些object的metadata信息。

语法

```
java -jar bosutil.jar -endpoint {endpoint} -ak {access_key} -sk {secret_key}
  -cmd SET_OBJECTS_META
  -bucketName {bucket_name}
  -objectName {object_name}
  -prefix {prefix}
  -marker {marker}
  -meta {metadata_string}
  -replace
  -threadNumber {concurrent_number}
```

选项

除了必需的ak、sk、endpoint、cmd参数外，其它参数参见下表：

选项	描述	示例	是否必选
bucketName	bucket名称	"bucket1"	是
objectName	object名称	"prefix1/object1"	否
prefix	待设置object的prefix，或修改的prefix	"prefix1"	否
marker	待设置object的marker	""	否
meta	object的meta信息	cacheControl:max-age=1800 或 cacheControl:|__user_meta_path:aa/bb|contentDisposition:attachment;filename="${object_basename}"	否
replace	设置meta信息时是否替换原有meta数据，取值为true和false	true	否
threadNumber	并发数, 默认值是1	10	否

必选参数meta和replace说明如下，其他额外参数若指定会忽略：

- meta: 以"key:value"方式给出metadata信息，多条信息以"|"连接，作为一个字符串参数提供。
  - meta value内容支持变量表达式:如**cacheControl:|__user_meta__path:aa/bb|contentDisposition:attachment;filename="${object_basename}"**，其中${object_basename}表示object的basename。在批量处理object的meta时，可以通过变量表达式设置所有Object的某一个meta字段，实现meta值不一样。
  - meta中用户可以自定义user meta, user meta的key必须以__user_meta__开头
- replace：该参数如果提供表示替换object原有的metadata信息，否则不进行替换（默认不替换）。不替换即将原先的meta和新的meta合并，如果新meta和原meta内容有重复则用新meta。

详细说明

该命令依据AK/SK信息，在指定的endpoint下，依据**选项**中说明的object获取方式，并根据meta和replace参数依次对获取的这些object的metadata信息进行设置。成功或错误信息会输出到命令行中。

使用如下方式获取需要设置metadata的objects：

-bucketName -objectName -prefix -marker方式: 即使用bucketName、prefix、marker、objectName进行匹配。该方式必须提供bucketName参数指定bucket，其余参数支持以objectName进行精确指定的方式进行，同

时支持使用prefix、marker的方式进行模糊匹配。其中marker的使用方式是:用户初次执行只需要输入prefix, 如果执行到一半失败了, 不必重新开始执行, 只需要找到当前目录下last_marker+${bucket_name}.txt这个文件, 拿到最后一行(格式为<object名字 已完成任务数量>), 将object名字作为marker的值再次执行命令则可继续上次未完成的批量操作任务。

使用示例

- linux/macOS

```
java -jar bosutil.jar -endpoint "http://bj.bcebos.com" -ak "your_access_key" -sk "your_secret_key" \
  -cmd SET_OBJECTS_META \
  -objectName "test-bueckt" \
  -prefix "test-" \
  -meta 'cacheControl:|\__user_meta__path:aa/bb|contentDisposition:attachment;filename="${object_basename}"' \
```

- windows：注意双引号`"`的转义

```shell
java -jar bosutil.jar -endpoint "http://bj.bcebos.com" -ak "your_access_key" -sk "your_secret_key" ^
 -cmd SET_OBJECTS_META ^
 -bucketName "test-bueckt" ^
 -prefix "test-" ^
 -meta "contentType:text/javascript|contentEncoding:gb18030|contentDisposition:attachment; filename=\"${object_basename}\"|__user_meta__path:a/b" ^
 -replace
```

DELETE_OBJECTS命令

功能

依据用户指定的object获取方式批量删除获取的Object。

> **注意：**该操作执行后不可恢复，请慎重操作。

语法 java -jar bosutil.jar -endpoint {endpoint} -ak {access_key} -sk {secret_key} -cmd DELETE_OBJECTS [-objectList {objectListFilePath}] | [-bucketName {bucket_name} -objectName {object_name} -prefix {prefix} -marker {marker} -include {include} -preview {preview}

选项

除了必需的ak、sk、endpoint、cmd参数外，其它参数参见下表：

选项	描述	示例	是否必选
bucketName	bucket 名称	"bucket1"	是
objectName	object 名称	"prefix1/object1"	否
prefix	列举object时的prefix，或修改的prefix	"prefix1"	否
marker	列举object时的marker	""	否
objectList	包含bucket与object名称列表的文件路径，文件中具体内容依赖使用的命令，文件编码必需为utf8，每行的格式为"BucketName\tObjectName"（其中\t代表tab制表符）。	"list-file-name"	否
include	删除指定bucket下符合正则表达式的object	"object.*jpeg"	否
preview	预览即将删除的object列表，可以配合include参数确定设置的正则表达式是否符合预期，不会实际删除object，默认值为true，执行实际删除命令时需要设置为false	true/false	否

详细说明

该命令依据AK/SK信息，在指定的endpoint下，批量删除**选项**中指定方式获取的Object，若有删除出错的object，会讲错误信息打印到命令行中。该命令在执行过程中若同时有object上传，且Object的prefix与正在删除的prefix匹配，此时无法保证新上传的object被删除。

使用include正则匹配删除object时，确保正则表达式是符合预期的情况下再执行命令，可以配合preview参数提前获取即将删除的object列表。

如果指定`objectList`，则只删除这个文件列举的所有文件，文件中各列按照`\t`(tab) 分隔，内容格式为

```
${bucketName}\t${objectName}
```

使用示例

```
java -jar bosutil.jar -endpoint "http://bj.bcebos.com" -ak "your_access_key" -sk "your_secret_key" \
  -cmd DELETE_OBJECTS \
  -objectList "./a/filelist"
```

```
java -jar bosutil.jar -endpoint "http://bj.bcebos.com" -ak "your_access_key" -sk "your_secret_key" \
  -cmd DELETE_OBJECTS \
  -bucketName "test-bucket" \
  -prefix "test-"
```

```
java -jar bosutil.jar -endpoint "http://bj.bcebos.com" -ak "your_access_key" -sk "your_secret_key" \
  -cmd DELETE_OBJECTS \
  -bucketName "test-bucket" \
  -prefix "test-"
  -include "test-.*jpeg"
  -preview "true"
```

RENAME_OBJECTS命令

功能

依据用户指定的object重命名信息的文件批量进行重命名操作。

语法

```
java -jar bosutil.jar -endpoint {endpoint} -ak {access_key} -sk {secret_key}
  -cmd RENAME_OBJECTS
  -objectList {objectListFilePath}
```

选项

除了必需的ak、sk、endpoint、cmd参数外，其它参数参见下表：

选项	描述	示例	是否必选
objectList	提供一个每一行都为一条重命名信息的文件的路径，各列使用\t(tab)分隔，每行的格式为：SrcBucketName\tSrcObjectName\tDstBucketName\tDstObjectName（其中\t代表tab制表符）。	"list-file-name"	是

详细说明

该命令依据AK/SK信息，在指定的endpoint下，对**选项**中指定的文件，批量操作该文件提供的每条重命名信息，进行重命名操作。该命令的操作分为两步，首先进行复制操作，再进行删除操作，因此如果有任何一步出错都会执行恢复操作进行回退，并输出出错信息。

该命令会自动读取文件并对每一行指定源object与新的名称进行批量重命名操作，处理的这个文件的进度信息保存在以"last_marker_"开头，后接objectList文件名的basename，再以".txt"为扩展名的文件中（如last_marker_objectlist.txt），保存了objectList行数与实际处理的object数目。其他额外参数若指定会忽略。

使用示例

```
java -jar bosutil.jar -endpoint "http://bj.bcebos.com" -ak "your_access_key" -sk "your_secret_key" \
  -cmd RENAME_OBJECTS \
  -objectList "./rename-file-list"
```

CHANGE_PREFIX命令

功能

依据用户指定的bucket下的prefix，对prefix进行修改。

语法

```
java -jar bosutil.jar -endpoint {endpoint} -ak {access_key} -sk {secret_key}
  -cmd CHANGE_PREFIX
  -bucketName {bucket_name}
  -prefix {original_preifx}
  -newPrefix {new_prefix}
```

选项

除了必需的ak、sk、endpoint、cmd参数外，其它参数参见下表：

选项	描述	示例	是否必选
bucketName	需要修改的bucket名称	"bucket1"	是
prefix	需要修改的prefix名称，类似需要修改的目录名称	"prefix1"	否
newPrefix	修改后的新的prefix名称，类似新文件夹名称	"prefix2"	否

其他额外参数若指定会忽略。

详细说明

该命令模拟文件夹方式进行目录重命名操作。依据AK/SK信息，在指定endpoint下的bucket中，找到preifx下所有obejct，依次对每个object构造新的prefix下的名称，批量进行重命名。每个重命名prefix的操作分为两步，首先进行复制操作，再进行删除操作，因此如果有任何一步出错都会执行恢复操作进行回退，并输出出错信息。在执行过程中若同时有object上传到相同prefix下不能保证新上传的Object被修改。

使用示例

```
java -jar bosutil.jar -endpoint "http://bj.bcebos.com" -ak "your_access_key" -sk "your_secret_key" \
  -cmd CHANGE_PREFIX \
  -bucketName "mybucket" \
  -prefix "test" \
  -newPrefix "demo"
```

COMPUTE_SIZE命令

功能

依据用户指定的bucket下的prefix，计算整个prefix下所有object的总大小。

语法

```
java -jar bosutil.jar -endpoint {endpoint} -ak {access_key} -sk {secret_key}
  -cmd COMPUTE_SIZE
  -bucketName {bucket_name}
  [-prefix {preifx}]
  [-friendly {friendly}]
  [-level {level}]
  [-threadNumber {concurrent_number}]
```

选项

除了必需的ak、sk、endpoint、cmd参数外，其它参数参见下表：

选项	描述	示例	是否必选
bucketName	操作的bucket名称	"bucket1"	是
prefix	需要计算大小prefix名称，类似需要计算大小的的目录名	"prefix1"	否
friendly	设置计算prefix的大小时是否以可读的友好方式输出，默认值为false	-	否
level	相对prefix的下级目录层级数，例如当设置level为2时，所有prefix/a/b/、prefix/a/c/目录的总个数，配合threadNumber使用，一般设置prefix/{level}/个数为threadNumber100倍以内，默认值为0，不可为负值	2	否
threadNumber	并发数，默认值是1	10	否

其他额外参数若指定会忽略。

详细说明

该命令模拟文件夹进行目录大小计算的操作。将依据AK/SK信息，在指定endpoint下的bucket中，找到preifx下所有obejct，依次获取每个object的大小并进行批量累积计算，最后输出该prefix的总大小，类似于计算整个目录的总大小。在执行过程中若同时有object上传到相同prefix下不能保证新上传的Object被计算到，可以通过设置合适的level和threadNumber值提高计算并发度，节省运行时间。该命令未统计碎片Object占用的空间，而控制台上的占用空间大小包括碎片。

使用示例

```
java -jar bosutil.jar -endpoint "http://bj.bcebos.com" -ak "your_access_key" -sk "your_secret_key" \
  -cmd COMPUTE_SIZE \
  -bucketName "mybucket" \
  -prefix "test"
```

```
java -jar bosutil.jar -endpoint "http://bj.bcebos.com" -ak "your_access_key" -sk "your_security_key" \
  -cmd COMPUTE_SIZE \
  -bucketName "mybucket" \
  -level "2" \
  -threadNumber "20"
```

TEST_SPEED命令

功能

测试下载速度。

语法

```
java -jar bosutil.jar -endpoint {endpoint} -ak {access_key} -sk {secret_key}
  -cmd TEST_SPEED
  [-testSize 10485760]
```

选项

除了必需的ak, sk, endpoint, cmd参数外, 其它参数参见下表:

选项	描述	示例	是否必选
testSize	测试的文件大小, 此文件完全放在内存, 默认为10485760	1000000	是

说明

其他额外参数若指定会忽略。

版本变更记录

- BOS Util工具包 [2023-10-23] 版本号0.1.7
  - 批量删除object支持设置正则匹配。
- BOS Util工具包 [2022-02-10] 版本号0.1.6
  - 功能优化，修复bug。
- BOS Util工具包 [2021-03-03] 版本号0.1.5
  - 功能优化。
- BOS Util工具包 [2020-11-19] 版本号0.1.4
  - 功能优化，修复bug。
- BOS Util工具包 [2020-10-16] 版本号0.1.3
  - 新增set_objects_acl命令支持批量设置给定object的acl信息。
- BOS Util工具包 [2017-03-16] 版本号0.1.2
  - set_objects_meta命令支持变量${object_basename}可以批量修改meta值功能。
  - 新增test_speed命令用于测试不同size的bos object上传下载速度。
  - 提高 compute size性能。
- BOS Util工具包 [2016-09-13] 版本号0.1.1

  更新README中的sample code。
- BOS Util工具包 [2016-09-02] 版本号0.1.0

  第一次发布。提供对BOS端数据的批量操作，主要包含批量设置metadata、批量删除文件、批量重命名文件、修改prefix、计算给定prefix(目录)下所有文件大小的总和等功能。

BOS-Probe错误检测工具

概述

当用户在使用百度智能云BOS上传或下载遇到问题时，可以使用BOSProbe 工具进行自我检测。

BOSPorbe 工具能够自动检测用户的上传和下载，并收集用户当前网络和命令执行的相关信息，最后形成一份可读的问题报告和错误处理建议。用户借助问题报告和错误处理建议可以简单、快速的排查出导致BOS上传下载出现问题的原因。用户在无法确认问题原因时，也可以将此报告通过提交工单发方式，发送给百度智能云客服，我们会帮助用户快速准确地排查出引起问题的根本原因，节省用户的时间。

通过BOSProbe检查上传下载

bcecmd 使用多层命令结构， 所有命令都以 bcecmd 开头，您可以用如下命令来使用 BOSProbe 提供的上传和下载检测功能：

```
$ bcecmd bosprobe <command> [options]
```

参数解释

参数项	取值	说明
command	取值为：**upload**、**download**。	各命令的作用请参见命令功能描述。
+options	BOSProbe命令的 Option，取值为**-a**、**-s**或**-b**等。	upload 和 download 可配置的option不一样，详细的使用方法请参见各命令行解释。

命令输出

- 网络检测结果；
- 上传检测结果；
- 错误信息；
- 建议；
- 检测报告保存地址。

上传检测

描述： 上传检测用于检测和收集上传过程中遇到的错误和收集网络信息，并生成报告和错误信息提示。上传的文件可以是BOSProbe生成的一个大小为1M的随机文件，也可以是您手动指定的随意大小的文件。

命令格式:

`bcecmd bosprobe upload -b <bucket> [-a <ak>] [-s <sk>] [-f local-path] [-o object] [-e endpoint]`

> **说明：**
>
> - `-b` 上传到指定的bucket （必需指定）；
> - `-a` 您使用的AK （bucket 为公共写时无需指定，非必需）；
> - `-s` 您使用的SK （bucket 为公共写时无需指定，非必需）；
> - `-f` 要上传的文档的路径, 如果为空，bosprobe则随机生成的文件;
> - `-o` 指定上传的文件在bucket中存储的名称，如果为空，则为本地文件名;
> - `-e` endpoint， 如果您指定了endpoint， 则上传数据到此endpoint 如果没指定，则根据bucket name推导出endpoint。
>
> **注意：**
>
> - 如果Bucket不是公共读写，执行命令时请指定AK/SK。

示例：

- **1.随机生成文件上传到指定endpoint**

  操作示例： `bcecmd bosprobe upload –a 123 –s 456 –b mybucket -e su.bcebos.com`

- **2.随机生成文件上传到指定ip**

  操作示例： `bcecmd bosprobe upload -a 123 -s 456 -b mybucket -e 123.234.1.233`

- **3.上传指定文件**

  操作示例： `bcecmd bosprobe upload –a 123 –s 456 –b mybucket –f ./file -o filename -e su.bcebos.com`

检测结果示例

- **1.上传成功**

```
  网络检测中.... [网络良好]
上传检测中.... [上传成功]

************************* 上传 ***************************
OBJECT NAME:    probe20171219210323660.temp
FILE SIZE :   1048576
USED TIME :   316 ms
SPEED     :   3.16 MB/s

************************* 建    议 ***************************
测试通过

如果提示没能解决问题，您也可以在百度智能云的管理控制台创建工单，将BOSProbe生成的日志文件(如果未能够生成日志文件，
请将BosProbe的所有输出信息复制到本地文件)反馈给我们，我们会尽快处理！

此次测试的日志保存于: /home/xxxx/bosprobe2017-xx-xx_xx_xx_xx_89568.log
```

- **2.上传失败**

```
 网络检测中.... [网络良好]
上传检测中.... [上传失败]

************************* 错误信息 ***************************
ERROR CODE: NoSuchBucket
ERROR MSG : [Code: NoSuchBucket; Message: The specified bucket does not exist.; RequestId: xxx]

************************* 建    议 ***************************
请检查您指定的 bucket 是否存在，如果存在，请您确认bucket name是否拼写正确、endpoint是否正确!

如果提示没能解决问题，您也可以在百度智能云的管理控制台创建工单，将BOSProbe生成的日志文件(如果未能够生成日志文件，
请将BosProbe的所有输出信息复制到本地文件)反馈给我们，我们会尽快处理！

此次测试的日志保存于: /home/xxxx/bosprobe2017-xx-xx_xx_xx_xx_9729.log
```

说明：

- 错误原因不同错误信息和建议会不同；
- 您可以根据检测结果中的建议来解决上传错误，如果给出的建议不能解决您的问题，您可以在百度智能云的管理控制台创建工单，并将BOSProbe生成的报告文件(如果未能够生成报告文件，请将BosProbe的所有输出信息复制到本地文件)反馈给我们，我们会尽快处理！

下载检测

描述：

下载检测用于检测和收集下载过程中遇到的错误和网络信息。下载检测时，您可以指定要下载的 object 名， 也可以不指定（如果不指定，BOSProbe将选择bucket中的第一个 object 用于下载测试）。

命令格式:

```
bcecmd bosprobe download [-a <ak>] [-s <sk>] [-f url] [-b bucket] [-o object] [-t to] [-e endpoint]
```

说明：

- `-a` 当前操作用户AK （bucket 为公共写和使用url下载测试时无需指定，非必需）；
- `-s` 当前操作用户SK （bucket 为公共写和使用url下载测试时无需指定，非必需）；
- `-f` object的url, 测试url下载时为必需；
- `-b` 要下载的文件所在的Bucket （测试url下载测试时非必需，非url下载测试时必需）；
- `-o` 要下载的object的名称 (非必需)；
- `-t` 文件保存的local地址 （非必需）；
- `-e` endpoint （非必需）。

注意：

- 不能即指定url又指定bucket 或object， 即 `-f` 不能与 `-b -o` 同时使用；
- 测试url下载时不需要指定 endpoint， 即 `-e` 不能与 `-f`一起使用。

示例：

- **1.url下载测试**

  操作示例：bcecmd bosprobe download -f http://bj.bcebos.com/mybucket/file1.txt 操作示例：bcecmd bosprobe download –f http://bj.bcebos.com/mybucket/file1.txt -t ./tmp/example.txt`

- **2.从指定bucket随机下载测试**

  操作示例： bcecmd bosprobe download –a 123 –s 456 –b mybucket

- **3.下载指定object** 操作示例: bcecmd bosprobe download –a 123 –s 456 –b mybucket –o file1.txt

- **4.下载指定object到本地指定位**

  操作示例: bcecmd bosprobe download –a 123 –s 456 –b mybucket –o file1.txt –t ./tmp/example.txt

检测结果示例

- **1.下载成功**

```
 网络检测中.... [网络良好]
下载检测中.... [下载成功]

************************* 下载 ***************************
   OBJECT NAME:    file1.txt
   FILE SIZE :    555
   USED TIME :    144 ms
   SPEED     :    0.00 MB/s

************************* 建    议 ***************************
测试通过

如果提示没能解决问题，您也可以在百度智能云的管理控制台创建工单，将BOSProbe生成的日志文件(如果未能够生成日志文件，
请将BosProbe的所有输出信息复制到本地文件)反馈给我们，我们会尽快处理！

此次测试的日志保存于: /home/xxxx/bosprobe2017-xx-xx_xx_xx_xx_84579.log
```

- **2.下载失败**

```
 网络检测中.... [网络良好]
下载检测中.... [下载失败]

************************* 错误信息 ***************************
   ERROR CODE: NoSuchKey
   ERROR MSG : [Code: NoSuchKey; Message: The specified key does not exist.; RequestId: xxx]

************************* 建    议 ***************************
您指定的object在BOS中不存在， 请您检查object name是否拼写正确! 如果指定的 object key 为目录，
请给object key 加上后缀 "/"。

如果提示没能解决问题，您也可以在百度智能云的管理控制台创建工单，将BOSProbe生成的日志文件(如果未能够生成日志文件，
请将BosProbe的所有输出信息复制到本地文件)反馈给我们，我们会尽快处理！

此次测试的日志保存于: /home/xxxx/bosprobe2017-xx-xx_xx_xx_xx_86018.log
```

安装BOSProbe

BOSProbe 是 bcecmd 的子命令，如果您已经安装了 bcecmd 就无需再安装 BOSProbe， 如果您还没有安装 bcecmd 您可以按如下方法安装 bcecmd:

1. 进入bcecmd下载页面， 地址链接: bcecmd。
2. 请根据自己的操作系统类型，下载对应版本的bcecmd，下载完成后只需解压zip包到您需要执行程序的目录，无需安装，您可以直接在终端中使用。

> **说明：**
>
> - Linux 用户解压命令为 $ unzip linux-bcecmd-0.1.zip；
> - Mac 用户解压命令为 $ unzip mac-bcecmd-0.1.zip；
> - 建议您将 bcecmd 所在目录加入操作系统的环境变量中，这样您在任何目录下都能够运行 bcecmd；
> - windows用户请在cmd中运行程序， linux用户请在终端中运行程序。

建议及报告查看

检测执行完成后会在控制台显示检测结果并生成测试报告（即测试日志）。

检测结果：

测试结果包含错误信息、建议和测试报告存放的地址。

测试报告：

测试报告包含网络测试结果、上传下载测试结果、错误信息、建议和调试信息。

> **说明：**
>
> 您可以根据测试结果中给出的建议来解决上传下载中出现的错误。如果给出的建议不能解决您的问题，您可以在百度智能云的管理控制台创建工单，并将BOSProbe生成的测试报告(如果未能够生成测试报告，请将BosProbe的所有输出信息复制到本地文件)作为工单的附件上传给我们，我们会尽快处理！

BOS-HDFS工具

概述

什么是BOS HDFS工具

BOS HDFS 工具是百度智能云基于 Hadoop 框架推出的专门用于解决大数据场景下BOS中数据的读写和使用问题的便捷工具。

大数据场景下的数据分析已经成为企业关注的核心业务。Hadoop 在分布式数据处理方面具有出色的能力，凭借其可靠、高效、可伸缩、并发处理的特点，已发展为当今最为主流的大数据开源框架之一。Hadoop 实现了一个分布式文件系统（Hadoop Distributed File System），简称为 HDFS。HDFS 具有高容错性的特点，并通过高吞吐量来支持应用程序的数据访问，适合超大数据集的业务场景。HDFS 为海量数据提供了可靠的存储性能，已成为 Hadoop 生态中最重要的一部分。数据量的与日俱增使原生的 Hadoop 面临一些新的问题，HDFS自建及运维成本非常高，同时本地 HDFS 上海量数据如何存放也是企业面临的巨大挑战。因此，在企业数据上云的趋势下，越来越多的企业选择将数据存储在云端，即存储在对象存储服务当中。但由于对象存储上层数据接口的限制，对象存储中数据和自建 HDFS 之间的访问和读写操作一直是大数据场景下的一个瓶颈，BOS HDFS 很好地解决了这个问题。

BOS HDFS 工具全面兼容 Hadoop 2.7+/3.1+ 系列，支持 HDFS 数据在 BOS 中的海量存储，并在上层数据运算中使用 HDFS 标准接口来对数据进行访问和读写，有效解决自建 HDFS 数据的高运维成本和低可扩展性问题。您可以通过调用该工具充分享受到 BOS 带来的超低价格、超高性能、高可靠和高吞吐的强大优势，满足企业在大数据场景中对数据的读写和使用需求。

BOS HDFS工具的优势

- 框架兼容：全面兼容 Hadoop 2.7+/3.1+
- 无感调用：实现对 BOS 中数据的无感调用
- 数据存储性价比高：融合对象存储服务 BOS 的超低价格、超高性能、高可靠性、高可用性和高吞吐优势

更新记录

【1.0.5】

- 支持append/truncate接口
- 支持多桶隔离配置ak/sk/endpoint
- 支持EnvironmentVariableCredentialsProvider
- 优化getFileStatus/create/delete接口
- 优化isFile/isDirectory接口
- 更新hadoop-common依赖到3.2.2版本

【1.0.4】

- 支持CRC32C Checksum校验

- 优化create接口
- 优化open接口
- 优化层级rename接口
- 默认分块调整10MB->12MB
- 顺序读策略优化

【1.0.3】

- 支持层级bucket
- 读cache优化，默认开启
- 优化顺序读
- 优化多文件删除
- 已知问题修复

配置和使用

您只需下载相应的SDK包，修改部分配置即可使用BOS HDFS工具。

下载

- 下载 BOS FS JAR，将解压后的jar包复制到 $hadoop_dir/share/hadoop/common中。

使用前准备

- 在Hadoop配置路径中修改 log4j.properties，调整BOS SDK的日志配置：log4j.logger.com.baidubce.http=WARN
- 在$hadoop_dir/etc/hadoop/core-site.xml文件中添加或者修改BOS HDFS相关配置。

```
  <property>
  <name>fs.bos.access.key</name>
  <value>{Your AK}</value>
</property>

<property>
  <name>fs.bos.secret.access.key</name>
  <value>{Your SK}</value>
</property>

<property>
  <name>fs.bos.endpoint</name>
  <value>http://bj.bcebos.com</value>
</property>

<property>
  <name>fs.bos.impl</name>
  <value>org.apache.hadoop.fs.bos.BaiduBosFileSystem</value>
</property>

<property>
  <name>fs.AbstractFileSystem.bos.impl</name>
  <value>org.apache.hadoop.fs.bos.BOS</value>
</property>
```

可配置的属性：

名称	描述
fs.bos.access.key	必须，BOS的访问accessKey。
fs.bos.secret.access.key	必须，BOS的访问secretKey。
fs.bos.endpoint	必须，BOS存储桶所在的endpoint。
fs.bos.session.token.key	可选，临时访问方式，如果配置，fs.bos.access.key和fs.bos.secret.access.key也应该是临时访问密钥。
fs.bos.credentials.provider	可选，credentials获取方式，可配置value值：org.apache.hadoop.fs.bos.credentials.BasicBOSCredentialsProvider、org.apache.hadoop.fs.bos.credentials.EnvironmentVariableCredentialsProvider，默认org.apache.hadoop.fs.bos.credentials.BasicBOSCredentialsProvider。
fs.bos.bucket.{your bucket}.access.key	可选，指定{your bucket}的BOS访问accessKey，访问{your bucket}资源优先使用此项。
fs.bos.bucket.{your bucket}.secret.access.key	可选，指定{your bucket}的BOS访问secretKey，访问{your bucket}资源优先使用此项。
fs.bos.bucket.{your bucket}.session.token.key	可选，指定{your bucket}的临时访问STS方式，如果配置，fs.bos.bucket.{your bucket}.access.key和fs.bos.bucket.{your bucket}.secret.access.key也应该是临时访问密钥，访问{your bucket}资源优先使用此项。
fs.bos.bucket.{your bucket}.endpoint	可选，指定{your bucket}的BOS存储桶所在的endpoint，访问{your bucket}资源优先使用此项。
fs.bos.block.size	可选，默认128MB， 模拟HDFS Block Size，在Distcp等需要验证Block信息的场景使用。
fs.bos.max.connections	可选，客户端支持的最大连接数，默认1000。
fs.bos.multipart.uploads.block.size	可选，客户端使用分块上传时候的分块大小，16777216，单位：byte。
fs.bos.multipart.uploads.concurrent.size	可选，客户端使用分块上传时候并发数，默认10。
fs.bos.object.dir.showtime	可选，list时候显示目录的修改时间，会额外增加一些交互次数，默认false。
fs.bos.tail.cache.enable	可选，缓存文件尾部部分数据，可以提升parquet/orc等文件的解析性能，默认true。
fs.bos.crc32c.checksum.enable	可选，开启crc32c checksum校验，开启后上传的文件才会计算CRC32C。
fs.bos.bucket.hierarchy	可选，bucket是否是namespace类型的，正确设置可节省一次校验开销，错误设置会到导致bucket访问失败。缺省，则自动查询选择。
fs.bos.rename.enable	可选，开启后支持元数据rename语义，当前小范围内测阶段，开启前需工单联系是否能使用，否则请求会返回403授权失败，默认false。

如使用EnvironmentVariableCredentialsProvider，需设置环境变量：

```
##### 必须，BOS的访问accessKey
FS_BOS_ACCESS_KEY
##### 必须，BOS的访问secretKey
FS_BOS_SECRET_ACCESS_KEY
##### 可选，临时访问方式
FS_BOS_SESSION_TOKEN_KEY
```

开始使用

使用BOS HDFS访问BOS服务时路径需要以`bos://`开始。如：

```
$ hdfs dfs -ls bos://{bucket}/
$ hdfs dfs -put ${local_file} bos://{bucket}/a/b/c
```

或者，在`$hadoop_dir/etc/core-site.xml`下配置默认文件目录

```
<property>
<name>fs.defaultFS</name>
<value>bos://{bucket}</value>
</property>
```

> 注意，在配置fs.defaultFS为BosFileSystem后，如果启动NameNode和DataNode可能会触发scheme检查失败。
>
> 建议在仅使用BosFileSystem时配置fs.defaultFS，且无需启动NameNode和DataNode；否则配置HDFS默认地址。

就像使用原生hdfs那样：

```
$ hdfs dfs -ls /
```

一次wordcount的实践

1. 创建数据目录

用于保存MapReduce任务的输入文件

```
$ hdfs dfs -mkdir -p bos://test-bucket/data/wordcount
```

用于保存MapReduce任务的输出文件

```
$ hdfs dfs -mkdir bos://test-bucket/output
```

查看这两个新创建的目录

```
$ hdfs dfs -ls bos://test-bucket/
```

```
Found 2 items
drwxrwxrwx   -          0 1970-01-01 08:00 bos://test-bucket/data
drwxrwxrwx   -          0 1970-01-01 08:00 bos://test-bucket/output
```

如果想要显示文件夹的准确创建时间，可以在`$hadoop_dir/etc/core-site.xml`下配置

```
<property>
<name>fs.bos.object.dir.showtime</name>
<value>true</value>
</property>
```

2. 写一个单词文件，并上传到hdfs

单词文件的内容

```
$ cat words.txt
```

```
hello baidu
hello bos
hadoop hdfs
hello hadoop
bos hdfs
```

上传words.txt到hdfs

```
$ hdfs dfs -put words.txt bos://test-bucket/data/wordcount
```

在hdfs中查看刚刚上传的文件

```
$ hdfs dfs -cat bos://test-bucket/data/wordcount/words.txt
```

```
hello baidu
hello bos
hadoop hdfs
hello hadoop
bos hdfs
```

3. 运行wordcount程序

hadoop自带的wordcount程序在`$hadoop_dir/share/hadoop/mapreduce/`下

```
$ hadoop jar hadoop-mapreduce-examples-2.7.7.jar wordcount bos://test-bucket/data/wordcount bos://test-bucket/output/wordcount
```

4. 查看统计结果

```
$ hdfs dfs -ls bos://test-bucket/output/wordcount/
-rw-rw-rw-   1          0 2020-06-12 16:55 bos://test-bucket/output/wordcount/_SUCCESS
-rw-rw-rw-   1         61 2020-06-12 16:55 bos://test-bucket/output/wordcount/part-r-00000
```

```
$ hdfs dfs -cat bos://test-bucket/output/wordcount/part-r-00000
baidu 1
bos     2
hadoop 2
hdfs 2
hello 3
```

hadoop distcp使用

DistCp（分布式拷贝）是hadoop自带的用于大规模集群内部和集群之间拷贝的工具。 它使用Map/Reduce实现文件分发，错误处理和恢复，以及报告生成。 它把文件和目录的列表作为map任务的输入，每个任务会完成源列表中部分文件的拷贝。

对于HDFS集群和BOS间的数据拷贝，可以借助BOS HDFS 工具，像标准的Hadoop Distcp那样。

前置工作

如上，下载BOS HDFS相关jar包，并做一些必要的配置。

以数据源端是HDFS集群，目的端是BOS为例，检查源和目的端的读写正常。

```
$ hadoop fs -get hdfs://host:port/xxx ./
$ hadoop fs -put xxx bos://bucket/xxx
```

开始拷贝

普通拷贝

```
##### 从hdfs的src，拷贝到bos指定bucket下的dst路径，默认情况会跳过已经存在的目标文件
$ hadoop distcp hdfs://host:port/src bos://bucket/dst
```

注意：使用CRC校验拷贝前后的数据，BOS HDFS需设置fs.bos.block.size和源HDFS一致，并开启fs.bos.crc32c.checksum.enable；仅支持HDFS的dfs.checksum.combine.mode=COMPOSITE_CRC校验算法。

更新和覆盖

```
##### 拷贝，执行覆盖的唯一标准是源文件和目标文件大小是否相同，如果不同，则源文件替换目标文件
$ hadoop distcp -update hdfs://host:port/src bos://bucket/dst

##### 拷贝，并覆盖已经存在的目标文件
$ hadoop distcp -overwrite hdfs://host:port/src bos://bucket/dst
```

拷贝多个源

```
##### 指定多个源路径
$ hadoop distcp hdfs://host:port/src1 hdfs://host:port/src2 bos://bucket/dst

##### 从文件里获得多个源
##### srcList的内容是类似hdfs://host:port/src1、hdfs://host:port/src2的多行源路径
$ hadoop distcp hdfs://host:port/srcList bos://bucket/dst
```

注意：多个源的拷贝可能会有冲突，参考标准hadoop distcp的处理。

更多配置

```
##### 指定拷贝数据时map的数目
##### 更多的map数量可能不会提升数据吞吐，反而会带来一些问题，map数量应该根据集群资源和拷贝数据规模综合设定
$ hadoop distcp -m 10 hdfs://host:port/src bos://bucket/dst

##### 忽略失败的map，但会保留失败操作日志
$ hadoop distcp -i hdfs://host:port/src bos://bucket/dst

##### 动态分配任务
##### 默认的分配策略是基于文件大小，在更新拷贝时候，因为有跳过文件，实际拷贝时候有类似"数据倾斜"的问题，耗时最长的map拖慢整体进度
$ hadoop distcp -strategy dynamic -update hdfs://host:port/src bos://bucket/dst
```

更多配置参数，输入hadoop distcp命令即可获取帮助信息。

使用进阶

由于自建Hadoop集群的可拓展性有限，且需要大量人力对集群进行运维，如果您对性能及安全性有更高要求，推荐使用百度智能云提供的 百度 MapReduce（BMR）。BMR 是全托管的 Hadoop/Spark 集群，您可以按需部署并弹性扩展集群，只需专注于大数据处理、分析、报告，拥有多年大规模分布式计算技术积累的百度运维团队全权负责集群运维，能够在性能，安全性和便捷性上有大幅提升。

大数据组件使用指南

Hive使用指南

Hive

hive是基于Hadoop的一个数据仓库工具，用来进行数据提取、转化、加载，这是一种可以存储、查询和分析存储在Hadoop中的大规模数据的机制。hive数据仓库工具能将结构化的数据文件映射为一张数据库表，并提供SQL查询功能，能将SQL语句转变成MapReduce任务来执行.

前提条件

首先参考BOS HDFS一文安装并配置BOS HDFS，本机安装的hadoop版本为hadoop-3.3.2，参考文中"开始使用"一节完成BOS HDFS的基本试用，并设置环境变量：

```
export HADOOP_HOME=/opt/hadoop-3.3.2
export HADOOP_CLASSPATH=`$HADOOP_HOME/bin/hadoop classpath
```

安装mysql mysql用来存储hive的元数据，可以选择本地安装，也可以直接远程连接已安装的mysql或者RDS。本机安装的版本是：mysql-5.1.61-4.el6.x86_64 安装完成之后，可用 service mysqld status 查看运行状态，并使用

```
/usr/bin/mysqladmin -u root -h ${IP} password ${new-password} #设置新密码
```

可以在mysql中创建hive专用的用户，并设置密码。 **安装hive** 本机安装的版本为2.3.9. 修改conf文件夹下的两个配置：

```
mv hive-env.sh.template hive-env.sh
mv hive-site.xml.template hive-site.xml
```

在hive-env.sh中添加：

```
export HIVE_CONF_DIR=/ssd2/apache-hive-2.3.9-bin/conf
```

在hive-site.xml中添加：

```
<property>
    <name>javax.jdo.option.ConnectionURL</name>
    <value>jdbc:mysql://localhost:3306/hive?createDatabaseIfNotExist=true</value>
    <description>MySQL</description>
</property>
<property>
    <name>javax.jdo.option.ConnectionDriverName</name>
    <value>com.mysql.jdbc.Driver</value>
    <description>JDBC</description>
</property>
<property>
    <name>javax.jdo.option.ConnectionUserName</name>
    <value>root</value>
    <description>username</description>
    </property>
<property>
    <name>javax.jdo.option.ConnectionPassword</name>
    <value>new-password</value>
    <description>password</description>
</property>
```

配置中 javax.jdo.option.ConnectionURL 是指mysql server的连接地址，javax.jdo.option.ConnectionUserName 就是用于hive的mysql用户名，javax.jdo.option.ConnectionPassword 是用户名对应的密码。设置完成后，把mysql的JBDC驱动复制到lib文件夹下，本机采用的驱动为 mysql-connector-java-5.1.32-bin.jar. 初始化msyql

```
./bin/schematool -dbType mysql -initSchema
```

启动hive

```
./bin/hive
```

hive测试

创建表

```
create database hive;     // 创建数据库
create table hive_test (a int, b string) //创建表
ROW FORMAT DELIMITED FIELDS TERMINATED BY ',';
```

1. 新建一个shell脚本，名为gen_data.sh

```
###### !/bin/bash
MAXROW=1000000 #指定生成数据行数
for((i = 0; i < $MAXROW; i++))
do
  echo $RANDOM, \"$RANDOM\"
done
```

2.运行脚本，生成测试数据

```
./gen_data.sh > hive_test.data
```

3.把数据加载到表中

```
load data inpath "bos://${bucket_name}/hive_test.data" into table hive.hive_test;
```

4.查询

```
hive> select count(*) from hive_test;
WARNING: Hive-on-MR is deprecated in Hive 2 and may not be available in the future versions. Consider using a different execution engine (i.e. spark, tez) or using Hive 1.X releases.
Query ID = root_20230528173013_6f5296db-562e-4342-917f-bcf14fc1480d
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks determined at compile time: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Job running in-process (local Hadoop)
2023-05-28 17:30:16,548 Stage-1 map = 0%,  reduce = 0%
2023-05-28 17:30:18,558 Stage-1 map = 100%,  reduce = 100%
Ended Job = job_local238749048_0001
MapReduce Jobs Launched:
Stage-Stage-1:  HDFS Read: 0 HDFS Write: 0 SUCCESS
Total MapReduce CPU Time Spent: 0 msec
OK
1000000
hive> select * from hive_test limit 10;
OK
11027   "11345"
10227   "24281"
32535   "16409"
24286   "24435"
2498    "10969"
16662   "16163"
5345    "26005"
21407   "5365"
30608   "4588"
19686   "11831"
```

Presto使用指南

概览

Presto是Facebook开发的数据查询引擎，可对海量数据进行快速地交互式分析，支持Hive，关系数据库等多种数据源。由于 BOS 在超低价格、超高性能、高可靠和高吞吐的强大存储优势，越来越多企业选择 BOS 作为大数据的存储媒介。因此，本文将对Presto在 BOS 上的使用方法作一个简要的介绍。

前提条件

参考Hive使用指南一文安装并配置Hive

安装配置 安装版本为349，可参考presto部署一文的过程。其中，如果本机也作为worker，在config.properties中可设置：

```
node-scheduler.include-coordinator=true #需要改为true
```

在etc/catalog/hive.properties中，配置为：

```
connector.name=hive-hadoop2
hive.config.resources=/ssd2/hadoop-3.3.2/etc/hadoop/core-site.xml,/ssd2/hadoop-3.3.2/etc/hadoop/hdfs-site.xml #这里的地址一定要正确
hive.metastore.uri=thrift://127.0.0.1:9083
hive.allow-drop-table=false
hive.storage-format=ORC
hive.metastore-cache-ttl=1s
hive.metastore-refresh-interval=1s
hive.metastore-timeout=35m
hive.max-partitions-per-writers=1000
hive.cache.enabled=true
hive.cache.location=/opt/hive-cache
```

把bos filesystem的jar包复制到plugin/hive-hadoop2/下，之后运行：

```
./bin/launcher start
```

启动presto-server，

```
./presto-cli --server localhost:8881 --catalog hive --schema default
```

运行

```
presto:default>use hive;
USE
presto:hive>select * from hive_test limit 10;
   a   |    b
-------+----------
 11027 | "11345"
 10227 | "24281"
 32535 | "16409"
 24286 | "24435"
  2498 | "10969"
 16662 | "16163"
  5345 | "26005"
 21407 | "5365"
 30608 | "4588"
 19686 | "11831"
(10 rows)

Query 20230601_084130_00004_dzvjb, FINISHED, 1 node
Splits: 18 total, 18 done (100.00%)
[Latency: client-side: 0:02, server-side: 0:02] [59.4K rows, 831KB] [28.9K rows/s, 404KB/s]
```

在以上示例中，我们就通过presto查询到了存储在bos中的数据。

基于S3的presto访问

presto访问存储在BOS中的数据只能是通过hive，但是hive访问BOS中的数据有两种方式，第一种就是通过上述的介绍，基于bos-hdfs；第二种就是直接通过S3协议访问BOS。 **hive配置** 安装metastore：

```
wget "https://repo1.maven.org/maven2/org/apache/hive/hive-standalone-metastore/3.1.2/hive-standalone-metastore-3.1.2-bin.tar.gz"
tar -zxvf hive-standalone-metastore-3.1.2-bin.tar.gz
sudo mv apache-hive-metastore-3.1.2-bin /usr/local/metastore
sudo chown user:user /usr/local/metastore
```

下载并使用hive-standalone-metastore，增加必需的jar包：

```
rm /usr/local/metastore/lib/guava-19.0.jar
cp /usr/local/hadoop/share/hadoop/common/lib/guava-27.0-jre.jar \
 /usr/local/metastore/lib/
cp /usr/local/hadoop/share/hadoop/tools/lib/hadoop-aws-3.2.1.jar \
 /usr/local/metastore/lib/
cp /usr/local/hadoop/share/hadoop/tools/lib/aws-java-sdk-bundle-1.11.375.jar \
 /usr/local/metastore/lib/
```

之后配置/usr/local/metastore/conf/metastore-site.xml

```
<property>
  <name>javax.jdo.option.ConnectionURL</name>
  <value>jdbc:mysql://localhost/metastore?createDatabaseIfNotExist=true</value>
</property>
<property>
  <name>javax.jdo.option.ConnectionDriverName</name>
  <value>com.mysql.jdbc.Driver</value>
</property>
<property>
  <name>javax.jdo.option.ConnectionUserName</name>
  <value>hive</value>
</property>
<property>
  <name>javax.jdo.option.ConnectionPassword</name>
  <value>hive</value>
</property>
<property>
  <name>hive.metastore.event.db.notification.api.auth</name>
  <value>false</value>
</property>
<property>
  <name>fs.s3a.access.key</name>
  <value>S3_ACCESS_KEY</value>
</property>
<property>
  <name>fs.s3a.secret.key</name>
  <value>S3_SECRET_KEY</value>
</property>
<property>
  <name>fs.s3a.connection.ssl.enabled</name>
  <value>false</value>
</property>
<property>
  <name>fs.s3a.path.style.access</name>
  <value>true</value>
</property>
<property>
  <name>fs.s3a.endpoint</name>
  <value>S3_ENDPOINT</value>
</property>
```

fs.s3a.endpoint 是指s3 的endponit，bos的s3 endpoint在BOS S3域名可查.

