天翼云
State Cloud

对象存储（经典版）I 型

(Object-Oriented Storage,OOS)

JavaScrip SDK 开发者指南 V6

天翼云科技有限公司

目录

1 前言

对象存储（经典版）Ⅰ型（Object-Oriented Storage，OOS）为客户提供一种海量、弹性、廉价、高可用的存储服务。客户只需花极少的钱就可以获得一个几乎无限的存储空间，可以随时根据需要调整对资源的占用，并只需为真正使用的资源付费。

OOS 提供了基于 Web 门户和基于 HTTP REST 接口两种访问方式，用户可以在任何地方通过互联网对数据进行管理和访问。OOS 提供的 REST 接口与 Amazon S3 兼容，因此基于 OOS 的业务可以非常轻松的与 Amazon S3 对接。

2 开始使用

2.1 要求

已注册天翼云账户，开通 OOS 服务。

创建 AccessKeyId 和 AccessSecretKey。AccessKeyId 和 AccessSecretKey 是您访问 OOS 的密钥，OOS 会通过它来验证您的资源请求，请妥善保管

2.2 使用方式

```
<!–引入本地资源-->
<script src="./oos-sdk-x.x.x.min.js"></script>
```

使用 new OOS 创建 oos 文件

OOS JS-SDK 目前只支持异步请求方式，通过 callback 方式处理，对于 err 处理，非 err 显示处理结果。下图示例初始化到返回结果流程：

```
var client = new OOS.S3({...});
var params;
client.listObjects(params, function (err, data) {
    if (err) console.log(err, err.stack); // an error occurred
    else     console.log(data);           // successful response
});
```

2.3 CORS 配置

从浏览器中直接访问 OOS 需要开通 Bucket 的 CORS：

- 将 来源（*）设置成: *
- 将允许的方法（*）设置成: PUT, GET, POST, DELETE, HEAD
- 将允许的 Headers 设置成: *
- 将暴露的 Headers：设置成: ETag

注意：请将该条 CORS 规则设置成所有 CORS 规则的第一条.

由于浏览器的同源策略，在浏览器里调用 JS-SDK 时，部分功能无法实现，包括 Service 的操作，Bucket 的新建，AccessKey 以及 SecretKey 的操作

2.4 options 配置项

OOS options 介绍

参数	描述	是否必须
accessKeyId	String 通过天翼云控制台创建的 access key	是
secretAccessKey	String 通过天翼云控制台创建的 secret access key;	是
endPoint	String OOS 域名	是
signatureVersion	String 计算签名版本，目前支持的版本为“v2”或“v4”	是
apiVersion	String 使用的 API 版本 默认‘2006-03-01’	是
s3ForcePathStyle	Boolean 是否强把请求的 url 格式化为 s3 协议所需的风格，目前遵循 s3 协议需要填写 true；	是
sessionToken	临时授权访问 OOS API 时需要将安全令牌，此时的 accessKeyId、secretAccessKey 也应是 STS 临时授权临时生成的	否

例：

```
var client = new OOS.S3({
    accessKeyId: accessKeyId,
    secretAccessKey: secretAccessKey,
    endpoint: endPoint,
    signatureVersion: 'v2/v4',
    apiVersion: '2006-03-01',
    s3ForcePathStyle: true,
sessionToken: sessionToken
});
```

3 SDK 列表

该部分主要介绍的内容是 OOS JS-SDK 对支持的所有功能，当用户发送请求给 OOS 时，可以通过签名认证的方式请求，也可以匿名访问。

由于浏览器同源策略，需要在自服务门户把 Bucket 的 CORS 支持配置完成才可用，配置方法见 2.3 CORS 配置。

一个完整的操作包括 params，callback，基本用法：

```
var params = {
Param1: 'Value1', /* 参数*/
    ...
};
client.OPERATE(params, function(err, data) {
    if (err) console.log(err, err.stack); // 显示错误信息
    else     console.log(data);           // 成功，可在此处进行下一步操作
});
```

以下所有操作均可按照此模板，在 params 里面放入所需参数，调用 client 的方法，通过 callback 进行下一步操作。

3.1 关于 Bucket 的操作

3.1.1 GET Bucket Location

此操作用来获取 Bucket 的索引位置和数据位置，只有根用户和拥有 GET Bucket Location 权限的子用户才能执行此操作。

请求方法

```
client.getBucketLocation(params, function(err, data) {
    ...
});
```

请求参数

名称	描述	是否必须
Bucket	Bucket 名称。	是

返回元素

- **响应结果**

名称	描述
BucketConfiguration	设置Bucket索引位置和数据位置的容器。 类型：容器
MetadataLocationConstraint	设置Bucket的索引位置。 类型：容器 父节点：CreateBucketConfiguration
DataLocationConstraint	设置Bucket的数据位置。 类型：容器 父节点：CreateBucketConfiguration
Type	数据位置的类型 类型：枚举

	有效值：Local(本地) |Specified（指定位置），默认Local。 父节点：DataLocationConstraint
LocationList	指定的数据位置。 类型：容器 父节点：DataLocationConstraint
Location	索引位置或数据位置。 类型：字符串 有效值： ● 父节点为MetadataLocationConstraint，表示索引位置，有效值为：ChengDu、FuZhou、GuiYang、HangZhou、LaSa、LanZhou、QingDao、ShenYang、ShenZhen、WuHan、WuHu、WuLuMuQi、ZhengZhou、SH2、SuZhou ● 父节点为LocationList，表示数据位置，有效值为：ChengDu、GuiYang、LaSa、LanZhou、QingDao、SH2、ShenYang、ShenZhen、SuZhou、WuHan、WuHu、WuLuMuQi、ZhengZhou 父节点：MetadataLocationConstraint或LocationList

3.1.2 GET Bucket Acl

此操作用来获取 Bucket ACL 信息，只有拥有 GET Bucket ACL 权限的用户才可以执行此操作。

请求方法

```
client.getBucketAcl(params, function(err, data) {
    ...
```

```
});
```

