**HOS-NODEJS-SDK使用手册**

北京思特奇信息技术股份有限公司

文档信息

|  |  |
| --- | --- |
| **文档名称** | HOS-NODEJS-SDK使用手册 |
| **链接地址** |  |
| **文件状态** | □ **草稿** ☑ **正式发布** □ **正在修改** |
| **负 责 人** |  |
| **作 者** | 武泽昭 |
| **审 核 人** | 丁荣波 |
| **批 准 人** |  |

变更记录

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **变更序号** | **变更原因** | **变更页码** | **变更前版本号** | **变更后版本号** | **更改人** | **批准人** | **更改日期** | **备注** |
| **1** | 创建 |  | **1.0.0** |  | 武泽昭 |  | 2023.04.23 |  |
| **2** |  |  |  |  |  |  |  |  |
| **3** |  |  |  |  |  |  |  |  |

目录

[HOS-NODEJS-SDK使用手册 1](#_Toc22385)

[1 概述 7](#_Toc29199)

[1.1 背景 7](#_Toc2359)

[1.2 适用范围 7](#_Toc21221)

[1.3 术语定义 7](#_Toc15320)

[1.4 参考资料 8](#_Toc5302)

[2 使用配置 9](#_Toc24040)

[2.1 基础环境 9](#_Toc6688)

[2.2 配置使用 9](#_Toc9341)

[2.2.1 服务加载 9](#_Toc29962)

[2.2.1.1 局域网加载 9](#_Toc28260)

[2.2.1.2 安装加载 10](#_Toc21385)

[2.2.2 使用实例 10](#_Toc14831)

[3 接口使用 12](#_Toc26863)

[3.1 服务认证 12](#_Toc2637)

[3.1.1 创建永久秘钥 12](#_Toc23622)

[3.1.2 获取永久秘钥 12](#_Toc16525)

[3.1.3 查询秘钥列表 13](#_Toc20431)

[3.1.4 删除永久秘钥 13](#_Toc19893)

[3.2 创建客户端 13](#_Toc28980)

[3.3 账户类(account) 14](#_Toc3731)

[3.3.1 查询账户详情 14](#_Toc16527)

[3.3.2 设置账户配额 16](#_Toc26430)

[3.3.3 查询账户配额 17](#_Toc27772)

[3.4 桶操作(bucket) 18](#_Toc9045)

[3.4.1 创建桶 18](#_Toc21442)

[3.4.2 查询桶详细信息 19](#_Toc2431)

[3.4.3 删除桶 21](#_Toc359)

[3.4.4 列举桶内对象 22](#_Toc17349)

[3.4.5 列举出桶内的版本对象 24](#_Toc24093)

[3.4.6 桶配额 26](#_Toc2986)

[3.4.6.1 设置桶的配额 26](#_Toc27251)

[3.4.6.2 查询桶配额 27](#_Toc16443)

[3.4.7 桶策略 29](#_Toc12075)

[3.4.7.1 桶策略语法 29](#_Toc32108)

[3.4.7.2 设置桶策略 32](#_Toc15782)

[3.4.7.3 查询桶策略 34](#_Toc18294)

[3.4.8 ACL管控 35](#_Toc10306)

[3.4.8.1 设置桶ACL 35](#_Toc12821)

[3.4.8.2 查询桶ACL 39](#_Toc23330)

[3.4.9 桶版本管理 41](#_Toc27358)

[3.4.9.1 开启/关闭桶的版本状态 41](#_Toc16310)

[3.4.9.2 查询桶的版本控制状态 44](#_Toc1785)

[3.4.10 生命周期管理 45](#_Toc10746)

[3.4.10.1 设置生命周期 45](#_Toc14347)

[3.4.10.2 查询生命周期 49](#_Toc17484)

[3.4.10.3 删除生命周期 52](#_Toc26902)

[3.4.11 桶标签 53](#_Toc5956)

[3.4.11.1 设置桶标签 53](#_Toc2955)

[3.4.11.2 查询桶标签 55](#_Toc29938)

[3.4.11.3 删除桶标签 56](#_Toc24055)

[3.5 对象操作(object) 57](#_Toc2927)

[3.5.1 对象上传 57](#_Toc224)

[3.5.1.1 流方式上传 57](#_Toc16557)

[3.5.1.2 本地文件上传 60](#_Toc5080)

[3.5.2 下载对象 63](#_Toc18903)

[3.5.2.1 下载文件 63](#_Toc9161)

[3.5.2.2 获取对象流 64](#_Toc17106)

[3.5.3 复制对象 66](#_Toc15626)

[3.5.4 解冻对象 68](#_Toc28311)

[3.5.5 查询对象信息 70](#_Toc18115)

[3.5.6 删除对象 72](#_Toc20345)

[3.5.7 对象ACL 74](#_Toc15001)

[3.5.7.1 设置对象ACL 74](#_Toc32414)

[3.5.7.2 查询对象ACL 77](#_Toc8910)

[3.5.8 对象标签 79](#_Toc9263)

[3.5.8.1 设置对象标签 79](#_Toc23085)

[3.5.8.2 查询对象标签 81](#_Toc15878)

[3.5.8.3 删除对象标签 82](#_Toc9332)

# 概述

## 背景

思特奇对象存储系统（NCOSS）为客户提供海量、弹性、低成本、高可用的存储服务。用户可以通过Web门户和HTTP REST接口进行数据的访问和管理，其中基于HTTP 方式调用，需要开发者了解NCOSS组件使用方式，步骤稍显繁琐，因此我们将NCOSS所能提供的功能封装成SDK提供给开发人员，方便开发人员使用。

## 适用范围

针对于对象存储系统提供Node.js版本的SDK，供开发者使用，当前HOS-NODEJS-SDK版本：1.0.0。

对象存储系统版本：3.4.0

UAAS系统版本：

## 术语定义

|  |  |
| --- | --- |
| **术语/缩略语** | **解释** |
| NCOSS | 对象存储系统，高性能、分布式、强一致性的对象存储系统，主要括bucket操作管理以及 bucket 内对象存储操作管理 |
| HOS | NCOSS 本身的HOS 风格API |
| Bucket | 存储空间，作为一个容器存放对象(Object) |
| Object | 对象，操作的基本数据单元 |
| account | 账户ID |
| endpoint | 对象存储服务访问端点，为URL路径格式.示例http://10.20.29.17:7079/v1 |
| AccessKey | UAAS 访问秘钥 Access Key |
| SecretKey | UAAS 访问秘钥Secret Key |

## 参考资料

《对象存储系统API文档V3.doc》

《UAAS-JAVA-SDK使用手册》

《NCDS统一认证授权系统UAAS使用手册》版本 V1.0

# 使用配置

## 基础环境

请确保主机有Node环境和npm环境。

调试版本：

* Node：v14.17.4
* Npm：6.14.14

## 配置使用

### 服务加载

服务加载是引入NODEJS包，提供两种引入方式，可根据具体的使用情况确定使用。

#### 局域网加载

因为是局域网搭建，所以请手动将NCOSS-NODEJS-SDK压缩包解压至Node项目中的node\_modules文件夹下。其中该SDK依赖列表如下，请确保您的项目安装以下依赖。

|  |
| --- |
| "dependencies": {  "async": "^3.1.0",  "block-stream2": "^2.0.0",  "browser-or-node": "^1.3.0",  "buffer-crc32": "^0.2.13",  "co": "^4.6.0",  "crypto-browserify": "^3.12.0",  "es6-error": "^4.1.1",  "fast-xml-parser": "^4.1.3",  "ipaddr.js": "^2.0.1",  "json-stream": "^1.0.0",  "lodash": "^4.17.21",  "mime-types": "^2.1.14",  "mkdirp": "^0.5.1",  "query-string": "^7.1.1",  "through2": "^3.0.1",  "web-encoding": "^1.1.5",  "xml": "^1.0.0",  "xml2js": "^0.4.15",  "xmlhttprequest": "^1.8.0"  } |

#### 安装加载

可以将该SDK上传到node官网库中，同时通过命令安装到node项目中。

|  |
| --- |
| npm install ncoss |

### 使用实例

服务实例使用首先先定义引入，根据分配的账户信息获取秘钥，进而进行桶对象的操作，秘钥分成临时秘钥(24小时)和永久秘钥(不过期)两种，可根据情况选择使用，具体使用如下：

|  |
| --- |
| // 引入ncoss模块  var NCOSS = require('ncoss')  // 永久秘钥创建客户端连接  var client= new NCOSS.Client({  endPoint: '172.18.232.192', // ncoss服务端IP地址  port: 8089, // ncoss服务端端口  path: '/v1', // ncoss服务端接口统一前缀  accessKey: 'MTXZY45CMR7ET151L7OV', // UAAS系统分配accessKey  secretKey: 'GR3Af1E7o1xosqy6rfBDNE1mmOSzMfjWLhccMMUN', // UAAS系统分配secretKey  accountId: '7c9dfff2139b11edbc330391d2a979b2', //UAAS系统分配的租户id  })  // 创建24小时秘钥客户端  var client= new NCOSS.Client({  endPoint: '172.18.232.192', // ncoss服务端IP地址  port: 8089, // ncoss服务端端口  path: '/v1', // ncoss服务端接口统一前缀  username: 'test\_user1', // UAAS系统分配的用户名  password: 'TEST#ps@857', // UAAS系统的用户名密码  scopeName:'test\_pro1', // UAAS系统分配的用户名  uaasURL:'http://172.18.232.192:6020/v3/auth/tokens' // UAAS系统申请秘钥URI  }) |

# 接口使用

## 服务认证

### 创建永久秘钥

请注意：一个用户只能创建四个永久秘钥。

请引入permanentKey.js脚本来获取秘钥信息。

|  |
| --- |
| // 引入永久秘钥脚本  var permanentKey = require('ncoss/dist/main/permanentKey');  /\*\*  \*创建永久秘钥  \* @param {\*} username 用户名  \* @param {\*} password 用户名密码  \* @param {\*} scopeName 租户名  \* @param {\*} ip UAAS系统IP  \* @param {\*} port UAAS系统端口  \*/  permanentKey.createPermanent(username, password, scopeName, ip, port).then(result =>{  console.log(result);  }) |

### 获取永久秘钥详情

请引入permanentKey.js脚本来获取秘钥信息。

|  |
| --- |
| // 引入永久秘钥脚本  var permanentKey = require('ncoss/dist/main/permanentKey');  /\*\*  \*创建永久秘钥  \* @param {\*} username 用户名  \* @param {\*} password 用户名密码  \* @param {\*} scopeName 租户名  \* @param {\*} ip UAAS系统IP  \* @param {\*} port UAAS系统端口  \* @param {\*} accessKey 需要查询秘钥的accessKey  \*/  permanentKey.getPermanent(username, password, scopeName, ip, port,accessKey).then(result =>{  console.log(result);  }) |

### 查询永久秘钥列表

请引入permanentKey.js脚本来获取秘钥信息。

|  |
| --- |
| // 引入永久秘钥脚本  var permanentKey = require('ncoss/dist/main/permanentKey');  /\*\*  \*创建永久秘钥  \* @param {\*} username 用户名  \* @param {\*} password 用户名密码  \* @param {\*} scopeName 租户名  \* @param {\*} ip UAAS系统IP  \* @param {\*} port UAAS系统端口  \*/  permanentKey.listPermanent(username, password, scopeName, ip, port).then(result =>{  console.log(result);  }) |

### 删除永久秘钥

请引入permanentKey.js脚本来获取秘钥信息。

|  |
| --- |
| // 引入永久秘钥脚本  var permanentKey = require('ncoss/dist/main/permanentKey');  /\*\*  \*创建永久秘钥  \* @param {\*} username 用户名  \* @param {\*} password 用户名密码  \* @param {\*} scopeName 租户名  \* @param {\*} ip UAAS系统IP  \* @param {\*} port UAAS系统端口  \*/  permanentKey.deletrAccessKey(username, password, scopeName, ip, port, accessKey); |

