

对象存储

高性能存储



腾讯云

【 版权声明 】

©2013–2026 腾讯云版权所有

本文档（含所有文字、数据、图片等内容）完整的著作权归腾讯云计算（北京）有限责任公司单独所有，未经腾讯云事先明确书面许可，任何主体不得以任何形式复制、修改、使用、抄袭、传播本文档全部或部分内容。前述行为构成对腾讯云著作权的侵犯，腾讯云将依法采取措施追究法律责任。

【 商标声明 】



及其它腾讯云服务相关的商标均为腾讯云计算（北京）有限责任公司及其关联公司所有。本文档涉及的第三方主体的商标，依法由权利人所有。未经腾讯云及有关权利人书面许可，任何主体不得以任何方式对前述商标进行使用、复制、修改、传播、抄录等行为，否则将构成对腾讯云及有关权利人商标权的侵犯，腾讯云将依法采取措施追究法律责任。

【 服务声明 】

本文档意在向您介绍腾讯云全部或部分产品、服务的当时的相关概况，部分产品、服务的内容可能不时有所调整。您所购买的腾讯云产品、服务的种类、服务标准等应由您与腾讯云之间的商业合同约定，除非双方另有约定，否则，腾讯云对本文档内容不做任何明示或默示的承诺或保证。

【 联系我们 】

我们致力于为您提供个性化的售前购买咨询服务，及相应的技术售后服务，任何问题请联系 4009100100或 95716。

文档目录

高性能存储

产品概述

快速入门

访问控制

API 文档

简介

公共请求头部

公共响应头部

错误码

存储桶操作

GetService

CreateSession

PutBucket

DeleteBucket

HeadBucket

存储桶策略

PutBucketPolicy

GetBucketPolicy

DeleteBucketPolicy

对象操作

PutObject

GetObject

HeadObject

DeleteObject

DeleteMultipleObjects

PutObjectCopy

GetBucket

RenameObject

分块上传操作

InitiateMultipartUpload

UploadPart

UploadPartCopy

CompleteMultipartUpload

AbortMultipartUpload

ListMultipartUploads

ListParts

SDK 文档

简介

Python SDK

快速入门

高级传输接口

高级复制对象

下载到本地文件

上传本地文件

上传可读流

访问授权

预签名 URL

存储桶操作

查询存储桶列表

创建访问会话

创建存储桶

删除存储桶

查询存储桶

存储桶策略

设置存储桶策略

获取存储桶策略

删除存储桶策略

对象操作

重命名对象

上传对象

下载对象

查询对象元数据

删除对象

批量删除对象

复制对象

列举对象

分块上传操作

初始化分块上传

上传分块

复制分块

完成分块上传

终止分块上传

列举分块上传

列举已上传分块

异常处理

Go SDK

快速入门

存储桶操作

查询存储桶列表

创建访问会话

创建存储桶

删除存储桶

查询存储桶

存储桶策略

设置存储桶策略

获取存储桶策略

删除存储桶策略

对象操作

重命名对象

上传对象

下载对象

查询对象元数据

删除对象

批量删除对象

复制对象

列举对象

分块上传操作

初始化分块上传

上传分块

复制分块

完成分块上传

终止分块上传

列举分块上传

列举已上传分块

异常处理

高性能存储

产品概述

最近更新时间：2026-10-21 18:34:03

高性能存储桶（COS Rapid Bucket）是腾讯云对象存储（COS）推出的面向 AI / HPC 场景的全新存储桶类型，兼容对象存储协议，提供高并发、低延迟的存储读写服务。

在 AI 训练和高性能计算场景中，传统对象存储的吞吐和延迟往往成为数据流水线的瓶颈——GPU 集群空转等待 I/O，昂贵算力被白白浪费。高性能存储桶基于全 SSD 介质、单可用区就近部署架构，将计算节点与存储之间的网络路径缩至最短，提供百万级 IOPS、毫秒级时延和 TB/s 级吞吐能力。同时兼容 S3 协议，支持 COS SDK、AWS SDK 直连；兼容 POSIX 接口，提供 FUSE 文件系统挂载；兼容 HCFS 协议，支撑大数据业务。上层应用仅需简单适配后即可接入。

核心概念

理解高性能存储桶，需要掌握以下几个核心概念：

- **高性能存储桶（Rapid Bucket）**：高性能存储桶是 COS 的全新存储桶类型，配备专属高性能数据通路和独立域名。高性能存储桶使用专有命名格式 `<BucketName>-x--<APPID>`，其中 `-x-` 是高性能存储桶的固定标识。例如：`examplebucket-x--1250000000`。
- **对象（Object）**：对象是高性能存储桶中的基本存储单元，由数据和元数据组成，可理解为任何格式类型的数据，例如图片、文档、模型文件、Checkpoint 和数据集等。单次上传（PutObject）支持最大 5GB 的对象，更大的文件可通过分片上传（Multipart Upload）完成。
- **目录对象（Directory Object）**：与普通 COS 存储桶的"前缀模拟目录"不同，高性能存储桶提供原生目录结构，目录是一等实体对象，具备真正的层级命名空间语义，支持 rename、list dir 等文件语义操作。API 通过专用错误码（DirAlreadyExists、DirNotEmpty、DirectoryObjectContainsData）保证目录操作的安全性，避免误删非空目录等问题。
- **地域（Region）**：是腾讯云托管机房的分布地区，高性能存储桶的数据存放在这些地域的高性能存储集群中。高性能存储桶采用单可用区部署，计算与存储就近部署以获得更低的访问延迟。

产品优势

- **超高性能**：基于全 SSD 介质构建，通过 Direct I/O 直写底层存储，提供百万级 IOPS、毫秒级平均时延和 TB/s 级吞吐能力。对于不超过 64KB 的小文件，数据直接内联存储于元数据中，一次 RPC 即完成写入，大幅提升海量小文件场景的性能表现。
- **单可用区就近部署**：计算节点与存储节点部署在同一可用区内，网络路径最短，大幅降低读写延迟，为延迟敏感型业务提供更优的读写响应速度。
- **原生目录结构**：提供真正的层级命名空间，目录是一等实体对象，原生支持 rename、list dir 等文件语义操作。训练数据集和 Checkpoint 的目录组织可直接映射，无需适配前缀模拟的"伪目录"。

- **生态兼容**：兼容标准 S3 协议，签名方式与 COS 完全一致，支持 COS SDK、AWS SDK 直连。提供 POSIX 文件语义的挂载访问，应用像访问本地文件系统一样读写高性能存储桶。支持 HCFS 协议，上层大数据组件可直接访问。多种协议兼容，PyTorch、TensorFlow、Spark / Hadoop 等框架简单适配即可接入。

应用场景

高性能存储桶作为统一存储底座，可广泛应用于以下场景：

AI/ML 模型训练与加速

在大规模深度学习训练中，模型训练的效率高度依赖数据供给速度。传统对象存储的吞吐和延迟往往导致 GPU 频繁等待 I/O，昂贵的算力被白白浪费。

高性能存储桶通过单可用区就近部署架构，将训练数据集与 AI 加速器部署在同一物理区域内，网络路径缩至最短。TB/s 级聚合吞吐和毫秒级时延，确保数千张 GPU 同时读取海量随机样本时不再被存储拖慢，最大化 GPU 利用率（MFU），缩短训练周期，降低算力总成本。

快速模型 Checkpoint

在长时间训练过程中，需要定期保存模型权重和优化器状态（Checkpoint），以便在节点故障时快速恢复，避免从零开始。传统方案下，写一个大规模 Checkpoint 需要数十分钟，期间所有 GPU 完全空转。

高性能存储桶提供超高并发写入能力——数千个训练 Worker 同时写各自的分片（Shard），聚合吞吐可达 TB/s 级，配合毫秒级写入延迟，Checkpoint 耗时从分钟级降至秒级。训练团队可以更频繁地保存 Checkpoint（如每几分钟一次），故障恢复只需回退数分钟而非数小时。

高并发 AI 模型推理

在线推理服务需要应对高并发请求，快速加载模型权重、LoRA 适配器、Embedding 索引等大体积资产到 GPU 显存。在流量突增时，推理平台自动扩容拉起新节点，冷启动延迟直接影响用户体验。

高性能存储桶支持百万级 IOPS 和 TB/s 级吞吐，配合单可用区就近部署，模型加载时间从分钟级降至秒级，LoRA 适配器切换近乎实时。结合 Kubernetes 自动扩缩容，实现流量来了秒级拉起、秒级加载、立即推理，流量走了释放资源不浪费 GPU。

AI/ML 数据集预处理

在模型训练之前，原始数据需要经过清洗、分词（Tokenization）、格式转换等预处理流程。这些环节产生大量中间文件的密集读写，对存储的 IOPS 和吞吐都有很高要求。

高性能存储桶轻松承接高性能数据预处理框架的密集读写事务负载。数据预处理器可以超高速率完成数据摄入和输出，配合原生目录结构，训练数据集的目录组织可以直接映射，无需适配前缀模拟的“伪目录”，简化数据管理流程。

批处理与流式分析

金融量化建模、HPC 仿真、实时日志 / IoT 遥测数据等场景，需要持续高速写入数据，并且新写入的数据需要立即可见供下游消费。传统方案往往需要叠加 Kafka / Redis 等多层中间件，架构复杂、运维成本高。

高性能存储桶支持百万级 QPS 持续写入，新写入数据可立即被读取端消费，无需额外中间件层。实时消费应用可以在数据持续写入的同时并发读取最新内容，将多套系统精简为一套存储，降低架构复杂度和运维负担。

访问域名

高性能存储桶提供独立的 VPC 内网访问域名，与普通 COS 存储桶隔离。高性能存储桶区分控制面域名和数据面域名。

- 控制面域名：`service.cosrapid.<Region>.myqcloud.com`
- 数据面域名：`<BucketName>-x--<APPID>.cosrapid.<Region>.myqcloud.com`

⚠ 注意：

高性能存储桶仅支持 VPC 内网访问，暂不支持公网访问，请确保您的客户端部署在腾讯云 VPC 网络环境中。使用 SDK 或 API 接入时，请注意将 Endpoint 配置为高性能存储桶的专属域名（`cosrapid.<Region>.myqcloud.com`），而非普通 COS 域名（`cos.<Region>.myqcloud.com`）。

支持地域

高性能存储桶目前支持的地域列表如下，其他地域会陆续上线：

地域	地域简称	控制面域名	数据面域名
上海	ap-shanghai	service.cosrapid.ap-shanghai.myqcloud.com	<BucketName>-x--<APPID>.cosrapid.ap-shanghai.myqcloud.com
新加坡	ap-singapore	service.cosrapid.ap-singapore.myqcloud.com	<BucketName>-x--<APPID>.cosrapid.ap-singapore.myqcloud.com

📌 说明：

高性能存储桶当前处于公测阶段。如需使用，请 [联系我们](#) 开通资格。

快速入门

最近更新时间：2026-10-21 18:34:03

本文档介绍如何快速上手高性能桶（Rapid Bucket），完成从准备到上传、下载对象的完整流程，主要包含四个环节：准备工作、创建高性能桶、上传对象、下载对象。

简介

高性能桶（COS Rapid Bucket）是面向 AI / HPC 场景的高性能存储桶类型，兼容 S3 对象存储协议，提供高并发、低延迟的读写能力。其数据面采用基于会话（Session）的轻量级鉴权机制，SDK 会自动完成会话创建与凭证复用，业务代码无需手动管理。

准备工作

开通 COS 服务

在 [腾讯云控制台](#) 搜索对象存储，进入对象存储控制台，按界面提示开通 COS 服务。（如已开通，请跳过此步骤。）

申请公测资格

当前高性能桶处于公测阶段，请 [联系我们](#) 申请公测资格。

准备 VPC 网络

高性能桶当前仅支持通过腾讯云 VPC 内网访问。请确保客户端部署在腾讯云 VPC 环境中，并准备好创建桶所需的 VPC 网络信息：VPC ID、子网 ID 与可用区（`x-cos-vpc-id`、`x-cos-subnet-id`、`x-cos-zone`）。

获取访问密钥

获取访问密钥 `SecretId` 与 `SecretKey`。建议使用子账号密钥并遵循最小权限原则，子账号密钥可在 [访问管理控制台](#) 获取。

操作步骤

下面以 Python SDK 为例演示完整流程。SDK 的安装与客户端初始化请参见 [Python SDK 快速入门](#)；高性能桶的完整名称形如 `<BucketName>-x--<APPID>`，例如 `examplebucket-x--1250000000`。

步骤 1：初始化客户端

访问高性能桶时打开 `EnableRapidDomain`，SDK 会自动使用高性能桶域名并在数据面请求时自动完成会话鉴权。

```
# -*- coding=utf-8 -*-
```

```
import os
from qcloud_cos import CosConfig, CosS3Client

# 高性能桶经 VPC 内网 HTTP 80 端口接入, SDK 的网关 DNS 负载均衡也仅对 HTTP 请求生效; SDK 默认 https, 此处需显式指定 http。
config = CosConfig(
    Scheme='http',
    Region='ap-guangzhou',
    SecretId=os.environ['COS_SECRET_ID'],
    SecretKey=os.environ['COS_SECRET_KEY'],
    Token=os.environ.get('COS_SECURITY_TOKEN'),
    EnableRapidDomain=True)
client = CosS3Client(config)

bucket = 'examplebucket-x--1250000000'
```

步骤 2: 创建高性能桶

创建高性能桶时必须指定 VPC 网络信息。也可跳过此步, 直接使用已创建的桶。

`create_bucket` 成功返回只表示创建请求已受理, 不代表存储桶已可用。请使用 `head_bucket` 查询 `x-cos-bucket-status`, 待其为 `Available` 后再进行对象操作。

```
client.create_bucket(
    Bucket=bucket,
    VpcId='vpc-xxxxxxx',
    SubnetId='subnet-xxxxxxx',
    Zone='ap-guangzhou-3')

# 创建请求受理后轮询桶状态, 待 Available 后再进行对象操作。
import time
for _ in range(60):
    status = client.head_bucket(Bucket=bucket).get('x-cos-bucket-status')
    if status == 'Available':
        break
    time.sleep(2)
else:
    raise RuntimeError('bucket not available, last status: %s' % status)
print('CreateBucket OK')
```

步骤 3：上传对象

首次对高性能桶发起数据面请求时，SDK 会自动创建 session 并签名。

```
key = 'quickstart/hello.txt'
client.put_object(Bucket=bucket, Key=key, Body=b'hello rapid bucket')
print('PutObject OK')
```

步骤 4：下载对象

```
resp = client.get_object(Bucket=bucket, Key=key)
body = resp['Body'].get_raw_stream().read()
print('GetObject OK: %s' % body)
```

后续操作

- 了解高性能桶的能力与限制，请参见 [产品概述](#)。
- 了解鉴权模型与权限配置，请参见 [访问控制](#)。
- 使用 Python SDK 的更多接口，请参见 [Python SDK 快速入门](#)。
- 使用 Go SDK，请参见 [Go SDK 快速入门](#)。
- 直接使用 REST API，请参见 [API 文档 · 简介](#)。

访问控制

最近更新时间：2026-10-21 18:34:03

概述

高性能桶（Rapid Bucket）采用与 AWS S3 Express One Zone 类似的双层鉴权模型，将桶级别操作（控制面）与对象级别操作（数据面）的鉴权机制分离，以在保障安全性的前提下实现数据面的低延迟访问。

核心设计理念：

- **控制面**（桶级别操作）：使用标准的 CAM（云访问管理）身份鉴权，通过 V4 签名校验请求合法性。
- **数据面**（对象级别操作）：使用基于会话（Session）的轻量级鉴权机制，通过 `CreateSession` 获取临时凭证，减少每次请求的鉴权开销。

访问控制边界

高性能桶的访问控制边界仅设置在桶级别。与标准 COS 通用桶不同，高性能桶不支持对象级别（前缀或标签）的精细权限策略。这意味着：

- 在 CAM 身份策略或桶策略中，高性能桶是唯一可以指定的资源粒度。
- 不支持基于对象前缀（prefix）或对象标签（tag）的条件策略。
- 桶默认私有，只有被明确授予权限的用户才能访问。

双层鉴权模型

控制面鉴权（桶级别操作）

桶级别操作使用标准的 CAM 身份鉴权模型，不涉及会话管理。权限通过 CAM 策略逐一授予。

适用的 API 操作：

API 操作	CAM Action	说明
PutBucket	cos:CreateBucket	创建高性能桶
DeleteBucket	cos>DeleteBucket	删除高性能桶
HeadBucket	cos:HeadBucket	查询桶是否存在
PutBucketPolicy	cos:PutBucketPolicy	设置桶策略
GetBucketPolicy	cos:GetBucketPolicy	查询桶策略
DeleteBucketPolicy	cos>DeleteBucketPolicy	删除桶策略

控制面操作的请求域名为区域端点（Regional Endpoint），格式：


```
<BucketName>-x--<APPID>.cosrapid.<Region>.myqcloud.com
```

数据面鉴权（对象级别操作）

对象级别操作使用基于会话的鉴权机制。客户端通过 `CreateSession` API 创建会话并获取临时凭证，后续数据面请求携带该临时凭证即可完成鉴权，无需每次请求都进行完整的 V4 签名计算。

适用的 API 操作：

API 操作	所需权限	说明
PutObject	cos:CreateSession	上传对象
GetObject	cos:CreateSession	下载对象
HeadObject	cos:CreateSession	查询对象元数据
DeleteObject	cos:CreateSession	删除单个对象
DeleteMultipleObjects	cos:CreateSession	批量删除对象
PutObjectCopy	cos:CreateSession	桶内复制对象
ListObjects	cos:CreateSession	列出对象
RenameObject	cos:CreateSession	重命名对象
InitiateMultipartUpload	cos:CreateSession	初始化分片上传
UploadPart	cos:CreateSession	上传分片
UploadPartCopy	cos:CreateSession	复制分片
CompleteMultipartUpload	cos:CreateSession	完成分片上传
AbortMultipartUpload	cos:CreateSession	终止分片上传
ListMultipartUploads	cos:CreateSession	列出正在进行的分片上传
ListParts	cos:CreateSession	列出已上传的分片

CreateSession 会话机制

工作原理

`CreateSession` 是高性能桶数据面鉴权的核心机制。通过将身份验证和授权集中到会话创建阶段，后续数据面请求只需携带轻量级会话凭证，从而降低每次请求的鉴权延迟。

工作流程如下：

1. 客户端使用 CAM 永久/临时凭证调用 `CreateSession`，传入目标桶名称。
2. 服务端校验调用方身份及其 `cos:CreateSession` 权限，校验通过后返回临时凭证。
3. 客户端使用返回的临时凭证（`AccessKeyId + SecretAccessKey + SessionToken`）直接调用数据面 API。
4. 临时凭证过期后，客户端需重新调用 `CreateSession` 获取新凭证。

临时凭证

`CreateSession` 返回的临时凭证包含：

字段	说明
<code>AccessKeyId</code>	临时访问密钥 ID
<code>SecretAccessKey</code>	临时访问密钥
<code>SessionToken</code>	会话令牌，数据面请求必须携带
<code>Expiration</code>	凭证过期时间

临时凭证的关键特性：

- **桶级别作用域**：凭证仅对创建时指定的高性能桶有效，不能跨桶使用。
- **有效期短**：临时凭证有效期为 5 分钟，过期后需重新创建会话。
- **权限继承**：临时凭证的权限与调用 `CreateSession` 时的身份权限一致。

会话模式（SessionMode）

创建会话时可以指定会话模式，控制临时凭证的读写权限：

模式	说明
<code>ReadWrite</code>	读写模式，允许所有对象级别操作
<code>ReadOnly</code>	只读模式，仅允许 <code>GetObject</code> 、 <code>HeadObject</code> 、 <code>ListObjects</code> 等读操作

如果未显式指定 `SessionMode`，系统将自动尝试最大权限：先尝试 `ReadWrite`，若策略不允许则降级为 `ReadOnly`。

可以在桶策略中通过条件键 `cos:SessionMode` 精细控制不同身份可以创建的会话类型。例如，限制某些只读角色只能创建 `ReadOnly` 会话。

权限策略

CAM 身份策略

通过 CAM 身份策略（附加到用户、用户组或角色）授予高性能桶的访问权限。策略示例：

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

上述策略授予对指定高性能桶的数据面完整访问权限（通过 `CreateSession` 授权所有对象操作）。

场景一：授予子账号只读访问权限

```
{
  "version": "2.0",
  "statement": [
    {
      "effect": "allow",
      "action": [
        "cos:CreateSession"
      ],
      "resource": [
        "qcs::cosrapid:ap-nanjing:uid/1250000000:examplebucket-x-1250000000/*"
      ],
      "condition": {
        "string_equal": {
          "cos:SessionMode": "ReadOnly"
        }
      }
    }
  ]
}
```

```
]
}
```

通过 `cos:SessionMode` 条件键限制子账号只能创建只读会话，从而限制其只能执行 `GetObject`、`HeadObject`、`ListObjects` 等读操作。

场景二：授予子账号对多个高性能桶的完整访问权限

```
{
  "version": "2.0",
  "statement": [
    {
      "effect": "allow",
      "action": [
        "cos:CreateSession"
      ],
      "resource": [
        "qcs::cosrapid:ap-nanjing:uid/1250000000:training-data-x-1250000000/*",
        "qcs::cosrapid:ap-nanjing:uid/1250000000:model-output-x-1250000000/*"
      ]
    }
  ]
}
```

一条策略同时授权多个高性能桶的读写访问，适用于 AI 训练场景中同时需要读取训练数据桶和写入模型输出桶的情况。

场景三：授予子账号桶管理权限（控制面）

```
{
  "version": "2.0",
  "statement": [
    {
      "effect": "allow",
      "action": [
        "cos:CreateBucket",
        "cos>DeleteBucket",
        "cos:HeadBucket",

```

```
        "cos:PutBucketPolicy",
        "cos:GetBucketPolicy",
        "cos>DeleteBucketPolicy"
    ],
    "resource": [
        "*"
    ]
}
]
```

授予子账号在指定地域下创建和管理所有高性能桶的权限。通常只授予管理员角色。

场景四：完整权限（控制面 + 数据面）

```
{
  "version": "2.0",
  "statement": [
    {
      "effect": "allow",
      "action": [
        "cos:CreateBucket",
        "cos>DeleteBucket",
        "cos:HeadBucket",
        "cos:PutBucketPolicy",
        "cos:GetBucketPolicy",
        "cos>DeleteBucketPolicy",
        "cos:CreateSession"
      ],
      "resource": [
        "qcs::cosrapid:ap-nanjing:uid/1250000000:examplebucket-x-1250000000/*"
      ]
    }
  ]
}
```

同时授予桶管理和数据读写的完整权限，适用于桶的完全管理者。

桶策略

高性能桶支持桶策略（Bucket Policy），可实现精细化的桶级别访问控制。桶策略示例：

```
{
  "version": "2.0",
  "statement": [
    {
      "effect": "allow",
      "principal": {
        "qcs": [
          "qcs::cam::uin/1000000000001:uin/1000000000002"
        ]
      },
      "action": [
        "cos:CreateSession"
      ],
      "resource": [
        "qcs::cosrapid:ap-nanjing:uid/1250000000:examplebucket-x-1250000000/*"
      ],
      "condition": {
        "string_equal": {
          "cos:SessionMode": "ReadOnly"
        }
      }
    }
  ]
}
```

上述策略授予指定用户以只读模式访问高性能桶的权限。

场景一：允许特定用户访问

```
{
  "version": "2.0",
  "statement": [
    {
      "effect": "allow",
      "principal": {
        "qcs": [
          "qcs::cam::uin/1000000000001:uin/1000000000002",
```

```
        "qcs::cam::uin/100000000001:uin/100000000003"
    ],
    },
    "action": [
        "cos:CreateSession"
    ],
    "resource": [
        "qcs::cosrapid:ap-nanjing:uid/1250000000:shared-data-x-12500000000/*"
    ]
}
]
```

允许多个用户以读写模式访问共享数据桶，适用于多团队协作场景。

场景二：授予特定用户读写模式访问

```
{
  "version": "2.0",
  "statement": [
    {
      "effect": "allow",
      "principal": {
        "qcs": [
          "qcs::cam::uin/100000000001:uin/100000000005"
        ]
      },
      "action": [
        "cos:CreateSession"
      ],
      "resource": [
        "qcs::cosrapid:ap-nanjing:uid/1250000000:examplebucket-x-12500000000/*"
      ],
      "condition": {
        "string_equal": {
          "cos:SessionMode": "ReadWrite"
        }
      }
    }
  ]
}
```

```
    }
  ]
}
```

明确授予特定用户 ReadWrite 模式的会话权限。在需要对方写入数据（如合作方上传训练结果）时使用。

场景三：拒绝特定账号访问（黑名单）

```
{
  "version": "2.0",
  "statement": [
    {
      "effect": "deny",
      "principal": {
        "qcs": [
          "qcs::cam::uin/1000000000001:uin/1000000000099"
        ]
      },
      "action": [
        "cos:CreateSession"
      ],
      "resource": [
        "qcs::cosrapid:ap-nanjing:uid/1250000000:examplebucket-x-1250000000/*"
      ]
    }
  ]
}
```

显式拒绝特定账号创建会话，即使该账号通过其他策略获得了权限。Deny 策略优先级高于 Allow。

策略条件键

高性能桶支持以下条件键用于精细权限控制：

条件键	说明	适用场景
cos:SessionMode	限制可创建的会话模式（ReadWrite/ReadOnly）	控制读写权限级别

与标准 COS 桶鉴权的对比

维度	标准 COS 桶	高性能桶（Rapid Bucket）
鉴权模型	每次请求 V4 签名	双层模型：控制面 V4 签名 + 数据面会话凭证
权限粒度	桶/前缀/对象标签	仅桶级别
匿名访问	支持（公有读）	不支持
临时凭证有效期	STS 凭证最长 2 小时	会话凭证固定 5 分钟
ACL	支持	不支持（仅策略）
Presigned URL	支持	不支持

最佳实践

- 使用临时凭证：**生产环境建议使用 STS 临时凭证调用 `CreateSession`，避免永久密钥暴露。
- 会话复用：**在 5 分钟有效期内复用同一会话凭证执行多次请求，减少 `CreateSession` 调用频率。
- 最小权限原则：**按需授予 `ReadOnly` 或 `ReadWrite` 模式，避免过度授权。
- SDK 自动管理：**推荐使用腾讯云 COS SDK（最新版），SDK 内置会话自动创建和刷新逻辑，无需手动管理。
- 网络隔离：**确保客户端与高性能桶在同一 VPC 内，结合安全组规则进一步限制访问来源。

常见问题

Q1：为什么高性能桶不支持对象级别的权限策略？

高性能桶面向 AI/ML 训练、HPC 等高吞吐场景，访问模式通常是批量读写整个桶的数据。桶级别的权限控制已足够满足这些场景的安全需求，同时避免了对象级别策略评估带来的额外延迟开销。

Q2：会话凭证过期后正在进行的请求会受影响吗？

已经建立连接的请求（如正在传输的大文件下载）不会因凭证过期而中断。但新的请求需要使用有效的凭证。建议在凭证过期前主动刷新。

Q3：CreateSession 本身的鉴权方式是什么？

`CreateSession` 请求使用标准的 V4 签名鉴权，与桶级别操作的鉴权方式相同。只有通过 V4 签名校验且具有 `cos:CreateSession` 权限的请求才能成功创建会话。

Q4：高性能桶是否支持匿名访问（公有读）？

不支持。高性能桶仅面向通过身份验证的用户，所有访问必须通过有效的 CAM 身份凭证。

API 文档

简介

最近更新时间：2026-10-21 18:34:03

Rapid Bucket 提供一套与腾讯云 COS 兼容的 S3 RESTful API，用于进行存储桶（Bucket）和对象（Object）的管理与读写。本文档面向直接使用 HTTP/HTTPS 调用 Rapid Bucket 接口的开发者，描述请求/响应格式以及错误码。

术语信息

使用 API 接口时会出现一些主要概念和术语，请见下表：

名称	描述
APPID	用户访问 Rapid Bucket 服务时的唯一资源标识，用以标识资源
Bucket	存储桶，Rapid Bucket 中用于存储数据的容器
BucketName-APPID	存储桶名称的完整格式，由 BucketName 和 APPID 通过 <code>--x--</code> 拼接，例如 BucketName 为 <code>examplebucket</code> 、APPID 为 <code>1250000000</code> 时，完整名称为 <code>examplebucket-x--1250000000</code>
Object	对象，Rapid Bucket 中存储的具体文件，是存储的基本实体
ObjectKey	对象键，对象（Object）在存储桶（Bucket）中的唯一标识，请求 URL 中 <code>/</code> 之后的部分即为 ObjectKey
Region	地域信息，例如 <code>ap-guangzhou</code> 、 <code>ap-beijing</code> 等
Multipart Uploads	分块上传，Rapid Bucket 为大文件上传提供的一种分块上传模式
Object Content	上传文件的二进制内容

请求域名

请求采用 `virtual-hosted-style`，Host 必须形如：

```
<BucketName-APPID>.cosrapid.<Region>.myqcloud.com
```

例如：

```
examplebucket-x--1250000000.cosrapid.ap-guangzhou.myqcloud.com
```

其中：

字段	含义	约束
BucketName-APPID	存储桶完整名称，由 BucketName 和 APPID 通过 <code>-x--</code> 拼接	BucketName 由小写字母、数字和 <code>-</code> 组成，APPID 为数字
Region	地域简称	必须与目标存储桶所在 Region 一致，否则返回 <code>InvalidRegionName</code>

`GetService`（`ListBuckets`）接口的 Host 不带 BucketName，固定为：

```
service.cosrapid.<Region>.myqcloud.com
```

接口分类

分类	包含接口
存储桶操作	<code>GetService</code> （ <code>ListBuckets</code> ）、 <code>PutBucket</code> 、 <code>HeadBucket</code> 、 <code>DeleteBucket</code> 、 <code>Createsession</code>
存储桶策略	<code>PutBucketPolicy</code> 、 <code>GetBucketPolicy</code> 、 <code>DeleteBucketPolicy</code>
对象操作	<code>GetBucket</code> （ <code>ListObjects</code> ）、 <code>PutObject</code> 、 <code>PutObjectCopy</code> 、 <code>GetObject</code> 、 <code>HeadObject</code> 、 <code>DeleteObject</code> 、 <code>DeleteMultipleObjects</code> 、 <code>RenameObject</code>
分块上传操作	<code>InitiateMultipartUpload</code> 、 <code>UploadPart</code> 、 <code>UploadPartCopy</code> 、 <code>CompleteMultipartUpload</code> 、 <code>AbortMultipartUpload</code> 、 <code>ListMultipartUploads</code> 、 <code>ListParts</code>

调用方式

所有接口遵循以下基本规则：

- 除对象上传/下载接口的 Body 为对象原始字节流外，其余接口的请求和响应 Body 均使用 XML 编码。
- 每次请求均通过响应头部 `x-cos-request-id` 返回唯一请求 ID，用于标识本次请求。
- 请求成功时 HTTP 状态码为 `2xx`，失败时为 `4xx` 或 `5xx`，并在 Body 中返回 XML 格式的错误信息。错误码列表详见 [错误码](#) 章节。
- 通用的请求/响应头部约定详见 [公共请求头部](#)、[公共响应头部](#) 章节，单接口文档只描述其私有头部。

公共请求头部

最近更新时间：2026-10-21 18:34:03

本文档将为您介绍在使用 API 时使用到的公共请求头部（Request Header），下文提到的头部在之后的具体 API 文档中不再赘述。

请求头部列表

Header 名称	描述	类型	是否必选
Authorization	携带鉴权信息，签名格式详见 请求签名 文档	string	否
Content-Length	RFC 2616 中定义的 HTTP 请求内容长度（字节）	integer	针对 PUT 和 POST 请求（指定 Transfer-Encoding 请求头部的 PUT Object 请求除外），强烈建议携带；针对 GET、HEAD、DELETE 和 OPTIONS 请求，不能指定此头部
Content-Type	RFC 2616 中定义的 HTTP 请求内容类型（MIME），例如 application/xml 或 image/jpeg。对于 PutObject、PutObjectCopy、InitiateMultipartUpload 等对象写入接口，该头部值会被作为对象元数据持久化保存，并在后续 GetObject、HeadObject 响应中原样返回	string	针对有请求体的 PUT 和 POST 请求，此头部为必选项；针对 GET、HEAD、DELETE 和 OPTIONS 请求，不能指定此头部
Content-MD5	RFC 1864 中定义的请求体内容的 16 字节（128 位）二进制 MD5 哈希值的 Base64 编码形式，用于完整性检查，最终取值长度应为 24 个字符，例如 ZzD3iDJdrMAAb00lgLLeig==。客户端可选携带；当请求中携带该头部时，服务端会比对请求体实际 MD5，不一致时返回 BadDigest	string	否；针对 GET、HEAD、DELETE 和 OPTIONS 请求，不能指定此头部
Date	RFC 1123 中定义的 GMT 格式当前时间，例如 Wed, 29 May 2019 04:10:12 GMT	string	否

Host	请求的主机，形式为 <code><BucketName-APPID>.cosrapid.<Region>.myqcloud.com</code> ； GetService 接口固定为 <code>service.cosrapid.<Region>.myqcloud.com</code>	string	是
------	--	--------	---

公共响应头部

最近更新时间：2026-10-21 18:34:03

本文将为您介绍在使用 API 时会出现的公共响应头部（Response Header），下文提到的头部在后续的具体 API 文档中不再赘述。

响应头部列表

Header 名称	描述	类型	是否必返
Content-Length	RFC 2616 中定义的 HTTP 响应内容长度（字节）。	string	视响应而定
Content-Type	RFC 2616 中定义的 HTTP 响应内容类型（MIME）。	string	视响应而定
Connection	RFC 2616 中定义，表明响应完成后是否会关闭网络连接。枚举值： <code>keep-alive</code> 、 <code>close</code> 。	enum	是
Date	RFC 1123 中定义的 GMT 格式服务端响应时间，例如 <code>Wed, 29 May 2019 04:10:12 GMT</code> 。	string	是
Server	接收请求并返回响应的服务器名称，固定为 <code>Rapid Bucket</code> 。	string	是
ETag	全称为 Entity Tag，是对象被创建时标识对象内容的信息标签，可用于检查对象的内容是否发生变化。格式为 <code>"<etag_value>"</code> ，前后携带双引号，中间为 ASCII 字符串，例如 <code>"ee8de918d05640145b18f70f4c3aa602"</code> 。仅对象相关接口（GetObject、HeadObject、PutObject、PutObjectCopy、UploadPart、UploadPartCopy、CompleteMultipartUpload）的成功响应会返回该头部。	string	否
Last-Modified	对象最近一次被修改的时间，格式为 RFC 1123 GMT，例如 <code>Tue, 28 May 2019 08:30:00 GMT</code> 。是否返回视具体接口而定。	string	否
x-cos-hash-crc64ecma	对象内容的 CRC64-ECMA 校验值，十进制字符串。仅文件对象会返回；目录对象（Key 以 <code>/</code> 结尾）不返回。	string	否

x-cos-request-id	服务端为每次请求生成的唯一 ID。无论请求成功还是失败，响应中都会携带；排查问题时请提供此 ID。	string	是
------------------	---	--------	---

错误码

最近更新时间：2026-10-21 18:34:03

错误响应

请求失败时，HTTP 响应状态码为 4xx 或 5xx，响应 Body 为 XML 格式的错误描述：

```
<?xml version='1.0' encoding='utf-8'?>
<Error>
  <Code>NoSuchBucket</Code>
  <Message>The specified bucket does not exist.</Message>
  <Resource></Resource>
  <RequestId>NjUwMDAwMDAwMDAwMDAw</RequestId>
  <TraceId></TraceId>
</Error>
```

字段说明：

节点名称	父节点	描述	类型
Error	无	包含所有的错误信息	Container
Code	Error	错误码，用来定位唯一的错误条件和确定错误场景，具体取值参见下方《错误码列表》一节	string
Message	Error	具体的错误信息。同一 ErrorCode 在不同上下文下的 Message 可能不同	string
Resource	Error	请求的资源	string
RequestId	Error	请求 ID，与响应头部 x-cos-request-id 一致；遇到问题时该 ID 能帮助快速定位	string
TraceId	Error	错误 Trace ID，遇到问题时与 RequestId 一同提供能帮助快速定位	string

错误码列表

下表汇总了 Rapid Bucket 对外可能返回的全部错误码。

4XX 类型错误

HTTP 状态码	错误码	描述
400 Bad Request	AnonymousPolicyFull	存储桶上匿名访问策略数量达到上限
400 Bad Request	AttachmentFull	被授权的账号已附加策略数量达到上限
400 Bad Request	BadDigest	客户端提供的 <code>Content-MD5</code> 值与服务端收到的请求体的 MD5 哈希值不一致
400 Bad Request	DirectoryObjectContainsData	目录对象（Key 以 <code>/</code> 结尾）携带了数据 Body
400 Bad Request	EntityTooLarge	上传的对象大小超过接口允许的最大值
400 Bad Request	EntityTooSmall	上传的对象大小不足接口允许的最小值，常见于分块上传中除最后一块以外的分块
400 Bad Request	ExpiredToken	会话临时凭证已过期，需重新调用 <code>CreateSession</code> 获取新凭证后重试（与 <code>InvalidAccessKeyId</code> 区分：后者是凭证不存在）
400 Bad Request	IncompleteBody	实际接收到的请求体字节数小于 <code>Content-Length</code> 请求头声明的值
400 Bad Request	InvalidArgument	请求参数取值不合法
400 Bad Request	InvalidBucketName	存储桶名称不合法
400 Bad Request	InvalidCopySource	复制对象源不合法
400 Bad Request	InvalidHostName	请求 Host 不符合 <code><BucketName-APPID>.cosrapid.<Region>.myqcloud.com</code> 形式
400 Bad Request	InvalidObjectName	对象名称不合法
400 Bad Request	InvalidPart	<code>CompleteMultipartUpload</code> 中指定的某个分块不存在，或其 ETag 与服务端记录不一致

400 Bad Request	InvalidPartOrder	CompleteMultipartUpload 中分块未按 PartNumber 升序排列
400 Bad Request	InvalidRegionName	Host 中的 Region 与目标存储桶所在 Region 不匹配
400 Bad Request	InvalidRenameSource	<code>x-cos-rename-source</code> 头部指定的源不合法
400 Bad Request	InvalidRequest	请求不合法（例如 Body 格式错误）
400 Bad Request	InvalidURI	URI 不合法，无法解析
400 Bad Request	MissingRequestBody	请求体缺失，但接口要求必须携带
400 Bad Request	NotSupportedStorageClass	指定的存储类型不支持，Rapid Bucket 仅支持 <code>RAPID</code>
400 Bad Request	PolicyFull	存储桶上策略数量达到上限
400 Bad Request	RequestTimeout	请求超时
403 Forbidden	AccessDenied	鉴权失败、签名过期或调用方对目标资源无权限
403 Forbidden	InvalidAccessKeyId	请求中引用的 Access Key Id 不存在
403 Forbidden	SignatureDoesNotMatch	请求签名与服务端计算结果不一致
404 Not Found	NoSuchBucket	指定的存储桶不存在
404 Not Found	NoSuchKey	指定的对象不存在
404 Not Found	NoSuchPolicy	存储桶未配置策略
404 Not Found	NoSuchUpload	分块上传时指定的 UploadId 不存在，或对应的分块上传已完成、已中止

405 Method Not Allowed	MethodNotAllowed	请求方法不被目标资源支持
409 Conflict	BucketAlreadyExists	创建存储桶时同名存储桶已存在
409 Conflict	BucketCreating	存储桶正在创建中，资源尚未就绪
409 Conflict	BucketDeleting	存储桶正在删除中，资源正在清理
409 Conflict	BucketNotEmpty	删除存储桶时，桶内仍存在对象
409 Conflict	BucketUpdating	存储桶正在更新中，资源状态变更尚未完成
409 Conflict	DirAlreadyExists	待写入路径已存在同名目录对象
409 Conflict	DirNotEmpty	删除的目录对象下还存在子对象
409 Conflict	FileAlreadyExists	上传对象时使用了 <code>x-cos-forbid-overwrite: true</code> 头部，但目标对象已存在
416 Requested Range Not Satisfiable	InvalidRange	请求的对象范围不合法（ <code>Range</code> 头部格式错误，或指定的字节范围超出对象大小）

5XX 类型错误

HTTP 状态码	错误码	描述
500 Internal Server Error	InternalServerError	服务端内部错误，请重试或联系管理员
501 Not Implemented	NotImplemented	请求中使用了 Rapid Bucket 当前不支持的功能
503 Service Unavailable	ServiceUnavailable	服务端暂时不可用，请稍后重试
503 Service Unavailable	SlowDown	请求频率过高，请降低调用频率后重试

存储桶操作

GetService

最近更新时间：2026-10-21 18:34:04

GetService 接口请求等同于 ListBuckets，可以列出调用方主账号下的全部存储桶。

注意事项

分页

当 `IsTruncated` 为 `true` 时，将响应中的 `NextMarker` 原样作为下次请求的 `marker` 传入，直到 `IsTruncated` 为 `false`。

请求

请求语法

```
GET /?max-keys=<MaxKeys>&marker=<Marker> HTTP/1.1
Host: service.cosrapid.<Region>.myqcloud.com
Date: <GMT Date>
Authorization: <Auth String>
```

说明：

- Authorization 字段的具体格式见 [公共请求头部](#)。
- Host 固定为 `service.cosrapid.<Region>.myqcloud.com`，不带存储桶名前缀；`<Region>` 指定本次要列举的存储桶所在地域，本接口仅返回该 Region 下的存储桶；如需查看其他 Region 下的存储桶，请分别调用。

请求参数

参数名称	描述	类型	是否必选
max-keys	单次返回的最大存储桶数量，取值范围 1 ~ 2000，默认值 2000	integer	否
marker	分页游标，取值为上一次 GetService 响应中的 NextMarker；首次请求无需携带	string	否

