对象存储服务

Node.js SDK 开发指南

文档版本	02
发布日期	2023-08-26

华为技术有限公司

商标声明


和其他华为商标均为华为技术有限公司的商标。
本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

您购买的产品、服务或特性等应受华为公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，华为公司对本文档内容不做任何明示或暗示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

安全声明

漏洞处理流程

华为公司对产品漏洞管理的规定以“漏洞处理流程”为准，该流程的详细内容请参见如下网址：
https://www.huawei.com/cn/psirt/vul-response-process
如企业客户须获取漏洞信息，请参见如下网址：
https://securitybulletin.huawei.com/enterprise/cn/security-advisory

目 录

1 SDK 下载

下载地址

- OBS NodeJS SDK最新版本源码：**最新源码下载**
- OBS NodeJS SDK历史版本下载地址：**历史版本下载**

API 文档

- 接口参考文档地址： **SDK API 文档**

兼容性

- 推荐使用的Node.js版本：Node 0.12.x，Node4.x，Node6.x，Node8.x，Node10.x。
- 接口函数：与旧版本（2.1.x）**不完全兼容，接口变化如下表**：

接口函数	变化说明
ObsClient.listBuckets	响应结果中InterfaceResult.Buckets字段变为数组，去掉InterfaceResult.Buckets.Bucket字段。
ObsClient.setBucketAcl	请求参数中Grants字段变为数组，去掉Grants.Grant字段。
ObsClient.getBucketAcl	响应结果中InterfaceResult.Grants字段变为数组，去掉InterfaceResult.Grants.Grant字段。
ObsClient.setObjectAcl	请求参数中Grants字段变为数组，去掉Grants.Grant字段。
ObsClient.getObjectAcl	响应结果中InterfaceResult.Grants字段变为数组，去掉InterfaceResult.Grants.Grant字段。
ObsClient.setBucketLogging	请求参数中LoggingEnabled.TargetGrants字段变为数组，去掉LoggingEnabled.TargetGrants.Grant字段。

接口函数	变化说明
ObsClient.getBucketLogging	响应结果中InterfaceResult.LoggingEnabled.TargetGrants字段变为数组，去掉InterfaceResult.LoggingEnabled.TargetGrants.Grant字段。
ObsClient.setBucketWebsite	请求参数中RoutingRules字段变为数组，去掉RoutingRules.RoutingRule字段。
ObsClient.getBucketWebsite	响应结果中InterfaceResult.RoutingRules字段变为数组，去掉InterfaceResult.RoutingRules.RoutingRule字段。
ObsClient.setBucketCors	请求参数中CorsRule字段改名为CorsRules。
ObsClient.getBucketCors	响应结果中InterfaceResult.CorsRule字段改名为InterfaceResult.CorsRules。
ObsClient.setBucketTagging	请求参数中TagSet字段改名为Tags并变为数组，去掉TagSet.Tag字段。
ObsClient.getBucketTagging	响应结果中InterfaceResult.TagSet字段改名为Tags并变为数组，去掉InterfaceResult.TagSet.Tag字段。

2 示例程序

OBS Node.js SDK提供了丰富的示例程序，方便用户参考或直接使用。您可以从OBS Node.js SDK开发包中获取示例程序，如eSDK_Storage_OBS_<VersionId>_Node.js.zip，解压后eSDK_Storage_OBS_<VersionId>_Node.js/examples为示例程序。您也可以从下面表格中直接下载示例程序。

示例包括以下内容：

示例代码	说明
bucket-operations-sample	展示了桶相关接口的用法
object-operations-sample	展示了对象相关接口的用法
download-sample	展示了下载对象的用法
create-folder-sample	展示了创建文件夹的用法
delete-objects-sample	展示了批量删除对象的用法
list-objects-sample	展示了列举对象的用法
list-versions-sample	展示了列举多版本对象的用法
list-objects-infolder-sample	展示了列举文件夹内对象的用法
object-meta-sample	展示了自定义对象元数据的用法
simple-multipart-upload-sample	展示了分段上传的基本用法
restore-object-sample	展示了下载归档存储对象的用法
concurrent-copypart-sample	展示了分段并发复制大对象的用法
concurrent-download-object-sample	展示了分段并发下载大对象的用法
concurrent-uploadpart-sample	展示了分段并发上传大对象的用法
post-object-sample	展示了表单上传对象的用法
temporary-signature-sample	展示了使用URL进行授权访问的用法

3 快速入门

3.1 使用前需知

须知

开发过程中，您有任何问题可以在github上**提交issue**。**接口参考文档**详细介绍了每个接口的参数和使用方法。

- 请确认您已经熟悉OBS的基本概念，如**桶（Bucket）**、**对象（Object）**、**访问密钥（AK和SK）**等。
- 您可以先参考**OBS客户端通用示例**，了解OBS Node.js SDK接口调用的通用方式。
- OBS客户端支持回调函数和Promise对象两种方式返回调用结果。
- 当前各区域特性开放不一致，部分特性只在部分区域开放，使用过程中如果接口HTTP状态码为405，请确认该区域是否支持该功能特性。

3.2 OBS 服务环境搭建

须知

开发过程中，您有任何问题可以在github上**提交issue**。**接口参考文档**详细介绍了每个接口的参数和使用方法。

步骤1 注册云服务账号

使用OBS之前必须要有一个云服务账号。

1. 打开浏览器。
2. 登录**公有云网站**。
3. 在页面右上角单击“注册”。
4. 按需填写注册信息并单击“同意协议并注册”。

步骤2 开通OBS服务

使用OBS服务之前必须先充值，才能正常使用OBS服务。

1. 登录**管理控制台**。
2. 单击页面右上角的“费用和成本”进入费用中心页面。
3. 选择“资金管理 > 充值”，系统自动跳转到充值窗口。
4. 根据界面提示信息，对账户进行充值。
5. 充值成功后，关闭充值窗口，返回管理控制台首页。
6. 在服务列表中选择“对象存储服务 OBS”，开通并进入OBS管理控制台。

步骤3 创建访问密钥

OBS通过用户账户中的AK和SK进行签名验证，确保通过授权的账户才能访问指定的OBS资源。以下是对AK和SK的解释说明：

- AK：Access Key ID，接入键标识，用户在对象存储服务系统中的接入键标识，一个接入键标识唯一对应一个用户，一个用户可以同时拥有多个接入键标识。对象存储服务系统通过接入键标识识别访问系统的用户。
- SK：Secret Access Key，安全接入键，用户在对象存储服务系统中的安全接入键，是用户访问对象存储服务系统的密钥，用户根据安全接入键和请求头域生成鉴权信息。安全接入键和接入键标识一一对应。

访问密钥分永久访问密钥（AK/SK）和临时访问密钥（AK/SK和SecurityToken）两种。每个用户最多可创建两个有效的永久访问密钥。临时访问密钥只在设置的有效期内能够访问OBS，过期后需要重新获取。出于安全性考虑，建议您使用临时访问密钥访问OBS，或使用永久访问密钥访问OBS时，定期更新您的访问密钥（AK/SK）。两种密钥的获取方式如下所示。

- 永久访问密钥：
  a. 登录**管理控制台**。
  b. 单击页面右上角的用户名，并选择“我的凭证”。
  c. 在“我的凭证”页面，单击左侧导航栏的“访问密钥”。
  d. 在“访问密钥”页面，单击“新增访问密钥”。

说明

每个用户最多可创建两个有效的访问密钥。

  e. 在弹出的“新增访问密钥”对话框中，输入描述内容（建议），单击“确定”。

f. （可选）在弹出的“身份验证”对话框中，选择合适的验证方式进行验证，单击“确定”。

g. 在弹出的“创建成功”提示框中，单击“立即下载”后，密钥会直接保存到浏览器默认的下载文件夹中。

h. 打开下载下来的“credentials.csv”文件既可获取到访问密钥（AK和SK）。

说明

- 在密钥文件中，Access Key ID列对应的值即AK，Secret Access Key列对应的值即SK。
- 为防止访问密钥泄露，建议您将其保存到安全的位置。如果用户在此提示框中单击“取消”，则不会下载密钥，后续也将无法重新下载。如果需要使用访问密钥，可以重新创建新的访问密钥。

- 临时访问密钥：

  临时AK/SK和SecurityToken是系统颁发给用户的临时访问令牌，通过接口设置有效期，范围为15分钟至24小时，过期后需要重新获取。临时AK/SK和SecurityToken遵循权限最小化原则。使用临时AK/SK鉴权时，临时AK/SK和SecurityToken必须同时使用。

获取临时访问密钥的接口请参考**获取临时AK/SK和securitytoken**。

须知

OBS属于全局级服务，所以在获取临时访问密钥时，需要设置Token的使用范围取值为domain，表示获取的Token可以作用于全局服务，全局服务不区分项目或者区域。

----结束

3.3 开发环境准备

须知

开发过程中，您有任何问题可以在github上**提交issue**。**接口参考文档**详细介绍了每个接口的参数和使用方法。

- 从**Node.js官网**下载并安装推荐使用的版本。
- 从**Eclipse官网**下载并安装Eclipse IDE for JavaScript and Web Developer最新版本。

3.4 安装 SDK

须知

开发过程中，您有任何问题可以在github上**提交issue**。**接口参考文档**详细介绍了每个接口的参数和使用方法。

使用 npm 安装（推荐）

步骤1 运行**npm -v**命令查看npm版本并确保npm已安装。

步骤2 运行**npm install esdk-obs-nodejs**命令执行安装。

----结束

说明

- 如果您使用的是Windows操作系统，当运行npm命令时提示“不是内部或外部命令”，请在Path环境变量中增加npm的安装目录（一般为Node.js的安装目录）。
- 您可能需要重启电脑使环境变量生效。
- 如果您使用npm安装依赖时出现网络错误，请使用代理。