## 创建客户端

创建客户端的方式有一种，有永久秘钥的方式和24小时秘钥的方式

|  |
| --- |
| // 引入ncoss模块  var NCOSS = require('ncoss')  // 永久秘钥创建客户端连接  var client= new NCOSS.Client({  endPoint: '172.18.232.192', // ncoss服务端IP地址  port: 8089, // ncoss服务端端口  path: '/v1', // ncoss服务端接口统一前缀  accessKey: 'MTXZY45CMR7ET151L7OV', // UAAS系统分配accessKey  secretKey: 'GR3Af1E7o1xosqy6rfBDNE1mmOSzMfjWLhccMMUN', // UAAS系统分配secretKey  accountId: '7c9dfff2139b11edbc330391d2a979b2', //UAAS系统分配的租户id  partSize:10\*1024\*1024, //大文件上传时每次分片大小  maximumPartSize: 5\*1024\*1024\*1024, // 设置每片的最大大小 5K~5G之间  maxObjectSize:5\*1024\*1024\*1024\*1024 // 对象的最大大小  })  // 创建24小时秘钥客户端  var client= new NCOSS.Client({  endPoint: '172.18.232.192', // ncoss服务端IP地址  port: 8089, // ncoss服务端端口  path: '/v1', // ncoss服务端接口统一前缀  username: 'test\_user1', // UAAS系统分配的用户名  password: 'TEST#ps@857', // UAAS系统的用户名密码  scopeName:'test\_pro1', // UAAS系统分配的用户名  uaasURL:'http://172.18.232.192:6020/v3/auth/tokens', // UAAS系统申请秘钥URI  partSize:10\*1024\*1024, //大文件上传时每次分片大小  maximumPartSize: 5\*1024\*1024\*1024, // 设置每片的最大大小 5K~5G之间  maxObjectSize:5\*1024\*1024\*1024\*1024 // 对象的最大大小  }) |

## 账户类(account)

### 查询账户详情

1. 方法：getAccountDetail

获取当前账户的详细信息，包括账户下桶(bucket)个数、对象(object)个数、账户存储空间使用情况。

1. 请求参数说明：

| **Name** | **Require** | **Type** | **Description** |
| --- | --- | --- | --- |
| callback | True | function | 回调函数，会将结果 |

1. 返回参数：

| **Name** | **Type** | **Description** |
| --- | --- | --- |
| bucketCount | number | 账户下桶个数 |
| objectCount | number | 账户下对象数量 |
| bytesUsed | number | 账户下已创建对象所占用的存储空间，单位Byte |

1. 状态码说明：

| **Name** | **Description** |
| --- | --- |
| 200- OK | 成功 |
| 400- Bad Request | 请求中的某些内容无效 |
| 403 - Forbidden | 请求没有权限 |

1. 示例代码:

|  |
| --- |
| var NCOSS = require('ncoss')  // 永久秘钥创建客户端连接  var client= new NCOSS.Client({  endPoint: '172.18.232.192',  port: 8089,  path: '/v1',  accessKey: 'MTXZY45CMR7ET151L7OV',  secretKey: 'GR3Af1E7o1xosqy6rfBDNE1mmOSzMfjWLhccMMUN',  accountId: '7c9dfff2139b11edbc330391d2a979b2'  })  // 调用函数获取账户详情  client.getAccountDetail(function (e, buckets) {  if (e) return console.log(e)  console.log('buckets :', buckets)  }) |

### 设置账户配额

1. 方法：setAccountQuota

账户配额值必须为非负整数，单位为Byte（字节）。默认配额为0，表 示没有配额限制。配额设置后，如果想取消配额限制，可以把配额设置为0。

1. 请求参数说明：

| **Name** | **Require** | **Type** | **Description** |
| --- | --- | --- | --- |
| accountQuota | True | number | 指定账户配额值，单位：Byte |
| callback | True | function | 回调函数 |

1. 返回值说明：

| **Name** | **Type** | **Description** |
| --- | --- | --- |
| err | object | 请求如果有错误，会返回该错误对象 |

1. 状态码：

| **Name** | **Description** |
| --- | --- |
| 200- OK | 成功 |
| 400 - Bad Request | 请求中的某些内容无效 |
| 403 - Forbidden | 请求没有权限 |

1. 示例代码：

|  |
| --- |
| var NCOSS = require('ncoss')  // 永久秘钥创建客户端连接  var client= new NCOSS.Client({  endPoint: '172.18.232.192',  port: 8089,  path: '/v1',  accessKey: 'MTXZY45CMR7ET151L7OV',  secretKey: 'GR3Af1E7o1xosqy6rfBDNE1mmOSzMfjWLhccMMUN',  accountId: '7c9dfff2139b11edbc330391d2a979b2'  })  // 调用函数设置账户配额  client.setAccountQuota(1024\*1024\*1024, (err) => {  if (err) throw err  console.log('Set bucket quota')  }) |

### 查询账户配额

1. 方法：**getAccountQuota**

查询账户配额信息。配额值的单位为Byte（字节），0代表不设上限。

1. 返回值说明：

| **Name** | **Type** | **Description** |
| --- | --- | --- |
| err | object | 请求如果有错误，会返回该错误对象 |
| result | object | 配额对象 |
| StorageQuota | number | 账户配额值，单位：Byte |

1. 状态码：

| **Name** | **Description** |
| --- | --- |
| 200- OK | 成功 |
| 400 - Bad Request | 请求中的某些内容无效 |
| 403 - Forbidden | 请求没有权限 |

1. 示例代码：

|  |
| --- |
| var NCOSS = require('ncoss')  // 永久秘钥创建客户端连接  var client= new NCOSS.Client({  endPoint: '172.18.232.192',  port: 8089,  path: '/v1',  accessKey: 'MTXZY45CMR7ET151L7OV',  secretKey: 'GR3Af1E7o1xosqy6rfBDNE1mmOSzMfjWLhccMMUN',  accountId: '7c9dfff2139b11edbc330391d2a979b2'  })  client.getAccountQuota(function (err, res) {  if (err) {  return console.log(err)  }  console.log(res)  console.log("Success")  }) |

## 桶操作(bucket)

### 创建桶

1. 方法：**makeBucket**

桶是用户用于存储对象的容器，所有的对象都必须隶属于某个桶。

1. 请求参数说明：

| **Name** | **Require** | **Type** | **Description** |
| --- | --- | --- | --- |
| bucketName | True | string | 桶的唯一名称，长度不能超过255字符，桶名称不能包含斜杠（/）字符，因为该字符分隔桶和对象名称 |
| callback | True | function | 回调函数 |

1. 返回值说明：

| **Name** | **Type** | **Description** |
| --- | --- | --- |
| err | object | 请求如果有错误，会返回该错误对象 |

1. 状态码：

| **Code** | **Reason** |
| --- | --- |
| 200- OK | 成功 |
| 400-Bad Request | 请求中的某些内容无效 |
| 403-Forbidden | 请求没有权限 |
| 409-Conflict | 存储冲突，要创建的存储桶名称已被占用 |

1. 示例代码：

|  |
| --- |
| var NCOSS = require('ncoss')  // 永久秘钥创建客户端连接  var client= new NCOSS.Client({  endPoint: '172.18.232.192',  port: 8089,  path: '/v1',  accessKey: 'MTXZY45CMR7ET151L7OV',  secretKey: 'GR3Af1E7o1xosqy6rfBDNE1mmOSzMfjWLhccMMUN',  accountId: '7c9dfff2139b11edbc330391d2a979b2'  })  // Create a bucket name my-bucketname.  client.makeBucket('jssdk', function(e) {  if (e) {  return console.log(e)  }  console.log("Success")  }) |

### 查询桶详细信息

1. 方法：**bucketDetail**

返回指定桶的详细信息，包括桶内对象个数、对象字节数统计。

1. 请求参数说明：

| **Name** | **Require** | **Type** | **Description** |
| --- | --- | --- | --- |
| bucketName | True | string | 需要查询详情的桶名称 |
| callback | True | function | 回调函数 |

1. 返回值说明：

| **Name** | **Type** | **Description** |
| --- | --- | --- |
| err | object | 请求如果有错误，会返回该错误对象 |
| result | object | 成功结果对象 |
| bytesUsed | number | 当前桶下已使用的空间。单位：Byte |
| objectCount | number | 当前桶下对象个数 |
| date | string | 创建日期，GMT格式时间字符串 |
| requestId | string | 请求ID |

1. 状态码：

| **Code** | **Reason** |
| --- | --- |
| 1. OK | 成功 |
| 1. BadRequest | 请求中的某些内容无效 |
| 403- Forbidden | 请求没有权限 |
| 404- Not Found | 请求的资源不存在 |

1. 示例代码：

|  |
| --- |
| var NCOSS = require('ncoss')  // 永久秘钥创建客户端连接  var client= new NCOSS.Client({  endPoint: '172.18.232.192',  port: 8089,  path: '/v1',  accessKey: 'MTXZY45CMR7ET151L7OV',  secretKey: 'GR3Af1E7o1xosqy6rfBDNE1mmOSzMfjWLhccMMUN',  accountId: '7c9dfff2139b11edbc330391d2a979b2'  })  client.getBucketDetail("my-bucketname", function(e, buckets) {  if (e) return console.log(e)  console.log('buckets :', buckets)  }) |

### 删除桶

1. 方法：**removeBucket**

不需要保留某个存储空间时，可以通过桶的名称将其删除，但是桶删除 时确保桶中不包含任何数据。

前提条件：

* 已删除Bucket中所有的对象（Object）
* 已删除Bucket中因分片上传或断点续传产生的碎片（Part）

1. 请求参数说明：

| **Name** | **Require** | **Type** | **Description** |
| --- | --- | --- | --- |
| bucketName | True | string | 需要删除的桶名称 |
| callback | True | function | 回调函数 |

1. 返回值说明：

| **Name** | **Type** | **Description** |
| --- | --- | --- |
| err | object | 请求如果有错误，会返回该错误对象 |

1. 状态码：

| **Code** | **Reason** |
| --- | --- |
| 200- OK | 成功 |
| 400-Bad Request | 请求中的某些内容无效 |
| 403-Forbidden | 请求没有权限 |
| 404-Not Found | 请求的资源不存在 |
| 409-Conflict | 存储冲突，删除桶不为空 |

1. 示例代码：

|  |
| --- |
| var NCOSS = require('ncoss')  // 永久秘钥创建客户端连接  var client= new NCOSS.Client({  endPoint: '172.18.232.192',  port: 8089,  path: '/v1',  accessKey: 'MTXZY45CMR7ET151L7OV',  secretKey: 'GR3Af1E7o1xosqy6rfBDNE1mmOSzMfjWLhccMMUN',  accountId: '7c9dfff2139b11edbc330391d2a979b2'  })  // Remove a bucket name my-bucketname.  // This operation will only work if your bucket is empty.  client.removeBucket('my-bucketname', function(e) {  if (e) {  return console.log(e)  }  console.log("Success")  }) |

### 列举桶内对象

1. 方法：**listObjects**

列出桶内对象列表，对象按名称排序。如果存储桶开启了版本控制，此请求对象列表中只返回对象的最新版本，不包含对象的历史版本。

1. 请求参数说明：

| **Name** | **Require** | **Type** | **Description** |
| --- | --- | --- | --- |
| bucketName | True | string | 桶名称 |
| prefix | False | string | 限制查询的对象名称前缀 |
| recursive | False | boolean | 虚拟目录 |
| listOpts | False | object | 查询条件对象 |
| Marker | False | string | 例如，对于一个字符串 x，约束列表为名称大于x的条目 |
| MaxKeys | False | number | 限制每次查询出的数量，有效值[0-1000] |

1. 返回值说明：

| **Name** | **Type** | **Description** |
| --- | --- | --- |
| err | object | 请求如果有错误，会返回该错误对象 |
| name | string | 对象名称 |
| lastModified | string | 最新修改时间 |
| etag | string | 对象标签 |
| size | number | 对象大小，单位：字节 |

1. 状态码说明：

| **Code** | **Reason** |
| --- | --- |
| 200- OK | 成功 |
| 400 - Bad Request | 请求中的某些内容无效 |
| 403 - Forbidden | 请求没有权限 |
| 404 - Not Found | 请求的资源不存在 |

1. 示例代码：

|  |
| --- |
| var NCOSS = require('ncoss')  // 永久秘钥创建客户端连接  var client= new NCOSS.Client({  endPoint: '172.18.232.192',  port: 8089,  path: '/v1',  accessKey: 'MTXZY45CMR7ET151L7OV',  secretKey: 'GR3Af1E7o1xosqy6rfBDNE1mmOSzMfjWLhccMMUN',  accountId: '7c9dfff2139b11edbc330391d2a979b2'  })  // List all object paths in bucket my-bucketname.  var objectsStream = client.listObjects('bucket','',true,{})  objectsStream.on('data', function (obj) {  console.log(obj)  })  objectsStream.on('error', function (e) {  console.log(e)  }) |

### 列举出桶内的版本对象

1. 方法：**listObjects**

列出桶内所有对象列表，包括历史对象按名称排序，如果存储桶开启了版本控制，此请求对象列表中包含对象的历史版本。

1. 请求参数说明：

| **Name** | **Require** | **Type** | **Description** |
| --- | --- | --- | --- |
| bucketName | True | string | 桶名称 |
| prefix | False | string | 限制查询的对象名称前缀 |
| recursive | False | boolean | 虚拟目录 |
| listOpts | True | object | 查询条件对象 |
| Marker | False | string | 例如，对于一个字符串 x，约束列表为名称大于x的条目 |
| MaxKeys | False | number | 限制每次查询出的数量，有效值[0-1000] |
| IncludeVersion | False | boolean | 如果需要查出版本对象，此项必须为true |

