对象存储（COS）

产品文档

版权所有：腾讯云金融专区

文档目录

- 产品简介
  - 产品介绍
  - 功能概览
  - 规格与限制
- 购买指南
  - 计费概述
  - 购买指引
  - 欠费说明
  - 查看消费明细
- 快速入门
  - 基本概念
  - 创建存储桶
  - 上传对象
  - 下载对象
  - 删除对象
  - 删除存储桶
  - 后续步骤
- 控制台指南
  - 控制台概述
  - 存储桶管理
    - 创建与删除
    - 浏览与查询
    - 设置访问权限
    - 设置防盗链
    - 设置对象锁定
    - 设置生命周期
    - 设置跨域访问
    - 设置静态网站
    - 添加存储桶策略
    - 设置存储桶标签
    - 设置版本控制
    - 设置跨地域复制
    - 设置存储桶加密
    - 设置自定义域名
    - 设置日志管理
    - 设置事件通知
  - 对象管理
    - 上传对象
    - 下载对象
    - 复制对象
    - 查看对象信息
    - 搜索对象
    - 排序或筛选对象
    - 文件重命名
    - 修改存储类型
    - 设置对象的访问权限
    - 服务端加密
    - 自定义 Headers
    - 删除对象
    - 文件夹管理
      - 创建文件夹
      - 删除文件夹
    - 设置对象标签
    - 还原历史版本对象
  - 数据统计

版权所有：腾讯云金融专区

基础数据统计
返回码统计
监控告警
设置监控告警
高级功能
支持IPV6访问
地域白名单
开发者指南
存储桶
存储桶概述
创建存储桶
删除存储桶
对象
对象概述
上传对象
简单上传
分块上传
预签名授权上传
获取对象
简单获取对象
预签名授权下载
列出对象键
复制对象
简单复制
分块复制
删除对象
删除单个对象
数据管理
生命周期管理
生命周期概述
生命周期配置元素
配置生命周期
托管静态网站
存储桶标签概述
日志管理
日志管理概述
日志管理限制
数据容灾
版本控制
版本控制概述
删除标记
版本控制配置
跨地域复制
跨地域复制概述
复制行为说明
跨地域复制配置
访问管理
访问策略语言概述
最佳实践
访问控制与权限管理
ACL 访问控制实践
访问管理实践
授权子账号访问COS
权限设置相关案例
COS API授权策略使用指引
用于前端直传COS的临时密钥安全指引
临时密钥生成及使用指引

- 授予其他主帐号下的子帐号操作名下存储桶的权限
- 性能优化
  - 请求速率与性能优化
- 数据迁移
  - 本地数据迁移至COS
  - 第三方云存储数据迁移至COS
  - 以URL作为源地址的数据迁移至COS
  - COS之间数据迁移
  - Hadoop文件系统与COS之间的数据迁移
- 数据容灾备份
  - 基于跨地域复制的容灾高可用架构
- 域名管理实践
  - 设置跨域访问
  - 托管静态网站
  - 使用COS静态网站功能搭建前端单页应用
- 数据直传
  - Web端直传实践
- 数据安全
  - 防盗链实践
- 快速搭建移动应用传输服务
- 使用AWS S3 SDK访问COS

工具指南
- 工具概览
- 环境安装与配置
  - Java 安装与配置
  - Python 安装与配置
  - Hadoop 安装与测试
  - COSBrowser工具配置
- COSBrowser工具
- COSCMD工具
- COS Migration工具
- Hadoop工具
- HDFS TO COS工具

SDK文档
- SDK概览
- Android SDK
  - 快速入门
  - 接口文档
- iOS SDK
  - 快速入门
  - 接口文档
- C SDK
  - 快速入门
  - 接口文档
- C++ SDK
  - 快速入门
  - 接口文档
- C# SDK
  - 快速入门
  - 接口文档
- Go SDK
  - 快速入门
  - 接口文档
- Java SDK
  - 快速入门
  - 接口文档
- JavaScript SDK

- 快速入门
- 接口文档
- Node.js SDK
  - 快速入门
  - 接口文档
- PHP SDK
  - 快速入门
  - 接口文档
- Python SDK
  - 快速入门
  - 接口文档
- COS API参考
  - 简介
  - 公共请求头部
  - 公共响应头部
  - 错误码
  - 请求签名
  - 操作列表
  - Service接口
    - 查询存储桶列表
  - Bucket接口
    - 基本操作
      - 创建存储桶
      - 查询对象列表
      - 检索存储桶及其权限
      - 删除存储桶
      - 查询对象版本
    - 访问控制（acl）
      - 设置存储桶ACL
      - 查询存储桶ACL
    - 跨域资源共享（cors）
      - 设置跨域配置
      - 查询跨域配置
      - 删除跨域配置
    - 生命周期（lifecycle）
      - 设置生命周期
      - 查询生命周期
      - 删除生命周期
    - 存储桶策略（policy）
      - 设置存储桶策略
      - 查询存储桶策略
      - 删除存储桶策略
    - 防盗链（referer）
      - 设置存储桶referer
      - 查询存储桶referer
    - 标签（tagging）
      - 设置存储桶标签
      - 查询存储桶标签
      - 删除存储桶标签
    - 静态网站（website）
      - 设置静态网站
      - 查询静态网站
      - 删除静态网站
    - 版本控制（versioning）
      - 设置版本控制
      - 查询版本控制
    - 跨地域复制（replication）

设置跨地域复制
查询跨地域复制
删除跨地域复制
日志管理（logging）
PUT Bucket logging
GET Bucket logging
存储桶加密（encryption）
PUT Bucket encryption
GET Bucket encryption
DELETE Bucket encryption
对象锁定（ObjectLock）
PUT Bucket ObjectLockConfiguration
GET Bucket ObjectLockConfiguration
GET Object retention
Object接口
基本操作
简单上传对象
设置对象复制
表单上传对象
下载对象
查询对象元数据
删除单个对象
删除多个对象
预请求跨域配置
恢复归档对象
访问控制
设置对象ACL
查询对象ACL
分块上传
初始化分块上传
上传分块
复制分块
完成分块上传
终止分块上传
查询分块上传
查询已上传块
常见问题
一般性问题
计费计量
API与SDK
工具问题
COSCMD工具
Hadoop工具
COSBrowser工具
COS Migration工具
上传与下载
数据安全
签名算法
词汇表
API文档

产品简介
产品介绍

最近更新时间: 2025-01-15 17:01:00

COS 简介

对象存储（Cloud Object Storage，简称：COS）是云平台提供的一种存储海量文件的分布式存储服务，用户可通过网络随时存储和查看数据。

- COS 使所有用户都能使用具备高扩展性、低成本、可靠和安全的数据存储服务。
- COS 通过控制台、API、SDK 等多样化方式简单、快速地接入，实现了海量数据存储和管理。
- COS 提供了直观的 Web 管理界面便于用户进行文件的上传、下载和管理等操作。

对象存储特点

对象存储是云存储中一种无层次结构的数据存储方法，其不使用目录树结构，各个单独的数据（对象）单元存在于存储池中的同一级别。每个对象都有唯一的识别名称，可用于列出检索操作，另外每个对象还可包含元数据。其对比传统文件存储方式，具有如下特点：

- 数据作为单独的对象进行存储。
- 数据并不放置在目录层次结构中，而是存在于平面地址空间内。
- 应用通过唯一地址来识别每个单独的数据对象。
- 专为使用 HTTP RESTful API 在应用级别进行访问而设计。

功能概览

最近更新时间: 2025-01-15 17:01:00

在使用 COS 之前，建议先阅读 COS 基本概念，了解使用 COS 需要的一些基本概念：存储桶、对象、地域以及访问域名的定义等。

COS 主要提供以下功能：

功能分类	功能	说明
操作	存储桶操作	支持创建、查询、删除、清空存储桶。
操作	对象操作	多种存储类型：根据访问频度的高低和容灾程度高低，COS 提供多种对象的存储类型，包括标准存储、低频存储。 对象/文件夹：上传、查询、下载、复制和删除操作。
数据管理	生命周期	对象存储 COS 支持用户设定规则，对指定对象在某个时间（天数）后进行自动删除。
数据管理	静态网站	将存储桶配置成静态网站托管模式，并通过存储桶域名访问该静态网站，详情请参见托管静态网站。
数据管理	存储桶标签	存储桶标签可以作为管理存储桶的一个标识，便于用户对存储桶进行分组管理。用户可以对指定的存储桶进行标签的设定、查询和删除操作。
数据管理	日志管理	日志管理功能够记录对于指定源存储桶的详细访问信息，并将这些信息以日志文件的形式保存在指定的存储桶中，以实现对存储桶更好的管理。
数据管理	事件管理	事件通知功能，能够对用户所关心的存储资源操作及时进行消息通知。
数据管理	对象标签	对象标签功能的实现是通过为对象添加一个键值对形式的标识，协助用户分组管理存储桶中的对象。对象标签由标签的键（tagKey）和标签的值（tagValue）与=相连组成，例如 group = IT。用户可以对指定的对象进行标签的设定、查询、删除操作。
异地容灾	版本控制	版本控制用于实现在相同存储桶中存放同一对象的多个版本。用户在为某一存储桶开启版本控制功能后，可以根据版本 ID 检索、删除或还原存放在存储桶中的对象。这有助于恢复被用户误删或应用程序故障而丢失的数据。
异地容灾	存储桶复制	用户可以通过配置存储桶复制规则，在不同存储桶中自动、异步地复制增量对象，实现数据的容灾与备份。
数据安全	防盗链	对象存储 COS 支持防盗链配置，用户可以通过控制台的防盗链功能配置黑/白名单，对数据资源进行安全防护。
访问管理	跨域访问	COS 提供 HTML5 标准中的跨域访问设置，帮助实现跨域访问。针对跨域访问，COS 支持响应 OPTIONS 请求，并根据开发者设定的规则向浏览器返回具体设置的规则。
访问管理	存储桶策略	用户可以为存储桶添加策略，可实现允许或禁止某个账号、某个来源 IP（或 IP 段）访问 COS 资源。
访问管理	访问控制	用户可以对存储桶和对象的访问权限进行管理，当收到某个资源的请求时，COS 将检查相应的 ACL 以验证请求者是否拥有所需的访问权限。
数据监控	监控与告警	对象存储 COS 的读写请求量、流量等数据是基于 云监控 来进行统计和展示的。用户可以在云监控的 控制台 查看到 COS 的读写请求量、流量等详细的监控数据。
数据监控	查看数据概览	对象存储 COS 提供存储数据的监控能力，您可通过监控数据窗口按照不同时间段查询不同存储类型数据的数据量及趋势。
工具	多种管理工具	COS 提供 COSBrowser、COSCMD、COS Migration 等多种实用工具，可方便用户进行数据管理或数据迁移。

版权所有：腾讯云金融专区

功能分类	功能	说明
API/SDK	多种 API 和 SDK	API：COS 提供丰富的 API 接口，包括功能接口的使用方法和参数，提供请求示例、响应示例以及错误码介绍。 COS 提供多种开发语言：Android、C、C++、.NET、Go、iOS、Java、JavaScript、Node.js、PHP、Python、小程序 SDK。

规格与限制

最近更新时间: 2025-01-15 17:01:00

分类	规格与限制	详细描述
存储桶	限制	1. 存储桶一旦创建成功，名称和所处地域不能修改 2. 同一用户账号下所有存储桶名称唯一且不支持重命名 3. 名称只支持小写字母和数字[a-z，0-9]、中划线"-"及其组合，支持1 - 50个字符
存储桶	存储桶数量	每个主账户最大200个（默认）
存储桶	对象数量	每个存储桶中，对象数不限
对象	限制	对象键长度支持1 - 850B
对象	上传	1. 控制台上传单个对象最大 512GB 2. API/SDK 上传单个对象最大 48.82TB (50,000GB） 上传接口规格： （a）简单上传：单个对象最大 5GB （b）分块上传：单个对象最大 48.82TB，块大小 1MB - 5GB，最后一个块可小于 1MB，分块数 1 - 10000
对象	复制	1. 支持单个账号在相同地域或跨地域进行对象复制 2. 同地域进行对象复制免费，跨地域进行对象复制会产生流量费用 3. 复制接口规格： （a）简单复制：复制单个对象最大 5GB （b）大于 5GB 必须用分块复制，复制单个对象最大 48.82TB
对象	批量删除	通过 API、SDK 发起批量删除，每次最多删除 1000 个对象
访问策略	规则数量	每个主账号（即同一个 APPID），存储桶和对象的 ACL、Policy 和 CAM 关联的策略数量总和最多为 1000 条
SDK种类	SDK种类	12种： Andriod、C、C++、.NET、Go、iOS、Java、JavaScript、Node.js、PHP、Python、小程序 SDK

购买指南
计费概述

最近更新时间: 2025-01-15 17:01:00

计费方式

按量计费（后付费）

按量计费为 COS 默认计费方式，支持所有地域。根据用户的存储容量、请求、流量等计量项的具体用量进行计费，对用户账户按日或者按月进行扣费结算。

计费项

COS 的费用由三部分组成：存储费用、请求费用、流量费用。

计费项	计费项说明	计费公式
COS 标准存储存储容量	根据存储容量的大小进行计算，不同存储类型的单价不同	存储容量费用 = 存储容量单价 * 日存储容量
COS 标准存储读请求	请求费用根据请求次数进行计算，不同存储类型的请求单价不同	请求费用 = 每万次请求单价 * 日累计请求次数 / 10000
COS 标准存储写请求	请求费用根据请求次数进行计算，不同存储类型的请求单价不同	请求费用 = 每万次请求单价 * 日累计请求次数 / 10000
COS 低频存储存储容量	根据存储容量的大小进行计算，不同存储类型的单价不同	存储容量费用 = 存储容量单价 * 日存储容量
COS 低频存储读请求	请求费用根据请求次数进行计算，不同存储类型的请求单价不同	请求费用 = 每万次请求单价 * 日累计请求次数 / 10000
COS 低频存储写请求	请求费用根据请求次数进行计算，不同存储类型的请求单价不同	请求费用 = 每万次请求单价 * 日累计请求次数 / 10000
COS 外网下行流量	外网下行流量	流量费用 = 每 GB 单价 * 日累计流量
COS 跨区域复制流量	跨区域复制流量	流量费用 = 每 GB 单价 * 日累计流量

计费周期

对象存储 COS 各项计费项的计费周期和计费顺序说明如下：

计费项	计费周期	计费周期说明
COS 标准存储存储容量	日	每日对上一日产生的费用进行结算，输出账单
COS 标准存储读请求	日	每日对上一日产生的费用进行结算，输出账单
COS 标准存储写请求	日	每日对上一日产生的费用进行结算，输出账单
COS 低频存储存储容量	日	每日对上一日产生的费用进行结算，输出账单
COS 低频存储读请求	日	每日对上一日产生的费用进行结算，输出账单
COS 低频存储写请求	日	每日对上一日产生的费用进行结算，输出账单
COS 外网下行流量	日	每日对上一日产生的费用进行结算，输出账单
COS 跨区域复制流量	日	每日对上一日产生的费用进行结算，输出账单

版权所有：腾讯云金融专区

购买指引

最近更新时间: 2025-01-15 17:01:00

对象存储 COS 支持按量计费的后付费方式。

按量计费（后付费）方式：

当您开始使用 COS 服务时，COS 默认使用按量计费方式进行计费，您无须主动购买。

欠费说明

最近更新时间: 2025-01-15 17:01:00

账单周期

下表是对象存储计费项和对应的计费（账单）周期介绍。

计费项	计费周期
COS 标准存储存储容量	日
COS 标准存储读请求	日
COS 标准存储写请求	日
COS 低频存储存储容量	日
COS 低频存储读请求	日
COS 低频存储写请求	日
COS 外网下行流量	日
COS 跨区域复制流量	日

欠费停服

当您的账户发生欠费时，对象存储 COS 会在24小时后停止服务，您的数据可以继续保留。

请在收到欠费通知后，及时前往**计费管理 > 资金管理 > 账户充值**进行充值，以免影响您的业务。

如您对消费明细有疑问，可以在**计费管理 > 账单管理 > 账单明细**页面查阅和核对您的消费明细。

注意：

1. 欠费停服后，**数据占用的存储容量会持续计费**，直到销毁数据。
2. 销毁数据后，不可恢复。
3. 用户续费使账户余额大于等于0后，服务会自动开启。
4. 存储在 COS 的数据若不再使用，请及时删除，以免继续扣费。

查看消费明细

最近更新时间: 2025-01-15 17:01:00

您可以在控制台的【计费管理】中查看您的账户使用对象存储服务所产生的费用信息。

操作步骤

1. 登录计费管理控制台。
2. 在左侧导航树中选择**账单管理 > 账单明细**，进入**资源ID账单**页面。
3. 选择 COS 产品，可按照地域、计费模式和交易类型等维度查看您的 COS 消费情况。

   勾选不显示0元费用，账单中心将自动隐藏0元的消费明细账单。
4. 在账单页面右上方，单击“下载”按钮，可下载您的月度 COS 消费明细。

5. 单击**明细账单**页签，选择COS产品，即可在线查看COS的明细账单。

快速入门
基本概念

最近更新时间: 2025-01-15 17:01:00

想要充分利用 COS，您需要了解一些相关的基本概念和术语。

存储桶

存储桶即 Bucket ，在 COS 中用于存储对象。一个存储桶中可以存储多个对象。存储桶名由用户自定义的字符串和系统自动生成的数字串用中划线链接而成，以保证该存储桶全球唯一。更多详细信息，请参阅 [存储桶管理]。

对象

对象即 Object，COS 中存储的基本单元。更多详细信息，请参阅 [对象管理]。

APPID

APPID 是云平台账户的账户标识之一，用于关联云资源。在用户成功申请云平台账户后，系统自动为用户分配一个 APPID。可通过 云控制台 的【账号信息】查看 APPID。

API 密钥

是用户访问云平台 API 进行身份验证时需要用到的安全凭证，由 SecretId 和 SecretKey 一起组成。一个用户帐号可以创建多个云 API 密钥，若用户还没有云 API 密钥，则需要在 [云 API 密钥控制台] 新建，否则就无法调用云 API 接口。

SecretId

云 API 密钥的一部分，SecretId 用于标识 API 调用者身份。

SecretKey

云 API 密钥的一部分，SecretKey 是用于加密签名字符串和服务器端验证签名字符串的密钥。

默认访问地址

默认访问地址由存储桶名、COS 所属地标识和对象名组成，通过默认访问地址可寻址 COS 中唯一对应的对象。在用户上传对象后，云平台会自动为对象创建默认访问地址。

创建存储桶

最近更新时间: 2025-01-15 17:01:00

存储桶的相关说明，请参考基本概念 。您可以通过对象存储控制台，在存储桶列表页面创建存储桶：

操作步骤

1. 登录对象存储控制台。
2. 在左侧导航树中单击**存储桶列表**，进入存储桶列表页面。
3. 单击**创建存储桶**。

4. 填写存储桶名称、所属地域等信息，单击**确定**。

   建议选择您的用户近的可用地域。访问权限默认为**私有读写**。

上传对象

最近更新时间: 2025-01-15 17:01:00

存储桶创建成功后，您可以上传对象至存储桶。

上传步骤

1. 进入文件列表

1. 登录对象存储控制台。

2. 在左侧导航树中，单击**存储桶列表**，进入存储桶列表。

3. 选择需要存储对象的存储桶，进入存储桶的文件列表页面。

4. 在文件列表页面，单击**上传文件**。

2. 选择上传对象

1. 在**上传文件**对话框中，单击**选择文件**或**选择文件夹**，可上传单个或多个本地文件/文件夹。
2. 选择本地待上传对象。
3. 单击**上传**快速开始上传文件，或单击**下一步**设置对象属性。

3. 设置对象属性（可选）

设置好待上传文件的访问权限，元数据信息（可选），单击**上传**。

访问权限默认继承权限，元数据信息可暂不填写。

对象上传成功后，系统会自动刷新列表，获取最新对象信息。

> **注意：**
>
> 部分浏览器不支持多文件上传，建议使用 IE10 以上、Firefox、Chrome、QQ 浏览器等主流浏览器。

下载对象

最近更新时间: 2025-01-15 17:01:00

已经上传到存储桶中的对象，可通过访问地址进行下载或访问。

一、查看对象信息

1. 登录对象存储控制台，选择**存储桶列表**，单击相应存储桶（如 example-1253833564），进入存储桶的文件列表。
2. 在文件列表中找到需要下载的对象（如 example.exe），单击**详情**，进入文件信息基本信息页面。

二、获取对象链接并下载

在文件信息基本信息页面中，可以查看文件链接。您可以点击下载图标直接下载；或点击复制按钮复制链接，粘贴至浏览器地址栏访问下载。 若对象所属存储桶的属性为私有读写，此处复制的地址后会自动计算签名添加后缀。

删除对象

最近更新时间: 2025-01-15 17:01:00

当您需要删除存储桶里的对象时,请参考以下步骤：

1. 登录对象存储控制台。
2. 选择左侧菜单栏**存储桶列表**，进入存储桶列表页面。
3. 单击待删除对象所在的存储桶，进入存储桶文件列表。
4. 选择想要删除的对象，单击**更多操作 > 删除**。
5. 弹出删除文件对话框，单击**确认**即可删除对象。

删除存储桶

最近更新时间: 2025-01-15 17:01:00

当您不再需要使用某个存储桶时，可以对其进行删除操作。

1. 在对象存储控制台找到您想要删除的存储桶，单击**删除**，弹出删除存储桶对话框。
2. 单击**确定**即可删除存储桶。

> **注意：**
>
> 删除存储桶时，需保证其中没有任何对象、目录，否则将无法删除。

后续步骤

最近更新时间: 2025-01-15 17:01:00

通过前面的步骤，可了解如何通过云平台对象存储控制台执行 COS 的部分基本任务。

通过使用对象存储控制台，可直接执行大部分基本任务。除此之外，还可通过调用 RESTful API，使用针对主流语言的 SDK 工具包等方式完成 COS 的更多基本和高级任务。

控制台指南

控制台概述

最近更新时间: 2025-01-15 17:01:00

简介

对象存储控制台是用户使用 COS 的工具之一。用户无需编写代码或运行程序，使用 COS 控制台即可进行存储桶管理、对象管理等操作。对象存储控制台提供以下多种功能，对应的操作指南如下表所示：

控制台菜单	功能
存储桶基础操作	• 创建存储桶 • 删除存储桶 • 查询存储桶 • 设置对象锁定
对象基础操作	• 上传对象 • 下载对象 • 复制对象 • 查看对象信息 • 搜索对象 • 排序或筛选对象 • 文件重命名 • 修改存储类型 • 自定义 Headers • 删除对象 • 还原历史版本对象
文件夹操作	• 创建文件夹 • 删除文件夹
生命周期	设置生命周期
标签管理	• 设置存储桶标签 • 设置对象标签
日志管理	设置日志管理
事件通知	设置事件通知
版本控制	设置版本控制
跨地域复制	设置跨地域复制
防盗链	设置防盗链
跨域访问	设置跨域访问
域名管理	用户自定义域名
存储桶策略	添加存储桶策略
访问控制	• 设置存储桶的访问权限 • 设置对象的访问权限
监控报表	• 基础数据统计 • 返回码统计

存储桶管理
创建与删除

最近更新时间: 2025-01-15 17:01:00

操作步骤

相关信息说明

名称

- 存储桶名称由自定义字符串和系统生成数字串（APPID）组成，两者由中划线 - 连接。用户只填写自定义字符串部分即可。对同一用户而言，不同存储桶的系统生成数字串（APPID）是相同的。
- 存储桶名称中，自定义字符串长度不能超过 40 字符。 仅支持小写字母、数字、中划线及其组合，不支持特殊符号及下划线。
- 同一用户创建的存储桶名称唯一且不支持重命名。

访问权限

存储桶默认提供三种访问权限：私有读写、公有读私有写和公有读写。存储桶权限可通过对象存储控制台存储桶的权限管理修改。

- 私有读写：只有该存储桶的创建者及有授权的账号才对该存储桶中的对象有读写权限，其他任何人对该存储桶中的对象都没有读写权限。推荐使用。
- 公有读私有写：任何人（包括匿名访问者）都对该存储桶中的对象有读权限，但只有存储桶创建者及有授权的账号才对该存储桶中的对象有写权限。
- 公有读写：任何人（包括匿名访问者）都对该存储桶中的对象有读权限和写权限，不推荐使用。

所属项目

- 支持在创建流程中选择"项目"字段，项目能够帮助您解决单个云账号内的资源分组和授权管理的复杂性问题。
- 通过**资源管理 > 项目管理**进行项目的创建。

创建存储桶

您可以通过对象存储控制台的存储桶列表创建存储桶。

1. 进入存储桶列表

登录对象存储控制台，点击左侧导航**存储桶列表**，进入存储桶列表。单击**创建存储桶**，弹出创建存储桶对话框。

2. 填写存储桶信息

请填写存储桶名称（如 test-1253833564），创建存储桶所属地域，限默认私有读写，单击**确定**即可快速创建一个存储桶。

删除存储桶

当您不再需要使用某个存储桶时，可以对其进行删除操作。

1. 在对象存储控制台找到您想要删除的存储桶，单击**删除**。
2. 弹出删除存储桶对话框，单击**确定**即可删除存储桶。

> 注意：
>
> 删除存储桶时，需保证其中没有任何对象、目录，否则将无法删除。

浏览与查询

最近更新时间: 2025-01-15 17:01:00

浏览存储桶

当您需要浏览自己所拥有的存储桶时，在对象存储控制台，单击左侧菜单栏**存储桶列表**，进入存储桶列表页，即可浏览所有存储桶。当您需要有关存储桶的更多详细信息时，可直接单击存储桶名称，进入存储桶查看。

查询存储桶

当存储桶数量较多，您需要迅速找到特定的存储桶时，可以通过控制台存储桶列表页面右侧的搜索框：

- 按照存储桶名称查询：可输入存储桶的名称进行查询，支持存储桶名称的前缀匹配查询。

- 按照标签查询：若您已为存储桶设置了标签（详见 设置存储桶标签），您也可以在存储桶列表页面右上角的搜索区，选择按标签查询，可输入标签键进行查询。
- 按照所属项目查询：若您已为存储桶设置了所属项目，您也可以在存储桶列表页面右上角的搜索区，选择所属项目，输入所属项目进行查询。

存储桶监控

当您想要知道自己存储桶内使用详情数据时，点击右侧**监控**即可快速查看。

存储桶列表导出

进入存储桶列表页，右上角**下载**按钮，可导出所有存储桶信息。

设置访问权限

最近更新时间: 2025-01-15 17:01:00

简介

用户可以通过控制台和 API 来修改存储桶访问权限。对象存储控制台默认支持三种访问权限：私有读写、公有读私有写和公有读写。

- 私有读写：只有该存储桶的创建者及有授权的账号才对该存储桶中的对象有读写权限，其他任何人对该存储桶中的对象都没有读写权限。存储桶访问权限默认为私有读写，推荐使用。
- 公有读私有写：任何人（包括匿名访问者）都对该存储桶中的对象有读权限，但只有存储桶创建者及有授权的账号才对该存储桶中的对象有写权限。
- 公有读写：任何人（包括匿名访问者）都对该存储桶中的对象有读权限和写权限，不推荐使用。

设置步骤

用户在创建存储桶时可以选择存储桶的访问权限。除此之外，可通过存储桶基础配置修改存储桶访问权限，具体步骤如下：

1. 登录对象存储控制台。

2. 在左侧导航树中，单击**存储桶列表**，进入存储桶列表页面。

3. 单击需要修改访问权限的存储桶名称，进入存储桶配置页面。

4. 单击**权限管理 > 存储桶访问权限**，可对存储桶的公共权限和用户权限进行设置。

5. 更改后单击**保存**即可。

设置防盗链

最近更新时间: 2025-01-15 17:01:00

简介

为了避免恶意程序使用资源 URL 盗刷公网流量或使用恶意手法盗用资源，给用户带来不必要的损失。对象存储支持防盗链配置，建议您通过控制台的防盗链设置配置黑/白名单，来进行安全防护。

设置步骤

1. 登录对象存储控制台。

2. 在左侧导航栏中，选择**存储桶列表**，进入存储桶列表页。

3. 单击需要设置防盗链的存储桶名称，进入存储桶的文件列表页面。

4. 选择**安全管理 > 防盗链设置**，单击**编辑**进入可编辑状态。

5. 修改当前状态为开启，选择名单类型（黑名单或白名单），设置好相应域名，设置完成单击**保存**即可，配置项说明如下：

- **黑名单**：**限制名单内的域名**访问存储桶的默认访问地址，若名单内的域名访问存储桶的默认访问地址，则返回403。
- **白名单**：**限制名单外的域名**访问存储桶的默认访问地址，若名单外的域名访问存储桶的默认访问地址，则返回403。
- **空 referer**：HTTP 请求中，header 为空 referer（即不带 referer 字段或 referer 字段为空）。
- **Referer**：支持设置最多10条域名且为相同前缀匹配，每条一行，多条请换行；支持域名、IP 和通配符 * 等形式的地址。示例如下：
  - 配置 www.example.com ：可限制如 www.example.com/123 、 www.example.com.cn 等以 www.example.com 为前缀的地址。
  - 支持带端口的域名和 IP，例如 www.example.com:8080 、 10.10.10.10:8080 等地址。
  - 配置 *.example.com ：可限制 a.b.example.com/123 、 a.example.com 等地址。

示例

APPID 为 1250000000 的用户创建了一个名为 examplebucket-1250000000 的存储桶，并在根目录下放置了一张图片 picture.jpg，COS 根据规则生成了一个默认访问地址 ：

```
examplebucket-1250000000.file.myqcloud.com/picture.jpg
```

用户 A 拥有网站：

```
www.example.com
```

并将该图片嵌入了首页 index.html 中。

此时站长 B 持有网站：

```
www.fake.com
```

站长 B 想把这张图片放入 www.fake.com 中。由于不想付流量费用，他便直接通过以下地址引用了 picture.jpg，并放置到 www.fake.com 网站首页 index.html。

```
examplebucket-1250000000.file.myqcloud.com/picture.jpg
```

为了避免用户 A 的损失，针对以上状况，我们提供两种开启防盗链的方式。

开启方式一

配置**黑名单**模式，域名设置填入 *.fake.com 并保存生效。

开启方式二

配置**白名单**模式，域名设置填入 *.example.com 并保存生效。

开启前

访问 http://imgcache.finance.cloud.tencent.com:80www.example.com/index.html 图片显示正常。 访问 http://imgcache.finance.cloud.tencent.com:80www.fake.com/index.html 图片也显示正常。

开启后

访问 http://imgcache.finance.cloud.tencent.com:80www.example.com/index.html 图片显示正常。 访问 http://imgcache.finance.cloud.tencent.com:80www.fake.com/index.html 图片无法显示。

设置对象锁定

最近更新时间: 2025-01-15 17:01:00

简介

您可以通过控制台开启对象锁定功能，以确保在设定期限内对象不能被改写或者删除，且可以立即访问。

操作步骤

1. 登录对象存储控制台。
2. 在左侧导航栏中，单击**存储桶列表**，进入存储桶列表页面。
3. 单击需要设置对象锁定的存储桶，进入文件列表页面。
4. 在左侧导航栏中，选择**安全管理 > 对象锁定**，单击**编辑**，将当前状态修改为“开启”。
5. 在配置窗口中输入锁定时长，单击**保存**。

填写正整数，对象锁定时长无法缩短，请合理配置。

设置生命周期

最近更新时间: 2025-01-15 17:01:00

简介

当您需要定期对指定对象进行删除以降低成本时，您可以使用生命周期管理功能。对象存储 COS 会按照您设定的规则对指定对象在指定的时间内自动进行删除。如需了解该功能的更多信息，请参见生命周期概述文档。

> **说明：**
>
> 生命周期的设置支持最长天数为3650天。

操作步骤

1. 登录对象存储控制台。

2. 在左侧导航中，单击**存储桶列表**，进入存储桶列表页面。

3. 找到需要开启生命周期功能的存储桶，单击其存储桶名称，进入存储桶文件列表页面。

4. 选择**基础配置 > 生命周期管理**，单击**添加规则**。

5. 根据您的需求添加生命周期规则。

   配置项说明如下：

   - **规则名称**：输入您的生命周期规则名称。
   - **应用范围**：本生命周期规则可以作用于整个存储桶，也可以作用于指定范围的对象，当前支持选择以下范围：

> **注意：**
>
> 可同时指定对象前缀和对象标签。
>
> 对象键（或前缀）的相关信息，请参见对象概述中的 对象键。有关生命周期的配置规则说明，请参见 规则描述 文档。

```
- **对象前缀**：可指定具有相同文件前缀的对象去执行生命周期规则，例如 prefix/。
- **对象标签**：可指定带有同一标签的对象去执行生命周期规则，支持指定多个标签，请区分英文字母大小写。
```

- **管理当前版本文件**：您可以通过开启管理当前版本对象的选项，设置文件沉降或者删除当前版本对象。支持存储桶中的对象由标准存储沉降至低频存储，支持对象到期后删除。时间是以文件在对象存储上的修改时间为标准开始计算，修改对象的行为等同于重新上传对象。
- **管理历史版本文件**：您可以通过开启管理历史版本对象的选项，设置文件沉降或者删除历史版本对象。若您未开启该选项，我们将默认仅处理最新版本的文件。
- **清理无历史版本的删除标记**：如果对象的最新版本是删除标记（Delete Marker）且该对象的历史版本均已被删除，开启此选项后该删除标记（Delete Marker）也将被删除。该选项不能与管理当前版本文件中的到期删除同时开启。
- **删除碎片**：文件上传的时候由于各种原因导致上传失败，只传输了其中的一部分，对于此类残损的文件可以设置定期删除。

6. 配置好生命周期规则后，单击**确定**，您即可看到生命周期规则。

7. 当需要停止生命周期规则时，单击**编辑**，只需要将对应规则的状态修改为**关闭**或者单击**删除**，直接删除生命周期规则。

设置跨域访问

最近更新时间: 2025-01-15 17:01:00

简介

您可以通过对象存储控制台，对存储桶中的对象设置跨域访问。COS 提供了响应 OPTIONS 请求的配置，支持多条规则。跨域访问即通过 HTTP 请求，从一个域去请求另一个域的资源。只要协议、域名、端口有任何一个不相同，都会被当作是不同的域。

对象存储服务针对跨域访问，支持响应 OPTIONS 请求 ，并根据开发者设定的规则向浏览器返回具体设置的规则。但服务端并不会校验随后发起的跨域请求是否符合规则。更多详细资料请参阅关于 HTTP 访问控制的说明。

操作步骤

1. 登录对象存储桶控制台。

2. 在左侧菜单栏中，选择**存储桶列表**，进入存储桶列表页面。

3. 单击需要设置跨域访问的存储桶名称，进入存储桶文件列表页面。

4. 选择**安全管理 > 跨域访问CORS设置**，单击**添加规则**。

5. 添加规则信息（带 * 号的为必填项），配置项说明如下：

   - **来源 Origin**：允许跨域请求的来源。
     - 可以同时指定多个来源，每行只能填写一个。
     - 配置支持 * ，表示全部域名都允许，不推荐。
     - 支持单个具体域名，形如 http://imgcache.finance.cloud.tencent.com:80www.abc.com 。
     - 支持二级泛域名，形如 http://imgcache.finance.cloud.tencent.com:80*.abc.com ，但是每行只能有一个 * 号。
     - 注意不要遗漏协议名 http 或 https，若端口不是默认的80，还需要带上端口。
   - **操作 Methods**：支持 GET、PUT、POST、DELETE、HEAD。枚举允许一个或多个跨域请求方法。
   - **Allow-Headers**：在发送 OPTIONS 请求时告知服务端，接下来的请求可以使用哪些自定义的 HTTP 请求头部，例如：x-cos-meta-md5。
     - 可以同时指定多个 Headers，每行只能填写一个。
     - Header 容易遗漏，没有特殊需求的情况下，建议设置为 * ，表示允许所有。
     - 支持英文大小写[a-z,A-Z]，不允许带有下划线 _ 。
     - 在 Access-Control-Request-Headers 中指定的每个 Header，都必须在 Allowed-Header 中有对应项。
   - **Expose-Headers**：Expose-Header 里返回的是 COS 的常用Header，详情请查阅公共请求头部。具体的配置需要根据应用的需求确定，默认推荐填写 Etag。不允许使用通配符，大小写不敏感，支持多行且每行只能填写一个。
   - **超时 Max-Age**：设置 OPTIONS 请求得到结果的有效期（秒）。数值必须为正整数，例如：600

6. 设置完成后，单击**确定**。

   此时您可以看到跨域访问规则已添加完成。如需修改，可单击**修改**按钮进行设置。

设置静态网站

最近更新时间: 2025-01-15 17:01:00

简介

您可以通过对象存储控制台，把一个存储桶设置为托管静态网站，并且通过访问存储桶的静态网站域名来访问静态网站。关于静态网站的相关说明，请参见 托管静态网站。

> **注意：**
>
> - 使用存储桶托管静态网站，您首先需要把存储桶的访问权限设置为**公有读私有写**。
> - 开启静态网站配置后，您需要使用静态网站域名访问 COS 源站才能生效，如果使用 COS 默认域名访问则无静态网站效果。

前提条件

已创建存储桶，详情请参见创建存储桶文档。

操作步骤

1. 登录对象存储控制台，单击左侧**存储桶列表**菜单栏，单击需要用来托管静态网站的存储桶，进入存储桶详情界面。

2. 选择**权限管理 > 存储桶访问权限**，在**公共权限**一栏中，选择**公有读私有写**并保存。

3. 选择**基础配置 > 静态网站**，单击**编辑**，把**当前状态**的按钮打开，然后依次填写静态网站的配置项。

   配置说明如下：

   - **忽略 html 扩展名（可选）**：访问路径为 index 时，会自动匹配 index.html 对象进行返回。
   - **索引文档（必选）**：索引文档即静态网站的首页，是当用户对网站的根目录或任何子目录发出请求时返回的网页，通常此页面被命名为 index.html。

     > **注意：**
     >
     > 如果存储桶中创建了文件夹，则需要在每个文件夹层级上都添加索引文档。

   - **错误文档（可选）**：错误文档指访问静态网站出错后返回的页面。该配置项方便您自行定义错误文档。当静态网站无法响应用户的请求时，将返回指定的自定义错误页面。例如您配置了命名为 error.html 的错误文档，当用户访问遇到 HTTP 错误时，页面将返回 error.html 页面，为其提供帮助指引。当您未配置错误文档时，此时用户访问遇到 HTTP 错误，页面将返回默认的错误信息。

     > **注意：**
     >
     > 错误文档配置可支持存储桶根目录或子目录下的文件，请使用浏览器可识别的 `.html` 或 `.htm` 等格式的文件。若使用了浏览器不可识别的文件，例如 `.zip` 文件，大部分浏览器将显示错误无法访问或拒绝访问请求。

   - **错误文档响应码**：如有设置错误文档则展示该项。可配置返回错误文档时的 HTTP 响应码为原始错误码或者200。
   - **重定向规则（可选）**：利用重定向规则，您可以根据特定的文件路径、请求中的前缀或者响应代码来按条件重定向请求。 例如，您在存储桶中删除或重命名某个文件。您可以添加一个重定向规则，将访问该文件的请求重定向至其他文件。
     - 错误码：目前重定向规则仅支持对 `4xx` 错误码（例如 404）进行重定向配置。您可以选择性地自定义错误页面，若用户触发了对应的 HTTP 错误，您可以在该错误页面中为您的用户提供其他指引。
     - 前缀匹配：您可以使用前缀匹配规则对存储桶内的文件或文件夹进行重定向设置。具体示例请参见 重定向规则示例。

4. 设置完成后，单击**保存**即可。

添加存储桶策略

最近更新时间: 2025-01-15 17:01:00

简介

您可以通过对象存储控制台，为存储桶添加策略，以允许或禁止某个账号、某个来源 IP（或 IP 段）访问策略所设定的 COS 资源。下面将为您详细介绍如何添加存储桶策略。

> **注意：**
>
> 每个主账号，创建的对象 ACL、存储桶 ACL 和存储桶策略总和最多为1000条。

操作步骤

1. 登录对象存储控制台。
2. 在左侧导航栏中，单击**存储桶列表**。
3. 选择需要添加存储桶策略的存储桶，进入存储桶。
4. 选择**权限管理 > Policy 权限设置**，COS 提供添加存储桶策略的方式为**图形设置**和**策略语法**，您可以选其中一种方式添加存储桶策略。
   - **图形设置** 设置示例如下：
   - **策略语法** 单击**编辑**，输入您所定义的策略语法。
5. 确认配置信息无误后，单击**保存**即可。此时使用子账号登录 COS 控制台，将只能访问策略所设定的资源范围。

设置存储桶标签

最近更新时间: 2025-01-15 17:01:00

简介

存储桶标签是一个键值对（key = value），由标签的键（key）和标签的值（value）与"="相连组成，例如 group = IT。它可以作为管理存储桶的一个标识，便于用户对存储桶进行分组管理。您可以通过控制台对指定的存储桶进行标签的设定、查询和删除操作。

> **注意：**
>
> - 同个存储桶下最多支持50个标签，且标签键不能重复。
> - 标签键和标签值不得使用qcs:、project、项目保留字段，更多限制请参见存储桶标签概述。

在新创建存储桶时添加标签

您可以在创建存储桶时添加存储桶标签，如下图所示：

在已创建存储桶中添加标签

若您在创建存储桶时未添加标签，您可以按照下述步骤为存储桶添加标签。

1. 在存储桶列表页，找到您需要添加标签的存储桶，单击其名称，进入存储桶配置页面。
2. 选择**基础配置** > **标签管理**，单击**添加标签**，添加存储桶标签。

3. 输入标签键和标签值后，单击**保存**即可添加标签。

设置版本控制

最近更新时间: 2025-01-15 17:01:00

简介

启用存储桶版本控制功能，您可以在存储桶中存放对象的多个版本，并且可以对指定版本的对象进行检索、删除或还原。这有助于恢复被用户误删或应用程序故障而丢失的数据。

操作步骤

1. 登录对象存储控制台。
2. 在左侧导航栏中，选择**存储桶列表**，进入存储桶列表页面。
3. 单击需要配置版本控制的存储桶名称，进入存储桶的文件列表页面。
4. 选择**容错容灾管理 > 版本控制**，单击**编辑**进入修改状态。
5. 打开**当前状态**的**开启**按钮并保存。
6. 在弹出对话框中，单击**确定**即可开启版本控制。

   当您不需要使用版本控制功能时，将当前状态修改为**关闭**即可暂停。
7. 开启版本控制后，若您将同名文件上传到该存储桶，那么在文件列表界面中，单击**列出历史版本**，您可查看到在不同时间点所上传的同名文件。并且您可对最新版本、历史版本以及删除标记版本进行多种操作。

   历史版本文件，支持**下载**、**详情**、**删除**操作。

设置跨地域复制

最近更新时间: 2025-01-15 17:01:00

简介

启用跨地域复制功能后，您可以将源存储桶中的**增量对象**进行自动、异步地复制到另一地域的目标存储桶中。您对源存储桶中的对象进行管理操作时（例如新增对象、删除对象），COS 将自动将这些操作复制到目标存储桶中。您可以根据实际需要，选择启用或关闭跨地域复制，更多信息请参见跨地域复制概述。

前提条件

开启跨地域复制功能前，请先确保源存储桶和目标存储桶均处于**不同地域**，且均已开启版本控制功能。

启用跨地域复制

1. 登录对象存储控制台。
2. 在左侧导航栏中，选择**存储桶列表**，进入存储桶列表页面。
3. 单击需要配置跨地域复制的源存储桶名称，进入存储桶的文件列表页面。
4. 选择**容错容灾管理** > **跨地域复制**，单击**新增规则**，进入跨地域复制规则配置状态。
5. 配置跨地域复制规则要求源存储桶和目标存储桶均开启版本控制规则。如果源存储桶和目标存储桶均未开启版本控制功能，请先开启版本控制功能后，再配置跨地域复制规则。
6. 规则配置完成后，单击**确定**。

   配置项说明如下：

   - **源地域**：您的源存储桶所属地域。
   - **应用范围**：指需要复制的源存储桶中的对象范围，如不设置则默认复制存储桶中的所有对象，假如指定前缀则复制具有特定前缀的内容（例如复制具有 `logs/` 前缀的文件 ，则填 `logs/` ）。
   - **资源路径**：您的源存储桶路径。
   - **目标存储桶**：指对象被复制后所存放的存储桶，其所属地域不能与源存储桶的地域相同，不支持选择非当前账号的存储桶。
   - **目标存储类型**：指对象被复制到目标存储桶后的存储类型，默认跟随源存储桶中的对象类型，您可选择其他存储类型以改变目标存储桶中被复制对象的类型。目前支持标准存储和低频存储两种存储类型。

> **注意**：
>
> - 当您完成配置规则后，可对规则进行管理操作。您可单击按钮启用或禁用当前规则，单击编辑按钮可修改当前规则。
> - 如果您在首次设置跨地域复制规则中将应用范围设置为复制存储桶内的所有内容，那么您将无法新增任何规则。您可以通过编辑当前规则，或删除当前规则后重新添加的方式进行修改。
> - 如果您在首次设置跨地域复制规则中将应用范围设置为某项前缀的内容，那么您仍然可以通过编辑当前规则，将应用范围修改为存储桶内的全部内容。

关闭跨地域复制

您可通过两种方式关闭跨地域复制功能：通过关闭状态按钮和删除规则。

- **通过关闭状态按钮**：通过关闭某个规则中的状态按钮暂时禁用当前规则，这一操作将暂停当前跨地域复制功能，已复制数据将被保留在目标存储桶中，源存储桶中的增量数据将不再复制到目标存储桶，如需再次使用，单击开启按钮即可。

- **删除规则**：在**跨地域复制**管理项里删除已添加的规则，删除规则后已配置的跨地域复制规则将失效，已复制数据将被保留在目标存储桶中，源存储桶中的增量数据将不再进行跨地域复制操作，如需再次使用，您需要重新添加规则。

> 注意：
>
> - 尚未完成的跨地域复制操作，将在关闭跨地域复制时中止，并将无法继续执行。
> - 对存储桶再次启用跨地域复制时，仅对开启完成后新增的对象执行跨地域复制操作。

设置存储桶加密

最近更新时间: 2025-01-15 17:01:00

简介

通过对象存储控制台，对存储桶设置服务端加密，可以实现对新上传到该存储桶的对象默认进行加密。

目前存储桶的加密方式支持 SSE-COS 加密（即由 COS 托管密钥的服务端加密）以及SSE-KMS加密（即由KMS托管密钥的服务端加密）。

操作步骤

创建存储桶时设置加密

在创建存储桶时，可以按照下述步骤为存储桶设置加密。

1. 在存储桶列表中，点击**创建存储桶**，填入名称、地域等基础设置。

2. 在**服务端加密**选项中，选择**SSE-COS**或**SSE-KMS**。
   - 选择SSE-COS时，加密算法将显示AES256，如果启用了SM4算法加密功能，加密算法还将显示SM4。
   - 选择SSE-KMS时，加密算法的显示同上，密钥可以选择**默认密钥**和**已有自定义密钥**。

3. 点击**确定**。

在已创建存储桶中设置加密

若您在创建存储桶时未设置加密，您可以按照下述步骤为存储桶设置加密。

1. 在存储桶列表页，找到您需要设置加密的存储桶，单击其名称，进入存储桶配置页面。

2. 选择**安全管理 > 服务端加密**，单击**编辑**，可以修改当前存储桶的加密属性：

   不加密：

   SSE-COS加密：

   SSE-KMS加密（使用默认密钥）：

   SSE-KMS加密（使用用户自定义密钥）：

   当选择使用用户自定义密钥时，请尽量避免禁用或删除用户自定义密钥，否则可能导致预期之外的误加密，或造成已加密对象无法解密的问题。

关于存储桶加密与上传对象时加密的问题

上传对象时加密有两种方式，一种是通过上传对象时，添加相应header `x-cos-server-side-encryption` 来实现（在控制台上传对象时选择加密方式，即属于这一种），另一种是通过设置存储桶加密来实现。 这两种加密方式的优先级如下所述：

1. 当未设置存储桶加密时，以上传对象时携带的header为主。此时可以是“不加密”，“SSE-COS”，“SSE-KMS”。
2. 当设置了存储桶加密时，若上传对象时未携带相应header（即“不加密”），此时将以存储桶加密为主，可以是“SSE-COS”或“SSE-KMS”。
3. 当设置了存储桶加密时，若上传对象时携带了相应header（即“SSE-COS”或“SSE-KMS”），此时将以header携带为主，无论存储桶加密设置为哪种加密类型。

选择指定的加密方式，然后单击**保存**即可完成存储桶加密配置。

设置自定义域名

最近更新时间: 2025-01-15 17:01:00

简介

用户可以将已备案的自定义域名，绑定至当前存储桶，通过自定义域名访问存储桶内对象。

> **说明：**
>
> 通过 COS 控制台添加自定义域名上限为20个。

操作步骤

1. 登录对象存储控制台 。
2. 在左侧导航栏中，单击**存储桶列表**，进入存储桶列表页面。
3. 单击需要配置域名的存储桶，进入存储桶配置页面。
4. 选择**域名管理**，单击**添加域名。**如果您的自定义域名已经在工信部备案且可CNAME到存储桶默认域名，请直接在**域名**输入框，填写您的自定义域名，并单击**保存。**

如果您还未为您的自定义域名添加解析，请在您的 DNS 厂商处修改 CNAME 信息。

设置日志管理

最近更新时间: 2025-01-15 17:01:00

简介

日志管理功能，可以为您记录跟存储桶操作相关的各种请求日志。开启日志管理功能，可以帮助您更好的管理和使用存储桶。

操作步骤

1. 登录对象存储控制台 。

2. 在左侧导航栏中，单击**存储桶列表**，进入存储桶列表页面。

3. 找到需要开启日志管理的源存储桶，单击该存储桶名称，进入该存储桶管理页面。

4. 选择**日志管理**，单击**编辑**，将当前状态修改为“开启”。

   - **目标存储桶**：开启日志管理的源存储桶和存放日志的目标存储桶必须在同一地域。COS 不推荐将源存储桶设置为存放日志的存储桶。
   - **路径前缀**：输入便于您查找日志的自定义路径前缀。如不填，则默认为目标存储桶的根路径。
   - **服务授权**：您需要授权 CLS 产品服务向您的存储桶中投递访问日志。

5. 确认输入信息无误后，单击**保存**。此时日志管理功能启用完毕。

> **说明：**
>
> 日志生成并投递到目标存储桶可能需要经过数分钟或更长，请您耐心等待。

6. 找到您此前配置日志存放的目标存储桶，即可看到生成的日志文件。
7. 日志文件下载后，可按照字段说明进行查看。

设置事件通知

最近更新时间: 2025-01-15 17:01:00

简介

事件通知功能，能够对用户所关心的存储资源操作及时进行消息通知。

前提条件

已创建ckafka实例，并配置topic。

操作步骤

1. 登录对象存储控制台 。

2. 在左侧导航栏中，单击**存储桶列表**，进入存储桶列表页面。

3. 找到需要设置事件通知的源存储桶，单击该存储桶名称，进入该存储桶管理页面。

4. 选择**基础配置 > 事件通知**，单击**添加事件通知**。

5. 根据您的需求添加事件通知规则。

6. 设置完成后，单击**确定**，您即可看到事件通知规则。 如需修改，可单击**编辑**进行设置。

对象管理
上传对象

最近更新时间: 2025-01-15 17:01:00

简介

您可以通过对象存储控制台，在存储桶的文件列表页面上传对象。

前提条件

上传对象前，请您确保已创建存储桶。

操作步骤

1. 进入文件列表

1. 登录对象存储控制台。
2. 在左侧导航树中，单击**存储桶列表**，进入存储桶列表。
3. 选择需要存储对象的存储桶，进入存储桶的文件列表页面。
4. 选择**文件列表**，单击**上传文件**。

2. 选择上传对象

1. 在**上传文件**对话框中，单击**选择文件**或**选择文件夹**，可上传单个或多个本地文件/文件夹。
2. 选择本地需要上传的对象。
3. 单击**上传**快速开始上传文件，或者单击**下一步**，设置对象属性。

3. 设置对象属性（可选）

设置待上传文件的访问权限、元数据，单击**上传**。配置说明如下：

- 存储类型：根据不同的业务场景，您可以为不同的对象设置不同的存储类型，默认存储类型为标准存储。
- 访问权限：可以为不同的对象设置不同的访问权限，默认访问权限为继承权限（即继承存储桶权限）。
- 对象标签：对象标签由标签的键（tagKey）、标签的值（tagValue）与"="相连组成，例如 group = IT。您可以对指定的对象进行标签的设定、查询、删除操作。
- 元数据：元数据是服务器以 HTTP 协议传 HTML 资料到浏览器前所送出的字串，又称为 HTTP Header。通过修改 HTTP Header，可以改变页面的响应形式，或者传达配置信息，例如修改缓存时间。修改对象的 HTTP Header 不会修改对象本身。

> **注意：**
>
> 部分浏览器不支持多文件上传，建议使用 IE10以上、Firefox、Chrome 等主流浏览器。

4. 确认上传完成

1. 单击**上传**，您可以在右上方的**上传完成**中查看当前的上传进度。
2. 上传完成后，您将可以在**文件列表**里查看到刚才已上传的对象。

下载对象

最近更新时间: 2025-01-15 17:01:00

简介

已经上传到存储桶中的对象，可通过访问地址进行下载或访问。

> **注意：**
>
> 目前对象存储只支持下载单个对象，不支持批量下载对象。

操作步骤

1. 登录对象存储控制台，单击**存储桶列表**，单击相应存储桶名称，进入存储桶的文件列表。

2. 在文件列表中找到需要下载的对象，直接单击**下载**按钮下载对象，或单击右侧的**详情**，跳转文件详情界面获取下载链接（见第3步操作）。

3. 在文件信息详情框中，您可以查看到对象地址。您可以单击**下载对象**直接下载、或单击**复制临时链接**复制链接，粘贴至浏览器地址栏回车即可下载。

> **说明：**
>
> - 若对象所属存储桶的属性为私有读写，此处复制的地址后会自动计算签名添加后缀。
> - 带有签名的临时链接在**查看对象详情**起1个小时内有效，也可通过**刷新有效期**按钮刷新签名的有效期。

复制对象

最近更新时间: 2025-01-15 17:01:00

简介

您可以通过对象存储控制台，对上传到存储桶中的单个或多个对象进行复制，实现将对象从源路径复制到目标路径。

操作步骤

1. 登录对象存储控制台。

2. 在左侧导航栏中，选择**存储桶列表**，进入存储桶列表页。

3. 找到对应的存储桶，单击其存储桶名称，进入存储桶的文件列表页面。

4. 选择想要复制的对象或文件夹，支持多选，单击**更多操作 > 复制**。

5. 提示复制成功后，可选择目标路径进行粘贴。例如粘贴至存储桶**test-1255000220**下的 target 文件夹。

> **注意：**
> 目标路径不能与源路径相同，否则将粘贴失败。

6. 粘贴成功后，即可看到对象或文件夹都被复制到了 target 文件夹。

查看对象信息

最近更新时间: 2025-01-15 17:01:00

简介

通过对象存储控制台可以查看对象的属性信息（如对象大小、对象地址）以及对象相关的配置（如设置对象的访问权限、存储类型更改等）。

操作步骤

1. 登录对象存储控制台，单击**存储桶列表**进入存储桶列表。

2. 单击对象所在的存储桶名称，进入存储桶的**文件列表**页面。

3. 单击对象右侧的**详情**按钮，可以查看对象大小、对象地址以及获取签名链接等相关信息。同时还可以对对象进行相关的配置。

版权所有：腾讯云金融专区

搜索对象

最近更新时间: 2025-01-15 17:01:00

简介

您可以通过对象存储控制台，对已上传的对象进行搜索。

搜索存储桶当前对象

操作步骤

1. 登录对象存储控制台，单击**存储桶列表**进入存储桶列表，单击对象所在的存储桶。

2. 在**文件列表**的右上角搜索框中输入对象的名称前缀，然后单击搜索按钮，即可显示出当前存储桶中带有**相同名称前缀**的对象或文件夹。如需搜索具体的对象，您可以输入完整的对象键，比如 `exampleobject.txt` 。

排序或筛选对象

最近更新时间: 2025-01-15 17:01:00

简介

您可以通过对象存储控制台，在存储桶内的文件列表，对文件进行排序、筛选操作。

> **说明：**
>
> 仅当文件及文件夹总数小于1000时支持排序、筛选操作。

操作步骤

1. 登录对象存储控制台。
2. 在左侧导航栏中，选择**存储桶列表**，进入存储桶列表页。
3. 找到对应的存储桶，单击其存储桶名称，进入存储桶的文件列表页面。
4. 根据实际需求，单击表头进行排序或者筛选。

> **说明：**
>
> 当前支持对文件列表按文件名、大小、修改时间进行排序，按存储类型进行筛选

文件重命名

最近更新时间: 2025-01-15 17:01:00

简介

您可以通过对象存储控制台，对已上传的对象进行重命名。

操作步骤

1. 登录对象存储控制台。
2. 在左侧导航栏中，选择**存储桶列表**，进入存储桶列表页。
3. 单击相应存储桶名称，进入存储桶的文件列表。
4. 光标移至对已上传的对象名中，点击编辑按钮。

> **说明：**
>
> 不能以 '/' 或 '\' 开头且长度在850以内

修改存储类型

最近更新时间: 2025-01-15 17:01:00

简介

您可以通过对象存储控制台，随时对已上传的对象进行存储类型的修改，以满足不同场景的业务需求。

> **说明：**
>
> 不支持对大于5GB的对象进行存储类型的修改。

操作步骤

1. 登录对象存储控制台。
2. 在左侧导航栏中，选择**存储桶列表**，进入存储桶列表页面。
3. 找到对象所在的存储桶，单击其存储桶名称，进入存储桶的文件列表页面。
4. 找到需要修改存储类型的单个对象，在其右侧操作栏中，单击右侧**更多操作 > 修改存储类型**进行设置。

若您需要对多个对象的存储类型进行批量修改，可勾选多个对象，并单击上方的**更多操作 > 修改存储类型**进行修改。

> **说明：**
>
> 仅支持同存储类型进行批量修改。

设置对象的访问权限

最近更新时间: 2025-01-15 17:01:00

简介

COS 提供基于对象维度的访问权限设置，且该权限优先级高于存储桶的访问权限。对象存储 COS 支持为对象设置两种权限类型：

- **公共权限**：包括继承权限、私有读写、公有读私有写。
- **用户权限**：主账号默认拥有对象所有权限（即完全控制）。另外 COS 支持添加子账号有数据读取、数据写入、权限读取、权限写入，甚至**完全控制**的最高权限。

> **注意：**
>
> 对象的访问权限只在用户通过默认域名访问时有效。通过自定义域名访问时，以存储桶访问权限为准。

操作步骤

1. 登录对象存储控制台。

2. 在左侧导航栏中，选择**存储桶列表**，单击对象所在的存储桶名称，进入存储桶文件列表页面。

3. 找到需要设置权限的对象，并单击右侧的**详情**。

4. 在**对象访问权限**配置中，修改访问权限后，单击**保存**。

   COS 支持为对象设置两种权限类型：

   - 公共权限：包括继承权限、私有读写、公有读私有写。
   - 用户权限：主账号默认拥有对象所有权限（即完全控制）。另外 COS 支持添加子账号有数据读取、数据写入、权限读取、权限写入，甚至完全控制的最高权限。

5. 若您需要对多个对象进行批量设置或修改访问权限，可勾选多个对象，并单击上方的**更多操作 > 修改访问权限**即可设置。

服务端加密

最近更新时间: 2025-01-15 17:01:00

简介

您可以通过对象存储控制台，对存放在存储桶中的对象设置加密，以防止信息被泄露。

> **说明：**
>
> 支持SSE-C方式的服务端加密（SSE-C 仅能通过 API 进行使用，不支持控制台操作）

案例：使用服务端加密 SSE-C

请求

```
PUT /exampleobject HTTP/1.1
Host: examplebucket-1250000000.cos.ap-beijing.myqcloud.com
Date: Fri, 10 Apr 2020 09:36:12 GMT
Content-Type: image/jpeg
x-cos-server-side-encryption-customer-algorithm: AES256
x-cos-server-side-encryption-customer-key: MDEyMzQ1Njc4OUFCQ0RFRjAxMjM0NTY3ODlBQkNERUY=
x-cos-server-side-encryption-customer-key-MD5: U5L61r7jcwdNvT7frmUG8g==
Content-Length: 16
Content-MD5: 7o3pGNBWQBRbGPcPTDqmAg==
Authorization: q-sign-algorithm=sha1&q-ak=AKID8A0fBVtYFrNm02oY1g1JQQF0c3JO****&q-sign-time=1586511372;1586518572&q-key-time=1586511372;1586518572&q-header-list=content-length;content-md5;content-type;date;host;x-cos-server-side-encryption-customer-algorithm;x-cos-server-side-encryption-customer-key;x-cos-server-side-encryption-customer-key-md5&q-url-param-list=&q-signature=4f6f9f0a6700930f70bff31e3a2b2e622711****
Connection: close
```

响应

```
HTTP/1.1 200 OK
Content-Length: 0
Connection: close
Date: Fri, 10 Apr 2020 09:36:13 GMT
ETag: "582d9105f71525f3c161984bc005efb5"
Server: tencent-cos
x-cos-hash-crc64ecma: 16749656579157681890
x-cos-request-id: NWU5MDNlMGNfZTFjODJhMDlfMzVlMDFfZTk1****
x-cos-server-side-encryption-customer-algorithm: AES256
x-cos-server-side-encryption-customer-key-MD5: U5L61r7jcwdNvT7frmUG8g==
```

自定义 Headers

最近更新时间: 2025-01-15 17:01:00

简介

对象的 HTTP 头部（Header ）是服务器以 HTTP 协议传送 HTML 资料到浏览器前所送出的字串。通过修改 HTTP 头部（Header ），可以改变页面的响应形式，或者传达配置信息，例如修改缓存时间。修改对象的 HTTP 头部不会修改对象本身。 例如：修改了 Header 中的 Content-Encoding 为 gzip，但是文件本身没有提前用 gz 压缩过，会出现解码错误。

操作步骤

1. 登录对象存储桶控制台。

2. 在左侧导航栏中，选择**存储桶列表**，进入存储桶列表页面。

3. 单击对象所在的存储桶，进入存储桶文件列表。

4. 找到需要设置头部的对象，单击对象右侧的**详情**。

5. 在**自定义 Headers**配置中，单击**添加 Header**，选择需要设置的参数类型（自定义内容需输入自定义名称），输入对应的值。

COS 提供了以下 6 种对象 HTTP 头部标识供配置。头部配置说明如下。

HTTP 头部	说明	示例
Content-Type	文件的 MIME 信息	image/jpeg
Cache-Control	文件的缓存机制	no-cache;max-age=200
Content-Disposition	MIME 协议的扩展	attachment;filename="fname.ext"
Content-Encoding	文件的编码格式	UTF-8
Expires	用来控制缓存的失效日期	Wed, 21 Oct 2015 07:28:00 GMT
x-cos-meta-[自定义内容]	自定义内容	自定义内容

6. 配置完成后，单击**保存**。

示例

在 APPID 为 1250000000 ，创建存储桶名称为 examplebucket-1250000000。存储桶根目录下上传了对象 exampleobject.txt。

未自定义对象的 HTTP 头部时，浏览器或客户端下载时得到的对象头部范例如下：

请求

```
GET /exampleobject.txt HTTP/1.1
Host: examplebucket-1250000000.file.myqcloud.com
Accept: */*
```

响应

```
HTTP/1.1 200 OK
Content-Language:zh-CN
Content-Type: text/plain
Content-Disposition: attachment; filename*="UTF-8''exampleobject.txt"
Access-Control-Allow-Origin: *
Last-Modified: Tue, 11 Jul 2017 15:30:35 GMT
```

添加如下配置：

再次发起请求，浏览器或客户端得到的对象头部范例如下：

请求

```
GET /exampleobject.txt HTTP/1.1
Host: examplebucket-1250000000.file.myqcloud.com
Accept: */*
```

响应

```
HTTP/1.1 200 OK
Content-Language:zh-CN
Cache-Control: no-cache
Content-Type: image/jpeg
Content-Disposition: attachment; filename*="abc.txt"
x-cos-meta-md5: 1234
Access-Control-Allow-Origin: *
Last-Modified: Tue, 11 Jul 2017 15:30:35 GMT
```

删除对象

最近更新时间: 2025-01-15 17:01:00

简介

您可以通过对象存储控制台，对上传到存储桶中的单个或多个对象进行删除。

删除单个对象

操作步骤

1. 登录对象存储控制台。
2. 在左侧导航栏中，选择**存储桶列表**，进入存储桶列表页面。
3. 单击对象所在的存储桶，进入存储桶文件列表。
   - 未列出历史版本

     找到您想要删除的对象，单击**更多操作 > 删除**。

   - 列出历史版本

     找到您想要删除的对象，单击**删除**。

4. 弹出删除文件对话框，单击**确认**即可删除对象。

删除多个对象

操作步骤

1. 在存储桶文件列表页面，勾选您想要删除的多个对象，单击**批量删除**。
2. 弹出删除文件对话框，单击**确认**即可批量删除对象。

文件夹管理
创建文件夹

最近更新时间: 2025-01-15 17:01:00

资源在 COS 中都是以对象的形式存储的，为延续用户使用习惯，在 COS 控制台可采用文件夹形式对对象进行管理。

> **注意：**
>
> 文件夹名称长度限制在 255 字符内，不支持保留字符和字段。

保留字符和字段

- 保留字段：[con]，[aux]，[nul]，[prn]，[com0]，[com1]，[com2]，[com3]，[com4]，[com5]，[com6]，[com7]，[com8]，[com9]，[lpt0]，[lpt1]，[lpt2]，[lpt3]，[lpt4]，[lpt5]，[lpt6]，[lpt7]，[lpt8]，[lpt9]。
- 保留 ASCII 控制字符： 字符上(↑)：CAN (24) 字符下(↓)：EM (25) 字符右(→)：SUB (26) 字符左(←)：ESC (27)

步骤

1. 登录对象存储桶控制台。

2. 在左侧导航栏中，选择**存储桶列表**，进入存储桶列表页面。

3. 单击需要创建文件夹的存储桶名称，进入存储桶的文件列表。

4. 单击**创建文件夹**。

5. 输入文件夹名称，单击**确定**。

删除文件夹

最近更新时间: 2025-01-15 17:01:00

步骤

1. 登录对象存储桶控制台。

2. 在左侧导航栏中，选择**存储桶列表**，进入存储桶列表页面。

3. 单击需要删除文件夹的存储桶名称，进入存储桶的文件列表页面。

4. 定位到待删除的文件夹，单击**删除**。

5. 单击**确认**，删除文件夹及该目录下所有文件。

设置对象标签

最近更新时间: 2025-01-15 17:01:00

简介

对象标签功能的实现是通过为对象添加一个键值对形式的标识，协助用户分组管理存储桶中的对象。对象标签由标签的键（tagKey）和标签的值（tagValue）与=相连组成，例如group = IT。用户可以对指定的对象进行标签的设定、查询、删除操作。

使用对象标签时需注意以下限制：

- 用户可为同一个对象最多添加10个对象标签，并且标签不可重复。
- 标签键不建议以qcs:、project、项目等作为开头，这些字符为系统预留标签键。
- 标签键和标签值支持 UTF-8 格式表示的字符、空格和数字以及特殊字符 + - = ._ : / @，长度范围均为1-127个字符，区分英文大小写。

上传对象时添加标签

1. 您可以在上传对象时添加，如下图所示：

2. 上传成功后，对象标签即可添加完成。

   您可以进入对象的详情页面中的**对象标签管理**配置里查看已添加好的标签：

3. 如果您需要修改或删除标签，可在**对象标签管理**配置项中，单击标签右侧的**编辑**或**删除**即可。

为已上传的对象添加标签

若您在上传对象时未添加标签，您可以按照下述步骤为对象添加标签。

1. 参见查看对象信息，进入需要添加标签的对象详情页面。

2. 在对象的详情页面中的**对象标签管理**配置项中，单击**添加标签**，为对象添加标签。

为已有的对象添加标签

如果您需要修改或删除标签，可在**对象标签管理**配置项中单击标签右侧的**编辑**或**删除**按钮即可。

还原历史版本对象

最近更新时间: 2025-01-15 17:01:00

简介

您可以通过对象存储控制台，还原对象的历史版本为最新版本。本文介绍如何在控制台对存储桶对象的历史版本进行还原。

> **说明：**
>
> - 还原对象指将历史版本还原为最新版本，并且历史版本仍保留。
> - 若您需要删除历史版本，请查看 设置版本控制、删除对象 操作。
> - 支持单个对象还原操作，不支持批量还原。
> - 加密对象不支持还原操作。

使用须知

还原对象的适用场景及规则如下：

适用场景	• 已开启版本控制的存储桶，支持还原对象操作。 • 开启版本控制后又暂停版本控制的存储桶，不支持还原对象操作。 • 未曾开启版本控制的存储桶，不支持还原对象操作。
还原规则	• 历史版本：支持还原。 • 最新版本：无法还原。 • 带有删除标记版本：无法还原。

前提条件

还原对象前，请您确保存储桶已开启版本控制。如未开启，请参考文档 设置版本控制 进行操作。

操作步骤

1. 登录对象存储控制台。
2. 在左侧导航栏中，单击**存储桶列表**，进入存储桶列表页面。
3. 找到对象所在的存储桶，单击存储桶名称，进入文件列表页面。
4. 打开**列出历史版本**开关，找到您需要还原的对象，单击**还原**。
5. 在还原文件弹框中，检查还原对象的版本信息。确认无误后，单击**确定**。

   还原对象操作完成后，在文件列表中可以看到，历史版本已被成功还原。

数据统计
基础数据统计

最近更新时间: 2025-01-15 17:01:00

简介

监控报表为用户提供对象存储的服务数据统计，用户可通过监控报表数据了解各数据的趋势。监控报表包括基础数据统计和返回码统计。基础数据统计页面为用户提供具体的服务统计数据，并且以趋势图的形式形象地展示服务的使用情况。

操作步骤

1. 登录对象存储控制台 。

2. 在左侧导航栏中，选择**监控报表** > **基础数据统计**，即可进入基础数据统计页面。

- 基础数据统计页面可切换地域和存储桶，查看当天、昨天、近 7 天、近 15 天、近 30 天的数据：
  - 对应时间段以具体统计数字显示：存储容量、对象数量、总访问流量、总读请求数、总写请求数；
  - 对应时间段以趋势图显示：存储量统计、对象数量、外网下行流量、内网流量、读请求统计和写请求统计。
- 页面支持导出数据到本地，单击日期栏右侧的导出按钮即可导出数据。数据可能存在延迟，可刷新页面重新加载。

版权所有：腾讯云金融专区

返回码统计

最近更新时间: 2025-01-15 17:01:00

操作步骤

1. 登录对象存储控制台。

2. 在左侧导航栏中，选择**监控报表 > 返回码统计**，即可进入返回码统计页面。

- 返回码统计页面以对应时间段趋势图和列表的形式，可切换项目和存储桶，查看当天、昨天、近 7 天、近 15 天、近 30 天的数据：
  - HTTP 请求成功率统计

    HTTP 请求成功率是由各个返回码出现次数占总请求次数的比例计算得出。该数据统计的是用户使用中，对 COS 发起 HTTP 请求（包括读和写请求）的成功百分率。
  - 返回码统计

    返回码统计是用户所选时间范围内，HTTP 请求返回码出现的次数的统计，可以帮助用户定位访问情况。
  - 请求类型统计

    请求类型统计按照请求类型分类统计请求的数量，主要包括 GET, HEAD, PUT, POST, DETELE 这5种请求。
- 页面上半部分显示趋势图，下半部分显示统计列表。统计列表中每个有数据的栏，可以查看该成功率的详细情况。
- 页面支持导出数据到本地，单击日期栏右侧的导出按钮即可导出数据。数据可能存在延迟，可刷新页面重新加载。

监控告警
设置监控告警

最近更新时间: 2025-01-15 17:01:00

简介

您可以通过云监控的告警策略来设置 COS 监控指标的阈值告警，告警策略包括名称、策略类型和告警触发条件、告警对象、告警通知模板五个必要组成部分。

> **说明：**
>
> 云监控是一项对云产品资源实时监控和告警的服务，用户可通过云监控对 COS 的监控指标设置告警规则，查看告警历史，接收告警通知。

操作步骤

1. 登录云监控控制台。

2. 在左侧导航栏中，选择**告警配置** > **告警策略**。

3. 单击**新建**，开始配置告警策略。

4. 配置完以上信息后，单击**完成**，即成功创建 COS 告警策略。

5. 根据已配置的告警策略进行触发，通过进入策略名称内可查看到告警历史记录

6. 当需要停止监控告警，只需要将对应规则的状态修改为**关闭**或者单击**删除**，直接删除告警策略。

高级功能
支持IPV6访问

最近更新时间: 2025-01-15 17:01:00

操作场景

客户端支持内网/外网 IPV6访问，并通过 IPV6 的形式访问COS

存储桶Policy设置IPV6

1. 进入存储桶文件列表页面，选择左侧**权限管理 > Policy权限设置**，单击**添加策略。**

2. 填写要设置的信息，并将条件设置IP等于IPV6条件值。

此时将禁止所有用户通过改IPV6地址访问存储桶所有操作。

跨域访问CORS设置IPV6

1. 进入存储桶文件列表页面，选择**安全管理 > 跨域访问CORS设置**，单击**添加规则。**

2. 将允许来源Origion设置为IPV6地址。

此时将允许该域进行跨域访问。

防盗链设置IPV6

1. 进入存储桶文件列表页面，选择**安全管理 > 防盗链设置**，单击**编辑。**

2. 将Referer设置为IPV6地址。

此时将允许名单内的域名访问存储桶。

地域白名单

最近更新时间: 2025-01-15 17:01:00

简介

支持地域白名单，进入白名单的租户（UIN）才可以访问本Region资源。

操作步骤

1. 登录运营端控制台。

2. 选择**平台运营 > 客户管理**。

3. 在左侧导航树中，选择**白名单管理**，可新增相应的白名单类型，如设置指定白名单账号为生效白名单类型。

4. 只有在白名单列表中的租户（UIN）才可以访问本region资源，否则无访问权限。

开发者指南
存储桶
存储桶概述

最近更新时间: 2025-01-15 17:01:00

定义

存储桶（Bucket）是对象的载体，可理解为存放对象的“容器”。用户可以通过云控制台、API、SDK 等多种方式管理存储桶以及配置属性。例如，用户可以配置存储桶用于静态网站托管、配置存储桶的访问权限等。

命名规范

存储桶名称由两部分组成：**用户自定义字符串**和**系统生成数字串（APPID）**，两者以中划线“-”相连。例如 examplebucket-1250000000 ，其中 examplebucket 为用户自定义字符串，1250000000 为系统生成数字串（APPID）。 在 API、SDK 的示例中，存储桶的命名格式为 <BucketName-APPID> 。

- 系统生成数字串 APPID：由系统自动分配，无需用户输入，其在腾讯云金融专区具有唯一性。
- 用户自定义字符串：由用户手动输入的一串字符，规范如下。

  自定义字符串的命名规范：

  - 仅支持小写英文字母和数字，即[a-z，0-9]、中划线“-”及其组合。
  - 用户自定义的字符串支持1 - 50个字符。
  - 存储桶命名不能以“-”开头或结尾。

以下是有效的存储桶命名示例：

- mybucket123-1250000000
- 1-newproject-1250000000

访问权限

存储桶默认提供两种权限类型：公共权限和用户权限。

公共权限

公共权限包括：私有读写、公有读私有写和公有读写。其访问权限可通过对象存储控制台上的存储桶的【权限管理】进行修改。

- 私有读写

  只有该存储桶的创建者及有授权的账号才对该存储桶中的对象有读写权限，其他任何人对该存储桶中的对象都没有读写权限。存储桶访问权限默认为私有读写，推荐使用。

- 公有读私有写

  任何人（包括匿名访问者）都对该存储桶中的对象有读权限，但只有存储桶创建者及有授权的账号才对该存储桶中的对象有写权限。

- 公有读写

  任何人（包括匿名访问者）都对该存储桶中的对象有读权限和写权限，不推荐使用。

用户权限

主账号默认拥有存储桶的所有权限（即完全控制）。另外 COS 支持添加子账号有数据读取、数据写入、权限读取、权限写入，甚至完全控制的最高权限。

相关说明

- 对象存储以扁平化结构来存放对象，无文件夹概念。详情请参见 对象概述 文档中的“文件夹和目录”部分。
- 同一用户账号下，可以创建多个存储桶，数量上限是200个（不区分地域），但是存储桶中的对象数量没有限制。
- 同一个 APPID 下的存储桶名称是唯一的，不能重名。
- 存储桶一旦创建后，将无法重命名。您只能删除后重新创建再命名存储桶。
- 用户在创建存储桶时，请确认好所属地域，地域一旦设置后将无法修改。

创建存储桶

最近更新时间: 2025-01-15 17:01:00

适用场景

在开始使用 COS 时，您需要先创建一个存储桶以便于对象的使用和管理。您可以通过控制台、API 或 SDK 的方式来创建存储桶。

当存储桶不存在时，您可以使用以下代码示例在指定地域创建存储桶，存储桶支持的参数为：

- Bucket：用于指定您的完整存储桶名称，形如 `testbuc-125235912`。
- Region：选择您的云平台服务地域，一旦创建将不可移动或修改存储桶。

使用方法

使用 REST API

您可以直接使用 REST API 发起一个创建存储桶的请求，可参考 Put Bucket 文档说明。

使用 Java SDK

对象存储 COS 的 Java SDK 中提供了此方法，可参考 Java SDK 接口文档 Put Bucket 部分。

步骤说明

1. 初始化客户端 cosclient。
2. 执行 createBucket 创建 Bucket，创建 Bucket 时可指定 Bucket 的权限（公有读写或私有读）。

代码示例

调用 createBucket 创建 Bucket，代码示例如下所示：

```
// 1 初始化用户身份信息(appid, secretId, secretKey)
COSCredentials cred = new BasicCOSCredentials("AKIDXXXXXXXX", "1A2Z3YYYYYYYYYY");
// 2 设置bucket的区域, COS地域的简称请参照 /document/product/436/6224
ClientConfig clientConfig = new ClientConfig(new Region("ap-beijing-1"));
// 3 生成cos客户端
COSClient cosclient = new COSClient(cred, clientConfig);
String bucketName = "publicreadbucket-1251668577";
CreateBucketRequest createBucketRequest = new CreateBucketRequest(bucketName);
// 设置bucket的权限为PublicRead(公有读私有写), 其他可选Private(私有读写), PublicReadWrite(公有读写)
createBucketRequest.setCannedAcl(CannedAccessControlList.PublicRead);
Bucket bucket = cosclient.createBucket(createBucketRequest);
```

删除存储桶

最近更新时间: 2025-01-15 17:01:00

适用场景

当您在某些情况下需要删除存储桶时，您可以通过控制台、API 或 SDK 的方式来删除存储桶。

> **注意：**
>
> 目前仅支持删除已经清空的存储桶，如果存储桶中仍有对象，将会删除失败。请在执行删除存储桶前确保存储桶内已经没有对象。

当删除存储桶时，您需要确保操作的身份已被授权该操作，并确认传入了正确的存储桶名称（Bucket）和地域（Region）参数。

使用方法

使用 REST API

您可以直接使用 REST API 发起一个删除存储桶请求，可参考 Delete Bucket 文档说明。

使用 Java SDK

对象存储 COS 的 Java SDK 中提供了此方法，可参考 Java SDK 接口文档 Delete Bucket 部分。

步骤说明

1. 初始化客户端 cosclient。
2. 执行 deleteBucket 删除 Bucket，Bucket 必须不包含任何数据，否则需要先清空数据。

代码示例

调用 deleteBucket 创建 Bucket，代码示例如下所示：

```
// 1 初始化用户身份信息(appid, secretId, secretKey)
COSCredentials cred = new BasicCOSCredentials("AKIDXXXXXXXX", "1A2Z3YYYYYYYYYY");
// 2 设置bucket的区域, COS地域的简称请参照 /document/product/436/6224
ClientConfig clientConfig = new ClientConfig(new Region("ap-beijing-1"));
// 3 生成cos客户端
COSClient cosclient = new COSClient(cred, clientConfig);

// bucket名称需包含appid
String bucketName = "publicreadbucket-1251668577";
// 删除bucket, 只能删除不包含任何数据的bucket
cosclient.deleteBucket(bucketName);
```

对象
对象概述

最近更新时间: 2025-01-15 17:01:00

定义

对象（Object）是对象存储的基本单元，对象被存放到存储桶中（例如一张照片存放到一个相册）。用户可以通过腾讯云金融专区控制台、API、SDK 等多种方式管理对象。在 API、SDK 示例中，对象的命名格式为 `<ObjectKey>` 。

> **注意：**
>
> 对象的上传分为两种，分别是简单上传和分块上传。
>
> - 使用简单上传，对象大小限制在5GB以内。
> - 使用分块上传，每块的大小限制在5GB以内，分块数量需要小于10000，即最大上传对象为48.82TB。

每个对象都由对象键（ObjectKey）、数据值（Value）、和对象元数据（Metadata）组成。

- 对象键（ObjectKey）：对象键是对象在存储桶中的唯一标识。
- 数据值（Value）：即上传的对象大小。
- 对象元数据（Metadata）：是一组名称值对，您可以在上传对象时对其进行设置。

用户可以通过控制台对对象进行相关配置。

- 搜索对象
- 查看对象信息
- 设置对象的访问权限
- 设置自定义 Headers

对象键

定义

COS 中的对象需具有合法的对象键，对象键（ObjectKey）是对象在存储桶中的唯一标识。 例如：在对象的访问地址 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/folder/picture.jpg` 中，对象键为 `folder/picture.jpg` 。

命名规范

- 键的名称可以使用任何 UTF-8 字符，为了使名称有助于确保与其他应用程序的最大兼容性，推荐使用大小写英文字母、数字，即[a-z，A-Z，0-9]和符号 `-`，`!`，`_`，`.`，`*` 及其组合。
- 编码长度最大为850个字节。
- 对象键中不支持 ASCII 控制字符中的字符上(↑)，字符下(↓)，字符右(→)，字符左(←)，分别对应 CAN(24)，EM(25)，SUB(26)，ESC(27)。
- 如果用户上传的文件或文件夹的名字带有中文，在访问和请求这个文件或文件夹时，中文部分将按照 URL Encode 规则转化为百分号编码。 例如：对 `文档.doc` 进行访问的时候，对象键为： `文档.doc` ，实际读取的按 URL Encode 规则转化的百分号编码为： `%e6%96%87%e6%a1%a3.doc` 。

以下是有效的对象键命名示例：

- my-organization
- my.great_photos-2016/01/me.jpg
- videos/2016/birthday/video.wmv

特殊字符

有些字符在对象键中可能需要以十六进制形式在 URL 中编码或引用，其中有些甚至是不可被打印的，因此浏览器可能无法处理它们，对于这些字符需要进行特殊处理。可能需要特殊处理的字符如下：

,	:	;	=
&	$	@	+
?	ASCII 字符范围：00-1F 十六进制（0-31 十进制）以及7F（127 十进制）	（空格）	

还有些字符因为需要进行大量的特殊处理才能在所有应用程序间保持一致性，所以建议直接避免使用。需要避免的字符如下：

`	^	"	\
{	}	[	]
~	%	#	|
>	<	ASCII 128-255 十进制	

相关说明

访问地址

对象的访问地址由存储桶访问地址和对象键组成，其结构形式为 [存储桶域名]/[对象键] 。

例如：上传对象 exampleobject.txt 到广州（华南）的存储桶 examplebucket-1250000000 中，那么 exampleobject.txt 的访问地址是： examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/exampleobject.txt 。

文件夹和目录

对象存储中本身是没有文件夹和目录的概念的，对象存储不会因为上传对象 project/a.txt 而创建一个 project 文件夹。为了满足用户使用习惯，对象存储在控制台中模拟了「文件夹」或「目录」的展示方式，具体实现是通过创建一个键值为 project/ ，内容为空的对象，展示方式上模拟了传统文件夹。

例如：通过 API、SDK 上传对象 project/doc/a.txt ，分隔符 / 会模拟「文件夹」的展示方式，于是可以看到控制台上出现「文件夹」 project 和 doc ，其中 doc 是 project 下一级「文件夹」，并包含了 a.txt 。

> 注意：
>
> 存储桶中不同对象是扁平的分布到不同的分布式集群中的，因此无法直接获取某对象键前缀容量的大小，只能通过累加各对象的大小得到。

对于文件夹和目录进行删除操作，情况会比较复杂，详情如下：

删除途径	删除	结果
控制台	文件夹`project`	对象键前缀为`project/`的全部对象都会被删除
控制台	对象`project/doc/a.txt`	`project`和`doc`文件夹仍会保留
API、SDK	对象`project/`或`project/doc/`	对象`project/doc/*.txt`仍会保留。如果想将文件夹内的对象一并删除，则需要用代码遍历实现删除文件夹内对象的功能

对象元数据

定义

对象元数据在对象中是一组名称值对，是服务器以 HTTP 协议传 HTML 资料到浏览器前所送出的字串，又称为 HTTP Header。通过在上传对象时修改 HTTP Header，可以改变页面的响应形式，或者传达配置信息，例如修改缓存时间。

对象元数据包括有两种元数据：系统元数据和用户定义的元数据。

> 说明：
>
> 修改对象的 HTTP Header 不会修改对象本身。

系统元数据

指的是对象的属性信息，如上传或修改的时间等。

名称	说明
Date	当前日期和时间
Content-Length	RFC 2616 中定义的 HTTP 请求内容长度（字节），常用于 PUT 类型的 API 的操作
Last-Modified	对象创建日期或上次修改日期（以较晚者为准）
Content-MD5	RFC 1864 中定义的经过 Base64 编码的 128-bit 内容 MD5 校验值。此头部用来校验文件内容是否发生变化

名称	说明
Authorization	携带鉴权信息， 用以验证请求合法性的签名信息。针对公有读的文件，无需携带此头部
x-cos-version-id	对象版本。在存储桶上启用版本控制时，返回对象的版本 ID
ETag	如通过 PUT Object 上传的对象，则为上传文件内容的 MD5 值；如通过分块上传或使用历史版本 API 上传的对象，为上传文件内容的唯一 ID，不具备校验功能
Expect	RFC 2616 中定义的 HTTP 请求内容长度（字节）
Connection	声明客户端与服务端之间的通信状态。 枚举值：keep-alive，close

用户定义元数据

指的是对象的可自定义参数，如 Content-Type，Cache-Control，Expires，x- cos-meta- 等。

名称	描述
Cache-Control	RFC 2616 中定义的缓存策略，将作为 Object 元数据保存
Content-Disposition/Encoding/Type	RFC 2616 中定义的文件名称/编码格式/内容类型（MIME），将作为 Object 元数据保存
Expires	对象缓存过期时间。
x-cos-acl	定义 Object 的 ACL 属性。有效值：private，public-read-write，public-read；默认值：private
x-cos-grant-*	赋予被授权者某种权限
x-cos-meta- *	允许用户自定义的头部信息，将作为 Object 元数据返回（大小限制为2K）
x-cos-storage-class	设置 Object 的存储级别，枚举值：STANDARD，STANDARD_IA，ARCHIVE，默认值：STANDARD
x-cos-server-side-encryption	指定是否为对象启用服务端加密的方式。 使用 COS 主密钥加密填写：AES256

对象子资源

COS 有与存储桶和对象相关联的子资源。子资源从属于对象，即子资源不会自行存在，它始终与某些其他实体 (例如对象或存储桶) 相关联。 访问控制列表（Access Control List）是指特定对象的访问控制信息列表，它是 COS 中对象的子资源。

访问控制列表包含可以识别被授权者和其被授予的许可的授权列表，来实现对对象的访问控制。创建对象时，ACL 将识别可以完全控制对象的对象所有者。用户可以检索对象 ACL 或将其替换为更新的授权列表。

> **说明：**
>
> 对 ACL 的任何更新都需要替换现有 ACL。

访问权限类型

对象存储 COS 支持对象设置两种权限类型：**公共权限**和**用户权限**。 **公共权限**：包括继承权限、私有读写和公有读私有写。

- 继承权限：对象继承存储桶的权限，与存储桶的访问权限一致。当访问对象时，COS 读取到对象权限为继承存储桶权限，会匹配存储桶的权限，来响应访问。任何新对象被添加时，默认继承存储桶权限。
- 私有读写：当访问对象时，COS 读取到对象的权限为私有读写，此时无论存储桶为何种权限，对象都需要通过签名鉴权才可访问。
- 公有读私有写：当访问对象时，COS 读取到对象的权限为公有读，此时无论存储桶为何种权限，对象都可以被直接下载。

用户权限：主账号默认拥有对象所有权限（即完全控制）。另外 COS 支持添加子账号有数据读取、数据写入、权限读取、权限写入，甚至完全控制的最高权限。

适用场景

在私有读写的存储桶中对特定对象设置允许公有访问，或在公有读写存储桶中对特定对象设置需要鉴权才可访问。

上传对象
简单上传

最近更新时间: 2025-01-15 17:01:00

适用场景

该操作适用于在单个请求中上传一个小于 5 GB 大小的对象，对于大于 5 GB 的对象，您必须使用分块上传的方式。

当您的对象较大（例如 100 MB）时，我们建议您在高带宽或弱网络环境中，优先使用分块上传的方式。

使用方法

使用 REST API

您可以直接使用 REST API 发起一个简单上传对象请求，可参考 PUT Object 文档说明。

使用 Java SDK

对象存储 COS 的 Java SDK 中提供了此方法，可参考 Java SDK 接口文档 PUT Object 部分 。

步骤说明

1. 初始化客户端 cosclient。
2. 执行 putObject 方法上传对象，支持将本地文件或者输入流上传到 COS。

代码示例

1. `PutObjectRequest` 封装了简单上传的请求，通过传入本地文件路径以及 COS 路径，支持设置存储类型，权限信息等，上传完成后会返回 `PutObjectResult` ，失败抛出异常。示例代码如下：

```
// 1 初始化用户身份信息(secretId, secretKey)
COSCredentials cred = new BasicCOSCredentials("AKIDXXXXXXXX", "1A2Z3YYYYYYYYYY");
// 2 设置bucket的区域, COS地域的简称请参照 /document/product/436/6224
ClientConfig clientConfig = new ClientConfig(new Region("ap-beijing-1"));
// 3 生成cos客户端
COSClient cosclient = new COSClient(cred, clientConfig);
// bucket名需包含appid
String bucketName = "mybucket-1251668577";

String key = "/aaa/bbb.txt";
File localFile = new File("src/test/resources/len10M.txt");
PutObjectRequest putObjectRequest = new PutObjectRequest(bucketName, key, localFile);
// 设置存储类型, 默认是标准(Standard), 低频(standard_ia), 近线(nearline)
putObjectRequest.setStorageClass(StorageClass.Standard_IA);
try {
PutObjectResult putObjectResult = cosclient.putObject(putObjectRequest);
// putobjectResult会返回文件的etag
String etag = putObjectResult.getETag();
} catch (CosServiceException e) {
e.printStackTrace();
} catch (CosClientException e) {
e.printStackTrace();
}

// 关闭客户端
cosclient.shutdown();
```

2. `PutObjectRequest` 同时支持传入输入流，从流式上传到COS，但需要指定长度，示例代码如下所示：

```
// 1 初始化用户身份信息(secretId, secretKey)
COSCredentials cred = new BasicCOSCredentials("AKIDXXXXXXXX", "1A2Z3YYYYYYYYYY");
// 2 设置bucket的区域, COS地域的简称请参照 /document/product/436/6224
ClientConfig clientConfig = new ClientConfig(new Region("ap-beijing-1"));
// 3 生成cos客户端
```

```
COSClient cosclient = new COSClient(cred, clientConfig);
// bucket名需包含appid
String bucketName = "mybucket-1251668577";

String key = "/aaa/bbb.txt";
File localFile = new File("src/test/resources/len10M.txt");

InputStream input = new ByteArrayInputStream(new byte[10]);
ObjectMetadata objectMetadata = new ObjectMetadata();
// 从输入流上传必须制定content length, 否则http客户端可能会缓存所有数据，存在内存OOM的情况
objectMetadata.setContentLength(10);
// 设置contenttype默认下载时根据cos路径key的后缀返回响应的contenttype, 上传时设置contenttype会覆盖默认值
objectMetadata.setContentType("image/jpeg");

PutObjectRequest putObjectRequest =
new PutObjectRequest(bucketName, key, input, objectMetadata);
// 设置存储类型, 默认是标准(Standard), 低频(standard_ia), 近线(nearline)
putObjectRequest.setStorageClass(StorageClass.Standard_IA);
try {
PutObjectResult putObjectResult = cosclient.putObject(putObjectRequest);
// putobjectResult会返回文件的etag
String etag = putObjectResult.getETag();
} catch (CosServiceException e) {
e.printStackTrace();
} catch (CosClientException e) {
e.printStackTrace();
}

// 关闭客户端
cosclient.shutdown();
```

分块上传

最近更新时间: 2025-01-15 17:01:00

适用场景

分块上传适合于在弱网络或高带宽环境下上传较大的对象。COS 的控制台和 SDK 会协助您将单个对象切成一组分块并完成上传，您也可以自行切分对象并分别调用 API 上传各个分块。使用分块上传有一些优势：

- 在弱网络环境中，使用较小的分块可以将网络失败导致的中断影响降低，实现对象续传。
- 在高带宽环境中，并发上传对象分块能充分利用网络带宽，乱序上传并不影响最终组合对象。
- 使用分块上传，您可以随时暂停和恢复单个大对象的上传。除非发起终止操作，所有未完成的对象将可随时继续上传。
- 分块上传也适用于在未知对象总大小的情况下上传对象，您可以先发起上传，再组合对象以获得完整大小。

上传时，这组分块将会按连续的序号编号，您可以独立上传或者按照任意顺序上传各个分块，最终 COS 将会根据分块编号顺序重新组合出该对象。任意分块传输失败，都可以重新传输当前分块，不会影响其他分块和内容完整性。一般在弱网络环境中，当单个对象大于 20 MB 可优先考虑分块上传，在大带宽环境中可将超过 100 MB 的对象进行分块上传。

使用方法

使用 REST API

您可以直接使用 REST API 发起一个分块上传的请求，可参考以下 API 文档部分：

- Initiate Multipart Upload
- Complete Multipart Upload
- Upload Part
- Abort Multipart Upload

使用 Java SDK

对象存储 COS 的 Java SDK 中提供了此方法，可参考 Java SDK 接口文档 PUT object 中的分块文件上传部分。

步骤说明

1. 初始化客户端 cosclient。
2. 使用 initiateMultipartUpload 初始化分块上传获取一个新的 uploadid，或者调用 listMultipartUploads 获取之前还未完成的分块上传，得到 uploadid。
3. 已上传的分块可使用 listParts 进行获取，未上传的分块使用 uploadPart 上传分块数据, 或者 copyPart 选择从另外一个文件 copy 分块到目前文件。以此达到断点续传的功能，如果对已上传的分块再次调用 uploadPart 或者 copyPart 则会覆盖已上传的分块数据。
4. 使用 completeMultipartUpload 完成分块上传或者调用 abortMultipartUpload 终止分块上传。

代码示例

1. `InitiateMultipartUploadRequest` 是初始化分块上传的请求，包含了分快上传的路径，存储类型等信息。通过调用 initiateMultipartUpload 可获得一个新的分快上传 ID。

```
// 1 初始化用户身份信息(secretId, secretKey)
COSCredentials cred = new BasicCOSCredentials("AKIDXXXXXXXX", "1A2Z3YYYYYYYYYY");
// 2 设置bucket的区域, COS地域的简称请参照 /document/product/436/6224
ClientConfig clientConfig = new ClientConfig(new Region("ap-beijing-1"));
// 3 生成cos客户端
COSClient cosclient = new COSClient(cred, clientConfig);
// bucket名需包含appid
String bucketName = "mybucket-1251668577";

String key = "aaa/bbb.txt";
InitiateMultipartUploadRequest request = new InitiateMultipartUploadRequest(bucketName, key);
// 设置存储类型, 默认是标准(Standard), 低频(standard_ia), 近线(nearline)
request.setStorageClass(StorageClass.Standard_IA);
try {
InitiateMultipartUploadResult initResult = cosclient.initiateMultipartUpload(request);
// 获取uploadid
String uploadId = initResult.getUploadId();
```

```
} catch (CosServiceException e) {
e.printStackTrace();
} catch (CosClientException e) {
e.printStackTrace();
}

cosclient.shutdown();
```

2. UploadPartRequest 是分块上传请求，包含了要上传的数据，分块号。通过调用uploadPart 上传分块，并获取分块的 partEtag。

```
// 1 初始化用户身份信息(secretId, secretKey)
COSCredentials cred = new BasicCOSCredentials("AKIDXXXXXXXX", "1A2Z3YYYYYYYYYY");
// 2 设置bucket的区域, COS地域的简称请参照 /document/product/436/6224
ClientConfig clientConfig = new ClientConfig(new Region("ap-beijing-1"));
// 3 生成cos客户端
COSClient cosclient = new COSClient(cred, clientConfig);
// bucket名需包含appid
String bucketName = "mybucket-1251668577";
String key = "aaa/bbb.txt";
// uploadid(通过initiateMultipartUpload或者ListMultipartUploads获取)
String uploadId = "1512380198aecfed004aeaaca195d08232f718f7b52a91b8f6e9d36c7dfead2b3d1c917a6f";

// 生成要上传的数据, 这里初始化一个1M的数据
byte data[] = new byte[1024 * 1024];
UploadPartRequest uploadPartRequest = new UploadPartRequest();
uploadPartRequest.setBucketName(bucketName);
uploadPartRequest.setKey(key);
uploadPartRequest.setUploadId(uploadId);
// 设置分块的数据来源输入流
uploadPartRequest.setInputStream(new ByteArrayInputStream(data));
// 设置分块的长度
uploadPartRequest.setPartSize(data.length); // 设置数据长度
uploadPartRequest.setPartNumber(10); // 假设要上传的part编号是10

try {
UploadPartResult uploadPartResult = cosclient.uploadPart(uploadPartRequest);
PartETag partETag = uploadPartResult.getPartETag();
} catch (CosServiceException e) {
e.printStackTrace();
} catch (CosClientException e) {
e.printStackTrace();
}

cosclient.shutdown();
```

3. CompleteMultipartUploadRequest 是完成分块请求，需要传入之前上传的所有分块的partEtag。通过调用completeMultipartUpload完成分块上传。示例代码如下：

```
// 1 初始化用户身份信息(secretId, secretKey)
COSCredentials cred = new BasicCOSCredentials("AKIDXXXXXXXX", "1A2Z3YYYYYYYYYY");
// 2 设置bucket的区域, COS地域的简称请参照 /document/product/436/6224
ClientConfig clientConfig = new ClientConfig(new Region("ap-beijing-1"));
// 3 生成cos客户端
COSClient cosclient = new COSClient(cred, clientConfig);
// bucket名需包含appid
String bucketName = "mybucket-1251668577";
String key = "aaa/bbb.txt";
// uploadid(通过initiateMultipartUpload或者ListMultipartUploads获取)
String uploadId = "1512380198aecfed004aeaaca195d08232f718f7b52a91b8f6e9d36c7dfead2b3d1c917a6f";

// 这里初始化一个空的队列， 用于保存已上传的分片信息, 代码不能直接运行， 需要通过之前的uploadpart或者list parts的结果获取,加入到partEtags队列中
List<PartETag> partETags = new LinkedList<>();

// 分片上传结束后，调用complete完成分片上传
CompleteMultipartUploadRequest completeMultipartUploadRequest =
new CompleteMultipartUploadRequest(bucketName, key, uploadId, partETags);
try {
CompleteMultipartUploadResult completeResult =
cosclient.completeMultipartUpload(completeMultipartUploadRequest);
String etag = completeResult.getETag();
} catch (CosServiceException e) {
```

```
e.printStackTrace();
} catch (CosClientException e) {
e.printStackTrace();
}

cosclient.shutdown();
```

4. AbortMultipartUploadRequest 是终止分块上传请求，用于终止一个未 complete 的分块上传，表示中断销毁之前已上传的分块。通过调用 abortMultipartUpload 终止分块上传请求，示例代码如下：

```
// 1 初始化用户身份信息(secretId, secretKey)
COSCredentials cred = new BasicCOSCredentials("AKIDXXXXXXXX", "1A2Z3YYYYYYYYYY");
// 2 设置bucket的区域, COS地域的简称请参照 /document/product/436/6224
ClientConfig clientConfig = new ClientConfig(new Region("ap-beijing-1"));
// 3 生成cos客户端
COSClient cosclient = new COSClient(cred, clientConfig);
// bucket名需包含appid
String bucketName = "mybucket-1251668577";
String key = "aaa/bbb.txt";
// uploadid(通过initiateMultipartUpload或者ListMultipartUploads获取)
String uploadId = "1512380198aecfed004aeaaca195d08232f718f7b52a91b8f6e9d36c7dfead2b3d1c917a6f";

AbortMultipartUploadRequest abortMultipartUploadRequest = new AbortMultipartUploadRequest(bucketName, key, uploadId);
try {
cosclient.abortMultipartUpload(abortMultipartUploadRequest);
} catch (CosServiceException e) {
e.printStackTrace();
} catch (CosClientException e) {
e.printStackTrace();
}

cosclient.shutdown();
```

5. ListPartsRequest 是列出已上传的分块的请求，可用于断点续传。通过调用 listParts 获取已上传的分块，示例代码如下：

```
// 1 初始化用户身份信息(secretId, secretKey)
COSCredentials cred = new BasicCOSCredentials("AKIDXXXXXXXX", "1A2Z3YYYYYYYYYY");
// 2 设置bucket的区域, COS地域的简称请参照 /document/product/436/6224
ClientConfig clientConfig = new ClientConfig(new Region("ap-beijing-1"));
// 3 生成cos客户端
COSClient cosclient = new COSClient(cred, clientConfig);

// bucket名需包含appid
String bucketName = "mybucket-1251668577";
String key = "aaa/bbb.txt";
// uploadid(通过initiateMultipartUpload或者ListMultipartUploads获取)
String uploadId = "1512380198aecfed004aeaaca195d08232f718f7b52a91b8f6e9d36c7dfead2b3d1c917a6f";

List<PartETag> partETags = new LinkedList<>(); // 用于保存已上传的分片信息
PartListing partListing = null;
ListPartsRequest listPartsRequest = new ListPartsRequest(bucketName, key, uploadId);
do {
try {
partListing = cosclient.listParts(listPartsRequest);
} catch (CosServiceException e) {
e.printStackTrace();
} catch (CosClientException e) {
e.printStackTrace();
}
for (PartSummary partSummary : partListing.getParts()) {
partETags.add(new PartETag(partSummary.getPartNumber(), partSummary.getETag()));
}
listPartsRequest.setPartNumberMarker(partListing.getNextPartNumberMarker());
} while (partListing.isTruncated());

cosclient.shutdown();
```

6. CopyPartResult 是分块 copy 的请求，表示该分块的数据来源于另外一个文件的某一部分。通过要拷贝的源文件的路径，bucket 信息，字节范围。调用 copyPart 可实现分块copy。示例代码如下所示：

```
// 1 初始化用户身份信息(secretId, secretKey)
COSCredentials cred = new BasicCOSCredentials("AKIDXXXXXXXX", "1A2Z3YYYYYYYYYY");
// 2 设置bucket的区域, COS地域的简称请参照 /document/product/436/6224
ClientConfig clientConfig = new ClientConfig(new Region("ap-beijing-1"));
// 3 生成cos客户端
COSClient cosclient = new COSClient(cred, clientConfig);
// bucket名需包含appid
String bucketName = "mybucket-1251668577";
String key = "aaa/bbb.txt";
// uploadid(通过initiateMultipartUpload或者ListMultipartUploads获取)
String uploadId = "1512380198aecfed004aeaaca195d08232f718f7b52a91b8f6e9d36c7dfead2b3d1c917a6f";

CopyPartRequest copyPartRequest = new CopyPartRequest();
// 要拷贝的源文件所在的region
copyPartRequest.setSourceBucketRegion(new Region("ap-beijing-1"));
// 要拷贝的源文件的bucket名称
copyPartRequest.setSourceBucketName(bucketName);
// 要拷贝的源文件的路径
copyPartRequest.setSourceKey("aaa/ccc.txt");
// 指定要拷贝的源文件的数据范围(类似content-range)
copyPartRequest.setFirstByte(0L);
copyPartRequest.setLastByte(1048575L);
// 目的bucket名称
copyPartRequest.setDestinationBucketName(bucketName);
// 目的路径名称
copyPartRequest.setDestinationKey(key);

// uploadid
copyPartRequest.setUploadId(uploadId);
try {
CopyPartResult copyPartResult = cosclient.copyPart(copyPartRequest);
PartETag partETag = copyPartResult.getPartETag();
} catch (CosServiceException e) {
e.printStackTrace();
} catch (CosClientException e) {
e.printStackTrace();
}

cosclient.shutdown();
```

预签名授权上传

最近更新时间: 2025-01-15 17:01:00

适用场景

在默认情况下，存储桶和对象都是私有的。如果您希望第三方可以上传对象到存储桶，又不希望对方使用 CAM 账户或临时密钥等方式时，您可以使用预签名 URL 的方式将签名提交给第三方，以供完成临时的上传操作。收到有效预签名 URL 的任何人都可以上传对象。

预签名 URL 时，您可以在签名中设置将对象键包含在签名中，只许可上传指定路径。您还可以指定 HTTP 的请求方法，限制具体的对象操作，例如：上传、下载、删除等。您也可以在程序中指定预签名 URL 的有效时间，以保证超时后该 URL 不会被未授权方使用。

使用方法

使用 REST API

您可以直接使用 REST API 发起一个获取对象请求，可参考 GET Object 文档说明。

使用 Java SDK

对象存储 COS 的 Java SDK 中提供了此方法，可参考 Java SDK 接口文档生成预签名链接部分。

步骤说明

1. 初始化客户端 cosclient。
2. 执行 generatePresignedUrl 方法获取上传签名，并传入 http 方法参数为 PUT。

代码示例

以下代码示例演示了生成预签名的上传链接，并用其进行上传：

```
// 1 初始化用户身份信息(secretId, secretKey)
COSCredentials cred = new BasicCOSCredentials("AKIDXXXXXXXX", "1A2Z3YYYYYYYYYY");
// 2 设置bucket的区域, COS地域的简称请参照 /document/product/436/6224
ClientConfig clientConfig = new ClientConfig(new Region("ap-beijing-1"));
// 3 生成 cos 客户端
COSClient cosclient = new COSClient(cred, clientConfig);
// bucket名需包含appid
String bucketName = "mybucket-1251668577";

String key = "aaa.txt";
Date expirationTime = new Date(System.currentTimeMillis() + 30 * 60 * 1000);
// 生成预签名上传 URL
URL url = cosclient.generatePresignedUrl(bucketName, key, expirationTime, HttpMethodName.PUT);

// 使用预签名的 URL 上传文件
try {
HttpURLConnection connection = (HttpURLConnection) url.openConnection();
connection.setDoOutput(true);
connection.setRequestMethod("PUT");
OutputStreamWriter out = new OutputStreamWriter(connection.getOutputStream());
// 写入要上传的数据
out.write("This text uploaded as object.");
out.close();
int responseCode = connection.getResponseCode();
System.out.println("Service returned response code " + responseCode);
} catch (ProtocolException e) {
e.printStackTrace();
} catch (IOException e) {
e.printStackTrace();
}

cosclient.shutdown();
```

获取对象
简单获取对象

最近更新时间: 2025-01-15 17:01:00

适用场景

您可以直接发起请求获取 COS 中的对象，获取对象支持以下功能：

- 获取完整的单个对象：直接发起 GET 请求即可获取完整的对象数据。
- 获取单个对象的部分内容：可在 GET 请求中传入 `Range` 请求头部，支持检索一个特定的字节范围。不支持检索多个范围。

对象的元数据将会作为 HTTP 响应头部随对象内容一同返回，GET 请求支持使用 URL 参数的方式覆盖响应的部分元数据值，例如 `Content-Dispositon` 的响应值。支持修改的响应头部包括：

- Content-Type
- Content-Language
- Expires
- Cache-Control
- Content-Disposition
- Content-Encoding

使用方法

使用 REST API

您可以直接使用 REST API 发起一个获取对象请求，可参考 GET Object 文档说明。

使用 Java SDK

对象存储 COS 的 Java SDK 中提供了此方法，可参考 Java SDK 接口文档 Get Object 部分。

步骤说明

1. 初始化客户端 cosclient。
2. 执行 getObject 方法获取输入流或者将内容保存到本地。

代码示例

1. 以下代码演示了如何下载对象（无版本控制）：

```
// 1 初始化用户身份信息(appid, secretId, secretKey)
COSCredentials cred = new BasicCOSCredentials("1250000", "AKIDXXXXXXXX", "1A2Z3YYYYYYYYYY");
// 2 设置bucket的区域, COS地域的简称请参照 /document/product/436/6224
ClientConfig clientConfig = new ClientConfig(new Region("ap-beijing-1"));
// 3 生成cos客户端
COSClient cosclient = new COSClient(cred, clientConfig);
// 设置bucket名
String bucketName = "mybucket";
String key = "aaa.txt";

try {
// 下载文件
COSObject cosObject = cosclient.getObject(bucketName, key);
// 获取输入流
COSObjectInputStream cosObjectInput = cosObject.getObjectContent();
// 关闭输入流
cosObjectInput.close();
} catch (CosServiceException e) {
e.printStackTrace();
} catch (CosClientException e) {
e.printStackTrace();
}
```

2. `GetObjectRequest` 支持指定要从对象检索的数据字节范围，以下代码演示了指定字节的方法：

```
// 1 初始化用户身份信息(appid, secretId, secretKey)
COSCredentials cred = new BasicCOSCredentials("1250000", "AKIDXXXXXXXX", "1A2Z3YYYYYYYYYY");
// 2 设置bucket的区域, COS地域的简称请参照 /document/product/436/6224
ClientConfig clientConfig = new ClientConfig(new Region("ap-beijing-1"));
// 3 生成cos客户端
COSClient cosclient = new COSClient(cred, clientConfig);
// 设置bucket名
String bucketName = "mybucket";
String key = "aaa.txt";

try {
GetObjectRequest getObjectRequest = new GetObjectRequest(bucketName, key);
// 设置下载前11个字节
getObjectRequest.setRange(0, 10);
// 下载文件
COSObject cosObject = cosclient.getObject(bucketName, key);
// 获取输入流
COSObjectInputStream cosObjectInput = cosObject.getObjectContent();
// 关闭输入流
cosObjectInput.close();
} catch (CosServiceException e) {
e.printStackTrace();
} catch (CosClientException e) {
e.printStackTrace();
}
```

3. 检索对象时还可以用 `ResponseHeaderOverrides` 对象并设置相应的请求属性来替换响应头部值，以下是该方法的示例：

```
// 1 初始化用户身份信息(appid, secretId, secretKey)
COSCredentials cred = new BasicCOSCredentials("1250000", "AKIDXXXXXXXX", "1A2Z3YYYYYYYYYY");
// 2 设置bucket的区域, COS地域的简称请参照 /document/product/436/6224
ClientConfig clientConfig = new ClientConfig(new Region("ap-beijing-1"));
// 3 生成cos客户端
COSClient cosclient = new COSClient(cred, clientConfig);
// 设置bucket名
String bucketName = "mybucket";
String key = "aaa.txt";

try {
GetObjectRequest getObjectRequest = new GetObjectRequest(bucketName, key);
ResponseHeaderOverrides responseHeaders = new ResponseHeaderOverrides();
String responseContentType="image/x-icon";
String responseContentEncoding = "gzip,deflate,compress";
String responseContentLanguage = "zh-CN";
String responseContentDispositon = "filename=\"abc.txt\"";
String responseCacheControl = "no-cache";
String expireStr = DateUtils.formatRFC822Date(new Date(System.currentTimeMillis() + 24 * 3600 * 1000));
responseHeaders.setContentType(responseContentType);
responseHeaders.setContentEncoding(responseContentEncoding);
responseHeaders.setContentLanguage(responseContentLanguage);
responseHeaders.setContentDisposition(responseContentDispositon);
responseHeaders.setCacheControl(responseCacheControl);
responseHeaders.setExpires(expireStr);
getObjectRequest.setResponseHeaders(responseHeaders);
// 下载文件
COSObject cosObject = cosclient.getObject(bucketName, key);
// 获取输入流
COSObjectInputStream cosObjectInput = cosObject.getObjectContent();
// 关闭输入流
cosObjectInput.close();
} catch (CosServiceException e) {
e.printStackTrace();
} catch (CosClientException e) {
e.printStackTrace();
}
```

预签名授权下载

最近更新时间: 2025-01-15 17:01:00

适用场景

在默认情况下，存储桶和对象都是私有的。如果您希望第三方可以下载对象，又不希望对方使用 CAM 账户或临时密钥等方式时，您可以使用预签名 URL 的方式将签名提交给第三方，以供完成下载操作。收到有效预签名 URL 的任何人都可以下载对象。

预签名 URL 时，您可以在签名中设置将对象键包含在签名中，只许可下载指定的对象。您也可以在程序中指定预签名 URL 的有效时间，以保证超时后该 URL 不会被未授权方使用。

使用方法

使用 Java SDK

对象存储 COS 的 Java SDK 中提供了此方法，可参考 Java SDK 接口文档生成预签名链接部分。

步骤说明

1. 初始化客户端 cosclient。
2. 执行 generatePresignedUrl 方法获取下载签名，下载传入 http 方法为 GET。

代码示例

1. 以下代码演示了生成预签名的下载链接：

```
// 1 初始化用户身份信息(appid, secretId, secretKey)
COSCredentials cred = new BasicCOSCredentials("AKIDXXXXXXXX", "1A2Z3YYYYYYYYYY");
// 2 设置bucket的区域, COS地域的简称请参照 /document/product/436/6224
ClientConfig clientConfig = new ClientConfig(new Region("ap-beijing-1"));
// 3 生成cos客户端
COSClient cosclient = new COSClient(cred, clientConfig);
// 设置bucket名, bucket名需包含appid
String bucketName = "mybucket-125110000";
String key = "aaa.txt";

GeneratePresignedUrlRequest req =
new GeneratePresignedUrlRequest(bucketName, key, HttpMethodName.GET);
// 设置签名过期时间(可选), 最大允许设置签名一个月有效, 若未进行设置, 则默认使用ClientConfig中的签名过期时间(5分钟)
// 这里设置签名在半个小时后过期
Date expirationDate = new Date(System.currentTimeMillis() + 30 * 60 * 1000);
req.setExpiration(expirationDate);

URL url = cosclient.generatePresignedUrl(req);
System.out.println(url.toString());
```

2. `GeneratePresignedUrlRequest` 支持设置下载时返回的http头，比如 content-type, content-disposition 等，示例代码如下：

```
// 1 初始化用户身份信息(appid, secretId, secretKey)
COSCredentials cred = new BasicCOSCredentials("AKIDXXXXXXXX", "1A2Z3YYYYYYYYYY");
// 2 设置bucket的区域, COS地域的简称请参照 /document/product/436/6224
ClientConfig clientConfig = new ClientConfig(new Region("ap-beijing-1"));
// 3 生成cos客户端
COSClient cosclient = new COSClient(cred, clientConfig);
// 设置bucket名, bucket名需包含appid
String bucketName = "mybucket-125110000";
String key = "aaa.txt";

GeneratePresignedUrlRequest req =
new GeneratePresignedUrlRequest(bucketName, key, HttpMethodName.GET);
// 设置下载时返回的http头
ResponseHeaderOverrides responseHeaders = new ResponseHeaderOverrides();
String responseContentType = "image/x-icon";
String responseContentLanguage = "zh-CN";
```

```
String responseContentDispositon = "filename=\"abc.txt\"";
String responseCacheControl = "no-cache";
String expireStr =
DateUtils.formatRFC822Date(new Date(System.currentTimeMillis() + 24 * 3600 * 1000));
responseHeaders.setContentType(responseContentType);
responseHeaders.setContentLanguage(responseContentLanguage);
responseHeaders.setContentDisposition(responseContentDispositon);
responseHeaders.setCacheControl(responseCacheControl);
responseHeaders.setExpires(expireStr);
req.setResponseHeaders(responseHeaders);
URL url = cosclient.generatePresignedUrl(req);

System.out.println(url.toString());
```

3. `GeneratePresignedUrlRequest` 同时支持生成匿名 bucket 的下载链接，匿名 bucket 下载链接无需包含签名，因此无需传入秘钥信息。示例代码如下：

```
// 1 对于匿名bucket, 无需传入身份信息
COSCredentials cred = new AnonymousCOSCredentials();
// 2 设置bucket的区域, COS地域的简称请参照 /document/product/436/6224
ClientConfig clientConfig = new ClientConfig(new Region("ap-beijing-1"));
// 3 生成cos客户端
COSClient cosclient = new COSClient(cred, clientConfig);
// 设置bucket名, bucket名需包含appid
String bucketName = "mybucket-125110000";
String key = "aaa.txt";

GeneratePresignedUrlRequest req =
new GeneratePresignedUrlRequest(bucketName, key, HttpMethodName.GET);
URL url = cosclient.generatePresignedUrl(req);

System.out.println(url.toString());
```

列出对象键

最近更新时间: 2025-01-15 17:01:00

适用场景

云平台 COS 支持按照前缀顺序列出对象键，您也可以在对象键中使用 `/` 字符来实现类似传统文件系统的层级结构，COS 也支持按照分隔符来做层级结构的选择和浏览。

您可以列出单个存储桶中的所有对象键，根据前缀的 UTF-8 二进制顺序列出，或选择指定前缀过滤对象键的列表。例如加入参数 `t` 将列出 `tapd` 的对象，而跳过以 `a` 或其他字符为前缀的对象。

加入 `/` 分隔符可将根据此分隔符重新组织对象键，您可以结合前缀和分隔符来实现类似文件夹检索的功能。例如加入前缀参数 `t` 并加入分隔符 `/` 将会直接列出类似 `tapd/file` 的对象键。

云平台 COS 在单个存储桶中支持无限数量的对象，因此对象键列表可能非常大。为了管理方便，单个列出对象接口将最多返回 1000 个键值的结果内容，同时会返回指示器来告知是否存在截断。您可以根据指示器和分隔符来发送一系列的列出对象键请求，实现列出所有键值，或寻找您所需要的内容。

使用方法

使用 REST API

您可以直接使用 REST API 发起一个获取对象请求，可参考 Get Bucket 文档说明。

使用 Java SDK

对象存储 COS 的 Java SDK 中提供了此方法，可参考 Java SDK 接口文档 Get Bucket (List Objects) 部分。

步骤说明

1. 初始化客户端 cosclient。
2. 使用 listObjects 列出 object，每次最多列出 1000 个 object，如果需要列出所有的或者超过 1000 个, 则需要循环调用 listObjects。

代码示例

1. `ListObjectsRequest` 包含了列出 Object 的请求, 可设置列出的 Object 的前缀, 分隔符, 示例代码如下:

```
// 1 初始化用户身份信息(secretId, secretKey)
COSCredentials cred = new BasicCOSCredentials("AKIDXXXXXXXX", "1A2Z3YYYYYYYYYY");
// 2 设置bucket的区域, COS地域的简称请参照 /document/product/436/6224
ClientConfig clientConfig = new ClientConfig(new Region("ap-beijing-1"));
// 3 生成cos客户端
COSClient cosclient = new COSClient(cred, clientConfig);
// bucket名需包含appid
String bucketName = "mybucket-1251668577";

ListObjectsRequest listObjectsRequest = new ListObjectsRequest();
// 设置bucket名称
listObjectsRequest.setBucketName(bucketName);
// prefix表示列出的object的key以prefix开始
listObjectsRequest.setPrefix("aaa/bbb");
// deliter表示分隔符, 设置为/表示列出当前目录下的object, 设置为空表示列出所有的object
listObjectsRequest.setDelimiter("");
// 如果object的路径中含有特殊字符, 建议使用url编码方式, 得到object的key后, 需要进行url decode
listObjectsRequest.setEncodingType("url");
// 设置最大遍历出多少个对象, 一次listobject最大支持1000
listObjectsRequest.setMaxKeys(1000);
ObjectListing objectListing = null;
try {
objectListing = cosclient.listObjects(listObjectsRequest);
} catch (CosServiceException e) {
e.printStackTrace();
} catch (CosClientException e) {
e.printStackTrace();
}
// common prefix表示表示被delimiter截断的路径, 如delimter设置为/, common prefix则表示所有子目录的路径
List<String> commonPrefixs = objectListing.getCommonPrefixes();
```

```
// object summary表示所有列出的object列表
List<COSObjectSummary> cosObjectSummaries = objectListing.getObjectSummaries();
for (COSObjectSummary cosObjectSummary : cosObjectSummaries) {
// 文件的路径key
String key = cosObjectSummary.getKey();
// 如果使用的encodingtype是url, 则进行url decode
try {
key = URLDecoder.decode(key, "utf-8");
} catch (UnsupportedEncodingException e) {
continue;
}
// 文件的etag
String etag = cosObjectSummary.getETag();
// 文件的长度
long fileSize = cosObjectSummary.getSize();
// 文件的存储类型
String storageClasses = cosObjectSummary.getStorageClass();
}

cosclient.shutdown();
```

2. 如果要获取超过 maxkey 数量的 Object 或者获取所有的 Object, 则需要循环调用 listobject, 用上一次返回的 next marker 作为下一次调用的 marker, 直到返回的 truncated 为 false。

```
// 1 初始化用户身份信息(secretId, secretKey)
COSCredentials cred = new BasicCOSCredentials("AKIDXXXXXXXX", "1A2Z3YYYYYYYYYY");
// 2 设置bucket的区域, COS地域的简称请参照 /document/product/436/6224
ClientConfig clientConfig = new ClientConfig(new Region("ap-beijing-1"));
// 3 生成cos客户端
COSClient cosclient = new COSClient(cred, clientConfig);
// bucket名需包含appid
String bucketName = "mybucket-1251668577";

ListObjectsRequest listObjectsRequest = new ListObjectsRequest();
// 设置bucket名称
listObjectsRequest.setBucketName(bucketName);
// prefix表示列出的object的key以prefix开始
listObjectsRequest.setPrefix("aaa/bbb");
// deliter表示分隔符, 设置为/表示列出当前目录下的object, 设置为空表示列出所有的object
listObjectsRequest.setDelimiter("");
// 如果object的路径中含有特殊字符, 建议使用url编码方式, 得到object的key后, 需要进行url decode
listObjectsRequest.setEncodingType("url");
// 设置最大遍历出多少个对象, 一次listobject最大支持1000
listObjectsRequest.setMaxKeys(1000);
ObjectListing objectListing = null;
do {

try {
objectListing = cosclient.listObjects(listObjectsRequest);
} catch (CosServiceException e) {
e.printStackTrace();
return;
} catch (CosClientException e) {
e.printStackTrace();
return;
}
// common prefix表示表示被delimiter截断的路径, 如delimter设置为/, common prefix则表示所有子目录的路径
List<String> commonPrefixs = objectListing.getCommonPrefixes();

// object summary表示所有列出的object列表
List<COSObjectSummary> cosObjectSummaries = objectListing.getObjectSummaries();
for (COSObjectSummary cosObjectSummary : cosObjectSummaries) {
// 文件的路径key
String key = cosObjectSummary.getKey();
// 如果使用的encodingtype是url, 则进行url decode
try {
key = URLDecoder.decode(key, "utf-8");
} catch (UnsupportedEncodingException e) {
continue;
}
```

```
// 文件的etag
String etag = cosObjectSummary.getETag();
// 文件的长度
long fileSize = cosObjectSummary.getSize();
// 文件的存储类型
String storageClasses = cosObjectSummary.getStorageClass();
}

// 获取下一次请求的next marker
String nextMarker = "";
try {
nextMarker = URLDecoder.decode(objectListing.getNextMarker(), "utf-8");
} catch (UnsupportedEncodingException e) {
e.printStackTrace();
return;
}
listObjectsRequest.setMarker(nextMarker);
} while (objectListing.isTruncated());

cosclient.shutdown();
```

复制对象
简单复制

最近更新时间: 2025-01-15 17:01:00

适用场景

您可以在 COS 中将已存储的对象通过简单的复制操作，创建一个新的对象副本。在单个操作中，您可以复制最大 5 GB 的对象；当对象超过 5 GB 时，您必须使用分块上传的接口来实现复制。复制对象有以下功能：

- 创建一个新的对象副本。
- 复制对象并更名，删除原始对象，实现重命名。
- 修改对象的存储类型，在复制时选择相同的源和目标对象键，修改存储类型。
- 修改对象的元数据，在复制时选择相同的源和目标对象键，并修改其中的元数据。

复制对象时，默认将继承原对象的元数据，但创建日期将会按新对象的时间计算。

使用方法

使用 REST API

您可以直接使用 REST API 发起一个获取对象请求，可参考 Put Object Copy 文档说明。

使用 Java SDK

对象存储 COS 的 Java SDK 中提供了此方法，可参考 Java SDK 接口文档 Put Object Copy 部分。

步骤说明

1. 初始化客户端 cosclient。
2. 使用 copyObject 接口来完成 copy。

代码示例

`CopyObjectRequest` 包含了 copy 对象的请求，通过设置源文件所在的园区，bucket 名称，路径以及目的文件的园区，bucket名称，路径。下列的代码示例演示了如何简单复制对象：

```
// 1 初始化用户身份信息(secretId, secretKey)
COSCredentials cred = new BasicCOSCredentials("AKIDXXXXXXXX", "1A2Z3YYYYYYYYYY");
// 2 设置bucket的区域, COS地域的简称请参照 /document/product/436/6224
ClientConfig clientConfig = new ClientConfig(new Region("ap-beijing-1"));
// 3 生成cos客户端
COSClient cosclient = new COSClient(cred, clientConfig);
// 要拷贝的bucket region, 支持跨园区拷贝
Region srcBucketRegion = new Region("ap-shanghai");
// 源bucket, bucket名需包含appid
String srcBucketName = "srcBucket-1251668577";
// 要拷贝的源文件
String srcKey = "aaa/bbb.txt";
// 目的bucket, bucket名需包含appid
String destBucketName = "destBucket-1251668577";
// 要拷贝的目的文件
String destKey = "ccc/ddd.txt";

CopyObjectRequest copyObjectRequest = new CopyObjectRequest(srcBucketRegion, srcBucketName,
srcKey, destBucketName, destKey);
try {
CopyObjectResult copyObjectResult = cosclient.copyObject(copyObjectRequest);
} catch (CosServiceException e) {
e.printStackTrace();
} catch (CosClientException e) {
e.printStackTrace();
}
cosclient.shutdown();
```

分块复制

最近更新时间: 2025-01-15 17:01:00

适用场景

当需要复制一个超过 5 GB 的对象时，您需要选择分块复制的方法来实现。使用分块上传的 API 来创建一个新的对象，并使用 Part Copy 的功能，携带 `x-cos-copy-source` 头部来指定源对象，流程包括：

1. 初始化一个分块上传的对象。
2. 复制源对象的数据，可指定 `x-cos-copy-range` 头部，每次只可复制最多 5 GB 数据。
3. 完成分块上传。

使用云平台 COS 提供的 SDK 可以轻松完成分块复制的功能。

使用方法

使用 Java SDK

对象存储 COS 的 Java SDK 中提供了此方法，可参考 Java SDK 接口文档拷贝文件部分 。

步骤说明

1. 初始化客户端 cosclient。
2. 使用 TransferManager 中提供的高级 API copy 接口来完成拷贝。

代码示例

对于 5G 以上的文件，需要通过分块上传中的 copypart 来实现，步骤较多，因此在 TransferManager 中封装了一个 copy 接口，不仅能根据文件大小自动的选择接口，同时能支持 5G 以上的文件拷贝。推荐使用该接口进行文件的 copy。示例代码如下：

```
// 1 初始化用户身份信息(secretId, secretKey)
COSCredentials cred = new BasicCOSCredentials("AKIDXXXXXXXX", "1A2Z3YYYYYYYYYY");
// 2 设置bucket的区域, COS地域的简称请参照 /document/product/436/6224
ClientConfig clientConfig = new ClientConfig(new Region("ap-beijing-1"));
// 3 生成cos客户端
COSClient cosclient = new COSClient(cred, clientConfig);


ExecutorService threadPool = Executors.newFixedThreadPool(32);
// 传入一个threadpool, 若不传入线程池, 默认TransferManager中会生成一个单线程的线程池。
TransferManager transferManager = new TransferManager(cosclient, threadPool);

// 要拷贝的bucket region, 支持跨园区拷贝
Region srcBucketRegion = new Region("ap-shanghai");
// 源bucket, bucket名需包含appid
String srcBucketName = "srcBucket-1251668577";
// 要拷贝的源文件
String srcKey = "aaa/bbb.txt";
// 目的bucket, bucket名需包含appid
String destBucketName = "destBucket-1251668577";
// 要拷贝的目的文件
String destKey = "ccc/ddd.txt";

CopyObjectRequest copyObjectRequest = new CopyObjectRequest(srcBucketRegion, srcBucketName,
srcKey, destBucketName, destKey);
try {
Copy copy = transferManager.copy(copyObjectRequest);
// 返回一个异步结果copy, 可同步的调用waitForCopyResult等待copy结束, 成功返回CopyResult, 失败抛出异常.
CopyResult copyResult = copy.waitForCopyResult();
} catch (CosServiceException e) {
e.printStackTrace();
} catch (CosClientException e) {
e.printStackTrace();
} catch (InterruptedException e) {
```

```
e.printStackTrace();
}

transferManager.shutdownNow();
cosclient.shutdown();
```

删除对象

删除单个对象

最近更新时间: 2025-01-15 17:01:00

适用场景

云平台 COS 支持直接删除一个或多个对象，当仅需要删除一个对象时，您只需要提供对象的名称（即对象键），就可以调用一个 API 请求来删除它。

使用方法

使用 REST API

您可以直接使用 REST API 发起一个获取对象请求，可参考 Delete Object 文档说明。

使用 Java SDK

对象存储 COS 的 Java SDK 中提供了此方法，可参考 Java SDK 接口文档 Delete Object 部分。

步骤说明

1. 初始化客户端 cosclient。
2. 执行 deleteObject 方法删除对象，传入 bucketName 和要删除的 key。

代码示例

调用 deleteObject 创建 object，代码示例如下所示：

```
// 1 初始化用户身份信息(secretId, secretKey)
COSCredentials cred = new BasicCOSCredentials("AKIDXXXXXXXX", "1A2Z3YYYYYYYYYY");
// 2 设置bucket的区域, COS地域的简称请参照 /document/product/436/6224
ClientConfig clientConfig = new ClientConfig(new Region("ap-beijing-1"));
// 3 生成cos客户端
COSClient cosclient = new COSClient(cred, clientConfig);
// bucket名需包含appid
String bucketName = "mybucket-1251668577";
String key = "/aaa/bbb.txt";
try {
cosclient.deleteObject(bucketName, key);
} catch (CosServiceException e) {
e.printStackTrace();
} catch (CosClientException e) {
e.printStackTrace();
}

// 关闭客户端
cosclient.shutdown();
```

数据管理
生命周期管理
生命周期概述

最近更新时间: 2025-01-15 17:01:00

对象存储 COS 支持基于对象的生命周期配置，其通过对存储桶下发指定的描述语言，可以让符合规则的对象在指定的条件下自动执行一些操作。

> 说明：生命周期的设置支持最长天数为3650天。

适用场景

日志记录

如果用户使用对象存储来存储日志数据，可以通过生命周期配置，使得日志数据在2年后自动删除。

支持说明

支持的操作

- 过期删除：设置对象的过期时间，使对象到期后被自动删除。

支持的资源

- 按前缀区分：匹配前缀规则的对象都会按照规则执行处理。
- 按版本管理：非当前版本的对象将会按照规则执行处理。
- 按删除标记：对象历史版本都清除时，可以指定移除删除标记。
- 按未完成分块上传：对未完成的分块上传任务执行处理。

支持的时间条件

- 按天计算：指明规则对应的动作，在对象最后被修改的日期过后多少天操作。
- 按日期计算：指明规则对应的动作在指定的日期执行操作。

注意事项

过期删除

处理逻辑

当对象匹配了指定的生命周期过期删除的规则时，腾讯云金融专区会将对象加入异步的删除队列，实际发生的删除时间将会与创建时间有一定的延时。您将可以通过 GET 或 HEAD Object 操作来获取对象的当前状态。

最终一致性

如果对同一组的对象配置了多条规则，且存在冲突性情况，对象存储会以最短过期时间为准执行，且**过期删除的执行效力大于转换存储类型**。

> 注意：
>
> COS 强烈提醒您不要针对同一组对象配置多个含冲突条件的生命周期规则，冲突执行可能导致不同的费用表现。

成本注意

执行说明

对于以任何时间下发的配置，COS 都将以北京时间（GMT+8）次日的0时为准开始执行操作，由于是异步队列执行，因此对于设置后上传的对象匹配规则的，通常最晚于次日的24时前完成操作。

生命周期执行效力不包含意外情况或存储桶中包含大量存量对象的情况，若因为其他情况没有完成，您将可以通过 GET 或 HEAD Object 操作来获取对象的当前状态。

对于生命周期的执行效力不提供账单承诺，即对象的计费将会在生命周期执行完成时发生改变。

不受大小限制

COS 不会检查文件的大小，将无条件按照指定的规则，执行对象的转换操作。

生命周期配置元素

最近更新时间: 2025-01-15 17:01:00

基本结构

生命周期配置使用 XML 描述方法，其可以配置一条或多条生命周期规则，基本结构如下：

```
<LifecycleConfiguration>
<Rule>
<ID>**your lifecycle name**</ID>
<Status>Enabled</Status>
<Filter>
<And>
<Prefix>projectA/</Prefix>
<Tag>
<Key>key1</Key>
<Value>value1</Value>
</Tag>
</And>
</Filter>
**transition/expiration actions**
</Rule>
<Rule>
...
</Rule>
</LifecycleConfiguration>
```

其中每一条规则包含如下内容：

- ID（可选）：可自定义的描述规则的内容。
- Status：可选择规则启用 Enabled 或禁用 Disabled 的状态。
- Filter：用于指定需要操作的对象的筛选条件。
- 操作：需要对符合以上描述的对象执行的操作。
- 时间：支持根据最后修改时间指定天数 Days，或指定某个具体的日期 Date 前修改的对象。

规则描述

Filter 元素

针对存储桶中的所有对象

指定空的筛选条件，将会应用于存储桶中的所有对象。

```
<LifecycleConfiguration>
<Rule>
<Filter>
</Filter>
<Status>Enabled</Status>
**transition/expiration actions**
</Rule>
</LifecycleConfiguration>
```

针对指定的对象键前缀

指定对象前缀，可以对一部分符合前缀描述的对象组执行操作，例如设置以 logs/ 为前缀的所有对象。

```
<LifecycleConfiguration>
<Rule>
<Filter>
<Prefix>logs/</Prefix>
</Filter>
<Status>Enabled</Status>
**transition/expiration actions**
</Rule>
</LifecycleConfiguration>
```

针对指定的对象标签

指定某些符合对象标签的 key 和 value 为筛选条件，对特定标签的对象执行操作，例如设置标签的 key=type 和 value=image 为筛选条件的对象：

```
<LifecycleConfiguration>
<Rule>
<Filter>
<Tag>
```

```
<Key>type</Key>
<Value>image</Value>
</Tag>
</Filter>
<Status>Enabled</Status>
**transition/expiration actions**
</Rule>
</LifecycleConfiguration>
```

合并使用多个筛选条件

COS 支持通过 AND 的逻辑来合并使用多个筛选条件，例如设置以 logs/ 为前缀，同时对象标签的 key=type 和 value=image 为筛选条件的对象：

```
<LifecycleConfiguration>
<Rule>
<Filter>
<And>
<Prefix>logs/</Prefix>
<Tag>
<Key>type</Key>
<Value>image</Value>
</Tag>
</And>
</Filter>
<Status>Enabled</Status>
**transition/expiration actions**
</Rule>
</LifecycleConfiguration>
```

操作元素

在生命周期规则中，可以对符合条件的一组对象执行一个或多个操作。

转换操作

指定 Transition 操作可以使对象从一个存储类型转换到另一个存储类型，如果存储桶开启了版本控制，其只对当前版本执行操作。最短的 Transition 设置时间为0天。例如，设置30天后沉降至归档存储：

```
<Transition>
<StorageClass>ARCHIVE</StorageClass>
<Days>30</Days>
</Transition>
```

过期删除

指定 Expiration 操作可以使符合规则的对象执行过期删除操作，如果存储桶从未启用过版本控制，则将永久删除对象。如果存储桶启用过版本控制，则将为过期的对象**添加一个 DeleteMarker 标记**，并将其设置为当前版本。例如，设置30天后删除对象：

```
<Expiration>
<Days>30</Days>
</Expiration>
```

未完成的分块上传

指定 AbortIncompleteMultipartUpload 操作可以允许分块上传的指定 UploadId 任务在保持一段时间后删除，其不再提供续传或可被检索的特性。例如，设置7天后清除未完成的分块上传任务：

```
<AbortIncompleteMultipartUpload>
<Days>7</Days>
</AbortIncompleteMultipartUpload>
```

非当前版本的对象

在启用过版本控制的存储桶中，转换只会对最新版本执行，过期操作只会添加删除标记，因此对象存储 COS 对非当前版本的推向提供了如下操作：

指定 NoncurrentVersionTransition 可以将非当前版的对象在指定时间转换到另一个存储类型。例如，设置历史版本在30天后沉降至归档存储：

```
<NoncurrentVersionTransition>
<StorageClass>ARCHIVE</StorageClass>
<Days>30</Days>
</NoncurrentVersionTransition>
```

指定 NoncurrentVersionExpiration 可以将非当前版本的对象在指定时间内过期删除。例如，设置历史版本在30天后删除：

```
<NoncurrentVersionExpiration>
<Days>30</Days>
</NoncurrentVersionExpiration>
```

指定 ExpiredObjectDeleteMarker 可以在对象键的所有历史版本都已经删除，且最新的对象版本是删除标记 DeleteMarker 时，清除掉这个删除标记。例如，设置31天后移除过期对象的删除标记：

```
<ExpiredObjectDeleteMarker>
<Days>31</Days>
</ExpiredObjectDeleteMarker>
```

时间元素

按天数计算

使用 Days 指定天数，是按照对象的最后修改时间来计算的。

- 例如，设置一个对象在0天后转换为归档存储类型，对象于2018-01-01 23:55:00 GMT+8上传，则其将于2018-01-02 00:00:00 GMT+8起进入转换存储类型的处理队列，最晚于2018-01-02 23:59:59 GMT+8前完成转换。
- 例如，设置一个对象在1天后过期删除，对象于2018-01-01 23:55:00 GMT+8上传，则其将于2018-01-03 00:00:00 GMT+8起进入过期删除的处理队列，最晚于2018-01-03 23:59:59 GMT+8前完成删除。

按指定日期

使用 Date 指定日期，将会在到达特定日期时，对符合筛选条件的所有对象执行该操作。目前仅支持指定 GMT+8 时区，设置为0时的 ISO8601 格式的时间。 例如，2018 年01月01日，描述为：2018-01-01T00:00:00+08:00。

配置生命周期

最近更新时间: 2025-01-15 17:01:00

适用场景

利用生命周期设置，可以让符合规则的对象在指定的条件下自动执行一些操作。例如：

- 转换存储类型：将创建的对象在指定时间后转换为低频存储类型 STANDARD_IA 或者归档存储类型 ARCHIVE。
- 过期删除：设置对象的过期时间，使对象到期后被自动删除。

详情请参见 生命周期概述 文档和 生命周期配置元素 文档。

使用方法

使用对象存储控制台

您可以使用对象存储控制台配置生命周期，详情请参见 设置生命周期 控制台指南文档。

使用 REST API

您可以直接使用 REST API 配置和管理存储桶中对象的生命周期，详情请参见以下 API 文档：

- PUT Bucket lifecycle
- GET Buket lifecycle
- DELETE Bucket lifecycle

托管静态网站

最近更新时间: 2025-01-15 17:01:00

基本概念

静态网站指包含静态内容（例如 HTML）或客户端脚本的网站，用户可以通过控制台对已绑定自定义域名的存储桶，配置静态网站。而动态网站的内容包含诸如 PHP、JSP 或 ASP.NET 等服务器端脚本，需要依赖服务器端处理。COS 支持静态网站的托管，不支持服务器端脚本编写。当您需要部署动态网站时，推荐使用云服务器 CVM 进行服务端代码部署。

示例

用户创建了名为 examplebucket-1250000000 的存储桶，上传了如下文件：

```
index.html
404.html
403.html
test.html
docs/a.html
images/
```

静态网站

开启前：使用如下默认访问域名访问存储桶，弹出下载提示，可以保存 `index.html` 文件到本地。

```
http://imgcache.finance.cloud.tencent.com:80examplebucket-1250000000.cos-website.ap-guangzhou.myqcloud.com/index.html
```

开启后：使用如下访问节点访问存储桶，可以直接在浏览器中查看 `index.html` 的页面内容。

```
http://imgcache.finance.cloud.tencent.com:80examplebucket-1250000000.cos-website.ap-guangzhou.myqcloud.com/index.html
```

索引文档

索引文档即静态网站的首页，是当用户对网站的根目录或任何子目录发出请求时返回的网页，通常此页面被命名为 `index.html`。 当用户使用存储桶访问域名（例如 `http://imgcache.finance.cloud.tencent.com:80examplebucketbucket-1250000000.cos-website.ap-guangzhou.myqcloud.com` ）访问静态网站时，且未请求特定的页面。在这种情况下，Web 服务器将返回首页。

您的用户访问存储桶包括根目录在内的任何目录，URL 地址以 `/` 为结尾的，会优先自动匹配该目录下的索引文档。根级 URL 的 `/` 是可选的，以下任意一个 URL 将返回索引文档。

```
http://imgcache.finance.cloud.tencent.com:80www.examplebucket.com/
http://imgcache.finance.cloud.tencent.com:80www.examplebucket.com
```

> **注意：**
>
> 如果存储桶中创建了文件夹，则需要在每个文件夹层级上都添加索引文档。

错误文档

假设您在配置错误文档前，访问以下页面，将返回404状态码，页面上显示为默认的错误页面信息。

```
http://imgcache.finance.cloud.tencent.com:80examplebucket-1250000000.cos-website.ap-guangzhou.myqcloud.com/webpage.html
```

配置错误文档后，访问以下页面，同样返回404状态码，但页面上将显示您所指定的错误页面信息。

```
http://imgcache.finance.cloud.tencent.com:80examplebucket-1250000000.cos-website.ap-guangzhou.myqcloud.com/webpage.html
```

重定向规则

> **说明：**
>
> 托管静态网站配置重定向规则，替换文档路径必须是存储桶内的对象路径。

配置错误码重定向

假设您为 webpage.html 这个文档设置了**私有读写**的公共访问权限，用户访问该文件时，将返回403错误。 配置403错误码重定向至 403.html 后：浏览器将返回 403.html 的内容。 如果您未配置 403.html 文档，浏览器将返回错误文档或默认错误信息。

配置前缀匹配

1. 当您将文件夹从docs/重命名为documents/后，用户在访问docs/文件夹会产生错误。所以，您可以将前缀docs/的请求重定向至 documents/。

2. 当您删除了 `images/` 文件夹（即删除了具有前缀 `images/` 的所有对象）。您可以添加重定向规则，将具有前缀 `images/` 的任何对象的请求重定向至 `test.html` 页面。

存储桶标签概述

最近更新时间: 2025-01-15 17:01:00

概述

存储桶标签是一个键值对（key = value），由标签的键（key）和标签的值（value）与“=”相连组成，例如 group = IT。它可以作为管理存储桶的一个标识，便于用户对存储桶进行分组管理。您可以对指定的存储桶进行标签的设定、查询和删除操作。

规格与限制

标签键限制

- 以 qcs:、project、项目等开头的标签键为系统预留标签键，系统预留标签键禁止创建。
- 支持 UTF-8 格式表示的字符、空格和数字以及特殊字符 `+ - = . _ : / @` 。
- 标签键长度为0 - 127个字符（采用 UTF-8 格式）。
- 标签键区分英文字母大小写。

标签值限制

- 支持 UTF-8 格式表示的字符、空格和数字以及特殊字符 `+ - = . _ : / @` 。
- 标签值长度为0 - 255个字符 （采用 UTF-8 格式）。
- 标签值区分英文字母大小写。

标签数量限制

- 存储桶维度：一个资源最多50个不同的存储桶标签。
- 标签维度：
- 单个用户最多1000个不同的 key。
- 一个 key 最多有1000个 value。
- 同个存储桶下不允许有多个相同的 key。

使用方法

您可以通过控制台、API 的方式设置存储桶标签。

使用对象存储控制台

您如需使用对象存储控制台设置存储桶标签，请参见 设置存储桶标签 控制台指南文档。

使用 REST API

您可以直接通过以下 API 管理存储桶标签：

- PUT Bucket tagging
- GET Bucket tagging
- DELETE Bucket tagging

日志管理
日志管理概述

最近更新时间: 2025-01-15 17:01:00

简介

日志管理功能能够记录对于指定源存储桶的详细访问信息，并将这些信息以日志文件的形式保存在指定的存储桶中，以实现对存储桶更好的管理。

在目标存储桶中，日志记录路径为：

```
目标存储桶/路径前缀{YYYY}/{MM}/{DD}/{time}_{random}_{index}
```

日志每5分钟生成一次，一条记录为一行，每条记录包含多个字段，字段之间以空格分割。需要注意的是，单个日志文件最大为256MB，如果您在这5分钟内产生的日志量超过256MB，那您的日志会被分割成多份日志文件。目前支持的日志字段如下：

字段序号	名 称	含 义	示例
1	eventVersion	记录版本	1.0
2	bucketName	存储桶名称	examplebucket-1250000000
3	qcsRegion	请求地域	ap-beijing
4	eventTime	事件时间（请求结束时间，UTC 0时 时间戳）	2018-12-01T11:02:33Z
5	eventSource	用户访问的域名	examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com
6	eventName	事件名称	UploadPart
7	remoteIp	来源 IP	192.168.0.1
8	userSecretKeyId	用户访问 KeyId	AKIDNYVCdoJQyGJ5brTf
9	reservedFiled	保留字段	保留字段，显示为`-`。
10	reqBytesSent	请求字节数（Bytes）	83886080
11	deltaDataSize	请求对存储量的改变（Bytes）	808
12	reqPath	请求的文件路径	/folder/text.txt
13	reqMethod	请求方法	put
14	userAgent	用户 UA	cos-go-sdk-v5.2.9
15	resHttpCode	HTTP 返回码	404
16	resErrorCode	错误码	NoSuchKey
17	resErrorMsg	错误信息	The specified key does not exist.
18	resBytesSent	返回字节数（Bytes）	197
19	resTotalTime	请求总耗时（毫秒，等于响应末字节的时间-请求首字节的时间）	4295
20	logSourceType	日志源类型	USER（用户访问请求）
21	storageClass	存储类型	STANDARD，STANDARD_IA，ARCHIVE
22	accountId	存储桶所有者 ID	100000000001
23	resTurnAroundTime	请求服务端耗时（毫秒，等于响应首字节的时间-请求末字节的时间）	4295

字段序号	名 称	含 义	示例
24	requester	访问者	主账号 ID：子账号 ID，如果是匿名访问则显示`-`。
25	requestId	请求 ID	NWQ1ZjY4MTBfMjZiMjU4NjRfOWI1N180NDBiYTY=
26	objectSize	对象大小（Bytes）	808，如果您使用分块上传，objectSize 字段只会在完成上传的时候显示，各个分块上传期间该字段显示`-`
27	versionId	对象版本 ID	随机字符串
28	targetStorageClass	目标存储类型，发起复制操作的请求会记录该字段	STANDARD，STANDARD_IA，ARCHIVE
29	referer	请求的 HTTP referer	`*.example.com`或者111.111.111.1
30	requestUri	请求 URI	"GET /fdgfdgsf%20/%E6%B5%AE%E7%82%B9%E6%95%B0 HTTP/1.1"
31	vpcId	请求是否是 VPC 请求	0/1

> **注意：**
>
> 存放日志的目标存储桶可以是源存储桶本身，但不推荐。
>
> 根据用户需求和业务发展情况，COS 可能在访问日志中新增字段，请在解析日志时务必做好相应处理。

启用日志管理

用户可以通过租户端控制台快速开启日志管理功能。

日志管理限制

最近更新时间: 2025-01-15 17:01:00

日志管理功能够针对指定源存储桶记录其详细访问信息，并将这些信息以日志文件的形式保存在指定的存储桶中，以实现对存储桶更好的管理。

目前访问日志管理功能的使用限制如下：

- 投递频率限制：日志每5分钟生成一次。
- 投递日志文件大小限制：每次投递的日志文件最大为256MB，超过该限制时将以新的文件投递。
- 投递日志格式：一条记录为一行，每条记录包含多个字段，字段之间以空格分割。
- 投递字段限制：有关投递字段，请参见“日志管理概述”中的说明。
- 无效字段说明：如果您的日志中存在-这一字符，说明该字段为无效或者缺省记录。

日志记录的内容

用户发起的上传、下载、删除对象，以及创建、删除存储桶，修改存储桶配置等请求。

日志不记录的内容

- 离线回源请求：用户配置回源后，如果 COS 中无对象，将会到用户指定的源站下载数据，这一下载操作即离线回源请求，不会被记录到日志中。
- 静态网站内部重定向操作：用户在静态网站功能中配置了重定向时，如果访问 index.html 可能会被重定向到其他页面，这类重定向操作不会被记录到日志中。
- 生命周期沉降、删除等操作：如果用户设置了生命周期功能，对对象进行过期沉降或者删除，这些沉降和删除操作由 COS 后端实现，不会记录到日志中。
- 清单列出对象和上传清单报告操作：清单功能代替用户周期性列出存储桶内全部或者指定对象，并将生成的清单报告投递到用户存储桶中，列出对象和投递清单报告的操作将不会被记录到日志中。
- 跨地域复制对象操作：跨地域复制功能需要从源存储桶中获取对象并将对象上传到目标存储桶中，这些操作由 COS 后端实现，不会记录到日志中。
- COS Select 功能中下载对象的操作：COS Select 功能协助用户检索对象，需要先从存储设备中获取数据后才能进行检索，下载对象的操作由 COS 后端实现，不会记录到日志中。

版权所有：腾讯云金融专区

数据容灾
版本控制
版本控制概述

最近更新时间: 2025-01-15 17:01:00

简介

版本控制用于实现在相同存储桶中存放同一对象的多个版本。例如，在一个存储桶中，您可以存放多个对象键同为 picture.jpg 的对象，但其版本 ID 不同，例如100000、100101和120002等。用户在为某一存储桶开启版本控制功能后，可以根据版本 ID 查询、删除或还原存放在存储桶中的对象。这有助于恢复被用户误删或应用程序故障而丢失的数据。例如，用户在对版本控制的对象进行删除操作时：

- 如果需要删除对象（非完全删除），COS 会为被删除的对象插入删除标记，该标记将作为当前对象版本，您可以根据**删除标记**恢复以前的版本。
- 如果需要替换对象，对象存储会为新上传的对象插入新的版本 ID，您仍然可以根据版本 ID 恢复被替换前的对象。

版本控制状态

存储桶可处于三种版本控制状态：未启用版本控制状态、启用版本控制状态和暂停版本控制状态。

- **未启用版本控制状态：**指存储桶的默认初始状态，此时版本控制功能关闭。
- **启用版本控制状态：**指开启存储桶版本控制功能，此时为版本控制开启状态，版本控制状态将应用到该存储桶中的**所有对象**。您对存储桶首次启用版本控制后，该存储桶中新上传的对象将拥有唯一的版本 ID。
- **暂停版本控制状态：**指存储桶的版本控制由开启状态变为暂停状态（无法返回未启用版本控制状态），此后往存储桶中上传的对象将不再存放版本控制的对象。

> 注意：
>
> 1. 一旦您对存储桶启用了版本控制，它将无法返回到未启用版本控制状态（初始状态）。但是，您可以对该存储桶暂停版本控制，这样后续新上传的对象将不会产生多个版本。
> 2. 在启用版本控制之前，存储在存储桶中的对象的版本 ID 均为 null。
> 3. 启用或暂停版本控制时，会改变对象存储在处理这些对象的请求方式，不会改变对象本身。
> 4. 只有主账号和授权子账号可以暂停存储桶的版本控制。

管理版本控制状态下的对象

存储桶处于不同的版本控制状态下，您均可对不同状态的存储桶中的对象进行上传、查询和删除操作。除了未启用版本控制状态，启用版本控制状态和暂停版本控制状态下，查询存储桶中的对象和删除对象的操作还包括不指定版本 ID 和指定版本 ID。

- 未启用版本控制状态下：上传、查询和删除对象等操作方式不变。
- 启用和暂停版本控制状态下：上传、查询和删除对象等操作方式，与以往方式的差别在于引入了版本 ID。其中执行删除对象操作还有“删除标记”的概念。

管理启用版本控制状态下的对象

启用存储桶版本控制前，已存储在存储桶中的对象，其版本 ID 为 null。启用版本控制后，不会改变存储桶中已有的对象，只会改变 COS 处理对已有对象的方式（如请求方式）。此时，新增加的同名对象将以不同的版本存在于同一个存储桶中。以下将介绍在已启用版本控制的存储桶中如何管理对象：

> 用户在未启用版本控制和启用版本控制的存储桶中上传对象的方式都是相同的，但其版本 ID 不同。通过第二种方式，对象存储会为对象分配特定版本 ID，而前一种方式上传的对象，其版本 ID 始终为 null。

上传对象

对存储桶启用版本控制后，当用户执行 PUT、POST 或 COPY 操作时，COS 会为存放到该存储桶中的对象自动添加唯一的版本 ID。 如下图所示，在启用了版本控制的存储桶中上传对象时，对象存储为该对象添加唯一的版本 ID。

列出版本控制对象

对象存储在与存储桶关联的 versions 参数中存储对象版本信息。COS 按照存储时间的先后顺序返回对象版本，最先返回最近存储的版本。

查询特定对象的所有版本

您可以通过以下过程，使用 versions 参数和 prefix 请求参数查询某对象的所有版本。有关 prefix 的更多信息，请参见 GET Bucket Object Versions 文档。

查询某一对象的所有版本，请求示例如下：

```
GET /?versions&prefix=OjectKey HTTP/1.1
```

查询数据元版本

用户使用 GET 请求时无指定版本 ID，将查询对象的当前版本。如下图所示，GET 请求将返回 123.txt 对象的当前版本（最近版本）。

如用户使用 GET 请求时指定其版本 ID，将查询指定版本 ID 的对象。如下图所示，GET versionId 请求查询指定版本（可以是当前版本）的对象。

查询对象版本的元数据

如果您只需查询对象的元数据（而不是其内容），您可以使用 HEAD 操作。默认情况下，您将获得最新版本的元数据。如要查询指定对象版本的元数据，则发送请求时需要指定其版本 ID。 查询指定版本的对象的元数据步骤如下：

- 将 versionId 配置为被查询对象元数据的版本 ID。
- 发送指定 versionId 的 HEAD 操作请求。

删除对象

您可以根据需要，随时删除不必要的对象版本。用户在已启用版本控制状态下，使用 DELETE 请求有以下两个场景：

1. 用户未指定版本 ID，执行一般 DELETE 操作。 此操作场景类似于将被删除对象放到了“回收站”，但没有完全移除对象，后续用户如有需要仍然可以恢复数据。 如下图所示，用户在 DELETE 操作时不指定版本 ID，实际上不会删除 Key=123.txt 的对象，而是插入一个新的删除标记，并添加新的版本 ID。

> 注意：
>
> COS 将在存储桶中为被删除对象插入一个拥有新版本 ID 的删除标记，该删除标记将成为被删除对象的当前版本。当您尝试对该删除标记的对象执行 GET 操作时，对象存储会认为该对象不存在，并返回404错误。

2. 用户指定版本 ID，执行操作删除对象版本，此场景可以永久删除版本控制的对象。

删除标记

删除标记用于版本控制的对象，删除标记在 COS 中可认为是“对象已被删除”的标记。删除标记与对象同样拥有对象键（Key）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。
- 删除标记执行 GET 请求会返回404错误，响应标头 `x-cos-delete-marker: true` 。
- 删除标记只支持 DELETE 操作（需要主账号下发出请求）。

删除“删除标记”

用户如需删除“删除标记”，则可以在 DELETE Object versionId 请求中指定它的版本 ID，实现永久删除“删除标记”。如果您未指定删除标记的版本 ID，对删除标记发出 DELETE 请求，COS 将不会删除该删除标记，而是再插入一个新的删除标记。 如下图所示，对删除标记执行一般 DELETE 请求，不会删除任何内容，而在存储桶里新增了一个新的删除标记。

在已启用版本控制的存储桶中，新增的删除标记将具有唯一的版本 ID。因此，在一个存储桶中，同一个对象可能有多个删除标记。要永久删除“删除标记”，必须在 DELETE Object versionId 请求中包含其版本 ID。 如下图所示，执行 DELETE Object versionId 请求永久删除“删除标记”。

> 注意：
>
> 只有主账号可以永久删除“删除标记”。

永久删除“删除标记”的步骤：

1. 设置 versionId 为删除标记的版本 ID。
2. 发送 DELETE Object versionId 请求。

还原早期版本

版本控制能够用来还原对象的早期版本，有两种方法可执行该操作：

1. 将对象的早期版本复制到同一存储桶中

   复制的对象将成为该对象的当前版本，且所有对象版本都保留。

2. 永久删除对象的当前版本

   当您删除当前对象版本时，实际上会将前一个版本转换为该对象的当前版本。

管理暂停版本控制状态下的对象

暂停版本控制时，存储桶中的现有对象不会更改。更改的是对象存储在以后的请求中处理对象的方式。以下将介绍在已暂停版本控制的存储桶中如何管理对象。

上传对象

在存储桶上暂停版本控制后，当用户执行 PUT、POST 或 COPY 操作时，COS 自动将版本 ID 为 null 添加到存放到该存储桶中的对象。如下图所示：

如果存储桶中存在版本控制的对象，则上传到存储桶的对象将成为当前版本，并且版本 ID 为 null。如下图所示：

如果存储桶中已存在空版本，则该空版本将被覆盖，原有的对象内容也会相应被替换，如下图所示：

查询数据元版本

在已暂停版本控制的存储桶上，用户发出 GET Object 请求将返回对象的当前版本。

删除对象

如果暂停了版本控制，执行 DELETE 请求有以下情况：

- 存储桶中存在空版本的对象，将删除其版本 ID 为 null 的对象。

  如下图所示，用户执行一般 DELETE 操作时，COS 会为空版本的对象插入删除标记。

> 注意：
>
> 删除标记不存在内容，在删除标记替换空版本时，空版本原先的内容会丢失。

- 存储桶中没有空版本的对象，存储桶中会新添加一个删除标记。

  如下图所示，在存储桶不存在空版本的情况下，用户执行 DELETE 操作不会删除任何内容，对象存储仅插入删除标记。

- 即使是在已暂停版本控制的存储桶中，主账号也可以永久删除指定版本。

  如下图所示，删除指定的对象版本将永久删除该对象。

> **注意：**
>
> 只有主账号可以删除指定的对象版本。

删除标记

最近更新时间: 2025-01-15 17:01:00

简介

删除标记用于版本控制的对象，删除标记在对象存储（Cloud Object Storage，COS）中可认为是“对象已被删除”的标记。删除标记与对象同样拥有对象键（Key）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。
- 删除标记执行 GET 请求会返回404错误。
- 删除标记只支持 DELETE 操作（需要主账号下发出请求）。

删除“删除标记”

用户如需删除“删除标记”，则可以在 DELETE Object versionId 请求中指定它的版本 ID，实现永久删除“删除标记”。如果您未指定删除标记的版本 ID，对删除标记发出 DELETE 请求，COS 将不会删除该删除标记，而是再插入一个新的删除标记。

如下图所示，对删除标记执行一般 DELETE 请求，不会删除任何内容，而在存储桶里新增了一个新的删除标记。

在已启用版本控制的存储桶中，新增的删除标记将具有唯一的版本 ID。因此，在一个存储桶中，同一个对象可能有多个删除标记。要永久删除“删除标记”，必须在 DELETE Object versionId 请求中包含其版本 ID。

如下图所示，执行 DELETE Object versionId 请求永久删除“删除标记”。

> **注意：**
>
> 只有经过主账号授权 `DeleteObject` 操作后才可删除“删除标记”。

永久删除“删除标记”的步骤：

1. 设置 versionId 为删除标记的版本 ID。
2. 发送 DELETE Object versionId 请求。

版本控制配置

最近更新时间: 2025-01-15 17:01:00

使用场景

利用版本控制功能，您可以在存储桶中存放对象的多个版本，并且实现检索、删除和还原指定版本对象。 了解版本控制详情信息，请参见 版本控制概述 文档。

> **注意：**
>
> 只有主账号和被授权的子账号可以配置存储桶的版本控制状态。

使用方法

使用 REST API

您可以直接使用 REST API 配置存储桶的版本控制和管理版本控制状态下存储桶中的对象，请参见 API 文档的下述部分：

- PUT Bucket versioning
- GET Buket versioning
- GET Bucket Object versions
- PUT Object
- GET Object
- DELETE Object
- DELETE Multiple Objects

使用 SDK

参考对应语言的 SDK 文档中，存储桶管理 > 版本控制 章节

跨地域复制
跨地域复制概述

最近更新时间: 2025-01-15 17:01:00

简介

跨地域复制是针对存储桶的一项配置，通过配置跨地域复制规则，可以在不同存储区域的存储桶中自动、异步地复制**增量对象**。启用跨地域复制后，COS 将精确复制源存储桶中的对象内容（如对象元数据和版本 ID 等）到目标存储桶中，复制的对象副本拥有完全一致的属性信息。此外，源存储桶中对于对象的操作，如添加对象、删除对象等操作，也将被复制到目标存储桶中。

> 注意：
>
> - 开启跨地域复制功能要求源存储桶和目标存储桶均处于不同的存储地域中，且源存储桶和目标存储桶均启用了版本控制功能。
> - 启用跨地域复制时，除非您在复制数据时明确指定了配置对象副本的存储类型，否则对象副本将保持与源对象相同的存储类型。
> - COS 复制时将复制源存储桶的访问控制列表（ACL），目前 COS 不支持两个不同账号的存储桶互为源存储桶和目标存储桶。

适用场景

- **异地容灾**：COS 为对象数据提供了11个9的可用性，但仍然存在各种不可抗因素如战争、自然灾害等因素导致数据丢失。如果您无法忍受因数据丢失带来的损失，希望显式地在不同地域维护一份数据副本，那么您可以通过跨地域复制实现数据的异地容灾，当某个数据中心因为不可抗因素损毁时，另一个地域的数据中心仍然可以提供副本数据以供您使用。
- **合规性要求**：COS 默认在物理盘中为数据提供多副本和纠删码等方式保障数据的可用性，但某些行业中可能存在合规性要求，规定您需要在不同的存储地域间保存数据副本。因此启用跨地域复制，可以实现在不同存储地域间复制数据以满足这些合规性要求。
- **减少访问延迟**：当您的客户在不同地理位置访问对象时，您可以通过跨地域复制，在与客户地理位置最近的可用存储地域中维护对象副本，最大限度上缩短客户的访问延迟，有利于提高您的产品体验。
- **操作原因**：如果您在两个不同地域中均具有计算集群，且这些计算集群需要处理同一套数据，则您可以通过跨地域复制在这两个不同的地域中维护对象副本。
- **数据迁移与备份**：您可以根据业务发展需要，将业务数据从一个可用地域复制到另一个可用地域，实现数据迁移和数据备份。

注意事项

复制时间限制

COS 复制对象所需的时间取决于对象大小、存储地域间的距离，以及对象的上传方式等因素。同步时间根据上述因素差异，在几分钟到几小时内不等。

- 对象大小。大型对象的复制需要消耗更多的时间，对于大型对象而言，建议使用分块上传的方式以减少对象的上传和同步时间。
- 存储地域间的距离。地域间距离更远的同步需要消耗更多的数据传输时间。
- 对象上传方式。简单上传方式不能做并发，只能在一条连接上串行地上传或者下载数据，分块上传方式能够做并发，因此大文件上传时通过分块上传能够加速上传及跨地域复制。

版本控制相关

跨地域复制配置需要用户在源存储桶和目标存储桶中均配置版本控制功能，版本控制功能的详细内容请详见 版本控制概述。开启版本控制后，需要注意关闭版本控制对跨地域复制功能的影响：

- 如果您尝试在已开启跨地域复制功能的存储桶中禁用版本控制，则 COS 会返回错误并提示您需要先删除跨地域复制规则后再禁用版本控制。
- 如果您尝试在一个目标存储桶中禁用版本控制，则 COS 会提示您关闭版本控制后跨地域复制功能将受影响，如果您仍确认关闭版本控制，则 COS 以该存储桶作为目标存储桶的跨地域复制规则将失效。

复制行为说明

最近更新时间: 2025-01-15 17:01:00

本文档主要介绍用户在对存储桶启用了跨地域复制功能后，对象存储 COS 会复制的内容和不复制的内容。

复制的内容

在启用了跨地域复制功能的源存储桶中，对象存储将会复制以下内容：

- 添加跨地域复制规则后，用户往源存储桶中新上传的任何对象。
- 对象的元数据和版本 ID 等对象属性信息。
- 有关对象的操作信息，如新增同名对象（等同于新增对象），删除对象等。
  - 如果您在源存储桶中指定删除某个对象版本，即指定了版本 ID，则该操作不会被复制。
  - 如果您在源存储桶中添加了存储桶级别配置，如生命周期规则，则因这些配置引起的对象操作也不会复制到目标存储桶中。

跨地域复制下的删除操作

如果从源存储桶中删除对象，则跨地域复制行为如下所示：

- 不指定对象版本 ID 执行 DELETE 请求时，COS 将源存储桶中添加删除标记，同时跨地域复制会将该标记复制到目标存储桶。关于版本控制和删除标记的详细信息可参阅版本控制配置 文档。
- 指定对象版本 ID 执行 DELETE 请求时，COS 将删除源存储桶中指定的对象版本，但不会在目标存储桶中复制这一删除操作，即 COS 不会在目标存储桶中删除指定的对象版本。此行为可防止恶意删除数据。

不会复制的内容

当源存储桶启用了跨地域复制功能，对象存储不会复制以下内容：

- 启用跨地域复制功能之前已存在的对象内容，即存量数据。
- 已加密的对象的加密信息，即加密对象被复制后将失去加密信息。
- 源存储桶中新增的数据是来自其它存储桶复制的对象数据。
- 存储桶级别的配置更新行为。
- 生命周期配置执行后的结果。
  - 对象数据在存储桶间的跨地域复制不具备传递性，如果您同时设置了 A 存储桶为源存储桶，B 存储桶为目标存储桶和 B 存储桶为源存储桶，C 存储桶为目标存储桶的两条跨地域复制规则，那么 A 存储桶中的新增对象数据仅会复制到 B 存储桶中，而不会进一步复制到 C 存储桶中。

    例如生命周期的配置，当您更新了源存储桶的生命周期配置，COS 不会将这一生命周期配置同步应用到目标存储桶。
  - 如果您只对源存储桶配置了生命周期规则，对象存储会为过期对象添加删除标记，而目标存储桶不会复制这些标记。如果您希望目标存储桶能够删除过期对象，则需要您单独对目标存储桶配置与源存储桶相同的生命周期规则。

跨地域复制配置

最近更新时间: 2025-01-15 17:01:00

适用场景

通过配置跨地域复制规则，用户可以将对象数据从源存储桶复制到另一地域的指定目标存储桶中。跨地域复制功能适用于异地容灾、满足行业合规性要求、数据迁移与备份、降低客户访问延迟、方便不同地域的集群访问数据等场景。

> 注意：
>
> 开启了版本控制后，新上传的对象将会产生多个版本并占用存储空间，因此这些版本的对象同样收取存储费用。

使用方法

使用 REST API

您可以直接使用 REST API 配置和管理存储桶的跨地域复制规则，具体可参考 API 文档中的下述部分：

- PUT Bucket replication
- GET Bucket replication
- DELETE Bucket replication

使用 SDK

参考对应语言的 SDK 文档中，存储桶管理 > 跨地域复制 章节

访问管理
访问策略语言概述

最近更新时间: 2025-01-15 17:01:00

概述

访问策略可用于授予访问 COS 资源的权限。访问策略使用基于 JSON 的访问策略语言。您可以通过访问策略语言授权指定委托人（principal）对指定的 COS 资源执行指定的操作。

访问策略中的元素

访问策略语言包含以下基本意义的元素：

- 委托人（principal）：描述策略授权的实体。例如用户（开发商、子账号、匿名用户）、用户组等。该元素对于存储桶访问策略有效，对用户访问策略则不应添加。
- 语句（statement）：描述一条或多条权限的详细信息。该元素包括效力、操作、资源等多个其他元素的权限或权限集合。一条策略有且仅有一个语句元素。
  - 效力（effect）：描述声明产生的结果是"允许"还是"显式拒绝"，包括 allow 和 deny 两种情况。该元素是必填项。
  - 操作（action）：描述允许或拒绝的操作。操作可以是 API（以 name 前缀描述）或者功能集（一组特定的 API，以 permid 前缀描述）。该元素是必填项。
  - 资源（resource）：描述授权的具体数据。资源是用六段式描述。每款产品的资源定义详情会有所区别。有关如何指定资源的信息，请参阅您编写的资源声明所对应的产品文档。该元素是必填项。

元素用法

元素用法

委托人 principal 元素用于指定被允许或拒绝访问资源的用户、账户、服务或其他实体。元素 principal 仅在存储桶中起作用；用户策略中不必指定，因为用户策略直接附加到特定用户。下面是指定 principal 的示例。

```
"principal": {
"qcs": [
"qcs::cam::uin/1:uin/1"
]
}
```

授予匿名用户权限：

```
"principal": {
"qcs": [
"qcs::cam::anonymous:anonymous"
]
}
```

授权主账户 UIN 1200000313 权限：

```
"principal": {
"qcs": [
"qcs::cam::uin/1200000313:uin/1200000313"
]
}
```

授权子账户 UIN 3030313（主账户 UIN 为 1200000313）权限：

> **注意：**
>
> 操作前需确保子账号已被添加到主账号的子账号列表中。

```
"principal": {
"qcs": [
"qcs::cam::uin/1200000313:uin/3030313"
]
}
```

指定效力

如果没有显式授予（允许）对资源的访问权限，则隐式拒绝访问。您也可显式拒绝（deny）对资源的访问，这样可确保用户无法访问该资源，即使有其他策略授予了访问权限的情况下也是如此。下面是指定允许效力的示例。

```
"effect" : "allow"
```

指定操作

COS 定义了可在策略中指定的某一个特定的 COS 操作，指定的操作与发起的 API 请求操作完全一致。

存储桶操作

描述	对应的 API 接口
name/cos:GetService	GET Service
name/cos:GetBucket	GET Bucket (List Object)
name/cos:PutBucket	PUT Bucket
name/cos:DeleteBucket	DELETE Bucket
name/cos:HeadBucket	HEAD Bucket
name/cos:GetBucketPolicy	GET Bucket Policy
name/cos:PutBucketPolicy	PUT Bucket Policy
name/cos:DeleteBucketPolicy	DELETE Bucket Policy
name/cos:GetBucketACL	GET Bucket ACL
name/cos:PutBucketACL	PUT Bucket ACL
name/cos:ListMultipartUploads	LIST in-progress multipart uploads

对象操作

描述	对应的 API 接口
name/cos:GetObject	GET Object
name/cos:PutObject	PUT Object
name/cos:HeadObject	HEAD Object
name/cos:DeleteObject	DELETE Object
name/cos:PutObjectCopy	COPY Object
name/cos:PostObject	POST Object
name/cos:GetObjectACL	GET Object ACL
name/cos:PutObjectACL	PUT Object ACL
name/cos:InitiateMultipartUpload	Initiate a multipart upload
name/cos:UploadPart	Upload a part in a multipart upload
name/cos:CompleteMultipartUpload	Complete a multipart upload
name/cos:AbortMultipartUpload	Abort a multipart upload

指定允许操作的示例如下：

```
"action": [
"name/cos:GetObject",
```

```
"name/cos:HeadObject"
]
```

指定资源

资源（resource）元素描述一个或多个操作对象，如 COS 存储桶或对象等。所有资源均可采用下述的六段式描述方式。

```
qcs:project_id:service_type:region:account:resource
```

其中：

- qcs 是 qcloud service 的简称。
- project_id 描述项目信息，仅为了兼容 CAM 早期逻辑。这里请不填。
- service_type 描述产品简称，如 COS。
- region 描述地域信息。
- account 描述资源拥有者的根账号信息。目前支持两种方式描述的资源拥有者。一种方式是 uin 方式，即根账号的 qq 号，表示为 uin/${uin}，如 uin/164256472。另外一种方式是 uid 方式，即根账号的 appid，表示为 uid/${appid}，如 uid/1000382392。目前 COS 的资源拥有者统一使用 uid 的方式表述，即根账号的开发商 appid。
- resource 描述具体资源详情，在 COS 服务中使用存储桶 XML API 访问域名来描述。

下面是指定存储桶 burningtest-1251500699 的示例。

```
"resource": ["qcs::cos:ap-guangzhou:uid/1251500699:burningtest-1251500699/*"]
```

下面是指定存储桶 burningtest-1251500699 中的 /test/ 文件夹下所有对象的示例。

```
"resource": ["qcs::cos:ap-guangzhou:uid/1251500699:burningtest-1251500699/test/*"]
```

下面是指定存储桶 burningtest-1251500699 中的 /test/1.txt 对象的示例。

```
"resource": ["qcs::cos:ap-guangzhou:uid/1251500699:burningtest-1251500699/test/1.txt"]
```

实际案例

当根账号允许匿名用户，在访问来源 IP 为 101.226.185/101.226.186 时，对华南地区存储桶 burningtest-1251500699 中的对象，执行 GET（下载）和 HEAD 操作，而无需鉴权。

```
{
"version": "2.0",
"principal": {
"qcs": [
"qcs::cam::anonymous:anonymous"
]
},
"statement": [
{
"action": [
"name/cos:GetObject",
"name/cos:HeadObject"
],
"effect": "allow",
"resource": [
"qcs::cos:cn-south:uid/1251500699:burningtest-1251500699/*"
]
}
]
}
```

最佳实践
访问控制与权限管理
ACL 访问控制实践

最近更新时间: 2025-01-15 17:01:00

ACL 概述

访问控制列表（ACL）是基于资源的访问策略选项之一 ，可用来管理对存储桶和对象的访问。使用 ACL 可向其他根账户和用户组授予基本的读、写权限。

与访问策略有所不同的是，ACL 的管理权限有一些限制：

- 仅支持对云平台的账户赋予权限。
- 仅支持读对象、写对象、读 ACL、写 ACL 和全部权限等五个操作组。
- 不支持赋予生效条件。
- 不支持显示拒绝效力。

ACL 支持的控制粒度：

- 存储桶（Bucket）
- 对象键前缀（Prefix）
- 对象（Object）

ACL 的控制元素

当创建存储桶或对象时，其资源所属的主账户将具备对资源的全部权限，且不可修改或删除。您可以使用 ACL 赋予其他云平台账户的访问权限。如下提供了一个存储桶的 ACL 示例。

```
<AccessControlPolicy>
<Owner>
<ID>qcs::cam::uin/12345:uin/12345</ID>
<DisplayName>qcs::cam::uin/12345:uin/12345</DisplayName>
</Owner>
<AccessControlList>
<Grant>
<Grantee xmlns:xsi="http://imgcache.finance.cloud.tencent.com:80www.w3.org/2001/XMLSchema-instance" xsi:type="RootAccount">
<ID>qcs::cam::uin/12345:uin/12345</ID>
<DisplayName>qcs::cam::uin/12345:uin/12345</DisplayName>
</Grantee>
<Permission>FULL_CONTROL</Permission>
</Grant>
<Grant>
<Grantee xmlns:xsi="http://imgcache.finance.cloud.tencent.com:80www.w3.org/2001/XMLSchema-instance" xsi:type="RootAccount">
<ID>qcs::cam::uin/54321:uin/54321</ID>
<DisplayName>qcs::cam::anyone:anyone</DisplayName>
</Grantee>
<Permission>READ</Permission>
</Grant>
</AccessControlList>
</AccessControlPolicy>
```

如上 ACL 包含了识别该存储桶所有者的 Owner 元素，该存储桶所有者具备该存储桶的全部权限。同时 Grant 元素授予了匿名的读取权限，其表述形式为 `qcs::cam::anyone:anyone` 的 READ 权限。

权限被授予者

根账户

您可以对其他根账户授予用户访问权限，使用 CAM 中对委托人（principal）的定义进行授权。描述为：

```
qcs::cam::uin/1238423:uin/1238423
```

匿名用户

您可以对匿名用户授予访问权限，使用 CAM 中对委托人（principal）的定义进行授权。描述为：

```
qcs::cam::anyone:anyone
```

权限操作组

以下表格提供了 ACL 支持的权限操作组。

操作组	授予存储桶	授予前缀	授予对象
READ	列出和读取存储桶中的对象	列出和读取目录下的对象	读取对象
WRITE	创建、覆盖和删除存储桶中的任意对象	创建、覆盖和删除目录下的任意对象	覆盖和删除对象
READ_ACP	读取存储桶的 ACL	读取目录下的 ACL	读取对象的 ACL
WRITE_ACP	修改存储桶的 ACL	修改目录下的 ACL	修改对象的 ACL
FULL_CONTROL	对存储桶和对象的任何操作	对目录下的对象做任何操作	对对象执行任何操作

标准 ACL 描述

COS 支持一系列的预定义授权，称之为标准 ACL，下表列出了标准 ACL 的授权含义。

> 注意：
>
> 由于主账户始终拥有 FULL_CONTROL 权限，以下表格中不再对此特别说明。

标准 ACL	含义
(空)	此为默认策略，其他人无权限，资源继承上级权限。
private	其他人没有权限。
public-read	匿名用户组具备 READ 权限。
public-read-write	匿名用户组具备 READ 和 WRITE 权限，通常不建议在存储桶赋予此权限。

ACL 示例

为存储桶设置 ACL

以下示例对存储桶授予了另一个根账户的读取权限：

为对象设置 ACL

以下示例对某个对象授予了另一个根账户的读取权限：

访问管理实践

最近更新时间: 2025-01-15 17:01:00

访问管理概述

访问管理（Cloud Access Management，CAM）是云平台提供的一种身份验证和授权服务，主要用于帮助客户安全管理云平台账户下的资源的访问权限。在授予权限时，可以对授权的对象、资源、操作进行管理，并支持设置一些策略限制。

访问管理功能

根账号资源的授权

可以将根账号的资源授权给其他人员，包括子账号或者是其他根账号，而不需要分享根账号相关的身份凭证。

精细化的权限管理

可以针对不同的资源为不同的人员授予不同的访问权限。例如可以允许某些子账号拥有某个 COS 存储桶的读权限，而另外一些子账号或者根账号可以拥有某个 COS 存储对象的写权限等。上述资源、访问权限、用户都可以批量打包。

访问管理应用场景

企业子账号权限管理

企业内不同岗位的员工需要拥有该企业云资源的最小化访问权限。

场景：某个企业拥有很多云资源，包括 CVM 、 VPC 实例、 CDN 实例、 COS 存储桶和对象等。该企业拥有很多员工，包括开发人员、测试人员、运维人员等。 部分开发人员需要拥有其所在项目相关的开发机云资源的读写权限，测试人员需要拥有其所在项目的测试机云资源的读写权限，运维人员负责机器的购买和日常运营。当企业员工职责或参与项目发生变更，将终止对应的权限。

不同企业之间的权限管理

不同企业间需要进行云资源的共享。

场景：某企业拥有很多云资源，该企业希望能专注产品研发，而将云资源运维工作授权给其他运营企业来实施。当运营企业的合同终止时，将收回对应的管理权限。

访问管理策略语法

策略（policy）由若干元素构成，用来描述授权的具体信息。核心元素包括委托人（principal）、操作（action）、资源（resource）、生效条件（condition）以及效力（effect）。

策略语法说明

- 元素仅支持小写。它们在描述上没有顺序要求。
- 对于策略没有特定条件约束的情况， condition 元素是可选项。
- 在控制台中不允许写入 principal 元素，仅支持在策略管理 API 中和策略语法相关的参数中使用 principal 。

版本 version

描述策略语法版本。目前仅允许值为 2.0。该元素是必填项。

委托人 principal

用于描述策略授权的实体。包括用户（开发商、子账号、匿名用户）、用户组，仅支持在策略管理 API 中策略语法相关的参数中使用该元素。

语句 statement

用于描述一条或多条权限的详细信息。该元素包括 action 、 resource 、 condition 、 effect 等多个其他元素的权限或权限集合。一条策略有且仅有一个 statement 元素。该元素是必填项。

操作 action

用于描述允许或拒绝的操作。操作可以是 API （以 name 前缀描述）或者功能集（一组特定的 API ，以 permid 前缀描述）。该元素是必填项。

资源 resource

用于描述授权的具体数据。资源用六段式描述。每款产品的资源定义详情会有所区别。有关如何指定资源的信息，请参阅您编写的资源声明所对应的产品文档。该元素是必填项。

生效条件 condition

用于描述策略生效的约束条件。条件包括操作符、操作键和操作值组成。条件值可包括时间、 IP 地址等信息。有些服务允许您在条件中指定其他值。该元素是非必填项。

效力 effect

用于描述声明产生的结果，包括 allow（允许）和 deny（显式拒绝）两种情况。该元素是必填项。

策略限制

限制项	限制值
一个根账号中的用户组数	20 个
一个根账号中的子用户数	1000 个
一个子用户可加入的用户组的数量	10 个
一个用户组中的子用户数	100 个
一个根账号的策略数	1000 个
关联到一个 CAM 用户、用户组的策略数	20 个
自定义策略字符数	4096 字符

授权子账号访问COS

最近更新时间: 2025-01-15 17:01:00

概述

对于对象存储资源，不同企业之间或同企业多团队之间，需要对不同的团队或人员配置不同的访问权限。您可通过 CAM（访问管理，以下简称 CAM）对存储桶或对象设置不同的操作权限，使得不同团队或人员能够相互协作。

首先，我们需要先了解几个关键概念：主账号、子账号（用户）和用户组。CAM 的相关术语、配置详细描述请参见访问管理。

主账号

主账号又被称为开发商。用户申请腾讯云金融专区账号时，系统会创建一个用于登录腾讯云金融专区服务的主账号身份。主账号是腾讯云金融专区资源使用计量计费的基本主体。 主账号默认拥有其名下所拥有的资源的完全访问权限，包括访问账单信息，修改用户密码，创建用户和用户组以及访问其他云服务资源等。默认情况下，资源只能被主账号所访问，任何其他用户访问都需要获得主账号的授权。

子账号（用户）和用户组

- 子账号是由主账号创建的实体，有确定的身份 ID 和身份凭证，拥有登录腾讯云金融专区控制台的权限。
- 子账号默认不拥有资源，必须由所属主账号进行授权。
  - 一个主账号可以创建多个子账号（用户）。
  - 一个子账号可以归属于多个主账号，分别协助多个主账号管理各自的云资源，但同一时刻，一个子账号只能登录到一个主账号下，管理该主账号的云资源。
- 子账号可以通过控制台切换开发商（主账号），从一个主账号切换到另外一个主账号。
  - 子账号登录控制台时，会自动切换到默认主账号上，并拥有默认主账号所授予的访问权限。
  - 切换开发商之后，子账号会拥有切换到的主账号授权的访问权限，而切换前的主账号授予的访问权限会立即失效。
- 用户组是多个相同职能的用户（子账号）的集合。您可以根据业务需求创建不同的用户组，为用户组关联适当的策略，以分配不同权限。

操作步骤

授权子账号访问 COS 分为三个步骤：创建子账号、对子账号授予权限、子账号访问 COS 资源。

步骤1：创建子账号

在 CAM 控制台可创建子账号，并配置授予子账号的访问权限。具体操作如下所示：

1. 登录 CAM 控制台。
2. 在左侧导航树中选择【用户管理】，进入用户管理页面。
3. 单击【新建用户】，选择**子用户**类型，进入【新建子用户】页面。
4. 按照要求填写用户相关信息。
   - **用户名**：输入子用户名称，例如 Sub_user。
   - **邮箱**：您需要为子用户添加邮箱来获取由腾讯云金融专区发出的绑定微信的邮件。
   - **访问类型**：选择编程访问或云平台管理控制台访问。
5. 填写用户信息完毕后，单击【下一步】，进行身份验证。
6. 身份验证完毕，设置子用户权限。根据系统提供的策略选择，可配置简单的策略，例如 COS 的存储桶列表的访问权限，只读权限等。如需配置更复杂的策略，可进行 步骤2：对子账号授予权限 。
7. 确认输入的用户信息无误后，单击【完成】即可创建子账号。

步骤2：对子账号授予权限

对子账号授予权限可通过 CAM，对子账号（用户）或用户组进行策略配置。

1. 登录 CAM 控制台。
2. 选择【策略管理】>【新建自定义策略】>【按策略语法创建】，进入策略创建页面。
3. 可供选择的模版有**空白模板**和与 COS 相关联的**预设策略模板**，选择您需要授予子账号的策略模板，单击【下一步】。

4. 输入便于您记忆的策略名称，若您选择**空白模板**，则需要输入您的策略语法，详情请参见 策略示例。您可将策略内容复制粘贴到【编辑策略内容】输入框内，单击【创建策略】。

5. 创建完成后，将刚才已创建的策略关联到子账号。

6. 勾选子账号确定授权后，子账号即可根据权限范围访问 COS 资源。

步骤3：子账号访问 COS 资源

COS 访问（API 或 SDK）需要如下资源：APPID、SecretId、SecretKey。 当使用子账号访问 COS 资源时，需要使用主账号的 APPID，子账号的 SecretId 和 SecretKey，您可以在访问管理控制台生成子账号的 SecretId 和 SecretKey。

1. 主账号登录 CAM 控制台 。
2. 选择【用户列表】，进入用户列表页面。
3. 单击子账号用户名称，进入子账号信息详情页。
4. 单击【API 密钥】页签，并单击【新建密钥】为该子账号创建 SecretId 和 SecretKey。

至此您就可以通过子账号的 SecretId 和 SecretKey、主账号的 APPID，访问 COS 资源。

> 注意：
>
> - 子账号需通过 XML API 或基于 XML API 的 SDK 访问 COS 资源。
> - 子账号访问 COS 资源时，需要使用主账号的 APPID，子账号的 SecretId 和 SecretKey。

基于 XML 的 Java SDK 访问示例

以基于 XML 的 Java SDK 命令行为例，需填入参数如下：

```
// 1 初始化身份信息
COSCredentials cred = new BasicCOSCredentials("<主账号APPID>", "<子账号SecretId>", "<子账号SecretKey>");
```

实例如下：

```
// 1 初始化身份信息
COSCredentials cred = new BasicCOSCredentials("1250000000", "AKIDasdfmRxHPa9oLhJp", "e8Sdeasdfas2238Vi");
```

COSCMD 命令行工具访问示例

以 COSCMD `config` 命令行为例，需填入参数如下：

```
coscmd config -u <主账号 APPID> -a <子账号 SecretId> -s <子账号SecretKey> -b <主账号 bucketname> -r <主账号 bucket 所属地域>
```

实例如下：

```
coscmd config -u 1250000000 -a AKIDasdfmRxHPa9oLhJp -s e8Sdeasdfas2238Vi -b examplebucket -r ap-beijing
```

策略示例

以下提供几种典型场景的策略示例，在配置自定义策略时，您可将以下参考策略复制粘贴至输入框【编辑策略内容】，根据实际配置修改即可。更多 COS 常见场景的策略语法请参见 CAM 产品文档 的**商用案例**部分。

为子账户配置读写权限

为子账户仅配置读写权限，具体策略如下所示：

```
{
"version": "2.0",
"statement": [
{
"action": [
"name/cos:*"
],
"resource": "*",
"effect": "allow"
},
```

```
{
"effect": "allow",
"action": "monitor:*",
"resource": "*"
}
]
}
```

为子帐户配置只读权限

为子账户仅配置只读权限，具体策略如下所示：

```
{
"version": "2.0",
"statement": [
{
"action": [
"name/cos:List*",
"name/cos:Get*",
"name/cos:Head*",
"name/cos:OptionsObject"
],
"resource": "*",
"effect": "allow"
},
{
"effect": "allow",
"action": "monitor:*",
"resource": "*"
}
]
}
```

为子账户配置某 IP 段的读写权限

本示例中限制仅 IP 网段为 192.168.1.0/24 和 192.168.2.0/24 的地址具有读写权限，如下所示。 更丰富的生效条件填写，请参见 生效条件。

```
{
"version": "2.0",
"statement": [
{
"action": [
"cos:*"
],
"resource": "*",
"effect": "allow",
"condition": {
"ip_equal": {
"qcs:ip": ["192.168.1.0/24", "192.168.2.0/24"]
}
}
}
]
}
```

权限设置相关案例

最近更新时间: 2025-01-15 17:01:00

通过存储桶策略（Policy）授权案例

准备工作

1. 创建存储桶

通过存储桶策略（Policy）授权仅针对特定的存储桶，因此您需要先 创建存储桶。如需针对账号维度进行授权，请参见本文的 通过访问管理（CAM）授权案例。

2. 准备被授权账号的 UIN

本文假设拥有目标存储桶的主账号 UIN 为100000000001，其下的子账号为100000000011，子账号需要被授权才能访问目标存储桶。

> 说明：
> - 如需查询主账号下所创建的子账号，可登录访问管理控制台，在 用户列表 中查看。
> - 如需创建新的子账号，请参见 新建子用户 文档。

3. 打开【添加策略】对话框

进入目标存储桶的【权限管理】，选择【Policy 权限设置】>【图形设置】，并单击打开【添加策略】对话框，随后请参见本文的授权案例进行配置。添加策略的详细操作指引，可参见 添加存储桶策略 文档。

以下列举了几种不同的授权案例，您可按照实际情况进行配置。

授权案例

案例一：授予子账号拥有特定目录的所有权限

配置信息如下：

配置项	配置值
效力	允许
用户类型	子账号
账号 ID	子账号的 UIN，该子账号必须为当前主账号下的子账号，例如100000000011
资源	指定资源
资源路径	特定目录前缀，例如`folder/sub-folder/`，需以`/`结尾
操作名称	所有操作

案例二：授予子账号拥有特定目录内文件的读权限

配置信息如下：

配置项	配置值
效力	允许
用户类型	子账号
账号 ID	子账号的 UIN，该子账号必须为当前主账号下的子账号，例如100000000011
资源	指定资源
资源路径	特定目录前缀，例如`folder/sub-folder/`，需以`/`结尾

配置项	配置值
操作名称	读操作(含列出对象列表)

案例三：授予子账号拥有特定文件的读写权限

配置信息如下：

配置项	配置值
效力	允许
用户类型	子账号
账号 ID	子账号的 UIN，该子账号必须为当前主账号下的子账号，例如100000000011
资源	指定资源
资源路径	特定对象键，例如`folder/sub-folder/exampleobject`
操作名称	所有操作

案例四：授予子账号拥有特定目录下所有文件的读写权限，并禁止拥有该目录下指定文件的读写权限

针对此案例，我们需要添加两个策略：**允许**策略和**禁止**策略。

1. 首先添加**允许**策略，配置信息如下：

配置项	配置值
效力	允许
用户类型	子账号
账号 ID	子账号的 UIN，该子账号必须为当前主账号下的子账号，例如100000000011
资源	指定资源
资源路径	特定目录前缀，例如`folder/sub-folder/`，需以`/`结尾
操作名称	所有操作

2. 随后添加**禁止**策略，配置信息如下：

配置项	配置值
效力	禁止
用户类型	子账号
账号ID	子账号的 UIN，该子账号必须为当前主账号下的子账号，例如100000000011
资源	指定资源
资源路径	需要禁止被访问的对象键，例如`folder/sub-folder/privateobject`
操作名称	所有操作

案例五：授权子账号对指定前缀的文件的读写权限

配置信息如下：

配置项	配置值
效力	允许
用户类型	子账号
账号ID	子账号的 UIN，该子账号必须为当前主账号下的子账号，例如100000000011
资源	指定资源
资源路径	特定前缀，例如`folder/sub-folder/prefix`
操作名称	所有操作

COS API授权策略使用指引

最近更新时间: 2025-01-15 17:01:00

> 注意：
>
> 在给子用户或协作者授予 API 操作权限时，请务必根据业务需要，按照最小权限原则进行授权。如果您直接授予子用户或者协作者所有资源 (resource:*) ，或所有操作 (action:*) 权限，则存在由于权限范围过大导致数据安全风险。

概述

对象存储 COS 使用临时密钥服务时，不同的 COS API 操作需要不同的操作权限，而且可以同时指定一个操作或一序列操作。

COS API 授权策略（policy）是一种 JSON 字符串。例如，授予 APPID 为1250000000，地域为 ap-beijing，存储桶为 examplebucket-1250000000 ，路径前缀为 doc 的上传操作（包括简单上传、表单上传、分块上传等操作）的权限，路径前缀为 doc2 的下载操作权限的策略内容如下所示：

```
{
"version": "2.0",
"statement": [{
"action": [
//简单上传操作
"name/cos:PutObject",
//表单上传对象
"name/cos:PostObject",
//分块上传：初始化分块操作
"name/cos:InitiateMultipartUpload",
//分块上传：List 进行中的分块上传
"name/cos:ListMultipartUploads",
//分块上传：List 已上传分块操作
"name/cos:ListParts",
//分块上传：上传分块块操作
"name/cos:UploadPart",
//分块上传：完成所有分块上传操作
"name/cos:CompleteMultipartUpload",
//取消分块上传操作
"name/cos:AbortMultipartUpload"
],
"effect": "allow",
"resource": [
"qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/doc/*"
]
},
{
"action": [
//下载操作
"name/cos:GetObject"
],
"effect": "allow",
"resource": [
"qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/doc2/*"
]
}
]
}
```

授权策略（policy）元素说明

名称	描述
version	策略语法版本，默认为2.0
effect	有 allow （允许）和 deny （显式拒绝）两种情况
resource	授权操作的具体数据，可以是任意资源、指定路径前缀的资源、指定绝对路径的资源或它们的组合
action	此处是指 COS API，根据需求指定一个或者一序列操作的组合或所有操作(*)
condition	约束条件，可以不填，具体说明请参见 condition 说明

下面列出了各 COS API 设置授权策略的示例。

Service API

查询存储桶列表

API 接口为 GET Service，若授予其操作权限，则策略的 action 为 name/cos:GetService，resource为 * 。

示例

授予查询存储桶列表操作权限的策略详细内容如下：

```
{
"version": "2.0",
"statement": [
{
"action": [
"name/cos:GetService"
],
"effect": "allow",
"resource": [
"*"
]
}
]
}
```

Bucket API

Bucket API 策略的 resource 可以归纳为以下几种情况：

- 可操作任意地域的存储桶，策略的 resource 为 * 。
- 只可操作指定地域的存储桶，如只可操作 APPID 为1250000000 ，地域为 ap-beijing 的存储桶 examplebucket-1250000000，则策略的 resource 为 qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/* 。
- 只可操作指定地域且指定名称的存储桶，如只可操作 APPID 为1250000000 ，地域为 ap-beijing 且名称为 examplebucket-1250000000 的存储桶， 则策略的 resource 为 qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/ 。