请求头部

此接口仅使用 [公共请求头部](#)，无私有请求头部。

请求体

该请求的请求体为空。

响应

响应头部

仅返回 [公共响应头部](#)，本接口无私有响应头部。

响应体

请求成功，返回 **application/xml** 数据：

```
<ListAllMyBucketsResult>
  <Owner>
    <ID>qcs::cam::uin/<OwnerUin>:uin/<OwnerUin></ID>
    <DisplayName><OwnerUin></DisplayName>
  </Owner>
  <Marker>string</Marker>
  <NextMarker>string</NextMarker>
  <IsTruncated>true|false</IsTruncated>
  <Buckets>
    <Bucket>
      <Name>string</Name>
      <Location>string</Location>
      <CreationDate>date</CreationDate>
      <VpcId>string</VpcId>
      <SubnetId>string</SubnetId>
      <Zone>string</Zone>
      <Status>string</Status>
    </Bucket>
    ...
  </Buckets>
</ListAllMyBucketsResult>
```

字段说明：

节点名称	父节点	描述	类型
------	-----	----	----

ListAllMyBucketsResult	无	响应根节点，包含 GetService 的全部结果	Container
Owner	ListAllMyBucketsResult	存储桶持有者信息	Container
Marker	ListAllMyBucketsResult	本次请求所使用的分页游标，回显请求参数 marker ；未指定时为空	string
NextMarker	ListAllMyBucketsResult	下一页的分页游标，仅当 IsTruncated 为 true 时返回非空	string
IsTruncated	ListAllMyBucketsResult	是否仍有未列举完的存储桶： true 表示存在下一页， false 表示已全部列举完毕	boolean
Buckets	ListAllMyBucketsResult	存储桶列表容器	Container

Owner 节点：

节点名称	父节点	描述	类型
ID	Owner	存储桶持有者完整 ID，格式为 qcs::cam::uin/<OwnerUin>:uin/<OwnerUin>	string
DisplayName	Owner	存储桶持有者主账号 UIN	string

Buckets.Bucket 节点：

节点名称	父节点	描述	类型
Name	Buckets.Bucket	存储桶完整名称，格式为 <BucketName>-x--<APPID> ，例如 examplebucket-x--1250000000	string
Location	Buckets.Bucket	存储桶所在地域，例如 ap-guangzhou	string

CreationDate	Buckets. Bucket	存储桶的创建时间，ISO 8601 格式（RFC3339），例如 2023-10-13T09:13:10Z	date
VpcId	Buckets. Bucket	存储桶绑定的 VPC ID，回显创建该桶时通过 x-cos-vpc-id 传入的值	string
SubnetId	Buckets. Bucket	存储桶绑定的子网 ID，回显创建该桶时通过 x-cos-subnet-id 传入的值	string
Zone	Buckets. Bucket	存储桶所在可用区，回显创建该桶时通过 x-cos-zone 传入的值，例如 ap-guangzhou-3	string
Status	Buckets. Bucket	存储桶状态（Creating：创建中；Available：正常；Deleting：删除中）	string

错误码

本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

响应

```
<DisplayName>100000000001</DisplayName>
</Owner>
<Marker></Marker>
<NextMarker></NextMarker>
<IsTruncated>false</IsTruncated>
<Buckets>
  <Bucket>
<Name>examplebucket1-x--1250000000</Name>
    <Location>ap-guangzhou</Location>
    <CreationDate>2023-10-12T11:49:50Z</CreationDate>
    <VpcId>vpc-abcdefgh</VpcId>
    <SubnetId>subnet-abcdefgh</SubnetId>
    <Zone>ap-guangzhou-3</Zone>
    <Status>Available</Status>
  </Bucket>
  <Bucket>
<Name>examplebucket2-x--1250000000</Name>
    <Location>ap-guangzhou</Location>
    <CreationDate>2023-10-13T09:13:10Z</CreationDate>
    <VpcId>vpc-ijklmnop</VpcId>
    <SubnetId>subnet-ijklmnop</SubnetId>
    <Zone>ap-guangzhou-4</Zone>
    <Status>Creating</Status>
  </Bucket>
</Buckets>
</ListAllMyBucketsResult>
```


CreateSession

最近更新时间：2026-10-21 18:34:04

CreateSession 接口请求用于为高性能桶创建访问会话，返回可用于后续数据面请求的临时凭证三元组。客户端使用返回的临时 SecretId、SecretKey 和 Token 对高性能桶对象接口发起签名请求。

请求

请求语法

```
GET /?session HTTP/1.1
Host: <BucketName>-x--<APPID>.cosrapid.<Region>.myqcloud.com
Date: <GMT Date>
Authorization: <Auth String>
x-cos-create-session-mode: <SessionMode>
```

说明：
Authorization 字段的具体格式见 [公共请求头部](#)。

请求参数

名称	描述	类型	是否必选
session	标识本次请求为创建访问会话请求。该参数仅作为子资源标识，无需填写参数值	string	是

请求头部

除 [公共请求头部](#) 外，本接口还支持以下私有请求头部：

名称	描述	类型	是否必选
x-cos-create-session	创建会话的权限模式，取值如下： - ReadOnly：只读会话，允许 GetObject、HeadObject、GetBucket、ListMultipartUploads、ListParts 等读类操作。 - ReadWrite：读写会话，允许只读会话的全部操作，并额外允许 PutObject、DeleteObject、DeleteMultipleObjects、InitiateMultipartUpload等写类操作。	Enum	否

ion-mod e	<p>artUpload、UploadPart、CompleteMultipartUpload、AbortMultipartUpload、PutObjectCopy、UploadPartCopy、RenameObject 等写类操作。</p> <p>不携带该请求头时，服务端按最大权限模式申请会话，等价于申请 ReadWrite。若桶策略或 CAM 策略通过 cos:SessionMode 条件限制了会话模式，本请求头取值必须与策略条件匹配，否则将返回 AccessDenied。</p>		
--------------	---	--	--

请求体

该请求的请求体为空。

响应

响应头部

此接口仅返回 公共响应头部，无私有响应头部。

响应体

请求成功时，响应体为 XML 格式，包含临时凭证信息：

节点名称	父节点	描述	类型
CreateSessionResult	无	响应体根节点	Container
Credentials	CreateSessionResult	临时凭证信息	Container
AccessKeyId	Credentials	临时凭证 AccessKeyId（对应临时 SecretId）	string
SecretAccessKey	Credentials	临时凭证 SecretAccessKey（对应临时 SecretKey）	string
SessionToken	Credentials	临时凭证 Token。后续使用临时凭证访问高性能桶对象接口时，需要通过 x-cos-security-token 携带该 Token	string
Expiration	Credentials	临时凭证过期时间，ISO 8601 UTC 时间格式	string

响应体示例：

错误码

实际案例

请求

响应

第39 共324页

```
<SessionToken>TOKENEXAMPLE</SessionToken>  
<Expiration>2026-07-24T12:00:00Z</Expiration>  
</Credentials>  
</CreateSessionResult>
```

PutBucket

最近更新时间：2026-10-21 18:34:04

PutBucket 接口请求用于在指定地域下创建一个 Rapid Bucket。该接口不支持匿名请求，调用方必须使用带签名的请求；创建成功后，调用方所属主账号即为存储桶持有者。

注意事项

存储桶命名

通过本接口创建的存储桶，其完整 BucketName 必须为 `<BucketName>-x--<APPID>` 格式，例如 `example-bucket-x--1250000000`。桶名不符合该格式时，接口将返回 `InvalidBucketName`。

同名存储桶已存在

请求指向的存储桶已存在时，按其当前状态返回对应的 HTTP 409 错误：

- 存储桶可用：返回 `BucketAlreadyExists`。
- 存储桶正在创建中：返回 `BucketCreating`。
- 存储桶正在删除中：返回 `BucketDeleting`。

请求

请求端点

创建高性能桶时，请求的连接端点与 `Host` 头指向不同的域名，需分别设置：

项目	取值	说明
连接端点（Endpoint）	<code>service.cosrapid.<Region>.myqcloud.com</code>	请求实际建立连接、发送的目标地址（service 域名）
<code>Host</code> 头	<code><BucketName>-x--<APPID>.cosrapid.<Region>.myqcloud.com</code>	标识要创建的目标桶，服务端据此解析桶名与 APPID

① 说明：

创建桶请求统一发往 `service` 域名建立连接，而 `Host` 头需独立填写为目标桶域名。二者不一致是正常且必需的：服务端从 `Host` 头解析出待创建的 `BucketName` 与 `APPID`，若 `Host` 头未按桶域名格式填写，将返回 `InvalidBucketName`。

请求语法

```
PUT / HTTP/1.1
# 连接端点为 service 域名; Host 头独立填目标桶域名
Host: <BucketName>-x--<APPID>.cosrapid.<Region>.myqcloud.com
Date: <GMT Date>
x-cos-vpc-id: <VpcId>
x-cos-subnet-id: <SubnetId>
x-cos-zone: <Zone>
Content-Length: 0
Authorization: <Auth String>
```

说明：
Authorization 字段的具体格式见 [公共请求头部](#)。

请求参数

此接口无请求参数。

请求头部

除 [公共请求头部](#) 外，本接口还使用以下私有请求头部：

名称	描述	类型	是否必选
x-cos-vpc-id	存储桶绑定的 VPC ID	string	是
x-cos-subnet-id	存储桶绑定的子网 ID	string	是
x-cos-zone	存储桶所在可用区，例如 ap-guangzhou-3	string	是

说明：
上述三个头部任一缺失或取值为空时，接口返回 `InvalidArgument`。

请求体

该请求的请求体为空。

响应

响应头部

仅返回 [公共响应头部](#)，本接口无私有响应头部。


```
Content-Length: 0
Date: Fri, 13 Oct 2023 09:13:10 GMT
x-cos-request-id: NjUwMDAwMDAwMDAwMDAwMDAw
```


DeleteBucket

最近更新时间：2026-10-21 18:34:04

DeleteBucket 接口请求用于删除指定的存储桶。

注意事项

- 删除存储桶前，请确保存储桶内的对象（含目录对象）已全部清空，否则接口返回 `409 Conflict`，错误码为 `BucketNotEmpty`。
- 删除目标存储桶不存在时，接口仍按删除成功处理，返回 `204 No Content`。

请求

请求语法

```
DELETE / HTTP/1.1
Host: <BucketName>-x--<APPID>.cosrapid.<Region>.myqcloud.com
Date: <GMT Date>
Authorization: <Auth String>
```

说明：
Authorization 字段的具体格式见 [公共请求头部](#)。

请求参数

此接口无请求参数。

请求头部

此接口仅使用 [公共请求头部](#)，无私有请求头部。

请求体

该请求的请求体为空。

响应

响应头部

仅返回 [公共响应头部](#)，本接口无私有响应头部。

响应体

错误码

实际案例

请求

响应

```
HTTP/1.1 204 No Content
Content-Length: 0
Date: Fri, 13 Oct 2023 09:13:10 GMT
x-cos-request-id: NjUwMDAwMDAwMDAwMDAwMDAw
```

HeadBucket

最近更新时间：2026-10-21 18:34:05

HeadBucket 接口请求用于确认存储桶是否存在以及调用方是否有权限访问，返回结果以 HTTP 状态码及响应头部呈现，无响应体。

注意事项

- 存储桶存在且调用方有访问权限：返回 200 OK。
- 调用方对该存储桶无访问权限：返回 403 Forbidden。
- 存储桶不存在：返回 404 Not Found。

请求

请求语法

```
HEAD / HTTP/1.1
Host: <BucketName>-x--<APPID>.cosrapid.<Region>.myqcloud.com
Date: <GMT Date>
Authorization: <Auth String>
```

① 说明：

Authorization 字段的具体格式见 [公共请求头部](#)。

请求参数

此接口无请求参数。

请求头部

此接口仅使用 [公共请求头部](#)，无私有请求头部。

请求体

该请求的请求体为空。

响应

响应头部

除 [公共响应头部](#) 外，请求成功时本接口还返回以下私有响应头部：

名称	描述	类型
x-cos-bucket-status	存储桶状态，取值为 <code>Creating</code> 、 <code>Available</code> 、 <code>Deleting</code> ，分别表示存储桶处于创建中、可用、删除中	Enum
x-cos-fusionfs-id	存储桶关联的 CFS Turbo 文件系统 ID	string
x-cos-owner-uin	存储桶所有者主账号 UIN	string
x-cos-vpc-id	存储桶绑定的 VPC ID	string
x-cos-subnet-id	存储桶绑定的子网 ID	string
x-cos-zone	存储桶所在可用区，例如 <code>ap-guangzhou-3</code>	string
x-cos-creation-date	存储桶的创建时间，ISO 8601 格式（RFC3339），例如 <code>2023-10-13T09:13:10Z</code>	date

响应体

该响应体为空。

错误码

本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

响应

```
HTTP/1.1 200 OK
Content-Length: 0
Date: Fri, 13 Oct 2023 09:13:10 GMT
x-cos-request-id: NjUwMDAwMDAwMDAwMDAwMDAw
x-cos-bucket-status: Available
x-cos-fusionfs-id: fsxxxxxxx
x-cos-owner-uin: 1000000000001
x-cos-vpc-id: vpc-abcdefgh
x-cos-subnet-id: subnet-abcdefgh
x-cos-zone: ap-guangzhou-3
x-cos-creation-date: 2023-10-13T09:13:10Z
```

存储桶策略

PutBucketPolicy

最近更新时间：2026-10-21 18:34:04

PutBucketPolicy 接口请求用于向存储桶写入授权策略。授权策略由若干条 statement 组成，描述允许或拒绝哪些账号对该存储桶执行哪些操作。

每次调用本接口将整体覆盖该存储桶上已存在的全部策略，请求体需提交完整的目标策略集合。

请求

请求语法

```
PUT /?policy HTTP/1.1
Host: <BucketName>-x--<APPID>.cosrapid.<Region>.myqcloud.com
Date: <GMT Date>
Content-Type: application/json
Content-Length: <Length>
Authorization: <Auth String>

<Policy JSON Body>
```

① 说明：

Authorization 字段的具体格式见 [公共请求头部](#)。

请求参数

参数名称	描述	类型	是否必选
policy	用于声明本次请求操作的是存储桶策略，固定为空串。	string	是

请求头部

此接口仅使用 [公共请求头部](#)，无私有请求头部。请求体为 JSON，需将 Content-Type 设置为 application/json。

请求体

请求体为 JSON 文本，结构如下：

```
{
  "version": "2.0",
  "statement": [
    {
      "sid": "string",
      "effect": "allow",
      "action": ["name/cos:GetObject"],
      "resource": [
        "qcs::cosrapid:<Region>:uid/<APPID>:<BucketName>-x--<APPID>/*"
      ],
      "principal": {
        "qcs": [
          "qcs::cam::uin/<OwnerUin>:uin/<SubUin>"
        ]
      },
      "condition": {}
    }
  ]
}
```

字段说明：

节点名称	父节点	描述	类型	是否必选
version	无	策略版本号，固定为 2.0	string	是
statement	无	策略 statement 数组，描述本次写入的全部授权条目	Container 数组	是
sid	statement	statement 的标识符。同一存储桶下用于区分不同条目；省略时由服务端自动生成	string	否
effect	statement	授权效果，取值为 allow 或 deny	string	是
action	statement	被授权的操作列表，元素形如 name/cos:<ActionName>，例如 name/cos:GetObject 、 name/cos:PutOb	string 数组	是

		ject、name/cos:HeadBucket 等		
resource	statement	被授权的资源列表。每个元素必须形如 qcs::cosrapid: <Region>:uid/<APPID>:<BucketName>-x--<APPID >/*, 且 <Region> 和 <BucketName>-x--<APPID > 必须与请求 Host 中的存储桶完全一致, 否则返回 InvalidArgument	string 数组	是
principal	statement	被授权的账号。键为 qcs, 值的元素形如 qcs::cam:: uin/<OwnerUin>:uin/<SubUin>, 表示腾讯云账号 (含子账号); 可使用 anyone 替换 <SubUin> 部分 以表示匿名访问	map[string] []string	是
condition	statement	生效条件, 键为条件运算符, 值为条件键到取值列表的映射; 留空表示无附加条件	map	否

响应

响应头部

仅返回 **公共响应头部**，本接口无私有响应头部。

响应体

请求成功，响应体为空。

错误码

本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

响应

版权所有：腾讯云计算（北京）有限责任公司

GetBucketPolicy

最近更新时间：2026-10-21 18:34:04

GetBucketPolicy 接口请求用于读取存储桶上当前生效的全部授权策略。响应以 JSON 格式返回，结构与 [PutBucketPolicy](#) 请求体一致。

请求

请求语法

```
GET /?policy HTTP/1.1
Host: <BucketName>-x--<APPID>.cosrapid.<Region>.myqcloud.com
Date: <GMT Date>
Authorization: <Auth String>
```

说明：

Authorization 字段的具体格式见 [公共请求头部](#)。

请求参数

参数名称	描述	类型	是否必选
policy	用于声明本次请求操作的是存储桶策略。该参数无需取值，以 <code>?policy</code> 形式附加在请求 URL 中，见请求语法。	string	是

请求头部

此接口仅使用 [公共请求头部](#)，无私有请求头部。

请求体

该请求的请求体为空。

响应

响应头部

除 [公共响应头部](#) 外，本接口的响应头部 `Content-Type` 固定为 `application/json`。

响应体

请求成功，返回 `application/json` 数据。

```
{
  "Version": "2.0",
  "Statement": [
    {
      "Sid": "string",
      "Effect": "allow",
      "Action": ["name/cos:GetObject"],
      "Resource": [
        "qcs::cosrapid:<Region>:uid/<APPID>:<BucketName>-x--<APPID>/*"
      ],
      "Principal": {
        "qcs": [
          "qcs::cam::uin/<OwnerUin>:uin/<SubUin>"
        ]
      },
      "Condition": {}
    }
  ]
}
```

响应体结构与 [PutBucketPolicy](#) 请求体一致。存储桶上没有配置任何策略时，本接口返回 `404 Not Found`，错误码为 `NoSuchPolicy`，响应体为标准错误 XML（`<Error>` 中 `Code` 为 `NoSuchPolicy`），而非返回 `statement` 为空数组的成功响应。各字段含义详见 [PutBucketPolicy](#) 请求体说明。

错误码

本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 /?policy HTTP/1.1
Host: examplebucket-x--1250000000.cosrapid.ap-guangzhou.myqcloud.com
Date: Fri, 13 Oct 2023 09:13:10 GMT
Authorization: q-sign-algorithm=sha1&q-ak=AKIDXXXXXXXXX&q-sign-time=1484814099;32557710099&q-key-time=1484814099;32557710099&q-header-
```

响应

第56 共324页

DeleteBucketPolicy

最近更新时间：2026-10-21 18:34:05

DeleteBucketPolicy 接口请求用于清除存储桶上当前配置的全部授权策略。

请求

请求语法

```
DELETE /?policy HTTP/1.1
Host: <BucketName>-x--<APPID>.cosrapid.<Region>.myqcloud.com
Date: <GMT Date>
Authorization: <Auth String>
```

说明：
Authorization 字段的具体格式见 [公共请求头部](#)。

请求参数

参数名称	描述	类型	是否必选
policy	用于声明本次请求操作的是存储桶策略，固定为空串。	string	是

请求头部

此接口仅使用 [公共请求头部](#)，无私有请求头部。

请求体

该请求的请求体为空。

响应

响应头部

仅返回 [公共响应头部](#)，本接口无私有响应头部。

响应体

请求成功，响应状态码为 204 No Content ，响应体为空。

错误码

实际案例

请求

响应

第58 共324页

对象操作

PutObject

最近更新时间：2026-10-21 18:34:05

PutObject 接口请求可以将本地的对象（Object）上传至指定存储桶（Bucket）中。

注意事项

文件大小限制

PutObject 接口最大支持上传 5GB 文件。如需上传大于 5GB 的文件，请使用 [分块上传](#) 系列 API 接口。

文件对象与目录对象

Rapid Bucket 区分文件对象与目录对象，以对象键是否以 `/` 结尾区分：对象键不以 `/` 结尾表示文件对象，以 `/` 结尾表示目录对象。两类对象在请求约束、可携带的头部以及响应内容上存在差异。

文件对象

对不以 `/` 结尾的对象键调用 PutObject 时：

- 例如 `doc/picture.png` 表示将 `picture.png` 上传到 `doc` 目录下。若 `doc` 目录不存在，服务会自动创建。
- 请求体即为对象的原始字节流。
- 响应会返回 `ETag`、`Last-Modified` 以及 `x-cos-hash-crc64ecma`。

目录对象

对以 `/` 结尾的对象键调用 PutObject 时：

- 例如 `doc/` 表示创建名为 `doc` 的目录。
- `Content-Length` 必须为 `0`，否则返回 `DirectoryObjectContainsData`。
- 不支持携带 `x-cos-meta-*`、`Cache-Control`、`Content-Disposition`、`Content-Encoding`、`Content-Language`、`Content-Type`、`Expires` 等元数据头部，否则返回 `InvalidArgument`。
- 若携带 `Content-MD5` 头部，其值必须为空内容的 MD5 Base64 值（`1B2M2Y8AsgTpgAmY7PhCfg=`），否则返回 `BadDigest`。
- 响应仅返回 `ETag` 和 `Last-Modified`，不返回 `x-cos-hash-crc64ecma`。

同名冲突与覆盖规则

文件对象与目录对象不支持同名共存，也不支持跨类型覆盖，具体规则如下：

1. 同名文件之间默认覆盖：上传同名文件对象时默认覆盖原有对象，返回 `200 OK`。如需禁止覆盖，可携带 `x-cos-forbid-overwrite: true`，此时目标对象已存在则返回 `409 FileAlreadyExists`。

2. 文件不能覆盖同名目录：上传 `foo` 时，若 `foo/` 已作为目录对象存在，请求返回 `409 DirAlreadyExists`。该限制与 `x-cos-forbid-overwrite` 无关。
3. 目录不能覆盖同名文件：上传 `foo/` 时，若 `foo` 已作为文件对象存在，请求返回 `409 FileAlreadyExists`。
4. 路径中间组件必须是目录：上传 `a/b/c` 时，若 `a/b` 已作为文件对象存在，请求返回 `409 FileAlreadyExists`。

请求

请求语法

```
PUT /<ObjectKey> HTTP/1.1
Host: <BucketName-APPID>.cosrapid.<Region>.myqcloud.com
Date: <GMT Date>
Content-Type: <Content Type>
Content-Length: <Content Length>
Content-MD5: <MD5>
Authorization: <Auth String>

[Object Content]
```

① 说明：

Authorization 字段的具体格式见 [公共请求头部](#)。

请求参数

此接口无请求参数。

请求头部

除 [公共请求头部](#) 外，本接口还支持以下请求头部：

常规请求头部

Header 名称	描述	类型	是否必选
Cache-Control	RFC 2616 中定义的缓存指令，将作为对象元数据保存	string	否
Content-	RFC 2616 中定义的文件名称，将作为对象元数据保存	string	否

Disposition			
Content-Encoding	RFC 2616 中定义的编码格式，将作为对象元数据保存	string	否
Content-Language	RFC 2616 中定义的内容语言，将作为对象元数据保存	string	否
Content-Type	RFC 2616 中定义的 HTTP 请求内容类型（MIME），用于描述待上传对象的内容类型，将作为对象元数据保存。例如 application/xml 或 image/jpeg 。文件对象不支持指定为 application/x-directory ，否则返回 InvalidArgument	string	否
Expires	RFC 2616 中定义的缓存失效时间，是绝对日期和时间，将作为对象元数据保存	string	否
Transfer-Encoding	RFC 2616 中定义的传输编码格式。如果希望在上传时分块传输，则指定 Transfer-Encoding: chunked 请求头部，并不再指定 Content-Length 请求头部	string	否

专用请求头部

Header 名称	描述	类型	是否必选
x-cos-meta-[自定义后缀]	用户自定义元数据头部，例如 x-cos-meta-test: test metadata 。自定义后缀仅支持减号（-）、数字、英文小写字母（a~z）	string	否
x-cos-storage-class	对象存储类型，当前仅支持 RAPID （默认值）。指定其他取值时返回 NotSupportedStorageClass	enum	否
x-cos-forbid-overwrite	是否禁止覆盖同名文件对象，取值为 true 时启用。行为契约详见「同名冲突与覆盖规则」	string	否

请求体

该请求的请求体（Body）为待上传对象的原始字节流。

响应

响应头部

除 [公共响应头部](#) 外，本接口还会返回以下响应头部：

Header 名称	描述	类型	是否必返
ETag	对象的 MD5 校验值，使用双引号包裹。客户端可据此验证上传完整性	string	是
Last-Modified	对象最近一次被修改的时间，格式为 RFC 1123 GMT，例如 <code>Wed, 29 May 2019 04:10:12 GMT</code>	string	是
x-cos-hash-crc64ecma	对象的 CRC64-ECMA 校验值，十进制字符串。仅文件对象返回，目录对象不返回	string	否

响应体

该接口的响应体为空。

错误码

本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

案例一：上传文件对象

请求

```
PUT /exampleobject HTTP/1.1
Host: examplebucket-x--1250000000.cosrapid.ap-guangzhou.myqcloud.com
Date: Wed, 29 May 2019 04:10:12 GMT
Content-Type: image/jpeg
```

响应

案例二：上传带用户自定义元数据的对象

请求

版权所有：腾讯云计算（北京）有限责任公司

案例三：使用 chunked 传输编码上传

请求

响应

```
HTTP/1.1 200 OK
Content-Length: 0
Connection: keep-alive
```

```
Date: Wed, 29 May 2019 04:10:12 GMT
ETag: "aa488bb80185a6be87f4a7b936a80752"
Last-Modified: Wed, 29 May 2019 04:10:12 GMT
x-cos-hash-crc64ecma: 7188322482464764960
x-cos-request-id: NjUwMDAwMDAwMDAwMDAwMDAw
```

案例四：使用 x-cos-forbid-overwrite 防覆盖

本案例在上传对象请求中携带了 `x-cos-forbid-overwrite: true`，表示禁止覆盖同名对象。

请求

```
PUT /exampleobject HTTP/1.1
x-cos-forbid-overwrite: true
Host: examplebucket-x--1250000000.cosrapid.ap-guangzhou.myqcloud.com
Date: Wed, 29 May 2019 04:10:12 GMT
Content-Type: image/jpeg
Content-Length: 16
Authorization: q-sign-algorithm=sha1&q-ak=AKIDXXXXXXXXX&q-sign-
time=1559074212;1559077812&q-key-time=1559074212;1559077812&q-header-
list=content-length;content-type;date;host;x-cos-forbid-overwrite&q-url-
param-list=&q-signature=xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
[Object Content]
```

响应

如果存储桶中不存在同名对象，则上传成功：

```
HTTP/1.1 200 OK
Content-Length: 0
Connection: keep-alive
Date: Wed, 29 May 2019 04:10:12 GMT
ETag: "aa488bb80185a6be87f4a7b936a80752"
Last-Modified: Wed, 29 May 2019 04:10:12 GMT
x-cos-hash-crc64ecma: 7188322482464764960
x-cos-request-id: NjUwMDAwMDAwMDAwMDAwMDAw
```

如果存储桶中存在同名对象，则返回报错：

案例五：创建目录对象

请求

响应

第66 共324页

GetObject

最近更新时间：2026-10-21 18:34:04

GetObject 接口请求可以将存储桶（Bucket）中的一个对象（Object）下载至本地。

注意事项

对象键末尾的 / 严格区分

对象键以 / 结尾代表目录对象，不以 / 结尾代表文件对象。两者属于不同的对象，本接口不会对对象键的末尾做任何容错。例如：

- 若实际只存在文件对象 `foo`，调用 `GET /foo/` 会返回 `404 NoSuchKey`。
- 若实际只存在目录对象 `foo/`，调用 `GET /foo` 会返回 `404 NoSuchKey`。

目录对象

对以 / 结尾的目录对象调用 GetObject 时，行为如下：

- `Content-Length` 固定为 `0`，响应体为空。
- `Content-Type` 固定为 `application/x-directory`。
- 不返回 `x-cos-hash-crc64ecma` 头部。
- 不支持 `Range` 请求头部，携带时将返回 `400 InvalidArgument` 错误。

请求

请求语法

```
GET /<ObjectKey> HTTP/1.1
Host: <BucketName-APPID>.cosrapid.<Region>.myqcloud.com
Date: <GMT Date>
Authorization: <Auth String>
Range: bytes=<first>-<last> # 可选；first 或 last 可省略，如 bytes=-10、
bytes=10-
```

说明：

Authorization 字段的具体格式见 [公共请求头部](#)。

请求参数

参数名称	描述	类型	是否必选
------	----	----	------

response-cache-control	设置响应中的 <code>Cache-Control</code> 头部的值	string	否
response-content-disposition	设置响应中的 <code>Content-Disposition</code> 头部的值	string	否
response-content-encoding	设置响应中的 <code>Content-Encoding</code> 头部的值	string	否
response-content-language	设置响应中的 <code>Content-Language</code> 头部的值	string	否
response-content-type	设置响应中的 <code>Content-Type</code> 头部的值	string	否
response-expires	设置响应中的 <code>Expires</code> 头部的值	string	否

请求头部

除 [公共请求头部](#) 外，本接口还支持以下请求头部：

Header 名称	描述	类型	是否必选
Range	RFC 2616 中定义的字节范围，仅文件对象支持。格式为 <code>bytes=first-last</code> ，仅支持单一范围，不支持多重范围。 <code>first</code> 与 <code>last</code> 均为基于 0 的偏移量，示例： <code>bytes=0-0</code> 表示前 1 字节、 <code>bytes=0-9</code> 表示前 10 字节、 <code>bytes=5-9</code> 表示第 6 到第 10 字节、 <code>bytes=-10</code> 表示末尾 10 字节、 <code>bytes=10-</code> 表示第 11 字节到末尾、 <code>bytes=0-</code> 表示整个对象。指定后返回 <code>206 Partial Content</code> 及 <code>Content-Range</code> 响应头部； <code>first</code> 与 <code>last</code> 均超出对象大小时返回 <code>416 Requested Range Not Satisfiable</code>	string	否

请求体

该请求的请求体为空。

响应

响应头部

除 [公共响应头部](#) 外，本接口还会返回以下响应头部：

Header 名称	描述	类型	是否必返
Accept-Ranges	RFC 2616 中定义的字节范围请求支持情况，固定为 <code>bytes</code>	string	是
Content-Range	RFC 2616 中定义的返回内容的字节范围，仅当请求中指定了 <code>Range</code> 请求头部时才会返回该头部	string	否
Cache-Control	RFC 2616 中定义的缓存指令，仅当对象元数据包含此项或通过请求参数指定了此项时才会返回该头部	string	否
Content-Disposition	RFC 2616 中定义的文件名称，仅当对象元数据包含此项或通过请求参数指定了此项时才会返回该头部	string	否
Content-Encoding	RFC 2616 中定义的编码格式，仅当对象元数据包含此项或通过请求参数指定了此项时才会返回该头部	string	否
Content-Language	RFC 2616 中定义的内容语言，仅当对象元数据包含此项或通过请求参数指定了此项时才会返回该头部	string	否
Expires	RFC 2616 中定义的缓存失效时间，仅当对象元数据包含此项或通过请求参数指定了此项时才会返回该头部	string	否
X-COS-meta-*	包括用户自定义元数据头部后缀和用户自定义元数据信息	string	否
X-COS-storage-class	对象存储类型，当前固定为 <code>RAPID</code>	enum	是

响应体

响应体为对象的原始字节流。

错误码

本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

案例一：下载完整对象

请求

响应

案例二：通过请求参数指定响应头部

请求

第70 共324页


```
Date: Wed, 29 May 2019 04:10:12 GMT
ETag: "ee8de918d05640145b18f70f4c3aa602"
Last-Modified: Tue, 28 May 2019 08:30:00 GMT
x-cos-storage-class: RAPID
x-cos-hash-crc64ecma: 16749565679157681890
x-cos-request-id: NjUwMDAwMDAwMDAwMDAwMDAwMDAwMDAw
```

Content

HeadObject

最近更新时间：2026-10-21 18:34:04

HeadObject 接口请求可以判断指定对象（Object）是否存在，并返回该对象的元数据。该接口仅返回响应头部，不返回响应体。

注意事项

对象键末尾的 / 严格区分

对象键以 / 结尾代表目录对象，不以 / 结尾代表文件对象。两者属于不同的对象，本接口不会对对象键的末尾做任何容错。例如：

- 若实际只存在文件对象 `foo`，调用 `HEAD /foo/` 会返回 `404 NoSuchKey`。
- 若实际只存在目录对象 `foo/`，调用 `HEAD /foo` 会返回 `404 NoSuchKey`。

目录对象

对以 / 结尾的目录对象调用 HeadObject 时，行为如下：

- `Content-Length` 固定为 `0`。
- `Content-Type` 固定为 `application/x-directory`。
- 不返回 `x-cos-hash-crc64ecma` 头部。

请求

请求语法

```
HEAD /<ObjectKey> HTTP/1.1
Host: <BucketName-APPID>.cosrapid.<Region>.myqcloud.com
Date: <GMT Date>
Authorization: <Auth String>
```

① 说明：

Authorization 字段的具体格式见 [公共请求头部](#)。

请求参数

此接口无请求参数。

请求头部

本接口仅使用 [公共请求头部](#)，无私有请求头部。

请求体

该请求的请求体为空。

响应

响应头部

除 [公共响应头部](#) 外，本接口还会返回以下响应头部：

Header 名称	描述	类型	是否必返
Cache-Control	RFC 2616 中定义的缓存指令，仅当对象元数据包含此项时才会返回该头部	string	否
Content-Disposition	RFC 2616 中定义的文件名称，仅当对象元数据包含此项时才会返回该头部	string	否
Content-Encoding	RFC 2616 中定义的编码格式，仅当对象元数据包含此项时才会返回该头部	string	否
Content-Language	RFC 2616 中定义的内容语言，仅当对象元数据包含此项时才会返回该头部	string	否
Expires	RFC 2616 中定义的缓存失效时间，仅当对象元数据包含此项时才会返回该头部	string	否
X-COS-meta-*	包括用户自定义元数据头部后缀和用户自定义元数据信息	string	否
X-COS-storage-class	对象存储类型，当前固定为 <code>RAPID</code>	enum	是

响应体

该接口的响应体为空。`Content-Length` 响应头部返回的是对象的实际大小（字节），而非响应体大小。

错误码

本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

案例一：查询文件对象元数据

请求

响应

案例二：查询包含用户自定义元数据的对象

请求

响应

第75 共324页

案例三：查询目录对象

请求

响应

版权所有：腾讯云计算（北京）有限责任公司

DeleteObject

最近更新时间：2026-10-21 18:34:04

DeleteObject 接口请求可以从指定存储桶（Bucket）中删除一个对象（Object）。

注意事项

文件对象与目录对象

对象键以 `/` 结尾代表目录对象，不以 `/` 结尾代表文件对象。本接口对两类对象的删除遵循统一规则：

- 对不存在的对象（包括对象键末尾类型与实际类型不匹配的情况，例如实际只存在文件对象 `foo` 时调用 `DELETE /foo/`，或实际只存在目录对象 `foo/` 时调用 `DELETE /foo`）调用本接口，均返回 `204 No Content`，且不会删除任何已存在的对象。
- 删除文件对象：仅删除该文件对象本身，不影响其所在的目录对象。
- 删除目录对象：默认仅当目录为空时才能成功删除，返回 `204 No Content`；目录非空时返回 `409 DirNotEmpty`。如需一次性删除非空目录及其下所有对象，可携带 `x-cos-delete-recursive: true`，服务将递归删除目录本身以及其下所有子对象，返回 `204 No Content`。
- x-cos-delete-recursive 的适用范围：** `x-cos-delete-recursive` 仅对目录对象有意义。对文件对象（对象键不以 `/` 结尾）调用本接口时携带 `x-cos-delete-recursive: true`，将返回 `400 InvalidArgument`。

请求

请求语法

```
DELETE /<ObjectKey> HTTP/1.1
Host: <BucketName-APPID>.cosrapid.<Region>.myqcloud.com
Date: <GMT Date>
Authorization: <Auth String>
```

说明：
Authorization 字段的具体格式见 [公共请求头部](#)。

请求参数

此接口无请求参数。

请求头部

除 [公共请求头部](#) 外，本接口还支持以下请求头部：

Header 名称	描述	类型	是否必选
x-cos-delete-recursive	是否递归删除目录下的所有对象，取值为 <code>true</code> 时启用。行为契约详见「 注意事项 」	string	否

请求体

该请求的请求体为空。

响应

响应头部

仅返回 **公共响应头部**，本接口无私有响应头部。

响应体

该接口的响应体为空。

错误码

本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

案例一：删除文件对象

请求

响应

```
HTTP/1.1 204 No Content
Content-Length: 0
```

案例二：删除空目录对象

请求

响应

案例三：删除非空目录对象

请求

响应

第79 共324页

案例四：使用 x-cos-delete-recursive 递归删除非空目录

请求

响应

第80 共324页

DeleteMultipleObjects

最近更新时间：2026-10-21 18:34:04

DeleteMultipleObjects 接口请求可以批量删除指定存储桶中的多个对象（Object），单次请求支持删除1-1000个对象。本接口提供两种响应模式：

- Quiet 模式：响应中仅包含删除失败的对象信息和错误信息。
- Verbose 模式：响应中包含每个对象的删除结果信息。

注意事项

单个对象的删除行为

单个对象的删除语义与 [DeleteObject](#) 一致：对象键以 / 结尾代表目录对象，不以 / 结尾代表文件对象，本接口不会对对象键末尾做任何容错。需重点关注的是：本接口起始状态码始终为 200 OK，单个对象删除失败不再以 HTTP 状态码呈现，而是以对象级错误码（例如非空目录对象的 `DirNotEmpty` 呈现于响应体的 `<Error>` 列表中）返回。

单次请求对象数量限制

单次请求中携带的对象数量必须为1-1000个，否则将返回 `400 InvalidArgument`。请求体中未携带任何 `<Object>` 节点、或携带超过1000个 `<Object>` 节点时均按此处理。

请求

请求语法

```
POST /?delete HTTP/1.1
Host: <BucketName-APPID>.cosrapid.<Region>.myqcloud.com
Date: <GMT Date>
Content-Type: application/xml
Content-Length: <Content Length>
Content-MD5: <MD5>
Authorization: <Auth String>

[Request Body]
```

① 说明：

Authorization 字段的具体格式见 [公共请求头部](#)。

请求参数

此接口无请求参数。

请求头部

除 [公共请求头部](#) 外，本接口还支持以下请求头部：

Header 名称	描述	类型	是否必选
Content-MD5	RFC 1864 中定义的请求体内容的 MD5 校验值（Base64 编码）。若携带该头部，服务端会与实际接收到的请求体计算结果进行比对，不一致时返回 <code>BadDigest</code>	string	否

请求体

请求体为 `application/xml` 格式，包含要删除的对象信息：

```
<Delete>
  <Quiet>boolean</Quiet>
  <Object>
    <Key>string</Key>
  </Object>
  <Object>
    <Key>string</Key>
  </Object>
</Delete>
```

具体节点说明：

节点名称	父节点	描述	类型	是否必选
Delete	无	包含 DeleteMultipleObjects 操作的所有请求信息	Container	是
Quiet	Delete	是否使用 Quiet 模式，默认为 <code>false</code> （Verbose 模式）；两种模式的差异详见上文接口简介	boolean	否
Object	Delete	单个要删除的目标对象的信息	Container	是

			r	
Key	Delete. Object	要删除的目标对象的对象键（无需以 <code>/</code> 开头）。对象键以 <code>/</code> 结尾代表目录对象	string	是

响应

响应头部

仅返回 [公共响应头部](#)，本接口无私有响应头部。

响应体

响应体为 `application/xml` 格式：

```
<DeleteResult>
  <Deleted>
    <Key>string</Key>
  </Deleted>
  <Error>
    <Key>string</Key>
    <Code>string</Code>
    <Message>string</Message>
  </Error>
</DeleteResult>
```

具体节点说明：

节点名称	父节点	描述	类型
Delete Result	无	保存 DeleteMultipleObjects 结果的所有信息	Container
Deleted	Delete Result	单个删除成功的对象条目，仅当请求中 <code><Quiet></code> 为 <code>false</code> 时才会返回该元素	Container
Error	Delete Result	单个删除失败的对象条目；无论 Quiet 模式如何，删除失败的对象均会出现在该列表中	Container

`Deleted` 节点包含的子节点：

节点名称	父节点	描述	类型
------	-----	----	----

Error 节点包含的子节点:

错误码

本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

案例一：批量删除（Verbose 模式）

请求

响应

案例二：批量删除（Quiet 模式）

请求

第85 共324页

```
</Object>
<Object>
  <Key>example-object-2.jpg</Key>
</Object>
</Delete>
```

响应

所有对象均删除成功时，响应体为空 `DeleteResult`：

```
HTTP/1.1 200 OK
Content-Type: application/xml
Connection: keep-alive
Date: Wed, 29 May 2019 04:10:12 GMT
x-cos-request-id: NjUwMDAwMDAwMDAwMDAwMDAw

<DeleteResult></DeleteResult>
```

案例三：批量删除目录对象（部分失败）

本案例演示批量删除时其中一个目录对象因非空而失败的场景。

请求

```
POST /?delete HTTP/1.1
Host: examplebucket-x--1250000000.cosrapid.ap-guangzhou.myqcloud.com
Date: Wed, 29 May 2019 04:10:12 GMT
Content-Type: application/xml
Content-Length: 200
Authorization: q-sign-algorithm=sha1&q-ak=AKIDXXXXXXXXX&q-sign-
time=1559074212;1559077812&q-key-time=1559074212;1559077812&q-header-
list=content-length;content-type;date;host&q-url-param-list=delete&q-
signature=xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

<Delete>
  <Quiet>>false</Quiet>
  <Object>
    <Key>example-object-1.jpg</Key>
  </Object>
</Object>
```

响应

