# 对象存储（CSP）

## 产品文档



# 文档目录

- 产品简介
  - 产品概述
  - 功能概览
- 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
- 购买指南
  - 计费概述
  - 购买指引
  - 欠费说明
  - 查看消费明细
- 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
    - 设置跨域访问
    - 设置静态网站
    - 设置生命周期
    - 添加存储桶策略
    - 设置存储桶标签
    - 设置版本控制
    - 设置存储桶加密
    - 设置自定义域名
    - 备份存储桶
    - 设置存储桶事件通知
  - 对象管理
    - 上传对象
    - 下载对象
    - 查看对象信息
    - 搜索对象
    - 设置对象的访问权限
    - 自定义 Headers
    - 删除对象
    - 设置对象标签
    - 还原历史版本对象
    - 文件夹管理
      - 创建文件夹
      - 删除文件夹
  - 监控报表
    - 基础数据统计
    - 返回码统计
- 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
  - 删除多个对象
- 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
- 数据安全
  - 服务端加密概述
  - 存储桶加密概述
- 对象存储权限组合判断逻辑说明

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CSP
  - 临时密钥生成及使用指引

用户工具
- 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

CSP API参考

- 简介
- 访问策略语言概述
- 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
  - 获取存储桶列表
- Bucket接口
  - 基本操作接口
    - 创建存储桶
    - 获取对象列表
    - 检索存储桶及其权限
    - 删除存储桶
  - 访问控制（acl）接口
    - 设置存储桶ACL
    - 获取存储桶ACL
  - 跨域资源共享（cors）接口
    - 设置跨域配置
    - 获取跨域配置
    - 删除跨域配置
  - 生命周期（lifecycle）接口
    - 设置生命周期
    - 查询生命周期
    - 删除生命周期
  - 存储桶策略（policy）接口
    - 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）接口
    - 设置存储桶referer
    - 查询存储桶referer
  - 标签
    - 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
版本控制接口
设置版本控制
查询版本控制
存储桶加密
设置存储桶加密
查询存储桶加密
删除存储桶加密
存储桶事件通知
设置存储桶事件通知
获取存储桶事件通知
Object接口
基本操作接口
上传对象
设置对象复制
获取对象
获取对象元数据
删除单个对象
删除多个对象
预请求跨域配置
访问控制接口
设置对象ACL
获取对象ACL
分块上传接口
初始化分块上传
上传分块
完成分块上传
终止分块上传
查询分块上传
查询已上传块
标签
设置对象标签
查询对象标签
删除对象标签
常见问题
一般性问题
工具问题
COSCMD工具
Hadoop工具
COSBrowser工具
COS Migration工具
词汇表
API文档

# 产品简介
# 产品概述

最近更新时间: 2024-12-19 17:12:00

## CSP 简介

对象存储（Cloud Storage Private，简称：CSP）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。

- CSP 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。
- CSP 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。
- CSP 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: 2024-12-19 17:12:00

在使用 CSP 之前，建议先阅读 CSP 基本概念，了解使用 CSP 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。

CSP 主要提供以下功能：

| 功能 | 描述 |
| --- | --- |
| 存储桶操作 | 支持创建、查询、删除 |
| 对象操作 | 对象/文件夹：上传、查询、下载、复制、粘贴和删除操作 |
| 生命周期 | 对象存储支持用户设定规则，对指定对象在某个时间（天数）后进行自动删除 |
| 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。 |
| 防盗链 | 对象存储支持防盗链配置，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 |
| 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）访问资源 |
| 跨域访问 | CSP 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，CSP 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 |
| 存储桶加密 | 通过对象存储控制台，对存储桶设置服务端加密，实现对新上传到该存储桶的对象默认进行加密。 |
| 自定义域名 | 用户可以对存储桶设置自定义域名。绑定了自定义域名的存储桶可以直接该域名替代Virtual-hosted Style URL中的域名部分访问存储桶（原有访问方式不受影响）。 |
| 存储桶备份 | 存储桶备份功能，能够将本地存储桶中的对象复制到远端存储桶。 例如：有新对象上传至本地存储桶内，该对象将被复制到远端存储桶内。 |
| 事件通知 | 事件通知功能，能够对存储桶中用户所关心的资源操作及时进行消息通知。 例如：有新数据上传到对象存储，重要文件被删除时发送消息到用户的Ckafka消息队列中。 |
| 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，CSP 将检查相应的 ACL 以验证请求者是否拥有所需的访问权限 |
| 查看数据概览 | 对象存储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 |
| 多种管理工具 | CSP 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 |
| 多种 API 和 SDK | API：CSP 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<br>CSP 提供多种开发语言：Android、iOS、C、C++、C#、Go、Java、JavaScript、Node.js、PHP、Python |

# 快速入门
# 基本概念

最近更新时间: 2024-12-19 17:12:00

想要充分利用 CSP，您需要了解一些相关的基本概念和术语。

### 存储桶

存储桶（Bucket）是对象的载体，可理解为存放对象的“容器”。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。相关文档请参见存储桶概述。

### 对象

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）。用户可以通过控制台、API、SDK 等多种方式管理对象。相关文档请参见对象概述。

### 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**【账号中心】>【账号信息】**中查看 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**【访问管理】>【云 API 密钥】**页面新建，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、CSP 所属地标识和对象名组成，通过默认访问地址可寻址 CSP 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: 2024-12-19 17:12:00

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CSP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】**，弹出创建存储桶对话框。
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**。

   存储桶加密功能允许设置存储桶级别的加密方式，将会对开启该开关后上传到该桶的对象的进行加密。后续可以在存储桶详情中修改加密配置。同样该配置只影响更改后上传对象的行为，而不会影响已上传对象的行为。

# 上传对象

最近更新时间: 2024-12-19 17:12:00

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**页签中，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】**。




默认访问权限为**【默认】**，在不设置Policy权限时，即私有读写。

服务端加密选项中，默认为不加密，可选项为SSE-COS加密与SSE-KMS加密（需要开通KMS服务）。

需要说明的是：

1. 如果存储桶加密设置了SSE-COS或SSE-KMS，则此处的"不加密"选项将不会生效，上传对象将以存储桶加密设置为准。
2. 如果选择此处的SSE-COS或SSE-KMS，则实际上传时加密类型以当前设置为准，即会覆盖存储桶加密设置。

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: 2024-12-19 17:12:00

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）或存储桶的**【文件列表】**，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】**，进入文件信息详情页。

## 二、获取对象链接并下载

1. 在文件列表页面的**基本信息**区域中，可以查看文件链接。您可以点击**【对象下载】直接下载；或点击【复制临时链接】**，粘贴至浏览器地址栏访问下载。
2. 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，签名生成方法详情请参考签名算法 。

# 删除对象

最近更新时间: 2024-12-19 17:12:00

当您需要删除存储桶里的对象时,请参考以下步骤：

1. 在存储桶的文件列表中找到您想要删除的对象，单击**【更多操作】>【删除】**，弹出删除文件对话框。
2. 单击**【确认】**即可删除对象。

# 删除存储桶

最近更新时间: 2024-12-19 17:12:00

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. 在存储桶列表页面找到您想要删除的存储桶，单击**【删除】**。
2. 在弹出的对话框中，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: 2024-12-19 17:12:00

通过前面的步骤，可了解如何通过对象存储控制台执行 CSP 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 CSP 的更多基本和高级任务。

# 购买指南

# 计费概述

最近更新时间: 2024-12-19 17:12:00

## 计费方式

**按量计费（后付费）**

对象存储 CSP 的计费方式分为按量计费，本文主要介绍按量计费的计费详情。按量计费方式适用于对象存储提供服务的所有地域。CSP 根据用户的存储容量、请求、流量等计量项用量进行计费，对用户账户按日扣费结算。

## 计费项

CSP 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 |
| --- | --- | --- |
| 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 |
| 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 |
| 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 |
| 下行流量 | 外网下行流量 | 流量费用 = 每 GB 单价 * 日累计流量 |

## 计费周期

对象存储 CSP 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 |
| --- | --- | --- |
| 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 |
| 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 |
| 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 |
| 下行流量 | 日 | 每日对上一日产生的费用进行结算，输出账单 |

版权所有：亿算云平台

# 购买指引

最近更新时间: 2024-12-19 17:12:00

对象存储 CSP 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 CSP 服务时，CSP 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: 2024-12-19 17:12:00

## 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 |
| --- | --- |
| 标准存储存储容量 | 日 |
| 标准存储读请求 | 日 |
| 标准存储写请求 | 日 |
| 下行流量 | 日 |

## 欠费停服

当您的账户发生欠费时，对象存储 CSP 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 CSP 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: 2024-12-19 17:12:00

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 CSP 产品，即可在线查看 CSP 的资源 ID 账单。
4. 单击**【明细账单】**页签，选择CSP产品，即可在线查看CSP的明细账单。

# 控制台指南
# 控制台概述

最近更新时间: 2024-12-19 17:12:00

控制台是用户使用 CSP 的工具之一。用户无需编写代码或运行程序，使用控制台即可直接进行存储桶管理、对象管理等操作。控制台提供的具体功能如下表：

| 控制台菜单 | 功能 |
| --- | --- |
| 存储桶管理 | • 创建与删除<br>• 浏览与查询<br>• 设置访问权限<br>• 设置防盗链<br>• 设置跨域访问<br>• 设置静态网站<br>• 设置生命周期<br>• 添加存储桶策略<br>• 设置存储桶标签<br>• 设置版本控制<br>• 设置存储桶加密<br>• 设置自定义域名<br>• 备份存储桶<br>• 设置存储桶事件通知 |
| 对象管理 | • 上传对象<br>• 下载对象<br>• 查看对象信息<br>• 搜索对象<br>• 设置对象的访问权限<br>• 自定义Headers<br>• 删除对象<br>• 设置对象标签<br>• 还原历史版本对象<br>• 创建文件夹<br>• 删除文件夹 |
| 监控报表 | • 基础数据统计<br>• 返回码统计 |

# 存储桶管理
# 创建与删除

最近更新时间: 2024-12-19 17:12:00

## 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：

**操作步骤**

**1. 进入存储桶列表**

登录CSP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。

**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 50 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。
- 为避免因对象数过多导致索引分片过大进而影响业务访问性能，新建存储桶时，自动设置对象数配额为2000万。

**访问权限**

存储桶默认提供三种访问权限：私有读写、获取对象列表和获取对象列表和写入数据。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。
- 获取对象列表：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。
- 获取对象列表和写入数据：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。

## 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。

- 在存储桶列表页面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。
- 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: 2024-12-19 17:12:00

## 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，输入存储桶的名称查询，支持前缀匹配查询。

# 设置访问权限

最近更新时间: 2024-12-19 17:12:00

## 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、获取对象列表和获取对象列表和写入数据。

- 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。
- 获取对象列表：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。
- 获取对象列表和写入数据：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CSP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. 在左侧导航树中，选择**【权限管理】>【存储桶访问权限】**，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: 2024-12-19 17:12:00

## 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。

## 操作步骤

1. 登录CSP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，设置完成后单击**【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.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， 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://www.example.com/index.html 图片显示正常。 访问 http://www.fake.com/index.html 图片也显示正常。

**开启后**

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

# 设置跨域访问

最近更新时间: 2024-12-19 17:12:00

## 简介

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。

## 设置说明

### Allow-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://www.abc.com` 。
- 支持二级泛域名，形如 `http://*.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。

### Allow-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-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，例如：x-cos-meta-md5。

- 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

- 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。

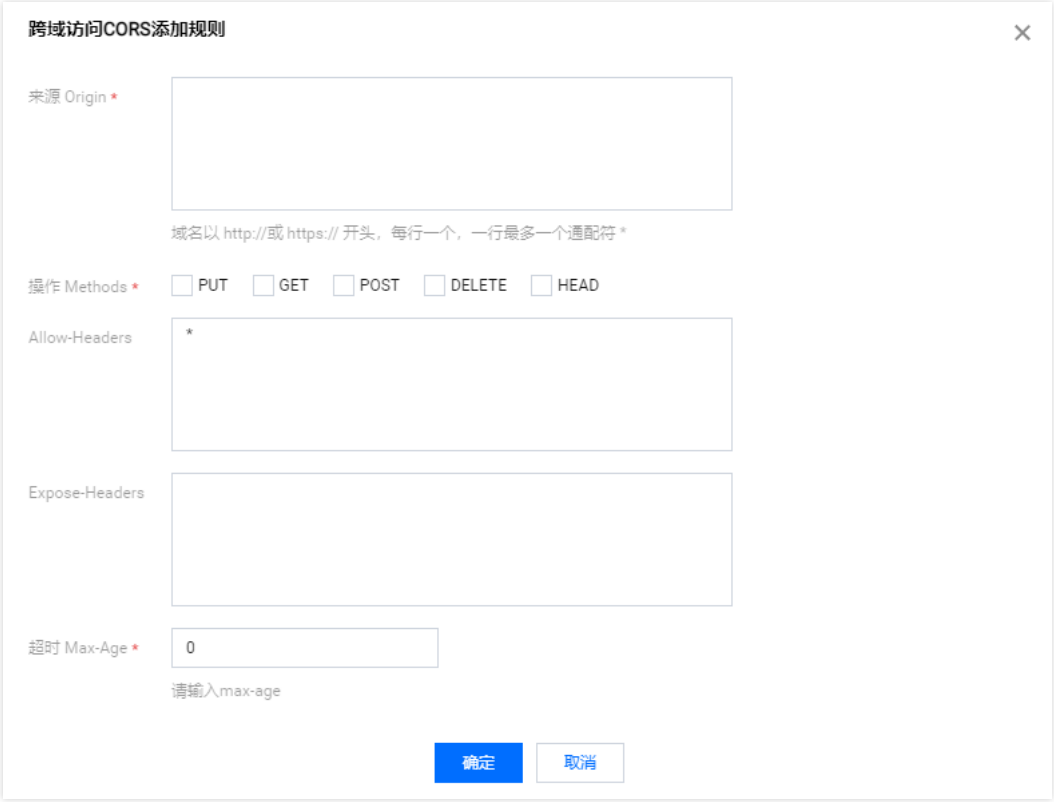
### Max-Age

设置 OPTIONS 请求得到结果的有效期。 例如：600

## 设置步骤

控制台提供了响应 OPTIONS 请求的配置，支持多条规则。

1. 登录CSP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. 单击需要配置回源的存储桶名称，进入存储桶的文件列表页面。
4. 单击**【安全管理】>【跨域访问CORS设置】，单击【添加规则】**。
5. 在弹出的窗口中，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://www.abc.com 。
- 支持二级泛域名，形如 http://*.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 CSP 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】**。

# 设置静态网站

最近更新时间: 2024-12-19 17:12:00

## 简介

您可以通过CSP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中对象的访问权限设置为**公有读**。
> - 开启静态网站配置后，您需要使用静态网站域名访问 CSP 源站才能生效，如果使用 CSP 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。

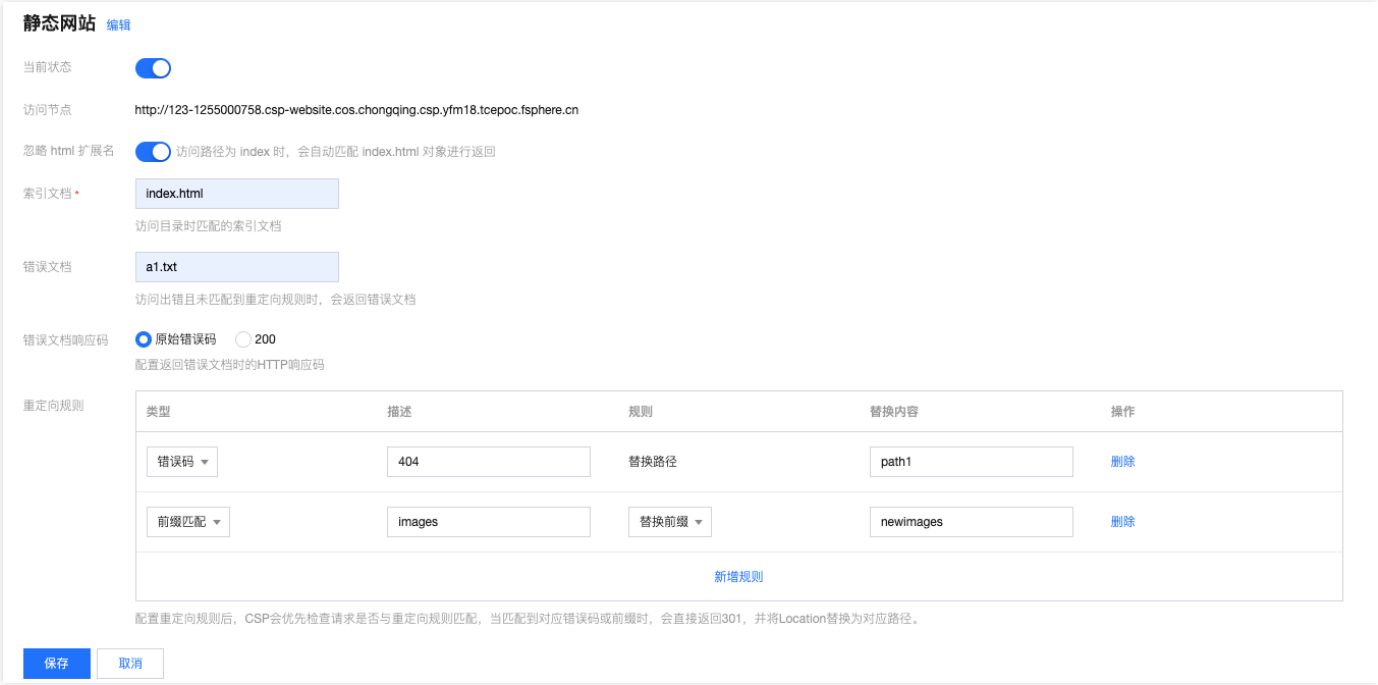
## 操作步骤

1. 登录CSP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**【文件列表】>** ***选择要访问的文件*** **>【详情】**，选择**【对象访问权限】>【公共权限】>【公有读】**并保存。




3. 选择**【基础配置】>【静态网站】，单击【编辑】，把**当前状态**的开启按钮打开，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。

# 设置生命周期

最近更新时间: 2024-12-19 17:12:00

## 简介

当您需要定期对指定对象进行存储类型转换或删除以降低成本时，您可以使用生命周期管理功能。CSP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存储类型转换或删除。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。
> - 开启多 AZ 配置的存储桶当前仅支持设置指定时间后过期删除，暂不支持沉降到低频或者归档的功能。

## 操作步骤

1. 登录CSP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，也可以作用于存储桶中某些带有特定前缀的对象，例如 example。当选择“指定前缀”时，则需要填写对象前缀。
- **对象前缀**：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。
- **管理当前版本文件**：设置文件上传后删除的时间点。

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。

# 添加存储桶策略

最近更新时间: 2024-12-19 17:12:00

## 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 CSP 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CSP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. 单击**【权限管理】>【Policy 权限设置】，CSP 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。



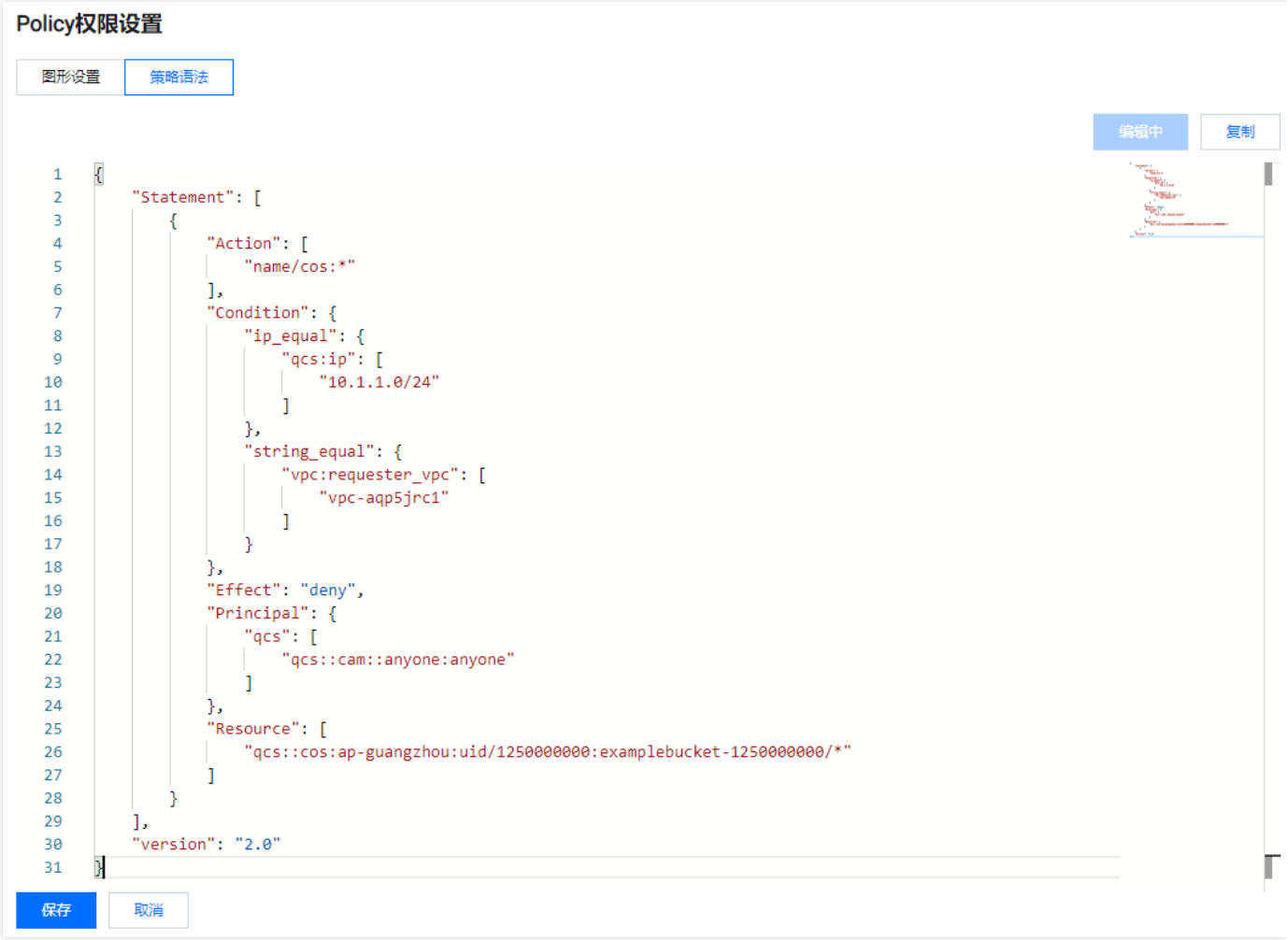

- **图形设置**

  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 CSP 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: 2024-12-19 17:12:00

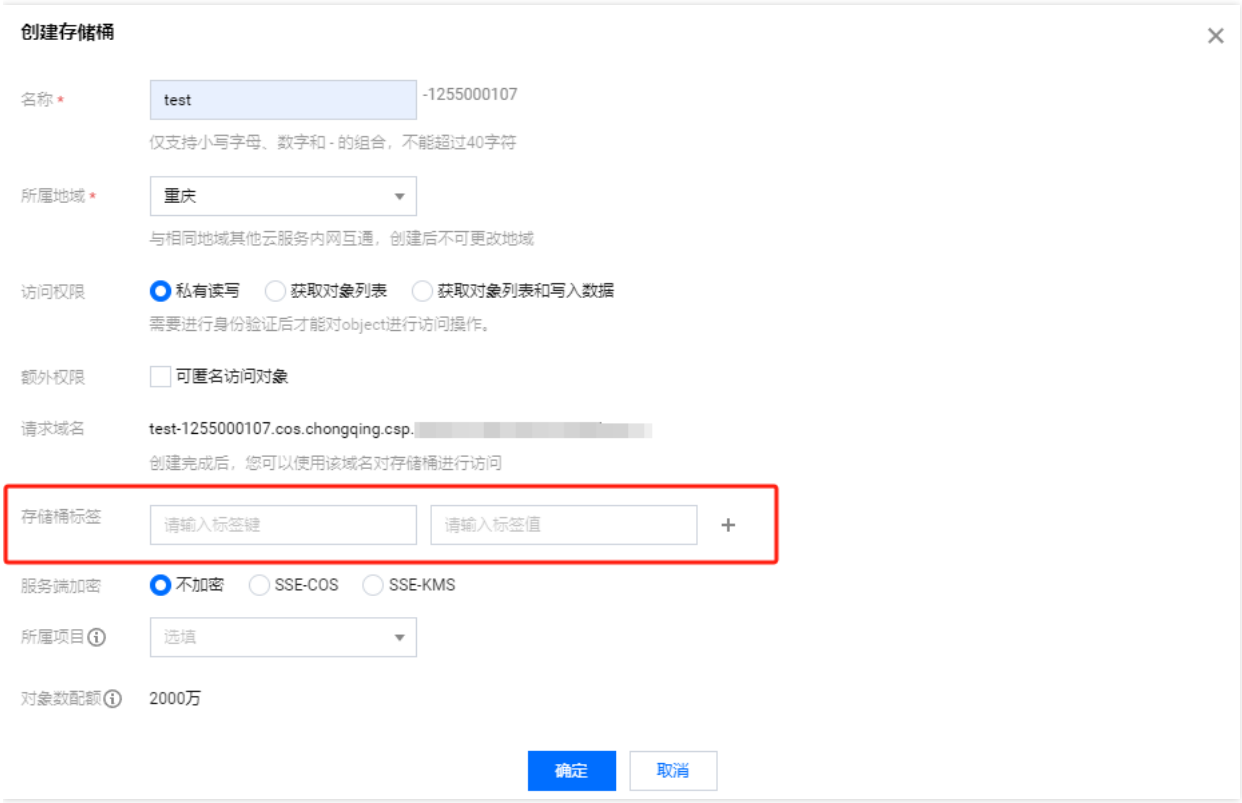
## 简介

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. 单击左侧的**【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】**。

# 设置版本控制

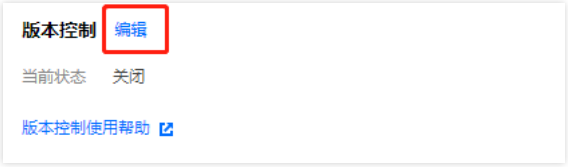
最近更新时间: 2024-12-19 17:12:00

## 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CSP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】**按钮，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，将当前的“开启”按钮修改为“关闭”即可。

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，开启**列出历史版本**，即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。

# 设置存储桶加密

最近更新时间: 2024-12-19 17:12:00

## 简介

您可以通过CSP控制台，对存储桶设置服务端加密，这样可以实现对新上传到该存储桶的对象默认进行加密。关于存储桶加密的详细信息，请参见存储桶加密概述。

目前存储桶的加密方式支持 SSE-COS 加密（即由 CSP 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）。

关于服务端加密的介绍，请参见服务端加密概述。

## 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】**。关于这两点的区别，请详见服务端加密概述。







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：



> 注意：
>
> 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: 2024-12-19 17:12:00

## 简介

您可以通过对象存储控制台，对存储桶设置自定义域名。绑定了自定义域名的存储桶可以直接该域名替代 Virtual-hosted Style URL 中的域名部分访问存储桶（原有访问方式不受影响）。

## 设置说明

1. 登录CSP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，选择**【域名管理】**。
4. 在**自定义域名**区域，单击**【添加域名】**。




5. 输入需要设置的自定义域名后，单击**【保存】**按钮将结果保存。

# 备份存储桶

最近更新时间: 2024-12-19 17:12:00

## 简介

存储桶备份功能，能够将本地存储桶中的对象复制到远端存储桶。

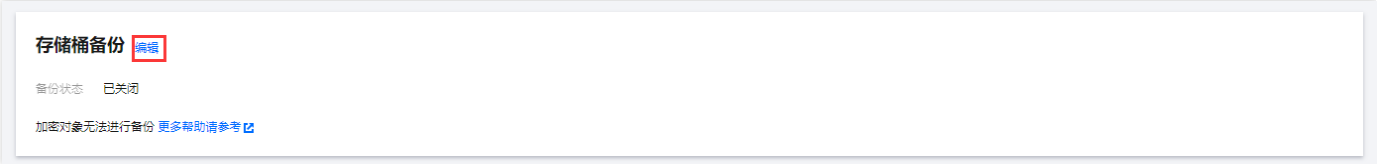
例如：有新对象上传至本地存储桶内，该对象将被复制到远端存储桶内。

## 前提条件

已经在本地和远端创建好存储桶，存储网关与远端对象存储集群网络互通。

## 操作步骤

1. 登录CSP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. 单击**【容错容灾管理】>【存储桶备份】，单击【编辑】**。




5. 根据您的需求添加存储桶备份规则。




配置项说明如下：

- **备份状态**：备份开关。
- **镜像回源**：开启后本地文件下载失败后会内部重定向至远端存储桶下载。
- **存储桶名称**：远端的存储桶名称。
- **对象存储域名**：远端的对象存储系统域名。
- **SecretId**：远端的存储桶所有者的SecretId。
- **SecretKey**：远端的存储桶所有者的SecretKey。
- **启用HTTPS**：采用HTTPS进行对象备份。

6. 设置完成后，单击**【保存】**。

   如需修改，可单击**【编辑】**进行设置。

> **建议：**
>
> 在配置完成后建议主动往本地存储桶内上传若干个对象，进行备份测试。

# 设置存储桶事件通知

最近更新时间: 2024-12-19 17:12:00

## 简介

事件通知功能，能够对存储桶中用户所关心的资源操作及时进行消息通知。

例如：有新数据上传到对象存储，重要文件被删除时发送消息到用户的Ckafka消息队列中。

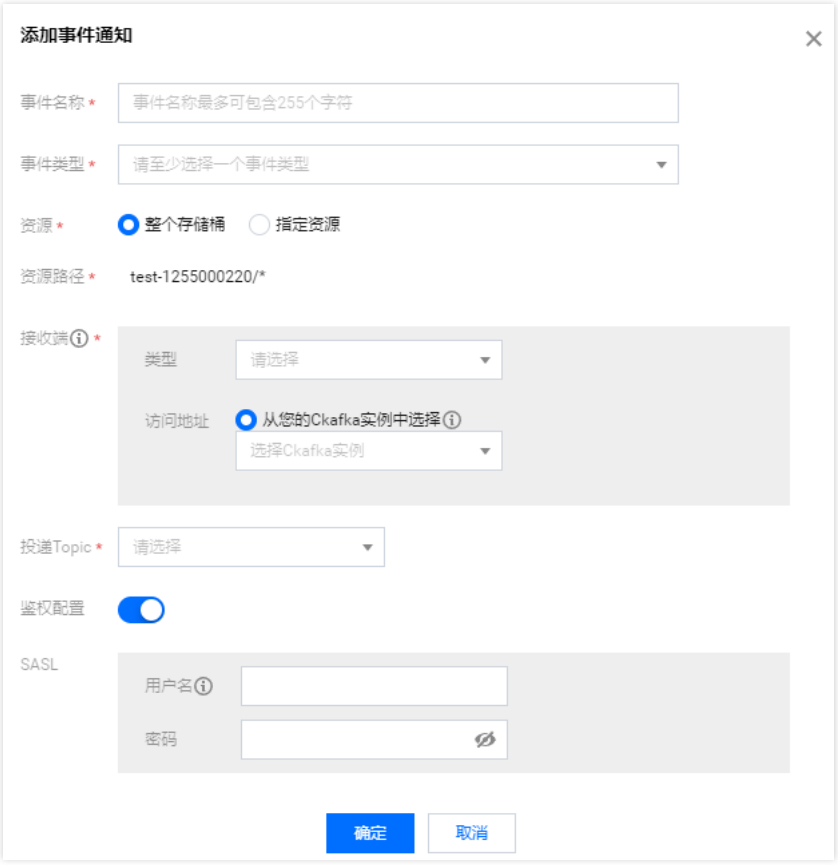
> **说明：**
>
> 一个存储桶下最多支持50条事件通知规则。

## 前提条件

已开通Ckafka服务并且创建实例，并配置topic。

## 操作步骤

1. 登录CSP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. 单击**【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。

   配置项说明如下：

- **事件名称**：输入您的事件名称。
- **事件类型**：选择需要配置的事件类型。
- **资源**：本事件规则可以作用于整个存储桶，也可以作用于存储桶中某些带有特定前缀的对象，例如 alarm。当选择"指定前缀"时，则需要填写对象前缀。资源路径根据资源自动生成。
- **接收端**：类型仅支持kafka，选择您对应的Ckafka实例。
- **投递Topic**：选择您对应的topic名称。
- **鉴权配置**：以您的Ckafka实例是否开启SASL鉴权为准，如果开启请填写正确的用户名和密码。

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。

> 建议：
>
> 在配置完成后建议进行一次对应事件操作进行测试。

# 对象管理
# 上传对象

最近更新时间: 2024-12-19 17:12:00

## 上传步骤

**1. 进入文件列表**

1. 登录CSP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**页签中，单击**【上传文件】**，如下图所示。




**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、服务端加密、元数据信息（可选），单击**【上传】**。




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。

## 上传说明

- 对象访问权限

  可以为不同的对象设置不同的访问权限，默认访问权限为**【默认】**，在不设置Policy权限时，即私有读写。

- 文件数量和容量大小限制

  每个存储桶对存储的文件数量无限制。通过对象存储控制台上传文件，单个文件最大支持 50 TB。

# 下载对象

最近更新时间: 2024-12-19 17:12:00

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CSP控制台。
2. 在左侧导航树中，选择**【存储桶列表】**，单击相应存储桶名称，进入存储桶的文件列表。
3. 在文件列表中找到需要下载的对象，单击**【详情】**，进入文件信息详情页。

## 二、获取对象链接并下载

1. 在文件列表页面的**基本信息**区域中，可以查看文件链接。您可以点击**【对象下载】直接下载；或点击【复制临时链接】**，粘贴至浏览器地址栏访问下载。
2. 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。

# 查看对象信息

最近更新时间: 2024-12-19 17:12:00

通过控制台可以查看对象的具体信息。具体步骤如下：

1. 登录CSP控制台。
2. 选择**【存储桶列表】**，单击相应存储桶名称，进入存储桶的文件列表。
3. 单击对象右侧的**【详情】**，即可获取对象的具体信息。

# 搜索对象

最近更新时间: 2024-12-19 17:12:00

用户可通过控制台对上传的对象和目录进行前缀搜索。

1. 登录CSP控制台。
2. 在左侧导航树中，选择**【存储桶列表】**，单击相应存储桶名称，进入存储桶的文件列表。
3. 在文件列表的右上角搜索框中输入搜索前缀，即可搜索存储桶根目录下的对象或文件夹。

对象存储控制台支持多级目录搜索。若要搜索文件夹内的对象，可在搜索框内输入完整路径以及对象前缀进行搜索。也可进入文件夹再进行前缀搜索。

例如： 需要搜索存储在 存储桶 example-1253833564 的test文件夹的 photo.jpg 对象时，在搜索框内输入“test/p”，单击搜索图标即可显示搜索结果。

# 设置对象的访问权限

最近更新时间: 2024-12-19 17:12:00

## 简介

CSP 提供基于对象维度的访问权限设置，且该配置优先级高于存储桶的访问权限。

> **说明：**
>
> - 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。

通过设置对象的访问权限，可以实现例如：在私有读写的存储桶中设置个别允许公有访问的对象，或在公有读写存储桶中设置个别需要鉴权才可以访问的对象。

CSP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：**
  - 默认：当访问对象时，若用户未设置policy策略，则默认权限为私有读，若设置了policy策略，则按照policy策略进行鉴权。
  - 私有读：当访问对象时，CSP 读取到对象的权限为私有读，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：当访问对象时，CSP 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：**
  - 根账号默认拥有对象所有权限（即完全控制）。
  - 子账号可拥有数据读取、权限读取、权限写入，甚至完全控制的最高权限。

## 设置步骤

1. 登录CSP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. 单击需要修改权限的对象所属存储桶名称，进入存储桶。
4. 找到需要设置权限的对象（如 example.exe）,单击对象右侧的**【详情】**，在详情页面设置权限。
5. 修改访问权限，单击**【保存】**保存即可。

# 自定义 Headers

最近更新时间: 2024-12-19 17:12:00

## 简介

对象的 HTTP 头部（Header ）是服务器以 HTTP 协议传 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。

