天翼云对象存储

API 参考

天翼云科技有限公司

目 录

1 使用前必读

1.1 概述

欢迎使用对象存储（Object Storage Service，OBS）。OBS 提供海量、安全、高可靠、低成本的数据存储能力，可供用户存储任意类型和大小的数据。适合企业备份/归档、视频点播、视频监控等多种数据存储场景。

您可以使用本文档提供的 API 对 OBS 进行相关操作，如创建、修改、删除桶，上传、下载、删除对象等。支持的全部操作请参见 2 API 概览。

在调用 OBS API 之前，请确保已经充分了解 OBS 相关概念。详细信息请参见 1.4 基本概念。

1.2 调用说明

OBS 提供了 REST（Representational State Transfer）风格 API，支持您通过 HTTP/HTTPS 请求调用，调用方法请参见 3 如何调用 API。

1.3 终端节点

终端节点即调用 API 的**请求地址**，不同服务不同区域的终端节点不同，您可以向企业管理员获取区域和终端节点信息。

1.4 基本概念

使用 OBS API 涉及的常用概念

- 账号

  用户注册时的账号，账号对其所拥有的资源及云服务具有完全的访问权限，可以重置用户密码、分配用户权限等。由于账号是付费主体，为了确保账号安全，建

议您不要直接使用账号进行日常管理工作，而是创建用户并使用他们进行日常管理工作。

- 用户

  由账号在 IAM 中创建的用户，是云服务的使用人员，具有身份凭证（密码和访问密钥）。

  在控制台“我的凭证”下，您可以查看账号 ID 和用户 ID，管理账号或 IAM 用户的访问密钥。

  通常在调用 API 的鉴权过程中，您需要用到账号和 IAM 用户的访问密钥。

- 桶

  桶是用于存放对象的容器，是 OBS 中最高等级的命名空间。每个对象都存放在一个桶中。例如，如果名为“picture.jpg”的对象存放在“photo”桶中，则可使用 URL（http://photo.obs.*region*.*example*.com/picture.jpg）对该对象进行寻址。

- 对象

  对象在 OBS 中是最基本的实体。在一个桶中可以存放多个对象，OBS 系统并不能区分对象的类型。在 OBS 系统中存储的对象是被序列化了的，因此它可能是一个文本文件或者一个视频文件。OBS 支持的数据大小范围可以是 0B 到 48.8TB（包含 0B 和 48.8TB），PutObject 接口上传对象时对象最大为 5GB，超过 5GB 对象需要使用多段方式上传。

- 区域

  指云资源所在的物理位置，同一区域（region）内可用区间内网互通，不同区域间内网不互通。通过在不同地区创建云资源，可以将应用程序设计的更接近特定客户的要求，或满足不同地区的法律或其他要求。

  OBS 中的桶归属于某个区域，这个区域是您在创桶时选定的，桶一旦创建，其归属区域信息不能改变。您可以根据地理位置、成本、满足法规要求等标准来选择桶的区域。可以选择的区域请参见 1.3 终端节点。

2 API 概览

桶基础操作接口

表2-1 桶基础操作接口

接口	说明
5.1.1 获取桶列表	查询自己创建的桶列表。
5.1.2 创建桶	创建一个新桶。可以在创建时添加不同的请求消息头来指定桶的区域、权限控制策略、等信息。
5.1.3 列举桶内对象	获取桶内对象列表。可以在创建时添加不同的请求消息头来获取符合指定前缀、标识符等要求的对象。
5.1.4 获取桶元数据	查询桶元数据是否存在。可以查询桶的区域、OBS 服务版本号、CORS 配置等信息。
5.1.5 获取桶区域位置	获取桶区域位置信息。
5.1.6 删除桶	删除指定的桶。删除之前需要确保桶内无对象。

桶高级配置接口

表2-2 桶高级配置接口

接口	说明
5.2.1 设置桶策略	创建或者修改一个桶的策略。如果桶已经存在一个策略，那么当前请求中的策略将完全覆盖桶中现存的策略。
5.2.2 获取桶策略	获取指定桶的策略信息。
5.2.3 删除桶策略	删除一个指定桶上的策略。

接口	说明
5.2.4 设置桶 ACL	设置一个指定桶的 ACL 信息。通过 ACL 可以控制桶的读写权限。
5.2.5 获取桶 ACL	获取一个指定桶的 ACL 信息。
5.2.6 设置桶日志管理配置	开启或关闭桶的日志管理功能。开启后，桶的每次操作将会产生一条日志，并将多条日志打包成一个日志文件存放在指定的位置。
5.2.7 获取桶日志管理配置	获取指定桶的日志管理配置信息。
5.2.8 设置桶的生命周期配置	指定规则来实现定时删除桶中对象。
5.2.9 获取桶的生命周期配置	获取指定桶已配置的生命周期规则。
5.2.10 删除桶的生命周期配置	删除指定桶的生命周期配置信息。
5.2.11 设置桶的多版本状态	开启或暂停桶的多版本功能。开启后，可以检索和还原各个版本的对象，在意外操作或应用程序故障时快速恢复数据。
5.2.12 获取桶的多版本状态	获取指定桶的多版本功能状态。
5.2.13 设置桶配额	设置桶的空间配额，用以限制桶的最大存储容量。
5.2.14 获取桶配额	获取桶的空间配额。
5.2.15 获取桶存量信息	获取桶中的对象个数及对象占用空间。

对象操作接口

表2-3 对象操作接口

接口	说明
5.3.1 PUT 上传	上传简单对象到指定的桶。
5.3.2 POST 上传	基于表单上传对象到指定的桶。
5.3.3 复制对象	为 OBS 上已经存在的对象创建一个副本。
5.3.4 获取对象内容	下载对象。
5.3.5 获取对象元数据	获取对象的元数据信息。包括对象的过期时间、版本号、CORS 配置等信息。
5.3.6 删除对象	删除指定的对象。也可以携带 versionId 删除指定版本的对象。
5.3.7 批量删除对象	将一个桶内的一部分对象一次性删除，删除后不

接口	说明
	可恢复。
5.3.8 设置对象 ACL	设置一个指定对象的 ACL 信息。通过 ACL 可以控制对象的读写权限。
5.3.9 获取对象 ACL	获取一个指定对象的 ACL 信息。
5.3.10 修改对象元数据	添加、修改或删除桶中已经上传的对象的元数据。

多段操作接口

表2-4 多段操作接口

接口	说明
5.4.1 列举桶中已初始化多段任务	查询一个桶中所有的初始化后还未合并以及未取消的多段上传任务。
5.4.2 初始化上传段任务	使用多段上传特性时，必须首先调用此接口初始化上传段任务，获取全局唯一的多段上传任务号，用于后续的上传段、合并段、列举段等操作。
5.4.3 上传段	为特定的任务上传段。
5.4.4 拷贝段	将已上传对象的一部分或全部拷贝为段。
5.4.5 列举已上传的段	查询一个任务所属的所有段信息。
5.4.6 合并段	将指定的段合并成一个完整的对象。
5.4.7 取消多段上传任务	取消一个多段上传的任务。

3 如何调用 API

3.1 构造请求

本节介绍 REST API 请求的组成。

请求 URI

OBS 根据桶和对象及带的资源参数来确定具体的 URI，当需要进行资源操作时，可以使用这个 URI 地址。

URI 的一般格式为（方括号内为可选项）：

protocol://[bucket.]domain[:port][/object][?param]

表3-1 URI 中的参数

参数	描述	是否必选
protocol	请求使用的协议类型，如 HTTP、HTTPs。HTTPs 表示通过安全的 HTTPs 访问该资源，OBS 支持 HTTP，HTTPs 两种传输协议。	必选
bucket	请求使用的桶资源路径，在整个系统中唯一标识一个桶。	可选
domain	存放资源的服务器的域名或 IP 地址。	必选
port	请求使用的端口号。根据软件服务器的部署不同而不同。缺省时使用默认端口，各种传输协议都有默认的端口号，如 HTTP 的默认端口为 80，HTTPs 的默认端口为 443。 OBS 对象存储服务的 http 方式访问端口为 80，HTTPs 方式访问端口为 443。	可选
object	请求使用的对象资源路径。	可选
param	请求使用的桶和对象的具体资源，缺省默认为请求桶或对象自身资源。	可选

须知

除获取桶列表之外的所有接口，都应当包含桶名。OBS 基于 DNS 解析性能和可靠性的考虑，要求凡是携带桶名的请求，在构造 URL 的时候都必须将桶名放在 domain 前面，形成三级域名形式，又称为虚拟主机访问域名。

例如，如果您有一个位于 a1 区域的名为 test-bucket 的桶，期望访问桶中一个名为 test-object 对象的 acl，正确的访问 URL 为 https://test-bucket.obs.a1.example.com/test-object?acl

请求方法

HTTP 方法（也称为操作或动词），它告诉服务你正在请求什么类型的操作。

表3-2 对象存储支持的 REST 请求方法

方法	说明
GET	请求服务器返回指定资源，如获取桶列表、下载对象等。
PUT	请求服务器更新指定资源，如创建桶、上传对象等。
POST	请求服务器新增资源或执行特殊操作，如初始化上传段任务、合并段等。
DELETE	请求服务器删除指定资源，如删除对象等。
HEAD	请求服务器返回指定资源的概要，如获取对象元数据等。
OPTIONS	请求服务器检查是否具有某个资源的操作权限，需要桶配置 CORS。

请求消息头

可选的附加请求头字段，如指定的 URI 和 HTTP 方法所要求的字段。详细的公共请求消息头字段请参见表 3-3。

表3-3 公共请求消息头

消息头名称	描述	是否必选
Authorization	请求消息中可带的签名信息。 类型：字符串。 默认值：无。 条件：匿名请求不需要带，其他请求必选。	有条件必选
Content-Length	RFC 2616 中定义的消息（不包含消息头）长度。 类型：字符串。 默认值：无。	有条件必选

消息头名称	描述	是否必选
	条件：PUT 操作和加载 XML 的操作必须带。	
Content-Type	资源内容的类型，例如： text/plain。 类型：字符串。 默认值：无。	否
Date	请求发起端的日期和时间。 类型：字符串。 默认值：无。 条件：如果是匿名请求或者消息头中带了 x-obs-date 字段，则可以不带该字段，其他情况下必选。	有条件必选
Host	表明主机地址。如 bucketname.obs.*region*.example.com。 类型：字符串。 默认值：无。	是

请求消息体（可选）

请求消息体通常以结构化格式（如 JSON 或 XML）发出，与请求消息头中 Content-type 对应，传递除请求消息头之外的内容。若请求消息体中参数支持中文，则中文字符必须为 UTF-8 编码。

每个接口的请求消息体内容不同，也并不是每个接口都需要有请求消息体（或者说消息体为空），GET、DELETE 操作类型的接口就不需要消息体，消息体具体内容需要根据具体接口而定。

发起请求

共有两种方式可以基于已构建好的请求消息发起请求，分别为：

- cURL

  cURL 是一个命令行工具，用来执行各种 URL 操作和信息传输。cURL 充当的是 HTTP 客户端，可以发送 HTTP 请求给服务端，并接收响应消息。cURL 适用于接口调试。关于 cURL 详细信息请参见 https://curl.haxx.se/。由于 cURL 无法计算签名，使用 cURL 时仅支持访问匿名的公共 OBS 资源。

- 编码

  通过编码调用接口，组装请求消息，并发送处理请求消息。

3.2 认证鉴权

3.2.1 用户签名验证

OBS 通过 AK/SK 对请求进行签名，在向 OBS 发送请求时，客户端发送的每个消息头需要包含由 SK、请求时间、请求类型等信息生成的签名信息。

- AK(Access Key ID)：访问密钥 ID。与私有访问密钥关联的唯一标识符；访问密钥 ID 和私有访问密钥一起使用，对请求进行加密签名。格式例如：HCY8BGCN1YM5ZWYOK1MH
- SK(Secret Access Key)：与访问密钥 ID 结合使用的密钥，对请求进行加密签名，可标识发送方，并防止请求被修改。格式例如：9zYwf1uabSQY0JTnFqbUqG7vcfqYBaTdXde2GUcq

用户可以在 IAM 服务中获取 AK 和 SK，获取的方法请参见 6.3 获取访问密钥（AK/SK）。

OBS 根据应用场景，提供了 3.2.2 Header 中携带签名、3.2.3 URL 中携带签名和 3.2.4 基于浏览器上传的表单中携带签名 3 种签名计算方式。

以 Header 中携带签名为例，用户签名验证流程如表 3-4 所示。Header 中携带签名方法的具体参数说明及代码示例，请参见 3.2.2 Header 中携带签名。

表3-4 OBS 签名计算和验证步骤

步骤		示例
签名计算	1. 构造 HTTP 消息	PUT /object HTTP/1.1 Host: bucket.obs.region.example.com Date: Tue, 04 Jun 2019 06:54:59 GMT Content-Type: text/plain Content-Length: 5913
	2. 按照签名规则计算 **StringToSign**	StringToSign = HTTP-Verb + "\n" + Content-MD5 + "\n" + Content-Type + "\n" + Date + "\n" + CanonicalizedHeaders + CanonicalizedResource
	3. 准备 AK 和 SK	AK: ****** SK: ******
	4. 计算签名 **Signature**	Signature = Base64(HMAC-SHA1(**SecretAccessKeyID**, UTF-8-Encoding-Of(**StringToSign**)))
	5. 添加签名头域发送到 OBS 服务	PUT /object HTTP/1.1 Host: bucket.obs.region.example.com Date: Tue, 04 Jun 2019 06:54:59 GMT Content-Type: text/plain Content-Length: 5913 Authorization: OBS **AccessKeyID**:**Signature**

步骤		示例
签名验证	6. 接收 HTTP 消息	PUT /object HTTP/1.1 Host: bucket.obs.region.example.com Date: Tue, 04 Jun 2019 06:54:59 GMT Content-Type: text/plain Content-Length: 5913 Authorization: OBS **AccessKeyID**:**Signature**
	7. 根据请求中的 AK 获取 SK	从头域 Authorization 中取出 AK，去 IAM 取回用户的 SK
	8. 按照签名规则计算 **StringToSign**	StringToSign = HTTP-Verb + "\n" + Content-MD5 + "\n" + Content-Type + "\n" + Date + "\n" + CanonicalizedHeaders + CanonicalizedResource
	9. 计算签名 **Signature**	Signature = Base64(HMAC-SHA1(**SecretAccessKeyID**, UTF-8-Encoding-Of(**StringToSign**)))
	10. 验证签名	验证头域 Authorization 中的 **Signature** 与服务端计算的 **Signature** 是否相等 相等：签名验证通过 不相等：签名验证失败

3.2.2 Header 中携带签名

OBS 的所有 API 接口都可以通过在 header 中携带签名方式来进行身份认证，也是最常用的身份认证方式。

在 Header 中携带签名是指将通过 HTTP 消息中 Authorization header 头域携带签名信息，消息头域的格式为：

```
Authorization: OBS AccessKeyID:signature
```

签名的计算过程如下：

1、构造请求字符串(StringToSign)。

2、对第一步的结果进行 UTF-8 编码。

3、使用 SK 对第二步的结果进行 HMAC-SHA1 签名计算。

4、对第三步的结果进行 Base64 编码，得到签名。

请求字符串(StringToSign)按照如下规则进行构造，各个参数的含义如表 3-5 所示。

```
StringToSign =
   HTTP-Verb + "\n" +
   Content-MD5 + "\n" +
   Content-Type + "\n" +
```

```
Date + "\n" +
CanonicalizedHeaders + CanonicalizedResource
```

表3-5 构造 StringToSign 所需参数说明

参数	描述
HTTP-Verb	指接口操作的方法，对 REST 接口而言，即为 http 请求操作的 VERB，如："PUT"，"GET"，"DELETE"等字符串。
Content-MD5	按照 RFC 1864 标准计算出消息体的 MD5 摘要字符串，即消息体 128-bit MD5 值经过 base64 编码后得到的字符串，可以为空。具体请参见表 3-10 以及表下方的计算方法示例。
Content-Type	内容类型，用于指定 消息类型，例如： text/plain。 当请求中不带该头域时，该参数按照空字符串处理，见表 3-6。
Date	生成请求的时间，该时间格式遵循 RFC 1123； 当有自定义字段 x-obs-date 时，该参数按照空字符串处理；见表 3-10。 如果进行临时授权方式操作（如临时授权方式获取对象内容等操作）时，该参数不需要。
Canonicalized Headers	HTTP 请求头域中的 OBS 请求头字段，即以“x-obs-”作为前辍的头域，如“x-obs-date，x-obs-acl，x-obs-meta-*”。 1. 请求头字段中关键字的所有字符要转为小写，需要添加多个字段时，要将所有字段按照关键字的字典序从小到大进行排序。 2. 在添加请求头字段时，如果有重名的字段，则需要进行合并。如：x-obs-meta-name:name1 和 x-obs-meta-name:name2，则需要先将重名字段的值（这里是 name1 和 name2）以逗号分隔，合并成 x-obs-meta-name:name1,name2。 3. 头域中的请求头字段中的关键字不允许含有非 ASCII 码或不可识别字符；请求头字段中的值也不建议使用非 ASCII 码或不可识别字符，如果一定要使用非 ASCII 码或不可识别字符，需要客户端自行做编解码处理，可以采用 URL 编码或者 Base64 编码，服务端不会做解码处理。 4. 当请求头字段中含有无意义空格或 table 键时，需要摒弃。例如：x-obs-meta-name: name（name 前带有一个无意义空格），需要转换为：x-obs-meta-name:name 5. 每一个请求头字段最后都需要另起新行，见表 3-8
Canonicalized Resource	表示 HTTP 请求所指定的 OBS 资源，构造方式如下： <桶名+对象名>+[子资源 1] + [子资源 2] + ... 1. 桶名和对象名，例如：/bucket/object。如果没有对象名，如列举桶，则为"/bucket/"，如桶名也没有，则为“/”。 2. 如果有子资源，则将子资源添加进来，例如?acl，?logging。 OBS 支持各种子资源，包括：CDNNotifyConfiguration, acl, append, attname, backtosource, cors, customdomain, delete, deletebucket, directcoldaccess, encryption, inventory, length, lifecycle,

参数	描述
	location, logging, metadata, modify, name, notification, orchestration, partNumber, policy, position, quota, rename, replication, requestPayment, response-cache-control, response-content-disposition, response-content-encoding, response-content-language, response-content-type, response-expires, restore, select, storagePolicy, storageinfo, tagging, torrent, truncate, uploadId, uploads, versionId, versioning, versions, website, x-image-process, x-image-save-bucket, x-image-save-object, x-obs-security-token。 3. 如果有多个子资源，在包含这些子资源时，需要首先将这些子资源按照其关键字的字典序从小到大排列，并使用**“&”**拼接。 说明 • 子资源通常是唯一的，不建议请求的 URL 包含多个相同关键字的子资源（例如，key=value1&key=value2），如果存在这种情况，OBS 服务端签名时只会计算第一个子资源且也只有第一个子资源的值会对实际业务产生作用； • 以获取对象（GetObject）接口为例，假设桶名为 bucket-test，对象名为 object-test，对象的版本号为 xxx，获取时需要重写 Content-Type 为 text/plain，那么签名计算出的 CanonicalizedResource 为：/bucket-test/object-test?response-content-type=text/plain&versionId=xxx。

下面的几张表提供了一些生成 StringToSign 的例子。

表3-6 获取对象

请求消息头	StringToSign
GET /object.txt HTTP/1.1 Host: bucket.obs.*region*.example.com Date: Sat, 12 Oct 2015 08:12:38 GMT	GET \n \n \n Sat, 12 Oct 2015 08:12:38 GMT\n /bucket/object.txt

表3-7 使用临时 AK/SK 和 securitytoken 上传对象

请求消息头	StringToSign
PUT /object.txt HTTP/1.1 User-Agent: curl/7.15.5 Host: bucket.obs.*region*.example.com x-obs-date:Tue, 15 Oct 2015 07:20:09 GMT x-obs-security-token: YwkaRTbdY8g7q.... content-type: text/plain Content-Length: 5913339	PUT\n \n text/plain\n \n x-obs-date:Tue, 15 Oct 2015 07:20:09 GMT\n x-obs-security-token:YwkaRTbdY8g7q....\n /bucket/object.txt

表3-8 带请求头字段上传对象

请求消息头	StringToSign
PUT /object.txt HTTP/1.1 User-Agent: curl/7.15.5 Host: bucket.obs.*region*.example.com Date: Mon, 14 Oct 2015 12:08:34 GMT x-obs-acl: public-read content-type: text/plain Content-Length: 5913339	PUT\n \n text/plain\n Mon, 14 Oct 2015 12:08:34 GMT\n x-obs-acl:public-read\n /bucket/object.txt

表3-9 获取对象 ACL

请求消息头	StringToSign
GET /object.txt?acl HTTP/1.1 Host: bucket.obs.*region*.example.com Date: Sat, 12 Oct 2015 08:12:38 GMT	GET \n \n \n Sat, 12 Oct 2015 08:12:38 GMT\n /bucket/object.txt?acl

表3-10 上传对象且携带 Content-MD5 头域

请求消息头	StringToSign
PUT /object.txt HTTP/1.1 Host: bucket.obs.*region*.example.com x-obs-date:Tue, 15 Oct 2015 07:20:09 GMT Content-MD5: I5pU0r4+sgO9Emgl1KMQUg== Content-Length: 5913339	PUT\n I5pU0r4+sgO9Emgl1KMQUg==\n \n \n x-obs-date:Tue, 15 Oct 2015 07:20:09 GMT\n /bucket/object.txt

Java 中 Content-MD5 的计算方法示例

```
import java.security.MessageDigest;
import sun.misc.BASE64Encoder;
import java.io.UnsupportedEncodingException;
import java.security.NoSuchAlgorithmException;

public class Md5{
```

```
    public static void main(String[] args) {
        try {
              String exampleString = "blog";
              MessageDigest messageDigest = MessageDigest.getInstance("MD5");
              BASE64Encoder encoder = new BASE64Encoder();
              String contentMd5 =
encoder.encode(messageDigest.digest(exampleString.getBytes("utf-8")));
              System.out.println("Content-MD5: " + contentMd5);
        } catch (NoSuchAlgorithmException | UnsupportedEncodingException e)
        {
              e.printStackTrace();
        }
    }
}
```

根据请求字符串(StringToSign)和用户 SK 使用如下算法生成 Signature，生成过程使用 HMAC 算法(hash-based authentication code algorithm)。

```
Signature = Base64( HMAC-SHA1( YourSecretAccessKeyID, UTF-8-Encoding-
Of( StringToSign ) ) )
```

例如在某区域创建桶名为 newbucketname2 的私有桶，客户端请求格式为：

```
PUT / HTTP/1.1
Host: newbucketname2.obs.region.example.com
Content-Length: length
Date: Fri, 06 Jul 2018 03:45:51 GMT
x-obs-acl:private
x-obs-storage-class:STANDARD
Authorization: OBS UDSIAMSTUBTEST000254:ydH8ffpcbS6YpeOMcEZfn0wE90c=

<CreateBucketConfiguration xmlns="http://obs.region.example.com/doc/2015-06-30/">
   <Location>region</Location>
</CreateBucketConfiguration>
```

Java 中签名的计算方法

```
import java.io.UnsupportedEncodingException;
import java.net.URLEncoder;
import java.security.InvalidKeyException;
import java.security.NoSuchAlgorithmException;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Base64;
import java.util.Collections;
import java.util.HashMap;
import java.util.List;
import java.util.Locale;
import java.util.Map;
import java.util.TreeMap;

import javax.crypto.Mac;
import javax.crypto.spec.SecretKeySpec;

import org.omg.CosNaming.IstringHelper;
```

```
public class SignDemo {

    private static final String SIGN_SEP = "\n";

    private static final String OBS_PREFIX = "x-obs-";

    private static final String DEFAULT_ENCODING = "UTF-8";

    private static final List<String> SUB_RESOURCES =
Collections.unmodifiableList(Arrays.asList(
             "CDNNotifyConfiguration", "acl", "append", "attname",  "cors",
"delete",
             "deletebucket", "length", "lifecycle", "location", "logging",
             "metadata", "modify", "name", "partNumber", "policy", "position",
"quota",
             "rename", "replication", "response-cache-control", "response-content-
disposition",
             "response-content-encoding", "response-content-language", "response-
content-type", "response-expires",
             "storagePolicy", "storageinfo", "torrent", "truncate",
             "uploadId", "uploads", "versionId", "versioning", "versions",
"website",
             "x-obs-security-token"));

    private String ak;

    private String sk;

     public String urlEncode(String input) throws UnsupportedEncodingException
   {
         return URLEncoder.encode(input, DEFAULT ENCODING)
       .replaceAll("%7E", "~") //for browser
       .replaceAll("%2F", "/");
   }

    private String join(List<?> items, String delimiter)
   {
       StringBuilder sb = new StringBuilder();
       for (int i = 0; i < items.size(); i++)
       {
    String item = items.get(i).toString();
          sb.append(item);
          if (i < items.size() - 1)
          {
             sb.append(delimiter);
          }
       }
       return sb.toString();
   }

    private boolean isValid(String input) {
        return input != null && !input.equals("");
    }
```

```
    public String hamcSha1(String input) throws NoSuchAlgorithmException,
InvalidKeyException, UnsupportedEncodingException {
        SecretKeySpec signingKey = new
SecretKeySpec(this.sk.getBytes(DEFAULT_ENCODING), "HmacSHA1");
        Mac mac = Mac.getInstance("HmacSHA1");
        mac.init(signingKey);
        return
Base64.getEncoder().encodeToString(mac.doFinal(input.getBytes(DEFAULT_ENCODING)));
    }

    private String stringToSign(String httpMethod, Map<String, String[]> headers,
Map<String, String> queries,
            String bucketName, String objectName) throws Exception{
        String contentMd5 = "";
        String contentType = "";
        String date = "";

        TreeMap<String, String> canonicalizedHeaders = new TreeMap<String,
String>();

        String key;
        List<String> temp = new ArrayList<String>();
        for(Map.Entry<String, String[]> entry : headers.entrySet()) {
            key = entry.getKey();
            if(key == null || entry.getValue() == null || entry.getValue().length
== 0) {
                continue;
            }

            key = key.trim().toLowerCase(Locale.ENGLISH);
            if(key.equals("content-md5")) {
                contentMd5 = entry.getValue()[0];
                continue;
            }

            if(key.equals("content-type")) {
                contentType = entry.getValue()[0];
                continue;
            }

            if(key.equals("date")) {
                date = entry.getValue()[0];
                continue;
            }

            if(key.startsWith(OBS PREFIX)) {

                for(String value : entry.getValue()) {
                    if(value != null) {
                        temp.add(value.trim());
                    }
                }
                canonicalizedHeaders.put(key, this.join(temp, ","));
                temp.clear();
            }
```

```
        }

        if(canonicalizedHeaders.containsKey("x-obs-date")) {
            date = "";
        }


        // handle method/content-md5/content-type/date
        StringBuilder stringToSign = new StringBuilder();
        stringToSign.append(httpMethod).append(SIGN_SEP)
            .append(contentMd5).append(SIGN_SEP)
            .append(contentType).append(SIGN_SEP)
            .append(date).append(SIGN_SEP);

        // handle canonicalizedHeaders
        for(Map.Entry<String, String> entry : canonicalizedHeaders.entrySet()) {

    stringToSign.append(entry.getKey()).append(":").append(entry.getValue()).append
(SIGN_SEP);
        }

        // handle CanonicalizedResource
        stringToSign.append("/");
        if(this.isValid(bucketName)) {
            stringToSign.append(bucketName).append("/");
            if(this.isValid(objectName)) {
                stringToSign.append(this.urlEncode(objectName));
            }
        }

        TreeMap<String, String> canonicalizedResource = new TreeMap<String,
String>();
        for(Map.Entry<String, String> entry : queries.entrySet()) {
            key = entry.getKey();
            if(key == null) {
                continue;
            }

            if(SUB RESOURCES.contains(key)) {
                canonicalizedResource.put(key, entry.getValue());
            }
        }

        if(canonicalizedResource.size() > 0) {
            stringToSign.append("?");
           for(Map.Entry<String, String> entry : canonicalizedResource.entrySet())
{
                stringToSign.append(this.urlEncode(entry.getKey()));
                if(this.isValid(entry.getValue())) {

    stringToSign.append("=").append(this.urlEncode(entry.getValue()));
                }
            }
        }
```

```
//      System.out.println(String.format("StringToSign:%s%s", SIGN SEP,
stringToSign.toString()));

        return stringToSign.toString();
    }

    public String headerSignature(String httpMethod, Map<String, String[]> headers,
Map<String, String> queries,
            String bucketName, String objectName) throws Exception {

        //1. stringToSign
        String stringToSign = this.stringToSign(httpMethod, headers, queries,
bucketName, objectName);

        //2. signature
        return String.format("OBS %s:%s", this.ak, this.hamcSha1(stringToSign));
    }


    public String querySignature(String httpMethod, Map<String, String[]> headers,
Map<String, String> queries,
            String bucketName, String objectName, long expires) throws Exception {
        if(headers.containsKey("x-obs-date")) {
            headers.put("x-obs-date", new String[] {String.valueOf(expires)});
        }else {
            headers.put("date", new String[] {String.valueOf(expires)});
        }
        //1. stringToSign
        String stringToSign = this.stringToSign(httpMethod, headers, queries,
bucketName, objectName);

        //2. signature
        return this.urlEncode(this.hamcSha1(stringToSign));
    }

    public static void main(String[] args) throws Exception {

        SignDemo demo = new SignDemo();
        demo.ak = "<your-access-key-id>";
        demo.sk = "<your-securet-key-id>";

        String bucketName = "bucket-test";
        String objectName = "hello.jpg";
        Map<String, String[]> headers = new HashMap<String, String[]>();
        headers.put("date", new String[] {"Sat, 12 Oct 2015 08:12:38 GMT"});
        headers.put("x-obs-acl", new String[] {"public-read"});
        headers.put("x-obs-meta-key1", new String[] {"value1"});
        headers.put("x-obs-meta-key2", new String[] {"value2", "value3"});
        Map<String, String> queries = new HashMap<String, String>();
        queries.put("acl", null);

        System.out.println(demo.headerSignature("PUT", headers, queries,
bucketName, objectName));
    }
```

```
}
```

签名计算的样例结果为（按照执行时间的不同变化）：
ydH8ffpcbS6YpeOMcEZfn0wE90c=

Python 中签名的计算方法

```
import sys
import hashlib
import hmac
import binascii
from datetime import datetime
IS PYTHON2 = sys.version info.major == 2 or sys.version < '3'

yourSecretAccessKeyID = '275hSvB6EEOorBNsMDEfOaICQnilYaPZhXUaSK64'
httpMethod = "PUT"
contentType = "application/xml"
# "date" is the time when the request was actually generated
date = datetime.utcnow().strftime('%a, %d %b %Y %H:%M:%S GMT')
canonicalizedHeaders = "x-obs-acl:private"
CanonicalizedResource = "/newbucketname2"
canonical string = httpMethod + "\n" + "\n" + contentType + "\n" + date + "\n" +
canonicalizedHeaders + CanonicalizedResource
if IS PYTHON2:
    hashed = hmac.new(yourSecretAccessKeyID, canonical string, hashlib.sha1)
    encode canonical = binascii.b2a base64(hashed.digest())[:-1]
else:
    hashed = hmac.new(yourSecretAccessKeyID.encode('UTF-8'),
canonical string.encode('UTF-8'),hashlib.sha1)
    encode canonical = binascii.b2a base64(hashed.digest())[:-1].decode('UTF-8')
print encode_canonical
```

签名计算的样例结果为（按照执行时间的不同变化）：
ydH8ffpcbS6YpeOMcEZfn0wE90c=

3.2.3 URL 中携带签名

URL 中携带签名：OBS 服务支持用户构造一个特定操作的 URL，这个 URL 中会包含用户 AK、签名、有效期、资源等信息，任何拿到这个 URL 的人均可执行这个操作，OBS 服务收到这个请求后认为该请求就是签发 URL 用户自己在执行操作。例如构造一个携带签名信息的下载对象的 URL，拿到相应 URL 的人能下载这个对象，但该 URL 只在 Expires 指定的失效时间内有效。URL 中携带签名主要用于在不提供给其他人 Secret Access Key 的情况下，让其他人能用预签发的 URL 来进行身份认证，并执行预定义的操作。

URL 中携带签名请求的消息格式如下：

```
GET /ObjectKey?AccessKeyId=AccessKeyID&Expires=ExpiresValue&Signature=signature
HTTP/1.1
Host: bucketname.obs.region.example.com
```

URL 中使用临时 AK，SK 和 securitytoken 下载对象消息格式如下：

```
GET /ObjectKey?AccessKeyId=AccessKeyID&Expires=ExpiresValue&Signature=signature&x-obs-security-token=securitytoken HTTP/1.1
Host: bucketname.obs.region.example.com
```

参数具体意义如表 3-11 所示。

表3-11 请求消息参数

参数名称	描述	是否必选
AccessKeyId	签发者的 AK 信息。OBS 根据 AK 确定签发者的身份，并认为 URL 就是签发者在访问。 类型：字符串。	是
Expires	临时授权失效的时间；临时授权失效的时间为 24 小时，但是如果含有了临时的 AK，其授权失效时间最大值也只能为 24 小时；UTC 时间，1970 年 1 月 1 日零时之后的指定的 Expires 时间内有效（以秒为单位）。 类型：字符串。	是
Signature	根据用户 SK、Expires 等参数计算出的签名信息。 类型：字符串。	是
x-obs-security-token	使用临时 AK/SK 鉴权时，临时 AK/SK 和 securitytoken 必须同时使用，请求头中需要添加“x-obs-security-token”字段	否

签名的计算过程如下：

1、构造请求字符串(StringToSign)。

2、对第一步的结果进行 UTF-8 编码。

3、使用 SK 对第二步的结果进行 HMAC-SHA1 签名计算。

4、对第三步的结果进行 Base64 编码。

5、对第四步的结果进行 URL 编码，得到签名。

请求字符串(StringToSign)按照如下规则进行构造，各个参数的含义如表 3-12 所示：

```
StringToSign =
    HTTP-Verb + "\n" +
    Content-MD5 + "\n" +
    Content-Type + "\n" +
    Expires + "\n" +
    CanonicalizedHeaders +  CanonicalizedResource;
```

表3-12 构造 StringToSign 所需参数说明

参数	描述
HTTP-Verb	指接口操作的方法，对 REST 接口而言，即为 http 请求操作的 VERB，如："PUT"，"GET"，"DELETE"等字符串。
Content-MD5	按照 RFC 1864 标准计算出消息体的 MD5 摘要字符串，即消息体 128-bit MD5 值经过 base64 编码后得到的字符串，可以为空。
Content-Type	内容类型，用于指定 消息类型，例如： text/plain。 当请求中不带该头域时，该参数按照空字符串处理。
Expires	临时授权的失效时间，即请求消息参数 Expires 的值 ExpiresValue。
Canonicalized Headers	HTTP 请求头域中的 OBS 请求头字段，即以“x-obs-”作为前辍的头域，如“x-obs-date，x-obs-acl，x-obs-meta-*”。 1. 请求头字段中关键字的所有字符要转为小写，需要添加多个字段时，要将所有字段按照关键字的字典序从小到大进行排序。 2. 在添加请求头字段时，如果有重名的字段，则需要进行合并。如：x-obs-meta-name:name1 和 x-obs-meta-name:name2，则需要先将重名字段的值（这里是 name1 和 name2）以逗号分隔，合并成 x-obs-meta-name:name1,name2。 3. 头域中的请求头字段中的关键字不允许含有非 ASCII 码或不可识别字符；请求头字段中的值也不建议使用非 ASCII 码或不可识别字符，如果一定要使用非 ASCII 码或不可识别字符，需要客户端自行做编解码处理，可以采用 URL 编码或者 Base64 编码，服务端不会做解码处理。 4. 当请求头字段中含有无意义空格或 table 键时，需要摒弃。例如：x-obs-meta-name: name（name 前带有一个无意义空格），需要转换为：x-obs-meta-name:name 5. 每一个请求头字段最后都需要另起新行。
Canonicalized Resource	表示 HTTP 请求所指定的 OBS 资源，构造方式如下： <桶名+对象名>+[子资源]+ [子资源 2] + ... 1. 桶名和对象名，例如：/bucket/object。如果没有对象名，如列举桶，则为"/bucket/"，如桶名也没有，则为“/”。 2. 如果有子资源，则将子资源添加进来，例如?acl，?logging。 OBS 支持各种子资源，包括：CDNNotifyConfiguration, acl, append, attname, backtosource, cors, customdomain, delete, deletebucket, directcoldaccess, encryption, inventory, length, lifecycle, location, logging, metadata, modify, name, notification, orchestration, partNumber, policy, position, quota, rename, replication, requestPayment, response-cache-control, response-content-disposition, response-content-encoding, response-content-language, response-content-type, response-expires, restore, select, storagePolicy, storageinfo, tagging, torrent, truncate, uploadId, uploads, versionId, versioning, versions, website, x-image-process, x-image-save-bucket, x-image-save-object, x-obs-security-token。 3. 如果有多个子资源，在包含这些子资源时，需要首先将这些子资

参数	描述
	源按照其关键字的字典序从小到大排列，并使用“&”拼接。 说明 • 子资源通常是唯一的，不建议请求的 URL 包含多个相同关键字的子资源（例如，key=value1&key=value2），如果存在这种情况，OBS 服务端签名时只会计算第一个子资源且也只有第一个子资源的值会对实际业务产生作用； • 以获取对象（GetObject）接口为例，假设桶名为 bucket-test，对象名为 object-test，对象的版本号为 xxx，获取时需要重写 Content-Type 为 text/plain，那么签名计算出的 CanonicalizedResource 为：/bucket-test/object-test?response-content-type=text/plain&versionId=xxx。

根据请求字符串(StringToSign)和用户 SK 使用如下算法生成 Signature，生成过程使用 HMAC 算法(hash-based authentication code algorithm)。

```
Signature = URL-Encode( Base64( HMAC-SHA1( YourSecretAccessKeyID, UTF-8-Encoding-Of( StringToSign ) ) ) )
```

URL 中的 Signature 计算方法和 Header 中携带的 Authorization 签名计算方法有两处不同：

- URL 中签名在 Base64 编码后还要经过 URL 编码。
- StringToSign 中的 Expires 和原来 Authorization 消息中的消息头 Date 对应。

使用 URL 携带签名方式为浏览器生成预定义的 URL 实例：

表3-13 下载对象在 URL 中携带签名的请求及 StringToSign

请求消息头	StringToSign
GET /objectkey?AccessKeyId=MFyfvK41ba2giqM7Uio6PznpdUKGpownRZlmVmHc&Expires=1532779451&Signature=0Akylf43Bm3mD1bh2rM3dmVp1Bo%3D HTTP/1.1 Host: examplebucket.obs.*region*.example.com	GET \n \n \n 1532779451\n /examplebucket/objectkey

表3-14 在 URL 中使用临时 AK/SK 和 securitytoken 下载对象请求及 StringToSign

请求消息头	StringToSign
GET /objectkey?AccessKeyId=MFyfvK41ba2giqM7Uio6PznpdUKGpownRZlmVmHc&Expires=1532779451&Signature=0Akylf43Bm3mD1bh2rM3dmVp1Bo%3D&x-obs-security-token=YwkaRTbdY8g7q.... HTTP/1.1	GET \n \n \n 1532779451\n /examplebucket/objectkey?x-obs-security-token:YwkaRTbdY8g7q....