```
HTTP/1.1 200 OK
Content-Type: application/xml
Connection: keep-alive
Date: Wed, 29 May 2019 04:10:12 GMT
x-cos-request-id: NjUwMDAwMDAwMDAwMDAwMDAw
```

PutObjectCopy

最近更新时间：2026-10-21 18:34:04

PutObjectCopy 接口请求创建一个已存在对象的副本，即将一个对象从源路径（对象键）复制到目标路径（对象键）。在复制过程中，可以通过 `x-cos-metadata-directive` 头部指定元数据的处理方式：默认情况下源对象的元数据会被复制到目标对象，也可以选择不复制源对象的元数据信息，而是使用本次请求中指定的新元数据。

注意事项

单对象大小限制

PutObjectCopy 接口最大支持复制 5 GiB 的对象，源对象大小超过 5 GiB 时返回 `EntityTooLarge`。如需复制大于 5 GiB 的对象，请使用 [分块上传](#) 系列 API 接口。

仅支持同桶拷贝

源对象与目标对象必须位于同一存储桶。当 `x-cos-copy-source` 中解析出的源存储桶名与请求 Host 中的存储桶名不一致时，请求返回 `InvalidCopySource`。

源对象与目标对象均需为文件对象

本接口只支持复制文件对象，不支持目录对象。`x-cos-copy-source` 中的源对象键末尾为 `/` 时，请求返回 `400 InvalidCopySource`；目标对象键末尾为 `/` 时，请求返回 `400 InvalidArgument`。

目标对象所在目录无需预先创建

- 若目标对象键中包含多级路径（例如 `a/b/c.txt`），当路径中的目录对象（`a/`、`a/b/`）尚未创建时，服务会自动补齐，调用方无需先调用 PutObject 建立目录对象。
- 若路径中的某一层已存在同名的文件对象（例如目标对象键为 `a/b/c.txt`，但 `a/b` 已作为文件对象存在），则无法在其下创建对象，返回 `409 FileAlreadyExists`。

请求

请求语法

```
PUT /<ObjectKey> HTTP/1.1
Host: <BucketName-APPID>.cosrapid.<Region>.myqcloud.com
Date: <GMT Date>
x-cos-copy-source: <BucketName-APPID>.cosrapid.
<Region>.myqcloud.com/<SourceObjectKey>
Content-Length: 0
Authorization: <Auth String>
```

说明：
Authorization 字段的具体格式见 [公共请求头部](#)。

请求参数

此接口无请求参数。

请求头部

除 [公共请求头部](#) 外，本接口还支持以下请求头部：

Header 名称	描述	类型	是否必选
x-cos-copy-source	源对象的 URL，格式为 <code><BucketName-APPID>.cosrapid.<Region>.myqcloud.com/<SourceObjectKey></code> ，其中对象键需经过 URL encode。源存储桶必须与目标存储桶相同，否则返回 <code>InvalidCopySource</code>	string	是
x-cos-metadata-directive	目标对象元数据的处理方式，枚举值 <code>Copy</code> 、 <code>Replaced</code> ，默认为 <code>Copy</code> 。取值为 <code>Copy</code> 时复制源对象的元数据，本次请求中携带的元数据头部会被忽略；取值为 <code>Replaced</code> 时使用本次请求头部中的元数据作为目标对象的元数据。取其他值时返回 <code>InvalidArgument</code>	enum	否
x-cos-storage-class	目标对象的存储类型，当前仅支持 <code>RAPID</code> （默认值）。指定其他取值时返回 <code>NotSupportedStorageClass</code>	enum	否
x-cos-forbid-overwrite	是否禁止覆盖同名对象，取值为 <code>true</code> 时启用；目标对象已存在则返回 <code>409 FileAlreadyExists</code>	string	否

目标对象元数据相关头部

下列头部用于设置目标对象的元数据信息，仅在 `x-cos-metadata-directive` 指定为 `Replaced` 时生效；当指定为 `Copy` 或不指定时，下列头部会被忽略，目标对象保留源对象的元数据。

Header 名称	描述	类型	是否必选
Cache-Control	RFC 2616 中定义的缓存指令，将作为目标对象元数据保存	string	否
Content-Disposition	RFC 2616 中定义的文件名称，将作为目标对象元数据保存	string	否
Content-Encoding	RFC 2616 中定义的编码格式，将作为目标对象元数据保存	string	否
Content-Language	RFC 2616 中定义的内容语言，将作为目标对象元数据保存	string	否
Content-Type	RFC 2616 中定义的 HTTP 请求内容类型（MIME），将作为目标对象元数据保存	string	否
Expires	RFC 2616 中定义的缓存失效时间，将作为目标对象元数据保存	string	否
x-cos-meta-[自定义后缀]	用户自定义元数据头部，例如 <code>x-cos-meta-test: test metadata</code> 。自定义后缀仅支持减号（-）、数字、英文小写字母（a~z），英文大写字母会被归一为小写	string	否

请求体

该请求的请求体为空。

响应

响应头部

仅返回 [公共响应头部](#)，本接口无私有响应头部。

响应体

响应体为 XML 格式：

```
<?xml version="1.0" encoding="UTF-8"?>
<CopyObjectResult>
  <ETag>string</ETag>
  <CRC64>string</CRC64>
  <LastModified>string</LastModified>
</CopyObjectResult>
```

具体节点说明：

节点名称	父节点	描述	类型
CopyObjectResult	无	保存 PutObjectCopy 结果的所有信息	Container
ETag	CopyObjectResult	目标对象的 MD5 校验值，使用双引号包裹，例如 "ee8de918d05640145b18f70f4c3aa602"	string
CRC64	CopyObjectResult	目标对象的 CRC64-ECMA 校验值，十进制字符串	string
LastModified	CopyObjectResult	目标对象的最近修改时间，UTC 时区，ISO 8601 格式，秒精度，例如 2019-05-29T04:10:12Z	string

错误码

本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

案例一：简单复制

请求

```
PUT /exampleobject HTTP/1.1
Host: examplebucket-x--1250000000.cosrapid.ap-guangzhou.myqcloud.com
Date: Wed, 29 May 2019 04:10:12 GMT
x-cos-copy-source: examplebucket-x--1250000000.cosrapid.ap-guangzhou.myqcloud.com/sourceobject
Content-Length: 0
Authorization: q-sign-algorithm=sha1&q-ak=AKIDXXXXXXXXX&q-sign-time=1559074212;1559077812&q-key-time=1559074212;1559077812&q-header-
```

响应

案例二：复制并替换元数据

请求

第92 共324页

[illegible]

```
PUT /exampleobject HTTP/1.1
Host: examplebucket-x--1250000000.cosrapid.ap-guangzhou.myqcloud.com
Date: Wed, 29 May 2019 04:10:12 GMT
x-cos-metadata-directive: Replaced
x-cos-meta-author: alice
x-cos-copy-source: examplebucket-x--1250000000.cosrapid.ap-
guangzhou.myqcloud.com/exampleobject
Content-Length: 0
Authorization: q-sign-algorithm=sha1&q-ak=AKIDXXXXXXXXX&q-sign-
time=1559074212;1559077812&q-key-time=1559074212;1559077812&q-header-
list=content-length;date;host;x-cos-copy-source;x-cos-meta-author;x-cos-
metadata-directive&q-url-param-list=&q-
signature=xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

```
HTTP/1.1 200 OK
```

案例四：使用 x-cos-forbid-overwrite 防覆盖

请求

响应

```
HTTP/1.1 200 OK
Content-Type: application/xml
Connection: keep-alive
Date: Wed, 29 May 2019 04:10:12 GMT
x-cos-request-id: NjUwMDAwMDAwMDAwMDAwMDAw
```

如果目标对象已存在，则返回报错：

GetBucket

最近更新时间：2026-10-21 18:34:04

GetBucket 接口请求等同于 ListObjects，可以列出指定存储桶下的部分或全部对象。

注意事项

Delimiter 与 Prefix 约束

本接口对 `delimiter` 和 `prefix` 参数有以下约束：

- `delimiter` 为必选参数，且取值必须为 `/`。未携带 `delimiter` 参数、或取值不为 `/` 时，将返回 `400 InvalidArgument`。
- 若携带 `prefix` 参数，则 `prefix` 必须以 `/` 结尾。即 `prefix` 必须是一个目录前缀，不支持以文件名前缀进行模糊筛选；不符合该约束时，将返回 `400 InvalidArgument`。
- 不携带 `prefix` 时，默认从存储桶根目录开始列举。

上述约束意味着每次请求只列举单一目录下的子对象，而不会递归列举子目录中的内容。

文件对象与目录对象的呈现方式

同一次列举的响应中：

- 文件对象（对象键不以 `/` 结尾）出现在 `<Contents>` 中，每项包含 `Key`、`Size`、`ETag`、`LastModified`、`StorageClass` 等元信息。
- 目录对象（对象键以 `/` 结尾）出现在 `<CommonPrefixes>` 中，仅以前缀字符串形式呈现，不携带大小、`ETag`、最后修改时间等元信息。

分页

当未列举完毕时，响应中的 `<IsTruncated>` 为 `true`，并通过 `<NextMarker>` 给出下一页的游标。客户端需在下次请求中将该值原样作为 `marker` 传入，直到响应中 `<IsTruncated>` 为 `false`。

存储类型

存储桶中的所有对象的 `StorageClass` 固定为 `RAPID`。

请求

请求语法

```
GET /?prefix=<prefix>&delimiter=/&marker=<marker>&max-keys=<max-keys>
HTTP/1.1
Host: <BucketName-APPID>.cosrapid.<Region>.myqcloud.com
Date: <GMT Date>
```

Authorization: <Auth String>

说明：
Authorization 字段的具体格式见 [公共请求头部](#)。

请求参数

参数名称	描述	类型	是否必选
prefix	限定返回的对象键必须以该字符串作为前缀。取值需以 <code>/</code> 结尾，详见上文「Delimiter 与 Prefix 约束」	string	否
delimiter	对象键的分组分隔符。详见上文「Delimiter 与 Prefix 约束」	string	是
marker	分页游标，取值为上一次 GetBucket 响应中的 <code><NextMarker></code> ；首次请求无需携带	string	否
max-keys	单次请求返回的最大条目数，取值范围 <code>[1, 1000]</code> ，默认值为 <code>1000</code>	int	否

请求头部

仅使用 [公共请求头部](#)，本接口无私有请求头部。

请求体

此接口无请求体。

响应

响应头部

仅返回 [公共响应头部](#)，本接口无私有响应头部。

响应体

响应体为 `application/xml` 格式：

```
<ListBucketResult>
  <Name>string</Name>
  <Prefix>string</Prefix>
  <Marker>string</Marker>
  <NextMarker>string</NextMarker>
  <Delimiter>string</Delimiter>
  <MaxKeys>int</MaxKeys>
  <IsTruncated>boolean</IsTruncated>
  <Contents>
    <Key>string</Key>
    <LastModified>string</LastModified>
    <ETag>string</ETag>
    <Size>int</Size>
    <StorageClass>string</StorageClass>
  </Contents>
  <CommonPrefixes>
    <Prefix>string</Prefix>
  </CommonPrefixes>
  <CommonPrefixes>
    <Prefix>string</Prefix>
  </CommonPrefixes>
</ListBucketResult>
```

具体节点说明：

节点名称	父节点	描述	类型
ListBucketResult	无	保存 GetBucket 结果的所有信息	Container
Name	ListBucketResult	存储桶名称	string
Prefix	ListBucketResult	本次列举使用的对象键前缀，与请求参数 <code>prefix</code> 一致	string
Marker	ListBucketResult	本次请求所使用的分页游标，回显请求参数 <code>marker</code> ；未指定时为空	string
NextMarker	ListBucketResult	下一页的分页游标，仅在 <code>IsTruncated</code> 为 <code>true</code> 时返回非空	string

Delimiter	ListBucketResult	本次列举使用的分隔符，与请求参数 <code>delimiter</code> 一致，固定为 <code>/</code>	string
MaxKeys	ListBucketResult	本次列举使用的最大条目数，与请求参数 <code>max-keys</code> 一致	int
IsTruncated	ListBucketResult	是否仍有未列举完的条目： <code>true</code> 表示存在下一页， <code>false</code> 表示已全部列举完	boolean
Contents	ListBucketResult	单个文件对象的元信息条目，可能出现 0 条或多条	Container
CommonPrefixes	ListBucketResult	单个目录对象的前缀条目，可能出现 0 条或多条	Container

`Contents` 节点包含的子节点：

节点名称	父节点	描述	类型
Key	ListBucketResult.Contents	文件对象的对象键	string
LastModified	ListBucketResult.Contents	文件对象的最后修改时间，UTC 时区，格式如 <code>2019-08-20T11:59:35.000Z</code>	string
ETag	ListBucketResult.Contents	文件对象的 ETag，由双引号包裹	string
Size	ListBucketResult.Contents	文件对象的大小，单位为字节	int
StorageClass	ListBucketResult.Contents	文件对象的存储类型，固定为 <code>RAPID</code>	string

`CommonPrefixes` 节点包含的子节点：

节点名称	父节点	描述	类型
Prefix	ListBucketResult.CommonPrefixes	目录对象的对象键，以 <code>/</code> 结尾	string

错误码

本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。

案例一：列举存储桶根目录下的对象

案例二：列举指定目录下的对象

请求

响应

版权所有：腾讯云计算（北京）有限责任公司

案例三：分页列举（响应被截断）

请求

响应

```
<Contents>
  <Key>folder-a/file-2.jpg</Key>
  <LastModified>2019-05-29T04:00:00.000Z</LastModified>
  <ETag>"d41d8cd98f00b204e9800998ecf8427e"</ETag>
  <Size>20480</Size>
  <StorageClass>RAPID</StorageClass>
</Contents>
</ListBucketResult>
```

下一次翻页请求中将上一次响应的 `<NextMarker>` 原样作为 `marker` 传入即可继续列举。

RenameObject

最近更新时间：2026-10-21 18:34:05

RenameObject 接口请求将一个已存在的对象从源路径（对象键）重命名到目标路径（对象键）。重命名操作是原子的，成功后源对象不再保留，目标对象即为原来的源对象。

注意事项

仅支持同桶重命名

源对象与目标对象必须位于同一存储桶，本接口不支持跨桶重命名。目标存储桶即为请求 Host 中的存储桶，源对象存储桶由 `x-cos-rename-source` 中的路径隐式指向该桶。

源对象与目标对象的类型必须一致

对象键末尾的 `/` 严格区分文件对象与目录对象。`x-cos-rename-source` 与请求路径中的对象键必须同为文件对象（均不以 `/` 结尾）或同为目录对象（均以 `/` 结尾），否则返回 `400 InvalidArgument`。

源对象必须存在

若 `x-cos-rename-source` 中指定的源对象不存在，返回 `404 NoSuchKey`。接口不会对对象键末尾的 `/` 做任何容错，例如实际只存在文件 `foo`，指定 `/foo/` 作为源对象将按“源对象不存在”处理，返回 `404 NoSuchKey`。

目标对象所在目录无需预先创建

- 若目标对象键中包含多级路径（例如 `a/b/c.txt`），当路径中的目录对象（`a/`、`a/b/`）尚未创建时，服务会自动补齐，调用方无需先调用 PutObject 建立目录对象。
- 若路径中的某一层已存在同名的文件对象（例如目标对象键为 `a/b/c.txt`，但 `a/b` 已作为文件对象存在），则无法在其下创建对象，返回 `409 FileAlreadyExists`。

目标对象已存在时的覆盖行为

目标对象已存在时，是否可覆盖取决于源对象类型：

- 源为文件对象：**默认覆盖目标路径上的同名文件对象；若不希望被覆盖，需携带请求头 `x-cos-forbid-overwrite: true`。
- 源为目录对象：**不支持覆盖同名对象，`x-cos-forbid-overwrite` 头部不生效。目标路径上已存在同名对象时，请求将失败。

请求

请求语法

```
PUT /<ObjectKey> HTTP/1.1
Host: <BucketName-APPID>.cosrapid.<Region>.myqcloud.com
Date: <GMT Date>
x-cos-rename-source: /<SourceObjectKey>
Content-Length: 0
Authorization: <Auth String>
```

说明：

Authorization 字段的具体格式见 [公共请求头部](#)。

请求参数

此接口无请求参数。

请求头部

除 [公共请求头部](#) 外，本接口还支持以下请求头部：

Header 名称	描述	类型	是否必选
x-cos-rename-source	源对象的路径，格式为 <code>/<SourceObjectKey></code> ，其中对象键需经过 URL encode	string	是
x-cos-forbid-overwrite	是否禁止覆盖目标路径上的同名文件对象，取值为 <code>true</code> 时启用。行为契约详见 目标对象已存在时的覆盖行为	string	否

请求体

该请求的请求体为空。

响应

响应头部

仅返回 [公共响应头部](#)，本接口无私有响应头部。

响应体

该请求的响应体为空。

错误码

实际案例

案例一：重命名文件对象

请求

响应

```
HTTP/1.1 200 OK
Content-Length: 0
Connection: keep-alive
Date: Wed, 29 May 2019 04:10:12 GMT
x-cos-request-id: NjUwMDAwMDAwMDAwMDAwMDAw
```

案例二：重命名目录对象

请求

案例三：使用 x-cos-forbid-overwrite 防覆盖

```
HTTP/1.1 200 OK
Content-Length: 0
Connection: keep-alive
Date: Wed, 29 May 2019 04:10:12 GMT
x-cos-request-id: NjUwMDAwMDAwMDAwMDAwMDAw
```

案例三：使用 x-cos-forbid-overwrite 防覆盖

如果目标对象不存在，则重命名成功：

```
HTTP/1.1 200 OK
Content-Length: 0
Connection: keep-alive
Date: Wed, 29 May 2019 04:10:12 GMT
x-cos-request-id: NjUwMDAwMDAwMDAwMDAwMDAw
```

如果目标对象已存在，则返回报错：

```
HTTP/1.1 409 Conflict
Content-Type: application/xml
Date: Wed, 29 May 2019 04:10:12 GMT
```

```
x-cos-request-id: NjUwMDAwMDAwMDAwMDAwMDAw
```

```
<?xml version='1.0' encoding='utf-8'?>
```

```
<Error>
```

```
  <Code>FileAlreadyExists</Code>
```

```
  <Message>File already exists.</Message>
```

```
  <Resource></Resource>
```

```
  <RequestId>NjUwMDAwMDAwMDAwMDAwMDAw</RequestId>
```

```
  <TraceId></TraceId>
```

```
</Error>
```


分块上传操作

InitiateMultipartUpload

最近更新时间：2026-10-21 18:34:04

InitiateMultipartUpload 接口请求实现初始化分块上传，成功执行此请求后将返回一个全局唯一的 `UploadId`，用于后续 [UploadPart](#)、[UploadPartCopy](#)、[CompleteMultipartUpload](#)、[AbortMultipartUpload](#)、[ListParts](#) 等接口。

注意事项

适用场景

分块上传仅用于上传文件对象，目标对象键不应以 `/` 结尾。目录对象的创建请直接使用 [PutObject](#)。

元数据头部归属

本接口在初始化阶段携带的 `Cache-Control`、`Content-Disposition`、`Content-Encoding`、`Content-Language`、`Content-Type`、`Expires`、`x-cos-meta-*` 等头部，将作为最终对象的元数据保存，并在 [CompleteMultipartUpload](#) 完成时一并固化。

未完成的分块上传

调用本接口后，若长时间未调用 [CompleteMultipartUpload](#) 或 [AbortMultipartUpload](#)，对应 `UploadId` 及其已上传分块会持续占用存储桶空间。建议调用方在适当时机主动调用 [AbortMultipartUpload](#) 释放，或通过 [ListMultipartUploads](#) 进行盘点。

请求

请求语法

```
POST /<ObjectKey>?uploads HTTP/1.1
Host: <BucketName-APPID>.cosrapid.<Region>.myqcloud.com
Date: <GMT Date>
Content-Type: <Content Type>
Content-Length: 0
Authorization: <Auth String>
```

① 说明：

Authorization 字段的具体格式见 [公共请求头部](#)。

请求参数

参数名称	描述	类型	是否必选
uploads	用于声明本次请求为初始化分块上传，固定为空串。	string	是

请求头部

除 [公共请求头部](#) 外，本接口还支持以下请求头部：

常规请求头部

Header 名称	描述	类型	是否必选
Cache-Control	RFC 2616 中定义的缓存指令，将作为对象元数据保存	string	否
Content-Disposition	RFC 2616 中定义的文件名称，将作为对象元数据保存	string	否
Content-Encoding	RFC 2616 中定义的编码格式，将作为对象元数据保存	string	否
Content-Language	RFC 2616 中定义的内容语言，将作为对象元数据保存	string	否
Content-Type	RFC 2616 中定义的 HTTP 请求内容类型（MIME），用于描述待上传对象的内容类型，将作为对象元数据保存。本接口不支持指定为 <code>application/x-directory</code> ，否则返回 <code>InvalidArgument</code>	string	否
Expires	RFC 2616 中定义的缓存失效时间，将作为对象元数据保存	string	否

专用请求头部

Header 名称	描述	类型	是否必选
x-cos-meta-[自定义后缀]	用户自定义元数据头部，例如 <code>x-cos-meta-test: test metadata</code> 。自定义后缀仅支持减号（-）、数字、英文小写字母（a~z），英文大写字母会被归一为小写	string	否
x-cos-storage-class	对象存储类型，当前仅支持 <code>RAPID</code> （默认值）。指定其他取值时返回 <code>NotSupportedStorageClass</code>	enum	否

请求体

该请求的请求体为空。

响应

响应头部

仅返回 [公共响应头部](#)，本接口无私有响应头部。

响应体

请求成功，返回 `application/xml` 数据，包含分块上传初始化信息。

```
<?xml version="1.0" encoding="UTF-8"?>
<InitiateMultipartUploadResult>
  <Bucket>string</Bucket>
  <Key>string</Key>
  <UploadId>string</UploadId>
</InitiateMultipartUploadResult>
```

具体节点说明：

节点名称	父节点	描述	类型
InitiateMultipartUploadResult	无	保存 InitiateMultipartUpload 结果的所有信息	Container

错误码

实际案例

案例一：初始化一个分块上传

请求

响应

[illegible]

案例二：初始化时指定元数据

请求

响应

第113 共324页

```
<UploadId>1590654934dfb1343b4323b711afc22569c18af51596d4f2e40faf392fe1bb  
469c5b77115f</UploadId>  
</InitiateMultipartUploadResult>
```

UploadPart

最近更新时间：2026-10-21 18:34:04

UploadPart 接口请求实现将对象按照分块的方式上传到指定存储桶。需先调用 [InitiateMultipartUpload](#) 接口获取 UploadId，再使用相同的对象键、UploadId 以及递增的 PartNumber 调用本接口上传各分块；分块上传完成后，调用 [CompleteMultipartUpload](#) 完成对象合并。

注意事项

分块编号

partNumber 必须为1到10000之间的整数（包含1和10000）。超出范围将返回 InvalidArgument。各分块允许乱序上传，最终对象内容按分块编号升序拼接。

单分块大小限制

单个分块的大小不能超过 5 GiB，超出时返回 EntityTooLarge。除最后一个分块外，其余分块在调用 CompleteMultipartUpload 时还需满足最小分块大小限制，详见 [CompleteMultipartUpload](#)。

分块覆盖

若使用相同的 UploadId 和 PartNumber 重复上传，后到达的请求将覆盖之前已上传的同编号分块。

UploadId 有效性

UploadId 不存在、已被 AbortMultipartUpload 终止、或已被 CompleteMultipartUpload 完成时，本接口返回 NoSuchUpload。

请求

请求语法

```
PUT /<ObjectKey>?partNumber=<PartNumber>&uploadId=<UploadId> HTTP/1.1
Host: <BucketName-APPID>.cosrapid.<Region>.myqcloud.com
Date: <GMT Date>
Content-Length: <Content Length>
Content-MD5: <MD5>
Authorization: <Auth String>

[Object Part]
```

 说明：

Authorization 字段的具体格式见 [公共请求头部](#)。

请求参数

参数名称	描述	类型	是否必选
partNumber	标识本次分块上传的编号，取值范围为 1 到 10000（包含两端）	integer	是
uploadId	调用 InitiateMultipartUpload 接口初始化分块上传时返回的 <code>UploadId</code>	string	是

请求头部

除 [公共请求头部](#) 外，本接口还支持以下请求头部：

Header 名称	描述	类型	是否必选
Content-Length	RFC 2616 中定义的 HTTP 请求内容长度（字节）。未指定 <code>Transfer-Encoding: chunked</code> 时必须携带	string	否
Content-MD5	RFC 2616 中定义的请求内容 MD5 摘要的 Base64 编码值。携带时服务端会校验，不一致返回 <code>BadDigest</code>	string	否
Transfer-Encoding	RFC 2616 中定义的传输编码格式。如果希望以分块方式传输请求体，则指定 <code>Transfer-Encoding: chunked</code> ，并不再指定 <code>Content-Length</code>	string	否

请求体

该请求的请求体（Body）为本次上传分块的原始字节流。

响应

响应头部

除 [公共响应头部](#) 外，本接口还会返回以下响应头部：

响应体

错误码

本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

案例一：上传一个分块

请求

版权所有：腾讯云计算（北京）有限责任公司

响应

案例二：使用 chunked 传输编码上传分块

请求

响应

```
HTTP/1.1 200 OK
Content-Length: 0
Connection: keep-alive
Date: Wed, 25 Mar 2020 10:08:01 GMT
ETag: "aa488bb80185a6be87f4a7b936a80752"
Last-Modified: Wed, 25 Mar 2020 10:08:01 GMT
x-cos-hash-crc64ecma: 7188322482464764960
x-cos-request-id: NjUwMDAwMDAwMDAwMDAwMDAw
```

UploadPartCopy

最近更新时间：2026-10-21 18:34:04

UploadPartCopy 接口请求实现将一个已存在对象的内容（或其某个字节范围）作为指定分块上传任务的一个分块。通过 `x-cos-copy-source` 指定源对象，`x-cos-copy-source-range` 指定字节范围，以避免在客户端重新下载与上传源对象数据。需先调用 [InitiateMultipartUpload](#) 接口获取目标对象的 `UploadId`，再使用本接口逐块复制源对象内容；分块上传完成后，调用 [CompleteMultipartUpload](#) 完成对象合并。

注意事项

仅支持同桶拷贝

源对象与目标对象必须位于同一存储桶。当 `x-cos-copy-source` 中解析出的源存储桶名与请求 Host 中的存储桶名不一致时，请求返回 `InvalidCopySource`。

源对象必须为文件对象

本接口只支持复制文件对象，不支持目录对象。`x-cos-copy-source` 中的源对象键末尾为 `/` 时，请求返回 `400 InvalidCopySource`。

分块编号

`partNumber` 必须为1到10000之间的整数（包含1和10000）。超出范围将返回 `InvalidArgument`。各分块允许乱序上传，最终对象内容按分块编号升序拼接。

单分块大小限制

单次复制的分块大小不能超过5 GiB，超出时返回 `EntityTooLarge`。当源对象大于5 GiB 时，必须通过 `x-cos-copy-source-range` 限定每次复制的字节范围，使每个分块大小不超过5 GiB。

分块覆盖

若使用相同的 `UploadId` 和 `PartNumber` 重复调用本接口或 [UploadPart](#)，后到达的请求将覆盖之前已上传的同编号分块。

UploadId 有效性

`UploadId` 不存在、已被 [AbortMultipartUpload](#) 终止、或已被 [CompleteMultipartUpload](#) 完成时，本接口返回 `NoSuchUpload`。

请求

请求语法

```
PUT /<ObjectKey>?partNumber=<PartNumber>&uploadId=<UploadId> HTTP/1.1
Host: <BucketName-APPID>.cosrapid.<Region>.myqcloud.com
Date: <GMT Date>
x-cos-copy-source: <BucketName-APPID>.cosrapid.
<Region>.myqcloud.com/<SourceObjectKey>
x-cos-copy-source-range: bytes=<first>-<last> # 可选；不指定时复制整个源对象
Content-Length: 0
Authorization: <Auth String>
```

① 说明：

Authorization 字段的具体格式见 [公共请求头部](#)。

请求参数

参数名称	描述	类型	是否必选
partNumber	标识本次分块上传的编号，取值范围为 1 到 10000（包含两端）	integer	是
uploadId	调用 InitiateMultipartUpload 接口初始化分块上传时返回的 <code>uploadId</code>	string	是

请求头部

除 [公共请求头部](#) 外，本接口还支持以下请求头部：

Header 名称	描述	类型	是否必选
x-cos-copy-source	源对象的 URL，格式为 <code><BucketName-APPID>.cosrapid.<Region>.myqcloud.com/<SourceObjectKey></code> ，其中对象键需经过 URL encode。源存储桶必须与目标存储桶相同，否则返回 <code>InvalidCopySource</code>	string	是
x-cos-copy	源对象的字节范围，格式为 <code>bytes=<first>-<last></code> ， <code>first</code> 和 <code>last</code> 都是基于 0 开始的偏移量，且必须同时指定，例如 <code>bytes=0-9</code> 表示拷贝源对	string	否

- source-range	象的开始 10 个字节。不指定时表示拷贝整个对象。格式不合法时返回 Invalid Range		
----------------	---	--	--

请求体

该请求的请求体为空。

响应

响应头部

仅返回 公共响应头部，本接口无私有响应头部。

响应体

响应体为 application/xml 格式：

```
<?xml version="1.0" encoding="UTF-8"?>
<CopyPartResult>
  <ETag>string</ETag>
  <LastModified>string</LastModified>
</CopyPartResult>
```

具体节点说明：

节点名称	父节点	描述	类型
CopyPartResult	无	保存 UploadPartCopy 结果的所有信息	Container
ETag	CopyPartResult	本分块的 MD5 校验值，使用双引号包裹。该值需要在调用 CompleteMultipartUpload 时作为对应分块的 ETag 一并提交	string
LastModified	CopyPartResult	本分块的最近一次上传时间，UTC 时区，ISO 8601 格式，秒精度，例如 2019-05-29T04:10:12Z	string

错误码

本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

响应

不指定 `x-cos-copy-source-range`，表示将整个源对象作为一个分块写入。源对象大小不能超过 5 GiB。

请求

响应

第124 共324页

CompleteMultipartUpload

最近更新时间：2026-10-21 18:34:04

CompleteMultipartUpload 接口请求实现完成整个分块上传。当使用 [UploadPart](#) 和 [UploadPartCopy](#) 上传所有分块完成后，必须调用本接口来完成最终对象的合并。在请求体中需提供每一个分块的 `PartNumber` 和 `ETag`，用于校验分块的完整性与顺序。

注意事项

分块顺序与完整性

请求体中的 `<Part>` 列表必须满足以下约束：

- 每个 `<Part>` 的 `PartNumber` 必须对应一个本次分块上传中实际已上传的分块；任一 `PartNumber` 不存在、或 `ETag` 与服务端已记录的不一致，将返回 `InvalidPart`。
- 列表必须按 `PartNumber` 升序排列且不允许重复，否则返回 `InvalidPartOrder`。
- 列表不能为空，否则返回 `InvalidArgument`。
- `PartNumber` 不要求连续，可以跳号（例如只提交编号为1、3、5的分块）。未在列表中出现的已上传分块在合并时将被丢弃。

最小分块大小

除请求体中 `PartNumber` 最大的分块（即合并后对象的最后一块）以外，其余每个分块的大小不得小于1MiB，否则返回 `EntityTooSmall`。最后一块不受此限制，可以小于1MiB。

目标对象键

本接口要求 URL 路径中的 `<ObjectKey>` 与 [InitiateMultipartUpload](#) 调用时使用的对象键完全一致（包括大小写、路径层级与末尾 `/`），不一致时本接口返回 `NoSuchUpload`。

防覆盖与同名冲突

合并阶段会按文件对象语义写入目标对象键，与 [PutObject](#) 一致：

- 携带 `x-cos-forbid-overwrite: true` 且目标文件对象已存在时，返回 `409 FileAlreadyExists`。
- 目标对象键的同名目录对象已存在时，返回 `409 DirAlreadyExists`。
- 目标对象键中的中间路径组件已是文件对象（例如目标键为 `a/b/c` 且 `a/b` 已是文件对象）时，返回 `409 FileAlreadyExists`。

UploadId 有效性

`UploadId` 不存在、已被 [AbortMultipartUpload](#) 终止、或已被本接口完成时，本接口返回 `NoSuchUpload`。

请求

请求语法

```
POST /<ObjectKey>?uploadId=<UploadId> HTTP/1.1
Host: <BucketName-APPID>.cosrapid.<Region>.myqcloud.com
Date: <GMT Date>
Content-Type: application/xml
Content-Length: <Content Length>
Authorization: <Auth String>

<CompleteMultipartUpload>
  <Part>
    <PartNumber>integer</PartNumber>
    <ETag>string</ETag>
  </Part>
  ...
</CompleteMultipartUpload>
```

说明：
Authorization 字段的具体格式见 [公共请求头部](#)。

请求参数

参数名称	描述	类型	是否必选
uploadId	调用 InitiateMultipartUpload 接口初始化分块上传时返回的 <code>UploadId</code>	string	是

请求头部

除 [公共请求头部](#) 外，本接口还支持以下请求头部：

Header 名称	描述	类型	是否必选
x-cos-forbid-overwrite	是否禁止覆盖同名对象，取值为 <code>true</code> 时启用。行为契约详见「防覆盖与同名冲突」	string	否

请求体

提交 `application/xml` 请求数据，包含所有分块信息：

```
<CompleteMultipartUpload>
  <Part>
    <PartNumber>integer</PartNumber>
    <ETag>string</ETag>
  </Part>
  <Part>
    <PartNumber>integer</PartNumber>
    <ETag>string</ETag>
  </Part>
</CompleteMultipartUpload>
```

具体节点说明：

节点名称	父节点	描述	类型	是否必选
CompleteMultipartUpload	无	包含本次合并请求中所有分块的信息	Container	是
Part	CompleteMultipartUpload	单个分块的信息条目	Container	是
PartNumber	CompleteMultipartUpload.Part	分块编号，与上传该分块时使用的 <code>partNumber</code> 一致	integer	是
ETag	CompleteMultipartUpload.Part	分块的 ETag，取值为 <code>UploadPart</code> 或 <code>UploadPartCopy</code> 响应中返回的 <code>ETag</code> ，使用双引号包裹	string	是

响应

响应头部

除 [公共响应头部](#) 外，本接口还会返回以下响应头部：

Header 名称	描述	类型	是否必返
-----------	----	----	------

ETag	合并后对象的 ETag，使用双引号包裹	string	是
Last-Modified	合并后对象的最后修改时间，格式为 RFC 1123 GMT，例如 Wed, 29 May 2019 04:10:12 GMT	string	是
x-cos-hash-crc64ecma	合并后对象的 CRC64-ECMA 校验值，十进制字符串	string	是
x-cos-storage-class	合并后对象的存储类型，固定为 RAPID	string	是

响应体

响应体为 application/xml 格式：

```
<?xml version="1.0" encoding="UTF-8"?>
<CompleteMultipartUploadResult>
  <Bucket>string</Bucket>
  <Key>string</Key>
  <ETag>string</ETag>
</CompleteMultipartUploadResult>
```

具体节点说明：

节点名称	父节点	描述	类型
CompleteMultipartUploadResult	无	保存 CompleteMultipartUpload 结果的所有信息	Container
Bucket	CompleteMultipartUploadResult	目标存储桶名称，形式为 <BucketName-APPID>	string
Key	CompleteMultipartUploadResult	目标对象键	string

ETag	CompleteMultipartUploadResult	合并后对象的 ETag，使用双引号包裹（XML 实体编码形式）	string
------	-------------------------------	---------------------------------	--------

错误码

本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

案例一：完成一次分块上传

请求

```
POST /exampleobject?
uploadId=1585130821cbb7df1d11846c073ad648e8f33b087cec2381df437acdc833cf6
54b9ecc6361 HTTP/1.1
Host: examplebucket-x--1250000000.cosrapid.ap-guangzhou.myqcloud.com
Date: Wed, 29 May 2019 04:10:12 GMT
Content-Type: application/xml
Content-Length: 353
Authorization: q-sign-algorithm=sha1&q-ak=AKIDXXXXXXXXX&q-sign-
time=1559074212;1559077812&q-key-time=1559074212;1559077812&q-header-
list=content-length;content-type;date;host&q-url-param-list=uploadid&q-
signature=xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

<CompleteMultipartUpload>
  <Part>
    <PartNumber>1</PartNumber>
    <ETag>"39270a968a357d24207e9911162507eb"</ETag>
  </Part>
  <Part>
    <PartNumber>2</PartNumber>
    <ETag>"d899fbd1e06109ea2e4550f5751c88d6"</ETag>
  </Part>
  <Part>
    <PartNumber>3</PartNumber>
    <ETag>"762890d6c9a871b7bd136037cb2260cd"</ETag>
  </Part>
</CompleteMultipartUpload>
```

响应

```
HTTP/1.1 200 OK
Content-Type: application/xml
Connection: keep-alive
Date: Wed, 29 May 2019 04:10:12 GMT
ETag: "aa259a62513358f69e98e72e59856d88"
Last-Modified: Wed, 29 May 2019 04:10:12 GMT
x-cos-hash-crc64ecma: 2290339086971918696
x-cos-storage-class: RAPID
x-cos-request-id: NjUwMDAwMDAwMDAwMDAwMDAw

<?xml version="1.0" encoding="UTF-8"?>
<CompleteMultipartUploadResult>
  <Bucket>examplebucket-x--1250000000</Bucket>
  <Key>exampleobject</Key>
  <ETag>"aa259a62513358f69e98e72e59856d88"</ETag>
</CompleteMultipartUploadResult>
```

案例二：使用 x-cos-forbid-overwrite 防覆盖

本案例在请求中携带 `x-cos-forbid-overwrite: true`，要求目标对象已存在时拒绝合并。

请求

```
POST /exampleobject?
uploadId=1585130821cbb7df1d11846c073ad648e8f33b087cec2381df437acdc833cf6
54b9ecc6361 HTTP/1.1
Host: examplebucket-x--1250000000.cosrapid.ap-guangzhou.myqcloud.com
Date: Wed, 29 May 2019 04:10:12 GMT
Content-Type: application/xml
Content-Length: 153
x-cos-forbid-overwrite: true
Authorization: q-sign-algorithm=sha1&q-ak=AKIDXXXXXXXX&q-sign-
time=1559074212;1559077812&q-key-time=1559074212;1559077812&q-header-
list=content-length;content-type;date;host;x-cos-forbid-overwrite&q-url-
param-list=uploadid&q-signature=xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

<CompleteMultipartUpload>
  <Part>
    <PartNumber>1</PartNumber>
```

响应

第131 共324页

AbortMultipartUpload

最近更新时间：2026-10-21 18:34:04

AbortMultipartUpload 接口请求用于舍弃一个分块上传任务并删除其已上传的所有分块。当客户端不再需要继续上传或合并某个 `UploadId` 时，可调用本接口释放该任务在存储桶中占用的存储空间。

注意事项

目标对象键

本接口要求 URL 路径中的 `<ObjectKey>` 与 [InitiateMultipartUpload](#) 调用时使用的对象键完全一致（包括大小写、路径层级与末尾 `/`），不一致时本接口返回 `NoSuchUpload`。

请求

请求语法

```
DELETE /<ObjectKey>?uploadId=<UploadId> HTTP/1.1
Host: <BucketName-APPID>.cosrapid.<Region>.myqcloud.com
Date: <GMT Date>
Authorization: <Auth String>
```

① 说明：

Authorization 字段的具体格式见 [公共请求头部](#)。

请求参数

参数名称	描述	类型	是否必选
uploadId	调用 InitiateMultipartUpload 接口初始化分块上传时返回的 <code>UploadId</code>	string	是

请求头部

此接口仅使用 [公共请求头部](#)，无私有请求头部。

请求体

该请求的请求体为空。

响应

仅返回 **公共响应头部**，本接口无私有响应头部。

请求成功，响应状态码为 204 No Content，响应体为空。

本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。

请求

```
HTTP/1.1 204 No Content
Connection: keep-alive
Date: Tue, 26 Oct 2013 21:22:00 GMT
x-cos-request-id: NjUwMDAwMDAwMDAwMDAwMDAw
```

ListMultipartUploads

最近更新时间：2026-10-21 18:34:05

ListMultipartUploads 接口请求用于查询存储桶中正在进行的分块上传任务。客户端可通过本接口盘点未完成的 UploadId，配合 [AbortMultipartUpload](#) 释放占用的存储空间，或通过 [ListParts](#) 查看具体进度。

注意事项

prefix 与 delimiter 参数

本接口不支持 prefix 和 delimiter 参数。请求中携带任一非空的 prefix 或 delimiter 时，将返回 400 InvalidArgument。如需筛选特定对象的分块上传任务，请在客户端对返回结果自行过滤。

分页

当 IsTruncated 为 true 时，将响应中的 NextKeyMarker 和 NextUploadIdMarker 原样作为下次请求的 key-marker 和 upload-id-marker 传入，直到 IsTruncated 为 false。

请求

请求语法

```
GET /?uploads&max-uploads=<MaxUploads>&key-marker=<KeyMarker>&upload-id-marker=<UploadIdMarker> HTTP/1.1
Host: <BucketName-APPID>.cosrapid.<Region>.myqcloud.com
Date: <GMT Date>
Authorization: <Auth String>
```

说明：
Authorization 字段的具体格式见 [公共请求头部](#)。

请求参数

参数名称	描述	类型	是否必选
uploads	用于声明本次请求为列举分块上传任务，固定为空串。	string	是
max-uploads	单次返回的最大条目数，取值范围为 1 到 1000（包含两端），默认值为 1000。超出范围或无法解析为整数时返回 InvalidArgument	integer	否

key-marker	分页起点的对象键，与 <code>upload-id-marker</code> 配合使用；不指定时从头开始	string	否
upload-id-marker	分页起点的 <code>UploadId</code> ，与 <code>key-marker</code> 配合使用；不指定时从头开始	string	否