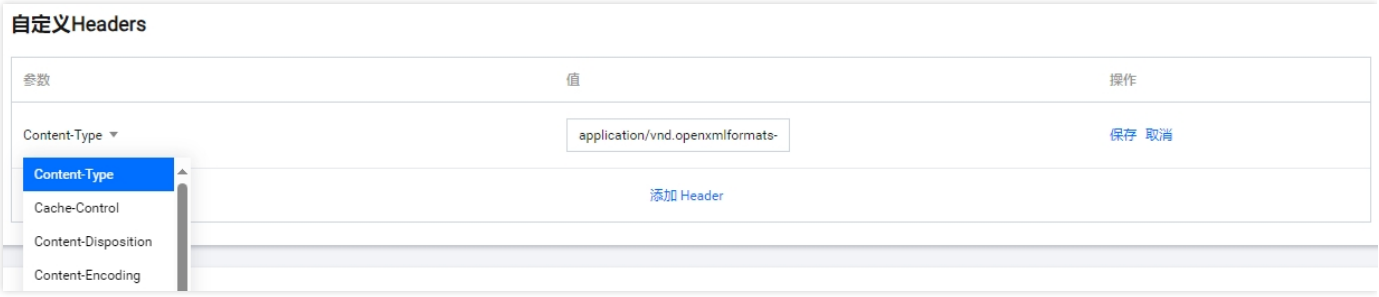
## 配置详情

CSP 提供了 5 种对象 HTTP 头部标识供配置：

| HTTP 头部 | 说明 | 示例 |
| --- | --- | --- |
| Cache-Control | 文件的缓存机制 | no-cache;max-age=200 |
| Content-Type | 文件的 MIME 信息 | text/html |
| Content-Disposition | MIME 协议的扩展 | attachment;filename="fname.ext" |
| Content-Language | 文件的语言 | zh-CN |
| Content-Encoding | 文件的编码格式 | UTF-8 |
| x-cos-meta-[自定义内容] | 自定义内容 | 自定义内容 |

## 配置步骤

1. 登录CSP控制台。
2. 在左侧导航树中，选择**【存储桶列表】**，单击相应存储桶名称，进入存储桶的文件列表。
3. 找到需要设置头部的对象（如 example.exe ）,单击对象右侧的**【详情】**，在详情页面设置权限。
4. 在**自定义Headers**配置项中，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。单击**【保存】**即可。




## 示例

在 APPID 为 123456790 ，创建存储桶名称为 example。存储桶根目录下上传了对象 example.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：

```
> GET /example.txt HTTP/1.1
> Host: example-1234567890.file.myqcloud.com
```

```
> Accept: */*

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

添加如下配置：

**自定义Headers**

| Parameter | Value | Action |
| --- | --- | --- |
| Cache-Control | no-cache | 编辑 删除 |
| Content-Type | attachment;filename*="abc.txt" | 编辑 删除 |
| Content-Type | image/jpeg | 编辑 删除 |
| x-cos-meta-md5 | 1234 | 编辑 删除 |
| | 添加 Header | |

再次发起请求，浏览器或客户端得到的对象头部范例如下：

```
> GET /example.txt HTTP/1.1
> Host: example-1234567890.file.myqcloud.com
> Accept: */*

< 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
```

版权所有：亿算云平台

# 删除对象

最近更新时间: 2024-12-19 17:12:00

## 简介

您可以通过控制台对上传到存储桶中的单个或多个对象进行删除。

## 操作步骤

**删除单个对象**

1. 登录CSP控制台 。
2. 选择**【存储桶列表】**，单击相应存储桶名称，进入存储桶的文件列表。

- 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】**，弹出删除文件对话框。

- 列出历史版本

  找到您想要删除的对象，单击右侧的**【删除】**，弹出删除文件对话框。

3. 单击**【确认】**即可删除对象。

**删除多个对象**

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】**，弹出删除文件对话框。
2. 单击**【确认】**即可删除对象。

# 设置对象标签

最近更新时间: 2024-12-19 17:12:00

## 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。

   如果您需要修改或删除标签，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: 2024-12-19 17:12:00

## 简介

您可以通过CSP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。<br>• 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。<br>• 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。<br>• 最新版本：无法还原。<br>• 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CSP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。

# 文件夹管理
# 创建文件夹

最近更新时间: 2024-12-19 17:12:00

资源在 CSP 中都是以对象的形式存储的，为延续用户使用习惯，在 CSP 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CSP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】**，弹出创建文件夹对话框。
5. 输入文件夹名称，单击**【确定】**保存即可。

版权所有：亿算云平台

# 删除文件夹

最近更新时间: 2024-12-19 17:12:00

## 步骤

1. 登录CSP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】**，弹出删除文件提示框。
5. 单击**【确认】**，删除文件夹。

# 监控报表
# 基础数据统计

最近更新时间: 2024-12-19 17:12:00

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。CSP 的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。

1. 登录CSP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项目和存储桶，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、读请求统计和写请求统计等。

3. 单击日期栏右侧的下载按钮，可导出对应的统计数据。

   数据可能存在延迟，可刷新页面重新加载。

# 返回码统计

最近更新时间: 2024-12-19 17:12:00

1. 登录CSP控制台。
2. 在左侧导航栏中，选择**【监控报表】>【返回码统计】**，进入返回码统计页面。




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。

返回码统计页面支持切换项目和存储桶，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 CSP 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。

3. 单击日期栏右侧的下载按钮，可导出对应的统计数据。

   数据可能存在延迟，可刷新页面重新加载。

# 开发者指南
# 存储桶
# 存储桶概述

最近更新时间: 2024-12-19 17:12:00

## 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，其在亿算云平台具有唯一性。
- 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、获取对象列表和获取对象列表和写入数据。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。

- 获取对象列表

  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。

- 获取对象列表和写入数据

  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 CSP 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1000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: 2024-12-19 17:12:00

## 适用场景

在开始使用 CSP 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，可使用的地域名称请参考 可用地域文档。

## 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 CSP 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
ClientConfig clientConfig = new ClientConfig(new Region("ap-beijing-1"));
// 3 生成csp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: 2024-12-19 17:12:00

## 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 CSP 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
ClientConfig clientConfig = new ClientConfig(new Region("ap-beijing-1"));
// 3 生成csp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: 2024-12-19 17:12:00

## 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

CSP 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 |
| 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。 使用主密钥加密填写：AES256 |

## 对象子资源

CSP 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 CSP 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 CSP 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**：包括默认、私有读和公有读。

- 默认：当访问对象时，若用户未设置policy策略，则默认权限为私有读，若设置了policy策略，则按照policy策略进行鉴权。
- 私有读：当访问对象时，CSP 读取到对象的权限为私有读，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：当访问对象时，CSP 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 CSP 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**

在私有读的存储桶中对特定对象设置允许公有访问，或在公有读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: 2024-12-19 17:12:00

## 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 CSP 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 CSP。

#### 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 CSP 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
ClientConfig clientConfig = new ClientConfig(new Region("ap-beijing-1"));
// 3 生成csp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CSP，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
ClientConfig clientConfig = new ClientConfig(new Region("ap-beijing-1"));
// 3 生成csp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: 2024-12-19 17:12:00

## 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。CSP 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 CSP 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 REST 接口文档部分：

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

### 使用 Java SDK

对象存储 CSP 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
ClientConfig clientConfig = new ClientConfig(new Region("ap-beijing-1"));
// 3 生成csp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
ClientConfig clientConfig = new ClientConfig(new Region("ap-beijing-1"));
// 3 生成csp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
ClientConfig clientConfig = new ClientConfig(new Region("ap-beijing-1"));
// 3 生成csp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
ClientConfig clientConfig = new ClientConfig(new Region("ap-beijing-1"));
// 3 生成csp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
ClientConfig clientConfig = new ClientConfig(new Region("ap-beijing-1"));
// 3 生成csp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
ClientConfig clientConfig = new ClientConfig(new Region("ap-beijing-1"));
// 3 生成csp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: 2024-12-19 17:12:00

## 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 CSP 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
ClientConfig clientConfig = new ClientConfig(new Region("ap-beijing-1"));
// 3 生成 csp 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: 2024-12-19 17:12:00

## 适用场景

您可以直接发起请求获取 CSP 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 CSP 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
ClientConfig clientConfig = new ClientConfig(new Region("ap-beijing-1"));
// 3 生成csp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
ClientConfig clientConfig = new ClientConfig(new Region("ap-beijing-1"));
// 3 生成csp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
ClientConfig clientConfig = new ClientConfig(new Region("ap-beijing-1"));
// 3 生成csp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: 2024-12-19 17:12:00

## 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 CSP 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
ClientConfig clientConfig = new ClientConfig(new Region("ap-beijing-1"));
// 3 生成csp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
ClientConfig clientConfig = new ClientConfig(new Region("ap-beijing-1"));
// 3 生成csp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
ClientConfig clientConfig = new ClientConfig(new Region("ap-beijing-1"));
// 3 生成csp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: 2024-12-19 17:12:00

## 适用场景

云平台 CSP 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，CSP 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 CSP 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 CSP 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
ClientConfig clientConfig = new ClientConfig(new Region("ap-beijing-1"));
// 3 生成csp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
ClientConfig clientConfig = new ClientConfig(new Region("ap-beijing-1"));
// 3 生成csp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: 2024-12-19 17:12:00

## 适用场景

您可以在 CSP 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 CSP 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
ClientConfig clientConfig = new ClientConfig(new Region("ap-beijing-1"));
// 3 生成csp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: 2024-12-19 17:12:00

## 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 CSP 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 CSP 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
ClientConfig clientConfig = new ClientConfig(new Region("ap-beijing-1"));
// 3 生成csp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: 2024-12-19 17:12:00

## 适用场景

云平台 CSP 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 CSP 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
ClientConfig clientConfig = new ClientConfig(new Region("ap-beijing-1"));
// 3 生成csp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();
```

# 删除多个对象

最近更新时间: 2024-12-19 17:12:00

## 适用场景

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

默认情况下，当删除任务都成功完成时，返回的内容通常为空。若有发生错误，则会返回错误的信息。

**注意：单次请求最多可删除 1000 个对象，若需要删除更多对象，请将列表拆分后分别发送请求。**

## 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 Multiple Object 文档说明。

### 使用 Java SDK

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

#### 步骤说明

1. 初始化客户端 cosclient。
2. 执行 deleteObjects 方法删除对象，需提供要删除的对象键名称。
3. 执行成功会返回 DeleteObjectsResult 对象，包含所有已删除的对象键。如果部分成功部分失败（如对该对象没有删除权限），则返回 MultiObjectDeleteException 类。其他失败导致的异常返回异常类（CosClientException/CosServiceException），请参照 SDK 异常类说明。

#### 代码示例

1. 以下代码演示了删除多个对象（无版本控制）的示例代码：

```
// 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
ClientConfig clientConfig = new ClientConfig(new Region("ap-beijing-1"));
// 3 生成csp客户端
COSClient cosclient = new COSClient(cred, clientConfig);
// bucket名需包含appid
String bucketName = "mybucket-1251668577";

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("/aaa.txt"));
keyList.add(new KeyVersion("/bbb.mp4"));
keyList.add(new KeyVersion("/ccc/ddd.jpg"));
deleteObjectsRequest.setKeys(keyList);

// 批量删除文件
try {
DeleteObjectsResult deleteObjectsResult = cosclient.deleteObjects(deleteObjectsRequest);
List<DeletedObject> deleteObjectResultArray = deleteObjectsResult.getDeletedObjects();
} catch (MultiObjectDeleteException mde) { // 如果部分产出成功部分失败, 返回MultiObjectDeleteException
List<DeletedObject> deleteObjects = mde.getDeletedObjects();
List<DeleteError> deleteErrors = mde.getErrors();
} catch (CosServiceException e) { // 如果是其他错误, 比如参数错误， 身份验证不过等会抛出CosServiceException
e.printStackTrace();
} catch (CosClientException e) { // 如果是客户端错误，比如连接不上CSP,会抛出CosClientException
e.printStackTrace();
}
```

2. 以下代码演示了删除多个对象（含有版本控制）的示例代码：

```
// 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
ClientConfig clientConfig = new ClientConfig(new Region("ap-beijing-1"));
// 3 生成csp客户端
COSClient cosclient = new COSClient(cred, clientConfig);
// bucket名需包含appid
String bucketName = "mybucket-1251668577";

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("/aaa.txt", "axbefagagaxxfafa"));
keyList.add(new KeyVersion("/bbb.mp4", "awcafa1faxg0lx"));
keyList.add(new KeyVersion("/ccc/ddd.jpg", "kafa1kxxaa2ymh"));
deleteObjectsRequest.setKeys(keyList);

// 批量删除文件
try {
DeleteObjectsResult deleteObjectsResult = cosclient.deleteObjects(deleteObjectsRequest);
List<DeletedObject> deleteObjectResultArray = deleteObjectsResult.getDeletedObjects();
} catch (MultiObjectDeleteException mde) { // 如果部分产出成功部分失败, 返回MultiObjectDeleteException
List<DeletedObject> deleteObjects = mde.getDeletedObjects();
List<DeleteError> deleteErrors = mde.getErrors();
} catch (CosServiceException e) { // 如果是其他错误, 比如参数错误， 身份验证不过等会抛出CosServiceException
e.printStackTrace();
} catch (CosClientException e) { // 如果是客户端错误，比如连接不上CSP,会抛出CosClientException
e.printStackTrace();
}
```

# 数据管理
# 生命周期管理
# 生命周期概述

最近更新时间: 2024-12-19 17:12:00

CSP 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，亿算云平台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。

> **注意：**
>
> CSP 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，CSP 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**

CSP 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: 2024-12-19 17:12:00

## 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>
**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>
**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>
**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>
**expiration actions**
</Rule>
</LifecycleConfiguration>
```

**合并使用多个筛选条件**

CSP 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>
**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。

**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，因此对象存储对非当前版本的推向提供了如下操作： 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: 2024-12-19 17:12:00

## 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：

- 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 REST 接口文档：

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

# 托管静态网站

最近更新时间: 2024-12-19 17:12:00

## 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。CSP 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。

```
https://examplebucket-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。

```
https://examplebucket-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 `https://examplebucketbucket-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://www.examplebucket.com/
http://www.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。

```
https://examplebucket-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。

```
https://examplebucket-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: 2024-12-19 17:12:00

## 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 。

**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

# 数据安全
# 服务端加密概述

最近更新时间: 2024-12-19 17:12:00

## 概述

对象存储 CSP 在数据写入数据中心内的磁盘之前，支持在对象级别上应用数据加密的保护策略，并在访问数据时自动解密。加密和解密这一操作过程都是在服务端完成，这种服务端加密功能可以有效保护静态数据。

> 注意：
>
> - 访问加密对象与访问未加密的对象在体验上并无差别，但前提是用户已拥有对象的访问权限。
> - 服务端加密仅加密对象数据而不加密对象元数据，而且使用服务端加密的对象必须使用有效签名访问，不可被匿名用户访问。

## 适用场景

- **私密数据存储场景：**对于私密数据的存储，服务端加密可以对存储的数据进行加密，保证用户的隐私，用户访问时会自动解密。
- **私密数据传输场景：**对于私密数据的传输，CSP 提供用 HTTPS 部署 SSL 证书实现加密的功能，在传输链路层上建立加密层，确保数据在传输过程中不会被窃取及篡改。

## 加密方式

CSP 支持多种服务端加密方式：SSE-COS、SSE-KMS、SSE-C。用户可以自行选择合适的加密方式对存放到 CSP 中的数据进行加密。

### SSE-COS 加密

SSE-COS 加密即 CSP 托管密钥的服务端加密。由 CSP 托管主密钥和管理数据。用户通过 CSP 直接对数据进行管理和加密。SSE-COS 采用了多因素强加密，确保使用唯一的密钥加密每个对象，同时采用 256 位高级加密标准（即 AES-256）来加密数据，并且会通过定期轮换的主密钥来对密钥本身进行加密。

> 注意：
>
> - 当使用POST操作上传对象时，需在表单字段中提供相同的信息，而不是提供 x-cos-server-side-encryption 头部。
> - 对于使用预签名 URL 上传的对象，则无法使用 SSE-COS 加密。只能使用 CSP 控制台或 HTTP 请求头部指定服务端加密。

#### 使用对象存储控制台

用户可以参见 上传对象 文档，了解如何通过控制台对对象进行 SSE-COS 加密。

#### 使用 REST API

> 注意：
>
> - 在列出存储桶中对象时，列表会返回所有对象的列表，无论对象是否加密。
> - 当使用 POST 操作上传对象时，请在表单字段中提供相同的信息，而不是提供该请求头部。

当用户请求以下接口时，可以通过提供 `x-cos-server-side-encryption` 头部来应用服务端加密。

- PUT Object
- Initiate Multipart Upload
- PUT Object - Copy
- POST Object

### SSE-KMS 加密

SSE-KMS 加密即使用 KMS 托管密钥的服务端加密。KMS 是亿算云平台推出的一款安全管理类服务，使用经过第三方认证的硬件安全模块 HSM（Hardware Security Module）来生成和保护密钥。它能够帮助用户轻松创建和管理密钥，满足用户多应用多业务的密钥管理需求以及满足监管和合规要求。

首次使用 SSE-KMS 加密，需要 [开通 KMS 服务]，开通 KMS 服务后，系统会自动为您创建一个默认主密钥（CMK）。您也可以通过 [KMS 控制台]自主创建密钥，定义密钥策略及使用方法，KMS 支持用户自主选择密钥材料来源为 **KMS** 或**外部**，更多信息请参见 创建密钥 和 外部密钥导入。

> 注意：

> - SSE-KMS 仅加密对象数据，不会加密任何对象元数据。
> - 使用 SSE-KMS 加密，会产生额外费用，由 KMS 收取。
> - 使用 SSE-KMS 加密的对象必须使用有效签名访问，不可被匿名用户访问。

#### 使用对象存储控制台

用户可以参见 上传对象 文档，了解如何通过控制台对对象进行 SSE-KMS 加密。

#### 使用 REST API

> 注意：
>
> - 在列出存储桶中对象时，列表会返回所有对象的列表，无论对象是否加密。
> - 当使用 POST 操作上传对象时，请在表单字段中提供相同的信息，而不是提供该请求头部。

当用户请求以下接口时，可以通过提供 `x-cos-server-side-encryption` 头部来应用服务端加密。

- PUT Object
- Initiate Multipart Upload
- PUT Object - Copy
- POST Object

#### 注意事项

若您未使用过 **CSP 控制台**进行 SSE-KMS 加密，而只使用 **API** 的方式进行 SSE-KMS 加密时，您需先创建 CAM 角色，具体创建步骤如下：

1. 登录访问管理控制台，进入【角色管理】列表页面。
2. 单击【新建角色】，根据界面提示输入相应参数，单击【下一步：配置角色策略】。
3. 配置角色策略，搜索并勾选【KMSAccessForCOSRole】，然后单击【下一步：审阅】。
4. 最后单击【完成】即可创建完毕。

### SSE-C 加密

SSE-C 加密即用户自定义密钥的服务端加密。加密密钥由用户自己提供，用户在上传对象时，CSP 将使用用户提供的加密密钥对用户的数据进行 AES-256 加密。

> 注意：
>
> - CSP 不存储用户提供的加密密钥，而是存储加密密钥添加了随机数据的 HMAC 值，该值用于验证用户访问对象的请求。CSP 无法使用随机数据的 HMAC 值来推导出加密密钥的值或解密加密对象的内容。因此，如果用户丢失了加密密钥，则无法再次获取到该对象。
> - 当使用 POST 操作上传对象时，需在表单字段中提供相同的信息，而不是提供x-cos-server-side-encryption-* 头部。
> - SSE-C 仅能通过 API 进行使用，不支持控制台操作。

#### 使用 REST API

当用户请求以下接口时，对于 PUT 和 POST 请求可以通过提供 `x-cos-server-side-encryption-*` 头部来应用服务端加密，对于 GET 和 HEAD 请求使用 SSE-C 加密的对象时，需要提供 x-cos-server-side-encryption-* 头部来解密指定对象。以下操作支持此头部：

- GET Object
- HEAD Object
- PUT Object
- Initiate Multipart Upload
- Upload Part
- POST Object
- PUT Object - Copy

# 存储桶加密概述

最近更新时间: 2024-12-19 17:12:00

## 简介

存储桶加密是针对存储桶的一项配置，通过设置存储桶加密，可对新上传至该存储桶的所有对象默认以指定的加密方式进行加密。

目前已支持的加密方式有：

- SSE-COS 加密：即由 CSP 托管密钥的服务端加密。
- SSE-KMS 加密：即由 TCE KMS托管密钥的服务端加密。

有关服务端加密的更多信息，请参见 服务端加密概述。

## 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设置存储桶加密，详情请参见 设置存储桶加密 控制台指南文档。

**使用 REST API**

您可以通过以下 API 配置存储桶加密：

- PUT Bucket encryption
- GET Bucket encryption
- DELETE Bucket encryption

## 注意事项

**在加密存储桶中上传对象**

对于需要配置存储桶加密功能的存储桶，需要注意以下几点：

- 存储桶加密不会对存储桶中已存在的对象产生加密操作。
- 配置了存储桶加密后，对于上传至该存储桶的对象：
  - 如果您的 PUT 请求中不包含加密信息，则上传的对象会以存储桶加密配置来加密。
  - 如果您的 PUT 请求中包含了加密信息，则上传的对象会以 PUT 请求中的加密信息来加密。
- 配置了存储桶加密后，对于投递至该存储桶的清单报告：
  - 若清单本身未配置加密，则投递的清单会以存储桶的加密配置来加密。
  - 若清单本身设置了加密，则投递的清单会以清单的加密配置来加密。

# 对象存储权限组合判断逻辑说明

最近更新时间: 2024-12-19 17:12:00

访问CSP对象存储存储桶及存储桶内资源时需要经过授权后才可进行访问。资源所属的租户的主账号默认对存储桶（Bucket）及存储桶内的资源拥有所有管理权限。其他访问管理（Cloud AccessManagement，CAM）用户账号需要授权后才可进行访问。

账号中的访问控制包括用户策略（User Policy）、存储桶访问控制列表（Access Control List,ACL）和存储桶策略（Bucket Policy）等不同的类型。

## 访问控制列表（ACL）

ACL用于对租户的主账号授权，ACL就是授权记录的列表，每条记录就是“主账号+授权”。以下可支持的权限列表和意义：

| 权限 | 桶（Bucket）权限说明 | 对象（Object）权限说明 |
| --- | --- | --- |
| READ<br>获取对象/列表 | 可读，列出Bucket下所有对象 | 读取Object的数据 |
| WRITE<br>写入数据 | 可写，删除存储桶、覆盖写桶内对象 | 无意义 |
| READ_ACP<br>权限读取 | 读取桶的ACL权限 | 读取Object的ACL权限 |
| WRITE_ACP<br>权限写入 | 修改和写入桶的ACL权限 | 修改和写入Object的ACL权限 |

Object的ACL可分为2类默认（Default）和私有（Private）：

- 默认：ACL中**没有**任何授权记录（上传对象，且不指定ACL时，Object自动获得这样的ACL）。
- 私有：ACL有**一条（含）以上**的授权记录。

## 策略（Policy）

相比于ACL，Policy更加丰富灵活，可以指定一组账号（Principal，常用账号列表指定）对特定一组资源（Resource，常用前缀模糊匹配方式指定）允许或拒绝某些操作（Action）。

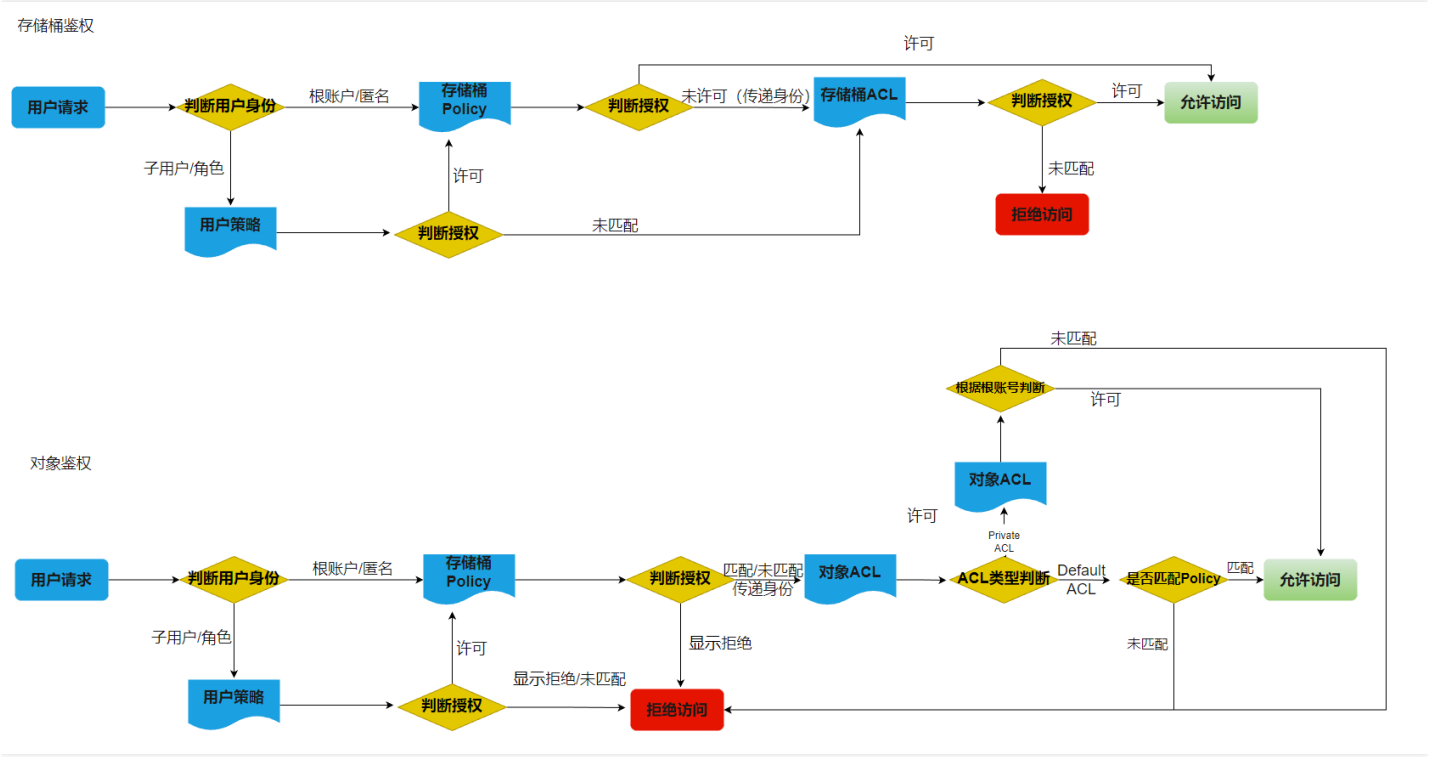
Policy分为两类：

- User Policy：**依附于账号**，一般由租户的主账号用于给**租户子账号授权**，需要从“云产品>访问管理”的租户端配置。
- Bucket Policy：**依附于存储桶**，一般用于授权其他**租户的主账号**访问本存储桶的对象，可从"云产品>对象存储(CSP)"的租户端控制台配置。

> **说明：**
>
> 一个Policy中的多条规则（Statement，"云产品>对象存储(CSP)"中“Policy权限设置”的每一行为一个Statement/规则）的判定方法如下：
>
> 1. 如果匹配到一条拒绝（Deny）规则，就认为Policy拒绝操作。
> 2. 如果匹配到至少一条允许（Allow）规则且没有匹配到任何一条拒绝（Deny）规则，就认为Policy允许操作。
> 3. 如果上述2条都不成立，则认为Policy对该操作无结论。

## 访问权限评估流程

在用户对CSP对象存储发起操作请求时，对存储桶的鉴权和对象的鉴权走不同流程，如下图：




## 跨租户子账号授权典型用法示例

**环境：**

- 租户A有根账号rootA、子账号subA和存储桶bucketA；
- 租户B有根账号rootB和子账号subB。

**授权目标：**使账号subB能够访问bucketA的资源。

**配置方法：**

- 在租户A的"云产品>对象存储(CSP)"租户端控制台使用bucketA的“Policy权限设置”（ Bucket Policy ）中授权rootB进行特定操作；
- 并且在租户B的"云产品>访问管理"租户端控制台中
  - 从“策略管理”，添加策略（ User Policy ），描述允许对bucketA进行哪些操作
  - 从“用户管理>用户>subB>关联策略”，将新添加的策略与subB进行关联

# 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: 2024-12-19 17:12:00

## 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://www.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://www.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**

CSP 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**

以下示例表示允许另一个主账号对某个存储桶有读取权限：




### 为对象设置 ACL

以下示例表示允许另一个主账号对某个对象有读取权限：



# 访问管理实践

最近更新时间: 2024-12-19 17:12:00

## 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 CSP 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 CSP 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、 CSP 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CSP

最近更新时间: 2024-12-19 17:12:00

## 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。

**主账号**

主账号又被称为开发商。用户申请云账号时，系统会创建一个用于登录云服务的主账号身份。主账号是云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，拥有登录云控制台的权限。
- 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 CSP 分为三个步骤：创建子账号、对子账号授予权限、子账号访问 CSP 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】>【用户】**，进入用户管理页面。
3. 单击**【新建用户】，弹出**选择新建用户类型**窗口，选择**【子用户】**，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云平台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。

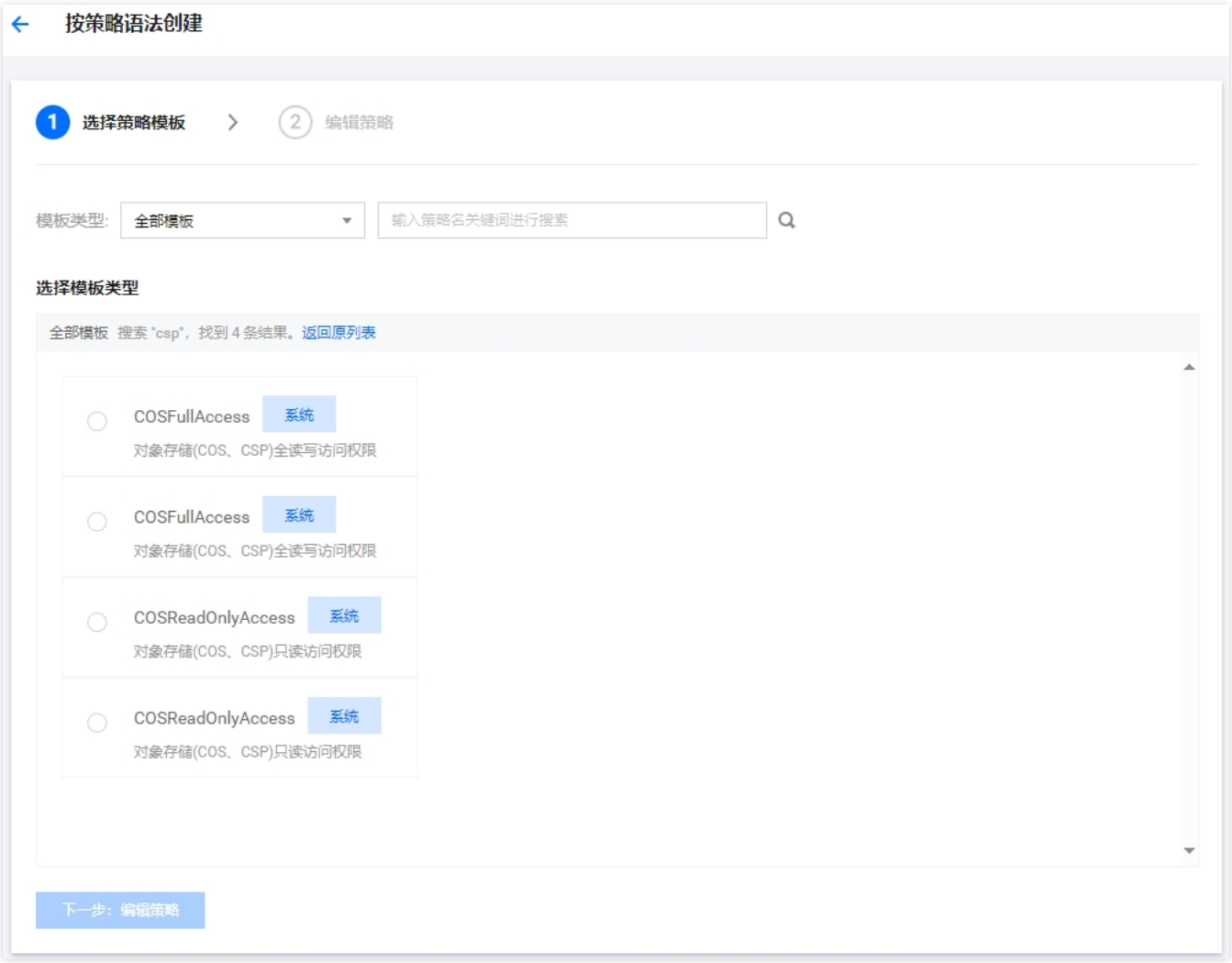
5. 填写用户信息完毕后，单击**【下一步：设定权限】**，进行权限设定。
6. 在设置用户权限页面，根据您的实际需求，选择不同的方式为当前新建的子用户设定权限。有三种方式可选：添加到现有用户组，复制现有用户权限，从策略列表中授权。

   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. 完成权限设定后，单击**【下一步：完成】**即可创建子账号。

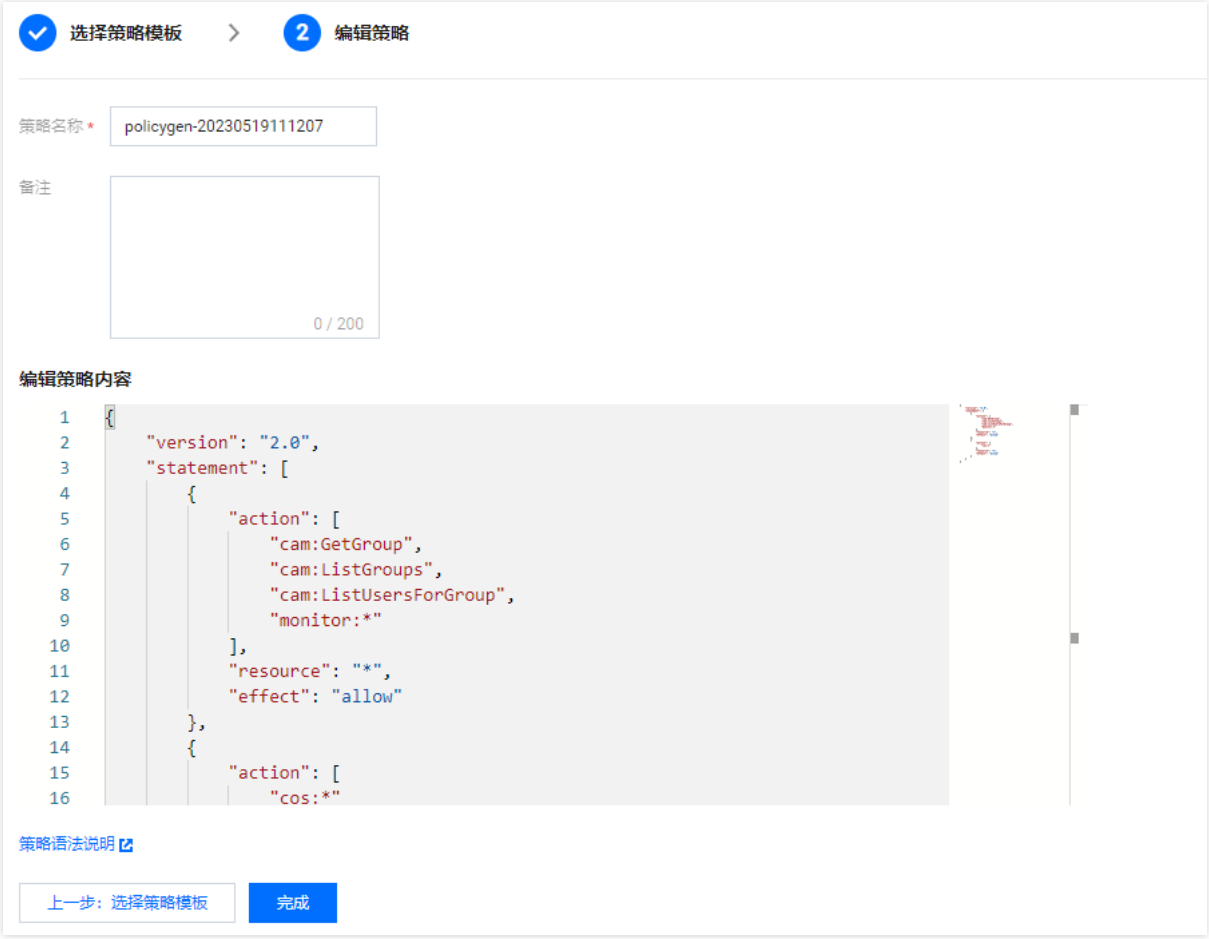
**步骤2：对子账号授予权限**

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. 在 CAM 控制台，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。
2. 可供选择的模版有**空白模板**和与 CSP 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:编辑策略】**。




3. 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【完成】。




4. 创建完成后，将刚才已创建的策略关联到子账号。

5. 勾选子账号确定授权后，子账号即可根据权限范围访问 CSP 资源。

### 步骤3：子账号访问 CSP 资源

CSP 访问（API 或 SDK）需要如下资源：APPID、SecretId、SecretKey。 当使用子账号访问 CSP 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**【用户管理】>【用户】**，进入用户管理页面。

3. 单击子账号用户名称，进入子账号信息详情页。

4. 在左侧导航树中，选择**【云API 密钥】，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 CSP 资源。

> 注意：
>
> - 子账号需通过 XML API 或基于 XML API 的 SDK 访问 CSP 资源。
> - 子账号访问 CSP 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 CSP 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。 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"]
}
}
}
]
}
```