1. 返回值说明：

| **Name** | **Type** | **Description** |
| --- | --- | --- |
| err | object | 请求如果有错误，会返回该错误对象 |
| name | string | 对象名称 |
| lastModified | string | 最新修改时间 |
| etag | string | 对象标签 |
| size | number | 对象大小，单位：字节 |
| versionId | string | 对象的版本ID |

1. 状态码说明：

| **Code** | **Reason** |
| --- | --- |
| 200- OK | 成功 |
| 400 - Bad Request | 请求中的某些内容无效 |
| 403 - Forbidden | 请求没有权限 |
| 404 - Not Found | 请求的资源不存在 |

1. 示例代码：

|  |
| --- |
| var NCOSS = require('ncoss')  // 永久秘钥创建客户端连接  var client= new NCOSS.Client({  endPoint: '172.18.232.192',  port: 8089,  path: '/v1',  accessKey: 'MTXZY45CMR7ET151L7OV',  secretKey: 'GR3Af1E7o1xosqy6rfBDNE1mmOSzMfjWLhccMMUN',  accountId: '7c9dfff2139b11edbc330391d2a979b2'  })  var objectsStreamWithVersions = client.listObjects('jssdk', '', true, {  IncludeVersion: true  })  objectsStreamWithVersions.on('data', function (obj) {  console.log(obj)  })  objectsStreamWithVersions.on('error', function (e) {  console.log(e)  }) |

### 桶配额

#### 设置桶的配额

1. 方法：setBucketQuota

桶配额值必须为非负整数。默认配额为0，表示没有限制配额。配额设置后，如果想取消配额限制，可以把配额设置为0，桶配额提供 storageQuota（字节数限制，单位为Byte） 与 storageMaxCount（桶内对象个数限制，非负整数）两项配置。

1. 请求参数说明：

| **Name** | **Require** | **Type** | **Description** |
| --- | --- | --- | --- |
| bucketName | True | string | 桶名称 |
| storageQuota | False | number | 字节数限制，单位为Byte |
| StorageMaxCount | False | number | 桶内对象 个数限制，非负整数，默认配额为0，表示没有限制配额 |
| callback | True | function | 回调函数 |

1. 返回值说明：

| **Name** | **Type** | **Description** |
| --- | --- | --- |
| err | object | 请求如果有错误，会返回该错误对象 |

1. 状态码说明：

| **Code** | **Reason** |
| --- | --- |
| 200- OK | 成功 |
| 400-Bad Request | 请求中的某些内容无效 |
| 403 - Forbidden | 请求没有权限 |
| 404 - Not Found | 请求的资源不存在 |

1. 示例代码：

|  |
| --- |
| var NCOSS = require('ncoss')  // 永久秘钥创建客户端连接  var client= new NCOSS.Client({  endPoint: '172.18.232.192',  port: 8089,  path: '/v1',  accessKey: 'MTXZY45CMR7ET151L7OV',  secretKey: 'GR3Af1E7o1xosqy6rfBDNE1mmOSzMfjWLhccMMUN',  accountId: '7c9dfff2139b11edbc330391d2a979b2'  })  client.setBucketQuota('jssdk', 1024\*1024\*1024,100, (err) => {  if (err) throw err  console.log('Set bucket quota')  }) |

#### 查询桶配额

1. 方法：getBucketQuota

查询桶的配额信息。

1. 请求参数说明：

| **Name** | **Required** | **Type** | **Description** |
| --- | --- | --- | --- |
| bucketName | True | string | 桶名称 |
| callback | True | function | 回调函数 |

1. 返回值说明：

| **Name** | **Type** | **Description** |
| --- | --- | --- |
| err | object | 请求如果有错误，会返回该错误对象 |
| result | object | 成功访问结果对象 |
| StorageQuota | number | 字节数限制，单位为Byte |
| StorageMaxCount | number | 桶内对象 个数限制，非负整数，默认配额为0，表示没有限制配额 |

1. 状态码说明

| **Code** | **Reason** |
| --- | --- |
| 200- OK | 成功 |
| 400-Bad Request | 请求中的某些内容无效 |
| 403-Forbidden | 请求没有权限 |
| 404 - Not Found | 请求的资源不存在 |

1. 示例代码

|  |
| --- |
| var NCOSS = require('ncoss')  // 永久秘钥创建客户端连接  var client= new NCOSS.Client({  endPoint: '172.18.232.192',  port: 8089,  path: '/v1',  accessKey: 'MTXZY45CMR7ET151L7OV',  secretKey: 'GR3Af1E7o1xosqy6rfBDNE1mmOSzMfjWLhccMMUN',  accountId: '7c9dfff2139b11edbc330391d2a979b2'  })  client.getBucketQuota('jssdk', function (err,res){  if (err) {  return console.log(err)  }  console.log(res)  console.log("Success")  }) |

### 桶策略

#### 桶策略语法

首先对策略组成进行描述

{  
 // 策略配置的版本号，由用户定义  
 "Version": "2012-10-17",   
 // 策略规则配置，可指定多条策略规则  
 "Statement": [{  
 // 策略规则ID，可用户指定，如不指定则服务自动生成一条  
 "Sid": "sid",  
 // 策略规则指定的HOS API接口标识，可参照下表填写  
 "Action": [  
 "action"  
 ],  
 // 策略规则效果，具体表现为允许或拒绝，有效值为Allow（允许），Deny（拒绝）；  
 "Effect": "Allow",  
 // 策略规则指定的资源参照下面resource配置  
 "Resource": "resource",  
 // 策略规则指定的授权人 参照下面Principal配置  
 "Principal": {  
 "HWS": [  
 "account\_id:root"  
 ]  
 }  
 }  
 ]  
}

**Resource格式：**

策略规则指定的资源,格式如下示例：

1. "\*" //资源范围为桶及桶内所有对象；

2. "${bucket}" //资源范围为桶；

3. "${bucket/\*}" //资源范围为桶内所有对象；

4. "${bucket/object}"， //资源范围为桶内指定对象；

5. ["${bucket/object1}","${bucket/object2}"] //资源范围为桶内指定的多个对象。

**Principal配置：**

策略规则指定的授权人，格式如下示例：

1."\*" //授权范围为说有人包括匿名访问；

2."HWS":["${account1\_id}:root","${account2\_id}:root"] //授权范围为整个账户，需指定账户ID；

3."HWS":["${account\_id}:user/${user1}","${account\_id}:user/${user2}"] //授权范围指定用户,需指定账户ID，用户名或ID；

4."CanonicalUser":["${user1\_id}",${user2\_id}"] //授权范围指定用户,需指定用户ID；

HOS API 可配置的 Action如下表示例：

|  |  |
| --- | --- |
| **Action** | **Description** |
| ListBucket | 查询桶内对象列表，包含查询桶内全部对象列表 |
| HeadBucket | 查询桶详细 |
| PutBucketPolicy | 设置更新桶策略 |
| DeleteBucketPolicy | 删除桶策略 |
| GetBucketPolicy | 查询桶策略 |
| PutBucketAcl | 设置更新桶ACL |
| GetBucketAcl | 查询桶ACL |
| PutBucketTagging | 设置桶标签 |
| GetBucketTagging | 查询桶标签 |
| DeleteBucketTagging | 删除桶标签 |
| PutBucketQuota | 设置更新桶配额 |
| GetBucketQuota | 查询桶配额 |
| PutBucketVersioning | 开启关闭桶的版本控制 |
| GetBucketVersioning | 查询桶的版本控制状态 |
| PutBucketLifecycle | 设置或更新桶的生命周期配置 |
| GetBucketLifecycle | 查询桶的生命周期配置 |
| DeleteBucketLifecycle | 删除桶的生命周期配置 |
| PutObject | 创建对象，包含复制对象，上传段对象 |
| PostObjectRestore | 解冻恢复对象 |
| GetObject | 下载对象 |
| DeleteObject | 删除对象 |
| PutObjectAcl | 设置或更新对象ACL |
| GetObjectAcl | 查询对象ACL |
| PutObjectTagging | 设置对象标签 |
| GetObjectTagging | 查询对象标签 |
| DeleteObjectTagging | 删除对象标签 |
| ListBucketMultipartUploads | 查询桶内已初始化的多段对象上传列表 |
| InitMultipartUpload | 初始化多段对象上传 |
| CompleteMultipartUpload | 完成多段对象上传 |
| AbortMultipartUpload | 终止多段对象上传 |
| ListMultiparts | 查询多段对象，已上传的段对象列表 |

#### 设置桶策略

1. 方法：**setBucketPolicy**

默认情况下资源（桶和对象）都是私有的，只有资源拥有者可以访问资源，其他用户在未经授权的情况下均无HOS访问权限。通过编写访问策略向其他帐户或者UAAS用户授予资源的控制权限。此操作属于覆盖操作，如果存在桶策略配置，新的策略将覆盖已存在的策略。

例子：您拥有一个桶，可以授权一个其他的UAAS用户上传对象到你的桶中；你也可以将桶开放给所有用户访问，即桶作为一个公共资源，能被互联网上任何人访问。

1. 请求参数说明：

| **Name** | **Require** | **Type** | **Description** |
| --- | --- | --- | --- |
| bucketName | True | string | 桶名称 |
| policy | True | string | 策略字符串，需满足语法。详细语法参照[桶策略语法](#_桶策略语法)  如果传入空串，代表删除该桶的所有策略 |
| callback | True | function | 回调函数 |

1. 返回值说明：

| **Name** | **Type** | **Description** |
| --- | --- | --- |
| err | object | 请求如果有错误，会返回该错误对象 |

1. 状态码说明：

| **Code** | **Reason** |
| --- | --- |
| 200- OK | 成功 |
| 400-Bad Request | 请求中的某些内容无效 |
| 403 - Forbidden | 请求没有权限 |
| 404 - Not Found | 请求的资源不存在 |

1. 示例代码

|  |
| --- |
| var NCOSS = require('ncoss')  // 永久秘钥创建客户端连接  var client= new NCOSS.Client({  endPoint: '172.18.232.192',  port: 8089,  path: '/v1',  accessKey: 'MTXZY45CMR7ET151L7OV',  secretKey: 'GR3Af1E7o1xosqy6rfBDNE1mmOSzMfjWLhccMMUN',  accountId: '7c9dfff2139b11edbc330391d2a979b2'  })  // Bucket policy - GET requests on "testbucket" bucket will not need authentication.  var policy = `  {  "Version": "2023-05-17",  "Statement": [  {  "Action": [  "ListBucket",  "HeadBucket"  ],  "Effect": "Allow",  "Principal": {  "HWS": [  "7c9dfff2139b11edbc330391d2a979b2:root"  ]  },  "Resource": [  "\*"  ],  "Sid": ""  },  {  "Action": [  "PutBucketPolicy",  "DeleteBucketPolicy"  ],  "Effect": "Allow",  "Principal": {  "HWS": [  "7c9dfff2139b11edbc330391d2a979b2:root"  ]  },  "Resource": [  "\*"  ],  "Sid": ""  }  ]  }  `  client.setBucketPolicy('jssdk', policy, (err) => {  if (err) throw err  console.log('Set bucket policy')  }) |

#### 查询桶策略

1. 方法：**getBucketPolicy**

查询桶设置的策略情况。

1. 请求参数说明：

| **Name** | **Require** | **Type** | **Description** |
| --- | --- | --- | --- |
| bucketName | True | string | 桶名称 |
| callback | True | function | 回调函数 |

1. 返回值说明：

| **Name** | **Type** | **Description** |
| --- | --- | --- |
| err | object | 请求如果有错误，会返回该错误对象 |
| policy | string | 策略字符串 |

1. 状态码说明：

| **Code** | **Reason** |
| --- | --- |
| 200- OK | 成功 |
| 400-BadRequest | 请求中的某些内容无效 |
| 403 - Forbidden | 请求没有权限 |
| 404 -Not Found | 请求的资源不存在 |

1. 示例代码：

|  |
| --- |
| var NCOSS = require('ncoss')  // 永久秘钥创建客户端连接  var client= new NCOSS.Client({  endPoint: '172.18.232.192',  port: 8089,  path: '/v1',  accessKey: 'MTXZY45CMR7ET151L7OV',  secretKey: 'GR3Af1E7o1xosqy6rfBDNE1mmOSzMfjWLhccMMUN',  accountId: '7c9dfff2139b11edbc330391d2a979b2'  })  // Retrieves the bucket policy and logs it to the console.  client.getBucketPolicy('jssdk', (err, policy) => {  if (err) throw err  console.log(`Bucket policy: ${policy}`)  }) |