启动hive metastore:

```
/usr/local/metastore/bin/start-metastore &
```

presto配置 preto的版本这里不作改变，hive.properties的内容改为：

```
connector.name=hive-hadoop2
hive.metastore.uri=thrift://localhost:9083
hive.s3.path-style-access=true
hive.s3.endpoint=S3_ENDPOINT
hive.s3.aws-access-key=S3_ACCESS_KEY
hive.s3.aws-secret-key=S3_SECRET_KEY
hive.s3.ssl.enabled=false
```

启动presto

```
/usr/local/trino/bin/launcher start
```

创建schema

```
 CREATE SCHEMA IF NOT EXISTS hive.iris
WITH (location = 's3a://my-bos-bucket/'); //注意，这里的location就是以s3a开头，如果bucket为my-bos-bucket，则location应为s3a://my-bos-bucket/开头
###### 创建表
CREATE TABLE IF NOT EXISTS hive.iris.iris_parquet (
  sepal_length DOUBLE,
  sepal_width  DOUBLE,
  petal_length DOUBLE,
  petal_width  DOUBLE,
  class        VARCHAR
)
WITH (
  external_location = 's3a://my-bos-bucket/iris_parquet',
  format = 'PARQUET'
);

SELECT
  sepal_length,
  class
FROM hive.iris.iris_parquet
LIMIT 10;
```

Impala使用指南

Impala

Impala是用于处理存储在Hadoop集群中的大量数据的MPP（大规模并行处理）SQL查询引擎。 它是一个用C ++和Java编写的开源软件。 与其他Hadoop的SQL引擎相比，它提供了高性能和低延迟。

安装步骤

安装metastore 参考Presto使用指南一文中"基于S3的presto访问"一节安装并配置metastore **安装impala**

1. 下载rpm包，地址http://archive.cloudera.com/cdh5/repo-as-tarball/5.14.0/cdh5.14.0-centos6.tar.gz 使用`tar -zxvf cdh5.14.0-centos6.tar.gz`解压之后，cd cdh/5.14.0，创建本地server，运行

```
 python -m SimpleHTTPServer 8092 &
```

2. 配置本地yum源

```
 vim /etc/yum.repos.d/localimp.repo
[localimp]
name=localimp
baseurl=http://127.0.0.1:8092/
gpgcheck=0
enabled=1
```

3. 使用如下命令安装

```
 yum install -y impala impala-server impala-state-store impala-catalog impala-shell
```

4. 将hive的配置文件(metastore-site.xml)复制到impala的配置路径下：

```
 ###### 把配置好的conf复制到/etc/impala/conf/路径下
cp metastore/conf/metastore-site.xml  /etc/impala/conf/hive-site.xml
```

5. 增加s3配置 vim /etc/impala/conf/core-site.xml，参考impala-s3配置

```
 <configuration>
 <property>
    <name>fs.s3a.block.size</name>
    <value>134217728 </value>
 </property>
<property>
   <name>fs.azure.user.agent.prefix</name>
   <value>User-Agent: APN/1.0 Hortonworks/1.0 HDP/None</value>
</property>
<property>
   <name>fs.s3a.connection.maximum</name>
   <value>1500</value>
</property>
<property>
   <name>fs.defaultFS</name>
      <value>s3a://${bucket}</value>
      </property>
<property>
   <name>fs.s3a.endpoint</name>
      <value>s3.bj.bcebos.com</value>
         <description>endpoint</description>
         </property>
<property>
   <name>fs.s3a.access.key</name>
      <value>${AK}</value>
         <description>AK</description>
         </property>
<property>
   <name>fs.s3a.secret.key</name>
      <value>${SK}</value>
         <description>SK</description>
         </property>
</configuration>
```

6. 修改bigtop的配置，设置JAVA_HOME，并确保impala用户也具有访问权限 修改bigtop的java_home路径（3台机器）

```
vim /etc/default/bigtop-utils
export JAVA_HOME=/export/servers/jdk1.8.0_65
```

7. 设置mysql驱动的软链接：

```
ln -s mysql-connector-java-5.1.32.jar /usr/share/java/mysql-connector-java.jar
```

8. 启动impala

```
service impala-state-store start
service impala-catalog start
service impala-server start
```

启动后可在/var/log/impala文件夹下查看日志 运行impala-shell命令：

```
[root@my-node impala]# impala-shell
Starting Impala Shell without Kerberos authentication
Connected to my-node:21000
Server version: impalad version 2.11.0-cdh5.14.0 RELEASE (build d68206561bce6b26762d62c01a78e6cd27aa7690)
***********************************************************************************
Welcome to the Impala shell.
(Impala Shell v2.11.0-cdh5.14.0 (d682065) built on Sat Jan  6 13:27:16 PST 2018)

When pretty-printing is disabled, you can use the '--output_delimiter' flag to set
the delimiter for fields in the same row. The default is ','.
***********************************************************************************
[my-node:21000] > show databases;
Query: show databases
+------------------+----------------------------------------------+
| name             | comment                                      |
+------------------+----------------------------------------------+
| _impala_builtins | System database for Impala builtin functions |
| default          | Default Hive database                        |
+------------------+----------------------------------------------+
Fetched 2 row(s) in 0.16s
[my-node:21000] > CREATE DATABASE db_on_s3 LOCATION 's3a://my-bigdata/impala/s3';
Query: create DATABASE db_on_s3 LOCATION 's3a://my-bigdata/impala/s3'
WARNINGS: Path 's3a://my-bigdata/impala' cannot be reached: Path does not exist.

Fetched 0 row(s) in 2.51s
[my-node:21000] > show databases;
Query: show databases
+------------------+----------------------------------------------+
| name             | comment                                      |
+------------------+----------------------------------------------+
| _impala_builtins | System database for Impala builtin functions |
| db_on_s3         |                                              |
| default          | Default Hive database                        |
+------------------+----------------------------------------------+
Fetched 3 row(s) in 0.01s
[my-node:21000] > use db_on_s3;
Query: use db_on_s3
[my-node:21000] > create table hive_test (a int, b string) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',';
Query: create table hive_test (a int, b string) ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
Fetched 0 row(s) in 2.11s
[my-node:21000] > insert into hive_test(a, b) values(1,'tom');
Query: insert into hive_test(a, b) values(1,'tom')
Query submitted at: 2023-09-13 19:20:26 (Coordinator: http://my-node:25000)
Query progress can be monitored at: http://my-node:25000/query_plan?query_id=ec4463f20d37dfe4:5192e94f00000000
Modified 1 row(s) in 7.57s
[my-node:21000] > insert into hive_test(a, b) values(2,'jerry');
Query: insert into hive_test(a, b) values(2,'jerry')
Query submitted at: 2023-09-13 19:20:42 (Coordinator: http://my-node:25000)
Query progress can be monitored at: http://my-node:25000/query_plan?query_id=694061adf492a154:4a24912d00000000
Modified 1 row(s) in 1.02s
```

在对应路径下可看见新生成的文件：



Spark使用指南

Spark

Apache Spark 是专为大规模数据处理而设计的快速通用的计算引擎。Spark是UC Berkeley AMP lab (加州大学伯克利分校的AMP实验室)所开源的类Hadoop MapReduce的通用并行框架，拥有Hadoop MapReduce所具有的优点;但不同于MapReduce的是--Job中间输出结果可以保存在内存中，从而不再需要读写HDFS，因此Spark能更好地适用于数据挖掘与机器学习等需要迭代的MapReduce的算法。

安装

1.Spark环境准备

```
###### 下载到一个路径
wget https://archive.apache.org/dist/spark/spark-3.1.1/spark-3.1.1-bin-hadoop3.2.tgz

###### 解压
tar -zxvf spark-3.1.1-bin-hadoop3.2.tgz

###### 设置环境变量
SPARK_HOME=/home/hadoop/spark-3.1.1-bin-hadoop3.2
PATH=$PATH:$HADOOP_HOME/bin:$SPARK_HOME/bin
```

2.添加依赖jar包 下载BOS-HDFS

```
###### 解压，拷贝到spark依赖路径
unzip bos-hdfs-sdk-1.0.3-community.jar.zip
cp bos-hdfs-sdk-1.0.3-community.jar {SPARK_HOME}/jars

###### 访问BOS的一些必要配置
cp {SPARK_HOME}/conf/spark-defaults.conf.template {SPARK_HOME}/conf/spark-defaults.conf
vim {SPARK_HOME}/conf/spark-defaults.conf
...
cat {SPARK_HOME}/spark-defaults.conf
spark.hadoop.fs.bos.access.key={your ak}
spark.hadoop.fs.bos.secret.access.key={your sk}
spark.hadoop.fs.bos.endpoint=http://bj.bcebos.com {your bucket endpiont}
spark.hadoop.fs.AbstractFileSystem.bos.impl=org.apache.hadoop.fs.bos.BOS
spark.hadoop.fs.bos.impl=org.apache.hadoop.fs.bos.BaiduBosFileSystem
```

使用

编写demo.py,读取my-bucket中student.parquet，统计age > 22的student。

```
from pyspark.sql import SparkSession

spark = SparkSession.builder.appName("example_app")\
    .config("spark.driver.bindAddress", "localhost")\
    .getOrCreate()
bosFile = "bos://my-bucket/student"

###### 写入
data = [("abc",22), ("def",17), ("ghi",34)]
df = spark.createDataFrame(data, ["name", "age"])
df.write.parquet(bosFile)

df = spark.read.parquet(bosFile)
df.printSchema()

df.createOrReplaceTempView("students")
sqlDF = spark.sql("SELECT * FROM students WHERE age > 22 LIMIT 10")
sqlDF.show()
```

运行

```
spark-submit demo.py
```

Flink使用指南

Flink

flink-bos-hadoop是百度云对象存储系统BOS针对 Flink 的文件系统实现，并且支持了 recoverwriter 接口，Flink 可以基于该文件系统实现读写 BOS 上的数据以及作为流应用的状态后端。

安装

1.Flink环境准备 以1.15.0版本为例。

```
###### 下载到一个路径
wget https://archive.apache.org/dist/flink/flink-1.15.0/flink-1.15.0-bin-scala_2.12.tgz

###### 解压缩
tar zxvf flink-1.15.0-bin-scala_2.12.tgz
```

2.添加依赖jar包和配置 在启动 Flink 之前，下载flink-bos-hadoop插件。

```
###### flink-bos-hadoop插件导入flink下
mkdir ./plugins/bos-fs-hadoop
cp flink-bos-hadoop-1.15.0-0.1.0.jar ./plugins/bos-fs-hadoop/

###### 访问BOS的一些必要配置
vim ./conf/flink-conf.yaml
...
cat ./conf/flink-conf.yaml
fs.bos.impl: org.apache.hadoop.fs.bos.BaiduBosFileSystem
fs.AbstractFileSystem.bos.impl: org.apache.hadoop.fs.bos.BOS
fs.bos.access.key: {your ak}
fs.bos.secret.access.key: {your sk}
fs.bos.endpoint: bj.bcebos.com {your bucket endpoint}
```

使用

启动

```
./bin/start-cluster.sh
```

提交作业

```
###### 通过以下格式指定路径，BOS 对象可类似于普通文件使用：bos://<your-bucket>/{object-name}
./bin/flink run examples/streaming/WordCount.jar --input "bos://my_bucket/students.txt" --output "bos://my_bucket/out"
```

查看运行结果

```
###### 查看wordcount统计结果
$ hadoop fs -ls bos://my_bucket/out
Found 1 items
drwxrwxrwx   -          0 1970-01-01 08:00 bos://my_bucket/out/2023-08-10--15

$ hadoop fs -ls bos://my_bucket/out/2023-08-10--15/
Found 1 items
-rw-rw-rw-   1       1792 2023-08-10 15:52 bos://my_bucket/out/2023-08-10--15/part-3053774f-2d8e-40c5-aa3c-01402ce4b6b4-0

$ hadoop fs -cat bos://my_bucket/out/2023-08-10--15/part-3053774f-2d8e-40c5-aa3c-01402ce4b6b4-0
(name,1)
(studentname,1)
(age,1)
...
```

Flume数据存储到BOS

Flume

Flume是一个分布式、可靠性和高可用的海量日志聚合系统，支持在系统中定制各类数据发送方，用于收集数据；同时，FLume提供对数据进行简单处理，并写到各种数据接收方（可定制）的能力。

Flume支持多种Sink类型，可以借助HDFS Sink将收集到的数据存储到BOS。

开始

1. 下载并安装apache-flume 略

2. 配置环境 如果已有hadoop环境，且已配置过访问BOS，本环节跳过； 否则

- 将bos-hdfs的jar包下载到/opt/apache-flume-1.xx.0-bin/lib目录下；
- 将hadoop下的配置文件core-site.xml添加访问BOS的相关配置，并复制到/opt/apache-flume-1.xx.0-bin/conf目录下。

3. 创建flume配置文件 把flume的StressSource作为source，使用内存channel，通过HDFS协议写入BOS。

```
###### ss2bos.properties
agent.sources = stress_source
agent.channels = mem_channel
agent.sinks = bos_hdfs_sink

agent.sources.stress_source.type = org.apache.flume.source.StressSource
agent.sources.stress_source.channels = mem_channel
agent.sources.stress_source.size = 1024
agent.sources.stress_source.maxTotalEvents = 1000
agent.sources.stress_source.maxEventsPerSecond = 10
agent.sources.stress_source.batchSize=10

agent.channels.mem_channel.type = memory
agent.channels.mem_channel.capacity = 1000000
agent.channels.mem_channel.transactionCapacity = 100

agent.sinks.bos_hdfs_sink.channel = mem_channel
agent.sinks.bos_hdfs_sink.type = hdfs
agent.sinks.bos_hdfs_sink.hdfs.useLocalTimeStamp = true
agent.sinks.bos_hdfs_sink.hdfs.filePrefix = %{host}_bos_hdfs_sink #host区分文件，避免并发写冲突
agent.sinks.bos_hdfs_sink.hdfs.path = bos://{your bucket}/flume/%Y-%m-%d-%H-%M #替换bucket路径
agent.sinks.bos_hdfs_sink.hdfs.fileType = DataStream
agent.sinks.bos_hdfs_sink.hdfs.writeFormat = Text
agent.sinks.bos_hdfs_sink.hdfs.rollSize = 0
agent.sinks.bos_hdfs_sink.hdfs.rollCount = 100
agent.sinks.bos_hdfs_sink.hdfs.rollInterval = 0
agent.sinks.bos_hdfs_sink.hdfs.batchSize = 100
agent.sinks.bos_hdfs_sink.hdfs.round = true
agent.sinks.bos_hdfs_sink.hdfs.roundValue = 10
agent.sinks.bos_hdfs_sink.hdfs.roundUnit = minute
```

4. 启动Flume agent

```
./bin/flume-ng agent -n agent -c conf/ -f ss2bos.properties
```

DataX读写BOS

DataX

DataX 是一个异构数据源离线同步工具，致力于实现包括关系型数据库(MySQL、Oracle等)、HDFS、Hive、ODPS、HBase、FTP等各种异构数据源之间稳定高效的数据同步功能。

配置

1. 下载并解压DataX；
2. 下载BOS-HDFS 解压缩后，将jar包拷贝到Datax解压路径plugin/reader/hdfsreader/libs/以及plugin/writer/hdfswriter/libs/下；
3. 打开DataX解压目录下的bin/datax.py脚本，修改脚本中的CLASS_PATH变量为如下：

```
CLASS_PATH = ("%s/lib/*:%s/plugin/reader/hdfsreader/libs/*:%s/plugin/writer/hdfswriter/libs/*:.") % (DATAX_HOME, DATAX_HOME, DATAX_HOME)
```

开始

示例 将{your bucket}下的testfile文件读出并写入到{your other bucket}存储桶。

testfile:

```
1 hello
2 bos
3 world
```

bos2bos.json:

```
{
  "job": {
    "setting": {
      "speed": {
        "channel": 1
      },
      "errorLimit": {
        "record": 0,
        "percentage": 0.02
      }
    },
    "content": [{
      "reader": {
        "name": "hdfsreader",
        "parameter": {
          "path": "/testfile",
          "defaultFS": "bos://{your bucket}/",
          "column": [
            {
              "index": 0,
              "type": "long"
            },
            {
              "index": 1,
              "type": "string"
            }
          ],
          "fileType": "text",
          "encoding": "UTF-8",
          "hadoopConfig": {
            "fs.bos.endpoint": "bj.bcebos.com",
            "fs.bos.impl": "org.apache.hadoop.fs.bos.BaiduBosFileSystem",
            "fs.bos.access.key": "{your ak}",
            "fs.bos.secret.access.key": "{your sk}"
          },
          "fieldDelimiter": " "
        }
      },
      "writer": {
        "name": "hdfswriter",
        "parameter": {
          "path": "/testtmp",
          "fileName": "testfile.new",
          "defaultFS": "bos://{your other bucket}/",
          "column": [{
              "name": "col1",
              "type": "string"
            },
            {
              "name": "col2",
              "type": "string"
            }
          ],
          "fileType": "text",
          "encoding": "UTF-8",
          "hadoopConfig": {
            "fs.defaultFS": ""
            "fs.bos.endpoint": "bj.bcebos.com",
            "fs.bos.impl": "org.apache.hadoop.fs.bos.BaiduBosFileSystem",
            "fs.bos.access.key": "{your ak}",
            "fs.bos.secret.access.key": "{your sk}"
          },
          "fieldDelimiter": " ",
          "writeMode": "append"
        }
      }
    }]
  }
}
```

> 按需替换配置中的{your bucket}、endpoint、{your sk}等选项；
>
> 支持仅reader或writer配置为BOS。

结果

```
python bin/datax.py bos2bos.json
```

执行成功后返回:

```
2023-10-16 14:51:44.907 [job-0] INFO  JobContainer -
任务启动时刻                    : 2023-10-16 14:51:32
任务结束时刻                    : 2023-10-16 14:51:44
任务总计耗时                    :                 12s
任务平均流量                    :               1B/s
记录写入速度                    :              0rec/s
读出记录总数                    :                   3
读写失败总数                    :                   0
```

BOS-AlluxioExtension工具

Alluxio是什么

Alluxio是一个开源的基于内存的分布式存储系统（a memory speed virtual distributed storage system）。

在大数据生态系统中，Alluxio 位于数据驱动框架或应用（如 Apache Spark等）和各种持久化存储系统（如BOS、HDFS、S3等）之间。Alluxio 统一了存储在这些不同存储系统中的数据，为其上层数据驱动型应用提供统一的客户端 API 和全局命名空间。

通过Alluxio使用BOS的优势

Alluxio内存至上的层次化架构使得对BOS数据的访问能够更便捷，对于频繁访问的数据，大幅减少对BOS接口的请求次数。

读

- 内存级的 I/O 吞吐率，Alluxio的层次化存储机制能够充分利用缓存的频繁访问的数据
- 有效降低对象存储中的一些操作（如列出目录、重命名）的时延
- 简化数据管理，Alluxio支持对多源数据的单点访问
- 兼容性，现有的数据分析应用程序，如 Spark 和 MapReduce 程序，无需更改任何代码就能在 Alluxio 上运行

写

智能缓存，可根据需要配置写策略：

- MUST_CACHE：只写cache，适用于不需要保存的临时数据，虽然丢失数据风险高，但性能最好
- THROUGH：对于近期不会使用的数据可以直接写进BOS，在Alluxio中为其他频繁读取的数据留出更多的内存空间
- CACHE_THROUGH： 同步写Alluxio和BOS，数据很快会被其他Alluxio应用使用
- ASYNC_THROUGH：默认模式，写cache，异步写BOS，适用于需要持久化，并且不需要立即使用的数据

快速开始

部署

1. 下载Alluxio、预编译的BOS底层存储alluxio-underfs-bos并解压
2. 安装alluxio-underfs-bos的jar包

```
$ cd {ALLUXIO_HOME}
$ ./bin/alluxio extensions install <path>/alluxio-underfs-bos-0.1.0.jar
```

3. 在${ALLUXIO_HOME}/conf目录下，根据模板文件创建conf/alluxio-site.properties配置文件，配置BOS的AK/SK，并支持临时授权访问STS。

```
$ cp conf/alluxio-site.properties.template conf/alluxio-site.properties
```

```
fs.bos.accessKey=<BOS_AK>
fs.bos.secretKey=<BOS_SK>
fs.bos.endpoint=<BOS_ENDPOINT>
alluxio.user.file.writetype.default=CACHE_THROUGH
```

Alluxio挂载BOS

1. 格式化 Alluxio 的日志和 worker 存储目录。

```
$ ./bin/alluxio format
```

2. 在 localhost 启动 Alluxio

```
$ ./bin/alluxio-start.sh local
```

3. 创建一个目录，并挂载BOS，BOS上需要有一个bucket，此处以"test-979"桶为例

```
$ ./bin/alluxio fs mkdir /mnt
Successfully created directory /mnt
```

```
$ ./bin/alluxio fs mount /mnt/bos bos://test-979
Mounted bos://test-979 at /mnt/bos
```

4. 复制本地文件到Alluxio

```
$ ./bin/alluxio fs copyFromLocal LICENSE /mnt/bos
Copied file:///alluxio-2.0.0/LICENSE to /mnt/bos
```

6. 使用list命令查看刚刚复制过去的文件，回显的各参数含义：权限、文件大小、是否被持久化到BOS、创建日期、Alluxio中这个文件的缓存占比、文件名

```
$ ./bin/alluxio fs ls /mnt/bos/LICENSE
-rwxrwxrwx       27040       PERSISTED 07-21-2020 15:06:46:000 100% /mnt/bos/LICENSE
```

7. 停止Alluxio

```
$ ./bin/alluxio-stop.sh local
```

体验Alluxio加速访问BOS数据

Alluxio利用内存存储数据，通过它可以更快地访问BOS数据。体验一下：

1. list查看/mnt/bos目录下文件列表，0%表示该文件不在Alluxio内存中

```
 $ time ./bin/alluxio fs ls /mnt/bos
-rwxrwxrwx       27040       PERSISTED 07-21-2020 15:06:46:000   0% /mnt/bos/LICENSE
-rwxrwxrwx       51307896    PERSISTED 07-21-2020 15:05:49:000   0% /mnt/bos/alluxio-underfs-bos-0.1.0.jar
real 0m2.297s
user 0m2.703s
sys 0m0.269s
```

2. 统计单词"the"在文件/mnt/bos/LICENSE中出现的次数

```
 $ time ./bin/alluxio fs cat /mnt/bos/LICENSE | grep -c the
200
real 0m3.357s
user 0m2.974s
sys 0m0.289s
```

3. 读取过的BOS数据会放在内存中，再次读取时可以更快访问到，100%表示该文件已全部被加载到Alluxio内存中

```
 $ ./bin/alluxio fs ls //mnt/bos/LICENSE
-rwxrwxrwx       27040       PERSISTED 07-21-2020 15:06:46:000 100% /mnt/bos/LICENSE
```

4. 再次统计单词"the"在文件/mnt/bos/LICENSE中出现的次数，**3.357s → 2.189s**，第二次无需从BOS拉取数据，耗时明显缩短

```
 $ time ./bin/alluxio fs cat /mnt/bos/LICENSE | grep -c the
200
real 0m2.189s
user 0m2.835s
sys 0m0.286s
```

5. 再次list查看/mnt/bos目录下文件列表，**2.297s → 1.793s**，也会更快

```
 $ time ./bin/alluxio fs ls /mnt/bos
-rwxrwxrwx       27040       PERSISTED 07-21-2020 15:06:46:000 100% /mnt/bos/LICENSE
-rwxrwxrwx       51307896    PERSISTED 07-21-2020 15:05:49:000   0% /mnt/bos/alluxio-underfs-bos-0.1.0.jar
real 0m1.793s
user 0m2.630s
sys 0m0.238s
```

BOS Connector for PyTorch

概述

BOS Connector for Pytorch 是百度智能云 BOS 针对 AI 训练场景所提供的高性能数据读写工具。可为 PyTorch 训练任务提供高性能访问存储在 BOS 中的数据。其会自动优化 BOS 读取和列出请求，提升 PyTorch 训练时对 BOS 中数据的加载和 Checkpoint 读写性能。

BOS Connector for Pytorch 可以用来从 BOS 中读取用于随机数据访问模式的 BosMapDataset 数据集，也支持读取用于顺序数据访问模式的 BosIterableDataset 数据集。另外可使用该工具将 Checkpoint 数据直接保存到 BOS 中，而无需先保存到本地存储。

相比通过 bosfs 等挂载工具使用对象存储，BOS Connector for Pytorch 有以下优势：

维度	bosfs 等挂载工具	BOS Connector for Pytorch
性能	低，无针对性优化	高，对训练集数据加载和 checkpoint 数据的读写做特定优化
数据加载方式	需提前下载（预热）数据	支持流式加载
数据访问	需要转接，经过一遍 fs 转义	直接读写 BOS
配置复杂度	较复杂	提供简单配置，开箱即用

功能及使用说明

功能特性

支持的功能	说明
Pytorch 中的 Iterable-style datasets 数据集	支持通过 prefix 或 object 列表的方式构建数据集
Pytorch 中的 Map-style datasets 数据集	支持通过 prefix 或 object 列表的方式构建数据集
支持 checkpoint 接口	支持将 checkpoint 数据直接写入 BOS；支持从 BOS 中读取 checkpoint 数据

使用说明 支持的部署环境：

- Linux: x86-64
- Python: >=3.8
- PyTorch： >=2.0

安装 BOS Connector for Pytorch

以下内容为 Python3.8 版本安装 bostorchconnector 示例。

在Linux操作系统，使用pip安装bostorchconnector

```
$ pip3 install bostorchconnector
```

查看是否安装成功

```
$ pip3 show bostorchconnector
```

```
(env) root@instance-lvi05jy5:~/env# pip3 show bostorchconnector
Name: bostorchconnector
Version: 1.0.0
Summary: bostorchconnector, a Python package with a precompiled shared library
Home-page:
Author:
Author-email:
License:
Location: /root/env/lib/python3.8/site-packages
Requires:
Required-by:
```

配置 BOS Connector for Pytorch

配置访问凭证

支持以下3种方式，完成任一方式即可，优先级如下：

1. 使用静态配置文件

`$ mkdir -p ~/.baidubce/ && vim ~/.baidubce/credentials`

添加如下配置：

```
[Defaults]
Ak=xxx
Sk=xxx
```

具体配置项说明

配置项	是否必须	说明
Ak	是	百度云的AccessKey ID；
Sk	是	百度云的AccessKey Secret；
Sts	否	使用STS的临时方式凭证，如设置，AccessKey ID和AccessKey Secret也应是临时凭证。

> **注意：**
>
> - 默认配置支持用户级别的隔离，在某个用户下运行bostorchconnector会使用对应~/路径下的配置；如需统一配置，可以在bos_client_config的credentials_path配置绝对路径；
> - 使用STS临时访问凭证时，需自行维护有效期，使用过期的STS Token会导致访问失败。

2. 如果使用过bcecmd工具，且默认配置在~/.go-bcecli/credentials，无需额外操作

`$ cat ~/.go-bcecli/credentials`

3. 使用临时环境变量方式

```
export BCE_ACCESS_KEY_ID=xxx
export BCE_SECRET_ACCESS_KEY=xxx
```

构建 Dataset

bostorchconnector 当前支持两种 dataset：

- BosMapDataset：继承自torch.utils.data.Dataset
- BosIterableDataset：继承自torch.utils.data.IterableDataset

两种 dataset 均支持以下两种方式构建：

- from_prefix：通过存储前缀构建
- from_objects：通过已知的object列表构建

构建dataset时候的bos_client_config配置参数如下：

配置项	默认值	说明
credentials_path	"~/.baidubce/credentials"	存储访问凭证ak/sk的路径
log_level	1	日志级别，默认是INFO级别。0表示DEBUG；1表示INFO；2表示WARN；3表示ERROR：4表示FATAL；
log_path	"/tmp/bostorchconnector/sdk.log"	日志存储路径
part_size	8388608	分块上传时的分块大小，默认8MB
prefect_limit_mb	4096	预取缓存限制，单位MB，默认4096MB

构建适用于随机读取的BosMapDataset

BosMapDataset 适用于数据量不大，需要随机访问的数据集。

以 from_prefix 构建为例：

```
from bostorchconnector import BosMapDataset

##### 填充 <BUCKET>、<PREFIX> 和对应的endpoint
DATASET_URI="bos://<BUCKET>/<PREFIX>"
ENDPOINT="http://bj.bcebos.com"
config = BosClientConfig(log_level=1)

map_dataset = BosMapDataset.from_prefix(DATASET_URI, endpoint=ENDPOINT, bos_client_config=config)

##### 随机读index=0的样本
item = map_dataset[0]

##### 获取bucket、key、数据等信息
bucket = item.bucket
key = item.key
content = item.read()
len(content)
```

以 from_objects 构建为例：

```
from bostorchconnector import BosMapDataset

##### 填充 <BUCKET>、ObjectName和对应的endpoint
DATASET_URIS = [
    "bos://<BUCKET>/img001.jpg",
    "bos://<BUCKET>/img002.jpg",
    "bos://<BUCKET>/img003.jpg"
]
ENDPOINT="http://bj.bcebos.com"
config = BosClientConfig(log_level=1)

map_dataset = BosMapDataset.from_objects(DATASET_URI, endpoint=ENDPOINT, bos_client_config=config)

##### 随机读index=0的样本
item = map_dataset[0]

##### 获取bucket、key、数据等信息
bucket = item.bucket
key = item.key
content = item.read()
len(content)
```

构建适用于流式顺序读取的BosIterableDataset

BosIterableDataset 适用于数据量大，需要顺序处理的数据集。

以 from_prefix 构建为例：

```
from bostorchconnector import BosIterableDataset

##### 填充 <BUCKET>、ObjectName和对应的endpoint
DATASET_URI="bos://<BUCKET>/<PREFIX>"
ENDPOINT="http://bj.bcebos.com"
config = BosClientConfig(log_level=1)

iterable_dataset = BosIterableDataset.from_prefix(DATASET_URI, endpoint=ENDPOINT, bos_client_config=config)

##### BosIterableDataset是可迭代对象
for item in iterable_dataset:
    data = item.read()
    print(len(data))
    print(item.key)
```

以 from_objects 构建为例：

```
from bostorchconnector import BosIterableDataset

##### 填充 <BUCKET>、ObjectName和对应的endpoint
DATASET_URIS = [
    "bos://<BUCKET>/img001.jpg",
    "bos://<BUCKET>/img002.jpg",
    "bos://<BUCKET>/img003.jpg"
]
ENDPOINT="http://bj.bcebos.com"
config = BosClientConfig(log_level=1)

iterable_dataset = BosIterableDataset.from_objects(DATASET_URI, endpoint=ENDPOINT, bos_client_config=config)

##### BosIterableDataset是可迭代对象
for item in iterable_dataset:
    data = item.read()
    print(len(data))
    print(item.key)
```

读写 Checkpoint

bostorchconnector 支持直接对存储在 BOS 上的 checkpoint 做读写。

以 resnet 在某次 epoch 的结果写入和加载 checkpoint 为例：

```
from bostorchconnector import BosCheckpoint

import torchvision
import torch

##### 填充 <BUCKET>、<KEY>和对应的endpoint
CHECKPOINT_URI="bos://<BUCKET>/<KEY>/"
ENDPOINT="http://bj.bcebos.com"
config = BosClientConfig(log_level=1)
checkpoint = BosCheckpoint(endpoint=ENDPOINT, bos_client_config=config)

model = torchvision.models.resnet18()

##### 保存checkpoint到Bos
with checkpoint.writer(CHECKPOINT_URI + "epoch0.ckpt") as writer:
    torch.save(model.state_dict(), writer)

##### 从Bos读取checkpoint
with checkpoint.reader(CHECKPOINT_URI + "epoch0.ckpt") as reader:
    state_dict = torch.load(reader)

model.load_state_dict(state_dict)
```

性能测试

测试数据：107GB 数据集，1,000,000 张图片，平均每张 110KB

测试环境：百度云 bcc.c5.c8m16，CPU 数 8 个，内存 16GB，内网带宽 3Gbps

数据集类型：以 from_prefix 构建 BosIterableDataset 为例

测试结果：

	batch_size	num_workers	数据集类型	构建方式	结果
bostorchconnector	256	8	BosIterableDataset	from_prefix	2785 img/s
bosfs	256	8	BosIterableDataset	from_prefix	48 img/s

测试代码：

```
def transform(data):
    data.read()
    return data.key

@time_it
def test_bos():
    config = BosClientConfig()
    BOS_URI = "bos://bos-torch/img_1M/"

    dataset = BosIterableDataset.from_prefix(BOS_URI, endpoint="http://su.bcebos.com", transform=transform, bos_client_config=config, enable_sharding=True)
    dataloader = torch.utils.data.DataLoader(dataset, batch_size=256, num_workers=8)
    for step, key in enumerate(dataloader):
        print(key)
        pass
```

fsspec-bosfs

概述

fsspec-bosfs (以下简称 bosfs) 是一个基于 Python 接口的文件系统实现，用以便捷访问百度云对象存储 BOS 服务。通过 bosfs，用户可以使用标准的 fsspec 接口操作存储于 BOS 中的数据。

安装

使用 pip 安装 pyftpdlib

```
$ pip install bosfs
```

查看是否安装成功

```
$ pip show bosfs
```

快速开始

配置访问凭证

两种方式，优先级如下：

1. 初始化 BOSFileSystem 时通过参数指定

```
import bosfs
fs = bosfs.BOSFileSystem(endpoint='http://bj.bcebos.com', access_key='{your ak}', secret_key='{your sk}', sts_token=None)
```

2. 配置环境变量

```
export BCE_ACCESS_KEY_ID=xxx
export BCE_SECRET_ACCESS_KEY=xxx
export BOS_ENDPOINT=xxx
```

示例：基本读写

1. 通过 bosfs 列举 BOS 上的数据

```
import bosfs
fs = bosfs.BOSFileSystem(endpoint='http://bj.bcebos.com', access_key='xxx', secret_key='xxx')
res = fs.ls('/mybucket/')
print(res)
```

2. 通过 bosfs 读取 BOS 上的数据

```
import bosfs

fs = bosfs.BOSFileSystem(endpoint='http://bj.bcebos.com', access_key='xxx', secret_key='xxx')
with fs.open('/mybucket/README.md') as f:
    print(f.readline())
```

3. 通过 bosfs 写入数据到 BOS

```
import bosfs
import os

fs = bosfs.BOSFileSystem(endpoint='http://bj.bcebos.com', access_key='xxx', secret_key='xxx')
object_name = "file1"
data = os.urandom(10 * 2**20)
with bosfs.open(object_name, "wb") as f_wb:
    f_wb.write(data)
```

更多使用方法可参考 fsspec 文档。

示例：Ray

Ray是一个分布式的Python开源库，能帮助用户将自己的程序快速分布式化。在Ray之上运行的PyTorch、TensorFlow等训练器可以利用Ray的特性，在异构集群上做数据预处理、分布式训练，还可以使用Ray Data并行化进行数据加载、预处理和训练。

在Ray上，通过fsspec接口可以直接访问bos上的数据并解析。

1. 生成两个csv文件，上传到bos

```
Name,Age,Gender,Grade
Alice,14,F,88
Bob,15,M,92
Charlie,14,M,85
David,15,M,90
Eva,14,F,95
```

```
Name,Age,Gender,Grade
Fiona,15,F,89
George,14,M,87
Hannah,15,F,91
Ian,14,M,84
Julia,15,F,93
```

2. 使用ray.data去取csv文件

```
import bosfs
import ray
import pyarrow
from pyarrow._fs import PyFileSystem

fs = bosfs.BOSFileSystem(endpoint='http://bj.bcebos.com', access_key='xxx', secret_key='xxx')

#### 读取mybucket的csv_dir/目录下的多个csv文件
ds = ray.data.read_csv(paths="bos://mybucket/csv_dir/", filesystem=fs)

#### 输出数据集的schema
print(ds.schema())

#### 输出数据集的记录数
print(ds.count())

#### 输出前3行
ds.show(3)
```

3. 输出

```
2025-04-03 21:43:28,551 INFO worker.py:1752 -- Started a local Ray instance.
Column  Type
------  ----
Name    string
Age     int64
Gender  string
Grade   int64
10
{'Name': 'Alice', 'Age': 14, 'Gender': 'F', 'Grade': 88}
{'Name': 'Bob', 'Age': 15, 'Gender': 'M', 'Grade': 92}
{'Name': 'Charlie', 'Age': 14, 'Gender': 'M', 'Grade': 85}
```

示例：PyArrow

PyArrow是Apache Arrow库的一个python实现，通过提供高性能的内存列存储格式，可以让应用程序和数据无缝连接。 它实现了不同数据处理系统之间的高效数据交换。 通过将其存储接口委托给fsspec，PyArrow可以通过统一的接口访问各种存储后端。

当然，PyArrow也可以通过bosfs来便捷地访问BOS。

示例1：

```
import bosfs
import pyarrow
from pyarrow._fs import PyFileSystem
import pyarrow.dataset as ads

fs = bosfs.BOSFileSystem(endpoint='http://bj.bcebos.com', access_key='xxx', secret_key='xxx')

#### 读取mybucket的student.parquet文件
ds = ads.dataset("bos://mybucket/student.parquet", filesystem=fs)

#### 输出数据集的schema
print(ds.schema)

#### 输出数据集的记录数
print(ds.count_rows())

#### 输出第一条记录
print(ds.take([0]))
```

示例2：

```
import bosfs
import pyarrow
from pyarrow._fs import PyFileSystem
from pyarrow.fs import FSSpecHandler

fs = bosfs.BOSFileSystem(endpoint='http://bj.bcebos.com', access_key='xxx', secret_key='xxx')

#### 生成一个PyArrow filesystem
py_fs = PyFileSystem(FSSpecHandler(fs))

#### 通过PyArrow filesystem接口读取BOS数据
with py_fs.open_input_file("bos://mybucket/student.parquet") as f:
    f.read()
```

读写性能

测试环境

- 测试机器：16 核 32G 内存, 3Gbps 网络带宽

性能测试结果

单文件顺序读写性能

file_size_mb	write_speed_mbps	read_speed_mbps	write_time_s	read_time_s
1	9.672178	4.640667	0.104893	0.216561
4	24.867180	15.987999	0.160897	0.250713
10	30.681564	30.890233	0.343196	0.325457
128	75.634190	93.547005	1.692568	1.369047
512	80.359467	97.840121	6.371391	5.233074

多文件并发读写性能

file_nums	file_size_mb	threads_nums	write_speed_mbps	read_speed_mbps	write_time_s	read_time_s
1000	4	16	381.91	301.56	10.47	13.26

使用限制

- 当前 bosfs 暂不支持 async 模式

BOS-CLI命令行工具

概述

为了方便用户使用百度智能云存储服务的API，BCE提供了可以通过命令行工具BCE CLI（Command Line Interface）调用BOS服务的方式，该工具基于BCE Python SDK实现。

BCE CLI当前只包含了BOS CLI的功能，所以下文都用BOS CLI代指BCE CLI。BOS CLI工具用于在命令行环境下使用BOS，可以通过CLI完成Bucket的创建和删除，Object的上传、下载、删除和拷贝等功能。

旧版CLI命令行工具目前已停止更新，推荐您使用新版BOS-CMD命令行工具，新版命令行工具安装更方便，执行效率更高，且使用方法相同。详见BOS-CMD命令行工具。

安装BOSCLI

前提条件

BOS CLI命令行工具（即BCE CLI）基于**Python 2.7**开发，根据操作系统安装相应**Python 2.7**，目前支持Windows 和Linux。

操作步骤

1.准备Python环境。

- Windows环境

  在Python官网下载python27.msi后，双击安装。

- Linux环境

  以Ubuntu和Redhat为例：

```
  # Ubuntu
$ sudo apt-get install python python-dev python-setuptools python-pip

# Redhat
$ sudo yum install python python-dev python-setuptools python-pip
```

> **说明：** 安装完python后请先执行`python -V`查看python版本信息，确保python版本为2.7或之后的版本。对于使用百度智能云BCC Centos6.5系统的用户请按照后面的**用百度智能云BCC Centos6.5系统的用户安装方法**安装

2.安装BOS CLI工具。下载地址

- Windows环境

  （1）解压CLI工具包。

  （2）在cmd中执行如下命令：

```
  cd $ BCE_PATH
$ python setup.py install
```

  （3）将bce拷贝到用户需要执行程序的目录下方便使用。

```
  $ python $YOUR_DIR/bce
```

- Linux环境

  （1）解压CLI工具包。

```
  $ unzip bce-cli-0.10.8.zip
```

  （2）将bcecli的库安装到系统的python目录下。

```
  $ python setup.py install
```

用百度智能云BCC Centos6.5系统的用户安装方法:

1. 首先下载BCC Centos6.5镜像专用版CLI 。
2. 解压CLI工具包。

```
  $ unzip bce-cli-0.10.8-centos6.5.zip
```

3. 将bce-cli-0.10.8-centos6.5的安装目录加入环境变量中。

```
  export PATH=bce-cli-path:$PATH
```

> **说明：** 比如bce-cli-0.10.8-centos6.5.zip 解压后的路径为 /home/user1/bce-cli-0.10.8-centos6.5，您需要将路径/home/user1/bce-cli-0.10.8-centos6.5/加入环境变量中。

配置BOSCLI

通用配置

配置支持中文显示的CLI环境

如需要配置支持中文显示的CLI环境，请执行如下命令配置：

`$ export LANG=en_US.UTF-8`

> **说明：**可先执行`$ locale`命令查看当前环境的语言支持情况，若不是`LANG=en_US.UTF-8`，可执行上述命令进行修改。

配置AK/SK、Region、Host信息

使用BOS CLI工具之前，推荐先设置Access Key、Secure Key、Region和Host。可以通过-c/--configure来设置AK、SK、Region和Host信息。

```
 $ bce -c  [<conf-path>]
$ BOS Access Key ID [None]: Enter Your AK
$ BOS Secret Access Key [None]: Enter Your SK
$ BCE Security Token [None]: Enter Your Security Token
$ Default region name [bj]: Enter Your region
$ Default domain [bj.bcebos.com]: Enter Your host
```

> **说明：**
>
> - `<conf-path>`为本地文件目录，当用户有多个百度智能云账户时，通过指定`<conf-path>`将不同账号的配置文件写到不同目录下，执行命令时可以指定目录选择对应账户的配置。
> - 不指定`<conf-path>`时，CLI在配置完成后将配置文件自动写到当前用户主目录`~/.bce`；指定`<conf-path>`时，将配置文件自动写到指定目录。
> - 即使不设置AK/SK，您也可以通过BOS CLI访问public权限的bucket。
> - **当且仅当**您使用临时授权访问提供的AK/SK访问BOS时，才需要您提供 Security Token。
> - 指定Region之后您可以不指定Host，BOS CLI可自动拼接Host为Region.bcebos.com。
> - 如果您指定了Host，那么BOS CLI不再自动拼接Host，而是访问您指定的Host。
> - 可以填写None将AK/SK/Region以及Domain设置为空。当Region设置为空时默认值为bj；当Domain设置为空时默认使用Region拼接，当两个同时为空时默认访问**bj.bcebos.com**。

> **注意：**
>
> - 如果您使用的AK/SK是直接从百度智能云控制台的安全认证中获取的，请将Security Token设置为空或者none。

BOS相关配置

bce -c命令除了可以进行通用设置，还支持设置自动切换域名，另还支持设置BOS上传的相关属性，如断点续传有效期、是否使用HTTPS协议及分块上传的多线程数。

设置自动切换域名

BOS拥有多个Region，如果您需要在不同Region下面进行操作，CLI的"自动切换域名"功能可以根据您当前操作的Bucket所在Region，来自动切换域名。为了方便您的使用，CLI会默认开启此功能。

如果您想让您所以的操作请求都发送到您指定的域名或者Host（通过bce -c配置的domain），你可以通过bce -c命令关闭"自动切换域名"功能, 设置为no表示关闭此功能。

```
 $ bce -c [<conf-path>]
$ Default use auto switch domain [yes]:
```

> **说明：**取值可以是yes和no，yes代表开启根据Bucket所在Region自动切换域名，no表示关闭。缺省情况下为开启。

设置断点续传有效期

CLI支持文件上传的断点续传。当某次上传因为意外中断，您可以重新执行命令，继续上一次的上传。断点续传功能需要设置有效期，有效期是指在该时间内，如出现中断，重新上传时可续传断点之后的部分；如超过该有效期，则会开始一次新的上传。

```
 $ bce -c [<conf-path>]
$ Default breakpoint_file_expiration [7] days: Enter the period of keep breakpoint file
```

> **说明：**续传有效期应设置为大于等于0的整数，设为为"-1"表示永久保留。缺省续传有效期为7天。

设置HTTPS协议上传

为了保证用户数据的安全性，建议用户使用HTTPS协议上传。是否使用HTTPS可以通过bce -c命令进行设置，设置为no代表使用HTTP协议上传。

```
 $ bce -c [<conf-path>]
$ Default use https protocol [no]:
```

> **说明：**取值可以是yes和no，缺省情况下使用HTTP协议上传。

设置分块并行上传线程数

大文件分块上传时，可以通过bce -c命令设置并行上传的线程数。

```
 $ bce -c [<conf-path>]
$ Default multi upload thread num [10]:
```

> **说明：**并行上传的线程数应设置为大于等于1的整数，缺省为10。

通过CLI使用BOS服务

用户可以通过bos命令来访问BOS服务，管理、操作Bucket和Object。如果BOS服务余额不足时进行操作会提示Access Denied。

升级说明

BOS 命令行工具由BOS CLI 全新升级为BOS CMD,BOS CLI 工具停止功能迭代，推荐您使用BOS CMD进行替换。

CLI使用多层命令结构，所有的命令以bce开头，[options]表示BCE CLI支持的选项，<service>表示CLI所支持的服务，如BOS等，每个服务拥有多个特定的子命令。

```
 $ bce [options] [<service> <command> [parameters [options]]]
```

参数解释

参数项	取值	说明
options	BCE通用的Option，取值为-**c**、-**v**、-**h**、-**d**和--**conf-path**。	-
service	当前取值只能是**bos**。	-
command	取值为：**mb**、**rb**、**ls**、**cp**、**rm**。	各命令的作用请参见命令功能描述。
parameters	取值为**<local-path>**或**<bos-path>**。	<bos-path>包含以下几种形式： - BOS根目录：bos:/ - BOS Bucket根目录：bos:/<bucket-name> - BOS Bucket下文件夹：bos:/<bucket-name>/<folder-name>/ - BOS Bucket下的Object：bos:/<bucket-name>/<folder-name>/<object-key> 注：在<bos-path>中Bucket后面如果出现连续多个"/"会处理成一个"/"。 参数的值可以是简单的字符串或数字，如$ bce bos cp localfile.txt bos:/mybucket/remotefile.txt。如果字符串中包含空格及特殊字符，在Linux中需要使用单引号，在Windows中需要使用双引号包围。如$ bce bos cp 'my object 1' 'bos:/mybucket/my object 1'
+options	BOS命令的Option，取值为-**r**、-**a**或-**s**等。	BOS各命令可配置的option不一样，详细的使用方法请参见各命令行解释。

CLI通用Option

- 查看帮助：可以在任意命令后面添加-h或--help来查看该命令的帮助信息。

```
$ bce -h
$ bce bos -h
$ bce bos ls --help
```

- 查看、设置AK、SK等配置信息：-c/--configure

```
$ bce -c [<conf-path>]
```

- 查看版本信息：-v/--version

```
$ bce -v
```

- 查看debug信息：-d/--debug

```
$ bce -d bos ls bos:/bce-test/pre/
```

-d/--debug命令不能单独使用，必须与bos命令结合使用。

- 使用指定配置文件：--conf-path

```
$ bce --conf-path ./conf-sample/ bos ls bos:/bce-test/pre/
```

不使用--conf-path命令时，默认使用用户主目录~/.bce下的配置。

Bucket管理

创建Bucket

描述： 创建一个空的Bucket。

命令格式： $ bce bos mb bos:/<bucket-name> [--region REGION]

> **说明：**
>
> - 仅在"自动切换域名"功能开启后，您才能为Bucket指定所属地域。
> - 如果您没有指定Bucket所属地域，我们将使用您在CLI中配置的默认地域（即使用bce -c配置的默认地域，如果您没有配置默认地域，CLI的默认地域将为bj）。

示例：

- **不指定创建Bucket所属地域**

  操作示例：

  ```
  $ bce bos mb bos:/mybucket
  ```

  返回示例：

  ```
  Make bucket: mybucket
  ```

- **指定创建Bucket所属地域**

  操作示例：

  ```
  $ bce bos mb bos:/mybucket -r bj
  ```

  返回示例：

  ```
  Make bucket: mybucket
  ```

删除Bucket

描述： 删除一个Bucket，要求Bucket内无Object和未完成的三步上传Part。

命令格式： $ bce bos rb bos:/<bucket-name> [-y, --yes] [-f, --force]

> **说明：**
>
> - -y, --yes：使用该选项时，默认跳过确认步骤。
> - -f, --force：如果需要删除非空Bucket，可以使用-f/--force选项进行强制删除。使用该选项时，Bucket不为空也可以将该Bucket及其内所有Objects一起删除。

操作示例： $ bce bos rb bos:/mybucket --force

返回示例：

```
 Do you really want to REMOVE bucket bos:/bce-test and all objects in it? (Y/N) **Y**
Delete object: bos:/mybucket/test1.txt
Delete object: bos:/mybucket/test2.txt
Remove bucket: mybucket
```

罗列Bucket

描述： 罗列Bucket列表。

命令格式： $ bce bos ls [bos:/]

操作示例： $ bce bos ls

返回示例：

```
2015-08-12 10:20:58      bj  bce-doc
2015-08-12 10:21:09      bj  bce-sdk
2016-07-19 11:11:41      bj  bce-temp
2016-07-19 10:53:37      bj  bce-test
```

> **说明：**示例输出第一列代表bucket的创建日期，第二列代表创建时间，第三列代表bucket所在区域，第四列表示Bucket的名称。

Object管理

上传Object

描述： BOS CLI支持同步上传、单文件上传、批量上传和流式上传四种方式，上传过程中可以指定Object的存储类型是标准存储还是低频存储、冷存储或归档存储。此外BOS CLI还支持：

- 自动断点续传
- 对于大于32M的文件自动分块并发上传，并设置上传的并发连接数等。

当您需要在本地和BOS之间进行数据上传，同步上传（sync）是最方便的上传命令。同步上传默认支持批量操作，可以将本地目录同步到BOS。如果BOS端已经有同名文件，同步上传会忽略这些文件，只同步新文件或有改动的文件。 sync命令会同时列举本地和BOS侧文件, 对于不同情况分别执行以下动作：

- 对于新增的文件（本地的某个文件BOS侧没有）：上传
- 对于已上传但是有改动的文件（本地文件的更新时间晚于BOS侧文件的创建时间(精确到秒)；或者本地文件的更新时间与BOS侧文件的创建时间相同(精确到秒)，但是本地文件的大小跟BOS侧不一致）：覆盖上传
- 对于已上传且没改动的文件：跳过不处理
- 对于已上传但是在本地没有的文件：根据--delete参数来做不同的处理。

命令格式：

- 同步上传：`$ bce bos sync <local_dir> bos:/<bucket_name>/[prefix] [--delete] [--yes] [--exclude] [--quiet] [--dryrun] [--storage-class [STORAGE_CLASS]] [--exclude-time TIME_RANGE] [--include-time TIME_RANGE]`

  说明：

  - --dryrun：列出会执行的操作但不实际执行。
  - --quiet：同步上传但不输出任何中间过程。
  - --delete：删除BOS有而本地没有的文件，使BOS和本地的数据完全一致。执行后默认会提示用户是否要执行删除。
  - --yes：执行删除动作时不需要确认直接删除。
  - --exclude：执行本地到bos的sync的时候, 需要跳过的文件, 支持通配 符*和多项配置, 如要跳过tmp文件和当前目录所有子目录下的svn目录: `--exclude ./*.tmp --exclude */.svn`
  - --storage-class [STORAGE_CLASS]：通过storage-class参数指定同步上传的Object的存储类型。指定storage-class为`STANDARD_IA`、`COLD`或`ARCHIVE`时表示存储为低频存储、冷存储或者归档存储类型,指定的storage-class为`STANDARD`或者不指定storage-class时为标准存储。
  - `local_dir`必须为本地目录，不能为单个文件。也不支持通配符匹配的语法，如`<local_dir>/*` 以及`<local_dir>/prefix*`等。
  - --exclude-time：执行sync时，跳过在指定范围内创建或发生修改的文件，TIME-RANGE 表示您指定的时间区间，它的格式为 '[start, end)'。时间区间必须用 '[' 和 ')' 括起来（时间区间为左闭右右开，所以时间区间包含左边界时间，而不包含右边界时间）。 例：如果您不想同步在2017-01-02 03:04:05到2017-01-03 03:04:05之间新生成或者发生修改的文件，您可以使用 --exclude-time '[2017-01-02 03:04:05, 2017-01-03 03:04:05)'。
  - --include-time: 执行sync时，只同步在指定时间范围内生成或发生修改的文件， 它的用法和--exclude-time相同。
  - --exclude-time 和 --include-time 不能同时使用。

  时间区间格式说明：

  时间区间的格式为'[start, end)'，开始时间和结束时间可以是相对时间或绝对时间，也可以为空，但是不能同时为相对时间也不能同时为空。时间区间的具体格式如下：

时间格式	说明
开始时间或结束时间为空	如果开始时间为空，表示开始时间无穷小。如果结束时间为空，表示结束时间无穷大。例： '[, end)'表示开始时间为无穷小， '[start,)' 表示结束时间无穷大。
'now'	此变量为cli定义的魔法变量，表示当前时间。 例：如果现在是北京时间 2017-10-18 14:23:35， 那么时间区间'[2017-10-01 00:00:00, now)' 表示从2017-10-01 00:00:00 到2017-10-18 14:23:35这个时间范围（由于区间是左闭右开，所以不包含当前时间）。
'midnight'	和'now'一样，此变量为cli中定义的魔法变量，表示今天晚上的24:00:00。
本地时间	本地时间的格式为 'YYYY-mm-DD HH:MM:SS'， 比如2017-10-01 00:00:00。
UTC时间	UTC时间的格式为 'YYYY-mm-DDTHH:MM:SSZ'， 比如2017-01-02T23:04:54Z。
时间戳	可以用时间戳表示开始时间或者结束时间。
相对时间	遵循ISO8601中对时间段的表示，具体格式为 'PnnYnnMnnWnnDTnnHnnMnnS'（注意：并不是所有单元都需要，比如 P1D表示一天，更多信息请参考 ISO8601）。 相对时间可以与'now'和'midnight'一起组成绝对时间， 比如'midnight-P1D'表示昨天晚上24:00:00。当开始时间为相对时间时，比如您指定的时间区间为 [duration, end)， 其中duration表示相对时间， end表时间区间的结束时间 （由于时间区间必须包含一个绝对时间，所以end必须为绝对时间），那么此时间区间的开始时间应该为start = end - duration，结束时间为end。同理，当结束时间时为相对时间时，如果开始时间为start，那么时间区间的结束时间为end = start + duration

- 单文件上传：`$ bce bos cp <local-path> <bos-path> [--storage-class [STORAGE_CLASS]]`
- 批量上传：`$ bce bos cp <local-path> <bos-path> --recursive [--storage-class [STORAGE_CLASS]]`
- 流式上传： `$ bce bos cp - <bos-path> [--storage-class [STORAGE_CLASS]]`

BOS的四种上传方式都支持大文件断点续传功能，大文件续传的详细操作如下：

1. 设定续传有效期：

```
$ bce -c

$ Default breakpoint_file_expiration [7] days: Enter the period of keep breakpoint file
```

> **说明：**续传有效期应设置为大于等于0的整数，设为为"-1"表示永久保留。

2. 当出现传输中断后，重新执行一次上传命令，即可实现断点续传。如需要忽略断点续传，重头开始上传文件，可以通过在命令里增加--restart参数实现。

> **说明：**
>
> - 执行bce bos cp /tmp/file1 bos:/bucket/file2命令时，file2可以与file1相同，如果file2在bucket中已存在则会被覆盖。
> - 执行bce bos cp /tmp/file bos:/bucket/命令时，自动使用本地文件名作为bos端文件名，如果已存在重名文件会被覆盖。
> - 执行bce bos cp /tmp/file bos:/bucket/prefix[/] -r命令时，本地为文件时会退化为单文件上传，bos路径结尾为"/"时prefix会当做目录，新文件保存在prefix下一级；结尾不为"/"时会当做object名直接上传。
> - 执行bce bos cp /tmp/dir/ bos:/bucket/prefix[/] -r命令时，本地为目录时bos路径结尾不论是否有"/"都会以prefix为目录名在bos端新建目录； 并将dir中Object全部拷贝到prefix目录下。
> - 当<local-path>为"-"时，表示从流中读数据或者写数据到流中，可用管道进行连接。cp中的流操作只支持Linux环境操作，不支持Windows环境。从流中上传Object时，Object的大小最大仅支持5G。
> - 执行批量上传时，local-path只能是本地目录且以"/"结尾，bos-path必须为bucket/prefix。
> - 上传Object过程中可以通过storage-class参数指定Object的存储类型，storage-class默认为STANDARD即标准存储，storage-class为STANDARD_IA、COLD或ARCHIVE时表示存储为低频存储、冷存储或者归档存储类型。如果storage-class输入参数错误依旧会按照STANDARD进行存储，不会提示错误信息。

示例：

- **同步当前目录下的sample目录到bos:/bce-test/test下**

  操作示例： `bce bos sync ./sample/ bos:/bce-test/test`

  返回示例：

  ```
  Upload: sample/BOS_Batch_Delete.png to bos:/bce-test/test/BOS_Batch_Delete.png
  Upload: sample/sample.log to bos:/bce-test/test/sample.log
  ```

- **列出会执行的同步操作但不执行**

  操作示例： `bce bos sync ./sample/ bos:/bce-test/test --dryrun`

  返回示例：

  ```
  Upload: sample/BOS_DataImport.png to bos:/bce-test/test/BOS_DataImport.png
  ```

- **同步上传但不输出任何中间过程**

  操作示例：`bce bos sync ./sample/ bos:/bce-test/test --quiet`

  返回示例： 无返回

- **同步sample目录到bos:/bce-test/test上且删除BOS有而本地没有的文件**

  操作示例：`bce bos sync ./sample/ bos:/bce-test/test --delete`

  返回示例：

  ```
  Delete object: bos:/bce-test/test/sample.log
  ```

- **同步sample目录到bos:/bce-test/test上并存储为低频存储类型**

  操作示例： `bce bos sync ./sample/ bos:/bce-test/test --storage-class STANDARD_IA`

  返回示例：

  ```
  Upload: sample/BOS_DataImport.png to bos:/bce-test/test/BOS_DataImport.png
  ```

- **同步sample目录到bos:/bce-test/test上并存储为冷存储类型**

  操作示例： `bce bos sync ./sample/ bos:/bce-test/test --storage-class COLD`

  返回示例：

  ```
  Upload: sample/BOS_DataImport.png to bos:/bce-test/test/BOS_DataImport.png
  ```

  > **说明：** 当没有文件触发sync操作时，无返回。在windows和linux系统下，硬链接文件和普通文件的处理方式一样，软链接文件夹和普通文件夹的处理方式一样,但软链接文件(指向文件的软链接)和普通文件的处理方式不一样。windows系统中当sync包含软链接文件的目录时，软链接文件每次都会触发sync执行上传。linux系统下只有当软链接文件指向的文件有更改操作或者软链接文件本身被删除，才会触发sync。

- **同步当前目录下的sample目录到bos:/bce-test/test下， 但是只同步今天生成或发生修改的文件**

  操作示例： `bce bos sync ./sample/ bos:/bce-test/test --include-time '[P1D, midnight)'`