请求头部

此接口仅使用 [公共请求头部](#)，无私有请求头部。

请求体

该请求的请求体为空。

响应

响应头部

仅返回 [公共响应头部](#)，本接口无私有响应头部。

响应体

请求成功，返回 `application/xml` 数据。

```
<?xml version="1.0" encoding="UTF-8"?>
<ListMultipartUploadsResult>
  <Bucket>string</Bucket>
  <KeyMarker>string</KeyMarker>
  <UploadIdMarker>string</UploadIdMarker>
  <NextKeyMarker>string</NextKeyMarker>
  <NextUploadIdMarker>string</NextUploadIdMarker>
  <MaxUploads>integer</MaxUploads>
  <IsTruncated>boolean</IsTruncated>
  <Upload>
    <Key>string</Key>
    <UploadId>string</UploadId>
    <StorageClass>RAPID</StorageClass>
    <Initiator>
      <ID>string</ID>
      <DisplayName>string</DisplayName>
    </Initiator>
  </Upload>
</ListMultipartUploadsResult>
```

```
<Owner>
  <ID>string</ID>
  <DisplayName>string</DisplayName>
</Owner>
<Initiated>string</Initiated>
</Upload>
...
</ListMultipartUploadsResult>
```

具体节点说明：

节点名称	父节点	描述	类型
ListMultipartUploadsResult	无	保存 ListMultipartUploads 结果的所有信息	Container
Bucket	ListMultipartUploadsResult	目标存储桶名称，形式为 <BucketName-APP ID>	string
KeyMarker	ListMultipartUploadsResult	本次分页的对象键起点，与请求参数 key-marker 对应；未指定时空	string
UploadIdMarker	ListMultipartUploadsResult	本次分页的 UploadId 起点，与请求参数 upload-id-marker 对应；未指定时空	string
NextKeyMarker	ListMultipartUploadsResult	下一页的对象键起点。仅在 IsTruncated 为 true 时返回	string
NextUploadIdMarker	ListMultipartUploadsResult	下一页的 UploadId 起点。仅在 IsTruncated 为 true 时返回	string
MaxUploads	ListMultipartUploadsResult	本次响应所采用的最大返回条目数，与请求参数 max-uploads 对应	integer
IsTruncated	ListMultipartUploadsResult	响应是否被截断。true 表示还有更多任务未返回，需结合 NextKeyMarker 和 NextUploadIdMarker 继续翻页	boolean
Upload	ListMultipartUploadsResult	描述单个分块上传任务的容器，可能出现多次	Container

Upload 容器内的字段：

节点名称	父节点	描述	类型
Key	Upload	分块上传任务的目标对象键	string
UploadId	Upload	本次分块上传的 UploadId	string
StorageClass	Upload	目标对象的存储类型，当前固定为 RAPID	string
Initiator	Upload	分块上传任务的发起者信息	Container
Owner	Upload	目标对象的所有者信息	Container
Initiated	Upload	分块上传任务的创建时间，UTC 时区，ISO 8601格式，毫秒精度，例如 2023-10-13T04:18:33.000Z	string

Initiator 与 Owner 容器内的字段：

节点名称	父节点	描述	类型
ID	Initiator / Owner	用户的唯一标识	string
DisplayName	Initiator / Owner	用户的显示名称	string

错误码

本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

案例一：列举存储桶中所有未完成的分块上传任务

请求

响应

第138 共324页

案例二：分页列举

第一页请求

第一页响应

第139 共324页

```
<NextKeyMarker>exampleobject2</NextKeyMarker>

<NextUploadIdMarker>1484727158f2b8034e5407d18cbf28e84f754b791ecab607d25a
2e52de9fee641e5f60707c</NextUploadIdMarker>
  <MaxUploads>2</MaxUploads>
  <IsTruncated>true</IsTruncated>
  <Upload>
    <Key>exampleobject1</Key>

    <UploadId>1484726657932bcb5b17f7a98a8cad9fc36a340ff204c79bd2f51e7dddf0b6
d1da6220520c</UploadId>
    <StorageClass>RAPID</StorageClass>
    <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>
    <Initiated>2023-10-13T04:18:33.000Z</Initiated>
  </Upload>
  <Upload>
    <Key>exampleobject2</Key>

    <UploadId>1484727158f2b8034e5407d18cbf28e84f754b791ecab607d25a2e52de9fee
641e5f60707c</UploadId>
    <StorageClass>RAPID</StorageClass>
    <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>
    <Initiated>2023-10-13T04:19:57.000Z</Initiated>
  </Upload>
</ListMultipartUploadsResult>
```


第二页请求

将上一页响应中的 `NextKeyMarker` 和 `NextUploadIdMarker` 原样作为 `key-marker` 和 `upload-id-marker` 传入。

```
GET /?uploads&max-uploads=2&key-marker=exampleobject2&upload-id-
marker=1484727158f2b8034e5407d18cbf28e84f754b791ecab607d25a2e52de9fee641
e5f60707c HTTP/1.1
Host: examplebucket-x--1250000000.cosrapid.ap-guangzhou.myqcloud.com
Date: Fri, 13 Oct 2023 09:13:11 GMT
Authorization: q-sign-algorithm=sha1&q-ak=AKIDXXXXXXX&q-sign-
time=1484727509;32557623509&q-key-time=1484727509;32557623509&q-header-
list=date;host&q-url-param-list=key-marker;max-uploads;upload-id-
marker;uploads&q-signature=xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

第二页响应

```
HTTP/1.1 200 OK
Content-Type: application/xml
Date: Fri, 13 Oct 2023 09:13:11 GMT
x-cos-request-id: NjUwMDAwMDAwMDAwMDAwMDAx

<?xml version="1.0" encoding="UTF-8"?>
<ListMultipartUploadsResult>
  <Bucket>examplebucket-x--1250000000</Bucket>
  <KeyMarker>exampleobject2</KeyMarker>

  <UploadIdMarker>1484727158f2b8034e5407d18cbf28e84f754b791ecab607d25a2e52
de9fee641e5f60707c</UploadIdMarker>
  <MaxUploads>2</MaxUploads>
  <IsTruncated>false</IsTruncated>
  <Upload>
    <Key>exampleobject3</Key>

    <UploadId>1484727270323ddb949d528c629235314a9ead80f0ba5d993a3d76b460e6a9
cceb9633b08e</UploadId>
    <StorageClass>RAPID</StorageClass>
    <Initiator>
      <ID>qcs::cam::uin/1000000000001:uin/1000000000001</ID>
      <DisplayName>1000000000001</DisplayName>
```

```
</Initiator>
  <Owner>
    <ID>qcs::cam::uin/1000000000001:uin/1000000000001</ID>
    <DisplayName>1000000000001</DisplayName>
  </Owner>
<Initiated>2023-10-13T04:25:26.000Z</Initiated>
</Upload>
</ListMultipartUploadsResult>
```

ListParts

最近更新时间：2026-10-21 18:34:05

ListParts 接口请求用于查询某个分块上传任务下已上传成功的所有分块。当客户端在分块上传过程中需要恢复上传进度，或在调用 [CompleteMultipartUpload](#) 之前核对各分块的 `PartNumber` 和 `ETag` 时，可使用本接口进行罗列。

注意事项

UploadId 有效性

`UploadId` 不存在、已被 [AbortMultipartUpload](#) 终止、或已被 [CompleteMultipartUpload](#) 完成时，本接口返回 `NoSuchUpload`。

分页

当已上传分块的总数超过 `max-parts` 限制时，响应中的 `IsTruncated` 为 `true`，并通过 `NextPartNumberMarker` 给出下一页的游标。客户端需在下一次请求中将该值原样作为 `part-number-marker` 传入，以继续翻页。

列举的范围

本接口仅列举处于“已上传成功”状态的分块；尚未完成上传或已被同编号分块覆盖的旧分块不会出现在结果中。

请求

请求语法

```
GET /<ObjectKey>?uploadId=<UploadId>&max-parts=<MaxParts>&part-number-
marker=<PartNumberMarker> HTTP/1.1
Host: <BucketName-APPID>.cosrapid.<Region>.myqcloud.com
Date: <GMT Date>
Authorization: <Auth String>
```

① 说明：

Authorization 字段的具体格式见 [公共请求头部](#)。

请求参数

参数名称	描述	类型	是否必选
------	----	----	------

uploadId	调用 <code>InitiateMultipartUpload</code> 接口初始化分块上传时返回的 <code>UploadId</code>	string	是
max-parts	单次返回的最大分块条目数，取值范围为 1 到 1000（包含两端），默认值为 1000。超出范围或无法解析为整数时返回 <code>InvalidArgument</code>	integer	否
part-number-marker	分页游标，取值为上一次 <code>ListParts</code> 响应中的 <code>NextPartNumberMarker</code> ；不指定时从头开始	string	否

请求头部

此接口仅使用 [公共请求头部](#)，无私有请求头部。

请求体

该请求的请求体为空。

响应

响应头部

仅返回 [公共响应头部](#)，本接口无私有响应头部。

响应体

请求成功，返回 `application/xml` 数据。

```
<?xml version="1.0" encoding="UTF-8"?>
<ListPartsResult>
  <Bucket>string</Bucket>
  <Key>string</Key>
  <UploadId>string</UploadId>
  <StorageClass>RAPID</StorageClass>
  <PartNumberMarker>string</PartNumberMarker>
  <NextPartNumberMarker>string</NextPartNumberMarker>
  <MaxParts>integer</MaxParts>
  <IsTruncated>boolean</IsTruncated>
  <Part>
    <PartNumber>integer</PartNumber>
    <LastModified>string</LastModified>
    <ETag>string</ETag>
```

```
<Size>integer</Size>

</Part>

...

</ListPartsResult>
```

具体节点说明：

节点名称	父节点	描述	类型
ListPartsResult	无	保存 ListParts 结果的所有信息	Container
Bucket	ListPartsResult	目标存储桶名称，形式为 <BucketName-APPID>	string
Key	ListPartsResult	目标对象键	string
UploadId	ListPartsResult	本次分块上传的 UploadId	string
StorageClass	ListPartsResult	目标对象的存储类型，当前固定为 RAPID	string
PartNumberMarker	ListPartsResult	本次请求所使用的分页游标，回显请求参数 part-number-marker；未指定时空	string
NextPartNumberMarker	ListPartsResult	下一页的分页游标，仅在 IsTruncated 为 true 时返回	string
MaxParts	ListPartsResult	本次响应所采用的最大返回条目数，与请求参数 max-parts 对应	integer
IsTruncated	ListPartsResult	响应是否被截断。true 表示还有更多分块未返回，需结合 NextPartNumberMarker 继续翻页	boolean
Part	ListPartsResult	描述单个已上传分块的容器，可能出现多个	Container

Part 容器内的字段：

节点名称	父节点	描述	类型
------	-----	----	----

错误码

实际案例

案例一：列举单个分块上传任务的所有分块

请求

响应

第146 共324页

案例二：分页列举

第一页请求

第一页响应

第二页请求

```
GET /exampleobject?  
uploadId=1585130821cbb7df1d11846c073ad648e8f33b087cec2381df437acdc833cf6
```



```
54b9ecc6361&max-parts=2&part-number-marker=eyJjdXJzb3IiOiJhYmMxMjMifQ
HTTP/1.1
Host: examplebucket-x--1250000000.cosrapid.ap-guangzhou.myqcloud.com
Date: Wed, 25 Mar 2020 10:07:26 GMT
Authorization: q-sign-algorithm=sha1&q-ak=AKIDXXXXXXXX&q-sign-
time=1585130846;1585138046&q-key-time=1585130846;1585138046&q-header-
list=date;host&q-url-param-list=max-parts;part-number-marker;uploadid&q-
signature=xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

第二页响应

```
HTTP/1.1 200 OK
Content-Type: application/xml
Date: Wed, 25 Mar 2020 10:07:26 GMT
x-cos-request-id: NjUwMDAwMDAwMDAwMDAwMDAx

<?xml version="1.0" encoding="UTF-8"?>
<ListPartsResult>
  <Bucket>examplebucket-x--1250000000</Bucket>
  <Key>exampleobject</Key>

  <UploadId>1585130821cbb7df1d11846c073ad648e8f33b087cec2381df437acdc833cf
654b9ecc6361</UploadId>
  <StorageClass>RAPID</StorageClass>
  <PartNumberMarker>eyJjdXJzb3IiOiJhYmMxMjMifQ</PartNumberMarker>
  <MaxParts>2</MaxParts>
  <IsTruncated>false</IsTruncated>
  <Part>
    <PartNumber>3</PartNumber>
    <LastModified>2020-03-25T10:07:13.000Z</LastModified>
    <ETag>"762890d6c9a871b7bd136037cb2260cd"</ETag>
    <Size>1048576</Size>
  </Part>
</ListPartsResult>
```

SDK 文档

简介

最近更新时间：2026-10-21 18:34:03

本文档介绍高性能桶（Rapid Bucket）SDK 的使用方式。目前提供 Go 与 Python 两种语言的高性能桶 SDK，二者均基于对应的腾讯云 COS SDK，面向高性能桶场景提供存储桶与对象的读写能力。

SDK 定位

高性能桶 SDK 在对应语言 COS SDK 的基础上，针对高性能桶的会话鉴权做了适配：在需要时自动完成会话创建、凭证签名与缓存复用，业务代码无需手动管理临时凭证，具体接口、参数与行为以对应高性能桶 SDK 文档为准。

- **Go SDK**：基于 github.com/tencentyun/cos-go-sdk-v5，通过 `RapidCredentialTransport` 使用基础凭证完成自动会话鉴权。
- **Python SDK**：基于 [cos-python-sdk-v5](https://github.com/tencentcloud-sdk/python-tencentcloud-sdk/blob/master/doc/faq.md)，`CosS3Client` 在识别到高性能桶后自动完成 `CreateSession`、签名与按桶缓存复用。

文档结构

本 SDK 文档涵盖以下操作分类：

- **存储桶操作**：创建存储桶、删除存储桶、查询存储桶、查询存储桶列表、创建访问会话。
- **对象操作**：上传对象、下载对象、删除对象、复制对象等。
- **分块上传操作**：适用于大文件的分块上传。
- **存储桶策略**：设置存储桶策略、获取存储桶策略、删除存储桶策略。
- **Python 高级传输接口**：上传本地文件、下载到本地文件、高级复制对象、上传可读流。
- **Python 访问授权**：预签名 URL 的生成、使用与有效期。

开始使用

建议先阅读对应语言的快速入门，了解环境要求、SDK 安装、鉴权说明与客户端初始化方式，再根据需要查阅各接口的详细文档。SDK 调用返回的错误处理方式请参见异常处理。

- Go SDK：[Go SDK 快速入门](#)、[Go SDK 异常处理](#)
- Python SDK：[Python SDK 快速入门](#)、[Python SDK 异常处理](#)

Python 的自动传输与临时授权用法：

- [Python SDK 上传本地文件](#)、[Python SDK 下载到本地文件](#)
- [Python SDK 高级复制对象](#)、[Python SDK 上传可读流](#)
- [Python SDK 预签名 URL](#)

Python SDK

快速入门

最近更新时间：2026-10-21 18:34:03

本文档介绍如何使用 Python SDK 访问高性能桶（Rapid Bucket），完成初始化客户端、创建访问会话以及上传下载对象等基础操作。

环境要求

- 高性能桶 SDK 已验证的 Python 版本为2.7、3.6、3.7；生产环境的 Python 与依赖版本请以所获取 SDK 的版本说明为准。
- 已开通高性能桶服务，并已创建高性能桶；也可以按 [Python SDK 创建存储桶](#) 通过 SDK 创建。高性能桶的完整名称形如 `<BucketName>-x--<APPID>`，例如 `examplebucket-x--1250000000`。
- 已获取访问密钥 `SecretId` 和 `SecretKey`。建议使用子账号密钥并遵循最小权限原则。

安装 SDK

本文适用于高性能桶 Python SDK 的内部测试版本（软件包版本 1.9.45）。请使用测试交付的专用源码包；所用版本需包含 `EnableRapidDomain`、`create_session`、`rename_object`，以及 `create_bucket` 的 `VpcId`、`SubnetId`、`Zone` 参数。

普通 COS SDK 的公开 PyPI 安装包不能作为本文功能的安装依据。取得匹配的 SDK 源码包后，解压并进入包含 `setup.py` 的目录，使用当前 Python 环境安装：

```
python -m pip install .
```

安装后可先进行零网络能力检查，避免误用同名的普通 COS SDK 包：

```
# -*- coding: utf-8 -*-
import inspect
from qcloud_cos import CosConfig, CosS3Client

get_args = getattr(inspect, 'getfullargspec', None) or
inspect.getargspec
checks = (
    (CosConfig.__init__, ('EnableRapidDomain',)),
    (CosS3Client.create_bucket, ('VpcId', 'SubnetId', 'Zone')),
)
for method, required in checks:
```

```
if not set(required).issubset(get_args(method).args):
    raise RuntimeError('SDK 与高性能桶测试交付版本不匹配')
for name in ('create_session', 'rename_object'):
    if not hasattr(CosS3Client, name):
        raise RuntimeError('SDK 缺少高性能桶接口: ' + name)
print('Rapid SDK API check OK')
```

鉴权说明

访问高性能桶的数据接口时，统一使用 `session` 临时凭证签名。`CosS3Client` 在识别到高性能桶后会自动完成这一步：在需要时调用 `CreateSession` 换取临时凭证并完成签名，临时凭证在有效期内按桶复用。对于可安全重试的请求，收到 403 时 SDK 最多刷新凭证并重试一次，业务代码通常无需手动管理会话与临时凭证。只需在初始化客户端时打开 `EnableRapidDomain`（或让 Endpoint/Domain 符合 Rapid 域名），后续按本文列出的高性能桶接口与参数调用。控制面接口（创建/删除/查询桶、存储桶策略、`CreateSession`）仍使用基础凭证签名，不走 `session`。

初始化客户端

示例从环境变量 `COS_SECRET_ID`、`COS_SECRET_KEY` 读取基础密钥。使用临时访问密钥时，还需设置 `COS_SECURITY_TOKEN`。请通过安全的配置方式提供这些值，不要将密钥或会话令牌写入源码或日志。当前内部测试接入使用 HTTP 默认端口 80，示例显式设置 `Scheme='http'`；SDK 的默认值是 HTTPS，不能省略此配置。访问高性能桶时，建议设置 `EnableRapidDomain=True`，SDK 会把 Bucket Endpoint 设为 `cosrapid.<Region>.myqcloud.com`，把 `ServiceDomain` 设为 `service.cosrapid.<Region>.myqcloud.com`：

```
# -*- coding=utf-8 -*-
import os
from qcloud_cos import CosConfig, CosS3Client

secret_id = os.environ['COS_SECRET_ID']
secret_key = os.environ['COS_SECRET_KEY']
region = 'ap-guangzhou'

# 高性能桶经 VPC 内网 HTTP 80 端口接入，SDK 的网关 DNS 负载均衡也仅对 HTTP 请求生效；SDK 默认 https，此处需显式指定 http。
config = CosConfig(
    Scheme='http',
    Region=region,
    SecretId=secret_id,
    SecretKey=secret_key,
```

```
Token=os.environ.get('COS_SECURITY_TOKEN'),
EnableRapidDomain=True)
client = CosS3Client(config)

# 高性能桶完整名称, 形如 <BucketName>-x--<APPID>。
bucket = 'examplebucket-x--1250000000'
```

字段说明:

参数	描述
Region	高性能桶所在地域, 例如 <code>ap-guangzhou</code>
Scheme	当前内部测试入口使用 <code>http</code> , 需显式设置
SecretId / SecretKey	主账号或子账号的访问密钥
EnableRapidDomain	为 <code>True</code> 时使用 Rapid Bucket Endpoint 与 ServiceDomain
EnableSessionAuth	<code>None</code> (默认) 自动判断; <code>True</code> 强制对高性能桶开启 session; <code>False</code> 关闭 (数据面会本地报错)
SessionMode	<code>CreateSession</code> 默认权限, <code>ReadWrite</code> (默认) 或 <code>ReadOnly</code>
CreateSessionTimeout	<code>CreateSession</code> 独立超时秒数, 默认 30

如需显式配置域名, `Endpoint` 与 `Domain` 的格式不同, 二选一即可:

- `Endpoint` 不含桶名前缀, 格式为 `cosrapid.<Region>.myqcloud.com`, 例如 `cosrapid.ap-guangzhou.myqcloud.com`; SDK 会根据 `Bucket` 补上桶名。
- `Domain` 是完整的桶域名, 格式为 `<Bucket>.cosrapid.<Region>.myqcloud.com`, 例如 `examplebucket-x--1250000000.cosrapid.ap-guangzhou.myqcloud.com`, 必须与请求的 `Bucket` 对应。

两者均只填写域名, 不包含协议、路径或显式端口; 域名中的地域须与 `Region` 一致。不要把完整桶域名传给 `Endpoint`。正确配置上述域名后, SDK 会自动启用高性能桶会话鉴权。

完整示例

以下示例演示初始化客户端后完成一次上传与下载:

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

```
from qcloud_cos import CosConfig, CosS3Client

# 高性能桶经 VPC 内网 HTTP 80 端口接入, SDK 的网关 DNS 负载均衡也仅对 HTTP 请求生效; SDK 默认 https, 此处需显式指定 http。
config = CosConfig(
    Scheme='http',
    Region='ap-guangzhou',
    SecretId=os.environ['COS_SECRET_ID'],
    SecretKey=os.environ['COS_SECRET_KEY'],
    Token=os.environ.get('COS_SECURITY_TOKEN'),
    EnableRapidDomain=True)
client = CosS3Client(config)

bucket = 'examplebucket-x--1250000000'
key = 'quickstart/hello.txt'

# 上传对象。首次对高性能桶发起请求时, SDK 会自动创建 session 并签名。
client.put_object(Bucket=bucket, Key=key, Body=b'hello rapid bucket')
print('PutObject OK')

# 下载对象。
resp = client.get_object(Bucket=bucket, Key=key)
body = resp['Body'].get_raw_stream().read()
print('GetObject OK: %s' % body)
```

后续操作

常用高级接口:

场景	接口文档
上传本地文件, 自动分块与符合条件的续传	Python SDK 上传本地文件
并发下载到本地并管理续传记录	Python SDK 下载到本地文件
同桶同地域自动选择复制方式	Python SDK 高级复制对象
读取二进制流并上传	Python SDK 上传可读流
生成带会话签名的临时访问链接	Python SDK 预签名 URL

- 存储桶操作: 创建存储桶、删除存储桶、查询存储桶、查询存储桶列表、创建访问会话。

- 对象操作：上传对象、下载对象、删除对象、复制对象、重命名对象等。
- 分块上传操作：适用于大文件的分块上传。
- 存储桶策略：设置、获取、删除存储桶策略。

各接口的详细用法请参见对应文档。SDK 调用返回的错误处理方式请参见 [Python SDK 异常处理](#)。

高级传输接口

高级复制对象

最近更新时间：2026-10-21 18:34:06

高级复制文件对象（`copy`）。SDK 查询源文件后选择直接复制或分块复制，分块路径自动组织上传分块与最终完成请求。

注意事项

- 操作的存储桶应处于 Available 状态；传输失败可能表现为服务端、客户端或本地文件异常，处理方式见 [Python SDK 异常处理](#)。
- 仅支持同桶、同地域的文件对象复制；源 Bucket 与目标 Bucket 相同，CopySource.Region 与客户端 Region 相同。对象键尾斜杠严格区分类型，不支持目录复制；源类型不匹配按不存在处理。由于包含源 HEAD，高级接口可能先返回 HTTP 404，而不是低层复制接口的参数错误。
- 默认复制阈值为 5GB；按本文示例不传 StorageClass 时，当前高性能桶通常在小于阈值时直接复制，达到阈值时分块复制。不要将 PartSize 当作选择分块复制的阈值。
- 显式传入与源一致的 StorageClass 可能优先选择直接复制，不受大小阈值控制；超大文件可能因此超过直接复制的 5GB 限制。需要可靠控制每个步骤时，请使用手动分块复制接口。
- 分块复制期间请保持源文件不变；本接口不提供对象版本快照。高性能桶不支持 CopySource.VersionId。
- `ForbidOverwrite='true'` 同时用于直接复制和最终 Complete；省略或传 "false" 允许覆盖已有文件。防覆盖失败时不能通过删除参数重试来保持原保护要求。
- `CopyStatus` 的元数据复制方式仅用于直接复制路径。分块路径的目标元数据来自本次初始化请求，不自动继承源对象元数据；需要保留时先查询源元数据，再将实际需要的 ContentType、Metadata 等显式传入。直接复制时若要用本次参数替换源元数据，还需设置 CopyStatus='Replaced'。
- 本接口不自动恢复上次失败的复制任务，也没有本地进度回调参数。分块失败可能遗留未完成任务；Complete 抛错时会尝试终止该任务，但不能保证任意失败都自动清理。响应丢失时先核对目标对象状态，放弃任务时按明确的 UploadId 清理。

请求

方法原型

```
def copy(self, Bucket, Key, CopySource, CopyStatus='Copy', PartSize=10,
MAXThread=5, **kwargs)
```

参数说明

参数	说明	类型	是否必选
Bucket	高性能桶完整名称，例如 <code>examplebucket-x--1250000000</code>	string	是
Key	文件对象键，不以 <code>/</code> 结尾	string	是
CopySource	字典，含 Bucket、Key、Region；源键不以 <code>/</code> 结尾	dict	是
CopyStatus	Copy（默认）或 Replaced；仅直接复制路径的元数据策略	string	否
PartSize	分块复制大小，正整数，默认 10，单位 MB；每块不超过 5GB	int	否
MAXThread	分块复制并发线程数，正整数，默认 5	int	否
ForbidOverwrite	关键字参数，仅 "true" / "false"	string	否
ContentType / Metadata	显式设置的目标元数据；自定义元数据键以 <code>x-cos-meta-</code> 开头	string / dict	否
kwargs	可选请求参数，使用本文列出的参数；按本页示例不传 <code>StorageClass</code>	dict	否

响应

返回字典，可读取 `ETag`；其他字段随直接复制或 Complete 路径而异，不承诺返回可访问的 Location URL。

错误码

本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

```
# -*- coding: utf-8 -*-
import os
from qcloud_cos import CosConfig, CosS3Client

# 高性能桶经 VPC 内网 HTTP 80 端口接入，SDK 的网关 DNS 负载均衡也仅对 HTTP 请求生效；SDK 默认 https，此处需显式指定 http。
config = CosConfig(
    Scheme='http', Region='ap-guangzhou',
    SecretId=os.environ['COS_SECRET_ID'],
```

```
SecretKey=os.environ['COS_SECRET_KEY'],
Token=os.environ.get('COS_SECURITY_TOKEN'),
EnableRapidDomain=True)
client = CosS3Client(config)
bucket = 'examplebucket-x--1250000000'

result = client.copy(
    Bucket=bucket, Key='copies/report.bin',
    CopySource={'Bucket': bucket, 'Key': 'uploads/report.bin', 'Region':
'ap-guangzhou'},
    PartSize=8, MAXThread=4, ForbidOverwrite='true')
print('copy completed, ETag: %s' % result['ETag'])
# 本例复制文件内容；如需保留源元数据，请按本页注意事项显式设置。
```

下载到本地文件

最近更新时间：2026-10-21 18:34:06

下载对象到本地文件（`download_file`）。SDK 先查询对象大小，小文件直接下载，大文件按字节范围并发下载，并通过本地记录支持符合条件的断点续传。

注意事项

- 操作的存储桶应处于 Available 状态；传输失败可能表现为服务端、客户端或本地文件异常，处理方式见 [Python SDK 异常处理](#)。
- 大小不超过 `PartSize × 1024 × 1024` 字节时直接下载；超过时走分块下载。分块大小至少为 1MB，并可能为满足 `MaxPartCount` 而增大。
- 对象键末尾 / 严格区分类型。源键类型不匹配按不存在处理，HTTP 404；HEAD 无错误正文时 SDK 错误码为 `NoSuchResource`。本页示例面向文件对象；读取目录请参见 [Python SDK 下载对象](#)。
- 多次请求及续传期间必须保持源对象不变。高性能桶没有对象版本保护；本接口不提供下载快照，CRC 校验也不是条件读取或版本锁定。
- 续传需要保留下载记录和对应该临时文件，并使用相同 Bucket、Key、目标绝对路径和记录目录重新调用。SDK 会核对源对象的 ETag、Last-Modified、Content-Length；记录不匹配或临时文件缺失时重新下载，不保证复用全部已下载内容。
- 默认记录目录为用户主目录下的 `.cos_download_tmp_file`；可用 `DumpRecordDir` 指定。记录目录与目标父目录需可写；请先创建目标父目录。同一目标文件不要同时运行多个下载任务。
- 默认先写临时文件，再替换目标路径，已有同名本地文件可能被替换。`DisableTempDestFilePath=True` 仅使直接下载分支直接写目标，失败可能留下部分内容；分块下载仍使用临时文件。
- `EnableCRC=True` 仅在分块下载路径执行最终 CRC64 校验，要求响应提供校验值。校验时会一次性将整个本地文件读入内存，增加内存和磁盘读取开销；校验发生在目标文件替换后，校验失败并不保证保留原目标文件。
- `progress_callback(finished, total)` 仅在分块路径调用；已恢复分块计入进度。小文件路径不执行此回调或高级 CRC 校验。进度满额不等于方法成功，必须等待正常返回。
- 分块传输失败时保留记录和临时文件供重试；成功后移除记录。遇到 CRC 错误先停止使用目标文件，核对源对象并处理对应记录后重新下载，不要仅凭本地文件存在判断成功。
- 本接口用于完整对象下载，不传 `Range` 来下载局部内容；范围读取请使用 [Python SDK 下载对象](#)。传入 `VersionId` 会在本地报错。

请求

方法原型

```
def download_file(self, Bucket, Key, DestFilePath, PartSize=20,
MAXThread=5, EnableCRC=False,
                    progress_callback=None, DumpRecordDir=None,
KeySimplifyCheck=True,
                    DisableTempDestFilePath=False, MaxPartCount=10000,
**Kwargs)
```

参数说明

参数	说明	类型	是否必选
Bucket	高性能桶完整名称，例如 <code>examplebucket-x--1250000000</code>	string	是
Key	文件对象键，不以 <code>/</code> 结尾	string	是
DestFilePath	本地目标文件路径，父目录需存在	string	是
PartSize	直接下载阈值和分块大小，正整数，默认 20，单位 MB	int	否
MAXThread	分块并发线程数，正整数，默认 5	int	否
MaxPartCount	分块下载最大分块数，正整数，默认 10000	int	否
EnableCRC	分块下载完成后校验 CRC64，默认 False	bool	否
progress_callback	分块下载进度回调，默认 None	callable	否
DumpRecordDir	续传记录目录；不指定时使用上述默认目录	string	否
KeySimplifyCheck	对象键安全检查，保持默认 True	bool	否
DisableTempDestFilePath	直接下载时是否直接写目标，建议保持默认 False	bool	否
Kwargs	可选参数请遵循本文；不传版本或范围参数	dict	否

响应

成功返回 `None`，内容写入 `DestFilePath`。发生异常时不要将目标文件视为已经完整下载并通过校验。

错误码

本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

```
# -*- coding: utf-8 -*-
import os
from qcloud_cos import CosConfig, CosS3Client

# 高性能桶经 VPC 内网 HTTP 80 端口接入，SDK 的网关 DNS 负载均衡也仅对 HTTP 请求生效；SDK 默认 https，此处需显式指定 http。
config = CosConfig(
    Scheme='http', Region='ap-guangzhou',
    SecretId=os.environ['COS_SECRET_ID'],
    SecretKey=os.environ['COS_SECRET_KEY'],
    Token=os.environ.get('COS_SECURITY_TOKEN'),
    EnableRapidDomain=True)
client = CosS3Client(config)
bucket = 'examplebucket-x--1250000000'

def progress(finished, total):
    print('processed bytes: %s / %s' % (finished, total))

client.download_file(
    Bucket=bucket, Key='uploads/report.bin', DestFilePath='downloaded-
report.bin',
    PartSize=8, MAXThread=4, EnableCRC=True,
    DumpRecordDir='./download-records', progress_callback=progress)
print('download completed')
# 只有分块路径执行上述高级 CRC 校验与进度回调。
```

上传本地文件

最近更新时间：2026-10-21 18:34:06

上传本地文件（`upload_file`）。SDK 根据文件大小选择直接上传或分块上传，并在分块路径中尝试复用符合条件的未完成上传任务。调用会等待上传完成后返回，数据请求由 SDK 自动进行会话鉴权。

注意事项

- 操作的存储桶应处于 Available 状态；传输失败可能表现为服务端、客户端或本地文件异常，处理方式见 [Python SDK 异常处理](#)。
- 文件大小不超过 `PartSize × 1024 × 1024` 字节时直接上传，超过时分块上传。`PartSize` 为正整数，按每 MB 为 `1024 × 1024` 字节设置；分块不得超过 5GB，SDK 会在需要时调整分块大小以控制分块数。请为超大文件选取合适的分块大小。
- 本页用于上传文件对象。对象键末尾 `/` 严格区分文件和目录；创建目录请使用 [Python SDK 上传对象](#) 的空内容示例。文件不能覆盖同名目录。
- 需要防覆盖时，每次调用（包括失败后重试）均显式传入 `ForbidOverwrite='true'`。直接上传与最终完成分块上传都会携带本次防覆盖条件；已有同名目标时返回 `409 FileAlreadyExists`。
- 自动续传按桶和完整对象键查找未完成任务，校验已上传分块的大小和 ETag 后才复用；不满足条件时创建新任务。上传和重试期间请保持本地文件、对象键及分块配置不变，不要并发操作同一目标的上传任务。
- 续传复用任务初始化时的对象元数据；更改本次 `ContentType` 或 `Metadata` 不会重建已有任务的元数据。需要重新初始化时，先确认并终止相应的旧任务。
- 当前自动续传仅支持整数形式的已上传分块分页游标。遇到游标格式不兼容时，请核对未完成任务，必要时使用 [Python SDK 列举已上传分块](#) 与手动分块接口原样回传游标。
- 分块失败或 Complete 失败不会在本接口中自动终止上传任务。需要放弃时，先 [Python SDK 列举分块上传](#)，确认目标 Key 和 UploadId，再 [Python SDK 终止分块上传](#)。响应丢失时先核对最终对象状态。
- `progress_callback(finished, total)` 接收累计处理字节数和文件总字节数，仅分块路径调用；续传跳过的已完成分块也计入进度。回调应简短且不抛异常；进度达到总量不等于 Complete 已成功，以方法正常返回为准。
- 高性能桶不支持对象版本与服务端回调，传入 `VersionId`、`Callback`、`CallbackVar` 会在本地报错。

请求

方法原型

```
def upload_file(self, Bucket, Key, LocalFilePath, PartSize=1,
MAXThread=5, EnableMD5=False,
progress_callback=None, **kwargs)
```

参数说明

参数	说明	类型	是否必选
Bucket	高性能桶完整名称，例如 <code>examplebucket-x--1250000000</code>	string	是
Key	文件对象键，不以 <code>/</code> 结尾	string	是
LocalFilePath	可读取的本地文件路径	string	是
PartSize	分块大小及直接上传阈值，默认 1，单位 MB	int	否
MAXThread	分块上传并发线程数，正整数，默认 5	int	否
EnableMD5	计算并发送 Content-MD5，直接上传及新上传分块均适用，默认 False	bool	否
progress_callback	本地进度回调，仅分块路径调用，默认 None	callable	否
ForbidOverwrite	关键字参数；仅字符串 "true" / "false"，省略允许覆盖文件	string	否
ContentType / Metadata	文件内容类型与自定义元数据；Metadata 键以 <code>x-cos-meta-</code> 开头	string / dict	否
kwargs	可选请求参数，使用本文列出的参数	dict	否

响应

返回字典。直接上传返回响应头，分块上传返回完成请求的响应头及结果字段；可读取 `ETag`。分块对象的 ETag 不应当作完整文件内容的 MD5。

错误码

本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

```
# -*- coding: utf-8 -*-
import os
from qcloud_cos import CosConfig, CosS3Client
```

高性能桶经 VPC 内网 HTTP 80 端口接入, SDK 的网关 DNS 负载均衡也仅对 HTTP 请求生效; SDK 默认 https, 此处需显式指定 http。

```
config = CosConfig(  
    Scheme='http', Region='ap-guangzhou',  
    SecretId=os.environ['COS_SECRET_ID'],  
    SecretKey=os.environ['COS_SECRET_KEY'],  
    Token=os.environ.get('COS_SECURITY_TOKEN'),  
    EnableRapidDomain=True)  
client = CosS3Client(config)  
bucket = 'examplebucket-x--1250000000'  
  
def progress(finished, total):  
    print('processed bytes: %s / %s' % (finished, total))  
  
# 使用已存在的本地文件; 重试时保留相同配置和 ForbidOverwrite。  
result = client.upload_file(  
    Bucket=bucket, Key='uploads/report.bin', LocalFilePath='report.bin',  
    PartSize=8, MAXThread=4, EnableMD5=True,  
    progress_callback=progress, ForbidOverwrite='true')  
print('upload completed, ETag: %s' % result['ETag'])
```


上传可读流

最近更新时间：2026-10-21 18:34:06

上传可读二进制流（`upload_file_from_buffer`）。适用于内容通过 `read(size)` 提供、无需先全部载入内存的场景；SDK 使用缓冲与并发分块上传，调用完成后返回结果。

注意事项

- 操作的存储桶应处于 Available 状态；传输失败可能表现为服务端、客户端或本地文件异常，处理方式见 [Python SDK 异常处理](#)。
- Body 必须实现 `read(size)`，返回字节串。仅在到达 EOF 时短读；若数据源可能暂时只返回少量内容，请先适配成满足该约定的流，避免短读被当作文件结束。空字节串表示 EOF。
- 第一次读取不足一个分块时直接上传，否则进入分块上传；与 `upload_file` 的“大小不超过阈值直接上传”边界不同。
- 流从当前位置开始读取；调用方负责关闭流，不要同时在其他线程读取或改变位置。本接口没有自动断点续传；失败后再次调用前应重新打开流或恢复到原点。
- 本页上传文件对象，对象键末尾 / 严格区分类型，不将流上传当作目录创建接口。文件不能覆盖同名目录，目录创建见 [Python SDK 上传对象](#)。
- ForbidOverwrite 仅接受 "true" / "false"；设为 "true" 时，直接上传和最终 Complete 均保留本次防覆盖条件，已有同名目标返回 409 FileAlreadyExists。
- 每个分块不超过 5GB，总分块数不超过 10000。本接口不根据总长度自动增大分块，请根据预计数据量选择 PartSize，不用于无限流。
- MaxBufferSize 用于限制等待调度的分块数量，不是进程内存硬上限；在途请求、当前读取块及其他开销也占内存。建议设置为不小于 PartSize 的整数倍。
- 本接口没有 EnableMD5 或 progress_callback 命名参数；不要照搬 `upload_file` 的参数。
- 分块传输或读取失败可能遗留未完成任务；Complete 抛错时会尝试终止任务。不能假定所有失败都已自动清理；重试或手动清理前先核对最终对象和相应 UploadId。
- 高性能桶不支持对象版本与服务端回调，传入 `VersionId`、`Callback`、`CallbackVar` 会在本地报错。

请求

方法原型

```
def upload_file_from_buffer(self, Bucket, Key, Body, MaxBufferSize=100,
                             PartSize=10, MAXThread=5, **kwargs)
```

参数说明

参数	说明	类型	是否必选
Bucket	高性能桶完整名称，例如 <code>examplebucket-x--1250000000</code>	string	是
Key	文件对象键，不以 <code>/</code> 结尾	string	是
Body	具有 <code>read(size)</code> 方法的二进制流	file-like	是
MaxBufferSize	排队缓冲预算，正整数，默认 100，单位 MB	int	否
PartSize	分块大小，正整数，默认 10，单位 MB	int	否
MAXThread	分块并发线程数，正整数，默认 5	int	否
ForbidOverwrite	关键字参数，仅 "true" / "false"	string	否
ContentType / Metadata	内容类型与自定义元数据；元数据键以 <code>x-cos-meta-</code> 开头	string / dict	否
kwargs	可选请求参数，使用本文列出的参数	dict	否

响应

返回字典，可读取 `ETag`。直接上传返回响应头，分块上传返回 `Complete` 结果；分块对象 `ETag` 不等同于完整内容 MD5。

错误码

本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

```
# -*- coding: utf-8 -*-
import os
from qcloud_cos import CosConfig, CosS3Client

# 高性能桶经 VPC 内网 HTTP 80 端口接入，SDK 的网关 DNS 负载均衡也仅对 HTTP 请求生效；SDK 默认 https，此处需显式指定 http。
config = CosConfig(
    Scheme='http', Region='ap-guangzhou',
    SecretId=os.environ['COS_SECRET_ID'],
    SecretKey=os.environ['COS_SECRET_KEY'],
```

```
Token=os.environ.get('COS_SECURITY_TOKEN'),
    EnableRapidDomain=True)
client = CosS3Client(config)
bucket = 'examplebucket-x--1250000000'

with open('report.bin', 'rb') as source:
    result = client.upload_file_from_buffer(
        Bucket=bucket, Key='streams/report.bin', Body=source,
        PartSize=8, MaxBufferSize=32, MAXThread=4,
        ForbidOverwrite='true', ContentType='application/octet-stream')
print('stream upload completed, ETag: %s' % result['ETag'])
```

访问授权

预签名 URL

最近更新时间：2026-10-21 18:34:06

为指定对象请求生成带会话签名的 URL。接收方可在授权与有效期允许的范围内使用该 URL 发起请求，无需持有基础密钥。生成 URL 可能先发起 CreateSession 请求，并非始终是纯本地操作。