# 临时密钥生成及使用指引

最近更新时间: 2024-12-19 17:12:00

## 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。<br>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。<br>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://sts.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=eyJzdGF0ZW1lbnQiOiBbeyJhY3Rpb24iOiBbIm5hbWUvY29zOkdldE9iamVjdCIsIm5hbWUvY29zOlB1dE9iamVjdCJdLCJlZmZlY3QiOiAiYWxsb3ciLCJyZXNvdXJjZSI6WyJxY3M6OmNvczpjbi1ub3J0aDp1aWQvMTI1MjQ0ODcwMzpwcmVmaXgvLzEyNTI0NDg3MDMvcmFiYml0bGI1dGovKiJdfV0sInZlcnNpb24iOiAiMi4wIn0=
```

**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 CSP

CSP 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 CSP，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 csp 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();
}
}
```

# 用户工具
# 环境安装与配置
# Java 安装与配置

最近更新时间: 2024-12-19 17:12:00

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 D:\Program Files\Java\jre1.8.0_31

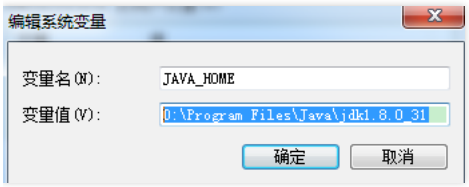



### 3. 配置

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*



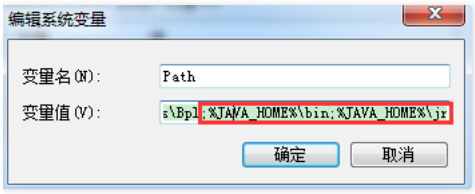

变量名(N)：**CLASSPATH**

变量值(V)：.;%JAVA_HOME%\lib\dt.jar;%JAVA_HOME%\lib\tools.jar; *//注意变量值开头有"."*



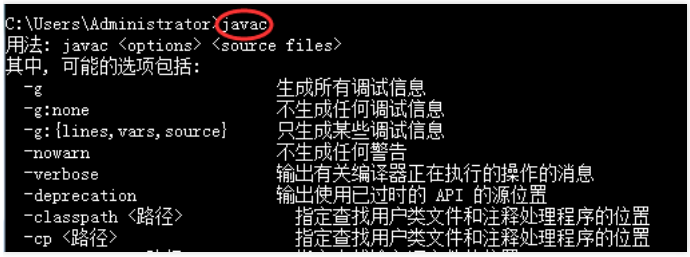

变量名(N)：**Path**

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: 2024-12-19 17:12:00

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。

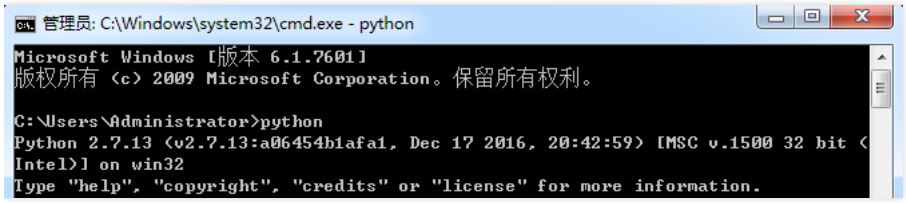
### 3. 环境变量配置

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://www.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
```

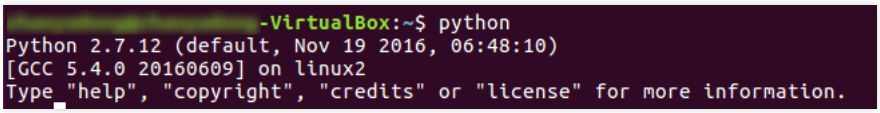
#### 5. 将系统 Python 命令指向 Python 2.7

```
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: 2024-12-19 17:12:00

Hadoop 工具依赖 Hadoop-2.7.2 及以上版本，实现了以云平台 CSP 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：CentOS-7-x86_64-DVD-1611.iso。
3. 安装 Java 环境：jdk-8u144-linux-x64.tar.gz，具体操作请参见 Java 安装与配置。
4. 安装 Hadoop-2.7.4 包：hadoop-2.7.4.tar.gz。

### 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-8u144-linux-x64.tar.gz 安装包到 `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-2.7.4.tar.gz 安装包到 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/
```

```
[root@VM_96_24_centos /usr/hadoop-2.7.4]# hadoop jar share/hadoop/mapreduce/hadoop-mapreduce-examples-2.7.4.jar wordcount /input /output/
17/07/18 23:04:51 INFO client.RMProxy: Connecting to ResourceManager at master/10.104.96.24:8032
17/07/18 23:04:53 INFO input.FileInputFormat: Total input paths to process : 1
17/07/18 23:04:53 INFO mapreduce.JobSubmitter: number of splits:1
17/07/18 23:04:54 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1500344813707_0002
17/07/18 23:04:54 INFO impl.YarnClientImpl: Submitted application application_1500344813707_0002
17/07/18 23:04:54 INFO mapreduce.Job: The url to track the job: http://master:8088/proxy/application_1500344813707_0002/
17/07/18 23:04:54 INFO mapreduce.Job: Running job: job_1500344813707_0002
17/07/18 23:05:01 INFO mapreduce.Job: Job job_1500344813707_0002 running in uber mode : false
17/07/18 23:05:01 INFO mapreduce.Job:  map 0% reduce 0%
17/07/18 23:05:05 INFO mapreduce.Job:  map 100% reduce 0%
17/07/18 23:05:11 INFO mapreduce.Job:  map 100% reduce 100%
17/07/18 23:05:12 INFO mapreduce.Job: Job job_1500344813707_0002 completed successfully
17/07/18 23:05:12 INFO mapreduce.Job: Counters: 49
```

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
```

```
[root@VM_96_24_centos /usr/hadoop-2.7.4]# hadoop fs -cat /output/part-r-00000
a       5
dasdada 1
ds      2
qwe     1
ret     1
s       1
v       2
vdfd    1
wqere   1
```

> **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: 2024-12-19 17:12:00

本文介绍Windows环境下COSBrowser工具登录时的配置说明。

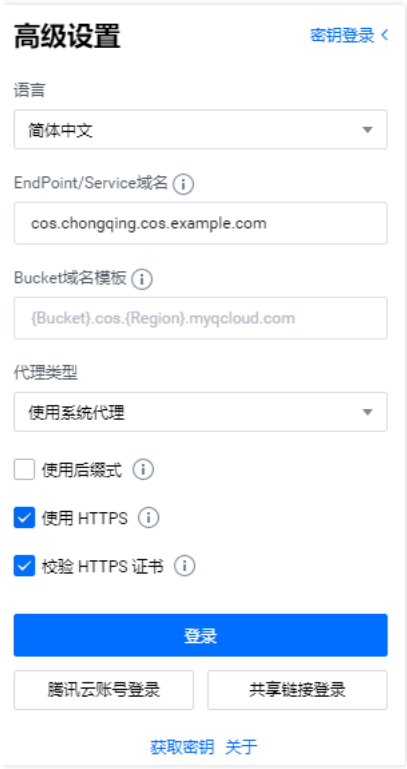
## 密钥登录

**基本配置**




| 配置参数 | 配置说明 |
| --- | --- |
| SecretID | 1. 登录访问管理。<br>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。<br>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。<br>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【域名信息】。<br>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-1255000220.cos.chongqing.csp.yfm18.tcepoc.fsphere.cn，那么 Endpoint 就是 cos.chongqing.csp.yfm18.tcepoc.fsphere.cn<br><br>![](http://imgcache-nginx-fileserver.tcs-system.svc/doc_pdf/static/imgs/f_2023-5-19_f_1728ab70-f619-11ed-a06c-9751a7edb870.png) |
| 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。<br>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: 2024-12-19 17:12:00

## COSBrowser简介

COSBrowser 是对象存储CSP 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 CSP 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<br>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，软件登录界面如下所示。 具体配置参数说明参见COSBrowser工具配置。

> **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**<br>2. 正确填写存储桶名称，并选择好地域、访问权限<br>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**<br>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**

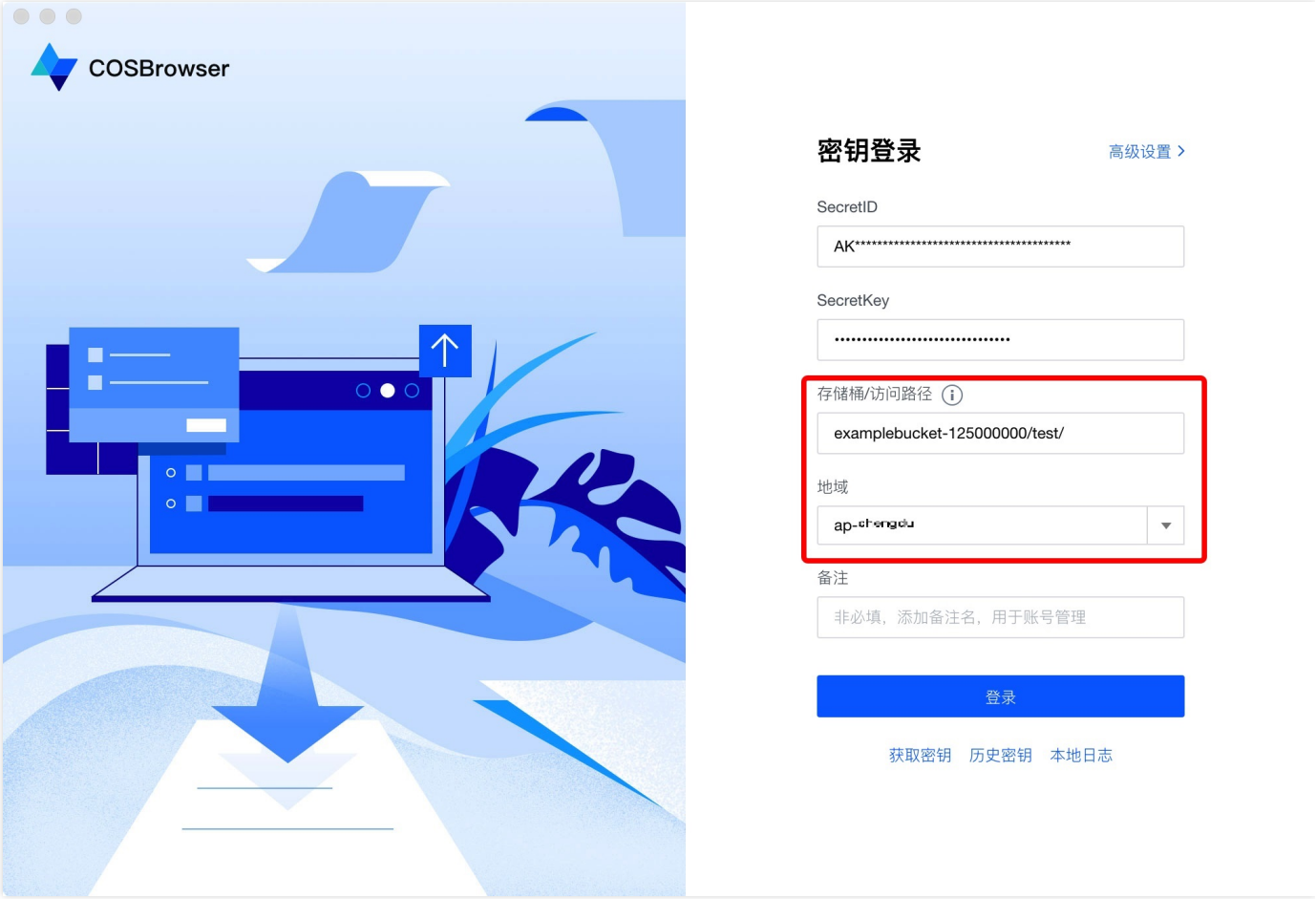
您可通过单击存储桶列表右侧的**详情**，查看存储桶详情。存储桶详细信息包含存储桶名称、访问权限、版本控制状态。

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。

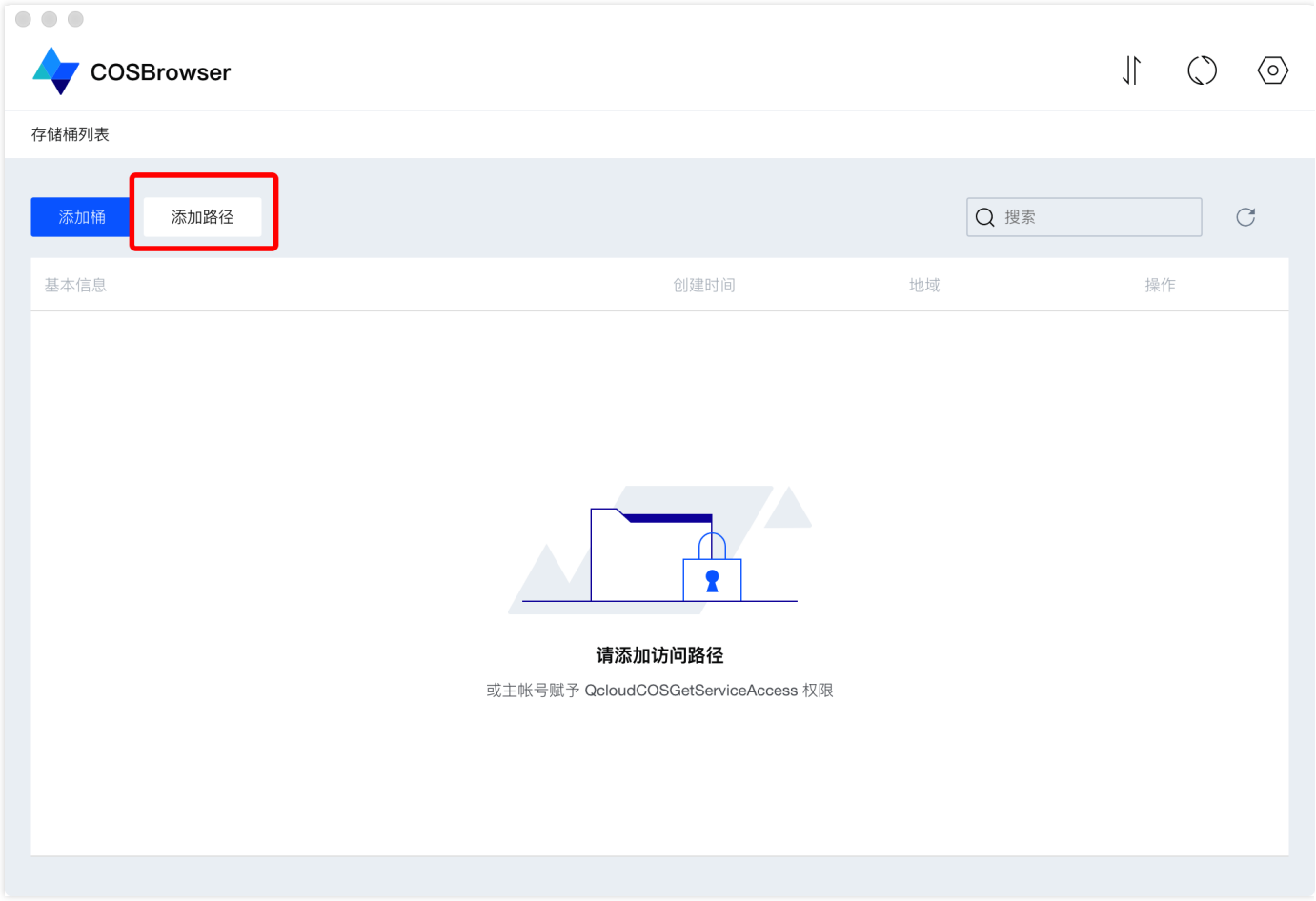
**3. 添加访问路径**

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：<br>1. 单击**上传**，选择文件后，直接上传文件。<br>2. 在文件列表空白处右键单击**上传文件**，进行上传。<br>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：<br>1. 单击**上传**，选择文件夹，直接上传文件夹。<br>2. 在文件列表空白处右键选择**上传文件夹**，进行上传。<br>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：<br>1. 使用上传文件夹的方式，单击下一步。<br>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：<br>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。<br>2. 选中文件，右键单击**下载**。<br>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：<br>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。<br>2. 右键单击**下载**，直接下载该文件夹。<br>3. 通过鼠标将文件夹拖拽至本地的方式进行下载。 |
| 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：<br>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。<br>2. 单击下拉框内的**高级下载**，在弹窗中选择**跳过**。<br>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。 |

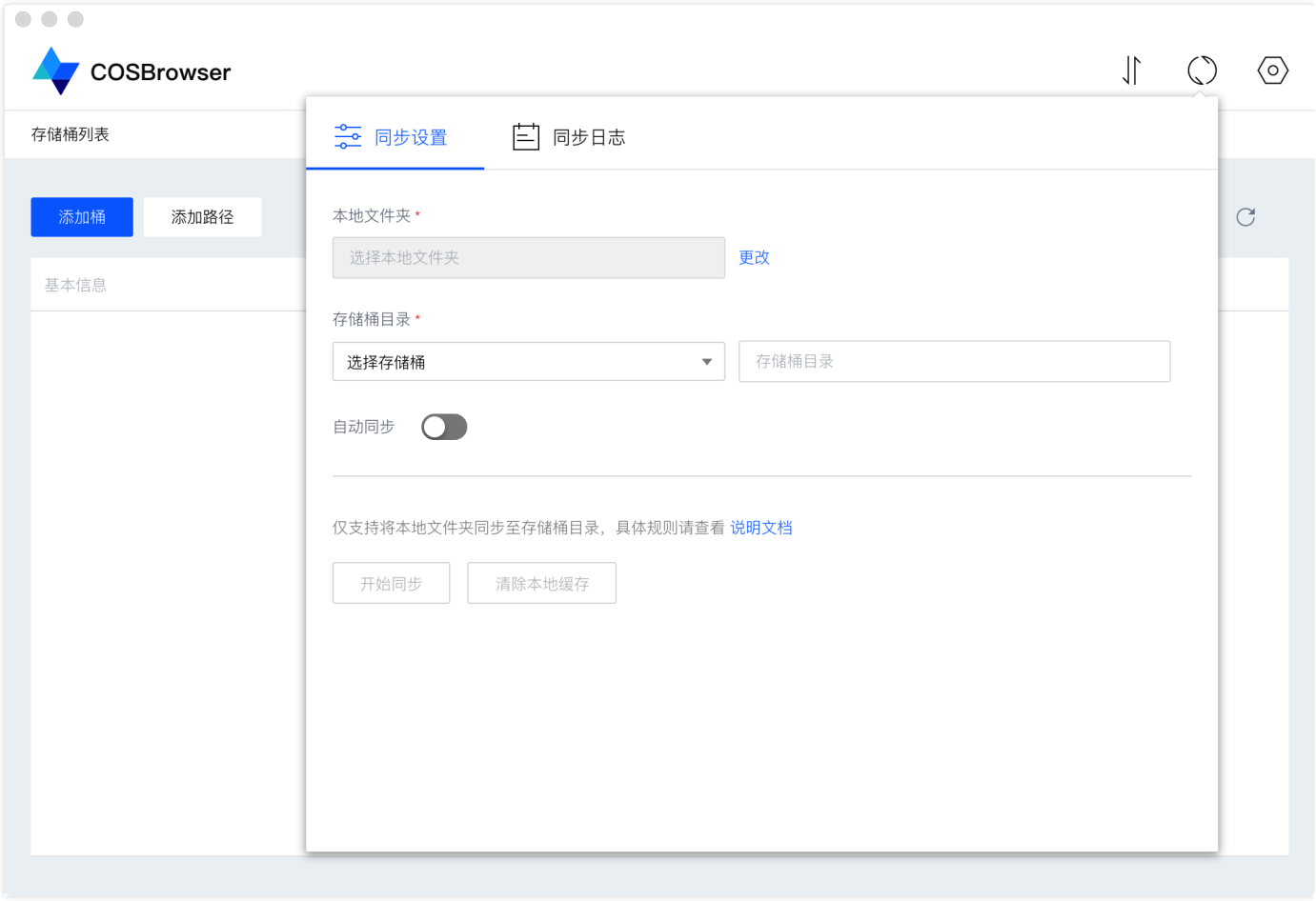
**6. 删除文件/文件夹**

选中想要删除的文件/文件夹，可通过单击界面上方**更多 > 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 > 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 > 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 CSP 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**

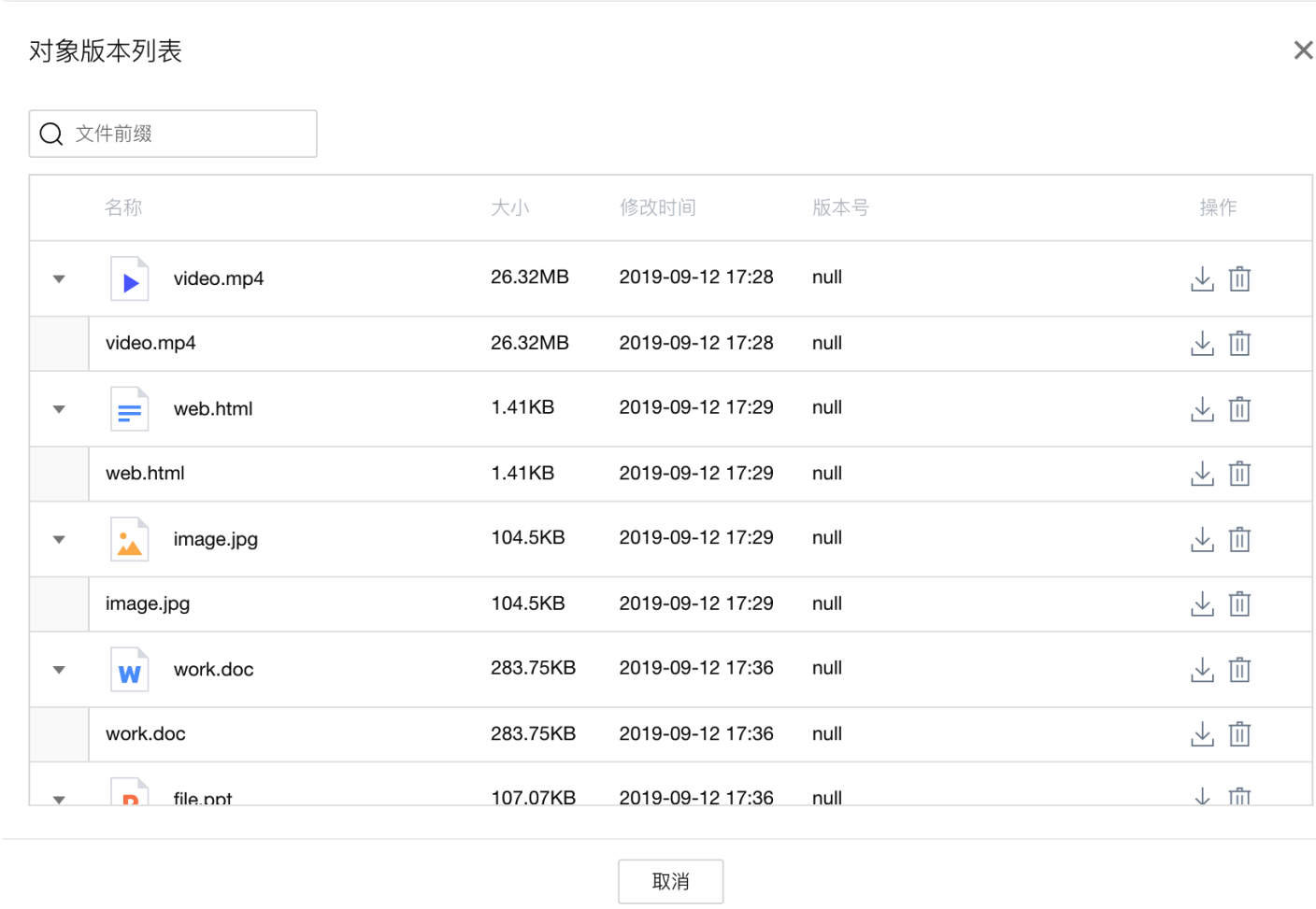
可通过存储桶内右上方的搜索框，输入文件名进行搜索。COSBrowser 支持文件名前缀搜索和文件模糊搜索。

**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**。<br>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**。<br>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**。<br>2. 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**。<br>2. 设定传输失败的重试次数，默认为5。 |
| 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**。<br>2. 勾选二次校验。 |

版权所有：亿算云平台

| 系统功能 | 说明 | 如何操作 |
| --- | --- | --- |
| 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**。<br>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**。<br>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: 2024-12-19 17:12:00

## 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 Python 安装与配置。
- 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

**pip 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 `-v` 或者 `--version` 命令查看当前的版本信息。

**pip 更新**

执行 `pip` 命令进行更新：

```
pip install coscmd -U
```

> 注意：当 pip 版本号大于等于10.0.0 时，升级或安装依赖库时可能会出现失败，建议使用 pip 版本 9.x（pip install pip==9.0.0）。

**源码安装（不推荐）**

源码下载地址：单击此处。

```
git clone https://github.com/tencentyun/coscmd.git
cd coscmd
python setup.py install
```

> 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**

> 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
```

## 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 CSP
download Download file from CSP to local
delete Delete file or files on CSP
abort Aborts upload parts on CSP
copy Copy file from CSP to CSP
list List files on CSP
listparts List upload parts
info Get the information of file on CSP
mget Download file from CSP 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>
[-s] <SECRET_KEY>
[-t] <TOKEN>
[-b] <BucketName-APPID>
[-r] <REGION> | [-e] <ENDPOINT>
[-m] <MAX_THREAD>
[-p] <PART_SIZE>
[--do-not-use-ssl]
[--anonymous]
```

通常情况下，如您只需要进行简单的配置，可参照如下操作示例配置即可：

```
coscmd config -a AChT4ThiXAbpBDEFGhT4ThiXAbp**** -s WE54wreefvds3462refgwewe**** -b examplebucket-1250000000 -r ap-beijing
```

> 说明：
>
> 其中 "[]" 中的字段为选项，"<>" 的字段为需要填写的参数。

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 请前往 API 密钥控制台 获取 | 是 | 字符串 |
| -s | 密钥 Key 请前往 API 密钥控制台 获取 | 是 | 字符串 |
| -t | 临时密钥 token，当使用临时密钥时需要配置，设置 x-cos-security-token 头部 | 否 | 字符串 |
| -b | 指定的存储桶名称，存储桶的命名格式为 BucketName-APPID | 是 | 字符串 |
| -r | 存储桶所在地域 | 是 | 字符串 |
| -e | 设置请求的 ENDPOINT，设置 ENDPOINT 参数后，REGION 参数会失效 | 否 | 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 |
| -p | 分块操作的单块大小（单位MB，默认为1MB，范围为1 - 1000） | 否 | 数字 |
| --do-not-use-ssl | 使用 HTTP 协议，而不使用 HTTPS | 否 | 字符串 |
| --anonymous | 匿名操作（不携带签名） | 否 | 字符串 |

> 说明：
>
> 1. 可以直接编辑 ~/.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，是通过 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 = AChT4ThiXAbpBDEFGhT4ThiXAbp****
>    secret_key = WE54wreefvds3462refgwewe****
>    bucket = examplebucket-1250000000
>    region = ap-beijing
>    max_thread = 5
>    part_size = 1
>    schema = https
>    ```

> 2. 可以在配置文件中增加 schema 项来选择 http/https ，默认为 https 。
>
> 3. 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 Region 的命令

- 通过 -b <BucketName-APPID> 指定 Bucket， 可以指定特定的 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。
- 通过 -r <region> 指定 Region， 可以指定特定的 Region。

```
#命令格式
coscmd -b <BucketName-APPID> -r <region> <action> ...
#操作示例-创建bucket
coscmd -b examplebucket-1250000000 -r ap-beijing createbucket
#操作示例-上传文件
coscmd -b examplebucket-1250000000 -r ap-beijing upload exampleobject exampleobject
```

### 创建存储桶

建议配合 -b <BucketName-APPID> 指定 Bucket 和 -r <region> 指定 Region 使用。

```
#命令格式
coscmd -b <BucketName-APPID> createbucket
#操作示例
coscmd createbucket
coscmd -b examplebucket-1250000000 -r ap-beijing createbucket
```

### 删除存储桶

- 建议配合 -b <BucketName-APPID> 指定 Bucket 和 -r <region> 指定 Region 使用。

  ```
  #命令格式
  coscmd -b <BucketName-APPID> deletebucket
  #操作示例
  coscmd deletebucket
  coscmd -b examplebucket-1250000000 -r ap-beijing deletebucket
  coscmd -b examplebucket-1250000000 -r ap-beijing 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 CSP 上存储的路径（cospath）。

> 注意：
>
> - 上传文件时需要将 CSP 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（CSP 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>` 。更多头部可参见 PUT Object 文档。
> - 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 CSP 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 CSP 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/exampleobjec
t
coscmd -b examplebucket1-1250000000 -r ap-guangzhou copy examplebucket2-1250000000.ap-beijing.myqcloud.com/data/exampleobject data/e
xamplefolder/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/e
xamplefolder/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/e
xamplefolder/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/ e
xamplefolder
```

```
coscmd -b examplebucket1-1250000000 -r ap-guangzhou copy -r examplebucket2-1250000000.cos.ap-guangzhou.myqcloud.com/examplefolder/ e
xamplefolder/
```

请将"<>"中的参数替换为您需要复制的 CSP 上文件的路径（sourcepath），和您需要复制到 CSP 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>`。更多头部请参见 PUT Object - Copy 文档。

### 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 CSP 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 CSP 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 CSP 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。

#### 恢复归档文件

- 恢复归档文件命令如下：

```
#命令格式
coscmd restore <cospath>
#操作示例
coscmd restore -d 3 -t Expedited exampleobject
```

- 批量恢复归档文件命令如下：

```
#命令格式
coscmd restore -r <cospath>
#操作示例
coscmd restore -r -d 3 -t Expedited examplefolder/
```

请将 "<>" 中的参数替换为您需要查询文件列表的 CSP 上文件的路径（cospath）。

- 使用 `-d day` 设置临时副本的过期时间，默认值：7。
- 使用 `-t tier` 具体复原过程类型，枚举值： Expedited （极速模式），Standard （标准模式），Bulk（批量模式），默认值：Standard。

#### 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

如您在使用 COSCMD 工具过程中，有相关的疑问，请参见 COSCMD 工具类常见问题。

# COS Migration工具

最近更新时间: 2024-12-19 17:12:00

## 功能说明

COS Migration 是一个集成了 CSP 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 CSP 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 CSP。
  - 其他云存储：目前支持 AWS S3，阿里云 OSS，七牛存储迁移至 CSP，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 CSP。
  - Bucket 相互复制：CSP 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 CSP。
- 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 CSP，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 CSP |
| migrateAws | 从 AWS S3 迁移至 CSP |
| migrateAli | 从阿里 OSS 迁移至 CSP |
| migrateQiniu | 从七牛迁移至 CSP |
| migrateUrl | 下载 URL 迁移到 CSP |
| migrateBucketCopy | 从源 Bucket 复制到目标 Bucket |
| migrateUpyun | 从又拍云迁移到 CSP |

#### 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 CSP 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 CSP 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 CSP 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 CSP 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 CSP，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 CSP，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 CSP 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 CSP，则进行该部分配置，具体配置项及说明如下：

```
# 从本地迁移到 CSP 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 CSP，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 CSP，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 CSP 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 CSP，则进行该部分配置，具体配置项及说明如下：

```
# 从 AWS 迁移到 CSP 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 CSP，则进行该部分配置，具体配置项及说明如下：

```
# 从七牛迁移到CSP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 CSP，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 CSP 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://aaa.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 CSP 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：

> 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 CSP 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 CSP，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，CSP 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 CSP 路径，则以 **-Dcommon.cosPath=/bbb/ddd** 表示。

## 迁移机制及流程

#### 迁移机制原理

CSP 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 CSP，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：

```
$ sh start_migrate.sh
[skip] task_info: [taskType: migrateAws] [bucket: chengwus3sdkgz-1251668577] [cosPath: /aws0403_17/aaa/bbbslash.txt]
[DownloadOk] [key: slash.txt] [byteDownload/ byteTotal/ percentage: 6/ 6/ 100.00%]
[DownloadInProgress] [key: aws_200M.txt] [byteDownload/ byteTotal/ percentage: 10240/ 209715200/ 0.00%]
[UploadInProgress] [key: /aws0403_17/slash.txt] [byteSent/ byteTotal/ percentage: 6/ 6/ 100.00%]
[ok] task_info: [taskType: migrateAws] [bucket: chengwus3sdkgz-1251668577] [cosPath: /aws0403_17/slash.txt]
[DownloadInProgress] [key: aws_200M.txt.copy] [byteDownload/ byteTotal/ percentage: 10240/ 209715200/ 0.00%]
[DownloadInProgress] [key: aws_200M.txt] [byteDownload/ byteTotal/ percentage: 236126/ 209715200/ 0.11%]
[DownloadInProgress] [key: aws_200M.txt] [byteDownload/ byteTotal/ percentage: 392798/ 209715200/ 0.19%]
[DownloadInProgress] [key: aws_200M.txt] [byteDownload/ byteTotal/ percentage: 514654/ 209715200/ 0.25%]
[DownloadInProgress] [key: aws_200M.txt.copy] [byteDownload/ byteTotal/ percentage: 96863/ 209715200/ 0.05%]
[DownloadInProgress] [key: aws_200M.txt] [byteDownload/ byteTotal/ percentage: 688734/ 209715200/ 0.33%]
[DownloadInProgress] [key: aws_200M.txt] [byteDownload/ byteTotal/ percentage: 827998/ 209715200/ 0.39%]
[DownloadInProgress] [key: aws_200M.txt] [byteDownload/ byteTotal/ percentage: 949854/ 209715200/ 0.45%]
[DownloadInProgress] [key: aws_200M.txt] [byteDownload/ byteTotal/ percentage: 1089118/ 209715200/ 0.52%]
[DownloadInProgress] [key: aws_200M.txt] [byteDownload/ byteTotal/ percentage: 1228382/ 209715200/ 0.59%]
[DownloadInProgress] [key: aws_200M.txt] [byteDownload/ byteTotal/ percentage: 1315422/ 209715200/ 0.63%]
[DownloadInProgress] [key: aws_200M.txt] [byteDownload/ byteTotal/ percentage: 1385054/ 209715200/ 0.66%]
[DownloadInProgress] [key: aws_200M.txt.copy] [byteDownload/ byteTotal/ percentage: 166495/ 209715200/ 0.08%]
[DownloadInProgress] [key: aws_200M.txt] [byteDownload/ byteTotal/ percentage: 1506910/ 209715200/ 0.72%]
```

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：