  > **说明：**
  >
  > - 今天早上00:00:00到晚上23:59:59 （包括23:59:59）。
  > - 由于时间区间为左闭右开，所以此时间区间不包含右边界时间midnight。

- **同步当前目录下的sample目录到bos:/bce-test/test下，但是只同步昨天生成或者发生修改的文件**

  操作示例： `bce bos sync ./sample/ bos:/bce-test/test --include-time '[P1D, midnight-P1D)'`

- **同步当前目录下的sample目录到bos:/bce-test/test下，但是只同步过去24小时内生成或者发生修改的文件**

  操作示例： `bce bos sync ./sample/ bos:/bce-test/test --include-time '[P1D, now)'`

- **同步当前目录下的sample目录到bos:/bce-test/test下，但是只同步前天和今天生成或者发生修改的文件**

  操作示例： `bce bos sync ./sample/ bos:/bce-test/test --include-time '[PT24H, midnight)' --include-time '[P1D, midnight-P2D)'`

  > **说明：**`--include-time` 可以多次使用。

- **同步当前目录下的sample目录到bos:/bce-test/test下，但是只同步本地时间2017年1月2号3时4分5秒到2017年1月3号3时4分5秒之内发生修改或者新生成的文件**

  操作示例： `bce bos sync ./sample/ bos:/bce-test/test --include-time '[2017-01-02 03:04:05, 2017-01-03 03:04:05)'`

  > **说明：** 由于时间区间为左闭右开，所以上面的两个时间区间均不包含时间2017-01-03 03:04:05， 如果需要包含此时间可以将此时间加1秒，表示为'[2017-01-02 03:04:05, 2017-01-03 03:04:06)'。

- **同步当前目录下的sample目录到bos:/bce-test/test下， 但是不同步前天和今天生成或者发生修改的文件**

  操作示例： `bce bos sync ./sample/ bos:/bce-test/test --exclude-time '[PT24H, midnight)' --exclude-time '[P1D, midnight-P2D)'`

  > **说明：** `--exclude-time` 可以多次使用。

- **单文件上传**

  操作示例：

  ```
  $ bce bos cp text.txt bos:/mybucket/test.txt
  ```

  返回示例：

  ```
  Upload: test.txt to bos:/mybucket/test.txt
  ```

- **批量上传**

  操作示例：

  ```
  $ bce bos cp ./temp/ bos:/bce-test -r
  ```

  返回示例：

  ```
  Upload: /home/users/username/temp/BAE.png to bos:/bce-test/BAE.png
  Upload: /home/users/username/tem/CreatePolicy.png to bos:/bce-test/CreatePolicy.png
  Upload: /home/users/username/tem/BAE_Pro.png to bos:/bce-test/BAE_Pro.png
  Upload: /home/users/username/tem/BCD.png to bos:/bce-test/BCD.png
  Upload: /home/users/username/tem/04Eng.png to bos:/bce-test/04Eng.png
  [5] object uploaded.
  ```

- **压缩文件输出至管道，从管道中读数据并上传Object**

  操作示例：

  ```
  $ tar -cz test.txt | bce bos cp - bos:/mybucket/test.txt.tar
  ```

  返回示例：

  ```
  $ Upload: stream to bos:/mybucket/test.txt.tar
  ```

- **上传Object并存储为低频存储类型**

  操作示例：

  ```
  $ bce bos cp text.txt bos:/mybucket/test.txt --storage-class STANDARD_IA
  ```

  返回示例：

  ```
  Upload: test.txt to bos:/mybucket/test.txt
  ```

- **上传Object并存储为冷存储类型**

  操作示例：

  ```
  $ bce bos cp text.txt bos:/mybucket/test.txt --storage-class COLD
  ```

  返回示例：

  ```
  Upload: test.txt to bos:/mybucket/test.txt
  ```

下载Object

描述： BOS CLI支持同步下载、单个文件下载、流式下载和批量文件下载。

同步下载默认支持批量操作，可以将BOS同步到本地目录。如果本地已经有同名文件，同步上传会忽略这些文件，只同步新文件或有改动的文件。

sync命令会同时列举本地和BOS侧文件, 对于不同情况分别执行以下动作：

- 对于新增的文件（BOS的某个文件本地没有）：下载
- 对于已下载但是有改动的文件（BOS侧文件的创建时间晚于本地文件的更新时间, 或者BOS侧文件的大小跟本地不一致）：覆盖下载
- 对于已下载且没改动的文件：跳过不处理
- 对于已下载但是BOS侧没有的文件：根据--delete参数来做不同的处理。

命令格式：

- 同步下载Object: `$ bce bos sync [--delete] [--yes] [--quiet] [--exclude] [--dryrun] <bos-dir> <local-dir> [--exclude-time TIME_RANGE] [--include-time TIME_RANGE]`
- 下载单个Object：`$ bce bos cp <bos-path> <local-path>`
- 批量下载Object：`$ bce bos cp <bos-path> <local-path> --recursive`
- 流操作下载Object：`$ bce bos cp <bos-path> -`

> **说明：**
>
> - 执行bce bos cp bos:/bucket/file /tmp/file命令时，本地文件名可以任意指定，若本地文件存在会提示是否覆盖。

- 执行bce bos cp bos:/bucket/file /tmp/命令时，本地文件名会自动存为bos端文件名，若存在会提示是否覆盖。
- 执行bce bos cp bos:/bucket/file /tmp/ -r命令时，bos端为文件时会退化为单个文件下载，若存在会提示是否覆盖。
- 执行bce bos cp bos:/bucket/prefix/ /tmp/ -r命令时，本地路径如果为一个已存在的文件会报错，本地目录不存在时会创建。
- 当<local-path>为"-"时，表示从流中读数据或者写数据到流中，可用管道进行连接。
- cp中的流操作只支持Linux环境操作，不支持Windows环境。
- 执行批量下载时，local-path只能是本地目录，bos-path必须为bucket/prefix。如果bos路径为Object，则为单个Object下载。
- 下载时不管是否指定参数storage-class，都不影响下载，该参数会被自动忽略，不起作用。
- 同步下载中，local-dir和bos-dir必须为目录，不能为单个文件。
- --dryrun：列出会执行的操作但不实际执行。
- --quiet：同步下载但不输出任何中间过程。
- --delete：删除本地有而BOS没有的文件，使本地和BOS的数据完全一致。执行后默认会提示用户是否要执行删除。
- --yes：执行删除动作时不需要确认直接删除。
- --exclude：执行bos到本地的sync的时候, 需要跳过的文件, 支持通配符*和多项配置. 例：--exclude 'bos:/mybucket/*tmp' --exclude 'bos:/mybucket/*.svn' --exclude 'bos:/mybucket/test**'。
- --exclude-time 与 --include-time: 用法与同步上传文件相同。

示例：

- **同步下载bos:/mybucket/pre/下文件到当前目录下的temp目录**

  操作示例：

  ```
  $ bce bos sync bos:/mybucket/pre/ ./temp/
  ```

  返回示例：

  ```
  Download: bos:/mybucket/pre/sts1.png to /home/users/username/temp/sts1.png
  Download: bos:/mybucket/pre/sts2.png to /home/users/username/temp/sts2.png
  Sync done: bos:/mybucket/pre/ to ./temp/, [2] success [0] failure
  ```

- **下载Object**

  操作示例：

  ```
  $ bce bos cp bos:/mybucket/test.txt text.txt
  ```

  返回示例：

  ```
  Download: bos:/mybucket/test.txt to test.txt
  ```

- **批量下载Object**

  操作示例：

  ```
  $ bce bos cp bos:/mybucket/pre/ temp -r
  ```

  返回示例：

  ```
  Download: bos:/mybucket/pre/sts1.png to /home/users/username/tmp/sts1.png
  Download: bos:/mybucket/pre/sts2.png to /home/users/username/tmp/sts2.png
  [2] objects downloaded.
  ```

- **下载Object至管道，从管道中读数据并解压**

  操作示例：

  ```
  $ bce bos cp bos:/mybucket/test.txt.tar - | tar -xz
  ```

  无返回

拷贝Object

描述： BOS CLI支持在bucket间拷贝文件，支持同步拷贝、单个文件拷贝和批量文件拷贝。

同步拷贝默认支持批量操作，如果目的Bucket已经有同名文件，同步拷贝会忽略这些文件，只同步新文件或有改动的文件。

sync命令会同时列举源Bucket和目的Bucket的文件, 对于不同情况分别执行以下动作：

- 对于新增的文件（源Bucket的某个文件目的Bucket没有）：拷贝
- 对于已拷贝但是有改动的文件（源Bucket文件的创建时间晚于目的Bucket文件的更新时间, 或者源Bucket侧文件的大小跟目的Bucket不一致）：覆盖拷贝
- 对于已拷贝且没改动的文件：跳过不处理
- 对于已拷贝但是源Bucket没有的文件：根据--delete参数来做不同的处理。

命令格式：

- 同步拷贝Object：`$ bce bos sync [--delete] [--yes] [--quiet] [--exclude] [--dryrun] <bos-dir> <bos-dir> [--exclude-time TIME_RANGE] [--include-time TIME_RANGE]`
- 拷贝单个Object：`$ bce bos cp <bos-path> <bos-path>`
- 批量拷贝Object：`$ bce bos cp <bos-path> <bos-path> --recursive`

说明：

- 执行bce bos cp bos:/bucket/obj1 bos:/bucket/obj2命令时，复制的两个路径相同时会报错，目的路径若已存在会被覆盖。
- 执行bce bos cp bos:/bucket/obj1 bos:/bucket/prefix/命令时，obj1会被拷贝到prefix目录下，目的路径必须以“/”结尾，否则会当做一个新文件名新建一个文件。
- 执行bce bos cp bos:/bucket/obj1 bos:/bucket/prefix/ -r命令时，源路径为单个文件时退化为单个文件拷贝，obj1会被拷贝到prefix目录下。
- 执行bce bos cp bos:/bucket/prefix1/ bos:/bucket/prefix2/ -r命令，实现目录到目录的拷贝，两个路径都必须以“/”的结尾；若目的路径结尾不以“/”结尾会当做单个文件从而报错。
- 执行批量拷贝时，bos-path必须为bucket/prefix。
- 用户必须具备对源bucket的读权限和目标bucket的写权限。
- 同步拷贝中，bos-dir必须为目录，不能为单个文件。
- --dryrun：列出会执行的操作但不实际执行。
- --quiet：同步上传但不输出任何中间过程。
- --delete：删除源Bucket有而目的Bucket没有的文件，使源和目的Bucket的数据完全一致。执行后默认会提示用户是否要执行删除。
- --yes：执行删除动作时不需要确认直接删除。
- --exclude：执行bos到bos的sync的时候, 需要跳过的文件, 支持通配符*和多项配置. 例：--exclude 'bos:/mybucket/*tmp' --exclude 'bos:/mybucket/*.svn' --exclude 'bos:/mybucket/test**'*。
- --exclude-time 与 --include-time: 用法与同步上传文件相同。

示例：

- **同步拷贝bos:/mybucket/pre/下文件到bos:/mybucket2/pre/**

  操作示例：

  ```
  $ bce bos sync bos:/mybucket/pre/ bos:/mybucket2/pre/
  ```

  返回示例：

  ```
  Copy: bos:/mybucket/pre/sts1.png to bos:/mybucket2/pre/sts1.png
  Copy: bos:/mybucket/pre/sts2.png to bos:/mybucket2/pre/sts2.png
  Sync done: bos:/mybucket/pre/ to bos:/mybucket2/pre/, [2] success [0] failure
  ```

- **拷贝单个Object**

  操作示例：

  ```
  $ bce bos cp bos:/mybucket/test.txt bos:/mybucket2/test.txt
  ```

  返回示例：

  ```
  Copy: bos:/mybucket/test.txt to bos:/mybucket2/test.txt
  ```

- **批量拷贝Object**

  操作示例：

  ```
  $ bce bos cp bos:/mybucket/pre1/ bos:/bce-test/pre/ -r
  ```

  返回示例：

  ```
  Copy: bos:/mybucket/pre1/s1.png to bos:/bce-test/pre/s1.png
  Copy: bos:/mybucket/pre1/s2.png to bos:/bce-test/pre/s2.png
  [2] objects remote copied.
  ```

罗列Object

描述： `ls`命令可以列出一个Bucket内的不超过1000条Objects，可以通过指定Prefix来过滤结果。`ls`会自动以/作为delimiter来达到模拟文件夹的效果。

命令格式： `$ bce bos ls bos:/<bucket-name>/[prefix]`

操作示例： `bce bos ls bos:/mybucket/web`

返回示例：

```
                           PRE  pre/
2016-04-01 16:43:01        8698     STANDARD  web_bos.png
2016-04-01 16:41:49       17188     STANDARD  web_server_bos.png
2016-04-01 16:43:53       24620  STANDARD_IA  webtobos_cors.png
2016-04-01 16:45:53       24623         COLD  cold_abc.png
```

说明

- -a, --all：超过1000个Object时会全部显示出来。
- -r, --recursive：不显示目录，直接显示下面的Object。
- -s, --summerize：显示个数以及总大小等统计信息。
- 示例输出第一列代表Object的创建日期，第二列代表创建时间，第三列代表Object的大小（以Byte为单位），第四列表示Object的存储类型，第五列表示Object的名称。
- pre/表示为文件夹，不会显示创建时间及大小信息。

获取Object的URL

描述： `gen_signed_url`命令可以获取指定Object的URL，根据提供的URL可以下载Object。

命令格式 `$ bce bos gen_signed_url bos:/<bucket-name>/<object-key> [--expires TIME]`

> **说明：**
>
> - 获取的URL地址输入浏览器中可以直接下载Object。生成URL前，需要确认配置的AK/SK与Bucket对应账户的AK/SK一致，否则生成的URL无效不能正常下载Object。
> - --expires: 指定URL过期时间（单位秒），值必须大于等于-1，其中-1表示URL永久有效。
> - 不指定过期时间时默认过期时间为1800秒。

示例：

- **不指定过期时间**

  操作示例：

  ```
  $ bce bos gen_signed_url bos:/bce-test/test/BOS_DataImport.png
  ```

  返回示例：

  ```
  http://bj.bcebos.com/bce-test/test/BOS_DataImport.png?authorization=bce-auth-v1%2Fd35a0b8750114e49b3567e0b0542c2f1%2F2016-08-25T02%3A03%3A57Z%2F1800%2F%2F96d77180ec2cb7e8de522c15ca42bf67c30bae0edb47bad495fe8e7953794a4f
  ```

- **指定过期时间**

  操作示例：

  ```
  $ bce bos gen_signed_url bos:/bce-test/test/BOS_DataImport.png -e 100
  ```

  返回示例：

  ```
  http://bj.bcebos.com/bce-test/test/BOS_DataImport.png?authorization=bce-auth-v1%2F637986f41b0046248e3a333817371502%2F2017-08-31T03%3A19%3A42Z%2F100%2F%2F3e77accca7f0206460c662b0a9596b7ec353599214d42b124bb797a48da7359a
  ```

删除Object

命令格式：

- 删除单个Object：`$ bce bos rm [-y, --yes] bos:/<bucket-name>/<object-key>`
- 批量删除Object：`$ bce bos rm [-y, --yes] [-r, --recursive] bos:/<bucket-name>/<prefix>`

> **说明：**
>
> - -y, --yes: 使用该选项时跳过确认步骤。
> - -r, --recursive：批量删除object。

示例：

- **删除单个Object**

  操作示例：

  ```
  $ bce bos rm bos:/bce-temp/s1.png
  ```

  返回示例：

  ```
  Do you really want to REMOVE object bos:/bce-temp/s1.png? (Y/N) **Y**
  Delete object: bos:/bce-temp/s1.png
  ```

- **批量删除Object**

  操作示例：

  ```
  $ bce bos rm bos:/bce-test/pre/ -r
  ```

  返回示例：

  ```
  Delete object: bos:/bce-test/pre/s1.png
  Delete object: bos:/bce-test/pre/s2.png
  [2] objects removed on remote.
  ```

BOSAPI接口管理

BOSAPI接口管理包括生命周期管理和日志管理。命令行通用格式为`bce bosapi [-h]`。

设置生命周期

描述： 按照提供的生命周期管理文件为指定的Bucket设置生命周期管理规则，或生成生命管理文件模板。

命令格式：

- 生成Transition的配置模板：`bce bosapi put-lifecycle --template`
- 按照指定的配置文件为Bucket设置生命周期管理规则： `bce bosapi put-lifecycle --lifecycle-config-file LIFECYCLE_CONFIG_FILE --bucket-name BUCKET_NAME`

示例：

- **生成Transition的配置模板**

  操作示例：

  ```
  $ bce bosapi put-lifecycle --template
  ```

  返回示例：

```
 {
  "rule": [
   {
     "action": {
     "name": "Transition",
     "storageClass": "STANDARD_IA"
    },
   "status": "enabled",
   "resource": [
   "${bucket_name}/${prefix}/*"
   ],
   "id": "sample-id",
   "condition": {
   "time:": {
     "dateGreaterThan": "$(lastModified)+P30D"
    }
    }
   }
  ]
}
```

- **按照配置文件1.txt为Bucket1设置生命周期管理规则**

  操作示例：

  ```
  bce bosapi put-lifecycle --lifecycle-config-file lifecycle_bj.json --bucket-name bucket1
  ```

  返回示例：

  > **说明：** 设置成功后会返回生命周期配置文件，否则报错。

获取生命周期配置

描述： 获取某个指定Bucket的生命周期配置。

命令格式： `bce bosapi get-lifecycle --bucket-name BUCKET_NAME`

操作示例： `bce bosapi get-lifecycle --bucket-name bucket1`

返回示例：

```
 {
  "rule": [
   {
     "action": {
     "name": "Transition",
     "storageClass": "STANDARD_IA"
    },
    "status": "enabled",
    "resource": [
     "bucket1/prefix1/*"
    ],
    "id": "sample-id",
    "condition": {
     "time:": {
      "dateGreaterThan": "$(lastModified)+P30D"
     }
    }
   }
  ]
}
```

删除生命周期配置

描述： 删除指定Bucket的生命周期管理配置。

命令格式： `bce bosapi delete-lifecycle --bucket-name BUCKET_NAME`

操作示例： `bce bosapi delete-lifecycle --bucket-name bucket1`

返回示例： 无

设置日志规则

描述： 为Bucket开启访问日志功能，并保存到指定Bucket的prefix中。

命令格式： `bce bosapi put-logging --target-bucket TARGET_BUCKET --target-prefix TARGET_PREFIX --bucket-name BUCKET_NAME`

操作示例： `bce bosapi put-logging --target-bucket bucket2 --target-prefix log --bucket-name bucket1`

返回示例： 无

获取日志规则

描述： 获取指定Bucket的访问日志配置情况。

命令格式： `bce bosapi get-logging --bucket-name BUCKET_NAME`

操作示例： `bce bosapi get-logging --bucket-name bucket1`

返回示例：

```
 $ bce bosapi get-logging --bucket-name bucket1
{
  "status": "enabled",
  "targetPrefix": "loggin/",
  "targetBucket": "bucket1"
}
```

删除日志规则

描述： 删除指定Bucket的访问日志设置。

命令格式： `bce bosapi delete-logging --bucket-name BUCKET_NAME`

操作示例： `bce bosapi delete-logging --bucket-name b1`

返回示例： 无

设置Bucket默认存储类型

描述： 设置某个指定Bucket的默认存储类型。

如果用户使用API、CLI或者SDK上传的Object未指定存储类型，则继承Bucket的默认存储类型。如果上传Object指定的存储类型和Bucket默认存储类型不一致时，以Object的存储类型为准。存储类型包含标准存储（STANDARD）、低频存储（STANDARD_IA）、冷存储（COLD）和归档存储（ARCHIVE）四种，具体使用场景和性能请参见分级存储，通过Console控制台设置Bucket默认存储类型请参见创建Bucket。

命令格式：

```
bce bosapi put-bucket-storage-class --bucket-name BUCKET_NAME --storage-class STORAGE_CLASS
```

操作示例：

```
bce bosapi put-bucket-storage-class --bucket-name bucket1 --storage-class COLD`
```

返回示例： 无返回

查询Bucket默认存储类型

描述： 该命令用于查询某个Bucket的默认存储类型，Bucket默认存储类型包括标准存储、低频存储和冷存储。

命令格式：

```
bce bosapi get-bucket-storage-class --bucket-name BUCKET_NAME
```

操作示例：

```
bce bosapi get-bucket-storage-class --bucket-name bucket1
```

返回示例：

```
{"storageClass":"COLD"}
```

获取文件meta信息

描述： 此命令可以用于判断object是否存在，也可用于获取objetc的meta信息。具体规则如下：

1. 如果您指定的object存在，bce 将输出此object的meta信息（输出的信息有 object名称、object大小、上次修改时间、object的存储类型），bce退出状态码为0；
2. 如果您指定的object不存在，bce将打印"Failed: The specified object does not exist"，并且退出状态码为2；
3. 如果bce执行时发生错误，bce将打印错误信息，并且退出状态码为1.

命令格式：

```
bce bosapi get-object-meta --bucket-name BUCKET_NAME --object-name OBJECT_NAME [--schema SCHEMA | --quiet]
```

> **说明：**
>
> - 如果文件存在，默认输出的meta信息有 object名称、object大小、上次修改时间、object的存储类型， 你也可以使用--schema定制需要输出的meta 信息。
> - --schema: 你可以指定要输出的meta信息，当前可以指定的值为 "object_name", "content_length", "last_modified", "bce_storage_class" （分表对应object名称、object大小、上次修改时间、object的存储类型）， 您也可以指定输出object的多个属性，各个属性名称之间用逗号分隔。 比如: 使用--schema "object_name,content_length" 指定输出object名称和object大小 ； 使用--schema "last_modified" 指定只输出object修改时间。
> - --quiet：bce 将不会输出任何中间过程（但是，如果发生错误，bce会打印错误信息）。

操作示例：

获取object的所有meta信息：

操作示例：

```
bce bosapi get-object-meta --bucket-name mybucket --object-name myobject
```

返回示例：

```
Success: {"object_name": "myobject", "content_length": "37", "last_modified": "Fri, 26 Jan 2018 08:51:38 GMT", "bce_storage_class": "STANDARD"}
```

获取object的大小和存储类型信息：

```
bce bosapi get-object-meta --bucket-name mybucket --object-name myobject  --schema "content_length,bce_storage_class"
```

返回示例：

```
Success: {"content_length": "37", "bce_storage_class": "STANDARD"}
```

获取不存在的object的meta信息：

操作示例：

```
bce bosapi get-object-meta --bucket-name mybucket --object-name noexist
```

返回示例：

```
Failed: The specified object does not exist
```

图像审核服务

描述： 对BOS上指定的图像进行审核，输出文字识别结果及图像类别归属的概率。

命令格式： `bce bos process img-censor bos:/<bucket-name>/<object-key> --parameter <local json file>`

说明：

- bos://:要审核的图像文件。
- --parameter: 本地json文件，内容是图像审核相关参数。参数是一个json描述，编码为无BOM的utf-8，如下所示：

```
 {
  "antiporn" : {},
  "terror" : {},
  "ocr" : {
     "detect_direction" : "false",
     "language_type" : "CHN_ENG",
     "recognize_granularity" : "big"
  }
 }
```

- parameter可以为多个子服务的按需任意组合，也可以为单个子服务如“鉴黄”。图像审核服务支持的子服务列表有：
  - ocr：通用文字识别
  - face：人脸探测
  - antiporn：色情识别
  - politician：政治人物识别
  - terror：暴恐识别
  - public 公众人物识别
  - webimage：网图OCR识别
  - disgust 恶心图识别
  - watermark：水印二维码识别
  - quality：图像质量识别 value为各底层服务参数，参照河图OCR通用文字识别服务入参。

操作示例：

1. 建立图像审核参数文件，假设存放在~/param.txt，其内容为

```
 {
  "ocr" : {
    "detect_direction" : "false",
    "language_type" : "CHN_ENG",
    "recognize_granularity" : "big"
  },
  "antiporn" : {}
 }
```

2. 调用cli

`bce bos process img-censor bos:/mybucket/img.jpg  --parameter ~/param.txt`

返回示例：

```
{
"result" : {
 "antiporn" : {
  "result" : [{
   "probability" : 0.000071,
   "class_name" : "色情"
  }, {
   "probability" : 0.000291,
   "class_name" : "性感"
  }, {
   "probability" : 0.999638,
   "class_name" : "正常"
  }
 ],
 "log_id" : 3066483724,
 "result_num" : 3
 },
 "ocr" : {
  "log_id" : 2471272194,
  "words_result_num" : 2,
  "words_result" : [{
   "words" : " TSINGTAO"
  }, {
   "words" : "青島啤酒"
  }
 ]
 }
},
"log_id" : 149803506446902
}
```


Android批量打包

描述： 按照BOS上的渠道配置文件对APK母包进行批量打包，生成多个渠道包。

命令格式：

- 批量打包：`bce bos process apk-pack bos:/<bucket-name>/<object-key> --parameter <local json file>`
- 批量打包并返回通知： `bce bos process apk-pack bos:/<bucket-name>/<object-key> --parameter <local json file> --url <notify url>`

> **说明：**
>
> - bos:// ： 是要打包的APK文件。
> - --parameter: 本地json文件，内容是打包相关参数。参数是一个json描述，编码为无BOM的utf-8。如下所示： { "algorithm" : "v1", "channelFile" : "", "targetPrefix" : "" }
>   - algorithm：打包算法，目前只有v1
>   - channelFile：打包的渠道文件,此渠道文件必须上传到bos上，可以为prefix或者直接文件名，渠道文件必须是一个TSV文件（tab seperated values），第一行必须为insertedFileName和distributedPackageName，其两列分别为空文件名、渠道包名，且两个字段之间必须以tab分隔，不可以用空格。示例如下： insertedFileName distributionPackageName file_123 my_123.apk file_222 bbb_222.apk
> - targetPrefix：打包输出APK所存放的prefix
> - --url：打包结束后，给 post 打包结果

操作示例：

假设要打包文件所在的bucket为mybucket，打包文件为/pre/mygame.apk，渠道文件为pre2/channelFile.tsv，打包输出APK所存放在mybucket下的p1/p2/。其中channelFile.tsv的内容为：

```
 insertedFileName distributionPackageName
file_123 my_123.apk
file_222 bbb_222.apk
```

1. 建立打包参数文件，假设存放在~/param.txt，其内容为

```
  {
   "algorithm" : "v1",
   "channelFile" : "pre2/channelFile.tsv",
   "targetPrefix" : "p1/p2/"
  }
```

2. 调用cli并返回回调接口

```
bce bos process apk-pack bos:/mybucket/pre/mygame.apk --parameter ~/param.txt --url http://myserver.com
```

返回示例：

```
{
  "code": 66,
  "requestId": "71a95440-c7b9-4503-b41a-a7644863d7dc",
  "jobIds": [
     "342-172707368",
     "341-172707368"
  ]
}
```

- 342-172707368：打包的jobid;
- 341-172707368: 回调参数的jobid。

当打包正常结束后，http://myserver.com 将收到结果请求：`code=success&jobId=342-172707368`。此时在mybucket下的p1/p2/将存放着已经打好包的文件my_123.apk和bbb_222.apk。

版本变更记录

- BCE CLI工具包 [2018-06-05] 版本号0.10.10

  变更记录：

  增加bosapi接口get-object-meta用于获取文件meta信息和判断文件是否存在;

- BCE CLI工具包 [2017-12-12] 版本号0.10.9

  变更记录：

  支持使用STS;

- BCE CLI工具包 [2017-11-02] 版本号0.10.8

  变更记录：

  - sync 支持指定和排除指定时间范围内的文件；
  - 支持三步COPY；

- BCE CLI工具包 [2017-09-12] 版本号0.10.7

  变更记录：

  为自动切换域名功能添加开关。

- BCE CLI工具包 [2017-09-05] 版本号0.10.6

  变更记录：

  - 支持命令`gen_signed_url`指定过期时间；
  - 支持创建Bucket时指定地域；
  - 去除CLI对Endpoint的依赖。

- BCE CLI工具包 [2017-08-17] 版本号0.10.5

  变更记录：

  - 支持设置Bucket的默认存储类型；
  - 优化sync性能。

- BCE CLI工具包 [2017-07-20] 版本号0.10.4

  变更记录：支持多配置。

- BCE CLI工具包 [2017-06-29] 版本号0.10.3

  变更记录：

  支持图像审核服务和Android批量打包服务。

- BCE CLI工具包 [2017-05-25] 版本号0.10.2

  变更记录：

  支持冷存储

- BCE CLI工具包 [2017-05-18] 版本号0.10.1

  变更记录：

  参数错误，修复bug。

- BCE CLI工具包 [2017-02-16] 版本号0.10.0

  变更记录：

  - 支持BOS->本地、本地->BOS及BOS->BOS的SYNC功能。
  - 支持BOSAPI接口管理，包含生命周期管理和日志管理。

- BCE CLI工具包 [2016-09-02] 版本号0.9.2

  变更记录：

  - 支持低频存储类型Object的上传和下载等功能。
  - 支持获取Object的URL地址。
  - 提升批量操作的性能。

- BCE CLI工具包 [2016-08-11] 版本号0.9.1

  变更记录：

  支持[sync功能]。

- BCE CLI工具包 [2016-07-22] 版本号0.9.0

  变更记录：

  - 支持设置HTTPS协议及分块上传并行线程数。
  - 支持批量上传、下载、拷贝和删除Object。

- BCE CLI工具包 [2016-03-15] 版本号0.8.3

  变更记录：

  支持大文件的断点续传功能。

- BCE CLI工具包 [2015-10-23] 版本号0.8.2

  变更记录：

  - 支持多线程上传。
  - 通过CLI完成Bucket的创建和删除，Object的上传、下载、删除和拷贝等功能、支持流式上传。

第三方工具

MinIO

概述

MinIO：一种支持Kubernetes部署的高性能对象存储解决方案。MinIO提供了与Amazon Web Services S3兼容的API，并支持所有核心S3功能。使用MinIO客户端，可以方便地实现bucket的创建、删除以及objcet的查看、复制、移动、删除等操作，除此之外，也可以管理bucket的生命周期、加密算法、版本控制等。

安装

前往https://min.io/download下载安装MinIO Client即可。

用法

通用的操作格式为：`mc [FLAGS] COMMAND [COMMAND FLAGS | -h] [ARGUMENTS...]`

绑定百度智能云账户 命令为：

```
mc alias set mys3 https://s3.bj.bcebos.com abcdefg ABCDEFG
```

其中mys3是设置的账户名，可以随意设置。https://s3.bj.bcebos.com 是S3域名，不同的Endpoint对应不同的S3域名，具体的对应关系可以参考这里。abcdefg和ABCDEFG分别为AK和SK，登录百度智能云控制台总览，点击右上方头像下面的安全认证即可查看或申请。 **bucket操作**

- 列出mys3下所有的bucket：`mc ls mys3`
- 在mys3下新建名为new-bu的bucket：`mc mb mys3/new-bu`
- 在mys3下删除名为new-bu的bucket：`mc rb mys3/new-bu` **object操作**
- 列出mys3/old-bu下所有的object：`mc ls mys3/old-bu`
- 文件上传：`mc cp /Users/zhangsan/Desktop/baidu.pptx  mys3/new-bu/baidu.pptx`
- 文件下载：`mc cp mys3/new-bu/baidu.pptx /Users/zhangsan/Desktop/baidu.pptx`
- 将old-bu下的test.jpg复制到new-bu下：`mc cp mys3/old-bu/test.jpg mys3/new-bu/test.jpg`
- 删除old-bu下的test.jpg：`mc rm mys3/old-bu/test.jpg` **其他**
- 命令中加上--json可以让结果显示为json形式。
- 输入`mc --help`可以查询到所有的操作指令。
- 详细的操作指令介绍可以参考这里。

rclone

概述

rclone是一个管理云存储上文件的命令行程序，也是云供应商web存储接口的一个功能丰富的替代方案。超过70种云存储产品支持rclone，包括百度智能云对象存储（BOS）、企业和消费者文件存储服务以及标准传输协议。rclone在Linux、Windows和Mac上广泛使用，第三方开发人员可以使用rclone命令行或API创建备份、恢复、和业务流程解决方案。

安装

前往这里根据使用的操作系统下载安装。

用法

配置 命令行输入`rclone config`指令，下面的步骤依次为：

```
 No remotes found, make a new one?
n) New remote
s) Set configuration password
q) Quit config
n/s/q> n
Enter name for new remote.
name> myBOS  #这里为新的远程连接指定名字

Storage> 4  #选择时请确认选择Amazon S3 Compliant Storage Providers开头的选项，具体数字可能会有所改变
provider> 34  #选择时请确认选择Any other S3 compatible provider(Other)，具体数字可能会有所改变
env_auth> 1

access_key_id> abcde #账户的Access Key，具体可以前往https://console.bce.baidu.com/iam/#/iam/accesslist查询
secret_access_key> ABCDE #账户的Secret Key, 具体可以前往https://console.bce.baidu.com/iam/#/iam/accesslist查询

region> 1  #请务必选择使用V4 signature的选项
endpoint> http://s3.bj.bcebos.com  #兼容S3协议的BOS服务域名，不同Endpoint对应不同的S3域名，详见https://cloud.baidu.com/doc/BOS/s/xjwvyq9l4
location_constraint>  #如不确定则不输入直接Enter
acl> 1  #选择你希望的权限模式，默认private

Edit advanced config?
y) Yes
n) No (default)
y/n> n

Current remotes:
Name                 Type
====                 ====
myBOS                s3

e) Edit existing remote
n) New remote
d) Delete remote
r) Rename remote
c) Copy remote
s) Set configuration password
q) Quit config
e/n/d/r/c/s/q> q  #如输入确认无误，则选择q退出配置
```

其中的配置信息可以根据个人需要进行改动，配置之后可根据提示退出。

指定访问方式

查看配置文件位置

`rclone config file`

指定访问方式为 Virtual-hosted–style

```
$ vi ~/.config/rclone/rclone.conf
```

在配置文件中新增如下的配置内容

```
$ force_path_style = false
```

通用flag

- `--help/-h` 帮助
- `-v` 显示执行时更为详细的步骤信息
- `-vv` 显示debug log定位错误
- `--interactive/-i` 执行前询问 **bucket**
- 查看所有的bucket: `rclone lsd myBOS:`
- 创建一个名为new-bu的新bucket: `rclone mkdir myBOS:new-bu`
- 删除一个名为new-bu的bucket: `rclone rmdir myBOS:new-bu`
- 强制删除一个名为new-bu的bucket（当bucket里存在文件时使用）：`rclone purge myBOS:new-bu`
- 请注意以上创建删除操作的自定义bucket名称须满足BOS的bucket名称定义规则，即：
  1. 只能包含小写字母、数字和"-"
  2. 开头结尾为小写字母或数字
  3. 长度在4-63之间
  4. 不同bucket之间名称不能重复
- bucket acl权限设置：
  - 在rclone config 最后的advanced config里选择yes进行设置
  - 所有的新建的bucket会自动按照bucket acl的权限生成，其他bucket不受任何影响

object

- 列出名为new-bu的bucket下所有的object： `rclone ls myBOS:new-bu`
- 文件上传：
  - `rclone copyto src dest` ## src为需上传的文件的本地地址, dest为目标BOS位置
  - 具体示例如下： `rclone copyto /Users/Desktop/baidu.pptx myBOS:new-bu/baidu.pptx`
  - 如需从url地址直接上传至云，可使用 `rclone copyurl https://example.com myBOS:new-bu`
- 文件下载：
  - `rclone copyto src dest` ## src为需下载目标BOS位置, dest为目标保存地址
  - 具体示例如下： `rclone copyto myBOS:new-bu/baidu.pptx /Users/Desktop/baidu.pptx`
- 文件删除： `rclone delete myBOS:new-bu/baidu.pptx`

文件夹同步至bucket

- 文件夹同步命令为 `rclone sync src dest`，src表示同步的源，dest为目标云存储位置
  - 例如将本地文件夹Desktop同步至new-bu: `rclone sync ~/Desktop myBOS:new-bu`
- 利用flag进行特殊规则的同步措施：
  - `--size-only` 仅当源文件与目标文件大小不同时更改目标文件，无额外事务
  - `--checksum` 仅当源文件与目标文件的大小或md5校验和不同时更改，无额外事务
  - `--update --use-server-modtime` 不使用额外事务，修改时间变为对象上传时间，适用于许多操作，但可能错过时间戳在过去创建的文件。

清理未完成的分部上传文件

- 清理24小时前的未完成的分部上传文件： `rclone cleanup myBOS:test-bu`
- 清理1小时1分钟1秒前未完成的分部上传文件，时间可调: `rclone backend cleanup myBOS:test-bu -o max-age=1h1m1s`

多线程flag

- `--multi-thread-write-buffer-size=SIZE`
  - 设置buffer size为SIZE，默认128k
  - 受磁盘读写限制可调整，推荐16k的倍数
- `--multi-thread-chunk-size=SizeSuffix`
  - 用于多线程传输的分块大小通常由后端设置。
  - 但对于本地和smb等后端（实现了OpenWriterAt但未实现OpenChunkWriter），没有自然分块大小，此时使用该选项设置的值（默认为64Mi）。
- `--multi-thread-cutoff=SIZE`
  - rclone在传输大于指定SIZE的文件时会使用多线程，默认5G

    - 这只适用于作为目标的受支持后端，但将适用于作为源的任何后端。
- --multi-thread-streams=N
    - 该指令设定多线程传输时使用的流数
    - 设为0则禁用多线程传输，rclone默认值为4,但BOS推荐10。
- 如上flag皆为具体命令调整所用，如需永久设置，可在rclone config内进行到advanced config时选择yes进行永久设置 **更多用法** 详见https://rclone.org/commands/

常见问题

1. region步骤如选择非V4签名可能会出现中文名称文件无法在云端找到的报错。
2. region一旦选择非V4，重新使用rclone config修改会无法将修改保存进config文件，需使用rclone config file 查询具体config文件位置直接修改config文件。
3. 如果遇到只能下载无法上传的情况，可能原因为使用的AKSK的权限为只读，建议检查AKSK是否具有写入权限。

Goofys

工具概述

Goofys是一个用Go语言开发的适用于Linux和macOS系统的基于FUSE的轻量级开源挂载工具，可以将BOS存储桶挂载到本地文件系统中，目前仅支持少量的POSIX文件系统功能，更适用于只读的场景上使用。

主要特点

1. 仅支持顺序写入，不支持随机写。
2. 不支持符号链接或硬链接，不存储文件的mode/owner/group等属性。
3. 不支持rename包含超过1000个子目录的目录。
4. 创建时间（ctime）、访问时间（atime）与修改时间（mtime）总是相同。
5. 不依赖本地缓存，保证close-to-open一致性。

安装方法

- Linux

```
 curl -SL https://github.com/kahing/goofys/releases/latest/download/goofys -o goofys
chmod +x goofys
```

- macOS

```
 ##### 安装方式一
brew install --cask osxfuse
brew install goofys

##### 安装方式二
export PATH=$PATH:$GOPATH/bin
git clone https://github.com/kahing/goofys.git
cd goofys
make install
goofys -v
```

配置挂载

1. 创建访问凭证文件和所在目录

```
 mkdir ~/.aws
vim ~/.aws/credentials
```

2. 配置访问AK/SK到credentials文件中

```
 [default]
aws_access_key_id = BOS的Access Key ID
aws_secret_access_key = BOS的Secret Access Key
```

3. 挂载北京地域的BOS gs-bucket到本地目录/mnt/bos-gs-bkt

```
 mkdir /mnt/bos-gs-bkt
goofys --endpoint https://bj.bcebos.com gs-bucket /mnt/bos-gs-bkt
```

4. 进入/mnt/bos-gs-bkt目录查看挂载结果，可以直接对gs-bucket内object进行读写和List操作

注意事项

1. macOS brew安装osxfuse可能存在版本不兼容问题Error - The FUSE for macOS installation package is not compatible with this version of macOS，请参考osxfuse文档手动安装正确的版本。
2. goofys也支持挂载bucket某个prefix到本地目录，把替换成<bucket:prefix>即可。
3. Linux环境中mount操作需要root权限，若不是root用户，需要使用sudo或者以root用户身份运行goofys进行挂载，否则会报错：main.FATAL Unable to mount file system, see syslog for details；详细报错日志可查看/var/log/messages文件。
4. 可通过umount命令卸载已挂载的挂载点，如：umount /mnt/bos-gs-bkt。

S3 Browser

概述

S3 Browser 是适用于 Windows 平台的图形化客户端，其支持管理 S3 和兼容 S3 API 的云存储服务。S3 Browser 可以通过图形化的方式操作管理对象存储中的数据。

如需要在 Windows 平台下同时使用和管理多家云存储服务时，可以使用 S3 Browser，其常用功能有：

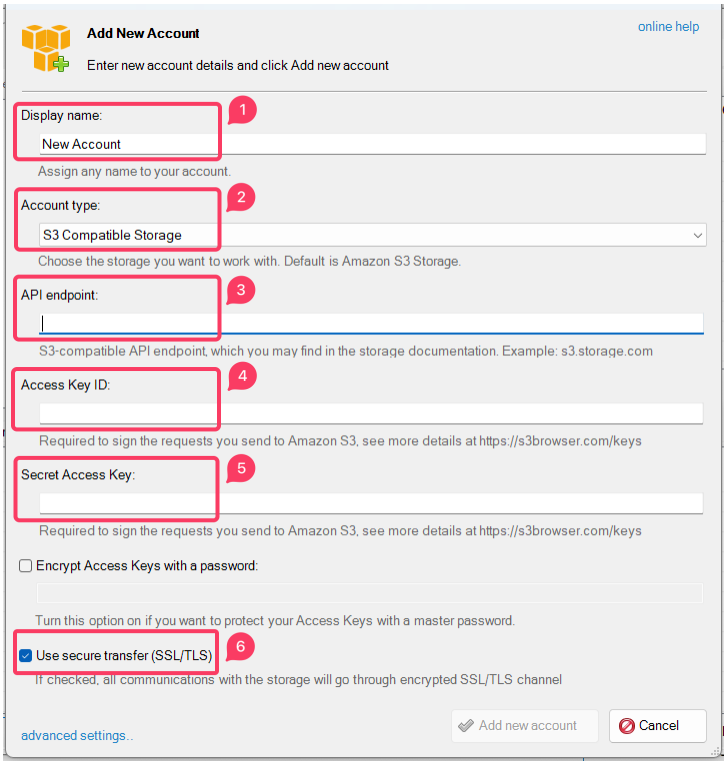
- 账户配置管理

- 创建和删除存储桶
- 查看 Bucket 信息
- 批量上传下载文件
- 复制/移动/重命名文件和文件夹
- 上传下载带宽限速
- 预览文件
- 客户端操作日志
- 同步数据

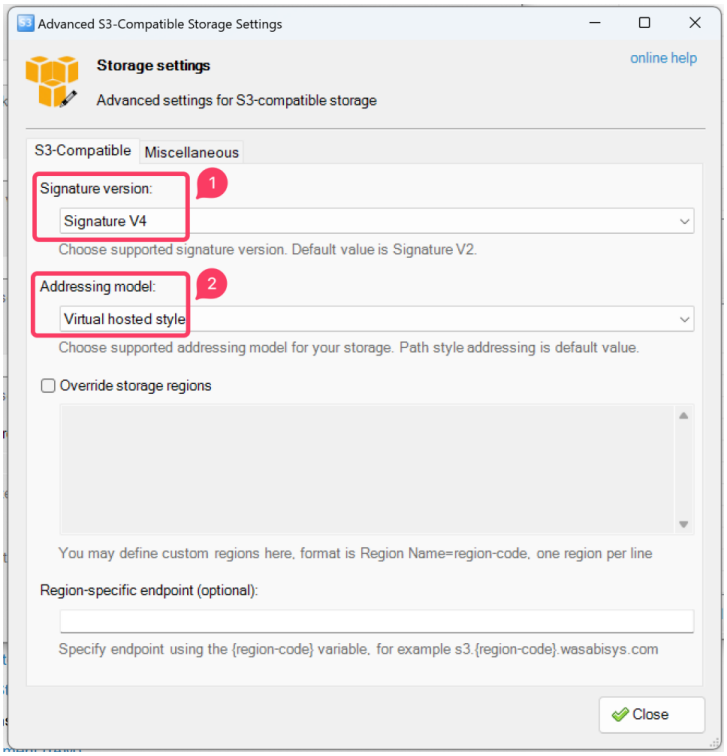
安装和配置

下载安装 访问 S3 Browser 官网 下载并安装。

配置 S3 Browser 管理 BOS



配置项	是否必填	输入/选择的内容	说明
Display name	是	BOS	自定义内容，为便于管理，建议输入为 BOS
Account type	是	S3 Compatible Storage	使用 S3 Browser 管理 BOS 时，该项必须选“S3 Compatible Storage”
API endpoint	是	https://s3.bj.bcebos.com	使用 BOS 的 S3 兼容域名
Access Key ID	是	百度云 Access Key ID	需具备访问 BOS 的权限
Secret Access Key	是	百度云 Secret Access Key	需具备访问 BOS 的权限
Use secure transfer (SSL/TLS)	否	勾选	建议勾选



配置项	是否必填	选项	说明
Signature version	是	Signature V4	BOS 兼容 AWS Signature Version 4
Addressing model	是	Virtual hosted style	必须选择该方式

配置完成后，即可通过 S3 Browser 管理百度云 BOS 对象存储。

其他工具

> **说明：**第三方工具主要由论坛的第三方开发者提供，由于这些工具并非官方提供，可能会存在一些bug，请大家谨慎使用。

- Wordpress插件：Wordpress是主流的博客建站工具之一。博客里的图片、音频、视频内容通常会占用大量空间,该插件可以把博客里的图片自动存储到BOS云端，一方面解决本地磁盘占用，一方面提高图片等多媒体浏览速度。
- Discuz插件：Discuz是主流的论坛建站工具之一，人们可以在论坛里发帖并上传附件。附件越来越多就会占用本地磁盘大量空间，所以附件的存储一直是个问题。该插件就是解决该问题的，它可以把论坛附件存储在BOS云端，一方面解决本地磁盘占用，一方面提高附件下载速度。
- UCenter插件：UCenter是Comsenz旗下的一个插件。借助UCenter站长可以无缝整合Comsenz系列产品，实现用户的一站式注册、登录、退出以及社区其他数据的交互的php框架。该插件的描述详细介绍了如何改写UCenter的部分代码把用户的头像图片上传到BOS，从而节约本地存储占用并提高头像的显示速度。
- Python flask：Flask是一个基于Python的web服务框架，使用Flask可以用几行代码实现复杂的web服务，该工具帮助用户在Flask框架里基于BOS Python SDK实现简单的文件上传、下载、删除操作。

典型实践

数据迁移

场景简介

数据上云的过程，可以分为“本地数据上云”、“从其他云迁移到 BOS ”两种场景，每种场景下都有存量和增量数据。百度智能云对象存储 BOS 提供了多种数据迁移方式，在数据一致性及无缝切换的前提下，尽量减少额外开销。您可以根据自身的业务情况，灵活选择数据迁移方式。

迁移场景	数据类型	迁移方案
本地数据上云	存量数据	BOS CMD 工具（在线）
		月光宝盒（离线）
		硬盘寄送迁移（离线）
	增量数据	BOS CMD sync 方式
从其他云迁移	存量数据	BOS Import 工具
		数据流转平台 CloudFlow
	增量数据	镜像回源

本地数据迁移到BOS

概览

针对本地数据上云的典型实践操作。如您的数据存储在自建 IDC，或存储在本地托管机房，您可以参考该篇指南进行数据的上云迁移。

需求场景

需求场景1：历史存量数据迁移上云

存量数据的特点是数据量一般比较大，迁移需要较长的时间，但数据基本固定，很少再进行修改。本地数据上云时，您可以根据自身网络条件选择在线方式或离线方式进行迁移。

需求场景2：增量数据迁移上云

当您在迁移历史存量数据的同时，您的源端可能还在源源不断的写入新数据。针对这部分增量数据，BOS 也提供相应的数据上云方案。

方案概述

BOS 提供命令行 CMD 工具，月光宝盒物理设备，硬盘寄送等多种数据迁移方式，您可以根据自己的需求选择合适的迁移方案。

实践操作

针对历史存量数据的操作

方式1:BOS CMD 工具在线迁移上云

本地数据上云的在线方式适用于客户本地网络条件较好（带宽充裕）的情况。在这种情况下，我们推荐通过 BOS CMD 工具的 Object 上传功能实现数据上云。

BOS CMD 是 BOS 提供的命令行工具，它提供了丰富的功能，方便用户对 BOS 资源进行各种操作和管理。 BOS CMD 支持批量上传、同步上传(sync)、单文件上传三种 Object 上传方式，使用批量上传和同步上传(sync)可以高效的解决本地存量数据在线上云的问题。在这里先介绍批量上传方式：

您可以在已安装了 BOS CMD 的机器上执行以下命令来使用批量上传：

```
$ bcecmd bos cp <local-path> <bos-path> --recursive [--storage-class [STORAGE_CLASS]] [--restart] [--quiet] [--yes] [--disable-bar]
```

其中 local-path 为本地上传文件的目录。具体操作及注意事项请参见：BOS CMD Object 上传功能

方式2:月光宝盒拷贝后物理寄送方式上云

如果客户侧的网络环境不好（带宽小、网络不稳定），且带宽成本非常昂贵时，在有限的带宽下进行传输，速度也不是特别理想，则推荐使用月光宝盒进行离线传输。

月光宝盒是百度智能云 BOS 提供的一种 TB 级数据传输解决方案，它使用存储设备在百度智能云和客户之间传输大量数据。它解决了大规模数据传输中出现的高昂网络成本、较长传输时间和安全等问题，且具有简单、快速、安全、成本低的优点。

和BOS提供的另一种离线方式“本地数据离线传输”相比，月光宝盒由BOS为您提供硬盘，因此，月光宝盒适用于迁移量大于等于 20TB 的情况。

月光宝盒正处于公测期间，使用前需要填写公测申请：月光宝盒公测申请

具体操作流程及注意事项请参见：月光宝盒操作流程

方式3: 自有硬盘寄送方式上云

BOS还提供了一种离线方式“本地数据离线传输”。它和“月光宝盒”的不同之处在于，离线传输仅支持客户自己提供硬盘。因此，离线传输适用于迁移量小于 20TB 的情况。

使用离线传输前，请先通过工单或者客户经理联系我们。您把数据拷贝到硬盘上后，请把硬盘寄送到下面地址：

地址	**北京市房山区窦店镇广茂路37号 光环新网数据中心**
电话	18110062722
联系人	BJDD 机房

硬盘寄回的地址，可以联系客户经理告知我们的接口人。硬盘寄送请选择保价，邮费请自负，寄回去时也会选择到付。北京同城1天，异地一般2天内可达。硬盘寄回之前，按照百度安全要求会统一清空硬盘，请关注。

针对源端增量数据的操作

方式1:BOS CMD sync 功能

BOS CMD 的同步上传(sync)功能可以方便快捷的解决本地数据上云过程中，增量数据的问题。同步上传默认支持批量操作，可以将本地目录同步到BOS。如果 BOS 端已经有名称相同且修改时间比本地文件新，同步上传会忽略此文件，只同步新文件或有改动的文件。 sync命令会同时列举本地和 BOS 侧文件, 对于不同情况分别执行不同的动作，以保证增量数据的准确安全上传。

您可以在已安装了 BOS CMD 的机器上执行以下命令来使用同步上传(sync)：

```
 $ bcecmd bos sync <local_dir> bos:/<bucket_name>/[prefix] [--exclude EXCLUDE] [--include INCLUDE] [--delete] [--exclude-delete EXCLUDE-DELETE] [--dryrun] [--yes] [--quiet] [--storage-class STORAGE-CLASS] [--sync-type SYNC-TYPE] [--concurrency CONCURRENCY] [--restart]
```

其中 local_DIR 为本地同步目录。具体操作及注意事项请参见：BOS CMD 同步上传(sync)功能

相关产品

- 数据流转平台 CloudFlow - 月光宝盒：月光宝盒适用于百 TB 级数据的传输和迁移上云，它使用特定存储设备在百度智能云和客户之间传输大量数据。它解决了大规模数据传输中出现的高昂网络成本、较长传输时间和安全等问题，且具有简单、快速、安全、成本低的优点。