请求参数

名称	描述	是否必须
Bucket	Bucket 名称。	是

返回元素

名称	描述
Grant	存储Permission和Grantee的容器。 类型：容器 父节点：AccessControlList 子节点：Grantee、URI
Grantee	用来存储Display和拥有ID的用户被承认的许可的容器。
ID	Bucket所属账户的ID信息。 类型：字符串 父节点：Owner
Owner	存储Bucket所属账户信息的容器。 类型：容器 父节点：AccessControlPolicy 子节点：ID、DisplayName
Permission	对一个Bucket认可的许可信息。 类型：字符串 取值： ● READ（公共读） ● FULL_CONTROL（公共读写） ● 空（私有） 父节点：Grant

3.1.3 GET Bucket

此操作用来返回 Bucket 中部分或者全部（最多 1000）的 Object 信息。用户可以在请求元素中设置选择条件来获取 Bucket 中的 Object 的子集。

示例代码

```
client.listObjects(params, function(err, data) {
    ...
});
```

请求参数

名称	描述	是否必须
Bucket	Bucket 名称。	是
Delimiter	分隔符，一个用来对关键字们进行分组的字符。所有的关键字都包含delimiter和prefix间的相同子串，prefix之后第一个遇到delimiter的字符串都会加到一个叫CommonPrefix的组中。如果没有特别定义prefix，所有的关键字都会被返回，但不会有CommonPrefix。delimiter 只支持”/”，不支持其他分隔符。 类型：字符串	否
Marker	分页标识。还有需要返回的用户时，上条响应结果中会返回该参数。查看未显示项时，请求参数中需要携带此参数。 类型：字符串 取值：与上条响应中返回的结果值相同。	否
MaxKeys	设置返回结果中最多显示的数量，如果查询结果超过最大数量，另一个变量是否翻页会标记为 true<IsTruncated>True<IsTruncated>,用来返回剩余的查询结果。	否
Prefix	前缀用来限制返回的结果必须以这个前缀开始，可以通过前缀将 Bucket 分为若干个组。	否

返回元素

名称	描述
Contents	每个 object 返回的元数据。
CommonPrefixes	当定义 delimiter 之后，返回结果中会包含 CommonPrefixes，CommonPrefix 中包含以 prefix 开头，delimiter 结束的左右字符串组合，比如 prefix 是 note/,同时 delimiter 是斜杠（/），结果中的 note/summer/ju 将返回 note/summer/,其余结果将按照 maxkey 要求返回。
Delimiter	将 prefix 和第一次出现 delimiter 的所有查询结果滚动的存入 CommonPrefix 组中。
DisplayName	Object 的所有者显示名称。
ETag	通过 MD5 的方式计算出标签，ETag 主要用来反映文件内容的信息发生了改变，并不反映元数据的变化。
ID	文件拥有者的 ID。
IsTruncated	通过是（True）或否（false）来表示返回的结果是否为所有要求的结果。
Key	文件的关键字。
LastModified	记录的文件最后一个被修改的日期和时间。
Marker	标记从哪个位置开始罗列出 Bucket 中的 object。
Name	Bucket 的名称。
Prefix	关键字以特定的前缀开始。
Size	记录文件 object 的大小。
StorageClass	存储类型。

3.1.4 DELETE Bucket

此操作用来删除 Bucket，但要求被删除 Bucket 中无 Object，即该 Bucket 中的所有 Object 都已被删除。

示例代码

```
client.deleteBucket(params, function(err, data) {
    ...
});
```

请求参数

名称	描述	是否必须
Bucket	Bucket 名称。	是

返回元素

成功没有返回值。

3.1.5 PUT Bucket Policy

在 PUT 操作的 url 中加上 Policy，可以进行添加或修改 Policy 的操作。如果 Bucket 已经存在了 Policy，此操作会替换原有 Policy。只有根用户和拥有 PUT Bucket Policy 权限的用户才能执行此操作，否则会返回 403 AccessDenied 错误。

注意：如果 Bucket 的属性为私有或者公共读，使用该接口配置允许任何用户可以向该 Bucket 写文件的策略时，请联系天翼云客服评估审核后开通。

示例代码

```
client.putBucketPolicy(params, function(err, data) {
    ...
})
```

请求参数

名称	描述	是否必须
Bucket	Bucket 名称	是
Policy	Policy 规则的字符串	是

返回元素

成功没有返回值。

3.1.6 GET Bucket Policy

此操作用来返回指定 Bucket 的 Policy。只有根用户和拥有 GET Bucket Policy 权限的子用户才能执行此操作，否则会返回 403 AccessDenied 错误。如果 Bucket 没有 Policy，返回 404，NoSuchPolicy 错误。

示例代码

```
client.getBucketPolicy(params, function(err, data) {
    ...
})
```

请求参数

名称	描述	是否必须
Bucket	Bucket 名称	是

返回元素

名称	描述
Policy	policy 的 String 格式

3.1.7 DELETE Bucket Policy

在 DELETE 操作的 url 中加上 Policy，可以删除指定 Bucket 的 Policy。只有根用户和拥有 DELETE Bucket Policy 权限的子用户才能执行此操作，否则会返回 403 AccessDenied 错误。如果 Bucket 配置了 Policy，删除成功，返回 200 0K。如果 Bucket 没有配置 Policy，返回 204 NoContent。

示例代码

```
client.deleteBucketPolicy(params, function(err, data) {
    ...
})
```

请求参数

名称	描述	是否必须
Bucket	Bucket 名称。	是

返回元素

成功没有返回值。

3.1.8 PUT Bucket WebSite

此操作用来配置网站托管属性。如果 Bucket 已经存在了 website，此操作会替换原有 website。只有根用户和拥有 PUT Bucket WebSite 权限的子用户才能执行此操作。