注意事项

- URL 生成阶段可能因配置、凭据或有效期要求抛出异常，处理方式见 [Python SDK 异常处理](#)。
- URL 含会话令牌及签名，应作为临时凭据保管，通过受控渠道交付，不输出完整 URL 到终端、日志或错误报告。HTTP 库异常也可能包含 URL，记录异常前应脱敏。
- GET / HEAD 默认使用独立的 ReadOnly 会话，不复用数据请求中的 ReadWrite 凭据。PUT 请求需要 ReadWrite 权限；只读会话不能用于上传。生成成功不代表对象存在，也不代表文件内容已上传。
- Expired 单位为秒，必须为正。SDK 会要求会话凭据剩余寿命覆盖请求的签名时长；无法获得足够长的有效凭据时本地报错，不会静默缩短 Expired。实际访问还受凭据有效性和服务端授权状态约束，不保证一直可用到设置的最后一秒。
- URL 在有效期和权限允许范围内可以重复使用，不是一次性链接。
- URL 生成后不会随 SDK 刷新凭据而自动更新。到期后需重新生成并交付；不要手工修改 URL 中的有效期或令牌。
- 必须按签名时的方法、对象路径、域名、查询参数及请求头发起请求，不随意追加参数、重编码 URL 或替换成 IP 地址。签入 Headers 的字段须由接收方按相同值携带；仅持有 URL 并不自动补齐这些头。
- Headers 使用真实 HTTP 头名，例如 x-cos-forbid-overwrite；这里不是 put_object 的 ForbidOverwrite 等 SDK 参数名称。上传防覆盖需在生成 URL 时签入该头，并在实际 PUT 请求中携带。
- 对象键末尾 / 严格区分类型。GET / HEAD 类型不匹配返回 HTTP 404，DELETE 类型不匹配按对象不存在返回 204。PUT 文件不能覆盖同名目录；创建目录需使用尾斜杠键、空请求体及 Content-Length: 0，不附加内容类型或自定义元数据。
- 高性能桶不支持对象版本。Params 中的 versionId 或不支持的版本、回调参数会在本地报错。签名不会使下载条件头具备服务端未实现的一致性保护。
- 本页展示 GET、HEAD、PUT、DELETE 对象请求。Headers/Params 应遵循对应已文档化接口的契约；生成 URL 不代替服务端对请求的校验。

请求

调用形式

```
url = client.get_session_presigned_url(
```

```
Bucket=bucket, Key=key, Method='GET', Expired=300)
```

参数说明

参数	说明	类型	是否必选
Bucket	高性能桶完整名称，例如 <code>examplebucket-x--1250000000</code>	string	是
Key	对象键，末尾 <code>/</code> 严格区分文件与目录	string	是
Method	HTTP 方法，本页使用 GET、HEAD、PUT、DELETE	string	是
Expired	请求的签名有效期，正整数，默认 300 秒	int	否
Params	签入 URL 的查询参数，默认空字典；令牌由 SDK 添加	dict	否
Headers	需签名的 HTTP 请求头；使用真实头名，默认空字典	dict	否
Mode	ReadOnly 或 ReadWrite；GET/HEAD 默认 ReadOnly，其余缺省取客户端 SessionMode	string	否
Sign Host	是否签入 Host，建议显式 True 并保持原域名	bool	否

也可使用通用入口，Rapid 桶上会采用相同的会话预签名流程：

```
url = client.get_presigned_url(
    Bucket=bucket, Key=key, Method='GET', Expired=300)
url = client.get_presigned_download_url(
    Bucket=bucket, Key=key, Expired=300)
```

通用入口不接收 Mode；需要显式选择权限模式时使用 `get_session_presigned_url`。普通桶的通用预签名行为保持原有规则。

响应

返回字符串 URL。GET/HEAD 等签名仅授权相应请求，不下载内容；PUT 签名不会发送请求体。接收方必须能访问 URL 对应的服务入口。

错误码

本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

生成下载 URL，并在当前进程中使用；实际交付时通过受控渠道传递 url，不记录其完整内容：

```
# -*- coding: utf-8 -*-
import os
from qcloud_cos import CosConfig, CosS3Client

# 高性能桶经 VPC 内网 HTTP 80 端口接入，SDK 的网关 DNS 负载均衡也仅对 HTTP 请求生效；SDK 默认 https，此处需显式指定 http。
config = CosConfig(
    Scheme='http', Region='ap-guangzhou',
    SecretId=os.environ['COS_SECRET_ID'],
    SecretKey=os.environ['COS_SECRET_KEY'],
    Token=os.environ.get('COS_SECURITY_TOKEN'),
    EnableRapidDomain=True)
client = CosS3Client(config)
bucket = 'examplebucket-x--1250000000'

import requests

url = client.get_session_presigned_url(
    Bucket=bucket, Key='uploads/report.bin', Method='GET',
    Expired=60, SignHost=True)
response = None
try:
    response = requests.get(url, stream=True, timeout=30,
allow_redirects=False)
except requests.RequestException:
    pass # 请求库异常可能携带完整 URL，示例不将其输出。
if response is None:
    raise RuntimeError('presigned download request failed')
try:
    if response.status_code != 200:
        raise RuntimeError('download HTTP status: %s' %
response.status_code)
    with open('presigned-download.bin', 'wb') as output:
```

```
        for chunk in response.iter_content(chunk_size=1024 * 1024):
            if chunk:
                output.write(chunk)
    finally:
        response.close()
    print('presigned download completed')
```

生成带防覆盖请求头的上传 URL，使用方须携带同一份 headers：

```
# -*- coding: utf-8 -*-
import os
from qcloud_cos import CosConfig, CosS3Client

# 高性能桶经 VPC 内网 HTTP 80 端口接入，SDK 的网关 DNS 负载均衡也仅对 HTTP 请求生效；SDK 默认 https，此处需显式指定 http。
config = CosConfig(
    Scheme='http', Region='ap-guangzhou',
    SecretId=os.environ['COS_SECRET_ID'],
    SecretKey=os.environ['COS_SECRET_KEY'],
    Token=os.environ.get('COS_SECURITY_TOKEN'),
    EnableRapidDomain=True)
client = CosS3Client(config)
bucket = 'examplebucket-x--1250000000'

import requests

headers = {
    'x-cos-forbid-overwrite': 'true',
    'Content-Type': 'application/octet-stream',
}
url = client.get_session_presigned_url(
    Bucket=bucket, Key='presigned/new-file.bin', Method='PUT',
    Expired=60, Mode='ReadWrite', SignHost=True, Headers=headers)
response = None
try:
    response = requests.put(
        url, data=b'hello rapid bucket', headers=headers,
        timeout=30, allow_redirects=False)
except requests.RequestException:
```

```
pass # 在 except 块外报告错误，避免异常链输出原 URL。
if response is None:
    raise RuntimeError('presigned upload request failed')
try:
    if response.status_code != 200:
        raise RuntimeError('upload HTTP status: %s' %
response.status_code)
finally:
    response.close()
print('presigned upload completed')
```


存储桶操作

查询存储桶列表

最近更新时间：2026-10-21 18:34:05

列出调用方主账号在指定地域下的所有高性能桶（GET Service，即 ListBuckets）。查询存储桶列表请求发往 service 域名，该域名不带存储桶名前缀，地域由 service 域名决定。

注意事项

- 查询存储桶列表请求发往 service 域名 `service.cosrapid.<Region>.myqcloud.com`，该域名不带存储桶名前缀；`EnableRapidDomain=True` 时由 SDK 自动设置。
- 本接口仅返回 service 域名对应 `<Region>` 下的存储桶；如需查看其他地域的存储桶，请更换 `Region` 后分别调用。
- 当 `IsTruncated` 为 `true` 时，将响应中的 `NextMarker` 作为下次请求的 `Marker` 传入，直到 `IsTruncated` 为 `false`。
- 本接口只能列出签名密钥所属账号的存储桶，属于控制面接口，使用基础凭证签名，不走 session。

请求

方法原型

```
def list_buckets(self, TagKey=None, TagValue=None, Region=None,
                  CreateTime=None,
                  Range=None, Marker="", MaxKeys=2000, **kwargs)
```

参数说明

参数名称	参数描述	类型	是否必选
Marker	分页游标，取值为上一次响应的 <code>NextMarker</code> ；首次请求无需携带	string	否
MaxKeys	单次返回的最大存储桶数量，取值范围1~2000，默认2000	int	否
kwargs	可选请求参数，使用本文列出的参数	dict	否

说明：

高性能桶列举请仅使用 `Marker` 与 `MaxKeys` 。 `TagKey / TagValue` 、 `Region` 、 `CreateTime / Range` 等普通 COS 桶过滤参数不属于高性能桶契约。

响应

返回值为解析后的 dict，对应 XML 根节点 `ListAllMyBucketsResult`：

```
{
  'Owner': {'ID': '...', 'DisplayName': '...'},
  'Marker': '',
  'NextMarker': '',
  'IsTruncated': 'false',
  'Buckets': {
    'Bucket': [
      {
        'Name': 'examplebucket-x--1250000000',
        'Location': 'ap-guangzhou',
        'CreationDate': '2023-10-13T09:13:10Z',
        'VpcId': 'vpc-abcdefgh',
        'SubnetId': 'subnet-abcdefgh',
        'Zone': 'ap-guangzhou-3',
        'Status': 'Available',
      },
      # ...
    ]
  },
}
```

参数名称	参数描述	类型
Owner	存储桶所有者信息，含 ID 、 DisplayName	dict
Marker	本次请求使用的分页游标	string
NextMarker	下一页分页游标，仅当 IsTruncated 为 true 时非空	string

		n g
IsTruncated	是否仍有未列举完的存储桶， <code>true</code> 表示存在下一页	st ri n g
Buckets.Bucket	存储桶列表，每项含 <code>Name</code> 、 <code>Location</code> 、 <code>CreationDate</code> 、 <code>VpcId</code> 、 <code>SubnetId</code> 、 <code>Zone</code> 、 <code>Status</code>	lis t
Bucket.Name	存储桶完整名称，格式为 <code><BucketName>-x--<APPID></code>	st ri n g
Bucket.Location	存储桶所在地域	st ri n g
Bucket.CreationDate	存储桶创建时间，ISO 8601 格式	st ri n g

错误码

本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

查询存储桶列表请求发往 **service 域名**。`EnableRapidDomain=True` 时 SDK 已自动把 `ServiceDomain` 设为 `service.cosrapid.<Region>.myqcloud.com`。

```
# -*- coding=utf-8 -*-
import os
from qcloud_cos import CosConfig, CosS3Client

# 高性能桶经 VPC 内网 HTTP 80 端口接入，SDK 的网关 DNS 负载均衡也仅对 HTTP 请求生效；SDK 默认 https，此处需显式指定 http。
config = CosConfig(
    Scheme='http',
    Region='ap-guangzhou',
```

```
SecretId=os.environ['COS_SECRET_ID'],
SecretKey=os.environ['COS_SECRET_KEY'],
Token=os.environ.get('COS_SECURITY_TOKEN'),
EnableRapidDomain=True)
client = CosS3Client(config)

# 分页列举该地域下的全部高性能桶。
marker = ''
while True:
    res = client.list_buckets(Marker=marker, MaxKeys=2000)
    buckets = (res.get('Buckets') or {}).get('Bucket') or []
    if isinstance(buckets, dict):
        buckets = [buckets]
    for bucket in buckets:
        print('bucket: %s, region: %s, created: %s' % (
            bucket['Name'], bucket.get('Location'),
            bucket.get('CreationDate')))
        if res.get('IsTruncated') != 'true':
            break
    marker = res.get('NextMarker', '')
```

创建访问会话

最近更新时间：2026-10-21 18:34:06

为高性能桶创建访问会话（CreateSession），返回可用于后续请求的临时凭证，是高性能桶专属能力。使用 Python SDK 访问高性能桶的数据面接口时，CosS3Client 会自动调用本接口并按桶缓存复用临时凭证，通常无需手动调用；本文档说明其手动调用方式及返回结构，便于需要自行管理临时凭证的场景使用。

注意事项

- 向其他调用方提供临时访问链接时，参见 [Python SDK 预签名 URL](#)。
- Credentials 中的访问密钥和会话令牌仅用于请求鉴权，不应输出到终端、日志或错误报告。诊断时使用 requestId、过期时间等信息。
- 临时凭证具有桶级作用域，仅对创建会话时指定的高性能桶有效。
- 临时凭证有效期较短，过期后需重新创建会话。使用自动会话管理时，后续请求会在需要时刷新凭据。
- CreateSession 自身使用基础凭证签名，独立超时（CosConfig.CreateSessionTimeout，默认 30 秒）且不跟随重定向；GET / HEAD 预签名默认申请独立的 ReadOnly 会话，不复用数据面已缓存的 ReadWrite。
- 若返回的凭证不完整或过期时间缺失/非法，SDK 会直接返回错误，不会下发无效凭证。
- 该接口仅支持高性能桶；对普通桶调用会本地报错。

请求

调用形式

```
client.create_session(Bucket=bucket, Mode='ReadWrite')
```

参数说明

参数名称	参数描述	类型	是否必选
Bucket	高性能桶完整名称，格式为 <BucketName>-x--<APPID>	string	是
Mode	会话权限，ReadWrite 或 ReadOnly，缺省取 CosConfig.SessionMode	string	否

响应

```
# -*- coding: utf-8 -*-
{
    'Credentials': {
        'AccessKeyId': '...',      # 临时 SecretId
        'SecretAccessKey': '...',   # 临时 SecretKey
        'SessionToken': '...',     # 会话令牌
        'Expiration': '2023-10-13T09:13:10Z',
    },
    'Expiration': '2023-10-13T09:13:10Z',
    'RequestId': '...',
}
```

参数名称	参数描述	类型
Credentials.AccessKeyId	临时凭证 AccessKeyId (临时 SecretId)	string
Credentials.SecretAccessKey	临时凭证 SecretAccessKey (临时 SecretKey)	string
Credentials.SessionToken	会话令牌，后续请求需携带	string
Credentials.Expiration	临时凭证过期时间，ISO 8601 格式	string
Expiration	临时凭证过期时间	string
RequestId	本次请求的唯一标识	string

错误码

本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

```
# -*- coding=utf-8 -*-
import os
from qcloud_cos import CosConfig, CosS3Client

# 高性能桶经 VPC 内网 HTTP 80 端口接入，SDK 的网关 DNS 负载均衡也仅对 HTTP 请求生效；SDK 默认 https，此处需显式指定 http。
config = CosConfig(
    Scheme='http',
    Region='ap-guangzhou',
```

```
SecretId=os.environ['COS_SECRET_ID'],
SecretKey=os.environ['COS_SECRET_KEY'],
Token=os.environ.get('COS_SECURITY_TOKEN'),
EnableRapidDomain=True)
client = CosS3Client(config)

res = client.create_session(Bucket='examplebucket-x--1250000000',
Mode='ReadWrite')
cred = res['Credentials']
print('RequestId: %s' % res.get('RequestId'))
print('Expiration: %s' % cred['Expiration'])
```

创建存储桶

最近更新时间：2026-10-21 18:34:06

在指定地域创建高性能桶。创建时需要提供 VPC ID、子网 ID 和可用区，可使用命名参数或 Metadata 请求头。

注意事项

- 要创建的目标桶名由 Bucket 参数指定，格式为 <BucketName>-x--<APPID>。
- 高性能桶的三个网络参数合并后必须齐全非空，否则 SDK 在发送请求前本地报错；同名头取值冲突同样本地报错。
- 命名参数仅用于高性能桶，普通桶传入非 None 值会在本地报错；普通桶原有 Metadata 行为保持不变，其他接口也不新增这些参数。
- 桶名在同一地域内需全局唯一。重复创建已存在的桶时，按其当前状态返回对应的 HTTP 409 错误：桶可用返回 BucketAlreadyExists，创建中返回 BucketCreating，删除中返回 BucketDeleting。
- 创建桶属于控制面接口，使用基础凭证签名，不走 session。

请求

调用形式

```
client.create_bucket(Bucket=bucket, VpcId=vpc_id,
                     SubnetId=subnet_id, Zone=zone)
```

参数说明

参数名称	参数描述	类型	是否必选
Bucket	待创建的高性能桶完整名称，格式为 <BucketName>-x--<APPID>，例如 examplebucket-x--1250000000	string	是
VpcId	高性能桶绑定的 VPC ID，例如 vpc-xxxxxxxx	string	与 Metadata 合并后必填
SubnetId	高性能桶绑定的子网 ID，例如 subnet-xxxxxxxx	string	与 Metadata 合并后必填
Zone	高性能桶所在可用区，例如 ap-guangzhou-3	string	与 Metadata 合并后必填

kwargs	可选请求参数，使用本文列出的参数；可用 Metadata 传入上述三个网络头	dict	否
--------	--	------	---

VpcId / SubnetId / Zone 对应 x-cos-vpc-id / x-cos-subnet-id / x-cos-zone 请求头：

- 命名参数为 None 表示未提供，允许由 Metadata 补齐；两者会按头名大小写不敏感合并。
- 合并后三个字段的值必须齐全且非空，缺失、空字符串或纯空白值会在发送请求前本地报错。
- 同名头取值冲突时本地报错，不静默覆盖。
- Metadata 的键是完整 HTTP 头名，不会自动添加 x-cos-meta- 前缀。
- 命名参数仅用于高性能桶；普通桶传入非 None 值会在本地报错。

响应

参数名称	参数描述	类型
None	创建请求被受理时无返回内容，对应 HTTP 200	None

返回 None 只表示创建请求已受理，不代表存储桶已可用。请使用 head_bucket 查询 x-cos-bucket-status，待服务端报告 Available 后再进行对象操作。

错误码

本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

```
# -*- coding=utf-8 -*-
import os
from qcloud_cos import CosConfig, CosS3Client

# 高性能桶经 VPC 内网 HTTP 80 端口接入，SDK 的网关 DNS 负载均衡也仅对 HTTP 请求生效；SDK 默认 https，此处需显式指定 http。
config = CosConfig(
    Scheme='http',
    Region='ap-guangzhou',
    SecretId=os.environ['COS_SECRET_ID'],
    SecretKey=os.environ['COS_SECRET_KEY'],
    Token=os.environ.get('COS_SECURITY_TOKEN'),
    EnableRapidDomain=True)
client = CosS3Client(config)

bucket = 'examplebucket-x--1250000000'
```

使用命名参数创建桶。

```
client.create_bucket(  
    Bucket=bucket,  
    VpcId='vpc-xxxxxxx',  
    SubnetId='subnet-xxxxxxx',  
    Zone='ap-guangzhou-3')  
  
# 创建请求受理后，轮询 x-cos-bucket-status，待 Available 后再进行对象操作。  
status = client.head_bucket(Bucket=bucket).get('x-cos-bucket-status')  
print('bucket status: %s' % status)
```

如需使用 `Metadata`，用下面的调用替换上面的创建调用；不要对同一桶连续执行两次创建：

```
client.create_bucket(  
    Bucket=bucket,  
    Metadata={  
        'x-cos-vpc-id': 'vpc-xxxxxxx',  
        'x-cos-subnet-id': 'subnet-xxxxxxx',  
        'x-cos-zone': 'ap-guangzhou-3',  
    })
```

删除存储桶

最近更新时间：2026-10-21 18:34:06

删除一个高性能桶（DELETE Bucket）。删除的目标桶由 `Bucket` 参数指定，请求使用基础凭证签名。

注意事项

- 删除桶前必须先清空桶内所有对象，否则会返回 `BucketNotEmpty` 错误。
- 删除的目标桶由 `Bucket` 参数指定，请确认桶名指向待删除的桶。
- 桶删除后不可恢复，请谨慎操作。
- 删除桶属于控制面接口，使用基础凭证签名，不走 session。

请求

方法原型

```
def delete_bucket(self, Bucket, **kwargs)
```

参数说明

参数名称	参数描述	类型	是否必选
Bucket	待删除的高性能桶完整名称，格式为 <code><BucketName>-x--<APPID></code>	string	是
kwargs	可选请求参数，使用本文列出的参数	dict	否

响应

参数名称	参数描述	类型
None	删除成功无返回内容，对应 HTTP 204	None

错误码

本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

```
# -*- coding=utf-8 -*-
import os
from qcloud_cos import CosConfig, CosS3Client

# 高性能桶经 VPC 内网 HTTP 80 端口接入, SDK 的网关 DNS 负载均衡也仅对 HTTP 请求生效; SDK 默认 https, 此处需显式指定 http。
config = CosConfig(
    Scheme='http',
    Region='ap-guangzhou',
    SecretId=os.environ['COS_SECRET_ID'],
    SecretKey=os.environ['COS_SECRET_KEY'],
    Token=os.environ.get('COS_SECURITY_TOKEN'),
    EnableRapidDomain=True)
client = CosS3Client(config)

client.delete_bucket(Bucket='examplebucket-x--1250000000')
print('bucket deleted')
```

查询存储桶

最近更新时间：2026-10-21 18:34:07

查询一个高性能桶是否存在以及当前调用方是否有权限访问（HEAD Bucket）。目标桶由 `Bucket` 参数指定，结果以 HTTP 状态码及响应头部呈现，无响应体。

注意事项

- HTTP 状态码可用于区分结果：200表示桶存在且有权限，403表示请求未获授权，不能据此判断桶是否存在，404表示桶不存在。
- 返回的响应头字典保留服务端原始头名和值，同时提供小写别名，可按 `x-cos-bucket-status` 等小写键读取。
- 查询桶属于控制面接口，使用基础凭证签名，不走 session。

请求

方法原型

```
def head_bucket(self, Bucket, **kwargs)
```

参数说明

参数名称	参数描述	类型	是否必选
Bucket	待查询的高性能桶完整名称，格式为 <code><BucketName>-x--<APPID></code>	string	是
kwargs	可选请求参数，使用本文列出的参数	dict	否

响应

返回值为响应头部字典：

参数名称	参数描述	类型
响应头 dict	成功时返回响应头部字典，可通过 <code>x-cos-bucket-status</code> 、 <code>x-cos-owner-uin</code> 、 <code>x-cos-vpc-id</code> 、 <code>x-cos-subnet-id</code> 、 <code>x-cos-zone</code> 、 <code>x-cos-creation-date</code> 等键读取桶状态与网络信息	dict

桶存在且有权限返回200；无权限返回403；不存在返回404。

错误码

本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

```
# -*- coding=utf-8 -*-
import os
from qcloud_cos import CosConfig, CosS3Client
from qcloud_cos.cos_exception import CosServiceError

# 高性能桶经 VPC 内网 HTTP 80 端口接入，SDK 的网关 DNS 负载均衡也仅对 HTTP 请求生效；SDK 默认 https，此处需显式指定 http。
config = CosConfig(
    Scheme='http',
    Region='ap-guangzhou',
    SecretId=os.environ['COS_SECRET_ID'],
    SecretKey=os.environ['COS_SECRET_KEY'],
    Token=os.environ.get('COS_SECURITY_TOKEN'),
    EnableRapidDomain=True)
client = CosS3Client(config)

bucket = 'examplebucket-x--1250000000'

# 桶存在且有权限时返回响应头字典；无权限或不存在时抛出 CosServiceError。
try:
    resp = client.head_bucket(Bucket=bucket)
    print('bucket status: %s' % resp.get('x-cos-bucket-status'))
except CosServiceError as e:
    print('HeadBucket failed, status: %s, code: %s' % (
        e.get_status_code(), e.get_error_code()))
```

存储桶策略

设置存储桶策略

最近更新时间：2026-10-21 18:34:05

为高性能桶设置存储桶策略（PUT Bucket Policy），通过声明式的授权语句控制账号对桶及对象的访问权限。该接口属于控制面，使用基础凭证签名，不走 session。

注意事项

- Resource 必须精确匹配当前请求的桶，高性能桶的资源形态为 `qcs::cosrapid:<Region>:uid/<APPID>:<BucketName>-x--<APPID>/*`；指向其他桶、跨地域或通配 `*` 均会被拒绝（InvalidArgument）。
- Principal 的 key 仅支持 `qcs` 与 `service`；`qcs` 形如 `qcs::cam::uin/<主账号uin>:uin/<子账号uin>`。
- Effect 取值为 `allow` 或 `deny`；Action 原样透传 CAM。
- `put_bucket_policy` 会以本次提交的策略整体覆盖桶上原有策略，而非增量合并，需提交完整的目标策略集合。
- 仅桶所属账号下获得授权的身份可调用。未获授权时返回 `403 AccessDenied`；请求已获授权但所属账号与桶所属账号不符时，返回 `405 MethodNotAllowed`。
- 设置存储桶策略属于控制面接口，使用基础凭证签名，不走 session。

请求

方法原型

```
def put_bucket_policy(self, Bucket, Policy, **kwargs)
```

参数说明

参数名称	参数描述	类型	是否必选
Bucket	高性能桶完整名称，格式为 <code><BucketName>-x--<APPID></code>	string	是
Policy	桶策略内容，可为 JSON 字符串或 JSON dict	string / dict	是
kwargs	可选请求参数，使用本文列出的参数	dict	否

策略字段：

字段	参数描述	类型	是否必选
Version	策略语法版本，通常为 2.0	string	否
Statement	授权语句列表	list	是
Statement.Principal	被授权者，key 仅支持 qcs 与 service；qcs 形如 qcs::cam::uin/<主账号uin>:uin/<子账号uin>	dict	是
Statement.Action	授权的操作列表，形如 name/cos:CreateSession、name/cos:GetObject	list	是
Statement.Effect	授权效果，取值 allow 或 deny（大小写不敏感）	string	是
Statement.Resource	授权作用的资源，需精确匹配当前请求的桶	list	是
Statement.Condition	生效条件，原样透传 CAM	dict	否
Statement.Sid	语句标识符	string	否

响应

参数名称	参数描述	类型
None	设置成功无返回内容，对应 HTTP 204	None

错误码

本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

```
# -*- coding=utf-8 -*-
import os
from qcloud_cos import CosConfig, CosS3Client

# 高性能桶经 VPC 内网 HTTP 80 端口接入，SDK 的网关 DNS 负载均衡也仅对 HTTP 请求生效；SDK 默认 https，此处需显式指定 http。
```



```
config = CosConfig(  
    Scheme='http',  
    Region='ap-guangzhou',  
    SecretId=os.environ['COS_SECRET_ID'],  
    SecretKey=os.environ['COS_SECRET_KEY'],  
    Token=os.environ.get('COS_SECURITY_TOKEN'),  
    EnableRapidDomain=True)  
client = CosS3Client(config)  
  
policy = {  
    'version': '2.0',  
    'Statement': [  
        {  
            'Sid': 'allow-get-and-session',  
            'Effect': 'allow',  
            'Principal': {  
                'qcs': ['qcs::cam::uin/1000000000001:uin/1000000000011'],  
            },  
            'Action': [  
                'name/cos:CreateSession',  
                'name/cos:GetObject',  
            ],  
            'Resource': [  
                'qcs::cosrapid:ap-guangzhou:uid/1250000000:examplebucket-x--1250000000/*',  
            ],  
        },  
    ],  
}  
client.put_bucket_policy(Bucket='examplebucket-x--1250000000',  
    Policy=policy)  
print('bucket policy set')
```

获取存储桶策略

最近更新时间：2026-10-21 18:34:06

获取高性能桶当前生效的存储桶策略（GET Bucket Policy）。返回内容为大写 key 的 JSON，`Version` 固定为 `2.0`。该接口属于控制面，使用基础凭证签名，不走 session。

注意事项

- 当桶上未设置任何策略时，接口会返回 404 `NoSuchPolicy`，可通过错误码判断。
- 返回的 `Policy` 为大写 key 的 JSON 字符串，需要按 JSON 解析后使用。
- 获取存储桶策略属于控制面接口，使用基础凭证签名，不走 session。

请求

方法原型

```
def get_bucket_policy(self, Bucket, **kwargs)
```

参数说明

参数名称	参数描述	类型	是否必选
Bucket	高性能桶完整名称，格式为 <code><BucketName>-x--<APPID></code>	string	是
kwargs	可选请求参数，使用本文列出的参数	dict	否

响应

返回值为字典：

参数名称	参数描述	类型
Policy	桶策略内容，JSON 字符串，与设置时的策略字段相同	string

`Policy` 解析后的字段与 [Python SDK 设置存储桶策略](#) 一致，包括 `Version`、`Statement` 等。

错误码

本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

```
# -*- coding=utf-8 -*-
import json
import os
from qcloud_cos import CosConfig, CosS3Client

# 高性能桶经 VPC 内网 HTTP 80 端口接入, SDK 的网关 DNS 负载均衡也仅对 HTTP 请求生效; SDK 默认 https, 此处需显式指定 http。
config = CosConfig(
    Scheme='http',
    Region='ap-guangzhou',
    SecretId=os.environ['COS_SECRET_ID'],
    SecretKey=os.environ['COS_SECRET_KEY'],
    Token=os.environ.get('COS_SECURITY_TOKEN'),
    EnableRapidDomain=True)
client = CosS3Client(config)

resp = client.get_bucket_policy(Bucket='examplebucket-x--1250000000')
policy = json.loads(resp['Policy'])
print('Version: %s' % policy.get('Version'))
for stmt in policy.get('Statement', []):
    print('Sid: %s, Effect: %s, Action: %s, Resource: %s' % (
        stmt.get('Sid'), stmt.get('Effect'), stmt.get('Action'),
        stmt.get('Resource')))
```

删除存储桶策略

最近更新时间：2026-10-21 18:34:07

删除高性能桶当前设置的存储桶策略（DELETE Bucket Policy）。该接口属于控制面，使用基础凭证签名，不走 session。

注意事项

- 删除桶策略后，桶将恢复为仅由 CAM 授权控制访问，请确认相关账号仍具备所需的 CAM 权限，避免误删策略导致访问中断。
- 删除后桶上将不再有任何授权策略，操作不可逆，请谨慎调用。
- 删除存储桶策略属于控制面接口，使用基础凭证签名，不走 session。

请求

方法原型

```
def delete_bucket_policy(self, Bucket, **kwargs)
```

参数说明

参数名称	参数描述	类型	是否必选
Bucket	高性能桶完整名称，格式为 <code><BucketName>-x--<APP ID></code>	string	是
kwargs	可选请求参数，使用本文列出的参数	dict	否

响应

参数名称	参数描述	类型
None	删除成功无返回内容，对应 HTTP 204	None

错误码

本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

```
# -*- coding=utf-8 -*-
```

```
import os
from qcloud_cos import CosConfig, CosS3Client

# 高性能桶经 VPC 内网 HTTP 80 端口接入, SDK 的网关 DNS 负载均衡也仅对 HTTP 请求生效; SDK 默认 https, 此处需显式指定 http。
config = CosConfig(
    Scheme='http',
    Region='ap-guangzhou',
    SecretId=os.environ['COS_SECRET_ID'],
    SecretKey=os.environ['COS_SECRET_KEY'],
    Token=os.environ.get('COS_SECURITY_TOKEN'),
    EnableRapidDomain=True)
client = CosS3Client(config)

client.delete_bucket_policy(Bucket='examplebucket-x--1250000000')
print('bucket policy deleted')
```

对象操作

重命名对象

最近更新时间：2026-10-21 18:34:05

在同一高性能桶内原子重命名对象。该操作原子地将源对象移至目标对象键。对高性能桶发起请求时，SDK 自动完成会话鉴权。

注意事项

- `RenameSource` 与 `Key` 的末尾 `/` 严格区分文件对象与目录对象，两者必须同为文件键（均不以 `/` 结尾）或目录键（均以 `/` 结尾），否则返回 `400 InvalidArgument`。
- 源键与源对象实际类型不匹配时，不做容错，按源对象不存在处理，返回 `404 NoSuchKey`。
- `Key` 与 `RenameSource` 均不能为空或为 `/`，且不能包含独立的 `.` 或 `..` 路径段。
- 文件对象可设置 `ForbidOverwrite='true'` 防止覆盖已有目标。目录对象不覆盖已有同名对象。
- 重命名请求遇到连接异常、超时、响应截断或 5xx 时，结果可能不确定；SDK 不自动重发，调用方应先确认结果，避免重复操作。
- 重命名属于数据面接口，SDK 使用 session 临时凭证签名。

请求

方法原型

```
def rename_object(self, Bucket, Key, RenameSource, ForbidOverwrite=None,
**kwargs)
```

参数说明

参数名称	参数描述	类型	是否必选
Bucket	高性能桶完整名称，格式为 <code><BucketName>-x--<APPID></code>	string	是
Key	目标对象键（Object Key）	string	是
RenameSource	源对象键（Object Key），按原始对象键填写，无需自行 URL 编码	string	是
ForbidOverwrite	文件对象的防覆盖条件，字符串 <code>"true"</code> 表示不覆盖已有目标	string	否

kwargs	可选请求参数，使用本文列出的参数	dict	否
--------	------------------	------	---

响应

返回值为响应头部字典：

参数名称	参数描述	类型
响应头 dict	成功时返回响应头部字典	dict

错误码

本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

```
# -*- coding=utf-8 -*-
import os
from qcloud_cos import CosConfig, CosS3Client

# 高性能桶经 VPC 内网 HTTP 80 端口接入，SDK 的网关 DNS 负载均衡也仅对 HTTP 请求生效；SDK 默认 https，此处需显式指定 http。
config = CosConfig(
    Scheme='http',
    Region='ap-guangzhou',
    SecretId=os.environ['COS_SECRET_ID'],
    SecretKey=os.environ['COS_SECRET_KEY'],
    Token=os.environ.get('COS_SECURITY_TOKEN'),
    EnableRapidDomain=True)
client = CosS3Client(config)

bucket = 'examplebucket-x--1250000000'

# 原子重命名，并开启防覆盖。
client.rename_object(
    Bucket=bucket,
    Key='newname.txt',
    RenameSource='oldname.txt',
    ForbidOverwrite='true')
print('RenameObject OK')
```


上传对象

最近更新时间：2026-10-21 18:34:05

上传文件对象或创建目录对象至高性能桶（PUT Object）。单次上传的对象大小不得超过 5GB，超过请使用 [Python SDK 分块上传](#)。对高性能桶发起请求时，SDK 自动完成会话鉴权。

注意事项

- 本地文件优先使用 [Python SDK 上传本地文件](#)；可读流可使用 [Python SDK 上传可读流](#)。
- 对象键不以 `/` 结尾表示文件对象，以 `/` 结尾表示目录对象；两者严格区分，不会自动转换类型。
- 创建目录对象时，`Body` 必须为空，实际请求的 `Content-Length` 必须为 `0`；否则返回 `400 DirectoryObjectContainsData`。传入 `Body=b''` 时 SDK 会发送 `Content-Length: 0`，无需另行设置 `ContentLength`。
- 目录对象不携带 `Metadata`、`CacheControl`、`ContentDisposition`、`ContentEncoding`、`ContentLanguage`、`ContentType`、`Expires` 等元数据参数，携带会返回 `400 InvalidArgument`。文件对象的 `ContentType` 不使用 `application/x-directory`。
- 文件与同名目录不能共存或相互覆盖。文件上传未开启防覆盖时，同名目录已存在会返回 `409 DirAlreadyExists`；文件上传设置 `ForbidOverwrite='true'` 时，同名文件或目录已存在均返回 `409 FileAlreadyExists`。创建目录时，同名文件已存在返回 `409 FileAlreadyExists`，同名目录已存在返回成功。
- 文件对象的防覆盖条件为 `ForbidOverwrite`；目录创建的行为按上述目录规则处理。
- 单次上传的对象大小不得超过 5GB，超过请使用分块上传（`upload_file` 或 [Python SDK 分块上传](#)）。
- 高性能桶不支持对象版本，传入 `VersionId` 会本地报错；也不支持服务端 `Callback` / `CallbackVar`。
- 文件对象的 `ForbidOverwrite` 仅接受字符串 `"true"` 或 `"false"`；置为 `"true"` 且同名文件已存在时会返回 `409 FileAlreadyExists`。
- 对象键不能包含独立的 `.` 或 `..` 路径段，SDK 会在本地报错。
- 上传属于数据面接口，SDK 使用 session 临时凭证签名。

请求

方法原型

```
def put_object(self, Bucket, Body, Key, EnableMD5=False, **kwargs)
```

参数说明

参数名称	参数描述	类型	是否必选
------	------	----	------

Bucket	高性能桶完整名称，格式为 <code><BucketName>-x--<AP PID></code>	string	是
Body	上传的对象内容，可为文件流、字节流或字符串	file / bytes / string	是
Key	对象键（Object Key），即对象在桶中的完整路径	string	是
EnableMD5	是否需要 SDK 计算 <code>Content-MD5</code> ，打开会增加上传耗时	bool	否
kwargs	可选请求参数，使用本文列出的参数	dict	否

下列内容类型、缓存与自定义元数据参数用于文件对象：

参数名称	参数描述	类型	是否必选
ContentType	内容类型，设置 <code>Content-Type</code>	string	否
ContentLength	设置传输长度	int	否
ContentMD5	请求体内容的 MD5 校验值（Base64 编码），用于完整性校验	string	否
CacheControl	缓存策略，设置 <code>Cache-Control</code>	string	否
ContentDisposition	文件名展示方式，设置 <code>Content-Disposition</code>	string	否
ContentEncoding	编码格式，设置 <code>Content-Encoding</code>	string	否
ContentLanguage	内容语言，设置 <code>Content-Language</code>	string	否
Expires	设置 <code>Expires</code>	string	否
Metadata	用户自定义的对象元信息，键须以 <code>x-cos-meta-</code> 开头	dict	否
StorageClasses	存储类型，高性能桶仅支持默认值（ <code>RAPID</code> ）	string	否

ForbidOverwrite	文件对象的防覆盖条件。字符串 <code>"true"</code> 表示不覆盖已存在的同名文件	string	否
-----------------	--	--------	---

响应

返回值为响应头部字典：

参数名称	参数描述	类型
响应头 dict	成功时返回响应头部字典，可通过 <code>ETag</code> 键读取上传对象的 ETag	dict
ETag	上传对象的 MD5 值，从响应头读取	string

错误码

本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

```
# -*- coding=utf-8 -*-
import os
from qcloud_cos import CosConfig, CosS3Client

# 高性能桶经 VPC 内网 HTTP 80 端口接入，SDK 的网关 DNS 负载均衡也仅对 HTTP 请求生效；SDK 默认 https，此处需显式指定 http。
config = CosConfig(
    Scheme='http',
    Region='ap-guangzhou',
    SecretId=os.environ['COS_SECRET_ID'],
    SecretKey=os.environ['COS_SECRET_KEY'],
    Token=os.environ.get('COS_SECURITY_TOKEN'),
    EnableRapidDomain=True)
client = CosS3Client(config)

bucket = 'examplebucket-x--1250000000'

# 上传字节流。
resp = client.put_object(
    Bucket=bucket,
    Key='exampleobject.txt',
    Body=b'hello rapid bucket',
```

```
        ContentType='text/plain')
print('ETag: %s' % resp['ETag'])

# 上传本地文件，并开启防覆盖。
with open('local_file.txt', 'rb') as fp:
    resp = client.put_object(
        Bucket=bucket,
        Key='uploaded-local-file.txt',
        Body=fp,
        ForbidOverwrite='true')

# 创建目录对象：空内容，不传内容类型或自定义元数据。
client.put_object(Bucket=bucket, Key='photos/', Body=b'')
```

下载对象

最近更新时间：2026-10-21 18:34:05

下载高性能桶中的一个对象（GET Object），支持通过 `Range` 下载指定字节范围。对高性能桶发起请求时，SDK 自动完成会话鉴权。

注意事项

- 下载到本地及断点续传见 [Python SDK 下载到本地文件](#)；临时授权下载见 [Python SDK 预签名 URL](#)。
- 对象键末尾的 `/` 严格区分文件对象与目录对象，不做容错。类型不匹配按对象不存在处理，返回 `404 NoSuchKey`。
- 下载目录对象时内容为空，`Content-Length` 为 `0`；目录对象不支持 `Range`，携带时返回 `400 InvalidArgument`。
- 对象键不能包含独立的 `.` 或 `..` 路径段。
- 大文件下载可使用高级接口 `download_file`。多次请求（例如并发分块或续传下载）期间，请保持源文件对象不变。
- 高性能桶不支持对象版本，传入 `VersionId` 会本地报错。
- 下载属于数据面接口，SDK 使用 `session` 临时凭证签名。

请求

方法原型

```
def get_object(self, Bucket, Key, KeySimplifyCheck=True, **kwargs)
```

参数说明

参数名称	参数描述	类型	是否必选
Bucket	高性能桶完整名称，格式为 <code><BucketName>-x--<APPID></code>	string	是
Key	对象键（Object Key），即对象在桶中的完整路径	string	是
KeySimplifyCheck	对象键安全检查开关，请保持默认值 <code>True</code>	bool	否
kwargs	可选请求参数，使用本文列出的参数	dict	否

常用 `kwargs`：

参数名称	参数描述	类型	是否必选
Range	下载的字节范围，格式为 <code>bytes=first-last</code>	string	否
ResponseContentType	设置响应中的 <code>Content-Type</code>	string	否
ResponseContentDisposition	设置响应中的 <code>Content-Disposition</code>	string	否

响应

返回值为字典，对象内容位于 `Body`（`StreamBody`）字段：

参数名称	参数描述	类型
Body	对象内容， <code>StreamBody</code> 类型，可获取文件流或直接下载到本地	StreamBody
响应头 dict	同时包含响应头部字段，如 <code>ETag</code> 、 <code>Content-Length</code> 、 <code>Content-Range</code> 、 <code>Last-Modified</code> 、 <code>Content-Type</code> 等	dict

`Body` 常用方法：

方法	描述
<code>get_raw_stream()</code>	获取原始流对象，可自行 <code>read()</code>
<code>get_stream_to_file(local_file)</code>	把对象内容直接写入本地文件

错误码

本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

```
# -*- coding=utf-8 -*-
import os
from qcloud_cos import CosConfig, CosS3Client
```

高性能桶经 VPC 内网 HTTP 80 端口接入，SDK 的网关 DNS 负载均衡也仅对 HTTP 请求生效；SDK 默认 https，此处需显式指定 http。

```
config = CosConfig(  
    Scheme='http',  
    Region='ap-guangzhou',  
    SecretId=os.environ['COS_SECRET_ID'],  
    SecretKey=os.environ['COS_SECRET_KEY'],  
    Token=os.environ.get('COS_SECURITY_TOKEN'),  
    EnableRapidDomain=True)  
client = CosS3Client(config)
```

```
bucket = 'examplebucket-x--1250000000'  
key = 'exampleobject.txt'
```

读取对象内容。

```
resp = client.get_object(Bucket=bucket, Key=key)  
body = resp['Body'].get_raw_stream().read()  
print('object content: %s' % body)
```

下载到本地文件。

```
resp = client.get_object(Bucket=bucket, Key=key)  
resp['Body'].get_stream_to_file('local_file.txt')
```

只下载前 1KB。

```
resp = client.get_object(Bucket=bucket, Key=key, Range='bytes=0-1023')  
print('Content-Range: %s' % resp.get('Content-Range'))
```