请求消息头	StringToSign
Host: examplebucket.obs.*region*.example.com	

根据签名计算规则

```
Signature = URL-Encode( Base64( HMAC-SHA1( YourSecretAccessKeyID, UTF-8-Encoding-Of( StringToSign ) ) ) )
```

计算出签名，然后将 Host 作为 URL 的前缀，可以生成预定义的 URL:

http(s)://examplebucket.obs.*region*.example.com/objectkey?AccessKeyId=AccessKeyID&Expires=1532779451&Signature=0Akylf43Bm3mD1bh2rM3dmVp1Bo%3D

在浏览器中直接输入该地址则可以下载 examplebucket 桶中的 objectkey 对象。这个链接的有效期是 1532779451(Sat Jul 28 20:04:11 CST 2018)。

在 Linux 环境上使用 curl 命令访问注意&字符需要\转义，如下命令将对象 objectkey 下载到 output 文件中：

curl http(s)://examplebucket.obs.*region*.example.com/objectkey?AccessKeyId=AccessKeyID\&Expires=1532779451\&Signature=0Akylf43Bm3mD1bh2rM3dmVp1Bo%3D -X GET -o output

说明

如果想要在浏览器中使用 URL 中携带签名生成的预定义 URL，则计算签名时不要使用只能携带在头域部分的“Content-MD5”、“Content-Type”、“CanonicalizedHeaders”来计算签名。否则浏览器不能携带这些参数，请求发送到服务端之后，会提示签名错误。

Java 中签名的计算方法

```
import java.io.UnsupportedEncodingException;
import java.net.URLEncoder;
import java.security.InvalidKeyException;
import java.security.NoSuchAlgorithmException;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Base64;
import java.util.Collections;
import java.util.HashMap;
import java.util.List;
import java.util.Locale;
import java.util.Map;
import java.util.TreeMap;

import javax.crypto.Mac;
import javax.crypto.spec.SecretKeySpec;

import org.omg.CosNaming.IstringHelper;


public class SignDemo {
```

```
    private static final String SIGN SEP = "\n";

    private static final String OBS_PREFIX = "x-obs-";

    private static final String DEFAULT_ENCODING = "UTF-8";

    private static final List<String> SUB_RESOURCES =
Collections.unmodifiableList(Arrays.asList(
             "CDNNotifyConfiguration", "acl", "append", "attname", "cors",
"delete",
             "deletebucket", "length", "lifecycle", "location", "logging",
             "metadata", "modify", "name", "partNumber", "policy", "position",
"quota",
             "rename", "replication", "response-cache-control", "response-content-
disposition",
             "response-content-encoding", "response-content-language", "response-
content-type", "response-expires",
             "select", "storagePolicy", "storageinfo", "torrent", "truncate",
             "uploadId", "uploads", "versionId", "versioning", "versions",
"website",
              "x-obs-security-token"));

    private String ak;

    private String sk;

     public String urlEncode(String input) throws UnsupportedEncodingException
   {
         return URLEncoder.encode(input, DEFAULT ENCODING)
       .replaceAll("%7E", "~") //for browser
       .replaceAll("%2F", "/");
   }

    private String join(List<?> items, String delimiter)
   {
       StringBuilder sb = new StringBuilder();
       for (int i = 0; i < items.size(); i++)
       {
    String item = items.get(i).toString();
          sb.append(item);
          if (i < items.size() - 1)
          {
             sb.append(delimiter);
          }
       }
       return sb.toString();
   }

    private boolean isValid(String input) {
        return input != null && !input.equals("");
    }

    public String hamcSha1(String input) throws NoSuchAlgorithmException,
InvalidKeyException, UnsupportedEncodingException {
        SecretKeySpec signingKey = new
```

```
SecretKeySpec(this.sk.getBytes(DEFAULT ENCODING), "HmacSHA1");
        Mac mac = Mac.getInstance("HmacSHA1");
        mac.init(signingKey);
        return
Base64.getEncoder().encodeToString(mac.doFinal(input.getBytes(DEFAULT_ENCODING)));
    }

    private String stringToSign(String httpMethod, Map<String, String[]> headers,
Map<String, String> queries,
            String bucketName, String objectName) throws Exception{
        String contentMd5 = "";
        String contentType = "";
        String date = "";

        TreeMap<String, String> canonicalizedHeaders = new TreeMap<String,
String>();

        String key;
        List<String> temp = new ArrayList<String>();
        for(Map.Entry<String, String[]> entry : headers.entrySet()) {
            key = entry.getKey();
            if(key == null || entry.getValue() == null || entry.getValue().length
== 0) {
                continue;
            }

            key = key.trim().toLowerCase(Locale.ENGLISH);
            if(key.equals("content-md5")) {
                contentMd5 = entry.getValue()[0];
                continue;
            }

            if(key.equals("content-type")) {
                contentType = entry.getValue()[0];
                continue;
            }

            if(key.equals("date")) {
                date = entry.getValue()[0];
                continue;
            }

            if(key.startsWith(OBS PREFIX)) {

                for(String value : entry.getValue()) {
                    if(value != null) {
                        temp.add(value.trim());
                    }
                }
                canonicalizedHeaders.put(key, this.join(temp, ","));
                temp.clear();
            }
        }

        if(canonicalizedHeaders.containsKey("x-obs-date")) {
```

```
            date = "";
        }


        // handle method/content-md5/content-type/date
        StringBuilder stringToSign = new StringBuilder();
        stringToSign.append(httpMethod).append(SIGN_SEP)
            .append(contentMd5).append(SIGN_SEP)
            .append(contentType).append(SIGN_SEP)
            .append(date).append(SIGN_SEP);

        // handle canonicalizedHeaders
        for(Map.Entry<String, String> entry : canonicalizedHeaders.entrySet()) {

    stringToSign.append(entry.getKey()).append(":").append(entry.getValue()).append
(SIGN_SEP);
        }

        // handle CanonicalizedResource
        stringToSign.append("/");
        if(this.isValid(bucketName)) {
            stringToSign.append(bucketName).append("/");
            if(this.isValid(objectName)) {
                stringToSign.append(this.urlEncode(objectName));
            }
        }

        TreeMap<String, String> canonicalizedResource = new TreeMap<String,
String>();
        for(Map.Entry<String, String> entry : queries.entrySet()) {
            key = entry.getKey();
            if(key == null) {
                continue;
            }

            if(SUB RESOURCES.contains(key)) {
                canonicalizedResource.put(key, entry.getValue());
            }
        }

        if(canonicalizedResource.size() > 0) {
            stringToSign.append("?");
           for(Map.Entry<String, String> entry : canonicalizedResource.entrySet())
{
                stringToSign.append(this.urlEncode(entry.getKey()));
                if(this.isValid(entry.getValue())) {

    stringToSign.append("=").append(this.urlEncode(entry.getValue()));
                }
                        stringToSign.append("&");
            }
                stringToSign.deleteCharAt(stringToSign.length()-1);
        }

//      System.out.println(String.format("StringToSign:%s%s", SIGN_SEP,
```

```
stringToSign.toString()));

        return stringToSign.toString();
    }

    public String headerSignature(String httpMethod, Map<String, String[]> headers,
Map<String, String> queries,
             String bucketName, String objectName) throws Exception {

        //1. stringToSign
        String stringToSign = this.stringToSign(httpMethod, headers, queries,
bucketName, objectName);

        //2. signature
        return String.format("OBS %s:%s", this.ak, this.hamcSha1(stringToSign));
    }


    public String querySignature(String httpMethod, Map<String, String[]> headers,
Map<String, String> queries,
             String bucketName, String objectName, long expires) throws Exception {
        if(headers.containsKey("x-obs-date")) {
            headers.put("x-obs-date", new String[] {String.valueOf(expires)});
        }else {
            headers.put("date", new String[] {String.valueOf(expires)});
        }
        //1. stringToSign
        String stringToSign = this.stringToSign(httpMethod, headers, queries,
bucketName, objectName);

        //2. signature
        return this.urlEncode(this.hamcSha1(stringToSign));
    }

    public static void main(String[] args) throws Exception {

        SignDemo demo = new SignDemo();
        demo.ak = "<your-access-key-id>";
        demo.sk = "<your-securet-key-id>";

        String bucketName = "bucket-test";
        String objectName = "hello.jpg";
        Map<String, String[]> headers = new HashMap<String, String[]>();
        headers.put("date", new String[] {"Sat, 12 Oct 2015 08:12:38 GMT"});
        headers.put("x-obs-acl", new String[] {"public-read"});
        headers.put("x-obs-meta-key1", new String[] {"value1"});
        headers.put("x-obs-meta-key2", new String[] {"value2", "value3"});
        Map<String, String> queries = new HashMap<String, String>();
        queries.put("acl", null);

        System.out.println(demo.headerSignature("PUT", headers, queries,
bucketName, objectName));
    }

}
```

3.2.4 基于浏览器上传的表单中携带签名

OBS 服务支持基于浏览器的 POST 上传对象请求，此类请求的签名信息通过表单的方式上传。POST 上传对象：首先，创建一个安全策略，指定请求中需要满足的条件，比如：桶名、对象名前缀；然后，创建一个基于此策略的签名，需要签名的请求表单中必须包含有效的 signature 和 policy；最后，创建一个表单将对象上传到桶中。

签名的计算过程如下：

1. 对 policy 内容进行 UTF-8 编码。
2. 对第一步的结果进行 Base64 编码。
3. 使用 SK 对第二步的结果进行 HMAC-SHA1 签名计算。
4. 对第三步的结果进行 Base64 编码，得到签名。

```
StringToSign = Base64( UTF-8-Encoding-Of( policy ) )
Signature = Base64( HMAC-SHA1( YourSecretAccessKeyID, StringToSign ) )
```

Policy 的内容如下：

```
{ "expiration": "2017-12-31T12:00:00.000Z",
  "conditions": [
    {"x-obs-acl": "public-read" },
    {"x-obs-security-token": "YwkaRTbdY8g7q...." },
    {"bucket": "book" },
    ["starts-with", "$key", "user/"]
  ]
}
```

Policy 策略中包含有效时间和条件元素。

Expiration

描述本次签名的有效时间 ISO 8601 UTC，如实例中"expiration": "2017-12-31T12:00:00.000Z"表示请求在 2017 年 12 月 31 日 12 点之后无效。该字段是 policy 中必选字段。

Conditions

Condition 是一个用于验证本次请求合法的一种机制，可以使用这些条件限制请求中必须包含的内容。实例中的条件要求请求的桶名必须是 book，对象名必须以 user/为前缀，对象的 acl 必须是公共可读。除了 AccessKeyId、signature、file、policy、token、field names 以及前缀为 x-ignore-外的表单中的所有项，都需要包含在 policy 中。下表是 condtion 中应该包含的项：

表3-15 policy 中应该包含的条件元素

元素名称	描述
x-obs-acl	请求中的 ACL。 支持精确匹配和 starts-with 条件匹配。
content-length-range	设置上传对象的最大最小长度，支持 range 匹配。
Cache-Control, Content-Type, Content-	REST 请求特定头域。

元素名称	描述
Disposition, Content-Encoding, Expires	支持精确匹配和 starts-with 条件匹配。
key	上传对象的名字。 支持精确匹配和 starts-with 条件匹配。
bucket	请求桶名。 支持精确匹配。
success_action_redirect	上传对象成功后重定向的 URL 地址。具体描述请参见 5.3.2 POST 上传。 支持精确匹配和 starts-with 条件匹配。
success_action_status	如果未指定 success_action_redirect，则成功上传时返回给客户端的状态码。具体描述请参见 5.3.2 POST 上传。 支持精确匹配。
x-obs-meta-*	用户自定义元数据。 元素中的关键字不允许含有非 ASCII 码或不可识别字符，如果一定要使用非 ASCII 码或不可识别字符，需要客户端自行做编解码处理，可以采用 URL 编码或者 Base64 编码，服务端不会做解码处理。 支持精确匹配和 starts-with 条件匹配。
x-obs-*	其他以 x-obs-为前缀的头域。 支持精确匹配和 starts-with 条件匹配。
x-obs-security-token	请求消息头中字段名。 临时 AK/SK 和 securitytoken 鉴权必加字段名。

Policy 条件匹配的方式如下：

表3-16 policy 条件匹配方式

条件	描述
Exact Matches	默认是完全匹配，post 表单中该项的值必须和 policy 的 conditions 中设置的值完全一样。例如：上传对象的同时设置对象 ACL 为 public-read，表单中 x-obs-acl 元素的值为 public-read，policy 中的 conditions 可以设置为 {"x-obs-acl": "public-read" }或者["eq", "$x-obs-acl", "public-read"]，这两者是等效的。
Starts With	如果使用该条件，则 post 表单中对应元素的值必须是固定字符串开始。例如：上传对象名以 user/为前缀，表单中 key 元素的值可以是 user/test1、user/test2，policy 的

条件	描述
	conditions 中该条件如下： ["starts-with", "$key", "user/"]
Matching Any Content	post 表单中对应元素的值可以是任意值。例如：请求成功后重定向的地址可以是任意地址，表单中 success_action_redirect 元素的值可以是任意值，policy 的 conditions 中该条件如下： ["starts-with", "$success_action_redirect", ""]
Specifying Ranges	post 表单中 file 元素文件的内容长度可以是一个指定的范围，只用于限制对象大小。例如上传对象大小为 1-10MB，表单中 file 元素的内容长度可以是 1048576-10485760，policy 的 conditions 中该条件如下，注意值没有双引号： ["content-length-range", 1048576, 10485760]

说明

policy 使用 json 格式，conditions 可以支持 { } 和 [ ] 两种方式，{ }中包含表单元素的 key 和 value 两项，以冒号分隔；[ ]中包含条件类型、key、value 三项，以逗号分隔，元素 key 之前使用$字符表示变量。

Policy 中必须转义的字符如下：

表3-17 policy 中必须转义的字符

转义后的字符	真实字符
\\	反斜杠(\)
\$	美元符号($)
\b	退格
\f	换页
\n	换行
\r	回车
\t	水平制表
\v	垂直制表
\u***xxxx***	所有 Unicode 字符

下面的几张表提供了一些请求和 Policy 的例子。

表3-18 在 examplebucket 桶中上传 testfile.txt 对象，并且设置对象 ACL 为公共可读

请求	policy

POST / HTTP/1.1
Host: examplebucket.obs.*region*.example.com
Content-Type: multipart/form-data; boundary=----------7e32233530b26
Content-Length: 1250
----------7e32233530b26
Content-Disposition: form-data; name="key"
testfile.txt
----------7e32233530b26
Content-Disposition: form-data; name="x-obs-acl"
public-read
----------7e32233530b26
Content-Disposition: form-data; name="content-type"
text/plain
----------7e32233530b26
Content-Disposition: form-data; name="AccessKeyId"
UDSIAMSTUBTEST000002
----------7e32233530b26
Content-Disposition: form-data; name="policy"
ewogICJleHBpcmF0aW9uIjogIjIwMTktMDctMDFUMTI6MDA6MDAuMDAwWiIsCiAgImNvbmRpdGlvbnMiOiBbCiAgICB7ImJ1Y2tldCI6ICJleGFtcGxlYnVja2V0IiB9LAogICAgWyJlcSIsICIka2V5IiwgInRlc3R maWxlLnR4dCJdLAogICAgeyJ4LW9icy1hY2wiOiAicHVibGljLXJlYWQiIH0sCiAgICBbImVxIiwgIiRDb250ZW50LVR5cGUiLCAidGV4dC9wbGFpbiJdLAogICAgWyJjb250ZW50LWxlbmd0aC1yYW5nZSIsIDYsIDEwXQogIF0KfQo=
----------7e32233530b26
Content-Disposition: form-data; name="signature"
xxl7bZs/5FgtBUggOdQ88DPZUo0=
----------7e32233530b26
Content-Disposition: form-data; name="file";
	{
	"expiration": "2019-07-01T12:00:00.000Z",
	"conditions": [
	{"bucket": "examplebucket" },
	["eq", "$key", "testfile.txt"],
	{"x-obs-acl": "public-read" },
	["eq", "$Content-Type", "text/plain"]
	]
	}

请求	policy
filename="E:\TEST_FILE\TEST.txt" Content-Type: text/plain 123456 ----------7e32233530b26 Content-Disposition: form-data; name="submit" Upload ----------7e32233530b26--	

表3-19 在 examplebucket 桶中上传 file/obj1 对象，并且设置对象的四个自定义元数据

请求	policy

POST / HTTP/1.1
Host: examplebucket.obs.*region*.example.com
Content-Type: multipart/form-data; boundary=----------7e329d630b26
Content-Length: 1597
----------7e3542930b26
Content-Disposition: form-data; name="key"
file/obj1
----------7e3542930b26
Content-Disposition: form-data; name="AccessKeyId"
UDSIAMSTUBTEST000002
----------7e3542930b26
Content-Disposition: form-data; name="policy"
ewogICJleHBpcmF0aW9uIjogIjIwMTktMDctMDFUMTI6MDA6MDAuMDAwWiIsCiAgImNvbmRpdGlvbnMiOiBbCiAgICB7ImJ1Y2tldCI6ICJleGFtcGxlYnVja2V0IiB9LAogICAgWyJzdGFydHMtd2l0aCIsICIka2V5IiwgImZpbGUvIl0sCiAgICB7Ingtb2JzLW1ldGEtdGVzdDEiOiJ2YWx1ZTEifSwKICAgIFsiZXEiLCAiJHgtb2JzLW1ldGEtdGVzdDIiLCAidmFsdWUyIl0sCiAgICBbInN0YXJ0cy13aXRoIiwgIiR4LW9icy1tZXRhLXRlc3QzIiwgImRvYyJdLAogICAgWyJzdGFydHMtd2l0aCIsICIkeC1vYnMtbWV0YS10ZXN0NCIsICIiXQogIF0KfQo=
	{
	"expiration": "2019-07-01T12:00:00.000Z",
	"conditions": [
	{"bucket": "examplebucket" },
	["starts-with", "$key", "file/"],
	{"x-obs-meta-test1":"value1"},
	["eq", "$x-obs-meta-test2", "value2"],
	["starts-with", "$x-obs-meta-test3", "doc"],
	["starts-with", "$x-obs-meta-test4", ""]
	]
	}

请求	policy
----------7e3542930b26 Content-Disposition: form-data; name="signature" HTId8OCBisn6FfdWKqSJP9RN4Oo= ----------7e3542930b26 Content-Disposition: form-data; name="x-obs-meta-test1" value1 ----------7e3542930b26 Content-Disposition: form-data; name="x-obs-meta-test2" value2 ----------7e3542930b26 Content-Disposition: form-data; name="x-obs-meta-test3" doc123 ----------7e3542930b26 Content-Disposition: form-data; name="x-obs-meta-test4" my ----------7e3542930b26 Content-Disposition: form-data; name="file"; filename="E:\TEST_FILE\TEST.txt" Content-Type: text/plain 123456 ----------7e3542930b26 Content-Disposition: form-data; name="submit" Upload ----------7e3542930b26--	

3.3 返回结果

请求发送以后，您会收到响应，包含状态码、响应消息头和消息体。

状态码

状态码是一组从 2xx（成功）到 4xx 或 5xx（错误）的数字代码，状态码表示了请求响应的状态，完整的状态码列表请参见 6.1 状态码。

响应消息头

对应请求消息头，响应同样也有消息头，如“Content-type”。

详细的公共响应消息头字段请参见表 3-20。

表3-20 公共响应消息头

消息头名称	描述
Content-Length	响应消息体的字节长度。 类型：字符串。 默认值：无。
Connection	指明与服务器的连接是长连接还是短连接。 类型：字符串。 有效值：keep-alive | close。 默认值：无。
Date	OBS 系统响应的时间。 类型：字符串。 默认值：无。
ETag	对象的 base64 编码的 128 位 MD5 摘要。ETag 是对象内容的唯一标识，可以通过该值识别对象内容是否有变化。比如上传对象时 ETag 为 A，下载对象时 ETag 为 B，则说明对象内容发生了变化。实际的 ETag 是对象的哈希值。ETag 只反映变化的内容，而不是其元数据。上传的对象或拷贝操作创建的对象，通过 MD5 加密后都有唯一的 ETag。如果通过多段上传对象，则无论加密方法如何，MD5 会拆分 ETag，此类情况 ETag 就不是 MD5 的摘要。 类型：字符串。
x-obs-id-2	帮助定位问题的特殊符号。 类型：字符串。 默认值：无。
x-obs-request-id	由 OBS 创建来唯一确定本次请求的值，可以通过该值来定位问题。 类型：字符串。 默认值：无。

响应消息体（可选）

响应消息体通常以结构化格式（如 JSON 或 XML）返回，与响应消息头中 Content-type 对应，传递除响应消息头之外的内容。

4 快速入门

4.1 创建桶

操作场景

桶是 OBS 中存储对象的容器。您需要先创建一个桶，然后才能在 OBS 中存储数据。

下面介绍如何调用 5.1.2 创建桶 API 在指定的区域创建一个桶，API 的调用方法请参见 3 如何调用 API。

前提条件

- 已获取 AK 和 SK，获取方法参见 6.3 获取访问密钥（AK/SK）。
- 您需要规划桶所在的区域信息，并根据区域确定调用 API 的 Endpoint，您可以向企业管理员获取区域和终端节点信息。

区域一旦确定，创建完成后无法修改。

在 a1 区域创建一个名为 bucket001 的桶

示例中使用通用的 Apache Http Client。

```
package com.obsclient;

import java.io.*;
import java.util.ArrayList;
import java.util.List;

import org.apache.http.Header;
import org.apache.http.HttpEntity;
import org.apache.http.NameValuePair;
import org.apache.http.client.entity.UrlEncodedFormEntity;
import org.apache.http.client.methods.CloseableHttpResponse;
import org.apache.http.client.methods.HttpGet;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.client.methods.HttpPut;
import org.apache.http.entity.InputStreamEntity;
import org.apache.http.entity.StringEntity;
import org.apache.http.impl.client.CloseableHttpClient;
```

```
import org.apache.http.impl.client.HttpClients;
import org.apache.http.message.BasicNameValuePair;


public class TestMain {

   public static String accessKey = "UDSIAMSTUBTEST000012"; //取值为获取的 AK
   public static String securityKey = "Udsiamstubtest000000UDSIAMSTUBTEST000012";
//取值为获取的 SK

   public static void main(String[] str) {

       createBucket();

   }

   private static void createBucket() {
      CloseableHttpClient httpClient = HttpClients.createDefault();
      String requesttime = DateUtils.formateDate(System.currentTimeMillis());


      HttpPut httpPut = new HttpPut("http://bucket001.obs.a1.example.com");
      httpPut.addHeader("Date", requesttime);

      /** 根据请求计算签名**/
      String contentMD5 = "";
      String contentType = "";
      String canonicalizedHeaders = "";
      String canonicalizedResource = "/bucket001/";
      // Content-MD5 、Content-Type 没有直接换行， data 格式为 RFC 1123，和请求中的时间一致
      String canonicalString = "PUT" + "\n" + contentMD5 + "\n" + contentType +
"\n" + requesttime + "\n" + canonicalizedHeaders + canonicalizedResource;
      System.out.println("StringToSign:[" + canonicalString + "]");
      String signature = null;
      CloseableHttpResponse httpResponse = null;
      try {
         signature = Signature.signWithHmacSha1(securityKey, canonicalString);

         // 增加签名头域 Authorization: OBS AccessKeyID:signature
         httpPut.addHeader("Authorization", "OBS " + accessKey + ":" + signature);
         httpResponse = httpClient.execute(httpPut);

         // 打印发送请求信息和收到的响应消息
         System.out.println("Request Message:");
         System.out.println(httpPut.getRequestLine());
         for (Header header : httpPut.getAllHeaders()) {
            System.out.println(header.getName() + ":" + header.getValue());
         }

         System.out.println("Response Message:");
         System.out.println(httpResponse.getStatusLine());
         for (Header header : httpResponse.getAllHeaders()) {
            System.out.println(header.getName() + ":" + header.getValue());
         }
         BufferedReader reader = new BufferedReader(new InputStreamReader(
```

```
                httpResponse.getEntity().getContent()));

          String inputLine;
          StringBuffer response = new StringBuffer();

          while ((inputLine = reader.readLine()) != null) {
             response.append(inputLine);
          }
          reader.close();

          // print result
          System.out.println(response.toString());
       } catch (UnsupportedEncodingException e) {
          e.printStackTrace();
       } catch (IOException e) {
          e.printStackTrace();
       } finally {
          try {
             httpClient.close();
          } catch (IOException e) {
             e.printStackTrace();
          }
       }

   }

}
```

其中 Date 头域 DateUtils 的格式为：

```
package com.obsclient;

import java.text.DateFormat;
import java.text.SimpleDateFormat;
import java.util.Locale;
import java.util.TimeZone;

public class DateUtils {

   public static String formateDate(long time)
   {
      DateFormat serverDateFormat = new SimpleDateFormat("EEE, dd MMM yyyy
HH:mm:ss z", Locale.ENGLISH);
      serverDateFormat.setTimeZone(TimeZone.getTimeZone("GMT"));
      return serverDateFormat.format(time);
   }
}
```

签名字符串 Signature 的计算方法为：

```
package com.obsclient;

import javax.crypto.Mac;
import javax.crypto.spec.SecretKeySpec;
import java.io.UnsupportedEncodingException;
import java.security.NoSuchAlgorithmException;
import java.security.InvalidKeyException;
```

```
import java.util.Base64;

public class Signature {
   public static String signWithHmacSha1(String sk, String canonicalString) throws
UnsupportedEncodingException {

      try {
         SecretKeySpec signingKey = new SecretKeySpec(sk.getBytes("UTF-8"),
"HmacSHA1");
         Mac mac = Mac.getInstance("HmacSHA1");
         mac.init(signingKey);
         return
Base64.getEncoder().encodeToString(mac.doFinal(canonicalString.getBytes("UTF-8")));
      } catch (NoSuchAlgorithmException | InvalidKeyException |
UnsupportedEncodingException e) {
         e.printStackTrace();
      }
      return null;
   }
}
```

4.2 获取桶列表

操作场景

如果用户想要查看自己创建的所有桶信息，可以使用获取桶列表接口查看。

下面介绍如何调用 5.1.1 获取桶列表 API，API 的调用方法请参见 3 如何调用 API。

前提条件

- 已获取 AK 和 SK，获取方法参见 6.3 获取访问密钥（AK/SK）。
- 您需要明确需要列举的桶所在的区域信息，并根据区域确定调用 API 的 Endpoint，您可以向企业管理员获取区域和终端节点信息。

获取 a1 区域的桶列表

示例中使用通用的 Apache Http Client。

```
package com.obsclient;

import java.io.*;
import java.util.ArrayList;
import java.util.List;

import org.apache.http.Header;
import org.apache.http.HttpEntity;
import org.apache.http.NameValuePair;
import org.apache.http.client.entity.UrlEncodedFormEntity;
import org.apache.http.client.methods.CloseableHttpResponse;
import org.apache.http.client.methods.HttpGet;
import org.apache.http.client.methods.HttpPost;
```

```
import org.apache.http.client.methods.HttpPut;
import org.apache.http.entity.InputStreamEntity;
import org.apache.http.entity.StringEntity;
import org.apache.http.impl.client.CloseableHttpClient;
import org.apache.http.impl.client.HttpClients;
import org.apache.http.message.BasicNameValuePair;


public class TestMain {

   public static String accessKey = "UDSIAMSTUBTEST000012"; //取值为获取的 AK
   public static String securityKey = "Udsiamstubtest000000UDSIAMSTUBTEST000012";
//取值为获取的 SK

   public static void main(String[] str) {

      listAllMyBuckets();

   }


   private static void listAllMyBuckets() {
      CloseableHttpClient httpClient = HttpClients.createDefault();
      String requesttime = DateUtils.formateDate(System.currentTimeMillis());


      HttpGet httpGet = new HttpGet("http://obs.a1.example.com");
      httpGet.addHeader("Date", requesttime);

      /** 根据请求计算签名**/
      String contentMD5 = "";
      String contentType = "";
      String canonicalizedHeaders = "";
      String canonicalizedResource = "/";
      // Content-MD5 、Content-Type 没有直接换行， data 格式为 RFC 1123，和请求中的时间一致
      String canonicalString = "GET" + "\n" + contentMD5 + "\n" + contentType +
"\n" + requesttime + "\n" + canonicalizedHeaders + canonicalizedResource;
      System.out.println("StringToSign:[" + canonicalString + "]");
      String signature = null;
      try {
         signature = Signature.signWithHmacSha1(securityKey, canonicalString);

         // 增加签名头域 Authorization: OBS AccessKeyID:signature
         httpGet.addHeader("Authorization", "OBS " + accessKey + ":" + signature);
         CloseableHttpResponse httpResponse = httpClient.execute(httpGet);

         // 打印发送请求信息和收到的响应消息
         System.out.println("Request Message:");
         System.out.println(httpGet.getRequestLine());
         for (Header header : httpGet.getAllHeaders()) {
            System.out.println(header.getName() + ":" + header.getValue());
         }

         System.out.println("Response Message:");
         System.out.println(httpResponse.getStatusLine());
```

```
            for (Header header : httpResponse.getAllHeaders()) {
               System.out.println(header.getName() + ":" + header.getValue());
            }
            BufferedReader reader = new BufferedReader(new InputStreamReader(
                  httpResponse.getEntity().getContent()));

            String inputLine;
            StringBuffer response = new StringBuffer();

            while ((inputLine = reader.readLine()) != null) {
               response.append(inputLine);
            }
            reader.close();
            // print result
            System.out.println(response.toString());
         } catch (UnsupportedEncodingException e) {
            e.printStackTrace();

         } catch (IOException e) {
            e.printStackTrace();
         } finally {
            try {
               httpClient.close();
            } catch (IOException e) {
               e.printStackTrace();
            }
         }

      }

}
```

其中 Date 头域 DateUtils 的格式为：

```
package com.obsclient;

import java.text.DateFormat;
import java.text.SimpleDateFormat;
import java.util.Locale;
import java.util.TimeZone;

public class DateUtils {

   public static String formateDate(long time)
   {
      DateFormat serverDateFormat = new SimpleDateFormat("EEE, dd MMM yyyy
HH:mm:ss z", Locale.ENGLISH);
      serverDateFormat.setTimeZone(TimeZone.getTimeZone("GMT"));
      return serverDateFormat.format(time);
   }
}
```

签名字符串 Signature 的计算方法为：

```
package com.obsclient;

import javax.crypto.Mac;
```

```
import javax.crypto.spec.SecretKeySpec;
import java.io.UnsupportedEncodingException;
import java.security.NoSuchAlgorithmException;
import java.security.InvalidKeyException;
import java.util.Base64;

public class Signature {
   public static String signWithHmacSha1(String sk, String canonicalString) throws
UnsupportedEncodingException {

      try {
         SecretKeySpec signingKey = new SecretKeySpec(sk.getBytes("UTF-8"),
"HmacSHA1");
         Mac mac = Mac.getInstance("HmacSHA1");
         mac.init(signingKey);
         return
Base64.getEncoder().encodeToString(mac.doFinal(canonicalString.getBytes("UTF-8")));
      } catch (NoSuchAlgorithmException | InvalidKeyException |
UnsupportedEncodingException e) {
         e.printStackTrace();
      }
      return null;
   }
}
```

4.3 上传对象

操作场景

您可以根据需要，将任何类型的文件上传到 OBS 桶中进行存储。

下面介绍如何调用 5.3.1 PUT 上传 API 在指定的桶中上传对象，API 的调用方法请参见 3 如何调用 API。

前提条件

- 已获取 AK 和 SK，获取方法参见 6.3 获取访问密钥（AK/SK）。
- 已创建了至少一个可用的桶。
- 已准备好了待上传的文件，并清楚文件所在的本地完整路径。
- 您需要知道待上传桶所在的区域信息，并根据区域确定调用 API 的 Endpoint，您可以向企业管理员获取区域和终端节点信息。

向 a1 区域的桶 bucket001 中上传对象，名称为 objecttest1

示例中使用通用的 Apache Http Client。

```
package com.obsclient;

import java.io.*;
import java.util.ArrayList;
import java.util.List;
```

```
import org.apache.http.Header;
import org.apache.http.HttpEntity;
import org.apache.http.NameValuePair;
import org.apache.http.client.entity.UrlEncodedFormEntity;
import org.apache.http.client.methods.CloseableHttpResponse;
import org.apache.http.client.methods.HttpGet;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.client.methods.HttpPut;
import org.apache.http.entity.InputStreamEntity;
import org.apache.http.entity.StringEntity;
import org.apache.http.impl.client.CloseableHttpClient;
import org.apache.http.impl.client.HttpClients;
import org.apache.http.message.BasicNameValuePair;


public class TestMain {

   public static String accessKey = "UDSIAMSTUBTEST000012"; //取值为获取的 AK
   public static String securityKey = "Udsiamstubtest000000UDSIAMSTUBTEST000012";
//取值为获取的 SK

   public static void main(String[] str) {

      putObjectToBucket();

   }


   private static void putObjectToBucket() {

      InputStream inputStream = null;
      CloseableHttpClient httpClient = HttpClients.createDefault();
      CloseableHttpResponse httpResponse = null;
      String requesttime = DateUtils.formateDate(System.currentTimeMillis());


      HttpPut httpPut = new
HttpPut("http://bucket001.obs.a1.example.com/objecttest1");
      httpPut.addHeader("Date", requesttime);

      /** 根据请求计算签名 **/
      String contentMD5 = "";
      String contentType = "";
      String canonicalizedHeaders = "";
      String canonicalizedResource = "/bucket001/objecttest1";
      // Content-MD5 、Content-Type 没有直接换行， data 格式为 RFC 1123，和请求中的时间一致
      String canonicalString = "PUT" + "\n" + contentMD5 + "\n" + contentType +
"\n" + requesttime + "\n" + canonicalizedHeaders + canonicalizedResource;
      System.out.println("StringToSign:[" + canonicalString + "]");
      String signature = null;
      try {
         signature = Signature.signWithHmacSha1(securityKey, canonicalString);
         // 上传的文件目录
         inputStream = new FileInputStream("D:\\OBSobject\\text01.txt");
```

```
            InputStreamEntity entity = new InputStreamEntity(inputStream);
            httpPut.setEntity(entity);

            // 增加签名头域 Authorization: OBS AccessKeyID:signature
            httpPut.addHeader("Authorization", "OBS " + accessKey + ":" + signature);
            httpResponse = httpClient.execute(httpPut);

            // 打印发送请求信息和收到的响应消息
            System.out.println("Request Message:");
            System.out.println(httpPut.getRequestLine());
            for (Header header : httpPut.getAllHeaders()) {
               System.out.println(header.getName() + ":" + header.getValue());
            }

            System.out.println("Response Message:");
            System.out.println(httpResponse.getStatusLine());
            for (Header header : httpResponse.getAllHeaders()) {
               System.out.println(header.getName() + ":" + header.getValue());
            }
            BufferedReader reader = new BufferedReader(new InputStreamReader(
                  httpResponse.getEntity().getContent()));

            String inputLine;
            StringBuffer response = new StringBuffer();

            while ((inputLine = reader.readLine()) != null) {
               response.append(inputLine);
            }
            reader.close();

            // print result
            System.out.println(response.toString());


         } catch (UnsupportedEncodingException e) {
            e.printStackTrace();

         } catch (IOException e) {
            e.printStackTrace();
         } finally {
            try {
               httpClient.close();
            } catch (IOException e) {
               e.printStackTrace();
            }
         }
      }

}
```

其中 Date 头域 DateUtils 的格式为：

```
package com.obsclient;

import java.text.DateFormat;
import java.text.SimpleDateFormat;
```

```
import java.util.Locale;
import java.util.TimeZone;

public class DateUtils {

   public static String formateDate(long time)
   {
      DateFormat serverDateFormat = new SimpleDateFormat("EEE, dd MMM yyyy
HH:mm:ss z", Locale.ENGLISH);
      serverDateFormat.setTimeZone(TimeZone.getTimeZone("GMT"));
      return serverDateFormat.format(time);
   }
}
```

签名字符串 Signature 的计算方法为：

```
package com.obsclient;

import javax.crypto.Mac;
import javax.crypto.spec.SecretKeySpec;
import java.io.UnsupportedEncodingException;
import java.security.NoSuchAlgorithmException;
import java.security.InvalidKeyException;
import java.util.Base64;

public class Signature {
   public static String signWithHmacSha1(String sk, String canonicalString) throws
UnsupportedEncodingException {

      try {
         SecretKeySpec signingKey = new SecretKeySpec(sk.getBytes("UTF-8"),
"HmacSHA1");
         Mac mac = Mac.getInstance("HmacSHA1");
         mac.init(signingKey);
         return
Base64.getEncoder().encodeToString(mac.doFinal(canonicalString.getBytes("UTF-8")));
      } catch (NoSuchAlgorithmException | InvalidKeyException |
UnsupportedEncodingException e) {
         e.printStackTrace();
      }
      return null;
   }
}
```

5 API

5.1 桶的基础操作

5.1.1 获取桶列表

功能介绍

OBS 用户可以通过请求查询自己创建的桶列表。

请求消息样式

```
GET / HTTP/1.1
Host: obs.region.example.com
Date: date
Authorization: authorization
```

请求消息参数

该请求消息中不带请求参数。

请求消息头

该请求消息头使用公共消息字段，具体请参见表 3-3。

请求消息元素

该请求消息中不带请求元素。

响应消息样式

```
GET HTTP/1.1 status code
Content-Type: type
Date: date
Content-Length: length

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ListAllMyBucketsResult xmlns="http://obs.region.example.com/doc/2015-06-30/">
   <Owner>
```

```
      <ID>id</ID>
      <DisplayName>displayName</DisplayName>
   </Owner>
   <Buckets>
      <Bucket>
         <Name>bucketName</Name>
         <CreationDate>date</CreationDate>
         <Location>region</Location>
      </Bucket>
      ...
   </Buckets>
</ListAllMyBucketsResult>
```