注意：

- 如果配置静态网站托管后，当匿名用户直接访问存储桶的域名，会将静态网站文件下载到本地。如果要实现访问静态网站时是预览网站内容，而非下载静态网站文件，静态网站域名须是存储桶绑定的已备案自定义域名，为存储桶绑定自定义域名请联系天翼云客服申请。
- OOS 自有网站托管域名不支持 HTTPS 访问，用户自定义域名支持 HTTPS 访问。如果需要支持 HTTPS 访问，请联系天翼云客服，提供域名证书，证书支持格式：crt+key 或者 PEM，请确保提供的证书在有效期内，建议证书有效期至少 1 年及以上，避免使用免费证书。
- 尽量避免目标存储桶名中带有“.”，否则通过 HTTPS 访问时可能出现客户端校验证书出错。

- 设置 Bucket 的网络配置请求消息体的上限是 10KiB。

示例代码

```
client.putBucketWebsite(params, function(err, data) {
    ...
})
```

请求参数

名称	描述	是否必须
Bucket	Bucket 名称。	是
WebsiteConfiguration	请求的容器。	是
IndexDocument	Suffix 元素的容器。	是
Suffix	在请求 website endpoint 上的路径时，Suffix 会被加在请求的后面。例如，如果 suffix 是 Index.html，而你请求的是 bucket/images/，那么返回的响应是名为 images/index.html 的 object。	是
ErrorDocument	Key 的容器。	否
Key	如果出现 4XX 错误，会返回指定的 Object。	是

参数格式

```
Params = {
    Bucket: Your_Bucket_Name,
    WebsiteConfiguration: {
        ErrorDocument:{
        Key: ‘err.html’
        },
        IndexDocument:{
        Suffix: ‘index.html’
        }
    }
```

```
}
```

返回元素

成功没有返回值。

3.1.9 GET Bucket WebSite

此操作用来获得指定 Bucket 的 website。只有根用户和拥有 GET Bucket WebSite 权限的子用户才能执行此操作，否则会返回 403 AccessDenied 错误。

示例代码

```
client.getBucketWebsite(params, function(err, data) {
    ...
})
```

请求参数

名称	描述	是否必须
Bucket	用户 Bucket 名称	是

返回元素

名称	描述
WebsiteConfiguration	请求的容器。
IndexDocument	Suffix 元素的容器。
Suffix	在请求 website endpoint 上的路径时，Suffix 会被加在请求的后面。例如，如果 suffix 是 Index.html，而你请求的是 bucket/images/，那么返回的响应是名为 images/index.html 的 object。
ErrorDocument	Key 的容器。
Key	如果出现 4XX 错误，会返回指定的 Object。

3.1.10 DELETE Bucket WebSite

此操作用来删除指定 Bucket 的 website。只有根用户和拥有 DELETE Bucket WebSite 权限的子用户才能执行此操作，否则会返回 403 AccessDenied 错误。如果 Bucket 没有设置 Website 或者 Website 删除成功，返回 200 OK。

示例代码

```
client.deleteBucketWebsite(params, function(err, data) {
    ...
})
```

请求参数

名称	描述	是否必须
Bucket	Bucket 名称。	是

返回元素

成功没有返回值。

3.1.11 List Multipart Uploads

该接口用于列出所有已经通过 Initiate Multipart Upload 请求初始化，但未完成或未终止的分片上传过程。

响应中最多返回 1000 个分片上传过程的信息，它既是响应能返回的最大分片上传过程数目，也是请求的默认值。用户也可以通过设置 max-uploads 参数来限制响应中的分片上传过程数目。如果当前的分片上传过程数超出了这个值，则响应中会包含一个值为 true 的 IsTruncated 元素。如果用户要列出多于这个值的分片上传过程信息，则需要继续调用 List Multipart Uploads 请求，并在请求中设置 *key-marker* 和 *upload-id-marker* 参数。

在响应体中，分片上传过程的信息通过 key 来排序。如果用户的应用程序中启动了多个使用同一 key 文件开头的分片上传过程，那么响应体中分片上传过程首先是通过 key 来排序，在相同 key 的分片上传内部则是按上传启动的起始时间的升序来进行排列。

示例代码

```
client.listMultiparyUploads(params, function(err, data) {
    ...
})
```

请求参数

名称	描述	是否必须
Bucket	Bucket 名称。	是
Delimiter	delimiter 是一个用来对关键字们进行分组的字符。所有的关键字都包含 delimiter 和 prefix 间的相同子串，prefix 之后第一个遇到 delimiter 的字符串都会加到一个叫 CommonPrefix 的组中。如果没有特别定义 prefix，所有的关键字都会被返回，但不会有 CommonPrefix。 类型：String	否
MaxUploads	设置返回的分片上传过程的最大数目，范围从 1 到 1000，1000 是响应体中能返回的分片上传信息的最大值。 类型：Integer 默认值：1000	否
KeyMarker	与 upload-id-marker 参数一起，该参数指定列表操作从什么位置后面开始。如果 upload-id-marker 参数没有被指定，那么只有比 key-marker 参数指定的 key 更大的 key 会被列出。如果 upload-id-marker 参数被指定，任何包含与 key-marker 指定值相等或大于 key 的分片上传过程都会被包括，假设这些分片上传过程包含了比 upload-id-marker 指定值更大的 ID。	否

	类型：String	
Prefix	该参数用于列出那些以 prefix 为前缀的正在进行的上传过程，用户可以使用多个 prefix 来把一个 Bucket 分成不同的组（可以考虑采用类似文件系统中的文件夹的作用那样，使用 prefix 来对 key 进行分组）。	否
UploadIdMarker	与 key-marker 参数一起，指定列表操作从什么位置后面开始。如果 key-marker 没有被指定，upload-id-marker 参数也会被忽略。否则，任何 key 值等于或大于 key-marker 的上传过程都会被包含在列表中，只要 ID 大于 upload-id-marker 指定的值。	否

返回元素

名称	描述
ListMultipartUploadsResult	包含整个响应的容器。 类型：容器 子节点：Bucket、KeyMarker、UploadIdMarker、NextKeyMarker、NextUploadIdMarker、Maxuploads、Delimiter、Prefix、CommonPrefixes、IsTruncated、EncodingType、Upload
Bucket	分片上传对应的存储桶名称。 类型：字符串 父节点：ListMultipartUploadsResult
KeyMarker	指定key值，在这个key当前位置或它之后开始列表操作。 类型：字符串 父节点：ListMultipartUploadsResult
UploadIdMarker	指定分片上传ID，在这个ID所在位置之后开始列表操作。 类型：字符串 父节点：ListMultipartUploadsResult