### ACL管控

#### 设置桶ACL

1. 方法：**setBucketAcl**

HOS支持对桶操作进行权限控制。默认情况下，只有桶的拥有者才有该桶的读写权限。您也可以设置其他的访问策略。此操作属于覆盖操作，新的配置将覆盖桶已存在的ACL配置。

1. 请求参数说明：

| **Name** | **Require** | **Type** | **Description** |
| --- | --- | --- | --- |
| bucketName | True | string | 桶名称 |
| ACL | False | object | ACL对象 |
| owner | True | string | 桶的拥有者信息 |
| accessControlList | True | array | 权限访问控制容器 |
| accessControl | True | object | 权限控制对象 |
| Permission | True | string | 授权人权限 READ | WRITE | READ\_ACP | WRITE\_ACP | FULL\_CONTROL，详细浏览HOS支持桶的ACL权限元素含义表 即Permission参数 |
| GranteeType | True | string | 分别为CanonicalUserGrantee和GroupGrantee   * 如果为CanonicalUserGrantee，需要指定好userId * 如果为GroupGrantee，一类是AllUsers，代表所有人包括匿名用户。另一个是AuthenticatedUsers经过身份认证的所有用户 |
| ID | True | string | 用户ID，只有GranteeType参数为  CanonicalUserGrantee时，此参数是必填项 |
| URI | True | string | 只有GranteeType参数为GroupGrantee时，此参数是必填项  有效值：  **1**. <http://www.heredata.com/groups/global/AllUsers> 代表所有人包括匿名用户；  **2**. <http://www.heredata.com/groups/global/AuthenticatedUsers> 代表经过身份认证的所有用户。 |
| ACLString | False | string | **与ACL参数冲突，需要您自行组装ACL字符串，该字符串符合ACL语法。详细请浏览下面ACL语法** |
| callback | True | function | 回调函数 |

HOS支持桶的ACL权限元素含义表 即**Permission参数**

|  |  |  |
| --- | --- | --- |
| **权限** | **描述** | **对应动作** |
| READ | 获取该桶内对象列表、桶内多段任务、桶的元数据、桶的多版本 | ListBucket  ListBucketMultipartUploads  HeadBucket  ListBucketVersions |
| WRITE | 可以上传、覆盖和删除该桶内任何对象和段 | PutObject  DeleteObject  DeleteObjectVersion  AbortMultipartUpload |
| READ\_ACP | 可以获取对应的桶的权限控制列表（ACL） | GetBucketAcl |
| WRITE\_ACP | 更新对应桶的权限控制列表（ACL） | PutBucketAcl |
| FULL\_CONTROL | 拥有READ、WRITE的权限 | 包含以上所有关于对象操作动作 |

ACL语法如下

|  |
| --- |
| <AccessControlPolicy>  <Owner>  <ID>c7a4c02ee77b11eaa1cc01cf93dcddb3</ID>  </Owner>  <AccessControlList>  <Grant>  <Grantee xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="CanonicalUser">  <ID>user\_id<ID>  </Grantee>  <Permission>permission</Permission>  </Grant>  <Grant>  <Grantee xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="Group">  <URI>http://www.heredata.com/groups/global/AllUsers</URI> </Grantee>  <Permission>READ</Permission>  </Grant>  </AccessControlList>  </AccessControlPolicy> |

1. 返回值说明：

| **Name** | **Type** | **Description** |
| --- | --- | --- |
| err | object | 请求如果有错误，会返回该错误对象 |

1. 状态码说明：

| **Code** | **Reason** |
| --- | --- |
| 200- OK | 成功 |
| 400-BadRequest | 请求中的某些内容无效 |
| 403 - Forbidden | 请求没有权限 |
| 404 - Not Found | 请求的资源不存在 |

1. 示例代码：

|  |
| --- |
| var NCOSS = require('ncoss')  // 永久秘钥创建客户端连接  var client= new NCOSS.Client({  endPoint: '172.18.232.192',  port: 8089,  path: '/v1',  accessKey: 'MTXZY45CMR7ET151L7OV',  secretKey: 'GR3Af1E7o1xosqy6rfBDNE1mmOSzMfjWLhccMMUN',  accountId: '7c9dfff2139b11edbc330391d2a979b2'  })  var ACL = {}  // 设置拥有者信息  ACL['OwnerID'] = '7c9dfff2139b11edbc330391d2a979b2'  // 设置权限控制集合  const accessControlList = [];  // 第一种类型权限控制  var accessControl1 = {};  // 授权人权限 READ | WRITE | READ\_ACP | WRITE\_ACP | FULL\_CONTROL  accessControl1['Permission'] = 'READ'  // 设置授权信信息容器，只有两个值可选  // 1:CanonicalUser:规范用户需指定用户ID，与 元素 ID 一起使用  // 2:Group: 范围授权需指定元素URI，有效值为：  // http://www.heredata.com/groups/global/AllUsers 代表所有人包括匿名用户  // http://www.heredata.com/groups/global/AuthenticatedUsers 代表经过身份认证的所有用户。  accessControl1['GranteeType'] = 'CanonicalUser'  accessControl1['ID'] = '7c9dfff2139b11edbc330391d2a979b2'  // 第二种类型权限控制  var accessControl2 = {};  // 授权人权限 READ | WRITE | READ\_ACP | WRITE\_ACP | FULL\_CONTROL  accessControl2['Permission'] = 'WRITE'  accessControl2['GranteeType'] = 'Group'  accessControl2['URI'] = 'http://www.heredata.com/groups/global/AllUsers'  // 将权限控制push到集合中  accessControlList.push(accessControl1)  accessControlList.push(accessControl2)  ACL['accessControlList'] = accessControlList  client.setBucketACL('jssdk', ACL, function (err) {  if (err) {  return console.log(err)  }  console.log("Success")  })  const ACLString = '<AccessControlPolicy><Owner><ID>7c9dfff2139b11edbc330391d2a979b2</ID></Owner><AccessControlList><Grant><Grantee xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="CanonicalUser"><ID>7c9dfff2139b11edbc330391d2a979b2</ID></Grantee><Permission>READ</Permission></Grant><Grant><Grantee xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="Group"><URI>http://www.heredata.com/groups/global/AllUsers</URI></Grantee><Permission>WRITE</Permission></Grant></AccessControlList></AccessControlPolicy>';  client.setBucketACL('bucket', ACLString, function (err) {  if (err) {  return console.log(err)  }  console.log("Success")  }) |

#### 查询桶ACL

1. 方法：**getBucketAcl**
2. 请求参数说明：

| **Name** | **Require** | **Type** | **Description** |
| --- | --- | --- | --- |
| bucketName | True | string | 桶名称 |
| callback | True | function | 回调函数 |

1. 返回值说明：

| **Name** | **Type** | **Description** |
| --- | --- | --- |
| err | object | 请求如果有错误，会返回该错误对象 |
| ACL | object | 策略字符串 |
| Owner | object | 桶的拥有者信息 |
| ID | string | 需指定桶的所属账户ID |
| AccessControlList | array | 权限控制列表容器 |
| Grant | array | 权限控制信息容器 |
| Grantee | object | 授权信信息容器，注意前缀格式，xsi:type 有效值为CanonicalUser、  Group：   * **CanonicalUser**：规范用户需指定用户ID，与 元素 ID 一起使用； * **Group**：范围授权需指定URI，有效值为：   **1**. <http://www.heredata.com/groups/global/AllUsers> 代表所有人包括匿名用户；  **2**. <http://www.heredata.com/groups/global/AuthenticatedUsers> 代表经过身份认证的所有用户。 |
| URI | string | 授权人类型为Group 时，需指定授权范围。 |
| ID | string | 授权人类型为CanonicalUser 时，需指定用户ID。 |
| Permission | string | 授权人权限，有效值为：READ | WRITE | READ\_ACP | WRITE\_ACP | FULL\_CONTROL |

1. 状态码说明：

| **Code** | **Reason** |
| --- | --- |
| 200- OK | 成功 |
| 400-BadRequest | 请求中的某些内容无效 |
| 403 - Forbidden | 请求没有权限 |
| 404- Not Found | 请求的资源不存在 |

1. 示例代码：

|  |
| --- |
| var NCOSS = require('ncoss')  // 永久秘钥创建客户端连接  var client= new NCOSS.Client({  endPoint: '172.18.232.192',  port: 8089,  path: '/v1',  accessKey: 'MTXZY45CMR7ET151L7OV',  secretKey: 'GR3Af1E7o1xosqy6rfBDNE1mmOSzMfjWLhccMMUN',  accountId: '7c9dfff2139b11edbc330391d2a979b2'  })  client.getBucketACL('jssdk', function (err,res){  if (err) {  return console.log(err)  }  console.log(JSON.stringify(res))  console.log("Success")  }) |

### 桶版本管理

#### 开启/关闭桶的版本状态

1. 方法：**setBucketVersioning**

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。

本接口是用来设置桶的多版本状态，用来开启或暂停桶的多版本功能，默认情况下，桶没有开启多版本功能，当多版本状态开启后就无法关闭，只能暂停或恢复桶的多版本状态。

* 设置桶的多版本状态为Enabled，开启或恢复桶的多版本功能：

1. 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；
2. 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；
3. 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；
4. 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；

* 设置桶的多版本状态为Suspended，暂停桶的多版本功能：

1. 旧的版本数据继续保留；
2. 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；
3. 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；
4. 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；
5. 请求参数说明：

| **Name** | **Type** | **Type** | **Description** |
| --- | --- | --- | --- |
| bucketName | True | string | 桶名称 |
| versionConfig | True | object | 桶版本配置类 |
| Status | True | string | 有效值：Enabled || Suspended |
| callback | True | function | 回调函数 |

1. 返回值说明：

| **Name** | **Type** | **Description** |
| --- | --- | --- |
| err | object | 请求如果有错误，会返回该错误对象 |

1. 状态码说明：

| **Code** | **Reason** |
| --- | --- |
| 200- OK | 成功 |
| 400-BadRequest | 请求中的某些内容无效 |
| 403- Forbidden | 请求没有权限 |
| 404-Not Found | 请求的资源不存在 |

1. 示例代码:

|  |
| --- |
| var NCOSS = require('ncoss')  // 永久秘钥创建客户端连接  var client= new NCOSS.Client({  endPoint: '172.18.232.192',  port: 8089,  path: '/v1',  accessKey: 'MTXZY45CMR7ET151L7OV',  secretKey: 'GR3Af1E7o1xosqy6rfBDNE1mmOSzMfjWLhccMMUN',  accountId: '7c9dfff2139b11edbc330391d2a979b2'  })  var versioningStateConfig = {Status:"Enabled"}  client.setBucketVersioning("jssdk", versioningStateConfig, function (error){  if (error) {  return console.log(error)  }  console.log("Success")  }) |

#### 查询桶的版本控制状态

1. 方法：**getBucketVersioning**
2. 请求参数说明：

| **Name** | **Require** | **Type** | **Description** |
| --- | --- | --- | --- |
| bucketName | True | string | 桶名称 |
| callback | True | function | 回调函数 |

1. 返回值说明：

| **Name** | **Type** | **Description** |
| --- | --- | --- |
| err | object | 请求如果有错误，会返回该错误对象 |
| versionConfig | object | 桶版本配置类 |
| Status | string | 有效值：Enabled || Suspended |

1. 状态码说明:

| **Code** | **Reason** |
| --- | --- |
| 200- OK | 成功 |
| 400-BadRequest | 请求中的某些内容无效 |
| 403- Forbidden | 请求没有权限 |
| 404-Not Found | 请求的资源不存在 |

1. 示例代码：

|  |
| --- |
| var NCOSS = require('ncoss')  // 永久秘钥创建客户端连接  var client= new NCOSS.Client({  endPoint: '172.18.232.192',  port: 8089,  path: '/v1',  accessKey: 'MTXZY45CMR7ET151L7OV',  secretKey: 'GR3Af1E7o1xosqy6rfBDNE1mmOSzMfjWLhccMMUN',  accountId: '7c9dfff2139b11edbc330391d2a979b2'  })  client.getBucketVersioning('jssdk', function (err,res){  if (err) {  return console.log(err)  }  console.log(res)  console.log("Success")  }) |