响应消息头

该请求的响应消息使用公共消息头，具体请参考表 3-20。

响应消息元素

该请求的响应消息中，会以 XML 形式将用户拥有的桶列出来，元素的具体含义如表 5-1 所示。

表5-1 响应消息元素

元素名称	描述
ListAllMyBucketsResult	用户的桶列表。 类型：XML。
Owner	桶拥有者信息，包含租户 ID 和租户名。 类型：XML。
ID	用户的 DomainID（账号 ID）。 类型：字符串。
DisplayName	用户的 DomainName（账号名）。 类型：字符串。
Buckets	用户所拥有的桶列表。 类型：XML。
Bucket	具体的桶信息。 类型：XML。
Name	桶名称。 类型：字符串。
CreationDate	桶的创建时间。 类型：字符串。
Location	桶的位置信息。

元素名称	描述
	类型：字符串

错误响应消息

该请求无特殊错误，所有错误已经包含在表 6-3 中。

请求示例

```
GET / HTTP/1.1
User-Agent: curl/7.29.0
Host: obs.region.example.com
Accept: */*
Date: Mon, 25 Jun 2018 05:37:12 +0000
Authorization: OBS GKDF4C7Q6SI0IPGTXTJN:9HXkVQIiQKw33UEmyBI4rWrzmic=
```

响应示例

```
HTTP/1.1 200 OK
Server: OBS
x-obs-request-id: BF260000016435722C11379647A8A00A
x-obs-id-2: 32AAAQAAEAABAAAQAAEAABAAAQAAEAABCSGGDRUM62QZi3hGP8Fz3gOloYCfZ39U
Content-Type: application/xml
Date: Mon, 25 Jun 2018 05:37:12 GMT
Content-Length: 460

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ListAllMyBucketsResult xmlns="http://obs.example.com/doc/2015-06-30/">
  <Owner>
    <ID>783fc6652cf246c096ea836694f71855</ID>
    <DisplayName>domainnamedom000003</DisplayName>
  </Owner>
  <Buckets>
    <Bucket>
      <Name>examplebucket01</Name>
      <CreationDate>2018-06-21T09:15:01.032Z</CreationDate>
      <Location>region</Location>
    </Bucket>
    <Bucket>
      <Name>examplebucket02</Name>
      <CreationDate>2018-06-22T03:56:33.700Z</CreationDate>
      <Location>region</Location>
    </Bucket>
  </Buckets>
</ListAllMyBucketsResult>
```

5.1.2 创建桶

功能介绍

创建桶是指按照用户指定的桶名创建一个新桶的操作。

说明

- 默认情况下，一个用户可以拥有的桶的数量不能超过 100 个。
- 用户删除桶后，需要等待 30 分钟才能创建同名桶。

新创建桶的桶名在 OBS 中必须是唯一的。如果是同一个用户重复创建同一区域的同名桶时返回成功。除此以外的其他场景重复创建同名桶返回桶已存在。用户可以在请求消息头中加入 x-obs-acl 等参数，设置要创建桶的权限控制策略。

请求消息样式

```
PUT / HTTP/1.1
Host: bucketname.obs.region.example.com
Content-Length: length
Date: date
Authorization: authorization
<CreateBucketConfiguration xmlns="http://obs.region.example.com/doc/2015-06-30/">
   <Location>location</Location>
</CreateBucketConfiguration>
```

请求消息参数

该请求消息中不带请求参数。

请求消息头

该操作消息头与普通请求一样，请参见表 3-3，但可以带附加消息头，附加请求消息头如下所示。

表5-2 附加请求消息头

消息头名称	描述	是否必选
x-obs-acl	创建桶时，可以加上此消息头设置桶的权限控制策略，使用的策略为预定义的常用策略，包括：private、public-read、public-read-write、public-read-delivered、public-read-write-delivered。 类型：字符串。	否
x-obs-grant-read	授权给指定 domain 下的所有用户有 READ 权限。允许列举桶内对象、列举桶中多段任务、列举桶中多版本对象、获取桶元数据。 类型：字符串 示例：x-obs-grant-read:id=租户 id	否
x-obs-grant-write	授权给指定 domain 下的所有用户有 WRITE 权限。允许创建、删除、覆盖桶内所有对象，允许初始化段、上传段、拷贝段、合并段、取消多段上传任务。 类型：字符串 示例：x-obs-grant-write:id=租户 id	否

消息头名称	描述	是否必选
x-obs-grant-read-acp	授权给指定 domain 下的所有用户有 READ_ACP 权限。允许读桶的 ACL 信息。 类型：字符串 示例：x-obs-grant-read-acp:id=租户 id	否
x-obs-grant-write-acp	授权给指定 domain 下的所有用户有 WRITE_ACP 权限，允许修改桶的 ACL 信息。 类型： 字符串 示例：x-obs-grant-write-acp:id=租户 id	否
x-obs-grant-full-control	授权给指定 domain 下的所有用户有 FULL_CONTROL 权限。 类型： 字符串 示例：x-obs-grant-full-control:id=租户 id	否
x-obs-grant-read-delivered	授权给指定 domain 下的所有用户有 READ 权限，并且在默认情况下，该 READ 权限将传递给桶内所有对象。 类型：字符串 示例：x-obs-grant-read-delivered:id=租户 id	否
x-obs-grant-full-control-delivered	授权给指定 domain 下的所有用户有 FULL_CONTROL 权限，并且在默认情况下，该 FULL_CONTROL 权限将传递给桶内所有对象。 类型：字符串 示例：x-obs-grant-full-control-delivered:id=租户 id	否

请求消息元素

该操作可以带附加请求消息元素，附加请求消息元素的具体描述如表 5-3 所示。

表5-3 附加请求消息元素

元素名称	描述	是否必选
Location	指定 Bucket 在哪个区域被创建。 请向企业管理员获取区域和终端节点信息。 类型：字符串。	否

响应消息样式

```
HTTP/1.1 status code
Location: location
Date: date
Content-Length: length
```

响应消息头

该请求的响应消息使用公共消息头，具体请参考表 3-20。

响应消息元素

该请求的响应中不带有响应元素。

错误响应消息

无特殊错误，所有错误已经包含在表 6-3 中。

请求示例 1

创建桶

```
PUT / HTTP/1.1
User-Agent: curl/7.29.0
Host: examplebucket.obs.region.example.com
Accept: */*
Date: WED, 01 Jul 2015 02:25:05 GMT
Authorization: OBS H4IPJX0TQTHTHEBQQCEC:75/Y4Ng1izvzc1nTGxpMXTE6ynw=
Content-Length: 157

<CreateBucketConfiguration xmlns="http://obs.region.example.com/doc/2015-06-30/">
   <Location>region</Location>
</CreateBucketConfiguration>
```

响应示例 1

```
HTTP/1.1 200 OK
Server: OBS
x-obs-request-id: BF260000016435CE298386946AE4C482
Location: /examplebucket
x-obs-id-2: 32AAAQAAEAABSAAgAAEAABAAAQAAEAABCT9W2tcvLmMJ+plfdopaD62S0npbaRUz
Date: WED, 01 Jul 2015 02:25:06 GMT
Content-Length: 0
```

请求示例 2

创建指定 ACL 的桶

```
PUT / HTTP/1.1
User-Agent: curl/7.29.0
Host: examplebucket.obs.region.example.com
Accept: */*
Date: WED, 01 Jul 2015 02:25:05 GMT
```

```
x-obs-acl:public-read
Authorization: OBS H4IPJX0TQTHTHEBQQCEC:75/Y4Ng1izvzc1nTGxpMXTE6ynw=
Content-Length: 157

<CreateBucketConfiguration xmlns="http://obs.region.example.com/doc/2015-06-30/">
   <Location>region</Location>
</CreateBucketConfiguration>
```

响应示例 2

```
HTTP/1.1 200 OK
Server: OBS
x-obs-request-id: BF260000016435CE298386946AE4C482
Location: /examplebucket
x-obs-id-2: 32AAAQAAEAABSAAgAAEAABAAAQAAEAABCT9W2tcvLmMJ+plfdopaD62S0npbaRUz
Date: WED, 01 Jul 2015 02:25:06 GMT
Content-Length: 0
```

5.1.3 列举桶内对象

功能介绍

对桶拥有读权限的用户可以执行获取桶内对象列表的操作。

如果用户在请求的 URI 里只指定了桶名，即 GET /BucketName，则返回信息中会包含桶内部分或所有对象的描述信息（一次最多返回 1000 个对象信息）；如果用户还指定了 prefix、marker、max-keys、delimiter 参数中的一个或多个，则返回的对象列表将按照如表 5-4 所示规定的语义返回指定的对象。

用户也可以请求参数中添加 versions 参数来执行列举桶内多版本对象的操作。

请求消息样式

```
GET / HTTP/1.1
Host: bucketname.obs.region.example.com
Date: date
Authorization: authorization
```

请求消息样式（多版本）

```
GET /?versions HTTP/1.1
Host: bucketname.obs.region.example.com
Date: date
Authorization: authorization
```

请求消息参数

该请求可以通过带参数，列举出桶内的一部分对象，参数的具体含义如表 5-4 所示。

表5-4 请求消息参数

参数名称	描述	是否必选

参数名称	描述	是否必选
prefix	列举以指定的字符串 prefix 开头的对象。 类型：字符串。	否
marker	列举桶内对象列表时，指定一个标识符，返回的对象列表将是按照字典顺序排序后这个标识符以后的所有对象。 类型：字符串。	否
max-keys	指定返回的最大对象数，返回的对象列表将是按照字典顺序的最多前 max-keys 个对象，范围是[1，1000]，超出范围时，按照默认的 1000 进行处理。 类型：整型。	否
delimiter	用来指定将对象名按照特定字符分割的分割符。如果指定了 prefix 参数，按 delimiter 对所有对象命名进行分割，多个对象分割后 prefix 到一个 delimiter 间都相同对象会形成一条 CommonPrefixes；如果没有携带 prefix 参数，按 delimiter 对所有对象命名进行分割，多个对象分割后从对象名开始到第一个 delimiter 之间相同的部分形成一条 CommonPrefixes。 类型：字符。	否
version-id-marker	与响应中的 key-marker 配合使用，返回的对象列表将是按照字典顺序排序后在该标识符以后的所有对象。如果 version-id-marker 不是 key-marker 的一个版本号，则该参数无效。 类型：字符串 有效值：对象的版本号	否

请求消息头

该请求使用公共的请求消息头，具体如表 3-3 所示。

请求消息元素

该请求消息头中不带消息元素。

响应消息样式

```
HTTP/1.1 status_code
Date: date
x-obs-bucket-location: region
Content-Type: application/xml
Content-Length: length
<Response Body>
```

响应消息头

该请求的响应消息使用公共消息头，具体请参考表 3-20。

响应消息元素

该请求的响应消息中，会以 XML 形式将桶中的对象列出来，元素的具体含义如表 5-5 所示。

表5-5 响应消息元素

元素名称	描述
ListBucketResult	桶中对象列表。 类型：XML
Contents	对象的元数据信息。 类型：XML 父节点：ListBucketResult
CommonPrefixes	请求中带 delimiter 参数时，返回消息带 CommonPrefixes 分组信息。 类型：XML 父节点：ListBucketResult
Delimiter	请求中携带的 delimiter 参数。 类型：字符串 父节点：ListBucketResult
ETag	对象的 base64 编码的 128 位 MD5 摘要。ETag 是对象内容的唯一标识，可以通过该值识别对象内容是否有变化。比如上传对象时 ETag 为 A，下载对象时 ETag 为 B，则说明对象内容发生了变化。实际的 ETag 是对象的哈希值。ETag 只反映变化的内容，而不是其元数据。上传的对象或拷贝操作创建的对象，通过 MD5 加密后都有唯一的 ETag。（当对象是服务端加密的对象时，ETag 值不是对象的 MD5 值，而是通过服务端加密计算出的唯一标识。） 类型：字符串 父节点：ListBucketResult.Contents
Type	对象类型，非 Normal 对象时返回。 类型：字符串 父节点：ListBucketResult.Contents
ID	对象拥有者的租户 ID。 类型：字符串 父节点：ListBucketResult.Contents.Owner

元素名称	描述
IsTruncated	表明是否本次返回的 ListBucketResult 结果列表被截断。“true”表示本次没有返回全部结果；“false”表示本次已经返回了全部结果。 类型：Boolean 父节点：ListBucketResult
Key	对象名。 类型：字符串 父节点：ListBucketResult.Contents
LastModified	对象最近一次被修改的时间。 类型：Date 父节点：ListBucketResult.Contents
Marker	列举对象时的起始位置。 类型：字符串 父节点：ListBucketResult
NextMarker	如果本次没有返回全部结果，响应请求中将包含此字段，用于标明本次请求列举到的最后一个对象。后续请求可以指定 Marker 等于该值来列举剩余的对象。 类型：字符串 父节点：ListBucketResult
MaxKeys	列举时最多返回的对象个数。 类型：字符串 父节点：ListBucketResult
Name	本次请求的桶名。 类型：字符串 父节点：ListBucketResult
Owner	用户信息，包含用户 DomainId 和用户名。 类型：XML 父节点：ListBucketResult.Contents
Prefix	对象名的前缀，表示本次请求只列举对象名能匹配该前缀的所有对象。 类型：字符串 父节点：ListBucketResult
Size	对象的字节数。 类型：字符串 父节点：ListBucketResult.Contents

表5-6 列举多版本对象响应消息元素

元素名称	描述
ListVersionsResult	保存列举桶中对象列表（含多版本）请求结果的容器。 类型：容器
Name	桶名。 类型：字符串 父节点：ListVersionsResult
Prefix	对象名的前缀，表示本次请求只列举对象名能匹配该前缀的所有对象。类型：字符串 父节点：ListVersionsResult
KeyMarker	列举对象时的起始位置。 类型：字符串 父节点：ListVersionsResult
VersionIdMarker	列举对象时的起始位置。 类型：字符串 父节点：ListVersionsResult
NextKeyMarker	如果本次没有返回全部结果，响应请求中将包含该元素，用于标明接下来请求的 KeyMarker 值。 类型：字符串 父节点: ListVersionsResult。
NextVersionIdMarker	如果本次没有返回全部结果，响应请求中将包含该元素，用于标明接下来请求的 VersionIdMarker 值。 类型: 字符串 父节点: ListVersionsResult。
MaxKeys	列举时最多返回的对象个数。 类型：字符串 父节点：ListVersionsResult
IsTruncated	表明是否本次返回的 ListVersionsResult 结果列表被截断。“true”表示本次没有返回全部结果；“false”表示本次已经返回了全部结果。 类型：布尔值

元素名称	描述
	父节点：ListVersionsResult
Version	保存版本信息的容器 类型：容器 父节点：ListVersionsResult
DeleteMarker	保存删除标记的容器 类型：容器 父节点：ListVersionsResult
Key	对象名。 类型：字符串 父节点：ListVersionsResult.Version | ListVersionsResult.DeleteMarker
VersionId	对象的版本号。 类型：字符串 父节点：ListVersionsResult.Version | ListVersionsResult.DeleteMarker
IsLatest	标识对象是否是最新的版本，true 代表是最新的版本。 类型：布尔值 父节点：ListVersionsResult.Version | ListVersionsResult.DeleteMarker
LastModified	对象最近一次被修改的时间。 类型：Date 父节点：ListVersionsResult.Version | ListVersionsResult.DeleteMarker
ETag	对象的 base64 编码的 128 位 MD5 摘要。ETag 是对象内容的唯一标识，可以通过该值识别对象内容是否有变化。实际标签是对象的哈希。比如上传对象时 ETag 为 A，下载对象时 ETag 为 B，则说明对象内容发生了变化。ETag 只反映变化的内容，而不是其元数据。上传的对象或拷贝操作创建的对象，通过 MD5 加密后都有唯一的 ETag。 类型：字符串 父节点：ListVersionsResult.Version
Type	对象类型，非 Normal 对象时返回。 类型：字符串

元素名称	描述
	父节点：ListVersionsResult.Version
Size	对象的字节数。 类型：字符串 父节点：ListVersionsResult.Version
Owner	用户信息，包含用户 DomainId 和用户名。 类型：容器 父节点：ListVersionsResult.Version | ListVersionsResult.DeleteMarker
ID	对象所有者的 DomainId。 类型：字符串 父节点：ListVersionsResult.Version.Owner | ListVersionsResult.DeleteMarker.Owner
CommonPrefixes	请求中带 delimiter 参数时，返回消息带 CommonPrefixes 分组信息。 类型：容器 父节点: ListVersionsResult。
Prefix	CommonPrefixes 分组信息中，表明不同的 Prefix。 类型：字符串 父节点: ListVersionsResult.CommonPrefixes。

错误响应消息

无特殊错误，所有错误已经包含在表 6-3 中。

请求示例 1

列举所有对象

```
GET / HTTP/1.1
User-Agent: curl/7.29.0
Host: examplebucket.obs.region.example.com
Accept: */*
Date: WED, 01 Jul 2015 02:28:25 GMT
Authorization: OBS H4IPJX0TQTHTHEBQQCEC:KiyoYze4pmRNPYfmlXBfRTVxt8c=
```

响应示例 1

```
HTTP/1.1 200 OK
Server: OBS
x-obs-request-id: BF260000016435D34E379ABD93320CB9
x-obs-id-2: 32AAAQAAEAABAAAQAAEAABAAAQAAEAABCSXiN7GPL/yXM6OSBaYCUV1zcY5OelWp
Content-Type: application/xml
Date: WED, 01 Jul 2015 02:23:30 GMT
Content-Length: 586

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ListBucketResult xmlns="http://obs.example.com/doc/2015-06-30/">
  <Name>examplebucket</Name>
  <Prefix/>
  <Marker/>
  <MaxKeys>1000</MaxKeys>
  <IsTruncated>false</IsTruncated>
  <Contents>
    <Key>object001</Key>
    <LastModified>2015-07-01T00:32:16.482Z</LastModified>
    <ETag>"2fa3bcaaec668adc5da177e67a122d7c"</ETag>
    <Size>12041</Size>
    <Owner>
      <ID>b4bf1b36d9ca43d984fbcb9491b6fce9</ID>
    </Owner>
  </Contents>
</ListBucketResult>
```

请求示例 2

筛选对象

用户有桶名为 examplebucket，桶内共有三个名为 newfile，obj001，obj002 的对象，如果只列需要查看对象名为 obj002 的对象，请求消息格式为：

```
GET /examplebucket/?marker=obj002&prefix=obj HTTP/1.1
User-Agent: curl/7.29.0
Host: examplebucket.obs.region.example.com
Accept: */*
Date: WED, 01 Jul 2015 02:28:25 GMT
Authorization: OBS H4IPJX0TQTHTHEBQQCEC:KiyoYze4pmRNPYfmlXBfRTVxt8c=
```

响应示例 2

```
HTTP/1.1 200 OK
Server: OBS
x-obs-request-id: BF2600000016435D758FBA857E0801874
x-obs-id-2: 32AAAQAAEAABAAAQAAEAABAAAQAAEAABCShn/xAyk/xHBX6qgGSB36WXrbco0X80
Content-Type: application/xml
Date: WED, 01 Jul 2015 02:29:48 GMT
Content-Length: 707

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ListBucketResult xmlns="http://obs.example.com/doc/2015-06-30/">
<Name>examplebucket</Name>
<Prefix>obj</Prefix>
```

```
<Marker>obj002</Marker>
<MaxKeys>1000</MaxKeys>
<IsTruncated>false</IsTruncated>
  <Contents>
    <Key>obj002</Key>
    <LastModified>2015-07-01T02:11:19.775Z</LastModified>
    <ETag>"a72e382246ac83e86bd203389849e71d"</ETag>
    <Size>9</Size>
    <Owner>
      <ID>b4bf1b36d9ca43d984fbcb9491b6fce9</ID>
    </Owner>
  </Contents>
</ListBucketResult>
```

请求示例 3

多版本

```
GET /?versions HTTP/1.1
User-Agent: curl/7.29.0
Host: examplebucket.obs.region.example.com
Accept: */*
Date: WED, 01 Jul 2015 02:29:45 GMT
Authorization: OBS H4IPJX0TQTHTHEBQQCEC:iZeDESIMxBK2YODk7vIeVpyO8DI=
```

响应示例 3

```
HTTP/1.1 200 OK
Server: OBS
x-obs-request-id: BF260000016435D758FBA857E0801874
x-obs-id-2: 32AAAQAAEAABAAAQAAEAABAAAQAAEAABCShn/xAyk/xHBX6qgGSB36WXrbco0X80
Content-Type: application/xml
Date: WED, 01 Jul 2015 02:29:48 GMT
Content-Length: 707

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ListVersionsResult xmlns="http://obs.example.com/doc/2015-06-30/">
 <Name>bucket02</Name>
  <Prefix/>
  <KeyMarker/>
  <VersionIdMarker/>
  <MaxKeys>1000</MaxKeys>
  <IsTruncated>false</IsTruncated>
  <Contents>
    <Key>object001</Key>

<VersionId>0001100000000000013F16000001643A22E476FFFF9046024ECA3655445346485a</Versio
nId>
    <LastModified>2015-07-01T00:32:16.482Z</LastModified>
    <ETag>"2fa3bcaaec668adc5da177e67a122d7c"</ETag>
    <Size>12041</Size>
    <Owner>
      <ID>b4bf1b36d9ca43d984fbcb9491b6fce9</ID>
    </Owner>
```

```
  </Contents>
</ListVersionsResult>
```

5.1.4 获取桶元数据

功能介绍

对桶拥有读权限的用户可以执行查询桶元数据是否存在的操作。

请求消息样式

```
HEAD / HTTP/1.1
Host: bucketname.obs.region.example.com
Date: date
Authorization: authorization
```

请求消息参数

该请求消息中不带消息参数。

请求消息头

该请求使用公共消息头，具体参见表 3-3。

如果想要获取 CORS 配置信息，则需要使用的消息头如下表 5-7 所示。

表5-7 获取 CORS 配置的请求消息头

消息头名称	描述	是否必选
Origin	预请求指定的跨域请求 Origin（通常为域名）。 类型：字符串	是
Access-Control-Request-Headers	实际请求可以带的 HTTP 头域，可以带多个头域。 类型：字符串	否

请求消息元素

该请求消息中不带消息元素。

响应消息样式

```
HTTP/1.1 status code
x-obs-bucket-location: region
Date: date
```

响应消息头

该请求的响应消息使用公共消息头，具体请参考表 3-20。

除公共响应消息头之外，还可能使用如下表 5-8 中的消息头。

表5-8 附加响应消息头

消息头名称	描述
x-obs-bucket-location	桶的区域位置信息。 类型：字符串。
x-obs-version	桶所在的 OBS 服务版本号。 类型：字符串
x-obs-epid	当前桶的企业项目 id。 类型：字符串
Access-Control-Allow-Origin	当桶设置了 CORS 配置，如果请求的 Origin 满足服务端的 CORS 配置，则在响应中包含这个 Origin。 类型：字符串
Access-Control-Allow-Headers	当桶设置了 CORS 配置，如果请求的 headers 满足服务端的 CORS 配置，则在响应中包含这个 headers。 类型：字符串
Access-Control-Max-Age	当桶设置了 CORS 配置，服务端 CORS 配置中的 MaxAgeSeconds。 类型：整数
Access-Control-Allow-Methods	当桶设置了 CORS 配置，如果请求的 Access-Control-Request-Method 满足服务端的 CORS 配置，则在响应中包含这条 rule 中的 Methods。 类型：字符串 有效值：GET、PUT、HEAD、POST 、DELETE
Access-Control-Expose-Headers	当桶设置了 CORS 配置，服务端 CORS 配置中的 ExposeHeader。 类型：字符串

响应消息元素

该请求的响应中不带有响应元素。

错误响应消息

无特殊错误，所有错误已经包含在表 6-3 中。

请求示例 1

未携带获取 CORS 配置

```
HEAD / HTTP/1.1
User-Agent: curl/7.29.0
Host: examplebucket.obs.region.example.com
Accept: */*
Date: WED, 01 Jul 2015 02:30:25 GMT
Authorization: OBS H4IPJX0TQTHTHEBQQCEC:niCQCuGIZpETKIyx1dttxHZyYlk=
```

响应示例 1

```
HTTP/1.1 200 OK
Server: OBS
x-obs-request-id: BF260000016439C734E0788404623FA8
Content-Type: application/xml
x-obs-id-2: 32AAAQAAEAABAAAQAAEAABAAAQAAEAABCSxwLpq9Hzf3OnaXr+pI/OPLKdrtiQAF
Date: WED, 01 Jul 2015 02:30:25 GMT
x-obs-bucket-location: region
x-obs-version: 3.0
Content-Length: 0
```

请求示例 2

桶设置了 CORS 后，获取桶元数据和 CORS 配置

```
HEAD / HTTP/1.1
User-Agent: curl/7.29.0
Host: examplebucket.obs.region.example.com
Accept: */*
Date: WED, 01 Jul 2015 02:30:25 GMT
Authorization: OBS H4IPJX0TQTHTHEBQQCEC:niCQCuGIZpETKIyx1dttxHZyYlk=
Origin:www.example.com
Access-Control-Request-Headers:AllowedHeader_1
```

响应示例 2

```
HTTP/1.1 200 OK
Server: OBS
x-obs-request-id: BF260000016439C734E0788404623FA8
Content-Type: application/xml
x-obs-id-2: 32AAAQAAEAABAAAQAAEAABAAAQAAEAABCSxwLpq9Hzf3OnaXr+pI/OPLKdrtiQAF
Date: WED, 01 Jul 2015 02:30:25 GMT
x-obs-bucket-location: region
Access-Control-Allow-Origin: www.example.com
Access-Control-Allow-Methods: POST,GET,HEAD,PUT
Access-Control-Allow-Headers: AllowedHeader_1
Access-Control-Max-Age: 100
Access-Control-Expose-Headers: ExposeHeader_1
```

```
x-obs-version: 3.0
Content-Length: 0
```

5.1.5 获取桶区域位置

功能介绍

对桶拥有读权限的用户可以执行获取桶区域位置信息的操作。

请求消息样式

```
GET /?location HTTP/1.1
Host: bucketname.obs.region.example.com
Date: date
Authorization: authorization
```

请求消息参数

该请求消息中不带消息参数。

请求消息头

该请求使用公共消息头，具体参见表 3-3。

请求消息元素

该请求消息中不带消息元素。

响应消息样式

```
HTTP/1.1 status code
Date: date
Content-Type: type
Content-Length: length

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Location xmlns="http://obs.region.example.com/doc/2015-06-30/">region</Location>
```

响应消息头

该请求的响应消息使用公共消息头，具体请参考表 3-20。

响应消息元素

该响应中将桶的区域信息以消息元素的形式返回，元素的具体含义如表 5-9 所示。

表5-9 响应消息元素

元素名称	描述
Location	桶的区域位置信息。

元素名称	描述
	类型：字符串。

错误响应消息

无特殊错误，所有错误已经包含在表 6-3 中。

请求示例

```
GET /?location HTTP/1.1
User-Agent: curl/7.29.0
Host: examplebucket.obs.region.example.com
Accept: */*
Date: WED, 01 Jul 2015 02:30:25 GMT
Authorization: OBS H4IPJX0TQTHTHEBQQCEC:1DrmbCV+lhz3zV7uywlj7lrh0MY=
```

响应示例

```
HTTP/1.1 200 OK
Server: OBS
x-obs-request-id: BF260000016435D9F27CB2758E9B41A5
x-obs-id-2: 32AAAQAAEAABAAAQAAEAABAAAQAAEAABCSKWoJmaMyRXqofHgapbETDyI2LM9rUw
Content-Type: application/xml
Date: WED, 01 Jul 2015 02:30:25 GMT
Content-Length: 128

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Location xmlns="http://obs.region.example.com/doc/2015-06-30/">region</Location>
```

5.1.6 删除桶

功能介绍

删除桶操作用于删除用户指定的桶。只有桶的所有者或者拥有桶的删桶 policy 权限的用户可以执行删除桶的操作，要删除的桶必须是空桶。如果桶中有对象或者有多段任务则认为桶不为空，可以使用列举桶内对象和列举出多段上传任务接口来确认桶是否为空。

注：

如果删除桶时，服务端返回 5XX 错误或超时，系统需要时间进行桶信息一致性处理，在此期间桶的信息会不准确，过一段时间再查看桶是否删除成功，查询到桶，需要再次发送删除桶消息。

请求消息样式

```
DELETE / HTTP/1.1
Host: bucketname.obs.region.example.com
Date: date
Authorization: authorization
```

请求消息参数

该请求消息中不使用消息参数。

请求消息头

该请求使用公共的请求消息头，具体请参见表 3-3。

请求消息元素

该请求消息中不使用消息元素。

响应消息样式

```
HTTP/1.1 status code
Date: date
```

响应消息头

该请求的响应消息使用公共消息头，具体请参考表 3-20。

响应消息元素

该请求的响应消息中不带消息元素。

错误响应消息

无特殊错误，错误已经包含在表 6-3 中。

请求示例

```
DELETE / HTTP/1.1
User-Agent: curl/7.29.0
Host: examplebucket.obs.region.example.com
Accept: */*
Date: WED, 01 Jul 2015 02:31:25 GMT
Authorization: OBS H4IPJX0TQTHTHEBQQCEC:jZiAT8Vx4azWEvPRMWi0X5BpJMA=
```

响应示例

```
HTTP/1.1 204 No Content
Server: OBS
x-obs-request-id: BF260000016435DE6D67C35F9B969C47
x-obs-id-2: 32AAAQAAEAABKAAQAAEAABAAAQAAEAABCTukraCnXLsb7lEw4ZKjzDWWhzXdgme3
Date: WED, 01 Jul 2015 02:31:25 GMT
```

5.2 桶的高级配置

5.2.1 设置桶策略

功能介绍

该接口的实现使用 policy 子资源创建或者修改一个桶的策略。如果桶已经存在一个策略，那么当前请求中的策略将完全覆盖桶中现存的策略。

要使用该接口，使用者要求必须是桶的所有者或者具有设置桶策略的权限。

请求消息样式

```
PUT /?policy HTTP/1.1
Host: bucketname.obs.region.example.com
Date: date
Authorization: signatureValue
Policy written in JSON
```

请求消息参数

该请求消息中不使用消息参数。

请求消息头

该请求使用公共消息头，具体请参见表 3-3。

请求消息元素

请求消息体是一个符合 JSON 格式的字符串，包含了桶策略的信息。

响应消息样式

```
HTTP/1.1 status code
Date: date
Content-Length: length
```

响应消息头

该请求的响应消息使用公共消息头，具体请参考表 3-20。

响应消息元素

该请求的响应消息中不带有响应元素。

错误响应消息

无特殊错误，所有错误已经包含在表 6-3 中。

请求示例 1

向 OBS 租户授予权限

给租户 ID 为 783fc6652cf246c096ea836694f71855 的租户授权。

如何获取租户 ID 请参考 6.4 获取账号 ID 和用户 ID。

```
PUT /?policy HTTP/1.1
Host: examplebucket.obs.region.example.com
Date: WED, 01 Jul 2015 02:32:25 GMT
Authorization: OBS H4IPJX0TQTHTHEBQQCEC:jZiAT8Vx4azWEvPRMWi0X5BpJMA=

{
   "Statement": [
      {
         "Sid": "Stmt1375240018061",
         "Action": [
            "GetBucketLogging"
         ],
         "Effect": "Allow",
         "Resource": "logging.bucket",
         "Principal": {
            "ID": [
               "domain/783fc6652cf246c096ea836694f71855:user/*"
            ]
         }
      }
   ]
}
```

响应示例 1

```
HTTP/1.1 204 No Content
x-obs-request-id: 7B6DFC9BC71DD58B061285551605709
x-obs-id-2: N0I2REZDOUJDNzFERDU4QjA2MTI4NTU1MTYwNTcwOUFBQUFBQUFBYmJiYmJiYmJD
Date: WED, 01 Jul 2015 02:32:25 GMT
Content-Length: 0
Server: OBS
```

请求示例 2

向 OBS 用户授予权限

用户 ID 为 71f3901173514e6988115ea2c26d1999，用户所属租户 ID 为 783fc6652cf246c096ea836694f71855。

如何获取租户 ID 和用户 ID 请参考 6.4 获取账号 ID 和用户 ID。

```
PUT /?policy HTTP/1.1
Host: examplebucket.obs.region.example.com
Date: WED, 01 Jul 2015 02:33:28 GMT
Authorization: OBS H4IPJX0TQTHTHEBQQCEC:jZiAT8Vx4azWEvPRMWi0X5BpJMA=

{
   "Statement": [
```

```
      {
         "Sid": "Stmt1375240018062",
         "Action": [
            "PutBucketLogging"
         ],
         "Effect": "Allow",
         "Resource": "examplebucket",
         "Principal": {
            "ID": [

"domain/783fc6652cf246c096ea836694f71855:user/71f3901173514e6988115ea2c26d1999"
            ]
         }
      }
   ]
}
```

响应示例 2

```
HTTP/1.1 204 No Content
x-obs-request-id: 7B6DFC9BC71DD58B061285551605709
x-obs-id-2: N0I2REZDOUJDNzFERDU4QjA2MTI4NTU1MTYwNTcwOUFBQUFBQUFBYmJiYmJiYmJD
Date: WED, 01 Jul 2015 02:33:28 GMT
Content-Length: 0
Server: OBS
```

请求示例 3

拒绝除了某个指定 OBS 用户的其他用户执行所有操作

用户 ID 为 71f3901173514e6988115ea2c26d1999，用户所属租户 ID 为 783fc6652cf246c096ea836694f71855。

如何获取租户 ID 和用户 ID 请参考 6.4 获取账号 ID 和用户 ID。

```
PUT /?policy HTTP/1.1
Host: examplebucket.obs.region.example.com
Date: WED, 01 Jul 2015 02:34:34 GMT
Authorization: OBS H4IPJX0TQTHTHEBQQCEC:jZiAT8Vx4azWEvPRMWi0X5BpJMA=

{
   "Statement": [
      {
         "Effect": "Deny",
         "Action": ["*"],
         "Resource": [
            "examplebucket/*",
            "examplebucket"
         ],
         "NotPrincipal": {
            "ID": [

"domain/783fc6652cf246c096ea836694f71855:user/71f3901173514e6988115ea2c26d1999",
               "domain/783fc6652cf246c096ea836694f71855"
            ]
         }
```

```
        }
    ]
}
```

响应示例 3

```
HTTP/1.1 204 No Content
x-obs-request-id: A603000001604A7DFE4A4AF31E301891
x-obs-id-2: BKOvGmTlt6sda5X4G89PuMO4fabObGYmnpRGkaMba1LqPt0fCACEuCMllAObRK1n
Date: WED, 01 Jul 2015 02:34:34 GMT
Content-Length: 0
Server: OBS
```

请求示例 4

拒绝除了某个指定的域名和不带 referer 头域的外链请求以实现防盗链白名单

防盗链白名单：http://storage.example.com

```
PUT /?policy HTTP/1.1
Host: examplebucket.obs.region.example.com
Date: WED, 01 Jul 2015 02:34:34 GMT
Authorization: OBS H4IPJX0TQTHTHEBQQCEC:jZiAT8Vx4azWEvPRMWi0X5BpJMA=

{
    "Statement": [{
        "Effect": "Deny",
        "Action": [
        "GetObject",
        "GetObjectVersion"
        ],
        "Principal": {
            "ID": ["*"]
        },
        "Resource": ["examplebucket/*"],
        "Condition": {
            "StringNotLike": {
                "Referer": [
                "http://storage.example.com*",
                "${null}"
                ]
            }
        }
    }]
}
```

响应示例 4

```
HTTP/1.1 204 No Content
x-obs-request-id: A603000001604A7DFE4A4AF31E301891
x-obs-id-2: BKOvGmTlt6sda5X4G89PuMO4fabObGYmnpRGkaMba1LqPt0fCACEuCMllAObRK1n
Date: WED, 01 Jul 2015 02:34:34 GMT
Content-Length: 0
Server: OBS
```

5.2.2 获取桶策略

功能介绍

该接口的实现使用 policy 子资源来将指定桶的策略返回给客户端。

要使用该接口，使用者必须是桶的所有者或者具有获取桶策略的权限。

以下两种场景无法使用此接口获取桶策略，系统将返回“404 NoSuchBucketPolicy”的错误：

- 指定桶的策略不存在
- 指定桶的标准桶策略为私有且未设置高级桶策略

请求消息样式

```
GET /?policy HTTP/1.1
Host: bucketname.obs.region.example.com
Date: date
Authorization: authorization
```

请求消息参数

该请求消息中不使用消息参数。

请求消息头

该请求使用公共消息头，具体参见表 3-3。

请求消息元素

该请求消息中不使用消息元素。

响应消息样式

```
HTTP/1.1 status code
Content-Type: application/xml
Date: date
Policy Content
```

响应消息头

该请求的响应消息使用公共消息头，具体请参考表 3-20。

响应消息元素

响应消息体是一个 JSON 格式的桶策略字符串。

错误响应消息

无特殊错误，所有错误已经包含在表 6-3 中。

请求示例

```
GET /?policy HTTP/1.1
Host: examplebucket.obs.region.example.com
Date: WED, 01 Jul 2015 02:35:46 GMT
Authorization: OBS H4IPJX0TQTHTHEBQQCEC:jZiAT8Vx4azWEvPRMWi0X5BpJMA=
```

响应示例

```
HTTP/1.1 200 OK
x-obs-request-id: A603000001604A7DFE4A4AF31E301891
x-obs-id-2: BKOvGmTlt6sda5X4G89PuMO4fabObGYmnpRGkaMba1LqPt0fCACEuCMllAObRK1n
Date: WED, 01 Jul 2015 02:35:46 GMT
Content-Length: 509
Server: OBS

{
   "Statement":[
      {
         "Sid":"Stmt1375240018061",
         "Effect":"Allow",
         "Principal":{
            "ID":[

"domain/domainiddomainiddomainiddo006666:user/useriduseriduseriduseridus004001",
                "domain/domainiddomainiddomainiddo006667:user/*"
            ]
         },
         "Action":[
            "*"
         ],
         "Resource":[
            "examplebucket"
         ]
      }
   ]
}
```