Bucket API 策略的 action 则因操作不同而取值不同，以下列举所有 Bucket API 授权策略。

创建存储桶

API 接口为 PUT Bucket，若授予其操作权限，则策略的 action 为 name/cos:PutBucket。

示例

授予可在 APPID 为1250000000 ，地域为 ap-beijing 中创建任意名称的存储桶的操作权限， 其策略详细内容如下：

```
{
"version": "2.0",
"statement": [
{
"action": [
"name/cos:PutBucket"
],
"effect": "allow",
"resource": [
"qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/"
]
}
]
}
```

检索存储桶及其权限

API 接口为 HEAD Bucket，若授予其操作权限，则策略的 action 为 name/cos:HeadBucket。

示例

授予只能检索 APPID 为1250000000 ，地域为 ap-beijing ，存储桶为 examplebucket-1250000000 的操作权限，其策略详细内容如下：

```
{
"version": "2.0",
"statement": [
{
"action": [
"name/cos:HeadBucket"
],
"effect": "allow",
"resource": [
"qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/"
]
}
]
}
```

查询对象列表

API 接口为 GET Bucket，若授予其操作权限，则策略的 action 为 name/cos:GetBucket。

示例

授予只能查询 APPID 为1250000000 ，地域为 ap-beijing，存储桶为 examplebucket-1250000000 的对象列表的操作权限，其策略详细内容如下：

```
{
"version": "2.0",
"statement": [
{
"action": [
"name/cos:GetBucket"
],
"effect": "allow",
"resource": [
"qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/"
]
}
]
}
```

删除存储桶

API 接口为 Delete Bucket，若授予其操作权限，则策略的 action 为 name/cos:DeleteBucket。

示例

授予只能删除 APPID 为1250000000 ，地域为 ap-beijing ，存储桶为 examplebucket-1250000000 的存储桶的操作权限，其策略详细内容如下：

```
{
"version": "2.0",
"statement": [
{
"action": [
"name/cos:DeleteBucket"
],
"effect": "allow",
"resource": [
"qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/"
]
}
]
}
```

设置存储桶 ACL

API 接口为 Put Bucket ACL，若授予其操作权限，则策略的 action 为 name/cos:PutBucketACL。

示例

授予只能设置 APPID 为1250000000 ，地域为 ap-beijing ，存储桶为 examplebucket-1250000000 的 ACL 的操作权限，其策略详细内容如下：

```
{
"version": "2.0",
"statement": [
{
"action": [
"name/cos:PutBucketACL"
],
"effect": "allow",
"resource": [
"qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/"
]
}
]
}
```

查询存储桶 ACL

API 接口为 GET Bucket acl，若授予其操作权限，则策略的 action 为 name/cos:GetBucketACL。

示例

授予只能获取 APPID 为1250000000 ，地域为 ap-beijing ，存储桶为 examplebucket-1250000000 的 ACL 的操作权限，其策略详细内容如下：

```
{
"version": "2.0",
"statement": [
{
"action": [
"name/cos:GetBucketACL"
],
"effect": "allow",
"resource": [
"qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/"
]
}
]
}
```

设置跨域配置

API 接口为 PUT Bucket cors，若授予其操作权限，则策略的 action 为 name/cos:PutBucketCORS。

示例

授予只能设置 APPID 为1250000000 ，地域为 ap-beijing ，存储桶为 examplebucket-1250000000 的跨域配置的操作权限，其策略详细内容如下：

```
{
"version": "2.0",
"statement": [
{
"action": [
"name/cos:PutBucketCORS"
],
"effect": "allow",
"resource": [
"qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/"
]
}
]
}
```

查询跨域配置

API 接口为 GET Bucket cors，若授予其权限，则策略的 action 为 name/cos:GetBucketCORS。

示例

授予只能查询 APPID 为1250000000 ，地域为 ap-beijing ，存储桶为 examplebucket-1250000000 的跨域配置的操作权限，其策略详细内容如下：

```
{
"version": "2.0",
"statement": [
{
"action": [
"name/cos:GetBucketCORS"
],
"effect": "allow",
"resource": [
"qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/"
]
}
]
}
```

删除跨域配置

API 接口为 DELETE Bucket cors，若授予其操作权限，则策略的 action 为 name/cos:DeleteBucketCORS。

示例

授予只能删除 APPID 为1250000000 ，地域为 ap-beijing ，存储桶为 examplebucket-1250000000 的跨域配置的操作权限，其策略详细内容如下：

```
{
"version": "2.0",
"statement": [
{
"action": [
"name/cos:DeleteBucketCORS"
],
"effect": "allow",
"resource": [
"qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/"
]
}
]
}
```

设置生命周期

API 接口为 PUT Bucket lifecycle，若授予其操作权限，则策略的 action 为 name/cos:PutBucketLifecycle 。

示例

授予只能设置 APPID 为1250000000 ，地域为 ap-beijing ，存储桶为 examplebucket-1250000000 的生命周期配置的操作权限，其策略详细内容如下：

```
{
"version": "2.0",
"statement": [
{
"action": [
"name/cos:PutBucketLifecycle"
],
"effect": "allow",
"resource": [
"qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/"
]
}
]
}
```

查询生命周期

API 接口为 GET Bucket lifecycle，若授予其操作权限，则策略的 action 为 name/cos:GetBucketLifecycle。

示例

授予只能查询 APPID 为1250000000 ，地域为 ap-beijing ，存储桶为 examplebucket-1250000000 的生命周期配置的操作权限，其策略详细内容如下：

```
{
"version": "2.0",
"statement": [
{
"action": [
"name/cos:GetBucketLifecycle"
],
"effect": "allow",
"resource": [
"qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/"
]
}
]
}
```

删除生命周期

API 接口为 DELETE Bucket lifecycle，若授予其操作权限，则策略的 action 为 name/cos:DeleteBucketLifecycle。

示例

授予只能删除 APPID 为1250000000 ，地域为 ap-beijing，存储桶为 examplebucket-1250000000 的生命周期配置的操作权限，其策略详细内容如下：

```
{
"version": "2.0",
"statement": [
{
"action": [
"name/cos:DeleteBucketLifecycle"
],
"effect": "allow",
"resource": [
"qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/"
]
}
]
}
```

查询分块上传

查询存储桶中正在分块上传信息，若授予其操作权限，则策略的 action 为 name/cos:ListMultipartUploads。

示例

授予只能查询 APPID 为1250000000 ，地域为 ap-beijing ，存储桶为 examplebucket-1250000000 中的正在分块上传信息的操作权限，其策略详细内容如下：

```
{
"version": "2.0",
"statement": [
{
"action": [
"name/cos:ListMultipartUploads"
],
"effect": "allow",
"resource": [
"qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/"
]
}
]
}
```

Object API

Object API 策略的 resource 可以归纳为以下几种情况：

- 可操作任意对象，策略的 resource 为 `*`。

- 只可操作指定存储桶中的任意对象，如只可操作 APPID 为1250000000 ， 地域为 ap-beijing，且名称为 examplebucket-1250000000 的存储桶中的任意对象，则策略的 resource 为 qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/* 。
- 只可操作指定存储桶 且 指定路径前缀下的任意对象，如只可操作 APPID 为1250000000 ， 地域为 ap-beijing，存储桶为 examplebucket-1250000000，路径前缀为 doc 下的任意对象，则策略的 resource 为 qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/doc/* 。
- 只可操作指定绝对路径的对象，如只可操作 APPID 为1250000000 ， 地域为 ap-beijing，存储桶为 examplebucket-1250000000，绝对路径为 doc/audio.mp3 的对象，则策略的 resource 为 qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/doc/audio.mp3 。

Object API 策略的 action 则因操作不同而取值不同，以下列举所有 Object API 授权策略。

简单上传对象

API 接口为 PUT Object，若授予其操作权限，则策略的 action为 name/cos:PutObject。

示例

授予只能在 APPID 为1250000000 ，地域为 ap-beijing，存储桶为 examplebucket-1250000000，路径前缀为 doc 下进行简单上传的操作权限，其策略详细内容如下：

```
{
"version": "2.0",
"statement": [
{
"action": [
"name/cos:PutObject"
],
"effect": "allow",
"resource": [
"qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/doc/*"
]
}
]
}
```

分块上传

分块上传包含 Initiate Multipart Upload，List Multipart Uploads，List Parts，Upload Part，Complete Multipart Upload，Abort Multipart Upload。若授予其操作权限，则策略的 action 为：
"name/cos:InitiateMultipartUpload","name/cos:ListMultipartUploads","name/cos:ListParts","name/cos:UploadPart","name/cos:CompleteMultipartUpload","name/cos:AbortMultipartUpload" 的集合。

示例

授予只能在 APPID 为1250000000 ，地域为 ap-beijing ，存储桶为 examplebucket-1250000000，路径前缀为 doc 下进行分块上传的操作权限，其策略详细内容如下：

```
{
"version": "2.0",
"statement": [
{
"action": [
"name/cos:InitiateMultipartUpload",
"name/cos:ListMultipartUploads",
"name/cos:ListParts",
"name/cos:UploadPart",
"name/cos:CompleteMultipartUpload",
"name/cos:AbortMultipartUpload"
],
"effect": "allow",
"resource": [
"qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/doc/*"
]
}
]
}
```

表单上传对象

API 接口为 POST Object，若授予其操作权限，则策略的 action 为 name/cos:PostObject。

示例

授予只能在 APPID 为1250000000 ，地域为 ap-beijing ，存储桶为 examplebucket-1250000000，路径前缀为 doc 下进行 POST 上传的操作权限，其策略详细内容如下：

```
{
"version": "2.0",
"statement": [
{
"action": [
"name/cos:PostObject"
],
"effect": "allow",
"resource": [
"qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/doc/*"
]
}
]
}
```

查询对象元数据

API 接口为 HEAD Object，若授予其操作权限，则策略的 action 为 name/cos:HeadObject。

示例

授予只能查询 APPID 为1250000000 ，地域为 ap-beijing ，存储桶为 examplebucket-1250000000，路径前缀为 doc 中的对象的操作权限，其策略详细内容如下：

```
{
"version": "2.0",
"statement": [
{
"action": [
"name/cos:HeadObject"
],
"effect": "allow",
"resource": [
"qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/doc/*"
]
}
]
}
```

下载对象

API 接口为 GET Object，若授予其操作权限，则策略的 action 为 name/cos:GetObject。

示例

授予只能下载 APPID 为1250000000 ，地域为 ap-beijing，存储桶为 examplebucket-1250000000，路径前缀为 doc 中的对象的操作权限，其策略详细内容如下：

```
{
"version": "2.0",
"statement": [
{
"action": [
"name/cos:GetObject"
],
"effect": "allow",
"resource": [
"qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/doc/*"
]
}
]
}
```

复制对象

API 接口为 Put Object Copy，若授予其操作权限，则策略的目标对象的 action 为 name/cos:PutObject，和源对象的 action 为 name/cos:GetObject。

示例

授予在 APPID 为1250000000 ，地域为 ap-beijing，存储桶为 examplebucket-1250000000 的路径前缀为 doc 和路径前缀为 doc2 间进行分块复制的操作权限，其策略详细内容如下：

```
{
"version": "2.0",
"statement": [
{
"action": [
"name/cos:PutObject"
],
"effect": "allow",
"resource": [
"qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/doc/*"
]
},
{
"action": [
"name/cos:GetObject"
],
"effect": "allow",
"resource": [
"qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/doc2/*"
]
}
]
}
```

其中 "qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/doc2/*" 为源对象。

复制分块

API 接口为 Upload Part - Copy，若授予其操作权限，则策略的目标对象的 action 为 action 为 "name/cos:InitiateMultipartUpload","name/cos:ListMultipartUploads","name/cos:ListParts","name/cos:PutObject","name/cos:CompleteMultipartUpload","name/cos:AbortMultipartUpload" 集合， 和源对象的 action 为 name/cos:GetObject。

示例

授予在 APPID 为1250000000 ，地域为 ap-beijing，存储桶为 examplebucket-1250000000 的路径前缀为 doc 和路径前缀为 doc2 间进行分块复制的操作权限，其策略详细内容如下：

```
{
"version": "2.0",
"statement": [
{
"action": [
"name/cos:InitiateMultipartUpload",
"name/cos:ListMultipartUploads",
"name/cos:ListParts",
"name/cos:PutObject",
"name/cos:CompleteMultipartUpload",
"name/cos:AbortMultipartUpload"
],
"effect": "allow",
"resource": [
"qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/doc/*"
]
},
{
"action": [
"name/cos:GetObject"
],
"effect": "allow",
"resource": [
"qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/doc2/*"
]
}
]
}
```

其中 "qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/doc2/*" 为源对象。

设置对象 ACL

API 接口为 Put Object ACL，若授予其操作权限，则策略的 action 为 name/cos:PutObjectACL 。

示例

授予只能设置 APPID 为1250000000 ，地域为 ap-beijing ，存储桶为 examplebucket-1250000000，路径前缀为 doc 中的对象 ACL 操作权限，其策略详细内容如下：

```
{
"version": "2.0",
"statement": [
{
"action": [
"name/cos:PutObjectACL"
],
"effect": "allow",
"resource": [
"qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/doc/*"
]
}
]
}
```

查询对象 ACL

API 接口为 Get Object ACL，若授予其操作权限，则策略的 action 为 name/cos:GetObjectACL。

示例

授予只能查询 APPID 为1250000000 ，地域为 ap-beijing，存储桶为 examplebucket-1250000000，路径前缀为 doc 中的对象 ACL 操作权限，其策略详细内容如下：

```
{
"version": "2.0",
"statement": [
{
"action": [
"name/cos:GetObjectACL"
],
"effect": "allow",
"resource": [
"qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/doc/*"
]
}
]
}
```

预请求跨域配置

API 接口为 OPTIONS Object，若授予其操作权限，则策略的 action 为 name/cos:OptionsObject。

示例

授予只能在 APPID 为1250000000 ，地域为 ap-beijing，存储桶为 examplebucket-1250000000，路径前缀为 doc 下进行 Options 请求 的操作权限，其策略详细内容如下：

```
{
"version": "2.0",
"statement": [
{
"action": [
"name/cos:OptionsObject"
],
"effect": "allow",
"resource": [
"qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/doc/*"
]
}
]
}
```

恢复归档对象

API 接口为 Post Object Restore，若其作权限，则策略的 action 为 name/cos:PostObjectRestore。

示例

授予只能在 APPID 为1250000000 ，地域为 ap-beijing，存储桶为 examplebucket-1250000000，路径前缀为 doc 下进行恢复归档的操作权限，其策略详细内容如下：

```
{
"version": "2.0",
"statement": [
{
"action": [
"name/cos:PostObjectRestore"
],
"effect": "allow",
"resource": [
"qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/doc/*"
]
}
]
}
```

删除单个对象

API 接口为 DELETE Object，若授予其操作权限，则策略的 action 为 name/cos:DeleteObject。

示例

授予只能删除 APPID 为1250000000 ，地域为 ap-beijing ，存储桶为 examplebucket-1250000000 中的 audio.mp3 这个对象的操作权限，其策略详细内容如下：

```
{
"version": "2.0",
"statement": [
{
"action": [
"name/cos:DeleteObject"
],
"effect": "allow",
"resource": [
"qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/audio.mp3"
]
}
]
}
```

删除多个对象

API 接口为 DELETE Multiple Objects，若授予其操作权限，则策略的 `action` 为 `name/cos:DeleteObject` 。

示例

授予只能批量删除 APPID 为1250000000 ，地域为 ap-beijing，存储桶为 examplebucket-1250000000 中的 audio.mp3 和 video.mp4 两个对象的操作权限，其策略详细内容如下：

```
{
"version": "2.0",
"statement": [
{
"action": [
"name/cos:DeleteObject"
],
"effect": "allow",
"resource": [
"qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/audio.mp3",
"qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/video.mp4"
]
}
]
}
```

常见场景授权策略

授予所有资源完全读写权限

授予所有资源完全读写权限，其策略详细内容如下：

```
{
"version": "2.0",
"statement": [
{
"action": [
"*"
],
"effect": "allow",
"resource": [
"*"
]
}
]
}
```

授予所有资源只读权限

授予所有资源只读权限，其策略详细内容如下：

```
{
"version": "2.0",
"statement": [
{
"action": [
"name/cos:HeadObject",
"name/cos:GetObject",
"name/cos:ListObject",
"name/cos:OptionsObject"
],
"effect": "allow",
"resource": [
"*"
]
}
]
}
```

授予指定路径前缀的读写操作

授予用户只能访问存储桶 examplebucket-1250000000 中路径前缀为 doc 下的文件，且无法操作其它路径文件的操作权限，该策略详细内容如下：

```
{
"version": "2.0",
"statement": [
{
"action": [
"*"
],
"effect": "allow",
"resource": [
"qcs::cos:ap-shanghai:uid/1250000000:examplebucket-1250000000/doc/*"
]
}
]
}
```

用于前端直传COS的临时密钥安全指引

最近更新时间: 2025-01-15 17:01:00

简介

在移动应用和 Web 应用中，您可以通过 iOS/Android/JavaScript SDK 在前端直接向 COS 发起请求，此时数据的上传和下载可以不经过您的后端服务器，既节约了您后端服务器的带宽和负载，还可以充分利用 COS 的带宽和全球加速等能力，提升您的应用体验。

在实际应用中，您需要使用临时密钥作为前端的 COS 请求签名，以防止永久密钥的泄漏及越权访问等问题。然而，即便是使用临时密钥，如果您在生成临时密钥时指定了过大的权限或路径，那么同样有可能发生越权访问等问题，将对您的应用带来一定的风险，本文重点介绍了部分风险案例，以及您应当遵守的安全规范，以便让您的应用能够安全的使用 COS。

前提条件

本文假定您已经充分理解临时密钥的相关概念，能够生成及使用临时密钥来请求 COS。有关临时密钥的生成及使用指引，请参见 临时密钥生成及使用指引 文档。

反面案例与安全规范

反面案例一：资源（resource）超范围限定

应用 A 在注册用户上传头像中使用到了 COS，每个注册用户的头像拥有固定的对象键 `app/avatar/<Username>.jpg` ，同时还会包含头像的不同尺寸，对应的对象键分别为 `app/avatar/<Username>_m.jpg` 和 `app/avatar/<Username>_s.jpg` ，后端为了方便使用，在生成临时密钥时直接将 resource 指定为 `<BucketName-APPID>/app/avatar/*` ，此时恶意用户通过网络抓包等手段获取到生成的临时密钥后，可以覆盖上传任何用户的头像，产生越权访问，用户的合法头像数据被覆盖导致丢失。

安全规范

resource 代表临时密钥所允许访问的资源路径，此时需要充分考虑该路径所覆盖的最终用户，原则上 resource 所指定的资源要求仅能被单一用户所使用。此案例中指定的 `<BucketName-APPID>/app/avatar/*` 显然会覆盖所有用户，因此存在安全漏洞。

在该案例中，可以考虑将用户的头像路径修改为 `app/avatar/<Username>/<size>.jpg` ，此时可以将 resource 指定为 `<BucketName-APPID>/app/avatar/<Username>/*` 来满足规范要求；此外，resource 字段支持以数组的形式传入多个值。因此，您也可以显式指定多个 resource 值来完全限定用户有权限访问的最终资源路径，例如：

```
"resource": [
"<BucketName-APPID>/app/avatar/<Username>.jpg",
"<BucketName-APPID>/app/avatar/<Username>_m.jpg",
"<BucketName-APPID>/app/avatar/<Username>_s.jpg"
]
```

反面案例二：操作（action）超范围限定

应用 B 提供一个公共的照片墙功能，所有照片均存放置在 `app/photos/*` 下面，客户端同时需要列出对象（GET Bucket）和下载对象（GET Object）操作，后端为了方便使用，在生成临时密钥时直接将 action 指定为了 `name/cos:*` ，此时恶意用户通过网络抓包等手段获取到生成的临时密钥后，可以对该资源路径下的任何对象执行所有对象操作（例如上传和删除等操作），产生越权访问，导致数据丢失，影响线上业务。

安全规范

action 代表临时密钥所允许请求的操作，原则上不允许使用 `name/cos:*` 等允许所有操作的临时密钥下发至前端，必须明确列出所有需要用到的操作，同时如果各操作所需的资源路径不通，则需要**操作**与**资源**路径单独匹配，而不应合并处理。

在该案例中，应当使用 `"action": [ "name/cos:GetBucket", "name/cos:GetObject" ]` 指明具体的操作。

反面案例三：资源与操作超范围限定

应用 C 提供一个管理工具，允许用户列出并下载所有人的文件（ `app/files/*` ），但只能上传和删除个人目录下的文件（ `app/files/<Username>/*` ），后端为了方便使用，在生成临时密钥时将2种权限，共4种操作（action）混合在一起。2种权限对应的资源路径也混合在一起，此时的临时密钥将具备资源路径中指定的更大权限，即用户可以列出、下载、上传和删除所有人的文件，恶意用户可以据此篡改或删除他人的文件，产生越权访问，用户的合法数据将暴露在风险之中。

安全规范

对于多个 action 与 resource 的组合，不应简单的将它们分别合并，而应当通过多个 statement 的形式组合在一起，避免简单的分别合并导致权限扩大化。

在该案例中，应当使用

```
"statement": [
{
"effect": "allow",
"action": [
"name/cos:GetBucket",
"name/cos:GetObject"
],
"resource": "<BucketName-APPID>/app/files/*"
},
{
"effect": "allow",
"action": [
"name/cos:PutObject",
"name/cos:DeleteObject"
],
"resource": "<BucketName-APPID>/app/files/<Username>/*"
}
]
```

反面案例四：越权获取临时密钥

应用 D 提供一个论坛应用，论坛的帖子附件保存在 COS 上，该论坛应用分为不同的用户级别，只有发帖数达到一定阈值的活跃用户才能看到某个版面内的帖子和附件，对于一些公共版面，所有用户均可以查看其中的帖子和附件。 在使用 COS 时，后端生成临时密钥的接口会根据前端传入的版面 ID ，生成允许下载对应版面附件的临时密钥。但在实现过程中，后端没有判断具体请求的用户是否有权限访问指定的版面 ID，导致任何人均可请求该接口获取可访问私密版面附件的临时密钥，产生越权访问，需要被限制访问的资源未能被有效限制，导致数据意外泄漏。

安全规范

对于获取临时密钥的接口，除了要保证获取到的临时密钥本身的权限在预期范围中，还应判断实际的业务场景，即正确的权限是否被正确的用户所请求，避免低权限用户拿到高权限的临时密钥。

在本案例中，后端接口还应该判断当前请求用户是否具备访问特定版面的权限，如无权访问则不允许返回临时密钥。

总结

以上几个案例说明了在预期外扩大临时密钥的权限可能导致的安全风险。由于前端直传 COS 时，恶意用户比较容易获取到临时密钥内容，因此该场景相对于通过后端访问 COS 来说，需要开发者们更加注重权限的控制。

本文提到的安全规范即最小权限原则，在实际应用中，根据 action 和 resource 可以枚举出所有可能的权限，例如3种 action 和2个 resource，可以计算出3 x 2=6个被允许访问的资源和对应的操作，您可以据此评估每一种情况是否符合预期，如超出预期权限范围，则应当考虑通过枚举多个 statement 的方式拆分权限。

此外，用于生成临时密钥的接口本身也要充分考虑身份认证和鉴权处理，只有获取临时密钥的行为是安全的，这样得到的临时密钥才能确保真正安全。安全链条上，不能有任何一处疏漏！

临时密钥生成及使用指引

最近更新时间: 2025-01-15 17:01:00

获取临时密钥

CAM 的 STS 临时秘钥，以云 API 的形式提供。目前云 API 提供各种语言的 SDK，用户可以使用云 API 或 SDK 来申请临时密钥三元组。

通过 API 获取

STS 云 API 接口参数

STS 云 API 的接口参数说明 如下：

字段	是否必选	类型	描述
name	是	String	用户昵称
policy	是	String	策略语法（需要 base64 编码）
durationSeconds	否	Int	指定临时证书的有效期，单位：秒。 不传的话默认 1800 秒，最长有效期：2 小时（7200 秒）。
Signature	是	String	请求签名，用来验证此次请求的合法性，需要用户根据实际的输入参数计算得出。
Timestamp	是	Int	时间戳
Nonce	是	Int	随机数
SecretId	是	String	在云 API 密钥上申请的标识身份的 SecretId，一个 SecretId 对应唯一的 SecretKey，而 SecretKey 会用来生成请求签名 Signature。
Action	是	String	Action = GetFederationToken
Region	否	String	地域参数，用来标识希望操作哪个地域的实例。

返回值

字段	类型	描述
credentials	Object	对象里面包含 Token，tmpSecretId，tmpSecretKey 三元组。
--token	String	Token 值做鉴权时使用。
--tmpSecretId	String	tmpSecretId 签名时使用。
--tmpSecretKey	String	tmpSecretKey 签名时使用。
federatedUser	String	返回标识用户。 示例：`qcs::sts::123456789012:federated-user/Bob`
expiredTime	Int	证书无效的时间戳

访问请求示例

```
http://imgcache.finance.cloud.tencent.com:80sts.api.qcloud.com/v2/index.php?Action=GetFederationToken&Nonce=652650920&Region=gz&RequestClient=SDK_JAVA_1.3&SecretId=SecretIDXXXXX&Signature=Bv4G9gCkDVy/lhiDHg2eOIo1PPI=&Timestamp=1494561662&name=Sevenyou&policy=eyJzdGF0ZW1lbnQiOiBbeyJhY3Rpb24iOiBbIm5hbWUvY29zOkdldE9iamVjdCIsIm5hbWUvY29zOlB1dE9iamVjdCJdLCJlZmZlY3QiOiAiYWxsb3ciLCJyZXNvdXJjZSI6WyJxY3M6OmNvczpjbi1ub3J0aDp1aWQvMTI1MjQ0ODcwMzpwcmVmaXgvLzEyNTI0NDg3MDMvcmFiYml0bGl1dGovKiJdfV0sInZlcnNpb24iOiAiMi4wIn0=
```

返回内容示例

```
{
"codeDesc": "Success",
"message": "",
"data": {
"expiredTime": 1494563462,
"credentials": {
```

```
"sessionToken": "sessionTokenXXXXX",
"tmpSecretId": "tmpSecretIdXXXXX",
"tmpSecretKey": "tmpSecretKeyXXXXX"
}
},
"code": 0
}
```

通过 SDK 获取

以云 API 提供的 Java SDK 为例：

```
import com.qcloud.Utilities.Json.JSONObject;

public class Demo {
public static void main(String[] args) {
/* 如果是循环调用下面举例的接口，需要从此处开始你的循环语句。切记！ */
TreeMap<String, Object> config = new TreeMap<String, Object>();
config.put("SecretId", "SecretIdAAAA");
config.put("SecretKey", "SecretKeyZZZZ");

/* 请求方法类型 POST、GET */
config.put("RequestMethod", "GET");

QcloudApiModuleCenter module = new QcloudApiModuleCenter(new Sts(),
config);

TreeMap<String, Object> params = new TreeMap<String, Object>();
/* 将需要输入的参数都放入 params 里面，必选参数是必填的。 */
/* DescribeInstances 接口的部分可选参数如下 */
params.put("name", "sevenyou");
String policy = "{\"statement\": [{\"action\": [\"name/cos:GetObject\",\"name/cos:PutObject\"],\"effect\": \"allow\",\"resource\":[\"qcs::cos:ap-beijing:uid/12
345678910:prefix//12345678910/sevenyou/*\"]}],\"version\": \"2.0\"}";
params.put("policy", policy);

/* 在这里指定所要用的签名算法，不指定默认为 HmacSHA1*/
//params.put("SignatureMethod", "HmacSHA256");

/* generateUrl 方法生成请求串, 可用于调试使用 */
System.out.println(module.generateUrl("GetFederationToken", params));
String result = null;
try {
/* call 方法正式向指定的接口名发送请求，并把请求参数 params 传入，返回即是接口的请求结果。 */
result = module.call("GetFederationToken", params);
JSONObject json_result = new JSONObject(result);
System.out.println(json_result);
} catch (Exception e) {
System.out.println("error..." + e.getMessage());
}
}
}
```

申请临时三元组时，需要描述策略 Policy，示例中以 IP 做限制的 Policy 可能如下：

```
{
"statement": [
{
"action": [
"name/cos:GetObject",
"name/cos:HeadObject"
],
"condition": {
"ip_equal": {
"qcs:ip": [
"101.226.226.185/32"
]
}
},
"effect": "allow",
"resource": [
```

```
"qcs::cos:cn-east:uid/12345678910:prefix//12345678910/sevenyou/*"
]
}
],
"version": "2.0"
}
```

Policy 描述请参考 策略语法。

> **注意：** resource 字段中的 prefix 后跟 // 。uid后面跟的是appid，而不是UIN。

使用临时密钥访问 COS

COS 访问通过 x-cos-security-token 字段来传递临时 Token，而临时 SecretId 和 SecretKey 则用来生成密钥，以 Jave SDK 为例使用临时密钥访问 COS，示例如下：

从 Github 下载：Java SDK

> **注意：** 下列代码，需要加入 x-cos-security-token 字段，传递 STS 的临时密钥。

```
public class Demo {
public static void main(String[] args) throws Exception {
// 用户基本信息
String appid = "12345678910";
String secret_id = "TmpSecretId";
String secret_key = "TmpSecretKey";
String sessionToken = "TmpToken";

// 设置秘钥
COSCredentials cred = new BasicCOSCredentials(appid, secret_id, secret_key);

// 生成 cos 客户端对象
COSClient cosClient = new COSClient(cred, clientConfig);

// 创建 bucket
// bucket 数量上限 200 个, bucket 是重操作, 一般不建议创建如此多的 bucket
// 重复创建同名 bucket 会报错
String bucketName = "rabbitliutj";
// 上传 object, 建议 20M 以下的文件使用该接口
File localFile = new File("D:\\test\\rabbit_test.txt");
String key = "/upload_single_demo.txt";
PutObjectRequest putObjectRequest = new PutObjectRequest(bucketName, key, localFile);
ObjectMetadata objectMetadata = new ObjectMetadata();
objectMetadata.setSecurityToken (sessionToken);
putObjectRequest.setMetadata(objectMetadata);
PutObjectResult putObjectResult = cosClient.putObject(putObjectRequest);
System.out.println(putObjectResult);

// 关闭客户端 (关闭后台线程)
cosClient.shutdown();
}
}
```

授予其他主帐号下的子帐号操作名下存储桶的权限

最近更新时间: 2025-01-15 17:01:00

操作场景

当前主账号 A（APPID 为1250000000）名下有存储桶 examplebucket1-1250000000 和 examplebucket2-1250000000 ，另有一主账号 B 及其子账号 B0，B0 由于业务需要，希望能够操作账号 A 名下的两个存储桶，下面将为您介绍如何进行相关的授权操作。

操作步骤

授予主账号 B 操作 A 名下存储桶的权限

1. 使用主账号 A 登录 对象存储 COS 控制台。

2. 单击【存储桶列表】，找到需要授权的存储桶，单击其名称进入存储桶详情页。

3. 单击【权限管理】页签，切换到权限管理页面。

4. 找到【Policy 权限设置】配置项，单击【添加策略】，填写如下表单项：

   - **效力**：允许。
   - **用户**：单击添加用户，用户类型选择根账号，账号 ID 填写主账号 B 的 UIN，例如100000000002。
   - **资源**：根据需要选择，默认为整个存储桶。
   - **资源路径**：仅指定资源时需要填写，根据需要填写。
   - **操作**：单击添加操作，选择所有操作，如仅需授权部分操作，也可以选择一个或多个实际需要的操作。
   - **条件**：根据需要填写，如不需要可留空。

5. 单击【确定】，此时主账号 B 将拥有操作该存储桶的相关权限。

6. 如需授权其他存储桶，可重复上述步骤。

授予子账号 B0 操作 A 名下存储桶的权限

1. 使用主账号 B 登录访问管理 CAM 控制台的 策略管理 页面。

2. 选择【新建自定义策略】>【按策略语法创建】，随后选择空白模板，单击【下一步】。

3. 填写如下表单：

   - **策略名称**：自行定义一个不重复且有意义的策略名称，例如 cos-child-account。
   - **备注**：可选，自行编写。
   - **编辑策略内容**：

```
{
"version": "2.0",
"statement": [
{
```

```
"action": "cos:*",
"effect": "allow",
"resource": "qcs::cos::uid/1250000000:examplebucket1-1250000000/*"
}
]
}
```

其中，`uid/1250000000` 中的1250000000为主账号 A 的 uid，`examplebucket1-1250000000` 为被授权的存储桶名称。存储桶资源 `examplebucket1-1250000000/*` 也可以直接使用 `*`，代表将所有主账号 B 有权操作的 A 名下的存储桶均授权给子账号 B0。

4. 单击【创建策略】，完成策略的创建。

5. 在策略列表中找到刚才已创建的策略，并单击右侧的【关联用户/组】。

6. 在**关联用户/用户组**对话框中，勾选子账号 B0，单击【确定】。

7. 完成授权操作，即可使用子账号 B0 的密钥，测试操作 A 名下的存储桶。

性能优化
请求速率与性能优化

最近更新时间: 2025-01-15 17:01:00

本文探讨请求速率性能优化在云平台 COS 对象存储上的最佳实践。

云平台对象存储提供的典型工作负载能力为每秒 100 个 PUT/LIST/DELETE 等请求，或者每秒 400 个 GET 请求。如果您的工作负载超过了上述能力，建议您遵循本指南实现请求速率的性能扩展和优化。

> **注意：** 请求负载指的是每秒发起的请求数量，非并发连接数。即您在保持数千连接数的同时，仍可以在 1s 时间内发送数百个新连接请求。

云平台对象存储支持性能扩展，以支持更高的请求速率。如果预计存储桶的请求速率会超过每秒 500 个 PUT/LIST/DELETE 请求，建议您联系我们，以便于为工作负载做好准备，避免遇到请求的限制。

> **注意：** 如果您的混合请求负载只是偶尔达到每秒 500 个，并在突发时不超过每秒 800 个，您可无需遵循本指南。

混合请求负载

当需要上传大量对象的时候，您选择的对象键可能会引发性能问题，以下将简述云平台对象存储对 Object 键值的存储方法。

云平台在对象存储的每一个服务地域都维护了存储桶 (Bucket) 和对象 (Object) 的键值作为索引，对象键以 UTF-8 二进制顺序保存在索引的多个分区中。对于大量的键值，例如使用时间戳或者字母顺序可能会耗尽键值所在分区的读写性能，以存储桶路径 `testbucket-123454321.cos.ap-beijing.myqcloud.com` 为例，以下列出了可能会耗尽索引性能的一些案例：

```
20170701/log120000.tar.gz
20170701/log120500.tar.gz
20170701/log121000.tar.gz
20170701/log121500.tar.gz
...
image001/indexpage1.jpg
image002/indexpage2.jpg
image003/indexpage3.jpg
...
```

如果您的业务典型负载超过每秒 500 个请求，则应避免使用上述案例中的顺序键值。当您的业务必须使用顺序编号或日期时间等字符作为对象键时，您可以使用一些方法向对象键名称添加随机前缀，即可实现在多个索引分区实现键值管理，提升集中负载的性能。以下提供了一些在键值中增加随机性的方法。

以下提供的所有方法，均是有可能提升单个存储桶访问性能的方法，如果您的业务典型负载超过每秒 500 个请求，在执行以下方法的同时，您仍需联系我们以便于为您的业务负载提前做好准备。

添加十六进制哈希前缀

最直接的增加对象键随机性的方式，就是在对象键名称的最前面添加哈希字符串作为前缀，例如可以在上传对象时，对路径键值进行 SHA1 或 MD5 哈希计算，并选取几位字符作为前缀添加到键值名称，通常 2~4 位的字符哈希前缀可以满足需要。

```
faf1-20170701/log120000.tar.gz
e073-20170701/log120500.tar.gz
333c-20170701/log121000.tar.gz
2c32-20170701/log121500.tar.gz
```

> **注意：** 由于云平台对象存储的键值索引是以 UTF-8 二进制顺序作为索引的，因此在执行列出对象（GET Bucket）操作时，您可能需要发起 65536 次列出对象操作，才能得到原来完整的 20170701 前缀结构。

添加枚举值前缀

如果您仍想要保留对象键的检索易用性，您可以针对您的文件类型枚举出一些前缀，实现对象分组，相同枚举值的前缀将共享所在索引分区的性能。

```
logs/20170701/log120000.tar.gz
logs/20170701/log120500.tar.gz
logs/20170701/log121000.tar.gz
...
images/image001/indexpage1.jpg
images/image002/indexpage2.jpg
```

```
images/image003/indexpage3.jpg
...
```

如果您在相同枚举前缀下，仍然有较高的访问负载，持续超过每秒 500 个请求，您可以参照上述添加十六进制哈希前缀的方式，在枚举值后继续添加哈希前缀执行多个索引分区，以实现更高性能的读写。

```
logs/faf1-20170701/log120000.tar.gz
logs/e073-20170701/log120500.tar.gz
logs/333c-20170701/log121000.tar.gz
...
images/0165-image001/indexpage1.jpg
images/a349-image002/indexpage2.jpg
images/ac00-image003/indexpage3.jpg
...
```

反转键值名称字符串

当您的业务不得不使用连续递增的 ID 或日期，或需要一次性上传大量的连续前缀对象时，如下述典型用法：

```
20170701/log0701A.tar.gz
20170701/log0701B.tar.gz
20170702/log0702A.tar.gz
20170702/log0702B.tar.gz
...
id16777216/album/hongkong/img20170701121314.jpg
id16777216/music/artist/tony/anythinggoes.mp3
id16777217/video/record20170701121314.mov
id16777218/live/show/date/20170701121314.mp4
...
```

如上的键值命名方式极易耗尽 `2017` 与 `id` 为前缀的键值所在的索引分区，此时将键值前缀的一部分反转后，则实现了一定的随机性。

```
10707102/log0701A.tar.gz
10707102/log0701B.tar.gz
20707102/log0702A.tar.gz
20707102/log0702B.tar.gz
...
61277761di/album/hongkong/img20170701121314.jpg
61277761di/music/artist/tony/anythinggoes.mp3
71277761di/video/record20170701121314.mov
81277761di/live/show/date/20170701121314.mp4
...
```

数据迁移
本地数据迁移至COS

最近更新时间: 2025-01-15 17:01:00

实践场景

对于拥有本地 IDC 的用户，对象存储 COS 在不同迁移类型上支持以下迁移方式，帮助用户将本地 IDC 的海量数据快速迁移至对象存储 COS。

迁移方式	说明
COS Migration（线上迁移）	COS Migration 是一个集成了 COS 数据迁移功能的一体化工具。用户只需要通过简单的配置操作，便可将数据快速迁移至 COS 中。

用户可依据数据迁移量、IDC 出口带宽、IDC 空闲机位资源、可接受的迁移完成时间等因素来考虑如何选择迁移方式。下图展示的是使用线上迁移时预估的时间消耗，可以看出，若此次迁移周期超过10天或者迁移数据量超过50TB，我们不建议您选择线上迁移。

> 注意：
>
> 1MB 以下的小文件数量较多、磁盘 IO 性能不足等也会影响到数据的迁移进度。

迁移实践

COS Migration

迁移操作步骤如下：

1. 安装 Java 环境。
2. 安装 COS Migration 工具。
3. 修改配置文件。
4. 启动工具。

具体的操作方法，请参见 COS Migration 工具 文档。

操作技巧

下面介绍如何配置 COS Migration 能最大程度提高迁移速度：

1. 根据自身网络环境调整区分大小文件的阈值和迁移并发度，实现大文件分块，小文件并发传输的最佳迁移方式。调整工具执行时间和设立带宽限制，保证自身业务运行不受迁移数据带宽占用影响。上述调整可在配置文件 config.ini 中 [common] 分节，修改如下参数进行调整：

参数名称	参数说明
smallFileThreshold	小文件阈值参数，大于等于这个阈值使用分块上传，默认设置为5MB。
bigFileExecutorNum	大文件并发度，默认值为8。 如果是通过外网来连接 COS，且带宽较小，请减小该并发度。
smallFileExecutorNum	小文件并发度，默认值为64。 如果是通过外网来连接 COS，且带宽较小，请减小该并发度。
executeTimeWindow	该参数定义迁移工具每天执行的时间段，其他时间则会进入休眠状态，休眠状态暂停迁移并会保留迁移进度，直到下一个时间窗口自动继续执行。

2. 采用分布式并行传输可以进一步加快迁移速度。用户可以考虑使用多台机器安装 COS Migration 并分别执行不同源数据的迁移任务。

第三方云存储数据迁移至COS

最近更新时间: 2025-01-15 17:01:00

实践背景

对于使用第三方云平台存储的用户，对象存储 COS 支持线上迁移，帮助用户将第三方云平台的存储数据快速迁移至对象存储 COS。

迁移方式	交互形式	区分大小文件的阈值	迁移并发度	HTTPS 安全传输
COS Migration	修改配置文件，非可视化操作	可自定义调整	针对大文件小文件分别定义并发度	可选择是否开启，关闭有利于加快迁移速度

线上迁移方式支持查看数据迁移进度、文件一致性校验、失败重传、断点续传等功能，能够满足用户数据基本的迁移需求。但这两种迁移方式在交互形式和功能特点等方面又有所差别，如上表所示。用户可根据以上的差异对比，选择最适合的一种方式进行数据迁移。

迁移实践

COS Migration

COS Migration 是一个集成了 COS 数据迁移功能的一体化工具。用户只需要通过简单的配置操作，便可将数据快速迁移至 COS 中。

迁移操作步骤如下：

1. 安装 Java 环境。
2. 安装 COS Migration 工具。
3. 修改配置文件。
4. 启动工具。

具体的操作方法，请参见 COS Migration 工具 文档。

操作技巧

下面介绍如何配置 COS Migration 能最大程度提高迁移速度：

1. 根据自身网络环境调整区分大小文件的阈值和迁移并发度，实现大文件分块，小文件并发传输的最佳迁移方式。调整工具执行时间和设立带宽限制，保证自身业务运行不受迁移数据带宽占用影响。上述调整可在配置文件 config.ini 中 [common] 分节，修改如下参数进行调整：

参数名称	参数说明
smallFileThreshold	小文件阈值参数，大于等于这个阈值使用分块上传，默认设置为5MB。
bigFileExecutorNum	大文件并发度，默认值为8。 如果是通过外网来连接 COS，且带宽较小，请减小该并发度。
smallFileExecutorNum	小文件并发度，默认值为64。 如果是通过外网来连接 COS，且带宽较小，请减小该并发度。
executeTimeWindow	该参数定义迁移工具每天执行的时间段，其他时间则会进入休眠状态，休眠状态暂停迁移并会保留迁移进度，直到下一个时间窗口自动继续执行。

2. 采用分布式并行传输可以进一步加快迁移速度。用户可以考虑使用多台机器安装 COS Migration 并分别执行不同源数据的迁移任务。

以URL作为源地址的数据迁移至COS

最近更新时间: 2025-01-15 17:01:00

实践背景

对于用户想要使用 URL 列表作为数据源地址进行数据迁移。对象存储 COS 支持线上迁移方式：

迁移方式	说明
COS Migration	COS Migration 是一个集成了 COS 数据迁移功能的一体化工具。用户只需要通过简单的配置操作，便可将数据快速迁移至 COS 中。

> 注意：
>
> 目前 COS 暂不支持含有鉴权信息的 URL 数据进行迁移。

迁移实践

COS Migration

迁移操作步骤如下：

1. 安装 Java 环境。
2. 安装 COS Migration 工具。
3. 修改配置文件。
4. 启动工具。

具体的操作方法，请参见 COS Migration 工具 文档。

操作技巧

下面介绍如何配置 COS Migration 能最大程度提高迁移速度：

1. 根据自身网络环境调整区分大小文件的阈值和迁移并发度，实现大文件分块，小文件并发传输的最佳迁移方式。调整工具执行时间和设立带宽限制，保证自身业务运行不受迁移数据带宽占用影响。上述调整可在配置文件 config.ini 中 [common] 分节，修改如下参数进行调整：

参数名称	参数说明
smallFileThreshold	小文件阈值参数，大于等于这个阈值使用分块上传，默认设置为5MB。
bigFileExecutorNum	大文件并发度，默认值为8。 如果是通过外网来连接 COS，且带宽较小，请减小该并发度。
smallFileExecutorNum	小文件并发度，默认值为64。 如果是通过外网来连接 COS，且带宽较小，请减小该并发度。
executeTimeWindow	该参数定义迁移工具每天执行的时间段，其他时间则会进入休眠状态，休眠状态暂停迁移并会保留迁移进度，直到下一个时间窗口自动继续执行。

2. 采用分布式并行传输可以进一步加快迁移速度。用户可以考虑使用多台机器安装 COS Migration 并分别执行不同源数据的迁移任务。

COS之间数据迁移

最近更新时间: 2025-01-15 17:01:00

实践背景

对于正在使用对象存储 COS 的用户，如果需要将一个存储桶的数据迁移至另一个对象存储 COS 中，我们建议您使用以下迁移方式：

- 线上迁移：跨地域复制

跨地域复制是对象存储 COS 针对存储桶的一项配置，通过配置跨地域复制规则，可以在不同地域的存储桶中自动、异步地复制**增量对象**。启用跨地域复制后，COS 将精确复制源存储桶中的对象内容（例如对象元数据和版本 ID 等）到目标存储桶中，复制的对象副本拥有完全一致的属性信息。此外，源存储桶中对于对象的操作，如上传对象、删除对象等操作，也将被复制到目标存储桶中。

Hadoop文件系统与COS之间的数据迁移

最近更新时间: 2025-01-15 17:01:00

简介

Hadoop Distcp（Distributed copy）主要是用于 Hadoop 文件系统内部或之间进行大规模数据复制的工具，它基于 Map/Reduce 实现文件分发、错误处理以及最终的报告生成。由于利用了 Map/Reduce 的并行处理能力，每个 Map 任务负责完成源路径中部分文件的复制，因此它可以充分利用集群资源来快速完成集群或 Hadoop 文件系统之间的大规模数据迁移。

由于 Hadoop-COS 实现了 Hadoop 文件系统的语义，因此利用 Hadoop Distcp 工具可以方便地在 COS 与其他 Hadoop 文件系统之间进行双向的数据迁移，本文就以 HDFS 为例，介绍 Hadoop 文件系统与 COS 之间利用 Hadoop Distcp 工具完成数据迁移的方式。

前提条件

1. Hadoop 集群中已经安装 Hadoop-COS 插件，并且正确配置了 COS 的访问密钥等。可使用如下 Hadoop 命令检查 COS 访问是否正常：

```
hadoop fs -ls cosn://examplebucket-1250000000/
```

如果能够正确地列出 COS Bucket 中的文件列表，则表示 Hadoop-COS 安装和配置正确，可以进行以下实践步骤。

2. COS 的访问账户必须要具备读写 COS 存储桶中目标路径的权限。

实践步骤

将 HDFS 中的数据复制到 COS 的存储桶中

通过 Hadoop Distcp 将本地 HDFS 集群中 `/test` 目录下的文件迁移到 COS 的 hdfs-test-1250000000 存储桶中。

1. 执行如下命令启动迁移：

```
hadoop distcp hdfs://10.0.0.3:9000/test cosn://hdfs-test-1250000000/
```

Hadoop Distcp 会启动 MapReduce 作业来执行文件复制任务，完成后会输出简单报表信息，如下图所示：

2. 执行 `hadoop fs -ls -R cosn://hdfs-test-1250000000/` 命令可以列出刚才已迁移到存储桶 hdfs-test-1250000000 的目录和文件。

将 COS 中存储桶的文件复制到本地 HDFS 集群

Hadoop Distcp 是一个支持不同集群和文件系统之间复制数据的工具，因此，将 COS 存储桶中的对象路径作为源路径，HDFS 的文件路径作为目标路径即可将 COS 中的数据文件复制到本地 HDFS：

```
hadoop distcp cosn://hdfs-test-1250000000/test hdfs://10.0.0.3:9000/
```

Hadoop distcp 的扩展参数

Hadoop distcp 工具支持丰富的运行参数，例如，可以通过 `-m` 来指定最大用于并行复制的 Map 任务数目， `-bandwidth` 来限制每个 map 所使用的最大带宽等。具体可参考 Apache Hadoop distcp 工具的官方文档：DistCp Guide。

数据容灾备份
基于跨地域复制的容灾高可用架构

最近更新时间: 2025-01-15 17:01:00

简介

对象存储 COS 为客户提供了99.95%的可用性和99.999999999%的可靠性。然而，由于自然灾害、光纤故障等诸多不可控因素的存在，云上数据的可用性和可靠性均无法达到100%，同时，部分行业由于业务的特殊性，例如金融行业，需要保证业务高可用和高可靠性。

为了实现企业业务的连续性和稳定性，满足企业对高可用和高可靠性的需求，对象存储提供了基于跨地域复制功能的数据容灾高可用方案。我们建议企业用云时，根据业务需要对云上数据进行容灾、备份，保障业务持续稳定运行。

本文主要介绍两个方面，首先介绍一种基于跨地域复制的云上业务主备切换的容灾方案，另一方面进一步介绍一种基于跨地域复制的高可用方案，通过跨地域复制、回源和 SCF、CDN 等多种产品和功能实现业务高可用。

基于跨地域复制的容灾备份方案

容灾需要满足三个要素：冗余（Redundance）、远距离（Remote）和数据全备份（Replication）。

- 冗余：即数据冗余，要求数据需要同时备份到另一个可用系统中。
- 远距离：指的是备份数据存储在相隔较远的另一个地域，因为灾害往往具有地理上的连续性，只有充分长的距离才能保障冗余数据的可用。
- 数据全备份：指的是备份数据零丢失。

COS 的跨地域复制功能可以实现增量数据的跨地域同步，用户上传的数据，根据其文件大小和地域距离远近，可以在几秒到几十分钟内拷贝到另一地域的存储桶中。基于跨地域复制，可以实现数据的异地冗余备份，从而实现业务容灾。有关跨地域复制的介绍，可参见 跨地域复制概述。开启跨地域复制需要先开启版本控制功能，有关版本控制的介绍可参见 版本控制概述。

基于跨地域复制的容灾备份架构示意图如下：

在这一架构下，客户的存储桶 A 和存储桶 B 互为主备。假设企业客户的数据存储在存储桶 A 上，另一地域的存储桶 B 是备用存储桶。该企业为了保障业务连续性和稳定性，为存储桶 A 和存储桶 B 分别配置了跨地域复制规则。在跨地域复制规则生效的情况下，存储桶 A 的增量数据会自动复制到存储桶 B 中，存储桶 B 的增量数据同样会自动复制到存储桶 A 中。

> 注意：
>
> 存储桶 A 中的增量数据复制到存储桶 B 后，虽然是存储桶 B 中的增量数据，但不会再被复制到存储桶 A 中。

正常情况下，企业的主读写请求链路均指向存储桶 A，所有增量数据将被自动增量同步复制到存储桶 B 中作为备份数据。客户侧可以在上传或者下载程序中加入网络质量检测的模块，在检测到主存储桶 A 宕机时，迅速将读写请求链路切换到备存储桶 B 中。

基于跨地域复制的高可用方案

上文介绍了一种基于跨地域复制的容灾备份方案，该方案能够利用云上已有产品和功能实现数据备份和容灾切换的工作。但真实业务运行状态可能复杂多样，上述的容灾备份方案未必能保障业务的高可用。因此，本小节提出一种基于跨地域复制的高可用方案，通过跨地域复制、回源和 SCF、CDN等多种产品和功能实现业务高可用。

基于跨地域复制的业务高可用架构示意图如下：

这一架构主要分为以下几个层次：

- **高可用层**：集成网络检测和业务调度，根据链路的连通率等指标进行链路切换，用户可以根据 SCF 实现（可参照上一小节介绍），也可以根据业务需求在客户端自行实现。
- **存储层**：一般情况下由 COS 不同地域的存储桶组成。

这一架构保障业务高可用的方式阐述如下：

1. 正常情况下，企业的主写请求链路均指向存储桶 A，所有增量数据将被自动同步复制到存储桶 B 中作为备份数据。

2. 当主存储桶 A 的链路不通时（例如拨测质量下降或者检测到上传失败），则客户端可以将写请求链路切换至主存储桶 B，此时所有增量数据同样将被自动同步复制到存储桶 A 中。
3. 客户还可以选择在自有源站或者其他云厂商上先备份一份冗余数据，同时给存储桶 B 配置回源策略。假设在极端情况下，主存储桶 A 和 B 链路都同时无法连通，那么在上传数据到存储桶 B 失败的情况下，存储桶 B 可以从源站拉取数据。

> 注意：
>
> - 全量冗余备份数据成本较高，客户也可以选择只冗余备份热数据（例如仅数小时内上传的文件），以减少数据存储成本。
> - 如果您选择了源站作为高可用架构中的一部分，那么您在设计该架构时请注意评估源站带宽以及其限制可能带来的影响。

参考文档

以下文档可能为您实现容灾高可用架构提供帮助：

- 版本控制概述
- 跨地域复制概述

域名管理实践
设置跨域访问

最近更新时间: 2025-01-15 17:01:00

同源策略

同源策略限制从一个源加载的文档或脚本与来自另一个源的资源进行交互的方式，是用于隔离潜在恶意文件的关键安全机制。同协议、同域名（或IP）、以及同端口视为同一个域，一个域内的脚本仅仅具有本域内的权限，即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。这种安全限制称为同源策略。

同源的定义

两个页面的协议、域名和端口（若指定了端口）相同，则视为同源。如下表给出了相对 `http://imgcache.finance.cloud.tencent.com:80www.example.com/dir/page.html` 的同源检测示例：

URL	结果	原因
`http://imgcache.finance.cloud.tencent.com:80www.example.com/dir2/other.html`	成功	协议、域名、端口相同
`http://imgcache.finance.cloud.tencent.com:80www.example.com/dir/inner/another.html`	成功	协议、域名、端口相同
`http://imgcache.finance.cloud.tencent.com:80www.example.com/secure.html`	失败	协议不同（https）
`http://imgcache.finance.cloud.tencent.com:80www.example.com:81/dir/etc.html`	失败	端口不同（81）
`http://imgcache.finance.cloud.tencent.com:80news.example.com/dir/other.html`	失败	域名不同

跨域访问

跨域资源共享（Cross-Origin Resource Sharing，CORS）机制，我们简称为跨域访问，允许 Web 应用服务器进行跨域访问控制，从而使跨域数据传输得以安全进行。CORS 需要浏览器和服务器同时支持。目前，所有浏览器都支持该功能，IE 浏览器要求版本 IE10 或以上。 整个 CORS 通信过程，都是浏览器自动完成，不需要用户参与。对于开发者来说，CORS 通信与同源的 AJAX 通信没有差别，代码完全一样。浏览器一旦发现 AJAX 请求跨源，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，但用户不会有感觉。 因此，实现 CORS 通信的关键是服务器。只要服务器实现了 CORS 接口，就可以跨源通信。

CORS 主要使用场景

CORS 使用一定是在使用浏览器的情况下，因为控制访问权限的是浏览器而非服务器。因此使用其它客户端的时候无需关心任何跨域问题。 CORS 的主要应用是实现无论上传或者下载，在浏览器端使用 AJAX 直接访问 COS 的数据，而无需通过用户本身的应用服务器中转。对于同时使用 COS 和使用 AJAX 技术的网站，建议使用 CORS 来实现与 COS 的直接通信。

COS 对 CORS 的支持

COS 支持对 CORS 规则的配置，从而根据需求允许或者拒绝相应的跨域请求。该 CORS 规则配置属于存储桶级别。 CORS 请求的通过与否和 COS 的身份验证等是完全独立的，即 COS 的 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。 目前 COS 的所有 Object 相关接口都提供了 CORS 相关的验证，另外 Multipart 相关的接口目前也已经完全支持 CORS 验证。

> **说明：**
>
> 当运行在同一个浏览器上的分别来源于 `www.a.com` 和 `www.b.com` 的两个页面同时请求同一跨域资源时，若 `www.a.com` 的请求先到达服务器，服务器会将资源带上 Access-Control-Allow-Origin 的 Header，并返回给 `www.a.com` 的用户。这时 `www.b.com` 又发起了请求，浏览器会将 Cache 的上一次请求响应返回给用户，Header 的内容和 CORS 的要求不匹配，就会导致 `www.b.com` 请求失败。

CORS 设置示例

以下简单示例介绍使用 AJAX 从 COS 获取数据的配置步骤。示例中使用的存储桶（Bucket）权限设置为公有（Public），访问权限为私有的存储桶（Bucket）只需要在请求中附加签名，其余配置相同。 以下示例中使用的存储桶名为 corstest，存储桶访问权限为公有读私有写。

设置 CORS 之前

1. **确认文件可正常访问**

上传一个 test.txt 的文本文档到 corstest。test.txt 的访问地址为 http://imgcache.finance.cloud.tencent.com:80corstest-125xxxxxxx.cos.ap-beijing-1.myqcloud.com/test.txt 。

使用 curl 直接访问该文本文档，请将以下地址替换为您的文件地址：

```
curl http://imgcache.finance.cloud.tencent.com:80corstest-125xxxxxxx.cos.ap-beijing-1.myqcloud.com/test.txt
```

返回 test.txt 文件的内容：test。表示该文档可以正常访问。

2. **使用 AJAX 技术访问文件**

我们现在尝试使用 AJAX 技术来直接访问该 test.txt 文件。

i. 写一个简单的 HTML 文件，将以下代码复制到本地保存成 HTML 文件后使用浏览器打开。因为没有设置自定义的头，因此该请求不需要预检。

```
<!doctype html>
<html lang="en">
<head>
<meta charset="UTF-8">
</head>
<body>
<a href="javascript:test()">Test CORS</a>
<script>
function test() {
var url = 'http://imgcache.finance.cloud.tencent.com:80corstest-125xxxxxxx.cos.ap-beijing-1.myqcloud.com/test.txt';
var xhr = new XMLHttpRequest();
xhr.open('HEAD', url);
xhr.onload = function () {
var headers = xhr.getAllResponseHeaders().replace(/\r\n/g, '\n');
alert('request success, CORS allow.\n' +
'url: ' + url + '\n' +
'status: ' + xhr.status + '\n' +
'headers:\n' + headers);
};
xhr.onerror = function () {
alert('request error, maybe CORS error.');
};
xhr.send();
}
</script>
</body>
</html>
```

ii. 在浏览器中打开该 HTML 文件，单击【Test CORS】的按钮发送请求后，出现以下错误，错误提示：无权限访问，原因是没有找到 Access-Control-Allow-Origin 这个 Header。显然，这是因为服务器没有配置 CORS。

iii. 访问失败，再进入 Header 界面检查原因，可见浏览器发送了带 Origin 的 Request，因此是一个跨域请求。

> **说明：**
>
> 我们将网页搭建在服务器上，地址为 http://imgcache.finance.cloud.tencent.com:80127.0.0.1:8081 ，所以 Origin 是 http://imgcache.finance.cloud.tencent.com:80127.0.0.1:8081 。

设置 CORS

确定访问不成功的原因之后，可以通过设置存储桶相关的 CORS 来解决以上问题。COS 控制台可以进行 CORS 设置，本示例使用控制台来完成 CORS 的设置。若您的 CORS 设置不是特别复杂，也建议使用控制台来完成 CORS 设置。

1. 登录 COS 控制台，单击**存储桶列表**，进入相关的存储桶。

2. 在左侧导航树中，选择**安全管理 > 跨域访问 CORS 设置**，进入跨域访问CORS设置页面。

3. 单击**添加规则**，添加第一条规则，使用最宽松的配置如下：

> **注意：**
>
> CORS 设置是由一条一条规则组成的，会从第一条开始逐条匹配，以最早匹配上的规则为准。

验证结果

配置完成后，重新尝试访问 test.txt 文本文件。结果如下，可以正常访问请求。

故障排除及建议

若想要排除跨域带来的访问问题，可以将 CORS 设置为以上最宽松的配置，该配置允许所有的跨域请求。该配置下依然出错，则表明错误出现在其他部分而不是 CORS。

除了最宽松的配置之外，您还可以配置更精细的控制机制来实现针对性的控制。比如对于本示例可以使用如下最小的配置匹配成功：

因此对于大部分场景来说，推荐您根据自己的使用场景来使用最小的配置以保证安全性。

CORS 配置项说明

CORS 配置有以下几项：

来源 Origin

允许跨域请求的来源。

- 可以同时指定多个来源,每行只能填写一个。
- 配置支持 `*` ，表示全部域名都允许，不推荐。
- 支持单个具体域名，形如 `http://imgcache.finance.cloud.tencent.com:80www.abc.com` 。
- 支持二级泛域名，形如 `http://imgcache.finance.cloud.tencent.com:80*.abc.com` ,但是每行只能有一个 `*` 号。
- 注意不要遗漏协议名 http 或 https，若端口不是默认的 80，还需要带上端口。

操作 Methods

枚举允许的跨域请求方法（一个或者多个）。 例如：GET、PUT、POST、DELETE、HEAD。

Allow-Headers

允许的跨域请求 Header。

- 可以同时指定多个来源,每行只能填写一个。
- Header 容易遗漏，没有特殊需求的情况下，建议设置为 * ，表示允许所有。
- 大小写不敏感。
- 在 Access-Control-Request-Headers 中指定的每个 Header，都必须在 Allowed-Header 中有对应项。

Expose-Headers

暴露给浏览器的 Header 列表，即用户从应用程序中访问的响应头（例如一个 Javascript 的 XMLHttpRequest 对象）。

- 具体的配置需要根据应用的需求确定，默认推荐填写 Etag。
- 不允许使用通配符，大小写不敏感，每行只能填写一个。

超时 Max-Age

浏览器对特定资源的预取请求（OPTIONS请求）返回结果的缓存时间，单位为秒。没有特殊情况时可以设置稍大一点，如 60 秒。 该项是可选配置项。

托管静态网站

最近更新时间: 2025-01-15 17:01:00

简介

在此实践中，用户可以在对象存储（以下简称 COS）上托管静态网站，访客可以通过 COS 提供的静态网站域名或者绑定的自定义域名（例如 www.example.com ）访问托管的静态网站。无论您是想在 COS 上托管已有静态网站还是从零开始建站，此实践可帮助您在 COS 上托管静态网站。以下是具体步骤：

> **注意：**
>
> 开启静态网站配置后，您需要使用静态网站域名访问 COS 源站才能生效，如果使用 COS 默认域名访问则无静态网站效果。

事前准备

以下是实践过程中，可能会用到的相关服务：

- **域名注册（可选）**：托管静态网站前，您需要拥有一个域名，例如 www.example.com 。若您使用的是 COS 提供的静态网站域名，可不使用该服务。
- **对象存储 COS**：使用 COS 创建存储桶，上传的网页内容将存放到存储桶。
- **内容分发网络 CDN**：结合 CDN 和 DNS 解析服务，使得域名和网站内容绑定的同时，还可以为静态网站加速，降低访问延迟，提升浏览体验。
- **DNS 解析 DNSPod**：使用 DNS 解析，实现使用自定义域名访问静态网站的目的。

> **说明：**
>
> 本指南中的所有步骤都使用 www.example.com 作为示例域名。实际操作中请使用您的自有域名替换此域名。

步骤1：注册域名与备案（可选）

域名注册是在互联网上建立任何服务的基础。注册域名之后，还需进行备案，网站才能正常访问。请根据您的具体情况进行操作：

- 已注册域名并备案，可跳过本步骤，进行 步骤2。
- 已注册域名但未备案，请进行 **域名备案**。
- 未注册域名，请先 **注册域名**，再进行 **域名备案**。

> **说明：**
>
> 若您使用的是 COS 提供的静态网站域名，可跳过此步骤。

步骤2：创建存储桶并上传内容

在完成域名注册及备案后，您需要在 COS 控制台中执行以下任务，以创建和配置网站内容：

- **创建存储桶**
- **配置存储桶并上传内容**

1. 创建存储桶

请使用账号登录 COS 控制台，为您的网站创建相应的存储桶，存储桶用于存储数据，您可以将网站内容存储在一个存储桶中。

如您首次使用 COS，可以通过控制台的概览界面上的【创建存储桶】直接创建存储桶，或者单击左侧【存储桶列表】导航栏进行创建，具体操作请参见 创建存储桶 文档。

2. 配置存储桶并上传内容

1. 开启存储桶的**静态网站**设置，方法是：

   登录 COS 控制台 ，在左侧菜单栏中单击【存储桶列表】，找到刚才已创建的存储桶，单击右侧的【配置管理】。

   在左侧菜单栏中选择【基础配置】>【静态网站】，单击【编辑】，将当前状态设置为开启，索引文档为 index.html，其余暂不配置，然后单击【保存】。

2. 将您的网站内容上传到已创建好的存储桶。具体操作请参见 上传对象 文档。

在存储桶中存放的内容可以是文本文件、照片、视频，任何您想要托管的内容。如果您还未构建网站，则只需按此实践创建一个文件。

例如，您可使用以下 HTML 创建文件，并将其上传到存储桶。网站主页的文件名通常为 index.html。在后续步骤中，您将提供此文件作为网站的索引文档。

```
<!DOCTYPE html>
<html>
<head>
<title>Hello COS!</title>
<meta charset="utf-8">
</head>
<body>
<p>欢迎使用 COS 的静态网站功能。</p>
<p>这是首页！</p>
</body>
</html>
```

> **注意：**
>
> 开启静态网站功能后，当用户访问任何不带文件指向的一级目录时，COS 默认优先匹配对应存储桶目录下 index.html，其次为 index.htm，若无此文件，则返回404。

使用COS静态网站功能搭建前端单页应用

最近更新时间: 2025-01-15 17:01:00

什么是单页应用？

单页应用（single-page application，SPA）是一种网络应用程序或网站的模型，它通过动态重写当前页面与用户进行交互，而非传统的从服务器重新加载整个新页面。这种方法避免了页面之间切换打断用户体验，使应用程序更像一个桌面应用程序。在单页应用中，所有必要的代码（HTML、JavaScript 和 CSS）都通过单个页面的加载而检索，或者根据需要（通常是为响应用户操作）动态装载适当的资源并添加到页面。

目前在前端开发领域，常见的单页应用开发框架有 React、Vue、Angular 等。

本文将使用目前较为热门的2个框架，一步步教您使用**对象存储（Cloud Object Storage，COS）**提供的**静态网站**功能快速搭建一个线上可用的单页应用，并提供一些常见问题的解决方案。

准备工作

1. 安装 Node.js 环境。
2. 确保能正常登录 COS 控制台。
3. 创建一个存储桶（为了方便测试可将权限设置为**公有读私有写**）。

编写前端代码

> **注意：**
>
> 如果已经自行实现了代码，可直接跳至 开启存储桶静态网站配置 步骤查看。

使用 Vue 快速搭建一个单页应用

1. 执行如下命令，安装 vue-cli：

```
npm install -g @vue/cli
```

2. 执行如下命令，使用 vue-cli 快速生成一个 vue 项目，可参见 官方文档。

```
vue create vue-spa
```

3. 执行如下命令，在项目根目录下安装 vue-router：

```
npm install vue-router -S（Vue2.x）
```

或者

```
npm install vue-router@4 -S（Vue3.x）
```

4. 修改项目里的 main.js 和 App.vue 文件。

main.js 如下图：

App.vue 主要修改组件的 template，如下图：

> **说明：**
>
> 由于篇幅有限，这里仅节选部分关键代码，完整代码可 单击此处 下载。

5. 修改代码后，执行如下命令，进行本地预览。

```
npm run serve
```

6. 调试并预览检查无误后，执行如下命令，打包生产环境代码。

```
npm run build
```

此时，将会在项目根目录下生成 dist 目录。至此，Vue 的程序代码已经准备完毕。

使用 React 快速搭建一个单页应用

1. 执行如下命令，安装 create-react-app：

```
npm install -g create-react-app
```

2. 使用 create-react-app 快速生成一个 react 项目，可参考 官方文档。
3. 执行如下命令，在项目根目录下安装 react-router-dom：

```
npm install react-router-dom -S
```

4. 修改项目里的 App.js 文件。

> **说明：**
>
> 由于篇幅有限，这里仅节选部分关键代码，完整代码可 单击此处 下载。

5. 修改代码后，执行如下命令，进行本地预览。

```
npm run start
```

6. 调试并预览检查无误后，执行如下命令，打包生产环境代码。

```
npm run build
```

此时将会在项目根目录下生成 build 目录。至此，React 的程序代码已经准备完毕。

开启存储桶静态网站配置

1. 进入已创建存储桶的详情页面，并在左侧导航栏中，单击**基础配置** > **静态网站**。

   在静态网站管理页面进行配置。操作详情请参见 设置静态网站。

部署至 COS

1. 找到之前已经配置了静态网站的存储桶，进入文件列表页面。

2. 将编译目录（Vue 默认为 dist 目录，react 默认为 build 目录）下的所有文件上传至存储桶的根目录下。操作详情请参见 上传对象。

3. 访问存储桶的静态网站域名（如下图中的访问节点）。

即可看到已经部署完成的应用主页，这里以 Vue 应用举例。

4. 尝试切换路由（Home、Foo、Bar），并刷新页面，验证是否符合预期（即在路由下刷新不会出现404报错）。

常见问题

我不想使用静态网站的默认域名怎么办？可以使用我自己的域名吗？

除了上面使用的默认静态网站域名，COS 还支持设置自定义源站域名。配置成功后即可使用自己想要的域名来访问应用。

部署好应用之后，切换路由能成功渲染，但页面一刷新就出现404报错，是什么原因？

原因可能是配置静态网站的时候，缺失配置或错误配置了错误文档导致，请再次回顾本文开头提供的标准配置截图，可以看到错误文档和索引文档均配置为 index.html 。

由于单页应用的特性，我们需要保证在任何情况下都要成功访问到应用入口（一般为 index.html），这样才能触发后续路由的一系列内部逻辑。

切换路由之后，页面能正常显示，但 HTTP 状态码依然为404，怎样才能正常返回200？

这里原因是配置静态网站的时候，缺少了配置错误文档响应码导致，可参考开头的配置截图，将其配置为200即可解决。

配置了错误文档之后，访问错误的路径还需要展示404的功能，应该如何处理？

这里推荐在前端代码里实现404逻辑，在路由配置最底部设置一个底层的匹配规则，当前面所有规则都匹配失败的时候则渲染一个404组件，组件内容可根据自行需求来设计实现。具体可参考本文提供的代码 demo 里的路由配置的最后一个配置。

访问页面出现 403 Access Denied 报错是什么原因？

原因可能是存储桶的权限设置了私有读写，可以修改为公有读私有写解决。

数据直传
Web端直传实践

最近更新时间: 2025-01-15 17:01:00

本文档介绍如何不依赖 SDK，用简单的代码，在网页（Web 端）直传文件到 COS 的存储桶。

> 本文档内容基于 XML API 。

一、前期准备

1. 登录 COS 控制台 并创建存储桶，得到 Bucket（存储桶名称） 和 Region（地域名称）。
2. 登录 控制台密钥管理 获取您的项目 SecretId 和 SecretKey。
3. 在 COS 控制台，单击新建的存储桶名称，进入存储桶的文件列表页面。
4. 单击【基础配置】页签，配置 CORS 规则。

二、计算签名

签名计算放在前端会暴露 SecretKey，因此我们把签名计算过程放在后端实现，前端通过 AJAX 向后端获取签名结果，正式部署时请在后端加一层您的网站本身的权限检验。指引参考 PHP 和 Node.js 的签名示例，其他语言请参照对应的 XML SDK 文档。

三、前端上传

方案 A：使用 AJAX 上传

AJAX 上传需要浏览器支持基本的 HTML5 特性，当前方案使用的是 XML API 的 PutObject 接口，操作指引：

1. 按照 步骤一、前期准备 的步骤，准备好存储桶。
2. 创建 test.html 文件，修改下方代码的 Bucket 和 Region，复制到 test.html 文件。
3. 部署好后端的签名服务，并修改 test.html 里的签名服务地址。
4. 把 test.html 放在 Web 服务器下，然后在浏览器访问页面，测试文件上传功能。

```
<!doctype html>
<html lang="en">
<head>
<meta charset="UTF-8">
<title>Ajax Put 简单上传</title>
<style>
h1, h2 {
font-weight: normal;
}

#msg {
margin-top: 10px;
}
</style>
</head>
<body>

<h1>Ajax Put 上传</h1>
<div>最低兼容到 ie10，支持 onprogress</div>

<input id="fileSelector" type="file">
<input id="submitBtn" type="submit">

<div id="msg"></div>

<script>
(function () {
// 请求用到的参数
var Bucket = 'test-1250000000';
var Region = 'ap-guangzhou';
```

```
var protocol = location.protocol === 'https:' ? 'https:' : 'http:';
var prefix = protocol + '//' + Bucket + '.cos.' + Region + '.myqcloud.com/';

// 计算签名
var getAuthorization = function (options, callback) {
var method = (options.Method || 'get').toLowerCase();
var key = options.Key || '';
var pathname = key.indexOf('/') === 0 ? key : '/' + key;

var url = '../server/auth.php';
var xhr = new XMLHttpRequest();
var data = {
method: method,
pathname: pathname,
};
xhr.open('POST', url, true);
xhr.setRequestHeader('content-type', 'application/json');
xhr.onload = function (e) {
if (e.target.responseText === 'action deny') {
alert('action deny');
} else {
callback(e.target.responseText);
}
};
xhr.send(JSON.stringify(data));
};

// 上传文件
var uploadFile = function (file, callback) {
var Key = 'dir/' + file.name; // 这里指定上传目录和文件名
getAuthorization({Method: 'PUT', Key: Key}, function (auth) {

var url = prefix + Key;
var xhr = new XMLHttpRequest();
xhr.open('PUT', url, true);
xhr.setRequestHeader('Authorization', auth);
xhr.onload = function () {
if (xhr.status === 200 || xhr.status === 206) {
var ETag = xhr.getResponseHeader('etag');
callback(null, {url: url, ETag: ETag});
} else {
callback('文件 ' + Key + ' 上传失败，状态码：' + xhr.status);
}
};
xhr.onerror = function () {
callback('文件 ' + Key + ' 上传失败，请检查是否没配置 CORS 跨域规则');
};
xhr.send(file);
});
};

// 监听表单提交
document.getElementById('submitBtn').onclick = function (e) {
var file = document.getElementById('fileSelector').files[0];
if (!file) {
document.getElementById('msg').innerText = '未选择上传文件';
return;
}
file && uploadFile(file, function (err, data) {
console.log(err || data);
document.getElementById('msg').innerText = err ? err : ('上传成功，ETag=' + data.ETag);
});
};
})();
</script>

</body>
</html>
```

执行效果如下图：

方案 B：使用 Form 表单上传

Form 表单上传支持低版本的浏览器的上传（如 IE8），当前方案使用的是 XML API 的 PostObject 接口。操作指引：

1. 按照 步骤一、前期准备 的步骤，准备好存储桶。
2. 创建 test.html 文件，修改下方代码的 Bucket 和 Region，复制到 test.html 文件。
3. 部署好后端的签名服务，并修改 test.html 里的签名服务地址。
4. 在 test.html 同一个目录下创建一个空的 empty.html ，用于上传成功时跳转回来。
5. 把 test.html 和 empty.html 放在 Web 服务器下，然后在浏览器访问页面，测试文件上传功能。

```
<!doctype html>
<html lang="en">
<head>
<meta charset="UTF-8">
<title>Form 表单简单上传</title>
<style>h1, h2 {font-weight: normal;}#msg {margin-top:10px;}</style>
</head>
<body>

<h1>Form 表单简单上传（兼容 IE8）</h1>
<div>最低兼容到 ie6 上传，不支持 onprogress</div>

<form id="form" target="submitTarget" action="" method="post" enctype="multipart/form-data" accept="*/*">
<input id="name" name="name" type="hidden" value="">
<input name="success_action_status" type="hidden" value="200">
<input id="success_action_redirect" name="success_action_redirect" type="hidden" value="">
<input id="key" name="key" type="hidden" value="">
<input id="Signature" name="Signature" type="hidden" value="">
<input id="x-cos-security-token" name="x-cos-security-token" type="hidden" value="">
<input id="fileSelector" name="file" type="file">
<input id="submitBtn" type="button" value="提交">
</form>
<iframe id="submitTarget" name="submitTarget" style="display:none;" frameborder="0"></iframe>

<div id="msg"></div>

<script>
(function () {

// 请求用到的参数
var Bucket = 'test-1250000000';
var Region = 'ap-guangzhou';
var protocol = location.protocol === 'https:' ? 'https:' : 'http:';
var prefix = protocol + '//' + Bucket + '.cos.' + Region + '.myqcloud.com/';
var form = document.getElementById('form');
form.action = prefix;

// 计算签名
var getAuthorization = function (options, callback) {
var method = (options.Method || 'get').toLowerCase();
var key = options.Key || '';
// var url = 'http://imgcache.finance.cloud.tencent.com:80127.0.0.1:3000/sts-post-object' +
var url = '../server/sts-post-object.php' +
'?method=' + method +
'&pathname=' + encodeURIComponent('/') +
'&key=' + encodeURIComponent(key);
var xhr = new XMLHttpRequest();
xhr.open('GET', url, true);
xhr.onreadystatechange = function (e) {
if (xhr.readyState === 4) {
if (xhr.status === 200) {
var data = JSON.parse(xhr.responseText);
callback(null, {
Authorization: data.authorization,
XCosSecurityToken: data.sessionToken,
});
```

```
} else {
callback('获取签名出错');
}
}
};
xhr.send();
};

// 监听上传完成
var Key;
var submitTarget = document.getElementById('submitTarget');
var showMessage = function (err, data) {
console.log(err || data);
document.getElementById('msg').innerText = err ? err : ('上传成功，ETag=' + data.ETag);
};
submitTarget.onload = function () {
var search;
try {
search = submitTarget.contentWindow.location.search.substr(1);
} catch (e) {
showMessage('文件 ' + Key + ' 上传失败');
}
if (search) {
var items = search.split('&');
var i, arr, data = {};
for (i = 0; i < items.length; i++) {
arr = items[i].split('=');
data[arr[0]] = decodeURIComponent(arr[1] || '');
}
showMessage(null, {url: prefix + Key, ETag: data.etag});
} else {
}
};

// 发起上传
document.getElementById('submitBtn').onclick = function (e) {
var filePath = document.getElementById('fileSelector').value;
if (!filePath) {
document.getElementById('msg').innerText = '未选择上传文件';
return;
}
Key = 'dir/' + filePath.match(/[\\\/]?([^\\\/]+)$/)[1]; // 这里指定上传目录和文件名
getAuthorization({Method: 'POST', Key: Key}, function (err, AuthData) {
// 在当前目录下放一个空的 empty.html 以便让接口上传完成跳转回来
document.getElementById('success_action_redirect').value = location.href.substr(0, location.href.lastIndexOf('/') + 1) + 'empty.html';
document.getElementById('key').value = Key;
document.getElementById('Signature').value = AuthData.Authorization;
document.getElementById('x-cos-security-token').value = AuthData.XCosSecurityToken;
form.submit();
});
};
})();
</script>

</body>
</html>
```

执行效果如下图：

相关文档

若您有更丰富的接口调用需求，请参考JavaScript SDK文档。

数据安全
防盗链实践

最近更新时间: 2025-01-15 17:01:00

功能介绍

云平台对象存储支持防盗链配置，建议您通过控制台的防盗链设置黑/白名单，来进行安全防护。

盗链案例

用户 A 在 COS 上传了图片资源 1.jpg ，得到可访问链接 http://imgcache.finance.cloud.tencent.com:80test-1250000000.cosgz.myqcloud.com/1.jpg ，并且在他自己的网页 http://imgcache.finance.cloud.tencent.com:80a.com/a.html 嵌入了该图片，能正常访问。 用户 B 在 a.com 上看到了图片，也在他自己的网页 http://imgcache.finance.cloud.tencent.com:80b.com/b.html 嵌入了 1.jpg 的链接，B 的网页也能正常显示图片。 以上案例中，A 的图片资源 1.jpg 就被 B 盗链了。此时 A 在不知情的情况下，COS 上的资源持续被 B 网页正常使用，A 负担了额外的流量费用，造成了费用损失。

防盗链判断原理

防盗链是通过请求 Header 里的 Referer 来判断的：

- Referer 是 Header 的一部分，当浏览器向 Web 服务器发送请求的时候，一般会带上 Referer，告诉服务器该请求是从哪个页面链接过来的，服务器就可以禁止或允许某些来源的网站访问资源。
- 如果直接在浏览器直接打开文件链接 http://imgcache.finance.cloud.tencent.com:80test-1250000000.cosgz.myqcloud.com/1.jpg ，请求 Header 里不会带有 Referer。

例如，下图是在 http://imgcache.finance.cloud.tencent.com:80127.0.0.1/test/test.html 嵌入了 1.jpg ，访问 test.html 时就带有 Referer 指向访问来源：

控制台设置说明

设置步骤

1. 登录对象存储控制台，在左侧导航栏中选择**存储桶列表**，进入存储桶列表页面。

2. 选择需要设置防盗链的存储桶名称，进入存储桶的文件列表页面。

3. 在左侧导航树中，选择**安全管理 > 防盗链设置**，进入防盗链设置页面。

4. 单击**编辑**，进入可编辑状态。

5. 确认当前状态为开启，选择名单类型（黑名单或白名单），设置好相应域名，设置完成单击【保存】即可。

   用户设置防盗链状态为开启后，必须填入相应的域名。

设置规则说明

- 名单类型黑、白名单二选一：
  - 黑名单：限制名单内的域名访问存储桶的默认访问地址，若名单内的域名访问存储桶的默认访问地址，则返回 403。

  - 白名单：限制名单外的域名访问存储桶的默认访问地址，若名单外的域名访问存储桶的默认访问地址，则返回 403。
- 配置规则示例：
  - 支持带端口的域名和 IP，如 test.com:8080 、 10.10.10.10:8080 等地址。
  - 配置 test.com ，可命中如 test.com/123 、 test.com.cn 等以 test.com 为前缀的地址。
  - 配置 test.com ，可命中如 http://imgcache.finance.cloud.tencent.com:80test.com 和 http://imgcache.finance.cloud.tencent.com:80test.com 为前缀的地址。
  - 配置 test.com ，可命中它的带端口域名 test.com:8080 。
  - 配置 test.com:8080 ，不会命中域名 test.com 。
  - 配置 *.test.com ，可限制它的二级、三级域名 test.com 、 b.test.com 、 a.b.test.com 。
- 设置域名支持最多十条域名且为前缀匹配；支持域名、IP 和通配符 * 等形式的地址；一个地址占一行，多个地址请换行。

设置示例

我们使用上文中的盗链案例来举例，介绍用户 A 如何通过防盗链设置防止用户 B 盗链图片：

1. 用户 A 给存储桶 test 设置了防盗链规则，有两种方式可以防止 b.com 盗链，您可以根据您的实际情况选择：
   - 开启方式一：配置黑名单模式，域名设置填入 *.b.com 并保存生效。
   - 开启方式二：配置白名单模式，域名设置填入 *.a.com 并保存生效。

2. 开启了防盗链配置之后：
   - 访问 http://imgcache.finance.cloud.tencent.com:80a.com/a.html 图片显示正常。
   - 访问 http://imgcache.finance.cloud.tencent.com:80b.com/b.html 图片无法显示，表现如下图。

快速搭建移动应用传输服务

最近更新时间: 2025-01-15 17:01:00

背景

移动互联网时代，App 作为移动互联网服务的基础设施，往往需要上传和下载大量的数据，数据的安全性和可靠性尤为重要。现在开发者可以将数据存储相关的问题交给 云平台 COS 服务，而只需要关心自己应用的业务逻辑即可，可减少很多工作量，提升开发效率。

本文主要介绍如何快速搭建一个基于 COS 的应用传输服务，在云平台 COS 上实现应用数据的上传下载，在您的服务器上只需要部署您自己的业务、生成和管理临时密钥。

架构

整体架构图如下所示：

其中：

- 应用 APP：即用户手机上的 App。
- COS：云平台对象存储，负责存储 App 上传的数据。
- CAM：云平台访问管理，用于生成 COS 的临时密钥。
- 应用服务器：用户自己的后台服务器，这里用于向 CAM 请求临时密钥，并返回给应用 App。

步骤

1. 应用向用户的应用服务器发送一个获取临时密钥的请求。 用户 App 在向 COS 发送请求时，必须通过密钥对请求进行签名。直接将用户的永久密钥放在终端上会存在很大的泄密风险，因此必须将您的永久密钥放在您的应用服务器上，然后由应用服务器向云平台 CAM 申请临时密钥，并最终返回给终端对请求进行签名。
2. 应用服务器收到请求后，向 CAM 申请临时密钥。 我们认为用户的应用服务器是可信任的，可以直接配置用户的永久密钥用于向 CAM 申请临时密钥。同时，在服务端您可以配置临时密钥的有效期以及策略（Policy）。
3. CAM 返回临时密钥给应用服务器。
4. 应用服务器返回临时密钥给应用 App。 我们对数据的返回格式没有做具体的要求，这里建议直接返回 CAM 的 JSON 数据即可。
5. 应用 App 向 COS 发送请求（如上传文件等）。 应用 App 获取到临时密钥后，可以通过我们的 SDK 很容易地对请求进行签名，来使用我们的 COS 服务。

示例

搭建一个应用服务器

本示例采用 Python 编写，您可以 点击下载。

1. **配置参数** 下载并解压好的文件夹根目录下有 `config.py` 目录，可以来配置以下参数：

```
class Config(object):

def __init__(self):
pass

# 用户昵称，非必选
NAME = "WANG"
# 策略
POLICY = COMMON_POLICY
# 临时证书有效期
DURATION_SECOND = 1800
# 用户的secret id
SECRET_ID = "您的 secret id"
# 用户的secret key
SECRET_KEY = "您的 secret key"
```

示例中提供的是一个拥有账户下所有资源读写权限的策略。

2. **解析返回的 JSON 数据**

```
{
"code":0,"message":"","codeDesc":"Success",
"data":
{
"credentials":
{
"sessionToken":"42f8151428b3960b1226f421b8f271c6242ad02c3",
"tmpSecretId":"AKIDxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx",
"tmpSecretKey":"ZfV5PVLvFLCvPefPt76qKYXIo56tSmrg"
},
"expiredTime":1508400619
}
}
```

应用服务器搭建临时密钥服务

下面以云平台 CVM Ubuntu 实例为例，详细介绍如何搭建一个简单的临时密钥服务。

1. **登录云平台 CVM**

购买 CVM 实例后，可直接用浏览器在控制台登录，或者使用终端直接 SSH 登录。

2. **Ubuntu 环境下配置 Python 开发环境**

```
sudo apt-get update
sudo apt-get install python-dev python-pip python-virtualenv
```

3. **下载源码并解压**

```
wget http://imgcache.finance.cloud.tencent.com:80rickenwang-1253653367.cosgz.myqcloud.com/resources/signer-release.zip
unzip signer-release.zip
```

> 下载后直接解压，解压后需要修改根目录下的 `secret_key` 和 `secret_id` 配置信息，具体值可以在云 API 密钥控制台上查询。

4. **激活 virtualenv 环境并进行配置**

i. 激活 virtualenv 环境：

```
cd 到flask项目目录
virtualenv venv
source venv/bin/activate
```

ii. 配置环境：

```
pip install gunicorn
pip install flask
pip install flask-script
```

5. **使用 supervisor 管理服务**

i. 安装 supervisor 并打开配置文件

```
pip install supervisor
echo_supervisord_conf > supervisor.conf
vi supervisor.conf
```

ii. 在配置文件 `supervisor.conf` 的底部添加以下内容：

```
[program:manage]
command=./venv/bin/gunicorn -w4 -b0.0.0.0:5000 manage:app
directory=./
startsecs=0
stopwaitsecs=0
autostart=false
```

```
autorestart=false
stdout_logfile=./log/gunicorn.log
stderr_logfile=./log/gunicorn.err
```

6. **启动服务并测试**

使用如下命令启动服务：

```
supervisord -c supervisor.conf
supervisorctl -c supervisor.conf start manage
```

应用服务器启动后，直接用浏览器访问 `http://imgcache.finance.cloud.tencent.com:80ip:5000/sign` 地址，若返回临时密钥 JSON 字符串，则说明应用服务器临时密钥服务正常运行。

> **注意：**
>
> 浏览器访问时 IP 请修改为您服务器的 IP 或者域名，默认的端口是 5000，您可自行在 `supervisor.conf` 文件中修改。

应用 App 接入应用服务器临时密钥服务

若您对云平台 COS 服务已有一些基本的了解，那么您可以很容易地使用我们 COS XML 终端 SDK，并将您的应用服务器的临时密钥服务接入进来，下面以 Android SDK 为例说明，总共分为三步：

> **注意：**
>
> 使用我们的 SDK 前，您需要了解一些基本的名词术语，如 APPID、Bucket、SecretId、SecretKey 等。

1. 定义 `MySessionCredentialProvider` 类继承抽象类 `BasicLifecycleCredentialProvider` ，并实现其 fetchNewCredentials() 方法。

```
public class MySessionCredentialProvider extends BasicLifecycleCredentialProvider {

@Override
protected QCloudLifecycleCredentials fetchNewCredentials() throws QCloudClientException {

SessionQCloudCredentials credentials;
String tempSecretId = "";
String tempSecretKey = "";
String sessionToken = "";
long expireTime;

// 1、访问应用服务器获取临时密钥 API，获取临时密钥 JSON 字符串
// ...


// 2、解析 JSON 字符串，初始化 tempSecretId, tempSecretKey, sessionToken, expiredTime 四个参数
// ...


// 3、返回 SessionQCloudCredentials
return new SessionQCloudCredential(tempSecretId, tempSecreKey, sessionToken, expireTime);

}

}
```

2. 用 `MySessionCredentialProvider` 初始化 `CosXmlService` 。

```
// 您的账户信息
String appid = "对象存储 的服务APPID";
String region = "存储桶 所在的地域";

// 创建 CosXmlServiceConfig 对象
CosXmlServiceConfig serviceConfig = new CosXmlServiceConfig.Builder()
.setAppidAndRegion(appid, region)
.build();
```

```
// 初始化 CosXmlService
CosXmlService cosXmlService = new CosXmlService(getApplicationContext(), cosXmlServiceConfig,
new MySessionCredentialProvider());
```

3. 通过 `CosXmlService` 对象的各种接口访问 COS 服务，下面以简单上传为例：

```
String bucket = "存储桶名称";
String cosPath = "远端路径，即存储到 COS 上的绝对路径"; //格式如 cosPath = "/test.txt";
String srcPath = "本地文件的绝对路径"; // 如 srcPath = Environment.getExternalStorageDirectory().getPath() + "/test.txt";
long signDuration = 600; //签名的有效期，单位为秒

PutObjectRequest putObjectRequest = new PutObjectRequest(bucket, cosPath, srcPath);

putObjectRequest.setSign(signDuration,null,null);

/*
* 设置进度显示
* 实现 QCloudProgressListener.onProgress(long progress, long max) 方法，
* progress 已上传的大小， max 表示文件的总大小
*/
putObjectRequest.setProgressListener(new QCloudProgressListener() {

@Override
public void onProgress(long progress, long max) {
float result = (float) (progress * 100.0/max);
Log.w("TEST","progress =" + (long)result + "%");
}
});

//使用同步方法上传
try {
PutObjectResult putObjectResult = cosXmlService.putObject(putObjectRequest);
Log.w("TEST","success: " + putObjectResult.accessUrl);

} catch (CosXmlClientException e) {
//抛出异常
Log.w("TEST","CosXmlClientException =" + e.toString());
} catch (CosXmlServiceException e) {
//抛出异常
Log.w("TEST","CosXmlServiceException =" + e.toString());
}
```

使用AWS S3 SDK访问COS

最近更新时间: 2022-10-25 15:39:59

简介

COS 提供了 AWS S3 兼容的 API，因此当您的数据从 S3 迁移到 COS 之后，只需要进行简单的配置修改，即可让您的客户端应用轻松兼容 COS 服务。本文主要介绍不同开发平台的 S3 SDK 的适配步骤。在完成添加适配步骤后，您就可以使用 S3 SDK 的接口来访问 COS 上的文件了。 AWS S3采用S3 V4版本签名方法进行签名，详见 http://imgcache.finance.cloud.tencent.com:80docs.aws.amazon.com/zh_cn/AmazonS3/latest/API/sig-v4-authenticating-requests.html?from_wecom=1

准备工作

1. 您已注册账号，并且从 访问管理控制台 上获取了密钥 SecretID 与 SecretKey。

2. 您已有一个集成了 S3 SDK，并能正常运行的客户端应用。

Android

下面以 AWS Android SDK 2.14.2 版本为例，介绍如何适配以便访问 COS 服务。对于终端访问 COS，将永久密钥放到客户端代码中有极大的泄露风险，我们建议您接入 STS 服务获取临时密钥，详情请参见 临时密钥生成及使用指引。

初始化

初始化实例时，您需要设置临时密钥提供者和 Endpoint，以存储桶所在地域是 `ap-guangzhou` 为例：

```
AmazonS3Client s3 = new AmazonS3Client(new AWSCredentialsProvider() {
@Override
public AWSCredentials getCredentials() {
// 这里后台请求 STS 得到临时密钥信息
return new BasicSessionCredentials(
"<TempSecretID>", "<TempSecretKey>", "<STSSessionToken>"
);
}

@Override
public void refresh() {
//
}
});

s3.setEndpoint("cos.ap-guangzhou.myqcloud.com");
```

iOS

以 AWS iOS SDK 2.10.2 版本为例，介绍如何适配以便访问 COS 服务。对于终端访问 COS，将永久密钥放到客户端代码中有极大的泄露风险，我们建议您接入 STS 服务获取临时密钥，详情请参见 临时密钥生成及使用指引。

1. 实现 AWSCredentialsProvider 协议

```
-(AWSTask<AWSCredentials *> *)credentials{
// 这里后台请求 STS 得到临时密钥信息
AWSCredentials *credential = [[AWSCredentials alloc]initWithAccessKey:@"<TempSecretID>" secretKey:@"<TempSecretKey>" sessionKey:@"<STSSessionToken>" expiration:[NSDate dateWithTimeIntervalSince1970:1565770577]];

return [AWSTask taskWithResult:credential];

}

- (void)invalidateCachedTemporaryCredentials{

}
```

2. 提供临时密钥提供者和 Endpoint

以存储桶所在地域是 `ap-guangzhou` 为例：

```
NSURL* bucketURL = [NSURL URLWithString:@"http://imgcache.finance.cloud.tencent.com:80cos.ap-guangzhou.myqcloud.com"];

AWSEndpoint* endpoint = [[AWSEndpoint alloc] initWithRegion:AWSRegionUnknown service:AWSServiceS3 URL:bucketURL];
AWSServiceConfiguration* configuration = [[AWSServiceConfiguration alloc]
initWithRegion:AWSRegionUSEast2 endpoint:endpoint
credentialsProvider:[MyCredentialProvider new]]; // MyCredentialProvider 实现了 AWSCredentialsProvider 协议

[[AWSServiceManager defaultServiceManager] setDefaultServiceConfiguration:configuration];
```

Node.js

下面以 AWS JS SDK 2.509.0 版本为例，介绍如何适配以便访问 COS 服务。

初始化

初始化实例时设置腾讯云金融专区密钥和 Endpoint，以存储桶所在地域是 `ap-guangzhou` 为例，代码示例如下：

```
var AWS = require('aws-sdk');

AWS.config.update({
accessKeyId: "COS_SECRETID",
secretAccessKey: "COS_SECRETKEY",
region: "ap-guangzhou",
endpoint: 'http://imgcache.finance.cloud.tencent.com:80cos.ap-guangzhou.myqcloud.com',
});

s3 = new AWS.S3({apiVersion: '2006-03-01'});
```

Java

下面以 AWS Java SDK 1.11.609 版本为例，介绍如何适配以便访问 COS 服务。

1. 修改 AWS 配置和证书文件

> 说明：
>
> 下面以 Linux 为例，修改 AWS 配置和证书文件。

AWS SDK 的默认配置文件通常在用户目录下，可以参考 配置和证书文件。

- 在配置文件（文件位置是 `~/.aws/config` ）中添加以下配置信息：

  ```
  [default]
  s3 =
  addressing_style = virtual
  ```

- 在证书文件（文件位置是 `~/.aws/credentials` ）中配置腾讯云金融专区的密钥：

  ```
  [default]
  aws_access_key_id = [COS_SECRETID]
  aws_secret_access_key = [COS_SECRETKEY]
  ```

2. 代码中设置 Endpoint

以存储桶所在地域是 `ap-guangzhou` 为例，代码示例如下：

```
AmazonS3 s3Client = AmazonS3ClientBuilder.standard()
.withEndpointConfiguration(new AwsClientBuilder.EndpointConfiguration(
"http://imgcache.finance.cloud.tencent.com:80cos.ap-guangzhou.myqcloud.com",
```

```
"ap-guangzhou"))
.build();
```

Python

下面以 AWS Python SDK 1.9.205 版本为例，介绍如何适配以便访问 COS 服务。

1. 修改 AWS 配置和证书文件

> 说明：
>
> 下面以 Linux 为例，修改 AWS 配置和证书文件。

AWS SDK 的默认配置文件通常在用户目录下，可以参考 配置和证书文件。

- 在配置文件（文件位置是 `~/.aws/config`）中添加以下配置：

```
[default]
s3 =
signature_version = s3
addressing_style = virtuall
```

- 在证书文件（文件位置是 `~/.aws/credentials`）中配置密钥：

```
[default]
aws_access_key_id = [COS_SECRETID]
aws_secret_access_key = [COS_SECRETKEY]
```

2. 代码中设置 Endpoint

以存储桶所在地域是 `ap-guangzhou` 为例：

```
client = boto3.client('s3', endpoint_url='"http://imgcache.finance.cloud.tencent.com:80cos.ap-guangzhou.myqcloud.com"')
```

PHP

下面以 AWS PHP SDK 3.109.3 版本为例，介绍如何适配以便访问 COS 服务。

1. 修改 AWS 配置和证书文件

> 说明：
>
> 下面以 Linux 为例，修改 AWS 配置和证书文件。

AWS SDK 的默认配置文件通常在用户目录下，可以参考 配置和证书文件。

- 在配置文件（文件位置是 `~/.aws/config`）中添加以下配置：

```
[default]
s3 =
addressing_style = virtual
```

- 在证书文件（文件位置是 `~/.aws/credentials`）中配置腾讯云金融专区的密钥：

```
[default]
aws_access_key_id = [COS_SECRETID]
aws_secret_access_key = [COS_SECRETKEY]
```

2. 代码中设置 Endpoint

以存储桶所在地域是 ap-guangzhou 为例：

```
$S3Client = new S3Client([
'region' => 'ap-guangzhou',
'version' => '2006-03-01',
'endpoint' => 'http://imgcache.finance.cloud.tencent.com:80cos.ap-guangzhou.myqcloud.com'
]);
```

.NET

下面以 AWS .NET SDK 3.3.104.12 版本为例，介绍如何适配以便访问 COS 服务。

初始化

初始化实例时设置腾讯云金融专区密钥和 Endpoint，以存储桶所在地域是 ap-guangzhou 为例：

```
string sAccessKeyId = "COS_SECRETID";
string sAccessKeySecret = "COS_SECRETKEY";
string region = "ap-guangzhou";

var config = new AmazonS3Config() { ServiceURL = "http://imgcache.finance.cloud.tencent.com:80cos." + region + ".myqcloud.com" };
var client = new AmazonS3Client(sAccessKeyId, sAccessKeySecret, config);
```

Go

下面以 AWS Go SDK 1.21.9 版本为例，介绍如何适配以便访问 COS 服务。

1. 根据密钥创建 session

以存储桶所在地域是 ap-guangzhou 为例：

```
func newSession() (*session.Session, error) {
creds := credentials.NewStaticCredentials("COS_SECRETID", "COS_SECRETKEY", "")
region := "ap-guangzhou"
endpoint := "http://imgcache.finance.cloud.tencent.com:80cos.ap-guangzhou.myqcloud.com"
config := &aws.Config{
Region: aws.String(region),
Endpoint: &endpoint,
S3ForcePathStyle: aws.Bool(true),
Credentials: creds,
// DisableSSL: &disableSSL,
}
return session.NewSession(config)
}
```

2. 根据 session 创建 server 发起请求

```
sess, _ := newSession()
service := s3.New(sess)

// 以上传文件为例
fp, _ := os.Open("yourLocalFilePath")
defer fp.Close()

ctx, cancel := context.WithTimeout(context.Background(), time.Duration(30)*time.Second)
defer cancel()

service.PutObjectWithContext(ctx, &s3.PutObjectInput{
Bucket: aws.String("examplebucket-1250000000"),
Key: aws.String("exampleobject"),
Body: fp,
})
```

C++

下面以 AWS C++ SDK 1.7.68 版本为例，介绍如何适配以便访问 COS 服务。

1. 修改 AWS 配置和证书文件

> 说明：
>
> 下面以 Linux 为例，修改 AWS 配置和证书文件。

AWS SDK 的默认配置文件通常在用户目录下，可以参考 配置和证书文件。

- 在配置文件（文件位置是 `~/.aws/config` ）中添加以下配置：

```
[default]
s3 =
addressing_style = virtual
```

- 在证书文件（文件位置是 `~/.aws/credentials` ）中配置腾讯云金融专区的密钥：

```
[default]
aws_access_key_id = [COS_SECRETID]
aws_secret_access_key = [COS_SECRETKEY]
```

2. 代码中设置 Endpoint

以存储桶所在地域是 `ap-guangzhou` 为例，代码示例如下：

```
Aws::Client::ClientConfiguration awsCC;
awsCC.scheme = Aws::Http::Scheme::HTTP;
awsCC.region = "ap-guangzhou";
awsCC.endpointOverride = "cos.ap-guangzhou.myqcloud.com";
Aws::S3::S3Client s3_client(awsCC);
```

工具指南

工具概览

最近更新时间: 2025-01-15 17:01:00

对象存储提供以下开发者工具，供用户使用：

工具	功能说明
COSCMD 工具	本工具支持用户使用简单的命令行指令实现对对象的批量上传、下载、删除等操作。
COS Browser 工具	本工具支持用户通过可视化界面，方便地进行数据的上传、下载等操作。
COS Migration 工具	本工具支持从 AWS S3, 阿里云 OSS 和七牛等服务中迁移文件到 COS；同时也支持文件列表迁移，即从一系列给定的 URL 中迁移文件到 COS。
Hadoop 工具	本工具支持用户使用 Hadoop 处理存储在 COS 上的对象，如 MapReduce，Hive 等。
HDFS TO COS 工具	本工具支持将 HDFS 上的数据拷贝到对象存储 COS 上。

环境安装与配置
Java 安装与配置

最近更新时间: 2025-01-15 17:01:00

JDK 是 Java 软件开发工具包，本文以 JDK 1.7 和 1.8 版本为例，分别介绍了 Windows 和 Linux 系统下 JDK 的安装与环境配置过程。

Windows

1.下载 JDK

进入 Oracle 官方网站 下载合适的 JDK 版本，准备安装。

2. 安装

根据提示一步步安装，安装过程中可以自定义安装目录(默认安装到 C 盘)，例如我们选择的安装目录为： `D:\Program Files\Java\jdk1.8.0_31`

3. 配置

安装完成后，右键单击【计算机】> 单击【属性】>【高级系统设置】>【环境变量】>【系统变量】>【新建】，分别配置软件。 变量名(N)：**JAVA_HOME**变量值(V)：D:\Program Files\Java\jdk1.8.0_31 *// 请根据自己的实际安装路径配置*

变量名(N)：**CLASSPATH**
变量值(V)：.;%JAVA_HOME%\lib\dt.jar;%JAVA_HOME%\lib\tools.jar; *//注意变量值开头有"."*

变量名(N)：**Path**
变量值(V)：%JAVA_HOME%\bin;%JAVA_HOME%\jre\bin;

4. 测试

测试配置是否成功：【开始】（或快捷键：Win+R）>【运行】（输入 `cmd` ）>【确定】（或按 Enter 键），输入命令 `javac` 并回车。出现如下图所示信息，则说明环境变量配置成功。

Linux

由于使用 yum 或者 apt-get 命令 安装 openjdk 可能存在类库不全，从而导致用户在安装后运行相关工具时可能报错的问题，所以此处我们推荐采用手动解压安装的方式来安装 JDK。具体步骤如下：

1.下载 JDK

进入 Oracle 官方网站 下载合适的 JDK 版本，准备安装。

> **注意：**
>
> 这里需要下载 Linux 版本。这里以jdk-8u151-linux-x64.tar.gz为例，你下载的文件可能不是这个版本，这没关系，只要后缀(.tar.gz)一致即可。

2. 创建目录

在 `/usr/` 目录下创建 `java` 目录

```
mkdir /usr/java
cd /usr/java
```

把下载的文件 jdk-8u151-linux-x64.tar.gz 放在/usr/java/目录下。

3. 解压 JDK

```
tar -zxvf jdk-8u151-linux-x64.tar.gz
```

4. 设置环境变量

```
修改 /etc/profile
```

在 profile 文件中添加如下内容并保存：

```
set java environment
JAVA_HOME=/usr/java/jdk1.8.0_151
JRE_HOME=/usr/java/jdk1.8.0_151/jre
CLASS_PATH=.:$JAVA_HOME/lib/dt.jar:$JAVA_HOME/lib/tools.jar:$JRE_HOME/lib
PATH=$PATH:$JAVA_HOME/bin:$JRE_HOME/bin
export JAVA_HOME JRE_HOME CLASS_PATH PATH
```

> 注意：
>
> 其中 JAVA_HOME， JRE_HOME 请根据自己的实际安装路径及 JDK 版本配置。

让修改生效：

```
source /etc/profile
```

5. 测试

```
java -version
```

显示 java 版本信息，则说明 JDK 安装成功：

```
java version "1.8.0_151"
Java(TM) SE Runtime Environment (build 1.8.0_151-b12)
Java HotSpot(TM) 64-Bit Server VM (build 25.151-b12, mixed mode)
```

Python 安装与配置

最近更新时间: 2025-01-15 17:01:00

本文档以 Python 2.7 版本为例，详细介绍 Windows 和 Linux 系统下 Python 的安装与环境配置过程。

Windows

1. 下载

进入 Python 官网 选择合适的版本下载，本示例中我们选择下载 Python 2.7.13 版本。

2. 安装

下载好 Python 安装包后双击 Python 安装包，按照默认提示，一步步进行安装。

3. 环境变量配置

安装完成后，右键单击【计算机】>单击【属性】>【高级系统设置】>【环境变量】>【系统变量(S)】 找到 “Path”（没有就新建），并在 “变量值” 末尾添加 Python 的安装路径：`;C:\Python27`（请更改为您的安装路径），单击【确定】保存。

4. 测试配置是否成功

单击【开始】（或快捷键：Win+R）>【运行】（输入 `cmd` ）>【确定】（或者按 Enter 键），在弹出的窗口中输入命令 Python 并回车。出现以下信息，说明 Python 2.7 已经安装配置好：

Linux

1. 查看 Python 版本

Linux 的 yum 自带 Python，首先查看默认 Python 版本

```
python -V
```

若已经是 Python 2.7 及以上版本，则忽略以下步骤；否则（此处假设现有版本为 Python 2.6.6），输入以下命令：

```
yum groupinstall "Development tools"
```

2. 安装编译 Python 需要的组件

```
yum install zlib-devel bzip2-devel openssl-devel ncurses-devel sqlite-devel
```

3. 下载并解压 Python 2.7

```
wget http://imgcache.finance.cloud.tencent.com:80www.python.org/ftp/python/2.7.12/Python-2.7.12.tar.xz
tar xf Python-2.7.12.tar.xz
```

4. 编译与安装 Python

```
cd Python-2.7.12 //进入目录
./configure -prefix=/usr/local
make && make install //安装
make clean
make distclean
```

5. 将系统 Python 命令指向 Python 2.7

```
mv /usr/bin/python /usr/bin/python2.6.6
ln -s /usr/local/bin/python2.7 /usr/bin/python
```

6. 测试配置是否成功

```
python
```

出现以下信息，说明 Python 2.7 已经安装配置好：

> **注意：** 如果出现权限的问题，建议在命令前添加 sudo 尝试解决。

Hadoop 安装与测试

最近更新时间: 2025-01-15 17:01:00

Hadoop 工具依赖 Hadoop-2.7.2 及以上版本，实现了以云平台 COS 作为底层存储文件系统运行上层计算任务的功能。启动 Hadoop 集群主要有单机、伪分布式和完全分布式等三种模式，本文主要以 Hadoop-2.7.4 版本为例进行 Hadoop 完全分布式环境搭建及 wordcount 简单测试介绍。

准备环境

安装

1. 准备若干台机器。
2. 安装配置系统，可前往 CentOS 官网 下载安装。本文使用 CentOS 7.3.1611系统版本。
3. 安装 Java 环境，具体操作请参见 Java 安装与配置。
4. 安装 Hadoop 可用包：Apache Hadoop Releases Download。

网络配置

使用 `ifconfig -a` 查看各台机器的 IP，相互 ping 一下，看是否可以 ping 通，同时记录每台机器的 IP。

配置 CentOS

配置 hosts

```
vi /etc/hosts
```

编辑内容：

```
202.xxx.xxx.xxx master
202.xxx.xxx.xxx slave1
202.xxx.xxx.xxx slave2
202.xxx.xxx.xxx slave3
//IP 地址替换为真实 IP
```

关闭防火墙

```
systemctl status firewalld.service //检查防火墙状态
systemctl stop firewalld.service //关闭防火墙
systemctl disable firewalld.service //禁止开机启动防火墙
```

时间同步

```
yum install -y ntp //安装 ntp 服务
ntpdate cn.pool.ntp.org //同步网络时间
```

安装配置 JDK

上传 JDK 安装包（如jdk-8u144-linux-x64.tar.gz）到 `root` 根目录。

```
mkdir /usr/java
tar -zxvf jdk-8u144-linux-x64.tar.gz -C /usr/java/
rm -rf jdk-8u144-linux-x64.tar.gz
```

各个主机之间复制 JDK

```
scp -r /usr/java slave1:/usr
scp -r /usr/java slave2:/usr
scp -r /usr/java slave3:/usr
.....
```

配置各个主机 JDK 环境变量

```
vi /etc/profile
```

编辑内容：

```
export JAVA_HOME=/usr/java/jdk1.8.0_144
export PATH=$JAVA_HOME/bin:$PATH
export CLASSPATH=.:$JAVA_HOME/lib/dt.jar:$JAVA_HOME/lib/tools.jar
source/etc/profile //使配置文件生效
java -version //查看 java 版本
```

配置 SSH 无密钥访问

分别在各个主机上检查 SSH 服务状态：

```
systemctl status sshd.service //检查 SSH 服务状态
yum install openssh-server openssh-clients //安装 SSH 服务，如果已安装，则不用执行该步骤
systemctl start sshd.service //启动 SSH 服务，如果已安装，则不用执行该步骤
```

分别在各个主机上生成密钥：

```
ssh-keygen -t rsa //生成密钥
```

在 slave1 上：

```
cp ~/.ssh/id_rsa.pub~/.ssh/slave1.id_rsa.pub
scp~/.ssh/slave1.id_rsa.pub master:~/.ssh
```

在 slave2 上：

```
cp ~/.ssh/id_rsa.pub~/.ssh/slave2.id_rsa.pub
scp ~/.ssh/slave2.id_rsa.pubmaster:~/.ssh
```

依此类推...

在 master 上：

```
cd ~/.ssh
cat id_rsa.pub >> authorized_keys
cat slave1.id_rsa.pub >>authorized_keys
cat slave2.id_rsa.pub >>authorized_keys
scp authorized_keys slave1:~/.ssh
scp authorized_keys slave2:~/.ssh
scp authorized_keys slave3:~/.ssh
```

安装配置 Hadoop

安装 Hadoop

上传 hadoop 安装包（如hadoop-2.7.4.tar.gz）到 `root` 根目录。

```
tar -zxvf hadoop-2.7.4.tar.gz -C /usr
rm -rf hadoop-2.7.4.tar.gz
mkdir /usr/hadoop-2.7.4/tmp
mkdir /usr/hadoop-2.7.4/logs
mkdir /usr/hadoop-2.7.4/hdf
mkdir /usr/hadoop-2.7.4/hdf/data
mkdir /usr/hadoop-2.7.4/hdf/name
```

进入 `hadoop-2.7.4/etc/hadoop` 目录下，进行下一步操作。

配置 Hadoop

1. 修改 `hadoop-env.sh` 文件，增加

```
export JAVA_HOME=/usr/java/jdk1.8.0_144
```

若 SSH 端口不是默认的 22，可在 hadoop-env.sh 文件里修改：

```
export HADOOP_SSH_OPTS="-p 1234"
```

2. 修改 yarn-env.sh

```
export JAVA_HOME=/usr/java/jdk1.8.0_144
```

3. 修改 slaves

配置内容：

```
删除：
localhost
添加：
slave1
slave2
slave3
```

4. 修改 core-site.xml

```
<configuration>
<property>
<name>fs.default.name</name>
<value>hdfs://master:9000</value>
</property>
<property>
<name>hadoop.tmp.dir</name>
<value>file:/usr/hadoop-2.7.4/tmp</value>
</property>
</configuration>
```

5. 修改 hdfs-site.xml

```
<configuration>
<property>
<name>dfs.datanode.data.dir</name>
<value>/usr/hadoop-2.7.4/hdf/data</value>
<final>true</final>
</property>
<property>
<name>dfs.namenode.name.dir</name>
<value>/usr/hadoop-2.7.4/hdf/name</value>
<final>true</final>
</property>
</configuration>
```

6. 修改 mapred-site.xml

```
<configuration>
<property>
<name>mapreduce.framework.name</name>
<value>yarn</value>
</property>
<property>
<name>mapreduce.jobhistory.address</name>
<value>master:10020</value>
</property>
<property>
<name>mapreduce.jobhistory.webapp.address</name>
<value>master:19888</value>
</property>
</configuration>
```

7. 修改 yarn-site.xml

```
<configuration>
<property>
<name>yarn.nodemanager.aux-services.mapreduce.shuffle.class</name>
<value>org.apache.mapred.ShuffleHandler</value>
</property>
<property>
<name>yarn.nodemanager.aux-services</name>
<value>mapreduce_shuffle</value>
</property>
<property>
<name>yarn.resourcemanager.address</name>
<value>master:8032</value>
</property>
<property>
<name>yarn.resourcemanager.scheduler.address</name>
<value>master:8030</value>
</property>
<property>
<name>yarn.resourcemanager.resource-tracker.address</name>
<value>master:8031</value>
</property>
<property>
<name>yarn.resourcemanager.admin.address</name>
<value>master:8033</value>
</property>
<property>
<name>yarn.resourcemanager.webapp.address</name>
<value>master:8088</value>
</property>
</configuration>
```

8. 各个主机之间复制 Hadoop

```
scp -r /usr/ hadoop-2.7.4 slave1:/usr
scp -r /usr/ hadoop-2.7.4 slave2:/usr
scp -r /usr/ hadoop-2.7.4 slave3:/usr
```

9. 各个主机配置 Hadoop 环境变量

打开配置文件：

```
vi /etc/profile
```

编辑内容：

```
export HADOOP_HOME=/usr/hadoop-2.7.4
export PATH=$HADOOP_HOME/bin:$HADOOP_HOME/sbin:$PATH
export HADOOP_LOG_DIR=/usr/hadoop-2.7.4/logs
export YARN_LOG_DIR=$HADOOP_LOG_DIR
```

使配置文件生效：

```
source /etc/profile
```

启动 Hadoop

1. 格式化 namenode

```
cd /usr/hadoop-2.7.4/sbin
hdfs namenode -format
```

2. 启动

```
cd /usr/hadoop-2.7.4/sbin
start-all.sh
```

3. 检查进程

master 主机包含 ResourceManager、SecondaryNameNode、NameNode 等，则表示启动成功，例如：

```
2212 ResourceManager
2484 Jps
1917 NameNode
2078 SecondaryNameNode
```

各个 slave 主机包含 DataNode、NodeManager 等，则表示启用成功，例如：

```
17153 DataNode
17334 Jps
17241 NodeManager
```

运行 wordcount

由于 Hadoop 自带 wordcount 例程，所以可以直接调用。在启动 Hadoop 之后，我们可以通过以下命令来对 HDFS 中的文件进行操作：

```
hadoop fs -mkdir input
hadoop fs -put input.txt /input
hadoop jar share/hadoop/mapreduce/hadoop-mapreduce-examples-2.7.4.jar wordcount /input /output/
```

出现如上图结果就说明 Hadoop 安装已经成功了。

查看输出目录

```
hadoop fs -ls /output
```

查看输出结果

```
hadoop fs -cat /output/part-r-00000
```

> **注意：**
>
> 单机模式与伪分布式模式的操作方法的详细过程可以参考官网：Hadoop入门。

COSBrowser工具配置

最近更新时间: 2025-01-15 17:01:00

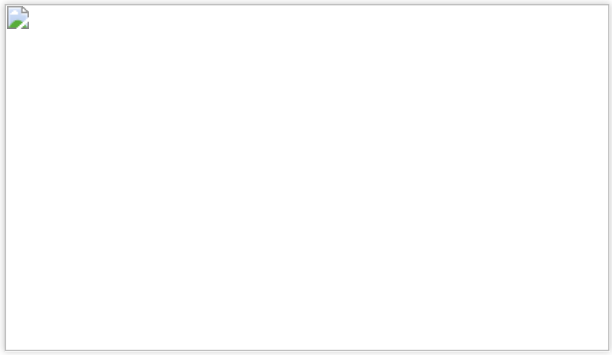
本文介绍Windows环境下COSBrowser工具登录时的配置说明。

密钥登录

基本配置

配置参数	配置说明
SecretID	1. 登录访问管理。 2. 在【云API密钥】页面获取用户SecretId信息。
SecretKey	在【云API密钥】页面获取用户SecretKey信息。
存储桶/访问路径	可选，如果不填写，登录时会进入该 Endpoint 指定的地域存储桶列表，若当前账号拥有所有Bucket的权限，则不用填；若当前账号只有某个存储桶或存储桶下某个目录的权限，则需要输入访问路径。 在存储桶列表页面复制存储桶名称。
地域	可选，选择存储桶对应的地域。 在存储桶列表页面可查看存储桶名称对应的地域。

高级设置

配置参数	配置说明
EndPoint/Service域名	1. 在存储桶列表页面，单击存储桶名称，进入存储桶详情，找到【访问域名】。 2. 复制存储桶名称之后的那段字符串。例如：存储桶域名是 test-1250000000.cos.chongqing.cos.example.com，那么 Endpoint 就是 cos.chongqing.cos.example.com 
Bucket域名模板	默认不需要填写，请求时会用 Endpoint 加上存储桶前缀作为请求域名，表示调用操作存储桶和对象的 API 时自定义请求域名。 可以填入域名模板，如"{Bucket}.cos.{Region}.cos.example.com"，即在调用 SDK API 时会使用参数中传入的 Bucket 和 Region 进行替换。
代理类型	根据实际情况选择。
使用后缀式/使用HTTPS/校验HTTPS证书	根据实际情况选择。

COSBrowser工具

最近更新时间: 2025-01-15 17:01:00

COSBrowser简介

COSBrowser 是对象存储 COS 推出的可视化界面工具，让您可以使用更简单的交互轻松实现对 COS 资源的查看、传输和管理。COSBrowser注重对资源的管理，用户可以通过 COSBrowser 批量的上传、下载数据。

> 注意：
>
> COSBrowser 会使用系统配置的代理来尝试网络连接，请确保您的代理配置正常或请停用无法连接互联网的代理配置。
>
> - Windows 用户可在操作系统的“Internet 选项”中查询。
> - macOS 用户可在“网络偏好设置”中查询。
> - Linux 用户可在系统设置 > 网络 > 网络代理中查询。

下载与安装

软件下载

支持平台	系统要求	下载地址
Windows	Windows 7 32/64位以上、Windows Server 2008 R2 64位以上	Windows
macOS	macOS 10.13以上	macOS
Linux	需带有图形界面并支持 AppImage 格式 注意：CentOS 启动客户端需在终端执行 `./cosbrowser.AppImage --no-sandbox`	Linux

软件安装

COSBrowser 安装包为可执行应用程序，用户下载完程序安装包后，双击安装包，然后按照系统提示即可进行安装。

登录软件

COSBrowser仅支持通过云 API 密钥进行登录使用，该密钥可在访问管理控制台的 API 密钥管理页面获取，成功登录后密钥将保存在**历史密钥**中，方便下次继续使用。

> 注意：
>
> COSBrowser 不支持使用项目密钥进行登录。

基本功能

1. 创建/删除存储桶

功能	说明	如何操作
创建存储桶	可以通过客户端直接创建存储桶	1. 在存储桶列表，单击左上角【添加桶】 2. 正确填写存储桶名称，并选择好地域、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，请确认存储桶中数据已清空	1. 在存储桶列表，单击对应存储桶右侧的【删除】 2. 确认存储桶中的数据已全部清空，然后单击【确定】进行删除

2. 查看存储桶详情

您可通过单击存储桶列表右侧的【详情】，查看存储桶详情。存储桶详细信息包含存储桶名称、访问权限、版本控制状态。

存储桶详情中，支持更改存储桶访问权限、开启或暂停版本控制功能。

3. 添加访问路径

若您使用无访问存储桶列表权限的子账号进行登录，可以通过**添加访问路径**的方式进行访问，COSBrowser 提供了两种添加访问路径的方式：

1. 在登录界面直接添加访问路径，并选择好对应的存储桶地域信息，登录完成后，即可管理资源。

2. 子账号登录后，在存储桶列表页左上角，单击【添加路径】，并输入指定的路径进入存储桶管理资源。

4. 上传文件/文件夹

上传功能	说明	如何操作
上传文件	COSBrowser 支持多种上传方式，支持单个或批量上传	上传文件的几种方式如下，在指定的存储桶或路径内： 1. 单击【上传】，选择文件后，直接上传文件。 2. 在文件列表空白处右键单击【上传文件】，进行上传。 3. 通过鼠标将文件拖拽至文件列表窗口进行上传。
上传文件夹及其文件	若存储桶或路径内存在同名文件或文件夹，则默认覆盖	上传文件夹的几种方式如下，在指定的存储桶或路径内： 1. 单击【上传】，选择文件夹，直接上传文件夹。 2. 在文件列表空白处右键选择【上传文件夹】，进行上传。 3. 通过鼠标将文件夹拖拽至文件列表窗口即可完成上传。
增量上传	增量上传指执行上传操作前，对上传文件与存储桶已有对象做比对， 若存在同名对象，则跳过该文件不执行上传操作	增量上传操作步骤如下，在指定的存储桶或路径内： 1. 使用上传文件夹的方式，单击下一步。 2. 【存储方式】选择【跳过】，单击【上传】后，即可完成增量上传。

5. 下载文件/文件夹

下载功能	说明	如何操作
下载文件	COSBrowser 支持多种下载方式，支持单个或批量下载文件	下载文件的几种方式如下： 1. 选中需要下载的文件，单击界面内的【下载】，即可下载该文件。 2. 选中文件，右键单击【下载】。 3. 通过鼠标将文件拖拽至本地的方式进行下载。
下载文件夹及其文件	若本地已存在同名文件或文件夹，则默认重命名	下载文件夹及其文件的几种方式如下： 1. 选中需要下载的文件夹，通过单击界面内的【下载】，即可下载。 2. 右键单击【下载】，直接下载该文件夹。 3. 通过鼠标将文件夹拖拽至本地的方式进行下载。

下载功能	说明	如何操作
增量下载	增量下载指执行下载操作前，将下载的对象与本地文件进行比对， 若存在同名对象，则跳过该对象不执行下载操作	增量下载操作步骤如下： 1. 选中想要下载的文件/文件夹，将鼠标移至【更多】出现下拉框。 2. 单击下拉框内的【高级下载】，在弹窗中选择【跳过】。 3. 然后单击【立即下载】即可完成不重名文件/文件夹的增量下载。

6. 删除文件/文件夹

选中想要删除的文件/文件夹，可通过单击界面上方【更多】中的【删除】或右键单击【删除】，完成文件/文件夹的删除，支持批量删除。

7. 文件同步

用户可以通过文件同步功能，将指定本地文件夹中的文件自动实时地上传至存储桶中。具体操作步骤如下：

1. 单击界面右上方的【同步】。
2. 在弹窗中指定本地文件夹和存储桶目录。
3. 单击【开始同步】，即可开启文件同步功能。
4. 可在同步日志中，查看文件的同步历史日志。

> **注意：**
>
> - 同步是指在上传文件时，系统会自动识别存储桶是否存在同样的文件，通过同步功能仅会上传存储桶中不存在的文件。
> - 目前仅支持将本地文件同步上传至存储桶，不支持逆向操作。
> - 文件同步功能支持设置手动同步和自动同步。

8. 复制粘贴文件

在指定的存储桶或路径内，选中想要复制的文件/文件夹，可通过单击界面上方【更多】中的【复制】或右键单击【复制】，完成文件/文件夹的复制，复制成功后可在**其他存储桶或路径**中进行粘贴，支持批量复制粘贴。

> **注意：**
>
> 对于已复制的文件或文件夹，若其粘贴的目标路径中包含同名文件，则默认覆盖。

9. 文件重命名

选中想要重命名的文件，右键选择【重命名】或单击文件右侧【更多操作】中的【重命名】，输入文件名并确定，即可完成文件的重命名。

> **说明：**
>
> 文件夹无法进行重命名操作。

10. 新建文件夹

在指定的存储桶或路径内，单击界面内的【新建文件夹】或右键单击【新建文件夹】，输入文件夹名并确认，完成文件夹的创建。

> **注意：**
>
> - 文件夹名称长度限制在255个字符内，可用数字、英文和可见字符的组合。
> - 文件夹名称不可包含 \ / : * ? " | < > 等特殊字符。
> - 不允许以 .. 作为文件夹名称。
> - 文件夹无法进行重命名操作，请谨慎命名。

11. 查看文件详情

可通过单击文件名或右键单击菜单中的【详情】来查看文件详情，文件详细信息包含文件名、文件大小、修改时间、访问权限、存储类型、ETag、Headers、对象地址、创建临时链接。

12. 生成文件链接

存在 COS 中的每个文件均可通过特定的链接来进行访问，若文件是私有读权限，则可通过请求临时签名的方式生成带有时效的临时访问链接。

COSBrowser 中有以下几种方式可生成文件链接：

- 列表视图下，单击文件右侧的分享图标，可一键生成链接并复制。若文件为公有读权限，则链接不带签名永久有效。若文件为私有读权限，则链接带签名，2小时内有效。
- 选择文件后，右键单击【复制链接】，可一键生成链接并复制。若文件为公有读权限，则链接不带签名永久有效。若文件为私有读权限，则链接带签名，2小时内有效。
- 在文件详情中，单击【创建临时链接】，可以设置临时链接的有效时间。

13. 文件预览

COSBrowser 支持预览媒体类文件，目前支持图片、视频、音频。可通过双击媒体格式文件或单击右键菜单中的【预览】或【播放】选项，即可打开文件预览界面。在文件预览或播放界面，您可以选择：

- **复制链接**：生成文件访问链接并复制。
- **下载**：将文件下载至本地，若本地存在同名文件，则默认覆盖。
- **手机查看**：在预览界面会生成文件访问二维码，通过手机扫码可直接在手机上查看此文件。

> **注意：**
>
> - 预览支持大多数图片格式，视频格式仅支持 mp4、webm，音频格式仅支持 mp3、wav。
> - 文件预览会产生下行流量，请酌情使用。

14. 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。COSBrowser 支持文件名前缀搜索和文件模糊搜索。

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. 查看多版本文件

当您的存储桶开启了版本控制后，可通过单击文件列表空白处右键菜单中的【查看多版本】选项，查看文件的历史版本。

软件设置

系统功能	说明	如何操作
设置网络代理	COSBrowser 默认会使用系统配置的代理来尝试网络连接，请确保您的代理配置正常或请停用无法连接互联网的代理配置	1. 选择【设置】>【代理】。 2. 设置网络代理来进行网络连接。
设置传输并发数	COSBrowser 支持批量上传和下载文件	1. 选择【设置】>【下载/上传】。 2. 设定批量传输的并发数，默认为5
设置传输分块数	COSBrowser 支持分块上传、下载文件，当传输的文件超过一定大小时，会默认使用分块的方式传输	1. 选择【设置】>【下载/上传】。 设定分块传输的并发数，默认为5。
设置传输失败重试数	COSBrowser 在文件传输的时候，会默认重试失败的任务	1. 选择【设置】>【下载/上传】。 2. 设定传输失败的重试次数，默认为5。

系统功能	说明	如何操作
设置上传二次校验	COSBrowser 支持在上传后进行二次校验，检查线上文件大小和状态是否正确	1. 选择【设置】>【上传】。 2. 勾选二次校验。
设置上传计算 md5	COSBrowser 支持在上传文件时计算文件的 md5，并以 x-cos-meta-md5 的自定义头部添加至文件的元数据中，下次请求该文件时，会返回该头部，可用于文件校验	1. 选择【设置】>【上传】。 2. 勾选计算 meta-md5 头。
查看本地日志	COSBrowser 会记录使用者的操作，以 cosbrowser.log 的日志形式保存在本地	1. 选择【设置】>【关于】。 2. 单击【本地日志】，系统将打开本地日志所在目录。

COSCMD工具

最近更新时间: 2025-01-15 17:01:00

功能说明

使用 COSCMD 工具，用户可通过简单的命令行指令实现对对象（Object）的批量上传、下载、删除等操作。

> **注意：**
>
> 使用该工具上传同名文件，会覆盖较旧的同名文件，不支持校对是否存在同名文件的功能。

使用环境

系统环境

支持 Windows、Linux 和 macOS 系统。

> **说明：**
>
> - 请保证本地字符格式为 UTF-8，否则操作中文件会出现异常。
> - 请确保本机时间已经与国际标准时间校准，如误差过大，将导致无法正常使用。

软件依赖

- Python 2.7/3.5/3.6。
- 最新版本的 pip。

安装及配置

- 环境安装与配置详细操作请参见官方文档 Python 安装和使用。
- pip 环境安装与配置详细操作请参见 官网 pip 安装说明。

下载与安装

通过 pip 命令直接安装

执行 `pip` 命令进行安装：

```
pip install coscmd
```

安装成功之后，用户可以通过 `coscmd -v` 或者 `coscmd --version` 命令查看当前的版本信息。

源码安装（不推荐）

源码地址：http://imgcache.finance.cloud.tencent.com:80github.com/tencentyun/coscmd.git。

在 shell 中执行以下三步即可完成安装：

```
# 通过 git 命令下载源码
git clone http://imgcache.finance.cloud.tencent.com:80github.com/tencentyun/coscmd.git

# 进入源代码的目录
cd coscmd

# 执行安装程序
python setup.py install
```

安装成功之后，用户可以通过 `coscmd -v` 或者 `coscmd --version` 命令查看当前的版本信息。

> **注意：**
>
> Python 版本为2.6时，pip 安装依赖库时容易失败，推荐使用该方法安装。

离线安装

需要一台有外网的机器，下载软件包，随后传到需要运行的机器上进行安装。

> 注意：
>
> 请确保两台机器的 Python 版本保持一致，否则会出现安装失败的情况。

```
# 在有外网的机器下运行如下命令
mkdir coscmd-packages
pip download coscmd -d coscmd-packages
tar -czvf coscmd-packages.tar.gz coscmd-packages

# 将安装包拷贝到没有外网的机器后运行如下命令
tar -xzvf coscmd-packages.tar.gz
pip install coscmd --no-index -f coscmd-packages
```

安装成功之后，用户可以通过 `coscmd -v` 或者 `coscmd --version` 命令查看当前的版本信息。

使用方法

查看 help

用户可通过 `-h` 或 `--help` 命令来查看工具的 help 信息。

```
coscmd -h //查看当前版本信息
```

help 信息如下所示：

```
usage: coscmd [-h] [-d] [-b BUCKET] [-r REGION] [-c CONFIG_PATH] [-l LOG_PATH]
[-v]

{info,restore,createbucket,signurl,listparts,mget,list,upload,deletebucket,abort,getbucketversioning,putbucketacl,getobjectacl,download,putobjectacl,copy,c
onfig,putbucketversioning,getbucketacl,delete}
...

an easy-to-use but powerful command-line tool. try 'coscmd -h' to get more
informations. try 'coscmd sub-command -h' to learn all command usage, likes
'coscmd upload -h'

positional arguments:
{info,restore,createbucket,signurl,listparts,mget,list,upload,deletebucket,abort,getbucketversioning,putbucketacl,getobjectacl,download,putobjectacl,copy,c
onfig,putbucketversioning,getbucketacl,delete}
config Config your information at first
upload Upload file or directory to COS
download Download file from COS to local
delete Delete file or files on COS
abort Aborts upload parts on COS
copy Copy file from COS to COS
list List files on COS
listparts List upload parts
info Get the information of file on COS
mget Download file from COS to local
restore Restore
signurl Get download url
createbucket Create bucket
deletebucket Delete bucket
putobjectacl Set object acl
getobjectacl Get object acl
putbucketacl Set bucket acl
getbucketacl Get bucket acl
putbucketversioning
Set the versioning state
getbucketversioning
Get the versioning state
probe Connection test

optional arguments:
-h, --help show this help message and exit
-d, --debug Debug mode
-b BUCKET, --bucket BUCKET
Specify bucket
```

```
-r REGION, --region REGION
Specify region
-c CONFIG_PATH, --config_path CONFIG_PATH
Specify config_path
-l LOG_PATH, --log_path LOG_PATH
Specify log_path
-v, --version show program's version number and exit
```

除此之外，用户还可以在每个命令后（不加参数）输入 `-h` 查看该命令的具体用法，例如：

```
coscmd upload -h //查看 upload 命令使用方法
```

配置参数

COSCMD 工具在使用前需要进行参数配置，用户可以通过如下命令来配置：

```
coscmd config [OPTION]...<FILE>...
[-h] --help
[-a] <SECRET_ID> // 该参数配置 Secret ID
[-s] <SECRET_KEY> // 该参数配置 Secret Key
[-t] <TOKEN>
[-b] <BucketName-APPID> // 该参数配置要访问的 Bucket
[-e] <ENDPOINT> | [-r] <REGION> // -r, -e 二选一，专有云只需通过 -e 配置访问节点
[-m] <MAX_THREAD>
[-p] <PART_SIZE>
[--do-not-use-ssl] // 该参数表示不使用 HTTPS 协议，即使用 HTTP 协议
[--anonymous]
```

参数配置说明如下：

选项	参数说明	是否必选	有效值
-a	密钥 ID	是	字符串
-s	密钥 Key	是	字符串
-b	指定的存储桶名称	是	字符串
-e	设置请求的 ENDPOINT，请参考 2.1.1 节的介绍	是	字符串
-m	多线程操作的最大线程数（默认为5，范围为1 - 30）	否	数字
--do-not-use-ssl	使用 HTTP 协议，而不使用 HTTPS	否	字符串

通常情况下，如您没有特殊的需求，可参照如下操作配置，将相关信息替换为您自己的，即可：

```
coscmd config -a AChT4ThiXAbpBDEFGhT4ThiXAbp**** -s WE54wreefvds3462refgwewe**** -b examplebucket-1250000000 -e cos.somewhere.example.com --do-not-use-ssl
```

> **说明：**
>
> 1. 生成的配置最终保存在 `~/.cos.conf` 文件（在 Windows 环境下，该文件是位于【我的文档】下的一个隐藏文件），该文件初始时不存在，可以通过 `coscmd config` 命令生成，用户也可以手动创建。

> 配置完成之后的 .cos.conf 文件内容示例如下所示：
>
> ```
> [common] secret_id = AKIDAtIM6U8rrVnQkgNXEEHyOR8TT7P***** secret_key = tRZNpv0C5coeAKXH1SXrqTkgGyM***** bucket = examplebucket-1255000000 endpoint = cos.chongqing.cospub.example.com max_thread = 5 part_size = 1 schema = http verify = md5 anonymous = False
> ```
>
> 1. 可以在配置文件中修改 `schema` 项来选择 `http/https` ，默认为 `https` 。
> 2. 可以在 `anonymous` 项中选择 `True/False` ，来使用匿名模式，即签名保持为空。

指定 Bucket 和 Endpoint 的命令

通过 `-b <BucketName-APPID>` 可以指定特定的 Bucket，通过 `-e <endpoint>` 可以指定特定的 endpoint。

存储桶的命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式。

```
#命令格式
coscmd -b <BucketName-APPID> -e <endpoint> <action> ...

#操作示例-创建bucket
coscmd -b examplebucket-1250000000 -e cos.somewhere.example.com createbucket

#操作示例-上传文件
coscmd -b examplebucket-1250000000 -e cos.somewhere.example.com upload exampleobject exampleobject
```

创建存储桶

可以配合 `-b <BucketName-APPID>` 指定 Bucket 和 `-e <endpoint>` 指定 Endpoint 使用。

```
#命令格式
coscmd -b <BucketName-APPID> createbucket

#操作示例
coscmd createbucket
coscmd -b examplebucket-1250000000 -e cos.somewhere.example.com createbucket
```

删除存储桶

可以配合 `-b <BucketName-APPID>` 指定 Bucket 和 `-e <endpoint>` 指定 Endpoint 使用。

```
#命令格式
coscmd -b <BucketName-APPID> deletebucket

#操作示例
coscmd deletebucket
coscmd -b examplebucket-1250000000 -e cos.somewhere.example.com deletebucket
coscmd -b examplebucket-1250000000 -e cos.somewhere.example.com deletebucket -f
```

使用 `-f` 参数则会强制删除该存储桶，包括所有文件、开启版本控制之后历史文件夹、上传产生的碎片。

上传文件或文件夹

上传文件命令如下：

```
#命令格式
coscmd upload <localpath> <cospath>

#操作示例
#将本地的 /data/exampleobject 文件上传到 cos 的 data/exampleobject 路径下
coscmd upload /data/exampleobject data/exampleobject
coscmd upload /data/exampleobject data/

#指定头部上传文件
#指定对象类型，上传一个归档的文件
coscmd upload /data/exampleobject data/exampleobject -H "{'x-cos-storage-class':'Archive'}"

#设置 meta 元属性
coscmd upload /data/exampleobject data/exampleobject -H "{'x-cos-meta-example':'example'}"
```

上传文件夹命令如下：

```
#命令格式
coscmd upload -r <localpath> <cospath>

#操作示例
coscmd upload -r /data/examplefolder data/examplefolder
coscmd upload -r /data/examplefolder data/examplefolder

#cos上的存储路径为 examplefolder2/examplefolder
coscmd upload -r /data/examplefolder examplefolder2/

#上传到 bucket 根目录
coscmd upload -r /data/examplefolder/ /

#同步上传，跳过md5相同的文件
```

```
coscmd upload -rs /data/examplefolder data/examplefolder

#忽略 .txt 和 .doc 的后缀文件
coscmd upload -rs /data/examplefolder data/examplefolder --ignore *.txt,*.doc
```

请将 "<>" 中的参数替换为您需要上传的本地文件路径（localpath），以及 COS 上存储的路径（cospath）。

> 注意：
>
> - 上传文件时需要将 COS 上的路径包括文件（文件夹）的名字补全（参考例子）。
> - COSCMD 支持大文件断点上传功能；当分片上传大文件失败时，重新上传该文件只会上传失败的分块，而不会从头开始（请保证重新上传的文件的目录以及内容和上传的目录保持一致）。
> - COSCMD 分块上传时会对每一块进行 MD5 校验。
> - COSCMD 上传默认会携带 `x-cos-meta-md5` 的头部，值为该文件的 md5 值。
> - 使用 -s 参数可以使用同步上传，跳过上传 md5 一致的文件（COS 上的原文件必须是由 1.8.3.2 之后的 COSCMD 上传的，默认带有 x-cos-meta-md5 的 header）。
> - 使用 -H 参数设置 HTTP header 时，请务必保证格式为 JSON，示例：`coscmd upload -H "{'x-cos-storage-class&':'Archive&','Content-Language':'zh-CN'}" <localpath> <cospath>`
> - 在上传文件夹时，使用 `--ignore` 参数可以忽略某一类文件，支持 shell 通配规则，支持多条规则，用逗号`,`分隔。当忽略一类后缀时，必须最后要输入`,` 或者加入 `""`。
> - 目前只支持上传最大40TB的单一文件。

下载文件或文件夹

下载文件命令如下：

```
#命令格式
coscmd download <cospath> <localpath>

#操作示例
coscmd download data/exampleobject /data/exampleobject
coscmd download data/exampleobject /data/
```

下载文件夹命令如下：

```
#命令格式
coscmd download -r <cospath> <localpath>

#操作示例
coscmd download -r data/examplefolder/ /data/examplefolder
coscmd download -r data/examplefolder/ /data/

#覆盖下载当前bucket根目录下所有的文件
coscmd download -rf / /data/examplefolder

#同步下载当前 bucket 根目录下所有的文件，跳过 md5校验相同的文件
coscmd download -rs / /data/examplefolder

#忽略 .txt 和 .doc 的后缀文件
coscmd download -rs / /data/examplefolder --ignore *.txt,*.doc
```

请将 "<>" 中的参数替换为您需要下载的 COS 上文件的路径（cospath），以及本地存储路径（localpath）。

> 注意：
>
> - 老版本的 mget 接口已经废除，download 接口使用分块下载，请使用 download 接口。
> - 若本地存在同名文件，则会下载失败，需要使用 `-f` 参数覆盖本地文件。
> - 使用 `-s` 或者 `--sync` 参数，可以在下载文件夹时跳过本地已存在的相同文件（前提是下载的文件是通过 COSCMD 的 upload 接口上传的，文件携带有 `x-cos-meta-md5` 头部）。
> - 在下载文件夹时，使用 `--ignore` 参数可以忽略某一类文件，支持 shell 通配规则，支持多条规则，用逗号`,`分隔。当忽略一类后缀时，必须最后要输入`,`或者加入 `""`。

删除文件或文件夹

删除文件命令如下：

```
#命令格式
coscmd delete <cospath>

#操作示例
coscmd delete data/exampleobject
```

删除文件夹命令如下：

```
#命令格式
coscmd delete -r <cospath>

#操作示例
coscmd delete -r /data/examplefolder/
coscmd delete -r /
```

请将"<>"中的参数替换为您需要删除的 COS 上文件的路径（cospath），工具会提示用户是否确认进行删除操作。

> **注意：**
>
> 批量删除需要输 `y` 入确定，使用 `-f` 参数则可以跳过确认直接删除。

查询分块上传文件碎片

命令如下：

```
#命令格式
coscmd listparts <cospath>

#操作示例
coscmd listparts examplefolder/
```

清除分块上传文件碎片

命令如下：

```
#命令格式
coscmd abort

#操作示例
coscmd abort
```

复制文件或文件夹

复制文件命令如下：

```
#命令格式
coscmd copy <sourcepath> <cospath>

#操作示例
#复制 examplebucket2-1250000000 存储桶下的 data/exampleobject 对象到 examplebucket1-1250000000 存储桶的 data/examplefolder/exampleobject
coscmd -b examplebucket1-1250000000 -r ap-guangzhou copy examplebucket2-1250000000.ap-beijing.myqcloud.com/data/exampleobject data/examplefolder/exampleobject

#修改存储类型，将文件类型改为低频
coscmd -b examplebucket1-1250000000 -r ap-guangzhou copy examplebucket2-1250000000.ap-beijing.myqcloud.com/data/exampleobject data/examplefolder/exampleobject -H "{'x-cos-storage-class':'STANDARD_IA'}"

#修改存储类型，将文件类型改为归档
coscmd -b examplebucket1-1250000000 -r ap-guangzhou copy examplebucket2-1250000000.ap-beijing.myqcloud.com/data/exampleobject data/examplefolder/exampleobject -H "{'x-cos-storage-class':'Archive'}"
```

复制文件夹命令如下：

```
#命令格式
coscmd copy -r <sourcepath> <cospath>

#操作示例
#复制 examplebucket2-1250000000 存储桶下的 examplefolder 目录到 examplebucket1-1250000000 存储桶的 examplefolder 目录
coscmd -b examplebucket1-1250000000 -r ap-guangzhou copy -r examplebucket2-1250000000.cos.ap-guangzhou.myqcloud.com/examplefolder/ exampl
```

```
efolder
coscmd -b examplebucket1-1250000000 -r ap-guangzhou copy -r examplebucket2-1250000000.cos.ap-guangzhou.myqcloud.com/examplefolder/ exampl
efolder/
```

请将"<>"中的参数替换为您需要复制的 COS 上文件的路径（sourcepath），和您需要复制到 COS 上文件的路径（cospath）。

> **说明：**
>
> - sourcepath 的格式为：`<BucketName-APPID>.cos.<region>.myqcloud.com/<cospath>`。
> - 使用 -d 参数可以设置 `x-cos-metadata-directive` 参数，可选值为 Copy 和 Replaced，默认为 Copy。
> - 使用 -H 参数设置 HTTP header 时，请务必保证格式为 JSON，示例：`coscmd copy -H -d Replaced "{'x-cos-storage-class':'Archive','Content-Language':'zh-CN'}" <localpath> <cospath>`

查询文件列表

查询命令如下：

```
#命令格式
coscmd list <cospath>

#操作示例
#递归查询该存储桶下所有的文件列表
coscmd list -ar

#递归查询 examplefolder 前缀的所有文件列表
coscmd list examplefolder/ -ar
```

请将"<>"中的参数替换为您需要查询文件列表的 COS 上文件的路径（cospath）。

- 使用 `-a` 查询全部文件。
- 使用 `-r` 递归查询，并且会在末尾返回列出文件的数量和大小之和。
- 使用 `-n num` 设置查询数量的最大值。

> `<cospath>` 为空默认查询当前存储桶根目录。

显示文件信息

命令如下：

```
#命令格式
coscmd info <cospath>

#操作示例
coscmd info exampleobject
```

请将"<>"中的参数替换为您需要显示的 COS 上文件的路径（cospath）。

获取带签名的下载 URL

命令如下：

```
#命令格式
coscmd signurl <cospath>

#操作示例
coscmd signurl exampleobject
coscmd signurl exampleobject -t 100
```

请将 "<>" 中的参数替换为您需要获取下载 URL 的 COS 上文件的路径（cospath）。 使用 `-t time` 设置查询签名的有效时间（单位为秒）。

开启/暂停版本控制

命令如下：

```
#命令格式
coscmd putbucketversioning <status>

#开启版本控制
coscmd putbucketversioning Enabled
```

```
#暂停版本控制
coscmd putbucketversioning Suspended
```

请将 "<>" 中的参数替换为您需要版本控制状态（status）。

> **注意：**
>
> 一旦您对存储桶启用了版本控制，它将无法返回到未启用版本控制状态（初始状态）。但是，您可以对该存储桶暂停版本控制，这样后续上传的对象将不会产生多个版本。

Debug 模式执行命令

在各命令前加上 `-d` 或者 `-debug` ，在命令执行的过程中，会显示详细的操作信息 。示例如下：

```
#显示 upload 的详细操作信息，命令格式：
coscmd -d upload <localpath> <cospath>

#操作示例
coscmd -d upload exampleobject exampleobject
```

常见问题

COSCMD 工具是否支持正则表达式？

不支持。

使用 COSCMD 工具，成功创建含有大写字符的存储桶，进行其他操作时使用大写字符报错？

COSCMD 工具会将大写字符自动转换为小写字符，存储桶名称只支持小写字母、数字、中划线及其组合，最多支持50个字符。

使用 COSCMD 工具下载根目录文件，是否支持排除某个目录？

支持。可使用 `coscmd download --ignore /folder/*` 方式过滤。当忽略某一类后缀时，必须最后要输入 `,` 或者加入 `""` 。

COS Migration工具

最近更新时间: 2025-01-15 17:01:00

功能说明

COS Migration 是一个集成了 COS 数据迁移功能的一体化工具。通过简单的配置操作，用户可以将源地址数据快速迁移至 COS 中，它具有以下特点：

- 丰富的数据源：
  - 本地数据：将本地存储的数据迁移到 COS。
  - 其他云存储：目前支持 AWS S3，阿里云 OSS，七牛存储迁移至 COS，后续会不断扩展。
  - URL 列表：根据指定的 URL 下载列表进行下载迁移到 COS。
  - Bucket 相互复制：COS 的 Bucket 数据相互复制，支持跨账号跨地域的数据复制。
- 断点续传：工具支持上传时断点续传。对于一些大文件，如果中途退出或者因为服务故障，可重新运行工具，会对未上传完成的文件进行续传。
- 分块上传：将对象按照分块的方式上传到 COS。
- 并行上传：支持多个对象同时上传。
- 迁移校验：对象迁移后的校验。

> **注意：**
>
> - COS Migration 的编码格式只支持 UTF-8 格式。
> - 使用该工具上传同名文件，会覆盖较旧的同名文件，不支持校对是否存在同名文件的功能。

使用环境

系统环境

Windows、Linux 和 macOS 系统。

软件依赖

- JDK 1.8 X64或以上，有关 JDK 的安装与配置请参见 Java 安装与配置。

使用方法

1. 获取工具

前往下载 COS Migration 工具。

2. 解压缩工具包

Windows

解压并保存到某个目录，例如

```
C:\Users\Administrator\Downloads\cos_migrate
```

Linux

解压并保存到某个目录

```
unzip cos_migrate_tool_v5-master.zip && cd cos_migrate_tool_v5-master
```

迁移工具结构

正确解压后的 COS Migration 工具目录结构如下所示：

```
COS_Migrate_tool
|——conf #配置文件所在目录
| |——config.ini #迁移配置文件
|——db #存储迁移成功的记录
|——dep #程序主逻辑编译生成的JAR包
|——log #工具执行中生成的日志
```

```
|——opbin #用于编译的脚本
|——src #工具的源码
|——tmp #临时文件存储目录
|——pom.xml #项目配置文件
|——README #说明文档
|——start_migrate.sh #Linux 下迁移启动脚本
|——start_migrate.bat #Windows 下迁移启动脚本
```

> **说明：**
>
> - db 目录主要记录工具迁移成功的文件标识，每次迁移任务会优先对比 db 中的记录，若当前文件标识已被记录，则会跳过当前文件，否则进行文件迁移。
> - log 目录记录着工具迁移时的所有日志，若在迁移过程中出现错误，请先查看该目录下的 error.log。

3. 修改 config.ini 配置文件

在执行迁移启动脚本之前，需先进行 config.ini 配置文件修改（路径：`./conf/config.ini`），config.ini 内容可以分为以下几部分：

3.1 配置迁移类型

type 表示迁移类型，用户根据迁移需求填写对应的标识。例如，需要将本地数据迁移至 COS，则 `[migrateType]` 的配置内容是 `type=migrateLocal`。

```
[migrateType]
type=migrateLocal
```

目前支持的迁移类型如下：

migrateType	描述
migrateLocal	从本地迁移至 COS
migrateAws	从 AWS S3 迁移至 COS
migrateAli	从阿里 OSS 迁移至 COS
migrateQiniu	从七牛迁移至 COS
migrateUrl	下载 URL 迁移到 COS
migrateBucketCopy	从源 Bucket 复制到目标 Bucket
migrateUpyun	从又拍云迁移到 COS

3.2 配置迁移任务

用户根据实际的迁移需求进行相关配置，主要包括迁移至目标 COS 信息配置及迁移任务相关配置。

```
# 迁移工具的公共配置分节，包含了需要迁移到目标 COS 的账户信息。
[common]
secretId=COS_SECRETID
secretKey=COS_SECRETKEY
bucketName=examplebucket-1250000000
region=ap-guangzhou
storageClass=Standard
cosPath=/
https=off
tmpFolder=./tmp
smallFileThreshold=5242880
smallFileExecutorNum=64
bigFileExecutorNum=8
entireFileMd5Attached=on
daemonMode=off
daemonModeInterVal=60
executeTimeWindow=00:00,24:00
encryptionType=sse-cos
```

名称	描述	默认值
secretId	用户密钥 SecretId，请将`COS_SECRETID`替换为您的真实密钥信息。可前往 访问管理控制台 中的云 API 密钥页面查看获取	-

名称	描述	默认值
secretKey	用户密钥 SecretKey，请将`COS_SECRETKEY`替换为您的真实密钥信息。可前往 访问管理控制台 中的云 API 密钥页面查看获取	-
bucketName	目的 Bucket 的名称, 命名格式为 ``，即 Bucket 名必须包含 APPID，例如 examplebucket-1250000000	-
region	目的 Bucket 的 Region 信息。	-
storageClass	存储类型：Standard（标准存储），Standard_IA（低频存储），Archive（归档存储）	Standard
cosPath	要迁移到的 COS 路径。`/`表示迁移到 Bucket 的根路径下，`/folder/doc/` 表示要迁移到 Bucket的`/folder/doc/`下，若 `/folder/doc/` 不存在，则会自动创建路径	/
https	是否使用 HTTPS 传输：on 表示开启，off 表示关闭。开启传输速度较慢，适用于对传输安全要求高的场景	off
tmpFolder	从其他云存储迁移至 COS 的过程中，用于存储临时文件的目录，迁移完成后会删除。要求格式为绝对路径： Linux 下分隔符为单斜杠，例如`/a/b/c` Windows 下分隔符为两个反斜杠，例如`E:\\a\\b\\c` 默认为工具所在路径下的 tmp 目录	./tmp
smallFileThreshold	小文件阈值的字节，大于等于这个阈值使用分块上传，否则使用简单上传，默认5MB	5242880
smallFileExecutorNum	小文件（文件小于 smallFileThreshold）的并发度，使用简单上传。如果是通过外网来连接 COS，且带宽较小，请减小该并发度	64
bigFileExecutorNum	大文件（文件大于等于 smallFileThreshold）的并发度，使用分块上传。如果是通过外网来连接 COS，且带宽较小，请减小该并发度	8
entireFileMd5Attached	表示迁移工具将全文的 MD5 计算后，存入文件的自定义头部 x-cos-meta-md5 中，用于后续的校验，因为 COS 的分块上传的大文件的 etag 不是全文的 MD5	on
daemonMode	是否启用 daemon 模式：on 表示开启，off 表示关闭。daemon 表示程序会循环不停的去执行同步，每一轮同步的间隔由 daemonModeInterVal 参数设置	off
daemonModeInterVal	表示每一轮同步结束后，多久进行下一轮同步，单位为秒	60
executeTimeWindow	执行时间窗口，时刻粒度为分钟，该参数定义迁移工具每天执行的时间段。例如： 参数 03:30,21:00，表示在凌晨 03:30 到晚上 21:00 之间执行任务，其他时间则会进入休眠状态，休眠态暂停迁移并会保留迁移进度, 直到下一个时间窗口自动继续执行	00:00,24:00
encryptionType	表示使用 sse-cos 服务端加密	默认不填，需要服务端加密时填写

3.3 配置数据源信息

根据 [migrateType] 的迁移类型配置相应的分节。例如 [migrateType] 的配置内容是 type=migrateLocal，则用户只需配置 [migrateLocal] 分节即可。

3.3.1 配置本地数据源 migrateLocal

若从本地迁移至 COS，则进行该部分配置，具体配置项及说明如下：

```
# 从本地迁移到 COS 配置分节
[migrateLocal]
localPath=E:\\code\\java\\workspace\\cos_migrate_tool\\test_data
excludes=
ignoreModifiedTimeLessThanSeconds=
```

配置项	描述
localPath	本地路径，要求格式为绝对路径： Linux 下分隔符为单斜杠，例如`/a/b/c` Windows 下分隔符为两个反斜杠，例如`E:\\a\\b\\c`
excludes	要排除的目录或者文件的绝对路径，表示将 localPath 下面某些目录或者文件不进行迁移，多个绝对路径之前用分号分割，不填表示 localPath 下面的全部迁移
ignoreModifiedTimeLessThanSeconds	排除更新时间与当前时间相差不足一定时间段的文件，单位为秒，默认不设置，表示不根据 lastmodified 时间进行筛选，适用于客户在更新文件的同时又在运行迁移工具，并要求不把正在更新的文件迁移上传到 COS，例如设置为300，表示只上传更新了5分钟以上的文件

3.3.2 配置阿里 OSS 数据源 migrateAli

若从阿里云 OSS 迁移至 COS，则进行该部分配置，具体配置项及说明如下：

```
# 从阿里 OSS 迁移到 COS 配置分节
[migrateAli]
bucket=bucket-aliyun
accessKeyId=yourAccessKeyId
accessKeySecret=yourAccessKeySecret
endPoint= oss-cn-hangzhou.aliyuncs.com
prefix=
proxyHost=
proxyPort=
```

配置项	描述
bucket	阿里云 OSS Bucket 名称
accessKeyId	将 yourAccessKeyId 替换为用户的密钥
accessKeySecret	将 yourAccessKeySecret 替换为用户的密钥
endPoint	阿里云 endpoint 地址
prefix	要迁移的路径的前缀，如果是迁移 Bucket 下所有的数据, 则 prefix 为空
proxyHost	如果要使用代理进行访问，则填写代理 IP 地址
proxyPort	代理的端口

3.3.3 配置 AWS 数据源 migrateAws

若从 AWS 迁移至 COS，则进行该部分配置，具体配置项及说明如下：

```
# 从 AWS 迁移到 COS 配置分节
[migrateAws]
bucket=bucket-aws
accessKeyId=AccessKeyId
accessKeySecret=SecretAccessKey
endPoint=s3.us-east-1.amazonaws.com
prefix=
proxyHost=
proxyPort=
```

配置项	描述
bucket	AWS 对象存储 Bucket 名称
accessKeyId	将 AccessKeyId 替换为用户的密钥
accessKeySecret	将 SecretAccessKey 替换为用户的密钥
endPoint	AWS 的 endpoint 地址，必须使用域名，不能使用 region
prefix	要迁移的路径的前缀，如果是迁移 Bucket 下所有的数据，则 prefix 为空
proxyHost	如果要使用代理进行访问，则填写代理 IP 地址
proxyPort	代理的端口

3.3.4 配置七牛数据源 migrateQiniu

若从七牛迁移至 COS，则进行该部分配置，具体配置项及说明如下：

```
# 从七牛迁移到COS配置分节
[migrateQiniu]
bucket=bucket-qiniu
accessKeyId=AccessKey
accessKeySecret=SecretKey
endPoint=www.bkt.clouddn.com
prefix=
```

```
proxyHost=
proxyPort=
```

配置项	描述
bucket	七牛对象存储 Bucket 名称
accessKeyId	将 AccessKey 替换为用户的密钥
accessKeySecret	将 SecretKey 替换为用户的密钥
endPoint	七牛下载地址，对应 downloadDomain
prefix	要迁移的路径的前缀，如果是迁移 Bucket 下所有的数据，则 prefix 为空
proxyHost	如果要使用代理进行访问，则填写代理 IP 地址
proxyPort	代理的端口

3.3.5 配置 URL 列表数据源 migrateUrl

若从指定 URL 列表迁移至 COS，则进行该部分配置，具体配置项及说明如下：

```
# 从 URL 列表下载迁移到 COS 配置分节
[migrateUrl]
urllistPath=D:\\folder\\urllist.txt
```

配置项	描述
urllistPath	URL 列表的地址，内容为 URL 文本，一行一条 URL 原始地址（例如 `http://imgcache.finance.cloud.tencent.com:80aaa.bbb.com/yyy/zzz.txt`，无需添加任何双引号或其他符号）。URL 列表的地址要求为绝对路径： Linux 下分隔符为单斜杠，例如`/a/b/c.txt` Windows 下分隔符为两个反斜杠，例如`E:\\a\\b\\c.txt` 如果填写的是目录，则会将该目录下的所有文件视为 urllist 文件去扫描迁移

3.3.6 配置 Bucket 相互复制 migrateBucketCopy

若从 COS 的一个指定 Bucket 迁移至另一个 Bucket，则进行该部分配置，具体配置项及说明如下：

> **注意：**
>
> 发起迁移的账号，需具备源读权限、目的写权限。

```
# 从源 Bucket 迁移到目标 Bucket 配置分节
[migrateBucketCopy]
srcRegion=ap-shanghai
srcBucketName=examplebucket-1250000000
srcSecretId=COS_SECRETID
srcSecretKey=COS_SECRETKEY
srcCosPath=/
```

配置项	描述
srcRegion	源 Bucket 的 Region 信息
srcBucketName	源 Bucket 的名称，命名格式为 ``，即 Bucket 名必须包含 APPID，例如 examplebucket-1250000000
srcSecretId	源 Bucket 隶属的用户的密钥 SecretId，可在 云 API 密钥 查看。如果是同一用户的数据，则 srcSecretId 和 common 中的 SecretId 相同，否则是跨账号 Bucket 拷贝
srcSecretKey	源 Bucket 隶属的用户的密钥 secret_key，可在 云 API 密钥 查看。如果是同一用户的数据，则 srcSecretKey 和 common 中的 secretKey 相同，否则是跨账号 Bucket 拷贝
srcCosPath	要迁移的 COS 路径，表示该路径下的文件要迁移至目标 Bucket

3.3.7 配置又拍云数据源 migrateUpyun

若从又拍云迁移至 COS，则进行该部分配置，具体配置项及说明如下：

```
[migrateUpyun]
# 从又拍迁移
bucket=xxx
#又拍云操作员的 ID
accessKeyId=xxx
#又拍云操作员的密码
accessKeySecret=xxx
prefix=

#又拍云 sdk 限制，这个 proxy 会被设置成全局的 proxy
proxyHost=
proxyPort=
```

配置项	描述
bucket	又拍云 USS Bucket 名称
accessKeyId	替换为又拍云操作员的 ID
accessKeySecret	替换为又拍云操作员的密码
prefix	要迁移的路径的前缀，如果是迁移 Bucket 下所有的数据，则 prefix 为空
proxyHost	如果要使用代理进行访问，则填写代理 IP 地址
proxyPort	代理的端口

4. 运行迁移工具

Windows

双击 **start_migrate.bat** 即可运行。

Linux

1.从 config.ini 配置文件读入配置，运行命令为：

```
sh start_migrate.sh
```

2.部分参数从命令行读入配置，运行命令为：

```
sh start_migrate.sh -Dcommon.cosPath=/savepoint0403_10/
```

> **说明：**
>
> - 工具支持配置项读取方式有两种：命令行读取或配置文件读取。
> - 命令行优先级高于配置文件，即相同配置选项会优先采用命令行里的参数。
> - 命令行中读取配置项的形式方便用户同时运行不同的迁移任务，但前提是两次任务中的关键配置项不完全一样，例如 Bucket 名称，COS 路径，要迁移的源路径等。因为不同的迁移任务写入的是不同的 db 目录，可以保证并发迁移。请参照前文中的工具结构中的 db 信息。
> - 配置项的形式为 **-D{sectionName}.{sectionKey}={sectionValue}** 的形式。其中 sectionName 是配置文件的分节名称，sectionKey 表示分节中配置项名称，sectionValue 表示分节中配置项值。如设置要迁移到的 COS 路径，则以 **-Dcommon.cosPath=/bbb/ddd** 表示。

迁移机制及流程

迁移机制原理

COS 迁移工具是有状态的，已经迁移成功的会记录在 db 目录下，以 KV 的形式存储在 leveldb 文件中。每次迁移前对要迁移的路径，先查找下 db 中是否存在， 如果存在，且属性和 db 中存在的一致， 则跳过迁移，否则进行迁移。这里的属性根据迁移类型的不同而不同，对于本地迁移，会判断 mtime。对于其他云存储迁移与 Bucket 复制，会判断源文件的 etag 和长度是否与 db 一致。因此，我们参照 db 中是否有过迁移成功的记录，而不是查找 COS，如果绕过了迁移工具，通过别的方式（如 COSCMD 或者控制台）删除修改了文件，那么运行迁移工具由于不会察觉到这种变化，是不会重新迁移的。

迁移流程步骤

1. 读取配置文件，根据迁移 type，读取相应的配置分节，并执行参数的检查。
2. 根据指定的迁移类型，扫描对比 db 下对所要迁移文件的标识，判断是否允许上传。
3. 迁移执行过程中会打印执行结果，其中 inprogress 表示迁移中，skip 表示跳过，fail 表示失败，ok 表示成功， condition_not_match 表示因为表示因不满足迁移条件而跳过的文件（如 lastmodifed 和 excludes）。失败的详细信息可以在 log 的 error 日志中查看。执行过程示意图如下图所示：

4. 整个迁移结束后会打印统计信息，包括累积的迁移成功量，失败量，跳过量，耗时。对于失败的情况，请查看 error 日志，或重新运行，因为迁移工具会跳过已迁移成功的，对未成功的会重新迁移。运行完成结果示意图如下图所示：

常见问题

如您在使用 COS Migration 工具过程中，遇到迁移失败、运行报错等异常情况，请参阅 COS Migration 工具类常见问题 寻求解决。

Hadoop工具

最近更新时间: 2025-01-15 17:01:00

功能说明

Hadoop-COS 基于对象存储COS 实现了标准的 Hadoop 文件系统，可以为 Hadoop、Spark 以及 Tez 等大数据计算框架集成 COS 提供支持，使其能够跟访问 HDFS 文件系统时相同，读写存储在 COS 上的数据。

Hadoop-COS 使用 cosn 作为 URI 的 scheme，因此也称为 Hadoop-COS 为 CosN 文件系统。

使用环境

系统环境

支持 Linux、Windows 和 macOS 系统。

软件依赖

Hadoop-2.6.0及以上版本。

下载与安装

获取 Hadoop-COS 插件

下载地址：Hadoop-COS 插件。

安装 Hadoop-COS 插件

1. 将 dep 目录下的 hadoop-cos-X.X.X-shaded.jar*， 拷贝到 `$HADOOP_HOME/share/hadoop/tools/lib` 下。

> **说明：**
>
> 根据 Hadoop 的具体版本选择对应的 jar 包，若 dep 目录中没有提供匹配版本的 jar 包，可自行通过修改 pom 文件中 Hadoop 版本号，重新编译生成。

2. 修改 hadoop_env.sh 文件。

进入 `$HADOOP_HOME/etc/hadoop` 目录，编辑 hadoop_env.sh 文件，增加以下内容，将 cosn 相关 jar 包加入 Hadoop 环境变量：

```
for f in $HADOOP_HOME/share/hadoop/tools/lib/*.jar; do
if [ "$HADOOP_CLASSPATH" ]; then
export HADOOP_CLASSPATH=$HADOOP_CLASSPATH:$f
else
export HADOOP_CLASSPATH=$f
fi
done
```

使用方法

配置项说明

属性键	说明	默认值
fs.cosn.userinfo.secretId/secretKey	填写您账户的 API 密钥信息。可登录 访问管理控制台 查看云 API 密钥。	无

属性键	说明	默认值
fs.cosn.credentials.provider	配置 secret id 和 secret key 的获取方式。当前支持三种获取方式： 1. org.apache.hadoop.fs.auth.SessionCredentialProvider：从请求 URI 中获取 secret id 和 secret key。 其格式为：`cosn://{secretId}:{secretKey}@examplebucket-1250000000/` 1. org.apache.hadoop.fs.auth.SimpleCredentialProvider： 从 core-site.xml 配置文件中读取 fs.cosn.userinfo.secretId 和 fs.cosn.userinfo.secretKey 来获取 secret id 和 secret key。 1. org.apache.hadoop.fs.auth.EnvironmentVariableCredentialProvider：从系统环境变量 COS_SECRET_ID 和 COS_SECRET_KEY 中获取。	如果不指定改配置项，默认会按照以下顺序读取 1. org.apache.hadoop.fs.auth.SessionCreden 2. org.apache.hadoop.fs.auth.SimpleCredent 3. org.apache.hadoop.fs.auth.EnvironmentVa
fs.cosn.impl	cosn 对 FileSystem 的实现类，固定为 org.apache.hadoop.fs.CosFileSystem。	无
fs.AbstractFileSystem.cosn.impl	cosn 对 AbstractFileSystem 的实现类，固定为 org.apache.hadoop.fs.CosN。	无
fs.cosn.bucket.region	请填写待访问 bucket 的地域信息。	无
fs.cosn.bucket.endpoint_suffix	指定要连接的 COS endpoint，该项为非必填项目。兼容原有配置：fs.cosn.userinfo.endpoint_suffix。	无
fs.cosn.tmp.dir	请设置一个实际存在的本地目录，运行过程中产生的临时文件会暂时放于此处。	/tmp/hadoop_cos
fs.cosn.upload.buffer	CosN 文件系统上传时依赖的缓冲区类型。当前支持三种类型的缓冲区：非直接内存缓冲区（non_direct_memory），直接内存缓冲区（direct_memory），磁盘映射缓冲区（mapped_disk）。非直接内存缓冲区使用的是 JVM 堆内存，直接内存缓冲区则使用的是堆外内存，而磁盘映射缓冲区则是基于内存文件映射得到的缓冲区。	mapped_disk
fs.cosn.upload.buffer.size	CosN 文件系统上传时依赖的缓冲区大小，如果指定为-1，则表示不限制。若不限制缓冲区大小，则缓冲区类型必须为mapped_disk。如果指定大小大于0，则要求该值至少大于等于一个 block 大小。兼容原有配置：fs.cosn.buffer.size。	-1（unlimited）
fs.cosn.block.size	CosN 文件系统每个 block 的大小，也是分块上传的每个 part size 的大小。由于 COS 的分块上传最多只能支持10000块，因此需要预估最大可能使用到的单文件大小。例如，block size 为8MB时，最大能够支持78GB的单文件上传。 block size 最大可以支持到2GB，即单文件最大可支持19TB。	8388608（8MB）
fs.cosn.upload_thread_pool	文件流式上传到 COS 时，并发上传的线程数目。	CPU核心数 X 5
fs.cosn.copy_thread_pool	目录拷贝操作时，可用于并发拷贝文件的线程数目。	CPU核心数目 X 3
fs.cosn.read.ahead.block.size	预读块的大小。	1048576（1MB）
fs.cosn.read.ahead.queue.size	预读队列的长度。	8
fs.cosn.maxRetries	访问 COS 出现错误时，最多重试的次数。	200
fs.cosn.retry.interval.seconds	每次重试的时间间隔。	3

Hadoop 配置

修改 `$HADOOP_HOME/etc/hadoop/core-site.xml` ，增加 COS 相关用户和实现类信息，例如：

```
<configuration>
<property>
<name>fs.defaultFS</name>
<value>cosn://examplebucket-1250000000</value>
</property>

<property>
```

```
<name>fs.cosn.credentials.provider</name>
<value>org.apache.hadoop.fs.auth.SimpleCredentialProvider</value>
<description>

This option allows the user to specify how to get the credentials.
Comma-separated class names of credential provider classes which implement
com.qcloud.cos.auth.COSCredentialsProvider:

1.org.apache.hadoop.fs.auth.SessionCredentialProvider: Obtain the secret id and secret key from the URI: cosn://secretId:secretKey@examplebucket-12500
00000/;
2.org.apache.hadoop.fs.auth.SimpleCredentialProvider: Obtain the secret id and secret key
from fs.cosn.userinfo.secretId and fs.cosn.userinfo.secretKey in core-site.xml;
3.org.apache.hadoop.fs.auth.EnvironmentVariableCredentialProvider: Obtain the secret id and secret key
from system environment variables named COS_SECRET_ID and COS_SECRET_KEY.

If unspecified, the default order of credential providers is:
1. org.apache.hadoop.fs.auth.SessionCredentialProvider
2. org.apache.hadoop.fs.auth.SimpleCredentialProvider
3. org.apache.hadoop.fs.auth.EnvironmentVariableCredentialProvider

</description>
</property>

<property>
<name>fs.cosn.userinfo.secretId</name>
<value>xxxxxxxxxxxxxxxxxxxxxxxxxxx</value>
<description>Tencent Cloud Secret Id</description>
</property>

<property>
<name>fs.cosn.userinfo.secretKey</name>
<value>xxxxxxxxxxxxxxxxxxxxxxxxxxx</value>
<description>Tencent Cloud Secret Key</description>
</property>

<property>
<name>fs.cosn.bucket.region</name>
<value>ap-xxx</value>
<description>The region where the bucket is located.</description>
</property>

<property>
<name>fs.cosn.bucket.endpoint_suffix</name>
<value>cos.ap-xxx.myqcloud.com</value>
<description>
COS endpoint to connect to.
For public cloud users, it is recommended not to set this option, and only the correct area field is required.
</description>
</property>

<property>
<name>fs.cosn.impl</name>
<value>org.apache.hadoop.fs.CosFileSystem</value>
<description>The implementation class of the CosN Filesystem.</description>
</property>

<property>
<name>fs.AbstractFileSystem.cosn.impl</name>
<value>org.apache.hadoop.fs.CosN</value>
<description>The implementation class of the CosN AbstractFileSystem.</description>
</property>

<property>
<name>fs.cosn.tmp.dir</name>
<value>/tmp/hadoop_cos</value>
<description>Temporary files will be placed here.</description>
</property>

<property>
<name>fs.cosn.upload.buffer</name>
<value>mapped_disk</value>
<description>The type of upload buffer. Available values: non_direct_memory, direct_memory, mapped_disk.</description>
```

```
</property>

<property>
<name>fs.cosn.upload.buffer.size</name>
<value>33554432</value>
<description>The total size of the buffer pool.</description>
</property>

<property>
<name>fs.cosn.block.size</name>
<value>8388608</value>
<description>Block size to use cosn filesysten, which is the part size for MultipartUpload.
Considering the COS supports up to 10000 blocks, user should estimate the maximum size of a single file.
For example, 8MB part size can allow writing a 78GB single file.</description>
</property>

<property>
<name>fs.cosn.maxRetries</name>
<value>3</value>
<description>
The maximum number of retries for reading or writing files to
COS, before we signal failure to the application.
</description>
</property>

<property>
<name>fs.cosn.retry.interval.seconds</name>
<value>3</value>
<description>The number of seconds to sleep between each COS retry.</description>
</property>

</configuration>
```

使用示例

命令格式为 `hadoop fs -ls -R cosn://<BucketName-APPID>/<路径>` ，或 `hadoop fs -ls -R /<路径>`（需要配置 `fs.defaultFS` 选项为 `cosn://BucketName-APPID` ），下例中以名称为 examplebucket-1250000000 的 bucket 为例，可在其后面加上具体路径。

```
hadoop fs -ls -R cosn://examplebucket-1250000000/
-rw-rw-rw- 1 root root 1087 2018-06-11 07:49 cosn://examplebucket-1250000000/LICENSE
drwxrwxrwx - root root 0 1970-01-01 00:00 cosn://examplebucket-1250000000/hdfs
drwxrwxrwx - root root 0 1970-01-01 00:00 cosn://examplebucket-1250000000/hdfs/2018
-rw-rw-rw- 1 root root 1087 2018-06-12 03:26 cosn://examplebucket-1250000000/hdfs/2018/LICENSE
-rw-rw-rw- 1 root root 2386 2018-06-12 03:26 cosn://examplebucket-1250000000/hdfs/2018/ReadMe
drwxrwxrwx - root root 0 1970-01-01 00:00 cosn://examplebucket-1250000000/hdfs/test
-rw-rw-rw- 1 root root 1087 2018-06-11 07:32 cosn://examplebucket-1250000000/hdfs/test/LICENSE
-rw-rw-rw- 1 root root 2386 2018-06-11 07:29 cosn://examplebucket-1250000000/hdfs/test/ReadMe
```

运行 MapReduce 自带的 wordcount，执行以下命令。

> **注意：**
>
> 以下命令中 hadoop-mapreduce-examples-2.7.2.jar 是以2.7.2版本为例，若版本不同，请修改成对应的版本号。

```
bin/hadoop jar share/hadoop/mapreduce/hadoop-mapreduce-examples-2.7.2.jar wordcount cosn://example/mr/input cosn://example/mr/output3
```

执行成功会返回统计信息，示例如下：

```
File System Counters
COSN: Number of bytes read=72
COSN: Number of bytes written=40
COSN: Number of read operations=0
COSN: Number of large read operations=0
COSN: Number of write operations=0
FILE: Number of bytes read=547350
FILE: Number of bytes written=1155616
FILE: Number of read operations=0
FILE: Number of large read operations=0
FILE: Number of write operations=0
HDFS: Number of bytes read=0
```

```
HDFS: Number of bytes written=0
HDFS: Number of read operations=0
HDFS: Number of large read operations=0
HDFS: Number of write operations=0
Map-Reduce Framework
Map input records=5
Map output records=7
Map output bytes=59
Map output materialized bytes=70
Input split bytes=99
Combine input records=7
Combine output records=6
Reduce input groups=6
Reduce shuffle bytes=70
Reduce input records=6
Reduce output records=6
Spilled Records=12
Shuffled Maps =1
Failed Shuffles=0
Merged Map outputs=1
GC time elapsed (ms)=0
Total committed heap usage (bytes)=653262848
Shuffle Errors
BAD_ID=0
CONNECTION=0
IO_ERROR=0
WRONG_LENGTH=0
WRONG_MAP=0
WRONG_REDUCE=0
File Input Format Counters
Bytes Read=36
File Output Format Counters
Bytes Written=40
```

HDFS TO COS工具

最近更新时间: 2025-01-15 17:01:00

功能说明

HDFS TO COS 工具用于将 HDFS 上的数据拷贝到对象存储 COS 上。

使用环境

系统环境

Linux 或 Windows 系统

软件依赖

JDK 1.7或1.8。

安装与配置

具体环境安装与配置请参见 Java 安装与配置。

配置及使用方法

配置方法

1. 安装 Hadoop-2.7.2 及以上版本，具体安装步骤请参见 Hadoop 安装与测试。
2. 在 GitHub 下载 HDFS TO COS 工具并解压缩。
3. 将要同步的 HDFS 集群的 core-site.xml 拷贝到 conf 文件夹中，其中 core-site.xml 中包含 NameNode 的配置信息。
4. 编辑配置文件 cos_info.conf ， 存储桶（Bucket）、地域（Region）以及 API 密钥信息，其中存储桶的名字，由用户自定义字符串和系统生成 APPID 数字串由中划线连接而成，例如 examplebucket-1250000000。
5. 在命令行参数中指定配置文件位置， 默认位置 conf/cos_info.conf。

> **注意：**
>
> 当命令行参数中的参数与配置文件重合时，以命令行为准。

使用方法

使用方法以 Linux 为例，如下。

查看帮助

```
./hdfs_to_cos_cmd -h
```

执行结果如下图所示：

文件拷贝

- 从 HDFS 拷贝到 COS，若 COS 上已存在同名文件， 则会覆盖原文件。

```
./hdfs_to_cos_cmd --hdfs_path=/tmp/hive --cos_path=/hdfs/20170224/
```

- 从 HDFS 拷贝到 COS，若 COS 上已存在同名且长度一致的文件时，则忽略上传（适用于拷贝一次后，重新拷贝）。

```
./hdfs_to_cos_cmd --hdfs_path=/tmp/hive --cos_path=/hdfs/20170224/ -skip_if_len_match
```

这里只做长度的判断，因为如果将 Hadoop 上的文件摘要算出，开销较大。

- 从 HDFS 拷贝到 COS，若 HDFS 中存在Har目录（Hadoop Archive 归档文件），通过指定 --decompress_har 参数可以自动解压har文件：

```
./hdfs_to_cos_cmd --decompress_har --hdfs_path=/tmp/hive --cos_path=/hdfs/20170224/
```

若未指定 --decompress_har 参数，默认按照普通的 HDFS 目录进行拷贝，即 .har 目录下的 index 和 masterindex 等文件原样拷贝。

目录信息

```
conf : 配置文件, 用于存放 core-site.xml 和 cos_info.conf
log : 日志目录
src : Java 源程序
dep : 编译生成的可运行的 JAR 包
```

问题与帮助

关于配置信息

请确保填写的配置信息正确，包括存储桶（Bucket）、地域（Region）以及 API 密钥信息，其中，存储桶的名字，由用户自定义字符串和系统生成 APPID 数字串由中划线连接而成，例如 examplebucket-1250000000。并保证机器的时间和北京时间一致（相差1分钟左右是正常的），如果相差较大，请重新设置机器时间。

关于 DateNode

请保证对于 DateNode，拷贝程序所在的机器也可以连接。NameNode 有外网 IP 可以连接，但获取的 block 所在的 DateNode 机器是内网 IP，是无法直接连接上的。因此建议同步程序放在 Hadoop 的某个节点上执行，保证对 NameNode 和 DateNode 皆可访问。

关于权限

请使用 Hadoop 命令下载文件，检查是否正常， 再使用同步工具同步 Hadoop 上的数据支持。

关于文件覆盖

对于 COS 上已存在的文件， 默认进行重传覆盖。除非用户明确的指定 -skip_if_len_match，当文件长度一致时则跳过上传。

关于 cos path

cos path 默认为是目录， 最终从 HDFS 上拷贝的文件都会存放在该目录下。

SDK文档
SDK概览

最近更新时间: 2025-01-15 17:01:00

SDK 概览

除了直接使用 API 接口外，COS 提供了丰富多样的 SDK 供开发者使用。

SDK	接入文档
Android SDK	Android SDK 快速入门
iOS SDK	iOS SDK 快速入门
C SDK	C SDK 快速入门
C++ SDK	C++ SDK 快速入门
C# SDK	C# SDK 快速入门
Go SDK	Go SDK 快速入门
Java SDK	Java SDK 快速入门
JavaScript SDK	JavaScript SDK 快速入门
Node.js SDK	Node.js SDK 快速入门
PHP SDK	PHP SDK 快速入门
Python SDK	Python SDK 快速入门

Android SDK
快速入门

最近更新时间: 2025-01-15 17:01:00

开发准备

SDK 获取

对象存储服务的 XML Android SDK 资源下载地址：XML Android SDK。 演示示例 Demo 下载地址：XML Android SDK Demo。

开发准备

1. SDK 支持 Android 2.2 及以上版本的手机系统；
2. 手机必须要有网络（GPRS、3G 或 WIFI 网络等）；
3. 手机可以没有存储空间，但会使部分功能无法正常工作；
4. 从 COS v5 控制台 获取 APPID、SecretId、SecretKey。

> **说明：**
>
> 关于文章中出现的 SecretId、SecretKey、Bucket 等名称的含义和获取方式请参考：COS 术语信息

SDK 配置

需要在工程项目中导入下列 jar 包，存放在 libs 文件夹下：

- cos-android-sdk-V5.4.3.jar
- qcloud-foundation.1.3.0.jar
- okhttp-3.8.1.jar
- okio-1.13.0.jar

或者使用gradle方式集成SDK到你的项目中，如下所示：

- compile 'com.tencent.qcloud:cosxml:5.4.3'

使用该 SDK 需要网络、存储等相关的一些访问权限，可在 AndroidManifest.xml 中增加如下权限声明（Android 5.0 以上还需要动态获取权限）：

```
<uses-permission android:name="android.permission.INTERNET"/>
<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" />
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>
```

快速入门

初始化

进行任何操作之前，都需要实例化 CosXmlService 和 CosXmlServiceConfig。

- CosXmlServiceConfig： 配置参数；
- CosXmlService：SDK 提供的服务类，可操作各种 COS 服务；

```
String appid = "对象存储的服务 APPID";
String region = "存储桶所在的地域";

String domain = "DOMAIN.com"; // 替换成用户的 Domain

String endpoint = String.format("cos.%s.%s", region, domain);

String secretId = "云 API 密钥 SecretId";
String secretKey ="云 API 密钥 SecretKey";
long keyDuration = 600; //SecretKey 的有效时间，单位秒
```

```
//创建 CosXmlServiceConfig 对象，根据需要修改默认的配置参数
CosXmlServiceConfig serviceConfig = new CosXmlServiceConfig.Builder()
.setEndpointSuffix(endpoint)
.setDebuggable(true)
.builder();

//创建获取签名类(请参考下面的生成签名示例，或者参考 sdk中提供的ShortTimeCredentialProvider类 )
LocalCredentialProvider localCredentialProvider = new LocalCredentialProvider(secretId, secretKey, keyDuration);

//创建 CosXmlService 对象，实现对象存储服务各项操作.
Context context = getApplicationContext()； //应用的上下文

CosXmlService cosXmlService = new CosXmlService(context,cosXmlServiceConfig, localCredentialProvider);
```

简单上传文件

```
String bucket = "存储桶名称"; // cos v5 的 bucket格式为：xxx-appid, 如 test-1253960454
String cosPath = "远端路径，即存储到 COS 上的绝对路径"; //格式如 cosPath = "/test.txt";
String srcPath = "本地文件的绝对路径"; // 如 srcPath = Environment.getExternalStorageDirectory().getPath() + "/test.txt";
long signDuration = 600; //签名的有效期，单位为秒

PutObjectRequest putObjectRequest = new PutObjectRequest(bucket, cosPath, srcPath);

putObjectRequest.setSign(signDuration,null,null); //若不调用，则默认使用sdk中sign duration（60s）

/*设置进度显示
实现 CosXmlProgressListener.onProgress(long progress, long max)方法，
progress 已上传的大小， max 表示文件的总大小
*/
putObjectRequest.setProgressListener(new CosXmlProgressListener() {
@Override
public void onProgress(long progress, long max) {
float result = (float) (progress * 100.0/max);
Log.w("TEST","progress =" + (long)result + "%");
}
});

//使用同步方法上传
try {
PutObjectResult putObjectResult = cosXmlService.putObject(putObjectRequest);

Log.w("TEST","success: " + putObjectResult.accessUrl);

} catch (CosXmlClientException e) {

//抛出异常
Log.w("TEST","CosXmlClientException =" + e.toString());
} catch (CosXmlServiceException e) {

//抛出异常
Log.w("TEST","CosXmlServiceException =" + e.toString());
}

//使用异步回调上传：sdk 为对象存储各项服务提供异步回调操作方法
/**

cosXmlService.putObjectAsync(putObjectRequest, new CosXmlResultListener() {
@Override
public void onSuccess(CosXmlRequest request, CosXmlResult result) {
Log.w("TEST","success =" + result.accessUrl);
}

@Override
public void onFail(CosXmlRequest cosXmlRequest, CosXmlClientException clientException, CosXmlServiceException serviceException) {

String errorMsg = clientException != null ? clientException.toString() : serviceException.toString();
Log.w("TEST",errorMsg);
}
});

*/
```

分片上传文件

分片上传一般需要经历：初始化分片上传->分块上传->上传完成 3 个阶段。

```
String bucket = "存储桶名称"; // cos v5 的 bucket格式为：xxx-appid, 如 test-1253960454
String cosPath = "远端路径，即存储到 COS 上的绝对路径";


//第一步，初始化分片上传，获取 uploadId，用于后续的分片上传、完成上传等。

String uploadId = null;

InitMultipartUploadRequest initMultipartUploadRequest = new InitMultipartUploadRequest(bucket, cosPath);

initMultipartUploadRequest.setSign(600,null,null);
try {
InitMultipartUploadResult initMultipartUploadResult =
cosXmlService.initMultipartUpload(initMultipartUploadRequest);

//若初始化成功，则获取 uploadId;
Log.w("TEST","success");
uploadId = initMultipartUploadResult.initMultipartUpload.uploadId;

} catch (CosXmlClientException e) {

//抛出异常
Log.w("TEST","CosXmlClientException =" + e.toString());
} catch (CosXmlServiceException e) {

//抛出异常
Log.w("TEST","CosXmlServiceException =" + e.toString());
}

//第二步，分片上传，需要参数 uploadId 和分片号 partNumber; 并获取对应的 eTag # 此处只演示只有一个分片的文件例子 #.
//分片号：此分片在所有分片中的编号，从 1 开始
//etag： 是此分片上传成功后，返回的此分片的 MD5+ 分片号组成的。

String srcPath = "本地文件的绝对路径";
int partNumber = 1; //上传分片编码，从 1 开始； 此处演示上传第一个分片

String eTag = null;

UploadPartRequest uploadPartRequest = new UploadPartRequest(bucket, cosPath, partNumber,
srcPath, uploadId);

uploadPartRequest.setSign(600,null,null);

/*设置进度显示
实现 CosXmlProgressListener.onProgress(long progress, long max)方法，
progress已上传的大小， max 表示文件的总大小
*/
uploadPartRequest.setProgressListener(new CosXmlProgressListener() {
@Override
public void onProgress(long progress, long max) {
float result = (float) (progress * 100.0/max);
Log.w("TEST","progress =" + (long)result + "%");
}
});

try {
UploadPartResult uploadPartResult = cosXmlService.uploadPart(uploadPartRequest);

Log.w("TEST","success");
eTag = uploadPartResult.eTag; // 获取分片文件的 eTag

} catch (CosXmlClientException e) {

//抛出异常
Log.w("TEST","CosXmlClientException =" + e.toString());
} catch (CosXmlServiceException e) {
```

```
//抛出异常
Log.w("TEST","CosXmlServiceException =" + e.toString());
}


//第三步，当确定所有分片全部上传完成之后，调用 CompleteMultiUploadRequest 完成分片上传结束。
//需要参数 uploadId， partNumber 和对应每块分片文件的 eTag 值

CompleteMultiUploadRequest completeMultiUploadRequest = new CompleteMultiUploadRequest(bucket, cosPath, uploadId, null);

completeMultiUploadRequest.setPartNumberAndEtag(partNumber, eTag); //此处只演示一个分片的例子

completeMultiUploadRequest.setSign(600,null,null);
try {
CompleteMultiUploadResult completeMultiUploadResult =
cosXmlService.completeMultiUpload(completeMultiUploadRequest);

Log.w("TEST","success: " + completeMultiUploadResult.accessUrl );

} catch (CosXmlClientException e) {

//抛出异常
Log.w("TEST","CosXmlClientException =" + e.toString());
} catch (CosXmlServiceException e) {

//抛出异常
Log.w("TEST","CosXmlServiceException =" + e.toString());
}
```

UploadService，推荐使用该方法进行分片上传

```
//UploadService 封装了上述分片上传请求一系列过程的类

UploadService.ResumeData resumeData = new UploadService.ResumeData();
resumeData.bucket = "存储桶名称";
resumeData.cosPath = "远端路径，即存储到 COS 上的绝对路径"; //格式如 cosPath = "/test.txt";
resumeData.srcPath = "本地文件的绝对路径"; // 如 srcPath =Environment.getExternalStorageDirectory().getPath() + "/test.txt";
resumeData.sliceSize = 1024 * 1024; //每个分片的大小
resumeData.uploadId = null; //若是续传，则uploadId不为空


UploadService uploadService = new UploadService(cosXmlService, resumeData);

/*设置进度显示
实现 CosXmlProgressListener.onProgress(long progress, long max)方法，
progress 已上传的大小， max 表示文件的总大小
*/
uploadService.setProgressListener(new CosXmlProgressListener() {
@Override
public void onProgress(long progress, long max) {
float result = (float) (progress * 100.0/max);
Log.w("TEST","progress =" + (long)result + "%");
}
});
try {
CosXmlResult cosXmlResult = uploadService.upload();

Log.w("TEST","success: " + cosXmlResult.accessUrl );

} catch (CosXmlClientException e) {

//抛出异常
Log.w("TEST","CosXmlClientException =" + e.toString());
} catch (CosXmlServiceException e) {

//抛出异常
Log.w("TEST","CosXmlServiceException =" + e.toString());
}
```

下载文件

```
String bucket = "存储桶名称"; // cos v5 的 bucket格式为：xxx-appid, 如 test-1253960454
String cosPath = "远端路径，即存储到 COS 上的绝对路径";
String savePath = "下载到本地的路径";

GetObjectRequest getObjectRequest = GetObjectRequest(bucket, cosPath, savePath);
getObjectRequest.setSign(signDuration,null,null);

/*设置进度显示
实现 CosXmlProgressListener.onProgress(long progress, long max)方法，
progress 已上传的大小， max 表示文件的总大小
*/
getObjectRequest.setProgressListener(new CosXmlProgressListener() {
@Override
public void onProgress(long progress, long max) {
float result = (float) (progress * 100.0/max);
Log.w("TEST","progress =" + (long)result + "%");
}
});

//使用同步方法下载
try {
GetObjectResult getObjectResult =cosXmlService.getObject(getObjectRequest);
Log.w("TEST","success： " + getObjectResult.xCOSStorageClass);
} catch (CosXmlClientException e) {
Log.w("TEST","CosXmlClientException =" + e.toString());
} catch (CosXmlServiceException e) {
Log.w("TEST","CosXmlServiceException =" + e.toString());
}

//**使用异步回调请求**
/**

cosXmlService.getObjectAsync(getObjectRequest, new CosXmlResultListener() {
@Override
public void onSuccess(CosXmlRequest cosXmlRequest, CosXmlResult cosXmlResult) {
Log.w("TEST","success");
}

@Override
public void onFail(CosXmlRequest cosXmlRequest, CosXmlClientException clientException, CosXmlServiceException serviceException) {
String errorMsg = clientException != null ? clientException.toString() : serviceException.toString();
Log.w("TEST",errorMsg);
}
});

*/
```

生成签名

若需要了解签名具体的生成过程请参照 请求签名。 在使用 SDK 时，SDK 中已提供了签名获取类，只需要继承 BasicLifecycleCredentialProvider 类，并重写 fetchNewCredentials() 方法，从而获取 SecretId，SecretKey， SecretKey Duration；若是使用临时密钥发送请求，则需要获取tempSecretKey, tempSecrekId, sessionToken, expiredTime，关于如何通过CAM获取临时密钥，请参考快速搭建移动应用传输服务。

示例

```
/**
方法一：使用永久密钥进行签名
*/
public class LocalCredentialProvider extends BasicLifecycleCredentialProvider{
private String secretKey;
private long keyDuration;
private String secretId;

public LocalCredentialProvider(String secretId, String secretKey, long keyDuration) {
this.secretId = secretId;
this.secretKey = secretKey;
```

```
this.keyDuration = keyDuration;
}

/**
返回 BasicQCloudCredentials
*/
@Override
public QCloudLifecycleCredentials fetchNewCredentials() throws CosXmlClientException {
long current = System.currentTimeMillis() / 1000L;
long expired = current + duration;
String keyTime = current+";"+expired;
return new BasicQCloudCredentials(secretId, secretKeyToSignKey(secretKey, keyTime), keyTime);
}

private String secretKeyToSignKey(String secretKey, String keyTime) {
String signKey = null;
try {
if (secretKey == null) {
throw new IllegalArgumentException("secretKey is null");
}
if (keyTime == null) {
throw new IllegalArgumentException("qKeyTime is null");
}
} catch (IllegalArgumentException e) {
e.printStackTrace();
}
try {
byte[] byteKey = secretKey.getBytes("utf-8");
SecretKey hmacKey = new SecretKeySpec(byteKey, "HmacSHA1");
Mac mac = Mac.getInstance("HmacSHA1");
mac.init(hmacKey);
signKey = StringUtils.toHexString(mac.doFinal(keyTime.getBytes("utf-8")));
} catch (UnsupportedEncodingException e) {
e.printStackTrace();
} catch (NoSuchAlgorithmException e) {
e.printStackTrace();
} catch (InvalidKeyException e) {
e.printStackTrace();
}
return signKey;
}
}

/**
方法二：使用临时密钥进行签名（推荐使用这种方法），此处假设已获取了临时密钥 tempSecretKey, tempSecrekId,
sessionToken, expiredTime.
*/
public class LocalSessionCredentialProvider extends BasicLifecycleCredentialProvider{
private String tempSecretId;
private String tempSecretKey;
private String sessionToken;
private long expiredTime;

public LocalCredentialProvider(String tempSecretId, String tempSecretKey, String sessionToken, long expiredTime) {
this.tempSecretId = tempSecretId;
this.tempSecretKey = tempSecretKey;
this.sessionToken = sessionToken;
this.expiredTime = keyDuration;
}

/**
返回 SessionQCloudCredential
*/
@Override
public QCloudLifecycleCredentials fetchNewCredentials() throws CosXmlClientException {
return new SessionQCloudCredentials(tmpSecretId, tmpSecretKey, sessionToken, expiredTime);
}
}
```

接口文档

最近更新时间: 2025-01-15 17:01:00

SDK 异常信息

SDK中，若是调用接口操作cos 对象失败，会抛出 CosXmlClientException 异常 或者 CosXmlServiceException 异常。如接口参数填写错误将抛出 CosXmlClientException 异常，cos服务端返回的错误将抛出 CosXmlServiceException 异常，其中 CosXmlServiceException 异常中 requestId 属性可以查到cos服务端返回错误原因.