查询对象元数据

最近更新时间：2026-10-21 18:34:05

查询高性能桶中对象的元数据，不返回对象内容（HEAD Object）。元数据通过响应头部返回。对高性能桶发起请求时，SDK 自动完成会话鉴权。

注意事项

- 对象键末尾的 `/` 严格区分文件对象与目录对象，不做容错；类型不匹配按对象不存在处理（`NoSuchKey` 语义），HTTP 状态码为 `404`。HEAD 响应无错误正文时，SDK 的 `get_error_code()` 为 `NoSuchResource`，请按 HTTP 状态码判断。
- HEAD 请求不返回对象内容，所有元数据均从响应头中读取。
- 若仅需判断对象是否存在，可捕获 `CosServiceError` 并判断 `get_status_code() == 404`。
- 对象键不能包含独立的 `.` 或 `..` 路径段。
- 查询元数据属于数据面接口，SDK 使用 session 临时凭证签名。

请求

方法原型

```
def head_object(self, Bucket, Key, **kwargs)
```

参数说明

参数名称	参数描述	类型	是否必选
Bucket	高性能桶完整名称，格式为 <code><BucketName>-x--<APPID></code>	string	是
Key	对象键（Object Key），即对象在桶中的完整路径	string	是
kwargs	可选请求参数，使用本文列出的参数	dict	否

响应

返回值为响应头部字典，常见头部：

头部	参数描述	类型
Content-Length	对象内容长度（字节）	string

Content-Type	对象的 MIME 类型	string
ETag	对象的 ETag	string
Last-Modified	对象的最后修改时间	string
x-cos-meta-*	用户自定义元数据	string

返回的响应头字典保留服务端原始头名和值，同时提供小写别名，以及 `ETag`、`Content-Length`、`Last-Modified`、`Content-Type` 等固定键，可按 `resp['ETag']` 或 `resp['etag']` 读取。

错误码

本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

```
# -*- coding=utf-8 -*-
import os
from qcloud_cos import CosConfig, CosS3Client

# 高性能桶经 VPC 内网 HTTP 80 端口接入，SDK 的网关 DNS 负载均衡也仅对 HTTP 请求生效；SDK 默认 https，此处需显式指定 http。
config = CosConfig(
    Scheme='http',
    Region='ap-guangzhou',
    SecretId=os.environ['COS_SECRET_ID'],
    SecretKey=os.environ['COS_SECRET_KEY'],
    Token=os.environ.get('COS_SECURITY_TOKEN'),
    EnableRapidDomain=True)
client = CosS3Client(config)

resp = client.head_object(
    Bucket='examplebucket-x--1250000000', Key='exampleobject.txt')
print('Content-Length: %s' % resp.get('Content-Length'))
print('ETag: %s' % resp.get('ETag'))
```

删除对象

最近更新时间：2026-10-21 18:34:06

删除高性能桶中的单个对象（DELETE Object）。对高性能桶发起请求时，SDK 自动完成会话鉴权。

注意事项

- 对象键末尾的 `/` 严格区分文件对象与目录对象，不做容错。类型不匹配按对象不存在处理，返回 `204 No Content`，不会删除另一种类型的同名对象。
- 删除目录对象前先清空目录；非空目录返回 `409 DirNotEmpty`。
- 删除不存在的对象时，高性能桶按幂等语义处理，通常仍返回成功。
- 高性能桶不支持对象版本，传入 `VersionId` 会本地报错。
- 需要一次删除多个对象时，请使用 [Python SDK 批量删除对象](#)。
- 删除属于数据面接口，SDK 使用 session 临时凭证签名。

请求

方法原型

```
def delete_object(self, Bucket, Key, **kwargs)
```

参数说明

参数名称	参数描述	类型	是否必选
Bucket	高性能桶完整名称，格式为 <code><BucketName>-x--<APPID></code>	string	是
Key	对象键（Object Key），即对象在桶中的完整路径	string	是
kwargs	可选请求参数，使用本文列出的参数	dict	否

响应

返回值为响应头部字典：

参数名称	参数描述	类型
响应头 dict	成功时返回响应头部字典	dict

错误码

本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

```
# -*- coding=utf-8 -*-
import os
from qcloud_cos import CosConfig, CosS3Client

# 高性能桶经 VPC 内网 HTTP 80 端口接入，SDK 的网关 DNS 负载均衡也仅对 HTTP 请求生效；SDK 默认 https，此处需显式指定 http。
config = CosConfig(
    Scheme='http',
    Region='ap-guangzhou',
    SecretId=os.environ['COS_SECRET_ID'],
    SecretKey=os.environ['COS_SECRET_KEY'],
    Token=os.environ.get('COS_SECURITY_TOKEN'),
    EnableRapidDomain=True)
client = CosS3Client(config)

client.delete_object(Bucket='examplebucket-x--1250000000',
Key='exampleobject.txt')
print('delete object OK')
```

批量删除对象

最近更新时间：2026-10-21 18:34:06

在一次请求中批量删除高性能桶中的多个对象（DELETE Multiple Objects），单次最多支持 1000 个对象。对高性能桶发起请求时，SDK 自动完成会话鉴权。

注意事项

- 每个 `Key` 末尾的 `/` 严格区分文件对象与目录对象。类型不匹配按对象不存在处理，计为该项删除成功，不会删除另一种类型的同名对象。
- 目录须先清空；非空目录的错误在该项 `Error` 中返回，请逐项检查结果。
- 单次请求删除对象数量为 1~1000，超出请分批调用。
- 即使整体请求返回成功，仍需检查 `Error` 字段，其中记录了删除失败的对象及原因。
- 将 `Quiet` 置为 `"true"` 可减少返回内容，仅返回删除失败的对象信息。
- 高性能桶不支持对象版本，`Object` 中传入 `VersionId` 会本地报错。
- 批量删除属于数据面接口，SDK 使用 session 临时凭证签名。

请求

方法原型

```
def delete_objects(self, Bucket, Delete={}, **kwargs)
```

参数说明

参数名称	参数描述	类型	是否必选
Bucket	高性能桶完整名称，格式为 <code><BucketName>-x--<APPID></code>	string	是
Delete	批量删除信息，含 <code>Quiet</code> 与 <code>Object</code> 字段	dict	是
kwargs	可选请求参数，使用本文列出的参数	dict	否

`Delete` 字段：

参数名称	参数描述	类型	是否必选
Quiet	结果模式。 <code>"false"</code> （Verbose，默认）返回每个对象的删除结果； <code>"t</code>	string	否

	<code>true</code> "（Quiet）仅返回删除失败的对象		
Object	待删除的对象列表，每项为含 <code>Key</code> 字段的 dict，数量为 1~1000	list	是

响应

返回值为字典：

参数名称	参数描述	类型
Deleted	删除成功的对象列表	list
Error	删除失败的对象列表，每项含 <code>Key</code> 、 <code>Code</code> 、 <code>Message</code>	list

错误码

本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

```
# -*- coding=utf-8 -*-
import os
from qcloud_cos import CosConfig, CosS3Client

# 高性能桶经 VPC 内网 HTTP 80 端口接入，SDK 的网关 DNS 负载均衡也仅对 HTTP 请求生效；SDK 默认 https，此处需显式指定 http。
config = CosConfig(
    Scheme='http',
    Region='ap-guangzhou',
    SecretId=os.environ['COS_SECRET_ID'],
    SecretKey=os.environ['COS_SECRET_KEY'],
    Token=os.environ.get('COS_SECURITY_TOKEN'),
    EnableRapidDomain=True)
client = CosS3Client(config)

objects = {
    'Quiet': 'false',
    'Object': [
        {'Key': 'a.txt'},
        {'Key': 'b.txt'},
    ]
}
```

```
}  
resp = client.delete_objects(  
    Bucket='examplebucket-x--1250000000', Delete=objects)  
print('deleted: %s, errors: %s' % (resp.get('Deleted'),  
    resp.get('Error')))
```

复制对象

最近更新时间：2026-10-21 18:34:06

在同一高性能桶内复制对象（PUT Object – Copy），常用于对象备份或修改对象元数据。对高性能桶发起请求时，SDK 自动完成会话鉴权。

注意事项

- 需要自动选择直接复制或分块复制时，使用 [Python SDK 高级复制对象](#)，并留意两种路径的元数据差异。
- 源与目标均为文件对象，对象键末尾的 `/` 严格区分类型，不做容错；源键以 `/` 结尾返回 `400 InvalidCopySource`，目标键以 `/` 结尾返回 `400 InvalidArgument`。
- 不带 `/` 的源键若实际对应同名目录，按源对象不存在处理，返回 `404 NoSuchKey`。
- 高性能桶仅支持同桶复制，源对象与目标对象必须位于同一个高性能桶，SDK 会在请求前本地校验。
- 高性能桶不支持对象版本，`CopySource` 中传入 `VersionId` 会本地报错。
- 超过 5GB 的对象请使用分块复制（详见 [Python SDK 复制分块](#)）。
- 复制属于数据面接口，SDK 使用 session 临时凭证签名。

请求

方法原型

```
def copy_object(self, Bucket, Key, CopySource, CopyStatus='Copy',
**kwargs)
```

参数说明

参数名称	参数描述	类型	是否必选
Bucket	目标高性能桶完整名称，格式为 <code><BucketName>-x--<APPID></code>	string	是
Key	目标对象键（Object Key）	string	是
CopySource	复制源信息，dict 类型，含 <code>Bucket</code> 、 <code>Key</code> 、 <code>Region</code> 等字段	dict	是

CopyStatus	元数据复制方式，可选 Copy（默认，复制源元数据）或 Replaced（用本次请求头替换元数据）	string	否
kwargs	可选请求参数，使用本文列出的参数	dict	否

CopySource 字段：

参数名称	参数描述	类型	是否必选
Bucket	源对象所在的高性能桶完整名称，须与目标桶相同	string	是
Key	源对象键	string	是
Region	源对象所在地域	string	是

响应

返回值为字典：

参数名称	参数描述	类型
ETag	复制生成对象的 ETag	string
LastModified	对象的最后修改时间	string
响应头 dict	同时包含响应头部字段	dict

错误码

本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

```
# -*- coding=utf-8 -*-
import os
from qcloud_cos import CosConfig, CosS3Client

# 高性能桶经 VPC 内网 HTTP 80 端口接入，SDK 的网关 DNS 负载均衡也仅对 HTTP 请求生效；SDK 默认 https，此处需显式指定 http。
config = CosConfig(
    Scheme='http',
```



```
Region='ap-guangzhou',
SecretId=os.environ['COS_SECRET_ID'],
SecretKey=os.environ['COS_SECRET_KEY'],
Token=os.environ.get('COS_SECURITY_TOKEN'),
EnableRapidDomain=True)
client = CosS3Client(config)

bucket = 'examplebucket-x--1250000000'

# 同桶复制，高性能桶要求源与目标在同一个桶。
resp = client.copy_object(
    Bucket=bucket,
    Key='dst.txt',
    CopySource={'Bucket': bucket, 'Key': 'src.txt', 'Region': 'ap-
guangzhou'})
print('ETag: %s' % resp['ETag'])
```

列举对象

最近更新时间：2026-10-21 18:34:07

列举高性能桶中的对象（GET Bucket），支持指定目录、列出当前层级内容并分页翻页。对高性能桶发起请求时，SDK 自动完成会话鉴权。高性能桶使用目录语义列举。

注意事项

- 高性能桶使用目录语义列举，`Delimiter` 为空时 SDK 仅对高性能桶规范化为 `/`，普通桶仍保留原来的空 `delimiter` 行为。
- 非空 `Prefix` 必须以 `/` 结尾，不支持按任意字符串前缀递归列举。
- 分页时保持 `Prefix` 不变，把返回的 `NextMarker` 直接作为下一次的 `Marker`。高性能桶的游标是当前目录内的相对名字，不要从 `Contents.Key` 推导或额外拼接 `Prefix`。
- 服务端对名称与游标使用 query 编码：空格编码为 `+`，字面加号编码为 `%2B`。SDK 默认按该规则解码后返回（`+` 还原为空格，`%2B` 还原为字面加号，百分号只解码一次）；显式传入 `EncodingType='url'` 时保留编码，调用方应使用 `urllib.parse.unquote_plus` 解码名称及 Key/Marker 后再使用，不要使用 `unquote` 等通用 URL 解码，否则 `+` 不会还原为空格。
- 列举对象属于数据接口，SDK 使用 session 临时凭证签名。

请求

方法原型

```
def list_objects(self, Bucket, Prefix="", Delimiter="", Marker="",
MaxKeys=1000,
EncodingType="", **kwargs)
```

参数说明

参数名称	参数描述	类型	是否必选
Bucket	高性能桶完整名称，格式为 <code><BucketName>-x--<APPID></code>	string	是
Prefix	待列举的目录前缀；非空值须以 <code>/</code> 结尾，仅列举该目录当前层级	string	否
Delimiter	分隔符，高性能桶按目录语义列举；为空时 SDK 自动规范化为 <code>/</code>	string	否

Marker	当前目录内的相对名称游标；使用上一页的 <code>NextMarker</code> ，不拼接 <code>Prefix</code>	string	否
MaxKeys	单次返回的最大对象数，取值范围 1~1000，默认 1000	int	否
Encoding Type	返回结果的编码方式，可选值： <code>url</code>	string	否
kwargs	可选请求参数，使用本文列出的参数	dict	否

响应

下面是一页最多返回2项目且需继续翻页的返回示例；文件与子目录合计2项：

```
{
  'Prefix': 'photos/',
  'NextMarker': 'sub',
  'MaxKeys': '2',
  'IsTruncated': 'true',
  'Contents': [
    { 'Key': 'photos/a.jpg', 'ETag': '...', 'Size': '1024',
      'LastModified': '2023-10-13T09:13:10Z' },
    # ...
  ],
  'CommonPrefixes': [
    { 'Prefix': 'photos/sub/' },
    # ...
  ],
}
```

参数名称	参数描述	类型
Prefix	本次列举使用的前缀	string
Marker	本次列举的起始标记	string
NextMarker	当前目录内的相对名称游标； <code>IsTruncated</code> 为 true 时原样用于下一次 <code>Marker</code>	string

MaxKeys	本次列举的最大对象数	string
IsTruncated	是否需要继续翻页；下一页可能为空，直到返回 <code>false</code> 为止	string
Contents	对象列表，每项含 <code>Key</code> 、 <code>ETag</code> 、 <code>Size</code> 、 <code>LastModified</code> 等	list
CommonPrefixes	使用分隔符时聚合出的公共前缀（相当于子目录）	list

错误码

本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

```
# -*- coding=utf-8 -*-
import os
from qcloud_cos import CosConfig, CosS3Client

# 高性能桶经 VPC 内网 HTTP 80 端口接入，SDK 的网关 DNS 负载均衡也仅对 HTTP 请求生效；SDK 默认 https，此处需显式指定 http。
config = CosConfig(
    Scheme='http',
    Region='ap-guangzhou',
    SecretId=os.environ['COS_SECRET_ID'],
    SecretKey=os.environ['COS_SECRET_KEY'],
    Token=os.environ.get('COS_SECURITY_TOKEN'),
    EnableRapidDomain=True)
client = CosS3Client(config)

bucket = 'examplebucket-x--1250000000'
marker = ''
while True:
    # 非空 Prefix 必须以 / 结尾；分页时保持 Prefix 不变。
    resp = client.list_objects(
        Bucket=bucket, Prefix='photos/', Marker=marker, MaxKeys=100)
    contents = resp.get('Contents') or []
    if isinstance(contents, dict):
        contents = [contents]
```

```
for obj in contents:
    print('Key: %s, Size: %s' % (obj['Key'], obj.get('Size')))
if resp.get('IsTruncated') != 'true':
    break
marker = resp.get('NextMarker', '')
print('ListObjects OK')
```

分块上传操作

初始化分块上传

最近更新时间：2026-10-21 18:34:05

初始化一次分块上传（Initiate Multipart Upload），成功后返回 `UploadId`，作为后续上传分块与完成上传的唯一标识。分块上传的完整流程为：初始化取得 `UploadId` → 多次上传分块并逐块记录 `PartNumber` 与 `ETag` → 完成分块上传提交。对高性能桶发起请求时，SDK 自动完成会话鉴权。

注意事项

- 自动分块上传与续传的完整用法见 [Python SDK 上传本地文件](#)。
- 分块上传用于文件对象，`Key` 不以 `/` 结尾；以 `/` 结尾会返回 `400 InvalidArgument`。目录对象请用 [Python SDK 上传对象](#) 创建。
- 如需防止覆盖已有文件对象，必须在 [Python SDK 完成分块上传](#) 调用中显式设置 `ForbidOverwrite='true'`；不能依赖初始化请求中的设置保护最终写入。
- 返回的 `UploadId` 需妥善保存，后续 [Python SDK 上传分块](#)、[Python SDK 完成分块上传](#)、[Python SDK 终止分块上传](#) 均需携带。
- 已上传但未完成提交的分块会占用存储空间，请及时 [Python SDK 完成分块上传](#) 或 [Python SDK 终止分块上传](#)。
- 高性能桶不支持对象版本，传入 `VersionId` 会本地报错。
- 大文件上传建议直接使用高级接口 `upload_file`，其内部会自动完成初始化、并发上传分块、完成分块上传，并支持断点续传与 `EnableMD5` 完整性校验。
- 初始化分块上传属于数据面接口，SDK 使用 session 临时凭证签名。

请求

方法原型

```
def create_multipart_upload(self, Bucket, Key, **kwargs)
```

参数说明

参数名称	参数描述	类型	是否必选
Bucket	高性能桶完整名称，格式为 <code><BucketName>-x--<APPID></code>	string	是
Key	对象键（Object Key），即对象在桶中的完整路径	string	是

		g	
kwargs	可选请求参数，使用本文列出的参数	dict	否

常用 kwargs：

参数名称	参数描述	类型	是否必选
ContentType	对象的 MIME 类型	string	否
CacheControl	缓存策略，设置 Cache-Control	string	否
Metadata	用户自定义的对象元信息，键须以 x-cos-meta- 开头	dict	否
StorageClasses	存储类型，高性能桶仅支持默认值（RAPID）	string	否

响应

返回值为解析后的 dict：

参数名称	参数描述	类型
Bucket	分块上传所在的桶名	string
Key	对象键	string
UploadId	本次分块上传的标识，后续上传分块、完成或终止上传均需携带	string

错误码

本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

```
# -*- coding=utf-8 -*-
import os
from qcloud_cos import CosConfig, CosS3Client

# 高性能桶经 VPC 内网 HTTP 80 端口接入，SDK 的网关 DNS 负载均衡也仅对 HTTP 请求生效；SDK 默认 https，此处需显式指定 http。
config = CosConfig(
    Scheme='http',
```

```
Region='ap-guangzhou',
SecretId=os.environ['COS_SECRET_ID'],
SecretKey=os.environ['COS_SECRET_KEY'],
Token=os.environ.get('COS_SECURITY_TOKEN'),
EnableRapidDomain=True)
client = CosS3Client(config)

resp = client.create_multipart_upload(
    Bucket='examplebucket-x--1250000000',
    Key='exampleobject.txt',
    ContentType='text/plain')
upload_id = resp['UploadId']
print('UploadId: %s' % upload_id)
```


上传分块

最近更新时间：2026-10-21 18:34:05

在初始化分块上传后上传单个分块（Upload Part）。每个分块由 `PartNumber` 标识（从 1 开始，支持乱序上传），上传成功后需从响应头读取该块的 `ETag`，并将 `PartNumber` 与 `ETag` 记录下来，供后续 [Python SDK 完成分块上传](#) 使用。对高性能桶发起请求时，SDK 自动完成会话鉴权。

注意事项

- `Key` 与 `UploadId` 必须对应同一次初始化分块上传；`Key` 表示不以 `/` 结尾的文件对象，不能改成目录对象键。
- 上传成功后必须读取响应头中的 `ETag`，并连同 `PartNumber` 一起记录，[Python SDK 完成分块上传](#) 时需按此校验。
- 除最后一块外，每个分块建议不小于 1MB；分块过小在完成上传时可能返回 `EntityTooSmall`。
- 单个分块大小不得超过 5GB。
- 相同 `UploadId` 与 `PartNumber` 重复上传时，后上传的分块会覆盖先前的分块。
- 上传分块属于数据接口，SDK 使用 session 临时凭证签名。

请求

方法原型

```
def upload_part(self, Bucket, Key, Body, PartNumber, UploadId,
EnableMD5=False, **kwargs)
```

参数说明

参数名称	参数描述	类型	是否必选
Bucket	高性能桶完整名称，格式为 <code><BucketName>-x--<APPID></code>	string	是
Key	对象键（Object Key）	string	是
Body	待上传分块的内容，可为文件流或字节流	file / bytes	是
PartNumber	分块编号，从 1 开始，支持乱序上传	int	是
UploadId	初始化分块上传返回的 <code>UploadId</code>	string	是

EnableMD5	是否需要 SDK 计算 <code>Content-MD5</code> ，打开会增加上传耗时	bool	否
kwargs	可选请求参数，使用本文列出的参数	dict	否

响应

返回值为响应头部字典：

参数名称	参数描述	类型
响应头 dict	成功时返回响应头部字典，可通过 <code>ETag</code> 键读取该分块的 ETag	dict

错误码

本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

```
# -*- coding=utf-8 -*-
import os
from qcloud_cos import CosConfig, CosS3Client

# 高性能桶经 VPC 内网 HTTP 80 端口接入，SDK 的网关 DNS 负载均衡也仅对 HTTP 请求生效；SDK 默认 https，此处需显式指定 http。
config = CosConfig(
    Scheme='http',
    Region='ap-guangzhou',
    SecretId=os.environ['COS_SECRET_ID'],
    SecretKey=os.environ['COS_SECRET_KEY'],
    Token=os.environ.get('COS_SECURITY_TOKEN'),
    EnableRapidDomain=True)
client = CosS3Client(config)

bucket = 'examplebucket-x--1250000000'
key = 'exampleobject.txt'

# 初始化分块上传，取得 UploadId。
init_resp = client.create_multipart_upload(Bucket=bucket, Key=key)
upload_id = init_resp['UploadId']
```

```
# 上传第1个分块，并记录 PartNumber 与 ETag。
part_number = 1
resp = client.upload_part(
    Bucket=bucket,
    Key=key,
    Body=b'this is the content of part 1',
    PartNumber=part_number,
    UploadId=upload_id)
print('part %d uploaded, ETag: %s' % (part_number, resp['ETag']))
```

复制分块

最近更新时间：2026-10-21 18:34:05

将已有对象（或其指定字节范围）作为一个分块复制到当前分块上传中（Upload Part – Copy），常用于大对象的分块复制、拼接场景。与 [Python SDK 上传分块](#) 类似，每个分块由 `PartNumber` 标识，复制成功后需记录返回的 `ETag`，供 [Python SDK 完成分块上传](#) 使用。对高性能桶发起请求时，SDK 自动完成会话鉴权。

注意事项

- 复制源必须为文件对象，源键末尾的 `/` 不做容错；以 `/` 结尾的源键返回 `400 InvalidCopySource`。不带 `/` 的源键若实际对应同名目录，返回 `404 NoSuchKey`。
- `Key` 与 `UploadId` 必须对应同一次初始化分块上传；`Key` 表示不以 `/` 结尾的文件对象，不能改成目录对象键。
- 高性能桶仅支持同桶复制，复制源与目标必须位于同一个高性能桶，SDK 会在请求前本地校验。
- 高性能桶不支持对象版本，`CopySource` 中传入 `VersionId` 会本地报错。
- 通过 `CopySourceRange` 可只复制源对象的指定字节范围，实现大对象的分段复制拼接。
- 复制成功返回的 `ETag` 需连同 `PartNumber` 一起记录，[Python SDK 完成分块上传](#) 时需按此校验。
- 相同 `UploadId` 与 `PartNumber` 重复复制时，后复制的分块会覆盖先前的分块。
- 复制分块属于数据面接口，SDK 使用 session 临时凭证签名。

请求

方法原型

```
def upload_part_copy(self, Bucket, Key, PartNumber, UploadId,
                    CopySource,
                    CopySourceRange='', **kwargs)
```

参数说明

参数名称	参数描述	类型	是否必选
Bucket	目标高性能桶完整名称，格式为 <code><BucketName>-x--<APPID></code>	string	是
Key	目标对象键（Object Key）	string	是

		n g	
PartNumber	分块编号，从1开始	int	是
UploadId	初始化分块上传返回的 <code>UploadId</code>	string	是
CopySource	复制源信息， <code>dict</code> 类型，含 <code>Bucket</code> 、 <code>Key</code> 、 <code>Region</code> 等字段；须与目标为同一个桶	dict	是
CopySourceRange	复制源的字节范围，格式为 <code>bytes=<start>--<end></code> ，如 <code>bytes=0-1048575</code>	string	否
kwargs	可选请求参数，使用本文列出的参数	dict	否

响应

返回值为字典：

参数名称	参数描述	类型
ETag	复制生成分块的 ETag，需记录供完成分块上传使用	string
LastModified	分块的最后修改时间	string
响应头 dict	同时包含响应头部字段	dict

错误码

本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

```
# -*- coding=utf-8 -*-
import os
from qcloud_cos import CosConfig, CosS3Client
```

高性能桶经 VPC 内网 HTTP 80 端口接入, SDK 的网关 DNS 负载均衡也仅对 HTTP 请求生效; SDK 默认 https, 此处需显式指定 http。

```
config = CosConfig(  
    Scheme='http',  
    Region='ap-guangzhou',  
    SecretId=os.environ['COS_SECRET_ID'],  
    SecretKey=os.environ['COS_SECRET_KEY'],  
    Token=os.environ.get('COS_SECURITY_TOKEN'),  
    EnableRapidDomain=True)  
client = CosS3Client(config)  
  
bucket = 'examplebucket-x--1250000000'  
key = 'exampleobject.txt'  
  
# 初始化分块上传, 取得 UploadId。  
init_resp = client.create_multipart_upload(Bucket=bucket, Key=key)  
upload_id = init_resp['UploadId']  
  
# 复制源对象的前 1MB 作为第1个分块。  
resp = client.upload_part_copy(  
    Bucket=bucket,  
    Key=key,  
    PartNumber=1,  
    UploadId=upload_id,  
    CopySource={'Bucket': bucket, 'Key': 'srckey', 'Region': 'ap-guangzhou'},  
    CopySourceRange='bytes=0-1048575')  
print('part 1 copied, ETag: %s' % resp['ETag'])
```

完成分块上传

最近更新时间：2026-10-21 18:34:06

完成整个分块上传（Complete Multipart Upload）。当所有分块上传完毕后，需按分块编号升序提交每个分块的 `PartNumber` 与 `ETag`，服务端据此将分块合并为完整对象。对高性能桶发起请求时，SDK 自动完成会话鉴权。

注意事项

- `Key` 必须与初始化时一致，表示不以 `/` 结尾的文件对象。
- 返回的 `ETag` 用于标识合并后的对象，不应将其当作完整对象内容的 MD5。
- `PartNumber` 取值为1~10000；提交列表必须严格递增且不能重复，否则返回 `InvalidPartOrder`。编号无需连续，例如已上传且 `ETag`、大小均有效的分块1、3可以按此顺序提交。
- 除最后一块外，分块小于 1MB 会返回 `EntityTooSmall`；`UploadId` 不存在会返回 `NoSuchUpload`。
- 防覆盖条件以本次完成请求为准。需要保留同名文件时，必须在本次调用中显式设置 `ForbidOverwrite='true'`，已有同名文件会返回 `409 FileAlreadyExists`；省略或设置 `'false'` 允许覆盖文件。初始化阶段的设置不能替代本次参数。
- 完成分块上传属于数据面接口，SDK 使用 session 临时凭证签名。

请求

方法原型

```
def complete_multipart_upload(self, Bucket, Key, UploadId,
    MultipartUpload={}, **kwargs)
```

参数说明

参数名称	参数描述	类型	是否必选
Bucket	高性能桶完整名称，格式为 <code><BucketName>-x--<APPID></code>	string	是
Key	对象键（Object Key）	string	是

UploadId	初始化分块上传返回的 UploadId	string	是
Multipart Upload	分块信息，dict 类型，含 Part 列表	dict	是
ForbidOverwrite	本次完成请求的文件防覆盖条件，字符串 "true" 或 "false" ；通过关键字传入	string	否
kwargs	可选请求参数，使用本文列出的参数	dict	否

MultipartUpload 结构：

```
# -*- coding: utf-8 -*-
{
    'Part': [
        {'PartNumber': 1, 'ETag': "..."},
        {'PartNumber': 3, 'ETag': "..."},
        # 按 PartNumber 升序排列
    ]
}
```

参数名称	参数描述	类型	是否必选
Part	待合并的分块列表，每项含 PartNumber 与 ETag ，且按 PartNumber 升序排列	list	是

响应

返回值为字典：

参数名称	参数描述	类型
Bucket	对象所在的桶名	string
Key	对象键	string

ETag	合并后对象的 ETag	string
------	-------------	--------

错误码

本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

```
# -*- coding=utf-8 -*-
import os
from qcloud_cos import CosConfig, CosS3Client

# 高性能桶经 VPC 内网 HTTP 80 端口接入，SDK 的网关 DNS 负载均衡也仅对 HTTP 请求生效；SDK 默认 https，此处需显式指定 http。
config = CosConfig(
    Scheme='http',
    Region='ap-guangzhou',
    SecretId=os.environ['COS_SECRET_ID'],
    SecretKey=os.environ['COS_SECRET_KEY'],
    Token=os.environ.get('COS_SECURITY_TOKEN'),
    EnableRapidDomain=True)
client = CosS3Client(config)

bucket = 'examplebucket-x--1250000000'
key = 'multipart-example.txt'

# 初始化分块上传。
init_resp = client.create_multipart_upload(Bucket=bucket, Key=key)
upload_id = init_resp['UploadId']

# 编号可以不连续；非最后一块至少 1MB。
parts = []
for part_number, data in ((1, b'a' * (1024 * 1024)), (3, b'last part')):
    resp = client.upload_part(
        Bucket=bucket, Key=key, Body=data,
        PartNumber=part_number, UploadId=upload_id)
    parts.append({'PartNumber': part_number, 'ETag': resp['ETag']})

# 按 PartNumber 升序提交完成。
result = client.complete_multipart_upload(
```

```
Bucket=bucket,  
Key=key,  
UploadId=upload_id,  
MultipartUpload={'Part': parts},  
ForbidOverwrite='true')  
print('complete OK, ETag: %s' % result['ETag'])
```

终止分块上传

最近更新时间：2026-10-21 18:34:06

放弃一个未完成的分块上传（Abort Multipart Upload），并删除已上传的所有分块以释放存储空间。对高性能桶发起请求时，SDK 自动完成会话鉴权。

注意事项

- `Key` 与 `UploadId` 必须对应同一次初始化分块上传；`Key` 表示不以 `/` 结尾的文件对象，不能改成目录对象键。
- 终止后，该 `UploadId` 下已上传的分块会被删除且不可恢复。
- 已初始化但未完成的分块上传会占用存储空间并产生存储费用，不再需要时应及时终止；可通过 [Python SDK 列举分块上传](#) 查询桶内未完成的分块上传任务。
- 若终止时仍有正在上传该 `UploadId` 分块的请求，这些 [Python SDK 上传分块](#) 请求会失败；`UploadId` 不存在会返回 `NoSuchUpload`。
- 终止分块上传属于数据面接口，SDK 使用 session 临时凭证签名。

请求

方法原型

```
def abort_multipart_upload(self, Bucket, Key, UploadId, **kwargs)
```

参数说明

参数名称	参数描述	类型	是否必选
Bucket	高性能桶完整名称，格式为 <code><BucketName>-x--<APPID></code>	string	是
Key	对象键（Object Key）	string	是
UploadId	初始化分块上传返回的 <code>UploadId</code>	string	是
kwargs	可选请求参数，使用本文列出的参数	dict	否

响应

参数名称	参数描述	类型
------	------	----

None	终止成功无返回内容，对应 HTTP 204	None
------	-----------------------	------

错误码

本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

```
# -*- coding=utf-8 -*-
import os
from qcloud_cos import CosConfig, CosS3Client

# 高性能桶经 VPC 内网 HTTP 80 端口接入，SDK 的网关 DNS 负载均衡也仅对 HTTP 请求生效；SDK 默认 https，此处需显式指定 http。
config = CosConfig(
    Scheme='http',
    Region='ap-guangzhou',
    SecretId=os.environ['COS_SECRET_ID'],
    SecretKey=os.environ['COS_SECRET_KEY'],
    Token=os.environ.get('COS_SECURITY_TOKEN'),
    EnableRapidDomain=True)
client = CosS3Client(config)

client.abort_multipart_upload(
    Bucket='examplebucket-x--1250000000',
    Key='exampleobject.txt',
    UploadId='exampleUploadId')
print('AbortMultipartUpload OK')
```

列举分块上传

最近更新时间：2026-10-21 18:34:06

列出桶内所有已初始化但未完成的分块上传任务（List Multipart Uploads），常用于清理长期未完成、占用存储空间的分块上传。对高性能桶发起请求时，SDK 自动完成会话鉴权。

注意事项

- `Prefix`、`Delimiter` 保持默认空字符串；非空值返回 `400 InvalidArgument`。如需筛选对象键，请分页获取后在客户端处理，SDK 不自动过滤结果。
- 单次列出数量取值范围1~1000，`IsTruncated` 为 `true` 时需用 `NextKeyMarker` 与 `NextUploadIdMarker` 翻页继续列举。
- 高性能桶的 `UploadId` 游标保持原样，不做 URL 解码。
- 若要查询某个 `UploadId` 下已上传的具体分块，请使用 [Python SDK 列举已上传分块](#)。
- 列举分块上传属于数据面接口，SDK 使用 session 临时凭证签名。

请求

方法原型

```
def list_multipart_uploads(self, Bucket, Prefix="", Delimiter="",
                             KeyMarker="",
                             UploadIdMarker="", MaxUploads=1000,
                             EncodingType="", **kwargs)
```

参数说明

参数名称	参数描述	类型	是否必选
Bucket	高性能桶完整名称，格式为 <code><BucketName>-x--<APPID></code>	string	是
Prefix	固定为空字符串，请保持默认值	string	否
Delimiter	固定为空字符串，请保持默认值	string	否
KeyMarker	从该对象键之后开始列举，用于翻页	string	否

UploadIdMarker	与 <code>KeyMarker</code> 配合使用，从该 <code>UploadId</code> 之后开始列举	string	否
MaxUploads	单次最多返回的分块上传数量，取值范围 1~1000，默认 1000	int	否
EncodingType	返回结果的编码方式，可选值： <code>url</code>	string	否
kwargs	可选请求参数，使用本文列出的参数	dict	否

响应

返回值为解析后的 dict：

参数名称	参数描述	类型
Bucket	桶名	string
KeyMarker	本次列举起始的对象键标记	string
UploadIdMarker	本次列举起始的 <code>UploadId</code> 标记	string
NextKeyMarker	下一页列举的对象键标记	string
NextUploadIdMarker	下一页列举的 <code>UploadId</code> 标记	string
MaxUploads	本次列举的最大数量	string
IsTruncated	结果是否被截断，为 true 时需用 Next 标记翻页	string
Upload	分块上传任务列表，每项含 <code>Key</code> 、 <code>UploadId</code> 、 <code>Initiated</code>	list

错误码

本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

```
# -*- coding=utf-8 -*-
import os
from qcloud_cos import CosConfig, CosS3Client
```

高性能桶经 VPC 内网 HTTP 80 端口接入, SDK 的网关 DNS 负载均衡也仅对 HTTP 请求生效; SDK 默认 https, 此处需显式指定 http。

```
config = CosConfig(  
    Scheme='http',  
    Region='ap-guangzhou',  
    SecretId=os.environ['COS_SECRET_ID'],  
    SecretKey=os.environ['COS_SECRET_KEY'],  
    Token=os.environ.get('COS_SECURITY_TOKEN'),  
    EnableRapidDomain=True)  
client = CosS3Client(config)  
  
# 筛选在客户端进行; 服务端请求保持空 Prefix/Delimiter。  
wanted_prefix = 'uploads/'  
key_marker = ''  
upload_id_marker = ''  
while True:  
    resp = client.list_multipart_uploads(  
        Bucket='examplebucket-x--1250000000', Prefix='', Delimiter='',  
        KeyMarker=key_marker, UploadIdMarker=upload_id_marker,  
        MaxUploads=100)  
    for up in resp.get('Upload') or []:  
        if up['Key'].startswith(wanted_prefix):  
            print('Key: %s, UploadId: %s' % (up['Key'], up['UploadId']))  
    if resp.get('IsTruncated') != 'true':  
        break  
    key_marker = resp['NextKeyMarker']  
    upload_id_marker = resp['NextUploadIdMarker']
```

列举已上传分块

最近更新时间：2026-10-21 18:34:07

列出某个 `UploadId` 下已上传的所有分块（List Parts），常用于 [Python SDK 完成分块上传](#) 前核对分块信息，或在断点续传场景中确认已上传分块。对高性能桶发起请求时，SDK 自动完成会话鉴权。

注意事项

- `Key` 与 `UploadId` 必须对应同一次初始化分块上传；`Key` 表示不以 `/` 结尾的文件对象，不能改成目录对象键。
- `IsTruncated` 为 `true` 时表示还有更多分块，需将 `NextPartNumberMarker` 原样作为下一次的 `PartNumberMarker` 继续翻页，不要将游标当作分块编号自行计算。
- 完整分页获取结果后，按 `PartNumber` 的数值升序整理 `PartNumber` 与 `ETag`，再组装 [Python SDK 完成分块上传](#) 所需的 `MultipartUpload`。
- 若要列举桶内所有未完成的分块上传任务（而非某个 `UploadId` 的分块），请使用 [Python SDK 列举分块上传](#)。
- 列举已上传分块属于数据面接口，SDK 使用 session 临时凭证签名。

请求

方法原型

```
def list_parts(self, Bucket, Key, UploadId, EncodingType='',
              MaxParts=1000,
              PartNumberMarker=0, **kwargs)
```

参数说明

参数名称	参数描述	类型	是否必选
Bucket	高性能桶完整名称，格式为 <code><BucketName>-x--<APPID></code>	string	是
Key	对象键（Object Key）	string	是
UploadId	初始化分块上传返回的 <code>UploadId</code>	string	是

EncodingType	返回结果的编码方式，可选值： <code>url</code>	string	否
MaxParts	单次最多返回的分块数量，取值范围1~1000，默认1000	int	否
PartNumberMarker	分页游标；首次调用可不传（默认为 <code>0</code> ），后续原样使用响应的 <code>NextPartNumberMarker</code>	string / int	否
kwargs	可选请求参数，使用本文列出的参数	dict	否

响应

返回值为解析后的 dict：

参数名称	参数描述	类型
Bucket	桶名	string
Key	对象键	string
UploadId	分块上传标识	string
MaxParts	本次列举的最大数量	string
PartNumberMarker	本次请求使用的分页游标	string
NextPartNumberMarker	下一页分页游标，原样传给下次请求	string
IsTruncated	结果是否被截断，为 true 时需翻页	string
Part	已上传分块列表，每项含 <code>PartNumber</code> 、 <code>ETag</code> 、 <code>Size</code> 、 <code>LastModified</code>	list

错误码

本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

```
# -*- coding=utf-8 -*-
import os
from qcloud_cos import CosConfig, CosS3Client

# 高性能桶经 VPC 内网 HTTP 80 端口接入, SDK 的网关 DNS 负载均衡也仅对 HTTP 请求生效; SDK 默认 https, 此处需显式指定 http。
config = CosConfig(
    Scheme='http',
    Region='ap-guangzhou',
    SecretId=os.environ['COS_SECRET_ID'],
    SecretKey=os.environ['COS_SECRET_KEY'],
    Token=os.environ.get('COS_SECURITY_TOKEN'),
    EnableRapidDomain=True)
client = CosS3Client(config)

marker = 0
while True:
    resp = client.list_parts(
        Bucket='examplebucket-x--1250000000',
        Key='exampleobject.txt',
        UploadId='exampleUploadId',
        PartNumberMarker=marker)
    for part in resp.get('Part') or []:
        print('PartNumber: %s, ETag: %s, Size: %s' % (
            part['PartNumber'], part['ETag'], part.get('Size')))
    if resp.get('IsTruncated') != 'true':
        break
    marker = resp['NextPartNumberMarker']
```

异常处理

最近更新时间：2026-10-21 18:34:03

本文档说明高性能桶 Python SDK 的错误处理方式。可按常见的客户端与服务端异常读取错误码、错误信息、请求 ID 及 HTTP 状态码，便于定位与处理问题。

错误类型

SDK 的异常定义在 `qcloud_cos.cos_exception` 中，调用方可根据异常类型区分错误来源：

异常类型	说明
<code>CosClientError</code>	客户端错误。如请求超时、网络异常，或 SDK 在本地校验失败（例如对高性能桶传入了不支持的参数、未启用 session 鉴权、Key 含非法路径段等）
<code>CosServiceError</code>	服务端错误。服务端返回了错误响应，可进一步读取错误码、错误信息与诊断 ID

`CosServiceError` 提供以下方法：

方法	描述
<code>get_error_code()</code>	获取错误码，如 <code>NoSuchBucket</code> 、 <code>AccessDenied</code> ；服务端返回格式异常时返回 <code>Unknown</code>
<code>get_error_msg()</code>	获取错误的详细描述信息
<code>get_status_code()</code>	获取 HTTP 状态码
<code>get_resource_location()</code>	获取发生错误的资源
<code>get_request_id()</code>	获取本次请求的唯一标识，排查问题时可提供给技术支持
<code>get_trace_id()</code>	获取服务端调用链追踪 ID
<code>get_origin_msg()</code>	获取服务端返回的原始错误内容
<code>get_digest_msg()</code>	获取解析后的 dict 形式错误信息

HEAD 请求没有错误正文时，SDK 可能使用通用错误码，例如 404 对应 `NoSuchResource`。判断对象或桶是否存在时，请使用 `get_status_code() == 404`。