5.2.3 删除桶策略

功能介绍

该接口的实现是通过使用 policy 子资源来删除一个指定桶上的策略。

要使用该接口，使用者必须是桶的所有者或者具有删除桶策略的权限。

无论桶的策略本身是否存在，删除成功后系统都直接返回“204 No Content”的结果。

请求消息样式

```
DELETE /?policy HTTP/1.1
Host: bucketname.obs.region.example.com
Date: date
Authorization: authorization
```

请求消息参数

该请求消息中不使用消息参数。

请求消息头

该请求使用公共消息头，具体参见表 3-3。

请求消息元素

该请求消息中不使用消息元素。

响应消息样式

```
HTTP/1.1 status code
Date: date
Content-Type: text/xml
Content-Length: length
```

响应消息头

该请求的响应消息使用公共消息头，具体请参考表 3-20。

响应消息元素

该请求的响应消息中不带有响应元素。

错误响应消息

无特殊错误，所有错误已经包含在表 6-3 中。

请求示例

```
DELETE /?policy HTTP/1.1
Host: examplebucket.obs.region.example.com
Date: WED, 01 Jul 2015 02:36:06 GMT
Authorization: OBS H4IPJX0TQTHTHEBQQCEC:jZiAT8Vx4azWEvPRMWi0X5BpJMA=
```

响应示例

```
HTTP/1.1 204 No Content
x-obs-request-id: 9006000001643AAAF70BF6152D71BE8A
x-obs-id-2: 32AAAQAAEAABSAAgAAEAABAAAQAAEAABCSB4oWmNX3gVGGLr1cRPWjOhffEbq1XV
Date: WED, 01 Jul 2015 02:36:06 GMT
Server: OBS
```

5.2.4 设置桶 ACL

功能介绍

OBS 支持对桶操作进行权限控制。默认情况下，只有桶的创建者才有该桶的读写权限。用户也可以设置其他的访问策略，比如对一个桶可以设置公共访问策略，允许所有人对其都有读权限。

OBS 用户在创建桶时可以设置权限控制策略，也可以通过 ACL 操作 API 接口对已存在的桶更改或者获取 ACL(access control list) 。

请求消息样式

```
PUT /?acl HTTP/1.1
Host: bucketname.obs.region.example.com
Date: date
Authorization: authorization
Content-Type: application/xml
Content-Length: length

<AccessControlPolicy>
   <Owner>
      <ID>ID</ID>
   </Owner>
   <AccessControlList>
      <Grant>
         <Grantee>
            <ID>domainId</ID>
         </Grantee>
         <Permission>permission</Permission>
         <Delivered>false</Delivered>
      </Grant>
   </AccessControlList>
</AccessControlPolicy>
```

请求消息参数

该操作请求不带消息参数。

请求消息头

使用者可以使用头域设置的方式来更改桶的 ACL，每一种头域设置的 ACL 都有一套自己预先定义好的被授权用户以及相应权限，通过头域设置的方式授予访问权限，使用者必须添加以下的头域并且指定取值。

表5-10 头域方式设置桶 ACL

名称	描述	是否必须
x-obs-acl	通过 canned ACL 的方式来设置桶的 ACL。 取值范围：private | public-read | public-read-write | public-read-delivered | public-read-write-delivered |	否

名称	描述	是否必须
	bucket-owner-full-control 类型：字符串	

请求消息元素

更改桶的 ACL 请求需要在消息元素中带上 ACL 信息，元素的具体含义如表 3-3 所示。

表5-11 附加请求消息元素

元素名称	描述	是否必选
Owner	桶的所有者信息，包含 ID。 类型：XML。	否
ID	被授权用户的租户 Id。 类型：字符串。	否
Grant	用于标记用户及用户的权限。 类型：XML。	否
Grantee	记录用户信息。 类型：XML。	否
Canned	向所有人授予权限。 取值范围：Everyone 类型：枚举类型。	否
Delivered	桶的 ACL 是否向桶内对象传递。 类型：布尔类型。默认 false。	否
Permission	授予的权限。 取值范围：READ | WRITE | FULL_CONTROL 类型：枚举类型。	否
AccessControlList	访问控制列表，包含 Grant、Grantee、Permission 三个元素。 类型：XML。	否

响应消息样式

```
HTTP/1.1 status code
Date: date
Content-Length: length
```

响应消息头

该请求的响应消息使用公共消息头，具体请参考表 3-20。

响应消息元素

该请求的响应消息中不带有响应元素。

错误响应消息

无特殊错误，所有错误已经包含在表 6-3 中。

请求示例

```
PUT /?acl HTTP/1.1
User-Agent: curl/7.29.0
Host: examplebucket.obs.region.example.com
Accept: */*
Date: WED, 01 Jul 2015 02:37:22 GMT
Authorization: OBS H4IPJX0TQTHTHEBQQCEC:iqSPeUBl66PwXDApxjRKk6hlcN4=
Content-Length: 727

<AccessControlPolicy xmlns="http://obs.example.com/doc/2015-06-30/">
  <Owner>
    <ID>b4bf1b36d9ca43d984fbcb9491b6fce9</ID>
  </Owner>
  <AccessControlList>
    <Grant>
      <Grantee>
        <ID>b4bf1b36d9ca43d984fbcb9491b6fce9</ID>
      </Grantee>
      <Permission>FULL CONTROL</Permission>
    </Grant>
    <Grant>
      <Grantee>
        <ID>783fc6652cf246c096ea836694f71855</ID>
      </Grantee>
      <Permission>READ</Permission>
      <Delivered>false</Delivered>
    </Grant>
    <Grant>
      <Grantee>
        <Canned>Everyone</Canned>
      </Grantee>
      <Permission>READ ACP</Permission>
    </Grant>
  </AccessControlList>
</AccessControlPolicy>
```

响应示例

```
HTTP/1.1 200 OK
Server: OBS
x-obs-request-id: BF2600000164361F2954B4D063164704
x-obs-id-2: 32AAAQAAEAABSAAgAAEAABAAAQAAEAABCT78HTIBuhe0FbtSptrb/akwELtwyPKs
Date: WED, 01 Jul 2015 02:37:22 GMT
Content-Length: 0
```

5.2.5 获取桶 ACL

功能介绍

用户执行获取桶 ACL 的操作，返回信息包含指定桶的权限控制列表信息。用户必须拥有对指定桶 READ_ACP 的权限或 FULL_CONTROL 权限，才能执行获取桶 ACL 的操作。

请求消息样式

```
GET /?acl HTTP/1.1
Host: bucketname.obs.region.example.com
Date: date
Authorization: authorization
```

请求消息参数

该请求消息中不使用消息参数。

请求消息头

该请求使用公共消息头，具体参见表 3-3。

请求消息元素

该请求消息中不使用消息元素。

响应消息样式

```
HTTP/1.1 status code
Date: date
Content-Length: length
Content-Type: application/xml

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<AccessControlPolicy xmlns="http://obs.region.example.com/doc/2015-06-30/">
   <Owner>
      <ID>id</ID>
   </Owner>
   <AccessControlList>
      <Grant>
         <Grantee>
            <ID>id</ID>
         </Grantee>
```

```
            <Permission>permission</Permission>
            <Delivered>false</Delivered>
        </Grant>
    </AccessControlList>
</AccessControlPolicy>
```

响应消息头

该请求的响应消息使用公共消息头，具体请参考表 3-20。

响应消息元素

该请求的响应中以消息元素的形式返回桶的 ACL 信息，元素的具体意义如表 5-12 所示。

表5-12 响应消息元素

元素	元素说明
Owner	桶的所有者信息。 类型：XML。
ID	用户所示租户的租户 Id。 类型：字符串。
AccessControlList	访问控制列表，记录了对该桶有访问权限的用户列表和这些用户具有的权限。 类型：XML。
Grant	用于标记用户及用户的权限。 类型：XML。
Grantee	记录用户信息。 类型：XML。
Canned	向所有人授予权限。 类型：枚举类型。其值只能是 Everyone。
Delivered	桶的 ACL 是否向桶内对象传递。 类型：布尔类型。
Permission	指定的用户对该桶所具有的操作权限。 类型：字符串。

错误响应消息

无特殊错误，所有错误已经包含在表 6-3 中。

请求示例

```
GET /?acl HTTP/1.1
User-Agent: curl/7.29.0
Host: examplebucket.obs.region.example.com
Accept: */*
Date: WED, 01 Jul 2015 02:39:28 GMT
Authorization: OBS H4IPJX0TQTHTHEBQQCEC:X7HtzGsIEkzJbd8vo1DRu30vVrs=
```

响应示例

```
HTTP/1.1 200 OK
Server: OBS
x-obs-request-id: BF260000016436B69D82F14E93528658
x-obs-id-2: 32AAAQAAEAABAAAQAAEAABAAAQAAEAABCSjTh8661+HF5y8uAnTOBIpNO133hji+
Content-Type: application/xml
Date: WED, 01 Jul 2015 02:39:28 GMT
Content-Length: 784

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<AccessControlPolicy xmlns="http://obs.example.com/doc/2015-06-30/">
  <Owner>
    <ID>b4bf1b36d9ca43d984fbcb9491b6fce9</ID>
  </Owner>
  <AccessControlList>
    <Grant>
      <Grantee>
        <ID>b4bf1b36d9ca43d984fbcb9491b6fce9</ID>
      </Grantee>
      <Permission>FULL CONTROL</Permission>
    </Grant>
    <Grant>
      <Grantee>
        <ID>783fc6652cf246c096ea836694f71855</ID>
      </Grantee>
      <Permission>READ</Permission>
      <Delivered>false</Delivered>
    </Grant>
    <Grant>
      <Grantee>
        <Canned>Everyone</Canned>
      </Grantee>
      <Permission>READ ACP</Permission>
    </Grant>
  </AccessControlList>
</AccessControlPolicy>
```

5.2.6 设置桶日志管理配置

功能介绍

创建桶时，默认是不生成桶的日志的，如果需要生成桶的日志，该桶需要打开日志配置管理的开关。桶日志功能开启后，桶的每次操作将会产生一条日志，并将多条日志打包成一个日志文件。日志文件存放位置需要在开启桶日志功能时指定，可以存放到

开启日志功能的桶中，也可以存放到其他你有权限的桶中，但需要和开启日志功能的桶在同一个 region 中。

由于日志文件是 OBS 产生，并且由 OBS 上传到存放日志的桶中，因此 OBS 需要获得委托授权，用于上传生成的日志文件，所以在配置桶日志管理前，需要先到统一身份认证服务生成一个对 OBS 服务的委托，并将委托名作为参数配置到桶上，并且在 xml 文件中<LoggingEnabled>标签下配置相应的日志管理功能。在为委托配置权限时只需设置目标桶的上传对象权限。

委托权限示例

```
{
   "Version": "1.1",
   "Statement": [
      {
         "Action": [
            "obs:object:PutObject"
         ],
         "Resource": [
            "OBS:*:*:object:mybucketlogs/*"
         ],
         "Effect": "Allow"
      }
   ]
}
```

关闭桶日志功能的方法是上传一个带有空的 BucketLoggingStatus 标签的 logging 文件。

请求消息样式

```
PUT /?logging HTTP/1.1
Host: bucketname.obs.region.example.com
Date: date
Authorization: signatureValue
<?xml version="1.0" encoding="UTF-8"?>
<BucketLoggingStatus>
  <Agency>agency-name</Agency>
  <LoggingEnabled>
    <TargetBucket>mybucketlogs</TargetBucket>
    <TargetPrefix>mybucket-access log-/</TargetPrefix>
    <TargetGrants>
      <Grant>
        <Grantee>
          <ID>domainID</ID>
        </Grantee>
        <Permission>READ</Permission>
      </Grant>
    </TargetGrants>
  </LoggingEnabled>
</BucketLoggingStatus>
```

请求消息参数

该请求消息中不使用消息参数。

请求消息头

该请求使用公共消息头，具体请参见表 3-3。

请求消息元素

表5-13 请求消息元素表

名字	描述	是否必选
BucketLoggingStatus	日志状态信息的容器。 类型：容器	是
Agency	目标桶 Owner 通过统一身份认证服务创建的对 OBS 服务的委托的名称。 类型：字符串	设置 logging 时必选。关闭 logging 时勿选。
LoggingEnabled	该元素起到对日志配置管理的使能作用（呈现此元素则打开日志配置，否则关闭配置）。在此元素下，可加入具体的日志配置信息。 类型：容器	设置 logging 时必选。关闭 logging 时勿选。
Grant	是被授权者及其权限的容器。用于描述谁有什么权限来访问产生的日志文件。 类型：容器	否
Grantee	作为被授权 logging 权限用户的容器。 类型：容器	否
ID	被授权者的租户 ID，全局唯一标识。 类型：字符串	否
Permission	产生的日志文件对被授权者的具体权限。 类型：字符串 权限有效值：FULL_CONTROL | READ | WRITE	否
TargetBucket	在生成日志时，配置日志桶的所有者可以指定一个桶用于存放产生的日志文	设置 logging 时必选。关闭 logging 时勿选。

名字	描述	是否必选
	件。需要保证配置日志文件的桶 owner 对存放日志文件的桶有 FULL_CONTROL 权限。支持多个桶生成的日志放在同一个目标桶中，如果这样做，就需要指定不同的 TargetPrefix 以达到为来自不同源桶的日志分类的目的。 类型：字符串	
TargetPrefix	通过该元素指定一个前缀，所有生成的日志对象的对象名都以此元素的内容为前缀。 类型：字符串	设置 logging 时必选。关闭 logging 时勿选。
TargetGrants	授权信息的容器。 类型：容器	否

存储访问日志的 object 命名规则

```
<TargetPrefix>YYYY-mm-DD-HH-MM-SS-<UniqueString>
```

- <TargetPrefix>为用户指定的目标前缀。
- YYYY-mm-DD-HH-MM-SS 为日志生成的日期与时间，各字段依次表示年、月、日、时、分、秒。
- <UniqueString>为 OBS 自动生成的字符串。

一个实际用于存储 OBS 访问日志的 object 名称实例如下：

```
bucket-log2015-06-29-12-22-07-N7MXLAF1BDG7MPDV
```

- "bucket-log"为用户指定的目标前缀。
- "2015-06-29-12-22-07"为日志生成的日期与时间。
- "N7MXLAF1BDG7MPDV"为 OBS 自动生成的字符串。

桶访问日志格式

以下所示为在目标桶生成的桶访问日志文件记录：

```
787f2f92b20943998a4fe2ab75eb09b8 bucket [13/Aug/2015:01:43:42 +0000] xx.xx.xx.xx
787f2f92b20943998a4fe2ab75eb09b8 281599BACAD9376ECE141B842B94535B
REST.GET.BUCKET.LOCATION - "GET /bucket?location HTTP/1.1" 200 - 211 - 6 6 "-"
"HttpClient" - -
```

每个桶访问日志都包含如下信息：

表5-14 Bucket Logging 格式

名称	示例	含义
BucketOwner	787f2f92b20943998a4fe2ab75eb09b8	桶的 ownerId
Bucket	bucket	桶名
Time	[13/Aug/2015:01:43:42 +0000]	请求时间戳
Remote IP	xx.xx.xx.xx	请求 IP
Requester	787f2f92b20943998a4fe2ab75eb09b8	请求者 ID
RequestID	281599BACAD9376ECE141B842B94535B	请求 ID
Operation	REST.GET.BUCKET.LOCATION	操作名称
Key	-	对象名
Request-URI	GET /bucket?location HTTP/1.1	请求 URI
HTTPStatus	200	返回码
ErrorCode	-	错误码
BytesSent	211	HTTP 响应的字节大小
ObjectSize	-	对象大小
TotalTime	6	服务端处理时间
Turn-AroundTime	6	总请求时间
Referer	-	请求的 referrer 头域
User-Agent	HttpClient	请求的 user-agent 头域
VersionID	-	请求中带的 versionId
STSLogUrn	-	联邦认证及委托授权信息

响应消息样式

```
HTTP/1.1 status code
Date: date
Content-Length: length
```

响应消息头

该请求的响应消息使用公共消息头，具体请参考表 3-20。

响应消息元素

该请求的响应消息中不带有响应元素。

错误响应消息

无特殊错误，所有错误已经包含在表 6-3 中。

请求示例

```
PUT /?logging HTTP/1.1
User-Agent: curl/7.29.0
Host: examplebucket.obs.region.example.com
Accept: */*
Date: WED, 01 Jul 2015 02:40:06 GMT
Authorization: OBS H4IPJX0TQTHTHEBQQCEC:mCOjER/L4ZZUY9qr6AOnkEiwvVk=
Content-Length: 528

<?xml version="1.0" encoding="UTF-8"?>
<BucketLoggingStatus>
  <Agency>agencyGrantPutLogging</Agency>
  <LoggingEnabled>
    <TargetBucket>log-bucket</TargetBucket>
    <TargetPrefix>mybucket-access log-/</TargetPrefix>
    <TargetGrants>
      <Grant>
        <Grantee>
          <ID>783fc6652cf246c096ea836694f71855</ID>
        </Grantee>
        <Permission>READ</Permission>
      </Grant>
    </TargetGrants>
  </LoggingEnabled>
</BucketLoggingStatus>
```

响应示例

```
HTTP/1.1 200 OK
Server: OBS
x-obs-request-id: BF26000001643663CE53B6AF31C619FD
x-obs-id-2: 32AAAQAAEAABSAAkpAIAABAAAQAAEAABCT9CjuOx8cETSRbqkm35s1dL/tLhRNdZ
Date: WED, 01 Jul 2015 02:40:06 GMT
Content-Length: 0
```

5.2.7 获取桶日志管理配置

功能介绍

该接口的目的是查询当前桶的日志管理配置情况。其实现是通过使用 http 的 get 方法再加入 logging 子资源来返回当前桶的日志配置情况。

要使用该接口，使用者必须是桶的所有者或者是被桶策略授权 GetBucketLogging 权限的用户。

请求消息样式

```
GET /?logging HTTP/1.1
Host: bucketname.obs.region.example.com
Date: date
Authorization: authorization
```

请求消息参数

该请求消息中不使用消息参数。

请求消息头

该请求使用公共消息头，具体参见表 3-3。

请求消息元素

该请求消息中不使用消息元素。

响应消息样式

```
HTTP/1.1 status code
Content-Type: application/xml
Date: date
Content-Length: length

<?xml version="1.0" encoding="UTF-8"?>
<BucketLoggingStatus xmlns="http://obs.region.example.com/doc/2015-06-30/">
<Agency>agency-name</Agency>
<LoggingEnabled>
   <TargetBucket>bucketName</TargetBucket>
   <TargetPrefix>prefix</TargetPrefix>
      <TargetGrants>
         <Grant>
            <Grantee>
               <ID>id</ID>
            </Grantee>
            <Permission>permission</Permission>
         </Grant>
      </TargetGrants>
   </LoggingEnabled>
</BucketLoggingStatus>
```

响应消息头

该请求的响应消息使用公共消息头，具体请参考表 3-20。

响应消息元素

该请求的响应中以消息元素的形式返回桶的日志信息，元素的具体意义如表 5-15 所示。

表5-15 响应消息元素

名字	描述
BucketLoggingStatus	logging 状态信息的容器。 类型：容器
Agency	产生 logging 日志桶 Owner 创建委托 OBS 上传 logging 日志的委托名。 类型：字符串
LoggingEnabled	用于 logging 信息的容器。并且该元素起到对 logging 配置管理的使能作用（呈现此元素则打开 logging 配置，否则关闭）。 类型：容器
Grant	是被授权者及其权限的容器。 类型：容器
Grantee	作为被授权 logging 权限用户的容器。 类型：容器
ID	被授权用户的 Domain Id，全局唯一标识。 类型：字符串
Permission	对于一个桶的 logging 权限来说，owner 在创桶时将自动获得对源桶的 FULL_CONTROL 权限。不同的权限决定了对不同日志的访问限制。 类型：字符串 权限有效值：FULL_CONTROL | READ | WRITE
TargetBucket	在生成日志时，源桶的 owner 可以指定一个目标桶，将生成的所有日志放到该桶中。在 OBS 系统中，支持多个源桶生成的日志放在同一个目标桶中，如果这样做，就需要指定不同的 TargetPrefix 以达到为来自不同源桶的日志分类的目的。 类型：字符串
TargetPrefix	通过该元素可以指定一个前缀给一类日志生成的对象。

名字	描述
	类型：字符串
TargetGrants	授权信息的容器。 类型：容器

错误响应消息

无特殊错误，所有错误已经包含在表 6-3 中。

请求示例

```
GET /?logging HTTP/1.1
User-Agent: curl/7.29.0
Host: examplebucket.obs.region.example.com
Accept: */*
Date: WED, 01 Jul 2015 02:42:46 GMT
Authorization: OBS H4IPJX0TQTHTHEBQQCEC:hUk+jTnR07hcKwJh4ousF2E1U3E=
```

响应示例

```
HTTP/1.1 200 OK
Server: OBS
x-obs-request-id: BF260000016436B8EEE7FBA2AA3335E3
x-obs-id-2: 32AAAQAAEAABAAAQAAEAABAAAQAAEAABCShuQJoWFpS77C8bOv1mqURv0UY+0ejx
Content-Type: application/xml
Date: WED, 01 Jul 2015 02:42:46 GMT
Content-Length: 429

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<BucketLoggingStatus xmlns="http://obs.example.com/doc/2015-06-30/">
  <Agency>agency-name</Agency>
  <LoggingEnabled>
    <TargetBucket>log-bucket</TargetBucket>
    <TargetPrefix>mybucket-access log-/</TargetPrefix>
    <TargetGrants>
      <Grant>
        <Grantee>
          <ID>b4bf1b36d9ca43d984fbcb9491b6fce9</ID>
        </Grantee>
        <Permission>READ</Permission>
      </Grant>
    </TargetGrants>
  </LoggingEnabled>
</BucketLoggingStatus>
```

5.2.8 设置桶的生命周期配置

功能介绍

OBS 系统支持指定规则来实现定时删除桶中对象，这就是生命周期配置。典型的应用场景如：

- 周期性上传的日志文件，可能只需要保留一个星期或一个月，到期后要删除它们。
- 某些文档在一段时间内经常访问，但是超过一定时间后就可能不会再访问了。这种文档您可能会先选择归档，然后在一定时间后删除。

本接口实现为桶创建或更新生命周期配置信息。

说明

对象生命周期到期以后，对象将会永久删除，无法恢复。

要正确执行此操作，需要确保执行者有 PutLifecycleConfiguration 权限。默认情况下只有桶的所有者可以执行此操作，也可以通过设置桶策略或用户策略授权给其他用户。

生命周期配置实现了定时删除对象的功能，所以如果想要阻止用户删除，以下几项操作的权限都应该被禁止：

- DeleteObject
- DeleteObjectVersion
- PutLifecycleConfiguration

如果想要阻止用户管理桶的生命周期配置，应该禁止 PutLifecycleConfiguration 权限。

请求消息样式

```
PUT /?lifecycle HTTP/1.1
Host: bucketname.obs.region.example.com
Content-Length: length
Date: date
Authorization: authorization
Content-MD5: MD5
<?xml version="1.0" encoding="UTF-8"?>
<LifecycleConfiguration>
   <Rule>
      <ID>id</ID>
      <Prefix>prefix</Prefix>
      <Status>status</Status>
      <Expiration>
         <Days>days</Days>
      </Expiration>
      <NoncurrentVersionExpiration>
         <NoncurrentDays>days</NoncurrentDays>
      </NoncurrentVersionExpiration>
   </Rule>
</LifecycleConfiguration>
```

请求消息参数

该请求消息中不使用消息参数。

请求消息头

该请求使用的消息头如下表 5-16 所示。

表5-16 请求消息头

消息头名称	描述	是否必选
Content-MD5	按照 RFC 1864 标准计算出消息体的 MD5 摘要字符串，即消息体 128-bit MD5 值经过 base64 编码后得到的字符串。 类型：字符串 示例：n58IG6hfM7vqI4K0vnWpog==	是

请求消息元素

在此请求中，需要在请求的消息体中配置桶的生命周期配置信息。配置信息以 XML 格式上传，具体的配置元素如表 5-17 描述。

- 如果桶的多版本是 Enabled 或者 Suspended，那么可以设置 NoncurrentVersionExpiration 来控制对象的历史版本的生命周期。一个历史版本的生命周期，取决于它成为历史版本的时刻（即被新版本覆盖的那个时刻）和 NoncurrentDays。例如 NoncurrentDays 配置为 1 的话，表示当一个版本成为历史版本之后，再过 1 天才能删除。对象 A 的版本 V1 创建于 1 号，5 号的时候又上传新的版本 V2，此时 V1 成为历史版本，那么再过 1 天，7 号的 0 点，V1 就过期了。（备注：对象过期后被删除的时间的时间可能会有一定的延迟，一般不超过 48 小时。）
- 如果桶的多版本是 Enabled 或者 Suspended，且最新版本对象满足 Expiration 规则时的处理：
  - 桶当前的多版本状态为 Enabled：

    如果对象的最新版本不是 deletemarker，则该对象会产生一个新的 deletemarker；

    如果最新版本是 deletemarker，且该对象只有这一个版本，则这个版本会被删除；

    如果最新版本是 deletemarker，且对象还有其他版本，则该对象的所有版本维持不变，没有新增和删除，也不会被修改（即无任何变化）。
  - 桶当前的多版本状态为 Suspended：

    如果对象的最新版本不是 deletemarker，且版本不是 null 版本，则会产生一个新的 null 版本的 deletemarker；

    如果对象的最新版本不是 deletemarker，且版本是 null 版本，则这个 null 版本会被新产生的 null 版本的 deletemarker 覆盖；

如果最新版本是 deletemarker，且该对象只有这一个版本，则这个版本会被删除；

如果最新版本是 deletemarker，且对象还有其他版本，则该对象的所有版本维持不变，没有新增和删除，也不会被修改（即无任何变化）。

表5-17 生命周期配置元素

名称	描述	是否必选
Date	指定 OBS 对该日期之前的对象执行生命周期规则。日期格式必须为 ISO8601 的格式，并且为 UTC 的零点。 例如：2018-01-01T00:00:00.000Z，表示将最后修改时间早于 2018-01-01T00:00:00.000Z 的对象删除，等于或晚于这个时间的对象不会被删除。 类型：字符串 父节点：Expiration	如果没有 Days 元素，则必选
Days	指定生命周期规则在对象最后更新过后多少天生效（仅针对对象的最新版本）。 类型：正整数 父节点：Expiration	如果没有 Date 元素，则必选
Expiration	生命周期配置中表示过期时间的 Container（仅针对对象的最新版本）。 类型：XML 子节点：Date 或 Days 父节点：Rule	是
ID	一条 Rule 的标识，由不超过 255 个字符的字符串组成。 类型：字符串 父节点：Rule	否
LifecycleConfiguration	生命周期配置 Rule 的 Container。可以配置多条 Rule，但需保证整个配置消息体总大小不超过 20KB。 类型：XML 子节点：Rule 父节点：无	是
NoncurrentDays	表示对象在成为历史版本之后第几天时规则生效（仅针对历史版本）。 类型：正整数 父节点：NoncurrentVersionExpiration	如果有 NoncurrentVersionExpiration 元素，则必选
NoncurrentVersionEx	生命周期配置中表示历史版本过期时间的	否

名称	描述	是否必选
piration	Container。您可以将该动作设置在已启用多版本（或暂停）的桶，来让系统删除对象的满足特定生命周期的历史版本（仅针对历史版本）。 类型：XML 子节点：NoncurrentDays 父节点：Rule	
Prefix	对象名前缀，用以标识哪些对象可以匹配到当前这条 Rule。 类型：字符串 父节点：Rule	是
Rule	具体某一条生命周期配置的 Container。 类型：容器 父节点：LifecycleConfiguration	是
Status	标识当前这条 Rule 是否启用。 类型：字符串 父节点：Rule 有效值：Enabled，Disabled	是

响应消息样式

```
HTTP/1.1 status_code
Date: date
Content-Length: length
```

响应消息头

该请求的响应消息使用公共消息头，具体请参考表 3-20。

响应消息元素

该请求的响应消息不带消息元素。

错误响应消息

无特殊错误，所有错误已经包含在表 6-3 中。

请求示例

```
PUT /?lifecycle HTTP/1.1
User-Agent: curl/7.29.0
Host: examplebucket.obs.region.example.com
```

```
Accept: */*
Date: WED, 01 Jul 2015 03:05:34 GMT
Authorization: OBS H4IPJX0TQTHTHEBQQCEC:DpSAlmLX/BTdjxU5HOEwflhM0WI=
Content-MD5: ujCZn5p3fmczNiQQxdsGaQ==
Content-Length: 919

<?xml version="1.0" encoding="utf-8"?>
<LifecycleConfiguration>
  <Rule>
    <ID>delete-2-days</ID>
    <Prefix>test/</Prefix>
    <Status>Enabled</Status>
    <Expiration>
      <Days>2</Days>
    </Expiration>
    <NoncurrentVersionExpiration>
      <NoncurrentDays>5</NoncurrentDays>
    </NoncurrentVersionExpiration>
  </Rule>
</LifecycleConfiguration>
```

响应示例

```
HTTP/1.1 200 OK
Server: OBS
x-obs-request-id: BF26000001643670AC06E7B9A7767921
x-obs-id-2: 32AAAQAAEAABSAAgAAEAABAAAQAAEAABCSvK6z8HV6nrJh49gsB5vqzpgtohkiFm
Date: WED, 01 Jul 2015 03:05:34 GMT
Content-Length: 0
```

5.2.9 获取桶的生命周期配置

功能介绍

获取该桶设置的生命周期配置信息。

要正确执行此操作，需要确保执行者有 GetLifecycleConfiguration 执行权限。默认情况下只有桶的所有者可以执行此操作，也可以通过设置桶策略或用户策略授权给其他用户。

请求消息样式

```
GET /?lifecycle HTTP/1.1
Host: bucketname.obs.region.example.com
Date: date
Authorization: authorization
```

请求消息参数

该请求消息中不使用消息参数。

请求消息头

该请求使用公共消息头，具体参见表 3-3。

请求消息元素

该请求消息中不使用消息元素。

响应消息样式

```
HTTP/1.1 status_code
Date: date
Content-Type: application/xml
Date: date
Content-Length: length

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<LifecycleConfiguration xmlns="http://obs.region.example.com/doc/2015-06-30/">
   <Rule>
      <ID>id</ID>
      <Prefix>prefix</Prefix>
      <Status>status</Status>
      <Expiration>
         <Date>date</Date>
      </Expiration>
      <NoncurrentVersionExpiration>
         <NoncurrentDays>days</NoncurrentDays>
      </NoncurrentVersionExpiration>
   </Rule>
</LifecycleConfiguration>
```

响应消息头

该请求的响应消息使用公共消息头，具体请参考表 3-20。

响应消息元素

在此请求返回的响应消息体中包含的配置元素如下表 5-18 描述。

表5-18 生命周期配置元素

名称	描述
Date	指定 OBS 对该日期之前的对象执行生命周期规则。日期格式必须为 ISO8601 的格式，并且为 UTC 的零点。 例如：2018-01-01T00:00:00.000Z，表示将最后修改时间早于 2018-01-01T00:00:00.000Z 的对象删除，等于或晚于这个时间的对象不会被删除。 类型：字符串 父节点：Expiration
Days	指定在对象最后修改时间的多少天后执行生命周期规则（仅针对对象的最新版本）。 类型：正整数 父节点：Expiration

名称	描述
Expiration	生命周期配置中表示过期时间的 Container。 类型：XML 子节点：Date 或 Days 父节点：Rule
ID	一条 Rule 的标识，由不超过 255 个字符的字符串组成。 类型：字符串 父节点：Rule
LifecycleConfiguration	生命周期配置 Rule 的 Container。可以配置多条 Rule，但需保证整个配置消息体总大小不超过 20KB。 类型：XML 子节点：Rule 父节点：无
NoncurrentDays	表示对象在成为历史版本之后第几天时规则生效。 类型：正整数 父节点：NoncurrentVersionExpiration
NoncurrentVersionExpiration	生命周期配置中表示历史版本过期时间的 Container。您可以将该动作设置在已启用多版本（或暂停）的桶，来让系统删除对象的满足特定生命周期的历史版本。 类型：XML 子节点：NoncurrentDays 父节点：Rule
Prefix	对象名前缀，用以标识哪些对象可以匹配到当前这条 Rule。 类型：字符串 父节点：Rule
Rule	具体某一条生命周期配置的 Container。 类型：容器 父节点：LifecycleConfiguration
Status	标识当前这条 Rule 是否启用。 类型：字符串 父节点：Rule 有效值：Enabled，Disabled

错误响应消息

此请求可能的特殊错误如下表 5-19 描述。

表5-19 特殊错误

错误码	描述	HTTP 状态码
NoSuchLifecycleConfiguration	桶的生命周期配置不存在	404 Not Found

其余错误已经包含在表 6-3 中。

请求示例

```
GET /?lifecycle HTTP/1.1
User-Agent: curl/7.29.0
Host: examplebucket.obs.region.example.com
Accept: */*
Date: WED, 01 Jul 2015 03:06:56 GMT
Authorization: OBS H4IPJX0TQTHTHEBQQCEC:/Nof9FCNANfzIXDS0NDp1IfDu8I=
```

响应示例

```
HTTP/1.1 200 OK
Server: OBS
x-obs-request-id: BF260000016436BA5684FF5A10370EDB
x-obs-id-2: 32AAAQAAEAABAAAQAAEAABAAAQAAEAABCSEMKZSIeboCA1eAukgYOOAd7oX3ZONn
Content-Type: application/xml
Date: WED, 01 Jul 2015 03:06:56 GMT
Content-Length: 919

<?xml version="1.0" encoding="utf-8"?>
<LifecycleConfiguration>
  <Rule>
    <ID>delete-2-days</ID>
    <Prefix>test/</Prefix>
    <Status>Enabled</Status>
    <Expiration>
      <Days>2</Days>
    </Expiration>
    <NoncurrentVersionExpiration>
      <NoncurrentDays>5</NoncurrentDays>
    </NoncurrentVersionExpiration>
  </Rule>
</LifecycleConfiguration>
```

5.2.10 删除桶的生命周期配置

功能介绍

删除指定桶的生命周期配置信息。删除后桶中的对象不会过期，OBS 不会自动删除桶中对象。

要正确执行此操作，需要确保执行者有 PutLifecycleConfiguration 权限。默认情况下只有桶的所有者可以执行此操作，也可以通过设置桶策略或用户策略授权给其他用户。

请求消息样式

```
DELETE /?lifecycle HTTP/1.1
Host: bucketname.obs.region.example.com
Date: date
Authorization: Authorization
```

请求消息参数

该请求消息中不使用消息参数。

请求消息头

该请求使用公共消息头，具体参见表 3-3。

请求消息元素

该请求消息中不使用消息元素。

响应消息样式

```
HTTP/1.1 status code
Date: date
Content-Type: text/xml
Date: date
```

响应消息头

该请求的响应消息使用公共消息头，具体请参考表 3-20。

响应消息元素

该请求的响应消息中不带消息元素。

错误响应消息

无特殊错误，所有错误已经包含在表 6-3 中。

请求示例

```
DELETE /?lifecycle HTTP/1.1
User-Agent: curl/7.29.0
```

```
Host: examplebucket.obs.region.example.com
Accept: */*
Date: WED, 01 Jul 2015 03:12:22 GMT
Authorization: OBS H4IPJX0TQTHTHEBQQCEC:5DGAS7SBbMC1YTC4tNXY57Zl2Fo=
```

响应示例

```
HTTP/1.1 204 No Content
Server: OBS
x-obs-request-id: BF2600000016436C2550A1EEA97614A98
x-obs-id-2: 32AAAQAAEAABSAAgAAEAABAAAQAAEAABCSB7A0KZEBOCutgcfZvaGVthTGOJSuyk
Date: WED, 01 Jul 2015 03:12:22 GMT
```

5.2.11 设置桶的多版本状态

功能介绍

多版本功能可在用户意外覆盖或删除对象的情况下提供一种恢复手段。用户可以使用多版本功能来保存、检索和还原对象的各个版本，这样用户能够从意外操作或应用程序故障中轻松恢复数据。多版本功能还可用于数据保留和存档。

默认情况下，桶没有设置多版本功能。

本接口设置桶的多版本状态，用来开启或暂停桶的多版本功能。

设置桶的多版本状态为 Enabled，开启桶的多版本功能：

- 上传对象时，系统为每一个对象创建一个唯一版本号，上传同名的对象将不再覆盖旧的对象，而是创建新的不同版本号的同名对象
- 可以指定版本号下载对象，不指定版本号默认下载最新对象；
- 删除对象时可以指定版本号删除，不带版本号删除对象仅产生一个带唯一版本号的删除标记，并不删除对象；
- 列出桶内对象列表时默认列出最新对象列表，可以指定列出桶内所有版本对象列表；
- 除了删除标记外，每个版本的对象存储均需计费（不包括对象元数据）。

设置桶的多版本状态为 Suspended，暂停桶的多版本功能：

- 旧的版本数据继续保留 ；
- 上传对象时创建对象的版本号为 null，上传同名的对象将覆盖原有同名的版本号为 null 的对象；
- 可以指定版本号下载对象，不指定版本号默认下载最新对象；
- 删除对象时可以指定版本号删除，不带版本号删除对象将产生一个版本号为 null 的删除标记，并删除版本号为 null 的对象；
- 除了删除标记外，每个版本的对象存储均需计费（不包括对象元数据）。

只有桶的所有者可以设置桶的多版本状态。

请求消息样式

```
PUT /?versioning HTTP/1.1
Host: bucketname.obs.region.example.com
Date: date
Authorization: authorization
Content-Length: length

<VersioningConfiguration>
   <Status>status</Status>
</VersioningConfiguration>
```

请求消息参数

该请求消息中不使用消息参数。

请求消息头

该请求使用公共消息头，具体参见表 3-3。

请求消息元素

在此请求中，需要在请求的消息体中配置桶的多版本状态，配置信息以 XML 格式上传。具体的配置元素如表 5-20 描述。

表5-20 桶的多版本状态配置元素

名称	描述	是否必选
VersioningConfiguration	多版本配置的根节点。 父节点：无	是
Status	标识桶的多版本状态。 类型：枚举值 父节点：VersioningConfiguration 有效值：Enabled，Suspended	是

