**HOS-NODEJS-SDK使用手册**

NCOSS-V4.\*

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** | 升级 |  | **1.0.9** | **2.0.3** | 武泽昭 |  | 2023.11.15 |  |
| **3** |  |  |  |  |  |  |  |  |

**目录**

[HOS-NODEJS-SDK使用手册 1](#_Toc3562)

[1 概述 6](#_Toc12618)

[1.1 背景 6](#_Toc9277)

[1.2 适用范围 6](#_Toc13727)

[1.3 术语定义 6](#_Toc32376)

[1.4 参考资料 7](#_Toc14818)

[2 使用配置 8](#_Toc24311)

[2.1 基础环境 8](#_Toc24111)

[2.2 配置使用 8](#_Toc30668)

[2.2.1 服务加载 8](#_Toc32022)

[2.2.1.1 局域网加载 8](#_Toc28422)

[2.2.1.2 安装加载 9](#_Toc27450)

[2.2.2 使用实例 9](#_Toc14917)

[3 接口使用 11](#_Toc5405)

[3.1 创建客户端 11](#_Toc13277)

[3.2 账户类(account) 12](#_Toc29750)

[3.2.1 查询账户详情 12](#_Toc7696)

[3.2.2 查询当前账户下桶的列表 13](#_Toc23999)

[3.3 桶操作(bucket) 14](#_Toc11567)

[3.3.1 创建桶 14](#_Toc217)

[3.3.2 查询桶详细信息 15](#_Toc14236)

[3.3.3 删除桶 17](#_Toc22271)

[3.3.4 列举桶内对象 18](#_Toc25348)

[3.3.5 列举出桶内的版本对象 20](#_Toc8411)

[3.3.6 桶策略 22](#_Toc8270)

[3.3.6.1 桶策略语法 22](#_Toc6908)

[3.3.6.2 设置桶策略 25](#_Toc5133)

[3.3.6.3 查询桶策略 27](#_Toc12541)

[3.3.7 桶版本管理 29](#_Toc14199)

[3.3.7.1 开启/关闭桶的版本状态 29](#_Toc27417)

[3.3.7.2 查询桶的版本控制状态 31](#_Toc7732)

[3.4 对象操作(object) 32](#_Toc29002)

[3.4.1 对象上传 32](#_Toc28547)

[3.4.1.1 流方式上传 32](#_Toc6236)

[3.4.1.2 本地文件上传 35](#_Toc21880)

[3.4.2 下载对象 38](#_Toc20637)

[3.4.2.1 下载文件 38](#_Toc31603)

[3.4.2.2 获取对象流 40](#_Toc19220)

[3.4.3 查询对象信息 42](#_Toc15608)

[3.4.4 删除对象 43](#_Toc4451)

[3.4.5 对象标签 45](#_Toc28813)

[3.4.5.1 设置对象标签 45](#_Toc28095)

[3.4.5.2 查询对象标签 47](#_Toc22093)

[3.4.5.3 删除对象标签 48](#_Toc11170)

# 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：2.0.3。

对象存储系统版本：4.\*

## 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 |
| 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V4.0.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：

|  |
| --- |
| var NCOSS = require('../dist/main/ncoss')  /\*\*  \* 导出ncoss-4.\*的客户端连接，底层通过path是"/HOSv1"还是"/v1"来决定请求对应的ncoss服务端，使用临时秘钥  \*/  exports.s3ClientV4 = new NCOSS.Client({  endPoint: '172.18.232.37',  port: 8089,  path: '/V1',  username: 'test\_user1',  password: 'TEST#ps@857',  scopeName: 'test\_pro1',  uaasURL: 'http://172.18.232.192:6020/v3/auth/tokens',  useSSL: false  })  /\*\*  \* 导出ncoss-4.\*的客户端连接，底层通过path是"/HOSv1"还是"/v1"来决定请求对应的ncoss服务端，使用永久秘钥  \*/  exports.s3ClientV4Permanent = new NCOSS.Client({  endPoint: '172.18.232.37',  port: 8089,  path: '/V1',  accessKey: '95EB1FFUV3ARTEMEGE9B',  secretKey: 'BXURiYpZzx2OakZBt8HKexpF4kpDxPo4F9f9JD3A',  useSSL: false  }) |

# 接口使用

## 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 |

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',  useSSL:false  })  // 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)  }) |

### 查询当前账户下桶的列表

1. 方法：**listBuckets**

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 |

1. 返回参数：

| **Name** | **Type** | **Description** |
| --- | --- | --- |
| name | string | 桶名称 |
| creationDate | Date | 创建日期 |

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'  })  ncossClient.listBuckets(function(e, buckets) {  if (e) return console.log(e)  console.log('buckets :', buckets)  }) |

## 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',  useSSL:false  })  // 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',  useSSL:false  })  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',  useSSL:false  })  // 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：**listObjectsV2**

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 |
| startAfter | False | string | 查询出的对象大于等于该值 |

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',  useSSL:false  })  // 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',  useSSL:false  })  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',  useSSL:false  })  // 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',  useSSL:false  })  // Retrieves the bucket policy and logs it to the console.  client.getBucketPolicy('jssdk', (err, policy) => {  if (err) throw err  console.log(`Bucket policy: ${policy}`)  }) |

### 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',  useSSL:false  })  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',  useSSL:false  })  client.getBucketVersioning('jssdk', function (err,res){  if (err) {  return console.log(err)  }  console.log(res)  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',  useSSL:false  })  // 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',  useSSL:false  })  // 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',  useSSL:false  })  // 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',  useSSL:false  })  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);  }) |

### 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',  useSSL:false  })  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',  useSSL:false  })  // 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',  useSSL:false  })  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',  useSSL:false  })  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',  useSSL:false  })  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")  // }) |