示例

判断错误类型并提取错误码、错误信息、请求 ID 与 HTTP 状态码：

```
# -*- coding=utf-8 -*-
import os
from qcloud_cos import CosConfig, CosS3Client
from qcloud_cos.cos_exception import CosClientError, CosServiceError

# 高性能桶经 VPC 内网 HTTP 80 端口接入，SDK 的网关 DNS 负载均衡也仅对 HTTP 请求生效；SDK 默认 https，此处需显式指定 http。
config = CosConfig(
    Scheme='http',
    Region='ap-guangzhou',
    SecretId=os.environ['COS_SECRET_ID'],
    SecretKey=os.environ['COS_SECRET_KEY'],
    Token=os.environ.get('COS_SECURITY_TOKEN'),
    EnableRapidDomain=True)
client = CosS3Client(config)

try:
    client.get_object(Bucket='examplebucket-x--1250000000', Key='not-exist-key')
except CosServiceError as e:
    # 服务端错误，可读取错误码、错误信息与诊断 ID。
    print('Code: %s' % e.get_error_code())
    print('Message: %s' % e.get_error_msg())
    print('StatusCode: %s' % e.get_status_code())
    print('RequestId: %s' % e.get_request_id())
except CosClientError as e:
    # 客户端错误，如超时、网络异常或本地参数校验失败。
    print('Client error: %s' % e)
```

常见错误码

错误码	HTTP 状态码	说明
NoSuchBucket	404	请求的桶不存在

NoSuchKey	404	请求的对象不存在
NoSuchPolicy	404	桶未设置存储桶策略
NoSuchUpload	404	分块上传的 <code>UploadId</code> 不存在
AccessDenied	403	无权限访问，请检查 CAM 授权或桶策略
ExpiredToken	400	会话临时凭证已过期，需使用有效临时凭证重新发起请求
InvalidAccessKeyId	403	凭证不存在
BucketNotEmpty	409	删除桶时桶内仍有对象，需先清空对象
EntityTooSmall	400	除最后一块外的分块小于允许的最小值
InvalidPart	400	提交的分块信息无效，例如 ETag 与已上传分块不匹配
InvalidPartOrder	400	完成分块上传时 <code>Parts</code> 未按分块编号升序排列

错误码可结合 `RequestId` 一并用于问题排查。完整错误码列表请参见 [错误码](#)。

Go SDK

快速入门

最近更新时间：2026-10-21 18:34:04

本文档介绍如何使用 Go SDK 访问高性能桶（Rapid Bucket），完成初始化客户端、创建访问会话以及上传下载对象等基础操作。

环境要求

- Go 1.18 及以上版本。
- 已开通高性能桶服务，并已创建高性能桶；也可以按 [Go SDK 创建存储桶](#) 通过 SDK 创建。高性能桶的完整名称形如 `<BucketName>-x--<APPID>`，例如 `examplebucket-x--1250000000`。
- 已获取访问密钥 `SecretId` 和 `SecretKey`。建议使用子账号密钥并遵循最小权限原则。

安装 SDK

在项目中引入 COS Go SDK：

```
go get github.com/tencentyun/cos-go-sdk-v5
```

在代码中导入：

```
import "github.com/tencentyun/cos-go-sdk-v5"
```

鉴权说明

访问高性能桶时，统一使用 `RapidCredentialTransport` 包装底层基础凭证。它会在需要时自动调用 `CreateSession` 换取临时凭证并完成签名，临时凭证在有效期内自动缓存复用，底层基础凭证变更后自动重新签发，业务代码无需手动管理会话与临时凭证。

只需在初始化客户端时用 `NewRapidCredentialTransport` 包裹基础凭证，后续所有接口调用方式与普通 COS SDK 一致。

初始化客户端

访问高性能桶时，必须使用 `NewRapidCredentialTransport` 包裹底层基础凭证，并将其作为 `http.Client` 的 `Transport` 传入 `NewClient`：

```
package main
```

```
import (
    "context"
    "net/http"
    "net/url"
    "os"
    "time"

    "github.com/tencentyun/cos-go-sdk-v5"
)

func main() {
    // 高性能桶域名形如 https://<BucketName>-x--<APPID>.cosrapid.
    <Region>.myqcloud.com
    u, _ := url.Parse("https://examplebucket-x--1250000000.cosrapid.ap-
    guangzhou.myqcloud.com")
    b := &cos.BaseURL{BucketURL: u}

    // 底层基础凭证 transport, 提供 SecretId / SecretKey。
    // RapidCredentialTransport 用它签发 CreateSession, 并继承其网络传输层。
    base := &cos.AuthorizationTransport{
        SecretID: os.Getenv("SECRETID"),
        SecretKey: os.Getenv("SECRETKEY"),
    }

    // 关键: 用 NewRapidCredentialTransport 包裹底层凭证, 作为
    http.Client.Transport。
    c := cos.NewClient(b, &http.Client{
        Transport: cos.NewRapidCredentialTransport(base),
        Timeout:   30 * time.Second,
    })

    ctx := context.Background()
    _ = ctx
    _ = c
}
```

字段说明:

参数	描述
----	----

BucketURL	高性能桶访问域名，格式为 <code>https://<BucketName>-x--<APPID>.cosrap id.<Region>.myqcloud.com</code>
AuthorizationTransport.SecretID	主账号或子账号的 SecretID
AuthorizationTransport.SecretKey	主账号或子账号的 SecretKey
NewRapidCredentialTransport	将基础凭证 transport 包裹为高性能桶专用 transport，自动完成会话鉴权

① 说明：

底层基础凭证 transport 除 `AuthorizationTransport` 外，也可使用 `CVMCredentialTransport`、`StsCredentialTransport`、`CredentialTransport` 等实现了 `BaseTransport` 接口的类型。

完整示例

以下示例演示初始化客户端后完成一次上传与下载。使用时将下面的代码替换上文 `main` 函数中 `ctx := context.Background()` 及其后的两行，并在 `import` 中补充 `"fmt"`、`"io"`、`"log"`、`"strings"`：

```
ctx := context.Background()

// 上传对象。首次对高性能桶发起请求时，transport 会自动创建 session 并签名。
key := "quickstart/hello.txt"
_, err := c.Object.Put(ctx, key, strings.NewReader("hello rapid
bucket"), nil)
if err != nil {
    log.Fatalf("PutObject failed: %v", err)
}

// 下载对象。
resp, err := c.Object.Get(ctx, key, nil)
if err != nil {
    log.Fatalf("GetObject failed: %v", err)
}
defer resp.Body.Close()
data, _ := io.ReadAll(resp.Body)
fmt.Printf("object content: %s\n", data)
```


后续操作

- 存储桶操作：创建存储桶、删除存储桶、查询存储桶、创建访问会话。
- 对象操作：上传对象、下载对象、删除对象、复制对象、重命名对象等。
- 分块上传操作：适用于大文件的分块上传。
- 存储桶策略：设置、获取、删除存储桶策略。

各接口的详细用法请参见对应文档。SDK 调用返回的错误处理方式请参见 [异常处理](#)。

存储桶操作

查询存储桶列表

最近更新时间：2026-10-21 18:34:05

功能说明

列出调用方主账号在指定地域下的所有高性能桶（GET Service，即 ListBuckets）。查询存储桶列表请求发往 service 域名，地域由 service 域名决定。

方法原型

```
func (s *ServiceService) Get(ctx context.Context, opt
... *ServiceGetOptions) (*ServiceGetResult, *Response, error)
```

参数说明

```
type ServiceGetOptions struct {
    MaxKeys int64
    Marker   string
}
```

参数名称	参数描述	类型	是否必选
ctx	请求上下文，用于控制超时与取消	context.Context	是
opt	查询选项，可省略或传 nil	*ServiceGetOptions	否
MaxKeys	单次返回的最大存储桶数量，取值范围 1~2000，默认 2000	int64	否
Marker	分页游标，取值为上一次响应的 NextMarker；首次请求无需携带	string	否

返回结果说明

```
type ServiceGetResult struct {
```

```

    Owner      *Owner
    Buckets    []Bucket
    Marker     string
    NextMarker string
    IsTruncated bool
}

type Bucket struct {
    Name        string
    Region      string
    CreationDate string
}

```

参数名称	参数描述	类型
Owner	存储桶持有者信息	*Owner
Buckets	存储桶列表	[]Bucket
Marker	本次请求使用的分页游标	string
NextMarker	下一页分页游标，仅当 IsTruncated 为 true 时非空	string
IsTruncated	是否仍有未列举完的存储桶，true 表示存在下一页	bool
Bucket.Name	存储桶完整名称，格式为 <BucketName>-x--<APPID>	string
Bucket.Region	存储桶所在地域（对应 XML 中的 Location）	string
Bucket.CreationDate	存储桶创建时间，ISO 8601 格式	string

使用案例

查询存储桶列表请求发往 service 域名。

```

package main

import (
    "context"
    "fmt"
    "log"
    "net/http"

```

```
"net/url"

"os"

"time"

"github.com/tencentyun/cos-go-sdk-v5"
)

func main() {
    // 请求发往 service 域名。
    su, _ := url.Parse("https://service.cosrapid.ap-
guangzhou.myqcloud.com")
    b := &cos.BaseURL{ServiceURL: su}

    base := &cos.AuthorizationTransport{
        SecretID: os.Getenv("SECRETID"),
        SecretKey: os.Getenv("SECRETKEY"),
    }
    c := cos.NewClient(b, &http.Client{
        Transport: cos.NewRapidCredentialTransport(base),
        Timeout:   30 * time.Second,
    })

    ctx := context.Background()

    // 分页列举该地域下的全部存储桶。
    marker := ""
    for {
        res, _, err := c.Service.Get(ctx, &cos.ServiceGetOptions{
            MaxKeys: 2000,
            Marker:  marker,
        })
        if err != nil {
            log.Fatalf("ListBuckets failed: %v", err)
        }
        for _, bucket := range res.Buckets {
            fmt.Printf("bucket: %s, region: %s, created: %s\n",
                bucket.Name, bucket.Region, bucket.CreationDate)
        }
        if !res.IsTruncated {
            break
        }
    }
}
```

```
    }  
    marker = res.NextMarker  
  }  
}
```

注意事项

- 查询存储桶列表请求发往 **service 域名** `https://service.cosrapid.<Region>.myqcloud.com`，该域名不带存储桶名前缀。
- 本接口仅返回 **service 域名**对应 `<Region>` 下的存储桶；如需查看其他地域的存储桶，请更换 **service 域名**中的地域后分别调用。
- 当 `IsTruncated` 为 `true` 时，将响应中的 `NextMarker` 作为下次请求的 `Marker` 传入，直到 `IsTruncated` 为 `false`。
- 本接口只能列出签名密钥所属账号的存储桶。

错误处理方式请参见 [异常处理](#)。

创建访问会话

最近更新时间：2026-10-21 18:34:06

功能说明

为高性能桶创建访问会话（CreateSession），返回可用于后续请求的临时凭证，是高性能桶专属能力。在使用 Go SDK 访问高性能桶时，RapidCredentialTransport 会自动调用本接口并缓存复用临时凭证，通常无需手动调用；本文档说明其手动调用方式及返回结构，便于需要自行管理临时凭证的场景使用。

方法原型

```
func (s *BucketService) CreateSession(ctx context.Context)
(*CreateSessionResult, *Response, error)
```

参数说明

参数名称	参数描述	类型	是否必选
ctx	请求上下文，用于控制超时与取消	context.Context	是

会话的读写模式（ReadWrite / ReadOnly）由服务端根据调用方在 CAM / 桶策略中被授予的权限自动决定，Go SDK 不通过入参指定。

返回结果说明

```
type CreateSessionResult struct {
    Credentials SessionCredential
}

type SessionCredential struct {
    SecretID      string
    SecretKey     string
    SessionToken  string
    Expiration    time.Time
}
```

参数名称	参数描述	类型
Credentials.SecretID	临时凭证 AccessKeyId（临时 SecretId）	string

Credentials.SecretKey	临时凭证 SecretAccessKey (临时 SecretKey)	string
Credentials.SessionToken	会话令牌，后续请求需携带	string
Credentials.Expiration	临时凭证过期时间	time.Time

使用案例

```
package main

import (
    "context"
    "fmt"
    "log"
    "net/http"
    "net/url"
    "os"
    "time"

    "github.com/tencentyun/cos-go-sdk-v5"
)

func main() {
    u, _ := url.Parse("https://examplebucket-x--1250000000.cosrapid.ap-guangzhou.myqcloud.com")
    b := &cos.BaseURL{BucketURL: u}
    base := &cos.AuthorizationTransport{
        SecretID:  os.Getenv("SECRETID"),
        SecretKey: os.Getenv("SECRETKEY"),
    }
    c := cos.NewClient(b, &http.Client{
        Transport: cos.NewRapidCredentialTransport(base),
        Timeout:   30 * time.Second,
    })

    ctx := context.Background()
    res, _, err := c.Bucket.CreateSession(ctx)
    if err != nil {
        log.Fatalf("CreateSession failed: %v", err)
    }
}
```

```
}

cred := res.Credentials
fmt.Printf("AccessKeyId: %s\n", cred.SecretID)
fmt.Printf("SessionToken: %s\n", cred.SessionToken)
fmt.Printf("Expiration: %s\n", cred.Expiration.Format(time.RFC3339))
}
```

注意事项

- 临时凭证具有桶级作用域，仅对创建会话时指定的高性能桶有效。
- 临时凭证有效期较短，过期后需重新创建会话。使用 `RapidCredentialTransport` 时，SDK 会在凭证接近过期时自动重新签发。
- 若返回的凭证不完整或过期时间缺失/非法，SDK 会直接返回错误，不会下发无效凭证。

错误处理方式请参见 [异常处理](#)。

创建存储桶

最近更新时间：2026-10-21 18:34:06

功能说明

在指定地域创建一个高性能桶（PUT Bucket）。创建桶请求发往 service 域名，要创建的目标桶名通过 `client.Host` 单独指定。高性能桶创建时必须指定 VPC 网络相关信息，通过自定义请求头传入。

方法原型

```
func (s *BucketService) Put(ctx context.Context, opt *BucketPutOptions) (*Response, error)
```

参数说明

参数名称	参数描述	类型	是否必选
ctx	请求上下文，用于控制超时与取消	context.Context	是
opt	创建桶选项，VPC 网络信息通过 <code>XOptionHeader</code> 传入	*BucketPutOptions	是

高性能桶创建时必须携带以下自定义请求头（通过 `opt.XOptionHeader` 设置），缺任意一个都会返回 `InvalidArgument`：

请求头	参数描述	类型	是否必选
x-cos-vpc-id	VPC ID	string	是
x-cos-subnet-id	子网 ID	string	是
x-cos-zone	可用区	string	是

返回结果说明

参数名称	参数描述	类型
*Response	响应对象，创建成功状态码为 200	*cos.Response
error	错误信息，成功时为 nil	error

使用案例

创建桶请求发往 **service** 域名，`client.Host` 指定要创建的目标桶名（`<BucketName>-x--<APPID>`），VPC 网络信息通过 `XOptionHeader` 传入。

```
package main

import (
    "context"
    "log"
    "net/http"
    "net/url"
    "os"
    "time"

    "github.com/tencentyun/cos-go-sdk-v5"
)

func main() {
    // 请求发往 service 域名。
    su, _ := url.Parse("https://service.cosrapid.ap-
guangzhou.myqcloud.com")
    b := &cos.BaseURL{BucketURL: su, ServiceURL: su}

    base := &cos.AuthorizationTransport{
        SecretID: os.Getenv("SECRETID"),
        SecretKey: os.Getenv("SECRETKEY"),
    }
    c := cos.NewClient(b, &http.Client{
        Transport: cos.NewRapidCredentialTransport(base),
        Timeout:   30 * time.Second,
    })

    // 用 client.Host 单独指定要创建的目标桶名。
    c.Host = "examplebucket-x--1250000000.cosrapid.ap-
guangzhou.myqcloud.com"

    // VPC 网络信息通过自定义请求头传入。
    header := &http.Header{}
    header.Set("x-cos-vpc-id", "vpc-xxxxxxx")
}
```

```
header.Set("x-cos-subnet-id", "subnet-xxxxxxx")
header.Set("x-cos-zone", "ap-guangzhou-3")

ctx := context.Background()
_, err := c.Bucket.Put(ctx, &cos.BucketPutOptions{XOptionHeader:
header})
if err != nil {
    log.Fatalf("PutBucket failed: %v", err)
}
log.Println("bucket created")
}
```

注意事项

- 创建桶请求发往 **service 域名** `https://service.cosrapid.<Region>.myqcloud.com`，要创建的目标桶名由 `client.Host` 单独指定，格式为 `<BucketName>-x--<APPID>.cosrapid.<Region>.myqcloud.com`。
- `x-cos-vpc-id`、`x-cos-subnet-id`、`x-cos-zone` 为必需请求头，缺任意一个都会返回 `InvalidArgument`。
- 桶名在同一地域内需全局唯一，重复创建已存在的桶会返回错误。

错误处理方式请参见 [异常处理](#)。

删除存储桶

最近更新时间：2026-10-21 18:34:06

功能说明

删除一个高性能桶（DELETE Bucket）。删除的目标桶由客户端初始化时的访问域名决定。

方法原型

```
func (s *BucketService) Delete(ctx context.Context, opt
... *BucketDeleteOptions) (*Response, error)
```

参数说明

参数名称	参数描述	类型	是否必选
ctx	请求上下文，用于控制超时与取消	context.Context	是
opt	删除桶选项，可省略，通常直接 <code>c.Bucket.Delete</code> (ctx)	... *BucketDeleteOptions	否

返回结果说明

参数名称	参数描述	类型
*Response	响应对象，删除成功状态码为 204	*cos.Response
error	错误信息，成功时为 nil	error

使用案例

```
package main

import (
    "context"
    "log"
    "net/http"
    "net/url"
    "os"
```

```
"time"

"github.com/tencentyun/cos-go-sdk-v5"
)

func main() {
    u, _ := url.Parse("https://examplebucket-x--1250000000.cosrapid.ap-
guangzhou.myqcloud.com")
    b := &cos.BaseURL{BucketURL: u}
    base := &cos.AuthorizationTransport{
        SecretID: os.Getenv("SECRETID"),
        SecretKey: os.Getenv("SECRETKEY"),
    }
    c := cos.NewClient(b, &http.Client{
        Transport: cos.NewRapidCredentialTransport(base),
        Timeout:   30 * time.Second,
    })

    ctx := context.Background()
    _, err := c.Bucket.Delete(ctx)
    if err != nil {
        log.Fatalf("DeleteBucket failed: %v", err)
    }
    log.Println("bucket deleted")
}
```

注意事项

- 删除桶前必须先清空桶内所有对象，否则会返回 `BucketNotEmpty` 错误。
- 删除的目标桶由客户端初始化时的访问域名决定，请确认域名指向待删除的桶。
- 桶删除后不可恢复，请谨慎操作。

错误处理方式请参见 [异常处理](#)。

查询存储桶

最近更新时间：2026-10-21 18:34:07

功能说明

查询一个高性能桶是否存在以及当前调用方是否有权限访问（HEAD Bucket）。目标桶由客户端初始化时的访问域名决定。

方法原型

```
func (s *BucketService) Head(ctx context.Context, opt
...*BucketHeadOptions) (*Response, error)
```

参数说明

参数名称	参数描述	类型	是否必选
ctx	请求上下文，用于控制超时与取消	context.Context	是
opt	查询桶选项，可省略，通常直接 <code>c.Bucket.Head(c</code> <code>tx)</code>	...*BucketHeadOptions	否

返回结果说明

参数名称	参数描述	类型
*Response	响应对象，可通过 <code>Response.StatusCode</code> 判断结果：桶存在且有权限返回 200，无权限返回 403，不存在返回 404	*cos.Response
error	错误信息，桶存在且有权限时为 nil	error

使用案例

```
package main

import (
    "context"
    "log"
```

```
"net/http"
"net/url"
"os"
"time"

"github.com/tencentyun/cos-go-sdk-v5"
)

func main() {
    u, _ := url.Parse("https://examplebucket-x--1250000000.cosrapid.ap-
guangzhou.myqcloud.com")
    b := &cos.BaseURL{BucketURL: u}
    base := &cos.AuthorizationTransport{
        SecretID: os.Getenv("SECRETID"),
        SecretKey: os.Getenv("SECRETKEY"),
    }
    c := cos.NewClient(b, &http.Client{
        Transport: cos.NewRapidCredentialTransport(base),
        Timeout:   30 * time.Second,
    })

    ctx := context.Background()

    // 方式一：直接调用 Head，根据错误与状态码判断。
    _, err := c.Bucket.Head(ctx)
    if err != nil {
        log.Fatalf("HeadBucket failed: %v", err)
    }
    log.Println("bucket exists and accessible")

    // 方式二：使用 IsExist 封装，直接返回桶是否存在。
    exist, err := c.Bucket.IsExist(ctx)
    if err != nil {
        log.Fatalf("IsExist failed: %v", err)
    }
    log.Printf("bucket exist: %v", exist)
}
```

注意事项

- `Head` 返回的 HTTP 状态码可用于区分结果：200 表示桶存在且有权限，403 表示桶存在但无访问权限，404 表示桶不存在。
- SDK 提供 `func (s *BucketService) IsExist(ctx context.Context) (bool, error)`，基于 `Head` 与 `IsNotFoundError` 封装，桶存在返回 `true`，404 时返回 `false`，其他错误原样返回，适合仅需判断桶是否存在的场景。

错误处理方式请参见 [异常处理](#)。

存储桶策略

设置存储桶策略

最近更新时间：2026-10-21 18:34:06

功能说明

为高性能桶设置存储桶策略（PUT Bucket Policy），通过声明式的授权语句控制账号对桶及对象的访问权限。策略存储于 CAM，proxy 完成 COS 形态与 CAM 形态的转换。

方法原型

```
func (s *BucketService) PutPolicy(ctx context.Context, opt
*BucketPutPolicyOptions) (*Response, error)
```

参数说明

```
type BucketPutPolicyOptions struct {
    Statement []BucketStatement
    Version   string
    Principal map[string][]string
}

type BucketStatement struct {
    Principal map[string][]string
    Action    []string
    Effect     string
    Resource  []string
    Condition map[string]map[string]interface{}
    Sid       string
}
```

参数名称	参数描述	类型	是否必选
ctx	请求上下文，用于控制超时与取消	context.Context	是
opt	桶策略内容	*BucketPutPolicyOptions	是

Statement	授权语句列表	[]BucketStatement	是
Version	策略语法版本，通常为 2.0	string	否
Principal	被授权者，key 仅支持 qcs 与 service；qcs 形如 qcs::cam::uin/<主账号uin>:uin/<子账号uin>	map[string][]string	是
Action	授权的操作列表，形如 name/cos:CreateSession、name/cos:GetObject	[]string	是
Effect	授权效果，取值 allow 或 deny（大小写不敏感）	string	是
Resource	授权作用的资源，需精确匹配当前请求的桶	[]string	是
Condition	生效条件，原样透传 CAM	map[string]map[string]interface{}	否
Sid	语句标识符	string	否

返回结果说明

参数名称	参数描述	类型
*Response	响应对象，设置成功状态码为 204	*cos.Response
error	错误信息，成功时为 nil	error

使用案例

```
package main

import (
    "context"
    "log"
    "net/http"
    "net/url"
    "os"
    "time"

    "github.com/tencentyun/cos-go-sdk-v5"
```

```
)

func main() {
    u, _ := url.Parse("https://examplebucket-x--1250000000.cosrapid.ap-
guangzhou.myqcloud.com")
    b := &cos.BaseURL{BucketURL: u}
    base := &cos.AuthorizationTransport{
        SecretID: os.Getenv("SECRETID"),
        SecretKey: os.Getenv("SECRETKEY"),
    }
    c := cos.NewClient(b, &http.Client{
        Transport: cos.NewRapidCredentialTransport(base),
        Timeout:   30 * time.Second,
    })

    ctx := context.Background()
    opt := &cos.BucketPutPolicyOptions{
        Version: "2.0",
        Statement: []cos.BucketStatement{
            {
                Sid:      "allow-get-and-session",
                Effect: "allow",
                Principal: map[string][]string{
                    "qcs":
{"qcs::cam::uin/100000000001:uin/100000000011"},
                },
                Action: []string{
                    "name/cos:CreateSession",
                    "name/cos:GetObject",
                },
                Resource: []string{
                    "qcs::cosrapid:ap-
guangzhou:uid/1250000000:examplebucket-x--1250000000/*",
                },
            },
        },
    }

    _, err := c.Bucket.PutPolicy(ctx, opt)
    if err != nil {
        log.Fatalf("PutPolicy failed: %v", err)
    }
}
```

```
}  
log.Println("bucket policy set")  
}
```

注意事项

- `Resource` 必须精确匹配当前请求的桶，高性能桶的资源形态为 `qcs::cosrapid:<Region>:uid/<APPID>:<BucketName>-x--<APPID>/*`；指向其他桶、跨地域或通配 `*` 均会被拒绝（`InvalidArgument`）。
- `Principal` 的 key 仅支持 `qcs` 与 `service`；`qcs` 形如 `qcs::cam::uin/<主账号uin>:uin/<子账号uin>`。
- `Effect` 取值为 `allow` 或 `deny`；`Action` 原样透传 CAM。
- `PutPolicy` 会以本次提交的策略整体覆盖桶上原有策略，而非增量合并，需提交完整的目标策略集合。
- 仅桶所有者可调用，非所有者调用返回 405 MethodNotAllowed。

错误处理方式请参见 [异常处理](#)。

获取存储桶策略

最近更新时间：2026-10-21 18:34:06

功能说明

获取高性能桶当前生效的存储桶策略（GET Bucket Policy）。返回内容为大写 key 的 JSON，Version 固定为 2.0。

方法原型

```
func (s *BucketService) GetPolicy(ctx context.Context)
(*BucketGetPolicyResult, *Response, error)
```

参数说明

参数名称	参数描述	类型	是否必选
ctx	请求上下文，用于控制超时与取消	context.Context	是

返回结果说明

BucketGetPolicyResult 与 BucketPutPolicyOptions 字段相同：

```
type BucketGetPolicyResult struct {
    Statement []BucketStatement
    Version   string
    Principal map[string][]string
}

type BucketStatement struct {
    Principal map[string][]string
    Action    []string
    Effect     string
    Resource  []string
    Condition map[string]map[string]interface{}
    Sid       string
}
```

参数名称	参数描述	类型
------	------	----

Version	策略语法版本，固定为2.0	string
Statement	授权语句列表	[]BucketStatement
Statement.Action	授权的操作列表	[]string
Statement.Effect	授权效果， allow 或 deny	string
Statement.Resource	授权作用的资源，转回 COS 完整桶名形态	[]string
Statement.Principal	被授权者	map[string][]string
Statement.Sid	语句标识符	string

使用案例

```
package main

import (
    "context"
    "fmt"
    "log"
    "net/http"
    "net/url"
    "os"
    "time"

    "github.com/tencentyun/cos-go-sdk-v5"
)

func main() {
    u, _ := url.Parse("https://examplebucket-x--1250000000.cosrapid.ap-
guangzhou.myqcloud.com")
    b := &cos.BaseURL{BucketURL: u}
    base := &cos.AuthorizationTransport{
        SecretID:  os.Getenv("SECRETID"),
        SecretKey: os.Getenv("SECRETKEY"),
    }
    c := cos.NewClient(b, &http.Client{
        Transport: cos.NewRapidCredentialTransport(base),
        Timeout:   30 * time.Second,
```

```
    })

    ctx := context.Background()
    res, _, err := c.Bucket.GetPolicy(ctx)
    if err != nil {
        log.Fatalf("GetPolicy failed: %v", err)
    }

    fmt.Printf("Version: %s\n", res.Version)
    for _, stmt := range res.Statement {
        fmt.Printf("Sid: %s, Effect: %s, Action: %v, Resource: %v\n",
            stmt.Sid, stmt.Effect, stmt.Action, stmt.Resource)
    }
}
```

注意事项

- 当桶上未设置任何策略时，`GetPolicy` 会返回错误，对应 `404 NoSuchPolicy`，可通过错误码或 `IsNotFound` 判断。
- 错误处理方式请参见 [异常处理](#)。

删除存储桶策略

最近更新时间：2026-10-21 18:34:07

功能说明

删除高性能桶当前设置的存储桶策略（DELETE Bucket Policy）。

方法原型

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

参数说明

参数名称	参数描述	类型	是否必选
ctx	请求上下文，用于控制超时与取消	context.Context	是

返回结果说明

参数名称	参数描述	类型
*Response	响应对象，删除成功状态码为 204	*cos.Response
error	错误信息，成功时为 nil	error

使用案例

```
package main

import (
    "context"
    "log"
    "net/http"
    "net/url"
    "os"
    "time"

    "github.com/tencentyun/cos-go-sdk-v5"
)
```



```
func main() {
    u, _ := url.Parse("https://examplebucket-x--1250000000.cosrapid.ap-
guangzhou.myqcloud.com")
    b := &cos.BaseURL{BucketURL: u}
    base := &cos.AuthorizationTransport{
        SecretID: os.Getenv("SECRETID"),
        SecretKey: os.Getenv("SECRETKEY"),
    }
    c := cos.NewClient(b, &http.Client{
        Transport: cos.NewRapidCredentialTransport(base),
        Timeout:   30 * time.Second,
    })

    ctx := context.Background()
    _, err := c.Bucket.DeletePolicy(ctx)
    if err != nil {
        log.Fatalf("DeletePolicy failed: %v", err)
    }
    log.Println("bucket policy deleted")
}
```

注意事项

- 删除桶策略后，桶将恢复为仅由 CAM 授权控制访问，请确认相关账号仍具备所需的 CAM 权限，避免误删策略导致访问中断。
- 删除后桶上将不再有任何授权策略，操作不可逆，请谨慎调用。

错误处理方式请参见 [异常处理](#)。

对象操作

重命名对象

最近更新时间：2026-10-21 18:34:06

功能说明

在同一高性能桶内原子重命名对象。该操作仅修改对象键，不搬移底层数据，是高性能桶专属能力。对高性能桶发起请求时，SDK 通过 `RapidCredentialTransport` 自动完成会话鉴权。

方法原型

```
func (s *ObjectService) RenameObject(ctx context.Context, name,
renameSource string, opt *ObjectRenameOptions) (*Response, error)
```

参数说明

```
type ObjectRenameOptions struct {
    XCosForbidOverwrite string
}
```

参数名称	参数描述	类型	是否必选
ctx	请求上下文，用于控制超时与取消	context.Context	是
name	目标对象键（Object Key）	string	是
renameSource	源对象键（Object Key），SDK 内部自动补前导 <code>/</code> 并进行 URL 编码	string	是
opt	重命名选项，可为 nil	*ObjectRenameOptions	否
XCosForbidOverwrite	防覆盖头。置为 <code>"true"</code> 时，若目标对象已存在则拒绝重命名	string	否

返回结果说明

参数名称	参数描述	类型
*Response	响应对象	*cos.Response
error	错误信息，成功时为 nil	error

使用案例

```
package main

import (
    "context"
    "log"
    "net/http"
    "net/url"
    "os"
    "time"

    "github.com/tencentyun/cos-go-sdk-v5"
)

func main() {
    u, _ := url.Parse("https://examplebucket-x--1250000000.cosrapid.ap-
guangzhou.myqcloud.com")
    b := &cos.BaseURL{BucketURL: u}
    base := &cos.AuthorizationTransport{
        SecretID: os.Getenv("SECRETID"),
        SecretKey: os.Getenv("SECRETKEY"),
    }
    c := cos.NewClient(b, &http.Client{
        Transport: cos.NewRapidCredentialTransport(base),
        Timeout:   30 * time.Second,
    })

    ctx := context.Background()
    name := "newname.txt"
    renameSource := "oldname.txt"

    opt := &cos.ObjectRenameOptions{
        XCosForbidOverwrite: "true",
    }
```

```
}  
  
_, err := c.Object.RenameObject(ctx, name, renameSource, opt)  
if err != nil {  
    log.Fatalf("RenameObject failed: %v", err)  
}  
  
log.Println("RenameObject OK")  
}
```

注意事项

- 该操作为原子重命名，不搬移底层数据，**仅高性能桶支持**。普通桶调用会在本地 fail-fast 直接返回错误，请改用 Copy + Delete 实现重命名。
- `name` 为目标 key，`renameSource` 为同桶内的源 key，二者均不能为空或为 `/`。
- 将 `XCosForbidOverwrite` 置为 `"true"` 可防止覆盖已存在的目标对象。

错误处理方式请参见 [异常处理](#)。

上传对象

最近更新时间：2026-10-21 18:34:05

功能说明

上传一个对象（文件/对象）至高性能桶（PUT Object）。对高性能桶发起上传请求时，SDK 通过 `RapidCredentialTransport` 自动完成会话鉴权。

方法原型

```
func (s *ObjectService) Put(ctx context.Context, name string, r
io.Reader, opt *ObjectPutOptions) (*Response, error)
func (s *ObjectService) PutFromFile(ctx context.Context, name string,
filePath string, opt *ObjectPutOptions) (*Response, error)
```

参数说明

```
type ObjectPutOptions struct {
    *ObjectPutHeaderOptions
}

type ObjectPutHeaderOptions struct {
    CacheControl      string
    ContentDisposition string
    ContentEncoding   string
    ContentType       string
    ContentMD5        string
    ContentLength     int64
    ContentLanguage   string
    Expires           string
    // 自定义的 x-cos-meta-* header
    XCosMetaXXX      *http.Header
    XCosStorageClass string
}
```

参数名称	参数描述	类型	是否 必选
------	------	----	----------

ctx	请求上下文，用于控制超时与取消	context.Context	是
name	对象键（Object Key），即对象在桶中的完整路径	string	是
r	上传对象的内容，可为文件流或字节流；当 r 不是 bytes.Buffer/bytes.Reader/strings.Reader 时，必须指定 <code>opt.ObjectPutHeaderOptions.ContentLength</code>	io.Reader	是
opt	上传选项，可为 nil	*ObjectPutOptions	否
ContentType	内容类型，设置 Content-Type	string	否
ContentLength	设置传输长度	int64	否
ContentMD5	请求体内容的 MD5 校验值（Base64 编码），用于完整性校验	string	否
CacheControl	缓存策略，设置 Cache-Control	string	否
ContentDisposition	文件名展示方式，设置 Content-Disposition	string	否
ContentEncoding	编码格式，设置 Content-Encoding	string	否
ContentLanguage	内容语言，设置 Content-Language	string	否
Expires	设置 Expires	string	否
XCosMetaXXX	用户自定义的对象元信息，必须以 <code>x-cos-meta-</code> 开头，否则会被忽略	*http.Header	否
XCosStorageClass	存储类型，高性能桶仅支持默认值（ <code>RAPID</code> ）	string	否

返回结果说明

返回的响应头可通过 `Response.Header` 获取，例如 ETag：

```
resp, err := c.Object.Put(context.Background(), name, f, nil)
```

```
etag := resp.Header.Get("ETag")
```

参数名称	参数描述	类型
*Response	响应对象，可从中读取响应头（如 ETag）	*cos.Response
ETag	上传对象的 MD5 值，从响应头读取	string
error	错误信息，成功时为 nil	error

使用案例

```
package main

import (
    "context"
    "log"
    "net/http"
    "net/url"
    "os"
    "strings"
    "time"

    "github.com/tencentyun/cos-go-sdk-v5"
)

func main() {
    u, _ := url.Parse("https://examplebucket-x--1250000000.cosrapid.ap-
guangzhou.myqcloud.com")
    b := &cos.BaseURL{BucketURL: u}
    base := &cos.AuthorizationTransport{
        SecretID:  os.Getenv("SECRETID"),
        SecretKey: os.Getenv("SECRETKEY"),
    }
    c := cos.NewClient(b, &http.Client{
        Transport: cos.NewRapidCredentialTransport(base),
        Timeout:   30 * time.Second,
    })

    ctx := context.Background()
```

```
name := "exampleobject.txt"
content := strings.NewReader("hello rapid bucket")

opt := &cos.ObjectPutOptions{
    ObjectPutHeaderOptions: &cos.ObjectPutHeaderOptions{
        ContentType: "text/plain",
    },
}

_, err := c.Object.Put(ctx, name, content, opt)
if err != nil {
    log.Fatalf("PutObject failed: %v", err)
}

log.Println("PutObject OK")
}
```

注意事项

- 上传本地文件时，可直接使用 `PutFromFile`，或用 `os.Open` 打开文件并将 `*os.File` 作为 `io.Reader` 传入 `Put`。
- 大文件建议使用分块上传，详见 [初始化分块上传](#)。
- 对象键不应以 `/` 开头，也不应为空。

错误处理方式请参见 [异常处理](#)。

下载对象

最近更新时间：2026-10-21 18:34:05

功能说明

下载一个对象（文件/对象）至本地（GET Object）。对高性能桶发起下载请求时，SDK 通过 `RapidCredentialTransport` 自动完成会话鉴权。

方法原型

```
func (s *ObjectService) Get(ctx context.Context, name string, opt
*ObjectGetOptions) (*Response, error)
func (s *ObjectService) GetToFile(ctx context.Context, name string,
localpath string, opt *ObjectGetOptions) (*Response, error)
```

参数说明

```
type ObjectGetOptions struct {
    ResponseContentType      string
    ResponseContentLanguage  string
    ResponseExpires          string
    ResponseCacheControl     string
    ResponseContentDisposition string
    ResponseContentEncoding  string
    Range                    string
}
```

参数名称	参数描述	类型	是否必选
ctx	请求上下文，用于控制超时与取消	context.Context	是
name	对象键（Object Key），即对象在桶中的完整路径	string	是
opt	下载选项，可为 nil	*ObjectGetOptions	否
Range	指定下载范围，如 <code>bytes=0-1023</code> ，用于分段下载	string	否

ResponseContentType	指定响应中返回的 Content-Type 头部	string	否
ResponseContentDisposition	指定响应中返回的 Content-Disposition 头部	string	否
ResponseContentEncoding	指定响应中返回的 Content-Encoding 头部	string	否
ResponseContentLanguage	指定响应中返回的 Content-Language 头部	string	否
ResponseCacheControl	指定响应中返回的 Cache-Control 头部	string	否
ResponseExpires	指定响应中返回的 Expires 头部	string	否

返回结果说明

参数名称	参数描述	类型
*Response	响应对象， <code>resp.Body</code> 为对象内容（ <code>io.ReadCloser</code> ）	*cos.Response
error	错误信息，成功时为 nil	error

使用案例

```
package main

import (
    "context"
    "io"
    "log"
    "net/http"
    "net/url"
    "os"
    "time"

    "github.com/tencentyun/cos-go-sdk-v5"
)

func main() {
```

```
u, _ := url.Parse("https://examplebucket-x--1250000000.cosrapid.ap-  
guangzhou.myqcloud.com")  
  
b := &cos.BaseURL{BucketURL: u}  
base := &cos.AuthorizationTransport{  
    SecretID: os.Getenv("SECRETID"),  
    SecretKey: os.Getenv("SECRETKEY"),  
}  
  
c := cos.NewClient(b, &http.Client{  
    Transport: cos.NewRapidCredentialTransport(base),  
    Timeout:   30 * time.Second,  
})  
  
ctx := context.Background()  
name := "exampleobject.txt"  
  
resp, err := c.Object.Get(ctx, name, nil)  
if err != nil {  
    log.Fatalf("GetObject failed: %v", err)  
}  
defer resp.Body.Close()  
  
data, err := io.ReadAll(resp.Body)  
if err != nil {  
    log.Fatalf("read object body failed: %v", err)  
}  
log.Printf("GetObject OK, size: %d", len(data))  
}
```

注意事项

- `resp.Body` 是 `io.ReadCloser`，使用完毕后必须调用 `Close` 释放连接，建议通过 `io.ReadAll` 读取全部内容。
- 若只需将对象直接落盘到本地文件，可使用 `GetToFile`。
- 通过 `Range` 字段可实现分段下载，用于断点续传或部分读取场景。
- 对象键不应以 `/` 开头，也不应为空。

错误处理方式请参见 [异常处理](#)。

查询对象元数据

最近更新时间：2026-10-21 18:34:05

功能说明

查询高性能桶中对象的元数据，不返回对象内容（HEAD Object）。元数据通过响应头部返回。对高性能桶发起请求时，SDK 通过 `RapidCredentialTransport` 自动完成会话鉴权。

方法原型

```
func (s *ObjectService) Head(ctx context.Context, name string, opt
*ObjectHeadOptions) (*Response, error)
```

参数说明

参数名称	参数描述	类型	是否必选
ctx	请求上下文，用于控制超时与取消	context.Context	是
name	对象键（Object Key），即对象在桶中的完整路径	string	是
opt	查询选项，高性能桶无额外可选项，传 nil 即可	*ObjectHeadOptions	否

返回结果说明

对象元数据从 `resp.Header` 中读取，常见头部：

头部	参数描述	类型
Content-Length	对象内容长度（字节）	string
Content-Type	对象的 MIME 类型	string
ETag	对象的 ETag	string
Last-Modified	对象的最后修改时间	string
x-cos-meta-*	用户自定义元数据	string

参数名称	参数描述	类型
*Response	响应对象，元数据通过 <code>resp.Header</code> 返回	*cos.Response

error	错误信息，成功时为 nil	error
-------	---------------	-------

使用案例

```
package main

import (
    "context"
    "log"
    "net/http"
    "net/url"
    "os"
    "time"

    "github.com/tencentyun/cos-go-sdk-v5"
)

func main() {
    u, _ := url.Parse("https://examplebucket-x--1250000000.cosrapid.ap-
guangzhou.myqcloud.com")
    b := &cos.BaseURL{BucketURL: u}
    base := &cos.AuthorizationTransport{
        SecretID: os.Getenv("SECRETID"),
        SecretKey: os.Getenv("SECRETKEY"),
    }
    c := cos.NewClient(b, &http.Client{
        Transport: cos.NewRapidCredentialTransport(base),
        Timeout:   30 * time.Second,
    })

    ctx := context.Background()
    name := "exampleobject.txt"

    resp, err := c.Object.Head(ctx, name, nil)
    if err != nil {
        log.Fatalf("HeadObject failed: %v", err)
    }
    log.Printf("Content-Length: %s", resp.Header.Get("Content-Length"))
    log.Printf("ETag: %s", resp.Header.Get("ETag"))
}
```

```
}
```