初始化

进行操作之前需要实例化 CosXmlService 和 CosXmlServiceConfig。

> **说明：**
>
> 关于文章中出现的 SecretId、SecretKey、Bucket 等名称的含义和获取方式请参考：COS 术语信息。

实例化 CosXmlServiceConfig

调用 CosXmlServiceConfig.Builder().builder() 实例化 CosXmlServiceConfig 对象。

参数说明

参数名称	参数描述	类型	必填
appid	对象存储的服务 APPID	String	是
region	存储桶 所在的地域	String	是

其它配置设置方法

方法	方法描述
setAppidAndRegion(String, String)	设置 appid 和 bucket 所属地域
isHttps(boolean)	true：https请求； false：http请求； 默认 http 请求
setDebuggable(boolean)	debug log调式

示例

```
String appid = "对象存储的服务 APPID";
String region = "存储桶所在的地域"; //所属地域：在创建好存储桶后，可通过对象存储控制台查看
CosXmlServiceConfig serviceConfig = new CosXmlServiceConfig.Builder()
.isHttps(true)
.setAppidAndRegion(appid, region)
.setDebuggable(true)
.builder();
```

实例化 CosXmlService

调用 `CosXmlService(Context context, CosXmlServiceConfig serviceConfig, QCloudCredentialProvider cloudCredentialProvider)` 构造方法，实例化 CosXmlService 对象。

参数说明

参数名称	参数描述	类型	必填
context	application 上下文	Context	是
serviceConfig	SDK 的配置设置类	CosXmlServiceConfig	是
basicLifecycleCredentialProvider	服务请求的签名获取类	BasicLifecycleCredentialProvider	是

示例

```
String appid = "对象存储的服务 APPID";
String region = "存储桶所在的地域";

//创建 CosXmlServiceConfig 对象，根据需要修改默认的配置参数
CosXmlServiceConfig serviceConfig = new CosXmlServiceConfig.Builder()
.isHttps(true)
.setAppidAndRegion(appid, region)
.setDebuggable(true)
.builder();

/**
*
* 创建 ShortTimeCredentialProvider 签名获取类对象，用于使用对象存储服务时计算签名.
* 参考 SDK 提供签名格式，可实现自己的签名方法(extends BasicLifecycleCredentialProvider 以及实现 * * fetchNewCredentials() 方法).
* 此处使用SDK提供的默认签名计算方法.
*
*/
String secretId = "云 API 密钥 secretId";
String secretKey ="云 API 密钥 secretKey";
long keyDuration = 600; //secretKey 的有效时间,单位秒
ShortTimeCredentialProvider localCredentialProvider = new ShortTimeCredentialProvider(secretId, secretKey, keyDuration);

//创建 CosXmlService 对象，实现对象存储服务各项操作.
Context context = getApplicationContext()； //应用的上下文
CosXmlService cosXmlService = new CosXmlService(context,cosXmlServiceConfig, localCredentialProvider);
```

生成签名

签名具体的生成和使用请参照 [请求签名] 文章。SDK 中已提供了签名获取类，用户只需要继承 BasicLifecycleCredentialProvider 类，并重写 fetchNewCredentials() 方法。其中，临时密钥获取获取方法，请参考快速搭建移动应用传输服务。

示例

```
/**
方法一：使用永久密钥进行签名
*/
public class LocalCredentialProvider extends BasicLifecycleCredentialProvider{
private String secretKey;
private long keyDuration;
private String secretId;

public LocalCredentialProvider(String secretId, String secretKey, long keyDuration) {
this.secretId = secretId;
this.secretKey = secretKey;
this.keyDuration = keyDuration;
}

/**
返回 BasicQCloudCredentials
*/
@Override
public QCloudLifecycleCredentials fetchNewCredentials() throws CosXmlClientException {
long current = System.currentTimeMillis() / 1000L;
long expired = current + duration;
String keyTime = current+";"+expired;
return new BasicQCloudCredentials(secretId, secretKeyToSignKey(secretKey, keyTime), keyTime);
}

private String secretKeyToSignKey(String secretKey, String keyTime) {
String signKey = null;
try {
if (secretKey == null) {
throw new IllegalArgumentException("secretKey is null");
}
if (keyTime == null) {
throw new IllegalArgumentException("qKeyTime is null");
```

```
}
} catch (IllegalArgumentException e) {
e.printStackTrace();
}
try {
byte[] byteKey = secretKey.getBytes("utf-8");
SecretKey hmacKey = new SecretKeySpec(byteKey, "HmacSHA1");
Mac mac = Mac.getInstance("HmacSHA1");
mac.init(hmacKey);
signKey = StringUtils.toHexString(mac.doFinal(keyTime.getBytes("utf-8")));
} catch (UnsupportedEncodingException e) {
e.printStackTrace();
} catch (NoSuchAlgorithmException e) {
e.printStackTrace();
} catch (InvalidKeyException e) {
e.printStackTrace();
}
return signKey;
}
}

/**
方法二：使用临时密钥进行签名（推荐使用这种方法），此处假设已获取了临时密钥 tempSecretKey, tempSecrekId,
sessionToken, expiredTime.
*/
public class LocalSessionCredentialProvider extends BasicLifecycleCredentialProvider{
private String tempSecretId;
private String tempSecretKey;
private String sessionToken;
private long expiredTime;

public LocalSessionCredentialProvider(String tempSecretId, String tempSecretKey, String sessionToken, long expiredTime) {
this.tempSecretId = tempSecretId;
this.tempSecretKey = tempSecretKey;
this.sessionToken = sessionToken;
this.expiredTime = keyDuration;
}

/**
返回 SessionQCloudCredential
*/
@Override
public QCloudLifecycleCredentials fetchNewCredentials() throws CosXmlClientException {
return new SessionQCloudCredentials(tmpSecretId, tmpSecretKey, sessionToken, expiredTime);
}
}
```

简单上传文件

调用此接口可以将本地的文件上传至指定 Bucket 中。具体步骤如下：

1. 调用 `PutObjectRequest（String, String, String）` 构造方法，实例化 PutObjectRequest 对象。
2. 调用 CosXmlService 的 putObject 方法，传入 PutObjectRequest，返回 PutObjectResult 对象。（或者调用 putObjectAsync 方法，传入 PutObjectRequest 和 CosXmlResultListener 进行异步回调操作）。

参数说明

参数名称	参数描述	类型	必填
bucket	存储桶名称(cos v5 的 bucket格式为：xxx-appid, 如 test-1253960454)	String	是
cosPath	远端路径，即存储到 COS 上的绝对路径	String	是
srcPath	本地文件的绝对路径	String	是
signDuration	签名的有效期，单位为秒	Long	是
checkHeaderListForSign	签名中需要验证的请求头	Set	否

参数名称	参数描述	类型	必填
checkParameterListForSing	签名中需要验证的请求参数	Set	否
qCloudProgressListener	上传进度回调	CosXmlProgressListener	否
cosXmlResultListener	上传结果回调	CosXmlResultListener	否

返回结果说明

通过 PutObjectResult 对象的成员变量返回请求结果。

成员变量名称	变量说明	类型
accessUrl	请求成功时，返回访问文件的地址	String
httpCode	[200, 300)之间请求成功， 否则请求失败	Int

示例

```
String bucket = "bucket";
String cosPath = "cosPath";
String srcPath = "本地文件的绝对路径";

PutObjectRequest putObjectRequest = new PutObjectRequest(bucket, cosPath, srcPath);
putObjectRequest.setSign(signDuration,null,null);

putObjectRequest.setProgressListener(new CosXmlProgressListener() {
@Override
public void onProgress(long progress, long max) {
float result = (float) (progress * 100.0/max);
Log.w("TEST","progress =" + (long)result + "%");
}
});

//使用同步方法上传
try {
PutObjectResult putObjectResult = cosXmlService.putObject(putObjectRequest);

Log.w("TEST","success: " + putObjectResult.accessUrl);

} catch (CosXmlClientException e) {

//抛出异常
Log.w("TEST","CosXmlClientException =" + e.toString());
} catch (CosXmlServiceException e) {

//抛出异常
Log.w("TEST","CosXmlServiceException =" + e.toString());
}

//**使用异步回调上传**
/**

cosXmlService.putObjectAsync(putObjectRequest, new CosXmlResultListener() {
@Override
public void onSuccess(CosXmlRequest cosXmlRequest, CosXmlResult cosXmlResult) {

Log.w("TEST","success");
}

@Override
public void onFail(CosXmlRequest cosXmlRequest, CosXmlClientException clientException, CosXmlServiceException

serviceException) {

String errorMsg = clientException != null ? clientException.toString() : serviceException.toString();
Log.w("TEST",errorMsg);
}
```

```
});
*/
```

分片上传(推荐使用 UploadServer 来完成分片上传)

初始化分片

调用此接口实现初始化分片上传，成功执行此请求以后会返回 UploadId 用于后续的 Upload Part 请求。具体步骤如下：

1. 调用 InitMultipartUploadRequest（String, String）构造方法，实例化 InitMultipartUploadRequest 对象。
2. 调用 CosXmlService 的 initMultipartUpload 方法，传入 InitMultipartUploadRequest，返回 InitMultipartUploadResult 对象。（或者调用 initMultipartUploadAsync 方法，传入 InitMultipartUploadRequest 和 CosXmlResultListener 进行异步回调操作）。

参数说明

参数名称	参数描述	类型	必填
bucket	存储桶名称(cos v5 的 bucket格式为：xxx-appid, 如 test-1253960454)	String	是
cosPath	远端路径，即存储到 COS 上的绝对路径	String	是
signDuration	签名的有效期，单位为秒	Long	是
checkHeaderListForSign	签名中需要验证的请求头	Set	否
checkParameterListForSing	签名中需要验证的请求参数	Set	否
cosXmlResultListener	上传结果回调	CosXmlResultListener	否

返回结果说明

通过 InitMultipartUploadResult 对象的成员变量返回请求结果。

成员变量名称	变量说明	类型
initMultipartUpload	请求成功的返回结果	InitMultipartUpload
httpCode	[200, 300)之间请求成功， 否则请求失败	Int

示例

```
String bucket = "bucket";
String cosPath = "cosPath";

InitMultipartUploadRequest initMultipartUploadRequest = new InitMultipartUploadRequest(bucket, cosPath);
initMultipartUploadRequest.setSign(signDuration,null,null);

String uploadId = null;

//使用同步方法请求
try {
InitMultipartUploadResult initMultipartUploadResult = cosXmlService.initMultipartUpload(initMultipartUploadRequest);

Log.w("TEST","success");
uploadId =initMultipartUploadResult.initMultipartUpload.uploadId;

} catch (CosXmlClientException e) {

Log.w("TEST","CosXmlClientException =" + e.toString());

} catch (CosXmlServiceException e) {

Log.w("TEST","CosXmlServiceException =" + e.toString());
}

//**使用异步回调请求**
/**
```

```
cosXmlService.initMultipartUploadAsync(initMultipartUploadRequest, new CosXmlResultListener() {
@Override
public void onSuccess(CosXmlRequest cosXmlRequest, CosXmlResult cosXmlResult) {

Log.w("TEST","success");
uploadId = ((InitMultipartUploadResult)cosXmlResult).initMultipartUpload.uploadId;
}

@Override
public void onFail(CosXmlRequest cosXmlRequest, CosXmlClientException clientException, CosXmlServiceException

serviceException) {

String errorMsg = clientException != null ? clientException.toString() : serviceException.toString();
Log.w("TEST",errorMsg);
}
});

*/
```

上传分片

调用此接口实现分块上传，支持的块的数量为 1 到 10000，块的大小为 1 MB 到 5 GB。具体步骤如下：

1. 调用 `UploadPartRequest（String, String, int, String, String）` 构造方法，实例化 UploadPartRequest 对象.
2. 调用 CosXmlService 的 uploadPart 方法，传入 UploadPartRequest，返回 UploadPartResult 对象。（或者调用 uploadPartAsync 方法，传入 UploadPartRequest 和 CosXmlResultListener 进行异步回调操作）。

参数说明

参数名称	参数描述	类型	必填
bucket	存储桶名称(cos v5 的 bucket格式为：xxx-appid, 如 test-1253960454)	String	是
cosPath	远端路径，即存储到 COS 上的绝对路径	String	是
uploadId	初始化分片上传，返回的 uploadId	String	是
partNumber	分片块的编号，从 1 开始起	Int	是
srcPath	本地文件的绝对路径	String	是
fileOffset	该分片在文件的中起始位置	Long	否
contentLength	该分片的内容大小	Long	否
signDuration	签名的有效期，单位为秒	Long	否
checkHeaderListForSign	签名中需要验证的请求头	Set	否
checkParameterListForSing	签名中需要验证的请求参数	Set	否
qCloudProgressListener	上传进度回调	CosXmlProgressListener	否
cosXmlResultListener	上传结果回调	CosXmlResultListener	否

返回结果说明

通过 UploadPartResult 对象的成员变量返回请求结果。

成员变量名称	类型	变量说明
eTag	请求成功，返回分片文件的 MD5 值，用于最后完成分片	String
httpCode	[200, 300)之间请求成功， 否则请求失败	Int

示例

```
String bucket = "bucket";
```

```
String cosPath = "cosPath";
String uploadId = "初始化分片返回的 uploadId";
int partNumber = 1;//此次上传分片的编号，从 1 开始
String srcPath = "本地文件的绝对路径";

UploadPartRequest uploadPartRequest = new UploadPartRequest(bucket, cosPath, partNumber, srcPath, uploadId);
uploadPartRequest.setSign(signDuration,null,null);

uploadPartRequest.setProgressListener(new CosXmlProgressListener() {
@Override
public void onProgress(long progress, long max) {
float result = (float) (progress * 100.0/max);
Log.w("TEST","progress =" + (long)result + "%");
}
});

String eTag = null;

//使用同步方法上传
try {
UploadPartResult uploadPartResult = cosXmlService.uploadPart(uploadPartRequest);

Log.w("TEST","success");
eTag = uploadPartResult.eTag; // 获取分片文件的 eTag

} catch (CosXmlClientException e) {

Log.w("TEST","CosXmlClientException =" + e.toString());

} catch (CosXmlServiceException e) {

Log.w("TEST","CosXmlServiceException =" + e.toString());
}
//**使用异步回调请求**
/**

cosXmlService.uploadPartAsync(uploadPartRequest, new CosXmlResultListener() {
@Override
public void onSuccess(CosXmlRequest cosXmlRequest, CosXmlResult cosXmlResult) {

Log.w("TEST","success");
eTag =((UploadPartResult)cosXmlResult).eTag;
}

@Override
public void onFail(CosXmlRequest cosXmlRequest, CosXmlClientException clientException, CosXmlServiceException

serviceException) {

String errorMsg = clientException != null ? clientException.toString() : serviceException.toString();
Log.w("TEST",errorMsg);
}
});

*/
```

完成整个分片上传

当上传完所有分块以后，必须调用此接口用来实现完成整个分块上传。具体步骤如下：

1. 调用 CompleteMultiUploadRequest（String, String, String, Map<Integer, String>）构造方法，实例化 CompleteMultiUploadRequest 对象。
2. 调用 CosXmlService 的 completeMultiUpload 方法，传入 CompleteMultiUploadRequest，返回 CompleteMultiUploadResult 对象。（或者调用 completeMultiUploadAsync 方法，传入 CompleteMultiUploadRequest 和 CosXmlResultListener 进行异步回调操作）。

参数说明

参数名称	参数描述	类型	必填
bucket	存储桶名称(cos v5 的 bucket格式为：xxx-appid, 如 test-1253960454)	String	是
cosPath	远端路径，即存储到 COS 上的绝对路径	String	是

参数名称	参数描述	类型	必填
uploadId	初始化分片上传，返回的 uploadId	String	是
partNumberAndETag	分片编号 和对应的分片 MD5 值	Map	是
signDuration	签名的有效期，单位为秒	Long	是
checkHeaderListForSign	签名中需要验证的请求头	Set	否
checkParameterListForSing	签名中需要验证的请求参数	Set	否
cosXmlResultListener	上传结果回调	CosXmlResultListener	否

返回结果说明

通过 CompleteMultiUploadResult 对象的成员变量返回请求结果。

成员变量名称	变量说明	类型
completeMultipartUpload	请求成功的返回结果	CompleteMultipartResult
accessUrl	请求成功时，返回访问文件的地址	String

示例

```
String bucket = "bucket";
String cosPath = "cosPath";
String uploadId = "初始化分片返回的 uploadId";
int partNumber = 1;
String etag = "编号为 partNumber 对应分片上传结束返回的 etag ";
Map<Integer, String> partNumberAndETag = new HashMap<>();
partNumberAndETag.put(partNumber, etag);

CompleteMultiUploadRequest completeMultiUploadRequest = new CompleteMultiUploadRequest(bucket, cosPath, uploudId,

partNumberAndETag);
completeMultiUploadRequest.setSign(signDuration,null,null);

//使用同步方法请求
try {
CompleteMultiUploadResult completeMultiUploadResult = cosXmlService.completeMultiUpload(completeMultiUploadRequest);

Log.w("TEST","success: "+ completeMultiUploadResult.completeMultipartUpload.toString()());
} catch (CosXmlClientException e) {

Log.w("TEST","CosXmlClientException =" + e.toString());

} catch (CosXmlServiceException e) {

Log.w("TEST","CosXmlServiceException =" + e.toString());
}

//**使用异步回调请求**
/**

cosXmlService.completeMultiUploadAsync(completeMultiUploadRequest, new CosXmlResultListener() {
@Override
public void onSuccess(CosXmlRequest cosXmlRequest, CosXmlResult cosXmlResult) {

Log.w("TEST","success");
}

@Override
public void onFail(CosXmlRequest cosXmlRequest, CosXmlClientException clientException, CosXmlServiceException

serviceException) {

String errorMsg = clientException != null ? clientException.toString() : serviceException.toString();
Log.w("TEST",errorMsg);
```

```
}
});

*/
```

列举已上传的分片

调用此接口用来查询特定分块上传中的已上传的块，即罗列出指定 UploadId 所属的所有已上传成功的分块。

1. 调用 ListPartsRequest（String, String, String）构造方法，实例化 ListPartsRequest 对象。
2. 调用 CosXmlService 的 listParts 方法，传入 ListPartsRequest，返回 ListPartsResult 对象。（或者调用 listPartsAsync 方法，传入 ListPartsRequest 和 CosXmlResultListener 进行异步回调操作）。

参数说明

参数名称	参数描述	类型	必填
bucket	存储桶名称(cos v5 的 bucket格式为：xxx-appid, 如 test-1253960454)	String	是
cosPath	远端路径，即存储到 COS 上的绝对路径	String	是
uploadId	初始化分片上传，返回的 uploadId	String	是
signDuration	签名的有效期，单位为秒	Long	是
checkHeaderListForSign	签名中需要验证的请求头	Set	否
checkParameterListForSing	签名中需要验证的请求参数	Set	否
cosXmlResultListener	上传结果回调	CosXmlResultListener	否

返回结果说明

通过 ListPartsResult 对象的成员变量返回请求结果。

成员变量名称	变量说明	类型
listParts	请求成功返回的结果	ListParts
httpCode	[200, 300)之间请求成功， 否则请求失败	Int

示例

```
String bucket = "bucket";
String cosPath = "cosPath";
String uploadId = "初始化分片返回的 uploadId";

ListPartsRequest listPartsRequest = new ListPartsRequest(bucket, cosPath, uploadId);
listPartsRequest.setSign(signDuration,null,null);

//使用同步方法请求
try {
ListPartsResult listPartsResult = cosXmlService.listParts(listPartsRequest);

Log.w("TEST","success: " + listPartsResult.listParts.toString());

} catch (CosXmlClientException e) {

Log.w("TEST","CosXmlClientException =" + e.toString());

} catch (CosXmlServiceException e) {

Log.w("TEST","CosXmlServiceException =" + e.toString());
}

//**使用异步回调请求**
/**

cosXmlService.listPartsAsync(listPartsRequest, new CosXmlResultListener() {
```

```
@Override
public void onSuccess(CosXmlRequest cosXmlRequest, CosXmlResult cosXmlResult) {

Log.w("TEST","success");
}

@Override
public void onFail(CosXmlRequest cosXmlRequest, CosXmlClientException clientException, CosXmlServiceException

serviceException) {

String errorMsg = clientException != null ? clientException.toString() : serviceException.toString();
Log.w("TEST",errorMsg);
}
});

*/
```

舍弃并删除已上传的分片

调用此接口用来用来实现舍弃一个分块上传并删除已上传的块。

1. 调用 AbortMultiUploadRequest（String, String, String）构造方法，实例化 AbortMultiUploadRequest 对象。
2. 调用 CosXmlService 的 abortMultiUpload 方法，传入 AbortMultiUploadRequest，返回 AbortMultiUploadResult 对象。（或者调用 abortMultiUploadAsync 方法，传入 AbortMultiUploadRequest 和 CosXmlResultListener 进行异步回调操作）。

参数说明

参数名称	参数描述	类型	必填
bucket	存储桶名称(cos v5 的 bucket格式为：xxx-appid, 如 test-1253960454)	String	是
cosPath	远端路径，即存储到 COS 上的绝对路径	String	是
uploadId	初始化分片上传，返回的 uploadId	String	是
signDuration	签名的有效期，单位为秒	Long	是
checkHeaderListForSign	签名中需要验证的请求头	Set	否
checkParameterListForSing	签名中需要验证的请求参数	Set	否
cosXmlResultListener	上传结果回调	CosXmlResultListener	否

返回结果说明

通过 AbortMultiUploadResult 对象的成员变量返回请求结果。

成员变量名称	变量说明	类型
httpCode	[200, 300)之间请求成功， 否则请求失败	Int

示例

```
String bucket = "bucket";
String cosPath = "cosPath";
String uploadId = "初始化分片返回的 uploadId";

AbortMultiUploadRequest abortMultiUploadRequest = new AbortMultiUploadRequest(bucket, cosPath, uploadId);
abortMultiUploadRequest.setSign(signDuration,null,null);

//使用同步方法请求
try {
AbortMultiUploadResult abortMultiUploadResult = cosXmlService.abortMultiUpload(abortMultiUploadRequest);
Log.w("TEST", "success");

} catch (CosXmlClientException e) {

Log.w("TEST","CosXmlClientException =" + e.toString());
```

```
} catch (CosXmlServiceException e) {

Log.w("TEST","CosXmlServiceException =" + e.toString());
}
//**使用异步回调请求**
/**

cosXmlService.abortMultiUploadAsync(abortMultiUploadRequest, new CosXmlResultListener() {
@Override
public void onSuccess(CosXmlRequest cosXmlRequest, CosXmlResult cosXmlResult) {

Log.w("TEST","success");
}

@Override
public void onFail(CosXmlRequest cosXmlRequest, CosXmlClientException clientException, CosXmlServiceException

serviceException) {

String errorMsg = clientException != null ? clientException.toString() : serviceException.toString();
Log.w("TEST",errorMsg);
}
});

*/
```

删除文件

删除单个文件

调用此接口可以在指定的 Bucket 中将一个文件删除。具体步骤如下：

1. 调用 DeleteObjectRequest（String, String）构造方法，实例化 DeleteObjectRequest 对象。
2. 调用 CosXmlService 的 completeMultiUpload 方法，传入 DeleteObjectRequest，返回 DeleteObjectResult 对象。（或者调用 deleteObjectAsync 方法，传入 DeleteObjectRequest 和 CosXmlResultListener 进行异步回调操作）。

参数说明

参数名称	参数描述	类型	必填
bucket	存储桶名称(cos v5 的 bucket格式为：xxx-appid, 如 test-1253960454)	String	是
cosPath	远端路径，即存储到 COS 上的绝对路径	String	是
signDuration	签名的有效期，单位为秒	Long	是
checkHeaderListForSign	签名中需要验证的请求头	Set	否
checkParameterListForSing	签名中需要验证的请求参数	Set	否
cosXmlResultListener	上传结果回调	CosXmlResultListener	否

返回结果说明

通过 DeleteObjectResult 对象的成员变量返回请求结果。

成员变量名称	变量说明	类型
httpCode	[200, 300)之间请求成功， 否则请求失败	Int

示例

```
String bucket = "bucket";
String cosPath = "cosPath";

DeleteObjectRequest deleteObjectRequest = new DeleteObjectRequest(bucket, cosPath);
deleteObjectRequest.setSign(signDuration,null,null);
```

```
//使用同步方法删除
try {
DeleteObjectResult deleteObjectResult = cosXmlService.deleteObject(deleteObjectRequest);
Log.w("TEST","success ");

} catch (CosXmlClientException e) {
Log.w("TEST","CosXmlClientException =" + e.toString());
} catch (CosXmlServiceException e) {
Log.w("TEST","CosXmlServiceException =" + e.toString());
}

//**使用异步回调请求**
/**

cosXmlService.deleteObjectAsync(deleteObjectRequest, new CosXmlResultListener() {
@Override
public void onSuccess(CosXmlRequest cosXmlRequest, CosXmlResult cosXmlResult) {

Log.w("TEST","success");
}

@Override
public void onFail(CosXmlRequest cosXmlRequest, CosXmlClientException clientException, CosXmlServiceException

serviceException) {

String errorMsg = clientException != null ? clientException.toString() : serviceException.toString();
Log.w("TEST",errorMsg);
}
});

*/
```

删除多个文件

调用此接口可以在指定存储桶中批量删除文件，单次请求最大支持批量删除 1000 个 文件。具体步骤如下：

1. 调用 DeleteMultiObjectRequest（String, List）构造方法，实例化 DeleteMultiObjectRequest 对象。
2. 调用 CosXmlService 的 deleteMultiObject 方法，传入 DeleteMultiObjectRequest，返回 DeleteMultiObjectResult 对象。（或者调用 deleteMultiObjectAsync 方法，传入 DeleteMultiObjectRequest 和 CosXmlResultListener 进行异步回调操作）。

参数说明

参数名称	参数描述	类型	必填
bucket	存储桶名称(cos v5 的 bucket格式为：xxx-appid, 如 test-1253960454)	String	是
quiet	true：只返回删除报错的文件信息； false：返回每个文件的删除结果	Boolean	是
objectList	需要删除的文件路径列表	List	是
signDuration	签名的有效期，单位为秒	Long	是
checkHeaderListForSign	签名中需要验证的请求头	Set	否
checkParameterListForSing	签名中需要验证的请求参数	Set	否
cosXmlResultListener	上传结果回调	CosXmlResultListener	否

返回结果说明

通过 DeleteMultiObjectResult 对象的成员变量返回请求结果。

成员变量名称	变量说明	类型
httpCode	[200, 300)之间请求成功， 否则请求失败	Int

示例

```
String bucket = "bucket";
List<String> objectList = new ArrayList<String>();
objectList.add("/2/test.txt");

DeleteMultiObjectRequest deleteMultiObjectRequest = new DeleteMultiObjectRequest();
deleteMultiObjectRequest.setQuiet(quiet);
deleteMultiObjectRequest.setSign(signDuration,null,null);

//使用同步方法删除
try {
DeleteMultiObjectResult deleteMultiObjectResult =cosXmlService.deleteMultiObject(deleteMultiObjectRequest);

Log.w("TEST","success： " + deleteMultiObjectResult.deleteResult.toString());
} catch (CosXmlClientException e) {

Log.w("TEST","CosXmlClientException =" + e.toString());

} catch (CosXmlServiceException e) {

Log.w("TEST","CosXmlServiceException =" + e.toString());
}

//**使用异步回调请求**
/**

cosXmlService.deleteMultiObjectAsync(deleteMultiObjectRequest, new CosXmlResultListener() {
@Override
public void onSuccess(CosXmlRequest cosXmlRequest, CosXmlResult cosXmlResult) {

Log.w("TEST","success");
}

@Override
public void onFail(CosXmlRequest cosXmlRequest, CosXmlClientException clientException, CosXmlServiceException

serviceException) {

String errorMsg = clientException != null ? clientException.toString() : serviceException.toString();
Log.w("TEST",errorMsg);
}
});

*/
```

下载文件

调用此接口将指定Bucket 中的一个文件下载至本地。具体步骤如下：

1. 调用 `GetObjectRequest（String, String, String）` 构造方法，实例化 GetObjectRequest 对象。
2. 调用 CosXmlService 的 getObject 方法，传入 GetObjectRequest，返回 GetObjectResult 对象。（或者调用 getObjectAsync 方法，传入 GetObjectRequest 和 CosXmlResultListener 进行异步回调操作）。

参数说明

参数名称	参数描述	类型	必填
bucket	存储桶名称(cos v5 的 bucket格式为：xxx-appid, 如 test-1253960454)	String	是
cosPath	远端路径，即存储到 COS 上的绝对路径	String	是
savaPath	文件下载到本地文件夹的绝对路径	String	是
start	请求文件的开始位置	Long	否
end	请求文件的结束位置	Long	否
signDuration	签名的有效期，单位为秒	Long	是

参数名称	参数描述	类型	必填
checkHeaderListForSign	签名中需要验证的请求头	Set	否
checkParameterListForSing	签名中需要验证的请求参数	Set	否
qCloudProgressListener	下载进度回调	CosXmlProgressListener	否
cosXmlResultListener	上传结果回调	CosXmlResultListener	否

返回结果说明

通过 GetObjectResult 对象的成员变量返回请求结果。

成员变量名称	变量说明	类型
httpCode	[200, 300)之间请求成功， 否则请求失败	Int

示例

```
String bucket = "bucket";
String cosPath = "cosPath";
String savePath = "savePath";

GetObjectRequest getObjectRequest = GetObjectRequest(bucket, cosPath, savePath);

getObjectRequest.setSign(signDuration,null,null);
getObjectRequest.setProgressListener(new CosXmlProgressListener() {
@Override
public void onProgress(long progress, long max) {
float result = (float) (progress * 100.0/max);
Log.w("TEST","progress =" + (long)result + "%");
}
});

//使用同步方法下载
try {
GetObjectResult getObjectResult =cosXmlService.getObject(getObjectRequest);

Log.w("TEST","success：  " + getObjectResult.xCOSStorageClass);

} catch (CosXmlClientException e) {

Log.w("TEST","CosXmlClientException =" + e.toString());

} catch (CosXmlServiceException e) {

Log.w("TEST","CosXmlServiceException =" + e.toString());
}

//**使用异步回调请求**
/**

cosXmlService.getObjectAsync(getObjectRequest, new CosXmlResultListener() {
@Override
public void onSuccess(CosXmlRequest cosXmlRequest, CosXmlResult cosXmlResult) {

Log.w("TEST","success");
}

@Override
public void onFail(CosXmlRequest cosXmlRequest, CosXmlClientException clientException, CosXmlServiceException

serviceException) {

String errorMsg = clientException != null ? clientException.toString() : serviceException.toString();
Log.w("TEST",errorMsg);
}
});
```

```
*/
```

复制对象

调用此接口实现将一个文件从源路径复制到目标路径，建议文件大小 1 M 到 5 G，超过 5 G 的文件请使用分块上传 Upload - Copy 。具体步骤如下：

1. 调用 `CopyObjectRequest（String，String, CopySourceStruct）` 构造方法，实例化 CopyObjectRequest 对象。
2. 调用 CosXmlService 的 copyObject 方法，传入 CopyObjectRequest，返回 CopyObjectResult 对象。（或者调用 copyObjectAsync 方法，传入 CopyObjectRequest 和 CosXmlResultListener 进行异步回调操作）。

参数说明

参数名称	参数描述	类型	必填
bucket	存储桶名称(cos v5 的 bucket格式为：xxx-appid, 如 test-1253960454)	String	是
cosPath	目标路径，即存储到 COS 上的绝对路径	String	是
copySourceStruct	源路径结构体	CopySourceStruct	是
signDuration	签名的有效期，单位为秒	Long	是
checkHeaderListForSign	签名中需要验证的请求头	Set	否
checkParameterListForSing	签名中需要验证的请求参数	Set	否
cosXmlResultListener	上传结果回调	CosXmlResultListener	否

返回结果说明

通过 CopyObjectResult 对象的成员变量返回请求结果。

成员变量名称	变量说明	类型
copyObject	返回复制结果信息	CopyObject
httpCode	[200, 300)之间请求成功， 否则请求失败	Int

示例

```
String bucket = "bucket";
String cosPath = "cosPath";
CopyObjectRequest.CopySourceStruct copySourceStruct = new CopyObjectRequest.CopySourceStruct("源文件的appid",
"源文件的bucket", "源文件的region", "源文件的cosPath");

CopyObjectRequest copyObjectRequest = new CopyObjectRequest(bucket, cosPath, copySourceStruct);
copyObjectRequest.setSign(signDuration,null,null);

//使用同步方法
try {
CopyObjectResult copyObjectResult = cosXmlService.copyObject(copyObjectRequest);
//成功
Log.w("TEST","success:" + copyObjectResult.copyObject);

} catch (CosXmlClientException e) {

Log.w("TEST","CosXmlClientException =" + e.toString());

} catch (CosXmlServiceException e) {

Log.w("TEST","CosXmlServiceException =" + e.toString());
}

//**使用异步回调请求**
/**
```

```
cosXmlService.copyObjectAsync(copyObjectRequest, new CosXmlResultListener() {
@Override
public void onSuccess(CosXmlRequest request, CosXmlResult result) {

Log.w("TEST","success");
}

@Override
public void onFail(CosXmlRequest cosXmlRequest, CosXmlClientException clientException, CosXmlServiceException

serviceException) {

String errorMsg = clientException != null ? clientException.toString() : serviceException.toString();
Log.w("TEST",errorMsg);
}
});

*/
```

创建一个 Bucket

调用此接口将在指定账号下创建一个 Bucket。具体步骤如下：

1. 调用 `PutBucketRequest(String)` 构造方法，实例化 PutBucketRequest 对象。
2. 调用 CosXmlService 的 putBucket 方法，传入 PutBucketRequest，返回 PutBucketResult 对象。（或者调用 putBucketAsync 方法，传入 PutBucketRequest 和 CosXmlResultListener 进行异步回调操作）。

参数说明

参数名称	参数描述	类型	必填
bucket	存储桶名称(cos v5 的 bucket格式为：xxx-appid, 如 test-1253960454)	String	是
signDuration	签名的有效期，单位为秒	Long	是
checkHeaderListForSign	签名中需要验证的请求头	Set	否
checkParameterListForSing	签名中需要验证的请求参数	Set	否
cosXmlResultListener	上传结果回调	CosXmlResultListener	否

返回结果说明

通过 PutBucketResult 对象的成员变量返回请求结果。

成员变量名称	变量说明	类型
httpCode	[200, 300)之间请求成功， 否则请求失败	Int

示例

```
PutBucketRequest putBucketRequest = new PutBucketRequest(bucket);
putBucketRequest.setSign(signDuration,null,null);

//定义 Object 的 ACL 属性。有效值：private，public-read-write，public-read；默认值：private
putBucketRequest.setXCOSACL("private");

//赋予被授权者读的权限
ACLAccount readACLS = new ACLAccount();
readACLS.addACLAccount("OwnerUin", "SubUin");
putBucketRequest.setXCOSGrantRead(readACLS);

//赋予被授权者写的权限
ACLAccount writeACLS = new ACLAccount();
writeACLS.addACLAccount("OwnerUin", "SubUin");
putBucketRequest.setXCOSGrantRead(writeACLS);

//赋予被授权者读写的权限
```

```
ACLAccount writeandReadACLS = new ACLAccount();
writeandReadACLS.addACLAccount("OwnerUin", "SubUin");
putBucketRequest.setXCOSGrantRead(writeandReadACLS);

//使用同步方法
try {
PutBucketResult putBucketResult = cosXmlService.putBucket(putBucketRequest);
//成功
Log.w("TEST","success");

} catch (CosXmlClientException e) {

Log.w("TEST","CosXmlClientException =" + e.toString());

} catch (CosXmlServiceException e) {

Log.w("TEST","CosXmlServiceException =" + e.toString());
}

//**使用异步回调请求**
/**

cosXmlService.putBucketAsync(putBucketRequest, new CosXmlResultListener() {
@Override
public void onSuccess(CosXmlRequest request, CosXmlResult result) {

Log.w("TEST","success");
}

@Override
public void onFail(CosXmlRequest cosXmlRequest, CosXmlClientException clientException, CosXmlServiceException

serviceException) {

String errorMsg = clientException != null ? clientException.toString() : serviceException.toString();
Log.w("TEST",errorMsg);
}
});

*/
```

确认 Bucket 是否存在

调用此接口将确认指定存储桶是否存在。具体步骤如下：

1. 调用 HeadBucketRequest（String）构造方法，实例化 HeadBucketRequest 对象。
2. 调用 CosXmlService 的 headBucket 方法，传入 HeadBucketRequest，返回 HeadBucketResult 对象。（或者调用 headBucketAsync 方法，传入 HeadBucketRequest 和 CosXmlResultListener 进行异步回调操作）。

参数说明

参数名称	参数描述	类型	必填
bucket	存储桶名称(cos v5 的 bucket格式为：xxx-appid, 如 test-1253960454)	String	是
signDuration	签名的有效期，单位为秒	Long	是
checkHeaderListForSign	签名中需要验证的请求头	Set	否
checkParameterListForSing	签名中需要验证的请求参数	Set	否
cosXmlResultListener	上传结果回调	CosXmlResultListener	否

返回结果说明

通过 PutBucketResult 对象的成员变量返回请求结果。

成员变量名称	变量说明	类型

成员变量名称	变量说明	类型
httpCode	[200, 300)之间请求成功， 否则请求失败	Int

示例

```
HeadBucketRequest headBucketRequest = new HeadBucketRequest(bucket);
headBucketRequest.setSign(signDuration,null,null);

//使用同步方法
try {
HeadBucketResult headBucketResult = cosXmlService.headBucket(headBucketRequest);
//成功
Log.w("TEST","success");

} catch (CosXmlClientException e) {

Log.w("TEST","CosXmlClientException =" + e.toString());

} catch (CosXmlServiceException e) {

Log.w("TEST","CosXmlServiceException =" + e.toString());
}

//**使用异步回调请求**
/**

cosXmlService.headBucketAsync(headBucketRequest, new CosXmlResultListener() {
@Override
public void onSuccess(CosXmlRequest request, CosXmlResult result) {

Log.w("TEST","success");
}

@Override
public void onFail(CosXmlRequest cosXmlRequest, CosXmlClientException clientException, CosXmlServiceException

serviceException) {

String errorMsg = clientException != null ? clientException.toString() : serviceException.toString();
Log.w("TEST",errorMsg);
}
});

*/
```

列举 Bucket

调用此接口将列出该 Bucket 下的部分或者全部 Object。具体步骤如下：

1. 调用 `GetBucketRequest(String)` 构造方法，实例化 GetBucketRequest 对象。
2. 调用 CosXmlService 的 getBucket 方法，传入 GetBucketRequest，返回 GetBucketResult 对象。（或者调用 getBucketAsync 方法，传入 GetBucketRequest 和 CosXmlResultListener 进行异步回调操作）。

参数说明

参数名称	参数描述	类型	必填
bucket	存储桶名称(cos v5 的 bucket格式为：xxx-appid, 如 test-1253960454)	String	是
signDuration	签名的有效期，单位为秒	Long	是
checkHeaderListForSign	签名中需要验证的请求头	Set	否
checkParameterListForSing	签名中需要验证的请求参数	Set	否
cosXmlResultListener	上传结果回调	CosXmlResultListener	否

返回结果说明

通过 GetBucketResult 对象的成员变量返回请求结果。

成员变量名称	变量说明	类型
listBucket	保存 Get Bucket 请求结果的所有信息	ListBucket
httpCode	[200, 300)之间请求成功， 否则请求失败	Int

示例

```
GetBucketRequest getBucketRequest = new GetBucketRequest(bucket);
getBucketRequest.setSign(signDuration,null,null);

//前缀匹配，用来规定返回的文件前缀地址
getBucketRequest.setPrefix("prefix");

//单次返回最大的条目数量，默认 1000
getBucketRequest.setMaxKeys(100);

//定界符为一个符号，如果有 Prefix，
//则将 Prefix 到 delimiter 之间的相同路径归为一类，定义为 Common Prefix，
//然后列出所有 Common Prefix。如果没有 Prefix，则从路径起点开始
getBucketRequest.setDelimiter('c');

//使用同步方法
try {
GetBucketResult getBucketResult = cosXmlService.getBucket(getBucketRequest);
//成功
Log.w("TEST","success :" + getBucketResult.listBucket.toString());

} catch (CosXmlClientException e) {

Log.w("TEST","CosXmlClientException =" + e.toString());

} catch (CosXmlServiceException e) {

Log.w("TEST","CosXmlServiceException =" + e.toString());
}

//**使用异步回调请求**
/**

cosXmlService.getBucketAsync(getBucketRequest, new CosXmlResultListener() {
@Override
public void onSuccess(CosXmlRequest request, CosXmlResult result) {

Log.w("TEST","success");
}

@Override
public void onFail(CosXmlRequest cosXmlRequest, CosXmlClientException clientException, CosXmlServiceException

serviceException) {

String errorMsg = clientException != null ? clientException.toString() : serviceException.toString();
Log.w("TEST",errorMsg);
}
});

*/
```

删除 Bucket

调用此接口可以在指定账号下删除 Bucket，删除之前要求 Bucket 内的内容为空，只有删除了 Bucket 内的信息，才能删除 Bucket 本身。具体步骤如下：

1. 调用 `DeleteBucketRequest(String)` 构造方法，实例化 DeleteBucketRequest 对象。

2. 调用 CosXmlService 的 deleteBucket 方法，传入 DeleteBucketRequest，返回 DeleteBucketResult 对象。（或者调用 deleteBucketAsync 方法，传入 DeleteBucketRequest 和 CosXmlResultListener 进行异步回调操作）。

参数说明

参数名称	参数描述	类型	必填
bucket	存储桶名称(cos v5 的 bucket格式为：xxx-appid, 如 test-1253960454)	String	是
signDuration	签名的有效期，单位为秒	Long	是
checkHeaderListForSign	签名中需要验证的请求头	Set	否
checkParameterListForSing	签名中需要验证的请求参数	Set	否
cosXmlResultListener	上传结果回调	CosXmlResultListener	否

返回结果说明

通过 DeleteBucketResult 对象的成员变量返回请求结果。

成员变量名称	变量说明	类型
httpCode	[200, 300)之间请求成功， 否则请求失败	Int

示例

```
DeleteBucketRequest deleteBucketRequest = new DeleteBucketRequest(bucket);
deleteBucketRequest.setSign(signDuration,null,null);

//使用同步方法
try {
DeleteBucketResult deleteBucketResult = cosXmlService.deleteBucket(deleteBucketRequest);
//成功
Log.w("TEST","success");

} catch (CosXmlClientException e) {

Log.w("TEST","CosXmlClientException =" + e.toString());

} catch (CosXmlServiceException e) {

Log.w("TEST","CosXmlServiceException =" + e.toString());
}

//**使用异步回调请求**
/**

cosXmlService.deleteBucketAsync(deleteBucketRequest, new CosXmlResultListener() {
@Override
public void onSuccess(CosXmlRequest request, CosXmlResult result) {

Log.w("TEST","success");
}

@Override
public void onFail(CosXmlRequest cosXmlRequest, CosXmlClientException clientException, CosXmlServiceException

serviceException) {

String errorMsg = clientException != null ? clientException.toString() : serviceException.toString();
Log.w("TEST",errorMsg);
}
});

*/
```

设置 Bucket ACL

调用此接口可以指定 Bucket 的访问控制权限。具体步骤如下：

1. 调用 `PutBucketACLRequest(String)` 构造方法，实例化 PutBucketACLRequest 对象。
2. 调用 CosXmlService 的 putBucketACL 方法，传入 PutBucketACLRequest，返回 PutBucketACLResult 对象。（或者调用 putBucketACLAsync 方法，传入 PutBucketACLRequest 和 CosXmlResultListener 进行异步回调操作）。

参数说明

参数名称	参数描述	类型	必填
bucket	存储桶名称(cos v5 的 bucket格式为：xxx-appid, 如 test-1253960454)	String	是
xcosACL	设置 Bucket 访问权限，有效值为：private，public-read-write，public-read；默认值：private	String	否
xcosGrantRead	赋予被授权者读的权限	ACLAccount	否
xcosGrantWrite	赋予被授权者写的权限	ACLAccount	否
xcosGrantRead	赋予被授权者读写的权限	ACLAccount	否
signDuration	签名的有效期，单位为秒	Long	是
checkHeaderListForSign	签名中需要验证的请求头	Set	否
checkParameterListForSing	签名中需要验证的请求参数	Set	否
cosXmlResultListener	上传结果回调	CosXmlResultListener	否

返回结果说明

通过 DeleteBucketResult 对象的成员变量返回请求结果。

成员变量名称	变量说明	类型
httpCode	[200, 300)之间请求成功， 否则请求失败	Int

示例

```
PutBucketACLRequest putBucketACLRequest = new PutBucketACLRequest(bucket);
putBucketACLRequest.setSign(signDuration,null,null);

//设置 bucket 访问权限
putBucketACLRequest.setXCOSACL("public-read");

//赋予被授权者读的权限
ACLAccount readACLS = new ACLAccount();
readACLS.addACLAccount("OwnerUin", "SubUin");
putBucketRequest.setXCOSGrantRead(readACLS);

//赋予被授权者写的权限
ACLAccount writeACLS = new ACLAccount();
writeACLS.addACLAccount("OwnerUin", "SubUin");
putBucketRequest.setXCOSGrantRead(writeACLS);

//赋予被授权者读写的权限
ACLAccount writeandReadACLS = new ACLAccount();
writeandReadACLS.addACLAccount("OwnerUin", "SubUin");
putBucketRequest.setXCOSGrantRead(writeandReadACLS);

//使用同步方法
try {
PutBucketACLResult putBucketACLResult = cosXmlService.putBucketACL(putBucketACLRequest);
//成功
Log.w("TEST","success");

} catch (CosXmlClientException e) {

Log.w("TEST","CosXmlClientException =" + e.toString());

} catch (CosXmlServiceException e) {
```

```
Log.w("TEST","CosXmlServiceException =" + e.toString());
}


//**使用异步回调请求**
/**

cosXmlService.putBucketACLAsync(putBucketACLRequest, new CosXmlResultListener() {
@Override
public void onSuccess(CosXmlRequest request, CosXmlResult result) {

Log.w("TEST","success");
}

@Override
public void onFail(CosXmlRequest cosXmlRequest, CosXmlClientException clientException, CosXmlServiceException

serviceException) {

String errorMsg = clientException != null ? clientException.toString() : serviceException.toString();
Log.w("TEST",errorMsg);
}
});

*/
```

获取 Bucket ACL

调用此接口用来获取 Bucket 的 ACL。具体步骤如下：

1. 调用 `GetBucketACLRequest(String)` 构造方法，实例化 GetBucketACLRequest 对象。
2. 调用 CosXmlService 的 getBucketACL 方法，传入 GetBucketACLRequest，返回 GetBucketACLResult 对象。（或者调用 getBucketACLAsync 方法，传入 GetBucketACLRequest 和 CosXmlResultListener 进行异步回调操作）。

参数说明

参数名称	参数描述	类型	必填
bucket	存储桶名称(cos v5 的 bucket格式为：xxx-appid, 如 test-1253960454)	String	是
signDuration	签名的有效期，单位为秒	Long	是
checkHeaderListForSign	签名中需要验证的请求头	Set	否
checkParameterListForSing	签名中需要验证的请求参数	Set	否
cosXmlResultListener	上传结果回调	CosXmlResultListener	否

返回结果说明

通过 GetBucketACLResult 对象的成员变量返回请求结果。

成员变量名称	变量说明	类型
accessControlPolicy	被授权者信息与权限信息	AccessControlPolicy
httpCode	[200, 300)之间请求成功， 否则请求失败	Int

示例

```
GetBucketACLRequest getBucketACLRequest = new DeleteBucketRequest(bucket);
getBucketACLRequest.setSign(signDuration,null,null);

//使用同步方法
try {
GetBucketACLResult getBucketACLResult = cosXmlService.getBucketACL(getBucketACLRequest);
//成功
Log.w("TEST","success: " +getBucketACLResult.accessControlPolicy.toString() );
```

```
} catch (CosXmlClientException e) {

Log.w("TEST","CosXmlClientException =" + e.toString());

} catch (CosXmlServiceException e) {

Log.w("TEST","CosXmlServiceException =" + e.toString());
}

//**使用异步回调请求**
/**

cosXmlService.getBucketACLAsync(getBucketACLRequest, new CosXmlResultListener() {
@Override
public void onSuccess(CosXmlRequest request, CosXmlResult result) {

Log.w("TEST","success");
}

@Override
public void onFail(CosXmlRequest cosXmlRequest, CosXmlClientException clientException, CosXmlServiceException

serviceException) {

String errorMsg = clientException != null ? clientException.toString() : serviceException.toString();
Log.w("TEST",errorMsg);
}
});

*/
```

设置跨域访问配置

调用此接口将指定 Bucket 中设置跨域访问配置信息。具体步骤如下：

1. 调用 `PutBucketCORSRequest（String）` 构造方法，实例化 PutBucketCORSRequest 对象。
2. 调用 CosXmlService 的 putBucketCORS 方法，传入 PutBucketCORSRequest，返回 PutBucketCORSResult 对象。（或者调用 putBucketCORSAsync 方法，传入 PutBucketCORSRequest 和 CosXmlResultListener 进行异步回调操作）。

参数说明

参数名称	参数描述	类型	必填
bucket	存储桶名称(cos v5 的 bucket格式为：xxx-appid, 如 test-1253960454)	String	是
cORSRule	跨域访问配置信息	CORSConfiguration.CORSRule	是
signDuration	签名的有效期，单位为秒	Long	是
checkHeaderListForSign	签名中需要验证的请求头	Set	否
checkParameterListForSing	签名中需要验证的请求参数	Set	否
cosXmlResultListener	上传结果回调	CosXmlResultListener	否

返回结果说明

通过 PutBucketCORSResult 对象的成员变量返回请求结果。

成员变量名称	变量说明	类型
httpCode	[200, 300)之间请求成功， 否则请求失败	Int

示例

```
PutBucketCORSRequest putBucketCORSRequest = new PutBucketCORSRequest(bucket);
```

```
/**
CORSConfiguration.cORSRule: 跨域访问配置信息
corsRule.id： 配置规则的 ID
cORSRule.allowedOrigin: 允许的访问来源，支持通配符 *，格式为：协议://域名[:端口]如：http://imgcache.finance.cloud.tencent.com:80www.qq.com
corsRule.maxAgeSeconds: 设置 OPTIONS 请求得到结果的有效期
corsRule.allowedMethod: 允许的 HTTP 操作，如：GET，PUT，HEAD，POST，DELETE
corsRule.allowedHeader：在发送 OPTIONS 请求时告知服务端，接下来的请求可以使用哪些自定义的 HTTP 请求头部，支持通配符 *
corsRule.exposeHeader： 设置浏览器可以接收到的来自服务器端的自定义头部信息
*/
CORSConfiguration.CORSRule corsRule = new CORSConfiguration.CORSRule();

corsRule.id = "123";
corsRule.allowedOrigin = "http://imgcache.finance.cloud.tencent.com:80finance.cloud.tencent.com";
corsRule.maxAgeSeconds = "5000";

List<String> methods = new LinkedList<>();
methods.add("PUT");
methods.add("POST");
methods.add("GET");
corsRule.allowedMethod = methods;

List<String> headers = new LinkedList<>();
headers.add("host");
headers.add("content-type");
corsRule.allowedHeader = headers;

List<String> exposeHeaders = new LinkedList<>();
headers.add("x-cos-metha-1");
corsRule.exposeHeader = headers;

//设置跨域访问配置信息
putBucketCORSRequest.addCORSRule(corsRule);

putBucketCORSRequest.setSign(signDuration,null,null);

//使用同步方法
try {
PutBucketCORSResult putBucketCORSResult = cosXmlService.putBucketCORS(putBucketCORSRequest);
//成功
Log.w("TEST","success");

} catch (CosXmlClientException e) {

Log.w("TEST","CosXmlClientException =" + e.toString());

} catch (CosXmlServiceException e) {

Log.w("TEST","CosXmlServiceException =" + e.toString());
}

//**使用异步回调请求**
/**

cosXmlService.putBucketCORSAsync(putBucketCORSRequest, new CosXmlResultListener() {
@Override
public void onSuccess(CosXmlRequest request, CosXmlResult result) {

Log.w("TEST","success");
}

@Override
public void onFail(CosXmlRequest cosXmlRequest, CosXmlClientException clientException, CosXmlServiceException

serviceException) {

String errorMsg = clientException != null ? clientException.toString() : serviceException.toString();
Log.w("TEST",errorMsg);
}
});

*/
```

获取跨域访问配置

调用此接口将指定 Bucket 中获取跨域访问配置信息。具体步骤如下：

1. 调用 `GetBucketCORSRequest(String)` 构造方法，实例化 GetBucketCORSRequest 对象。
2. 调用 CosXmlService 的 getBucketCORS 方法，传入 GetBucketCORSRequest，返回 GetBucketCORSResult 对象。（或者调用 getBucketCORSAsync 方法，传入 GetBucketCORSRequest 和 CosXmlResultListener 进行异步回调操作）。

参数说明

参数名称	参数描述	类型	必填
bucket	存储桶名称(cos v5 的 bucket格式为：xxx-appid, 如 test-1253960454)	String	是
signDuration	签名的有效期，单位为秒	Long	是
checkHeaderListForSign	签名中需要验证的请求头	Set	否
checkParameterListForSing	签名中需要验证的请求参数	Set	否
cosXmlResultListener	上传结果回调	CosXmlResultListener	否

返回结果说明

通过 GetBucketCORSResult 对象的成员变量返回请求结果。

成员变量名称	类型	变量说明
cORSConfiguration	跨域资源共享配置的所有信息	CORSConfiguration
httpCode	[200, 300)之间请求成功， 否则请求失败	Int

示例

```
GetBucketCORSRequest getBucketCORSRequest = new GetBucketCORSRequest(bucket);
getBucketCORSRequest.setSign(signDuration,null,null);

//使用同步方法
try {
GetBucketCORSResult getBucketCORSResult = cosXmlService.getBucketCORS(getBucketCORSRequest);
//成功
Log.w("TEST","success :" + getBucketCORSResult.corsConfiguration.toString());

} catch (CosXmlClientException e) {

Log.w("TEST","CosXmlClientException =" + e.toString());

} catch (CosXmlServiceException e) {

Log.w("TEST","CosXmlServiceException =" + e.toString());
}

//**使用异步回调请求**
/**

cosXmlService.getBucketCORSAsync(getBucketCORSRequest, new CosXmlResultListener() {
@Override
public void onSuccess(CosXmlRequest request, CosXmlResult result) {

GetBucketCORSResult getBucketCORSResult = (GetBucketCORSResult)result;
Log.w("TEST","success:" + getBucketCORSResult.corsConfiguration.toString()););
}

@Override
public void onFail(CosXmlRequest cosXmlRequest, CosXmlClientException clientException, CosXmlServiceException

serviceException) {

String errorMsg = clientException != null ? clientException.toString() : serviceException.toString();
Log.w("TEST",errorMsg);
```

```
}
});

*/
```

删除跨域访问配置

调用此接口将指定 Bucket 中删除跨域访问配置信息.具体步骤如下：

1. 调用 DeleteBucketCORSRequest(String) 构造方法，实例化 DeleteBucketCORSRequest 对象。
2. 调用 CosXmlService 的 deleteBucketCORS 方法，传入 DeleteBucketCORSRequest，返回 DeleteBucketCORSResult 对象（或者调用 deleteBucketCORSAsync 方法，传入 DeleteBucketCORSRequest 和 CosXmlResultListener 进行异步回调操作）。

参数说明

参数名称	参数描述	类型	必填
bucket	存储桶名称(cos v5 的 bucket格式为：xxx-appid, 如 test-1253960454)	String	是
signDuration	签名的有效期，单位为秒	Long	是
checkHeaderListForSign	签名中需要验证的请求头	Set	否
checkParameterListForSing	签名中需要验证的请求参数	Set	否
cosXmlResultListener	上传结果回调	CosXmlResultListener	否

返回结果说明

通过 DeleteBucketCORSResult 对象的成员变量返回请求结果。

成员变量名称	变量说明	类型
httpCode	[200, 300)之间请求成功， 否则请求失败	Int

示例

```
DeleteBucketCORSRequest deleteBucketCORSRequest = new DeleteBucketCORSRequest(bucket);
deleteBucketCORSRequest.setSign(signDuration,null,null);

//使用同步方法
try {
DeleteBucketCORSResult deleteBucketCORSResult = cosXmlService.deleteBucketCORS(deleteBucketCORSRequest);
//成功
Log.w("TEST","success");

} catch (CosXmlClientException e) {

Log.w("TEST","CosXmlClientException =" + e.toString());

} catch (CosXmlServiceException e) {

Log.w("TEST","CosXmlServiceException =" + e.toString());
}

//**使用异步回调请求**
/**

cosXmlService.deleteBucketCORSAsync(deleteBucketCORSRequest, new CosXmlResultListener() {
@Override
public void onSuccess(CosXmlRequest request, CosXmlResult result) {

Log.w("TEST","success");
}

@Override
public void onFail(CosXmlRequest cosXmlRequest, CosXmlClientException clientException, CosXmlServiceException
```

```
serviceException) {

String errorMsg = clientException != null ? clientException.toString() : serviceException.toString();
Log.w("TEST",errorMsg);
}
});

*/
```

获取 Bucket 地域信息

调用此接口用于获取 Bucket 所在的地域信息。具体步骤如下：

1. 调用 `GetBucketLocationRequest(String)` 构造方法，实例化 GetBucketLocationRequest 对象。
2. 调用 CosXmlService 的 getBucketLocation 方法，传入 GetBucketLocationRequest，返回 GetBucketLocationResult 对象。（或者调用 getBucketLocationAsync 方法，传入 GetBucketLocationRequest 和 CosXmlResultListener 进行异步回调操作）。

参数说明

参数名称	参数描述	类型	必填
bucket	存储桶名称(cos v5 的 bucket格式为：xxx-appid, 如 test-1253960454)	String	是
signDuration	签名的有效期，单位为秒	Long	是
checkHeaderListForSign	签名中需要验证的请求头	Set	否
checkParameterListForSing	签名中需要验证的请求参数	Set	否
cosXmlResultListener	上传结果回调	CosXmlResultListener	否

返回结果说明

通过 GetBucketLocationResult 对象的成员变量返回请求结果。

成员变量名称	变量说明	类型
locationConstraint	Bucket 所在区域	LocationConstraint
httpCode	[200, 300)之间请求成功， 否则请求失败	Int

示例

```
GetBucketLocationRequest getBucketLocationRequest = new DeleteBucketCORSRequest(bucket);
getBucketLocationRequest.setSign(signDuration,null,null);

//使用同步方法
try {
GetBucketLocationResult getBucketLocationResult = cosXmlService.getBucketLocation(getBucketLocationRequest);
//成功
Log.w("TEST","success : " + getBucketLocationResult.LocationConstraint.location);

} catch (CosXmlClientException e) {

Log.w("TEST","CosXmlClientException =" + e.toString());

} catch (CosXmlServiceException e) {

Log.w("TEST","CosXmlServiceException =" + e.toString());
}

//**使用异步回调请求**
/**

cosXmlService.getBucketLocationAsync(getBucketLocationRequest, new CosXmlResultListener() {
@Override
public void onSuccess(CosXmlRequest request, CosXmlResult result) {
```

```
Log.w("TEST","success");
}

@Override
public void onFail(CosXmlRequest cosXmlRequest, CosXmlClientException clientException, CosXmlServiceException

serviceException) {

String errorMsg = clientException != null ? clientException.toString() : serviceException.toString();
Log.w("TEST",errorMsg);
}
});

*/
```

设置 Bucket 生命周期

调用此接口用于设置 Bucket 的生命周期信息。具体步骤如下：

1. 调用 `PutBucketLifecycleRequest(String)` 构造方法，实例化 PutBucketLifecycleRequest 对象。
2. 调用 CosXmlService 的 putBucketLifecycle 方法，传入 PutBucketLifecycleRequest，返回 PutBucketLifecycleResult 对象。（或者调用 putBucketLifecycleAsync 方法，传入 PutBucketLifecycleRequest 和 CosXmlResultListener 进行异步回调操作）。

参数说明

参数名称	参数描述	类型	必填
bucket	存储桶名称(cos v5 的 bucket格式为：xxx-appid, 如 test-1253960454)	String	是
rule	生命周期配置规则	LifecycleConfiguration.Rule	是
signDuration	签名的有效期，单位为秒	Long	是
checkHeaderListForSign	签名中需要验证的请求头	Set	否
checkParameterListForSing	签名中需要验证的请求参数	Set	否
cosXmlResultListener	上传结果回调	CosXmlResultListener	否

返回结果说明

通过 PutBucketLifecycleResult 对象的成员变量返回请求结果。

成员变量名称	变量说明	类型
httpCode	[200, 300)之间请求成功， 否则请求失败	Int

示例

```
PutBucketLifecycleRequest putBucketLifecycleRequest = new PutBucketLifecycleRequest(bucket);
putBucketLifecycleRequest.setSign(signDuration,null,null);

//声明周期配置规则信息
LifecycleConfiguration.Rule rule = new LifecycleConfiguration.Rule();
rule.id = "Lifecycle ID";
LifecycleConfiguration.Filter filter = new LifecycleConfiguration.Filter();
filter.prefix = "prefix/";
rule.filter = filter;
rule.status = "Enabled or Disabled";
LifecycleConfiguration.Transition transition = new LifecycleConfiguration.Transition();
transition.days = 100;
transition.storageClass = COSStorageClass.NEARLINE.getStorageClass();
putBucketLifecycleRequest.setRuleList(rule);

//使用同步方法
try {
PutBucketLifecycleResult putBucketLifecycleResult = cosXmlService.putBucketLifecycle(putBucketLifecycleRequest);
//成功
```

```
Log.w("TEST","success : " + getBucketLocationResult.region);

} catch (CosXmlClientException e) {

Log.w("TEST","CosXmlClientException =" + e.toString());

} catch (CosXmlServiceException e) {

Log.w("TEST","CosXmlServiceException =" + e.toString());
}

//**使用异步回调请求**
/**

cosXmlService.putBucketLifecycleAsync(putBucketLifecycleRequest, new CosXmlResultListener() {
@Override
public void onSuccess(CosXmlRequest request, CosXmlResult result) {

Log.w("TEST","success");
}

@Override
public void onFail(CosXmlRequest cosXmlRequest, CosXmlClientException clientException, CosXmlServiceException

serviceException) {

String errorMsg = clientException != null ? clientException.toString() : serviceException.toString();
Log.w("TEST",errorMsg);
}
});

*/
```

获取 Bucket 生命周期

调用此接口用于获取 Bucket 的生命周期信息。具体步骤如下：

1. 调用 `GetBucketLifecycleRequest(String)` 构造方法，实例化 GetBucketLifecycleRequest 对象。
2. 调用 CosXmlService 的 getBucketLifecycle 方法，传入 GetBucketLifecycleRequest，返回 GetBucketLifecycleResult 对象。（或者调用 getBucketLifecycleAsync 方法，传入 GetBucketLifecycleRequest 和 CosXmlResultListener 进行异步回调操作）。

参数说明

参数名称	参数描述	类型	必填
bucket	存储桶名称(cos v5 的 bucket格式为：xxx-appid, 如 test-1253960454)	String	是
signDuration	签名的有效期，单位为秒	Long	是
checkHeaderListForSign	签名中需要验证的请求头	Set	否
checkParameterListForSing	签名中需要验证的请求参数	Set	否
cosXmlResultListener	上传结果回调	CosXmlResultListener	否

返回结果说明

通过 getBucketLifecycle 对象的成员变量返回请求结果。

成员变量名称	变量说明	类型
lifecycleConfiguration	生命周期配置信息	LifecycleConfiguration
httpCode	[200, 300)之间请求成功， 否则请求失败	Int

示例

```
GetBucketLifecycleRequest getBucketLifecycleRequest = new DeleteBucketCORSRequest(bucket);
getBucketLifecycleRequest.setSign(signDuration,null,null);

//使用同步方法
try {
GetBucketLifecycleResult getBucketLifecycleResult = cosXmlService.getBucketLifecycle(getBucketLifecycleRequest);
//成功
Log.w("TEST","success : " + getBucketLifecycleResult.lifecycleConfiguration.toString());

} catch (CosXmlClientException e) {

Log.w("TEST","CosXmlClientException =" + e.toString());

} catch (CosXmlServiceException e) {

Log.w("TEST","CosXmlServiceException =" + e.toString());
}

//**使用异步回调请求**
/**

cosXmlService.getBucketLifecycleAsync(getBucketLifecycleResult, new CosXmlResultListener() {
@Override
public void onSuccess(CosXmlRequest request, CosXmlResult result) {

Log.w("TEST","success");
}

@Override
public void onFail(CosXmlRequest cosXmlRequest, CosXmlClientException clientException, CosXmlServiceException

serviceException) {

String errorMsg = clientException != null ? clientException.toString() : serviceException.toString();
Log.w("TEST",errorMsg);
}
});

*/
```

删除 Bucket 生命周期

调用此接口用于删除 Bucket 的生命周期信息。具体步骤如下：

1. 调用 `DeleteBucketLifecycleRequest(String)` 构造方法，实例化 DeleteBucketLifecycleRequest 对象。
2. 调用 CosXmlService 的 deleteBucketLifecycle 方法，传入 DeleteBucketLifecycleRequest，返回 DeleteBucketLifecycleResult 对象。（或者调用 deleteBucketLifecycleAsync 方法，传入 DeleteBucketLifecycleRequest 和 CosXmlResultListener 进行异步回调操作）。

参数说明

参数名称	参数描述	类型	必填
bucket	存储桶名称(cos v5 的 bucket格式为：xxx-appid, 如 test-1253960454)	String	是
signDuration	签名的有效期，单位为秒	Long	是
checkHeaderListForSign	签名中需要验证的请求头	Set	否
checkParameterListForSing	签名中需要验证的请求参数	Set	否
cosXmlResultListener	上传结果回调	CosXmlResultListener	否

返回结果说明

通过 DeleteBucketLifecycleResult 对象的成员变量返回请求结果。

成员变量名称	变量说明	类型

成员变量名称	变量说明	类型
httpCode	[200, 300)之间请求成功， 否则请求失败	Int

示例

```
DeleteBucketLifecycleRequest deleteBucketLifecycleRequest = new DeleteBucketCORSRequest(bucket);
deleteBucketLifecycleRequest.setSign(signDuration,null,null);

//使用同步方法
try {
DeleteBucketLifecycleResult deleteBucketCORSResult = cosXmlService.deleteBucketLifecycle(deleteBucketLifecycleRequest);
//成功
Log.w("TEST","success " ) ;

} catch (CosXmlClientException e) {

Log.w("TEST","CosXmlClientException =" + e.toString());

} catch (CosXmlServiceException e) {

Log.w("TEST","CosXmlServiceException =" + e.toString());
}

//**使用异步回调请求**
/**

cosXmlService.deleteBucketLifecycleAsync(deleteBucketLifecycleRequest, new CosXmlResultListener() {
@Override
public void onSuccess(CosXmlRequest request, CosXmlResult result) {

Log.w("TEST","success");
}

@Override
public void onFail(CosXmlRequest cosXmlRequest, CosXmlClientException clientException, CosXmlServiceException

serviceException) {

String errorMsg = clientException != null ? clientException.toString() : serviceException.toString();
Log.w("TEST",errorMsg);
}
});

*/
```

查询 Bucket 中正在上传的分块

调用此接口用于获取 Bucket 中正在进行中的分块上传，单次请求操作最多列出 1000 个正在进行中的分块上传。具体步骤如下：

1. 调用 `ListMultiUploadsRequest(String)` 构造方法，实例化 ListMultiUploadsRequest 对象。
2. 调用 CosXmlService 的 listMultiUploads 方法，传入 ListMultiUploadsRequest，返回 ListMultiUploadsResult 对象。（或者调用 listMultiUploadsAsync 方法，传入 ListMultiUploadsRequest 和 CosXmlResultListener 进行异步回调操作）。

参数说明

参数名称	参数描述	类型	必填
bucket	存储桶名称(cos v5 的 bucket格式为：xxx-appid, 如 test-1253960454)	String	是
signDuration	签名的有效期，单位为秒	Long	是
checkHeaderListForSign	签名中需要验证的请求头	Set	否
checkParameterListForSing	签名中需要验证的请求参数	Set	否
cosXmlResultListener	上传结果回调	CosXmlResultListener	否

返回结果说明

通过 ListMultiUploadsResult 对象的成员变量返回请求结果。

成员变量名称	变量说明	类型
listMultipartUploads	所有分块上传的信息	ListMultipartUploads
httpCode	[200, 300)之间请求成功， 否则请求失败	Int

示例

```
ListMultiUploadsRequest listMultiUploadsRequest = new ListMultiUploadsRequest(bucket);
listMultiUploadsRequest.setSign(signDuration,null,null);

//使用同步方法
try {
ListMultiUploadsResult listMultiUploadsResult = cosXmlService.listMultiUploads(listMultiUploadsRequest);
//成功
Log.w("TEST","success：  " + listMultiUploadsResult.listMultipartUploads.toString() ) ;

} catch (CosXmlClientException e) {

Log.w("TEST","CosXmlClientException =" + e.toString());

} catch (CosXmlServiceException e) {

Log.w("TEST","CosXmlServiceException =" + e.toString());
}

//**使用异步回调请求**
/**

cosXmlService.listMultiUploadsAsync(listMultiUploadsRequest, new CosXmlResultListener() {
@Override
public void onSuccess(CosXmlRequest request, CosXmlResult result) {

Log.w("TEST","success");
}

@Override
public void onFail(CosXmlRequest cosXmlRequest, CosXmlClientException clientException, CosXmlServiceException

serviceException) {

String errorMsg = clientException != null ? clientException.toString() : serviceException.toString();
Log.w("TEST",errorMsg);
}
});

*/
```

iOS SDK
快速入门

最近更新时间: 2025-01-15 17:01:00

开发准备

SDK 获取

对象存储服务的 iOS SDK 地址：XML iOS SDK 。 需要下载打包好的 Framework 格式的 SDK 可以从 realease 中选择需要的版本进行下载。

更多示例可参考 Demo： XML iOS SDK Demo 。

开发准备

- SDK 支持 iOS 8.0 及以上版本的系统；
- 手机必须要有网络（GPRS、3G 或 Wifi 网络等）；
- 从 COS V5 控制台 获取 APPID、SecretId、SecretKey。

> **说明：**
>
> 关于文章中出现的 SecretID、SecretKey、Bucket 等名称的含义和获取方式请参考：COS 术语信息

SDK 配置

SDK 导入

您可以通过 cocoapods 或者下载打包好的动态库的方式来集成 SDK。在这里我们推荐您使用 cocoapods 的方式来进行导入。

使用Cocoapods导入(推荐)

在Podfile文件中使用：

```
pod 'QCloudCOSXML'
```

使用打包好的动态库导入

将 **QCloudCOSXML.framework和QCloudCore.framework** 拖入到工程中：

并添加以下依赖库：

1. CoreTelephony
2. Foundation
3. SystemConfiguration
4. libstdc++.tbd

工程配置

在 Build Settings 中设置 Other Linker Flags，加入参数

```
-ObjC
-all_load
```

云平台对象存储 XML iOS 的 SDK 使用的是 HTTP 协议。为了在 iOS 系统上可以运行，您需要开启允许通过 HTTP 传输。 您可以通过以下两种方式开启允许通过 HTTP 传输：

- **手动设置方式：** 在工程 info.plist 文件中添加 App Transport Security Settings 类型，然后在 App Transport Security Settings 下添加 Allow Arbitrary Loads 类型 Boolean，值设为YES。
- **代码设置方式：** 您可以在集成 SDK 的 APP 的 info.plist 中需要添加如下代码：

```
<key>NSAppTransportSecurity</key>
<dict>
<key>NSExceptionDomains</key>
<dict>
<key>myqcloud.com</key>
<dict>
<key>NSIncludesSubdomains</key>
<true/>
<key>NSTemporaryExceptionAllowsInsecureHTTPLoads</key>
<true/>
</dict>
</dict>
</dict>
```

初始化

在使用 SDK 的功能之前，需要导入一些必要的头文件和进行一些初始化工作。

引入上传 SDK 的头文件：

```
QCloudCore.h,
QCloudCOSXML/QCloudCOSXML.h
```

另外，使用 SDK 操作前，首先要实例化一个云服务配置对象 *QCloudServiceConfiguration*，其次需要实例化 *QCloudCOSXMLService* 和 *QCloudCOSTransferManagerService* 对象。

方法原型

实例化 *QCloudServiceConfiguration* 对象：

```
QCloudServiceConfiguration* configuration = [QCloudServiceConfiguration new];
configuration.appID = @""//项目ID;
```

实例化 *QCloudCOSXMLService* 对象：

```
+ (QCloudCOSXMLService*) registerDefaultCOSXMLWithConfiguration:(QCloudServiceConfiguration*)configuration;
```

实例化 *QCloudCOSTransferManagerService* 对象：

```
+ (QCloudCOSTransferMangerService*) registerDefaultCOSTransferMangerWithConfiguration:(QCloudServiceConfiguration*)configuration;
```

QCloudServiceConfiguration 参数说明

参数名称	说明	类型	必填
appID	项目 ID，即 APP ID。	NSString *	是

初始化示例

下面用到的 APPID， SecretId， SecretKey 等可以从 COS v4 控制台 中获取。

```
//AppDelegate.m

- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary *)launchOptions {
QCloudServiceConfiguration* configuration = [QCloudServiceConfiguration new];
QCloudServiceConfiguration* configuration = [QCloudServiceConfiguration new];
configuration.appID = @"*****";
configuration.signatureProvider = self;
QCloudCOSXMLEndPoint* endpoint = [[QCloudCOSXMLEndPoint alloc] init];

NSString *region = @"REGION"; // 替换成用户的 Region
NSString *domain = @"DOMAIN.com"; // 替换成用户的 Domain

NSString *endpointSuffix = [NSString stringWithFormat:@"cos.%@.%@", region, domain];
endpoint.suffix = endpointSuffix;
endpoint.serviceName = domain;
configuration.endpoint = endpoint;
```

```
[QCloudCOSXMLService registerDefaultCOSXMLWithConfiguration:configuration];
[QCloudCOSTransferMangerService registerDefaultCOSTransferMangerWithConfiguration:configuration];

}
```

快速入门

这里演示的上传和下载的基本流程，更多细节可以参考 XML iOS SDK Demo。具体每一个接口如何使用请参照 Demo 中提供的单元测试文件。

> **注意：**
>
> 在进行这一步之前必须在 云平台控制台 上申请 COS 业务的 APPID。

STEP - 1 初始化

示例

注意：*QCloudServiceConfiguration* 的 *signatureProvider* 对象需要实现 *QCloudSignatureProvider* 协议。

```
//AppDelegate.m
- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary *)launchOptions {
QCloudServiceConfiguration* configuration = [QCloudServiceConfiguration new];
configuration.appID = @"******";
configuration.signatureProvider = self;
QCloudCOSXMLEndPoint* endpoint = [[QCloudCOSXMLEndPoint alloc] init];
endpoint.regionName = @"ap-beijing";//服务地域名称，可用的地域请参考注释
configuration.endpoint = endpoint;

[QCloudCOSXMLService registerDefaultCOSXMLWithConfiguration:configuration];
[QCloudCOSTransferMangerService registerDefaultCOSTransferMangerWithConfiguration:configuration];
}
```

示例

```
//AppDelegate.m
- (void) signatureWithFields:(QCloudSignatureFields*)fileds
request:(QCloudBizHTTPRequest*)request
urlRequest:(NSURLRequest*)urlRequst
compelete:(QCloudHTTPAuthentationContinueBlock)continueBlock
{
//实现签名的过程，我们推荐在服务器端实现签名的过程，具体请参考接下来的"生成签名"这一章。
}
```

STEP - 2 上传文件

这里，假设您已经申请了自己业务 Bucket。事实上，SDK 所有的请求对应了相应的 Request 类，只要生成相应的请求，设置好相应的属性，然后将请求交给 QCloudCOSXMLService 对象，就可以完成相应的动作。其中，request 的 body 部分传入需要上传的文件在本地的 URL（NSURL* 类型）。

上传文件的接口需要用到签名来进行身份认证，发错的请求会自动向初始化时指定的遵循 QCloudSignatureProvider 协议的对象去请求签名。签名如何生成可以参考下一章节中的生成签名。

> **注意：**
>
> URL 所对应的文件在上传过程中是不能进行变更的，否则会导致出错。

示例

```
QCloudCOSXMLUploadObjectRequest* put = [QCloudCOSXMLUploadObjectRequest new];

NSURL* url = /*文件的URL*/;
put.object = @"文件名.jpg";
put.bucket = @"test-123456789";
put.body = url;
[put setSendProcessBlock:^(int64_t bytesSent, int64_t totalBytesSent, int64_t totalBytesExpectedToSend) {
NSLog(@"upload %lld totalSend %lld aim %lld", bytesSent, totalBytesSent, totalBytesExpectedToSend);
}];
[put setFinishBlock:^(id outputObject, NSError* error) {
```

```
}];
[[QCloudCOSTransferMangerService defaultCOSTransferManager] UploadObject:put];
```

QCloudCOSXMLUploadObjectRequest 参数含义

参数名称	说明	类型	必填
Object	上传文件（对象）的文件名，也是对象的key	NSString *	是
bucket	存储桶名,可在COSV5控制台上面看到，格式为- ,例如 testBucket-1253653367	NSString *	是
body	需要上传的文件的路径。填入NSURL * 类型变量	BodyType	是
storageClass	对象的存储级别	QCloudCOSStorageClass	是
cacheControl	RFC 2616 中定义的缓存策略	NSString *	否
contentDisposition	RFC 2616中定义的文件名称	NSString *	否
expect	当使用expect=@"100-Continue"时，在收到服务端确认后才会发送请求内容	NSString *	否
expires	RFC 2616中定义的过期时间	NSString *	否
initMultipleUploadFinishBlock	如果该 request 产生了分片上传的请求，那么在分片上传初始化完成后，会通过这个 block 来回调，可以在该回调 block 中获取分片完成后的 bucket， key， uploadID，以及用于后续上传失败后恢复上传的ResumeData。	block	否
accessControlList	定义 Object 的 ACL 属性。有效值：private，public-read-write，public-read；默认值：private	NSString *	否
grantRead	赋予被授权者读的权限。格式： id=" ",id=" "；当需要给子账户授权时，id="qcs::cam::uin/:uin/"，当需要给根账户授权时，id="qcs::cam::uin/:uin/" 其中 OwnerUin 指的是根账户的 ID，而 SubUin 指的是子账户的 ID	NSString *	否
grantWrite	授予被授权者写的权限。格式同上。	NSString *	否
grantFullControl	授予被授权者读写权限。格式同上。	NSString *	否

STEP - 3 下载文件

示例

```
QCloudGetObjectRequest* request = [QCloudGetObjectRequest new];
//设置下载的路径 URL，如果设置了，文件将会被下载到指定路径中
request.downloadingURL = [NSURL URLWithString:QCloudTempFilePathWithExtension(@"downding")];
request.object = @“你的Object-Key”;
request.bucket = @"test-123456789";
[request setFinishBlock:^(id outputObject, NSError \*error) {
//additional actions after finishing
}];
[request setDownProcessBlock:^(int64_t bytesDownload, int64_t totalBytesDownload, int64_t totalBytesExpectedToDownload) {
//下载过程中的进度
}];
[[QCloudCOSXMLService defaultCOSXML] GetObject:request];
```

生成签名

SDK 中的请求需要用到签名，以确认访问的用户的身份，也保障了访问的安全性。当签名不正确时，大部分 COS 的服务将无法访问并且返回 403 错误。在 SDK 中可以生成签名，每个请求会向 QCloudServiceConfiguration 对象中的signatureProvider 对象来请求生成签名。我们可以将负责生成签名的对象在一开始赋值给 signatureProvider，该生成签名的对象需要遵循 QCloudSignatureProvider 协议，并实现生成签名的方法：

```
- (void) signatureWithFields:(QCloudSignatureFields* )fileds
request:(QCloudBizHTTPRequest* )request
urlRequest:(NSURLRequest* )urlRequst
compelete:(QCloudHTTPAuthentationContinueBlock)continueBlock
```

最佳实践：接入CAM系统实现临时签名

虽然在本地提供了永久的 SecretId 和 SecretKey 来生成签名的接口，但请注意，将永久的 SecretId 和SecretKey 存储在本地是非常危险的行为，容易造成泄露引起不必要的损失。因此基于安全性的考虑，建议您在服务器端实现签名的过程。

推荐您在自己的签名服务器内接入云平台的 CAM（Cloud Access Manager， 访问管理）来实现整个签名流程。

至于如何搭建签名服务器接入CAM系统，可以参考快速搭建移动应用传输服务。

签名服务器接入 CAM 系统后，当客户端向签名服务器端请求签名时，签名服务器端会向 CAM 系统请求临时证书，然后返回给客户端。CAM 系统会根据您的永久 SecretId 和 SecretKey 来生成临时的 Secret ID, Secret Key 和临时Token 来生成签名，可以最大限度地提高安全性。终端收到这些临时密钥的信息后，通过它们构建一个QCloudCredential对象，然后通过这个QCloudCredentail对象生成QCloudAuthentationCreator，最后通过使用这个Creator来生成包含签名信息的QCloudSignature对象。具体的操作可以参考下面的例子：

```
- (void) signatureWithFields:(QCloudSignatureFields*)fileds
request:(QCloudBizHTTPRequest*)request
urlRequest:(NSURLRequest*)urlRequst
compelete:(QCloudHTTPAuthentationContinueBlock)continueBlock
{

/*向签名服务器请求临时的 Secret ID,Secret Key,Token*/
QCloudCredential* credential = [QCloudCredential new];
credential.secretID = @"从 CAM 系统获取的临时 Secret ID";
credential.secretKey = @"从 CAM 系统获取的临时 Secret Key";
credential.token = @"从 CAM 系统返回的 Token，为会话 ID"
credential.expiretionDate = /*签名过期时间*/
QCloudAuthentationCreator* creator = [[QCloudAuthentationCreator alloc] initWithCredential:credential];
QCloudSignature* signature = [creator signatureForCOSXMLRequest:request];
continueBlock(signature, nil);
}
```

在终端使用永久密钥生成签名（不推荐，有极大的泄密风险）

```
- (void) signatureWithFields:(QCloudSignatureFields*)fileds
request:(QCloudBizHTTPRequest*)request
urlRequest:(NSMutableURLRequest*)urlRequst
compelete:(QCloudHTTPAuthentationContinueBlock)continueBlock
{

QCloudCredential* credential = [QCloudCredential new];
credential.secretID = @"永久的SecretID";
credential.secretKey = @"永久的SecretKey";
QCloudAuthentationV5Creator* creator = [[QCloudAuthentationV5Creator alloc] initWithCredential:credential];
QCloudSignature* signature = [creator signatureForData:urlRequst];
continueBlock(signature, nil);
}
```

使用脚手架工具管理异步签名过程

其实到这一步，您已经可以生成签名正常使用 SDK 里面的接口了。但为了方便您实现临时签名，从服务器端获取tempSecretKey 等临时签名需要的信息，我们提供了脚手架工具可供使用。您可以依照前面的代码来生成签名，也可以通过我们的脚手架工具 QCloudCredentailFenceQueue 来方便地获取临时签名。
QCloudCredentailFenceQueue 提供了栅栏机制，也就是说您使用 QCloudCredentailFenceQueue 获取签名的话，所有需要获取签名的请求会等待签名完成后再执行，免去了自己管理异步过程。使用 QCloudCredentailFenceQueue，我们需要先生成一个实例。

```
//AppDelegate.m
//AppDelegate需遵循QCloudCredentailFenceQueueDelegate协议
//
- (BOOL)application:(UIApplication * )application didFinishLaunchingWithOptions:(NSDictionary * )launchOptions {
// init step
self.credentialFenceQueue = [QCloudCredentailFenceQueue new];
self.credentialFenceQueue.delegate = self;
return YES;
}
```

然后调用 QCloudCredentailFenceQueue 的类需要遵循 QCloudCredentailFenceQueueDelegate 并实现协议内定义的方法：

```
- (void) fenceQueue:(QCloudCredentailFenceQueue * )queue requestCreatorWithContinue:(QCloudCredentailFenceQueueContinue)continueBlock
```

当通过 QCloudCredentailFenceQueue 去获取签名时，所有需要签名的 SDK 里的请求都会等待该协议定义的方法内拿到了签名所需的参数并生成有效的签名后执行。请看以下例子：

```
//AppDelegate.m
- (void) fenceQueue:(QCloudCredentailFenceQueue * )queue requestCreatorWithContinue:(QCloudCredentailFenceQueueContinue)continueBlock
{
QCloudCredential* credential = [QCloudCredential new];
//在这里可以同步过程从服务器获取临时签名需要的secretID,secretKey,expiretionDate和token参数
credential.secretID = @"****";
credential.secretKey = @"****";
credential.experationDate = [NSDate dateWithTimeIntervalSince1970:1504183628];
credential.token = @"****";
QCloudAuthentationV5Creator* creator = [[QCloudAuthentationV5Creator alloc] initWithCredential:credential];
continueBlock(creator, nil);
}
```

至此，就可以通过我们提供的脚手架工具来生成临时签名了。您也可以自己去实现具体的签名过程。

接口文档

最近更新时间: 2025-01-15 17:01:00

在接口文档中，我们假设您已经完成了初始化的过程。接口文档重点在于列出详细的接口列表，并且举例如何使用。查询时建议Control+F搜索到想要查询的接口，然后看我们给出的接口简单说明，复制示例到您的工程中运行。

如果需要更多的功能，或者不明白返回的参数是什么意义，建议直接查看代码里的注释，可以三指轻按、Force-touch重按或者将鼠标停留在变量上，按Control+Command+D查看它的释义。

Service 操作

列出所有Bucket - Get Service

Get Service 接口是用来获取请求者名下的所有存储空间列表（Bucket list）。

返回结果QCloudListAllMyBucketsResult参数说明

参数名称	描述	类型
owner	存储桶持有者的信息	QCloudOwner *
buckets	所有的bucket信息	NSArray *

示例

```
QCloudGetServiceRequest* request = [[QCloudGetServiceRequest alloc] init];
[request setFinishBlock:^(QCloudListAllMyBucketsResult* result, NSError* error) {
//
}];
[[QCloudCOSXMLService defaultCOSXML] GetService:request];
```

存储桶操作

列举存储桶内的内容

方法原型

进行存储桶操作之前，我们需要导入头文件 QCloudCOSXML/QCloudCOSXML.h。在此之前您需要完成前文中的 STEP-1 初始化操作。先生成一个 QCloudGetBucketRequest 实例，然后填入一些需要的额外限制条件，通过并获得内容。具体步骤如下：

1. 实例化 QCloudGetBucketRequest，填入需要的参数。
2. 调用 QCloudCOSXMLService 对象中的 GetBucket 方法发出请求。
3. 从回调的 finishBlock 中的 QCloudListBucketResult 获取具体内容。

QCloudGetBucketRequest 参数说明

参数名称	描述	类型	必填
bucket	存储桶名,可在COSV5控制台上面看到，格式为- ,例如 testBucket-1253653367	NSString *	是
region	前缀匹配，用来规定返回的文件前缀地址	NSString *	否
delimiter	定界符为一个符号，如果有 Prefix，则将 Prefix 到 delimiter 之间的相同路径归为一类，定义为 Common Prefix，然后列出所有 Common Prefix。如果没有 Prefix，则从路径起点开始	NSString *	否
encodingType	规定返回值的编码方式，可选值：url	NSString *	否
marker	默认以 UTF-8 二进制顺序列出条目，所有列出条目从 marker 开始	NSString *	否
maxKeys	单次返回的最大条目数量，默认 1000	int	否

示例

```
QCloudGetBucketRequest* request = [QCloudGetBucketRequest new];
request.bucket = @“testBucket-123456789“;
request.maxKeys = 1000;
[request setFinishBlock:^(QCloudListBucketResult * result, NSError* error) {
//additional actions after finishing
}];
[[QCloudCOSXMLService defaultCOSXML] GetBucket:request];
```

获取存储桶的 ACL（Access Control List）

方法原型

进行存储桶操作之前，需要导入头文件 QCloudCOSXML/QCloudCOSXML.h。在此之前您需要完成前文中的 STEP-1 初始化操作。先生成一个 QCloudGetBucketACLRequest 实例，然后填入一些需要的额外限制条件，通过并获得内容。具体步骤如下：

1. 实例化 QCloudGetBucketACLRequest，填入获取 ACL 的存储桶。
2. 调用 QCloudCOSXMLService 对象中的 GetBucketACL 方法发出请求。
3. 从回调的 finishBlock 中的 QCloudACLPolicy 获取具体内容。

QCloudGetBucketACLRequest 参数说明

参数名称	描述	类型	必填
bucket	存储桶名,可在COSV5控制台上面看到，格式为-,例如 testBucket-1253653367	NSString *	是

返回结果 QCloudACLPolicy 参数说明

参数名称	描述	类型
owner	存储桶持有者的信息	QCloudACLOwner *
list	accessControl被授权者与权限的信息	NSArray *

示例

```
QCloudGetBucketACLRequest* getBucketACl = [QCloudGetBucketACLRequest new];
getBucketACl.bucket = @"testbucket-123456789";
[getBucketACl setFinishBlock:^(QCloudACLPolicy * _Nonnull result, NSError * _Nonnull error) {
//QCloudACLPolicy中包含了 Bucket 的 ACL 信息。
}];

[[QCloudCOSXMLService defaultCOSXML] GetBucketACL:getBucketACl];
```

设置存储桶的 ACL(Access Control List)

方法原型

进行存储桶操作之前，需要导入头文件 QCloudCOSXML/QCloudCOSXML.h。在此之前您需要完成前文中的 STEP-1 初始化操作。先生成一个 QCloudPutBucketACLRequest 实例，然后填入一些需要的额外限制条件，通过并获得内容。具体步骤如下：

1. 实例化 QCloudPutBucketACLRequest，填入需要设置的存储桶，然后根据设置值的权限类型分别填入不同的参数。
2. 调用 QCloudCOSXMLService 对象中的 PutBucketACL 方法发出请求。
3. 从回调的 finishBlock 中的获取设置是否成功，并做设置成功后的一些额外动作。

QCloudPutBucketACLRequest 参数说明

参数名称	描述	类型	必填
bucket	存储桶名,可在COSV5控制台上面看到，格式为- ,例如 testBucket-1253653367	NSString *	是
accessControlList	定义 Object 的 ACL 属性。有效值：private，public-read-write，public-read；默认值：private	NSString *	否
grantRead	赋予被授权者读的权限。格式： id=" ",id=" "； 当需要给子账户授权时，id="qcs::cam::uin/:uin/"， 当需要给根账户授权时，id="qcs::cam::uin/:uin/" 其中 OwnerUin 指的是根账户的 ID，而 SubUin 指的是子账户的 ID	NSString *	否

参数名称	描述	类型	必填
grantWrite	授予被授权者写的权限。格式同上。	NSString *	否
grantFullControl	授予被授权者读写权限。格式同上。	NSString *	否

示例

```
QCloudPutBucketACLRequest* putACL = [QCloudPutBucketACLRequest new];
NSString* appID = @"您的 APP ID";
NSString *ownerIdentifier = [NSString stringWithFormat:@"qcs::cam::uin/%@:uin/%@", appID, appID];
NSString *grantString = [NSString stringWithFormat:@"id=\"%@\"",ownerIdentifier];
putACL.grantFullControl = grantString;
putACL.bucket = @"testBucket-123456789";
[putACL setFinishBlock:^(id outputObject, NSError *error) {
//error occucs if error != nil
}];
[[QCloudCOSXMLService defaultCOSXML] PutBucketACL:putACL];
```

获取存储桶的 CORS(跨域访问)设置

方法原型

进行存储桶操作之前，需要导入头文件 QCloudCOSXML/QCloudCOSXML.h。在此之前您需要完成前文中的 STEP-1 初始化操作。先生成一个 QCloudPutBucketCORSRequest 实例，然后填入一些需要的额外限制条件，通过并获得内容。具体步骤如下：

1. 实例化 QCloudPutBucketCORSRequest，填入需要获取 CORS 的存储桶。
2. 调用 QCloudCOSXMLService 对象中的 GetBucketCORS 方法发出请求。
3. 从回调的 finishBlock 中获取结果。结果封装在了 QCloudCORSConfiguration 对象中，该对象的 rules 属性是一个数组，数组里存放着一组 QCloudCORSRule，具体的 CORS 设置就封装在 QCloudCORSRule 对象里。

QCloudPutBucketCORSRequest 参数说明

参数名称	描述	类型	必填
bucket	存储桶名,可在COSV5控制台上面看到，格式为- ,例如 testBucket-1253653367	NSString *	是

返回结果 QCloudCORSConfiguration 参数说明

参数名称	描述	类型
rules	放置 CORS 的数组, 数组内容为 QCloudCORSRule 实例	NSArray `*`

QCloudCORSRule 参数说明

参数名称	描述	类型
identifier	配置规则的 ID	NSString *
allowedMethod	允许的 HTTP 操作，枚举值：GET，PUT，HEAD，POST，DELETE	NSArray `*`
allowedOrigin	允许的访问来源，支持通配符 *，格式为：协议://域名[:端口]如：`http://imgcache.finance.cloud.tencent.com:80www.qq.com`	NSString *
allowedHeader	在发送 OPTIONS 请求时告知服务端，接下来的请求可以使用哪些自定义的 HTTP 请求头部，支持通配符 *	NSArray `*`
maxAgeSeconds	设置 OPTIONS 请求得到结果的有效期	int
exposeHeader	设置浏览器可以接收到的来自服务器端的自定义头部信息	NSString *

示例

```
QCloudGetBucketCORSRequest* corsReqeust = [QCloudGetBucketCORSRequest new];
corsReqeust.bucket = @"testBucket-123456789";
```

```
[corsReqeust setFinishBlock:^(QCloudCORSConfiguration * _Nonnull result, NSError * _Nonnull error) {
//CORS设置封装在result中。
}];

[[QCloudCOSXMLService defaultCOSXML] GetBucketCORS:corsReqeust];
```

设置存储桶的 CORS（跨域访问）

方法原型

进行存储桶操作之前，需要导入头文件 QCloudCOSXML/QCloudCOSXML.h。在此之前您需要完成前文中的 STEP-1 初始化操作。先生成一个 QCloudPutBucketCORSRequest 实例，然后填入一些需要的额外限制条件，通过并获得内容。具体步骤如下：

1. 实例化 QCloudPutBucketCORSRequest，设置存储桶，并且将需要的 CORS 装入 QCloudCORSRule 中，如果有多组 CORS 设置，可以将多个 QCloudCORSRule 放在一个 NSArray 里，然后将该数组填入 QCloudCORSConfiguration的rules 属性里，放在 request 中。
2. 调用 QCloudCOSXMLService 对象中的 PutBucketCORS 方法发出请求。
3. 从回调的 finishBlock 中的获取设置成功与否（error 是否为空），并且做一些设置后的额外动作。

QCloudPutBucketCORSRequest 参数说明

参数名称	描述	类型	必填
bucket	存储桶名,可在COSV5控制台上面看到，格式为- ,例如 testBucket-1253653367	NSString *	是
corsConfiguration	封装了 CORS 的具体参数	QCloudCORSConfiguration *	是

QCloudCORSConfiguration 参数说明

参数名称	描述	类型
rules	放置 CORS 的数组, 数组内容为 QCloudCORSRule 实例	NSArray *

QCloudCORSRule 参数说明

参数名称	描述	类型
identifier	配置规则的ID	NSString *
allowedMethod	允许的 HTTP 操作，枚举值：GET，PUT，HEAD，POST，DELETE	NSArray *
allowedOrigin	允许的访问来源，支持通配符 *，格式为：协议://域名[:端口]如：`http://imgcache.finance.cloud.tencent.com:80www.qq.com`	NSString *
allowedHeader	在发送 OPTIONS 请求时告知服务端，接下来的请求可以使用哪些自定义的 HTTP 请求头部，支持通配符 *	NSArray *
maxAgeSeconds	设置 OPTIONS 请求得到结果的有效期	int
exposeHeader	设置浏览器可以接收到的来自服务器端的自定义头部信息	NSString *

示例

```
QCloudPutBucketCORSRequest* putCORS = [QCloudPutBucketCORSRequest new];
QCloudCORSConfiguration* cors = [QCloudCORSConfiguration new];

QCloudCORSRule* rule = [QCloudCORSRule new];
rule.identifier = @"sdk";
rule.allowedHeader = @[@"origin",@"host",@"accept",@"content-type",@"authorization"];
rule.exposeHeader = @"ETag";
rule.allowedMethod = @[@"GET",@"PUT",@"POST", @"DELETE", @"HEAD"];
rule.maxAgeSeconds = 3600;
rule.allowedOrigin = @"*";

cors.rules = @[rule];
```

```
putCORS.corsConfiguration = cors;
putCORS.bucket = @"testBucket-123456789";
[putCORS setFinishBlock:^(id outputObject, NSError *error) {
if (!error) {
//success
}
}];
[[QCloudCOSXMLService defaultCOSXML] PutBucketCORS:putCORS];
```

获取存储桶的地域信息

方法原型

进行存储桶操作之前，需要导入头文件 QCloudCOSXML/QCloudCOSXML.h。在此之前您需要完成前文中的 STEP-1 初始化操作。先生成一个 QCloudGetBucketLocationRequest 实例，然后填入一些需要的额外限制条件，通过并获得内容。具体步骤如下：

1. 实例化 QCloudGetBucketLocationRequest，填入 Bucket 名。
2. 调用 QCloudCOSXMLService 对象中的 GetBucketLocation 方法发出请求。
3. 从回调的 finishBlock 中的获取具体内容。

QCloudGetBucketLocationRequest 参数说明

参数名称	描述	类型	必填
bucket	存储桶名,可在COSV5控制台上面看到，格式为- ,例如 testBucket-1253653367	NSString *	是

返回结果 QCloudBucketLocationConstraint 参数说明

参数名称	描述	类型
locationConstraint	说明 Bucket 所在区域	NSString*

示例

```
QCloudGetBucketLocationRequest* locationReq = [QCloudGetBucketLocationRequest new];
locationReq.bucket = @"testBucket-123456789";
__block QCloudBucketLocationConstraint* location;
[locationReq setFinishBlock:^(QCloudBucketLocationConstraint * _Nonnull result, NSError * _Nonnull error) {
location = result;
}];
[[QCloudCOSXMLService defaultCOSXML] GetBucketLocation:locationReq];
```

删除存储桶 CORS 设置

方法原型

进行存储桶操作之前，需要导入头文件 QCloudCOSXML/QCloudCOSXML.h。在此之前您需要完成前文中的 STEP-1 初始化操作。先生成一个 QCloudDeleteBucketCORSRequest 实例，然后填入一些需要的额外限制条件，通过并获得内容。具体步骤如下：

1. 实例化 QCloudDeleteBucketCORSRequest，填入需要的参数。
2. 调用 QCloudCOSXMLService 对象中的方法发出请求。
3. 从回调的 finishBlock 中的获取具体内容。

QCloudDeleteBucketCORSRequest 参数说明

参数名称	描述	类型	必填
bucket	存储桶名,可在COSV5控制台上面看到，格式为- ,例如 testBucket-1253653367	NSString *	是

示例

```
QCloudDeleteBucketCORSRequest* deleteCORS = [QCloudDeleteBucketCORSRequest new];
deleteCORS.bucket = @"testBucket-123456789";
[deleteCORS setFinishBlock:^(id outputObject, NSError *error) {
//success if error == nil
```

```
}];
[[QCloudCOSXMLService defaultCOSXML] DeleteBucketCORS:deleteCORS];
```

查询 Bucket 中正在进行的分块上传

方法原型

进行存储桶操作之前，需要导入头文件 QCloudCOSXML/QCloudCOSXML.h。在此之前您需要完成前文中的 STEP-1 初始化操作。先生成一个 QCloudListBucketMultipartUploadsRequest 实例，然后填入一些需要的额外限制条件，通过并获得内容。具体步骤如下：

1. 实例化 QCloudListBucketMultipartUploadsRequest，填入需要的参数，如返回结果的前缀、编码方式等。
2. 调用 QCloudCOSXMLService 对象中的 ListBucketMultipartUploads 方法发出请求。
3. 从回调的 finishBlock 中的获取具体内容。

QCloudListBucketMultipartUploadsRequest 参数说明

参数名称	描述	类型	必填
bucket	存储桶名,可在COSV5控制台上面看到，格式为- ,例如 testBucket-1253653367	NSString *	是
prefix	限定返回的 Object key 必须以 Prefix 作为前缀。注意使用 prefix 查询时，返回的 key 中仍会包含 Prefix	NSString *	否
delimiter	定界符为一个符号，如果有 Prefix，则将 Prefix 到 delimiter 之间的相同路径归为一类，定义为 Common Prefix，然后列出所有 Common Prefix。如果没有 Prefix，则从路径起点开始	NSString *	否
encodingType	规定返回值的编码方式，可选值:url	NSString *	否
keyMarker	列出条目从该 key 值开始	NSString *	否
uploadIDMarker	列出条目从该 UploadId 值开始	int	否
maxUploads	设置最大返回的 multipart 数量，合法值 1 到 1000	int	否

返回结果 QCloudListMultipartUploadsResult 参数说明

参数名称	描述	类型
bucket	存储桶名,可在COSV5控制台上面看到，格式为- ,例如 testBucket-1253653367	NSString *
prefix	限定返回的 Object key 必须以 Prefix 作为前缀。注意使用 prefix 查询时，返回的 key 中仍会包含 Prefix	NSString *
delimiter	定界符为一个符号，如果有 Prefix，则将 Prefix 到 delimiter 之间的相同路径归为一类，定义为 Common Prefix，然后列出所有 Common Prefix。如果没有 Prefix，则从路径起点开始	NSString *
encodingType	规定返回值的编码方式，可选值：url	NSString *
keyMarker	列出条目从该 key 值开始	NSString *
maxUploads	设置最大返回的 multipart 数量，合法值 1 到 1000	int
uploads	所有已经上传的分片信息	NSArray*

示例

```
QCloudListBucketMultipartUploadsRequest* uploads = [QCloudListBucketMultipartUploadsRequest new];
uploads.bucket = @"testBucket-123456789";
uploads.maxUploads = 100;
__block NSError* resulError;
__block QCloudListMultipartUploadsResult* multiPartUploadsResult;
[uploads setFinishBlock:^(QCloudListMultipartUploadsResult* result, NSError *error) {
multiPartUploadsResult = result;
localError = error;
```

```
}];
[[QCloudCOSXMLService defaultCOSXML] ListBucketMultipartUploads:uploads];
```

Head Bucket

Head Bucket 请求可以确认该 Bucket 是否存在，是否有权限访问。Head 的权限与 Read 一致。当该 Bucket 存在时，返回200；当该 Bucket 无访问权限时，返回 403；当该 Bucket 不存在时，返回 404。

QCloudHeadBucketRequest 参数说明

参数名称	描述	类型	必填
bucket	存储桶名,可在COSV5控制台上面看到，格式为-,例如 testBucket-1253653367	NSString *	是

示例

```
QCloudHeadBucketRequest* request = [QCloudHeadBucketRequest new];
request.bucket = @"testBucket-123456789";
[request setFinishBlock:^(id outputObject, NSError* error) {
//设置完成回调。如果没有error，则可以正常访问bucket。如果有error，可以从error code和messasge中获取具体的失败原因。
}];
[[QCloudCOSXMLService defaultCOSXML] HeadBucket:request];
```

Put Bucket Tagging

QCloudPutBucketTaggingRequest参数说明

参数名称	描述	类型	必填
bucket	存储桶名,可在COSV5控制台上面看到，格式为- ,例如 testBucket-1253653367	NSString*	是
taggings	存放需要设置的tag数组	QCloudBucketTagging	是

示例

```
QCloudPutBucketTaggingRequest* putTagging = [QCloudPutBucketTaggingRequest new];
QCloudBucketTagging* tagging = [QCloudBucketTagging new];
QCloudBucketTag* tag = [QCloudBucketTag new];
tag.key = @"tag的key";
tag.value = @"tag的值";
tagging.tagset = @[tag];

putTagging.bucket = @"testBucket-123456789";
putTagging.taggings = tagging;

[putTagging setFinishBlock:^(id outputObject, NSError *error) {
//完成回调
}];
[[QCloudCOSXMLService defaultCOSXML] PutBucketTagging:putTagging];
```

Delete Bucket Tagging

QCloudDeleteBucketTaggingRequest 参数说明

参数名称	描述	类型	必填
bucket	存储桶名,可在COSV5控制台上面看到，格式为- ,例如 testBucket-1253653367	NSString*	是

示例

```
QCloudDeleteBucketTaggingRequest* request = [QCloudDeleteBucketTaggingRequest new];
request.bucket = @"testBucket-123456789";
[request setFinishBlock:^(id outputObject, NSError* error) {
//删除完成回调
}];
[[QCloudCOSXMLService defaultCOSXML] DeleteBucketTagging:request];
```

Get Bucket Tagging

QCloudGetBucketTaggingRequest 参数说明

参数名称	描述	类型	必填
bucket	存储桶名,可在COSV5控制台上面看到，格式为- ,例如 testBucket-1253653367	NSString*	是

返回结果 QCloudBucketTagging 参数说明

参数名称	描述	类型
tagest	装有bucket的tag的数组	NSArray*

示例

```
QCloudGetBucketTaggingRequest* request = [QCloudGetBucketTaggingRequest new];
request.bucket = @"testBucket-123456789" ;

[request setFinishBlock:^(QCloudBucketTagging* result, NSError* error) {
//设置完成回调
}];
[[QCloudCOSXMLService defaultCOSXML] GetBucketTagging:request];
```

Put Bucket Lifecycle

COS 支持用户以生命周期配置的方式来管理 Bucket 中 Object 的生命周期。生命周期配置包含一个或多个将应用于一组对象规则的规则集 (其中每个规则为 COS 定义一个操作)。 这些操作分为以下两种：

转换操作：定义对象转换为另一个存储类的时间。例如，您可以选择在对象创建 30 天后将其转换为 STANDARD_IA (IA，适用于不常访问) 存储类别。 过期操作：指定 Object 的过期时间。COS 将会自动为用户删除过期的 Object。

Put Bucket Lifecycle 用于为 Bucket 创建一个新的生命周期配置。如果该 Bucket 已配置生命周期，使用该接口创建新的配置的同时则会覆盖原有的配置。

参数说明

参数名称	描述	类型	必填
bucket	存储桶名,可在COSV5控制台上面看到，格式为-,例如 testBucket-1253653367	NSString*	是
lifeCycle	生命周期配置	QCloudLifecycleConfiguration*	是

QCloudLifecycleConfiguration 参数说明

参数名称	描述	类型	必填
rules	规则描述集合的数组	NSArray *	是

QCloudLifecycleRule 参数说明

参数名称	描述	类型	必填
identifier	用于唯一地标识规则，长度不能超过 255 个字符	NSString*	是
filter	Filter 用于描述规则影响的 Object 集合	QCloudLifecycleRuleFilter*	
status	指明规则是否启用，枚举值：Enabled，Disabled	QCloudLifecycleStatue	是
abortIncompleteMultipartUpload	设置允许分片上传保持运行的最长时间	QCloudLifecycleAbortIncompleteMultipartUpload	否
transition	规则转换属性，对象何时转换被转换为 Standard_IA 或 Nearline	QCloudLifecycleTransition*	否
expiration	规则过期属性	QCloudLifecycleExpiration*	否
noncurrentVersionExpiration	指明非当前版本对象何时过期	QCloudLifecycleExpiration*	否
noncurrentVersionTransition	指明非当前版本对象何时转换被转换为 STANDARD_IA 或 NEARLINE	QCloudNoncurrentVersionExpiration*	否

示例

```
QCloudPutBucketLifecycleRequest* request = [QCloudPutBucketLifecycleRequest new];
request.bucket = @"填入bucket名";
__block QCloudLifecycleConfiguration* configuration = [[QCloudLifecycleConfiguration alloc] init];
QCloudLifecycleRule* rule = [[QCloudLifecycleRule alloc] init];
rule.identifier = @"identifier";
rule.status = QCloudLifecycleStatueEnabled;
QCloudLifecycleRuleFilter* filter = [[QCloudLifecycleRuleFilter alloc] init];
filter.prefix = @"0";
rule.filter = filter;

QCloudLifecycleTransition* transition = [[QCloudLifecycleTransition alloc] init];
transition.days = 100;
transition.storageClass = QCloudCOSStorageNearline;
rule.transition = transition;
request.lifeCycle = configuration;
request.lifeCycle.rules = @[rule];
[request setFinishBlock:^(id outputObject, NSError* error) {
//设置完成回调
}];
[[QCloudCOSXMLService defaultCOSXML] PutBucketLifecycle:request];
```

Get Bucket Lifecycle

返回结果 QCloudLifecycleConfiguration 参数说明

与 Put Bucket Lifecycle 接口中的QCloudLifecycleConfiguration一致。

示例

```
QCloudGetBucketLifecycleRequest* request = [QCloudGetBucketLifecycleRequest new];
request.bucket = @"testBucket-123456789";
[request setFinishBlock:^(QCloudLifecycleConfiguration* result,NSError* error) {
//设置完成回调
}];
[[QCloudCOSXMLService defaultCOSXML] GetBucketLifecycle:request];
```

Delete Bucket Lifecycle

返回结果 QCloudLifecycleConfiguration 参数说明

与 Put Bucket Lifecycle 接口中的QCloudLifecycleConfiguration一致。

示例

```
QCloudDeleteBucketLifeCycleRequest* request = [[QCloudDeleteBucketLifeCycleRequest alloc ] init];
request.bucket = @"testBucket-123456789";
[request setFinishBlock:^(QCloudLifecycleConfiguration* result, NSError* error) {
//设置完成回调
}];
[[QCloudCOSXMLService defaultCOSXML] DeleteBucketLifeCycle:request];
```

Put Bucket Versioning

Put Bucket Versioning 接口实现启用或者暂停存储桶的版本控制功能。请注意这是一个不可逆的接口，开启以后不可撤销。

QCloudPutBucketVersioningRequest 参数说明

参数名称	描述	类型	必填
bucket	存储桶名,可在COSV5控制台上面看到，格式为- ,例如 testBucket-1253653367	NSString*	是
configuration	版本控制的具体信息	QCloudBucketVersioningConfiguration*	是

QCloudBucketVersioningConfiguration 参数说明

参数名称	描述	类型	必填

参数名称	描述	类型	必填
status	说明版本是否开启，枚举值：Suspended\Enabled	QCloudCOSBucketVersioningStatus	是

示例

```
QCloudPutBucketVersioningRequest* request = [[QCloudPutBucketVersioningRequest alloc] init];
request.bucket = @"testBucket-123456789";
QCloudBucketVersioningConfiguration* configuration = [[QCloudBucketVersioningConfiguration alloc] init];
request.configuration = configuration;
configuration.status = QCloudCOSBucketVersioningStatusEnabled;
[request setFinishBlock:^(id outputObject, NSError* error) {
//设置完成回调
}];
[[QCloudCOSXMLService defaultCOSXML] PutBucketVersioning:request];
```

Get Bucket Versioning

返回结果 QCloudBucketVersioningConfiguration 参数说明

参数名称	描述	类型
status	说明版本是否开启，枚举值：Suspended\Enabled	QCloudCOSBucketVersioningStatus

示例

```
QCloudGetBucketVersioningRequest* request = [[QCloudGetBucketVersioningRequest alloc] init];
request.bucket = @"testBucket-123456789";
[request setFinishBlock:^(QCloudBucketVersioningConfiguration* result, NSError* error) {
//设置完成回调
}];
[[QCloudCOSXMLService defaultCOSXML] GetBucketVersioning:request];
```

Put Bucket Replication

Put Bucket Replication 请求用于向开启版本管理的存储桶添加 replication 配置。如果存储桶已经拥有 replication 配置，那么该请求会替换现有配置。

QCloudPutBucketReplicationRequest参数说明

参数名称	描述	类型	必填
bucket	存储桶名,可在COSV5控制台上面看到，格式为- ,例如 testBucket-1253653367	NSString*	是
configuation	QCloudBucketReplicationConfiguation*	是	

注意： 使用该接口存储桶必须已经开启版本管理，版本管理详细请参见 Put Bucket Versioning。

返回结果 参数说明

示例

```
QCloudPutBucketReplicationRequest* request = [[QCloudPutBucketReplicationRequest alloc] init];
request.bucket = @"source-bucket";
QCloudBucketReplicationConfiguation* configuration = [[QCloudBucketReplicationConfiguation alloc] init];
configuration.role = [NSString identifierStringWithID:@"uin" :@"uin"];
QCloudBucketReplicationRule* rule = [[QCloudBucketReplicationRule alloc] init];

rule.identifier = @"identifier";
rule.status = QCloudQCloudCOSXMLStatusEnabled;

QCloudBucketReplicationDestination* destination = [[QCloudBucketReplicationDestination alloc] init];
//qcs:id/64295985640140800:cos:[region]:appid/[AppId]:[bucketname]
NSString* destinationBucket = @"destinationBucket";
NSString* region = @"destinationRegion"
destination.bucket = [NSString stringWithFormat:@"qcs:id/64295985640140800:cos:%@:appid/%@:%@",@"region",@"appid",@"destinationBucket"];
rule.destination = destination;
configuration.rule = @[rule];
request.configuation = configuration;
```

```
[request setFinishBlock:^(id outputObject, NSError* error) {
//设置完成回调
}];
[[QCloudCOSXMLService defaultCOSXML] PutBucketRelication:request];
```

Get Bucket Replication

Get Bucket Replication 接口请求实现读取存储桶中用户跨区域复制配置信息。

返回结果 QCloudBucketReplicationConfiguation 参数说明

参数名称	描述	类型
role	发起者身份标识，格式为：qcs::cam::uin/:uin/	NSString*
rule	具体配置信息，最多支持 1000 个，所有策略只能指向一个目标存储桶	NSArray *

QCloudBucketReplicationRule 参数说明

参数名称	描述	类型
status	标志Rule是否生效	QCloudQCloudCOSXMLStatus
identifier	用来标注具体rule的名称	NSString*
prefix	前缀匹配策略，不可重叠，重叠返回错误。前缀匹配根目录为空	NSString*
destination	目标存储桶信息	QCloudBucketReplicationDestination*

示例

```
QCloudGetBucketReplicationRequest* request = [[QCloudGetBucketReplicationRequest alloc] init];
request.bucket = @"testBucket-123456789";

[request setFinishBlock:^(QCloudBucketReplicationConfiguation* result, NSError* error) {
//设置完成回调
}];
[[QCloudCOSXMLService defaultCOSXML] GetBucketReplication:request];
```

Delete Bucket Replication

Delete Bucket Replication 接口请求实现删除存储桶中用户跨区域复制配置。

参数说明

返回结果 参数说明

示例

```
QCloudDeleteBucketReplicationRequest* request = [[QCloudDeleteBucketReplicationRequest alloc] init];
request.bucket = @"testBucket-123456789";
[request setFinishBlock:^(id outputObject, NSError* error) {
//设置完成回调
}];
[[QCloudCOSXMLService defaultCOSXML] DeleteBucketReplication:request];
```

文件操作

在 COS 中，每个文件就是一个 Object（对象）。对文件的操作，其实也就是对对象的操作。

简单上传 (Put Object)

简单上传仅限于小文件（20MB以下）。简单上传支持从内存中上传文件。

QCloudPutObjectRequest 参数说明

参数名称	说明	类型	必填

参数名称	说明	类型	必填
Object	上传文件（对象）的文件名，也是对象的key	NSString *	是
bucket	存储桶名,可在COSV5控制台上面看到，格式为- ,例如 testBucket-1253653367	NSString *	是
body	如果文件存放在硬盘中，这里是需要上传的文件的路径，填入NSURL * 类型变量。如果文件存放在内存中，则这里可以填入包含文件二进制数据的NSData * 类型变量	BodyType	是
storageClass	对象的存储级别	QCloudCOSStorageClass	是
cacheControl	RFC 2616 中定义的缓存策略	NSString *	否
contentDisposition	RFC 2616中定义的文件名称	NSString *	否
expect	当使用expect=@"100-Continue"时，在收到服务端确认后才会发送请求内容	NSString *	否
expires	RFC 2616中定义的过期时间	NSString *	否
initMultipleUploadFinishBlock	如果该 request 产生了分片上传的请求，那么在分片上传初始化完成后，会通过这个 block 来回调，可以在该回调 block 中获取分片完成后的 bucket， key， uploadID，以及用于后续上传失败后恢复上传的ResumeData。	block	否
accessControlList	定义 Object 的 ACL 属性。有效值：private，public-read-write，public-read；默认值：private	NSString *	否
grantRead	赋予被授权者读的权限。格式： id=" ",id=" "；当需要给子账户授权时，id="qcs::cam::uin/:uin/"，当需要给根账户授权时，id="qcs::cam::uin/:uin/" 其中 OwnerUin 指的是根账户的 ID，而 SubUin 指的是子账户的 ID	NSString *	否
grantWrite	授予被授权者写的权限。格式同上。	NSString *	否
grantFullControl	授予被授权者读写权限。格式同上。	NSString *	否

示例

```
QCloudPutObjectRequest* put = [QCloudPutObjectRequest new];
put.object = @"object-name";
put.bucket = @"bucket-12345678";
put.body = [@"testFileContent" dataUsingEncoding:NSUTF8StringEncoding];
[put setFinishBlock:^(id outputObject, NSError *error) {
//完成回调
if (nil == error) {
//成功
}
}];
[[QCloudCOSXMLService defaultCOSXML] PutObject:put];
```

查询对象的 ACL（Access Control List）

方法原型

进行文件操作之前，需要导入头文件 QCloudCOSXML/QCloudCOSXML.h。在此之前您需要完成前文中的 STEP-1 初始化操作。先生成一个 QCloudGetObjectACLRequest 实例，然后填入一些需要的额外限制条件，通过并获得内容。具体步骤如下：

1. 实例化 QCloudGetObjectACLRequest，填入存储桶的名称，和需要查询对象的名称。
2. 调用 QCloudCOSXMLService 对象中的 GetObjectACL 方法发出请求。
3. 从回调的 finishBlock 中的获取的 QCloudACLPolicy 对象中获取封装好的 ACL 的具体信息。

QCloudGetObjectACLRequest 参数说明

参数名称	描述	类型	必填
bucket	存储桶名,可在COSV5控制台上面看到，格式为- ,例如 testBucket-1253653367	NSString *	是
object	对象名	NSString *	是

示例

```
request.bucket = self.aclBucket;
request.object = @"对象的名称";
request.bucket = @"testBucket-123456789"
__block QCloudACLPolicy* policy;
[request setFinishBlock:^(QCloudACLPolicy * _Nonnull result, NSError * _Nonnull error) {
policy = result;
}];
[[QCloudCOSXMLService defaultCOSXML] GetObjectACL:request];
```

设置对象的 ACL（Access Control List）

方法原型

进行对象操作之前，需要导入头文件 QCloudCOSXML/QCloudCOSXML.h。在此之前您需要完成前文中的 STEP-1 初始化操作。先生成一个 QCloudPutObjectACLRequest 实例，然后填入一些需要的额外限制条件，通过并获得内容。具体步骤如下：

1. 实例化 QCloudPutObjectACLRequest，填入存储桶名，和一些额外需要的参数，如授权的具体信息等。
2. 调用 QCloudCOSXMLService 对象中的方法发出请求。
3. 从回调的 finishBlock 中获取设置的完成情况，若 error 为空，则设置成功。

QCloudPutObjectACLRequest 参数说明

参数名称	描述	类型	必填
bucket	存储桶名,可在COSV5控制台上面看到，格式为-,例如 testBucket-1253653367	NSString *	是
object	对象名	NSString *	是
accessControlList	定义 Object 的 ACL 属性。有效值：private，public-read-write，public-read；默认值：private	NSString *	否
grantRead	赋予被授权者读的权限。格式： id=" ",id=" "； 当需要给子账户授权时，id="qcs::cam::uin/:uin/"， 当需要给根账户授权时，id="qcs::cam::uin/:uin/" 其中 OwnerUin 指的是根账户的 ID，而 SubUin 指的是子账户的 ID	NSString *	否
grantWrite	授予被授权者写的权限。格式同上。	NSString *	否
grantFullControl	授予被授权者读写权限。格式同上。	NSString *	否

示例

```
QCloudPutObjectACLRequest* request = [QCloudPutObjectACLRequest new];
request.object = @"需要设置 ACL 的对象名";
request.bucket = @"testBucket-123456789";
NSString *ownerIdentifier = [NSString stringWithFormat:@"qcs::cam::uin/%@:uin/%@",self.appID, self.appID];
NSString *grantString = [NSString stringWithFormat:@"id=\"%@\"",ownerIdentifier];
request.grantFullControl = grantString;
__block NSError* localError;
[request setFinishBlock:^(id outputObject, NSError *error) {
localError = error;
}];

[[QCloudCOSXMLService defaultCOSXML] PutObjectACL:request];
```

下载文件

方法原型

进行文件操作之前，需要导入头文件 QCloudCOSXML/QCloudCOSXML.h。在此之前您需要完成前文中的 STEP-1 初始化操作。先生成一个实例，然后填入一些需要的额外限制条件，通过并获得内容。具体步骤如下：

1. 实例化，填入需要的参数。
2. 调用 QCloudCOSXMLService 对象中的方法发出请求。
3. 从回调的 finishBlock 中的获取具体内容。

参数说明

参数名称	描述	类型	必填

参数名称	描述	类型	必填
bucket	存储桶名,可在COSV5控制台上面看到，格式为-,例如 testBucket-1253653367	NSString *	是
object	对象名	NSString *	是
range	RFC 2616 中定义的指定文件下载范围，以字节（bytes）为单位	NSString *	否
ifModifiedSince	如果文件修改时间晚于指定时间，才返回文件内容。否则返回 412 (not modified)	NSString *	否
responseContentType	设置响应头部中的 Content-Type 参数	NSString *	否
responseContentLanguage	设置响应头部中的 Content-Language 参数	NSString *	否
responseContentExpires	设置响应头部中的 Content-Expires 参数	NSString *	否
responseCacheControl	设置响应头部中的 Cache-Control 参数	NSString *	否
responseContentDisposition	设置响应头部中的 Content-Disposition 参数。	NSString *	否
responseContentEncoding	设置响应头部中的 Content-Encoding 参数	NSString *	否

示例

```
QCloudGetObjectRequest* request = [QCloudGetObjectRequest new];
//设置下载的路径 URL，如果设置了，文件将会被下载到指定路径中
request.downloadingURL = [NSURL URLWithString:QCloudTempFilePathWithExtension(@"downding")];
request.object = @“你的 Object-Key”;
request.bucket = @"testBucket-123456789";
[request setFinishBlock:^(id outputObject, NSError *error) {
//additional actions after finishing
}];
[request setDownProcessBlock:^(int64_t bytesDownload, int64_t totalBytesDownload, int64_t totalBytesExpectedToDownload) {
//下载过程中的进度
}];
[[QCloudCOSXMLService defaultCOSXML] GetObject:request];
```

Object 跨域访问配置的预请求

方法原型

进行文件操作之前，需要导入头文件 QCloudCOSXML/QCloudCOSXML.h。在此之前您需要完成前文中的 STEP-1 初始化操作。先生成一个 QCloudOptionsObjectRequest 实例，然后填入一些需要的额外限制条件，通过并获得内容。具体步骤如下：

1. 实例化 QCloudOptionsObjectRequest，填入需要设置的对象名、存储桶名、模拟跨域访问请求的 http 方法和模拟跨域访问允许的访问来源。
2. 调用 QCloudCOSXMLService 对象中的方法发出请求。
3. 从回调的 finishBlock 中的获取具体内容。

QCloudOptionsObjectRequest 参数说明

参数名称	描述	类型	必填
object	对象名	NSString *	是
bucket	存储桶名,可在COSV5控制台上面看到，格式为- ,例如 testBucket-1253653367	NSString *	是
accessControlRequestMethod	模拟跨域访问的请求HTTP方法	NSArray *	是
origin	模拟跨域访问允许的访问来源，支持通配符 *，格式为：协议://域名[:端口]如：`http://imgcache.finance.cloud.tencent.com:80www.qq.com`	NSString *	是
allowedHeader	在发送 OPTIONS 请求时告知服务端，接下来的请求可以使用哪些自定义的 HTTP 请求头部，支持通配符 *	NSArray *	否

示例

```
QCloudOptionsObjectRequest* request = [[QCloudOptionsObjectRequest alloc] init];
request.bucket =@"存储桶名";
request.origin = @"*";
request.accessControlRequestMethod = @"get";
request.accessControlRequestHeaders = @"host";
request.object = @"对象名";
__block id resultError;
[request setFinishBlock:^(id outputObject, NSError* error) {
resultError = error;
}];

[[QCloudCOSXMLService defaultCOSXML] OptionsObject:request];
```

删除单个对象

方法原型

进行文件操作之前，需要导入头文件 QCloudCOSXML/QCloudCOSXML.h。在此之前您需要完成前文中的 STEP-1 初始化操作。先生成一个 QCloudDeleteObjectRequest 实例，然后填入一些需要的额外限制条件，通过并获得内容。具体步骤如下：

1. 实例化 QCloudDeleteObjectRequest，填入需要的参数。
2. 调用 QCloudCOSXMLService 对象中的方法发出请求。
3. 从回调的 finishBlock 中的获取具体内容。

QCloudDeleteObjectRequest 参数说明

参数名称	类型	必填	描述
object	对象名	NSString *	是
bucket	存储桶名,可在COSV5控制台上面看到，格式为-,例如 testBucket-1253653367	NSString *	是

示例

```
QCloudDeleteObjectRequest* deleteObjectRequest = [QCloudDeleteObjectRequest new];
deleteObjectRequest.bucket = @"testBucket-123456789";
deleteObjectRequest.object = @"对象名";

__block NSError* resultError;
[deleteObjectRequest setFinishBlock:^(id outputObject, NSError *error) {
resultError = error;
}];
[[QCloudCOSXMLService defaultCOSXML] DeleteObject:deleteObjectRequest];
```

删除多个对象

方法原型

进行文件操作之前，需要导入头文件 QCloudCOSXML/QCloudCOSXML.h。在此之前您需要完成前文中的 STEP-1 初始化操作。先生成一个 QCloudDeleteMultipleObjectRequest 实例，然后填入一些需要的额外限制条件，通过并获得内容。具体步骤如下：

1. 实例化 QCloudDeleteMultipleObjectRequest，填入需要的参数。
2. 调用 QCloudCOSXMLService 对象中的方法发出请求。
3. 从回调的 finishBlock 中的获取具体内容。

QCloudDeleteMultipleObjectRequest 参数说明

参数名称	描述	类型	必填
object	对象名	NSString *	是
deleteObjects	封装了需要批量删除的多个对象的信息	QCloudDeleteInfo *	是

QCloudDeleteInfo参数说明

参数名称	描述	类型	必填
objects	存放需要删除对象信息的数组	NSArray *	是

QCloudDeleteObjectInfo 参数说明

参数名称	描述	类型	必填
key	对象名	NSString *	是

示例

```
QCloudDeleteMultipleObjectRequest* delteRequest = [QCloudDeleteMultipleObjectRequest new];
delteRequest.bucket = @"testBucket-123456789";

QCloudDeleteObjectInfo* deletedObject0 = [QCloudDeleteObjectInfo new];
deletedObject0.key = @"第一个对象名";

QCloudDeleteObjectInfo* deleteObject1 = [QCloudDeleteObjectInfo new];
deleteObject1.key = @"第二个对象名";

QCloudDeleteInfo* deleteInfo = [QCloudDeleteInfo new];
deleteInfo.quiet = NO;
deleteInfo.objects = @[ deletedObject0,deleteObject2];

delteRequest.deleteObjects = deleteInfo;

__block NSError* resultError;
[delteRequest setFinishBlock:^(QCloudDeleteResult* outputObject, NSError *error) {
localError = error;
deleteResult = outputObject;
}];

[[QCloudCOSXMLService defaultCOSXML] DeleteMultipleObject:delteRequest];
```

初始化分片上传

方法原型

进行文件操作之前，需要导入头文件 QCloudCOSXML/QCloudCOSXML.h。在此之前您需要完成前文中的 STEP-1 初始化操作。先生成一个 QCloudInitiateMultipartUploadRequest 实例，然后填入一些需要的额外限制条件，通过并获得内容。具体步骤如下：

1. 实例化 QCloudInitiateMultipartUploadRequest，填入需要的参数。
2. 调用 QCloudCOSXMLService 对象中的 InitiateMultipartUpload 方法发出请求。
3. 从回调的 finishBlock 中的获取具体内容。

参数说明

参数名称	描述	类型	必填
Object	上传文件（对象）的文件名，也是对象的key	NSString *	是
bucket	存储桶名,可在COSV5控制台上面看到，格式为- ,例如 testBucket-1253653367	NSString *	是
storageClass	对象的存储级别	QCloudCOSStorageClass	是
cacheControl	RFC 2616 中定义的缓存策略	NSString *	否
contentDisposition	RFC 2616中 定义的文件名称	NSString *	否
expect	当使用 `expect=@"100-continue" `时，在收到服务端确认后才会发送请求内容	NSString *	否
expires	RFC 2616 中定义的过期时间	NSString *	否
storageClass	对象的存储级别	QCloudCOSStorageClass	否
accessControlList	定义 Object 的 ACL 属性。有效值：private，public-read-write，public-read；默认值：private	NSString *	否

参数名称	描述	类型	必填
grantRead	赋予被授权者读的权限。格式： id=" ",id=" "； 当需要给子账户授权时，id="qcs::cam::uin/:uin/"， 当需要给根账户授权时，id="qcs::cam::uin/:uin/" 其中 OwnerUin 指的是根账户的 ID，而 SubUin 指的是子账户的 ID	NSString *	否
grantWrite	授予被授权者写的权限。格式同上。	NSString *	否
grantFullControl	授予被授权者读写权限。格式同上。	NSString *	否

示例

```
QCloudInitiateMultipartUploadRequest* initrequest = [QCloudInitiateMultipartUploadRequest new];
initrequest.bucket = @"testBucket-123456789";
initrequest.object = @"对象名";
__block QCloudInitiateMultipartUploadResult* initResult;
[initrequest setFinishBlock:^(QCloudInitiateMultipartUploadResult* outputObject, NSError *error) {
initResult = outputObject;
}];
[[QCloudCOSXMLService defaultCOSXML] InitiateMultipartUpload:initrequest];
```

获取对象meta信息

方法原型

进行文件操作之前，需要导入头文件 QCloudCOSXML/QCloudCOSXML.h。在此之前您需要完成前文中的 STEP-1 初始化操作。先生成一个 QCloudHeadObjectRequest 实例，然后填入一些需要的额外限制条件，通过并获得内容。具体步骤如下：

1. 实例化 QCloudHeadObjectRequest，填入需要的参数。
2. 调用 QCloudCOSXMLService 对象中的 HeadObject 方法发出请求。
3. 从回调的 finishBlock 中的获取具体内容。

QCloudHeadObjectRequest 参数说明

参数名称	描述	类型	必填
Object	对象名	NSString *	是
bucket	存储桶名,可在COSV5控制台上面看到，格式为-,例如 testBucket-1253653367	NSString *	是
ifModifiedSince	如果文件修改时间晚于指定时间，才返回文件内容。否则返回 304 (not modified)	NSString *	是

示例

```
QCloudHeadObjectRequest* headerRequest = [QCloudHeadObjectRequest new];
headerRequest.object = @“对象名“;
headerRequest.bucket = @"testBucket-123456789";

__block id resultError;
[headerRequest setFinishBlock:^(NSDictionary* result, NSError *error) {
resultError = error;
}];

[[QCloudCOSXMLService defaultCOSXML] HeadObject:headerRequest];
```

追加文件

Append Object 接口请求可以将一个 Object（文件）以分块追加的方式上传至指定存储桶中。Object 属性为 Appendable 时，才能使用 Append Object 接口上传。Object 属性可以在 Head Object 操作中查询到，发起 Head Object 请求时，会返回自定义 Header 的『x-cos-object-type』，该 Header 只有两个枚举值：Normal 或者 Appendable。通过 Append Object 操作创建的 Object 类型为 Appendable 文件；通过 Put Object 上传的 Object 是 Normal 文件。 当 Appendable 的 Object 被执行 Put Object 的请求操作以后，原 Object 被覆盖，属性改变为 Normal 。 追加上传的 Object 建议大小 1M-5G。如果 Position 的值和当前 Object 的长度不致，COS 会返回 409 错误。如果 Append 一个 Normal 属性的文件，COS 会返回 409 ObjectNotAppendable。

方法原型

进行文件操作之前，需要导入头文件 QCloudCOSXML/QCloudCOSXML.h。在此之前您需要完成前文中的 STEP-1 初始化操作。先生成一个 QCloudAppendObjectRequest 实例，然后填入一些需要的额外限制条件，通过并获得内容。具体步骤如下：

1. 实例化 QCloudAppendObjectRequest，填入需要的参数。
2. 调用 QCloudCOSXMLService 对象中的 AppendObject 方法发出请求。
3. 从回调的 finishBlock 中的获取具体内容。

QCloudAppendObjectRequest 参数说明

参数名称	描述	类型	必填
Object	上传文件（对象）的文件名，也是对象的 key	NSString *	是
bucket	存储桶名,可在COSV5控制台上面看到，格式为-,例如 testBucket-1253653367	NSString *	是
position	追加操作的起始点，单位：字节；首次追加 position=0，后续追加 position= 当前 Object 的 content-length	int	是
body	需要上传的文件的路径。填入NSURL * 类型变量	BodyType	是
storageClass	QCloudCOSStorageClass	是	对象的存储级别
cacheControl	RFC 2616中定义的缓存策略	NSString *	否
contentDisposition	RFC 2616中定义的文件名称	NSString *	否
expect	当使用 `expect=@"100-continue"` 时，在收到服务端确认后才会发送请求内容	NSString *	否
expires	RFC 2616 中定义的过期时间	NSString *	否
storageClass	对象的存储级别	QCloudCOSStorageClass	否
accessControlList	定义 Object 的 ACL 属性。有效值：private，public-read-write，public-read；默认值：private	NSString *	否
grantRead	赋予被授权者读的权限。格式： id=" ",id=" "； 当需要给子账户授权时，id="qcs::cam::uin/:uin/"， 当需要给根账户授权时，id="qcs::cam::uin/:uin/" 其中 OwnerUin 指的是根账户的 ID，而 SubUin 指的是子账户的 ID	NSString *	否
grantWrite	授予被授权者写的权限。格式同上。	NSString *	否
grantFullControl	授予被授权者读写权限。格式同上。	NSString *	否

示例

```
QCloudAppendObjectRequest* put = [QCloudAppendObjectRequest new];
put.object = [NSUUID UUID].UUIDString;
put.bucket = @testBucket-123456789;
put.body = 文件的URL，NSURL*类型
__block NSDictionary* result = nil;
[put setFinishBlock:^(id outputObject, NSError *error) {
result = outputObject;
}];
[[QCloudCOSXMLService defaultCOSXML] AppendObject:put];
```

服务器端加密(Server side encryption)说明

COS支持服务器端加密(Server side encryption)，该特性的作用是object在上传到COS后会进行服务器端加密，然后再存储。下载时候会拿出原始没有解密的数据，进行解密后返回。这个过程对客户端是透明的，无需关心具体的加密过程。

如果需要使用该特性，那么在上传object时，需要在上传HTTP请求中加入一个额外的头部，key为 x-cos-server-side-encryption, 值为AES256。在下载时候直接照常GetObject即可。

利用SDK中自定义头部的功能可以实现这一需求。只需要在上传请求中加入自定义头部即可：

```
QCloudCOSXMLUploadObjectRequest* put = [QCloudCOSXMLUploadObjectRequest new];
__block NSString* object = [NSUUID UUID].UUIDString;
put.object = @"object";
put.bucket = @"存储桶名";
put.body = @"文件在本地的URL";
put.customHeaders = @{@"x-cos-server-side-encryption":@"AES256"};
[put setFinishBlock:^(QCloudUploadObjectResult *result, NSError *error) {
//完成回调
}];
[[QCloudCOSTransferMangerService defaultCOSTransferManager] UploadObject:put];
```

下载时候无需关心加解密过程：

```
QCloudGetObjectRequest* getObjectRequest = [[QCloudGetObjectRequest alloc] init];
getObjectRequest.bucket = self.bucket;
getObjectRequest.object = object;
NSURL* downloadPath = @“下载到本地的路径”;
getObjectRequest.downloadingURL = downloadPath;
[getObjectRequest setFinishBlock:^(id outputObject, NSError *error) {
//完成回调
}];
[[QCloudCOSXMLService defaultCOSXML] GetObject:getObjectRequest];
```

C SDK
快速入门

最近更新时间: 2025-01-15 17:01:00

下载与安装

相关资源

- 对象存储的 XML C SDK 源码下载地址：XML C SDK。
- 演示示例 Demo 下载地址：XML C SDK Demo。

环境依赖

依赖库：libcurl apr apr-util minixml。

安装 SDK

1. 安装 CMake 工具（建议 2.6.0 及以上版本），单击 这里 下载，安装方式如下：

```
./configure
make
make install
```

2. 安装 libcurl（建议 7.32.0 及以上版本），单击 这里 下载，安装方式如下：

```
./configure
make
make install
```

3. 安装 apr（建议 1.5.2 - 1.6.5 版本），单击 这里 下载，安装方式如下：

```
./configure
make
make install
```

4. 安装 apr-util（建议 1.5.4 及以上版本），单击 这里 下载，安装时需要指定 `--with-apr` 选项，安装方式如下：

```
./configure --with-apr=/your/apr/install/path
make
make install
```

5. 安装 minixml（建议 2.8 - 2.12 版本），单击 这里 下载，安装方式如下：

```
./configure
make
make install
```

6. 编译 COS C SDK。下载 XML C SDK 源码，执行如下编译命令：

```
cmake .
make
make install
```

术语解释

名称	描述

名称	描述
APPID	开发者访问 COS 服务时拥有的用户维度唯一资源标识，用以标识资源
SecretId	开发者拥有的项目身份识别 ID，用以身份认证
SecretKey	开发者拥有的项目身份密钥
Bucket	COS 中用于存储数据的容器
Object	COS 中存储的具体文件，是存储的基本实体
Region	域名中的地域信息
Endpoint	Endpoint 由 Region 和域名组成，具体格式为：“.”，其中 Domain 为自定义的域名。 在控制台创建 Bucket 时，可以看到对应的访问地址为：“.”，Bucket 后面的部分即为 Endpoint。
ACL	访问控制列表（Access Control List），是指特定 Bucket 或 Object 的访问控制信息列表
CORS	跨域资源共享（Cross-Origin Resource Sharing），指发起请求的资源所在域不同于该请求所指向资源所在的域的 HTTP 请求
Multipart Uploads	分块上传，COS 服务为上传文件提供的一种分块上传模式

开始使用

下面为您介绍使用 XML 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 cos_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 (cos_http_io_initialize(NULL, 0) != COSE_OK) {
exit(1);
}

/* 调用 COS 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 cos_http_io_deinitialize 方法释放之前分配的全局资源 */
cos_http_io_deinitialize();
return 0;
}
```

初始化请求选项

```
/* 等价于 apr_pool_t，用于内存管理的内存池，实现代码在 apr 库中 */
cos_pool_t *pool;
cos_request_options_t *options;

/* 重新创建一个新的内存池，第二个参数是 NULL，表示没有继承自其它内存池 */
cos_pool_create(&pool, NULL);

/* 创建并初始化 options，这个参数内部主要包括endpoint,access_key_id,acces_key_secret，is_cname, curl参数等全局配置信息
* options的内存是由pool分配的，后续释放掉pool后，options的内存也相当于释放掉了，不再需要单独释放内存
*/
options = cos_request_options_create(pool);
options->config = cos_config_create(options->pool);

/* cos_str_set 是用 char* 类型的字符串初始化 cos_string_t 类型*/
cos_str_set(&options->config->endpoint, "<用户的Endpoint>"); //Endpoint 依据用户所在地域的 COS 服务域名填写
cos_str_set(&options->config->access_key_id, "<用户的SecretId>"); //用户注册 COS 服务后所获得的 SecretId
cos_str_set(&options->config->access_key_secret, "<用户的SecretKey>"); //用户注册 COS 服务后所获得的 SecretKey
```

```
cos_str_set(&options->config->appid, "<用户的AppId>"); //用户注册 COS 服务后所获得的 AppId

/* 可以通过设置 sts_token 来使用临时密钥，当使用临时密钥时，access_key_id 和access_key_secret 均需要设置为对应临时密钥所配套的 SecretId 和 SecretKey */
//cos_str_set(&options->config->sts_token, "MyTokenString");
/* 是否使用了 CNAME */
options->config->is_cname = 0;

/* 用于设置网络相关参数，比如超时时间等*/
options->ctl = cos_http_controller_create(options->pool, 0);

/* 用于设置上传请求是否自动添加 Content-MD5 头部，enable 为 COS_FALSE 时上传请求将不自动添加 Content-MD5 头部，enable 为 COS_TRUE 时上传请求将自动添加Content-MD5 头部，如果不设置此项则默认将添加 Content-MD5 头部 */
cos_set_content_md5_enable(options->ctl, COS_FALSE);

/* 用于设置请求路由地址和端口，一般情况下无需设置此参数，请求将按域名解析结果路由 */
//cos_set_request_route(options->ctl, "192.168.12.34", 80);
```

创建存储桶

```
cos_pool_t *p = NULL;
int is_cname = 0;
cos_status_t *s = NULL;
cos_request_options_t *options = NULL;
cos_acl_e cos_acl = COS_ACL_PRIVATE;
cos_string_t bucket;
cos_table_t *resp_headers = NULL;

/* 重新创建一个新的内存池，第二个参数是 NULL，表示没有继承自其它内存池 */
cos_pool_create(&p, NULL);

/* 创建并初始化 options，这个参数内部主要包括endpoint,access_key_id,acces_key_secret，is_cname, curl参数等全局配置信息
* options 的内存是由 pool 分配的，后续释放掉 pool 后，options的内存也相当于释放掉了，不再需要单独释放内存
*/
options = cos_request_options_create(p);
options->config = cos_config_create(options->pool);
init_test_config(options->config, is_cname);

/* 设置 appid，endpoint，access_key_id，acces_key_secret，is_cname, curl参数等配置信息 */
cos_str_set(&options->config->endpoint, TEST_COS_ENDPOINT);
cos_str_set(&options->config->access_key_id, TEST_ACCESS_KEY_ID);
cos_str_set(&options->config->access_key_secret, TEST_ACCESS_KEY_SECRET);
cos_str_set(&options->config->appid, TEST_APPID);
options->config->is_cname = is_cname;
options->ctl = cos_http_controller_create(options->pool, 0);
/* 存储桶的命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式 */
cos_str_set(&bucket, TEST_BUCKET_NAME);

/* 调用api创建存储桶 */
s = cos_create_bucket(options, &bucket, cos_acl, &resp_headers);
if (cos_status_is_ok(s)) {
printf("create bucket succeeded\n");
} else {
printf("create bucket failed\n");
}

//destroy memory pool
cos_pool_destroy(p);
```

查询对象列表

```
cos_pool_t *p = NULL;
int is_cname = 0;
cos_status_t *s = NULL;
cos_request_options_t *options = NULL;
cos_list_object_params_t *list_params = NULL;
cos_string_t bucket;
cos_table_t *resp_headers = NULL;

/* 重新创建一个新的内存池，第二个参数是NULL，表示没有继承自其它内存池 */
cos_pool_create(&p, NULL);
```

```
/* 创建并初始化options，这个参数内部主要包括endpoint,access_key_id,acces_key_secret，is_cname, curl参数等全局配置信息
* options的内存是由pool分配的，后续释放掉pool后，options的内存也相当于释放掉了，不再需要单独释放内存
*/
options = cos_request_options_create(p);
options->config = cos_config_create(options->pool);
init_test_config(options->config, is_cname);

/* 设置 appid，endpoint，access_key_id，acces_key_secret，is_cname, curl参数等配置信息 */
cos_str_set(&options->config->endpoint, TEST_COS_ENDPOINT);
cos_str_set(&options->config->access_key_id, TEST_ACCESS_KEY_ID);
cos_str_set(&options->config->access_key_secret, TEST_ACCESS_KEY_SECRET);
cos_str_set(&options->config->appid, TEST_APPID);
options->config->is_cname = is_cname;
options->ctl = cos_http_controller_create(options->pool, 0);
/* 存储桶的命名格式为 BucketName-APPID， ，此处填写的存储桶名称必须为此格式 */
cos_str_set(&bucket, TEST_BUCKET_NAME);

/* 调用api查询对象列表 */
list_params = cos_create_list_object_params(p);
cos_str_set(&list_params->encoding_type, "url");
s = cos_list_object(options, &bucket, list_params, &resp_headers);
if (cos_status_is_ok(s)) {
printf("list object succeeded\n");
} else {
printf("list object failed\n");
}

//destroy memory pool
cos_pool_destroy(p);
```

上传对象

```
cos_pool_t *p = NULL;
int is_cname = 0;
cos_status_t *s = NULL;
cos_request_options_t *options = NULL;
cos_string_t bucket;
cos_string_t object;
cos_string_t file;
cos_table_t *resp_headers = NULL;

/* 重新创建一个新的内存池，第二个参数是NULL，表示没有继承自其它内存池 */
cos_pool_create(&p, NULL);

/* 创建并初始化options，这个参数内部主要包括endpoint,access_key_id,acces_key_secret，is_cname, curl参数等全局配置信息
* options的内存是由pool分配的，后续释放掉pool后，options的内存也相当于释放掉了，不再需要单独释放内存
*/
options = cos_request_options_create(p);
options->config = cos_config_create(options->pool);
init_test_config(options->config, is_cname);

/* 设置appid，endpoint，access_key_id，acces_key_secret，is_cname, curl参数等配置信息 */
cos_str_set(&options->config->endpoint, TEST_COS_ENDPOINT);
cos_str_set(&options->config->access_key_id, TEST_ACCESS_KEY_ID);
cos_str_set(&options->config->access_key_secret, TEST_ACCESS_KEY_SECRET);
cos_str_set(&options->config->appid, TEST_APPID);
options->config->is_cname = is_cname;
options->ctl = cos_http_controller_create(options->pool, 0);
/* 存储桶的命名格式为 BucketName-APPID， ，此处填写的存储桶名称必须为此格式 */
cos_str_set(&bucket, TEST_BUCKET_NAME);

/* 调用api上传对象 */
cos_str_set(&file, TEST_DOWNLOAD_NAME);
cos_str_set(&object, TEST_OBJECT_NAME);
s = cos_put_object_from_file(options, &bucket, &object, &file, NULL, &resp_headers);
if (cos_status_is_ok(s)) {
printf("put object succeeded\n");
} else {
printf("put object failed\n");
}
```

```
//destroy memory pool
cos_pool_destroy(p);
```

下载对象

```
cos_pool_t *p = NULL;
int is_cname = 0;
cos_status_t *s = NULL;
cos_request_options_t *options = NULL;
cos_string_t bucket;
cos_string_t object;
cos_string_t file;
cos_table_t *resp_headers = NULL;

/* 重新创建一个新的内存池，第二个参数是NULL，表示没有继承自其它内存池 */
cos_pool_create(&p, NULL);

/* 创建并初始化options，这个参数内部主要包括endpoint,access_key_id,acces_key_secret，is_cname, curl参数等全局配置信息
* options的内存是由pool分配的，后续释放掉pool后，options的内存也相当于释放掉了，不再需要单独释放内存
*/
options = cos_request_options_create(p);
options->config = cos_config_create(options->pool);
init_test_config(options->config, is_cname);

/* 设置appid，endpoint，access_key_id，acces_key_secret，is_cname, curl参数等配置信息 */
cos_str_set(&options->config->endpoint, TEST_COS_ENDPOINT);
cos_str_set(&options->config->access_key_id, TEST_ACCESS_KEY_ID);
cos_str_set(&options->config->access_key_secret, TEST_ACCESS_KEY_SECRET);
cos_str_set(&options->config->appid, TEST_APPID);
options->config->is_cname = is_cname;
options->ctl = cos_http_controller_create(options->pool, 0);
/* 存储桶的命名格式为 BucketName-APPID，，此处填写的存储桶名称必须为此格式 */
cos_str_set(&bucket, TEST_BUCKET_NAME);

/* 调用api下载对象 */
cos_str_set(&file, TEST_DOWNLOAD_NAME);
cos_str_set(&object, TEST_OBJECT_NAME);
s = cos_get_object_to_file(options, &bucket, &object, NULL, NULL, &file, &resp_headers);
if (cos_status_is_ok(s)) {
printf("get object succeeded\n");
} else {
printf("get object failed\n");
}

//destroy memory pool
cos_pool_destroy(p);
```

删除对象

```
cos_pool_t *p = NULL;
int is_cname = 0;
cos_status_t *s = NULL;
cos_request_options_t *options = NULL;
cos_string_t bucket;
cos_string_t object;
cos_table_t *resp_headers = NULL;

/* 重新创建一个新的内存池，第二个参数是 NULL，表示没有继承自其它内存池 */
cos_pool_create(&p, NULL);

/* 创建并初始化 options，这个参数内部主要包括 endpoint,access_key_id,acces_key_secret，is_cname, curl 参数等全局配置信息
* options的内存是由pool分配的，后续释放掉 pool 后，options 的内存也相当于释放掉了，不再需要单独释放内存
*/
options = cos_request_options_create(p);
options->config = cos_config_create(options->pool);
init_test_config(options->config, is_cname);

/* 设置 appid，endpoint，access_key_id，acces_key_secret，is_cname, curl参数等配置信息 */
cos_str_set(&options->config->endpoint, TEST_COS_ENDPOINT);
```

```
cos_str_set(&options->config->access_key_id, TEST_ACCESS_KEY_ID);
cos_str_set(&options->config->access_key_secret, TEST_ACCESS_KEY_SECRET);
cos_str_set(&options->config->appid, TEST_APPID);
options->config->is_cname = is_cname;
options->ctl = cos_http_controller_create(options->pool, 0);
/* 存储桶的命名格式为 BucketName-APPID ，，此处填写的存储桶名称必须为此格式 */
cos_str_set(&bucket, TEST_BUCKET_NAME);

/* 调用 api 删除对象 */
cos_str_set(&object, TEST_OBJECT_NAME);
s = cos_delete_object(options, &bucket, &object, &resp_headers);
if (cos_status_is_ok(s)) {
printf("delete object succeeded\n");
} else {
printf("delete object failed\n");
}

//destroy memory pool
cos_pool_destroy(p);
```

接口文档

最近更新时间: 2025-01-15 17:01:00

存储桶操作

简介

本文档提供关于存储桶的基本操作和访问控制列表（ACL）的相关 API 概览以及 SDK 示例代码。

基本操作

API	操作名	操作描述
PUT Bucket	创建存储桶	在指定账号下创建一个存储桶
DELETE Bucket	删除存储桶	删除指定账号下的空存储桶

访问控制列表

API	操作名	操作描述
PUT Bucket acl	设置存储桶 ACL	设置指定存储桶访问权限控制列表
GET Bucket acl	查询存储桶 ACL	查询存储桶的访问控制列表

基本操作

创建存储桶

功能说明

在指定账号下创建一个存储桶。

方法原型

```
cos_status_t *cos_create_bucket(const cos_request_options_t *options,
const cos_string_t *bucket,
cos_acl_e cos_acl,
cos_table_t **resp_headers);
```

参数说明

参数名称	参数描述	类型
options	COS 请求选项	Struct
bucket	存储桶名称，存储桶的命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String
cos_acl	允许用户自定义权限。 有效值：COS_ACL_PRIVATE(0)，COS_ACL_PUBLIC_READ(1)，COS_ACL_PUBLIC_READ_WRITE(2) 默认值：COS_ACL_PRIVATE(0)	Enum
resp_headers	返回 HTTP 响应消息的头域	Struct

返回结果说明

返回结果	描述	类型
code	错误码	Int
error_code	错误码内容	String

返回结果	描述	类型
error_msg	错误码描述	String
req_id	请求消息 ID	String

示例

```
cos_pool_t *p = NULL;
int is_cname = 0;
cos_status_t *s = NULL;
cos_request_options_t *options = NULL;
cos_acl_e cos_acl = COS_ACL_PRIVATE;
cos_string_t bucket;
cos_table_t *resp_headers = NULL;

//创建内存池
cos_pool_create(&p, NULL);

//初始化请求选项
options = cos_request_options_create(p);
options->config = cos_config_create(options->pool);
init_test_config(options->config, is_cname);
cos_str_set(&options->config->endpoint, TEST_COS_ENDPOINT);
cos_str_set(&options->config->access_key_id, TEST_ACCESS_KEY_ID);
cos_str_set(&options->config->access_key_secret, TEST_ACCESS_KEY_SECRET);
cos_str_set(&options->config->appid, TEST_APPID);
options->config->is_cname = is_cname;
options->ctl = cos_http_controller_create(options->pool, 0);
cos_str_set(&bucket, TEST_BUCKET_NAME);

//创建存储桶
s = cos_create_bucket(options, &bucket, cos_acl, &resp_headers);
if (cos_status_is_ok(s)) {
printf("create bucket succeeded\n");
} else {
printf("create bucket failed\n");
}

//销毁内存池
cos_pool_destroy(p);
```

删除存储桶

功能说明

删除指定账号下的空存储桶。

方法原型

```
cos_status_t *cos_delete_bucket(const cos_request_options_t *options,
const cos_string_t *bucket,
cos_table_t **resp_headers);
```

参数说明

参数名称	参数描述	类型
options	COS 请求选项	Struct
bucket	存储桶名称，存储桶的命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String
resp_headers	返回 HTTP 响应消息的头域	Struct

返回结果说明

返回结果	描述	类型
code	错误码	Int

返回结果	描述	类型
error_code	错误码内容	String
error_msg	错误码描述	String
req_id	请求消息 ID	String

示例

```
cos_pool_t *p = NULL;
int is_cname = 0;
cos_status_t *s = NULL;
cos_request_options_t *options = NULL;
cos_string_t bucket;
cos_table_t *resp_headers = NULL;

//创建内存池
cos_pool_create(&p, NULL);

//初始化请求选项
options = cos_request_options_create(p);
options->config = cos_config_create(options->pool);
init_test_config(options->config, is_cname);
cos_str_set(&options->config->endpoint, TEST_COS_ENDPOINT);
cos_str_set(&options->config->access_key_id, TEST_ACCESS_KEY_ID);
cos_str_set(&options->config->access_key_secret, TEST_ACCESS_KEY_SECRET);
cos_str_set(&options->config->appid, TEST_APPID);
options->config->is_cname = is_cname;
options->ctl = cos_http_controller_create(options->pool, 0);
cos_str_set(&bucket, TEST_BUCKET_NAME);

//删除存储桶
s = cos_delete_bucket(options, &bucket, &resp_headers);
if (cos_status_is_ok(s)) {
printf("create bucket succeeded\n");
} else {
printf("create bucket failed\n");
}

//销毁内存池
cos_pool_destroy(p);
```

设置存储桶 ACL

功能说明

设置指定存储桶访问权限控制列表。

方法原型

```
cos_status_t *cos_put_bucket_acl(const cos_request_options_t *options,
const cos_string_t *bucket,
cos_acl_e cos_acl,
const cos_string_t *grant_read,
const cos_string_t *grant_write,
const cos_string_t *grant_full_ctrl,
cos_table_t **resp_headers);
```

参数说明

参数名称	参数描述	类型
options	COS 请求选项	Struct
bucket	存储桶名称，存储桶的命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String

参数名称	参数描述	类型
cos_acl	允许用户自定义权限。 有效值：COS_ACL_PRIVATE(0)，COS_ACL_PUBLIC_READ(1)，COS_ACL_PUBLIC_READ_WRITE(2) 默认值：COS_ACL_PRIVATE(0)	Enum
grant_read	读权限授予者	String
grant_write	写权限授予者	String
grant_full_ctrl	读写权限授予者	String
resp_headers	返回 HTTP 响应消息的头域	Struct

返回结果说明

返回结果	描述	类型
code	错误码	Int
error_code	错误码内容	String
error_msg	错误码描述	String
req_id	请求消息 ID	String

示例

```
cos_pool_t *p = NULL;
int is_cname = 0;
cos_status_t *s = NULL;
cos_request_options_t *options = NULL;
cos_string_t bucket;
cos_table_t *resp_headers = NULL;

//创建内存池
cos_pool_create(&p, NULL);

//初始化请求选项
options = cos_request_options_create(p);
options->config = cos_config_create(options->pool);
init_test_config(options->config, is_cname);
cos_str_set(&options->config->endpoint, TEST_COS_ENDPOINT);
cos_str_set(&options->config->access_key_id, TEST_ACCESS_KEY_ID);
cos_str_set(&options->config->access_key_secret, TEST_ACCESS_KEY_SECRET);
cos_str_set(&options->config->appid, TEST_APPID);
options->config->is_cname = is_cname;
options->ctl = cos_http_controller_create(options->pool, 0);
cos_str_set(&bucket, TEST_BUCKET_NAME);

//设置存储桶 ACL
cos_string_t read;
cos_str_set(&read, "id=\"qcs::cam::uin/100000000001:uin/100000000001\", id=\"qcs::cam::uin/100000000011:uin/100000000011\"");
s = cos_put_bucket_acl(options, &bucket, cos_acl, &read, NULL, NULL, &resp_headers);
if (cos_status_is_ok(s)) {
printf("put bucket acl succeeded\n");
} else {
printf("put bucket acl failed\n");
}

//销毁内存池
cos_pool_destroy(p);
```

查询存储桶 ACL

功能说明

查询存储桶的访问控制列表。

方法原型

```
cos_status_t *cos_get_bucket_acl(const cos_request_options_t *options,
const cos_string_t *bucket,
cos_acl_params_t *acl_param,
cos_table_t **resp_headers)
```

参数说明

参数名称	参数描述	类型
options	COS 请求选项	Struct
bucket	存储桶名称，存储桶的命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String
acl_param	请求操作参数	Struct
owner_id	请求操作返回的存储桶持有者 ID	String
owner_name	请求操作返回的存储桶持有者的名称	String
object_list	请求操作返回的被授权者信息与权限信息	Struct
type	请求操作返回的被授权者账户类型	String
id	请求操作返回的被授权者用户 ID	String
name	请求操作返回的被授权者用户名称	String
permission	请求操作返回的被授权者权限信息	String
resp_headers	返回 HTTP 响应消息的头域	Struct

返回结果说明

返回结果	描述	类型
code	错误码	Int
error_code	错误码内容	String
error_msg	错误码描述	String
req_id	请求消息 ID	String

示例

```
cos_pool_t *p = NULL;
int is_cname = 0;
cos_status_t *s = NULL;
cos_request_options_t *options = NULL;
cos_string_t bucket;
cos_table_t *resp_headers = NULL;

//创建内存池
cos_pool_create(&p, NULL);

//初始化请求选项
options = cos_request_options_create(p);
options->config = cos_config_create(options->pool);
init_test_config(options->config, is_cname);
cos_str_set(&options->config->endpoint, TEST_COS_ENDPOINT);
cos_str_set(&options->config->access_key_id, TEST_ACCESS_KEY_ID);
cos_str_set(&options->config->access_key_secret, TEST_ACCESS_KEY_SECRET);
cos_str_set(&options->config->appid, TEST_APPID);
options->config->is_cname = is_cname;
options->ctl = cos_http_controller_create(options->pool, 0);
cos_str_set(&bucket, TEST_BUCKET_NAME);

//获取存储桶 ACL
cos_acl_params_t *acl_params = NULL;
acl_params = cos_create_acl_params(p);
```

```
s = cos_get_bucket_acl(options, &bucket, acl_params, &resp_headers);
if (cos_status_is_ok(s)) {
printf("get bucket acl succeeded\n");
printf("acl owner id:%s, name:%s\n", acl_params->owner_id.data, acl_params->owner_name.data);
cos_acl_grantee_content_t *acl_content = NULL;
cos_list_for_each_entry(cos_acl_grantee_content_t, acl_content, &acl_params->grantee_list, node) {
printf("acl grantee type:%s, id:%s, name:%s, permission:%s\n", acl_content->type.data, acl_content->id.data, acl_content->name.data, acl_content->permis
sion.data);
}
} else {
printf("get bucket acl failed\n");
}

//销毁内存池
cos_pool_destroy(p);
```

对象操作

简介

本文档提供关于对象的简单操作、分块操作等其他操作相关的 API 概览以及 SDK 示例代码。

简单操作

API	操作名	操作描述
GET Bucket（List Object）	查询对象列表	查询存储桶下的部分或者全部对象
PUT Object	简单上传对象	上传一个对象至存储桶
HEAD Object	查询对象元数据	查询对象的元数据信息
GET Object	下载对象	下载一个对象至本地
PUT Object - Copy	设置对象复制	复制文件到目标路径
DELETE Object	删除单个对象	在存储桶中删除指定对象
DELETE Multiple Objects	删除多个对象	在存储桶中批量删除对象

分块操作

API	操作名	操作描述
List Multipart Uploads	查询分块上传	查询正在进行中的分块上传信息
Initiate Multipart Upload	初始化分块上传	初始化分块上传任务
Upload Part	上传分块	分块上传文件
Upload Part - Copy	复制分块	将其他对象复制为一个分块
List Parts	查询已上传块	查询特定分块上传操作中的已上传的块
Complete Multipart Upload	完成分块上传	完成整个文件的分块上传
Abort Multipart Upload	终止分块上传	终止一个分块上传操作并删除已上传的块

其他操作

API	操作名	操作描述
PUT Object acl	设置对象 ACL	设置存储桶中某个对象的访问控制列表

API	操作名	操作描述
GET Object acl	查询对象 ACL	查询对象的访问控制列表

简单操作

查询对象列表

功能说明

查询存储桶下的部分或者全部对象。

方法原型

```
cos_status_t *cos_list_object(const cos_request_options_t *options,
const cos_string_t *bucket,
cos_list_object_params_t *params,
cos_table_t **resp_headers);
```

参数说明

参数名称	参数描述	类型
options	COS 请求选项	Struct
bucket	存储桶名称，Bucket 的命名规则为 BucketName-APPID ，此处填写的存储桶名称必须为此格式	String
params	列表操作参数信息	Struct
encoding_type	规定返回值的编码方式	String
prefix	前缀匹配，用来规定返回的文件前缀地址	String
marker	默认以 UTF-8 二进制顺序列出条目，所有列出条目从 marker 开始	String
delimiter	查询分隔符，用于对对象键进行分组	String
max_ret	单次返回最大的条目数量，默认1000	Struct
truncated	返回条目是否被截断，'true' 或者 'false'	Boolean
next_marker	假如返回条目被截断，则返回 NextMarker 就是下一个条目的起点	String
object_list	Get Bucket 操作返回的对象信息列表	Struct
key	Get Bucket 操作返回的 Object 名称	Struct
last_modified	Get Bucket 操作返回的 Object 最后修改时间	Struct
etag	Get Bucket 操作返回的对象的 SHA-1 算法校验值	Struct
size	Get Bucket 操作返回的对象大小，单位 Byte	Struct
owner_id	Get Bucket 操作返回的对象拥有者 UID 信息	Struct
storage_class	Get Bucket 操作返回的对象存储级别	Struct
common_prefix_list	将 Prefix 到 delimiter 之间的相同路径归为一类，定义为 Common Prefix	Struct
resp_headers	返回 HTTP 响应消息的头域	Struct

返回结果说明

返回结果	描述	类型
code	错误码	Int
error_code	错误码内容	String
error_msg	错误码描述	String

返回结果	描述	类型
req_id	请求消息 ID	String

示例

```
cos_pool_t *p = NULL;
int is_cname = 0;
cos_status_t *s = NULL;
cos_request_options_t *options = NULL;
cos_string_t bucket;
cos_table_t *resp_headers = NULL;

//创建内存池
cos_pool_create(&p, NULL);

//初始化请求选项
options = cos_request_options_create(p);
options->config = cos_config_create(options->pool);
init_test_config(options->config, is_cname);
cos_str_set(&options->config->endpoint, TEST_COS_ENDPOINT);
cos_str_set(&options->config->access_key_id, TEST_ACCESS_KEY_ID);
cos_str_set(&options->config->access_key_secret, TEST_ACCESS_KEY_SECRET);
cos_str_set(&options->config->appid, TEST_APPID);
options->config->is_cname = is_cname;
options->ctl = cos_http_controller_create(options->pool, 0);
cos_str_set(&bucket, TEST_BUCKET_NAME);

//获取对象列表
cos_list_object_params_t *list_params = NULL;
cos_list_object_content_t *content = NULL;
list_params = cos_create_list_object_params(p);
s = cos_list_object(options, &bucket, list_params, &resp_headers);
if (cos_status_is_ok(s)) {
printf("list object succeeded\n");
cos_list_for_each_entry(cos_list_object_content_t, content, &list_params->object_list, node) {
key = printf("%.*s\n", content->key.len, content->key.data);
}
} else {
printf("list object failed\n");
}

//销毁内存池
cos_pool_destroy(p);
```

简单上传对象

功能说明

上传一个对象至存储桶。

方法原型

```
cos_status_t *cos_put_object_from_file(const cos_request_options_t *options,
const cos_string_t *bucket,
const cos_string_t *object,
const cos_string_t *filename,
cos_table_t *headers,
cos_table_t **resp_headers);
```

参数说明

参数名称	参数描述	类型
options	COS 请求选项	Struct
bucket	存储桶名称，存储桶的命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String
object	Object 名称	String

参数名称	参数描述	类型
filename	Object 本地保存文件名称	String
headers	COS 请求附加头域	Struct
resp_headers	返回 HTTP 响应消息的头域	Struct

返回结果说明

返回结果	描述	类型
code	错误码	Int
error_code	错误码内容	String
error_msg	错误码描述	String
req_id	请求消息 ID	String

示例

```
cos_pool_t *p = NULL;
int is_cname = 0;
cos_status_t *s = NULL;
cos_request_options_t *options = NULL;
cos_string_t bucket;
cos_string_t object;
cos_string_t file;
cos_table_t *resp_headers = NULL;

//创建内存池
cos_pool_create(&p, NULL);

//初始化请求选项
options = cos_request_options_create(p);
options->config = cos_config_create(options->pool);
init_test_config(options->config, is_cname);
cos_str_set(&options->config->endpoint, TEST_COS_ENDPOINT);
cos_str_set(&options->config->access_key_id, TEST_ACCESS_KEY_ID);
cos_str_set(&options->config->access_key_secret, TEST_ACCESS_KEY_SECRET);
cos_str_set(&options->config->appid, TEST_APPID);
options->config->is_cname = is_cname;
options->ctl = cos_http_controller_create(options->pool, 0);
cos_str_set(&bucket, TEST_BUCKET_NAME);

//上传对象
cos_str_set(&file, TEST_DOWNLOAD_NAME);
cos_str_set(&object, TEST_OBJECT_NAME);
s = cos_put_object_from_file(options, &bucket, &object, &file, NULL, &resp_headers);
if (cos_status_is_ok(s)) {
printf("put object succeeded\n");
} else {
printf("put object failed\n");
}

//销毁内存池
cos_pool_destroy(p);
```

查询对象元数据

功能说明

查询对象的元数据信息。

方法原型

```
cos_status_t *cos_head_object(const cos_request_options_t *options,
const cos_string_t *bucket,
const cos_string_t *object,
```

```
cos_table_t *headers,
cos_table_t **resp_headers);
```

参数说明

参数名称	参数描述	类型
options	COS 请求选项	Struct
bucket	存储桶名称，存储桶的命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String
object	Object 名称	String
headers	COS 请求附加头域	Struct
resp_headers	返回 HTTP 响应消息的头域	Struct

返回结果说明

返回结果	描述	类型
code	错误码	Int
error_code	错误码内容	String
error_msg	错误码描述	String
req_id	请求消息 ID	String

示例

```
cos_pool_t *p = NULL;
int is_cname = 0;
cos_status_t *s = NULL;
cos_request_options_t *options = NULL;
cos_string_t bucket;
cos_string_t object;
cos_table_t *resp_headers = NULL;

//创建内存池
cos_pool_create(&p, NULL);

//初始化请求选项
options = cos_request_options_create(p);
options->config = cos_config_create(options->pool);
init_test_config(options->config, is_cname);
cos_str_set(&options->config->endpoint, TEST_COS_ENDPOINT);
cos_str_set(&options->config->access_key_id, TEST_ACCESS_KEY_ID);
cos_str_set(&options->config->access_key_secret, TEST_ACCESS_KEY_SECRET);
cos_str_set(&options->config->appid, TEST_APPID);
options->config->is_cname = is_cname;
options->ctl = cos_http_controller_create(options->pool, 0);
cos_str_set(&bucket, TEST_BUCKET_NAME);

//获取对象元数据
cos_str_set(&object, TEST_OBJECT_NAME);
s = cos_head_object(options, &bucket, &object, NULL, &resp_headers);
if (cos_status_is_ok(s)) {
printf("head object succeeded\n");
} else {
printf("head object failed\n");
}

//销毁内存池
cos_pool_destroy(p);
```

下载对象

功能说明

下载一个对象至本地。该操作需要对目标对象具有读权限或该目标对象已对所有人都开放了读权限（公有读）。

方法原型

```
cos_status_t *cos_get_object_to_file(const cos_request_options_t *options,
const cos_string_t *bucket,
const cos_string_t *object,
cos_table_t *headers,
cos_table_t *params,
cos_string_t *filename,
cos_table_t **resp_headers);
```

参数说明

参数名称	参数描述	类型
options	COS 请求选项	Struct
bucket	存储桶名称，存储桶的命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String
object	Object 名称	String
headers	COS 请求附加头域	Struct
params	COS 请求操作参数	Struct
filename	Object 本地保存文件名称	String
resp_headers	返回 HTTP 响应消息的头域	Struct

返回结果说明

返回结果	描述	类型
code	错误码	Int
error_code	错误码内容	String
error_msg	错误码描述	String
req_id	请求消息 ID	String

示例

```
cos_pool_t *p = NULL;
int is_cname = 0;
cos_status_t *s = NULL;
cos_request_options_t *options = NULL;
cos_string_t bucket;
cos_string_t object;
cos_string_t file;
cos_table_t *resp_headers = NULL;

//创建内存池
cos_pool_create(&p, NULL);

//初始化请求选项
options = cos_request_options_create(p);
options->config = cos_config_create(options->pool);
init_test_config(options->config, is_cname);
cos_str_set(&options->config->endpoint, TEST_COS_ENDPOINT);
cos_str_set(&options->config->access_key_id, TEST_ACCESS_KEY_ID);
cos_str_set(&options->config->access_key_secret, TEST_ACCESS_KEY_SECRET);
cos_str_set(&options->config->appid, TEST_APPID);
options->config->is_cname = is_cname;
options->ctl = cos_http_controller_create(options->pool, 0);
cos_str_set(&bucket, TEST_BUCKET_NAME);

//获取对象
cos_str_set(&file, TEST_DOWNLOAD_NAME);
```

```
cos_str_set(&object, TEST_OBJECT_NAME);
s = cos_get_object_to_file(options, &bucket, &object, NULL, NULL, &file, &resp_headers);
if (cos_status_is_ok(s)) {
printf("get object succeeded\n");
} else {
printf("get object failed\n");
}

//销毁内存池
cos_pool_destroy(p);
```

设置对象复制

功能说明

复制文件到目标路径。

方法原型

```
cos_status_t *cos_copy_object(const cos_request_options_t *options,
const cos_string_t *copy_source,
const cos_string_t *dest_bucket,
const cos_string_t *dest_object,
cos_table_t *headers,
cos_copy_object_params_t *copy_object_param,
cos_table_t **resp_headers);
```

参数说明

参数名称	参数描述	类型
options	COS 请求选项	Struct
copy_source	源文件路径	String
dest_bucket	目的存储桶名称，存储桶的命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String
dest_object	目的 Object 名称	String
headers	COS 请求附加头域	Struct
copy_object_param	Put Object Copy 操作参数	Struct
etag	返回文件的 MD5 算法校验值	String
last_modify	返回文件最后修改时间，GMT 格式	String
resp_headers	返回 HTTP 响应消息的头域	Struct

返回结果说明

返回结果	描述	类型
code	错误码	Int
error_code	错误码内容	String
error_msg	错误码描述	String
req_id	请求消息 ID	String

示例

```
cos_pool_t *p = NULL;
int is_cname = 0;
cos_status_t *s = NULL;
cos_request_options_t *options = NULL;
cos_string_t bucket;
cos_string_t object;
cos_table_t *resp_headers = NULL;
```

```
//创建内存池
cos_pool_create(&p, NULL);

//初始化请求选项
options = cos_request_options_create(p);
options->config = cos_config_create(options->pool);
init_test_config(options->config, is_cname);
cos_str_set(&options->config->endpoint, TEST_COS_ENDPOINT);
cos_str_set(&options->config->access_key_id, TEST_ACCESS_KEY_ID);
cos_str_set(&options->config->access_key_secret, TEST_ACCESS_KEY_SECRET);
cos_str_set(&options->config->appid, TEST_APPID);
options->config->is_cname = is_cname;
options->ctl = cos_http_controller_create(options->pool, 0);
cos_str_set(&bucket, TEST_BUCKET_NAME);

//设置对象复制
cos_str_set(&object, TEST_OBJECT_NAME);
cos_string_t copy_source;
cos_str_set(&copy_source, TEST_COPY_SRC);
cos_copy_object_params_t *params = NULL;
params = cos_create_copy_object_params(p);
s = cos_copy_object(options, &copy_source, &bucket, &object, NULL, params, &resp_headers);
if (cos_status_is_ok(s)) {
printf("put object copy succeeded\n");
} else {
printf("put object copy failed\n");
}

//销毁内存池
cos_pool_destroy(p);
```

删除单个对象

功能说明

在存储桶中删除指定对象。

方法原型

```
cos_status_t *cos_delete_object(const cos_request_options_t *options,
const cos_string_t *bucket,
const cos_string_t *object,
cos_table_t **resp_headers);
```

参数说明

参数名称	参数描述	类型
options	COS 请求选项	Struct
bucket	存储桶名称，存储桶的命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String
object	Object 名称	String
resp_headers	返回 HTTP 响应消息的头域	Struct

返回结果说明

返回结果	描述	类型
code	错误码	Int
error_code	错误码内容	String
error_msg	错误码描述	String
req_id	请求消息 ID	String

示例

```
cos_pool_t *p = NULL;
int is_cname = 0;
cos_status_t *s = NULL;
cos_request_options_t *options = NULL;
cos_string_t bucket;
cos_string_t object;
cos_table_t *resp_headers = NULL;

//创建内存池
cos_pool_create(&p, NULL);

//初始化请求选项
options = cos_request_options_create(p);
options->config = cos_config_create(options->pool);
init_test_config(options->config, is_cname);
cos_str_set(&options->config->endpoint, TEST_COS_ENDPOINT);
cos_str_set(&options->config->access_key_id, TEST_ACCESS_KEY_ID);
cos_str_set(&options->config->access_key_secret, TEST_ACCESS_KEY_SECRET);
cos_str_set(&options->config->appid, TEST_APPID);
options->config->is_cname = is_cname;
options->ctl = cos_http_controller_create(options->pool, 0);
cos_str_set(&bucket, TEST_BUCKET_NAME);

//删除单个对象
cos_str_set(&object, TEST_OBJECT_NAME);
s = cos_delete_object(options, &bucket, &object, &resp_headers);
if (cos_status_is_ok(s)) {
printf("delete object succeeded\n");
} else {
printf("delete object failed\n");
}

//销毁内存池
cos_pool_destroy(p);
```

删除多个对象

功能说明

在存储桶中批量删除对象，最大支持单次删除1000个对象。对于返回结果，COS 提供 Verbose 和 Quiet 两种结果模式。Verbose 模式将返回每个 Object 的删除结果。Quiet 模式只返回报错的 Object 信息。

方法原型

```
cos_status_t *cos_delete_objects(const cos_request_options_t *options,
const cos_string_t *bucket,
cos_list_t *object_list,
int is_quiet,
cos_table_t **resp_headers,
cos_list_t *deleted_object_list);
```

参数说明

参数名称	参数描述	类型
options	COS 请求选项	Struct
bucket	存储桶名称，存储桶的命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String
object_list	Object 待删除列表	Struct
key	待删除 Object 名称	String
is_quiet	决定是否启动 Quiet 模式 True(1)：启动 Quiet 模式，False(0)：启动 Verbose 模式。 默认为 False(0)	Boolean
resp_headers	返回 HTTP 响应消息的头域	Struct
deleted_object_list	Object 删除信息列表	Struct

返回结果说明

返回结果	描述	类型
code	错误码	Int
error_code	错误码内容	String
error_msg	错误码描述	String
req_id	请求消息 ID	String

示例

```
cos_pool_t *p = NULL;
int is_cname = 0;
cos_status_t *s = NULL;
cos_request_options_t *options = NULL;
cos_string_t bucket;
cos_string_t object;
cos_table_t *resp_headers = NULL;

//创建内存池
cos_pool_create(&p, NULL);

//初始化请求选项
options = cos_request_options_create(p);
options->config = cos_config_create(options->pool);
init_test_config(options->config, is_cname);
cos_str_set(&options->config->endpoint, TEST_COS_ENDPOINT);
cos_str_set(&options->config->access_key_id, TEST_ACCESS_KEY_ID);
cos_str_set(&options->config->access_key_secret, TEST_ACCESS_KEY_SECRET);
cos_str_set(&options->config->appid, TEST_APPID);
options->config->is_cname = is_cname;
options->ctl = cos_http_controller_create(options->pool, 0);
cos_str_set(&bucket, TEST_BUCKET_NAME);

//设置批量删除对象
char *object_name1 = TEST_OBJECT_NAME1;
char *object_name2 = TEST_OBJECT_NAME2;
cos_object_key_t *content1 = NULL;
cos_object_key_t *content2 = NULL;
cos_list_t object_list;
cos_list_t deleted_object_list;
cos_list_init(&object_list);
cos_list_init(&deleted_object_list);
content1 = cos_create_cos_object_key(p);
cos_str_set(&content1->key, object_name1);
cos_list_add_tail(&content1->node, &object_list);
content2 = cos_create_cos_object_key(p);
cos_str_set(&content2->key, object_name2);
cos_list_add_tail(&content2->node, &object_list);

//批量删除对象
int is_quiet = COS_TRUE;
cos_str_set(&object, TEST_OBJECT_NAME);
s = cos_delete_objects(options, &bucket, &object_list, is_quiet, &resp_headers, &deleted_object_list);
if (cos_status_is_ok(s)) {
printf("delete objects succeeded\n");
} else {
printf("delete objects failed\n");
}

//销毁内存池
cos_pool_destroy(p);
```

分块操作

查询分块上传

功能说明

查询正在进行中的分块上传信息。单次最多列出1000个正在进行中的分块上传。

方法原型

```
cos_status_t *cos_list_multipart_upload(const cos_request_options_t *options,
const cos_string_t *bucket,
cos_list_multipart_upload_params_t *params,
cos_table_t **resp_headers);
```

参数说明

参数名称	参数描述	类型
options	COS 请求选项	Struct
bucket	存储桶名称，存储桶的命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String
params	List Multipart Uploads 操作参数	Struct
encoding_type	规定返回值的编码方式	String
prefix	前缀匹配，用来规定返回的文件前缀地址	String
upload_id_marker	假如返回条目被截断，则返回 NextMarker 就是下一个条目的起点	String
delimiter	界符为一个符号。 如果有 Prefix，则将 Prefix 到 delimiter 之间的相同路径归为一类，定义为 Common Prefix，然后列出所有 Common Prefix。 如果没有 Prefix，则从路径起点开始	String
max_ret	单次返回最大的条目数量，默认1000	String
key_marker	与 upload-id-marker 一起使用。 当 upload-id-marker 未被指定时，ObjectName 字母顺序大于 key-marker 的条目将被列出。 当 upload-id-marker 被指定时，ObjectName 字母顺序大于 key-marker 的条目被列出，ObjectName 字母顺序等于 key-marker 同时 UploadID 大于 upload-id-marker 的条目将被列出	String
upload_id_marker	与 key-marker 一起使用。 当 key-marker 未被指定时，upload-id-marker 将被忽略。 当 key-marker 被指定时，ObjectName 字母顺序大于 key-marker 的条目被列出，ObjectName 字母顺序等于 key-marker 同时 UploadID 大于 upload-id-marker 的条目将被列出	String
truncated	返回条目是否被截断，'true' 或者 'false'	Boolean
next_key_marker	假如返回条目被截断，则返回 NextMarker 就是下一个条目的起点	String
next_upload_id_marker	假如返回条目被截断，则返回 NextMarker 就是下一个条目的起点	String
upload_list	分块上传的信息	Struct
key	Object 的名称	String
upload_id	标示本次分块上传的 ID	String
initiated	标示本次分块上传任务的启动时间	String
resp_headers	返回 HTTP 响应消息的头域	Struct

```
typedef struct {
cos_list_t node;
cos_string_t key;
cos_string_t upload_id;
cos_string_t initiated;
} cos_list_multipart_upload_content_t;
```

返回结果说明

返回结果	描述	类型
code	错误码	Int
error_code	错误码内容	String
error_msg	错误码描述	String
req_id	请求消息 ID	String

示例

```
cos_pool_t *p = NULL;
cos_string_t bucket;
int is_cname = 0;
cos_table_t *resp_headers = NULL;
cos_request_options_t *options = NULL;
cos_status_t *s = NULL;
cos_list_multipart_upload_params_t *list_multipart_params = NULL;

//创建内存池 & 初始化请求选项
cos_pool_create(&p, NULL);
options = cos_request_options_create(p);
init_test_request_options(options, is_cname);
cos_str_set(&bucket, TEST_BUCKET_NAME);

//查询分块上传
list_multipart_params = cos_create_list_multipart_upload_params(p);
list_multipart_params->max_ret = 999;
s = cos_list_multipart_upload(options, &bucket, list_multipart_params, &resp_headers);
log_status(s);

//销毁内存池
cos_pool_destroy(p);
```

初始化分块上传

功能说明

Initiate Multipart Upload 请求实现初始化分片上传，成功执行此请求以后会返回 Upload ID 用于后续的 Upload Part 请求。

方法原型

```
cos_status_t *cos_init_multipart_upload(const cos_request_options_t *options,
const cos_string_t *bucket,
const cos_string_t *object,
cos_string_t *upload_id,
cos_table_t *headers,
cos_table_t **resp_headers);
```

参数说明

参数名称	参数描述	类型
options	COS 请求选项	Struct
bucket	存储桶名称，存储桶的命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String
object	Object 名称	String
upload_id	操作返回的 Upload ID	String
headers	COS 请求附加头域	Struct
resp_headers	返回 HTTP 响应消息的头域	Struct

返回结果说明

返回结果	描述	类型

返回结果	描述	类型
code	错误码	Int
error_code	错误码内容	String
error_msg	错误码描述	String
req_id	请求消息 ID	String

示例

```
cos_pool_t *p = NULL;
int is_cname = 0;
cos_status_t *s = NULL;
cos_request_options_t *options = NULL;
cos_string_t bucket;
cos_string_t object;
cos_string_t file;
cos_table_t *resp_headers = NULL;

//创建内存池
cos_pool_create(&p, NULL);

//初始化请求选项
options = cos_request_options_create(p);
options->config = cos_config_create(options->pool);
init_test_config(options->config, is_cname);
cos_str_set(&options->config->endpoint, TEST_COS_ENDPOINT);
cos_str_set(&options->config->access_key_id, TEST_ACCESS_KEY_ID);
cos_str_set(&options->config->access_key_secret, TEST_ACCESS_KEY_SECRET);
cos_str_set(&options->config->appid, TEST_APPID);
options->config->is_cname = is_cname;
options->ctl = cos_http_controller_create(options->pool, 0);
cos_str_set(&bucket, TEST_BUCKET_NAME);

//初始化分块上传
cos_str_set(&object, TEST_OBJECT_NAME);
s = cos_init_multipart_upload(options, &bucket, &object,
&upload_id, headers, &resp_headers);
if (cos_status_is_ok(s)) {
printf("init multipart upload succeeded\n");
} else {
printf("init multipart upload failed\n");
}

//销毁内存池
cos_pool_destroy(p);
```

查询已上传块

功能说明

查询特定分块上传操作中的已上传的块。

方法原型

```
cos_status_t *cos_list_upload_part(const cos_request_options_t *options,
const cos_string_t *bucket,
const cos_string_t *object,
const cos_string_t *upload_id,
cos_list_upload_part_params_t *params,
cos_table_t **resp_headers);
```

参数说明

参数名称	参数描述	类型
options	COS 请求选项	Struct

参数名称	参数描述	类型
bucket	存储桶名称，存储桶的命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String
object	Object 名称	String
upload_id	上传任务编号	String
params	List Parts 操作参数	Struct
part_number_marker	默认以 UTF-8 二进制顺序列出条目，所有列出条目从 marker 开始	String
encoding_type	规定返回值的编码方式	String
max_ret	单次返回最大的条目数量，默认1000	String
truncated	返回条目是否被截断，'true' 或者 'false'	Boolean
next_part_number_marker	假如返回条目被截断，则返回 NextMarker 就是下一个条目的起点	String
part_list	完成分块的信息	Struct
part_number	分块编号	String
size	分块大小，单位 Byte	String
etag	分块的 SHA-1 算法校验值	String
last_modified	分块最后修改时间	String
resp_headers	返回 HTTP 响应消息的头域	Struct

返回结果说明

返回结果	描述	类型
code	错误码	Int
error_code	错误码内容	String
error_msg	错误码描述	String
req_id	请求消息 ID	String

示例

```
cos_pool_t *p = NULL;
int is_cname = 0;
cos_status_t *s = NULL;
cos_request_options_t *options = NULL;
cos_string_t bucket;
cos_string_t object;
cos_string_t file;
cos_table_t *resp_headers = NULL;
cos_list_part_content_t *part_content = NULL;
cos_complete_part_content_t *complete_part_content = NULL;
int part_num = 1;
int64_t pos = 0;
int64_t file_length = 0;

//创建内存池
cos_pool_create(&p, NULL);

//初始化请求选项
options = cos_request_options_create(p);
options->config = cos_config_create(options->pool);
init_test_config(options->config, is_cname);
cos_str_set(&options->config->endpoint, TEST_COS_ENDPOINT);
cos_str_set(&options->config->access_key_id, TEST_ACCESS_KEY_ID);
cos_str_set(&options->config->access_key_secret, TEST_ACCESS_KEY_SECRET);
cos_str_set(&options->config->appid, TEST_APPID);
options->config->is_cname = is_cname;
```

```
options->ctl = cos_http_controller_create(options->pool, 0);
cos_str_set(&bucket, TEST_BUCKET_NAME);

//查询已上传块
params = cos_create_list_upload_part_params(p);
params->max_ret = 1000;
cos_list_init(&complete_part_list);
s = cos_list_upload_part(options, &bucket, &object, &upload_id,
params, &resp_headers);

if (cos_status_is_ok(s)) {
printf("List multipart succeeded\n");
} else {
printf("List multipart failed\n");
cos_pool_destroy(p);
return;
}

cos_list_for_each_entry(cos_list_part_content_t, part_content, &params->part_list, node) {
complete_part_content = cos_create_complete_part_content(p);
cos_str_set(&complete_part_content->part_number, part_content->part_number.data);
cos_str_set(&complete_part_content->etag, part_content->etag.data);
cos_list_add_tail(&complete_part_content->node, &complete_part_list);
}

//完成分块上传
s = cos_complete_multipart_upload(options, &bucket, &object, &upload_id,
&complete_part_list, complete_headers, &resp_headers);

if (cos_status_is_ok(s)) {
printf("Complete multipart upload from file succeeded, upload_id:%.*s\n",
upload_id.len, upload_id.data);
} else {
printf("Complete multipart upload from file failed\n");
}

//销毁内存池
cos_pool_destroy(p);
```

上传分块

功能说明

分块上传文件。Upload Part 请求实现在初始化以后的分块上传，支持的块的数量为1到10000，块的大小为1MB到5GB。在每次请求 Upload Part 时，需要携带 partNumber 和 uploadID，partNumber 为块的编号，支持乱序上传。

方法原型

```
cos_status_t *cos_upload_part_from_file(const cos_request_options_t *options,
const cos_string_t *bucket,
const cos_string_t *object,
const cos_string_t *upload_id,
int part_num,
cos_upload_file_t *upload_file,
cos_table_t **resp_headers);
```

参数说明

参数名称	参数描述	类型
options	COS 请求选项	Struct
bucket	存储桶名称，存储桶的命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String
object	Object 名称	String
upload_id	上传任务编号	String
part_num	分块编号	Int

参数名称	参数描述	类型
upload_file	待上传本地文件信息	Struct
resp_headers	返回 HTTP 响应消息的头域	Struct

返回结果说明

返回结果	描述	类型
code	错误码	Int
error_code	错误码内容	String
error_msg	错误码描述	String
req_id	请求消息 ID	String

示例

```
cos_pool_t *p = NULL;
int is_cname = 0;
cos_status_t *s = NULL;
cos_request_options_t *options = NULL;
cos_string_t bucket;
cos_string_t object;
cos_string_t file;
cos_table_t *resp_headers = NULL;
int part_num = 1;
int64_t pos = 0;
int64_t file_length = 0;

//创建内存池
cos_pool_create(&p, NULL);

//初始化请求选项
options = cos_request_options_create(p);
options->config = cos_config_create(options->pool);
init_test_config(options->config, is_cname);
cos_str_set(&options->config->endpoint, TEST_COS_ENDPOINT);
cos_str_set(&options->config->access_key_id, TEST_ACCESS_KEY_ID);
cos_str_set(&options->config->access_key_secret, TEST_ACCESS_KEY_SECRET);
cos_str_set(&options->config->appid, TEST_APPID);
options->config->is_cname = is_cname;
options->ctl = cos_http_controller_create(options->pool, 0);
cos_str_set(&bucket, TEST_BUCKET_NAME);

//上传分块
int res = COSE_OK;
cos_upload_file_t *upload_file = NULL;
cos_file_buf_t *fb = cos_create_file_buf(p);
res = cos_open_file_for_all_read(p, TEST_MULTIPART_FILE, fb);
if (res != COSE_OK) {
cos_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 = cos_create_upload_file(p);
cos_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 = cos_upload_part_from_file(options, &bucket, &object, &upload_id,
part_num++, upload_file, &resp_headers);

if (cos_status_is_ok(s)) {
printf("upload part succeeded\n");
} else {
```

```
printf("upload part failed\n");
}
}

//销毁内存池
cos_pool_destroy(p);
```

复制分块

功能说明

将其他对象复制为一个分块。

方法原型

```
cos_status_t *cos_upload_part_copy(const cos_request_options_t *options,
cos_upload_part_copy_params_t *params,
cos_table_t *headers,
cos_table_t **resp_headers);
```

参数说明

参数名称	参数描述	类型
options	COS 请求选项	Struct
params	复制分块参数信息	Struct
copy_source	源文件路径	String
dest_bucket	目的 存储桶名称，存储桶的命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String
dest_object	目的 Object 名称	String
upload_id	上传任务编号	String
part_num	分块编号	Int
range_start	源文件起始偏移	Int
range_end	源文件终止偏移	Int
rsp_content	复制分块结果信息	Struct
etag	返回文件的 MD5 算法校验值	String
last_modify	返回文件最后修改时间，GMT 格式	String
resp_headers	返回 HTTP 响应消息的头域	Struct

返回结果说明

返回结果	描述	类型
code	错误码	Int
error_code	错误码内容	String
error_msg	错误码描述	String
req_id	请求消息 ID	String

示例

```
cos_pool_t *p = NULL;
cos_request_options_t *options = NULL;
cos_string_t bucket;
cos_string_t object;
cos_string_t file;
int is_cname = 0;
cos_string_t upload_id;
```

```
cos_list_upload_part_params_t *list_upload_part_params = NULL;
cos_upload_part_copy_params_t *upload_part_copy_params1 = NULL;
cos_upload_part_copy_params_t *upload_part_copy_params2 = NULL;
cos_table_t *headers = NULL;
cos_table_t *query_params = NULL;
cos_table_t *resp_headers = NULL;
cos_table_t *list_part_resp_headers = NULL;
cos_list_t complete_part_list;
cos_list_part_content_t *part_content = NULL;
cos_complete_part_content_t *complete_content = NULL;
cos_table_t *complete_resp_headers = NULL;
cos_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 = "cos_test_upload_part_copy_source_object";
char *dest_object_name = "cos_test_upload_part_copy_dest_object";
FILE *fd = NULL;
cos_string_t download_file;
cos_string_t dest_bucket;
cos_string_t dest_object;
int64_t range_start1 = 0;
int64_t range_end1 = 6000000;
int64_t range_start2 = 6000001;
int64_t range_end2;
cos_string_t data;

cos_pool_create(&p, NULL);
options = cos_request_options_create(p);

//创建一个10MB本地随机文件
make_rand_string(p, 10 * 1024 * 1024, &data);
fd = fopen(local_filename, "w");
fwrite(data.data, sizeof(data.data[0]), data.len, fd);
fclose(fd);

//使用本地文件上传对象
init_test_request_options(options, is_cname);
cos_str_set(&bucket, "source-1253666666");
cos_str_set(&object, "cos_test_upload_part_copy_source_object");
cos_str_set(&file, local_filename);
s = cos_put_object_from_file(options, &bucket, &object, &file, NULL, &resp_headers);
log_status(s);

//初始化分块上传
cos_str_set(&object, dest_object_name);
s = cos_init_multipart_upload(options, &bucket, &object,
&upload_id, NULL, &resp_headers);
log_status(s);

//使用已上传对象复制分块1
upload_part_copy_params1 = cos_create_upload_part_copy_params(p);
cos_str_set(&upload_part_copy_params1->copy_source, "mybucket-1253666666.cn-south.myqcloud.com/cos_test_upload_part_copy_source_object");
cos_str_set(&upload_part_copy_params1->dest_bucket, TEST_BUCKET_NAME);
cos_str_set(&upload_part_copy_params1->dest_object, dest_object_name);
cos_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 = cos_table_make(p, 0);
s = cos_upload_part_copy(options, upload_part_copy_params1, headers, &resp_headers);
log_status(s);
printf("last modified:%s, etag:%s\n", upload_part_copy_params1->rsp_content->last_modify.data, upload_part_copy_params1->rsp_content->etag.data);

//使用已上传对象复制分块2
resp_headers = NULL;
range_end2 = get_file_size(local_filename) - 1;
upload_part_copy_params2 = cos_create_upload_part_copy_params(p);
cos_str_set(&upload_part_copy_params2->copy_source, "mybucket-1253666666.cn-south.myqcloud.com/cos_test_upload_part_copy_source_object");
cos_str_set(&upload_part_copy_params2->dest_bucket, TEST_BUCKET_NAME);
cos_str_set(&upload_part_copy_params2->dest_object, dest_object_name);
```

```
cos_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 = cos_table_make(p, 0);
s = cos_upload_part_copy(options, upload_part_copy_params2, headers, &resp_headers);
log_status(s);
printf("last modified:%s, etag:%s\n", upload_part_copy_params1->rsp_content->last_modify.data, upload_part_copy_params1->rsp_content->etag.data);

//列出已上传对象
list_upload_part_params = cos_create_list_upload_part_params(p);
list_upload_part_params->max_ret = 10;
cos_list_init(&complete_part_list);

cos_str_set(&dest_bucket, TEST_BUCKET_NAME);
cos_str_set(&dest_object, dest_object_name);
s = cos_list_upload_part(options, &dest_bucket, &dest_object, &upload_id,
list_upload_part_params, &list_part_resp_headers);
log_status(s);
cos_list_for_each_entry(cos_list_part_content_t, part_content, &list_upload_part_params->part_list, node) {
complete_content = cos_create_complete_part_content(p);
cos_str_set(&complete_content->part_number, part_content->part_number.data);
cos_str_set(&complete_content->etag, part_content->etag.data);
cos_list_add_tail(&complete_content->node, &complete_part_list);
}