响应消息样式

```
HTTP/1.1 status code
Date: date

Content-Length: length
```

响应消息头

该请求的响应消息使用公共消息头，具体请参考表 3-20。

响应消息元素

该请求的响应消息中不带消息元素。

错误响应消息

无特殊错误，所有错误已经包含在表 6-3 中。

请求示例

```
PUT /?versioning HTTP/1.1
User-Agent: curl/7.29.0
Host: examplebucket.obs.region.example.com
Accept: */*
Date: WED, 01 Jul 2015 03:14:18 GMT
Authorization: OBS H4IPJX0TQTHTHEBQQCEC:sc2PM13Wlfcoc/YZLK0MwsI2Zpo=
Content-Length: 89

<VersioningConfiguration>
   <Status>Enabled</Status>
</VersioningConfiguration>
```

响应示例

```
HTTP/1.1 200 OK
Server: OBS
x-obs-request-id: BF26000001643672B973EEBC5FBBF909
x-obs-id-2: 32AAAQAAEAABSAAgAAEAABAAAQAAEAABCSH6rPRHjQCa62fcNpCCPs7+1Aq/hKzE
Date: Date: WED, 01 Jul 2015 03:14:18 GMT
Content-Length: 0
```

5.2.12 获取桶的多版本状态

功能介绍

桶的所有者可以获取指定桶的多版本状态。

如果从未设置桶的多版本状态，则此操作不会返回桶的多版本状态。

请求消息样式

```
GET /?versioning HTTP/1.1
Host: bucketname.obs.region.example.com
Date: date
Authorization: authorization
```

请求消息参数

该请求消息中不使用消息参数。

请求消息头

该请求使用公共消息头，具体参见表 3-3。

请求消息元素

该请求消息中不使用消息元素。

响应消息样式

```
HTTP/1.1 status_code
Date: date
Content-Type: type
Content-Length: length

<VersioningConfiguration xmlns="http://obs.region.example.com/doc/2015-06-30/">
   <Status>status</Status>
</VersioningConfiguration>
```

响应消息头

该请求的响应消息使用公共消息头，具体请参考表 3-20。

响应消息元素

该请求的响应中以消息元素的形式返回桶的多版本状态，元素的具体意义如表 5-21 所示。

表5-21 响应消息元素

名字	描述
VersioningConfiguration	多版本状态信息的元素。 类型：元素
Status	标识桶的多版本状态。 类型：枚举值 有效值：Enabled，Suspended

错误响应消息

无特殊错误，所有错误已经包含在表 6-3 中。

请求示例

```
GET /?versioning HTTP/1.1
User-Agent: curl/7.29.0
Host: examplebucket.obs.region.example.com
Accept: */*
Date: WED, 01 Jul 2015 03:15:20 GMT
Authorization: OBS H4IPJX0TQTHTHEBQQCEC:4N5qQIoluLO9xMY0m+8lIn/UWXM=
```

响应示例

```
HTTP/1.1 200 OK
Server: OBS
x-obs-request-id: BF2600000016436BBA4930622B4FC9F17
x-obs-id-2: 32AAAQAAEAABAAAQAAEAABAAAQAAEAABCSQIrNJ5/Ag6EPN8DAwWlPWgBc/xfBnx
Content-Type: application/xml
Date: WED, 01 Jul 2015 03:15:20 GMT
Content-Length: 180

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<VersioningConfiguration xmlns="http://obs.example.com/doc/2015-06-30/">
  <Status>Enabled</Status>
</VersioningConfiguration>
```

5.2.13 设置桶配额

功能介绍

桶空间配额值必须为非负整数，单位为 Byte（字节），能设的最大值为 2^{63}-1。桶的默认配额为 0，表示没有限制桶配额。

说明

1. 桶配额设置后，如果想取消配额限制，可以把配额设置为 0。
2. 由于桶配额的校验依赖于桶存量，而桶存量是后台计算，因此桶配额可能不会及时生效，存在滞后性。可能会出现桶存量超出配额或者删除数据后存量未能及时回落的情况。
3. 桶存量查询接口请参见 5.2.15 获取桶存量信息。
4. 桶存量超出配额后再上传对象，会返回 HTTP 状态码 403 Forbidden，错误码 InsufficientStorageSpace。请扩大配额，或取消配额限制（设置为 0），或删除不需要的对象。

请求消息样式

```
PUT /?quota HTTP/1.1
Host: bucketname.obs.region.example.com
Date: date
Content-Length: length
Authorization: authorization

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Quota xmlns="http://obs.region.example.com/doc/2015-06-30/">
   <StorageQuota>value</StorageQuota>
</Quota>
```

请求消息参数

该请求在请求消息中没有带有参数。

请求消息头

该请求没有特殊的请求消息头，公共部分参见表 3-3。

请求消息元素

该操作需要附加请求消息元素来指定桶的空间配额，具体见表 5-22。

表5-22 附加请求消息元素

元素名称	描述	是否必选
StorageQuota	指定桶空间配额值单位为字节。 类型：整型。	是

响应消息样式

```
HTTP/1.1 status code
Date: date
Content-Length: length
```

响应消息头

该请求的响应消息使用公共消息头，具体请参考表 3-20。

响应消息元素

该请求的响应中不带有响应元素。

错误响应消息

无特殊错误，所有错误已经包含在表 6-3 中。

请求示例

```
PUT /?quota HTTP/1.1
User-Agent: curl/7.29.0
Host: examplebucket.obs.region.example.com
Accept: */*
Date: WED, 01 Jul 2015 03:24:37 GMT
Authorization: OBS H4IPJX0TQTHTHEBQQCEC:k/rbwnYaqYf0Ae6F0M3OJQ0dmI8=
Content-Length: 106

<Quota xmlns="http://obs.region.example.com/doc/2015-06-30/">
  <StorageQuota>10240000</StorageQuota>
</Quota>
```

响应示例

```
HTTP/1.1 100 Continue
HTTP/1.1 200 OK
Server: OBS
x-obs-request-id: BF260000016435E09A2BCA388688AA08
x-obs-id-2: 32AAAQAAEAABSAAgAAEAABAAAQAAEAABCSHbmBecv7ohDSvqaRObpxzgzJ9+l8xT
```

```
Date: WED, 01 Jul 2015 03:24:37 GMT
Content-Length: 0
```

5.2.14 获取桶配额

功能介绍

桶的拥有者可以执行获取桶配额信息的操作。桶的拥有者的状态是 inactive 状态不可以查询桶配额信息。桶空间配额值的单位为 Byte（字节），0 代表不设上限。

请求消息样式

```
GET /?quota HTTP/1.1
Host: bucketname.obs.region.example.com
Date: date
Authorization: authorization
```

请求消息参数

该请示消息中不带消息参数。

请求消息头

该请求使用公共的请求消息头，具体参见表 3-3。

请求消息元素

该请求消息不使用消息元素。

响应消息样式

```
HTTP/1.1 status_code
Date: date
Content-Type: application/xml
Content-Length: length

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Quota xmlns="http://obs.region.example.com/doc/2015-06-30/">
   <StorageQuota>quota</StorageQuota>
</Quota>
```

响应消息头

该请求的响应消息使用公共消息头，具体请参考表 3-20。

响应消息元素

该响应以消息元素的形式返回桶的配额信息，元素的具体意义如表 5-23 所示。

表5-23 响应消息元素

元素名称	描述
Quota	桶的配额，包含配额量元素。 类型：XML。
StorageQuota	桶的配额量。单位字节。 类型：字符串。

错误响应消息

无特殊错误，所有错误已经包含在表 6-3 中。

请求示例

```
GET /?quota HTTP/1.1
User-Agent: curl/7.29.0
Host: examplebucket.obs.region.example.com
Accept: */*
Date: WED, 01 Jul 2015 03:27:45 GMT
Authorization: OBS H4IPJX0TQTHTHEBQQCEC:8m4bW1gFCNeXQlfu45uO2gpo7l8=
```

响应示例

```
HTTP/1.1 200 OK
Server: OBS
x-obs-request-id: BF260000016436B55D8DED9AE26C4D18
x-obs-id-2: 32AAAQAAEAABAAAQAAEAABAAAQAAEAABCSs2Q5vz5AfpAJ/CMNgCfo2hmDowp7M9
Content-Type: application/xml
Date: WED, 01 Jul 2015 03:27:45 GMT
Content-Length: 150

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Quota xmlns="http://obs.example.com/doc/2015-06-30/">
  <StorageQuota>0</StorageQuota>
</Quota>
```

5.2.15 获取桶存量信息

功能介绍

查询桶桶对象个数及对象占用空间，对象占用空间大小值为非负整数，单位为 Byte（字节）。

说明

由于 OBS 桶存量是后台统计，因此存量会有一定的时延，不能实时更新，因此不建议对存量做实时校验。

请求消息样式

```
GET /?storageinfo HTTP/1.1
Host: bucketname.obs.region.example.com
Date: date
Authorization: authorization
```

请求消息参数

该请求不使用请求消息参数。

请求消息头

该请求使用公共消息头，具体参见表 3-3。

请求消息元素

该请求消息中不使用请求消息元素。

响应消息样式

```
HTTP/1.1 status code
Date: date
Content-Type: type
Content-Length: length
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<GetBucketStorageInfoResult xmlns="http://obs.region.example.com/doc/2015-06-30/">
<Size>size</Size>
<ObjectNumber>number</ObjectNumber>
</GetBucketStorageInfoResult>
```

响应消息头

该请求的响应消息使用公共消息头，具体请参考表 3-20。

响应消息元素

该响应中将桶存量信息以消息元素的形式返回，元素的具体含义如表 5-24 所示。

表5-24 响应消息元素

元素名称	描述
GetBucketStorageInfoResult	保存桶存量请求结果，包含存量大小和对象个数。 类型：XML。
Size	返回存量大小。 类型：整型。
ObjectNumber	返回对象个数。 类型：整型。

错误响应消息

无特殊错误，所有错误已经包含在表 6-3 中。

请求示例

```
GET /?storageinfo HTTP/1.1
User-Agent: curl/7.29.0
Host: examplebucket.obs.region.example.com
Accept: */*
Date: WED, 01 Jul 2015 03:31:18 GMT
Authorization: OBS H4IPJX0TQTHTHEBQQCEC:bLcdeJGYWw/eEEjMhPZx2MK5R9U=
```

响应示例

```
HTTP/1.1 200 OK
Server: OBS
x-obs-request-id: BF260000016435DD2958BFDCDB86B55E
x-obs-id-2: 32AAAQAAEAABAAAQAAEAABAAAQAAEAABCSitZctaPYVnat49fVMd1O+OWIP1yrg3
Content-Type: application/xml
WED, 01 Jul 2015 03:31:18 GMT
Content-Length: 206

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<GetBucketStorageInfoResult xmlns="http://obs.example.com/doc/2015-06-30/">
  <Size>25490</Size>
  <ObjectNumber>24</ObjectNumber>
</GetBucketStorageInfoResult>
```

5.3 对象操作

5.3.1 PUT 上传

功能介绍

上传对象操作是指在指定的桶内增加一个对象，执行该操作需要用户拥有桶的写权限。

说明

同一个桶中存储的对象名是唯一的，在桶未开启多版本的情况下，重复上传同名对象，前一次上传的对象将被后一次上传的对象覆盖。

如果在指定的桶内已经有相同的对象键值的对象，用户上传的新对象会覆盖原来的对象；为了确保数据在传输过程中没有遭到破坏，用户可以在请求消息头中加入 Content-MD5 参数。在这种情况下，OBS 收到上传的对象后，会对对象进行 MD5 校验，如果不一致则返回出错信息。

用户还可以在上传对象时指定 x-obs-acl 参数，设置对象的权限控制策略。如果匿名用户在上传对象时未指定 x-obs-acl 参数，则该对象默认可以被所有 OBS 用户访问。

用户在 OBS 系统中创建了桶之后，可以采用 PUT 操作的方式将对象上传到桶中。

单次上传对象大小范围是[0, 5GB]，如果需要上传超过 5GB 的大文件，需要通过 5.4 多段操作来分段上传。

与 POST 上传的区别

PUT 上传中参数通过请求头域传递；POST 上传则作为消息体中的表单域传递。

PUT 上传需在 URL 中指定对象名；POST 上传提交的 URL 为桶域名，无需指定对象名。两者的请求行分别为：

```
PUT /ObjectName HTTP/1.1
POST / HTTP/1.1
```

关于 POST 上传的更多详细信息，请参考 5.3.2 POST 上传。

多版本

如果桶的多版本状态是开启的，系统会自动为对象生成一个唯一的版本号，并且会在响应报头 x-obs-version-id 返回该版本号。如果桶的多版本状态是暂停的，则对象的版本号为 **null**。关于桶的多版本状态，参见 5.2.11 设置桶的多版本状态。

请求消息样式

```
PUT /ObjectName HTTP/1.1
Host: bucketname.obs.region.example.com
Content-Type: application/xml
Content-Length: length
Authorization: authorization
Date: date
<Optional Additional Header>
<object Content>
```

请求消息参数

该请求消息中不使用参数。

请求消息头

该请求可以使用附加的消息头，具体如表 5-25 所示。

说明

OBS 支持在上传对象时在请求里携带 HTTP 协议规定的 6 个请求头：Cache-Control、Expires、Content-Encoding、Content-Disposition、Content-Type、Content-Language。如果上传 Object 时设置了这些请求头，OBS 会直接将这些头域的值保存下来。这 6 个值也可以通过 OBS 提供的修改对象元数据 API 接口进行修改。在该 Object 被下载或者 HEAD 的时候，这些保存的值将会被设置到对应的 HTTP 头域中返回客户端。

表5-25 请求消息头

消息头名称	描述	是否必选

消息头名称	描述	是否必选
Content-MD5	按照 RFC 1864 标准计算出消息体的 MD5 摘要字符串，即消息体 128-bit MD5 值经过 base64 编码后得到的字符串。 类型：字符串 示例：n58IG6hfM7vqI4K0vnWpog==。	否
x-obs-acl	创建对象时，可以加上此消息头设置对象的权限控制策略，使用的策略为预定义的常用策略，包括：private；public-read；public-read-write 类型：字符串 说明：字符串形式的预定义策略。 实例 x-obs-acl: public-read。	否
x-obs-grant-read	创建对象时，使用此头域授权租户下所有用户有读对象和获取对象元数据的权限。 类型：字符串。 实例 x-obs-grant-read: id=domainID。如果授权给多个租户，需要通过','分割。	否
x-obs-grant-read-acp	创建对象时，使用此头域授权租户下所有用户有获取对象 ACL 的权限。 类型：字符串。 实例 x-obs-grant-read-acp: id=domainID。如果授权给多个租户，需要通过','分割。	否
x-obs-grant-write-acp	创建对象时，使用此头域授权 domain 下所有用户有写对象 ACL 的权限。 类型：字符串。 实例 x-obs-grant-write-acp: id=domainID。如果授权给多个租户，需要通过','分割。	否
x-obs-grant-full-control	创建对象时，使用此头域授权 domain 下所有用户有读对象、获取对象元数据、获取对象 ACL、写对象 ACL 的权限。 类型：字符串。 实例 x-obs-grant-full-control: id=domainID。如果授权给多个租户，需要通过','分割。	否
x-obs-meta-*	创建对象时，可以在 HTTP 请求中加入以“x-obs-meta-”开头的消息头，用来加入自定义的元数据，以便对对象进行自定义管理。当用户获取此对象或查询此对象元数据时，加入的自定义元数据将会在返回消息的头中出现。HTTP 请求不包含消息体，长度不能超过 8KB。	否

消息头名称	描述	是否必选
	类型：字符串 示例：x-obs-meta-test: test metadata 约束：请求头字段中的关键字不允许含有非 ASCII 码或不可识别字符，如果一定要使用非 ASCII 码或不可识别字符，需要客户端自行做编解码处理，可以采用 URL 编码或者 Base64 编码，服务端不会做解码处理。例如“中文”是非 ASCII 码，可以做 URL 编码为“%E4%B8%AD%E6%96%87”，头域 x-obs-meta-%E4%B8%AD%E6%96%87: test%E4%B8%AD%E6%96%87。自定义元数据就是 x-obs-meta-%E4%B8%AD%E6%96%87，服务端只会作为字符串处理，不会做解码。 说明：请求头支持大小写字母，服务端会把请求消息头的 key 转换成小写，value 不变。例如：x-obs-meta-Test1: Test Meta1，设置成功之后，下载对象返回的消息头是：x-obs-meta-test1: Test Meta1。	
x-obs-website-redirect-location	当桶设置了 Website 配置，可以将获取这个对象的请求重定向到桶内另一个对象或一个外部的 URL，OBS 将这个值从头域中取出，保存在对象的元数据中。 例如，重定向请求到桶内另一对象： x-obs-website-redirect-location:/anotherPage.html 或重定向请求到一个外部 URL： x-obs-website-redirect-location:http://www.example.com/ 类型：字符串 默认值：无 约束：必须以“/”、“http://”或“https://”开头，长度不超过 2KB。	否
success-action-redirect	此参数的值是一个 URL，用于指定当此次请求操作成功响应后的重定向的地址。 • 如果此参数值有效且操作成功，响应码为 303，Location 头域由此参数以及桶名、对象名、对象的 ETag 组成。 • 如果此参数值无效，则 OBS 忽略此参数的作用，响应码为 204，Location 头域为对象地址。 类型：字符串。	否
x-obs-expires	表示对象的过期时间，单位是天。过期之后对象会被自动删除。（从对象最后修改时间开始计算） 类型：整型。 示例：x-obs-expires:3	否

请求消息元素

该请求消息中不使用消息元素，在消息体中带的是对象的数据。

响应消息样式

```
HTTP/1.1 status code
Content-Length: length
Content-Type: type
```

响应消息头

该请求的响应消息使用公共消息头，具体请参考表 3-20。

除公共响应消息头之外，还可能使用如表 5-26 中的消息头。

表5-26 附加响应消息头

消息头名称	描述
x-obs-version-id	对象的版本号。如果桶的多版本状态为开启，则会返回对象的版本号。 类型：字符串

响应消息元素

该请求的响应消息不带消息元素。

错误响应消息

该请求的返回无特殊错误，所有错误已经包含在表 6-3 中。

请求示例 1

上传对象

```
PUT /object01 HTTP/1.1
User-Agent: curl/7.29.0
Host: examplebucket.obs.region.example.com
Accept: */*
Date: WED, 01 Jul 2015 04:11:15 GMT
Authorization: OBS H4IPJX0TQTHTHEBQQCEC:gYqplLq30dEX7GMi2qFWyjdFsyw=
Content-Length: 10240
Expect: 100-continue

[1024 Byte data content]
```

响应示例 1

```
HTTP/1.1 200 OK
Server: OBS
x-obs-request-id: BF2600000164364C10805D385E1E3C67
ETag: "d41d8cd98f00b204e9800998ecf8427e"
x-obs-id-2: 32AAAWJAMAABAAAQAAEAABAAAQAAEAABCTzu4Jp2lquWuXsjnLyPPiT3cfGhqPoY
Date: WED, 01 Jul 2015 04:11:15 GMT
Content-Length: 0
```

请求示例 2

上传对象的同时设置 ACL

```
PUT /object01 HTTP/1.1
User-Agent: curl/7.29.0
Host: examplebucket.obs.region.example.com
Accept: */*
Date: WED, 01 Jul 2015 04:13:55 GMT
x-obs-grant-
read:id=52f24s3593as5730ea4f722483579ai7,id=a93fcas852f24s3596ea8366794f7224
Authorization: OBS H4IPJX0TQTHTHEBQQCEC:gYqplLq30dEX7GMi2qFWyjdFsyw=
Content-Length: 10240
Expect: 100-continue

[1024 Byte data content]
```

响应示例 2

```
HTTP/1.1 200 OK
Server: OBS
x-obs-request-id: BB7800000164845759E4F3B39ABEE55E
ETag: "d41d8cd98f00b204e9800998ecf8427e"
x-obs-id-2: 32AAAQAAEAABAAAQAAEAABAAAQAAEAABCSReVRNuas0knI+Y96iXrZA7BLUgj06Z
Date: WED, 01 Jul 2015 04:13:55 GMT
Content-Length: 0
```

请求示例 4

桶开启多版本时上传对象

```
PUT /object01 HTTP/1.1
User-Agent: curl/7.29.0
Host: examplebucket.obs.region.example.com
Accept: */*
Date: WED, 01 Jul 2015 04:17:12 GMT
Authorization: OBS H4IPJX0TQTHTHEBQQCEC:uFVJhp/dJqj/CJIVLrSZ0gpw3ng=
Content-Length: 10240
Expect: 100-continue

[1024 Byte data content]
```

响应示例 4

```
HTTP/1.1 200 OK
Server: OBS
```

```
x-obs-request-id: DCD2FC9CAB78000001439A51DB2B2577
ETag: "d41d8cd98f00b204e9800998ecf8427e"
X-OBS-ID-2: GcVgfeOJHx8JZHTHrRqkPsbKdB583fYbr3RBbHT6mMrBstReVILBZbMAdLiBYy1l
Date: WED, 01 Jul 2015 04:17:12 GMT
x-obs-version-id: AAABQ4q2M9_c0vycq3gAAAAAVURTRkha
Content-Length: 0
```

请求示例 5

上传对象时携带 MD5

```
PUT /object01 HTTP/1.1
User-Agent: curl/7.29.0
Host: examplebucket.obs.region.example.com
Accept: */*
Date: WED, 01 Jul 2015 04:17:50 GMT
Authorization: OBS H4IPJX0TQTHTHEBQQCEC:uFVJhp/dJqj/CJIVLrSZ0gpw3ng=
Content-Length: 10
Content-MD5: 6Afx/PgtEy+bsBjKZzihnw==
Expect: 100-continue

1234567890
```

响应示例 5

```
HTTP/1.1 200 OK
Server: OBS
x-obs-request-id: BB7800000164B165971F91D82217D105
X-OBS-ID-2: 32AAAUJAIAABAAAQAAEAABAAAQAAEAABCSEKhBpS4BB3dSMNqMtuNxQDD9XvOw5h
ETag: "1072e1b96b47d7ec859710068aa70d57"
Date: WED, 01 Jul 2015 04:17:50 GMT
Content-Length: 0
```

请求示例 6

桶设置了 Website 配置，上传对象时设置下载对象时重定向

```
PUT /object01 HTTP/1.1
User-Agent: curl/7.29.0
Host: examplebucket.obs.region.example.com
Accept: */*
Date: WED, 01 Jul 2015 04:17:12 GMT
x-obs-website-redirect-location: http://www.example.com/
Authorization: OBS H4IPJX0TQTHTHEBQQCEC:uFVJhp/dJqj/CJIVLrSZ0gpw3ng=
Content-Length: 10240
Expect: 100-continue

[1024 Byte data content]
```

响应示例 6

```
HTTP/1.1 200 OK
Server: OBS
x-obs-request-id: DCD2FC9CAB78000001439A51DB2B2577
x-obs-id-2: 32AAAUJAIAABAAAQAAEAABAAAQAAEAABCTmxB5ufMj/7/GzP8TFwTbp33u0xhn2Z
```

```
ETag: "1072e1b96b47d7ec859710068aa70d57"
Date: WED, 01 Jul 2015 04:17:12 GMT
x-obs-version-id: AAABQ4q2M9_c0vycq3gAAAAAVURTRkha
Content-Length: 0
```

请求示例 7

在 URL 中携带签名并上传对象

```
PUT
/object02?AccessKeyId=H4IPJX0TQTHTHEBQQCEC&Expires=1532688887&Signature=EQmDuOhWLUr
zrzRNZxwS72CXeXM%3D HTTP/1.1
User-Agent: curl/7.29.0
Host: examplebucket.obs.region.example.com
Accept: */*
Content-Length: 1024

[1024 Byte data content]
```

响应示例 7

```
HTTP/1.1 200 OK
Server: OBS
x-obs-request-id: DCD2FC9CAB78000001439A51DB2B2577
x-obs-id-2: 32AAAUJAIAABAAAQAAEAABAAAQAAEAABCTmxB5ufMj/7/GzP8TFwTbp33u0xhn2Z
ETag: "1072e1b96b47d7ec859710068aa70d57"
Date: Fri, 27 Jul 2018 10:52:31 GMT
x-obs-version-id: AAABQ4q2M9 c0vycq3gAAAAAVURTRkha
Content-Length: 0
```

5.3.2 POST 上传

功能介绍

上传对象操作是指在指定的桶内增加一个对象，执行该操作需要用户拥有桶的写权限。

说明

同一个桶中存储的对象名是唯一的，在桶未开启多版本的情况下，重复上传同名对象，前一次上传的对象将被后一次上传的对象覆盖。

如果在指定的桶内已经有相同的对象键值的对象，用户上传的新对象会覆盖原来的对象；为了确保数据在传输过程中没有遭到破坏，用户可以在请求消息头中加入 Content-MD5 参数。在这种情况下，OBS 收到上传的对象后，会对对象进行 MD5 校验，如果不一致则返回出错信息。用户还可以在上传对象时指定 x-obs-acl 参数，设置对象的权限控制策略。

用户除了可以用 PUT 直接上传对象外，还可以使用 POST 上传对象。

单次上传对象大小范围是[0, 5GB]，如果需要上传超过 5GB 的大文件，需要通过 5.4 多段操作来分段上传。

与 PUT 上传的区别

PUT 上传中参数通过请求头域传递；POST 上传则作为消息体中的表单域传递。

PUT 上传需在 URL 中指定对象名；POST 上传提交的 URL 为桶域名，无需指定对象名。两者的请求行分别为：

```
PUT /ObjectName HTTP/1.1
POST / HTTP/1.1
```

关于 PUT 上传的更多详细信息，请参考 5.3.1 PUT 上传。

多版本

如果桶的多版本状态是开启的，系统会自动为对象生成一个唯一的版本号；如果桶的多版本状态是暂停的，则系统生成的对象版本号为 **null**，并由响应报头 x-obs-version-id 返回该版本号。关于桶的多版本状态，参见 5.2.11 设置桶的多版本状态。

请求消息样式

```
POST / HTTP/1.1
Host: bucketname.obs.region.example.com
User-Agent: browser data
Accept: file types
Accept-Language: Regions
Accept-Encoding: encoding
Accept-Charset: character set
Keep-Alive: 300
Connection: keep-alive
Content-Type: multipart/form-data; boundary=-9431149156168
Content-Length: length


--9431149156168
Content-Disposition: form-data; name="key"
acl
--9431149156168
Content-Disposition: form-data; name="success action redirect"
success redirect
--9431149156168
Content-Disposition: form-data; name="content-Type"
content type
--9431149156168
Content-Disposition: form-data; name="x-obs-meta-uuid"
uuid
--9431149156168
Content-Disposition: form-data; name="x-obs-meta-tag"
metadata
--9431149156168
Content-Disposition: form-data; name="AccessKeyId"
access-key-id
--9431149156168
Content-Disposition: form-data; name="policy"
encoded policy
--9431149156168
```

```
Content-Disposition: form-data; name="signature"
signature=
--9431149156168
Content-Disposition: form-data; name="file"; filename="MyFilename"
Content-Type: image/jpeg
file_content
--9431149156168
Content-Disposition: form-data; name="submit"
Upload to OBS
--9431149156168--
```

请求消息参数

该请求消息中不使用参数。

请求消息头

该请求使用公共的消息头，具体请参见表 3-3。

如果想要获取 CORS 配置信息，则需要使用的消息头如下表 5-27 所示。

表5-27 获取 CORS 配置的请求消息头

消息头名称	描述	是否必选
Origin	预请求指定的跨域请求 Origin（通常为域名）。 类型：字符串	是
Access-Control-Request-Headers	实际请求可以带的 HTTP 头域，可以带多个头域。 类型：字符串	否

请求消息元素

该请求消息的消息元素以表单形式组织，表单字段的具体含义如表 5-28 所示。

表5-28 请求消息表单元素

参数名称	描述	是否必选
file	上传的对象的内容。 类型：二进制或文本类型。 约束条件：此参数必须为最后一个参数，否则此参数之后的参数会被丢弃；一个请求中只能含有一个 file 参数。	是
key	通过此请求创建的对象名称。	是

参数名称	描述	是否必选
	类型：字符串。	
AccessKeyId	用来指明请求发起者的 Access Key。 类型：字符串。 约束条件：如果该请求包括安全策略参数，则必须包括此参数。	否
policy	该请求的安全策略描述。policy 格式请参考 3.2.4 基于浏览器上传的表单中携带签名章节中 policy 格式 类型：字符串。	是
token	用来统一指明请求发起者的 Access Key，请求签名和请求的安全策略。token 的优先级高于单独指定的 Access Key，请求签名和请求的安全策略。 类型：字符串。 示例： HTML 中：<input type="text" name="token" value="ak:signature:policy" />	否
x-obs-acl	创建对象时，可以加上此消息头设置对象的权限控制策略，使用的策略为预定义的常用策略，包括：private；public-read；public-read-write；public-read-delivered ; public-read-write-delivered。 类型：字符串。 示例： POLICY 中：{"acl": "public-read" }, HTML 中：<input type="text" name="acl" value="public-read" />	否
x-obs-grant-read	创建对象时，使用此头域授权 domain 下所有用户有读对象和获取对象元数据的权限。 类型：字符串。 示例： POLICY 中：{"grant-read": "id=domainId1" }, HTML 中：<input type="text" name="grant-read" value="id=domainId1" />	否
x-obs-grant-read-acp	创建对象时，使用此头域授权 domain 下所有用户有获取对象 ACL 的权限。 类型：字符串。 示例： POLICY 中：{"grant-read-acp": "id=domainId1" },	否

参数名称	描述	是否必选
	HTML 中：<input type="text" name="grant-read-acp" value="id=domainId1" />	
x-obs-grant-write-acp	创建对象时，使用此头域授权 domain 下所有用户有写对象 ACL 的权限。 类型：字符串。 示例： POLICY 中：{"grant-write-acp": "id=domainId1" }, HTML 中：<input type="text" name="grant-write-acp" value="id=domainId1" />	否
x-obs-grant-full-control	创建对象时，使用此头域授权 domain 下所有用户有读对象、获取对象元数据、获取对象 ACL、写对象 ACL 的权限。 类型：字符串。 示例： POLICY 中：{"grant-full-control": "id=domainId1" }, HTML 中：<input type="text" name="grant-full-control" value="id=domainId1" />	否
Cache-Control, Content-Type, Content-Disposition, Content-Encoding Expires	这 5 个参数是 HTTP 标准消息头，OBS 将这些参数记录下来，当用户下载此对象或 Head Object 时，在响应消息头中携带这些参数。 类型：字符串。 示例： POLICY 中：["starts-with", "$Content-Type", "text/"], HTML 中：<input type="text" name="content-type" value="text/plain" />	否
success_action_redirect	此参数的值是一个 URL，用于指定当此次请求操作成功响应后的重定向的地址。 • 如果此参数值有效且操作成功，响应码为 303，Location 头域由此参数以及桶名、对象名、对象的 ETag 组成。 • 如果此参数值无效，则 OBS 忽略此参数的作用，响应码为 204，Location 头域为对象地址。 类型：字符串。 示例： POLICY 中：{"success_action_redirect": "http://123458.com"}, HTML 中：<input type="text" name="success_action_redirect"	否

参数名称	描述	是否必选
	value="http://123458.com" />	
x-obs-meta-*	创建对象时，可以在 HTTP 请求中加入“x-obs-meta-”消息头或以“x-obs-meta-”开头的消息头，用来加入自定义的元数据，以便对对象进行自定义管理。当用户获取此对象或查询此对象元数据时，加入的自定义元数据将会在返回消息的消息头中出现。 类型：字符串。 示例： POLICY 中：{" x-obs-meta-test ": " test metadata " }, HTML 中：<input type="text" name=" x-obs-meta-test " value=" test metadata " />	否
success_action_status	这个参数指定成功响应的状态码，允许设定的值为 200，201，204。 • 如果此参数值被设定为 200 或 204，OBS 响应消息中 body 为空。 • 如果此参数值被设定为 201，则 OBS 响应消息中包含一个 XML 文档描述此次请求的响应。 • 当请求不协带此参数或参数无效时，OBS 响应码为 204。 类型：字符串。 示例： POLICY 中：["starts-with", "$success_action_status", ""], HTML 中：<input type="text" name="success_action_status" value="200" />	否
x-obs-website-redirect-location	当桶设置了 Website 配置，可以将获取这个对象的请求重定向到桶内另一个对象或一个外部的 URL，OBS 将这个值从头域中取出，保存在对象的元数据中。 默认值：无 约束：必须以“/”、“http://”或“https://”开头，长度不超过 2K。	否
x-obs-expires	表示对象的过期时间，单位是天。过期之后对象会被自动删除。（从对象最后修改时间开始计算） 类型：整型。 示例：x-obs-expires:3	否

响应消息样式

```
HTTP/1.1 status code
Content-Type: application/xml
Location: location
Date: date
ETag: etag
```

响应消息头

该请求的响应消息使用公共消息头，具体请参考表 3-20。

除公共响应消息头之外，还可能使用如表 5-29 中的消息头。

表5-29 附加响应消息头

消息头名称	描述
x-obs-version-id	对象的版本号。如果桶的多版本状态为开启，则会返回对象的版本号。如果桶的多版本状态为暂停，则会返回 **null**。 类型：字符串
Access-Control-Allow-Origin	当桶设置了 CORS 配置，如果请求的 Origin 满足服务端的 CORS 配置，则在响应中包含这个 Origin。 类型：字符串
Access-Control-Allow-Headers	当桶设置了 CORS 配置，如果请求的 headers 满足服务端的 CORS 配置，则在响应中包含这个 headers。 类型：字符串
Access-Control-Max-Age	当桶设置了 CORS 配置，服务端 CORS 配置中的 MaxAgeSeconds。 类型：整数
Access-Control-Allow-Methods	当桶设置了 CORS 配置，如果请求的 Access-Control-Request-Method 满足服务端的 CORS 配置，则在响应中包含这条 rule 中的 Methods。 类型：字符串 有效值：GET、PUT、HEAD、POST 、DELETE
Access-Control-Expose-Headers	当桶设置了 CORS 配置，服务端 CORS 配置中的 ExposeHeader。 类型：字符串
x-obs-server-side-encryption	如果服务端加密是 SSE-KMS 方式，响应包含该头域。 类型：字符串

消息头名称	描述
	示例：x-obs-server-side-encryption：kms

响应消息元素

该请求的响应消息中不带消息元素。

错误响应消息

无特殊错误，所有错误已经包含在表 6-3 中。

请求示例 1

普通 POST 上传

```
POST / HTTP/1.1
Date: WED, 01 Jul 2015 04:15:23 GMT
Host: examplebucket.obs.region.example.com
Content-Type: multipart/form-data; boundary=----------------------------7db143f50da2
Content-Length: 2424
Origin: www.example.com
Access-Control-Request-Headers:acc header 1

------------------------------7db143f50da2
Content-Disposition: form-data; name="key"
object01
------------------------------7db143f50da2
Content-Disposition: form-data; name="acl"
public-read
------------------------------7db143f50da2
Content-Disposition: form-data; name="content-type"
text/plain
------------------------------7db143f50da2
Content-Disposition: form-data; name="expires"
WED, 01 Jul 2015 04:16:15 GMT
------------------------------7db143f50da2
Content-Disposition: form-data; name="AccessKeyId"
14RZT432N80TGDF2Y2G2
------------------------------7db143f50da2
Content-Disposition: form-data; name="policy"
ew0KICAiZXhwaXJhdGlvbiI6ICIyMDE1LTA3LTAxVDEyOjAwOjAwLjAwMFoiLA0KICAiY29uZGl0aW9ucyI
6IFsNCiAgICB7ImJ1Y2tldCI6ICJleG1hcGxlYnVja2V0IiB9LA0KICAgIHsiYWNsIjogInB1YmxpYy1yZW
FkIiB9LA0KICAgIHsiRXhwaXJlcyI6ICIxMDAwIiB9LA0KICAgIFsiZXEiLCAiJGtleSIsICJvYmplY3QwM
SJdLA0KICAgIFsic3RhcnRzLXdpdGgiLCAiJENvbnRlbnQtVHlwZSIsICJ0ZXh0LyJdLA0KICBdDQp9DQo=
------------------------------7db143f50da2
Content-Disposition: form-data; name="signature"
Vk6rwO0Nq09BLhvNSIYwSJTRQ+k=
------------------------------7db143f50da2
Content-Disposition: form-data; name="file";
filename="C:\Testtools\UpLoadFiles\object\1024Bytes.txt"
Content-Type: text/plain
```

```
01234567890
----------------------------7db143f50da2
Content-Disposition: form-data; name="submit"
Upload
----------------------------7db143f50da2--
```

响应示例 1

桶配置 cors 后，响应会包含 Access-Control-*的信息。

```
HTTP/1.1 204 No Content
x-obs-request-id: 90E2BA00C26C00000133B442A90063FD
x-obs-id-2: OTBFMkJBMDBDMjZDMDAwMDAxMzNCNDQyQTkwMDYzRkRBQUFBQUFBQWJiYmJiYmJi
Access-Control-Allow-Origin: www.example.com
Access-Control-Allow-Methods: POST,GET,HEAD,PUT
Access-Control-Allow-Headers: acc_header_01
Access-Control-Max-Age: 100
Access-Control-Expose-Headers: exp_header_01
Content-Type: text/xml
Location: http://examplebucket.obs.region.example.com/object01
Date: WED, 01 Jul 2015 04:15:23 GMT
ETag: "ab7abb0da4bca5323ab6119bb5dcd296"
```

5.3.3 复制对象

功能介绍

复制对象（Copy Object）特性用来为 OBS 上已经存在的对象创建一个副本。

当进行复制对象操作时，目标对象默认复制源对象的元数据；用户也可以将目标对象的元数据替换为本次请求中所带的元数据。新建的目标对象不会复制源对象的 ACL 信息，默认的新建对象的 ACL 是 private，用户可以使用设置 ACL 的操作接口来重新设定新对象的 ACL。

复制对象操作的请求需要通过头域携带拷贝的原桶和对象信息，不能携带消息实体。

目标对象大小范围是[0, 5GB]，如果源对象大小超过 5GB，只能使用 Range 拷贝部分对象。

多版本

默认情况下，x-obs-copy-source 标识复制源对象的最新版本。如果源对象的最新版本是删除标记，则认为该对象已删除。要复制指定版本的对象，可以在 x-obs-copy-source 请求消息头中携带 versionId 参数。

如果目标对象的桶的多版本状态是开启的，系统为目标对象生成唯一的版本号（此版本号与源对象的版本号不同），并且会在响应报头 x-obs-version-id 返回该版本号。如果目标对象的桶的多版本状态是暂停的，则目标对象的版本号为 **null**。

须知

在桶没有开启多版本的情况下，将源对象 object_A 复制为目标对象 object_B，如果在复制操作之前对象 object_B 已经存在，复制操作执行之后老对象 object_B 则会被新复制对象 object_B 覆盖，复制成功后，只能下载到新的对象 object_B，老对象 object_B 将会被删除。因此在使用 copy 接口时请确保目标对象不存在或者已无价值，避免因 copy 导致数据误删除。复制过程中源对象 object_A 无任何变化。