使用源码安装

以安装OBS Node.js SDK最新版本为例，步骤如下：

步骤1 下载OBS Node.js SDK开发包。

步骤2 解压该开发包，可以看到其中包含examples文件夹（示例代码），lib文件夹（SDK源码），package.json文件（依赖配置文件）和README.txt（SDK版本特性描述文件）。

步骤3 命令行切换到SDK开发包解压目录，运行**npm install**安装依赖库，生成node_modules文件夹。

步骤4 （可选）在Eclipse JavaScript项目中导入源码。打开Eclipse JavaScript IDE，选择Import > Projects from Folder or Archive，并在Import source中选择SDK开发包解压目录。

----结束

说明

安装完成后，您的目录结构应该像下面这样：

```
├── examples
├── lib
├── node_modules
├── package.json
└── README.txt
```

3.5 获取服务地址

须知

开发过程中，您有任何问题可以在github上**提交issue**。**接口参考文档**详细介绍了每个接口的参数和使用方法。

- 您可以从**这里**查看OBS当前开通的服务地址和区域信息。

须知

SDK支持带协议名和不带协议名两种方式传入服务地址，例如获取到的服务地址为“your-endpoint”，则初始化OBS客户端时传入的服务地址可以为“http://your-endpoint”、“https://your-endpoint”和“your-endpoint”三种形式。

3.6 初始化 OBS 客户端

须知

开发过程中，您有任何问题可以在github上**提交issue**。**接口参考文档**详细介绍了每个接口的参数和使用方法。

向OBS发送任一HTTP/HTTPS请求之前，必须先创建一个ObsClient实例：


```
// 引入obs库
// 使用npm安装
var ObsClient = require('esdk-obs-nodejs');
// 使用源码安装
// var ObsClient = require('./lib/obs');

// 创建ObsClient实例
var obsClient = new ObsClient({
    //推荐通过环境变量获取AKSK，这里也可以使用其他外部引入方式传入，如果使用硬编码可能会存在泄露风险。
    //您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    access_key_id: process.env.ACCESS_KEY_ID,
    secret_access_key: process.env.SECRET_ACCESS_KEY,
    server : 'https://your-endpoint'
});

// 使用访问OBS

// 关闭obsClient
// obsClient.close();
```

⚠ 注意

- 由于JavaScript是异步编程语言，所以不能在访问OBS期间调用close方法。
- obsClient在调用obsClient.close方法关闭后不能再次使用。

说明

- 更多关于OBS客户端初始化的内容请参考“初始化”章节。
- 日志配置详见**配置SDK日志**。
- 代理配置详见**配置SDK代理**。

3.7 创建桶

须知

开发过程中，您有任何问题可以在github上**提交issue**。**接口参考文档**详细介绍了每个接口的参数和使用方法。

桶是OBS全局命名空间，相当于数据的容器、文件系统的根目录，可以存储若干对象。以下代码展示如何新建一个桶：

```
obsClient.createBucket({
    Bucket : 'bucketname',
}, (err, result) => {
    if(err){
        console.error('Error-->' + err);
    }else{
        console.log('Status-->' + result.CommonMsg.Status);
    }
});
```


说明

- 桶的名字是全局唯一的，所以您需要确保不与已有的桶名称重复。
- 桶命名规则如下：
  - 3～63个字符，数字或字母开头，支持小写字母、数字、“-”、“.”。
  - 禁止使用类IP地址。
  - 禁止以“-”或“.”开头及结尾。
  - 禁止两个“.”相邻（如：“my..bucket”）。
  - 禁止“.”和“-”相邻（如：“my-.bucket”和“my.-bucket”）。
- 同一用户多次创建同名桶不会报错，创建的桶属性以第一次请求为准。
- 更多创建桶的信息，请参见**创建桶**。

须知

- 创建桶时，如果使用的终端节点归属于默认区域华北-北京一（cn-north-1），则可以不指定区域；如果使用的终端节点归属于其他区域，则必须指定区域，且指定的区域必须与终端节点归属的区域一致。当前有效的区域名称可从**这里**查询。比如初始化时使用的终端节点EndPoint是obs. ap-southeast-1.myhuaweicloud.com，那么在创建桶的时候必须指定Location： ap-southeast-1才会创建成功，否则会返回状态码400。

3.8 上传对象

须知

开发过程中，您有任何问题可以在github上**提交issue**。**接口参考文档**详细介绍了每个接口的参数和使用方法。

以下代码展示如何上传对象至OBS：

```
obsClient.putObject({
    Bucket : 'bucketname',
    Key : 'objectname',
    Body : 'Hello OBS'
}, (err, result) => {
    if(err){
        console.error('Error-->' + err);
    }else{
        console.log('Status-->' + result.CommonMsg.Status);
    }
});
```

说明

更多上传对象的信息，请参见**上传对象**。

3.9 下载对象

须知

开发过程中，您有任何问题可以在github上**提交issue**。**接口参考文档**详细介绍了每个接口的参数和使用方法。

以下代码展示如何获取对象的内容：

```
obsClient.getObject({
    Bucket : 'bucketname',
    Key : 'objectname'
}, (err, result) => {
    if(err){
        console.error('Error-->' + err);
    }else{
        console.log('Status-->' + result.CommonMsg.Status);
        if(result.CommonMsg.Status < 300 && result.InterfaceResult){
            console.log(result.InterfaceResult.Content.toString());
        }
    }
});
```

说明

更多下载对象的信息，请参见**下载对象**。

3.10 列举对象

须知

开发过程中，您有任何问题可以在github上**提交issue**。**接口参考文档**详细介绍了每个接口的参数和使用方法。

当完成一系列上传对象操作后，可能需要查看桶中包含哪些对象。以下代码展示如何列举指定桶中的对象：

```
obsClient.listObjects({
    Bucket : 'bucketname'
}, (err, result) => {
    if(err){
        console.error('Error-->' + err);
    }else{
        console.log('Status-->' + result.CommonMsg.Status);
        if(result.CommonMsg.Status < 300 && result.InterfaceResult){
            const { Contents = [] } = result.InterfaceResult;
            for(let i=0;i<Contents.length;i++){
                console.log('Contents[' + i +  ']:');
                console.log('Key-->' + Contents[i]['Key']);
                console.log('LastModified-->' + Contents[i]['LastModified']);
                console.log('Size-->' + result.InterfaceResult.Contents[i]['Size']);
            }
        }
    }
});
```

说明

- 上面的代码默认列举1000个对象（Object）。
- 更丰富的列举功能，请参见**列举对象**。

3.11 删除对象

须知

开发过程中，您有任何问题可以在github上**提交issue**。**接口参考文档**详细介绍了每个接口的参数和使用方法。

以下代码展示如何删除指定的对象：

```
obsClient.deleteObject({
    Bucket : 'bucketname',
    Key : 'objectname'
}, (err, result) => {
    if(err){
        console.error('Error-->' + err);
    }else{
        console.log('Status-->' + result.CommonMsg.Status);
    }
});
```

3.12 OBS 客户端通用示例

须知

开发过程中，您有任何问题可以在github上**提交issue**。**接口参考文档**详细介绍了每个接口的参数和使用方法。

回调函数返回结果：

OBS客户端可通过回调函数的形式返回结果，回调函数依次包含异常信息和**SDK公共结果对象**两个参数。如果回调函数中异常信息参数不为空，则表明接口调用异常；反之，则表明接口调用完成，此时应从**SDK公共结果对象**中获取HTTP状态码，判断操作是否成功。示例代码如下：

```
// 引入obs库
// 使用npm安装
var ObsClient = require('esdk-obs-nodejs');
// 使用源码安装
// var ObsClient = require('./lib/obs');

// 创建ObsClient实例
var obsClient = new ObsClient({
    //推荐通过环境变量获取AKSK，这里也可以使用其他外部引入方式传入，如果使用硬编码可能会存在泄露风险。
    //您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    access_key_id: process.env.ACCESS_KEY_ID,
    secret_access_key: process.env.SECRET_ACCESS_KEY,
    server : 'https://your-endpoint'
```



```
});

// 构造桶操作请求参数
var requestParam1 = {
    Bucket : 'bucketname'
    // 其他字段
};

var callback1 = (err, result) => {
    // 处理桶操作调用结果
};

// 调用桶操作接口，如创建桶
obsClient.createBucket(requestParam1, callback1);

//构造对象操作请求参数
var requestParam2 = {
    Bucket : 'bucketname',
    Key : 'objectname'
    // 其他字段
};

var callback2 = (err, result) => {
    // 处理对象操作调用结果
};

// 调用对象操作接口，如下载对象
obsClient.getObject(requestParam2, callback2);
```