//完成分块上传
headers = cos_table_make(p, 0);
s = cos_complete_multipart_upload(options, &dest_bucket, &dest_object,
&upload_id, &complete_part_list, headers, &complete_resp_headers);
log_status(s);

//对比复制分块上传生成的对象和本地文件是否匹配
headers = cos_table_make(p, 0);
cos_str_set(&download_file, download_filename);
s = cos_get_object_to_file(options, &dest_bucket, &dest_object, headers,
query_params, &download_file, &resp_headers);
log_status(s);
printf("local file len = %"APR_INT64_T_FMT", download file len = %"APR_INT64_T_FMT, get_file_size(local_filename), get_file_size(download_filename));
remove(download_filename);
remove(local_filename);

//销毁内存池
cos_pool_destroy(p);
```

完成分块上传

功能说明

完成整个文件的分块上传。当您已经使用 Upload Parts 上传所有块以后，您可以用该 API 完成上传。在使用该 API 时，您必须在 Body 中给出每一个块的 PartNumber 和 ETag，用来校验块的准确性。

方法原型

```
cos_status_t *cos_complete_multipart_upload(const cos_request_options_t *options,
const cos_string_t *bucket,
const cos_string_t *object,
const cos_string_t *upload_id,
cos_list_t *part_list,
cos_table_t *headers,
cos_table_t **resp_headers);
```

参数说明

参数名称	参数描述	类型
options	COS 请求选项	Struct
bucket	存储桶名称，存储桶的命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String

参数名称	参数描述	类型
object	Object 名称	String
upload_id	上传任务编号	String
part_list	完成分块上传的参数	Struct
part_number	分块编号	String
etag	分块的 ETag 值，为 sha1 校验值，需要在校验值前后加上双引号，如 "3a0f1fd698c235af9cf098cb74aa25bc"	String
headers	COS 请求附加头域	Struct
resp_headers	返回 HTTP 响应消息的头域	Struct

返回结果说明

返回结果	描述	类型
code	错误码	Int
error_code	错误码内容	String
error_msg	错误码描述	String
req_id	请求消息 ID	String

示例

```
cos_pool_t *p = NULL;
int is_cname = 0;
cos_status_t *s = NULL;
cos_request_options_t *options = NULL;
cos_string_t bucket;
cos_string_t object;
cos_string_t file;
cos_table_t *resp_headers = NULL;
cos_list_part_content_t *part_content = NULL;
cos_complete_part_content_t *complete_part_content = NULL;
int part_num = 1;
int64_t pos = 0;
int64_t file_length = 0;

//创建内存池
cos_pool_create(&p, NULL);

//初始化请求选项
options = cos_request_options_create(p);
options->config = cos_config_create(options->pool);
init_test_config(options->config, is_cname);
cos_str_set(&options->config->endpoint, TEST_COS_ENDPOINT);
cos_str_set(&options->config->access_key_id, TEST_ACCESS_KEY_ID);
cos_str_set(&options->config->access_key_secret, TEST_ACCESS_KEY_SECRET);
cos_str_set(&options->config->appid, TEST_APPID);
options->config->is_cname = is_cname;
options->ctl = cos_http_controller_create(options->pool, 0);
cos_str_set(&bucket, TEST_BUCKET_NAME);

//查询已上传分块
params = cos_create_list_upload_part_params(p);
params->max_ret = 1000;
cos_list_init(&complete_part_list);
s = cos_list_upload_part(options, &bucket, &object, &upload_id,
params, &resp_headers);

if (cos_status_is_ok(s)) {
printf("List multipart succeeded\n");
} else {
printf("List multipart failed\n");
cos_pool_destroy(p);
```

```
return;
}

cos_list_for_each_entry(cos_list_part_content_t, part_content, &params->part_list, node) {
complete_part_content = cos_create_complete_part_content(p);
cos_str_set(&complete_part_content->part_number, part_content->part_number.data);
cos_str_set(&complete_part_content->etag, part_content->etag.data);
cos_list_add_tail(&complete_part_content->node, &complete_part_list);
}

//完成分块上传
s = cos_complete_multipart_upload(options, &bucket, &object, &upload_id,
&complete_part_list, complete_headers, &resp_headers);

if (cos_status_is_ok(s)) {
printf("Complete multipart upload from file succeeded, upload_id:%.*s\n",
upload_id.len, upload_id.data);
} else {
printf("Complete multipart upload from file failed\n");
}

//销毁内存池
cos_pool_destroy(p);
```

终止分块上传

功能说明

终止一个分块上传操作并删除已上传的块。当您调用 Abort Multipart Upload 时，如果有正在使用这个 Upload Parts 上传块的请求，则 Upload Parts 会返回失败。

方法原型

```
cos_status_t *cos_abort_multipart_upload(const cos_request_options_t *options,
const cos_string_t *bucket,
const cos_string_t *object,
cos_string_t *upload_id,
cos_table_t **resp_headers);
```

参数说明

参数名称	参数描述	类型
options	COS 请求选项	Struct
bucket	存储桶名称，存储桶的命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String
object	Object 名称	String
upload_id	上传任务编号	String
resp_headers	返回 HTTP 响应消息的头域	Struct

返回结果说明

返回结果	描述	类型
code	错误码	Int
error_code	错误码内容	String
error_msg	错误码描述	String
req_id	请求消息 ID	String

示例

```
cos_pool_t *p = NULL;
cos_string_t bucket;
cos_string_t object;
int is_cname = 0;
```

```
cos_table_t *headers = NULL;
cos_table_t *resp_headers = NULL;
cos_request_options_t *options = NULL;
cos_string_t upload_id;
cos_status_t *s = NULL;

//创建内存池 & 初始化请求选项
cos_pool_create(&p, NULL);
options = cos_request_options_create(p);
init_test_request_options(options, is_cname);
headers = cos_table_make(p, 1);
cos_str_set(&bucket, TEST_BUCKET_NAME);
cos_str_set(&object, TEST_MULTIPART_OBJECT);

//初始化分块上传
s = cos_init_multipart_upload(options, &bucket, &object,
&upload_id, headers, &resp_headers);

if (cos_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");
cos_pool_destroy(p);
return;
}

//终止分块上传
s = cos_abort_multipart_upload(options, &bucket, &object, &upload_id,
&resp_headers);

if (cos_status_is_ok(s)) {
printf("Abort multipart upload succeeded, upload_id::%.*s\n",
upload_id.len, upload_id.data);
} else {
printf("Abort multipart upload failed\n");
}

//销毁内存池
cos_pool_destroy(p);
```

其他操作

设置对象 ACL

功能说明

设置存储桶中某个对象的访问控制列表。

方法原型

```
cos_status_t *cos_put_object_acl(const cos_request_options_t *options,
const cos_string_t *bucket,
const cos_string_t *object,
cos_acl_e cos_acl,
const cos_string_t *grant_read,
const cos_string_t *grant_write,
const cos_string_t *grant_full_ctrl,
cos_table_t **resp_headers);
```

参数说明

参数名称	参数描述	类型
options	COS 请求选项	Struct
bucket	存储桶名称，存储桶的命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String
object	Object 名称	String

参数名称	参数描述	类型
cos_acl	允许用户自定义权限。 有效值：COS_ACL_PRIVATE(0)，COS_ACL_PUBLIC_READ(1) 默认值：COS_ACL_PRIVATE(0)	Enum
grant_read	读权限授予者	String
grant_write	写权限授予者	String
grant_full_ctrl	读写权限授予者	String
resp_headers	返回 HTTP 响应消息的头域	Struct

返回结果说明

返回结果	描述	类型
code	错误码	Int
error_code	错误码内容	String
error_msg	错误码描述	String
req_id	请求消息 ID	String

示例

```
cos_pool_t *p = NULL;
int is_cname = 0;
cos_status_t *s = NULL;
cos_request_options_t *options = NULL;
cos_string_t bucket;
cos_string_t object;
cos_table_t *resp_headers = NULL;

//创建内存池
cos_pool_create(&p, NULL);

//初始化请求选项
options = cos_request_options_create(p);
options->config = cos_config_create(options->pool);
init_test_config(options->config, is_cname);
cos_str_set(&options->config->endpoint, TEST_COS_ENDPOINT);
cos_str_set(&options->config->access_key_id, TEST_ACCESS_KEY_ID);
cos_str_set(&options->config->access_key_secret, TEST_ACCESS_KEY_SECRET);
cos_str_set(&options->config->appid, TEST_APPID);
options->config->is_cname = is_cname;
options->ctl = cos_http_controller_create(options->pool, 0);
cos_str_set(&bucket, TEST_BUCKET_NAME);

//设置对象 ACL
cos_str_set(&object, TEST_OBJECT_NAME);
cos_string_t read;
cos_str_set(&read, "id=\"qcs::cam::uin/12345:uin/12345\", id=\"qcs::cam::uin/45678:uin/45678\"");
s = cos_put_object_acl(options, &bucket, &object, cos_acl, &read, NULL, NULL, &resp_headers);
if (cos_status_is_ok(s)) {
printf("put object acl succeeded\n");
} else {
printf("put object acl failed\n");
}

//销毁内存池
cos_pool_destroy(p);
```

查询对象 ACL

功能说明

查询对象的访问控制列表。

方法原型

```
cos_status_t *cos_get_object_acl(const cos_request_options_t *options,
const cos_string_t *bucket,
const cos_string_t *object,
cos_acl_params_t *acl_param,
cos_table_t **resp_headers)
```

参数说明

参数名称	参数描述	类型
options	COS 请求选项	Struct
bucket	存储桶名称，存储桶的命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String
object	Object 名称	String
acl_param	请求操作参数	Struct
owner_id	请求操作返回的 Bucket 持有者 ID	String
owner_name	请求操作返回的 Bucket 持有者的名称	String
object_list	请求操作返回的被授权者信息与权限信息	Struct
type	请求操作返回的被授权者账户类型	String
id	请求操作返回的被授权者用户 ID	String
name	请求操作返回的被授权者用户名称	String
permission	请求操作返回的被授权者权限信息	String
resp_headers	返回 HTTP 响应消息的头域	Struct

返回结果说明

返回结果	描述	类型
code	错误码	Int
error_code	错误码内容	String
error_msg	错误码描述	String
req_id	请求消息 ID	String

示例

```
cos_pool_t *p = NULL;
int is_cname = 0;
cos_status_t *s = NULL;
cos_request_options_t *options = NULL;
cos_string_t bucket;
cos_string_t object;
cos_table_t *resp_headers = NULL;

//创建内存池
cos_pool_create(&p, NULL);

//初始化请求选项
options = cos_request_options_create(p);
options->config = cos_config_create(options->pool);
init_test_config(options->config, is_cname);
cos_str_set(&options->config->endpoint, TEST_COS_ENDPOINT);
cos_str_set(&options->config->access_key_id, TEST_ACCESS_KEY_ID);
cos_str_set(&options->config->access_key_secret, TEST_ACCESS_KEY_SECRET);
cos_str_set(&options->config->appid, TEST_APPID);
options->config->is_cname = is_cname;
options->ctl = cos_http_controller_create(options->pool, 0);
```

```
cos_str_set(&bucket, TEST_BUCKET_NAME);

//获取对象 ACL
cos_acl_params_t *acl_params2 = NULL;
acl_params2 = cos_create_acl_params(p);
s = cos_get_object_acl(options, &bucket, &object, acl_params2, &resp_headers);
if (cos_status_is_ok(s)) {
printf("get object acl succeeded\n");
printf("acl owner id:%s, name:%s\n", acl_params2->owner_id.data, acl_params2->owner_name.data);
acl_content = NULL;
cos_list_for_each_entry(cos_acl_grantee_content_t, acl_content, &acl_params2->grantee_list, node) {
printf("acl grantee id:%s, name:%s, permission:%s\n", acl_content->id.data, acl_content->name.data, acl_content->permission.data);
}
} else {
printf("get object acl failed\n");
}

//销毁内存池
cos_pool_destroy(p);
```

存储桶管理

简介

本文档提供关于跨域访问、生命周期、版本控制、跨地域复制相关的 API 概览以及 SDK 示例代码。

跨域访问

API	操作名	操作描述
PUT Bucket cors	设置跨域配置	设置存储桶的跨域访问权限
GET Bucket cors	查询跨域配置	查询存储桶的跨域访问配置信息
DELETE Bucket cors	删除跨域配置	删除存储桶的跨域访问配置信息

版本控制

API	操作名	操作描述
PUT Bucket versioning	设置版本控制	设置存储桶的版本控制功能
GET Bucket versioning	查询版本控制	查询存储桶的版本控制信息

跨地域复制

API	操作名	操作描述
PUT Bucket replication	设置跨地域复制	设置存储桶的跨地域复制规则
GET Bucket replication	查询跨地域复制	查询存储桶的跨地域复制规则
DELETE Bucket replication	删除跨地域复制	删除存储桶的跨地域复制规则

跨域访问

设置跨域配置

功能说明

设置存储桶的跨域访问权限。

方法原型

```
cos_status_t *cos_put_bucket_cors(const cos_request_options_t *options,
const cos_string_t *bucket,
cos_list_t *cors_rule_list,
cos_table_t **resp_headers);
```

参数说明

参数名称	参数描述	类型
options	COS 请求选项	Struct
bucket	存储桶名称，存储桶的命名格式为 BucketName-APPID ，此处填写的存储桶名称必须为此格式	String
cors_rule_list	存储桶跨域配置信息	Struct
id	配置规则 ID	String
allowed_origin	允许的访问来源，支持通配符`*`	String
allowed_method	允许的 HTTP 操作，枚举值：GET，PUT，HEAD，POST，DELETE	String
allowed_header	在发送 OPTIONS 请求时告知服务端，接下来的请求可以使用哪些自定义的 HTTP 请求头部，支持通配符`*`	String
expose_header	设置浏览器可以接收到的来自服务器端的自定义头部信息	String
max_age_seconds	设置 OPTIONS 请求得到结果的有效期	Int
resp_headers	返回 HTTP 响应消息的头域	Struct

返回结果说明

返回结果	描述	类型
code	错误码	Int
error_code	错误码内容	String
error_msg	错误码描述	String
req_id	请求消息 ID	String

示例

```
cos_pool_t *p = NULL;
int is_cname = 0;
cos_status_t *s = NULL;
cos_request_options_t *options = NULL;
cos_string_t bucket;
cos_table_t *resp_headers = NULL;

//创建内存池
cos_pool_create(&p, NULL);

//初始化请求选项
options = cos_request_options_create(p);
options->config = cos_config_create(options->pool);
init_test_config(options->config, is_cname);
cos_str_set(&options->config->endpoint, TEST_COS_ENDPOINT);
cos_str_set(&options->config->access_key_id, TEST_ACCESS_KEY_ID);
cos_str_set(&options->config->access_key_secret, TEST_ACCESS_KEY_SECRET);
cos_str_set(&options->config->appid, TEST_APPID);
options->config->is_cname = is_cname;
options->ctl = cos_http_controller_create(options->pool, 0);
cos_str_set(&bucket, TEST_BUCKET_NAME);

//设置跨域配置信息
cos_list_t rule_list;
cos_list_init(&rule_list);
cos_cors_rule_content_t *rule_content = NULL;
```

```
rule_content = cos_create_cors_rule_content(p);
cos_str_set(&rule_content->id, "testrule1");
cos_str_set(&rule_content->allowed_origin, "http://imgcache.finance.cloud.tencent.com:80www.qq1.com");
cos_str_set(&rule_content->allowed_method, "GET");
cos_str_set(&rule_content->allowed_header, "*");
cos_str_set(&rule_content->expose_header, "xxx");
rule_content->max_age_seconds = 3600;
cos_list_add_tail(&rule_content->node, &rule_list);

rule_content = cos_create_cors_rule_content(p);
cos_str_set(&rule_content->id, "testrule2");
cos_str_set(&rule_content->allowed_origin, "http://imgcache.finance.cloud.tencent.com:80www.qq2.com");
cos_str_set(&rule_content->allowed_method, "GET");
cos_str_set(&rule_content->allowed_header, "*");
cos_str_set(&rule_content->expose_header, "yyy");
rule_content->max_age_seconds = 7200;
cos_list_add_tail(&rule_content->node, &rule_list);

rule_content = cos_create_cors_rule_content(p);
cos_str_set(&rule_content->id, "testrule3");
cos_str_set(&rule_content->allowed_origin, "http://imgcache.finance.cloud.tencent.com:80www.qq3.com");
cos_str_set(&rule_content->allowed_method, "GET");
cos_str_set(&rule_content->allowed_header, "*");
cos_str_set(&rule_content->expose_header, "zzz");
rule_content->max_age_seconds = 60;
cos_list_add_tail(&rule_content->node, &rule_list);

//设置跨域配置
s = cos_put_bucket_cors(options, &bucket, &rule_list, &resp_headers);
if (cos_status_is_ok(s)) {
printf("put bucket cors succeeded\n");
} else {
printf("put bucket cors failed\n");
}

//销毁内存池
cos_pool_destroy(p);
```

查询跨域配置

功能说明

查询存储桶的跨域访问配置信息。

方法原型

```
cos_status_t *cos_get_bucket_cors(const cos_request_options_t *options,
const cos_string_t *bucket,
cos_list_t *cors_rule_list,
cos_table_t **resp_headers);
```

参数说明

参数名称	参数描述	类型
options	COS 请求选项	Struct
bucket	存储桶名称，存储桶的命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String
cors_rule_list	存储桶跨域配置信息	Struct
id	配置规则 ID	String
allowed_origin	允许的访问来源，支持通配符`*`	String
allowed_method	允许的 HTTP 操作，枚举值：GET，PUT，HEAD，POST，DELETE	String
allowed_header	在发送 OPTIONS 请求时告知服务端，接下来的请求可以使用哪些自定义的 HTTP 请求头部，支持通配符`*`	String
expose_header	设置浏览器可以接收到的来自服务器端的自定义头部信息	String

参数名称	参数描述	类型
max_age_seconds	设置 OPTIONS 请求得到结果的有效期	Int
resp_headers	返回 HTTP 响应消息的头域	Struct

返回结果说明

返回结果	描述	类型
code	错误码	Int
error_code	错误码内容	String
error_msg	错误码描述	String
req_id	请求消息 ID	String

示例

```
cos_pool_t *p = NULL;
int is_cname = 0;
cos_status_t *s = NULL;
cos_request_options_t *options = NULL;
cos_string_t bucket;
cos_table_t *resp_headers = NULL;

//创建内存池
cos_pool_create(&p, NULL);

//初始化请求选项
options = cos_request_options_create(p);
options->config = cos_config_create(options->pool);
init_test_config(options->config, is_cname);
cos_str_set(&options->config->endpoint, TEST_COS_ENDPOINT);
cos_str_set(&options->config->access_key_id, TEST_ACCESS_KEY_ID);
cos_str_set(&options->config->access_key_secret, TEST_ACCESS_KEY_SECRET);
cos_str_set(&options->config->appid, TEST_APPID);
options->config->is_cname = is_cname;
options->ctl = cos_http_controller_create(options->pool, 0);
cos_str_set(&bucket, TEST_BUCKET_NAME);

//获取跨域配置
cos_list_t rule_list_ret;
cos_list_init(&rule_list_ret);
s = cos_get_bucket_cors(options, &bucket, &rule_list_ret, &resp_headers);
if (cos_status_is_ok(s)) {
printf("get bucket cors succeeded\n");
cos_cors_rule_content_t *content = NULL;
cos_list_for_each_entry(cos_cors_rule_content_t, content, &rule_list_ret, node) {
printf("cors id:%s, allowed_origin:%s, allowed_method:%s, allowed_header:%s, expose_header:%s, max_age_seconds:%d\n",
content->id.data, content->allowed_origin.data, content->allowed_method.data, content->allowed_header.data, content->expose_header.data, content->
max_age_seconds);
}
} else {
printf("get bucket cors failed\n");
}

//销毁内存池
cos_pool_destroy(p);
```

删除跨域配置

功能说明

删除存储桶的跨域访问配置信息。

方法原型

```
cos_status_t *cos_delete_bucket_cors(const cos_request_options_t *options,
const cos_string_t *bucket,
cos_table_t **resp_headers);
```

参数说明

参数名称	参数描述	类型
options	COS 请求选项	Struct
bucket	存储桶名称，存储桶的命名格式为 BucketName-APPID ，此处填写的存储桶名称必须为此格式	String
resp_headers	返回 HTTP 响应消息的头域	Struct

返回结果说明

返回结果	描述	类型
code	错误码	Int
error_code	错误码内容	String
error_msg	错误码描述	String
req_id	请求消息 ID	String

示例

```
cos_pool_t *p = NULL;
int is_cname = 0;
cos_status_t *s = NULL;
cos_request_options_t *options = NULL;
cos_string_t bucket;
cos_table_t *resp_headers = NULL;

//创建内存池
cos_pool_create(&p, NULL);

//初始化请求选项
options = cos_request_options_create(p);
options->config = cos_config_create(options->pool);
init_test_config(options->config, is_cname);
cos_str_set(&options->config->endpoint, TEST_COS_ENDPOINT);
cos_str_set(&options->config->access_key_id, TEST_ACCESS_KEY_ID);
cos_str_set(&options->config->access_key_secret, TEST_ACCESS_KEY_SECRET);
cos_str_set(&options->config->appid, TEST_APPID);
options->config->is_cname = is_cname;
options->ctl = cos_http_controller_create(options->pool, 0);
cos_str_set(&bucket, TEST_BUCKET_NAME);

//删除跨域配置
s = cos_delete_bucket_cors(options, &bucket, &resp_headers);
if (cos_status_is_ok(s)) {
printf("delete bucket cors succeeded\n");
} else {
printf("delete bucket cors failed\n");
}

//销毁内存池
cos_pool_destroy(p);
```

版本控制

设置版本控制

功能说明

设置指定存储桶的版本控制功能。

方法原型

```
cos_status_t *cos_put_bucket_versioning(const cos_request_options_t *options,
const cos_string_t *bucket,
cos_versioning_content_t *versioning,
cos_table_t **resp_headers);
```

参数说明

参数名称	参数描述	类型
options	COS 请求选项	Struct
bucket	存储桶名称，存储桶的命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String
versioning	版本控制请求操作参数	Struct
status	版本是否开启，枚举值：Suspended，Enabled	String
resp_headers	返回 HTTP 响应消息的头域	Struct

返回结果说明

返回结果	描述	类型
code	错误码	Int
error_code	错误码内容	String
error_msg	错误码描述	String
req_id	请求消息 ID	String

示例

开启版本控制

```
cos_pool_t *p = NULL;
int is_cname = 0;
cos_status_t *s = NULL;
cos_request_options_t *options = NULL;
cos_string_t bucket;
cos_table_t *resp_headers = NULL;

//创建内存池
cos_pool_create(&p, NULL);

//初始化请求选项
options = cos_request_options_create(p);
options->config = cos_config_create(options->pool);
init_test_config(options->config, is_cname);
cos_str_set(&options->config->endpoint, TEST_COS_ENDPOINT);
cos_str_set(&options->config->access_key_id, TEST_ACCESS_KEY_ID);
cos_str_set(&options->config->access_key_secret, TEST_ACCESS_KEY_SECRET);
cos_str_set(&options->config->appid, TEST_APPID);
options->config->is_cname = is_cname;
options->ctl = cos_http_controller_create(options->pool, 0);
cos_str_set(&bucket, TEST_BUCKET_NAME);

//put bucket versioning
cos_versioning_content_t *versioning = NULL;
versioning = cos_create_versioning_content(p);
cos_str_set(&versioning->status, "Enabled");
s = cos_put_bucket_versioning(options, &bucket, versioning, &resp_headers);
if (cos_status_is_ok(s)) {
printf("put bucket versioning succeeded\n");
} else {
printf("put bucket versioning failed\n");
```

```
}

//销毁内存池
cos_pool_destroy(p);
```

暂停版本控制

```
cos_pool_t *p = NULL;
int is_cname = 0;
cos_status_t *s = NULL;
cos_request_options_t *options = NULL;
cos_string_t bucket;
cos_table_t *resp_headers = NULL;

//创建内存池
cos_pool_create(&p, NULL);

//初始化请求选项
options = cos_request_options_create(p);
options->config = cos_config_create(options->pool);
init_test_config(options->config, is_cname);
cos_str_set(&options->config->endpoint, TEST_COS_ENDPOINT);
cos_str_set(&options->config->access_key_id, TEST_ACCESS_KEY_ID);
cos_str_set(&options->config->access_key_secret, TEST_ACCESS_KEY_SECRET);
cos_str_set(&options->config->appid, TEST_APPID);
options->config->is_cname = is_cname;
options->ctl = cos_http_controller_create(options->pool, 0);
cos_str_set(&bucket, TEST_BUCKET_NAME);

//put bucket versioning
cos_versioning_content_t *versioning = NULL;
versioning = cos_create_versioning_content(p);
cos_str_set(&versioning->status, "Suspended");
s = cos_put_bucket_versioning(options, &bucket, versioning, &resp_headers);
if (cos_status_is_ok(s)) {
printf("put bucket versioning succeeded\n");
} else {
printf("put bucket versioning failed\n");
}

//销毁内存池
cos_pool_destroy(p);
```

查询版本控制

功能说明

查询指定存储桶的版本控制信息。

方法原型

```
cos_status_t *cos_get_bucket_versioning(const cos_request_options_t *options,
const cos_string_t *bucket,
cos_versioning_content_t *versioning,
cos_table_t **resp_headers);
```

参数说明

参数名称	参数描述	类型
options	COS 请求选项	Struct
bucket	存储桶名称，存储桶的命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String
versioning	版本控制请求操作参数	Struct
status	版本是否开启，枚举值：Suspended，Enabled	String
resp_headers	返回 HTTP 响应消息的头域	Struct

返回结果说明

返回结果	描述	类型
code	错误码	Int
error_code	错误码内容	String
error_msg	错误码描述	String
req_id	请求消息 ID	String

示例

```
cos_pool_t *p = NULL;
int is_cname = 0;
cos_status_t *s = NULL;
cos_request_options_t *options = NULL;
cos_string_t bucket;
cos_table_t *resp_headers = NULL;

//创建内存池
cos_pool_create(&p, NULL);

//初始化请求选项
options = cos_request_options_create(p);
options->config = cos_config_create(options->pool);
init_test_config(options->config, is_cname);
cos_str_set(&options->config->endpoint, TEST_COS_ENDPOINT);
cos_str_set(&options->config->access_key_id, TEST_ACCESS_KEY_ID);
cos_str_set(&options->config->access_key_secret, TEST_ACCESS_KEY_SECRET);
cos_str_set(&options->config->appid, TEST_APPID);
options->config->is_cname = is_cname;
options->ctl = cos_http_controller_create(options->pool, 0);
cos_str_set(&bucket, TEST_BUCKET_NAME);

//get bucket versioning
cos_versioning_content_t *versioning = NULL;
versioning = cos_create_versioning_content(p);
s = cos_get_bucket_versioning(options, &bucket, versioning, &resp_headers);
if (cos_status_is_ok(s)) {
printf("put bucket versioning succeeded\n");
printf("bucket versioning status: %s\n", versioning->status.data);
} else {
printf("put bucket versioning failed\n");
}

//销毁内存池
cos_pool_destroy(p);
```

跨地域复制

设置跨地域复制

功能说明

设置指定存储桶的跨地域复制规则。

方法原型

```
cos_status_t *cos_put_bucket_replication(const cos_request_options_t *options,
const cos_string_t *bucket,
cos_replication_params_t *replication_param,
cos_table_t **resp_headers);
```

参数说明

参数名称	参数描述	类型

参数名称	参数描述	类型
options	COS 请求选项。	Struct
bucket	存储桶名称，存储桶的命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String
replication_param	跨地域复制请求操作参数	Struct
role	操作者账户信息	String
rule_list	跨地域复制规则配置信息	Struct
id	用来标注具体规则的名称	String
status	标识规则是否生效，枚举值：Enabled，Disabled	String
prefix	匹配前缀。不可重叠，重叠返回错误	String
dst_bucket	目的存储桶标识，格式为：资源标识符`qcs::cos:[region]::[bucketname-AppId]`	String
resp_headers	返回 HTTP 响应消息的头域	Struct

返回结果说明

返回结果	描述	类型
code	错误码	Int
error_code	错误码内容	String
error_msg	错误码描述	String
req_id	请求消息 ID	String

示例

```
cos_pool_t *p = NULL;
int is_cname = 0;
cos_status_t *s = NULL;
cos_request_options_t *options = NULL;
cos_string_t bucket;
cos_table_t *resp_headers = NULL;

//创建内存池
cos_pool_create(&p, NULL);

//初始化请求选项
options = cos_request_options_create(p);
options->config = cos_config_create(options->pool);
init_test_config(options->config, is_cname);
cos_str_set(&options->config->endpoint, TEST_COS_ENDPOINT);
cos_str_set(&options->config->access_key_id, TEST_ACCESS_KEY_ID);
cos_str_set(&options->config->access_key_secret, TEST_ACCESS_KEY_SECRET);
cos_str_set(&options->config->appid, TEST_APPID);
options->config->is_cname = is_cname;
options->ctl = cos_http_controller_create(options->pool, 0);
cos_str_set(&bucket, TEST_BUCKET_NAME);

//设置跨地域复制规则
cos_replication_params_t *replication_param = NULL;
replication_param = cos_create_replication_params(p);
cos_str_set(&replication_param->role, "qcs::cam::uin/100000616666:uin/100000616666");

cos_replication_rule_content_t *rule = NULL;
rule = cos_create_replication_rule_content(p);
cos_str_set(&rule->id, "Rule_01");
cos_str_set(&rule->status, "Enabled");
cos_str_set(&rule->prefix, "test1");
cos_str_set(&rule->dst_bucket, "qcs::cos:ap-shanghai::replicationtest-1253686666");
cos_list_add_tail(&rule->node, &replication_param->rule_list);
```

```
rule = cos_create_replication_rule_content(p);
cos_str_set(&rule->id, "Rule_02");
cos_str_set(&rule->status, "Disabled");
cos_str_set(&rule->prefix, "test2");
cos_str_set(&rule->dst_bucket, "qcs::cos:ap-shanghai::replicationtest-1253686666");
cos_list_add_tail(&rule->node, &replication_param->rule_list);

rule = cos_create_replication_rule_content(p);
cos_str_set(&rule->id, "Rule_03");
cos_str_set(&rule->status, "Enabled");
cos_str_set(&rule->prefix, "test3");
cos_str_set(&rule->dst_bucket, "qcs::cos:ap-shanghai::replicationtest-1253686666");
cos_list_add_tail(&rule->node, &replication_param->rule_list);

s = cos_put_bucket_replication(options, &bucket, replication_param, &resp_headers);
if (cos_status_is_ok(s)) {
printf("put bucket replication succeeded\n");
} else {
printf("put bucket replication failed\n");
}

//销毁内存池
cos_pool_destroy(p);
```

查询跨地域复制

功能说明

查询指定存储桶的跨地域复制规则。

方法原型

```
cos_status_t *cos_get_bucket_replication(const cos_request_options_t *options,
const cos_string_t *bucket,
cos_replication_params_t *replication_param,
cos_table_t **resp_headers);
```

参数说明

参数名称	参数描述	类型
options	COS 请求选项	Struct
bucket	存储桶名称，存储桶的命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String
replication_param	跨地域复制请求操作参数	Struct
role	操作者账户信息	String
rule_list	跨地域复制规则配置信息	Struct
id	用来标注具体规则的名称	String
status	标识规则是否生效，枚举值：Enabled，Disabled	String
prefix	匹配前缀。不可重叠，重叠返回错误	String
dst_bucket	目的存储桶标识，格式为：资源标识符`qcs::cos:[region]::[bucketname-AppId]`	String
resp_headers	返回 HTTP 响应消息的头域	Struct

返回结果说明

返回结果	描述	类型
code	错误码	Int
error_code	错误码内容	String
error_msg	错误码描述	String

返回结果	描述	类型
req_id	请求消息 ID	String

示例

```
cos_pool_t *p = NULL;
int is_cname = 0;
cos_status_t *s = NULL;
cos_request_options_t *options = NULL;
cos_string_t bucket;
cos_table_t *resp_headers = NULL;

//创建内存池
cos_pool_create(&p, NULL);

//初始化请求选项
options = cos_request_options_create(p);
options->config = cos_config_create(options->pool);
init_test_config(options->config, is_cname);
cos_str_set(&options->config->endpoint, TEST_COS_ENDPOINT);
cos_str_set(&options->config->access_key_id, TEST_ACCESS_KEY_ID);
cos_str_set(&options->config->access_key_secret, TEST_ACCESS_KEY_SECRET);
cos_str_set(&options->config->appid, TEST_APPID);
options->config->is_cname = is_cname;
options->ctl = cos_http_controller_create(options->pool, 0);
cos_str_set(&bucket, TEST_BUCKET_NAME);

//查询跨地域复制规则
cos_replication_params_t *replication_param2 = NULL;
replication_param2 = cos_create_replication_params(p);
s = cos_get_bucket_replication(options, &bucket, replication_param2, &resp_headers);

if (cos_status_is_ok(s)) {
printf("get bucket replication succeeded\n");
printf("ReplicationConfiguration role: %s\n", replication_param2->role.data);
cos_replication_rule_content_t *content = NULL;
cos_list_for_each_entry(cos_replication_rule_content_t, content, &replication_param2->rule_list, node) {
printf("ReplicationConfiguration rule, id:%s, status:%s, prefix:%s, dst_bucket:%s, storage_class:%s\n",
content->id.data, content->status.data, content->prefix.data, content->dst_bucket.data, content->storage_class.data);
}
} else {
printf("get bucket replication failed\n");
}

//销毁内存池
cos_pool_destroy(p);
```

删除跨地域复制

功能说明

删除指定存储桶的跨地域复制规则。

方法原型

```
cos_status_t *cos_delete_bucket_replication(const cos_request_options_t *options,
const cos_string_t *bucket,
cos_table_t **resp_headers);
```

参数说明

参数名称	参数描述	类型
options	COS 请求选项	Struct
bucket	存储桶名称，存储桶的命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String
resp_headers	返回 HTTP 响应消息的头域	Struct

返回结果说明

返回结果	描述	类型
code	错误码	Int
error_code	错误码内容	String
error_msg	错误码描述	String
req_id	请求消息 ID	String

示例

```
cos_pool_t *p = NULL;
int is_cname = 0;
cos_status_t *s = NULL;
cos_request_options_t *options = NULL;
cos_string_t bucket;
cos_table_t *resp_headers = NULL;

//创建内存池
cos_pool_create(&p, NULL);

//初始化请求选项
options = cos_request_options_create(p);
options->config = cos_config_create(options->pool);
init_test_config(options->config, is_cname);
cos_str_set(&options->config->endpoint, TEST_COS_ENDPOINT);
cos_str_set(&options->config->access_key_id, TEST_ACCESS_KEY_ID);
cos_str_set(&options->config->access_key_secret, TEST_ACCESS_KEY_SECRET);
cos_str_set(&options->config->appid, TEST_APPID);
options->config->is_cname = is_cname;
options->ctl = cos_http_controller_create(options->pool, 0);
cos_str_set(&bucket, TEST_BUCKET_NAME);

//删除跨地域复制规则
s = cos_delete_bucket_replication(options, &bucket, &resp_headers);
if (cos_status_is_ok(s)) {
printf("delete bucket replication succeeded\n");
} else {
printf("delete bucket replication failed\n");
}

//销毁内存池
cos_pool_destroy(p);
```

预签名 URL

简介

C SDK 提供获取请求预签名 URL 接口，详细操作请查看本文说明和示例。

获取请求预签名 URL

生成请求预签名URL

功能说明

该接口用于生成请求预签名 URL。

方法原型

```
int cos_gen_presigned_url(const cos_request_options_t *options,
const cos_string_t *bucket,
const cos_string_t *object,
const int64_t expire,
http_method_e method,
cos_string_t *presigned_url);
```

参数说明

参数名称	参数描述	类型
options	COS 请求选项	Struct
bucket	存储桶名称，存储桶的命名格式为 BucketName-APPID ，此处填写的存储桶名称必须为此格式	String
object	Object 名称	String
expire	签名有效时间，单位为秒	Int
method	HTTP 请求方法枚举类型，分别为 HTTP_GET、HTTP_HEAD、HTTP_PUT、HTTP_POST、HTTP_DELETE	Enum
presigned_url	生成的请求预签名 URL	String

返回结果说明

返回结果	描述	类型
code	错误码	Int

预签名请求示例

可在 options 参数中设置永久密钥或临时密钥来获取预签名 URL。

```
cos_pool_t *p = NULL;
int is_cname = 0;
cos_request_options_t *options = NULL;
cos_string_t bucket;
cos_string_t object;
cos_string_t presigned_url;

//create memory pool
cos_pool_create(&p, NULL);

//init request options
options = cos_request_options_create(p);
options->config = cos_config_create(options->pool);
init_test_config(options->config, is_cname);
cos_str_set(&options->config->endpoint, TEST_COS_ENDPOINT);
cos_str_set(&options->config->access_key_id, TEST_ACCESS_KEY_ID);
cos_str_set(&options->config->access_key_secret, TEST_ACCESS_KEY_SECRET);
/* 可以通过设置 sts_token 来使用临时密钥，当使用临时密钥时，access_key_id 和 access_key_secret 均需要设置为对应临时密钥所配套的 SecretId 和 SecretKey */
//cos_str_set(&options->config->sts_token, "MyTokenString");
cos_str_set(&options->config->appid, TEST_APPID);
options->config->is_cname = is_cname;
options->ctl = cos_http_controller_create(options->pool, 0);
cos_str_set(&bucket, TEST_BUCKET_NAME);
cos_str_set(&object, TEST_OBJECT_NAME);

//generate presigned URL
cos_gen_presigned_url(options, &bucket, &object, 300, HTTP_GET, &presigned_url);
printf("presigned url: %s\n", presigned_url.data);

//destroy memory pool
cos_pool_destroy(p);
```

异常处理

简介

SDK 调用失败时，结果信息包含在 API 返回的 cos_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。

```
cos_status_t *s = NULL;
s = cos_put_object_from_file(options, &bucket, &object, &file, &headers, &resp_headers);
if (!s && !cos_status_is_ok(s)) {
// 按需要进行异常场景的日志输出和处理
cos_warn_log("failed to put object from file", buf);
if (s->error_code) cos_warn_log("status->error_code: %s", s->error_code);
if (s->error_msg) cos_warn_log("status->error_msg: %s", s->error_msg);
if (s->req_id) cos_warn_log("status->req_id: %s", s->req_id);
}
```

客户端异常

当 cos_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 cos_error_code_e 定义。

```
typedef enum {
COSE_OK = 0,
COSE_OUT_MEMORY = -1000,
COSE_OVER_MEMORY = -999,
COSE_FAILED_CONNECT = -998,
COSE_ABORT_CALLBACK = -997,
COSE_INTERNAL_ERROR = -996,
COSE_REQUEST_TIMEOUT = -995,
COSE_INVALID_ARGUMENT = -994,
COSE_INVALID_OPERATION = -993,
COSE_CONNECTION_FAILED = -992,
COSE_FAILED_INITIALIZE = -991,
COSE_NAME_LOOKUP_ERROR = -990,
COSE_FAILED_VERIFICATION = -989,
COSE_WRITE_BODY_ERROR = -988,
COSE_READ_BODY_ERROR = -987,
COSE_SERVICE_ERROR = -986,
COSE_OPEN_FILE_ERROR = -985,
COSE_FILE_SEEK_ERROR = -984,
COSE_FILE_INFO_ERROR = -983,
COSE_FILE_READ_ERROR = -982,
COSE_FILE_WRITE_ERROR = -981,
COSE_XML_PARSE_ERROR = -980,
COSE_UTF8_ENCODE_ERROR = -979,
COSE_CRC_INCONSISTENT_ERROR = -978,
COSE_FILE_FLUSH_ERROR = -977,
COSE_FILE_TRUNC_ERROR = -976,
COSE_UNKNOWN_ERROR = -100
} cos_error_code_e;
```

服务端异常

当 cos_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 cos_status_t 结构的描述：

cos_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,详情请参阅 [错误码] 文档	Int
error_code	请求失败时 body 返回的 Error Code，详情请参阅 [错误码] 文档	String

cos_status_t 成员	描述	类型
error_msg	请求失败时 body 返回的 Error Message，详情请参阅 [错误码] 文档	String
req_id	请求 ID，用于标识用户唯一请求	String

C++ SDK
快速入门

最近更新时间: 2025-01-15 17:01:00

下载与安装

相关资源

- 对象存储 COS 的 XML C++ SDK 源码下载地址： XML C++ SDK 。
- 示例 Demo 下载地址： COS XML C++ SDK 示例。

环境依赖

- COS XML C++ SDK 支持 Linux，不支持 Windows 系统。
- 依赖静态库：jsoncpp boost_system boost_thread Poco （在 lib 文件夹下）。
- 依赖动态库：ssl crypto rt z（需要安装）。

SDK 中提供了 JsonCpp 的库以及头文件。若您想要自行安装，请先按照以下步骤安装库并编译完成后，替换 SDK 中相应的库和头文件即可。若以上库已安装到系统，也可删除 SDK 中相应的库和头文件。

安装 SDK

1. 安装 CMake 工具

```
yum install -y gcc gcc-c++ make automake
//安装 gcc 等必备程序包（已安装则略过此步）
yum install -y wget

// cmake 版本要大于3.5
wget http://imgcache.finance.cloud.tencent.com:80cmake.org/files/v3.5/cmake-3.5.2.tar.gz
tar -zxvf cmake-3.5.2.tar.gz
cd cmake-3.5.2
./bootstrap --prefix=/usr
gmake
gmake install
```

2. 安装 Boost 的库和头文件

```
wget http://imgcache.finance.cloud.tencent.com:80sourceforge.net/projects/boost/files/boost/1.54.0/boost_1_54_0.tar.gz
tar -xzvf boost_1_54_0.tar.gz
cd boost_1_54_0
./bootstrap.sh --prefix=/usr/local
./b2 install --with=all
#Boost 库被安装在 /usr/local/lib 目录下
```

3. 安装 OpenSSL

方式一

```
yum install openssl openssl-devel
```

方式二

```
wget http://imgcache.finance.cloud.tencent.com:80www.openssl.org/source/openssl-1.0.1t.tar.gz
tar -xzvf ./openssl-1.0.1t.tar.gz
cd openssl-1.0.1t/
./config --prefix=/usr/local/ssl --openssldir=/usr/local/ssl

cd /usr/local/
ln -s ssl openssl
echo "/usr/local/openssl/lib" >> /etc/ld.so.conf
ldconfig

# 添加头文件/库文件查找路径(可以写入到~/.bashrc中)
```

```
LIBRARY_PATH=/usr/local/ssl/lib/:$LIBRARY_PATH
CPLUS_INCLUDE_PATH=/usr/local/ssl/include/:$CPLUS_INCLUDE_PATH
```

4. 安装 Poco 的库和头文件

从 Poco 官网 获取并安装 Poco 的库和头文件（下载 complete 版本）。

```
./configure --omit=Data/ODBC,Data/MySQL
make
make install
```

您可以通过修改 CMakeList.txt 文件，指定本地 Boost 头文件路径，修改如下语句：

```
SET(BOOST_HEADER_DIR "/root/boost_1_61_0")
```

5. 使动态链接目录生效

装完 Boost 和 Poco 以后，执行如下命令

```
echo "/usr/local/lib" >> /etc/ld.so.conf
ldconfig
```

6. 编译 COS CPP SDK

下载 XML C++ SDK 源码 集成到您的开发环境，然后执行以下命令：

```
cd ${cos-cpp-sdk}
mkdir -p build
cd build
cmake ..
make
```

示例 Demo 里面有常见 API 的例子。生成的 cos_demo 可以直接运行，生成的静态库名称为：libcossdk.a。生成的 libcossdk.a 放到您自己的工程里 lib 路径下，include 目录拷贝到自己的工程的 include 路径下。

开始使用

下面为您介绍如何使用 COS C++ SDK 完成一个基础操作，如初始化客户端、创建存储桶、查询存储桶列表、上传对象、查询对象列表、下载对象和删除对象。

关于文章中出现的 SecretId、SecretKey、Bucket 等名称的含义和获取方式请参见 COS 术语信息。

初始化

配置文件各字段介绍：

> **注意：**
>
> 私有云配置重点关注 DestDomain 以及 IsDomainSameToHost。

```
// V5.4.3 版本之前的 SDK 配置文件请使用"AccessKey"
"SecretId":"COS_SECRETID",
"SecretKey":"COS_SECRETKEY",

// COS 地域
"Region":"Region",

// 访问存储桶完整的域名, 在创建存储桶时, 界面上显示的完整的域名
"DestDomain": "<bucket>-<appid>.cos.<region>.example.com",

// 当设置 DestDomain 时, 该项为 true
"IsDomainSameToHost": true,

// 签名超时时间, 单位：s
"SignExpiredTime":360,

// connect 超时时间, 单位：ms
"ConnectTimeoutInms":6000,
```

```
// http 超时时间, 单位：ms
"HttpTimeoutInms":60000,

// 上传文件分块大小，范围：1MB- 5GB, 默认为1MB
"UploadPartSize":1048576,

// 单文件分块上传线程池大小
"UploadThreadPoolSize":5,

// 下载文件分片大小
"DownloadSliceSize":4194304,

// 单文件下载线程池大小
"DownloadThreadPoolSize":5,

// 异步上传下载线程池大小
"AsynThreadPoolSize":2,

// 日志输出类型，0：不输出，1：输出到屏幕，2：输出到 syslog
"LogoutType":1,

// 日志级别，1：ERR，2：WARN，3：INFO，4：DBG
"LogLevel":3,
```

上传对象

```
#include "cos_api.h"
#include "cos_sys_config.h"
#include "cos_defines.h"

int main(int argc, char *argv[]) {
// 1. 指定配置文件路径，初始化 CosConfig
qcloud_cos::CosConfig config("./config.json");
qcloud_cos::CosAPI cos(config);

// 2. 构造上传文件的请求
// 目的 Bucket 名称, 和配置文件中 DestDomain 的 bucket 相同
std::string bucket_name = "examplebucket-1250000000";
std::string object_name = "exampleobject"; //exampleobject 即为对象键（Key），是对象在存储桶中的唯一标识。例如，在对象的访问域名 examplebucket-1250
000000.cos.ap-guangzhou.myqcloud.com/doc/pic.jpg 中，对象键为 doc/pic.jpg。
// request 的构造函数中需要传入本地文件路径
qcloud_cos::PutObjectByFileReq req(bucket_name, object_name, "/path/to/local/file");
req.SetXCosStorageClass("STANDARD_IA"); // 调用 Set 方法设置元数据等
qcloud_cos::PutObjectByFileResp resp;

// 3. 调用上传文件接口
qcloud_cos::CosResult result = cos.PutObject(req, &resp);

// 4. 处理调用结果
if (result.IsSucc()) {
// 上传文件成功
} else {
// 上传文件失败，可以调用 CosResult 的成员函数输出错误信息，比如 requestID 等
std::cout << "ErrorInfo=" << result.GetErrorInfo() << std::endl;
std::cout << "HttpStatus=" << result.GetHttpStatus() << std::endl;
std::cout << "ErrorCode=" << result.GetErrorCode() << std::endl;
std::cout << "ErrorMsg=" << result.GetErrorMsg() << std::endl;
std::cout << "ResourceAddr=" << result.GetResourceAddr() << std::endl;
std::cout << "XCosRequestId=" << result.GetXCosRequestId() << std::endl;
std::cout << "XCosTraceId=" << result.GetXCosTraceId() << std::endl;
}
}
```

查询对象列表

```
#include "cos_api.h"
#include "cos_sys_config.h"
#include "cos_defines.h"
```

```
int main(int argc, char *argv[]) {
// 1. 指定配置文件路径，初始化 CosConfig
qcloud_cos::CosConfig config("./config.json");
qcloud_cos::CosAPI cos(config);

// 2. 构造创建存储桶的请求
// 目的 Bucket 名称, 和配置文件中 DestDomain 的 bucket 相同
std::string bucket_name = "examplebucket-1250000000";
qcloud_cos::GetBucketReq req(bucket_name);
qcloud_cos::GetBucketResp resp;
qcloud_cos::CosResult result = cos.GetBucket(req, &resp);

std::vector<qcloud_cos::Content> cotents = resp.GetContents();
for (std::vector<qcloud_cos::Content>::const_iterator itr = cotents.begin(); itr != cotents.end(); ++itr) {
const qcloud_cos::Content& content = *itr;
std::cout << "key name=" << content.m_key << ", lastmodified ="
<< content.m_last_modified << ", size=" << content.m_size << std::endl;
}

// 3. 处理调用结果
if (result.IsSucc()) {
// 上传文件成功
} else {
// 上传文件失败，可以调用 CosResult 的成员函数输出错误信息，比如 requestID 等
std::cout << "ErrorInfo=" << result.GetErrorInfo() << std::endl;
std::cout << "HttpStatus=" << result.GetHttpStatus() << std::endl;
std::cout << "ErrorCode=" << result.GetErrorCode() << std::endl;
std::cout << "ErrorMsg=" << result.GetErrorMsg() << std::endl;
std::cout << "ResourceAddr=" << result.GetResourceAddr() << std::endl;
std::cout << "XCosRequestId=" << result.GetXCosRequestId() << std::endl;
std::cout << "XCosTraceId=" << result.GetXCosTraceId() << std::endl;
}
}
```

下载对象

```
#include "cos_api.h"
#include "cos_sys_config.h"
#include "cos_defines.h"

int main(int argc, char *argv[]) {
// 1. 指定配置文件路径，初始化 CosConfig
qcloud_cos::CosConfig config("./config.json");
qcloud_cos::CosAPI cos(config);

// 2. 构造创建存储桶的请求
// 目的 Bucket 名称, 和配置文件中 DestDomain 的 bucket 相同
std::string bucket_name = "examplebucket-1250000000";
std::string object_name = "exampleobject"; // exampleobject 即为对象键（Key），是对象在存储桶中的唯一标识。例如，在对象的访问域名 examplebucket-125
0000000.cos.ap-guangzhou.myqcloud.com/doc/pic.jpg 中，对象键为 doc/pic.jpg。
std::string local_path = "/tmp/exampleobject";
// request 需要提供 appid、bucketname、object,以及本地的路径（包含文件名）
qcloud_cos::GetObjectByFileReq req(bucket_name, object_name, local_path);
qcloud_cos::GetObjectByFileResp resp;

// 3. 调用创建存储桶接口
qcloud_cos::CosResult result = cos.GetObject(req, &resp);

// 4. 处理调用结果
if (result.IsSucc()) {
// 下载文件成功
} else {
// 下载文件失败，可以调用 CosResult 的成员函数输出错误信息，例如 requestID 等
std::cout << "ErrorInfo=" << result.GetErrorInfo() << std::endl;
std::cout << "HttpStatus=" << result.GetHttpStatus() << std::endl;
std::cout << "ErrorCode=" << result.GetErrorCode() << std::endl;
std::cout << "ErrorMsg=" << result.GetErrorMsg() << std::endl;
std::cout << "ResourceAddr=" << result.GetResourceAddr() << std::endl;
std::cout << "XCosRequestId=" << result.GetXCosRequestId() << std::endl;
std::cout << "XCosTraceId=" << result.GetXCosTraceId() << std::endl;
```

```
}
}
```

删除对象

```
#include "cos_api.h"
#include "cos_sys_config.h"
#include "cos_defines.h"

int main(int argc, char *argv[]) {
// 1. 指定配置文件路径，初始化 CosConfig
qcloud_cos::CosConfig config("./config.json");
qcloud_cos::CosAPI cos(config);

// 2. 构造创建存储桶的请求
std::string bucket_name = "examplebucket-1250000000"; // 目标 Bucket 名称, 和配置文件中 DestDomain 的 bucket 相同
std::string object_name = "exampleobject"; // exampleobject 即为对象键（Key），是对象在存储桶中的唯一标识。例如，在对象的访问域名 examplebucket-125
0000000.cos.ap-guangzhou.myqcloud.com/doc/pic.jpg 中，对象键为 doc/pic.jpg。
// 3. 调用创建存储桶接口
qcloud_cos::DeleteObjectReq req(bucket_name, object_name);
qcloud_cos::DeleteObjectResp resp;
qcloud_cos::CosResult result = cos.DeleteObject(req, &resp);

// 4. 处理调用结果
if (result.IsSucc()) {
// 下载文件成功
} else {
// 下载文件失败，可以调用 CosResult 的成员函数输出错误信息，例如 requestID 等
std::cout << "ErrorInfo=" << result.GetErrorInfo() << std::endl;
std::cout << "HttpStatus=" << result.GetHttpStatus() << std::endl;
std::cout << "ErrorCode=" << result.GetErrorCode() << std::endl;
std::cout << "ErrorMsg=" << result.GetErrorMsg() << std::endl;
std::cout << "ResourceAddr=" << result.GetResourceAddr() << std::endl;
std::cout << "XCosRequestId=" << result.GetXCosRequestId() << std::endl;
std::cout << "XCosTraceId=" << result.GetXCosTraceId() << std::endl;
}
}
```

接口文档

最近更新时间: 2025-01-15 17:01:00

存储桶操作

简介

本文档提供关于存储桶的基本操作和访问控制列表（ACL）的相关 API 概览以及 SDK 示例代码。

基本操作

API	操作名	操作描述
PUT Bucket	创建存储桶	在指定账号下创建一个存储桶
HEAD Bucket	检索存储桶及其权限	检索存储桶是否存在且是否有权限访问
DELETE Bucket	删除存储桶	删除指定账号下的空存储桶

访问控制列表

API	操作名	操作描述
PUT Bucket acl	设置存储桶 ACL	设置指定存储桶访问权限控制列表
GET Bucket acl	查询存储桶 ACL	查询存储桶的访问控制列表

基本操作

创建存储桶

功能说明

创建一个存储桶（PUT Bucket）。

方法原型

```
CosResult PutBucket(const PutBucketReq& req, PutBucketResp* resp)
```

请求示例

```
qcloud_cos::CosConfig config("./config.json");
qcloud_cos::CosAPI cos(config);

// 在配置文件的 DestDomain 中已经包含了 bucket name, 此处不再填写
qcloud_cos::PutBucketReq req("");
qcloud_cos::PutBucketResp resp;
qcloud_cos::CosResult result = cos.PutBucket(req, &resp);

// 调用成功，调用 resp 的成员函数获取返回内容
if (result.IsSucc()) {
// ...
} else {
// 可以调用 CosResult 的成员函数输出错误信息，如 requestID 等
}
```

参数说明

参数	参数描述
req	PutBucketReq，PutBucket 操作的请求

参数	参数描述
resp	PutBucketResp，PutBucket 操作的返回

PutBucketReq 提供以下成员函数：

```
// 定义 Bucket 的 ACL 属性,有效值：private,public-read-write,public-read
// 默认值：private
void SetXCosAcl(const std::string& str);

// 赋予被授权者读的权限.格式：x-cos-grant-read: id=" ",id=" ".
// 当需要给子账户授权时,id="qcs::cam::uin/<OwnerUin>:uin/<SubUin>"
// 当需要给根账户授权时,id="qcs::cam::uin/<OwnerUin>:uin/<OwnerUin>"
void SetXCosGrantRead(const std::string& str);

// 赋予被授权者写的权限,格式：x-cos-grant-write: id=" ",id=" "./
// 当需要给子账户授权时,id="qcs::cam::uin/<OwnerUin>:uin/<SubUin>",
// 当需要给根账户授权时,id="qcs::cam::uin/<OwnerUin>:uin/<OwnerUin>"
void SetXCosGrantWrite(const std::string& str);

// 赋予被授权者读写权限.格式：x-cos-grant-full-control: id=" ",id=" ".
// 当需要给子账户授权时,id="qcs::cam::uin/<OwnerUin>:uin/<SubUin>",
// 当需要给根账户授权时,id="qcs::cam::uin/<OwnerUin>:uin/<OwnerUin>"
void SetXCosGrantFullControl(const std::string& str);
```

检索存储桶及其权限

功能说明

检索存储桶是否存在且是否有权限访问。

方法原型

```
CosResult HeadBucket(const HeadBucketReq& req, HeadBucketResp* resp)
```

请求示例

```
qcloud_cos::CosConfig config("./config.json");
qcloud_cos::CosAPI cos(config);

// 在配置文件的 DestDomain 中已经包含了 bucket name, 此处不再填写
qcloud_cos::HeadBucketReq req("");
qcloud_cos::HeadBucketResp resp;
qcloud_cos::CosResult result = cos.HeadBucket(req, &resp);

// 调用成功，调用 resp 的成员函数获取返回内容
if (result.IsSucc()) {
// ...
} else {
// 可以调用 CosResult 的成员函数输出错误信息，如 requestID 等
}
```

参数说明

参数	参数描述
req	HeadBucketReq，HeadBucket 操作的请求
resp	HeadBucketResp，HeadBucket 操作的返回

删除存储桶

功能说明

删除指定账号下的空存储桶。

方法原型

```
CosResult DeleteBucket(const DeleteBucketReq& req, DeleteBucketResp* resp)
```

请求示例

```
qcloud_cos::CosConfig config("./config.json");
qcloud_cos::CosAPI cos(config);

// 在配置文件的 DestDomain 中已经包含了 bucket name, 此处不再填写
qcloud_cos::DeleteBucketReq req("");
qcloud_cos::DeleteBucketResp resp;
qcloud_cos::CosResult result = cos.DeleteBucket(req, &resp);

// 调用成功，调用 resp 的成员函数获取返回内容
if (result.IsSucc()) {
// ...
} else {
// 可以调用 CosResult 的成员函数输出错误信息，如 requestID 等
}
```

参数说明

参数	参数描述
req	DeleteBucketReq，DeleteBucket 操作的请求
resp	DeletBucketResp，DeletBucket 操作的返回

访问控制列表

设置存储桶 ACL

功能说明

设置指定存储桶访问权限控制列表。

方法原型

```
CosResult PutBucketACL(const DPutBucketACLReq& req, PutBucketACLResp* resp)
```

请求示例

```
qcloud_cos::CosConfig config("./config.json");
qcloud_cos::CosAPI cos(config);

std::string bucket_name = "examplebucket-1250000000";

// 在配置文件的 DestDomain 中已经包含了 bucket name, 此处不再填写
qcloud_cos::PutBucketACLReq req("");
qcloud_cos::ACLRule rule;
rule.m_id = "123";
rule.m_allowed_headers.push_back("x-cos-meta-test");
rule.m_allowed_origins.push_back("http://imgcache.finance.cloud.tencent.com:80www.qq.com");
rule.m_allowed_origins.push_back("http://imgcache.finance.cloud.tencent.com:80cloud.tentent.com");
rule.m_allowed_methods.push_back("PUT");
rule.m_allowed_methods.push_back("GET");
rule.m_max_age_secs = "600";
rule.m_expose_headers.push_back("x-cos-expose");
req.AddRule(rule);

qcloud_cos::PutBucketACLResp resp;
qcloud_cos::CosResult result = cos.PutBucketACL(req, &resp);

// 调用成功，调用 resp 的成员函数获取返回内容
if (result.IsSucc()) {
// ...
} else {
```

```
// 设置 ACL，可以调用 CosResult 的成员函数输出错误信息，比如 requestID 等
}
```

参数说明

参数	参数描述
req	PutBucketACLReq，PutBucketACL 操作的请求
resp	PutBucketACLResp，PutBucketACL 操作的返回

PutBucketACLReq 提供以下成员函数：

```
// 定义 Bucket 的 ACL 属性,有效值：private,public-read-write,public-read
// 默认值：private
void SetXCosAcl(const std::string& str);

// 赋予被授权者读的权限.格式：x-cos-grant-read: id=" ",id=" ".
// 当需要给子账户授权时,id="qcs::cam::uin/<OwnerUin>:uin/<SubUin>"
// 当需要给根账户授权时,id="qcs::cam::uin/<OwnerUin>:uin/<OwnerUin>"
void SetXCosGrantRead(const std::string& str);

// 赋予被授权者写的权限.格式：x-cos-grant-write: id=" ",id=" "./
// 当需要给子账户授权时,id="qcs::cam::uin/<OwnerUin>:uin/<SubUin>",
// 当需要给根账户授权时,id="qcs::cam::uin/<OwnerUin>:uin/<OwnerUin>"
void SetXCosGrantWrite(const std::string& str);

// 赋予被授权者读写权限.格式：x-cos-grant-full-control: id=" ",id=" ".
// 当需要给子账户授权时,id="qcs::cam::uin/<OwnerUin>:uin/<SubUin>",
// 当需要给根账户授权时,id="qcs::cam::uin/<OwnerUin>:uin/<OwnerUin>"
void SetXCosGrantFullControl(const std::string& str);

// Bucket 持有者 ID
void SetOwner(const Owner& owner);

// 设置被授权者信息与权限信息
void SetAccessControlList(const std::vector<Grant>& grants);

// 添加单个 Bucket 的授权信息
void AddAccessControlList(const Grant& grant);
```

> SetXCosAcl、SetXCosGrantRead、SetXCosGrantWrite、SetXCosGrantFullControl 这类接口与 SetAccessControlList、AddAccessControlList 不可同时使用。因为前者实际是通过设置 HTTP Header 实现，而后者是在 Body 中添加了 XML 格式的内容，二者只能二选一。 SDK 内部优先使用第一类。

ACLRule 定义如下：

```
struct Grantee {
// type 类型可以为 RootAccount， SubAccount
// 当 type 类型为 RootAccount 时，可以在 id 中 uin 填写帐号 ID，也可以用 anyone（指代所有类型用户）代替 uin/<OwnerUin> 和 uin/<SubUin>
// 当 type 类型为 RootAccount 时，uin 代表主账号，Subaccount 代表子账号
std::string m_type;
std::string m_id; // qcs::cam::uin/<OwnerUin>:uin/<SubUin>
std::string m_display_name; // 非必选
std::string m_uri;
};

struct Grant {
Grantee m_grantee; // 被授权者资源信息
std::string m_perm; // 指明授予被授权者的权限信息，枚举值：READ，WRITE，FULL_CONTROL
};
```

查询存储桶 ACL

功能说明

查询存储桶的访问控制列表。

方法原型

```
CosResult GetBucketACL(const DGetBucketACLReq& req, GetBucketACLResp* resp)
```

请求示例

```
qcloud_cos::CosConfig config("./config.json");
qcloud_cos::CosAPI cos(config);

std::string bucket_name = "examplebucket-1250000000";

// 在配置文件的 DestDomain 中已经包含了 bucket name, 此处不再填写
qcloud_cos::GetBucketACLReq req("");
qcloud_cos::GetBucketACLResp resp;
qcloud_cos::CosResult result = cos.GetBucketACL(req, &resp);

// 调用成功，调用 resp 的成员函数获取返回内容
if (result.IsSucc()) {
// ...
} else {
// 获取 ACL 失败，可以调用 CosResult 的成员函数输出错误信息，比如 requestID 等
}
```

参数说明

参数	参数描述
req	GetBucketACLReq，GetBucketACL 操作的请求
resp	GetBucketACLResp，GetBucketACL 操作的返回

GetBucketACLResp 提供以下成员函数：

```
std::string GetOwnerID();
std::string GetOwnerDisplayName();
std::vector<Grant> GetAccessControlList();
```

对象操作

简介

本文档提供关于对象的简单操作、分块操作等其他操作相关的 API 概览以及 SDK 示例代码。

简单操作

API	操作名	操作描述
GET Bucket（List Object）	查询对象列表	查询存储桶下的部分或者全部对象
PUT Object	简单上传对象	上传一个对象至存储桶
HEAD Object	查询对象元数据	查询对象元数据信息
GET Object	下载对象	下载一个对象至本地
PUT Object - Copy	设置对象复制	复制文件到目标路径
DELETE Object	删除单个对象	在存储桶中删除指定对象
DELETE Multiple Objects	删除多个对象	在存储桶中批量删除对象

分块操作

API	操作名	操作描述
List Multipart Uploads	查询分块上传	查询正在进行中的分块上传信息
Initiate Multipart Upload	初始化分块上传	初始化分块上传任务
Upload Part	上传分块	分块上传文件
Upload Part - Copy	复制分块	将其他对象复制为一个分块
List Parts	查询已上传块	查询特定分块上传操作中的已上传的块
Complete Multipart Upload	完成分块上传	完成整个文件的分块上传
Abort Multipart Upload	终止分块上传	终止一个分块上传操作并删除已上传的块

其他操作

API	操作名	操作描述
PUT Object acl	设置对象 ACL	设置存储桶中某个对象的访问控制列表
GET Object acl	查询对象 ACL	查询对象的访问控制列表

简单操作

查询对象列表

功能说明

查询存储桶下的部分或者全部对象 。

方法原型

```
CosResult GetBucket(const GetBucketReq& req, GetBucketResp* resp)
```

请求示例

```
qcloud_cos::CosConfig config("./config.json");
qcloud_cos::CosAPI cos(config);

std::string bucket_name = "examplebucket-1250000000";

// 目的 Bucket 名称, 和配置文件中 DestDomain 的 bucket 相同
qcloud_cos::GetBucketReq req("");
qcloud_cos::GetBucketResp resp;
qcloud_cos::CosResult result = cos.GetBucket(req, &resp);

// 调用成功，调用 resp 的成员函数获取返回内容
if (result.IsSucc()) {
std::cout << "Name=" << resp.GetName() << std::endl;
std::cout << "Prefix=" << resp.GetPrefix() << std::endl;
std::cout << "Marker=" << resp.GetMarker() << std::endl;
std::cout << "MaxKeys=" << resp.GetMaxKeys() << std::endl;
} else {
std::cout << "ErrorInfo=" << result.GetErrorInfo() << std::endl;
std::cout << "HttpStatus=" << result.GetHttpStatus() << std::endl;
std::cout << "ErrorCode=" << result.GetErrorCode() << std::endl;
std::cout << "ErrorMsg=" << result.GetErrorMsg() << std::endl;
std::cout << "ResourceAddr=" << result.GetResourceAddr() << std::endl;
std::cout << "XCosRequestId=" << result.GetXCosRequestId() << std::endl;
std::cout << "XCosTraceId=" << result.GetXCosTraceId() << std::endl;
}
```

参数说明

参数	参数描述
req	GetBucketReq，GetBucket 操作的请求
resp	GetBucketResp，GetBucket 操作的返回

GetBucketResp 提供以下成员函数，用于获取 Get Bucket 返回的 XML 格式中的具体内容。

```
std::vector<Content> GetContents();
std::string GetName();
std::string GetPrefix();
std::string GetMarker();
uint64_t GetMaxKeys();
bool IsTruncated();
std::vector<std::string> GetCommonPrefixes();
```

其中 Content 的定义如下：

```
struct Content {
std::string m_key; // Object 的 Key
std::string m_last_modified; // Object 最后被修改时间
std::string m_etag; // 文件的 MD-5 算法校验值
std::string m_size; // 文件大小，单位是 Byte
std::vector<std::string> m_owner_ids; // Bucket 持有者信息
std::string m_storage_class; // Object 的存储类别
};
```

简单上传对象

功能说明

上传对象到指定的存储桶中。

方法原型

```
/// 通过 Stream 进行上传
CosResult PutObject(const PutObjectByStreamReq& req, PutObjectByStreamResp* resp)

/// 上传本地文件
CosResult PutObject(const PutObjectByFileReq& req, PutObjectByFileResp* resp)
```

请求示例

```
qcloud_cos::CosConfig config("./config.json");
qcloud_cos::CosAPI cos(config);

// 目的 Bucket 名称, 和配置文件中 DestDomain 的 bucket 相同
std::string bucket_name = "examplebucket-1250000000";
std::string object_name = "object_name";

// 简单上传(流)
{
std::istringstream iss("put object");
// request 的构造函数中需要传入 istream
qcloud_cos::PutObjectByStreamReq req(bucket_name, object_name, iss);
// 关闭MD5校验，开启使用req.TurnOnComputeConentMd5()，默认情况开启
req.TurnOffComputeConentMd5();
qcloud_cos::PutObjectByStreamResp resp;
qcloud_cos::CosResult result = cos.PutObject(req, &resp);

if (result.IsSucc()) {
// 调用成功，调用 resp 的成员函数获取返回内容
do sth
} else {
// 调用失败，调用 result 的成员函数获取错误信息
std::cout << "ErrorInfo=" << result.GetErrorInfo() << std::endl;
std::cout << "HttpStatus=" << result.GetHttpStatus() << std::endl;
std::cout << "ErrorCode=" << result.GetErrorCode() << std::endl;
```

```
std::cout << "ErrorMsg=" << result.GetErrorMsg() << std::endl;
std::cout << "ResourceAddr=" << result.GetResourceAddr() << std::endl;
std::cout << "XCosRequestId=" << result.GetXCosRequestId() << std::endl;
std::cout << "XCosTraceId=" << result.GetXCosTraceId() << std::endl;
}
}

// 简单上传(文件)
{
// request 的构造函数中需要传入本地文件路径
qcloud_cos::PutObjectByFileReq req(bucket_name, object_name, "/path/to/local/file");
// 关闭 MD5 校验，开启使用 req.TurnOnComputeConentMd5()，默认情况开启
req.TurnOffComputeConentMd5();
qcloud_cos::PutObjectByFileResp resp;
qcloud_cos::CosResult result = cos.PutObject(req, &resp);
if (result.IsSucc()) {
// 调用成功，调用 resp 的成员函数获取返回内容
do sth
} else {
// 调用失败，调用 result 的成员函数获取错误信息
std::cout << "ErrorInfo=" << result.GetErrorInfo() << std::endl;
std::cout << "HttpStatus=" << result.GetHttpStatus() << std::endl;
std::cout << "ErrorCode=" << result.GetErrorCode() << std::endl;
std::cout << "ErrorMsg=" << result.GetErrorMsg() << std::endl;
std::cout << "ResourceAddr=" << result.GetResourceAddr() << std::endl;
std::cout << "XCosRequestId=" << result.GetXCosRequestId() << std::endl;
std::cout << "XCosTraceId=" << result.GetXCosTraceId() << std::endl;
}
}
```

参数说明

参数	参数描述
req	PutObjectByStreamReq/PutObjectByFileReq，PutObject 操作的请求
resp	PutObjectByStreamResp/PutObjectByFileResp，PutObject 操作的返回

参数 Req 包括如下成员函数：

```
// Cache-Control RFC 2616 中定义的缓存策略，将作为 Object 元数据保存
void SetCacheControl(const std::string& str);

// Content-Disposition RFC 2616 中定义的文件名称，将作为 Object 元数据保存
void SetContentDisposition(const std::string& str);

// Content-Encoding RFC 2616 中定义的编码格式，将作为 Object 元数据保存-
void SetContentEncoding(const std::string& str);

// Content-Type RFC 2616 中定义的内容类型（MIME），将作为 Object 元数据保存
void SetContentType(const std::string& str);

// Expect 当使用 Expect: 100-continue 时，在收到服务端确认后，才会发送请求内容
void SetExpect(const std::string& str);

// Expires RFC 2616 中定义的过期时间，将作为 Object 元数据保存
void SetExpires(const std::string& str);

// 允许用户自定义的头部信息,将作为 Object 元数据返回.大小限制2K
void SetXCosMeta(const std::string& key, const std::string& value);

// 定义 Object 的 ACL 属性,有效值：private,public-read
// 默认值：private
void SetXcosAcl(const std::string& str);

// 赋予被授权者读的权限.格式：x-cos-grant-read: id=" ",id=" ".
// 当需要给子账户授权时,id="qcs::cam::uin/<OwnerUin>:uin/<SubUin>"
// 当需要给根账户授权时,id="qcs::cam::uin/<OwnerUin>:uin/<OwnerUin>"
void SetXcosGrantRead(const std::string& str);
```

```
// 赋予被授权者读写权限.格式：x-cos-grant-full-control: id=" ",id=" ".
// 当需要给子账户授权时,id="qcs::cam::uin/<OwnerUin>:uin/<SubUin>",
// 当需要给根账户授权时,id="qcs::cam::uin/<OwnerUin>:uin/<OwnerUin>"
void SetXcosGrantFullControl(const std::string& str);

/// 设置Server端加密使用的算法, 目前支持AES256
void SetXCosServerSideEncryption(const std::string& str);
```

参数 Resp 包括如下成员函数：

```
/// 获取Object的版本号, 如果Bucket未开启多版本, 返回空字符串
std::string GetVersionId();

/// Server端加密使用的算法
std::string GetXCosServerSideEncryption();
```

查询对象元数据

功能说明

查询对象元数据信息。

方法原型

```
CosResult HeadObject(const HeadObjectReq& req, HeadObjectResp* resp)
```

请求示例

```
key := "test/hello.txt"
resp, err := client.Object.Head(context.Background(), key, nil)
```

参数说明

```
qcloud_cos::CosConfig config("./config.json");
qcloud_cos::CosAPI cos(config);

// 目的 Bucket 名称, 和配置文件中 DestDomain 的 bucket 相同
std::string bucket_name = "examplebucket-1250000000";
std::string object_name = "object_name";
qcloud_cos::HeadObjectReq req(bucket_name, object_name);
qcloud_cos::HeadObjectResp resp;
qcloud_cos::CosResult result = cos.HeadObject(req, &resp);
if (result.IsSucc()) {
// 下载成功，可以调用 HeadObjectResp 的成员函数
} else {
// 下载失败，可以调用 CosResult 的成员函数输出错误信息，比如 requestID 等
}
```

参数说明

参数	参数描述
req	HeadObjectReq，HeadObject 操作的请求
resp	HeadObjectResp，HeadObject 操作的返回

HeadObjectResp 除了读取公共头部的成员函数外，还提供以下成员函数：

```
std::string GetXCosObjectType();

std::string GetXCosStorageClass();

// 获取自定义的 meta, 参数可以为 x-cos-meta-* 中的 *
std::string GetXCosMeta(const std::string& key);

// 以 map 形式返回所有自定义的 meta, map 的 key 均不包含"x-cos-meta-"前缀
std::map<std::string, std::string> GetXCosMetas();
```

```
// 获取 Server 端加密使用的算法
std::string GetXCosServerSideEncryption();
```

下载对象

功能说明

下载对象到本地（Get Object）。

方法原型

```
// 将 Object 下载到本地文件中
CosResult GetObject(const GetObjectByFileReq& req, GetObjectByFileResp* resp)

// 将 Object 下载到流中
CosResult GetObject(const GetObjectByStreamReq& req, GetObjectByStreamResp* resp)

// 将 Object 下载到本地文件中（多线程）
CosResult GetObject(const MultiGetObjectReq& req, MultiGetObjectResp* resp)
```

请求示例

```
qcloud_cos::CosConfig config("./config.json");
qcloud_cos::CosAPI cos(config);

// 目的 Bucket 名称, 和配置文件中 DestDomain 的 bucket 相同
std::string bucket_name = "examplebucket-1250000000";
std::string object_name = "object_name";
std::string local_path = "/tmp/object_name";

// 下载到本地文件
{
// request 需要提供 appid、bucketname、object,以及本地的路径（包含文件名）
qcloud_cos::GetObjectByFileReq req(bucket_name, object_name, local_path);
qcloud_cos::GetObjectByFileResp resp;
qcloud_cos::CosResult result = cos.GetObject(req, &resp);
if (result.IsSucc()) {
// 下载成功，可以调用 GetObjectByFileResp 的成员函数
} else {
// 可以调用 CosResult 的成员函数输出错误信息，比如 requestID 等
}
}

// 下载到流中
{
// request 需要提供 appid、bucketname、object, 以及输出流
std::ostringstream os;
qcloud_cos::GetObjectByStreamReq req(bucket_name, object_name, os);
qcloud_cos::GetObjectByStreamResp resp;
qcloud_cos::CosResult result = cos.GetObject(req, &resp);
if (result.IsSucc()) {
// 下载成功，可以调用 GetObjectByStreamResp 的成员函数
} else {
// 下载失败，可以调用 CosResult 的成员函数输出错误信息，比如 requestID 等
}
}

// 多线程下载文件到本地
{
// request需要提供 appid、bucketname、object以及本地的路径（包含文件名）
qcloud_cos::MultiGetObjectReq req(bucket_name, object_name, local_path);
qcloud_cos::MultiGetObjectResp resp;
qcloud_cos::CosResult result = cos.GetObject(req, &resp);
if (result.IsSucc()) {
// 下载成功，可以调用 MultiGetObjectResp 的成员函数
} else {
// 下载失败，可以调用 CosResult 的成员函数输出错误信息，比如 requestID 等
}
}
```

参数说明

参数	参数描述
req	GetObjectByFileReq/GetObjectByStreamReq/MultiGetObjectReq，GetObject 操作的请求
resp	GetObjectByFileResp/GetObjectByStreamResp/MultiGetObjectResp，GetObject 操作的返回

成员函数如下：

```
// 设置响应头部中的 Content-Type 参数
void SetResponseContentType(const std::string& str);

// 设置响应头部中的 Content-Language 参数
void SetResponseContentLang(const std::string& str);

// 设置响应头部中的 Content-Expires 参数
void SetResponseExpires(const std::string& str);

// 设置响应头部中的 Cache-Control 参数
void SetResponseCacheControl(const std::string& str);

// 设置响应头部中的 Content-Disposition 参数
void SetResponseContentDisposition(const std::string& str);

// 设置响应头部中的 Content-Encoding 参数
void SetResponseContentEncoding(const std::string& str);
```

GetObjectResp 除了读取公共头部的成员函数外，还提供以下成员函数：

```
// 获取 Object 最后被修改的时间, 字符串格式 Date, 类似"Wed, 28 Oct 2014 20:30:00 GMT"
std::string GetLastModified();

// 获取 Object type, 表示 Object 是否可以被追加上传，枚举值：normal 或者 appendable
std::string GetXCosObjectType();

// 以 map 形式返回所有自定义的 meta, map 的 key 均不包含"x-cos-meta-"前缀
std::map<std::string, std::string> GetXCosMetas();

// 获取自定义的 meta, 参数可以为 x-cos-meta-*中的*
std::string GetXCosMeta(const std::string& key);

// 获取Server端加密使用的算法
std::string GetXCosServerSideEncryption();
```

设置对象复制

复制文件到目标路径。

方法原型

```
CosResult PutObjectCopy(const PutObjectCopyReq& req, PutObjectCopyResp* resp)
```

请求示例

```
qcloud_cos::CosConfig config("./config.json");
qcloud_cos::CosAPI cos(config);

// 目的 Bucket 名称, 和配置文件中 DestDomain 的 bucket 相同
std::string bucket_name = "examplebucket-1250000000";
std::string object_name = "sevenyou";

qcloud_cos::PutObjectCopyReq req(bucket_name, object_name);
req.SetXCosCopySource("sevenyousouthtest-12345656.cn-south.myqcloud.com/sevenyou_source_obj");
qcloud_cos::PutObjectCopyResp resp;
qcloud_cos::CosResult result = cos.PutObjectCopy(req, &resp);
```

参数说明

参数	参数描述
req	PutObjectCopyReq，PutObjectCopy 操作的请求
resp	PutObjectCopyResp，PutObjectCopy 操作的返回

PutObjectCopyReq 包含以下成员函数：

```
// 源文件 URL 路径，可以通过 versionid 子资源指定历史版本
void SetXCosCopySource(const std::string& str);

// 是否拷贝元数据，枚举值：Copy, Replaced，默认值 Copy。
// 假如标记为 Copy，忽略 Header 中的用户元数据信息直接复制；
// 假如标记为 Replaced，按 Header 信息修改元数据。
// 当目标路径和原路径一致，即用户试图修改元数据时，必须为 Replaced
void SetXCosMetadataDirective(const std::string& str);

// 当 Object 在指定时间后被修改，则执行操作，否则返回 412。
// 可与 x-cos-copy-source-If-None-Match 一起使用，与其他条件联合使用返回冲突。
void SetXCosCopySourceIfModifiedSince(const std::string& str);

// 当 Object 在指定时间后未被修改，则执行操作，否则返回 412。
// 可与 x-cos-copy-source-If-Match 一起使用，与其他条件联合使用返回冲突。
void SetXCosCopySourceIfUnmodifiedSince(const std::string& str);

// 当 Object 的 Etag 和给定一致时，则执行操作，否则返回 412。
// 可与x-cos-copy-source-If-Unmodified-Since 一起使用，与其他条件联合使用返回冲突
void SetXCosCopySourceIfMatch(const std::string& str);

// 当 Object 的 Etag 和给定不一致时，则执行操作，否则返回 412。
// 可与 x-cos-copy-source-If-Modified-Since 一起使用，与其他条件联合使用返回冲突。
void SetXCosCopySourceIfNoneMatch(const std::string& str);

// 定义 Object 的 ACL 属性,有效值：private,public-read
// 默认值：private
void SetXCosAcl(const std::string& str);

// 赋予被授权者读的权限。格式：id="[OwnerUin]"
void SetXCosGrantRead(const std::string& str);

// 赋予被授权者所有的权限。格式：id="[OwnerUin]"
void SetXCosGrantFullControl(const std::string& str);

// 允许用户自定义的头部信息,将作为 Object 元数据返回.大小限制2K
void SetXCosMeta(const std::string& key, const std::string& value);

/// 设置 Server 端加密使用的算法, 目前支持 AES256
void SetXCosServerSideEncryption(const std::string& str);
```

PutObjectCopyResp 包含以下成员函数：

```
// 返回文件的 MD5 算法校验值。ETag 的值可以用于检查 Object 的内容是否发生变化。
std::string GetEtag();

// 返回文件最后修改时间，GMT 格式
std::string GetLastModified();

// 返回版本号
std::string GetVersionId();

/// Server端加密使用的算法
std::string GetXCosServerSideEncryption();
```

删除单个对象

功能说明

在存储桶中删除指定对象。

方法原型

```
CosResult DeleteObject(const DeleteObjectReq& req, DeleteObjectResp* resp)
```

请求示例

```
qcloud_cos::CosConfig config("./config.json");
qcloud_cos::CosAPI cos(config);

// 目的 Bucket 名称, 和配置文件中 DestDomain 的 bucket 相同
std::string bucket_name = "examplebucket-1250000000";
std::string object_name = "test_object";

qcloud_cos::DeleteObjectReq req(bucket_name, object_name);
qcloud_cos::DeleteObjectResp resp;
qcloud_cos::CosResult result = cos.DeleteObject(req, &resp);

// 调用成功，调用 resp 的成员函数获取返回内容
if (result.IsSucc()) {
// ...
} else {
// 可以调用 CosResult 的成员函数输出错误信息，比如 requestID 等
}
```

参数说明

参数	参数描述
req	DeleteObjectReq，DeleteObject 操作的请求
resp	DeletObjectResp，DeletObject 操作的返回

删除多个对象

功能说明

在存储桶中批量删除对象。

方法原型

```
CosResult DeleteObjects(const DeleteObjectsReq& req, DeleteObjectsResp* resp)
```

请求示例

```
qcloud_cos::CosConfig config("./config.json");
qcloud_cos::CosAPI cos(config);

// 目的 Bucket 名称, 和配置文件中 DestDomain 的 bucket 相同
std::string bucket_name = "examplebucket-1250000000";

std::vector<std::string> objects;
std::vector<ObjectVersionPair> to_be_deleted;
objects.push_back("batch_delete_test_00");
objects.push_back("batch_delete_test_01");
objects.push_back("batch_delete_test_02");
objects.push_back("batch_delete_test_03");
for (size_t idx = 0; idx < objects.size(); ++idx) {
ObjectVersionPair pair;
pair.m_object_name = objects[idx];
to_be_deleted.push_back(pair);
}
qcloud_cos::DeleteObjectsReq req(bucket_name, to_be_deleted); qcloud_cos::DeleteObjectsResp resp; qcloud_cos::CosResult result = cos.DeleteObjects(re
q, &resp);
// 调用成功，调用 resp 的成员函数获取返回内容
if (result.IsSucc()) {
// ...
} else {
// 可以调用 CosResult 的成员函数输出错误信息，比如 requestID 等
}
```

参数说明

参数	参数描述
req	DeleteObjectsReq，DeleteObjects 操作的请求
resp	DeleteObjectsResp，DeleteObjects 操作的返回

DeleteObjectsReq 包含以下成员函数：

```
// 添加对象，并指定版本
void AddObjectVersion(const std::string& object, const std::string& version)
// 添加对象，非多版本
void AddObject(const std::string& object)
```

DeleteObjectsResp 包含以下成员函数：

```
// 获取删除成功的 objects 信息
std::vector<DeletedInfo> GetDeletedInfos() const

// 获取删除失败的 objects 信息
std::vector<ErrorInfo> GetErrorinfos() const
```

对应DeletedInfo 和 ErrorInfo 的结构如下：

```
struct DeletedInfo{
std::string m_key; // object key
}
struct ErrorInfo{
std::string m_key; // object key
std::string m_code; // error code
std::string m_message; // error message
}
```

分块操作

查询分块上传

功能说明

查询指定存储桶中正在进行的分块上传（List Multipart Uploads）。

方法原型

```
CosResult CosAPI::ListMultipartUpload(const ListMultipartUploadReq& request, ListMultipartUploadResp* response)
```

请求示例

```
qcloud_cos::CosConfig config("./config.json");
qcloud_cos::CosAPI cos(config);

// 目的 Bucket 名称, 和配置文件中 DestDomain 的 bucket 相同
std::string bucket_name = "examplebucket-1250000000";
std::string object_name = "test_object";

qcloud_cos::ListMultipartUploadReq req(bucket_name, object_name);
qcloud_cos::ListMultipartUploadResp resp;
qcloud_cos::CosResult result = cos.ListMultipartUpload(req, &resp);

for (std::vector<qcloud_cos::Upload>::const_iterator itr = rst.begin(); itr != rst.end(); ++itr) {
const qcloud_cos::Upload& upload = *itr;
std::cout << "key = " << upload.m_key << ", uploadid= " << upload.m_uploadid << ", storagen class = " << upload.m_storage_class << ", m_initiated= "
<< upload.m_initiated << std::endl;
}
```

```
// 调用成功，调用 resp 的成员函数获取返回内容
if (result.IsSucc()) {
// ...
} else {
// 可以调用 CosResult 的成员函数输出错误信息，比如 requestID 等
}
```

参数说明

参数	参数描述
req	ListMultipartUploadReq，ListMultipartUpload 操作的请求
resp	ListMultipartUploadResp，ListMultipartUpload 操作的返回

ListMultipartUploadReq 成员函数：

```
// 限定返回的 Object key 必须以 Prefix 作为前缀。注意使用 prefix 查询时，返回的 key 中仍会包含 Prefix。
void SetPrefix(const std::string& prefix);

// 定界符为一个符号，对 Object 名字包含指定前缀且第一次出现 delimiter 字符之间的 Object 作为一组元素：common prefix。如果没有 prefix，则从路径起点开始
void SetDelimiter(const std::string& delimiter);

// 规定返回值的编码格式，合法值：url
void SetEncodingType(const std::string& encoding_type);

// 与 upload-id-marker 一起使用当 upload-id-marker 未被指定时，ObjectName 字母顺序大于 key-marker 的条目将被列出，当 upload-id-marker 被指定时，Ob
jectName 字母顺序大于 key-marker 的条目被列出，ObjectName 字母顺序等于 key-marker 同时 UploadID 大于 upload-id-marker 的条目将被列出。
void SetKeyMarker(const std::string& marker);

// 设置最大返回的 multipart 数量，合法取值从1到1000，默认1000
void SetMaxUploads(const std::string& max_uploads);

// 与 key-marker 一起使用，当 key-marker 未被指定时，upload-id-marker 将被忽略，当 key-marker 被指定时，ObjectName字母顺序大于 key-marker 的条目被
列出，ObjectName 字母顺序等于 key-marker 同时 UploadID 大于 upload-id-marker 的条目将被列出。
void SetUploadIdMarker(const std::string& upload_id_marker);
```

ListMultipartUploadResp 成员函数：

```
// 获取Bucket中Object对应的元信息
std::vector<Upload> GetUpload()；
// Bucket 名称
std::string GetName()；
// 编码格式
std::string GetEncodingType() const；
// 默认以UTF-8二进制顺序列出条目，所有列出条目从marker开始
std::string GetMarker() const；
// 列出条目从该 UploadId 值开始
std::string GetUploadIdMarker() const；
// 假如返回条目被截断，则返回 NextKeyMarker 就是下一个条目的起点
std::string GetNextKeyMarker() const；
// 假如返回条目被截断，则返回 UploadId 就是下一个条目的起点
std::string GetNextUploadIdMarker() const；
// 最大返回的 multipart 数量，合法取值从0到1000
std::string GetMaxUploads () const；
// 响应请求条目是否被截断，布尔值：true，false
bool IsTruncated()；
// 返回的文件前缀
std::string GetPrefix() const；
// 获取定界符
std::string GetDelimiter() const；
// 将 Prefix 到 delimiter 之间的相同路径归为一类，定义为 Common Prefix
std::vector<std::string> GetCommonPrefixes() const
```

分块上传对象

分块上传对象可包括的操作：

- 分块上传对象： 初始化分块上传， 上传分块， 完成所有分块上传。

- 删除已上传分块。

初始化分块上传

功能说明

初始化分块上传，获取对应的 uploadId（Initiate Multipart Upload）。

方法原型

```
CosResult InitMultiUpload(const InitMultiUploadReq& req, InitMultiUploadResp* resp)
```

请求示例

```
qcloud_cos::CosConfig config("./config.json");
qcloud_cos::CosAPI cos(config);
// 目的 Bucket 名称, 和配置文件中 DestDomain 的 bucket 相同
std::string bucket_name = "examplebucket-1250000000";
std::string object_name = "object_name";

qcloud_cos::InitMultiUploadReq req(bucket_name, object_name);
qcloud_cos::InitMultiUploadResp resp;
qcloud_cos::CosResult result = cos.InitMultiUpload(req, &resp);

std::string upload_id = "";
if (result.IsSucc()) {
upload_id = resp.GetUploadId();
}
```

参数说明

参数	参数描述
req	InitMultiUploadReq，InitMultiUpload 操作的请求
resp	InitMultiUploadResp，InitMultiUpload 操作的返回

InitMultiUploadReq 的成员函数如下：

```
// Cache-Control RFC 2616 中定义的缓存策略，将作为 Object 元数据保存
void SetCacheControl(const std::string& str);

// Content-Disposition RFC 2616 中定义的文件名称，将作为 Object 元数据保存
void SetContentDisposition(const std::string& str);

// Content-Encoding RFC 2616 中定义的编码格式，将作为 Object 元数据保存-
void SetContentEncoding(const std::string& str);

// Content-Type RFC 2616 中定义的内容类型（MIME），将作为 Object 元数据保存
void SetContentType(const std::string& str);

// Expires RFC 2616 中定义的过期时间，将作为 Object 元数据保存
void SetExpires(const std::string& str);

// 允许用户自定义的头部信息,将作为 Object 元数据返回.大小限制2K
void SetXCosMeta(const std::string& key, const std::string& value);

// 定义 Object 的 ACL 属性,有效值：private,public-read
// 默认值：private
void SetXcosAcl(const std::string& str);

// 赋予被授权者读的权限.格式：x-cos-grant-read: id=" ",id=" ".
// 当需要给子账户授权时,id="qcs::cam::uin/<OwnerUin>:uin/<SubUin>"
// 当需要给根账户授权时,id="qcs::cam::uin/<OwnerUin>:uin/<OwnerUin>"
void SetXcosGrantRead(const std::string& str);

// 赋予被授权者读写权限.格式：x-cos-grant-full-control: id=" ",id=" ".
// 当需要给子账户授权时,id="qcs::cam::uin/<OwnerUin>:uin/<SubUin>",
// 当需要给根账户授权时,id="qcs::cam::uin/<OwnerUin>:uin/<OwnerUin>"
```

```
void SetXcosGrantFullControl(const std::string& str);

/// 设置 Server 端加密使用的算法, 目前支持 AES256
void SetXCosServerSideEncryption(const std::string& str);
```

当成功执行此请求后，返回的 response 中会包含 bucket、key、uploadId， 分别表示分块上传的目标 Bucket、Object 名称以及后续分块上传所需的编号。

InitMultiUploadResp 的成员函数如下:

```
std::string GetBucket();
std::string GetKey();
std::string GetUploadId();

// Server端加密使用的算法
std::string GetXCosServerSideEncryption();
```

上传分块

上传分块（Upload Part）。

方法原型

```
CosResult UploadPartData(const UploadPartDataReq& request, UploadPartDataResp* response)
```

请求示例

```
qcloud_cos::CosConfig config("./config.json");
qcloud_cos::CosAPI cos(config);

// 目的 Bucket 名称, 和配置文件中 DestDomain 的 bucket 相同
std::string bucket_name = "examplebucket-1250000000";
std::string object_name = "test_object";

// 上传第一个分块
{
std::fstream is("demo_5M.part1");
qcloud_cos::UploadPartDataReq req(bucket_name, object_name, upload_id, is);
req.SetPartNumber(1);
// 关闭 MD5 校验，开启使用 req.TurnOnComputeConentMd5()，默认情况开启
req.TurnOffComputeConentMd5();
qcloud_cos::UploadPartDataResp resp;
qcloud_cos::CosResult result = cos.UploadPartData(req, &resp);

// 上传成功需要记录分块编号以及返回的 ETag
if (result.IsSucc()) {
etags.push_back(resp.GetEtag());
part_numbers.push_back(1);
}
is.close();
}

// 上传第二个分块
{
std::fstream is("demo_5M.part2");
qcloud_cos::UploadPartDataReq req(bucket_name, object_name,
upload_id, is);
req.SetPartNumber(2);
qcloud_cos::UploadPartDataResp resp;
qcloud_cos::CosResult result = cos.UploadPartData(req, &resp);

// 上传成功需要记录分块编号以及返回的 ETag
if (result.IsSucc()) {
etags.push_back(resp.GetEtag());
part_numbers.push_back(2);
}
is.close();
}
```

参数说明

参数	参数描述
req	UploadPartDataReq，UploadPartData 操作的请求
resp	UploadPartDataResp，UploadPartData 操作的返回

UploadPartDataReq 在构造时，需要指明请求的 APPID、Bucket、Object、初始化成功后获取的 UploadId，以及上传的数据流（调用完成后，流由调用方自己负责关闭）。

```
UploadPartDataReq(const std::string& bucket_name,
const std::string& object_name, const std::string& upload_id,
std::istream& in_stream);
```

此外，请求还需要设置分块编号, 这个分块在完成分块上传时也会用到。

```
void SetPartNumber(uint64_t part_number);
```

UploadPartDataResp 的成员函数如下：

```
/// Server 端加密使用的算法
std::string GetXCosServerSideEncryption();
```

复制分块

将其他对象复制为一个分块。

方法原型

```
CosResult UploadPartCopyData(const UploadPartCopyDataReq& request,UploadPartCopyDataResp* response)
```

请求示例

```
qcloud_cos::CosConfig config("./config.json");
qcloud_cos::CosAPI cos(config);

// 目的 Bucket 名称, 和配置文件中 DestDomain 的 bucket 相同
std::string bucket_name = "examplebucket-1250000000";
std::string object_name = "test_object";

std::string upload_id;
std::vector<uint64_t> numbers;
std::vector<std::string> etags;
std::string etag1 = "", etag2 = "";
InitMultiUpload(cos, bucket_name, object_name, &upload_id);

// First part
qcloud_cos::UploadPartCopyDataReq req(bucket_name, object_name, upload_id, 1);
req.SetXCosCopySource("sevenyousouth-1251668577.cos.ap-guangzhou.myqcloud.com/seven_10G.tmp");
req.SetXCosCopySourceRange("bytes=0-1048576000"); qcloud_cos::UploadPartCopyDataResp resp; qcloud_cos::CosResult result = cos.UploadPartCopyDat
a(req, &resp);
if ( result.IsSucc()) {
etag1 = resp.GetEtag();
}
numbers.push_back(1);
etags.push_back(etag1);

// Second part
qcloud_cos::UploadPartCopyDataReq req2(bucket_name, object_name, upload_id, 2); req2.SetXCosCopySource("sevenyoutest-7319456.cos.cn-north.myqcl
oud.com/sevenyou_2G_part");
req2.SetXCosCopySourceRange("bytes=1048576000-2097152000");
qcloud_cos::UploadPartCopyDataResp resp2;
qcloud_cos::CosResult result = cos.UploadPartCopyData(req2, &resp2);
if ( result.IsSucc()) {
etag2 = resp2.GetEtag();
}
numbers.push_back(2) ;
etags.push_back(etag2);
```

```
CompleteMultiUpload(cos, bucket_name, object_name, upload_id, etags, numbers);
```

参数说明

参数	参数描述
req	UploadPartCopyDataReq，UploadPartCopyData 操作的请求
resp	UploadPartCopyDataResp，UploadPartCopyData 操作的返回

```
/// 设置本次分块复制的 ID
void SetUploadId(const std::string& upload_id)
/// 设置本次分块复制的编号
void SetPartNumber(uint64_t part_number)
/// 设置本次分块复制的源文件 URL 路径，可以通过 versionid 子资源指定历史版本
void SetXCosCopySource(const std::string& src)
/// 设置源文件的字节范围，范围值必须使用 bytes=first-last 格式。
void SetXCosCopySourceRange(const std::string& range)
/// 当 Object 在指定时间后被修改，则执行操作，否则返回 412
void SetXCosCopySourceIfModifiedSince(const std::string& date)
/// 当 Object 在指定时间后未被修改，则执行操作，否则返回 412
void SetXCosCopySourceIfUnmodifiedSince(const std::string& date)
/// 当 Object 的 Etag 和给定一致时，则执行操作，否则返回 412
void SetXCosCopySourceIfMatch(const std::string& etag)
/// 当 Object 的 Etag 和给定不一致时，则执行操作，否则返回 412
void SetXCosCopySourceIfNoneMatch(const std::string& etag)


/// 获取返回文件的MD5算法校验值。
std::string GetEtag() const
/// 返回文件最后修改时间，GMT 格式
std::string GetLastModified() const
/// Server端加密使用的算法
std::string GetXCosServerSideEncryption() const
```

查询已上传块

功能说明

查询特定分块上传操作中的已上传的块。

方法原型

```
CosResult ListParts(const ListPartsReq& req, ListPartsResp* resp)
```

请求示例

```
qcloud_cos::CosConfig config("./config.json");
qcloud_cos::CosAPI cos(config);

// 目的 Bucket 名称, 和配置文件中 DestDomain 的 bucket 相同
std::string bucket_name = "examplebucket-1250000000";
std::string object_name = "test_object";

// uploadId 是调用 InitMultiUpload 后获取的
qcloud_cos::ListPartsReq req(bucket_name, object_name, upload_id);
req.SetMaxParts(1);
req.SetPartNumberMarker("1");
qcloud_cos::ListPartsResp resp;
qcloud_cos::CosResult result = cos.ListParts(req, &resp);

// 调用成功，调用 resp 的成员函数获取返回内容
if (result.IsSucc()) {
// ...
} else {
// 可以调用 CosResult 的成员函数输出错误信息，比如 requestID 等
}
```

参数说明

参数	参数描述
req	ListPartsReq，ListParts 操作的请求
resp	ListPartsResp，ListParts 操作的返回

ListPartsReq 包含以下成员函数：

```
// 构造函数，Bucket 名、Object 名、分块上传的 ID
ListPartsReq(const std::string& bucket_name,
const std::string& object_name,
const std::string& upload_id);

// \brief 规定返回值的编码方式
void SetEncodingType(const std::string& encoding_type);

// \brief 单次返回最大的条目数量，若不设置，默认1000
void SetMaxParts(uint64_t max_parts);

// \brief 默认以 UTF-8 二进制顺序列出条目，所有列出条目从 marker 开始
void SetPartNumberMarker(const std::string& part_number_marker);
```

ListPartsResp 包含以下成员函数：

```
// 分块上传的目标 Bucket
std::string GetBucket();

// 规定返回值的编码方式
std::string GetEncodingType();

// Object 的名称
std::string GetKey();

// 标识本次分块上传的 ID
std::string GetUploadId();

// 用来表示本次上传发起者的信息
Initiator GetInitiator();

// 用来表示这些分块所有者的信息
Owner GetOwner();

// 默认以 UTF-8 二进制顺序列出条目，所有列出条目从 marker 开始
uint64_t GetPartNumberMarker();

// 返回每一个块的信息
std::vector<Part> GetParts();

// 假如返回条目被截断，则返回 NextMarker 就是下一个条目的起点
uint64_t GetNextPartNumberMarker();

// 单次返回最大的条目数量
uint64_t GetMaxParts();

// 返回条目是否被截断，布尔值：TRUE，FALSE
bool IsTruncated();
```

其中 Part、Owner、Initiator 的定义如下：

```
struct Initiator {
std::string m_id; // 创建者的一个唯一标识
std::string m_display_name; // 创建者的用户名描述
};

struct Owner {
std::string m_id; // 用户的一个唯一标识
std::string m_display_name; // 用户名描述
};
```

```
struct Part {
uint64_t m_part_num; // 块的编号
uint64_t m_size; // 块大小，单位 Byte
std::string m_etag; // Object 块的 MD5 算法校验值
std::string m_last_modified; // 块最后修改时间
};
```

完成分块上传

功能说明

完成整个文件的分块上传。

方法原型

```
CosResult CompleteMultiUpload(const CompleteMultiUploadReq& request, CompleteMultiUploadResp* response)
```

请求示例

```
qcloud_cos::CompleteMultiUploadReq req(bucket_name, object_name, upload_id);
qcloud_cos::CompleteMultiUploadResp resp;
req.SetEtags(etags);
req.SetPartNumbers(part_numbers);

qcloud_cos::CosResult result = cos.CompleteMultiUpload(req, &resp);
```

参数说明

参数	参数描述
req	CompleteMultiUploadReq，CompleteMultiUpload 操作的请求
resp	CompleteMultiUploadResp，CompleteMultiUpload 操作的返回

CompleteMultiUploadReq 在构造时，需要指明请求的 APPID、Bucket、Object、初始化成功后获取的 UploadId。

```
CompleteMultiUploadReq(const std::string& bucket_name,
const std::string& object_name, const std::string& upload_id)
```

此外，request 还需要设置所有上传的分块编号和 ETag。

```
// 调用下列方法时，应注意编号和 ETag 的顺序必须一一对应
void SetPartNumbers(const std::vector<uint64_t>& part_numbers);
void SetEtags(const std::vector<std::string>& etags) ;

// 添加 part_number 和 ETag 对
void AddPartEtagPair(uint64_t part_number, const std::string& etag);

/// 设置 Server 端加密使用的算法, 目前支持 AES256
void SetXCosServerSideEncryption(const std::string& str);
```

CompleteMultiUploadResp 的返回内容中包括 Location、Bucket、Key、ETag，分别表示创建的 Object 的外网访问域名、分块上传的目标 Bucket、Object 的名称、合并后文件的 MD5 算法校验值。可以调用下列成员函数对 response 中的内容进行访问。

```
std::string GetLocation();
std::string GetKey();
std::string GetBucket();
std::string GetEtag();

// Server端加密使用的算法
std::string GetXCosServerSideEncryption();
```

终止分块上传

功能说明

终止一个分块上传操作并删除已上传的块。

方法原型

```
CosResult AbortMultiUpload(const AbortMultiUploadReq& request, AbortMultiUploadResp* response)
```

请求示例

```
qcloud_cos::AbortMultiUploadReq req(bucket_name, object_name, upload_id);
qcloud_cos::AbortMultiUploadResp resp;
qcloud_cos::CosResult result = cos.AbortMultiUpload(req, &resp);
```

参数说明

参数	参数描述
req	AbortMultiUploadReq，AbortMultiUpload 操作的请求
resp	AbortMultiUploadResp，AbortMultiUpload 操作的返回

AbortMultiUploadReq 需要在构造的时候指明 Bucket、Object 以及 Upload_id。

```
AbortMultiUploadReq(const std::string& bucket_name,
const std::string& object_name, const std::string& upload_id);
```

无特殊方法，可调用 BaseResp 的成员函数来获取公共头部内容。

其他操作

设置对象 ACL

功能说明

设置对象的访问控制列表。

方法原型

```
CosResult PutObjectACL(const PutObjectACLReq& req, PutObjectACLResp* resp)
```

请求示例

```
qcloud_cos::CosConfig config("./config.json");
qcloud_cos::CosAPI cos(config);

// 目的 Bucket 名称, 和配置文件中 DestDomain 的 bucket 相同
std::string bucket_name = "examplebucket-1250000000";
std::string object_name = "sevenyou";

// 1 设置 ACL 配置（通过 Body, 设置 ACL 可以通过 Body、Header 两种方式，但只能二选一，否则会有冲突）
{
qcloud_cos::PutObjectACLReq req(bucket_name, object_name);
qcloud_cos::Owner owner = {"qcs::cam::uin/xxxxx:uin/xxx", "qcs::cam::uin/xxxxxx:uin/xxxxx" };
qcloud_cos::Grant grant;
req.SetOwner(owner);
grant.m_grantee.m_type = "Group";
grant.m_grantee.m_uri = "http://imgcache.finance.cloud.tencent.com:80cam.qcloud.com/groups/global/AllUsers";
grant.m_perm = "READ";
req.AddAccessControlList(grant);

qcloud_cos::PutObjectACLResp resp;
qcloud_cos::CosResult result = cos.PutObjectACL(req, &resp);
// 调用成功，调用 resp 的成员函数获取返回内容
if (result.IsSucc()) {
// ...
} else {
// 设置 ACL，可以调用 CosResult 的成员函数输出错误信息，比如 requestID 等
```

```
}
}

// 2 设置 ACL 配置（通过 Header, 设置 ACL 可以通过 Body、Header 两种方式，但只能二选一，否则会有冲突）
{
qcloud_cos::PutObjectACLReq req(bucket_name, object_name);
req.SetXCosAcl("public-read-write");

qcloud_cos::PutObjectACLResp resp;
qcloud_cos::CosResult result = cos.PutObjectACL(req, &resp);
// 调用成功，调用 resp 的成员函数获取返回内容
if (result.IsSucc()) {
// ...
} else {
// 可以调用 CosResult 的成员函数输出错误信息，比如 requestID 等
}
}
```

参数说明

参数	参数描述
req	PutObjectACLReq，PutObjectACL 操作的请求
resp	PutObjectACLResp，PutObjectACL 操作的返回

PutObjectACLReq 包含以下成员函数：

```
// 定义 Object 的 ACL 属性,有效值：private,public-read
// 默认值：private
void SetXCosAcl(const std::string& str);

// 赋予被授权者读的权限。格式：id="[OwnerUin]"
void SetXCosGrantRead(const std::string& str);

// 赋予被授权者所有的权限。格式：id="[OwnerUin]"
void SetXCosGrantFullControl(const std::string& str);

// Object 持有者 ID
void SetOwner(const Owner& owner);

// 设置被授权者信息与权限信息
void SetAccessControlList(const std::vector<Grant>& grants);

// 添加单个 Object 的授权信息
void AddAccessControlList(const Grant& grant);
```

> SetXCosAcl/SetXCosGrantRead/SetXCosGrantWrite/SetXCosGrantFullControl 这类接口与 SetAccessControlList/AddAccessControlList 不可同时使用。因为前者实际是通过设置 HTTP Header 实现，而后者是在Body 中添加了 XML 格式的内容，二者只能二选一。SDK 内部优先使用第一类。

ACLRule 定义如下：

```
struct Grantee {
// type 类型可以为 RootAccount， SubAccount
// 当 type 类型为 RootAccount 时，可以在 id 中 uin 填写帐号 ID，也可以用 anyone（指代所有类型用户）代替 uin/<OwnerUin> 和 uin/<SubUin>
// 当 type 类型为 RootAccount 时，uin 代表根账户账号，Subaccount 代表子账户账号
std::string m_type;
std::string m_id; // qcs::cam::uin/<OwnerUin>:uin/<SubUin>
std::string m_display_name; // 非必选
std::string m_uri;
};

struct Grant {
Grantee m_grantee; // 被授权者资源信息
std::string m_perm; // 指明授予被授权者的权限信息，枚举值：READ，FULL_CONTROL
};
```

查询对象 ACL

功能说明

查询对象的访问控制列表。

方法原型

```
CosResult GetObjectACL(const DGetObjectACLReq& req, GetObjectACLResp* resp)
```

请求示例

```
qcloud_cos::CosConfig config("./config.json");
qcloud_cos::CosAPI cos(config);

// 目的 Bucket 名称, 和配置文件中 DestDomain 的 bucket 相同
std::string bucket_name = "examplebucket-1250000000";
std::string object_name = "exampleobject";

// GetObjectACLReq 的构造函数需要传入 Object_name
qcloud_cos::GetObjectACLReq req(bucket_name, object_name);
qcloud_cos::GetObjectACLResp resp;
qcloud_cos::CosResult result = cos.GetObjectACL(req, &resp);

// 调用成功，调用 resp 的成员函数获取返回内容
if (result.IsSucc()) {
// ...
} else {
// 可以调用 CosResult 的成员函数输出错误信息，比如 requestID 等
}
```

参数说明

参数	参数描述
req	GetObjectACLReq，GetObjectACL 操作的请求
resp	GetObjectACLResp，GetObjectACL 操作的返回

GetObjectACLResp 包含以下成员函数：

```
std::string GetOwnerID();
std::string GetOwnerDisplayName();
std::vector<Grant> GetAccessControlList();
```

高级接口（推荐）

复合上传

功能说明

封装分块上传各接口，并发上传。

方法原型

```
CosResult MultiUploadObject(const MultiUploadObjectReq& request, MultiUploadObjectResp* response)
```

请求示例

```
qcloud_cos::CosConfig config("./config.json");
qcloud_cos::CosAPI cos(config);

// 目的 Bucket 名称, 和配置文件中 DestDomain 的 bucket 相同
std::string bucket_name = "examplebucket-1250000000";
std::string object_name = "exampleobject";
std::string local_file = "./test"

qcloud_cos::MultiUploadObjectReq req(bucket_name, object_name, local_file);
```

```
// Complete 接口内部 chunk 保活，建议设置较长时间的 timeout。
req.SetRecvTimeoutInms(1000 * 60);
qcloud_cos::MultiUploadObjectResp resp;
qcloud_cos::CosResult result = cos.MultiUploadObject(req, &resp);

// 调用成功，调用 resp 的成员函数获取返回内容
if (result.IsSucc()) {
// ...
} else {
// 可以调用 CosResult 的成员函数输出错误信息，比如 requestID 等
}
```

参数说明

参数	参数描述
req	MultiUploadObjectReq，MultiUploadObject 操作的请求
resp	MultiUploadObjectResp，MultiUploadObject 操作的返回

MultiUploadObjectReq 包含以下成员函数：

```
// 设置分块大小，若小于1M，则按1M计算；若大于5G，则按5G计算
void SetPartSize(uint64_t bytes)
// 允许用户自定义的头部信息，将作为 Object 元数据返回，大小限制2K
void SetXCosMeta(const std::string& key, const std::string& value)
// 设置 Server 端加密使用的算法，目前支持 AES256
void SetXCosServerSideEncryption(const std::string& str)
// 设置内部线程池大小
void SetThreadPoolSize(int size)
```

MultiUploadObjectResp 包含以下成员函数：

```
std::string GetRespTag()
/// Server 端加密使用的算法
std::string GetXCosServerSideEncryption() const
```

复合下载

功能说明

并发 Range 下载。

方法原型

```
CosResult GetObject(const MultiGetObjectReq& request,
MultiGetObjectResp* response)
```

请求示例

```
qcloud_cos::CosConfig config("./config.json");
qcloud_cos::CosAPI cos(config);

// 目的 Bucket 名称, 和配置文件中 DestDomain 的 bucket 相同
std::string bucket_name = "examplebucket-1250000000";
std::string object_name = "exampleobject";
std::string file_path = "./test";

qcloud_cos::MultiGetObjectReq req(bucket_name, object_name, file_path); qcloud_cos::MultiGetObjectResp resp; qcloud_cos::CosResult result = cos.GetObj
ect(req, &resp);

// 调用成功，调用 resp 的成员函数获取返回内容
if (result.IsSucc()) {
// ...
} else {
// 可以调用 CosResult 的成员函数输出错误信息，比如 requestID 等
}
```

参数说明

参数	参数描述
req	MultiGetObjectReq，GetObject 操作的请求
resp	MultiGetObjectResp，GetObject 操作的返回

MultiGetObjectReq 包含以下成员函数：

```
// 设置分块大小
void SetSliceSize(uint64_t bytes)
// 设置线程池大小
void SetThreadPoolSize(int size)
```

MultiGetObjectResp 包含以下成员函数：

```
/// Server 端加密使用的算法
std::string GetXCosServerSideEncryption() const
```

存储桶管理

简介

本文档提供关于跨域访问、生命周期、版本控制、跨地域复制相关的 API 概览以及 SDK 示例代码。

跨域访问

API	操作名	操作描述
PUT Bucket cors	设置跨域配置	设置存储桶的跨域访问权限
GET Bucket cors	查询跨域配置	查询存储桶的跨域访问配置信息
DELETE Bucket cors	删除跨域配置	删除存储桶的跨域访问配置信息

版本控制

API	操作名	操作描述
PUT Bucket versioning	设置版本控制	设置存储桶的版本控制功能
GET Bucket versioning	查询版本控制	查询存储桶的版本控制信息

跨地域复制

API	操作名	操作描述
PUT Bucket replication	设置跨地域复制	设置存储桶的跨地域复制规则
GET Bucket replication	查询跨地域复制	查询存储桶的跨地域复制规则
DELETE Bucket replication	删除跨地域复制	删除存储桶的跨地域复制规则

跨域访问

设置跨域配置

功能说明

设置存储桶的跨域访问权限。

方法原型

```
CosResult PutBucketCORS(const PutBucketCORSReq& req, PutBucketCORSResp* resp)
```

请求示例

```
qcloud_cos::CosConfig config("./config.json");
qcloud_cos::CosAPI cos(config);

// 目的 Bucket 名称, 和配置文件中 DestDomain 的 bucket 相同
std::string bucket_name = "examplebucket-1250000000";

// PutBucketCORSReq 的构造函数需要传入 bucket_name
qcloud_cos::PutBucketCORSReq req(bucket_name);
qcloud_cos::CORSRule rule;
rule.m_id = "123";
rule.m_allowed_headers.push_back("x-cos-meta-test");
rule.m_allowed_origins.push_back("http://imgcache.finance.cloud.tencent.com:80www.qq.com");
rule.m_allowed_origins.push_back("http://imgcache.finance.cloud.tencent.com:80cloud.tentent.com");
rule.m_allowed_methods.push_back("PUT");
rule.m_allowed_methods.push_back("GET");
rule.m_max_age_secs = "600";
rule.m_expose_headers.push_back("x-cos-expose");
req.AddRule(rule);

qcloud_cos::PutBucketCORSResp resp;
qcloud_cos::CosResult result = cos.PutBucketCORS(req, &resp);

// 调用成功，调用 resp 的成员函数获取返回内容
if (result.IsSucc()) {
// ...
} else {
// 可以调用 CosResult 的成员函数输出错误信息，如 requestID 等
}
```

参数说明

参数	参数描述
req	PutBucketCORSReq，PutBucketCORS 操作的请求
resp	PutBucketCORSResp，PutBucketCORS 操作的返回

PutBucketCORSReq 提供以下成员函数：

```
// 新增 CORSRule
void AddRule(const CORSRule& rule);

// 设置 CORSRule
void SetRules(const std::vector<CORSRule>& rules)
```

CORSRule 定义如下：

```
struct CORSRule {
std::string m_id; // 配置规则的 ID，可选填
std::string m_max_age_secs; // 设置 OPTIONS 请求得到结果的有效期
std::vector<std::string> m_allowed_headers; // 在发送 OPTIONS 请求时告知服务端，接下来的请求可以使用哪些自定义的 HTTP 请求头部，支持通配符 *
std::vector<std::string> m_allowed_methods; // 允许的 HTTP 操作，枚举值：GET，PUT，HEAD，POST，DELETE
std::vector<std::string> m_allowed_origins; // 允许的访问来源，支持通配符 * ，格式为：协议://域名[:端口]如：http://imgcache.finance.cloud.tencent.com:80www.qq.com
std::vector<std::string> m_expose_headers; // 设置浏览器可以接收到的来自服务器端的自定义头部信息
};
```

查询跨域配置

功能说明

查询存储桶的跨域访问配置信息。

方法原型

```
CosResult GetBucketCORS(const GetBucketCORSReq& req, GetBucketCORSResp* resp)
```

请求示例

```
qcloud_cos::CosConfig config("./config.json");
qcloud_cos::CosAPI cos(config);

// 目的 Bucket 名称, 和配置文件中 DestDomain 的 bucket 相同
std::string bucket_name = "examplebucket-1250000000";

// GetBucketCORSReq 的构造函数需要传入 bucket_name
qcloud_cos::GetBucketCORSReq req(bucket_name);
qcloud_cos::GetBucketCORSResp resp;
qcloud_cos::CosResult result = cos.GetBucketCORS(req, &resp);

// 调用成功，调用 resp 的成员函数获取返回内容
if (result.IsSucc()) {
// ...
} else {
// 可以调用 CosResult 的成员函数输出错误信息，如 requestID 等
}
```

参数说明

参数	参数描述
req	GetBucketCORSReq，GetBucketCORS 操作的请求
resp	GetBucketCORSResp，GetBucketCORS 操作的返回

GetBucketCORSResp 提供以下成员函数：

```
// 获取 CORSRules, CORSRule 定义参见 Put Bucket CORS
std::vector<CORSRule> GetCORSRules();
```

删除跨域配置

功能说明

删除指定存储桶的跨域访问配置。

方法原型

```
CosResult DeleteBucketCORS(const DeleteBucketCORSReq& req, DeleteBucketCORSResp* resp)
```

请求示例

```
qcloud_cos::CosConfig config("./config.json");
qcloud_cos::CosAPI cos(config);

// 目的 Bucket 名称, 和配置文件中 DestDomain 的 bucket 相同
std::string bucket_name = "examplebucket-1250000000";

// DeleteBucketCORSReq 的构造函数需要传入 bucket_name
qcloud_cos::DeleteBucketCORSReq req(bucket_name);
qcloud_cos::DeleteBucketCORSResp resp;
qcloud_cos::CosResult result = cos.DeleteBucketCORS(req, &resp);

// 调用成功，调用 resp 的成员函数获取返回内容
if (result.IsSucc()) {
// ...
} else {
// 可以调用 CosResult 的成员函数输出错误信息，如 requestID 等
}
```

参数说明

参数	参数描述
req	DeleteBucketCORSReq，DeleteBucketCORS 操作的请求
resp	DeleteBucketCORSResp，DeleteBucketCORS 操作的返回

版本控制

设置版本控制

功能说明

设置指定存储桶的版本控制功能。

方法原型

```
CosResult PutBucketVersioning(const PutBucketVersioningReq& request, PutBucketVersioningResp* response)
```

请求示例

```
qcloud_cos::CosConfig config("./config.json");
qcloud_cos::CosAPI cos(config);

// 目的 Bucket 名称, 和配置文件中 DestDomain 的 bucket 相同
std::string bucket_name = "examplebucket-1250000000";

// PutBucketVersioningReq 的构造函数需要传入 bucket_name
qcloud_cos::PutBucketVersioningReq req(bucket_name);
req.SetStatus(true);
qcloud_cos::PutBucketVersioningResp resp;
qcloud_cos::CosResult result = cos.PutBucketVersioning(req, &resp);

// 调用成功，调用 resp 的成员函数获取返回内容
if (result.IsSucc()) {
// ...
} else {
// 可以调用 CosResult 的成员函数输出错误信息，如 requestID 等
}
```

参数说明

参数	参数描述
req	PutBucketVersioningReq，PutBucketVersioning 操作的请求
resp	PutBucketVersioningResp，PutBucketVersioning 操作的返回

PutBucketVersioningReq 提供以下成员函数：

```
// 版本是否开启，一经开启不能关闭，只能suspend
void SetStatus(bool is_enable);
```

查询版本控制

功能说明

查询指定存储桶的版本控制信息。

方法原型

```
CosResult GetBucketVersioning(const GetBucketVersioningReq& request, GetBucketVersioningResp* response)
```

请求示例

```
qcloud_cos::CosConfig config("./config.json");
qcloud_cos::CosAPI cos(config);

// 目的 Bucket 名称, 和配置文件中 DestDomain 的 bucket 相同
std::string bucket_name = "examplebucket-1250000000";

qcloud_cos::GetBucketVersioningReq req(bucket_name);
qcloud_cos::GetBucketVersioningResp resp;
qcloud_cos::CosResult result = cos.GetBucketVersioning(req, &resp);

// 调用成功，调用 resp 的成员函数获取返回内容
if (result.IsSucc()) {
// ...
} else {
// 可以调用 CosResult 的成员函数输出错误信息，如 requestID 等
}
```

参数说明

参数	参数描述
req	GetBucketVersioningReq，GetBucketVersioning 操作的请求
resp	GetBucketVersioningResp，GetBucketVersioning 操作的返回

GetBucketVersioningResp 提供以下成员函数：

```
// 返回bucket的版本状态,0: 从未开启版本管理, 1: 版本管理生效中, 2: 暂停
int GetStatus() const
```

跨地域复制

设置跨地域复制

功能说明

设置指定存储桶的跨地域复制规则。

方法原型

```
func (s *BucketService) PutBucketReplication(ctx context.Context, opt *PutBucketReplicationOptions) (*Response, error)
```

请求示例

```
qcloud_cos::CosConfig config("./config.json");
qcloud_cos::CosAPI cos(config);

// 目的 Bucket 名称, 和配置文件中 DestDomain 的 bucket 相同
std::string bucket_name = "examplebucket-1250000000";

qcloud_cos::PutBucketReplicationReq req(bucket_name);
req.SetRole("qcs::cam::uin/100000000001:uin/100000000001");
qcloud_cos::ReplicationRule rule("", "qcs::cos:ap-guangzhou::examplebucket-1250000000", "", "", true);

req.AddReplicationRule(rule);
qcloud_cos::PutBucketReplicationResp resp;
qcloud_cos::CosResult result = cos.PutBucketReplication(req, &resp);

// 调用成功，调用 resp 的成员函数获取返回内容
if (result.IsSucc()) {
// ...
} else {
// 可以调用 CosResult 的成员函数输出错误信息，如 requestID 等
}
```

参数说明

参数	参数描述
req	PutBucketReplicationReq，PutBucketReplication 操作的请求
resp	PutBucketReplicationResp，PutBucketReplication 操作的返回

PutBucketReplicationReq 提供以下成员函数：

```
// 发起者身份标示：qcs::cam::uin/<OwnerUin>:uin/<SubUin>
void SetRole(const std::string& role)；
// 添加具体配置信息，最多支持1000个，所有策略只能指向一个目标存储桶
void AddReplicationRule(const ReplicationRule& rule)；

// ReplicationRule 结构如下：
struct ReplicationRule {
bool m_is_enable;
std::string m_id; // 非必须
std::string m_prefix;
std::string m_dest_bucket;
std::string m_dest_storage_class; // 非必须
}
```

查询跨地域复制

功能说明

查询指定存储桶的跨地域复制规则。

方法原型

```
CosResult GetBucketReplication(const GetBucketReplicationReq& request, GetBucketReplicationResp* response)
```

请求示例

```
qcloud_cos::CosConfig config("./config.json");
qcloud_cos::CosAPI cos(config);

// 目的 Bucket 名称, 和配置文件中 DestDomain 的 bucket 相同
std::string bucket_name = "examplebucket-1250000000";

qcloud_cos::GetBucketReplicationReq req(bucket_name);
qcloud_cos::GetBucketReplicationResp resp;
qcloud_cos::CosResult result = cos.GetBucketReplication(req, &resp);

// 调用成功，调用 resp 的成员函数获取返回内容
if (result.IsSucc()) {
// ...
} else {
// 可以调用 CosResult 的成员函数输出错误信息，如 requestID 等
}
```

参数说明

参数	参数描述
req	GetBucketReplicationReq，GetBucketReplication 操作的请求
resp	GetBucketReplicationResp，GetBucketReplication 操作的返回

GetBucketReplicationResp 提供以下成员函数：

```
// 获取发起者身份标示：qcs::cam::uin/<OwnerUin>:uin/<SubUin>
std::string GetRole();
// 获取具体配置信息，最多支持 1000 个，所有策略只能指向一个目标存储桶
std::vector<ReplicationRule> GetRules();
// ReplicationRule 结构如PutBucketReplication中描述
```

删除跨地域复制

功能说明

删除指定存储桶的跨地域复制规则。

方法原型

```
CosResult DeleteBucketReplication(const DeleteBucketReplicationReq& request, DeleteBucketReplicationResp* response)
```

请求示例

```
qcloud_cos::CosConfig config("./config.json");
qcloud_cos::CosAPI cos(config);

std::string bucket_name = "examplebucket-1250000000";

qcloud_cos::DeleteBucketReplicationReq req(bucket_name);
qcloud_cos::DeleteBucketReplicationResp resp;
qcloud_cos::CosResult result = cos.DeleteBucketReplication(req, &resp);

// 调用成功，调用 resp 的成员函数获取返回内容
if (result.IsSucc()) {
// ...
} else {
// 可以调用 CosResult 的成员函数输出错误信息，如 requestID 等
}
```

参数说明

参数	参数描述
req	DeleteBucketReplicationReq，DeleteBucketReplication 操作的请求
resp	DeleteBucketReplicationResp，DeleteBucketReplication 操作的返回

预签名 URL

简介

C++ SDK 提供生成签名和获取请求预签名 URL 接口，详细操作请查看本文说明和示例。

生成签名

功能说明

计算并生成签名。

方法原型一

```
static std::string Sign(const std::string& secret_id,
const std::string& secret_key,
const std::string& http_method,
const std::string& in_uri,
const std::map<std::string, std::string>& headers,
const std::map<std::string, std::string>& params);
```

参数说明

参数名称	参数描述	类型

参数名称	参数描述	类型
secret_id	开发者拥有的项目身份识别 ID，用以身份认证	String
secret_key	开发者拥有的项目身份密钥	String
http_method	HTTP 方法，如 POST/GET/HEAD/PUT 等， 传入大小写不敏感	String
in_uri	HTTP uri	String
headers	HTTP header 的键值对	map
params	HTTP params 的键值对	map

返回结果说明

返回签名字符串，可以在指定的有效期内（通过 CosSysConfig 设置，默认60s）使用，返回空串表示计算签名失败。

方法原型二

```
static std::string Sign(const std::string& secret_id,
const std::string& secret_key,
const std::string& http_method,
const std::string& in_uri,
const std::map<std::string, std::string>& headers,
const std::map<std::string, std::string>& params,
uint64_t start_time_in_s,
uint64_t end_time_in_s);
```

参数说明

参数名称	参数描述	类型
secret_id	开发者拥有的项目身份识别 ID，用以身份认证	String
secret_key	开发者拥有的项目身份密钥	String
http_method	HTTP 方法，如 POST/GET/HEAD/PUT 等，传入大小写不敏感	String
in_uri	HTTP uri	String
headers	HTTP header 的键值对	map
params	HTTP params 的键值对	map
start_time_in_s	签名生效的开始时间	uint64_t
end_time_in_s	签名生效的截止时间	uint64_t

返回结果说明

返回签名字符串，可以在指定的有效期内（通过 CosSysConfig 设置，默认60s）使用，返回空串表示计算签名失败。

获取请求预签名 URL

```
std::string GeneratePresignedUrl(const GeneratePresignedUrlReq& req)
```

参数说明

参数	参数描述
req	GeneratePresignedUrlReq，GeneratePresignedUrl 操作的请求

HTTP_METHOD 枚举定义如下：

```
typedef enum {
HTTP_HEAD,
```

```
HTTP_GET,
HTTP_PUT,
HTTP_POST,
HTTP_DELETE,
HTTP_OPTIONS
} HTTP_METHOD;
```

预签名请求示例

可根据 CosConfig 类设置永久密钥或临时密钥发起预签名请求，具体配置文件内容请参阅 快速入门 文档。

```
qcloud_cos::CosConfig config("./config.json");
qcloud_cos::CosAPI cos(config);

// 目的 Bucket 名称, 和配置文件中 DestDomain 的 bucket 相同
std::string bucket_name = "examplebucket-1250000000";
std::string object_name = "exampleobject";

// 添加存储桶名称和对象键，以及 HTTP 请求方法。
qcloud_cos::GeneratePresignedUrlReq req(bucket_name, object_name, qcloud_cos::HTTP_GET);
std::string presigned_url = cos.GeneratePresignedUrl(req);
```

异常处理

简介

API 返回 CosResult 结构，如果成功可以获取对应 Response 结构中的数据，失败可以通过 CosResult 获取详细信息。

服务端异常

CosResult 封装了请求出错时返回的错误码和对应错误信息，详情请参阅 [错误码]。

SDK 内部封装的请求均会返回 CosResult 对象，每次调用完成后，均要使用 IsSucc() 成员函数判断本次调用是否成功。

成员函数

函数	函数描述
bool isSucc()	返回本次调用成功或失败 当返回 false 时：后续的 CosResult 成员函数才有意义 当返回 True 时：可以从 OperatorResp 中获取具体返回内容
string GetErrorCode()	获取 COS 返回的错误码，用来确定错误场景
string GetErrorMsg()	包含具体的错误信息
string GetResourceAddr()	资源地址，Bucket 地址或 Object 地址
string GetXCosRequestId()	当请求发送时，服务端将会自动为请求生成一个唯一的 ID。使用遇到问题时，request-id 能更快地协助 COS 定位问题
string GetXCosTraceId()	当请求出错时，服务端将会自动为这个错误生成一个唯一的 ID，trace-id 与 request-id 一一对应。使用遇到问题时，trace-id 能更快地协助 COS 定位问题
string GetErrorInfo()	获取 SDK 内部错误信息
int GetHttpStatus()	获取 HTTP 状态码

请求和响应基类

BaseReq、BaseResp 封装了请求和返回， 调用者只需要根据不同的操作类型生成不同的 OperatorReq，并填充 OperatorReq 的内容即可。 函数返回后，调用对应 BaseResp 的成员函数获取请求结果。

- 对于 Request，如无特殊说明，仅需要关注 Request 的构造函数。
- 对于 Response，所有方法的 Response 均有获取公共返回头部的成员函数。Response 的公共成员函数如下，具体字段含义请参阅 [公共返回头部]，此处不再赘述。

```
uint64_t GetContentLength();
std::string GetContentType();
std::string GetEtag();
std::string GetConnection();
std::string GetDate();
std::string GetServer();
std::string GetXCosRequestId();
std::string GetXCosTraceId();
```

C# SDK
快速入门

最近更新时间: 2025-01-15 17:01:00

下载与安装

相关资源

- 对象存储的 XML .NET SDK 源码下载地址： COS XML .NET SDK 。

环境依赖

COS XML .NET SDK 基于 .NET Standard 2.0 开发。

- Windows：安装 .NET Core 2.0 及以上版本，或者 .NET Framework 4.6.1 及以上版本。
- Linux/Mac：安装 .NET Core 2.0 及以上版本。

添加 SDK

我们提供 Nuget 的集成方式，您可以在工程的 csproj 文件里添加：

```
<PackageReference Include="Tencent.QCloud.Cos.Sdk" Version="5.4.*" />
```

如果是用 .NET CLI，请使用如下命令安装：

```
dotnet add package Tencent.QCloud.Cos.Sdk
```

您也可以在 这里 手动下载我们的SDK。

其他依赖

我们使用了 Newtonsoft.Json 作为第三方依赖，如果您本地没有自动拉取，可以在 csproj 文件里手动添加：

```
<PackageReference Include="Newtonsoft.Json" Version="12.0.2" />
```

开始使用

下面为您介绍如何使用 COS C# SDK 完成一个基础操作，如初始化客户端、创建存储桶、查询存储桶列表、上传对象、查询对象列表、下载对象和删除对象。

> **说明：**
>
> - 关于文章中出现的 SecretId、SecretKey、Bucket 等名称的含义和获取方式请参阅 COS 术语信息。
> - SDK 中常用的命名空间有： `using COSXML;using COSXML.Auth;using COSXML.Model.Object;using COSXML.Model.Bucket;using COSXML.CosException.`

初始化

在执行任何和 COS 服务相关请求之前，都需要先实例化 `CosXmlConfig` , `QCloudCredentialProvider` , `CosXmlServer` 3个对象。其中：

- `CosXmlConfig` 提供配置 SDK 接口。
- `QCloudCredentialProvider` 提供设置密钥信息接口。
- `CosXmlServer` 提供各种 COS API 服务接口。

```
//初始化 CosXmlConfig
string region = "ap-beijing"; //设置一个默认的存储桶地域
string domain = "DOMAIN.com"; // 替换成用户的 Domain

string endpoint = String.format("cos.%s.%s", region, domain);

CosXmlConfig config = new CosXmlConfig.Builder()
.setEndpointSuffix(endpoint)
.SetConnectionTimeoutMs(60000) //设置连接超时时间，单位 毫秒 ，默认 45000ms
.SetReadWriteTimeoutMs(40000) //设置读写超时时间，单位 毫秒 ，默认 45000ms
.IsHttps(true) //设置默认 https 请求
```

```
.SetRegion(region) //设置一个默认的存储桶地域
.SetDebugLog(true) //显示日志
.Build(); //创建 CosXmlConfig 对象

//初始化 QCloudCredentialProvider ，SDK中提供了3种方式：永久密钥 、 临时密钥 、 自定义
QCloudCredentialProvider cosCredentialProvider = null;

string secretId = "COS_SECRETID"; //"云 API 密钥 SecretId";
string secretKey = "COS_SECRETKEY"; //"云 API 密钥 SecretKey";
long durationSecond = 600; //secretKey 有效时长,单位为 秒
cosCredentialProvider = new DefaultQCloudCredentialProvider(secretId, secretKey, durationSecond);

//初始化 CosXmlServer
CosXmlServer cosXml = new CosXmlServer(config, cosCredentialProvider);
```

创建存储桶

```
try
{
string bucket = "examplebucket-1250000000"; //存储桶名称 格式：BucketName-APPID
PutBucketRequest request = new PutBucketRequest(bucket);
//设置签名有效时长
request.SetSign(TimeUtils.GetCurrentTime(TimeUnit.SECONDS), 600);
//执行请求
PutBucketResult result = cosXml.PutBucket(request);
//请求成功
Console.WriteLine(result.GetResultInfo());
}
catch (COSXML.CosException.CosClientException clientEx)
{
//请求失败
Console.WriteLine("CosClientException: " + clientEx.Message);
}
catch (COSXML.CosException.CosServerException serverEx)
{
//请求失败
Console.WriteLine("CosServerException: " + serverEx.GetInfo());
}
```

查询存储桶列表

```
try
{
GetServiceRequest request = new GetServiceRequest();
//设置签名有效时长
request.SetSign(TimeUtils.GetCurrentTime(TimeUnit.SECONDS), 600);
//执行请求
GetServiceResult result = cosXml.GetService(request);
//请求成功
Console.WriteLine(result.GetResultInfo());
}
catch (COSXML.CosException.CosClientException clientEx)
{
//请求失败
Console.WriteLine("CosClientException: " + clientEx.Message);
}
catch (COSXML.CosException.CosServerException serverEx)
{
//请求失败
Console.WriteLine("CosServerException: " + serverEx.GetInfo());
}
```

上传对象

```
try
{
string bucket = "examplebucket-1250000000"; //存储桶，格式：BucketName-APPID
string key = "exampleobject"; //对象在存储桶中的位置，即称对象键.
```

```
string srcPath = @"F:\exampleobject";//本地文件绝对路径
PutObjectRequest request = new PutObjectRequest(bucket, key, srcPath);
//设置签名有效时长
request.SetSign(TimeUtils.GetCurrentTime(TimeUnit.SECONDS), 600);
//设置进度回调
request.SetCosProgressCallback(delegate(long completed, long total)
{
Console.WriteLine(String.Format("progress = {0:##.##}%", completed * 100.0 / total));
});
//执行请求
PutObjectResult result = cosXml.PutObject(request);
//请求成功
Console.WriteLine(result.GetResultInfo());
}
catch (COSXML.CosException.CosClientException clientEx)
{
//请求失败
Console.WriteLine("CosClientException: " + clientEx.Message);
}
catch (COSXML.CosException.CosServerException serverEx)
{
//请求失败
Console.WriteLine("CosServerException: " + serverEx.GetInfo());
}

**// 大文件需要使用分片上传(), 可参考 SDK 中封装的 TransferManager 和 COSXMLUploadTask 类, 如下示例 **
TransferManager transferManager = new TransferManager(cosXml, new TransferConfig());
COSXMLUploadTask uploadTask = new COSXMLUploadTask(bucket, null, key);
uploadTask.SetSrcPath(srcPath);
uploadTask.progressCallback = delegate (long completed, long total)
{
Console.WriteLine(String.Format("progress = {0:##.##}%", completed * 100.0 / total));
};
uploadTask.successCallback = delegate (CosResult cosResult)
{
COSXML.Transfer.COSXMLUploadTask.UploadTaskResult result = cosResult as COSXML.Transfer.COSXMLUploadTask.UploadTaskResult;
Console.WriteLine(result.GetResultInfo());
};
uploadTask.failCallback = delegate (CosClientException clientEx, CosServerException serverEx)
{
if (clientEx != null)
{
Console.WriteLine("CosClientException: " + clientEx.Message);
}
if (serverEx != null)
{
Console.WriteLine("CosServerException: " + serverEx.GetInfo());
}
};
transferManager.Upload(uploadTask);
```

查询对象列表

```
try
{
string bucket = "examplebucket-1250000000"; //格式：BucketName-APPID
GetBucketRequest request = new GetBucketRequest(bucket);
//设置签名有效时长
request.SetSign(TimeUtils.GetCurrentTime(TimeUnit.SECONDS), 600);
//执行请求
GetBucketResult result = cosXml.GetBucket(request);
//请求成功
Console.WriteLine(result.GetResultInfo());
}
catch (COSXML.CosException.CosClientException clientEx)
{
//请求失败
Console.WriteLine("CosClientException: " + clientEx.Message);
}
catch (COSXML.CosException.CosServerException serverEx)
{
```

```
//请求失败
Console.WriteLine("CosServerException: " + serverEx.GetInfo());
}
```

下载对象

```
try
{
string bucket = "examplebucket-1250000000"; //存储桶，格式：BucketName-APPID
string key = "exampleobject"; //对象在存储桶中的位置，即称对象键.
string localDir = @"F:\";//下载到本地指定文件夹
string localFileName = "exampleobject"; //指定本地保存的文件名
GetObjectRequest request = new GetObjectRequest(bucket, key, localDir, localFileName);
//设置签名有效时长
request.SetSign(TimeUtils.GetCurrentTime(TimeUnit.SECONDS), 600);
//设置进度回调
request.SetCosProgressCallback(delegate(long completed, long total)
{
Console.WriteLine(String.Format("progress = {0:##.##}%", completed * 100.0 / total));
});
//执行请求
GetObjectResult result = cosXml.GetObject(request);
//请求成功
Console.WriteLine(result.GetResultInfo());
}
catch (COSXML.CosException.CosClientException clientEx)
{
//请求失败
Console.WriteLine("CosClientException: " + clientEx.Message);
}
catch (COSXML.CosException.CosServerException serverEx)
{
//请求失败
Console.WriteLine("CosServerException: " + serverEx.GetInfo());
}
```

删除对象

```
try
{
string bucket = "examplebucket-1250000000"; //存储桶，格式：BucketName-APPID
string key = "exampleobject"; //对象在存储桶中的位置，即称对象键.
DeleteObjectRequest request = new DeleteObjectRequest(bucket, key);
//设置签名有效时长
request.SetSign(TimeUtils.GetCurrentTime(TimeUnit.SECONDS), 600);
//执行请求
DeleteObjectResult result = cosXml.DeleteObject(request);
//请求成功
Console.WriteLine(result.GetResultInfo());
}
catch (COSXML.CosException.CosClientException clientEx)
{
//请求失败
Console.WriteLine("CosClientException: " + clientEx.Message);
}
catch (COSXML.CosException.CosServerException serverEx)
{
//请求失败
Console.WriteLine("CosServerException: " + serverEx.GetInfo());
}
```

接口文档

最近更新时间: 2025-01-15 17:01:00

Bucket操作

创建存储桶

功能说明

在指定账号下创建一个存储桶。

方法原型

```
PutBucketResult PutBucket(PutBucketRequest request);

void PutBucket(PutBucketRequest request, COSXML.Callback.OnSuccessCallback<CosResult> successCallback, COSXML.Callback.OnFailedCallback failCallback);
```

请求示例

```
try
{
string bucket = "examplebucket-1250000000"; //格式：BucketName-APPID
PutBucketRequest request = new PutBucketRequest(bucket);
//设置签名有效时长
request.SetSign(TimeUtils.GetCurrentTime(TimeUnit.SECONDS), 600);
//执行请求
PutBucketResult result = cosXml.PutBucket(request);
//请求成功
Console.WriteLine(result.GetResultInfo());
}
catch (COSXML.CosException.CosClientException clientEx)
{
//请求失败
Console.WriteLine("CosClientException: " + clientEx.Message);
}
catch (COSXML.CosException.CosServerException serverEx)
{
//请求失败
Console.WriteLine("CosServerException: " + serverEx.GetInfo());
}

/**
//异步方法
string bucket = "examplebucket-1250000000"; //格式：BucketName-APPID
PutBucketRequest request = new PutBucketRequest(bucket);
//设置签名有效时长
request.SetSign(TimeUtils.GetCurrentTime(TimeUnit.SECONDS), 600);
cosXml.PutBucket(request,
delegate(COSXML.Model.CosResult cosResult)
{
//请求成功
PutBucketResult result = cosResult as PutBucketResult;
Console.WriteLine(result.GetResultInfo());

},
delegate(COSXML.CosException.CosClientException clientEx, COSXML.CosException.CosServerException serverEx)
{
//请求失败
if (clientEx != null)
{
Console.WriteLine("CosClientException: " + clientEx.Message);
}
else if (serverEx != null)
{
Console.WriteLine("CosServerException: " + serverEx.GetInfo());
}
```

```
});
*/
```

参数说明

参数名称	设置方法	描述	类型
bucket	构造方法	存储桶名称，格式：BucketName-APPID	string
signStartTimeSecond	SetSign	签名有效期起始时间（Unix 时间戳），例如1557902800	long
durationSecond	SetSign	签名有效期时长（单位为秒），例如签名有效时期为1分钟：60	long
headerKeys	SetSign	签名是否校验 header	`List`
queryParameterKeys	SetSign	签名是否校验请求 url 中查询参数	`List`

返回结果说明

通过 PutBucketResult 返回请求结果。

成员变量	类型	描述
httpCode	int	HTTP Code， [200，300)之间表示操作成功，否则表示操作失败

> **说明：**
>
> 操作失败时，系统将抛出 CosClientException（客户端异常）或 CosServerException（服务端异常）异常。

检索存储桶及其权限

功能说明

检索存储桶是否存在且是否有权限访问。

方法原型

```
HeadBucketResult HeadBucket(HeadBucketRequest request);

void HeadBucket(HeadBucketRequest request, COSXML.Callback.OnSuccessCallback<CosResult> successCallback, COSXML.Callback.OnFailedCallback fail
Callback);
```

请求示例

```
try
{
string bucket = "examplebucket-1250000000"; //格式：BucketName-APPID
HeadBucketRequest request = new HeadBucketRequest(bucket);
//设置签名有效时长
request.SetSign(TimeUtils.GetCurrentTime(TimeUnit.SECONDS), 600);
//执行请求
HeadBucketResult result = cosXml.HeadBucket(request);
//请求成功
Console.WriteLine(result.GetResultInfo());
}
catch (COSXML.CosException.CosClientException clientEx)
{
//请求失败
Console.WriteLine("CosClientException: " + clientEx.Message);
}
catch (COSXML.CosException.CosServerException serverEx)
{
//请求失败
Console.WriteLine("CosServerException: " + serverEx.GetInfo());
}

/**
//异步方法
string bucket = "examplebucket-1250000000"; //格式：BucketName-APPID
HeadBucketRequest request = new HeadBucketRequest(bucket);
```

```
//设置签名有效时长
request.SetSign(TimeUtils.GetCurrentTime(TimeUnit.SECONDS), 600);
cosXml.HeadBucket(request,
delegate(COSXML.Model.CosResult cosResult)
{
//请求成功
HeadBucketResult result = cosResult as HeadBucketResult;
Console.WriteLine(result.GetResultInfo());

},
delegate(COSXML.CosException.CosClientException clientEx, COSXML.CosException.CosServerException serverEx)
{
//请求失败
if (clientEx != null)
{
Console.WriteLine("CosClientException: " + clientEx.Message);
}
else if (serverEx != null)
{
Console.WriteLine("CosServerException: " + serverEx.GetInfo());
}
});
*/
```

删除存储桶

功能说明

删除指定账号下的空存储桶。

方法原型

```
DeleteBucketResult DeleteBucket(DeleteBucketRequest request);

void DeleteBucket(DeleteBucketRequest request, COSXML.Callback.OnSuccessCallback<CosResult> successCallback, COSXML.Callback.OnFailedCallback fa
ilCallback);
```

请求示例

```
try
{
string bucket = "examplebucket-1250000000"; //格式：BucketName-APPID
DeleteBucketRequest request = new DeleteBucketRequest(bucket);
//设置签名有效时长
request.SetSign(TimeUtils.GetCurrentTime(TimeUnit.SECONDS), 600);
//执行请求
DeleteBucketResult result = cosXml.DeleteBucket(request);
//请求成功
Console.WriteLine(result.GetResultInfo());
}
catch (COSXML.CosException.CosClientException clientEx)
{
//请求失败
Console.WriteLine("CosClientException: " + clientEx.Message);
}
catch (COSXML.CosException.CosServerException serverEx)
{
//请求失败
Console.WriteLine("CosServerException: " + serverEx.GetInfo());
}

/**
//异步方法
string bucket = "examplebucket-1250000000"; //格式：BucketName-APPID
DeleteBucketRequest request = new DeleteBucketRequest(bucket);
//设置签名有效时长
request.SetSign(TimeUtils.GetCurrentTime(TimeUnit.SECONDS), 600);
cosXml.DeleteBucket(request,
delegate(COSXML.Model.CosResult cosResult)
{
```

```
//请求成功
DeleteBucketResult result = cosResult as DeleteBucketResult;
Console.WriteLine(result.GetResultInfo());

},
delegate(COSXML.CosException.CosClientException clientEx, COSXML.CosException.CosServerException serverEx)
{
//请求失败
if (clientEx != null)
{
Console.WriteLine("CosClientException: " + clientEx.Message);
}
else if (serverEx != null)
{
Console.WriteLine("CosServerException: " + serverEx.GetInfo());
}
});
*/
```

参数说明

参数名称	设置方法	描述	类型
bucket	构造方法	存储桶名称，格式：BucketName-APPID	string
signStartTimeSecond	SetSign	签名有效期起始时间（Unix 时间戳），例如1557902800	long
durationSecond	SetSign	签名有效期时长（单位为秒），例如签名有效时期为1分钟：60	long
headerKeys	SetSign	签名是否校验 header	`List`
queryParameterKeys	SetSign	签名是否校验请求 url 中查询参数	`List`

返回结果说明

通过 DeleteBucketResult 返回请求结果。

成员变量	类型	描述
httpCode	int	HTTP Code， [200，300)之间表示操作成功，否则表示操作失败

> **说明：**
>
> 操作失败时，系统将抛出 CosClientException（客户端异常） 或 CosServerException（服务端异常）异常。

设置存储桶 ACL

功能说明

设置指定存储桶访问权限控制列表。

方法原型

```
PutBucketACLResult PutBucketACL(PutBucketACLRequest request);

void PutBucketACL(PutBucketACLRequest request, COSXML.Callback.OnSuccessCallback<CosResult> successCallback, COSXML.Callback.OnFailedCallback failCallback);
```

请求示例

```
try
{
string bucket = "examplebucket-1250000000"; //格式：BucketName-APPID
PutBucketACLRequest request = new PutBucketACLRequest(bucket);
//设置签名有效时长
request.SetSign(TimeUtils.GetCurrentTime(TimeUnit.SECONDS), 600);
//设置私有读写权限
request.SetCosACL(CosACL.PRIVATE);
//授予1131975903账号读权限
COSXML.Model.Tag.GrantAccount readAccount = new COSXML.Model.Tag.GrantAccount();
```

```
readAccount.AddGrantAccount("1131975903", "1131975903");
request.SetXCosGrantRead(readAccount);
//执行请求
PutBucketACLResult result = cosXml.PutBucketACL(request);
//请求成功
Console.WriteLine(result.GetResultInfo());
}
catch (COSXML.CosException.CosClientException clientEx)
{
//请求失败
Console.WriteLine("CosClientException: " + clientEx.Message);
}
catch (COSXML.CosException.CosServerException serverEx)
{
//请求失败
Console.WriteLine("CosServerException: " + serverEx.GetInfo());
}

/**
//异步方法
string bucket = "examplebucket-1250000000"; //格式：BucketName-APPID
PutBucketACLRequest request = new PutBucketACLRequest(bucket);
//设置签名有效时长
request.SetSign(TimeUtils.GetCurrentTime(TimeUnit.SECONDS), 600);
//设置私有读写权限
request.SetCosACL(CosACL.PRIVATE);
//授予1131975903账号读权限
COSXML.Model.Tag.GrantAccount readAccount = new COSXML.Model.Tag.GrantAccount();
readAccount.AddGrantAccount("1131975903", "1131975903");
request.SetXCosGrantRead(readAccount);
//执行请求
cosXml.PutBucketACL(request,
delegate(COSXML.Model.CosResult cosResult)
{
//请求成功
PutBucketACLResult result = cosResult as PutBucketACLResult;
Console.WriteLine(result.GetResultInfo());

},
delegate(COSXML.CosException.CosClientException clientEx, COSXML.CosException.CosServerException serverEx)
{
//请求失败
if (clientEx != null)
{
Console.WriteLine("CosClientException: " + clientEx.Message);
}
else if (serverEx != null)
{
Console.WriteLine("CosServerException: " + serverEx.GetInfo());
}
});
*/
```

参数说明

参数名称	设置方法	描述	类型
bucket	构造方法	存储桶名称，格式：BucketName-APPID	string
cosAcl	SetCosAcl	设置存储桶的 ACL 权限	string
grantAccount	SetXCosGrantRead 或 SetXCosGrantWrite 或 SetXCosReadWrite	授予用户读写权限	GrantAccount
signStartTimeSecond	SetSign	签名有效期起始时间（Unix 时间戳），例如1557902800	long
durationSecond	SetSign	签名有效期时长（单位为秒），例如签名有效时期为1分钟：60	long
headerKeys	SetSign	签名是否校验 header	`List`

参数名称	设置方法	描述	类型
queryParameterKeys	SetSign	签名是否校验请求 url 中查询参数	`List`

返回结果说明

通过 PutBucketACLResult 返回请求结果。

成员变量	类型	描述
httpCode	int	HTTP Code，[200，300)之间表示操作成功，否则表示操作失败

> **说明：**
>
> 操作失败时，系统将抛出 CosClientException（客户端异常）或 CosServerException（服务端异常）异常。

查询存储桶 ACL

功能说明

查询存储桶的访问控制列表。

方法原型

```
GetBucketACLResult GetBucketACL(GetBucketACLRequest request);

void GetBucketACL(GetBucketACLRequest request, COSXML.Callback.OnSuccessCallback<CosResult> successCallback, COSXML.Callback.OnFailedCallback failCallback);
```

请求示例

```
try
{
string bucket = "examplebucket-1250000000"; //格式：BucketName-APPID
GetBucketACLRequest request = new GetBucketACLRequest(bucket);
//设置签名有效时长
request.SetSign(TimeUtils.GetCurrentTime(TimeUnit.SECONDS), 600);
//执行请求
GetBucketACLResult result = cosXml.GetBucketACL(request);
//请求成功
Console.WriteLine(result.GetResultInfo());
}
catch (COSXML.CosException.CosClientException clientEx)
{
//请求失败
Console.WriteLine("CosClientException: " + clientEx.Message);
}
catch (COSXML.CosException.CosServerException serverEx)
{
//请求失败
Console.WriteLine("CosServerException: " + serverEx.GetInfo());
}

/**
//异步方法
string bucket = "examplebucket-1250000000"; //格式：BucketName-APPID
GetBucketACLRequest request = new GetBucketACLRequest(bucket);
//设置签名有效时长
request.SetSign(TimeUtils.GetCurrentTime(TimeUnit.SECONDS), 600);
cosXml.GetBucketACL(request,
delegate(COSXML.Model.CosResult cosResult)
{
//请求成功
GetBucketACLResult result = cosResult as GetBucketACLResult;
Console.WriteLine(result.GetResultInfo());

},
delegate(COSXML.CosException.CosClientException clientEx, COSXML.CosException.CosServerException serverEx)
{
```

```
//请求失败
if (clientEx != null)
{
Console.WriteLine("CosClientException: " + clientEx.Message);
}
else if (serverEx != null)
{
Console.WriteLine("CosServerException: " + serverEx.GetInfo());
}
});
*/
```

参数说明

参数名称	设置方法	描述	类型
bucket	构造方法	存储桶名称，格式：BucketName-APPID	string
signStartTimeSecond	SetSign	签名有效期起始时间（Unix 时间戳），例如1557902800	long
durationSecond	SetSign	签名有效期时长（单位为秒），例如签名有效时期为1分钟：60	long
headerKeys	SetSign	签名是否校验 header	`List`
queryParameterKeys	SetSign	签名是否校验请求 url 中查询参数	`List`

返回结果说明

通过 GetBucketACLResult 返回请求结果。

成员变量	类型	描述
httpCode	int	HTTP Code， [200，300)之间表示操作成功，否则表示操作失败
accessControlPolicy	AccessControlPolicy	返回 Bucket 访问权限列表信息

> **说明：**
>
> 操作失败时，系统将抛出 CosClientException（客户端异常）或 CosServerException（服务端异常）异常。

Get Bucket CORS

功能说明

Get Bucket CORS 实现跨域访问读取。

操作方法原型

- 调用 Get Bucket CORS 操作

```
var params = {
Bucket : 'STRING_VALUE', /* 必须 */
Region : 'STRING_VALUE' /* 必须 */
};

cos.getBucketCors(params, function(err, data) {
if(err) {
console.log(err);
} else {
console.log(data);
}
});
```

操作参数说明

参数名	参数描述	类型	必填
Bucket	Bucket 的名称。命名规则为{name}-{appid} ，此处填写的存储桶名称必须为此格式	String	是

参数名	参数描述	类型	必填
Region	Bucket 所在区域。	String	是

回调函数说明

```
function(err, data) { ... }
```

回调参数说明

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误。如果请求成功，则为空	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
CORSRule	配置的信息集合	Array
AllowedMethod	允许的 HTTP 操作，枚举值：Get，Put，Head，Post，Delete	Array
AllowedOrigin	允许的访问来源，支持『 * 』通配符	Array
AllowedHeader	在发送 OPTIONS 请求时告知服务端，接下来的请求可以使用哪些自定义的 HTTP 请求头部	Array
ExposeHeader	设置浏览器可以接收到的来自服务器端的自定义头部信息	Array
MaxAgeSeconds	设置 OPTIONS 请求得到结果的有效期	String
ID	规则名称	String

设置跨域配置

功能说明

设置指定存储桶的跨域访问配置信息。

方法原型

```
PutBucketCORSResult PutBucketCORS(PutBucketCORSRequest request);

void PutBucketCORS(PutBucketCORSRequest request, COSXML.Callback.OnSuccessCallback<CosResult> successCallback, COSXML.Callback.OnFailedCallback failCallback);
```

请求示例

```
try
{
string bucket = "examplebucket-1250000000"; //格式：BucketName-APPID
PutBucketCORSRequest request = new PutBucketCORSRequest(bucket);
//设置签名有效时长
request.SetSign(TimeUtils.GetCurrentTime(TimeUnit.SECONDS), 600);
//设置跨域访问配置 CORS
COSXML.Model.Tag.CORSConfiguration.CORSRule corsRule = new COSXML.Model.Tag.CORSConfiguration.CORSRule();
corsRule.id = "corsconfigureId";
corsRule.maxAgeSeconds = 6000;
corsRule.allowedOrigin = "http://imgcache.finance.cloud.tencent.com:80finance.cloud.tencent.com";

corsRule.allowedMethods = new List<string>();
corsRule.allowedMethods.Add("PUT");

corsRule.allowedHeaders = new List<string>();
corsRule.allowedHeaders.Add("Host");

corsRule.exposeHeaders = new List<string>();
corsRule.exposeHeaders.Add("x-cos-meta-x1");

request.SetCORSRule(corsRule);

//执行请求
```

```
PutBucketCORSResult result = cosXml.PutBucketCORS(request);
//请求成功
Console.WriteLine(result.GetResultInfo());
}
catch (COSXML.CosException.CosClientException clientEx)
{
//请求失败
Console.WriteLine("CosClientException: " + clientEx.Message);
}
catch (COSXML.CosException.CosServerException serverEx)
{
//请求失败
Console.WriteLine("CosServerException: " + serverEx.GetInfo());
}

/**
//异步方法
string bucket = "examplebucket-1250000000"; //格式：BucketName-APPID
PutBucketCORSRequest request = new PutBucketCORSRequest(bucket);
//设置签名有效时长
request.SetSign(TimeUtils.GetCurrentTime(TimeUnit.SECONDS), 600);

//设置跨域访问配置 CORS
COSXML.Model.Tag.CORSConfiguration.CORSRule corsRule = new COSXML.Model.Tag.CORSConfiguration.CORSRule();
corsRule.id = "corsconfigureId";
corsRule.maxAgeSeconds = 6000;
corsRule.allowedOrigin = "http://imgcache.finance.cloud.tencent.com:80finance.cloud.tencent.com";

corsRule.allowedMethods = new List<string>();
corsRule.allowedMethods.Add("PUT");

corsRule.allowedHeaders = new List<string>();
corsRule.allowedHeaders.Add("Host");

corsRule.exposeHeaders = new List<string>();
corsRule.exposeHeaders.Add("x-cos-meta-x1");

request.SetCORSRule(corsRule);

cosXml.PutBucketCORS(request,
delegate(COSXML.Model.CosResult cosResult)
{
//请求成功
PutBucketCORSResult result = cosResult as PutBucketCORSResult;
Console.WriteLine(result.GetResultInfo());

},
delegate(COSXML.CosException.CosClientException clientEx, COSXML.CosException.CosServerException serverEx)
{
//请求失败
if (clientEx != null)
{
Console.WriteLine("CosClientException: " + clientEx.Message);
}
else if (serverEx != null)
{
Console.WriteLine("CosServerException: " + serverEx.GetInfo());
}
});
*/
```

参数说明

参数名称	设置方法	描述	类型
bucket	构造方法	存储桶名称，格式：BucketName-APPID	string
corsRule	SetCORSRule	设置存储桶的跨域访问配	CORSConfiguration.CORSRule
signStartTimeSecond	SetSign	签名有效期起始时间（Unix 时间戳），例如1557902800	long

参数名称	设置方法	描述	类型
durationSecond	SetSign	签名有效期时长（单位为秒），例如签名有效时期为1分钟：60	long
headerKeys	SetSign	签名是否校验 header	`List`
queryParameterKeys	SetSign	签名是否校验请求 url 中查询参数	`List`

返回结果说明

通过 PutBucketCORSResult 返回请求结果。

成员变量	类型	描述
httpCode	int	HTTP Code， [200, 300)之间表示操作成功，否则表示操作失败

查询跨域配置

功能说明

查询指定存储桶的跨域访问配置信息。

方法原型

```
GetBucketCORSResult GetBucketCORS(GetBucketCORSRequest request);

void GetBucketCORS(GetBucketCORSRequest request, COSXML.Callback.OnSuccessCallback<CosResult> successCallback, COSXML.Callback.OnFailedCallback failCallback);
```

请求示例

```
try
{
string bucket = "examplebucket-1250000000"; //格式：BucketName-APPID
GetBucketCORSRequest request = new GetBucketCORSRequest(bucket);
//设置签名有效时长
request.SetSign(TimeUtils.GetCurrentTime(TimeUnit.SECONDS), 600);
//执行请求
GetBucketCORSResult result = cosXml.GetBucketCORS(request);
//请求成功
Console.WriteLine(result.GetResultInfo());
}
catch (COSXML.CosException.CosClientException clientEx)
{
//请求失败
Console.WriteLine("CosClientException: " + clientEx.Message);
}
catch (COSXML.CosException.CosServerException serverEx)
{
//请求失败
Console.WriteLine("CosServerException: " + serverEx.GetInfo());
}

/**
//异步方法
string bucket = "examplebucket-1250000000"; //格式：BucketName-APPID
GetBucketCORSRequest request = new GetBucketCORSRequest(bucket);
//设置签名有效时长
request.SetSign(TimeUtils.GetCurrentTime(TimeUnit.SECONDS), 600);
//执行请求
cosXml.GetBucketCORS(request,
delegate(COSXML.Model.CosResult cosResult)
{
//请求成功
GetBucketCORSResult result = cosResult as GetBucketCORSResult;
Console.WriteLine(result.GetResultInfo());

},
delegate(COSXML.CosException.CosClientException clientEx, COSXML.CosException.CosServerException serverEx)
{
```

```
//请求失败
if (clientEx != null)
{
Console.WriteLine("CosClientException: " + clientEx.Message);
}
else if (serverEx != null)
{
Console.WriteLine("CosServerException: " + serverEx.GetInfo());
}
});
*/
```

参数说明

参数名称	设置方法	描述	类型
bucket	构造方法	存储桶名称，格式：BucketName-APPID	string
signStartTimeSecond	SetSign	签名有效期起始时间（Unix 时间戳），例如1557902800	long
durationSecond	SetSign	签名有效期时长（单位为秒），例如签名有效时期为1分钟：60	long
headerKeys	SetSign	签名是否校验 header	`List`
queryParameterKeys	SetSign	签名是否校验请求 url 中查询参数	`List`

返回结果说明

通过 GetBucketCORSResult 返回请求结果。

成员变量	类型	描述
httpCode	int	HTTP Code， [200, 300)之间表示操作成功，否则表示操作失败
corsConfiguration	CORSConfiguration	返回 Bucket 跨域资源共享配置的信息

删除跨域配置

功能说明

删除指定存储桶的跨域访问配置。

方法原型

```
DeleteBucketCORSResult DeleteBucketCORS(DeleteBucketCORSRequest request);

void DeleteBucketCORS(DeleteBucketCORSRequest request, COSXML.Callback.OnSuccessCallback<CosResult> successCallback, COSXML.Callback.OnFaile
dCallback failCallback);
```

请求示例

```
try
{
string bucket = "examplebucket-1250000000"; //格式：BucketName-APPID
DeleteBucketCORSRequest request = new DeleteBucketCORSRequest(bucket);
//设置签名有效时长
request.SetSign(TimeUtils.GetCurrentTime(TimeUnit.SECONDS), 600);
//执行请求
DeleteBucketCORSResult result = cosXml.DeleteBucketCORS(request);
//请求成功
Console.WriteLine(result.GetResultInfo());
}
catch (COSXML.CosException.CosClientException clientEx)
{
//请求失败
Console.WriteLine("CosClientException: " + clientEx.Message);
}
catch (COSXML.CosException.CosServerException serverEx)
{
//请求失败
```

```
Console.WriteLine("CosServerException: " + serverEx.GetInfo());
}

/**
//异步方法
string bucket = "examplebucket-1250000000"; //格式：BucketName-APPID
DeleteBucketCORSRequest request = new DeleteBucketCORSRequest(bucket);
//设置签名有效时长
request.SetSign(TimeUtils.GetCurrentTime(TimeUnit.SECONDS), 600);
//执行请求
cosXml.DeleteBucketCORS(request,
delegate(COSXML.Model.CosResult cosResult)
{
//请求成功
DeleteBucketCORSResult result = cosResult as DeleteBucketCORSResult;
Console.WriteLine(result.GetResultInfo());

},
delegate(COSXML.CosException.CosClientException clientEx, COSXML.CosException.CosServerException serverEx)
{
//请求失败
if (clientEx != null)
{
Console.WriteLine("CosClientException: " + clientEx.Message);
}
else if (serverEx != null)
{
Console.WriteLine("CosServerException: " + serverEx.GetInfo());
}
});
*/
```

参数说明

参数名称	设置方法	描述	类型
bucket	构造方法	存储桶名称，格式：BucketName-APPID	string
signStartTimeSecond	SetSign	签名有效期起始时间（Unix 时间戳），例如1557902800	long
durationSecond	SetSign	签名有效期时长（单位为秒），例如签名有效时期为1分钟：60	long
headerKeys	SetSign	签名是否校验 header	`List`
queryParameterKeys	SetSign	签名是否校验请求 url 中查询参数	`List`

返回结果说明

通过 DeleteBucketCORSResult 返回请求结果。

成员变量	类型	描述
httpCode	int	HTTP Code， [200, 300)之间表示操作成功，否则表示操作失败

Object操作

简单上传对象

功能说明

上传一个对象至存储桶。

方法原型

```
PutObjectResult PutObject(PutObjectRequest request);

void PutObject(PutObjectRequest request, COSXML.Callback.OnSuccessCallback<CosResult> successCallback, COSXML.Callback.OnFailedCallback failCallback);
```

请求示例

```
try
{
string bucket = "examplebucket-1250000000"; //存储桶，格式：BucketName-APPID
string key = "exampleobject"; //对象在存储桶中的位置，即称对象键
string srcPath = @"F:\exampleobject";//本地文件绝对路径
PutObjectRequest request = new PutObjectRequest(bucket, key, srcPath);
//设置签名有效时长
request.SetSign(TimeUtils.GetCurrentTime(TimeUnit.SECONDS), 600);
//设置进度回调
request.SetCosProgressCallback(delegate(long completed, long total)
{
Console.WriteLine(String.Format("progress = {0:##.##}%", completed * 100.0 / total));
});
//执行请求
PutObjectResult result = cosXml.PutObject(request);
//请求成功
Console.WriteLine(result.GetResultInfo());
}
catch (COSXML.CosException.CosClientException clientEx)
{
//请求失败
Console.WriteLine("CosClientException: " + clientEx.Message);
}
catch (COSXML.CosException.CosServerException serverEx)
{
//请求失败
Console.WriteLine("CosServerException: " + serverEx.GetInfo());
}

/**
//异步方法
string bucket = "examplebucket-1250000000"; //存储桶，格式：BucketName-APPID
string key = "exampleobject"; //对象在存储桶中的位置，即称对象键
string srcPath = @"F:\exampleobject"; //本地文件绝对路径
PutObjectRequest request = new PutObjectRequest(bucket, key, srcPath);
//设置签名有效时长
request.SetSign(TimeUtils.GetCurrentTime(TimeUnit.SECONDS), 600);
//设置进度回调
request.SetCosProgressCallback(delegate(long completed, long total)
{
Console.WriteLine(String.Format("progress = {0:##.##}%", completed * 100.0 / total));
});
//执行请求
cosXml.PutObject(request,
delegate(COSXML.Model.CosResult cosResult)
{
//请求成功
PutObjectResult result = cosResult as PutObjectResult;
Console.WriteLine(result.GetResultInfo());

},
delegate(COSXML.CosException.CosClientException clientEx, COSXML.CosException.CosServerException serverEx)
{
//请求失败
if (clientEx != null)
{
Console.WriteLine("CosClientException: " + clientEx.Message);
}
else if (serverEx != null)
{
Console.WriteLine("CosServerException: " + serverEx.GetInfo());
}
});
*/
```

参数说明

参数名称	设置方法	描述	类型

参数名称	设置方法	描述	类型
bucket	构造方法	存储桶名称，格式：BucketName-APPID	string
key	构造方法或 SetCosPath	存储于 COS 上 Object 的 对象键	string
srcPath	构造方法	用于上传到 COS 的本地文件的绝对路径	string
data	构造方法	用于上传到 COS 的 byte 数组	byte[]
progressCallback	SetCosProgressCallback	设置上传进度回调	Callback.OnProgressCallback
signStartTimeSecond	SetSign	签名有效期起始时间（Unix 时间戳），例如1557902800	long
durationSecond	SetSign	签名有效期时长（单位为秒），例如签名有效时期为1分钟：60	long
headerKeys	SetSign	签名是否校验 header	`List`
queryParameterKeys	SetSign	签名是否校验请求 url 中查询参数	`List`

返回结果说明

通过 PutObjectResult 返回请求结果。

成员变量	类型	描述
httpCode	int	HTTP Code，[200， 300)之间表示操作成功，否则表示操作失败
eTag	string	返回对象的 eTag

> **说明：**
>
> 操作失败时，系统将抛出 CosClientException（客户端异常）或 CosServerException（服务端异常）异常。

查询对象元数据

功能说明

查询对象的元数据信息。

方法原型

```
HeadObjectResult HeadObject(HeadObjectRequest request);

void HeadObject(HeadObjectRequest request, COSXML.Callback.OnSuccessCallback<CosResult> successCallback, COSXML.Callback.OnFailedCallback failCallback);
```

请求示例

```
try
{
string bucket = "examplebucket-1250000000"; //存储桶，格式：BucketName-APPID
string key = "exampleobject"; //对象在存储桶中的位置，即称对象键
HeadObjectRequest request = new HeadObjectRequest(bucket, key);
//设置签名有效时长
request.SetSign(TimeUtils.GetCurrentTime(TimeUnit.SECONDS), 600);
//执行请求
HeadObjectResult result = cosXml.HeadObject(request);
//请求成功
Console.WriteLine(result.GetResultInfo());
}
catch (COSXML.CosException.CosClientException clientEx)
{
//请求失败
Console.WriteLine("CosClientException: " + clientEx.Message);
}
catch (COSXML.CosException.CosServerException serverEx)
{
//请求失败
Console.WriteLine("CosServerException: " + serverEx.GetInfo());
```

```
}

/**
//异步方法
string bucket = "examplebucket-1250000000"; //存储桶，格式：BucketName-APPID
string key = "exampleobject"; //对象在存储桶中的位置，即称对象键
HeadObjectRequest request = new HeadObjectRequest(bucket, key);
//设置签名有效时长
request.SetSign(TimeUtils.GetCurrentTime(TimeUnit.SECONDS), 600);
//执行请求
cosXml.HeadObject(request,
delegate(COSXML.Model.CosResult cosResult)
{
//请求成功
HeadObjectResult result = cosResult as HeadObjectResult;
Console.WriteLine(result.GetResultInfo());

},
delegate(COSXML.CosException.CosClientException clientEx, COSXML.CosException.CosServerException serverEx)
{
//请求失败
if (clientEx != null)
{
Console.WriteLine("CosClientException: " + clientEx.Message);
}
else if (serverEx != null)
{
Console.WriteLine("CosServerException: " + serverEx.GetInfo());
}
});
*/
```

参数说明

参数名称	设置方法	描述	类型
bucket	构造方法	存储桶名称，格式：BucketName-APPID	string
key	构造方法或 SetCosPath	存储于 COS 上 Object 的对象键	string
signStartTimeSecond	SetSign	签名有效期起始时间（Unix 时间戳），例如1557902800	long
durationSecond	SetSign	签名有效期时长（单位为秒），例如签名有效时期为1分钟：60	long
headerKeys	SetSign	签名是否校验 header	`List`
queryParameterKeys	SetSign	签名是否校验请求 url 中查询参数	`List`

返回结果说明

通过 HeadObjectResult 返回请求结果。

成员变量	类型	描述
httpCode	int	HTTP Code，[200， 300)之间表示操作成功，否则表示操作失败
eTag	string	返回对象的 eTag

> 说明：
>
> 操作失败时，系统将抛出 CosClientException 或 CosServerException（服务端异常）异常。

下载对象

功能说明

下载一个对象至本地。

方法原型

```
GetObjectResult GetObject(GetObjectRequest request);

void GetObject(GetObjectRequest request, COSXML.Callback.OnSuccessCallback<CosResult> successCallback, COSXML.Callback.OnFailedCallback failCallb
ack);

GetObjectBytesResult GetObject(GetObjectBytesRequest request);

void GetObject(GetObjectBytesRequest request, COSXML.Callback.OnSuccessCallback<CosResult> successCallback, COSXML.Callback.OnFailedCallback fai
lCallback);
```

请求示例

```
try
{
string bucket = "examplebucket-1250000000"; //存储桶，格式：BucketName-APPID
string key = "exampleobject"; //对象在存储桶中的位置，即称对象键
string localDir = @"F:\";//下载到本地指定文件夹
string localFileName = "exampleobject"; //指定本地保存的文件名
GetObjectRequest request = new GetObjectRequest(bucket, key, localDir, localFileName);
//设置签名有效时长
request.SetSign(TimeUtils.GetCurrentTime(TimeUnit.SECONDS), 600);
//设置进度回调
request.SetCosProgressCallback(delegate(long completed, long total)
{
Console.WriteLine(String.Format("progress = {0:##.##}%", completed * 100.0 / total));
});
//执行请求
GetObjectResult result = cosXml.GetObject(request);
//请求成功
Console.WriteLine(result.GetResultInfo());
}
catch (COSXML.CosException.CosClientException clientEx)
{
//请求失败
Console.WriteLine("CosClientException: " + clientEx.Message);
}
catch (COSXML.CosException.CosServerException serverEx)
{
//请求失败
Console.WriteLine("CosServerException: " + serverEx.GetInfo());
}

/**
//异步方法
string bucket = "examplebucket-1250000000"; //存储桶，格式：BucketName-APPID
string key = "exampleobject"; //对象在存储桶中的位置，即称对象键
string localDir = @"F:\";//下载到本地指定文件夹
string localFileName = "exampleobject"; //指定本地保存的文件名
GetObjectRequest request = new GetObjectRequest(bucket, key, localDir, localFileName);
//设置签名有效时长
request.SetSign(TimeUtils.GetCurrentTime(TimeUnit.SECONDS), 600);
//设置进度回调
request.SetCosProgressCallback(delegate(long completed, long total)
{
Console.WriteLine(String.Format("progress = {0:##.##}%", completed * 100.0 / total));
});
//执行请求
cosXml.GetObject(request,
delegate(COSXML.Model.CosResult cosResult)
{
//请求成功
GetObjectResult result = cosResult as GetObjectResult;
Console.WriteLine(result.GetResultInfo());

},
delegate(COSXML.CosException.CosClientException clientEx, COSXML.CosException.CosServerException serverEx)
{
//请求失败
if (clientEx != null)
{
```

```
Console.WriteLine("CosClientException: " + clientEx.Message);
}
else if (serverEx != null)
{
Console.WriteLine("CosServerException: " + serverEx.GetInfo());
}
});

//下载返回 bytes 数据
try
{
string bucket = "examplebucket-1250000000"; //存储桶，格式：BucketName-APPID
string key = "exampleobject"; //对象在存储桶中的位置，即称对象键

GetObjectBytesRequest request = new GetObjectBytesRequest(bucket, key);
//设置签名有效时长
request.SetSign(TimeUtils.GetCurrentTime(TimeUnit.SECONDS), 600);
//设置进度回调
request.SetCosProgressCallback(delegate(long completed, long total)
{
Console.WriteLine(String.Format("progress = {0:##.##}%", completed * 100.0 / total));
});
//执行请求
GetObjectBytesResult result = cosXml.GetObject(request);
//获取内容
byte[] content = result.content;
//请求成功
Console.WriteLine(result.GetResultInfo());
}
catch (COSXML.CosException.CosClientException clientEx)
{
//请求失败
Console.WriteLine("CosClientException: " + clientEx.Message);
}
catch (COSXML.CosException.CosServerException serverEx)
{
//请求失败
Console.WriteLine("CosServerException: " + serverEx.GetInfo());
}

//异步方法
string bucket = "examplebucket-1250000000"; //存储桶，格式：BucketName-APPID
string key = "exampleobject"; //对象在存储桶中的位置，即称对象键

GetObjectBytesRequest request = new GetObjectBytesRequest(bucket, key);
//设置签名有效时长
request.SetSign(TimeUtils.GetCurrentTime(TimeUnit.SECONDS), 600);
//设置进度回调
request.SetCosProgressCallback(delegate(long completed, long total)
{
Console.WriteLine(String.Format("progress = {0:##.##}%", completed * 100.0 / total));
});
//执行请求
cosXml.GetObject(request,
delegate(COSXML.Model.CosResult cosResult)
{
//请求成功
GetObjectBytesResult result = cosResult as GetObjectBytesResult;
//获取内容
byte[] content = result.content;
Console.WriteLine(result.GetResultInfo());

},
delegate(COSXML.CosException.CosClientException clientEx, COSXML.CosException.CosServerException serverEx)
{
//请求失败
if (clientEx != null)
{
Console.WriteLine("CosClientException: " + clientEx.Message);
}
else if (serverEx != null)
{
```

```
Console.WriteLine("CosServerException: " + serverEx.GetInfo());
}
});

*/
```

参数说明

参数名称	设置方法	描述	类型
bucket	构造方法	存储桶名称，格式：BucketName-APPID	string
key	构造方法或 SetCosPath	存储于 COS 上 Object 的 对象键	string
localDir	构造方法	下载对象到本地保存的绝对文件夹路径	string
localFileName	构造方法	下载对象到本地保存的文件名	string
progressCallback	SetCosProgressCallback	设置下载进度回调	Callback.OnProgressCallback
signStartTimeSecond	SetSign	签名有效期起始时间（Unix 时间戳），例如1557902800	long
durationSecond	SetSign	签名有效期时长（单位为秒），例如签名有效时期为1分钟：60	long
headerKeys	SetSign	签名是否校验 header	`List`
queryParameterKeys	SetSign	签名是否校验请求 url 中查询参数	`List`

返回结果说明

通过 GetObjectResult 返回请求结果。

成员变量	类型	描述
httpCode	int	HTTP Code，[200， 300)之间表示操作成功，否则表示操作失败
eTag	string	返回对象的 eTag

> **说明：**
>
> 操作失败时，系统将抛出 CosClientException（客户端异常）或 CosServerException（服务端异常）异常。

删除单个对象

功能说明

在存储桶中删除指定对象。

方法原型

```
DeleteObjectResult DeleteObject(DeleteObjectRequest request);

void DeleteObject(DeleteObjectRequest request, COSXML.Callback.OnSuccessCallback<CosResult> successCallback, COSXML.Callback.OnFailedCallback fa
ilCallback);
```

请求示例

```
try
{
string bucket = "examplebucket-1250000000"; //存储桶，格式：BucketName-APPID
string key = "exampleobject"; //对象在存储桶中的位置，即称对象键
DeleteObjectRequest request = new DeleteObjectRequest(bucket, key);
//设置签名有效时长
request.SetSign(TimeUtils.GetCurrentTime(TimeUnit.SECONDS), 600);
//执行请求
DeleteObjectResult result = cosXml.DeleteObject(request);
//请求成功
Console.WriteLine(result.GetResultInfo());
}
catch (COSXML.CosException.CosClientException clientEx)
```

```
{
//请求失败
Console.WriteLine("CosClientException: " + clientEx.Message);
}
catch (COSXML.CosException.CosServerException serverEx)
{
//请求失败
Console.WriteLine("CosServerException: " + serverEx.GetInfo());
}

/**
//异步方法
string bucket = "examplebucket-1250000000"; //存储桶，格式：BucketName-APPID
string key = "exampleobject"; //对象在存储桶中的位置，即称对象键
DeleteObjectRequest request = new DeleteObjectRequest(bucket, key);
//设置签名有效时长
request.SetSign(TimeUtils.GetCurrentTime(TimeUnit.SECONDS), 600);
//执行请求
cosXml.DeleteObject(request,
delegate(COSXML.Model.CosResult cosResult)
{
//请求成功
DeleteObjectResult getObjectResult = result as DeleteObjectResult;
Console.WriteLine(result.GetResultInfo());

},
delegate(COSXML.CosException.CosClientException clientEx, COSXML.CosException.CosServerException serverEx)
{
//请求失败
if (clientEx != null)
{
Console.WriteLine("CosClientException: " + clientEx.Message);
}
else if (serverEx != null)
{
Console.WriteLine("CosServerException: " + serverEx.GetInfo());
}
});
*/
```

参数说明

参数名称	设置方法	描述	类型
bucket	构造方法	存储桶名称，格式：BucketName-APPID	string
key	构造方法或 SetCosPath	存储于 COS 上 Object 的 对象键	string
signStartTimeSecond	SetSign	签名有效期起始时间（Unix 时间戳），例如1557902800	long
durationSecond	SetSign	签名有效期时长（单位为秒），例如签名有效时期为1分钟：60	long
headerKeys	SetSign	签名是否校验 header	`List`
queryParameterKeys	SetSign	签名是否校验请求 url 中查询参数	`List`

返回结果说明

通过 DeleteObjectResult 返回请求结果。

成员变量	类型	描述
httpCode	int	HTTP Code， [200， 300)之间表示操作成功，否则表示操作失败

> **说明：**
>
> 操作失败时，系统将抛出 CosClientException（客户端异常） 或 CosServerException（服务端异常）异常。

删除多个对象

功能说明

在存储桶中批量删除对象。

方法原型

```
DeleteMultiObjectResult DeleteMultiObjects(DeleteMultiObjectRequest request);

void DeleteMultiObjects(DeleteObjectRequest request, COSXML.Callback.OnSuccessCallback<CosResult> successCallback, COSXML.Callback.OnFailedCallback failCallback);
```

请求示例

```
try
{
string bucket = "examplebucket-1250000000"; //存储桶，格式：BucketName-APPID
DeleteMultiObjectRequest request = new DeleteMultiObjectRequest (bucket);
//设置签名有效时长
request.SetSign(TimeUtils.GetCurrentTime(TimeUnit.SECONDS), 600);
//设置返回结果形式
request.SetDeleteQuiet(false);
//对象key
string key = "exampleobject"; //对象在存储桶中的位置，即称对象键
List<string> objects = new List<string>();
objects.Add(key);
request.SetObjectKeys(objects);
//执行请求
DeleteMultiObjectResult result = cosXml.DeleteMultiObjects(request);
//请求成功
Console.WriteLine(result.GetResultInfo());
}
catch (COSXML.CosException.CosClientException clientEx)
{
//请求失败
Console.WriteLine("CosClientException: " + clientEx.Message);
}
catch (COSXML.CosException.CosServerException serverEx)
{
//请求失败
Console.WriteLine("CosServerException: " + serverEx.GetInfo());
}

/**
//异步方法
string bucket = "examplebucket-1250000000"; //存储桶，格式：BucketName-APPID
DeleteMultiObjectRequest request = new DeleteMultiObjectRequest(bucket);
//设置签名有效时长
request.SetSign(TimeUtils.GetCurrentTime(TimeUnit.SECONDS), 600);
//设置返回结果形式
request.SetDeleteQuiet(false);
//对象key
string key = "exampleobject"; //对象在存储桶中的位置，即称对象键
List<string> objects = new List<string>();
objects.Add(key);
request.SetObjectKeys(objects);
//执行请求
cosXml.DeleteMultiObjects(request,
delegate(COSXML.Model.CosResult cosResult)
{
//请求成功
DeleteMultiObjectResult result = cosResult as DeleteMultiObjectResult ;
Console.WriteLine(result.GetResultInfo());

},
delegate(COSXML.CosException.CosClientException clientEx, COSXML.CosException.CosServerException serverEx)
{
//请求失败
if (clientEx != null)
{
Console.WriteLine("CosClientException: " + clientEx.Message);
}
```

```
else if (serverEx != null)
{
Console.WriteLine("CosServerException: " + serverEx.GetInfo());
}
});
*/
```

参数说明

参数名称	设置方法	描述	类型
bucket	构造方法	存储桶名称，格式：BucketName-APPID	string
quiet	SetDeleteQuiet	结果返回模式：false，verbose 模式，true，quiet 模式	bool
keys	SetObjectKeys	删除对象 key 的集合	`List`
signStartTimeSecond	SetSign	签名有效期起始时间（Unix 时间戳），例如1557902800	long
durationSecond	SetSign	签名有效期时长（单位为秒），例如签名有效时期为1分钟：60	long
headerKeys	SetSign	签名是否校验 header	`List`
queryParameterKeys	SetSign	签名是否校验请求 url 中查询参数	`List`

返回结果说明

通过 DeleteMultiObjectResult 返回请求结果。

成员变量	类型	描述
httpCode	int	HTTP Code， [200， 300)之间表示操作成功，否则表示操作失败
deleteResult	DeleteResult	批量删除对象返回的结果

> **说明：**
>
> 操作失败时，系统将抛出 CosClientException（客户端异常） 或 CosServerException（服务端异常）异常。

分块操作

分块上传对象可包括的操作：

- 分块上传对象：初始化分块上传，上传分块，完成所有分块上传。
- 分块续传：查询已上传的分块，上传分块，完成所有分块上传。
- 删除已上传分块。

查询分块上传

功能说明

查询正在进行中的分块上传信息。

方法原型

```
ListMultiUploadsResult ListMultiUploads(ListMultiUploadsRequest request);

void ListMultiUploads(ListMultiUploadsRequest request, COSXML.Callback.OnSuccessCallback<CosResult> successCallback, COSXML.Callback.OnFailedCallback failCallback);
```

请求示例

```
try
{
string bucket = "examplebucket-1250000000"; //格式：BucketName-APPID
ListMultiUploadsRequest request = new ListMultiUploadsRequest(bucket);
//设置签名有效时长
request.SetSign(TimeUtils.GetCurrentTime(TimeUnit.SECONDS), 600);
```

```
//执行请求
ListMultiUploadsResult result = cosXml.ListMultiUploads(request);
//请求成功
Console.WriteLine(result.GetResultInfo());
}
catch (COSXML.CosException.CosClientException clientEx)
{
//请求失败
Console.WriteLine("CosClientException: " + clientEx.Message);
}
catch (COSXML.CosException.CosServerException serverEx)
{
//请求失败
Console.WriteLine("CosServerException: " + serverEx.GetInfo());
}

/**
//异步方法
string bucket = "examplebucket-1250000000"; //格式：BucketName-APPID
ListMultiUploadsRequest request = new ListMultiUploadsRequest(bucket);
//设置签名有效时长
request.SetSign(TimeUtils.GetCurrentTime(TimeUnit.SECONDS), 600);
//执行请求
cosXml.ListMultiUploads(request,
delegate(COSXML.Model.CosResult cosResult)
{
//请求成功
ListMultiUploadsResult result = cosResult as ListMultiUploadsResult;
Console.WriteLine(result.GetResultInfo());

},
delegate(COSXML.CosException.CosClientException clientEx, COSXML.CosException.CosServerException serverEx)
{
//请求失败
if (clientEx != null)
{
Console.WriteLine("CosClientException: " + clientEx.Message);
}
else if (serverEx != null)
{
Console.WriteLine("CosServerException: " + serverEx.GetInfo());
}
});
*/
```

参数说明

参数名称	设置方法	描述	类型
bucket	构造方法	存储桶名称，格式：BucketName-APPID	string
signStartTimeSecond	SetSign	签名有效期起始时间（Unix 时间戳），例如1557902800	long
durationSecond	SetSign	签名有效期时长（单位为秒），例如签名有效时期为1分钟：60	long
headerKeys	SetSign	签名是否校验 header	`List`
queryParameterKeys	SetSign	签名是否校验请求 url 中查询参数	`List`

返回结果说明

通过 ListMultiUploadsResult 返回请求结果。

成员变量	类型	描述
httpCode	int	HTTP Code， [200， 300)之间表示操作成功，否则表示操作失败
listMultipartUploads	ListMultipartUploads	返回 Bucket 中所有正在进行分块上传的信息

> 说明：

> 操作失败时，系统将抛出 CosClientException（客户端异常）或 CosServerException（服务端异常）异常。

初始化分块上传

功能说明

初始化分块上传任务。

方法原型

```
InitMultipartUploadResult InitMultipartUpload(InitMultipartUploadRequest request);

void InitMultipartUpload(InitMultipartUploadRequest request, COSXML.Callback.OnSuccessCallback<CosResult> successCallback, COSXML.Callback.OnFailedCallback failCallback);
```

请求示例

```
try
{
string bucket = "examplebucket-1250000000"; //存储桶，格式：BucketName-APPID
string key = "exampleobject"; //对象在存储桶中的位置，即称对象键
InitMultipartUploadRequest request = new InitMultipartUploadRequest(bucket, key);
//设置签名有效时长
request.SetSign(TimeUtils.GetCurrentTime(TimeUnit.SECONDS), 600);
//执行请求
InitMultipartUploadResult result = cosXml.InitMultipartUpload(request);
//请求成功
string uploadId = result.initMultipartUpload.uploadId; //用于后续分块上传的 uploadId
Console.WriteLine(result.GetResultInfo());
}
catch (COSXML.CosException.CosClientException clientEx)
{
//请求失败
Console.WriteLine("CosClientException: " + clientEx.Message);
}
catch (COSXML.CosException.CosServerException serverEx)
{
//请求失败
Console.WriteLine("CosServerException: " + serverEx.GetInfo());
}

/**
//异步方法
string bucket = "examplebucket-1250000000"; //存储桶，格式：BucketName-APPID
string key = "exampleobject"; //对象在存储桶中的位置，即称对象键
InitMultipartUploadRequest request = new InitMultipartUploadRequest(bucket, key);
//设置签名有效时长
request.SetSign(TimeUtils.GetCurrentTime(TimeUnit.SECONDS), 600);
//执行请求
cosXml.InitMultipartUpload(request,
delegate(COSXML.Model.CosResult cosResult)
{
//请求成功
InitMultipartUploadResult result = cosResult as InitMultipartUploadResult;
string uploadId = result.initMultipartUpload.uploadId; //用于后续分块上传的 uploadId
Console.WriteLine(result.GetResultInfo());

},
delegate(COSXML.CosException.CosClientException clientEx, COSXML.CosException.CosServerException serverEx)
{
//请求失败
if (clientEx != null)
{
Console.WriteLine("CosClientException: " + clientEx.Message);
}
else if (serverEx != null)
{
Console.WriteLine("CosServerException: " + serverEx.GetInfo());
}
```

```
});
*/
```

参数说明

参数名称	设置方法	描述	类型
bucket	构造方法	存储桶名称，格式：BucketName-APPID	string
key	构造方法或 SetCosPath	存储于 COS 上 Object 的 对象键	string
signStartTimeSecond	SetSign	签名有效期起始时间（Unix 时间戳），例如1557902800	long
durationSecond	SetSign	签名有效期时长（单位为秒），例如签名有效时期为1分钟：60	long
headerKeys	SetSign	签名是否校验 header	`List`
queryParameterKeys	SetSign	签名是否校验请求 url 中查询参数	`List`

返回结果说明

通过 InitMultipartUploadResult 返回请求结果。

成员变量	类型	描述
httpCode	int	HTTP Code， [200， 300)之间表示操作成功，否则表示操作失败
initMultipartUpload	InitiateMultipartUpload	返回 对象 初始化分块上传的 uploadId

> **说明：**
>
> 操作失败时，系统将抛出 CosClientException（客户端异常） 或 CosServerException（服务端异常）异常。

查询已上传块

功能说明

查询特定分块上传操作中的已上传的块。

方法原型

```
ListPartsResult ListParts(ListPartsRequest request);

void ListParts(ListPartsRequest request, COSXML.Callback.OnSuccessCallback<CosResult> successCallback, COSXML.Callback.OnFailedCallback failCallbac
k);
```

请求示例

```
try
{
string bucket = "examplebucket-1250000000"; //存储桶，格式：BucketName-APPID
string key = "exampleobject"; //对象在存储桶中的位置，即称对象键
string uploadId ="xxxxxxxx"; //初始化分块上传返回的uploadId
ListPartsRequest request = new ListPartsRequest(bucket, key, uploadId);
//设置签名有效时长
request.SetSign(TimeUtils.GetCurrentTime(TimeUnit.SECONDS), 600);
//执行请求
ListPartsResult result = cosXml.ListParts(request);
//请求成功
//列举已上传的分块
List<COSXML.Model.Tag.ListParts.Part> alreadyUploadParts = result.listParts.parts;
Console.WriteLine(result.GetResultInfo());
}
catch (COSXML.CosException.CosClientException clientEx)
{
//请求失败
Console.WriteLine("CosClientException: " + clientEx.Message);
}
catch (COSXML.CosException.CosServerException serverEx)
{
```

```
//请求失败
Console.WriteLine("CosServerException: " + serverEx.GetInfo());
}

/**
//异步方法
string bucket = "examplebucket-1250000000"; //存储桶，格式：BucketName-APPID
string key = "exampleobject"; //对象在存储桶中的位置，即称对象键
string uploadId ="xxxxxxxx"; //初始化分块上传返回的uploadId
ListPartsRequest request = new ListPartsRequest(bucket, key, uploadId);
//设置签名有效时长
request.SetSign(TimeUtils.GetCurrentTime(TimeUnit.SECONDS), 600);
//执行请求
cosXml.ListParts(request,
delegate(COSXML.Model.CosResult cosResult)
{
//请求成功
ListPartsResult result = cosResult as ListPartsResult;
//列举已上传的分块
List<COSXML.Model.Tag.ListParts.Part> alreadyUploadParts = result.listParts.parts;
Console.WriteLine(result.GetResultInfo());

},
delegate(COSXML.CosException.CosClientException clientEx, COSXML.CosException.CosServerException serverEx)
{
//请求失败
if (clientEx != null)
{
Console.WriteLine("CosClientException: " + clientEx.Message);
}
else if (serverEx != null)
{
Console.WriteLine("CosServerException: " + serverEx.GetInfo());
}
});
*/
```

参数说明

参数名称	设置方法	描述	类型
bucket	构造方法	存储桶名称，格式：BucketName-APPID	string
key	构造方法或 SetCosPath	存储于 COS 上 Object 的 对象键	string
uploadId	构造方法或 SetUploadId	标识指定分块上传的 uploadId	string
signStartTimeSecond	SetSign	签名有效期起始时间（Unix 时间戳），例如1557902800	long
durationSecond	SetSign	签名有效期时长（单位为秒），例如签名有效时期为1分钟：60	long
headerKeys	SetSign	签名是否校验 header	`List`
queryParameterKeys	SetSign	签名是否校验请求 url 中查询参数	`List`

返回结果说明

通过 ListPartsResult 返回请求结果。

成员变量	类型	描述
httpCode	int	HTTP Code， [200， 300)之间表示操作成功，否则表示操作失败
listParts	ListParts	返回指定 uploadId 分块上传中的已上传的块信息

> **说明：**
>
> 操作失败时，系统将抛出 CosClientException（客户端异常） 或 CosServerException（服务端异常）异常。

上传分块

功能说明

分块上传文件。

方法原型

```
UploadPartResult UploadPart(UploadPartRequest request);

void UploadPart(UploadPartRequest, COSXML.Callback.OnSuccessCallback<CosResult> successCallback, COSXML.Callback.OnFailedCallback failCallback);
```

请求示例

```
try
{
string bucket = "examplebucket-1250000000"; //存储桶，格式：BucketName-APPID
string key = "exampleobject"; //对象在存储桶中的位置，即称对象键
string uploadId ="xxxxxxxx"; //初始化分块上传返回的uploadId
int partNumber = 1; //分块编号，必须从1开始递增
string srcPath = @"F:\exampleobject"; //本地文件绝对路径
UploadPartRequest request = new UploadPartRequest(bucket, key, partNumber, uploadId, srcPath);
//设置签名有效时长
request.SetSign(TimeUtils.GetCurrentTime(TimeUnit.SECONDS), 600);
//设置进度回调
request.SetCosProgressCallback(delegate(long completed, long total)
{
Console.WriteLine(String.Format("progress = {0:##.##}%", completed * 100.0 / total));
});
//执行请求
UploadPartResult result = cosXml.UploadPart(request);
//请求成功
//获取返回分块的eTag,用于后续CompleteMultiUploads
string eTag = result.eTag;
Console.WriteLine(result.GetResultInfo());
}
catch (COSXML.CosException.CosClientException clientEx)
{
//请求失败
Console.WriteLine("CosClientException: " + clientEx.Message);
}
catch (COSXML.CosException.CosServerException serverEx)
{
//请求失败
Console.WriteLine("CosServerException: " + serverEx.GetInfo());
}