### 生命周期管理

#### 设置生命周期

1. 方法：**setBucketLifecycle**

NCOSS支持设置生命周期（Lifecycle）规则，自动删除过期的文件（Object）和碎片，或将到期的文件转储为低频或归档存储类型，从而节省存储空间。

此操作将开启桶的生命周期配置，通过支持指定规则来实现定时删除或迁移桶中对象，如果桶已存在生命周期配置则将覆盖之前的配置，如果没有则新建。

生命周期规则配置格式校验规则：

|  |
| --- |
| 1. 规则ID唯一，最大长度255字节 |
| 2. 规则条数限制 1000条, (请求的 xml 大小不超过2MB) |
| 3. 如果过滤器设置了tag、 则这条规则不能设置 删除对象标记 或 删除初始化多段对象任务 |
| 4. 每条规则标签最多支持10个， Key 最大不超过128，Value 最大不超过256 |
| 5. 条件筛选，两个或两个以上条件只能使用And 元素，最多只有一个prefix，零个或多个tag |
| 7. 时间类型不可冲突，（Date 或 Days） |
| 8. Data 时间格式 ISO 8601，days 整数 |
| 9. Expiration 元素 Days 与 ExpiredObjectDeleteMarker 不可同时设置 |
| 10. 对于当前版本对象，一个规则内，同时设置过期与 存储类转换操作， 时间设置类型需相同 |
| 11. 过期时间必须大于前两个转换时间的最大值 |
| 12. 过滤器为空， 则适配桶中所有对象 |
| 13. 如果已存在按前缀配置的生命周期规则，则不允许再新增配置到整个桶的规则  如果已存在配置到整个桶的生命周期规则，则不允许再新增按前缀配置的规则 |
| 14. 生命周期规则设置，不支持逆向存储类转化 |
| 15. 时间节点判断基准对象的最后更新时间 |

1. 请求参数说明：

| **Name** | **Require** | **Type** | **Description** |
| --- | --- | --- | --- |
| bucketName | True | string | 桶名称 |
| lifecycleConfig | True | object | 生命周期规则集合，元素属性如下 |
| Rule | True | array | 生命周期规则配置容器，如果配置生命周期规则格式错误请参下方照表9做格式校验 |
| ID | False | string | 规则ID，用来标识一条规则, 最长不超过255 字节，同一个配置中，ID不可重复，如果不指定，对象存储默认生成一个UUID 作为其规则ID |
| Filter | True | object | Filter用于标识适用于生命周期规则的对象。Filter必须指定Prefix、Tag或And中的一个 |
| Prefix | False | string | 指定规则所适用的前缀（Prefix）, Prefix不可重复。  ● 若指定了Prefix，则表示此规则仅适用于Bucket中与Prefix匹配的Object。  ● 若Prefix置空，则表示此规则适用于Bucket中的所有Object。  注：如果有规则 将prefix 设置为空，则不可在设置有前缀的规则。 |
| Tag | False | string | 指定规则所适用的对象标签，可设置多个。 |
| Status | True | string | 有效值：Enabled、Suspended  Enabled： 启用规则  Suspended：禁用规则 |
| Expiration | False | object | 当前版本对象的过期配置 |
| Days | False | integer | 指定对象过期时间，单位days（天），以对象最后更新时间为准，向后计算过期时间。 |
| Date | Fasle | string | 指定一个日期，对象存储系统会对最后更新时间早于该日期的数据执行生命周期规则。日期必须服从ISO8601的格式，且要求是UTC的零点。  例：2021-07-23T5:00:00.000 实际北京时间 + 8 为 2021-07-23T13:00:00.000。 |
| ExpiredObjectDeleteMarker | False | boolean | 对象删除标记过期，有效值 ”true ”，与 Days 冲突。 |
| Transition | False | object | 指定Object在有效生命周期中, 何时将Object转储为IA、Archive存储类型 。  Bucket中的Standard Object可以转储为IA或Archive存储类型，但转储Archive存储类型的时间必须比转储IA存储类型的时间长。 |
| Days | Fasle | number | 指定对象类型转换时间，单位days（天），以对象最后更新时间为准，向后计算类型转换时间。 |
| Date | Fasle | string | 指定一个日期，NCOSS会对最后更新时间早于该日期的数据执行生命周期规则。日期必须服从ISO8601的格式，且要求是UTC的零点。  例：2021-07-23T5:00:00.000 实际北京时间 + 8 为 2021-07-23T13:00:00.000。 |
| StorageClass | Fasle | string | 指定Object转储的存储类型。  有效值：   * **IA ：低频访问存储 (温)** * **ARCHIVE ：归档存储 （冷）** |
| NoncurrentVersionExpiration | False | object | 历史版本对象的过期配置。 |
| NoncurrentDays | False | number | 指定历史版本对象过期时间，单位noncurrentDays（天），以对象最后更新时间为准，向后计算过期时间。 |
| NoncurrentVersionTransition | False | object | 指定Object历史版本在有效生命周期中, 何时将Object 历史版本转储为IA、Archive存储类型 。 |
| NoncurrentDays | False | number | 指定历史版本对象转换存储类型时间，单位noncurrentDays（天），以对象最后更新时间为准，向后计算转换时间。 |
| AbortIncompleteMultipartUpload | False | object | 初始化多段对象任务过期时间配置。 |
| DaysAfterInitiation | False | number | 初始化多段对象任务过期时间，单位DaysAfterInitiation(天)，以初始化对象任务最后更新时间为准，向后计算过期时间。 |
| callback | True | function | 回调函数 |

1. 返回值说明：

| **Name** | **Type** | **Description** |
| --- | --- | --- |
| err | object | 请求如果有错误，会返回该错误对象 |

1. 状态码说明：

| **Code** | **Reason** |
| --- | --- |
| 200- OK | 成功 |
| 400-BadRequest | 请求中的某些内容无效 |
| 403 - Forbidden | 请求没有权限 |
| 404 - Not Found | 请求的资源不存在 |

1. 示例代码：

|  |
| --- |
| var NCOSS = require('ncoss')  // 永久秘钥创建客户端连接  var client= new NCOSS.Client({  endPoint: '172.18.232.192',  port: 8089,  path: '/v1',  accessKey: 'MTXZY45CMR7ET151L7OV',  secretKey: 'GR3Af1E7o1xosqy6rfBDNE1mmOSzMfjWLhccMMUN',  accountId: '7c9dfff2139b11edbc330391d2a979b2'  })  const lifecycleConfig= {  Rule: [{  "Status": "Enabled",  "Filter": {  "Prefix":"2023",  },  "Expiration": {  "Days": "3650"  }  }  ]  }  client.setBucketLifecycle('jssdk',lifecycleConfig, function (err) {  if (err) {  return console.log(err)  }  console.log("Success")  })  // //Example to demonstrate Expiration Date  // const expirationDate = new Date()  // expirationDate.setDate(expirationDate.getDate() + 1) // Expire next day.  // expirationDate.setUTCHours(0,0,0,0)//Should be start of the day.(midnight)  // const lifecycleConfigWithExpirationDate= {  // Rule: [{  // "Status": "Enabled",  // "Filter": {  // "Prefix":"2023",  // },  // "Expiration": {  // "Date":expirationDate.toISOString()  // }  // }  // ]  // }  // client.setBucketLifecycle('bucket',lifecycleConfigWithExpirationDate, function (err) {  // if (err) {  // return console.log(err)  // }  // console.log("Success")  // }) |

#### 查询生命周期

1. 方法：**getBucketLifecycle**
2. 请求参数说明：

| **Name** | **Require** | **Type** | **Description** |
| --- | --- | --- | --- |
| bucketName | True | string | 桶名称 |
| callback | True | function | 回调函数 |

1. 返回值说明：

| **Name** | **Type** | **Description** |
| --- | --- | --- |
| lifecycleConfig | object |  |
| Rule | array | 生命周期规则集合，元素属性如下 |
| ID | string | 生命周期规则配置容器，如果配置生命周期规则格式错误请参下方照表9做格式校验 |
| Filter | object | 规则ID，用来标识一条规则, 最长不超过255 字节，同一个配置中，ID不可重复，如果不指定，对象存储默认生成一个UUID 作为其规则ID |
| Prefix | string | Filter用于标识适用于生命周期规则的对象。Filter必须指定Prefix、Tag或And中的一个 |
| Tag | string | 指定规则所适用的前缀（Prefix）, Prefix不可重复。  ● 若指定了Prefix，则表示此规则仅适用于Bucket中与Prefix匹配的Object。  ● 若Prefix置空，则表示此规则适用于Bucket中的所有Object。  注：如果有规则 将prefix 设置为空，则不可在设置有前缀的规则。 |
| Status | string | 指定规则所适用的对象标签，可设置多个。 |
| Expiration | object | 有效值：Enabled、Suspended  Enabled： 启用规则  Suspended：禁用规则 |
| Days | integer | 当前版本对象的过期配置 |
| Date | string | 指定对象过期时间，单位days（天），以对象最后更新时间为准，向后计算过期时间。 |
| ExpiredObjectDeleteMarker | boolean | 指定一个日期，对象存储系统会对最后更新时间早于该日期的数据执行生命周期规则。日期必须服从ISO8601的格式，且要求是UTC的零点。  例：2021-07-23T5:00:00.000 实际北京时间 + 8 为 2021-07-23T13:00:00.000。 |
| Transition | object | 对象删除标记过期，有效值 ”true ”，与 Days 冲突。 |
| Days | number | 指定Object在有效生命周期中, 何时将Object转储为IA、Archive存储类型 。  Bucket中的Standard Object可以转储为IA或Archive存储类型，但转储Archive存储类型的时间必须比转储IA存储类型的时间长。 |
| Date | string | 指定对象类型转换时间，单位days（天），以对象最后更新时间为准，向后计算类型转换时间。 |
| StorageClass | string | 指定一个日期，NCOSS会对最后更新时间早于该日期的数据执行生命周期规则。日期必须服从ISO8601的格式，且要求是UTC的零点。  例：2021-07-23T5:00:00.000 实际北京时间 + 8 为 2021-07-23T13:00:00.000。 |
| NoncurrentVersionExpiration | object | 指定Object转储的存储类型。  有效值：   * **IA ：低频访问存储 (温)** * **ARCHIVE ：归档存储 （冷）** |
| NoncurrentDays | number | 历史版本对象的过期配置。 |
| NoncurrentVersionTransition | object | 指定历史版本对象过期时间，单位noncurrentDays（天），以对象最后更新时间为准，向后计算过期时间。 |
| NoncurrentDays | number | 指定Object历史版本在有效生命周期中, 何时将Object 历史版本转储为IA、Archive存储类型 。 |
| AbortIncompleteMultipartUpload | object | 指定历史版本对象转换存储类型时间，单位noncurrentDays（天），以对象最后更新时间为准，向后计算转换时间。 |
| DaysAfterInitiation | number | 初始化多段对象任务过期时间配置。 |

1. 状态码说明：

| **Code** | **Reason** |
| --- | --- |
| 200- OK | 成功 |
| 400-BadRequest | 请求中的某些内容无效 |
| 403 - Forbidden | 请求没有权限 |
| 404 - Not Found | 请求的资源不存在 |

1. 示例代码：

|  |
| --- |
| var NCOSS = require('ncoss')  // 永久秘钥创建客户端连接  var client= new NCOSS.Client({  endPoint: '172.18.232.192',  port: 8089,  path: '/v1',  accessKey: 'MTXZY45CMR7ET151L7OV',  secretKey: 'GR3Af1E7o1xosqy6rfBDNE1mmOSzMfjWLhccMMUN',  accountId: '7c9dfff2139b11edbc330391d2a979b2'  })  client.getBucketLifecycle('jssdk', function (err,res){  if (err) {  return console.log(err)  }  console.log(res)  console.log("Success")  }) |

#### 删除生命周期

1. 方法：**removeBucketLifecycle**
2. 请求参数说明：

| **Name** | **Require** | **Type** | **Description** |
| --- | --- | --- | --- |
| bucketName | True | string | 桶名称 |
| callback | True | function | 回调函数 |

1. 返回值说明：

| **Name** | **Type** | **Description** |
| --- | --- | --- |
| err | object | 请求如果有错误，会返回该错误对象 |

1. 状态码说明：

| **Code** | **Reason** |
| --- | --- |
| 200- OK | 成功 |
| 400-BadRequest | 请求中的某些内容无效 |
| 403 - Forbidden | 请求没有权限 |
| 404 - Not Found | 请求的资源不存在 |

1. 示例代码：

|  |
| --- |
| var NCOSS = require('ncoss')  // 永久秘钥创建客户端连接  var client= new NCOSS.Client({  endPoint: '172.18.232.192',  port: 8089,  path: '/v1',  accessKey: 'MTXZY45CMR7ET151L7OV',  secretKey: 'GR3Af1E7o1xosqy6rfBDNE1mmOSzMfjWLhccMMUN',  accountId: '7c9dfff2139b11edbc330391d2a979b2'  })  client.removeBucketLifecycle('jssdk', function (err) {  if (err) {  return console.log(err)  }  console.log("Success")  }) |