月光宝盒由 BOS 为您提供移动存储设备，该设备具有大容量、可视化操作、军工级别防护加密等特性。单台月光宝盒设备的标定容量为 96 TB，实际使用容量为 83 TB 左右/每台，适用于百 TB 级别的本地 IDC 数据迁移上云场景。

- 对象存储 BOS - 硬盘寄送方式上云：它和“月光宝盒”的不同之处在于，设备需要由客户自行准备。您可以将数据写入自有硬盘，将硬盘批量寄送到百度智能云机房，我们将为您免费传输上云。该方案适用于百 TB 级别的本地 IDC 数据迁移上云场景。
- BOS CMD 同步上传(sync)功能：方便快捷的解决本地数据上云过程中，增量数据的问题。同步上传默认支持批量操作，可以将本地目录同步到BOS。如果 BOS 端已经有名称相同且修改时间比本地文件新，同步上传会忽略此文件，只同步新文件或有改动的文件。 sync命令会同时列举本地和 BOS 侧文件, 对于不同情况分别执行不同的动作，以保证增量数据的准确安全上传。

三方云数据迁移到BOS

概览

若您希望将其他云服务商中对象存储的数据迁移到 BOS,您可以使用 BOS 提供的如下若干方案。您可以根据自身需求选择合适的迁移方式。

需求场景

需求场景1:其他云服务商历史数据迁移

历史数据具有数据量较大的特点。针对其他云历史存量数据迁移到 BOS 的场景，百度智能云对象存储 BOS提供了两种迁移方式，分别是 BOS Import 工具和数据流转平台 CloudFlow。

需求场景2:其他云服务商增量数据迁移

当历史数据处于迁移过程时，源端可能还在源源不断的写入新数据。BOS 针对源端的增量数据提供迁移和同步方案，您可以使用 BOS Import 工具、数据流转平台 CloudFlow 和 BOS 镜像回源。

方案概述

针对客户从其他云迁移到 BOS 的场景，百度智能云对象存储 BOS 提供了三种迁移方式，分别是 CloudFlow 数据流转平台、BOS Import 工具、镜像回源。

- **数据流转平台 CloudFlow（推荐使用）**：是百度智能云存储推出的存储数据全托管迁移平台，支持可视化创建迁移任务，并对迁移任务的进度进行查看。您可以通过数据流转平台对其他云服务商中的历史存量数据和增量数据发起跨云迁移，也可以对本地自建的对象存储集群进行上云迁移。CloudFlow 支持公网和专线环境的数据迁移。

- **BOS Import 脚本化工具** ：具有可靠、安全、稳定的特点，您可以从 BOS 官网中下载并安装到本地，配置后即可发起数据迁移，适用于 1 TB 以上的数据迁移；
- **BOS 镜像回源**：您可以为存储桶 Bucket 配置镜像回源。当用户访问该 Bucket 中某一 Object 时，若发现该 Object 在 BOS 中并不存在，则 BOS 会立刻向预先配置好的回源地址请求该 Object，保证访问正常进行。在请求过程中，源站返回的数据会自动给到请求用户而用户对该过程无感。同时，该数据也会自动存入 BOS，提升日后该类请求的流畅性。

实践操作

历史存量数据跨云迁移

方式1：CloudFlow数据流转平台（推荐使用）

数据流转平台 CloudFlow 是百度智能云存储推出的存储数据全托管迁移平台，通过可视化配置的方式实现三方云到百度智能云的数据迁移。支持的迁移类型为三方对象存储数据到百度智能云对象存储，并支持对迁移任务的进度和结果进行查看。对失败的迁移任务，支持发起迁移重试，并支持对迁移任务发起中断、删除等操作。

数据流转平台支持如下几方面的功能：

- 任务创建：启动一个数据迁移进程前，需要先创建相应的迁移任务；
- 支持源端：数据流转平台支持由阿里云 OSS、腾讯云 COS、金山云 KS3、火山云 TOS、七牛云 KODO 、华为云 OBS 、AWS 中国区、本地自建 S3 集群移到百度智能云对象存储 BOS；
- 任务管理：当迁移任务创建后，您可以对任务进行管理，如中断、删除、下载失败文件、重启等操作；
- 任务进度查看：迁移任务创建后，您可以对任务的迁移进度进行查看，如当前已经完成迁移的文件数、完成中成功迁移的文件数、失败的文件数、待迁移的文件数等等；
- 支持存量数据和增量迁移的迁移。

您可以登录百度智能云 BOS 控制台，点击左侧导航中的数据迁移来创建迁移任务，也可以直接登录CloudFlow控制台创建迁移任务并发起数据的迁移。详细迁移步骤请参考CloudFlow帮助文档。

方式2：BOS Import工具

BOS Import 工具是百度智能云对象存储 BOS 推出的用于将其它云存储的数据迁移到 BOS 的工具，具有“可拓展性良好”、“支持迁移任务热启停”等多种优点。工具基于 C-S 结构实现，client 端负责读取文件列表, 任务调度和进度统计等工作, 所有和迁移相关的参数/配置都在 client 端进行；迁移任务在 server 端完成, client 端通过 RPC 请求通知 server 对哪些文件进行迁移。

具体使用方式及注意事项请参见：BOS Import 工具

源端增量数据的跨云迁移

您可以通过配置镜像回源规则或 CloudFlow 数据流转平台来完成增量数据的跨云同步。

方式1：CloudFlow数据流转平台

数据流转平台 CloudFlow 是百度智能云存储推出的存储数据全托管迁移平台，通过可视化配置的方式实现三方云到百度智能云的数据迁移，支持对存量数据和增量数据发起跨云迁移和跨云同步。

方式2：镜像回源

镜像回源是一项 Bucket 级别的配置，主要用于帮助用户在不停服的情况进行数据热迁移，解决从其他云迁移到BOS的增量数据问题。

对一个 Bucket 配置了镜像回源后，当用户访问（GetObject）一个 BOS 中的 Object，BOS 发现该 Object 在 BOS 中并不存在时，BOS 会向回源地址请求该 Object，并将源站所返回数据一边返回给请求用户，一边存入 BOS。

具体使用方式及注意事项请参见：BOS镜像回源

相关产品

- 数据流转平台 CloudFlow - 跨云迁移：是百度智能云存储推出的存储数据全托管迁移平台，通过可视化配置的方式实现三方云到百度智能云的数据迁移。支持的迁移类型为三方对象存储数据到百度智能云对象存储，并支持对迁移任务的进度和结果进行查看。对失败的迁移任务，支持发起迁移重试，并支持对迁移任务发起中断、删除等操作。支持由阿里云 OSS、腾讯云 COS、金山云 KS3、火山云 TOS、七牛云 KODO 、华为云 CBS 、AWS 中国区、本地自建 S3 集群移到百度智能云对象存储 BOS。

AWS-lambda同步S3数据到BOS

概览

利用 AWS Lambda 来实时同步用户上传到 S3 Bucket 的 Object 到 BOS Bucket。

需求场景

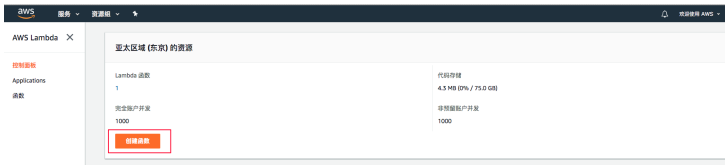
AWS Lambda 是一项计算服务，无需预配置或管理服务器即可运行代码，你可以通过配置 AWS Lambda的触发器来执行你上传的函数代码。因此，我们可以利用 AWS Lambda 来实时同步用户上传到 S3 Bucket 的 Object 到 BOS Bucket。

方案概述

您可以登录 AWS，使用 Lambda 设置针对文件上传的监听规则，并根据该操作触发文件同步到 BOS 的操作。

实践步骤

1. 登录AWS 控制台，进入Lambda服务控制台，点击“创建函数”

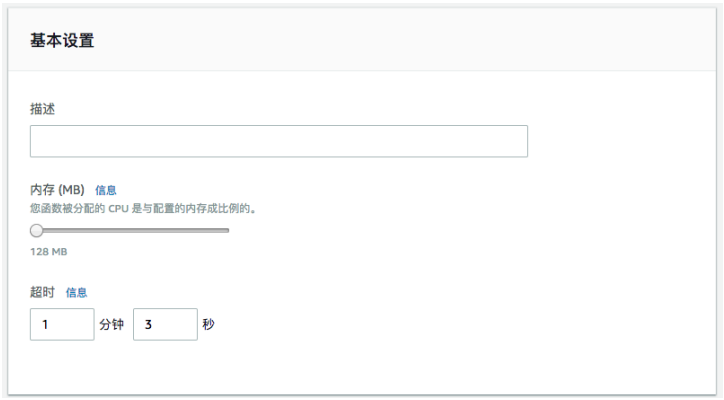


2. 从头开始创建函数，该示例中运行语言选择Go 1.x（后续示例lambda程序也是Go程序，用户可以选择自己擅长的语言来实现lambda函数），角色选择从模板创建新角色即可



3. 配置Lambda函数Desiger、函数代码、并发、内存分配、超时时间，环境变量等，按需配置

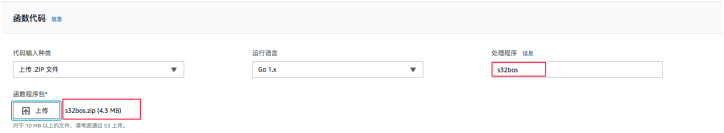
- **基本配置**：按需配置，例如你的代码执行需要的内存大小，超时时间、环境变量



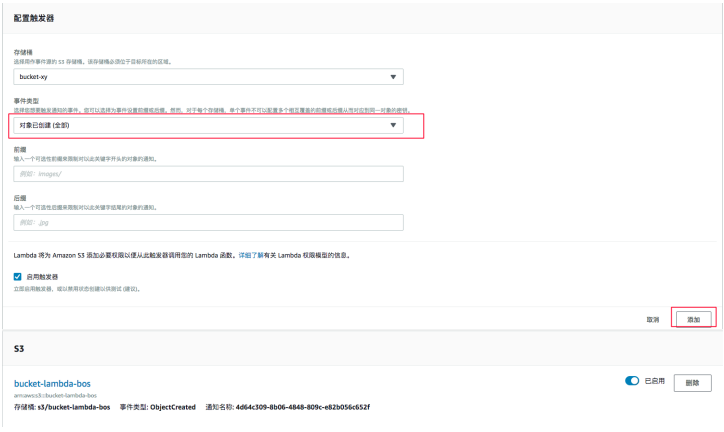
注意事项

1. 超时时间建议填1分钟以上，Lambda函数从S3下载文件，再上传到BOS耗时与Object大小有关，有时会比较长，按实际情况设置即可；
2. Lambda函数保存Object到磁盘，只能选择/tmp/目录，其他目录没有写权限；
3. 内存配置与函数运行时需要的内存大小有关，按实际需要设置即可。

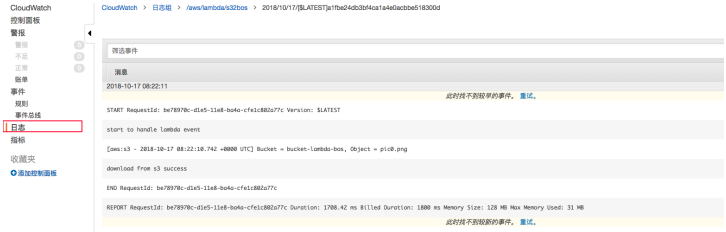
- **上传lambda代码**：上传代码必须为zip格式，处理程序填写Go文件的文件名



- **配置触发器**：我们选择S3触发器，进一步选择需要监听的S3存储桶，还能通过object前后缀缩小监听事件范围；添加之后会看到触发器内容为：存储桶: s3/bucket-name 事件类型: ObjectCreated



4. 保存配置，用户每次上传object到S3 bucket，就会执行上传的Go lambda代码，同步该object到BOS；进入CloudWatch控制台可以查看程序执行的日志



Lambda函数代码

- **库环境准备**：参考AWS S3 Golang sdk和BOS Golang sdk使用教程

```
  go get -u github.com/aws/aws-sdk-go/...
go get github.com/aws/aws-lambda-go/lambda
go get github.com/baidubce/bce-sdk-go
```

- 代码保存在s32bos.go文件中，编译成二进制文件之后压缩成zip文件上传，命令如下

```
  GOOS=linux GOARCH=amd64 go build -o s32bos s32bos.go
zip s32bos.zip s32bos
```

- **s32bos.go代码**：

```
 package main

import (
 "context"
 "fmt"
 "os"
)

import (
 "github.com/aws/aws-lambda-go/events"
 "github.com/aws/aws-lambda-go/lambda"
 "github.com/aws/aws-sdk-go/aws"
 "github.com/aws/aws-sdk-go/aws/credentials"
 "github.com/aws/aws-sdk-go/aws/session"
 "github.com/aws/aws-sdk-go/service/s3"
 "github.com/aws/aws-sdk-go/service/s3/s3manager"
 "github.com/baidubce/bce-sdk-go/services/bos"
)

var (
 S3_REGION = "ap-northeast-1"
 S3_AK     = "AKIAJ*****LWGRYAULYTA"
 S3_SK     = "ABZrl****RqhrE8VzvuESvQMWXcsOPiMJ7"

 BOS_AK       = "388e4e0b*****cfff3cbecb5"
 BOS_SK       = "2db53e648c53456e***5b7d378"
 BOS_ENDPOINT = "http://bj.bcebos.com"
 BOS_BUCKET   = "bucket-name"
)

func downloadFromS3(bucket string, object string) {
 file, err := os.Create("/tmp/" + object)
 if err != nil {
  fmt.Printf("Unable to open file, err %v", err)
 }
 defer file.Close()
 // init s3 downloader
 sess, err := session.NewSession(&aws.Config{
  Region:      aws.String(S3_REGION),
  Credentials: credentials.NewStaticCredentials(S3_AK, S3_SK, ""),
 })
 downloader := s3manager.NewDownloader(sess)

 numBytes, err := downloader.Download(file,
  &s3.GetObjectInput{
   Bucket: aws.String(bucket),
   Key:    aws.String(object),
  })
 if err != nil {
  fmt.Printf("Unable to download object %s, size %d, err %v", object, numBytes, err)
 }
 fmt.Println("download from s3 success")
}

func upload2Bos(bucket string, object string) {
 bosClient, err := bos.NewClient(BOS_AK, BOS_SK, BOS_ENDPOINT)
 if err != nil {
  fmt.Printf("create bos client err %v", err)
  return
 }

 etag, err := bosClient.PutObjectFromFile(bucket, object, "/tmp/"+object, nil)
 if err != nil {
  fmt.Printf("put object to bos err %v", err)
  return
 }
 fmt.Printf("upload2Bos success etag: %s", etag)
}

func HandleLambdaEvent(ctx context.Context, s3Event events.S3Event) {
 fmt.Println("start to handle lambda event")
 for _, record := range s3Event.Records {
  s3 := record.S3
  fmt.Printf("[%s - %s] Bucket = %s, Object = %s \n", record.EventSource, record.EventTime, s3.Bucket.Name, s3.Object.Key)

  downloadFromS3(s3.Bucket.Name, s3.Object.Key)
  upload2Bos(BOS_BUCKET, s3.Object.Key)

 }

 fmt.Println("handle lambda event success")
}

func main() {
 lambda.Start(HandleLambdaEvent)
}
```

相关产品

1. Lambda 函数处理程序 (Go)
2. AWS SDK for Go Developer Guide
3. BOS Go SDK
4. aws-lambda-go

如何解决浏览器跨域CORS问题

简介

浏览器同源策略(same-origion policy)

同源策略是Netscape公司在1995年引入浏览器的一个著名安全策略，它是浏览器最核心也最基本的安全功能，可以概括为本域脚本只能读写本域内的资源，而无法访问其它域的资源，以防止信息泄露。例如A, B两个网站属于两个不同的域(比如www.a.com和www.b.com)，A网站中的JavaScript脚本就只能访问A网站的资源，而不能访问B网站的资源，因为同源策略的限制使得跨域访问是会被浏览器拒绝的。

所谓同源是指域名相同、协议相同且端口相同。举例如下：

- 不同域名：`http://www.baidu.com`与`http://www.baidu.cn`为不同源；
- 不同协议：`http://www.baidu.com`与`https://www.baidu.com`为不同源；
- 不同端口：`http://www.baidu.com:80`与`http://www.baidu.com:81`为不同源；
- 其它：`http://www.baidu.com/a`与`http://www.baidu.com/b`为同源，因为域名协议和端口都相同。

什么是跨域资源共享(CORS)

在实际应用中会经常遇到跨域访问的情况，例如，用户的网站A(www.a.com)后端使用了BOS存储，用户想在该网站的Web应用程序中引用存储在BOS上的资源，但该页面只能请求本域资源，向BOS发送的请求会被浏览器限制，无法直接访问带来不便。为了解决这类跨域访问问题，HTML5提供了一套标准跨域解决方案即CORS。

跨域资源共享（Cross-Origin Resource Sharing，简称 CORS），它是由浏览器共同遵循的一套控制策略，通过HTTP的Header来进行交互。浏览器在识别到发起的请求是跨域请求的时候，会将Origin的Header加入HTTP请求发送给服务器，比如上面那个例子，Origin的Header就是www.a.com。服务器端接收到这个请求之后，会根据一定的规则判断是否允许该来源域的请求。如果允许的话，服务器在返回的响应中会附带上Access-Control-Allow-Origin这个Header，内容为www.a.com来表示允许该次跨域访问。如果服务器允许所有的跨域请求的话，将Access-Control-Allow-Origin的Header设置为*即可，浏览器根据是否返回了对应的Header来决定该跨域请求是否成功，如果没有附加对应的Header，浏览器将会拦截该请求。

CORS需要浏览器和服务器同时支持，整个CORS通信过程，都是浏览器自动完成，不需要用户参与。浏览器一旦发现AJAX请求跨域，就会自动添加一些附加的头信息，有时还会多出一次附加的请求。只要服务器实现了CORS接口，就可以跨域通信。

浏览器将CORS请求分成两类：简单请求（simple request）和非简单请求（not-so-simple request）。

简单请求是指请求方式是HEAD/GET/POST三种方式中的某一种，而且http头信息不超出以下几种字段：

- Accept
- Accept-Language
- Content-Language
- Last-Event-ID
- Content-Type：只限于三个值application/x-www-form-urlencoded、multipart/form-data、text/plain

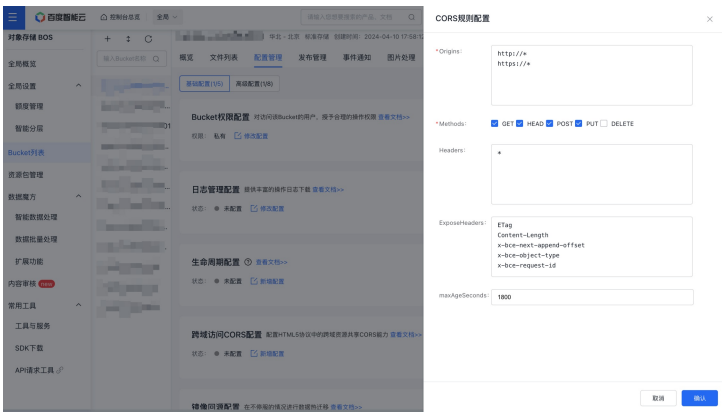
不符合以上条件的则是非简单请求。非简单请求的CORS请求，会在正式通信之前，增加一次HTTP OPTIONS查询请求，称为"预检"请求, 请求头信息里包括Origin、Access-Control-Request-Method和Access-Control-Request-Headers三个特殊字段，服务器收到"预检"请求以后，检查了三个字段以后，确认允许跨源请求，就可以做出回应。服务器的回应，都会有一个Access-Control-Allow-Origin头信息字段。

BOS配置CORS

BOS为开发者提供了两种方式来配置bucket资源的跨域访问权限，一种是直接在BOS控制台对bucket进行CORS规则设置；另一种是调用CORS相关的API接口来控制bucket资源的访问权限。

> **注意：**
>
> - BOS中CORS配置是在Bucket级别的；
> - CORS请求是否通过和BOS的身份验证是完全独立的，因为CORS规则仅仅是用来决定是否附加CORS相关的Header的一个规则，是否拦截该请求完全由浏览器决定。

- 控制台设置方法:
  1. 点击"基础配置"页签，选择"跨域访问CORS设置"并点击"修改配置"。
  2. 点击"确定"，保存规则



- API控制方法:
  1. PutBucketCors接口: 用来在指定的Bucket上设定一个跨域资源共享（CORS）的规则，如果原规则存在则覆盖原规则。
  2. GetBucketCors接口: 用于获取指定的Bucket当前的CORS规则。
  3. DeleteBucketCors接口: 用于关闭指定Bucket的CORS功能并清空所有规则。
  4. OPTIONS Object接口: 浏览器在发送跨域请求之前会发送一个preflight请求（OPTIONS）并带上特定的来源域，HTTP方法和Header信息等给BOS以决定是否发送真正的请求，此接口即响应这种请求。

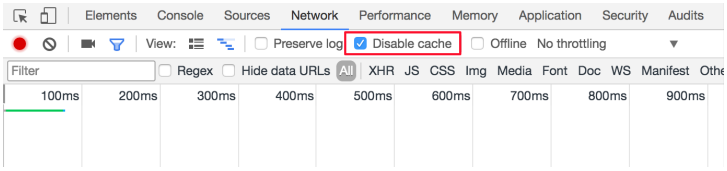
实战案例

以下我们会详细展示简单请求和非简单请求的跨域访问BOS资源时的情况。其中简单请求以GET请求为例， 非简单请求以POST请求为例。

准备条件

1. 登录BOS控制台新建一个Bucket，读写权限设置为“公共读写”，然后上传一个Object。在本例中，我们新建一个名为bos-demo的Bucket，然后上传一个bos.txt文件，文件内容为“This is a bos demo for CORS test !”。点击“复制链接”，可以看到bos.txt这个object的访问地址: http://bos-demo.bj.bcebos.com/bos.txt。

2. 关闭浏览器cache功能，防止因为浏览器缓存了服务器上次返回的header内容导致和CORS的要求不匹配，影响请求结果，这里我们以chrome浏览器为例, 打开“开发者工具”，勾选“Disable cache”。



跨域请求实例

1. 首先用curl访问bos.txt文件，可以发现该object已经可以正常访问。

   curl http://bos-demo.bj.bcebos.com/bos.txt

   This is a bos demo for CORS test !

2. 接着我们尝试使用Fetch API来访问这个object，可以将以下代码复制到本地保存成html文件然后使用浏览器打开，代码中提供了发送GET和POST两种请求的函数。

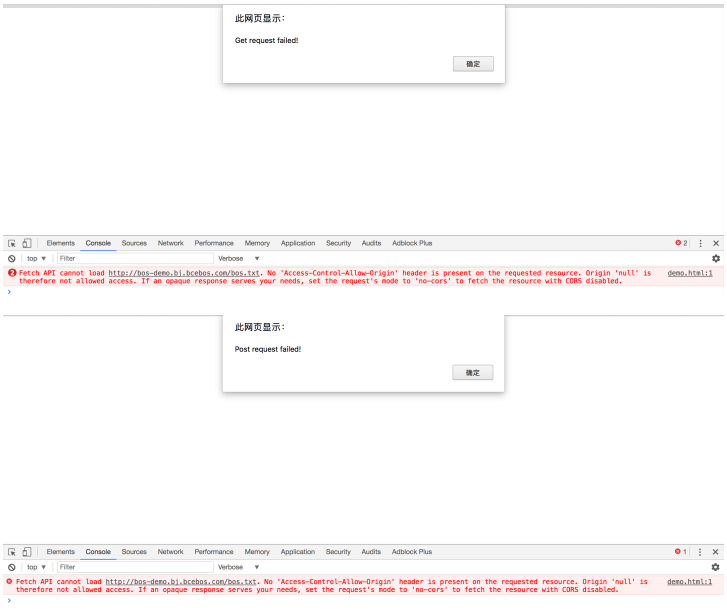
```
<!DOCTYPE html>
<html>
<body>
 <p align="center" style="font-size: 30px;">
  <button onclick="sendGetCorsRequest()">Send Get Request</button>
  <button onclick="sendPostCorsRequest()">Send Post Request</button>
 </p>

 <script type="text/javascript">
  var url = "http://bos-demo.bj.bcebos.com/bos.txt";

  function sendGetCorsRequest() {
   fetch(url).then(function(res) {
     if (res.ok) {
       alert("The response is ok, get request success!");
     } else {
       alert("The response wasn't ok, got status ", res.status);
     }
   }, function(e) {
     alert("Get request failed!", e);
   });
  }

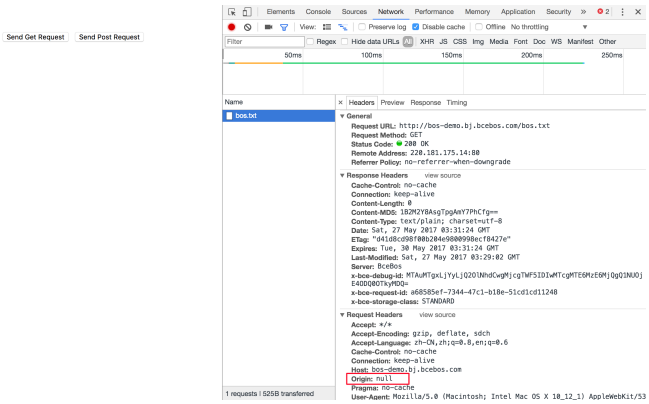
 function sendPostCorsRequest() {
  fetch(url, { method: "POST" }).then(function(res) {
     if (res.ok) {
       alert("The response is ok, post request success!");
     } else {
       alert("The response wasn't ok, got status ", res.status);
     }
   }, function(e) {
     alert("Post request failed!", e);
   });
  }
 </script>
</body>
</html>
```

打开html文件之后(以chrome为例), 点击“Send Get Request”或者“Send Post Request”按钮，会发现两个请求都发送失败。



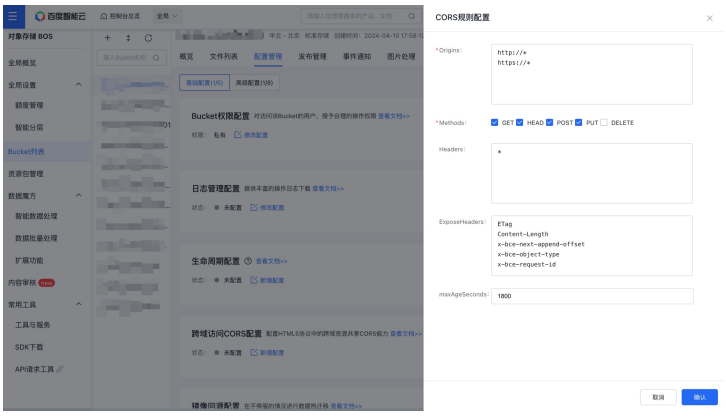
通过看chrome Console中报错提示我们看到，是因为没有找到Access-Control-Allow-Origin这个头。显然，这就是因为服务器没有配置CORS。

再进入Header界面，可见浏览器发送了带Origin的Request，因此是一个跨域请求，在Chrome上因为是打开的本地文件所以Origin是null。



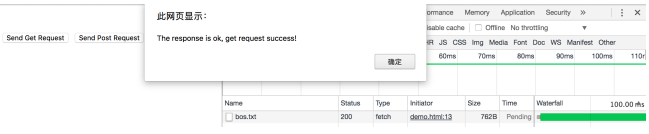
配置Bucket CORS规则

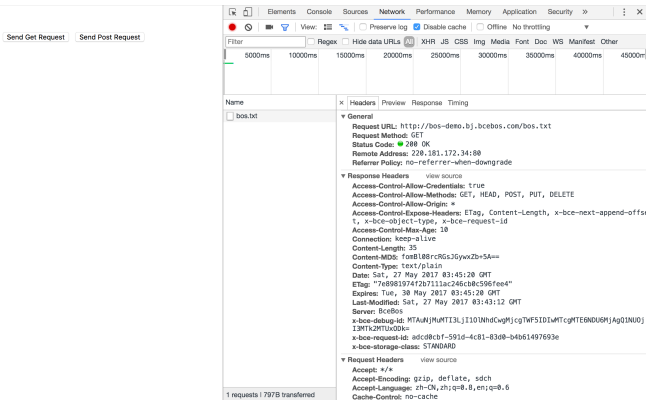
CORS设置是由一条一条规则组成的，真正匹配的时候会从第一条开始逐条匹配，以最早匹配上的规则为准。现在添加第一条规则，使用最宽松的配置：



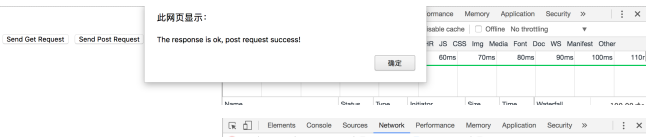
上图的配置代表着所有的Origin都允许访问，所有的请求类型都允许访问，所有的Header都允许，最大的缓存时间为10s，具体参数含义可以参考BOS产品配置文档。配置完成之后重新测试，结果如下：

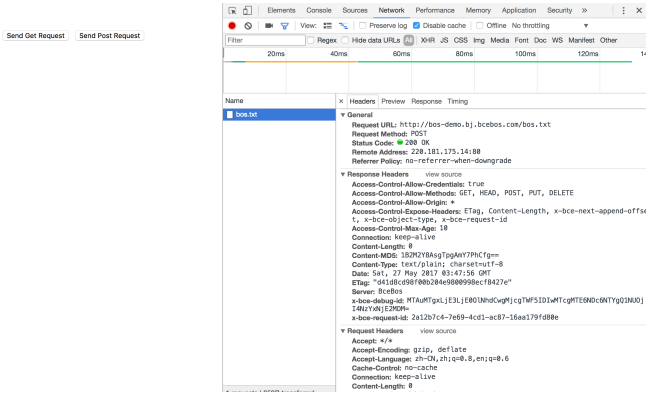
Get请求结果：





Post请求结果：





可以发现，当我们配置了CORS规则之后，POST和GET请求都可以成功发送了。 除了最宽松的配置之外，还可以配置更精细的控制机制来实现针对性的控制。比如对某个bucket只允许get请求，不允许其它请求，或者只允许某个域名访问该bucket等，这些都可以在控制台进行配置。对于大部分场景来说，用户最好根据自己的使用场景来使用最小的配置以保证安全性。

注意事项

CORS配置项有以下注意事项：

- Origins：配置的时候要带上完整的域信息，不要遗漏协议名如http，如果端口号不是默认的还要带上端口号。如果不确定的话，可以打开浏览器的调试功能查看Origin头。这一项支持使用通配符*，但是只支持一个，可以根据实际需要灵活配置。
- Methods：按照需求开通对应的方法即可。
- Headers：允许通过的Header列表，此建议没有特殊需求的情况下设置为*，大小写不敏感。
- ExposeHeader：暴露给浏览器的Header列表，不允许使用通配符。具体的配置需要根据应用的需求来选择，只暴露需要使用的Header，如ETag等。如果不需要暴露这些信息，这里可以不填。如果有特殊需求可以单独指定，大小写不敏感。

客户端加密实践

概览

客户端加密，是指由用户在本地自行完成文件的加密和解密过程，百度智能云对象存储不参与加密和解密过程，只负责文件的上传、存储、下载过程，明文密钥由用户自行保管在本地。客户端加密增强了文件安全性，即使文件意外泄露，别人也无法解密得到原始数据。本文档提供一种客户端加密方案。

> **注意事项：**
>
> - 由于用户需要自行负责保管密钥，如果明文密钥丢失，**您将无法获得原文件内容**。

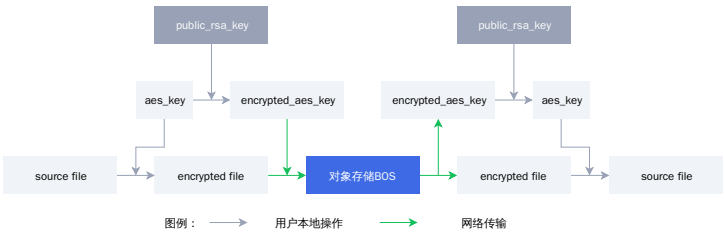
需求场景

客户自己希望在本地自行完成文件的加密和解密过程，切明文密钥由用户自行在本地保管。

方案概述

1. 本地使用RSA算法生成非对称密钥(private_rsa_key和public_rsa_key)；
2. 上传文件时，使用AES 256的CTR模式生成对称密钥（aes_key）；
3. 对于普通文件，使用对称密钥aes_key加密原始数据；使用公钥public_rsa_key加密aes_key生成encrypted_aes_key，作为object的meta，并上传文件到Bos。对于大文件，使用Bos的分块上传，每次取一个分块加密并上传，分块大小为16字节的整数倍，同样使用公钥public _rsa_key加密aes_key生成encrypted_aes_key，作为object的meta。
4. 下载文件时，首先获取文件的meta信息，得到encrypted_aes_key，利用本地私钥private_rsa_key解密encrypted_aes_key得到aes_key。对于普通文件，下载加密后文件并使用aes_key解密得到原始数据。对于大文件，可以使用分块下载，使用Python SDK的函数get_object(bucket_name,object_key,[range_start,range_end])下载大文件的指定区间[range_start,range_end]中的字节,包含结尾位置range_end处的字节，其中0≤range_start≤range_end≤文件大小。

> 注意：使用AES 256方式加密，AES算法加密时的块大小是128bits=16byte,因此在大文件分块上传时，每个分块大小应该是16字节的整数倍。



实践步骤

以下以 Python 代码进行示例。

准备工作

1.用户需要安装python SDk，请参考安装python SDK工具包。

2.执行下列命令，安装PyCrypto库。

```
pip install pycrypto
```

3.修改配置。修改示例代码中的HOST和AK、SK,作为本示例的配置项。

示例代码如下

```
#coding=utf-8

'''
The file aims to help client to encrypt data on python sdk with RSA algorithm  and symmetric encryption algorithm
'''

import os
import shutil
import base64
import random
import string

#引入配置文件和对象存储模块
from baidubce.bce_client_configuration import BceClientConfiguration
from baidubce.auth.bce_credentials import BceCredentials
from baidubce import exception
from baidubce.services import bos
from baidubce.services.bos import canned_acl
from baidubce.services.bos.bos_client import BosClient
```

```
#引入Crypto加密模块
from Crypto import Random
from Crypto.Cipher  import AES
from Crypto.Cipher  import PKCS1_OAEP
from Crypto.PublicKey import RSA
from Crypto.Util    import Counter

#设置对称密钥长度为128bits
_AES256_KEY_SIZE = 32
#设置AES CTR模式的计数器Counter长度
_COUNTER_BITS_LENGTH_AES_CTR = 8*16

class CipherWithAES:
    # start为CTR计数器初始值
    def __init__(self, key= None, start= None):
        if not key:
            key = Random.new().read(_AES256_KEY_SIZE)
        if not start:
            start = random.randint(1,100)
        self.key = key
        self.start = start
        #生成计数器
        my_counter = Counter.new(_COUNTER_BITS_LENGTH_AES_CTR, initial_value=self.start)
        #生成AES对象
        self.cipher = AES.new(self.key, AES.MODE_CTR, counter = my_counter)

    #加密数据
    def encrypt(self, plaintext):
        return self.cipher.encrypt(plaintext)

    #解密数据
    def decrypt(self, ciphertext):
        return self.cipher.decrypt(ciphertext)

class CipherWithRSA:
    # 输入参数为公钥文件名和私钥文件名
    def __init__(self, public_key_file_name = None, private_key_file_name = None):
        self.public_key_file_name=public_key_file_name
        self.private_key_file_name = private_key_file_name
        if not self.public_key_file_name:
            self.public_key_file_name = "rsa_public_key.pem"
        if not self.private_key_file_name:
            self.private_key_file_name = "rsa_private_key.pem"
        #如果没有输入公钥和私钥文件，则本地产生RSA密钥对
        if not (os.path.isfile(self.public_key_file_name) and os.path.isfile(self.private_key_file_name)):
            self._generate_rsa_key()
            return
        #从文件读取公钥和私钥，并产生RSA对象
        with open(self.public_key_file_name) as file_key:
            public_key_obj_rsa = RSA.importKey(file_key.read())
            self.encrypt_obj = PKCS1_OAEP.new(public_key_obj_rsa)
        with open(self.private_key_file_name) as file_key:
            private_key_obj_rsa = RSA.importKey(file_key.read())
            self.decrypt_obj = PKCS1_OAEP.new(private_key_obj_rsa)

    #本地生成RSA密钥对
    def _generate_rsa_key(self):
        private_key_obj_rsa = RSA.generate(2048)
        public_key_obj_rsa = private_key_obj_rsa.publickey()
        self.encrypt_obj = PKCS1_OAEP.new(public_key_obj_rsa)
        self.decrypt_obj = PKCS1_OAEP.new(private_key_obj_rsa)
        #将生成的密钥对存入本地
        with open(self.public_key_file_name,"w") as file_export:
            file_export.write(public_key_obj_rsa.exportKey())
        with open(self.private_key_file_name,"w") as file_export:
            file_export.write(private_key_obj_rsa.exportKey())

    #加密数据
    def encrypt(self,plaintext_key):
        return self.encrypt_obj.encrypt(plaintext_key)

    #解密数据
    def decrypt(self,ciphertext_key):
        return self.decrypt_obj.decrypt(ciphertext_key)
#产生随机文件名
def _random_string(length):
    return ''.join(random.choice(string.ascii_lowercase + string.digits) for _ in range(length))

################put super file##################
#分块上传大文件到百度对象存储，"part_size"为分块大小，必须为16字节的整数倍
def put_super_file(bos_client, bucket_name, super_object_key, super_file, part_size,cipher_rsa):
    """
    Put super file to baidu object storage by multipart uploading."part_size" must be multiple of 16 bytes

    :param bos_client:bos client
    :type bos_client:baidubce.services.bos.bos_client.BosClient

    :param bucket_name:None
    :type bucket_name:string

    :param super_object_key: destion object key of super file
    :type super_object_key: string

    :param super_file: super file name
    :type super_file:string

    :param part_size: size of part to upload once,"part_size" must be multiple of 16 bytes and more than 5MB
    :type: int
```

```
    :param cipher_rsa: encrypt symmetric key
    :type cipher_rsa: CipherWithRSA

    :return :**Http Response**

    """
    #1.initial
    try:
        if not isinstance(bos_client,BosClient):
            raise Exception("bos client is None!")
        if not (bucket_name and super_object_key):
            raise Exception("bucket or object is invalid!")
        if not os.path.isfile(super_file):
            raise Exception("source file is invalid!")
        if not isinstance(cipher_rsa,CipherWithRSA):
            raise Exception("cipher_rsa is invalid!")
    except Exception as e:
        print e
        exit()
    if not part_size:
        part_size = 10*1024*1024
    #temp_file加密后的文件分块
    temp_file = _random_string(20)
    cipher_aes = CipherWithAES()
    #2.分块上传
    #分块上传分三步，第一步初始化，获取upload_id
    upload_id = bos_client.initiate_multipart_upload(
        bucket_name = bucket_name,
        key = super_object_key,
        ).upload_id
    left_size = os.path.getsize(super_file)
    offset = 0
    part_number = 1
    part_list = []
    fs = open(super_file,"r")
    while left_size > 0:
        if left_size < part_size:
            part_size = left_size
        # 读取分块并加密
        part_content = fs.read(part_size)
        encrypted_part_content = cipher_aes.encrypt(part_content)
        #分块加密后重新写入temp_file
        with open(temp_file,"w") as ft:
            ft.write(encrypted_part_content)
        #分块上传第二步，上传分块
        response = bos_client.upload_part_from_file(
                bucket_name, super_object_key, upload_id, part_number, part_size, temp_file, offset)
        left_size -= part_size
        #保存part number和etag用于调用complete_multipart_upload()
        part_list.append({
            "partNumber": part_number,
            "eTag": response.metadata.etag
            })
        part_number += 1
    os.remove(temp_file)
    fs.close()
    #使用公钥加密AES密钥，并作为object的meta
    user_metadata = {
        "key": base64.b64encode(cipher_rsa.encrypt(str(cipher_aes.key))),
        "start":base64.b64encode(cipher_rsa.encrypt(str(cipher_aes.start)))
    }
    #分块上传第三步，完成分块上传
    return bos_client.complete_multipart_upload(bucket_name, super_object_key, upload_id, part_list,user_metadata = user_metadata)

######################## Get super file########################
#获取大文件的分块，其中range_start, range_end分别为欲获取分块在文件中的起止位置,包含技术位置的字节
def get_part_super_file(bos_client, bucket_name, super_object_key, range_start, range_end,cipher_rsa):

    """
    Get part of  super file from baidu object storage

    :param bos_client:bos client
    :type bos_client:baidubce.services.bos.bos_client.BosClient

    :param bucket_name:None
    :type bucket_name:string

    :param super_object_key: destion object key of super file
    :type super_object_key: string

    :param range_start: index of first bytes of part,minimum  value is 0
    :type super_file:int

    :param range_end: index of last bytes of part
    :type: int

    :param cipher_rsa: dencrypt symmetric key
    :type cipher_rsa: CipherWithRSA

    :return :**Http Response**

    """

    try:
        if not isinstance(bos_client,BosClient):
            raise Exception("bos client is None!")
        if not (bucket_name and super_object_key):
            raise Exception("bucket or object is invalid!")
        if not (range_start and range_end):
            raise Exception("range is invalid!")
```

```
        if not isinstance(cipher_rsa,CipherWithRSA):
            raise Exception("cipher_rsa is invalid!")
    except Exception as e:
        print e
        exit()
    #1.对齐分块起始位置到16字节的整数倍
    left_offset = range_start%16
    right_offset = 15 -range_end%16
    test_range = [range_start-left_offset,range_end+right_offset]
    #2.获取object的meta
    response = bos_client.get_object_meta_data(bucket_name, super_object_key)
    #使用本地公钥来解密AES密钥密文，得到AES密钥明文
    download_aes_key = base64.b64decode(getattr(response.metadata,"bce_meta_key"))
    download_aes_start = base64.b64decode(str(getattr(response.metadata,"bce_meta_start")))
    aes_key = cipher_rsa.decrypt(download_aes_key)
    aes_start = cipher_rsa.decrypt(download_aes_start)
    #根据欲获取分块的起始位置调整计数器初始值
    offset_start = int(aes_start)+range_start/16
    cipher_aes = CipherWithAES(aes_key,int(offset_start))
    #3. 下载分块密文数据并使用AES密钥解密
    response = bos_client.get_object(bucket_name, super_object_key,test_range)
    download_content = response.data.read()
    plaintext_content = cipher_aes.decrypt(download_content)
    #截取用户指定区段的明文数据并返回
    return plaintext_content[left_offset:range_end-range_start+left_offset+1]

############  put common file ####################
#加密上传普通文件
def put_common_file(bos_client, bucket_name, object_key, file_name, cipher_rsa):
    """
    Put file to baidu object storage

    :param bos_client:bos client
    :type bos_client:baidubce.services.bos.bos_client.BosClient

    :param bucket_name:None
    :type bucket_name:string

    :param object_key: destion object key of file
    :type object_key: string

    :param file_name: source file name
    :type file_name:string

    :param cipher_rsa: encrypt symmetric key
    :type cipher_rsa: CipherWithRSA

    :return :**Http Response**

    """

    try:
        if not isinstance(bos_client,BosClient):
            raise Exception("bos client is None!")
        if not (bucket_name and object_key):
            raise Exception("bucket or object is invalid!")
        if not os.path.isfile(file_name):
            raise Exception("file name  is invalid!")
        if not isinstance(cipher_rsa,CipherWithRSA):
            raise Exception("cipher_rsa is invalid!")
    except Exception as e:
        print e
        exit()
    temp_file = _random_string(20)
    #读取欲上传文件数据
    content=""
    with open(file_name,"r") as fp:
        content = fp.read()
    cipher_aes = CipherWithAES()
    #加密数据并写到temp_file
    encrypt_content = cipher_aes.encrypt(content)
    with  open(temp_file,"w") as ft:
        ft.write(encrypt_content)
    cipher_rsa = CipherWithRSA()
    #加密AES密钥作为Object的meta
    user_metadata = {
            "key": base64.b64encode(cipher_rsa.encrypt(str(cipher_aes.key))),
            "start":base64.b64encode(cipher_rsa.encrypt(str(cipher_aes.start)))
            }
    #上传加密后文件
    response = bos_client.put_object_from_file(bucket = bucket_name,
            key = object_key,
            file_name = temp_file,
            user_metadata = user_metadata)
    os.remove(temp_file)
    return response

################get common file#####################
#下载加密的普通文件
def get_common_file(bos_client, bucket_name, object_key, des_file, cipher_rsa):

    """
    Put file to baidu object storage

    :param bos_client:bos client
    :type bos_client:baidubce.services.bos.bos_client.BosClient

    :param bucket_name:None
    :type bucket_name:string
```

```
    :param object_key: destion object key of file
    :type object_key: string

    :param des_file: destination file name
    :type des_file:string

    :param cipher_rsa: dencrypt symmetric key
    :type cipher_rsa: CipherWithRSA

    :return :**Http Response**

    """
    try:
        if not isinstance(bos_client,BosClient):
            raise Exception("bos client is None!")
        if not (bucket_name and object_key):
            raise Exception("bucket or object is invalid!")
        if not des_file:
            raise Exception("destination file is invalid!")
        if not isinstance(cipher_rsa,CipherWithRSA):
            raise Exception("cipher_rsa is invalid!")
    except Exception as e:
        print e
        exit()
    #下载获取meta
    response = bos_client.get_object_meta_data(bucket_name, object_key)
    download_aes_key = base64.b64decode(getattr(response.metadata,"bce_meta_key"))
    download_aes_start = base64.b64decode(str(getattr(response.metadata,"bce_meta_start")))
    #下载加密数据到本地
    download_content = bos_client.get_object_as_string(bucket_name, object_key)
    #解密得到AES密钥明文
    aes_key = cipher_rsa.decrypt(download_aes_key)
    aes_start = cipher_rsa.decrypt(download_aes_start)

    cipher_aes = CipherWithAES(aes_key,int(aes_start))
    plaintext_content = cipher_aes.decrypt(download_content)
    with open(des_file,"w") as fd:
        fd.write(plaintext_content)

if __name__ == "__main__":
 #以北京地区对象存储为例，替换AK、SK、bucket_name为用户的数据
    HOST = 'bj.bcebos.com'
    AK = 'Your_Access_Key'
    SK = 'Your_Secret_Access_Key'
  bucket_name = "Your-Bucket-Name"
    super_file= "super_file"
    super_object_key = "my-super-object"
    #获取bos client
    config = BceClientConfiguration(credentials=BceCredentials(AK, SK), endpoint=HOST)
    bos_client = BosClient(config)

    #1.1上传大文件
    #设置分块大小
    part_size = 10*1024*1024
    cipher_rsa = CipherWithRSA()
    #分块上传大文件
    put_super_file(bos_client, bucket_name, super_object_key, super_file, part_size,cipher_rsa)
    #1.2 获取大文件分块
    #设置欲获取的分块起始位置，建议设置成16字节的整数倍
    range_start = 16*10
    range_end = 16*11-1
    #获取明文数据块
    result = get_part_super_file(bos_client, bucket_name, super_object_key, range_start, range_end,cipher_rsa)
    print "#"*20
    print result
    print "#"*20
    print "length:",len(result)

    #2.1上传普通文件
    object_key = "myobject"
    source_file = "myobject.txt"
    put_common_file(bos_client,bucket_name,object_key,source_file,cipher_rsa)
    #2.2 下载普通文件
    des_file = "des_myobject.txt"
    get_common_file(bos_client,bucket_name,object_key,des_file,cipher_rsa)
```

相关产品

BOS服务端加密、密钥管理服务KMS

防盗链

简介

A网站将自己的静态资源如图片或视频等存放在百度智能云存储的BOS上。B网站在未经A允许的情况下，使用A网站的图片或视频资源，放置到自己的网站中。由于BOS是按照使用量收费，这样网站B盗取了网站A的空间和流量，而A没有获取任何利益却承担了资源使用费。B盗用A资源放到自己网站的行为即为盗链。

原理

防盗链的主要方法是通过限制引用页来实现的，有设置Referer防盗链和设置签名URL防盗链两种方式。

- 设置Referer防盗链的原理将HTTP请求表头的Referer属性（保存了该请求是从哪个URL发送到服务器的）和服务端许可的白名单进行比对，如果一致则表明是站内请求，或者为自己信任的站点请求，否则视为盗链。
- 但是Referer属性可以被恶意的篡改，针对这种情况，可以通过签名URL来实现防盗链。

配置BOS防盗链功能

设置Referer防盗链

什么是Referer

Referer是HTTP请求表头的一个参数，不同场景下HTTP请求表头的Referer属性会有差异：

- 直接访问服务器端资源：即浏览器直接请求BOS的资源，则HTTP请求中不包含Referer属性
- 在网站A中引用BOS的资源：浏览器访问的是网站test.com的index页面，该页面上引用了BOS的资源。浏览器向BOS请求资源时，HTTP的请求头中包含了Referer属性，该属性指明是 http://test.com/index 页面引用了该资源。

不同Referer设置的测试结果

为了测试防盗链的功能需要搭建两个网站，域名分别为源网站test.com和盗链网站test-steal.com。在这两个网站下分别部署一个index页面，该页面引用了BOS的图片资源。code如下：

views/index.jade (test-project) — Brackets

```
1  H1 test
2  img(src="http://bos-test-f.bj.bcebos.com/bos.jpg")
```

配置Bucket的Referer操作及原理请参考设置Referer白名单。下面详细介绍不同设置对应的不同测试结果：

- 设置Bucket的访问控制为"不允许refer为空"的同时也设置Referer白名单，能实现防盗链功能。



测试结果如下：

浏览器输入	说明	结果
http://bos-test-f.bj.bcebos.com/bos.jpg	直接访问，Referer为空	不允许空Referer的请求，返回403
http://test.com/index	请求来自于源站，Referer不为空	访问成功
http://test-steal.com/index	请求来自于盗链网站	BOS返回403，防盗链成功。

- 设置"允许refer为空"的同时也设置Referer白名单，能实现防盗链功能。



测试结果如下：

浏览器输入	说明	结果
http://bos-test-f.bj.bcebos.com/bos.jpg	直接访问，Referer为空	访问成功
http://test.com/index	请求来自于源站，Referer不为空	访问成功
http://test-steal.com/index	请求来自于盗链网站	BOS返回403，防盗链成功。

- 只设置Bucket的的访问控制为"不允许referer为空"，不能实现防盗链功能，所以不推荐该配置。



测试结果如下：

浏览器输入	说明	结果
http://bos-test-f.bj.bcebos.com/bos.jpg	直接访问，Referer为空。	不允许空Referer的请求，返回403
http://test.com/index	请求来自于源站，Referer不为空	访问成功
http://test-steal.com/index	请求来自于盗链网站,但是Referer不为空	访问成功。

Referer防盗链的优缺点

Referer防盗链的优点是设置简单，控制台即可操作。缺点是无法防止恶意伪造Referer，如果盗链是通过应用程序模拟HTTP请求伪造Referer，则会绕过用户防盗链设置。如果对防盗链有更高要求的则需要通过签名URL实现防盗链。

配置签名URL防盗链

签名URL防盗链的原理即将文件设为私有访问，然后生成一个预签名的URL，提供给用户一个临时的访问UEL。生成预签名URL时可以通过指定URL的有效时长限制用户长时间访问。如调用Java SDK实现预签名的URL请参考获取Object的URL。