说明

对于桶操作接口，请求对象中固定包含Bucket字段用于指定桶名；对于对象操作接口，请求对象中固定包含Bucket字段和Key字段分别用于指定桶名与对象名。

以下代码展示了使用回调函数返回调用结果的通用示例：

```
// 引入obs库
// 使用npm安装
var ObsClient = require('esdk-obs-nodejs');
// 使用源码安装
// var ObsClient = require('./lib/obs');

// 创建ObsClient实例
var obsClient = new ObsClient({
    //推荐通过环境变量获取AKSK，这里也可以使用其他外部引入方式传入，如果使用硬编码可能会存在泄露风险。
    //您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    access_key_id: process.env.ACCESS_KEY_ID,
    secret_access_key: process.env.SECRET_ACCESS_KEY,
    server : 'https://your-endpoint'
});

// 调用接口进行操作，例如上传对象
obsClient.putObject({
    Bucket : 'bucketname',
    Key : 'objectname',
    Body : 'Hello OBS'
}, (err, result) => {
    // 异常信息不为空，表明接口调用异常
    if(err){
        console.log('Error-->' + err);
    }else{
        // 异常信息为空，表明接口调用完成，应进一步判断HTTP状态码
        if(result.CommonMsg.Status < 300){// 操作成功
            if(result.InterfaceResult){
                // 处理操作成功后业务逻辑
            }
        }else{// 操作失败，获取详细异常信息
            console.log('Code-->' + result.CommonMsg.Code);
```



```
                console.log('Message-->' + result.CommonMsg.Message);
                console.log('HostId-->' + result.CommonMsg.HostId);
                console.log('RequestId-->' + result.CommonMsg.RequestId);
            }
        }
});
```

Promise 对象返回结果：

OBS客户端可通过Promise对象返回结果，如果通过Promise对象的catch方法没有捕获到异常，则表明接口调用完成，此时应从**SDK公共结果对象**中获取HTTP状态码，判断操作是否成功。示例代码如下：

```
// 引入obs库
// 使用npm安装
var ObsClient = require('esdk-obs-nodejs');
// 使用源码安装
// var ObsClient = require('./lib/obs');

// 创建ObsClient实例
var obsClient = new ObsClient({
    //推荐通过环境变量获取AKSK，这里也可以使用其他外部引入方式传入，如果使用硬编码可能会存在泄露风险。
    //您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    access_key_id: process.env.ACCESS_KEY_ID,
    secret_access_key: process.env.SECRET_ACCESS_KEY,
    server : 'https://your-endpoint'
});

// 构造桶操作请求参数
var requestParam1 = {
    Bucket : 'bucketname'
    // 其他字段
};

// 调用桶操作接口，如创建桶
var promise1 = obsClient.createBucket(requestParam1);
promise1.then((result) => {
  // 处理桶操作调用结果
}).catch((err)=>{
  // 处理错误
});

//构造对象操作请求参数
var requestParam2 = {
    Bucket : 'bucketname',
    Key : 'objectname'
    // 其他字段
};

// 调用对象操作接口，如下载对象
var promise2 = obsClient.getObject(requestParam2);
promise2.then((result) => {
  // 处理对象操作调用结果
}).catch((err)=>{
  // 处理错误
});
```

说明

对于桶操作接口，请求对象中固定包含Bucket字段用于指定桶名；对于对象操作接口，请求对象中固定包含Bucket字段和Key字段分别用于指定桶名与对象名。

以下代码展示了使用Promise对象返回调用结果的通用示例：

```
// 引入obs库
// 使用npm安装
```



```
var ObsClient = require('esdk-obs-nodejs');
// 使用源码安装
// var ObsClient = require('./lib/obs');

// 创建ObsClient实例
var obsClient = new ObsClient({
      //推荐通过环境变量获取AKSK，这里也可以使用其他外部引入方式传入，如果使用硬编码可能会存在泄露风险。
      //您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
      access_key_id: process.env.ACCESS_KEY_ID,
      secret_access_key: process.env.SECRET_ACCESS_KEY,
      server : 'https://your-endpoint'
});

// 调用接口进行操作，例如上传对象
obsClient.putObject({
      Bucket : 'bucketname',
      Key : 'objectname',
      Body : 'Hello OBS'
}).then((result) => {
    // 接口调用完成，应进一步判断HTTP状态码
    if(result.CommonMsg.Status < 300){// 操作成功
        if(result.InterfaceResult){
            // 处理操作成功后业务逻辑
        }
    }else{// 操作失败，获取详细异常信息
        console.log('Code-->' + result.CommonMsg.Code);
        console.log('Message-->' + result.CommonMsg.Message);
        console.log('HostId-->' + result.CommonMsg.HostId);
        console.log('RequestId-->' + result.CommonMsg.RequestId);
    }
}).catch((err) => {
    // 接口调用发生异常
    console.error('Error-->' + err);
});
```

3.13 预定义常量

须知

开发过程中，您有任何问题可以在github上**提交issue**。**接口参考文档**详细介绍了每个接口的参数和使用方法。

OBS Node.js SDK提供了一组预定义常量，方便用户直接使用。您可以通过**obsClient.enums**或**ObsClient.prototype.enums**获取预定义常量对象。更多关于预定义常量的介绍详见《对象存储服务Node.js SDK API参考》。

4 初始化

4.1 配置密钥

须知

开发过程中，您有任何问题可以在github上**提交issue**。**接口参考文档**详细介绍了每个接口的参数和使用方法。

要接入OBS服务，您需要拥有一组有效的访问密钥（AK和SK）用来进行签名认证。具体可参考**OBS服务环境搭建**。

获取AK和SK之后，可以通过创建OBS客户端，调用SDK接口。

4.2 创建 OBS 客户端

须知

开发过程中，您有任何问题可以在github上**提交issue**。**接口参考文档**详细介绍了每个接口的参数和使用方法。

OBS客户端（ObsClient）是访问OBS服务的Node.js客户端，它为调用者提供一系列与OBS服务进行交互的接口，用于管理、操作桶（Bucket）和对象（Object）等OBS服务上的资源。使用OBS Node.js SDK发起OBS请求，您需要初始化一个ObsClient实例，并根据需要修改客户端初始化配置参数。

通过构造函数创建

- 永久访问密钥（AK/SK）创建OBS客户端代码如下：

```
// 引入obs库
// 使用npm安装
var ObsClient = require('esdk-obs-nodejs');
// 使用源码安装
// var ObsClient = require('./lib/obs');
```

```
// 通过构造函数创建ObsClient实例
var obsClient = new ObsClient({
    //推荐通过环境变量获取AKSK，这里也可以使用其他外部引入方式传入，如果使用硬编码可能会存在泄露风险。
    //您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    access_key_id: process.env.ACCESS_KEY_ID,
    secret_access_key: process.env.SECRET_ACCESS_KEY,
    server : 'https://your-endpoint'
});

// 使用访问OBS

// 关闭obsClient
// obsClient.close();
```

- 临时访问密钥（AK/SK/SecurityToken）创建OBS客户端代码如下：

```
// 引入obs库
// 使用npm安装
var ObsClient = require('esdk-obs-nodejs');
// 使用源码安装
// var ObsClient = require('./lib/obs');

// 通过构造函数创建ObsClient实例
var obsClient = new ObsClient({
    //推荐通过环境变量获取AKSK，这里也可以使用其他外部引入方式传入，如果使用硬编码可能会存在泄露风险。
    //您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    access_key_id: process.env.ACCESS_KEY_ID,
    secret_access_key: process.env.SECRET_ACCESS_KEY,
    security_token: process.env.SECURITY_TOKEN,
    server : 'https://your-endpoint'
});

// 使用访问OBS

// 关闭obsClient
// obsClient.close();
```

通过工厂方法创建

- 永久访问密钥（AK/SK）创建OBS客户端代码如下：

```
// 引入obs库
// 使用npm安装
var ObsClient = require('esdk-obs-nodejs');
// 使用源码安装
// var ObsClient = require('./lib/obs');

// 通过工厂方法初始化ObsClient实例
var obsClient = new ObsClient();
obsClient.factory({
    //推荐通过环境变量获取AKSK，这里也可以使用其他外部引入方式传入，如果使用硬编码可能会存在泄露风险。
    //您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    access_key_id: process.env.ACCESS_KEY_ID,
    secret_access_key: process.env.SECRET_ACCESS_KEY,
    security_token: process.env.SECURITY_TOKEN,
    server : 'https://your-endpoint'
});

// 使用访问OBS

// 关闭obsClient
// obsClient.close();
```

- 临时访问密钥（AK/SK/SecurityToken）创建OBS客户端代码如下：

```
// 引入obs库
// 使用npm安装
```



```
var ObsClient = require('esdk-obs-nodejs');
// 使用源码安装
// var ObsClient = require('./lib/obs');

// 通过工厂方法初始化ObsClient实例
var obsClient = new ObsClient();
obsClient.factory({
        //推荐通过环境变量获取AKSK，这里也可以使用其他外部引入方式传入，如果使用硬编码可能会存在泄露风险。
        //您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
        access_key_id: process.env.ACCESS_KEY_ID,
        secret_access_key: process.env.SECRET_ACCESS_KEY,
        security_token: process.env.SECURITY_TOKEN,

        server : 'https://your-endpoint'
});

// 使用访问OBS

// 关闭obsClient
// obsClient.close();
```

说明

- 您的工程中可以有多个obsClient实例，也可以只有一个。
- obsClient实例在调用close方法关闭后不能再次使用。
- 配置SDK代理请参见**配置SDK代理**。

4.3 配置 OBS 客户端

须知

开发过程中，您有任何问题可以在github上**提交issue**。**接口参考文档**详细介绍了每个接口的参数和使用方法。

您可通过初始化参数对ObsClient进行配置，可以配置的参数见下表：

参数	描述	建议值
access_key_id	访问密钥中的AK。	N/A
secret_access_key	访问密钥中的SK。	N/A
server	连接OBS的服务地址。可包含协议类型、域名、端口号。示例：https://your-endpoint:443。（出于安全性考虑，建议使用https协议）	N/A
max_retry_count	HTTP/HTTPS连接异常时的请求重试次数。默认为3次。	[1，5]
timeout	HTTP/HTTPS请求超时时间（单位：秒）。默认为60秒。	[10, 60]
http_agent	配置使用http代理，可选项。默认不配置	N/A