NextKeyMarker	当此次列表不能将所有正在执行的分片上传过程列举完成时，NextKeyMarker作为下一次列表请求的key-marker参数的值。 类型：字符串 父节点：ListMultipartUploadsResult
NextUploadIdMarker	当此次列表不能将所有正在执行的分片上传过程列举完成时，NextKeyMarker作为下一次列表请求的upload-id-marker参数的值。 类型：字符串 父节点：ListMultipartUploadsResult
MaxUploads	响应中包含的上传过程的最大数目。 类型：字符串 父节点：ListMultipartUploadsResult
IsTruncated	标识此次分片上传过程中的所有片段是否全部被列出，如果为true则表示没有全部列出。如果分片上传过程的片段数超过了MaxParts元素指定的最大数，则会导致一次列表请求无法将所有片段数列出。 类型：Boolean 父节点：ListMultipartUploadsResult
EncodingType	KeyMarker、NextKeyMarker、Key、Prefix、Delimiter的编码类型。 类型：字符串 父节点：ListMultipartUploadsResult
Upload	某个分片上传过程的容器，响应体中可能包含0个或多个Upload元素。 类型: 容器 父节点: ListMultipartUploadsResult 子节点: Key、UploadId、Initiator、Owner、StorageClass、Initiated

Key	分片上传过程起始位置文件的Key值。 类型: String 父节点: Upload
UploadId	分片上传过程的ID号。 类型: 整型 父节点: Upload
Initiator	指定执行此次分片上传过程的用户账户。 类型：容器 父节点：Upload 子节点：ID，DisplayName
ID	OOS账户的ID。 类型：字符串 父节点：Initiator
StorageClass	文件的存储类型。 类型：字符串 父节点：Upload
Initiated	分片上传过程启动的时间。 类型：Date 父节点：Upload
Prefix	如果请求中包含了Prefix参数，则这个字段会包含Prefix的值。返回的结果只包含那些key值以Prefix开头的文件。 类型：字符串 父节点：ListMultipartUploadsResult
Delimiter	分割符，用来对关键字们进行分组的字符。所有的关键字都包含delimiter和prefix间的相同子串，prefix之后第一个遇到delimiter的字符串都会加到一个叫CommonPrefix的组中。如果没有特别定义prefix，所有的关键字都会被返回，但不会有CommonPrefix。

	类型：字符串 父节点：ListMultipartUploadsResult
CommonPrefixes	当定义delimiter之后，返回结果中会包含CommonPrefixes，CommonPrefix中包含以prefix开头，delimiter结束的字符串组合，比如prefix是note/,同时delimiter是斜杠（/），结果中的note/summer/july/lotus.jpg将返回note/summer/，其余结果将按照max-uploads要求返回。 类型：容器 父节点：ListMultipartUploadsResult 子节点：Prefix
CommonPrefixes.Prefix	如果请求中不包含Prefix参数，那么这个元素只显示那些在delimiter字符第一次出现之前的key的子字符串，且这些key不在响应的其它位置出现。 如果请求中包含Prefix参数，那么这个元素显示在prefix之后，到第一次出现delimiter之间的子串。 类型：字符串 父节点：CommonPrefixes

3.1.12 PUT Bucket Logging

此操作用来添加/修改/删除 logging 的操作。如果 Bucket 已经存在了 logging，此操作会替换原有 logging。只有根用户和拥有 PUT Bucket Logging 权限的子用户才能执行此操作，否则会返回 403 AccessDenied 错误。

示例代码

```
client.putBucketLogging(params, function(err, data) {
    ...
})
```

名称	描述	是否必须
Bucket	Bucket 名称	是
BucketLoggingStatus	请求的容器。 类型：容器 子节点：LoggingEnabled	是
LoggingEnabled	日志信息的容器，当启动日志时，需要包含这个元素。 类型：容器。 父节点：BucketLoggingStatus 子节点：TargetBucket、TargetPrefix	否
TargetBucket	指定要保存 log 的 Bucket，OOS 会向此 Bucket 存储日志。可以设置任意一个你拥有的 Bucket 作为 TargetBucket，包括启动日志的 Bucket 本身。你也可以设置将多个 Bucket 的日志存放到一个 TargetBucket 中，在这种情况下，你需要为每个源 Bucket 设置不同的 TargetPrefix，以便不同 Bucket 的 log 可以被区分出来。 类型：字符串 父节点：LoggingEnabled	否
TargetPrefix	生成的 log 文件将以此为前缀命名。 类型：字符串 父节点：LoggingEnabled	否

参数格式

```
Params = {
    Bucket: Your_Bucket_Name,
    BucketLoggingStatus: {
        LoggingEnabled: {
            TargetBucket: Target_Bucket_Name,
            TargetPrefix: Target_Prefix
        }
}
```

```
}
```

返回元素

成功没有返回值。

3.1.13 GET Bucket Logging

此操作用来获得指定 Bucket 的 logging。只有根用户和拥有 GET Bucket Logging 权限的子用户才能执行此操作，否则会返回 403 AccessDenied 错误。

示例代码

```
client.getBucketLogging(params, function(err, data) {
    ...
})
```

请求参数

名称	描述	是否必须
Bucket	Bucket 名称。	是

返回元素

名称	描述
BucketLoggingStatus	响应的容器。
LoggingEnabled	日志信息的容器，当启动日志时，包含这个元素；否则此元素及其子元素都不显示。
TargetBucket	保存 log 的 Bucket，OOS 会向此 Bucket 存储日志。
TargetPrefix	生成的 log 文件将以此为前缀命名。

3.1.14 Head Bucket

此操作用于判断 Bucket 是否存在，而且用户是否有权限访问。如果 Bucket 存在，而且用户有权限访问时，此操作返回 200 OK。否则，返回 404 不存在，或者 403 没有权限。