签名URL防盗链使用nodejs的sails框架来调用BOS的javascript SDK来实现该功能，可以在code中指定URL的过期时间`expirationInSeconds`，具体实现code如下：

```
 var BosClient = require("@baiducloud/sdk").BosClient;

module.exports = {

   showImage:function(req,res){
      var config = {
        endpoint: "http://bj.bcebos.com",
        credentials: {
            ak: "AK",     //您的AK
            sk: "SK"      //您的SK
        }
      };

      var client = new BosClient(config);

      var bucketName = "bos-test-f";
      var key = "bos.jpg";
      var timestamp = Date.now() / 1000;
      var expirationInSeconds = 1800;
      var headers = {};
      var params = {};
      var headersToSign = {};

      var signedUrl = client.generatePresignedUrl(bucketName, key, timestamp, expirationInSeconds, headers, params, headersToSign, config);

      return res.view("image", {signedUrl : signedUrl});
   }
};
```

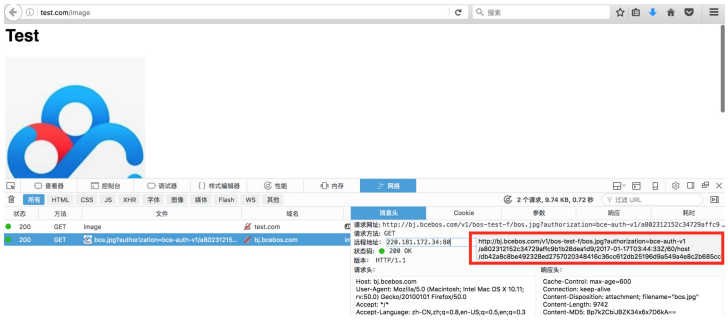
指定image资源的地址为签名URL：

```
 Hi Test
img(src="#{signedUrl}")
```

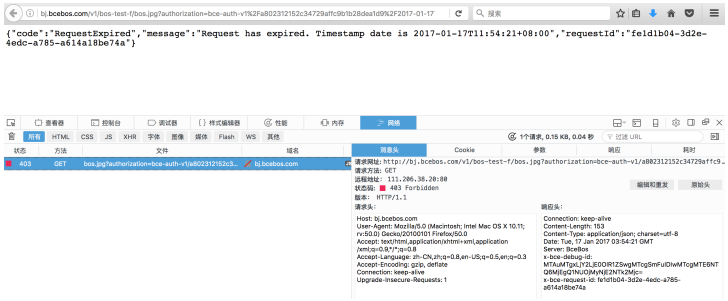
测试结果

如图所示，每次访问该页面时都会重新生成一个签名的URL来访问BOS的资源，该签名的URL是有过期时间的。如果过期则返回403错误拒绝访问，从而实现防盗链功能。

- 未过期签名URL结果



- 过期拒绝访问结果



Web数据直传至BOS

基础篇：在浏览器中直接上传文件到BOS

使用流程

使用 bce-bos-uploader，可以参考下面的内容完成如何在浏览器中直接上传文件到 BOS 。使用流程：

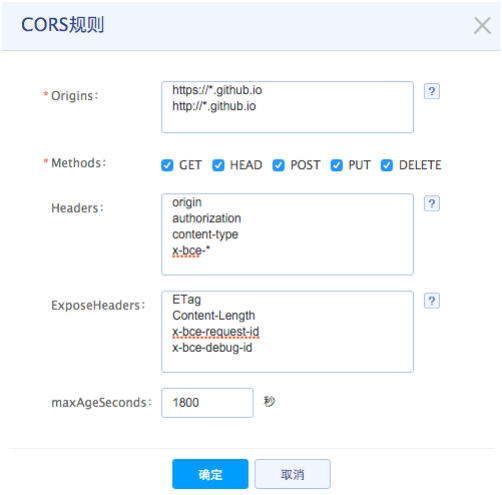
> 特别说明：bce-bos-uploader 已不再更新维护，直传BOS推荐使用JS SDK

1. 开启 Bucket 的跨域访问设置
2. 查询 ak/sk
3. 初始化 bce-bos-uploader 参数

开启 Bucket 的跨域访问

受浏览器安全限制，如果想直接在浏览器中访问 BOS 服务，必须正确设置好相关 bucket 的跨域功能。设置方法如下：

1. 登录百度智能云控制台。
2. 选择 Bucket 并进入 Bucket 管理页面。
3. 点击左侧『Bucket 属性』，进入 Bucket 配置的页面。
4. 点击右侧『CORS 设置』，进入 CORS 设置页面。
5. 点击『添加规则』按钮，可以添加一条或者多条 CORS 的规则。



查询 AK/SK

在百度智能云控制台首页右上角账号下的"安全认证"查询AK和SK的信息，也可以在 Bucket 管理中查看。详细操作可参见管理ACCESSKEY。

获取 bce-bos-uploader

有两种方式可以获取 bce-bos-uploader 的代码：

- 第一种：通过 npm 安装

```
npm install @baiducloud/bos-uploader
```

- 第二种：直接引用 CDN 上面的资源（测试专用，不建议用于生产环境）

```
<script src=" https://bj.bcebos.com/v1/bce-cdn/lib/@baiducloud/bos-uploader/<version>/bce-bos-uploader.bundle.min.js"></script>
```

初始化 bce-bos-uploader

```
<!doctype html>
<html>
 <head>
  <meta charset="utf-8" />
  <title>bce-bos-uploader simple demo</title>
  <!--[if lt IE 8]><script src="https://unpkg.com/json3@3.3.2/lib/json3.min.js"></script><![endif]-->
  <!--[if lt IE 9]><script src="https://unpkg.com/js-polyfills@0.1.42/es5.js"></script><![endif]-->
  <!--[if lt IE 10]><script src="https://unpkg.com/mOxie@1.5.7/bin/js/moxie.min.js"></script><![endif]-->
  <script src="https://unpkg.com/jquery@3.3.1/dist/jquery.min.js"></script>
  <script src="http://bce-cdn.bj.bcebos.com/lib/@baiducloud/bos-uploader/1.4.0-rc.0/bce-bos-uploader.bundle.min.js"></script>
 </head>
 <body>
  <input type="file" id="file" >
  <button type="submit">开始上传</button>
  <script>
   var uploader = new baidubce.bos.Uploader({
    browse_button: '#file',
    bos_bucket: '<your bucket>',
    bos_endpoint: '<your host>',
    bos_ak: '<your ak>',
    bos_sk: '<your sk>',
    max_file_size: '1Gb',
    init: {
     FileUploaded: function (_, file, info) {
      var bucket = info.body.bucket;
      var object = info.body.object;
      var url = '<your host>' + bucket + '/' + object;
      $(document.body).append($('<div><a href="' + url + '">' + url + '</a></div>'));
     },
     UploadComplete: function() {
      $(document.body).append('<div>上传结束！</div>');
     }
    }
   });
   $('button[type=submit]').click(function () {
    uploader.start();
    return false;
   });

  </script>
 </body>
</html>
```

将上述代码保存为 index.html，下面会启动 webserver 来访问这个页面

启动 webserver

- 通过 PHP 来启动

```
php -S 0.0.0.0:9999
```

- 通过 Python 来启动

```
python -m SimpleHTTPServer 9999
```

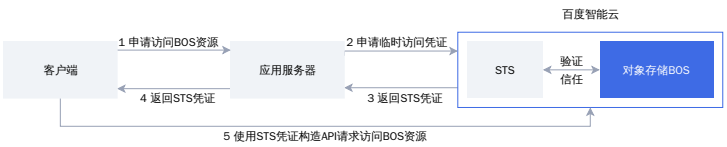
- 其它方式请参考相关的文档

启动 webserver 之后，在浏览器里面访问http://localhost:9999/index.html打开刚才的页面，开始测试是否可以正常上传。

进阶篇一：STS临时认证

进阶篇一：STS 临时认证

Bce-bos-uploader 支持 STS（Security Token Service）临时授权的方式。服务端生成一组具体特定操作权限、具有一定时效性的临时AK/SK，这组临时的 AK/SK 可以暴露给浏览器端直接使用。用户只需要将服务端返回的 AK/SK 及 SessionToken 设置为 bce-bos-uploader 对应的 bos-ak、bos-sk 和 uptoken 参数。 下图简单介绍了整个业务交互过程，关于 STS 方面的介绍请参考临时授权访问。



代码实现分为应用服务器端和客户端两部分，实现过程如下：

1. 配置应用服务器端，以 Nodejs 实现为例，服务器端会返回 AK/SK/SessionToken 。

2. 配置浏览器，根据服务器端返回的 AK/SK/SessionToken 初始化 bce-bos-uploader 参数。

应用服务器端 Nodejs 实现

```
 var http = require('http');
var url = require('url');
var util = require('util');

var STS = require('@baiducloud/sdk').STS;

var kCredentials = {
    ak: '您的AK',
    sk: '您的SK'
};

function buildStsResponse() {
    var stsClient = new STS({
        credentials: kCredentials,
        region: 'bj'
    });
    return stsClient.getSessionToken(60 * 60 * 24, {
        accessControlList: [{
            service: 'bce:bos',
            resource: ['bce-javascript-sdk-demo-test'],
            region: '*',
            effect: 'Allow',
            permission: ['READ', 'WRITE']
        }]
    }).then(function (response) {
        var body = response.body;
        return {
            AccessKeyId: body.accessKeyId,
            SecretAccessKey: body.secretAccessKey,
            SessionToken: body.sessionToken,
            Expiration: body.expiration
        };
    });
}

http.createServer(function (req, res) {
    console.log(req.url);

    var query = url.parse(req.url, true).query;

    var promise = null;

    if (query.sts) {
        promise = buildStsResponse();
    }

    promise.then(function (payload) {
        res.writeHead(200, {
            'Content-Type': 'text/javascript; charset=utf-8',
            'Access-Control-Allow-Origin': '*'
        });

        if (query.callback) {
            res.end(util.format('%s(%s)', query.callback, JSON.stringify(payload)));
        }
        else {
            res.end(JSON.stringify(payload));
        }
    });
}).listen(1337);
console.log('Server running at http://0.0.0.0:1337/');
```

在服务器端，用与创建 bosClient 实例类似的方式创建一个 stsClient 实例。对于 stsClient 实例，主要有一个方法，那就是 getSessionToken 。这个方法接收两个参数，第一个参数是临时授权的有效期，以秒为单位；第二个单位是具体的权限控制，参见STS服务接口。

这个方法会异步访问 STS 授权服务器，返回一个 promise 对象。 STS 授权服务器会返回类似如下内容：

```
 {
    body: {
        "accessKeyId": "d87a16e5ce1d47c1917b38ed03fbb329",
        "secretAccessKey": "e9b6f59ce06c45cdaaea2296111dab46",
         "sessionToken":
"MjUzZjQzNTY4OTE0NDRkNjg3N2E4YzJhZTc4YmU5ZDh8AAAAABwCAAB/HfHDVV2bu5xUf6rApt2YdSLG6+21UTC62EHvluiaamtuMQQKNkR9PU2NJGVbuWgBn8Ot0atk0HnWYQGgwgyew24HtbrX3GFiR/cDymCowm0TI6
",
        "createTime": "2016-02-16T14:01:29Z",
        "expiration": "2016-02-16T15:41:29Z",
        "userId": "5e433c4a8fe74765a7ec6fc147e25c80"
    }
}
```

服务器端需要把 accessKeyId、secretAccessKey、sessionToken 三个字段下发给浏览器端。

配置浏览器端 bce-bos-uploader 参数

使用 STS 临时授权机制时，只需要在各个服务初始化的时候把上面所说的参数 accessKeyId、secretAccessKey、sessionToken 引入就可以了。

```
<!doctype html>
<html>
 <head>
  <meta charset="utf-8" />
  <title>bce-bos-uploader simple demo</title>
  <!--[if lt IE 8]><script src="http://websdk.cdn.bcebos.com/bos/json3/lib/json3.min.js"></script><![endif]-->
  <!--[if lt IE 9]><script src="http://websdk.cdn.bcebos.com/bos/js-polyfills/es5.js"></script><![endif]-->
  <!--[if lt IE 10]><script src="http://websdk.cdn.bcebos.com/bos/moxie/bin/js/moxie.js"></script><![endif]-->
  <!-- BOS不提供jquery.min.js，开发者可以自行引入在线资源或本地资源 -->
  <script src="./node_modules/jquery/dist/jquery.min.js"></script>
  <!-- 引入bce-bos-uploader.bundle.js，建议通过npm安装成功后，引入本地资源 -->
  <script src="./node_modules/@baiducloud/bos-uploader/dist/bce-bos-uploader.bundle.js"></script>
 </head>
 <body>

  <input type="file" id="file" >
  <script>
  var uploader = new baidubce.bos.Uploader({
    browse_button: '#file',
    bos_bucket: '<your bucket>',
    bos_endpoint: 'http://bj.bcebos.com',
    bos_ak: '<your ak>',
    bos_sk: '<your sk>',
    uptoken: '<your sessionToken>'
  });
  </script>
 </body>
</html>
```

进阶篇二：通过PostObject接口处理IE低版本

进阶篇二：通过 PostObject 接口处理 IE 低版本

因为 IE 低版本（IE8,IE9）对 html5 支持的不完善，为了在这些浏览器里面实现文件直传的功能， BOS 开发了 PostObject 接口，通过一个 multipart/form-data 的格式，就可以把文件上传到 BOS 服务器。Postobject 接口的签名模式下应用服务器端对 policy 生成签名，再返回给客户端。



bce-bos-uploader 已经实现了对这个接口的支持，使用之前需要进行额外的配置工作：

配置应用服务器端

1. 上传 crossdomain.xml

基于 html5 的跨域方案，我们需要设置跨域范文（CORS）；如果通过 flash 来完成跨域数据交互的话，需要设置 crossdomain.xml，可以直接把如下内容保存为 crossdomain.xml，然后上传到 Bucket 的根目录。

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE cross-domain-policy SYSTEM "http://www.macromedia.com/xml/dtds/cross-domain-policy.dtd">
<cross-domain-policy>
 <site-control permitted-cross-domain-policies="all"/>
 <allow-access-from domain="*" secure="false" />
 <allow-http-request-headers-from domain="*" headers="*" secure="false"/>
</cross-domain-policy>
```

如果 bucket 是 private 的，需要设置 bucket 为自定义权限来允许 crossdomain.xml 为 public-read 。在 Console 控制台选择 bucket 并进入“基础设置”页签，选择“ Bucket 权限设置”对应的“修改配置”，修改 bucket 的权限设置为“自定义权限”并添加权限。



2. 服务器端返回 uptoken_url 参数。

使用 ostObject 处理 IE 低版本时，需要在 bce-bos-uploader 中配置 uptoken_url 参数。

配置浏览器端 bce-bos-uploader 参数

- 以 html 方式初始化 bce-bos-uploader 中的 uptoken_url 参数。

  Choose File No file selected

- 以js方式初始化 bce-bos-uploader 中的 uptoken_url 参数。

  Choose File No file selected

原理篇一：在浏览器中直接上传文件到BOS

原理篇一：在浏览器中直接上传文件到BOS

如您不使用 bce-bos-uploader ，可以参考下面的内容完成如何在浏览器中直接上传文件到 BOS 。使用流程：

1. 开启 Bucket 的跨域访问设置
2. 查询 ak/sk
3. 初始化 BosClient
4. 处理上传逻辑

开启 Bucket 的跨域访问

受浏览器安全限制，如果想直接在浏览器中访问 BOS 服务，必须正确设置好相关 bucket 的跨域功能。设置方法请参考开启 Bucket 的跨域访问。

初始化设置

```
var bosConfig = {
  credentials: {
      ak: '从百度智能云控制台查询您的ak',
      sk: '从百度智能云控制台查询上面这个ak所对应的sk'
  },
  endpoint: 'http://bj.bcebos.com' // 根据您选用bos服务的区域配置相应的endpoint
};
var bucket = 'bce-javascript-sdk-demo-test'; // 设置您想要操作的bucket
var client = new baidubce.sdk.BosClient(bosConfig);
```

后续我们可以使用 client 这个实例来进行 BOS 相关操作。

上传逻辑

我们可以通过调用 client.putObjectFromBlob(bucket, key, blob, options) 来完成文件的上传操作。

这个函数支持4个参数，其中 options 是可选的。如果需要手工设置文件的的 Content-Type，可以放到 options 参数里面。 如果没有手工设置，默认的 Content-Type 是 application/oceat-stream 。 另外，可以通过调用 baidubce.sdk.MimeType.guess(ext) 来根据后缀名得到一些常用的 Content-Type 。

> **注意：**因为 Firefox 兼容性的一个问题，如果上传的文件是 text/* 类型，Firefox 会自动添加 charset=utf-8 因此我们给 options 设置 Content-Type 的时候，需要手工加上 charset=utf-8，否则会导致浏览器计算的签名跟服务器计算的签名不一致，导致上传失败。

```
// 监听文件上传的事件，假设页面中有：<input type="file" id="upload" /> $('#upload').on('change', function (evt) {
  var file = evt.target.files[0]; // 获取要上传的文件
  var key = file.name; // 保存到bos时的key，您可更改，默认以文件名作为key
  var blob = file;

  var ext = key.split(/\./g).pop();
  var mimeType = baidubce.sdk.MimeType.guess(ext);
  if (/^text\//.test(mimeType)) {
      mimeType += '; charset=UTF-8';
  }
  var options = {
      'Content-Type': mimeType
  };

   client.putObjectFromBlob(bucket, key, blob, options)
      .then(function (res) {
          // 上传完成，添加您的代码
          console.log('上传成功');
      })
      .catch(function (err) {
          // 上传失败，添加您的代码
          console.error(err);
      });
});
```

如果想获悉当前上传的进度，可以监听 progress 事件。

```
 client.on('progress', function (evt)
{
   // 监听上传进度
   if (evt.lengthComputable)
{
      // 添加您的代码
      var percentage = (evt.loaded / evt.total) * 100;
      console.log('上传中，已上传了' + percentage + '%');
   }
});
```

原理篇二：大文件分块上传

原理篇二：大文件分块上传

用户在使用浏览器上传文件到 BOS 的时候，如果遇到文件过大，需要先将文件分块然后再上传。上传过程中有可能会遇到页面关闭、浏览器崩溃、网络连接中断等问题，从而导致上传失败。BOS 支持分块上传和断点续传功能。分块上传请参见“ Object 的分块上传”，下面介绍“断点续传”的实现方法。

实现原理

在我们使用文件分块上传（multipartUpload）的时候， BOS 首先会为这个上传过程分配一个 uploadId 。然后我们将一个文件被分成了若干 part ，每个 part 独立上传，上传完成后，BOS 服务会为这个 part 生成一个 eTag 。当所有 part 都上传完成的时候，BOS 服务根据这些 eTag 和 uploadId 把正确的 part 找出来，并组合成原本的文件。

在这个过程中，BOS 并不需要所有的 part 一下子全部上传完毕，而是可以分多次进行。这也就是说，上传过程中，当页面意外关闭时，我们可以不必从头开始重新上传，而只需要把未上传成功的 part 的再次上传就可以。当然，前提是我们需要把此次上传的 uploadId 和上传完成的 part 的 etag 保存下来（不过，更推荐的做法是通过 listParts 接口来查询更精确的已上传分块信息）。在上传一个 part 之前，可以先检查一下，这个 part 是否已经上传过了，如果以前已上传成功，那就直接跳过这个 part 的上传过程。

对于 uploadId 的存储，需要满足不受页面关闭的影响，比较理想的做法是存储在 localStorage 中。

本地存储

在保存 uploadId 时，我们需要为它指定一个 key ，让不同的文件、不同的上传过程区分开。本示例采用文件名、文件大小、分区大小、 bucket 名称、 object 名称组合成这个 key ：

```
var generateLocalKey = function (blob, chunkSize, bucket, object) {
    return [blob.name, blob.size, chunkSize, bucket, object].join('&');
};
```

注意：用这个方式生成的 key 并不准确，如果两次上传过程中，选择了两个文件名相同、文件大小相同，但内容不同的文件，那么用这样的方式并不能正确区分这两个文件。更严谨的方式是根据文件名和文件内容计算 MD5 ，并以此为 key 。

存储方式我们选择 localStorage ：

```
var getUploadId = function (key) {
    return localStorage.getItem(key);
};
var setUploadId = function (key, uploadId) {
    return localStorage.setItem(key, uploadId);
};
var removeUploadId = function (key) {
    return localStorage.removeItem(key);
};
```

初始化分块上传

在初始化分块上传时，有两种可能：

- 如果已经存在此文件的 uploadId ，那么跳过 initiateMultipartUpload() 方法，改为调用 listParts() 来获取已上传分块信息；
- 如果没有此文件的 uploadId，那么调用 initiateMultipartUpload() 方法获得新的 uploadId ，并将这个 uploadId 保存在 localStorage 中。

```
// ...省略BosClient初始化过程
// var bosClient = new BosClient(bosConfig);

var initiateMultipartUpload = function (file, chunkSize, bucket, object) {
    // 根据文件生成localStorage的key
    var key = generateLocalKey(file, chunkSize, bucket, object);
    // 获取对应的`uploadId`
    var uploadId = getUploadId(key);
    if (uploadId) {
        // `uploadId`存在，说明有未完成的分块上传。
        // 那么调用`listParts()`获取已上传分块信息。
        return BosClient.listParts(bucket, object, uploadId)
            .then(function (response) {
                // response.body.parts里包含了已上传分块的信息
                response.body.uploadId = uploadId;
                return response;
            });
    }
    else {
        // `uploadId`不存在，那么用正常的流程初始化
        return BosClient.initiateMultipartUpload(bucket, object)
            .then(function (response) {
                // response.body.uploadId为新生成的`uploadId`
                response.body.parts = [];
                // 为了下次能使用断点续传，我们需要把新生成的`uploadId`保存下来
                setUploadId(key, response.body.uploadId);
                return response;
            });
    }
}
```

分块上传

在对大文件分割分块时，我们可以跟以上传的分块列表进行比较，以确定是否需要真的进行上传。

```
function getEtag(partNumber, parts){
    // 从已上传part列表中找出特定partNumber的part的eTag
    for(var i = 0, l = parts.length; i < l; i++){
        if (parts[i].partNumber === partNumber) {
            return parts[i].eTag;
        }
    }
    return null;
}
function getTasks (file, uploadId, chunkSize, bucket, object, parts) {
    var leftSize = file.size;
    var offset = 0;
    var partNumber = 1;

    var tasks = [];

    while (leftSize > 0) {
        var partSize = Math.min(leftSize, chunkSize);
        var task = {
            file: file,
            uploadId: uploadId,
            bucket: bucket,
            object: object,
            partNumber: partNumber,
            partSize: partSize,
            start: offset,
            stop: offset + partSize - 1
        };

        // 如果在已上传完成的分块列表中找到这个分块的etag，那么记录下来
        var etag = getEtag(partNumber, parts);
        if (etag){
            task.etag = etag;
        }
        tasks.push(task);
        leftSize -= partSize;
        offset += partSize;
        partNumber += 1;
    }
    return tasks;
}
```

在进行分块上传处理的时候，根据是否已带有etag字段来决定是否需要上传：

```
function uploadPartFile(state, bosClient) {
    return function(task, callback) {
        if (task.etag) {
            // 如果有etag字段，则直接跳过上传
            callback(null, {
                http_headers: {
                    etag: task.etag
                },
                body: {}
            });
        }
        else {
            // 否则进行上传
            var blob = task.file.slice(task.start, task.stop + 1);
            bosClient.uploadPartFromBlob(task.bucketName, task.key, task.uploadId, task.partNumber, task.partSize, blob)
                .then(function(res) {
                    ++state.loaded;
                  callbacknull(res);
                })
                .catch(function(err) {
                    callback(err);
                });
        }
    };
}
```

流程代码

我们对每个步骤的代码做了一些小修改，但整个流程的代码与分块上传很类似：

```
  var chunkSize = 5 * 1024 * 1024; // 分块大小
var uploadId;
initiateMultipartUpload(file, chunkSize, bucket, object)
    .then(function(response) {
        uploadId = response.body.uploadId; // uploadId，可能是服务器刚刚生成的，也可能是从localStorage获取的
        var parts = response.body.parts || []; // 已上传的分块列表。如果是新上传，则为空数组
        var deferred = sdk.Q.defer();
        var tasks = getTasks(blob, uploadId, chunkSize, bucket, key, parts);
        var state = {
            lengthComputable: true,
            loaded: parts.length, // 已上传的分块数
            total: tasks.length
        };
        // 如果已上传的分块数大于0，可以先修改一下文件上传进度
        bosClient.emit('progress', state);
        // 为了管理分块上传，使用了async（https://github.com/caolan/async）库来进行异步处理
        var THREADS = 2; // 同时上传的分块数量
        async.mapLimit(tasks, THREADS, uploadPartFile(state, bosClient), function(err, results) {
            if (err) {
                deferred.reject(err);
            } else {
                deferred.resolve(results);
            }
        });
        return deferred.promise;
    })
    .then(function(allResponse) {
        var partList = [];
        allResponse.forEach(function(response, index) {
            // 生成分块清单
            partList.push({
                partNumber: index + 1,
                eTag: response.http_headers.etag
            });
        });

        // 所有分块上传完成后，可以删除对应的`uploadId`了
        removeUploadId(key, uploadId);

        return bosClient.completeMultipartUpload(bucket, key, uploadId, partList); // 完成上传
    })
    .then(function (res) {
        // 上传完成
    })
    .catch(function (err) {
        // 上传失败，添加您的代码
        console.error(err);
    });
```

原理篇三：STS临时认证

原理篇三：STS 临时认证

Bce-bos-uploader 支持 STS（Security Token Service） 临时授权的方式。服务端生成一组具体特定操作权限、具有一定时效性的临时 AK/SK ，这组临时的 AK/SK 可以暴露给浏览器端直接使用。用户只需要将服务端返回的 AK/SK 及 SessionToken 设置为 bce-bos-uploader 对应的 bos-ak、bos-sk 和 uptoken 参数。

关于 STS 方面的介绍请参考临时授权访问。使用流程：

1. 配置应用服务器端 Nodejs 实现
2. 获取临时 AK/SK/SessionToken
3. 初始化 bce-bos-uploader 参数

应用服务器端 Nodejs 实现

```
 var http = require('http');
var url = require('url');
var util = require('util');

var STS = require('@baiducloud/sdk').STS;

var kCredentials = {
    ak: '您的AK',
    sk: '您的SK'
};

function buildStsResponse() {
    var stsClient = new STS({
        credentials: kCredentials,
        region: 'bj'
    });
    return stsClient.getSessionToken(60 * 60 * 24, {
        accessControlList: [{
            service: 'bce:bos',
            resource: ['bce-javascript-sdk-demo-test'],
            region: '*',
            effect: 'Allow',
            permission: ['READ', 'WRITE']
        }]
    }).then(function (response) {
        var body = response.body;
        return {
            AccessKeyId: body.accessKeyId,
            SecretAccessKey: body.secretAccessKey,
            SessionToken: body.sessionToken,
            Expiration: body.expiration
        };
    });
}

http.createServer(function (req, res) {
    console.log(req.url);

    var query = url.parse(req.url, true).query;

    var promise = null;

    if (query.sts) {
        promise = buildStsResponse();
    }

    promise.then(function (payload) {
        res.writeHead(200, {
            'Content-Type': 'text/javascript; charset=utf-8',
            'Access-Control-Allow-Origin': '*'
        });

        if (query.callback) {
            res.end(util.format('%s(%s)', query.callback, JSON.stringify(payload)));
        }
        else {
            res.end(JSON.stringify(payload));
        }
    });
}).listen(1337);
console.log('Server running at http://0.0.0.0:1337/');
```

在服务器端，用与创建 bosClient 实例类似的方式创建一个 stsClient 实例。对于 stsClient 实例，主要有一个方法，那就是 getSessionToken 。这个方法接收两个参数，第一个参数是临时授权的有效期，以秒为单位；第二个单位是具体的权限控制，参见STS服务接口。

这个方法会异步访问 STS 授权服务器，返回一个 promise 对象。 STS 授权服务器会返回类似如下内容：

```
 {
    body: {
        "accessKeyId": "d87a16e5ce1d47c1917b38ed03fbb329",
        "secretAccessKey": "e9b6f59ce06c45cdaaea2296111dab46",
        "sessionToken":
"MjUzZjQzNTY4OTE0NDRkNjg3N2E4YzJhZTc4YmU5ZDh8AAAAABwCAAB/HfHDVV2bu5xUf6rApt2YdSLG6+21UTC62EHvluiaamtuMQQKNkR9PU2NJGVbuWgBn8Ot0atk0HnWYQGgwgyew24HtbrX3GFiR/cDymCowm0TI6
",
        "createTime": "2016-02-16T14:01:29Z",
        "expiration": "2016-02-16T15:41:29Z",
        "userId": "5e433c4a8fe74765a7ec6fc147e25c80"
    }
}
```

服务器端需要把 accessKeyId、secretAccessKey、sessionToken 三个字段下发给浏览器端。

浏览器前端实现

前端使用 STS 临时授权机制时，只需要在各个服务初始化的时候把上面所说的参数 accessKeyId、secretAccessKey、sessionToken 引入就可以了。以 BOS 为例：

```
 var bosConfig = {
    credentials: {
        ak: '{accessKeyId}', // STS服务器下发的临时ak
        sk: '{secretAccessKey}' // STS服务器下发的临时sk
    },
    sessionToken: '{sessionToken}',  // STS服务器下发的sessionToken
    endpoint: 'http://bj.bcebos.com'
};
var client = new baidubce.sdk.BosClient(bosConfig);
```

手机美图APP实践

概述及场景介绍

概览

本教程能帮助用户快速搭建一个基于 BOS 的文件直传+图片处理的手机 APP，主要基于 STS 临时授权、Android SDK 和图片处理 API 三个模块实现。

- 由于用户的移动端应用存在泄密的风险所以不可能直接存储 AK/SK 信息，必须使用 STS 临时授权模式访问 BOS 。STS 临时授权模式中会生成一个临时 Token，该 Token 具有一定的时效性，即 APP 应用只有在 Token 的时效性内访问才可以完成上传或下载图片服务，过了时效需要重新获取。
- Andriod SDK 帮助用户实现新建 BOS 客户端并将文件存储到 BOS 或从 BOS 中下载文件。
- 图片处理 API 主要实现图片的处理如缩略、裁剪、格式转换、旋转、加文字/图片水印等功能。

需求场景

移动互联时代手机上传数据的场景随处可见，为了方便开发者聚焦于产品的业务逻辑，用户可以直接将文件存储到 BOS 上。 BOS 产品基于 STS 授权方式为用户提供了安全的上传和下载方式，BOS 还支持图片处理服务。BOS 具有成本低、支持海量存储和弹性扩展的特性，能帮助开发者更方便实现移动 APP 业务的开发。手机美图 APP 的整体搭建逻辑如下：



方案实践——搭建美图APP

美图 APP 示例

手机美图示例 APP 下载地址：

下载完 APP 并安装完成后可以直接通过应用服务器地址访问 BOS ，并进行图片处理。应用服务器地址是指搭建移动应用的后台服务器，默认开启的端口为8080。关于 BOS 的区域和 Bucket 设置都需要在应用服务器进行配置。



APP操作方法

该 APP 支持用户上传、下载和下载缩放图三个功能。

- 上传：用户填写应用服务器地址后，选择本地需要上传的图片，图片会显示在操作界面下方，点击上传即可。上传成功会显示 “File Uploaded”。
- 下载：用户填写需要下载的文件名称，点击下载按钮即可。下载成功会显示 “File Downloaded”。
- 下载缩放图：下载缩放图时必须指定明确的图片后缀如 jpg ，然后设置下载图形的宽和高以及旋转角度，点击下载缩放图，则会获取经过处理的图形。下载成功会显示 “File Downloaded”。 Demo 版本下载后的图片都不会存储到本地。

如何搭建美图APP

搭建美图 APP 包含以下几个步骤：

1. 开启 BOS 服务并创建 Bucket 用于存储图片，开通和创建 Bucket 的详细操作请参见创建 Bucket。如果要下载缩放图，需要保证指定的 Bucket 开启了图片处理服务。
2. 开通 STS 服务，用于保证上传和下载图片的安全性。
3. 部署应用服务器，实现和 BOS 及客户端的交互，美图 APP 代码请参考：BOS美图APP代码。
4. 下载安装美图APP。

如何部署应用服务器

1. 从 github 上下载sample code的代码包，代码包主要包含 “bos_meitu_app” 和 “bos_meitu_app_server” 两部分，其中 “bos_meitu_app” 主要用于定义APP界面及相关动作，“bos_meitu_app_server” 为应用服务器相关配置。
2. 修改 “bos_meitu_app_server” 中的 “MeituAppServerHandler.java” 文件，其中定义了 ak/sk、BOS 服务器对应的 Endpoint 和 Bucket 名称等信息。

```
public String getBosInfo(String bosRequestType) {
    //配置ak、sk
    String bosAk = "开发者的ak";
    String bosSk = "开发者的SK";
    //百度智能云提供的stsendpoint
    String stsEndpoint = "http://sts.bj.baidubce.com";

    BceCredentials credentials = new DefaultBceCredentials(bosAk, bosSk);
    BceClientConfiguration clientConfig = new BceClientConfiguration();
    clientConfig.setCredentials(credentials);
    clientConfig.setEndpoint(stsEndpoint);
    StsClient stsClient = new StsClient(clientConfig);
    GetSessionTokenRequest stsReq = new GetSessionTokenRequest();
    // request expiration time
    stsReq.setDurationSeconds(1800);
    GetSessionTokenResponse stsToken = stsClient.getSessionToken(stsReq);
    String stsTokenAk = stsToken.getCredentials().getAccessKeyId();
    String stsTokenSk = stsToken.getCredentials().getSecretAccessKey();
    String stsTokenSessionToken = stsToken.getCredentials().getSessionToken();

    // BOS服务的Endpoint地址，及Bucket名称。
    String bosEndpoint = "http://bj.bcebos.com";
    String bucketName = "Bucket名称";
    if (bosRequestType.equalsIgnoreCase("download-processed")) {
        // the binded image processing domain set by App developer on bce console
        bosEndpoint = "http://" + bucketName + ".bj.bcebos.com";
    }

    // prefix is the bucket name, and does not specify the object name
    BosInfo bosInfo = new BosInfo(stsTokenAk, stsTokenSk, stsTokenSessionToken, bosEndpoint,
            bucketName, "", bucketName);

    String res = "";
    ObjectMapper mapper = new ObjectMapper();
    try {
        res = mapper.writerWithDefaultPrettyPrinter().writeValueAsString(bosInfo);
    } catch (JsonProcessingException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
        res = "";
    }
    System.out.println(res);
    try {
        res = new String(res.getBytes(), "utf8");
    } catch (UnsupportedEncodingException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
        res = "";
    }
    return res;
}
```

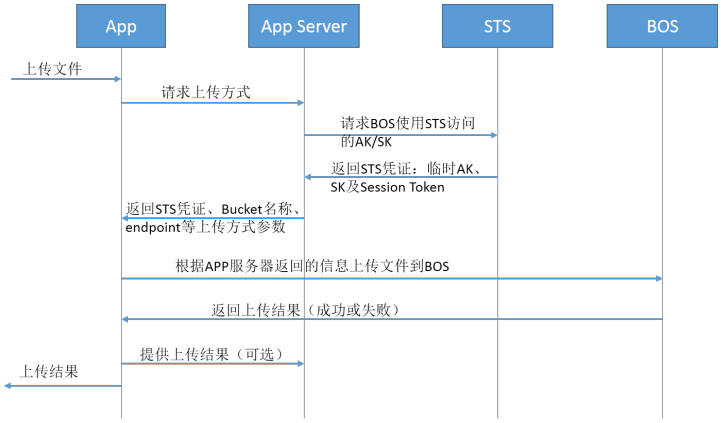
3. 将修改完成的服务器代码重新编译打包为 bos_meitu_app_server.jar，将 jar 包上传到应用服务器上并执行命令 `java -jar bos_meitu_app_server.jar`。

方案实践——使用美图APP

交互过程

上传图片到 BOS

上传图片到 BOS 过程中 APP、APP Server 和 BOS 的交互过程如下图所示：

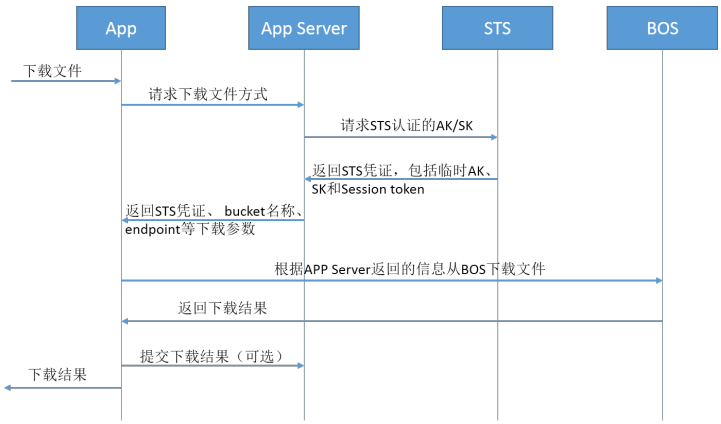


1. APP 上传图片时向 APP Server 发送获取上传方式请求。

2. APP Server 向 STS 服务器请求 BOS 使用 STS 访问的 AK/SK，STS 服务器向 APP Server 返回 STS 凭证，包括临时 AK、SK 和 Session Token。
3. APP Server 将 STS 凭证及上传方式参数返回，上传方式参数包含 Bucket 名称、Endpoint 等。
4. APP 根据返回的信息将文件上传到 BOS 上，BOS 会将上传结果返回给 APP。
5. APP 可以根据需要将上传结果提供给 APP Server。

从 BOS 下载图片

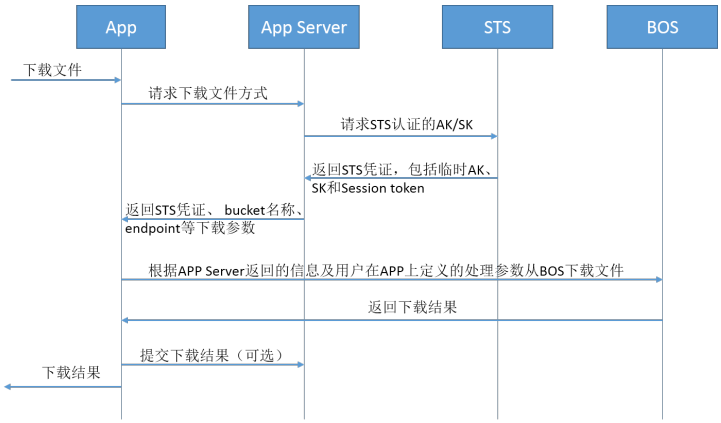
从 BOS 下载图片过程中 APP、APP Server 和 BOS 的交互过程如下图所示：



1. APP 下载图片时向 APP Server 发送获取下载方式请求。
2. APP Server 向STS服务器请求 BOS 使用 STS 访问的 AK/SK ，STS 服务器向 APP Server 返回 STS 凭证，包括临时 AK、SK 和Session Token。
3. APP Server 将 STS 凭证及下载方式参数返回 APP，下载方式参数包含 Bucket 名称、Endpoint 等。
4. APP 根据返回的信息
5. 从 BOS 上下载文件，BOS 会将下载结果返回给 APP。
6. APP 可以根据需要将下载结果提供给 APP Server。

从 BOS 下载缩放图

从 BOS 下载缩放图过程中 APP、APP Server 和 BOS 的交互过程如下图所示：



从 BOS 下载缩放图和从 BOS 上下载图片交互过程基本类似，只是在从 BOS 下载缩放图时需要携带 APP 上设定的图片处理参数，如图片宽、高和旋转角度等。

代码示例

代码示例

示例代码以 Java 语言为例讲解美图 APP 的实现，代码分为 APP 客户端和应用服务器端两部分。

APP 客户端代码样例

APP 端代码主要包括 BOSClient 初始化、从 APP Server 端获取 BOS 信息、及上传文件到 BOS 三个功能模块。

BOSClient 初始化

```
public class bos {
private string ak = null;
  private string sk = null;
  private string endpoint = null;
  private string ststoken = null;
  private bosclient client = null;

  public bos(string ak, string sk, string endpoint, string ststoken) {
    this.ak = ak;
    this.sk = sk;
    this.endpoint = endpoint;
    this.ststoken = ststoken;
    client = createclient();
  }

  public bosclient createclient() {
    bosclientconfiguration config = new bosclientconfiguration();
    bcecredentials credentials = null;
    if (ststoken != null && !ststoken.equalsignorecase("")) {
      credentials = new defaultbcesessioncredentials(ak, sk, ststoken);
    } else {
      credentials = new defaultbcecredentials(ak, sk);
    }
    config.setendpoint(endpoint);
    config.setcredentials(credentials);
    return new bosclient(config);
  }

  public void uploadfile(string bucket, string object, file file) {
    client.putobject(bucket, object, file);
  }

  public void uploadfile(string bucket, string object, inputstream inputstream) {
    client.putobject(bucket, object, inputstream);
  }

  public void uploadfile(string bucket, string object, byte[] data) {
    client.putobject(bucket, object, data);
  }

  public byte[] downloadfilecontent(string bucket, string object) {
    return client.getobjectcontent(bucket, object);
  }
}
```

上传文件到 BOS 代码实现

```
public void uploadPicToBos() {
  // 1. get pic params from ui: file name, file location uri etc
  // 2. send params to app server and get sts, bucket name and region
  // 3. upload selected pic to bos with sts etc, which bos client needs

  EditText et = (EditText) findViewByld(R.id.app_server_addr_edittext);
  final String appServerAddr = et.getText().toString();

   new Thread(new Runnable() {
    @Override
    public void run() {
      Map<String, Object> bosInfo = AppServer.getBosInfoFromAppServer(appServerAddr, "user-demo",
          AppServer.BosOperationType.UPLOAD);

      if (bosInfo == null) {
        return;
      }
      showToast(bosInfo.toString(), Toast.LENGTH_LONG);

      String ak = (String) bosInfo.get("ak");
      String sk = (String) bosInfo.get("sk");
      String stsToken = (String) bosInfo.get("stsToken");
      String endpoint = (String) bosInfo.get("endpoint");
      String bucketName = (String) bosInfo.get("bucketName");
      String objectName = (String) bosInfo.get("objectName");
      String prefix = (String) bosInfo.get("prefix");
      Log.i("UploadFileToBos", bosInfo.toString());

      // specify a object name if the app server does not specify one
      if (objectName == null || objectName.equalsIgnoreCase("")) {
        objectName = ((EditText) findViewByld(R.id.bos_object_name_edittext)).getText().toString();
        if (prefix != null && !prefix.equalsIgnoreCase("")) {
          objectName = prefix + "/" + objectName;
        }
      }

      Bos bos = new Bos(ak, sk, endpoint, stsToken);
      try {
        byte[] data = Utils.readAllFromStream(MainActivity.this.getContentResolver().openInputStream(selectedPicUri));
        bos.uploadFile(bucketName, objectName, data);
      } catch (Throwable e) {
        Log.e("MainActivity/Upload", "Failed to upload file to bos: " + e.getMessage());
        showToast("Failed to upload file: " + e.getMessage());
        return;
      }
      // finished uploading file, send a message to inform ui
      handler.sendEmptyMessage(UPLOAD_FILE_FINISHED);
    }
  }).start();
}
```

从 APP Server 上获取 BOS 信息代码实现

```
public class AppServer {
  /**
   * get info from app server for the file to upload to or download from BOS
   *
   * @param appServerEndpoint app server
   * @param userName          the app user's name, registered in app server
   * @param bosOperationType  download? upload? or?
   * @return STS, and BOS endpoint, bucketName, prefix, path, object name etc
   */
  public static Map<String, Object> getBosInfoFromAppServer(String appServerEndpoint, String userName, BosOperationType bosOperationType) {
      String type = "";
      switch (bosOperationType) {
          // to simplify
          case UPLOAD: {
              type = "upload";
              break;
          }
          case DOWNLOAD: {
              type = "download";
              break;
          }
          case DOWNLOAD_PROCESSED: {
              type = "download-processed";
              break;
          }
          default:{
              break;
          }
      }
      // TODO: this url should be url encoded
      String appServerUrl = appServerEndpoint + "/?" + "userName=" + userName + "&command=stsToken&type=" + type;

      // create a http client to contact app server to get sts
      HttpParams httpParameters = new BasicHttpParams();
      HttpClient httpClient = new DefaultHttpClient(httpParameters);

      HttpGet httpGet = new HttpGet(appServerUrl);
      httpGet.addHeader("User-Agent", "bos-meitu-app/demo");
      httpGet.setHeader("Accept", "*/*");
      try {
          httpGet.setHeader("Host", new URL(appServerUrl).getHost());
      } catch (MalformedURLException e) {
          e.printStackTrace();
      }
      httpGet.setHeader("Accept-Encoding", "identity");

      Map<String, Object> bosInfo = new HashMap<String, Object>();
      try {
          HttpResponse response = httpClient.execute(httpGet);
          if (response.getStatusLine().getStatusCode() != 200) {
              return null;
          }
          HttpEntity entity = response.getEntity();
          long len = entity.getContentLength();
          InputStream is = entity.getContent();
          int off = 0;
          byte[] b = new byte[(int) len];
          while (true) {
              int readCount = is.read(b, off, (int) len);
              if (readCount < 0) {
                  break;
              }
              off += readCount;
          }
          Log.d("AppServer", new String(b, "utf8"));
          JSONObject jsonObject = new JSONObject(new String(b, "utf8"));
          Iterator<String> keys = jsonObject.keys();
          while (keys.hasNext()) {
              String key = keys.next();
              bosInfo.put(key, jsonObject.get(key));
          }
      } catch (IOException e) {
          e.printStackTrace();
          return null;
      } catch (JSONException e) {
          e.printStackTrace();
          return null;
      }
      return bosInfo;
  }

  public enum BosOperationType {
      UPLOAD,
      DOWNLOAD,
      DOWNLOAD_PROCESSED,
  }
}
```

APP Server端代码样例

APP Server 端基于 Jetty 框架，接口主要处理 Android APP 获取 BOS 信息的请求。APP Server 端会返回临时 AK、SK、Session Token、bucket 名称、资源路径和资源请求的 Endpoint 等参数。以下为 Jetty 处理的代码示例。

```
 @Override
public void handle(String target, Request baseRequest, HttpServletRequest request, HttpServletResponse response)
        throws IOException, ServletException {

    // Inform jetty that this request has now been handled
    baseRequest.setHandled(true);

    if (!request.getMethod().equalsIgnoreCase("GET")) {
        response.setStatus(HttpServletResponse.SC_NOT_IMPLEMENTED);
        return;
    }

    // expected url example: localhost:8080/?command=stsToken&type=download
    Map<String, String[]> paramMap = request.getParameterMap();
    if (paramMap.get("command") == null || paramMap.get("type") == null) {
        // invalid request
        response.setStatus(HttpServletResponse.SC_BAD_REQUEST);
        return;
    }

    if (!paramMap.get("command")[0].equalsIgnoreCase("stsToken")) {
        response.setStatus(HttpServletResponse.SC_NOT_IMPLEMENTED);
        return;
    }

    String responseBody = "";
    responseBody = getBosInfo(paramMap.get("type")[0]);

    // Declare response encoding and types
    response.setContentType("application/json; charset=utf-8");

    // Declare response status code
    response.setStatus(HttpServletResponse.SC_OK);

    // Write back response, utf8 encoded
    response.getWriter().println(responseBody);
}

/**
 * Generates bos info needed by app according to requset type(upload, download etc)
 * this is the key part for uploading file to bos with sts token
 * @param bosRequestType
 * @return utf8 encoded json string
 */
public String getBosInfo(String bosRequestType) {
    // configuration for getting stsToken
    // bce bos credentials ak sk
    String bosAk = "your_bos_ak";
    String bosSk = "your_bos_sk";
    // bce sts service endpoint
    String stsEndpoint = "http://sts.bj.baidubce.com";

    BceCredentials credentials = new DefaultBceCredentials(bosAk, bosSk);
    BceClientConfiguration clientConfig = new BceClientConfiguration();
    clientConfig.setCredentials(credentials);
    clientConfig.setEndpoint(stsEndpoint);
    StsClient stsClient = new StsClient(clientConfig);
    GetSessionTokenRequest stsReq = new GetSessionTokenRequest();
    // request expiration time
    stsReq.setDurationSeconds(1800);
    GetSessionTokenResponse stsToken = stsClient.getSessionToken(stsReq);
    String stsTokenAk = stsToken.getCredentials().getAccessKeyId();
    String stsTokenSk = stsToken.getCredentials().getSecretAccessKey();
    String stsTokenSessionToken = stsToken.getCredentials().getSessionToken();

    // **to simplify this demo there is no difference between "download" and "upload"**
    // parts of bos info
    String bosEndpoint = "http://bj.bcebos.com";
    String bucketName = "bos-android-sdk-app";
    if (bosRequestType.equalsIgnoreCase("download-processed")) {
        // the binded image processing domain set by App developer on bce console
        bosEndpoint = "http://" + bucketName + ".bj.bcebos.com";
    }

    // prefix is the bucket name, and does not specify the object name
    BosInfo bosInfo = new BosInfo(stsTokenAk, stsTokenSk, stsTokenSessionToken, bosEndpoint,
            bucketName, "", bucketName);