```
migrateAli over! op statistics:
                    op_status : ALL_OK
                   migrate_ok : 530038
                 migrate_fail : 0
                 migrate_skip : 496264
                   start_time : 2018-03-19 15:52:02
                     end_time : 2018-03-19 16:54:38
                    used_time : 3756 s
```

## 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: 2024-12-19 17:12:00

## 功能说明

Hadoop-COS 基于对象存储实现了标准的 Hadoop 文件系统，可以为 Hadoop、Spark 以及 Tez 等大数据计算框架集成 CSP 提供支持，使其能够跟访问 HDFS 文件系统时相同，读写存储在 CSP 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 |
| 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：<br>1.org.apache.hadoop.fs.auth.SessionCredentialProvider：从请求 URI 中获取 secret id 和 secret key。<br>其格式为： `cosn://{secretId}:{secretKey}@examplebucket-1250000000/` 。<br>2.org.apache.hadoop.fs.auth.SimpleCredentialProvider：<br>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。<br>3.org.apache.hadoop.fs.auth.EnvironmentVariableCredentialProvider：<br>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：<br>1.org.apache.hadoop.fs.auth.SessionCredenti<br>2.org.apache.hadoop.fs.auth.SimpleCredentia<br>3.org.apache.hadoop.fs.auth.EnvironmentVari |

| 属性键 | 说明 | 默认值 |
| --- | --- | --- |
| 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 CSP 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 CSP 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 CSP 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 CSP 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 CSP 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>xxxxxxxxxxxxxxxxxxxxxxxxx</value>
<description>Tencent Cloud Secret Id</description>
</property>

<property>
<name>fs.cosn.userinfo.secretKey</name>
<value>xxxxxxxxxxxxxxxxxxxxxxxx</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 CSP 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
CSP, 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 CSP 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: 2024-12-19 17:12:00

## 功能说明

HDFS TO COS 工具用于将 HDFS 上的数据拷贝到对象存储 CSP 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。

> 当命令行参数中的参数与配置文件重合时，以命令行为准。

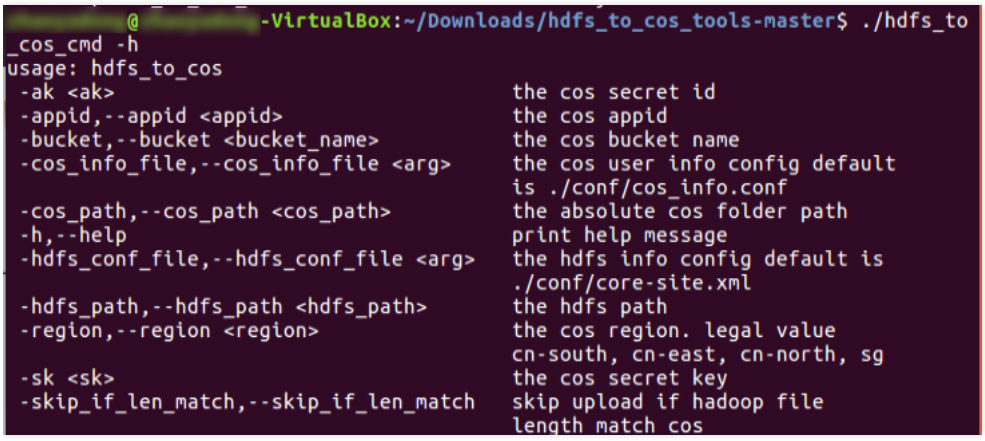
**使用方法**

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 CSP，若 CSP 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 CSP，若 CSP 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 CSP，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 CSP 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: 2024-12-19 17:12:00

## SDK 概览

除了直接使用 API 接口外，CSP 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 |

版权所有：亿算云平台

# Android SDK
# 快速入门

最近更新时间: 2024-12-19 17:12:00

## 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 CSP 控制台 获取 APPID、SecretId、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：CSP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

- 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 CSP 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"; // csp 的 bucket格式为：xxx-appid, 如 test-1253960454
String cosPath = "远端路径，即存储到 CSP 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

/*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"; // csp 的 bucket格式为：xxx-appid, 如 test-1253960454
String cosPath = "远端路径，即存储到 CSP 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 CSP 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"; // csp 的 bucket格式为：xxx-appid, 如 test-1253960454
String cosPath = "远端路径，即存储到 CSP 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，请参考临时密钥生成及使用指引。

**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: 2024-12-19 17:12:00

## SDK 异常信息

SDK中，若是调用接口操作csp 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，csp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csp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：CSP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，请参考临时密钥生成及使用指引。

**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(csp 的 bucket格式为：xxx-appid, 如 test-1253960454) | String | 是 |
| cosPath | 远端路径，即存储到 CSP 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(csp 的 bucket格式为：xxx-appid, 如 test-1253960454) | String | 是 |
| cosPath | 远端路径，即存储到 CSP 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 | String | 是 |
| cosPath | 远端路径，即存储到 CSP 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 | String | 是 |
| cosPath | 远端路径，即存储到 CSP 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 | String | 是 |
| cosPath | 远端路径，即存储到 CSP 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 | String | 是 |
| cosPath | 远端路径，即存储到 CSP 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 | String | 是 |
| cosPath | 远端路径，即存储到 CSP 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(csp 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 | String | 是 |
| cosPath | 远端路径，即存储到 CSP 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 | String | 是 |
| cosPath | 目标路径，即存储到 CSP 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 | 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 | 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**

```
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 | 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(csp 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 | 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(csp 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(csp 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://www.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://gsesgpucloud.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(csp 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(csp 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(csp 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);
}
});

*/
```

## 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(csp 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: 2024-12-19 17:12:00

## 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 CSP 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：CSP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。

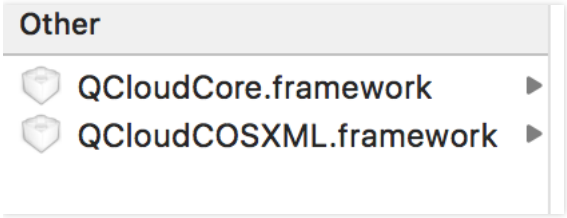
##### 使用Cocoapods导入(推荐)

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：



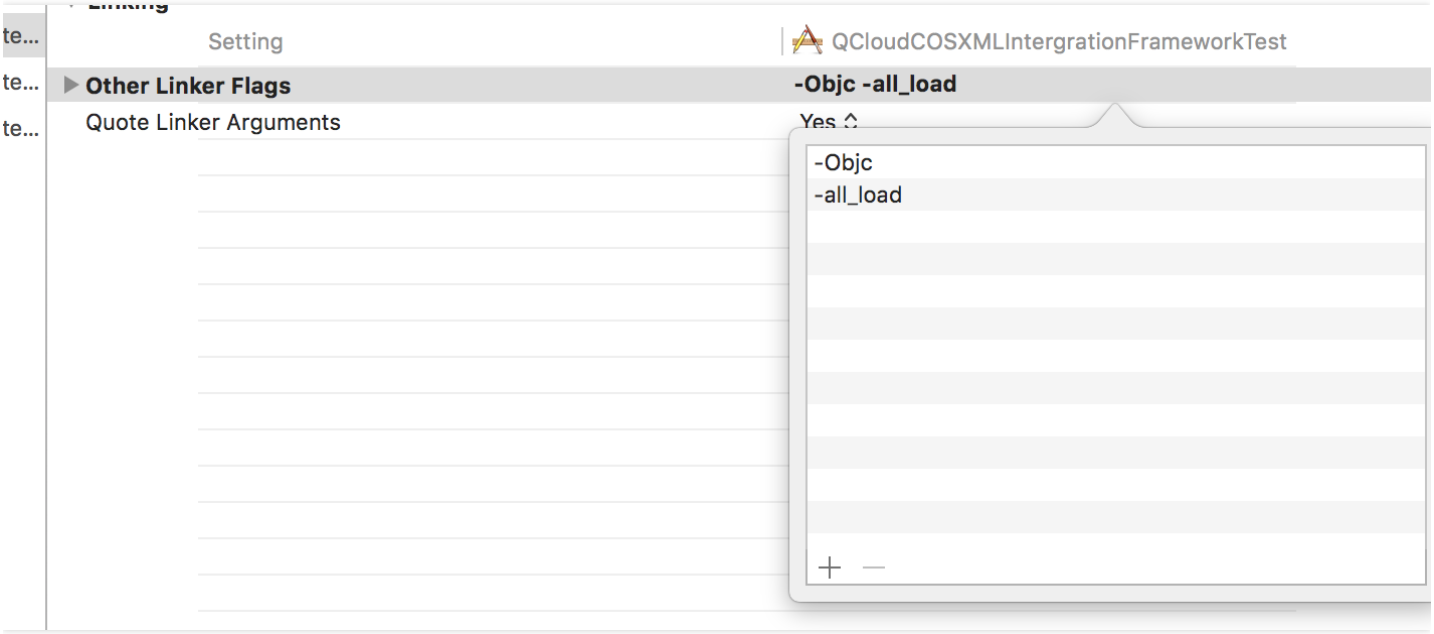

并添加以下依赖库：

- CoreTelephony
- Foundation
- SystemConfiguration
- 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 CSP 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控制台上申请 CSP 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 CSP 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 |

| 参数名称 | 说明 | 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 |

**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

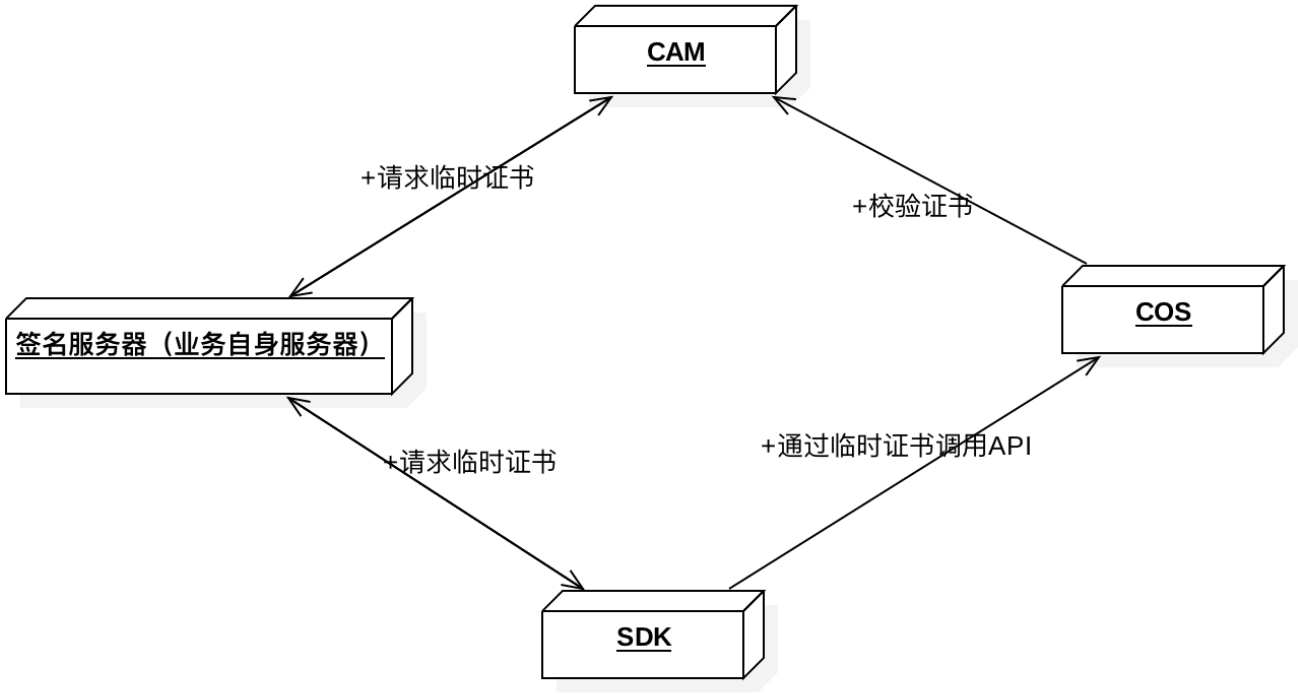
SDK 中的请求需要用到签名，以确认访问的用户的身份，也保障了访问的安全性。当签名不正确时，大部分 CSP 的服务将无法访问并且返回 403 错误。在 SDK 中可以生成签名，每个请求会向 QCloudServiceConfiguration 对象中的signatureProvider 对象来请求生成签名。我们可以将负责生成签名的对象在一开始赋值给 signatureProvider，该生成签名的对象需要遵循 QCloudSignatureProvider 协议，并实现生成签名的方法：

```
- (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。




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QCloudCredential对象生成QCloudAuthenticationCreator，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 QCloudCredentialFenceQueue 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 QCloudCredentialFenceQueue，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 QCloudCredentialFenceQueue 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 QCloudCredentialFenceQueue 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,expirationDate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: 2024-12-19 17:12:00

> **说明：**
>
> 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CSP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 |

| 参数名称 | 描述 | 类型 | 必填 |
| --- | --- | --- | --- |
| 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CSP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CSP控制台上面看到，格式为- ,例如 testBucket-1253653367 | NSString * | 是 |

| 参数名称 | 描述 | 类型 | 必填 |
| --- | --- | --- | --- |
| 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=" "；<br>当需要给子账户授权时，id="qcs::cam::uin/:uin/"，<br>当需要给根账户授权时，id="qcs::cam::uin/:uin/"<br>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**

```
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CSP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://www.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CSP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://www.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CSP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CSP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CSP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CSP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CSP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];
```

## 文件操作

在 CSP 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 |
| --- | --- | --- | --- |
| 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CSP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CSP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CSP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=" "；<br>当需要给子账户授权时，id="qcs::cam::uin/:uin/"，<br>当需要给根账户授权时，id="qcs::cam::uin/:uin/"<br>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 |
| --- | --- | --- | --- |
| bucket | 存储桶名,可在CSP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CSP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://www.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 | 描述 | 类型 | 必填 |
| --- | --- | --- | --- |
| object | 对象名 | NSString * | 是 |
| bucket | 存储桶名,可在CSP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CSP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 |
| grantRead | 赋予被授权者读的权限。格式： id=" ",id=" " ；<br>当需要给子账户授权时，id="qcs::cam::uin/:uin/"，<br>当需要给根账户授权时，id="qcs::cam::uin/:uin/"<br>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CSP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];
```

## 服务器端加密(Server side encryption)说明

CSP支持服务器端加密(Server side encryption)，该特性的作用是object在上传到CSP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: 2024-12-19 17:12:00

## 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 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 CSP 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 | CSP 中用于存储数据的容器 |
| Object | CSP 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。<br>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，CSP 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 CSP 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 CSP 服务域名填写
cos_str_set(&options->config->access_key_id, "<用户的SecretId>"); //用户注册 CSP 服务后所获得的 SecretId
cos_str_set(&options->config->access_key_secret, "<用户的SecretKey>"); //用户注册 CSP 服务后所获得的 SecretKey
```

```
cos_str_set(&options->config->appid, "<用户的AppId>"); //用户注册 CSP 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: 2024-12-19 17:12:00

# 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 | 获取存储桶 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 | CSP 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。<br>有效值：COS_ACL_PRIVATE(0)，COS_ACL_PUBLIC_READ(1)，COS_ACL_PUBLIC_READ_WRITE(2)<br>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 | CSP 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 | CSP 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。<br>有效值：COS_ACL_PRIVATE(0)，COS_ACL_PUBLIC_READ(1)，COS_ACL_PUBLIC_READ_WRITE(2)<br>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 | CSP 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） | 获取对象列表 | 查询存储桶下的部分或者全部对象 |
| PUT Object | 上传对象 | 上传一个对象至存储桶 |
| HEAD Object | 获取对象元数据 | 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 |
| 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 | 获取对象 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 | CSP 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 |
| 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 | CSP 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 |
| filename | Object 本地保存文件名称 | String |
| headers | CSP 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 | CSP 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 |
| headers | CSP 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 | CSP 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 | CSP 请求附加头域 | Struct |
| params | CSP 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 | CSP 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 | CSP 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 | CSP 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，CSP 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 | CSP 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<br>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 |

| 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_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 | CSP 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 | 定界符为一个符号。<br>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。<br>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。<br>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。<br>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。<br>当 key-marker 未被指定时，upload-id-marker 将被忽略。<br>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 |

| 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;
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 | CSP 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 | CSP 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 |

| 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_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 | CSP 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 |
| --- | --- | --- |
| 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 | CSP 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 |
| 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 | CSP 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 | CSP 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_list | 完成分块上传的参数 | Struct |
| part_number | 分块编号 | String |

| 参数名称 | 参数描述 | 类型 |
| --- | --- | --- |
| 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 | CSP 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 | CSP 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 | CSP 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 |
| cos_acl | 允许用户自定义权限。有效值：COS_ACL_PRIVATE(0)，COS_ACL_PUBLIC_READ(1)<br>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 |

| 参数名称 | 参数描述 | 类型 |
| --- | --- | --- |
| 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 | CSP 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 | 获取跨域配置 | 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 |

## 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 | CSP 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);

//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://www.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://www.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://www.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 | CSP 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);

//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 | CSP 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);
```

# 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 | CSP 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 |
| 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 |
| 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: 2024-12-19 17:12:00

## 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 。
- 示例 Demo 下载地址： XML C++ SDK 示例。

### 环境依赖

- 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 https://cmake.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://sourceforge.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 https://www.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 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
```

> **说明：**
>
> 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 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见 CSP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",

// CSP 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-1250000000.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: 2024-12-19 17:12:00

## 生成签名

### Sign 功能说明

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 |
| --- | --- | --- |
| 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，可以在指定的有效期内（通过 CosSysConfig 设置, 默认 60 s）使用，返回空串表示签名失败。

#### 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 |

| 参数名称 | 参数描述 | 类型 |
| --- | --- | --- |
| end_time_in_s | 签名生效的截止时间 | uint64_t |

**返回结果说明**

- String， 返回签名，可以在指定的有效期内使用, 返回空串表示签名失败。

## Service/Bucket/Object 操作

所有与 Service/Bucket/Object 相关的方法原型，均表现为如下形式：

```
CosResult Operator(BaseReq, BaseResp)
```

### CosResult

CosResult 封装了请求出错时返回的错误码和对应错误信息，详见 错误码。

> **注意：**
>
> 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。<br>当返回false时：后续的 CosResult 成员函数才有意义；<br>当返回True时：可以从OperatorResp中获取具体返回内容。 |
| string GetErrorCode() | 获取 CSP 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。<br>使用遇到问题时，request-id 能更快地协助 CSP 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。<br>使用遇到问题时，trace-id 能更快地协助 CSP 定位问题。<br>当请求出错时，trace-id 与 request-id 一一对应。 |
| string GetErrorInfo() | 获取 SDK 内部错误信息。 |
| int GetHttpStatus() | 获取 http 状态码。 |

**BaseReq/BaseResp**

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（如后文介绍的 GetBucketReq），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，具体字段含义参见 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();
```

# Bucket 操作

## 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。相关 API 文档参见 Get Bucket。

### 方法原型

```
CosResult GetBucket(const GetBucketReq& req, GetBucketResp* resp);
```

### 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，枚举值：STANDARD，STANDARD_IA，NEARLINE
};
```

### 示例

```
qcloud_cos::CosConfig config("./config.json");
qcloud_cos::CosAPI cos(config);

std::string bucket_name = "cpp_sdk_v5-123456789";

// GetBucketReq 的构造函数需要传入 bucket_name
qcloud_cos::GetBucketReq req(bucket_name);
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;
}
```

### Put Bucket

#### 功能说明

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。相关 API 文档参见 Put Bucket。

#### 方法原型

```
CosResult PutBucket(const PutBucketReq& req, PutBucketResp* resp);
```

#### 参数说明

| 参数 | 参数描述 |
| --- | --- |
| req | PutBucketReq, PutBucket 操作的请求。 |
| 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);
```

#### 示例

```
qcloud_cos::CosConfig config("./config.json");
qcloud_cos::CosAPI cos(config);

std::string bucket_name = "cpp_sdk_v5-123456789";

qcloud_cos::PutBucketReq req(bucket_name);
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 {
// 创建 Bucket 失败，可以调用 CosResult 的成员函数输出错误信息，如requestID 等
}
```

### 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。相关 API 文档参见 Delete Bucket。

**方法原型**

```
CosResult DeleteBucket(const DeleteBucketReq& req, DeleteBucketResp* resp);
```

**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。 |

**示例**

```
qcloud_cos::CosConfig config("./config.json");
qcloud_cos::CosAPI cos(config);

std::string bucket_name = "cpp_sdk_v5-123456789";

// DeleteBucketReq 的构造函数需要传入 bucket_name
qcloud_cos::DeleteBucketReq req(bucket_name);
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 {
// 删除 Bucket 失败，可以调用 CosResult 的成员函数输出错误信息，如requestID 等
}
```

## Put Bucket CORS

**功能说明**

CPut 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。相关 API 文档参见 Put Bucket CORS。

**方法原型**

```
CosResult PutBucketCORS(const DPutBucketCORSReq& req, PutBucketCORSResp* resp);
```

**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。 |

```
// 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://www.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
};
```

**示例**

```
qcloud_cos::CosConfig config("./config.json");
qcloud_cos::CosAPI cos(config);

std::string bucket_name = "cpp_sdk_v5-123456789";

// 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://www.qq.com");
rule.m_allowed_origins.push_back("http://gsesgpucloud.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 {
// 设置生命周期失败，可以调用 CosResult 的成员函数输出错误信息，如 requestID 等
}
```

## 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 即跨域资源共享，全称 Cross-Origin Resource Sharing，是一个 W3C 标准。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。相关 API 文档参见 Get Bucket CORS。

**方法原型**

```
CosResult GetBucketCORS(const DGetBucketCORSReq& req, GetBucketCORSResp* resp);
```

**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。 |

```
// 获取 CORSRules, CORSRule 定义参见 Put Bucket CORS
std::vector<CORSRule> GetCORSRules();
```

**示例**

```
qcloud_cos::CosConfig config("./config.json");
qcloud_cos::CosAPI cos(config);

std::string bucket_name = "cpp_sdk_v5-123456789";

// 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 {
```

```
// 获取生命周期配置失败，可以调用 CosResult 的成员函数输出错误信息，如 requestID 等
}
```

### Delete Bucket CORS

**功能说明**

删除指定存储桶的跨域访问配置。相关 API 文档参见 Delete Bucket CORS。

**方法原型**

```
CosResult DeleteBucketCORS(const DDeleteBucketCORSReq& req, DeleteBucketCORSResp* resp);
```

**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。 |

**示例**

```
qcloud_cos::CosConfig config("./config.json");
qcloud_cos::CosAPI cos(config);

std::string bucket_name = "cpp_sdk_v5-123456789";

// 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 {
// 删除生命周期配置失败，可以调用 CosResult 的成员函数输出错误信息，如 requestID 等
}
```

### Put Bucket ACL

**功能说明**

Put Bucket ACL 接口用来写入 Bucket 的 ACL（Access Control List）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。相关 API 文档参见 Put Bucket ACL。

**方法原型**

```
CosResult PutBucketACL(const DPutBucketACLReq& req, PutBucketACLResp* resp);
```

**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。 |

```
// 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);
```

> **注意：**
>
> 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 中填写 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，WRITE，FULL_CONTROL
};
```

**示例**

```
qcloud_cos::CosConfig config("./config.json");
qcloud_cos::CosAPI cos(config);

std::string bucket_name = "cpp_sdk_v5-123456789";

// PutBucketACLReq 的构造函数需要传入 bucket_name
qcloud_cos::PutBucketACLReq req(bucket_name);
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://www.qq.com");
rule.m_allowed_origins.push_back("http://gsesgpucloud.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
}
```

### Get Bucket ACL

**功能说明**

Get Bucket ACL 接口用来获取 Bucket 的 ACL， 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。相关 API 文档参见 Get Bucket ACL。

**方法原型**

```
CosResult GetBucketACL(const DGetBucketACLReq& req, GetBucketACLResp* resp);
```

**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。 |

```
std::string GetOwnerID();
std::string GetOwnerDisplayName();
std::vector<Grant> GetAccessControlList();
```

**示例**

```
qcloud_cos::CosConfig config("./config.json");
qcloud_cos::CosAPI cos(config);

std::string bucket_name = "cpp_sdk_v5-123456789";

// GetBucketACLReq 的构造函数需要传入 bucket_name
qcloud_cos::GetBucketACLReq req(bucket_name);
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
}
```

## Object 操作

### Get Object

**功能说明**

Get Object 请求可以将一个文件（Object）下载至本地或指定流中。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）。

**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);
```

**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();

// 获取 Object 的存储类别，枚举值：STANDARD，STANDARD_IA，NEARLINE
std::string GetXCosStorageClass();

// 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();
```

**示例**

```
qcloud_cos::CosConfig config("./config.json");
qcloud_cos::CosAPI cos(config);

std::string bucket_name = "cpp_sdk_v5-123456789";
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 {
// 下载失败，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
}
}
```

## 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。

**方法原型**

```
CosResult HeadObject(const HeadObjectReq& req, HeadObjectResp* resp);
```

**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();
```

**示例**

```
qcloud_cos::CosConfig config("./config.json");
qcloud_cos::CosAPI cos(config);

std::string bucket_name = "cpp_sdk_v5-123456789";
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
}
```

## Put Object

### 功能说明

Put Object 请求可以将一个文件（Object）上传至指定 Bucket。

### 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);
```

### 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。 |

PutObject*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);

// x-cos-storage-class 设置 Object 的存储级别，枚举值：STANDARD,STANDARD_IA，NEARLINE，
// 默认值：STANDARD（目前仅支持华南园区）
void SetXCosStorageClass(const std::string& storage_class);

// 定义 Object 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);

/// 设置Server端加密使用的算法, 目前支持AES256
void SetXCosServerSideEncryption(const std::string& str);
```

PutObject*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();
```

**示例**

```
qcloud_cos::CosConfig config("./config.json");
qcloud_cos::CosAPI cos(config);

std::string bucket_name = "cpp_sdk_v5-123456789";
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);
// 调用 Set 方法设置元数据或者 ACL 等
req.SetXCosStorageClass("STANDARD_IA");
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");
// 调用 Set 方法设置元数据或者 ACL 等
req.SetXCosStorageClass("STANDARD_IA");
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;
}
}
```

### Delete Object

**功能说明**

Delete Object 接口请求可以在 CSP 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。相关 API 文档参见 Delete Object。

**方法原型**

```
CosResult DeleteObject(const DeleteObjectReq& req, DeleteObjectResp* resp);
```

**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。 |

**示例**

```
qcloud_cos::CosConfig config("./config.json");
qcloud_cos::CosAPI cos(config);

std::string bucket_name = "cpp_sdk_v5-123456789";
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 {
// 删除 Object 失败，可以调用 CosResult 的成员函数输出错误信息，比如 requestID 等
}
```

## 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。

**方法原型**

```
CosResult InitMultiUpload(const InitMultiUploadReq& req, InitMultiUploadResp* resp);
```

**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);

// x-cos-storage-class 设置 Object 的存储级别，枚举值：STANDARD,STANDARD_IA，NEARLINE，
// 默认值：STANDARD
void SetXCosStorageClass(const std::string& storage_class);

// 定义 Object 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);

/// 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， 分别表示分片上传的目标 Bucket、Object 名称以及后续分片上传所需的编号。

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();
```

**示例**

```
qcloud_cos::CosConfig config("./config.json");
qcloud_cos::CosAPI cos(config);
std::string bucket_name = "cpp_sdk_v5-123456789";
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();
}
```

## 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，需要携带 partNumber 和 uploadID，partNumber 为块的编号，支持乱序上传。

**方法原型**

```
CosResult UploadPartData(const UploadPartDataReq& request, UploadPartDataResp* response);
```

**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，请求还需要设置分片编号, 这个分片在完成分片上传时也会用到。

```
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();
```

**示例**

```
// 上传第一个分片
{
std::fstream is("demo_5M.part1");
qcloud_cos::UploadPartDataReq req(bucket_name, object_name,
upload_id, is);
req.SetPartNumber(1);
qcloud_cos::UploadPartDataResp resp;
qcloud_cos::CosResult result = cos.UploadPartData(req, &resp);

// 上传成功需要记录分片编号以及返回的 ETag
if (result.IsSucc()) {
etags.push_back(resp.GetEtag());
part_numbers.push_back(1);
}
is.close();
}

// 上传第二个分片
{
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片编号以及返回的 ETag
if (result.IsSucc()) {
etags.push_back(resp.GetEtag());
part_numbers.push_back(2);
}
is.close();
}
```

## 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。

**方法原型**

```
CosResult CompleteMultiUpload(const CompleteMultiUploadReq& request, CompleteMultiUploadResp* response);
```

**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片编号和 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();
```

**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);
```

## Multipart Upload

**功能说明**

Multipart Upload 封装了初始化分块上传、分块上传、完成分块上传三步, 只需要在请求中指明上传的文件。

**方法原型**

```
CosResult MultiUploadObject(const MultiUploadObjectReq& request, MultiUploadObjectResp* response);
```

**参数说明**

| 参数 | 参数描述 |
| --- | --- |

| 参数 | 参数描述 |
| --- | --- |
| req | MultiUploadObjectReq, MultiUploadObject 操作的请求。 |
| resp | MultiUploadObjectResp, MultiUploadObject 操作的返回。 |

MultiUploadObjectReq 需要在构造的时候指明 Bucket、Object 以及待上传文件的本地路径， 如果不指明本地路径，则默认是当前工作路径下与 Object 同名的文件。

```
MultiUploadObjectReq(const std::string& bucket_name,
const std::string& object_name, const std::string& local_file_path = "");

/// 设置Server端加密使用的算法, 目前支持AES256
void SetXCosServerSideEncryption(const std::string& str);
```

- 分块上传成功的情况下，该 Response 的返回内容与 CompleteMultiUploadResp 一致。
- 分块上传失败的情况下，该 Response 根据不同的失败情况，返回内容与 InitMultiUploadResp、UploadPartDataResp、CompleteMultiUploadResp 一致。可调用 `GetRespTag()` 来获取具体失败在哪一步。

```
// 返回 Init、Upload、Complete
std::string GetRespTag();

/// Server端加密使用的算法
std::string GetXCosServerSideEncryption();
```

**示例**

```
qcloud_cos::MultiUploadObjectReq req( bucket_name, object_name, "/temp/demo_6G.tmp");
qcloud_cos::MultiUploadObjectResp resp;
qcloud_cos::CosResult result = cos.MultiUploadObject(req, &resp);

if (result.IsSucc()) {
std::cout << resp.GetLocation() << std::endl;
std::cout << resp.GetKey() << std::endl;
std::cout << resp.GetBucket() << std::endl;
std::cout << resp.GetEtag() << std::endl;
} else {
// 获取具体失败在哪一步
std::string resp_tag = resp.GetRespTag();
if ("Init" == resp_tag) {
// print result
} else if ("Upload" == resp_tag) {
// print result
} else if ("Complete" == resp_tag) {
// print result
}
}
```

### 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。

**方法原型**

```
CosResult AbortMultiUpload(const AbortMultiUploadReq& request, AbortMultiUploadResp* response);
```

**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。

**示例**

```
qcloud_cos::AbortMultiUploadReq req(bucket_name, object_name,
upload_id);
qcloud_cos::AbortMultiUploadResp resp;
qcloud_cos::CosResult result = cos.AbortMultiUpload(req, &resp);
```

## 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。相关 API 文档参见 List Parts。

**方法原型**

```
CosResult ListParts(const ListPartsReq& req, ListPartsResp* resp);
```

**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。 |

```
// 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);


// 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();

// 用来表示这些分块的存储级别，枚举值：Standard，Standard_IA，nearline
std::string GetStorageClass();

// 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
};
```

**示例**

```
qcloud_cos::CosConfig config("./config.json");
qcloud_cos::CosAPI cos(config);

std::string bucket_name = "cpp_sdk_v5-123456789";
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
}
```

## Put Object ACL

**功能说明**

Put Object ACL 接口用来写入 Object 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。相关 API 文档参见 Put Object ACL。

**方法原型**

```
CosResult PutObjectACL(const PutObjectACLReq& req, PutObjectACLResp* resp);
```

**参数说明**

| 参数 | 参数描述 |
| --- | --- |

| 参数 | 参数描述 |
| --- | --- |
| req | PutObjectACLReq, PutObjectACL 操作的请求。 |
| resp | PutObjectACLResp, PutObjectACL 操作的返回。 |

```
// 定义 Object 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);

// 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);
```

> **注意：**
>
> 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 中填写 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，WRITE，FULL_CONTROL
};
```

**示例**

```
qcloud_cos::CosConfig config("./config.json");
qcloud_cos::CosAPI cos(config);

std::string bucket_name = "cpp_sdk_v5-123456789";
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://cam.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 {
// 设置 ACL，可以调用 CosResult 的成员函数输出错误信息，比如 requestID 等
}
}
```

### Get Object ACL

**功能说明**

Get Object ACL 接口用来获取 Object 的 ACL， 即对象（文件，Object）的访问权限控制列表。 此 API 接口只有 Object 的持有者有权限操作。相关 API 文档参见 Get Object ACL。

**方法原型**

```
CosResult GetObjectACL(const DGetObjectACLReq& req, GetObjectACLResp* resp);
```

**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。 |

```
std::string GetOwnerID();
std::string GetOwnerDisplayName();
std::vector<Grant> GetAccessControlList();
```

**示例**

```
qcloud_cos::CosConfig config("./config.json");
qcloud_cos::CosAPI cos(config);

std::string Object_name = "cpp_sdk_v5-123456789";

// GetObjectACLReq 的构造函数需要传入 Object_name
qcloud_cos::GetObjectACLReq req(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 {
// 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
}
```

### 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。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。建议文件大小 1MB 到 5GB，超过 5GB 的文件请使用分块上传 Upload - Copy。相关 API 文档参见 Put Object Copy。

**方法原型**

```
CosResult PutObjectCopy(const PutObjectCopyReq& req, PutObjectCopyResp* 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。 |

```
// 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);

// x-cos-storage-class 设置 Object 的存储级别，枚举值：STANDARD,STANDARD_IA，NEARLINE，
// 默认值：STANDARD（目前仅支持华南园区）
void SetXCosStorageClass(const std::string& storage_class);

// 定义Object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);

// 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);


// 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();
```

**示例**

```
qcloud_cos::CosConfig config("./config.json");
qcloud_cos::CosAPI cos(config);

std::string bucket_name = "cpp_sdk_v5-123456789";
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);
```

# C# SDK
# 快速入门

最近更新时间: 2024-12-19 17:12:00

## 下载与安装

### 相关资源

- 对象存储的 XML .NET SDK 源码下载地址： XML .NET SDK 。

### 环境依赖

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 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 CSP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 CSP 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 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: 2024-12-19 17:12:00

> **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：CSP术语信息

## 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 |

| 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 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 |
| 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://gsesgpucloud.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://gsesgpucloud.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 |

| 参数名称 | 设置方法 | 描述 | 类型 |
| --- | --- | --- | --- |
| 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 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 |
| --- | --- | --- | --- |
| 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 CSP 上 Object 的 对象键 | string |
| srcPath | 构造方法 | 用于上传到 CSP 的本地文件的绝对路径 | string |
| data | 构造方法 | 用于上传到 CSP 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 CSP 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 failCallback);

GetObjectBytesResult GetObject(GetObjectBytesRequest request);

void GetObject(GetObjectBytesRequest 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 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 CSP 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 CSP 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 CSP 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 CSP 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 CSP 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 CSP 的本地文件的绝对路径 | string |
| data | 构造方法 | 用于上传到 CSP 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 CSP 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 CSP 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: 2024-12-19 17:12:00

## 下载与安装

### 相关资源

- 对象存储的 XML Go SDK 源码下载地址： XML Go SDK 。
- 示例 Demo 下载地址： XML Go SDK 示例。
- 更多信息请参见 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 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 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 CSP 域名生成 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://examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com
u, _ := url.Parse("http://<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://examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com
u, _ := url.Parse("http://<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://examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com
u, _ := url.Parse("http://<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://examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com
u, _ := url.Parse("http://<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://examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com
u, _ := url.Parse("http://<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://examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com
u, _ := url.Parse("http://<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: 2024-12-19 17:12:00

# 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
}
```

```<table class="doc-table-container indent-undefined"><thead><tr><td class="">参数名称</td><td class="">参数描述</td><td class="">类型</td><td class="">必填</td></tr></thead><tbody><tr><td class="">XCosACL</td><td class="">设置 Bucket 的 ACL，如 private，public-read，public-read-write</td><td class="">string</td><td class="">否</td></tr><tr><td class="">XCosGrantFullControl</td><td class="">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"`</td><td class="">string</td><td class="">否</td></tr><tr><td class="">XCosGrantRead</td><td class="">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"`</td><td class="">string</td><td class="">否</td></tr><tr><td class="">XCosGrantWrite</td><td class="">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"`</td><td class="">string</td><td class="">否</td></tr></tbody></table>

### 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

```go
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
}
```

```<table class="doc-table-container indent-undefined"><thead><tr><td class="">参数名称</td><td class="">参数描述</td><td class="">类型</td><td class="">必填</td></tr></thead><tbody><tr><td class="">XCosACL</td><td class="">设置 Bucket 的 ACL，如 private，public-read，public-read-write</td><td class="">string</td><td class="">否</td></tr><tr><td class="">XCosGrantFullControl</td><td class="">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"`</td><td class="">string</td><td class="">否</td></tr><tr><td class="">XCosGrantRead</td><td class="">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"`</td><td class="">string</td><td class="">否</td></tr><tr><td class="">XCosGrantWrite</td><td class="">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"`</td><td class="">string</td><td class="">否</td></tr><tr><td class="">ACLXML</td><td class="">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</td><td class="">struct</td><td class="">否</td></tr></tbody></table>

### 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