复制对象的结果不能仅根据 HTTP 返回头域中的 status_code 来判断请求是否成功，头域中 status_code 返回 200 时表示服务端已经收到请求，且开始处理复制对象请求。复制是否成功会在响应消息的 body 中，只有 body 体中有 ETag 标签才表示成功，否则表示复制失败。

请求消息样式

```
PUT /destinationObjectName HTTP/1.1
Host: bucketname.obs.region.example.com
x-obs-copy-source: /sourceBucket/sourceObject
x-obs-metadata-directive: metadata directive
x-obs-copy-source-if-match: etag
x-obs-copy-source-if-none-match: etag
x-obs-copy-source-if-unmodified-since: time stamp
x-obs-copy-source-if-modified-since: time stamp
Authorization: signature
Date: date
```

请求消息参数

该请求消息中不使用消息参数。

请求消息头

该消息可以带附加的消息头指定复制的信息，具体如表 3-3 所示。

表5-30 请求消息头

消息头名称	描述	是否必选
x-obs-acl	复制对象时，可以加上此消息头设置对象的权限控制策略，使用的策略为预定义的常用策略，包括：private；public-read；public-read- write。 类型：字符串。 示例：x-obs-acl: acl	否
x-obs-grant-read	创建对象时，使用此头域授权 domain 下所有用户有读对象和获取对象元数据的权限 类型：字符串。	否
x-obs-grant-read-acp	创建对象时，使用此头域授权 domain 下所有用户有获	否

消息头名称	描述	是否必选
	取对象 ACL 的权限。 类型：字符串。	
x-obs-grant-write-acp	创建对象时，使用此头域授权 domain 下所有用户有写对象 ACL 的权限。 类型：字符串。	否
x-obs-grant-full-control	创建对象时，使用此头域授权 domain 下所有用户有读对象、获取对象元数据、获取对象 ACL、写对象 ACL 的权限。 类型：字符串。	否
x-obs-copy-source	用来指定复制对象操作的源桶名以及源对象名。当源对象存在多个版本时，通过 versionId 参数指定版本源对象。 类型：字符串。 示例：x-obs-copy-source: /source_bucket/sourceObject	是
x-obs-metadata-directive	此参数用来指定新对象的元数据是从源对象中复制，还是用请求中的元数据替换。 类型：字符串。 有效取值：COPY 或 REPLACE。 默认值：COPY。 示例：x-obs-metadata-directive: metadata_directive 约束条件：如果此参数的值不是 COPY 或 REPLACE，则 OBS 立即返回 400 错误；如果用户进行修改元数据操作（源对象与目标对象相同），则此参数只能为 REPLACE，否则此请求作为无效请求，服务端响应 400。此参数不支持将加密的对象更改成非加密对象（源对象与目标对象相同）。如果用户使用此参数更改加密的对象，系统将返回 400。	否
x-obs-copy-source-if-match	只有当源对象的 Etag 与此参数指定的值相等时才进行复制对象操作，否则返回 412（前置条件不满足）。 类型：字符串。 示例：x-obs-copy-source-if-match: etag 约束条件：此参数可与 x-obs-copy-source-if-unmodified-since 一起使用，但不能与其它条件复制参数一起使用。	否
x-obs-copy-source-if-none-match	只有当源对象的 Etag 与此参数指定的值不相等时才进行复制对象操作，否则返回 412（前置条件不满足）。 类型：字符串。 示例：x-obs-copy-source-if-none-match: etag	否

消息头名称	描述	是否必选
	约束条件：此参数可与 x-obs-copy-source-if-modified-since 一起使用，但不能与其它条件复制参数一起使用。	
x-obs-copy-source-if-unmodified-since	只有当源对象在此参数指定的时间之后没有修改过才进行复制对象操作，否则返回 412（前置条件不满足），此参数可与 x-obs-copy-source-if-match 一起使用，但不能与其它条件复制参数一起使用。 类型：符合 http://www.ietf.org/rfc/rfc2616.txt 规定格式的 HTTP 时间字符串。 示例：x-obs-copy-source-if-unmodified -since: time-stamp	否
x-obs-copy-source-if-modified-since	只有当源对象在此参数指定的时间之后修改过才进行复制对象操作，否则返回 412（前置条件不满足），此参数可与 x-obs-copy-source-if-none-match 一起使用，但不能与其它条件复制参数一起使用。 类型：符合 http://www.ietf.org/rfc/rfc2616.txt 规定格式的 HTTP 时间字符串。 示例：x-obs-copy-source-if-modified-since: time-stamp	否
x-obs-website-redirect-location	当桶设置了 Website 配置，可以将获取这个对象的请求重定向到桶内另一个对象或一个外部的 URL，OBS 将这个值从头域中取出，保存在对象的元数据中。 类型：字符串 默认值：无 约束：必须以“/”、“http://”或“https://”开头，长度不超过 2K。	否
success_action_redirect	此参数的值是一个 URL，用于指定当此次请求操作成功响应后的重定向的地址。 • 如果此参数值有效且操作成功，响应码为 303，Location 头域由此参数以及桶名、对象名、对象的 ETag 组成。 • 如果此参数值无效，则 OBS 忽略此参数的作用，响应码为 204，Location 头域为对象地址。 类型：字符串。	否

其他消息头请参见表 3-3 章节。

请求消息元素

该请求在消息中不使用消息元素。

响应消息样式

```
HTTP/1.1 status code
Content-Type: application/xml
Date: date
Content-Length: length

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<CopyObjectResult xmlns="http://obs.region.example.com/doc/2015-06-30/">
   <LastModified>modifiedDate</LastModified>
   <ETag>etagValue</ETag>
</CopyObjectResult>
```

响应消息头

该请求的响应消息使用公共消息头，具体请参考表 3-20。

除公共响应消息头之外，还可能使用如下表 5-31 中的消息头。

表5-31 附加响应消息头

消息头名称	描述
x-obs-copy-source-version-id	源对象的版本号。 类型：字符串。
x-obs-version-id	目标对象的版本号。 类型：字符串

响应消息元素

该请求的响应消息通过消息元素来返回复制结果，元素具体含义如表 5-32 所示。

表5-32 响应消息元素

元素名称	描述
CopyObjectResult	复制对象结果的 Container。 类型：XML。
LastModified	对象上次修改的时间。 类型：字符串。
ETag	新对象的 base64 编码的 128 位 MD5 摘要。ETag 是对象内容的唯一标识，可以通过该值识别对象内容是否有变化。比如上传对象时 ETag 为 A，下载对象时 ETag 为 B，则说明对象内容发生了变化。 类型：字符串。

错误响应消息

无特殊错误，所有错误已经包含在表 6-3 中。

请求示例 1

普通对象拷贝，将桶 bucket 中的对象 srcobject 拷贝到桶 examplebucket 中 destobject 对象

```
PUT /destobject HTTP/1.1
User-Agent: curl/7.29.0
Host: examplebucket.obs.region.example.com
Accept: */*
Date: WED, 01 Jul 2015 04:19:21 GMT
Authorization: OBS H4IPJX0TQTHTHEBQQCEC:2rZR+iaH8xUewvUKuicLhLHpNoU=
x-obs-copy-source: /bucket/srcobject
```

响应示例 1

```
HTTP/1.1 200 OK
Server: OBS
x-obs-request-id: 001B21A61C6C00000134031BE8005293
x-obs-id-2: MDAxQjIxQTYxQzZDMDAwMDAxMzQwMzFCRTgwMDUyOTNBQUFBQUFBQWJiYmJiYmJi
Date: WED, 01 Jul 2015 04:19:21 GMT
Content-Length: 249

<?xml version="1.0" encoding="utf-8"?>
<CopyObjectResult xmlns="http://obs.region.example.com/doc/2015-06-30/">
  <LastModified>2015-07-01T00:48:07.706Z</LastModified>
  <ETag>"507e3fff69b69bf57d303e807448560b"</ETag>
</CopyObjectResult>
```

请求示例 2

拷贝一个多版本对象，将桶 bucket 中的版本号为 AAABQ4uBLdLc0vycq3gAAAAEVURTRkha 的对象 srcobject 拷贝到桶 examplebucket 中 destobject 对象

```
PUT /destobject HTTP/1.1
User-Agent: curl/7.29.0
Host: examplebucket.obs.region.example.com
Accept: */*
Date: WED, 01 Jul 2015 04:20:29 GMT
Authorization: OBS H4IPJX0TQTHTHEBQQCEC:4BLYv+1UxfRSHBMvrhVLDszxvcY=
x-obs-copy-source: /bucket/srcobject?versionId=AAABQ4uBLdLc0vycq3gAAAAEVURTRkha
```

响应示例 2

```
HTTP/1.1 200 OK
Server: OBS
x-obs-request-id: DCD2FC9CAB78000001438B8A9C898B79
x-obs-id-2: DB/qBZmbN6AIoX9mrrSNYdLxwvbO0tLR/l6/XKTT4NmZspzhWrwp5Z74ybAYVOgr
Content-Type: application/xml
x-obs-version-id: AAABQ4uKnOrc0vycq3gAAAAFVURTRkha
x-obs-copy-source-version-id: AAABQ4uBLdLc0vycq3gAAAAEVURTRkha
```

```
Date: WED, 01 Jul 2015 04:20:29 GMT
Transfer-Encoding: chunked

<?xml version="1.0" encoding="utf-8"?>
<CopyObjectResult xmlns="http://obs.region.example.com/doc/2015-06-30/">
  <LastModified>2015-07-01T01:48:07.706Z</LastModified>
  <ETag>"507e3fff69b69bf57d303e807448560b"</ETag>
</CopyObjectResult>
```

5.3.4 获取对象内容

功能介绍

GET 操作从对象存下载对象。使用 GET 接口前，请确认必须拥有对象的 READ 权限。如果对象 Owner 授予对匿名用户 READ 访问权限，则访问该对象。

多版本

默认情况下，获取的是最新版本的对象。如果最新版本的对象是删除标记，则返回对象不存在。如果要获取指定版本的对象，请求可携带 versionId 消息参数。

请求消息样式

```
GET /ObjectName HTTP/1.1
Host: bucketname.obs.region.example.com
Date: date
Authorization: authorization
Range:bytes=byte range
<Optional Additional Header>
```

说明

其中 Range 字段可选，如果没有的话得到全部内容。

请求消息参数

GET 操作获取对象内容时，允许用户通过请求参数的方式对一些消息头值进行重写，可以重写的消息头有：Content-Type、Content-Language、Expires、Cache-Control、Content-Disposition 以及 Content-Encoding 共 6 个。另外所需取回的对象拥有多个版本时，可以通过 versionId 参数，指定需要下载的版本。具体的说明如表 5-33 所示。

说明

OBS 不会处理请求中携带的 Accept-Encoding，也不会对上传的数据做任何压缩、解压的操作，压缩解压的操作由客户端决定。某些 HTTPClient 在默认情况下可能会根据服务端返回的 Content-Encoding 对数据做相应的解压处理，客户端程序需要根据自己的需求决定是否做解压处理以及如何解压（修改 OBS 端保存的对象元数据 Content-Encoding 或者在下载对象时对 Content-Encoding 进行重写）。如果在下载对象的请求中指明了重写消息头，OBS 返回的 HTTP 标准消息头中将以请求中指定的重写内容为准。

表5-33 请求消息参数

参数名称	描述	是否必选

参数名称	描述	是否必选
response-content-type	重写响应中的 Content-Type 头。 类型：字符串。	否
response-content-language	重写响应中的 Content-Language 头。 类型：字符串。	否
response-expires	重写响应中的 Expires 头。 类型：字符串。	否
response-cache-control	重写响应中的 Cache-Control 头。 类型：字符串。	否
response-content-disposition	重写响应中的 Content-Disposition 头。 类型：字符串。 示例： response-content-disposition=attachment; filename*=utf-8''name1 下载对象重命名为“name1”。	否
response-content-encoding	重写响应中的 Content-Encoding 头。 类型：字符串。	否
versionId	指定获取对象的版本号。 类型：字符串。	否
attname	重写响应中的 Content-Disposition 头。 类型：字符串。 示例： attname=name1 下载对象重命名为“name1”。	否

请求消息头

该请求除使用公共消息头外，还可以使用附加的消息头来完成获取对象的功能，消息头的意义如表 5-34 所示。

表5-34 请求消息头

消息头名称	描述	是否必选
Range	获取对象时，获取在 Range 范围内的对	否

消息头名称	描述	是否必选
	象内容。如果 Range 不合法则忽略此字段获取整个对象。 Range 是一个范围，它的起始值最小为 0，最大为对象长度减 1。Range 范围的起始值为必填项，如果 Range 只包含起始值，表示获取起始值到对象长度减 1 这个区间的对象内容。 携带 Range 头域后，响应消息的 ETag 仍是对象的 ETag，而不是 Range 范围内对象的 ETag。 类型：字符串。 bytes=byte_range 示例 1：bytes=0-4 示例 2：bytes=1024 示例 3：bytes=10-20,30-40（表示多个区间）	
If-Modified-Since	如果对象在请求中指定的时间之后有修改，则返回对象内容；否则的话返回 304（not modified）。 类型：符合 http://www.ietf.org/rfc/rfc2616.txt 规定格式的 HTTP 时间字符串。	否
If-Unmodified-Since	如果对象在请求中指定的时间之后没有修改，则返回对象内容；否则的话返回 412（precondition failed）。 类型：符合 http://www.ietf.org/rfc/rfc2616.txt 规定格式的 HTTP 时间字符串。	否
If-Match	如果对象的 ETag 和请求中指定的 ETag 相同，则返回对象内容，否则的话返回 412（precondition failed）。 类型：字符串。 （ETag 值，例：0f64741bf7cb1089e988e4585d0d3434。）	否
If-None-Match	如果对象的 ETag 和请求中指定的 ETag 不相同，则返回对象内容，否则的话返回 304（not modified）。 类型：字符串。 （ETag 值，例：0f64741bf7cb1089e988e4585d0d3434。）	否

请求消息元素

该请求消息中不使用消息元素。

响应消息样式

```
HTTP/1.1 status code
Content-Type: type
7Date: date
Content-Length: length
Etag: etag
Last-Modified: time

<Object Content>
```

响应消息头

该请求的响应消息使用公共消息头，具体请参考表 3-20。

除公共响应消息头之外，还可能使用如下表 5-35 中的消息头。

表5-35 附加响应消息头

消息头名称	描述
x-obs-expiration	当对象单独设置了对象 lifecycle，过期时间以对象 lifecycle 为准，该消息头用 expiry-date 描述对象的详细过期信息；如果对象没有设置对象 lifecycle，设置了桶级别 lifecycle，过期时间以桶级别 lifecycle 为准，该消息头用 expiry-date 和 rule-id 两个键值对描述对象的详细过期信息；否则不显示该头域。 类型：字符串。
x-obs-website-redirect-location	当桶设置了 Website 配置，就可以设置对象元数据的这个属性，Website 接入点返回 301 重定向响应，将请求重定向到该属性指定的桶内的另一个对象或外部的 URL。 类型：字符串
x-obs-delete-marker	标识对象是否是删除标记。如果不是，则响应中不会出现该消息头。 类型：布尔值 有效值：true|false 默认值：false
x-obs-version-id	对象的版本号。如果该对象无版本号，则响应中不会出现该消息头。

消息头名称	描述
	有效值：字符串 默认值：无
x-obs-object-type	对象为非 Normal 对象时，会返回此头域，可取值为：Appendable。 类型：字符串
x-obs-next-append-position	对象为 Appendable 对象时，会返回此头域。 类型：整型

响应消息元素

该请求的响应消息中不带消息元素。

错误响应消息

无特殊错误，所有错误已经包含在表 6-3 中。

请求示例 1

下载整个对象

```
GET /object01 HTTP/1.1
User-Agent: curl/7.29.0
Host: examplebucket.obs.region.example.com
Accept: */*
Date: WED, 01 Jul 2015 04:24:33 GMT
Authorization: OBS H4IPJX0TQTHTHEBQQCEC:NxtSMS0jaVxlLnxlO9awaMTn47s=
```

响应示例 1

```
HTTP/1.1 200 OK
Server: OBS
x-obs-request-id: 8DF400000163D3F2A89604C49ABEE55E
Accept-Ranges: bytes
ETag: "3b46eaf02d3b6b1206078bb86a7b7013"
Last-Modified: WED, 01 Jul 2015 01:20:29 GMT
Content-Type: binary/octet-stream
x-obs-id-2: 32AAAQAAEAABAAAQAAEAABAAAQAAEAABCSQwxJ2I1VvxD/Xgwuw2G2RQax30gdXU
Date: WED, 01 Jul 2015 04:24:33 GMT
Content-Length: 4572

[4572 Bytes object content]
```

请求示例 2

指定 Range 下载对象（下载对象单个区间内容）

```
GET /object01 HTTP/1.1
User-Agent: curl/7.29.0
Host: examplebucket.obs.region.example.com
Accept: */*
Date: Mon, 14 Sep 2020 09:59:04 GMT
Range:bytes=20-30
Authorization: OBS H4IPJX0TQTHTHEBQQCEC:mNPLWQMDWg30PTkAWiqJaLl3ALg=
```

指定 Range 下载对象（下载对象多个区间内容）

```
GET /object01 HTTP/1.1
User-Agent: curl/7.29.0
Host: examplebucket.obs.region.example.com
Accept: */*
Date: Mon, 14 Sep 2020 10:02:43 GMT
Range:bytes=20-30,40-50
Authorization: OBS H4IPJX0TQTHTHEBQQCEC:ZwM7Vk2d7sD9o8zRsRKehgKQDkk=
```

响应示例 2

指定 Range 下载对象（下载对象单个区间内容）

```
HTTP/1.1 206 Partial Content
Server: OBS
x-obs-request-id: 000001748C0DBC35802E360C9E869F31
Accept-Ranges: bytes
ETag: "2200446c2082f27ed2a569601ca4e360"
Last-Modified: Mon, 14 Sep 2020 01:16:20 GMT
Content-Range: bytes 20-30/4583
Content-Type: binary/octet-stream
x-obs-id-2: 32AAAQAAEAABAAAQAAEAABAAAQAAEAABCSn2JHu4okx9NBRNZAvBGaws3lt3g31g
Date: Mon, 14 Sep 2020 09:59:04 GMT
Content-Length: 11

[ 11 Bytes object content]
```

指定 Range 下载对象（下载对象多个区间内容）

```
HTTP/1.1 206 Partial Content
Server: OBS
x-obs-request-id: 8DF400000163D3F2A89604C49ABEE55E
Accept-Ranges: bytes
ETag: "2200446c2082f27ed2a569601ca4e360"
Last-Modified: Mon, 14 Sep 2020 01:16:20 GMT
Content-Type: multipart/byteranges;boundary=35bcf444-e65f-4c76-9430-7e4a68dd3d26
x-obs-id-2: 32AAAQAAEAABAAAQAAEAABAAAQAAEAABCSIBWFOVW8eeWujkqSnoIANC2mNR1cdF
Date: Mon, 14 Sep 2020 10:02:43 GMT
Content-Length: 288

--35bcf444-e65f-4c76-9430-7e4a68dd3d26
Content-type: binary/octet-stream
Content-range: bytes 20-30/4583
[ 11 Bytes object content]
--35bcf444-e65f-4c76-9430-7e4a68dd3d26
Content-type: binary/octet-stream
Content-range: bytes 40-50/4583
```

```
[ 11 Bytes object content]
--35bcf444-e65f-4c76-9430-7e4a68dd3d26
```

请求示例 4

如果对象 Etag 值匹配则下载该对象

```
GET /object01 HTTP/1.1
User-Agent: curl/7.29.0
Host: examplebucket.obs.region.example.com
Accept: */*
Date: WED, 01 Jul 2015 04:24:33 GMT
If-Match: 682e760adb130c60c120da3e333a8b09
Authorization: OBS H4IPJX0TQTHTHEBQQCEC:NxtSMS0jaVxlLnxlO9awaMTn47s=
```

响应示例 4-1（Etag 不匹配）

如果存储的对象的 Etag 值不是 682e760adb130c60c120da3e333a8b09，则提示下载失败

```
HTTP/1.1 412 Precondition Failed
Server: OBS
x-obs-request-id: 8DF400000163D3F2A89604C49ABEE55E
Content-Type: application/xml
x-obs-id-2: 32AAAQAAEAABAAAQAAEAABAAAQAAEAABCSQwxJ2I1VvxD/Xgwuw2G2RQax30gdXU
Date: WED, 01 Jul 2015 04:20:51 GMT

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Error>
  <Code>PreconditionFailed</Code>
  <Message>At least one of the pre-conditions you specified did not hold</Message>
  <RequestId>5DEB00000164A214CEC54C30A3A769E3</RequestId>
  <HostId>Hw0ZGaSKVm+uLOrCXXtx4Qn1aLzvoeblctVXRAqA7pty10mzUUW/yOzFue04lBqu</HostId>
  <Condition>If-Match</Condition>
</Error>
```

响应示例 4-2（下载成功）

如果存储的对象的 Etag 值是 682e760adb130c60c120da3e333a8b09，则下载成功

```
HTTP/1.1 200 OK
Server: OBS
x-obs-request-id: 5DEB00000164A21E1FC826C58F6BA001
Accept-Ranges: bytes
ETag: "682e760adb130c60c120da3e333a8b09"
Last-Modified: Mon, 16 Jul 2015 08:03:34 GMT
Content-Type: application/octet-stream
x-obs-id-2: 32AAAQAAEAABAAAQAAEAABAAAQAAEAABCSbkdml1sLSvKnoHaRcOwRI+6+ustDwk
Date: Mon, 16 Jul 2015 08:04:00 GMT
Content-Length: 8

[ 8 Bytes object content]
```

请求示例 5

在 URL 中携带签名下载对象

```
GET
/object02?AccessKeyId=H4IPJX0TQTHTHEBQQCEC&Expires=1532688887&Signature=EQmDuOhWLUr
zrzRNZxwS72CXeXM%3D HTTP/1.1
User-Agent: curl/7.29.0
Host: examplebucket.obs.region.example.com
Accept: */*
Date: Fri, 27 Jul 2018 10:52:31 GMT
```

响应示例 5

```
HTTP/1.1 200 OK
Server: OBS
x-obs-request-id: 804F00000164DB5E5B7FB908D3BA8E00
ETag: "682e760adb130c60c120da3e333a8b09"
Last-Modified: Mon, 16 Jul 2015 08:03:34 GMT
Content-Type: application/octet-stream
x-obs-id-2: 32AAAUJAIAABAAAQAAEAABAAAQAAEAABCTlpxILjhVK/heKOWIP8Wn2IWmQoerfw
Date: Fri, 27 Jul 2018 10:52:31 GMT
Content-Length: 8

[ 8 Bytes object content]
```

请求示例 6

下载对象并重命名，使用 response-content-disposition 参数实现

```
GET /object01?response-content-disposition=attachment; filename*=utf-8''name1
HTTP/1.1
User-Agent: curl/7.29.0
Host: examplebucket.obs.region.example.com
Accept: */*
Date: WED, 01 Jul 2015 04:24:33 GMT
Authorization: OBS H4IPJX0TQTHTHEBQQCEC:NxtSMS0jaVxlLnxlO9awaMTn47s=
```

响应示例 6

```
HTTP/1.1 200 OK
Server: OBS
x-obs-request-id: 804F00000164DB5E5B7FB908D3BA8E00
ETag: "682e760adb130c60c120da3e333a8b09"
Last-Modified: Mon, 16 Jul 2015 08:03:34 GMT
Content-Type: application/octet-stream
x-obs-id-2: 32AAAUJAIAABAAAQAAEAABAAAQAAEAABCTlpxILjhVK/heKOWIP8Wn2IWmQoerfw
Date: Fri, 27 Jul 2018 10:52:31 GMT
Content-Length: 8
Content-Disposition: attachment; filename*=utf-8''name1

[ 8 Bytes object content]
```

请求示例 7

下载对象并重命名，使用 attname 参数实现

```
GET /object01?attname=name1 HTTP/1.1
User-Agent: curl/7.29.0
Host: examplebucket.obs.region.example.com
Accept: */*
Date: WED, 01 Jul 2015 04:24:33 GMT
Authorization: OBS H4IPJX0TQTHTHEBQQCEC:NxtSMS0jaVxlLnxlO9awaMTn47s=
```

响应示例 7

```
HTTP/1.1 200 OK
Server: OBS
x-obs-request-id: 804F00000164DB5E5B7FB908D3BA8E00
ETag: "682e760adb130c60c120da3e333a8b09"
Last-Modified: Mon, 16 Jul 2015 08:03:34 GMT
Content-Type: application/octet-stream
x-obs-id-2: 32AAAUJAIAABAAAQAAEAABAAAQAAEAABCTlpxILjhVK/heKOWIP8Wn2IWmQoerfw
Date: Fri, 27 Jul 2018 10:52:31 GMT
Content-Length: 8
Content-Disposition: attachment; filename*=utf-8''name1

[ 8 Bytes object content]
```

5.3.5 获取对象元数据

功能介绍

拥有对象读权限的用户可以执行 HEAD 操作命令获取对象元数据，返回信息包含对象的元数据信息。

多版本

默认情况下，获取的是最新版本的对象元数据。如果最新版本的对象是删除标记，则返回 404。如果要获取指定版本的对象元数据，请求可携带 versionId 消息参数。

请求消息样式

```
HEAD /ObjectName HTTP/1.1
Host: bucketname.obs.region.example.com
Date: date
Authorization: authorization
```

请求消息参数

请求参数说明如表 5-36 所示。

表5-36 请求消息参数

参数名称	描述	是否必选

参数名称	描述	是否必选
versionId	对象的版本号。 类型：字符串	否

请求消息头

该请求使用公共消息头，具体请参考表 3-3。

另外该请求可以使用附加的消息头，具体如表 5-37 所示。

表5-37 请求消息头

消息头名称	描述	是否必选
Origin	预请求指定的跨域请求 Origin（通常为域名）。 类型：字符串	是
Access-Control-Request-Headers	实际请求可以带的 HTTP 头域，可以带多个头域。 类型：字符串	否

请求消息元素

该请求消息中不使用消息元素。

响应消息样式

```
HTTP/1.1 status code
Content-Type: type
Date: date
Content-Length: length
Etag: etag
Last-Modified: time
```

响应消息头

该请求的响应消息使用公共消息头，具体请参考表 3-20。

除公共响应消息头之外，还可能使用如下表 5-38 中的消息头。

表5-38 附加响应消息头

消息头名称	描述
x-obs-expiration	当对象单独设置了对象 lifecycle，过期时间以对象 lifecycle 为准，该消息头用 expiry-date 描述对象的详细过期信息；如果对象没有设置对象

消息头名称	描述
	lifecycle，设置了桶级别 lifecycle，过期时间以桶级别 lifecycle 为准，该消息头用 expiry-date 和 rule-id 两个键值对描述对象的详细过期信息；否则不显示该头域。 类型：字符串。
x-obs-website-redirect-location	当桶设置了 Website 配置，就可以设置对象元数据的这个属性，Website 接入点返回 301 重定向响应，将请求重定向到该属性指定的桶内的另一个对象或外部的 URL。 类型：字符串
x-obs-version-id	对象的版本号。如果该对象无版本号，则响应中不会出现该消息头。 类型：字符串 默认值：无
Access-Control-Allow-Origin	当桶设置了 CORS 配置，如果请求的 Origin 满足服务端的 CORS 配置，则在响应中包含这个 Origin。 类型：字符串
Access-Control-Allow-Headers	当桶设置了 CORS 配置，如果请求的 headers 满足服务端的 CORS 配置，则在响应中包含这个 headers。 类型：字符串
Access-Control-Max-Age	当桶设置了 CORS 配置，服务端 CORS 配置中的 MaxAgeSeconds。 类型：整数
Access-Control-Allow-Methods	当桶设置了 CORS 配置，如果请求的 Access-Control-Request-Method 满足服务端的 CORS 配置，则在响应中包含这条 rule 中的 Methods。 类型：字符串 有效值：GET、PUT、HEAD、POST 、DELETE
Access-Control-Expose-Headers	当桶设置了 CORS 配置，服务端 CORS 配置中的 ExposeHeader。 类型：字符串
x-obs-object-type	对象为非 Normal 对象时，会返回此头域，可取值为：Appendable 类型：字符串
x-obs-next-append-position	对象为 Appendable 对象时，会返回此头域

消息头名称	描述
	类型：整型

响应消息元素

该请求的响应消息中不带消息元素。

错误响应消息

无特殊错误；所有错误已经包含在表 6-3 中。

请求示例

```
HEAD /object1 HTTP/1.1
User-Agent: curl/7.29.0
Host: examplebucket.obs.region.example.com
Accept: */*
Date: WED, 01 Jul 2015 04:19:25 GMT
Authorization: OBS H4IPJX0TQTHTHEBQQCEC:/cARjk81l2iExMfQqn6iT3qEZ74=
```

响应示例

```
HTTP/1.1 200 OK
Server: OBS
x-obs-request-id: 8DF400000163D3E4BB5905C41B6E65B6
Accept-Ranges: bytes
ETag: "3b46eaf02d3b6b1206078bb86a7b7013"
Last-Modified: WED, 01 Jul 2015 01:19:21 GMT
Content-Type: binary/octet-stream
x-obs-id-2: 32AAAQAAEAABAAAQAAEAABAAAQAAEAABCSD3nAiTaBoeyt9oHp9vTYtXnLDmwV6D
Date: WED, 01 Jul 2015 04:19:21 GMT
Content-Length: 4572
```

5.3.6 删除对象

功能介绍

删除对象的操作。如果要删除的对象不存在，则仍然返回成功信息。

多版本

当桶的多版本状态是开启时，不指定版本删除对象将产生一个带唯一版本号的删除标记，并不删除对象；当桶的多版本状态是 Suspended 时，不指定版本删除将删除版本号为 null 的对象，并将产生一个版本号为 null 的删除标记。

如果要删除指定版本的对象，请求可携带 versionId 消息参数。

请求消息样式

```
DELETE /ObjectName HTTP/1.1
Host: bucketname.obs.region.example.com
Date: date
Authorization: authorization
```

请求消息参数

请求参数说明如表 5-39 所示。

须知

删除对象时，请求消息参数仅支持表 5-39 中列出的参数信息，如果包含了 OBS 无法识别的参数信息，服务端将返回 400 错误。

表5-39 请求消息参数

参数名称	描述	是否必选
versionId	待删除对象的版本号。 类型：字符串	否

请求消息头

该请求使用公共消息头，具体请参考表 3-3。

请求消息元素

该请求消息中不使用消息元素。

响应消息样式

```
HTTP/1.1 status_code
Date: date
```

响应消息头

该请求的响应消息使用公共消息头，具体请参考表 3-20。

除公共响应消息头之外，还可能使用如下表 5-40 中的消息头。

表5-40 附加响应消息头

消息头名称	描述
x-obs-delete-marker	标识对象是否标记删除。如果不是，则响应中不会出现该消息头。

消息头名称	描述
	类型：布尔值 有效值：true|false 默认值：false
x-obs-version-id	对象的版本号。如果该对象无版本号，则响应中不会出现该消息头。 有效值：字符串 默认值：无

响应消息元素

该请求的响应消息中不带消息元素。

错误响应消息

无特殊错误，所有错误已经包含在表 6-3 中。

请求示例

```
DELETE /object2 HTTP/1.1
User-Agent: curl/7.29.0
Host: examplebucket.obs.region.example.com
Accept: */*
Date: WED, 01 Jul 2015 04:19:21 GMT
Authorization: OBS H4IPJX0TQTHTHEBQQCEC:MfK9JCnSFHCrJmjv7iRkRrrce2s=
```

响应示例

```
HTTP/1.1 204 No Content
Server: OBS
x-obs-request-id: 8DF400000163D3F51DEA05AC9CA066F1
x-obs-id-2: 32AAAUgAIAABAAAQAAEAABAAAQAAEAABCSgkM4Dij80gAeFY8pAZIwx72QhDeBZ5
Date: WED, 01 Jul 2015 04:19:21 GMT
```

5.3.7 批量删除对象

功能介绍

批量删除对象特性用于将一个桶内的部分对象一次性删除，删除后不可恢复。批量删除对象要求返回结果里包含每个对象的删除结果。OBS 的批量删除对象使用同步删除对象的方式，每个对象的删除结果返回给请求用户。

批量删除对象支持两种响应方式：verbose 和 quiet。Verbose 是指在返回响应时，不管对象是否删除成功都将删除结果包含在 XML 响应里；quiet 是指在返回响应时，只返回删除失败的对象结果，没有返回的认为删除成功。OBS 默认使用 verbose 模式，如果用户在请求消息体中指定 quiet 模式的话， 使用 quiet 模式。

批量删除的请求消息头中必须包含 Content-MD5 以及 Content-Length，用以保证请求的消息体在服务端检测到网络传输如果有错，则可以检测出来。

请求消息样式

```
POST /?delete HTTP/1.1
Host: bucketname.obs.region.example.com
Authorization: authorization
Content-MD5: MD5
Content-Length: length

<?xml version="1.0" encoding="UTF-8"?>
<Delete>
   <Quiet>true</Quiet>
   <Object>
      <Key>Key</Key>
      <VersionId>VersionId</VersionId>
   </Object>
   <Object>
      <Key>Key</Key>
   </Object>
</Delete>
```

请求消息参数

该请求的请求消息中不使用消息参数。

请求消息头

该请求使用公共消息头，具体请参考表 3-3。

请求消息元素

该请求通过在请求消息的消息元素中指定要批量删除的对象列表，元素的具体含义如表 5-41 所示。

表5-41 请求消息元素

元素名称	描述	是否必须
Quiet	用于指定使用 quiet 模式，只返回删除失败的对象结果；如果有此字段，则必被置为 True，如果为 False 则被系统忽略掉。 类型：布尔值。	否
Delete	待删除的对象列表。 类型：XML。	是
Object	待删除的对象。 类型：XML。	是
Key	待删除的对象 Key。	是

元素名称	描述	是否必须
	类型：字符串。	
VersionId	待删除的对象版本号。 类型：字符串。	否

批量删除对象一次能接收最大对象数目为 1000 个，如果超出限制，服务端会返回请求不合法。

并发任务分配后，在循环删除多个对象过程中， 如果发生内部错误， 有可能出现数据不一致的情况（某个对象索引数据删除但还有元数据）。

响应消息样式

```
HTTP/1.1 status code
Date: date
Content-Type: application/xml
Content-Length: length

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<DeleteResult xmlns="http://obs.region.example.com/doc/2015-06-30/">
<Deleted>
   <Key>Key</Key>
</Deleted>
<Error>
   <Key>Key</Key>
   <Code>ErrorCode</Code>
   <Message>messagee</Message>
</Error>
</DeleteResult>
```

响应消息头

该请求的响应消息使用公共消息头，具体请参考表 3-20。

响应消息元素

该请求的响应通过消息元素返回删除的结果，元素的具体意义如表 5-42 中所示。

表5-42 响应消息元素

字段名称	描述
DeleteResult	批删响应的根节点。 类型：容器。
Deleted	删除成功结果的 Container。 类型：容器。
Error	删除失败结果的 Container。

字段名称	描述
	类型：容器。
Key	每个删除结果的对象名。 类型：字符串。
Code	删除失败结果的错误码。 类型：字符串。
Message	删除失败结果的错误消息。 类型：字符串。
VersionId	删除对象的版本号 类型：字符串。
DeleteMarker	当批量删除请求访问的桶是多版本桶时，如果创建或删除一个删除标记，返回消息中该元素的值为 true。 类型：布尔值。
DeleteMarkerVersionId	请求创建或删除的删除标记版本号。 当批量删除请求访问的桶是多版本桶时，如果创建或删除一个删除标记，响应消息会返回该元素。该元素在以下两种情况中会出现： • 用户发送不带版本删除请求，即请求只有对象名，无版本号。这种情况下，系统会创建一个删除标记，并在响应中返回该删除标记的版本号。 • 用户发送带版本删除请求，即请求同时包含对象名以及版本号，但是该版本号标识一个删除标记。这种情况下，系统会删除此删除标记，并在响应中返回该删除标记的版本号。 类型：布尔值。

错误响应消息

1、用户收到请求后首先进行 XML 的解析，如果超过 1000 个对象返回 400 Bad Request。

2、如果 XML 消息体中包含的对象 Key 不合法，比如超过 1024 字节，OBS 返回 400 Bad Request。

3、如果请求消息头中不包含 Content-MD5，OBS 则返回 400 Bad Request。

其他错误已经包含在表 6-3 中。

请求示例

```
POST /test333?delete HTTP/1.1
User-Agent: curl/7.29.0
```

```
Host: 127.0.0.1
Accept: */*
Date: WED, 01 Jul 2015 04:34:21 GMT
Authorization: OBS H4IPJX0TQTHTHEBQQCEC:8sjZWJlWmYmYnK5JqXaFFQ+vHEg=
Content-MD5: ZPzz8L+hdRJ6qCqYbU/pCw==
Content-Length: 188

<?xml version="1.0" encoding="utf-8"?>
<Delete>
  <Quiet>true</Quiet>
  <Object>
    <Key>obja02</Key>
  </Object>
  <Object>
    <Key>obja02</Key>
  </Object>
</Delete>
```

响应示例

```
HTTP/1.1 200 OK
Server: OBS
x-obs-request-id: 8DF400000163D3FE4CE80340D30B0542
x-obs-id-2: 32AAAQAAEAABAAAQAAEAABAAAQAAEAABCRhY0FBWRm6qjOE1ACBZwS+0KYlPBq0f
Content-Type: application/xml
Date: WED, 01 Jul 2015 04:34:21 GMT
Content-Length: 120

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<DeleteResult xmlns="http://obs.example.com/doc/2015-06-30/"/>
```

5.3.8 设置对象 ACL

功能介绍

OBS 支持对对象的操作进行权限控制。默认情况下，只有对象的创建者才有该对象的读写权限。用户也可以设置其他的访问策略，比如对一个对象可以设置公共访问策略，允许所有人对其都有读权限。

OBS 用户在上传对象时可以设置权限控制策略，也可以通过 ACL 操作 API 接口对已存在的对象更改或者获取 ACL(access control list) 。