示例代码

```
client.headBuck(params, function(err, data) {
    ...
})
```

请求参数

名称	描述	是否必须
Bucket	Bucket 名称	是

返回元素

成功没有返回值。

3.1.15 PUT Bucket Lifecycle

Put Bucket Lifecycle 接口用于设置 Bucket 的生命周期，如果生命周期的配置已经存在，将会被替换。用户可以通过设置生命周期，来让 OOS 删除过期的文件。

示例代码

```
client.putBucketLifecycle(params, function(err, data) {
    ...
})
```

请求参数

名称	描述	是否必须
Bucket	Bucket 名称。	是

LifecycleConfiguration	容器，最多包含 100 个规则。 类型：容器 子节点：Rule 父节点：无	是
Rule	生命周期规则的容器。 类型：object 类型 Array 父节点：LifecycleConfiguration	是
ID	规则的唯一标识，最长 255 个字符。 类型：字符串 父节点：Rule	否
Filter	配置过滤器。 类型：object 父节点：Rule	是
Prefix	指明要使用规则的文件前缀，最长 1024 个字符。 类型：字符串 父节点：Filter	是
Status	如果是 Enabled，那么规则立即生效。如果是 Disabled，那么规则不会生效。 类型：字符串 父节点：Rule	是
Expiration	描述过期动作的容器。 类型：容器 子节点：Days 父节点：Rule	是
Days	以天数来描述生命周期，值是正整数。 类型：整数 父节点：Expiration	Days 和 Date 二选一

Date	生成时间早于此时间的文件将被认为是过期文件。 日期必需服从 ISO8601 的格式，并且总是 UTC 的零点。 例如：2002-10-11T00:00:00.000Z。 类型：String 父节点：Expiration	Days 和 Date 二选一

参数格式

```
Params = {
    Bucket: Your_Bucket_Name,
    LifecycleConfiguration: {
        Rules: [{
        Expiration: {
            Days: 3650
        },
        Filter: {
            Prefix: 'documents/'
        },
        ID: 'TestOnly',
        Status: 'Enabled'
    }]
}}
```

返回元素

成功没有返回值。

3.1.16 GET Bucket Lifecycle

此接口用于返回配置的 Bucket 生命周期。

示例代码

```
client.getBucketLifecycle(params, function(err, data) {
    ...
})
```

请求参数

名称	描述	是否必须
Bucket	Bucket 名称。	是

返回元素

名称	描述
LifecycleConfiguration	容器，最多包含 100 个规则。 类型：容器 子节点：Rule 父节点：无
Rule	生命周期规则的容器。 类型：容器 父节点：LifecycleConfiguration
ID	规则的唯一标示，最长 255 个字符。 类型：字符串 父节点：Rule
Prefix	指明要使用规则的文件前缀。 类型：字符串 父节点：Rule
Status	如果是 Enabled，那么规则立即生效。如果是 Disabled，那么规则不会生效。 类型：字符串 父节点：Rule
Expiration	描述过期动作的容器。

	类型：容器 子节点：Days 父节点：Rule
Days	以天数来描述生命周期，值是正整数。 类型：整数 父节点：Expiration
Date	生成时间早于此时间的文件将被认为是过期文件。 类型：字符串 父节点：Expiration

3.1.17 DLETE Bucket Lifecycle

此操作用于删除配置的 Bucket 生命周期，OOS 将会删除指定 Bucket 的所有生命周期配置规则。用户的文件将永远不会到期，OOS 也不会再自动删除文件。只有根用户和拥有 DELETE Bucket Lifecycle 权限的子用户才能执行此操作。

示例代码

```
client.deleteBucketLifecycle(params, function(err, data) {
    ...
})
```

请求参数

名称	描述	是否必须
Bucket	用户 Bucket 名称	是

返回元素

成功没有返回值。

3.2 关于 Object 的操作

3.2.1 PUT Object

此操作用来向指定 Bucket 中添加一个文件，要求发送请求者对该 Bucket 有写权限，用户必须添加完整的文件。

示例代码

```
client.putObject(params, function(err, data) {
    ...
})
```

请求参数

名称	描述	是否必须
Bucket	Bucket 名称。	是
Body	File 内容。	是
Key	文件名。	是
CacheControl	按照请求/回应的方式用来定义缓存行为。	否
ContentDisposition	指出文件的描述性的信息。	否
ContentEncoding	指出文件所使用的编码格式。	否
ContentMD5	按照 RFC 1864，使用 base64 编码格式生成信息的 128 位 MD5 值。	否
ContentType	标准的 MIME 类型用来描述内容格式。	否
Expires	new Date，文件不再被缓存的时间。 类型：字符串	否
Metadata	任何头以这个前缀开始都会被认为是用户的元数据，当用户检索时，它将会和文件一起被存储并返	否

	回。PUT 请求头大小限制为 8KB。在 PUT 请求头中，用户定义的元数据大小限制为 2KB。	
StorageClass	数据的存储类型。 类型: String 取值： ● STANDARD：标准存储。 ● STANDARD_IA：低频访问存储。 默认值为 STANDARD。	否
DataLocation	设置数据存储的位置。 类型：字符串 取值： 格式为： type=Local,scheduleStrategy=*scheduleStrategy*或者 type=Specified,location=*location*,scheduleStrategy=*scheduleStrategy* ● type：指定数据存储位置的类型，取值为 Local 或者 Specified。local 表示就近写入，Specified 表示指定位置。如果 type 取值为 Specified，则需要指定具体的数据位置 location，location 可以填写多个，以逗号分隔，可取值为 ChengDu、GuiYang、LaSa、LanZhou、QingDao、SH2、ShenYang、ShenZhen、SuZhou、WuHan、WuHu、WuLuMuQi、ZhengZhou。 ● scheduleStrategy：调度策略，取值为： Allowed：允许 OOS 自动调度数据存储位置。	否