```go
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
}
```<table class="doc-table-container indent-undefined"><thead><tr><td class="">参数名称</td><td class="">参数描述</td><td class="">类型</td></tr></thead><tbody><tr><td class="">Owner</td><td class="">Bucket 拥有者的信息，包括 DisplayName 和 ID</td><td class="">struct</td></tr><tr><td class="">AccessControlList</td><td class="">Bucket 权限授予者的信息，包括 Grantee和 Permission</td><td class="">struct</td></tr><tr><td class="">Grantee</td><td class="">权限授予者的信息，包括 DisplayName，Type，ID 和 UIN</td><td class="">struct</td></tr><tr><td class="">Type</td><td class="">权限授予者的类型，类型为 CanonicalUser 或者 Group</td><td class="">string</td></tr><tr><td class="">ID</td><td class="">Type 为 CanonicalUser 时，对应权限授予者的 ID</td><td class="">string</td></tr><tr><td class="">DisplayName</td><td class="">权限授予者的名字</td><td class="">string</td></tr><tr><td class="">UIN</td><td class="">Type 为 Group 时，对应权限授予者的 UIN</td><td class="">string</td></tr><tr><td class="">Permission</td><td class="">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</td><td class="">string</td></tr></tbody></table>

# 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**

<table class="doc-table-container indent-undefined"><thead><tr><td class="">API</td><td class="">操作名</td><td class="">操作描述</td></tr></thead><tbody><tr><td class="">GET Bucket（List Object）</td><td class="">查询对象列表</td><td class="">查询存储桶下的部分或者全部对象</td></tr><tr><td class="">PUT Object</td><td class="">简单上传对象</td><td class="">上传一个 Object（文件/对象）至 Bucket</td></tr><tr><td class="">HEAD Object</td><td class="">查询对象元数据</td><td class="">查询 Object 的 Meta 信息</td></tr><tr><td class="">GET Object</td><td class="">下载对象</td><td class="">下载一个 Object（文件/对象）至本地</td></tr><tr><td class="">PUT Object - Copy</td><td class="">设置对象复制</td><td class="">复制文件到目标路径</td></tr><tr><td class="">DELETE Object</td><td class="">删除单个对象</td><td class="">在 Bucket 中删除指定 Object（文件/对象）</td></tr><tr><td class="">DELETE Multiple Object</td><td class="">删除多个对象</td><td class="">在 Bucket 中批量删除 Object（文件/对象）</td></tr></tbody></table>

**分块操作**

<table class="doc-table-container indent-undefined"><thead><tr><td class="">API</td><td class="">操作名</td><td class="">操作描述</td></tr></thead><tbody><tr><td class="">List Multipart Uploads</td><td class="">查询分块上传</td><td class="">查询正在进行中的分块上传信息</td></tr><tr><td class="">Initiate Multipart Upload</td><td class="">初始化分块上传</td><td class="">初始化 Multipart Upload 上传操作</td></tr><tr><td class="">Upload Part</td><td class="">上传分块</td><td class="">分块上传文件</td></tr><tr><td class="">List Parts</td><td class="">查询已上传块</td><td class="">查询特定分块上传操作中的已上传的块</td></tr><tr><td class="">Complete Multipart Upload</td><td class="">完成分块上传</td><td class="">完成整个文件的分块上传</td></tr><tr><td class="">Abort Multipart Upload</td><td class="">终止分块上传</td><td class="">终止一个分块上传操作并删除已上传的块</td></tr></tbody></table>

**其他操作**

<table class="doc-table-container indent-undefined"><thead><tr><td class="">API</td><td class="">操作名</td><td class="">操作描述</td></tr></thead><tbody><tr><td class="">PUT Object acl</td><td class="">设置对象 ACL</td><td class="">设置 Bucket 中某个 Object（文件/对象）的 ACL</td></tr><tr><td class="">GET Object acl</td><td class="">查询对象 ACL</td><td class="">查询 Object（文件/对象）的 ACL</td></tr></tbody></table>

## 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

```go
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

```go
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

```go
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
}
```

<table class="doc-table-container indent-undefined"><thead><tr><td class="">参数名称</td><td class="">参数描述</td><td class="">类型</td><td class="">必填</td></tr></thead><tbody><tr><td class="">r</td><td class="">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`</td><td class="">io.Reader</td><td class="">是</td></tr><tr><td class="">key</td><td class="">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</td><td class="">string</td><td class="">是</td></tr><tr><td class="">XCosACL</td><td class="">设置文件的 ACL，如 private，public-read，public-read-write</td><td class="">string</td><td class="">否</td></tr><tr><td class="">XCosGrantFullControl</td><td class="">赋予被授权者所有的权限。格式：id="[OwnerUin]"</td><td class="">string</td><td class="">否</td></tr><tr><td class="">XCosGrantRead</td><td class="">赋予被授权者读的权限。格式：id="[OwnerUin]"</td><td class="">string</td><td class="">否</td></tr><tr><td class="">XCosStorageClass</td><td class="">设置文件的存储类型，默认值：STANDARD</td><td class="">string</td><td class="">否</td></tr><tr><td class="">Expires</td><td class="">设置 Content-Expires</td><td class="">string</td><td class="">否</td></tr><tr><td class="">CacheControl</td><td class="">缓存策略，设置 Cache-Control</td><td class="">string</td><td class="">否</td></tr><tr><td class="">ContentType</td><td class="">内容类型，设置 Content-Type</td><td class="">string</td><td class="">否</td></tr><tr><td class="">ContentDisposition</td><td class="">文件名称，设置 Content-Disposition</td><td class="">string</td><td class="">否</td></tr><tr><td class="">ContentEncoding</td><td class="">编码格式，设置 Content-Encoding</td><td class="">string</td><td class="">否</td></tr><tr><td class="">ContentLength</td><td class="">设置传输长度</td><td class="">string</td><td class="">否</td></tr><tr><td class="">XCosMetaXXX</td><td class="">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</td><td class="">http.Header</td><td class="">否</td></tr></tbody></table>

#### 返回结果说明

```go
{
```

'ETag': 'string',
'x-cos-expiration': 'string'
}
```<table class="doc-table-container indent-undefined"><thead><tr><td class="">参数名称</td><td class="">参数描述</td><td class="">类型</td></tr></thead><tbody><tr><td class="">ETag</td><td class="">上传文件的 MD5 值</td><td class="">string</td></tr><tr><td class="">x-cos-expiration</td><td class="">设置生命周期后，返回文件过期规则</td><td class="">string</td></tr></tbody></table>

### 查询对象元数据

#### 功能说明

查询 Object 的 Meta 信息（HEAD Object）。

#### 方法原型

``` go
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
}
```<table class="doc-table-container indent-undefined"><thead><tr><td class="">参数名称</td><td class="">参数描述</td><td class="">类型</td><td class="">必填</td></tr></thead><tbody><tr><td class="">key</td><td class="">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</td><td class="">string</td><td class="">是</td></tr><tr><td class="">IfModifiedSince</td><td class="">在指定时间后被修改才返回</td><td class="">string</td><td class="">否</td></tr></tbody></table>

#### 返回结果说明

``` go
{
'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=='
}
```<table class="doc-table-container indent-undefined"><thead><tr><td class="">参数名称</td><td class="">参数描述</td><td class="">类型</td></tr></thead><tbody><tr><td class="">文件元信息</td><td class="">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</td><td class="">string</td></tr></tbody></table>

### 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

``` go
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
}
```

<table class="doc-table-container indent-undefined"><thead><tr><td class="">参数名称</td><td class="">参数描述</td><td class="">类型</td><td class="">必填</td></tr></thead><tbody><tr><td class="">key</td><td class="">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</td><td class="">string</td><td class="">是</td></tr><tr><td class="">localfile</td><td class="">设置响应头部 Content-Type</td><td class="">string</td><td class="">是</td></tr><tr><td class="">ResponseContentType</td><td class="">设置响应头部 Content-Type</td><td class="">string</td><td class="">否</td></tr><tr><td class="">ResponseContentLanguage</td><td class="">设置响应头部 Content-Language</td><td class="">string</td><td class="">否</td></tr><tr><td class="">ResponseExpires</td><td class="">设置响应头部 Content-Expires</td><td class="">string</td><td class="">否</td></tr><tr><td class="">ResponseCacheControl</td><td class="">设置响应头部 Cache-Control</td><td class="">string</td><td class="">否</td></tr><tr><td class="">ResponseContentDisposition</td><td class="">设置响应头部 Content-Disposition</td><td class="">string</td><td class="">否</td></tr><tr><td class="">ResponseContentEncoding</td><td class="">设置响应头部 Content-Encoding</td><td class="">string</td><td class="">否</td></tr><tr><td class="">Range</td><td class="">设置下载文件的范围，格式为 bytes=first-last</td><td class="">string</td><td class="">否</td></tr><tr><td class="">IfModifiedSince</td><td class="">在指定时间后被修改才返回</td><td class="">string</td><td class="">否</td></tr></tbody></table>

#### 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=='
}
```

<table class="doc-table-container indent-undefined"><thead><tr><td class="">参数名称</td><td class="">参数描述</td><td class="">类型</td></tr></thead><tbody><tr><td class="">Body</td><td class="">下载文件的内容</td><td class="">StreamBody</td></tr><tr><td class="">文件元信息</td><td class="">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</td><td class="">string</td></tr></tbody></table>

### 设置对象复制

复制文件到目标路径（PUT Object-Copy）。

#### 方法原型

```go
func (s *ObjectService) Copy(ctx context.Context, key, sourceURL string, opt *ObjectCopyOptions) (*ObjectCopyResult, *Response, error)
```

**请求示例**

```
u, _ := url.Parse("http://examplebucket-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
}
```

<table class="doc-table-container indent-undefined"><thead><tr><td class="">参数名称</td><td class="">参数描述</td><td class="">类型</td><td class="">必填</td></tr></thead><tbody><tr><td class="">key</td><td class="">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</td><td class="">string</td><td class="">是</td></tr><tr><td class="">sourceURL</td><td class="">描述拷贝源文件的 URL</td><td class="">string</td><td class="">是</td></tr><tr><td class="">XCosACL</td><td class="">设置文件的 ACL，如 private，public-read，public-read-write</td><td class="">string</td><td class="">否</td></tr><tr><td class="">XCosGrantFullControl</td><td class="">赋予被授权者所有的权限。格式：id="[OwnerUin]"</td><td class="">string</td><td class="">否</td></tr><tr><td class="">XCosGrantRead</td><td class="">赋予被授权者读的权限。格式：id="[OwnerUin]"</td><td class="">string</td><td class="">否</td></tr><tr><td class="">XCosMetadataDirective</td><td class="">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</td><td class="">string</td><td class="">是</td></tr><tr><td class="">XCosCopySourceIfModifiedSince</td><td class="">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</td><td class="">string</td><td class="">否</td></tr><tr><td class="">XCosCopySourceIfUnmodifiedSince</td><td class="">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</td><td class="">string</td><td class="">否</td></tr><tr><td class="">XCosCopySourceIfMatch</td><td class="">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</td><td class="">string</td><td class="">否</td></tr><tr><td class="">XCosCopySourceIfNoneMatch</td><td class="">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</td><td class="">string</td><td class="">否</td></tr><tr><td class="">XCosStorageClass</td><td class="">设置文件的存储类型，默认值：STANDARD</td><td class="">string</td><td class="">否</td></tr><tr><td class="">XCosMetaXXX</td><td class="">用户自定义的文件元信息</td><td class="">http.Header</td><td class="">否</td></tr><tr><td class="">XCosCopySource</td><td class="">源文件 URL 路径，可以通过 versionid 子资源指定历史版本</td><td class="">string</td><td class="">否</td></tr></tbody></table>

#### 返回结果说明

上传文件的属性：

```go
type ObjectCopyResult struct {
ETag string
LastModified string
}
```<table class="doc-table-container indent-undefined"><thead><tr><td class="">参数名称</td><td class="">参数描述</td><td class="">类型</td></tr></thead><tbody><tr><td class="">ETag</td><td class="">拷贝文件的 MD5 值</td><td class="">string</td></tr><tr><td class="">LastModified</td><td class="">拷贝文件的最后一次修改时间</td><td class="">string</td></tr></tbody></table>

### 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

```go
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
}
```

<table class="doc-table-container indent-undefined"><thead><tr><td class="">参数名称</td><td class="">参数描述</td><td class="">类型</td><td class="">必填</td></tr></thead><tbody><tr><td class="">Quiet</td><td class="">布尔值，这个值决定了是否启动 Quiet 模式。对于响应结果，CSP 提供 Verbose 和 Quiet 两种模式：<br>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</td><td class="">Boolean</td><td class="">否</td></tr><tr><td class="">Objects</td><td class="">说明每个将要删除的目标 Object 信息</td><td class="">Container</td><td class="">是</td></tr><tr><td class="">Key</td><td class="">目标 Object 文件名称</td><td class="">String</td><td class="">是</td></tr></tbody></table>

#### 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
}
}
```

<table class="doc-table-container indent-undefined"><thead><tr><td class="">参数名称</td><td class="">参数描述</td><td class="">类型</td></tr></thead><tbody><tr><td class="">DeletedObjects</td><td class="">说明每个将要删除的目标 Object 信息</td><td class="">Container</td></tr><tr><td class="">Errors</td><td class="">说明本次删除的失败 Object 信息</td><td class="">Container</td></tr><tr><td class="">Key</td><td class="">删除失败的 Object 的名称</td><td class="">string</td></tr><tr><td class="">Code</td><td class="">删除失败的错误代码</td><td class="">string</td></tr><tr><td class="">Message</td><td class="">删除失败的错误信息</td><td class="">string</td></tr></tbody></table>

## 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
}
```<table class="doc-table-container indent-undefined"><thead><tr><td class="">参数名称</td><td class="">参数描述</td><td class="">类型</td><td class="">必填</td></tr></thead><tbody><tr><td class="">Delimiter</td><td class="">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</td><td class="">string</td><td class="">否</td></tr><tr><td class="">EncodingType</td><td class="">规定返回值的编码格式，合法值：url</td><td class="">string</td><td class="">否</td></tr><tr><td class="">Prefix</td><td class="">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</td><td class="">string</td><td class="">否</td></tr><tr><td class="">MaxUploads</td><td class="">设置最大返回的 multipart 数量，合法取值从1到1000，默认1000</td><td class="">string</td><td class="">否</td></tr><tr><td class="">KeyMarker</td><td class="">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</td><td class="">string</td><td class="">否</td></tr><tr><td class="">UploadIDMarker</td><td class="">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</td><td class="">string</td><td class="">否</td></tr></tbody></table>

#### 返回结果说明

``` go
// 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
}
```<table class="doc-table-container indent-undefined"><thead><tr><td class="">参数名称</td><td class="">参数描述</td><td class="">类型</td></tr></thead><tbody><tr><td class="">Bucket</td><td class="">分块上传的目标 Bucket，格式为 BucketName，如：examplebucket-1250000000</td><td class="">string</td></tr><tr><td class="">EncodingType</td><td class="">默认不编码，规定返回值的编码方式，可选值：url</td><td class="">string</td></tr><tr><td class="">KeyMarker</td><td class="">列出条目从该 key 值开始</td><td class="">string</td></tr><tr><td class="">UploadIDMarker</td><td class="">列出条目从该 UploadId 值开始</td><td class="">string</td></tr><tr><td class="">NextKeyMarker</td><td class="">假如返回条目被截断，则返回 NextKeyMarker 就是下一个条目的起点</td><td class="">string</td></tr><tr><td class="">NextUploadIDMarker</td><td class="">假如返回条目被截断，则返回 UploadId 就是下一个条目的起点</td><td class="">string</td></tr><tr><td class="">MaxUploads</td><td class="">最多返回的分块的数量，默认为最大的1000</td><td class="">string</td></tr><tr><td class="">IsTruncated</td><td class="">表示返回的分块是否被截断</td><td class="">bool</td></tr><tr><td class="">Uploads</td><td class="">每个 Upload 的信息</td><td class="">Container</td></tr><tr><td class="">Key</td><td class="">Object 的名称</td><td class="">string</td></tr><tr><td class="">UploadID</td><td class="">标示本次分块上传的 ID</td><td class="">string</td></tr><tr><td class="">Key</td><td class="">表示返回的分块是否被截断</td><td class="">bool</td></tr><tr><td class="">StorageClass</td><td class="">用来表示分块的存储级别，默认值：STANDARD</td><td class="">string</td></tr><tr><td class="">Initiator</td><td class="">用来表示本次上传发起者的信息</td><td class="">Container</td></tr><tr><td class="">Owner</td><td class="">用来表示这些分块所有者的信息</td><td class="">Container</td></tr><tr><td class="">Initiated</td><td class="">分块上传的起始时间</td><td class="">string</td></tr><tr><td class="">Prefix</td><td class="">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</td><td class="">struct</td></tr>
```

><tr><td class="">Delimiter</td><td class="">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</td><td class="">string</td></tr><tr><td class="">CommonPrefixes</td><td class="">将 prefix 到 delimiter 之间的相同路径归为一类，定义为 Common Prefix</td><td class="">string</td></tr><tr><td class="">ID</td><td class="">用户唯一的 CAM 身份 ID</td><td class="">string</td></tr><tr><td class="">DisplayName</td><td class="">用户身份 ID 的简称（UIN）</td><td class="">string</td></tr></tbody></table>

### 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

``` go
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
}
```
<table class="doc-table-container indent-undefined"><thead><tr><td class="">参数名称</td><td class="">参数描述</td><td class="">类型</td><td class="">必填</td></tr></thead><tbody><tr><td class="">r</td><td class="">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`</td><td class="">io.Reader</td><td class="">是</td></tr><tr><td class="">key</td><td class="">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</td><td class="">string</td><td class="">是</td></tr><tr><td class="">XCosACL</td><td class="">设置文件的ACL，如 private，public-read</td><td class="">string</td><td class="">否</td></tr><tr><td class="">XCosGrantFullControl</td><td class="">赋予被授权者所有的权限。格式：id="[OwnerUin]"</td><td class="">string</td><td class="">否</td></tr><tr><td class="">XCosGrantRead</td><td class="">赋予被授权者读的权限。格式：id="[OwnerUin]"</td><td class="">string</td><td class="">否</td></tr><tr><td class="">XCosStorageClass</td>

<td class="">设置文件的存储类型，默认值：STANDARD</td><td class="">string</td><td class="">否</td></tr><tr><td class="">Expires</td><td class="">设置 Content-Expires</td><td class="">string</td><td class="">否</td></tr><tr><td class="">CacheControl</td><td class="">缓存策略，设置 Cache-Control</td><td class="">string</td><td class="">否</td></tr><tr><td class="">ContentType</td><td class="">内容类型，设置 Content-Type</td><td class="">string</td><td class="">否</td></tr><tr><td class="">ContentDisposition</td><td class="">文件名称，设置 Content-Disposition</td><td class="">string</td><td class="">否</td></tr><tr><td class="">ContentEncoding</td><td class="">编码格式，设置 Content-Encoding</td><td class="">string</td><td class="">否</td></tr><tr><td class="">ContentLength</td><td class="">设置传输长度</td><td class="">string</td><td class="">否</td></tr><tr><td class="">XCosMetaXXX</td><td class="">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</td><td class="">http.Header</td><td class="">否</td></tr></tbody></table>

#### 返回结果说明

```go
type InitiateMultipartUploadResult struct {
Bucket string
Key string
UploadID string
}
```<table class="doc-table-container indent-undefined"><thead><tr><td class="">参数名称</td><td class="">参数描述</td><td class="">类型</td></tr></thead><tbody><tr><td class="">UploadId</td><td class="">标识分块上传的 ID</td><td class="">string</td></tr><tr><td class="">Bucket</td><td class="">Bucket 名称，由 bucket-appid 组成</td><td class="">string</td></tr><tr><td class="">Key</td><td class="">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</td><td class="">string</td></tr></tbody></table>

### <span id = "MULIT_UPLOAD_PART"> 上传分块 </span>

分块上传对象（Upload Part）。

#### 方法原型

```go
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
}
```<table class="doc-table-container indent-undefined"><thead><tr><td class="">参数名称</td><td class="">参数描述</td><td class="">类型</td><td class="">必填</td></tr></thead><tbody><tr><td class="">key</td><td class="">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</td><td class="">string</td><td class="">是</td></tr><tr><td class="">UploadId</td><td class="">标识分块上传的 ID，由 InitiateMultipartUpload 生成</td><td class="">string</td><td class="">是</td></tr><tr><td class="">PartNumber</td><td class="">标识上传分块的序号</td><td class="">int</td><td class="">是</td></tr><tr><td class="">r</td><td class="">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</td><td class="">io.Reader</td><td class="">是</td></tr><tr><td class="">ContentLength</td><td class="">设置传输长度</td><td class="">int</td><td class="">否</td></tr></tbody></table>

#### 返回结果说明

```go
{
'ETag': 'string'
}
```<table class="doc-table-container indent-undefined"><thead><tr><td class="">参数名称</td><td class="">参数描述</td><td class="">类型</td></tr></thead><tbody><tr><td class="">ETag</td><td class="">上传分块的 MD5 值</td><td class="">string</td></tr></tbody></table>

### <span id = "LIST_MULIT_UPLOAD"> 查询已上传块 </span>

#### 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

```go
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
}
```<table class="doc-table-container indent-undefined"><thead><tr><td class="">参数名称</td><td class="">参数描述</td><td class="">类型</td><td class="">必填</td></tr></thead><tbody><tr><td class="">key</td><td class="">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</td><td class="">string</td><td class="">是</td></tr><tr><td class="">UploadId</td><td class="">标识分块上传的 ID，由 InitiateMultipartUpload 生成</td><td class="">string</td><td class="">是</td></tr><tr><td class="">EncodingType</td><td class="">规定返回值的编码方式</td><td class="">string</td><td class="">否</td></tr><tr><td class="">MaxParts</td><td class="">单次返回最大的条目数量，默认1000</td><td class="">string</td><td class="">否</td></tr><tr><td class="">PartNumberMarker</td><td class="">默认以 UTF-8 二进制顺序列出条目，所有列出条目从 marker 开始</td><td class="">string</td><td class="">否</td></tr></tbody></table>

#### 返回结果说明

```go
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
}
```<table class="doc-table-container indent-undefined"><thead><tr><td class="">参数名称</td><td class="">参数描述</td><td class="">类型</td></tr></thead><tbody><tr><td class="">Bucket</td><td class="">存储桶名称，格式：BucketName-APPID。例如 examplebucket-1250000000</td><td class="">string</td></tr><tr><td class="">EncodingType</td><td class="">默认不编码，规定返回值的编码方式，可选值：url</td><td class="">string</td></tr><tr><td class="">Key</td><td class="">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</td><td class="">string</td></tr><tr><td class="">UploadId</td><td class="">标识分块上传的 ID，由 InitiateMultipartUpload 生成</td><td class="">string</td></tr><tr><td class="">Initiator</td><td class="">分块上传的创建者，包括 DisplayName，UIN 和 ID</td><td class="">struct</td></tr><tr><td class="">Owner</td><td class="">文件拥有者的信息，包括 DisplayName ，UIN 和 ID</td><td class="">struct</td></tr><tr><td class="">StorageClass</td><td class="">文件的存储类型，默认值：STANDARD</td><td class="">string</td></tr><tr><td class="">PartNumberMarker</td><td class="">默认为0，从第一块列出分块，从 PartNumberMarker 下一个分块开始列出</td><td class="">string</td></tr><tr><td class="">NextPartNumberMarker</td><td class="">指明下一次列出分块的起始位置</td><td class="">string</td></tr><tr><td class="">MaxParts</td><td class="">最多返回的分块的数量，默认为最大的1000</td><td class="">string</td></tr><tr><td class="">IsTruncated</td><td class="">表示返回的分块是否被截断</td><td class="">bool</td></tr><tr><td class="">Part</td><td class="">上传分块的相关信息，包括 ETag，PartNumber，Size，LastModified</td><td class="">struct</td></tr></tbody></table>

### <span id = "COMPLETE_MULIT_UPLOAD"> 完成分块上传 </span>

#### 功能说明

完成整个文件的分块上传（Complete Multipart Upload）。

#### 方法原型

```go
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
}
```
<table class="doc-table-container indent-undefined"><thead><tr><td class="">参数名称</td><td class="">参数描述</td><td class="">类型</td><td class="">必填</td></tr></thead><tbody><tr><td class="">key</td><td class="">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</td><td class="">string</td><td class="">是</td></tr><tr><td class="">UploadId</td><td class="">标识分块上传的 ID，由 InitiateMultipartUpload 生成</td><td class="">string</td><td class="">是</td></tr><tr><td class="">CompleteMultipartUploadOptions</td><td class="">所有分块的 ETag 和 PartNumber 信息</td><td class="">struct</td><td class="">是</td></tr></tbody></table>

#### 返回结果说明

```
type CompleteMultipartUploadResult struct {
Location string
Bucket string
Key string
ETag string
}
```
<table class="doc-table-container indent-undefined"><thead><tr><td class="">参数名称</td><td class="">参数描述</td><td class="">类型</td></tr></thead><tbody><tr><td class="">Location</td><td class="">URL 地址</td><td class="">string</td></tr><tr><td class="">Bucket</td><td class="">存储桶名称，格式：BucketName-APPID。例如 examplebucket-1250000000</td><td class="">string</td></tr><tr><td class="">Key</td><td class="">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</td><td class="">string</td></tr><tr><td class="">ETag</td><td class="">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</td><td class="">string</td></tr></tbody></table>

###  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

```go
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
}
```<table class="doc-table-container indent-undefined"><thead><tr><td class="">参数名称</td><td class="">参数描述</td><td class="">类型</td><td class="">必填</td></tr></thead><tbody><tr><td class="">key</td><td class="">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</td><td class="">string</td><td class="">是</td></tr><tr><td class="">XCosACL</td><td class="">设置 Object 的 ACL，如 private，public-read</td><td class="">string</td><td class="">否</td></tr><tr><td class="">XCosGrantFullControl</td><td class="">赋予被授权者所有的权限。格式：id="[OwnerUin]"</td><td class="">string</td><td class="">否</td></tr><tr><td class="">XCosGrantRead</td><td class="">赋予被授权者读的权限。格式：id="[OwnerUin]"</td><td class="">string</td><td class="">否</td></tr><tr><td class="">ACLXML</td><td class="">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</td><td class="">struct</td><td class="">否</td></tr></tbody></table>

### 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

```go
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
}
```<table class="doc-table-container indent-undefined"><thead><tr><td class="">参数名称</td><td class="">参数描述</td><td class="">类型</td></tr></thead><tbody><tr><td class="">Owner</td><td class="">Bucket 拥有者的信息，包括 DisplayName 和 ID</td><td class="">struct</td></tr><tr>

<td class="">AccessControlList</td><td class="">Bucket 权限授予者的信息，包括 Grantee 和 Permission</td><td class="">struct</td></tr><tr><td class="">Grantee</td><td class="">权限授予者的信息，包括 DisplayName，Type，ID 和 UIN</td><td class="">struct</td></tr><tr><td class="">Type</td><td class="">权限授予者的类型，类型为 CanonicalUser 或者 Group</td><td class="">string</td></tr><tr><td class="">ID</td><td class="">Type 为 CanonicalUser 时，对应权限授予者的 ID</td><td class="">string</td></tr><tr><td class="">DisplayName</td><td class="">权限授予者的名字</td><td class="">string</td></tr><tr><td class="">UIN</td><td class="">Type 为 Group 时，对应权限授予者的 UIN</td><td class="">string</td></tr><tr><td class="">Permission</td><td class="">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</td><td class="">string</td></tr></tbody></table>

# 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**

<table class="doc-table-container indent-undefined"><thead><tr><td class="">API</td><td class="">操作名</td><td class="">操作描述</td></tr></thead><tbody><tr><td class="">PUT Bucket cors</td><td class="">设置跨域配置</td><td class="">设置存储桶的跨域访问权限</td></tr><tr><td class="">GET Bucket cors</td><td class="">查询跨域配置</td><td class="">查询存储桶的跨域访问配置信息</td></tr><tr><td class="">DELETE Bucket cors</td><td class="">删除跨域配置</td><td class="">删除存储桶的跨域访问配置信息</td></tr></tbody></table>

**版本控制**

<table class="doc-table-container indent-undefined"><thead><tr><td class="">API</td><td class="">操作名</td><td class="">操作描述</td></tr></thead><tbody><tr><td class="">PUT Bucket versioning</td><td class="">设置版本控制</td><td class="">设置存储桶的版本控制功能</td></tr><tr><td class="">GET Bucket versioning</td><td class="">查询版本控制</td><td class="">查询存储桶的版本控制信息</td></tr></tbody></table>

**跨地域复制**

<table class="doc-table-container indent-undefined"><thead><tr><td class="">API</td><td class="">操作名</td><td class="">操作描述</td></tr></thead><tbody><tr><td class="">PUT Bucket replication</td><td class="">设置跨地域复制</td><td class="">设置存储桶的跨地域复制规则</td></tr><tr><td class="">GET Bucket replication</td><td class="">查询跨地域复制</td><td class="">查询存储桶的跨地域复制规则</td></tr><tr><td class="">DELETE Bucket replication</td><td class="">删除跨地域复制</td><td class="">删除存储桶的跨地域复制规则</td></tr></tbody></table>

## 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

```go
func (s *BucketService) PutCORS(ctx context.Context, opt *BucketPutCORSOptions) (*Response, error)
```

**请求示例**

```
opt := &cos.BucketPutCORSOptions{
Rules: []cos.BucketCORSRule{
{
AllowedOrigins: []string{"http://www.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://www.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
}
```

<table class="doc-table-container indent-undefined"><thead><tr><td class="">参数名称</td><td class="">参数描述</td><td class="">类型</td><td class="">必填</td></tr></thead><tbody><tr><td class="">BucketCORSRule</td><td class="">设置对应的跨域规则，包括 ID，MaxAgeSeconds，AllowedOrigin，AllowedMethod，AllowedHeader，ExposeHeader</td><td class="">struct</td><td class="">是</td></tr><tr><td class="">ID</td><td class="">设置规则的 ID</td><td class="">string</td><td class="">否</td></tr><tr><td class="">AllowedMethods</td><td class="">设置允许的方法，如 GET，PUT，HEAD，POST，DELETE</td><td class="">[]string</td><td class="">是</td></tr><tr><td class="">AllowedOrigins</td><td class="">设置允许的访问来源，如 `"http://gsesgpucloud.com"`，支持通配符 *</td><td class="">[]string</td><td class="">是</td></tr><tr><td class="">AllowedHeaders</td><td class="">设置请求可以使用哪些自定义的 HTTP 请求头部，支持通配符 *</td><td class="">[]string</td><td class="">否</td></tr><tr><td class="">MaxAgeSeconds</td><td class="">设置 OPTIONS 请求得到结果的有效期</td><td class="">int</td><td class="">否</td></tr><tr><td class="">ExposeHeaders</td><td class="">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</td><td class="">[]string</td><td class="">否</td></tr></tbody></table>

### 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

```go
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
}
```

```<table class="doc-table-container indent-undefined"><thead><tr><td class="">参数名称</td><td class="">参数描述</td><td class="">类型</td><td class="">必填</td></tr></thead><tbody><tr><td class="">BucketCORSRule</td><td class="">设置对应的跨域规则，包括 ID，MaxAgeSeconds，AllowedOrigin，AllowedMethod，AllowedHeader，ExposeHeader</td><td class="">struct</td><td class="">是</td></tr><tr><td class="">ID</td><td class="">设置规则的 ID</td><td class="">string</td><td class="">否</td></tr><tr><td class="">AllowedMethods</td><td class="">设置允许的方法，如 GET，PUT，HEAD，POST，DELETE</td><td class="">[]string</td><td class="">是</td></tr><tr><td class="">AllowedOrigins</td><td class="">设置允许的访问来源，如 `"http://gsesgpucloud.com"`，支持通配符 *</td><td class="">[]string</td><td class="">是</td></tr><tr><td class="">AllowedHeaders</td><td class="">设置请求可以使用哪些自定义的 HTTP 请求头部，支持通配符 *</td><td class="">[]string</td><td class="">否</td></tr><tr><td class="">MaxAgeSeconds</td><td class="">设置 OPTIONS 请求得到结果的有效期</td><td class="">int</td><td class="">否</td></tr><tr><td class="">ExposeHeaders</td><td class="">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</td><td class="">[]string</td><td class="">否</td></tr></tbody></table>

### 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

```go
func (s *BucketService) DeleteCORS(ctx context.Context) (*Response, error)
```

**请求示例**

```
resp, err := client.Bucket.DeleteCORS(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 |
| 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 [CSP 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[ CSP 错误码] | string |

# Java SDK
# 快速入门

最近更新时间: 2024-12-19 17:12:00

## 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：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。
- CSP 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 CSP 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 CSP 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 | CSP 中用于存储数据的容器。 |

| 名称 | 描述 |
| --- | --- |
| Object | CSP 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。<br>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，CSP 服务为上传文件提供的一种分块上传模式。 |

### 导入类名

CSP 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 CSP 服务相关请求之前，都需要先生成 COSClient 类的对象， COSClient 是调用 CSP 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 CSP 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 CSP 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 CSP 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 CSP 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 CSP 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 CSP，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 CSP 上已存在同样 Key 的对象，上传时则会覆盖旧的对象。
- 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 CSP 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），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 CSP 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 CSP 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 CSP 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: 2024-12-19 17:12:00

# 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<br>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）<br>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 CSP。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 CSP 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 CSP
public PutObjectResult putObject(String bucketName, String key, File file)
throws CosClientException, CosServiceException;
// 方法2 输入流上传到 CSP
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。<br>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 |
| --- | --- | --- |

| 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 |
| --- | --- | --- |

| 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CSP,会抛出异常
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 |
| --- | --- | --- |

| 参数名称 | 描述 | 类型 |
| --- | --- | --- |
| bucketName | Bucket 的命名格式为 BucketName-APPID | String |
| 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 CSP 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 CSP 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 CSP 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 CSP 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 CSP 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 CSP 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。

> **说明：**
>
> 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**

> 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 CSP 网络充足(如使用亿算云平台的 CVM，同地域上传 CSP)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。<br>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 CSP 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
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 |
| --- | --- | --- |
| 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://mail.qq.com", "http://www.qq.com", "http://video.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);
```

# 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 CSP 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 CSP 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
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 CSP 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 CSP 签名
public String buildAuthorizationStr(HttpMethodName methodName, String resouce_path,
COSCredentials cred, Date expiredTime);

// 构造 CSP 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 CSP 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 |
| 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 [CSP 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 [CSP 错误信息] | String |
| errorMessage | 请求失败时 body 返回的 Error Message 请参照 [CSP 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: 2024-12-19 17:12:00

## 开发准备

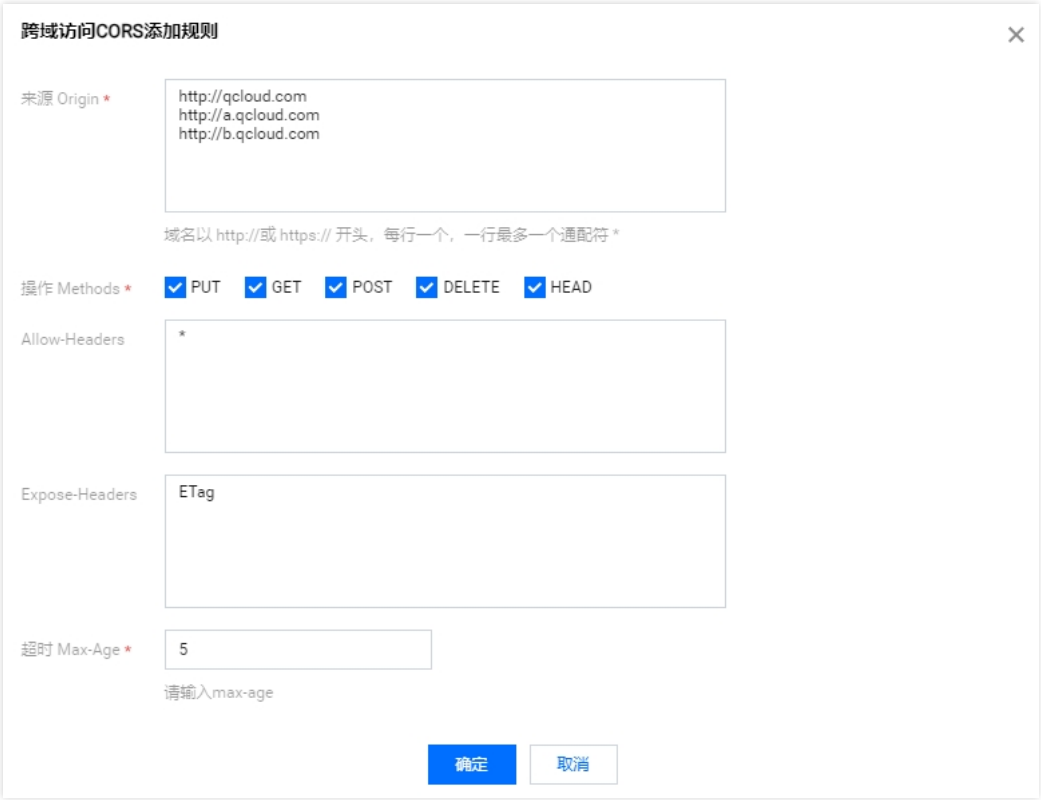
### SDK 获取

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 CSP 控制台创建存储桶，得到 Bucket（存储桶名称）和 Region（地域名称）。
3. 到控制台密钥管理获取您的项目 SecretId 和 SecretKey。
4. 配置 CORS 规则，配置例子如下图：




> **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：CSP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: 2024-12-19 17:12:00

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：CSP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 CSP SDK 实例：

```
var cos = new COS({
// 必选参数
getAuthorization: function (options, callback) {
$.get('http://example.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://example.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://example.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 |
| getAuthorization | 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 | 计算得到的签名字符串。 | 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：

- 实例化时传入 SecretId、SecretKey，每次需要签名都由实例内部计算。
- 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。
- 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

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：

- 放在 header 参数里使用，字段名: authorization。
- 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 |
| --- | --- | --- | --- |

| 参数名 | 参数描述 | 类型 | 必填 |
| --- | --- | --- | --- |
| SecretId | 用户的 SecretId。 | String | 是 |
| 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：