本节将介绍如何更改对象 ACL，改变对象的访问权限。

多版本

默认情况下，更改的是最新版本的对象 ACL。要设置指定版本的对象 ACL，请求可以带参数 versionId。

请求消息格式

```
PUT /ObjectName?acl HTTP/1.1
Host: bucketname.obs.region.example.com
Date: date
```

```
Authorization: authorization

<AccessControlPolicy>
   <Owner>
      <ID>ID</ID>
   </Owner>
   <Delivered>true</Delivered>
   <AccessControlList>
      <Grant>
         <Grantee>
            <ID>ID</ID>
         </Grantee>
         <Permission>permission</Permission>
      </Grant>
   </AccessControlList>
</AccessControlPolicy>
```

请求消息参数

请求参数说明如表 5-43 所示。

表5-43 请求消息参数

参数名称	描述	是否必选
versionId	对象的版本号。标示更改指定版本的对象 ACL。 类型：字符串	否

请求消息头

该请求使用公共请求消息头，具体参见表 3-3。

请求消息元素

该请求消息通过带消息元素来传递对象的 ACL 信息，元素的意义如表 5-44 所示。

表5-44 请求消息元素

元素名称	描述	是否必选
ID	用户 DomainId。 类型：字符串。	否
Delivered	对象 ACL 是否继承桶的 ACL。 类型：布尔类型。默认 true。	否
Permission	授予权限。 类型：枚举类型。	否

响应消息样式

```
HTTP/1.1 status code
Content-Length: length
Content-Type: application/xml
```

响应消息头

该请求的响应消息使用公共消息头，具体请参考表 3-20。

除公共响应消息头之外，还可能使用如表 5-45 中的消息头。

表5-45 附加响应消息头

消息头名称	描述
x-obs-version-id	被更改 ACL 的对象的版本号。 类型：字符串

响应消息元素

该请求的响应消息中不带有消息元素。

错误响应消息

该请求的响应无特殊错误，所有错误已经包含在表 6-3 中。

请求示例

```
PUT /obj2?acl HTTP/1.1
User-Agent: curl/7.29.0
Host: examplebucket.obs.region.example.com
Accept: */*
Date: WED, 01 Jul 2015 04:42:34 GMT
Authorization: OBS H4IPJX0TQTHTHEBQQCEC:8xAODun1ofjkwHm8YhtN0QEcy9M=
Content-Length: 727

<AccessControlPolicy xmlns="http://obs.example.com/doc/2015-06-30/">
  <Owner>
    <ID>b4bf1b36d9ca43d984fbcb9491b6fce9</ID>
  </Owner>
  <Delivered>false</Delivered>
  <AccessControlList>
    <Grant>
      <Grantee>
        <ID>b4bf1b36d9ca43d984fbcb9491b6fce9</ID>
      </Grantee>
      <Permission>FULL CONTROL</Permission>
    </Grant>
    <Grant>
      <Grantee>
        <ID>783fc6652cf246c096ea836694f71855</ID>
```

```
    </Grantee>
    <Permission>READ</Permission>
  </Grant>
  <Grant>
    <Grantee>
      <Canned>Everyone</Canned>
    </Grantee>
    <Permission>READ</Permission>
  </Grant>
 </AccessControlList>
</AccessControlPolicy>
```

响应示例

```
HTTP/1.1 200 OK
Server: OBS
x-obs-request-id: 8DF400000163D3F0FD2A03D2D30B0542
x-obs-id-2: 32AAAUgAIAABAAAQAAEAABAAAQAAEAABCTjCqTmsA1XRpIrmrJdvcEWvZyjbztdd
Date: WED, 01 Jul 2015 04:42:34 GMT
Content-Length: 0
```

5.3.9 获取对象 ACL

功能介绍

用户执行获取对象 ACL 的操作，返回信息包含指定对象的权限控制列表信息。用户必须拥有对指定对象读 ACP(access control policy)的权限，才能执行获取对象 ACL 的操作。

多版本

默认情况下，获取最新版本的对象 ACL。如果最新版本的对象是删除标记，则返回 404。如果要获取指定版本的对象 ACL，请求可携带 versionId 消息参数。

请求消息样式

```
GET /ObjectName?acl HTTP/1.1
Host: bucketname.obs.region.example.com
Date: date
Authorization: authorization
```

请求消息参数

该请求需要在请求消息参数中指定是在获取对象 ACL，参数意义如表 5-46 所示。

表5-46 请求消息参数

参数名称	描述	是否必选
acl	指定该请求是获取对象的 acl。 类型：字符串。	是

参数名称	描述	是否必选
versionId	指定对象的版本号。 类型：字符串。	否

请求消息头

该请求使用公共消息头，具体请参考表 3-3。

请求消息元素

该请求消息中不使用消息元素。

响应消息样式

```
HTTP/1.1 status code
Date: date
Content-Length: length
Content-Type: application/xml

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<AccessControlPolicy xmlns="http://obs.region.example.com/doc/2015-06-30/">
   <Owner>
      <ID>id</ID>
   </Owner>
   <Delivered>true</Delivered>
   <AccessControlList>
      <Grant>
         <Grantee>
            <ID>id</ID>
         </Grantee>
         <Permission>permission</Permission>
      </Grant>
   </AccessControlList>
</AccessControlPolicy>
```

响应消息头

该请求的响应消息使用公共消息头，具体请参考表 3-20。

除公共响应消息头之外，还可能使用如下表 5-47 中的消息头。

表5-47 附加响应消息头

消息头名称	描述
x-obs-version-id	指定对象的版本号。 有效值：字符串 默认值：无

响应消息元素

该请求的响应消息中通过消息元素返回对象的 ACL 信息，元素的具体意义如表 5-48 所示。

表5-48 响应消息元素

元素名称	描述
ID	用户的 DomainId。 类型：字符串。
AccessControlList	访问控制列表，记录了对该桶有访问权限的用户列表。 类型：XML。
Grant	用于标记用户及用户的权限。 类型：XML。
Grantee	记录用户信息。 类型：XML。
Delivered	对象 ACL 是否继承桶的 ACL。 类型：布尔类型。
Permission	指定的用户对该对象所具有的操作权限。 类型：字符串。

错误响应消息

无特殊错误，所有错误已经包含在表 6-3 中。

请求示例

```
GET /object011?acl HTTP/1.1
User-Agent: curl/7.29.0
Host: examplebucket.obs.region.example.com
Accept: */*
Date: WED, 01 Jul 2015 04:45:55 GMT
Authorization: OBS H4IPJX0TQTHTHEBQQCEC:YcmvNQxItGjFeeC1K2HeUEp8MMM=
```

响应示例

```
HTTP/1.1 200 OK
Server: OBS
x-obs-request-id: 8DF400000163D3E650F3065C2295674C
x-obs-id-2: 32AAAQAAEAABAAAQAAEAABAAAQAAEAABCS+wsHqRuA2Tx+mXUpNtBbWLPMle9CIx
Content-Type: application/xml
```

```
Date: WED, 01 Jul 2015 04:45:55 GMT
Content-Length: 769

<?xml version="1.0" encoding="utf-8"?>
<AccessControlPolicy xmlns="http://obs.region.example.com/doc/2015-06-30/">
  <Owner>
    <ID>b4bf1b36d9ca43d984fbcb9491b6fce9</ID>
  </Owner>
  <Delivered>false</Delivered>
  <AccessControlList>
    <Grant>
      <Grantee>
        <ID>b4bf1b36d9ca43d984fbcb9491b6fce9</ID>
      </Grantee>
      <Permission>FULL_CONTROL</Permission>
    </Grant>
    <Grant>
      <Grantee>
        <ID>783fc6652cf246c096ea836694f71855</ID>
      </Grantee>
      <Permission>READ</Permission>
      </Grant>
    <Grant>
      <Grantee>
        <Canned>Everyone</Canned>
      </Grantee>
      <Permission>READ ACP</Permission>
    </Grant>
  </AccessControlList>
</AccessControlPolicy>
```

5.3.10 修改对象元数据

功能介绍

用户可以通过本接口添加、修改或删除桶中已经上传的对象的元数据。

请求消息样式

```
PUT /ObjectName?metadata HTTP/1.1
Host: bucketname.obs.region.example.com
Content-Type: application/xml
Content-Length: length
Authorization: authorization
Date: date
<Optional Additional Header>
<object Content>
```

请求消息参数

表5-49 请求消息参数

参数名称	描述	是否必选

参数名称	描述	是否必选
versionId	对象的版本号。 类型：字符串	否

请求消息头

说明

OBS 支持在修改对象元数据的请求里携带 HTTP 协议规定的 6 个请求头：Cache-Control、Expires、Content-Encoding、Content-Disposition、Content-Type、Content-Language，OBS 会直接将这些头域的值保存在对象元数据中，在下载对象或者 HEAD 对象的时候，这些保存的值将会被设置到对应的 HTTP 头域中返回客户端。

表5-50 请求消息头

消息头名称	描述	是否必选
x-obs-metadata-directive	元数据操作指示符。 取值为 REPLACE_NEW 或 REPLACE。 REPLACE_NEW 表示：对于已经存在值的元数据进行替换，不存在值的元数据进行赋值，未指定的元数据保持不变。 REPLACE 表示：使用当前请求中携带的头域完整替换，未指定的元数据（本表中除 x-obs-storage-class 的其它元数据）会被删除。 类型：字符串	是
Cache-Control	指定 Object 被下载时的网页的缓存行为。 类型：字符串	否
Content-Disposition	指定 Object 被下载时的名称。 类型：字符串	否
Content-Encoding	指定 Object 被下载时的内容编码格式。 类型：字符串	否
Content-Language	指定 Object 被下载时的内容语言格式。 类型：字符串	否
Content-Type	Object 文件类型。 类型：字符串	否
Expires	指定 Object 被下载时的网页的缓存过期时间。 类型：字符串	否
x-obs-website-redirect-	当桶设置了 Website 配置，可以将获取这个对象的请求重定向到桶内另一个对象或一个外部的 URL。	否

消息头名称	描述	是否必选
location	例如，重定向请求到桶内另一对象： x-obs-website-redirect-location:/anotherPage.html 或重定向请求到一个外部 URL： x-obs-website-redirect-location:http://www.example.com/ 类型：字符串 约束：必须以“/”、“http://”或“https://”开头，长度不超过 2KB	
x-obs-meta-*	可以在请求中加入以“x-obs-meta-”开头的消息头，用来加入自定义的元数据，以便对对象进行自定义管理。当用户获取此对象或查询此对象元数据时，加入的自定义元数据将会在返回消息的头中出现。HTTP 请求不包含消息体，长度不能超过 8KB。 类型：字符串 示例：x-obs-meta-test: test metadata	否

请求消息元素

该请求消息中不使用消息元素。

响应消息样式

```
HTTP/1.1 status code
Content-Type: type
Date: date
Content-Length: length
Etag: etag
Last-Modified: time
```

响应消息头

表5-51 附加响应消息头

消息头名称	描述
x-obs-metadata-directive	元数据操作指示符。 取值为 REPLACE_NEW 或 REPLACE。 类型：字符串
Cache-Control	指定 Object 被下载时的网页的缓存行为。如果请求携带了此头域，那么响应的消息中应该包含此消息头。 类型：字符串
Content-	指定 Object 被下载时的名称。如果请求携带了此头域，那么响

消息头名称	描述
Disposition	应的消息中应该包含此消息头。 类型：字符串
Content-Encoding	指定 Object 被下载时的内容编码格式。如果请求携带了此头域，那么响应的消息中应该包含此消息头。 类型：字符串
Content-Language	指定 Object 被下载时的内容语言格式。如果请求携带了此头域，那么响应的消息中应该包含此消息头。 类型：字符串
Content-Type	Object 文件类型。如果请求携带了此头域，那么响应的消息中应该包含此消息头。 类型：字符串
Expires	指定 Object 被下载时的网页的缓存过期时间。如果请求携带了此头域，那么响应的消息中应该包含此消息头。 类型：字符串
x-obs-website-redirect-location	当桶设置了 Website 配置，可以将获取这个对象的请求重定向到桶内另一个对象或一个外部的 URL。如果请求携带了此头域，那么响应的消息中应该包含此消息头。 类型：字符串
x-obs-meta-*	用户自定义的元数据，以便对对象进行自定义管理。如果请求携带了此头域，那么响应的消息中应该包含此消息头。 类型：字符串

响应消息元素

该请求的响应消息中不带消息元素。

错误响应消息

无特殊错误；所有错误已经包含在表 6-3 中。

请求示例 1

添加对象元数据

给对象 object 添加元数据：Content-Type:application/zip 和 x-obs-meta-test:meta。

```
PUT /object?metadata HTTP/1.1
User-Agent: curl/7.29.0
Host: examplebucket.obs.region.example.com
Accept: */*
Date: WED, 01 Jul 2015 14:24:33 GMT
```

```
Authorization: OBS H4IPJX0TQTHTHEBQQCEC:NxtSMS0jaVxlLnxlO9awaMTn47s=
x-obs-metadata-directive:REPLACE NEW
Content-Type:application/zip
x-obs-meta-test:meta
```

响应示例 1

```
HTTP/1.1 200 OK
Server: OBS
x-obs-request-id: 8DF400000163D3E4BB5905C41B6E65B6
Accept-Ranges: bytes
Content-Type: binary/octet-stream
x-obs-id-2: 32AAAQAAEAABAAAQAAEAABAAAQAAEAABCSD3nAiTaBoeyt9oHp9vTYtXnLDmwV6D
Date: WED, 01 Jul 2015 04:19:21 GMT
Content-Length: 0
x-obs-metadata-directive:REPLACE NEW
Content-Type:application/zip
x-obs-meta-test:meta
```

5.4 多段操作

5.4.1 列举桶中已初始化多段任务

功能介绍

用户可以通过本接口查询一个桶中所有的初始化后还未合并以及未取消的多段上传任务。

请求消息样式

```
GET /?uploads&max-uploads=max HTTP/1.1
Host: bucketname.obs.region.example.com
Date: date
Authorization: authorization
```

请求消息参数

该请求可以通过在请求消息中指定参数，查询指定范围的多段上传任务，请求参数说明如表 5-52 所示。

表5-52 请求消息参数

参数名称	描述	是否必选
delimiter	对于名字中包含 delimiter 的对象的任务，其对象名（如果请求中指定了 prefix，则此处的对象名需要去掉 prefix）中从首字符至第一个 delimiter 之间的字符串将作为 CommonPrefix 在响应中返回。对象名包含 CommonPrefix 的任务被视为一个分组，作为一条记录在响应中返回，该记录不包含任务的信	否

参数名称	描述	是否必选
	息，仅用于提示用户该分组下存在多段上传任务。 类型：字符串。	
prefix	如果请求中指定了 prefix，则响应中仅包含对象名以 prefix 开始的任务信息。 类型：字符串。	否
max-uploads	列举的多段任务的最大条目，取值范围为[1,1000]，当超出范围时，按照默认的 1000 进行处理。 类型：整型。	否
key-marker	列举时返回指定的 key-marker 之后的多段任务。 类型：字符串。	否
upload-id-marker	只有和 key-marker 一起使用才有意义， 列举时返回指定的 key-marker 的 upload-id-marker 之后的多段任务。 类型：字符串。	否

请求消息头

该请求使用公共请求消息头，具体请见表 3-3。

请求消息元素

该请求消息中不使用消息元素。

响应消息样式

```
HTTP/1.1 status_code
Date: date
Content-Length: length

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ListMultipartUploadsResult xmlns="http://obs.region.example.com/doc/2015-06-30/">
   <Bucket>bucketname</Bucket>
   <KeyMarker/>
   <UploadIdMarker/>
   <NextKeyMarker>nextMarker</NextKeyMarker>
   <NextUploadIdMarker>idMarker</NextUploadIdMarker>
   <MaxUploads>maxUploads</MaxUploads>
   <IsTruncated>true</IsTruncated>
   <Upload>
      <Key>key</Key>
      <UploadId>uploadID</UploadId>
      <Initiator>
         <ID>domainID/domainID:userID/userID</ID>
      </Initiator>
```

```
      <Owner>
         <ID>ownerID</ID
      </Owner>

      <Initiated>initiatedDate</Initiated>
   </Upload>
</ListMultipartUploadsResult>
```

响应消息头

该请求的响应消息使用公共消息头，具体请参考表 3-20。

响应消息元素

该请求的响应消息中通过消息元素返回多段上传任务，元素的具体意义如表 5-53 所示。

表5-53 响应消息元素

元素名称	描述
ListMultipartUploadsResult	保存 List Multipart Upload 请求结果的容器。 类型：容器。 子节点： Bucket, KeyMarker , UploadIdMarker, NextKeyMarker, NextUploadIdMarker, MaxUploads, Delimiter, Prefix, Upload, CommonPrefixes, IsTruncated。 父节点：None。
Bucket	初始化任务所在的桶名。 类型：字符串。 父节点: ListMultipartUploadsResult。
KeyMarker	列举时的起始对象位置。 类型：字符串。 父节点: ListMultipartUploadsResult。
UploadIdMarker	列举时的起始 UploadId 位置。 类型：字符串。 父节点: ListMultipartUploadsResult。
NextKeyMarker	如果本次没有返回全部结果，响应请求中将包含 NextKeyMarker 字段，用于标明接下来请求的 KeyMarker 值。 类型：字符串。 父节点: ListMultipartUploadsResult。
NextUploadIdMarker	如果本次没有返回全部结果，响应请求中将包含 NextUploadMarker 字段，用于标明接下来请求的 UploadMarker 值。

元素名称	描述
	类型: 字符串。 父节点: ListMultipartUploadsResult。
MaxUploads	返回的最大多段上传任务数目。 类型：整型。 父节点: ListMultipartUploadsResult。
IsTruncated	表明是否本次返回的 Multipart Upload 结果列表被截断。“true”表示本次没有返回全部结果；“false”表示本次已经返回了全部结果. 类型: Boolean。 父节点: ListMultipartUploadsResult。
Upload	保存 Multipart Upload 任务信息的容器。 类型：容器。 子节点: Key, UploadId, InitiatorOwner, Initiated。 父节点: ListMultipartUploadsResult。
Key	初始化 Multipart Upload 任务的 Object 名字。 类型：字符串。 父节点: Upload。
UploadId	Multipart Upload 任务的 ID。 类型:：字符串。 父节点: Upload。
Initiator	Multipart Upload 任务的创建者。 子节点: ID。 类型：容器。 父节点: Upload。
ID	创建者的 DomainId。 类型：字符串。 父节点: Initiator, Owner。
Owner	段的所有者。 类型:容器。 子节点: ID 父节点: Upload。
Initiated	Multipart Upload 任务的初始化时间。 类型: Date。 父节点: Upload。

元素名称	描述
ListMultipartUploadsResult.Prefix	请求中带的 Prefix。 类型：字符串。 父节点: ListMultipartUploadsResult。
Delimiter	请求中带的 Delimiter。 类型：字符串。 父节点: ListMultipartUploadsResult。
CommonPrefixes	请求中带 Delimiter 参数时，返回消息带 CommonPrefixes 分组信息。 类型：容器。 父节点: ListMultipartUploadsResult。
CommonPrefixes. Prefix	CommonPrefixes 分组信息中，表明不同的 Prefix。 类型：字符串。 父节点: CommonPrefixes。

错误响应消息

OBS 系统对 maxUploads 进行判断，如果 maxUploads 不为整数类型或者小于 0，OBS 返回 400 Bad Request。

其他错误已经包含在表 6-3 中。

请求示例 1

不带任何参数列举已初始化的段任务

```
GET /?uploads HTTP/1.1
User-Agent: curl/7.29.0
Host: examplebucket.obs.region.example.com
Accept: */*
Date: WED, 01 Jul 2015 04:51:21 GMT
Authorization: OBS UDSIAMSTUBTEST000008:XdmZgYQ+ZVy1rjxJ9/KpKq+wrU0=
```

响应示例 1

```
HTTP/1.1 200 OK
Server: OBS
x-obs-request-id: 8DF400000163D405534D046A2295674C
x-obs-id-2: 32AAAQAAEAABAAAQAAEAABAAAQAAEAABCSDaHP+a+Bp0RI6Mm9XvCOrf7q3qvBQW
Content-Type: application/xml
Date: WED, 01 Jul 2015 04:51:21 GMT
Content-Length: 681

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ListMultipartUploadsResult xmlns="http://obs.example.com/doc/2015-06-30/">
```

```
  <Bucket>examplebucket</Bucket>
  <KeyMarker/>
  <UploadIdMarker/>
  <Delimiter/>
  <Prefix/>
  <MaxUploads>1000</MaxUploads>
  <IsTruncated>false</IsTruncated>
  <Upload>
    <Key>obj2</Key>
    <UploadId>00000163D40171ED8DF4050919BD02B8</UploadId>
    <Initiator>

<ID>domainID/b4bf1b36d9ca43d984fbcb9491b6fce9:userID/71f390117351534r88115ea2c26d19
99</ID>
    </Initiator>
    <Owner>
      <ID>b4bf1b36d9ca43d984fbcb9491b6fce9</ID>
    </Owner>
    <Initiated>2015-07-01T02:30:54.582Z</Initiated>
  </Upload>
</ListMultipartUploadsResult>
```

请求示例 2

带 prefix 和 delimeter 列举已初始化的段任务

例如，用户桶 examplebucket 中 2 个段任务，对象名分别为 multipart-object001 和 part2-key02，列举段任务时，设置 prefix 为“multipart"，delimeter 设置为 object001，列举已初始化的段任务。

```
GET /?uploads&delimiter=object001&prefix=multipart HTTP/1.1
User-Agent: curl/7.29.0
Host: examplebucket.obs.region.example.com
Accept: */*
Date: WED, 01 Jul 2015 04:51:21 GMT
Authorization: OBS UDSIAMSTUBTEST000008:XdmZgYQ+ZVy1rjxJ9/KpKq+wrU0=
```

响应示例 2

```
HTTP/1.1 200 OK
Server: OBS
x-obs-request-id: 5DEB00000164A27A1610B8250790D703
x-obs-id-2: 32AAAQAAEAABAAAQAAEAABAAAQAAEAABCSq3ls2ZtLDD6pQLcJq1yGITXgspSvBR
Content-Type: application/xml
Date: WED, 01 Jul 2015 04:51:21 GMT
Content-Length: 681
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ListMultipartUploadsResult xmlns="http://obs.example.com/doc/2015-06-30/">
  <Bucket>newbucket0001</Bucket>
  <KeyMarker></KeyMarker>
  <UploadIdMarker>
  </UploadIdMarker>
  <Delimiter>object</Delimiter>
  <Prefix>multipart</Prefix>
  <MaxUploads>1000</MaxUploads>
```

```
  <IsTruncated>false</IsTruncated>
  <CommonPrefixes>
    <Prefix>multipart-object001</Prefix>
  </CommonPrefixes>
</ListMultipartUploadsResult>
```

5.4.2 初始化上传段任务

功能介绍

使用多段上传特性时，用户必须首先调用创建多段上传任务接口创建任务，系统会给用户返回一个全局唯一的多段上传任务号，作为任务标识。后续用户可以根据这个标识发起相关的请求，如：上传段、合并段、列举段等。创建多段上传任务不影响已有的同名对象；同一个对象可以同时存在多个多段上传任务；每个多段上传任务在初始化时可以附加消息头信息，包括 acl、用户自定义元数据和通用的 HTTP 消息头 contentType、contentEncoding 等，这些附加的消息头信息将先记录在多段上传任务元数据中。

请求消息样式

```
POST /ObjectName?uploads  HTTP/1.1
Host: bucketname.obs.region.example.com
Date: date
Authorization: authorization
```

请求消息参数

该请求需要在消息中指定参数，表明这是多段上传，参数意义如表 5-54 所示。

表5-54 请求消息参数

参数名称	描述	是否必选
uploads	表明这是多段上传任务。 类型：字符串。	是

请求消息头

该请求可以使用附加的消息头，具体如表 5-55 所示。

表5-55 请求消息头

消息头名称	描述	是否必选
x-obs-acl	初始化多段上传任务时，可以加上此消息头设置对象的权限控制策略，使用的策略为预定义的常用策略，包括：private；public-read；public-read-write。 类型：字符串	否

消息头名称	描述	是否必选
	说明：字符串形式的预定义策略。 实例 x-obs-acl: public-read-write。	
x-obs-grant-read	初始化多段上传任务时，使用此头域授权 domain 下所有用户有读对象和获取对象元数据的权限 类型：字符串。 实例 x-obs-grant-read: ID=domainID。如果授权给多个租户，需要通过','分割。	否
x-obs-grant-read-acp	初始化多段上传任务时，使用此头域授权 domain 下所有用户有获取对象 ACL 的权限。 类型：字符串。 实例 x-obs-grant-read-acp: ID=domainID。如果授权给多个租户，需要通过','分割。	否
x-obs-grant-write-acp	初始化多段上传任务时，使用此头域授权 domain 下所有用户有写对象 ACL 的权限。 类型：字符串。 实例 x-obs-grant-write-acp: ID=domainID。如果授权给多个租户，需要通过','分割。	否
x-obs-grant-full-control	初始化多段上传任务时，使用此头域授权 domain 下所有用户有读对象、获取对象元数据、获取对象 ACL、写对象 ACL 的权限。 类型：字符串。 实例 x-obs-grant-full-control: ID=domainID。如果授权给多个租户，需要通过','分割。	否
x-obs-website-redirect-location	当桶设置了 Website 配置，可以将获取这个对象的请求重定向到桶内另一个对象或一个外部的 URL，OBS 将这个值从头域中取出，保存在对象的元数据中。 类型：字符串 默认值：无 约束：必须以“/”、“http://”或“https://”开头，长度不超过 2K。	否
x-obs-expires	表示对象的过期时间，单位是天。过期之后对象会被自动删除。（从对象最后修改时间开始计算） 类型：整型。 示例：x-obs-expires:3	否

其他公共消息头请参考表 3-3。

请求消息元素

该请求消息中不使用消息元素。

响应消息样式

```
HTTP/1.1 status_code
Date: date
Content-Length: length
Connection: status

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<InitiateMultipartUploadResult xmlns="http://obs.region.example.com/doc/2015-06-30/">
   <Bucket>BucketName</Bucket>
   <Key>ObjectName</Key>
   <UploadId>uploadID</UploadId>
</InitiateMultipartUploadResult>
```

响应消息头

该请求的响应消息使用公共消息头，具体请参考表 3-20。

响应消息元素

该请求响应消息中通过返回消息元素，返回本次多段上传任务的多段上传任务号、桶名、对象名，供后续上传段、合并段使用，元素的具体意义如表 5-56 所示。

表5-56 响应消息元素

元素名称	描述
InitiateMultipartUploadResult	描述多段上传任务的容器。 类型：XML。
Bucket	多段上传对象所在桶的桶名。 类型：字符串。
Key	多段上传对象的 key。 类型：字符串。
UploadId	多段上传 id，后面进行多段上传时，利用这个 id 指定多段上传任务。 类型：字符串。

错误响应消息

1、如果 AccessKey 或签名无效，OBS 返回 403 Forbidden，错误码为 AccessDenied。

2、如果桶不存在，OBS 返回 404 Not Found，错误码为 NoSuchBucket。

3、检查用户是否具有指定桶的写权限，如果没有权限，OBS 返回 403 Forbidden，错误码为 AccessDenied。

其他错误已包含在表 6-3 中。

请求示例 1

初始化段

```
POST /objectkey?uploads  HTTP/1.1
Host: examplebucket.obs.region.example.com
Date: WED, 01 Jul 2015 05:14:52 GMT
Authorization: OBS AKIAIOSFODNN7EXAMPLE:VGhpcyBtZXNzYWdlIHNpZ25lZGGieSRlbHZpbmc=
```

响应示例 1

```
HTTP/1.1 200 OK
Server: OBS
x-obs-id-2: Weag1LuByRx9e6j5Onimru9pO4ZVKnJ2Qz7/C1NPcfTWAtRPfTaOFg==
x-obs-request-id: 996c76696e6727732072657175657374
Date: WED, 01 Jul 2015 05:14:52 GMT
Content-Length: 303

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<InitiateMultipartUploadResult xmlns="http://obs.region.example.com/doc/2015-06-30/">
  <Bucket>bucketname</Bucket>
  <Key>objectkey</Key>
  <UploadId>DCD2FC98B4F70000013DF578ACA318E7</UploadId>
</InitiateMultipartUploadResult>
```

请求示例 2

初始化段的同时携带 ACL

```
POST /objectkey?uploads  HTTP/1.1
Host: examplebucket.obs.region.example.com
Date: WED, 01 Jul 2015 05:15:43 GMT
x-obs-grant-write-
acp:ID=52f24s3593as5730ea4f722483579ai7,ID=a93fcas852f24s3596ea8366794f7224
Authorization: OBS AKIAIOSFODNN7EXAMPLE:VGhpcyBtZXNzYWdlIHNpZ25lZGGieSRlbHZpbmc=
```

响应示例 2

```
HTTP/1.1 200 OK
Server: OBS
x-obs-id-2: 32AAAQAAEAABAAAQAAEAABAAAQAAEAABCTnv+dtB51p+IVhAvWN7s5rSKhcWqDFs
x-obs-request-id: BB78000001648457112DF37FDFADD7AD
Date: WED, 01 Jul 2015 05:15:43 GMT
Content-Length: 303

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<InitiateMultipartUploadResult xmlns="http://obs.region.example.com/doc/2015-06-30/">
  <Bucket>bucketname</Bucket>
```

```
  <Key>objectkey</Key>
  <UploadId>000001648453845DBB78F2340DD460D8</UploadId>
</InitiateMultipartUploadResult>
```

5.4.3 上传段

功能介绍

多段上传任务创建后，用户可以通过指定多段上传任务号，通过上传段接口为特定的任务上传段，从客户端上传新数据。同一个对象的同一个多段上传任务在上传段时，上传的顺序对后续的合并操作没有影响，也即支持多个段并发上传。

段大小范围是[100KB，5GB]，但在进行合并段操作时，最后一个段的大小范围为[0,5GB]。上传的段的编号也有范围限制，其范围是[1,10000]。

须知

段任务中的 partNumber 是唯一的，重复上传相同 partNumber 的段，后一次上传会覆盖前一次上传内容。多并发上传同一对象的同一 partNumber 时，服务端遵循 Last Write Win 策略，但“Last Write”的时间定义为段元数据创建时间。为了保证数据准确性，客户端需要加锁保证同一对象的同一个段上传的并发性。同一对象的不同段并发上传不需要加锁。

请求消息样式

```
PUT /ObjectName?partNumber=partNum&uploadId=uploadID  HTTP/1.1
Host: bucketname.obs.region.example.com
Date: date
Content-Length: length
Authorization: authorization
Content-MD5: md5
<object Content>
```

请求消息参数

在上传段的时候需要通过在消息参数中指定多段上传任务号和段号来上传指定段，参数的具体意义如表 5-57 所示。

表5-57 请求消息参数

参数名称	描述	是否必选
partNumber	上传段的段号。取值为从 1 到 10000 的整数 类型：整型。	是
uploadId	多段上传任务 Id。 类型：字符串。	是

请求消息头

该请求使用公共消息头，具体请参考表 3-3。

请求消息元素

该请求消息中不使用消息元素。

响应消息样式

```
HTTP/1.1 status code
Date: date
ETag: etag
Content-Length: length
```

响应消息头

该请求的响应消息使用公共消息头，具体请参考表 3-20。

响应消息元素

该请求的响应消息中不带消息元素。

错误响应消息说明

1. 如果段序号超过范围[1,10000]，则返回错误 400 Bad Request。
2. 如果段大小超过 5G，则返回错误 400 Bad Request。
3. 如果 AccessKey 或签名无效，OBS 返回 403 Forbidden， 错误码为 AccessDenied。
4. 查询桶是否存在，如果桶不存在，OBS 返回 404 Not Found，错误码为 NoSuchBucket。
5. 检查桶的 ACL，判断用户 DomainId 是否具有指定桶的写权限，如果没有权限，则 OBS 返回 403 Forbidden，错误码为 AccessDenied。
6. 检查多段上传任务是否存在，如果不存在，OBS 返回 404 Not Found，错误码为 NoSuchUpload。
7. 检查请求用户是否是多段上传任务的发起者（Initiator），如果不是，OBS 返回 403 Forbidden， 错误码为 AccessDenied。

其他错误已包含在表 6-3 中。

请求示例

```
PUT /object02?partNumber=1&uploadId=00000163D40171ED8DF4050919BD02B8 HTTP/1.1
User-Agent: curl/7.29.0
Host: examplebucket.obs.region.example.com
Accept: */*
Date: WED, 01 Jul 2015 05:15:55 GMT
Authorization: OBS H4IPJX0TQTHTHEBQQCEC:ZB0hFwaHubi1aKHv7dSZjJts40g=
Content-Length: 102015348

[102015348 Byte part content]
```

响应示例

```
HTTP/1.1 200 OK
Server: OBS
x-obs-request-id: 8DF400000163D40956A703289CA066F1
ETag: "b026324c6904b2a9cb4b88d6d61c81d1"
x-obs-id-2: 32AAAQAAEAABAAAQAAEAABAAAQAAEAABCUQu/EOEVSMa04GXVwy0z9WI+BsDKvfh
Date: WED, 01 Jul 2015 05:15:55 GMT
Content-Length: 0
```

5.4.4 拷贝段

功能介绍

多段上传任务创建后，用户可以通过指定多段上传任务号，为特定的任务上传段。添加段的方式还包括调用段拷贝接口。允许客户将已上传对象的一部分或全部拷贝为段。

须知

拷贝段的结果不能仅根据 HTTP 返回头域中的 status_code 来判断请求是否成功，头域中 status_code 返回 200 时表示服务端已经收到请求，且开始处理拷贝段请求。拷贝是否成功会在响应消息的 body 中，只有 body 体中有 ETag 标签才表示成功，否则表示拷贝失败。

将源对象 object 拷贝为一个段 part1，如果在拷贝操作之前 part1 已经存在，拷贝操作执行之后老段数据 part1 会被新拷贝的段数据覆盖。拷贝成功后，只能列举到最新的段 part1，老段数据将会被删除。因此在使用拷贝段接口时请确保目标段不存在或者已无价值，避免因拷贝段导致数据误删除。拷贝过程中源对象 object 无任何变化。

请求消息样式

```
PUT /ObjectName?partNumber=partNum&uploadId=UploadID HTTP/1.1
Host: bucketname.obs.region.example.com
Date: date
x-obs-copy-source: sourceobject
x-obs-copy-source-range:bytes=start-end
Authorization: authorization
Content-Length: length
```

请求消息参数

拷贝段需要在参数中指定目标段的段号和多段上传任务号，参数的具体意义如表 5-58 所示。

表5-58 请求消息参数

参数名称	描述	是否必选
partNumber	上传段的段号。	是

参数名称	描述	是否必选
	类型：整型。	
uploadId	多段上传任务 Id。 类型：字符串。	是

请求消息头

该请求的除了使用公共消息头外，还使用了两个扩展的消息头。公共消息头如表 3-3 所示。

表5-59 请求消息头

消息头名称	描述	是否必选
x-obs-copy-source	拷贝的源对象。 类型：字符串。	是
x-obs-copy-source-range	源对象中待拷贝的段的字节范围（start - end），start 为段起始字节，end 为段结束字节。 类型：整型。	否

请求消息元素

该请求消息中不使用消息元素。

响应消息样式

```
HTTP/1.1 status code
Date: date

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<CopyPartResult xmlns="http://obs.region.example.com/doc/2015-06-30/">
   <LastModified>modifiedDate</LastModified>
   <ETag>etag</ETag>
</CopyPartResult>
```

响应消息头

该请求的响应消息使用公共消息头，具体请参考表 3-20。

响应消息元素

该请求的响应消息使用消息元素来返回段拷贝的结果，元素的意义如表 5-60 所示。

表5-60 响应消息元素

元素名称	描述
LastModified	对象上次修改时间。 类型：字符串。
ETag	目标段的 ETag 值，是段内容的唯一标识，用于段合并时校验数据一致性。 类型：字符串。

错误响应消息

1. 如果 AccessKey 或签名无效，OBS 返回 403 Forbidden，错误码为 AccessDenied。
2. 查询源桶或目的桶是否存在，如果不存在，OBS 返回 404 Not Found，错误码为 NoSuchBucket。
3. 如果源对象不存在，OBS 返回 404 Not Found，错误码为 NoSuchKey。
4. 如果用户对指定对象没有读权限，OBS 返回 403 Forbidden，错误码为 AccessDenied。
5. 如果用户对目的桶没有写权限，OBS 返回 403 Forbidden，错误码为 AccessDenied。
6. 查询指定的任务不存在，OBS 返回 404 Not Found，错误码为 NoSuchUpload。
7. 如果用户不是多段上传任务的发起者，OBS 返回 403 Forbidden，错误码为 AccessDenied。
8. 当拷贝的单段超过 5G 时，OBS 返回 400 Bad Request。
9. 如果段序号超过范围[1,10000]，OBS 返回错误 400 Bad Request。

其他错误已包含在表 6-3 中。

请求示例

```
PUT /tobject02?partNumber=2&uploadId=00000163D40171ED8DF4050919BD02B8 HTTP/1.1
User-Agent: curl/7.29.0
Host: examplebucket.obs.region.example.com
Accept: */*
Date: WED, 01 Jul 2015 05:16:32 GMT
Authorization: OBS H4IPJX0TQTHTHEBQQCEC:dSnpnNpawDSsLg/xXxaqFzrAmMw=
x-obs-copy-source: /destbucket/object01
```

响应示例

```
HTTP/1.1 200 OK
Server: OBS
x-obs-request-id: 8DF400000163D40ABBD20405D30B0542
x-obs-id-2: 32AAAQAAEAABAAAQAAEAABAAAQAAEAABCTIJpD2efLy5o8sTTComwBb2He0j11Ne
Content-Type: application/xml
Date: WED, 01 Jul 2015 05:16:32 GMT
Transfer-Encoding: chunked
```

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<CopyPartResult xmlns="http://obs.example.com/doc/2015-06-30/">
  <LastModified>2015-07-01T05:16:32.344Z</LastModified>
  <ETag>"3b46eaf02d3b6b1206078bb86a7b7013"</ETag>
</CopyPartResult>
```

5.4.5 列举已上传的段

功能介绍

用户可以通过本接口查询一个任务所属的所有段信息。此接口列举的各个段大小和分段上传的各个段大小一致。

请求消息样式