参数	描述	建议值
https_agent	配置使用https代理，可选项。默认不配置	N/A
ssl_verify	验证服务端证书参数。可能的取值： • 服务端pem格式根证书文件路径； • true：使用默认的CAs验证服务端证书； • false：表示不验证服务端证书。 默认为false。	N/A
long_conn_param	长连接模式参数（单位：秒）。当该参数大于等于0时，开启长连接模式，并将该参数作为TCP Keep-Alive数据包的**初始延迟**。 默认为空，代表关闭长连接模式。	N/A
is_cname	是否通过自定义域名访问OBS服务。默认为false。	N/A

说明

- 建议值为N/A的表示需要根据实际情况进行设置。
- 如网络状况不佳，建议增大timeout的值。
- 如果设置的server不带协议类型，则默认使用HTTPS协议。

须知

- 如果启用了长连接模式，使用完OBS客户端后必须调用ObsClient.close方法显式关闭，回收连接资源。
- 出于DNS解析性能和OBS服务可靠性的考虑，不允许将server设置为IP，必须使用域名访问OBS服务。
- 配置SDK代理请参见**配置SDK代理**。

4.4 配置 SDK 日志

须知

开发过程中，您有任何问题可以在github上**提交issue**。**接口参考文档**详细介绍了每个接口的参数和使用方法。

OBS Node.js SDK基于log4js开源库提供了日志功能，您可以通过ObsClient.initLog开启日志功能并进行配置。示例代码如下：

```
obsClient.initLog({
    file_full_path:'./logs/OBS-SDK.log', // 配置日志文件路径
    max_log_size:20480, // 配置日志文件大小，单位：字节
    backups:10, // 配置最大可保留的日志文件个数
    level:'warn', // 配置日志级别
    log_to_console:true // 配置是否将日志打印到console
});
```

说明

- 日志功能默认是关闭的，需要主动开启。
- 您可以从**日志分析**章节获取更多关于SDK日志的信息。

4.5 配置 SDK 代理

须知

开发过程中，您有任何问题可以在github上**提交issue**。**接口参考文档**详细介绍了每个接口的参数和使用方法。

步骤1 在项目工程根目录下使用npm命令下载**proxy-agent**依赖包。

```
npm install --save proxy-agent
```

步骤2 在项目工程里面声明并配置代理。

```
const ProxyAgent = require("proxy-agent");
const proxyAgent = new ProxyAgent(`http://username:password@proxyhost:proxyPort`);
```

步骤3 在项目工程里面初始化OBS客户端并配置代理。

```
var obsClient = new ObsClient({
    //推荐通过环境变量获取AKSK，这里也可以使用其他外部引入方式传入，如果使用硬编码可能会存在泄露风险。
    //您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    access_key_id: process.env.ACCESS_KEY_ID,
    secret_access_key: process.env.SECRET_ACCESS_KEY,
    security_token: process.env.SECURITY_TOKEN,
    server : 'https://your-endpoint',
    https_agent:proxyAgent,
    http_agent: proxyAgent,
});
```

----结束

5 管理桶

5.1 创建桶

须知

开发过程中，您有任何问题可以在github上**提交issue**。**接口参考文档**详细介绍了每个接口的参数和使用方法。

您可以通过ObsClient.createBucket创建桶。创建桶时可以指定桶的访问权限、存储类型和区域位置。OBS支持的桶的存储类型有三类，参见**桶存储类型**。示例代码如下：

```
// 引入obs库
// 使用npm安装
var ObsClient = require('esdk-obs-nodejs');
// 使用源码安装
// var ObsClient = require('./lib/obs');

// 创建ObsClient实例
var obsClient = new ObsClient({
    //推荐通过环境变量获取AKSK，这里也可以使用其他外部引入方式传入，如果使用硬编码可能会存在泄露风险。
    //您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    access_key_id: process.env.ACCESS_KEY_ID,
    secret_access_key: process.env.SECRET_ACCESS_KEY,
    server : 'https://your-endpoint'
});

// 创建桶
obsClient.createBucket({
    Bucket : 'bucketname',
    // 设置桶访问权限为公共读，默认是私有读写
    ACL : obsClient.enums.AclPublicRead,
    // 设置桶的存储类型为归档存储类型
    StorageClass : obsClient.enums.StorageClassStandard,
    // 设置桶区域位置
    Location : 'bucketlocation',


}, (err, result) => {
    if(err){
        console.error('Error-->' + err);
    }else{
        console.log('Status-->' + result.CommonMsg.Status);
```

```
    }
});
```

说明

- 桶的名字是全局唯一的，所以您需要确保不与已有的桶名称重复。
- 桶命名规则如下：
  - 3～63个字符，数字或字母开头，支持小写字母、数字、“-”、“.”。
  - 禁止使用类IP地址。
  - 禁止以“-”或“.”开头及结尾。
  - 禁止两个“.”相邻（如：“my..bucket”）。
  - 禁止“.”和“-”相邻（如：“my-.bucket”和“my.-bucket”）。
- 同一用户在同一个区域多次创建同名桶不会报错，创建的桶属性以第一次请求为准。
- 本示例创建的桶的**访问权限**默认是私有读写，存储类型默认是标准类型，区域位置是默认区域。
- 使用**ACL参数**指定桶的访问权限；使用**StorageClass参数**指定桶的存储类型；使用**Location参数**指定桶的区域位置。

须知

- 创建桶时，如果使用的终端节点归属于默认区域华北-北京一（cn-north-1），则可以不指定区域；如果使用的终端节点归属于其他区域，则必须指定区域，且指定的区域必须与终端节点归属的区域一致。当前有效的区域名称可从**这里**查询。比如初始化时使用的终端节点EndPoint是obs. ap-southeast-1.myhuaweicloud.com，那么在创建桶的时候必须指定Location： ap-southeast-1才会创建成功，否则会返回状态码400。

5.2 列举桶

须知

开发过程中，您有任何问题可以在github上**提交issue**。**接口参考文档**详细介绍了每个接口的参数和使用方法。

您可以通过ObsClient.listBuckets列举桶。以下代码展示如何获取桶列表：

```
// 引入obs库
// 使用npm安装
var ObsClient = require('esdk-obs-nodejs');
// 使用源码安装
// var ObsClient = require('./lib/obs');

// 创建ObsClient实例
var obsClient = new ObsClient({
    //推荐通过环境变量获取AKSK，这里也可以使用其他外部引入方式传入，如果使用硬编码可能会存在泄露风险。
    //您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    access_key_id: process.env.ACCESS_KEY_ID,
    secret_access_key: process.env.SECRET_ACCESS_KEY,
    server : 'https://your-endpoint'
});
```



```
// 列举桶
obsClient.listBuckets({
    QueryLocation : true
}, (err, result) => {
    if(err){
        console.error('Error-->' + err);
    }else{
        console.log('Status-->' + result.CommonMsg.Status);
        if(result.CommonMsg.Status < 300 && result.InterfaceResult){
            console.log('RequestId-->' + result.InterfaceResult.RequestId);
            console.log('Owner:');
            console.log('ID-->' + result.InterfaceResult.Owner.ID);
            console.log('Name-->' + result.InterfaceResult.Owner.Name);
            console.log('Buckets:');
             for(let i=0;i<result.InterfaceResult.Buckets.length;i++){
                console.log('Bucket[' + i + ']:');
                console.log('BucketName-->' + result.InterfaceResult.Buckets[i].BucketName);
                console.log('CreationDate-->' + result.InterfaceResult.Buckets[i].CreationDate);
                console.log('Location-->' + result.InterfaceResult.Buckets[i].Location);
             }
        }
    }
});
```

说明

- 获取到的桶列表将按照桶名字典顺序排列。
- 设置**QueryLocation参数**为true后，可在列举桶时查询桶的区域位置

5.3 删除桶

须知

开发过程中，您有任何问题可以在github上**提交issue**。**接口参考文档**详细介绍了每个接口的参数和使用方法。

您可以通过ObsClient.deleteBucket删除桶。以下代码展示如何删除一个桶：

```
// 引入obs库
// 使用npm安装
var ObsClient = require('esdk-obs-nodejs');
// 使用源码安装
// var ObsClient = require('./lib/obs');

// 创建ObsClient实例
var obsClient = new ObsClient({
    //推荐通过环境变量获取AKSK，这里也可以使用其他外部引入方式传入，如果使用硬编码可能会存在泄露风险。
    //您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    access_key_id: process.env.ACCESS_KEY_ID,
    secret_access_key: process.env.SECRET_ACCESS_KEY,
    server : 'https://your-endpoint'
});

// 删除桶
obsClient.deleteBucket({
    Bucket : 'bucketname'
}, (err, result) => {
    if(err){
        console.error('Error-->' + err);
    }else{
```



```
        console.log('Status-->' + result.CommonMsg.Status);
    }
});
```

⚠ 注意

删除30分钟后，可再次使用该名称创建其他区域新桶或并行文件系统。

说明

- 如果桶不为空（包含对象或分段上传碎片），则该桶无法删除。
- 删除桶非幂等操作，删除不存在的桶会失败。

5.4 判断桶是否存在

须知

开发过程中，您有任何问题可以在github上**提交issue**。**接口参考文档**详细介绍了每个接口的参数和使用方法。

您可以通过ObsClient.headBucket接口判断该桶是否已存在。以下代码展示如何判断指定桶是否存在：

```
// 引入obs库
// 使用npm安装
var ObsClient = require('esdk-obs-nodejs');
// 使用源码安装
// var ObsClient = require('./lib/obs');

// 创建ObsClient实例
var obsClient = new ObsClient({
    //推荐通过环境变量获取AKSK，这里也可以使用其他外部引入方式传入，如果使用硬编码可能会存在泄露风险。
    //您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    access_key_id: process.env.ACCESS_KEY_ID,
    secret_access_key: process.env.SECRET_ACCESS_KEY,
    server : 'https://your-endpoint'
});

obsClient.headBucket({
    Bucket : 'bucketname'
}, (err, result) => {
    if(err){
        console.error('Error-->' + err);
    }else{
        if(result.CommonMsg.Status < 300){
            console.log('Bucket exists');
        }else if(result.CommonMsg.Status === 404){
            console.log('Bucket does not exist');
        }
    }
});
```