    String res = "";
    ObjectMapper mapper = new ObjectMapper();
    try {
        res = mapper.writerWithDefaultPrettyPrinter().writeValueAsString(bosInfo);
    } catch (JsonProcessingException e) {
        e.printStackTrace();
        res = "";
    }
    try {
        res = new String(res.getBytes(), "utf8");
    } catch (UnsupportedEncodingException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
        res = "";
    }
    return res;
}
```

HTTPS传输加密实践

HTTP与HTTPS

HTTP与HTTPS的概念

HTTP（HyperText Transfer Protocol），即超文本传输协议，是互联网上应用最为广泛的一种网络协议。设计HTTP最初的目的是为了提供一种发布和接收HTML页面的方法。通过HTTP或者HTTPS协议请求的资源由统一资源标识符（Uniform Resource Identifiers，URI）来标识。

HTTP协议以明文方式发送内容，不提供任何方式的数据加密，如果攻击者截取了Web浏览器和网站服务器之间的传输报文，就可以直接读懂其中的信息，因此HTTP协议不适合传输一些敏感信息，比如信用卡号、密码等。

HTTPS（Hypertext Transfer Protocol over Secure Socket Layer），是以安全为目标的HTTP通道，简单讲是HTTP的安全版。HTTPS在HTTP的基础上加入了SSL协议，SSL依靠证书来验证服务器的身份，并为浏览器和服务器之间的通信加密。

HTTP与HTTPS的区别

HTTPS和HTTP的区别主要集中在以下4点：

- HTTPS协议需要服务端先到CA申请证书。
- HTTP是明文传输，HTTPS 则是具有安全性的SSL加密传输协议。
- HTTP和HTTPS使用的是完全不同的连接方式，用的端口也不一样，前者是80，后者是443。
- HTTP的连接很简单，是无状态的，而HTTPS协议是由SSL+HTTP协议构建的可进行加密传输、身份认证的网络协议，比HTTP协议更安全。

使用HTTPS保障BOS数据安全性

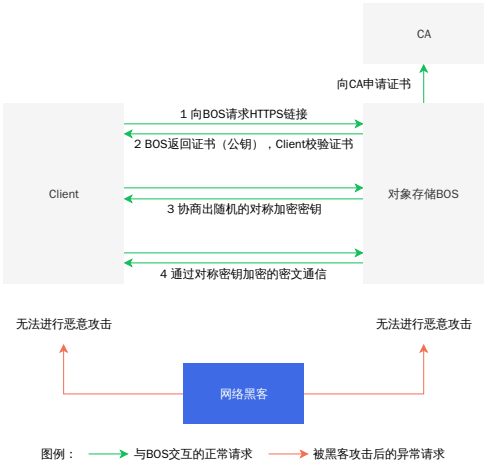
BOS目前已经全面支持HTTP与HTTPS的传输协议，使用HTTP和HTTPS与BOS的交互过程如下图所示。

使用HTTP



Client端使用HTTP协议进行传输时，由于HTTP属于明文传输，在传输过程中可能存在被中间方截取、篡改等恶意攻击行为，存在一定的安全隐患。

使用HTTPS



HTTPS属于加密传输，BOS已向CA申请了证书。该证书只有用于BOS服务器的时候，Client端才信任该主机。

在传输过程中，BOS会先返回给Client端该证书信息，并由Client端进行合法性认证，认证通过后，Client端与BOS会协商出一个随机的对称加密密钥，其后Client端与BOS之间的通讯请求都会被安全的对称加密算法进行加密处理。因此， 即使传输过程中被恶意的中间方截获，由于是密文，也无法进行恶意攻击，可以有效的保障 传输数据的安全性。

性能影响

由于HTTP是明文传输，HTTPS是加密传输，因此两者在响应时间QPS、吞吐量（网卡）、资源消耗三方面都存在区别，其中响应时间包括RTT（Round Trip Time）、加密解密耗时、后端读写处理耗时等。具体影响程度会跟OpenSSL的版本、使用的加密算法、和到BOS Server端的RRT等诸多因素有关。

与HTTP相比，HTTPS需要经过SSL交互过程，因此会比HTTP过程多两个RTT的延时损耗。比如：Ping时间是40ms，HTTPS比HTTP的延时会增加80ms。

总体来看，在网络条件较好，且文件较大时，选择HTTPS几乎没有影响；当网络条件较差，文件较小时，如图片处理等场景，HTTPS的延时会放大。

配置示例

为了您的数据在传输过程中的可靠性，我们推荐您使用HTTPS协议。目前，BOS的API和SDK已经全面支持配置HTTPS。

API方式

使用API访问时，根据您使用的语言不同，将对应的端口协议配置为HTTPS即可。

SDK方式

使用SDK访问时，不同语言SDK的配置方法请参考：

- Java SDK- 配置HTTPS协议访问BOS
- Python SDK- 配置HTTPS协议访问BOS

- PHP SDK- 配置HTTPS协议访问BOS
- C# SDK- 配置HTTPS协议访问BOS
- Android SDK- 配置HTTPS协议访问BOS

混合云中使用专线-VPN访问BOS

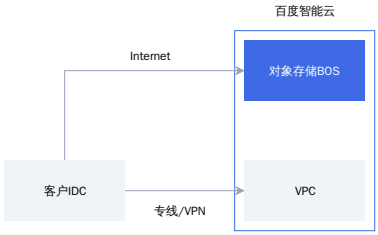
概览

该教程为混合云中使用专线/ VPN 通过内网访问 BOS 提供解决方案。

应用场景

混合云融合了公有云和私有云，是近年来云计算的主要模式和发展方向。企业出于安全考虑更愿意将数据存放在私有云中，但是同时又希望获得公有云的计算资源。百度智能云混合云方案中客户 IDC 和百度智能云 VPC 通过专线/ VPN 打通，实现云上云下业务紧密连接，既可以利用云上资源的按需使用和易扩展的特性，又可以利用本地 IDC 满足合规性要求。

专线/ VPN 能为客户 IDC 访问 BOS 提供高速安全的传输通道。但在混合云方案中，客户 IDC 访问 BOS 时（例如把本地数据备份到云上），由于客户 IDC 解析 BOS 域名得到的是 BOS 公网 IP 地址，所以数据流还是走公网传输，无法直接通过专线/ VPN内网访问BOS服务。



实践方案

方案1：使用VPC服务网卡

服务网卡是一个高可用高扩展性的 VPC 访问 BOS 代理，可以将 BOS 等 VPC 外部公共服务映射到 VPC 内部，也可以挂载服务发布点生成的服务域名，用户可以在 VPC 内或者混合云对端通过内网便捷、安全地访问这些服务。

> **配置前您需要了解**：
>
> - 每个VPC可支持20个服务网卡。
> - 一块服务网卡仅属于一个子网，其IP地址必须属于该子网CIDR块中。
> - 单服务每个子网仅支持一个服务网卡。
> - 服务网卡后端可挂载本地域服务。

Step1：创建服务网卡

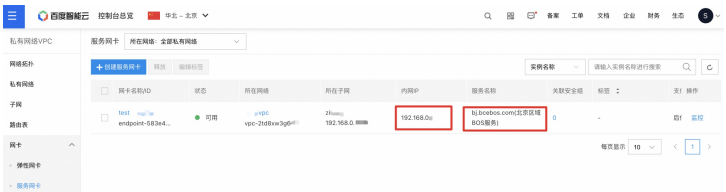
登录服务网卡控制台点击创建服务网卡，对服务网卡进行配置。

> **配置项说明**：
>
> - 请选择与需要访问的bucket相同的地域，跨地域访问依然会通过公网访问。
> - 服务网卡的IP地址可在所在的子网段内指定。
> - 挂载服务选择对象存储，并选择对应的BOS服务Endpoint。



Step2：确认服务网卡配置

创建完成后请在服务网卡列表中查看创建的服务网卡相关配置，并确认服务网卡对应的服务名称与要访问的BOS服务Endpoint相同。



Step3：通过服务网卡访问BOS服务

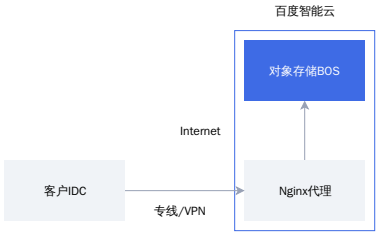
确认服务网卡和专线/VPN的配置均无误后，可在IDC内将访问BOS服务的域名替换为服务网卡的内网IP，即可通过专线/VPN直接通过内网访问BOS服务。

方案2：使用BCC作为访问代理

概述

混合云中使用专线/ VPN 访问 BOS 方案的核心是使用 BCC 作为访问代理：

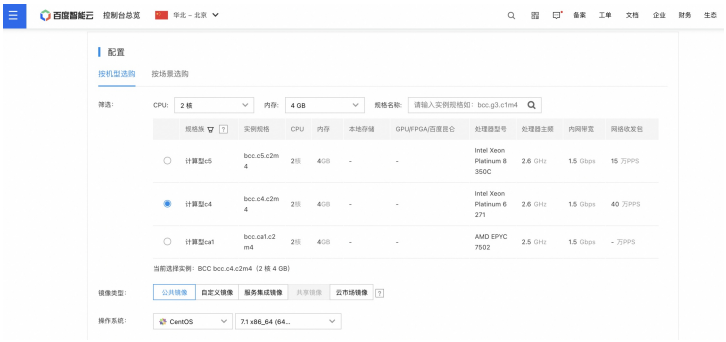
1. 首先申请一台 BCC 并安装 Nginx，作为到 BOS 的反向代理；
2. 修改客户 IDC 机器的本地 DNS 记录，把访问 BOS 的请求指向 Nginx 代理。



> **注意：**本方案只能保证同区域下访问 BOS 走专线/ VPN 和内网。例如客户 IDC 和百度智能云的北京 VPC 打通，本方案可以利用专线/ VPN 访问位于北京的 BOS Bucket；但如果客户去访问位于广州的 BOS Bucket，数据还是会走公网传输。

配置Nginx反向代理

1. **首先申请一台 BCC**。推荐配置：2 核 CPU、4G 内存、CentOS 7.1，无需配置公网 IP。



2. **安装 Nginx**。由于我们需要额外的 stream 模块来提供 TCP 协议代理，所以还需要安装 stream 模块。

a) `yum install nginx nginx-mod-stream`

b) 配置 Nginx：

```
 cd /nginx/
vim nginx.conf
```

把 nignx.conf 修改成如下配置，注意下述配置适用于北京区域，如果想访问广州区域的 BOS，把配置文件里的 bj 替换成 gz 即可：

```
 #user  work;
worker_processes  auto;

events {
    worker_connections  1024;
}

stream {
    error_log logs/access.log info;

    # The backend address of BOS in Beijing
    upstream httpbosbj {
        server bj.bcebos.com:80;
    }

    upstream httpsbosbj {
        server bj.bcebos.com:443;
    }

    server {
        listen       80;

        proxy_pass httpbosbj;
        # Proxy settings
        proxy_connect_timeout   5s;
        proxy_timeout           90s;
        proxy_buffer_size       4k;
    }

    server {
        listen       443;

        proxy_pass httpsbosbj;
        # Proxy settings
        proxy_connect_timeout   5s;
        proxy_timeout           90s;
        proxy_buffer_size       4k;
    }
}
```

c) 验证配置文件并启动 Nginx（需要 root ）：

```
 nginx -t
mkdir –p /usr/share/nginx/logs
```

确认 Nginx 成功启动：

```
[root@host-192-168-1-5 sbin]# ps aux | grep nginx
root      15545  0.0  0.0  24472   512 ?        Ss   10:28   0:00 nginx: master process /usr/local/nginx/sbin/nginx
nobody    15546  0.0  0.1  26916  1296 ?        S    10:28   0:00 nginx: worker process
root      15550  0.0  0.0 112644   976 pts/0    R+   10:28   0:00 grep --color=auto nginx
```

配置客户IDC节点

配置本地 DNS 记录，让客户 IDC 机器把访问 BOS 的请求指向到 Nginx 中转机。例如在 CentOS 环境下，在 /etc/hosts 中添加记录：

`nginxIp bj.bcebos.com`

这里 nginxIp 就是上面配置的 Nginx 反向代理的内网 IP 地址，eg. 192.168.1.5。Windows 等环境下修改本地 DNS 记录同理。

测试

配置完成后您就可以使用专线/ VPN 从 IDC 机器访问 BOS 了，您可以使用 BOS SDK、CLI、BOS 桌面等各种工具进行文件上传和管理。

此外，您还可以登录 Nginx 代理节点检查日志来确认 Nginx 反向代理工作状态是否正常：

`less /usr/local/nginx/logs/access.log`

```
2016/11/15 16:58:21 [info] 27465#0: *37 client 192.168.1.6:48862 connected to 0.0.0.0:443
2016/11/15 16:58:21 [info] 27465#0: *37 proxy 192.168.1.5:36816 connected to 10.16.83.135:443
2016/11/15 16:58:22 [info] 27465#0: *37 client disconnected, bytes from/to client:763/3990, byt
es from/to upstream:3990/763
```

> **说明**:
>
> - 本教程提供的方案可以满足大多数情况下的访问 BOS 需求。在 2 核 4G 配置下，代理节点可以支撑 1000 并发连接（这是由于 BOS 出于防 DDoS 原因限制了单 IP 的并发访问连接数）和 1Gbps 的访问速度。
> - 不过本方案有如下限制：(1) Nginx 代理是潜在的单点故障，您需要维护这台 BCC 实例保证其运行正常；(2)如果需要更高的上传速度和更多的并发连接，可以进一步提高 Nginx 节点配置或增加额外 Nginx 节点等。

相关产品

云服务器BCC、专线 ET

CDN动态加速上传数据到BOS

概览

本文主要介绍如何利用 DRCDN 的动态加速特性来提升客户端数据上传 BOS 过程的传输速度和稳定性。

需求场景

为了满足用户在使用 BOS 上传数据场景中的加速需求。该功能充分发挥百度智能云 DRCDN 业界领先的架构优势，通过上传链路优化、私有协议、实时适配、智能路径切换等手段，极大地缩短了数据上行传输的时间，提高文件上传到 BOS 的速率。

BOS 的 DRCDN 动态加速加速功能可以满足客户在音频、视频、图片等多种数据上行传输应用场景下的加速需求，如：

- 普通文件上传加速场景
- 小文件上传场景
- 远距离的文件上传场景
- 网络拥塞等弱网环境下文件上传场景
- 跨网跨运营商文件上传场景

优势

使用 DRCDN 动态加速上传数据到 BOS 有以下优势：

- 通过百度遍布全球的优质 DRCDN 节点，为用户解决跨运营商、跨国、网络不稳定等因素导致的上传 BOS 速度慢、数据丢包、服务不稳定等问题，为用户提供快速、可靠和流畅的数据上传体验；
- 百度自研的专门针对文件上行传输全程的综合优化加速方案，通过链路优化、协议层优化、路由优化、传输优化、智能路径切换等核心技术，中心节点多线接入，实时探测和智能选择回源链路，获取用于传输的最优路径以显著提升上传加速效果；
- DRCDN 动态加速支持动态 HTTPS 请求加速，快速传输加密数据，同时具备 DDoS、CC 攻击抵御能力，保障数据上传过程的可用性和安全性。

实践步骤

以用户上传视频、图片等二进制文件为例，使用 DRCDN 动态加速上传数据到 BOS 的存储方案可以对上传过程优化，在有限增加成本的情况下提升上传速度和上传体验。具体的解决方案和步骤如下：

步骤1: 开启DRCDN上行加速功能

1. 登录控制台并打开需要开通上行加速的 Bucket 发布管理页面，开启该 Bucket 官方域名加速，选择加速类型为 DRCDN动态加速，开启后会产生 DRCDN 动态加速费用。
2. 详细的开启方法请参考 控制台操作指南-数据发布-DRCDN上行加速。

步骤2: 使用DRCDN加速上传数据

BOS 提供了 console 上传、使用 API 或 SDK 上传、使用 BOS CLI 上传等多种上传文件的方案，您可以选择使用 API 或 SDK 上传。

通过API加速上传文件 通过 BOS API 的 PutObject 接口上传数据到 BOS 时，只需要将 API 请求的 HOST 替换为 DRCDN 加速域名（即您自定义的加速域名或者官方 CDN 加速域名+'/'+'文件名'），即通过 DRCDN 加速上传。例如： 原上传请求：

```
PUT /<ObjectName> HTTP/1.1
Host: <BucketName>.bj.bcebos.com
Date: <Date>
Authorization: <AuthorizationString>
Content-Type: text/plain
Content-Length: <Content_Length>
```

CDN加速上传请求：

```
PUT /<ObjectName> HTTP/1.1
Host: <BucketName>.cdn.bcebos.com
Date: <Date>
Authorization: <AuthorizationString>
Content-Type: text/plain
Content-Length: <Content_Length>
```

通过SDK加速上传文件

不同语言版本的 SDK 配置使用 DRCDN加速上传的方式不同，本例以通过 Go SDK 上传一个文件为例，其它语言 SDK 使用方式参考对应的 SDK 文档。

1. SDK 配置 DRCDN 加速上传的加速域名作为 ENDPOINT，例如`cdn-test.cdn.bcebos.com`是在控制台开通 DRCDN 动态加速功能之后得到的 cdn-test 这个 bucket 对应的 CDN 加速域名，代码如下：

```
AK, SK := "xxx", "yyy"
ENDPOINT := "cdn-test.cdn.bcebos.com"
bosClient, _ := bos.NewClient(AK, SK, ENDPOINT)
```

2. 配置开启 DRCDN 加速上传开关，需要注意的是，CnameEnabled 这个配置项表示的是是否使用自定义域名上传文件到 BOS，如果你的 ENDPOINT 是官方提供的以`.cdn.bcebos.com`为后缀的加速域名，可以不配置该开关；如果你的 ENDPOINT 是自定义加速域名，那么一定要把 CnameEnabled 设置为 true 才会生效。

```
bosClient.Config.CnameEnabled = true
```

3. 使用配置后的 bosClient 进行文件上传操作体验 CDN 加速，上传完成后检查文件是否上传成功。

```
bosClient.PutObjectFromFile(bucketName, objectName, filePath, nil)
```

配置重试备用域名

在支持配置 DRCDN 域名加速上传文件到 BOS 的基础上，考虑到上传过程的稳定性，SDK 进一步支持了主域名上传失败时重试备用域名机制，使用方式如下：

```
// 如果使用ENDPOINT上传文件失败，SDK会自动重试BackupEndpoint域名
bosClient.Config.BackupEndpoint = "cdn-test.bj.bcebos.com"
```

相关产品

内容分发网络CDN、动态加速 DRCDN

使用CDN加速BOS

概览

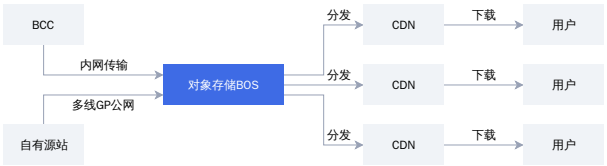
本文介绍如何利用内容分发网络 CDN 对 BOS 中数据进行加速分发。

需求场景

存储分发场景下，BOS用于存放网站的静态图片、视频文件和应用服务的下载内容等文件。存储分发场景通常有以下特点：

- 静态文件访问量大，访问频率高，服务器负载高
- 静态文件面对用户基数非常大，并发访问强度大，网络环境又极为复杂
- 某个时间段需要高速、集中的下载，且并发下载量高，比如应用更新包、程序补丁等等

使用CDN加速BOS，通过CDN服务加速分发能帮助用户就近获得BOS存储服务中存放的文件，极大地提高用户体验。



优势

使用CDN加速BOS搭建存储分发场景有以下优势：

- 海量存储空间，存储对象数量不受限制；
- 存储费用极低，BOS的存储费用远低于云磁盘，且支持标准存储和低频存储等多种存储类型；
- 极大地降低Web服务器负载，对静态文件访问全部通过CDN来承担；
- 通过百度分布在全国的CDN节点分发，可以极大地缩短用户查看内容的延迟，提高用户访问网站的响应速度与网站的可用性，解决网络带宽小、用户访问量大、网点分布不均等问题。

实践操作

以搭建某个站点为例，通常网站包含主站、静动态文件及程序代码。使用CDN加速BOS的存储分发方案可以对网站进行架构优化，在有限增加成本的情况下提升用户访问体验。具体的解决方案和步骤如下：

1. 对BCC或任意位置上的网站程序进行整理，区分出动态文件和静态资源，并将静态文件存储到BOS中。
    - 建立一个resources目录，用于存放所有网站静态资源的目录；
    - 在BOS上创建Bucket用于存放网站静态资源的。

> **注意：** Bucket的“读写权限”权限必须为“公共读”以上权限才能实现外网对于资源的匿名访问。如果已经创建好Bucket，则可以在Bucket设置中进行修改。
>
> 如果需要防止盗链或访问权限保护，可以使用Referer字段或使用AK/SK计算的签名来访问，具体请参考防盗链。

2. 开通CDN

   CDN加速域名开通有“使用官方的默认加速域名”和“使用个人/公司申请的备案域名”两种方法。

   - 使用官方的默认加速域名
     - 创建Bucket的时候，将CDN官方加速选项设置为ON，即可直接获得官方CDN加速的域名。
     - 如果Bucket已经创建完成，可以在Bucket的“发布管理”页面中开启CDN加速。详情参考CDN官方域名加速。
   - 使用自定义的加速域名

     自定义的加速域名指用户已经使用个人或公司申请的完成备案的域名。

     选中要加速的Bucket并进入“数据发布”页面，点击“添加自定义加速域名”。域名需要解析成功才能通过CDN加速。

3. 上传文件并体验CDN加速

   BOS提供了多种上传文件的方案，您可以选择从console上传、使用API或SDK上传、使用BOS CLI上传等方式。

   本例以通过console上传一个文件为例。

   - 上传完成后，进入“Bucket管理”的“数据管理”页签，检查上传文件对象是否存在。如果存在，那么该文件的CDN加速的访问地址，通常为自定义的加速域名或者官方CDN加速域名+'/'+'文件名'。

     如果您有多个CDN加速域名与该Bucket绑定，则多个CDN加速域名都可以生效。
   - 在应用系统中，将对原静态资源访问的URL替换为加速访问的地址（您自定义的加速域名或者官方CDN加速域名+'/'+'文件名'），即通过CDN加速后访问。

     修改完成后对网站静态文件的访问都会通过BOS+CDN形式访问，不仅减少了您机器资源的消耗，也提高了用户访问的速度，提升用户体验

4. 测试CDN加速

   使用浏览器，通过您自定义的加速域名或者官方CDN加速域名+'/'+'文件名'，访问BOS上的资源。

相关参考

- CDN预加载
  - 启用预加载功能可以让用户首次访问的时候就体验到CDN加速（预热到CDN中），具体操作参考CDN预热缓存。
- CDN配置规则
  - 若没有在CDN中配置过期时间，且源站没有显示提供http缓存控制头，CDN不缓存数据。
  - 可在CDN中配置过期时间，用以自动更新缓存，具体操作请参考配置缓存过期时间。

Python API请求访问BOS

基础(必看)：鉴权字符串生成

鉴权字符串生成

使用原始API进行访问BOS，需要自行计算鉴权信息。使用流程：

1. 准备 ak/sk
2. python3.6及以上环境 **生成鉴权字符串完整demo** 示例生成auth_key以GET请求访问Bucket为例

- 鉴权字符串算法代码

```
 import hashlib
import hmac
import string
import datetime


AUTHORIZATION = "authorization"
BCE_PREFIX = "x-bce-"
DEFAULT_ENCODING = 'UTF-8'


##### 保存AK/SK的类
class BceCredentials(object):
    def __init__(self, access_key_id, secret_access_key):
        self.access_key_id = access_key_id
        self.secret_access_key = secret_access_key


##### 根据RFC 3986，除了：
##### 1.大小写英文字符
##### 2.阿拉伯数字
##### 3.点'.'、波浪线'~'、减号'-'以及下划线'_'
##### 以外都要编码
RESERVED_CHAR_SET = set(string.ascii_letters + string.digits + '.~-_')


def get_normalized_char(i):
    char = chr(i)
    if char in RESERVED_CHAR_SET:
        return char
    else:
        return '%%%02X' % i


NORMALIZED_CHAR_LIST = [get_normalized_char(i) for i in range(256)]


##### 规范化字符串
def normalize_string(in_str, encoding_slash=True):
    if in_str is None:
        return ''
```

```
    # 如果输入是unicode，则先使用UTF8编码之后再编码
    # in_str = in_str.encode(DEFAULT_ENCODING) if isinstance(in_str, unicode) else str(in_str)
    in_str = in_str if isinstance(in_str, str) else str(in_str)

    # 在生成规范URI时。不需要对斜杠'/'进行编码，其他情况下都需要
    if encoding_slash:
        encode_f = lambda c: NORMALIZED_CHAR_LIST[ord(c)]
    else:
        # 仅仅在生成规范URI时。不需要对斜杠'/'进行编码
        encode_f = lambda c: NORMALIZED_CHAR_LIST[ord(c)] if c != '/' else c

    # 按照RFC 3986进行编码
    return ''.join([encode_f(ch) for ch in in_str])


##### 生成规范URI
def get_canonical_uri(path):
    # 规范化URI的格式为：/{bucket}/{object}，并且要对除了斜杠"/"之外的所有字符编码
    return normalize_string(path, False)


##### 生成规范query string
def get_canonical_querystring(params):
    if params is None:
        return ''

    # 除了authorization之外，所有的query string全部加入编码
    result = ['%s=%s' % (k, normalize_string(v)) for k, v in params.items() if k.lower != AUTHORIZATION]

    # 按字典序排序
    result.sort()

    # 使用&符号连接所有字符串并返回
    return '&'.join(result)


##### 生成规范header
def get_canonical_headers(headers, headers_to_sign=None):
    headers = headers or {}

    # 没有指定header_to_sign的情况下，默认使用：
    #   1.host
    #   2.content-md5
    #   3.content-length
    #   4.content-type
    #   5.所有以x-bce-开头的header项
    # 生成规范header
    if headers_to_sign is None or len(headers_to_sign) == 0:
        headers_to_sign = {"host", "content-md5", "content-length", "content-type"}

    # 对于header中的key，去掉前后的空白之后需要转化为小写
    # 对于header中的value，转化为str之后去掉前后的空白
    # f = lambda (key, value): (key.strip().lower(), str(value).strip())
    f = lambda item: (item[0].strip().lower(), str(item[1]).strip())

    result = []
    for k, v in map(f, headers.items()):
        # 无论何种情况，以x-bce-开头的header项都需要被添加到规范header中
        if k.startswith(BCE_PREFIX) or k in headers_to_sign:
            result.append("%s:%s" % (normalize_string(k), normalize_string(v)))

    # 按照字典序排序
    result.sort()

    # 使用\n符号连接所有字符串并返回
    return '\n'.join(result)


##### 签名主算法
def sign(credentials, http_method, path, headers, params, expiration_in_seconds=1800, headers_to_sign=None):
    headers = headers or {}
    params = params or {}
    x_bce_date = headers.get("x-bce-date")
    # timestamp = int(datetime.datetime.strptime(x_bce_date, "%Y-%m-%dT%H:%M:%SZ").timestamp())
    # 1.生成sign key
    # 1.1.生成auth-string，格式为：bce-auth-v1/{accessKeyId}/{timestamp}/{expirationPeriodInSeconds}
    sign_key_info = 'bce-auth-v1/%s/%s/%d' % (
        credentials.access_key_id,
        x_bce_date,
        expiration_in_seconds)
    # 1.2.使用auth-string加上SK，用SHA-256生成sign key
    sign_key = hmac.new(
        credentials.secret_access_key.encode("utf-8"),
        sign_key_info.encode("utf-8"),
        hashlib.sha256).hexdigest()

    # 2.生成规范化uri
    canonical_uri = get_canonical_uri(path)

    # 3.生成规范化query string
    canonical_querystring = get_canonical_querystring(params)

    # 4.生成规范化header
    canonical_headers = get_canonical_headers(headers, headers_to_sign)

    # 5.使用'\n'将HTTP METHOD和2、3、4中的结果连接起来，成为一个大字符串
    string_to_sign = '\n'.join(
        [http_method, canonical_uri, canonical_querystring, canonical_headers])
```

```
    # 6.使用5中生成的签名串和1中生成的sign key，用SHA-256算法生成签名结果
    sign_result = hmac.new(sign_key.encode("utf-8"), string_to_sign.encode("utf-8"), hashlib.sha256).hexdigest()

    # 7.拼接最终签名结果串
    if headers_to_sign:
        # 指定header to sign
        result = '%s/%s/%s' % (sign_key_info, ';'.join(headers_to_sign), sign_result)
    else:
        # 不指定header to sign情况下的默认签名结果串
        result = '%s//%s' % (sign_key_info, sign_result)
    return result


def get_x_bce_date(date=None):
    """获取UTC格式的时间"""
    if date is None:
        format_string = "%Y-%m-%dT%H:%M:%SZ"
        date = datetime.datetime.utcnow().strftime(format_string)
    return date

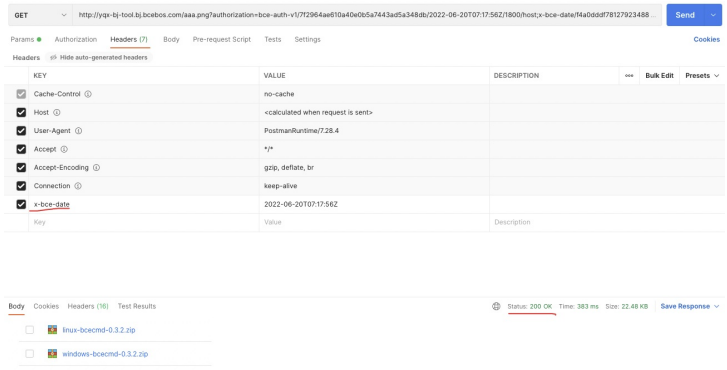

def generate_bos_auth(path, ak, sk, method, headers, params, expire_seconds=1800):
    """生成bos的鉴权字符串，过期时间默认1800s"""
    credentials = BceCredentials(ak, sk)
    return sign(credentials, method, path, headers, params, expire_seconds, headers.keys())
```

- 调用鉴权字符串

```
 access_key = "AK"
secret_key = "SK"
bucket_name = "bucketName"              # 替换为需要访问bucketName
object_key = "/aaa.png"                 # 在请求url没有参数时，访问的key的名字就是path，注意以 '/'开头
region = "bj.bcebos.com"                # bucket所在的区域，不同区域服务域名不同
request_method = "GET"                  # 请求方法类型 GET、PUT、HEAD、POST、DELETE
host = f"{bucket_name}.{region}"        # 根据bucket和服务域名组装访问域名
##### 构造访问的header信息，可以根据需要进行额外添加，例如：content-type字段
header = {
    "host": host,
    "x-bce-date": get_x_bce_date()
}
##### 如果请求在url中有带参数，也要参与鉴权
param = {}
##### 生成鉴权字符串
auth_key = generate_bos_auth(object_key, access_key, secret_key, request_method, header, param)
##### 构建完整的访问url
request_url = f"http://{host}{object_key}?{AUTHORIZATION}={auth_key}"
```

*将<鉴权字符串算法代码><调用鉴权字符串>两部分代码组成一个py文件即可执行***注意：**

1. 生成的auth_key只是一个authorization字段，需要拼接为request_url后才能进行正常访问。
2. request_url访问时需要带上x-bce-date字段，该字段尽量与生成签名时的x-bce-date字段相同，postman请求示例：



3. 更多详情参考生成签名字符串。

GET请求下载Bucket文件到本地

依赖

使用原始API进行访问Bucket，需要自行计算鉴权信息。使用流程：

1. 准备 ak/sk
2. python3.6及以上环境
3. 安装reqeusts包， pip3 install requests **GET请求下载Bucket文件demo**

- **永久ak-sk方式**

```
##### copy《基础(必看)：鉴权字符串生成》鉴权字符串算法代码

def default_ak_sk():
    """使用固定的ak-sk方式，即永久性的ak-sk"""
    access_key = "ak"     # 固定 ak
    secret_key = "sk"     # 固定 sk
    download(access_key, secret_key)


def download(ak, sk, token=None):
    """从Bucket中下载文件到本地"""
    bucket_name = "bucketName"        # 访问的bucket名称
    object_key = "/aaa.png"           # 在请求url没有参数时，访问的key的名字就是path，注意以 '/'开头
    region = "bj.bcebos.com"          # bucket所在的区域，不同区域服务域名不同
    http_method = "GET"               # 请求的方法类型 GET、PUT、HEAD、POST、DELETE
    host = f"{bucket_name}.{region}"    # 根据bucket和服务域名组装访问域名,有多种方式，这里只展示一种
    # 构造访问的header信息，可以根据需要进行额外添加，例如：content-type字段
    header = {
        "host": host,
        "x-bce-date": get_x_bce_date()
    }
    # 如果请求在url中有带参数，也要参与鉴权
    param = {}
    # 生成鉴权字符串
    auth_key = generate_bos_auth(object_key, ak, sk, http_method, header, param)
    # 构建完整的访问url
    request_url = f"http://{host}{object_key}?{AUTHORIZATION}={auth_key}"
    print(request_url)
    # 如果有token参数则添加到header中
    if token is not None:
        header['x-bce-security-token'] = token
    # GET请求
    resp = requests.get(url=request_url, headers=header)
    # 本地保存文件路径
    local_path = "/路径/文件名"
    save_local(resp, local_path)


if __name__ == '__main__':
    # 固定ak-sk方式下载文件
    default_ak_sk()
```

- **临时sts ak-sk-token方式**

  获取sts ak-sk-token

```
##### copy《基础(必看)：鉴权字符串生成》鉴权字符串算法代码

def sts_ak_sk():
    """使用临时ak-sk-token方式"""
    access_key = "8702e0c7f08211ec8168058160a92b86"      # 临时申请的 sts ak
    secret_key = "315fddebae1f4d65b561a296c5fbc2f3"      # 临时申请的 sts sk
    # 临时sts token信息示例，请替换为自行申请的ak/sk/token
    token =
"ZjkyZmQ2YmQxZTQ3NDcyNjk0ZTg1ZjYyYjlkZjNjODB8AAAAAJMCAACg8ipvvc5Ox8EFDupU5wGhWMBSjmRlbOexzgTxXUZHf+ZFUEWBFiDGGsLxoRG8FnEdmON/7tRwpmHk0mf9cnCwE7pzD1ovRASiLt7FfguU57A42Pk99

    download(access_key, secret_key, token)


def download(ak, sk, token=None):
    """从Bucket中下载文件到本地"""
    bucket_name = "bucketName"        # 访问的bucket名称
    object_key = "/aaa.png"           # 在请求url没有参数时，访问的key的名字就是path，注意以 '/'开头
    region = "bj.bcebos.com"          # bucket所在的区域，不同区域服务域名不同
    http_method = "GET"               # 请求的方法类型 GET、PUT、HEAD、POST、DELETE
    host = f"{bucket_name}.{region}"    # 根据bucket和服务域名组装访问域名,有多种方式，这里只展示一种
    # 构造访问的header信息，可以根据需要进行额外添加，例如：content-type字段
    header = {
        "host": host,
        "x-bce-date": get_x_bce_date()
    }
    # 如果请求在url中有带参数，也要参与鉴权
    param = {}
    # 生成鉴权字符串
    auth_key = generate_bos_auth(object_key, ak, sk, http_method, header, param)
    # 构建完整的访问url
    request_url = f"http://{host}{object_key}?{AUTHORIZATION}={auth_key}"
    # 如果有token参数则添加到header中
    if token is not None:
        header['x-bce-security-token'] = token
    # GET请求
    resp = requests.get(url=request_url, headers=header)
    # 本地保存文件路径
    local_path = "/路径/文件名"
    save_local(resp, local_path)


if __name__ == '__main__':
    # 临时ak-sk方式下载文件
    sts_ak_sk()
```

临时ak-sk-token和永久ak-sk生成鉴权auth_key没有区别，只是在header头信息中需要添加x-bce-security-token参数等于申请的token，将以上示例中完整代码保存到单个py文件即可执行。

PUT请求上传文件到Bucket

依赖

使用原始API进行访问Bucket，需要自行计算鉴权信息。使用流程：

1. 准备 ak/sk
2. python3.6及以上环境
3. 安装reqeusts包， `pip3 install requests` **PUT请求上传文件到Bucket Demo**

- **永久ak-sk方式**

```
 ##### copy《基础(必看)：鉴权字符串生成》鉴权字符串算法代码

def default_ak_sk():
    """使用固定的ak-sk方式，即永久性的ak-sk"""
    access_key = "ak"     # 固定 ak
    secret_key = "sk"     # 固定 sk
    upload(access_key, secret_key)


def upload(ak, sk, token=None):
    """上传文件"""
    bucket_name = "bucketName"
    object_key = "/aaabbb.png"          # object_key即上传后保存的名称，注意以 '/'开头
    region = "bj.bcebos.com"            # bucket所在的区域，不同区域服务域名不同
    http_method = "PUT"                 # 请求的方法类型 GET、PUT、HEAD、POST、DELETE
    host = f"{bucket_name}.{region}"    # 根据bucket和服务域名组装访问域名
    # 构造访问的header信息，可以根据需要进行额外添加，例如：content-type字段
    header = {
        "host": host,
        "x-bce-date": get_x_bce_date()
    }
    # 如果请求在url中有带参数，也要参与鉴权
    param = {}
    # 生成鉴权字符串
    auth_key = generate_bos_auth(object_key, ak, sk, http_method, header, param)
    # 构建完整的访问url
    request_url = f"http://{host}{object_key}?{AUTHORIZATION}={auth_key}"
    # 如果有token参数则添加到header中
    if token is not None:
        header['x-bce-security-token'] = token

    # 将文件读取为二进制流
    local_path = "/本地路径/文件名"
    with open(local_path, "rb") as f:
        # 发起put上传请求
        resp = requests.put(url=request_url, headers=header, data=f)
        print(resp)


if __name__ == '__main__':
    # 固定ak-sk方式上传文件
    default_ak_sk()
```

- **临时sts ak-sk-token方式**

  获取sts ak-sk-token

```
##### copy《基础(必看)：鉴权字符串生成》鉴权字符串算法代码

def sts_ak_sk():
    """使用临时ak-sk-token方式"""
    access_key = "8702e0c7f08211ec8168058160a92b86"      # 临时申请的 sts ak
    secret_key = "315fddebae1f4d65b561a296c5fbc2f3"      # 临时申请的 sts sk
    # 临时sts token信息示例，请替换为自行申请的ak/sk/token
    token =
"ZjkyZmQ2YmQxZTQ3NDcyNjk0ZTg1ZjYyYjlkZjNjODB8AAAAAJMCAACg8ipvvc5Ox8EFDupU5wGhWMBSjmRlbOexzgTxXUZHf+ZFUEWBFiDGGsLxoRG8FnEdmON/7tRwpmHk0mf9cnCwE7pzD1ovRASiLt7FfguU57A42Pk99

    upload(access_key, secret_key, token)

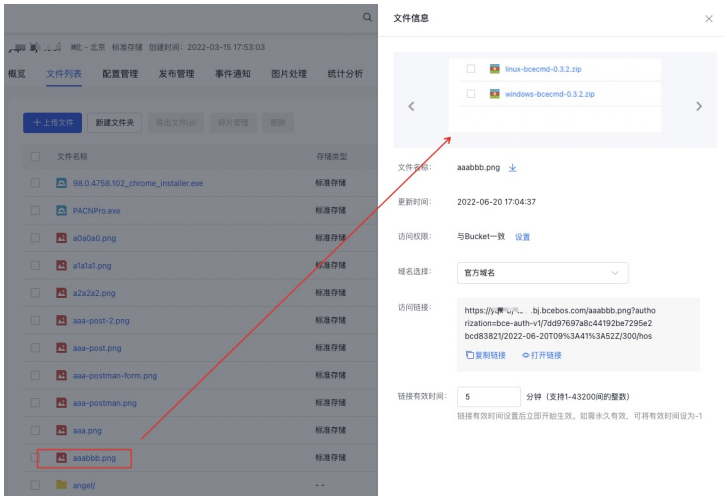

def upload(ak, sk, token=None):
    """上传文件"""
    bucket_name = "bucketName"
    object_key = "/aaabbb.png"          # object_key即上传后保存的名称，注意以 '/'开头
    region = "bj.bcebos.com"            # bucket所在的区域，不同区域服务域名不同
    http_method = "PUT"                 # 请求的方法类型 GET、PUT、HEAD、POST、DELETE
    host = f"{bucket_name}.{region}"    # 根据bucket和服务域名组装访问域名
    # 构造访问的header信息，可以根据需要进行额外添加，例如：content-type字段
    header = {
        "host": host,
        "x-bce-date": get_x_bce_date()
    }
    # 如果请求在url中有带参数，也要参与鉴权
    param = {}
    # 生成鉴权字符串
    auth_key = generate_bos_auth(object_key, ak, sk, http_method, header, param)
    # 构建完整的访问url
    request_url = f"http://{host}{object_key}?{AUTHORIZATION}={auth_key}"
    # 如果有token参数则添加到header中
    if token is not None:
        header['x-bce-security-token'] = token

    # 将文件读取为二进制流
    local_path = "/本地路径/文件名"
    with open(local_path, "rb") as f:
        # 发起put上传请求
        resp = requests.put(url=request_url, headers=header, data=f)
        print(resp)


if __name__ == '__main__':
    # 临时ak-sk方式上传文件
    sts_ak_sk()
```

临时ak-sk-token和永久ak-sk生成鉴权auth_key没有区别，只是在header头信息中需要添加x-bce-security-token参数等于申请的token，将以上示例中完整代码保存到单个py文件即可执行。 **注意：**

1. 将以上示例中完整代码保存到单个py文件即可执行。
2. 请求响应200即表示文件上传成功，也可以在控制台查看是否有相关文件



BOS专线回源IDC最佳实践

概览

该教程为混合云中使用BOS专线回源IDC最佳实践方案

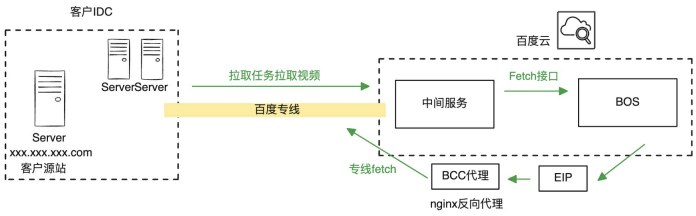
应用场景

该方案适用于客户通过回源方式从私有云往公有云迁移数据。混合云融合了公有云和私有云，企业希望在保证自己数据的隐私性条件下，利用到公有云的扩展性和稳定性能力来增加自己业务的鲁棒性。百度智能云混合云方案中客户 IDC 和百度智能云 VPC 通过专线打通，实现云上云下业务紧密连接。然后企业利用BOS自身的回源功能来实现客户私有云到BOS公有云的数据迁移。一方面，利用专线回源可以避开公网回源遇到的网络不稳定、网络带宽竞争等问题；另一方面，通过专线回源也能提高回源速度，方便用户快速数据上云。

数据迁移会有多种方案：数据流转平台CloudFlow迁移、BOS Import 脚本化工具、BOS镜像回源。本教程主要介绍专线下的回源方案

参考样例：客户站点数据上云

某客户希望通过镜像回源的方式，将企业站点的数据备份到云平台。客户对回源的稳定性和耗时比较敏感，因此可以拉通专线，与百度内网vpc环境相连，并采用本文描述的专线回源IDC方案。客户大致架构如下：



实践方案

概述

BOS具备公网回源的能力，如果客户配置了公网回源地址，数据流会从公网传输。为了将数据流导向客户的专线，需要进行一些配置。整体数据流如图所示：

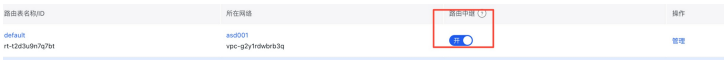
步骤

Step1：创建VPC

用户需要再指定region创建VPC



创建vpc后，需要在已创建的VPC种开启"路由中继"功能，这样才能通过该VPC转发IDC网络的流量



Step2：创建子网

Step3：创建EIP

加白后创建定制线路的EIP



Step4：创建物理专线

通过企业认证后，可以申请创建物理专线，申请后会有专门的工作人员拉通并调试物理专线

Step5：创建专线通道

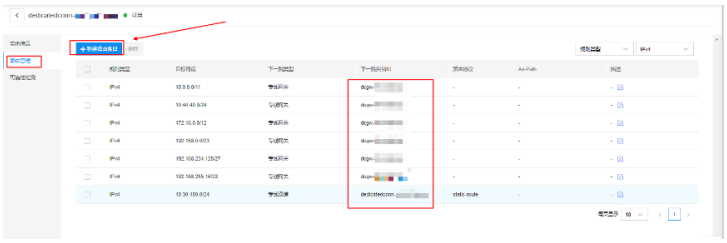
1. 在专线ET界面，用户点击左侧导航栏，选择专线通道，点击右侧"+新建通道列表"按钮，进行创建新增专线通道操作



2. 新增专线通道输入框

> • 目前专线通道VLAN支持范围为0和2-4009；VLAN 0为三层模式，该物理专线下仅能创建一个专线通道，VLAN 2-4009支持Trunk模式，支持一条物理专线下创建多个专线通道
> • 云端网络互联IP：指百度智能云侧互联IP（同一个网段任意两个IP）
> • IDC互联IP：指用户侧互联IP
> • 云端网络及IDC互联IP地址可以是10.0.0.0/8、192.168.0.0/16、172.16.0.0/12三个网段内任意一个没有被使用的IP地址。但不能与云智能网中要互通的地址冲突，也不能和百度智能云云端、客户IDC侧IP地址冲突。互联地址建议掩码为30
> • 专线通道支持静态路由和动态路由，动态路由为BGP协议。百度智能云BGP ASN为45085，百度智能云控制台需要输入用户IDC侧的BGP ASN，支持输入范围为1 - 4294967295

3. 专线通道需添加路由。请用户点击专线通道ID，进入左侧导航栏，选择"路由管理"，并点击"+新建路由条目"按钮进行创建



Step6：创建专线网关

点击“私有网络VPC”，选择网络连接“专线网关”

> • 出口带宽：在专线网关处对绑定的专线通道进行限速，支持2M—100G（根据用户购买的物理端口而定），限速范围不得超过单条物理专线对接的物理端口，否则不生效。

Step7：添加路由

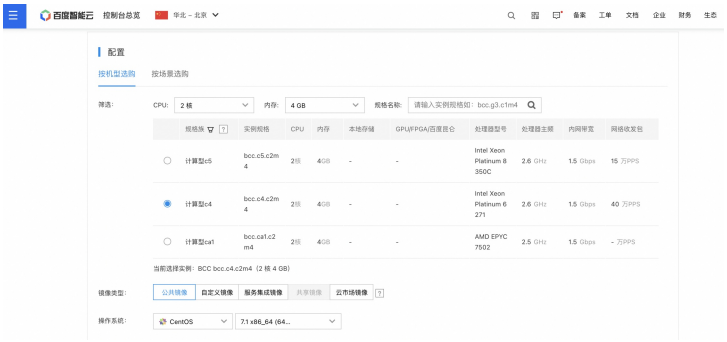
1. 配置VPC路由表。专线网关创建成功并绑定指定专线后，用户需要在所在 VPC 的路由表和 IDC 网络中配置到对端网络的路由完成流量互通。进入VPC的详情页，导航栏选择“路由表”，在路由表列表中点击“添加路由”

源网段选择“自定义配置”，然后网段填写0.0.0.0/0。目的网段填写 IDC 区域网段，可以分多次添加。路由类型，选择“专线网关”。下一跳实例，选择已创建的专线网关。多于一条物理专线时，请依次选择路径 1 和路径 2。

2. 配置专线通道路由。进入专线接入 ET控制台，在左侧导航栏中选择“专线通道”，点击专线通道 ID进入专线通道的详情页面，在路由管理页面，点击“新建路由条目”。目标网段填写VPC的网段，下一跳实例选择已创建的专线网关

Step8：申请BCC实例做反向代理

1. 根据业务场景，申请一台或多台BCC。推荐配置：2 核 CPU、4G 内存、CentOS 7.1



2. 绑定EIP。将申请到的BCC绑定之前创建的EIP

3. 安装Nginx。由于我们需要额外的 stream 模块来提供 TCP 协议代理，所以还需要安装 stream 模块

```
 // 安装
yum install nginx nginx-mod-stream

// 配置nginx
cd /nginx/
vim nginx.conf

// 修改nginx配置，可以参考如下配置
server {
    listen       80;
    listen       [::]:80;
    # server_name  _;
    # root         /usr/share/nginx/html;

    # Load configuration files for the default server block.
    include /etc/nginx/default.d/*.conf;

    error_page 404 /404.html;
    location = /404.html {
    }

    error_page 500 502 503 504 /50x.html;
    location = /50x.html {
    }
    location /video {
        root   html;
        index  index.html index.htm;
        proxy_pass  http://10.xxx.xxx.xxx:80;
        proxy_max_temp_file_size 0;
    }
    location /image {
        root   html;
        index  index.html index.htm;
        proxy_pass  http://10.xxx.xxx.xxx:80;
        proxy_max_temp_file_size 0;
    }
}

// 验证nginx配置并启动nginx
nginx -t
sudo systemctl start nginx
```

参考配置文件中，proxy_pass参数需要填写 **Step6** 配置的专线通道目标网段

测试

配置完成后，可以通过ping和wget命令来登陆反向代理BCC测试代理到客户IDC链路的连通性

```
 ping xxx
wget http://xxx/1.jpg
```

然后，您还可以通过BOS SDK或者CLI、BOS 桌面等各种工具进行镜像回源操作，验证是否能正常抓取站点数据。此外，您还可以登录 Nginx 代理节点检查日志来确认Nginx反向代理工作状态是否正常

```
 less /usr/local/nginx/logs/access.log
```

相关产品

云服务器、专线ET、 弹性公网EIP

通过安卓SDK使用HTTPDNS服务

概览

本文介绍如何使用 HTTPDNS 服务访问 BOS。

需求场景

- 基于 BOS 的使用情况来看，目前移动端上传的问题主要集中在如下几个方面：
  （1）DNS 解析失败，请求出错；
  （2）域名劫持，数据在传输中有篡改，给用户带来诸如网络钓鱼、隐私窃取等网络访问风险；
  （3）弱网下，速度慢，上传数据可能有超时。
- 若使用 HTTPDNS 服务访问 BOS，有以下优势：
  （1）安全防劫持，可有效降低由域名劫持引起的成功率下降问题；
  （2）精准调度可提供最优接入点，降低用户访问时延；
  （3）域名解析结果变更时，HTTPDNS服务没有传统DNS服务多级缓存的影响，能够更快的令移动端获取新的解析结果，避免多层缓存的影响，可有效缩短域名切换的生效时间。

因此，**如果您对 BOS 的请求成功率、延迟、故障止损效速度较为敏感**，您可以使用 HTTPDNS 服务访问 BOS。

方案概述

使用 HTTPDNS 服务访问 BOS，您需要开通 HTTPDNS 服务，并添加项目依赖，重载 OkHttpDns，并调整 Bosclient 配置。

> **注意事项：**
>
> - HTTPDNS 虽然有一定免费额度（目前每自然月有 300 万次免费域名解析次数），超过初始额度后会产生一定的费用，通过 d 的参数配置，可以减少对httpdns server的访问，例如：缓存过期处理策略、使用https协议或http协议等。
> - APP刚启动时，默认HTTPDNS降级标志为是。SDK、网络库自动使用DNS解析的方式发起请求。这属于正常情况，不会影响后续使用HTTPDNS服务来访问BOS。

实践操作

1.开通HTTPDNS服务

- 进入 HTTPDNS 官方文档，完成服务的开通；
- 在【配置管理】的域名管理中添加*.bcebos.com，或者业务使用的自定义域名或者 CDN 域名；
- 测试 HTTPDNS。

2.添加项目依赖

我们提供了方便的远程依赖便于用户使用 httpdns。

- 打开需要引用该 sdk 的 module 的 build.gradle，在最后增加：

```
repositories{
  jcenter()
  maven {
     url "https://raw.githubusercontent.com/bdhttpdns/BDHttpDnsSDK/master"
  }
}
```

- 在dependencies下增加：

```
dependencies{
  compile 'com.baidu:httpdns:1.3'
}
```

3.重载OkHttpDns

设置刚刚开通 HTTPDNS 服务的账号和密码，对常用域名进行预加载，避免初次请求时缓存不命中。同时，还可以设置多种处理策略，以更好地完成业务需要，参考移动域名解析 Android_SDK 设置接口。

```
static class OkHttpDns implements Dns {
  private static OkHttpDns instance = null;
  private BDHttpDns httpDns;

  private OkHttpDns(Context context) {
     this.httpDns = BDHttpDns.getService(context);
     this.httpDns.setAccountID(account);
     this.httpDns.setSecret(secret);
     // 预加载域名
     ArrayList<String> preResolveHosts = new ArrayList<String>();
     preResolveHosts.add("bj.bcebos.com");
     this.httpDns.setPreResolveHosts(preResolveHosts);
  }

  public static OkHttpDns getInstance(Context context) {
     if(instance == null) {
        instance = new OkHttpDns(context);
     }
     return instance;
  }

  @Override
  public List<InetAddress> lookup(final String hostname) throws UnknownHostException {
     // 通过同步解析接口获取ip
     final BDHttpDnsResult httpDnsResult = httpDns.syncResolve(hostname, false);
     Log.v("dns", "HttpDns resolve type: " + httpDnsResult.getResolveType());
     // 优先使用ipv6结果，对于ipv6结果，需要在ip前后增加[]字符
     // 若不存在ipv6结果，则使用ipv4结果
     ArrayList<String> ipv6List = httpDnsResult.getIpv6List();
     ArrayList<String> ipv4List = httpDnsResult.getIpv4List();
     String ip = null;
     if (ipv6List != null && !ipv6List.isEmpty()) {
        ip = "[" + ipv6List.get(0) + "]";
     } else if (ipv4List != null && !ipv4List.isEmpty()) {
        ip = ipv4List.get(0);
     } else {
        Log.v("DNS", "Get empty iplist from httpdns, use origin url");
     }
     if(ip != null) {
        final String finalIp = ip;
        //如果ip不为null，直接使用该ip进行网络请求
        List<InetAddress> inetAddresses = Arrays.asList(InetAddress.getAllByName(ip));
        Log.d("DNS", "inetAddresses:" + inetAddresses);
        return inetAddresses;
     }
     //如果返回null，走系统DNS服务解析域名
     return Dns.SYSTEM.lookup(hostname);
  }
}
```

4.调整Bosclient配置

使用 setDns。

```
 BosClientConfiguration config = new BosClientConfiguration();
config.setCredentials(new DefaultBceCredentials(accessKeyId, secretAccessKey));
config.setEndpoint(bos_endpoint);
config.setProtocol(Protocol.HTTP);
config.setDns(OkHttpDns.getInstance(getApplicationContext()));
```

5.立即体验

以上流程完成后，正常使用BOS就可以体验HTTPDNS域名解析服务了。

相关产品

HTTPDNS服务

将 BOS 作为 Jfrog Artifactory 的后端存储服务

概览

该文档介绍如何将 BOS 配置为 Jfrog Artifactory 的后端存储服务。

应用场景

JFrog Artifactory 是一款通用的二进制制品仓库管理工具，属于 DevOps 平台的组件，用于集中管理软件开发生命周期中的所有二进制文件。其主要有以下应用场景：

- CI/CD 流水线集成
- 安全与合规
- 依赖管理与代理

JFrog Artifactory 支持将对象存储配置为其后端存储服务，带来扩展性和成本优化、高可用、简化运维等价值。

注：Jfrog Artifactory 仅商业版支持将对象存储配置为后端存储服务，开源版本不支持对接对象存储服务。

配置方式

概述

JFrog Artifactory 默认是本地存储，可以通过修改配置文件，将存储方式配置为使用 S3 兼容存储的方式。

步骤

Step1：备份默认的初始配置

```
cp /opt/jfrog/artifactory/var/etc/artifactory/binarystore.xml /opt/jfrog/artifactory/var/etc/artifactory/binarystore.xml.backup
```

Step2：创建新配置文件

```
vi /opt/jfrog/artifactory/var/etc/artifactory/binarystore.xml
```

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<config version="2">
    <chain template="s3-storage-v3-direct"/>
    <provider type="s3-storage-v3" id="s3-storage-v3">
        <bucketName>xxxxx</bucketName>
        <path>artifactory/filestore</path>
        <endpoint>s3.bj.bcebos.com</endpoint>
        <credential>xxxxxxxxxxxxx</credential>
        <identity>xxxxxxxxx</identity>
        <region>bj</region>
    </provider>
</config>
```

参数	内容示例	说明
bucketName	my-bucket-001	已经创建好的 BOS Bucket
endpoint	s3.bj.bce.bos	S3 兼容域名
credential	akxxxxxxxxx	具备 BOS 读写权限的 Access Key
identity	skxxxxxxxxx	具备 BOS 读写权限的 Secret Key
region	bj	Bucket 所属 Region

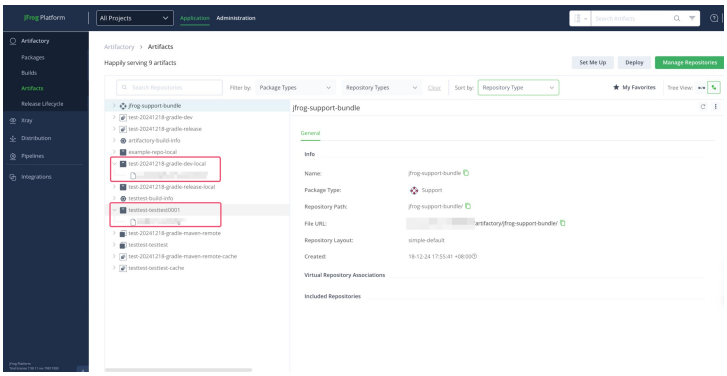
Step3：启动服务并检查日志

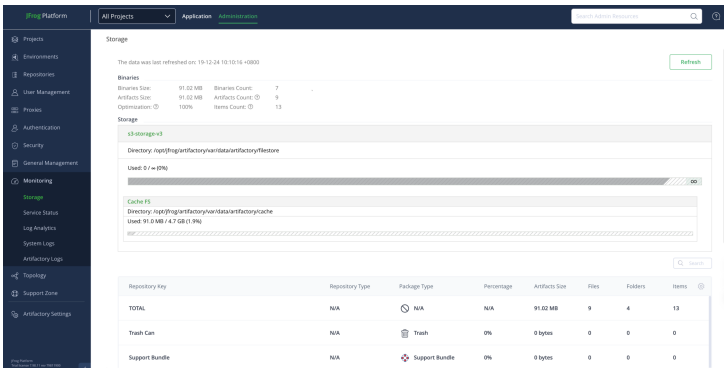
```
sudo systemctl start artifactory.service
sudo systemctl start xray.service
```

```
tail -f /opt/jfrog/artifactory/var/log/artifactory-service.log
```

Step4：创建仓库并选择存储位置

以控制台为例：





常见问题

常见问题总览

一般类问题

- 什么是BOS？
- BOS可以用来做什么？
- 如何选择合适的存储类型？
- 如何使用低频及归档存储？
- 如何开始使用BOS？
- 如何上传数据到BOS？
- 外网流出流量,能否看到具体是哪个文件产生的流量？
- BOS对应的AKSK如何确定？
- 对象存储BOS与普通百度网盘有什么区别？

计费类问题

- 欠费后数据会丢失吗？
- 开启镜像回源后的流量扣费标准是什么？
- BOS通过GetObject方法调用获取到的下载文件，会有流量消费吗？
- BOS上重复上传未修改过的文件会消耗请求次数么？
- BOS欠费已充值依然无法打开怎么处理？
- 购买了BOS存储包如何使用？
- 欠费后BOS资源找不到了怎么办？

性能类问题

- 用户可以在BOS上存储多少数据？
- 用户可以存储哪种类型的数据？
- BOS的日志多久生成?
- BOS上传和下载有限速么？
- Bucket提供的各项监控数据如写入量，读取量等采集时间间隔是多少?