```
GET /ObjectName?uploadId=uploadid&max-parts=max&part-number-marker=marker HTTP/1.1
Host: bucketname.obs.region.example.com
Date: date
Authorization: auth
```

请求消息参数

该请求通过请求消息参数指定多段上传任务以及列出的段数量，参数的具体含义如表 5-61 所示。

表5-61 请求消息参数

参数名称	描述	是否必选
uploadId	多段上传任务的 id。 类型：字符串。 默认值：无。	是
max-parts	规定在列举已上传段响应中的最大 Part 数目。 类型：字符串。 默认值：1,000。	否
part-number-marker	指定 List 的起始位置，只有 Part Number 数目大于该参数的 Part 会被列出。 类型：字符串。 默认值：无。	否

请求消息头

该请求使用公共消息头，具体请参考表 3-3。

请求消息元素

该请求消息中不使用消息元素。

响应消息样式

```
HTTP/1.1 status_code
Date: date
Content-Length: length

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ListPartsResult xmlns="http://obs.region.example.com/doc/2015-06-30/">
   <Bucket>BucketName</Bucket>
   <Key>object</Key>
   <UploadId>uploadid</UploadId>
   <Initiator>
      <ID>id</ID>
   </Initiator>
   <Owner>
      <ID>ownerid</ID>
   </Owner>
   <PartNumberMarker>partNmebermarker</PartNumberMarker>
   <NextPartNumberMarker>nextPartnumberMarker</NextPartNumberMarker>
   <MaxParts>maxParts</MaxParts>
   <IsTruncated>true</IsTruncated>
   <Part>
      <PartNumber>partNumber</PartNumber>
      <LastModified>modifiedDate</LastModified>
      <ETag>etag</ETag>
      <Size>size</Size>
   </Part>
</ListPartsResult>
```

响应消息头

该请求的响应消息使用公共消息头，具体请参考表 3-20。

响应消息元素

该请求的响应通过消息元素返回已上传了的段信息，元素的具体含义如表 5-62 所示。

表5-62 响应消息元素

响应字段名称	描述
ListPartsResult	保存 List Part 请求结果的容器。 类型：容器。 子节点： Bucket, Key, UploadId, PartNumberMarker,NextPartNumberMarker, MaxParts, IsTruncated, Part。 父节点：无。

响应字段名称	描述
Bucket	Bucket 名称。 类型：字符串。 父节点：ListPartsResult。
Key	Object 名称。 类型：字符串。 父节点：ListPartsResult。
UploadId	Upload 任务 ID。 类型：字符串。 父节点：ListPartsResult。
Initiator	Upload 任务的创建者。 类型：容器。 子节点：ID。 父节点：ListPartsResult。
Owner	和 Initiator 相同。 类型：容器。 子节点: ID。 父节点: ListPartsResult。
ID	创建者的 DomainId。 类型：字符串。 父节点: Initiator、Owner。
PartNumberMarker	本次 List 结果的 Part Number 起始位置。 类型：整数。 父节点：ListPartsResult。
NextPartNumberMarker	如果本次没有返回全部结果，响应请求中将包含 NextPartNumberMarker 元素，用于标明接下来请求的 PartNumberMarker 值。 类型：整数。 父节点：ListPartsResult。
MaxParts	返回请求中最大的 Part 数目。 类型：整数。 父节点：ListPartsResult。
IsTruncated	标明是否本次返回的 List Part 结果列表被截断。“true”表示本次没有返回全部结果；“false”表示本次已经返回了全部结果。 类型：boolean。

响应字段名称	描述
	父节点：ListPartsResult。
Part	保存 Part 信息的容器。 类型：字符串。 子节点：PartNumber, LastModified, ETag, Size。 父节点：ListPartsResult。 （PartNumber 标示 Part 的数字。）
PartNumber	已上传 Part 的编号。 类型：整型。 父节点：ListPartsResult.Part。
LastModified	Part 上传的时间。 类型：日期。 父节点：ListPartsResult.part。
ETag	已上传段内容的 ETag，是段内容的唯一标识，用于段合并时校验数据一致性。 类型：字符串。 父节点：ListPartsResult.Part
Size	已上传 Part 大小。 类型：整数。 父节点：ListPartsResult.Part。

错误响应消息

1. 如果 AccessKey 或签名无效，OBS 返回 403 Forbidden，错误码为 AccessDenied。
2. 如果请求的桶不存在，OBS 返回 404 Forbidden，错误码为 NoSuchBucket。
3. 如果请求的多段上传任务不存在，OBS 返回 404 Not Found，错误码为 NoSuchUpload。
4. OBS 判断用户 DomainId 是否具有指定桶的读权限，如果没有权限，则 OBS 返回 403 Forbidden，错误码为 AccessDenied。

其他错误已经包含在表 6-3 中。

请求示例

```
GET /object02?uploadId=00000163D40171ED8DF4050919BD02B8 HTTP/1.1
User-Agent: curl/7.29.0
Host: examplebucket.obs.region.example.com
Accept: */*
Date: WED, 01 Jul 2015 05:20:35 GMT
Authorization: OBS H4IPJX0TQTHTHEBQQCEC:xkABdSrBPrz5yqzuZdJnK5oL/yU=
```

响应示例

```
HTTP/1.1 200 OK
Server: OBS
x-obs-request-id: 8DF400000163D40C099A04EF4DD1BDD9
x-obs-id-2: 32AAAQAAEAABAAAQAAEAABAAAQAAEAABCSK71fr+hDnzB0JBvQC1B9+S12AWxC41
Content-Type: application/xml
Date: WED, 01 Jul 2015 05:20:35 GMT
Content-Length: 888

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ListPartsResult xmlns="http://obs.example.com/doc/2015-06-30/">
  <Bucket>test333</Bucket>
  <Key>obj2</Key>
  <UploadId>00000163D40171ED8DF4050919BD02B8</UploadId>
  <Initiator>

<ID>domainID/domainiddomainiddomainiddo000008:userID/useriduseriduseriduseridus0000
08</ID>
  </Initiator>
  <Owner>
    <ID>domainiddomainiddomainiddo000008</ID>
  </Owner>
  <PartNumberMarker>0</PartNumberMarker>
  <NextPartNumberMarker>2</NextPartNumberMarker>
  <MaxParts>1000</MaxParts>
  <IsTruncated>false</IsTruncated>
  <Part>
    <PartNumber>1</PartNumber>
    <LastModified>2018-06-06T07:39:32.522Z</LastModified>
    <ETag>"b026324c6904b2a9cb4b88d6d61c81d1"</ETag>
    <Size>2058462721</Size>
  </Part>
  <Part>
    <PartNumber>2</PartNumber>
    <LastModified>2018-06-06T07:41:03.344Z</LastModified>
    <ETag>"3b46eaf02d3b6b1206078bb86a7b7013"</ETag>
    <Size>4572</Size>
  </Part>
</ListPartsResult>
```

5.4.6 合并段

功能介绍

如果用户上传完所有的段，就可以调用合并段接口，系统将在服务端将用户指定的段合并成一个完整的对象。在执行“合并段”操作以前，用户不能下载已经上传的数据。在合并段时需要将多段上传任务初始化时记录的附加消息头信息拷贝到对象元数据中，其处理过程和普通上传对象带这些消息头的处理过程相同。在并发合并段的情况下，仍然遵循 Last Write Win 策略，但“Last Write”的时间定义为段任务的初始化时间。

已经上传的段，只要没有取消对应的多段上传任务，都要占用用户的容量配额；对应的多段上传任务“合并段”操作完成后，只有指定的多段数据占用容量配额，用户上

传的其他此多段任务对应的段数据如果没有包含在“合并段”操作制定的段列表中，“合并段”完成后删除多余的段数据，且同时释放容量配额。

合并完成的多段上传数据可以通过已有的下载对象接口，下载整个多段上传对象或者指定 Range 下载整个多段上传对象的某部分数据。

合并完成的多段上传数据可以通过已有的删除对象接口，删除整个多段上传对象的所有分段数据，删除后不可恢复。

合并完成的多段上传数据不记录整个对象的 MD5 作为 Etag，在下载多段数据或 List 桶内对象看到的多段数据其 Etag 的生成方式为：*MD5（M_1M_2……M_N）-N*，其中，M_n 表示第 n 段的 MD5 值， 如 11.6 示例所示，有 3 个分段，每个分段都有对应的 MD5 值，合并段 ETag 的生成是先将 3 个分段的 MD5 合并起来再进行 MD5 计算得到一个新值，再拼接-N 作为合并段的 ETag 值，-N 表示总共有多少段，在该示例中即为-3。

多版本

如果桶的多版本状态是开启的，则合并段后得到的对象生成一个唯一的版本号，并且会在响应报头 x-obs-version-id 返回该版本号。如果桶的多版本状态是暂停的，则合并段后得到的对象版本号为 **null**。关于桶的多版本状态，参见 5.2.11 设置桶的多版本状态。

须知

如果上传了 10 个段，但合并时只选择了 9 个段进行合并，那么未被合并的段将会被系统自动删除，未被合并的段删除后不能恢复。在进行合并之前请使用列出已上传的段接口进行查询，仔细核对所有段，确保没有段被遗漏。

请求消息样式

```
POST /ObjectName?uploadId=uploadID HTTP/1.1
Host: bucketname.obs.region.example.com
Date: date
Content-Length: length
Authorization: authorization
<CompleteMultipartUpload>
   <Part>
      <PartNumber>partNum</PartNumber>
      <ETag>etag</ETag>
   </Part>
   <Part>
      <PartNumber>partNum</PartNumber>
      <ETag>etag</ETag>
   </Part>
   <Part>
      <PartNumber>partNum</PartNumber>
      <ETag>etag</ETag>
   </Part>
</CompleteMultipartUpload>
```

请求消息参数

该请求在消息参数中指定多段上传任务号来标明它要合并哪一个上传段任务，参数意义如表 5-63 所示。

表5-63 请求消息参数

参数名称	描述	是否必选
uploadId	指明多段上传任务。 类型：字符串。	是

请求消息头

该请求使用公共消息头，具体请参考表 3-3。

请求消息元素

该请求需要在消息中带消息元素，指定要合并的段列表，元素的具体意义如表 5-64 中所示

表5-64 请求消息元素

元素名称	描述	是否必选
CompleteMultipartUpload	合并的段列表。 类型：XML。	是
PartNumber	段号。 类型：整型。	是
ETag	上传段成功后返回的 ETag 值，是段内容的唯一标识。该值用于合并段时进行数据一致性校验。 类型：字符串。	是

响应消息样式

```
HTTP/1.1 status code
Date: date
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<CompleteMultipartUploadResult xmlns="http://obs.region.example.com/doc/2015-06-30/">
   <Location>http://example-Bucket.obs.region.example.com/example-Object</Location>
   <Bucket>bucketname</Bucket>
   <Key>ObjectName</Key>
   <ETag>ETag</ETag>
</CompleteMultipartUploadResult>
```

响应消息头

该请求的响应消息使用公共消息头，具体请参考表 3-20。

除公共响应消息头之外，还可能使用如表 5-65 中的消息头。

表5-65 附加响应消息头

消息头名称	描述
x-obs-version-id	合并得到的对象的版本号。 类型：字符串

响应消息元素

该请求的响应消息中通过返回消息元素来返回合并段的结果，元素的具体意义如表 5-66 所示。

表5-66 响应消息元素

元素	描述
Location	合并后得到的对象的 url。 类型：字符串。
Bucket	合并段所在的桶。 类型：字符串。
Key	合并得到对象的 key。 类型：字符串。
ETag	根据各个段的 ETag 计算得出的结果，是对象内容的唯一标识。 类型：字符串。

错误响应消息

1. 如果没有消息体，OBS 返回 400 Bad Request。
2. 如果消息体格式不正确，OBS 返回 400 Bad Request。
3. 消息体中如果段信息未按照段序号升序排列，OBS 返回 400 Bad Request，错误码为 InvalidPartOrder。
4. 如果 AccessKey 或签名无效，OBS 返回 403 Forbidden， 错误码为 AccessDenied。
5. 如果请求的桶不存在，OBS 返回 404 Not Found，错误码为 NoSuchBucket。
6. 如果请求的多段上传任务不存在，OBS 返回 404 Not Found，包含错误信息 NoSuchUpload。

7. 如果用户不是该任务的发起者（initiator），OBS 返回 403 Forbidden，错误码为 AccessDenied。
8. 在合并段时如果请求段列表中包含了不存在的段，OBS 返回 400 Bad Request，错误码为 InvalidPart。
9. 如果请求段列表中包含的段的 Etag 错误，OBS 返回 400 Bad Request，错误码为 InvalidPart。
10. 除最后一个段之外的其它段尺寸过小（小于 5MB），OBS 返回 400 Bad Request。
11. 对象在合并完成后总大小如果超过 48.8TB，OBS 返回 400 Bad Request。

其他错误已包含在表 6-3 中。

请求示例

```
POST /object02?uploadId=00000163D46218698DF407362295674C HTTP/1.1
User-Agent: curl/7.29.0
Host: examplebucket.obs.region.example.com
Accept: */*
Date: WED, 01 Jul 2015 05:23:46 GMT
Authorization: OBS H4IPJX0TQTHTHEBQQCEC:dOfK9iILcKxo58tRp3fWeDoYzKA=
Content-Length: 422

<?xml version="1.0" encoding="utf-8"?>
<CompleteMultipartUpload>
  <Part>
    <PartNumber>1</PartNumber>
    <ETag>a54357aff0632cce46d942af68356b38</ETag>
  </Part>
  <Part>
    <PartNumber>2</PartNumber>
    <ETag>0c78aef83f66abc1fa1e8477f296d394</ETag>
  </Part>
  <Part>
    <PartNumber>3</PartNumber>
    <ETag>acbd18db4cc2f85cedef654fccc4a4d8</ETag>
  </Part>
</CompleteMultipartUpload>
```

响应示例

```
HTTP/1.1 200 OK
Server: OBS
x-obs-request-id: 8DF400000163D4625BE3075019BD02B8
x-obs-id-2: 32AAAQAAEAABAAAQAAEAABAAAQAAEAABCSN8D1AfQcIvyGBZ9+Ee+jU6zv1iYdO4
Content-Type: application/xml
Date: WED, 01 Jul 2015 05:23:46 GMT
Content-Length: 326

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<CompleteMultipartUploadResult xmlns="http://obs.example.com/doc/2015-06-30/">
  <Location>/examplebucket/object02</Location>
  <Bucket>examplebucket</Bucket>
  <Key>object02</Key>
```

```
  <ETag>"03f814825e5a691489b947a2e120b2d3-3"</ETag>
</CompleteMultipartUploadResult>
```

5.4.7 取消多段上传任务

功能介绍

如果用户希望取消一个任务，可以调用取消多段上传任务接口取消任务。合并段或取消任务接口被调用后，用户不能再对任务进行上传段和列举段的操作。

请求消息样式

```
DELETE /ObjectName?uploadId=uplaodID HTTP/1.1
Host: bucketname.obs.region.example.com
Date: date
Authorization: auth
```

请求消息参数

该请求通过消息参数，指定要取消的段任务的多段上传任务号，参数的意义如表 5-67 所示。

表5-67 请求消息参数

参数名称	描述	是否必选
uploadId	指明多段上传任务。 类型：字符串。	是

请求消息头

该请求使用公共消息头，具体请参考表 3-3。

请求消息元素

该请求消息中不使用消息元素。

响应消息样式

```
HTTP/1.1 status_code
Date: date
```

响应消息头

该请求的响应消息使用公共消息头，具体请参考表 3-20。

响应消息元素

该请求的响应消息中不带消息元素。

错误响应消息

1. 如果 AccessKey 或签名无效，OBS 返回 403 Forbidden，错误码为 AccessDenied。
2. 如果请求的桶不存在，OBS 返回 404 Not Found，错误码为 NoSuchBucket。
3. 用户执行取消多段上传任务操作时判断用户是否是任务初始化者或是桶的所有者，如果不是则 OBS 则返回 403 Forbidden。
4. 操作成功，OBS 向用户返回 204 No Content。

其他错误已包含在表 6-3 中。

请求示例

```
DELETE /object02?uploadId=00000163D46218698DF407362295674C HTTP/1.1
User-Agent: curl/7.29.0
Host: examplebucket.obs.region.example.com
Accept: */*
Date: WED, 01 Jul 2015 05:28:27 GMT
Authorization: OBS H4IPJX0TQTHTHEBQQCEC:QmM2d1DBXZ/b8drqtEv1QJHPbM0=
```

响应示例

```
HTTP/1.1 204 No Content
Server: OBS
x-obs-request-id: 8DF400000163D463E02A07EC2295674C
x-obs-id-2: 32AAAQAAEAABAAAQAAEAABAAAQAAEAABCTp5YDlzn0UgqG3laRfkHLGyz7RpR9ON
Date: WED, 01 Jul 2015 05:28:27 GMT
```

6 附录

6.1 状态码

服务器向用户返回的状态码和提示信息如表 6-1 所示：

表6-1 状态码

状态码	说明
2xx	服务器成功返回用户请求的数据。
4xx	客户端发出的请求有错误，服务器没有进行新建或修改数据的操作。
5xx	服务器发生错误，用户将无法判断发出的请求是否成功。

6.2 错误码

调用接口出错后，将不会返回结果数据。调用方可根据每个接口对应的错误码来定位错误原因。 当调用出错时，HTTP 请求返回一个 3xx，4xx 或 5xx 的 HTTP 状态码。返回的消息体中是具体的错误代码及错误信息。

错误响应消息格式

当错误发生时，响应消息头中都会包含：

- Content-Type: application/xml
- 错误对应的 3xx，4xx 或 5xx 的 HTTP 状态码。

响应消息体中同样会包含对错误的描述信息。下面的错误响应消息体示例展示了所有 REST 错误响应中公共的元素。

```
<?xml version="1.0" encoding="UTF-8"?>
<Error>
<Code>NoSuchKey</Code>
<Message>The resource you requested does not exist</Message>
```

```
<Resource>/example-bucket/object</Resource>
<RequestId>001B21A61C6C0000013402C4616D5285</RequestId>
<HostId>RkRCRDJENDc5MzdGQkQ4OUY3MTI4NTQ3NDk2Mjg0M0FB
QUFBQUFBYmJiYmJiYmJD</HostId>
</Error>
```

各元素的具体含义如表 6-2 所示。

表6-2 错误响应消息元素

元素名称	描述
Error	错误响应消息体 XML 结构中描述错误信息的根节点元素 。
Code	错误响应消息体 XML 中错误响应对应的 HTTP 消息返回码，具体的错误码请参见表 6-3。
Message	错误响应消息体 XML 中具体错误更全面、详细的英文解释，具体的错误消息请参见表 6-3。
RequestId	本次错误请求的请求 ID，用于错误定位。
HostId	返回该消息的服务端 ID。
Resource	该错误相关的桶或对象资源。

说明

许多错误响应包含其他的更丰富的错误信息， 建议将所有错误信息记入日志，方便程序员在诊断程序错误时阅读和理解。

错误码说明

在向 OBS 系统发出请求后，如果遇到错误，会在响应中包含响应的错误码描述错误信息。对象存储访问服务的错误码如表 6-3 所示。

表6-3 错误码

HTTP 状态码	错误码	错误信息	处理措施
301 Moved Permanently	PermanentRedirect	尝试访问的桶必须使用指定的地址，请将以后的请求发送到这个地址。	按照返回的重定向地址发送请求。
301 Moved Permanently	WebsiteRedirect	Website 请求缺少 bucketName。	携带桶名后重试。
307 Moved Temporarily	TemporaryRedirect	临时重定向，当 DNS 更新时，请求将被重定向到桶。	会自动重定向，也可以将请求发送到重定向地址。
400 Bad	BadDigest	客户端指定的对象内容的	检查头域中携带的

HTTP 状态码	错误码	错误信息	处理措施
Request		MD5 值与系统接收到的内容 MD5 值不一致。	MD5 与消息体计算出来的 MD5 是否一致。
400 Bad Request	BadDomainName	域名不合法。	使用合法的域名。
400 Bad Request	BadRequest	请求参数不合法。	根据返回的错误消息体提示进行修改。
400 Bad Request	CustomDomainAreadyExist	配置了已存在的域。	已经配置过了，不需要再配置。
400 Bad Request	CustomDomainNotExist	删除不存在的域。	未配置或已经删除，无需删除。
400 Bad Request	EntityTooLarge	用户 POST 上传的对象大小超过了条件允许的最大大小。	修改 POST 上传的 policy 中的条件或者减少对象大小。
400 Bad Request	EntityTooSmall	用户 POST 上传的对象大小小于条件允许的最小大小。	修改 POST 上传的 policy 中的条件或者增加对象大小。
400 Bad Request	IllegalLocationConstraintException	用户未带 Location 在非默认 Region 创桶。	请求发往默认 Region 创桶或带非默认 Region 的 Location 创桶。
400 Bad Request	IncompleteBody	由于网络原因或其他问题导致请求体未接受完整。	重新上传对象。
400 Bad Request	IncorrectNumberOfFilesInPostRequest	每个 POST 请求都需要带一个上传的文件。	带上一个上传文件。
400 Bad Request	InvalidArgument	无效的参数。	根据返回的错误消息体提示进行修改。
400 Bad Request	InvalidBucket	请求访问的桶已不存在。	更换桶名。
400 Bad Request	InvalidBucketName	请求中指定的桶名无效，超长或带不允许的特殊字符。	更换桶名。
400 Bad Request	InvalidEncryptionAlgorithmError	错误的加密算法。下载 SSE-C 加密的对象，携带的加密头域错误，导致不能解密。	携带正确的加密头域下载对象。
400 Bad Request	InvalidLocationConstraint	创建桶时，指定的 Location 不合法或不存	指定正确的 Location 创桶。

HTTP 状态码	错误码	错误信息	处理措施
		在。	
400 Bad Request	InvalidPart	一个或多个指定的段无法找到。这些段可能没有上传，或者指定的 entity tag 与段的 entity tag 不一致。	按照正确的段和 entity tag 合并段。
400 Bad Request	InvalidPartOrder	段列表的顺序不是升序，段列表必须按段号升序排列。	按段号升序排列后重新合并。
400 Bad Request	InvalidPolicyDocument	表单中的内容与策略文档中指定的条件不一致。	根据返回的错误消息体提示修改构造表单的 policy 重试。
400 Bad Request	InvalidRedirectLocation	无效的重定向地址。	指定正确的地址。
400 Bad Request	InvalidRequest	无效请求。	根据返回的错误消息体提示进行修改。
400 Bad Request	InvalidRequestBody	请求体无效，需要消息体的请求没有上传消息体。	按照正确的格式上传消息体。
400 Bad Request	InvalidTargetBucketForLogging	delivery group 对目标桶无 ACL 权限。	对目标桶配置 ACL 权限后重试。
400 Bad Request	KeyTooLongError	提供的 Key 过长。	使用较短的 Key。
400 Bad Request	MalformedACLError	提供的 XML 格式错误，或者不符合我们要求的格式。	使用正确的 XML 格式重试。
400 Bad Request	MalformedError	请求中携带的 XML 格式不正确。	使用正确的 XML 格式重试。
400 Bad Request	MalformedLoggingStatus	Logging 的 XML 格式不正确。	使用正确的 XML 格式重试。
400 Bad Request	MalformedPolicy	Bucket policy 检查不通过。	根据返回的错误消息体提示结合桶 policy 的要求进行修改。
400 Bad Request	MalformedQuotaError	Quota 的 XML 格式不正确。	使用正确的 XML 格式重试。
400 Bad Request	MalformedXML	当用户发送了一个配置项的错误格式的 XML 会出现这样的错误。	使用正确的 XML 格式重试。

HTTP 状态码	错误码	错误信息	处理措施
400 Bad Request	MaxMessageLengthExceeded	拷贝对象，带请求消息体。	拷贝对象不带消息体重试。
400 Bad Request	MetadataTooLarge	元数据消息头超过了允许的最大元数据大小。	减少元数据消息头。
400 Bad Request	MissingRegion	请求中缺少 Region 信息，且系统无默认 Region。	请求中携带 Region 信息。
400 Bad Request	MissingRequestBodyError	当用户发送一个空的 XML 文档作为请求时会发生。	提供正确的 XML 文档。
400 Bad Request	MissingRequiredHeader	请求中缺少必要的头域。	提供必要的头域。
400 Bad Request	MissingSecurityHeader	请求缺少一个必须的头。	提供必要的头域。
400 Bad Request	TooManyBuckets	用户拥有的桶的数量达到了系统的上限，并且请求试图创建一个新桶。	删除部分桶后重试。
400 Bad Request	TooManyCustomDomains	配置了过多的用户域	删除部分用户域后重试。
400 Bad Request	TooManyWrongSignature	因高频错误请求被拒绝服务。	更换正确的 Access Key 后重试。
400 Bad Request	UnexpectedContent	该请求需要消息体而客户端没带，或该请求不需要消息体而客户端带了。	根据说明重试。
400 Bad Request	UserKeyMustBeSpecified	该操作只有特殊用户可使用。	请联系技术支持。
403 Forbidden	AccessDenied	拒绝访问，请求没有携带日期头域或者头域格式错误。	请求携带正确的日期头域。
403 Forbidden	AccessForbidden	权限不足，桶未配置 CORS 或者 CORS 规则不匹配。	修改桶的 CORS 配置，或者根据桶的 CORS 配置发送匹配的 OPTIONS 请求。
403 Forbidden	AllAccessDisabled	用户无权限执行某操作。桶名为禁用关键字。	更换桶名。
403 Forbidden	DeregisterUserId	用户已经注销。	充值或重新开户。

HTTP 状态码	错误码	错误信息	处理措施
403 Forbidden	InArrearOrInsufficientBalance	用户欠费或余额不足而没有权限进行某种操作。	充值。
403 Forbidden	InsufficientStorageSpace	存储空间不足。	超过配额限制，增加配额或删除部分对象。
403 Forbidden	InvalidAccessKeyId	系统记录中不存在客户提供的 Access Key Id。	携带正确的 Access Key Id。
403 Forbidden	NotSignedUp	你的帐户还没有在系统中注册，必须先在系统中注册了才能使用该帐户。	先注册 OBS 服务。
403 Forbidden	RequestTimeTooSkewed	请求的时间与服务器的时间相差太大。	检查客户端时间是否与当前时间相差太大。
403 Forbidden	SignatureDoesNotMatch	请求中带的签名与系统计算得到的签名不一致。	检查你的 Secret Access Key 和签名计算方法。
403 Forbidden	Unauthorized	用户未实名认证。	请实名认证后重试。
404 Not Found	NoSuchBucket	指定的桶不存在。	先创桶再操作。
404 Not Found	NoSuchBucketPolicy	桶 policy 不存在。	先配置桶 policy。
404 Not Found	NoSuchCORSConfiguration	CORS 配置不存在。	先配置 CORS。
404 Not Found	NoSuchCustomDomain	请求的用户域不存在。	先设置用户域。
404 Not Found	NoSuchKey	指定的 Key 不存在。	先上传对象。
404 Not Found	NoSuchLifecycleConfiguration	请求的 LifeCycle 不存在。	先配置 LifeCycle。
404 Not Found	NoSuchUpload	指定的多段上传不存在。Upload ID 不存在，或者多段上传已经终止或完成。	使用存在的段或重新初始化段。
404 Not Found	NoSuchVersion	请求中指定的 version ID 与现存的所有版本都不匹配。	使用正确的 version ID。

HTTP 状态码	错误码	错误信息	处理措施
404 Not Found	NoSuchWebsiteConfiguration	请求的 Website 不存在。	先配置 Website。
405 Method Not Allowed	MethodNotAllowed	指定的方法不允许操作在请求的资源上。	方法不允许。
408 Request Timeout	RequestTimeout	用户与 Server 之间的 socket 连接在超时时间内没有进行读写操作。	检查网络后重试，或联系技术支持。
409 Conflict	BucketAlreadyExists	请求的桶名已经存在。桶的命名空间是系统中所有用户共用的，选择一个不同的桶名再重试一次。	更换桶名。
409 Conflict	BucketAlreadyOwnedByYou	发起该请求的用户已经创建过了这个名字的桶，并拥有这个桶。	不需要再创桶了。
409 Conflict	BucketNotEmpty	用户尝试删除的桶不为空。	先删除桶中对象，然后再删桶。
409 Conflict	InvalidBucketState	无效的桶状态，配置跨 Region 复制后不允许关闭桶多版本。	不关闭桶的多版本或取消跨 Region 复制。
409 Conflict	OperationAborted	另外一个冲突的操作当前正作用在这个资源上，请重试。	等待一段时间后重试。
409 Conflict	ServiceNotSupported	请求的方法服务端不支持。	服务端不支持，请联系技术支持。
411 Length Required	MissingContentLength	必须要提供 HTTP 消息头中的 Content-Length 字段。	提供 Content-Length 消息头。
412 Precondition Failed	PreconditionFailed	用户指定的先决条件中至少有一项没有包含。	根据返回消息体中的 Condition 提示进行修改。
416 Client Requested Range Not Satisfiable	InvalidRange	请求的 range 不可获得。	携带正确的 range 重试。
500 Internal Server Error	InternalError	系统遇到内部错误，请重试。	请联系技术支持。
501 Not Implemented	ServiceNotImplemented	请求的方法服务端没有实现。	当前不支持，请联系技术支持。

HTTP 状态码	错误码	错误信息	处理措施
503 Service Unavailable	ServiceUnavailable	服务器过载或者内部错误异常。	等待一段时间后重试，或联系技术支持。
503 Service Unavailable	SlowDown	请降低请求频率。	请降低请求频率。

6.3 获取访问密钥（AK/SK）

在调用接口的时候，需要使用 AK/SK 进行签名验证。AK/SK 获取步骤如下：

步骤 1 登录控制台。

步骤 2 鼠标指向界面右上角的登录用户名，在下拉列表中单击“我的凭证”。

步骤 3 单击“管理访问密钥”。

步骤 4 单击“新增访问密钥”，进入“新增访问密钥”页面。

步骤 5 输入当前用户的登录密码。

步骤 6 通过邮箱进行验证，输入对应的验证码。

步骤 7 单击“确定”，下载访问密钥。

说明

为防止访问密钥泄露，建议您将其保存到安全的位置。

----结束

6.4 获取账号 ID 和用户 ID

在调用接口的时候，部分请求中需要指定账号 ID（DomainID）和用户 ID（UserID），所以需要先在控制台上获取。获取步骤如下：

步骤 1 登录控制台。

步骤 2 单击用户名，在下拉列表中单击“我的凭证”。

在“我的凭证”页面查看和用户 ID。

----结束

6.5 并发一致性说明

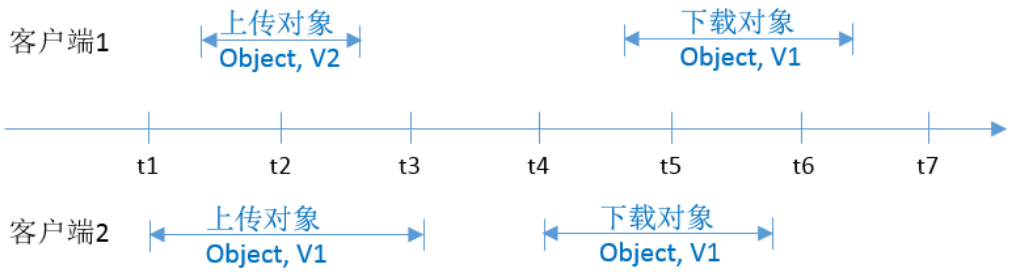
当客户端发起的写/删除请求返回成功之后，客户端可以获取到最新数据。当写操作客户端等待超时、服务端返回 500 或者 503 的 HTTP 响应错误码时，之后的读取操作有可能成功读取到数据，也有可能读不到数据。建议客户端在出现上述错误时，查询数据是否已经上传成功，如果不成功则重新上传。

针对同一个对象或桶的操作，比如多个客户端对同一个对象并行上传、查询和删除时，具体操作结果依赖于操作到达系统的时间和系统内部处理的时延，可能返回不一致的结果。比如，当多个客户端并行上传同一个对象时，系统最后收到的上传请求会覆盖前一个上传的对象。如果需要避免同一个对象被并行访问，需要在上层应用中增加对象的锁机制。

并发操作举例

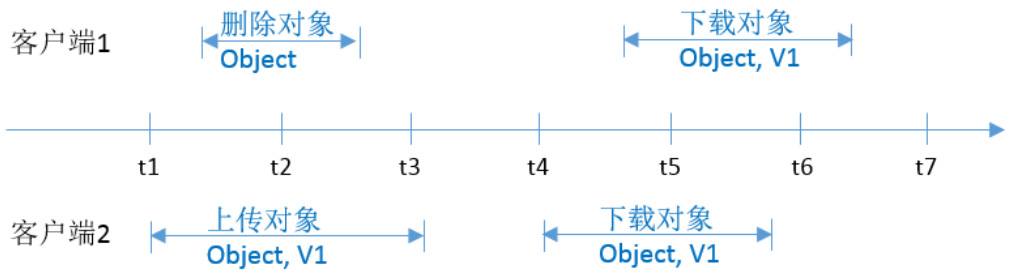
1. 当客户端 2 正在上传一个对象 v1 时，客户端 1 同时上传一个同名的对象 v2 成功后，不管是客户端 1 还是客户端 2 都能够读取最新的对象数据 v1，如图 6-1 所示。

图6-1 并发成功上传同一个对象



2. 当客户端 2 上传一个对象 v1 的时候，如果在对象数据上传且还没有写入对象元数据的过程中，客户端 1 删除同名的对象成功后，客户端 2 的上传对象操作仍然返回成功，并且不论客户端 1 还是客户端 2 都能读取到对象数据 v1，如图 6-2 所示。

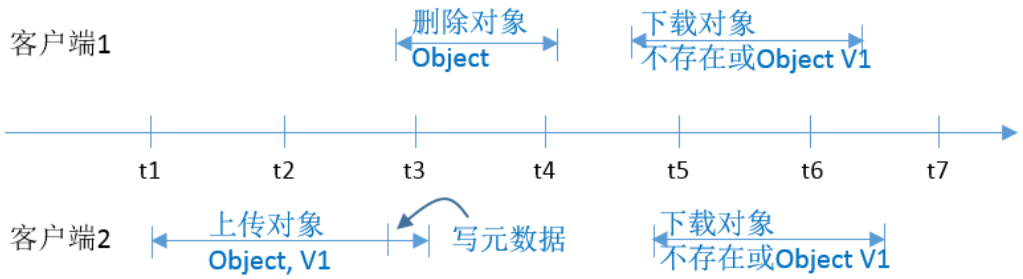
图6-2 并发上传和删除同一个对象（1）



3. 当客户端 2 上传一个对象 v1 的时候，如果在对象数据上传完成，系统写入对象元数据的短暂过程中，客户端 1 发起删除同名的对象成功后，客户端 2 的上传对象操作仍

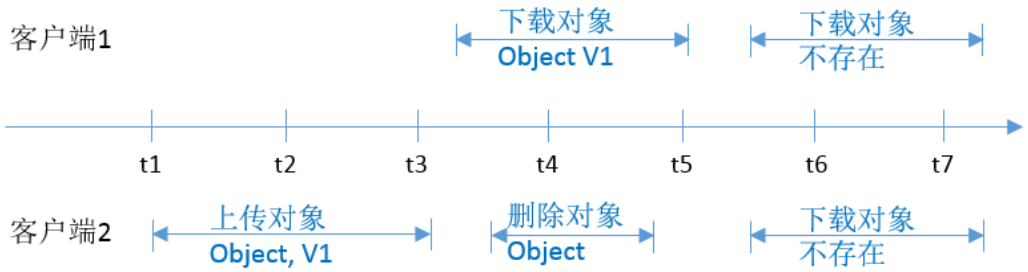
然返回成功，但是客户端 1 和客户端 2 下载对象 Object1 时，有可能读到对象数据 v1，也有可能返回对象不存在，如图 6-3 所示。

图6-3 并发上传和删除同一个对象（2）



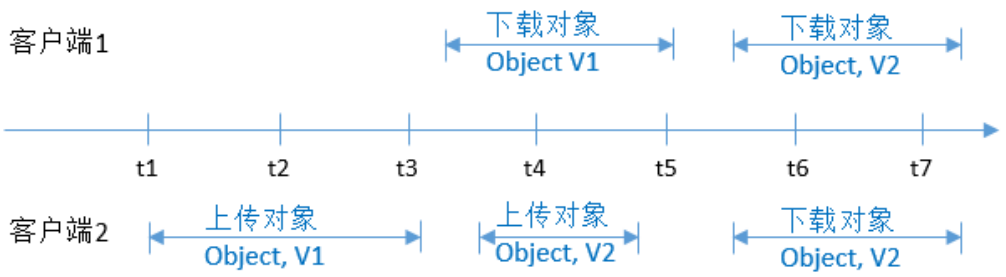
4. 当客户端 1 下载一个对象的过程中，客户端 2 发起删除同名对象操作，此时客户端 1 可能下载到完整的对象数据，也有可能只能下载到对象的一部分数据。当客户端 2 的删除操作返回成功后，再发起下载对象请求，则返回对象不存在，如图 6-4 所示。

图6-4 并发下载和删除同一个对象



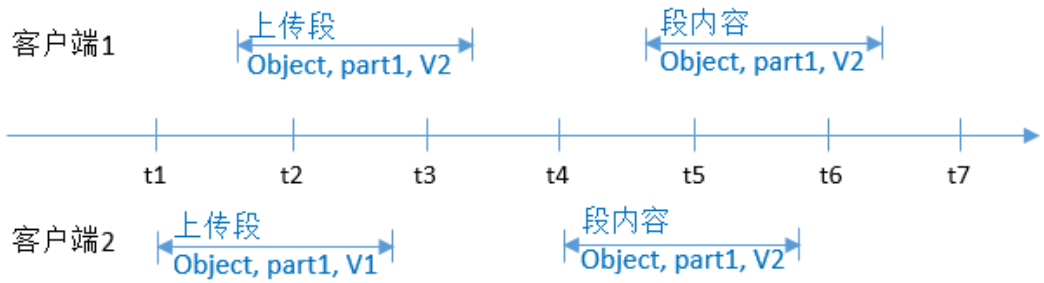
5. 当客户端 1 下载一个对象的过程中，客户端 2 发起更新同名对象操作，此时客户端 1 可能下载到完整的对象数据，也有可能只能下载到对象的一部分数据。当客户端 2 的更新操作返回成功后，再发起下载对象请求，则返回最新的对象数据，如图 6-5 所示。

图6-5 并发下载和更新同一个对象



6. 当客户端 2 正在上传一个对象的段 v1 时，客户端 1 同时上传同一个对象的相同段号的段 v2 成功后，不管是客户端 1 还是客户端 2 列举段时都能够列举 etag 为 v2 的段信息，如图 6-6 所示。

图6-6 并发上传同名对象的同名段



A 修订记录

发布日期	修订记录
2020-11-18	第一次正式发布。
2021-9-3	修改了公司名称
2022-10-27	修改了公司新 Logo