5.5 获取桶元数据

须知

开发过程中，您有任何问题可以在github上**提交issue**。**接口参考文档**详细介绍了每个接口的参数和使用方法。

您可以通过ObsClient.getBucketMetadata接口获取桶元数据。以下代码展示如何获取桶元数据：

```
// 引入obs库
// 使用npm安装
var ObsClient = require('esdk-obs-nodejs');
// 使用源码安装
// var ObsClient = require('./lib/obs');

// 创建ObsClient实例
var obsClient = new ObsClient({
    //推荐通过环境变量获取AKSK，这里也可以使用其他外部引入方式传入，如果使用硬编码可能会存在泄露风险。
    //您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    access_key_id: process.env.ACCESS_KEY_ID,
    secret_access_key: process.env.SECRET_ACCESS_KEY,
    server : 'https://your-endpoint'
});

// 获取桶元数据
obsClient.getBucketMetadata({
    Bucket : 'bucketname',
    Origin : 'http://www.a.com',
}, (err, result) => {
    if(err){
        console.error('Error-->' + err);
    }else{
        console.log('Status-->' + result.CommonMsg.Status);
        if(result.CommonMsg.Status < 300 && result.InterfaceResult){
            console.log('StorageClass-->' + result.InterfaceResult.StorageClass);
            console.log('AllowOrigin-->' + result.InterfaceResult.AllowOrigin);
            console.log('MaxAgeSeconds-->' + result.InterfaceResult.MaxAgeSeconds);
            console.log('ExposeHeader-->' + result.InterfaceResult.ExposeHeader);
            console.log('AllowMethod-->' + result.InterfaceResult.AllowMethod);
            console.log('AllowHeader-->' + result.InterfaceResult.AllowHeader);
        }
    }
});
```

5.6 管理桶访问权限

须知

开发过程中，您有任何问题可以在github上**提交issue**。**接口参考文档**详细介绍了每个接口的参数和使用方法。

桶访问权限（**ACL**）可以通过三种方式设置：

1. 创建桶时指定预定义访问策略。
2. 调用ObsClient.setBucketAcl指定预定义访问策略。
3. 调用ObsClient.setBucketAcl直接设置。

OBS支持的桶或对象权限包含五类，见下表：

权限	描述	OBS Node.js SDK对应值
读权限	如果有桶的读权限，则可以获取该桶内对象列表和桶的元数据。 如果有对象的读权限，则可以获取该对象内容和元数据。	ObsClient.enums.PermissionRead
写权限	如果有桶的写权限，则可以上传、覆盖和删除该桶内任何对象。 此权限在对象上不适用。	ObsClient.enums.PermissionWrite
读ACP权限	如果有读ACP的权限，则可以获取对应的桶或对象的权限控制列表（ACL）。 桶或对象的所有者永远拥有读对应桶或对象ACP的权限。	ObsClient.enums.PermissionReadAcp
写ACP权限	如果有写ACP的权限，则可以更新对应桶或对象的权限控制列表（ACL）。 桶或对象的所有者永远拥有写对应桶或对象的ACP的权限。 拥有了写ACP的权限，由于可以更改权限控制策略，实际上意味着拥有了完全访问的权限。	ObsClient.enums.PermissionWriteAcp
完全控制权限	如果有桶的完全控制权限意味着拥有读权限、写权限、读ACP权限和写ACP权限的权限。 如果有对象的完全控制权限意味着拥有读权限、读ACP权限和写ACP权限的权限	ObsClient.enums.PermissionFullControl

OBS预定义的访问策略包含五类，见下表：

权限	描述	OBS Node.js SDK对应值
私有读写	桶或对象的所有者拥有完全控制的权限，其他任何人都没有访问权限。	ObsClient.enums.AclPrivate

权限	描述	OBS Node.js SDK对应值
公共读	设在桶上，所有人可以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。 设在对象上，所有人可以获取该对象内容和元数据。	ObsClient.enums.AclPublicRead
公共读写	设在桶上，所有人可以获取该桶内对象列表、桶内多段任务、桶的元数据、上传对象、删除对象、初始化段任务、上传段、合并段、拷贝段、取消多段上传任务。 设在对象上，所有人可以获取该对象内容和元数据。	ObsClient.enums.AclPublicReadWrite
桶公共读，桶内对象公共读	设在桶上，所有人可以获取该桶内对象列表、桶内多段任务、桶的元数据，可以获取该桶内对象的内容和元数据。 不能应用于对象。	ObsClient.enums.AclPublicReadDelivered
桶公共读写，桶内对象公共读写	设在桶上，所有人可以获取该桶内对象列表、桶内多段任务、桶的元数据、上传对象、删除对象、初始化段任务、上传段、合并段、拷贝段、取消多段上传任务，可以获取该桶内对象的内容和元数据。 不能应用于对象。	ObsClient.enums.AclPublicReadWriteDelivered

创桶时指定预定义访问策略

以下代码展示如何在创建桶时指定预定义访问策略：

```
// 引入obs库
// 使用npm安装
var ObsClient = require('esdk-obs-nodejs');
// 使用源码安装
// var ObsClient = require('./lib/obs');

// 创建ObsClient实例
var obsClient = new ObsClient({
        //推荐通过环境变量获取AKSK，这里也可以使用其他外部引入方式传入，如果使用硬编码可能会存在泄露风险。
        //您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
        access_key_id: process.env.ACCESS_KEY_ID,
        secret_access_key: process.env.SECRET_ACCESS_KEY,
        server : 'https://your-endpoint'
});

// 创建桶
obsClient.createBucket({
        Bucket : 'bucketname',
```



```
    // 设置桶访问权限为公共读写
    ACL : obsClient.enums.AclPublicReadWrite
}, (err, result) => {
    if(err){
        console.error('Error-->' + err);
    }else{
        console.log('Status-->' + result.CommonMsg.Status);
    }
});
```

为桶设置预定义访问策略

以下代码展示如何为桶设置预定义访问策略：

```
// 引入obs库
// 使用npm安装
var ObsClient = require('esdk-obs-nodejs');
// 使用源码安装
// var ObsClient = require('./lib/obs');

// 创建ObsClient实例
var obsClient = new ObsClient({
    //推荐通过环境变量获取AKSK，这里也可以使用其他外部引入方式传入，如果使用硬编码可能会存在泄露风险。
    //您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    access_key_id: process.env.ACCESS_KEY_ID,
    secret_access_key: process.env.SECRET_ACCESS_KEY,
    server : 'https://your-endpoint'
});

// 用预定义访问策略设置桶权限
obsClient.setBucketAcl({
    Bucket : 'bucketname',
    // 设置桶访问权限为私有读写
    ACL : obsClient.enums.AclPrivate
}, (err, result) => {
    if(err){
        console.error('Error-->' + err);
    }else{
        console.log('Status-->' + result.CommonMsg.Status);
    }
});
```

说明

使用**ACL参数**指定桶的访问权限。

直接设置桶访问权限

以下代码展示如何直接设置桶访问权限：

```
// 引入obs库
// 使用npm安装
var ObsClient = require('esdk-obs-nodejs');
// 使用源码安装
// var ObsClient = require('./lib/obs');

// 创建ObsClient实例
var obsClient = new ObsClient({
    //推荐通过环境变量获取AKSK，这里也可以使用其他外部引入方式传入，如果使用硬编码可能会存在泄露风险。
    //您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    access_key_id: process.env.ACCESS_KEY_ID,
    secret_access_key: process.env.SECRET_ACCESS_KEY,
    server : 'https://your-endpoint'
```

```
});

// 直接设置桶访问权限
obsClient.setBucketAcl({
      Bucket : 'bucketname',
      // 设置桶所有者
      Owner:{'ID':'ownerid'},
      Grants:[
           // 为指定用户设置完全控制权限
          { Grantee : {Type : 'CanonicalUser',ID : 'userid'}, Permission : obsClient.enums.PermissionFullControl},
           // 为所有用户设置读权限
          { Grantee : {Type : 'Group',URI : obsClient.enums.GroupAllUsers}, Permission :
obsClient.enums.PermissionRead}
     ]
}, (err, result) => {
      if(err){
            console.error('Error-->' + err);
      }else{
            console.log('Status-->' + result.CommonMsg.Status);
      }
});
```

说明

- 使用**Owner参数**指定桶的所有者信息；使用**Grants参数**指定被授权的用户信息。
- ACL中需要填写的所有者（Owner）或者被授权用户（Grantee）的ID，是指用户的账户ID，可通过OBS控制台“我的凭证”页面查看。
- 当前OBS桶支持的可被授权的用户组为：
  - 所有用户：ObsClient.enums.GroupAllUsers