	NotAllowed：不允许 OOS 自动调度数据存储位置。	

参数格式

```
var params = {
    Bucket: 'STRING_VALUE', /* required */
    Key: 'STRING_VALUE', /* required */
    Body: new Buffer('...') || 'STRING_VALUE' || streamObject,
    CacheControl: 'STRING_VALUE',
    ContentDisposition: 'STRING_VALUE',
    ContentEncoding: 'STRING_VALUE',
    ContentLanguage: 'STRING_VALUE',
    ContentLength: 0,
    ContentMD5: 'STRING_VALUE',
    ContentType: 'STRING_VALUE',
    Expires: new Date,
    Metadata: {
        <MetadataKey>: 'STRING_VALUE',
        /* '<MetadataKey>': ... */
    },
StorageClass: STANDARD,
DataLocation: 'STRING_VALUE'
};
```

返回元素

成功没有返回值。

3.2.2 GET Object

此操作用来检索在 OOS 中的文件信息，执行此操作，用户必须对 Object 所在的 Bucket 有读权限。如果 Bucket 是 public read 的权限，匿名用户也可以通过非授权的方式进行读操作。

注意：此方法并不能下载，只能获取到 Object 的 Uint8Array 编码字节流，若想下载，可通过 getSignedUrl 方法 return 出可下载的 url。

示例代码

```
client.getObject(params, function(err, data) {
    ...
})
```

请求参数

名称	描述	是否必须
Bucket	Bucket 名称。	是
Key	文件名。	是

返回元素

名称	描述
Body	OOS 中的文件信息
ContentType	文件类型
LastModified	最后更改时间
ContentLength	文件大小
Metadata	文件的元数据
DataLocation	获取 Bucket 的数据位置。 若要使用此字段需要在跨域配置-暴露的 Headers 选项中添加 x-ctyun-data-location。
MetaDataLocation	获取文件的索引位置。若要使用此字段需要在跨域配置-暴露的 Headers 选项中添加 x-ctyun-metadata-location。

3.2.3 DELETE Object

此操作用来删除指定的文件，用户要对文件所在的 Bucket 拥有写权限才可以执行此操作。

示例代码

```
client.deleteObject(params, function(err, data) {
    ...
```

```
})
```

请求参数

名称	描述	是否必须
Bucket	Bucket 名称。	是
Key	文件名	是

返回元素

成功没有返回值。

3.2.4 PUT Object - Copy

此操作用来创建一个存储在 OOS 里的文件拷贝。类似于执行一个 GET，然后再执行一次 PUT。要执行拷贝请求，用户需要对源文件有读权限，对目标 Bucket 有写权限。

增加参数 CopySource，使用 PUT 操作将源文件存入指定 Bucket。CopySource 数据需要做 encodeURIComponent()处理。

示例代码

```
client.copyObject(params, function(err, data) {
    ...
})
```

请求参数

名称	描述	是否必须
Bucket	目标 Bucket 名称。	是
CopySource	复制的源 Bucket/key。	是
Key	目标文件名。	是
CacheControl	按照请求/回应的方式用来定义缓存行为。	否
ContentDisposition	指出文件的描述性的信息。	否
ContentEncoding	指出文件所使用的编码格式。	否

ContentMD5	按照 RFC 1864，使用 base64 编码格式生成信息的 128 位 MD5 值。	否
ContentType	标准的 MIME 类型用来描述内容格式。	否
Expires	new Date，文件不再被缓存的时间。 类型：字符串	否
Metadata	任何头以这个前缀开始都会被认为是用户的元数据，当用户检索时，它将会和文件一起被存储并返回。PUT 请求头大小限制为 8KB。在 PUT 请求头中，用户定义的元数据大小限制为 2KB。	否
StorageClass	数据的存储类型。 类型：字符串 取值： ● STANDARD：标准存储。 ● STANDARD_IA：低频访问存储。 默认值为 STANDARD。	否
DataLocation	设置 Bucket 的数据位置 类型：key-value 形式。 有效值： type=[Local|Specified],location=[ChengDu|ShenYang|…] scheduleStrategy=[Allowed|NotAllowed] type=local 表示就近写入本地。type= Specified 表示指定位置。location 表示指定的数据位置，可以填写多个，以逗号分隔。scheduleStrategy 表示调度策略，是否允许 OOS 自动调度数据存储位置。	否

返回元素

成功没有返回值。

3.2.5 Initial Multipart Upload

本接口初始化一个分片上传（Multipart Upload）操作，并返回一个上传 ID。此 ID 用来将此次分片上传操作中上传的所有片段合并成一个文件。用户在执行每一次子上传请求（见 Upload Part）时都必须指定该 ID。用户也可以在表示整个分片上传完成的合并分片的请求中指定该 ID。或者在用户放弃该分片上传操作时指定该 ID。

示例代码

```
client.createMultipartUpload(params, function(err, data) {
    ...
})
```

请求参数

名称	描述	是否必须
Bucket	Bucket 名称。	是
Key	文件名。	是
CacheControl	按照请求/回应的方式用来定义缓存行为。	否
ContentDisposition	指出文件的描述性的信息。	否
ContentEncoding	指出文件所使用的编码格式。	否
ContentType	标准的 MIME 类型用来描述内容格式。	否
Expires	new Date，文件不再被缓存的时间。 类型：字符串	否
Metadata	任何头以这个前缀开始都会被认为是用户的元数据，当用户检索时，它将会和文件一起被存储并返回。PUT 请求头大小限制为 8KB。在 PUT 请求头中，用户定义的元数据大小限制为 2KB。	否
StorageClass	数据的存储类型。 类型: String 取值： ● STANDARD：标准存储。	否

	● STANDARD_IA：低频访问存储。 默认值为 STANDARD。	
DataLocation	设置Bucket的数据位置。 类型：key-value形式 取值： 格式为：type=Local,scheduleStrategy=*scheduleStrategy*或者 type=Specified,location=*location*,scheduleStrategy=*scheduleStrategy* ● type：指定数据存储位置的类型，取值为 Local 或者 Specified。local 表示就近写入，Specified 表示指定位置。如果 type 取值为 Specified，则需要指定具体的数据位置 location，location 可以填写多个，以逗号分隔，可取值为 ChengDu、GuiYang、LaSa、LanZhou、QingDao、SH2、ShenYang、ShenZhen、SuZhou、WuHan、WuHu、WuLuMuQi、ZhengZhou。 ● scheduleStrategy：调度策略，取值为： Allowed：允许 OOS 自动调度数据存储位置。 NotAllowed：不允许 OOS 自动调度数据存储位置。	否