/**
//异步方法
string bucket = "examplebucket-1250000000"; //存储桶，格式：BucketName-APPID
string key = "exampleobject"; //对象在存储桶中的位置，即称对象键
string uploadId ="xxxxxxxx"; //初始化分块上传返回的uploadId
int partNumber = 1; //分块编号，必须从1开始递增
string srcPath = @"F:\exampleobject"; //本地文件绝对路径
UploadPartRequest request = new UploadPartRequest(bucket, key, partNumber, uploadId, srcPath);
//设置签名有效时长
request.SetSign(TimeUtils.GetCurrentTime(TimeUnit.SECONDS), 600);
//设置进度回调
request.SetCosProgressCallback(delegate(long completed, long total)
{
Console.WriteLine(String.Format("progress = {0:##.##}%", completed * 100.0 / total));
});
//执行请求
cosXml.UploadPart(request,
delegate(COSXML.Model.CosResult cosResult)
{
//请求成功
UploadPartResult result = cosResult as UploadPartResult;
//获取返回分块的eTag,用于后续CompleteMultiUploads
string eTag = result.eTag;
Console.WriteLine(result.GetResultInfo());

},
```

```
delegate(COSXML.CosException.CosClientException clientEx, COSXML.CosException.CosServerException serverEx)
{
//请求失败
if (clientEx != null)
{
Console.WriteLine("CosClientException: " + clientEx.Message);
}
else if (serverEx != null)
{
Console.WriteLine("CosServerException: " + serverEx.GetInfo());
}
});
*/
```

参数说明

参数名称	设置方法	描述	类型
bucket	构造方法	存储桶名称，格式：BucketName-APPID	string
key	构造方法或 SetCosPath	存储于 COS 上 Object 的 对象键	string
uploadId	构造方法或 SetUploadId	标识指定分块上传的 uploadId	string
partNumber	构造方法或 SetPartNumber	标识指定分块的编号，必须 >= 1	int
srcPath	构造方法	用于上传到 COS 的本地文件的绝对路径	string
data	构造方法	用于上传到 COS 的 byte 数组	byte[]
progressCallback	SetCosProgressCallback	设置上传进度回调	Callback.OnProgressCallback
signStartTimeSecond	SetSign	签名有效期起始时间（Unix 时间戳），例如1557902800	long
durationSecond	SetSign	签名有效期时长（单位为秒），例如签名有效时期为1分钟：60	long
headerKeys	SetSign	签名是否校验 header	`List`
queryParameterKeys	SetSign	签名是否校验请求 url 中查询参数	`List`

返回结果说明

通过 UploadPartResult 返回请求结果。

成员变量	类型	描述
httpCode	int	HTTP Code， [200， 300)之间表示操作成功，否则表示操作失败
eTag	string	返回对象的分块的 eTag

> **说明：**
>
> 操作失败时，系统将抛出 CosClientException（客户端异常） 或 CosServerException（服务端异常）异常。

完成分块上传

功能说明

完成整个文件的分块上传。

方法原型

```
CompleteMultipartUploadResult CompleteMultiUpload(CompleteMultipartUploadRequest request);

void CompleteMultiUpload(CompleteMultipartUploadRequest request, COSXML.Callback.OnSuccessCallback<CosResult> successCallback, COSXML.Callback.OnFailedCallback failCallback);
```

请求示例

```
try
{
string bucket = "examplebucket-1250000000"; //存储桶，格式：BucketName-APPID
string key = "exampleobject"; //对象在存储桶中的位置，即称对象键
string uploadId ="xxxxxxxx"; //初始化分块上传返回的uploadId
CompleteMultipartUploadRequest request = new CompleteMultipartUploadRequest(bucket, key, uploadId);
//设置签名有效时长
request.SetSign(TimeUtils.GetCurrentTime(TimeUnit.SECONDS), 600);
//设置已上传的parts,必须有序，按照partNumber递增
request.SetPartNumberAndETag(1, "partNumber1 eTag");
//执行请求
CompleteMultipartUploadResult result = cosXml.CompleteMultiUpload(request);
//请求成功
Console.WriteLine(result.GetResultInfo());
}
catch (COSXML.CosException.CosClientException clientEx)
{
//请求失败
Console.WriteLine("CosClientException: " + clientEx.Message);
}
catch (COSXML.CosException.CosServerException serverEx)
{
//请求失败
Console.WriteLine("CosServerException: " + serverEx.GetInfo());
}

/**
//异步方法
string bucket = "examplebucket-1250000000"; //存储桶，格式：BucketName-APPID
string key = "exampleobject"; //对象在存储桶中的位置，即称对象键
string uploadId ="xxxxxxxx"; //初始化分块上传返回的uploadId
CompleteMultipartUploadRequest request = new CompleteMultipartUploadRequest(bucket, key, uploadId);
//设置签名有效时长
request.SetSign(TimeUtils.GetCurrentTime(TimeUnit.SECONDS), 600);
//设置已上传的parts,必须有序，按照partNumber递增
request.SetPartNumberAndETag(1, "partNumber1 eTag");
//执行请求
cosXml.CompleteMultiUpload(request,
delegate(COSXML.Model.CosResult cosResult)
{
//请求成功
CompleteMultipartUploadResult result = cosResult as CompleteMultipartUploadResult;
Console.WriteLine(result.GetResultInfo());

},
delegate(COSXML.CosException.CosClientException clientEx, COSXML.CosException.CosServerException serverEx)
{
//请求失败
if (clientEx != null)
{
Console.WriteLine("CosClientException: " + clientEx.Message);
}
else if (serverEx != null)
{
Console.WriteLine("CosServerException: " + serverEx.GetInfo());
}
});
*/
```

参数说明

参数名称	设置方法	描述	类型
bucket	构造方法	存储桶名称，格式：BucketName-APPID	string
key	构造方法或 SetCosPath	存储于 COS 上 Object 的 对象键	string
uploadId	构造方法或 SetUploadId	标识指定分块上传的 uploadId	string
partNumber	SetPartNumberAndETag	标识指定分块的编号，必须 >= 1	int

参数名称	设置方法	描述	类型
eTag	SetPartNumberAndETag	标识指定分块的上传返回的 eTag	string
partNumberAndETags	SetPartNumberAndETag	标识分块的编号和上传返回的 eTag	`Dictionary`
signStartTimeSecond	SetSign	签名有效期起始时间（Unix 时间戳），例如1557902800	long
durationSecond	SetSign	签名有效期时长（单位为秒），例如签名有效时期为1分钟：60	long
headerKeys	SetSign	签名是否校验 header	`List`
queryParameterKeys	SetSign	签名是否校验请求 url 中查询参数	`List`

返回结果说明

通过 CompleteMultipartUploadResult 返回请求结果。

成员变量	类型	描述
httpCode	int	HTTP Code， [200， 300)之间表示操作成功，否则表示操作失败
CompleteResult	CompleteMultipartUploadResult	返回所有分块上传成功信息

> **说明：**
>
> 操作失败时，系统将抛出 CosClientException（客户端异常）或 CosServerException（服务端异常）异常。

终止分块上传

功能说明

终止一个分块上传操作并删除已上传的块。

方法原型

```
AbortMultipartUploadResult AbortMultiUpload(AbortMultipartUploadRequest request);

void AbortMultiUpload(AbortMultipartUploadRequest request, COSXML.Callback.OnSuccessCallback<CosResult> successCallback, COSXML.Callback.OnFailedCallback failCallback);
```

请求示例

```
try
{
string bucket = "examplebucket-1250000000"; //存储桶，格式：BucketName-APPID
string key = "exampleobject"; //对象在存储桶中的位置，即称对象键
string uploadId ="xxxxxxxx"; //初始化分块上传返回的uploadId
AbortMultipartUploadRequest request = new AbortMultipartUploadRequest(bucket, key, uploadId);
//设置签名有效时长
request.SetSign(TimeUtils.GetCurrentTime(TimeUnit.SECONDS), 600);
//执行请求
AbortMultipartUploadResult result = cosXml.AbortMultiUpload(request);
//请求成功
Console.WriteLine(result.GetResultInfo());
}
catch (COSXML.CosException.CosClientException clientEx)
{
//请求失败
Console.WriteLine("CosClientException: " + clientEx.Message);
}
catch (COSXML.CosException.CosServerException serverEx)
{
//请求失败
Console.WriteLine("CosServerException: " + serverEx.GetInfo());
}

/**
//异步方法
string bucket = "examplebucket-1250000000"; //存储桶，格式：BucketName-APPID
```

```
string key = "exampleobject"; //对象在存储桶中的位置，即称对象键
string uploadId ="xxxxxxxx"; //初始化分块上传返回的uploadId
AbortMultipartUploadRequest request = new AbortMultipartUploadRequest(bucket, key, uploadId);
//设置签名有效时长
request.SetSign(TimeUtils.GetCurrentTime(TimeUnit.SECONDS), 600);
//执行请求
cosXml.AbortMultiUpload(request,
delegate(COSXML.Model.CosResult cosResult)
{
//请求成功
AbortMultipartUploadResult result = cosResult as AbortMultipartUploadResult;
Console.WriteLine(result.GetResultInfo());

},
delegate(COSXML.CosException.CosClientException clientEx, COSXML.CosException.CosServerException serverEx)
{
//请求失败
if (clientEx != null)
{
Console.WriteLine("CosClientException: " + clientEx.Message);
}
else if (serverEx != null)
{
Console.WriteLine("CosServerException: " + serverEx.GetInfo());
}
});
*/
```

参数说明

参数名称	设置方法	描述	类型
bucket	构造方法	存储桶名称，格式：BucketName-APPID	string
key	构造方法或 SetCosPath	存储于 COS 上 Object 的 对象键	string
uploadId	构造方法或 SetUploadId	标识指定分块上传的 uploadId	string
signStartTimeSecond	SetSign	签名有效期起始时间（Unix 时间戳），例如1557902800	long
durationSecond	SetSign	签名有效期时长（单位为秒），例如签名有效时期为1分钟：60	long
headerKeys	SetSign	签名是否校验 header	`List`
queryParameterKeys	SetSign	签名是否校验请求 url 中查询参数	`List`

返回结果说明

通过 AbortMultipartUploadResult 返回请求结果。

成员变量	类型	描述
httpCode	int	HTTP Code， [200， 300)之间表示操作成功，否则表示操作失败

> **说明：**
>
> 操作失败时，系统将抛出 CosClientException（客户端异常）或 CosServerException（服务端异常）异常。

Go SDK
快速入门

最近更新时间: 2025-01-15 17:01:00

下载与安装

相关资源

- 对象存储 COS 的 XML Go SDK 源码下载地址： XML Go SDK 。
- 示例 Demo 下载地址： COS XML Go SDK 示例。
- 更多信息请参见 COS Go SDK API 文档。

环境依赖

- Golang：用于下载和安装 Go 编译运行环境，请前往 Golang 官网进行下载。

安装 SDK

执行以下命令安装 COS Go SDK：

```
go get -u github.com/tencentyun/cos-go-sdk-v5/
```

开始使用

下面为您介绍如何使用 COS Go SDK 完成一个基础操作，如初始化客户端、创建存储桶、上传对象、查询对象列表、下载对象和删除对象。

初始化

使用 COS 域名生成 COS GO 客户端 Client 结构。

方法原型

```
func NewClient(uri *BaseURL, httpClient *http.Client) *Client
```

请求示例

```
// 将 <BucketName-APPID>、<Region>、<MyDomain> 修改为真实的信息
// 例如：http://imgcache.finance.cloud.tencent.com:80examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com
u, _ := url.Parse("http://imgcache.finance.cloud.tencent.com:80<BucketName-APPID>.cos.<Region>.<MyDomain>")
b := &cos.BaseURL{BucketURL: u}
// 1. 永久密钥
client := cos.NewClient(b, &http.Client{
Transport: &cos.AuthorizationTransport{
SecretID: "COS_SECRETID",
SecretKey: "COS_SECRETKEY",
},
})
// 2. 临时密钥
client := cos.NewClient(b, &http.Client{
Transport: &cos.AuthorizationTransport{
SecretID: "COS_SECRETID",
SecretKey: "COS_SECRETKEY",
SessionToken："COS_SECRETTOKEN",
},
})
```

创建存储桶

```
package main
import (
"context"
"io/ioutil"
```

```
"net/http"
"net/url"
"os"
"time"

"github.com/tencentyun/cos-go-sdk-v5"
)

func main() {
// 将 <BucketName-APPID>、<Region>、<MyDomain> 修改为真实的信息
// 例如：http://imgcache.finance.cloud.tencent.com:80examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com
u, _ := url.Parse("http://imgcache.finance.cloud.tencent.com:80<BucketName-APPID>.cos.<Region>.<MyDomain>")
b := &cos.BaseURL{BucketURL: u}
c := cos.NewClient(b, &http.Client{
Transport: &cos.AuthorizationTransport{
SecretID: "COS_SECRETID",
SecretKey: "COS_SECRETKEY",
},
})

_, err := c.Bucket.Put(context.Background(), nil)
if err != nil {
panic(err)
}
}
```

上传对象

```
package main
import (
"context"
"net/url"
"os"
"strings"
"net/http"

"github.com/tencentyun/cos-go-sdk-v5"
)

func main() {
// 将 <BucketName-APPID>、<Region>、<MyDomain> 修改为真实的信息
// 例如：http://imgcache.finance.cloud.tencent.com:80examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com
u, _ := url.Parse("http://imgcache.finance.cloud.tencent.com:80<BucketName-APPID>.cos.<Region>.<MyDomain>")
b := &cos.BaseURL{BucketURL: u}
c := cos.NewClient(b, &http.Client{
Transport: &cos.AuthorizationTransport{
SecretID: "COS_SECRETID",
SecretKey: "COS_SECRETKEY",
},
})
// 对象键（Key）是对象在存储桶中的唯一标识。
// 例如，在对象的访问域名 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/test/objectPut.go` 中，对象键为 test/objectPut.go
name := "test/objectPut.go"
// 1. Normal put string content
f := strings.NewReader("test")

_, err := c.Object.Put(context.Background(), name, f, nil)
if err != nil {
panic(err)
}
// 2. Put object by local file path
_, err = c.Object.PutFromFile(context.Background(), name, "./test", nil)
if err != nil {
panic(err)
}
}
```

查询对象列表

```
package main

import (
"context"
"fmt"
"os"
"net/url"
"net/http"

"github.com/tencentyun/cos-go-sdk-v5"
)

func main() {
// 将 <BucketName-APPID>、<Region>、<MyDomain> 修改为真实的信息
// 例如：http://imgcache.finance.cloud.tencent.com:80examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com
u, _ := url.Parse("http://imgcache.finance.cloud.tencent.com:80<BucketName-APPID>.cos.<Region>.<MyDomain>")
b := &cos.BaseURL{BucketURL: u}
c := cos.NewClient(b, &http.Client{
Transport: &cos.AuthorizationTransport{
SecretID: "COS_SECRETID",
SecretKey: "COS_SECRETKEY",
},
})

opt := &cos.BucketGetOptions{
Prefix: "test",
MaxKeys: 3,
}
v, _, err := c.Bucket.Get(context.Background(), opt)
if err != nil {
panic(err)
}

for _, c := range v.Contents {
fmt.Printf("%s, %d\n", c.Key, c.Size)
}
}
```

下载对象

```
package main

import (
"context"
"fmt"
"net/url"
"os"
"io/ioutil"
"net/http"

"github.com/tencentyun/cos-go-sdk-v5"
)

func main() {
// 将 <BucketName-APPID>、<Region>、<MyDomain> 修改为真实的信息
// 例如：http://imgcache.finance.cloud.tencent.com:80examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com
u, _ := url.Parse("http://imgcache.finance.cloud.tencent.com:80<BucketName-APPID>.cos.<Region>.<MyDomain>")
b := &cos.BaseURL{BucketURL: u}
c := cos.NewClient(b, &http.Client{
Transport: &cos.AuthorizationTransport{
SecretID: "COS_SECRETID",
SecretKey: "COS_SECRETKEY",
},
})
// 1.Get object content by resp body.
name := "test/hello.txt"
resp, err := c.Object.Get(context.Background(), name, nil)
if err != nil {
panic(err)
}
```

```
bs, _ := ioutil.ReadAll(resp.Body)
resp.Body.Close()
fmt.Printf("%s\n", string(bs))
// 2.Get object to local file path.
_, err = c.Object.GetToFile(context.Background(), name, "example", nil)
if err != nil {
panic(err)
}
}
```

删除对象

```
package main

import (
"context"
"fmt"
"net/url"
"os"
"io/ioutil"
"net/http"

"github.com/tencentyun/cos-go-sdk-v5"
)

func main() {
// 将 <BucketName-APPID>、<Region>、<MyDomain> 修改为真实的信息
// 例如：http://imgcache.finance.cloud.tencent.com:80examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com
u, _ := url.Parse("http://imgcache.finance.cloud.tencent.com:80<BucketName-APPID>.cos.<Region>.<MyDomain>")
b := &cos.BaseURL{BucketURL: u}
c := cos.NewClient(b, &http.Client{
Transport: &cos.AuthorizationTransport{
SecretID: "COS_SECRETID",
SecretKey: "COS_SECRETKEY",
},
})
name := "test_object"
_, err := c.Object.Delete(context.Background(), name)
if err != nil {
panic(err)
}
}
```

接口文档

最近更新时间: 2025-01-15 17:01:00

存储桶操作

简介

本文档提供关于存储桶的基本操作和访问控制列表（ACL）的相关 API 概览以及 SDK 示例代码。

基本操作

API	操作名	操作描述
PUT Bucket	创建存储桶	在指定账号下创建一个存储桶
HEAD Bucket	检索存储桶及其权限	检索存储桶是否存在且是否有权限访问
DELETE Bucket	删除存储桶	删除指定账号下的空存储桶

访问控制列表

API	操作名	操作描述
PUT Bucket acl	设置存储桶 ACL	设置存储桶的 ACL 配置
GET Bucket acl	查询存储桶 ACL	查询存储桶的 ACL 配置

基本操作

创建存储桶

功能说明

在指定账号下创建一个存储桶。

方法原型

```
func (s *BucketService) Put(ctx context.Context, opt *BucketPutOptions) (*Response, error)
```

请求示例

```
opt := &cos.BucketPutOptions{
XCosACL: "public-read",
}
resp, err := client.Bucket.Put(context.Background(), opt)
```

参数说明

```
type BucketPutOptions struct {
XCosACL string
XCosGrantRead string
XCosGrantWrite string
XCosGrantFullControl string
}
```

参数名称	参数描述	类型	必填
XCosACL	设置 Bucket 的 ACL，如 private，public-read，public-read-write	string	否

参数名称	参数描述	类型	必填
XCosGrantFullControl	赋予指定账户对 Bucket 的读写权限。格式为`id=" ",id=" "`。当需要给子账户授权时，格式为`id="qcs::cam::uin/{OwnerUin}:uin/{SubUin}"`，当需要给主账户授权时，格式为`id="qcs::cam::uin/{OwnerUin}:uin/{OwnerUin}"`。例如`id="qcs::cam::uin/100000000001:uin/100000000011",id="qcs::cam::uin/100000000001:uin/100000000001"`	string	否
XCosGrantRead	赋予指定账户对 Bucket 的读权限。格式为`id=" ",id=" "`。当需要给子账户授权时，格式为`id="qcs::cam::uin/{OwnerUin}:uin/{SubUin}"`，当需要给主账户授权时，格式为`id="qcs::cam::uin/{OwnerUin}:uin/{OwnerUin}"`。例如`id="qcs::cam::uin/100000000001:uin/100000000011",id="qcs::cam::uin/100000000001:uin/100000000001"`	string	否
XCosGrantWrite	赋予指定账户对 Bucket 的写权限。格式为`id=" ",id=" "`。当需要给子账户授权时，格式为`id="qcs::cam::uin/{OwnerUin}:uin/{SubUin}"`，当需要给主账户授权时，格式为`id="qcs::cam::uin/{OwnerUin}:uin/{OwnerUin}"`。例如`id="qcs::cam::uin/100000000001:uin/100000000011",id="qcs::cam::uin/100000000001:uin/100000000001"`	string	否

检索存储桶及其权限

功能说明

检索存储桶是否存在且是否有权限访问。

方法原型

```
func (s *BucketService) Head(ctx context.Context) (*Response, error)
```

请求示例

```
resp, err := client.Bucket.Head(context.Background())
```

删除存储桶

功能说明

删除指定账号下的空存储桶。

方法原型

```
func (s *BucketService) Delete(ctx context.Context) (*Response, error)
```

请求示例

```
resp, err := client.Bucket.Delete(context.Background())
```

访问控制列表

设置存储桶 ACL

功能说明

设置指定存储桶访问权限控制列表（PUT Bucket acl）。

方法原型

```
func (s *BucketService) PutACL(ctx context.Context, opt *BucketPutACLOptions) (*Response, error)
```

请求示例

```
// 1. Set Bucket ACL by header.
opt := &cos.BucketPutACLOptions{
Header: &cos.ACLHeaderOptions{
//private，public-read，public-read-write
XCosACL: "private",
},
```

```
}
resp, err := client.Bucket.PutACL(context.Background(), opt)

// 2. Set Bucket ACL by body.
opt = &cos.BucketPutACLOptions{
Body: &cos.ACLXml{
Owner: &cos.Owner{
ID: "qcs::cam::uin/100000760461:uin/100000760461",
},
AccessControlList: []cos.ACLGrant{
{
Grantee: &cos.ACLGrantee{
Type: "RootAccount",
ID:"qcs::cam::uin/100000760461:uin/100000760461",
},
Permission: "FULL_CONTROL",
},
},
},
}
resp, err := client.Bucket.PutACL(context.Background(), opt)
```

参数说明

```
type ACLHeaderOptions struct {
XCosACL string
XCosGrantRead string
XCosGrantWrite string
XCosGrantFullControl string
}
```

参数名称	参数描述	类型	必填
XCosACL	设置 Bucket 的 ACL，如 private，public-read，public-read-write	string	否
XCosGrantFullControl	赋予指定账户对 Bucket 的读写权限。格式为`id=" ",id=" "`。当需要给子账户授权时，格式为`id="qcs::cam::uin/{OwnerUin}:uin/{SubUin}"`，当需要给主账户授权时，格式为`id="qcs::cam::uin/{OwnerUin}:uin/{OwnerUin}"`。例如`id="qcs::cam::uin/100000000001:uin/100000000011",id="qcs::cam::uin/100000000001:uin/100000000001"`	string	否
XCosGrantRead	赋予指定账户对 Bucket 的读权限。格式为`id=" ",id=" "`。当需要给子账户授权时，格式为`id="qcs::cam::uin/{OwnerUin}:uin/{SubUin}"`，当需要给主账户授权时，格式为`id="qcs::cam::uin/{OwnerUin}:uin/{OwnerUin}"`。例如`id="qcs::cam::uin/100000000001:uin/100000000011",id="qcs::cam::uin/100000000001:uin/100000000001"`	string	否
XCosGrantWrite	赋予指定账户对 Bucket 的写权限。格式为`id=" ",id=" "`。当需要给子账户授权时，格式为`id="qcs::cam::uin/{OwnerUin}:uin/{SubUin}"`，当需要给主账户授权时，格式为`id="qcs::cam::uin/{OwnerUin}:uin/{OwnerUin}"`。例如`id="qcs::cam::uin/100000000001:uin/100000000011",id="qcs::cam::uin/100000000001:uin/100000000001"`	string	否
ACLXML	赋予指定账户对 Bucket 的访问权限，具体格式见 GET Bucket acl 返回结果说明	struct	否

查询存储桶 ACL

功能说明

查询指定存储桶的访问权限控制列表（GET Bucket acl）。

方法原型

```
func (s *BucketService) GetACL(ctx context.Context) (*BucketGetACLResult, *Response, error)
```

请求示例

```
v, resp, err := client.Bucket.GetACL(context.Background())
```

返回结果说明

通过 GetBucketACLResult 返回请求结果。

```
type ACLXml struct {
Owner *Owner
AccessControlList []ACLGrant
}
type Owner struct {
ID string
DisplayName string
}
type ACLGrant struct {
Grantee *ACLGrantee
Permission string
}
type ACLGrantee struct {
Type string
ID string
DisplayName string
UIN string
}
```

参数名称	参数描述	类型
Owner	Bucket 拥有者的信息，包括 DisplayName 和 ID	struct
AccessControlList	Bucket 权限授予者的信息，包括 Grantee和 Permission	struct
Grantee	权限授予者的信息，包括 DisplayName，Type，ID 和 UIN	struct
Type	权限授予者的类型，类型为 CanonicalUser 或者 Group	string
ID	Type 为 CanonicalUser 时，对应权限授予者的 ID	string
DisplayName	权限授予者的名字	string
UIN	Type 为 Group 时，对应权限授予者的 UIN	string
Permission	授予者所拥有的 Bucket 的权限，可选值有 FULL_CONTROL，WRITE，READ，分别对应读写权限、写权限、读权限	string

对象操作

简介

本文档提供关于对象的简单操作、分块操作等其他操作相关的 API 概览以及 SDK 示例代码。

简单操作

API	操作名	操作描述
GET Bucket（List Object）	查询对象列表	查询存储桶下的部分或者全部对象
PUT Object	简单上传对象	上传一个 Object（文件/对象）至 Bucket
HEAD Object	查询对象元数据	查询 Object 的 Meta 信息
GET Object	下载对象	下载一个 Object（文件/对象）至本地
PUT Object - Copy	设置对象复制	复制文件到目标路径
DELETE Object	删除单个对象	在 Bucket 中删除指定 Object（文件/对象）
DELETE Multiple Object	删除多个对象	在 Bucket 中批量删除 Object（文件/对象）

分块操作

API	操作名	操作描述

API	操作名	操作描述
List Multipart Uploads	查询分块上传	查询正在进行中的分块上传信息
Initiate Multipart Upload	初始化分块上传	初始化 Multipart Upload 上传操作
Upload Part	上传分块	分块上传文件
List Parts	查询已上传块	查询特定分块上传操作中的已上传的块
Complete Multipart Upload	完成分块上传	完成整个文件的分块上传
Abort Multipart Upload	终止分块上传	终止一个分块上传操作并删除已上传的块

其他操作

API	操作名	操作描述
PUT Object acl	设置对象 ACL	设置 Bucket 中某个 Object（文件/对象）的 ACL
GET Object acl	查询对象 ACL	查询 Object（文件/对象）的 ACL

简单操作

查询对象列表

功能说明

查询存储桶下的部分或者全部对象。

方法原型

```
func (s *BucketService) Get(ctx context.Context, opt *BucketGetOptions) (*BucketGetResult, *Response, error)
```

请求示例

```
opt := &cos.BucketGetOptions{
Prefix: "test",
MaxKeys: 100,
}
v, resp, err := client.Bucket.Get(context.Background(), opt)
```

参数说明

```
type BucketGetOptions struct {
Prefix string
Delimiter string
EncodingType string
Marker string
MaxKeys int
}
```

参数名称	参数描述	类型	必填
Prefix	默认为空，对对象键进行筛选，匹配前缀 prefix 为相同值的 objects	string	否
Delimiter	默认为空，如需模拟文件夹，可设置分隔符`/`	string	否
EncodingType	默认不编码，规定返回值的编码方式，可选值：url	string	否
Marker	默认以 UTF-8 二进制顺序列出条目，标记返回 objects 的 list 的起点位置	string	否
MaxKeys	最多返回的 objects 数量，默认为最大的1000	int	否

返回结果说明

```
type BucketGetResult struct {
Name string
Prefix string
Marker string
NextMarker string
Delimiter string
MaxKeys int
IsTruncated bool
Contents []Object
CommonPrefixes []string
EncodingType string
}
```

参数名称	参数描述	类型
Name	存储桶名称，格式：BucketName-APPID，例如 examplebucket-1250000000	string
Prefix	默认为空，对对象键进行筛选，匹配前缀 prefix 为相同值的 objects	string
Marker	默认以 UTF-8 二进制顺序列出条目，标记返回 objects 的 list 的起点位置	string
NextMarker	当 IsTruncated 为 true 时，标记下一次返回 objects 的 list 的起点位置	string
Delimiter	默认为空，如需模拟文件夹，可设置分隔符`/`	string
MaxKeys	最多返回的 objects 数量，默认为最大的1000	int
IsTruncated	表示返回的 objects 是否被截断	bool
Contents	包含所有 object 元信息的 list，每个 Object 类型包括 ETag，StorageClass，Key，Owner，LastModified，Size 等信息	[]Object
CommonPrefixes	所有以 Prefix 开头，以 Delimiter 结尾的 Key 被归到同一类	[]string
EncodingType	默认不编码，规定返回值的编码方式，可选值：url	string

简单上传对象

功能说明

上传一个 Object（文件/对象）至存储桶（PUT Object）。

方法原型

```
func (s *ObjectService) Put(ctx context.Context, key string, r io.Reader, opt *ObjectPutOptions) (*Response, error)
```

请求示例

```
key := "put_option.go"
f, err := os.Open("./test")
opt := &cos.ObjectPutOptions{
ObjectPutHeaderOptions: &cos.ObjectPutHeaderOptions{
ContentType: "text/html",
},
ACLHeaderOptions: &cos.ACLHeaderOptions{
XCosACL: "private",
},
}
resp, err = client.Object.Put(context.Background(), key, f, opt)


type ObjectPutOptions struct {
*ACLHeaderOptions
*ObjectPutHeaderOptions
}
type ACLHeaderOptions struct {
XCosACL string
XCosGrantRead string
XCosGrantWrite string
XCosGrantFullControl string
}
type ObjectPutHeaderOptions struct {
```

```
CacheControl string
ContentDisposition string
ContentEncoding string
ContentType string
ContentLength int
Expires string
// 自定义的 x-cos-meta-* header
XCosMetaXXX *http.Header
XCosStorageClass string
}
```

参数名称	参数描述	类型	必填
r	上传文件的内容，可以为文件流或字节流，当 r 不是`bytes.Buffer/bytes.Reader/strings.Reader`时，必须指定`opt.ObjectPutHeaderOptions.ContentLength`	io.Reader	是
key	对象键（Key）是对象在存储桶中的唯一标识。例如，在对象的访问域名 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/pic.jpg` 中，对象键为 doc/pic.jpg	string	是
XCosACL	设置文件的 ACL，如 private，public-read，public-read-write	string	否
XCosGrantFullControl	赋予被授权者所有的权限。格式：id="[OwnerUin]"	string	否
XCosGrantRead	赋予被授权者读的权限。格式：id="[OwnerUin]"	string	否
XCosStorageClass	设置文件的存储类型，默认值：STANDARD	string	否
Expires	设置 Content-Expires	string	否
CacheControl	缓存策略，设置 Cache-Control	string	否
ContentType	内容类型，设置 Content-Type	string	否
ContentDisposition	文件名称，设置 Content-Disposition	string	否
ContentEncoding	编码格式，设置 Content-Encoding	string	否
ContentLength	设置传输长度	string	否
XCosMetaXXX	用户自定义的文件元信息， 必须以 x-cos-meta 开头，否则会被忽略	http.Header	否

返回结果说明

```
{
'ETag': 'string',
'x-cos-expiration': 'string'
}
```

参数名称	参数描述	类型
ETag	上传文件的 MD5 值	string
x-cos-expiration	设置生命周期后，返回文件过期规则	string

查询对象元数据

功能说明

查询 Object 的 Meta 信息（HEAD Object）。

方法原型

```
func (s *ObjectService) Head(ctx context.Context, key string, opt *ObjectHeadOptions) (*Response, error)
```

请求示例

```
key := "test/hello.txt"
resp, err := client.Object.Head(context.Background(), key, nil)
```

参数说明

```
type ObjectHeadOptions struct {
IfModifiedSince string
}
```

参数名称	参数描述	类型	必填
key	对象键（Key）是对象在存储桶中的唯一标识。例如，在对象的访问域名 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/pic.jpg` 中，对象键为 doc/pic.jpg	string	是
IfModifiedSince	在指定时间后被修改才返回	string	否

返回结果说明

```
{
'Content-Type': 'application/octet-stream',
'Content-Length': '16807',
'ETag': '"9a4802d5c99dafe1c04da0a8e7e166bf"',
'Last-Modified': 'Wed, 28 Oct 2014 20:30:00 GMT',
'X-Cos-Request-Id': 'NTg3NzQ3ZmVfYmRjMzVfMzE5N182NzczMQ=='
}
```

参数名称	参数描述	类型
文件元信息	获取文件的元信息，包括 Etag 和 X-Cos-Request-Id 等信息，也会包含设置的文件元信息	string

下载对象

功能说明

下载一个 Object（文件/对象）至本地（GET Object）。

方法原型

```
func (s *ObjectService) Get(ctx context.Context, key string, opt *ObjectGetOptions) (*Response, error)

func (s *ObjectService) GetToFile(ctx context.Context, key, localfile string, opt *ObjectGetOptions) (*Response, error)
```

请求示例

```
key := "test/hello.txt"
opt := &cos.ObjectGetOptions{
ResponseContentType: "text/html",
Range: "bytes=0-3",
}
// opt 可选，无特殊设置可设为 nil
// 1. Get object by resp body.
resp, err := client.Object.Get(context.Background(), key, opt)
bs, _ = ioutil.ReadAll(resp.Body)
resp.Body.Close()
fmt.Printf("%s\n", string(bs))

// 2.Get object to local file.
_, err = c.Object.GetToFile(context.Background(), name, "hello_1.txt", nil)
if err != nil {
panic(err)
}
```

参数说明

```
type ObjectGetOptions struct {
ResponseContentType string
ResponseContentLanguage string
ResponseExpires string
ResponseCacheControl string
ResponseContentDisposition string
ResponseContentEncoding string
```

```
Range string
IfModifiedSince string
}
```

参数名称	参数描述	类型	必填
key	对象键（Key）是对象在存储桶中的唯一标识。例如，在对象的访问域名 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/pic.jpg` 中，对象键为 doc/pic.jpg	string	是
localfile	设置响应头部 Content-Type	string	是
ResponseContentType	设置响应头部 Content-Type	string	否
ResponseContentLanguage	设置响应头部 Content-Language	string	否
ResponseExpires	设置响应头部 Content-Expires	string	否
ResponseCacheControl	设置响应头部 Cache-Control	string	否
ResponseContentDisposition	设置响应头部 Content-Disposition	string	否
ResponseContentEncoding	设置响应头部 Content-Encoding	string	否
Range	设置下载文件的范围，格式为 bytes=first-last	string	否
IfModifiedSince	在指定时间后被修改才返回	string	否

返回结果说明

```
{
'Body': '',
'Accept-Ranges': 'bytes',
'Content-Type': 'application/octet-stream',
'Content-Length': '16807',
'Content-Disposition': 'attachment; filename="filename.jpg"',
'Content-Range': 'bytes 0-16086/16087',
'ETag': '"9a4802d5c99dafe1c04da0a8e7e166bf"',
'Last-Modified': 'Wed, 28 Oct 2014 20:30:00 GMT',
'X-Cos-Request-Id': 'NTg3NzQ3ZmVfYmRjMzVfMzE5N182NzczMQ=='
}
```

参数名称	参数描述	类型
Body	下载文件的内容	StreamBody
文件元信息	下载文件的元信息，包括 Etag 和 X-Cos-Request-Id 等信息，也会返回设置的文件元信息	string

设置对象复制

复制文件到目标路径（PUT Object-Copy）。

方法原型

```
func (s *ObjectService) Copy(ctx context.Context, key, sourceURL string, opt *ObjectCopyOptions) (*ObjectCopyResult, *Response, error)
```

请求示例

```
u, _ := url.Parse("http://imgcache.finance.cloud.tencent.com:80examplebucket-12500000000.cos.ap-guangzhou.myqcloud.com")
source := "test/objectMove_src"
soruceURL := fmt.Sprintf("%s/%s", u.Host, source)
dest := "test/objectMove_dest"
//opt := &cos.ObjectCopyOptions{}
r, resp, err := client.Object.Copy(context.Background(), dest, soruceURL, nil)
```

参数说明

```
type ObjectCopyOptions struct {
*ObjectCopyHeaderOptions
*ACLHeaderOptions
```

```
}
type ACLHeaderOptions struct {
XCosACL string
XCosGrantRead string
XCosGrantWrite string
XCosGrantFullControl string
}
type ObjectCopyHeaderOptions struct {
XCosMetadataDirective string
XCosCopySourceIfModifiedSince string
XCosCopySourceIfUnmodifiedSince string
XCosCopySourceIfMatch string
XCosCopySourceIfNoneMatch string
XCosStorageClass string
// 自定义的 x-cos-meta-* header
XCosMetaXXX *http.Header
XCosCopySource string
}
```

参数名称	参数描述	类型	必填
key	对象键（Key）是对象在存储桶中的唯一标识。例如，在对象的访问域名 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/pic.jpg` 中，对象键为 doc/pic.jpg	string	是
sourceURL	描述拷贝源文件的 URL	string	是
XCosACL	设置文件的 ACL，如 private，public-read，public-read-write	string	否
XCosGrantFullControl	赋予被授权者所有的权限。格式：id="[OwnerUin]"	string	否
XCosGrantRead	赋予被授权者读的权限。格式：id="[OwnerUin]"	string	否
XCosMetadataDirective	可选值为 Copy,Replaced，设置为 Copy 时，忽略设置的用户元数据信息直接复制，设置为 Replaced 时，按设置的元信息修改元数据，当目标路径和源路径一样时，必须设置为 Replaced	string	是
XCosCopySourceIfModifiedSince	当 Object 在指定时间后被修改，则执行操作，否则返回412。可与 XCosCopySourceIfNoneMatch 一起使用，与其他条件联合使用返回冲突	string	否
XCosCopySourceIfUnmodifiedSince	当 Object 在指定时间后未被修改，则执行操作，否则返回412。可与 XCosCopySourceIfMatch 一起使用，与其他条件联合使用返回冲突	string	否
XCosCopySourceIfMatch	当 Object 的 Etag 和给定一致时，则执行操作，否则返回412。可与 XCosCopySourceIfUnmodifiedSince 一起使用，与其他条件联合使用返回冲突	string	否
XCosCopySourceIfNoneMatch	当 Object 的 Etag 和给定不一致时，则执行操作，否则返回412。可与 XCosCopySourceIfModifiedSince 一起使用，与其他条件联合使用返回冲突	string	否
XCosStorageClass	设置文件的存储类型，默认值：STANDARD	string	否
XCosMetaXXX	用户自定义的文件元信息	http.Header	否
XCosCopySource	源文件 URL 路径，可以通过 versionid 子资源指定历史版本	string	否

返回结果说明

上传文件的属性：

```
type ObjectCopyResult struct {
ETag string
LastModified string
}
```

参数名称	参数描述	类型
ETag	拷贝文件的 MD5 值	string
LastModified	拷贝文件的最后一次修改时间	string

删除单个对象

功能说明

在 Bucket 中删除指定 Object（文件/对象）。

方法原型

```
func (s *ObjectService) Delete(ctx context.Context, key string) (*Response, error)
```

请求示例

```
key := "test/objectPut.go"
resp, err := client.Object.Delete(context.Background(), name)
```

参数说明

参数名称	参数描述	类型	必填
key	对象键（Key）是对象在存储桶中的唯一标识。例如，在对象的访问域名 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/pic.jpg` 中，对象键为 doc/pic.jpg	string	是

删除多个对象

功能说明

在 Bucket 中删除多个 Object（文件/对象）。单次请求最大支持批量删除1000个 Object。

方法原型

```
func (s *ObjectService) DeleteMulti(ctx context.Context, opt *ObjectDeleteMultiOptions) (*ObjectDeleteMultiResult, *Response, error)
```

请求示例

```
var objects []string
objects = append(objects, []string{"a", "b", "c"}...)
obs := []cos.Object{}
for _, v := range names {
obs = append(obs, cos.Object{Key: v})
}
opt := &cos.ObjectDeleteMultiOptions{
Objects: obs,
// 布尔值，这个值决定了是否启动 Quiet 模式。
// 值为 true 启动 Quiet 模式，值为 false 则启动 Verbose 模式，默认值为 False
// Quiet: true,
}

v, _, err := c.Object.DeleteMulti(ctx, opt)
```

参数说明

```
type ObjectDeleteMultiOptions struct {
Quiet bool
Objects []Object
}
// Object is the meta info of the object
type Object struct {
Key string
// And other params not relate to this api
}
```

参数名称	参数描述	类型	必填
Quiet	布尔值，这个值决定了是否启动 Quiet 模式。对于响应结果，COS 提供 Verbose 和 Quiet 两种模式： Verbose 模式将返回每个 Object 的删除结果，Quiet 模式只返回报错的 Object 信息。值为 true 启动 Quiet 模式，值为 false 则启动 Verbose 模式，默认值为 False	Boolean	否
Objects	说明每个将要删除的目标 Object 信息	Container	是
Key	目标 Object 文件名称	String	是

返回结果说明

上传文件的属性：

```
// ObjectDeleteMultiResult is the result of DeleteMulti
type ObjectDeleteMultiResult struct {
DeletedObjects []Object
Errors []struct {
Key string
Code string
Message string
}
}
```

参数名称	参数描述	类型
DeletedObjects	说明每个将要删除的目标 Object 信息	Container
Errors	说明本次删除的失败 Object 信息	Container
Key	删除失败的 Object 的名称	string
Code	删除失败的错误代码	string
Message	删除失败的错误信息	string

分块操作

查询分片上传

功能说明

查询指定存储桶中正在进行的分块上传信息（List Multipart Uploads）。

方法原型

```
func (s *BucketService) ListMultipartUploads(ctx context.Context, opt *ListMultipartUploadsOptions) (*ListMultipartUploadsResult, *Response, error)
```

请求示例

```
v, resp, err := c.Bucket.ListMultipartUploads(context.Background(), opt)
```

参数说明

```
type ListMultipartUploadsOptions struct {
Delimiter string
EncodingType string
Prefix string
MaxUploads int
KeyMarker string
UploadIDMarker string
}
```

参数名称	参数描述	类型	必填
Delimiter	定界符为一个符号，对 Object 名字包含指定前缀且第一次出现 delimiter 字符之间的 Object 作为一组元素：common prefix。如果没有 prefix，则从路径起点开始	string	否
EncodingType	规定返回值的编码格式，合法值：url	string	否
Prefix	限定返回的 Object key 必须以 Prefix 作为前缀。注意使用 prefix 查询时，返回的 key 中仍会包含 Prefix	string	否
MaxUploads	设置最大返回的 multipart 数量，合法取值从1到1000，默认1000	string	否
KeyMarker	与 upload-id-marker 一起使用当 upload-id-marker 未被指定时，ObjectName 字母顺序大于 key-marker 的条目将被列出当 upload-id-marker 被指定时，ObjectName 字母顺序大于 key-marker 的条目被列出，ObjectName 字母顺序等于 key-marker 同时 UploadID 大于 upload-id-marker 的条目将被列出	string	否

参数名称	参数描述	类型	必填
UploadIDMarker	与 key-marker 一起使用当 key-marker 未被指定时，upload-id-marker 将被忽略当 key-marker 被指定时，ObjectName 字母顺序大于 key-marker 的条目被列出，ObjectName 字母顺序等于 key-marker 同时 UploadID 大于 upload-id-marker 的条目将被列出	string	否

返回结果说明

```
// ListMultipartUploadsResult is the result of ListMultipartUploads
type ListMultipartUploadsResult struct {
Bucket string
EncodingType string
KeyMarker string
UploadIDMarker string
NextKeyMarker string
NextUploadIDMarker string
MaxUploads int
IsTruncated bool
Uploads []struct {
Key string
UploadID string
StorageClass string
Initiator *Initiator
Owner *Owner
Initiated string
}
Prefix string
Delimiter string
CommonPrefixes []string
}
// Use the same struct with the Owner
type Initiator Owner
// Owner defines Bucket/Object's owner
type Owner struct {
ID string
DisplayName string
}
```

参数名称	参数描述	类型
Bucket	分块上传的目标 Bucket，格式为 BucketName，如：examplebucket-1250000000	string
EncodingType	默认不编码，规定返回值的编码方式，可选值：url	string
KeyMarker	列出条目从该 key 值开始	string
UploadIDMarker	列出条目从该 UploadId 值开始	string
NextKeyMarker	假如返回条目被截断，则返回 NextKeyMarker 就是下一个条目的起点	string
NextUploadIDMarker	假如返回条目被截断，则返回 UploadId 就是下一个条目的起点	string
MaxUploads	最多返回的分块的数量，默认为最大的1000	string
IsTruncated	表示返回的分块是否被截断	bool
Uploads	每个 Upload 的信息	Container
Key	Object 的名称	string
UploadID	标示本次分块上传的 ID	string
Key	表示返回的分块是否被截断	bool
StorageClass	用来表示分块的存储级别，默认值：STANDARD	string
Initiator	用来表示本次上传发起者的信息	Container
Owner	用来表示这些分块所有者的信息	Container
Initiated	分块上传的起始时间	string

参数名称	参数描述	类型
Prefix	限定返回的 Objectkey 必须以 Prefix 作为前缀，注意使用 prefix 查询时，返回的 key 中仍会包含 Prefix	struct
Delimiter	定界符为一个符号，对 object 名字包含指定前缀且第一次出现 delimiter 字符之间的 object 作为一组元素：common prefix。如果没有prefix，则从路径起点开始	string
CommonPrefixes	将 prefix 到 delimiter 之间的相同路径归为一类，定义为 Common Prefix	string
ID	用户唯一的 CAM 身份 ID	string
DisplayName	用户身份 ID 的简称（UIN）	string

分片上传对象

分片上传对象可包括的操作：

- 分片上传对象： 初始化分片上传，上传分片块，完成分块上传。
- 删除已上传分片块。

初始化分片上传

功能说明

初始化 Multipart Upload 上传操作，获取对应的 uploadId（Initiate Multipart Upload）。

方法原型

```
func (s *ObjectService) InitiateMultipartUpload(ctx context.Context, name string, opt *InitiateMultipartUploadOptions) (*InitiateMultipartUploadResult, *Response, error)
```

请求示例

```
name := "test_multipart"
//可选opt
v, resp, err := client.Object.InitiateMultipartUpload(context.Background(), name, nil )
```

参数说明

```
type InitiateMultipartUploadOptions struct {
*ACLHeaderOptions
*ObjectPutHeaderOptions
}
type ACLHeaderOptions struct {
XCosACL string
XCosGrantRead string
XCosGrantWrite string
XCosGrantFullControl string
}
type ObjectPutHeaderOptions struct {
CacheControl string
ContentDisposition string
ContentEncoding string
ContentType string
ContentLength int
Expires string
// 自定义的 x-cos-meta-* header
XCosMetaXXX *http.Header
XCosStorageClass string
}
```

参数名称	参数描述	类型	必填
r	上传文件的内容，可以为文件流或字节流，当 r 不是`bytes.Buffer/bytes.Reader/strings.Reader`时，必须指定`opt.ObjectPutHeaderOptions.ContentLength`	io.Reader	是
key	对象键（Key）是对象在存储桶中的唯一标识。例如，在对象的访问域名 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/pic.jpg` 中，对象键为 doc/pic.jpg	string	是

参数名称	参数描述	类型	必填
XCosACL	设置文件的ACL，如 private，public-read	string	否
XCosGrantFullControl	赋予被授权者所有的权限。格式：id="[OwnerUin]"	string	否
XCosGrantRead	赋予被授权者读的权限。格式：id="[OwnerUin]"	string	否
XCosStorageClass	设置文件的存储类型，默认值：STANDARD	string	否
Expires	设置 Content-Expires	string	否
CacheControl	缓存策略，设置 Cache-Control	string	否
ContentType	内容类型，设置 Content-Type	string	否
ContentDisposition	文件名称，设置 Content-Disposition	string	否
ContentEncoding	编码格式，设置 Content-Encoding	string	否
ContentLength	设置传输长度	string	否
XCosMetaXXX	用户自定义的文件元信息， 必须以 x-cos-meta 开头，否则会被忽略	http.Header	否

返回结果说明

```
type InitiateMultipartUploadResult struct {
Bucket string
Key string
UploadID string
}
```

参数名称	参数描述	类型
UploadId	标识分块上传的 ID	string
Bucket	Bucket 名称，由 bucket-appid 组成	string
Key	对象键（Key）是对象在存储桶中的唯一标识。例如，在对象的访问域名 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/pic.jpg ` 中，对象键为 doc/pic.jpg	string

上传分块

分块上传对象（Upload Part）。

方法原型

```
func (s *ObjectService) UploadPart(ctx context.Context, key, uploadID string, partNumber int, r io.Reader, opt *ObjectUploadPartOptions) (*Response, error)
```

请求示例

```
// 注意，上传分块的块数最多10000块
key := "test/test_multi_upload.go"
f := strings.NewReader("test heoo")
// opt可选
_, err := client.Object.UploadPart(
context.Background(), key, uploadID, 1, f, nil,
)
```

参数说明

```
type ObjectUploadPartOptions struct {
ContentLength int
}
```

参数名称	参数描述	类型	必填

参数名称	参数描述	类型	必填
key	对象键（Key）是对象在存储桶中的唯一标识。例如，在对象的访问域名 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/pic.jpg` 中，对象键为 doc/pic.jpg	string	是
UploadId	标识分块上传的 ID，由 InitiateMultipartUpload 生成	string	是
PartNumber	标识上传分块的序号	int	是
r	上传分块的内容，可以为本地文件流或输入流。当 r 不是`bytes.Buffer/bytes.Reader/strings.Reader`时，必须指定 opt.ContentLength	io.Reader	是
ContentLength	设置传输长度	int	否

返回结果说明

```
{
'ETag': 'string'
}
```

参数名称	参数描述	类型
ETag	上传分块的 MD5 值	string

查询已上传块

功能说明

查询特定分块上传操作中的已上传的块（List Parts）。

方法原型

```
func (s *ObjectService) ListParts(ctx context.Context, name, uploadID string, opt *ObjectListPartsOptions) (*ObjectListPartsResult, *Response, error)
```

请求示例

```
key := "test/test_list_parts.go"
v, resp, err := client.Object.ListParts(context.Background(), key, uploadID, nil)
```

参数说明

```
type ObjectListPartsOptions struct {
EncodingType string
MaxParts string
PartNumberMarker string
}
```

参数名称	参数描述	类型	必填
key	对象键（Key）是对象在存储桶中的唯一标识。例如，在对象的访问域名 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/pic.jpg ` 中，对象键为 doc/pic.jpg	string	是
UploadId	标识分块上传的 ID，由 InitiateMultipartUpload 生成	string	是
EncodingType	规定返回值的编码方式	string	否
MaxParts	单次返回最大的条目数量，默认1000	string	否
PartNumberMarker	默认以 UTF-8 二进制顺序列出条目，所有列出条目从 marker 开始	string	否

返回结果说明

```
type ObjectListPartsResult struct {
Bucket string
EncodingType string
Key string
UploadID string
```

```
Initiator *Initiator
Owner *Owner
StorageClass string
PartNumberMarker string
NextPartNumberMarker string
MaxParts string
IsTruncated bool
Parts []Object
}
type Initiator struct {
UIN string
ID string
DisplayName string
}
type Owner struct {
UIN string
ID string
DisplayName string
}
type Object struct {
Key string
ETag string
Size int
PartNumber int
LastModified string
StorageClass string
Owner *Owner
}
```

参数名称	参数描述	类型
Bucket	存储桶名称，格式：BucketName-APPID。例如 examplebucket-1250000000	string
EncodingType	默认不编码，规定返回值的编码方式，可选值：url	string
Key	对象键（Key）是对象在存储桶中的唯一标识。例如，在对象的访问域名 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/pic.jpg` 中，对象键为 doc/pic.jpg	string
UploadId	标识分块上传的 ID，由 InitiateMultipartUpload 生成	string
Initiator	分块上传的创建者，包括 DisplayName，UIN 和 ID	struct
Owner	文件拥有者的信息，包括 DisplayName ，UIN 和 ID	struct
StorageClass	文件的存储类型，默认值：STANDARD	string
PartNumberMarker	默认为0，从第一块列出分块，从 PartNumberMarker 下一个分块开始列出	string
NextPartNumberMarker	指明下一次列出分块的起始位置	string
MaxParts	最多返回的分块的数量，默认为最大的1000	string
IsTruncated	表示返回的分块是否被截断	bool
Part	上传分块的相关信息，包括 ETag，PartNumber，Size，LastModified	struct

完成分块上传

功能说明

完成整个文件的分块上传（Complete Multipart Upload）。

方法原型

```
func (s *ObjectService) CompleteMultipartUpload(ctx context.Context, key, uploadID string, opt *CompleteMultipartUploadOptions) (*CompleteMultipartUploadResult, *Response, error)
```

请求示例

```
// 封装 UploadPart 接口返回 etag 信息
func uploadPart(c *cos.Client, name string, uploadID string, blockSize, n int) string {
```

```
b := make([]byte, blockSize)
if _, err := rand.Read(b); err != nil {
panic(err)
}
s := fmt.Sprintf("%X", b)
f := strings.NewReader(s)

// 当传入参数 f 不是 bytes.Buffer/bytes.Reader/strings.Reader 时，必须指定 opt.ContentLength
// opt := &cos.ObjectUploadPartOptions{
// ContentLength: size,
// }

resp, err := c.Object.UploadPart(
context.Background(), name, uploadID, n, f, nil,
)
if err != nil {
panic(err)
}
return resp.Header.Get("Etag")
}

// Init, UploadPart and Complete process
key := "test/test_complete_upload.go"
v, resp, err := client.Object.InitiateMultipartUpload(context.Background(), key, nil)
uploadID := v.UploadID
blockSize := 1024 * 1024 * 3

opt := &cos.CompleteMultipartUploadOptions{}
for i := 1; i < 5; i++ {
// 调用上面封装的接口获取 etag 信息
etag := uploadPart(c, key, uploadID, blockSize, i)
opt.Parts = append(opt.Parts, cos.Object{
PartNumber: i, ETag: etag},
)
}

v, resp, err = client.Object.CompleteMultipartUpload(
context.Background(), key, uploadID, opt,
)
```

参数说明

```
type CompleteMultipartUploadOptions struct {
Parts []Object
}
type Object struct {
ETag string
PartNumber int
}
```

参数名称	参数描述	类型	必填
key	对象键（Key）是对象在存储桶中的唯一标识。例如，在对象的访问域名 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/pic.jpg` 中，对象键为 doc/pic.jpg	string	是
UploadId	标识分块上传的 ID，由 InitiateMultipartUpload 生成	string	是
CompleteMultipartUploadOptions	所有分块的 ETag 和 PartNumber 信息	struct	是

返回结果说明

```
type CompleteMultipartUploadResult struct {
Location string
Bucket string
Key string
ETag string
}
```

参数名称	参数描述	类型

参数名称	参数描述	类型
Location	URL 地址	string
Bucket	存储桶名称，格式：BucketName-APPID。例如 examplebucket-1250000000	string
Key	对象键（Key）是对象在存储桶中的唯一标识。例如，在对象的访问域名 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/pic.jpg ` 中，对象键为 doc/pic.jpg	string
ETag	合并后对象的唯一标签值，该值不是对象内容的 MD5 校验值，仅能用于检查对象唯一性。如需校验文件内容，可以在上传过程中校验单个分块的 ETag 值	string

终止分块上传

功能说明

终止一个分块上传操作并删除已上传的块（Abort Multipart Upload）。

方法原型

```
func (s *ObjectService) AbortMultipartUpload(ctx context.Context, key, uploadID string) (*Response, error)
```

请求示例

```
key := "test_multipart.txt"
v, _, err := client.Object.InitiateMultipartUpload(context.Background(), key, nil)
// Abort
resp, err := client.Object.AbortMultipartUpload(context.Background(), key, v.UploadID)
```

参数说明

参数名称	参数描述	类型	必填
key	对象键（Key）是对象在存储桶中的唯一标识。例如，在对象的访问域名 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/pic.jpg ` 中，对象键为 doc/pic.jpg	string	是
UploadId	标识分块上传的 ID	string	是

其他操作

设置对象 ACL

功能说明

设置对象访问权限控制列表（PUT Object acl）。

方法原型

```
func (s *ObjectService) PutACL(ctx context.Context, key string, opt *ObjectPutACLOptions) (*Response, error)
```

请求示例

```
// 1.Set by header
opt := &cos.ObjectPutACLOptions{
Header: &cos.ACLHeaderOptions{
XCosACL: "private",
},
}
key := "test/hello.txt"
resp, err := client.Object.PutACL(context.Background(), key, opt)
// 2.Set by body
opt = &cos.ObjectPutACLOptions{
Body: &cos.ACLXml{
Owner: &cos.Owner{
ID: "qcs::cam::uin/100000760461:uin/100000760461",
},
AccessControlList: []cos.ACLGrant{
```

```
{
Grantee: &cos.ACLGrantee{
Type: "RootAccount",
ID: "qcs::cam::uin/100000760461:uin/100000760461",
},

Permission: "FULL_CONTROL",
},
},
},
}

resp, err = client.Object.PutACL(context.Background(), key, opt)
```

参数说明

```
type ACLHeaderOptions struct {
XCosACL string
XCosGrantRead string
XCosGrantWrite string
XCosGrantFullControl string
}
```

参数名称	参数描述	类型	必填
key	对象键（Key）是对象在存储桶中的唯一标识。例如，在对象的访问域名 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/pic.jpg ` 中，对象键为 doc/pic.jpg	string	是
XCosACL	设置 Object 的 ACL，如 private，public-read	string	否
XCosGrantFullControl	赋予被授权者所有的权限。格式：id="[OwnerUin]"	string	否
XCosGrantRead	赋予被授权者读的权限。格式：id="[OwnerUin]"	string	否
ACLXML	赋予指定账户对 Bucket 的访问权限，具体格式见 get object acl 返回结果说明	struct	否

查询对象 ACL

功能说明

查询 Object（文件/对象）的 ACL（GET Object acl）。

方法原型

```
func (s *ObjectService) GetACL(ctx context.Context, key string) (*ObjectGetACLResult, *Response, error)
```

请求示例

```
key := "test/hello.txt"
v, resp, err := client.Object.GetACL(context.Background(), key)
```

参数说明

参数名称	参数描述	类型	必填
key	对象键（Key）是对象在存储桶中的唯一标识。例如，在对象的访问域名 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/pic.jpg` 中，对象键为 doc/pic.jpg	string	是

返回结果说明

```
type ACLXml struct {
Owner *Owner
AccessControlList []ACLGrant
}
type Owner struct {
ID string
DisplayName string
}
```

```
type ACLGrant struct {
Grantee *ACLGrantee
Permission string
}
type ACLGrantee struct {
Type string
ID string
DisplayName string
UIN string
}
```

参数名称	参数描述	类型
Owner	Bucket 拥有者的信息，包括 DisplayName 和 ID	struct
AccessControlList	Bucket 权限授予者的信息，包括 Grantee 和 Permission	struct
Grantee	权限授予者的信息，包括 DisplayName，Type，ID 和 UIN	struct
Type	权限授予者的类型，类型为 CanonicalUser 或者 Group	string
ID	Type 为 CanonicalUser 时，对应权限授予者的 ID	string
DisplayName	权限授予者的名字	string
UIN	Type 为 Group 时，对应权限授予者的 UIN	string
Permission	授予者所拥有的 Bucket 的权限，可选值有 FULL_CONTROL，WRITE，READ，分别对应读写权限、写权限、读权限	string

存储桶管理

简介

本文档提供关于跨域访问、生命周期、版本控制和跨地域复制相关的 API 概览以及 SDK 示例代码。

跨域访问

API	操作名	操作描述
PUT Bucket cors	设置跨域配置	设置存储桶的跨域访问权限
GET Bucket cors	查询跨域配置	查询存储桶的跨域访问配置信息
DELETE Bucket cors	删除跨域配置	删除存储桶的跨域访问配置信息

版本控制

API	操作名	操作描述
PUT Bucket versioning	设置版本控制	设置存储桶的版本控制功能
GET Bucket versioning	查询版本控制	查询存储桶的版本控制信息

跨地域复制

API	操作名	操作描述
PUT Bucket replication	设置跨地域复制	设置存储桶的跨地域复制规则
GET Bucket replication	查询跨地域复制	查询存储桶的跨地域复制规则
DELETE Bucket replication	删除跨地域复制	删除存储桶的跨地域复制规则

跨域访问

设置跨域配置

功能说明

设置指定存储桶的跨域访问配置信息（PUT Bucket cors）。

方法原型

```
func (s *BucketService) PutCORS(ctx context.Context, opt *BucketPutCORSOptions) (*Response, error)
```

请求示例

```
opt := &cos.BucketPutCORSOptions{
Rules: []cos.BucketCORSRule{
{
AllowedOrigins: []string{"http://imgcache.finance.cloud.tencent.com:80www.qq.com"},
AllowedMethods: []string{"PUT", "GET"},
AllowedHeaders: []string{"x-cos-meta-test", "x-cos-xx"},
MaxAgeSeconds: 500,
ExposeHeaders: []string{"x-cos-meta-test1"},
},
{
ID: "1234",
AllowedOrigins: []string{"http://imgcache.finance.cloud.tencent.com:80www.baidu.com", "twitter.com"},
AllowedMethods: []string{"PUT", "GET"},
MaxAgeSeconds: 500,
},
},
}
resp, err := client.Bucket.PutCORS(context.Background(), opt)
```

参数说明

```
type BucketCORSRule struct {
ID string
AllowedMethods []string
AllowedOrigins []string
AllowedHeaders []string
MaxAgeSeconds int
ExposeHeaders []string
}
```

参数名称	参数描述	类型	必填
BucketCORSRule	设置对应的跨域规则，包括 ID，MaxAgeSeconds，AllowedOrigin，AllowedMethod，AllowedHeader，ExposeHeader	struct	是
ID	设置规则的 ID	string	否
AllowedMethods	设置允许的方法，如 GET，PUT，HEAD，POST，DELETE	[]string	是
AllowedOrigins	设置允许的访问来源，如 `"http://imgcache.finance.cloud.tencent.com:80finance.cloud.tencent.com"`，支持通配符 *	[]string	是
AllowedHeaders	设置请求可以使用哪些自定义的 HTTP 请求头部，支持通配符 *	[]string	否
MaxAgeSeconds	设置 OPTIONS 请求得到结果的有效期	int	否
ExposeHeaders	设置浏览器可以接收到的来自服务器端的自定义头部信息	[]string	否

查询跨域配置

功能说明

查询存储桶的跨域访问配置信息（GET Bucket cors）。

方法原型

```
func (s *BucketService) GetCORS(ctx context.Context) (*BucketGetCORSResult, *Response, error)
```

请求示例

```
v, resp, err := client.Bucket.GetCORS(context.Background())
```

返回结果说明

通过 GetBucketCORSResult 返回请求结果。

```
type BucketCORSRule struct {
ID string
AllowedMethods []string
AllowedOrigins []string
AllowedHeaders []string
MaxAgeSeconds int
ExposeHeaders []string
}
```

参数名称	参数描述	类型	必填
BucketCORSRule	设置对应的跨域规则，包括 ID，MaxAgeSeconds，AllowedOrigin，AllowedMethod，AllowedHeader，ExposeHeader	struct	是
ID	设置规则的 ID	string	否
AllowedMethods	设置允许的方法，如 GET，PUT，HEAD，POST，DELETE	[]string	是
AllowedOrigins	设置允许的访问来源，如 `"http://imgcache.finance.cloud.tencent.com:80finance.cloud.tencent.com"`，支持通配符 *	[]string	是
AllowedHeaders	设置请求可以使用哪些自定义的 HTTP 请求头部，支持通配符 *	[]string	否
MaxAgeSeconds	设置 OPTIONS 请求得到结果的有效期	int	否
ExposeHeaders	设置浏览器可以接收到的来自服务器端的自定义头部信息	[]string	否

删除跨域配置

功能说明

删除指定存储桶的跨域访问配置（DELETE Bucket cors）。

方法原型

```
func (s *BucketService) DeleteCORS(ctx context.Context) (*Response, error)
```

请求示例

```
resp, err := client.Bucket.DeleteCORS(context.Background())
```

版本控制

设置版本控制

功能说明

设置指定存储桶的版本控制功能（PUT Bucket versioning）。

方法原型

```
func (s *BucketService) PutVersioning(ctx context.Context, opt *BucketPutVersionOptions) (*Response, error)
```

请求示例

```
opt := &cos.BucketPutVersionOptions{
// Enabled or Suspended, the versioning once opened can not close.
Status: "Enabled",
```

```
}
resp, err := c.Bucket.PutVersioning(context.Background(), opt)
```

参数说明

```
type BucketPutVersionOptions struct {
Status string
}
```

参数名称	描述	类型
BucketPutVersionOptions	版本控制策略	struct
Status	说明版本是否开启，枚举值：Suspended（暂停版本控制）、Enabled（开启版本控制）	string

查询版本控制

功能说明

查询指定存储桶的版本控制信息（GET Bucket versioning）。

方法原型

```
func (s *BucketService) GetVersioning(ctx context.Context) (*BucketGetVersionResult, *Response, error)
```

请求示例

```
v, resp, err := c.Bucket.GetVersioning(context.Background())
```

返回结果说明

```
type BucketGetVersionResult struct {
Status string
}
```

参数名称	描述	类型
BucketGetVersionResult	版本控制策略	struct
Status	说明版本是否开启，枚举值：Suspended（暂停版本控制）、Enabled（开启版本控制）	string

跨地域复制

设置跨地域复制

功能说明

设置指定存储桶的跨地域复制规则（PUT Bucket replication）。

方法原型

```
func (s *BucketService) PutBucketReplication(ctx context.Context, opt *PutBucketReplicationOptions) (*Response, error)
```

请求示例

```
opt := &cos.PutBucketReplicationOptions{
// qcs::cam::uin/[UIN]:uin/[Subaccount]
Role: "qcs::cam::uin/100000000001:uin/100000000001",
Rule: []cos.BucketReplicationRule{
{
ID: "1",
// Enabled or Disabled
Status: "Enabled",
Destination: &cos.ReplicationDestination{
// qcs::cos:[Region]::[Bucketname-Appid]
```

```
Bucket: "qcs::cos:ap-guangzhou::examplebucket-1250000000",
},
},
},
}
resp, err := c.Bucket.PutBucketReplication(context.Background(), opt)
```

参数说明

```
type PutBucketReplicationOptions struct {
Role string
Rule []BucketReplicationRule
}
type BucketReplicationRule struct {
ID string
Status string
Prefix string
Destination *ReplicationDestination
}
type ReplicationDestination struct {
Bucket string
StorageClass string
}
```

参数名称	描述	类型
PutBucketReplicationOptions	跨地域复制规则	struct
Role	发起者身份标示：`qcs::cam::uin/:uin/`	string
Rule	具体配置信息，最多支持1000个，所有策略只能指向一个目标存储桶	struct
ID	用来标注具体 Rule 的名称	string
Status	标识 Rule 是否生效，枚举值：Enabled, Disabled	string
Prefix	前缀匹配策略，不可重叠，重叠返回错误。前缀匹配根目录为空	string
Destination	目标存储桶信息	struct
Bucket	资源标识符：`qcs::cos:[region]::[bucketname-AppId]`	string
StorageClass	存储级别，枚举值：STANDARD，TANDARD_IA。默认值：原存储桶级别	string

查询跨地域复制

功能说明

查询指定存储桶的跨地域复制规则（GET Bucket replication）。

方法原型

```
func (s *BucketService) GetBucketReplication(ctx context.Context) (*GetBucketReplicationResult, *Response, error)
```

请求示例

```
v, resp, err := c.Bucket.GetBucketReplication(context.Background())
```

返回结果说明

```
type GetBucketReplicationResult struct {
Role string
Rule []BucketReplicationRule
}
type BucketReplicationRule struct {
ID string
Status string
Prefix string
Destination *ReplicationDestination
```

```
}
type ReplicationDestination struct {
Bucket string
StorageClass string
}
```

参数名称	描述	类型
GetBucketReplicationResult	跨地域复制规则	struct
Role	发起者身份标示：`qcs::cam::uin/:uin/`	string
Rule	具体配置信息，最多支持1000个，所有策略只能指向一个目标存储桶	struct
ID	用来标注具体 Rule 的名称	string
Status	标识 Rule 是否生效，枚举值：Enabled，isabled	string
Prefix	前缀匹配策略，不可重叠，重叠返回错误。前缀匹配根目录为空	string
Destination	目标存储桶信息	struct
Bucket	资源标识符：`qcs::cos:[region]::[bucketname-AppId]`	string
StorageClass	存储级别，枚举值：STANDARD，ANDARD_IA。默认值：原存储桶级别	string

删除跨地域复制

功能说明

删除指定存储桶的跨地域复制规则（DELETE Bucket replication）。

方法原型

```
func (s *BucketService) DeleteBucketReplication(ctx context.Context) (*Response, error)
```

请求示例

```
resp, err := c.Bucket.DeleteBucketReplication(context.Background())
```

预签名

简介

Go SDK 提供获取请求预签名 URL 接口，详细操作请查看本文示例。

获取请求预签名 URL

```
func (s *ObjectService) GetPresignedURL(ctx context.Context, httpMethod, name, ak, sk string, expired time.Duration, opt interface{}) (*url.URL, error)
```

参数说明

参数名称	类型	描述
httpMethod	string	HTTP 请求方法
name	string	HTTP 请求路径，即对象键
ak	string	SecretId
sk	string	SecretKey

参数名称	类型	描述
expired	time.Duration	签名有效时长
opt	interface{}	扩展项，可填 nil

永久密钥预签名请求示例

上传请求示例

```
name := "test/objectPut.go"
ctx := context.Background()
// NewReader create file content
f := strings.NewReader("test")

// 1.Normal add auth header way to put object
_, err := c.Object.Put(ctx, name, f, nil)
if err != nil {
panic(err)
}
// Get presigned
presignedURL, err := c.Object.PresignedURL(ctx, http.MethodPut, name, ak, sk, time.Hour, nil)
if err != nil {
panic(err)
}
// 2.Put object content by presinged url
data := "test upload with presignedURL"
f = strings.NewReader(data)
req, err := http.NewRequest(http.MethodPut, presignedURL.String(), f)
if err != nil {
panic(err)
}
// Can set request header.
req.Header.Set("Content-Type", "text/html")
_, err = http.DefaultClient.Do(req)
if err != nil {
panic(err)
}
```

下载请求示例

```
name := "test"
ctx := context.Background()
// 1.Normal add auth header way to get object
resp, err := c.Object.Get(ctx, name, nil)
if err != nil {
panic(err)
}
bs, _ := ioutil.ReadAll(resp.Body)
resp.Body.Close()
// Get presigned
presignedURL, err := c.Object.GetPresignedURL(ctx, http.MethodGet, name, ak, sk, time.Hour, nil)
if err != nil {
panic(err)
}
// 2.Get object content by presinged url
resp2, err := http.Get(presignedURL.String())
if err != nil {
panic(err)
}
bs2, _ := ioutil.ReadAll(resp2.Body)
resp2.Body.Close()
fmt.Printf("result2 is : %s\n", string(bs2))
fmt.Printf("%v\n\n", bytes.Compare(bs2, bs) == 0)
```

异常处理

简介

API 返回的 Response 为 Golang HTTP 标准库 Response 类型。用户可通过 err.Error() 获取错误提示，服务端返回的具体信息，请参见 [COS 错误码]。

服务端异常

API 返回的 Response 结构中包含调用结构，如下所示：

成员	描述	类型
X-Cos-Request-Id	Response 中响应头，请求 ID，用于表示一个请求，对于排查问题十分重要	string
StatusCode	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，详情请参见[COS 错误码]	string

Java SDK
快速入门

最近更新时间: 2025-01-15 17:01:00

下载与安装

相关资源

- 对象存储服务的 XML Java SDK 资源下载地址：XML Java SDK。
- 示例 Demo 下载地址： COS XML Java SDK 示例。

环境依赖

- SDK 支持 JDK 1.7、1.8及以上版本。
- JDK 安装方式请参见 Java 安装与配置。
- COS Java SDK 中的常见类所在包分别为：
- 客户端配置相关类在包 com.qcloud.cos.* 下。
- 权限相关类在 com.qcloud.cos.auth.* 子包下。
- 异常相关类在 com.qcloud.cos.exception.* 子包下。
- 请求相关类在 com.qcloud.cos.model.* 子包下。
- 地域相关类在 com.qcloud.cos.region.* 子包下。
- 高级 API 接口在 com.qcloud.cos.transfer.* 子包下。

安装 SDK

用户可以通过 maven 和源码两种方式安装 Java SDK：

- maven 安装 在 maven 工程的 pom.xml 文件中添加相关依赖，内容如下：

```
<dependency>
<groupId>com.qcloud</groupId>
<artifactId>cos_api</artifactId>
<version>5.6.3</version>
</dependency>
```

- 源码安装 从 XML Java SDK 下载源码，通过 maven 导入。比如 eclipse，依次选择 【File】>【Import】>【maven】>【Existing Maven Projects】。

卸载 SDK

通过删除 pom 依赖或源码即可卸载 SDK。

开始使用

下面为您介绍如何使用 COS Java SDK 完成一个基础操作，如初始化客户端、创建存储桶、查询存储桶列表、上传对象、查询对象列表、下载对象和删除对象。

术语解释

名称	描述
APPID	开发者访问 COS 服务时拥有的用户维度唯一资源标识，用以标识资源
SecretId	开发者拥有的项目身份识别 ID，用以身份认证
SecretKey	开发者拥有的项目身份密钥
Bucket	COS 中用于存储数据的容器
Object	COS 中存储的具体文件，是存储的基本实体
Region	域名中的地域信息
Endpoint	Endpoint 由 Region 和域名组成，具体格式为： "."，其中 Domain 为自定义的域名。 在控制台创建 Bucket 时，可以看到对应的访问地址为："."，Bucket 后面的部分即为 Endpoint。

名称	描述
ACL	访问控制列表（Access Control List），是指特定 Bucket 或 Object 的访问控制信息列表
CORS	跨域资源共享（Cross-Origin Resource Sharing），指发起请求的资源所在域不同于该请求所指向资源所在的域的 HTTP 请求
Multipart Uploads	分块上传，COS 服务为上传文件提供的一种分块上传模式

导入类名

COS Java SDK 的包名为 com.qcloud.cos.* ，您可以通过 Eclipse 或者 Intellij 等 IDE 工具，导入程序运行所需要的类。

初始化客户端

在执行任何和 COS 服务相关请求之前，都需要先生成 COSClient 类的对象， COSClient 是调用 COS API 接口的对象。在生成一个 COSClient 实例后可反复使用，且线程安全。最后程序或服务退出时，需要关闭客户端。

在初始化客户端，首先需要通过实现 SelfDefinedEndpointBuilder 类，以便于构造出接下来的请求中所需要的服务器信息，包括 Endpoint、GetServiceEndpoint 等。

```
// 实现 EndpointBuilder 接口中的两个函数
class SelfDefinedEndpointBuilder implements EndpointBuilder {
private String region;
private String domain;

public SelfDefinedEndpointBuilder(String region, String domain) {
super();
// 格式化 Region
this.region = Region.formatRegion(new Region(region));
this.domain = domain;
}

@Override
public String buildGeneralApiEndpoint(String bucketName) {
// 构造 Endpoint
String endpoint = String.format("%s.%s", this.region, this.domain);
// 构造 Bucket 访问域名
return String.format("%s.%s", bucketName, endpoint);
}

@Override
public String buildGetServiceApiEndpoint() {
return String.format("%s.%s", this.region, this.domain);
}
}
```

若您使用永久密钥初始化 COSClient，可以先在访问管理控制台中的 API 密钥管理 页面获取 SecretId、SecretKey，使用永久密钥适用于大部分的应用场景，参考示例如下：

```
// 步骤1: 初始化用户身份信息
String secretId = "COS_SECRETID";
String secretKey = "COS_SECRETKEY";

COSCredentials cred = new BasicCOSCredentials(secretId, secretKey);

// 步骤2: 通过 Region, Domain 以及上一步中实现的类, 来初始化客户端配置
String region = "REGION";
String domain = "DOMAIN.COM";

// 上文中实现的 SelfDefinedEndpointBuilder 类, 填入 region 以及 domain
SelfDefinedEndpointBuilder selfDefinedEndpointBuilder = new SelfDefinedEndpointBuilder(region, domain);
ClientConfig clientConfig = new ClientConfig(new Region(region));
clientConfig.setEndpointBuilder(selfDefinedEndpointBuilder);

// 步骤3: 生成 COS 客户端
COSClient cosClient = new COSClient(cred, clientConfig);
```

您也可以使用临时密钥初始化 COSClient，临时密钥生成和使用可参见 临时密钥生成及使用指引，参考示例如下：

```
// 步骤1: 初始化用户身份信息
String tmpSecretId = "COS_SECRETID";
```

```
String tmpSecretKey = "COS_SECRETKEY";
String sessionToken = "COS_TOKEN";

BasicSessionCredentials cred = new BasicSessionCredentials(tmpSecretId, tmpSecretKey, sessionToken);

// 步骤2: 通过 Region, Domain 以及步骤1中实现的类, 来初始化客户端配置
String region = "REGION";
String domain = "DOMAIN.COM";

// 上文中实现的 SelfDefinedEndpointBuilder 类
SelfDefinedEndpointBuilder selfDefinedEndpointBuilder = new SelfDefinedEndpointBuilder();
ClientConfig clientConfig = new ClientConfig(new Region(region));
clientConfig.setEndpointBuilder(selfDefinedEndpointBuilder);

// 步骤3: 生成 COS 客户端
COSClient cosClient = new COSClient(cred, clientConfig);
```

ClientConfig 类为配置信息类，主要的成员如下：

成员名	设置方法	描述	类型
region	构造函数或 set 方法	存储桶所在的区域	Region
httpProtocol	set 方法	请求所使用的协议，默认使用 HTTP 协议与 COS 交互	HttpProtocol
signExpired	set 方法	请求签名的有效时间，默认为1小时	int
connectionTimeout	set 方法	连接 COS 服务的超时时间，默认为30s	int
socketTimeout	set 方法	客户端读取数据的超时时间，默认为30s	int
httpProxyIp	set 方法	代理服务器的 IP	String
httpProxyPort	set 方法	代理服务器的端口	int

创建存储桶

用户确定存储桶名称后，参考如下示例创建存储桶：

```
//存储桶名称，格式：BucketName-APPID
String bucket = "examplebucket-1250000000";
CreateBucketRequest createBucketRequest = new CreateBucketRequest(bucket);

// 设置 bucket 的权限为 PublicRead(公有读私有写), 其他可选有私有读写, 公有读写
createBucketRequest.setCannedAcl(CannedAccessControlList.PublicRead);
try{
// 通过上一步骤中生成的 COS 客户端发出 createBucket 请求
Bucket bucketResult = cosClient.createBucket(createBucketRequest);
} catch (CosServiceException serverException) {
serverException.printStackTrace();
} catch (CosClientException clientException) {
clientException.printStackTrace();
}
```

查询存储桶列表

查询用户的存储桶列表，参考示例如下：

```
try {
List<Bucket> buckets = cosClient.listBuckets();
System.out.println(buckets);
} catch (CosServiceException serverException) {
serverException.printStackTrace();
} catch (CosClientException clientException) {
clientException.printStackTrace();
}
```

上传对象

将本地文件或者已知长度的输入流内容上传到 COS，适用于20M以下图片类小文件上传，最大支持上传不超过5GB文件。5GB以上的文件必须使用分块上传或高级 API 接口上传。

- 若本地文件大部分在 20M 以上，建议您参考使用高级 API 接口进行上传。
- 若 COS 上已存在同样 Key 的对象，上传时则会覆盖旧的对象。
- 若要创建目录对象，请参见 SDK 如何创建目录。
- 对象键（Key）是对象在存储桶中的唯一标识。例如，在对象的访问域名 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/images/picture.jpg` 中，对象键为 images/picture.jpg，详情请参见 对象键 的说明。

上传不超过5GB的文件，参考示例如下：

```
try {
// 指定要上传的文件
File localFile = new File("exampleobject");
// 指定要上传到的存储桶
String bucketName = "examplebucket-1250000000";
// 指定要上传到 COS 上对象键
String key = "exampleobject";
PutObjectRequest putObjectRequest = new PutObjectRequest(bucketName, key, localFile);
PutObjectResult putObjectResult = cosClient.putObject(putObjectRequest);
} catch (CosServiceException serverException) {
serverException.printStackTrace();
} catch (CosClientException clientException) {
clientException.printStackTrace();
}
```

查询对象列表

查询存储桶中对象列表，参考示例如下：

```
try {
String bucket = "examplebucket-1250000000";
ListObjectsRequest listObjectsRequest = new ListObjectsRequest();
// 设置 bucket 名称
listObjectsRequest.setBucketName(bucket);
// prefix 表示列出的 object 的 key 以 prefix 开始
listObjectsRequest.setPrefix("");
// 设置最大遍历出多少个对象, 一次 listobject 最大支持1000
listObjectsRequest.setMaxKeys(1000);
listObjectsRequest.setDelimiter("/");
ObjectListing objectListing = cosClient.listObjects(listObjectsRequest);
for (COSObjectSummary cosObjectSummary : objectListing.getObjectSummaries()) {
// 对象的路径 key
String key = cosObjectSummary.getKey();
// 对象的 etag
String etag = cosObjectSummary.getETag();
// 对象的长度
long fileSize = cosObjectSummary.getSize();
// 对象的存储类型
String storageClass = cosObjectSummary.getStorageClass();
System.out.println("key:" + key + "; etag:" + etag + "; fileSize:" + fileSize + "; storageClass:" + storageClass);
}
} catch (CosServiceException serverException) {
serverException.printStackTrace();
} catch (CosClientException clientException) {
clientException.printStackTrace();
}
```

下载对象

上传对象后，您可以用同样的 key，调用 GetObject 接口将对象下载到本地，也可以生成预签名链接（下载请指定 method 为 GET，详情请参见 预签名 URL），分享到其他终端来进行下载。但如果您的文件设置了私有读权限，那么请注意预签名链接的有效期。 将文件下载到本地指定路径，参考示例如下：

```
try{
// 指定对象所在的存储桶
String bucketName = "examplebucket-1250000000";
// 指定对象在 COS 上的对象键
String key = "exampleobject";
```

```
// 指定要下载到的本地路径
File downFile = new File("exampleobject");
GetObjectRequest getObjectRequest = new GetObjectRequest(bucketName, key);
ObjectMetadata downObjectMeta = cosClient.getObject(getObjectRequest, downFile);
} catch (CosServiceException serverException) {
serverException.printStackTrace();
} catch (CosClientException clientException) {
clientException.printStackTrace();
}
```

删除对象

删除 COS 上指定路径的对象，代码如下:

```
try {
// 指定对象所在的存储桶
String bucketName = "examplebucket-1250000000";
// 指定对象在 COS 上的对象键
String key = "exampleobject";
cosClient.deleteObject(bucketName, key);
} catch (CosServiceException serverException) {
serverException.printStackTrace();
} catch (CosClientException clientException) {
clientException.printStackTrace();
}
```

关闭客户端

关闭 cosClient，并释放 HTTP 连接的后台管理线程，代码如下。

```
// 关闭客户端(关闭后台线程)
cosClient.shutdown();
```

接口文档

最近更新时间: 2025-01-15 17:01:00

存储桶操作

简介

本文档提供关于存储桶的基本操作和访问控制列表（ACL）的相关 API 概览以及 SDK 示例代码。

基本操作

API	操作名	操作描述
GET Service	查询存储桶列表	查询当前地域下所有的存储桶列表
PUT Bucket	创建存储桶	在指定账号下创建一个存储桶
HEAD Bucket	检索存储桶及其权限	检索存储桶是否存在且是否有权限访问
DELETE Bucket	删除存储桶	删除指定账号下的空存储桶

访问控制列表

API	操作名	操作描述
PUT Bucket acl	设置存储桶 ACL	设置存储桶的 ACL 配置
GET Bucket acl	查询存储桶 ACL	查询存储桶的 ACL 配置

基本操作

查询存储桶列表

功能说明

查询当前地域下所有的存储桶列表。

方法原型

```
public List<Bucket> listBuckets() throws CosClientException, CosServiceException;
```

参数说明

无

返回结果说明

- 成功：返回一个 所有 Bucket 类的列表，Bucket 类包含了 bucket 成员，location 等信息。
- 失败：发生错误（如 Bucket 不存在），抛出异常 CosClientException 或者 CosServiceException。

请求示例

```
List<Bucket> buckets = cosClient.listBuckets();
for (Bucket bucketElement : buckets) {
String bucketName = bucketElement.getName();
String bucketLocation = bucketElement.getLocation();
}
```

创建存储桶

功能说明

在指定账号下创建一个存储桶。同一用户账号下，可以创建多个存储桶，数量上限是200个（不区分地域），存储桶中的对象数量没有限制。创建存储桶是低频操作，一般建议在控制台创建 Bucket，在 SDK 进行 Object 的操作。

方法原型

```
public Bucket createBucket(String bucketName) throws CosClientException, CosServiceException;
```

参数说明

参数名称	描述	类型
bucketName	Bucket 的命名规则为 BucketName-APPID	String

返回结果说明

- 成功： Bucket 类，包含有关 Bucket 的描述（Bucket 的名称，owner 和创建日期）。
- 失败： 发生错误（如身份认证失败），抛出异常 CosClientException 或者 CosServiceException。

请求示例

```
// bucket的命名规则为 BucketName-APPID ，此处填写的存储桶名称必须为此格式
String bucketName = "examplebucket-1250000000";
Bucket bucket = cosClient.createBucket(bucketName);
```

检索存储桶及其权限

功能说明

检索存储桶是否存在且是否有权限访问。

方法原型

```
public boolean doesBucketExist(String bucketName)
throws CosClientException, CosServiceException;
```

参数说明

参数名称	描述	类型
bucketName	Bucket 的命名规则为 BucketName-APPID	String

返回结果说明

- 成功：存在返回 true，否则 false。
- 失败：发生错误（如身份认证失败），抛出异常 CosClientException 或者 CosServiceException。

请求示例

```
// bucket的命名规则为 BucketName-APPID ，此处填写的存储桶名称必须为此格式
String bucketName = "examplebucket-1250000000";
boolean bucketExistFlag = cosClient.doesBucketExist(bucketName);
```

删除存储桶

功能说明

删除指定账号下的空存储桶。

方法原型

```
public void deleteBucket(String bucketName) throws CosClientException, CosServiceException;
```

参数说明

参数名称	描述	类型

参数名称	描述	类型
bucketName	Bucket 的命名规则为 BucketName-APPID	String

返回结果说明

- 成功：无返回值。
- 失败：发生错误（如身份认证失败），抛出异常 CosClientException 或者 CosServiceException。

请求示例

```
// bucket的命名规则为 BucketName-APPID ，此处填写的存储桶名称必须为此格式
String bucketName = "examplebucket-1250000000";
cosClient.deleteBucket(bucketName);
```

访问控制列表

设置存储桶 ACL

功能说明

设置指定存储桶的访问权限控制列表（PUT Bucket acl）。该操作是覆盖操作，会覆盖已有的权限设置。ACL 包括预定义权限策略（CannedAccessControlList）或者自定义的权限控制（AccessControlList）。两类权限当同时设置时将忽略预定义策略，以自定义策略为主。

方法原型

```
// 方法 1 (设置自定义策略)
public void setBucketAcl(String bucketName, AccessControlList acl)
throws CosClientException, CosServiceException;
// 方法 2 (设置预定义策略)
public void setBucketAcl(String bucketName, CannedAccessControlList acl) throws CosClientException, CosServiceException;
// 方法 3 (以上两种方法的封装, 包含两种策略设置，如果同时设置以自定定义策略为主)
public void setBucketAcl(SetBucketAclRequest setBucketAclRequest)
throws CosClientException, CosServiceException;
```

参数说明

方法3参数同时包含1和2，因此以方法3为例进行介绍。

参数名称	描述	类型
setBucketAclRequest	权限设置请求类	SetBucketAclRequest

Request 成员说明：

Request 成员	设置方法	描述	类型
bucketName	构造函数或 set 方法	Bucket 的命名规则为 BucketName-APPID	String
acl	构造函数或 set 方法	自定义权限策略	AccessControlList
cannedAcl	构造函数或 set 方法	预定义策略如公有读、公有读写、私有读	CannedAccessControlList

成员名	描述	类型
List	包含所有要授权的信息	数组
owner	表示 Object 或者 Owner 的拥有者	Owner 类

Grant 类成员说明：

成员名	描述	类型
grantee	被授权人的身份信息	Grantee

成员名	描述	类型
permission	被授权的权限信息（如可读，可写，可读可写）	Permission

Owner 类成员说明：

成员名	描述	类型
id	拥有者的身份信息	String
displayname	拥有者的名字（目前和 id 相同）	String

CannedAccessControlList 表示预设的策略，针对的是所有人。是一个枚举类，枚举值如下所示：

枚举值	描述
Private	私有读写（仅有 owner 可以读写）
PublicRead	公有读私有写（ owner 可以读写， 其他客户可以读）
PublicReadWrite	公有读写（即所有人都可以读写）

返回结果说明

- 成功：无返回值。
- 失败：发生错误（如身份认证失败），抛出异常 CosClientException 或者 CosServiceException。

请求示例

```
// bucket的命名规则为 BucketName-APPID ，此处填写的存储桶名称必须为此格式
String bucketName = "examplebucket-1250000000";
// 设置自定义 ACL
AccessControlList acl = new AccessControlList();
Owner owner = new Owner();
owner.setId("qcs::cam::uin/2779643970:uin/2779643970");
acl.setOwner(owner);
String id = "qcs::cam::uin/2779643970:uin/734505014";
UinGrantee uinGrantee = new UinGrantee("qcs::cam::uin/2779643970:uin/734505014");
uinGrantee.setIdentifier(id);
acl.grantPermission(uinGrantee, Permission.FullControl);
cosClient.setBucketAcl(bucketName, acl);

// 设置预定义 ACL
// 设置私有读写（默认新建的 bucket 都是私有读写）
cosClient.setBucketAcl(bucketName, CannedAccessControlList.Private);
// 设置公有读私有写
cosClient.setBucketAcl(bucketName, CannedAccessControlList.PublicRead);
// 设置公有读写
cosClient.setBucketAcl(bucketName, CannedAccessControlList.PublicReadWrite);
```

查询存储桶 ACL

功能说明

查询指定存储桶的访问权限控制列表。

方法原型

```
public AccessControlList getBucketAcl(String bucketName)
throws CosClientException, CosServiceException
```

参数说明

参数名称	描述	类型
bucketName	Bucket 的命名格式为 BucketName-APPID	String

返回结果说明

- 成功：返回一个 Bucket 的 ACL。
- 失败：发生错误（如身份认证失败），抛出异常 CosClientException 或者 CosServiceException。

请求示例

```
// bucket的命名规则为 BucketName-APPID ，此处填写的存储桶名称必须为此格式
String bucketName = "examplebucket-1250000000";
AccessControlList acl = cosClient.getBucketAcl(bucketName);
```

对象操作

简介

本文档提供关于对象的简单操作、分块操作等其他操作相关的 API 概览以及 SDK 示例代码。

简单操作

API	操作名	操作描述
GET Bucket（List Object）	查询对象列表	查询存储桶下的部分或者全部对象
PUT Object	简单上传对象	上传一个对象至存储桶
HEAD Object	查询对象元数据	查询 Object 的 Meta 信息
GET Object	下载对象	下载一个 Object（文件/对象）至本地
PUT Object - Copy	设置对象复制	复制文件到目标路径
DELETE Object	删除单个对象	在 Bucket 中删除指定 Object（文件/对象）
DELETE Multiple Objects	删除多个对象	在存储桶中批量删除指定对象

分块操作

API	操作名	操作描述
List Multipart Uploads	查询分块上传	查询正在进行中的分块上传信息
Initiate Multipart Upload	初始化分块上传	初始化 Multipart Upload 上传操作
Upload Part	上传分块	分块上传文件
Upload Part - Copy	复制分块	将其他对象复制为一个分块
List Parts	查询已上传块	查询特定分块上传操作中的已上传的块
Complete Multipart Upload	完成分块上传	完成整个文件的分块上传
Abort Multipart Upload	终止分块上传	终止一个分块上传操作并删除已上传的块

其他操作

API	操作名	操作描述
PUT Object acl	设置对象 ACL	设置存储桶中某个对象的访问控制列表
GET Object acl	查询对象 ACL	查询对象的访问控制列表

简单操作

查询对象列表

功能说明

查询存储桶下的部分或者全部对象。

方法原型

```
public ObjectListing listObjects(ListObjectsRequest listObjectsRequest) throws CosClientException, CosServiceException;
```

参数说明

参数名称	描述	类型
listObjectsRequest	获取文件列表请求	ListObjectsRequest

Request 成员说明 ：

Request 成员	设置方法	描述	类型
bucketName	构造函数或 set 方法	Bucket 的命名格式为 BucketName-APPID	String
prefix	构造函数或 set 方法	限制返回的结果对象，以 prefix 为前缀。默认不进行限制，即 Bucket 下所有的成员 默认值为空： ""	String
marker	构造函数或 set 方法	标记 list 的起点位置，第一次可设置为空，后续请求需设置为上一次 listObjects 返回值中的 nextMarker	String
delimiter	构造函数或 set 方法	分隔符，限制返回的是以 prefix 开头，并以 delimiter 第一次出现的结束的路径	String
maxKeys	构造函数或 set 方法	最大返回的成员个数（不得超过 1000） 默认值： 1000	Integer

返回结果说明

- 成功：返回 ObjectListing 类型， 包含所有的成员， 以及 nextMarker。
- 失败：抛出异常 CosClientException 或者 CosServiceException。

请求示例

```
// Bucket的命名格式为 BucketName-APPID ，此处填写的存储桶名称必须为此格式
String bucketName = "examplebucket-1250000000";
ListObjectsRequest listObjectsRequest = new ListObjectsRequest();
// 设置bucket名称
listObjectsRequest.setBucketName(bucketName);
// prefix表示列出的object的key以prefix开始
listObjectsRequest.setPrefix("images/");
// deliter表示分隔符, 设置为/表示列出当前目录下的object, 设置为空表示列出所有的object
listObjectsRequest.setDelimiter("/");
// 设置最大遍历出多少个对象, 一次listobject最大支持1000
listObjectsRequest.setMaxKeys(1000);
ObjectListing objectListing = null;
do {
try {
objectListing = cosclient.listObjects(listObjectsRequest);
} catch (CosServiceException e) {
e.printStackTrace();
return;
} catch (CosClientException e) {
e.printStackTrace();
return;
}
// common prefix表示表示被delimiter截断的路径, 如delimter设置为/, common prefix则表示所有子目录的路径
List<String> commonPrefixs = objectListing.getCommonPrefixes();
```

```
// object summary表示所有列出的object列表
List<COSObjectSummary> cosObjectSummaries = objectListing.getObjectSummaries();
for (COSObjectSummary cosObjectSummary : cosObjectSummaries) {
// 文件的路径key
String key = cosObjectSummary.getKey();
// 文件的etag
String etag = cosObjectSummary.getETag();
// 文件的长度
long fileSize = cosObjectSummary.getSize();
// 文件的存储类型
String storageClasses = cosObjectSummary.getStorageClass();
}

String nextMarker = objectListing.getNextMarker();
listObjectsRequest.setMarker(nextMarker);
} while (objectListing.isTruncated());
```

简单上传对象

功能说明

上传对象到指定的存储桶中（Put Object）。将本地文件或者已知长度的输入流内容上传到 COS。适用于图片类小文件上传（20MB以下），最大支持5GB（含），5GB以上请使用分块上传或高级 API 上传。

- 上传过程中默认会对文件长度与 MD5 进行校验（关闭 MD5 校验参见示例代码）。
- 若 COS 上已存在同样 Key 的对象，上传时则会进行覆盖。
- 当前访问策略条目限制为1000条，如果您不需要进行对象 ACL 控制，上传时请不要设置，默认继承 Bucket 权限。
- 上传之后，您可以用同样的 key，调用 GetObject 接口将文件下载到本地，也可以生成预签名链接（下载请指定 method 为 GET，具体接口说明见下文），发送到其他端来进行下载。

方法原型

```
// 方法1 将本地文件上传到 COS
public PutObjectResult putObject(String bucketName, String key, File file)
throws CosClientException, CosServiceException;
// 方法2 输入流上传到 COS
public PutObjectResult putObject(String bucketName, String key, InputStream input, ObjectMetadata metadata)
throws CosClientException, CosServiceException;
// 方法3 对以上两个方法的包装, 支持更细粒度的参数控制, 如 content-type, content-disposition 等
public PutObjectResult putObject(PutObjectRequest putObjectRequest)
throws CosClientException, CosServiceException;
```

参数说明

参数名称	描述	类型
putObjectRequest	上传文件请求	PutObjectRequest

Request 成员说明：

Request 成员	设置方法	描述	类型
bucketName	构造函数或 set 方法	Bucket 的命名格式为 BucketName-APPID	String
key	构造函数或 set 方法	对象键（Key）是对象在存储桶中的唯一标识。 例如，在对象的访问域名`examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/picture.jpg`中，对象键为 doc/picture.jpg，详情请参见 对象键	String
file	构造函数或 set 方法	本地文件	File
input	构造函数或 set 方法	输入流	InputStream
metadata	构造函数或 set 方法	文件的元数据	ObjectMetadata

ObjectMetadata 类用于记录对象的元信息，其主要成员说明如下：

成员名称	描述	类型
httpExpiresDate	缓存的超时时间，为 HTTP 响应头部中 Expires 字段的值	Date
ongoingRestore	正在从归档存储类型恢复该对象	Boolean
userMetadata	前缀为 x-cos-meta- 的用户自定义元信息	Map
metadata	除用户自定义元信息以外的其他头部	Map

返回结果说明

- 成功：PutObjectResult，包含文件的 ETag 等信息。
- 失败：发生错误（如身份认证失败），抛出异常 CosClientException 或者 CosServiceException。

请求示例

```
// Bucket的命名格式为 BucketName-APPID ，此处填写的存储桶名称必须为此格式
String bucketName = "examplebucket-1250000000";
// 方法1 本地文件上传
File localFile = new File("exampleobject");
String key = "exampleobject";
PutObjectResult putObjectResult = cosClient.putObject(bucketName, key, localFile);
String etag = putObjectResult.getETag(); // 获取文件的 etag

// 方法2 从输入流上传(需提前告知输入流的长度, 否则可能导致 oom)
FileInputStream fileInputStream = new FileInputStream(localFile);
ObjectMetadata objectMetadata = new ObjectMetadata();
// 设置输入流长度为500
objectMetadata.setContentLength(500);
// 设置 Content type, 默认是 application/octet-stream
objectMetadata.setContentType("application/pdf");
PutObjectResult putObjectResult = cosClient.putObject(bucketName, key, fileInputStream, objectMetadata);
String etag = putObjectResult.getETag();
// 关闭输入流...

// 方法3 提供更多细粒度的控制, 常用的设置如下
// 1 content-type, 对于本地文件上传, 默认根据本地文件的后缀进行映射，如 jpg 文件映射 为image/jpeg
// 对于流式上传 默认是 application/octet-stream
// 2 上传的同时指定权限(也可通过调用 API set object acl 来设置)
// 3 若要全局关闭上传MD5校验, 则设置系统环境变量，此设置会对所有的会影响所有的上传校验。 默认是进行校验的。
// 关闭MD5校验： System.setProperty(SkipMd5CheckStrategy.DISABLE_PUT_OBJECT_MD5_VALIDATION_PROPERTY, "true");
// 再次打开校验 System.setProperty(SkipMd5CheckStrategy.DISABLE_PUT_OBJECT_MD5_VALIDATION_PROPERTY, null);
File localFile = new File("picture.jpg");
String key = "picture.jpg";
PutObjectRequest putObjectRequest = new PutObjectRequest(bucketName, key, localFile);
// 设置自定义属性(如 content-type, content-disposition 等)
ObjectMetadata objectMetadata = new ObjectMetadata();
// 设置 Content type, 默认是 application/octet-stream
objectMetadata.setContentType("image/jpeg");
putObjectRequest.setMetadata(objectMetadata);
PutObjectResult putObjectResult = cosClient.putObject(putObjectRequest);
String etag = putObjectResult.getETag(); // 获取文件的 etag
```

查询对象元数据

功能说明

查询存储桶中是否存在指定的对象。

方法原型

```
public ObjectMetadata getObjectMetadata(String bucketName, String key)
throws CosClientException, CosServiceException;
```

参数说明

参数名称	描述	类型
bucketName	Bucket 的命名格式为 BucketName-APPID	String
key	对象键（Key）是对象在存储桶中的唯一标识。例如，在对象的访问域名 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/picture.jpg` 中，对象键为 doc/picture.jpg，详情请参见 对象键	String

返回结果说明

- 成功：无返回值。
- 失败：发生错误（如身份认证失败），抛出异常 CosClientException 或者 CosServiceException。

请求示例

```
// Bucket的命名格式为 BucketName-APPID ，此处填写的存储桶名称必须为此格式
String bucketName = "examplebucket-1250000000";
String key = "exampleobject";
ObjectMetadata objectMetadata = cosClient.getObjectMetadata(bucketName, key);
```

下载对象

功能说明

下载对象到本地（Get Object）。

方法原型

```
// 方法1 下载文件，并获取输入流
public COSObject getObject(GetObjectRequest getObjectRequest)
throws CosClientException, CosServiceException;
// 方法2 下载文件到本地.
public ObjectMetadata getObject(GetObjectRequest getObjectRequest, File destinationFile)
throws CosClientException, CosServiceException;
```

参数说明

参数名称	描述	类型
getObjectRequest	下载文件请求	GetObjectRequest
destinationFile	本地的保存文件	File

Request 成员说明：

Request 成员	设置方法	描述	类型
bucketName	构造函数或 set 方法	Bucket 的命名格式为 BucketName-APPID	String
key	构造函数或 set 方法	对象键（Key）是对象在存储桶中的唯一标识。例如，在对象的访问域名 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/picture.jpg` 中，对象键为 doc/picture.jpg, 详情请参见 对象键	String
range	set方法	下载的 range 范围	Long[]

返回结果说明

- **方法1（获取下载输入流）**
- 成功：返回 COSObject 类，包含输入流以及对象属性。
- 失败：发生错误（如身份认证失败），抛出异常 CosClientException 或者 CosServiceException。
- **方法2（下载文件到本地）**
- 成功：返回文件的属性 ObjectMetadata，包含文件的自定义头和 content-type 等属性。
- 失败：发生错误（如身份认证失败），抛出异常 CosClientException 或者 CosServiceException。

请求示例

```
// Bucket的命名格式为 BucketName-APPID ，此处填写的存储桶名称必须为此格式
String bucketName = "examplebucket-1250000000";
String key = "exampleobject";
// 方法1 获取下载输入流
GetObjectRequest getObjectRequest = new GetObjectRequest(bucketName, key);
COSObject cosObject = cosClient.getObject(getObjectRequest);
COSObjectInputStream cosObjectInput = cosObject.getObjectContent();

// 方法2 下载文件到本地
File downFile = new File("output/exampleobject");
GetObjectRequest getObjectRequest = new GetObjectRequest(bucketName, key);
ObjectMetadata downObjectMeta = cosClient.getObject(getObjectRequest, downFile);
```

设置对象复制

功能说明

将一个对象复制到另一个对象（Put Object Copy）。支持跨地域跨账号跨 Bucket 拷贝，需要拥有对源文件的读取权限以及目的文件的写入权限。最大支持5G文件 copy，5G以上文件请使用高级 API copy。

方法原型

```
public CopyObjectResult copyObject(CopyObjectRequest copyObjectRequest)
throws CosClientException, CosServiceException
```

参数说明

参数名称	描述	类型
copyObjectRequest	拷贝文件请求	CopyObjectRequest

Request 成员说明：

参数名称	描述	类型
sourceBucketRegion	源 Bucket region。默认值：与当前 clientConfig 的 region 一致，表示同地域拷贝	String
sourceBucketName	源存储桶名称，命名格式为 BucketName-APPID	String
sourceKey	源对象键，对象键（Key）是对象在存储桶中的唯一标识。例如，在对象的访问域名`examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/picture.jpg`中，对象键为 doc/picture.jpg，详情请参见 对象键	String
sourceVersionId	源文件 version id（适用于开启了版本控制的源 Bucket）。默认值：源文件当前最新版本	String
destinationBucketName	目标存储桶名称, Bucket 的命名格式为 BucketName-APPID ，name由字母数字和中划线构成	String
destinationKey	目的对象键, 对象键（Key）是对象在存储桶中的唯一标识。例如，在对象的访问域名`examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/picture.jpg`中，对象键为 doc/picture.jpg，详情请参见 对象键	String
storageClass	拷贝的目的文件的存储类型。默认值：Standard	String

返回结果说明

- 成功：返回 CopyObjectResult，包含新文件的 Etag 等信息。
- 失败：发生错误（如身份认证失败），抛出异常 CosClientException 或者 CosServiceException。

请求示例

```
// 同地域同账号拷贝
// 源 Bucket, Bucket的命名格式为 BucketName-APPID ，此处填写的存储桶名称必须为此格式
String srcBucketName = "srcBucket-1250000000";
// 要拷贝的源文件
String srcKey = "srcobject";
// 目标存储桶, Bucket的命名格式为 BucketName-APPID ，此处填写的存储桶名称必须为此格式
String destBucketName = "examplebucket-1250000000";
// 要拷贝的目的文件
String destKey = "exampleobject";
CopyObjectRequest copyObjectRequest = new CopyObjectRequest(srcBucketName, srcKey, destBucketName, destKey);
```

```
CopyObjectResult copyObjectResult = cosClient.copyObject(copyObjectRequest);

// 跨账号跨地域拷贝（需要拥有对源文件的读取权限以及目的文件的写入权限）
String srcBucketNameOfDiffAppid = "srcBucket-125100000";
Region srcBucketRegion = new Region("ap-shanghai");
copyObjectRequest = new CopyObjectRequest(srcBucketRegion, srcBucketNameOfDiffAppid, srcKey, destBucketName, destKey);
copyObjectResult = cosClient.copyObject(copyObjectRequest);
```

删除单个对象

功能说明

删除指定的对象（Delete Object）。

方法原型

```
public void deleteObject(String bucketName, String key)
throws CosClientException, CosServiceException;
```

参数说明

参数名称	描述	类型
bucketName	Bucket 的命名格式为 BucketName-APPID	String
key	对象键（Key）是对象在存储桶中的唯一标识。例如，在对象的访问域名 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/picture.jpg` 中，对象键为 doc/picture.jpg，详情请参见 对象键	String

返回结果说明

- 成功：无返回值。
- 失败：发生错误（如身份认证失败），抛出异常 CosClientException 或者 CosServiceException。

请求示例

```
// Bucket的命名格式为 BucketName-APPID ，此处填写的存储桶名称必须为此格式
String bucketName = "examplebucket-1250000000";
String key = "exampleobject";
cosClient.deleteObject(bucketName, key);
```

删除多个对象

功能说明

删除多个指定的对象（DELETE Multiple Objects）。

方法原型

```
public DeleteObjectsResult deleteObjects(DeleteObjectsRequest deleteObjectsRequest)
throws MultiObjectDeleteException, CosClientException, CosServiceException;
```

参数说明

参数名称	描述	类型
deleteObjectsRequest	请求	DeleteObjectsRequest

Request 成员说明：

参数名称	描述	类型
bucketName	Bucket 的命名格式为 BucketName-APPID	String
quiet	指明删除的返回结果方式，可选值为 true，false，默认为 false。设置为 true 只返回失败的错误信息，设置为 false 时返回成功和失败的所有信息	boolean
keys	对象路径列表，对象的版本号为可选	`List`

KeyVersion 成员说明：

参数名称	描述	类型
key	对象键（Key）是对象在存储桶中的唯一标识。例如，在对象的访问域名 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/picture.jpg` 中，对象键为 doc/picture.jpg，详情请参见 对象键	String
version	在开启存储桶多版本时，指定删除被对象的版本号，可选	String

返回结果说明

- 成功：无返回值。
- 失败：发生错误（如身份认证失败），抛出异常 CosClientException 或者 CosServiceException，

请求示例

```
// Bucket的命名格式为 BucketName-APPID ，此处填写的存储桶名称必须为此格式
String bucketName = "examplebucket-1250000000";

DeleteObjectsRequest deleteObjectsRequest = new DeleteObjectsRequest(bucketName);
// 设置要删除的key列表, 最多一次删除1000个
ArrayList<KeyVersion> keyList = new ArrayList<>();
// 传入要删除的文件名
keyList.add(new KeyVersion("project/folder1/picture.jpg"));
keyList.add(new KeyVersion("project/folder2/text.txt"));
keyList.add(new KeyVersion("project/folder2/music.mp3"));
deleteObjectsRequest.setKeys(keyList);

// 批量删除文件
try {
DeleteObjectsResult deleteObjectsResult = cosclient.deleteObjects(deleteObjectsRequest);
List<DeletedObject> deleteObjectResultArray = deleteObjectsResult.getDeletedObjects();
} catch (MultiObjectDeleteException mde) { // 如果部分删除成功部分失败, 返回MultiObjectDeleteException
List<DeletedObject> deleteObjects = mde.getDeletedObjects();
List<DeleteError> deleteErrors = mde.getErrors();
} catch (CosServiceException e) { // 如果是其他错误，例如参数错误， 身份验证不过等会抛出 CosServiceException
e.printStackTrace();
} catch (CosClientException e) { // 如果是客户端错误，例如连接不上COS
e.printStackTrace();
}
```

分块操作

查询分块上传

功能说明

查询指定存储桶中正在进行的分块上传（List Multipart Uploads）。

方法原型

```
public MultipartUploadListing listMultipartUploads(
ListMultipartUploadsRequest listMultipartUploadsRequest)
throws CosClientException, CosServiceException
```

参数说明

参数名称	描述	类型
listMultipartUploadsRequest	请求	ListMultipartUploadsRequest

Request 成员说明：

参数名称	描述	类型
bucketName	Bucket 的命名格式为 BucketName-APPID	String

参数名称	描述	类型
keyMarker	列出条目从该 Key 值开始	String
delimiter	定界符为一个符号，如果有 Prefix，则将 Prefix 到 delimiter 之间的相同路径归为一类，定义为 Common Prefix，然后列出所有 Common Prefix。如果没有 Prefix，则从路径起点开始	String
prefix	限定返回的 Object key 必须以 Prefix 作为前缀。注意使用 prefix 查询时，返回的 key 中仍会包含 Prefix	String
uploadIdMarker	列出条目从该 UploadId 值开始	String
maxUploads	设置最大返回的 multipart 数量，合法值1到1000	String
encodingType	规定返回值的编码方式，可选值：url	String

返回结果说明

- 成功：返回 MultipartUploadListing，包含正在进行分块上传的信息。
- 失败：发生错误（如身份认证失败），抛出异常 CosClientException 或者 CosServiceException。

请求示例

```
// Bucket的命名格式为 BucketName-APPID ，此处填写的存储桶名称必须为此格式
String bucketName = "examplebucket-1250000000";
ListMultipartUploadsRequest listMultipartUploadsRequest = new ListMultipartUploadsRequest(bucketName);
listMultipartUploadsRequest.setDelimiter("/");
listMultipartUploadsRequest.setMaxUploads(100);
listMultipartUploadsRequest.setPrefix("");
listMultipartUploadsRequest.setEncodingType("url");
MultipartUploadListing multipartUploadListing = cosClient.listMultipartUploads(listMultipartUploadsRequest);
```

分块上传对象

分块上传对象可包括的操作：

- 分块上传对象：初始化分块上传，上传分块，完成分块上传。
- 分块续传：查询已上传块，上传分块，完成分块上传。
- 终止分块上传。

初始化分块上传

功能说明

初始化分块上传任务（Initiate Multipart Upload）。

方法原型

```
public InitiateMultipartUploadResult initiateMultipartUpload(
InitiateMultipartUploadRequest request) throws CosClientException, CosServiceException;
```

参数说明

参数名称	描述	类型
initiateMultipartUploadRequest	请求	InitiateMultipartUploadRequest

Request 成员说明：

参数名称	设置方法	描述	类型
bucketName	构造函数或 set 方法	Bucket 的命名格式为 BucketName-APPID	String
key	构造函数或 set 方法	存储于 COS 上 Object 的 对象键	String

返回结果说明

- 成功：返回 InitiateMultipartUploadResult ，包含标志本次分块上传的 uploadId。
- 失败：发生错误（如身份认证失败），抛出异常 CosClientException 或者 CosServiceException。

请求示例

```
// Bucket的命名格式为 BucketName-APPID
String bucketName = "examplebucket-1250000000";
String key = "exampleobject";
InitiateMultipartUploadRequest initRequest = new InitiateMultipartUploadRequest(bucketName, key);
InitiateMultipartUploadResult initResponse = cosClient.initiateMultipartUpload(initRequest);
String uploadId = initResponse.getUploadId();
```

上传分块

上传分块（Upload Part）。

方法原型

```
public UploadPartResult uploadPart(UploadPartRequest uploadPartRequest) throws CosClientException, CosServiceException;
```

参数说明

参数名称	描述	类型
uploadPartRequest	请求	UploadPartRequest

Request 成员说明：

参数名称	设置方法	描述	类型
bucketName	set方法	Bucket 的命名格式为 BucketName-APPID	String
key	set方法	存储于 COS 上 Object 的 对象键	String
uploadId	set方法	标识指定分片上传的 uploadId	String
partNumber	set方法	标识指定分片的编号，必须 >= 1	int
inputStream	set方法	待上传分块的输入流	InputStream

返回结果说明

- 成功：返回 UploadPartResult，包含上传分块的eTag信息。
- 失败：发生错误（如身份认证失败），抛出异常 CosClientException 或者 CosServiceException。

请求示例

```
// 上传分块, 最多10000个分块, 分块大小支持为1M - 5G.
// 分块大小设置为4M. 如果总计 n 个分块, 则 1~n-1 的分块大小一致, 最后一块小于等于前面的分块大小
List<PartETag> partETags = new ArrayList<PartETag>();
int partNumber = 1;
int partSize = 4 * 1024 * 1024;
// partStream 代表 part 数据的输入流, 流长度为 partSize
UploadPartRequest uploadRequest = new UploadPartRequest().withBucketName(bucketName).
withUploadId(uploadId).withKey(key).withPartNumber(partNumber).
withInputStream(partStream).withPartSize(partSize);
UploadPartResult uploadPartResult = cosClient.uploadPart(uploadRequest);
String eTag = uploadPartResult.getETag(); // 获取 part 的 Etag
partETags.add(new PartETag(partNumber, eTag)); // partETags 记录所有已上传的 part 的 Etag 信息
// ... 上传 partNumber 第2个到第 n 个分块
```

复制分块

功能说明

将一个对象的分块内容从源路径复制到目标路径（Upload Part Copy）。

方法原型

```
public CopyPartResult copyPart(CopyPartRequest copyPartRequest) throws CosClientException, CosServiceException
```

参数说明

参数名称	描述	类型
copyPartRequest	请求	CopyPartRequest

Request 成员说明：

参数名称	设置方法	描述	类型
destinationBucketName	set 方法	目标存储桶名称，Bucket 的命名格式为 BucketName-APPID	String
destinationKey	set 方法	目标对象名称，存储于 COS 上 Object 的 对象键	String
uploadId	set 方法	标识指定分片上传的 uploadId	String
partNumber	set 方法	标识指定分片的编号，必须 >= 1	int
sourceBucketRegion	set 方法	源存储桶的区域	Region
sourceBucketName	set 方法	源存储桶的名称	String
sourceKey	set 方法	源对象名称，存储于 COS 上 Object 的 对象键	String
firstByte	set 方法	源对象的首字节偏移	Long
lastByte	set 方法	源对象的最后一字节偏移	Long

返回结果说明

- 成功：返回 CopyPartResult，包含分块的 ETag 信息。
- 失败：发生错误（如身份认证失败），抛出异常 CosClientException 或者 CosServiceException。

请求示例

```
// 1 初始化用户身份信息（secretId, secretKey）。
String secretId = "COS_SECRETID";
String secretKey = "COS_SECRETKEY";
COSCredentials cred = new BasicCOSCredentials(secretId, secretKey);
// 采用了新的 region 名字，可用 region 的列表可以在官网文档中获取，也可以参见下面的 XML SDK 和 JSON SDK 的地域对照表
ClientConfig clientConfig = new ClientConfig(new Region("ap-guangzhou"));
COSClient cosclient = new COSClient(cred, clientConfig);
// 存储桶名称，格式为：BucketName-APPID
// 设置目标存储桶名称，对象名称和分块上传id
String destinationBucketName = "examplebucket-1250000000";
String destinationTargetKey = "target_exampleobject";
String uploadId = "1559020734eeca6640329099e680457e0ef653a72dd70d4eade64205d270b532c22a38649e";
int partNumber = 1;
CopyPartRequest copyPartRequest = new CopyPartRequest();
copyPartRequest.setDestinationBucketName(destinationBucketName);
copyPartRequest.setDestinationKey(destinationTargetKey);
copyPartRequest.setUploadId(uploadId);
copyPartRequest.setPartNumber(partNumber);
// 设置源存储桶的区域和名称，以及对象名称，偏移量区间
String sourceBucketRegion = "ap-guangzhou";
String sourceBucketName = "examplebucket-1250000000";
String sourceKey = "exampleobject";
Long firstByte = 1L;
Long lastByte = 5242880L;
copyPartRequest.setSourceBucketRegion(new Region(sourceBucketRegion));
copyPartRequest.setSourceBucketName(sourceBucketName);
copyPartRequest.setSourceKey(sourceKey);
copyPartRequest.setFirstByte(firstByte);
copyPartRequest.setLastByte(lastByte);

CopyPartResult copyPartResult = cosclient.copyPart(copyPartRequest);
```

查询已上传块

功能说明

查询特定分块上传操作中的已上传的块（List Parts）。

方法原型

```
public PartListing listParts(ListPartsRequest request)
throws CosClientException, CosServiceException;
```

参数说明

参数名称	设置方法	描述	类型
bucketName	构造函数或 set 方法	Bucket 的命名格式为 BucketName-APPID	String
key	构造函数或 set 方法	对象的名称	String
uploadId	构造函数或 set 方法	本次要查询的分块上传的uploadId	String
maxParts	set 方法	单次返回最大的条目数量，默认1000	String
partNumberMarker	set 方法	默认以 UTF-8 二进制顺序列出条目，所有列出条目从 marker 开始	String
encodingType	set 方法	规定返回值的编码方式	String

返回结果说明

- 成功：返回 PartListing，包含每一分块的 ETag 和编号，以及下一次 list 的起点 marker。
- 失败：发生错误（如身份认证失败），抛出异常 CosClientException 或者 CosServiceException。

请求示例

```
// ListPart 用于在 complete 分块上传前或者 abort 分块上传前获取 uploadId 对应的已上传的分块信息, 可以用来构造 partEtags
List<PartETag> partETags = new ArrayList<PartETag>();
ListPartsRequest listPartsRequest = new ListPartsRequest(bucketName, key, uploadId);
do {
PartListing partListing = cosClient.listParts(listPartsRequest);
for (PartSummary partSummary : partListing.getParts()) {
partETags.add(new PartETag(partSummary.getPartNumber(), partSummary.getETag()));
}
listPartsRequest.setPartNumberMarker(partListing.getNextPartNumberMarker());
} while (partListing.isTruncated());
```

完成分块上传

功能说明

实现完成整个分块上传（Complete Multipart Upload）。

方法原型

```
public CompleteMultipartUploadResult completeMultipartUpload(CompleteMultipartUploadRequest request) throws CosClientException, CosServiceExcep
tion;
```

参数说明

参数名称	设置方法	描述	类型
bucketName	构造函数或 set 方法	Bucket 的命名格式为 BucketName-APPID	String
key	构造函数或 set 方法	存储于 COS 上 Object 的 对象键	String
uploadId	构造函数或 set 方法	标识指定分片上传的 uploadId	String
partETags	构造函数或 set 方法	标识分片块的编号和上传返回的 eTag	`List`

返回结果说明

- 成功：返回 CompleteMultipartUploadResult，包含完成对象的 eTag 信息。
- 失败：发生错误（如身份认证失败），抛出异常 CosClientException 或者 CosServiceException。

请求示例

```
// complete 完成分块上传.
CompleteMultipartUploadRequest compRequest = new CompleteMultipartUploadRequest(bucketName, key, uploadId, partETags);
CompleteMultipartUploadResult result = cosClient.completeMultipartUpload(compRequest);
```

终止分块上传

功能说明

终止一个分块上传操作并删除已上传的块（Abort Multipart Upload）。

方法原型

```
public void abortMultipartUpload(AbortMultipartUploadRequest request) throws CosClientException, CosServiceException;
```

参数说明

参数名称	设置方法	描述	类型
bucketName	构造函数或 set 方法	Bucket 的命名格式为 BucketName-APPID	String
key	构造函数或 set 方法	存储于 COS 上 Object 的 对象键	String
uploadId	构造函数或 set 方法	标识指定分片上传的 uploadId	String

返回结果说明

- 成功：无返回值。
- 失败：发生错误（如身份认证失败），抛出异常 CosClientException 或者 CosServiceException。

请求示例

```
// abortMultipartUpload 用于终止一个还未 complete 的分块上传
AbortMultipartUploadRequest abortMultipartUploadRequest = new AbortMultipartUploadRequest(bucketName, key, uploadId);
cosClient.abortMultipartUpload(abortMultipartUploadRequest);
```

其他操作

设置对象 ACL

功能说明

设置存储桶中某个对象的访问控制列表。

> **注意：**
>
> 当前访问策略条目限制为1000条，如果您不需要进行对象 ACL 控制，请不要设置，默认继承 Bucket 权限。

ACL 包括预定义权限策略（CannedAccessControlList）或者自定义的权限控制（AccessControlList）。两类权限当同时设置时将忽略预定义策略，以自定义策略为主。

方法原型

```
// 方法1 (设置自定义策略)
public void setObjectAcl(String bucketName, String key, AccessControlList acl)
throws CosClientException, CosServiceException
// 方法2 (设置预定义策略)
public void setObjectAcl(String bucketName, String key, CannedAccessControlList acl)
throws CosClientException, CosServiceException
// 方法3 (以上两种方法的封装, 包含两种策略设置，如果同时设置以自定定义策略为主)
public void setObjectAcl(SetObjectAclRequest setObjectAclRequest)
throws CosClientException, CosServiceException;
```

参数说明

- **方法3** 参数同时包含1和2，因此以方法3为例进行介绍。

参数名称	描述	类型
SetObjectAclRequest	请求类	setObjectAclRequest

Request 成员说明：

Request 成员	设置方法	描述	类型
bucketName	构造函数或 set 方法	存储桶的命名格式为 BucketName-APPID	String
key	构造函数或 set 方法	对象键（Key）是对象在存储桶中的唯一标识。例如，在对象的访问域名 `examplebucket-1250000000cos.ap-guangzhou.myqcloud.com/doc/picture.jpg` 中，对象键为 doc/picture.jpg, 详情请参见 对象键	String
acl	构造函数或 set 方法	自定义权限策略	AccessControlList
cannedAcl	构造函数或 set 方法	预定义策略如公有读、公有读写、私有读	CannedAccessControlList

成员名	描述	类型
List	包含所有要授权的信息	数组
owner	表示 Object 或者 Owner 的拥有者	Owner 类

Grant 类成员说明：

成员名	描述	类型
grantee	被授权人的身份信息	Grantee
permission	被授权的权限信息（例如可读，可写，可读可写）	Permission

Owner 类成员说明：

成员名	描述	类型
id	拥有者的身份信息	String
displayname	拥有者的名字（目前和 ID 相同）	String

CannedAccessControlList 表示预设的策略，针对的是所有人。是一个枚举类，枚举值如下所示：

枚举值	描述
Private	私有读写（仅有 owner 可以读写）
PublicRead	公有读私有写（ owner 可以读写， 其他客户可以读）
PublicReadWrite	公有读写（即所有人都可以读写）

返回结果说明

- 成功：无返回值。
- 失败：发生错误（如身份认证失败），抛出异常 CosClientException 或者 CosServiceException。

请求示例

```
// 权限信息中身份信息有格式要求, 对于主账号与子账号的范式如下：
// 下面的 root_uin 和 sub_uin 都必须是有效的 QQ 号
// 主账号 qcs::cam::uin/<root_uin>:uin/<root_uin> 表示授予主账号 root_uin 这个用户(即前后填的 uin 一样)
```

```
// 如 qcs::cam::uin/2779643970:uin/2779643970
// 子账号 qcs::cam::uin/<root_uin>:uin/<sub_uin> 表示授予 root_uin 的子账号 sub_uin 这个客户
// 如 qcs::cam::uin/2779643970:uin/73001122
// 存储桶的命名格式为 BucketName-APPID
String bucketName = "examplebucket-1250000000";
String key = "exampleobject";
// 设置自定义 ACL
AccessControlList acl = new AccessControlList();
Owner owner = new Owner();
// 设置 owner 的信息, owner 只能是主账号
owner.setId("qcs::cam::uin/2779643970:uin/2779643970");
acl.setOwner(owner);

// 授权给主账号73410000可读可写权限
UinGrantee uinGrantee1 = new UinGrantee("qcs::cam::uin/73410000:uin/73410000");
acl.grantPermission(uinGrantee1, Permission.FullControl);
// 授权给 2779643970 的子账号 72300000 可读权限
UinGrantee uinGrantee2 = new UinGrantee("qcs::cam::uin/2779643970:uin/72300000");
acl.grantPermission(uinGrantee2, Permission.Read);
// 授权给 2779643970 的子账号 7234444 可写权限
UinGrantee uinGrantee3 = new UinGrantee("qcs::cam::uin/7234444:uin/7234444");
acl.grantPermission(uinGrantee3, Permission.Write);
cosClient.setObjectAcl(bucketName, key, acl);

// 设置预定义 ACL
// 设置私有读写（Object 的权限默认集成 Bucket的）
cosClient.setObjectAcl(bucketName, key, CannedAccessControlList.Private);
// 设置公有读私有写
cosClient.setObjectAcl(bucketName, key, CannedAccessControlList.PublicRead);
// 设置公有读写
cosClient.setObjectAcl(bucketName, key, CannedAccessControlList.PublicReadWrite);
```

获取对象 ACL

功能说明

获取对象访问权限控制列表（ACL）（Get Object ACL）。

方法原型

```
public AccessControlList getObjectAcl(String bucketName, String key)
throws CosClientException, CosServiceException;
```

参数说明

参数名称	描述	类型
bucketName	存储桶的命名格式为 BucketName-APPID	String
key	对象键（Key）是对象在存储桶中的唯一标识。例如，在对象的访问域名`examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/picture.jpg`中，对象键为 doc/picture.jpg，详情请参见 对象键	String

返回结果说明

- 成功：返回一个 Object 所在的 ACL。
- 失败：发生错误（如身份认证失败），抛出异常 CosClientException 或者 CosServiceException。

请求示例

```
// 存储桶的命名格式为 BucketName-APPID
String bucketName = "examplebucket-1250000000";
String key = "exampleobject";
AccessControlList acl = cosClient.getObjectAcl(bucketName, key);
```

高级接口（推荐）

高级 API 由类 TransferManger 通过封装上传以及下载接口，内部有一个线程池，接受用户的上传和下载请求，因此用户可选择异步的提交任务。

```
// 线程池大小，建议在客户端与 COS 网络充足(如使用腾讯云金融专区的 CVM，同地域上传 COS)的情况下，设置成16或32即可, 可较充分的利用网络资源
// 对于使用公网传输且网络带宽质量不高的情况，建议减小该值，避免因网速过慢，造成请求超时。
ExecutorService threadPool = Executors.newFixedThreadPool(32);
// 传入一个 threadpool, 若不传入线程池, 默认 TransferManager 中会生成一个单线程的线程池。
TransferManager transferManager = new TransferManager(cosClient, threadPool);
// .....(提交上传下载请求, 如下文所属)
// 关闭 TransferManger
transferManager.shutdownNow();
```

上传对象

功能说明

上传接口根据用户文件的长度，自动选择简单上传以及分块上传， 降低用户的使用门槛。用户不用关心分块上传的每个步骤。

Tips 有关其他一些设置属性，存储类别，MD5 校验等可参见 Put Object Api。

方法原型

```
// 上传对象
public Upload upload(final PutObjectRequest putObjectRequest)
throws CosServiceException, CosClientException;
```

参数说明

参数名称	描述	类型
putObjectRequest	上传文件请求	PutObjectRequest

Request 成员说明：

Request 成员	设置方法	描述	类型
bucketName	构造函数或 set 方法	存储桶的命名格式为 BucketName-APPID	String
key	构造函数或 set 方法	对象键（Key）是对象在存储桶中的唯一标识。 例如，在对象的访问域名 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/picture.jpg` 中，对象键为 doc/picture.jpg，详情请参见 对象键	String
file	构造函数或 set 方法	本地文件	File
input	构造函数或 set 方法	输入流	InputStream
metadata	构造函数或 set 方法	文件的元数据	ObjectMetadata

返回值

- 成功：返回 Upload，可以查询上传是否结束，也可同步的等待上传结束。
- 失败：发生错误（如身份认证失败），抛出异常 CosClientException 或者 CosServiceException。

请求示例

```
// 示例1：
// 存储桶的命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式
String bucketName = "examplebucket-1250000000";
String key = "/doc/picture.jpg";
File localFile = new File("/doc/picture.jpg");
PutObjectRequest putObjectRequest = new PutObjectRequest(bucketName, key, localFile);
// 本地文件上传
Upload upload = transferManager.upload(putObjectRequest);
// 等待传输结束（如果想同步的等待上传结束，则调用 waitForCompletion）
UploadResult uploadResult = upload.waitForUploadResult();
```

```
// 示例2：对大于分块大小的文件，使用断点续传
// 步骤一：获取 PersistableUpload
String bucketName = "examplebucket-1250000000";
String key = "exmpleobject";
File localFile = new File("exmpleobject");
PutObjectRequest putObjectRequest = new PutObjectRequest(bucketName, key, localFile);
// 本地文件上传
PersistableUpload persistableUpload = null;
Upload upload = transferManager.upload(putObjectRequest);
// 等待"分块上传初始化"完成，并获取到 persistableUpload （包含uploadId等）
while(persistableUpload == null) {
persistableUpload = upload.getResumeableMultipartUploadId();
Thread.sleep(100);
}
// 保存 persistableUpload

// 步骤二：当由于网络等问题，大文件的上传被中断，则根据 PersistableUpload 恢复该文件的上传，只上传未上传的分块
Upload newUpload = transferManager.resumeUpload(persistableUpload);
// 等待传输结束（如果想同步的等待上传结束，则调用 waitForCompletion）
UploadResult uploadResult = newUpload.waitForUploadResult();
```

下载对象

功能说明

将 COS 上的对象下载到本地。

方法原型

```
// 下载对象
public Download download(final GetObjectRequest GetObjectRequest, final File file);
```

参数说明

参数名称	描述	类型
getObjectRequest	下载对象请求	GetObjectRequest
file	要下载到的本地文件	File

Request 成员说明：

Request 成员	设置方法	描述	类型
bucketName	构造函数或 set 方法	存储桶的命名格式为 BucketName-APPID	String
key	构造函数或 set 方法	对象键（Key）是对象在存储桶中的唯一标识。例如，在对象的访问域名 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/picture.jpg` 中，对象键为 doc/picture.jpg，详情请参见 对象键	String
range	set 方法	下载的 range 范围	Long[]

返回值

- 成功：返回 Download，可以查询下载是否结束，也可同步的等待下载结束。
- 失败：发生错误（如身份认证失败），抛出异常 CosClientException 或者 CosServiceException。

请求示例

```
// Bucket 的命名格式为 BucketName-APPID ，此处填写的存储桶名称必须为此格式
String bucketName = "examplebucket-1250000000";
String key = "/doc/picture.jpg";
File localDownFile = new File("/doc/picture.jpg");
GetObjectRequest getObjectRequest = new GetObjectRequest(bucketName, key);
// 下载文件
Download download = transferManager.download(getObjectRequest, localDownFile);
```

```
// 等待传输结束（如果想同步的等待上传结束，则调用 waitForCompletion）
download.waitForCompletion();
```

复制对象

功能说明

Copy 接口支持根据对象大小自动选择简单复制或者分块复制，用户无需关心复制的文件大小。

方法原型

```
// 上传对象
public Copy copy(final CopyObjectRequest copyObjectRequest);
```

参数说明

参数名称	描述	类型
copyObjectRequest	拷贝文件请求	CopyObjectRequest

Request 成员说明：

参数名称	描述	类型
sourceBucketRegion	源 Bucket Region 。默认值：与当前 clientconfig 的 region 一致，表示统一地域拷贝	String
sourceBucketName	源存储桶名称，存储桶的命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String
sourceKey	源对象键，对象键（Key）是对象在存储桶中的唯一标识。例如，在对象的访问域名 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/picture.jpg` 中，对象键为 doc/picture.jpg，详情请参见 对象键	String
sourceVersionId	源文件 version id（适用于开启了版本控制的源 Bucket）。默认值：源文件当前最新版本	String
destinationBucketName	目标存储桶名称，存储桶的命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式	String
destinationKey	目的对象键，对象键（Key）是对象在存储桶中的唯一标识。例如，在对象的访问域名 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/picture.jpg` 中，对象键为 doc/picture.jpg，详情请参见 对象键	String
storageClass	拷贝的目的文件的存储类型。默认值：Standard	String

返回值

- 成功：返回 Copy，可以查询 Copy 是否结束，也可同步的等待上传结束。
- 失败：发生错误（如身份认证失败），抛出异常 CosClientException 或者 CosServiceException。

请求示例

```
// 要拷贝的 bucket region, 支持跨地域拷贝
Region srcBucketRegion = new Region("ap-shanghai");
// 源 Bucket, 存储桶的命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式
String srcBucketName = "srcBucket-1251668577";
// 要拷贝的源文件
String srcKey = "exampleobject";
// 目的 Bucket, 存储桶的命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式
String destBucketName = "examplebucket-1250000000";
// 要拷贝的目的文件
String destKey = "exampleobject";

// 生成用于获取源文件信息的 srcCOSClient
COSClient srcCOSClient = new COSClient(cred, new ClientConfig(srcBucketRegion));
CopyObjectRequest copyObjectRequest = new CopyObjectRequest(srcBucketRegion, srcBucketName,
srcKey, destBucketName, destKey);
try {
Copy copy = transferManager.copy(copyObjectRequest, srcCOSClient, null);
// 返回一个异步结果 copy, 可同步的调用 waitForCopyResult 等待 copy 结束, 成功返回 CopyResult, 失败抛出异常.
CopyResult copyResult = copy.waitForCopyResult();
} catch (CosServiceException e) {
e.printStackTrace();
} catch (CosClientException e) {
```

```
e.printStackTrace();
} catch (InterruptedException e) {
e.printStackTrace();
}
```

客户端加密

功能说明

Java sdk 支持客户端加密, 将文件加密后再进行上传，并在下载时进行解密。客户端加密支持对称 AES 与非对称 RSA 加密。 这里的对称和非对称只是用来加密每次生成的随机密钥, 对文件数据的加密始终使用 AES256 对称加密。 客户端加密适用于存储敏感数据的客户，客户端加密会牺牲部分上传速度，SDK 内部对于分块上传会使用串行的方式进行上传。

使用客户端加密前准备事项

客户端加密内部使用 AES256 来对数据进行加密，默认 JDK6 - JDK8 早期的版本不支持256位加密，如果运行时会报出以下异常 java.security.InvalidKeyException: Illegal key size or default parameters 。那么我们需要补充 oracle 的 JCE 无政策限制权限文件，将其部署在 JRE 的环境中。 请根据目前使用的 JDK 版本，分别下载对应的文件，将其解压后保存在 JAVA_HOME 下的 jre/lib/security 目录下。

1. JDK6 JCE 补充包
2. JDK7 JCE 补充包
3. JDK8 JCE 补充包

上传加密流程

1. 每次上传一个文件对象前，我们随机生成一个对称加密密钥，随机生成的密钥通过用户的提供的对称或非对称密钥进行加密，将加密后的结果 base64 编码存储在对象的元数据中。
2. 进行文件对象的上传，上传时在内存使用 AES256 算法加密。

下载解密流程

1. 获取文件元数据中加密必要的信息，base64 解码后使用用户密钥进行解密，得到当时加密数据的密钥。
2. 使用密钥对下载输入流进行使用 AES256 解密，得到解密后的文件输入流。

请求示例

示例1：使用对称 AES256 加密每次生成的随机密钥示例，完整的示例代码请参见 客户端对称密钥加密完整示例。

```
// 初始化用户身份信息(secretId, secretKey)
String secretId = "COS_SECRETID";
String secretKey = "COS_SECRETKEY";
COSCredentials cred = new BasicCOSCredentials(secretId, secretKey);
// 设置存储桶地域
ClientConfig clientConfig = new ClientConfig(new Region("ap-beijing"));

// 加载保存在文件中的密钥, 如果不存在，请先使用 buildAndSaveSymmetricKey 生成密钥
// buildAndSaveSymmetricKey();
SecretKey symKey = loadSymmetricAESKey();

EncryptionMaterials encryptionMaterials = new EncryptionMaterials(symKey);
// 使用 AES/GCM 模式，并将加密信息存储在文件元数据中.
CryptoConfiguration cryptoConf = new CryptoConfiguration(CryptoMode.AuthenticatedEncryption)
.withStorageMode(CryptoStorageMode.ObjectMetadata);

// 生成加密客户端 EncryptionClient, COSEncryptionClient 是 COSClient 的子类, 所有 COSClient 支持的接口他都支持。
// EncryptionClient 覆盖了 COSClient 上传下载逻辑，操作内部会执行加密操作，其他操作执行逻辑和 COSClient 一致
COSEncryptionClient cosEncryptionClient =
new COSEncryptionClient(new COSStaticCredentialsProvider(cred),
new StaticEncryptionMaterialsProvider(encryptionMaterials), clientConfig,
cryptoConf);

// 上传文件
// 这里给出 putObject 的示例, 对于高级 API 上传，只用在生成 TransferManager 时传入 COSEncryptionClient 对象即可
String bucketName = "examplebucket-1250000000";
String key = "exampleobject";
File localFile = new File("src/test/resources/plain.txt");
```

```
PutObjectRequest putObjectRequest = new PutObjectRequest(bucketName, key, localFile);
cosEncryptionClient.putObject(putObjectRequest);
```

示例2：使用非对称 RSA 加密每次生成的随机密钥示例，完整的示例代码请参见 客户端非对称密钥加密完整示例。

```
// 初始化用户身份信息(secretId, secretKey)
String secretId = "COS_SECRETID";
String secretKey = "COS_SECRETKEY";
COSCredentials cred = new BasicCOSCredentials(secretId, secretKey);
// 设置存储桶地域，COS 地域的简称请参照 /document/product/436/6224
ClientConfig clientConfig = new ClientConfig(new Region("ap-beijing"));


// 加载保存在文件中的密钥, 如果不存在，请先使用 buildAndSaveAsymKeyPair 生成密钥
buildAndSaveAsymKeyPair();
KeyPair asymKeyPair = loadAsymKeyPair();

EncryptionMaterials encryptionMaterials = new EncryptionMaterials(asymKeyPair);
// 使用 AES/GCM 模式，并将加密信息存储在文件元数据中.
CryptoConfiguration cryptoConf = new CryptoConfiguration(CryptoMode.AuthenticatedEncryption)
.withStorageMode(CryptoStorageMode.ObjectMetadata);

// 生成加密客户端 EncryptionClient, COSEncryptionClient 是 COSClient 的子类, 所有COSClient 支持的接口他都支持。
// EncryptionClient 覆盖了 COSClient 上传下载逻辑，操作内部会执行加密操作，其他操作执行逻辑和 COSClient 一致
COSEncryptionClient cosEncryptionClient =
new COSEncryptionClient(new COSStaticCredentialsProvider(cred),
new StaticEncryptionMaterialsProvider(encryptionMaterials), clientConfig,
cryptoConf);

// 上传文件
// 这里给出 putObject 的示例，对于高级 API 上传，只用在生成 TransferManager 时传入 COSEncryptionClient 对象即可
String bucketName = "examplebucket-1250000000";
String key = "exampleobject";
File localFile = new File("src/test/resources/plain.txt");
PutObjectRequest putObjectRequest = new PutObjectRequest(bucketName, key, localFile);
cosEncryptionClient.putObject(putObjectRequest);
```

存储桶管理

简介

本文档提供关于跨域访问、生命周期、版本控制和跨地域复制相关的 API 概览以及 SDK 示例代码。

跨域访问

API	操作名	操作描述
PUT Bucket cors	设置跨域配置	设置存储桶的跨域访问权限
GET Bucket cors	查询跨域配置	查询存储桶的跨域访问配置信息
DELETE Bucket cors	删除跨域配置	删除存储桶的跨域访问配置信息

版本控制

API	操作名	操作描述
PUT Bucket versioning	设置版本控制	设置存储桶的版本控制功能
GET Bucket versioning	查询版本控制	查询存储桶的版本控制信息

跨地域复制

API	操作名	操作描述

API	操作名	操作描述
PUT Bucket replication	设置跨地域复制	设置存储桶的跨地域复制规则
GET Bucket replication	查询跨地域复制	查询存储桶的跨地域复制规则
DELETE Bucket replication	删除跨地域复制	删除存储桶的跨地域复制规则

跨域访问

设置跨域配置

功能说明

设置指定存储桶的跨域访问配置信息（PUT Bucket cors）。

方法原型

```
public void setBucketCrossOriginConfiguration(String bucketName, BucketCrossOriginConfiguration bucketCrossOriginConfiguration);
```

参数说明

参数名称	描述	类型
bucketName	存储桶的命名格式为 BucketName-APPID	String
bucketCrossOriginConfiguration	设置的存储桶跨域策略	BucketCrossOriginConfiguration

返回结果说明

- 成功：无返回值。
- 失败：发生错误（如身份认证失败），抛出异常 CosClientException 或者 CosServiceException。

请求示例

```
// bucket的命名格式为 BucketName-APPID ，此处填写的存储桶名称必须为此格式
String bucketName = "examplebucket-1250000000";
BucketCrossOriginConfiguration bucketCORS = new BucketCrossOriginConfiguration();
List<CORSRule> corsRules = new ArrayList<>();
CORSRule corsRule = new CORSRule();
// 规则名称
corsRule.setId("set-bucket-cors-test");
// 允许的 HTTP 方法
corsRule.setAllowedMethods(AllowedMethods.PUT, AllowedMethods.GET, AllowedMethods.HEAD);
corsRule.setAllowedHeaders("x-cos-grant-full-control");
corsRule.setAllowedOrigins("http://imgcache.finance.cloud.tencent.com:80mail.qq.com", "http://imgcache.finance.cloud.tencent.com:80www.qq.com", "htt
p://imgcache.finance.cloud.tencent.com:80video.qq.com");
corsRule.setExposedHeaders("x-cos-request-id");
corsRule.setMaxAgeSeconds(60);
corsRules.add(corsRule);
bucketCORS.setRules(corsRules);
cosClient.setBucketCrossOriginConfiguration(bucketName, bucketCORS);
```

查询跨域配置

功能说明

查询指定存储桶的跨域访问配置信息（GET Bucket cors）。

方法原型

```
public BucketCrossOriginConfiguration getBucketCrossOriginConfiguration(String bucketName)
throws CosClientException, CosServiceException;
```

参数说明

参数名称	描述	类型
bucketName	存储桶的命名格式为 BucketName-APPID	String

返回结果说明

- 成功：返回存储桶的跨域规则。
- 失败：发生错误（如身份认证失败），抛出异常 CosClientException 或者 CosServiceException。

请求示例

```
// bucket的命名格式为 BucketName-APPID ，此处填写的存储桶名称必须为此格式
String bucketName = "examplebucket-1250000000";
BucketCrossOriginConfiguration corsGet = cosClient.getBucketCrossOriginConfiguration(bucketName);
List<CORSRule> corsRules = corsGet.getRules();
for (CORSRule rule : corsRules) {
List<AllowedMethods> allowedMethods = rule.getAllowedMethods();
List<String> allowedHeaders = rule.getAllowedHeaders();
List<String> allowedOrigins = rule.getAllowedOrigins();
List<String> exposedHeaders = rule.getExposedHeaders();
int maxAgeSeconds = rule.getMaxAgeSeconds();
}
```

删除跨域配置

功能说明

删除指定存储桶的跨域访问配置（DELETE Bucket cors）。

方法原型

```
public void deleteBucketCrossOriginConfiguration(String bucketName)
throws CosClientException, CosServiceException;
```

参数说明

参数名称	描述	类型
bucketName	存储桶的命名格式为 BucketName-APPID	String

返回结果说明

- 成功：无返回值。
- 失败：发生错误（如身份认证失败），抛出异常 CosClientException 或者 CosServiceException。

请求示例

```
//存储桶的命名格式为 BucketName-APPID
String bucketName = "examplebucket-1250000000";
cosClient.deleteBucketCrossOriginConfiguration(bucketName);
```

版本控制

设置版本控制

功能说明

设置指定存储桶的版本控制功能（PUT Bucket versioning）。

方法原型

```
public void setBucketVersioningConfiguration(SetBucketVersioningConfigurationRequest setBucketVersioningConfigurationRequest)
throws CosClientException, CosServiceException;
```

参数说明

参数名称	描述	类型
setBucketVersioningConfigurationRequest	版本控制配置	SetBucketVersioningConfigurationRequest

返回结果说明

- 成功：无返回值。
- 失败：发生错误（如身份认证失败），抛出异常 CosClientException 或者 CosServiceException。

请求示例

开启版本控制

```
String bucketName = "examplebucket-1250000000";
// 开启版本控制
cosClient.setBucketVersioningConfiguration(
new SetBucketVersioningConfigurationRequest(bucketName,
new BucketVersioningConfiguration(BucketVersioningConfiguration.ENABLED)));
```

暂停版本控制

```
String bucketName = "examplebucket-1250000000";
// 暂停版本控制
cosClient.setBucketVersioningConfiguration(
new SetBucketVersioningConfigurationRequest(bucketName,
new BucketVersioningConfiguration(BucketVersioningConfiguration.SUSPENDED)));
```

查询版本控制

功能说明

查询指定存储桶的版本控制信息（GET Bucket versioning）。

方法原型

```
// 方法1 传入存储桶名称即可
public BucketVersioningConfiguration getBucketVersioningConfiguration(String bucketName)
throws CosClientException, CosServiceException;

// 方法2 通过GetBucketVersioningConfigurationRequest 获取
public BucketVersioningConfiguration getBucketVersioningConfiguration(
GetBucketVersioningConfigurationRequest getBucketVersioningConfigurationRequest)
throws CosClientException, CosServiceException;
```

参数说明

参数名称	描述	类型
bucketName	存储桶的命名格式为 BucketName-APPID	String
getBucketVersioningConfigurationRequest	获取版本控制配置请求	GetBucketVersioningConfigurationRequest

返回结果说明

- 成功：返回存储桶的多版本配置。
- 失败：发生错误（如身份认证失败），抛出异常 CosClientException 或者 CosServiceException。

请求示例

```
String bucketName = "examplebucket-1250000000";
// 获取版本控制
BucketVersioningConfiguration bvc =
cosClient.getBucketVersioningConfiguration(bucketName);
// 获取版本控制
BucketVersioningConfiguration bvc2 = cosClient.getBucketVersioningConfiguration(
new GetBucketVersioningConfigurationRequest(bucketName));
```

跨地域复制

设置跨地域复制

功能说明

设置指定存储桶的跨地域复制规则（PUT Bucket replication）。

方法原型

```
public void setBucketReplicationConfiguration(
SetBucketReplicationConfigurationRequest setBucketReplicationConfigurationRequest)
throws CosClientException, CosServiceException;
```

参数说明

参数名	参数描述	类型
bucketName	存储桶的命名格式为 BucketName-APPID	String
setBucketReplicationConfigurationRequest	跨园区复制配置	SetBucketReplicationConfigurationRequest

返回结果说明

- 成功：无返回值。
- 失败：发生错误（如身份认证失败），抛出异常 CosClientException 或者 CosServiceException。

请求示例

```
// 源存储桶名称，需包含 appid
String bucketName = "examplebucket-1250000000";

BucketReplicationConfiguration bucketReplicationConfiguration = new BucketReplicationConfiguration();
// 设置发起者身份, 格式为： qcs::cam::uin/<OwnerUin>:uin/<SubUin>
bucketReplicationConfiguration.setRoleName("qcs::cam::uin/123456789:uin/987654543");

// 设置目标存储桶和存储类型，QCS 的格式为：qcs::cos:[region]::[bucketname-AppId]
ReplicationDestinationConfig replicationDestinationConfig = new ReplicationDestinationConfig();
replicationDestinationConfig.setBucketQCS("qcs::cos:ap-chongqing::examplebucket-target-1250000000");
replicationDestinationConfig.setStorageClass(StorageClass.Standard);

// 设置规则状态和前缀
ReplicationRule replicationRule = new ReplicationRule();
replicationRule.setStatus(ReplicationRuleStatus.Enabled);
replicationRule.setPrefix("");
replicationRule.setDestinationConfig(replicationDestinationConfig);
// 添加规则
String ruleId = "replication-to-chongqing";
bucketReplicationConfiguration.addRule("replication-to-chongqing", replicationRule);

try {
SetBucketReplicationConfigurationRequest setBucketReplicationConfigurationRequest =
new SetBucketReplicationConfigurationRequest(bucketName, bucketReplicationConfiguration);
cosclient.setBucketReplicationConfiguration(setBucketReplicationConfigurationRequest);
} catch (CosServiceException e) {
e.printStackTrace();
} catch (CosClientException e) {
e.printStackTrace();
} finally {
cosclient.shutdown();
}
```

查询跨地域复制

功能说明

查询指定存储桶的跨地域复制规则（GET Bucket replication）。

方法原型

```
// 获取存储桶跨地域复制配置方法1
public BucketReplicationConfiguration getBucketReplicationConfiguration(String bucketName)
throws CosClientException, CosServiceException;

// 获取存储桶跨地域复制方法2
public BucketReplicationConfiguration getBucketReplicationConfiguration(
GetBucketReplicationConfigurationRequest getBucketReplicationConfigurationRequest)
throws CosClientException, CosServiceException;
```

参数说明

参数名	参数描述	类型
bucketName	存储桶的命名格式为 BucketName-APPID	String
getBucketReplicationConfigurationRequest	获取跨地域复制配置请求	GetBucketReplicationConfigurationRequest

返回结果说明

- 成功：返回存储桶的跨地域复制规则。
- 失败：发生错误（如身份认证失败），抛出异常 CosClientException 或者 CosServiceException。

请求示例

```
String bucketName = "examplebucket-1250000000";

// 获取存储桶跨地域复制配置方法1
BucketReplicationConfiguration brcfRet = cosClient.getBucketReplicationConfiguration(bucketName);

// 获取存储桶跨地域复制配置方法2
BucketReplicationConfiguration brcfRet2 = cosClient.getBucketReplicationConfiguration(
new GetBucketReplicationConfigurationRequest(bucketName));
```

删除跨地域复制

功能说明

删除指定存储桶的跨地域复制规则（DELETE Bucket replication）。

方法原型

```
// 删除存储桶跨地域复制配置方法1
public void deleteBucketReplicationConfiguration(String bucketName)
throws CosClientException, CosServiceException;

// 删除存储桶跨地域复制方法2
public void deleteBucketReplicationConfiguration(
DeleteBucketReplicationConfigurationRequest deleteBucketReplicationConfigurationRequest)
throws CosClientException, CosServiceException;
```

参数说明

参数名	参数描述	类型
bucketName	存储桶的命名格式为 BucketName-APPID	String
deleteBucketReplicationConfigurationRequest	删除跨地域复制配置请求	DeleteBucketReplicationConfigurationRequest

返回结果说明

- 成功：无返回值。
- 失败：发生错误（如身份认证失败），抛出异常 CosClientException 或者 CosServiceException。

请求示例

```
String bucketName = "examplebucket-1250000000";
```

```
// 删除存储桶跨地域复制配置方法1
cosClient.deleteBucketReplicationConfiguration(bucketName);

// 删除存储桶跨地域复制配置方法2
cosClient.deleteBucketReplicationConfiguration(new DeleteBucketReplicationConfigurationRequest(bucketName));
```

预签名 URL

简介

Java SDK 提供获取请求预签名 URL 和生成签名接口，可以分发给客户端，用于下载或者上传。如果您的文件是私有读权限，那么请注意预签名链接只有一定的有效期。

获取请求预签名 URL

方法原型

```
public URL generatePresignedUrl(GeneratePresignedUrlRequest req) throws CosClientException
```

参数说明

参数名称	描述	类型
req	预签名请求类	GeneratePresignedUrlRequest

Request 成员说明：

Request 成员	设置方法	描述	类型
method	构造函数或 set 方法	HTTP 方法，可选：GET、POST、PUT、DELETE、HEAD	HttpMethodName
bucketName	构造函数或 set 方法	存储桶名称，存储桶的命名格式为 BucketName-APPID	String
key	构造函数或 set 方法	对象键（Key）是对象在存储桶中的唯一标识，详情请参见 对象键	String
expiration	set 方法	签名过期的时间	Date
contentType	set 方法	要签名的请求中的 Content-Type	String
contentMd5	set 方法	要签名的请求中的 Content-Md5	String
responseHeaders	set 方法	签名的下载请求中要覆盖的返回的 HTTP 头	ResponseHeaderOverrides
versionId	set 方法	在存储桶开启多版本的时候，指定对象的版本号	String

示例1

使用永久密钥生成一个带签名的下载链接，示例代码如下：

```
// 初始化用户身份信息
String secretId = "COS_SECRETID";
String secretKey = "COS_SECRETKEY";
COSCredentials cred = new BasicCOSCredentials(secretId, secretKey);

// 初始化客户端配置
String region = "REGION";
String domain = "DOMAIN.COM";
SelfDefinedEndpointBuilder selfDefinedEndpointBuilder = new SelfDefinedEndpointBuilder();
ClientConfig clientConfig = new ClientConfig(new Region(region));
clientConfig.setEndpointBuilder(selfDefinedEndpointBuilder);

// 生成 COS 客户端
COSClient cosClient = new COSClient(cred, clientConfig);
```

```
// 存储桶的命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式
String bucketName = "examplebucket-1250000000";
String key = "exampleobject";
GeneratePresignedUrlRequest req =
new GeneratePresignedUrlRequest(bucketName, key, HttpMethodName.GET);
// 设置签名过期时间(可选), 若未进行设置, 则默认使用 ClientConfig 中的签名过期时间(1小时)
// 这里设置签名在半个小时后过期
Date expirationDate = new Date(System.currentTimeMillis() + 30L * 60L * 1000L);
req.setExpiration(expirationDate);
URL url = cosClient.generatePresignedUrl(req);
System.out.println(url.toString());
cosClient.shutdown();
```

示例2

使用临时密钥生成一个带签名的下载链接，并设置覆盖要返回的一些公共头部（例如 content-type，content-language），示例代码如下：

```
// 传入获取到的临时密钥 (tmpSecretId, tmpSecretKey, sessionToken)
String tmpSecretId = "COS_SECRETID";
String tmpSecretKey = "COS_SECRETKEY";
String sessionToken = "COS_TOKEN";
COSCredentials cred = new BasicSessionCredentials(tmpSecretId, tmpSecretKey, sessionToken);

// 初始化客户端配置
String region = "REGION";
String domain = "DOMAIN.COM";
SelfDefinedEndpointBuilder selfDefinedEndpointBuilder = new SelfDefinedEndpointBuilder();
ClientConfig clientConfig = new ClientConfig(new Region(region));
clientConfig.setEndpointBuilder(selfDefinedEndpointBuilder);

// 生成 COS 客户端
COSClient cosClient = new COSClient(cred, clientConfig);

// 存储桶的命名格式为 BucketName-APPID
String bucketName = "examplebucket-1250000000";
String key = "exampleobject";
GeneratePresignedUrlRequest req =
new GeneratePresignedUrlRequest(bucketName, key, HttpMethodName.GET);

// 设置下载时返回的 http 头
ResponseHeaderOverrides responseHeaders = new ResponseHeaderOverrides();
String responseContentType = "image/x-icon";
String responseContentLanguage = "zh-CN";
String responseContentDispositon = "filename=\"exampleobject\"";
String responseCacheControl = "no-cache";
String cacheExpireStr =
DateUtils.formatRFC822Date(new Date(System.currentTimeMillis() + 24L * 3600L * 1000L));
responseHeaders.setContentType(responseContentType);
responseHeaders.setContentLanguage(responseContentLanguage);
responseHeaders.setContentDisposition(responseContentDispositon);
responseHeaders.setCacheControl(responseCacheControl);
responseHeaders.setExpires(cacheExpireStr);
req.setResponseHeaders(responseHeaders);

// 设置签名过期时间(可选)，若未进行设置，则默认使用 ClientConfig 中的签名过期时间(1小时)
// 这里设置签名在半个小时后过期
Date expirationDate = new Date(System.currentTimeMillis() + 30L * 60L * 1000L);
req.setExpiration(expirationDate);
URL url = cosClient.generatePresignedUrl(req);
System.out.println(url.toString());
cosClient.shutdown();
```

示例3

生成公有读 Bucket（匿名可读），不需要签名的链接，示例代码如下：

```
// 生成匿名的请求签名，需要重新初始化一个匿名的 cosClient
// 初始化用户身份信息, 匿名身份不用传入 SecretId、SecretKey 等密钥信息
COSCredentials cred = new AnonymousCOSCredentials();
```

```
// 设置 bucket 的区域，COS 地域的简称请参照 /document/product/436/6224
ClientConfig clientConfig = new ClientConfig(new Region("ap-beijing"));
// 生成 cos 客户端
COSClient cosClient = new COSClient(cred, clientConfig);
// bucket 名需包含 appid
String bucketName = "examplebucket-1250000000";

String key = "exampleobject";
GeneratePresignedUrlRequest req =
new GeneratePresignedUrlRequest(bucketName, key, HttpMethodName.GET);
URL url = cosClient.generatePresignedUrl(req);
System.out.println(url.toString());
cosClient.shutdown();
```

示例4

生成一些预签名的上传链接，可直接分发给客户端进行文件的上传，示例代码如下：

```
// 存储桶的命名格式为 BucketName-APPID，此处填写的存储桶名称必须为此格式
String bucketName = "examplebucket-1250000000";
String key = "exampleobject";
// 设置签名过期时间(可选), 若未进行设置, 则默认使用 ClientConfig 中的签名过期时间(1小时)
// 这里设置签名在半个小时后过期
Date expirationTime = new Date(System.currentTimeMillis() + 30L * 60L * 1000L);
URL url = cosClient.generatePresignedUrl(bucketName, key, expirationTime, HttpMethodName.PUT);
System.out.println(url.toString());
cosClient.shutdown();
```

生成签名

COSSigner 类提供构造 COS 签名的方法，用于分发给移动端 SDK，进行文件的上传和下载。签名的路径和分发后要进行操作的 key 相匹配。

方法原型

```
// 构造 COS 签名
public String buildAuthorizationStr(HttpMethodName methodName, String resouce_path,
COSCredentials cred, Date expiredTime);

// 构造 COS 签名
// 第二个方法比第一个方法额外提供对部分 HTTP Header 和所有传入的 URL 中的参数进行签名
// 用于更复杂的签名控制, 生成的签名必须在上传下载等操作时，也要携带对应的 header 和 param
public String buildAuthorizationStr(HttpMethodName methodName, String resouce_path,
Map<String, String> headerMap, Map<String, String> paramMap, COSCredentials cred,
Date expiredTime);
```

参数说明

参数名称	描述	类型
methodName	HTTP 请求方法，可设置 PUT、GET、DELETE、HEAD、POST	HttpMethodName
resouce_path	要签名的路径, 同上传文件的 key，需要以`/`开始	HttpMethodName
cred	密钥信息	COSCredentials
expiredTime	过期时间	Date
headerMap	要签名的 HTTP Header map，只对传入的 Content-Type，Content-Md5 和以 x 开头的 header 进行签名	Map
paramMap	要签名的 URL Param map	Map

返回值

签名字符串，类型为 String。

示例1：生成一个上传签名

```
String secretId = "COS_SECRETID";
String secretKey = "COS_SECRETKEY";
COSCredentials cred = new BasicCOSCredentials(secretId, secretKey);
COSSigner signer = new COSSigner();
//设置过期时间为1个小时
Date expiredTime = new Date(System.currentTimeMillis() + 3600L * 1000L);
// 要签名的 key, 生成的签名只能用于对应此 key 的上传
String key = "/exampleobject";
String sign = signer.buildAuthorizationStr(HttpMethodName.PUT, key, cred, expiredTime);
```

示例2：生成一个下载签名

```
String secretId = "COS_SECRETID";
String secretKey = "COS_SECRETKEY";
COSCredentials cred = new BasicCOSCredentials(secretId, secretKey);
COSSigner signer = new COSSigner();
// 设置过期时间为1个小时
Date expiredTime = new Date(System.currentTimeMillis() + 3600L * 1000L);
// 要签名的 key, 生成的签名只能用于对应此 key 的下载
String key = "/exampleobject";
String sign = signer.buildAuthorizationStr(HttpMethodName.GET, key, cred, expiredTime);
```

示例3：生成一个删除签名

```
String secretId = "COS_SECRETID";
String secretKey = "COS_SECRETKEY";
COSCredentials cred = new BasicCOSCredentials(secretId, secretKey);
COSSigner signer = new COSSigner();
// 设置过期时间为1个小时
Date expiredTime = new Date(System.currentTimeMillis() + 3600L * 1000L);
// 要签名的 key, 生成的签名只能用于对应此 key 的删除
String key = "/exampleobject";
String sign = signer.buildAuthorizationStr(HttpMethodName.DELETE, key, cred, expiredTime);
```

异常处理

简介

调用 SDK 请求 COS 服务失败时，抛出的异常皆是 RuntimeExcpetion，目前 SDK 常见的异常有 CosClientException，CosServiceException 和 IllegalArgumentException。

客户端异常

客户端异常 CosClientException，是由于客户端原因导致无法和服务端完成正常的交互而导致的失败，如客户端无法连接到服务端，无法解析服务端返回的数据，读取本地文件发生 IO 异常等。CosClientException 继承自 RuntimeException，没有自定义的成员变量，使用方法同 RuntimeException。

服务端异常

服务端异常 CosServiceException，用于指交互正常完成，但是操作失败的场景。例如客户端访问一个不存在 Bucket，删除一个不存在的文件，没有权限进行某个操作， 服务端故障异常等。CosServiceException 包含了服务端返回的状态码，requestid，出错明细等。捕获异常后，建议对整个异常进行打印，异常包含了必须的排查因素。以下是异常成员变量的描述：

request 成员	描述	类型
requestId	请求 ID， 用于表示一个请求， 对于排查问题十分重要	String
traceId	辅助排查问题的 ID，	String

request 成员	描述	类型
statusCode	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。 请参照 [COS 错误信息]	String
errorType	枚举类， 表示异常的种类， 分为 Client， Service， Unknown	ErrorType
errorCode	请求失败时 body 返回的 Error Code 请参照 [COS 错误信息]	String
errorMessage	请求失败时 body 返回的 Error Message 请参照 [COS 错误信息]	String

常见问题

若您在使用 Java SDK 过程中，有相关的疑问，请联系维护工程师。

JavaScript SDK 快速入门

最近更新时间: 2025-01-15 17:01:00

开发准备

SDK 获取

对象存储服务的 XML JS SDK 资源下载地址：XML JS SDK。 演示示例 Demo 下载地址：XML JS SDK Demo。

开发准备

1. 首先，JS SDk 需要浏览器支持基本的 HTML5 特性，以便支持 ajax 上传文件和计算文件 md5 值。

2. 到 COS 对象存储控制台创建存储桶，得到 Bucket（存储桶名称） 和 Region（地域名称）。

3. 到云API密钥获取您的项目 SecretId 和 SecretKey。

4. 配置 CORS 规则，配置例子如下图：

> 关于文章中出现的 SecretId、SecretKey、Bucket 等名称的含义和获取方式请参考：COS 术语信息

快速入门

计算签名

由于签名计算放在前端会暴露 SecretId 和 SecretKey，我们把签名计算过程放在后端实现，前段通过 ajax 向后端获取签名结果，正式部署时请再后端加一层自己网站本身的权限检验。

这里提供 PHP 和 NodeJS 的签名例子，其他语言，请参照对应的 XML SDK

上传例子

1. 创建 test.html，填入下面的代码，修改里面的 Bucket 和 Region。
2. 部署好后端的签名服务，并修改 getAuthorization 里的签名服务地址。
3. 把 test.html 放在 Web 服务器下，然后在浏览器访问页面，测试文件上传。

```
<input id="file-selector" type="file">
<script src="dist/cos-js-sdk-v5.min.js"></script>
<script>
var bucket = 'BUCKET'; // 替换成用户的 Bucket
var region = 'REGION'; // 替换成用户的 Region
var domain = 'DOMAIN.COM'; // 替换成用户的 Domain

var endpoint = 'cos.' + region + '.' + domain;

// 初始化实例
var cos = new COS({
getAuthorization: function (options, callback) {
// 异步获取签名
$.get('../server/auth.php', {
method: (options.Method || 'get').toLowerCase(),
pathname: '/' + (options.Key || '')
}, function (authorization) {
callback(authorization);
}, 'text');
},
ServiceDomain: endpoint,
Domain: '{Bucket}.' + endpoint // 传入模板字符串
```

```
});

// 监听选文件
document.getElementById('file-selector').onchange = function () {

var file = this.files[0];
if (!file) return;

// 分片上传文件
cos.sliceUploadFile({
Bucket: bucket,
Region: region,
Key: file.name,
Body: file,
}, function (err, data) {
console.log(err, data);
});

};
</script>
```

webpack 引入方式

支持 webpack 打包的场景，可以用 npm 引入作为模块。

```
npm i cos-js-sdk-v5 --save
```

其他文档和例子

1. 更多例子请参阅 XML JavaScript SDK Demo。
2. 完整接口文档请参阅 XML JavaScript SDK 接口文档。

接口文档

最近更新时间: 2025-01-15 17:01:00

本文针对 JavaScript SDK 的接口做详细的介绍说明。

下文中在代码里出现的 COS 代表 SDK 的 类名，cos 代表 SDK 的实例。

下文中出现的 SecretId、SecretKey、Bucket、Region 等名称的含义和获取方式请参考：COS 术语信息

下文中参数名称前的 - 代表"子参数"。

构造函数

new COS({})

直接 script 标签引用 SDK 时，SDK 占用了全局变量名 COS，通过它的构造函数可以创建 SDK 实例。

使用示例

创建一个 COS SDK 实例：

```
var cos = new COS({
// 必选参数
getAuthorization: function (options, callback) {
$.get('http://imgcache.finance.cloud.tencent.com:80example.com/server/auth.php', {
method: options.Method,
pathname: '/' + options.Key,
}, function (authorization) {
callback(authorization);
});
},
// 可选参数
FileParallelLimit: 3, // 控制文件上传并发数
ChunkParallelLimit: 3, // 控制单个文件下分片上传并发数
ProgressInterval: 1000, // 控制上传的 onProgress 回调的间隔
});
```

- 使用临时密钥格式一：

```
var cos = new COS({
// 必选参数
getAuthorization: function (options, callback) {
$.get('http://imgcache.finance.cloud.tencent.com:80example.com/server/sts-auth.php', {
method: options.Method,
pathname: '/' + options.Key,
}, function (data) {
callback({
Authorization: data.Authorization,
XCosSecurityToken: data.XCosSecurityToken
});
});
}
});
```

- 使用临时密钥格式二：

```
var cos = new COS({
// 必选参数
getAuthorization: function (options, callback) {
$.get('http://imgcache.finance.cloud.tencent.com:80example.com/server/sts.php', {
bucket: options.Bucket,
region: options.Region,
}, function (data) {
callback({
SecretId: data.SecretId,
SecretKey: data.SecretKey,
```

```
XCosSecurityToken: data.XCosSecurityToken,
ExpiredTime: data.ExpiredTime,
});
});
}
});
```

- 使用固定密钥，前端计算签名（建议只在调试使用，避免泄露密钥）：

```
var cos = new COS({
SecretId: 'AKIDxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
SecretKey: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
});
```

构造函数参数说明

参数名	参数描述	类型	必填
SecretId	用户的 SecretId	String	否
SecretKey	用户的 SecretKey，建议只在前端调试时使用，避免暴露密钥	String	否
FileParallelLimit	同一个实例下上传的文件并发数，默认值 3	Number	否
ChunkParallelLimit	同一个上传文件的分片并发数，默认值 3	Number	否
ChunkSize	分片上传时，每片的大小字节数，默认值 1048576 (1MB)	Number	否
ProgressInterval	上传进度的回调方法 onProgress 的回调频率，单位 ms ，默认值 1000	Number	否
Protocol	自定义的请求协议，可选项 `https:`、`http:`，默认判断当前页面是 `http:` 时使用 `http:`，否则使用 `https:`	String	否
getAthorization	获取签名的回调方法，如果没有 SecretKey 或者 etSTS 这个参数必选	Function	否
getSTS	获取临时密钥的回调方法，每次过期会调用一次	Function	否

getAuthorization 的函数说明

```
function(options, callback) { ... }
```

getAuthorization 的函数说明回调参数说明：

参数名	参数描述	类型	必填
options	获取签名需要的参数对象	Function	否
- Method	当前请求的 Method	Function	否
- Key	当前请求的 Key	Function	否
- Query	当前请求的 query 参数对象，{key: 'val'} 的格式	Object	否
- Headers	当前请求的 header 参数对象，{key: 'val'} 的格式	Function	否
callback	临时密钥获取完成后的回调	Function	否

- getAuthorization 计算完成后，callback 回传一个签名字符串或一个对象： 回传签名字符串时，回传字符串类型，是请求要用的鉴权凭证 Authorization。
- 回传对象时，回传对象属性列表如下：

属性名	参数描述	类型	必填
Authorization	获取回来的临时密钥的	String	是
XCosSecurityToken	获取回来的临时密钥的 sessionToken，对应 header 的 x-cos-security-token 字段	String	否

getSTS 回调函数说明

```
function(options, callback) { ... }
```

getSTS 的回调参数说明：

参数名	参数描述	类型
options	获取临时密钥需要的参数对象	Function
- Bucket	Bucket 的名称。命名规则为{name}-{appid} ，此处填写的存储桶名称必须为此格式	String
- Region	Bucket 所在区域。	String
callback	临时密钥获取完成后的回传方法	Function

getSTS 计算完成后，callback 回传一个对象，回传对象的属性列表如下：

属性名	参数描述	类型	必填
SecretId	获取回来的临时密钥的 tmpSecretId	String	是
SecretKey	获取回来的临时密钥的 tmpSecretKey	String	否
XCosSecurityToken	获取回来的临时密钥的 sessionToken，对应 header 的 x-cos-security-token 字段	String	否
ExpiredTime	获取回来的临时密钥的 expiredTime，超时时间	String	否

获取鉴权凭证

实例本身鉴权凭证可以通过实例化时传入的参数控制如何或获取，有三种获取方式：

1. 实例化时传入 SecretId、SecretKey，每次需要签名都由实例内部计算。
2. 实例化时，传入 getAuthorization 回调，每次需要签名通过这个回调计算完返回签名给实例。
3. 实例化时，传入 getSTS 回调，每次需要临时密钥通过这个回调回去完返回给实例，在每次请求时实例内部使用临时密钥计算得到签名。

静态方法

COS.getAuthorization

COS XML API 的请求里，私有资源操作都需要鉴权凭证 Authorization，用于判断当前请求是否合法。

鉴权凭证使用方式有两种：

1. 放在 header 参数里使用，自断名: authorization
2. 放在 url 参数里使用，字段名：sign

COS.getAuthorization 方法用于计算鉴权凭证（Authorization），用以验证请求合法性的签名信息。

> **注意：**
>
> 该方法推荐只在前端调试时使用，项目上线不推荐使用前端计算签名的方法，有暴露秘钥的风险。

使用示例

获取文件下载的鉴权凭证：

```
var Authorization = COS.getAuthorization({
SecretId: 'AKIDxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
SecretKey: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
Method: 'get',
Key: 'a.jpg',
Expires: 60,
Query: {},
Headers: {}
});
```

参数说明

参数名	参数描述	类型	必填
SecretId	用户的 SecretId	String	是

参数名	参数描述	类型	必填
SecretKey	用户的 SecretKey	String	是
Method	操作方法，如 get，post，delete， head 等 HTTP 方法	String	是
Key	操作的 Object 名称，**如果请求操作是对文件的，则为文件名，且为必须参数。**如果操作是对于 Bucket，则为空	String	否
Expires	签名超时秒数，默认 900 秒	Number	否
Query	请求的 query 参数对象	Object	否
Headers	请求的 header 参数对象	Object	否

返回值说明

返回值是计算得到的鉴权凭证字符串 authorization。

工具方法

Get Auth

cos.getAuth 方法是 COS.getAuthorization 挂在实例上的版本，区别是 cos.getAuth 不需要传入 SecretId 和 SecretKey，会使用对象本身获取鉴权凭证的方法。

使用示例

```
var authorization = cos.getAuth({
Method: 'get',
Key: '1.jpg'
});
```

参数说明

参数名	参数描述	类型	必填
Method	操作方法，如 get，post，delete， head 等 HTTP 方法	String	是
Key	操作的 object 名称，**如果请求操作是对文件的，则为文件名，且为必须参数。**如果操作是对于 Bucket，则为空	String	否
Expires	签名超时秒数，默认 900 秒	Number	否
Query	请求的 query 参数对象	Object	否
Headers	请求的 header 参数对象	Object	否

返回值说明

返回值是计算得到的鉴权凭证字符串 authorization

Get Object Url

使用示例

// 获取不带签名 Object Url

```
var url = cos.getObjectUrl({
Key: '1.jpg',
Sign: false
});
```

// 获取带签名的 Object Url

```
cos.getObjectUrl({
Key: '1.jpg',
Sign: true
}, function (err, data) {
console.log(err || data.Url);
});
```

参数说明

参数名	参数描述	类型	必填
Bucket	Bucket 的名称。命名规则为{name}-{appid} ，此处填写的存储桶名称必须为此格式	String	是
Region	Bucket 所在区域。	String	是
Key	操作的 object 名称，**如果请求操作是对文件的，则为文件名，且为必须参数。**如果操作是对于 Bucket，则为空	String	是
Sign	是否返回带有签名的 Url	Boolean	否
Method	操作方法，如 get，post，delete， head 等 HTTP 方法，默认 get	String	否
Query	参与签名计算的 query 参数对象	Object	否
Headers	参与签名计算的 header 参数对象	Object	否

返回值说明

返回值是一个字符串，两种情况：

1. 如果签名计算可以同步计算（如：实例化传入了 SecretId 和 SecretKey），则默认返回带签名的 url
2. 否则返回不带签名的 url

回调函数说明

```
function(err, data) { ... }
```

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误。如果请求成功，则为空字符串	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- Url	计算得到的 Url	String

Bucket 操作

Head Bucket

功能说明

Head Bucket 请求可以确认该 Bucket 是否存在，是否有权限访问。Head 的权限与 Read 一致。当该 Bucket 存在时，返回 HTTP 状态码 200；当该 Bucket 无访问权限时，返回 HTTP 状态码 403；当该 Bucket 不存在时，返回 HTTP 状态码 404。

使用示例

```
cos.headBucket({
Bucket: 'test-1250000000', /* 必须 */
Region: 'ap-guangzhou', /* 必须 */
}, function(err, data) {
console.log(err || data);
});
```

参数说明

参数名	参数描述	类型	必填
Bucket	Bucket 的名称。命名规则为{name}-{appid} ，此处填写的存储桶名称必须为此格式	String	是
Region	Bucket 所在区域。	String	是

回调函数说明

```
function(err, data) { ... }
```

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误。如果请求成功，则为空字符串	Object
- statusCode	请求返回的 HTTP 状态码，如 200，403，404 等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，如 200，403，404 等	Number
- headers	请求返回的头部信息	Object

Get Bucket

功能说明

Get Bucket 请求等同于 List Object 请求，可以列出该 Bucket 下的部分或者全部 Object。此 API 调用者需要对 Bucket 有 Read 权限。

使用示例

列出目录 a 的所有文件

```
cos.getBucket({
Bucket: 'test-1250000000', /* 必须 */
Region: 'ap-guangzhou', /* 必须 */
Prefix: 'a/', /* 非必须 */
}, function(err, data) {
console.log(err || data);
});
```

列出目录 a 的文件，不深度遍历

```
cos.getBucket({
Bucket: 'test-1250000000', /* 必须 */
Region: 'ap-guangzhou' /* 必须 */
Prefix: 'a/', /* 非必须 */
Delimiter: '/', /* 非必须 */
}, function(err, data) {
console.log(err || data);
});
```

参数说明

参数名	参数描述	类型	必填
Bucket	Bucket 的名称。命名规则为{name}-{appid} ，此处填写的存储桶名称必须为此格式	String	是
Region	Bucket 所在区域。	String	是
Prefix	前缀匹配，用来规定返回的文件前缀地址	String	否
Delimiter	定界符为一个符号，如果有 Prefix，则将 Prefix 到 delimiter 之间的相同路径归为一类，定义为 Common Prefix，然后列出所有 Common Prefix。如果没有 Prefix，则从路径起点开始	String	否
Marker	默认以 UTF-8 二进制顺序列出条目，所有列出条目从 marker 开始	String	否
MaxKeys	单次返回最大的条目数量，默认1000	String	否
EncodingType	规定返回值的编码方式，可选值：url	String	否

回调函数说明

```
function(err, data) { ... }
```

回调参数说明

参数名	参数描述	类型

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误。如果请求成功，则为空	Object
- statusCode	请求返回的 HTTP 状态码，如 200，403，404 等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- headers	请求返回的头部信息	Object
- statusCode	请求返回的 HTTP 状态码，如 200，403，404 等	Number
- CommonPrefixes	将 Prefix 到 delimiter 之间的相同路径归为一类，定义为 Common Prefix	Array
- - Prefix	单条 Common 的前缀	String
- - Name	说明 Bucket 的信息	String
- Prefix	前缀匹配，用来规定返回的文件前缀地址	String
- Marker	默认以 UTF-8 二进制顺序列出条目，所有列出条目从 marker 开始	String
- MaxKeys	单次响应请求内返回结果的最大的条目数量	String
- IsTruncated	响应请求条目是否被截断，字符串，'true' 或者 'false'	String
- NextMarker	假如返回条目被截断，则返回NextMarker就是下一个条目的起点	String
- Encoding-Type	返回值的编码方式，作用于Delimiter，Marker，Prefix，NextMarker，Key	String
- Contents	元数据信息	Array
- - ETag	文件的 MD-5 算法校验值，如 `"22ca88419e2ed4721c23807c678adbe4c08a7880"`， ** 注意前后携带双引号 **	String
- - Size	说明文件大小，单位是 Byte	String
- - Key	Object名称	String
- - LastModified	说明 Object 最后被修改时间，如 2017-06-23T12:33:27.000Z	String
- - Owner	Bucket 持有者信息	Object
- ID	Bucket 的 AppID	String
- StorageClass	Object 的存储级别，枚举值：STANDARD，STANDARD_IA，NEARLINE	String

Delete Bucket

功能说明

Delete Bucket 接口请求可以在指定账号下删除 Bucket，删除之前要求 Bucket 内的内容为空，只有删除了 Bucket 内的信息，才能删除 Bucket 本身。注意，如果删除成功，则返回的 HTTP 状态码为 200 或 204 。

使用示例

调用 Delete Bucket 操作：

```
cos.deleteBucket({
Bucket: 'test-1250000000', /* 必须 */
Region: 'ap-guangzhou' /* 必须 */
}, function(err, data) {
console.log(err || data);
});
```

参数说明

参数名	参数描述	类型	必填
Bucket	Bucket 的名称。命名规则为{name}-{appid} ，此处填写的存储桶名称必须为此格式	String	是

参数名	参数描述	类型	必填
Region	Bucket 所在区域。	String	是

回调函数说明

```
function(err, data) { ... }
```

回调参数说明

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误。如果请求成功，则为空	Object
- statusCode	请求返回的 HTTP 状态码，如 200，403，404 等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，如 200，403，404 等	Number
- headers	请求返回的头部信息	Object

Get Bucket ACL

功能说明

Get Bucket ACL 接口用来获取 Bucket 的 ACL(access control list)， 即存储桶（Bucket）的访问权限控制列表。 此 API 接口只有 Bucket 的持有者有权限操作。

使用示例

```
cos.getBucketAcl({
Bucket: 'test-1250000000', /* 必须 */
Region: 'ap-guangzhou' /* 必须 */
}, function(err, data) {
console.log(err || data);
});
```

参数说明

参数名	参数描述	类型	必填
Bucket	Bucket 的名称。命名规则为{name}-{appid} ，此处填写的存储桶名称必须为此格式	String	是
Region	Bucket 所在区域。	String	是

回调函数说明

```
function(err, data) { ... }
```

回调参数说明

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误。如果请求成功，则为空	Object
- statusCode	请求返回的 HTTP 状态码，如 200，403，404 等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，如 200，403，404 等	Number
- headers	请求返回的头部信息	Object
- Owner	Bucket 持有者信息	Object

参数名	参数描述	类型
- - DisplayName	Bucket 持有者的名称	String
- - ID	Bucket 持有者 ID， 格式：qcs::cam::uin/:uin/ 如果是根帐号， 和 是同一个值	String
- Grants	被授权者信息与权限信息列表	Array
- - Permission	指明授予被授权者的权限信息，枚举值：READ，WRITE，FULL_CONTROL	String
- - Grantee	说明被授权者的信息。type 类型可以为 RootAccount， Subaccount； 当 type 类型为 RootAccount 时，ID 中指定的是根帐号； 当 type 类型为 Subaccount 时，ID 中指定的是子帐号	Object
- - - DisplayName	用户的名称	String
- - - ID	用户的 ID， 如果是根帐号，格式为：qcs::cam::uin/:uin/ 或 qcs::cam::anyone:anyone（指代所有用户） 如果是子帐号，格式为： qcs::cam::uin/:uin/	String

Put Bucket ACL

功能说明

Put Bucket ACL 接口用来写入 Bucket 的 ACL 表，您可以通过 Header："x-cos-acl"，"x-cos-grant-read"，"x-cos-grant-write"，"x-cos-grant-full-control" 传入 ACL 信息，或者通过 Body 以 XML 格式传入 ACL 信息。

使用示例

设置 Bucket 公有读

```
cos.putBucketAcl({
Bucket: 'test-1250000000', /* 必须 */
Region: 'ap-guangzhou', /* 必须 */
ACL: 'public-read'
}, function(err, data) {
console.log(err || data);
});
```

为某个用户赋予 Bucket 读写权限

```
cos.putBucketAcl({
Bucket: 'test-1250000000', /* 必须 */
Region: 'ap-guangzhou', /* 必须 */
GrantFullControl: 'id="qcs::cam::uin/1001:uin/1001",id="qcs::cam::uin/1002:uin/1002"' // 1001 是 uin
}, function(err, data) {
console.log(err || data);
});
```

为某个用户赋予 Bucket 读写权限

```
cos.putBucketAcl({
Bucket: 'test-1250000000', /* 必须 */
Region: 'ap-guangzhou', /* 必须 */
GrantFullControl: 'id="qcs::cam::uin/1001:uin/1001",id="qcs::cam::uin/1002:uin/1002"' // 1001 是 uin
}, function(err, data) {
console.log(err || data);
});
```

通过 AccessControlPolicy 修改 Bucket 权限

```
cos.putBucketAcl({
Bucket: 'test-1250000000', /* 必须 */
Region: 'ap-guangzhou', /* 必须 */
AccessControlPolicy: {
"Owner": { // AccessControlPolicy 里必须有 owner
"ID": 'qcs::cam::uin/459000000:uin/459000000' // 459000000 是 Bucket 所属用户的 QQ 号
```

```
},
"Grants": [{
"Grantee": {
"ID": "qcs::cam::uin/10002:uin/10002", // 10002 是 QQ 号
},
"Permission": "WRITE"
}]
}
}, function(err, data) {
console.log(err || data);
});
```

参数说明

参数名	参数描述	类型	必填
Bucket	Bucket 的名称。命名规则为{name}-{appid} ，此处填写的存储桶名称必须为此格式	String	是
Region	Bucket 所在区域。	String	是
ACL	定义 Object 的 ACL 属性。有效值：private，public-read-write，public-read；默认值：private	String	否
GrantRead	赋予被授权者读的权限。格式：id=" ",id=" "； 当需要给子账户授权时，id="qcs::cam::uin/:uin/"， 当需要给根账户授权时，id="qcs::cam::uin/:uin/"， 例如：'id="qcs::cam::uin/123:uin/123", id="qcs::cam::uin/123:uin/456"'	String	否
GrantWrite	赋予被授权者写的权限。格式：id=" ",id=" "； 当需要给子账户授权时，id="qcs::cam::uin/:uin/"， 当需要给根账户授权时，id="qcs::cam::uin/:uin/"， 例如：'id="qcs::cam::uin/123:uin/123", id="qcs::cam::uin/123:uin/456"'	String	否
GrantFullControl	赋予被授权者读写权限。格式：id=" ",id=" "； 当需要给子账户授权时，id="qcs::cam::uin/:uin/"， 当需要给根账户授权时，id="qcs::cam::uin/:uin/"， 例如：'id="qcs::cam::uin/123:uin/123", id="qcs::cam::uin/123:uin/456"'	String	否
AccessControlPolicy	说明跨域资源共享配置的所有信息列表	Object	否
- Owner	代表存储桶所有者的对象	Object	否
- - ID	代表用户 ID 的字符串，格式如 qcs::cam::uin/1001:uin/1001，1001 是 uin	Object	否
- Grants	说明跨域资源共享配置的所有信息列表	Object	否
- - Permission	说明跨域资源共享配置的所有信息列表，可选项 READ、WRITE、FULL_CONTROL、READ_ACP、WRITE_ACP	String	否
- - Grantee	说明跨域资源共享配置的所有信息列表	Array	否
- - - ID	代表用户 ID 的字符串，格式如 qcs::cam::uin/1001:uin/1001，1001 是 uin	String	否
- - - DisplayName	代表用户名称的字符串，一般填写成和 ID 一致的字符串	String	否

回调函数说明

```
function(err, data) { ... }
```

回调参数说明

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误。如果请求成功，则为空	Object
- statusCode	请求返回的 HTTP 状态码，如 200，403，404 等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，如 200，403，404 等	Number

参数名	参数描述	类型
- headers	请求返回的头部信息	Object

Get Bucket CORS

功能说明

Get Bucket CORS 接口实现 Bucket 持有者在 Bucket 上进行跨域资源共享的信息配置。（CORS 是一个 W3C 标准，全称是"跨域资源共享"（Cross-origin Resource Sharing））。默认情况下，Bucket 的持有者直接有权限使用该 API 接口，Bucket 持有者也可以将权限授予其他用户。

使用示例

调用 Get Bucket CORS 操作：

```
cos.getBucketCors({
Bucket: 'test-1250000000', /* 必须 */
Region: 'ap-guangzhou', /* 必须 */
}, function(err, data) {
console.log(err || data);
});
```

参数说明

参数名	参数描述	类型	必填
Bucket	Bucket 的名称。命名规则为{name}-{appid} ，此处填写的存储桶名称必须为此格式	String	是
Region	Bucket 所在区域。	String	是

回调函数说明

```
function(err, data) { ... }
```

回调参数说明

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误。如果请求成功，则为空	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- CORSRules	说明跨域资源共享配置的所有信息列表	Array
- - AllowedMethods	允许的 HTTP 操作，枚举值：GET，PUT，HEAD，POST，DELETE	Array
- - AllowedOrigins	允许的访问来源，支持通配符 * 格式为：协议://域名[:端口]如： `http://imgcache.finance.cloud.tencent.com:80www.qq.com`	Array
- - AllowedHeaders	在发送 OPTIONS 请求时告知服务端，接下来的请求可以使用哪些自定义的 HTTP 请求头部，支持通配符 *	Array
- - ExposeHeaders	设置浏览器可以接收到的来自服务器端的自定义头部信息	Array
- - MaxAgeSeconds	设置 OPTIONS 请求得到结果的有效期	String
- - ID	配置规则的 ID	String

Put Bucket CORS

> **注意：**
>
> 1. 如果要在前端修改 `跨域访问配置` ，需要该 Bucket 本身支持跨域，可以在 `控制台` 进行 `跨域访问配置` 。
> 2. 在修改 `跨域访问配置` 时，请注意不要影响到当前的 Origin 下的跨域请求。

功能说明

Put Bucket CORS 接口用来请求设置 Bucket 的跨域资源共享权限，您可以通过传入 XML 格式的配置文件来实现配置，文件大小限制为64 KB。默认情况下，Bucket 的持有者直接有权限使用该 API 接口，Bucket 持有者也可以将权限授予其他用户。

使用示例

调用 Put Bucket CORS 操作：

```
cos.putBucketCors({
Bucket: 'test-1250000000', /* 必须 */
Region: 'ap-guangzhou', /* 必须 */
CORSRules: [{
"AllowedOrigin": ["*"],
"AllowedMethod": ["GET", "POST", "PUT", "DELETE", "HEAD"],
"AllowedHeader": ["*"],
"ExposeHeader": ["ETag", "x-cos-acl", "x-cos-version-id", "x-cos-delete-marker", "x-cos-server-side-encryption"],
"MaxAgeSeconds": "5"
}]
}, function(err, data) {
console.log(err || data);
});
```

参数说明

参数名	参数描述	类型	必填
Bucket	Bucket 的名称。命名规则为{name}-{appid} ，此处填写的存储桶名称必须为此格式	String	是
Region	Bucket 所在区域。	String	是
CORSRules	说明跨域资源共享配置的所有信息列表	Array	否
- ID	配置规则的 ID，可选填	String	否
- AllowedMethods	允许的 HTTP 操作，枚举值：GET，PUT，HEAD，POST，DELETE	Array	是
- AllowedOrigins	允许的访问来源，支持通配符 * 格式为：协议://域名[:端口]如：`http://imgcache.finance.cloud.tencent.com:80www.qq.com`	Array	是
- AllowedHeaders	在发送 OPTIONS 请求时告知服务端，接下来的请求可以使用哪些自定义的 HTTP 请求头部，**暂不支持通配符 "*"**	Array	否
- ExposeHeaders	设置浏览器可以接收到的来自服务器端的自定义头部信息	Array	否
- MaxAgeSeconds	设置 OPTIONS 请求得到结果的有效期	String	否

回调函数说明

```
function(err, data) { ... }
```

回调参数说明

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误。如果请求成功，则为空	Object
- statusCode	请求返回的 HTTP 状态码，如 200，403，404 等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，如 200，403，404 等	Number
- headers	请求返回的头部信息	Object

Delete Bucket CORS

> **注意：**
>
> 1. 删除当前 Bucket 的 `跨域访问配置` 信息，会导致所有请求跨域失败，请谨慎操作。
> 2. 不推荐在浏览器端使用该方法。

功能说明

Delete Bucket CORS 接口请求实现删除跨域访问配置信息。

使用示例

调用 Delete Bucket CORS 操作：

```
cos.deleteBucketCors({
Bucket: 'test-1250000000', /* 必须 */
Region: 'ap-guangzhou', /* 必须 */
}, function(err, data) {
console.log(err || data);
});
```

参数说明

参数名	参数描述	类型	必填
Bucket	Bucket 的名称。命名规则为{name}-{appid} ，此处填写的存储桶名称必须为此格式	String	是
Region	Bucket 所在区域。	String	是

回调函数说明

```
function(err, data) { ... }
```

回调参数说明

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误。如果请求成功，则为空	Object
- statusCode	请求返回的 HTTP 状态码，如 200，403，404 等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，如 200，403，404 等	Number
- headers	请求返回的头部信息	Object

Get Bucket Location

功能说明

Get Bucket Location 接口用于获取 Bucket 所在的地域信息，该 GET 操作使用 location 子资源返回 Bucket 所在的区域，只有 Bucket 持有者才有该 API 接口的操作权限。

使用示例

调用 Get Bucket Location 操作：

```
cos.getBucketLocation({
Bucket: 'test-1250000000', /* 必须 */
Region: 'ap-guangzhou', /* 必须 */
}, function(err, data) {
console.log(err || data);
});
```

参数说明

参数名	参数描述	类型	必填
Bucket	Bucket 的名称。命名规则为{name}-{appid} ，此处填写的存储桶名称必须为此格式	String	是
Region	Bucket 所在区域。	String	是

回调函数说明

```
function(err, data) { ... }
```

回调参数说明

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误。如果请求成功，则为空	Object
- statusCode	请求返回的 HTTP 状态码，如 200，403，404 等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，如 200，403，404 等	Number
- headers	请求返回的头部信息	Object
- LocationConstraint	Bucket 所在区域。	String

Object 操作

Head Object

功能说明

Head Object 接口请求可以获取对应 Object 的 meta 信息数据，Head 的权限与 Get 的权限一致。

使用示例

调用 Head Object 操作：

```
cos.headObject({
Bucket: 'test-1250000000', /* 必须 */
Region: 'ap-guangzhou', /* 必须 */
Key: '1.jpg', /* 必须 */
}, function(err, data) {
console.log(err || data);
});
```

参数说明

参数名	参数描述	类型	必填
Bucket	Bucket 的名称。命名规则为{name}-{appid} ，此处填写的存储桶名称必须为此格式	String	是
Region	Bucket 所在区域。	String	是
Key	文件名称	String	是
IfModifiedSince	当 Object 在指定时间后被修改，则返回对应 Object 的 meta 信息，否则返回 304	String	否

回调函数说明

```
function(err, data) { ... }
```

回调参数说明

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误。如果请求成功，则为空	Object
- statusCode	请求返回的 HTTP 状态码，如 200，403，404 等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，如 200，304 等，如果在指定时间后未被修改，则返回 304	Number
- headers	请求返回的头部信息	Object

参数名	参数描述	类型
- x-cos-object-type	用来表示 Object 是否可以被追加上传，枚举值：normal 或者 appendable	String
- x-cos-storage-class	Object 的存储级别，枚举值：STANDARD, STANDARD_IA, NEARLINE	String
- x-cos-meta- *	用户自定义的 meta	String
- NotModified	Object 是否在指定时间后未被修改	Boolean

Get Object

功能说明

Get Object 接口请求可以在 COS 的 Bucket 中将一个文件（Object）下载至本地。该操作需要请求者对目标 Object 具有读权限或目标 Object 对所有人都开放了读权限（公有读）。

使用示例

调用 Get Object 操作：

```
cos.getObject({
Bucket: 'test-1250000000', /* 必须 */
Region: 'ap-guangzhou', /* 必须 */
Key: '1.jpg', /* 必须 */
}, function(err, data) {
console.log(err || data.Body);
});
```

指定 Range 获取文件内容

```
cos.getObject({
Bucket: 'test-1250000000', /* 必须 */
Region: 'ap-guangzhou', /* 必须 */
Key: '1.jpg', /* 必须 */
Range: 'bytes=1-3', /* 非必须 */
}, function(err, data) {
console.log(err || data.Body);
});
```

参数说明

参数名	参数描述	类型	必填
Bucket	Bucket 的名称。命名规则为{name}-{appid} ，此处填写的存储桶名称必须为此格式	String	是
Region	Bucket 所在区域。	String	是
Key	文件名称	String	是
ResponseContentType	设置响应头部中的 Content-Type 参数	String	否
ResponseContentLanguage	设置返回头部中的 Content-Language 参数	String	否
ResponseExpires	设置返回头部中的 Content-Expires 参数	String	否
ResponseCacheControl	设置返回头部中的 Cache-Control 参数	String	否
ResponseContentDisposition	设置返回头部中的 Content-Disposition 参数	String	否
ResponseContentEncoding	设置返回头部中的 Content-Encoding 参数	String	否
Range	RFC 2616 中定义的指定文件下载范围，以字节（bytes）为单位，如 Renge: 'bytes=1-3'	String	否
IfModifiedSince	当Object在指定时间后被修改，则返回对应 Object meta 信息，否则返回 304	String	否
IfUnmodifiedSince	如果文件修改时间早于或等于指定时间，才返回文件内容。否则返回 412 (precondition failed)	String	否
IfMatch	当 ETag 与指定的内容一致，才返回文件。否则返回 412（precondition failed）	String	否
IfNoneMatch	当 ETag 与指定的内容不一致，才返回文件。否则返回 304（not modified）	String	否

回调函数说明