获取桶访问权限

您可以通过ObsClient.getBucketAcl获取桶的访问权限。以下代码展示如何获取桶访问权限：

```
// 引入obs库
// 使用npm安装
var ObsClient = require('esdk-obs-nodejs');
// 使用源码安装
// var ObsClient = require('./lib/obs');

// 创建ObsClient实例
var obsClient = new ObsClient({
      //推荐通过环境变量获取AKSK，这里也可以使用其他外部引入方式传入，如果使用硬编码可能会存在泄露风险。
      //您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
      access_key_id: process.env.ACCESS_KEY_ID,
      secret_access_key: process.env.SECRET_ACCESS_KEY,
      server : 'https://your-endpoint'
});

obsClient.getBucketAcl({
      Bucket : 'bucketname',
}, (err, result) => {
      if(err){
            console.error('Error-->' + err);
      }else{
            console.log('Status-->' + result.CommonMsg.Status);
            if(result.CommonMsg.Status < 300 && result.InterfaceResult){
                  console.log('RequestId-->' + result.InterfaceResult.RequestId);
                  console.log('Owner[ID]-->' + result.InterfaceResult.Owner.ID);
                  console.log('Grants:');
                  for(let i=0;i<result.InterfaceResult.Grants.length;i++){
                        console.log('Grant[' + i + ']:');
                        console.log('Grantee[ID]-->' + result.InterfaceResult.Grants[i]['Grantee']['ID']);
```

```
                    console.log('Grantee[URI]-->' + result.InterfaceResult.Grants[i]['Grantee']['URI']);
                    console.log('Permission-->' + result.InterfaceResult.Grants[i]['Permission']);
                }
            }
        }
});
```

5.7 管理桶策略

须知

开发过程中，您有任何问题可以在github上**提交issue**。**接口参考文档**详细介绍了每个接口的参数和使用方法。

除了桶访问权限外，桶的拥有者还可以通过桶策略，提供对桶和桶内对象的集中访问控制。

更多关于桶策略的内容请参考**桶策略**。

设置桶策略

您可以通过ObsClient.setBucketPolicy设置桶策略。示例代码如下：

```
// 引入obs库
// 使用npm安装
var ObsClient = require('esdk-obs-nodejs');
// 使用源码安装
// var ObsClient = require('./lib/obs');

// 创建ObsClient实例
var obsClient = new ObsClient({
    //推荐通过环境变量获取AKSK，这里也可以使用其他外部引入方式传入，如果使用硬编码可能会存在泄露风险。
    //您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    access_key_id: process.env.ACCESS_KEY_ID,
    secret_access_key: process.env.SECRET_ACCESS_KEY,
    server : 'https://your-endpoint'
});

// 桶名
const bucketName = 'bucketname';
// 桶策略
const policy =  "{\"Statement\":[{\"Principal\":\"*\",\"Effect\":\"Allow\",\"Action\":\"ListBucket\",\"Resource\":\""+bucketName+"\"}]}";
// 设置桶策略
obsClient.setBucketPolicy({
    Bucket: bucketName,
    Policy: policy
}, function(err, result) {
    if(err){
        console.error('Error-->' + err);
    }else{
        console.log('Status-->' + result.CommonMsg.Status);
    }
});
```

说明

桶策略内容的具体格式（JSON格式字符串）请参考《对象存储服务 API参考》。

获取桶策略

您可以通过ObsClient.getBucketPolicy获取桶策略。示例代码如下：

```
// 引入obs库
// 使用npm安装
var ObsClient = require('esdk-obs-nodejs');
// 使用源码安装
// var ObsClient = require('./lib/obs');

// 创建ObsClient实例
var obsClient = new ObsClient({
    //推荐通过环境变量获取AKSK，这里也可以使用其他外部引入方式传入，如果使用硬编码可能会存在泄露风险。
    //您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    access_key_id: process.env.ACCESS_KEY_ID,
    secret_access_key: process.env.SECRET_ACCESS_KEY,
    server : 'https://your-endpoint'
});

// 获取桶策略
obsClient.getBucketPolicy({
    Bucket : 'bucketname',
}, (err, result) => {
    if(err){
        console.error('Error-->' + err);
    }else{
        console.log('Status-->' + result.CommonMsg.Status);
        if(result.CommonMsg.Status < 300 && result.InterfaceResult){
            console.log('Policy-->' + result.InterfaceResult.Policy);
        }
    }
});
```

删除桶策略

您可以通过ObsClient.deleteBucketPolicy删除桶策略。示例代码如下：

```
// 引入obs库
// 使用npm安装
var ObsClient = require('esdk-obs-nodejs');
// 使用源码安装
// var ObsClient = require('./lib/obs');

// 创建ObsClient实例
var obsClient = new ObsClient({
    //推荐通过环境变量获取AKSK，这里也可以使用其他外部引入方式传入，如果使用硬编码可能会存在泄露风险。
    //您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    access_key_id: process.env.ACCESS_KEY_ID,
    secret_access_key: process.env.SECRET_ACCESS_KEY,
    server : 'https://your-endpoint'
});

// 删除桶策略
obsClient.deleteBucketPolicy({
    Bucket : 'bucketname'
}, (err, result) => {
    if(err){
        console.error('Error-->' + err);
    }else{
        console.log('Status-->' + result.CommonMsg.Status);
    }
});
```


5.8 获取桶区域位置

须知

开发过程中，您有任何问题可以在github上**提交issue**。**接口参考文档**详细介绍了每个接口的参数和使用方法。

您可以通过ObsClient.getBucketLocation获取桶的区域位置。以下代码展示如何获取桶区域位置：

```
// 引入obs库
// 使用npm安装
var ObsClient = require('esdk-obs-nodejs');
// 使用源码安装
// var ObsClient = require('./lib/obs');

// 创建ObsClient实例
var obsClient = new ObsClient({
    //推荐通过环境变量获取AKSK，这里也可以使用其他外部引入方式传入，如果使用硬编码可能会存在泄露风险。
    //您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    access_key_id: process.env.ACCESS_KEY_ID,
    secret_access_key: process.env.SECRET_ACCESS_KEY,
    server : 'https://your-endpoint'
});

obsClient.getBucketLocation({
    Bucket : 'bucketname',
}, (err, result) => {
    if(err){
        console.error('Error-->' + err);
    }else{
        console.log('Status-->' + result.CommonMsg.Status);
        if(result.CommonMsg.Status < 300 && result.InterfaceResult){
            console.log('Location-->' + result.InterfaceResult.Location);
        }
    }
});
```

说明

创建桶时可以指定桶的区域位置，请参见**创建桶**。

5.9 获取桶存量信息

须知

开发过程中，您有任何问题可以在github上**提交issue**。**接口参考文档**详细介绍了每个接口的参数和使用方法。

桶存量信息包括桶的空间大小以及桶包含的对象个数。您可以通过ObsClient.getBucketStorageInfo获取桶的存量信息。以下代码展示如何获取桶存量信息：


```
// 引入obs库
// 使用npm安装
var ObsClient = require('esdk-obs-nodejs');
// 使用源码安装
// var ObsClient = require('./lib/obs');

// 创建ObsClient实例
var obsClient = new ObsClient({
    //推荐通过环境变量获取AKSK，这里也可以使用其他外部引入方式传入，如果使用硬编码可能会存在泄露风险。
    //您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    access_key_id: process.env.ACCESS_KEY_ID,
    secret_access_key: process.env.SECRET_ACCESS_KEY,
    server : 'https://your-endpoint'
});

obsClient.getBucketStorageInfo({
    Bucket : 'bucketname',
}, (err, result) => {
    if(err){
        console.error('Error-->' + err);
    }else{
        console.log('Status-->' + result.CommonMsg.Status);
        if(result.CommonMsg.Status < 300 && result.InterfaceResult){
            console.log('Size-->' + result.InterfaceResult.Size);
            console.log('ObjectNumber-->' + result.InterfaceResult.ObjectNumber);
        }
    }
});
```

5.10 桶配额

须知

开发过程中，您有任何问题可以在github上**提交issue**。**接口参考文档**详细介绍了每个接口的参数和使用方法。

设置桶配额

您可以通过ObsClient.setBucketQuota设置桶配额。以下代码展示如何设置桶配额：

```
// 引入obs库
// 使用npm安装
var ObsClient = require('esdk-obs-nodejs');
// 使用源码安装
// var ObsClient = require('./lib/obs');

// 创建ObsClient实例
var obsClient = new ObsClient({
    //推荐通过环境变量获取AKSK，这里也可以使用其他外部引入方式传入，如果使用硬编码可能会存在泄露风险。
    //您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    access_key_id: process.env.ACCESS_KEY_ID,
    secret_access_key: process.env.SECRET_ACCESS_KEY,
    server : 'https://your-endpoint'
});

//设置桶配额为100MB
obsClient.setBucketQuota({
    Bucket : 'bucketname',
```



```
    StorageQuota: 1024 * 1024 * 100
}, (err, result) => {
    if(err){
        console.error('Error-->' + err);
    }else{
        console.log('Status-->' + result.CommonMsg.Status);
    }
});
```

说明

- 使用**StorageQuota参数**指定桶的配额大小。
- 桶配额值必须为非负整数，单位为字节，支持的最大值为2^{63} - 1。

获取桶配额

您可以通过ObsClient.getBucketQuota获取桶配额。以下代码展示如何获取桶配额：

```
// 引入obs库
// 使用npm安装
var ObsClient = require('esdk-obs-nodejs');
// 使用源码安装
// var ObsClient = require('./lib/obs');

// 创建ObsClient实例
var obsClient = new ObsClient({
    //推荐通过环境变量获取AKSK，这里也可以使用其他外部引入方式传入，如果使用硬编码可能会存在泄露风险。
    //您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    access_key_id: process.env.ACCESS_KEY_ID,
    secret_access_key: process.env.SECRET_ACCESS_KEY,
    server : 'https://your-endpoint'
});

obsClient.getBucketQuota({
    Bucket : 'bucketname'
}, (err, result) => {
    if(err){
        console.error('Error-->' + err);
    }else{
        console.log('Status-->' + result.CommonMsg.Status);
        if(result.CommonMsg.Status < 300 && result.InterfaceResult){
            console.log('StorageQuota-->' + result.InterfaceResult.StorageQuota);
        }
    }
});
```