返回元素

名称	描述
InitiateMultipartUploadResult	包含所有返回元素的容器 类型：容器 子节点：Bucket，Key，UploadId 父节点：无

Bucket	分片上传对应的 Bucket 的名称 类型：字符串 父节点：InitiateMultipartUploadResult
Key	分片上传对应的文件名称 类型：字符串 父节点：InitiateMultipartUploadResult
UploadId	分片上传 ID 类型：字符串 父节点：InitiateMultipartUploadResult

3.2.6 Upload Part

该接口用于实现分片上传操作中片段的上传。

在上传任何一个分片之前，必须执行 Initial Multipart Upload 操作来初始化分片上传操作，初始化成功后，OOS 会返回一个上传 ID，这是一个唯一的标识，用户必须在调用 Upload Part 接口时加入该 ID。

分片号 PartNumber 可以唯一标识一个片段并且定义该分片在文件中的位置，范围从 1 到 10000。如果用户用之前上传过的片段的分片号来上传新的分片，之前的分片将会被覆盖。

除了最后一个分片外，所有分片的都不小于 5M，最后一个分片的大小不受限制。

为了确保数据不会由于网络传输而毁坏，需要在每个分片上传请求中指定 Content-MD5 头，OOS 通过提供的 Content-MD5 值来检查数据的完整性，如果不匹配，则会返回一个错误信息。

示例代码

```
client.uploadPart(params, function(err, data) {
    ...
})
```

请求参数

名称	描述	是否必须
Bucket	分片所属的 Bucket 名。	是
Key	分片的名称。	是
PartNumber	分片的 id。	是
UploadId	初始化分片后返回的 id。	是
Body	File 数据。	否
ContentLength	该分片的大小，以字节为单位。 类型：Integer 默认值：None	否
ContentMD5	该分片数据的 128 位采用 base64 编码的 MD5 值。这个头可以用来验证该分片数据是否与原始数据值保持一致。尽管这个值是可选的，我们仍然推荐使用 Content-MD5 机制来执行端到端的一致性校验。 类型：String 默认值：None	否
Expect	如果用户的应用中设置该头为 100-continue，则应用在接收到请求回应之前不会发送请求实体。如果基于头的消息被拒绝，消息的实体也不会被发送。 类型：String 默认值：None 可选值：100-continue	否

返回元素

名称	描述
ETag	上传文件返回的实体标签。

3.2.7 Complete Multipart Upload

该接口通过合并之前的上传片段来完成一次分片上传过程。

示例代码

```
client.completeMultipartUpload(params, function(err, data) {
    ...
})
```

请求参数

名称	描述	是否必须
Bucket	分片 object 所属 Bucket 名称。	是
Key	分片的 object 名称。	是
UploadId	初始化时返回的 uploadID。	是
MultipartUpload	请求的容器。 类型：object 子节点：1 个或多个 Part 元素。	是
Parts	描述 part 的信息 父容器：MultipartUpload 类型 object 类型数组 包括：ETag: 上传每个分片时返回的 ETag（实体标签）。PartNumber: 上传的每个分片对应的编号。	是

返回元素

名称	描述
CompleteMultipartUploadResult	包含整个响应的容器。

	类型：容器 子节点：Location、Bucket、Key、ETag
Location	新创建文件的URI地址。 类型：URI 父节点：CompleteMultipartUploadResult
Bucket	分片上传对应的存储桶。 类型：字符串 父节点：CompleteMultipartUploadResult
Key	新创建的文件的Key。 类型：字符串 父节点：CompleteMultipartUploadResult
ETag	ETag用来标识新创建的文件数据。 类型：字符串 父节点：CompleteMultipartUploadResult

3.2.8 Abort Multipart Upload

该接口用于终止一次分片上传操作。分片上传操作被终止后，用户不能再通过上传 ID 上传其它片段，之前已上传完成的片段所占用的存储空间将被释放。如果此时任何片段正在上传，该上传过程可能会也可能不会成功。所以，为了释放所有片段所占用的存储空间，可能需要多次终止分片上传操作。

示例代码

```
client.abortMultipartUpload(params, function(err, data) {
    ...
})
```

请求参数

名称	描述	是否必须
Bucket	分片 object 所属 Bucket 名称。	是
Key	分片的 object 名称。	是
UploadId	初始化时返回的 uploadID。	是

返回元素

成功没有返回值。

3.2.9 List Part

操作用于列出一次分片上传过程中已经上传完成的所有片段。

该操作必须包含一个通过 Initial Multipart Upload 操作获取的上传 ID。该请求最多返回 1000 个上传片段信息，默认返回的片段数是 1000。用户可以通过指定 max-parts 参数来指定一次请求返回的片段数。如果用户的分片上传过程超过 1000 个片段，响应中的 IsTruncated 字段的值则被设置成 true，并且指定一个 NextPartNumberMarker 元素。用户可以在下一个连续的 List Part 请求中加入 *part-number-marker* 参数，并把它的值设置成上一个请求返回的 NextPartNumberMarker 值。

示例代码

```
client.listParts(params, function(err, data) {
    ...
})
```

请求参数

名称	描述	是否必须
Bucket	分片 object 所属 Bucket 名称。	是
Key	分片的 object 名称。	是
UploadId	初始化时返回的 uploadID。	是
MaxParts	设置返回的最大的分片数量。	否

PartNumberMarker	设定此值，只会返回分片号大于此值得分片。	否

返回元素

名称	描述
ListPartsResult	包含整个响应的容器。 类型：容器 子节点：Bucket、EncodingType、Key、UploadId、Initiator、Owner、StorageClass、PartNumberMarker、NextPartNumberMarker、MaxParts、IsTruncated、Part
Bucket	分片上传对应的Bucket名称。 类型：字符串 父节点：ListPartsResult
EncodingType	Key 的编码类型。 类型：字符串 父节点：ListPartsResult
Key	新创建的文件的Key。 类型：字符串 父节点：ListPartsResult。
UploadId	分片上传ID。 类型：字符串 父节点：ListPartsResult
Initiator	指定执行此次分片上传过程的用户账户。 类型：容器 父节点：ListPartsResult 子节点：ID，DisplayName
ID	OOS账户的ID。 类型：字符串 父节点：Initiator