安全类问题

- 用户数据的安全性如何？
- 数据在BOS上是否会丢失或损坏？
- DDoS攻击对BOS有影响么？
- BOS上文件的上传下载是基于http协议传输的吗？
- 如果上传到BOS的文件是有病毒的怎么办？

认证及权限管理

- Bucket ACL和STS有什么区别？
- GetObject权限和READ权限的区别？
- Object权限和Bucket权限不一致时以哪个为准？
- BOS中如何设置账户拥有创建和读写权限，但不能删除？

Bucket访问及管理

- BOS如何查看Bucket域名地址？
- BCC访问其他地域的BOS Bucket也是走内网吗？
- BOS支持在Bucekt下创建文件夹么？
- BOS支持跨域访问吗？

- BOS支持防盗链吗？
- BOS如何绑定https证书？
- Bucket已经配置了跨域请求,为什么还是无法进行跨域的资源请求？
- 如何删除刚新建的Bucket？
- BOS开启CDN如何操作？
- 什么是BOS的内网流量，通过BAE访问BOS的流量算不算内网流量?
- BOS系统的域名被微信小程序后台屏蔽无法使用怎么办？
- 如何监控BOS的流出流量？
- 如何获取BOS的访问日志？
- 绑定域名后如何访问Bucket中的数据？
- 如何查看哪些网站引用了BOS的链接？
- 外网流出流量中能否看到具体是哪个文件产生的流量？
- BOS是否可以在线解压？
- 如何让CDN访问BOS中的其中一个私有BUCKET？

Object上传下载及管理

- BOS支持自定义文件属性吗？
- 如何更新Meta信息，例如Content-type?
- 如何进行文件分享？
- PutObject、PostObject、AppendObject和MultipartUpload这几种上传接口的区别是什么？
- CopyObject是否支持跨region拷贝?
- 如何跨bucket拷贝数据?
- 如何进行大文件上传？
- 如何批量删除BOS中的文件？
- 如何快速查看Bucket中是否存在某个文件？
- BOS是否支持断点续传？
- 如何将标准BOS修改为低频BOS？
- BOS如何上传下载文件或者文件夹？
- 如何设置文件的Header属性？
- 是否支持浏览器上传文件？
- BOS误删除的数据是否可以找回？
- 如何设置才能直接浏览Object而不是下载？
- BOS文件上传后能否自动进行MD5校验？
- BOS如何获取缩略图？
- 如何在ListObjects接口同时获取Object的User和Meta信息？
- BOS的Object获取地址能否用自定义域名？

API&SDK&工具

- BOS的错误响应如何定位？
- 在使用API或SDK上传Object时，如何判断一个Object是否上传成功？
- PutObject、PostObject、AppendObject和MultipartUpload这几种上传接口的区别是什么？
- 如何跨bucket拷贝数据?
- 如何验证一个Object/Bucket是否存在？
- BOS如何挂到linux服务器上？
- BOS如何集成到APP中？
- 运行PHP SDK的Sample Code时，对phpunit版本有什么要求？
- 如何使用BOS搭建FTP服务？
- UCenter和Python flask怎么对接BOS？

数据迁移

- 如何将云磁盘CDS里面的内容转存到BOS上？
- 如何将其它云平台的数据同步到BOS中？
- 如何将BOS中的资源迁移到其他Bucket？

图片服务

- BOS开启原图保护后，访问图片链接报“Access Denied”错误。

- BOS中的图片添加水印后，水印内容是否能自动适应图片？

一般类问题

什么是BOS？

百度对象存储BOS（Baidu Object Storage）是一种面向 Internet 的存储服务，提供稳定、安全、高效以及高扩展存储服务，支持单文件最大48.8TB的文本、多媒体、二进制等任何类型的数据存储。数据多地域跨集群的存储，以实现资源统一利用，降低使用难度，提高工作效率。

BOS可以用来做什么？

BOS提供一个简单的 Web 服务接口，可用于存储和提取任意数量的数据。开发人员可以轻松地构建利用互联网存储的应用程序。BOS具有很高的可扩展性，用户只需按实际用量付费，因此开发人员可以从较小用量起步，根据需要扩展应用程序，而不影响性能或可靠性。如灾备系统、数据分析存储、内容存储分享、网盘等应用。

如何选择合适的存储类型？

BOS目前支持标准存储、低频存储、冷存储和归档存储四种。四种存储类型的应用场景及产品定价信息请参见BOS分级存储介绍。

如何使用低频及归档存储？

百度智能云支持通过Console控制台、BOS CLI和BOS API三种方式使用低频存储。

- 通过Console使用低频存储即在上传Object的时候指定存储类型，详细操作请参考上传Object。
- 通过BOS CLI使用低频存储即在执行`bce bos cp`命令时指定存储类型参数storage-class，详细操作请参考上传Object。
- 通过BOS API使用低频存储即在请求或返回中增加storage-class属性，详细描述请参考低频存储和标准存储API接口的差异。

如何开始使用BOS？

请参考入门指南。

如何上传数据到BOS？

上传数据到BOS有4种方式：

- 控制台上传：上传少量文件，而且文件较小。控制台支持单文件最大5G，最多300个文件上传。
- SDK上传：使用手机App，Web应用来上传文件。
- 工具上传：有大量文件需要上传，需要使用到断点续传，分片上传来加速上传。
- 数据导入服务：用户有TB级别的数据需要上传；普通的方式需要很长时间（如：10TB数据，上行链路为20Mb/s,约需要20天）；而且需要保证数据的完整性，安全性等。

外网流出流量,能否看到具体是哪个文件产生的流量？

目前有两种方式可以确认：

1. 通过设置并查看日志进行确认，设置详细操作请查看设置日志
2. 查看统计分析-热点分析，指定对应的Bucket确认即可。

BOS对应的AKSK如何确定？

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。

对象存储BOS与普通百度网盘有什么区别？

BOS并非百度网盘，这个是两个产品，互不相通。无法互相调用交叉使用，如果您遇到了百度网盘的问题，建议您去往百度网盘右上角问题反馈中反馈，会有相关百度网盘人员处理。如果您在使用BOS的过程中遇到了任何问题，建议提交工单处理。

使用BOS必须高级实名认证吗？

由于工信部要求，BOS类产品都需要做企业高级实名认证后才可进行使用。

BOS支持存储的文件类型或者限制说明?

对象存储BOS服务，支持存储文本文件、图片、视频等任意格式类型的数据文件。

SignatureDoesNotMatch，常见错误分析?

由于鉴权有误导致，您可以参考官网提供的API鉴权教程和示例文档和视频进行调试，看看是否正常。

文档：https://cloud.baidu.com/doc/Reference/s/njwvz1yfu中提供了 在线签名工具生成和Postman脚本直接生成，可以参考使用

视频：https://cloud.baidu.com/video-center/video.html?id=364

如何设置回收站功能？

可以在管理控制台中进行选择开启或关闭回收站，进行控制回收站功能是否生效，具体请参考：设置回收站

计费类问题

1. 欠费后数据会丢失吗？

欠费处理：

- 当您的账户余额不足0元无法支付BOS账单，即为欠费，欠费时系统会发送欠费通知。
- 欠费后立即停服，系统会发送欠费停服通知，数据为您保留15天，期间收取存储空间的费用。若15天内仍未充值则清除数据，清除数据的前1天和清除数据时系统都会发送清除数据通知。

2. 开启镜像回源后的流量扣费标准是什么？

镜像回源是一个Bucket级别的设置，目前镜像回源抓取到的文件会默认存到标准存储里，计费项包含存储、请求和流量三部分费用，具体计费介绍请查看产品定价。

3. BOS通过GetObject方法调用获取到的下载文件，会有流量消费吗？

GetObject方法调用获取到的下载文件会收取请求费用 +流量费用 ，如果是低频存储和冷存储还会收取数据取回费用 ，具体费用介绍请查看产品定价。

4. BOS上重复上传未修改过的文件会消耗请求次数么？

每一次的上传都会请求对应的上传接口，会计算请求次数。

5. BOS欠费已充值依然无法打开怎么处理？

BOS欠费充值后一般是10-20分钟左右才可以生效并正常使用，请耐心等待，10分钟后尝试重新刷新浏览器缓存。

6. 代金券无法抵扣BOS费用是什么原因？

由于代金券有区域类型，目前BOS仅支持全局区域类型代金券，其它区域类型代金券无法抵扣BOS费用。代金券类型查看方式：控制台 - 财务 - 代金券



7. 购买了BOS存储包如何使用？

BOS支持包年包月（预付费）形式的存储包购买。购买存储包后，您的BOS消费会首先用存储包进行抵扣（无需特殊设置），存储包规格内的用量免费，超出部分按量付费。您可以直接在管理控制台的"BOS->存储包"页签进行存储包购买，购买时按照提示选择区域、规格、购买时长和生效时间即可。详细说明可以参考如下文档BOS存储包计费。

8. 欠费后BOS资源找不到了怎么办？

当BOS服务处于欠费状态时，将立即停服。BOS将拒绝您的读写请求，但是数据会为您保留15天，期间您数据所占用的存储空间仍会继续扣费，因此欠费金额会累计。若您在欠费后及时进行充值，那么您的服务将恢复正常使用且数据可用。若您持续欠费超过15天，将视为您主动放弃BOS存储服务，存储空间将被回收，存储空间内的数据会被清理，数据不可恢复。详细描述请参考余额不足和欠费提醒。

9. BOS标准存储包是否支持按年购买?

BOS标准存储包是支持按年购买。

10. BOS购买了资源包，为什么还是扣费了？

BOS的计费由四部分存储费用，流量费用，处理费用和请求费用四部分组成，您存储包抵扣的只有存储费用，具体费用说明请查看文档。

11. BOS到期删除后能找回吗？

BOS欠费后立即停服，系统会发送欠费停服通知，数据为您保留15天，期间收取存储空间的费用。若15天内仍未充值则清除数据，清除数据的前1天和清除数据时系统都会发送清除数据通知，已经清除的数据无法找回。

12. BOS资源包购买错区域怎么办？

购买的资源包必须与bucket在同一区域，否则是无法进行抵消费用的，您可以提交财务类型工单，进行退订资源包，重新购买。

13. 如何查看bucket的费用明细情况？

bucket的费用明细可以在消费总览-账单明细中查看产品费用情况：

1. 登录管理控制台，点击"财务"->"消费总览"，进入财务中心。



2. 点击左侧导航栏"消费中心"->"账单明细"，在"流水账单"页面，找到BOS产品，点击右侧"查看"，即可查看产品费用情况。



14. 调用copyobject完成数据复制，费用情况？

copyobject 为同地域复制，将产生接口调用费用，以及目标对象的存储费用。

15. 购买对象存储资源包,使用完后会怎么处理？

目前对象存储BOS服务属于后付费产品，当购买的资源包使用完后，会扣除账号余额，而账号欠费后，会停止使用，欠费超过15天后会自动释放资源，具体请参考：余额不足和欠费提醒。

16. 百度智能云对象存储,外网下行流量包,有没有按年付费的方式？

目前bos外网流量包，暂不支持按年购买，建议您可以购买多个bos外网流量包，每个流量包指定固定时间生效。或者进行续费资源包，其会在自动指定续费的资源包生效时间。

17. 归档存储涉及哪些计费项？

1. 与归档存储类型直接相关的计费项：

> 注：价格请以对象存储BOS官网详情页价格为准

计费项	官网价	说明	举例	计费方式
归档存储最小存储周期	0.015元/GB/月 × 剩余存储周期（月）× 归档存储文件容量	归档存储类型文件有 180 天最小存储时间要求。若文件在未存满 180 天时被删除/覆盖写，会产生一笔最小存储周期费用，BOS 将一次性收取**剩余未存满时间内的容量费用**。	若您有1GB归档存储类型文件，上传3天后将该归档存储类型文件删除，那么最小存储周期的费用为：归档存储容量（GB）× 归档存储月单价 × 剩余最小存储周期（月）= 1×0.015×（180-3）/30	按需付费
归档存储文件取回缓存	0.003967元/GB/日	当您需要对归档存储类型文件进行读操作时，需要先将该文件取回，BOS会为您创建副本以便您短期内随时读取。触发取回操作时，您需要设置文件取回副本的缓存时间，该副本文件将会产生相应缓存费用。	若您有1GB归档存储类型文件，设置了取回后副本的缓存时间为10天，那么该文件将一次性收取 10 天的缓存费用，则取回缓存费用为：归档存储容量（GB）× 缓存时间（天）× 0.003967 = 1×10×0.003967	按需付费
归档存储容量	**1. 若小于64KB：**0.015元/GB/月 × 64/1024/1024GB × 存储时间（月） **2. 若大于等于64KB：**0.015元/GB/月 × 归档存储文件容量（GB）× 存储时间（月）	若归档存储类型文件不足 64 KB，则按照 64 KB 大小进行计费；若文件大于等于 64 KB，则按照实际大小进行计费。	若您的归档存储文件容量为1GB，存储了12个月，则存储费用为：归档存储文件容量（GB）× 0.015 ×存储时间（月）= 1×0.015×12	按需付费、资源包计费（资源包价格见官网）
归档存储读请求API	0.5元/万次	所有对 BOS 的操作实际上都是对 BOS API 的调用。当您**读取 BOS 中数据时会调用 BOS 读相关的 API 如 GET 等**，将产生读请求费用。BOS 控制台操作也是通过 BOS API 调用实现的，因此使用 BOS 控制台也会产生读请求次数。系统会每小时结算一次读请求费用。	若您对归档存储类型文件进行了1万次的读操作，那么归档存储读请求的费用为：请求次数（万）× 0.5 = 1×0.5	按需付费
归档存储写请求API	0.5元/万次	所有对 BOS 的操作实际上都是对 BOS API 的调用。当您**向 BOS 中写入数据（上传、覆盖）时会调用 BOS 写相关的 API 如 PUT 等**，将产生写请求费用。BOS 控制台操作也是通过 BOS API 调用实现的，因此使用 BOS 控制台也会产生写请求次数。系统会每小时结算一次写请求费用。	若您上传归档存储类型文件时，进行了1万次的写操作，那么归档存储写请求的费用为：写请求次数（万）× 0.5 = 1×0.5	按需付费
归档存储加急取回	0.35元/GB	归档存储文件取回需要您通过 API/SDK/BOS官方工具/控制台手动触发，取回成功后才进行访问或者下载。归档存储取回有三种模式，分别为加急取回、标准取回和延缓取回，每种取回模式费用不同**（加急取回：30分钟内取回）**。	若您有1GB的归档存储文件需要取回，您选择了加急取回，则取回费用为：归档存储容量（GB）× 0.35 = 1×0.35	按需付费
归档存储加急取回次数	30元/万次	在您取回归档存储文件时，会调用取回API。根据取回模式收费。	若您取回归档存储类型文件时，进行了1万次的取回请求，则调用取回请求的费用为：取回API调用次数（万）× 30 = 1×30	按需付费
归档存储标准取回	0.12元/GB	归档存储文件取回需要您通过 API/SDK/BOS官方工具/控制台手动触发，取回成功后才进行访问或者下载。归档存储取回有三种模式，分别为加急取回、标准取回和延缓取回，每种取回模式费用不同**（标准取回：5小时内取回）**。	若您有1GB归档存储类型文件，选择标准取回，则取回费用为：归档存储容量（GB）× 0.12 = 1×0.12	按需付费
归档存储标准取回次数	3元/万次	在您取回归档存储文件时，会调用取回API。根据取回模式收费。	若您取回归档存储类型文件时，进行了1万次的取回请求，则调用取回请求的费用为：取回API调用次数（万）× 3 = 1×3	按需付费
归档存储延缓取回	0.06元/GB	归档存储文件取回需要您通过 API/SDK/BOS官方工具/控制台手动触发，取回成功后才进行访问或者下载。归档存储取回有三种模式，分别为加急取回、标准取回和延缓取回，每种取回模式费用不同**（延缓取回：12小时内取回）**。	若您有1GB的归档存储文件需要取回，您选择了延缓取回，则取回费用为：归档存储容量（GB）× 0.06 = 1×0.06	按需付费
归档存储延缓取回次数	0.3元/万次	在您取回归档存储文件时，会调用取回API。根据取回模式收费。	若您取回归档存储类型文件时，进行了1万次的取回请求，则调用取回请求的费用为：取回API调用次数（万）× 0.3 = 1×0.3	按需付费

2. 与存储类型无关，但与使用功能有关的计费项：

> 说明：
>
> - 根据客户不同的使用情况，所涉及的计费不同
> - 内容审核具体计费项详见 ICR 定价文档

计费项	说明	特点
外网流出流量、CDN 回源流出流量、跨区域复制流量、CDN 上行加速	同时满足以下两点，则会产生对应费用： 1. 使用功能 2. 归档存储文件处于取回完成态（有直接读取的缓存文件）时	不区分存储类型，若使用对应的功能则会产生对应的用量及费用，详见对象存储BOS计价
Select 扫描、视频处理、图片处理、文档预览、内容审核	同时满足以下两点，则会产生对应费用： 1. 使用对应的功能 2. 归档存储文件处于取回完成态（有直接读取的缓存文件）时	
清单费用	使用对应的功能，会产生对应费用	

18. BOS计费项名称优化说明

- 为了增强BOS计费项中文名称的易读性、一致性，消除由于名称不准确而引发的歧义，于 **2023年5月26日** 对BOS部分 **计费项中文名称** 进行修改优化。具体变更如下：

分类	（原）计费项中文名称	（现）计费项中文名称
流量	读取流量	外网流出流量
	CDN回源请求产生的流出流量	CDN回源流出流量
	跨区域流出流量	跨区域复制流量
读写次数	标准-读取次数	标准存储-读取API调用次数
	标准-写入次数	标准存储-写入API调用次数
	低频-读取次数	低频存储-读取API调用次数
	低频-写入次数	低频存储-写入API调用次数
	冷存储-读取次数	冷存储-读取API调用次数
	冷存储-写入次数	冷存储-写入API调用次数
	归档存储读请求API	归档存储-读取API调用次数
	归档存储写请求API	归档存储-写入API调用次数
	标准多AZ-读取次数	标准存储多AZ-读取API调用次数
	标准多AZ-写入次数	标准存储多AZ-写入API调用次数
	低频多AZ-读取次数	低频存储多AZ-读取API调用次数
	低频多AZ-写入次数	低频存储多AZ-写入API调用次数
	归档存储加急取回次数	归档存储-加急取回API调用次数
	归档存储标准取回次数	归档存储-标准取回API调用次数
	归档存储延缓取回次数	归档存储-延缓取回API调用次数
容量	标准-占用空间	标准存储-容量费用
	低频-占用空间	低频存储-容量费用
	冷存储-占用空间	冷存储-容量费用
	归档存储容量	归档存储-容量费用
	标准多AZ-占用空间	标准存储多AZ-容量费用
	低频多AZ-占用空间	低频存储多AZ-容量费用
	标准存储（专属）-占用空间	标准存储（专属）-容量费用
	低频存储（专属）-占用空间	低频存储（专属）-容量费用
	冷存储（专属）-占用空间	冷存储（专属）-容量费用
	归档存储（专属）-占用空间	归档存储（专属）-容量费用
	低频-最小存储周期	低频存储-最小存储周期
	归档存储最小存储周期	归档存储-最小存储周期
	低频多AZ-最小存储周期	低频存储多AZ-最小存储周期
数据取回	低频-数据取回	低频存储-数据取回
	归档存储加急取回	归档存储-加急数据取回
	归档存储标准取回	归档存储-标准数据取回
	归档存储延缓取回	归档存储-延缓数据取回
	归档存储文件取回缓存	归档存储-数据取回缓存
	低频多AZ-数据取回	低频存储多AZ-数据取回

性能类问题

用户可以在BOS上存储多少数据？

用户在BOS上存储的数据总量和Object数量不受限制，且上传单文件最大 48.8 TB（50000 GB）的Object。

用户可以存储哪种类型的数据？

用户在BOS上存储任何格式的任何类型数据。

BOS的日志多久生成?

大多数情况下，一个访问请求从发生到对应日志存储到目标Bucket会在2-3个小时内完成。但受到网络环境等因素影响，有的日志可能会有延迟。

BOS上传和下载有限速么？

BOS对单Bucket设置的带宽阈值为10Gbit/s，对并发没有限制，不针对人数，只针对带宽阈值有限制。当用户的上传或下载行为达到带宽阈值时，系统会给予RequestRateLimitExceeded错误的告警，此限制目前对BOS所有Region都生效。如果您上传慢，一般是由于您本地上传带宽导致，建议您检查本地上传带宽。具体限制请参考使用限制。

如果有特殊的业务需求，希望在短时间内上调带宽阈值，建议提交工单处理。

Bucket提供的各项监控数据如写入量，读取量等采集时间间隔是多少?

Bucket提供的各项监控数据，写入量，读取量等采集时间间隔是5分钟。

安全类问题

如何对BOS资源状况进行定期的全面检查？

您可开通“云顾问 Cloud Advisor”服务，定期获得云上资源在安全、可用性、性能、成本方面的检查报告。报告中包含若干BOS相关检查项目，如：BOS-Bucket开启回收站、BOS-Bucket日志记录、BOS-Bucket权限、BOS-服务端加密等。了解或开通云顾问服务，请前往 云顾问主页。

用户数据的安全性如何？

存储在BOS中的数据非常安全。BOS提供多种访问控制机制来保证用户的数据的安全性，如签名验证机制、访问控制列表等。

数据在BOS上是否会丢失或损坏？

BOS提供多重备份，和数据自动修复的能力，提供了99.99999999%的数据可靠性和99.9%的服务可用性。

DDoS攻击对BOS有影响么？

DDoS流量攻击一般针对是固定IP，BOS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DDoS攻击。

BOS上文件的上传下载是基于http协议传输的吗？

BOS文件上传下载支持http和https两种方式。

如果上传到BOS的文件是有病毒的怎么办？

BOS对上传的文件是没有限制的。但是上传后BOS会进行抽查，如果发现客户Bucket中的文件有病毒就会对该客户的Bucket进行封禁。另外BOS也提供图像智能审核服务，基于深度学习的智能内容审核方案，准确识别图片和视频中的涉黄、涉暴涉恐、政治敏感、微商广告、恶心等内容，详细参考图片审核服务。

存放在bos里面的视频链接,会被别的网站使用吗，如何防护？

您可以通过设置bucket权限，以及定义referer白名单方式进行加强资源防护，具体请参考：bucket权限 ， referer白名单 。

BOS流量如何设置流量报警功能?

bos本身不支持监控报警，但是可以通过云监控服务，监控对象存储的容量，流量，请求等指标，并设置报警规则，请登录控制台，设置报警规则。

认证及权限管理

Bucket ACL和STS有什么区别？

BOS的权限可以分成两类：基于资源的权限和基于用户的权限。

- 基于资源的权限是指控制某个资源可以被哪些人访问，Bucket ACL就是基于资源的权限。
- 基于用户的权限是指管理某个用户可以访问哪些资源，STS临时授权就是基于用户的权限的一种。

Bucket ACL和STS的权限控制都是基于ACL文件实现。语法很接近，都支持grantee（权限授予的主体，STS由于是基于用户的权限所以不需要该字段）、permission（操作）、resource（资源）、condition（条件）等字段。

从使用场景上来说，Bucket ACL比较适合grantee是某个百度智能云账号或者全部用户（含匿名用户）的场景；STS比较适合临时授权的场景，比方说某个手机需要临时上传一张图片。

GetObject权限和READ权限的区别？

GetObject权限仅包括GetObject和GetObjectMeta两个API操作；而READ权限除了GetObject权限所含的API操作外，还包含GetBucketLocation、HeadBucket、ListParts这三个API操作。更多细节请见Bucket权限控制。

Object权限和Bucket权限不一致时以哪个为准？

Object权限设置中，Object权限默认与所属Bucket的权限一致，用户也可按需将Object设置为公共读或私有权限。当Object读操作权限与所属Bucket权限不一致时，以Object权限为准。

BOS中如何设置账户拥有创建和读写权限，但不能删除？

可以通过上传ACL文件设置Bucket的细粒度权限来实现，详细信息参考上传acl文件方式的权限控制。

存储桶域名请求风格

BOS 存储桶域名请求风格更新

为保证服务的安全和稳定，2024年6月1日起，百度智能云对象存储 BOS (Baidu Object Storage) 对于官方域名的请求风格将作出变更。新创建的存储桶，不支持使用 path-style 的请求风格，默认支持 virtual-hosted 的请求风格。

2024年6月1日之前所创建的存储桶，使用官方域名时，支持继续使用 path-style 的请求风格。对于存量的使用 path-style 请求风格的存储桶，建议可切换至 virtual-hosted 的请求风格。切换过程如有疑问，可参考开发者指南。如需我们的帮助，请与我们联系。

Bucket访问及管理

BOS如何查看Bucket域名地址？

登录BOS控制台，进入Bucket对应的概览页，可以看到官网域名和自定义域名及CDN加速情况。还可以点击“域名发布信息”的“管理”进入域名编辑模块，添加自定义域名并进行CDN加速及发布缓存的相关配置。



BCC访问其他地域的BOS Bucket也是走内网吗？

BCC如果访问同一地域的BOS Bucket，会走内网；如果BCC访问的是其他地域的BOS Bucket（如：北京BCC访问广州Bucket），走的是公网，此时需要BCC绑定公网IP才能进行访问。

BOS支持在Bucket下创建文件夹么？

BOS本质上是一个以Bucket为存储空间的<key, value>系统。因此管理控制台console上所展示的文件夹其实是一种“模拟文件夹”。可以通过两种方式创建“模拟文件夹”：

- 创建一个以“/”结尾的文件，如名为“fakeFolder/”的文件；
- 在文件名中带上前缀，如名为“fakeFolder/img.jpg”的文件，具有相同前缀的文件会表现为在同一个模拟文件夹下。

BOS支持通过一下两种方式创建文件夹功能：

- 通过控制台操作，请参考管理文件夹。
- 通过代码操作，请参考SDK文档Java。

BOS支持跨域访问吗？

BOS支持设置跨域访问(CORS)，API接口参见跨域资源共享。如何在控制台上设置CORS见设置跨域访问。

BOS支持防盗链吗？

BOS支持通过添加自定义权限来实现防盗链。Bucket自定义权限设置包括：

- 允许referer为空，
- 不允许referer为空；
- 设置匹配的referer字符串。
- 具体设置方法请参考设置Referer白名单。

BOS如何绑定https证书？

BOS支持自定义域名开启https。（香港，香港二区不支持），设置方式： 登录“控制台-BOS对象存储-Bucket名称”，点击“发布管理-自定义域名编辑”，选择开启HTTPS配置即可。如果是香港和香港二区想要开启自定义域名https，需要利用CDN来实现，具体实现步骤是先配置bos自定义域名开启cdn加速服务，然后在CDN上配置HTTPS之后访问即可。

Bucket已经配置了跨域请求,为什么还是无法进行跨域的资源请求？

配置浏览器上的Bucket跨域(CORS)可参考CORS跨域典型实践。如配制后不生效，一般是浏览器缓存问题，建议清空浏览器缓存后再测试。

如何删除刚新建的Bucket？

对象存储Bucket创建后无法删除，但只要不向BOS中上传文件或者浏览所上传的文件，是不会产生计费的。

BOS开启CDN如何操作？

BOS目前支持使用CDN加速指定的Bucket。选择需要加速的Bukcet后，将对应Bucket的域名的DNS解析修改为CDN提供的CNAME域名，从而接入CDN节点开启CDN加速。详细操作请参考CDN加速发布。

什么是BOS的内网流量，通过BAE访问BOS的流量算不算内网流量?

同一账号同一区域的BCC，BAE与BOS默认内网互通即走内网流量，BCH暂时不支持与BOS内网互通。不同地域内网不通，需要走公网传输。

内网传输数据时，不收取流量费用，但是收取存储费用和请求费用。如果购买了存储包，存储包抵用的只是存储费用。

如果通过外网进行访问，会收取请求费用以及流量费用，如果同时开启了CDN功能，则会收取CDN费用。

BOS系统的域名被微信小程序后台屏蔽无法使用怎么办？

微信侧对于外部链接地址有访问限制，直接使用默认分配的域名在微信是无法访问的，建议绑定自定义域名后，使用自定义域名去获取资源，目前一个域名仅支持绑定一个Bucket。

如何监控BOS的流出流量？

流出流量包含外网读取量、回源流量和跨区域复制流量三部分。BOS控制台支持查看读取量，写入量，读取次数，写入次数，CDN回源流量等监控信息，监控项的具体介绍请参考统计与监控。

如何获取BOS的访问日志？

需要获取BOS的访问日志时需要先开启Bucket的访问日志功能，详细操作方法请参见设置访问日志。日志含义和格式详细描述可参考访问日志。

一个访问请求从发生到对应日志存储到目标Bucket约在2-3个小时内完成，但受网络环境等因素影响，有的日志可能会延迟较长时间才能存储到目标Bucket，请耐心等待。

绑定域名后如何访问Bucket中的数据？

BOS绑定自定义域名可以实现直接用浏览器URL访问您的Object，绑定的自定义域名需要完成备案才可以正常访问的，绑定一般3分钟生效。绑定域名的详细操作参见绑定Bucket域名。

如何查看哪些网站引用了BOS的链接？

BOS暂不支持查看具体的链接引用信息。如果不希望其他人引用BOS链接，可通过BOS控制台设置Referer字段的白名单。设置白名单后，只有Referer字段在白名单内的用户才可访问Bucket中存储的数据，不在白名单内的请求会被拒绝。但若Referer为空时，默认可访问，不受白名单限制。详细配置参考防盗链。

外网流出流量中能否看到具体是哪个文件产生的流量？

目前有两种方式可以确认：

1. 通过设置并查看日志进行确认。
2. 查看统计分析-热点分析，指定对应的Bucket确认即可。

详细操作请查看统计与监控。

BOS是否可以在线解压？

BOS目前支持在线解压功能，您需要先在BOS控制台--事件通知中配置好解压规则，上传压缩包匹配到规则就会自动触发解压，详细请参考：文件自动解压。

如何让CDN访问BOS中的其中一个私有BUCKET？

可以通过设置私有Bucket回源来实现，详细的操作请参考回源配置。

BOS私有bucket临时访问文件权限问题？

私有权限文件生成有有效期时间的临时访问url，

1.您可以通过控制台文件信息中获取到

2.您可以通过sdk获取临时url。

参考文档：获取临时url（可自定义了访问地址的过期时间）

BOS内不同的bucket可以绑定同一个自定义域名吗？

同一自有域名仅可以绑定一个bucket，但是一个bucket可以绑定20个不重复的域名。自定义域名更多使用参考：https://cloud.baidu.com/doc/BOS/s/ckaqihkra

BOS支持挂载到windows环境吗？

BOS对象存储支持挂载到您的服务器上，不过仅支持linux系统的服务器，Windows系统暂不支持，具体流程您可参考bosfs这个工具的教程安装bosfs挂载Linux服务器。

微信不能直接访问BOS的文件地址？

BOS的访问地址可能会被微信屏蔽，您可以在百度智能云控制台绑定您自定义的域名，通过您的自有域名链接即可访问，请参考自定义域名：https://cloud.baidu.com/doc/BOS/s/ckaqihkra

对象存储可以映射到我们本地的服务器吗？

目前BOS并不支持webdav和smb，无法在Windows环境挂载。BOS仅针对linux系统提供了bosfs进行挂载映射。在windows系统中操作上传，您可以参考使用，如：bos桌面、bcecmd等，具体请参考：BOS周边工具。

对象存储BOS,前端使用js上传视频之后,能获得这个视频的时长吗,比如一个30分钟的视频？

对象存储BOS中，并不支持获取音视频文件的时长。

BOS android sdk是否支持断点续传？

对象存储中，Android SDK并不支持断点续传，支持断点续传的SDK，如下：Java SDK、Python SDK、PHP SDK、Ruby SDK、C++ SDK。

如何获取bos资源访问地址？

对象存储BOS中，可以通过以下方式获取对应文件访问URL地址：

- 方式1: 通过BOS管理控制台--Bucket列表--文件列表--文件信息--复制链接



- 方式2: 手动拼接URL

a) bucket公共读权限：http://<bucketname>.<region>.bcebos.com/<objectname>

b) bucket私有权限：http://<bucketname>.<region>.bcebos.com/<objectname>?authorization=<authorization>

注：需要替换bucketname、region、objectname、authorization（需要结合getobject接口进行生成鉴权）

- 方式3: 通过SDK进行获取URL，例如：java sdk

请问BOS的API是否支持下载某个目录下所有的文件？

您好,BOS API接口暂是没有下载目录下所有object的功能，需要您先进行listobject接口获取目录下资源文件列表，然后再通过getobject下载文件。

listobject参考文档：ListObjects

getobject参考文档：GetObject

对象存储BOS,Bucket名称被他人占用，能否重复创建？

对象存储BOS的Bucket名称遵循全局唯一规则，被其他用户创建后，无法再重复创建。

能获取存储对象的【访问链接】吗？

对象存储BOS中，可以通过以下方式获取对应文件访问url地址：

方式1: 通过BOS管理控制台--Bucket列表--文件列表--文件信息--复制链接

方式2: 手动拼接url

a) bucket公共读权限：http://<bucketname>.<region>.bcebos.com/<objectname>;

b) bucket私有权限：http://<bucketname>.<region>.bcebos.com/<objectname>?authorization=<authorization>;

注：需要替换bucketname、region、objectname、authorization（需要结合getobject接口进行生成鉴权）

方式3: 通过sdk进行获取url，例如：java sdk

BOS的文件如何批量下载？

对象存储BOS可以通过以下方法进行批量下载：

1. 使用BOS周边工具，如：BOS桌面，Bcecmd等；

2. 使用API/SDK进行ListObject，再循环调用GetObject接口进行下载。

下载BOS内的文件是否如何验证文件完整性？

您可以使用getobject接口或者getobjectmate接口，返回值中有etag参数，然后您在本地计算您文件的MD5值，和bos返回的etag值进行比对，来验证文件的完整性。参考文档：BOS API文档

如何更新文件内容，但保持bos文件地址不变？

对象存储BOS中，可以通过上传相同文件名文件，进行覆盖源站文件，其访问url就可以保持不变。

不计算签名认证信息，是否可以长久的访问对象？

对象存储BOS中，可以设置Bucket权限为公共读后，获取的访问url就不会携带authorization鉴权信息，且可以永久访问。

请问有提供批量上传和下载多个object的SDK么？

对象存储BOS中，并不支持批量上传或者批量下载object接口，可以通过上传或者下载接口，在业务层进行实现。

文件上传成功后,如何获取上传的这个文件大小？

对象存储BOS中，提供了GetObjectMeta接口，可以获取文件大小等信息。

上海区域bucket是否支持https？

对象存储BOS服务中，仅支持北京、广州、苏州区域绑定https证书，而其他区域并不支持。

文件下载/如何生成访问链接？

对象存储BOS提供了下载Object API接口。

上传到BOS中的资源，如何自动到CDN中？

对象存储BOS服务，提供了"缓存管理"功能，其可以把上传的资源，自动预加载到CDN缓存，或者更新CDN缓存数据。

私有文件如何获取访问链接？

对象存储BOS服务，当Bucket设置为私有权限后，可以通过以下方式进行获取访问url：

1. 通过BOS控制台--Bucket文件列表--文件信息--选择访问URL进行复制；
2. 通过BOS SDK，调用接口获取访问url地址，如：Java SDK中获取文件下载url

BOS开启了CDN加速，点击复制的链接为什么不是CDN地址？

BOS内默认都会是BOS官方的链接。如需CDN的链接，可以点击文件信息，然后选择CDN链接。

在BOS管理控制台中如何重命名文件？

可以通过以下两种方式进行处理：

1. 在BOS管理控制台-bucket文件列表-更多-重命名，具体请参考：重命名文件
2. 调用bos api/sdk接口进行重命名，如：通过copyobject接口进行重命名文件，再通过deleteobject删除原始文件即可。请参考：CopyObject接口 DeleteObject接口

BOS bucket如何通过自己的域名访问？

目前bucket是支持绑定自定义域名，可以通过自定义域名进行访问。具体请参考：创建bucket自定义域名。

如何删除bucket回收站所有资源?

您可以在管理控制台中查看回收站所在目录，再通过bos周边工具进行删除对应目录下数据即可，如：bcecmd，具体请参考：删除object 。

BOS修改bucket的属性，从公共读到私有，这个设置是立即生效的吗？

是的,bucket的权限修改后是立即生效的。

bucket配置生存周期后，多久生效?

目前配置bucket生命周期后，正常是在24小时内生效，具体请参考：注意事项 。

sts token最长有效时间是多少?

目前sts生成token的最长有效时间是129600秒（36小时），具体请参考：sts服务接口。

想设置某些用户只访问指定的bucket，其他bucket也看不到，该如何设置？

可以通过配置多用户访问控制，进行控制不同用户的操作查看权限，具体请参考：配置多用户访问控制 。

是否支持针对每个不同用户设置不同的bucket权限？

目前bos支持设置多用户访问控制，可以针对每个不同的用户授予不同的操作权限，具体请参考：配置多用户访问控制 。

子用户创建bucket失败,显示无权限怎么处理？

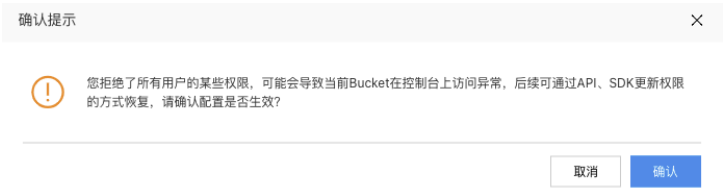
目前子用户针对对象存储Bos服务进行操作，需要授予对应的操作权限，具体权限设置请参考：配置多用户访问控制。

bucket只允许内网访问怎么设置？

对象存储BOS提供了IP白名单功能，可以指定允许的IP地址进行访问。

bucket误设置禁止所有人访问怎么恢复？

对象存储BOS控制台Bucket权限设置可设置禁止所有人访问（设置时有如图提示），设置后会导致控制台无法正常访问该bucket。



修复方式：

1. 使用API方式
   百度智能云API Explorer工具提供页面调用API功能，可在线快速实现相关API调用。将全部拒绝权限修改为私有权限示例



更多权限修改信息参考BOS API权限控制。

2. 使用SDK方式
   以Python SDK为例，将全部拒绝权限修改为私有权限

`bos_client.set_bucket_canned_acl(bucket_name, canned_acl.PRIVATE)`

3. 使用BOSCMD工具
   首先使用命令bcecmd -c yourConfPath配置相关参数：ak、sk、region、domain、AutoSwitchDomain等等。此时region、domain参数必须和要修改权限bucket的region、domain一致，而参数AutoSwitchDomain=no，配置完成后执行如下命令即可将全部拒绝权限修改为私有权限。
   `bcecmd --conf-path yourConfPath bosapi put-bucket-acl --bucket-name bucket1 --canned private`
   更多权限修改请参考设置Bucket ACL。

注意：以上三种恢复方式使用的AK-SK必须是**主账号**AK-SK。

Object上传下载及管理

BOS支持自定义文件属性吗？

BOS支持系统Meta和自定义Meta。系统Meta是由BOS维护的Meta信息如Content-Length、Last-Modified。自定义Meta是以"x-bce-meta-"开头的<key,value>对。您可以通过自定义meta来保存您希望存储的额外文件属性如x-bce-meta-owner: xiaoming。

如何更新Meta信息，例如Content-type?

您可以通过CopyObject来实现Meta信息更新。只需要把目标文件设为跟源文件一致，并加入您需要更新/添加的Meta信息即可。此外，控制台即将推出设置Meta功能，未来您可以在控制台console上完成Meta信息（如Content-Type）的更新。

如何进行文件分享？

BOS支持文件分享，即获取Object文件的链接然后进行分享。对于私有bucket，您可以在控制台或者通过SDK生成一个有时效性的临时分享链接。通过控制台生成分享链接见下载Object，通过SDK生成临时分享链接可参考Java sdk的获取Object的URL。

PutObject、PostObject、AppendObject和MultipartUpload这几种上传接口的区别是什么？

这几种上传接口都可以实现文件上传。其主要区别如下：

- PutObject通常用于小文件上传。
- PostObject主要用于浏览器环境下的上传。
- AppendObject主要用于增量上传场景，如监控录像实时上传、日志传输等。Appendable的文件随时可以查看。
- MultipartUpload主要用于大文件上传。MultipartUpload通过把大文件分块然后逐块上传，大幅度提高上传成功率和上传速度。需要注意的是，MultipartUpload在上传完成（commit）前是无法通过ListObjects查看该文件的。

CopyObject是否支持跨region拷贝?

CopyObject支持跨region拷贝Object。当进行跨区域Object复制时，复制产生的流量会收取跨区域流量费，跨区域收费标准参见产品定价。

如何跨bucket拷贝数据?

BOS支持CopyObject的API接口实现数据拷贝。为了便于您的使用，BOS console、CLI、SDK均封装了该接口。 当您有大量文件需要进行跨bucket拷贝时，您可以通过SDK来实现：首先通过ListObject获取需要拷贝的文件列表，然后用CopyObject执行拷贝。 需要注意的是，执行CopyObject需要您拥有源文件的读权限和目标文件的写入权限。如果目标bucket是另一个用户所有，您可以让bucket所有者为其bucket设置ACL来允许您的写入操作。更多细节请见Bucket权限控制。

如何进行大文件上传？

BOS的文件上传方案由数据传输服务（Data Import，俗称"寄盘"），命令行工具CLI，SDK和Console组成。

- 当文件非常多，通过网络上传变得不可行时，推荐通过Data Import服务进行上传。Data Import服务提供通过邮寄硬盘的方式快速完成数据导入。如果对此感兴趣，请通过提交工单咨询。
- 如果文件数量适中，或者网络上行带宽条件较好，您可以考虑通过CLI进行文件上传。CLI支持大文件断点续传。如何使用CLI进行上传请参考BCE CLI。CLI基于BOS Python SDK封装而成，如果您需要在其他语言环境下进行文件上传，也可以通过BOS SDK编写上传代码。
- 当文件数量较小时，您也可以直接通过管理控制台Console上传。

如何批量删除BOS中的文件？

BOS支持多种方式批量删除。

- 登录控制台并选中多个文件批量删除。
- BOS提供了批量删除的API接口DeleteMultipleObjects接口,可以实现通过一个HTTP请求删除同一个Bucket下的多个Object。
- 可以通过工具BOS CMD进行批量删除，详细操作指导请参考删除Object。

如何快速查看Bucket中是否存在某个文件？

可以通过Listobjects接口进行循环判断。

BOS是否支持断点续传？

BOS使用控制台上传文件时不支持断线续传，但BOS提供的API接口、SDK及工具都支持断点续传。

如何将标准BOS修改为低频BOS？

生命周期管理可以转换存储类型从标准存储到低频存储，BOS生命周期规则设置后会在一天内生效。规则生效后，BOS会对符合条件的Object进行相应的处理，但处理需要一定的时间(一般情况下为几小时)，所以设置规则后不一定能马上看到效果。规则中计算的时间（即Object的"年龄"）以Object的创建时间为准，而不是生命周期规则的创建／修改时间。BOS只保存文件的最后修改时间，即last-modified时间；如果不更新meta或者覆盖文件，那么last-modified就是创建时间。所以生命周期中的"创建时间"其实是last-modified时间。详细操作方法请参考管理生命周期。

BOS如何上传下载文件或者文件夹？

BOS支持以下多种方式上传文件：

1. 控制台直接上传Object。
2. 使用API或SDK进行上传下载，其中文件夹上传可以参见模拟文件夹功能；
3. 使用周边工具进行上传下载。Windows系统上传文件夹功能可以通过BOS桌面工具实现。

控制台最大支持上传5G的数据，批量上传下载或超过5G的大文件建议使用SDK或周边工具。

如何设置文件的Header属性？

可以通过Copyobject接口进行设置。其中CopyObject接口只能修改userMeta，需要您在上传时设置好content-type。

是否支持浏览器上传文件？

可以使用bce bos uploader工具实现，详细操作请参考web端直传实践。

BOS误删除的数据是否可以找回？

用户可以通过配置Bucket的回收站功能，保证删除后的数据在回收站内继续保留，用于后续对删除数据的找回。默认情况下，Bucket的回收站功能为关闭状态，详细描述请参见回收站。

如何设置才能直接浏览Object而不是下载？

如果想要设置访问BOS的资源是查看而不是直接下载，可以通过设置Object的Meta信息来指定，具体介绍和操作请参考设置Object的meta。

BOS文件上传后能否自动进行MD5校验？

如果是想要校验上传文件的完整性，API中可以使用content_md5参数进行文件数据校验。设置后BOS会启用文件内容MD5校验，把提供的MD5与文件的MD5比较，不一致会提示错误。详细描述请参考PutObject接口。使用分块上传时，建议在分块上传中使用每个分块BOS返回的Content-MD5值分别验证已上传分块数据的正确性。当所有分块数据合成一个Object后，不再含MD5值。

BOS提供的SDK均支持MD5校验，以JAVA为例，详细描述请参考文件管理。

BOS如何获取缩略图？

用户可以使用BOS提供的视频截帧功能生成略缩图。也可以使用VOD获取封面，具体接口参考查询指定媒资。

如何在ListObjects接口同时获取Object的User和Meta信息？

如果想要批量获取Object的User和Meta信息，可以使用GetObjectMeta接口。

BOS的Object获取地址能否用自定义域名？

可以使用自定义域名绑定Bucket后访问，详细操作请参考绑定Bucket域名。

怎样增量上传文件到BOS？

对象存储BOS中，可以通过以下方法迁移与备份增量数据：

1. BOS周边工具，如：Bcecmd中通过sync进行迁移与备份数据，Bos桌面提供了本地目录数据迁移与备份功能；

2. 镜像回源，可以通过Bucket域名访问对应文件资源，如果文件不存在会进行回源获取，并存储在Bucket中。

BOS图片处理中定义样式是否支持点号？

不支持，请参考图片样式名称限制说明：规则与限制

请问下BOS,上传视频后,能获取到视频的预览图吗？

对象存储BOS无法针对上传的视频文件，进行自动取帧生成缩略图，可以使用音视频转码MCT，其可以创建缩略图任务，进行生成缩略图。

BOS视频链接有没有类似于图片处理可以加参数获取一些数据?

对象存储BOS服务，目前针对音视频资源可以进行数据处理，具体请参考：音视频处理。

我上传文件到BOS里面,会产生哪些费用?

BOS 服务的计费项包括存储费用、流量费用、请求费用、数据处理费用四部分。根据您的使用情况，每一项会产生不同的费用。

询问下mp4文件是否能预览,文件信息中有预览功能,想看mp4的文件截图预览,怎么实现?

可以结合视频处理，实现对视频文件截取首帧或者指定帧数截取生成对应的封面图片，进行封面预览，具体参考音视频处理。

BOS存储图片/视频等资源，是否有接口支持查询视频的meta信息如视频尺寸，视频相关信息?

对象存储BOS并不支持查询存储视频的尺寸等信息，需要您通过第三方工具进行校验，如：ffprobe 请求BOS视频地址。

是否可以获取bucket中视频文件时长?