5.11 桶存储类型

须知

开发过程中，您有任何问题可以在github上**提交issue**。**接口参考文档**详细介绍了每个接口的参数和使用方法。

OBS允许您对桶配置不同的存储类型，桶中对象的存储类型默认将与桶的存储类型保持一致。不同的存储类型可以满足客户业务对存储性能、成本的不同诉求。桶的存储类型分为三类，见下表：

类型	说明	OBS Node.js SDK对应值
标准存储	标准存储拥有低访问时延和较高的吞吐量，适用于有大量热点对象（平均一个月多次）或小对象（<1MB），且需要频繁访问数据的业务场景。	ObsClient.enums.StorageClassStandard
低频访问存储	低频访问存储适用于不频繁访问（平均一年少于12次）但在需要时也要求能够快速访问数据的业务场景。	ObsClient.enums.StorageClassWarm
归档存储	归档存储适用于很少访问（平均一年访问一次）数据的业务场景。	ObsClient.enums.StorageClassCold

更多关于桶存储类型的内容请参考**桶的存储类别**。

设置桶存储类型

您可以通过ObsClient.setBucketStoragePolicy设置桶存储类型。以下代码展示如何设置桶存储类型：

```
// 引入obs库
// 使用npm安装
var ObsClient = require('esdk-obs-nodejs');
// 使用源码安装
// var ObsClient = require('./lib/obs');

// 创建ObsClient实例
var obsClient = new ObsClient({
    //推荐通过环境变量获取AKSK，这里也可以使用其他外部引入方式传入，如果使用硬编码可能会存在泄露风险。
    //您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    access_key_id: process.env.ACCESS_KEY_ID,
    secret_access_key: process.env.SECRET_ACCESS_KEY,
    server : 'https://your-endpoint'
});

obsClient.setBucketStoragePolicy({
    Bucket : 'bucketname',
    StorageClass: obsClient.enums.StorageClassWarm
}, (err, result) => {
    if(err){
        console.error('Error-->' + err);
    }else{
        console.log('Status-->' + result.CommonMsg.Status);
    }
});
```

说明

使用**StorageClass参数**指定桶的存储类型。

获取桶存储类型

您可以通过ObsClient.getBucketStoragePolicy获取桶存储类型。以下代码展示如何获取桶存储类型：

```
// 引入obs库
// 使用npm安装
```



```
var ObsClient = require('esdk-obs-nodejs');
// 使用源码安装
// var ObsClient = require('./lib/obs');

// 创建ObsClient实例
var obsClient = new ObsClient({
    //推荐通过环境变量获取AKSK，这里也可以使用其他外部引入方式传入，如果使用硬编码可能会存在泄露风险。
    //您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    access_key_id: process.env.ACCESS_KEY_ID,
    secret_access_key: process.env.SECRET_ACCESS_KEY,
    server : 'https://your-endpoint'
});

obsClient.getBucketStoragePolicy({
    Bucket : 'bucketname'
}, (err, result) => {
    if(err){
        console.error('Error-->' + err);
    }else{
        console.log('Status-->' + result.CommonMsg.Status);
        if(result.CommonMsg.Status < 300 && result.InterfaceResult){
            console.log('StorageClass-->' + result.InterfaceResult.StorageClass);
        }
    }
});
```


6 上传对象

6.1 对象上传简介

须知

开发过程中，您有任何问题可以在github上**提交issue**。**接口参考文档**详细介绍了每个接口的参数和使用方法。

在OBS中，用户操作的基本数据单元是对象。OBS Node.js SDK提供了丰富的对象上传接口，可以通过以下方式上传对象：

- **文本上传**
- **流式上传**
- **文件上传**
- **分段上传**
- **追加上传**
- **断点续传上传**
- **基于表单上传**

SDK支持上传0KB~5GB的对象。文本上传、流式上传、文件上传和追加上传每次上传内容大小不能超过5GB；当上传较大文件时，请使用分段上传，分段上传每段内容大小不能超过5GB；基于表单上传提供了基于浏览器表单上传对象的方式。

如果上传的对象权限设置为匿名用户读取权限，对象上传成功后，匿名用户可通过链接地址访问该对象数据。对象链接地址格式为：https://*桶名.域名/文件夹目录层级/对象名*。如果该对象存在于桶的根目录下，则链接地址将不需要有文件夹目录层级。

6.2 文本上传

须知

开发过程中，您有任何问题可以在github上**提交issue**。**接口参考文档**详细介绍了每个接口的参数和使用方法。

文本上传用于直接上传字符串。您可以通过ObsClient.putObject直接上传字符串到OBS。以下代码展示了如何进行文本上传：

```
// 引入obs库
// 使用npm安装
var ObsClient = require('esdk-obs-nodejs');
// 使用源码安装
// var ObsClient = require('./lib/obs');

// 创建ObsClient实例
var obsClient = new ObsClient({
    //推荐通过环境变量获取AKSK，这里也可以使用其他外部引入方式传入，如果使用硬编码可能会存在泄露风险。
    //您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    access_key_id: process.env.ACCESS_KEY_ID,
    secret_access_key: process.env.SECRET_ACCESS_KEY,
    server : 'https://your-endpoint'
});

obsClient.putObject({
    Bucket : 'bucketname',
    Key : 'objectname',
    Body : 'Hello OBS'
}, (err, result) => {
    if(err){
        console.error('Error-->' + err);
    }else{
        console.log('Status-->' + result.CommonMsg.Status);
    }
});
```

说明

- 使用**Body参数**指定待上传的字符串。
- 上传内容大小不能超过5GB。

6.3 流式上传

须知

开发过程中，您有任何问题可以在github上**提交issue**。**接口参考文档**详细介绍了每个接口的参数和使用方法。

流式上传使用stream.Readable作为对象的数据源。以下代码展示了如何进行流式上传：

上传网络流

```
// 引入obs库
// 使用npm安装
const ObsClient = require('esdk-obs-nodejs');
// 使用源码安装
// const ObsClient = require('./lib/obs');

// 创建ObsClient实例
var obsClient = new ObsClient({
        //推荐通过环境变量获取AKSK，这里也可以使用其他外部引入方式传入，如果使用硬编码可能会存在泄露风险。
        //您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
        access_key_id: process.env.ACCESS_KEY_ID,
        secret_access_key: process.env.SECRET_ACCESS_KEY,
        server : 'https://your-endpoint'
});

// 定义网络流URL。
const url = 'https://www.example.com'
// 引入http库、https库。
const http = require('http');
const https = require('https');
// 根据URL选择对应的请求库。
const request = url.startsWith('http') ? http : https;
// 获取网络流。
request.get(url, (res) => {
    if (res.statusCode === 200) {
        // 获取网络流成功后调用putObject方法。
        obsClient.putObject({
            // Bucket名称，例如examplebucket。
            Bucket: 'examplebucket',
            // 填写Object完整路径，例如exampledir/exampleobject.txt。Object完整路径中不能包含Bucket名称。
            Key: 'exampledir/exampleobject.txt',
            // res 是http.IncomingMessage类的实例，是一个可读流。
            Body: res
        }, (err, result) => {
            if (err) {
                console.error('Error-->' + err);
                // 这里处理网络异常或者其他异常。
            } else {
                console.log('Status-->' + result.CommonMsg.Status);
                // 这里处理业务代码。
            }
        });
    }
});
```

上传文件流

```
// 引入obs库
// 使用npm安装
const ObsClient = require('esdk-obs-nodejs');
// 使用源码安装
// const ObsClient = require('./lib/obs');

// 创建ObsClient实例
var obsClient = new ObsClient({
        //推荐通过环境变量获取AKSK，这里也可以使用其他外部引入方式传入，如果使用硬编码可能会存在泄露风险。
        //您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
        access_key_id: process.env.ACCESS_KEY_ID,
        secret_access_key: process.env.SECRET_ACCESS_KEY,
        server : 'https://your-endpoint'
});

const fs = require('fs');
```



```
// 填写本地文件的完整路径，从本地文件中读取数据流。
// 如果本地文件的完整路径中未指定本地路径，则默认从示例程序所属项目对应本地路径中上传文件。
const stream = fs.createReadStream('D:\\localpath\\examplefile.txt');
obsClient.putObject({
    // Bucket名称，例如examplebucket。
    Bucket: 'examplebucket',
    // 填写Object完整路径，例如exampledir/exampleobject.txt。Object完整路径中不能包含Bucket名称。
    Key: 'exampledir/exampleobject.txt',
    // stream 是一个文件可读流。
    Body: stream
}, (err, result) => {
      if(err){
            console.error('Error-->' + err);
      }else{
            console.log('Status-->' + result.CommonMsg.Status);
      }
});
```

须知

- 使用**Body参数**指定待上传的流数据时，其值必须是一个stream.Readable实例。
- 大文件上传建议使用**分段上传**。
- 上传内容大小不能超过5GB。