- 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。
- 否则返回不带签名的 url。

**回调函数说明**

function(err, data) { ... }
```<table class="doc-table-container indent-undefined"><thead><tr><td class="">参数名</td><td class="">参数描述</td><td class="">类型</td></tr></thead><tbody><tr><td class="">err</td><td class="">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。</td><td class="">Object</td></tr><tr><td class="">data</td><td class="">请求成功时返回的对象，如果请求发生错误，则为空。</td><td class="">Object</td></tr><tr><td class="">- Url</td><td class="">计算得到的 Url。</td><td class="">String</td></tr></tbody></table>

## 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

```js
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 |
```

| 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) { ... }
```<table class="doc-table-container indent-undefined"><thead><tr><td class="">参数名</td><td class="">参数描述</td><td class="">类型</td></tr></thead><tbody><tr><td class="">err</td><td class="">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。</td><td class="">Object</td></tr><tr><td class="">- statusCode</td><td class="">请求返回的 HTTP 状态码，如 200，403，404 等。</td><td class="">Number</td></tr><tr><td class="">- headers</td><td class="">请求返回的头部信息。</td><td class="">Object</td></tr><tr><td class="">data</td><td class="">请求成功时返回的对象，如果请求发生错误，则为空。</td><td class="">Object</td></tr><tr><td class="">- statusCode</td><td class="">请求返回的 HTTP 状态码，如 200，403，404 等。</td><td class="">Number</td></tr><tr><td class="">- headers</td><td class="">请求返回的头部信息。</td><td class="">Object</td></tr></tbody></table>

### 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：

```js
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 |
```

| 参数名 | 参数描述 | 类型 | 必填 |
| --- | --- | --- | --- |
| 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 |
| --- | --- | --- |
| 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"`，<br>**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 |
| 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 |

| 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 200，403，404 等。 | Number |
| - headers | 请求返回的头部信息。 | Object |
| - Owner | Bucket 持有者信息。 | Object |
| - - DisplayName | Bucket 持有者的名称。 | String |
| - - ID | Bucket 持有者 ID，<br>格式：qcs::cam::uin/:uin/<br>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；<br>当 type 类型为 RootAccount 时，ID 中指定的是根帐号。<br>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，<br>如果是根帐号，格式为：qcs::cam::uin/:uin/<br>或 qcs::cam::anyone:anyone （指代所有用户）<br>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);
});
```

- 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=" " ；<br>当需要给子账户授权时，id="qcs::cam::uin/:uin/" ，<br>当需要给根账户授权时，id="qcs::cam::uin/:uin/" ，<br>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=" " ；<br>当需要给子账户授权时，id="qcs::cam::uin/:uin/" ，<br>当需要给根账户授权时，id="qcs::cam::uin/:uin/" ，<br>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=" " ；<br>当需要给子账户授权时，id="qcs::cam::uin/:uin/" ，<br>当需要给根账户授权时，id="qcs::cam::uin/:uin/" ，<br>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 |

| 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 |

## 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://www.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：
>
> - 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 。
> - 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://www.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：**
>
> - 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。
> - 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 |

| 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 200，304 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 |
| - 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 CSP 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 |

版权所有：亿算云平台

| 参数名 | 参数描述 | 类型 | 必填 |
| --- | --- | --- | --- |
| 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，<br>**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：**
>
> - Key（文件名）不能以 / 结尾，否则会被识别为文件夹。
> - 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);
});
```

- 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=" "；<br>当需要给子账户授权时，id="qcs::cam::uin/:uin/"，<br>当需要给根账户授权时，id="qcs::cam::uin/:uin/"，<br>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=" "；<br>当需要给子账户授权时，id="qcs::cam::uin/:uin/"，<br>当需要给根账户授权时，id="qcs::cam::uin/:uin/"，<br>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=" "；<br>当需要给子账户授权时，id="qcs::cam::uin/:uin/"，<br>当需要给根账户授权时，id="qcs::cam::uin/:uin/"，<br>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 |

| 参数名 | 参数描述 | 类型 | 必填 |
| --- | --- | --- | --- |
| progressData.loaded | 已经下载的文件部分大小，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，<br>**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 CSP 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 |
| --- | --- | --- |

| 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，**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 CSP，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://www.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 |

| 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 |
| - 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 |
| - 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 |

| 参数名 | 参数描述 | 类型 |
| --- | --- | --- |
| - Owner | 标识资源的所有者。 | Object |
| - ID | Object 持有者 ID，格式：qcs::cam::uin/:uin/<br>如果是根帐号， 和 是同一个值。 | String |
| - 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/<br>或 qcs::cam::anyone:anyone（指代所有用户）<br>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=" "；<br>当需要给子账户授权时，id="qcs::cam::uin/:uin/"，<br>当需要给根账户授权时，id="qcs::cam::uin/:uin/"，<br>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=" "；<br>当需要给子账户授权时，id="qcs::cam::uin/:uin/"，<br>当需要给根账户授权时，id="qcs::cam::uin/:uin/"，<br>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。<br>格式：id=" ",id=" "；当需要给子账户授权时，id="qcs::cam::uin/:uin/"，<br>当需要给根账户授权时，id="qcs::cam::uin/:uin/"，<br>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，CSP 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 |
| - 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 |
| 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=" "；<br>当需要给子账户授权时，id="qcs::cam::uin/:uin/"，<br>当需要给根账户授权时，id="qcs::cam::uin/:uin/"，<br>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=" "；<br>当需要给子账户授权时，id="qcs::cam::uin/:uin/"，<br>当需要给根账户授权时，id="qcs::cam::uin/:uin/"，<br>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=" "；<br>当需要给子账户授权时，id="qcs::cam::uin/:uin/"，<br>当需要给根账户授权时，id="qcs::cam::uin/:uin/"，<br>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 |
| --- | --- | --- |

| 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 |
| - 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=" "；<br>当需要给子账户授权时，id="qcs::cam::uin/:uin/"，<br>当需要给根账户授权时，id="qcs::cam::uin/:uin/"，<br>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=" "；<br>当需要给子账户授权时，id="qcs::cam::uin/:uin/"，<br>当需要给根账户授权时，id="qcs::cam::uin/:uin/"，<br>例如：'id="qcs::cam::uin/123:uin/123", id="qcs::cam::uin/123:uin/456"' | 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：id=" ",id=" "；<br>当需要给子账户授权时，id="qcs::cam::uin/:uin/"，<br>当需要给根账户授权时，id="qcs::cam::uin/:uin/"，<br>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 |

**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 |

| 参数名 | 参数描述 | 类型 | 必填 |
| --- | --- | --- | --- |
| 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，CSP 就立即返回 200 的状态码，在合并的过程中，CSP 会周期性的返回空格信息来保持连接活跃，直到合并完成，CSP 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 |

| 参数名 | 参数描述 | 类型 | 必填 |
| --- | --- | --- | --- |
| 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 |

| 参数名 | 参数描述 | 类型 | 必填 |
| --- | --- | --- | --- |
| 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/<br>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 |
| 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，<br>• 当 upload-id-marker 未被指定时：<br><br>ObjectName 字母顺序大于 key-marker 的条目将被列出，<br>• 当upload-id-marker被指定时：<br><br>ObjectName 字母顺序大于key-marker的条目被列出，<br>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，<br>• 当 key-marker 未被指定时：<br><br>upload-id-marker 将被忽略，<br>• 当 key-marker 被指定时：<br><br>ObjectName字母顺序大于 key-marker 的条目被列出，<br>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 |

版权所有：亿算云平台

| 参数名 | 参数描述 | 类型 |
| --- | --- | --- |
| - 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 |
| - 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/<br>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/<br>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 |
| 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：校验 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：上传 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 |

| 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 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: 2024-12-19 17:12:00

## 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. 到控制台密钥管理获取您的项目 SecretId 和 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：CSP术语信息

## 快速入门

1. 到 CSP 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）。
2. 到控制台密钥管理获取您的项目 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: 2024-12-19 17:12:00

> **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：CSP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 |

| 参数名 | 参数描述 | 类型 |
| --- | --- | --- |
| 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=" "，<br>当需要给子账户授权时，uin="RootAcountID/SubAccountID"，<br>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=" "，<br>当需要给子账户授权时，uin="RootAcountID/SubAccountID"，<br>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=" "，<br>当需要给子账户授权时，uin="RootAcountID/SubAccountID"，<br>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 |
| 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 |
| Subaccount | 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 |

| 参数名 | 参数描述 | 类型 | 必填 |
| --- | --- | --- | --- |
| 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=" "，<br>当需要给子账户授权时，uin="RootAcountID/SubAccountID"，<br>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=" "，<br>当需要给子账户授权时，uin="RootAcountID/SubAccountID"，<br>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=" "，<br>当需要给子账户授权时，uin="RootAcountID/SubAccountID"，<br>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 |
| --- | --- | --- |
| 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 接口请求实现删除跨域访问配置信息。

**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=" "，<br>当需要给子账户授权时，uin="RootAcountID/SubAccountID"，<br>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=" "，<br>当需要给子账户授权时，uin="RootAcountID/SubAccountID"，<br>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=" "，<br>当需要给子账户授权时，uin="RootAcountID/SubAccountID"，<br>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 |
| Subaccount | 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=" "，<br>当需要给子账户授权时，uin="RootAcountID/SubAccountID"，<br>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=" "，<br>当需要给子账户授权时，uin="RootAcountID/SubAccountID"，<br>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=" "，<br>当需要给子账户授权时，uin="RootAcountID/SubAccountID"，<br>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，CSP 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=" "，<br>当需要给子账户授权时，uin="RootAcountID/SubAccountID"，<br>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=" "，<br>当需要给子账户授权时，uin="RootAcountID/SubAccountID"，<br>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=" "，<br>当需要给子账户授权时，uin="RootAcountID/SubAccountID"，<br>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 | 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 |
| 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，<br>• 当 upload-id-marker 未被指定时，<br>ObjectName 字母顺序大于 key-marker 的条目将被列出；<br>• 当 upload-id-marker 被指定时，<br>ObjectName 字母顺序大于 key-marker 的条目被列出，<br>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<br>• 当key-marker未被指定时，<br>upload-id-marker将被忽略；<br>• 当key-marker被指定时，<br>ObjectName字母顺序大于key-marker的条目被列出，<br>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 |

| 参数名 | 参数描述 | 类型 |
| --- | --- | --- |
| 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 |
| 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 |

| 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 |
| 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: 2024-12-19 17:12:00

## 下载与安装

### 相关资源

- 对象存储的 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。

> **说明：**
>
> 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 https://getcomposer.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 CSP 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 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 CSP 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 | CSP 中用于存储数据的容器 |
| Object | CSP 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。<br>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，CSP 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。

**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。

#### 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: 2024-12-19 17:12:00

# 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 |

### 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 |
| 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://lewzylucd2-1251668577.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 |
| --- | --- | --- | --- |
| 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 |

| 参数名称 | 类型 | 描述 | 必填 |
| --- | --- | --- | --- |
| SaveAs | String | 保存到本地的本地文件路径 | 否 |
| 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 |

| 参数名称 | 类型 | 描述 | 父节点 |
| --- | --- | --- | --- |
| Expires | String | Content-Expires | 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 |
| 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 |

| 参数名称 | 类型 | 描述 | 必填 |
| --- | --- | --- | --- |
| 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 |

| 参数名称 | 类型 | 描述 | 必填 |
| --- | --- | --- | --- |
| 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 |

版权所有：亿算云平台

| 参数名称 | 类型 | 描述 | 父节点 |
| --- | --- | --- | --- |
| 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 |
| 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 CSP 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、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 |

| API | 操作名 | 操作描述 |
| --- | --- | --- |
| 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 |

| 参数名称 | 类型 | 描述 | 必填 |
| --- | --- | --- | --- |
| 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://www.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://www.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://www.qq.com` | CORSRule |
| 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 |

# 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 CSP 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，更多详情请参阅 错误码 | string |
| errorCode | 请求失败时 body 返回的 Error Code，更多详情请参阅 错误码 | string |
| errorMessage | 请求失败时 body 返回的 Error Message，更多详情请参阅 错误码 | 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: 2024-12-19 17:12:00

## 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 CSP 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 | CSP 中用于存储数据的容器 |
| Object | CSP 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。<br>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，CSP 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 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 CSP，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 https://github.com/tencentyun/cos-python-sdk-v5/blob/master/qcloud_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('/130619361938403328/130619386799091712')
```

```
# 网络流将以 Transfer-Encoding:chunked 的方式传输到 CSP
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: 2024-12-19 17:12:00

# 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'
},
]
}
```

<table class="doc-table-container indent-undefined"><thead><tr><td class="">参数名称</td><td class="">参数描述</td><td class="">类型</td></tr></thead><tbody><tr><td class="">Owner</td><td class="">存储桶拥有者的信息，包括 DisplayName 和 ID</td><td class="">Dict</td></tr><tr><td class="">Grant</td><td class="">存储桶权限授予者的信息，包括 Grantee 和 Permission</td><td class="">List</td></tr><tr><td class="">Grantee</td><td class="">权限授予者的信息，包括 DisplayName，Type，ID 和 URI</td><td class="">Dict</td></tr><tr><td class="">DisplayName</td><td class="">权限授予者的名字</td><td class="">String</td></tr><tr><td class="">Type</td><td class="">权限授予者的类型，类型为 CanonicalUser 或者 Group</td><td class="">String</td></tr><tr><td class="">ID</td><td class="">Type 为 CanonicalUser 时，对应权限授予者的 ID</td><td class="">String</td></tr><tr><td class="">URI</td><td class="">Type 为 Group 时，对应权限授予者的 URI</td><td class="">String</td></tr><tr><td class="">Permission</td><td class="">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</td><td class="">String</td></tr></tbody></table>

# 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**

<table class="doc-table-container indent-undefined"><thead><tr><td class="">API</td><td class="">操作名</td><td class="">操作描述</td></tr></thead><tbody><tr><td class="">GET Bucket（List Object）</td><td class="">查询对象列表</td><td class="">查询存储桶下的部分或者全部对象</td></tr><tr><td class="">GET Bucket Object Versions</td><td class="">查询对象及其历史版本列表</td><td class="">查询存储桶下的部分或者全部对象及其历史版本信息</td></tr><tr><td class="">PUT Object</td><td class="">简单上传对象</td><td class="">上传一个对象至存储桶</td></tr><tr><td class="">HEAD Object</td><td class="">查询对象元数据</td><td class="">查询对象的元数据信息</td></tr><tr><td class="">GET Object</td><td class="">下载对象</td><td class="">下载一个对象至本地</td></tr><tr><td class="">PUT Object - Copy</td><td class="">设置对象复制</td><td class="">复制对象到目标路径</td></tr><tr><td class="">DELETE Object</td><td class="">删除单个对象</td><td class="">在存储桶中删除指定对象</td></tr><tr><td class="">DELETE Multiple Objects</td><td class="">删除多个对象</td><td class="">在存储桶中批量删除指定对象</td></tr></tbody></table>

**分块操作**

<table class="doc-table-container indent-undefined"><thead><tr><td class="">API</td><td class="">操作名</td><td class="">操作描述</td></tr></thead><tbody><tr><td class="">List Multipart Uploads</td><td class="">查询分块上传</td><td class="">查询正在进行中的分块上传信息</td></tr><tr><td class="">Initiate Multipart Upload</td><td class="">初始化分块上传</td><td class="">初始化分块上传任务</td></tr><tr><td class="">Upload Part</td><td class="">上传分块</td><td class="">分块上传文件</td></tr><tr><td class="">Upload Part - Copy</td><td class="">复制分块</td><td class="">将其他对象复制为一个分块</td></tr><tr><td class="">List Parts</td><td class="">查询已上传块</td><td class="">查询特定分块上传操作中的已上传的块</td></tr><tr><td class="">Complete Multipart Upload</td><td class="">完成分块上传</td><td class="">完成整个文件的分块上传</td></tr><tr><td class="">Abort Multipart Upload</td><td class="">终止分块上传</td><td class="">终止一个分块上传操作并删除已上传的块</td></tr></tbody></table>

版权所有：亿算云平台

**其他操作**
<table class="doc-table-container indent-undefined"><thead><tr><td class="">API</td><td class="">操作名</td><td class="">操作描述</td></tr></thead><tbody><tr><td class="">PUT Object acl</td><td class="">设置对象 ACL</td><td class="">设置存储桶中某个对象的访问控制列表</td></tr><tr><td class="">GET Object acl</td><td class="">查询对象 ACL</td><td class="">查询存储桶中某个对象的访问控制列表</td></tr></tbody></table>

## 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

```python
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'
},
],
}
```

<table class="doc-table-container indent-undefined"><thead><tr><td class="">参数名称</td><td class="">参数描述</td><td class="">类型</td></tr></thead><tbody><tr><td class="">MaxKeys</td><td class="">最多返回的对象数量，默认为最大的1000</td><td class="">String</td></tr><tr><td class="">Prefix</td><td class="">默认为空，匹配 Prefix 为前缀的对象</td><td class="">String</td></tr><tr><td class="">Delimiter</td><td class="">默认为空，设置分隔符，例如设置`/`来模拟文件夹</td><td class="">String</td></tr><tr><td class="">Marker</td><td class="">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</td><td class="">String</td></tr><tr><td class="">NextMarker</td><td class="">当 IsTruncated 为 true 时，标记下一次返回对象的 list 的起点位置</td><td class="">String</td></tr><tr><td class="">Name</td><td class="">存储桶名称，由 BucketName-APPID 构成</td><td class="">String</td></tr><tr><td class="">IsTruncated</td><td class="">表示返回的对象否被截断</td><td class="">String</td></tr><tr><td class="">EncodingType</td><td class="">默认不编码，规定返回值的编码方式，可选值：url</td><td class="">String</td></tr><tr><td class="">Contents</td><td class="">包含所有对象元数据的 list，包括 'ETag'，'StorageClass'，'Key'，'Owner'，'LastModified'，'Size' 等信息</td><td class="">List</td></tr><tr><td class="">CommonPrefixes</td><td class="">所有以 Prefix 开头，以 Delimiter 结尾的对象被归到同一类</td><td class="">List</td></tr></tbody></table>

### 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

``` python
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'
}
```
```<table class="doc-table-container indent-undefined"><thead><tr><td class="">参数名称</td><td class="">参数描述</td><td class="">类型</td></tr></thead><tbody><tr><td class="">ETag</td><td class="">上传对象的 MD5 值</td><td class="">String</td></tr><tr><td class="">x-cos-version-id</td><td class="">开启版本控制后，对象的版本号</td><td class="">String</td></tr></tbody></table>
```

### 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

```python
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=='
}
```

<table class="doc-table-container indent-undefined"><thead><tr><td class="">参数名称</td><td class="">参数描述</td><td class="">类型</td></tr></thead><tbody><tr><td class="">ETag</td><td class="">对象的 MD5 值</td><td class="">String</td></tr><tr><td class="">Last-Modified</td><td class="">对象最后修改时间</td><td class="">String</td></tr><tr><td class="">Cache-Control</td><td class="">缓存策略， HTTP 标准头部</td><td class="">String</td></tr><tr><td class="">Content-Type</td><td class="">内容类型，HTTP 标准头部</td><td class="">String</td></tr><tr><td class="">Content-Disposition</td><td class="">文件名称，HTTP 标准头部</td><td class="">String</td></tr><tr><td class="">Content-Encoding</td><td class="">编码格式，HTTP 标准头部</td><td class="">String</td></tr><tr><td class="">Content-Language</td><td class="">语言类型，HTTP 标准头部</td><td class="">String</td></tr><tr><td class="">Content-Length</td><td class="">对象大小</td><td class="">String</td></tr><tr><td class="">Expires</td><td class="">缓存过期时间， HTTP 标准头部</td><td class="">String</td></tr><tr><td class="">x-cos-meta-*</td><td class="">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</td><td class="">String</td></tr><tr><td class="">x-cos-version-id</td><td class="">开启版本控制后，对象的版本号</td><td class="">String</td></tr></tbody></table>

### 下载对象

#### 功能说明

下载一个对象到本地（GET Object）。

#### 方法原型

```python
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 |
| 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=='
}
```

<table class="doc-table-container indent-undefined"><thead><tr><td class="">参数名称</td><td class="">参数描述</td><td class="">类型</td></tr></thead><tbody><tr><td class="">Body</td><td class="">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</td><td class="">StreamBody</td></tr><tr><td class="">ETag</td><td class="">对象的 MD5 值</td><td class="">String</td></tr><tr><td class="">Last-Modified</td><td class="">对象最后修改时间</td><td class="">String</td></tr><tr><td class="">Accept-Ranges</td><td class="">范围单位， HTTP 标准头部</td><td class="">String</td></tr><tr><td class="">Content-Range</td><td class="">内容范围， HTTP 标准头部</td><td class="">String</td></tr><tr><td class="">Cache-Control</td><td class="">缓存策略， HTTP 标准头部</td><td class="">String</td></tr><tr><td class="">Content-Type</td><td class="">内容类型，HTTP 标准头部</td><td class="">String</td></tr><tr><td class="">Content-Disposition</td><td class="">文件名称，HTTP 标准头部</td><td class="">String</td></tr><tr><td class="">Content-Encoding</td><td class="">编码格式，HTTP 标准头部</td><td class="">String</td></tr><tr><td class="">Content-Language</td><td class="">语言类型，HTTP 标准头部</td><td class="">String</td></tr><tr><td class="">Content-Length</td><td class="">对象大小</td><td class="">String</td></tr><tr><td class="">Expires</td><td class="">缓存过期时间， HTTP 标准头部</td><td class="">String</td></tr><tr><td class="">x-cos-meta-*</td><td class="">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</td><td class="">String</td></tr><tr><td class="">x-cos-version-id</td><td class="">开启版本控制后，对象的版本号</td><td class="">String</td></tr></tbody></table>

### 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

```
python
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',
'VesionId': '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'
}
```<table class="doc-table-container indent-undefined"><thead><tr><td class="">参数名称</td><td class="">参数描述</td><td class="">类型</td></tr></thead><tbody><tr><td class="">ETag</td><td class="">拷贝对象的 MD5 值</td><td class="">String</td></tr><tr><td class="">LastModified</td><td class="">拷贝对象的最后一次修改时间</td><td class="">String</td></tr><tr><td class="">VersionId</td><td class="">开启版本控制后，目的对象的版本号</td><td class="">String</td></tr><tr><td class="">x-cos-copy-source-version-id</td><td class="">源对象的版本号</td><td class="">String</td></tr></tbody></table>

### 删除单个对象

#### 功能说明

删除指定的对象（DELETE Object）。

#### 方法原型

```python
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',
}
```
```<table class="doc-table-container indent-undefined"><thead><tr><td class="">参数名称</td><td class="">参数描述</td><td class="">类型</td></tr></thead><tbody><tr><td class="">x-cos-version-id</td><td class="">删除对象的版本号</td><td class="">String</td></tr><tr><td class="">x-cos-delete-marker</td><td class="">标识删除的对象是否为delete marker</td><td class="">String</td></tr></tbody></table>

### 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

```python
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'
},
]
}
```

```<table class="doc-table-container indent-undefined"><thead><tr><td class="">参数名称</td><td class="">参数描述</td><td class="">类型</td></tr></thead><tbody><tr><td class="">Deleted</td><td class="">删除成功的 Object 信息</td><td class="">List</td></tr><tr><td class="">Key</td><td class="">删除成功的 Object 的路径</td><td class="">String</td></tr><tr><td class="">VersionId</td><td class="">删除成功的 Object 的版本号</td><td class="">String</td></tr><tr><td class="">DeleteMarker</td><td class="">删除成功的 Object 是否为 delete marker</td><td class="">String</td></tr><tr><td class="">DeleteMarkerVersionId</td><td class="">删除成功的 Object 的 delete marker 的版本号</td><td class="">String</td></tr><tr><td class="">Error</td><td class="">删除失败的 Object 信息</td><td class="">List</td></tr><tr><td class="">Key</td><td class="">删除失败的 Object 的路径</td><td class="">String</td></tr><tr><td class="">VersionId</td><td class="">删除失败的 Object 的版本号</td><td class="">String</td></tr><tr><td class="">Code</td><td class="">删除失败的 Object 对应的错误码</td><td class="">String</td></tr><tr><td class="">Message</td><td class="">删除失败的 Object 对应的错误信息</td><td class="">String</td></tr></tbody></table>

## 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

```python
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 |
| 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'
},
],
}
```

<table class="doc-table-container indent-undefined"><thead><tr><td class="">参数名称</td><td class="">参数描述</td><td class="">类型</td></tr></thead><tbody><tr><td class="">Bucket</td><td class="">存储桶名称，由 BucketName-APPID 构成</td><td class="">String</td></tr><tr><td class="">Prefix</td><td class="">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</td><td class="">String</td></tr><tr><td class="">Delimiter</td><td class="">默认为空，设置分隔符</td><td class="">String</td></tr><tr><td class="">KeyMarker</td><td class="">和 UploadIdMarker 一起使用，指明列出分块上传的 key 起始位置</td><td class="">String</td></tr><tr><td class="">UploadIdMarker</td><td class="">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</td><td class="">String</td></tr><tr><td class="">NextKeyMarker</td><td class="">当 IsTruncated 为 true 时，指明下一次列出分块上传的 key 的起始位置</td><td class="">String</td></tr><tr><td class="">NextUploadIdMarker</td><td class="">当 IsTruncated 为 true 时，指明下一次列出分块上传的 uploadid 的起始位置</td><td class="">String</td></tr><tr><td class="">MaxUploads</td><td class="">最多返回的分块上传的数量，默认为最大的1000</td><td class="">Int</td></tr><tr><td class="">IsTruncated</td><td class="">表示返回的分块上传否被截断</td><td class="">String</td></tr><tr><td class="">EncodingType</td><td class="">默认不编码，规定返回值的编码方式，可选值：url</td><td class="">String</td></tr><tr><td class="">Upload</td><td class="">包含所有分块上传的 list，包括 'UploadId'，'StorageClass'，'Key'，'Owner'，'Initiator'，'Initiated' 等信息</td><td class="">List</td></tr><tr><td class="">CommonPrefixes</td><td class="">所有以 Prefix 开头，以 Delimiter 结尾的 Key 被归到同一类</td><td class="">List</td></tr></tbody></table>

### 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

```python
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'
}
```

```<table class="doc-table-container indent-undefined"><thead><tr><td class="">参数名称</td><td class="">参数描述</td><td class="">类型</td></tr></thead><tbody><tr><td class="">UploadId</td><td class="">标识分块上传的 ID</td><td class="">String</td></tr><tr><td class="">Bucket</td><td class="">存储桶名称，由 BucketName-APPID 组成</td><td class="">String</td></tr><tr><td class="">Key</td><td class="">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</td><td class="">String</td></tr></tbody></table>

### 上传分块

分块上传对象（Upload Part）。

#### 方法原型

```python
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'
}
```<table class="doc-table-container indent-undefined"><thead><tr><td class="">参数名称</td><td class="">参数描述</td><td class="">类型</td></tr></thead><tbody><tr><td class="">ETag</td><td class="">上传分块的 MD5 值。</td><td class="">String</td></tr></tbody></table>

### 复制分块

将其他对象复制为一个分块（Upload Part - Copy）。

#### 方法原型

```python
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',
}
```
```<table class="doc-table-container indent-undefined"><thead><tr><td class="">参数名称</td><td class="">参数描述</td><td class="">类型</td></tr></thead><tbody><tr><td class="">ETag</td><td class="">拷贝分块的 MD5 值</td><td class="">String</td></tr><tr><td class="">LastModified</td><td class="">拷贝分块的最后一次修改时间</td><td class="">String</td></tr><tr><td class="">x-cos-copy-source-version-id</td><td class="">源对象的版本号</td><td class="">String</td></tr></tbody></table>

### 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

``` python
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'
}
}
```<table class="doc-table-container indent-undefined"><thead><tr><td class="">参数名称</td><td class="">参数描述</td><td class="">类型</td></tr></thead><tbody><tr><td class="">Bucket</td><td class="">存储桶名称，由 BucketName-APPID 构成</td><td class="">String</td></tr><tr><td class="">Key</td><td class="">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</td><td class="">String</td></tr><tr><td class="">UploadId</td><td class="">标识分块上传的 ID</td><td class="">String</td></tr><tr><td class="">EncodingType</td><td class="">默认不编码，规定返回值的编码方式，可选值：url</td><td class="">String</td></tr><tr><td class="">MaxParts</td><td class="">最多返回的分块的数量，默认为最大的1000</td><td class="">String</td></tr><tr><td class="">IsTruncated</td><td class="">表示返回的分块是否被截断</td><td class="">String</td></tr><tr><td class="">PartNumberMarker</td><td class="">默认为0，从第一块列出分块，从 PartNumberMarker 下一个分块开始列出</td><td class="">String</td></tr><tr><td class="">NextPartNumberMarker</td><td class="">指明下一次列出分块的起始位置</td><td class="">String</td></tr><tr><td class="">StorageClass</td><td class="">对象的存储类型，默认值 STANDARD</td><td class="">String</td></tr><tr><td class="">Part</td><td class="">上传分块的相关信息，包括 ETag，PartNumber，Size，LastModified</td><td class="">String</td></tr><tr><td class="">Initiator</td><td class="">分块上传的创建者，包括 DisplayName 和 ID</td><td class="">Dict</td></tr><tr><td class="">Owner</td><td class="">对象拥有者的信息，包括 DisplayName 和 ID</td><td class="">Dict</td></tr></tbody></table>

### 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

```python
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'
}
```

```<table class="doc-table-container indent-undefined"><thead><tr><td class="">参数名称</td><td class="">参数描述</td><td class="">类型</td></tr></thead><tbody><tr><td class="">ETag</td><td class="">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。</td><td class="">String</td></tr><tr><td class="">Bucket</td><td class="">存储桶名称，由 BucketName-APPID 构成</td><td class="">String</td></tr><tr><td class="">Location</td><td class="">URL 地址</td><td class="">String</td></tr><tr><td class="">Key</td><td class="">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</td><td class="">String</td></tr></tbody></table>

### 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

```python
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 |
| --- | --- | --- | --- |
| 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'
},
]
}
```

<table class="doc-table-container indent-undefined"><thead><tr><td class="">参数名称</td><td class="">参数描述</td><td class="">类型</td></tr></thead><tbody><tr><td class="">Owner</td><td class="">对象拥有者的信息，包括 DisplayName 和 ID</td><td class="">Dict</td></tr><tr><td class="">Grant</td><td class="">对象权限授予者的信息，包括 Grantee 和 Permission</td><td class="">List</td></tr><tr><td class="">Grantee</td><td class="">对象权限授予者的信息，包括 DisplayName，Type，ID 和 URI</td><td class="">Dict</td></tr><tr><td class="">DisplayName</td><td class="">权限授予者的名字</td><td class="">String</td></tr><tr><td class="">Type</td><td class="">权限授予者的类型，类型为 CanonicalUser 或者 Group</td><td class="">String</td></tr><tr><td class="">ID</td><td class="">Type 为 CanonicalUser 时，对应权限授予者的 ID</td><td class="">String</td></tr><tr><td class="">URI</td><td class="">Type 为 Group 时，对应权限授予者的 URI</td><td class="">String</td></tr><tr><td class="">Permission</td><td class="">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</td><td class="">String</td></tr></tbody></table>

## 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

``` python
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'
}
```
```<table class="doc-table-container indent-undefined"><thead><tr><td class="">参数名称</td><td class="">参数描述</td><td class="">类型</td></tr></thead><tbody><tr><td class="">ETag</td><td class="">上传对象的 MD5 值</td><td class="">String</td></tr></tbody></table>

# 存储桶管理

## 简介

本文档提供关于跨域访问、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**
<table class="doc-table-container indent-undefined"><thead><tr><td class="">API</td><td class="">操作名</td><td class="">操作描述</td></tr></thead><tbody><tr><td class="">PUT Bucket cors</td><td class="">设置跨域配置</td><td class="">设置存储桶的跨域访问权限</td></tr><tr><td class="">GET Bucket cors</td><td class="">查询跨域配置</td><td class="">查询存储桶的跨域访问配置信息</td></tr><tr><td class="">DELETE Bucket cors</td><td class="">删除跨域配置</td><td class="">删除存储桶的跨域访问配置信息</td></tr></tbody></table>

**版本控制**
<table class="doc-table-container indent-undefined"><thead><tr><td class="">API</td><td class="">操作名</td><td class="">操作描述</td></tr></thead><tbody><tr><td class="">PUT Bucket versioning</td><td class="">设置版本控制</td><td class="">设置存储桶的版本控制功能</td></tr><tr><td class="">GET Bucket versioning</td><td class="">查询版本控制</td><td class="">查询存储桶的版本控制信息</td></tr></tbody></table>

**跨地域复制**
<table class="doc-table-container indent-undefined"><thead><tr><td class="">API</td><td class="">操作名</td><td class="">操作描述</td></tr></thead><tbody><tr><td class="">PUT Bucket replication</td><td class="">设置跨地域复制</td><td class="">设置存储桶的跨地域复制规则</td></tr><tr><td class="">GET Bucket replication</td><td class="">查询跨地域复制</td><td class="">查询存储桶的跨地域复制规则</td></tr><tr><td class="">DELETE Bucket replication</td><td class="">删除跨地域复制</td><td class="">删除存储桶的跨地域复制规则</td></tr></tbody></table>

## 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

```python
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://gsesgpucloud.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',
],
}
]
}
```

```<table class="doc-table-container indent-undefined"><thead><tr><td class="">参数名称</td><td class="">参数描述</td><td class="">类型</td></tr></thead><tbody><tr><td class="">CORSRule</td><td class="">跨域规则，包括 ID，MaxAgeSeconds，AllowedOrigin，AllowedMethod，AllowedHeader，ExposeHeader</td><td class="">List</td></tr><tr><td class="">ID</td><td class="">规则的 ID</td><td class="">String</td></tr><tr><td class="">MaxAgeSeconds</td><td class="">OPTIONS 请求得到结果的有效期</td><td class="">String</td></tr><tr><td class="">AllowedOrigin</td><td class="">允许的访问来源，如 `"http://gsesgpucloud.com"`，支持通配符`*`</td><td class="">Dict</td></tr><tr><td class="">AllowedMethod</td><td class="">允许的方法，如 GET，PUT，HEAD，POST，DELETE</td><td class="">Dict</td></tr><tr><td class="">AllowedHeader</td><td class="">请求可以使用哪些自定义的 HTTP 请求头部，支持通配符`*`</td><td class="">Dict</td></tr><tr><td class="">ExposeHeader</td><td class="">浏览器可以接收到的来自服务器端的自定义头部信息</td><td class="">Dict</td></tr></tbody></table>

### 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

```python
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。

# 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

CSP XML Python SDK 操作成功会返回一个 dict 或者 None。若调用 SDK 接口请求 CSP 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 CSP 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 CSP 服务端交互正常，但操作 CSP 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，更多详情请参见 CSP [错误码] | string |
| error_code | 请求失败时 body 返回的 Error Code，更多详情请参见 CSP [错误码] | string |
| error_msg | 请求失败时 body 返回的 Error Message，更多详情请参见 CSP [错误码] | 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 CSP 定义的错误码
e.get_error_msg() # 获取 CSP 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
```

# CSP API参考

## 简介

最近更新时间: 2024-12-19 17:12:00

### 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（CSP）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 CSP ，而必须要了解的主要概念和术语。

| 名称 | 描述 |
| --- | --- |
| AppID | 开发者访问 CSP 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 CSP 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 CSP 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。

# 访问策略语言概述

最近更新时间: 2024-12-19 17:12:00

## 概述

访问策略可用于授予访问 CSP 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 CSP 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

CSP 定义了可在策略中指定的某一个特定的 CSP 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 CSP 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：