DisplayName	OOS账户的账户名。 类型：字符串 父节点：Initiator
StorageClass	文件的存储类型。 类型：字符串 父节点：ListPartsResult
PartNumberMarker	列表起始位置的片段的分片号。 类型：整型 父节点：ListPartsResult
NextPartNumberMarker	当此次请求没有将所有片段列举完时，此元素指定列表中的最后一个片段的分片号。此分片号用于作为下一次连续列表请求的*part-number-marker*参数的值。 类型：整型 父节点：ListPartsResult
MaxParts	响应中片段的最大数目。 类型：整型 父节点：ListPartsResult
IsTruncated	标识此次分片上传过程中的所有片段是否全部被列出，如果为true则表示没有全部列出。如果分片上传过程的片段数超过了MaxParts元素指定的最大数，则会导致一次列表请求无法将所有片段数列出。 类型：Boolean 父节点：ListPartsResult
Part	与某个片段对应的容器，响应中可能包含0个或多个Part元素。 类型：容器 父节点：ListPartsResult 子节点：PartNumber、LastModified、ETag、Size
PartNumber	标识片段的分片号。

	类型：整型 父节点：Part
LastModified	片段上传完成的日期。 类型：Date 父节点：Part
ETag	片段上传完成时返回的ETag值。 类型：字符串。 父节点：Part
Size	片段的数据大小。 类型：整型 父节点：Part

3.2.10 Copy Part

可以将已经存在的 Object 作为分段上传的片段，拷贝生成一个新的片段。需要通过参数 CopySource 来定义拷贝源。如果想拷贝源 object 中的一部分，可以加参数 CopySourceRange。CopySource 数据需要做 encodeURIComponent()处理。

示例代码

```
client.uploadPartCopy(params, function(err, data) {
    ...
})
```

请求参数

名称	描述	是否必须
Bucket	分片 object 所属 Bucket 名称。	是

Key	分片的 object 名称。	是
PartNumber	分片的编号。	是
UploadId	初始化时返回的 uploadID。	是
CopySource	已经存在的 object 地址。	是

返回元素

- **响应结果**

名称	描述
CopyPartResult	包含整个响应的容器。 类型：容器 子节点：LastModified、ETag
LastModified	分片的最后修改时间。 类型：字符串 父节点：CopyPartResult
ETag	新分片的ETag。 类型：字符串 父节点：CopyPartResult

3.2.11 Delete Multiple Objects

批量删除 Object 功能支持用一个 HTTP 请求删除一个 Bucket 中的多个 Object。如果你知道你想删除的 Object 名字，此功能可以批量删除这些 Object，而不用发送多个单独的删除请求。

示例代码

```
client.deleteObjects(params, function(err, data) {
    ...
})
```

请求参数

名称	描述	是否必须
Bucket	需要删除 objects 的 Bucket 名称。	是
Delete	包含整个请求的容器。 类型：容器 父节点：无	是
Quiet	使用简明信息模式来返回响应，当使用此元素时，需要指定 true。 类型：Boolean 父节点：Delete	否
Object	包含被删除 object 的容器。 类型：容器 父节点：Delete	是
Key	被删除 object 名。 类型：String 父节点：Object	是

参数格式

```
Params = {
    Bucket: Your_Bucket_Name,
    Delete: {
        Objects: [
        {
            Key: Key_1
        },
        {
            Key: Key_2
        },
        {
            Key: Key_3
        }],
        Quiet: true
```

```
    }
}
```

返回元素

名称	描述
DeleteResult	包含整个响应的容器。 类型：容器 子节点：Deleted、Error
Deleted	成功删除的容器，包含成功删除的 object。 类型：容器 父节点：DeleteResult
Key	尝试删除的 object 名。 类型：字符串 父节点：Deleted，或 Error
Error	删除失败的容器，包含删除失败的 object 信息。 类型：容器 父节点：DeleteResult 子节点：Key、Code、Message
Code	删除失败的状态码：AccessDenied、InternalError。 类型：字符串 父节点：Error
Message	错误的描述。 类型：字符串 父节点：Error

3.2.12 生成共享链接

对于私有或只读 Bucket，可以通过生成 Object 的共享链接的方式，将 Object 分享给其他人。

示例代码

```
var url = client.getSignedUrl('getObject', params);
return url;
```

请求参数

名称	描述	是否必须
Bucket	要分享的 Bucket 名称。	是
Key	要分享的 Object 名称。	是
Expires	过期时间，单位是秒，默认 900 秒。	否

返回元素

返回一个非拥有者也可以下载的 url。

3.2.13 HEAD Object

此操作用于获取文件的元数据信息，而不返回数据本身。当只希望获取文件的属性信息时，可以使用此操作。

示例代码

```
client.headObject(params, function(err, data) {
    ...
})
```

请求参数

名称	描述	是否必须
Bucket	Bucket 名称。	是
Key	object 名称。	是
Range	下载一个文件的指定字节范围。	否

IfModifiedSince	返回一个在指定时间点后被修改的文件，无结果则返回 304 错误。	否
IfUnmodifiedSince	返回一个在指定时间点后未被修改的文件，无结果则返回 412 错误。	否
IfMatch	当文件的 ETag 与指定值一致时，返回此文件。否则返回 412 错误。	否
IfNoneMatch	当文件的 ETag 与指定值不一致时，返回此文件。否则返回 304 错误。	否

返回元素

名称	描述
ETag	Object 的 Etag 值
ContentType	文件类型
LastModified	最后更改时间
Metadata	文件的元数据
Expiration	文件的过期时间。注：若要使用此字段需要在跨域配置-暴露的 Headers 选项中添加 x-amz-expiration