```
function(err, data) { ... }
```

回调参数说明

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误。如果请求成功，则为空	Object
- statusCode	请求返回的 HTTP 状态码，如 200，403，404 等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，如 200，304，403，404 等	Number
- headers	请求返回的头部信息	Object
- x-cos-object-type	用来表示 object 是否可以被追加上传，枚举值：normal 或者 appendable	String
- x-cos-storage-class	Object 的存储级别，枚举值：STANDARD，STANDARD_IA，NEARLINE， **注意：如果没有返回该头部，则说明文件存储级别为 STANDARD（标准存储）**	String
- x-cos-meta- *	用户自定义的元数据	String
- NotModified	如果请求时带有 IfModifiedSince 则返回该属性，如果文件未被修改，则为 true，否则为 false	Boolean
- Body	返回的文件内容，默认为 String 形式	String

Put Object

功能说明

Put Object 接口请求可以将本地的文件（Object）上传至指定 Bucket 中。该操作需要请求者对 Bucket 有 WRITE 权限。

> **注意：**
>
> 1. Key（文件名）不能以 `/` 结尾，否则会被识别为文件夹。
> 2. 单个 Bucket 下 ACL 策略限制 1000 条，因此在单个 Bucket 下，最多允许对 999 个文件设置 ACL 权限。

使用示例

调用 Put Object 上传文件：

```
cos.putObject({
Bucket: 'test-1250000000', /* 必须 */
Region: 'ap-guangzhou', /* 必须 */
Key: '1.jpg', /* 必须 */
StorageClass: 'STANDARD',
Body: file, // 上传文件对象
onProgress: function(progressData) {
console.log(JSON.stringify(progressData));
}
}, function(err, data) {
console.log(err || data);
});
```

上传字符串作为文件内容：

```
cos.putObject({
Bucket: 'test-1250000000', /* 必须 */
Region: 'ap-guangzhou', /* 必须 */
Key: '1.jpg', /* 必须 */
Body: 'hello!',
}, function(err, data) {
console.log(err || data);
});
```

上传字符串作为文件内容：

```
cos.putObject({
Bucket: 'test-1250000000', /* 必须 */
Region: 'ap-guangzhou', /* 必须 */
Key: '1.jpg', /* 必须 */
Body: 'hello!',
}, function(err, data) {
console.log(err || data);
});
```

创建目录：

```
cos.putObject({
Bucket: 'test-1250000000', /* 必须 */
Region: 'ap-guangzhou', /* 必须 */
Key: 'a/', /* 必须 */
Body: '',
}, function(err, data) {
console.log(err || data);
});
```

参数说明

参数名	参数描述	类型	必填
Bucket	Bucket 的名称。命名规则为{name}-{appid} ，此处填写的存储桶名称必须为此格式	String	是
Region	Bucket 所在区域。	String	是
Key	文件名称	String	是
CacheControl	RFC 2616 中定义的缓存策略，将作为 Object 元数据保存	String	否
ContentDisposition	RFC 2616 中定义的文件名称，将作为 Object 元数据保存	String	否
ContentEncoding	RFC 2616 中定义的编码格式，将作为 Object 元数据保存	String	否
ContentLength	RFC 2616 中定义的 HTTP 请求内容长度（字节）	String	否
ContentType	RFC 2616 中定义的内容类型（MIME），将作为 Object 元数据保存	String	否
Expect	当使用 Expect: 100-continue 时，在收到服务端确认后，才会发送请求内容	String	否
Expires	RFC 2616 中定义的过期时间，将作为 Object 元数据保存	String	否
ACL	定义 Object 的 ACL 属性。有效值：private，public-read-write，public-read；默认值：private	String	否
GrantRead	赋予被授权者读的权限。格式：id=" ",id=" "； 当需要给子账户授权时，id="qcs::cam::uin/:uin/"， 当需要给根账户授权时，id="qcs::cam::uin/:uin/"， 例如：'id="qcs::cam::uin/123:uin/123", id="qcs::cam::uin/123:uin/456"'	String	否
GrantWrite	赋予被授权者写的权限。格式：id=" ",id=" "； 当需要给子账户授权时，id="qcs::cam::uin/:uin/"， 当需要给根账户授权时，id="qcs::cam::uin/:uin/"， 例如：'id="qcs::cam::uin/123:uin/123", id="qcs::cam::uin/123:uin/456"'	String	否
GrantFullControl	赋予被授权者读写权限。格式：id=" ",id=" "； 当需要给子账户授权时，id="qcs::cam::uin/:uin/"， 当需要给根账户授权时，id="qcs::cam::uin/:uin/"， 例如：'id="qcs::cam::uin/123:uin/123", id="qcs::cam::uin/123:uin/456"'	String	否
StorageClass	设置 Object 的存储级别，枚举值：STANDARD，STANDARD_IA，NEARLINE，默认值：STANDARD	String	否
x-cos-meta- *	允许用户自定义的头部信息，将作为 Object 元数据返回。大小限制 2K	String	否
Body	上传文件的内容，可以为`字符串`，`File 对象`或者 `Blob 对象`	String \ File\ Blob	是
onProgress	进度的回调函数，进度回调响应对象（progressData）属性如下	Function	否
progressData.loaded	已经下载的文件部分大小，以字节（bytes）为单位	Number	否

参数名	参数描述	类型	必填
progressData.total	整个文件的大小，以字节（Bytes）为单位	Number	否
progressData.speed	文件的下载速度，以字节/秒（Bytes/s）为单位	Number	否
progressData.percent	文件下载的百分比，以小数形式呈现，例如：下载 50% 即为 0.5	Number	否

回调函数说明

```
function(err, data) { ... }
```

回调参数说明

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误。如果请求成功，则为空	Object
- statusCode	请求返回的 HTTP 状态码，如 200，403，404 等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，如 200，403，404 等	Number
- headers	请求返回的头部信息	Object
- ETag	返回文件的 MD5 算法校验值。ETag 的值可以用于检查 Object 在上传过程中是否有损坏， **注意：这里的 ETag 值字符串前后带有双引号，例如** `"09cba091df696af91549de27b8e7d0f6"`	String

Delete Object

功能说明

Delete Object 接口请求可以在 COS 的 Bucket 中将一个文件（Object）删除。该操作需要请求者对 Bucket 有 WRITE 权限。

使用示例

调用 Delete Object 操作：

```
cos.deleteObject({
Bucket: 'test-1250000000', /* 必须 */
Region: 'ap-guangzhou', /* 必须 */
Key: '1.jpg' /* 必须 */
}, function(err, data) {
console.log(err || data);
});
```

参数说明

参数名	参数描述	类型	必填
Bucket	Bucket 的名称。命名规则为{name}-{appid} ，此处填写的存储桶名称必须为此格式	String	是
Region	Bucket 所在区域。	String	是
Key	文件名称	String	是

回调函数说明

```
function(err, data) { ... }
```

回调参数说明

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误。如果请求成功，则为空	Object

参数名	参数描述	类型
- statusCode	请求返回的 HTTP 状态码，如 200，403，404 等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，如 200，204，403，404 等，**如果删除成功或者文件不存在则返回 204 或 200，如果找不到指定的 Bucket，则返回 404**	Number
- headers	请求返回的头部信息	Object

Options Object

功能说明

Options Object 接口实现 Object 跨域访问配置的预请求。即在发送跨域请求之前会发送一个 OPTIONS 请求并带上特定的来源域，HTTP 方法和 HEADER 信息等给 COS，以决定是否可以发送真正的跨域请求。当 CORS 配置不存在时，请求返回 403 Forbidden。 **可以通过 Put Bucket CORS 接口来开启 Bucket 的 CORS 支持。**

使用示例

调用 Options Object 操作：

```
cos.optionsObject({
Bucket: 'test-1250000000', /* 必须 */
Region: 'ap-guangzhou', /* 必须 */
Key: '1.jpg', /* 必须 */
Origin: 'http://imgcache.finance.cloud.tencent.com:80www.qq.com', /* 必须 */
AccessControlRequestMethod: 'PUT', /* 必须 */
AccessControlRequestHeaders: 'origin,accept,content-type' /* 非必须 */
}, function(err, data) {
console.log(err || data);
});
```

参数说明

参数名	参数描述	类型	必填
Bucket	Bucket 的名称。命名规则为 {name}-{appid} ，此处填写的存储桶名称必须为此格式	String	是
Region	Bucket 所在区域。	String	是
Key	文件名称	String	是
Origin	模拟跨域访问的请求来源域名	String	是
AccessControlRequestMethod	模拟跨域访问的请求 HTTP 方法	String	是
AccessControlRequestHeaders	模拟跨域访问的请求头部	String	否

回调函数说明

```
function(err, data) { ... }
```

回调参数说明

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误。如果请求成功，则为空	Object
- statusCode	请求返回的 HTTP 状态码，如 200，403，404 等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- headers	请求返回的头部信息	Object
- statusCode	请求返回的 HTTP 状态码，如 200，403，404 等	Number

参数名	参数描述	类型
- AccessControlAllowOrigin	模拟跨域访问的请求来源域名，中间用逗号间隔，当来源不允许的时候，此Header不返回。例如：*	String
- AccessControlAllowMethods	模拟跨域访问的请求HTTP方法，中间用逗号间隔，当请求方法不允许的时候，此Header不返回。例如：PUT，GET，POST，DELETE，HEAD	String
- AccessControlAllowHeaders	模拟跨域访问的请求头部，中间用逗号间隔，当模拟任何请求头部不允许的时候，此 Header 不返回该请求头部。例如：accept，content-type，origin，authorization	String
- AccessControlExposeHeaders	跨域支持返回头部，中间用逗号间隔。例如：ETag	String
- AccessControlMaxAge	设置 OPTIONS 请求得到结果的有效期。例如：3600	String
- OptionsForbidden	OPTIONS 请求是否被禁止，如果返回的 HTTP 状态码为 403，则为 true	Boolean

Get Object ACL

功能说明

Get Object ACL 接口用来获取某个 Bucket 下的某个 Object 的访问权限。只有 Bucket 的持有者才有权限操作。

使用示例

调用 Get Object ACL 操作：

```
cos.getObjectAcl({
Bucket: 'test-1250000000', /* 必须 */
Region: 'ap-guangzhou', /* 必须 */
Key: '1.jpg', /* 必须 */
}, function(err, data) {
console.log(err || data);
});
```

参数说明

参数名	参数描述	类型	必填
Bucket	Bucket 的名称。命名规则为 {name}-{appid} ，此处填写的存储桶名称必须为此格式	String	是
Region	Bucket 所在区域。	String	是
Key	文件名称	String	是

回调函数说明

```
function(err, data) { ... }
```

回调参数说明

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误。如果请求成功，则为空	Object
- statusCode	请求返回的 HTTP 状态码，如 200，403，404 等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，如 200，403，404 等	Number
- headers	请求返回的头部信息	Object
- Owner	标识资源的所有者	Object
- ID	Object 持有者 ID，格式：qcs::cam::uin/:uin/ 如果是根帐号， 和 是同一个值	String

参数名	参数描述	类型
- DisplayName	Object 持有者的名称	String
- Grants	被授权者信息与权限信息列表	Array
- Permission	指明授予被授权者的权限信息，枚举值：READ，WRITE，FULL_CONTROL	String
- Grantee	说明被授权者的信息。type 类型可以为 RootAccount， Subaccount；当 type 类型为 RootAccount 时，ID 中指定的是根帐号;当 type 类型为 Subaccount 时，ID 中指定的是子帐号	Object
- DisplayName	用户的名称	String
- ID	用户的 ID，如果是根帐号，格式为：qcs::cam::uin/:uin/ 或 qcs::cam::anyone:anyone （指代所有用户）如果是子帐号， 格式为： qcs::cam::uin/:uin/	String

Put Object ACL

功能说明

Put Object ACL 接口用来对某个 Bucket 中的某个的 Object 进行 ACL 表的配置 **单个 Bucket 下 ACL 策略限制 1000 条，因此在单个 Bucket 下，最多允许对 999 个文件设置 ACL 权限**

使用示例

调用 Put Object ACL 修改文件权限：

```
cos.putObjectAcl({
Bucket: 'test-1250000000', /* 必须 */
Region: 'ap-guangzhou', /* 必须 */
Key: '1.jpg', /* 必须 */
ACL: 'public-read', /* 非必须 */
}, function(err, data) {
console.log(err || data);
});
```

为某个用户赋予文件读写权限

```
cos.putObjectAcl({
Bucket: 'test-1250000000', /* 必须 */
Region: 'ap-guangzhou', /* 必须 */
Key: '1.jpg', /* 必须 */
GrantFullControl: 'id="qcs::cam::uin/1001:uin/1001",id="qcs::cam::uin/1002:uin/1002"' // 1001 是 uin
}, function(err, data) {
console.log(err || data);
});
```

通过 AccessControlPolicy 修改 Bucket 权限

```
cos.putBucketAcl({
Bucket: 'test-1250000000', /* 必须 */
Region: 'ap-guangzhou', /* 必须 */
Key: '1.jpg', /* 必须 */
AccessControlPolicy: {
"Owner": { // AccessControlPolicy 里必须有 owner
"ID": 'qcs::cam::uin/459000000:uin/459000000' // 459000000 是 Bucket 所属用户的 QQ 号
},
"Grants": [{
"Grantee": {
"ID": "qcs::cam::uin/10002:uin/10002", // 10002 是 QQ 号
},
"Permission": "WRITE"
}]
}
}, function(err, data) {
console.log(err || data);
});
```

参数说明

参数名	参数描述	类型	必填
Bucket	Bucket 的名称。命名规则为 {name}-{appid} ，此处填写的存储桶名称必须为此格式	String	是
Region	Bucket 所在区域。	String	是
Key	文件名称	String	是
ACL	定义 Object 的 ACL 属性。有效值：private，public-read-write，public-read；默认值：private	String	否
GrantRead	赋予被授权者读的权限。格式：id=" ",id=" "； 当需要给子账户授权时，id="qcs::cam::uin/:uin/"， 当需要给根账户授权时，id="qcs::cam::uin/:uin/"， 例如：'id="qcs::cam::uin/123:uin/123", id="qcs::cam::uin/123:uin/456"'	String	否
GrantWrite	赋予被授权者写的权限。格式：id=" ",id=" "； 当需要给子账户授权时，id="qcs::cam::uin/:uin/"， 当需要给根账户授权时，id="qcs::cam::uin/:uin/"， 例如：'id="qcs::cam::uin/123:uin/123", id="qcs::cam::uin/123:uin/456"'	String	否
GrantFullControl	赋予被授权者读写权限。 格式：id=" ",id=" "；当需要给子账户授权时，id="qcs::cam::uin/:uin/"， 当需要给根账户授权时，id="qcs::cam::uin/:uin/"， 例如：'id="qcs::cam::uin/123:uin/123", id="qcs::cam::uin/123:uin/456"'	String	否

回调函数说明

```
function(err, data) { ... }
```

回调参数说明

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误。如果请求成功，则为空	Object
- statusCode	请求返回的 HTTP 状态码，如 200，403，404 等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，如 200，204，403，404等，	Number
- headers	请求返回的头部信息	Object

Delete Multiple Object

功能说明

Delete Multiple Object 接口请求实现在指定 Bucket 中批量删除 Object，单次请求最大支持批量删除 1000 个 Object。对于响应结果，COS 提供 Verbose 和 Quiet 两种模式：Verbose 模式将返回每个 Object 的删除结果；Quiet 模式只返回报错的 Object 信息。

使用示例

删除多个文件：

```
cos.deleteMultipleObject({
Bucket: 'test-1250000000', /* 必须 */
Region: 'ap-guangzhou', /* 必须 */
Objects: [
{Key: '1.jpg'},
{Key: '2.zip'},
]
}, function(err, data) {
console.log(err || data);
});
```

参数说明

参数名	参数描述	类型	必填
Bucket	Bucket 的名称。命名规则为{name}-{appid} ，此处填写的存储桶名称必须为此格式	String	是
Region	Bucket 所在区域。	String	是
Key	要删除的文件名称	String	是
Quiet	布尔值，这个值决定了是否启动 Quiet 模式。值为 true 启动 Quiet 模式，值为 false 则启动 Verbose 模式，默认值为 false	Boolean	否
Objects	要删除的文件列表	Array	是

回调函数说明

```
function(err, data) { ... }
```

回调参数说明

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误。如果请求成功，则为空	Object
- statusCode	请求返回的 HTTP 状态码，如 200，204，403，404 等，	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，如 200，204，403，404 等，	Number
- headers	请求返回的头部信息	Object
- Deleted	说明本次删除的成功 Object 信息	Array
- Key	Object 的名称	String
- Error	说明本次删除的失败 Object 信息	Array
- Key	Object 的名称	String
- Code	删除失败的错误码	String
- Message	删除错误信息	String

Put Object Copy

功能说明

Put Object Copy 请求实现将一个文件从源路径复制到目标路径。建议文件大小 1MB 到 5GB，超过 5GB 的文件请使用分块上传 Upload - Copy。在拷贝的过程中，文件元属性和 ACL 可以被修改。用户可以通过该接口实现文件移动，文件重命名，修改文件属性和创建副本。

使用示例

调用 Put Object Copy 操作：

```
cos.putObjectCopy({
Bucket: 'test-1250000000', /* 必须 */
Region: 'ap-guangzhou', /* 必须 */
Key: '1.jpg', /* 必须 */
CopySource: 'test1.cos.ap-guangzhou.myqcloud.com/2.jpg', /* 必须 */
}, function(err, data) {
console.log(err || data);
});
```

参数说明

参数名	参数描述	类型	必填
Bucket	Bucket 的名称。命名规则为{name}-{appid} ，此处填写的存储桶名称必须为此格式	String	是

参数名	参数描述	类型	必填
Region	Bucket 所在区域。	String	是
Key	文件名称	String	是
CopySource	源文件 URL 路径，可以通过 versionid 子资源指定历史版本	String	是
ACL	定义 Object 的 ACL 属性。有效值：private，public-read-write，public-read；默认值：private	String	否
GrantRead	赋予被授权者读的权限。格式：id=" ",id=" "； 当需要给子账户授权时，id="qcs::cam::uin/:uin/"， 当需要给根账户授权时，id="qcs::cam::uin/:uin/"， 例如：'id="qcs::cam::uin/123:uin/123", id="qcs::cam::uin/123:uin/456"'	String	否
GrantWrite	赋予被授权者写的权限。格式：id=" ",id=" "； 当需要给子账户授权时，id="qcs::cam::uin/:uin/"， 当需要给根账户授权时，id="qcs::cam::uin/:uin/"， 例如：'id="qcs::cam::uin/123:uin/123", id="qcs::cam::uin/123:uin/456"'	String	否
GrantFullControl	赋予被授权者读写权限。格式：id=" ",id=" "； 当需要给子账户授权时，id="qcs::cam::uin/:uin/"， 当需要给根账户授权时，id="qcs::cam::uin/:uin/"， 例如：'id="qcs::cam::uin/123:uin/123", id="qcs::cam::uin/123:uin/456"'	String	否
MetadataDirective	是否拷贝元数据，枚举值：Copy, Replaced，默认值 Copy。假如标记为 Copy，忽略 Header 中的用户元数据信息直接复制；假如标记为 Replaced，按 Header 信息修改元数据。**当目标路径和原路径一致，即用户试图修改元数据时，必须为 Replaced**	String	否
CopySourceIfModifiedSince	当 Object 在指定时间后被修改，则执行操作，否则返回 412。**可与 CopySourceIfNoneMatch 一起使用，与其他条件联合使用返回冲突**	String	否
CopySourceIfUnmodifiedSince	当 Object 在指定时间后未被修改，则执行操作，否则返回 412。**可与 CopySourceIfMatch 一起使用，与其他条件联合使用返回冲突**	String	否
CopySourceIfMatch	当 Object 的 Etag 和给定一致时，则执行操作，否则返回 412。**可与CopySourceIfUnmodifiedSince 一起使用，与其他条件联合使用返回冲突**	String	否
CopySourceIfNoneMatch	当 Object 的 Etag 和给定不一致时，则执行操作，否则返回 412。**可与 CopySourceIfModifiedSince 一起使用，与其他条件联合使用返回冲突**	String	否
StorageClass	存储级别，枚举值：存储级别，枚举值：Standard, Standard_IA，Nearline；默认值：Standard	String	否
x-cos-meta- *	其他自定义的文件头部	String	否
CacheControl	指定所有缓存机制在整个请求/响应链中必须服从的指令	String	否
ContentDisposition	MIME 协议的扩展，MIME 协议指示 MIME 用户代理如何显示附加的文件	String	否
ContentEncoding	HTTP 中用来对「采用何种编码格式传输正文」进行协定的一对头部字段	String	否
ContentType	RFC 2616 中定义的 HTTP 请求内容类型（MIME），例如`text/plain`	String	否
Expect	请求的特定的服务器行为	String	否
Expires	响应过期的日期和时间	String	否

回调函数说明

```
function(err, data) { ... }
```

回调参数说明

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误。如果请求成功，则为空	Object
- statusCode	请求返回的 HTTP 状态码，如 200，403，404 等	Number
- headers	请求返回的头部信息	Object

参数名	参数描述	类型
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，如 200，403，404 等	Number
- headers	请求返回的头部信息	Object
- ETag	文件的 MD-5 算法校验值，如 `"22ca88419e2ed4721c23807c678adbe4c08a7880"`，**注意前后携带双引号**	String
- LastModified	说明 Object 最后被修改时间，如 2017-06-23T12:33:27.000Z	String

分块上传操作

Initiate Multipart Upload

功能说明

Initiate Multipart Upload 请求实现初始化分片上传，成功执行此请求以后会返回 Upload ID 用于后续的 Upload Part 请求

使用示例

调用 Initiate Multipart Upload 操作：

```
cos.multipartInit({
Bucket: 'test-1250000000', /* 必须 */
Region: 'ap-guangzhou', /* 必须 */
Key: '1.jpg', /* 必须 */
}, function(err, data) {
console.log(err || data);
});
```

参数说明

参数名	参数描述	类型	必填
Bucket	Bucket 的名称。命名规则为{name}-{appid} ，此处填写的存储桶名称必须为此格式	String	是
Region	Bucket 所在区域。	String	是
Key	文件名称	String	是
CacheControl	RFC 2616 中定义的缓存策略，将作为 Object 元数据保存	String	否
ContentDisposition	RFC 2616 中定义的文件名称，将作为 Object 元数据保存	String	否
ContentEncoding	RFC 2616 中定义的编码格式，将作为 Object 元数据保存	String	否
ContentType	RFC 2616 中定义的内容类型（MIME），将作为 Object 元数据保存	String	否
Expires	RFC 2616 中定义的过期时间，将作为 Object 元数据保存	String	否
ACL	定义 Object 的 ACL 属性。有效值：private，public-read-write，public-read；默认值：private	String	否
GrantRead	赋予被授权者读的权限。格式：id=" ",id=" "； 当需要给子账户授权时，id="qcs::cam::uin/:uin/"， 当需要给根账户授权时，id="qcs::cam::uin/:uin/"， 例如：'id="qcs::cam::uin/123:uin/123", id="qcs::cam::uin/123:uin/456"'	String	否
GrantWrite	赋予被授权者写的权限。格式：id=" ",id=" "； 当需要给子账户授权时，id="qcs::cam::uin/:uin/"， 当需要给根账户授权时，id="qcs::cam::uin/:uin/"， 例如：'id="qcs::cam::uin/123:uin/123", id="qcs::cam::uin/123:uin/456"'	String	否
GrantFullControl	赋予被授权者读写权限。格式：id=" ",id=" "； 当需要给子账户授权时，id="qcs::cam::uin/:uin/"， 当需要给根账户授权时，id="qcs::cam::uin/:uin/"， 例如：'id="qcs::cam::uin/123:uin/123", id="qcs::cam::uin/123:uin/456"'	String	否
StorageClass	设置Object的存储级别，枚举值：STANDARD，STANDARD_IA，NEARLINE，默认值：STANDARD	String	否

参数名	参数描述	类型	必填
x-cos-meta- *	允许用户自定义的头部信息，将作为 Object 元数据返回。大小限制2K	String	否

回调函数说明

```
function(err, data) { ... }
```

回调参数说明

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误。如果请求成功，则为空	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
Bucket	分片上传的目标 Bucket	String
Key	Object 的名称	String
UploadId	在后续上传中使用的 ID	String

Upload Part

功能说明

Upload Part 接口请求实现在初始化以后的分块上传，支持的块的数量为 1 到 10000，块的大小为 1MB 到 5GB。 使用 Initiate Multipart Upload 接口初始化分片上传时会得到一个 uploadId，该 ID 不但唯一标识这一分块数据，也标识了这分块数据在整个文件内的相对位置。在每次请求 Upload Part 时候，需要携带 partNumber 和 uploadId，partNumber 为块的编号，支持乱序上传。 当传入 uploadId 和 partNumber 都相同的时候，后传入的块将覆盖之前传入的块。当 uploadId 不存在时会返回 404 错误，NoSuchUpload。

使用示例

调用 Upload Part 操作：

```
cos.multipartUpload({
Bucket: 'test-1250000000', /* 必须 */
Region: 'ap-guangzhou', /* 必须 */
Key: '1.jpg', /* 必须 */
}, function(err, data) {
console.log(err || data);
});
```

参数说明

参数名	参数描述	类型	必填
Bucket	Bucket 的名称。命名规则为 {name}-{appid} ，此处填写的存储桶名称必须为此格式	String	是
Region	Bucket 所在区域。	String	是
Key	文件名称	String	是
ContentLength	RFC 2616 中定义的 HTTP 请求内容长度（字节）	String	是
PartNumber	分块的编号	String	是
UploadId	上传任务编号	String	是
Body	上传文件分块的内容，可以为`字符串`，`File 对象`或者 `Blob 对象`	String \ File \ Blob	是
Expect	当使用 `Expect: 100-continue` 时，在收到服务端确认后，才会发送请求内容	String	否
ContentMD5	RFC 1864 中定义的经过 Base64 编码的128-bit 内容 MD5 校验值。此头部用来校验文件内容是否发生变化	String	否

回调函数说明

```
function(err, data) { ... }
```

回调参数说明

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误。如果请求成功，则为空	Object
- statusCode	请求返回的 HTTP 状态码，如 200，403，404 等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，如 200，403，404 等	Number
- headers	请求返回的头部信息	Object
- ETag	文件的 MD-5 算法校验值，如 `"22ca88419e2ed4721c23807c678adbe4c08a7880"`，**注意前后携带双引号**	String

Complete Multipart Upload

功能说明

Complete Multipart Upload 接口请求用来实现完成整个分块上传。当使用 Upload Parts 上传完所有块以后，必须调用该 API 来完成整个文件的分块上传。在使用该 API 时，您必须在请求 Body 中给出每一个块的 PartNumber 和 ETag，用来校验块的准确性。 由于分块上传完后需要合并，而合并需要数分钟时间，因而当合并分块开始的时候，COS 就立即返回 200 的状态码，在合并的过程中，COS 会周期性的返回空格信息来保持连接活跃，直到合并完成，COS 会在 Body 中返回合并后块的内容。

- 当上传块小于 1 MB 的时候，在调用该 API 时，会返回 400 EntityTooSmall；
- 当上传块编号不连续的时候，在调用该 API 时，会返回 400 InvalidPart；
- 当请求 Body 中的块信息没有按序号从小到大排列的时候，在调用该 API 时，会返回 400 InvalidPartOrder；
- 当 UploadId 不存在的时候，在调用该 API 时，会返回 404 NoSuchUpload。

使用示例

调用 Complete Multipart Upload 操作：

```
cos.multipartComplete({
Bucket: 'test-1250000000', /* 必须 */
Region: 'ap-guangzhou', /* 必须 */
Key: '1.zip', /* 必须 */
UploadId: '1521389146c60e7e198202e4e6670c5c78ea5d1c60ad62f1862f47294ec0fb8c6b7f3528a2', /* 必须 */
Parts: [
{PartNumber: '1', ETag: '"0cce40bdbaf2fa0ff204c20fc965dd3f"'},
]
}, function(err, data) {
console.log(err || data);
});
```

参数说明

参数名	参数描述	类型	必填
Bucket	Bucket 的名称。命名规则为{name}-{appid} ，此处填写的存储桶名称必须为此格式	String	是
Region	Bucket 所在区域。	String	是
Key	文件名称	String	是
UploadId	上传任务编号	String	是
Parts	用来说明本次分块上传中块的信息列表	Array	是
PartNumber	分块的编号	String	是
ETag	每个块文件的 MD5 算法校验值，如 `"22ca88419e2ed4721c23807c678adbe4c08a7880"`，**注意前后携带双引号**	String	是

回调函数说明

```
function(err, data) { ... }
```

回调参数说明

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误。如果请求成功，则为空	Object
- statusCode	请求返回的 HTTP 状态码，如 200，403，404 等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，如 200，403，404 等	Number
- headers	请求返回的头部信息	Object
- Location	创建的 Object 的外网访问域名	String
- Bucket	分块上传的目标 Bucket	String
- Key	Object 的名称	String
- ETag	合并后文件的 MD5 算法校验值，如 `"22ca88419e2ed4721c23807c678adbe4c08a7880"`，**注意前后携带双引号**	String

List Parts

功能说明

List Parts 用来查询特定分块上传中的已上传的块，即罗列出指定 UploadId 所属的所有已上传成功的分块。

使用示例

调用 List Parts 操作：

```
cos.multipartListPart({
Bucket: 'test-1250000000', /* 必须 */
Region: 'ap-guangzhou', /* 必须 */
Key: '1.jpg', /* 必须 */
UploadId: '1521389146c60e7e198202e4e6670c5c78ea5d1c60ad62f1862f47294ec0fb8c6b7f3528a2', /* 必须 */
}, function(err, data) {
console.log(err || data);
});
```

参数说明

参数名	参数描述	类型	必填
Bucket	Bucket 的名称。命名规则为{name}-{appid} ，此处填写的存储桶名称必须为此格式	String	是
Region	Bucket 所在区域。	String	是
Key	文件名称	String	是
UploadId	标识本次分块上传的 ID。使用 Initiate Multipart Upload 接口初始化分片上传时会得到一个 uploadId，该 ID 不但唯一标识这一分块数据，也标识了这分块数据在整个文件内的相对位置。	String	是
EncodingType	规定返回值的编码方式	String	否
MaxParts	单次返回最大的条目数量，默认 1000	String	否
PartNumberMarker	默认以 UTF-8 二进制顺序列出条目，所有列出条目从 marker 开始	String	否

回调函数说明

```
function(err, data) { ... }
```

回调参数说明

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误。如果请求成功，则为空	Object
- statusCode	请求返回的 HTTP 状态码，如 200，403，404 等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，如 200，403，404 等	Number
- headers	请求返回的头部信息	Object
- Bucket	分块上传的目标 Bucket	String
- Encoding-type	规定返回值的编码方式	String
- Key	Object 的名称	String
- UploadId	标识本次分块上传的 ID	String
- Initiator	用来表示本次上传发起者的信息	Object
- - DisplayName	上传发起者的名称	String
- - ID	上传发起者 ID，格式：qcs::cam::uin/:uin/ 如果是根帐号， 和 是同一个值	String
- Owner	用来表示这些分块所有者的信息	Object
- - DisplayName	Bucket 持有者的名称	String
- - ID	Bucket 持有者 ID，一般为用户的 UIN	String
- StorageClass	用来表示这些分块的存储级别，枚举值：Standard，Standard_IA，Nearline	String
- PartNumberMarker	默认以 UTF-8 二进制顺序列出条目，所有列出条目从 marker 开始	String
- NextPartNumberMarker	假如返回条目被截断，则返回 NextMarker 就是下一个条目的起点	String
- MaxParts	单次返回最大的条目数量	String
- IsTruncated	返回条目是否被截断，'true' 或者 'false'	String
- Part	分块信息列表	Array
- - PartNumber	块的编号	String
- - LastModified	块最后修改时间	String
- - ETag	块的 MD5 算法校验值	String
- - Size	块大小，单位 Byte	String

Abort Multipart Upload

功能说明

Abort Multipart Upload 用来实现舍弃一个分块上传并删除已上传的块。当您调用 Abort Multipart Upload 时，如果有正在使用这个 Upload Parts 上传块的请求，则 Upload Parts 会返回失败。当该 UploadId 不存在时，会返回 404 NoSuchUpload。

建议您及时完成分块上传或者舍弃分块上传，因为已上传但是未终止的块会占用存储空间进而产生存储费用。

使用示例

调用 Abort Multipart Upload 操作：

```
cos.multipartAbort({
Bucket: 'test-1250000000', /* 必须 */
Region: 'ap-guangzhou', /* 必须 */
Key: '1.zip', /* 必须 */
UploadId: '1521389146c60e7e198202e4e6670c5c78ea5d1c60ad62f1862f47294ec0fb8c6b7f3528a2' /* 必须 */
```

```
}, function(err, data) {
console.log(err || data);
});
```

参数说明

参数名	参数描述	类型	必填
Bucket	Bucket 的名称。命名规则为 {name}-{appid} ，此处填写的存储桶名称必须为此格式	String	是
Region	Bucket 所在区域。	String	是
Key	文件名称	String	是
UploadId	标识本次分块上传的 ID。使用 Initiate Multipart Upload 接口初始化分片上传时会得到一个 uploadId，该 ID 不但唯一标识这一分块数据，也标识了这分块数据在整个文件内的相对位置	String	是

回调函数说明

```
function(err, data) { ... }
```

回调参数说明

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误。如果请求成功，则为空	Object
- statusCode	请求返回的 HTTP 状态码，如 200，403，404 等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，如 200，403，404 等	Number
- headers	请求返回的头部信息	Object

List Multipart Uploads

功能说明

List Multiparts Uploads 用来查询正在进行中的分块上传。单次最多列出 1000 个正在进行中的分块上传。

使用示例

获取前缀为 1.zip 的未完成的 UploadId 列表

```
cos.multipartList({
Bucket: 'test-1250000000', /* 必须 */
Region: 'ap-guangzhou', /* 必须 */
Prefix: '1.zip', /* 非必须 */
}, function(err, data) {
console.log(err || data);
});
```

参数说明

参数名	参数描述	类型	必填
Bucket	Bucket 的名称。命名规则为{name}-{appid} ，此处填写的存储桶名称必须为此格式	String	是
Region	Bucket 所在区域。	String	是
Delimiter	定界符为一个符号，对 Object 名字包含指定前缀且第一次出现 delimiter 字符之间的 Object 作为一组元素：common prefix。如果没有 prefix，则从路径起点开始	String	否
EncodingType	规定返回值的编码格式，合法值：url	String	否
Prefix	限定返回的 Object key 必须以 Prefix 作为前缀。注意使用 prefix 查询时，返回的 key 中仍会包含 Prefix	String	否

参数名	参数描述	类型	必填
MaxUploads	设置最大返回的 multipart 数量，合法取值从1到1000，默认1000	String	否
KeyMarker	与 upload-id-marker 一起使用， • 当 upload-id-marker 未被指定时： ObjectName 字母顺序大于 key-marker 的条目将被列出， • 当upload-id-marker被指定时： ObjectName 字母顺序大于key-marker的条目被列出， ObjectName 字母顺序等于 key-marker 且 UploadID 大于 upload-id-marker 的条目将被列出。	String	否
UploadIdMarker	与 key-marker 一起使用， • 当 key-marker 未被指定时： upload-id-marker 将被忽略， • 当 key-marker 被指定时： ObjectName字母顺序大于 key-marker 的条目被列出， ObjectName 字母顺序等于 key-marker 且 UploadID 大于 upload-id-marker 的条目将被列出。	String	否

回调函数说明

```
function(err, data) { ... }
```

回调参数说明

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误。如果请求成功，则为空	Object
- statusCode	请求返回的 HTTP 状态码，如 200，403，404 等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，如 200，403，404 等	Number
- headers	请求返回的头部信息	Object
- Bucket	分块上传的目标 Bucket	String
- Encoding-Type	规定返回值的编码格式，合法值：url	String
- KeyMarker	列出条目从该 key 值开始	String
- UploadIdMarker	列出条目从该 UploadId 值开始	String
- NextKeyMarker	假如返回条目被截断，则返回 NextKeyMarker 就是下一个条目的起点	String
- NextUploadIdMarker	假如返回条目被截断，则返回 UploadId 就是下一个条目的起点	String
- MaxUploads	设置最大返回的 multipart 数量，合法取值从 1 到 1000	String
- IsTruncated	返回条目是否被截断，'true' 或者 'false'	String
- Delimiter	定界符为一个符号，对 object 名字包含指定前缀且第一次出现 delimiter 字符之间的 object 作为一组元素：common prefix。如果没有 prefix，则从路径起点开始	String
- Prefix	限定返回的 Object key 必须以 Prefix 作为前缀。注意使用 prefix 查询时，返回的 key 中仍会包含 Prefix	String
- CommonPrefixs	将 prefix 到 delimiter 之间的相同路径归为一类，定义为 Common Prefix	Array
- Prefix	显示具体的 CommonPrefixs	String

参数名	参数描述	类型
- Upload	Upload 的信息集合	Array
- - Key	Object 的名称	String
- - UploadId	标示本次分块上传的 ID	String
- StorageClass	用来表示分块的存储级别，枚举值：STANDARD，STANDARD_IA，NEARLINE	String
- Initiator	用来表示本次上传发起者的信息	Object
- - DisplayName	上传发起者的名称	String
- - ID	上传发起者 ID，格式：qcs::cam::uin/:uin/ 如果是根帐号， 和 是同一个值	String
- Owner	用来表示这些分块所有者的信息	Object
- - DisplayName	Bucket 持有者的名称	String
- - ID	Bucket 持有者 ID，格式：qcs::cam::uin/:uin/ 如果是根帐号， 和 是同一个值	String
- Initiated	分块上传的起始时间	String

分块上传任务操作

该类方法是对上面原生方法的封装，实现了分块上传的全过程，支持并发分块上传，支持断点续传，支持上传任务的取消，暂停和重新开始等。

Slice Upload File

功能说明

Slice Upload File 可用于实现文件的分块上传。

使用示例

调用 Slice Upload File 操作：

```
cos.sliceUploadFile({
Bucket: 'test-1250000000', /* 必须 */
Region: 'ap-guangzhou', /* 必须 */
Key: '1.zip', /* 必须 */
Body: file, /* 必须 */
TaskReady: function(taskId) { /* 非必须 */
console.log(taskId);
},
onHashProgress: function (progressData) { /* 非必须 */
console.log(JSON.stringify(progressData));
},
onProgress: function (progressData) { /* 非必须 */
console.log(JSON.stringify(progressData));
}
}, function(err, data) {
console.log(err || data);
});
```

参数说明

参数名	参数描述	类型	必填
Bucket	Bucket 的名称。命名规则为{name}-{appid} ，此处填写的存储桶名称必须为此格式	String	是
Region	Bucket 所在区域。	String	是
Key	Object 名称	String	是
Body	上传文件的内容，可以为 File 对象 或者 Blob 对象	File \ Blob	是
SliceSize	分块大小	String	否

参数名	参数描述	类型	必填
AsyncLimit	分块的并发量	String	否
StorageClass	Object 的存储级别，枚举值：STANDARD，STANDARD_IA，NEARLINE	String	否
TaskReady	上传任务创建时的回调函数，返回一个 taskId，唯一标识上传任务，可用于上传任务的取消（cancelTask），停止（pauseTask）和重新开始（restartTask）	Function	否
- taskId	上传任务的编号	String	否
onHashProgress	计算文件 MD5 值的进度回调函数，回调参数为进度对象 progressData	Function	否
- progressData.loaded	已经校验的文件部分大小，以字节（bytes）为单位	Number	否
- progressData.total	整个文件的大小，以字节（bytes）为单位	Number	否
- progressData.speed	文件的校验速度，以字节/秒（bytes/s）为单位	Number	否
- progressData.percent	文件的校验百分比，以小数形式呈现，例如：下载 50% 即为 0.5	Number	否
onProgress	上传文件的进度回调函数，回调参数为进度对象 progressData	Function	否
- progressData.loaded	已经上传的文件部分大小，以字节（bytes）为单位	Number	否
- progressData.total	整个文件的大小，以字节（bytes）为单位	Number	否
- progressData.speed	文件的上传速度，以字节/秒（bytes/s）为单位	Number	否
- progressData.percent	文件的上传百分比，以小数形式呈现，例如：下载 50% 即为 0.5	Number	否

回调函数说明

```
function(err, data) { ... }
```

回调参数说明

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误。如果请求成功，则为空	Object
- statusCode	请求返回的 HTTP 状态码，如 200，403，404 等	Number
- headers	请求返回的头部信息	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
- statusCode	请求返回的 HTTP 状态码，如 200，403，404 等	Number
- headers	请求返回的头部信息	Object
- Location	创建的 Object 的外网访问域名	String
- Bucket	分块上传的目标 Bucket	String
- Key	Object的名称	String
- ETag	合并后文件的 MD5 算法校验值，如 `"22ca88419e2ed4721c23807c678adbe4c08a7880"`，**注意前后携带双引号**	String

Cancel Task

功能说明

根据 taskId 取消分块上传任务。

使用示例

调用 Cancel Task 操作。

```
var taskId = 'xxxxx'; /* 必须 */
cos.cancelTask(taskId);
```

参数说明

参数名	参数描述	类型	必填
taskId	文件上传任务的编号，在调用 sliceUploadFile 方法时，其 TaskReady 回调会返回该上传任务的 taskId	String	是

Pause Task

功能说明

根据 taskId 暂停分块上传任务。

使用示例

调用 Pause Task 操作。

```
var taskId = 'xxxxx'; /* 必须 */
cos.pauseTask(taskId);
```

参数说明

参数名	参数描述	类型	必填
taskId	文件上传任务的编号，在调用 sliceUploadFile 方法时，其 TaskReady 回调会返回该上传任务的 taskId	String	是

Restart Task

功能说明

根据 taskId 重新开始上传任务，可以用于开启用户手动停止的（调用 pauseTask 停止）或者因为上传错误而停止的上传任务。

使用示例

调用 Restart Task 操作：

```
var taskId = 'xxxxx'; /* 必须 */
cos.restartTask(taskId);
```

参数说明

参数名	参数描述	类型	必填
taskId	文件上传任务的编号，在调用 sliceUploadFile 方法时，其 TaskReady 回调会返回该上传任务的 taskId	String	是

Node.js SDK
快速入门

最近更新时间: 2025-01-15 17:01:00

开发准备

SDK 获取

对象存储服务的 XML JS SDK 资源下载地址：XML Node.js SDK 。 演示示例 Demo 下载地址：XML Node.js SDK Demo 。

npm 引入

开发前需先安装环境依赖： npm 地址

```
npm i cos-nodejs-sdk-v5 --save
```

开发环境

1. 使用 SDK 需要您的运行环境包含 nodejs 以及 npm，nodejs 版本建议 7.0 版本以上。
2. 安装好 npm 之后记得在 sdk 的解压目录 npm install 一次（安装依赖包）；
3. 到**访问管理 > 云API密钥**页面获取您的项目 SecretId 和 SecretKey。

> 关于文章中出现的 SecretId、SecretKey、Bucket 等名称的含义和获取方式请参考：COS 术语信息

快速入门

1. 到 COS 控制台创建存储桶，得到 Bucket（存储桶名称） 和 Region（地域名称）。
2. 到**访问管理 > 云API密钥**页面获取您的项目 SecretId 和 SecretKey。
3. 参照以下代码，修改 SecretId、SecretKey、Bucket、Region，测试上传文件。

```
// 引入模块
var COS = require('cos-nodejs-sdk-v5');

var region = 'REGION'; // 替换成用户的 Region
var domain = 'DOMAIN.COM'; // 替换成用户的 Domain

var endpoint = 'cos.' + region + '.' + domain;

// 创建实例
var cos = new COS({
AppId: '1250000000',
SecretId: 'AKIDxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
SecretKey: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
ServiceDomain: endpoint,
Domain: '{Bucket}.' + endpoint // 传入模板字符串
});
// 分片上传
cos.sliceUploadFile({
Bucket: 'test',
Region: 'ap-guangzhou',
Key: '1.zip',
FilePath: './1.zip'
}, function (err, data) {
console.log(err, data);
});
```

相关文档

1. 更多例子请参阅 XML Node.js SDK Demo 。

2. 完整接口文档请参阅 XML Node.js SDK 接口文档。

接口文档

最近更新时间: 2025-01-15 17:01:00

> **说明：**
>
> 关于文章中出现的 SecretId、SecretKey、Bucket 等名称的含义和获取方式请参考：COS 术语信息

Bucket操作

Head Bucket

功能说明

Head Bucket 请求可以确认是否存在该 Bucket，是否有权限访问，Head 的权限与 Read 一致。

操作方法原型

调用 Head Bucket 操作：

```
var params = {
Bucket : 'STRING_VALUE', /* 必须 */
Region : 'STRING_VALUE' /* 必须 */
};

cos.headBucket(params, function(err, data) {
if(err) {
console.log(err);
} else {
console.log(data);
}
});
```

操作参数说明

参数名	参数描述	类型	必填
Bucket	Bucket 的名称。命名规则为{name}-{appid} ，此处填写的存储桶名称必须为此格式	String	是
Region	Bucket 所在区域。	String	是

回调函数说明

```
function(err, data) { ... }
```

回调参数说明

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误。如果请求成功，则为空	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
BucketExist	Bucket 是否存在	Boolean
BucketAuth	是否拥有该 Bucket 的权限	Boolean

Get Bucket

功能说明

Get Bucket 请求等同于 List Object 请求，可以列出该 Bucket 下部分或者所有 Object，发起该请求需要拥有 Read 权限。

操作方法原型

调用 Get Bucket 操作：

```
var params = {
Bucket : 'STRING_VALUE', /* 必须 */
Region : 'STRING_VALUE', /* 必须 */
Prefix : 'STRING_VALUE', /* 非必须 */
Delimiter : 'STRING_VALUE', /* 非必须 */
Marker : 'STRING_VALUE', /* 非必须 */
MaxKeys : 'STRING_VALUE', /* 非必须 */
EncodingType : 'STRING_VALUE', /* 非必须 */
```

```
};

cos.getBucket(params, function(err, data) {
if(err) {
console.log(err);
} else {
console.log(data);
}
});
```

操作参数说明

参数名	参数描述	类型	必填
Bucket	Bucket 的名称。命名规则为{name}-{appid} ，此处填写的存储桶名称必须为此格式	String	是
Region	Bucket 所在区域。	String	是
Prefix	前缀匹配，用来规定返回的文件前缀地址	String	否
Delimiter	定界符为一个符号，如果有 Prefix，则将 Prefix 到 delimiter 之间的相同路径归为一类，定义为Common Prefix，然后列出所有 Common Prefix。如果没有 Prefix，则从路径起点开始	String	否
Marker	默认以 UTF-8 二进制顺序列出条目，所有列出条目从marker开始	String	否
MaxKeys	单次返回最大的条目数量，默认 1000	String	否
EncodingType	规定返回值的编码方式	String	否

回调函数说明

```
function(err, data) { ... }
```

回调参数说明

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误。如果请求成功，则为空	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
CommonPrefixes	将 Prefix 到 delimiter 之间的相同路径归为一类，定义为 Common Prefix	Array
Prefix	前缀名称	String
Name	Bucket 名字	String
Prefix	前缀匹配，用来规定返回的文件前缀地址	String
Marker	默认以 UTF-8 二进制顺序列出条目，所有列出条目从 marker 开	始String
MaxKeys	单次返回最大的条目数量	String
IsTruncated	返回条目是否被截断，'true' 或者 'false'	String
NextMarker	假如返回条目被截断，则返回 NextMarker 就是下一个条目的起点	String
Encoding-Type	编码类型，作用于Delimiter，Marker，Prefix，NextMarker，Key	String
Contents	元数据信息	Array
ETag	文件的 SHA-1 算法校验值	String
Size	文件大小，单位 Byte	String
Key	Object 名称	String
LastModified	Object 最后修改时间	String
Owner	Bucket 所有者信息	Object
ID	Bucket 拥有者的 UID 信息	String

Put Bucket

功能说明

Put Bucket 请求可以在指定账号下创建一个 Bucket。

操作方法原型

调用 Put Bucket 操作：

```
var params = {
Bucket : 'STRING_VALUE', /* 必须 */
Region : 'STRING_VALUE', /* 必须 */
ACL : 'STRING_VALUE', /* 非必须 */
GrantRead : 'STRING_VALUE', /* 非必须 */
GrantWrite : 'STRING_VALUE', /* 非必须 */
GrantFullControl : 'STRING_VALUE' /* 非必须 */
};

cos.putBucket(params, function(err, data) {
if(err) {
console.log(err);
} else {
console.log(data);
}
});
```

操作参数说明

参数名	参数描述	类型	必填
Bucket	Bucket 的名称。命名规则为{name}-{appid} ，此处填写的存储桶名称必须为此格式	String	是
Region	Bucket 所在区域。	String	是
ACL	允许用户自定义文件权限。有效值：private，public-read默认值：private。	String	否
GrantRead	赋予被授权者读的权限，格式 x-cos-grant-read: uin=" ",uin=" "， 当需要给子账户授权时，uin="RootAcountID/SubAccountID"， 当需要给根账户授权时，uin="RootAcountID"。	String	否
GrantWrite	赋予被授权者写的权限，格式 x-cos-grant-write: uin=" ",uin=" "， 当需要给子账户授权时，uin="RootAcountID/SubAccountID"， 当需要给根账户授权时，uin="RootAcountID"。	String	否
GrantFullControl	赋予被授权者读写权限，格式 x-cos-grant-full-control: uin=" ",uin=" "， 当需要给子账户授权时，uin="RootAcountID/SubAccountID"， 当需要给根账户授权时，uin="RootAcountID"。	String	否

回调函数说明

```
function(err, data) { ... }
```

回调参数说明

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误。如果请求成功，则为空	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
Location	创建成功后，Bucket 的操作地址	String

Delete Bucket

功能说明

Delete Bucket 请求可以在指定账号下删除 Bucket，删除之前要求 Bucket为空。

操作方法原型

调用 Delete Bucket 操作：

```
var params = {
Bucket : 'STRING_VALUE', /* 必须 */
Region : 'STRING_VALUE' /* 必须 */
};

cos.deleteBucket(params, function(err, data) {
if(err) {
console.log(err);
} else {
console.log(data);
}
});
```

操作参数说明

参数名	参数描述	类型	必填
Bucket	Bucket 的名称。命名规则为{name}-{appid} ，此处填写的存储桶名称必须为此格式	String	是
Region	Bucket 所在区域。	String	是

回调函数说明

```
function(err, data) { ... }
```

回调参数说明

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误。如果请求成功，则为空	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
DeleteBucketSuccess	删除是否成功	Boolean

Get Bucket ACL

功能说明

使用 API 读取 Bucket 的 ACL 表，只有所有者有权操作。

操作方法原型

调用 Get Bucket ACL 操作：

```
var params = {
Bucket : 'STRING_VALUE', /* 必须 */
Region : 'STRING_VALUE' /* 必须 */
};

cos.getBucketAcl(params, function(err, data) {
if(err) {
console.log(err);
} else {
console.log(data);
}
});
```

操作参数说明

参数名	参数描述	类型	必填
Bucket	Bucket 的名称。命名规则为{name}-{appid} ，此处填写的存储桶名称必须为此格式	String	是

参数名	参数描述	类型	必填
Region	Bucket 所在区域。	String	是

回调函数说明

```
function(err, data) { ... }
```

回调参数说明

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误。如果请求成功，则为空	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
Owner	标识资源的所有者	Object
uin	用户QQ号	String
AccessControlList	被授权者信息与权限信息	Object
Grant	具体的授权信息	Array
Permission	权限信息，枚举值：READ，WRITE，FULL_CONTROL	String
Grantee	被授权者资源信息	Object
uin	被授权者的 QQ 号码或者 'anonymous'	String
Subacount	子账户 QQ 账号	String

Put Bucket ACL

功能说明

使用 API 写入 Bucket 的 ACL 表，Put Bucket ACL 是一个覆盖操作，传入新的 ACL 将覆盖原有 ACL。只有所有者有权操作。

操作方法原型

调用 Put Bucket ACL 操作：

```
var params = {
Bucket : 'STRING_VALUE', /* 必须 */
Region : 'STRING_VALUE', /* 必须 */
ACL : 'STRING_VALUE', /* 非必须 */
GrantRead : 'STRING_VALUE', /* 非必须 */
GrantWrite : 'STRING_VALUE', /* 非必须 */
GrantFullControl : 'STRING_VALUE' /* 非必须 */
};

cos.putBucketAcl(params, function(err, data) {
if(err) {
console.log(err);
} else {
console.log(data);
}
});
```

操作参数说明

参数名	参数描述	类型	必填
Bucket	Bucket 的名称。命名规则为{name}-{appid} ，此处填写的存储桶名称必须为此格式	String	是
Region	Bucket 所在区域。	String	是
ACL	允许用户自定义文件权限。有效值：private，public-read默认值：private。	String	否

参数名	参数描述	类型	必填
GrantRead	赋予被授权者读的权限，格式 x-cos-grant-read: uin=" ",uin=" "， 当需要给子账户授权时，uin="RootAcountID/SubAccountID"， 当需要给根账户授权时，uin="RootAcountID"。	String	否
GrantWrite	赋予被授权者写的权限，格式 x-cos-grant-write: uin=" ",uin=" "， 当需要给子账户授权时，uin="RootAcountID/SubAccountID"， 当需要给根账户授权时，uin="RootAcountID"。	String	否
GrantFullControl	赋予被授权者读写权限，格式 x-cos-grant-full-control: uin=" ",uin=" "， 当需要给子账户授权时，uin="RootAcountID/SubAccountID"， 当需要给根账户授权时，uin="RootAcountID"。	String	否

回调函数说明

```
function(err, data) { ... }
```

回调参数说明

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误。如果请求成功，则为空	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
BucketGrantSuccess	授权是否成功	Boolean

Get Bucket CORS

功能说明

Get Bucket CORS 实现跨域访问读取。

操作方法原型

- 调用 Get Bucket CORS 操作

```
var params = {
Bucket : 'STRING_VALUE', /* 必须 */
Region : 'STRING_VALUE' /* 必须 */
};

cos.getBucketCors(params, function(err, data) {
if(err) {
console.log(err);
} else {
console.log(data);
}
});
```

操作参数说明

参数名	参数描述	类型	必填
Bucket	Bucket 的名称。命名规则为{name}-{appid} ，此处填写的存储桶名称必须为此格式	String	是
Region	Bucket 所在区域。	String	是

回调函数说明

```
function(err, data) { ... }
```

回调参数说明

参数名	参数描述	类型

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误。如果请求成功，则为空	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
CORSRule	配置的信息集合	Array
AllowedMethod	允许的 HTTP 操作，枚举值：Get，Put，Head，Post，Delete	Array
AllowedOrigin	允许的访问来源，支持『 * 』通配符	Array
AllowedHeader	在发送 OPTIONS 请求时告知服务端，接下来的请求可以使用哪些自定义的 HTTP 请求头部	Array
ExposeHeader	设置浏览器可以接收到的来自服务器端的自定义头部信息	Array
MaxAgeSeconds	设置 OPTIONS 请求得到结果的有效期	String
ID	规则名称	String

Put Bucket CORS

功能说明

Put Bucket CORS 实现跨域访问读写。

操作方法原型

调用 Put Bucket CORS 操作：

```
var params = {
Bucket : 'STRING_VALUE', /* 必须 */
Region : 'STRING_VALUE', /* 必须 */
CORSRules : [
{
ID : 'STRING_VALUE', /* 非必须 */
AllowedMethods: [ /* 必须 */
'STRING_VALUE',
...
],
AllowedOrigins: [ /* 必须 */
'STRING_VALUE',
...
],
AllowedHeaders: [ /* 非必须 */
'STRING_VALUE',
...
],
ExposeHeaders: [ /* 非必须 */
'STRING_VALUE',
...
],
MaxAgeSeconds: 'STRING_VALUE' /* 非必须 */
},
....
]
};

cos.putBucketCors(params, function(err, data) {
if(err) {
console.log(err);
} else {
console.log(data);
}
});
```

操作参数说明

参数名	参数描述	类型	必填

参数名	参数描述	类型	必填
Bucket	Bucket 的名称。命名规则为{name}-{appid} ，此处填写的存储桶名称必须为此格式	String	是
Region	Bucket 所在区域。	String	是
CORSRules	跨域规则集合	Array	否
ID	规则名称	String	否
AllowedMethods	允许的HTTP操作，枚举值：Get，Put，Head，Post，Delete	Array	是
AllowedOrigins	允许的访问来源，支持『 * 』通配符，协议，端口和域名必须一致	Array	是
AllowedHeaders	在发送 OPTIONS 请求时告知服务端，接下来的请求可以使用哪些自定义的 HTTP 请求头部，支持『 * 』通配符	Array	否
ExposeHeaders	设置浏览器可以接收到的来自服务器端的自定义头部信息	Array	否
MaxAgeSeconds	设置 OPTIONS 请求得到结果的有效期	String	否

回调函数说明

```
function(err, data) { ... }
```

回调参数说明

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误。如果请求成功，则为空	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
PutBucketCorsSuccess	设置Bucket CORS 是否成功	Boolean

Delete Bucket CORS

功能说明

Delete Bucket CORS 实现跨域访问读取。

操作方法原型

调用 Delete Bucket CORS 操作：

```
var params = {
Bucket : 'STRING_VALUE', /* 必须 */
Region : 'STRING_VALUE' /* 必须 */
};

cos.deleteBucketCors(params, function(err, data) {
if(err) {
console.log(err);
} else {
console.log(data);
}
});
```

操作参数说明

参数名	参数描述	类型	必填
Bucket	Bucket 的名称。命名规则为{name}-{appid} ，此处填写的存储桶名称必须为此格式	String	是
Region	Bucket 所在区域。	String	是

回调函数说明

```
function(err, data) { ... }
```

回调参数说明

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误。如果请求成功，则为空	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
DeleteBucketCorsSuccess	删除 Bucket CORS 是否成功	Boolean

Get Bucket Location

功能说明

Get Bucket Location 接口获取 Bucket 所在地域信息，只有 Bucket 所有者有权限读取信息。

操作方法原型

调用 Get Bucket Location 操作：

```
var params = {
Bucket : 'STRING_VALUE', /* 必须 */
Region : 'STRING_VALUE' /* 必须 */
};

cos.getBucketLocation(params, function(err, data) {
if(err) {
console.log(err);
} else {
console.log(data);
}
});
```

操作参数说明

参数名	参数描述	类型	必填
Bucket	Bucket 的名称。命名规则为{name}-{appid} ，此处填写的存储桶名称必须为此格式	String	是
Region	Bucket 所在区域。	String	是

回调函数说明

```
function(err, data) { ... }
```

回调参数说明

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误。如果请求成功，则为空	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
LocationConstraint	Bucket 所在区域，枚举值：china-east，china-south，china-north，china-west，singapore	String

Object操作

Head Object

功能说明

Head Object 请求可以取回对应 Object 的元数据，Head的权限与 Get 的权限一致。

操作方法原型

调用 Head Object 操作：

```
var params = {
Bucket : 'STRING_VALUE', /* 必须 */
Region : 'STRING_VALUE', /* 必须 */
Key : 'STRING_VALUE', /* 必须 */
IfModifiedSince : 'STRING_VALUE' /* 非必须 */
};

cos.headObject(params, function(err, data) {
if(err) {
console.log(err);
} else {
console.log(data);
}
});
```

操作参数说明

参数名	参数描述	类型	必填
Bucket	Bucket 的名称。命名规则为{name}-{appid} ，此处填写的存储桶名称必须为此格式	String	是
Region	Bucket 所在区域。	String	是
Key	文件名称	String	是
IfModifiedSince	当 Object 在指定时间后被修改，则返回对应 Object 元信息	String	否

回调函数说明

```
function(err, data) { ... }
```

回调参数说明

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误。如果请求成功，则为空	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
x-cos-object-type	用来表示object是否可以被追加上传，枚举值：normal或者appendable	String
x-cos-storage-class	Object的存储级别，枚举值：Standard，Standard_IA，Nearline	String
x-cos-meta- *	用户自定义的元数据	String
NotModified	如果请求时带有 IfModifiedSince ，且文件未被修改，则为 true	Boolean

Get Object

功能说明

Get Object 请求可以将一个文件（Object）下载至本地。该操作需要对目标 Object 具有读权限或目标 Object 对所有人都开放了读权限（公有读）。

操作方法原型

调用 Get Object 操作：

```
var params = {
Bucket : 'STRING_VALUE', /* 必须 */
Region : 'STRING_VALUE', /* 必须 */
Key : 'STRING_VALUE', /* 必须 */
ResponseContentType : 'STRING_VALUE', /* 非必须 */
ResponseContentLanguage : 'STRING_VALUE', /* 非必须 */
ResponseExpires : 'STRING_VALUE', /* 非必须 */
ResponseCacheControl : 'STRING_VALUE', /* 非必须 */
ResponseContentDisposition : 'STRING_VALUE', /* 非必须 */
ResponseContentEncoding : 'STRING_VALUE', /* 非必须 */
```

```
Range : 'STRING_VALUE', /* 非必须 */
IfModifiedSince : 'STRING_VALUE', /* 非必须 */
Output : 'STRING_VALUE' || 'WRITE_STRING' /* 必须 */
};

cos.getObject(params, function(err, data) {
if(err) {
console.log(err);
} else {
console.log(data);
}
});
```

操作参数说明

参数名	参数描述	类型	必填
Bucket	Bucket 的名称。命名规则为{name}-{appid} ，此处填写的存储桶名称必须为此格式	String	是
Region	Bucket 所在区域。	String	是
Key	String	文件名称	是
ResponseContentType	设置返回头部中的 Content-Type 参数	String	否
ResponseContentLanguage	设置返回头部中的 Content-Language 参数	String	否
ResponseExpires	设置返回头部中的 Content-Expires 参数	String	否
ResponseCacheControl	设置返回头部中的 Cache-Control 参数	String	否
ResponseContentDisposition	设置返回头部中的 Content-Disposition 参数	String	否
ResponseContentEncoding	设置返回头部中的 Content-Encoding 参数	String	否
Range	RFC 2616 中定义的指定文件下载范围，以字节（bytes）为单位	String	否
IfModifiedSince	如果文件修改时间晚于指定时间，才返回文件内容	String	否
Output	需要输出的文件路径或者一个写流	String / WriteStream	是

回调函数说明

```
function(err, data) { ... }
```

回调参数说明

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误。如果请求成功，则为空	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
x-cos-object-type	用来表示object是否可以被追加上传，枚举值：normal或者appendable	String
x-cos-storage-class	Object的存储级别，枚举值：Standard，Standard_IA，Nearline	String
x-cos-meta- *	用户自定义的元数据	String
NotModified	如果请求时带有 IfModifiedSince ，且文件未被修改，则为 true	Boolean

Put Object

功能说明

Put Object 请求可以将一个文件（Object）上传至指定 Bucket。

> **注意：**
>
> Key（文件名）不能以 `/` 结尾，否则会被识别为文件夹 。

操作方法原型

调用 Put Object 操作：

```
var params = {
Bucket : 'STRING_VALUE', /* 必须 */
Region : 'STRING_VALUE', /* 必须 */
Key : 'STRING_VALUE', /* 必须 */
CacheControl : 'STRING_VALUE', /* 非必须 */
ContentDisposition : 'STRING_VALUE', /* 非必须 */
ContentEncoding : 'STRING_VALUE', /* 非必须 */
ContentLength : 'STRING_VALUE', /* 必须 */
ContentType : 'STRING_VALUE', /* 非必须 */
Expect : 'STRING_VALUE', /* 非必须 */
Expires : 'STRING_VALUE', /* 非必须 */
ContentSha1 : 'STRING_VALUE', /* 非必须 */
ACL : 'STRING_VALUE', /* 非必须 */
GrantRead : 'STRING_VALUE', /* 非必须 */
GrantWrite : 'STRING_VALUE', /* 非必须 */
GrantFullControl : 'STRING_VALUE', /* 非必须 */
'x-cos-meta-*' : 'STRING_VALUE', /* 非必须 */
Body: fs.createReadStream('./a.zip'), /* 必须 */
onProgress: function (progressData) {
console.log(progressData);
},
};
cos.putObject(params, function(err, data) {
if(err) {
console.log(err);
} else {
console.log(data);
}
});
```

操作参数说明

参数名	参数描述	类型	必填
Bucket	Bucket 的名称。命名规则为{name}-{appid} ，此处填写的存储桶名称必须为此格式	String	是
Region	Bucket 所在区域。	String	是
Key	文件名称	String	是
CacheControl	RFC 2616 中定义的缓存策略，将作为 Object 元数据保存	String	否
ContentDisposition	RFC 2616 中定义的文件名称，将作为 Object 元数据保存	String	否
ContentEncoding	RFC 2616 中定义的编码格式，将作为 Object 元数据保存	String	否
ContentLength	RFC 2616 中定义的 HTTP 请求内容长度（字节）	String	是
ContentType	RFC 2616 中定义的内容类型（MIME），将作为 Object 元数据保存	String	否
Expect	当使用 Expect: 100-continue 时，在收到服务端确认后，才会发送请求内容	String	否
Expires	RFC 2616 中定义的过期时间，将作为 Object 元数据保存	String	否
ContentSha1	RFC 3174 中定义的 160-bit 内容 SHA-1 算法校验值	String	否
ACL	允许用户自定义文件权限，有效值：private，public-read	String	否
GrantRead	赋予被授权者读的权限，格式x-cos-grant-read: uin=" ",uin=" "， 当需要给子账户授权时，uin="RootAcountID/SubAccountID"， 当需要给根账户授权时，uin="RootAcountID"	String	否
GrantWrite	赋予被授权者写的权限，格式x-cos-grant-write: uin=" ",uin=" "， 当需要给子账户授权时，uin="RootAcountID/SubAccountID"， 当需要给根账户授权时，uin="RootAcountID"	String	否

参数名	参数描述	类型	必填
GrantFullControl	赋予被授权者读写权限，格式x-cos-grant-full-control: uin=" ",uin=" "， 当需要给子账户授权时，uin="RootAcountID/SubAccountID"， 当需要给根账户授权时，uin="RootAcountID"	String	否
x-cos-meta- *	允许用户自定义的头部信息，将作为 Object 元数据返回。大小限制2K。	String	否
Body	传入文件路径或文件流	String/ Stream	是
onProgress	进度回调函数，回调是一个对象，包含进度信息	Function	否

回调函数说明

```
function(err, data) { ... }
```

回调参数说明

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误。如果请求成功，则为空	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
ETag	返回文件的 MD5 算法校验值。ETag 的值可以用于检查 Object 的内容是否发生变化	String

Delete Object

功能说明

Delete Object 请求可以将一个文件（Object）删除。

操作方法原型

调用 Delete Object 操作：

```
var params = {
Bucket : 'STRING_VALUE', /* 必须 */
Region : 'STRING_VALUE', /* 必须 */
Key : 'STRING_VALUE' /* 必须 */
};

cos.deleteObject(params, function(err, data) {
if(err) {
console.log(err);
} else {
console.log(data);
}
});
```

操作参数说明

参数名	参数描述	类型	必填
Bucket	Bucket 的名称。命名规则为{name}-{appid} ，此处填写的存储桶名称必须为此格式	String	是
Region	Bucket 所在区域。	String	是
Key	文件名称	String	是

回调函数说明

```
function(err, data) { ... }
```

回调参数说明

参数名	参数描述	类型

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误。如果请求成功，则为空	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
DeleteObjectSuccess	删除文件是否成功	Boolean
BucketNotFound	如果找不到指定的 Bucket，则为 true	Boolean

Options Object

功能说明

Options Object 请求实现跨域访问的预请求。即发出一个 OPTIONS 请求给服务器以确认是否可以进行跨域操作。

操作方法原型

调用 Options Object 操作：

```
var params = {
Bucket : 'STRING_VALUE', /* 必须 */
Region : 'STRING_VALUE', /* 必须 */
Key : 'STRING_VALUE', /* 必须 */
Origin : 'STRING_VALUE', /* 必须 */
AccessControlRequestMethod : 'STRING_VALUE', /* 必须 */
AccessControlRequestHeaders : 'STRING_VALUE' /* 非必须 */
};

cos.optionsObject(params, function(err, data) {
if(err) {
console.log(err);
} else {
console.log(data);
}
});
```

操作参数说明

参数名	参数描述	类型	必填
Bucket	Bucket 的名称。命名规则为{name}-{appid} ，此处填写的存储桶名称必须为此格式	String	是
Region	Bucket 所在区域。	String	是
Key	文件名称	String	是
Origin	模拟跨域访问的请求来源域名	String	是
AccessControlRequestMethod	模拟跨域访问的请求HTTP方法	String	是
AccessControlRequestHeaders	模拟跨域访问的请求头部	String	否

回调函数说明

```
function(err, data) { ... }
```

回调参数说明

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误。如果请求成功，则为空	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
AccessControlAllowOrigin	模拟跨域访问的请求来源域名，当来源不允许的时候，此Header不返回	String
AccessControlAllowMethods	模拟跨域访问的请求HTTP方法，当请求方法不允许的时候，此Header不返回	String

参数名	参数描述	类型
AccessControlAllowHeaders	模拟跨域访问的请求头部，当模拟任何请求头部不允许的时候，此Header不返回该请求头部	String
AccessControlExposeHeaders	跨域支持返回头部，用逗号区分	String
AccessControlMaxAge	设置 OPTIONS 请求得到结果的有效期	String

Get Object ACL

功能说明

Get Object ACL 接口实现使用 API 读取 Object 的 ACL 表，只有所有者有权操作。

操作方法原型

调用 Get Object ACL 操作：

```
var params = {
Bucket : 'STRING_VALUE', /* 必须 */
Region : 'STRING_VALUE', /* 必须 */
Key : 'STRING_VALUE' /* 必须 */
};

cos.getObjectAcl(params, function(err, data) {
if(err) {
console.log(err);
} else {
console.log(data);
}
});
```

操作参数说明

参数名	参数描述	类型	必填
Bucket	Bucket 的名称。命名规则为{name}-{appid} ，此处填写的存储桶名称必须为此格式	String	是
Region	Bucket 所在区域。	String	是
Key	文件名称	String	是

回调函数说明

```
function(err, data) { ... }
```

回调参数说明

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误。如果请求成功，则为空	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
Owner	标识资源的所有者	Object
uin	用户QQ号	String
AccessControlList	被授权者信息与权限信息	Object
Grant	具体的授权信息	Array
Permission	权限信息，枚举值：READ，WRITE，FULL_CONTROL	String
Grantee	被授权者资源信息	Object
uin	被授权者的 QQ 号码或者 'anonymous'	String
Subacount	子账户 QQ 账号	String

Put Object ACL

功能说明

Put Object ACL使用 API 写入 Object 的 ACL 表。

操作方法原型

调用 Put Object ACL 操作：

```
var params = {
Bucket : 'STRING_VALUE', /* 必须 */
Region : 'STRING_VALUE', /* 必须 */
Key : 'STRING_VALUE', /* 必须 */
ACL : 'STRING_VALUE', /* 非必须 */
GrantRead : 'STRING_VALUE', /* 非必须 */
GrantWrite : 'STRING_VALUE', /* 非必须 */
GrantFullControl : 'STRING_VALUE' /* 非必须 */
};

cos.putObjectAcl(params, function(err, data) {
if(err) {
console.log(err);
} else {
console.log(data);
}
});
```

操作参数说明

参数名	参数描述	类型	必填
Bucket	Bucket 的名称。命名规则为{name}-{appid} ，此处填写的存储桶名称必须为此格式	String	是
Region	Bucket 所在区域。	String	是
Key	文件名称	String	是
ACL	允许用户自定义文件权限。有效值：private，public-read默认值：private。	String	否
GrantRead	赋予被授权者读的权限，格式 x-cos-grant-read: uin=" ",uin=" "， 当需要给子账户授权时，uin="RootAcountID/SubAccountID"， 当需要给根账户授权时，uin="RootAcountID"。	String	否
GrantWrite	赋予被授权者写的权限，格式 x-cos-grant-write: uin=" ",uin=" "， 当需要给子账户授权时，uin="RootAcountID/SubAccountID"， 当需要给根账户授权时，uin="RootAcountID"。	String	否
GrantFullControl	赋予被授权者读写权限，格式 x-cos-grant-full-control: uin=" ",uin=" "， 当需要给子账户授权时，uin="RootAcountID/SubAccountID"， 当需要给根账户授权时，uin="RootAcountID"。	String	否

回调函数说明

```
function(err, data) { ... }
```

回调参数说明

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误。如果请求成功，则为空	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object

Delete Multiple Object

功能说明

Delete Multiple Object 请求实现批量删除文件，最大支持单次删除 1000 个文件。对于返回结果，COS 提供 Verbose 和 Quiet 两种结果模式。Verbose 模式将返回每个 Object 的删除结果；Quiet 模式只返回报错的 Object 信息。

操作方法原型

调用 Delete Multiple Object 操作：

```
var params = {
Bucket : 'STRING_VALUE', /* 必须 */
Region : 'STRING_VALUE', /* 必须 */
Quiet : 'BOOLEAN_VALUE', /* 非必须 */
Objects : [
{
Key : 'STRING_VALUE' /* 必须 */
}
]
};

cos.deleteMultipleObject(params, function(err, data) {
if(err) {
console.log(err);
} else {
console.log(data);
}
});
```

操作参数说明

参数名	参数描述	类型	必填
Bucket	Bucket 的名称。命名规则为{name}-{appid} ，此处填写的存储桶名称必须为此格式	String	是
Region	Bucket 所在区域。	String	是
Quiet	布尔值，这个值决定了是否启动Quiet模式，True启动Quiet模式，False启动Verbose模式，默认False	Boolean	否
Objects	要删除的文件列表	Array	否
Key	要删除的文件名称	String	是

回调函数说明

```
function(err, data) { ... }
```

回调参数说明

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误。如果请求成功，则为空	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
Deleted	说明本次删除的成功 Object 信息	Array
Key	Object 的名称	String
Error	说明本次删除的失败 Object 信息	Array
Key	Object 的名称	String
Code	删除失败的错误码	String
Message	删除错误信息	String

分块上传操作

Initiate Multipart Upload

功能说明

Initiate Multipart Upload 请求实现初始化分片上传，成功执行此请求以后会返回 Upload ID 用于后续的 Upload Part 请求。

操作方法原型

调用 Initiate Multipart Upload 操作：

```
var params = {
Bucket : 'STRING_VALUE', /* 必须 */
Region : 'STRING_VALUE', /* 必须 */
Key : 'STRING_VALUE', /* 必须 */
CacheControl : 'STRING_VALUE', /* 非必须 */
ContentDisposition : 'STRING_VALUE', /* 非必须 */
ContentEncoding : 'STRING_VALUE', /* 非必须 */
ContentType : 'STRING_VALUE', /* 非必须 */
Expires : 'STRING_VALUE', /* 非必须 */
ACL : 'STRING_VALUE', /* 非必须 */
GrantRead : 'STRING_VALUE', /* 非必须 */
GrantWrite : 'STRING_VALUE', /* 非必须 */
GrantFullControl : 'STRING_VALUE', /* 非必须 */
StorageClass : 'STRING_VALUE', /* 非必须 */
'x-cos-meta-*' : 'STRING_VALUE' /* 非必须 */
};

cos.multipartInit(params, function(err, data) {
if(err) {
console.log(err);
} else {
console.log(data);
}
});
```

操作参数说明

参数名	参数描述	类型	必填
Bucket	Bucket 的名称。命名规则为{name}-{appid} ，此处填写的存储桶名称必须为此格式	String	是
Region	Bucket 所在区域。	String	是
Key	文件名称	String	是
CacheControl	RFC 2616 中定义的缓存策略，将作为 Object 元数据保存	String	否
ContentDisposition	RFC 2616 中定义的文件名称，将作为 Object 元数据保存	String	否
ContentEncoding	RFC 2616 中定义的编码格式，将作为 Object 元数据保存	String	否
ContentType	RFC 2616 中定义的内容类型（MIME），将作为 Object 元数据保存	String	否
Expires	RFC 2616 中定义的过期时间，将作为 Object 元数据保存	String	否
ACL	允许用户自定义文件权限，有效值：private，public-read	String	否
GrantRead	赋予被授权者读的权限，格式x-cos-grant-read: uin=" ",uin=" "， 当需要给子账户授权时，uin="RootAcountID/SubAccountID"， 当需要给根账户授权时，uin="RootAcountID"	String	否
GrantWrite	赋予被授权者写的权限，格式x-cos-grant-write: uin=" ",uin=" "， 当需要给子账户授权时，uin="RootAcountID/SubAccountID"， 当需要给根账户授权时，uin="RootAcountID"	String	否
GrantFullControl	赋予被授权者读写权限，格式x-cos-grant-full-control: uin=" ",uin=" "， 当需要给子账户授权时，uin="RootAcountID/SubAccountID"， 当需要给根账户授权时，uin="RootAcountID"	String	否
StorageClass	设置Object的存储级别，枚举值：Standard，Standard_IA，Nearline，默认值：Standard（目前只支持华南园区）	String	否
x-cos-meta- *	允许用户自定义的头部信息，将作为 Object 元数据返回。大小限制 2 K	String	否

回调函数说明

```
function(err, data) { ... }
```

回调参数说明

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误。如果请求成功，则为空	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
Bucket	分片上传的目标 Bucket	String
Key	Object 的名称	String
UploadId	在后续上传中使用的ID	String

Upload Part

功能说明

Upload Part 请求实现在初始化以后的分块上传，支持的块的数量为 1 到 10000，块的大小为 1 MB 到 5 GB。在每次请求Upload Part 时候，需要携带 partNumber 和 uploadID，partNumber 为块的编号，支持乱序上传。

操作方法原型

调用 Upload Part 操作：

```
var params = {
Bucket : 'STRING_VALUE', /* 必须 */
Region : 'STRING_VALUE', /* 必须 */
Key : 'STRING_VALUE', /* 必须 */
ContentLength : 'STRING_VALUE', /* 必须 */
Expect : 'STRING_VALUE', /* 非必须 */
ContentSha1 : 'STRING_VALUE', /* 非必须 */
PartNumber : 'STRING_VALUE', /* 必须 */
UploadId : 'STRING_VALUE', /* 必须 */
};

cos.multipartUpload(params, function(err, data) {
if(err) {
console.log(err);
} else {
console.log(data);
}
});
```

操作参数说明

参数名	参数描述	类型	必填
Bucket	Bucket 的名称。命名规则为{name}-{appid} ，此处填写的存储桶名称必须为此格式	String	是
Region	Bucket 所在区域。	String	是
Key	文件名称	String	是
ContentLength	RFC 2616 中定义的 HTTP 请求内容长度（字节）	String	是
Expect	当使用 Expect: 100-continue 时，在收到服务端确认后，才会发送请求内容	String	否
ContentSha1	RFC 3174 中定义的 160-bit 内容 SHA-1 算法校验值	String	否
PartNumber	分块的编号	String	是
UploadId	上传任务编号	String	是

回调函数说明

```
function(err, data) { ... }
```

回调参数说明

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误。如果请求成功，则为空	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
ETag	分块的 ETag 值，为 sha1 校验值	String

Complete Multipart Upload

功能说明

Complete Multipart Upload 用来实现完成整个分块上传。当您已经使用 Upload Parts 上传所有块以后，你可以用该 API 完成上传。在使用该 API 时，您必须在 Body 中给出每一个块的 PartNumber 和 ETag，用来校验块的准确性。

操作方法原型

调用 Complete Multipart Upload 操作：

```
var params = {
Bucket : 'STRING_VALUE', /* 必须 */
Region : 'STRING_VALUE', /* 必须 */
Key : 'STRING_VALUE', /* 必须 */
UploadId : 'STRING_VALUE', /* 必须 */
Parts : [
{
PartNumber : 'STRING_VALUE', /* 必须 */
ETag : 'STRING_VALUE' /* 必须 */
},
...
]
};

cos.multipartComplete(params, function(err, data) {
if(err) {
console.log(err);
} else {
console.log(data);
}
});
```

操作参数说明

参数名	参数描述	类型	必填
Bucket	Bucket 的名称。命名规则为{name}-{appid} ，此处填写的存储桶名称必须为此格式	String	是
Region	Bucket 所在区域。	String	是
Key	文件名称	String	是
UploadId	上传任务编号	String	是
Parts	分块的ETag 信息	Array	是
PartNumber	分块的编号	String	是
ETag	分块的ETag 值，为 sha1 校验值，需要在校验值前后加上双引号，如 "3a0f1fd698c235af9cf098cb74aa25bc"	String	是

回调函数说明

```
function(err, data) { ... }
```

回调参数说明

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误。如果请求成功，则为空	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
Location	创建的 Object 的外网访问域名	String
Bucket	分块上传的目标 Bucket	String
Key	Object 的名称	String
ETag	合并后文件的 MD5 算法校验值	String

List Parts

功能说明

List Parts用来查询特定分块上传中的已上传的块

操作方法原型

调用 List Parts 操作：

```
var params = {
Bucket : 'STRING_VALUE', /* 必须 */
Region : 'STRING_VALUE', /* 必须 */
Key : 'STRING_VALUE', /* 必须 */
UploadId : 'STRING_VALUE', /* 必须 */
EncodingType : 'STRING_VALUE', /* 非必须 */
MaxParts : 'STRING_VALUE', /* 非必须 */
PartNumberMarker : 'STRING_VALUE' /* 非必须 */
};

cos.multipartListPart(params, function(err, data) {
if(err) {
console.log(err);
} else {
console.log(data);
}
});
```

操作参数说明

参数名	参数描述	类型	必填
Bucket	Bucket 的名称。命名规则为{name}-{appid} ，此处填写的存储桶名称必须为此格式	String	是
Region	Bucket 所在区域。	String	是
Key	文件名称	String	是
UploadId	上传任务编号	String	是
EncodingType	规定返回值的编码方式	String	否
MaxParts	单次返回最大的条目数量，默认1000	String	否
PartNumberMarker	默认以 UTF-8 二进制顺序列出条目，所有列出条目从 marker 开始	String	否

回调函数说明

```
function(err, data) { ... }
```

回调参数说明

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误。如果请求成功，则为空	Object

参数名	参数描述	类型
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
Bucket	分块上传的目标 Bucket	String
Encoding-type	规定返回值的编码方式	String
Key	Object 的名称	String
UploadID	标示本次分块上传的 ID	String
Initiator	用来表示本次上传发起者的信息，子节点包括 UID	Object
UID	开发商 APPID	String
Owner	用来表示这些分块所有者的信息，子节点包括 UID	Object
UID	拥有者 qq	String
StorageClass	用来表示这些分块的存储级别，枚举值：Standard，Standard_IA，Nearline	String
PartNumberMarker	默认以 UTF-8 二进制顺序列出条目，所有列出条目从 marker 开始	String
NextPartNumberMarker	假如返回条目被截断，则返回 NextMarker 就是下一个条目的起点	String
MaxParts	单次返回最大的条目数量	String
IsTruncated	返回条目是否被截断，'true' 或者 'false'	String
Part	分块信息集合	Array
PartNumber	分块编号	String
LastModified	块最后修改时间	String
Etag	块的 SHA-1 算法校验值	String
Size	块大小，单位 Byte	String

Abort Multipart Upload

功能说明

Abort Multipart Upload 用来实现舍弃一个分块上传并删除已上传的块。当您调用 Abort Multipart Upload 时，如果有正在使用这个 Upload Parts 上传块的请求，则 Upload Parts 会返回失败。

操作方法原型

调用 Abort Multipart Upload 操作：

```
var params = {
Bucket : 'STRING_VALUE', /* 必须 */
Region : 'STRING_VALUE', /* 必须 */
Key : 'STRING_VALUE', /* 必须 */
UploadId : 'STRING_VALUE' /* 必须 */
};

cos.multipartAbort(params, function(err, data) {
if(err) {
console.log(err);
} else {
console.log(data);
}
});
```

操作参数说明

参数名	参数描述	类型	必填
Bucket	Bucket 的名称。命名规则为{name}-{appid} ，此处填写的存储桶名称必须为此格式	String	是

参数名	参数描述	类型	必填
Region	Bucket 所在区域。	String	是
Key	文件名称	String	是
UploadId	上传任务编号	String	是

回调函数说明

```
function(err, data) { ... }
```

回调参数说明

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误。如果请求成功，则为空	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
MultipartAbortSuccess	Multipart Abort 是否成功	Boolean

List Multipart Uploads

功能说明

List Multiparts Uploads 用来查询正在进行中的分块上传。单次最多列出 1000 个正在进行中的分块上传。

操作方法原型

调用 List Multipart Uploads 操作：

```
var params = {
Bucket : 'STRING_VALUE', /* 必须 */
Region : 'STRING_VALUE', /* 必须 */
Delimiter : 'STRING_VALUE', /* 非必须 */
EncodingType : 'STRING_VALUE', /* 非必须 */
Prefix : 'STRING_VALUE', /* 非必须 */
MaxUploads : 'STRING_VALUE', /* 非必须 */
KeyMarker : 'STRING_VALUE', /* 非必须 */
UploadIdMarker : 'STRING_VALUE' /* 非必须 */
};

cos.multipartList(params, function(err, data) {
if(err) {
console.log(err);
} else {
console.log(data);
}
});
```

操作参数说明

参数名	参数描述	类型	必填
Bucket	Bucket 的名称。命名规则为{name}-{appid} ，此处填写的存储桶名称必须为此格式	String	是
Region	Bucket 所在区域。	String	是
Delimiter	界符为一个符号，如果有Prefix，则将Prefix到delimiter之间的相同路径归为一类，定义为Common Prefix，然后列出所有Common Prefix。如果没有Prefix，则从路径起点开始	String	否
EncodingType	规定返回值的编码方式	String	否
Prefix	前缀匹配，用来规定返回的文件前缀地址	String	否
MaxUploads	单次返回最大的条目数量，默认1000	String	否

参数名	参数描述	类型	必填
KeyMarker	与upload-id-marker一起使用， • 当 upload-id-marker 未被指定时， ObjectName 字母顺序大于 key-marker 的条目将被列出； • 当 upload-id-marker 被指定时， ObjectName 字母顺序大于 key-marker 的条目被列出， ObjectName 字母顺序等于 key-marker 且 UploadID 大于 upload-id-marker 的条目将被列出。	String	否
UploadIdMarker	与key-marker一起使用 • 当key-marker未被指定时， upload-id-marker将被忽略； • 当key-marker被指定时， ObjectName字母顺序大于key-marker的条目被列出， ObjectName字母顺序等于 key-marker 且 UploadID 大于 upload-id-marker 的条目将被列出	String	否

回调函数说明

```
function(err, data) { ... }
```

回调参数说明

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误。如果请求成功，则为空	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object
Bucket	分块上传的目标 Bucket	String
Encoding-type	规定返回值的编码方式	String
KeyMarker	列出条目从该 key 值开始	String
UploadIdMarker	列出条目从该 UploadId 值开始	String
NextKeyMarker	假如返回条目被截断，则返回 NextKeyMarker 就是下一个条目的起点	String
NextUploadIdMarker	假如返回条目被截断，则返回 UploadId 就是下一个条目的起点	String
IsTruncated	返回条目是否被截断，'true' 或者 'false'	String
delimiter	定界符为一个符号，如果有 Prefix，则将 Prefix 到 delimiter 之间的相同路径归为一类，定义为 Common Prefix，然后列出所有 Common Prefix。如果没有 Prefix，则从路径起点开始	String
CommonPrefixs	将 Prefix 到 delimiter 之间的相同路径归为一类，定义为 Common Prefix	Array
Prefix	具体的 Prefix 值	String
Upload	Upload 的信息集合	Array
Key	Object 的名称	String
UploadID	标示本次分块上传的 ID	String
StorageClass	用来表示分块的存储级别，枚举值：Standard，Standard_IA，Nearline	String
Initiator	用来表示本次上传发起者的信息，子节点包括 UID	Object
UID	开发商 APPID	String
Owner	用来表示这些分块所有者的信息，子节点包括 UID	Object

参数名	参数描述	类型
UID	拥有者 qq	String
Initiated	分块上传的起始时间	String

Slice Upload File

功能说明

Slice Upload File 可用于实现文件的分块上传。

操作方法原型

调用 Slice Upload File 操作：

```
var params = {
Bucket: 'STRING_VALUE', /* 必须 */
Region: 'STRING_VALUE', /* 必须 */
Key: 'STRING_VALUE', /* 必须 */
FilePath: 'STRING_VALUE', /* 必须 */
SliceSize: 'STRING_VALUE', /* 非必须 */
AsyncLimit: 'NUMBER_VALUE', /* 非必须 */
onHashProgress: function (progressData) {
console.log(JSON.stringify(progressData));
},
onProgress: function (progressData) {
console.log(JSON.stringify(progressData));
},
};

cos.sliceUploadFile(params, function(err, data) {
if(err) {
console.log(err);
} else {
console.log(data);
}
});
```

操作参数说明

参数名	参数描述	类型	必填
Bucket	Bucket 的名称。命名规则为{name}-{appid} ，此处填写的存储桶名称必须为此格式	String	是
Region	Bucket 所在区域。	String	是
Key	Object 名称	String	是
FilePath	本地文件路径	String	是
SliceSize	分块大小	String	否
AsyncLimit	分块的并发量	String	否
onHashProgress	计算文件 sha1 值的进度回调函数，回调是一个对象，包含进度信息	Function	否
onProgress	进度回调函数，回调是一个对象，包含进度信息	Function	否

回调函数说明

```
function(err, data) { ... }
```

回调参数说明

参数名	参数描述	类型
err	请求发生错误时返回的对象，包括网络错误和业务错误。如果请求成功，则为空	Object
data	请求成功时返回的对象，如果请求发生错误，则为空	Object

参数名	参数描述	类型
Location	创建的 Object 的外网访问域名	String
Bucket	分块上传的目标 Bucket	String
Key	Object 的名称	String
ETag	合并后文件的 SHA-1 算法校验值	String

进度回调参数

参数名	参数描述	类型
SliceSize	分块大小	String
PartNumber	上传成功的分块编号	Number
FileSize	文件总大小	Number

PHP SDK
快速入门

最近更新时间: 2025-01-15 17:01:00

下载与安装

相关资源

- 对象存储 COS 的 XML PHP SDK 源码地址： XML PHP SDK 。
- 示例 Demo 程序地址：PHP sample。

环境依赖

- PHP 5.3+ 您可以通过 `php -v` 命令查看当前的 PHP 版本。
- cURL 扩展 您可以通过 php -m 命令查看 cURL 扩展是否已经安装好。
- Ubuntu 系统中，您可以使用 apt-get 包管理器安装 PHP 的 cURL 扩展，安装命令如下：

```
sudo apt-get install php-curl
```

- CentOS 系统中，您可以使用 yum 包管理器安装 PHP 的 cURL 扩展。

```
sudo yum install php-curl
```

安装 SDK

SDK 安装有三种方式：Composer 方式、Phar 方式 和 源码方式。

Composer 方式

推荐使用 Composer 安装 cos-php-sdk-v5，Composer 是 PHP 的依赖管理工具，允许您声明项目所需的依赖，然后自动将它们安装到您的项目中。

> 您可以在 Composer 官网 上找到更多关于如何安装 Composer，配置自动加载以及用于定义依赖项的其他最佳实践等相关信息。

安装步骤

1. 打开终端。
2. 下载 Composer，执行以下命令。

```
curl -sS http://imgcache.finance.cloud.tencent.com:80getcomposer.org/installer | php
```

3. 创建一个名为 `composer.json` 的文件，内容如下。

```
{
"require": {
"qcloud/cos-sdk-v5": ">=1.0"
}
}
```

4. 使用 Composer 安装，执行以下命令。

```
php composer.phar install
```

使用该命令后会在当前目录中创建一个 vendor 文件夹，里面包含 SDK 的依赖库和一个 autoload.php 脚本，方便在项目中调用。

5. 通过 autoloader 脚本调用 cos-php-sdk-v5。

```
require '/path/to/sdk/vendor/autoload.php';
```

至此，您的项目已经可以使用 COS XML PHP SDK 了。

Phar 方式

Phar 方式安装 SDK 的步骤如下：

1. 在 GitHub 发布页面 下载相应的 phar 文件。
2. 在代码中引入 phar 文件：

```
require '/path/to/cos-sdk-v5.phar';
```

源码方式

源码方式安装 SDK 的步骤如下：

1. 在 SDK 下载页面 下载 `cos-sdk-v5.tar.gz` 压缩文件。（注意：`Source code` 压缩包为 Github 默认打包的代码包，里面不包含 `vendor` 目录）
2. 解压后通过 `autoload.php` 脚本加载 SDK：

```
require '/path/to/sdk/vendor/autoload.php';
```

开始使用

下面为您介绍如何使用 COS PHP SDK 完成一个基础操作，如初始化客户端、创建存储桶、查询存储桶列表、上传对象、查询对象列表、下载对象和删除对象。

术语解释

名称	描述
APPID	开发者访问 COS 服务时拥有的用户维度唯一资源标识，用以标识资源
SecretId	开发者拥有的项目身份识别 ID，用以身份认证
SecretKey	开发者拥有的项目身份密钥
Bucket	COS 中用于存储数据的容器
Object	COS 中存储的具体文件，是存储的基本实体
Region	域名中的地域信息
Endpoint	Endpoint 由 Region 和域名组成，具体格式为：`"."`，其中 Domain 为自定义的域名。 在控制台创建 Bucket 时，可以看到对应的访问地址为：`"."`，Bucket 后面的部分即为 Endpoint。
ACL	访问控制列表（Access Control List），是指特定 Bucket 或 Object 的访问控制信息列表
CORS	跨域资源共享（Cross-Origin Resource Sharing），指发起请求的资源所在域不同于该请求所指向资源所在的域的 HTTP 请求
Multipart Uploads	分块上传，COS 服务为上传文件提供的一种分块上传模式

初始化

```
$secretId = "COS_SECRETID"; // 替换为用户的 SecretId
$secretKey = "COS_SECRETKEY"; // 替换为用户的 SecretKey

$region = "REGION"; // 替换为用户的 Region
$domain = "DOMAIN.COM"; // 替换为用户的 Domain

$formatRegion = sprintf("cos.%s", $region); // 格式化为标准的 Region (前方加 cos.)
$endpoint = sprintf("%s.%s", $formatRegion, $domain);
// 通过 FormatRegion 和 Domain 生成 Endpoint

$cosClient = new Qcloud\Cos\Client(
array(
'schema' => 'http', // 协议头部，默认为http
'region' => $region, // Region
'endpoint' => $endpoint, // Endpoint
'credentials'=> array(
'secretId' => $secretId ,
'secretKey' => $secretKey)));
```

若您使用 临时密钥 初始化，请用下面方式创建实例。

```
$secretId = "COS_SECRETID"; // 临时密钥 SecretId
$secretKey = "COS_SECRETKEY"; // 临时密钥 SecretKey
$tmpToken = "COS_TMPTOKEN" // 临时密钥 Token

$region = "REGION"; // 替换为用户的 Region
$domain = "DOMAIN.COM"; // 替换为用户的 Domain

$formatRegion = sprintf("cos.%s", $region); // 格式化为标准的 Region (前方加 cos.)
$endpoint = sprintf("%s.%s", $formatRegion, $domain);
// 通过 FormatRegion 和 Domain 生成 Endpoint

$cosClient = new Qcloud\Cos\Client(
array(
'schema' => 'http', // 协议头部，默认为http
'region' => $region, // Region
'endpoint' => $endpoint, // Endpoint
'credentials'=> array(
'secretId' => $secretId ,
'secretKey' => $secretKey,
'token' => $tmpToken)));
```

创建存储桶

```
try {
$bucket = "examplebucket-1250000000"; //存储桶名称 格式：BucketName-APPID
$result = $cosClient->createBucket(array('Bucket' => $bucket));
//请求成功
print_r($result);
} catch (\Exception $e) {
//请求失败
echo($e);
}
```

上传对象

- 使用 putObject 接口上传文件（最大5G）。
- 使用 Upload 接口分块上传文件。

```
# 上传文件
## putObject(上传接口，最大支持上传5G文件)
### 上传内存中的字符串
try {
$bucket = "examplebucket-1250000000"; //存储桶名称 格式：BucketName-APPID
$key = "exampleobject";
$result = $cosClient->putObject(array(
'Bucket' => $bucket,
'Key' => $key,
'Body' => 'Hello World!'));
print_r($result);
} catch (\Exception $e) {
echo "$e\n";
}

### 上传文件流
try {
$bucket = "examplebucket-1250000000"; //存储桶名称 格式：BucketName-APPID
$key = "exampleobject";
$srcPath = "F:/exampleobject";//本地文件绝对路径
$result = $cosClient->putObject(array(
'Bucket' => $bucket,
'Key' => $key,
'Body' => fopen($srcPath, 'rb')));
print_r($result);
} catch (\Exception $e) {
echo "$e\n";
}
```

```
### 设置header和meta
try {
$bucket = "examplebucket-1250000000"; //存储桶名称 格式：BucketName-APPID
$key = "exampleobject";
$srcPath = "F:/exampleobject";//本地文件绝对路径
$result = $cosClient->putObject(array(
'Bucket' => $bucket,
'Key' => $key,
'Body' => fopen($srcPath, 'rb'),
'ACL' => 'string',
'CacheControl' => 'string',
'ContentDisposition' => 'string',
'ContentEncoding' => 'string',
'ContentLanguage' => 'string',
'ContentLength' => integer,
'ContentType' => 'string',
'Expires' => 'mixed type: string (date format)|int (unix timestamp)|\DateTime',
'GrantFullControl' => 'string',
'GrantRead' => 'string',
'GrantWrite' => 'string',
'Metadata' => array(
'string' => 'string',
),
'StorageClass' => 'string'));
print_r($result);
} catch (\Exception $e) {
echo "$e\n";
}

## Upload(高级上传接口，默认使用分块上传最大支持50T)
### 上传内存中的字符串
try {
$bucket = "examplebucket-1250000000"; //存储桶名称 格式：BucketName-APPID
$key = "exampleobject";
$result = $cosClient->Upload(
$bucket = $bucket,
$key = $key,
$body = 'Hello World!');
print_r($result);
} catch (\Exception $e) {
echo "$e\n";
}

### 上传文件流
try {
$bucket = "examplebucket-1250000000"; //存储桶名称 格式：BucketName-APPID
$key = "exampleobject";
$srcPath = "F:/exampleobject";//本地文件绝对路径
$result = $cosClient->Upload(
$bucket = $bucket,
$key = $key,
$body = fopen($srcPath, 'rb'));
print_r($result);
} catch (\Exception $e) {
echo "$e\n";
}

### 设置header和meta
try {
$bucket = "examplebucket-1250000000"; //存储桶名称 格式：BucketName-APPID
$key = "exampleobject";
$srcPath = "F:/exampleobject";//本地文件绝对路径
$result = $cosClient->Upload(
$bucket= $bucket,
$key = $key,
$body = fopen($srcPath, 'rb'),
$options = array(
'ACL' => 'string',
'CacheControl' => 'string',
'ContentDisposition' => 'string',
'ContentEncoding' => 'string',
```

```
'ContentLanguage' => 'string',
'ContentLength' => integer,
'ContentType' => 'string',
'Expires' => 'mixed type: string (date format)|int (unix timestamp)|\DateTime',
'GrantFullControl' => 'string',
'GrantRead' => 'string',
'GrantWrite' => 'string',
'Metadata' => array(
'string' => 'string',
),
'StorageClass' => 'string'));
print_r($result);
} catch (\Exception $e) {
echo "$e\n";
}
```

查询对象列表

```
try {
$bucket = "examplebucket-1250000000"; //存储桶名称 格式：BucketName-APPID
$result = $cosClient->listObjects(array(
'Bucket' => $bucket
));
// 请求成功
foreach ($result['Contents'] as $rt) {
print_r($rt);
}
} catch (\Exception $e) {
// 请求失败
echo($e);
}
```

单次调用 listObjects 接口一次只能查询1000个对象，如需要查询所有的对象，则需要循环调用

```
try {
$bucket = "examplebucket-1250000000"; //存储桶名称 格式：BucketName-APPID
$prefix = ''; //列出对象的前缀
$marker = ''; //上次列出对象的断点
while (true) {
$result = $cosClient->listObjects(array(
'Bucket' => $bucket,
'Marker' => $marker,
'MaxKeys' => 1000 //设置单次查询打印的最大数量，最大为1000
));
foreach ($result['Contents'] as $rt) {
// 打印key
echo($rt['Key'] . "\n");
}
$marker = $result['NextMarker']; //设置新的断点
if (!$result['IsTruncated']) {
break; //判断是否已经查询完
}
}
} catch (\Exception $e) {
echo($e);
}
```

下载对象

- 使用 getObject 接口下载文件。
- 使用 getObjectUrl 接口获取文件下载 URL。

```
# 下载文件
## getObject(下载文件)
### 下载到内存
try {
$bucket = "examplebucket-1250000000"; //存储桶，格式：BucketName-APPID
$key = "exampleobject"; //对象在存储桶中的位置，即称对象键
```

```
$result = $cosClient->getObject(array(
'Bucket' => $bucket,
'Key' => $key));
// 请求成功
echo($result['Body']);
} catch (\Exception $e) {
// 请求失败
echo "$e\n";
}

### 下载到本地
try {
$bucket = "examplebucket-1250000000"; //存储桶，格式：BucketName-APPID
$key = "exampleobject"; //对象在存储桶中的位置，即称对象键
$localPath = @"F:/exampleobject";//下载到本地指定路径
$result = $cosClient->getObject(array(
'Bucket' => $bucket,
'Key' => $key,
'SaveAs' => $localPath));
} catch (\Exception $e) {
// 请求失败
echo "$e\n";
}

### 指定下载范围
/*
* Range 字段格式为 'bytes=a-b'
*/
try {
$bucket = "examplebucket-1250000000"; //存储桶，格式：BucketName-APPID
$key = "exampleobject"; //对象在存储桶中的位置，即称对象键
$localPath = @"F:/exampleobject";//下载到本地指定路径
$result = $cosClient->getObject(array(
'Bucket' => $bucket,
'Key' => $key,
'Range' => 'bytes=0-10',
'SaveAs' => $localPath));
} catch (\Exception $e) {
// 请求失败
echo "$e\n";
}

### 设置返回header
try {
$bucket = "examplebucket-1250000000"; //存储桶，格式：BucketName-APPID
$key = "exampleobject"; //对象在存储桶中的位置，即称对象键
$localPath = @"F:/exampleobject";//下载到本地指定路径
$result = $cosClient->getObject(array(
'Bucket' => $bucket,
'Key' => $key,
'ResponseCacheControl' => 'string',
'ResponseContentDisposition' => 'string',
'ResponseContentEncoding' => 'string',
'ResponseContentLanguage' => 'string',
'ResponseContentType' => 'string',
'ResponseExpires' => 'mixed type: string (date format)|int (unix timestamp)|\DateTime',
'SaveAs' => $localPath));
} catch (\Exception $e) {
// 请求失败
echo "$e\n";
}

## getObjectUrl(获取文件UrL)
try {
$bucket = "examplebucket-1250000000"; //存储桶，格式：BucketName-APPID
$key = "exampleobject"; //对象在存储桶中的位置，即称对象键
$signedUrl = $cosClient->getObjectUrl($bucket, $key, '+10 minutes');
// 请求成功
echo $signedUrl;
} catch (\Exception $e) {
// 请求失败
```

```
print_r($e);
}
```

删除对象

```
# 删除object
## deleteObject
try {
$bucket = "examplebucket-1250000000"; //存储桶，格式：BucketName-APPID
$key = "exampleobject"; //对象在存储桶中的位置，即称对象键
$result = $cosClient->deleteObject(array(
'Bucket' => $bucket,
'Key' => $key,
'VersionId' => 'string'
));
// 请求成功
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo($e);
}
# 删除多个object
## deleteObjects
try {
$bucket = "examplebucket-1250000000"; //存储桶，格式：BucketName-APPID
$key1 = "exampleobject1"; //对象在存储桶中的位置，即称对象键
$key2 = "exampleobject2"; //对象在存储桶中的位置，即称对象键
$result = $cosClient->deleteObjects(array(
'Bucket' => $bucket,
'Objects' => array(
array(
'Key' => $key1,
),
array(
'Key' => $key2,
),
//...
),
));
// 请求成功
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo($e);
}
```

接口文档

最近更新时间: 2025-01-15 17:01:00

存储桶操作

简介

本文档提供关于存储桶的基本操作和访问控制列表（ACL）的相关 API 概览以及 SDK 示例代码。

基本操作

API	操作名	操作描述
PUT Bucket	创建存储桶	在指定账号下创建一个存储桶
HEAD Bucket	检索存储桶及其权限	检索存储桶是否存在且是否有权限访问
DELETE Bucket	删除存储桶	删除指定账号下的空存储桶

访问控制列表（ACL）

API	操作名	操作描述
PUT Bucket acl	设置存储桶 ACL	设置指定存储桶的访问权限控制列表（ACL）
GET Bucket acl	查询存储桶 ACL	获取指定存储桶的访问权限控制列表（ACL）

基本操作

创建存储桶

功能说明

在指定账号下创建一个存储桶（PUT Bucket）。

方法原型

```
public Guzzle\Service\Resource\Model createBucket(array $args = array());
```

请求示例

```
try {
$result = $cosClient->createBucket(array(
'Bucket' => 'examplebucket-1250000000' //格式：BucketName-APPID
));
// 请求成功
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo($e);
}
```

参数说明

参数名称	类型	描述	父节点
Bucket	String	存储桶名称，格式：BucketName-APPID	无

检索存储桶及其权限

功能说明

确认 Bucket 是否存在且是否有权限访问（HEAD Bucket）。

方法原型

```
public Guzzle\Service\Resource\Model headBucket(array $args = array());
```

请求示例

```
try {
$result = $cosClient->headBucket(array(
'Bucket' => 'examplebucket-1250000000' //格式：BucketName-APPID
));
// 请求成功
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo($e);
}
```

参数说明

参数名称	类型	描述	父节点
Bucket	String	存储桶名称，格式：BucketName-APPID	无

删除存储桶

功能说明

删除指定账号下的空存储桶。

方法原型

```
public Guzzle\Service\Resource\Model deleteBucket(array $args = array());
```

请求示例

```
try {
$result = $cosClient->deleteBucket(array(
'Bucket' => 'examplebucket-1250000000' //格式：BucketName-APPID
));
// 请求成功
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo($e);
}
```

参数说明

参数名称	类型	描述	父节点
Bucket	String	存储桶名称，格式：BucketName-APPID	无

访问控制列表

设置存储桶 ACL

功能说明

设置指定存储桶的访问权限控制列表（ACL）。

方法原型

```
public Guzzle\Service\Resource\Model putBucketAcl(array $args = array());
```

请求示例

```
try {
$result = $cosClient->putBucketAcl(array(
'Bucket' => 'examplebucket-1250000000', //格式：BucketName-APPID
'ACL' => 'private',
'Grants' => array(
array(
'Grantee' => array(
'DisplayName' => 'qcs::cam::uin/100000000001:uin/100000000001',
'ID' => 'qcs::cam::uin/100000000001:uin/100000000001',
'Type' => 'CanonicalUser',
),
'Permission' => 'FULL_CONTROL',
),
// ... repeated
),
'Owner' => array(
'DisplayName' => 'qcs::cam::uin/100000000001:uin/100000000001',
'ID' => 'qcs::cam::uin/100000000001:uin/100000000001',
)));
// 请求成功
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo "$e\n";
}
```

参数说明

参数名称	类型	描述	父节点
Bucket	String	存储桶名称，格式：BucketName-APPID	无
Grants	Array	ACL权限列表	无
Grant	Array	ACL权限信息	Grants
Grantee	Array	ACL权限信息	Grant
Type	String	所有者权限类型	Grantee
Permission	String	权限类型，可选值： FULL_CONTROL 、 WRITE 、 READ	Grant
ACL	String	整体权限类型，可选值：private 、 public-read 、public-read-write	无
Owner	String	存储桶所有者信息	无
DisplayName	String	权限所有者的名字信息	Grantee/Owner
ID	String	权限所有者 ID	Grantee/Owner

查询存储桶 ACL

功能说明

获取指定存储桶的访问权限控制列表（ACL）。

方法原型

```
public Guzzle\Service\Resource\Model getBucketAcl(array $args = array());
```

请求示例

```
try {
$result = $cosClient->getBucketAcl(array(
'Bucket' => 'examplebucket-1250000000' //格式：BucketName-APPID
));
// 请求成功
```

```
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo($e);
}
```

返回结果示例

```
Array
(
[data:protected] => Array
(
[Owner] => Array
(
[ID] => qcs::cam::uin/100000000001:uin/100000000001
[DisplayName] => qcs::cam::uin/100000000001:uin/100000000001
)

[Grants] => Array
(
[0] => Array
(
[Grantee] => Array
(
[ID] => qcs::cam::uin/100000000001:uin/100000000001
[DisplayName] => qcs::cam::uin/100000000001:uin/100000000001
)

[Permission] => FULL_CONTROL
)

)

[RequestId] => NWE3YzhjMTRfYzdhMzNiMGFfYjdiOF8yYzZmMzU=
)
)
```

返回结果说明

参数名称	类型	描述	父节点
Grants	Array	ACL 权限列表	无
Grant	Array	ACL 权限信息	Grants
Grantee	Array	ACL 权限信息	Grant
Permission	String	权限类型，可选值： FULL_CONTROL 、WRITE 、 READ	Grant
Owner	String	存储桶所有者信息	无
DisplayName	String	权限所有者的名字信息	Grantee/Owner
ID	String	权限所有者 ID	Grantee/Owner

对象操作

简介

本文档提供关于对象的简单操作、分块操作等其他操作相关的 API 概览以及 SDK 示例代码。

简单操作

API	操作名	操作描述

API	操作名	操作描述
GET Bucket（List Object）	查询对象列表	查询存储桶下的部分或者全部对象
GET Bucket Object Versions	查询对象及其历史版本列表	查询存储桶下的部分或者全部对象及其历史版本信息
PUT Object	简单上传对象	上传一个 Object（文件/对象）至 Bucket
POST Object	表单上传对象	使用表单请求上传对象
HEAD Object	查询对象元数据	查询 Object 的 Meta 信息
GET Object	下载对象	下载一个 Object（文件/对象）至本地
PUT Object - Copy	设置对象复制	复制文件到目标路径
DELETE Object	删除单个对象	在 Bucket 中删除指定 Object（文件/对象）
DELETE Multiple Object	删除多个对象	在 Bucket 中批量删除 Object（文件/对象）

分块操作

API	操作名	操作描述
List Multipart Uploads	查询分块上传	查询正在进行中的分块上传信息
Initiate Multipart Upload	初始化分块上传	初始化 Multipart Upload 上传操作
Upload Part	上传分块	分块上传对象
Upload Part - Copy	复制分块	将其他对象复制为一个分块
List Parts	查询已上传块	查询特定分块上传操作中的已上传的块
Complete Multipart Upload	完成分块上传	完成整个对象的分块上传
Abort Multipart Upload	终止分块上传	终止一个分块上传操作并删除已上传的块

其他操作

API	操作名	操作描述
PUT Object acl	设置对象 ACL	设置 Bucket 中某个 Object（文件/对象）的 ACL
GET Object acl	查询对象 ACL	查询 Object（文件/对象）的 ACL

简单操作

查询对象列表

功能说明

查询指定存储桶中所有的对象（List Object）。

方法原型

```
public Guzzle\Service\Resource\Model listObjects(array $args = array());
```

请求示例

```
try {
$result = $cosClient->listObjects(array(
'Bucket' => 'examplebucket-1250000000', //格式：BucketName-APPID
'Delimiter' => '/',
'EncodingType' => 'url',
```

```
'Marker' => 'doc/picture.jpg',
'Prefix' => 'doc',
'MaxKeys' => 1000,
));
// 请求成功
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo($e);
}
```

参数说明

参数名称	类型	描述	必填
Bucket	String	存储桶名称，格式：BucketName-APPID	是
Delimiter	String	默认为空，设置分隔符，比如设置`/`来模拟文件夹	否
EncodingType	String	默认不编码，规定返回值的编码方式，可选值：url	否
Marker	String	默认以 UTF-8 二进制顺序列出条目，标记返回 objects 的 list 的起点位置	否
Prefix	String	默认为空，对 object 的 key 进行筛选，匹配指定前缀（prefix）的 objects	否
MaxKeys	Int	最多返回的 objects 数量，默认为最大的1000	否

返回结果示例

```
Guzzle\Service\Resource\Model Object
(
[structure:protected] =>
[data:protected] => Array
(
[Name] => examplebucket-1250000000
[Prefix] => doc
[Marker] => doc/picture.jpg
[MaxKeys] => 10
[IsTruncated] => 1
[NextMarker] => doc/exampleobject
[Contents] => Array
(
[0] => Array
(
[Key] => doc/exampleobject
[LastModified] => 2019-02-14T12:20:40.000Z
[ETag] => "e37b429559d82e852af0b2f5b4d078ab72b90208"
[Size] => 6532594
[Owner] => Array
(
[ID] => 100000000001
[DisplayName] => 100000000001
)

[StorageClass] => STANDARD
)

[1] => Array
(
[Key] => doc/exampleobject2
[LastModified] => 2019-03-04T06:34:43.000Z
[ETag] => "988f9f28e68eba9b8c1f5f98ccce0a3c"
[Size] => 28
[Owner] => Array
(
[ID] => 100000000001
[DisplayName] => 100000000001
)

[StorageClass] => STANDARD
)
```

```
)
[RequestId] => NWNhMzM0MmZfOWUxYzBiMDlfOTk2YV83ZWE3ODE=
)

)
```

返回结果说明

参数名称	类型	描述	父节点
Name	String	存储桶名称，格式：BucketName-APPID	无
Delimiter	String	设置分隔符，比如设置`/`来模拟文件夹	无
EncodingType	String	规定返回值的编码方式	无
Marker	String	默认以 UTF-8 二进制顺序列出条目，标记返回 objects 的 list 的起点位置	无
Prefix	String	对 object 的 key 进行筛选，匹配指定前缀（prefix）的 objects	无
MaxKeys	Int	最多返回的 objects 数量，默认为最大的1000	无
IsTruncated	Int	表示返回的 objects 否被截断	无
Contents	Array	返回的对象列表	无
Content	Array	返回的对象属性，包含所有 objects 元信息的 list，包括 'ETag'，'StorageClass'，'Key'，'Owner'，'LastModified'，'Size' 等信息	Contents

查询对象及其历史版本列表

功能说明

查询存储桶下的部分或者全部对象及其历史版本信息。

方法原型

```
public Guzzle\Service\Resource\Model listObjectVersions(array $args = array());
```

请求示例

```
try {
$result = $cosClient->listObjectVersions(array(
'Bucket' => 'examplebucket-1250000000', //格式：BucketName-APPID
'Delimiter' => '/',
'EncodingType' => 'url',
'KeyMarker' => 'string',
'VersionIdMarker' => 'string',
'Prefix' => 'doc',
'MaxKeys' => 1000,
));
print_r($result);
} catch (\Exception $e) {
echo($e);
}
```

参数说明

参数名称	类型	描述	必填
Bucket	String	存储桶名称，由 BucketName-APPID 构成	是
Prefix	String	默认为空，对对象的对象键进行筛选，匹配 prefix 为前缀的对象	否
Delimiter	String	默认为空，设置分隔符，比如设置`/`来模拟文件夹	否
KeyMarker	String	默认以 UTF-8 二进制顺序列出条目，标记返回对象的 list 的 Key 的起点位置	否
VersionIdMarker	String	默认以 UTF-8 二进制顺序列出条目，标记返回对象的 list 的 VersionId 的起点位置	否

参数名称	类型	描述	必填
MaxKeys	Int	最多返回的对象数量，默认为最大的1000	否
EncodingType	String	默认不编码，规定返回值的编码方式，可选值：url	否

返回结果示例

```
Guzzle\Service\Resource\Model Object
(
[structure:protected] =>
[data:protected] => Array
(
[Name] => examplebucket-1250000000
[Prefix] => doc
[KeyMarker] => string
[VersionIdMarker] => string
[MaxKeys] => 10
[IsTruncated] => 1
[NextKeyMarker] => string
[NextVersionIdMarker] => string
[Versions] => Array
(
[0] => Array
(
[Key] => doc/exampleobject1
[VersionId] => null
[IsLatest] => 1
[LastModified] => 2019-06-13T09:24:52.000Z
[ETag] => "96e79218965eb72c92a549dd5a330112"
[Size] => 6
[StorageClass] => STANDARD
[Owner] => Array
(
[UID] => 1251668577
)
)

[1] => Array
(
[Key] => doc/exampleobject2
[VersionId] => MTg0NDUxODMyMTE2ODY0OTExOTk
[IsLatest] => 1
[LastModified] => 2019-06-18T12:47:03.000Z
[ETag] => "698d51a19d8a121ce581499d7b701668"
[Size] => 3
[StorageClass] => STANDARD
[Owner] => Array
(
[UID] => 1251668577
)
)
)
[RequestId] => NWQwOGVkZGRfMjViMjU4NjRfODNjN18xMTE5YWI4
)

)
```

返回结果说明

参数名称	类型	描述	父节点
Name	String	存储桶名称，格式：BucketName-APPID	无
Delimiter	String	设置分隔符，比如设置`/`来模拟文件夹	无
EncodingType	String	规定返回值的编码方式	无
KeyMarker	String	默认以 UTF-8 二进制顺序列出条目，标记返回对象的 list 的 Key 的起点位置	无

版权所有：腾讯云金融专区

参数名称	类型	描述	父节点
VersionIdMarker	String	默认以 UTF-8 二进制顺序列出条目，标记返回对象的 list 的 VersionId 的起点位置	无
NextKeyMarker	String	当 IsTruncated 为 true 时，标记下一次返回对象的 list 的 Key 的起点位置	无
NextVersionIdMarker	String	当 IsTruncated 为 true 时，标记下一次返回对象的 list 的 VersionId 的起点位置	无
Prefix	String	对 object 的 key 进行筛选，匹配指定前缀（prefix）的 objects	无
MaxKeys	Int	最多返回的 objects 数量，默认为最大的1000	无
IsTruncated	Int	表示返回的 objects 否被截断	无
Versions	Array	包含所有多个版本对象元数据的 list	无
Version	Array	包含所有多个版本对象元数据的 list，包括 'ETag'，'StorageClass'，'Key'，'VersionId'，'IsLatest'，'Owner'，'LastModified'，'Size' 等信息	Versions
CommonPrefixes	Array	所有以 Prefix 开头，以 Delimiter 结尾的对象被归到同一类	无

简单上传对象

功能说明

上传对象到指定的存储桶中（PUT Object），最大支持5G大小的文件，如果需要上传超过5G的文件，建议使用高级接口中的复合上传接口。

方法原型

```
public Guzzle\Service\Resource\Model putObject(array $args = array())
```

请求示例

```
try {
$result = $cosClient->putObject(array(
'Bucket' => 'examplebucket-1250000000', //格式：BucketName-APPID
'Key' => 'exampleobject',
'Body' => fopen('/data/exampleobject', 'rb'),
/*
'ACL' => 'string',
'CacheControl' => 'string',
'ContentDisposition' => 'string',
'ContentEncoding' => 'string',
'ContentLanguage' => 'string',
'ContentLength' => integer,
'ContentType' => 'string',
'Expires' => 'string',
'GrantFullControl' => 'string',
'GrantRead' => 'string',
'GrantWrite' => 'string',
'Metadata' => array(
'string' => 'string',
),
'ContentMD5' => 'string',
'ServerSideEncryption' => 'string',
'StorageClass' => 'string'
*/
));
// 请求成功
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo($e);
}
```

参数说明

参数名称	类型	描述	必填
Bucket	String	存储桶名称，格式：BucketName-APPID	是

参数名称	类型	描述	必填
Key	String	对象键（Key）是对象在存储桶中的唯一标识。例如，在对象的访问域名 examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/pic.jpg 中，对象键为 doc/pic.jpg	否
ACL	String	设置对象的 ACL，如 private、public-read	否
Body	File/String	上传的内容	是
CacheControl	String	缓存策略，设置 Cache-Control	否
ContentDisposition	String	文件名称，设置 Content-Disposition	否
ContentEncoding	String	编码格式，设置 Content-Encoding	否
ContentLanguage	String	语言类型，设置 Content-Language	否
ContentLength	Int	设置传输长度	否
ContentType	String	内容类型，设置 Content-Type	否
Expires	String	设置 Content-Expires	否
Metadata	Array	用户自定义的文件元信息	否
StorageClass	String	文件的存储类型，默认值：STANDARD	否
ContentMD5	String	设置上传文件的 MD5 值用于校验	否
ServerSideEncryption	String	服务端加密方法	否

返回结果示例

```
Guzzle\Service\Resource\Model Object
(
[structure:protected] =>
[data:protected] => Array
(
[ETag] => "698d51a19d8a121ce581499d7b701668"
[VersionId] => MTg0NDUxODMyMTE2ODY0OTExOTk
[RequestId] => NWQwOGRkNDdfMjJiMjU4NjRfNzVjXzEwNmVjY2M=
[ObjectURL] => http://imgcache.finance.cloud.tencent.com:80lewzylucd2-1251668577.cos.ap-chengdu.myqcloud.com/123
)

)
```

返回结果说明

参数名称	类型	描述	父节点
ETag	String	上传文件的 MD5 值	无
VersionId	String	开启多版本后，文件的版本号	无

查询对象元数据

功能说明

查询 Object 的 Meta 信息（HEAD Object）。

方法原型

```
public Guzzle\Service\Resource\Model headObject(array $args = array());
```

请求示例

```
$cosClient = new Qcloud\Cos\Client(
array(
'schema' => 'http', // 协议头部，默认为http
'region' => $region, // Region
```

```
'endpoint' => $endpoint, // Endpoint
'credentials'=> array(
'secretId' => $secretId ,
'secretKey' => $secretKey)));
try {
$result = $cosClient->headObject(array(
'Bucket' => 'examplebucket-1250000000', //格式：BucketName-APPID
'Key' => 'exampleobject',
));
// 请求成功
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo($e);
}
```

参数说明

参数名称	类型	描述	必填
Bucket	String	存储桶名称，格式：BucketName-APPID	是
Key	String	对象键（Key）是对象在存储桶中的唯一标识。例如，在对象的访问域名examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/pic.jpg 中，对象键为 doc/pic.jpg	是
VersionId	String	开启多版本后，指定文件的具体版本	否

返回结果示例

```
Guzzle\Service\Resource\Model Object
(
[structure:protected] =>
[data:protected] => Array
(
[DeleteMarker] =>
[AcceptRanges] =>
[Expiration] =>
[Restore] =>
[LastModified] => Tue, 02 Apr 2019 12:38:09 GMT
[ContentLength] => 238186
[ETag] => "af9f3b8eaf64473278909183abba1e31"
[MissingMeta] =>
[VersionId] =>
[CacheControl] =>
[ContentDisposition] =>
[ContentEncoding] =>
[ContentLanguage] =>
[ContentType] => text/plain; charset=utf-8
[Expires] =>
[ServerSideEncryption] =>
[Metadata] => Array
(
[md5] => af9f3b8eaf64473278909183abba1e31
)
[SSECustomerAlgorithm] =>
[SSECustomerKeyMD5] =>
[SSEKMSKeyId] =>
[StorageClass] =>
[RequestCharged] =>
[ReplicationStatus] =>
[RequestId] => NWNhMzU3Y2ZfMzFhYzM1MGFfODdhMF8xOTExM2U=
)

)
```

返回结果说明

参数名称	类型	描述	父节点

参数名称	类型	描述	父节点
CacheControl	String	缓存策略，设置 Cache-Control	无
ContentDisposition	String	文件名称，设置 Content-Disposition	无
ContentEncoding	String	编码格式，设置 Content-Encoding	无
ContentLanguage	String	语言类型，设置 Content-Language	无
ContentLength	Int	设置传输长度	无
ContentType	String	内容类型，设置 Content-Type	无
Metadata	Array	用户自定义的文件元信息	无
StorageClass	String	文件的存储类型，默认值：STANDARD	无
ServerSideEncryption	String	服务端加密方法	无
ETag	String	文件的MD5值	无
Restore	String	归档文件的回热信息	无

下载对象

功能说明

下载对象到本地（GET Object）。

方法原型

```
public Guzzle\Service\Resource\Model getObject(array $args = array());
```

请求示例

```
try {
$result = $cosClient->getObject(array(
'Bucket' => 'examplebucket-1250000000', //格式：BucketName-APPID
'Key' => 'exampleobject',
'SaveAs' => '/data/exampleobject',
/*
'Range' => 'bytes=0-10',
'VersionId' => 'string',
'ResponseCacheControl' => 'string',
'ResponseContentDisposition' => 'string',
'ResponseContentEncoding' => 'string',
'ResponseContentLanguage' => 'string',
'ResponseContentType' => 'string',
'ResponseExpires' => 'string',
*/
));
// 请求成功
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo($e);
}
```

参数说明

参数名称	类型	描述	必填
Bucket	String	存储桶名称，格式：BucketName-APPID	是
Key	String	对象键（Key）是对象在存储桶中的唯一标识。例如，在对象的访问域名examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/pic.jpg 中，对象键为 doc/pic.jpg	是
SaveAs	String	保存到本地的本地文件路径	否

参数名称	类型	描述	必填
VersionId	String	开启多版本后，指定文件的具体版本	否
Range	String	设置下载文件的范围，格式为 bytes=first-last	否
ResponseCacheControl	String	设置响应头部 Cache-Control	否
ResponseContentDisposition	String	设置响应头部 Content-Disposition	否
ResponseContentEncoding	String	设置响应头部 Content-Encoding	否
ResponseContentLanguage	String	设置响应头部 Content-Language	否
ResponseContentType	String	设置响应头部 Content-Type	否
ResponseExpires	String	设置响应头部 Content-Expires	否

返回结果示例

```
Guzzle\Service\Resource\Model Object
(
[structure:protected] =>
[data:protected] => Array
(
[Body] =>
[DeleteMarker] =>
[AcceptRanges] => bytes
[Expiration] =>
[Restore] =>
[LastModified] => Tue, 02 Apr 2019 20:38:09 GMT
[ContentLength] => 238186
[ETag] => "af9f3b8eaf64473278909183abba1e31"
[MissingMeta] =>
[VersionId] =>
[CacheControl] =>
[ContentDisposition] =>
[ContentEncoding] =>
[ContentLanguage] =>
[ContentRange] =>
[ContentType] => text/plain; charset=utf-8
[Expires] =>
[WebsiteRedirectLocation] =>
[ServerSideEncryption] =>
[Metadata] => Array
(
[md5] => af9f3b8eaf64473278909183abba1e31
)

[SSECustomerAlgorithm] =>
[SSECustomerKeyMD5] =>
[SSEKMSKeyId] =>
[StorageClass] =>
[RequestCharged] =>
[ReplicationStatus] =>
[RequestId] => NWNhNDBmYzBfNmNhYjM1MGFfMmUzYzFfMWIzMDYz
)

)
```

返回结果说明

参数名称	类型	描述	父节点
Body	File/String	下载内容	无
ETag	String	文件的 MD5 值	无
Expires	String	Content-Expires	无

参数名称	类型	描述	父节点
Metadata	Array	用户自定义的文件元信息	无
StorageClass	String	文件的存储类型，默认值：STANDARD	无
ContentMD5	String	设置上传文件的 MD5 值用于校验	无
ServerSideEncryption	String	服务端加密方法	无
CacheControl	String	缓存策略，设置 Cache-Control	无
ContentDisposition	String	文件名称，设置 Content-Disposition	无
ContentEncoding	String	编码格式，设置 Content-Encoding	无
ContentLanguage	String	语言类型，设置 Content-Language	无
ContentLength	Int	设置传输长度	无
ContentType	String	内容类型，设置 Content-Type	无
Metadata	Array	用户自定义的文件元信息	无
Restore	String	归档文件的回热信息	无

设置对象复制

将一个对象复制到目标路径（PUT Object - Copy）。

方法原型

```
public Guzzle\Service\Resource\Model copyObject(array $args = array());
```

请求示例

```
try {
$result = $cosClient->copyObject(array(
'Bucket' => 'examplebucket-1250000000', //格式：BucketName-APPID
'Key' => 'exampleobject',
'CopySource' => 'examplebucket2-1250000000.cos.ap-guangzhou.myqcloud.com/exampleobject',
/*
'MetadataDirective' => 'string',
'ACL' => 'string',
'CacheControl' => 'string',
'ContentDisposition' => 'string',
'ContentEncoding' => 'string',
'ContentLanguage' => 'string',
'ContentLength' => integer,
'ContentType' => 'string',
'Expires' => 'string',
'GrantFullControl' => 'string',
'GrantRead' => 'string',
'GrantWrite' => 'string',
'Metadata' => array(
'string' => 'string',
),
'ContentMD5' => 'string',
'ServerSideEncryption' => 'string',
'StorageClass' => 'string'
*/
));
// 请求成功
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo($e);
}
```

参数说明

参数名称	类型	描述	必填
Bucket	String	存储桶名称，格式：BucketName-APPID	是
Key	String	对象键（Key）是对象在存储桶中的唯一标识。例如，在对象的访问域名examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/pic.jpg 中，对象键为 doc/pic.jpg	是
CopySource	String	描述拷贝源文件的路径，包含 Appid、Bucket、Key、Region，例如 examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/pic.jpg	是
MetadataDirective	String	可选值为 Copy、Replaced。设置为 Copy 时，忽略设置的用户元数据信息直接复制，设置为 Replaced 时，按设置的元信息修改元数据，当目标路径和源路径一样时，必须设置为 Replaced	否

删除单个对象

功能说明

在存储桶中删除指定 Object（文件/对象）。

方法原型

```
public Guzzle\Service\Resource\Model deleteObject(array $args = array());
```

请求示例

```
try {
$result = $cosClient->deleteObject(array(
'Bucket' => 'examplebucket-1250000000', //格式：BucketName-APPID
'Key' => 'exampleobject',
'VersionId' => 'string'
));
// 请求成功
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo($e);
}
```

参数说明

参数名称	类型	描述	必填
Bucket	String	存储桶名称，格式：BucketName-APPID	是
Key	String	对象键（Key）是对象在存储桶中的唯一标识。例如，在对象的访问域名examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/pic.jpg 中，对象键为 doc/pic.jpg	是
VersionId	String	删除文件的版本号	否

删除多个对象

功能说明

在存储桶中批量删除 Object（文件/对象）。

方法原型

```
public Guzzle\Service\Resource\Model deleteObjects(array $args = array());
```

请求示例

```
try {
$result = $cosClient->deleteObjects(array(
'Bucket' => 'examplebucket-1250000000', //格式：BucketName-APPID
'Objects' => array(
array(
'Key' => 'exampleobject',
'VersionId' => 'string'
```

```
),
// ... repeated
),
));
// 请求成功
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo($e);
}
```

参数说明

参数名称	类型	描述	必填
Bucket	String	存储桶名称，格式：BucketName-APPID	是
Objects	Array	删除对象列表	是
Object	Array	删除的对象	是
Key	String	对象键（Key）是对象在存储桶中的唯一标识。例如，在对象的访问域名examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/pic.jpg 中，对象键为 doc/pic.jpg	是
VersionId	String	删除文件的版本号	否

返回结果示例

```
Guzzle\Service\Resource\Model Object
(
[structure:protected] =>
[data:protected] => Array
(
[Deleted] => Array
(
[0] => Array
(
[Key] => exampleobject1
)
)
[Errors] => Array
(
[0] => Array
(
[Key] => exampleobject2
[Code] =>
[Message] =>
)
)
[RequestId] => NWNhZWYzYWNfMTlhYTk0MGFfNGRjX2MzZTVhOQ==
)
 )
```

返回结果说明

参数名称	类型	描述	父节点
Deleted	Array	成功删除的对象的列表	无
Errors	Array	失败删除的对象的列表	无
Key	String	对象键	Deleted/Errors
Code	String	失败错误码	Errors
Message	String	失败错误信息	Errors

分块操作

查询分片上传

功能说明

查询指定存储桶中正在进行的分片上传（List Multipart Uploads）。

方法原型

```
public Guzzle\Service\Resource\Model listMultipartUploads(array $args = array());
```

请求示例

```
try {
$result = $cosClient->listMultipartUploads(array(
'Bucket' => 'examplebucket-1250000000', //格式：BucketName-APPID
'Delimiter' => '/',
'EncodingType' => 'url',
'KeyMarker' => 'string',
'UploadIdMarker' => 'string',
'Prefix' => 'prfix',
'MaxUploads' => 1000,
));
// 请求成功
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo($e);
}
```

参数说明

参数名称	类型	描述	必填
Bucket	String	存储桶名称，格式：BucketName-APPID	是
Delimiter	String	默认为空，设置分隔符，比如设置`/`来模拟文件夹	否
EncodingType	String	默认不编码，规定返回值的编码方式，可选值：url	否
KeyMarker	String	标记返回 parts 的 list 的起点位置	否
UploadIdMarker	String	标记返回 parts 的 list 的起点位置	否
Prefix	String	默认为空，对 parts 的 key 进行筛选，匹配指定前缀（prefix）的 objects	否
MaxUploads	Int	最多返回的 parts 数量，默认为最大的1000	否

返回结果示例

```
Guzzle\Service\Resource\Model Object
(
[structure:protected] =>
[data:protected] => Array
(
[Bucket] => examplebucket-1250000000
[EncodingType] =>
[KeyMarker] =>
[UploadIdMarker] =>
[MaxUploads] => 1000
[Prefix] =>
[IsTruncated] =>
[Uploads] => Array
(
[0] => Array
(
[Key] => exampleobject
[UploadId] => 1551693693b1e6d0e00eec30c534059865ec89c9393028b60bfaf167e9420524b25eeb2940
[Initiator] => Array
(
[ID] => qcs::cam::uin/100000000001:uin/100000000001
```

```
[DisplayName] => 100000000001
)

[Owner] => Array
(
[ID] => qcs::cam::uin/100000000001:uin/100000000001
[DisplayName] => 100000000001
)

[StorageClass] => STANDARD
[Initiated] => 2019-03-04T10:01:33.000Z
)

[1] => Array
(
[Key] => exampleobject
[UploadId] => 155374001100563fe0e9d37964d53077e54e9d392bce78f630359cd3288e62acee2b719534
[Initiator] => Array
(
[ID] => qcs::cam::uin/100000000001:uin/100000000001
[DisplayName] => 100000000001
)

[Owner] => Array
(
[ID] => qcs::cam::uin/100000000001:uin/100000000001
[DisplayName] => 100000000001
)

[StorageClass] => STANDARD
[Initiated] => 2019-03-28T02:26:51.000Z
)

)

[RequestId] => NWNhNDJmNzBfZWFhZDM1MGFfMjYyM2FfMWIyNzhh
)

)
```

返回结果说明

参数名称	类型	描述	父节点
Bucket	String	存储桶名称，格式：BucketName-APPID	无
IsTruncated	Int	表示返回的 objects 否被截断	无
Uploads	Array	返回的分块列表	无
Upload	Array	返回的分块属性	Uploads
Key	String	对象健名	Upload
UploadId	String	对象的分块上传 ID	Upload
Initiator	String	初始化该分片的操作者	Upload
Owner	String	分块拥有者	Upload
StorageClass	String	分块存储类型	Upload
Initiated	String	分块初始化时间	Upload

分片上传对象

分片上传对象可包括的操作：

- 分片上传对象： 初始化分片上传， 上传分片块， 完成分块上传。
- 分片续传：查询已上传块， 上传分片块，完成分块上传。
- 删除已上传分片块。

初始化分片上传

功能说明

初始化 Multipart Upload 上传操作（Initiate Multipart Upload）。

方法原型

```
public Guzzle\Service\Resource\Model createMultipartUpload(array $args = array());
```

请求示例

```
try {
$result = $cosClient->createMultipartUpload(array(
'Bucket' => 'examplebucket-1250000000', //格式：BucketName-APPID
'Key' => 'exampleobject',
/*
'CacheControl' => 'string',
'ContentDisposition' => 'string',
'ContentEncoding' => 'string',
'ContentLanguage' => 'string',
'ContentLength' => integer,
'ContentType' => 'string',
'Expires' => 'string',
'Metadata' => array(
'string' => 'string',
),
'StorageClass' => 'string'
*/
));
// 请求成功
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo($e);
}
```

参数说明

参数名称	类型	描述	必填
Bucket	String	存储桶名称，格式：BucketName-APPID	是
Key	String	对象键（Key）是对象在存储桶中的唯一标识。例如，在对象的访问域名examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/pic.jpg 中，对象键为 doc/pic.jpg	是
CacheControl	String	缓存策略，设置 Cache-Control	否
ContentDisposition	String	文件名称，设置 Content-Disposition	否
ContentEncoding	String	编码格式，设置 Content-Encoding	否
ContentLanguage	String	语言类型，设置 Content-Language	否
ContentLength	Int	设置传输长度	否
ContentType	String	内容类型，设置 Content-Type	否
Expires	String	设置 Content-Expires	否
Metadata	Array	用户自定义的文件元信息	否
StorageClass	String	文件的存储类型，默认值：STANDARD	否
ContentMD5	String	设置上传文件的 MD5 值用于校验	否
ServerSideEncryption	String	服务端加密方法	否

返回结果示例

```
Guzzle\Service\Resource\Model Object
(
[structure:protected] =>
[data:protected] => Array
(
[Bucket] => examplebucket-1250000000
[Key] => exampleobject
[UploadId] => 1554277569b3e83df05c730104c325eb7b56000449fb7d51300b0728aacde02a6ea7f6c033
[RequestId] => NWNhNDY0YzFfMmZiNTM1MGFfNTM2YV8xYjliMTg=
)

)
```

返回结果说明

参数名称	类型	描述	父节点
Bucket	String	存储桶名称，格式：BucketName-APPID	无
Key	String	对象键	无
UploadId	String	对象分块上传的ID	无

查询已上传块

功能说明

查询特定分块上传操作中的已上传的块（List Parts）。

方法原型

```
public Guzzle\Service\Resource\Model listParts(array $args = array());
```

请求示例

```
try {
$result = $cosClient->listParts(array(
'Bucket' => 'examplebucket-1250000000', //格式：BucketName-APPID
'Key' => 'exampleobject',
'UploadId' => 'NWNhNDY0YzFfMmZiNTM1MGFfNTM2YV8xYjliMTg',
'PartNumberMarker' => 1,
'MaxParts' => 1000,
));
// 请求成功
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo($e);
}
```

参数说明

参数名称	类型	描述	必填
Bucket	String	存储桶名称，格式：BucketName-APPID	是
Key	String	对象键	是
UploadId	String	对象分块上传的 ID	是
PartNumberMarker	Int	标记返回 parts 的 list 的起点位置	否
MaxParts	Int	最多返回的 parts 数量，默认最大值为1000	否

返回结果示例

```
Guzzle\Service\Resource\Model Object
(
```

```
[structure:protected] =>
[data:protected] => Array
(
[Bucket] => examplebucket-1250000000
[Key] => exampleobject
[UploadId] => 1554279643cf19d71bb5fb0d29613e5541131f3a96387d9e168cd939c23a3d608c9eb94707
[Owner] => Array
(
[ID] => 1250000000
[DisplayName] => 1250000000
)
[PartNumberMarker] => 1
[Initiator] => Array
(
[ID] => qcs::cam::uin/100000000001:uin/100000000001
[DisplayName] => 100000000001
)
[StorageClass] => Standard
[MaxParts] => 1000
[IsTruncated] =>
[Parts] => Array
(
[0] => Array
(
[PartNumber] => 2
[LastModified] => 2019-04-03T08:21:28.000Z
[ETag] => "b948e77469189ac94b98e09755a6dba9"
[Size] => 1048576
)
[1] => Array
(
[PartNumber] => 3
[LastModified] => 2019-04-03T08:21:22.000Z
[ETag] => "9e5060e2994ec8463bfbebd442fdff16"
[Size] => 1048576
)
)
[RequestId] => NWNhNDZkNTJfOGNiMjM1MGFfMTRlYl8xYmJiOTU=
)

)
```

返回结果说明

参数名称	类型	描述	父节点
Bucket	String	存储桶名称，格式：BucketName-APPID	无
Key	String	对象键	无
UploadId	String	对象分块上传的 ID	无
IsTruncated	Int	表示返回的 objects 否被截断	无
PartNumberMarker	Int	标记返回 parts 的 list 的起点位置	无
MaxParts	Int	最多返回的 parts 数量，默认最大值为1000	无
Initiator	String	初始化该分片的操作者	无
Parts	Array	返回的分块列表	无
Part	Array	返回的分块属性	Parts
PartNumber	Int	分块标号	Part
LastModified	String	分块的最后上传时间	Part
ETag	String	分块的 MD5 值	Part
Size	String	分块的大小	Part

上传分块

分块上传文件（Upload Part）。

方法原型

```
public Guzzle\Service\Resource\Model uploadPart(array $args = array());
```

请求示例

```
try {
$result = $cosClient->uploadPart(array(
'Bucket' => 'examplebucket-1250000000', //格式：BucketName-APPID
'Key' => 'exampleobject',
'Body' => 'string',
'UploadId' => 'NWNhNDY0YzFfMmZiNTM1MGFfNTM2YV8xYjliMTg',
'PartNumber' => integer,
/*
'ContentMD5' => 'string',
'ContentLength' => integer,
*/
));
// 请求成功
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo($e);
}
```

参数说明

参数名称	类型	描述	必填
Bucket	String	存储桶名称，格式：BucketName-APPID	是
Key	String	对象键	是
UploadId	String	对象分块上传的 ID	是
Body	File/String	上传的内容	是
PartNumber	Int	上传分块的编号	是
ContentLength	Int	设置传输长度	否
ContentMD5	String	设置上传文件的 MD5 值用于校验	否

返回结果示例

```
Guzzle\Service\Resource\Model Object
(
[structure:protected] =>
[data:protected] => Array
(
[ETag] => "96e79218965eb72c92a549dd5a330112"
[RequestId] => NWNhNDdjYWFfNjNhYjM1MGFfMjk2NF8xY2ViMWM=
)

)
```

返回结果说明

参数名称	类型	描述	父节点
ETag	String	分块的 MD5 值	无

完成分块上传

功能说明

完成整个文件的分块上传（Complete Multipart Upload）。

方法原型

```
public Guzzle\Service\Resource\Model completeMultipartUpload(array $args = array());
```

请求示例

```
try {
$result = $cosClient->completeMultipartUpload(array(
'Bucket' => 'examplebucket-1250000000', //格式：BucketName-APPID
'Key' => 'exampleobject',
'UploadId' => 'string',
'Parts' => array(
array(
'ETag' => 'string',
'PartNumber' => integer,
),
array(
'ETag' => 'string',
'PartNumber' => integer,
)),
// ... repeated
));
// 请求成功
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo($e);
}
```

参数说明

参数名称	类型	描述	必填
Bucket	String	存储桶名称，格式：BucketName-APPID	是
Key	String	对象键	是
UploadId	String	对象分块上传的 ID	是
Parts	Array	分块信息列表	是
Part	Array	上传分块的内容信息	是
ETag	String	分块内容的 MD5	是
PartNumber	Int	分块编号	是

终止分块上传

功能说明

终止一个分块上传操作并删除已上传的块（Abort Multipart Upload）。

方法原型

```
public Guzzle\Service\Resource\Model abortMultipartUpload(array $args = array());
```

请求示例

```
try {
$result = $cosClient->abortMultipartUpload(array(
'Bucket' => 'examplebucket-1250000000', //格式：BucketName-APPID
'Key' => 'exampleobject',
'UploadId' => 'string',
));
// 请求成功
print_r($result);
```

```
} catch (\Exception $e) {
// 请求失败
echo($e);
}
```

参数说明

参数名称	类型	描述	必填
Bucket	String	存储桶名称，格式：BucketName-APPID	是
Key	String	对象键	是
UploadId	String	对象分块上传的 ID	是

其他操作

设置对象 ACL

功能说明

设置指定对象访问权限控制列表（ACL）（PUT Object acl）。

方法原型

```
public Guzzle\Service\Resource\Model putObjectAcl(array $args = array());
```

请求示例

```
try {
$result = $cosClient->putObjectAcl(array(
'Bucket' => 'examplebucket-1250000000', //格式：BucketName-APPID
'Key' => 'exampleobject',
'ACL' => 'private',
'Grants' => array(
array(
'Grantee' => array(
'DisplayName' => 'qcs::cam::uin/100000000001:uin/100000000001',
'ID' => 'qcs::cam::uin/100000000001:uin/100000000001',
'Type' => 'CanonicalUser',
),
'Permission' => 'FULL_CONTROL',
),
// ... repeated
),
'Owner' => array(
'DisplayName' => 'qcs::cam::uin/100000000001:uin/100000000001',
'ID' => 'qcs::cam::uin/100000000001:uin/100000000001',
)));
// 请求成功
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo "$e\n";
}
```

参数说明

参数名称	类型	描述	必填
Bucket	String	存储桶名称，格式：BucketName-APPID	是
Key	String	对象键	是
Grants	Array	ACL权限列表	否
Grant	Array	ACL权限信息	否

参数名称	类型	描述	必填
Grantee	Array	ACL权限信息	否
Type	String	所有者权限类型	否
Permission	String	权限类型，可选值: FULL_CONTROL 、WRITE 、READ	否
ACL	String	整体权限类型，可选值: private 、 public-read	否
Owner	String	存储桶所有者信息	否
DisplayName	String	权限所有者的名字信息	否
ID	String	权限所有者 ID	否

获取对象 ACL

功能说明

获取指定对象的访问权限控制列表（ACL）（GET Object acl）。

方法原型

```
public Guzzle\Service\Resource\Model getObjectAcl(array $args = array());
```

请求示例

```
try {
$result = $cosClient->getObjectAcl(array(
'Bucket' => 'examplebucket-1250000000' //格式：BucketName-APPID
'Key' => 'exampleobject',
));
// 请求成功
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo($e);
}
```

返回结果示例

```
Array
(
[data:protected] => Array
(
[Owner] => Array
(
[ID] => qcs::cam::uin/100000000001:uin/100000000001
[DisplayName] => qcs::cam::uin/100000000001:uin/100000000001
)

[Grants] => Array
(
[0] => Array
(
[Grantee] => Array
(
[ID] => qcs::cam::uin/100000000001:uin/100000000001
[DisplayName] => qcs::cam::uin/100000000001:uin/100000000001
)

[Permission] => FULL_CONTROL
)

)

[RequestId] => NWE3YzhjMTRfYzdhMzNiMGFfYjdiOF8yYzZmMzU=
)
)
```

返回结果说明

参数名称	类型	描述	父节点
Grants	Array	ACL权限列表	无
Grant	Array	ACL权限信息	Grants
Grantee	Array	ACL权限信息	Grant
Permission	String	权限类型，可选值: FULL_CONTROL 、WRITE 、 READ	Grant
Owner	String	存储桶所有者信息	无
DisplayName	String	权限所有者的名字信息	Grantee / Owner
ID	String	权限所有者 ID	Grantee / Owner

高级接口（推荐）

该小节主要讲述由 COS 提供的封装了上传和复制操作的高级接口，用户只需要设置相应的参数，该接口内部会根据文件大小决定是进行简单上传（复制）还是分片上传（复制），使用接口前请确认已完成了 快速入门 中指引的初始化步骤。

复合上传

功能说明

该接口内部会根据文件大小，对小文件调用简单上传接口，对大文件调用分块上传接口。

请求示例

```
try {
$result = $cosClient->Upload(
$bucket = 'examplebucket-1250000000', //格式：BucketName-APPID
$key = 'exampleobject',
$body = fopen('/data/exampleobject', 'rb')
/*
$options = array(
'ACL' => 'string',
'CacheControl' => 'string',
'ContentDisposition' => 'string',
'ContentEncoding' => 'string',
'ContentLanguage' => 'string',
'ContentLength' => integer,
'ContentType' => 'string',
'Expires' => 'string',
'GrantFullControl' => 'string',
'GrantRead' => 'string',
'GrantWrite' => 'string',
'Metadata' => array(
'string' => 'string',
),
'ContentMD5' => 'string',
'ServerSideEncryption' => 'string',
'StorageClass' => 'string'
)
*/
);
// 请求成功
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo($e);
}
```

复合复制

功能说明

该接口内部会根据文件大小，对小文件调用设置对象复制接口，对大文件调用分块复制接口。

请求示例

```
try {
$result = $cosClient->Copy(
$bucket = 'examplebucket-1250000000', //格式：BucketName-APPID
$key = 'exampleobject',
$copysource = 'examplebucket2-1250000000.cos.ap-guangzhou.myqcloud.com/exampleobject'
/*
$options = array(
'ACL' => 'string',
'MetadataDirective' => 'string',
'CacheControl' => 'string',
'ContentDisposition' => 'string',
'ContentEncoding' => 'string',
'ContentLanguage' => 'string',
'ContentLength' => integer,
'ContentType' => 'string',
'Expires' => 'string',
'GrantFullControl' => 'string',
'GrantRead' => 'string',
'GrantWrite' => 'string',
'Metadata' => array(
'string' => 'string',
),
'ContentMD5' => 'string',
'ServerSideEncryption' => 'string',
'StorageClass' => 'string'
)
*/
);
// 请求成功
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo($e);
}
```

存储桶管理

简介

本文档提供关于跨域访问、生命周期、版本控制、跨地域复制相关的 API 概览以及 SDK 示例代码。

跨域访问

API	操作名	操作描述
PUT Bucket cors	设置跨域配置	设置存储桶的跨域名访问权限
GET Bucket cors	查询跨域配置	查询存储桶的跨域名访问配置信息
DELETE Bucket cors	删除跨域配置	删除存储桶的跨域名访问配置信息

版本控制

API	操作名	操作描述
PUT Bucket versioning	设置版本控制	设置存储桶的版本控制功能
GET Bucket versioning	查询版本控制	查询存储桶的版本控制信息

跨地域复制

API	操作名	操作描述
PUT Bucket replication	设置跨地域复制	设置存储桶的跨地域复制规则
GET Bucket replication	查询跨地域复制	查询存储桶的跨地域复制规则
DELETE Bucket replication	删除跨地域复制	删除存储桶的跨地域复制规则

跨域访问

设置跨域配置

功能说明

设置指定存储桶的跨域名访问配置信息（PUT Bucket cors）。

方法原型

```
public Guzzle\Service\Resource\Model putBucketCors(array $args = array());
```

请求示例

```
try {
$result = $cosClient->putBucketCors(array(
'Bucket' => 'examplebucket-1250000000', //格式：BucketName-APPID
'CORSRules' => array(
array(
'AllowedHeaders' => array('*',),
'AllowedMethods' => array('Put', ),
'AllowedOrigins' => array('*', ),
'ExposeHeaders' => array('*', ),
'MaxAgeSeconds' => 1,
),
// ... repeated
)
));
// 请求成功
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo "$e\n";
}
```

参数说明

参数名称	类型	描述	必填
Bucket	String	存储桶名称，格式：BucketName-APPID	是
CORSRules	Array	跨域信息列表	是
CORSRule	Array	跨域信息	是
AllowedMethods	String	允许的 HTTP 操作，枚举值：GET，PUT，HEAD，POST，DELETE	是
AllowedOrigins	String	允许的访问来源，支持通配符 `*`，格式为：协议://域名[:端口]如：`http://imgcache.finance.cloud.tencent.com:80www.qq.com`	是
AllowedHeaders	String	在发送 OPTIONS 请求时告知服务端，接下来的请求可以使用哪些自定义的 HTTP 请求头部，支持通配符`*`	否
ExposeHeaders	String	设置浏览器可以接收到的来自服务器端的自定义头部信息	否
MaxAgeSeconds	Int	设置 OPTIONS 请求得到结果的有效期	否
ID	String	配置规则的 ID	是

查询跨域配置

功能说明

查询指定存储桶的跨域名访问配置信息（GET Bucket cors）。

方法原型

```
public Guzzle\Service\Resource\Model getBucketCors(array $args = array());
```

请求示例

```
try {
$result = $cosClient->getBucketCors(array(
'Bucket' => 'examplebucket-1250000000' //格式：BucketName-APPID
));
// 请求成功
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo($e);
}
```

参数说明

参数名称	类型	描述	必填
Bucket	String	存储桶名称，格式：BucketName-APPID	是

返回结果示例

```
Guzzle\Service\Resource\Model Object
(
[data:protected] => Array
(
[CORSRules] => Array
(
[0] => Array
(
[ID] => 1234
[AllowedHeaders] => Array
(
[0] => *
)
[AllowedMethods] => Array
(
[0] => PUT
)
[AllowedOrigins] => Array
(
[0] => http://imgcache.finance.cloud.tencent.com:80www.qq.com
)
)
)
[RequestId] => NWE3YzhkMmRfMTdiMjk0MGFfNTQzZl8xNWUwMGU=
)
)
```

返回结果说明

参数名称	类型	描述	父节点
CORSRules	Array	跨域信息列表	无
CORSRule	Array	跨域信息	CORSRules
AllowedMethods	String	允许的 HTTP 操作，枚举值：GET，PUT，HEAD，POST，DELETE	CORSRule
AllowedOrigins	String	允许的访问来源，支持通配符`*`，格式为：协议://域名[:端口]。例如：`http://imgcache.finance.cloud.tencent.com:80www.qq.com`	CORSRule

参数名称	类型	描述	父节点
AllowedHeaders	String	在发送 OPTIONS 请求时告知服务端，接下来的请求可以使用哪些自定义的 HTTP 请求头部，支持通配符`*`	CORSRule
ExposeHeaders	String	设置浏览器可以接收到的来自服务器端的自定义头部信息	CORSRule
MaxAgeSeconds	Int	设置 OPTIONS 请求得到结果的有效期	CORSRule
ID	String	配置规则的 ID	CORSRule

删除跨域配置

功能说明

删除指定存储桶的跨域名访问配置（DELETE Bucket cors）。

方法原型

```
public Guzzle\Service\Resource\Model deleteBucketCors(array $args = array());
```

请求示例

```
try {
$result = $cosClient->deleteBucketCors(array(
'Bucket' => 'examplebucket-1250000000' //格式：BucketName-APPID
));
// 请求成功
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo($e);
}
```

参数说明

参数名称	类型	描述	必填
Bucket	String	存储桶名称，格式：BucketName-APPID	是

版本控制

设置版本控制

功能说明

设置指定存储桶的版本控制功能（PUT Bucket versioning）。

方法原型

```
public Guzzle\Service\Resource\Model putBucketVersioning(array $args = array());
```

请求示例

开启版本控制

```
try {
$result = $cosClient->putBucketVersioning(array(
'Bucket' => 'examplebucket-125000000', //格式：BucketName-APPID
'Status' => 'Enabled'
));
// 请求成功
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo "$e\n";
}
```

暂停版本控制

```
try {
$result = $cosClient->putBucketVersioning(array(
'Bucket' => 'examplebucket-125000000', //格式：BucketName-APPID
'Status' => 'Suspended'
));
// 请求成功
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo "$e\n";
}
```

参数说明

参数名称	类型	描述	必填
Bucket	String	存储桶名称，格式：BucketName-APPID	是
Status	String	版本控制策略，可选值为Suspended/Enabled	是

查询版本控制

功能说明

查询指定存储桶的版本控制信息（GET Bucket versioning）。

方法原型

```
public Guzzle\Service\Resource\Model getBucketVersioning(array $args = array());
```

请求示例

```
try {
$result = $cosClient->getBucketVersioning(array(
'Bucket' => 'examplebucket-125000000', //格式：BucketName-APPID
));
// 请求成功
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo "$e\n";
}
```

参数说明

参数名称	类型	描述	必填
Bucket	String	存储桶名称，格式：BucketName-APPID	是

返回结果说明

参数名称	类型	描述	父节点
Status	String	版本控制策略,可选值为Suspended/Enabled或空	无

跨地域复制

设置跨地域复制

功能说明

设置指定存储桶的跨地域复制规则（PUT Bucket replication）。

方法原型

```
public Guzzle\Service\Resource\Model putBucketReplication(array $args = array());
```

请求示例

```
try {
$result = $cosClient->putBucketReplication(array(
'Bucket' => 'examplebucket-125000000', //格式：BucketName-APPID
'Role' => 'qcs::cam::uin/100000000001:uin/100000000001',
'Rules'=>array(
array(
'Status' => 'Enabled',
'ID' => 'string',
'Prefix' => 'string',
'Destination' => array( 'Bucket' => 'qcs::cos:ap-guangzhou::examplebucket2-125000000',
'StorageClass' => 'standard', ),
// ...repeated ),
), ));
// 请求成功 print_r($result);
} catch (\Exception $e) { // 请求失败
echo "$e\n";
}
```

参数说明

参数名称	类型	描述	必填
Bucket	String	源存储桶名称，格式：BucketName-APPID	是
Role	String	发起者身份标示, 格式为 qcs::cam::uin/:uin/	是
Rules	Array	设置对应的规则，包括 ID，Status，Prefix，Destination	是
ID	String	设置规则的 ID	是
Status	String	设置 Rule 是否启用，可选值为 Enabled 或者 Disabled	是
Prefix	String	设置 Rule 的前缀匹配规则，为空时表示作用存储桶中的所有 objects	是
Destination	String	描述目的资源, 包括 Bucket 和 StorageClass	是
Bucket	String	设置跨地域复制的目标存储桶，格式为 qcs::cos:[region]::[BucketName-APPID]	是
StorageClass	String	设置目的文件的存储类型，默认值：'STANDARD'	否

查询跨地域复制

功能说明

查询指定存储桶的跨地域复制规则（GET Bucket replication）。

方法原型

```
public Guzzle\Service\Resource\Model getBucketReplication(array $args = array());
```

请求示例

```
try {
$result = $cosClient->getBucketReplication(array(
'Bucket' => 'examplebucket-125000000', //格式：BucketName-APPID
));
// 请求成功
print_r($result);
} catch (\Exception $e) {
// 请求失败
echo "$e\n";
}
```

参数说明

参数名称	类型	描述	必填
Bucket	String	存储桶名称，格式：BucketName-APPID	是

返回结果示例

```
Guzzle\Service\Resource\Model Object
(
[data:protected] => Array
(
[Role] => qcs::cam::uin/100000000001:uin/100000000001
[Rules] => Array
(
[0] => Array
(
[ID] => string
[Status] => Enabled
[Prefix] => string
[Destination] => Array
(
[Bucket] => qcs::cos:ap-guangzhou::examplebucket2-125000000
[StorageClass] =>
)
)
)
[RequestId] => NWQwOGI5MGVfNWFiMjU4NjRfNDUzY19mNzRhMTU=
)
)
```

返回结果说明

参数名称	类型	描述	父节点
Role	String	发起者身份标示, 格式为 qcs::cam::uin/:uin/	无
Rules	Array	设置对应的规则，包括 ID，Status，Prefix，Destination	无
Rule	Array	设置对应的规则，包括 ID，Status，Prefix，Destination	Rules
ID	String	设置规则的 ID	Rule
Status	String	设置 Rule 是否启用，可选值为 Enabled 或者 Disabled	Rule
Prefix	String	设置 Rule 的前缀匹配规则，为空时表示作用存储桶中的所有 objects	Rule
Destination	String	描述目的资源, 包括 Bucket 和 StorageClass	Rule
Bucket	String	设置跨地域复制的目标存储桶，格式为 qcs::cos:[region]::[BucketName-APPID]	Destination
StorageClass	String	设置目的文件的存储类型，默认值：'STANDARD'	Destination

删除跨地域复制

功能说明

删除指定存储桶的跨地域复制规则（DELETE Bucket replication）。

方法原型

```
public Guzzle\Service\Resource\Model deleteBucketReplication(array $args = array());
```

请求示例

```
try {
$result = $cosClient->deleteBucketReplication(array(
'Bucket' => 'examplebucket-125000000', //格式：BucketName-APPID
));
// 请求成功
print_r($result);
```

```
} catch (\Exception $e) {
// 请求失败
echo "$e\n";
}
```

参数说明

参数名称	类型	描述	必填
Bucket	String	存储桶名称，格式：BucketName-APPID	是

预签名 URL

简介

PHP SDK 提供获取请求预签名 URL 接口。

永久密钥预签名请求示例

上传请求示例

```
$secretId = "COS_SECRETID"; // 替换为用户的 SecretId
$secretKey = "COS_SECRETKEY"; // 替换为用户的 SecretKey

$region = "REGION"; // 替换为用户的 Region
$domain = "DOMAIN.COM"; // 替换为用户的 Domain

$formatRegion = sprintf("cos.%s", $region); // 格式化为标准的 Region (前方加 cos.)
$endpoint = sprintf("%s.%s", $formatRegion, $domain);
// 通过 FormatRegion 和 Domain 生成 Endpoint

$cosClient = new Qcloud\Cos\Client(
array(
'schema' => 'http', // 协议头部，默认为http
'region' => $region, // Region
'endpoint' => $endpoint, // Endpoint
'credentials'=> array(
'secretId' => $secretId ,
'secretKey' => $secretKey)));

### 简单上传预签名
try {
$command = $cosClient->getCommand('putObject', array(
'Bucket' => "examplebucket-1250000000", //存储桶，格式：BucketName-APPID
'Key' => "exampleobject", //对象在存储桶中的位置，即对象键
'Body' => '', //
));
$signedUrl = $command->createPresignedUrl('+10 minutes');
// 请求成功
echo ($signedUrl);
} catch (\Exception $e) {
// 请求失败
echo($e);
}

### 分块上传预签名
try {
$command = $cosClient->getCommand('uploadPart', array(
'Bucket' => "examplebucket-1250000000", //存储桶，格式：BucketName-APPID
'Key' => "exampleobject", //对象在存储桶中的位置，即对象键
'UploadId' => '',
'PartNumber' => '1',
'Body' => '',
```

```
));
$signedUrl = $command->createPresignedUrl('+10 minutes');
// 请求成功
echo ($signedUrl);
} catch (\Exception $e) {
// 请求失败
echo($e);
}
```

下载请求示例

```
$secretId = "COS_SECRETID"; // 替换为用户的 SecretId
$secretKey = "COS_SECRETKEY"; // 替换为用户的 SecretKey

$region = "REGION"; // 替换为用户的 Region
$domain = "DOMAIN.COM"; // 替换为用户的 Domain

$formatRegion = sprintf("cos.%s", $region); // 格式化为标准的 Region (前方加 cos.)
$endpoint = sprintf("%s.%s", $formatRegion, $domain);
// 通过 FormatRegion 和 Domain 生成 Endpoint

$cosClient = new Qcloud\Cos\Client(
array(
'schema' => 'http', // 协议头部，默认为http
'region' => $region, // Region
'endpoint' => $endpoint, // Endpoint
'credentials'=> array(
'secretId' => $secretId ,
'secretKey' => $secretKey)));

### 简单下载预签名
try {
$command = $cosClient->getCommand('getObject', array(
'Bucket' => "examplebucket-1250000000", //存储桶，格式：BucketName-APPID
'Key' => "exampleobject" //对象在存储桶中的位置，即对象键
));
$signedUrl = $command->createPresignedUrl('+10 minutes');
// 请求成功
echo ($signedUrl);
} catch (\Exception $e) {
// 请求失败
echo($e);
}

### 使用封装的 getObjectUrl 获取下载签名
try {
$bucket = "examplebucket-1250000000"; //存储桶，格式：BucketName-APPID
$key = "exampleobject"; //对象在存储桶中的位置，即对象键
$signedUrl = $cosClient->getObjectUrl($bucket, $key, '+10 minutes');
// 请求成功
echo $signedUrl;
} catch (\Exception $e) {
// 请求失败
print_r($e);
}
```

临时密钥预签名请求示例

上传请求示例

```
$secretId = "COS_SECRETID"; // 临时密钥 SecretId
$secretKey = "COS_SECRETKEY"; // 临时密钥 SecretKey
$tmpToken = "COS_TMPTOKEN" // 临时密钥 Token

$region = "REGION"; // 替换为用户的 Region
$domain = "DOMAIN.COM"; // 替换为用户的 Domain

$formatRegion = sprintf("cos.%s", $region); // 格式化为标准的 Region (前方加 cos.)
```

```
$endpoint = sprintf("%s.%s", $formatRegion, $domain);
// 通过 FormatRegion 和 Domain 生成 Endpoint

$cosClient = new Qcloud\Cos\Client(
array(
'schema' => 'http', // 协议头部，默认为http
'region' => $region, // Region
'endpoint' => $endpoint, // Endpoint
'credentials'=> array(
'secretId' => $secretId ,
'secretKey' => $secretKey,
'token' => $tmpToken)));

### 简单上传预签名
try {
$command = $cosClient->getCommand('putObject', array(
'Bucket' => "examplebucket-1250000000", //存储桶，格式：BucketName-APPID
'Key' => "exampleobject", //对象在存储桶中的位置，即对象键
'Body' => '',
));
$signedUrl = $command->createPresignedUrl('+10 minutes');
// 请求成功
echo ($signedUrl);
} catch (\Exception $e) {
// 请求失败
echo($e);
}

### 分块上传预签名
try {
$command = $cosClient->getCommand('uploadPart', array(
'Bucket' => "examplebucket-1250000000", //存储桶，格式：BucketName-APPID
'Key' => "exampleobject", //对象在存储桶中的位置，即对象键
'UploadId' => '',
'PartNumber' => '1',
'Body' => '',
));
$signedUrl = $command->createPresignedUrl('+10 minutes');
// 请求成功
echo ($signedUrl);
} catch (\Exception $e) {
// 请求失败
echo($e);
}
```

下载请求示例

```
$secretId = "COS_SECRETID"; // 临时密钥 SecretId
$secretKey = "COS_SECRETKEY"; // 临时密钥 SecretKey
$tmpToken = "COS_TMPTOKEN" // 临时密钥 Token

$region = "REGION"; // 替换为用户的 Region
$domain = "DOMAIN.COM"; // 替换为用户的 Domain

$formatRegion = sprintf("cos.%s", $region); // 格式化为标准的 Region (前方加 cos.)
$endpoint = sprintf("%s.%s", $formatRegion, $domain);
// 通过 FormatRegion 和 Domain 生成 Endpoint

$cosClient = new Qcloud\Cos\Client(
array(
'schema' => 'http', // 协议头部，默认为http
'region' => $region, // Region
'endpoint' => $endpoint, // Endpoint
'credentials'=> array(
'secretId' => $secretId ,
'secretKey' => $secretKey,
'token' => $tmpToken)));

### 简单下载预签名
try {
$command = $cosClient->getCommand('getObject', array(
```

```
'Bucket' => "examplebucket-1250000000", //存储桶，格式：BucketName-APPID
'Key' => "exampleobject" //对象在存储桶中的位置，即对象键
));
$signedUrl = $command->createPresignedUrl('+10 minutes');
// 请求成功
echo ($signedUrl);
} catch (\Exception $e) {
// 请求失败
echo($e);
}

### 使用封装的 getObjectUrl 获取下载签名
try {
$bucket = "examplebucket-1250000000"; //存储桶，格式：BucketName-APPID
$key = "exampleobject"; //对象在存储桶中的位置，即对象键
$signedUrl = $cosClient->getObjectUrl($bucket, $key, '+10 minutes');
// 请求成功
echo $signedUrl;
} catch (\Exception $e) {
// 请求失败
print_r($e);
}
```

异常处理

简介

调用 SDK 接口请求 COS 服务失败时，如返回码为4xx或者5xx，系统将抛出（Qcloud\Cos\Exception\ServiceResponseException）异常。

服务端异常

CosServerException 包含了服务端返回的状态码、requestid 和出错明细等。捕获异常后，建议对整个异常进行打印，异常包含了必须的排查因素。以下是异常成员变量的描述以及异常捕获示例：

成员	描述	类型
requestId	请求 ID，用于表示一个请求，对于排查问题十分重要	string
statusCode	response 的 status 状态码	string
errorCode	请求失败时 body 返回的 Error Code	string
errorMessage	请求失败时 body 返回的 Error Message	string

异常捕获示例

```
try {
$cosClient->listBuckets()
} catch (\Exception $e) {
$statusCode = $e->getStatusCode(); // 获取错误码
$errorMessage = $e->getMessage(); // 获取错误信息
$requestId = $e->getRequestId(); // 获取错误的 requestId
$errorCode = $e->getCosErrorCode(); // 获取错误名称
$request = $e->getRequest(); // 获取完整的请求
$response = $e->getResponse(); // 获取完整的响应
 }
```

Python SDK
快速入门

最近更新时间: 2025-01-15 17:01:00

下载与安装

相关资源

对象存储的 XML Python SDK 资源下载地址： XML Python SDK 。 演示示例 Demo 下载地址：XML Python Demo。

环境依赖

对象存储的 XML Python SDK 目前可以支持 Python 2.6、Python 2.7 以及 Python 3.x。

安装 SDK

安装 SDK 有三种安装方式：pip 安装、手动安装和离线安装。

- 使用 pip 安装（推荐）

```
pip install -U cos-python-sdk-v5
```

- 手动安装 从 XML Python SDK 下载源码，通过 setup 手动安装，执行以下命令。

```
python setup.py install
```

- 离线安装

```
# 在有外网的机器下运行如下命令
mkdir cos-python-sdk-packages
pip download cos-python-sdk-v5 -d cos-python-sdk-packages
tar -czvf cos-python-sdk-packages.tar.gz cos-python-sdk-packages
# 将安装包拷贝到没有外网的机器后运行如下命令
# 请确保两台机器的 python 版本保持一致，否则会出现安装失败的情况
tar -xzvf cos-python-sdk-packages.tar.gz
pip install cos-python-sdk-v5 --no-index -f cos-python-sdk-packages
```

术语解释

名称	描述
APPID	开发者访问 COS 服务时拥有的用户维度唯一资源标识，用以标识资源
SecretId	开发者拥有的项目身份识别 ID，用以身份认证
SecretKey	开发者拥有的项目身份密钥
Bucket	COS 中用于存储数据的容器
Object	COS 中存储的具体文件，是存储的基本实体
Region	域名中的地域信息
Endpoint	Endpoint 由 Region 和域名组成，具体格式为： "."，其中 Domain 为自定义的域名。 在控制台创建 Bucket 时，可以看到对应的访问地址为："."，Bucket 后面的部分即为 Endpoint。
ACL	访问控制列表（Access Control List），是指特定 Bucket 或 Object 的访问控制信息列表
CORS	跨域资源共享（Cross-Origin Resource Sharing），指发起请求的资源所在域不同于该请求所指向资源所在的域的 HTTP 请求
Multipart Uploads	分块上传，COS 服务为上传文件提供的一种分块上传模式

开始使用

下面为您介绍如何使用 COS Python SDK 完成一个基础操作，如初始化客户端、创建存储桶、查询存储桶列表、上传对象、查询对象列表、下载对象和删除对象。

初始化

请参考以下示例代码：

```
# Bucket 由 BucketName-APPID 组成
# 1. 设置用户配置, 包括 SecretId，SecretKey 以及 Region
from qcloud_cos import CosConfig
from qcloud_cos import CosS3Client
from qcloud_cos.cos_comm import format_region

secret_id = 'COS_SECRETID' # 替换为用户的 SecretId
secret_key = 'COS_SECRETKEY' # 替换为用户的 SecretKey

region = "REGION" # 替换为用户的 Region
domain = "DOMAIN.COM" # 替换为用户的 Domain
endpoint = "{}.{}".format(format_region(region), domain)
# 通过 Region 和 Domain 生成 Endpoint, 注意使用 format_region

token = None # 使用临时密钥需要传入 Token，默认为空，可不填
scheme = 'http' # 指定使用 http/https 协议来访问 COS，默认为 https
config = CosConfig(Scheme=scheme, Secret_id=secret_id, Secret_key=secret_key, Endpoint=endpoint, Token=token)

# 2. 获取客户端对象
client = CosS3Client(config)

# 参照下文的描述。或者参照 Demo 程序，详见 http://imgcache.finance.cloud.tencent.com:80github.com/tencentyun/cos-python-sdk-v5/blob/master/qclou
d_cos/demo.py
```

创建存储桶

```
response = client.create_bucket(
Bucket='examplebucket-1250000000'
)
```

上传对象

```
#### 文件流简单上传
# 强烈建议您以二进制模式(binary mode)打开文件,否则可能会导致错误
with open('picture.jpg', 'rb') as fp:
response = client.put_object(
Bucket='examplebucket-1250000000',
Body=fp,
Key='picture.jpg',
StorageClass='STANDARD',
EnableMD5=False
)
print(response['ETag'])

#### 字节流简单上传
response = client.put_object(
Bucket='examplebucket-1250000000',
Body=b'bytes',
Key='picture.jpg',
EnableMD5=False
)
print(response['ETag'])


#### chunk 简单上传
import requests
stream = requests.get('/64296333127741440/64296342553391104')

# 网络流将以 Transfer-Encoding:chunked 的方式传输到 COS
response = client.put_object(
Bucket='examplebucket-1250000000',
Body=stream,
Key='picture.jpg'
```

```
)
print(response['ETag'])

#### 高级上传接口（推荐）
根据文件大小自动选择简单上传或分块上传，分块上传具备断点续传功能。
response = client.upload_file(
Bucket='examplebucket-1250000000',
LocalFilePath='local.txt',
Key='picture.jpg',
PartSize=1,
MAXThread=10,
EnableMD5=False
)
print(response['ETag'])
```

查询对象列表

```
response = client.list_objects(
Bucket='examplebucket-1250000000',
Prefix='folder1'
)
```

单次调用 `list_objects` 接口一次只能查询1000个对象，如需要查询所有的对象，则需要循环调用。

```
marker = ""
while True:
response = client.list_objects(
Bucket='examplebucket-1250000000',
Prefix='folder1',
Marker=marker
)
print(response['Contents'])
if response['IsTruncated'] == 'false':
break
marker = response['NextMarker']
```

下载对象

```
#### 获取文件到本地
response = client.get_object(
Bucket='examplebucket-1250000000',
Key='picture.jpg',
)
response['Body'].get_stream_to_file('output.txt')

#### 获取文件流
response = client.get_object(
Bucket='examplebucket-1250000000',
Key='picture.jpg',
)
fp = response['Body'].get_raw_stream()
print(fp.read(2))

#### 设置 Response HTTP 头部
response = client.get_object(
Bucket='examplebucket-1250000000',
Key='picture.jpg',
ResponseContentType='text/html; charset=utf-8'
)
print response['Content-Type']
fp = response['Body'].get_raw_stream()
print(fp.read(2))

#### 指定下载范围
response = client.get_object(
Bucket='examplebucket-1250000000',
Key='picture.jpg',
Range='bytes=0-10'
)
```

```
fp = response['Body'].get_raw_stream()
print(fp.read())
```

删除对象

```
# 删除object
## deleteObject
response = client.delete_object(
Bucket='examplebucket-1250000000',
Key='exampleobject'
)

# 删除多个object
## deleteObjects
response = client.delete_objects(
Bucket='examplebucket-1250000000',
Delete={
'Object': [
{
'Key': 'exampleobject1',
},
{
'Key': 'exampleobject2',
},
],
'Quiet': 'true'|'false'
}
)
```

接口文档

最近更新时间: 2025-01-15 17:01:00

存储桶操作

简介

本文档提供关于存储桶的基本操作和访问控制列表（ACL）的相关 API 概览以及 SDK 示例代码。

基本操作

API	操作名	操作描述
PUT Bucket	创建存储桶	在指定账号下创建一个存储桶
HEAD Bucket	检索存储桶及其权限	检索存储桶是否存在且是否有权限访问
DELETE Bucket	删除存储桶	删除指定账号下的空存储桶

访问控制列表

API	操作名	操作描述
PUT Bucket acl	设置存储桶 ACL	设置指定存储桶访问权限控制列表
GET Bucket acl	查询存储桶 ACL	查询存储桶的访问控制列表

基本操作

创建存储桶

功能说明

在指定账号下创建一个存储桶（PUT Bucket）。

方法原型

```
create_bucket(Bucket, **kwargs)
```

请求示例

```
response = client.create_bucket(
Bucket='examplebucket-1250000000',
ACL='private'|'public-read'|'public-read-write',
GrantFullControl='string',
GrantRead='string',
GrantWrite='string'
)
```

参数说明

参数名称	参数描述	类型	必填
Bucket	待创建的存储桶名称，由 BucketName-APPID 构成	String	是
ACL	设置存储桶的 ACL，例如 'private'，'public-read'，'public-read-write'	String	否
GrantFullControl	赋予指定账户对存储桶的读写权限，格式为`id="OwnerUin"`	String	否
GrantRead	赋予指定账户对存储桶的读权限，格式为`id="OwnerUin"`	String	否

参数名称	参数描述	类型	必填
GrantWrite	赋予指定账户对存储桶的写权限，格式为`id="OwnerUin"`	String	否

返回结果说明

该方法返回值为 None。

检索存储桶及其权限

功能说明

检索存储桶是否存在且是否有权限访问（HEAD Bucket）。

方法原型

```
head_bucket(Bucket)
```

请求示例

```
response = client.head_bucket(
Bucket='examplebucket-1250000000'
)
```

参数说明

参数名称	参数描述	类型	必填
Bucket	待查询的存储桶名称，由 BucketName-APPID 构成	String	是

返回结果说明

该方法返回值为 None。

删除存储桶

功能说明

删除指定账号下的空存储桶（DELETE Bucket）。

方法原型

```
delete_bucket(Bucket)
```

请求示例

```
response = client.delete_bucket(
Bucket='examplebucket-1250000000'
)
```

参数说明

参数名称	参数描述	类型	必填
Bucket	待删除的存储桶名称，由 BucketName-APPID 构成	String	是

返回结果说明

该方法返回值为 None。

访问控制列表

设置存储桶 ACL

功能说明

设置指定存储桶的访问权限控制列表（PUT Bucket acl）。AccessControlPolicy 参数与其它权限参数是互斥的，无法同时指定。

方法原型

```
put_bucket_acl(Bucket, AccessControlPolicy={}, **kwargs)
```

请求示例

```
response = client.put_bucket_acl(
Bucket='examplebucket-1250000000',
ACL='private'|'public-read'|'public-read-write',
GrantFullControl='string',
GrantRead='string',
GrantWrite='string',
AccessControlPolicy={
'AccessControlList': {
'Grant': [
{
'Grantee': {
'DisplayName': 'string',
'Type': 'CanonicalUser'|'Group',
'ID': 'string',
'URI': 'string'
},
'Permission': 'FULL_CONTROL'|'WRITE'|'READ'
},
]
},
'Owner': {
'DisplayName': 'string',
'ID': 'string'
}
}
)
```

参数说明

参数名称	参数描述	类型	必填
Bucket	存储桶名称，由 BucketName-APPID 构成	String	是
ACL	设置存储桶的 ACL，例如 'private'，'public-read'，'public-read-write'	String	否
GrantFullControl	赋予指定账户对存储桶的读写权限，格式为`id="OwnerUin"`	String	否
GrantRead	赋予指定账户对存储桶的读权限，格式为`id="OwnerUin"`	String	否
GrantWrite	赋予指定账户对存储桶的写权限，格式为`id="OwnerUin"`	String	否
AccessControlPolicy	赋予指定账户对存储桶的访问权限，具体格式见 GET Bucket acl 返回结果说明	Dict	否

返回结果说明

该方法返回值为 None。

查询存储桶 ACL

功能说明

查询指定存储桶的访问权限控制列表（GET Bucket acl）。

方法原型

```
get_bucket_acl(Bucket, **kwargs)
```

请求示例

```
response = client.get_bucket_acl(
Bucket='examplebucket-1250000000',
)
```

参数说明

参数名称	参数描述	类型	必填
Bucket	Bucket 名称，由 BucketName-APPID 构成	String	是

返回结果说明

Bucket ACL 信息，类型为 dict。

```
{
'Owner': {
'DisplayName': 'string',
'ID': 'string'
},
'Grant': [
{
'Grantee': {
'DisplayName': 'string',
'Type': 'CanonicalUser'|'Group',
'ID': 'string',
'URI': 'string'
},
'Permission': 'FULL_CONTROL'|'WRITE'|'READ'
},
]
}
```

参数名称	参数描述	类型
Owner	存储桶拥有者的信息，包括 DisplayName 和 ID	Dict
Grant	存储桶权限授予者的信息，包括 Grantee 和 Permission	List
Grantee	权限授予者的信息，包括 DisplayName，Type，ID 和 URI	Dict
DisplayName	权限授予者的名字	String
Type	权限授予者的类型，类型为 CanonicalUser 或者 Group	String
ID	Type 为 CanonicalUser 时，对应权限授予者的 ID	String
URI	Type 为 Group 时，对应权限授予者的 URI	String
Permission	授予者所拥有的存储桶的权限，可选值有 FULL_CONTROL，WRITE，READ，分别对应读写权限、写权限、读权限	String

对象操作

简介

本文档提供关于对象的简单操作、分块操作等其他操作相关的 API 概览以及 SDK 示例代码。

简单操作

API	操作名	操作描述
GET Bucket（List Object）	查询对象列表	查询存储桶下的部分或者全部对象
GET Bucket Object Versions	查询对象及其历史版本列表	查询存储桶下的部分或者全部对象及其历史版本信息
PUT Object	简单上传对象	上传一个对象至存储桶
HEAD Object	查询对象元数据	查询对象的元数据信息

API	操作名	操作描述
GET Object	下载对象	下载一个对象至本地
PUT Object - Copy	设置对象复制	复制对象到目标路径
DELETE Object	删除单个对象	在存储桶中删除指定对象
DELETE Multiple Objects	删除多个对象	在存储桶中批量删除指定对象

分块操作

API	操作名	操作描述
List Multipart Uploads	查询分块上传	查询正在进行中的分块上传信息
Initiate Multipart Upload	初始化分块上传	初始化分块上传任务
Upload Part	上传分块	分块上传文件
Upload Part - Copy	复制分块	将其他对象复制为一个分块
List Parts	查询已上传块	查询特定分块上传操作中的已上传的块
Complete Multipart Upload	完成分块上传	完成整个文件的分块上传
Abort Multipart Upload	终止分块上传	终止一个分块上传操作并删除已上传的块

其他操作

API	操作名	操作描述
PUT Object acl	设置对象 ACL	设置存储桶中某个对象的访问控制列表
GET Object acl	查询对象 ACL	查询存储桶中某个对象的访问控制列表

简单操作

查询对象列表

功能说明

查询存储桶下的部分或者全部对象。

方法原型

```
list_objects(Bucket, Delimiter="", Marker="", MaxKeys=1000, Prefix="", EncodingType="", **kwargs)
```

请求示例

```
response = client.list_objects(
Bucket='examplebucket-1250000000',
Prefix='string'
)
```

全部参数请求示例

```
response = client.list_objects(
Bucket='examplebucket-1250000000',
Prefix='string',
Delimiter='/',
Marker='string',
MaxKeys=100,
EncodingType='url'
)
```

参数说明

参数名称	参数描述	类型	必填
Bucket	存储桶名称，由 BucketName-APPID 构成	String	是
Prefix	默认为空，对对象的对象键进行筛选，匹配 prefix 为前缀的对象	String	否
Delimiter	默认为空，设置分隔符，例如设置`/`来模拟文件夹	String	否
Marker	默认以 UTF-8 二进制顺序列出条目，标记返回对象的 list 的起点位置	String	否
MaxKeys	最多返回的对象数量，默认为最大的1000	Int	否
EncodingType	默认不编码，规定返回值的编码方式，可选值：url	String	否

返回结果说明

获取对象的元信息，类型为 dict：

```
{
'MaxKeys': '1000',
'Prefix': 'string',
'Delimiter': 'string',
'Marker': 'string',
'NextMarker': 'string',
'Name': 'examplebucket-1250000000',
'IsTruncated': 'false'|'true',
'EncodingType': 'url',
'Contents':[
{
'ETag': '"a5b2e1cfb08d10f6523f7e6fbf3643d5"',
'StorageClass': 'STANDARD',
'Key': 'exampleobject',
'Owner': {
'DisplayName': '1250000000',
'ID': '1250000000'
},
'LastModified': '2017-08-08T09:43:35.000Z',
'Size': '23'
},
],
'CommonPrefixes':[
{
'Prefix': 'string'
},
],
}
```

参数名称	参数描述	类型
MaxKeys	最多返回的对象数量，默认为最大的1000	String
Prefix	默认为空，匹配 Prefix 为前缀的对象	String
Delimiter	默认为空，设置分隔符，例如设置`/`来模拟文件夹	String
Marker	默认以 UTF-8 二进制顺序列出条目，标记返回对象的 list 的起点位置	String
NextMarker	当 IsTruncated 为 true 时，标记下一次返回对象的 list 的起点位置	String
Name	存储桶名称，由 BucketName-APPID 构成	String
IsTruncated	表示返回的对象否被截断	String
EncodingType	默认不编码，规定返回值的编码方式，可选值：url	String
Contents	包含所有对象元数据的 list，包括 'ETag'，'StorageClass'，'Key'，'Owner'，'LastModified'，'Size' 等信息	List
CommonPrefixes	所有以 Prefix 开头，以 Delimiter 结尾的对象被归到同一类	List

查询对象及其历史版本列表

功能说明

查询存储桶下的部分或者全部对象及其历史版本信息。

方法原型

```
list_objects_versions(Bucket, Prefix="", Delimiter="", KeyMarker="", VersionIdMarker="", MaxKeys=1000, EncodingType="", **kwargs)
```

请求示例

```
response = client.list_objects_versions(
Bucket='examplebucket-1250000000',
Prefix='string'
)
```

全部参数请求示例

```
response = client.list_objects_versions(
Bucket='examplebucket-1250000000',
Prefix='string',
Delimiter='/',
KeyMarker='string',
VersionIdMarker='string',
MaxKeys=100,
EncodingType='url'
)
```

参数说明

参数名称	参数描述	类型	必填
Bucket	存储桶名称，由 BucketName-APPID 构成	String	是
Prefix	默认为空，对对象的对象键进行筛选，匹配 prefix 为前缀的对象	String	否
Delimiter	默认为空，设置分隔符，例如设置`/`来模拟文件夹	String	否
KeyMarker	默认以 UTF-8 二进制顺序列出条目，标记返回对象的 list 的 Key 的起点位置	String	否
VersionIdMarker	默认以 UTF-8 二进制顺序列出条目，标记返回对象的 list 的 VersionId 的起点位置	String	否
MaxKeys	最多返回的对象数量，默认为最大的1000	Int	否
EncodingType	默认不编码，规定返回值的编码方式，可选值：url	String	否

返回结果说明

获取对象的元信息，类型为 dict：

```
{
'MaxKeys': '1000',
'Prefix': 'string',
'Delimiter': 'string',
'KeyMarker': 'string',
'VersionIdMarker': 'string',
'NextMarker': 'string',
'NextVersionIdMarker': 'string',
'Name': 'examplebucket-1250000000',
'IsTruncated': 'false'|'true',
'EncodingType': 'url',
'Version':[
{
'ETag': '"a5b2e1cfb08d10f6523f7e6fbf3643d5"',
'StorageClass': 'STANDARD',
'Key': 'exampleobject',
'VersionId': 'string',
'IsLatest': 'true'|'false',
```

```
'Owner': {
'DisplayName': '1250000000',
'ID': '1250000000'
},
'LastModified': '2017-08-08T09:43:35.000Z',
'Size': '23'
},
],
'DeleteMarker': [
{
'Key': 'exampleobject',
'VersionId': 'string',
'IsLatest': 'true'|'false',
'Owner': {
'DisplayName': '1250000000',
'ID': '1250000000'
},
'LastModified': '2017-08-08T09:43:35.000Z'
},
],
'CommonPrefixes':[
{
'Prefix': 'string'
},
],
}
```

参数名称	参数描述	类型
MaxKeys	最多返回的对象数量，默认为最大的1000	String
Prefix	默认为空，匹配 Prefix 为前缀的对象	String
Delimiter	默认为空，设置分隔符，例如设置`/`来模拟文件夹	String
KeyMarker	默认以 UTF-8 二进制顺序列出条目，标记返回对象的 list 的 Key 的起点位置	String
VersionIdMarker	默认以 UTF-8 二进制顺序列出条目，标记返回对象的 list 的 VersionId 的起点位置	String
NextMarker	当 IsTruncated 为 true 时，标记下一次返回对象的 list 的 Key 的起点位置	String
NextVersionIdMarker	当 IsTruncated 为 true 时，标记下一次返回对象的 list 的 VersionId 的起点位置	String
Name	存储桶名称，由 BucketName-APPID 构成	String
IsTruncated	表示返回的对象否被截断	String
EncodingType	默认不编码，规定返回值的编码方式，可选值：url	String
Version	包含所有多个版本对象元数据的 list，包括 'ETag'，'StorageClass'，'Key'，'VersionId'，'IsLatest'，'Owner'，'LastModified'，'Size' 等信息	List
DeleteMarker	包含所有delete marker 对象元数据的 list，包括 'Key'，'VersionId'，'IsLatest'，'Owner'，'LastModified' 等信息	List
CommonPrefixes	所有以 Prefix 开头，以 Delimiter 结尾的对象被归到同一类	List

简单上传对象

功能说明

上传一个对象至存储桶（PUT Object）。

方法原型

```
put_object(Bucket, Body, Key, **kwargs)
```

请求示例

```
response = client.put_object(
Bucket='examplebucket-1250000000',
Body=b'bytes'|file,
```

```
Key='exampleobject',
EnableMD5=False
)
```

全部参数请求示例

```
response = client.put_object(
Bucket='examplebucket-1250000000',
Body=b'bytes'|file,
Key='exampleobject',
EnableMD5=False|True,
ACL='private'|'public-read', # 请慎用此参数,否则会达到1000条 ACL 上限
GrantFullControl='string',
GrantRead='string',
StorageClass='STANDARD',
Expires='string',
CacheControl='string',
ContentType='string',
ContentDisposition='string',
ContentEncoding='string',
ContentLanguage='string',
ContentLength='123',
ContentMD5='string',
Metadata={
'x-cos-meta-key1': 'value1',
'x-cos-meta-key2': 'value2'
}
)
```

参数说明

参数名称	参数描述	类型	必填
Bucket	存储桶名称，由 BucketName-APPID 构成	String	是
Body	上传对象的内容，可以为文件流或字节流	file/bytes	是
Key	对象键（Key）是对象在存储桶中的唯一标识。例如，在对象的访问域名`examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/pic.jpg`中，对象键为 doc/pic.jpg	String	是
EnableMD5	是否需要 SDK 计算 Content-MD5，默认关闭，打开后将增加上传耗时	Bool	否
ACL	设置对象的 ACL，例如 'private'，'public-read'	String	否
GrantFullControl	赋予被授权者所有的权限，格式为`id="OwnerUin"`	String	否
GrantRead	赋予被授权者读的权限，格式为`id="OwnerUin"`	String	否
StorageClass	设置对象的存储类型，默认值 STANDARD	String	否
Expires	设置 Expires	String	否
CacheControl	缓存策略，设置 Cache-Control	String	否
ContentType	内容类型，设置 Content-Type	String	否
ContentDisposition	对象名称，设置 Content-Disposition	String	否
ContentEncoding	编码格式，设置 Content-Encoding	String	否
ContentLanguage	语言类型，设置 Content-Language	String	否
ContentLength	设置传输长度	String	否
ContentMD5	设置上传对象的 MD5 值用于校验	String	否
Metadata	用户自定义的对象元数据， 必须以 x-cos-meta 开头，否则会被忽略	Dict	否

返回结果说明

上传对象的属性，类型为 dict：

```
{
'ETag': 'string',
'x-cos-version-id': 'string'
}
```

参数名称	参数描述	类型
ETag	上传对象的 MD5 值	String
x-cos-version-id	开启版本控制后，对象的版本号	String

查询对象元数据

功能说明

查询对象的元数据信息（HEAD Object）。

方法原型

```
head_object(Bucket, Key, **kwargs)
```

请求示例

```
response = client.head_object(
Bucket='examplebucket-1250000000',
Key='exampleobject',
VersionId='string',
IfModifiedSince='Wed, 28 Oct 2014 20:30:00 GMT',
)
```

参数说明

参数名称	参数描述	类型	必填
Bucket	Bucket 名称，由 BucketName-APPID 构成	String	是
Key	对象键（Key）是对象在存储桶中的唯一标识。例如，在对象的访问域名`examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/pic.jpg`中，对象键为 doc/pic.jpg	String	是
VersionId	开启版本控制后，指定对象的具体版本	String	否
IfModifiedSince	在指定时间后被修改才返回，时间格式为 GMT	String	否

返回结果说明

获取对象的元信息，类型为 dict：

```
{
'ETag': '"9a4802d5c99dafe1c04da0a8e7e166bf"',
'Last-Modified': 'Wed, 28 Oct 2014 20:30:00 GMT',
'Cache-Control': 'max-age=1000000',
'Content-Type': 'application/octet-stream',
'Content-Disposition': 'attachment; filename="filename.jpg"',
'Content-Encoding': 'gzip',
'Content-Language': 'zh-cn',
'Content-Length': '16807',
'Expires': 'Wed, 28 Oct 2019 20:30:00 GMT',
'x-cos-meta-test': 'test',
'x-cos-version-id': 'MTg0NDUxODMzMTMwMDM2Njc1ODA',
'x-cos-request-id': 'NTg3NzQ3ZmVfYmRjMzVfMzE5N182NzczMQ=='
}
```

参数名称	参数描述	类型
ETag	对象的 MD5 值	String
Last-Modified	对象最后修改时间	String

参数名称	参数描述	类型
Cache-Control	缓存策略， HTTP 标准头部	String
Content-Type	内容类型，HTTP 标准头部	String
Content-Disposition	文件名称，HTTP 标准头部	String
Content-Encoding	编码格式，HTTP 标准头部	String
Content-Language	语言类型，HTTP 标准头部	String
Content-Length	对象大小	String
Expires	缓存过期时间， HTTP 标准头部	String
x-cos-meta-*	用户自定义的对象元数据， 必须以 x-cos-meta 开头，否则会被忽略	String
x-cos-version-id	开启版本控制后，对象的版本号	String

下载对象

功能说明

下载一个对象到本地（GET Object）。

方法原型

```
get_object(Bucket, Key, **kwargs)
```

请求示例

```
response = client.get_object(
Bucket='examplebucket-1250000000',
Key='exampleobject',
Range='string',
IfMatch='"9a4802d5c99dafe1c04da0a8e7e166bf"',
IfModifiedSince='Wed, 28 Oct 2014 20:30:00 GMT',
IfNoneMatch='"9a4802d5c99dafe1c04da0a8e7e166bf"',
IfUnmodifiedSince='Wed, 28 Oct 2014 20:30:00 GMT',
ResponseCacheControl='string',
ResponseContentDisposition='string',
ResponseContentEncoding='string',
ResponseContentLanguage='string',
ResponseContentType='string',
ResponseExpires='string',
VersionId='string'
)
```

参数说明

参数名称	参数描述	类型	必填
Bucket	存储桶名称，由 BucketName-APPID 构成	String	是
Key	对象键（Key）是对象在存储桶中的唯一标识。例如，在对象的访问域名`examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/pic.jpg`中，对象键为 doc/pic.jpg	String	是
Range	设置下载对象的范围，格式为 bytes=first-last	String	否
IfMatch	ETag 与指定的内容一致时才返回	String	否
IfModifiedSince	在指定时间后被修改才返回，时间格式为 GMT	String	否
IfNoneMatch	ETag 与指定的内容不一致才返回	String	否
IfUnmodifiedSince	对象修改时间早于或等于指定时间才返回，时间格式为 GMT	String	否
ResponseCacheControl	设置响应头部 Cache-Control	String	否

版权所有：腾讯云金融专区

参数名称	参数描述	类型	必填
ResponseContentDisposition	设置响应头部 Content-Disposition	String	否
ResponseContentEncoding	设置响应头部 Content-Encoding	String	否
ResponseContentLanguage	设置响应头部 Content-Language	String	否
ResponseContentType	设置响应头部 Content-Type	String	否
ResponseExpires	设置响应头部 Expires	String	否
VersionId	指定下载对象的版本	String	否

返回结果说明

下载对象的 Body 和元信息，类型为 dict：