### 桶标签

#### 设置桶标签

1. 方法：**setBucketTagging**

给桶添加标签配置，如果桶已存在标签配置则覆盖之前的标签配置。此操作属于覆盖操作，标签最多支持10对。

1. 请求参数说明：

| **Name** | **Require** | **Type** | **Description** |
| --- | --- | --- | --- |
| bucketName | True | string | 桶名称 |
| tagsObj | True | object | 标签列表的容器 |
| callback | True | function | 回调函数 |

1. 返回值说明：

| **Name** | **Type** | **Description** |
| --- | --- | --- |
| err | object | 请求如果有错误，会返回该错误对象 |

1. 状态码说明：

| **Code** | **Reason** |
| --- | --- |
| 200- OK | 成功 |
| 400-BadRequest | 请求中的某些内容无效 |
| 403 - Forbidden | 请求没有权限 |
| 404 - Not Found | 请求的资源不存在 |

1. 示例代码：

|  |
| --- |
| var NCOSS = require('ncoss')  // 永久秘钥创建客户端连接  var client= new NCOSS.Client({  endPoint: '172.18.232.192',  port: 8089,  path: '/v1',  accessKey: 'MTXZY45CMR7ET151L7OV',  secretKey: 'GR3Af1E7o1xosqy6rfBDNE1mmOSzMfjWLhccMMUN',  accountId: '7c9dfff2139b11edbc330391d2a979b2'  })  var tagsMap = {"tagkey":"tagvalue"}  client.setBucketTagging('jssdk', tagsMap, function (err){  if (err) {  return console.log(err)  }  console.log("Success")  }) |

#### 查询桶标签

1. 方法：**getBucketTagging**
2. 请求参数说明：

| **Name** | **Require** | **Type** | **Description** |
| --- | --- | --- | --- |
| bucketName | True | string | 桶名称 |
| callback | True | function | 回调函数 |

1. 返回值说明：

| **Name** | **Type** | **Description** |
| --- | --- | --- |
| err | object | 请求如果有错误，会返回该错误对象 |
| tagsObj | object | 标签列表的容器 |

1. 状态码说明：

| **Code** | **Reason** |
| --- | --- |
| 200- OK | 成功 |
| 400-BadRequest | 请求中的某些内容无效 |
| 403 - Forbidden | 请求没有权限 |
| 404- Not Found | 请求的资源不存在 |

1. 示例代码：

|  |
| --- |
| var NCOSS = require('ncoss')  // 永久秘钥创建客户端连接  var client= new NCOSS.Client({  endPoint: '172.18.232.192',  port: 8089,  path: '/v1',  accessKey: 'MTXZY45CMR7ET151L7OV',  secretKey: 'GR3Af1E7o1xosqy6rfBDNE1mmOSzMfjWLhccMMUN',  accountId: '7c9dfff2139b11edbc330391d2a979b2'  })  client.getBucketTagging('jssdk', function (err, tagsList){  if (err) {  return console.log(err)  }  console.log("Success", tagsList)  }) |

#### 删除桶标签

1. 方法：**deleteBucketTagging**
2. 请求参数说明：

| **Name** | **Require** | **Type** | **Description** |
| --- | --- | --- | --- |
| bucketName | True | string | 桶名称 |
| callback | True | function | 回调函数 |

1. 返回值说明：

| **Name** | **Type** | **Description** |
| --- | --- | --- |
| err | object | 请求如果有错误，会返回该错误对象 |

1. 状态码说明：

| **Code** | **Reason** |
| --- | --- |
| 200- OK | 成功 |
| 400-BadRequest | 请求中的某些内容无效 |
| 403 - Forbidden | 请求没有权限 |
| 404- Not Found | 请求的资源不存在 |

1. 示例代码：

|  |
| --- |
| var NCOSS = require('ncoss')  // 永久秘钥创建客户端连接  var client= new NCOSS.Client({  endPoint: '172.18.232.192',  port: 8089,  path: '/v1',  accessKey: 'MTXZY45CMR7ET151L7OV',  secretKey: 'GR3Af1E7o1xosqy6rfBDNE1mmOSzMfjWLhccMMUN',  accountId: '7c9dfff2139b11edbc330391d2a979b2'  })  client.removeBucketTagging('jssdk', function (err){  if (err) {  return console.log(err)  }  console.log("Success")  }) |

## 对象操作(object)

### 对象上传

#### 流方式上传

1. **putObject**

上传逻辑分为两种：

* 普通上传：当文件大小<10MB的时候，进行普通上传。
* 分片上传：当文件大小>=10MB的时候，进行分片上传。

设置分片时请详细查看[创建客户端](#_创建客户端)小节。

1. 请求参数说明：

| **Name** | **Require** | **Type** | **Description** |
| --- | --- | --- | --- |
| bucketName | True | string | 桶名称 |
| objectName | True | string | 指定对象名称 |
| stream | True | string | Buffer | 对象内容，可以为字符串或流 |
| size | False | number | 对象的存储类型  有效值：   * STANDARD： 标准存储 （热） * IA： 低频访问存储 (温) * ARCHIVE： 归档存储 （冷） |
| objectMetadata | False | ObjectMetadata | 对象元数据 |
| callback | True | function | 回调函数 |

1. 返回值说明:

| **Name** | **Type** | **Description** |
| --- | --- | --- |
| err | object | 请求如果有错误，会返回该错误对象 |
| eTag | String | 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   * 对于PutObject请求创建的Object，ETag值是其内容的MD5值 * 对于其他方式创建的Object，ETag值是其内容的UUID   说明 : ETag值可以用于检查Object内容是否发生变化。不建议使用ETag作为Object内容的MD5来校验数据完整性 |

1. 状态码说明:

| **Code** | **Reason** |
| --- | --- |
| 200 - OK | 成功 |
| 400 - Bad Request | 请求中的某些内容无效 |
| 403 - Forbidden | 请求没有权限 |
| 404 - Not Found | 请求的资源不存在 |

1. 示例代码：

|  |
| --- |
| var Fs = require('fs')  var NCOSS = require('ncoss');  const ObjectMetadata = require('../dist/main/ObjectMetadata');  // 永久秘钥创建客户端连接  var client= new NCOSS.Client({  endPoint: '172.18.232.192',  port: 8089,  path: '/v1',  accessKey: 'MTXZY45CMR7ET151L7OV',  secretKey: 'GR3Af1E7o1xosqy6rfBDNE1mmOSzMfjWLhccMMUN',  accountId: '7c9dfff2139b11edbc330391d2a979b2'  })  // Upload a stream  var file = 'localFilePath'  var fileStream = Fs.createReadStream(file);  // 设置元数据  var objectMetadata = new ObjectMetadata();  objectMetadata.addUserMetadata('example', 'value');  objectMetadata.addUserMetadata('example2', 'vlaue2');  var fileStat = Fs.stat(file, function (e, stat) {  if (e) {  return console.log(e)  }  client.putObject('jssdk', '2023/04/26/NCOSSFileSystemStore2.avi', fileStream, stat.size, objectMetadata, function (e,result) {  if (e) {  return console.log(e)  }  console.log(result);    console.log("Successfully uploaded the stream")  })  })  // Upload a buffer  var buf = new Buffer(10)  buf.fill('a')  client.putObject('bucket', '2023/04/18/bbb.txt', buf, 'application/octet-stream', function(e) {  if (e) {  return console.log(e)  }  console.log("Successfully uploaded the buffer")  })  // Upload a string  var str = "random string to be uploaded"  client.putObject('bucket', '2023/04/18/ccc.txt', str, 'text/plain', function(e) {  if (e) {  return console.log(e)  }  console.log("Successfully uploaded the string")  })  // Upload a Buffer without content-type (default: 'application/octet-stream')  client.putObject('bucket', '2023/04/18/ddd.txt', buf, function(e) {  if (e) {  return console.log(e)  }  console.log("Successfully uploaded the Buffer")  }) |

#### 本地文件上传

1. **fPutObject**

上传逻辑分为两种：

* 普通上传：当文件大小<10MB的时候，进行普通上传。
* 分片上传：当文件大小>=10MB的时候，进行分片上传。

设置分片时请详细查看[创建客户端](#_创建客户端)小节。

1. 请求参数说明：

| **Name** | **Require** | **Type** | **Description** |
| --- | --- | --- | --- |
| bucketName | True | string | 桶名称 |
| objectName | True | string | 指定对象名称 |
| filePath | True | string | 本地文件路径 |
| objectMetadata | False | ObjectMetadata | 对象元数据 |
| callback | True | function | 回调函数 |

1. 返回值说明:

| **Name** | **Type** | **Description** |
| --- | --- | --- |
| err | object | 请求如果有错误，会返回该错误对象 |
| result | object | 成功请求后的结果对象 |
| eTag | string | 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   * 对于PutObject请求创建的Object，ETag值是其内容的MD5值 * 对于其他方式创建的Object，ETag值是其内容的UUID   说明 : ETag值可以用于检查Object内容是否发生变化。不建议使用ETag作为Object内容的MD5来校验数据完整性 |

1. 状态码说明:

| **Code** | **Reason** |
| --- | --- |
| 200 - OK | 成功 |
| 400 - Bad Request | 请求中的某些内容无效 |
| 403 - Forbidden | 请求没有权限 |
| 404 - Not Found | 请求的资源不存在 |

1. 示例代码：

|  |
| --- |
| var Fs = require('fs')  var NCOSS = require('ncoss');  const ObjectMetadata = require('../dist/main/ObjectMetadata');  // 永久秘钥创建客户端连接  var client= new NCOSS.Client({  endPoint: '172.18.232.192',  port: 8089,  path: '/v1',  accessKey: 'MTXZY45CMR7ET151L7OV',  secretKey: 'GR3Af1E7o1xosqy6rfBDNE1mmOSzMfjWLhccMMUN',  accountId: '7c9dfff2139b11edbc330391d2a979b2'  })  // Upload a stream  var file = 'localFilePath'  var fileStream = Fs.createReadStream(file);  // 设置元数据  var objectMetadata = new ObjectMetadata();  objectMetadata.addUserMetadata('example', 'value');  objectMetadata.addUserMetadata('example2', 'vlaue2');  var fileStat = Fs.stat(file, function (e, stat) {  if (e) {  return console.log(e)  }  client.putObject('jssdk', '2023/04/26/NCOSSFileSystemStore2.avi', fileStream, stat.size, objectMetadata, function (e,result) {  if (e) {  return console.log(e)  }  console.log(result);    console.log("Successfully uploaded the stream")  })  })  // Upload a buffer  var buf = new Buffer(10)  buf.fill('a')  client.putObject('bucket', '2023/04/18/bbb.txt', buf, 'application/octet-stream', function(e) {  if (e) {  return console.log(e)  }  console.log("Successfully uploaded the buffer")  })  // Upload a string  var str = "random string to be uploaded"  client.putObject('bucket', '2023/04/18/ccc.txt', str, 'text/plain', function(e) {  if (e) {  return console.log(e)  }  console.log("Successfully uploaded the string")  })  // Upload a Buffer without content-type (default: 'application/octet-stream')  client.putObject('bucket', '2023/04/18/ddd.txt', buf, function(e) {  if (e) {  return console.log(e)  }  console.log("Successfully uploaded the Buffer")  }) |

### 下载对象

#### 下载文件

1. 方法：**fGetObject**

文件下载功能包含普通文件下载和大文件下载，普通文件可按实例下载，获取文件信息，下载到本地。

1. 请求参数说明：

| **Name** | **Require** | **Type** | **Description** |
| --- | --- | --- | --- |
| bucketName | True | string | 桶名称 |
| objectName | True | string | 对象名称 |
| localFile | True | string | 下载到本地的路径 |
| getOpts | False | object | 可选项对象 |
| versionId | False | string | 版本id |
| callback | True | function | 回调函数 |

1. 返回值说明

| **Name** | **Type** | **Description** |
| --- | --- | --- |
| err | object | 请求如果有错误，会返回该错误对象 |

1. 状态码说明：

| **Code** | **Reason** |
| --- | --- |
| 200 - OK | 成功 |
| 400 - Bad Request | 请求中的某些内容无效 |
| 403 - Forbidden | 请求没有权限 |
| 404- Not Found | 请求的资源不存在 |

1. 示例代码：

|  |
| --- |
| var NCOSS = require('ncoss')  // 永久秘钥创建客户端连接  var client= new NCOSS.Client({  endPoint: '172.18.232.192',  port: 8089,  path: '/v1',  accessKey: 'MTXZY45CMR7ET151L7OV',  secretKey: 'GR3Af1E7o1xosqy6rfBDNE1mmOSzMfjWLhccMMUN',  accountId: '7c9dfff2139b11edbc330391d2a979b2'  })  // Get a full object.  client.fGetObject('bucket', '2023/04/23/a.avi', 'E:\\比洛巴乔\\Desktop\\新建文件夹\\cc.avi', function(e) {  if (e) {  return console.log(e)  }  console.log('done')  })  //To get a specific version of an object  client.fGetObject('my-bucketname', 'my-objectname', '/tmp/objfile', {versionId:"03fd1247-90d9-4b71-a27e-209d484a234b"}, function(e) {  if (e) {  return console.log(e)  }  console.log('success')  }) |