1. qcs 是 qcloud service 的简称。
2. project_id 描述项目信息，仅为了兼容 CAM 早期逻辑。这里请不填。
3. service_type 描述产品简称，如 CSP。
4. region 描述地域信息。
5. 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 CSP 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。
6. resource 描述具体资源详情，在 CSP 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或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/*"
]
}
]
}
```

# 公共请求头部

最近更新时间: 2024-12-19 17:12:00

## 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。<br>有效值：100-continue。 | String | 否 |
| Host | 请求的主机，形式为 -.cos..myqcloud.com。 | String | 是 |

## 服务端加密专用头部

对于支持服务端加密（Server Side Encryption，SSE）的接口，根据不同的加密方式适用如下请求头部，请参阅具体的接口文档确定是否适用 SSE。下列头部的是否必选仅针对使用 SSE 的场景，如果请求不支持 SSE 的接口或不使用 SSE ，则以下头部均无需携带。详情请参见 服务端加密概述。

**SSE-COS**

| Header 名称 | 描述 | 类型 | 是否必选 |
| --- | --- | --- | --- |
| x-cos-server-side-encryption | 服务端加密算法，使用 SSE-COS 时根据所选加密算法可指定为 AES256 或 SM4 | string | 上传或复制对象（包括简单上传/复制与分块上传/复制）时必选，下载对象时不能指定此头部 |

**SSE-KMS**

| Header 名称 | 描述 | 类型 | 是否必选 |
| --- | --- | --- | --- |
| x-cos-server-side-encryption | 服务端加密算法，使用 SSE-KMS 时根据所选加密算法可指定为 cos/kms 或 cos/kms/sm4 | string | 上传或复制对象（包括简单上传/复制与分块上传/复制）时必选，下载对象时不能指定此头部 |
| x-cos-server-side-encryption-cos-kms-key-id | 当 x-cos-server-side-encryption 值为 cos/kms 或 cos/kms/sm4 时，用于指定 KMS 的用户主密钥 CMK，如不指定，则使用 CSP 默认创建的 CMK，更多详细信息可参见 SSE-KMS 加密 | string | 否 |

**SSE-C**

| Header 名称 | 描述 | 类型 | 是否必选 |
| --- | --- | --- | --- |
| x-cos-server-side-encryption-customer-algorithm | 服务端加密算法，目前仅支持 AES256 | string | 是 |

| Header 名称 | 描述 | 类型 | 是否必选 |
| --- | --- | --- | --- |
| x-cos-server-side-encryption-customer-key | 服务端加密密钥的 Base64 编码，例如 `MDEyMzQ1Njc4OUFCQ0RFRjAxMjM0NTY3ODlBQkNERUY=` | string | 是 |
| x-cos-server-side-encryption-customer-key-MD5 | 服务端加密密钥的 MD5 哈希值，使用 Base64 编码，例如`U5L61r7jcwdNvT7frmUG8g==` | string | 是 |

# 公共响应头部

最近更新时间: 2024-12-19 17:12:00

## 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。<br>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。<br>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: 2024-12-19 17:12:00

## 概述

此文将为您介绍请求出错时返回的错误码和对应错误信息。

## 错误信息返回格式

### 返回头部

Content-Type：application/xml。

对应HTTP状态码：3XX，4XX，5XX。

### 返回内容

**语法格式**

```
<?xml version="1.0" encoding="UTF-8"?>
<Error>
<Code>[错误码]</Code>
<Message>[错误信息]</Message>
<Resource>[资源地址]</Resource>
<RequestId>[请求ID]</RequestId>
<TraceId>[错误ID]</TraceId>
</Error>
```

**元素说明**

| 元素名称 | 描述 | 类型 |
| --- | --- | --- |
| Error | 包含所有的错误信息。 | Container |
| Code | 错误码用来定位唯一的错误条件，用来确定错误场景，具体错误码见下文。 | String |
| Message | 包含具体的错误信息。 | String |
| Resource | 资源地址：Bucket地址或者Object地址。 | String |
| RequestId | 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 CSP 定位问题。 | String |
| TraceId | 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。使用遇到问题时，trace-id能更快地协助 CSP 定位问题。当请求出错时，trace-id与request-id一一对应。 | 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 |
| --- | --- | --- |
| BadDigest | 提供的x-cos-SHA-1值与服务端收到的文件SHA-1值不符合。 | 400 Bad Request |

| 错误码 | 描述 | HTTP状态码 |
| --- | --- | --- |
| EntityTooSmall | 上传的文件大小 不足要求的最小值，常见于分片上传。 | 400 Bad Request |
| EntityTooLarge | 上传的文件大小超过要求的最大值。 | 400 Bad Request |
| ImcompleteBody | 请求的实际内容长度和指定的Content-Length不符。 | 400 Bad Request |
| IncorrectNumberOfFilesInPostRequest | Post请求每次只允许上传一个文件。 | 400 Bad Request |
| InlineDataTooLarge | 内链数据大小高于要求的最大值。 | 400 Bad Request |
| InvalidArgument | 请求参数不合法。 | 400 Bad Request |
| InvalidBucketName | Bucket名称不合法。 | 400 Bad Request |
| InvalidDigest | x-cos-SHA-1值不合法。 | 400 Bad Request |
| InvalidPart | 分片缺失或者SectionID出错。 | 400 Bad Request |
| InvalidPolicyDocument | 策略配置文件不合法。 | 400 Bad Request |
| InvalidURI | URI不合法。 | 400 Bad Request |
| KeyTooLong | 自定义头部过长。 | 400 Bad Request |
| MalformedACLError | 描述的ACL策略不符合XML语法。 | 400 Bad Request |
| MalformedPOSTRequest | 该POST请求的Body内容不合法。 | 400 Bad Request |
| MalformedXML | body的XML格式不符合XML语法。 | 400 Bad Request |
| MaxMessageLengthExceeded | 请求过长。 | 400 Bad Request |
| MaxPostPreDataLengthExceededError | 该POST请求的数据前缀过长，常见于分片上传。 | 400 Bad Request |
| MetadataTooLarge | 元数据大小超过要求的最大值。 | 400 Bad Request |
| MissingRequestBodyError | 请求Body缺失。 | 400 Bad Request |
| MissingSecurityHeader | 必要Header缺失。 | 400 Bad Request |
| MissingContentMD5 | 请求头中缺少Content-MD5。 | 400 Bad Request |
| MissingAppid | 请求头中缺少Appid。 | 400 Bad Request |
| MissingHost | 请求头中缺少Host。 | 400 Bad Request |
| RequestIsNotMultiPartContent | Post请求 Content-Type不合法。 | 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 |
| TooManyBucket | bucket数量超过200限制 。 | 400 Bad Request |
| UnexpectedContent | 请求不支持相关内容。 | 400 Bad Request |
| UnresolvableGrantByUID | 提供的UID不存在。 | 400 Bad Request |
| UserKeyMustBeSpecified | 针对Bucket的Post操作必须指定明确路径。 | 400 Bad Request |
| AccessDenied | 签名或者权限不正确，拒绝访问。 | 403 Forbidden |
| AccountProblem | 您的账号拒绝了此次操作。 | 403 Forbidden |
| InvalidAccessKeyId | AccessKey不存在。 | 403 Forbidden |
| InvalidObjectState | 请求内容与Object属性相冲突。 | 403 Forbidden |
| InvalidSecurity | 签名串不合法。 | 403 Forbidden |
| RequestTimeTooSkewed | 请求时间超过权限有效时间。 | 403 Forbidden |
| SignatureDoesNotMatch | 提供的签名不符合规则。 | 403 Forbidden |

版权所有：亿算云平台

| 错误码 | 描述 | HTTP状态码 |
| --- | --- | --- |
| NoSuchBucket | 指定的Bucket不存在。 | 404 Not Found |
| NoSuchUpload | 指定的分片上传不存在。 | 404 Not Found |
| NoSuchBucket | 指定的Bucket策略不存在。 | 404 Not Found |
| MethodMotAllowed | 此资源不支持该HTTP方法。 | 405 Method Not Allowed |
| BucketAlreadyExists | CreateBucket指定的BucketName已经使用，请选择新的BucketName。 | 409 Conflict |
| BucketNotEmpty | DeleteBucket前请先删除文件和未完成的分片上传任务。 | 409 Conflict |
| InvalidBucketState | bucket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 |
| OperationAborted | 指定资源不支持此类操作。 | 409 Conflict |
| MissingContentLength | Header Content-Length缺失。 | 411 Length Required |
| PreconditionFailed | 前置条件匹配失败。 | 412 Precondition |
| InvalidRange | 请求的文件范围不合法。 | 416 Requested Range Not Satisfiable |
| InvalidSHA1Digest | 请求内容sha1校验不合法。 | 400 Bad Request |
| NoSuchUpload | 分块上传时指定的uploadid不存在。 | 400 Bad Request |
| InvalidPart | 分块缺失。 | 400 Bad Request |
| InvalidPartOrder | 分块上传编号不连续。 | 400 Bad Request |
| ObjectNotAppendable | 指定的文件不能追加。 | 400 Bad Request |
| AppendPositionErr | Append:文件长度和position不一致。 | 400 Bad Request |
| NoSuchVersion | 指定版本不存在。 | 400 Bad Request |
| NoLifecycle | 生命周期不存在。 | 400 Bad Request |
| PreconditionFailed | 前置条件匹配失败。 | 400 Bad Request |
| UnexpectedContent | 请求不支持相关内容。 | 400 Bad Request |
| MultiBucketNotSupport | 跨区域复制只能设一个目的bucket。 | 400 Bad Request |
| NotSupportedStorageClass | 指定的存储类型不合法。 | 400 Bad Request |
| InvalidAccessKeyId | AccessKey不存在。 | 403 Forbidden |
| ExpiredToken | 签名串已过期。 | 403 Forbidden |

**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 | Header中存在无法处理的方法。 | 501 Not Implemented |
| ServiceUnavailable | 服务器内部错误，请重试。 | 503 Service Unavailable |
| SlowDown | 请降低访问频率。 | 503 Slow Down |

**其他类型错误**

| 错误码 | 描述 | HTTP状态码 |
| --- | --- | --- |
| InvalidAddressingHeader | 必须使用匿名角色访问。 | N/A |

# 请求签名

最近更新时间: 2024-12-19 17:12:00

> **注意：**
>
> 1. 此文档仅适用于 CSP XML 版本，版本可登录后在 CSP 控制台首页查看。
> 2. 此文档不适用于 POST object 的 HTTP 请求。

使用对象存储服务 CSP 时，可通过 RESTful API 对 CSP 发起 HTTP 匿名请求或 HTTP 签名请求，对于签名请求，CSP 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，CSP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。

云平台对象存储，基于密钥 HMAC (Hash Message Authentication Code) 的自定义 HTTP 方案进行身份验证。

## 签名使用场景

在 CSP 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。

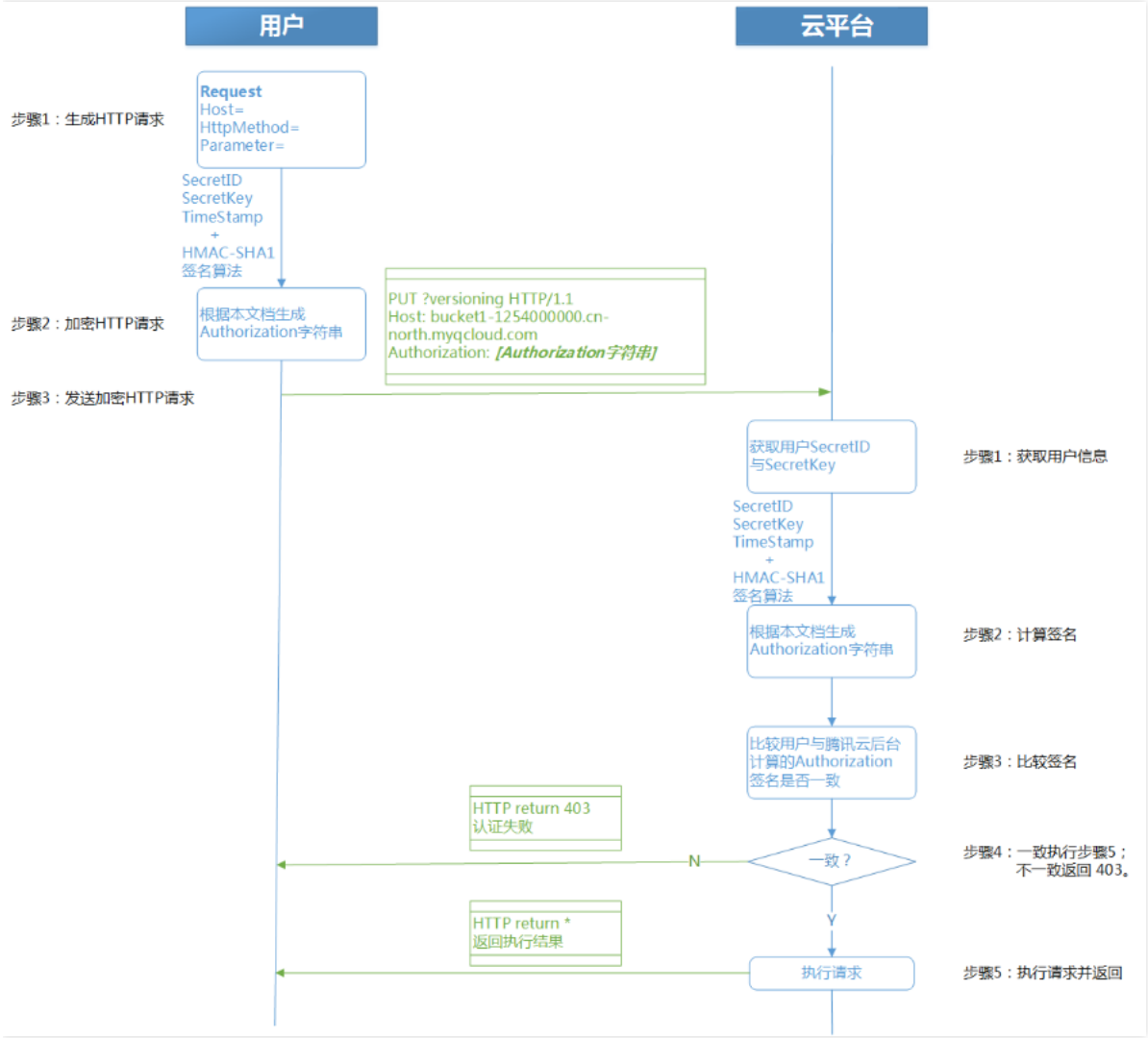
> **注意：**
>
> 本文所描述的 API 请求签名，如果您使用 SDK 进行开发，则已包含在内。**仅在您希望通过原始 API 进行二次开发时，需要根据本文所描述步骤进行操作。**

在以下场景中，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。**对数据加密并检验，保障传输内容完整性。
3. **防止签名被盗用。**对签名设置时效，且进行数据加密，避免签名盗用并重复使用。

## 签名流程

客户通过对 HTTP 请求进行签名，并将签名后的请求发送至云平台进行签名验证，具体流程如下图所示。




## 准备工作

1. APPID、SecretId 和 SecretKey。

   在控制台**云API密钥**页面可获取。

2. 确定开发语言：

   支持但不限于 java、php 、c sharp、c++、 node.js、python，根据不同的开发语言，确定对应的 HMAC-SHA1、SHA1 函数。

## 签名内容

通过 RESTful API 对 CSP 发起的 HTTP 签名请求，使用标准的 HTTP Authorization 头部来传递，如下例所示：

```
PUT ?versioning HTTP/1.1
Host: bucket1-1254000000.cos.ap-beijing.myqcloud.com
Authorization: q-sign-algorithm=sha1&q-ak=AKIDxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx&q-sign-time=1480932292;1481012298&q-key-time=1480932292;1481012298&q-header-list=host&q-url-param-list=versioning&q-signature=438023e4a4207299d87bb75d1c739c06cc9406bb
```

其中，签名内容由多个 key=value 对，通过 “&” 连接而成，格式如下：

```
q-sign-algorithm=sha1&q-ak=[SecretID]&q-sign-time=[SignTime]&
q-key-time=[KeyTime]&q-header-list=[SignedHeaderList]&
q-url-param-list=[SignedParameterList]&q-signature=[Signature]
```

**键值描述**

其中，组成签名内容的键值(key=value)描述如下：

| 键（key） | 值(value) | 描述 |
| --- | --- | --- |
| q-sign-algorithm | sha1 | 必填。<br>签名算法，目前仅支持 sha1，即为 sha1 。 |
| q-ak | 参数[*SecretID*] | 必填。<br>帐户 ID，即 SecretId，在控制台**云 API 密钥**页面可获取。<br>**如：** AKIDxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx 。 |
| q-sign-time | 参数[*SignTime*] | 必填。<br>本签名的有效起止时间，通过 Unix 时间戳[注1]<br>描述起始和结束时间，以秒为单位，格式为 [*start-seconds*];[*end-seconds*]。<br>**如：** 1480932292;1481012298 。 |
| q-key-time | 参数[*KeyTime*] | 必填。<br>与 q-sign-time 值相同。 |
| q-header-list | 参数[*SignedHeaderList*] | 必填。<br>HTTP 请求头部。需从 key:value 中提取部分或全部 key，且 key 需转化为小写，并将多个 key 之间以字典顺序排序，如有多组 key，可用";"连接。<br>**如：** HTTP 请求 " Host: bucket1-1254000000.cos.ap-beijing.myqcloud.com Content-Type: image/jpeg "，其 SignedHeaderList 为 content-type;host 。 |
| q-url-param-list | 参数[*SignedParameterList*] | 必填。<br>HTTP 请求参数。需从 key=value 中提取部分或全部 key，且 key 需转化为小写，并将多个 key 之间以字典顺序排序，如有多组 key，可用";"连接。<br>**如：** HTTP 请求 " GET /?prefix=abc&max-keys=20 "，其则 SignedParameterList 为 max-keys;prefix 或者 prefix 。 |
| q-signature | 参数[Signature] | 必填。<br>HTTP内容签名，请查看 Signature 计算 。 |

> **说明：**
>
> 注1：**关于q-sign-time 和 q-key-time**
>
> 1. Unix 时间戳是从格林威治时间1970 年 01 月 01 日 00 时 00 分 00 秒(北京时间 1970 年 01 月 01 日 08 时 00 分 00 秒)起至现在的总秒数。
> 2. 结束时间戳需要大于起始时间戳，否则将会导致签名马上过期。

### Signature 计算

Signature 的计算分为四个步骤：

1. 对临时密钥的有效起止时间加密计算值 SignKey。
2. 根据固定格式组合生成 HttpString。
3. 加密 HttpString，并根据固定格式组合生成 StringToSign。
4. 加密 StringToSign，生成Signature。

**代码说明**

其伪代码为：

```
SignKey = HMAC-SHA1(SecretKey,"[q-key-time]")
HttpString = [HttpMethod]\n[HttpURI]\n[HttpParameters]\n[HttpHeaders]\n
StringToSign = [q-sign-algorithm]\n[q-sign-time]\nSHA1-HASH(HttpString)\n
Signature = HMAC-SHA1(SignKey,StringToSign)
```

其中，在不同的开发语言环境，请用不同的语言规范更新代码，包含：

1. 变量的定义与赋值，请根据所使用开发语言的规范更新。
2. 伪函数 SHA1_FUNC，输出为 16 进制小写。请替换为所使用开发语言中对应的函数，如下表所示：

| 伪函数 | php | java |
| --- | --- | --- |
| HMAC-SHA1 | hash_hmac | HmacUtils.hmacSha1Hex |
| SHA1-HASH | sha1 | DigestUtils.sha1Hex |

**代码实例（PHP）**

如 php 开发环境中，以上代码为：

```
$SignKey = hash_hmac($SecretKey,"[q-key-time]")
$HttpString = [HttpMethod]\n[HttpURI]\n[HttpParameters]\n[HttpHeaders]\n
$StringToSign = [q-sign-algorithm]\n[q-sign-time]\nsha1($HttpString)\n
$Signature = hash_hmac($SignKey,$StringToSign)
```

**参数说明**

| 参数 | 说明 |
| --- | --- |
| [q-key-time] | 必须与键值描述中所填写的 q-key-time 保持一致。 |
| [HttpMethod] | HTTP请求方法。仅支持小写，即可为 get，post，put，delete，head，options。<br>**如：** HTTP 请求 “ GET /testfile ”，其 HttpMethod 为 get。 |
| [HttpURI] | HTTP 请求 URI 部分。即从“/”虚拟根路径开始部分。<br>**如：** HTTP请求 “GET /testfile ”，其 HttpURI 为 /testfile 。 |
| [HttpParameters] | HTTP 请求参数。即为 URI 中“?”之后由“&”连接的部分，需选取全部或部分 key=value，且 key 和value 均需转换为小写，多对 key=value 对之间以 “&” 相连接，并以字典顺序排序。<br>**如：** HTTP 请求 “ GET /?prefix=abc&max-keys=20 ”，其 HttpParameters 为 max-keys=20&prefix=abc 或 prefix=abc 。<br>**注意：**<br>所包含的 key=value 对中的 key 必须与键值描述中所填写的 q-url-param-list 中的 key 保持一致。 |
| [HttpHeaders] | HTTP 请求头部。需选取全部或部分 key:value，并将其转化为 key=value 格式，且 key 需转换为小写，value 需进行 URLEncode 转换，多对 key=value 对之间以 “&” 相连接，并以字典顺序排序。<br>**如：** HTTP请求 “ Host: bucket1-1254000000.cos.ap-beijing.myqcloud.com Content-Type: image/jpeg ”，其 HttpHeaders 为 content-type=image%2Fjpeg&host=bucket1-1254000000.cos.ap-beijing.myqcloud.com 。<br>**注意：**<br>所包含的 key=value 对中的 key 必须与键值描述中所填写的 q-header-list 中的 key 保持一致。 |
| [q-sign-algorithm] | sha1。目前仅支持 sha1 加密算法。 |
| [q-sign-time] | 必须与键值描述中所填写的 q-sign-time 保持一致。 |

**参数实例**

| 参数 | 值 |
| --- | --- |
| [q-key-time] | 1417773892;1417853898 |
| [HttpMethod] | get |
| [HttpURI] | /testfile |
| [HttpParameters] | max-keys=20&prefix=abc |
| [HttpHeaders] | host=bucket1-1254000000.cos.ap-beijing.myqcloud.com |
| [q-sign-algorithm] | sha1 |
| [q-sign-time] | 1417773892;1417853898 |

## 举例说明

如某用户希望使用 API 调用方式下载和上传对象，并对调用进行签名。

**准备工作**

通过登录**云 API 密钥**页面获取到其 APPID、SecretId 和 SecretKey，并确定开发语言，分别如下：

| APPID | SecretId | SecretKey | 开发语言 |
| --- | --- | --- | --- |
| 1254000000 | AKIDxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx | BQYIM75p8x0iWVFSIgqEKwFprpRSVHlz | php |

### 上传对象

需求：上传对象到存储桶 bucket1。

原始HTTP请求为：

```
PUT /testfile2 HTTP/1.1
Host: bucket1-1254000000.cos.ap-beijing.myqcloud.com
x-cos-content-sha1: 7b502c3a1f48c8609ae212cdfb639dee39673f5e

Hello world
```

签名后的HTTP请求为：

```
PUT /testfile2 HTTP/1.1
Host: bucket1-1254000000.cos.ap-beijing.myqcloud.com
x-cos-content-sha1: 7b502c3a1f48c8609ae212cdfb639dee39673f5e
Authorization: q-sign-algorithm=sha1&q-ak=AKIDxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx&q-sign-time=1417773892;1417853898&q-key-time=1417773892;14
17853898&q-header-list=host;x-cos-content-sha1;x-cos-storage-class&q-url-param-list=&q-signature=84f5be2187452d2fe276dbdca932143ef8161145

Hello world
```

各参数对应的值和描述如下：

| 键（key） | 值(value) | 备注 |
| --- | --- | --- |
| q-sign-algorithm | sha1 | 目前仅支持 sha1 签名算法。 |
| q-ak | AKIDxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx | SecretId 字段 |
| q-sign-time | 1417773892;1417853898 | 2014/12/5 18:04:52 到 2014/12/6 16:18:18 |
| q-key-time | 1417773892;1417853898 | 2014/12/5 18:04:52 到 2014/12/6 16:18:18 |
| q-header-list | host;x-cos-content-sha1;x-cos-storage-class | HTTP 头部 key 的字典顺序排序列表 |
| q-url-param-list | | HTTP 参数列表为空 |
| q-signature | 84f5be2187452d2fe276dbdca932143ef8161145 | 通过代码计算所得 |

其中，q-signature 的计算代码为：

```
$signTime = '1417773892;1417853898';
$signKey = hash_hmac('sha1', $signTime, 'BQYIM75p8x0iWVFSIgqEKwFprpRSVHlz');
$httpString = "put\n/testfile2\n\nhost=bucket1-1254000000.cos.ap-beijing.myqcloud.com&x-cos-content-sha1=7b502c3a1f48c8609ae212cdfb639dee39
673f5e&x-cos-storage-class=nearline\n";
$sha1edHttpString = sha1($httpString);
$stringToSign = "sha1\n$signTime\n$sha1edHttpString\n";
$signature = hash_hmac('sha1', $stringToSign, $signKey);
```

### 下载对象

需求：下载存储桶 bucket1 中的对象前 4 个字节。

原始HTTP请求为：

```
GET /testfile HTTP/1.1
Host: bucket1-1254000000.cos.ap-beijing.myqcloud.com
Range: bytes=0-3
```

签名后的 HTTP 请求为：

```
GET /testfile HTTP/1.1
Host: bucket1-1254000000.cos.ap-beijing.myqcloud.com
Range: bytes=0-3
Authorization: q-sign-algorithm=sha1&q-ak=AKIDxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx&q-sign-time=1417773892;1417853898&q-key-time=1417773892;1417853898&q-header-list=host;range&q-url-param-list=&q-signature=4b6cbab14ce01381c29032423481ebffd514e8be
```

各参数对应的值和描述如下：

| 键（key） | 值(value) | 备注 |
| --- | --- | --- |
| q-sign-algorithm | sha1 | 目前仅支持 sha1 签名算法。 |
| q-ak | AKIDxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx | SecretId 字段 |
| q-sign-time | 1417773892;1417853898 | 2014/12/5 18:04:52 到 2014/12/6 16:18:18 |
| q-key-time | 1417773892;1417853898 | 2014/12/5 18:04:52 到 2014/12/6 16:18:18 |
| q-header-list | host;range | HTTP 头部 key 的列表 |
| q-url-param-list | | HTTP 参数列表为空 |
| q-signature | 4b6cbab14ce01381c29032423481ebffd514e8be | 通过代码计算所得 |

其中，q-signature的计算代码为：

```
$signTime = '1417773892;1417853898';
$signKey = hash_hmac('sha1', $signTime, 'BQYIM75p8x0iWVFSIgqEKwFprpRSVHlz');
$httpString = "get\n/testfile\n\nhost=bucket1-1254000000.cos.ap-beijing.myqcloud.com&range=bytes%3D0-3\n";
$sha1edHttpString = sha1($httpString);
$stringToSign = "sha1\n$signTime\n$sha1edHttpString\n";
$signature = hash_hmac('sha1', $stringToSign, $signKey);
```

# 操作列表

最近更新时间: 2024-12-19 17:12:00

对象存储服务（CSP）相关接口及说明如下：

## 访问策略语言概述

访问策略语言概述

## 请求签名

请求签名

## Service 接口

| 操作名 | API | 操作描述 |
| --- | --- | --- |
| 获取存储桶列表 | GET Service | 获取指定账号下所有 Bucket List |

## Bucket 接口

**基本操作接口**

| 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 |
| 获取对象列表 | 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 |

**访问控制（acl）接口**

| 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 |
| 获取存储桶ACL | GET Bucket acl | 查询存储桶的访问控制列表 |

**跨域资源共享（cors）接口**

| 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 |
| 获取跨域配置 | 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）接口**

| 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 |

版权所有：亿算云平台

**存储桶策略（policy）接口**

| 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）接口**

| 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）接口**

| 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）接口**

| 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 |

**存储桶加密**

| 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 |
| 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 |

**存储桶事件通知**

| 操作名 | API | 操作描述 |
| --- | --- | --- |
| 设置存储桶事件通知 | PUT Bucket notification | 为 Bucket 创建一个新的事件通知配置。如果该 Bucket 已配置事件通知，使用该接口创建新的配置的同时则会覆盖原有的配置 |
| 获取存储桶事件通知 | GET Bucket notification | 查询指定存储桶下的事件通知配置 |

## Object 接口

**基本操作接口**

| 操作名 | API | 操作描述 |
| --- | --- | --- |
| 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 |
| 获取对象 | GET Object | 下载一个对象至本地 |
| 获取对象元数据 | 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 |

**访问控制接口**

| 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 |
| 获取对象ACL | GET Object acl | 查询对象的访问控制列表 |

**分块上传接口**

| 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 |
| 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 |

**标签**

| 操作名 | API | 操作描述 |
| --- | --- | --- |
| 设置对象标签 | PUT Object tagging | 为对象添加键值对作为对象标签，可以协助您分组管理已有的对象资源 |
| 查询对象标签 | GET Object tagging | 查询指定对象下已有的对象标签 |
| 删除对象标签 | DELETE Object tagging | 删除指定对象下已有的对象标签 |

版权所有：亿算云平台

# Service接口
# 获取存储桶列表

最近更新时间: 2024-12-19 17:12:00

## 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: cos.<Region>.myqcloud.com
Date: GMT Date
Authorization: Auth String
```

> **说明：**
>
> 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 |
| --- | --- | --- | --- | --- |
| 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: cos.ap-beijing.myqcloud.com
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
# 基本操作接口
# 创建存储桶

最近更新时间: 2024-12-19 17:12:00

## 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

1. 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

### 请求示例

```
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。<br>有效值：private，public-read-write，public-read；<br>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=" "；<br>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=" "；<br>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=" "；<br>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 CSP 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: arlenhuangtestsgnoversion-1251668577.cos.ap-beijing.myqcloud.com
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
```

# 获取对象列表

最近更新时间: 2024-12-19 17:12:00

## 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: examplebucket-1250000000.cos.ap-beijing.myqcloud.com
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;3
2557109451&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: 2024-12-19 17:12:00

## 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: examplebucket-1250000000.cos.ap-beijing.myqcloud.com
```

```
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: 2024-12-19 17:12:00

## 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

### 请求示例

```
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 | HTTP状态码 | 描述 |
| --- | --- | --- |
| 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 | HTTP状态码 | 描述 |
| --- | --- | --- |
| 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 CSP 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: arlenhuangtestsgnoversion-1251668577.cos.ap-beijing.myqcloud.com
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
```

# 访问控制（acl）接口
# 设置存储桶ACL

最近更新时间: 2024-12-19 17:12:00

## 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。

> **注意：**
>
> - 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。<br>有效值： private，public-read-write，public-read ；<br>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，<br>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，<br>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，<br>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 | HTTP 状态码 | 描述 |
| --- | --- | --- |
| InvalidDigest | 400 Bad Request | 用户带的 Content-MD5 和 CSP 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: examplebucket-1250000000.cos.ap-beijing.myqcloud.com
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>
</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
```

# 获取存储桶ACL

最近更新时间: 2024-12-19 17:12:00

## 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://www.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，<br>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: examplebucket-1250000000.cos.ap-beijing.myqcloud.com
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://www.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）接口
# 设置跨域配置

最近更新时间: 2024-12-19 17:12:00

## 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 | 否 |
| 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://www.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: arlenhuangtestsgnoversion-1251668577.cos.ap-beijing.myqcloud.com
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
```

# 获取跨域配置

最近更新时间: 2024-12-19 17:12:00

## 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://www.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 | 否 |
| 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://www.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: examplebucket-1250000000.cos.ap-beijing.myqcloud.com
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://www.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: 2024-12-19 17:12:00

## 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

### 请求示例

```
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: arlenhuangtestsgnoversion-1251668577.cos.ap-beijing.myqcloud.com
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;3
2557712036&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: OGVmYzZiMmQzYjA2OWNhODk0NTRkMTBiOWVmMDAxODczNTBmNjMwZmQ0MTZkMjg0NjlkNTYyNmY4ZTRkZTk0N2M2MTdkZGZlMGN
hOWQyYjk3MWNmNWNkYzFhMjQzNzRiZTE1NjgzNzFhOGI5M2EwZDMyNGM4Y2ZmMzhiNTllMjk=
```

# 生命周期（lifecycle）接口
# 设置生命周期

最近更新时间: 2024-12-19 17:12:00

## 功能描述

CSP 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 CSP 定义一个操作)。 目前支持操作为：

- **过期操作：**指定 Object 的过期时间。CSP 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>r1</ID>
<Filter>
<Prefix>picture</Prefix>
</Filter>
<Status>Enabled</Status>
<Expiration>
<Days>30</Days>
</Expiration>
<NoncurrentVersionExpiration>
<NoncurrentDays>10</NoncurrentDays>
</NoncurrentVersionExpiration>
</Rule>
<Rule>
<ID>r2</ID>
<Filter>
```

```
<Prefix>record</Prefix>
</Filter>
<Status>Enabled</Status>
<Expiration>
<Days>60</Days>
</Expiration>
<NoncurrentVersionTransition>
<NoncurrentDays>20</NoncurrentDays>
</NoncurrentVersionTransition>
</Rule>
<Rule>
<ID>r3</ID>
<Filter>
<Prefix>alarm</Prefix>
</Filter>
<Status>Enabled</Status>
<Expiration>
<ExpiredObjectDeleteMarker>true</ExpiredObjectDeleteMarker>
</Expiration>
<NoncurrentVersionExpiration>
<NoncurrentDays>40</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<br>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 |
| 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 |

| 节点名称（关键字） | 父节点 | 描述 | 类型 | 必选 |
| --- | --- | --- | --- | --- |
| 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 CSP 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 Reques | 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>
<Expiration>
<Days>20</Days>
</Expiration>
</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: 2024-12-19 17:12:00

## 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>
<Expiration>
<Days>30</Days>
</Expiration>
</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: 2024-12-19 17:12:00

## 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=49c1414c700643f11139219384332a3ec4e9485b
```

**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）接口
# 设置存储桶策略

最近更新时间: 2024-12-19 17:12:00

## 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。

#### 非公共头部

该请求操作无特殊的请求头部信息。

### 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:examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com
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: 2024-12-19 17:12:00

## 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:examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com
Authorization:q-sign-algorithm=sha1&q-ak=AKIDxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx&q-sign-time=1484814099;32557710099&q-key-time=1484814099;3
2557710099&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: 2024-12-19 17:12:00

## 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:examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com
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）接口

# 设置存储桶referer

最近更新时间: 2024-12-19 17:12:00

## 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 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<br>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;
326200011134&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: 2024-12-19 17:12:00

## 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：Eabled，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 |

| 名称 | 父节点 | 描述 | 类型 | 必选 |
| --- | --- | --- | --- | --- |
| 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
# 设置存储桶标签

最近更新时间: 2024-12-19 17:12:00

## 功能描述

CSP 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。CSP 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: 2024-12-19 17:12:00

## 功能描述

CSP 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 |

| 节点名称（关键字） | 父节点 | 描述 | 类型 |
| --- | --- | --- | --- |
| 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 |
| 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，CSP 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: 2024-12-19 17:12:00

## 功能描述

CSP 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。CSP 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: 2024-12-19 17:12:00

## 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。

## 请求

**请求示例**

```
PUT /?website HTTP/1.1
Host: <BucketName-APPID>.<Endpoint>
Authorization: Auth String
[Request Body]
```

> **说明：**
>
> - `Host: <BucketName-APPID>.<Endpoint>` ，其中 `<BucketName-APPID>` 为带 APPID 后缀的存储桶名字，例如 examplebucket-1250000000，可参阅 存储桶概述 > 命名规范 文档；
> - 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>index.html</Suffix>
</IndexDocument>
<ErrorDocument>
<Key>error.html</Key>
<OriginalHttpStatus>Disabled</OriginalHttpStatus>
</ErrorDocument>
<AutoAddressing>
<Status>Enabled</Status>
</AutoAddressing>
<RoutingRules>
<RoutingRule>
<Condition>
<HttpErrorCodeReturnedEquals>403</HttpErrorCodeReturnedEquals>
</Condition>
<Redirect>
<Protocol>http</Protocol>
<ReplaceKeyWith>123.txt</ReplaceKeyWith>
</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 |

版权所有：亿算云平台

**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: <BucketName-APPID>.<Endpoint>
Date: Mon, 17 Jun 2019 08:37:35 GMT
Authorization: signatureValue

<WebsiteConfiguration>
<IndexDocument>
<Suffix>index.html</Suffix>
</IndexDocument>
<ErrorDocument>
<Key>error.html</Key>
<OriginalHttpStatus>Disabled</OriginalHttpStatus>
</ErrorDocument>
<AutoAddressing>
<Status>Enabled</Status>
</AutoAddressing>
<RoutingRules>
```

```
<RoutingRule>
<Condition>
<HttpErrorCodeReturnedEquals>403</HttpErrorCodeReturnedEquals>
</Condition>
<Redirect>
<Protocol>http</Protocol>
<ReplaceKeyWith>123.txt</ReplaceKeyWith>
</Redirect>
</RoutingRule>
</RoutingRules>
</WebsiteConfiguration>
```

**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****
```

# 查询静态网站

最近更新时间: 2024-12-19 17:12:00

## 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。

## 请求

**请求示例**

```
GET /?website HTTP/1.1
Host: <BucketName-APPID>.<Endpoint>
Date: GMT Date
Authorization: Auth String
```

> **说明：**
>
> - `Host: <BucketName-APPID>.<Endpoint>` ，其中 `<BucketName-APPID>` 为带 APPID 后缀的存储桶名字，例如 examplebucket-1250000000，可参阅 存储桶概述 > 命名规范 文档；
> - 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.<br>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: <BucketName-APPID>.<Endpoint>
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: 2024-12-19 17:12:00

## 功能描述

DELETE Bucket website 请求用于删除存储桶中的静态网站配置。

## 请求

**请求示例**

```
DELETE /?website HTTP/1.1
Host: <BucketName-APPID>.<Endpoint>
Date: GMT Date
Authorization: Auth String
```

> **说明：**
>
> - `Host: <BucketName-APPID>.<Endpoint>` ，其中 `<BucketName-APPID>` 为带 APPID 后缀的存储桶名字，例如 examplebucket-1250000000，可参阅 存储桶概述 > 命名规范 文档；
> - 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
```