```
{
'Body': StreamBody(),
'ETag': '"9a4802d5c99dafe1c04da0a8e7e166bf"',
'Last-Modified': 'Wed, 28 Oct 2014 20:30:00 GMT',
'Accept-Ranges': 'bytes',
'Content-Range': 'bytes 0-16086/16087',
'Cache-Control': 'max-age=1000000',
'Content-Type': 'application/octet-stream',
'Content-Disposition': 'attachment; filename="filename.jpg"',
'Content-Encoding': 'gzip',
'Content-Language': 'zh-cn',
'Content-Length': '16807',
'Expires': 'Wed, 28 Oct 2019 20:30:00 GMT',
'x-cos-meta-test': 'test',
'x-cos-version-id': 'MTg0NDUxODMzMTMwMDM2Njc1ODA',
'x-cos-request-id': 'NTg3NzQ3ZmVfYmRjMzVfMzE5N182NzczMQ=='
}
```

参数名称	参数描述	类型
Body	下载对象的内容，get_raw_stream() 方法可以得到一个文件流，get_stream_to_file(local_file_path) 方法可以将对象内容下载到指定本地文件中	StreamBody
ETag	对象的 MD5 值	String
Last-Modified	对象最后修改时间	String
Accept-Ranges	范围单位， HTTP 标准头部	String
Content-Range	内容范围， HTTP 标准头部	String
Cache-Control	缓存策略， HTTP 标准头部	String
Content-Type	内容类型，HTTP 标准头部	String
Content-Disposition	文件名称，HTTP 标准头部	String
Content-Encoding	编码格式，HTTP 标准头部	String
Content-Language	语言类型，HTTP 标准头部	String
Content-Length	对象大小	String
Expires	缓存过期时间， HTTP 标准头部	String
x-cos-meta-*	用户自定义的对象元数据， 必须以 x-cos-meta 开头，否则会被忽略	String
x-cos-version-id	开启版本控制后，对象的版本号	String

设置对象复制

功能说明

复制文件到目标路径（PUT Object - Copy）。

方法原型

```
copy_object(Bucket, Key, CopySource, CopyStatus='Copy', **kwargs)
```

请求示例

```
response = client.copy_object(
Bucket='examplebucket-1250000000',
Key='exampleobject',
CopySource={
'Bucket': 'examplebucket-1250000000',
'Key': 'exampleobject',
'Endpoint': 'example.endpoint',
'VersionId': 'string'
},
CopyStatus='Copy'|'Replaced',
ACL='private'|'public-read',
GrantFullControl='string',
GrantRead='string',
StorageClass='STANDARD',
Expires='string',
CacheControl='string',
ContentType='string',
ContentDisposition='string',
ContentEncoding='string',
ContentLanguage='string',
Metadata={
'x-cos-meta-key1': 'value1',
'x-cos-meta-key2': 'value2'
}
)
```

参数说明

参数名称	参数描述	类型	必填
Bucket	存储桶名称，由 BucketName-APPID 构成	String	是
Key	对象键（Key）是对象在存储桶中的唯一标识。例如，在对象的访问域名`examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/pic.jpg`中，对象键为 doc/pic.jpg	String	是
CopySource	描述拷贝源对象的路径，包含 Bucket、Key、Endpoint、VersionId	Dict	是
CopyStatus	可选值为 'Copy'，'Replaced'，设置为 'Copy' 时，忽略设置的用户元数据信息直接复制，设置为 'Replaced' 时，按设置的元信息修改元数据，当目标路径和源路径一样时，必须设置为 'Replaced'	String	是
ACL	设置对象的ACL，如 'private'，'public-read'	String	否
GrantFullControl	赋予指定账户对对象的所有权限，格式为`id="OwnerUin"`	String	否
GrantRead	赋予指定账户对对象的读权限，格式为`id="OwnerUin"`	String	否
StorageClass	设置对象的存储类型，默认值 STANDARD	String	否
Expires	设置 Expires	String	否
CacheControl	缓存策略，设置 Cache-Control	String	否
ContentType	内容类型，设置 Content-Type	String	否
ContentDisposition	文件名称，设置 Content-Disposition	String	否
ContentEncoding	编码格式，设置 Content-Encoding	String	否
ContentLanguage	语言类型，设置 Content-Language	String	否
Metadata	用户自定义的对象元数据	Dict	否

返回结果说明

上传对象的属性，类型为 dict：

```
{
'ETag': 'string',
'LastModified': 'string',
'VersionId': 'string',
'x-cos-copy-source-version-id': 'string'
}
```

参数名称	参数描述	类型
ETag	拷贝对象的 MD5 值	String
LastModified	拷贝对象的最后一次修改时间	String
VersionId	开启版本控制后，目的对象的版本号	String
x-cos-copy-source-version-id	源对象的版本号	String

删除单个对象

功能说明

删除指定的对象（DELETE Object）。

方法原型

```
delete_object(Bucket, Key, **kwargs)
```

请求示例

```
response = client.delete_object(
Bucket='examplebucket-1250000000',
Key='exampleobject',
VersionId='string',
)
```

参数说明

参数名称	参数描述	类型	必填
Bucket	存储桶名称，由 BucketName-APPID 构成	String	是
Key	对象键（Key）是对象在存储桶中的唯一标识。例如，在对象的访问域名`examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/pic.jpg`中，对象键为 doc/pic.jpg	String	是
VersionId	开启版本控制后，指定对象的具体版本	String	否

返回结果说明

删除对象的信息，类型为dict。

```
{
'x-cos-version-id': 'string',
'x-cos-delete-marker': 'true'|'false',
}
```

参数名称	参数描述	类型
x-cos-version-id	删除对象的版本号	String
x-cos-delete-marker	标识删除的对象是否为delete marker	String

删除多个对象

功能说明

删除多个指定的对象（DELETE Multiple Objects）。

方法原型

```
delete_objects(Bucket, Delete={}, **kwargs)
```

请求示例

```
response = client.delete_objects(
Bucket='examplebucket-1250000000',
Delete={
'Object': [
{
'Key': 'exampleobject1',
'VersionId': 'string'
},
{
'Key': 'exampleobject2',
'VersionId': 'string'
},
],
'Quiet': 'true'|'false'
}
)
```

参数说明

参数名称	参数描述	类型	必填
Bucket	Bucket 名称，由 BucketName-APPID 构成	String	是
Delete	说明本次删除的返回结果方式和目标 Object	Dict	是
Object	说明每个将要删除的目标 Object 信息	List	是
Key	对象键（Key）是对象在存储桶中的唯一标识。例如，在对象的访问域名`examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/pic.jpg`中，对象键为 doc/pic.jpg	String	否
VersionId	开启版本控制后，目的对象的版本号	String	
Quiet	指明删除的返回结果方式，可选值为 'true'，'false'，默认为 'false'。设置为 'true' 只返回失败的错误信息，设置为 'false' 时返回成功和失败的所有信息	String	否

返回结果说明

批量删除对象的结果，类型为 dict：

```
{
'Deleted': [
{
'Key': 'string',
'VersionId': 'string',
'DeleteMarker': 'true'|'false',
'DeleteMarkerVersionId': 'string'
},
{
'Key': 'string',
},
],
'Error': [
{
'Key': 'string',
'VersionId': 'string',
'Code': 'string',
'Message': 'string'
},
]
}
```

参数名称	参数描述	类型

参数名称	参数描述	类型
Deleted	删除成功的 Object 信息	List
Key	删除成功的 Object 的路径	String
VersionId	删除成功的 Object 的版本号	String
DeleteMarker	删除成功的 Object 是否为 delete marker	String
DeleteMarkerVersionId	删除成功的 Object 的 delete marker 的版本号	String
Error	删除失败的 Object 信息	List
Key	删除失败的 Object 的路径	String
VersionId	删除失败的 Object 的版本号	String
Code	删除失败的 Object 对应的错误码	String
Message	删除失败的 Object 对应的错误信息	String

分块操作

分块上传对象可包括的操作：

- 分块上传对象：初始化分块上传，上传分块，完成所有分块上传。
- 分块续传：查询已上传的分块，上传分块，完成所有分块上传。
- 删除已上传分块。

查询分块上传

功能说明

查询指定存储桶正在进行中的分块上传信息（List Multipart Uploads）。

方法原型

```
list_multipart_uploads(Bucket, Prefix="", Delimiter="", KeyMarker="", UploadIdMarker="", MaxUploads=1000, EncodingType="", **kwargs)
```

请求示例

```
response = client.list_multipart_uploads(
Bucket='examplebucket-1250000000',
Prefix='string',
Delimiter='string',
KeyMarker='string',
UploadIdMarker='string',
MaxUploads=100,
EncodingType='url'
)
```

参数说明

参数名称	参数描述	类型	必填
Bucket	存储桶名称，由 BucketName-APPID 构成	String	是
Prefix	默认为空，对分块上传的 key 进行筛选，匹配 prefix 为前缀的分块上传	String	否
Delimiter	默认为空，设置分隔符	String	否
KeyMarker	和 UploadIdMarker 一起使用，指明列出分块上传的起始位置	String	否
UploadIdMarker	和 KeyMarker 一起使用，指明列出分块上传的起始位置。如果未指定 KeyMarker，UploadIdMarker 将被忽略	String	否
MaxUploads	最多返回的分块上传的数量，默认为最大的1000	Int	否

参数名称	参数描述	类型	必填
EncodingType	默认不编码，规定返回值的编码方式，可选值：url	String	否

返回结果说明

获取分块上传的信息，类型为 dict：

```
{
'Bucket': 'examplebucket-1250000000',
'Prefix': 'string',
'Delimiter': 'string',
'KeyMarker': 'string',
'UploadIdMarker': 'string',
'NextKeyMarker': 'string',
'NextUploadIdMarker': 'string',
'MaxUploads': '1000',
'IsTruncated': 'true'|'false',,
'EncodingType': 'url',
'Upload':[
{
'UploadId': 'string',
'Key': 'string',
'Initiated': 'string',
'StorageClass': 'STANDARD',
'Owner': {
'DisplayName': 'string',
'ID': 'string'
},
'Initiator': {
'ID': 'string',
'DisplayName': 'string'
}
},
],
'CommonPrefixes':[
{
'Prefix': 'string'
},
],
}
```

参数名称	参数描述	类型
Bucket	存储桶名称，由 BucketName-APPID 构成	String
Prefix	默认为空，对分块上传的 key 进行筛选，匹配 prefix 为前缀的分块上传	String
Delimiter	默认为空，设置分隔符	String
KeyMarker	和 UploadIdMarker 一起使用，指明列出分块上传的 key 起始位置	String
UploadIdMarker	和 KeyMarker 一起使用，指明列出分块上传的 uploadid 起始位置。如果未指定 KeyMarker，UploadIdMarker 将被忽略	String
NextKeyMarker	当 IsTruncated 为 true 时，指明下一次列出分块上传的 key 的起始位置	String
NextUploadIdMarker	当 IsTruncated 为 true 时，指明下一次列出分块上传的 uploadid 的起始位置	String
MaxUploads	最多返回的分块上传的数量，默认为最大的1000	Int
IsTruncated	表示返回的分块上传否被截断	String
EncodingType	默认不编码，规定返回值的编码方式，可选值：url	String
Upload	包含所有分块上传的 list，包括 'UploadId'，'StorageClass'，'Key'，'Owner'，'Initiator'，'Initiated' 等信息	List
CommonPrefixes	所有以 Prefix 开头，以 Delimiter 结尾的 Key 被归到同一类	List

初始化分块上传

功能说明

初始化分块上传，获取对应的 uploadId（Initiate Multipart Upload）。

方法原型

```
create_multipart_upload(Bucket, Key, **kwargs):
```

请求示例

```
response = client.create_multipart_upload(
Bucket='examplebucket-1250000000',
Key='multipart.txt',
StorageClass='STANDARD',
Expires='string',
CacheControl='string',
ContentType='string',
ContentDisposition='string',
ContentEncoding='string',
ContentLanguage='string',
Metadata={
'x-cos-meta-key1': 'value1',
'x-cos-meta-key2': 'value2'
},
ACL='private'|'public-read',
GrantFullControl='string',
GrantRead='string'
)
# 获取UploadId供后续接口使用
uploadid = response['UploadId']
```

参数说明

参数名称	参数描述	类型	必填
Bucket	Bucket 名称，由 BucketName-APPID 构成	String	是
Key	对象键（Key）是对象在存储桶中的唯一标识。例如，在对象的访问域名`examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/pic.jpg`中，对象键为 doc/pic.jpg	String	是
StorageClass	设置对象的存储类型，默认值 STANDARD	String	否
Expires	设置 Expires	String	否
CacheControl	缓存策略，设置 Cache-Control	String	否
ContentType	内容类型，设置 Content-Type	String	否
ContentDisposition	文件名称，设置 Content-Disposition	String	否
ContentEncoding	编码格式，设置 Content-Encoding	String	否
ContentLanguage	语言类型，设置 Content-Language	String	否
Metadata	用户自定义的对象元数据	Dict	否
ACL	设置对象的 ACL，例如 'private'，'public-read'	String	否
GrantFullControl	赋予被授权者所有的权限，格式为`id="OwnerUin"`	String	否
GrantRead	赋予被授权者读的权限，格式为`id="OwnerUin"`	String	否

返回结果说明

获取分块上传的初始化信息，类型为 dict：

```
{
'UploadId': '150219101333cecfd6718d0caea1e2738401f93aa531a4be7a2afee0f8828416f3278e5570',
'Bucket': 'examplebucket-1250000000',
'Key': 'exampleobject'
}
```

参数名称	参数描述	类型
UploadId	标识分块上传的 ID	String
Bucket	存储桶名称，由 BucketName-APPID 组成	String
Key	对象键（Key）是对象在存储桶中的唯一标识。例如，在对象的访问域名`examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/pic.jpg`中，对象键为 doc/pic.jpg	String

上传分块

分块上传对象（Upload Part）。

方法原型

```
upload_part(Bucket, Key, Body, PartNumber, UploadId, **kwargs)
```

请求示例

```
# 注意，上传分块的块数最多10000块
response = client.upload_part(
Bucket='examplebucket-1250000000',
Key='exampleobject',
Body=b'bytes'|file,
PartNumber=1,
UploadId='string',
EnableMD5=False|True,
ContentLength='123',
ContentMD5='string'
)
```

参数说明

参数名称	参数描述	类型	必填
Bucket	Bucket 名称，由 BucketName-APPID 构成	String	是
Key	对象键（Key）是对象在存储桶中的唯一标识。例如，在对象的访问域名`examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/pic.jpg`中，对象键为 doc/pic.jpg	String	是
Body	上传分块的内容，可以为本地文件流或输入流	file/bytes	是
PartNumber	标识上传分块的序号	Int	是
UploadId	标识分块上传的 ID	String	是
EnableMD5	是否需要 SDK 计算 Content-MD5，默认关闭，打开后会增加上传耗时	Bool	否
ContentLength	设置传输长度	String	否
ContentMD5	设置上传对象的 MD5 值用于校验	String	否

返回结果说明

上传分块的属性，类型为 dict：

```
{
'ETag': 'string'
}
```

参数名称	参数描述	类型
ETag	上传分块的 MD5 值。	String

复制分块

将其他对象复制为一个分块（Upload Part - Copy）。

方法原型

```
upload_part_copy(Bucket, Key, PartNumber, UploadId, CopySource, CopySourceRange='', **kwargs)
```

请求示例

```
response = client.upload_part_copy(
Bucket='examplebucket-1250000000',
Key='exampleobject',
PartNumber=100,
UploadId='string',
CopySource={
'Bucket': 'examplebucket-1250000000',
'Key': 'exampleobject',
'Endpoint': 'example.endpoint',
'VersionId': 'string'
},
CopySourceRange='string',
CopySourceIfMatch='string',
CopySourceIfModifiedSince='string',
CopySourceIfNoneMatch='string',
CopySourceIfUnmodifiedSince='string'
)
```

参数说明

参数名称	参数描述	类型	必填
Bucket	存储桶名称，由 BucketName-APPID 构成	String	是
Key	对象键（Key）是对象在存储桶中的唯一标识。例如，在对象的访问域名`examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/pic.jpg`中，对象键为 doc/pic.jpg	String	是
PartNumber	标识上传分块的序号	Int	是
UploadId	标识分块上传的 ID	String	是
CopySource	描述拷贝源对象的路径，包含 Bucket、Key、Endpoint、VersionId	Dict	是
CopySourceRange	描述拷贝源对象的范围，格式为 bytes=first-last。不指定时，默认拷贝整个源对象	String	否
CopySourceIfMatch	源对象的 Etag 与给定的值相同时才拷贝	String	否
CopySourceIfModifiedSince	源对象在给定的时间后被修改相才拷贝	String	否
CopySourceIfNoneMatch	源对象的 Etag 与给定的值不相同时才拷贝	String	否
CopySourceIfUnmodifiedSince	源对象在给定的时间后没有修改相才拷贝	String	否

返回结果说明

复制分块的属性，类型为 dict：

```
{
'ETag': 'string',
'LastModified': 'string',
'x-cos-copy-source-version-id': 'string',
}
```

参数名称	参数描述	类型
ETag	拷贝分块的 MD5 值	String
LastModified	拷贝分块的最后一次修改时间	String
x-cos-copy-source-version-id	源对象的版本号	String

查询已上传块

功能说明

查询特定分块上传操作中的已上传的块（List Parts）。

方法原型

```
list_parts(Bucket, Key, UploadId, MaxParts=1000, PartNumberMarker=0, EncodingType='', **kwargs)
```

请求示例

```
response = client.list_parts(
Bucket='examplebucket-1250000000',
Key='exampleobject',
UploadId=uploadid,
MaxParts=1000,
PartNumberMarker=100,
EncodingType='url'
)
```

参数说明

参数名称	参数描述	类型	必填
Bucket	存储桶名称，由 BucketName-APPID 构成	String	是
Key	对象键（Key）是对象在存储桶中的唯一标识。例如，在对象的访问域名`examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/pic.jpg`中，对象键为 doc/pic.jpg	String	是
UploadId	标识分块上传的 ID	String	是
MaxParts	最多返回的分块的数量，默认为最大的1000	Int	否
PartNumberMarker	默认为0，从第一块列出分块，从 PartNumberMarker 下一个分块开始列出	Int	否
EncodingType	默认不编码，规定返回值的编码方式，可选值：url	String	否

返回结果说明

所有上传分块的信息，类型为 dict：

```
{
'Bucket': 'examplebucket-1250000000',
'Key': 'exampleobject',
'UploadId': '1502192444bdb382add546a35b2eeab81e06ed84086ca0bb75ea45ca7fa073fa9cf74ec4f2',
'EncodingType': None,
'MaxParts': '1000',
'IsTruncated': 'true',
'PartNumberMarker': '0',
'NextPartNumberMarker': '1000',
'StorageClass': 'Standard',
'Part': [
{
'LastModified': '2017-08-08T11:40:48.000Z',
'PartNumber': '1',
'ETag': '"8b8378787c0925f42ccb829f6cc2fb97"',
'Size': '10485760'
},
],
'Initiator': {
'DisplayName': '3333333333',
'ID': 'qcs::cam::uin/3333333333:uin/3333333333'
},
'Owner': {
'DisplayName': '124564654654',
'ID': '124564654654'
}
}
```

参数名称	参数描述	类型
Bucket	存储桶名称，由 BucketName-APPID 构成	String

参数名称	参数描述	类型
Key	对象键（Key）是对象在存储桶中的唯一标识。例如，在对象的访问域名`examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/pic.jpg`中，对象键为 doc/pic.jpg	String
UploadId	标识分块上传的 ID	String
EncodingType	默认不编码，规定返回值的编码方式，可选值：url	String
MaxParts	最多返回的分块的数量，默认为最大的1000	String
IsTruncated	表示返回的分块是否被截断	String
PartNumberMarker	默认为0，从第一块列出分块，从 PartNumberMarker 下一个分块开始列出	String
NextPartNumberMarker	指明下一次列出分块的起始位置	String
StorageClass	对象的存储类型，默认值 STANDARD	String
Part	上传分块的相关信息，包括 ETag，PartNumber，Size，LastModified	String
Initiator	分块上传的创建者，包括 DisplayName 和 ID	Dict
Owner	对象拥有者的信息，包括 DisplayName 和 ID	Dict

完成分块上传

功能说明

完成整个对象的分块上传（Complete Multipart Upload）。

方法原型

```
complete_multipart_upload(Bucket, Key, UploadId, MultipartUpload={}, **kwargs)
```

请求示例

```
response = client.complete_multipart_upload(
Bucket='examplebucket-1250000000',
Key='exampleobject',
UploadId=uploadid,
MultipartUpload={
'Part': [
{
'ETag': 'string',
'PartNumber': 1
},
{
'ETag': 'string',
'PartNumber': 2
},
]
},
)
```

参数说明

参数名称	参数描述	类型	必填
Bucket	Bucket 名称，由 BucketName-APPID 构成	String	是
Key	对象键（Key）是对象在存储桶中的唯一标识。例如，在对象的访问域名`examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/pic.jpg`中，对象键为 doc/pic.jpg	String	是
UploadId	标识分块上传的 ID	String	是
MultipartUpload	所有分块的 ETag 和 PartNumber 信息	Dict	是

返回结果说明

组装后的对象的相关信息，类型为 dict：

```
{
'ETag': '"3f866d0050f044750423e0a4104fa8cf-2"',
'Bucket': 'examplebucket-1250000000',
'Location': 'examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/exampleobject',
'Key': 'exampleobject'
}
```

参数名称	参数描述	类型
ETag	合并后对象的唯一标签值，该值不是对象内容的 MD5 校验值，仅能用于检查对象唯一性。如需校验对象内容，可以在上传过程中校验单个分块的ETag值。	String
Bucket	存储桶名称，由 BucketName-APPID 构成	String
Location	URL 地址	String
Key	对象键（Key）是对象在存储桶中的唯一标识。例如，在对象的访问域名`examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/pic.jpg`中，对象键为 doc/pic.jpg	String

终止分块上传

功能说明

终止一个分块上传操作并删除已上传的块（Abort Multipart Upload）。

方法原型

```
abort_multipart_upload(Bucket, Key, UploadId, **kwargs)
```

请求示例

```
response = client.abort_multipart_upload(
Bucket='examplebucket-1250000000',
Key='exampleobject',
UploadId=uploadid
)
```

参数说明

参数名称	参数描述	类型	必填
Bucket	存储桶名称，由 BucketName-APPID 构成	String	是
Key	对象键（Key）是对象在存储桶中的唯一标识。例如，在对象的访问域名`examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/pic.jpg`中，对象键为 doc/pic.jpg	String	是
UploadId	标识分块上传的 ID	String	是

返回结果说明

该方法返回值为 None。

其他操作

设置对象 ACL

功能说明

设置存储桶中某个对象的访问控制列表（PUT Object acl）。AccessControlPolicy 参数与其它权限参数是互斥的，无法同时指定。

方法原型

```
put_object_acl(Bucket, Key, AccessControlPolicy={}, **kwargs)
```

请求示例

```
response = client.put_object_acl(
Bucket='examplebucket-1250000000',
Key='exampleobject',
ACL='private'|'public-read',
GrantFullControl='string',
GrantRead='string',
AccessControlPolicy={
'AccessControlList': {
'Grant': [
{
'Grantee': {
'DisplayName': 'string',
'Type': 'CanonicalUser'|'Group',
'ID': 'string',
'URI': 'string'
},
'Permission': 'FULL_CONTROL'|'WRITE'|'READ'
},
]
},
'Owner': {
'DisplayName': 'string',
'ID': 'string'
}
}
)
```

参数说明

参数名称	参数描述	类型	必填
Bucket	存储桶名称，由 BucketName-APPID 构成	String	是
Key	对象键（Key）是对象在存储桶中的唯一标识。例如，在对象的访问域名`examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/pic.jpg`中，对象键为 doc/pic.jpg	String	是
ACL	设置对象的 ACL，例如 'private'，'public-read'	String	否
GrantFullControl	赋予指定账户对对象的所有权限，格式为`id="OwnerUin"`	String	否
GrantRead	赋予指定账户对对象的读权限，格式为`id="OwnerUin"`	String	否
AccessControlPolicy	赋予指定账户对对象的访问权限，具体格式见 GET Object acl 返回结果说明	Dict	否

返回结果说明

该方法返回值为 None。

查询对象 ACL

功能说明

查询对象的访问控制列表（GET Object acl）。

方法原型

```
get_object_acl(Bucket, Key, **kwargs)
```

请求示例

```
response = client.get_object_acl(
Bucket='examplebucket-1250000000',
Key='exampleobject'
)
```

参数说明

参数名称	参数描述	类型	必填

参数名称	参数描述	类型	必填
Bucket	存储桶名称，由 BucketName-APPID 构成	String	是
Key	对象键（Key）是对象在存储桶中的唯一标识。例如，在对象的访问域名`examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/pic.jpg`中，对象键为 doc/pic.jpg	String	是

返回结果说明

Bucket ACL 信息，类型为 Dict：

```
{
'Owner': {
'DisplayName': 'string',
'ID': 'string'
},
'Grant': [
{
'Grantee': {
'DisplayName': 'string',
'Type': 'CanonicalUser'|'Group',
'ID': 'string',
'URI': 'string'
},
'Permission': 'FULL_CONTROL'|'WRITE'|'READ'
},
]
}
```

参数名称	参数描述	类型
Owner	对象拥有者的信息，包括 DisplayName 和 ID	Dict
Grant	对象权限授予者的信息，包括 Grantee 和 Permission	List
Grantee	对象权限授予者的信息，包括 DisplayName，Type，ID 和 URI	Dict
DisplayName	权限授予者的名字	String
Type	权限授予者的类型，类型为 CanonicalUser 或者 Group	String
ID	Type 为 CanonicalUser 时，对应权限授予者的 ID	String
URI	Type 为 Group 时，对应权限授予者的 URI	String
Permission	授予者所拥有的对象的权限，可选值有 FULL_CONTROL，WRITE，READ，分别对应所有权限、写权限、读权限	String

高级接口（推荐）

上传对象（断点续传）

功能说明

该高级接口根据用户文件的长度自动选择简单上传以及分块上传，对于小于等于20M的文件调用简单上传，对于大于20MB的文件调用分块上传，对于分块上传未完成的文件会自动进行断点续传。

方法原型

```
upload_file(Bucket, Key, LocalFilePath, PartSize=1, MAXThread=5, **kwargs)
```

请求示例

```
response = client.upload_file(
Bucket='examplebucket-1250000000',
Key='exampleobject',
LocalFilePath='local.txt',
EnableMD5=False
)
```

全部参数请求示例

```
response = client.upload_file(
Bucket='examplebucket-1250000000',
Key='exampleobject',
LocalFilePath='local.txt',
PartSize=1,
MAXThread=5,
EnableMD5=False|True,
ACL='private'|'public-read', # 请慎用此参数,否则会达到1000条ACL上限
GrantFullControl='string',
GrantRead='string',
StorageClass='STANDARD',
Expires='string',
CacheControl='string',
ContentType='string',
ContentDisposition='string',
ContentEncoding='string',
ContentLanguage='string',
ContentLength='123',
ContentMD5='string',
Metadata={
'x-cos-meta-key1': 'value1',
'x-cos-meta-key2': 'value2'
}
)
```

参数说明

参数名称	参数描述	类型	必填
Bucket	存储桶名称，由 BucketName-APPID 构成	String	是
Key	对象键（Key）是对象在存储桶中的唯一标识。例如，在对象的访问域名`examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/pic.jpg`中，对象键为 doc/pic.jpg	String	是
LocalFilePath	本地文件的路径名	String	是
PartSize	分块上传的分块大小，默认为1MB	Int	否
MAXThread	分块上传的并发数量，默认为5个线程上传分块	Int	否
EnableMD5	是否需要 SDK 计算 Content-MD5，默认关闭，打开后会增加上传耗时	Bool	否
ACL	设置对象的 ACL，例如 'private'，'public-read'	String	否
GrantFullControl	赋予被授权者所有的权限，格式为`id="OwnerUin"`	String	否
GrantRead	赋予被授权者读的权限，格式为`id="OwnerUin"`	String	否
StorageClass	设置对象的存储类型，默认值 STANDARD	String	否
Expires	设置 Expires	String	否
CacheControl	缓存策略，设置 Cache-Control	String	否
ContentType	内容类型，设置 Content-Type	String	否
ContentDisposition	文件名称，设置 Content-Disposition	String	否
ContentEncoding	编码格式，设置 Content-Encoding	String	否
ContentLanguage	语言类型，设置 Content-Language	String	否
ContentLength	设置传输长度	String	否
ContentMD5	设置上传对象的 MD5 值用于校验	String	否
Metadata	用户自定义的对象元数据	Dict	否

返回结果说明

上传对象的属性，类型为 dict：

```
{
'ETag': 'string'
}
```

参数名称	参数描述	类型
ETag	上传对象的 MD5 值	String

存储桶管理

简介

本文档提供关于跨域访问、生命周期、版本控制和跨地域复制相关的 API 概览以及 SDK 示例代码。

跨域访问

API	操作名	操作描述
PUT Bucket cors	设置跨域配置	设置存储桶的跨域访问权限
GET Bucket cors	查询跨域配置	查询存储桶的跨域访问配置信息
DELETE Bucket cors	删除跨域配置	删除存储桶的跨域访问配置信息

版本控制

API	操作名	操作描述
PUT Bucket versioning	设置版本控制	设置存储桶的版本控制功能
GET Bucket versioning	查询版本控制	查询存储桶的版本控制信息

跨地域复制

API	操作名	操作描述
PUT Bucket replication	设置跨地域复制	设置存储桶的跨地域复制规则
GET Bucket replication	查询跨地域复制	查询存储桶的跨地域复制规则
DELETE Bucket replication	删除跨地域复制	删除存储桶的跨地域复制规则

跨域访问

设置跨域配置

功能说明

设置指定存储桶的跨域访问配置信息（PUT Bucket cors）。

方法原型

```
put_bucket_cors(Bucket, CORSConfiguration={}, **kwargs)
```

请求示例

```
response = client.put_bucket_cors(
Bucket='examplebucket-1250000000',
CORSConfiguration={
'CORSRule': [
```

```
{
'ID': 'string',
'MaxAgeSeconds': 100,
'AllowedOrigin': [
'string',
],
'AllowedMethod': [
'string',
],
'AllowedHeader': [
'string',
],
'ExposeHeader': [
'string',
]
}
]
},
)
```

参数说明

参数名称	参数描述	类型	必填
Bucket	存储桶名称，由 BucketName-APPID 构成	String	是
CORSRule	设置对应的跨域规则，包括 ID，MaxAgeSeconds，AllowedOrigin，AllowedMethod，AllowedHeader，ExposeHeader	List	是
ID	设置规则的 ID	String	否
MaxAgeSeconds	设置 OPTIONS 请求得到结果的有效期	Int	否
AllowedOrigin	设置允许的访问来源，如 `"http://imgcache.finance.cloud.tencent.com:80finance.cloud.tencent.com"`，支持通配符`*`	Dict	是
AllowedMethod	设置允许的方法，如 GET，PUT，HEAD，POST，DELETE	Dict	是
AllowedHeader	设置请求可以使用哪些自定义的 HTTP 请求头部，支持通配符`*`	Dict	否
ExposeHeader	设置浏览器可以接收到的来自服务器端的自定义头部信息	Dict	否

返回结果说明

该方法返回值为 None。

查询跨域配置

功能说明

查询指定存储桶的跨域访问配置信息（GET Bucket cors）。

方法原型

```
get_bucket_cors(Bucket, **kwargs)
```

请求示例

```
response = client.get_bucket_cors(
Bucket='examplebucket-1250000000',
)
```

参数说明

参数名称	参数描述	类型	必填
Bucket	存储桶名称，由 BucketName-APPID 构成	String	是

返回结果说明

存储桶跨域配置，类型为 dict。

```
{
'CORSRule': [
{
'ID': 'string',
'MaxAgeSeconds': 100,
'AllowedOrigin': [
'string',
],
'AllowedMethod': [
'string',
],
'AllowedHeader': [
'string',
],
'ExposeHeader': [
'string',
],
}
]
}
```

参数名称	参数描述	类型
CORSRule	跨域规则，包括 ID，MaxAgeSeconds，AllowedOrigin，AllowedMethod，AllowedHeader，ExposeHeader	List
ID	规则的 ID	String
MaxAgeSeconds	OPTIONS 请求得到结果的有效期	String
AllowedOrigin	允许的访问来源，如 `"http://imgcache.finance.cloud.tencent.com:80finance.cloud.tencent.com"`，支持通配符`*`	Dict
AllowedMethod	允许的方法，如 GET，PUT，HEAD，POST，DELETE	Dict
AllowedHeader	请求可以使用哪些自定义的 HTTP 请求头部，支持通配符`*`	Dict
ExposeHeader	浏览器可以接收到的来自服务器端的自定义头部信息	Dict

删除跨域配置

功能说明

删除指定存储桶的跨域访问配置（DELETE Bucket cors）。

方法原型

```
delete_bucket_cors(Bucket, **kwargs)
```

请求示例

```
response = client.delete_bucket_cors(
Bucket='examplebucket-1250000000',
)
```

参数说明

参数名称	参数描述	类型	必填
Bucket	存储桶名称，由 BucketName-APPID 构成	String	是

返回结果说明

该方法返回值为 None。

版本控制

设置版本控制

功能说明

设置指定存储桶的版本控制功能（PUT Bucket versioning）。

方法原型

```
put_bucket_versioning(Bucket, Status, **kwargs)
```

请求示例

开启版本控制

```
response = client.put_bucket_versioning(
Bucket='examplebucket-1250000000',
Status='Enabled'
)
```

暂停版本控制

```
response = client.put_bucket_versioning(
Bucket='examplebucket-1250000000',
Status='Suspended'
)
```

参数说明

参数名称	参数描述	类型	必填
Bucket	存储桶名称，由 BucketName-APPID 构成	String	是
Status	设置存储桶版本控制的状态，可选值为 'Enabled'， 'Suspended'	String	是

返回结果说明

该方法返回值为 None。

查询版本控制

功能说明

查询指定存储桶的版本控制信息（GET Bucket versioning）。

方法原型

```
get_bucket_versioning(Bucket, **kwargs)
```

请求示例

```
response = client.get_bucket_versioning(
Bucket='examplebucket-1250000000',
)
```

参数说明

参数名称	参数描述	类型	必填
Bucket	存储桶名称，由 BucketName-APPID 构成	String	是

返回结果说明

Bucket 版本控制配置，类型为 dict。

```
{
'Status': 'Enabled'|'Suspended'
}
```

参数名称	参数描述	类型
Status	存储桶版本控制的状态，可选值为 'Enabled'，Suspended'	String

跨地域复制

设置跨地域复制

功能说明

设置指定存储桶的跨地域复制规则（PUT Bucket replication）。

方法原型

```
put_bucket_replication(Bucket, ReplicationConfiguration={}, **kwargs)
```

请求示例

```
response = client.put_bucket_replication(
Bucket='examplebucket-1250000000',
ReplicationConfiguration={
'Role': 'qcs::cam::uin/100000000001:uin/100000000001',
'Rule': [
{
'ID': 'string',
'Status': 'Enabled'|'Disabled',
'Prefix': 'string',
'Destination': {
'Bucket': 'qcs::cos:ap-shanghai::examplebucket1-1250000000',
'StorageClass': 'STANDARD'
}
},
{
'ID': 'string',
'Status': 'Enabled'|'Disabled',
'Prefix': 'string',
'Destination': {
'Bucket': 'qcs::cos:ap-guangzhou::examplebucket2-1250000000',
'StorageClass': 'STANDARD'
}
},
]
}
)
```

参数说明

参数名称	参数描述	类型	必填
Bucket	源存储桶名称，由 BucketName-APPID 构成	String	是
Role	发起者身份标示, 格式为`qcs::cam::uin/:uin/`	String	否
Rule	设置对应的规则，包括 ID，Status，Prefix，Destination	List	是
ID	设置规则的 ID	String	否
Status	设置 Rule 是否启用，可选值为 Enabled 或者 Disabled	String	是
Prefix	设置 Rule 的前缀匹配规则，为空时表示作用存储桶中的所有对象	String	是
Destination	描述目的资源，包括 Bucket 和StorageClass	Dict	是
Bucket	设置跨地域复制的目标存储桶，格式为`qcs::cos:[region]::[BucketName-APPID]`	String	是
StorageClass	设置目的文件的存储类型，可选值为 'STANDARD'	String	否

返回结果说明

该方法返回值为 None。

查询跨地域复制

功能说明

查询指定存储桶的跨地域复制规则（GET Bucket replication）。

方法原型

```
get_bucket_replication(Bucket, **kwargs)
```

请求示例

```
response = client.get_bucket_replication(
Bucket='examplebucket-1250000000'
)
```

参数说明

参数名称	参数描述	类型	必填
Bucket	存储桶名称，由 BucketName-APPID 构成	String	是

返回结果说明

Bucket 跨地域复制配置，类型为 dict。

```
{
'Role': 'qcs::cam::uin/100000000001:uin/100000000001',
'Rule': [
{
'ID': 'string',
'Status': 'Enabled'|'Disabled',
'Prefix': 'string',
'Destination': {
'Bucket': 'qcs::cos:ap-shanghai::examplebucket1-1250000000',
'StorageClass': 'STANDARD'
}
},
{
'ID': 'string',
'Status': 'Enabled'|'Disabled',
'Prefix': 'string',
'Destination': {
'Bucket': 'qcs::cos:ap-guangzhou::examplebucket2-1250000000',
'StorageClass': 'STANDARD'
}
},
]
}
```

参数名称	参数描述	类型
Role	发起者身份标示, 格式为`qcs::cam::uin/:uin/`	String
Rule	跨地域复制对应的规则，包括 ID，Status，Prefix，Destination	List
ID	跨地域复制规则的 ID	String
Status	跨地域复制 Rule 是否启用，可选值为 Enabled 或者 Disabled	String
Prefix	跨地域复制 Rule 的前缀匹配规则，为空时表示作用存储桶中的所有对象	String
Destination	描述目的资源，包括 Bucket 和 StorageClass	Dict
Bucket	跨地域复制的目标存储桶，格式为`qcs::cos:[region]::[BucketName-APPID]`	String
StorageClass	目的文件的存储类型，可选值为 'STANDARD'	String

删除跨地域复制

功能说明

删除指定存储桶的跨地域复制规则（DELETE Bucket replication）。

方法原型

```
delete_bucket_replication(Bucket, **kwargs)
```

请求示例

```
response = client.delete_bucket_replication(
Bucket='examplebucket-1250000000',
)
```

参数说明

参数名称	参数描述	类型	必填
Bucket	存储桶名称，由 BucketName-APPID 构成	String	是

返回结果说明

该方法返回值为 None。

预签名 URL

简介

Python SDK 提供获取签名，获取请求预签名 URL 接口以及获取对象下载预签名 URL 接口。使用永久密钥或临时密钥获取预签名 URL 的调用方法相同，使用临时密钥时需要在 header 或 query_string 中加上 x-cos-security-token。

获取签名

功能说明

获取指定操作的签名，常用于移动端的签名分发。

方法原型

```
get_auth(Method, Bucket, Key, Expired=300, Headers={}, Params={})
```

上传请求示例

```
response = client.get_auth(
Method='PUT',
Bucket='examplebucket-1250000000',
Key='exampleobject'
)
```

下载请求示例

```
response = client.get_auth(
Method='GET',
Bucket='examplebucket-1250000000',
Key='exampleobject'
)
```

全部参数请求示例

```
response = client.get_auth(
Method='PUT'|'POST'|'GET'|'DELETE'|'HEAD',
Bucket='examplebucket-1250000000',
Key='exampleobject',
Expired=300,
Headers={
'Content-Length': 'string',
'Content-MD5': 'string'
},
Params={
'param1': 'string',
'param2': 'string'
}
)
```

参数说明

参数名称	参数描述	类型	必填
Method	对应操作的 Method, 可选值为 'PUT'，'POST'，'GET'，'DELETE'，'HEAD'	String	是
Bucket	存储桶名称，由 BucketName-APPID 构成	String	是
Key	Bucket 操作填入根路径`/`，object 操作填入文件的路径	String	是
Expired	签名过期时间，单位为秒	Int	否
Headers	需要签入签名的请求头部	Dict	否
Params	需要签入签名的请求参数	Dict	否

返回结果说明

该方法返回值为对应操作的签名值。

获取预签名 URL

功能说明

获取预签名链接用于分发。

上传请求示例

```
response = client.get_presigned_url(
Method='PUT',
Bucket='examplebucket-1250000000',
Key='exampleobject'
)
```

下载请求示例

```
response = client.get_presigned_url(
Method='GET',
Bucket='examplebucket-1250000000',
Key='exampleobject'
)
```

方法原型

```
get_presigned_url(Bucket, Key, Method, Expired=300, Params={}, Headers={})
```

请求示例

```
response = client.get_presigned_url(
Bucket='examplebucket-1250000000',
Key='exampleobject',
```

```
Method='PUT'|'POST'|'GET'|'DELETE'|'HEAD',
Expired=300,
Headers={
'Content-Length': 'string',
'Content-MD5': 'string'
},
Params={
'param1': 'string',
'param2': 'string'
}
)
```

参数说明

参数名称	参数描述	类型	必填
Bucket	存储桶名称，由 BucketName-APPID 构成	String	是
Key	对象键（Key）是对象在存储桶中的唯一标识。例如，在对象的访问域名 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/pic.jpg` 中，对象键为 doc/pic.jpg	String	是
Method	对应操作的 Method, 可选值为 'PUT'，'POST'，'GET'，'DELETE'，'HEAD'	String	是
Expired	签名过期时间，单位为秒	Int	否
Params	签名中要签入的请求参数	Dict	否
Headers	签名中要签入的请求头部	Dict	否

返回结果说明

该方法返回值为预签名的 URL。

获取预签名下载 URL

功能说明

获取预签名下载链接用于直接下载。

方法原型

```
get_presigned_download_url(Bucket, Key, Expired=300, Params={}, Headers={})
```

请求示例

```
response = client.get_presigned_download_url(
Bucket='examplebucket-1250000000',
Key='exampleobject'
)
```

参数说明

参数名称	参数描述	类型	必填
Bucket	存储桶名称，由 BucketName-APPID 构成	String	是
Key	对象键（Key）是对象在存储桶中的唯一标识。例如，在对象的访问域名 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/doc/pic.jpg` 中，对象键为 doc/pic.jpg	String	是
Expired	签名过期时间，单位为秒	Int	否
Params	签名中要签入的请求参数	Dict	否
Headers	签名中要签入的请求头部	Dict	否

返回结果说明

该方法返回值为预签名的下载 URL。

异常处理

简介

COS XML Python SDK 操作成功会返回一个 dict 或者 None。若调用 SDK 接口请求 COS 服务失败，系统将抛出 CosClientError（客户端异常）或者 CosServiceError（服务端异常）。

- CosClientError 是由于客户端无法和 COS 服务端正常进行交互所引起。如客户端无法连接到服务端，无法解析服务端返回的数据，读取本地文件发生 IO 异常等。
- CosServiceError 是客户端和 COS 服务端交互正常，但操作 COS 资源失败。如客户端访问一个不存在的存储桶，删除一个不存在的对象，没有权限进行某个操作等。

客户端异常

CosClientError 一般指如 timeout 引起的客户端错误，用户捕获后可以选择重试或其它操作。

服务端异常

CosServiceError 提供服务端返回的具体信息，包含了服务端返回的状态码、requestid 和出错明细等。捕获异常后，建议对整个异常进行打印，异常包含了必须的排查因素。以下是异常成员变量的描述以及异常捕获示例。

成员	描述	类型
request_id	请求 ID，用于唯一标识一个请求，对于排查问题十分重要	string
status_code	response 的 status 状态码	string
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

异常捕获示例

```
from qcloud_cos import CosServiceError

except CosServiceError as e:
e.get_origin_msg() # 获取原始错误信息，格式为XML
e.get_digest_msg() # 获取处理过的错误信息，格式为dict
e.get_status_code() # 获取 http 错误码（如4XX，5XX）
e.get_error_code() # 获取 COS 定义的错误码
e.get_error_msg() # 获取 COS 错误码的具体描述
e.get_trace_id() # 获取请求的 trace_id
e.get_request_id() # 获取请求的 request_id
e.get_resource_location() # 获取 URL 地址
```

COS API参考

简介

最近更新时间: 2025-01-15 17:01:00

描述

对象存储的 XML API 是一种轻量级的，无连接状态的接口。您可以调用此套接口直接通过 http/https 发出请求和接受响应，从而实现与云平台对象存储（COS）后台进行交互操作。此套API的请求和响应内容都为XML格式。

在查阅其他的 API 文档之前，首先请详细阅读签名鉴权 。

基本信息

本部分介绍是为了您更有效的使用 COS ，而必须要了解的主要概念和术语。

名称	描述
AppID	开发者访问 COS 服务时拥有的用户维度唯一资源标识，用以标示资源。
SecretID	SecretID 是开发者拥有的项目身份识别 ID，用以身份认证
SecretKey	SecretKey 是开发者拥有的项目身份密钥。
Bucket	存储桶是 COS 中用于存储数据的容器，是用户存储在 Appid 下的第一级目录，每个对象都存储在一个存储桶中。
Object	对象是 COS 中存储的具体文件，是存储的基本实体。
Region	域名中的地域信息，枚举值：cn-east（华东），cn-north（华北），cn-south（华南），sg（新加坡）

快速入门

要使用对象存储 API，需要先执行以下步骤：

1. 获取 Appid、SecretID、SecretKey 内容
2. 编写一个 签名鉴权 算法程序（或使用任何一种服务端 SDK）
3. 计算签名，调用 API 执行操作

公共请求头部

最近更新时间: 2025-01-15 17:01:00

描述

此篇文档将为您介绍在使用 API 时候会使用到的公共请求头部（Request Header），下文提到的头部会在之后的具体 API 文档中不再赘述。

请求头部列表

Header名称	描述	类型	必选
Authorization	携带鉴权信息， 用以验证请求合法性的签名信息。针对公有读的文件，无需携带此头部。	String	否
Content-Length	RFC 2616 中定义的 HTTP 请求内容长度（字节），常用于 PUT 类型的 API 的操作。	String	否
Content-Type	RFC 2616 中定义的 HTTP 请求内容类型（MIME），例如text/plain。	String	否
Content-MD5	RFC 1864 中定义的经过 Base64 编码的 128-bit 内容 MD5 校验值。此头部用来校验文件内容是否发生变化。	String	否
Date	RFC 1123 中定义的 GMT 时间，例如 Wed, 30 Mar. 2016 23:00:00 GMT。	String	否
Expect	当使用 Expect: 100-continue 时，在收到服务端确认后，才会发送请求内容。该选项可以被用于验证头部是否有效，而无需发送数据内容。 有效值：100-continue。	String	否
Host	请求的主机，形式为 -.cos..myqcloud.com。	String	是

公共响应头部

最近更新时间: 2025-01-15 17:01:00

描述

此篇文档将为您介绍在使用API时候会出现的公共返回头部（Response Header），下文提到的头部会在之后的具体API文档中不再赘述。

响应头部列表

Header名称	描述	类型
Content-Length	RFC 2616 中定义的 HTTP 请求内容长度（字节）。	String
Content-Type	RFC 2616 中定义的 HTTP 请求内容类型（MIME）。	String
Connection	声明客户端与服务端之间的通信状态。 枚举值：keep-alive，close。	Enum
Date	服务器端的响应时间，根据 RFC 1123 中定义的 GMT 时间。	String
Etag	ETag 全称 Entity Tag 是 Object 被创建时用于标识 Object 内容的信息标签。此参数并不一定返回 MD5 值，请根据不同请求的情况参考。ETag 的值可以用于检查 Object 的内容是否发生变化。	String
x-cos-request-id	每次请求发送时，服务端将会自动为请求生成一个ID。	String
x-cos-trace-id	每次请求出错时，服务端将会自动为这个错误生成一个ID。	String

错误码

最近更新时间: 2025-01-15 17:01:00

概述

本文将为您介绍请求出错时返回的错误码和对应错误信息。

错误响应

Content-Type：application/xml

对应 HTTP 状态码：3XX，4XX，5XX。特别的，对于PUT Object - Copy接口，即使 HTTP 状态码为 200 也有可能在响应体中包含错误。

响应体

```
<?xml version='1.0' encoding='utf-8' ?>
<Error>
<Code>string</Code>
<Message>string</Message>
<Resource>string</Resource>
<RequestId>string</RequestId>
<TraceId>string</TraceId>
</Error>
```

具体的节点描述如下：

节点名称（关键字）	父节点	描述	类型
Error	无	包含所有的错误信息	Container

Container 节点 Error 的内容：

节点名称（关键字）	父节点	描述	类型
Code	Error	错误码，用来定位唯一的错误条件和确定错误场景，具体错误码见下文	string
Message	Error	具体的错误信息	string
Resource	Error	请求的资源，存储桶地址或对象地址	string
RequestId	Error	每次请求发送时，服务端将会自动为请求生成一个 ID，遇到问题时，该 ID 能更快地协助 COS 定位问题	string
TraceId	Error	每次请求出错时，服务端将会自动为这个错误生成一个ID，遇到问题时，该 ID 能更快地协助 COS 定位问题	string

错误码列表

3XX 类型错误

错误码	描述	HTTP状态码
PermanentRedirect	该资源已经被永久改变了位置，请利用 HTTP Location 响应头部来重定向到正确的新位置	301 Moved Permanently
TemporaryRedirect	该资源已经被临时改变了位置，请利用 HTTP Location 响应头部来重定向到正确的新位置	302 Moved Temporarily
Redirect	临时重定向	307 Moved Temporarily
TemporaryRedirect	在 DNS 更新期间，您将被临时重定向	307 Moved Temporarily

4XX 类型错误

错误码	描述	HTTP 状态码

错误码	描述	HTTP 状态码
AppendPositionErr	Append 操作时，对象长度和 Position 不一致	400 Bad Request
AttachmentFull	ACL 和 Policy 数量到达上限	400 Bad Request
BadDigest	提供的 Content-MD5 值与服务端收到的请求体的 MD5 哈希值不一致	400 Bad Request
EntityTooLarge	上传的对象大小超过规定的最大值	400 Bad Request
EntityTooSmall	上传的对象大小不足规定的最小值，常见于分块上传	400 Bad Request
IncorrectNumberOfFilesInPostRequest	POST 请求每次只允许上传一个对象	400 Bad Request
InvalidArgument	请求参数不合法	400 Bad Request
InvalidBucketName	存储桶名称不合法	400 Bad Request
InvalidCopySource	不合法的复制对象源	400 Bad Request
InvalidDigest	给定的 Content-MD5 值不合法	400 Bad Request
InvalidPart	分块缺失	400 Bad Request
InvalidPartOrder	分块上传编号不连续	400 Bad Request
InvalidRegionName	不合法的地域名	400 Bad Request
InvalidRequest	请求不合法	400 Bad Request
InvalidSHA1Digest	请求内容 SHA1 校验不合法	400 Bad Request
InvalidURI	URI 不合法	400 Bad Request
KeyTooLong	对象键过长	400 Bad Request
LifeCycleIdNotUnique	生命周期 ID 不唯一	400 Bad Request
LifeCycleRuleConflicted	生命周期设置存在冲突	400 Bad Request
MalformedPOSTRequest	该 POST 请求的请求体内容不合法	400 Bad Request
MalformedXML	请求体的 XML 格式不符合 XML 语法	400 Bad Request
MissingAppid	请求头中缺少 APPID	400 Bad Request
MissingContentMD5	请求头中缺少 Content-MD5	400 Bad Request
MissingHost	请求头中缺少 Host	400 Bad Request
MissingRequestBodyError	请求体缺失	400 Bad Request
MultiBucketNotSupport	跨区域复制只能设置一个目的存储桶	400 Bad Request
NoSuchVersion	指定版本不存在	400 Bad Request
NotSupportedStorageClass	指定的存储类型不支持	400 Bad Request
ObjectNotAppendable	指定的对象不能追加	400 Bad Request
PolicyFull	ACL 和 Policy 数量到达上限	400 Bad Request
RequestTimeOut	读取数据超时，检查网络是否过慢或上传并发数过大	400 Bad Request
TooManyBuckets	存储桶数目达到上限 200	400 Bad Request
UnexpectedContent	请求不支持相关内容	400 Bad Request
VerifyAlgorithmNotSupported	校验算法不支持	400 Bad Request
WebsiteURLInvalid	自定义域名 URL 不合法	400 Bad Request
XMLSizeLimit	XML 长度超过限制	400 Bad Request

版权所有：腾讯云金融专区

错误码	描述	HTTP 状态码
AccessDenied	签名或者权限不正确，拒绝访问	403 Forbidden
ExpiredToken	签名串已过期	403 Forbidden
InvalidAccessKeyId	SecretID 不存在	403 Forbidden
InvalidObjectState	请求内容与对象属性相冲突	403 Forbidden
InvalidObjectStorage	不合法的存储类型	403 Forbidden
RequestTimeTooSkewed	本地时间与服务器时间相差过大，超过 15 分钟	403 Forbidden
SignatureDoesNotMatch	客户端计算的签名与 COS 服务端计算的签名不一致	403 Forbidden
NoSuchBucket	指定的存储桶不存在	404 Not Found
NoSuchCopySource	复制对象源不存在	404 Not Found
NoSuchCORSConfiguration	指定的跨域资源共享设置不存在	404 Not Found
NoSuchKey	指定的对象键不存在	404 Not Found
NoSuchLifecycleConfiguration	指定的生命周期设置不存在	404 Not Found
NoSuchTagSet	指定的标签集合不存在	404 Not Found
NoSuchUpload	分块上传时指定的 UploadId 不存在	404 Not Found
NoSuchWebsiteConfiguration	静态网站配置不存在	404 Not Found
MethodNotAllowed	此资源不支持该 HTTP 方法	405 Method Not Allowed
RestoreNonArchiveObject	不允许对非归档对象进行回热	405 Method Not Allowed
BucketAlreadyExists	指定的存储桶已存在	409 Conflict
BucketAlreadyExistsDiffRegion	指定的存储桶已存在于其他地域	409 Conflict
BucketAlreadyOwnedByYou	指定的存储桶已存在	409 Conflict
BucketNotEmpty	存储桶不为空	409 Conflict
InvalidBucketState	存储桶状态与操作请求冲突，比如多版本管理与跨区域复制的冲突	409 Conflict
PathConflict	存在同名对象的毫秒级并发冲突	409 Conflict
RestoreAlreadyInProgress	该对象正在回热中	409 Conflict
MissingContentLength	Content-Length 请求头部缺失	411 Length Required
PreconditionFailed	前置条件匹配失败	412 Precondition
InvalidRange	请求的对象范围不合法	416 Requested Range Not Satisfiable
UnavailableForLegalReasons	因法律原因不可用	451 Unavailable For Legal Reasons

5XX 类型错误

错误码	描述	HTTP状态码
InternalErrror	服务端内部错误	500 Internal Server
NotImplemented	请求尚未实现	501 Not Implemented
ServiceUnavailable	服务暂不可用，请重试	503 Service Unavailable
SlowDown	请降低访问频率	503 Slow Down

请求签名

最近更新时间: 2022-10-25 15:37:02

> 注意：
>
> 1. 此文档仅适用于 COS XML 版本。
> 2. 此文档不适用于 POST Object 的 HTTP 请求。

使用对象存储服务 COS 时，可通过 RESTful API 对 COS 发起 HTTP 匿名请求或 HTTP 签名请求，对于签名请求，COS 服务器端将会进行对请求发起者的身份验证。

- 匿名请求：HTTP 请求不携带任何身份标识和鉴权信息，通过 RESTful API 进行 HTTP 请求操作。
- 签名请求：HTTP 请求时添加签名，COS 服务器端收到消息后，进行身份验证，验证成功则可接受并执行请求，否则将会返回错误信息并丢弃此请求。

对象存储 COS 基于密钥 HMAC（Hash Message Authentication Code）的自定义方案进行身份验证,支持HMAC-SHA1、HMAC-SHA256（兼容AWS S3）签名算法。

签名使用场景

在 COS 对象存储服务使用的场景中，对于需要对外发布类的数据，通常可将对象设置为公有读私有写。即所有人可查看，通过 ACL 策略指定账号可写入。此时，可将 ACL 策略与 API 请求签名相结合，对访问进行身份验证，并对操作进行权限和有效期的控制。

> 注意：
>
> 本文所描述的 API 请求签名，如果您使用 SDK 进行开发，则已包含在内。**仅在您希望通过原始 API 进行二次开发时，需要根据本文所描述步骤进行操作。**

在以上场景中，可对 API 请求进行多方面的安全防护：

1. **请求者身份验证。**通过访问者唯一 ID 和密钥确定请求者身份。
2. **防止传输数据篡改。**对数据签名并检验，保障传输内容完整性。
3. **防止签名被盗用。**对签名设置时效，避免签名盗用并重复使用。

准备工作

1. APPID、SecretId 和 SecretKey。 在访问管理控制台的API 密钥管理页面中获取。
2. 确定开发语言： 支持但不限于 Java、PHP、C#、C++、Node.js、Python，根据不同的开发语言，确定对应的 HMAC-SHA1、SHA1 和 UrlEncode 函数。 其中，HMAC-SHA1 和 SHA1 函数以 UTF-8 编码字符串为输入，以 16 进制小写字符串为输出。UrlEncode 基于 UTF-8 编码，此外对于 ASCII 范围内的可打印字符，下列特殊符号也应被编码：

字符	十进制	十六进制	字符	十进制	十六进制
(空格)	32	20	;	59	3B
!	33	21	<	60	3C
"	34	22	=	61	3D
#	35	23	>	62	3E
$	36	24	?	63	3F
%	37	25	@	64	40
&	38	26	[	91	5B
'	39	27	\	92	5C
(	40	28	]	93	5D
)	41	29	^	94	5E
*	42	2A	`	96	60
+	43	2B	{	123	7B
,	44	2C	|	124	7C

字符	十进制	十六进制	字符	十进制	十六进制
/	47	2F	}	125	7D
:	58	3A			

签名流程

生成 KeyTime

1. 获取当前时间对应的 Unix 时间戳 StartTimestamp，Unix 时间戳是从 UTC（协调世界时，或 GMT 格林威治时间）1970年1月1日0时0分0秒（北京时间 1970年1月1日8时0分0秒）起至现在的总秒数。
2. 根据上述时间戳和期望的签名有效时长算出签名过期时间对应的 Unix 时间戳 EndTimestamp。
3. 拼接签名有效时间，格式为 StartTimestamp;EndTimestamp ，即为 KeyTime。

示例： 1557902800;1557910000

生成 SignKey

使用 HMAC-SHA1 以 SecretKey 为密钥，以 KeyTime 为消息，计算消息摘要（哈希值），即为 SignKey。

示例： 36bcd76dbb8c9f066472fec403df8a34cab34c77

生成 UrlParamList 和 HttpParameters

1. 遍历 HTTP 请求参数，生成 key 到 value 的映射 Map 及 key 的列表 KeyList，其中 key 转换为小写形式，value 使用 UrlEncode 编码。 其中，没有 value 的参数认为 value 为空字符串，如请求路径为 /?acl ，则认为是 /?acl= 。
2. 对 KeyList 按照字典序排序。
3. 对 Map 和 KeyList 中的 key 使用 UrlEncode 编码并再次转换为小写形式。
4. 按照 KeyList 的顺序拼接 Map 中的每一个键值对，格式为 key1=value1&key2=value2&key3=value3 ，即为 HttpParameters。
5. 按照 KeyList 的顺序拼接 KeyList 中的每一项，格式为 key1;key2;key3 ，即为 UrlParamList。

示例：

- 示例一： 请求路径： /?prefix=example-folder%2F&delimiter=%2F&max-keys=10 UrlParamList: delimiter;max-keys;prefix HttpParameters: delimiter=%2F&max-keys=10&prefix=example-folder%2F
  > 注意：请求路径中的请求参数在实际发送请求时也会进行 UrlEncode，因此要注意不要重复执行 UrlEncode。
- 示例二： 请求路径： /exampleobject?acl UrlParamList: acl HttpParameters: acl=

生成 HeaderList 和 HttpHeaders

生成 HeaderList 和 HttpHeaders 与 生成 UrlParamList 和 HttpParameters 相同，其中用于生成的 HTTP 请求参数替换为 HTTP 请求头，生成的 HttpParameters 即为 HttpHeaders，生成的 UrlParamList 即为 HeaderList。

示例： 请求头：

```
Date: Thu, 16 May 2019 03:15:06 GMT
Host: <BucketName-APPID>.<Endpoint>
x-cos-acl: private
x-cos-grant-read: uin="100000000011"
```

计算得到 HeaderList 为 date;host;x-cos-acl;x-cos-grant-read ，HttpHeaders 为 date=Thu%2C%2016%20May%202019%2003%3A15%3A06%20GMT&host=examplebucket-1250000000.cos.ap-shanghai.myqcloud.com&x-cos-acl=private&x-cos-grant-read=uin%3D%22100000000011%22 。

生成 HttpString

根据 HTTP 方法、HTTP 请求路径、HttpParameters 和 HttpHeaders 生成 HttpString，格式为 HttpMethod\nUriPathname\nHttpParameters\nHttpHeaders\n 。 其中：HttpMethod 转换为小写，如 get 或 put ；UriPathname 为请求路径，如 / 或 /exampleobject ；\n 为换行符；如果其中有字符串为空，前后的换行符需要保留，如 get\n/exampleobject\n\n\n 。

生成 StringToSign

根据 KeyTime 和 HttpString 生成 StringToSign，格式为 sha1\nKeyTime\nSHA1(HttpString)\n 。 其中：SHA1(HttpString) 为使用 SHA1 对 HttpString 计算的消息摘要。

生成 Signature

使用 HMAC-SHA1 以 SignKey 为密钥，以 StringToSign 为消息，计算消息摘要，即为 Signature。

生成签名

根据 SecretId、KeyTime、HeaderList、UrlParamList 和 Signature 生成签名，格式为：

```
q-sign-algorithm=sha1
&q-ak=SecretId
&q-sign-time=KeyTime
&q-key-time=KeyTime
&q-header-list=HeaderList
&q-url-param-list=UrlParamList
&q-signature=Signature
```

> 注意：上述格式中的换行仅用于更好的阅读，实际格式并不包含换行。

签名使用

通过 RESTful API 对 COS 发起的 HTTP 签名请求，可以通过以下几种方式传递签名：

1. 通过标准的 HTTP Authorization 头，如 Authorization: q-sign-algorithm=sha1&q-ak=...&q-sign-time=1557989753;1557996953&...&q-signature=...
2. 作为 HTTP 请求参数，请注意 UrlEncode，如 /exampleobject?q-sign-algorithm=sha1&q-ak=...&q-sign-time=1557989753%3B1557996953&...&q-signature=...

> 说明：上述示例中使用 ... 省略了部分具体签名内容。

代码示例

伪代码

```
KeyTime = [Now];[Expires]
SignKey = HMAC-SHA1([SecretKey], KeyTime)
HttpString = [HttpMethod]\n[HttpURI]\n[HttpParameters]\n[HttpHeaders]\n
StringToSign = sha1\nKeyTime\nSHA1(HttpString)\n
Signature = HMAC-SHA1(SignKey, StringToSign)
```

消息摘要算法示例

不同语言如何调用 HMAC-SHA1 可以参考下面的示例：

PHP

```
$sha1HttpString = sha1('ExampleHttpString');

$signKey = hash_hmac('sha1', 'ExampleKeyTime', 'YourSecretKey');
```

Java

```
import org.apache.commons.codec.digest.DigestUtils;
import org.apache.commons.codec.digest.HmacUtils;

String sha1HttpString = DigestUtils.sha1Hex("ExampleHttpString");

String signKey = HmacUtils.hmacSha1Hex("YourSecretKey", "ExampleKeyTime");
```

Python

```
import hmac
import hashlib

sha1 = hashlib.sha1()
sha1_http_string = sha1.update('ExampleHttpString'.encode('utf-8')).hexdigest()

sign_key = hmac.new('YourSecretKey'.encode('utf-8'), 'ExampleKeyTime'.encode('utf-8'), hashlib.sha1).hexdigest()
```

Node.js

```
var crypto = require('crypto');

var sha1HttpString = crypto.createHash('sha1').update('ExampleHttpString').digest('hex');
var signKey = crypto.createHmac('sha1', 'YourSecretKey').update('ExampleKeyTime').digest('hex');
```

Go

```
import (
"crypto/hmac"
"crypto/sha1"
)

h := sha1.New()
h.Write([]byte("ExampleHttpString"))
sha1HttpString := h.Sum(nil)

var hashFunc = sha1.New
h = hmac.New(hashFunc, []byte("YourSecretKey"))
h.Write([]byte("ExampleKeyTime"))
signKey := h.Sum(nil)
```

实际案例

准备工作

登录访问管理控制台的API 密钥管理页面获取其 APPID、SecretId 和 SecretKey，举例如下：

APPID	SecretId	SecretKey
1250000000	AKIDQjz3ltompVjBni5LitkWHFlFpwkn9U5q	BQYIM75p8x0iWVFSIgqEKwFprpRSVHlz

上传对象

原始请求

```
PUT /exampleobject(%E8%85%BE%E8%AE%AF%E4%BA%91) HTTP/1.1
Date: Thu, 16 May 2019 06:45:51 GMT
Host: <BucketName-APPID>.<Endpoint>
Content-Type: text/plain
Content-Length: 13
Content-MD5: mQ/fVh815F3k6TAUm8m0eg==
x-cos-acl: private
x-cos-grant-read: uin="100000000011"

ObjectContent
```

中间变量

- KeyTime = `1557989151;1557996351`
- SignKey = `eb2519b498b02ac213cb1f3d1a3d27a3b3c9bc5f`
- UrlParamList = (empty string)
- HttpParameters = (empty string)
- HeaderList = `content-length;content-md5;content-type;date;host;x-cos-acl;x-cos-grant-read`
- HttpHeaders = `content-length=13&content-md5=mQ%2FfVh815F3k6TAUm8m0eg%3D%3D&content-type=text%2Fplain&date=Thu%2C%2016%20May%202019%2006%3A45%3A51%20GMT&host=examplebucket-1250000000.cos.ap-beijing.myqcloud.com&x-cos-acl=private&x-cos-grant-read=uin%3D%22100000000011%22`
- HttpString = `put\n/exampleobject(腾讯云金融专区)\n\ncontent-length=13&content-md5=mQ%2FfVh815F3k6TAUm8m0eg%3D%3D&content-type=text%2Fplain&date=Thu%2C%2016%20May%202019%2006%3A45%3A51%20GMT&host=examplebucket-1250000000.cos.ap-beijing.myqcloud.com&x-cos-acl=private&x-cos-grant-read=uin%3D%22100000000011%22\n`
- StringToSign = `sha1\n1557989151;1557996351\n8b2751e77f43a0995d6e9eb9477f4b685cca4172\n`
- Signature = `3b8851a11a569213c17ba8fa7dcf2abec6935172`

其中，(empty string) 代表长度为 0 的空字符串，\n 代表换行符。

签名后的请求

```
PUT /exampleobject(%E8%85%BE%E8%AE%AF%E4%BA%91) HTTP/1.1
Date: Thu, 16 May 2019 06:45:51 GMT
Host: <BucketName-APPID>.<Endpoint>
Content-Type: text/plain
Content-Length: 13
Content-MD5: mQ/fVh815F3k6TAUm8m0eg==
x-cos-acl: private
x-cos-grant-read: uin="100000000011"
Authorization: q-sign-algorithm=sha1&q-ak=AKIDQjz3ltompVjBni5LitkWHFlFpwkn9U5q&q-sign-time=1557989151;1557996351&q-key-time=1557989151;1557996351&q-header-list=content-length;content-md5;content-type;date;host;x-cos-acl;x-cos-grant-read&q-url-param-list=&q-signature=3b8851a11a569213c17ba8fa7dcf2abec6935172

ObjectContent
```

下载对象

原始请求

```
GET /exampleobject(%E8%85%BE%E8%AE%AF%E4%BA%91)?response-content-type=application%2Foctet-stream&response-cache-control=max-age%3D600 HTTP/1.1
Date: Thu, 16 May 2019 06:55:53 GMT
Host: <BucketName-APPID>.<Endpoint>
```

中间变量

- KeyTime = 1557989753;1557996953
- SignKey = 937914bf490e9e8c189836aad2052e4feeb35eaf
- UrlParamList = response-cache-control;response-content-type
- HttpParameters = response-cache-control=max-age%3D600&response-content-type=application%2Foctet-stream
- HeaderList = date;host
- HttpHeaders = date=Thu%2C%2016%20May%202019%2006%3A55%3A53%20GMT&host=examplebucket-1250000000.cos.ap-beijing.myqcloud.com
- HttpString = get\n/exampleobject(腾讯云金融专区)\nresponse-cache-control=max-age%3D600&response-content-type=application%2Foctet-stream\ndate=Thu%2C%2016%20May%202019%2006%3A55%3A53%20GMT&host=examplebucket-1250000000.cos.ap-beijing.myqcloud.com\n
- StringToSign = sha1\n1557989753;1557996953\n54ecfe22f59d3514fdc764b87a32d8133ea611e6\n
- Signature = 01681b8c9d798a678e43b685a9f1bba0f6c0e012

其中，\n 代表换行符。

签名后的请求

```
GET /exampleobject(%E8%85%BE%E8%AE%AF%E4%BA%91)?response-content-type=application%2Foctet-stream&response-cache-control=max-age%3D600 HTTP/1.1
Date: Thu, 16 May 2019 06:55:53 GMT
Host: <BucketName-APPID>.<Endpoint>
Authorization: q-sign-algorithm=sha1&q-ak=AKIDQjz3ltompVjBni5LitkWHFlFpwkn9U5q&q-sign-time=1557989753;1557996953&q-key-time=1557989753;1557996953&q-header-list=date;host&q-url-param-list=response-cache-control;response-content-type&q-signature=01681b8c9d798a678e43b685a9f1bba0f6c0e012
```

操作列表

最近更新时间: 2025-01-15 17:01:00

对象存储服务（COS）相关接口及说明如下：

请求签名

请求签名

Service 接口

操作名	API	操作描述
查询存储桶列表	GET Service	查询指定账号下所有存储桶列表

Bucket 接口

基本操作

操作名	API	操作描述
创建存储桶	PUT Bucket	在指定账号下创建一个存储桶
查询对象列表	GET Bucket（List Object）	查询存储桶下的部分或者全部对象
检索存储桶及其权限	HEAD Bucket	确认存储桶是否存在且是否有权限访问
删除存储桶	DELETE Bucket	删除指定账号下的空存储桶
查询对象版本	GET Bucket Object Versions	查询存储桶下的部分或者全部对象及其历史版本信息

访问控制（acl）

操作名	API	操作描述
设置存储桶ACL	PUT Bucket acl	设置指定存储桶访问权限控制列表
查询存储桶ACL	GET Bucket acl	查询存储桶的访问控制列表

跨域资源共享（cors）

操作名	API	操作描述
设置跨域配置	PUT Bucket cors	设置存储桶的跨域访问权限
查询跨域配置	GET Bucket cors	查询存储桶的跨域访问配置信息
删除跨域配置	DELETE Bucket cors	删除存储桶的跨域访问配置信息

生命周期（lifecycle）

操作名	API	操作描述
设置生命周期	PUT Bucket lifecycle	设置存储桶生命周期管理的配置
查询生命周期	GET Bucket lifecycle	查询存储桶生命周期管理的配置
删除生命周期	DELETE Bucket lifecycle	删除存储桶生命周期管理的配置

存储桶策略（policy）

操作名	API	操作描述

操作名	API	操作描述
设置存储桶策略	PUT Bucket policy	设置指定存储桶的权限策略
查询存储桶策略	GET Bucket policy	查询指定存储桶的权限策略
删除存储桶策略	DELETE Bucket policy	删除指定存储桶的权限策略

防盗链（referer）

操作名	API	操作描述
设置存储桶referer	PUT Bucket referer	设置存储桶 Referer 白名单或者黑名单
查询存储桶referer	GET Bucket referer	查询存储桶 Referer 白名单或者黑名单

标签（tagging）

操作名	API	操作描述
设置存储桶标签	PUT Bucket tagging	为已存在的存储桶设置标签
查询存储桶标签	GET Bucket tagging	查询指定存储桶下已有的存储桶标签
删除存储桶标签	DELETE Bucket tagging	删除指定的存储桶标签

静态网站（website）

操作名	API	操作描述
设置静态网站	PUT Bucket website	为存储桶配置静态网站
查询静态网站	GET Bucket website	查询与存储桶关联的静态网站配置信息
删除静态网站	DELETE Bucket website	删除存储桶中的静态网站配置

版本控制（versioning）

操作名	API	操作描述
设置版本控制	PUT Bucket versioning	启用或者暂停存储桶的版本控制功能
查询版本控制	GET Bucket versioning	查询存储桶的版本控制信息

跨地域复制（replication）

操作名	API	操作描述
设置跨地域复制	PUT Bucket replication	对已启用版本控制的存储桶配置跨地域复制规则
查询跨地域复制	GET Bucket replication	查询存储桶的跨地域复制配置信息
删除跨地域复制	DELETE Bucket replication	删除存储桶的跨地域复制配置信息

日志管理（logging）

操作名	API	操作描述
设置日志管理	PUT Bucket logging	为源存储桶开启日志记录，将源存储桶的访问日志保存到指定的目标存储桶中
查询日志管理	GET Bucket logging	获取源存储桶的日志配置信息

存储桶加密（encryption）接口

操作名	API	操作描述
设置存储桶加密	PUT Bucket encryption	设置指定存储桶下的默认加密配置
查询存储桶加密	GET Bucket encryption	查询指定存储桶下的默认加密配置

版权所有：腾讯云金融专区

操作名	API	操作描述
删除存储桶加密	DELETE Bucket encryption	删除指定存储桶下的默认加密配置

对象锁定（ObjectLock）接口

操作名	API	操作描述
设置对象锁定	PUT Bucket ObjectLockConfiguration	为已存在的存储桶设置对象锁定
查询对象锁定	GET Bucket ObjectLockConfiguration	查询已生效的对象锁定配置
查询对象锁定的到期日期	GET Object retention	查询对象锁定的到期日期

Object 接口

基本操作

操作名	API	操作描述
简单上传对象	PUT Object	上传一个对象至存储桶
设置对象复制	PUT Object - Copy	复制文件到目标路径
表单上传对象	POST Object	使用表单请求上传对象
下载对象	GET Object	下载一个对象至本地
查询对象元数据	HEAD Object	查询对象的元数据信息
删除单个对象	DELETE Object	在存储桶中删除指定对象
删除多个对象	DELETE Multiple Objects	在存储桶中批量删除对象
预请求跨域配置	OPTIONS Object	用预请求来确认是否可以发送真正的跨域请求
恢复归档对象	POST Object restore	将归档类型的对象取回访问

访问控制

操作名	API	操作描述
设置对象ACL	PUT Object acl	设置存储桶中某个对象的访问控制列表
查询对象ACL	GET Object acl	查询对象的访问控制列表

分块上传

操作名	API	操作描述
初始化分块上传	Initiate Multipart Upload	初始化分块上传任务
上传分块	Upload Part	分块上传文件
复制分块	Upload Part - Copy	将其他对象复制为一个分块
完成分块上传	Complete Multipart Upload	完成整个文件的分块上传
终止分块上传	Abort Multipart Upload	终止一个分块上传操作并删除已上传的块
查询分块上传	List Multipart Uploads	查询正在进行中的分块上传信息
查询已上传块	List Parts	查询特定分块上传操作中的已上传的块

版权所有：腾讯云金融专区

Service接口
查询存储桶列表

最近更新时间: 2025-01-15 17:01:00

功能描述

GET Service 接口是用来获取请求者名下的所有存储空间列表（Bucket list）。

请求

请求示例

```
GET / HTTP/1.1
Host: <Endpoint>
Date: GMT Date
Authorization: Auth String
```

> Authorization: Auth String (详情参见请求签名章节)
>
> Host：查询特定地域下的存储桶列表

请求头

公共头部

该请求操作的实现使用公共请求头，了解公共请求头详情请参见 公共请求头部 章节。

非公共头部

该请求操作无特殊的请求头部信息。

请求体

该请求的请求体为空。

响应

响应头

公共响应头

该响应使用公共响应头，了解公共响应头详情请参见 公共响应头部 章节。

特有响应头

该请求操作无特殊的响应头部信息。

响应体

查询成功，返回 application/xml 数据，包含 Bucket 中的对象信息。

```
<?xml version="1.0" encoding="UTF-8" ?>
<ListAllMyBucketsResult>
<Owner>
<ID>string</ID>
<DisplayName>string</DisplayName>
</Owner>
<Buckets>
<Bucket>
<Name>string</Name>
<Location>string</Location>
<CreateDate>string</CreateDate>
</Bucket>
<Bucket>
<Name>string</Name>
```

```
<Location>string</Location>
<CreateDate>string</CreateDate>
</Bucket>
</Buckets>
</ListAllMyBucketsResult>
```

具体的数据描述如下：

节点名称（关键字）	父节点	描述	类型	必选
ListAllMyBucketsResult	无	说明本次响应的所有信息	Container	是

Container 节点 ListAllMyBucketsResult 的内容：

节点名称（关键字）	父节点	描述	类型	必选
Owner	ListAllMyBucketsResult	说明 Bucket 持有者的信息	Container	是
Buckets	ListAllMyBucketsResult	说明本次响应的所有 Bucket 列表信息	Container	是

Container 节点 Owner 的内容：

节点名称（关键字）	父节点	描述	类型	必选
ID	ListAllMyBucketsResult.Owner	Bucket 所有者的 ID	string	是
DisplayName	ListAllMyBucketsResult.Owner	Bucket 所有者的名字信息	string	是

Container 节点 Buckets 的内容：

节点名称（关键字）	父节点	描述	类型	必选
Bucket	ListAllMyBucketsResult.Buckets	单个 Bucket 的信息	Container	是

Container 节点 Bucket 的内容：

节点名称（关键字）	父节点	描述	类型	必选
Name	ListAllMyBucketsResult.Buckets.Bucket	Bucket 的名称	string	是
Location	ListAllMyBucketsResult.Buckets.Bucket	Bucket 所在地域。枚举值参见可用地域文档，如：ap-beijing, ap-hongkong, eu-frankfurt 等	string	是
CreateDate	ListAllMyBucketsResult.Buckets.Bucket	Bucket 创建时间。ISO8601 格式，例如 2016-11-09T08:46:32.000Z	string	是

错误码

错误码	描述	HTTP 状态码
InvalidBucketName	Bucket 名称不合法	400 Bad Request
SignatureDoesNotMatch	提供的签名不符合规则，返回该错误码	403 Forbidden
NoSuchBucket	Bucket 不存在，返回该错误码	404 Not Found

实际案例

请求

```
GET / HTTP/1.1
Host: <Endpoint>
Date: Fri, 24 May 2019 11:59:51 GMT
Authorization: q-sign-algorithm=sha1&q-ak=AKID8A0fBVtYFrNm02oY1g1JQQF0c3JO****&q-sign-time=1558699191;1558706391&q-key-time=1558699191;1558706391&q-header-list=date;host&q-url-param-list=&q-signature=c3f55f4ce2800fb343cf85ff536a9185a0c1****
Connection: close
```

响应

```
HTTP/1.1 200 OK
Content-Type: application/xml
Content-Length: 495
Connection: close
Date: Fri, 24 May 2019 11:59:51 GMT
Server: tencent-cos
x-cos-request-id: NWNlN2RjYjdfZjhjODBiMDlfOWNlNF9hYzc2****

<ListAllMyBucketsResult>
<Owner>
<ID>qcs::cam::uin/100000000001:uin/100000000001</ID>
<DisplayName>100000000001</DisplayName>
</Owner>
<Buckets>
<Bucket>
<Name>examplebucket1-1250000000</Name>
<Location>ap-beijing</Location>
<CreationDate>2019-05-24T11:49:50Z</CreationDate>
</Bucket>
<Bucket>
<Name>examplebucket2-1250000000</Name>
<Location>ap-beijing</Location>
<CreationDate>2019-05-24T11:51:50Z</CreationDate>
</Bucket>
</Buckets>
</ListAllMyBucketsResult>
```

Bucket接口
基本操作
创建存储桶

最近更新时间: 2025-01-15 17:01:00

功能描述

PUT Bucket 接口请求可以在指定账号下创建一个 Bucket。该 API 接口不支持匿名请求，您需要使用带 Authorization 签名认证的请求才能创建新的 Bucket 。创建 Bucket 的用户默认成为 Bucket 的持有者。

细节分析

创建 Bucket 时，如果没有指定访问权限，则默认使用私有读写（private）权限。

请求

语法示例：

```
PUT / HTTP/1.1
Host: <BucketName-APPID>.<Endpoint>
Date: GMT Date
Authorization: Auth String
```

> **说明：**
>
> Authorization: Auth String (详细参见请求签名章节)

请求行

```
PUT / HTTP/1.1
```

该 API 接口接受 PUT 请求。

请求头

公共头部 该请求操作的实现使用公共请求头,了解公共请求头详细请参见 公共请求头部 章节。

非公共头部 该请求操作的实现可以用 PUT 请求中的 x-cos-acl 头来设置 Bucket 访问权限。目前有三种 Bucket 的访问权限：public-read-write，public-read 和 private。如果不设置，默认为 private 权限。也可以单独明确赋予用户读、写或读写权限。内容如下：

> 了解更多 acl 请求可详细请参见 Put Bucket ACL 文档。

名称	描述	类型	必选
x-cos-acl	定义 Object 的 acl 属性。有效值：private，public-read-write，public-read；默认值：private	String	否
x-cos-grant-read	赋予被授权者读的权限。格式：x-cos-grant-read: id=" ",id=" "； 只能给根账户授权，id="qcs::cam::uin/:uin/"	String	否
x-cos-grant-write	赋予被授权者写的权限。格式：x-cos-grant-write: id=" ",id=" "； 只能给根账户授权，id="qcs::cam::uin/:uin/"	String	否
x-cos-grant-full-control	赋予被授权者读写权限。格式：x-cos-grant-full-control: id=" ",id=" "； 只能给根账户授权，id="qcs::cam::uin/:uin/"	String	否

请求体

该请求的请求体为空。

响应

响应头

公共响应头

该响应使用公共响应头,了解公共响应头详细请参见 公共响应头部 章节。

特有响应头

该响应无特殊的响应头。

响应体

该响应体返回为空。

错误分析

以下描述此请求可能会发生的一些特殊的且常见的错误情况：

错误码	HTTP 状态码	描述
BucketAlreadyExists	409 Conflict	当请求创建的 Bucket 已经存在，并且请求创建的用户就是拥有者
InvalidBucketName	400 Bad Request	Bucket 的命名不规范 具体原因可参考 message 的描述
InvalidRequest	400 Bad Request	Bucket 的命名不规范 具体原因可参考 message 的描述

如果 Bucket 设置的 ACL 不正确，也会导致创建 Bucket 失败，同时会返回 “Failed to set access control authority for the bucket” 的错误信息。具体错误原因，可根据返回的错误码参考 Put Bucket ACL 相关的文档

获取更多关于 COS 的错误码的信息，或者产品所有的错误列表，请查看 错误码 文档。

实际案例

请求

```
PUT / HTTP/1.1
Host: <BucketName-APPID>.<Endpoint>
Date: Thu, 12 Jan 2016 19:12:22 GMT
Authorization: q-sign-algorithm=sha1&q-ak=AKIDxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx&q-sign-time=1484708728;32557604728&q-key-time=1484708728;32557604728&q-header-list=host&q-url-param-list=&q-signature=b394a86624cbcc705b11bc6fc505843c5e2dd9c9
```

响应

```
HTTP /1.1 200 OK
Content-Type: application/xml
Content-Length: 0
Connection: keep-alive
Date: Thu, 12 Jan 2016 19:12:22 GMT
Server: tencent-cos
x-cos-request-id: NTg3ZWRiODJfOWIxZjRlXzZmNDBfMTUz
```

查询对象列表

最近更新时间: 2025-01-15 17:01:00

功能描述

GET Bucket 请求等同于 List Object 请求，可以列出该 Bucket 下的部分或者全部 Object。该 API 的操作者需要对 Bucket 有 Read 权限。

请求

请求示例

```
GET / HTTP/1.1
Host: <BucketName-APPID>.<Endpoint>
Date: GMT Date
Authorization: Auth String
```

> **说明：**
>
> Authorization: Auth String（详情请参阅请求签名文档）。

请求头

公共头部

该请求操作的实现使用公共请求头，了解公共请求头详情请参阅 公共请求头部 文档。

非公共头部

该请求操作无特殊的请求头部信息。

请求参数

名称	类型	描述	必选
prefix	string	前缀匹配，用来规定返回的文件前缀地址	否
delimiter	string	定界符为一个符号，如果有 Prefix，则将 Prefix 到 delimiter 之间的相同路径归为一类，定义为 Common Prefix，然后列出所有 Common Prefix。如果没有 Prefix，则从路径起点开始	否
encoding-type	string	规定返回值的编码方式，可选值：url	否
marker	string	默认以 UTF-8 二进制顺序列出条目，所有列出条目从 marker 开始	否
max-keys	string	单次返回最大的条目数量，默认值为1000，最大为1000	否

请求体

该请求请求体为空。

响应

响应头

公共响应头

该响应包含公共响应头，了解公共响应头详情请参阅 公共响应头部 章节。

特有响应头

该响应无特殊的响应头。

响应体

查询成功，返回 application/xml 数据，包含 Bucket 中的对象信息。

```
<?xml version='1.0' encoding='utf-8' ?>
<ListBucketResult>
<Name>examplebucket-1250000000</Name>
<Encoding-Type>url</Encoding-Type>
<Prefix>ela</Prefix>
<Marker/>
<MaxKeys>1000</MaxKeys>
<Delimiter>/</Delimiter>
<IsTruncated>false</IsTruncated>
<NextMarker>exampleobject.txt</NextMarker>
<Contents>
<Key>photo</Key>
<LastModified>2017-06-23T12:33:26.000Z</LastModified>
<ETag>\"79f2a852fac7e826c9f4dbe037f8a63b\"</ETag>
<Size>10485760</Size>
<Owner>
<ID>1250000000</ID>
</Owner>
<StorageClass>STANDARD</StorageClass>
</Contents>
<CommonPrefixes>
<Prefix>example</Prefix>
</CommonPrefixes>
</ListBucketResult>
```

具体的数据描述如下：

节点名称（关键字）	父节点	描述	类型
ListBucketResult	无	保存 Get Bucket 请求结果的所有信息	Container

Container 节点 ListBucketResult 内容：

节点名称（关键字）	父节点	描述	类型
Name	ListBucketResult	说明 Bucket 的信息	string
Encoding-Type	ListBucketResult	编码格式	string
Prefix	ListBucketResult	前缀匹配，用来规定响应请求返回的文件前缀地址	string
Marker	ListBucketResult	默认以 UTF-8 二进制顺序列出条目，所有列出条目从 marker 开始	string
MaxKeys	ListBucketResult	单次响应请求内返回结果的最大的条目数量	string
Delimiter	ListBucketResult	定界符为一个符号，如果有 Prefix，则将 Prefix 到 delimiter 之间的相同路径归为一类，定义为 Common Prefix，然后列出所有 Common Prefix。如果没有 Prefix，则从路径起点开始	string
IsTruncated	ListBucketResult	响应请求条目是否被截断，布尔值：true，false	boolean
NextMarker	ListBucketResult	假如返回条目被截断，则返回 NextMarker 就是下一个条目的起点	string
Contents	ListBucketResult	元数据信息	Container
CommonPrefixes	ListBucketResult	将 Prefix 到 delimiter 之间的相同路径归为一类，定义为 Common Prefix	Container

Container 节点 Contents 内容：

节点名称（关键字）	父节点	描述	类型
Key	ListBucketResult.Contents	Object 的 Key	string
LastModified	ListBucketResult.Contents	说明 Object 最后被修改时间	string
ETag	ListBucketResult.Contents	文件的 MD-5 算法校验值	string
Size	ListBucketResult.Contents	说明文件大小，单位是 Byte	string
Owner	ListBucketResult.Contents	Bucket 持有者信息	Container

节点名称（关键字）	父节点	描述	类型
StorageClass	ListBucketResult.Contents	Object 的存储级别，枚举值：STANDARD，STANDARD_IA，ARCHIVE	string

Container 节点 Owner 内容：

节点名称（关键字）	父节点	描述	类型
ID	ListBucketResult.Contents.Owner	Bucket 的 AppID	string

Container 节点 CommonPrefixes 内容：

节点名称（关键字）	父节点	描述	类型
Prefix	ListBucketResult.CommonPrefixes	单条 Common 的前缀	string

错误码

该请求操作无特殊错误信息，常见的错误信息请参阅 错误码 文档。

实际案例

请求

```
GET / HTTP/1.1
Host: <BucketName-APPID>.<Endpoint>
Date: Wed, 18 Oct 2014 22:32:00 GMT
Authorization: q-sign-algorithm=sha1&q-ak=AKIDxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx&q-sign-time=1484213451;32557109451&q-key-time=1484213451;32557109451&q-header-list=host&q-url-param-list=&q-signature=0336a1fc8350c74b6c081d4dff8e7a2db9007dc
```

响应

```
HTTP/1.1 200 OK
Content-Type: application/xml
Content-Length: 1132
Connection: keep-alive
Vary: Accept-Encoding
Date: Wed, 18 Oct 2014 22:32:00 GMT
Server: tencent-cos
x-cos-request-id: NTg3NzRjY2VfYmRjMzVfMTc5M182MmIyNg==

<?xml version='1.0' encoding='utf-8' ?>
<ListBucketResult>
<Name>examplebucket-1250000000</Name>
<Encoding-Type>url</Encoding-Type>
<Prefix>ela</Prefix>
<Marker/>
<MaxKeys>1000</MaxKeys>
<Delimiter>/</Delimiter>
<IsTruncated>false</IsTruncated>
<NextMarker>exampleobject.txt</NextMarker>
<Contents>
<Key>photo</Key>
<LastModified>2017-06-23T12:33:26.000Z</LastModified>
<ETag>\"79f2a852fac7e826c9f4dbe037f8a63b\"</ETag>
<Size>10485760</Size>
<Owner>
<ID>1250000001</ID>
</Owner>
<StorageClass>STANDARD</StorageClass>
</Contents>
<Contents>
<Key>picture</Key>
<LastModified>2017-06-23T12:33:26.000Z</LastModified>
<ETag>\"3f9a5dbff88b25b769fa6304902b5d9d\"</ETag>
<Size>10485760</Size>
```

```
<Owner>
<ID>1250000002</ID>
</Owner>
<StorageClass>STANDARD</StorageClass>
</Contents>
<Contents>
<Key>file</Key>
<LastModified>2017-06-23T12:33:26.000Z</LastModified>
<ETag>\"39bfb88c11c65ed6424d2e1cd4db1826\"</ETag>
<Size>10485760</Size>
<Owner>
<ID>1250000003</ID>
</Owner>
<StorageClass>STANDARD</StorageClass>
</Contents>
<Contents>
<Key>world</Key>
<LastModified>2017-06-23T12:33:26.000Z</LastModified>
<ETag>\"fb31459ad10289ff49327fd91a3e1f6a\"</ETag>
<Size>4</Size>
<Owner>
<ID>1250000004</ID>
</Owner>
<StorageClass>STANDARD</StorageClass>
</Contents>
</ListBucketResult>
```

检索存储桶及其权限

最近更新时间: 2025-01-15 17:01:00

功能描述

HEAD Bucket 请求可以确认该 Bucket 是否存在，是否有权限访问。HEAD 的权限与 Read 一致。有以下几种情况：

- Bucket 存在，返回 HTTP 状态码为200。
- Bucket 无访问权限，返回 HTTP 状态码为403。
- Bucket 不存在，返回 HTTP 状态码为404。

> **注意：**
>
> 目前我们还没有公开获取 Bucket 属性的接口（即可以返回 acl 等信息）。

请求

请求示例

```
HEAD / HTTP/1.1
Host: <BucketName-APPID>.<Endpoint>
Date: GMT Date
Authorization: Auth String
```

> **说明：**
>
> Authorization：Auth String（详细参见请求签名章节）。

请求头

公共头部

该请求操作的实现使用公共请求头，了解公共请求头详细请参见 公共请求头部 章节。

非公共头部

该请求操作无特殊的请求头部信息。

请求体

该请求的请求体为空。

响应

响应头

公共响应头

该响应包含公共响应头，了解公共响应头详情请参见 公共响应头部 章节。

特有响应头

该响应无特殊的响应头。

响应体

该响应体为空。

实际案例

请求

```
HEAD / HTTP/1.1
Host: <BucketName-APPID>.<Endpoint>
```

```
Date: Thu, 27 Oct 2015 20:32:00 GMT
Authorization: q-sign-algorithm=sha1&q-ak=AKIDxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx&q-sign-time=1484640517;32557536517&q-key-time=1484640517;3
2557536517&q-header-list=host&q-url-param-list=&q-signature=7bedc2f84a0a3d29df85fe727d0c1e388c410376
```

响应

```
HTTP/1.1 200 OK
Content-Type: application/xml
Content-Length: 0
Date: Thu, 27 Oct 2015 20:32:00 GMT
x-cos-request-id: NTg3ZGQxNDNfNDUyMDRlXzUyOWNfMjY5
```

删除存储桶

最近更新时间: 2025-01-15 17:01:00

功能描述

Delete Bucket 接口请求可以在指定账号下删除 Bucket，删除之前要求 Bucket 内的内容为空，只有删除了 Bucket 内的信息，才能删除 Bucket 本身。

请求

语法示例：

```
DELETE / HTTP/1.1
Host: <BucketName-APPID>.<Endpoint>
Date: GMT Date
Authorization: Auth String
```

> **说明：**
>
> Authorization: Auth String (详细参见请求签名章节)。

请求行

```
DELETE / HTTP/1.1
```

该 API 接口接受 DELETE 请求。

请求头

公共头部

该请求操作的实现使用公共请求头,了解公共请求头详细请参见 公共请求头部 章节。

非公共头部

该请求操作无特殊的请求头部信息。

请求体

该请求的请求体为空。

响应

响应头

公共响应头

该响应使用公共响应头,了解公共响应头详细请参见 公共响应头部 章节。

特有响应头

该响应无特殊的响应头。

响应体

该响应体返回为空。

错误分析

以下描述此请求可能会发生的一些特殊的且常见的错误情况：

错误码	描述	HTTP状态码
BucketNotEmpty	409 Conflict	不能删除一个非空的 Bucket。
AccessDenied	403 Forbidden	删除 Bucket 同样需要携带签名，如果试图删除一个没有访问权限的 Bucket，就会返回该错误。

错误码	描述	HTTP状态码
NoSuchBucket	404 Not Found	如果删除一个不存在的 Bucket，就回返回该错误。

获取更多关于 COS 的错误码的信息，或者产品所有的错误列表，请查看 错误码 文档。

实际案例

请求

```
DELETE / HTTP/1.1
Host: <BucketName-APPID>.<Endpoint>
Date: Wed, 23 Oct 2016 21:32:00 GMT
Authorization: q-sign-algorithm=sha1&q-ak=AKIDxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx&q-sign-time=1484708950;32557604950&q-key-time=1484708950;3
2557604950&q-header-list=host&q-url-param-list=&q-signature=2b27b72ad2540ff2dde341dc7579a66ee8cb2afc
```

响应

```
HTTP/1.1 204 OK
Content-Type: application/xml
Content-Length: 0
Connection: keep-alive
Date: Wed, 23 Oct 2016 21:32:00 GMT
x-cos-request-id: NTg3ZWRjNjBfOTgxZjRlXzZhYjlfMTg0
```

查询对象版本

最近更新时间: 2025-01-15 17:01:00

功能描述

GET Bucket Object versions 接口用于拉取存储桶内的所有对象及其历史版本信息，您可以通过指定参数筛选出存储桶内部分对象及其历史版本信息。该 API 的请求者需要对存储桶有读取权限。

请求

请求示例

```
GET /?versions HTTP/1.1
Host: <BucketName-APPID>.<Endpoint>
Date: GMT Date
Authorization: Auth String
```

> **说明：**
>
> Authorization: Auth String（详情请参见 请求签名文档）。

请求参数

名称	描述	类型	是否必选
prefix	对象键匹配前缀，限定响应中只包含指定前缀的对象键	string	否
delimiter	一个字符的分隔符，用于对对象键进行分组。所有对象键中从 prefix 或从头（若未指定 prefix）到首个 delimiter 之间相同的部分将作为 CommonPrefixes 下的一个 Prefix 节点。被分组的对象键不再出现在后续对象列表中，具体场景和用法可参考下面的实际案例	string	否
encoding-type	规定返回值的编码方式，可选值：url，代表返回的对象键为 URL 编码（百分号编码）后的值，例如“腾讯云金融专区”将被编码为`%E8%85%BE%E8%AE%AF%E4%BA%91`	string	否
key-marker	起始对象键标记，从该标记之后（不含）按照 UTF-8 字典序返回对象版本条目	string	否
version-id-marker	起始版本 ID 标记，从该标记之后（不含）返回对象版本条目	string	否
max-keys	单次返回最大的条目数量，默认值为1000，最大为1000	integer	否

请求头

此接口仅使用公共请求头部，详情请参见 公共请求头部 文档。

请求体

此接口无请求体。

响应

响应头

此接口仅返回公共响应头部，详情请参见 公共响应头部 文档。

响应体

查询成功，返回 **application/xml** 数据，包含存储桶中的对象版本信息。不同场景下的响应体请参见下方的实际案例。

```
<ListVersionsResult>
<Name>string</Name>
<Prefix>string</Prefix>
<KeyMarker>string</KeyMarker>
<VersionIdMarker>string</VersionIdMarker>
<MaxKeys>integer</MaxKeys>
<IsTruncated>boolean</IsTruncated>
```

```
<Delimiter>string</Delimiter>
<CommonPrefixes>
<Prefix>string</Prefix>
</CommonPrefixes>
<CommonPrefixes>
<Prefix>string</Prefix>
</CommonPrefixes>
<Version>
<Key>string</Key>
<VersionId>string</VersionId>
<IsLatest>boolean</IsLatest>
<LastModified>date</LastModified>
<ETag>string</ETag>
<Size>integer</Size>
<StorageClass>Enum</StorageClass>
<Owner>
<ID>string</ID>
<DisplayName>string</DisplayName>
</Owner>
</Version>
<DeleteMarker>
<Key>string</Key>
<VersionId>string</VersionId>
<IsLatest>boolean</IsLatest>
<LastModified>date</LastModified>
<Owner>
<ID>string</ID>
<DisplayName>string</DisplayName>
</Owner>
</DeleteMarker>
<DeleteMarker>
<Key>string</Key>
<VersionId>string</VersionId>
<IsLatest>boolean</IsLatest>
<LastModified>date</LastModified>
<Owner>
<ID>string</ID>
<DisplayName>string</DisplayName>
</Owner>
</DeleteMarker>
<Version>
<Key>string</Key>
<VersionId>string</VersionId>
<IsLatest>boolean</IsLatest>
<LastModified>date</LastModified>
<ETag>string</ETag>
<Size>integer</Size>
<StorageClass>Enum</StorageClass>
<Owner>
<ID>string</ID>
<DisplayName>string</DisplayName>
</Owner>
</Version>
</ListVersionsResult>
```

具体的节点描述如下：

节点名称（关键字）	父节点	描述	类型
ListVersionsResult	无	保存 GET Bucket Object versions 结果的所有信息	Container

Container 节点 ListVersionsResult 的内容：

节点名称（关键字）	父节点	描述	类型
Name	ListVersionsResult	存储桶的名称，格式为``，例如`examplebucket-1250000000`	string
EncodingType	ListVersionsResult	编码格式，对应请求中的 encoding-type 参数，且仅当请求中指定了 encoding-type 参数才会返回该节点	string
Prefix	ListVersionsResult	对象键匹配前缀，对应请求中的 prefix 参数	string

节点名称（关键字）	父节点	描述	类型
KeyMarker	ListVersionsResult	起始对象键标记，从该标记之后（不含）按照 UTF-8 字典序返回对象版本条目，对应请求中的 key-marker 参数	string
VersionIdMarker	ListVersionsResult	起始版本 ID 标记，从该标记之后（不含）返回对象版本条目，对应请求中的 version-id-marker 参数	string
MaxKeys	ListVersionsResult	单次响应返回结果的最大条目数量，对应请求中的 max-keys 参数	integer
Delimiter	ListVersionsResult	分隔符，对应请求中的 delimiter 参数，且仅当请求中指定了 delimiter 参数才会返回该节点	string
IsTruncated	ListVersionsResult	响应条目是否被截断，布尔值，例如 true 或 false	boolean
NextKeyMarker	ListVersionsResult	仅当响应条目有截断（IsTruncated 为 true）才会返回该节点，该节点的值为当前响应条目中的最后一个对象键，当需要继续请求后续条目时，将该节点的值作为下一次请求的 key-marker 参数传入	string
NextVersionIdMarker	ListVersionsResult	仅当响应条目有截断（IsTruncated 为 true）才会返回该节点，该节点的值为当前响应条目中的最后一个对象的版本 ID，当需要继续请求后续条目时，将该节点的值作为下一次请求的 version-id-marker 参数传入	string
CommonPrefixes	ListVersionsResult	从 prefix 或从头（若未指定 prefix）到首个 delimiter 之间相同的部分，定义为 Common Prefix。仅当请求中指定了 delimiter 参数才有可能返回该节点	Container
Version	ListVersionsResult	对象版本条目	Container
DeleteMarker	ListVersionsResult	对象删除标记条目	Container

Container 节点 ListVersionsResult.CommonPrefixes 的内容：

节点名称（关键字）	父节点	描述	类型
Prefix	ListVersionsResult.CommonPrefixes	单条 Common Prefix 的前缀	string

Container 节点 ListVersionsResult.Version 的内容：

节点名称（关键字）	父节点	描述	类型
Key	ListVersionsResult.Version	对象键	string
VersionId	ListVersionsResult.Version	对象的版本 ID	string
IsLatest	ListVersionsResult.Version	当前版本是否为该对象的最新版本	boolean
LastModified	ListVersionsResult.Version	当前版本的最后修改时间，为 ISO8601 格式，例如2019-05-24T10:56:40Z	date
ETag	ListVersionsResult.Version	对象的实体标签（Entity Tag），是对象被创建时标识对象内容的信息标签，可用于检查对象的内容是否发生变化，例如"8e0b617ca298a564c3331da28dcb50df"。此头部并不一定返回对象的 MD5 值，而是根据对象上传和加密方式而有所不同	string
Size	ListVersionsResult.Version	对象大小，单位为 Byte	integer
StorageClass	ListVersionsResult.Version	对象存储类型。枚举值请参见 [存储类型] 文档，例如 STANDARD_IA，ARCHIVE	Enum
Owner	ListVersionsResult.Version	对象持有者信息	Container

Container 节点 ListVersionsResult.Version.Owner 的内容：

节点名称（关键字）	父节点	描述	类型
ID	ListVersionsResult.Version.Owner	对象持有者的 APPID	string
DisplayName	ListVersionsResult.Version.Owner	对象持有者的名字	string

Container 节点 ListVersionsResult.DeleteMarker 的内容：

节点名称（关键字）	父节点	描述	类型

节点名称（关键字）	父节点	描述	类型
Key	ListVersionsResult.DeleteMarker	对象键	string
VersionId	ListVersionsResult.DeleteMarker	对象的删除标记的版本 ID	string
IsLatest	ListVersionsResult.DeleteMarker	当前删除标记是否为该对象的最新版本	boolean
LastModified	ListVersionsResult.DeleteMarker	当前删除标记的删除时间，为 ISO8601 格式，例如2019-05-24T10:56:40Z	date
Owner	ListVersionsResult.DeleteMarker	对象持有者信息	Container

Container 节点 ListVersionsResult.DeleteMarker.Owner 的内容：

节点名称（关键字）	父节点	描述	类型
ID	ListVersionsResult.DeleteMarker.Owner	对象持有者的 APPID	string
DisplayName	ListVersionsResult.DeleteMarker.Owner	对象持有者的名字	string

错误码

此接口无特殊错误信息，全部错误信息请参见 错误码 文档。

实际案例

案例一：未启用版本控制

请求

```
GET /?versions HTTP/1.1
Host: <BucketName-APPID>.<Endpoint>
Date: Thu, 15 Aug 2019 12:03:09 GMT
Authorization: q-sign-algorithm=sha1&q-ak=AKID8A0fBVtYFrNm02oY1g1JQQF0c3JO****&q-sign-time=1565870589;1565877789&q-key-time=1565870589;1565877789&q-header-list=date;host&q-url-param-list=versions&q-signature=1d8fcb8522df7be9fa52d94cd79462f92eb3****
Connection: close
```

响应

```
HTTP/1.1 200 OK
Content-Type: application/xml
Content-Length: 1262
Connection: close
Date: Thu, 15 Aug 2019 12:03:10 GMT
Server: tencent-cos
x-cos-request-id: NWQ1NTQ5ZmVfYjliYjBiMDlfMmFhNGZfY2Jm****

<ListVersionsResult>
<Name>examplebucket-1250000000</Name>
<Prefix/>
<KeyMarker/>
<VersionIdMarker/>
<MaxKeys>1000</MaxKeys>
<IsTruncated>false</IsTruncated>
<Version>
<Key>example-object-1.jpg</Key>
<VersionId/>
<IsLatest>true</IsLatest>
<LastModified>2019-08-15T12:03:10.000Z</LastModified>
<ETag>&amp;quot;0f0cd12c48979d1bf3f95255a36cb861&amp;quot;</ETag>
<Size>20</Size>
<StorageClass>STANDARD</StorageClass>
<Owner>
<ID>1250000000</ID>
<DisplayName>1250000000</DisplayName>
</Owner>
</Version>
<Version>
```

```
<Key>example-object-2.jpg</Key>
<VersionId/>
<IsLatest>true</IsLatest>
<LastModified>2019-08-15T12:03:09.000Z</LastModified>
<ETag>&amp;quot;51370fc64b79d0d3c7c609635be1c41f&amp;quot;</ETag>
<Size>20</Size>
<StorageClass>STANDARD</StorageClass>
<Owner>
<ID>1250000000</ID>
<DisplayName>1250000000</DisplayName>
</Owner>
</Version>
<Version>
<Key>example-object-3.jpg</Key>
<VersionId/>
<IsLatest>true</IsLatest>
<LastModified>2019-08-15T12:03:08.000Z</LastModified>
<ETag>&amp;quot;b2f1d893c5fde000ee8ea6eca18ed81f&amp;quot;</ETag>
<Size>20</Size>
<StorageClass>STANDARD</StorageClass>
<Owner>
<ID>1250000000</ID>
<DisplayName>1250000000</DisplayName>
</Owner>
</Version>
</ListVersionsResult>
```

案例二：启用版本控制

请求

```
GET /?versions HTTP/1.1
Host: <BucketName-APPID>.<Endpoint>
Date: Thu, 15 Aug 2019 12:03:41 GMT
Authorization: q-sign-algorithm=sha1&q-ak=AKID8A0fBVtYFrNm02oY1g1JQQF0c3JO****&q-sign-time=1565870621;1565877821&q-key-time=1565870621;1565877821&q-header-list=date;host&q-url-param-list=versions&q-signature=36400914186e87b3e88cc8049a79da5e3d79****
Connection: close
```

响应

```
HTTP/1.1 200 OK
Content-Type: application/xml
Content-Length: 3477
Connection: close
Date: Thu, 15 Aug 2019 12:03:41 GMT
Server: tencent-cos
x-cos-request-id: NWQ1NTRhMWRfODhjMjJhMDlfMWNkOF8xZTRm****

<ListVersionsResult>
<Name>examplebucket-1250000000</Name>
<Prefix/>
<KeyMarker/>
<VersionIdMarker/>
<MaxKeys>1000</MaxKeys>
<IsTruncated>false</IsTruncated>
<Version>
<Key>example-object-1.jpg</Key>
<VersionId>null</VersionId>
<IsLatest>true</IsLatest>
<LastModified>2019-08-15T12:03:10.000Z</LastModified>
<ETag>&amp;quot;0f0cd12c48979d1bf3f95255a36cb861&amp;quot;</ETag>
<Size>20</Size>
<StorageClass>STANDARD</StorageClass>
<Owner>
<ID>1250000000</ID>
<DisplayName>1250000000</DisplayName>
</Owner>
</Version>
<DeleteMarker>
<Key>example-object-2.jpg</Key>
```

<VersionId>MTg0NDUxNzgyMDMwODg0NzI0Njc</VersionId>
<IsLatest>true</IsLatest>
<LastModified>2019-08-15T12:03:41.000Z</LastModified>
<Owner>
<ID>1250000000</ID>
<DisplayName>1250000000</DisplayName>
</Owner>
</DeleteMarker>
<Version>
<Key>example-object-2.jpg</Key>
<VersionId>null</VersionId>
<IsLatest>false</IsLatest>
<LastModified>2019-08-15T12:03:09.000Z</LastModified>
<ETag>"51370fc64b79d0d3c7c609635be1c41f"</ETag>
<Size>20</Size>
<StorageClass>STANDARD</StorageClass>
<Owner>
<ID>1250000000</ID>
<DisplayName>1250000000</DisplayName>
</Owner>
</Version>
<DeleteMarker>
<Key>example-object-3.jpg</Key>
<VersionId>MTg0NDUxNzgyMDMwODg0NzA1NzM</VersionId>
<IsLatest>true</IsLatest>
<LastModified>2019-08-15T12:03:41.000Z</LastModified>
<Owner>
<ID>1250000000</ID>
<DisplayName>1250000000</DisplayName>
</Owner>
</DeleteMarker>
<Version>
<Key>example-object-3.jpg</Key>
<VersionId>MTg0NDUxNzgyMDMwODk5NTUxNzU</VersionId>
<IsLatest>false</IsLatest>
<LastModified>2019-08-15T12:03:39.000Z</LastModified>
<ETag>"e5c7403f4ac3ace73477eb8b1fd183f7"</ETag>
<Size>30</Size>
<StorageClass>STANDARD</StorageClass>
<Owner>
<ID>1250000000</ID>
<DisplayName>1250000000</DisplayName>
</Owner>
</Version>
<Version>
<Key>example-object-3.jpg</Key>
<VersionId>null</VersionId>
<IsLatest>false</IsLatest>
<LastModified>2019-08-15T12:03:08.000Z</LastModified>
<ETag>"b2f1d893c5fde000ee8ea6eca18ed81f"</ETag>
<Size>20</Size>
<StorageClass>STANDARD</StorageClass>
<Owner>
<ID>1250000000</ID>
<DisplayName>1250000000</DisplayName>
</Owner>
</Version>
<Version>
<Key>example-object-4.jpg</Key>
<VersionId>MTg0NDUxNzgyMDMwODg4OTI0MDc</VersionId>
<IsLatest>true</IsLatest>
<LastModified>2019-08-15T12:03:41.000Z</LastModified>
<ETag>"c5c4d52f90ec328890953bbe4ae08230"</ETag>
<Size>30</Size>
<StorageClass>STANDARD</StorageClass>
<Owner>
<ID>1250000000</ID>
<DisplayName>1250000000</DisplayName>
</Owner>
</Version>
<Version>
<Key>example-object-4.jpg</Key>

```
<VersionId>MTg0NDUxNzgyMDMwODkyMTg2NjI</VersionId>
<IsLatest>false</IsLatest>
<LastModified>2019-08-15T12:03:40.000Z</LastModified>
<ETag>&amp;quot;e9ec8bcb980d2e4d8526c346eb3b2585&amp;quot;</ETag>
<Size>20</Size>
<StorageClass>STANDARD</StorageClass>
<Owner>
<ID>1250000000</ID>
<DisplayName>1250000000</DisplayName>
</Owner>
</Version>
<Version>
<Key>example-object-5.jpg</Key>
<VersionId>MTg0NDUxNzgyMDMwODk4NTkzMzM</VersionId>
<IsLatest>true</IsLatest>
<LastModified>2019-08-15T12:03:39.000Z</LastModified>
<ETag>&amp;quot;201669a14bdf051d8a9d6f9828d3f4c4&amp;quot;</ETag>
<Size>20</Size>
<StorageClass>STANDARD</StorageClass>
<Owner>
<ID>1250000000</ID>
<DisplayName>1250000000</DisplayName>
</Owner>
</Version>
</ListVersionsResult>
```

案例三：带 delimiter 参数（列出根目录下的对象和子目录）

请求

```
GET /?versions&delimiter=%2F HTTP/1.1
Host: <BucketName-APPID>.<Endpoint>
Date: Fri, 16 Aug 2019 10:45:53 GMT
Authorization: q-sign-algorithm=sha1&q-ak=AKID8A0fBVtYFrNm02oY1g1JQQF0c3JO****&q-sign-time=1565952353;1565959553&q-key-time=1565952353;1565959553&q-header-list=date;host&q-url-param-list=delimiter;versions&q-signature=c3130139bcac870247d1a070dbc8ee1c7ad5****
Connection: close
```

响应

```
HTTP/1.1 200 OK
Content-Type: application/xml
Content-Length: 2529
Connection: close
Date: Fri, 16 Aug 2019 10:45:53 GMT
Server: tencent-cos
x-cos-request-id: NWQ1Njg5NjFfNjFiMDJhMDlfMWE5ZV8yMDY4****

<ListVersionsResult>
<Name>examplebucket-1250000000</Name>
<Prefix/>
<KeyMarker/>
<VersionIdMarker/>
<MaxKeys>1000</MaxKeys>
<IsTruncated>false</IsTruncated>
<Delimiter>/</Delimiter>
<CommonPrefixes>
<Prefix>example-folder-1/</Prefix>
</CommonPrefixes>
<CommonPrefixes>
<Prefix>example-folder-2/</Prefix>
</CommonPrefixes>
<Version>
<Key>example-object-1.jpg</Key>
<VersionId>MTg0NDUxNzgxMjEzNTU3NTk1Mjg</VersionId>
<IsLatest>true</IsLatest>
<LastModified>2019-08-16T10:45:53.000Z</LastModified>
<ETag>&amp;quot;5d1143df07a17b23320d0da161e2819e&amp;quot;</ETag>
<Size>30</Size>
<StorageClass>STANDARD</StorageClass>
<Owner>
```

```
<ID>1250000000</ID>
<DisplayName>1250000000</DisplayName>
</Owner>
</Version>
<DeleteMarker>
<Key>example-object-1.jpg</Key>
<VersionId>MTg0NDUxNzgxMjEzNjE1OTcxMzM</VersionId>
<IsLatest>false</IsLatest>
<LastModified>2019-08-16T10:45:47.000Z</LastModified>
<Owner>
<ID>1250000000</ID>
<DisplayName>1250000000</DisplayName>
</Owner>
</DeleteMarker>
<Version>
<Key>example-object-1.jpg</Key>
<VersionId>MTg0NDUxNzgxMjEzNjYzNzU0MjE</VersionId>
<IsLatest>false</IsLatest>
<LastModified>2019-08-16T10:45:43.000Z</LastModified>
<ETag>&amp;quot;0f0cd12c48979d1bf3f95255a36cb861&amp;quot;</ETag>
<Size>20</Size>
<StorageClass>STANDARD</StorageClass>
<Owner>
<ID>1250000000</ID>
<DisplayName>1250000000</DisplayName>
</Owner>
</Version>
<DeleteMarker>
<Key>example-object-2.jpg</Key>
<VersionId>MTg0NDUxNzgxMjEzNTYzMDY3NzY</VersionId>
<IsLatest>true</IsLatest>
<LastModified>2019-08-16T10:45:53.000Z</LastModified>
<Owner>
<ID>1250000000</ID>
<DisplayName>1250000000</DisplayName>
</Owner>
</DeleteMarker>
<Version>
<Key>example-object-2.jpg</Key>
<VersionId>MTg0NDUxNzgxMjEzNjI5OTc5OTU</VersionId>
<IsLatest>false</IsLatest>
<LastModified>2019-08-16T10:45:46.000Z</LastModified>
<ETag>&amp;quot;574c289a7906c7d8fecef028216afeca&amp;quot;</ETag>
<Size>30</Size>
<StorageClass>STANDARD</StorageClass>
<Owner>
<ID>1250000000</ID>
<DisplayName>1250000000</DisplayName>
</Owner>
</Version>
<Version>
<Key>example-object-2.jpg</Key>
<VersionId>MTg0NDUxNzgxMjEzNjgyODc1OTY</VersionId>
<IsLatest>false</IsLatest>
<LastModified>2019-08-16T10:45:41.000Z</LastModified>
<ETag>&amp;quot;51370fc64b79d0d3c7c609635be1c41f&amp;quot;</ETag>
<Size>20</Size>
<StorageClass>STANDARD</StorageClass>
<Owner>
<ID>1250000000</ID>
<DisplayName>1250000000</DisplayName>
</Owner>
</Version>
</ListVersionsResult>
```

案例四：带 prefix 和 delimiter 参数（列出指定子目录下的对象和子目录）

请求

```
GET /?versions&prefix=example-folder-1%2F&delimiter=%2F HTTP/1.1
Host: <BucketName-APPID>.<Endpoint>
```

```
Date: Fri, 16 Aug 2019 10:45:53 GMT
Authorization: q-sign-algorithm=sha1&q-ak=AKID8A0fBVtYFrNm02oY1g1JQQF0c3JO****&q-sign-time=1565952353;1565959553&q-key-time=1565952353;1565959553&q-header-list=date;host&q-url-param-list=delimiter;prefix;versions&q-signature=6d7b99a4b379b5fefb8b903ee491bae63590****
Connection: close
```

响应

```
HTTP/1.1 200 OK
Content-Type: application/xml
Content-Length: 2682
Connection: close
Date: Fri, 16 Aug 2019 10:45:53 GMT
Server: tencent-cos
x-cos-request-id: NWQ1Njg5NjFfMzdiMDJhMDlfODA1NV8yMDI2****

<ListVersionsResult>
<Name>examplebucket-1250000000</Name>
<Prefix>example-folder-1/</Prefix>
<KeyMarker/>
<VersionIdMarker/>
<MaxKeys>1000</MaxKeys>
<IsTruncated>false</IsTruncated>
<Delimiter>/</Delimiter>
<CommonPrefixes>
<Prefix>example-folder-1/sub-folder-1/</Prefix>
</CommonPrefixes>
<CommonPrefixes>
<Prefix>example-folder-1/sub-folder-2/</Prefix>
</CommonPrefixes>
<Version>
<Key>example-folder-1/example-object-1.jpg</Key>
<VersionId>MTg0NDUxNzgxMjEzNTY0MzYyNzE</VersionId>
<IsLatest>true</IsLatest>
<LastModified>2019-08-16T10:45:53.000Z</LastModified>
<ETag>&amp;quot;1a54e134fda29e15d225cd226c4b49a3&amp;quot;</ETag>
<Size>47</Size>
<StorageClass>STANDARD</StorageClass>
<Owner>
<ID>1250000000</ID>
<DisplayName>1250000000</DisplayName>
</Owner>
</Version>
<DeleteMarker>
<Key>example-folder-1/example-object-1.jpg</Key>
<VersionId>MTg0NDUxNzgxMjEzNjE2MTE1NTI</VersionId>
<IsLatest>false</IsLatest>
<LastModified>2019-08-16T10:45:47.000Z</LastModified>
<Owner>
<ID>1250000000</ID>
<DisplayName>1250000000</DisplayName>
</Owner>
</DeleteMarker>
<Version>
<Key>example-folder-1/example-object-1.jpg</Key>
<VersionId>MTg0NDUxNzgxMjEzNjcwNjI2MTU</VersionId>
<IsLatest>false</IsLatest>
<LastModified>2019-08-16T10:45:42.000Z</LastModified>
<ETag>&amp;quot;f173c1199e3d3b53dd91223cae16fb42&amp;quot;</ETag>
<Size>37</Size>
<StorageClass>STANDARD</StorageClass>
<Owner>
<ID>1250000000</ID>
<DisplayName>1250000000</DisplayName>
</Owner>
</Version>
<DeleteMarker>
<Key>example-folder-1/example-object-2.jpg</Key>
<VersionId>MTg0NDUxNzgxMjEzNTYyOTQ1MDY</VersionId>
<IsLatest>true</IsLatest>
<LastModified>2019-08-16T10:45:53.000Z</LastModified>
<Owner>
```

```
<ID>1250000000</ID>
<DisplayName>1250000000</DisplayName>
</Owner>
</DeleteMarker>
<Version>
<Key>example-folder-1/example-object-2.jpg</Key>
<VersionId>MTg0NDUxNzgxMjEzNjI3NTcyNzU</VersionId>
<IsLatest>false</IsLatest>
<LastModified>2019-08-16T10:45:46.000Z</LastModified>
<ETag>&amp;quot;f43741c189c2142b257767bed5b4ce3a&amp;quot;</ETag>
<Size>47</Size>
<StorageClass>STANDARD</StorageClass>
<Owner>
<ID>1250000000</ID>
<DisplayName>1250000000</DisplayName>
</Owner>
</Version>
<Version>
<Key>example-folder-1/example-object-2.jpg</Key>
<VersionId>MTg0NDUxNzgxMjEzNjgwNjgwNDk</VersionId>
<IsLatest>false</IsLatest>
<LastModified>2019-08-16T10:45:41.000Z</LastModified>
<ETag>&amp;quot;c9d28698978bb6fef6c1ed1c439a17d3&amp;quot;</ETag>
<Size>37</Size>
<StorageClass>STANDARD</StorageClass>
<Owner>
<ID>1250000000</ID>
<DisplayName>1250000000</DisplayName>
</Owner>
</Version>
</ListVersionsResult>
```

案例五：需分页时获取第一页

请求

```
GET /?versions HTTP/1.1
Host: <BucketName-APPID>.<Endpoint>
Date: Fri, 23 Aug 2019 11:31:31 GMT
Authorization: q-sign-algorithm=sha1&q-ak=AKID8A0fBVtYFrNm02oY1g1JQQF0c3JO****&q-sign-time=1566559891;1566567091&q-key-time=15665598
91;1566567091&q-header-list=date;host&q-url-param-list=versions&q-signature=2b146233465c2164c60e0e0b2385f5386a61****
Connection: close
```

响应

```
HTTP/1.1 200 OK
Content-Type: application/xml
Content-Length: 372125
Connection: close
Date: Fri, 23 Aug 2019 11:31:32 GMT
Server: tencent-cos
x-cos-request-id: NWQ1ZmNlOTRfMTljMDJhMDlfNTg5OV8yZWYz****

<ListVersionsResult>
<Name>examplebucket-1250000000</Name>
<Prefix/>
<KeyMarker/>
<VersionIdMarker/>
<MaxKeys>1000</MaxKeys>
<IsTruncated>true</IsTruncated>
<NextKeyMarker>example-object-0135.jpg</NextKeyMarker>
<NextVersionIdMarker>MTg0NDUxNzc1MTM4MjM0ODYyNjQ</NextVersionIdMarker>
<Version>
<Key>example-object-0001.jpg</Key>
<VersionId>MTg0NDUxNzc1MTM4Mjg1NzI5Mjc</VersionId>
<IsLatest>true</IsLatest>
<LastModified>2019-08-23T11:31:20.000Z</LastModified>
<ETag>&amp;quot;3dd9c0ec8b6669abec786e52b64e0497&amp;quot;</ETag>
<Size>36</Size>
<StorageClass>STANDARD</StorageClass>
```

```
<Owner>
<ID>1250000000</ID>
<DisplayName>1250000000</DisplayName>
</Owner>
</Version>
<DeleteMarker>
<Key>example-object-0002.jpg</Key>
<VersionId>MTg0NDUxNzc1MTM4MjY5NDY2MzQ</VersionId>
<IsLatest>true</IsLatest>
<LastModified>2019-08-23T11:31:22.000Z</LastModified>
<Owner>
<ID>1250000000</ID>
<DisplayName>1250000000</DisplayName>
</Owner>
</DeleteMarker>
<Version>
<Key>example-object-0002.jpg</Key>
<VersionId>MTg0NDUxNzc1MTM4MjgzMDg3OTE</VersionId>
<IsLatest>false</IsLatest>
<LastModified>2019-08-23T11:31:21.000Z</LastModified>
<ETag>&amp;quot;626f60ee8f3eb987342554379d63259f&amp;quot;</ETag>
<Size>36</Size>
<StorageClass>STANDARD</StorageClass>
<Owner>
<ID>1250000000</ID>
<DisplayName>1250000000</DisplayName>
</Owner>
</Version>
...
<DeleteMarker>
<Key>example-object-0004.jpg</Key>
<VersionId>MTg0NDUxNzc1MTM4MjcwNjg2Mzc</VersionId>
<IsLatest>false</IsLatest>
<LastModified>2019-08-23T11:31:22.000Z</LastModified>
<Owner>
<ID>1250000000</ID>
<DisplayName>1250000000</DisplayName>
</Owner>
</DeleteMarker>
...
<DeleteMarker>
<Key>example-object-0135.jpg</Key>
<VersionId>MTg0NDUxNzc1MTM4MjA2NTk1MTc</VersionId>
<IsLatest>false</IsLatest>
<LastModified>2019-08-23T11:31:28.000Z</LastModified>
<Owner>
<ID>1250000000</ID>
<DisplayName>1250000000</DisplayName>
</Owner>
</DeleteMarker>
...
<Version>
<Key>example-object-0135.jpg</Key>
<VersionId>MTg0NDUxNzc1MTM4MjM0ODYyNjQ</VersionId>
<IsLatest>false</IsLatest>
<LastModified>2019-08-23T11:31:26.000Z</LastModified>
<ETag>&amp;quot;56d3a714c81ba76baa6a0004126a2718&amp;quot;</ETag>
<Size>36</Size>
<StorageClass>STANDARD</StorageClass>
<Owner>
<ID>1250000000</ID>
<DisplayName>1250000000</DisplayName>
</Owner>
</Version>
</ListVersionsResult>
```

案例六：需分页时获取后续页（使用 key-marker 和 version-id-marker 请求参数）

请求

```
GET /?versions&key-marker=example-object-0135.jpg&version-id-marker=MTg0NDUxNzc1MTM4MjM0ODYyNjQ HTTP/1.1
Host: <BucketName-APPID>.<Endpoint>
Date: Fri, 23 Aug 2019 11:32:56 GMT
Authorization: q-sign-algorithm=sha1&q-ak=AKID8A0fBVtYFrNm02oY1g1JQQF0c3JO****&q-sign-time=1566559976;1566567176&q-key-time=1566559976;1566567176&q-header-list=date;host&q-url-param-list=key-marker;version-id-marker;versions&q-signature=5b0787a354752f3161c75d014b75d9f2bd68****
Connection: close
```

响应

```
HTTP/1.1 200 OK
Content-Type: application/xml
Content-Length: 8358
Connection: close
Date: Fri, 23 Aug 2019 11:32:56 GMT
Server: tencent-cos
x-cos-request-id: NWQ1ZmNlZThfNjFiMDJhMDlfYTgwOF8yZjIw****

<ListVersionsResult>
<Name>examplebucket-1250000000</Name>
<Prefix/>
<KeyMarker>example-object-0135.jpg</KeyMarker>
<VersionIdMarker>MTg0NDUxNzc1MTM4MjM0ODYyNjQ</VersionIdMarker>
<MaxKeys>1000</MaxKeys>
<IsTruncated>false</IsTruncated>
<Version>
<Key>example-object-0135.jpg</Key>
<VersionId>MTg0NDUxNzc1MTM4MjQ4MDkyNjk</VersionId>
<IsLatest>false</IsLatest>
<LastModified>2019-08-23T11:31:24.000Z</LastModified>
<ETag>&amp;quot;5dd4f1fb98a2a6d74c8482f2856ece6b&amp;quot;</ETag>
<Size>36</Size>
<StorageClass>STANDARD</StorageClass>
<Owner>
<ID>1250000000</ID>
<DisplayName>1250000000</DisplayName>
</Owner>
</Version>
<Version>
<Key>example-object-0135.jpg</Key>
<VersionId>MTg0NDUxNzc1MTM4MjYwOTQ5MTk</VersionId>
<IsLatest>false</IsLatest>
<LastModified>2019-08-23T11:31:23.000Z</LastModified>
<ETag>&amp;quot;91e59eed612971f0e00ac483bf5c7329&amp;quot;</ETag>
<Size>36</Size>
<StorageClass>STANDARD</StorageClass>
<Owner>
<ID>1250000000</ID>
<DisplayName>1250000000</DisplayName>
</Owner>
</Version>
...
<DeleteMarker>
<Key>example-object-0136.jpg</Key>
<VersionId>MTg0NDUxNzc1MTM4MTg1MjA4MDA</VersionId>
<IsLatest>true</IsLatest>
<LastModified>2019-08-23T11:31:31.000Z</LastModified>
<Owner>
<ID>1250000000</ID>
<DisplayName>1250000000</DisplayName>
</Owner>
</DeleteMarker>
...
<DeleteMarker>
<Key>example-object-0136.jpg</Key>
<VersionId>MTg0NDUxNzc1MTM4MjA2OTA5MDg</VersionId>
<IsLatest>false</IsLatest>
<LastModified>2019-08-23T11:31:28.000Z</LastModified>
<Owner>
<ID>1250000000</ID>
```

```
<DisplayName>1250000000</DisplayName>
</Owner>
</DeleteMarker>
...
<DeleteMarker>
<Key>example-object-0137.jpg</Key>
<VersionId>MTg0NDUxNzc1MTM4MjA2OTQ0MzA</VersionId>
<IsLatest>false</IsLatest>
<LastModified>2019-08-23T11:31:28.000Z</LastModified>
<Owner>
<ID>1250000000</ID>
<DisplayName>1250000000</DisplayName>
</Owner>
</DeleteMarker>
...
<Version>
<Key>example-object-0137.jpg</Key>
<VersionId>MTg0NDUxNzc1MTM4Mjc1MjQ3OTY</VersionId>
<IsLatest>false</IsLatest>
<LastModified>2019-08-23T11:31:22.000Z</LastModified>
<ETag>&amp;quot;9022b9bf1b1503902d46cbe976c94eea&amp;quot;</ETag>
<Size>36</Size>
<StorageClass>STANDARD</StorageClass>
<Owner>
<ID>1250000000</ID>
<DisplayName>1250000000</DisplayName>
</Owner>
</Version>
</ListVersionsResult>
```

案例七：使用 key-marker 指定起始对象

请求

```
GET /?versions&key-marker=example-object-0135.jpg HTTP/1.1
Host: <BucketName-APPID>.<Endpoint>
Date: Fri, 23 Aug 2019 11:32:56 GMT
Authorization: q-sign-algorithm=sha1&q-ak=AKID8A0fBVtYFrNm02oY1g1JQQF0c3JO****&q-sign-time=1566559976;1566567176&q-key-time=1566559976;1566567176&q-header-list=date;host&q-url-param-list=key-marker;versions&q-signature=8b601589aae7ffdb2211694dde4ad73c9634****
Connection: close
```

响应

```
HTTP/1.1 200 OK
Content-Type: application/xml
Content-Length: 7111
Connection: close
Date: Fri, 23 Aug 2019 11:32:56 GMT
Server: tencent-cos
x-cos-request-id: NWQ1ZmNlZThfNjFiMDJhMDlfYTgwY18yZjRi****

<ListVersionsResult>
<Name>examplebucket-1250000000</Name>
<Prefix/>
<KeyMarker>example-object-0135.jpg</KeyMarker>
<VersionIdMarker/>
<MaxKeys>1000</MaxKeys>
<IsTruncated>false</IsTruncated>
<DeleteMarker>
<Key>example-object-0136.jpg</Key>
<VersionId>MTg0NDUxNzc1MTM4MTg1MjA4MDA</VersionId>
<IsLatest>true</IsLatest>
<LastModified>2019-08-23T11:31:31.000Z</LastModified>
<Owner>
<ID>1250000000</ID>
<DisplayName>1250000000</DisplayName>
</Owner>
</DeleteMarker>
<Version>
<Key>example-object-0136.jpg</Key>
```

```
<VersionId>MTg0NDUxNzc1MTM4MTg5NjQzNDQ</VersionId>
<IsLatest>false</IsLatest>
<LastModified>2019-08-23T11:31:30.000Z</LastModified>
<ETag>&amp;quot;20f730cd4cab72c0edcbf37c40f9dabe&amp;quot;</ETag>
<Size>36</Size>
<StorageClass>STANDARD</StorageClass>
<Owner>
<ID>1250000000</ID>
<DisplayName>1250000000</DisplayName>
</Owner>
</Version>
<Version>
<Key>example-object-0136.jpg</Key>
<VersionId>MTg0NDUxNzc1MTM4MTk3MDAwMDY</VersionId>
<IsLatest>false</IsLatest>
<LastModified>2019-08-23T11:31:29.000Z</LastModified>
<ETag>&amp;quot;339baa9b8d4450e71fc268aaa5fa250e&amp;quot;</ETag>
<Size>36</Size>
<StorageClass>STANDARD</StorageClass>
<Owner>
<ID>1250000000</ID>
<DisplayName>1250000000</DisplayName>
</Owner>
</Version>
<DeleteMarker>
<Key>example-object-0136.jpg</Key>
<VersionId>MTg0NDUxNzc1MTM4MjA2OTA5MDg</VersionId>
<IsLatest>false</IsLatest>
<LastModified>2019-08-23T11:31:28.000Z</LastModified>
<Owner>
<ID>1250000000</ID>
<DisplayName>1250000000</DisplayName>
</Owner>
</DeleteMarker>
...
<DeleteMarker>
<Key>example-object-0137.jpg</Key>
<VersionId>MTg0NDUxNzc1MTM4MjA2OTQ0MzA</VersionId>
<IsLatest>false</IsLatest>
<LastModified>2019-08-23T11:31:28.000Z</LastModified>
<Owner>
<ID>1250000000</ID>
<DisplayName>1250000000</DisplayName>
</Owner>
</DeleteMarker>
...
<Version>
<Key>example-object-0137.jpg</Key>
<VersionId>MTg0NDUxNzc1MTM4Mjc1MjQ3OTY</VersionId>
<IsLatest>false</IsLatest>
<LastModified>2019-08-23T11:31:22.000Z</LastModified>
<ETag>&amp;quot;9022b9bf1b1503902d46cbe976c94eea&amp;quot;</ETag>
<Size>36</Size>
<StorageClass>STANDARD</StorageClass>
<Owner>
<ID>1250000000</ID>
<DisplayName>1250000000</DisplayName>
</Owner>
</Version>
</ListVersionsResult>
```

访问控制（acl）
设置存储桶ACL

最近更新时间: 2025-01-15 17:01:00

功能描述

PUT Bucket acl 接口用来写入 Bucket 的 acl 表，您可以通过 Header："x-cos-acl"，"x-cos-grant-read"，"x-cos-grant-full-control" 传入 acl 信息，或者通过 Body 以 XML 格式传入 acl 信息。

> **说明：**
>
> - Header 和 Body 只能选择其中一种，否则响应返回会冲突。
> - PUT Bucket acl 是一个覆盖操作，传入新的 acl 将覆盖原有 acl。
> - 只有 Bucket 创建者才有权限操作。

细节分析

1. 既可以通过头部设置，也可以通过 xml body 设置，建议只使用一种方法。
2. 私有 Bucket 可以给某个文件夹设置为公开，那么该文件夹下的文件都为公开；但是把文件夹设置成私有后，在该文件夹中设置的公开属性，不会生效。

请求

请求示例

```
PUT /?acl HTTP/1.1
Host: <BucketName-APPID>.<Endpoint>
Date: GMT Date
Authorization: Auth String
```

> **说明：**
>
> Authorization: Auth String (详情请参阅请求签名文档)

请求头

公共头部 该请求操作的实现使用公共请求头，了解公共请求头详情请参阅 公共请求头部 文档。

非公共头部 该请求操作的实现可以用 PUT 请求中的 x-cos-acl 头来设置 Bucket 访问权限。目前 Bucket 有三种访问权限：public-read-write，public-read 和 private。如果不设置，默认为 private 权限，也可以单独明确赋予用户读、写或读写权限。内容如下：

名称	描述	类型	必选
x-cos-acl	定义 Bucket 的 acl 属性。有效值： private，public-read-write，public-read ；默认值：private	String	否
x-cos-grant-read	赋予被授权者读的权限。格式：x-cos-grant-read: id="[OwnerUin]"	String	否
x-cos-grant-write	赋予被授权者写的权限。格式：x-cos-grant-write: id="[OwnerUin]"	String	否
x-cos-grant-full-control	赋予被授权者所有的权限。格式：x-cos-grant-full-control: id="[OwnerUin]"	String	否

请求体

该请求操作的实现也可以在请求体中带特定请求参数来设置 Bucket 访问权限，但请求体带参数方式和请求头带 acl 子资源方式两者只能选一种。 带所有节点的示例：

```
<AccessControlPolicy>
<Owner>
<ID>qcs::cam::uin/100000000001:uin/100000000001</ID>
</Owner>
<AccessControlList>
<Grant>
<Grantee>
<ID>qcs::cam::uin/100000000001:uin/100000000001</ID>
</Grantee>
<Permission>FULL_CONTROL</Permission>
```

```
</Grant>
<Grant>
<Grantee>
<ID>qcs::cam::uin/100000000001:uin/100000000001</ID>
</Grantee>
<Permission>READ</Permission>
</Grant>
</AccessControlList>
</AccessControlPolicy>
```

具体的数据内容如下：

节点名称（关键字）	父节点	描述	类型	必选
AccessControlPolicy	无	保存 GET Bucket acl 结果的容器	Container	是

Container 节点 AccessControlPolicy 的内容：

节点名称（关键字）	父节点	描述	类型	必选
Owner	AccessControlPolicy	Bucket 持有者信息	Container	是
AccessControlList	AccessControlPolicy	被授权者信息与权限信息	Container	是

Container 节点 Owner 的内容：

节点名称（关键字）	父节点	描述	类型	必选
ID	AccessControlPolicy.Owner	Bucket 持有者的 ID， 格式：qcs::cam::uin/:uin/ ，这里必须是主帐号，所以 和 是同一个值	String	是

Container 节点 AccessControlList 的内容：

节点名称（关键字）	父节点	描述	类型	必选
Grant	AccessControlPolicy.AccessControlList	单个 Bucket 的授权信息，一个 AccessControlList 可以拥有100条 Grant	Container	是

Container 节点 Grant 的内容：

节点名称（关键字）	父节点	描述	类型	必选
Grantee	AccessControlPolicy.AccessControlList.Grant	被授权者资源信息。type 类型为 RootAccount；	Container	是
Permission	AccessControlPolicy.AccessControlList.Grant	指明授予被授权者的权限信息，枚举值：READ，WRITE，FULL_CONTROL	String	是

Container 节点 Grantee 的内容：

节点名称（关键字）	父节点	描述	类型	必选
ID	AccessControlPolicy.AccessControlList.Grant.Grantee	用户的 ID， 格式：qcs::cam::uin/:uin/ ，这里必须是主帐号，所以， 和 是同一个值，也可以用 anyone（指代所有类型用户）代替 uin/ 和 uin/	String	是

响应

响应头

公共响应头

该响应使用公共响应头，了解公共响应头详情请参阅 公共响应头部 文档。

特有响应头

该响应无特殊的响应头。

响应体

该响应体返回为空。

错误分析

以下描述此请求可能会发生的一些特殊的且常见的错误情况：

错误码	描述	HTTP 状态码
InvalidDigest	400 Bad Request	用户带的 Content-MD5 和 COS 计算 body 的 Content-MD5 不一致
MalformedXML	400 Bad Request	传入的 xml 格式有误，请跟 restful api 文档仔细比对
InvalidArgument	400 Bad Request	参数错误，具体可以参考错误信息

实际案例

请求

```
PUT /?acl HTTP/1.1
Host: <BucketName-APPID>.<Endpoint>
Date: Fri, 25 Feb 2017 04:10:22 GMT
Authorization: q-sign-algorithm=sha1&q-ak=AKIDxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx&q-sign-time=1484724784;32557620784&q-key-time=1484724784;3
2557620784&q-header-list=host&q-url-param-list=acl&q-signature=785d9075b8154119e6a075713c1b9e56ff0bddfc
Content-Length: 229
Content-Type: application/x-www-form-urlencoded

<AccessControlPolicy>
<Owner>
<ID>qcs::cam::uin/100000000001:uin/100000000001</ID>
</Owner>
<AccessControlList>
<Grant>
<Grantee>
<ID>qcs::cam::uin/100000000001:uin/100000000001</ID>
</Grantee>
<Permission>FULL_CONTROL</Permission>
</Grant>
<Grant>
<Grantee>
<ID>qcs::cam::uin/100000000001:uin/100000000001</ID>
</Grantee>
<Permission>READ</Permission>
</Grant>
</AccessControlList>
</AccessControlPolicy>
```

响应

```
HTTP/1.1 200 OK
Content-Type: application/xml
Content-Length: 0
Connection: keep-alive
Date: Fri, 25 Feb 2017 04:10:22 GMT
Server: tencent-cos
x-cos-request-id: NTg3ZjFjMmJfOWIxZjRlXzZmNDhfMjIw
```

查询存储桶ACL

最近更新时间: 2025-01-15 17:01:00

功能描述

GET Bucket acl 接口用来获取存储桶的访问权限控制列表。

请求

请求示例

```
GET /?acl HTTP/1.1
Host: <BucketName-APPID>.<Endpoint>
Date: GMT Date
Authorization: Auth String
```

> **说明：**
>
> Authorization: Auth String（详情请参阅请求签名章节）

请求头

公共头部

该请求操作的实现使用公共请求头，了解公共请求头详情请参阅 公共请求头部 章节。

非公共头部

该请求操作无特殊的请求头部信息。

请求体

该请求的请求体为空。

响应

响应头

公共响应头

该响应包含公共响应头，了解公共响应头详情请参阅 公共响应头部 章节。

特有响应头

该响应无特殊的响应头。

响应体

该响应体返回为 **application/xml** 数据，包含完整节点数据的内容展示如下：

```
<AccessControlPolicy>
<Owner>
<ID>qcs::cam::uin/1250000000:uin/1250000000</ID>
<DisplayName>qcs::cam::uin/1250000000:uin/1250000000</DisplayName>
</Owner>
<AccessControlList>
<Grant>
<Grantee xmlns:xsi=\"http://imgcache.finance.cloud.tencent.com:80www.w3.org/2001/XMLSchema-instance\" xsi:type=\"CanonicalUser\">
<ID>qcs::cam::uin/1250000000:uin/1250000000</ID>
<DisplayName>qcs::cam::uin/1250000000:uin/1250000000</DisplayName>
</Grantee>
<Permission>FULL_CONTROL</Permission>
</Grant>
</AccessControlList>
</AccessControlPolicy>
```

具体的数据内容如下：

节点名称（关键字）	父节点	描述	类型
AccessControlPolicy	无	保存 Get Bucket ACL 结果的容器	Container

Container 节点 AccessControlPolicy 的内容：

节点名称（关键字）	父节点	描述	类型
Owner	AccessControlPolicy	Bucket 持有者信息	Container
AccessControlList	AccessControlPolicy	被授权者信息与权限信息	Container

Container 节点 Owner 的内容：

节点名称（关键字）	父节点	描述	类型
ID	AccessControlPolicy.Owner	Bucket 持有者 ID， 格式：qcs::cam::uin/:uin/，这里必须是根帐号，所以 和 是同一个值	String
DisplayName	AccessControlPolicy.Owner	Bucket 持有者的名称	String

Container 节点 AccessControlList 的内容：

节点名称（关键字）	父节点	描述	类型
Grant	AccessControlPolicy.AccessControlList	单个 Bucket 的授权信息。一个 AccessControlList 可以拥有100条 Grant	Container

Container 节点 Grant 的内容：

节点名称（关键字）	父节点	描述	类型
Grantee	AccessControlPolicy.AccessControlList.Grant	说明被授权者的信息。如果type 为 RootAccount或 CanonicalUser，ID 中指定的是根帐号。	Container
Permission	AccessControlPolicy.AccessControlList.Grant	指明授予被授权者的权限信息，枚举值：READ，WRITE，FULL_CONTROL	String

Container 节点 Grantee 的内容：

节点名称（关键字）	父节点	描述	类型
ID	AccessControlPolicy.Owner	用户的 ID，必须是根帐号，格式为：qcs::cam::uin/:uin/ 或 qcs::cam::anyone:anyone（指代所有用户）	String
DisplayName	AccessControlPolicy.Owner	用户的名称	String

错误码

该请求操作无特殊错误信息，常见的错误信息请参见 错误码 章节。

实际案例

请求

```
GET /?acl HTTP/1.1
Host: <BucketName-APPID>.<Endpoint>
Date: Fri, 10 Mar 2016 09:45:46 GMT
Authorization: q-sign-algorithm=sha1&q-ak=AKIDxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx&q-sign-time=1484213027;32557109027&q-key-time=1484213027;32557109027&q-header-list=host&q-url-param-list=acl&q-signature=dcc1eb2022b79cb2a780bf062d3a40e120b40652
```

响应

```
HTTP/1.1 200 OK
Content-Type: application/xml
Content-Length: 266
Connection: keep-alive
Date: Fri, 10 Mar 2016 09:45:46 GMT
Server: tencent-cos
x-cos-request-id: NTg3NzRiMjVfYmRjMzVfMTViMl82ZGZmNw==
<AccessControlPolicy>
<Owner>
<ID>qcs::cam::uin/1250000000:uin/1250000000</ID>
<DisplayName>qcs::cam::uin/1250000000:uin/1250000000</DisplayName>
</Owner>
<AccessControlList>
<Grant>
<Grantee xmlns:xsi=\"http://imgcache.finance.cloud.tencent.com:80www.w3.org/2001/XMLSchema-instance\" xsi:type=\"CanonicalUser\">
<ID>qcs::cam::uin/1250000000:uin/1250000000</ID>
<DisplayName>qcs::cam::uin/1250000000:uin/1250000000</DisplayName>
</Grantee>
<Permission>FULL_CONTROL</Permission>
</Grant>
</AccessControlList>
</AccessControlPolicy>
```

跨域资源共享（cors）
设置跨域配置

最近更新时间: 2025-01-15 17:01:00

功能描述

PUT Bucket cors 接口用来请求设置 Bucket 的跨域资源共享权限，您可以通过传入 XML 格式的配置文件来实现配置，文件大小限制为 64 KB。默认情况下，Bucket 的持有者直接有权限使用该 API 接口，Bucket 持有者也可以将权限授予其他用户。

请求

请求示例：

```
PUT /?cors HTTP/1.1
Host: <BucketName-APPID>.<Endpoint>
Date: GMT Date
Content-Length: length
Content-Type: application/xml
Content-MD5: MD5
Authorization: Auth String

<XML file>
```

> **说明：**
>
> Authorization: Auth String (详细参见请求签名章节)

请求行

```
PUT /?cors HTTP/1.1
```

该 API 接口接受 `PUT` 请求。

请求头

公共头部

该请求操作的实现使用公共请求头，了解公共请求头详细请参见 公共请求头部 章节。

非公共头部

该请求操作无特殊的请求头部信息。

请求体

请求的请求体为跨域规则。

```
<?xml version="1.0" encoding="UTF-8" ?>
<CORSConfiguration>
<CORSRule>
<ID>string</ID>
<AllowedOrigin>string</AllowedOrigin>
<AllowedMethod>string</AllowedMethod>
<AllowedHeader>string</AllowedHeader>
<MaxAgeSeconds>0</MaxAgeSeconds>
<ExposeHeader>string</ExposeHeader>
</CORSRule>
</CORSConfiguration>
```

具体的数据描述如下：

节点名称（关键字）	父节点	描述	类型	必选
CORSConfiguration	无	说明跨域资源共享配置的所有信息，最多可以包含 100 条 CORSRule	Container	是

Container 节点 CORSConfiguration 的内容：

节点名称（关键字）	父节点	描述	类型	必选
CORSRule	CORSConfiguration	说明跨域资源共享配置的所有信息，最多可以包含 100 条 CORSRule	Container	是

Container 节点 CORSRule 的内容：

节点名称（关键字）	父节点	描述	类型	必选
ID	CORSConfiguration.CORSRule	配置规则的 ID，可选填	string	是
AllowedOrigin	CORSConfiguration.CORSRule	允许的访问来源，支持通配符 * 格式为：协议://域名[:端口]如：http://imgcache.finance.cloud.tencent.com:80www.qq.com	strings	是
AllowedMethod	CORSConfiguration.CORSRule	允许的 HTTP 操作，枚举值：GET，PUT，HEAD，POST，DELETE	strings	是
AllowedHeader	CORSConfiguration.CORSRule	在发送 OPTIONS 请求时告知服务端，接下来的请求可以使用哪些自定义的 HTTP 请求头部，支持通配符 *	strings	是
MaxAgeSeconds	CORSConfiguration.CORSRule	设置 OPTIONS 请求得到结果的有效期	integer	是
ExposeHeader	CORSConfiguration.CORSRule	设置浏览器可以接收到的来自服务器端的自定义头部信息	strings	是

响应

响应头

公共响应头

该响应使用公共响应头,了解公共响应头详细请参见 公共响应头部 章节。

特有响应头

该请求操作无特殊的响应头部信息。

响应体

该请求响应体为空。

错误码

错误码	描述	HTTP 状态码
SignatureDoesNotMatch	提供的签名不符合规则，返回该错误码	403 Forbidden
NoSuchBucket	如果试图添加的规则所在的 Bucket 不存在，返回该错误码	404 Not Found

实际案例

请求

```
PUT /?cors HTTP/1.1
Host: <BucketName-APPID>.<Endpoint>
Date: Fri, 10 Mar 2017 09:45:46 GMT
Authorization: q-sign-algorithm=sha1&q-ak=AKIDxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx&q-sign-time=1484814927;32557710927&q-key-time=1484814927;32557710927&q-header-list=host&q-url-param-list=cors&q-signature=8b9f05dabce2578f3a79d732386e7cbade9033e3
Content-Type: application/xml
Content-Length: 280

<CORSConfiguration>
<CORSRule>
<ID>1234</ID>
<AllowedOrigin>http: //www.qq.com</AllowedOrigin>
<AllowedMethod>PUT</AllowedMethod>
<AllowedHeader>x-cos-meta-test</AllowedHeader>
<MaxAgeSeconds>500</MaxAgeSeconds>
<ExposeHeader>x-cos-meta-test1</ExposeHeader>
```

```
</CORSRule>
</CORSConfiguration>
```

响应

```
HTTP/1.1 200 OK
Content-Type: application/xml
Content-Length: 0
Connection: keep-alive
Date: Fri, 10 Mar 2017 09:45:46 GMT
Server: tencent-cos
x-cos-request-id: NTg4MDdiZWRfOWExZjRlXzQ2OWVfZGY0
```

查询跨域配置

最近更新时间: 2025-01-15 17:01:00

功能描述

GET Bucket cors 接口实现 Bucket 持有者在 Bucket 上进行跨域资源共享的信息配置。（cors 是一个 W3C 标准，全称是"跨域资源共享"（Cross-origin resource sharing））。默认情况下，Bucket 的持有者直接有权限使用该 API 接口，Bucket 持有者也可以将权限授予其他用户。

请求

请求示例

```
GET /?cors HTTP/1.1
Host: <BucketName-APPID>.<Endpoint>
Date: GMT Date
Authorization: Auth String
```

> 说明：
>
> Authorization: Auth String（详细参见请求签名章节）。

请求头

公共头部

该请求操作的实现使用公共请求头，了解公共请求头详细请参见 公共请求头部 章节。

非公共头部

该请求操作无特殊的请求头部信息。

请求体

该请求的请求体为空。

响应

响应头

公共响应头

该响应包含公共响应头，了解公共响应头详情请参见 公共响应头部 章节。

特有响应头

该响应无特殊的响应头。

响应体

获取跨域资源共享的信息配置成功。

```
<?xml version="1.0" encoding="UTF-8" ?>
<CORSConfiguration>
<CORSRule>
<ID>bucketid</ID>
<AllowedOrigin>http://imgcache.finance.cloud.tencent.com:80www.qq.com</AllowedOrigin>
<AllowedMethod>PUT</AllowedMethod>
<AllowedHeader>x-cos-meta-test</AllowedHeader>
<ExposeHeader>x-cos-meta-test1</ExposeHeader>
<MaxAgeSeconds>500</MaxAgeSeconds>
</CORSRule>
</CORSConfiguration>
```

具体的数据描述如下：

节点名称（关键字）	父节点	描述	类型	必选
CORSConfiguration	无	说明跨域资源共享配置的所有信息，最多可以包含100条 CORSRule	Container	是

Container 节点 CORSConfiguration 的内容：

节点名称（关键字）	父节点	描述	类型	必选
CORSRule	CORSConfiguration	说明跨域资源共享配置的所有信息，最多可以包含100条 CORSRule	Container	是

Container 节点 CORSRule 的内容：

节点名称（关键字）	父节点	描述	类型	必选
ID	CORSConfiguration.CORSRule	配置规则的 ID，可选填	string	是
AllowedOrigin	CORSConfiguration.CORSRule	允许的访问来源，支持通配符`*`，格式为：协议://域名[:端口]如：`http://imgcache.finance.cloud.tencent.com:80www.qq.com`	strings	是
AllowedMethod	CORSConfiguration.CORSRule	允许的 HTTP 操作，枚举值：GET，PUT，HEAD，POST，DELETE	strings	是
AllowedHeader	CORSConfiguration.CORSRule	在发送 OPTIONS 请求时告知服务端，接下来的请求可以使用哪些自定义的 HTTP 请求头部，支持通配符 *	strings	是
MaxAgeSeconds	CORSConfiguration.CORSRule	设置 OPTIONS 请求得到结果的有效期	integer	是
ExposeHeader	CORSConfiguration.CORSRule	设置浏览器可以接收到的来自服务器端的自定义头部信息	strings	是

错误码

该请求操作无特殊错误信息，常见的错误信息请参见 错误码 章节。

实际案例

请求

```
GET /?cors HTTP/1.1
Host: <BucketName-APPID>.<Endpoint>
Date: Wed, 28 Oct 2016 21:32:00 GMT
Authorization: q-sign-algorithm=sha1&q-ak=AKIDxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx&q-sign-time=1484815944;32557711944&q-key-time=1484815944;32557711944&q-header-list=host&q-url-param-list=cors&q-signature=a2d28e1b9023d09f9277982775a4b3b705d0e23e
```

响应

```
HTTP/1.1 200 OK
Content-Type: application/xml
Content-Length: 345
Connection: keep-alive
Date: Wed, 28 Oct 2016 21:32:00 GMT
Server: tencent-cos
x-cos-request-id: NTg4MDdlNGZfNDYyMDRlXzM0YWFfZTBh

<CORSConfiguration>
<CORSRule>
<ID>bucketid</ID>
<AllowedOrigin>http://imgcache.finance.cloud.tencent.com:80www.qq.com</AllowedOrigin>
<AllowedMethod>PUT</AllowedMethod>
<AllowedHeader>x-cos-meta-test</AllowedHeader>
<ExposeHeader>x-cos-meta-test1</ExposeHeader>
<MaxAgeSeconds>500</MaxAgeSeconds>
</CORSRule>
</CORSConfiguration>
```

删除跨域配置

最近更新时间: 2025-01-15 17:01:00

功能描述

Delete Bucket CORS 接口请求实现删除跨域访问配置信息。

请求

语法示例：

```
DELETE /?cors HTTP/1.1
Host: <BucketName-APPID>.<Endpoint>
Date: GMT Date
Authorization: Auth String
```

> **说明：**
>
> Authorization: Auth String (详细参见请求签名章节)。

请求行

```
DELETE /?cors HTTP/1.1
```

该 API 接口接受 DELETE 请求。

请求头

公共头部

该请求操作的实现使用公共请求头,了解公共请求头详细请参见 公共请求头部 章节。

非公共头部

该请求操作无特殊的请求头部信息。

请求体

该请求的请求体为空。

响应

响应头

公共响应头

该响应使用公共响应头,了解公共响应头详细请参见 公共响应头部 章节。

特有响应头

该响应无特殊的响应头。

响应体

该响应体返回为空。

实际案例

请求

```
DELETE /?cors HTTP/1.1
Host: <BucketName-APPID>.<Endpoint>
Date: Tue, 23 Oct 2016 21:32:00 GMT
```

```
Authorization: q-sign-algorithm=sha1&q-ak=AKIDxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx&q-sign-time=1484816036;32557712036&q-key-time=1484816036;32557712036&q-header-list=host&q-url-param-list=cors&q-signature=e92eecbf0022fe7e5fd39b2c500b22da062be50a
```

响应

```
HTTP/1.1 204 No Content
Content-Type: application/xml
Content-Length: 405
Connection: keep-alive
Date: Tue, 23 Oct 2016 21:32:00 GMT
x-cos-request-id: NTg4MDdlYWNfOTgxZjRlXzZhYTlfZjAz
x-cos-trace-id: OGVmYzZiMmQzYjA2OWNhODk0NTRkMTBiOWVmMDAxODczNTBmNjMwZmQ0MTZkMjg0NjlkNTYyNmY4ZTRkZTk0N2M2MTdkZGZlMGNhOWQyYjk3MWNmNWNkYzFhMjQzNzRiZTE1NjgzNzFhOGI5M2EwZDMyNGM4Y2ZmMzhiNTllMjk=
```

生命周期（lifecycle）
设置生命周期

最近更新时间: 2025-01-15 17:01:00

功能描述

COS 支持用户以生命周期配置的方式来管理 Bucket 中 Object 的生命周期。生命周期配置包含一个或多个将应用于一组对象规则的规则集 (其中每个规则为 COS 定义一个操作)。 这些操作分为以下两种：

- **转换操作：**定义对象转换为另一个存储类的时间。例如，您可以选择在对象创建30天后将其转换为低频存储（STANDARD_IA，适用于不常访问）存储类别。同时也支持将数据沉降到归档存储（Archive，成本更低，目前支持国内地域）。具体参数查看请求示例说明中 Transition 项。
- **过期操作：**指定 Object 的过期时间。COS 将会自动为用户删除过期的 Object。

细节分析

PUT Bucket lifecycle 用于为 Bucket 创建一个新的生命周期配置。如果该 Bucket 已配置生命周期，使用该接口创建新的配置的同时则会覆盖原有的配置。

请求

请求示例

```
PUT /?lifecycle HTTP/1.1
Host: <BucketName-APPID>.cos.<Region>.myqcloud.com
Content-Length: length
Date: GMT Date
Authorization: Auth String
Content-MD5: MD5
```

> 说明：
>
> Authorization: Auth String（详情请参见请求签名文档）

请求头

公共头部

该请求操作的实现使用公共请求头，了解公共请求头详情请参见 公共请求头部 文档。

非公共头部

必选头部 该请求操作的实现使用如下必选头部：

名称	描述	类型	必选
Content-MD5	RFC 1864 中定义的经过 **Base64** 编码的 128-bit 内容 MD5 校验值。此头部用来校验文件内容是否发生变化。	String	是

请求体

该 API 接口请求的请求体具体节点内容为：

```
<LifecycleConfiguration>
<Rule>
<ID></ID>
<Filter>
<Prefix></Prefix>
</Filter>
<Status></Status>
<Transition>
<Days></Days>
<StorageClass></StorageClass>
</Transition>
<NoncurrentVersionExpiration>
<NoncurrentDays></NoncurrentDays>
</NoncurrentVersionExpiration>
</Rule>
```

```
<Rule>
<ID></ID>
<Filter>
<Prefix></Prefix>
</Filter>
<Status></Status>
<Transition>
<Days></Days>
<StorageClass></StorageClass>
</Transition>
<NoncurrentVersionTransition>
<NoncurrentDays></NoncurrentDays>
<StorageClass></StorageClass>
</NoncurrentVersionTransition>
</Rule>
<Rule>
<ID></ID>
<Filter>
<Prefix></Prefix>
</Filter>
<Status></Status>
<Expiration>
<ExpiredObjectDeleteMarker></ExpiredObjectDeleteMarker>
</Expiration>
<NoncurrentVersionExpiration>
<NoncurrentDays></NoncurrentDays>
</NoncurrentVersionExpiration>
</Rule>
</LifecycleConfiguration>
```

具体内容描述如下：

节点名称（关键字）	父节点	描述	类型	必选
LifecycleConfiguration	无	生命周期配置	Container	是
Rule	LifecycleConfiguration	规则描述	Container	是
Filter	LifecycleConfiguration.Rule	Filter 用于描述规则影响的 Object 集合	Container	是
Status	LifecycleConfiguration.Rule	指明规则是否启用，枚举值：Enabled，Disabled	Container	是
ID	LifecycleConfiguration.Rule	用于唯一地标识规则，长度不能超过255个字符	String	否
And	LifecycleConfiguration.Rule.Filter	用于组合 Prefix	Container	否
Prefix	LifecycleConfiguration.Rule.Filter 或 LifecycleConfiguration.Rule.Filter.And	指定规则所适用的前缀。匹配前缀的对象受该规则影响，Prefix 最多只能有一个	Container	否
Expiration	LifecycleConfiguration.Rule	规则过期属性	Container	否
Transition	LifecycleConfiguration.Rule	规则转换属性，对象何时转换为 Standard_IA 或 Archive	Container	否
Days	LifecycleConfiguration.Rule.Transition 或 Expiration	指明规则对应的动作在对象最后的修改日期过后多少天操作，如果是 Transition，该字段有效值是非负整数；如果是 Expiration，该字段有效值为正整数，最大支持3650天	Integer	否
Date	LifecycleConfiguration.Rule.Transition 或 Expiration	指明规则对应的动作在何时操作	String	否
ExpiredObjectDeleteMarker	LifecycleConfiguration.Rule.Expiration	删除过期对象删除标记，枚举值 true，false	String	否

节点名称（关键字）	父节点	描述	类型	必选
AbortIncompleteMultipartUpload	LifecycleConfiguration.Rule	设置允许分片上传保持运行的最长时间	Container	否
DaysAfterInitiation	LifecycleConfiguration.Rule.AbortIncompleteMultipartUpload	指明分片上传开始后多少天内必须完成上传	Integer	是
NoncurrentVersionExpiration	LifecycleConfiguration.Rule	指明非当前版本对象何时过期	Container	否
NoncurrentVersionTransition	LifecycleConfiguration.Rule	指明非当前版本对象何时转换为 STANDARD_IA 或 ARCHIVE	Container	否
NoncurrentDays	LifecycleConfiguration.Rule.NoncurrentVersionExpiration 或 NoncurrentVersionTransition	指明规则对应的动作在对象变成非当前版本多少天后执行，如果是 Transition，该字段有效值是非负整数；如果是Expiration，该字段有效值为正整数，最大支持3650天	Integer	否
StorageClass	LifecycleConfiguration.Rule.Transition 或 NoncurrentVersionTransition	指定 Object 转储到的目标存储类型，枚举值：STANDARD_IA, ARCHIVE	String	是

响应

响应头

公共响应头

该响应使用公共响应头，了解公共响应头详情请参见 公共响应头部 文档。

特有响应头

该响应无特殊的响应头。

响应体

该响应体返回为空。

错误码

以下描述此请求可能会发生的一些特殊的且常见的错误情况。具体的错误原因可参考返回的 message 进行排查。获取更多关于 COS 的错误码的信息，或者产品所有的错误列表，请参见 错误码 文档。

错误码	HTTP 状态码	描述
NoSuchBucket	404 Not Found	当访问的 Bucket 不存在
MalformedXML	400 Bad Request	XML 格式不合法，请跟 restful api 文档仔细比对
InvalidRequest	400 Bad Request	请求不合法，如果错误描述中显示"Conflict lifecycle rule"，那么表示 xml 数据中的多条 rule 有相互冲突的部分。
InvalidArgument	400 Bad Request	请求参数不合法，如果错误描述中显示"Rule ID must be unique. Found same ID for more than one rule"， 那么表示有多个 Rule 的 ID 字段相同。

实际案例

请求

```
PUT /?lifecycle HTTP/1.1
Host:examplebucket-1250000000.cos.ap-beijing.myqcloud.com
Date: Wed, 16 Aug 2017 11:59:33 GMT
Authorization:q-sign-algorithm=sha1&q-ak=AKIDxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx&q-sign-time=1502855771;1502935771&q-key-time=1502855771;1502935771&q-header-list=content-md5;host&q-url-param-list=lifecycle&q-signature=f3aa2c708cfd8d4d36d658de56973c9cf1c24654
```

```
Content-MD5: LcNUuow8OSZMrEDnvndw1Q==
Content-Length: 348
Content-Type: application/x-www-form-urlencoded

<LifecycleConfiguration>
<Rule>
<ID>id1</ID>
<Filter>
<Prefix>documents/</Prefix>
</Filter>
<Status>Enabled</Status>
<Transition>
<Days>100</Days>
<StorageClass>ARCHIVE</StorageClass>
</Transition>
</Rule>
<Rule>
<ID>id2</ID>
<Filter>
<Prefix>logs/</Prefix>
</Filter>
<Status>Enabled</Status>
<Expiration>
<Days>10</Days>
</Expiration>
</Rule>
</LifecycleConfiguration>
```

响应

```
HTTP/1.1 200 OK
Content-Type: application/xml
Content-Length: 0
Date: Wed, 16 Aug 2017 11:59:33 GMT
Server: tencent-cos
x-cos-request-id: NTk5NDMzYTRfMjQ4OGY3Xzc3NGRfMWY=
```

查询生命周期

最近更新时间: 2025-01-15 17:01:00

功能描述

GET Bucket lifecycle 用于查询 Bucket 的生命周期配置。如果该 Bucket 没有配置生命周期规则会返回 NoSuchLifecycleConfiguration。

请求

请求示例

```
GET /?lifecycle HTTP/1.1
Host: <BucketName-APPID>.cos.<Region>.myqcloud.com
Date: GMT Date
Authorization: Auth String
```

> **说明：**
>
> Authorization：Auth String（详情请参见请求签名文档）。

请求头

公共头部

该请求操作的实现使用公共请求头，了解公共请求头详情请参见 公共请求头部 文档。

非公共头部

该请求操作无特殊的请求头部信息。

请求体

该请求的请求体为空。

响应

响应头

公共响应头

该响应包含公共响应头，了解公共响应头详情请参见 公共响应头部 文档。

特有响应头

该响应无特殊的响应头。

响应体

响应体中各个元素的内容及含义与 PUT Buket lifecycle 时的请求体一致。详情请参见 设置生命周期 文档中的请求体节点描述内容。

错误码

该请求操作无特殊错误信息，常见的错误信息请参见 错误码 文档。

实际案例

请求

```
GET /?lifecycle HTTP/1.1
Host: examplebucket-1250000000.cos.ap-beijing.myqcloud.com
Date: Wed, 16 Aug 2017 12:23:54 GMT
Authorization:q-sign-algorithm=sha1&q-ak=AKIDxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx&q-sign-time=1502857357;1502937357&q-key-time=1502857357;1502937357&q-header-list=host&q-url-param-list=lifecycle&q-signature=da155cda3461bee7422ee95367ac8013ef847e02
```

响应

```
HTTP/1.1 200 OK
Content-Type: application/xml
Content-Length: 312
Date: Wed, 16 Aug 2017 12:23:54 GMT
Server: tencent-cos
x-cos-request-id: NTk5NDM5NWFfMjQ4OGY3Xzc3NGRfMjA=

<LifecycleConfiguration>
<Rule>
<ID>id1</ID>
<Filter>
<Prefix>documents/</Prefix>
</Filter>
<Status>Enabled</Status>
<Transition>
<Days>100</Days>
<StorageClass>STANDARD_IA</StorageClass>
</Transition>
</Rule>
<Rule>
<ID>id2</ID>
<Filter>
<Prefix>logs/</Prefix>
</Filter>
<Status>Enabled</Status>
<Expiration>
<Days>10</Days>
</Expiration>
</Rule>
</LifecycleConfiguration>
```

删除生命周期

最近更新时间: 2025-01-15 17:01:00

功能描述

DELETE Bucket lifecycle 用于删除 Bucket 的生命周期配置。如果该 Bucket 未配置生命周期规则，将返回 NoSuchLifecycleConfiguration。

请求

请求示例

```
DELETE /?lifecycle HTTP/1.1
Host: <BucketName-APPID>.cos.<Region>.myqcloud.com
Date: GMT Date
Authorization: Auth String
```

> **说明：**
>
> Authorization: Auth String ，详情信息请参见请求签名文档。

请求头

公共头部

该请求操作的实现使用公共请求头，了解公共请求头详情请参见 公共请求头部 文档。

非公共头部

该请求操作无特殊的请求头部信息。

请求体

该请求的请求体为空。

响应

响应头

公共响应头

该响应包含公共响应头，了解公共响应头详情请参见 公共响应头部 文档。

特有响应头

该响应无特殊的响应头。

响应体

该响应体为空。

错误码

该请求操作返回如下错误信息，常见的错误信息请参见 错误码 文档。

错误码	描述	HTTP 状态码
None	删除成功，响应体返回为空	204 No Content
NoSuchBucket	当访问的 Bucket 不存在，返回该错误码	404 Not Found

实际案例

请求

```
DELETE /?lifecycle HTTP/1.1
Host: examplebucket-1250000000.cos.ap-beijing.myqcloud.com
Date: Wed, 16 Aug 2017 12:59:09 GMT
Authorization:q-sign-algorithm=sha1&q-ak=AKIDxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx&q-sign-time=1502859472;1502939472&q-key-time=1502859472;1502939472&q-header-list=host&q-url-param-list=lifecycle&q-signature=49c1414c700643f111392193843332a3ec4e9485b
```

响应

```
HTTP /1.1 204 No Content
Content-Type: application/xml
Date: Wed, 16 Aug 2017 12:59:09 GMT
Server: tencent-cos
x-cos-request-id: NTk5NDQxOWNfMjQ4OGY3Xzc3NGRfMjE=
```

存储桶策略（policy）
设置存储桶策略

最近更新时间: 2025-01-15 17:01:00

功能描述

PUT Bucket policy 请求可以向 Bucket 写入权限策略，当 Bucket 已存在权限策略时，该请求上传的策略将覆盖原有的权限策略。

请求

请求示例

```
PUT /?policy HTTP/1.1
Host: <BucketName-APPID>.<Endpoint>
Date: date
Content-Type:application/json
Content-MD5:MD5
Authorization: Auth String
```

> **说明：**
>
> Authorization: Auth String（详细参见请求签名文档）。

请求头

公共头部

该请求操作的实现使用公共请求头，了解公共请求头详情，请查阅 公共请求头部 章节。

请求参数

无特殊请求参数。

请求体

关于访问策略中的元素介绍，请参阅 访问策略语言概述 章节。

```
{
"Statement": [
{
"Principal": {
"qcs": [
"qcs::cam::uin/${owner_uin}:uin/${sub_uin}"
]
},
"Effect": "${effect}",
"Action": [
"name/cos:${action}"
],
"Resource": [
"qcs::cos:${region}:uid/${appid}:${bucket}-${appid}/*"
]
}
],
"version": "2.0"
}
```

响应

响应头

公共响应头

该响应使用公共响应头，了解公共响应头详情，请参见 公共响应头部 章节。

特有响应头

该请求操作无特殊的响应头部信息。

响应体

该请求响应体为空。

错误码

无返回特殊错误码。一般常见错误码，请参阅 错误码 文档。

实际案例

请求

```
PUT /?policy HTTP/1.1
Host: <BucketName-APPID>.<Endpoint>
Authorization:q-sign-algorithm=sha1&q-ak=AKIDxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx&q-sign-time=1484813288;32557709288&q-key-time=1484813288;32557709288&q-header-list=host&q-url-param-list=policy&q-signature=05f7fc936369f910a94a0c815e1f1752f034d47a
Content-Type: application/json
Content-Length: 233

{
"Statement": [
{
"Principal": {
"qcs": [
"qcs::cam::uin/1250000000:uin/1250000000"
]
},
"Effect": "allow",
"Action": [
"name/cos:GetBucket"
],
"Resource": [
"qcs::cos:ap-chengdu:uid/1250000000:examplebucket-1250000000/*"
]
}
],
"version": "2.0"
}
```

响应

```
HTTP/1.1 204 No Content
Content-Type: application/xml
Content-Length: 0
Connection: keep-alive
Date: Thu Jan 19 16:19:22 2017
Server: tencent-cos
x-cos-request-id: NTg4MDc2OGFfNDUyMDRlXzc3NTlfZTc4
```

查询存储桶策略

最近更新时间: 2025-01-15 17:01:00

功能描述

GET Bucket policy 请求可以向 Bucket 读取权限策略。

请求

请求示例

```
GET /?policy HTTP/1.1
Host: <BucketName-APPID>.<Endpoint>
Date:date
Authorization: Auth String
```

> **说明：**
>
> Authorization: Auth String（详情请参见请求签名文档）。

请求头

公共头部

该请求操作的实现使用公共请求头，了解公共请求头详情请参见 公共请求头部 文档。

非公共头部

该请求操作无特殊的请求头部信息。

请求体

该请求的请求体为空。

响应

响应头

公共响应头

该响应包含公共响应头，了解公共响应头详情请参见 公共响应头部 文档。

特有响应头

该响应无特殊的响应头。

响应体

响应返回权限策略。

```
{
"Statement": [
{
"Principal": {
"qcs": [
"qcs::cam::uin/100000000001:uin/100000000001"
]
},
"Effect": "allow",
"Action": [
"name/cos:GetBucket"
],
"Resource": [
"qcs::cos:ap-guangzhou:uid/1250000000:examplebucket-1250000000/*"
]
}
```

```
],
"version": "2.0"
}
```

错误码

该请求操作无特殊错误信息，常见的错误信息请参见 错误码 文档。

实际案例

请求

```
GET /?policy HTTP/1.1
Host: <BucketName-APPID>.<Endpoint>
Authorization:q-sign-algorithm=sha1&q-ak=AKIDxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx&q-sign-time=1484814099;32557710099&q-key-time=1484814099;32557710099&q-header-list=host&q-url-param-list=policy&q-signature=0523d7c6305b6676611c44798d2c48b659e68869
```

响应

```
HTTP/1.1 200 OK
Content-Type: application/xml
Content-Length: 237
Connection: keep-alive
Date: Thu Jan 19 16:21:46 2017
Server: tencent-cos
x-cos-request-id: NTg4MDc3MWFfOWIxZjRlXzZmNDVfZTBl

{
"Statement": [
{
"Principal": {
"qcs": [
"qcs::cam::uin/100000000001:uin/100000000001"
]
},
"Effect": "allow",
"Action": [
"name/cos:GetBucket"
],
"Resource": [
"qcs::cos:ap-guangzhou:uid/1250000000:examplebucket-1250000000/*"
]
}
],
"version": "2.0"
}
```

删除存储桶策略

最近更新时间: 2025-01-15 17:01:00

功能描述

DELETE Bucket policy 请求可以向 Bucket 删除权限策略。

> **注意：**
>
> 只有 Bucket 所有者有权限发起该请求。如果权限策略不存在，将返回204 No Content。

请求

请求示例

```
DELETE /?policy HTTP/1.1
Host: <BucketName-APPID>.<Endpoint>
Date: date
Authorization: Auth String
```

> **说明：**
>
> Authorization: Auth String（详细参见请求签名文档）。

请求头

公共头部

该请求操作的实现使用公共请求头，了解公共请求头详细请参见 公共请求头部 文档。

非公共头部

该请求操作无特殊的请求头部信息。

请求体

该请求的请求体为空。

响应

响应头

公共响应头

该响应包含公共响应头，了解公共响应头详情请参见 公共响应头部 文档。

特有响应头

该响应无特殊的响应头。

响应体

该响应体为空。

实际案例

请求

```
DELETE /?policy HTTP/1.1
Host: <BucketName-APPID>.<Endpoint>
Authorization:q-sign-algorithm=sha1&q-ak=AKIDxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx&q-sign-time=1484814613;32557710613&q-key-time=1484814613;32557710613&q-header-list=host&q-url-param-list=policy&q-signature=57c9a3f67b786ddabd2c208641944ec7f9b02f98
```

响应

```
HTTP/1.1 204 No Content
Content-Type: application/xml
Content-Length: 0
Connection: keep-alive
Date: Thu Jan 19 16:30:21 2017
Server: tencent-cos
x-cos-request-id: NTg4MDc5MWRfNDQyMDRlXzNiMDRfZTEw
```

防盗链（referer）

设置存储桶referer

最近更新时间: 2025-01-15 17:01:00

功能描述

PUT Bucket referer 接口用于为存储桶设置 Referer 白名单或者黑名单。

请求

请求示例

```
PUT /?referer HTTP 1.1
Host: <BucketName-APPID>.<Endpoint>
Date:GMT Date
Authorization: Auth String
Content-Length:length
Content-MD5:MD5
```

> **说明：**
>
> Authorization：Auth String（详情请参见请求签名文档）。

请求头

公共头部

该请求操作的实现使用公共请求头，了解公共请求头详情请参见 公共请求头部 文档。

非公共头部

必选头部

名称	描述	类型	必选
Content-Length	RFC 2616中定义的 HTTP 请求内容长度（字节）	String	是
Content-MD5	RFC 1864中定义的经过 Base64 编码的 128-bit 内容的 MD5 校验值，此头部用来校验文件内容是否发生变化	String	是

请求体

该请求的请求体具体节点内容为：

```
<RefererConfiguration>
<Status>Enabled</Status>
<RefererType>White-List</RefererType>
<DomainList>
<Domain>*.qq.com</Domain>
<Domain>*.qcloud.com</Domain>
</DomainList>
<EmptyReferConfiguration>Allow</EmptyReferConfiguration>
</RefererConfiguration>
```

具体内容描述如下：

名称	父节点	描述	类型	必选
RefererConfiguration	无	防盗链配置信息	Container	是
Status	RefererConfiguration	是否开启防盗链，枚举值：Enabled、Disabled	String	是
RefererType	RefererConfiguration	防盗链类型，枚举值：Black-List、White-List	String	是
DomainList	RefererConfiguration	生效域名列表， 支持多个域名且为前缀匹配， 支持带端口的域名和 IP， 支持通配符 `*`，做二级域名或多级域名的通配	Container	是

名称	父节点	描述	类型	必选
Domain	DomainList	单条生效域名 例如`www.qq.com/example`，`192.168.1.2:8080`，`*.qq.com`	String	是
EmptyReferConfiguration	RefererConfiguration	是否允许空 Referer 访问，枚举值：Allow、Deny，默认值为 Deny	String	否

响应

响应头

此接口仅返回公共响应头部，详情请参见 公共响应头部 文档。

响应体

该响应体为空。

错误码

该请求操作无特殊错误信息，全部错误信息请参见 错误码 文档。

实际案例

请求

```
PUT /?referer HTTP 1.1
Host: <BucketName-APPID>.<Endpoint>
Date: Fri, 25 Feb 2017 04:10:22 GMT
Authorization:q-sign-algorithm=sha1&q-ak=AKIDZfbOAo7cllgPvF9cXFrJD0a1ICvR****&q-sign-time=1547105134;32526689134&q-key-time=1547105134;32620001134&q-header-list=content-md5;content-type;host&q-url-param-list=referer&q-signature=0f7fef5b1d2180deaf6f92fa2ee0cf87ae83f****
Content-MD5: kOz7g54LMJZjxKs070V9jQ==
Content-Type: application/xml

<RefererConfiguration>
<Status>Enabled</Status>
<RefererType>White-List</RefererType>
<DomainList>
<Domain>*.qq.com</Domain>
<Domain>*.qcloud.com</Domain>
</DomainList>
<EmptyReferConfiguration>Allow</EmptyReferConfiguration>
</RefererConfiguration>
```

响应

```
HTTP/1.1 200 OK
Content-Length: 0
Connection: keep-alive
Date: Fri, 25 Feb 2017 04:10:22 GMT
Server: tencent-cos
x-cos-request-id: NTg3ZjFjMmJfOWIxZjRlXzZmNDhf****
```

查询存储桶referer

最近更新时间: 2025-01-15 17:01:00

功能描述

GET Bucket referer 接口用于读取存储桶 Referer 白名单或者黑名单。

请求

请求示例

```
GET /?referer HTTP 1.1
Host: <BucketName-APPID>.<Endpoint>
Date: GMT Date
Authorization: Auth String
```

> **说明：**
>
> Authorization：Auth String（详请请参见请求签名文档）。

请求头

此接口仅使用公共请求头部，详情请参见 公共请求头部 文档。

请求体

该请求的请求体为空。

响应

响应头

此接口仅返回公共响应头部，详情请参见 公共响应头部 文档。

响应体

该响应体返回为 application/xml 数据，包含完整节点数据的内容展示如下：

```
<RefererConfiguration>
<Status>Enabled</Status>
<RefererType>White-List</RefererType>
<DomainList>
<Domain>*.qq.com</Domain>
<Domain>*.qcloud.com</Domain>
</DomainList>
<EmptyReferConfiguration>Allow</EmptyReferConfiguration>
</RefererConfiguration>
```

具体的数据内容如下：

名称	父节点	描述	类型	必选
RefererConfiguration	无	防盗链配置信息	Container	是
Status	RefererConfiguration	是否开启防盗链，枚举值：Enabled，Disabled	String	是
RefererType	RefererConfiguration	防盗链类型，枚举值：Black-List，White-List	String	是
DomainList	RefererConfiguration	生效域名列表。 支持多个域名且为前缀匹配， 支持带端口的域名和 IP， 支持通配符 `*`，做二级域名或多级域名的通配	Container	是
Domain	DomainList	单条生效域名，例如 `www.qq.com/example`，`192.168.1.2:8080`， `*.qq.com`	String	是
EmptyReferConfiguration	RefererConfiguration	是否允许空 Referer 访问，枚举值：Allow，Deny，默认值为 Deny	String	否

错误码

该请求操作无特殊错误信息，全部错误信息请参见 错误码 文档。

实际案例

请求

```
GET /?referer HTTP 1.1
Host: <BucketName-APPID>.<Endpoint>
Date: Fri, 25 Feb 2017 04:10:22 GMT
Authorization:q-sign-algorithm=sha1&q-ak=AKIDZfbOAo7cllgPvF9cXFrJD0a1ICvR****&q-sign-time=1547105134;32526689134&q-key-time=1547105134;
32620001134&q-header-list=content-md5;content-type;host&q-url-param-list=referer&q-signature=0f7fef5b1d2180deaf6f92fa2ee0cf87ae83****
```

响应

```
HTTP/1.1 200 OK
Content-Type: application/xml
Content-Length: 260
Connection: keep-alive
Date: Fri, 25 Feb 2017 04:10:22 GMT
Server: tencent-cos
x-cos-request-id: NTg3ZjFjMmJfOWIxZjRlXzZmNDhf****

<RefererConfiguration>
<Status>Enabled</Status>
<RefererType>White-List</RefererType>
<DomainList>
<Domain>*.qq.com</Domain>
<Domain>*.qcloud.com</Domain>
</DomainList>
<EmptyReferConfiguration>Allow</EmptyReferConfiguration>
</RefererConfiguration>
```

标签（tagging）
设置存储桶标签

最近更新时间: 2025-01-15 17:01:00

功能描述

COS 支持为已存在的 Bucket 设置标签（Tag）。PUT Bucket tagging 接口用于为存储桶设置键值对作为存储桶标签，可以协助您管理已有的存储桶资源，并通过标签进行成本管理。

> **注意：**
>
> 目前存储桶标签功能最多支持一个存储桶下设置50个不同的标签。

请求

请求示例

```
PUT /?tagging HTTP 1.1
Host:<BucketName-APPID>.<Endpoint>
Date: GMT Date
Authorization: Auth String
Content-MD5: MD5
Content-Length: Content Length
Content-Type: application/xml

[Request Body]
```

> **说明：**
>
> Authorization: Auth String（详情请参见请求签名文档）。

请求头

此接口除使用公共请求头部外，还支持以下请求头部，了解公共请求头部详情请参见 公共请求头部 文档。

名称	描述	类型	是否必选
Content-MD5	RFC 1864 中定义的经过 Base64 编码的请求体内容 MD5 哈希值，用于完整性检查，验证请求体在传输过程中是否发生变化	string	是

请求体

该请求需要设置如下标签集合：

```
<?xml version="1.0" encoding="UTF-8" ?>
<Tagging>
<TagSet>
<Tag>
<Key>age</Key>
<Value>18</Value>
</Tag>
<Tag>
<Key>name</Key>
<Value>xiaoming</Value>
</Tag>
</TagSet>
</Tagging>
```

具体的数据描述如下：

节点名称（关键字）	父节点	描述	类型	必选
Tagging	无	标签集合	Container	是

节点名称（关键字）	父节点	描述	类型	必选
TagSet	Tagging	标签集合	Container	是
Tag	Tagging.TagSet	标签集合，最多支持10个标签	Containers	是
Key	Tagging.TagSet.Tag	标签的 Key，长度不超过128字节, 支持英文字母、数字、空格、加号、减号、下划线、等号、点号、冒号、斜线	String	是
Value	Tagging.TagSet.Tag	标签的 Value，长度不超过256字节, 支持英文字母、数字、空格、加号、减号、下划线、等号、点号、冒号、斜线	String	是

响应

响应头

此接口仅返回公共响应头部，详情请参见 公共响应头部 文档。

响应体

该请求响应体为空。

错误码

以下描述此请求可能会发生的一些特殊的且常见的错误情况：

错误码	描述	HTTP 状态码
SignatureDoesNotMatch	提供的签名不符合规则，返回该错误码	403 Forbidden
NoSuchBucket	如果试图添加的规则所在的 Bucket 不存在，返回该错误码	404 Not Found
MalformedXML	XML 格式不合法，请跟 restful api 文档仔细比对	400 Bad Request
BadRequest	如超过了允许一个 Bucket 最大设置的 Tag 数量，目前最大支持10个	400 Bad Request
InvalidTag	Tag 的 key 和 value 中包含了保留字符串 cos: 或者 Project	400 Bad Request

实际案例

请求

如下请求向存储桶 `examplebucket-1250000000` 中写入了{age:18}和{name:xiaoming}两个标签。COS 配置标签成功并返回204成功请求。

```
PUT /?tagging HTTP/1.1
User-Agent: curl/7.29.0
Accept: */*
Host: examplebucket-1250000000.<Endpoint>
Authorization: q-sign-algorithm=sha1&q-ak=AKIDxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx&q-sign-time=1516361923;1517361973&q-key-time=1516361923;1517361973&q-url-param-list=tagging&q-header-list=content-md5;host&q-signature=71251feb4501494edcfbd01747fa873003759404
Content-MD5: LIbd5t5HLPhuNWYkP6qHcQ==
Content-Length: 127
Content-Type: application/xml

<Tagging>
<TagSet>
<Tag>
<Key>age</Key>
<Value>18</Value>
</Tag>
<Tag>
<Key>name</Key>
<Value>xiaoming</Value>
</Tag>
</TagSet>
</Tagging>
```

响应

```
HTTP/1.1 204 No Content
Content-Type: application/xml
Content-Length: 0
Connection: keep-alive
Date: Fri, 19 Jan 2018 11:40:22 GMT
Server: tencent-cos
x-cos-request-id: NWE2MWQ5MjZfMTBhYzM1MGFfMTA5ODVfMTVjNDM=
```

查询存储桶标签

最近更新时间: 2025-01-15 17:01:00

功能描述

COS 支持为已存在的 Bucket 查询标签（Tag）。GET Bucket tagging 接口用于查询指定存储桶下已有的存储桶标签。

> **说明：**
>
> 若您使用子账号调用此项接口，请确保您已经在主账号处获取了 GET Bucket tagging 这个接口的权限。

请求

请求示例

```
GET /?tagging HTTP 1.1
Host:<BucketName-APPID>.<Endpoint>
Date: GMT Date
Authorization: Auth String
```

> **说明：**
>
> Authorization: Auth String（详情请参见请求签名文档）。

请求头

此接口仅使用公共请求头部，详情请参见 公共请求头部 文档。

请求体

该请求的请求体为空。

响应

响应头

此接口仅返回公共响应头部，详情请参见 公共响应头部 文档。

响应体

查询成功，返回 application/xml 数据，包含存储桶下已有的标签信息。

```
<Tagging>
<TagSet>
<Tag>
<Key>string</Key>
<Value>string</Value>
</Tag>
<Tag>
<Key>string</Key>
<Value>string</Value>
</Tag>
</TagSet>
</Tagging>
```

具体的节点描述如下：

节点名称（关键字）	父节点	描述	类型
Tagging	无	标签集合	Container
TagSet	Tagging	标签集合	Container
Tag	Tagging.TagSet	标签集合, 最多支持50个标签	Containers
Key	Tagging.TagSet.Tag	标签键, 长度不超过128字节，支持英文字母、数字、空格、加号、减号、下划线、等号、点号、冒号、斜线	String

节点名称（关键字）	父节点	描述	类型
Value	Tagging.TagSet.Tag	标签值，长度不超过256字节，支持英文字母、数字、空格、加号、减号、下划线、等号、点号、冒号、斜线	String

错误码

以下描述此请求可能会发生的一些特殊的且常见的错误情况：

错误码	描述	HTTP 状态码
SignatureDoesNotMatch	提供的签名不符合规则，返回该错误码	403 Forbidden
NoSuchBucket	如果试图添加的规则所在的 Bucket 不存在，返回该错误码	404 Not Found
NoSuchTagSetError	请求的存储桶中未设置存储桶标签	404 Not Found

实际案例

请求

如下请求申请查询存储桶 examplebucket-1250000000 下的标签信息，COS 解析该请求后，并返回了该存储桶下已有的 {age:18} 和 {name:xiaoming} 两个标签。

```
GET /?tagging HTTP/1.1
User-Agent: curl/7.29.0
Accept: */*
Host: examplebucket-1250000000.<Endpoint>
Authorization: q-sign-algorithm=sha1&q-ak=AKIDxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx&q-sign-time=1516361923;1517361973&q-key-time=1516361923;1517361973&q-url-param-list=tagging&q-header-list=content-md5;host&q-signature=71251feb4501494edcfbd01747fa873003759404
Content-Length: 127
Content-Type: application/xml
```

响应

```
HTTP/1.1 200 OK
Content-Type: application/xml
Connection: close
Date: Fri, 19 Jan 2018 11:40:22 GMT
Server: tencent-cos
<Tagging>
<TagSet>
<Tag>
<Key>age</Key>
<Value>18</Value>
</Tag>
<Tag>
<Key>name</Key>
<Value>xiaoming</Value>
</Tag>
</TagSet>
</Tagging>
```

删除存储桶标签

最近更新时间: 2025-01-15 17:01:00

功能描述

COS 支持为已存在的 Bucket 删除标签（Tag）。DELETE Bucket tagging 接口用于删除指定存储桶下已有的存储桶标签。

> **说明：**
>
> 如您使用子账号调用此项接口，请确保您已经在主账号处获取了 `DELETE Bucket tagging` 这个接口的权限。

请求

请求示例

```
DELETE /?tagging HTTP 1.1
Host:<BucketName-APPID>.<Endpoint>
Date:date
Authorization: Auth String
```

> **说明：**
>
> Authorization: Auth String（详情请参见请求签名文档）。

请求头

公共头部

该请求操作的实现使用公共请求头，了解公共请求头详情，请参见 公共请求头部 文档。

非公共头部

该请求操作无特殊的请求头部信息。

请求体

该请求的请求体为空。

响应

响应头

公共响应头

该响应使用公共响应头，了解公共响应头详情，请参见 公共响应头部 文档。

特有响应头

该请求操作无特殊的响应头部信息。

响应体

该请求无特殊响应体信息。

错误码

以下描述此请求可能会发生的一些特殊的且常见的错误情况：

错误码	描述	HTTP 状态码
SignatureDoesNotMatch	提供的签名不符合规则，返回该错误码	403 Forbidden
NoSuchBucket	如果试图添加的规则所在的 Bucket 不存在，返回该错误码	404 Not Found
NoSuchTagSetError	请求的存储桶中未设置存储桶标签	404 Not Found

实际案例

请求

如下请求申请删除存储桶 examplebucket-1250000000 下已有的标签信息。COS 解析该请求后删除该存储桶下所有标签。

```
DELETE /?tagging HTTP/1.1
User-Agent: curl/7.29.0
Accept: */*
Host: examplebucket-1250000000.<Endpoint>
Authorization: q-sign-algorithm=sha1&q-ak=AKIDxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx&q-sign-time=1516361923;1517361973&q-key-time=1516361923;1517361973&q-url-param-list=tagging&q-header-list=content-md5;host&q-signature=71251feb4501494edcfbd01747fa873003759404
Content-Md5: LIbd5t5HLPhuNWYkP6qHcQ==
Content-Length: 127
Content-Type: application/xml
```

响应

```
HTTP/1.1 204 No Content
Content-Type: application/xml
Connection: close
Date: Fri, 19 Jan 2018 11:40:22 GMT
```

静态网站（website）
设置静态网站

最近更新时间: 2025-01-15 17:01:00

功能描述

PUT Bucket website 请求用于为存储桶配置静态网站，您可以通过传入 XML 格式的配置文件进行配置，文件大小限制为64KB。

推荐使用运营端控制台 **平台管理 > 云Api管理 > API Explorer** 进行调试。

> **说明：**
>
> API Explorer 提供了在线调用、签名验证、SDK 代码生成和快速检索接口等能力。您可查看每次调用的请求内容和返回结果以及自动生成 SDK 调用示例。

请求

请求示例

```
PUT /?website HTTP/1.1
Host: <BucketName-APPID>.cos.<Region>.myqcloud.com
Date: GMT Date
Content-Type: application/xml
Content-Length: Content Length
Content-MD5: MD5
Authorization: Auth String

[Request Body]
```

> **说明：**
>
> - `Host: <BucketName-APPID>.cos.<Region>.myqcloud.com`，其中 `<BucketName-APPID>` 为带 APPID 后缀的存储桶名字，例如 examplebucket-1250000000，`<Region>` 为 COS 的可用地域。
> - Authorization: Auth String（详情请参见 请求签名 文档）。

请求参数

此接口无请求参数。

请求头

此接口仅使用公共请求头部，详情请参见 公共请求头部 文档。

请求体

提交 **application/xml** 请求数据，包含完整的存储桶静态网站配置信息。

```
<WebsiteConfiguration>
<IndexDocument>
<Suffix>String</Suffix>
</IndexDocument>
<RedirectAllRequestsTo>
<Protocol>String</Protocol>
</RedirectAllRequestsTo>
<AutoAddressing>
<Status>Enabled|Disabled</Status>
</AutoAddressing>
<ErrorDocument>
<Key>String</Key>
<OriginalHttpStatus>Enabled|Disabled</OriginalHttpStatus>
</ErrorDocument>
<RoutingRules>
<RoutingRule>
<Condition>
<HttpErrorCodeReturnedEquals>Integer</HttpErrorCodeReturnedEquals>
</Condition>
<Redirect>
```