#### 获取对象流

1. 方法：getObject

如果对象小于200MB，可以使用此方法；

如果对象大于200MB，不建议使用此方法，文件太大容易导致内存溢出。

1. 请求参数说明：

| **Name** | **Require** | **Type** | **Description** |
| --- | --- | --- | --- |
| bucketName | True | string | 桶名称 |
| key | True | string | 对象名称 |
| getObjectRequest | True | GetObjectRequest | 请求项对象 |
| range | False | array | 需要取的流的范围，默认整个文件  格式：[start,end]  取值范围：[start,end) |
| includeInputStream | True | boolean | 是否返回流，设置为true |
| callback | False | function | 回调函数 |

1. 返回值说明

| **Name** | **Type** | **Description** |
| --- | --- | --- |
| err | object | 请求如果有错误，会返回该错误对象 |
| result | object | 请求成功后返回的结果对象 |
| bucketName | string | 桶名 |
| objectName | string | 对象名称 |
| objectMetadata | ObjectMetadata | 对象元数据 |
| objectContent | Buffer | 对象流 |
| eTag | string | 对象标签值 |
| lastModified | string | 对象最新修改时间 |
| size | number | 对象大小 |
| mimeType | string | 对象类型 |

1. 状态码说明：

| **Code** | **Reason** |
| --- | --- |
| 200 - OK | 成功 |
| 400 - Bad Request | 请求中的某些内容无效 |
| 403 - Forbidden | 请求没有权限 |
| 404- Not Found | 请求的资源不存在 |

1. 示例代码：

|  |
| --- |
| var NCOSS = require('ncoss')  var ObjectDomain = require('../dist/main/ObjectDomain')  // 永久秘钥创建客户端连接  var client= new NCOSS.Client({  endPoint: '172.18.232.192',  port: 8089,  path: '/v1',  accessKey: 'MTXZY45CMR7ET151L7OV',  secretKey: 'GR3Af1E7o1xosqy6rfBDNE1mmOSzMfjWLhccMMUN',  accountId: '7c9dfff2139b11edbc330391d2a979b2'  })  var getObjectRequest = new ObjectDomain.GetObjectRequest({  includeInputStream:true  });  // Get a full object.  client.getObject('jssdk', '2023/04/20/NCOSSFileSystemStore.java', getObjectRequest, function (e, dataStream) {  let aa = Buffer.from(dataStream.getObjectContent()).toString('utf8');  console.log('数据流：：'+aa);    console.log('其他数据信息',dataStream);  }) |

### 复制对象

1. 方法：**copyObject**

复制对象是为对象存储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。

1. 请求参数说明：

| **Name** | **Require** | **Type** | **Description** |
| --- | --- | --- | --- |
| destBucket | True | string | 源桶 |
| destKey | True | string | 源对象 |
| srcBucket | True | string | 目标桶 |
| srcKey | True | string | 目标对象 |
| objectMetadata | False | ObjectMetadata | 新对象的元数据  属性值：userObjectMetadata，metadata |
| userObjectMetaata | True | object | 自定义元数据对象 |
| metadata | True | object | 对象存储系统特定含义元数据，例如存储类型，content-type,content-length等头属性 |
| callback | True | function | 回调函数 |

1. 返回值说明

| **Name** | **Type** | **Description** |
| --- | --- | --- |
| err | object | 请求如果有错误，会返回该错误对象 |
| result | object | 请求成功结果对象 |
| eTag | string | 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   * 对于简单上传请求创建的Object，ETag值是其内容的MD5值 * 对于其他方式创建的Object，ETag值是其内容的UUID   说明 : ETag值可以用于检查Object内容是否发生变化。不建议使用ETag作为Object内容的MD5来校验数据完整性 |
| lastModified | string | 对象最后一次修改时间 |

1. 状态码说明：

| **Code** | **Reason** |
| --- | --- |
| 200- OK | 成功 |
| 400-BadRequest | 请求中的某些内容无效 |
| 403 - Forbidden | 请求没有权限 |
| 404- Not Found | 请求的资源不存在 |

1. 示例代码：

|  |
| --- |
| vvar NCOSS = require('ncoss')  var ObjectMetadata = require('../dist/main/ObjectMetadata')  // 永久秘钥创建客户端连接  var client= new NCOSS.Client({  endPoint: '172.18.232.192',  port: 8089,  path: '/v1',  accessKey: 'MTXZY45CMR7ET151L7OV',  secretKey: 'GR3Af1E7o1xosqy6rfBDNE1mmOSzMfjWLhccMMUN',  accountId: '7c9dfff2139b11edbc330391d2a979b2'  })  // 创建对象元数据  var objectMetadata = new ObjectMetadata({});  objectMetadata.setObjectStorageClass('STANDARD'); //设置存储类型  objectMetadata.addUserMetadata('meta1', 'value1'); // 设置自定义元数据  objectMetadata.addUserMetadata('meta2', 'value2');  objectMetadata.addUserMetadata('meta3', 'value3');  objectMetadata.setObjectDirective('REPLACE\_NEW'); // 设置复制类型  // 调用函数进行复制  client.copyObject('bucket', '2023/04/21/dest.txt', 'bucket', '2023/04/18/ccc.txt', objectMetadata, function (e, data) {  if (e) {  return console.log(e)  }  console.log("Successfully copied the object:")  console.log("etag = " + data.etag + ", lastModified = " + data.lastModified)  }) |

### 解冻对象

1. 方法：**restoreObject**

* 如果要获取归档存储对象(即storageClass为ARCHIVE)的内容 （下载），需要先将对象解冻恢复，然后再执行下载数据的操作。
* 对象解冻恢复后，会产生一个标准存储类型的对象副本，也就是 说会同时存在标准存储类型的对象副本和归档存储类型的对象，在取 回对象的保存时间到期后标准存储类型的对象副本会自动删除。

由于对归档对象的解冻过程是异步操作，所以，返回码解释如下：

* 首次对处于冷冻状态的归档类型Object提交解冻请求后，对象 将处于解冻中状态，会返回202 Accepted，代表已接受解冻请求；
* 对处于解冻中状态的归档类型Object再次提交解冻请求，将返 回409 Conflict，代表对象正解冻处理中；
* 对处于解冻状态的归档类型Object提交解冻请求，将返回200 OK,代表解冻完成。

1. 请求参数说明：

| **Name** | **Require** | **Type** | **Description** |
| --- | --- | --- | --- |
| bucketName | True | string | 桶名称 |
| key | True | string | 需要解冻的对象名称 |
| days | True | number | 临时对象副本保存时间，时间单位:天，以对象的最后更新时间为准 |
| callback | True | function | 回调函数 |

1. 返回值说明：

| **Name** | **Type** | **Description** |
| --- | --- | --- |
| err | object | 请求如果有错误，会返回该错误对象 |

1. 状态码说明：

| **Code** | **Reason** |
| --- | --- |
| 200 - OK | 成功 |
| 202 - Accepted | 解冻请求已被接受执行 |
| 400-BadRequest | 请求中的某些内容无效 |
| 403 - Forbidden | 请求没有权限 |
| 404- Not Found | 请求的资源不存在 |
| 409- Conflict | 请求的对象正在解冻中 |

1. 示例代码：

|  |
| --- |
| var NCOSS = require('ncoss')  // 永久秘钥创建客户端连接  var client= new NCOSS.Client({  endPoint: '172.18.232.192',  port: 8089,  path: '/v1',  accessKey: 'MTXZY45CMR7ET151L7OV',  secretKey: 'GR3Af1E7o1xosqy6rfBDNE1mmOSzMfjWLhccMMUN',  accountId: '7c9dfff2139b11edbc330391d2a979b2'  })  //  client.restoreObject('jssdk', '2023/04/20/c.txt',1, function (err){  if (err) {  return console.log(err)  }  console.log("Success")  }) |

### 查询对象信息

1. 方法：**getObject**
2. 请求参数说明：

| **Name** | **Require** | **Type** | **Description** |
| --- | --- | --- | --- |
| bucketName | True | string | 桶名称 |
| key | True | string | 对象名称 |
| callback | True | function | 回调函数 |

1. 返回值说明：

| **Name** | **Type** | **Description** |
| --- | --- | --- |
| err | object | 请求如果有错误，会返回该错误对象 |
| result | object | 成功请求后返回的对象 |
| size | number | 对象大小 |
| metaData | object | 对象元数据 |
| lastModified | string | 对象最新修改时间  例如：2023-04-18T08:22:56.000Z |
| versionId | string | 对象版本id |
| etag | string | 对象标签 |

1. 状态码说明：

| **Code** | **Reason** |
| --- | --- |
| 200 - OK | 成功 |
| 400 - Bad Request | 请求中的某些内容无效 |
| 403- Forbidden | 请求没有权限 |
| 404 -Not Found | 请求的资源不存在 |

1. 示例代码：

|  |
| --- |
| var NCOSS = require('ncoss')  var ObjectDomain = require('../dist/main/ObjectDomain')  // 永久秘钥创建客户端连接  var client= new NCOSS.Client({  endPoint: '172.18.232.192',  port: 8089,  path: '/v1',  accessKey: 'MTXZY45CMR7ET151L7OV',  secretKey: 'GR3Af1E7o1xosqy6rfBDNE1mmOSzMfjWLhccMMUN',  accountId: '7c9dfff2139b11edbc330391d2a979b2'  })  client.getObject("bucket", "2023/04/18/ccc.txt", function (e, dataStream) {  if(e) console.log(e);    console.log('最终结果',dataStream);    }) |

### 删除对象

1. 方法：**deleteObject**
2. 请求参数说明：

| **Name** | **Require** | **Type** | **Description** |
| --- | --- | --- | --- |
| bucketName | True | string | 桶名称 |
| key | True | string | 对象名称 |
| removeOpts | False | object | 可选项对象 |
| versionId | False | string | 指定删除对象的版本号 |
| callback | True | function | 回调函数 |

1. 返回值说明：

| **Name** | **Type** | **Description** |
| --- | --- | --- |
| err | object | 请求如果有错误，会返回该错误对象 |

1. 状态码说明：

| **Code** | **Reason** |
| --- | --- |
| 200 - OK | 成功 |
| 400 - Bad Request | 请求中的某些内容无效 |
| 403- Forbidden | 请求没有权限 |
| 404- Not Found | 请求的资源不存在 |

1. 示例代码：

|  |
| --- |
| var NCOSS = require('ncoss')  // 永久秘钥创建客户端连接  var client= new NCOSS.Client({  endPoint: '172.18.232.192',  port: 8089,  path: '/v1',  accessKey: 'MTXZY45CMR7ET151L7OV',  secretKey: 'GR3Af1E7o1xosqy6rfBDNE1mmOSzMfjWLhccMMUN',  accountId: '7c9dfff2139b11edbc330391d2a979b2'  })  // Remove an object name my-objectname.  client.removeObject('jssdk', '2023/04/20/c.txt', function(e) {  if (e) {  return console.log(e)  }  console.log("Success")  })  // // Remove an object with name 'my-objectname' and a versionId.  // client.removeObject('my-bucketname', 'my-objectname', {versionId:"my-versionId"}, function(e) {  // if (e) {  // return console.log(e)  // }  // console.log("Success")  // }) |

### 对象ACL

#### 设置对象ACL

1. 方法：**setObjectAcl**

HOS支持对对象操作进行权限控制。默认情况下，只有对象的拥有者才有该对象的读写权限。此操作属于覆盖操作，新的配置将覆盖对象已存在的ACL配置。

1. 请求参数说明：

| **Name** | **Require** | **Type** | **Description** |
| --- | --- | --- | --- |
| bucketName | True | string | 桶名称 |
| objectName | True | string | 对象名称 |
| ACL | False | object | ACL对象 |
| owner | True | string | 桶的拥有者信息 |
| accessControlList | True | array | 权限访问控制容器 |
| accessControl | True | object | 权限控制对象 |
| Permission | True | string | 授权人权限 READ | WRITE | READ\_ACP | WRITE\_ACP | FULL\_CONTROL，详细浏览HOS支持桶的ACL权限元素含义表 即Permission参数 |
| GranteeType | True | string | 分别为CanonicalUserGrantee和GroupGrantee   * 如果为CanonicalUserGrantee，需要指定好userId * 如果为GroupGrantee，一类是AllUsers，代表所有人包括匿名用户。另一个是AuthenticatedUsers经过身份认证的所有用户 |
| ID | True | string | 用户ID，只有GranteeType参数为  CanonicalUserGrantee时，此参数是必填项 |
| URI | True | string | 只有GranteeType参数为GroupGrantee时，此参数是必填项  有效值：  **1**. <http://www.heredata.com/groups/global/AllUsers> 代表所有人包括匿名用户；  **2**. <http://www.heredata.com/groups/global/AuthenticatedUsers> 代表经过身份认证的所有用户。 |
| ACLString | False | string | **与ACL参数冲突，需要您自行组装ACL字符串，该字符串符合ACL语法。详细请浏览下面ACL语法** |
| callback | True | function | 回调函数 |