目前bos并没有接口支持获取对应视频文件时长，建议可以通过第三方工具进行获取，如：ffprobe等。

借助baidubce-sdk或bce-bos-uploader如何上传文件到buket中的指定子目录？？

可以通过objectkey进行指定对应资源子目录，如：object是“temp/abc.jpg”，那么其就是存储在temp目录下abc.jpg文件。

标准存储里面东西太多，怎么删除？

您可以通过以下方式进行批量删除资源：

1. 使用bos生命周期，其可以自动删除匹配规则的资源，具体请参考：管理生命周期

2. 使用bos周边工具，进行批量删除，如：bcecmd，具体请参考：删除object

BOS带宽不够，量级大的时候会有加载图片慢怎么办？

目前BOS 对单 Bucket 设置的带宽阈值为 10 Gbit/s，正常大量请求是可以满足的，通常影响加载慢是由于客户端网络带宽导致，可以更换网络环境进行测试。同时，如果您需要优化bos资源加载，也可以使用cdn，把资源缓存到cdn节点，这样相对直接请求bos资源会有所改善。

API&SDK&工具

BOS的错误响应如何定位？

当用户访问BOS出现错误时，BOS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。

在使用API或SDK上传Object时，如何判断一个Object是否上传成功？

- 使用API 上传Object时，如返回值为200，则表示上传成功，可以使用GetObjectMeta接口验证Object是否存在。
- 使用SDK上传Object时，如没有抛出异常，则表示上传成功，可参考各SDK文档中的“获取ObjectMeta信息”验证Object是否存在。

PutObject、PostObject、AppendObject和MultipartUpload这几种上传接口的区别是什么？

这几种上传接口都可以实现文件上传。其主要区别如下：

- PutObject通常用于小文件上传。

- PostObject主要用于浏览器环境下的上传。
- AppendObject主要用于增量上传场景，如监控录像实时上传、日志传输等。Appendable的文件随时可以查看。
- MultipartUpload主要用于大文件上传。MultipartUpload通过把大文件分块然后逐块上传，大幅度提高上传成功率和上传速度。需要注意的是，MultipartUpload在上传完成（commit）前是无法通过ListObjects查看该文件的。

如何跨bucket拷贝数据?

BOS支持CopyObject的API接口实现数据拷贝。为了便于您的使用，BOS console、CLI、SDK均封装了该接口。 当您有大量文件需要进行跨bucket拷贝时，您可以通过SDK来实现：首先通过ListObject获取需要拷贝的文件列表，然后用CopyObject执行拷贝。 需要注意的是，执行CopyObject需要您拥有源文件的读权限和目标文件的写入权限。如果目标bucket是另一个用户所有，您可以让bucket所有者为其bucket设置ACL来允许您的写入操作。更多细节请见Bucket权限控制。

如何验证一个Object/Bucket是否存在？

BOS支持GetObjectMeta和HeadBucket API。您可以通过这两个API来验证一个Object/Bucket是否存在。

BOS如何挂到linux服务器上？

BOS挂载到linux系统的BCC服务器上可以通过BOSFS实现（windos系统暂不支持挂载）。BOSFS用于实现将远程BOS服务挂载到本地文件系统中，实现类似于网络文件系统的功能，目的是便于用户使用系统命令或者标准的POSIX接口访问BOS上的文件，就像使用本地磁盘一样方便。如果在配置过程中遇到任何问题，建议提交工单处理。

BOS如何集成到APP中？

BOS的iOS和Android SDK对于集成没有限制。如果使用demo正常但集成报错，优先建议您按照demo集成，或把项目集成到demo工程中。如果使用demo异常，建议提交工单处理。

运行PHP SDK的Sample Code时，对phpunit版本有什么要求？

要求phpunit版本为4.5.0及以上版本。

如何使用BOS搭建FTP服务？

BOS无法直接当作FTP使用，但是您可以通过周边工具实现下载上传的动作来模拟FTP。或者调用API接口以及SDK自行开发，或者参考基于Python实现的BOS的FTP server工具BOSFTP。

UCenter和Python flask怎么对接BOS？

如果搭建了Wordpress，Discuz，UCenter，Python flask想要结合BOS服务使用，建议安装对应插件，一方面解决本地磁盘占用，另一方面提高图片等多媒体浏览速度。具体使用请参考第三方工具。

java sdk使用InputStream流的方式上传如何指定文件路径以及文件名？

如果想要实现javaSDK使用inputStream流的方式指定文件路径以及文件名，那么需要您将文件路径和文件名放在一起就可以了，在这个参数里把路径写全。

例如:a/b/c/1.txt那么到BOS里存储的文件路径就是目录是 a/b/c/ 文件名是1.txt。

请问有小程序直传的接口吗？

BOS对象存储服务目前没有微信小程序上传到BOS的案例。您可以使用BOS的API接口自行编写。参考：BOS对象存储API接口文档 ， 如还有问题请提交BOS对象存储工单。

BOS java SDK如何获取https资源地址？

您可以把java SDK中的endpoint设置为https，再调用generatePresignedUrl接口获取的https。具体参考：配置https协议访问BOS。

存储网关能否跨账号指定BOS的bucket？

支持，BOS并不会限制一个服务器中挂载多个bucket，且查看BSG是支持创建不同的共享目录，每个共享目录可以指定不同账号下的ak、sk和bucket。

bcecmd怎么排除文件夹？

BOS CMD中提供了--exclude参数进行排除指定目录上传资源。

如：bcecmd bos sync ./sample/ bos:/bce-test/test --exclude "./sample/app01" 表示：把本地目录sample下数据（sample下的app01目录不迁移与备份）迁移与备份到BOS中。

数据迁移与备份

如何将云磁盘CDS里面的内容转存到BOS上？

CDS磁盘内的数据是不可以直接转到BOS中的，但是可以在挂载此CDS磁盘的服务器中操作。例如若服务器为windows系统则安装BOS的周边工具直接复制粘贴上传即可，若为linux系统则可以使用BOS CLI等工具，也可以通过API接口访问或调用SDK实现。周边工具和BOS CLI详细介绍请参见BOS周边工具。

如何将其它云平台的数据同步到BOS中？

如果在其它云平台上存储的数据量比较大且想要迁移到BOS，可以使用百度智能云的数据迁移服务，详细操作请参见数据上云方案。

如何将BOS中的资源迁移到其他Bucket？

同账号下Bucket之间数据快速移动分为以下两种情况：

1. 如果是不同区域的Bucket，可以创建跨区域同步任务进行同步，详细操作请参考设置数据同步。

2. 如果是相同区域，可以使用BOS提供的周边工具进行复制，例如BOS CMD或BOS CLI等，具体请参考BOS周边工具。

是否可以将BOS里面的一部分数据复制到另外一个账号下的BOS？

跨账号数据迁移，可以使用迁移工具迁移，也可以配置在线迁移任务。

BOS存储资源目前是否支持跨账号迁移与备份?

BOS是支持资源跨账号迁移与备份的，您参考下设置数据迁移与备份这个文档，然后在控制台进行操作即可。

如何将bucket里的数据迁移到其他账号下？

如果想要将bucket里的数据迁移到其他账号下，目前有以下两种方案可用来进行bucket的数据迁移。（需要目标bucket赋予原bucket所在账号写权限）

1、配置跨账号的bucket数据迁移与备份。确认数据迁移与备份完成后，请记得取消数据迁移与备份，以免删除源bucket数据时导致目标bucket的数据被迁移与备份删除

2、使用bcecmd工具的sync指令实现bucket的数据迁移与备份。如果数据量较大，建议您使用nohup指令后台执行。

如何把华北保定区域的bucket数据迁移到华北北京地域的bucket里？

BOS 提供数据迁移与备份功能，可以实现不同地域的 2 个 Bucket 之间建立数据的自动迁移与备份关系，达到数据传输的目的。

具体步骤：

1.登录 对象存储 BOS 管理控制台。

2.在左侧 Bucket 列表中，选择需要设置权限的 Bucket，点击 Bucket 名称进入 Bucket 管理目录。

3.在上方导航栏选择 **配置管理** 页签。

4.在 **配置管理** 页面中选择 **高级配置**，在 **数据同步配置** 区域点击 **新增配置** 对该 Bucket 的数据迁移与备份进行配置。

5.在数据同步配置操作栏中，可以按需选择数据迁移与备份策略、数据迁移与备份对象、历史数据是否迁移与备份、迁移与备份目标区域、目标Bucket以及目标存储类型。

BOS同一个账号下两个bucket之间文件复制怎么操作,用哪个工具？

对象存储BOS中，可以通过以下方法进行不同Bucket间数据复制。

1. 使用BOS提供数据迁移与备份功能，其可以把不同账号的数据进行迁移；

2. 使用bos周边工具，其提供了部分云平台数据迁移方式，如：bos import，其可以实现跨账号迁移bos资源。

如何进行外部迁移?

外部资源迁移到对象存储BOS服务，可以通过以下方案操作：

1. 使用在线导入方式，提供对应迁移资源任务文件，bos会根据对应任务规则进行自动抓取保存；

2. 使用bosimport，其目前支持不同云平台资源迁移到对象存储BOS服务，如：阿里云OSS迁移到百度智能云对象存储BOS；

3. 使用bos api / sdk进行开发，可以结合自身业务开发集成，进行上传业务资源。

BOS服务有备份功能吗?

BOS对象存储服务是支持备份功能的。

1. 您可以把bucket内的数据备份到其他的bucket内，参考bucket数据迁移与备份功能 ；

2. 也可以开启bucket回收站功能，开启后删除的文件会自动保存在回收站内，您可以在回收站恢复数据。参考 回收站功能

拷贝object可以在不同账号之间进行吗?

BOS是支持进行跨账号拷贝数据的，但是您需要注意bucket权限（允许目标账号进行操作）

1. 通过copyobject接口进行操作，具体请参考：CopyObject接口；

2. 使用bucket数据迁移与备份，具体请参考：设置数据迁移与备份。

图片服务

BOS开启原图保护后，访问图片链接报“Access Denied”错误。

BOS开启原图保护后，如果图片获取者未验证被授权的签名，则只能通过样式方式访问处理后的图像，直接访问原图或通过处理参数访问的方式会被拒绝，详情参见原图保护。

BOS中的图片添加水印后，水印内容是否能自动适应图片？

目前BOS图片服务中的水印没有自适应图片功能，需要设定缩略设置、输出设置、水印设置，详情参考图片服务。

bucket图片样式是否有长度限制?

bos图片处理v2版本中，样式名称支持数字，字母和下划线组合，且首位不能为下划线，最长 64 个字符。具体请参考：规则与限制 。

BOS图片处理中，heic图片是否支持置灰?

目前heic格式是支持通过图片处理进行置灰的。感谢您对百度智能云的支持。

分析与统计

Python SDK是否支持统计BOS某个目录的文件数等信息？

对象存储BOS中，Python SDK并不支持统计Bucket某个目录的文件数等信息，而当前Java SDK是支持获取目录概览数据。

BOS可以对日志进行分析处理吗？

目前BOS暂时是不支持直接对访问日志进行分析处理的，BOS开启访问日志后，日志会保存到目标bucket中。您可以选择将bucket上保存的日志定时迁移与备份到百度智能云BCC主机上，然后利用BLS的收集器来自动采集服务器保存的BOS日志，来实现BOS日志分析的功能。但这样效果可能不太理想，BOS的日志也存在一定的延时。

文件总数量在哪里看？

对象存储BOS中，可以进行查看每个Bucket的文件数量。

查看方式：对象存储BOS控制台-Bucket列表-Bucket名称-概览

AWS S3 兼容

概述

AWS 的S3接口协议具有广泛的开源工具支持，BOS兼容AWS S3主要常用接口，对于基于S3接口协议开发的各类应用及服务，可以轻松迁移到BOS。

接口的具体兼容情况，在下文中做了详细叙述。

BOS的S3协议接口提供了对象存储的基本功能，为了更好的使用BOS丰富的产品功能，更推荐使用BOS接口。

服务域名

使用如下服务域名通过AWS S3协议访问BOS：

区域	访问Endpoint	协议
北京	s3.bj.bcebos.com	HTTP、 HTTPS
广州	s3.gz.bcebos.com	HTTP、 HTTPS
苏州	s3.su.bcebos.com	HTTP、 HTTPS
保定	s3.bd.bcebos.com	HTTP 、HTTPS
武汉	s3.fwh.bcebos.com	HTTP
香港	s3.hkg.bcebos.com	HTTP

通过AWS S3协议访问BOS和直接访问BOS类似，也支持path-style和 bucket virtual hosting两种方式，以GetObject举例，两种风格请求语法如下：

风格	请求语法
bucket virtual hosting	GET /object HTTP/1.1 Host: bucket.s3.bj.bcebos.com
Path-style	GET /bucket/object HTTP/1.1 Host: s3.bj.bcebos.com

> **注意**： 不支持使用自定义域名通过AWS S3协议访问BOS。

兼容签名认证

兼容最新的AWS Signature Version 4，签名方法见Authenticating Requests (AWS Signature Version 4)。

签名Header示例

以下是使用S3签名认证方法访问BOS的示例，使用Authorization请求Header提供身份验证信息，具体内容如下：

```
 Authorization: AWS4-HMAC-SHA256
Credential=82fa964ae**********0dfeea44c0683/20230216/bj/s3/aws4_request,
SignedHeaders=host;x-amz-content-sha256;x-amz-date,
Signature=98afff082015a6490a50567b2fa9a0e64f0ae81105a3a62da86bc50806c293fb
```

> **注意事项**
>
> 1. AWS4-HMAC-SHA256：用于计算签名的算法，该字符串指定AWS签名版本即AWS4和签名算法HMAC-SHA256。
> 2. Credential：包括用于计算签名的Access Key、日期、区域和服务，格式：`<access-key>/<date>/<bos-region>/s3/aws4_request`， 其中`<date>`使用日期格式为YYYYMMDD，`<bos-region>`对应BOS区域如下：

区域	Region
北京	bj
保定	bd
苏州	su
广州	gz
武汉	fwh
香港	hkg

> 详细说明请参考sigv4-auth-using-authorization-header。
>
> 3. 签名计算目前仅支持 Transfer Payload in a Single Chunk 和 Transfer Payload in Multiple Chunks 。

完整请求示例：

```
 GET / HTTP/1.1
Host: s3.bj.bcebos.com
Accept-Encoding: identity
User-Agent: Boto3/1.26.72 Python/3.9.6 Darwin/22.1.0 Botocore/1.29.72 Resource
X-Amz-Date: 20230216T025415Z
X-Amz-Content-SHA256: e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855
Authorization: AWS4-HMAC-SHA256 Credential=82fa964ae**********0dfeea44c0683/20230216/bj/s3/aws4_request, SignedHeaders=host;x-amz-content-sha256;x-amz-date,
Signature=98afff082015a6490a50567b2fa9a0e64f0ae81105a3a62da86bc50806c293fb
amz-sdk-invocation-id: d2f1690c-ea14-4298-8bf5-052f797d4b4d
amz-sdk-request: attempt=1

HTTP/1.1 200 OK
Date: Thu, 16 Feb 2023 02:54:16 GMT
Content-Type: application/xml
Content-Length: 9061
Connection: keep-alive
Server: BceBos
X-Amz-Id-2: vl7FaflEg8tsA058XjrWu/PAaxp5HdsDsBHYVzKdQM/Dz0M6Xk1zqL5ckWgBdMcwhQC3fHuZqNA3S1FrzeM/PA==
X-Amz-Request-Id: 05fb9355-e743-4900-be4d-e1be627d1ce2
```

使用aws sts访问bos

bos支持在使用s3方式访问bos时用sts鉴权。跟s3签名类似，需要额外在header中加入X-Amz-Security-Token。具体内容如下：

```
Authorization: AWS4-HMAC-SHA256
Credential=82fa964ae**********0dfeea44c0683/20230216/bj/s3/aws4_request,
SignedHeaders=host;x-amz-content-sha256;x-amz-date,
Signature=98afff082015a6490a50567b2fa9a0e64f0ae81105a3a62da86bc50806c293fb
X-Amz-Security-
Token=IQoJb3JpZ2luX2VjEMv%2F%2F%2F%2F%2F%2F%2F%2F%2FwEaCXVzLWVhc3QtMSJGMEQCIBSUbVdj9YGs2g0HkHsOHFdkwOozjARSKHL987NhhOC8AiBPepRU1obMvlbGU0T%2BWphFPgK%2Fqpxaf5Snvm5D
```

注意事项

1. sts需要使用百度云的sts系统，参考百度云STS。利用百度sts client生成的ak、sk、sts-token来计算签名。ak和sk按照S3签名认证方法计算出签名字符串填充到Signature，然后sts-token填充到字段 X-Amz-Security-Token

使用s3 python sdk的参考示例如下：

```
import boto3
import botocore
from botocore.client import Config
s3_client = boto3.client(
    's3',
    aws_access_key_id='xxx', # 百度云利用sts生成的ak
    aws_secret_access_key='xxx', # 百度云利用sts生成的sk
    aws_session_token='xxx',  # 百度云利用sts生成的token
    region_name='gz',
    endpoint_url='http://s3.gz.bcebos.com',
    config=Config(signature_version='s3v4', s3={
        'addressing_style': 'path',
        }),
)
response = s3_client.list_buckets()
print(response)
```

兼容公共头

公共请求头

AWS S3 请求头	描述
Authorization	兼容以AWS4-HMAC-SHA256开头V4版本认证
Content-Length	兼容
Content-Type	兼容
Content-MD5	兼容
Date	兼容
Expect	兼容
Host	兼容
x-amz-content-sha256	兼容
x-amz-date	兼容
x-amz-security-token	兼容

公共响应头

AWS S3 响应头	描述
Content-Length	兼容
Content-Type	兼容
Connection	兼容
Date	兼容
ETag	兼容
Server	设置为BceBos
x-amz-checksum-crc32	兼容
x-amz-delete-marker	未兼容
x-amz-request-id	设置为与原生响应头x-bce-request-id相同
x-amz-id-2	设置为与原生响应头x-bce-debug-id相同
x-amz-version-id	未兼容

兼容接口

Service API

AWS S3接口	描述
GET Service/List Buckets	兼容

Bucket API

AWS S3接口	描述
DELETE Bucket	兼容
GET Bucket/List Objects	兼容GET Bucket (List Objects) Version 1
GET Bucket acl	兼容
GET Bucket Location	兼容
HEAD Bucket	兼容
List Multipart Uploads	兼容
PUT Bucket	未支持x-amz-grant-*
PUT Bucket ACL	兼容
List Objects	兼容
List Objects V2	未支持 fetch-owner
OPTIONS	兼容
PUT Bucket LifeCycle	兼容，同时也支持配置多版本、标签生命周期。未支持配置 NoncurrentVersionTransition、ObjectSizeGreaterThan、ObjectSizeLessThan
GET Bucket LifeCycle	兼容
DELETE Bucket LifeCycle	兼容
PUT Bucket Logging	兼容，仅支持配置 TargetBucket、TargetPrefix
GET Bucket Logging	兼容
PUT Bucket Cors	兼容
GET Bucket Cors	兼容
DELETE Bucket Cors	兼容
PUT Bucket Website	兼容，仅支持配置 ErrorDocument、IndexDocument
GET Bucket Website	兼容
DELETE Bucket Website	兼容
PUT Bucket RequestPayment	兼容
GET Bucket RequestPayment	兼容
PUT Bucket Versioning	兼容
GET Bucket Versioning	兼容
List Object Version	兼容

Object API

AWS S3接口	描述
DELETE Object	未支持x-amz-mfa
DELETE Multiple Objects	未支持x-amz-mfa
GET Object	兼容
HEAD Object	未支持x-amz-server-side-encryption-*
PUT Object	未支持x-amz-grant-*，x-amz-server-side-encryption-*，x-amz-storage-class支持STANDARD，STANDARD_IA
PUT Object - Copy	兼容
Abort Multipart Upload	兼容
Complete Multipart Upload	兼容
Initiate Multipart Upload	未支持x-amz-grant-*，x-amz-server-side-encryption-*；x-amz-storage-class支持STANDARD，STANDARD_IA
List Parts	兼容
Upload Part	未支持x-amz-server-side-encryption-*
Upload Part Copy	兼容
POST Object	兼容
OPTIONS Object	兼容
POST Object restore	兼容，仅支持配置 Days和 GlacierJobParameters
PUT Object Acl	兼容
GET Object Acl	兼容
PUT Object Tagging	兼容
GET Object Tagging	兼容
DELETE Object Tagging	兼容

兼容工具

兼容说明

大多数基于AWS S3开发的工具都可以设置访问地址。通过设置这些工具的访问地址为BOS的AWS S3服务域名，您就可以使用这些工具访问BOS。下面以一些常用SDK和工具为例，说明如何对接BOS。

> **说明：**
>
> | 字串 | 含义 |
> | --- | --- |
> | $ACCESS_KEY | 百度智能云账号的Access key |
> | $SECRET_KEY | 百度智能云账号的Secret key |

AWS SDK for Python

1. 安装Boto类库：

```
pip install boto3
```

2. 使用AWS SDK for python访问Bos

```
import boto3
from botocore.client import Config
s3 = boto3.client(
        's3',
        aws_access_key_id=$ACCESS_KEY,
        aws_secret_access_key=$SECRET_KEY,
        endpoint_url='http://s3.bj.bcebos.com',
        region_name='bj',
        config = Config(
            signature_version='s3v4',
        )
    )
# Use S3 client
s3.create_bucket(...)
```

AWS SDK for Java

1. 添加依赖包到pom.xml

```
//添加下面AWS Java SDK依赖包到pom.xml
<dependency>
  <groupId>com.amazonaws</groupId>
  <artifactId>aws-java-sdk</artifactId>
  <version>1.11.82</version>
</dependency>
```

2. 使用AWS SDK for java访问Bos

```
import java.io.IOException;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3Client;
import com.amazonaws.services.s3.model.*;
import com.amazonaws.services.s3.S3ClientOptions;
import com.amazonaws.auth.BasicAWSCredentials;
import com.amazonaws.SDKGlobalConfiguration;

public class S3Sample {
    public static void main(String[] args) throws IOException {
        System.setProperty(SDKGlobalConfiguration.ENABLE_S3_SIGV4_SYSTEM_PROPERTY, "true");
        AmazonS3 s3 = new AmazonS3Client(new BasicAWSCredentials($ACCESS_KEY,$SECRET_KEY));
        s3.setEndpoint("s3.bj.bcebos.com");
        S3ClientOptions options = new S3ClientOptions();
        options.withChunkedEncodingDisabled(true);
        s3.setS3ClientOptions(options);

        // Use S3 Client
        s3.createBucket(...);
    }

}
```

3. 编译代码

```
mvn package
```

AWS PHP SDK

1. 安装:下载aws.phar，更多安装方式见AWS PHP SDK安装方式。

2. 使用AWS SDK for PHP访问Bos

```
<?php
require 'aws.phar';
use Aws\S3\S3Client;
use Aws\Exception\AwsException;

$s3Client = new S3Client([
    'version'     => 'latest',
    'region'      => 'bj',
    'credentials' => [
        'key'    => $ACCESS_KEY,
        'secret' => $SECRET_KEY,
    ],
    'endpoint' => 'https://s3.bj.bcebos.com',
    'signature_version' => 'v4',
]);

$buckets = $s3Client->listBuckets();
foreach ($buckets['Buckets'] as $bucket){
    echo $bucket['Name']."\n";
}
```

AWS Golang SDK

1. 安装

```
go get -u github.com/aws/aws-sdk-go
```

2. 使用AWS SDK for Golang访问Bos

```
 import (
   "github.com/aws/aws-sdk-go/aws"
   "github.com/aws/aws-sdk-go/aws/session"
   "github.com/aws/aws-sdk-go/service/s3"
   "github.com/aws/aws-sdk-go/aws/credentials"
)
conf := &aws.Config{
      Region:           aws.String("bj"),
      Endpoint:         aws.String("s3.bj.bcebos.com"),
      Credentials:      credentials.NewStaticCredentials($ACCESS_KEY, $SECRET_KEY,, ""),
   }
sess := session.Must(session.NewSessionWithOptions(session.Options{Config:*conf}))
svc := s3.New(sess)
getObjectParams := &s3.GetObjectInput{
      Bucket:             aws.String("my-bucket"),
      Key:                aws.String("my-object"),
}
getObjectResp, err := svc.GetObject(getObjectParams)
if err != nil {
   fmt.Println(err.Error())
   return
}
```

AWS JavaScript SDK

1. 安装与前置依赖

环境要求

- npm 或 yarn 包管理器
- Node.js环境：Node.js 14.x 或更高版本
- 浏览器环境
  - 现代浏览器环境（支持ES6+）
  - 支持ES Modules或使用打包工具（如Webpack、Vite等）

安装依赖包

```
 #### 安装核心S3客户端
npm install @aws-sdk/client-s3

#### 安装分片上传工具
npm install @aws-sdk/lib-storage

#### 安装STS客户端（用于临时凭证）
npm install @aws-sdk/client-sts

#### 安装凭证提供者（浏览器环境）
npm install @aws-sdk/credential-providers
```

浏览器环境CDN 可以通过skypack、jsdelivr等CDN服务引入在线的ES模块

```
 <script type="module">
 import {S3Client, CreateBucketCommand, PutObjectCommand, GetObjectCommand, DeleteObjectCommand} from 'https://cdn.skypack.dev/@aws-sdk/client-s3@3.826.0';
 import {Upload} from 'https://cdn.skypack.dev/@aws-sdk/lib-storage@3.826.0';
</script>
```

服务域名

- 例如北京：https://s3.bj.bcebos.com
- 全部兼容域名请参考：https://cloud.baidu.com/doc/BOS/s/xjwvyq9l4

2. 初始化配置

2.1 Node.js 服务端访问方式（固定AK/SK）

```
 import { S3Client } from '@aws-sdk/client-s3';

const s3Client = new S3Client({
   region: 'bj',
   endpoint: 'https://s3.bj.bcebos.com',
   credentials: {
      accessKeyId: '<your-access-key>',
      secretAccessKey: '<your-secret-access-key>'
   }
});
```

2.2 浏览器STS访问方式

```
 import {S3Client} from '@aws-sdk/client-s3';

const s3Client = new S3Client({
   region: 'bj',
   endpoint: 'https://s3.bj.bcebos.com',
   credentials: {
      accessKeyId: '<your-access-key>',
      secretAccessKey: '<your-secret-access-key>'
      /** 临时凭证，可选，通过百度云STS服务获取 */
      sessionToken: '<your-session-token>'
   }
});
```

3. 常见方法示例

3.1 创建存储桶

```
import { CreateBucketCommand } from '@aws-sdk/client-s3';

/**
 * 创建存储桶
 *
 * @param {string} bucketName 存储桶名称
 */
async function createBucket(bucketName) {
  try {
    const command = new CreateBucketCommand({
      /** 存储桶名称 */
      Bucket: bucketName,
      /** 存储桶配置 */
      CreateBucketConfiguration: {
        /** 存储桶所在区域 */
        LocationConstraint: 'bj'
      }
    });

    const response = await s3Client.send(command);
    console.log('存储桶创建成功:', response);
    return response;
  } catch (error) {
    console.error('创建存储桶失败:', error);
    throw error;
  }
}

// 使用示例
await createBucket('my-test-bucket-beijing-2024');
```

3.2 简单上传（PutObject） Node.js环境

```
import fs from 'fs';
import {PutObjectCommand} from '@aws-sdk/client-s3';

/**
 * Node.js环境上传文件
 *
 * @param {string} bucketName 存储桶名称
 * @param {string} key 对象名称
 * @param {string} filePath 本地文件路径
 * @returns
 */
async function uploadFile(bucketName, key, filePath) {
  try {
    const fileContent = fs.readFileSync(filePath);

    const command = new PutObjectCommand({
      Bucket: bucketName,
      Key: key,
      Body: fileContent,
      ContentType: 'application/octet-stream',
      Metadata: {
        'uploaded-by': 'aws-sdk-v3',
        'upload-time': new Date().toISOString()
      }
    });

    const response = await s3Client.send(command);
    console.log('文件上传成功:', response);
    return response;
  } catch (error) {
    console.error('文件上传失败:', error);
    throw error;
  }
}

// 使用示例
await uploadFile('my-test-bucket', 'documents/test.pdf', './local-file.pdf');
```

浏览器环境

```
import {PutObjectCommand} from '@aws-sdk/client-s3';

/**
 * 浏览器环境上传文件
 *
 * @param {string} bucketName 存储桶名称
 * @param {string} key 对象名称
 * @param {File} file 文件对象
 * @returns
 */
async function uploadFileFromBrowser(bucketName, key, file) {
  try {
    const command = new PutObjectCommand({
      Bucket: bucketName,
      Key: key,
      Body: file,
      ContentType: file.type || 'application/octet-stream'
    });

    const response = await s3Client.send(command);
    console.log('文件上传成功:', response);
    return response;
  } catch (error) {
    console.error('文件上传失败:', error);
    throw error;
  }
}

// 使用示例
// 假设页面中存在上传控件：<input type="file" id="fileUpload">
const fileInput = document.getElementById('fileUpload');
const file = fileInput.files[0];
await uploadFileFromBrowser('my-test-bucket', 'documents/test.pdf', file);
```

3.3 分片上传 Node.js环境

```
import fs from 'fs';
import {Upload} from '@aws-sdk/lib-storage';


/**
 * Node.js环境分片上传文件
 *
 * @param {string} bucketName 存储桶名称
 * @param {string} key 对象名称
 * @param {string} filePath 本地文件路径
 */
async function multipartUpload(bucketName, key, filePath) {
  try {
    const fileStream = fs.createReadStream(filePath);
    const upload = new Upload({
      client: s3Client,
      params: {
        Bucket: bucketName,
        Key: key,
        Body: fileStream,
        ContentType: 'application/octet-stream'
      },
      /** 并发上传数量，可选参数 */
      queueSize: 4,
      /** 分片大小，可选参数，至少5MB */
      partSize: 5 * 1024 * 1024,
      /** 失败时是否保留已上传的分片，可选参数 */
      leavePartsOnError: false
    });

    /** 监听上传进度 */
    upload.on('httpUploadProgress', (progress) => {
      console.log(`上传进度: ${Math.round(progress.loaded / progress.total * 100)}%`);
    });

    const response = await upload.done();
    console.log('分片上传完成:', response);
    return response;
  } catch (error) {
    console.error('分片上传失败:', error);
    throw error;
  }
}

// 使用示例
await multipartUpload('my-test-bucket', 'large-files/video.mp4', './large-video.mp4');
```

浏览器环境

```
import {Upload} from '@aws-sdk/lib-storage';

/**
 * 浏览器环境分片上传文件
 *
 * @param {string} bucketName 存储桶名称
 * @param {string} key 对象名称
 * @param {File} file 文件对象
 */
async function multipartUploadBrowser(bucketName, key, file) {
  try {
    const upload = new Upload({
      client: s3Client,
      params: {
        Bucket: bucketName,
        Key: key,
        Body: file,
        ContentType: file.type || 'application/octet-stream'
      },
      /** 并发上传数量，可选参数 */
      queueSize: 4,
      /** 分片大小，可选参数，至少5MB */
      partSize: 5 * 1024 * 1024,
      /** 失败时是否保留已上传的分片，可选参数 */
      leavePartsOnError: false
    });

    upload.on('httpUploadProgress', (progress) => {
      const percent = Math.round(progress.loaded / progress.total * 100);
      console.log(`上传进度: ${percent}%`);
      // 更新进度条UI
    });

    const response = await upload.done();
    console.log('分片上传完成:', response);
    return response;
  } catch (error) {
    console.error('分片上传失败:', error);
    throw error;
  }
}

// 使用示例
// 假设页面中存在上传控件：<input type="file" id="fileUpload">
const fileInput = document.getElementById('fileUpload');
const file = fileInput.files[0];
await multipartUploadBrowser('my-test-bucket', 'large-files/video.mp4', file);
```

3.4 下载文件 Node.js环境

```
import fs from 'node:fs';
import {pipeline} from 'node:stream/promises';
import {GetObjectCommand} from '@aws-sdk/client-s3';

/**
 * Node.js环境下载文件
 *
 * @param {string} bucketName 存储桶名称
 * @param {string} key 对象名称
 * @param {string} downloadPath 本地下载路径
 */
async function downloadFile(bucketName, key, downloadPath) {
  try {
    const command = new GetObjectCommand({
      Bucket: bucketName,
      Key: key
    });

    const response = await s3Client.send(command);

    const writeStream = fs.createWriteStream(downloadPath);
    await pipeline(response.Body, writeStream);
    console.log('文件下载成功:', downloadPath);
    return downloadPath;
  } catch (error) {
    console.error('文件下载失败:', error);
    throw error;
  }
}

// 使用示例
await downloadFile('my-test-bucket', 'documents/test.pdf', './downloaded-file.pdf');
```

浏览器环境

```
import {GetObjectCommand} from '@aws-sdk/client-s3';

/**
 * 浏览器环境下载文件
 *
 * @param {string} bucketName 存储桶名称
 * @param {string} key 对象名称
 */
async function downloadFileInBrowser(bucketName, key) {
  try {
    const command = new GetObjectCommand({
      Bucket: bucketName,
      Key: key
    });

    const response = await s3Client.send(command);

    const blob = await response.Body.transformToByteArray();
    const file = new Blob([blob], {
      type: response.ContentType || 'application/octet-stream'
    });

    const url = URL.createObjectURL(file);
    const downloadLink = document.createElement('a');
    downloadLink.href = url;
    downloadLink.download = key.split('/').pop();
    downloadLink.click();
    URL.revokeObjectURL(url);


    console.log(`文件下载成功: ${key}`);
    return blob;
  } catch (error) {
    console.error('文件下载失败:', error);
    throw error;
  }
}

// 使用示例
await downloadFileInBrowser('my-test-bucket', 'documents/test.pdf');
```

3.5 获取文件下载链接

```
import {GetObjectCommand} from '@aws-sdk/client-s3';
import {getSignedUrl} from '@aws-sdk/s3-request-presigner';

/**
 * 获取对象的下载链接
 *
 * @param {string} bucketName 存储桶名称
 * @param {string} key 对象名称
 * @param {number} expiresIn 过期时间，单位秒
 * @returns
 */
async function getDownloadUrl(bucketName, key, expiresIn = 3600) {
  try {
    const command = new GetObjectCommand({
      Bucket: bucketName,
      Key: key
    });

    const url = await getSignedUrl(s3Client, command, {expiresIn});
    console.log("预签名下载URL:", url);
    return url;
  } catch (error) {
    console.error('生成下载URL失败:', error);
    throw error;
  }
}

// 使用示例
const downloadUrl = await getDownloadUrl('my-test-bucket', 'documents/test.pdf', 7200);
```

3.6 删除文件

```
import {DeleteObjectCommand, DeleteObjectsCommand } from '@aws-sdk/client-s3';

/**
 * 删除单文件
 *
 * @param {string} bucketName 存储桶名称
 * @param {string} key 对象名称
 */
async function deleteFile(bucketName, key) {
  try {
    const command = new DeleteObjectCommand({
      Bucket: bucketName,
      Key: key
    });

    const response = await s3Client.send(command);
    console.log("文件删除成功:", response);
    return response;
  } catch (error) {
    console.error("文件删除失败:", error);
    throw error;
  }
}

/**
 * 批量删除文件
 *
 * @param {string} bucketName 存储桶名称
 * @param {string[]} keys 对象名称数组
 */
async function deleteMultipleFiles(bucketName, keys) {
  try {
    const command = new DeleteObjectsCommand({
      Bucket: bucketName,
      Delete: {
        /** 对象名称列表 */
        Objects: keys.map(key => ({Key: key})),
        /** 是否开启静默删除 */
        Quiet: false
      }
    });

    const response = await s3Client.send(command);
    console.log("文件批量删除成功:", response);
    return response;
  } catch (error) {
    console.error("文件批量删除失败:", error);
    throw error;
  }
}

// 使用示例
await deleteFile('my-test-bucket', 'documents/test.pdf');
await deleteMultipleFiles('my-test-bucket', ['file1.txt', 'file2.txt', 'folder/file3.jpg']);
```

4. 常见错误说明

4.1 服务端错误

常见的错误和对应的描述参考：https://cloud.baidu.com/doc/BOS/s/Ajwvysfpl

```
// 错误处理示例
async function handleS3Errors() {
  try {
    // 您的S3操作代码
  } catch (error) {
    console.error('S3操作失败:', error);
    /** 错误码 */
    console.log(error.Code);
    /** 错误消息 */
    console.log(error.message);
    /** HTTP状态码 */
    console.log(error.$metadata.httpStatusCode);
    /** 请求ID */
    console.log(error.RequestId);
    console.log(error.$metadata.requestId);
    /** x-bce-debug-id */
    console.log(error.$metadata.extendedRequestId);
  }
}
```

4.2 常见问题及解决方案

```
问题1：权限不足
错误：AccessDenied: User is not authorized to perform: s3:PutObject
解决：检查IAM策略是否包含必要权限
```

```
问题2：CORS配置问题（浏览器环境）
错误：Access to XMLHttpRequest blocked by CORS policy
解决：在存储桶中配置CORS
```

4.3 调试建议

1. 启用详细日志：

```
import { S3Client } from '@aws-sdk/client-s3';

const s3Client = new S3Client({
    region: 'bj',
    endpoint: 'https://s3.bj.bcebos.com',
    credentials: {
        accessKeyId: '<your-access-key>',
        secretAccessKey: '<your-secret-access-key>'
    }
    /* 启用详细日志 */
    logger: console,
    /* 请求设置 */
    requestHandler: {
        connectionTimeout: 5000,
        socketTimeout: 10000
    }
});
```

2. 检查网络连接：

```
#### 测试Endpoint连接
curl -I https://s3.bj.bcebos.com
```

AWS CLI工具

1. 安装AWS CLI工具

```
pip install awscli
```

2. 使用AWS CLI访问BOS

- 编辑配置文件

```
$ aws configure

AWS Access Key ID [None]: <access_key_id>
AWS Secret Access Key [None]: <access_key_secret>
Default region name [None]: auto
Default output format [None]: json
```

- 执行命令示例

```
aws s3api list-buckets --endpoint-url https://s3.bj.bcebos.com
aws s3api list-objects --bucket bucketname --endpoint-url https://s3.bj.bcebos.com
```

- 参考文档

https://docs.aws.amazon.com/cli/latest/userguide/cli-chap-welcome.html

Hadoop S3A工具

S3A为hadoop官方提供的在hadoop系统中使用S3的工具包。通过S3A您可以像操作hdfs一样操作S3存储，目前BOS已经支持大部分常用的S3A的功能。关于S3A更为详细的介绍可参见：S3 Support in Apache Hadoop和Hadoop-AWS module: Integration with Amazon Web Services。

1. 下载依赖包
    - 确保hadoop系统中存在如下依赖包：
        - aws-java-sdk-1.11.82.jar
        - aws-java-sdk-core-1.11.82.jar
        - aws-java-sdk-kms-1.11.82.jar
        - aws-java-sdk-s3-1.11.82.jar
    - 使用BOS适用版Jar包：由于目前BOS对S3的部分功能暂未实现兼容，为了更好的产品体验，请务必使用该定制版的工具包。

BOS适用版Jar包	MD5码
hadoop-aws-2.8.0.jar	6ffbdc9352b9399e169005aeeb09ee98

2. 修改S3A的相关配置

```
 <property>
 <name>fs.s3a.endpoint</name>
 <value>http://s3.bj.bcebos.com</value>
</property>
<property>
 <name>fs.s3a.signing-algorithm</name>
 <value>AWSS3V4SignerType</value>
</property>
<!-- 开启bos后端接口-->
<property>
    <name>fs.s3a.bos.compat.access</name>
 <value>true</value>
</property>
<property>
<name>fs.s3a.access.key</name>
<value>$ACCESS_KEY</value>
</property>
<property>
<name>fs.s3a.secret.key</name>
<value>$SECRET_KEY</value>
</property>
```

3. 使用S3A在hadoop环境下体验BOS服务，执行命令

```
hadoop fs -ls s3a://$YOUR_BUCKET_NAME
```

4. 如何自行修改hadoop-aws

（1）. 下载hadoop源码 （2）. 修改hadoop-2.8.0-src/hadoop-tools/hadoop-aws/src/main/java/org/apache/hadoop/fs/s3a/S3ClientFactory.java文件。由于BOS暂未支持Chunked Upload，而Java sdk默认写操作会使用该方式，所以需要在代码中关闭sdk的这个操作。

```
 // 找到s3a使用的aws client,例如：
// 在createAmazonS3Client函数下的
// AmazonS3 s3 = new AmazonS3Client(credentials, awsConf);
// 为aws client添加禁止chunkedencoding
s3.setS3ClientOptions(new S3ClientOptions().withChunkedEncodingDisabled(true));
```

(3). 编译得到jar包

```
 cd hadoop-2.8.0-src/hadoop-tools/hadoop-aws
mvn package
```

CloudBerry Explorer for Amazon S3

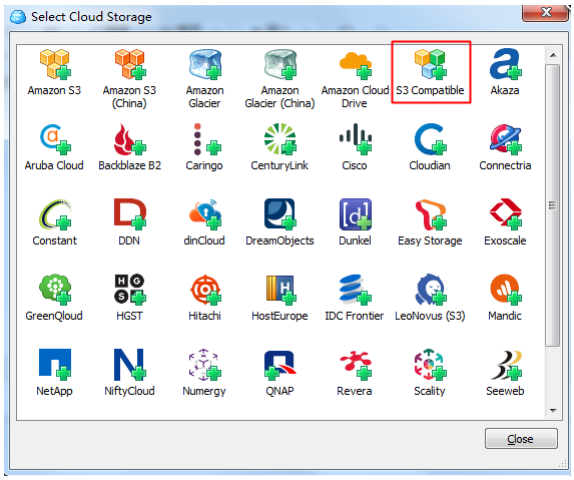
CloudBerry Explorer for Amazon S3是CloudBerry Lab提供的S3图形化数据管理软件，图形化界面非常强大，支持数据从本地到云端的上传、下载、同步、删除等管理操作。

现在百度智能云也支持通过CloudBerry Explorer来管理BOS上的资源了！

1. 选择和您操作系统相匹配的软件版本，下载后安装软件。

> **说明：** 该软件支持免费和付费两个版本。您可以直接使用免费版本。

2. 安装完成后先配置云存储，选择“S3 Compatible”：

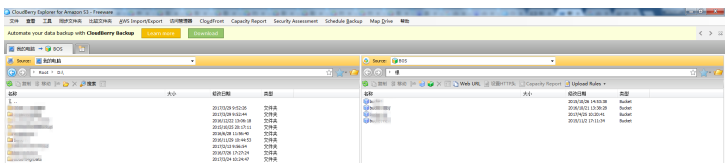


3. 添加新的S3 Compatible的账号，其中显示名可以填写BOS，服务点即BOS的服务域名，访问密钥和密码Key填写百度智能云的AK/SK。AWS S3兼容服务域名暂时不支持HTTPS，请去掉“使用SSL”的勾选，Signature version选4。



4. 点击“测试连接”测试连通性，显示“Connection success”表示连通成功。

5. 下面就可以使用CloudBerry Explorer进行各种数据管理啦！



视频专区

了解BOS

BOS分级存储

Object上传

通过域名访问BOS资源

服务等级协议SLA

BOS服务等级协议SLA(V3.0)

协议生效时间：2023年 11 月 22 日

本服务等级协议（Service Level Agreement，简称“SLA”） 规定了百度智能云向客户提供的对象存储服务BOS（Baidu Object Storage）的服务可用性等级指标及赔偿方案。

1. 定义

服务周期：一个服务周期为一个自然月。
失败请求：BOS将HTTP状态码为5XX的请求和因为BOS服务故障导致的用户正常请求未能到达BOS服务器端的请求视为失败请求，但不包括以下类型的请求：
(1) 跨区域复制(Region Replication) 和生命周期管理(Lifecycle)的后端异步处理的失败请求；
(2) 获取Bucket列表产生的失败请求；
(3) 由BOS服务发起的合理升级、变更、停机而导致的错误请求或服务不可用情况；
(4) 客户的应用程序受到黑客攻击而导致被BOS限制的请求；
(5) 来自于国外，向国内BOS服务产生的请求。
有效请求：BOS服务器端收到的请求视为有效请求，但不包括以下类型的请求：
(1) 未通过BOS身份验证和鉴权失败的请求、欠费请求；
(2) 跨区域复制(Region Replication)和生命周期管理(Lifecycle)的后端异步处理发起的所有请求；
(3) 获取Bucket列表发起的所有请求；
(4) 客户的应用程序受到黑客攻击而发起的请求。
每5分钟错误率：根据地域及存储类型分别以每5分钟为单位，按照如下公式进行计算：

$$每\ 5\ 分钟错误率 = \left(\frac{每\ 5\ 分钟失败请求数}{每\ 5\ 分钟总有效请求数}\right) \times 100\%$$

月度服务费用：指客户的某一个百度智能云账号在一个自然月中使用对象存储服务（BOS）按地域和存储类型分别统计的服务费用。如果用户通过预付费存储包的方式抵扣了费用，则将抵扣的容量部分按照后付费模式计算费用。

2. 服务可用性

2.1 服务可用性计算公式

可用性按服务周期统计，一个服务周期为一个自然月。为客户某一百度智能云账号下BOS的地域及存储类型（标准存储、低频存储、冷存储、归档存储）分别统计服务可用性。同一账号下相同地域、相同存储类型的多个Bucket，将合并计算服务可用性。

服务可用性将根据服务周期内每5分钟错误率之和除以服务周期内5分钟的总个数计算出每5分钟错误率的平均值，按照如下方式计算：

$$服务可用性 = \left(1 - \frac{服务周期内\sum 每\ 5\ 分钟错误率}{服务周期内\ 5\ 分钟个数}\right) \times 100\%$$

（注：服务周期内5分钟总个数=12 * 24 * 该服务周期的天数）

2.2 服务可用性承诺

存储类型	服务可用性
标准存储-多AZ	99.995%
低频存储-多AZ	99.995%
标准存储	99.99%
低频存储	99.95%
冷存储	99.95%
归档存储	99.00%

如BOS未达到上述服务可用性承诺，客户可以申请获得赔偿。
赔偿范围不包括以下原因所导致的服务不可用：
(1) 百度智能云预先通知用户后进行系统维护所引起的，包括合理升级、变更、停机、割接、维修和模拟故障演练；
(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) 用户内容违规或其他原因导致域名被封禁而产生的错误；
(4) 用户的应用程序或数据信息受到黑客攻击而引起的；
(5) 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；
(6) 用户的疏忽或由用户授权的操作所引起的；
(7) 客户未遵循百度智能云产品使用文档或使用建议引起的，如客户在控制台、API或者CLI等控制方式对BOS进行操作引起的不可用；
(8) 由于客户所安装软件或者其他非百度智能云直接运营的第三方软件或者配置引起的错误；
(9) 不可抗力以及意外事件引起的；
(10) 其他非百度智能云原因所造成的不可用。

3. 赔偿方案

3.1 赔偿标准

根据客户某一百度智能云账号下BOS的地域及存储类型分别统计月度服务可用性，按照下表中的标准计算赔偿金额。赔偿方式仅限于用于支付BOS产品的代金券，且赔偿总额不超过未达到服务可用性承诺的当月该地域下该存储类型所支付月度服务费用总额的50%（此处的月度服务费不含用代金券、优惠券、服务费减免等抵扣的费用）。

存储类型	服务可用性	赔偿代金券金额
标准存储-多AZ/低频存储-多AZ	低于99.995%但等于或高于99.90%	月度服务费用的10%
	低于99.90%但等于或高于95.00%	月度服务费用的30%
	低于95.00%	月度服务费用的50%
标准存储	低于99.99%但等于或高于99.90%	月度服务费用的10%
	低于99.90%但等于或高于95.00%	月度服务费用的30%
	低于95.00%	月度服务费用的50%
低频存储/冷存储	低于99.95%但等于或高于 99.90%	月度服务费用的10%
	低于99.90%但等于或高于 95.00%	月度服务费用的30%
	低于95.00%	月度服务费用的50%
归档存储	低于99.00%但等于或高于95.00%	月度服务费用的30%
	低于95.00%	月度服务费用的50%

3.2 赔偿申请时限

客户可以在每月第五（5）个工作日后对上个月没有达到可用性的服务提出赔偿申请。赔偿申请必须限于在BOS没有达到可用性的相关月份结束后两（2）个月内提出。超出申请时限的赔偿申请将不被受理。

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条款做出修改。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，您有权停止使用对象存储服务，如您继续使用对象存储服务，则视为您接受修改后的SLA。
(3) 如因法律法规、监管规定等外部环境发生变化需对协议进行必要变更或终止，百度智能云将根据实际情况及监管要求对产品服务功能进行必要的变更或终止。百度智能云会与您协商做好服务变更或终止的过渡安排。
(4) 本协议项下百度智能云BOS服务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；该类通知于发送之日视为已送达收件人。
(5 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。
(6) 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。
(7) 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。
(8) 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。
(9) 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。

相关协议

百度智能云对象存储联动百度网盘使用协议

请您在使用百度智能云对象存储（以下简称BOS）联动百度网盘存储分享功能前，仔细阅读并了解本《百度智能云对象存储联动百度网盘产品功能的使用协议》（以下简称本协议）。

本协议由《百度智能云用户服务协议》通用条款以及以下专用服务条款共同构成，如遇通用服务条款与专用服务条款二者不一致之处，优先适用专用服务条款。

1. 本协议适用的核心业务功能如下：通过授予BOS对百度网盘独立空间的访问和操作权利，将BOS中的文件或文件夹同步至百度网盘企业定制服务专属空间（以下简称网盘专属空间），并生成网盘分享链接以便于您可以通过网盘完成相关文件的分享。
2. 如您希望使用上述功能，您知悉并同意授权BOS产品端读取、使用您的企业定制网盘账号信息，百度网盘AK、SK、APPID。
3. 您知悉并同意，BOS读取使用您的上述信息，目的是为您提供BOS文件的网盘分享服务。如您不再使用上述服务，BOS将根据您的要求停止使用上述信息。
4. BOS将会为您的上述信息采取符合当前法律要求与行业标准水平的安全保障措施，包括但不限于加密。
5. 除适用于为您开通上述功能外，未经您的同意或法律法规强制性要求，BOS不进行对外共享、转让、披露您的企业定制网盘账号信息。
6. 您有更改、删除您企业定制网盘账号信息的权利，您可通过BOS控制台解除本协议项下的授权，具体方法详见BOS控制台网盘联动分享解绑操作。
7. 如您停止使用上述功能，请您按第6条的约定停止本协议项下授权。如您未停止授权，BOS将继续按本协议的约定提供服务。您理解并承诺，您在使用本协议项下网盘分享服务期间，您分享的文件内容、分享的范围、方式等均需符合法律法规、部门规章或国家政策的要求，如违反法律法规、部门规章或国家政策，百度智能云有权暂停或终止为您提供网盘分享服务，因此造成的全部结果及责任由您自行承担，并应赔偿给百度智能云造成的全部损失。
8. 您理解并认可，您在使用百度网盘企业定制服务期间，应遵守百度网盘企业定制平台的相关服务规定，如您未遵守百度网盘产品相关规定，或因为使用百度网盘产品本身产生的问题或争议，您需要与百度网盘企业定制平台沟通处理。例如，如您未购买网盘服务、或存在欠费、或因企业定制网盘服务网络故障等可能引发网盘联动功能无法正常使用的后果，包括但不限于同步失败、分享链接无效、分享失败等，BOS不对这些后果负责。
9. 您理解并认可，您在使用本功能时，需确认网盘专属空间中的文件数据与BOS中源数据的一致性。您应确认一致性后，再进行BOS中源数据的修改或删除。
10. 您理解并认可，BOS为向您提供稳定的网盘联动功能，需对百度网盘企业定制服务进行必要的配置操作，例如分配转存数据需使用您的企业定制服务网盘存储空间，上传数据至网盘存储空间需对您的用户身份进行回调验证。
11. 您理解本协议上述条款约定均是为您提供服务之必需。如您不同意本协议内容，将导致您无法正常使用本协议项下网盘分享服务。 如您有任何疑问，可联系您的商务经理或者通过提交工单的方式联系我们。