```
<Protocol>String</Protocol>
<ReplaceKeyWith>String</ReplaceKeyWith>
</Redirect>
</RoutingRule>
<RoutingRule>
<Condition>
<KeyPrefixEquals>String</KeyPrefixEquals>
</Condition>
<Redirect>
<Protocol>String</Protocol>
<ReplaceKeyPrefixWith>String</ReplaceKeyPrefixWith>
</Redirect>
</RoutingRule>
</RoutingRules>
</WebsiteConfiguration>
```

具体的节点描述如下：

节点名称（关键字）	父节点	描述	类型	是否必选
WebsiteConfiguration	无	包含 PUT Bucket website 操作的所有请求信息	Container	否

Container 节点 WebsiteConfiguration 的内容：

节点名称（关键字）	父节点	描述	类型	是否必选
IndexDocument	WebsiteConfiguration	索引文档配置	Container	是
RedirectAllRequestsTo	WebsiteConfiguration	重定向所有请求配置	Container	否
AutoAddressing	WebsiteConfiguration	用于配置是否忽略扩展名	Container	否
ErrorDocument	WebsiteConfiguration	错误文档配置	Container	否
RoutingRules	WebsiteConfiguration	重定向规则配置，最多设置100条 RoutingRule	Container	否

Container 节点 IndexDocument 的内容：

节点名称（关键字）	父节点	描述	类型	是否必选
Suffix	WebsiteConfiguration.IndexDocument	指定索引文档的对象键后缀。例如指定为`index.html`，那么当访问到存储桶的根目录时，会自动返回 index.html 的内容，或者当访问到`article/`目录时，会自动返回 `article/index.html`的内容	String	是

Container 节点 RedirectAllRequestsTo 的内容：

节点名称（关键字）	父节点	描述	类型	是否必选
Protocol	WebsiteConfiguration.RedirectAllRequestsTo	指定重定向所有请求的目标协议，只能设置为 https	String	是

Container 节点 AutoAddressing 的内容：

节点名称（关键字）	父节点	描述	类型	是否必选
Status	WebsiteConfiguration.AutoAddressing	用于配置是否忽略 HTML 拓展名，可选值为 Enabled 或 Disabled，默认为 Disabled	String	否

Container 节点 ErrorDocument 的内容：

节点名称（关键字）	父节点	描述	类型	是否必选
Key	WebsiteConfiguration.ErrorDocument	指定通用错误文档的对象键，当发生错误且未命中重定向规则中的错误码重定向时，将返回该对象键的内容	String	是
OriginalHttpStatus	WebsiteConfiguration.ErrorDocument	用于配置命中错误文档的 HTTP 状态码，可选值为 Enabled 或 Disabled，默认为 Enabled	String	否

Container 节点 RoutingRules 的内容：

节点名称（关键字）	父节点	描述	类型	是否必选
RoutingRule	WebsiteConfiguration.RoutingRules	单条重定向规则配置	Container	是

Container 节点 RoutingRules.RoutingRule 的内容：

节点名称（关键字）	父节点	描述	类型	是否必选
Condition	WebsiteConfiguration.RoutingRules.RoutingRule	重定向规则的条件配置	Container	是
Redirect	WebsiteConfiguration.RoutingRules.RoutingRule	重定向规则的具体重定向目标配置	Container	是

Container 节点 RoutingRules.RoutingRule.Condition 的内容：

节点名称（关键字）	父节点	描述	类型	是否必选
HttpErrorCodeReturnedEquals	WebsiteConfiguration.RoutingRules.RoutingRule.Condition	指定重定向规则的错误码匹配条件，只支持配置4XX返回码，例如403或404	Integer	HttpErrorCodeReturnedEquals 与 KeyPrefixEquals 必选其一
KeyPrefixEquals	WebsiteConfiguration.RoutingRules.RoutingRule.Condition	指定重定向规则的对象键前缀匹配条件	String	HttpErrorCodeReturnedEquals 与 KeyPrefixEquals 必选其一

Container 节点 RoutingRules.RoutingRule.Redirect 的内容：

节点名称（关键字）	父节点	描述	类型	是否必选
Protocol	WebsiteConfiguration.RoutingRules.RoutingRule.Redirect	指定重定向规则的目标协议，只能设置为 https	String	否
ReplaceKeyWith	WebsiteConfiguration.RoutingRules.RoutingRule.Redirect	指定重定向规则的具体重定向目标的对象键，替换方式为替换整个原始请求的对象键	String	ReplaceKeyWith 与 ReplaceKeyPrefixWith 必选其一
ReplaceKeyPrefixWith	WebsiteConfiguration.RoutingRules.RoutingRule.Redirect	指定重定向规则的具体重定向目标的对象键，替换方式为替换原始请求中所匹配到的前缀部分，仅可在 Condition 为 KeyPrefixEquals 时设置	String	ReplaceKeyWith 与 ReplaceKeyPrefixWith 必选其一

响应

响应头

此接口仅返回公共响应头部，详情请参见 公共响应头部 文档。

响应体

此接口响应体为空。

错误码

此接口遵循统一的错误响应和错误码，详情请参见 错误码 文档。

实际案例

请求

```
PUT /?website HTTP/1.1
Host: examplebucket-1250000000.cos.ap-beijing.myqcloud.com
Date: Wed, 20 May 2020 09:33:38 GMT
Content-Type: application/xml
Content-Length: 1209
Content-MD5: VHzj4Uwb++HLyCJp7jUzWg==
```

```
Authorization: q-sign-algorithm=sha1&q-ak=AKID8A0fBVtYFrNm02oY1g1JQQF0c3JO****&q-sign-time=1589967218;1589974418&q-key-time=1589967218;1589974418&q-header-list=content-length;content-md5;content-type;date;host&q-url-param-list=website&q-signature=4666493555640e834a879c78afaa4fd9b16a****
Connection: close

<WebsiteConfiguration>
<IndexDocument>
<Suffix>index.html</Suffix>
</IndexDocument>
<RedirectAllRequestsTo>
<Protocol>https</Protocol>
</RedirectAllRequestsTo>
<ErrorDocument>
<Key>pages/error.html</Key>
</ErrorDocument>
<RoutingRules>
<RoutingRule>
<Condition>
<HttpErrorCodeReturnedEquals>403</HttpErrorCodeReturnedEquals>
</Condition>
<Redirect>
<Protocol>https</Protocol>
<ReplaceKeyWith>pages/403.html</ReplaceKeyWith>
</Redirect>
</RoutingRule>
<RoutingRule>
<Condition>
<HttpErrorCodeReturnedEquals>404</HttpErrorCodeReturnedEquals>
</Condition>
<Redirect>
<ReplaceKeyWith>pages/404.html</ReplaceKeyWith>
</Redirect>
</RoutingRule>
<RoutingRule>
<Condition>
<KeyPrefixEquals>assets/</KeyPrefixEquals>
</Condition>
<Redirect>
<ReplaceKeyWith>index.html</ReplaceKeyWith>
</Redirect>
</RoutingRule>
<RoutingRule>
<Condition>
<KeyPrefixEquals>article/</KeyPrefixEquals>
</Condition>
<Redirect>
<Protocol>https</Protocol>
<ReplaceKeyPrefixWith>archived/</ReplaceKeyPrefixWith>
</Redirect>
</RoutingRule>
</RoutingRules>
</WebsiteConfiguration>
```

响应

```
HTTP/1.1 200 OK
Content-Length: 0
Connection: close
Date: Wed, 20 May 2020 09:33:38 GMT
Server: tencent-cos
x-cos-request-id: NWVjNGY5NzJfOThjMjJhMDlfMjg5Ml8yYzNi****
```

查询静态网站

最近更新时间: 2025-01-15 17:01:00

功能描述

GET Bucket website 请求用于查询与存储桶关联的静态网站配置信息。

推荐使用运营端控制台 **平台管理 > 云Api管理 > API Explorer** 进行调试。

> **说明：**
>
> API Explorer 提供了在线调用、签名验证、SDK 代码生成和快速检索接口等能力。您可查看每次调用的请求内容和返回结果以及自动生成 SDK 调用示例。

请求

请求示例

```
GET /?website HTTP/1.1
Host: <BucketName-APPID>.cos.<Region>.myqcloud.com
Date: GMT Date
Authorization: Auth String
```

> **说明：**
>
> - `Host: <BucketName-APPID>.cos.<Region>.myqcloud.com` ，其中 `<BucketName-APPID>` 为带 APPID 后缀的存储桶名字，例如 examplebucket-1250000000， `<Region>` 为 COS 的可用地域。
> - Authorization: Auth String（详情请参见 请求签名 文档）。

请求参数

此接口无请求参数。

请求头

此接口仅使用公共请求头部，详情请参见 公共请求头部 文档。

请求体

此接口无请求体。

响应

响应头

此接口仅返回公共响应头部，详情请参见 公共响应头部 文档。

响应体

查询成功，返回 **application/xml** 数据，包含完整的存储桶静态网站配置信息。

```
<WebsiteConfiguration>
<IndexDocument>
<Suffix>string</Suffix>
</IndexDocument>
<RedirectAllRequestsTo>
<Protocol>string</Protocol>
</RedirectAllRequestsTo>
<ErrorDocument>
<Key>string</Key>
</ErrorDocument>
<RoutingRules>
<RoutingRule>
<Condition>
<HttpErrorCodeReturnedEquals>integer</HttpErrorCodeReturnedEquals>
</Condition>
<Redirect>
```

```
<Protocol>string</Protocol>
<ReplaceKeyWith>string</ReplaceKeyWith>
</Redirect>
</RoutingRule>
<RoutingRule>
<Condition>
<KeyPrefixEquals>string</KeyPrefixEquals>
</Condition>
<Redirect>
<Protocol>string</Protocol>
<ReplaceKeyPrefixWith>string</ReplaceKeyPrefixWith>
</Redirect>
</RoutingRule>
</RoutingRules>
</WebsiteConfiguration>
```

具体的节点描述如下：

节点名称（关键字）	父节点	描述	类型
WebsiteConfiguration	无	保存 GET Bucket website 结果的所有信息	Container

Container 节点 WebsiteConfiguration 的内容：

节点名称（关键字）	父节点	描述	类型
IndexDocument	WebsiteConfiguration	索引文档配置	Container
RedirectAllRequestsTo	WebsiteConfiguration	重定向所有请求配置	Container
ErrorDocument	WebsiteConfiguration	错误文档配置	Container
RoutingRules	WebsiteConfiguration	重定向规则配置	Container

Container 节点 IndexDocument 的内容：

节点名称（关键字）	父节点	描述	类型
Suffix	WebsiteConfiguration.IndexDocument	指定索引文档的对象键后缀。例如指定为`index.html`，那么当访问到存储桶的根目录时，会自动返回 index.html 的内容，或者当访问到`article/`目录时，会自动返回`article/index.html`的内容	string

Container 节点 RedirectAllRequestsTo 的内容：

节点名称（关键字）	父节点	描述	类型
Protocol	WebsiteConfiguration.RedirectAllRequestsTo	指定重定向所有请求的目标协议	string

Container 节点 ErrorDocument 的内容：

节点名称（关键字）	父节点	描述	类型
Key	WebsiteConfiguration.ErrorDocument	指定通用错误文档的对象键	string

Container 节点 RoutingRules 的内容：

节点名称（关键字）	父节点	描述	类型
RoutingRule	WebsiteConfiguration.RoutingRules	单条重定向规则配置	Container

Container 节点 RoutingRules.RoutingRule 的内容：

节点名称（关键字）	父节点	描述	类型
Condition	WebsiteConfiguration.RoutingRules.RoutingRule	重定向规则的条件配置	Container
Redirect	WebsiteConfiguration.RoutingRules.RoutingRule	重定向规则的具体重定向目标配置	Container

Container 节点 RoutingRules.RoutingRule.Condition 的内容：

节点名称（关键字）	父节点	描述	类型
HttpErrorCodeReturnedEquals	WebsiteConfiguration.RoutingRules.RoutingRule.Condition	指定重定向规则的错误码匹配条件	integer
KeyPrefixEquals	WebsiteConfiguration.RoutingRules.RoutingRule.Condition	指定重定向规则的对象键前缀匹配条件	string

Container 节点 RoutingRules.RoutingRule.Redirect 的内容：

节点名称（关键字）	父节点	描述	类型
Protocol	WebsiteConfiguration.RoutingRules.RoutingRule.Redirect	指定重定向规则的目标协议	string
ReplaceKeyWith	WebsiteConfiguration.RoutingRules.RoutingRule.Redirect	指定重定向规则的具体重定向目标的对象键，替换方式为替换整个原始请求的对象键	string
ReplaceKeyPrefixWith	WebsiteConfiguration.RoutingRules.RoutingRule.Redirect	指定重定向规则的具体重定向目标的对象键，替换方式为替换原始请求中所匹配到的前缀部分	string

错误码

此接口遵循统一的错误响应和错误码，详情请参见 错误码 文档。

实际案例

请求

```
GET /?website HTTP/1.1
Host: examplebucket-1250000000.cos.ap-beijing.myqcloud.com
Date: Wed, 20 May 2020 09:33:49 GMT
Authorization: q-sign-algorithm=sha1&q-ak=AKID8A0fBVtYFrNm02oY1g1JQQF0c3JO****&q-sign-time=1589967229;1589974429&q-key-time=1589967229;1589974429&q-header-list=date;host&q-url-param-list=website&q-signature=50a22a30b02b59e5da4a0820d15a36805ea7****
Connection: close
```

响应

```
HTTP/1.1 200 OK
Content-Type: application/xml
Content-Length: 1163
Connection: close
Date: Wed, 20 May 2020 09:33:49 GMT
Server: tencent-cos
x-cos-request-id: NWVjNGY5N2RfYTdjMjJhMDlfNjZkY18yYWUx****

<WebsiteConfiguration>
<IndexDocument>
<Suffix>index.html</Suffix>
</IndexDocument>
<RedirectAllRequestsTo>
<Protocol>https</Protocol>
</RedirectAllRequestsTo>
<ErrorDocument>
<Key>pages/error.html</Key>
</ErrorDocument>
<RoutingRules>
<RoutingRule>
<Condition>
<HttpErrorCodeReturnedEquals>403</HttpErrorCodeReturnedEquals>
</Condition>
<Redirect>
<Protocol>https</Protocol>
<ReplaceKeyWith>pages/403.html</ReplaceKeyWith>
</Redirect>
</RoutingRule>
```

```
<RoutingRule>
<Condition>
<HttpErrorCodeReturnedEquals>404</HttpErrorCodeReturnedEquals>
</Condition>
<Redirect>
<ReplaceKeyWith>pages/404.html</ReplaceKeyWith>
</Redirect>
</RoutingRule>
<RoutingRule>
<Condition>
<KeyPrefixEquals>assets/</KeyPrefixEquals>
</Condition>
<Redirect>
<ReplaceKeyWith>index.html</ReplaceKeyWith>
</Redirect>
</RoutingRule>
<RoutingRule>
<Condition>
<KeyPrefixEquals>article/</KeyPrefixEquals>
</Condition>
<Redirect>
<Protocol>https</Protocol>
<ReplaceKeyPrefixWith>archived/</ReplaceKeyPrefixWith>
</Redirect>
</RoutingRule>
</RoutingRules>
</WebsiteConfiguration>
```

删除静态网站

最近更新时间: 2025-01-15 17:01:00

功能描述

DELETE Bucket website 请求用于删除存储桶中的静态网站配置。

推荐使用运营端控制台 **平台管理 > 云Api管理 > API Explorer** 进行调试。

> **说明：**
>
> API Explorer 提供了在线调用、签名验证、SDK 代码生成和快速检索接口等能力。您可查看每次调用的请求内容和返回结果以及自动生成 SDK 调用示例。

请求

请求示例

```
DELETE /?website HTTP/1.1
Host: <BucketName-APPID>.cos.<Region>.myqcloud.com
Date: GMT Date
Authorization: Auth String
```

> **说明：**
>
> - `Host: <BucketName-APPID>.cos.<Region>.myqcloud.com`，其中 `<BucketName-APPID>` 为带 APPID 后缀的存储桶名字，例如 examplebucket-1250000000，`<Region>` 为 COS 的可用地域。
> - Authorization: Auth String（详情请参见 请求签名 文档）。

请求参数

此接口无请求参数。

请求头

此接口仅使用公共请求头部，详情请参见 公共请求头部 文档。

请求体

此接口无请求体。

响应

响应头

此接口仅返回公共响应头部，详情请参见 公共响应头部 文档。

响应体

此接口响应体为空。

错误码

此接口遵循统一的错误响应和错误码，详情请参见 错误码 文档。

实际案例

请求

```
DELETE /?website HTTP/1.1
Host: examplebucket-1250000000.cos.ap-beijing.myqcloud.com
Date: Tue, 19 May 2020 07:57:10 GMT
Authorization: q-sign-algorithm=sha1&q-ak=AKID8A0fBVtYFrNm02oY1g1JQQF0c3JO****&q-sign-time=1589875030;1589882230&q-key-time=1589875030;1589882230&q-header-list=date;host&q-url-param-list=website&q-signature=e000543b192f0739b36f420456708fcfb553****
Connection: close
```

响应

```
HTTP/1.1 204 No Content
Connection: close
Date: Tue, 19 May 2020 07:57:10 GMT
Server: tencent-cos
x-cos-request-id: NWVjMzkxNTZfY2ZhZjJhMDlfNWI2OV8yYWFh****
Content-Length: 0
```

版本控制（versioning）

设置版本控制

最近更新时间: 2025-01-15 17:01:00

功能描述

PUT Bucket versioning 接口实现启用或者暂停存储桶的版本控制功能。

细节分析

1. 如果您从未在存储桶上启用过版本控制，则 GET Bucket versioning 请求不返回版本状态值。
2. 开启版本控制功能后，只能暂停，不能关闭。
3. 设置版本控制状态值为 Enabled 或者 Suspended，表示开启版本控制和暂停版本控制。
4. 设置存储桶的版本控制功能，您需要有存储桶的写权限。

请求

请求示例

```
PUT /?versioning HTTP 1.1
Host: <BucketName-APPID>.<Endpoint>
Date: GMT date
Authorization: Auth String
```

> **说明：**
>
> Authorization: Auth String（详情请参见请求签名文档）。

请求头

公共头部

该请求操作的实现使用公共请求头，了解公共请求头详情请参见 公共请求头部 文档。

非公共头部

该请求操作无特殊的请求头部信息。

请求体

```
<VersioningConfiguration>
<Status></Status>
</VersioningConfiguration>
```

具体的数据内容如下：

节点名称（关键字）	父节点	描述	类型
VersioningConfiguration	无	说明版本控制的具体信息	Container
Status	VersioningConfiguration	说明版本是否开启，枚举值：Suspended、Enabled	Enum

响应

响应头

公共响应头

该响应包含公共响应头，了解公共响应头详情请参见 公共响应头部 文档。

特有响应头

该响应无特殊的响应头。

响应体

该响应体为空。

错误分析

以下描述此请求可能会发生的一些特殊的且常见的错误情况。获取更多关于 COS 的错误码的信息，或者产品所有的错误列表，请参见 错误码 。

错误码	HTTP状态码	描述
InvalidArgument	400 Bad Request	如果开启版本控制的 xml body 为空，会返回 InvalidArgument
InvalidDigest	400 Bad Request	1. 携带的 Content-MD5 和服务端计算的请求 body 的不一致 2. 开启版本控制的状态只有 Enabled 和 Suspended 两个合法值，如果写了其他状态，将返回 InvalidArgument

实际案例

请求

```
PUT /?versioning HTTP/1.1
Host: <BucketName-APPID>.<Endpoint>
Connection: keep-alive
Accept-Encoding: gzip, deflate
Accept: */*
User-Agent: python-requests/2.12.4
Content-Type: application/xml
Authorization: q-sign-algorithm=sha1&q-ak=AKID15IsskiBQKTZbAo6WhgcBqVls9Sm****&q-sign-time=1480932292;1981012292&q-key-time=1480932292;1981012292&q-url-param-list=versioning&q-header-list=host&q-signature=47ec2b80c73788ecd394d3b9ad90e120a32f****
Content-Length: 83

<VersioningConfiguration>
<Status>Enabled</Status>
</VersioningConfiguration>
```

响应

```
HTTP/1.1 200 OK
Content-Type: application/xml
Content-Length: 0
Connection: keep-alive
Date: Wed, 23 Aug 2017 08:14:53 GMT
Server: tencent-cos
x-cos-request-id: NTk5ZDM5N2RfMjNiMjM1MGFfMmRiX2Y0****
```

查询版本控制

最近更新时间: 2025-01-15 17:01:00

功能描述

GET Bucket versioning 接口用于实现获得存储桶的版本控制信息。

细节分析

1. 获取存储桶版本控制的状态，需要有该存储桶的读权限。

2. 有三种版本控制状态：未启用版本控制、启用版本控制和暂停版本控制。

- 如果您从未在存储桶上启用（或暂停）版本控制，则响应为：

```
<VersioningConfiguration/>
```

- 如果您启用了存储桶的版本控制功能，则响应为：

```
<VersioningConfiguration>
<Status>Enabled</Status>
</VersioningConfiguration>
```

- 如果您暂停了存储桶的版本控制功能，则响应为：

```
<VersioningConfiguration>
<Status>Suspended</Status>
</VersioningConfiguration>
```

请求

请求示例

```
GET /?versioning HTTP 1.1
Host: <BucketName-APPID>.<Endpoint>
Date: GMT date
Authorization: Auth String
```

> **说明：**
>
> Authorization: Auth String（详情请参见请求签名文档）。

请求头

公共头部

该请求操作的实现使用公共请求头，了解公共请求头详情请参见 公共请求头部 文档。

非公共头部

该请求操作无特殊的请求头部信息。

请求体

该请求的请求体为空。

响应

响应头

公共响应头

该响应包含公共响应头，了解公共响应头详情请参见 公共响应头部 文档。

特有响应头

该响应无特殊的响应头。

响应体

```
<VersioningConfiguration>
<Status></Status>
</VersioningConfiguration>
```

具体的数据内容如下：

节点名称（关键字）	父节点	描述	类型
VersioningConfiguration	无	说明版本控制的具体信息	Container
Status	VersioningConfiguration	说明版本是否开启，枚举值：Suspended、Enabled	Enum

实际案例

请求

```
GET /?versioning HTTP/1.1
Host: <BucketName-APPID>.<Endpoint>
Connection: keep-alive
Accept-Encoding: gzip, deflate
Accept: */*
User-Agent: python-requests/2.12.4
Authorization: q-sign-algorithm=sha1&q-ak=AKID15IsskiBQKTZbAo6WhgcBqVls9Sm****&q-sign-time=1480932292;1981012292&q-key-time=1480932292;1981012292&q-url-param-list=versioning&q-header-list=host&q-signature=5118a936049f9d44482bbb61309235cf4abe****
```

响应

```
HTTP/1.1 200 OK
Content-Type: application/xml
Content-Length: 120
Connection: keep-alive
Date: Wed, 23 Aug 2017 08:15:16 GMT
Server: tencent-cos
x-cos-request-id: NTk5ZDM5OTRfZDNhZDM1MGFfMjYyMTFfZmU3****

<?xml version='1.0' encoding='utf-8' ?>
<VersioningConfiguration>
<Status>Enabled</Status>
</VersioningConfiguration>
```

跨地域复制（replication）
设置跨地域复制

最近更新时间: 2025-01-15 17:01:00

功能描述

PUT Bucket replication 用于向已启用版本控制的存储桶中配置跨地域复制规则。如果存储桶已经配置了跨地域复制规则，那么该请求会替换现有配置。

> **注意：**
>
> 使用该接口时，需确保存储桶已经开启版本控制，开启版本控制的 API 文档请参见 PUT Bucket versioning 接口文档。

请求

请求示例

```
PUT /?replication HTTP/1.1
Host: <BucketName-APPID>.<Endpoint>
Date: GMT Date
Content-MD5: MD5
Authorization: Auth String
request body
```

> **说明：**
>
> Authorization: Auth String（详情请参见请求签名文档）。

请求头

此接口仅使用公共请求头部，详情请参见 公共请求头部 文档。

请求体

用户在请求体中设置跨地域复制的具体配置信息。配置信息包括跨地域复制规则的启用状态、复制内容、目标存储桶的存储桶名和存储区域等信息。对于每一个已启用版本控制的存储桶，COS 目前仅支持一条跨地域复制规则。

```
<ReplicationConfiguration>
<Role>qcs::cam::uin/<OwnerUin>:uin/<SubUin></Role>
<Rule>
<Status></Status>
<ID></ID>
<Prefix></Prefix>
<Destination>
<Bucket>qcs::cos:<Region>::<BucketName-APPID></Bucket>
</Destination>
</Rule>
</ReplicationConfiguration>
```

具体内容描述如下：

节点名称（关键字）	父节点	描述	类型	必选
ReplicationConfiguration	无	说明所有跨地域配置信息	Container	是
Role	ReplicationConfiguration	发起者身份标示：`qcs::cam::uin/:uin/`	String	是
Rule	ReplicationConfiguration	具体配置信息，最多支持1000个	Container	是
ID	ReplicationConfiguration.Rule	用来标注具体 Rule 的名称	String	否
Status	ReplicationConfiguration.Rule	标识 Rule 是否生效，枚举值：Enabled, Disabled	String	是
Prefix	ReplicationConfiguration.Rule	前缀匹配策略，不可重叠，重叠返回错误。前缀匹配根目录为空	String	是
Destination	ReplicationConfiguration.Rule	目标存储桶信息	Container	是

节点名称（关键字）	父节点	描述	类型	必选
Bucket	ReplicationConfiguration.Rule.Destination	资源标识符： `qcs::cos:::`	String	是
StorageClass	ReplicationConfiguration.Rule.Destination	存储级别，枚举值：STANDARD，STANDARD_IA。默认值：原存储桶级别 **注意：** 目前跨地域复制暂不支持将复制后的对象指定为归档存储这一存储类型，如果您需要将对象副本设置为归档存储类型，可在目标存储桶中配置生命周期管理，详细操作请参见 PUT Bucket lifecycle	String	否

响应

响应头

此接口仅返回公共响应头部，详情请参见 公共响应头部 文档。

响应体

该响应体为空。

错误分析

该请求可能会发生的一些常见的特殊错误如下，全部错误信息请参见 错误码 文档。

错误代码	描述	状态码
InvalidBucketState	当前存储桶未开启版本控制，导致无法开启跨地域复制功能	409 Conflict
InvalidArgument	不合法的参数内容	400 Bad Request

实际案例

请求

以下 PUT Bucket replication 请求向存储桶 `originbucket-1250000000` 中添加一条跨地域复制配置。该跨地域复制配置中，指定复制前缀为 `testPrefix` 的对象内容，目标存储桶为广州的 `destinationbucket-1250000000` 。

```
PUT /?replication HTTP/1.1
Date: Mon, 28 Aug 2017 02:53:38 GMT
Authorization: q-sign-algorithm=sha1&q-ak=AKIDZfbOAo7cllgPvF9cXFrJD0a1ICvR****&q-sign-time=1503888878;1503889238&q-key-time=1503888878;1503889238&q-header-list=host&q-url-param-list=replication&q-signature=254bf9cd3d6615e89a36ab652437f9d45c5f****
Content-MD5: AAq9nzrpsz5LJ4UEe1f6Q==
Host: <BucketName-APPID>.<Endpoint>
Content-Length: 312

<ReplicationConfiguration>
<Role>qcs::cam::uin/100000000001:uin/100000000001</Role>
<Rule>
<Status>Enabled</Status>
<ID>RuleId_01</ID>
<Prefix>testPrefix</Prefix>
<Destination>
<Bucket>qcs::cos:ap-guangzhou::destinationbucket-1250000000</Bucket>
</Destination>
</Rule>
</ReplicationConfiguration>
```

响应

上述请求后，COS 返回以下响应，表明当前该跨地域配置已经成功设置完毕。

```
HTTP/1.1 200 OK
Content-Type: application/xml
Content-Length: 0
Connection: keep-alive
```

```
Date: Fri, 14 Apr 2019 07:06:19 GMT
Server: tencent-cos
x-cos-bucket-region: ap-guangzhou
x-cos-request-id: NWQwMzQ3NmJfMjRiMjU4NjRfOTM4NV82ZDU1****
x-cos-trace-id: OGVmYzZiMmQzYjA2OWNhODk0NTRkMTBiOWVmMDAxODc0OWRkZjk0ZDM1NmI1M2E2MTRlY2MzZDhmNmI5MWI1OWE4OGMxZjNjY2JiNTBmMTVmMWY1MzAzYzkyZGQ2ZWM4MzUyZTg1NGRhNWY0NTJiOGUyNTViYzgyNzgxZTEwOTY=
```

查询跨地域复制

最近更新时间: 2025-01-15 17:01:00

功能描述

GET Bucket replication 接口用于查询存储桶中用户跨地域复制配置信息。用户发起该请求时需获得请求签名，表明该请求已获得许可。

请求

请求示例

```
GET /?replication HTTP/1.1
Host: <BucketName-APPID>.<Endpoint>
Date: GMT Date
Authorization: Auth String
```

> **说明：**
>
> Authorization: Auth String（详情请参见请求签名文档）。

请求头

公共头部

该请求操作的实现使用公共请求头，了解公共请求头详情请参见 公共请求头部 文档。

非公共头部

该请求操作无特殊的请求头部信息。

请求体

该请求的请求体为空。

响应

响应头

公共响应头

该响应包含公共响应头，了解公共响应头详情请参见 公共响应头部 文档。

特有响应头

参数项	类型	描述
x-cos-replication-rule-creation-time	UTC 时间戳格式	跨地域复制规则创建时间

响应体

该响应体返回为 **application/xml** 数据，包含完整节点数据的内容展示如下：

```
<ReplicationConfiguration>
<Role>qcs::cam::uin/[UIN]:uin/[Subaccount]</Role>
<Rule>
<Status></Status>
<ID></ID>
<Prefix></Prefix>
<Destination>
<Bucket>qcs::cos:[Region]::[BucketName-APPID]</Bucket>
</Destination>
</Rule>
</ReplicationConfiguration>
```

具体内容描述如下：

节点名称（关键字）	父节点	描述	类型	必选
ReplicationConfiguration	无	说明所有跨地域配置信息	Container	是

节点名称（关键字）	父节点	描述	类型	必选
Role	ReplicationConfiguration	发起者身份标示：`qcs::cam::uin/:uin/`	String	是
Rule	ReplicationConfiguration	具体配置信息，最多支持1000个，所有策略只能指向一个目标存储桶	Container	是
ID	ReplicationConfiguration.Rule	用来标注具体 Rule 的名称	String	否
Status	ReplicationConfiguration.Rule	标识 Rule 是否生效，枚举值：Enabled，Disabled	String	是
Prefix	ReplicationConfiguration.Rule	前缀匹配策略，不可重叠，重叠返回错误，前缀匹配根目录为空	String	是
Destination	ReplicationConfiguration.Rule	目标存储桶信息	Container	是
Bucket	ReplicationConfiguration.Rule.Destination	资源标识符：`qcs::cos:[region]::[bucketname-Appid]`	String	是
StorageClass	ReplicationConfiguration.Rule.Destination	存储级别，枚举值：Standard，Standard_IA，默认值：原存储桶级别	String	否

错误分析

该请求可能会发生的一些常见的特殊错误如下，常见的错误信息请参见 错误码 文档。

错误代码	描述	状态码
ReplicationConfigurationNotFoundError	无查询的跨地域复制规则	404 Not Found

实际案例

请求

下述请求示例展示了从存储桶 originbucket-1250000000 中查询跨地域配置信息。

```
GET /?replication HTTP/1.1
Date: Fri, 14 Apr 2019 07:17:19 GMT
Authorization: q-sign-algorithm=sha1&q-ak=AKIDxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx&q-sign-time=1503895278;1503895638&q-key-time=1503895278;1503895638&q-header-list=host&q-url-param-list=replication&q-signature=f77900be432072b16afd8222b4b349aabd837cb9
Host: <BucketName-APPID>.<Endpoint>
Content-Length: 0
```

响应

上述请求后，COS 返回以下响应，表明当前该存储桶内的跨地域复制配置处于启用状态。该规则配置信息中，复制的内容为存储桶 originbucket-1250000000 内包含 testPrefix 前缀的所有对象。对象副本的存储类型默认跟随源存储桶内对象的存储类型。

```
Content-Type: application/xml
Content-Length: 309
Connection: keep-alive
Date: Fri, 14 Apr 2019 07:17:19 GMT
Server: tencent-cos
x-cos-replication-rule-creation-time: Fri, 14 Apr 2019 07:06:19 GMT
x-cos-request-id: NWQwMzQ5ZmZfMjBiNDU4NjRfNjAwOV84MzA2MjE=
<ReplicationConfiguration>
<Role>qcs::cam::uin/100000000001:uin/100000000001</Role>
<Rule>
<Status>Enabled</Status>
<ID>RuleId_01</ID>
<Prefix>testPrefix</Prefix>
<Destination>
<Bucket>qcs::cos:ap-guangzhou::destinationBucket-1250000000</Bucket>
</Destination>
```

```
</Rule>
</ReplicationConfiguration>
```

删除跨地域复制

最近更新时间: 2025-01-15 17:01:00

功能描述

DELETE Bucket replication 用来删除存储桶中的跨地域复制配置。用户发起该请求时需获得请求签名，表明该请求已获得许可。

请求

请求示例

```
DELETE /?replication HTTP/1.1
Host: <BucketName-APPID>.<Endpoint>
Date: GMT Date
Authorization: Auth String
```

> **说明：**
>
> Authorization: Auth String（详情请参见请求签名文档）。

请求头

公共头部

该请求操作的实现使用公共请求头，了解公共请求头详情请参见 公共请求头部 文档。

非公共头部

该请求操作无特殊的请求头部信息。

请求体

该请求的请求体为空。

响应

响应头

公共响应头

该响应包含公共响应头，了解公共响应头详情请参见 公共响应头部 文档。

特有响应头

该响应无特殊的响应头。

响应体

该响应体为空。

实际案例

请求

下述请求示例展示了从存储桶 originBucet-1250000000 中删除跨地域配置信息。

```
DELETE /?replication HTTP/1.1
Date: Fri, 14 Apr 2019 07:47:35 GMT
Authorization: q-sign-algorithm=sha1&q-ak=AKIDxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx&q-sign-time=1503901499;1503901859&q-key-time=1503901499;1503901859&q-header-list=host&q-url-param-list=replication&q-signature=761f3f6449c6a11684464f4b09c6f292f0a4e7e0
Host: <BucketName-APPID>.<Endpoint>
```

响应

上述请求后，COS 返回 204 No Content 的响应表明已成功删除了该存储桶内的跨地域复制配置。删除跨地域复制配置后，COS 将不再复制源存储桶中的对象到目标存储桶中，目标存储桶中已有的对象数据将被保留。

```
Content-Length: 0
Connection: keep-alive
Date: Fri, 14 Apr 2019 07:47:35 GMT
Server: tencent-cos
x-cos-request-id: NWQwMzUxMTdfMjBiNDU4NjRfNWZlZF84MjdmZTE=
```

```
x-cos-trace-id: OGVmYzZiMmQzYjA2OWNhODk0NTRkMTBiOWVmMDAxODc0OWRkZjk0ZDM1NmI1M2E2MTRlY2MzZDhmNmI5MWI1OWE4OGMxZjNjY2JiNTBmMTVmMWY1MzAzYzkyZGQ2ZWM4MzUyZTg1NGRhNWY0NTJiOGUyNTViYzgyNzgxZTEwOTY=
```

日志管理（logging）
PUT Bucket logging

最近更新时间: 2025-01-15 17:01:00

功能描述

PUT Bucket Logging 接口用于为源存储桶开启日志记录，将源存储桶的访问日志保存到指定的目标存储桶中。

> **注意：**
>
> - 只有源存储桶拥有者才可进行该请求操作。
> - 开启访问日志功能，需要授予日志服务（CLS）产品写入 COS 的权限，授权流程请参见 启用日志管理。

请求

请求示例

```
PUT /?logging HTTP 1.1
Host: <BucketName-APPID>.cos.<Region>.myqcloud.com
Date:date
Content-Length: length
Content-Type: application/xml
Content-MD5: MD5
Authorization: Auth String
```

> **说明：**
>
> - `Host: <BucketName-APPID>.cos.<Region>.myqcloud.com` ，其中 `<BucketName-APPID>` 为带 APPID 后缀的存储桶名字，例如 examplebucket-1250000000； `<Region>` 为 COS 的可用地域。
> - Authorization: Auth String（详情请参见请求签名文档）。

请求头

此接口仅使用公共请求头部，详情请参见 公共请求头部 文档。

请求体

该请求操作的实现需要有请求体。带所有节点的请求体内容示例如下：

```
<BucketLoggingStatus>
<LoggingEnabled>
<TargetBucket>examplebucket-1250000000</TargetBucket>
<TargetPrefix>prefix</TargetPrefix>
</LoggingEnabled>
</BucketLoggingStatus>
```

具体的数据描述如下：

节点名称（关键字）	父节点	描述	类型	是否必选
BucketLoggingStatus	无	说明日志记录配置的状态，如果无子节点信息则意为关闭日志记录	Container	是

Container 节点 BucketLoggingStatus 的内容：

节点名称（关键字）	父节点	描述	类型	是否必选
LoggingEnabled	BucketLoggingStatus	存储桶 logging 设置的具体信息，主要是目标存储桶	Container	否

Container 节点 LoggingEnabled 的内容：

节点名称（关键字）	父节点	描述	类型	是否必选
TargetBucket	LoggingEnabled	存放日志的目标存储桶，可以是同一个存储桶（但不推荐），或同一账户下、同一地域的存储桶	String	否
TargetPrefix	LoggingEnabled	日志存放在目标存储桶的指定路径	String	否

> **说明：**
>
> 用户指定存放日志的存储桶和路径后，生成的日志文件名格式为：`目标存储桶/路径前缀{YYYY}/{MM}/{DD}/{time}_{random}_{index}.gz` 。

响应

响应头

此接口仅返回公共响应头部，详情请参见 公共响应头部 文档。

响应体

该响应体为空。

错误码

此接口遵循统一的错误响应和错误码，详情请参见 错误码 文档。

实际案例

请求

```
PUT /?logging HTTP 1.1
Host: examplebucket-1250000000.cos.ap-beijing.myqcloud.com
Date: Fri, 10 Mar 2017 09:45:46 GMT
Authorization: q-sign-algorithm=sha1&q-ak=AKIDWtTCBYjM5OwLB9CAwA1Qb2ThTSUj****&q-sign-time=1484814927;32557710927&q-key-time=14848
14927;32557710927&q-header-list=host&q-url-param-list=accelerate&q-signature=8b9f05dabce2578f3a79d732386e7cbade90****
Content-Type: application/xml
Content-Length: 147

<BucketLoggingStatus>
<LoggingEnabled>
<TargetBucket>examplebucket-1250000000</TargetBucket>
<TargetPrefix>prefix</TargetPrefix>
</LoggingEnabled>
</BucketLoggingStatus>
```

响应

```
HTTP/1.1 200 OK
Content-Type: application/xml
Content-Length: 0
Connection: keep-alive
Date: Fri, 10 Mar 2017 09:45:46 GMT
Server: tencent-cos
x-cos-request-id: NTg4MDdiZWRfOWExZjRlXzQ2OWVf****
```

GET Bucket logging

最近更新时间: 2025-01-15 17:01:00

功能描述

GET Bucket logging 用于获取源存储桶的日志配置信息。

> **注意：**
>
> 只有源存储桶拥有者才可进行该请求操作。

请求

请求示例

```
GET /?logging HTTP 1.1
Host: <BucketName-APPID>.cos.<Region>.myqcloud.com
Date: GMT Date
Authorization: Auth String
```

> **说明：**
>
> - `Host: <BucketName-APPID>.cos.<Region>.myqcloud.com` ，其中 `<BucketName-APPID>` 为带 APPID 后缀的存储桶名字，例如 examplebucket-1250000000； `<Region>` 为 COS 的可用地域。
> - Authorization: Auth String（详情请参见请求签名文档）。

请求头

此接口仅使用公共请求头部，详情请参见 公共请求头部 文档。

请求体

该请求的请求体为空。

响应

响应头

此接口仅返回公共响应头部，详情请参见 公共响应头部 文档。

响应体

该响应体返回为 **application/xml** 数据，包含完整节点数据的内容展示如下：

```
<BucketLoggingStatus>
<LoggingEnabled>
<TargetBucket>examplebucket-1250000000</TargetBucket>
<TargetPrefix>prefix</TargetPrefix>
</LoggingEnabled>
</BucketLoggingStatus>
```

具体的数据内容如下：

节点名称（关键字）	父节点	描述	类型
BucketLoggingStatus	无	存储桶日志状态信息	Container

Container 节点 BucketLoggingStatus 的内容：

节点名称（关键字）	父节点	描述	类型
LoggingEnabled	BucketLoggingStatus	存储桶日志记录配置详细信息	Container

Container 节点 LoggingEnabled 的内容：

节点名称（关键字）	父节点	描述	类型
TargetBucket	LoggingEnabled	存放日志的目标存储桶，可以是同一个存储桶（但不推荐），或同一账户下、同一地域的存储桶	String
TargetPrefix	LoggingEnabled	日志存放在目标存储桶的指定路径	String

错误码

此接口遵循统一的错误响应和错误码，详情请参见 错误码 文档。

实际案例

请求

```
GET /?logging HTTP 1.1
Host: examplebucket-1250000000.cos.ap-beijing.myqcloud.com
Date: Wed, 28 Oct 2017 21:32:00 GMT
Authorization: q-sign-algorithm=sha1&q-ak=AKIDWtTCBYjM5O****&q-sign-time=1484815944;32557711944&q-key-time=1484815944;32557711944&q-header-list=host&q-url-param-list=accelerate&q-signature=a2d28e1b9023d09f9277982775a4b3b705d0****
```

响应

```
HTTP/1.1 200 OK
Content-Type: application/xml
Content-Length: 142
Connection: keep-alive
Date: Wed, 28 Oct 2017 21:32:00 GMT
Server: tencent-cos
x-cos-request-id: NTg4MDdlNGZfNDYyMDRlXzM0YWFf****

<BucketLoggingStatus>
<LoggingEnabled>
<TargetBucket>examplebucket-1250000000</TargetBucket>
<TargetPrefix>prefix</TargetPrefix>
</LoggingEnabled>
</BucketLoggingStatus>
```

存储桶加密（encryption）
PUT Bucket encryption

最近更新时间: 2025-01-15 17:01:00

功能描述

PUT Bucket encryption 接口用于设置指定存储桶下的默认加密配置。

要执行此接口，必须拥有 PutBucketEncryption 权限。默认情况下，Bucket 的持有者直接拥有权限使用该 API 接口，Bucket 持有者也可以将权限授予其他用户。

请求

请求示例

```
PUT /?encryption HTTP 1.1
Host: <BucketName-APPID>.cos.<Region>.myqcloud.com
Date: GMT Date
Authorization: Auth String
```

> **说明：**
>
> - `Host: <BucketName-APPID>.cos.<Region>.myqcloud.com`，其中 `<BucketName-APPID>` 为带 APPID 后缀的存储桶名字，例如 examplebucket-1250000000，`<Region>` 为 COS 的可用地域。
> - Authorization: Auth String（详情请参见 请求签名 文档）。

请求参数

此接口无请求参数。

请求头

此接口仅使用公共请求头部，详情请参见 公共请求头部 文档。

请求体

用户在请求体中使用 XML 语言设置存储桶默认加密配置信息。加密配置信息主要为加密项。

以下是用于设置 SSE-COS 的请求体：

```
<ServerSideEncryptionConfiguration>
<Rule>
<ApplyServerSideEncryptionByDefault>
<SSEAlgorithm>AES256|KMS</SSEAlgorithm>
<KMSMasterKeyID>String</KMSMasterKeyID>
</ApplyServerSideEncryptionByDefault>
</Rule>
</ServerSideEncryptionConfiguration>
```

具体元素如下：

元素名称	父节点	描述	类型	是否必选
ServerSideEncryptionConfiguration	无	包含默认加密的配置参数	Container	是
Rule	ServerSideEncryptionConfiguration	默认的服务端加密配置规则	Container	是
ApplyServerSideEncryptionByDefault	ServerSideEncryptionConfiguration.Rule	服务端加密的默认配置信息	Container	是
SSEAlgorithm	ServerSideEncryptionConfiguration.Rule.ApplyServerSideEncryptionByDefault	支持枚举值 AES256、KMS，AES256 代表使用 SSE-COS 模式，加密算法为 AES256；KMS 代表 SSE-KMS 模式	String	是

元素名称	父节点	描述	类型	是否必选
KMSMasterKeyID	ServerSideEncryptionConfiguration.Rule.ApplyServerSideEncryptionByDefault	当 SSEAlgorithm 的值为 KMS 时，用于指定 KMS 的用户主密钥 CMK，如不指定，则使用 COS 默认创建的 CMK，更多详细信息可参见 SSE-KMS 加密	String	是

响应

响应头

此接口仅返回公共响应头部，详情请参见 公共响应头部 文档。

响应体

该请求的响应体返回为空。

错误码

此接口遵循统一的错误响应和错误码，详情请参见 错误码 文档。

实际案例

请求

以下示例表示给存储桶 examplebucket-1250000000 设置 SSE-COS 加密。

```
PUT /?encryption HTTP 1.1
Host: examplebucket-1250000000.cos.ap-beijing.myqcloud.com
Date: Mon, 17 Jun 2019 08:37:35 GMT
Authorization: signatureValue



<ServerSideEncryptionConfiguration>
<Rule>
<ApplyServerSideEncryptionByDefault>
<SSEAlgorithm>AES256</SSEAlgorithm>
</ApplyServerSideEncryptionByDefault>
</Rule>
</ServerSideEncryptionConfiguration>
```

响应

```
HTTP/1.1 200 OK
Content-Type: application/xml
Content-Length: 0
Date: Mon, 17 Jun 2019 08:37:36 GMT
Server: tencent-cos
x-cos-request-id: NWQwNzUxNTBfMzdiMDJhMDlfOWM0Nl85NDFk****
```

GET Bucket encryption

最近更新时间: 2025-01-15 17:01:00

功能描述

GET Bucket encryption 接口用于查询指定存储桶下的默认加密配置。

要执行此接口，必须拥有 GetBucketEncryption 权限。默认情况下，Bucket 的持有者直接拥有权限使用该 API 接口，Bucket 持有者也可以将权限授予其他用户。

请求

请求示例

```
GET /?encryption HTTP 1.1
Host: <BucketName-APPID>.cos.<Region>.myqcloud.com
Date: GMT Date
Authorization: Auth String
```

> **说明：**
>
> - `Host: <BucketName-APPID>.cos.<Region>.myqcloud.com` ，其中 `<BucketName-APPID>` 为带 APPID 后缀的存储桶名字，例如 examplebucket-1250000000， `<Region>` 为 COS 的可用地域。
> - Authorization: Auth String（详情请参见 请求签名 文档）。

请求参数

此接口无请求参数。

请求头

此接口仅使用公共请求头部，详情请参见 公共请求头部 文档。

请求体

此接口无请求体。

响应

响应头

此接口仅返回公共响应头部，详情请参见 公共响应头部 文档。

响应体

以下是返回 SSE-COS 加密的响应体：

```
<ServerSideEncryptionConfiguration>
<Rule>
<ApplyServerSideEncryptionByDefault>
<SSEAlgorithm>AES256|KMS</SSEAlgorithm>
<KMSMasterKeyID>String</KMSMasterKeyID>
</ApplyServerSideEncryptionByDefault>
</Rule>
</ServerSideEncryptionConfiguration>
```

具体元素如下：

元素名称	父节点	描述	类型
ServerSideEncryptionConfiguration	无	包含默认加密的配置参数	Container
Rule	ServerSideEncryptionConfiguration	默认的服务端加密配置规则	Container

元素名称	父节点	描述	类型
ApplyServerSideEncryptionByDefault	ServerSideEncryptionConfiguration.Rule	服务端加密的默认配置信息	Container
SSEAlgorithm	ServerSideEncryptionConfiguration.Rule.ApplyServerSideEncryptionByDefault	支持枚举值 AES256、KMS，AES256 代表使用 SSE-COS 模式，加密算法为 AES256；KMS 代表 SSE-KMS 模式	String
KMSMasterKeyID	ServerSideEncryptionConfiguration.Rule.ApplyServerSideEncryptionByDefault	当 SSEAlgorithm 的值为 KMS 时，用于指定 KMS 的用户主密钥 CMK，如不指定，则使用 COS 默认创建的 CMK，更多详细信息可参见 SSE-KMS 加密	String

错误码

此接口遵循统一的错误响应和错误码，详情请参见 错误码 文档。

实际案例

请求

```
GET /?encryption HTTP 1.1
Host: examplebucket-1250000000.cos.ap-beijing.myqcloud.com
Date: Mon, 17 Jun 2019 08:37:35 GMT
Authorization: signatureValue
```

响应

```
HTTP/1.1 200 OK
Content-Type: application/xml
Content-Length: xxxx
Date: Mon, 17 Jun 2019 08:37:36 GMT
Server: tencent-cos
x-cos-request-id: NWQwNzUxNTBfMzdiMDJhMDlfOWM0Nl85NDFk****


<?xml version = "1.0" encoding = "UTF-8">
<ServerSideEncryptionConfiguration>
<Rule>
<ApplyServerSideEncryptionByDefault>
<SSEAlgorithm>AES256</SSEAlgorithm>
</ApplyServerSideEncryptionByDefault>
</Rule>
</ServerSideEncryptionConfiguration>
```

DELETE Bucket encryption

最近更新时间: 2025-01-15 17:01:00

功能描述

DELETE Bucket encryption 接口用于删除指定存储桶下的默认加密配置。

要执行此接口，必须拥有 DeleteBucketEncryption 权限。默认情况下，Bucket 的持有者直接拥有权限使用该 API 接口，Bucket 持有者也可以将权限授予其他用户。

请求

请求示例

```
DELETE /?encryption HTTP 1.1
Host: <BucketName-APPID>.cos.<Region>.myqcloud.com
Date: GMT Date
Authorization: Auth String
```

> 说明：
>
> - `Host: <BucketName-APPID>.cos.<Region>.myqcloud.com` ，其中 `<BucketName-APPID>` 为带 APPID 后缀的存储桶名字，例如 examplebucket-1250000000， `<Region>` 为 COS 的可用地域。
> - Authorization: Auth String（详情请参见 请求签名 文档）。

请求参数

此接口无请求参数。

请求头

此接口仅使用公共请求头部，详情请参见 公共请求头部 文档。

请求体

该请求的请求体为空。

响应

响应头

此接口仅返回公共响应头部，详情请参见 公共响应头部 文档。

响应体

该请求的响应体返回为空。

错误码

此接口遵循统一的错误响应和错误码，详情请参见 错误码 文档。

实际案例

请求

以下示例表示从存储桶 examplebucket-1250000000 中删除默认 SSE-COS 加密配置。

```
DELETE /?encryption HTTP 1.1
Host: examplebucket-1250000000.cos.ap-beijing.myqcloud.com
Date: Mon, 17 Jun 2019 08:37:35 GMT
Authorization: signatureValue
```

响应

```
HTTP/1.1 204 No Content
Server: tencent-cos
Date: Mon, 17 Jun 2019 08:37:36 GMT
x-cos-request-id: NWQwNzUxNTBfMzdiMDJhMDlfOWM0Nl85NDFk****
```

对象锁定（ObjectLock）
PUT Bucket ObjectLockConfiguration

最近更新时间: 2025-01-15 17:01:00

功能描述

COS 支持为已存在的存储桶设置对象锁定。PUT Bucket ObjectLockConfiguration 接口用于为存储桶设置对象锁定功能，以满足合规需求。

请求

请求示例

```
PUT /?object-lock HTTP/1.1
Host: <BucketName-APPID>.cos.<Region>.myqcloud.com
Date: GMT Date
Authorization: Auth String
```

> 说明：
>
> - `Host: <BucketName-APPID>.cos.<Region>.myqcloud.com`，其中 `<BucketName-APPID>` 为带 APPID 后缀的存储桶名字，例如 examplebucket-1250000000，`<Region>` 为 COS 的可用地域。
> - Authorization: Auth String（详情请参见 请求签名 文档）。

请求头

此接口仅使用公共请求头部，详情请参见 公共请求头部 文档。

请求体

```
<?xml version="1.0" encoding="UTF-8" ?>
<ObjectLockConfiguration>
<ObjectLockEnabled>Enabled</ObjectLockEnabled>
<Rule>
<DefaultRetention>
<Days>30</Days>
</DefaultRetention>
</Rule>
</ObjectLockConfiguration>
```

具体数据描述如下：

节点名称（关键字）	父节点	描述	类型	是否必选
ObjectLockConfiguration	无	对象锁定配置	Container	是
ObjectLockEnabled	ObjectLockConfiguration	是否开启对象锁定	String	是
Rule	ObjectLockConfiguration	对象锁定规则	Containers	是
DefaultRetention	ObjectLockConfiguration.Rule	对象锁定默认周期	Containers	是
Days	ObjectLockConfiguration.Rule.DefaultRetention	对象锁定默认周期时长（范围为1-36500）	Int	是

> 注意：
>
> - ObjectLockEnabled 参数仅支持配置为 Enabled，配置为 Enabled 即为该存储桶开启了对象锁定功能。
> - 开启对象锁定功能后不支持关闭，请您谨慎操作。

响应

响应头

此接口仅返回公共响应头部，详情请参见 公共响应头部 文档。

响应体

该响应体为空。

错误码

此接口遵循统一的错误响应和错误码，除了以下错误信息，其他错误码请参见 错误码 文档。

HTTP 状态码	错误码	描述
409 Conflict	InvalidLockedTime	当 Days 天数小于原有时间，会返回报错：存储桶对象锁定时间不能小于原有时间，该值必须在 1 - 36500 天之间

实际案例

请求

以下示例表示对存储桶 examplebucket-1250000000 设置对象锁定，保留期限为1天。

```
PUT /?object-lock= HTTP/1.1
Host: exmaplebucket-1250000000.cos.ap-beijing.myqcloud.com
Content-Length: 281
Content-Type: application/x-www-form-urlencoded
Authorization: Auth String


<ObjectLockConfiguration>
<ObjectLockEnabled>Enabled</ObjectLockEnabled>
<Rule>
<DefaultRetention>
<Days>1</Days>
</DefaultRetention>
</Rule>
</ObjectLockConfiguration>
```

响应

```
HTTP/1.1 200 OK
Content-Length: 0
Connection: keep-alive
Date: Fri, 09 Dec 2022 08:17:20 GMT
Server: tencent-cos
x-cos-request-id: NjM5MmVmMTBfNmM0ZTQ0MGJfMjA4****
```

GET Bucket ObjectLockConfiguration

最近更新时间: 2025-01-15 17:01:00

功能描述

COS 支持为已存在的存储桶设置对象锁定。GET Bucket ObjectLockConfiguration 接口用于获取已生效的对象锁定配置。

请求

请求示例

```
GET /?object-lock HTTP/1.1
Host: <BucketName-APPID>.cos.<Region>.myqcloud.com
Date: GMT Date
Authorization: Auth String
```

> **说明：**
>
> - Host: <BucketName-APPID>.cos.<Region>.myqcloud.com ，其中 <BucketName-APPID> 为带 APPID 后缀的存储桶名字，例如 examplebucket-1250000000， <Region> 为 COS 的可用地域。
> - Authorization: Auth String（详情请参见 请求签名 文档）。

请求头

此接口仅使用公共请求头部，详情请参见 公共请求头部 文档。

请求体

该请求的请求体为空。

响应

响应头

此接口仅返回公共响应头部，详情请参见 公共响应头部 文档。

响应体

```
<?xml version="1.0" encoding="UTF-8" ?>
<ObjectLockConfiguration>
<ObjectLockEnabled>Enabled</ObjectLockEnabled>
<Rule>
<DefaultRetention>
<Days>30</Days>
</DefaultRetention>
</Rule>
</ObjectLockConfiguration>
```

具体数据描述如下：

节点名称（关键字）	父节点	描述	类型
ObjectLockConfiguration	无	对象锁定配置	Container
ObjectLockEnabled	ObjectLockConfiguration	是否开启对象锁定	String
Rule	ObjectLockConfiguration	对象锁定规则	Containers
DefaultRetention	ObjectLockConfiguration.Rule	对象锁定默认周期	Containers
Days	ObjectLockConfiguration.Rule.DefaultRetention	对象锁定默认周期时长（范围为1-36500天）	Int

错误码

此接口遵循统一的错误响应和错误码，详情请参见 错误码 文档。

实际案例

请求

以下示例表示获取存储桶 examplebucket-1250000000 的对象锁定配置。

```
GET /?object-lock= HTTP/1.1
Host: examplebucket-1250000000.cos.ap-beijing.myqcloud.com
Authorization: Auth String
```

响应

```
HTTP/1.1 200 OK
Content-Type: application/xml
Content-Length: 180
Connection: keep-alive
Date: Fri, 09 Dec 2022 08:31:25 GMT
Server: tencent-cos
x-cos-request-id: NjM5MmYyNWNfMzBkMDM4MGJfMmUzNzFfM****

<ObjectLockConfiguration>
<ObjectLockEnabled>Enabled</ObjectLockEnabled>
<Rule>
<DefaultRetention>
<Days>1</Days>
</DefaultRetention>
</Rule>
</ObjectLockConfiguration>
```

GET Object retention

最近更新时间: 2025-01-15 17:01:00

功能描述

该接口用于获取对象锁定的到期日期。

请求

请求示例

```
GET /<object-key>?retention HTTP/1.1
Host: <BucketName-APPID>.cos.<Region>.myqcloud.com
Date: GMT Date
Authorization: Auth String
```

> **说明：**
>
> - `Host: <BucketName-APPID>.cos.<Region>.myqcloud.com`，其中 `<BucketName-APPID>` 为带 APPID 后缀的存储桶名字，例如 examplebucket-1250000000，`<Region>` 为 COS 的可用地域。
> - Authorization: Auth String（详情请参见 请求签名 文档）。

请求头

此接口仅使用公共请求头部，详情请参见 公共请求头部 文档。

请求体

该请求的请求体为空。

响应

响应头

此接口仅返回公共响应头部，详情请参见 公共响应头部 文档。

响应体

```
<?xml version="1.0" encoding="UTF-8" ?>
<Retention>
<RetainUntilDate>timestamp</RetainUntilDate>
</Retention>
```

具体数据描述如下：

节点名称（关键字）	父节点	描述	类型
Retention	无	周期	Container
RetainUntilDate	Retention	具体到期日期	String Date

错误码

此接口遵循统一的错误响应和错误码，详情请参见 错误码 文档。

实际案例

请求

```
GET /temp.txt?retention HTTP/1.1
Host: examplebucket-1250000000.cos.ap-beijing.myqcloud.com
```

```
Authorization: Auth String
```

响应

```
HTTP/1.1 200 OK
Content-Type: application/xml
Content-Length: 87
Connection: keep-alive
Date: Fri, 09 Dec 2022 08:35:49 GMT
Server: tencent-cos
x-cos-request-id: NjM5MmYzNjVfMjBkMDM4MGJfMWM2ND****

<Retention>
<RetainUntilDate>2022-12-10T08:34:48.000Z</RetainUntilDate>
</Retention>
```

Object接口
基本操作
简单上传对象

最近更新时间: 2025-01-15 17:01:00

功能描述

PUT Object 接口请求可以将本地的对象（Object）上传至指定存储桶中。该操作需要请求者对存储桶有写入权限。

细节分析

1. 需要有 Bucket 的写权限。
2. 如果请求头的 Content-Length 值小于实际请求体（body）中传输的数据长度，COS 仍将成功创建文件，但 Object 大小只等于 Content-Length 中定义的大小，其他数据将被丢弃。
3. 如果试图添加的 Object 的同名文件已经存在，那么新上传的文件，将覆盖原来的文件，成功时返回200 OK。

请求

请求示例

```
PUT /<ObjectKey> HTTP/1.1
Host: <BucketName-APPID>.<Endpoint>
Date: GMT Date
Authorization: Auth String
```

> **说明：**
>
> Authorization: Auth String（详请请参阅请求签名文档）。

请求头

公共头部

该请求操作的实现使用公共请求头，了解公共请求头详情请参阅 公共请求头部 文档。

非公共头部

该操作的实现还可以使用以下请求头。

名称	描述	类型	必选
Content-Disposition	RFC 2616 中定义的文件名称，将作为 Object 元数据保存	string	否
Content-Encoding	RFC 2616 中定义的编码格式，将作为 Object 元数据保存	string	否
Expect	当使用 Expect：100-continue 时，在收到服务端确认后，才会发送请求内容	string	否
Expires	RFC 2616 中定义的缓存策略，将作为 Object 元数据保存	string	否
x-cos-meta-*	包括用户自定义头部后缀和用户自定义头部信息，将作为 Object 元数据返回，大小限制为2KB **注意**：用户自定义头部信息支持下划线，但用户自定义头部后缀不支持下划线	string	否
x-cos-storage-class	设置 Object 的存储级别，枚举值：STANDARD，STANDARD_IA，ARCHIVE。默认值：STANDARD	string	否
x-cos-acl	定义 Object 的 ACL 属性，有效值：private，public-read，default；默认值：default（继承 Bucket 权限） **注意**：当前访问策略条目限制为1000条，如果您不需要进行 Object ACL 控制，请填 default 或者此项不进行设置，默认继承 Bucket 权限	string	否
x-cos-grant-read	赋予被授权者读的权限，格式：x-cos-grant-read: id="[OwnerUin]"	String	否
x-cos-grant-full-control	赋予被授权者所有的权限，格式：x-cos-grant-full-control: id="[OwnerUin]"	String	否

服务端加密（SSE）相关头部

在上传对象时可以使用服务端加密，请参见 [服务端加密专用头部]。

请求体

该请求的请求体为 Object 文件内容。

响应

响应头

公共响应头

该响应包含公共响应头，了解公共响应头详情请参阅 公共响应头部 文档。

特有响应头

该请求操作的响应头具体数据为：

名称	类型	描述
ETag	string	上传文件内容的 MD5 值

版本控制相关头部

在启用版本控制的存储桶中上传对象，将返回下列响应头部：

名称	描述	类型
x-cos-version-id	对象的版本 ID	string

服务端加密（SSE）相关头部

如果在上传对象时使用了服务端加密，则此接口将返回服务端加密专用头部，请参见 [服务端加密专用头部]。

响应体

该请求响应体为空。

错误码

该请求操作无特殊错误信息，常见的错误信息请参见 错误码 文档。

实际案例

请求

```
PUT /picture.jpg HTTP/1.1
Host: <BucketName-APPID>.<Endpoint>
Date: Wed, 28 Oct 2015 20:32:00 GMT
Authorization:q-sign-algorithm=sha1&q-ak=AKIDxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx&q-sign-time=1484639384;32557535384&q-key-time=1484639384;32557535384&q-header-list=host&q-url-param-list=&q-signature=5c07b7c67d56497d9aacb1adc19963135b7d00dc
Content-Length: 64

[Object]
```

响应

```
HTTP/1.1 200 OK
Content-Type: application/xml
Content-Length: 0
Date: Wed,16 Aug 2017 11: 59: 33 GMT
Server: tencent-cos
x-cos-request-id: NTk5NDMzYTRfMjQ4OGY3Xzc3NGRfMWY=
```

案例：使用服务端加密 SSE-C

请求

```
PUT /exampleobject HTTP/1.1
Host: examplebucket-1250000000.cos.ap-beijing.myqcloud.com
Date: Fri, 10 Apr 2020 09:36:12 GMT
Content-Type: image/jpeg
x-cos-server-side-encryption-customer-algorithm: AES256
x-cos-server-side-encryption-customer-key: MDEyMzQ1Njc4OUFCQ0RFRjAxMjM0NTY3ODlBQkNERUY=
x-cos-server-side-encryption-customer-key-MD5: U5L61r7jcwdNvT7frmUG8g==
Content-Length: 16
Content-MD5: 7o3pGNBWQBRbGPcPTDqmAg==
Authorization: q-sign-algorithm=sha1&q-ak=AKID8A0fBVtYFrNm02oY1g1JQQF0c3JO****&q-sign-time=1586511372;1586518572&q-key-time=1586511372;1586518572&q-header-list=content-length;content-md5;content-type;date;host;x-cos-server-side-encryption-customer-algorithm;x-cos-server-side-encryption-customer-key;x-cos-server-side-encryption-customer-key-md5&q-url-param-list=&q-signature=4f6f9f0a6700930f70bff31e3a2b2e622711****
Connection: close



[Object Content]
```

响应

```
HTTP/1.1 200 OK
Content-Length: 0
Connection: close
Date: Fri, 10 Apr 2020 09:36:13 GMT
ETag: "582d9105f71525f3c161984bc005efb5"
Server: tencent-cos
x-cos-hash-crc64ecma: 16749656579157681890
x-cos-request-id: NWU5MDNlMGNfZTFjODJhMDlfMzVlMDFfZTk1****
x-cos-server-side-encryption-customer-algorithm: AES256
x-cos-server-side-encryption-customer-key-MD5: U5L61r7jcwdNvT7frmUG8g==
```

设置对象复制

最近更新时间: 2025-01-15 17:01:00

功能描述

PUT Object - Copy 请求实现将一个文件从源路径复制到目标路径。建议文件大小1M 到 5G，超过5G 的文件请使用分块上传 Upload - Copy。在拷贝的过程中，文件元属性和 acl 可以被修改。 用户可以通过该接口实现文件移动，文件重命名，修改文件属性和创建副本。

请求

请求示例

```
PUT /<ObjectKey> HTTP/1.1
Host: <BucketName-APPID>.<Endpoint>
Date: GMT Date
Authorization: Auth String
x-cos-copy-source: <BucketName-APPID>.<Endpoint>/filepath
```

> **说明：**
>
> Authorization: Auth String（详细请参阅请求签名章节）。

请求头

公共头部

该请求操作的实现使用公共请求头，了解公共请求头详情请参阅 公共请求头部 章节。

非公共头部

名称	描述	类型	必选
x-cos-copy-source	源文件 URL 路径，可以通过 versionid 子资源指定历史版本	string	是
x-cos-metadata-directive	是否拷贝源文件的元数据，枚举值：Copy, Replaced，默认值 Copy。假如标记为 Copy，则拷贝源文件的元数据；假如标记为 Replaced，则按本次请求的 Header 信息修改元数据。当目标路径和源路径一致，即用户试图修改元数据时，则标记必须为 Replaced	string	否
x-cos-copy-source-If-Modified-Since	当 Object 在指定时间后被修改，则执行操作，否则返回412。可与 x-cos-copy-source-If-None-Match 一起使用，与其他条件联合使用返回冲突	string	否
x-cos-copy-source-If-Unmodified-Since	当 Object 在指定时间后未被修改，则执行操作，否则返回412。可与 x-cos-copy-source-If-Match 一起使用，与其他条件联合使用返回冲突	string	否
x-cos-copy-source-If-Match	当 Object 的 Etag 和给定一致时，则执行操作，否则返回412。可与 x-cos-copy-source-If-Unmodified-Since 一起使用，与其他条件联合使用返回冲突	string	否
x-cos-copy-source-If-None-Match	当 Object 的 Etag 和给定不一致时，则执行操作，否则返回412。可与 x-cos-copy-source-If-Modified-Since 一起使用，与其他条件联合使用返回冲突	string	否
x-cos-acl	定义 Object 的 ACL 属性。有效值：private，public-read；默认值：private	string	否
x-cos-grant-read	赋予被授权者读的权限。格式：x-cos-grant-read: id="[OwnerUin]"	string	否
x-cos-grant-write	赋予被授权者写的权限。格式：x-cos-grant-write: id="[OwnerUin]"	string	否
x-cos-grant-full-control	赋予被授权者所有的权限。格式：x-cos-grant-full-control: id="[OwnerUin]"	string	否
x-cos-meta-*	包括用户自定义头部后缀和用户自定义头部信息，将作为 Object 元数据返回，大小限制为2KB。 **注意：**用户自定义头部信息支持下划线，但用户自定义头部后缀不支持下划线	string	否

请求体

该请求的请求体为空。

响应

响应头

公共响应头

该响应包含公共响应头，了解公共响应头详情请参阅 公共响应头部 文档。

响应体

该响应体返回为 **application/xml** 数据，包含完整节点数据的内容展示如下：

```
<CopyObjectResult>
<ETag>"ba82b57cfdfda8bd17ad4e5879ebb4fe"</ETag>
<LastModified>2017-08-04T02:41:45</LastModified>
</CopyObjectResult>
```

具体的数据内容如下：

名称	描述	类型
CopyObjectResult	返回复制结果信息	String
ETag	返回文件的 MD5 算法校验值。ETag 的值可以用于检查 Object 的内容是否发生变化。	String
LastModified	返回文件最后修改时间，GMT 格式	String

实际案例

请求

```
PUT /exampleobject HTTP/1.1
Host: <BucketName-APPID>.<Endpoint>
Date: Fri, 04 Aug 2017 02:41:45 GMT
Connection: keep-alive Accept-Encoding: gzip, deflate Accept: */*
User-Agent: python-requests/2.12.4
Authorization: q-sign-algorithm=sha1&q-ak=AKIDxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx&q-sign-time=1480932292;1981012292&q-key-time=1480932292;1981012292&q-url-param-list=&q-header-list=host&q-signature=eacefe8e2a0dc8a18741d9a29707b1dfa5aa47cc
x-cos-copy-source: sourcebucket-1250000001.<Endpoint>/picture.jpg
Content-Length: 0
```

响应

```
HTTP/1.1 200 OK
Content-Type: application/xml
Content-Length: 133
Connection: keep-alive
Date: Fri, 04 Aug 2017 02:41:45 GMT
Server: tencent-cos
x-cos-request-id: NTk4M2RlZTlfZDRiMDM1MGFfYTA1ZV8xMzNlYw==

<CopyObjectResult>
<ETag>"ba82b57cfdfda8bd17ad4e5879ebb4fe"</ETag>
<LastModified>2017-08-04T02:41:45</LastModified>
</CopyObjectResult>
```

表单上传对象

最近更新时间: 2025-01-15 17:01:00

功能描述

POST Object 接口请求允许使用者用表单的形式将文件（Object）上传至指定 Bucket 中。该操作需要请求者对 Bucket 有 WRITE 权限。所有由 HTTP 头部携带的 API 参数，都使用表单字段请求。

细节分析

1. 需要有 Bucket 的写权限。
2. 如果试图添加的 Object 的同名文件已经存在，那么新上传的文件，将覆盖原来的文件，成功时返回200 OK。

请求

请求示例

```
POST / HTTP/1.1
Host: <BucketName-APPID>.<Endpoint>
Content-Length: length
Headers
Form
```

请求头

公共头部

该请求操作的实现使用公共请求头，了解公共请求头详情，请参阅 公共请求头部 文档。

非公共头部

该请求操作需要用到如下必选请求头：

名称	描述	类型	必选
Content-Length	RFC 2616中定义的 HTTP 请求内容长度（字节）	String	是

表单字段

名称	描述	类型	必选
acl	定义 Object 的 ACL 属性， 有效值：private，public-read，default； 默认值：default(继承 Bucket 权限)； 注意：当前访问策略条目限制为1000条，如果您不需要进行 Object ACL 控制，请填 default 或者此项不进行设置，默认继承 Bucket 权限	String	否
Cache-Control, Content-Type, Content-Disposition, Content-Encoding, Expires	RFC 2616 中定义的头部，详见 文档	String	否
file	文件内容，作为表单的最后一个字段	String	是
key	上传后的文件名，使用 ${filename} 则会进行替换，例如a/b/${filename}，上传文件 photo.jpg，那么最终的上传路径就是 a/b/photo.jpg	String	是
x-cos-meta-*	包括用户自定义头部后缀和用户自定义头部信息，将作为 Object 元数据返回，大小限制为2KB。 注意：用户自定义头部信息支持下划线，但用户自定义头部后缀不支持下划线	String	否
policy	Base64 编码。用于做请求检查，如果请求的内容和 Policy 指定的条件不符，返回 403 AccessDenied	String	否

签名保护

当发起一个表单上传 HTTP POST 请求需要签名保护时，表单需要包含以下的内容结构 form-data 的内容：

表单字段	描述
policy	经过 Base64 加密的策略内容，策略内容将用于对请求内容的检查。如果请求内容与策略指定条件不符，则请求将被拒绝。
q-sign-algorithm	用于计算签名的算法，COS目前支持SHA1，此处填写小写 `sha1`。
q-ak	账户密钥 ID，即 SecretId
q-key-time	用于请求签名的密钥有效起止时间，通过 Unix 时间戳描述起始和结束时间，以秒为单位，格式为 [start-seconds];[end-seconds]。例如`1480932292;1481012298`
q-signature	使用上述元素计算的请求签名，COS 将会使用表单元素与签名内容做校验，若签名与所签内容不一致，则请求将被拒绝

签名计算

签名 q-signature 的计算分为三个步骤：

1. 使用密钥内容对 q-key-time 的时间加密计算值 SignKey。
2. 创建一个 POST 请求 policy 并将内容进行 sha1 加密，得到 StringToSign。
3. 使用 SignKey 对 StringToSign 进行加密，生成签名 Signature。

Policy

以下是一个完整的 policy 示例：

```
{ "expiration": "2007-12-01T12:00:00.000Z",
"conditions": [
{"acl": "public-read" },
{"bucket": "examplebucket-1250000000" },
["starts-with", "$key", "user/eric/"],
{"q-sign-algorithm": "sha1" },
{"q-ak": "AKIDxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx" },
{"q-sign-time": "1480932292;1481012298" }
]
}
```

Expiration 设置该 POST Policy 的超时时间，使用 ISO8601 GMT 时间，例如 2017-12-01T12:00:00.000Z。

Conditions 规则

类型	描述
完全匹配	使用`{"key": "value"}`或`["eq", "$key", "value"]`方式表达
前缀匹配	使用`["starts-with", "$key", "value"]`方式表达，value 可留空
范围匹配	仅用于`["content-length-range", int1, int2]`则文件字节数必须在 int1 和 int2 范围内

Conditions 参数 所有参数均为非必选，不填可以不校验。

名称	描述	匹配方式
acl	文件 ACL 属性的许可范围，可不填	完全、前缀
bucket	指定上传的 Bucket	完全
content-length-range	指定文件的上传大小范围	范围
key	对象的存储路径	完全、前缀
x-cos-meta-*	包括用户自定义头部后缀和用户自定义头部信息，将作为 Object 元数据返回，大小限制为 2KB **注意：**用户自定义头部信息支持下划线，但用户自定义头部后缀不支持下划线	完全、前缀
x-cos-*	其他需要签署的 COS 头部	完全

响应

响应头

公共响应头

该响应使用公共响应头，了解公共响应头详情，请参阅 公共响应头部 文档。

响应参数

名称	描述	类型
ETag	返回文件的 MD5 算法校验值。ETag 的值可以用于检查 Object 在上传过程中是否有损坏	String

响应体

该响应体返回为 application/xml 数据，包含完整节点数据的内容展示如下：

```
<PostResponse>
<Location>http://imgcache.finance.cloud.tencent.com:80examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/photo.jpg</Location>
<Bucket>examplebucket-1250000000</Bucket>
<Key>photo.jpg</Key>
<ETag>d41d8cd98f00b204e9800998ecf8427e</ETag>
</PostResponse>
```

具体的数据描述如下：

节点名称（关键字）	父节点	描述	类型	必选
PostResponse	无	保存 POST Object 结果的容器	Container	是

Container 节点 PostResponse 的内容：

节点名称（关键字）	父节点	描述	类型	必选
Location	PostResponse	对象的完整路径	String	是
Bucket	PostResponse	对象所在的存储桶	String	是
Key	PostResponse	对象 key 名	String	是
ETag	PostResponse	Etag 内容	String	是

错误码

以下描述此请求可能会发生的一些特殊的且常见的错误情况。关于 COS 更多的错误码信息，请查阅 错误码 文档。

错误码	HTTP 状态码	描述
InvalidDigest	400 Bad Request	如果用户上传文件时携带 Content-MD5 头部，COS 会校验 body 的 Md5 和用户携带的 MD5 是否一致，如果不一致将返回 InvalidDigest
KeyTooLong	400 Bad Request	上传文件时携带的以 x-cos-meta 开头的自定义头部，每个自定义头部的 key 和 value 加起来不能超过4k，否则返回 KeyTooLong 错误
MissingContentLength	411 Length Required	如果上传文件时，没有添加 Content-Length 头部，会返回该错误码
NoSuchBucket	404 Not Found	如果试图添加的 Object 所在的 Bucket 不存在，返回 404 Not Found 错误，错误码：NoSuchBucket
EntityTooLarge	400 Bad Request	如果添加的文件长度超过5G，会返回 EntityTooLarge，并返回错误信息`"Your proposed upload exceeds the maximum allowed object size"`
InvalidURI	400 Bad Request	对象 key 长度限制为850，如果超过850会返回 InvalidURI

实际案例

请求

```
POST / HTTP/1.1
Connection: keep-alive
```

```
Accept-Encoding: gzip, deflate
Accept: */*
User-Agent: python-requests/2.12.4
Host: <BucketName-APPID>.<Endpoint>
Content-Length: 1352
Content-Type: multipart/form-data; boundary=e07f2a7876ae4755ae18d300807ad879

--e07f2a7876ae4755ae18d300807ad879
Content-Disposition: form-data; name="key"

a/${filename}
--e07f2a7876ae4755ae18d300807ad879
Content-Disposition: form-data; name="Acl"

public-read
--e07f2a7876ae4755ae18d300807ad879
Content-Disposition: form-data; name="Signature"

q-sign-algorithm=sha1&q-ak=AKIDxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx&q-sign-time=1512983814;1512984814&q-key-time=1512983814;1512984814&q-url-param-list=&q-header-list=host&q-signature=2ffd2ae714e7445a8da000ec5d51771ff5056500
--e07f2a7876ae4755ae18d300807ad879
Content-Disposition: form-data; name="policy"

eyJjb25kaXRpb25zIjogW3siYnVja2V0IjogImtpdG1hbnMzdGVzdDEifSwgWyJjb250ZW50LWxlbmd0aC1yYW5nZSIsIDAsIDEwMDAwMDAwXSwgWyJzdGFydHMtd2l0aCIsICJ4LWNvcy1tZXRhLWJiIiwgIjEyIl1dLCAiZXhwaXJhdGlvbiI6ICIyMDQ3LTEyLTAxVDEyOjAwOjAwLjAwMFoifQ==
--e07f2a7876ae4755ae18d300807ad879
Content-Disposition: form-data; name="x-Cos-meta-bb"

124
--e07f2a7876ae4755ae18d300807ad879
Content-Disposition: form-data; name="key1"

1
--e07f2a7876ae4755ae18d300807ad879
Content-Disposition: form-data; name="file"; filename="empty:a"


--e07f2a7876ae4755ae18d300807ad879--
```

响应

```
HTTP/1.1 204
Content-Type: application/xml
Content-Length: 232
Connection: keep-alive
Date: Mon, 11 Dec 2017 09:16:56 GMT
ETag: "d41d8cd98f00b204e9800998ecf8427e"
Location: http://imgcache.finance.cloud.tencent.com:80examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/photo.jpg
Server: tencent-cos
x-cos-request-id: NWEyZTRkMDZfMjQ4OGY3MGFfNTE4Yl81
```

下载对象

最近更新时间: 2025-01-15 17:01:00

功能描述

GET Object 接口请求可以在 COS 的存储桶中将一个文件（对象）下载至本地。该操作需要请求者对目标对象具有读权限或目标对象对所有人都开放了读权限（公有读）。

请求

请求示例：

```
GET /<ObjectName> HTTP/1.1
Host: <BucketName-APPID>.<Endpoint>
Date: GMT Date
Authorization: Auth String
```

> **说明：**
>
> Authorization: Auth String (详细参见请求签名章节)。

请求行

```
GET /{ObjectName} HTTP/1.1
```

该 API 接口接受 `GET` 请求。

请求参数

名称	类型	必选	描述
response-content-type	string	否	设置响应头部中的 Content-Type 参数
response-content-language	string	否	设置响应头部中的 Content-Language 参数
response-expires	string	否	设置响应头部中的 Content-Expires 参数
response-cache-control	string	否	设置响应头部中的 Cache-Control 参数
response-content-disposition	string	否	设置响应头部中的 Content-Disposition 参数
response-content-encoding	string	否	设置响应头部中的 Content-Encoding 参数
versionId	string	否	当启用版本控制时，指定要下载的版本 ID，如不指定则下载对象的最新版本

请求头

公共头部

该请求操作的实现使用公共请求头，了解公共请求头详细请参见 公共请求头部 章节。

非公共头部

名称	类型	必选	描述
Range	string	否	RFC 2616 中定义的指定文件下载范围，以字节（bytes）为单位
If-Unmodified-Since	string	否	如果文件修改时间早于或等于指定时间，才返回文件内容。否则返回 412 (precondition failed)
If-Modified-Since	string	否	当 Object 在指定时间后被修改，则返回对应 Object meta 信息，否则返回 304(not modified)
If-Match	string	否	当 ETag 与指定的内容一致，才返回文件。否则返回 412 (precondition failed)
If-None-Match	string	否	当 ETag 与指定的内容不一致，才返回文件。否则返回 304 (not modified)

请求体

该请求请求体为空。

响应

响应头

此接口除返回公共响应头部外，还返回以下响应头部，了解公共响应头部详情请参见 公共响应头部 文档。

名称	描述	类型
Cache-Control	RFC 2616 中定义的缓存指令，仅当对象元数据包含此项或通过请求参数指定了此项时才会返回该头部	string
Content-Disposition	RFC 2616 中定义的文件名称，仅当对象元数据包含此项或通过请求参数指定了此项时才会返回该头部	string
Content-Encoding	RFC 2616 中定义的编码格式，仅当对象元数据包含此项或通过请求参数指定了此项时才会返回该头部	string
Content-Range	RFC 2616 中定义的返回内容的字节范围，仅当请求中指定了 Range 请求头部时才会返回该头部	string
Expires	RFC 2616 中定义的缓存失效时间，仅当对象元数据包含此项或通过请求参数指定了此项时才会返回该头部	string
x-cos-meta-*	包括用户自定义元数据头部后缀和用户自定义元数据信息	string
x-cos-storage-class	对象存储类型，例如 STANDARD_IA，ARCHIVE。仅当对象不是标准存储（STANDARD）时才会返回该头部	Enum

版本控制相关头部

启用版本控制的存储桶内的对象将返回下列响应头部：

名称	描述	类型
x-cos-version-id	对象的版本 ID	string

响应体

下载成功，文件内容在响应体中。

错误码

错误码	描述	HHTP 状态码
InvalidArgument	提供的参数错误	400 Bad Request
SignatureDoesNotMatch	提供的签名不符合规则，返回该错误码	403 Forbidden
NoSuchKey	如果下载的文件不存在，返回该错误码	404 Not Found

实际案例

请求一

```
GET /123 HTTP/1.1
Host: <BucketName-APPID>.<Endpoint>
Date: Wed, 28 Oct 2014 22:32:00 GMT
Authorization: q-sign-algorithm=sha1&q-ak=AKIDxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx&q-sign-time=1484212200;32557108200&q-key-time=1484212200;32557108200&q-header-list=host&q-url-param-list=&q-signature=11522aa3346819b7e5e841507d5b7f156f34e639
```

响应一

```
HTTP/1.1 200 OK
Date: Wed, 28 Oct 2014 22:32:00 GMT
Content-Type: application/octet-stream
Content-Length: 16087
Connection: keep-alive
Accept-Ranges: bytes
Content-Disposition: attachment; filename=\"filename.jpg\"
```

```
Content-Range: bytes 0-16086/16087
ETag: \"9a4802d5c99dafe1c04da0a8e7e166bf\"
Last-Modified: Wed, 28 Oct 2014 20:30:00 GMT
x-cos-request-id: NTg3NzQ3ZmVfYmRjMzVfMzE5N182NzczMQ==

[Object]
```

请求二

携带 response-xxx 参数

```
GET /123?response-content-type=application%2fxml HTTP/1.1
Host: <BucketName-APPID>.<Endpoint>
Date: Wed, 28 Oct 2014 22:32:00 GMT
Authorization: q-sign-algorithm=sha1&q-ak=AKIDxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx&q-sign-time=1484212200;32557108200&q-key-time=1484212200;3
2557108200&q-header-list=host&q-url-param-list=&q-signature=11522aa3346819b7e5e841507d5b7f156f34e639
```

响应二

```
HTTP/1.1 200 OK
Date: Wed, 28 Oct 2014 22:32:00 GMT
Content-Type: application/xml
Content-Length: 16087
Connection: keep-alive
Accept-Ranges: bytes
Content-Disposition: attachment; filename=\"filename.jpg\"
Content-Range: bytes 0-16086/16087
ETag: \"9a4802d5c99dafe1c04da0a8e7e166bf\"
Last-Modified: Wed, 28 Oct 2014 20:30:00 GMT
x-cos-request-id: NTg3NzQ3ZmVfYmRjMzVfMzE5N182NzczMQ==

[Object]
```

版权所有：腾讯云金融专区

查询对象元数据

最近更新时间: 2025-01-15 17:01:00

功能描述

HEAD Object 接口请求可以获取对应 Object 的 meta 信息数据，HEAD 的权限与 GET 的权限一致。

请求

请求示例：

```
HEAD /<ObjectKey> HTTP/1.1
Host: <BucketName-APPID>.<Endpoint>
Date: GMT Date
Authorization: Auth String
```

> **说明：**
>
> Authorization: Auth String（详情请参阅请求签名文档）。

请求头

公共头部

该请求操作的实现使用公共请求头，了解公共请求头详情请参阅 公共请求头部 文档。

非公共头部

名称	类型	必选	描述
If-Modified-Since	string	否	当 Object 在指定时间后被修改，则返回对应 Object 的 meta 信息，否则返回304

请求体

该请求请求体为空。

响应

响应头

公共响应头

该响应使用公共响应头，了解公共响应头详情请参阅 公共响应头部 文档。

特有响应头

该请求操作的响应头具体数据为：

名称	类型	描述
x-cos-meta- *	string	用户自定义的 meta

响应体

该请求响应体为空。

实际案例

请求

```
HEAD /exampleobject HTTP/1.1
Host: <BucketName-APPID>.<Endpoint>
```

```
Date: Thu, 12 Jan 2017 17:26:53 GMT
Authorization: q-sign-algorithm=sha1&q-ak=AKIDxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx&q-sign-time=1484213210;32557109210&q-key-time=1484213210;3
2557109210&q-header-list=host&q-url-param-list=&q-signature=ac61b8eb61964e7e6b935e89de163a479a25c210
```

响应

```
HTTP/1.1 200 OK
Content-Type: application/xml
Content-Length: 16087
Connection: keep-alive
Date: Thu, 12 Jan 2017 17:26:53 GMT
ETag: \"9a4802d5c99dafe1c04da0a8e7e166bf\"
Last-Modified: Wed, 11 Jan 2017 07:30:07 GMT
Server: tencent-cos
x-cos-request-id: NTg3NzRiZGRfYmRjMzVfM2Y2OF81N2YzNA==
```

删除单个对象

最近更新时间: 2025-01-15 17:01:00

功能描述

Delete Object 接口请求可以在 COS 的 Bucket 中将一个文件（Object）删除。该操作需要请求者对 Bucket 有 WRITE 权限。

细节分析

1. 在 Delete Object 请求中删除一个不存在的 Object，仍然认为是成功的，返回 `204 No Content`。
2. Delete Object 要求用户对该 Object 要有写权限。

版本控制

如需删除对象的指定版本（包括删除标记，下同），请使用 versionId 请求参数指定对应的版本 ID（包括删除标记的版本 ID，下同），此时响应将返回 x-cos-version-id 响应头部，代表该请求操作删除的版本 ID。

如未指定 versionId 请求参数：

- 当版本控制为启用时，该 DELETE 操作将创建一个删除标记作为指定对象的最新版本，此时响应将返回 x-cos-version-id 响应头部，代表该请求操作创建的删除标记的版本 ID。
- 当版本控制为暂停时，该 DELETE 操作将创建一个版本 ID 为 null 的删除标记作为指定对象的最新版本，同时删除任何已存在的版本 ID 为 null 的其他版本（如有）。

当该 DELETE 操作创建或删除了删除标记，那么将返回 x-cos-delete-marker: true 响应头部，代表该 DELETE 操作创建或删除了指定对象的删除标记。

有关版本控制的启用或暂停状态说明，请参见 版本控制概述。

请求

语法示例：

```
DELETE /ObjectName HTTP/1.1
Host: <BucketName-APPID>.<Endpoint>
Date: GMT Date
Content-Length: length
Authorization: Auth String
```

> **说明：**
>
> Authorization: Auth String (详细参见请求签名章节)。

请求行

```
DELETE /ObjectName HTTP/1.1
```

该 API 接口接受 DELETE 请求。

请求参数

名称	描述	类型	是否必选
versionId	指定要删除的版本 ID	string	否

请求头

公共头部

该请求操作的实现使用公共请求头,了解公共请求头详细请参见 公共请求头部 章节。

非公共头部

该请求操作无特殊的请求头部信息。

请求体

该请求的请求体空。

响应

响应头

公共响应头

该响应使用公共响应头,了解公共响应头详细请参见 公共响应头部 章节。

特有响应头

该请求操作无特殊的响应头。

版本控制相关头部

删除启用版本控制的存储桶内的对象或对象的指定版本将返回下列响应头部：

名称	描述	类型
x-cos-version-id	对象的版本 ID 或删除标记的版本 ID	string
x-cos-delete-marker	• 当使用 versionId 请求参数指定删除标记的版本 ID 时，将返回此响应头部且值为 true，代表删除的版本 ID 对应的是一个删除标记 • 当未使用 versionId 请求参数，且指定对象所在的存储桶启用了版本控制时，将返回此响应头部且值为 true，代表该删除请求创建了一个删除标记作为对象的最新版本	boolean

响应体

该请求的响应体为空。

错误分析

以下描述此请求可能会发生的一些特殊的且常见的错误情况：

错误码	HTTP状态码	描述
NoSuchBucket	404 Not Found	Bucket 不存在

获取更多关于COS的错误码的信息，或者产品所有的错误列表，请查看错误码文档。

实际案例

请求

```
DELETE /123 HTTP/1.1
Host: <BucketName-APPID>.<Endpoint>
Date: Wed, 23 Oct 2016 21:32:00 GMT
Authorization: q-sign-algorithm=sha1&q-ak=AKIDxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx&q-sign-time=1484213409;32557109409&q-key-time=1484213409;32557109409&q-header-list=host&q-url-param-list=&q-signature=1c24fe260ffe79b8603f932c4e916a6cbb0af44a
```

响应

```
HTTP/1.1 204 No Content
Content-Type: application/xml
Content-Length: 0
Connection: keep-alive
Date: Wed, 23 Oct 2016 21:32:00 GMT
x-cos-request-id: NTg3NzRjYTRfYmRjMzVfMzFhOF82MmM3Yg==
```

删除多个对象

最近更新时间: 2025-01-15 17:01:00

功能描述

DELETE Multiple Object 接口请求实现在指定 Bucket 中批量删除 Object，单次请求最大支持批量删除1000个 Object。对于响应结果，COS 提供 Verbose 和 Quiet 两种模式：Verbose 模式将返回每个 Object 的删除结果；Quiet 模式只返回报错的 Object 信息。

> **注意：**
>
> 此请求必须携带 Content-MD5 用来校验 Body 的完整性。

细节分析

1. 每一个批量删除请求，最多只能包含1000需要删除的对象。
2. 批量删除支持二种模式的放回，verbose 模式和 quiet 模式，默认为 verbose 模式。verbose 模式返回每个 key 的删除情况，quiet 模式只返回删除失败的 key 的情况。
3. 批量删除需要携带 Content-MD5 头部，用以校验请求 body 没有被修改。
4. 批量删除请求允许删除一个不存在的 key，仍然认为成功。

版本控制

当启用版本控制时，该请求操作可以为每一个要删除的对象指定版本 ID，此时将永久删除对象的指定版本或指定删除标记，否则将创建一个删除标记作为指定对象的最新版本。

当针对某个对象的删除操作创建或删除了删除标记，那么该对象的删除结果将同时返回 <DeleteMarker>true</DeleteMarker> 和 <DeleteMarkerVersionId> 元素，代表该请求操作创建或删除了指定对象的删除标记。

当针对某个对象的删除操作永久删除了特定的版本 ID（包括删除标记的版本 ID），那么该对象的删除结果将返回 <VersionId> 元素，代表该请求操作删除的版本 ID。

请求

语法示例：

```
POST /?delete HTTP/1.1
Host: <BucketName-APPID>.<Endpoint>
Date: GMT Date
Content-Length: length
Content-Type: application/xml
Content-MD5: MD5
Authorization: Auth String

<Delete>
<Quiet></Quiet>
<Object>
<Key></Key>
</Object>
<Object>
<Key></Key>
</Object>
...
</Delete>
```

> **说明：**
>
> Authorization: Auth String（详细参见请求签名文档）。

请求行

```
POST /?delete HTTP/1.1
```

该 API 接口接受 POST 请求。

请求头

公共头部

该请求操作的实现使用公共请求头，了解公共请求头详细请参见 公共请求头部 文档。

非公共头部

必选头部 该请求操作的实现使用如下必选头部：

名称	描述	类型	必选
Content-Length	RFC 2616 中定义的 HTTP 请求内容长度（字节）	String	是
Content-MD5	RFC 1864 中定义的经过 Base64 编码的 128-bit 内容 MD5 校验值。此头部用来校验文件内容是否发生变化	String	是

请求体

该请求的请求体具体节点内容为：

```
<Delete>
<Quiet></Quiet>
<Object>
<Key></Key>
</Object>
<Object>
<Key></Key>
</Object>
...
</Delete>
```

具体内容描述如下：

节点名称（关键字）	父节点	描述	类型	必选
DELETE	无	说明本次删除的返回结果方式和目标 Object	Container	是
Quiet	DELETE	布尔值，这个值决定了是否启动 Quiet 模式。 值为 true 启动 Quiet 模式，值为 false 则启动 Verbose 模式，默认值为 False	Boolean	否
Object	DELETE	说明每个将要删除的目标 Object 信息	Container	是
Key	DELETE.Object	目标 Object 文件名称	String	是

响应

响应头

公共响应头

该响应使用公共响应头,了解公共响应头详细请参见 公共响应头部 文档。

特有响应头

该请求操作无特殊的响应头。

响应体

该响应体返回为 **application/xml** 数据，包含完整节点数据的内容展示如下：

```
<DeleteResult>
<Deleted>
<Key></Key>
</Deleted>
<Error>
<Key></Key>
<Code></Code>
<Message></Message>
</Error>
</DeleteResult>
```

具体内容如下：

节点名称（关键字）	父节点	描述	类型

节点名称（关键字）	父节点	描述	类型
DeleteResult	无	说明本次删除返回结果的方式和目标 Object	Container

Container 节点 DeleteResult 的内容：

节点名称（关键字）	父节点	描述	类型
Deleted	DeleteResult	说明本次删除的成功 Object 信息	Boolean
Error	DeleteResult	说明本次删除的失败 Object 信息	Container

Container 节点 Deleted 的内容：

节点名称（关键字）	父节点	描述	类型
Key	DeleteResult.Deleted	Object 的名称	String

Container 节点 Error 的内容：

节点名称（关键字）	父节点	描述	类型
Key	DeleteResult.Error	删除失败的 Object 的名称	String
Code	DeleteResult.Error	删除失败的错误代码	String
Message	DeleteResult.Error	删除失败的错误信息	String

错误分析

以下描述此请求可能会发生的一些特殊的且常见的错误情况：

错误码	HTTP状态码	描述
InvalidRequest	400 Bad Request	没有携带必填字段 Content-MD5，同时返回 `Missing required header for this request: Content-MD5` 错误信息
MalformedXML	400 Bad Request	如果请求的 key 的个数，超过1000，则会返回 MalformedXML 错误，同时返回 `delete key size is greater than 1000`
InvalidDigest	400 Bad Request	携带的 Content-MD5 和服务端计算的请求 body 的不一致

获取更多关于 COS 的错误码的信息，或者产品所有的错误列表，请查看 错误码 文档。

实际案例

请求

```
POST /?delete HTTP/1.1
Host: <BucketName-APPID>.<Endpoint>
Date: Wed, 23 Oct 2016 21:32:00 GMT
Connection: keep-alive
Accept-Encoding: gzip, deflate
Accept: */*
User-Agent: python-requests/2.12.4
Authorization: q-sign-algorithm=sha1&q-ak=AKIDxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx&q-sign-time=1480932292;19810122922&q-key-time=1480932292;19810122922&q-url-param-list=delete&q-header-list=host&q-signature=c54f22fd92232a76972ba599cba25a8a733d2fef
Content-MD5: yoLiNjQuvB7lu8cEmPafrQ==
Content-Length: 125

<Delete>
<Quiet>true</Quiet>
<Object>
<Key>aa</Key>
</Object>
```

```
<Object>
<Key>aaa</Key>
</Object>
</Delete>
```

响应

```
HTTP/1.1 200 OK
Content-Type: application/xml
Content-Length: 17
Connection: keep-alive
Date: Tue, 22 Aug 2017 12:00:48 GMT
Server: tencent-cos
x-cos-request-id: NTk5YzFjZjBfZWFhZDM1MGFfMjkwZV9lZGM3ZQ==

<DeleteResult/>
```

请求

```
POST /?delete HTTP/1.1
Host: <BucketName-APPID>.<Endpoint>
Date: Tue, 22 Aug 2017 12:16:35 GMT
Connection: keep-alive
Accept-Encoding: gzip, deflate
Accept: */*
User-Agent: python-requests/2.12.4
Authorization: q-sign-algorithm=sha1&q-ak=AKIDxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx&q-sign-time=1480932292;19810122 92&q-key-time=1480932292;19810122 92&q-url-param-list=delete&q-header-list=host&q-signature=c54f22fd92232a76972ba599cba25a8a733d2fef
Content-MD5: V0XuU8V7aqMYeWyD3BC2nQ==
Content-Length: 126

<Delete>
<Quiet>false</Quiet>
<Object>
<Key>aa</Key>
</Object>
<Object>
<Key>aaa</Key>
</Object>
</Delete>
```

响应

```
HTTP/1.1 200 OK
Content-Type: application/xml
Content-Length: 111
Connection: keep-alive
Date: Tue, 22 Aug 2017 12:16:35 GMT
Server: tencent-cos
x-cos-request-id: NTk5YzIwYTNfMzFhYzM1MGFfMmNmOWZfZWVhNjQ=

<DeleteResult>
<Deleted>
<Key>aa</Key>
</Deleted>
<Deleted>
<Key>aaa</Key>
</Deleted>
</DeleteResult>
```

预请求跨域配置

最近更新时间: 2025-01-15 17:01:00

功能描述

OPTIONS Object 接口实现 Object 跨域访问配置的预请求。即在发送跨域请求之前会发送一个 OPTIONS 请求并带上特定的来源域，HTTP 方法和 Header 信息等给 COS，以决定是否可以发送真正的跨域请求。当 CORS 配置不存在时，请求返回 403 Forbidden。可以通过 PUT Bucket cors 接口来开启 Bucket 的 CORS 支持。

请求

请求示例

```
OPTIONS /<ObjectKey> HTTP/1.1
Host: <BucketName-APPID>.<Endpoint>
Date: GMT Date
Origin: Origin
Access-Control-Request-Method: HTTPMethod
Access-Control-Request-Headers: RequestHeader
Authorization: Auth String
```

> **说明：**
>
> Authorization: Auth String（详情请参阅请求签名章节）。

请求头

公共头部

该请求操作的实现使用公共请求头，了解公共请求头详情请参阅 公共请求头部 章节。

非公共头部

名称	类型	描述	必选
Origin	string	模拟跨域访问的请求来源域名	是
Access-Control-Request-Method	string	模拟跨域访问的请求 HTTP 方法	是
Access-Control-Request-Headers	string	模拟跨域访问的请求头部	否

请求体

该请求的请求体为空。

响应

响应头

公共响应头

该响应包含公共响应头，了解公共响应头详情请参阅 公共响应头部 章节。

特有响应头

该请求操作的特有响应头具体数据为：

名称	类型	描述
Access-Control-Allow-Origin	string	模拟跨域访问的请求来源域名，当来源不允许的时候，此 Header 不返回

名称	类型	描述
Access-Control-Allow-Methods	string	模拟跨域访问的请求 HTTP 方法，当请求方法不允许的时候，此 Header 不返回
Access-Control-Allow-Headers	string	模拟跨域访问的请求头部，当模拟任何请求头部不允许的时候，此 Header 不返回该请求头部
Access-Control-Expose-Headers	string	模拟跨域访问的请求 HTTP 方法，当请求方法不允许的时候，此 Header 不返回
Access-Control-Max-Age	string	设置 OPTIONS 请求得到结果的有效期

响应体

该响应体为空。

实际案例

请求

```
OPTIONS /exampleobject HTTP/1.1
Host: <BucketName-APPID>.<Endpoint>
Date: Thu, 12 Jan 2017 17:26:53 GMT
Origin: http://imgcache.finance.cloud.tencent.com:80www.qq.com
Access-Control-Request-Method: PUT
Authorization: q-sign-algorithm=sha1&q-ak=AKIDxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx&q-sign-time=1487070734;32466654734&q-key-time=1487070734;32559966734&q-header-list=host&q-url-param-list=&q-signature=2ac3ada19910f44836ae0df72a0ec1003f34324b
```

响应

```
HTTP/1.1 200 OK
Content-Type: application/xml
Content-Length: 16087
Connection: keep-alive
x-cos-request-id: NTg3NzRiZGRfYmRjMzVfM2Y2OF81N2YzNA==
Date: Thu, 12 Jan 2017 17:26:53 GMT
ETag: \"9a4802d5c99dafe1c04da0a8e7e166bf\"
Access-Control-Allow-Origin: http://imgcache.finance.cloud.tencent.com:80www.qq.com
Access-Control-Allow-Methods: PUT
Access-Control-Expose-Headers: x-cos-request-id
Server: tencent-cos
```

恢复归档对象

最近更新时间: 2025-01-15 17:01:00

功能描述

POST Object restore 接口可以对一个通过 COS 归档为 archive 类型的对象进行恢复，恢复出的可读取对象是临时的，您可以设置需要保持可读，以及随后删除该临时副本的时间。您可以用 Days 参数来指定临时对象的过期时间，若超出该时间且期间您没有发起任何复制、延长等操作，该临时对象将被系统自动删除。临时对象仅为 archive 类型对象的副本，被归档的源对象在此期间将始终存在。

请求

请求示例

```
POST /<ObjectKey>?restore HTTP/1.1
Host: <BucketName-APPID>.<Endpoint>
Date: GMT Date
Authorization: Auth String
```

请求头

公共头部

该请求操作的实现使用公共请求头。

非公共头部

该请求操作无特殊的请求头部信息。

请求体

该请求操作的实现需要有如下请求体。

```
<RestoreRequest>
<Days>2</Days>
<CASJobParameters>
<Tier>Bulk</Tier>
</CASJobParameters>
</RestoreRequest>
```

具体的数据描述如下：

节点名称（关键字）	父节点	描述	类型	必选
RestoreRequest	无	用于恢复数据的容器	Container	是
Days	无	设置临时副本的过期时间	integer	是
CASJobParameters	无	归档存储工作参数的容器	Container	是
Tier	无	恢复数据时，Tier 可以指定为 CAS 支持的三种恢复模式，分别为 Standard（标准模式，恢复任务在3 - 5小时内完成）、Expedited（极速模式，恢复任务在15分钟内可完成）以及 Bulk（批量模式，恢复任务在5 - 12小时内完成）	Enum	是

响应

响应头

公共响应头

该响应包含公共响应头。

特有响应头

该响应无特殊的响应头。

响应体

该响应体为空。

错误码

该请求操作可能会出现如下错误信息。

错误码	描述	HTTP 状态码
None	恢复成功	202 Accepted
RestoreAlreadyInProgress	对象已经在恢复中	409 Conflict

实际案例

请求

```
POST /exampleobject?restore HTTP/1.1
Accept: */*
Authorization:q-sign-algorithm=sha1&q-ak=AKIDxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx&q-sign-time=1497530202;1497610202&q-key-time=1497530202;1497610202&q-header-list=&q-url-param-list=&q-signature=28e9a4986df11bed0255e97ff90500557e0ea057
Host: <BucketName-APPID>.<Endpoint>
Content-Length: 105
Content-Type: application/x-www-form-urlencoded

<RestoreRequest>
<Days>2</Days>
<CASJobParameters>
<Tier>Bulk</Tier>
</CASJobParameters>
</RestoreRequest>
```

响应

```
HTTP/1.1 202 Accepted
Content-Type: application/xml
Content-Length: 0
Connection: keep-alive
Date: Thu, 15 Jun 2017 12:37:29 GMT
Server: tencent-cos
x-cos-request-id: NTk0MjdmODlfMjQ4OGY3XzYzYzhfMjc=
```

访问控制
设置对象ACL

最近更新时间: 2025-01-15 17:01:00

功能描述

PUT Object acl 接口用来对某个 Bucket 中的某个的 Object 进行 ACL 表的配置，您可以通过 Header："x-cos-acl"，"x-cos-grant-read"，"x-cos-grant-full-control" 传入 ACL 信息，或者通过 Body 以 XML 格式传入 ACL 信息。

请求

请求示例

```
PUT /<ObjectKey>?acl HTTP/1.1
Host: <BucketName-APPID>.<Endpoint>
Date: GMT Date
Authorization: Auth String
```

> **说明：**
>
> Authorization: Auth String（详情请参阅请求签名文档）。

请求头

公共头部

该请求操作的实现使用公共请求头，了解公共请求头详情请参阅 公共请求头部 文档。

非公共头部

名称	描述	类型	必选
x-cos-acl	定义 Object 的 ACL 属性，有效值：private，public-read，default；默认值：default（继承 Bucket 权限）。 注：当前访问策略条目限制为1000条，如果您不需要进行 Object ACL 控制，请填 default 或者此项不进行设置，默认继承 Bucket 权限	String	否
x-cos-grant-read	赋予被授权者读的权限。格式：x-cos-grant-read: id="[OwnerUin]"	String	否
x-cos-grant-full-control	赋予被授权者所有的权限。格式：x-cos-grant-full-control: id="[OwnerUin]"	String	否

请求体

该响应体返回为 **application/xml** 数据，包含完整节点数据的内容展示如下：

```
<AccessControlPolicy>
<Owner>
<ID>qcs::cam::uin/100000000001:uin/100000000001</ID>
<DisplayName>qcs::cam::uin/100000000001:uin/100000000001</DisplayName>
</Owner>
<AccessControlList>
<Grant>
<Grantee xmlns:xsi="http://imgcache.finance.cloud.tencent.com:80www.w3.org/2001/XMLSchema-instance" xsi:type="Group">
<URI>http://imgcache.finance.cloud.tencent.com:80cam.qcloud.com/groups/global/AllUsers</URI>
</Grantee>
<Permission>READ</Permission>
</Grant>
<Grant>
<Grantee xmlns:xsi="http://imgcache.finance.cloud.tencent.com:80www.w3.org/2001/XMLSchema-instance" xsi:type="CanonicalUser">
<ID>qcs::cam::uin/100000000001:uin/100000000001</ID>
<DisplayName>qcs::cam::uin/100000000001:uin/100000000001</DisplayName>
</Grantee>
<Permission>FULL_CONTROL</Permission>
</Grant>
```

```
</AccessControlList>
</AccessControlPolicy>
```

具体的数据内容如下：

节点名称（关键字）	父节点	描述	类型	必选
AccessControlPolicy	无	保存 GET Object acl 结果的容器	Container	是

Container 节点 AccessControlPolicy 的内容：

节点名称（关键字）	父节点	描述	类型	必选
Owner	AccessControlPolicy	Object 持有者信息	Container	是
AccessControlList	AccessControlPolicy	被授权者信息与权限信息	Container	是

Container 节点 Owner 的内容：

节点名称（关键字）	父节点	描述	类型	必选
ID	AccessControlPolicy.Owner	Object 持有者 ID， 格式为：qcs::cam::uin/:uin/如果是主帐号， 和 是同一个值	String	是
DisplayName	AccessControlPolicy.Owner	Object 持有者的名称	String	是

Container 节点 AccessControlList 的内容：

节点名称（关键字）	父节点	描述	类型	必选
Grant	AccessControlPolicy.AccessControlList	单个 Object 的授权信息。一个 AccessControlList 可以拥有100条 Grant	Container	是

Container 节点 Grant 的内容：

节点名称（关键字）	父节点	描述	类型	必选
Grantee	AccessControlPolicy.AccessControlList.Grant	说明被授权者的信息。type 类型可以为 RootAccount，Subaccount；当 type 类型为 RootAccount 时，ID 中指定的是主帐号;当 type 类型为 Subaccount 时，ID 中指定的是子帐号	Container	是
Permission	AccessControlPolicy.AccessControlList.Grant	指明授予被授权者的权限信息，枚举值：READ，FULL_CONTROL	String	是

Container 节点 Grantee 的内容：

节点名称（关键字）	父节点	描述	类型	必选
URI	AccessControlPolicy.AccessControlList.Grant.Grantee	指定所有用户	String	是
ID	AccessControlPolicy.AccessControlList.Grant.Grantee	用户的 ID，格式为：qcs::cam::uin/:uin/ 如果是主帐号， 和 是同一个值	String	是
DisplayName	AccessControlPolicy.AccessControlList.Grant.Grantee	用户的名称	String	是

响应

响应头

公共响应头

该响应使用公共响应头，了解公共响应头详情请参阅 公共响应头部 文档。

特有响应头

该请求操作无特殊的响应头部信息。

响应体

该请求响应体为空。

错误码

该响应可能会出现如下错误码信息，常见的错误信息请参阅 错误码 文档。

错误码	描述	状态码
SignatureDoesNotMatch	提供的签名不符合规则，返回该错误码	403 Forbidden
NoSuchBucket	如果试图添加的规则所在的 Bucket 不存在，返回该错误码	404 Not Found
MalformedXML	XML 格式不合法，请跟 Restful API 文档仔细比对	400 Bad Request
InvalidRequest	请求不合法，如果错误描述中显示"header acl and body acl conflict"，那么表示不能头部和 body 都有 acl 参数	400 Bad Request

实际案例

请求

```
PUT /exampleobject?acl HTTP/1.1
Host: <BucketName-APPID>.<Endpoint>
Date: Fri, 25 Feb 2017 04:10:22 GMT
Authorization: q-sign-algorithm=sha1&q-ak=AKIDxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx&q-sign-time=1484724784;32557620784&q-key-time=1484724784;32557620784&q-header-list=host&q-url-param-list=acl&q-signature=785d9075b8154119e6a075713c1b9e56ff0bddfc
Content-Length: 229
Content-Type: application/x-www-form-urlencoded

<AccessControlPolicy>
<Owner>
<ID>qcs::cam::uin/100000000001:uin/100000000001</ID>
<DisplayName>qcs::cam::uin/100000000001:uin/100000000001</DisplayName>
</Owner>
<AccessControlList>
<Grant>
<Grantee xmlns:xsi="http://imgcache.finance.cloud.tencent.com:80www.w3.org/2001/XMLSchema-instance" xsi:type="Group">
<URI>http://imgcache.finance.cloud.tencent.com:80cam.qcloud.com/groups/global/AllUsers</URI>
</Grantee>
<Permission>READ</Permission>
</Grant>
<Grant>
<Grantee xmlns:xsi="http://imgcache.finance.cloud.tencent.com:80www.w3.org/2001/XMLSchema-instance" xsi:type="CanonicalUser">
<ID>qcs::cam::uin/100000000001:uin/100000000001</ID>
<DisplayName>qcs::cam::uin/100000000001:uin/100000000001</DisplayName>
</Grantee>
<Permission>FULL_CONTROL</Permission>
</Grant>
</AccessControlList>
</AccessControlPolicy>
```

响应

```
HTTP/1.1 200 OK
Content-Type: application/xml
Content-Length: 0
Connection: keep-alive
Date: Fri, 25 Feb 2017 04:10:22 GMT\
Server: tencent-cos
x-cos-request-id: NTg3ZjFjMmJfOWIxZjRlXzZmNDhfMjIw
```

查询对象ACL

最近更新时间: 2025-01-15 17:01:00

功能描述

GET Object acl 接口用来获取某个存储桶下的某个对象的访问权限，只有存储桶的持有者才有权限操作。

请求

请求示例

```
GET /<ObjectKey>?acl HTTP/1.1
Host: <BucketName-APPID>.<Endpoint>
Date: GMT Date
Authorization: Auth String
```

> **说明：**
>
> Authorization: Auth String (详情请参阅请求签名文档)。

请求头

公共头部

该请求操作的实现使用公共请求头，了解公共请求头详情请参阅 公共请求头部 文档。

非公共头部

必选头部 该请求操作的实现使用如下必选头部：

名称	描述	类型	必选
Authorization	签名串	String	是

请求体

该请求的请求体为空。

响应

响应头

公共响应头

该响应使用公共响应头，了解公共响应头详情请参阅 公共响应头部 文档。

特有响应头

该响应无特殊的响应头。

响应体

该响应体返回为 **application/xml** 数据，包含完整节点数据的内容展示如下：

```
<AccessControlPolicy>
<Owner>
<ID>qcs::cam::uin/100000000001:uin/100000000001</ID>
<DisplayName>qcs::cam::uin/100000000001:uin/100000000001</DisplayName>
</Owner>
<AccessControlList>
<Grant>
<Grantee xmlns:xsi="http://imgcache.finance.cloud.tencent.com:80www.w3.org/2001/XMLSchema-instance" xsi:type="Group">
<URI>http://imgcache.finance.cloud.tencent.com:80cam.qcloud.com/groups/global/AllUsers</URI>
</Grantee>
<Permission>READ</Permission>
```

```
</Grant>
<Grant>
<Grantee xmlns:xsi="http://imgcache.finance.cloud.tencent.com:80www.w3.org/2001/XMLSchema-instance" xsi:type="CanonicalUser">
<ID>qcs::cam::uin/100000000001:uin/100000000001</ID>
<DisplayName>qcs::cam::uin/100000000001:uin/100000000001</DisplayName>
</Grantee>
<Permission>FULL_CONTROL</Permission>
</Grant>
</AccessControlList>
</AccessControlPolicy>
```

具体的数据内容如下：

节点名称（关键字）	父节点	描述	类型
AccessControlPolicy	无	保存 GET Object acl 结果的容器	Container

Container 节点 AccessControlPolicy 的内容：

节点名称（关键字）	父节点	描述	类型
Owner	AccessControlPolicy	Object 持有者信息	Container
AccessControlList	AccessControlPolicy	被授权者信息与权限信息	Container

Container 节点 Owner 的内容：

节点名称（关键字）	父节点	描述	类型
ID	AccessControlPolicy.Owner	Object 持有者 ID， 格式为：qcs::cam::uin/:uin/ 如果是主帐号，和 是同一个值	String
DisplayName	AccessControlPolicy.Owner	Object 持有者的名称	String

Container 节点 AccessControlList 的内容：

节点名称（关键字）	父节点	描述	类型
Grant	AccessControlPolicy.AccessControlList	单个 Object 的授权信息。一个 AccessControlList 可以拥有 100 条 Grant	Container

Container 节点 Grant 的内容：

节点名称（关键字）	父节点	描述	类型
Grantee	AccessControlPolicy.AccessControlList.Grant	说明被授权者的信息。type 类型可以为 RootAccount， Subaccount；当 type 类型为 RootAccount 时，ID 中指定的是主帐号;当 type 类型为 Subaccount 时，ID 中指定的是子帐号	Container
Permission	AccessControlPolicy.AccessControlList.Grant	指明授予被授权者的权限信息，枚举值：READ，FULL_CONTROL	String

Container 节点 Grantee 的内容：

节点名称（关键字）	父节点	描述	类型
URI	AccessControlPolicy.AccessControlList.Grant.Grantee	指定所有用户	String
ID	AccessControlPolicy.AccessControlList.Grant.Grantee	用户的 ID，格式为：qcs::cam::uin/:uin/ 如果是主帐号，和 是同一个值。	String
DisplayName	AccessControlPolicy.AccessControlList.Grant.Grantee	用户的名称	String

实际案例

请求

```
GET /exampleobject?acl HTTP/1.1
Host: <BucketName-APPID>.<Endpoint>
Date: Fri, 10 Mar 2016 09:45:46 GMT
Authorization: q-sign-algorithm=sha1&q-ak=AKIDxxxxxxxxxxxx&q-sign-time=1484213027;32557109027&q-key-time=1484213027;32557109027&q-header-list=host&q-url-param-list=acl&q-signature=dcc1eb2022b79cb2a780bf062d3a40e120b4065
```

响应

```
HTTP/1.1 200 OK
Content-Type: application/xml
Content-Length: 266
Connection: keep-alive
Date: Fri, 10 Mar 2016 09:45:46 GMT
Server: tencent-cos
x-cos-request-id: NTg3NzRiMjVfYmRjMzVfMTViMl82ZGZmNw==

<AccessControlPolicy>
<Owner>
<ID>qcs::cam::uin/100000000001:uin/100000000001</ID>
<DisplayName>qcs::cam::uin/100000000001:uin/100000000001</DisplayName>
</Owner>
<AccessControlList>
<Grant>
<Grantee xmlns:xsi="http://imgcache.finance.cloud.tencent.com:80www.w3.org/2001/XMLSchema-instance" xsi:type="Group">
<URI>http://imgcache.finance.cloud.tencent.com:80cam.qcloud.com/groups/global/AllUsers</URI>
</Grantee>
<Permission>READ</Permission>
</Grant>
<Grant>
<Grantee xmlns:xsi="http://imgcache.finance.cloud.tencent.com:80www.w3.org/2001/XMLSchema-instance" xsi:type="CanonicalUser">
<ID>qcs::cam::uin/100000000001:uin/100000000001</ID>
<DisplayName>qcs::cam::uin/100000000001:uin/100000000001</DisplayName>
</Grantee>
<Permission>FULL_CONTROL</Permission>
</Grant>
</AccessControlList>
</AccessControlPolicy>
```

分块上传
初始化分块上传

最近更新时间: 2025-01-15 17:01:00

功能描述

Initiate Multipart Upload 接口请求实现初始化分片上传，成功执行此请求以后会返回 UploadId 用于后续的 Upload Part 请求。

请求

请求示例

```
POST /<ObjectKey>?uploads HTTP/1.1
Host: <BucketName-APPID>.<Endpoint>
Date: GMT Date
Authorization: Auth String
```

> **说明：**
>
> Authorization: Auth String（详情请参阅请求签名文档）。

请求头

公共头部

该请求操作的实现使用公共请求头，了解公共请求头详情，请参阅 公共请求头部 文档。

非公共头部

推荐头部 该请求操作的实现使用如下推荐请求头部信息：

名称	描述	类型	必选
Cache-Control	RFC 2616 中定义的缓存策略，将作为 Object 元数据保存	String	否
Content-Disposition	RFC 2616 中定义的文件名称，将作为 Object 元数据保存	String	否
Content-Encoding	RFC 2616 中定义的编码格式，将作为 Object 元数据保存	String	否
Content-Type	RFC 2616 中定义的内容类型（MIME），将作为 Object 元数据保存	String	否
Expires	RFC 2616 中定义的文件日期和时间，将作为 Object 元数据保存	String	否
x-cos-meta-*	包括用户自定义头部后缀和用户自定义头部信息，将作为 Object 元数据返回，大小限制为2KB **注意**：用户自定义头部信息支持下划线，但用户自定义头部后缀不支持下划线	String	否

权限相关头部

> **说明：**
>
> 了解更多 ACL 请求请参阅 ACL 概述文档。

名称	描述	类型	必选
x-cos-acl	定义 Object 的 ACL 属性，有效值：private，public-read，default；默认值：default（继承 Bucket 权限） **注意**：当前访问策略条目限制为1000条，如果您不需要进行 Object ACL 控制，请填 default 或者此项不进行设置，默认继承 Bucket 权限	String	否
x-cos-grant-read	赋予被授权者读的权限，格式：x-cos-grant-read: id="[OwnerUin]"	String	否
x-cos-grant-full-control	赋予被授权者所有的权限，格式：x-cos-grant-full-control: id="[OwnerUin]"	String	否

请求体

该请求的操作请求体为空。

响应

响应头

公共响应头

该响应使用公共响应头，了解公共响应头详情请参阅 公共响应头部 文档。

响应体

该响应体返回为 **application/xml** 数据，包含完整节点数据的内容展示如下：

```
<InitiateMultipartUploadResult>
<Bucket>examplebucket-1250000000</Bucket>
<Key>exampleobject</Key>
<UploadId>1484727270323ddb949d528c629235314a9ead80f0ba5d993a3d76b460e6a9cceb9633b08e</UploadId>
</InitiateMultipartUploadResult>
```

具体的数据内容如下：

节点名称（关键字）	父节点	描述	类型
InitiateMultipartUploadResult	无	说明所有返回信息	Container

Container 节点 InitiateMultipartUploadResult 的内容：

节点名称（关键字）	父节点	描述	类型
Bucket	InitiateMultipartUploadResult	分片上传的目标 Bucket，由用户自定义字符串和系统生成 APPID 数字串由中划线连接而成，如：examplebucket-1250000000	Container
Key	InitiateMultipartUploadResult	Object 的名称	Container
UploadId	InitiateMultipartUploadResult	在后续上传中使用的 ID	Container

实际案例

请求

```
POST /exampleobject?uploads HTTP/1.1
Host: <BucketName-APPID>.<Endpoint>
Date: Fri, 10 Mar 2016 09:45:46 GMT
Authorization: q-sign-algorithm=sha1&q-ak=AKIDxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx&q-sign-time=1484727259;32557623259&q-key-time=1484727259;3
2557623259&q-header-list=host&q-url-param-list=uploads&q-signature=b5f46c47379aeaee74be7578380b193c01b28045
```

响应

```
HTTP/1.1 200 OK
Content-Type: application/xml
Content-Length: 230
Connection: keep-alive
Date: Fri, 10 Mar 2016 09:45:46 GMT
Server: tencent-cos
x-cos-request-id: NTg3ZjIzZTZfOWIxZjRlXzZmMzhfMWRj

<InitiateMultipartUploadResult>
<Bucket>examplebucket-1250000000</Bucket>
<Key>exampleobject</Key>
<UploadId>1484727270323ddb949d528c629235314a9ead80f0ba5d993a3d76b460e6a9cceb9633b08e</UploadId>
</InitiateMultipartUploadResult>
```

上传分块

最近更新时间: 2025-01-15 17:01:00

功能描述

Upload Part 接口请求实现将对象按照分块的方式上传到 COS。最多支持 10000 分块，每个分块大小为 1 MB 到 5 GB ，最后一个分块可以小于 1 MB。

细节分析

1. 分块上传首先需要进行初始化，使用 Initiate Multipart Upload 接口实现，初始化后会得到一个 uploadId ，唯一标识本次上传。
2. 在每次请求 Upload Part 时，需要携带 partNumber 和 uploadId，partNumber 为块的编号，支持乱序上传。
3. 当传入 uploadId 和 partNumber 都相同的时候，后传入的块将覆盖之前传入的块。当 uploadId 不存在时会返回 404 错误，NoSuchUpload。

请求

语法示例：

```
PUT /ObjectName?partNumber=PartNumber&uploadId=UploadId HTTP/1.1
Host: <BucketName-APPID>.<Endpoint>
Date: GMT Date
Content-Length: Size
Authorization: Auth String
```

> **说明：**
>
> Authorization: Auth String (详细参见请求签名章节)。

请求行

```
PUT /ObjectName?partNumber=PartNumber&uploadId=UploadId HTTP/1.1
```

该 API 接口接受 PUT 请求。

请求参数

包含所有请求参数的请求行示例：

```
PUT /ObjectName?partNumber=PartNumber&uploadId=UploadId HTTP/1.1
```

具体内容如下：

参数名称	描述	类型	必选
partNumber	标识本次分块上传的编号。	String	是
uploadId	标识本次分块上传的 ID。 使用 Initiate Multipart Upload 接口初始化分片上传时会得到一个 uploadId，该 ID 不但唯一标识这一分块数据，也标识了这分块数据在整个文件内的相对位置。	String	是

请求头

公共头部

该请求操作的实现使用公共请求头,了解公共请求头详细请参见 公共请求头部 章节。

非公共头部

必选头部 该请求操作需要请求头使用必选头部，具体内容如下：

名称	描述	类型	必选
Content-Length	RFC 2616 中定义的 HTTP 请求内容长度（字节）。	String	是

推荐头部 该请求操作推荐请求头使用推荐头部，具体内容如下：

名称	描述	类型	必选
Expect	RFC 2616 中定义的 HTTP 请求内容长度（字节）。	String	否
Content-MD5	RFC 1864 中定义的经过Base64编码的128-bit 内容 MD5 校验值。此头部用来校验文件内容是否发生变化。	String	否

请求体

该请求的操作请求体为空。

响应

响应头

公共响应头

该响应使用公共响应头,了解公共响应头详细请参见 公共响应头部 章节。

响应体

该响应的响应体为空。

实际案例

请求

```
PUT /ObjectName?partNumber=1&uploadId=1484727270323ddb949d528c629235314a9ead80f0ba5d993a3d76b460e6a9cceb9633b08e HTTP/1.1
Host: <BucketName-APPID>.<Endpoint>
Date: Wed，18 Jan 2017 16:17:03 GMT
Authorization: q-sign-algorithm=sha1&q-ak=AKIDxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx&q-sign-time=1484727403;32557623403&q-key-time=1484727403;32557623403&q-header-list=host&q-url-param-list=partNumber;uploadId&q-signature=bfc54518ca8fc31b3ea287f1ed2a0dd8c8e88a1d
Content-Length: 10485760

[Object]
```

响应

```
HTTP/1.1 200 OK
Content-Type: application/xml
Content-Length: 0
Connection: keep-alive
Date: Wed，18 Jan 2017 16:17:03 GMT
Etag: "e1e5b4965bc7d30880ed6d226f78a5390f1c09fc"
x-cos-request-id: NTg3ZjI0NzlfOWIxZjRlXzZmNGJfMWYy
```

复制分块

最近更新时间: 2025-01-15 17:01:00

功能描述

Upload Part - Copy 请求实现将一个对象的分块内容从源路径复制到目标路径。通过指定 x-cos-copy-source 来指定源对象，x-cos-copy-source-range 指定字节范围（允许分块的大小为5MB - 5GB）。

> **注意：**
>
> - 如果目标对象和源对象不属于同一个地域，且目标对象分块会超过5GB, 那么需要使用分块上传或者分块拷贝的接口来复制对象。
> - 使用上传分块对象，必须先初始化分块上传。在初始化分块上传的响应中，会返回一个唯一的描述符（upload ID），您需要在分块上传请求中携带此 ID。

版本

当存储桶启用了版本控制，x-cos-copy-source 标识被复制的对象的当前版本。如果当前版本是删除标记，并且 x-cos-copy-source 不指定版本，则对象存储会认为该对象已删除并返回404错误。如果您在 x-cos-copy-sourceand 中指定 versionId 且 versionId 是删除标记，则对象存储会返回 HTTP 400错误，因为删除标记不允许作为 x-cos-copy-source 的版本。

请求

请求示例

```
PUT /examplebucket?partNumber=PartNumber&uploadId=UploadId HTTP/1.1
Host: <BucketName-APPID>.<Endpoint>
Date: GMT Date
Authorization: Auth String
x-cos-copy-source: <Bucketname>-<APPID>.cos.<Region>.myqcloud.com/filepath
x-cos-copy-source-range: bytes=first-last
x-cos-copy-source-if-match: etag
x-cos-copy-source-if-none-match : etag
x-cos-copy-source-if-unmodified-since: time_stamp
x-cos-copy-source-if-modified-since: time_stamp
```

请求头

公共头部

该请求操作的实现使用公共请求头。

非公共头部

必选头部

该请求操作的实现使用如下必选头部：

名称	描述	类型	必选
x-cos-copy-source	源对象 URL 路径，可以通过 versionid 子资源指定历史版本	String	是

推荐头部

该请求操作的实现使用如下推荐请求头部信息：

名称	描述	类型	必选
x-cos-copy-source-range	源对象的字节范围，范围值必须使用 bytes=first-last 格式，first 和 last 都是基于 0 开始的偏移量。 例如 bytes=0-9 表示您希望拷贝源对象的开头10个字节的数据 ， 如果不指定，则表示拷贝整个对象	String	否
x-cos-copy-source-If-Modified-Since	当 Object 在指定时间后被修改，则执行操作，否则返回412 可与 x-cos-copy-source-If-None-Match 一起使用，与其他条件联合使用返回冲突	String	否
x-cos-copy-source-If-Unmodified-Since	当 Object 在指定时间后未被修改，则执行操作，否则返回412 可与 x-cos-copy-source-If-Match 一起使用，与其他条件联合使用返回冲突	String	否

名称	描述	类型	必选
x-cos-copy-source-If-Match	当 Object 的 Etag 和给定一致时，则执行操作，否则返回412 可与 x-cos-copy-source-If-Unmodified-Since 一起使用，与其他条件联合使用返回冲突	String	否
x-cos-copy-source-If-None-Match	当 Object 的 Etag 和给定不一致时，则执行操作，否则返回412 可与 x-cos-copy-source-If-Modified-Since 一起使用，与其他条件联合使用返回冲突	String	否

请求参数

名称	描述	类型	必选
partNumber	分块拷贝的块号	String	是
uploadId	使用上传分块文件，必须先初始化分块上传。在初始化分块上传的响应中，会返回一个唯一的描述符（upload ID），您需要在分块上传请求中携带此 ID	String	是

请求体

该请求的请求体为空。

响应

响应头

公共响应头

该响应包含公共响应头。

特有响应头

名称	描述	类型
x-cos-copy-source-version-id	如果已在源存储桶上启用版本控制，则复制源对象的版本	String
x-cos-server-side-encryption	如果通过 COS 管理的服务端加密来存储对象，响应将包含此头部和所使用的加密算法的值，AES256	String

响应体

该响应体返回为 **application/xml** 数据，包含完整节点数据的内容展示如下：

```
<?xml version="1.0" encoding="UTF-8" ?>
<CopyPartResult>
<ETag>"ba82b57cfdfda8bd17ad4e5879ebb4fe"</ETag>
<LastModified>2017-09-04T04:45:45</LastModified>
</CopyPartResult>
```

具体的数据内容如下：

名称	描述	类型
CopyPartResult	返回复制结果信息	String
ETag	返回对象的 MD5 算法校验值。ETag 的值可以用于检查 Object 的内容是否发生变化。	String
LastModified	返回对象最后修改时间，GMT 格式	String

实际案例

请求

```
PUT /exampleobject?partNumber=1&uploadId=1505706248ca8373f8a5cd52cb129f4bcf85e11dc8833df34f4f5bcc456c99c42cd1ffa2f9 HTTP/1.1
User-Agent: curl/7.19.7 (x86_64-redhat-linux-gnu) libcurl/7.19.7 NSS/3.13.1.0 zlib/1.2.3 libidn/1.18 libssh2/1.2.2
Accept: */*
```

```
x-cos-copy-source:examplebucket-1250000000.cos.ap-shanghai.myqcloud.com/exampleobject1
x-cos-copy-source-range: bytes=10-100
Host: <BucketName-APPID>.<Endpoint>
Authorization:q-sign-algorithm=sha1&q-ak=AKIDxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx&q-sign-time=1507530223;1508530223&q-key-time=1507530223;15
08530223&q-header-list=&q-url-param-list=&q-signature=d02640c0821c49293e5c289fa07290e6b2f05cb2
```

响应

```
HTTP/1.1 200 OK
Content-Type: application/xml
Content-Length: 133
Connection: keep-alive
Date: Mon, 04 Sep 2017 04:45:45 GMT
Server: tencent-cos
x-cos-request-id: NTlkYjFjYWJfMjQ4OGY3MGFfNGIzZV9k

<CopyPartResult>
<ETag>"ba82b57cfdfda8bd17ad4e5879ebb4fe"</ETag>
<LastModified>2017-09-04T04:45:45</LastModified>
</CopyPartResult>
```

完成分块上传

最近更新时间: 2025-01-15 17:01:00

功能描述

Complete Multipart Upload 接口请求用来实现完成整个分块上传。当使用 Upload Parts 上传完所有块以后，必须调用该 API 来完成整个文件的分块上传。在使用该 API 时，您必须在请求 Body 中给出每一个块的 PartNumber 和 ETag，用来校验块的准确性。 由于分块上传完后需要合并，而合并需要数分钟时间，因而当合并分块开始的时候，COS 就立即返回200的状态码，在合并的过程中，COS 会周期性的返回空格信息来保持连接活跃，直到合并完成，COS会在 Body 中返回合并后块的内容。

- 当上传块小于1MB的时候，在调用该 API 时，会返回400 EntityTooSmall。
- 当上传块编号不连续的时候，在调用该 API 时，会返回400 InvalidPart。
- 当请求 Body 中的块信息没有按序号从小到大排列的时候，在调用该 API 时，会返回400 InvalidPartOrder。
- 当 UploadId 不存在的时候，在调用该 API 时，会返回404 NoSuchUpload。

> **注意：**
>
> 建议您及时完成分块上传或者舍弃分块上传，因为已上传但是未终止的块会占用存储空间进而产生存储费用。

请求

请求示例

```
POST /<ObjectKey>?uploadId=UploadId HTTP/1.1
Host: <BucketName-APPID>.<Endpoint>
Date: GMT Date
Content-length: Size
Authorization: Auth String
```

> **说明：**
>
> Authorization: Auth String（详情请参阅请求签名章节）。

请求头

公共头部

该请求操作的实现使用公共请求头,了解公共请求头详细请参见 公共请求头部 章节。

非公共头部

该请求操作无特殊的请求头部信息。

请求体

该 API 接口请求的请求体具体节点内容为：

```
<CompleteMultipartUpload>
<Part>
<PartNumber>1</PartNumber>
<ETag>"fc392a65890e447ff4e2d256489a9773"</ETag>
</Part>
<Part>
<PartNumber>2</PartNumber>
<ETag>"fc392a65890e447ff4e2d256489a9773"</ETag>
</Part>
...
</CompleteMultipartUpload>
```

具体的数据内容如下：

节点名称（关键字）	父节点	描述	类型	必选
CompleteMultipartUpload	无	用来说明本次分块上传的所有信息	Container	是

Container 节点 CompleteMultipartUpload 的内容：

节点名称（关键字）	父节点	描述	类型	必选
Part	CompleteMultipartUpload	用来说明本次分块上传中每个块的信息	Container	是

Container 节点 Part 的内容：

节点名称（关键字）	父节点	描述	类型	必选
PartNumber	CompleteMultipartUpload.Part	块编号	Integer	是
ETag	CompleteMultipartUpload.Part	每个块文件的 MD5 算法校验值	String	是

响应

响应头

公共响应头

该响应使用公共响应头,了解公共响应头详细请参见 公共响应头部 章节。

响应体

该响应体返回为 **application/xml** 数据，包含完整节点数据的内容展示如下：

```
<CompleteMultipartUploadResult>
<Location>examplebucket-1250000000.cos.ap-beijing.myqcloud.com/ObjectName</Location>
<Bucket>examplebucket-1250000000</Bucket>
<Key>examplebucket</Key>
<ETag>"3a0f1fd698c235af9cf098cb74aa25bc"</ETag>
</CompleteMultipartUploadResult>
```

具体的数据内容如下：

节点名称（关键字）	父节点	描述	类型
CompleteMultipartUploadResult	无	说明所有返回信息	Container

Container 节点 CompleteMultipartUploadResult 的内容：

节点名称（关键字）	父节点	描述	类型
Location	CompleteMultipartUploadResult	创建 Object 的外网访问域名	URL
Bucket	CompleteMultipartUploadResult	分块上传的目标Bucket，由用户自定义字符串和系统生成appid数字串由中划线连接而成，如：examplebucket-1250000000	String
Key	CompleteMultipartUploadResult	Object 名称	String
ETag	CompleteMultipartUploadResult	合并后对象的唯一标签值，该值不是对象内容的 MD5 校验值，仅能用于检查对象唯一性	String

实际案例

请求

```
POST /exampleobject?uploadId=1484728886e63106e87d8207536ae8521c89c42a436fe23bb58854a7bb5e87b7d77d4ddc48 HTTP/1.1
Host: <BucketName-APPID>.<Endpoint>
Date: Wed，18 Jan 2017 16:17:03 GMT
Authorization: q-sign-algorithm=sha1&q-ak=AKIDxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx&q-sign-time=1484729794;32557625794&q-key-time=1484729794;32557625794&q-header-list=host&q-url-param-list=uploadId&q-signature=23627c8fddb3823cce4257b33c663fd83f9f820d
Content-Length: 138

<CompleteMultipartUpload>
<Part>
```

```
<PartNumber>1</PartNumber>
<ETag>"fc392a65890e447ff4e2d256489a9773"</ETag>
</Part>
<Part>
<PartNumber>2</PartNumber>
<ETag>"fc392a65890e447ff4e2d256489a9774"</ETag>
</Part>
</CompleteMultipartUpload>
```

响应

```
HTTP/1.1 200 OK
Content-Type: application/xml
Content-Length: 277
Connection: keep-alive
Date: Wed，18 Jan 2017 16:17:03 GMT
Server: tencent-cos
x-cos-request-id: NTg3ZjJlMjVfNDYyMDRlXzM0YzRfMjc1

<CompleteMultipartUploadResult>
<Location>examplebucket-1250000000.cos.ap-beijing.myqcloud.com/ObjectName</Location>
<Bucket>examplebucket-1250000000</Bucket>
<Key>examplebucket</Key>
<ETag>"3a0f1fd698c235af9cf098cb74aa25bc"</ETag>
</CompleteMultipartUploadResult>
```

终止分块上传

最近更新时间: 2025-01-15 17:01:00

功能描述

Abort Multipart Upload 用来实现舍弃一个分块上传并删除已上传的块。当您调用 Abort Multipart Upload 时，如果有正在使用这个 Upload Parts 上传块的请求，则 Upload Parts 会返回失败。当该 UploadId 不存在时，会返回404 NoSuchUpload。

> **注意：**
>
> 建议您及时完成分块上传或者舍弃分块上传，因为已上传但是未终止的块会占用存储空间进而产生存储费用。

请求

请求示例

```
DELETE /<ObjectKey>?uploadId=UploadId HTTP/1.1
Host: <BucketName-APPID>.<Endpoint>
Date: GMT Date
Authorization: Auth String
```

> **说明：**
>
> Authorization： Auth String（详细参见请求签名文档）。

请求参数

具体内容如下：

参数名称	描述	类型	必选
uploadId	标识本次分块上传的 ID。 使用 Initiate Multipart Upload 接口初始化分片上传时会得到一个 uploadId，该 ID 不但唯一标识这一分块数据，也标识了这分块数据在整个文件内的相对位置	String	是

请求头

公共头部

该请求操作的实现使用公共请求头,了解公共请求头详情请参阅 公共请求头部 文档。

非公共头部

该请求操作无特殊的请求头部信息。

请求体

该请求的请求体空。

响应

响应头

公共响应头

该响应使用公共响应头,了解公共响应头详情请参阅 公共响应头部 文档。

特有响应头

该请求操作无特殊的响应头。

响应体

该请求的响应体为空。

实际案例

请求

```
DELETE /exampleobject?uploadId=1484727270323ddb949d528c629235314a9ead80f0ba5d993a3d76b460e6a9cceb9633b08e HTTP/1.1
Host: <BucketName-APPID>.<Endpoint>
Date: Tue, 26 Oct 2013 21:22:00 GMT
Authorization: q-sign-algorithm=sha1&q-ak=AKIDxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx&q-sign-time=1484728626;32557624626&q-key-time=1484728626;3
2557624626&q-header-list=host&q-url-param-list=uploadId&q-signature=2d3036b57cade4a257b48a3a5dc922779a562b18
```

响应

```
HTTP/1.1 204 OK
Content-Type: application/xml
Content-Length: 0
Connection: keep-alive
Date: Tue, 26 Oct 2013 21:22:00 GMT
Server: tencent-cos
x-cos-request-id: NTg3ZjI5MzlfOTgxZjRlXzZhYjNfMjBh
```

查询分块上传

最近更新时间: 2025-01-15 17:01:00

功能描述

List Multipart Uploads 用来查询正在进行中的分块上传。单次请求操作最多列出1000个正在进行中的分块上传。

> **注意：**
>
> 该请求需要有 Bucket 的读权限。

请求

请求示例

```
GET /?uploads HTTP/1.1
Host: <BucketName-APPID>.<Endpoint>
Date: GMT Date
Authorization: Auth String
```

> **说明：**
>
> Authorization: Auth String（详情请参阅请求签名文档）。

请求头

公共头部

该请求操作的实现使用公共请求头，了解公共请求头详情请参阅 公共请求头部 文档。

非公共头部

该请求操作无特殊的请求头部信息。

请求参数

具体内容如下：

名称	描述	类型	必选
delimiter	定界符为一个符号，对 Object 名字包含指定前缀且第一次出现 delimiter 字符之间的 Object 作为一组元素：common prefix。如果没有 prefix，则从路径起点开始	String	否
encoding-type	规定返回值的编码格式，合法值：url	String	否
prefix	限定返回的 Object key 必须以 Prefix 作为前缀。 注意使用 prefix 查询时，返回的 key 中仍会包含 Prefix	String	否
max-uploads	设置最大返回的 multipart 数量，合法取值从1到1000，默认1000	String	否
key-marker	与 upload-id-marker 一起使用 当 upload-id-marker 未被指定时，ObjectName 字母顺序大于 key-marker 的条目将被列出 当 upload-id-marker 被指定时，ObjectName 字母顺序大于 key-marker 的条目被列出，ObjectName 字母顺序等于 key-marker 同时 UploadID 大于 upload-id-marker 的条目将被列出	String	否
upload-id-marker	与 key-marker 一起使用 当 key-marker 未被指定时，upload-id-marker 将被忽略 当 key-marker 被指定时，ObjectName字母顺序大于 key-marker 的条目被列出，ObjectName 字母顺序等于 key-marker 同时 UploadID 大于 upload-id-marker 的条目将被列出	String	否

请求体

该请求的请求体为空。

响应

响应头

公共响应头

该响应包含公共响应头，了解公共响应头详情请参阅 公共响应头部 文档。

特有响应头

该响应无特殊的响应头。

响应体

该响应体返回为 **application/xml** 数据，包含完整节点数据的内容展示如下：

```
<ListMultipartUploadsResult>
<Bucket></Bucket>
<Encoding-Type></Encoding-Type>
<KeyMarker></KeyMarker>
<UploadIdMarker></UploadIdMarker>
<NextKeyMarker></NextKeyMarker>
<NextUploadIdMarker></NextUploadIdMarker>
<MaxUploads></MaxUploads>
<IsTruncated></IsTruncated>
<Prefix></Prefix>
<Delimiter></Delimiter>
<Upload>
<Key></Key>
<UploadID></UploadID>
<StorageClass></StorageClass>
<Initiator>
<ID></ID>
<DisplayName></DisplayName>
</Initiator>
<Owner>
<ID></ID>
<DisplayName></DisplayName>
</Owner>
<Initiated></Initiated>
</Upload>
<CommonPrefixes>
<Prefix></Prefix>
</CommonPrefixes>
</ListMultipartUploadsResult>
```

具体的数据内容如下：

节点名称（关键字）	父节点	描述	类型
ListMultipartUploadsResult	无	用来表述所有分块上传的信息	Container

Container 节点 ListMultipartUploadsResult 的内容：

节点名称（关键字）	父节点	描述	类型
Bucket	ListMultipartUploadsResult	分块上传的目标 Bucket，由用户自定义字符串和系统生成 APPID 数字串由中划线连接而成，如：examplebucket-1250000000	String
Encoding-Type	ListMultipartUploadsResult	规定返回值的编码格式，合法值：url	String
KeyMarker	ListMultipartUploadsResult	列出条目从该 key 值开始	String
UploadIdMarker	ListMultipartUploadsResult	列出条目从该 UploadId 值开始	String
NextKeyMarker	ListMultipartUploadsResult	假如返回条目被截断，则返回 NextKeyMarker 就是下一个条目的起点	String
NextUploadIdMarker	ListMultipartUploadsResult	假如返回条目被截断，则返回 UploadId 就是下一个条目的起点	String
MaxUploads	ListMultipartUploadsResult	设置最大返回的 multipart 数量，合法取值从0到1000	String
IsTruncated	ListMultipartUploadsResult	返回条目是否被截断，布尔值：TRUE，FALSE	Boolean

节点名称（关键字）	父节点	描述	类型
Prefix	ListMultipartUploadsResult	限定返回的 Objectkey 必须以 Prefix 作为前缀， 注意使用 prefix 查询时，返回的 key 中仍会包含 Prefix	String
Delimiter	ListMultipartUploadsResult	定界符为一个符号，对 object 名字包含指定前缀且第一次出现 delimiter 字符之间的 object 作为一组元素：common prefix。如果没有 prefix，则从路径起点开始	String
Upload	ListMultipartUploadsResult	每个 Upload 的信息	Container
CommonPrefixes	ListMultipartUploadsResult	将 prefix 到 delimiter 之间的相同路径归为一类，定义为 Common Prefix	Container

Container 节点 Upload 的内容：

节点名称（关键字）	父节点	描述	类型
Key	ListMultipartUploadsResult.Upload	Object 的名称	String
UploadID	ListMultipartUploadsResult.Upload	标示本次分块上传的 ID	String
StorageClass	ListMultipartUploadsResult.Upload	用来表示分块的存储级别，枚举值：STANDARD，STANDARD_IA，ARCHIVE	String
Initiator	ListMultipartUploadsResult.Upload	用来表示本次上传发起者的信息	Container
Owner	ListMultipartUploadsResult.Upload	用来表示这些分块所有者的信息	Container
Initiated	ListMultipartUploadsResult.Upload	分块上传的起始时间	Date

Container 节点 Initiator 的内容：

节点名称（关键字）	父节点	描述	类型
ID	ListMultipartUploadsResult.Upload.Initiator	用户唯一的 CAM 身份 ID	String
DisplayName	ListMultipartUploadsResult.Upload.Initiator	用户身份 ID 的简称（UIN）	String

Container 节点 Owner 的内容：

节点名称（关键字）	父节点	描述	类型
ID	ListMultipartUploadsResult.Upload.Owner	用户唯一的 CAM 身份 ID	String
DisplayName	ListMultipartUploadsResult.Upload.Owner	用户身份 ID 的简称（UIN）	String

Container 节点 CommonPrefixes 的内容：

节点名称（关键字）	父节点	描述	类型
Prefix	ListMultipartUploadsResult.CommonPrefixes	显示具体的 CommonPrefixes	String

错误分析

以下描述此请求可能会发生的一些特殊的且常见的错误情况：

错误码	HTTP 状态码	描述
InvalidArgument	400 Bad Request	max-uploads 必须是整数，且值介于0 - 1000之间，否则返回 InvalidArgument encoding-type 只能取值 url，否则会返回 InvalidArgument

获取更多关于 COS 的错误码的信息，或者产品所有的错误列表，请查看 错误码 文档。

实际案例

请求

```
GET /?uploads HTTP/1.1
Host: <BucketName-APPID>.<Endpoint>
Date: Wed, 18 Jan 2015 21:32:00 GMT
Authorization: q-sign-algorithm=sha1&q-ak=AKIDxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx&q-sign-time=1484727508;32557623508&q-key-time=1484727508;3
2557623508&q-header-list=host&q-url-param-list=uploads&q-signature=5bd4759a7309f7da9a0550c224d8c61589c9dbbf
```

响应

```
HTTP/1.1 200 OK
Content-Type: application/xml
Content-Length: 1203
Date: Wed, 18 Jan 2015 21:32:00 GMT
Server: tencent-cos
x-cos-request-id: NTg3ZjI0ZGRfNDQyMDRlXzNhZmRfMjRl

<ListMultipartUploadsResult>
<Bucket>examplebucket-1250000000</Bucket>
<Encoding-Type/>
<KeyMarker/>
<UploadIdMarker/>
<MaxUploads>1000</MaxUploads>
<Prefix/>
<Delimiter>/</Delimiter>
<IsTruncated>false</IsTruncated>
<Upload>
<Key>Object</Key>
<UploadID>1484726657932bcb5b17f7a98a8cad9fc36a340ff204c79bd2f51e7dddf0b6d1da6220520c</UploadID>
<Initiator>
<ID>qcs::cam::uin/100000000001:uin/100000000001</ID>
<DisplayName>100000000001</DisplayName>
</Initiator>
<Owner>
<ID>qcs::cam::uin/100000000001:uin/100000000001</ID>
<DisplayName>100000000001</DisplayName>
</Owner>
<StorageClass>Standard</StorageClass>
<Initiated>Wed Jan 18 16:04:17 2017</Initiated>
</Upload>
<Upload>
<Key>Object</Key>
<UploadID>1484727158f2b8034e5407d18cbf28e84f754b791ecab607d25a2e52de9fee641e5f60707c</UploadID>
<Initiator>
<ID>qcs::cam::uin/100000000001:uin/100000000001</ID>
<DisplayName>100000000001</DisplayName>
</Initiator>
<Owner>
<ID>qcs::cam::uin/100000000001:uin/100000000001</ID>
<DisplayName>100000000001</DisplayName>
</Owner>
<StorageClass>Standard</StorageClass>
<Initiated>Wed Jan 18 16:12:38 2017</Initiated>
</Upload>
<Upload>
<Key>exampleobject</Key>
<UploadID>1484727270323ddb949d528c629235314a9ead80f0ba5d993a3d76b460e6a9cceb9633b08e</UploadID>
<Initiator>
<ID>qcs::cam::uin/100000000001:uin/100000000001</ID>
<DisplayName>100000000001</DisplayName>
</Initiator>
<Owner>
<ID>qcs::cam::uin/100000000001:uin/100000000001</ID>
<DisplayName>100000000001</DisplayName>
</Owner>
<StorageClass>Standard</StorageClass>
<Initiated>Wed Jan 18 16:14:30 2017</Initiated>
</Upload>
</ListMultipartUploadsResult>
```

查询已上传块

最近更新时间: 2025-01-15 17:01:00

功能描述

List Parts 用来查询特定分块上传中的已上传的块，即罗列出指定 UploadId 所属的所有已上传成功的分块。

请求

请求示例

```
GET /<ObjectKey>?uploadId=UploadId HTTP/1.1
Host: <BucketName-APPID>.<Endpoint>
Date: GMT Date
Authorization: Auth String
```

> **说明：**
>
> Authorization: Auth String (详情请参见请求签名章节)。

请求头

公共头部

该请求操作的实现使用公共请求头，了解公共请求头详情请参见 公共请求头部 章节。

非公共头部

该请求操作无特殊的请求头部信息。

请求参数

名称	类型	必选	描述
UploadId	string	是	标识本次分块上传的 ID。使用 Initiate Multipart Upload 接口初始化分片上传时会得到一个 uploadId，该 ID 不但唯一标识这一分块数据，也标识了这分块数据在整个文件内的相对位置
encoding-type	string	否	规定返回值的编码方式
max-parts	string	否	单次返回最大的条目数量，默认1000
part-number-marker	string	否	默认以 UTF-8 二进制顺序列出条目，所有列出条目从 marker 开始

请求体

该请求请求体为空。

响应

响应头

公共响应头

该响应包含公共响应头，了解公共响应头详情请参见 公共响应头部 章节。

特有响应头

该响应无特殊的响应头。

响应体

查询成功，返回 **application/xml** 数据，包含已完成的分片信息。

```
<?xml version="1.0" encoding="UTF-8" ?>
<ListPartsResult>
<Bucket>examplebucket-1250000000</Bucket>
```

```
<Encoding-type/>
<Key>exampleobject</Key>
<UploadId>14846420620b1f381e5d7b057692e131dd8d72dfa28f2633cfbbe4d0a9e8bd0719933545b0</UploadId>
<Initiator>
<ID>1250000000</ID>
<DisplyName>1250000000</DisplyName>
</Initiator>
<Owner>
<ID>qcs::cam::uin/100000000001:uin/100000000001</ID>
<DisplyName>100000000001</DisplyName>
</Owner>
<PartNumberMarker>0</PartNumberMarker>
<Part>
<PartNumber>1</PartNumber>
<LastModified>Tue Jan 17 16:43:37 2017</LastModified>
<ETag>"a1f8e5e4d63ac6970a0062a6277e191fe09a1382"</ETag>
<Size>5242880</Size>
</Part>
<NextPartNumberMarker>1</NextPartNumberMarker>
<StorageClass>STANDARD</StorageClass>
<MaxParts>1</MaxParts>
<IsTruncated>true</IsTruncated>
</ListPartsResult>
```

具体的数据描述如下：

节点名称（关键字）	父节点	描述	类型
ListPartsResult	无	保存 List Parts 请求结果的所有信息	Container

Container 节点 ListPartsResult 的内容：

节点名称（关键字）	父节点	描述	类型
Bucket	ListPartsResult	分块上传的目标 Bucket，存储桶的名字，由用户自定义字符串和系统生成 APPID 数字串由中划线连接而成，如：examplebucket-1250000000	string
Encoding-Type	ListPartsResult	编码格式	string
Key	ListPartsResult	Object 的名字	string
UploadId	ListPartsResult	标识本次分块上传的 ID	string
Initiator	ListPartsResult	用来表示这些分块所有者的信息	Container
Owner	ListPartsResult	用来表示这些分块所有者的信息	Container
StorageClass	ListPartsResult	用来表示这些分块的存储级别，枚举值：STANDARD，STANDARD_IA，ARCHIVE	string
PartNumberMarker	ListPartsResult	默认以 UTF-8 二进制顺序列出条目，所有列出条目从 marker 开始	string
NextPartNumberMarker	ListPartsResult	假如返回条目被截断，则返回 NextMarker 就是下一个条目的起点	string
MaxParts	ListPartsResult	单次返回最大的条目数量	string
IsTruncated	ListPartsResult	响应请求条目是否被截断，布尔值：true，false	boolean
Part	ListPartsResult	元数据信息	Container

Container 节点 Initiator 的内容：

节点名称（关键字）	父节点	描述	类型
ID	ListPartsResult.Initiator	创建者的一个唯一标识	string
DisplayName	ListPartsResult.Initiator	创建者的用户名描述	string

Container 节点 Owner 的内容：

节点名称（关键字）	父节点	描述	类型

节点名称（关键字）	父节点	描述	类型
ID	ListPartsResult.Owner	创建者的一个唯一标识	string
DisplayName	ListPartsResult.Owner	创建者的用户名描述	string

Container 节点 Part 的内容：

节点名称（关键字）	父节点	描述	类型
PartNumber	ListPartsResult.Part	块的编号	string
LastModified	ListPartsResult.Part	说明块最后被修改时间	string
ETag	ListPartsResult.Part	块的 MD-5 算法校验值	string
Size	ListPartsResult.Part	说明块大小，单位是 Byte	string

实际案例

请求

```
GET /exampleobject?uploadId=14846420620b1f381e5d7b057692e131dd8d72dfa28f2633cfbbe4d0a9e8bd0719933545b0&max-parts=1 HTTP/1.1
Host: <BucketName-APPID>.<Endpoint>
Date: Wed，18 Jan 2017 16:17:03 GMT
Authorization:q-sign-algorithm=sha1&q-ak=AKIDxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx&q-sign-time=1484643123;1484646723&q-key-time=1484643123;1484646723&q-header-list=host&q-url-param-list=max-parts;uploadid&q-signature=b8b4055724e64c9ad848190a2f7625fd3f9d3e87
```

响应

```
HTTP/1.1 200 OK
Content-Type: application/xml
Content-Length: 661
Connection: keep-alive
Date: Wed，18 Jan 2017 16:17:03 GMT
x-cos-request-id: NTg3ZGRiMzhfMmM4OGY3XzdhY2NfYw==

<ListPartsResult>
<Bucket>examplebucket-1250000000</Bucket>
<Encoding-type/>
<Key>exampleobject</Key>
<UploadId>14846420620b1f381e5d7b057692e131dd8d72dfa28f2633cfbbe4d0a9e8bd0719933545b0</UploadId>
<Initiator>
<ID>1250000000</ID>
<DisplyName>1250000000</DisplyName>
</Initiator>
<Owner>
<ID>qcs::cam::uin/100000000001:uin/100000000001</ID>
<DisplyName>100000000001</DisplyName>
</Owner>
<PartNumberMarker>0</PartNumberMarker>
<Part>
<PartNumber>1</PartNumber>
<LastModified>Tue Jan 17 16:43:37 2017</LastModified>
<ETag>"a1f8e5e4d63ac6970a0062a6277e191fe09a1382"</ETag>
<Size>5242880</Size>
</Part>
<NextPartNumberMarker>1</NextPartNumberMarker>
<StorageClass>STANDARD</StorageClass>
<MaxParts>1</MaxParts>
<IsTruncated>true</IsTruncated>
</ListPartsResult>
```

常见问题
一般性问题

最近更新时间: 2025-01-15 17:01:00

什么是对象存储 COS？

对象存储 COS 是在云上提供无层次结构的分布式存储产品，为用户提供单价较低且快速可靠的数据存储方案。COS 以冗余的方式跨多个可用区存储用户数据，并允许多个不同的客户端或应用程序线程同时对这些数据进行读或写操作。

用户可通过云服务器实例或互联网使用 Web API 接口存储和检索数据。在 COS 上的数据，用户使用指定域名的 URL 地址，通过 HTTP/HTTPS 协议存储和检索每个独立的数据对象内容。

有关对象存储的更多信息，请参见 COS 产品文档 。

对象存储和文件存储的区别是什么？

对象存储 无目录层次结构、无数据格式限制，可存储任意数量的数据，存储桶空间无容量上限，无需分区管理。数据支持高可用架构部署，设计保障数据最终一致性，不支持文件锁等特性。API 使用 HTTP/HTTPS 的协议访问，并提供 SDK 和工具等方式与业务集成，上传到 COS 的对象可通过 URL 地址直接访问或下载。

文件存储 使用常用的网络文件传输协议，可创建文件系统并实现大规模扩展，需挂载在云服务器中使用。文件存储可为网站、在线发行、存档各种应用存储。 计算吞吐量高，具有极高的可用性和持久性，也适用于并发较高或需要共享存储的需求。

对象存储和云硬盘的区别是什么？

对象存储 具备无文件系统、目录结构、文件数量和空间上限的特性，需通过 Web API 接口管理和访问存储，提供了 SDK 和工具等集成，可以不依托云服务器单独使用。对象存储支持大规模数据的访问，但不适合毫秒级响应或随机读写的场景。

云硬盘 需要搭配云服务器，使用文件系统分区或格式化后，才可以被挂载使用。根据云硬盘不同的类型，针对不同的性能指标提供了区别 IOPS 和吞吐性能的产品，可满足单机使用的不同场景。

为何公有读文件的访问链接会失效？

首先从 对象存储控制台 的对象详情页可获取对象地址和签名链接。

如您的文件为公有读文件：

- 当您希望其他人可以一直访问到您的文件时，建议您直接使用对象地址。
- 当您只允许其他人在一定时间内可以访问到您的文件时，建议您直接复制签名链接。签名链接携带签名参数，有效期为1小时。

如您的文件为私有文件：

- 当您希望其他人可以一直访问到您的文件时，建议您将文件访问权限改为公有读私有写，并使用对象地址。
- 当您希望其他人在一定时间内可以访问到您的文件时，建议您直接复制签名链接。签名链接携带签名参数，有效期为1小时。

如何理解 COS 的“文件夹”或“目录”？

对象存储中不存在文件夹和目录的概念，但为了兼顾不同用户的使用习惯，对象存储借鉴传统文件管理的目录结构，在控制台上模拟了“文件夹”的展示方式。更多详情请参见 文件夹和目录 文档。

COS 文件删除后能不能恢复？

对象存储 COS 的数据冗余存储机制是针对服务器等硬件出现故障时需要数据恢复的场景进行设计的。若您主动对 COS 的数据手动删除或进行配置删除后，腾讯云金融专区将按照您的指令删除数据，并且无法恢复。

主动删除的途径有以下几点：

- 通过 COS 控制台删除单个文件、批量删除文件，清空碎片或者存储桶。
- 通过 COSCMD工具删除文件。
- 通过 COS API 或 SDK 删除文件。
- 通过 COS 生命周期管理功能，定期删除文件。
- 通过 COS 跨地域复制的全量同步功能，同步不同地域的存储桶间的新增、修改、删除操作造成同步目标存储桶中同名文件被覆盖、删除。

如何避免误删？

- 对存储桶文件做定时备份操作：
  - 使用 COSCMD工具 将 COS 内对象下载至本地或第三方服务器。
  - 使用跨地域复制功能实现同地域或跨地域的存储桶数据备份。

- 定期使用 COS API、SDK，将数据备份到 COS 的其他存储桶。
- 使用版本控制保存您的历史版本数据。

- 使用 COS 权限管理，参考 访问管理实践：
  - 读写权限分离，对于只需要读数据的业务、只使用具有读权限的子账号或临时密钥进行访问。
  - 存储桶（Bucket）权限分离，针对不同的业务，只授权对应业务范围内的存储桶、目录和操作权限。
  - 不使用主账号访问 COS。
  - 使用临时密钥访问 COS。
  - 妥善保管数据访问的凭据，如账号密码、CAM 子账号访问凭据、API密钥等。

COS 支持数据统计功能吗？

对象存储 COS 提供存储数据的监控能力，用户可通过监控数据窗口了解各数据的状况及趋势。如需查看全盘数据趋势，您可以在 COS 控制台 的【概览】页面，根据不同存储类型的维度，查看其存储量、请求数、流量等数据。 若需查看单一存储桶的数据统计情况，可参见 基础数据统计。

除此之外，您也可以在 云监控 页面查看不同存储桶的监控信息，并根据业务需求配置不同的告警策略。

COS 支持图片压缩吗？

对象存储服务是面向非结构化数据的分布式存储服务，服务本身不支持图片压缩。

COS 支持提供缩略图功能吗？

对象存储服务是面向非结构化数据的分布式存储服务，服务本身不支持图片压缩。

COS 能对视频文件转码吗？

对象存储服务是面向非结构化数据的分布式存储服务，服务本身不支持视频文件转码。

COS 支持文件上传后自动解压吗？

对象存储服务是面向非结构化数据的分布式存储服务，服务本身不支持文件解压。

计费计量

最近更新时间: 2025-01-15 17:01:00

COS 如何收费？

COS 的费用由三部分组成：存储费用、请求费用、流量费用。

对象存储 COS 的计费方式目前提供按量计费方式。

计费项	计费项说明	计费公式
COS 标准存储存储容量	根据存储容量的大小进行计算，不同存储类型的单价不同	存储容量费用 = 存储容量单价 * 日存储容量
COS 标准存储读请求	请求费用根据请求次数进行计算，不同存储类型的请求单价不同	请求费用 = 每万次请求单价 * 日累计请求次数 / 10000
COS 标准存储写请求	请求费用根据请求次数进行计算，不同存储类型的请求单价不同	请求费用 = 每万次请求单价 * 日累计请求次数 / 10000
COS 外网下行流量	外网下行流量	流量费用 = 每 GB 单价 * 日累计流量
COS 跨区域复制流量	跨区域复制流量	流量费用 = 每 GB 单价 * 日累计流量

如何查看账单？

您可以通过控制台费用中心查看您的账户使用对象存储服务所产生的费用信息。

COS 与 CVM 之间传输数据，请求数和流量是否收费？

同地域的 COS 与 CVM 之间传输数据，属于内网传输，流量是免费的。不同地域的 COS 与 CVM 之间传输数据，流量是收费的。

COS 与 CVM 之间传输数据产生的请求次数，不区分地域和内外网，都会计费。

COS 的外网下行流量如何收费？

外网下行流量是数据通过互联网从 COS 传输到客户端产生的流量。直接通过对象链接下载对象或通过静态网站访问节点浏览对象产生的流量为外网下行流量，对应费用为外网下行流量费用。

上传文件到 COS 存储桶是否会产生流量费用？

不会。

同地域内云产品之间相互访问会产生流量费用吗？

在相同地域内，云产品之间访问，将会自动使用内网连接，不产生流量费用

欠费停服后，COS 控制台还能不能访问文件及下载文件？

欠费停服后，将禁止对控制台的访问。在此期间您没有进行续费使账户余额大于等于0，数据将会持续被隔离。

存储在 COS 的对象受到 DDoS 攻击是否产生流量费用？

对象受到 DDoS 不产生流量费用。对象存储的存储桶具有防盗链功能，用户开启防盗链设置后可减少 DDoS 带来的影响。

版权所有：腾讯云金融专区

API与SDK

最近更新时间: 2025-01-15 17:01:00

API 与其他 SDK 问题

调用 API 接口时，出现“Request has expired”等错误信息，该如何处理？

出现该提示，存在两种可能：

- 一是因为您发起请求的时间超过了签名的有效时间。
- 二是您本地系统时间和所在时区的时间不一致。

针对第一种可能，建议重新获取有效的请求签名再进行 API 操作。若是第二种可能，请将您的本地系统时间按照所在时区的时间进行校正。

如何调用 API 删除掉未完成上传文件？

首先调用接口 ListMultipartUploads 列出未完成上传文件，然后调用 Abort Multipart upload 接口舍弃一个分块上传并删除已上传的块。

调用批量删除接口返回正确，但实际文件删除失败怎么办？

请检查删除的文件路径，文件路径不需要以 / 开头。

通过 JSON API 创建的存储桶和上传的对象，是否可以使用 XML API 管理？

可以，XML API 是基于 COS 底层架构，可以通过 XML API 操作由 JSON API 产生的数据。

XML API 与 JSON API 之间的关系？

JSON API 接口即从2016年9月起用户接入 COS 使用的 API，上传域名为 `<Region>.file.myqcloud.com` 。 JSON API 接口将保持维护状态，可以正常使用但是不发展新特性。其与标准 XML API 底层架构相同，数据互通，可以交叉使用，但是接口不兼容，域名不一致。

XML API 与 JSON API 的密钥是否通用？

通用。有关密钥信息可前往 访问管理控制台 中的**云 API 密钥**页面进行查看和获取。

XML API 与 JSON API 的签名是否通用？

不通用，XML API 和 JSON API 各自有各自的签名方式。

XML API 与 JSON API 设置的 ACL 权限是否通用？

不通用，XML API 和 JSON API 各自有各自的 ACL 权限。

如何获取 Python SDK 下载文件的临时链接？

详情请参见 预签名 URL 文档。

SDK 能否使用 CDN 加速域名进行访问？

支持。

小程序 SDK 类问题

小程序里请求多个域名，或者存储桶名称不确定，怎么解决白名单配置和限制问题？

SDK 实例化时，使用 `ForcePathStyle:true` 可以打开后缀式，只需要真正请求 url 格式如下 `http://imgcache.finance.cloud.tencent.com:80cos-ap-beijing.myqcloud.com/<BucketName-APPID>/<Key>` 后缀式请求，在签名时存储桶名称 `<BucketName-APPID>` 也会加入签名计算。

小程序如何保存图片到本地？

先预先通过 `cos.getObjectUrl` 获取图片 url，而后调用 `wx.downloadFile` 下载图片得到临时路径，界面显示保存图片按钮，用户单击按钮后，调用 `wx.saveImageToPhotosAlbum` 保存到相册。

Java SDK 类问题

引入 SDK 运行后，出现 java.lang.NoSuchMethodError 的异常？

原因一般是发生了 JAR 包冲突，例如，用户的工程中的 httpclient 库中 的 JAR 包版本没有A方法，但是 SDK 依赖的 JAR 包使用了 A 方法。此时，由于运行时加载顺序的问题，加载了用户工程中的 httpclient 库，运行时便会抛出 NoSuchMethodError 的异常。 解决方法：将工程中引起 NoSuchMethodError 包的版本，改成和 SDK 中 pom.xml 里的对应库的版本一致。

SDK 上传速度慢，日志频繁打印 IOException？

原因与解决办法：

a. 首先确认下是否是通过公网访问 COS，目前同地域 CVM 访问 COS 走内网（内网域名解析出的 IP 是10、100、169网段），如果是通过公网确认出口带宽是否较小，或者是否有其他程序占用带宽资源。 b. 确保在生产环境中的日志级别不是 DEBUG，推荐使用 INFO 日志。 c. 目前简单上传速度可达10MB，高级 API 在32并发的情况下速度可达60MB,如果速度远低于此两个值，请参考 a 和 b。 d. 如果 WARN 日志打印 IOException 可以忽略，SDK 会进行重试. IOException 的原因可能是网速过慢，原因可参考 a 和 b。

SDK 如何创建目录？

对象存储中文件和目录都是对象，目录只是以 / 结尾的对象。创建文件时，不需要创建目录。如创建一个对象键为 xxx/yyy/zzz.txt 的文件，只用把 key 设置为 xxx/yyy/zzz.txt 即可，不用建立 xxx/yyy/ 这个对象。在控制台上展示时，也会以 / 作为分隔，展示出目录的层级效果。但这些目录对象是不存在的。如果想创建一个目录对象，可使用以下的示例代码：

```
String bucketName = "examplebucket-1250000000";
String key = "folder/images/";
// 目录对象即是一个/结尾的空文件，上传一个长度为 0 的 byte 流
InputStream input = new ByteArrayInputStream(new byte[0]);
ObjectMetadata objectMetadata = new ObjectMetadata();
objectMetadata.setContentLength(0);

PutObjectRequest putObjectRequest =
new PutObjectRequest(bucketName, key, input, objectMetadata);
PutObjectResult putObjectResult = cosClient.putObject(putObjectRequest);
```

SDK 如何使用 HTTPS?

SDK 中相关的配置都统一放在 ClientConfig 类中，示例代码如下：

```
// 初始化用户身份信息(secretId, secretKey)
String secretId = "COS_SECRETID";
String secretKey = "COS_SECRETKEY";
COSCredentials cred = new BasicCOSCredentials(secretId, secretKey);
// 设置 bucket 的区域
ClientConfig clientConfig = new ClientConfig(new Region("ap-beijing-1"));
// 配置使用 https
clientConfig.setHttpProtocol(HttpProtocol.https);
// 生成 cos 客户端
COSClient cosClient = new COSClient(cred, clientConfig);
```

SDK 如何使用代理？

对于需要使用代理访问 COS 的客户，可在 ClientConfig 类中，配置使用代理 IP （或域名）以及端口，示例代码如下：

```
// 初始化用户身份信息(secretId, secretKey)
String secretId = "COS_SECRETID";
String secretKey = "COS_SECRETKEY";
COSCredentials cred = new BasicCOSCredentials(secretId, secretKey);
// 设置 bucket 的区域
ClientConfig clientConfig = new ClientConfig(new Region("ap-beijing-1"));
// 配置使用代理(IP 和端口需要同时设置)
// 设置代理 IP (也可传入域名)
clientConfig.setHttpProxyIp("192.168.2.3");
// 设置代理端口
clientConfig.setHttpProxyPort(8080);
// 生成 cos 客户端
COSClient cosClient = new COSClient(cred, clientConfig);
```

如何设置自定义 EndpointBuilder？

您的场景也许需要指定 API 请求的 Endpoint，此时，您需要实现 EndpointBuilder 接口中的 buildGeneralApiEndpoint 和 buildGetServiceApiEndpoint 中的两个函数，分别为普通 API 请求和 GETService 请求指定远端的 Endpoint。使用示例如下：

```
// 步骤1：实现 EndpointBuilder 接口中的两个函数
class SelfDefinedEndpointBuilder implements EndpointBuilder {
@Override
public String buildGeneralApiEndpoint(String bucketName) {
return String.format("%s.%s", bucketName, "mytest.com");
}

@Override
public String buildGetServiceApiEndpoint() {
return "service.mytest.com";
}
}

// 步骤2：初始化客户端
String secretId = "COS_SECRETID";
String secretKey = "COS_SECRETKEY";
COSCredentials cred = new BasicCOSCredentials(secretId, secretKey);
SelfDefinedEndpointBuilder selfDefinedEndpointBuilder = new SelfDefinedEndpointBuilder();
ClientConfig clientConfig = new ClientConfig(new Region("ap-beijing"));
clientConfig .setEndpointBuilder(selfDefinedEndpointBuilder);
COSClient cosClient = new COSClient(cred, clientConfig);
```

SDK 的上传、下载、批量删除等操作中，使用的 key 值是否需要添加 '/' 前缀？

对象存储的 key 值无需携带 '/' 前缀。例如，您将对象 key 值设置为 exampleobject 上传的对象，可以通过 URL：http://imgcache.finance.cloud.tencent.com:80cos.ap-guangzhou.myqcloud.com/exampleobject 进行访问。

> **注意：**
>
> 在批删请求中，请勿传入 '/' 前缀的 key，这将导致对象删除失败。

工具问题
COSCMD工具

最近更新时间: 2025-01-15 17:01:00

COSCMD 工具是否支持正则表达式？

不支持。

使用 COSCMD 工具，成功创建含有大写字符的存储桶，进行其他操作时使用大写字符报错？

COSCMD 工具会将大写字符自动转换为小写字符，存储桶名称只支持小写字母、数字、中划线及其组合，最多支持50个字符。

使用 COSCMD 工具下载根目录文件，是否支持排除某个目录？

支持。可使用 `coscmd download --ignore /folder/*` 方式过滤。当忽略某一类后缀时，必须最后要输入，或者加入 `""` 。

Hadoop工具

最近更新时间: 2025-01-15 17:01:00

在执行计算任务过程中抛出异常信息 com.qcloud.cos.exception: CosServiceException: Reduce your request rate. (Status Code: 503; Error Code: SlowDown; Request ID: NWXXXXXXXXXX)，该如何处理？

大数据计算任务通常会并行地读取 COS 存储桶中的数据，因而触发了访问频率的限制。COS 默认对每个账号有1200QPS的操作限制，建议增加 `fs.cosn.maxRetries` 配置值，使其通过多次重试来保证作业的正常运行。

在执行计算任务过程中抛出异常信息 java.net.ConnectException: Cannot assign requested address (connect failed) (state=42000,code=40000)，该如何处理？

出现 Cannot assign requested address 错误一般是因为用户在短时间内建立了大量的 TCP 短连接，而连接关闭后，本地端口并不会被立即回收，而是默认经过一个60秒的超时阶段，因此导致客户端在这段时间内，没有可用端口用于与 Server 端建立 Socket 连接。

解决方法：修改 `/etc/sysctl.conf` 文件，调整如下内核参数进行规避：

```
net.ipv4.tcp_timestamps = 1 #打开 TCP 时间戳的支持
net.ipv4.tcp_tw_reuse = 1 #支持将处于 TIME_WAIT 状态的 socket 用于新的 TCP 连接
net.ipv4.tcp_tw_recycle = 1 #启用处于 TIME-WAIT 状态的 socket 的快速回收
net.ipv4.tcp_syncookies=1 #表示开启 SYN Cookies。当出现 SYN 等待队列溢出时，启用 cookie 来处理，可防范少量的 SYN 攻击。默认为0
net.ipv4.tcp_fin_timeout = 10 #端口释放后的等待时间
net.ipv4.tcp_keepalive_time = 1200 #TCP 发送 KeepAlive 消息的频度。缺省是2小时，改为20分钟
net.ipv4.ip_local_port_range = 1024 65000 #对外连接的端口范围。默认是32768至61000，改为1024至65000
net.ipv4.tcp_max_tw_buckets = 10240 #TIME_WAIT 状态 Socket 的数量限制，如果超过了这个数量，新来的 TIME_WAIT 套接字会被直接释放，默认值是180000。适当地降低该参数可以减小处于 TIME_WAIT 状态 Socket 的数量
```

COSBrowser工具

最近更新时间: 2025-01-15 17:01:00

什么是 COSBrowser工具？

COSBrowser 是腾讯云金融专区对象存储 COS 推出的可视化界面工具，让您可以使用更简单的交互轻松实现对 COS 资源的查看、传输和管理。目前 COSBrowser 提供桌面端（Windows、macOS、Linux）和移动端（Android、iOS），详细介绍请参见 COSBrowser 简介。

如何下载 COSBrowser 工具?

下载地址和使用说明请参见 COSBrowser 简介。

子账号登录 COSBrowser，为什么不显示存储路径？

1. 请确认子账号是否有访问 COS 的相关权限，相关文档可参见 授权子账号访问 COS。
2. 若子账号只有某个存储桶或存储桶下某个目录的权限，则子账号在登录 COSBrowser 工具时，需要手动添加存储路径和选择存储桶所在的地域。存储格式路径为 Bucket 或 Bucket/Object-prefix，例如 examplebucket-1250000000。

如何提高大量文件情况下的传输速度？

以 Windows COSBrowser 工具为例，可进入【高级设置】，调整【上传】、【下载】的文件并发数和分块数来提高传输速度。

COS Migration工具

最近更新时间: 2025-01-15 17:01:00

迁移工具中途异常退出怎么办？

工具支持上传时断点续传, 对于一些大文件，如果中途退出或者因为服务故障，可重新运行工具，会对未上传完的文件进行续传。

对于迁移成功的文件，用户通过控制台或其他方式删除了 COS 上的文件，迁移工具会将这些文件进行重新上传吗？

不会。原因是，所有迁移成功的文件会被记录在 db 中，迁移工具运行之前会先扫描 db 目录，对于已被记录的文件不会再次上传，具体原因请参照 迁移机制及流程。

迁移失败，日志显示403 Access Deny，该如何处理？

请确认密钥信息，Bucket 信息，Region 信息是否正确，并且是否具有操作权限。如果是子账号，请让父账号授予相应的权限；如果是本地迁移和其他云存储迁移，需要对 Bucket 具有数据写入和读取权限；如果是 Bucket copy，还需要对源 Bucket 具有数据读取权限。

从其他云存储迁移 COS 失败，显示 Read timed out，该如何处理？

一般来说，这种失败情况是由网络带宽不足所造成，导致从其他云存储下载数据超时。例如，将 AWS 海外的数据迁移到 COS，在下载数据到本地时由于带宽能力不足，导致时延较高，可能会出现 read time out。因此，解决方法为增大机器的网络带宽能力，建议在迁移之前用 wget 测试下载速度。

迁移失败，日志显示503 Slow Down，该如何处理？

这是触发频控所导致，COS 目前对一个账号具有每秒800QPS 的操作限制。建议调小配置中小文件的并发度，并重新运行工具，则会将失败的重新运行。

迁移失败，日志显示404 NoSuchBucket，该如何处理？

请确认您的密钥信息，Bucket 信息，Region 信息是否正确。

运行异常，显示如下的信息该怎么办?

此问题是因为工具使用了 rocksdb，需要使用64位的 JDK，请检查 JDK 版本是 X64的 JDK 。

在 Windows 环境下报找不到 rocksdb 的 jni 库，该如何处理？

在 Windows 环境下，工具需要在 Microsoft Visual Studio 2015环境下编译。若出现以上报错，需安装 Visual C++ Redistributable for Visual Studio 2015。

如何修改日志级别？

修改文件 src/main/resources/log4j.properties，把 log4j.rootLogger 的值复制为对应的日志级别，如 DEBUG、INFO、ERROR。

如遇其他问题，请您尝试重新运行迁移工具。若仍然失败，请将配置信息（密钥信息请隐藏）与 log 目录打包后提交工单。

上传与下载

最近更新时间: 2025-01-15 17:01:00

COS 对上传和下载带宽是否有限制？

COS 不对上传和下载带宽进行限制，具体的上传和下载速度与您本地带宽有关。

如何使文件直接在浏览器中预览，而不是下载？

存储桶域名格式为 `<BucketName-APPID>.cos.<Region>.myqcloud.com` 即为 XML 版本域名。只要是浏览器支持直接预览的文件类型，访问该格式域名对应的对象链接，即可实现在浏览器中预览文件。

存储桶域名格式为 `<BucketName-APPID>.<region>.myqcloud.com` 即为 JSON 版本域名。JSON 版本域名对应的对象链接域在浏览器中访问会弹出下载，想要在浏览器中预览文件，有两种方案：

1. 升级 COS 控制台版本到 新版控制台，使用 XML 版本域名对应的对象链接访问（强烈推荐）。
2. 绑定自定义域名并开启静态网站，使用自定义域名访问。

示例：

以北京地域的 examplebucket-1250000000 存储桶根目录下 picture.jpg 文件为例说明：

- 若对象地址为 `http://imgcache.finance.cloud.tencent.com:80examplebucket-1250000000.cos.ap-beijing.myqcloud.com/picture.jpg` 形式，您可以直接使用该地址在浏览器中预览 picture.jpg 文件。
- 若对象地址为 `http://imgcache.finance.cloud.tencent.com:80examplebucket-1250000000.cosbj.myqcloud.com/picture.jpg` 形式，想要直接在浏览器中预览对象，有两种方案：
  i. 升级 COS 控制台版本到 新版控制台，使用 XML 版本域名对应的对象链接访问（强烈推荐）。
  ii. 绑定自定义域名并开启静态网站，使用自定义域名访问。

如何使文件直接在浏览器中下载，而不是预览？

您可以通过 COS 控制台 将对象自定义 Headers 中的 Content-Disposition 参数值设为 attachment。控制台操作指南请参见 自定义 Headers。

也可以通过设置 GET Object 接口中请求参数 response-content-disposition 的值为 attachment 来实现浏览器中弹出下载文件。参考文档请参见 GET Object。

> **注意：**
>
> 请求中要使用 response-* 参数，则请求必须带签名。

如何判断您是否通过内网访问 COS？

腾讯云金融专区对象存储 COS 的访问域名使用了智能 DNS 解析，通过互联网在不同的运营商环境下，我们会检测并指向最优链路供您访问 COS。如果您在腾讯云金融专区内部署了服务用于访问 COS，则同地域范围内访问将会自动被指向到内网地址，跨地域暂不支持内网访问，默认将会解析到外网地址。

内网访问判断方法

相同地域内云产品访问，将会自动使用内网连接，产生的内网流量不计费。因此选购不同产品时，建议尽量选择相同地域，减少您的费用。

确认是否内网访问请参考如下方法：

以 CVM 访问 COS 为例，判断是否使用内网访问 COS ，可以在 CVM 上 使用 `nslookup` 命令解析 COS 域名，若返回内网 IP，则表明 CVM 和 COS 之间是内网访问，否则为外网访问。

> **说明：**
>
> 内网 IP 地址一般形如 `10.*.*.*` 、 `100.*.*.*` ，VPC 网络一般为 `169.254.*.*` 等。

假设 `examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com` 为目标存储桶地址，其下方的 `Address: 10.148.214.13` 表示从内网访问。

```
nslookup examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com

Server: 10.138.224.65
Address: 10.138.224.65 #53

Name: examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com
Address: 10.148.214.13
Name: examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com
Address: 10.148.214.14
```

如何下载文件夹？

您可登录 COSBrowser 工具，选中需要下载的文件夹，单击【下载】进行文件夹或批量文件下载。或通过 COSCMD 工具实现下载文件夹，详情请参见 COSCMD 工具。

进行上传下载等操作时，报错"403 Forbidden"、权限拒绝等该如何处理？

请按照以下步骤逐步排查问题：

1. 请检查您的以下配置信息是否正确： BucketName、APPID、Region、SecretId、SecretKey 等。
2. 确保上述信息正确的前提下，请检查是否使用子账号操作，若使用子账号请检查主账号是否已对子账号授权。否则，请先登录主账号对子账号授权。授权操作请参见 访问管理权限设置相关案例。
3. 若使用临时密钥进行操作，请检查当前操作是否在获取临时密钥时设置的 Policy 中。否则请修改相关 Policy 设置，详情请参见 临时密钥指引。
4. 若以上步骤仍无法解决问题，请提交工单联系我们。

COS 如何实现批量上传或批量下载文件？

COS 支持通过 API 或 SDK 编程的方式批量操作文件，也提供了命令行工具 COSCMD 和图形化程序 COSBrowser 实现批量操作。

上传文件至存储桶，已存在同名文件，是直接覆盖还是新增不同版本的文件？

COS 现已支持版本控制功能，当存储桶未启用版本控制功能，上传相同名称的文件至存储桶，会直接覆盖已存在的同名文件。当存储桶启用了版本控制功能，上传相同名称的文件至存储桶，会同时存在该对象的多个版本。

COS 分块上传方式，最小分块大小是多少呢？

每块最小1MB。详情请参见 规格与限制 文档。

大文件分块上传过程中，签名失效后是否可以换签名继续上传分块？

可以。

数据安全

最近更新时间: 2025-01-15 17:01:00

密钥问题

在哪里查看 APPID、SecretId、SecretKey 等密钥信息呢？

存储桶名称的后半部分即为 APPID 信息，您可以登录 对象存储控制台 查看。SecretId、SecretKey 等信息，请登录访问管理控制台的 API 密钥管理 中查看。

临时密钥的有效时间是多长？

最长2小时，即7200秒。临时密钥过期后，持有过期临时密钥的请求将会被拒绝。有关临时密钥的介绍请参见 临时密钥生成及使用指引。

密钥相关信息如 APPID、SecretId 等信息泄露了，如何处理？

用户可删除已泄露的密钥，并新建一个密钥。

如何对私有读写的文件生成具有时效性的访问链接？

详情请参见 临时密钥生成及使用指引 文档，设定密钥有效时间。

ACL+ Policy 等权限问题

进行上传下载等操作时，报错"403 Forbidden"、权限拒绝等该如何处理？

请按照以下步骤逐步排查问题：

1. 请检查您的以下配置信息是否正确：

   BucketName、APPID、Region、SecretId、SecretKey 等。

2. 确保上述信息正确的前提下，请检查是否使用子账号操作，若使用子账号请检查主账号是否已对子账号授权。否则，请先登录主账号对子账号授权。

   授权详情请参见 访问管理权限设置相关案例。

3. 若使用临时密钥进行操作，请检查当前操作是否在获取临时密钥时设置的 Policy 中。否则请修改相关 Policy 设置。

4. 若以上步骤仍无法解决问题，请 提交工单 联系我们。

使用存储桶默认域名访问公有读存储桶时会返回文件列表，如何隐藏文件列表信息？

您可以为对应存储桶设置一条 deny anyone 的 Get Bucket 权限。操作步骤如下：

登录对象存储控制台，选择存储桶列表，进入对应存储桶的**权限管理**页面。

方法 1：

1. 找到 **Policy权限设置**，在【图形设置】下单击【添加策略】。
2. 按照下图所示添加对应权限设置，单击【确定】保存。

方法 2：

找到 **Policy 权限设置**，单击【策略语法】>【编辑】，输入以下表达式：

```
{
"Statement": [
{
"Action": [
"name/cos:GetBucket",
"name/cos:GetBucketObjectVersions"
],
"Effect": "Deny",
```

```
"Principal": {
"qcs": [
"qcs::cam::anyone:anyone"
]
},
"Resource": [
"qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/*"
]
}
],
"version": "2.0"
}
```

> **注意：**
>
> 请将“qcs::cos:ap-beijing:uid/1250000000:examplebucket-1250000000/*”中的相关信息进行以下替换：
>
> - “ap-beijing”替换为您的存储桶所在地域。
> - “1250000000”替换为您的 APPID 信息。
> - “examplebucket-1250000000”替换为您的存储桶名称。

> 其中，APPID 为存储桶名称的后半部分，您可以在 对象存储控制台 查看存储桶名称。

COS 的 ACL 限制是针对存储桶还是账号？上传文件时是否可以指定权限？

ACL 限制针对账号。不建议上传文件时指定权限，容易导致 ACL + Policy 策略超过1000条而出现报错。

如何授权协作者访问指定存储桶？

协作者账号是一类特殊的子账号，详情请参见 访问策略语言概述。

多个业务需要对存储桶进行操作，是否可以根据存储桶或其他维度隔离权限？

登录 访问管理控制台，进入用户管理页面，可以给不同的业务开启子账号，并赋予不同的授权操作。

进行上传文件或创建存储桶等操作时，报错“your policy or acl has reached the limit (Status Code: 400; Error Code: PolicyFull)”该如何处理？

COS 每个主账号下存储桶和对象 ACL + Policy 的规则数量最多为1000条，当设置的相关 ACL 或 Policy 策略大于1000条时，会出现此报错，因此建议删除无用的 ACL 或 Policy 策略。

> **注意：**
>
> 我们不建议使用文件级别的 ACL 或 Policy。建议您在调用 API 或 SDK 时，若不需要对文件进行特别的 ACL 控制时， 请将 ACL 相关参数（如 x-cos-acl、ACL 等）置空，保持继承存储桶权限。

如何为子公司或员工创建子账号，并授予特定存储桶的访问权限？

详情请参见 授权子账号访问 COS ，创建子账号并对其授权。

如何授权某些特定子账号只对某个存储桶有操作权限？

若希望子账号只有特定存储桶的操作权限，可以使用子账号添加路径。

如何使用 A 账号对 B 账号授权 A 账号下存储桶的写权限？

详情请参见 ACL 访问控制实践 和 CAM 访问管理实践 进行授权。

防盗链问题

能否设置白名单允许访问，并且浏览器单独打开链接也允许访问？

在设置防盗链时选择允许空 referer，即可在设置白名单的情况下，实现浏览器单独打开链接也可以访问。

设置了存储桶 test 的防盗链白名单，允许 a.com 访问，但是 a.com 下的网页播放器却不能播放存储桶 test 下的视频文件？

网页中使用 Windows Media Player、Flash Player 等播放器播放视频链接时，在请求里的 referer 为空，导致没命中白名单，建议设置白名单时允许空 referer。

签名算法

最近更新时间: 2025-01-15 17:01:00

1 签名与鉴权

腾讯云金融专区移动服务通过签名来验证请求的合法性。开发者通过将签名授权给客户端，使其具备上传下载及管理指定资源的能力。签名分为**多次有效签名**和**单次有效签名**：

- **多次有效签名**：签名中不绑定文件fileid，需要设置大于当前时间的有效期，有效期内此签名可多次使用，有效期最长可设置三个月。
- **单次有效签名**：签名中绑定文件fileid，有效期必须设置为0，此签名只可使用一次，且只能应用于被绑定的文件。

2 签名算法

2.1 获取签名所需信息

生成签名所需信息包括项目ID（appid），空间名称（bucket,文件资源的组织管理单元），项目的Secret ID和Secret Key。获取这些信息的方法如下：

1. 登录对象存储, 进入对象存储空间；
2. 如开发者未创建空间，可添加空间，空间名称（bucket）由用户自行输入；
3. 点击“获取secretKey”，获取Appid，Secret ID和Secret Key；

2.2 拼接签名串

拼接多次有效签名串：

a=[appid]&b=[bucket]&k=[SecretID]&e=[expiredTime]&t=[currentTime]&r=[rand]&f=

拼接单次有效签名串：

a=[appid]&b=[bucket]&k=[SecretID]&e=[expiredTime]&t=[currentTime]&r=[rand]&f=[fileid]

签名串中各字段含义如下：

字段	解释
a	开发者的项目ID，接入对象存储服务创建空间时系统生成的唯一标识项目的ID，即Appid
b	空间名称bucket
k	项目的Secret ID
e	签名的有效期，是一个符合UNIX Epoch时间戳规范的数值，单位为秒；**单次签名时，e必须设置为0**
t	当前时间戳，是一个符合UNIX Epoch时间戳规范的数值，单位为秒，**多次签名时，e应大于t**
r	随机串，无符号10进制整数，用户需自行生成，最长10位
fileid	资源存储的唯一标识，格式为"/appid/bucketname/用户自定义路径或资源名"，**并且需要对其中非'/'字符进行urlencode编码**

> **注意：**
>
> - 拼接单次有效签名的签名串时，过期时间e必须设置为0，以保证此签名只能只能针对固定资源且只能使用一次；
> - 删除文件必须使用单次有效签名；
> - 上传必须使用多次有效签名；

2.3 生成签名

1. COS（对象存储服务）使用 HMAC-SHA1 算法对请求进行签名；

2. 签名串需要使用 Base64 编码。

即生成签名的公式如下：

SignTmp = HMAC-SHA1(SecretKey, orignal)

Sign = Base64(SignTmp.orignal)

其中SecretKey为2.1节获取的项目Secret Key，orignal为2.2节中拼接好的签名串，首先对orignal使用HMAC-SHA1算法进行签名，然后将orignal附加到签名结果的末尾，再进行Base64编码，得到最终的sign。

> 注意：
>
> 此处使用的是标准的Base64编码，不是urlsafe的Base64编码。

3 实例

本节介绍生成签名的算法实例，实例中使用PHP语言，如果开发者使用其他与开发，请使用对应的算法。

3.1 获取签名所需信息

获取得到的签名所需信息如下。

- 项目ID：200001
- 空间名称（bucket）：newbucket
- Secret ID：AKIDxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
- Secret Key：bLcPnl88WU30VY57ipRhSePfPdOfSruK

3.2 拼接签名串

拼接的多次有效签名串如下：

a=200001&b=newbucket&k=AKIDxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx&e=1438669115&t=1436077115&r=11162&f=

拼接的单次有效签名串如下：

a=10001290&b=tencentyun&k=AKIDxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx&e=0&t=1436077115&r=11162&f=tencentyunSignTest

```
$appid = "200001";
$bucket = "newbucket";
$secret_id = "AKIDxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx";
$secret_key = "bLcPnl88WU30VY57ipRhSePfPdOfSruK";
$expired = time() + 60;
$onceExpired = 0;
$current = time();
$rdm = rand();
$userid = "0";
$fileid = "/200001/newbucket/tencent_test.jpg";

$srcStr = 'a='.$appid.'&b='.$bucket.'&k='.$secret_id.'&e='.$expired.'&t='.$current.'&r='.$rdm.'&f=';

$srcStrOnce= 'a='.$appid.'&b='.$bucket.'&k='.$secret_id.'&e='.$onceExpired .'&t='.$current.'&r='.$rdm
.'&f='.$fileid;
```

3.3 生成签名

```
$signStr = base64_encode(hash_hmac('SHA1', $srcStr, $secret_key, true).$srcStr);

$signStrOnce = base64_encode(hash_hmac('SHA1',$srcStrOnce,$secret_key, true).$srcStrOnce);

echo $signStr."\n";

echo $signStrOnce."\n";
```

最终得到的多次有效签名为：

vxzLR6vzMNhBMUVzMTWKUB+LMeVhPTIwMDAwMSZrPUFLSURVZkxVRVVpZ1FpWHFtN0
NWU3NwS0pudWFpSUt0eHFBdiZlPTE0Mzc5OTU3MDQmdD0xNDM3OTk1NjQ0JnI9MjA4 MTY2MDQyMSZmPSZiPW5ld2J1Y2tldA==

单次有效签名为：

f11dDSuw86CR02Ko1INzsZstbRlhPTIwMDAwMSZrPUFLSURVZkxVRVVpZ1FpWHFtN0
NWU3NwS0pudWFpSUt0eHFBdiZlPTAmdD0xNDM3OTk1NjQ1JnI9MTE2NjcxMDc5MiZm

PS8yMDAwMDEvbmV3YnVja2V0L3RlbmNlbnRfdGVzdC5qcGcmYj1uZXdidWNrZXQ=

4 签名适用场景

COS（对象存储服务）对签名的适用场景做了如下限制：

场景	适用签名
下载（不开启token防盗链）	不验证签名
上传	**多次有效签名**
查询目录、文件	多次有效签名
创建目录	多次有效签名
下载（开启token防盗链）	多次有效签名
删除目录、文件	**单次有效签名**
更新目录、文件	单次有效签名

词汇表

最近更新时间: 2025-01-15 17:01:00

APPID

APPID 是云平台账户的账户标识之一，用于关联云资源。在用户成功申请云平台账户后，系统自动为用户分配一个 APPID，一个 APPID 下可以创建多个项目。可通过 云平台控制台 【账号信息】查看 APPID。

存储桶

存储桶即 Bucket ，在 COS 中用于存储对象。一个存储桶中可以存储多个对象。

对象

对象即 Object，COS 中存储的基本单元。

SecretId

SecretId & SecretKey 合称为云 API 密钥，是用户访问云平台 API 进行身份验证时需要用到的安全凭证。SecretId 用于标识 API 调用者身份。一个 APPID 可以创建多个云 API 密钥，若用户还没有云 API 密钥，则需要在 云 API 密钥控制台 新建，否则就无法调用云 API 接口。

SecretKey

SecretId & SecretKey 合称为云 API 密钥，是用户访问云平台 API 进行身份验证时需要用到的安全凭证。SecretKey 是用于加密签名字符串和服务器端验证签名字符串的密钥。一个 APPID 可以创建多个云 API 密钥，若用户还没有云 API 密钥，则需要在 云 API 密钥控制台 新建，否则就无法调用云 API 接口。

项目

项目为一个虚拟概念，用户可以在一个账户下面建立多个项目，每个项目中管理不同的资源。用户可以从控制台访问 项目管理。

版权所有：腾讯云金融专区

API文档