HOS支持对象的ACL权限元素含义表 即**Permission参数**

|  |  |  |
| --- | --- | --- |
| **权限** | **描述** | **对应动作** |
| READ | 可以获取该对象内容和元数据。 | GetObject、HeadObject |
| WRITE | 此权限在对象上不适用。 | 无 |
| READ\_ACP | 可以获取对应的对象的权限控制列表（ACL）。 | GetObjectAcl |
| WRITE\_ACP | 更新对应对象的权限控制列表（ACL）。 | PutObjectAcl |
| FULL\_CONTROL | 拥有READ、WRITE、READ\_ACP和WRITE\_ACP的权限。 | 包含以上所有动作。 |

ACL语法如下

|  |
| --- |
| <AccessControlPolicy>  <Owner>  <ID>c7a4c02ee77b11eaa1cc01cf93dcddb3</ID>  </Owner>  <AccessControlList>  <Grant>  <Grantee xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="CanonicalUser">  <ID>user\_id<ID>  </Grantee>  <Permission>permission</Permission>  </Grant>  <Grant>  <Grantee xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="Group">  <URI>http://www.heredata.com/groups/global/AllUsers</URI> </Grantee>  <Permission>READ</Permission>  </Grant>  </AccessControlList>  </AccessControlPolicy> |

1. 返回值说明：

| **Name** | **Type** | **Description** |
| --- | --- | --- |
| err | object | 请求如果有错误，会返回该错误对象 |

1. 状态码说明：

| **Code** | **Reason** |
| --- | --- |
| 200- OK | 成功 |
| 400-BadRequest | 请求中的某些内容无效 |
| 403 - Forbidden | 请求没有权限 |
| 404 - Not Found | 请求的资源不存在 |

1. 示例代码：

|  |
| --- |
| var NCOSS = require('ncoss')  // 永久秘钥创建客户端连接  var client= new NCOSS.Client({  endPoint: '172.18.232.192',  port: 8089,  path: '/v1',  accessKey: 'MTXZY45CMR7ET151L7OV',  secretKey: 'GR3Af1E7o1xosqy6rfBDNE1mmOSzMfjWLhccMMUN',  accountId: '7c9dfff2139b11edbc330391d2a979b2'  })  var ACL = {}  // 设置拥有者信息  ACL['OwnerID'] = '7c9dfff2139b11edbc330391d2a979b2'  // 设置权限控制集合  const accessControlList = [];  // 第一种类型权限控制  var accessControl1 = {};  // 授权人权限 READ | WRITE | READ\_ACP | WRITE\_ACP | FULL\_CONTROL  accessControl1['Permission'] = 'READ'  // 设置授权信信息容器，只有两个值可选  // 1:CanonicalUser:规范用户需指定用户ID，与 元素 ID 一起使用  // 2:Group: 范围授权需指定元素URI，有效值为：  // http://www.heredata.com/groups/global/AllUsers 代表所有人包括匿名用户  // http://www.heredata.com/groups/global/AuthenticatedUsers 代表经过身份认证的所有用户。  accessControl1['GranteeType'] = 'CanonicalUser'  accessControl1['ID'] = '7c9dfff2139b11edbc330391d2a979b2'  // 第二种类型权限控制  var accessControl2 = {};  // 授权人权限 READ | WRITE | READ\_ACP | WRITE\_ACP | FULL\_CONTROL  accessControl2['Permission'] = 'WRITE'  accessControl2['GranteeType'] = 'Group'  accessControl2['URI'] = 'http://www.heredata.com/groups/global/AllUsers'  // 将权限控制push到集合中  accessControlList.push(accessControl1)  accessControlList.push(accessControl2)  ACL['accessControlList'] = accessControlList  client.setObjectACL('jssdk', '2023/04/20/b.txt', ACL, function (err) {  if (err) {  return console.log(err)  }  console.log("Success")  })  // const ACLString = '<AccessControlPolicy><Owner><ID>7c9dfff2139b11edbc330391d2a979b2</ID></Owner><AccessControlList><Grant><Grantee xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="CanonicalUser"><ID>7c9dfff2139b11edbc330391d2a979b2</ID></Grantee><Permission>READ</Permission></Grant><Grant><Grantee xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="Group"><URI>http://www.heredata.com/groups/global/AllUsers</URI></Grantee><Permission>WRITE</Permission></Grant></AccessControlList></AccessControlPolicy>';  // client.setBucketACL('bucket', ACLString, function (err) {  // if (err) {  // return console.log(err)  // }  // console.log("Success")  // }) |

#### 查询对象ACL

1. 方法：**getObjectAcl**
2. 请求参数说明：

| **Name** | **Require** | **Type** | **Description** |
| --- | --- | --- | --- |
| bucketName | True | string | 桶名称 |
| key | True | string | 对象名称 |
| callback | True | function | 回调函数 |

1. 返回值说明：

| **Name** | **Type** | **Description** |
| --- | --- | --- |
| err | object | 请求如果有错误，会返回该错误对象 |
| ACL | object | 策略字符串 |
| Owner | object | 桶的拥有者信息 |
| ID | string | 需指定桶的所属账户ID |
| AccessControlList | array | 权限控制列表容器 |
| Grant | array | 权限控制信息容器 |
| Grantee | object | 授权信信息容器，注意前缀格式，xsi:type 有效值为CanonicalUser、  Group：   * **CanonicalUser**：规范用户需指定用户ID，与 元素 ID 一起使用； * **Group**：范围授权需指定URI，有效值为：   **1**. <http://www.heredata.com/groups/global/AllUsers> 代表所有人包括匿名用户；  **2**. <http://www.heredata.com/groups/global/AuthenticatedUsers> 代表经过身份认证的所有用户。 |
| URI | string | 授权人类型为Group 时，需指定授权范围。 |
| ID | string | 授权人类型为CanonicalUser 时，需指定用户ID。 |
| Permission | string | 授权人权限，有效值为：READ | WRITE | READ\_ACP | WRITE\_ACP | FULL\_CONTROL |

1. 状态码说明：

| **Code** | **Reason** |
| --- | --- |
| 200 - OK | 成功 |
| 400 - Bad Request | 请求中的某些内容无效 |
| 403 - Forbidden | 请求没有权限 |
| 404- Not Found | 请求的资源不存在 |

1. 示例代码：

|  |
| --- |
| var NCOSS = require('ncoss')  // 永久秘钥创建客户端连接  var client= new NCOSS.Client({  endPoint: '172.18.232.192',  port: 8089,  path: '/v1',  accessKey: 'MTXZY45CMR7ET151L7OV',  secretKey: 'GR3Af1E7o1xosqy6rfBDNE1mmOSzMfjWLhccMMUN',  accountId: '7c9dfff2139b11edbc330391d2a979b2'  })  client.getObjectACL('jssdk', '2023/04/20/b.txt', function (err,res){  if (err) {  return console.log(err)  }  console.log(res)  console.log("Success")  }) |

### 对象标签

#### 设置对象标签

1. 方法：**setObjectTagging**

给对象添加标签配置，如果对象已存在标签配置则覆盖之前的标签配置。此操作属于覆盖操作，标签最多支持40对。

1. 请求参数说明：

| **Name** | **Require** | **Type** | **Description** |
| --- | --- | --- | --- |
| bucketName | True | string | 桶名称 |
| key | True | string | 对象名称 |
| removeOpts | False | object | 可选项对象 |
| versionId | False | string | 指定删除对象的版本号 |
| tagSet | True | object | 标签列表的容器 |
| callback | True | function | 回调函数 |

1. 返回值说明：

| **Name** | **Type** | **Description** |
| --- | --- | --- |
| err | object | 请求如果有错误，会返回该错误对象 |

1. 状态码说明：

| **Code** | **Reason** |
| --- | --- |
| 200 - OK | 成功 |
| 400 - Bad Request | 请求中的某些内容无效 |
| 403 - Forbidden | 请求没有权限 |
| 404- Not Found | 请求的资源不存在 |

1. 示例代码：

|  |
| --- |
| var NCOSS = require('ncoss')  // 永久秘钥创建客户端连接  var client= new NCOSS.Client({  endPoint: '172.18.232.192',  port: 8089,  path: '/v1',  accessKey: 'MTXZY45CMR7ET151L7OV',  secretKey: 'GR3Af1E7o1xosqy6rfBDNE1mmOSzMfjWLhccMMUN',  accountId: '7c9dfff2139b11edbc330391d2a979b2'  })  var tagsMap = {"tagkey":"tagvalue"}  client.setObjectTagging('jssdk', "2023/04/20/b.avi", tagsMap, function (err){  if (err) {  return console.log(err)  }  console.log("Success")  })  // //Put tags on a version of an object  // client.setObjectTagging('bucketname', "object-name", tagsMap, { versionId: "my-version-id" }, function (err){  // if (err) {  // return console.log(err)  // }  // console.log("Success")  // }) |

#### 查询对象标签

1. 方法：**getObjectTagging**
2. 请求参数说明：

| **Name** | **Require** | **Type** | **Description** |
| --- | --- | --- | --- |
| bucketName | True | string | 桶名称 |
| key | True | string | 对象名称 |
| callback | True | function | 回调函数 |

1. 返回值说明：

| **Name** | **Type** | **Description** |
| --- | --- | --- |
| err | object | 请求如果有错误，会返回该错误对象 |
| tagsObj | object | 标签列表的容器 |

1. 状态码说明：

| **Code** | **Reason** |
| --- | --- |
| 200 - OK | 成功 |
| 400 - Bad Request | 请求中的某些内容无效 |
| 403- Forbidden | 请求没有权限 |
| 404- Not Found | 请求的资源不存在 |

1. 示例代码：

|  |
| --- |
| var NCOSS = require('ncoss')  // 永久秘钥创建客户端连接  var client= new NCOSS.Client({  endPoint: '172.18.232.192',  port: 8089,  path: '/v1',  accessKey: 'MTXZY45CMR7ET151L7OV',  secretKey: 'GR3Af1E7o1xosqy6rfBDNE1mmOSzMfjWLhccMMUN',  accountId: '7c9dfff2139b11edbc330391d2a979b2'  })  client.getObjectTagging('jssdk', '2023/04/20/b.avi', function (err, tagsList){  if (err) {  return console.log(err)  }  console.log("Success",tagsList)  })  // //Get tags on a version of an object.  // client.getObjectTagging('bucketname', "objectName", { versionId: "" }, function (err, tagsList){  // if (err) {  // return console.log(err)  // }  // console.log("Success",tagsList)  // }) |

#### 删除对象标签

1. 方法：**deleteObjectTagging**
2. 请求参数说明：

| **Name** | **Require** | **Type** | **Description** |
| --- | --- | --- | --- |
| bucketName | True | string | 桶名称 |
| key | True | string | 对象名称 |

1. 返回值说明：

| **Name** | **Type** | **Description** |
| --- | --- | --- |
| err | object | 请求如果有错误，会返回该错误对象 |

1. 状态码说明：

| **Code** | **Reason** |
| --- | --- |
| 200 - OK | 成功 |
| 400 - Bad Request | 请求中的某些内容无效 |
| 403- Forbidden | 请求没有权限 |
| 404- Not Found | 请求的资源不存在 |

1. 示例代码：

|  |
| --- |
| var NCOSS = require('ncoss')  // 永久秘钥创建客户端连接  var client= new NCOSS.Client({  endPoint: '172.18.232.192',  port: 8089,  path: '/v1',  accessKey: 'MTXZY45CMR7ET151L7OV',  secretKey: 'GR3Af1E7o1xosqy6rfBDNE1mmOSzMfjWLhccMMUN',  accountId: '7c9dfff2139b11edbc330391d2a979b2'  })  client.removeObjectTagging("bucket", "2023/04/18/ccc.txt", function (err){  if (err) {  return console.log(err)  }  console.log("Success")  })  // //remove tags on a version of an object  // client.removeObjectTagging('bucketname', 'object-name', { versionId: "my-object-version-id" }, function (err){  // if (err) {  // return console.log(err)  // }  // console.log("Success")  // }) |