注意事项

- Head 请求不返回对象内容，所有元数据均从 `resp.Header` 中读取。
- 若仅需判断对象是否存在，可使用 `IsExist`：`func (s *ObjectService) IsExist(ctx context.Context, name string) (bool, error)`，它会屏蔽 `NotFound` 错误并直接返回布尔结果。
- 对象键不应以 `/` 开头，也不应为空。

错误处理方式请参见 [异常处理](#)。

删除对象

最近更新时间：2026-10-21 18:34:06

功能说明

删除高性能桶中的单个对象（DELETE Object）。对高性能桶发起删除请求时，SDK 通过 `RapidCredential Transport` 自动完成会话鉴权。

方法原型

```
func (s *ObjectService) Delete(ctx context.Context, name string, opt
... *ObjectDeleteOptions) (*Response, error)
```

参数说明

参数名称	参数描述	类型	是否必选
ctx	请求上下文，用于控制超时与取消	context.Context	是
name	对象键（Object Key），即待删除对象在桶中的完整路径	string	是
opt	删除选项，高性能桶无额外可选项，可不传	... *ObjectDeleteOptions	否

返回结果说明

参数名称	参数描述	类型
*Response	响应对象，删除成功返回 204 No Content	*cos.Response
error	错误信息，成功时为 nil	error

使用案例

```
package main

import (
    "context"
    "log"
```

```
"net/http"
"net/url"
"os"
"time"

"github.com/tencentyun/cos-go-sdk-v5"
)

func main() {
    u, _ := url.Parse("https://examplebucket-x--1250000000.cosrapid.ap-
guangzhou.myqcloud.com")
    b := &cos.BaseURL{BucketURL: u}
    base := &cos.AuthorizationTransport{
        SecretID: os.Getenv("SECRETID"),
        SecretKey: os.Getenv("SECRETKEY"),
    }
    c := cos.NewClient(b, &http.Client{
        Transport: cos.NewRapidCredentialTransport(base),
        Timeout:   30 * time.Second,
    })

    ctx := context.Background()
    name := "exampleobject.txt"

    _, err := c.Object.Delete(ctx, name)
    if err != nil {
        log.Fatalf("DeleteObject failed: %v", err)
    }
    log.Println("DeleteObject OK")
}
```

注意事项

- 删除操作具有幂等性，删除不存在的对象同样返回成功（204）。
- 对象键不能为空，也不能为 `/`，否则会直接返回错误。
- 如需一次删除多个对象，请使用 [批量删除对象](#) 以减少请求次数。

错误处理方式请参见 [异常处理](#)。

批量删除对象

最近更新时间：2026-10-21 18:34:06

功能说明

在一次请求中批量删除高性能桶中的多个对象（DELETE Multiple Objects），单次最多支持 1000 个对象。对高性能桶发起请求时，SDK 通过 `RapidCredentialTransport` 自动完成会话鉴权。

方法原型

```
func (s *ObjectService) DeleteMulti(ctx context.Context, opt
*ObjectDeleteMultiOptions) (*ObjectDeleteMultiResult, *Response, error)
```

参数说明

```
type ObjectDeleteMultiOptions struct {
    Quiet    bool
    Objects []Object
}

type Object struct {
    Key string
}
```

参数名称	参数描述	类型	是否必选
ctx	请求上下文，用于控制超时与取消	context.Context	是
opt	批量删除选项，指定待删除的对象列表	*ObjectDeleteMultiOptions	是
Quiet	结果模式。false（Verbose）返回每个对象的删除结果；true（Quiet）仅返回删除失败的对象	bool	否
Objects	待删除的对象列表，每个 <code>Object</code> 通过 <code>Key</code> 字段指定对象键，数量为1~1000	[]Object	是

返回结果说明

```
type ObjectDeleteMultiResult struct {
    DeletedObjects []Object
    Errors          []struct {
        Key      string
        Code      string
        Message   string
    }
}
```

参数名称	参数描述	类型
DeletedObjects	删除成功的对象列表	[]Object
Errors	删除失败的对象列表，每项含 Key、Code、Message	[]struct
*Response	响应对象	*cos.Response
error	错误信息，成功时为 nil	error

使用案例

```
package main

import (
    "context"
    "log"
    "net/http"
    "net/url"
    "os"
    "time"

    "github.com/tencentyun/cos-go-sdk-v5"
)

func main() {
    u, _ := url.Parse("https://examplebucket-x--1250000000.cosrapid.ap-guangzhou.myqcloud.com")
```

```
b := &cos.BaseURL{BucketURL: u}
base := &cos.AuthorizationTransport{
    SecretID:  os.Getenv("SECRETID"),
    SecretKey: os.Getenv("SECRETKEY"),
}
c := cos.NewClient(b, &http.Client{
    Transport: cos.NewRapidCredentialTransport(base),
    Timeout:   30 * time.Second,
})

ctx := context.Background()
opt := &cos.ObjectDeleteMultiOptions{
    Objects: []cos.Object{
        {Key: "a.txt"},
        {Key: "b.txt"},
    },
}

res, _, err := c.Object.DeleteMulti(ctx, opt)
if err != nil {
    log.Fatalf("DeleteMulti failed: %v", err)
}
log.Printf("DeleteMulti OK, deleted: %d, errors: %d",
len(res.DeletedObjects), len(res.Errors))
}
```

注意事项

- 单次请求删除对象数量为 1~1000，超出请分批调用。
- 即使整体请求返回成功，仍需检查 `res.Errors`，其中记录了删除失败的对象及原因。
- 将 `Quiet` 置为 `true` 可减少返回内容，仅返回删除失败的对象信息。

错误处理方式请参见 [异常处理](#)。

复制对象

最近更新时间：2026-10-21 18:34:06

功能说明

将一个对象复制到高性能桶中的目标路径（PUT Object – Copy），可在复制过程中沿用或替换对象元数据。对高性能桶发起复制请求时，SDK 通过 `RapidCredentialTransport` 自动完成会话鉴权。

方法原型

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

参数说明

```
type ObjectCopyOptions struct {
    *ObjectCopyHeaderOptions
}

type ObjectCopyHeaderOptions struct {
    CacheControl          string
    ContentDisposition    string
    ContentEncoding        string
    ContentType           string
    ContentLanguage       string
    Expires                string
    XCosMetadataDirective string
    // 自定义的 x-cos-meta-* header
    XCosMetaXXX           *http.Header
    XCosStorageClass      string
}
```

参数名称	参数描述	类型	是否必选
ctx	请求上下文，用于控制超时与取消	context.Context	是
name	目标对象键（Object Key）	string	是

sourceURL	源对象地址，不带 <code>http://</code> 或 <code>https://</code> 前缀，形如 <code>examplebucket-x--1250000000.cosrapid.ap-guangzhou.myqcloud.com/srckey</code> ；须与目标为同一个桶	string	是
opt	复制选项，可为 nil	*ObjectCopyOptions	否
XCosMetadataDirective	元数据处理方式， <code>Copy</code> 沿用源对象元数据， <code>Replaced</code> 使用请求中指定的元数据	string	否
ContentType	目标对象的 MIME 类型（配合 <code>Replaced</code> 使用）	string	否
XCosStorageClass	目标对象的存储类型，高性能桶仅支持默认值（ <code>RAPID</code> ）	string	否
XCosMetaXX	用户自定义的对象元信息，必须以 <code>x-cos-meta-</code> 开头（配合 <code>Replaced</code> 使用）	*http.Header	否

返回结果说明

```
type ObjectCopyResult struct {
    ETag          string
    LastModified  string
    CRC64         string
    VersionId     string
}
```

参数名称	参数描述	类型
ETag	目标对象的 ETag	string
LastModified	目标对象的最后修改时间	string
CRC64	目标对象的 CRC64 校验值	string
*Response	响应对象	*cos.Response
error	错误信息，成功时为 nil	error

使用案例

```
package main

import (
    "context"
    "log"
    "net/http"
    "net/url"
    "os"
    "time"

    "github.com/tencentyun/cos-go-sdk-v5"
)

func main() {
    u, _ := url.Parse("https://examplebucket-x--1250000000.cosrapid.ap-
guangzhou.myqcloud.com")
    b := &cos.BaseURL{BucketURL: u}
    base := &cos.AuthorizationTransport{
        SecretID: os.Getenv("SECRETID"),
        SecretKey: os.Getenv("SECRETKEY"),
    }
    c := cos.NewClient(b, &http.Client{
        Transport: cos.NewRapidCredentialTransport(base),
        Timeout:   30 * time.Second,
    })

    ctx := context.Background()
    name := "destobject.txt"
    sourceURL := "examplebucket-x--1250000000.cosrapid.ap-
guangzhou.myqcloud.com/srcobject.txt"

    res, _, err := c.Object.Copy(ctx, name, sourceURL, nil)
    if err != nil {
        log.Fatalf("CopyObject failed: %v", err)
    }
    log.Printf("CopyObject OK, ETag: %s", res.ETag)
}
```

注意事项

- 不支持跨桶复制，源对象与目标对象必须位于同一个高性能桶。
- `sourceURL` 不能带 `http://` 或 `https://` 前缀，否则会直接返回错误。
- Copy 即使返回 HTTP 200，响应体中也可能内嵌错误信息，必须检查返回的 `error`，不能仅凭状态码判断成功。
- 超过 5GB 的对象请使用分块复制（请参见 [复制分块](#)）。

错误处理方式请参见 [异常处理](#)。

列举对象

最近更新时间：2026-10-21 18:34:07

功能说明

列举高性能桶中的对象（GET Bucket），支持按前缀过滤、目录分隔与分页翻页。该方法定义在 `BucketService` 上。对高性能桶发起请求时，SDK 通过 `RapidCredentialTransport` 自动完成会话鉴权。

方法原型

```
func (s *BucketService) Get(ctx context.Context, opt *BucketGetOptions)
(*BucketGetResult, *Response, error)
```

参数说明

```
type BucketGetOptions struct {
    Prefix      string
    Delimiter   string
    Marker      string
    MaxKeys     int
    EncodingType string
}
```

参数名称	参数描述	类型	是否必选
ctx	请求上下文，用于控制超时与取消	context.Context	是
opt	列举选项，可为 nil	*BucketGetOptions	否
Prefix	对象键前缀，仅列举以该前缀开头的对象	string	否
Delimiter	分隔符，常用 <code>/</code> 实现按目录层级列举	string	否
Marker	起始标记，从该键之后开始列举，用于翻页	string	否
MaxKeys	单次返回的最大对象数，取值范围1~1000，默认1000	int	否

Encoding Type	返回结果的编码方式，可选值：url	string	否
---------------	-------------------	--------	---

返回结果说明

```
type BucketGetResult struct {
    Name           string
    Prefix          string
    Marker          string
    NextMarker      string
    Delimiter       string
    MaxKeys         int
    IsTruncated     bool
    Contents        []Object
    CommonPrefixes []string
    EncodingType    string
}
```

参数名称	参数描述	类型
Name	桶名称	string
Prefix	本次列举使用的前缀	string
Marker	本次列举的起始标记	string
NextMarker	下一页的起始标记， <code>IsTruncated</code> 为 true 时有效	string
Delimiter	本次列举使用的分隔符	string
MaxKeys	本次列举的最大对象数	int
IsTruncated	结果是否被截断（是否还有下一页）	bool
Contents	对象列表，每个 <code>Object</code> 含 Key、ETag、Size、LastModified 等	[]Object
CommonPrefixes	使用分隔符时聚合出的公共前缀（相当于子目录）	[]string

使用案例

```
package main

import (
    "context"
    "log"
    "net/http"
    "net/url"
    "os"
    "time"

    "github.com/tencentyun/cos-go-sdk-v5"
)

func main() {
    u, _ := url.Parse("https://examplebucket-x--1250000000.cosrapid.ap-  
guangzhou.myqcloud.com")
    b := &cos.BaseURL{BucketURL: u}
    base := &cos.AuthorizationTransport{
        SecretID: os.Getenv("SECRETID"),
        SecretKey: os.Getenv("SECRETKEY"),
    }
    c := cos.NewClient(b, &http.Client{
        Transport: cos.NewRapidCredentialTransport(base),
        Timeout:   30 * time.Second,
    })

    ctx := context.Background()
    marker := ""
    for {
        opt := &cos.BucketGetOptions{
            Prefix: "photos/",
            MaxKeys: 100,
            Marker: marker,
        }
        res, _, err := c.Bucket.Get(ctx, opt)
        if err != nil {
            log.Fatalf("ListObjects failed: %v", err)
        }
        for _, obj := range res.Contents {
```

```
        log.Printf("Key: %s, Size: %d", obj.Key, obj.Size)
    }
    if !res.IsTruncated {
        break
    }
    marker = res.NextMarker
}
log.Println("ListObjects OK")
}
```

注意事项

- 列举对象调用的是 `c.Bucket.Get`，而非 `c.Object` 下的方法。
- 当 `IsTruncated` 为 `true` 时，表示还有更多对象未返回，需将 `NextMarker` 赋给下一次请求的 `Marker` 继续翻页。
- 配合 `Delimiter="/"` 与 `Prefix` 可模拟目录层级浏览，聚合出的子目录通过 `CommonPrefixes` 返回。
- `MaxKeys` 取值范围为1~1000，默认1000。

错误处理方式请参见 [异常处理](#)。

分块上传操作

初始化分块上传

最近更新时间：2026-10-21 18:34:05

功能说明

初始化一次分块上传（Initiate Multipart Upload），成功后返回 `UploadID`，作为后续上传分块与完成上传的唯一标识。分块上传的完整流程为：初始化取得 `UploadID` → 多次上传分块并逐块记录 `PartNumber` 与 `ETag` → 完成分块上传提交。对高性能桶发起请求时，SDK 通过 `RapidCredentialTransport` 自动完成会话鉴权。

方法原型

```
func (s *ObjectService) InitiateMultipartUpload(ctx context.Context,
name string, opt *InitiateMultipartUploadOptions)
(*InitiateMultipartUploadResult, *Response, error)
```

参数说明

```
type InitiateMultipartUploadOptions struct {
    *ObjectPutHeaderOptions
}

type ObjectPutHeaderOptions struct {
    CacheControl      string
    ContentDisposition string
    ContentEncoding   string
    ContentType       string
    ContentLanguage   string
    Expires           string
    // 自定义的 x-cos-meta-* header
    XCosMetaXXX      *http.Header
    XCosStorageClass string
}
```

参数名称	参数描述	类型	是否必选
------	------	----	------

ctx	请求上下文，用于控制超时与取消	context.Context	是
name	对象键（Object Key），即对象在桶中的完整路径	string	是
opt	初始化选项，可为 nil	*InitiateMultipartUploadOptions	否
ContentType	对象的 MIME 类型	string	否
CacheControl	缓存策略，设置 Cache-Control	string	否
XCosMetaXXX	用户自定义的对象元信息，必须以 x-cos-meta- 开头	*http.Header	否
XCosStorageClass	存储类型，高性能桶仅支持默认值（RAPID）	string	否

返回结果说明

```
type InitiateMultipartUploadResult struct {
    Bucket    string
    Key       string
    UploadID  string
}
```

参数名称	参数描述	类型
Bucket	分块上传所在的桶名	string
Key	对象键	string
UploadID	本次分块上传的标识，后续上传分块、完成或终止上传均需携带	string

使用案例

```
package main

import (
    "context"
```

```
"log"
"net/http"
"net/url"
"os"
"time"

"github.com/tencentyun/cos-go-sdk-v5"
)

func main() {
    u, _ := url.Parse("https://examplebucket-x--1250000000.cosrapid.ap-
guangzhou.myqcloud.com")
    b := &cos.BaseURL{BucketURL: u}
    base := &cos.AuthorizationTransport{
        SecretID: os.Getenv("SECRETID"),
        SecretKey: os.Getenv("SECRETKEY"),
    }
    c := cos.NewClient(b, &http.Client{
        Transport: cos.NewRapidCredentialTransport(base),
        Timeout:   30 * time.Second,
    })

    ctx := context.Background()
    name := "exampleobject.txt"

    opt := &cos.InitiateMultipartUploadOptions{
        ObjectPutHeaderOptions: &cos.ObjectPutHeaderOptions{
            ContentType: "text/plain",
        },
    }
    res, _, err := c.Object.InitiateMultipartUpload(ctx, name, opt)
    if err != nil {
        log.Fatalf("InitiateMultipartUpload failed: %v", err)
    }
    log.Printf("UploadID: %s\n", res.UploadID)
}
```

注意事项

- 返回的 `UploadID` 需妥善保存，后续 [上传分块](#)、[完成分块上传](#)、[终止分块上传](#) 均需携带。

- 已初始化但未完成的分块上传会占用存储空间并产生存储费用，请及时 [完成分块上传](#) 或 [终止分块上传](#)。
- `opt` 可为 `nil`，此时按对象类型的默认头部初始化。

错误处理方式请参见 [异常处理](#)。

上传分块

最近更新时间：2026-10-21 18:34:05

功能说明

在初始化分块上传后上传单个分块（Upload Part）。每个分块由 `partNumber` 标识（从 1 开始，支持乱序上传），上传成功后需从响应头读取该块的 `ETag`，并将 `partNumber` 与 `ETag` 记录下来，供后续 [完成分块上传](#) 使用。对高性能桶发起请求时，SDK 通过 `RapidCredentialTransport` 自动完成会话鉴权。

方法原型

```
func (s *ObjectService) UploadPart(ctx context.Context, name, uploadID
string, partNumber int, r io.Reader, opt *ObjectUploadPartOptions)
(*Response, error)
```

参数说明

```
type ObjectUploadPartOptions struct {
    ContentLength int64
    ContentMD5    string
}
```

参数名称	参数描述	类型	是否必选
ctx	请求上下文，用于控制超时与取消	context.Context	是
name	对象键（Object Key）	string	是
uploadID	初始化分块上传返回的 UploadID	string	是
partNumber	分块编号，从 1 开始，支持乱序上传	int	是
r	待上传分块的内容	io.Reader	是
opt	上传选项，可为 nil	*ObjectUploadPartOptions	否

ContentLength	分块内容长度。当 r 不是 bytes.Buffer / bytes.Reader / strings.Reader 时必须指定	int64	否
ContentMD5	分块内容的 MD5 校验值（Base64 编码），用于完整性校验	string	否

返回结果说明

参数名称	参数描述	类型
*Response	响应对象，需从中读取该块的 ETag: resp.Header.Get("ETag")	*cos.Response
error	错误信息，成功时为 nil	error

使用案例

```
package main

import (
    "context"
    "fmt"
    "log"
    "net/http"
    "net/url"
    "os"
    "strings"
    "time"

    "github.com/tencentyun/cos-go-sdk-v5"
)

func main() {
    u, _ := url.Parse("https://examplebucket-x--1250000000.cosrapid.ap-guangzhou.myqcloud.com")
    b := &cos.BaseURL{BucketURL: u}
    base := &cos.AuthorizationTransport{
        SecretID: os.Getenv("SECRETID"),
        SecretKey: os.Getenv("SECRETKEY"),
    }
```

```
c := cos.NewClient(b, &http.Client{
    Transport: cos.NewRapidCredentialTransport(base),
    Timeout:   30 * time.Second,
})

ctx := context.Background()
name := "exampleobject.txt"

// 初始化分块上传, 取得 UploadID
initRes, _, err := c.Object.InitiateMultipartUpload(ctx, name, nil)
if err != nil {
    log.Fatalf("InitiateMultipartUpload failed: %v", err)
}
uploadID := initRes.UploadID

// 上传第 1 个分块, 并记录 partNumber 与 ETag
partNumber := 1
content := strings.NewReader("this is the content of part 1")
resp, err := c.Object.UploadPart(ctx, name, uploadID, partNumber,
content, nil)
if err != nil {
    log.Fatalf("UploadPart failed: %v", err)
}
etag := resp.Header.Get("ETag")
fmt.Printf("part %d uploaded, ETag: %s\n", partNumber, etag)
}
```

注意事项

- 上传成功后必须从 `resp.Header.Get("ETag")` 读取分块 ETag, 并连同 `partNumber` 一起记录, **完成分块上传** 时需按此校验。
- 除最后一块外, 每个分块建议不小于 1MB; 分块过小在完成上传时可能返回 `EntityTooSmall`。
- 当 `r` 不是 `bytes.Buffer` / `bytes.Reader` / `strings.Reader` 时 (例如自定义 Reader), 必须通过 `opt.ContentLength` 显式指定分块长度。
- 相同 `uploadID` 与 `partNumber` 重复上传时, 后上传的分块会覆盖先前的分块。

错误处理方式请参见 [异常处理](#)。

复制分块

最近更新时间：2026-10-21 18:34:05

功能说明

将已有对象（或其指定字节范围）作为一个分块复制到当前分块上传中（Upload Part – Copy），常用于大对象的分块复制、拼接场景。与 [上传分块](#) 类似，每个分块由 `partNumber` 标识，复制成功后需记录返回的 `ETag`，供 [完成分块上传](#) 使用。对高性能桶发起请求时，SDK 通过 `RapidCredentialTransport` 自动完成会话鉴权。

方法原型

```
func (s *ObjectService) CopyPart(ctx context.Context, name, uploadID
string, partNumber int, sourceURL string, opt *ObjectCopyPartOptions)
(*CopyPartResult, *Response, error)
```

参数说明

```
type ObjectCopyPartOptions struct {
    XCosCopySource      string
    XCosCopySourceRange string
}
```

参数名称	参数描述	类型	是否必选
ctx	请求上下文，用于控制超时与取消	context.Context	是
name	目标对象键（Object Key）	string	是
uploadID	初始化分块上传返回的 UploadID	string	是
partNumber	分块编号，从 1 开始	int	是
sourceURL	复制源地址，格式为 <code><bucket>-x--<appid>.cosrapid.<region>.myqcloud.com/<srckey></code> ，不带协议前缀；须与目标为同一个桶	string	是
opt	复制选项，可为 nil	*ObjectCopyP	否

		artOptions	
XCosCopySourceRange	复制源的字节范围，格式为 <code>bytes=<start>--<end></code> ，如 <code>bytes=0-1048575</code>	string	否

返回结果说明

```
type CopyPartResult struct {
    ETag          string
    LastModified string
}
```

参数名称	参数描述	类型
ETag	复制生成分块的 ETag，需记录供完成分块上传使用	string
LastModified	分块的最后修改时间	string

使用案例

```
package main

import (
    "context"
    "fmt"
    "log"
    "net/http"
    "net/url"
    "os"
    "time"

    "github.com/tencentyun/cos-go-sdk-v5"
)

func main() {
    u, _ := url.Parse("https://examplebucket-x--1250000000.cosrapid.ap-guangzhou.myqcloud.com")
    b := &cos.BaseURL{BucketURL: u}
    base := &cos.AuthorizationTransport{
        SecretID: os.Getenv("SECRETID"),
```

```
    SecretKey: os.Getenv("SECRETKEY"),
}

c := cos.NewClient(b, &http.Client{
    Transport: cos.NewRapidCredentialTransport(base),
    Timeout:   30 * time.Second,
})

ctx := context.Background()
name := "exampleobject.txt"

// 初始化分块上传, 取得 UploadID
initRes, _, err := c.Object.InitiateMultipartUpload(ctx, name, nil)
if err != nil {
    log.Fatalf("InitiateMultipartUpload failed: %v", err)
}
uploadID := initRes.UploadID

// 复制源对象的前 1MB 作为第 1 个分块
partNumber := 1
sourceURL := "examplebucket-x--1250000000.cosrapid.ap-
guangzhou.myqcloud.com/srckey"
opt := &cos.ObjectCopyPartOptions{
    XCosCopySourceRange: "bytes=0-1048575",
}
res, _, err := c.Object.CopyPart(ctx, name, uploadID, partNumber,
sourceURL, opt)
if err != nil {
    log.Fatalf("CopyPart failed: %v", err)
}
fmt.Printf("part %d copied, ETag: %s\n", partNumber, res.ETag)
}
```

注意事项

- 不支持跨桶复制，复制源与目标必须位于同一个高性能桶。
- `sourceURL` 不能带 `http://` 或 `https://` 前缀，否则会返回格式错误。
- `ObjectCopyPartOptions.XCosCopySource` 无需设置，SDK 会根据 `sourceURL` 自动填充该字段，手动设置的值会被覆盖。
- 通过 `XCosCopySourceRange` 可只复制源对象的指定字节范围，实现大对象的分段复制拼接。

- 复制成功返回的 `ETag` 需连同 `partNumber` 一起记录，[完成分块上传](#) 时需按此校验。
- 相同 `uploadID` 与 `partNumber` 重复复制时，后复制的分块会覆盖先前的分块。

错误处理方式请参见 [异常处理](#)。

完成分块上传

最近更新时间：2026-10-21 18:34:06

功能说明

完成整个分块上传（Complete Multipart Upload）。当所有分块上传完毕后，需按分块编号升序提交每个分块的 `PartNumber` 与 `ETag`，服务端据此将分块合并为完整对象。对高性能桶发起请求时，SDK 通过 `RapidCredentialTransport` 自动完成会话鉴权。

方法原型

```
func (s *ObjectService) CompleteMultipartUpload(ctx context.Context,
name, uploadID string, opt *CompleteMultipartUploadOptions)
(*CompleteMultipartUploadResult, *Response, error)
```

参数说明

```
type CompleteMultipartUploadOptions struct {
    Parts []Object
}

type Object struct {
    PartNumber int
    ETag       string
}
```

参数名称	参数描述	类型	是否必选
ctx	请求上下文，用于控制超时与取消	context.Context	是
name	对象键（Object Key）	string	是
uploadID	初始化分块上传返回的 UploadID	string	是
opt	完成选项，包含分块列表	*CompleteMultipartUploadOptions	是

Parts	待合并的分块列表，每项需包含 <code>PartNumber</code> 与 <code>ETag</code> ，且按 <code>PartNumber</code> 升序排列	<code>[]Object</code>	是
-------	---	-----------------------	---

返回结果说明

```
type CompleteMultipartUploadResult struct {
    Location string
    Bucket   string
    Key      string
    ETag     string
}
```

参数名称	参数描述	类型
Location	合并后对象的访问地址	string
Bucket	对象所在的桶名	string
Key	对象键	string
ETag	合并后对象的 ETag	string

使用案例

```
package main

import (
    "context"
    "log"
    "net/http"
    "net/url"
    "os"
    "strings"
    "time"

    "github.com/tencentyun/cos-go-sdk-v5"
)

func main() {
```



```
u, _ := url.Parse("https://examplebucket-x--1250000000.cosrapid.ap-  
guangzhou.myqcloud.com")  
  
b := &cos.BaseURL{BucketURL: u}  
  
base := &cos.AuthorizationTransport{  
    SecretID: os.Getenv("SECRETID"),  
    SecretKey: os.Getenv("SECRETKEY"),  
}  
  
c := cos.NewClient(b, &http.Client{  
    Transport: cos.NewRapidCredentialTransport(base),  
    Timeout:   30 * time.Second,  
})  
  
ctx := context.Background()  
name := "exampleobject.txt"  
  
// 初始化分块上传  
initRes, _, err := c.Object.InitiateMultipartUpload(ctx, name, nil)  
if err != nil {  
    log.Fatalf("InitiateMultipartUpload failed: %v", err)  
}  
uploadID := initRes.UploadID  
  
// 上传分块, 记录每块的 PartNumber 与 ETag  
var parts []cos.Object  
partNumber := 1  
resp, err := c.Object.UploadPart(ctx, name, uploadID, partNumber,  
    strings.NewReader("this is the content of part 1"), nil)  
if err != nil {  
    log.Fatalf("UploadPart failed: %v", err)  
}  
parts = append(parts, cos.Object{  
    PartNumber: partNumber,  
    ETag:       resp.Header.Get("ETag"),  
})  
  
// 按 PartNumber 升序提交完成  
opt := &cos.CompleteMultipartUploadOptions{Parts: parts}  
res, _, err := c.Object.CompleteMultipartUpload(ctx, name, uploadID,  
opt)  
if err != nil {
```

```
log.Fatalf("CompleteMultipartUpload failed: %v", err)
}
log.Printf("complete OK, Location: %s, ETag: %s\n", res.Location,
res.ETag)
}
```

注意事项

- `Parts` 必须按 `PartNumber` 升序排列，否则会返回 `InvalidPartOrder`；分块编号不连续会返回 `InvalidPart`。
- 完成分块上传即使返回 200，响应体中仍可能内嵌错误（例如合并阶段出错），SDK 在返回结果的 `ETag` 为空时会将其视为失败并返回 `error`，务必检查返回的 `error`。
- 除最后一块外，分块小于 1MB 会返回 `EntityTooSmall`；`uploadID` 不存在会返回 `NoSuchUpload`。

错误处理方式请参见 [异常处理](#)。

终止分块上传

最近更新时间：2026-10-21 18:34:06

功能说明

放弃一个未完成的分块上传（Abort Multipart Upload），并删除已上传的所有分块以释放存储空间。对高性能桶发起请求时，SDK 通过 `RapidCredentialTransport` 自动完成会话鉴权。

方法原型

```
func (s *ObjectService) AbortMultipartUpload(ctx context.Context, name,
uploadID string, opt ...*AbortMultipartUploadOptions) (*Response, error)
```

参数说明

参数名称	参数描述	类型	是否必选
ctx	请求上下文，用于控制超时与取消	context.Context	是
name	对象键（Object Key）	string	是
uploadID	初始化分块上传返回的 UploadID	string	是
opt	终止选项，可省略	...*AbortMultipartUploadOptions	否

返回结果说明

参数名称	参数描述	类型
*Response	响应对象，终止成功状态码为 204	*cos.Response
error	错误信息，成功时为 nil	error

使用案例

```
package main

import (
    "context"
```

```
"log"
"net/http"
"net/url"
"os"
"time"

"github.com/tencentyun/cos-go-sdk-v5"
)

func main() {
    u, _ := url.Parse("https://examplebucket-x--1250000000.cosrapid.ap-
guangzhou.myqcloud.com")
    b := &cos.BaseURL{BucketURL: u}
    base := &cos.AuthorizationTransport{
        SecretID: os.Getenv("SECRETID"),
        SecretKey: os.Getenv("SECRETKEY"),
    }
    c := cos.NewClient(b, &http.Client{
        Transport: cos.NewRapidCredentialTransport(base),
        Timeout:   30 * time.Second,
    })

    ctx := context.Background()
    name := "exampleobject.txt"
    uploadID := "exampleUploadId"

    _, err := c.Object.AbortMultipartUpload(ctx, name, uploadID)
    if err != nil {
        log.Fatalf("AbortMultipartUpload failed: %v", err)
    }
    log.Println("AbortMultipartUpload OK")
}
```

注意事项

- 终止后，该 `uploadID` 下已上传的分块会被删除且不可恢复。
- 已初始化但未完成的分块上传会占用存储空间并产生存储费用，不再需要时应及时终止；可通过 [列举分块上传](#) 查询桶内未完成的分块上传任务。

- 若终止时仍有正在上传该 `uploadID` 分块的请求，这些 [上传分块](#) 请求会失败；`uploadID` 不存在会返回 `NoSuchUpload`。

错误处理方式请参见 [异常处理](#)。

列举分块上传

最近更新时间：2026-10-21 18:34:06

功能说明

列出桶内所有已初始化但未完成的分块上传任务（List Multipart Uploads），常用于清理长期未完成、占用存储空间的分块上传。该方法属于 `BucketService`，通过 `c.Bucket.ListMultipartUploads` 调用。对高性能桶发起请求时，SDK 通过 `RapidCredentialTransport` 自动完成会话鉴权。

方法原型

```
func (s *BucketService) ListMultipartUploads(ctx context.Context, opt
*ListMultipartUploadsOptions) (*ListMultipartUploadsResult, *Response,
error)
```

参数说明

```
type ListMultipartUploadsOptions struct {
    Delimiter      string
    EncodingType   string
    Prefix         string
    MaxUploads     int
    KeyMarker      string
    UploadIDMarker string
}
```

参数名称	参数描述	类型	是否必选
ctx	请求上下文，用于控制超时与取消	context.Context	是
opt	列举选项，可为 nil	*ListMultipartUploadsOptions	否
Prefix	仅列出对象键匹配该前缀的分块上传	string	否
Delimiter	分隔符，用于对对象键进行分组	string	否
EncodingType	返回结果的编码方式，可选值：url	string	否

MaxUploads	单次最多返回的分块上传数量，取值范围 1~1000，默认 1000	int	否
KeyMarker	从该对象键之后开始列举，用于翻页	string	否
UploadIDMarker	与 KeyMarker 配合使用，从该 UploadID 之后开始列举	string	否

返回结果说明

```
type ListMultipartUploadsResult struct {  
    Bucket      string  
    KeyMarker   string  
    UploadIDMarker string  
    NextKeyMarker string  
    NextUploadIDMarker string  
    MaxUploads  int  
    IsTruncated bool  
    Uploads     []struct {  
        Key      string  
        UploadID string  
        Initiated string  
    }  
}
```

参数名称	参数描述	类型
Bucket	桶名	string
KeyMarker	本次列举起始的对象键标记	string
UploadIDMarker	本次列举起始的 UploadID 标记	string
NextKeyMarker	下一页列举的对象键标记	string
NextUploadIDMarker	下一页列举的 UploadID 标记	string
MaxUploads	本次列举的最大数量	int
IsTruncated	结果是否被截断，为 true 时需用 Next 标记翻页	bool

Uploads	分块上传任务列表，每项含 <code>Key</code> 、 <code>UploadID</code> 、 <code>Initiate</code> <code>d</code>	<code>[]struct</code> <code>t</code>
---------	---	---

使用案例

```
package main

import (
    "context"
    "fmt"
    "log"
    "net/http"
    "net/url"
    "os"
    "time"

    "github.com/tencentyun/cos-go-sdk-v5"
)

func main() {
    u, _ := url.Parse("https://examplebucket-x--1250000000.cosrapid.ap-
guangzhou.myqcloud.com")
    b := &cos.BaseURL{BucketURL: u}
    base := &cos.AuthorizationTransport{
        SecretID: os.Getenv("SECRETID"),
        SecretKey: os.Getenv("SECRETKEY"),
    }
    c := cos.NewClient(b, &http.Client{
        Transport: cos.NewRapidCredentialTransport(base),
        Timeout:   30 * time.Second,
    })

    ctx := context.Background()
    opt := &cos.ListMultipartUploadsOptions{
        Prefix:      "example",
        MaxUploads: 100,
    }
    res, _, err := c.Bucket.ListMultipartUploads(ctx, opt)
    if err != nil {
```



```
log.Fatalf("ListMultipartUploads failed: %v", err)
}
for _, up := range res.Uploads {
    fmt.Printf("Key: %s, UploadID: %s, Initiated: %s\n", up.Key,
up.UploadID, up.Initiated)
}
}
```

注意事项

- 单次列出数量取值范围 1~1000，`IsTruncated` 为 `true` 时需用 `NextKeyMarker` 与 `NextUploadIDMarker` 翻页继续列举。
- 该方法属于 `BucketService`，须通过 `c.Bucket.ListMultipartUploads` 调用，而非 `c.Object`。
- 若要查询某个 `UploadID` 下已上传的具体分块，请使用 [列举已上传分块](#)。

错误处理方式请参见 [异常处理](#)。

列举已上传分块

最近更新时间：2026-10-21 18:34:07

功能说明

列出某个 `UploadID` 下已上传的所有分块（List Parts），常用于 [完成分块上传](#) 前核对分块信息，或在断点续传场景中确认已上传分块。对高性能桶发起请求时，SDK 通过 `RapidCredentialTransport` 自动完成会话鉴权。

方法原型

```
func (s *ObjectService) ListParts(ctx context.Context, name, uploadID
string, opt *ObjectListPartsOptions) (*ObjectListPartsResult, *Response,
error)
```

参数说明

```
type ObjectListPartsOptions struct {
    EncodingType      string
    MaxParts          string
    PartNumberMarker string
}
```

参数名称	参数描述	类型	是否必选
ctx	请求上下文，用于控制超时与取消	context.Context	是
name	对象键（Object Key）	string	是
uploadID	初始化分块上传返回的 UploadID	string	是
opt	列举选项，可为 nil	*ObjectListPart sOptions	否
MaxParts	单次最多返回的分块数量，取值范围 1~1000，默认 1000	string	否
PartNumber Marker	从该分块编号之后开始列举，用于翻页	string	否

EncodingType	返回结果的编码方式，可选值：url	string	否
--------------	-------------------	--------	---

返回结果说明

```
type ObjectListPartsResult struct {
    Bucket          string
    Key             string
    UploadID        string
    MaxParts        string
    PartNumberMarker string
    NextPartNumberMarker string
    IsTruncated     bool
    Parts           []Object
}
```

参数名称	参数描述	类型
Bucket	桶名	string
Key	对象键	string
UploadID	分块上传标识	string
MaxParts	本次列举的最大数量	string
PartNumberMarker	本次列举起始的分块编号标记	string
NextPartNumberMarker	下一页列举的分块编号标记	string
IsTruncated	结果是否被截断，为 true 时需翻页	bool
Parts	已上传分块列表，每项含 <code>PartNumber</code> 、 <code>ETag</code> 、 <code>Size</code> 、 <code>LastModified</code>	[]Object

使用案例

```
package main

import (
```

```
"context"
"fmt"
"log"
"net/http"
"net/url"
"os"
"time"

"github.com/tencentyun/cos-go-sdk-v5"
)

func main() {
    u, _ := url.Parse("https://examplebucket-x--1250000000.cosrapid.ap-
guangzhou.myqcloud.com")
    b := &cos.BaseURL{BucketURL: u}
    base := &cos.AuthorizationTransport{
        SecretID: os.Getenv("SECRETID"),
        SecretKey: os.Getenv("SECRETKEY"),
    }
    c := cos.NewClient(b, &http.Client{
        Transport: cos.NewRapidCredentialTransport(base),
        Timeout:   30 * time.Second,
    })

    ctx := context.Background()
    name := "exampleobject.txt"
    uploadID := "exampleUploadId"

    res, _, err := c.Object.ListParts(ctx, name, uploadID, nil)
    if err != nil {
        log.Fatalf("ListParts failed: %v", err)
    }
    for _, p := range res.Parts {
        fmt.Printf("PartNumber: %d, ETag: %s, Size: %d, LastModified:
%s\n",
            p.PartNumber, p.ETag, p.Size, p.LastModified)
    }
}
```

注意事项

- `IsTruncated` 为 `true` 时表示还有更多分块，需用 `NextPartNumberMarker` 作为下一次的 `PartNumberMarker` 继续翻页。
- 结果中的 `PartNumber` 与 `ETag` 可直接用于组装 [完成分块上传](#) 所需的 `Parts` 列表。
- 若要列举桶内所有未完成的分块上传任务（而非某个 `UploadID` 的分块），请使用 [列举分块上传](#)。

错误处理方式请参见 [异常处理](#)。

异常处理

最近更新时间：2026-10-21 18:34:05

本文档说明高性能桶 Go SDK 的错误处理方式。当 SDK 调用失败时，会返回统一的错误类型 `*cos.ErrorResponse`，可从中获取错误码、错误信息、请求 ID 及 HTTP 状态码，便于定位与处理问题。

错误类型

SDK 调用失败时返回的错误可断言为 `*cos.ErrorResponse`，其结构如下：

字段	类型	描述
Code	string	错误码，如 <code>NoSuchBucket</code> 、 <code>AccessDenied</code>
Message	string	错误的详细描述信息
Resource	string	发生错误的资源
RequestId	string	本次请求的唯一标识，排查问题时可提供给技术支持
TraceID	string	服务端调用链追踪 ID
Response	<code>*http.Response</code>	原始 HTTP 响应，可通过 <code>Response.StatusCode</code> 获取 HTTP 状态码

辅助函数

SDK 提供以下辅助函数便于判断与提取错误：

函数	描述
<code>func IsNotFoundError(e error) bool</code>	判断错误是否为资源不存在（HTTP 404），如对象或桶不存在
<code>func IsCOSError(e error) (*ErrorResponse, bool)</code>	类型断言，若为 COS 错误则取出 <code>*ErrorResponse</code> 并返回 <code>true</code>

示例

判断错误并提取错误码、错误信息、请求 ID 与 HTTP 状态码：

```
package main
```

```
import (
    "context"
    "log"
    "net/http"
    "net/url"
    "os"
    "time"

    "github.com/tencentyun/cos-go-sdk-v5"
)

func main() {
    u, _ := url.Parse("https://examplebucket-x--1250000000.cosrapid.ap-
guangzhou.myqcloud.com")
    b := &cos.BaseURL{BucketURL: u}
    base := &cos.AuthorizationTransport{
        SecretID: os.Getenv("SECRETID"),
        SecretKey: os.Getenv("SECRETKEY"),
    }
    c := cos.NewClient(b, &http.Client{
        Transport: cos.NewRapidCredentialTransport(base),
        Timeout:   30 * time.Second,
    })

    ctx := context.Background()
    _, err := c.Object.Get(ctx, "not-exist-key", nil)
    if err != nil {
        // 使用 IsNotFoundError 判断对象/桶是否存在。
        if cos.IsNotFoundError(err) {
            log.Println("object not found")
            return
        }
        // 取出 *ErrorResponse, 读取详细字段。
        if e, ok := cos.IsCOSError(err); ok {
            log.Printf("Code: %s", e.Code)
            log.Printf("Message: %s", e.Message)
            log.Printf("RequestID: %s", e.RequestID)
            if e.Response != nil {
                log.Printf("StatusCode: %d", e.Response.StatusCode)
            }
        }
    }
}
```

```
        return
    }
    // 其他错误（如网络错误）。
    log.Fatalf("request failed: %v", err)
}
}
```

常见错误码

错误码	HTTP 状态码	说明
NoSuchBucket	404	请求的桶不存在
NoSuchKey	404	请求的对象不存在
NoSuchPolicy	404	桶未设置存储桶策略
AccessDenied	403	无权限访问，请检查 CAM 授权或桶策略
BucketNotEmpty	409	删除桶时桶内仍有对象，需先清空对象

错误码可结合 `RequestID` 一并用于问题排查。