# 版本控制接口
# 设置版本控制

最近更新时间: 2024-12-19 17:12:00

# 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 CSP 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<br>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=14809322
92;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: 2024-12-19 17:12:00

## 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=14809322
92;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>
```

# 存储桶加密
# 设置存储桶加密

最近更新时间: 2024-12-19 17:12:00

## 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>.<Endpoint>
Date: GMT Date
Authorization: Auth String
```

### 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</SSEAlgorithm>
</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 |
| Rules | ServerSideEncryptionConfiguration | 默认的服务端加密配置规则 | Container | 是 |
| ApplyServerSideEncryptionByDefault | Rules | 服务端加密的默认配置信息 | Container | 是 |
| SSEAlgorithm | ApplyServerSideEncryptionByDefault | 要使用的服务端加密算法，枚举值：AES256 或者 SM4 | 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: <BucketName-APPID>.<Endpoint>
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****
```

# 查询存储桶加密

最近更新时间: 2024-12-19 17:12:00

## 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>.<Endpoint>
Date: GMT Date
Authorization: Auth String
```

### 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

返回以下响应元素：

| 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 |
| Rules | ServerSideEncryptionConfiguration | 默认的服务端加密配置规则 | Container |
| ApplyServerSideEncryptionByDefault | Rules | 服务端加密的默认配置信息 | Container |
| SSEAlgorithm | ApplyServerSideEncryptionByDefault | 要使用的服务端加密算法，枚举值：AES256 或者 SM4 | 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>
```

# 删除存储桶加密

最近更新时间: 2024-12-19 17:12:00

## 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>.<Endpoint>
Date: GMT Date
Authorization: Auth String
```

### 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****
```

# 存储桶事件通知
# 设置存储桶事件通知

最近更新时间: 2024-12-19 17:12:00

## 功能描述

CSP 支持用户为 Bucket 配置事件通知，能够对用户所关心的对象操作及时进行消息通知，事件通知配置包含一个或多个 topic 通知规则。 目前对象操作包括上传对象类和删除对象类，消息通知端点为kafka。

**细节分析**

PUT Bucket notification 用于为 Bucket 创建一个新的事件通知配置。如果该 Bucket 已配置事件通知，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 /?notification HTTP/1.1
Host: <BucketName-APPID>.<Endpoint>
Authorization: Auth String
```

**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 语言设置存储桶事件通知配置。

以下为请求体：

```
<NotificationConfiguration xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
<TopicConfiguration>
<Id>123</Id>
<Topic>topic1</Topic>
<Endpoint>kafka://user1:passwd@100.99.173.49:9092</Endpoint>
<KafkaID>123#456</KafkaID>
<Event>cos:ObjectRemove:Delete</Event>
<Event>cos:ObjectRemove:DeleteMarkerCreated</Event>
</TopicConfiguration>
</NotificationConfiguration>
```

具体元素如下：

| 节点名称（关键字） | 父节点 | 描述 | 类型 | 必选 |
| --- | --- | --- | --- | --- |
| NotificationConfiguration | 无 | 保存TopicConfiguration的实体清单列表 | Container | 是 |
| TopicConfiguration | NotificationConfiguration | 通知规则描述，缺省时为清理存储桶事件通知 | Container | 否（最大50条） |
| Id | TopicConfiguration | 通知名称 | String | 是 |
| Topic | TopicConfiguration | 通知的topic名称 | String | 是 |
| Endpoint | TopicConfiguration | 通知端点信息，格式：kafka://[\:\@]\[:\] | String | 是 |
| KafkaID | TopicConfiguration | kafka实例唯一ID | String | 否 |
| Event | TopicConfiguration | 事件列表：缺省时为所有事件 | String | 否 |
| Filter | TopicConfiguration | 过滤规则 | Container | 否 |
| 3Key | Filter | 对象名过滤器 | Container | 是 |
| FilterRule | 3Key | 包含Name和Value实体 | Container | 是 |

| 节点名称（关键字） | 父节点 | 描述 | 类型 | 必选 |
| --- | --- | --- | --- | --- |
| Name | FilterRule | 过滤方法，取值只能为prefix或者suffix | string | 是 |
| Value | FilterRule | 过滤关键字 | string | 是 |

支持的事件类型列表如下：

| 事件类型 | 描述 |
| --- | --- |
| cos:ObjectCreated: * | 上传文件 |
| cos:ObjectCreated:Put | PUTObject |
| cos:ObjectCreated:Copy | PUTObjectCopy |
| cos:ObjectCreated:Post | POSTObject |
| cos:ObjectCreated:CompleteMultipartUpload | CompleteMultipartUpload |
| cos:ObjectRemove: * | 删除文件 |
| cos:ObjectRemove:Delete | 在未开启版本控制的存储桶下，使用 DELETE Object 接口删除对象，或者使用 versionid 删除指定版本的对象 |
| cos:ObjectRemove:DeleteMarkerCreated | 在开启或者暂停版本控制的存储桶下，使用 DELETE Object 接口删除对象 |

## 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

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 CSP 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 |
| InvalidArgument | 400 Bad Request | Id非法，Event非法，或者TopicConfiguration超过50条 |
| NotificationIDConflict | 400 Bad Request | ID冲突 |
| NotificationPrefixAndEventConflict | 400 Bad Request | 前缀以及事件类型冲突 |
| InvalidNotificationEndpoint | 400 Bad Reques | Endpoint格式错误 |

## 实际案例

### 请求

以下示例表示给存储桶 examplebucket-1250000000 设置存储桶事件通知。

```
PUT /?notification HTTP 1.1
Host: <BucketName-APPID>.<Endpoint>
Date: Mon, 17 Jun 2019 08:37:35 GMT
Authorization: signatureValue

<NotificationConfiguration xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
<TopicConfiguration>
```

```
<Id>123</Id>
<Topic>topic1</Topic>
<Endpoint>kafka://user1:passwd@100.99.173.49:9092</Endpoint>
<KafkaID>123#456</KafkaID>
<Event>cos:ObjectRemove:*</Event>
<Filter>
<S3Key>
<FilterRule>
<Name>prefix</Name>
<Value>pic</Value>
</FilterRule>
</S3Key>
</Filter>
</TopicConfiguration>
</NotificationConfiguration>
```

**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****
```

# 获取存储桶事件通知

最近更新时间: 2024-12-19 17:12:00

## 功能描述

GET Bucket notification 接口用于查询指定存储桶下的事件通知配置。

要执行此接口，必须拥有 GetBucketNotification 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 /?notification HTTP 1.1
Host: <BucketName-APPID>.<Endpoint>
Date: GMT Date
Authorization: Auth String
```

### 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

响应体中各个元素的内容及含义与 PUT Buket notification 时的请求体一致。详情请参见 设置存储桶事件通知文档中的请求体节点描述内容。

### 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 /?notification 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">
<NotificationConfiguration xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
<TopicConfiguration>
<Id>123</Id>
<Topic>topic1</Topic>
<Endpoint>kafka://user1:passwd@100.99.173.49:9092</Endpoint>
<KafkaID>123#456</KafkaID>
<Event>cos:ObjectRemove:*</Event>
</TopicConfiguration>
</NotificationConfiguration>
```

# Object接口
# 基本操作接口
# 上传对象

最近更新时间: 2024-12-19 17:12:00

## 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，CSP 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<br>**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）<br>**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 |

### 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: examplebucket-1250000000.cos.ap-beijing.myqcloud.com
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=
```

### 使用服务端加密SSE-COS AES256

#### 请求

```
PUT /object0 HTTP/1.1
Host: bucket0-1255000078.cos.chongqing.cos.****-v6-iaas.tcecloud.fsphere.cn
x-cos-server-side-encryption: AES256
x-cos-meta-md5: d41d8cd98f00b204e9800998ecf8427e
Content-Length: 0
Authorization: q-sign-algorithm=sha1&q-ak=AKID53h0z****&q-sign-time=1627895179;1627905239&q-key-time=1627895179;1627905239&q-header-list=host&q-url-param-list=&q-signature=****
```

#### 响应

```
HTTP/1.1 200 OK
Accept-Ranges: bytes
```

```
Content-Length: 0
Date: Mon, 02 Aug 2021 09:07:19 GMT
Etag: "d41d8cd98f00b204e9800998ecf8427e"
Server: nginx
X-Cache: MISS from SZ-SQUIDWEB-81
X-Cache-Lookup: MISS from SZ-SQUIDWEB-81:8080
X-Cos-Request-Id: tx0000000000000008eb048-006107b5d8-d2f0f8-default
X-Cos-Server-Side-Encryption: AES256
X-Response-Csp-Component: proxy-raw
```

### 使用服务端加密SSE-COS SM4

**请求**

```
PUT /object0 HTTP/1.1
Host: bucket0-1255000078.cos.chongqing.cos.****-v6-iaas.tcecloud.fsphere.cn
x-cos-server-side-encryption: SM4
x-cos-meta-md5: d41d8cd98f00b204e9800998ecf8427e
Content-Length: 0
Authorization: q-sign-algorithm=sha1&q-ak=AKID53h0z****&q-sign-time=1627895179;1627905239&q-key-time=1627895179;1627905239&q-header-list=host&q-url-param-list=&q-signature=****
```

**响应**

```
HTTP/1.1 200 OK
Accept-Ranges: bytes
Content-Length: 0
Date: Mon, 02 Aug 2021 09:07:19 GMT
Etag: "d41d8cd98f00b204e9800998ecf8427e"
Server: nginx
X-Cache: MISS from SZ-SQUIDWEB-81
X-Cache-Lookup: MISS from SZ-SQUIDWEB-81:8080
X-Cos-Request-Id: tx000000000000000b2a5ec-00621f3560-3f0d34-default
X-Cos-Server-Side-Encryption: SM4
X-Response-Csp-Component: proxy-raw
```

### 使用服务端加密SSE-KMS AES256（使用云产品密钥）

**请求**

```
PUT /object0 HTTP/1.1
Host: bucket0-1255000078.cos.chongqing.cos.****-v6-iaas.tcecloud.fsphere.cn
x-cos-server-side-encryption: cos/kms
x-cos-meta-md5: d41d8cd98f00b204e9800998ecf8427e
Content-Length: 0
Authorization: q-sign-algorithm=sha1&q-ak=AKID****&q-sign-time=1627895711;1627905771&q-key-time=1627895711;1627905771&q-header-list=host&q-url-param-list=&q-signature=****
```

**响应**

```
HTTP/1.1 200 OK
Accept-Ranges: bytes
Content-Length: 0
Date: Mon, 02 Aug 2021 09:16:14 GMT
Etag: "af1713a5330ed6211480563e170b0317"
Server: nginx
X-Cache: MISS from SZ-SQUIDWEB-81
X-Cache-Lookup: MISS from SZ-SQUIDWEB-81:8080
X-Cos-Request-Id: tx0000000000000008ec55b-006107b7ee-d2f0f8-default
X-Cos-Server-Side-Encryption: cos/kms
X-Cos-Server-Side-Encryption-Cos-Kms-Key-Id: 828b44f0-d4ef-****-****-6e3f5087083f
X-Response-Csp-Component: proxy-raw
```

### 使用服务端加密SSE-KMS SM4（使用云产品密钥）

**请求**

```
PUT /object0 HTTP/1.1
Host: bucket0-1255000078.cos.chongqing.cos.****-v6-iaas.tcecloud.fsphere.cn
x-cos-server-side-encryption: cos/kms/sm4
x-cos-meta-md5: d41d8cd98f00b204e9800998ecf8427e
Content-Length: 0
Authorization: q-sign-algorithm=sha1&q-ak=AKID****&q-sign-time=1627895711;1627905771&q-key-time=1627895711;1627905771&q-header-list=host
&q-url-param-list=&q-signature=****
```

**响应**

```
HTTP/1.1 200 OK
Accept-Ranges: bytes
Content-Length: 0
Date: Mon, 02 Aug 2021 09:16:14 GMT
Etag: "7378e9d047ad878a867d761c8334672e"
Server: nginx
X-Cache: MISS from SZ-SQUIDWEB-81
X-Cache-Lookup: MISS from SZ-SQUIDWEB-81:8080
X-Cos-Request-Id: tx000000000000000b2b518-00621f3c2e-3f0d34-default
X-Cos-Server-Side-Encryption: cos/kms/sm4
X-Cos-Server-Side-Encryption-Cos-Kms-Key-Id: 828b44f0-d4ef-****-****-6e3f5087083f
X-Response-Csp-Component: proxy-raw
```

### 使用服务端加密SSE-KMS AES256（使用用户自定义密钥）

**请求**

```
PUT /object0 HTTP/1.1
Host: bucket0-1255000078.cos.chongqing.cos.****-v6-iaas.tcecloud.fsphere.cn
x-cos-server-side-encryption: cos/kms
x-cos-server-side-encryption-cos-kms-key-id: 7addf848-eb91-****-****-6e3f5087083f
x-cos-meta-md5: d41d8cd98f00b204e9800998ecf8427e
Content-Length: 0
Authorization: q-sign-algorithm=sha1&q-ak=AKID****&q-sign-time=1627896177;1627906237&q-key-time=1627896177;1627906237&q-header-list=host
&q-url-param-list=&q-signature=****
```

**响应**

```
HTTP/1.1 200 OK
Accept-Ranges: bytes
Content-Length: 0
Date: Mon, 02 Aug 2021 09:23:57 GMT
Etag: "bb5a92aba42f8b5657af6823024f1370"
Server: nginx
X-Cache: MISS from SZ-SQUIDWEB-81
X-Cache-Lookup: MISS from SZ-SQUIDWEB-81:8080
X-Cos-Request-Id: tx0000000000000008edac0-006107b9bd-d2f0f8-default
X-Cos-Server-Side-Encryption: cos/kms
X-Cos-Server-Side-Encryption-Cos-Kms-Key-Id: 7addf848-eb91-****-****-6e3f5087083f
X-Response-Csp-Component: proxy-raw
```

### 使用服务端加密SSE-KMS SM4（使用用户自定义密钥）

**请求**

```
PUT /object0 HTTP/1.1
Host: bucket0-1255000078.cos.chongqing.cos.****-v6-iaas.tcecloud.fsphere.cn
x-cos-server-side-encryption: cos/kms/sm4
x-cos-server-side-encryption-cos-kms-key-id: 7addf848-eb91-****-****-6e3f5087083f
x-cos-meta-md5: d41d8cd98f00b204e9800998ecf8427e
Content-Length: 0
Authorization: q-sign-algorithm=sha1&q-ak=AKID****&q-sign-time=1627896177;1627906237&q-key-time=1627896177;1627906237&q-header-list=host
&q-url-param-list=&q-signature=****
```

**响应**

```
HTTP/1.1 200 OK
Accept-Ranges: bytes
Content-Length: 0
Date: Mon, 02 Aug 2021 09:23:57 GMT
Etag: "ed360e9f2fc4af5d2537b8172e5e11a9"
Server: nginx
X-Cache: MISS from SZ-SQUIDWEB-81
X-Cache-Lookup: MISS from SZ-SQUIDWEB-81:8080
X-Cos-Request-Id: tx000000000000000b2b8eb-00621f3df2-3f0d34-default
X-Cos-Server-Side-Encryption: cos/kms/sm4
X-Cos-Server-Side-Encryption-Cos-Kms-Key-Id: 7addf848-eb91-****-****-6e3f5087083f
X-Response-Csp-Component: proxy-raw
```

# 设置对象复制

最近更新时间: 2024-12-19 17:12:00

## 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。<br>**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: destinationbucket-1250000000.cos.ap-beijing.myqcloud.com
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;19
81012292&q-url-param-list=&q-header-list=host&q-signature=eacefe8e2a0dc8a18741d9a29707b1dfa5aa47cc
x-cos-copy-source: sourcebucket-1250000001.cos.ap-beijing.myqcloud.com/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>
```

# 获取对象

最近更新时间: 2024-12-19 17:12:00

## 功能描述

GET Object 接口请求可以在 CSP 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 |

### 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

### 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 | 用户自定义的元数据 |

## 响应体

下载成功，文件内容在响应体中。

## 错误码

| 错误码 | 描述 | HTTP 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: zuhaotestnorth-1251668577.cos.ap-beijing.myqcloud.com
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: zuhaotestnorth-1251668577.cos.ap-beijing.myqcloud.com
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]
```

# 获取对象元数据

最近更新时间: 2024-12-19 17:12:00

## 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: examplebucket-1250000000.cos.ap-beijing.myqcloud.com
```

```
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: 2024-12-19 17:12:00

## 功能描述

Delete Object 接口请求可以在 CSP 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。

## 请求

### 请求示例

```
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。

### 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CSP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: zuhaotestnorth-1251668577.cos.ap-beijing.myqcloud.com
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: 2024-12-19 17:12:00

## 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，CSP 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。

## 请求

### 请求示例

```
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 |
| --- | --- | --- | --- |

| 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。<br>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 |
| --- | --- | --- | --- |
| 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 CSP 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: lelu06-1252400000.cos.ap-guangzhou.myqcloud.com
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;1981012292&q-key-time=1480932292;1981012292&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: lelu06-1252400000.cos.ap-guangzhou.myqcloud.com
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;1981012292&q-key-time=1480932292;19
81012292&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: 2024-12-19 17:12:00

## 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 CSP，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 |
| 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 |

| 名称 | 类型 | 描述 |
| --- | --- | --- |
| 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: examplebucket-1250000000.cos.ap-beijing.myqcloud.com
Date: Thu, 12 Jan 2017 17:26:53 GMT
Origin: http://www.qq.com
Access-Control-Request-Method: PUT
Authorization: q-sign-algorithm=sha1&q-ak=AKIDxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx&q-sign-time=1487070734;32466654734&q-key-time=1487070734;3
2559966734&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://www.qq.com
Access-Control-Allow-Methods: PUT
Access-Control-Expose-Headers: x-cos-request-id
Server: tencent-cos
```

# 访问控制接口
# 设置对象ACL

最近更新时间: 2024-12-19 17:12:00

## 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）。<br>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://www.w3.org/2001/XMLSchema-instance" xsi:type="Group">
<URI>http://cam.qcloud.com/groups/global/AllUsers</URI>
</Grantee>
<Permission>READ</Permission>
</Grant>
<Grant>
<Grantee xmlns:xsi="http://www.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，<br>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 | 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 |
| 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: examplebucket-1250000000.cos.ap-beijing.myqcloud.com
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://www.w3.org/2001/XMLSchema-instance" xsi:type="Group">
<URI>http://cam.qcloud.com/groups/global/AllUsers</URI>
</Grantee>
<Permission>READ</Permission>
</Grant>
<Grant>
<Grantee xmlns:xsi="http://www.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
```

# 获取对象ACL

最近更新时间: 2024-12-19 17:12:00

## 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://www.w3.org/2001/XMLSchema-instance" xsi:type="Group">
<URI>http://cam.qcloud.com/groups/global/AllUsers</URI>
</Grantee>
<Permission>READ</Permission>
```

```
</Grant>
<Grant>
<Grantee xmlns:xsi="http://www.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，<br>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: examplebucket-1250000000.cos.ap-beijing.myqcloud.com
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://www.w3.org/2001/XMLSchema-instance" xsi:type="Group">
<URI>http://cam.qcloud.com/groups/global/AllUsers</URI>
</Grantee>
<Permission>READ</Permission>
</Grant>
<Grant>
<Grantee xmlns:xsi="http://www.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>
```

# 分块上传接口
# 初始化分块上传

最近更新时间: 2024-12-19 17:12:00

## 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<br>**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）<br>**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

### 案例一：简单案例

#### 请求

```
POST /exampleobject?uploads HTTP/1.1
Host: examplebucket-1250000000.cos.ap-beijing.myqcloud.com
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;32557623259&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>
```

**案例二：使用服务端加密 SSE-COS**

**请求**

```
POST /exampleobject?uploads= HTTP/1.1
Host: bucket0-1255000078.cos.chongqing.cos.****-v6-iaas.tcecloud.fsphere.cn
x-cos-server-side-encryption: AES256
x-cos-meta-md5: f0eae458e583c2658030a677a61c5aed
Content-Length: 0
Authorization: q-sign-algorithm=sha1&q-ak=AKID****&q-sign-time=1627907003;1627917063&q-key-time=1627907003;1627917063&q-header-list=content-length;x-cos-meta-md5;x-cos-server-side-encryption&q-url-param-list=uploads&q-signature=216811396aec7e449e751d24f3d8573da0cfb185
```

**响应**

```
HTTP/1.1 200 OK
Content-Length: 264
Content-Type: application/xml
Date: Mon, 02 Aug 2021 12:24:23 GMT
Server: nginx
X-Cache: MISS from SZ-SQUIDWEB-81
X-Cache-Lookup: MISS from SZ-SQUIDWEB-81:8080
X-Cos-Request-Id: tx00000000000000909d76-006107e407-d2f0f8-default
X-Cos-Server-Side-Encryption: AES256
X-Response-Csp-Component: proxy-raw

<?xml version="1.0" encoding="UTF-8"?><InitiateMultipartUploadResult xmlns="http://s3.amazonaws.com/doc/2006-03-01/"><Bucket>bucket0-1255000078</Bucket><Key>.....Nginx.pdf</Key><UploadId>2~64VtuaVu-zSWDWRliLAhbhXKy-PMQmc</UploadId></InitiateMultipartUploadResult>
```

# 上传分块

最近更新时间: 2024-12-19 17:12:00

## 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 CSP。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

### 请求示例

```
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。<br>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

**案例一：简单案例**

**请求**

```
PUT /ObjectName?partNumber=1&uploadId=1484727270323ddb949d528c629235314a9ead80f0ba5d993a3d76b460e6a9cceb9633b08e HTTP/1.1
Host: arlenhuangtestsgnoversion-1251668577.cos.ap-beijing.myqcloud.com
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
```

**案例二：使用服务端加密SSE-COS**

**请求**

```
PUT /exampleobject?partNumber=1&uploadId=2~64VtuaVu-zSWDWRliLAhbhXKy-PMQmc HTTP/1.1
Host: bucket0-1255000078.cos.chongqing.cos.****-v6-iaas.tcecloud.fsphere.cn
User-agent: coscmd-v1.8.6.24
Accept-Encoding: gzip, deflate
Accept: */*
Connection: keep-alive
x-cos-server-side-encryption: AES256
Content-Length: 1048576
Authorization: q-sign-algorithm=sha1&q-ak=AKID****&q-sign-time=1627907003;1627917063&q-key-time=1627907003;1627917063&q-header-list=host&q-url-param-list=&q-signature=ff0358a03c39a8df5ef6df8fa8d443b327ddf37b
[Object Content]
```

**响应**

```
HTTP/1.1 200 OK
Accept-Ranges: bytes
Content-Length: 0
Date: Mon, 02 Aug 2021 12:24:47 GMT
Etag: "1082d964b2be64ce18e9f0204a447be6"
Server: nginx
X-Cache: MISS from SZ-SQUIDWEB-81
X-Cache-Lookup: MISS from SZ-SQUIDWEB-81:8080
X-Cos-Request-Id: tx00000000000000000908ba3-006107e41f-d2e862-default
X-Cos-Server-Side-Encryption: AES256
X-Response-Csp-Component: proxy-raw
```

# 完成分块上传

最近更新时间: 2024-12-19 17:12:00

## 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，CSP 就立即返回200的状态码，在合并的过程中，CSP 会周期性的返回空格信息来保持连接活跃，直到合并完成，CSP 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

### 案例一：简单案例

#### 请求

```
POST /exampleobject?uploadId=1484728886e63106e87d8207536ae8521c89c42a436fe23bb58854a7bb5e87b7d77d4ddc48 HTTP/1.1
Host: examplebucket-1250000000.cos.ap-beijing.myqcloud.com
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>
```

### 案例二：使用服务端加密SSE-COS

**请求**

```
POST /exampleobject?uploadId=2~64VtuaVu-zSWDWRliLAhbhXKy-PMQmc HTTP/1.1
Host: bucket0-1255000078.cos.chongqing.cos.****-v6-iaas.tcecloud.fsphere.cn
User-Agent: coscmd-v1.8.6.24
Accept-Encoding: gzip, deflate
Accept: */*
Connection: keep-alive
Content-Length: 6115
Authorization: q-sign-algorithm=sha1&q-ak=AKID****&q-sign-time=1627907027;1627917087&q-key-time=1627907027;1627917087&q-header-list=content-length&q-url-param-list=uploadid&q-signature=cd34070708e1c0765e46197f7360bafd23d90b62

<?xml version="1.0" encoding="utf-8"?><CompleteMultipartUpload><Part><PartNumber>1</PartNumber><ETag>24351c2ab023527f8a73327ae89c0f23</ETag></Part><Part><PartNumber>2</PartNumber><ETag>6a0f112c302ffa3a92b3876238105441</ETag></Part><Part><PartNumber>3</PartNumber><ETag>6d665d678354e21190dd409b3f43d8ed</ETag></Part></CompleteMultipartUpload>
```

**响应**

```
HTTP/1.1 200 OK
Content-Length: 376
Content-Type: application/xml
Date: Mon, 02 Aug 2021 12:24:47 GMT
Server: nginx
X-Cache: MISS from SZ-SQUIDWEB-81
X-Cache-Lookup: MISS from SZ-SQUIDWEB-81:8080
X-Cos-Request-Id: tx000000000000000a93744-006107e420-caabb2-default
X-Cos-Server-Side-Encryption: AES256
X-Response-Csp-Component: proxy-raw

<?xml version="1.0" encoding="UTF-8"?><CompleteMultipartUploadResult xmlns="http://s3.amazonaws.com/doc/2006-03-01/"><Location>cos.chongqing.cos.****-v6-iaas.tcecloud.fsphere.cn/bucket0-1255000078/.....Nginx.pdf</Location><Bucket>bucket0-1255000078</Bucket><Key>.....Nginx.pdf</Key><ETag>&amp;quot;4233b029a4c1195daa4d605d9ba4640a-71&amp;quot;</ETag></CompleteMultipartUploadResult>
```

# 终止分块上传

最近更新时间: 2024-12-19 17:12:00

## 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。<br>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: examplebucket-1250000000.cos.ap-beijing.myqcloud.com
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;32557624626&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: 2024-12-19 17:12:00

## 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。<br>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<br>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<br>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<br>当 key-marker 未被指定时，upload-id-marker 将被忽略<br>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 |

| 节点名称（关键字） | 父节点 | 描述 | 类型 |
| --- | --- | --- | --- |
| IsTruncated | ListMultipartUploadsResult | 返回条目是否被截断，布尔值：TRUE，FALSE | Boolean |
| Prefix | ListMultipartUploadsResult | 限定返回的 Objectkey 必须以 Prefix 作为前缀，<br>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<br>encoding-type 只能取值 url，否则会返回 InvalidArgument |

获取更多关于 CSP 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: examplebucket-1250000000.cos.ap-beijing.myqcloud.com
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: 2024-12-19 17:12:00

## 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 |
| --- | --- | --- | --- |
| 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:examplebucket-1250000000.cos.ap-beijing.myqcloud.com
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>
```

# 标签
# 设置对象标签

最近更新时间: 2024-12-19 17:12:00

## 功能描述

CSP 支持为已存在的对象设置标签。PUT Object tagging 接口通过为对象添加键值对作为对象标签，可以协助您分组管理已有的对象资源。

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 PUT Object tagging 这个接口的权限。

> **注意：**
>
> 目前对象标签功能最多支持一个对象下设置10个不同的标签，如果超出设置上限，CSP 将返回失败。

**版本控制**

如果您的存储桶开启了版本控制，并且需要对指定版本的对象添加标签，可以在发起请求时携带 VersionId 参数，对象标签将添加到指定的对象版本中。

## 请求

**请求示例**

```
PUT /<ObjectKey>?tagging&VersionId=VersionId 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 |
| --- | --- | --- | --- |
| versionId | 当启用版本控制时，指定要操作的对象版本 ID，如不指定则将添加标签到最新版本的对象 | string | 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**

该请求需要设置如下标签集合：

```
<?xml version="1.0" encoding="UTF-8" ?>
<Tagging>
<TagSet>
<Tag>
<Key>string</Key>
<Value>string</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 | 是否必选 |
| --- | --- | --- | --- | --- |
| Tagging | 无 | 标签集合 | Container | 是 |
| 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 |

| 节点名称（关键字） | 父节点 | 描述 | 类型 | 是否必选 |
| --- | --- | --- | --- | --- |
| 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 | 是 |
| 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 | 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 |
| NoSuchKey | 如果试图添加的规则所在的对象不存在，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 | 如超过了允许一个对象最大设置的 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` 中的对象 `exampleobject.txt` 写入了 `{age:18}` 和 `{name:xiaoming}` 两个标签。CSP 配置标签成功并返回204成功请求。

```
PUT /exampleobject.txt?tagging HTTP/1.1
User-Agent: curl/7.29.0
Accept: */*
Host: examplebucket-1250000000.<Endpoint>
Authorization: Auth String
Content-Length: 127
Content-MD5:MD5 String
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 2020 11:40:22 GMT
Server: tencent-cos
x-cos-request-id: NWE2MWQ5MjZfMTBhYzM1MGFfMTA5ODVfMTVj****
```

# 查询对象标签

最近更新时间: 2024-12-19 17:12:00

# 功能描述

GET Object tagging 接口用于查询指定对象下已有的对象标签。

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 GET Object tagging 这个接口的权限。

**版本控制**

如果您的存储桶开启了版本控制，并且需要查询指定版本的对象的标签，可以在发起请求时携带 VersionId 参数，此时将查询指定版本对象的标签信息。

# 请求

**请求示例**

```
GET /<ObjectKey>?tagging&VersionId=VersionId 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 |
| --- | --- | --- | --- |
| versionId | 当启用版本控制时，指定要操作的对象版本 ID，如不指定则查询最新版本对象的对象标签 | string | 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**

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**

该请求返回的响应体节点描述如下：

| 节点名称（关键字） | 父节点 | 描述 | 类型 | 是否必选 |
| --- | --- | --- | --- | --- |
| Tagging | 无 | 标签集合 | Container | 是 |
| 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 | 是 |
| 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 | 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：

| 错误码 | 描述 | HTTP 状态码 |
| --- | --- | --- |

| 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 |
| NoSuchKey | 如果试图查询的规则所在的对象不存在，返回该错误码 | 404 Not Found |

## 实际案例

**请求**

如下请求申请查询存储桶 examplebucket-1250000000 中的对象 exampleobject.txt 下的标签信息，CSP 解析该请求后并返回该存储桶下已有的 {age:18} 和 {name:xiaoming} 两个标签。

```
GET /exampleobject.txt?tagging HTTP/1.1
User-Agent: curl/7.29.0
Accept: */*
Host: <BucketName-APPID>.<Endpoint>
Authorization: Auth String
Content-Md5: MD5 String
Content-Length: 127
Content-Type: application/xml
```

**响应**

```
HTTP/1.1 200 OK
Content-Type: application/xml
Connection: close
Date: Fri, 19 Jan 2020 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>
```

版权所有：亿算云平台

# 删除对象标签

最近更新时间: 2024-12-19 17:12:00

## 功能描述

DELETE Object tagging 接口用于删除指定对象下已有的对象标签。

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 Object tagging 这个接口的权限。

**版本控制**

如果您的存储桶开启了版本控制，并且需要删除指定版本对象的标签，可以在发起请求时携带 VersionId 参数，此时将删除指定版本 ID 对象的对象标签集。

## 请求

**请求示例**

```
DELETE /<ObjecKey>?tagging&VersionId=VersionId 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）。

**请求参数**

| 名称 | 描述 | 类型 | 是否必选 |
| --- | --- | --- | --- |
| versionId | 当启用版本控制时，指定要操作的对象版本 ID，如不指定则删除最新版本对象的标签 | string | 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**

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 |
| NoSuchKey | 如果试图删除的规则所在的对象不存在，返回该错误码 | 404 Not Found |

## 实际案例

**案例一：删除对象标签（未开启版本控制）**

**请求**

如下请求申请删除存储桶 examplebucket-1250000000 中的对象 exampleobject.txt 下已有的标签信息。CSP 解析该请求后删除该对象中的所有标签。

```
DELETE /exampleobject.txt?tagging HTTP/1.1
User-Agent: curl/7.29.0
Accept: */*
Host: examplebucket-1250000000.<Endpoint>
Authorization: Auth String
Content-Length: 0
Content-Type: application/xml
```

**响应**

```
HTTP/1.1 204 No Content
Content-Type: application/xml
Connection: close
Date: Fri, 19 Jan 2020 11:40:22 GMT
```

### 案例二：删除对象标签（开启版本控制）

**请求**

如下请求申请删除存储桶 examplebucket-1250000000 中的对象 exampleobject.txt 下已有的标签信息，删除时指定对象版本号为 MTg0NDUxNTgwODg2MTEzMTQyOTI 。CSP解析该请求后删除该版本的对象中的所有标签。

```
DELETE /exampleobject.txt?tagging&VersionId=MTg0NDUxNTgwODg2MTEzMTQyOTI HTTP/1.1
User-Agent: curl/7.29.0
Accept: */*
Host: examplebucket-1250000000.<Endpoint>
Authorization: Auth String
Content-Length: 0
Content-Type: application/xml
```

**响应**

```
HTTP/1.1 204 No Content
Content-Type: application/xml
Connection: close
Date: Fri, 19 Jan 2020 11:45:22 GMT
```

# 常见问题

## 一般性问题

最近更新时间: 2024-12-19 17:12:00

### 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。

### 存储桶创建数量有限制吗？

每个主账户最大1000个（默认）。每个存储桶中，单个bucket对象个数建议不要超过2000万。

### CSP 有数据统计功能吗？

在 CSP 控制台的监控列表页面，可切换项目和存储桶，查看存储容量、对象数量、总访问流量、请求数等信息。

### 如何使用自有域名访问对象？

可通过绑定自定义域名实现。

### CSP 是否支持 HTTPS 加密传输数据呢？

A：CSP 在所有可用地域的访问节点 都提供了 SSL 传输的支持，且在 SDK 和控制台都默认启用 HTTPS。**强烈建议您使用 HTTPS 保护传输的数据链路，使用不加密的 HTTP 连接将可能面临链路被监听或数据被窃取的风险。**

### 回源地址的作用？

进行数据迁移。当 CSP 上没有用户请求访问的资源时，会从回源地址拉取。

### 设置回源时，CSP 上不存在回源地址的相应资源或路径时，CSP 会在用户首次访问后自动创建目录吗？

会，CSP 会自动拉取并创建目录。

# 工具问题
# COSCMD工具

最近更新时间: 2024-12-19 17:12:00

### 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: 2024-12-19 17:12:00

### 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存储桶中的数据，因而触发了访问频率的限制。CSP 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: 2024-12-19 17:12:00

### 什么是 COSBrowser工具？

COSBrowser 是亿算云平台对象存储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 CSP 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对象存储的相关权限，相关文档可参见 授权子账号访问 CSP。
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。


上传
下载
通用
代理
关于
账号
文件并发数
5
分块并发数
5
失败重试次数
5
计算并添加 x-cos-meta-md5 头部 (增加耗时)
(用于在其他渠道下载对象时，校验完成性)
上传完二次校验
(上传完成后，获取对象信息，对比本地文件)
还原默认值
保存

# COS Migration工具

最近更新时间: 2024-12-19 17:12:00

### 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 CSP 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 CSP 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 CSP，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，CSP 目前对一个账号具有每秒800 QPS 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?

```
Exception in thread "main" java.lang.ExceptionInInitializerError
        at org.rocksdb.RocksDB.loadLibrary(RocksDB.java:64)
        at org.rocksdb.RocksDB.<clinit>(RocksDB.java:35)
        at org.rocksdb.Options.<clinit>(Options.java:25)
        at com.qcloud.cos_migrate_tool.record.RecordDb.init(RecordDb.java:43)
        at com.qcloud.cos_migrate_tool.task.TaskExecutor.initRecord(TaskExecutor.java:94)
        at com.qcloud.cos_migrate_tool.task.TaskExecutor.run(TaskExecutor.java:146)
        at com.qcloud.cos_migrate_tool.app.App.main(App.java:60)
Caused by: java.lang.UnsupportedOperationException: Cannot determine JNI library name for ARCH='x86' OS='windows 10' nam
e='rocksdb'
        at org.rocksdb.util.Environment.getJniLibraryName(Environment.java:78)
        at org.rocksdb.NativeLibraryLoader.<clinit>(NativeLibraryLoader.java:19)
        ... 7 more
```

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。

# 词汇表

最近更新时间: 2024-12-19 17:12:00

### 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**账号中心**查看 APPID。

### 存储桶

存储桶（Bucket）是对象的载体，可理解为存放对象的"容器"。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。相关文档请参见存储桶概述。

### 对象

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）。用户可以通过控制台、API、SDK 等多种方式管理对象。相关文档请参见对象概述。

### 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**访问管理 > 云 API 密钥**页面新建，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**访问管理 > 云 API 密钥**页面新建，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。

# API文档