6.4 文件上传

须知

开发过程中，您有任何问题可以在github上**提交issue**。**接口参考文档**详细介绍了每个接口的参数和使用方法。

文件上传使用本地文件作为对象的数据源。以下代码展示了如何进行文件上传：

```
// 引入obs库
// 使用npm安装
var ObsClient = require('esdk-obs-nodejs');
// 使用源码安装
// var ObsClient = require('./lib/obs');

// 创建ObsClient实例
var obsClient = new ObsClient({
      //推荐通过环境变量获取AKSK，这里也可以使用其他外部引入方式传入，如果使用硬编码可能会存在泄露风险。
      //您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
      access_key_id: process.env.ACCESS_KEY_ID,
      secret_access_key: process.env.SECRET_ACCESS_KEY,
      server : 'https://your-endpoint'
});

obsClient.putObject({
      Bucket : 'bucketname',
      Key : 'objectname',
      SourceFile : 'localfile'  // localfile为待上传的本地文件路径，需要指定到具体的文件名
}, (err, result) => {
      if(err){
            console.error('Error-->' + err);
      }else{
```

```
        console.log('Status-->' + result.CommonMsg.Status);
    }
});
```

说明

- 使用**SourceFile参数**指定待上传的文件路径。
- **SourceFile参数**和**Body参数**不能同时使用。
- 上传内容大小不能超过5GB。

6.5 创建文件夹

须知

开发过程中，您有任何问题可以在github上**提交issue**。**接口参考文档**详细介绍了每个接口的参数和使用方法。

OBS本身是没有文件夹的概念的，桶中存储的元素只有对象。创建文件夹实际上是创建了一个大小为0且对象名以“/”结尾的对象，这类对象与其他对象无任何差异，可以进行下载、删除等操作，只是OBS控制台会将这类以“/”结尾的对象以文件夹的方式展示。

```
// 引入obs库
// 使用npm安装
var ObsClient = require('esdk-obs-nodejs');
// 使用源码安装
// var ObsClient = require('./lib/obs');

// 创建ObsClient实例
var obsClient = new ObsClient({
    //推荐通过环境变量获取AKSK，这里也可以使用其他外部引入方式传入，如果使用硬编码可能会存在泄露风险。
    //您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    access_key_id: process.env.ACCESS_KEY_ID,
    secret_access_key: process.env.SECRET_ACCESS_KEY,
    server : 'https://your-endpoint'
});

obsClient.putObject({
    Bucket : 'bucketname',
    Key : 'parent_directory/'
}, (err, result) => {
    if(err){
        console.error('Error-->' + err);
    }else{
        console.log('Status-->' + result.CommonMsg.Status);
    }
});
// 在文件夹下创建对象
obsClient.putObject({
    Bucket : 'bucketname',
    Key : 'parent_directory/objectname'
}, (err, result) => {
    if(err){
        console.error('Error-->' + err);
    }else{
        console.log('Status-->' + result.CommonMsg.Status);
    }
});
```


说明

- 创建文件夹本质上来说是创建了一个大小为0且对象名以“/”结尾的对象。
- 多级文件夹创建最后一级即可，比如src1/src2/src3/，创建src1/src2/src3/即可，无需创建src1/、src1/src2/。

6.6 设置对象属性

须知

开发过程中，您有任何问题可以在github上**提交issue**。**接口参考文档**详细介绍了每个接口的参数和使用方法。

您可以在上传对象时设置对象属性。对象属性包含对象长度、对象MIME类型、对象MD5值（用于校验）、对象存储类型、对象自定义元数据。对象属性可以在多种上传方式下（文本上传、流式上传、文件上传、分段上传），或**复制对象**时进行设置。

对象属性详细说明见下表：

名称	描述	默认值
对象长度（Content-Length）	上传对象的长度，超过流/文件的长度会截断。	流/文件实际长度
对象MIME类型（Content-Type）	对象的MIME类型，定义对象的类型及网页编码，决定浏览器将以什么形式、什么编码读取对象。	binary/octet-stream
对象MD5值（Content-MD5）	对象数据的MD5值（经过Base64编码），提供给OBS服务端，校验数据完整性。	无
对象存储类型	对象的存储类型，不同的存储类型可以满足客户业务对存储性能、成本的不同诉求。默认与桶的存储类型保持一致，可以设置为与桶的存储类型不同。	无
对象自定义元数据	用户对上传到桶中对象的自定义属性描述，以便对对象进行自定义管理。	无

设置对象长度

```
// 引入obs库
// 使用npm安装
var ObsClient = require('esdk-obs-nodejs');
// 使用源码安装
// var ObsClient = require('./lib/obs');

// 创建ObsClient实例
```

```
var obsClient = new ObsClient({
    //推荐通过环境变量获取AKSK，这里也可以使用其他外部引入方式传入，如果使用硬编码可能会存在泄露风险。
    //您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    access_key_id: process.env.ACCESS_KEY_ID,
    secret_access_key: process.env.SECRET_ACCESS_KEY,
    server : 'https://your-endpoint'
});

obsClient.putObject({
    Bucket : 'bucketname',
    Key : 'objectname',
    SourceFile : 'localfile',
    ContentLength : 1024 * 1024 // 1MB
}, (err, result) => {
    if(err){
        console.error('Error-->' + err);
    }else{
        console.log('Status-->' + result.CommonMsg.Status);
    }
});
```

说明

使用**ContentLength参数**指定对象长度。

设置对象 MIME 类型

```
// 引入obs库
// 使用npm安装
var ObsClient = require('esdk-obs-nodejs');
// 使用源码安装
// var ObsClient = require('./lib/obs');

// 创建ObsClient实例
var obsClient = new ObsClient({
    //推荐通过环境变量获取AKSK，这里也可以使用其他外部引入方式传入，如果使用硬编码可能会存在泄露风险。
    //您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    access_key_id: process.env.ACCESS_KEY_ID,
    secret_access_key: process.env.SECRET_ACCESS_KEY,
    server : 'https://your-endpoint'
});

// 上传图片
obsClient.putObject({
    Bucket : 'bucketname',
    Key : 'objectname.jpg',
    SourceFile : 'localimage.jpg',
    ContentType : 'image/jpeg'
}, (err, result) => {
    if(err){
        console.error('Error-->' + err);
    }else{
        console.log('Status-->' + result.CommonMsg.Status);
    }
});
```

说明

- 使用**ContentType参数**指定对象MIME类型。
- 如果不设置对象MIME类型，SDK会根据上传对象的后缀名自动判断对象MIME类型，如.xml判断为application/xml文件；.html判断为text/html文件。

设置对象 MD5 值

```
// 引入obs库
// 使用npm安装
var ObsClient = require('esdk-obs-nodejs');
// 使用源码安装
// var ObsClient = require('./lib/obs');

// 创建ObsClient实例
var obsClient = new ObsClient({
    //推荐通过环境变量获取AKSK，这里也可以使用其他外部引入方式传入，如果使用硬编码可能会存在泄露风险。
    //您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    access_key_id: process.env.ACCESS_KEY_ID,
    secret_access_key: process.env.SECRET_ACCESS_KEY,
    server : 'https://your-endpoint'
});

obsClient.putObject({
    Bucket : 'bucketname',
    Key : 'objectname',
    SourceFile : 'localimage.jpg',
    ContentMD5 : 'your md5 which should be encoded by base64'
}, (err, result) => {
    if(err){
        console.error('Error-->' + err);
    }else{
        console.log('Status-->' + result.CommonMsg.Status);
    }
});
```

说明

- 使用**ContentMD5参数**指定对象MD5值。
- 对象数据的MD5值必须经过Base64编码。
- OBS服务端会将该MD5值与对象数据计算出的MD5值进行对比，如果不匹配则上传失败，返回HTTP 400错误。
- 如果不设置对象的MD5值，OBS服务端会忽略对对象数据的MD5值校验。

设置对象存储类型

```
// 引入obs库
// 使用npm安装
var ObsClient = require('esdk-obs-nodejs');
// 使用源码安装
// var ObsClient = require('./lib/obs');

// 创建ObsClient实例
var obsClient = new ObsClient({
    //推荐通过环境变量获取AKSK，这里也可以使用其他外部引入方式传入，如果使用硬编码可能会存在泄露风险。
    //您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    access_key_id: process.env.ACCESS_KEY_ID,
    secret_access_key: process.env.SECRET_ACCESS_KEY,
    server : 'https://your-endpoint'
});

obsClient.putObject({
    Bucket : 'bucketname',
    Key : 'objectname',
    SourceFile : 'localfile',
    // 设置对象存储类型为归档存储

    StorageClass : ObsClient.enums.StorageClassCold
}, (err, result) => {
```



```
    if(err){
        console.error('Error-->' + err);
    }else{
        console.log('Status-->' + result.CommonMsg.Status);
    }
});
```

说明

- 使用**StorageClass参数**指定对象的存储类型。
- 如果不设置，对象的存储类型默认与桶的存储类型保持一致。
- 对象的存储类型分为三类，其含义与**桶存储类型**一致。
- 下载归档存储类型的对象前必须将其恢复。

设置对象自定义元数据

```
// 引入obs库
// 使用npm安装
var ObsClient = require('esdk-obs-nodejs');
// 使用源码安装
// var ObsClient = require('./lib/obs');

// 创建ObsClient实例
var obsClient = new ObsClient({
    //推荐通过环境变量获取AKSK，这里也可以使用其他外部引入方式传入，如果使用硬编码可能会存在泄露风险。
    //您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    access_key_id: process.env.ACCESS_KEY_ID,
    secret_access_key: process.env.SECRET_ACCESS_KEY,
    server : 'https://your-endpoint'
});

obsClient.putObject({
    Bucket : 'bucketname',
    Key : 'objectname',
    SourceFile : 'localfile',
    Metadata : {'property1':'property-value1', 'property2' : 'property-value2'},
}, (err, result) => {
    if(err){
        console.error('Error-->' + err);
    }else{
        console.log('Status-->' + result.CommonMsg.Status);
    }
});
```

说明

- 使用**Metadata参数**指定对象自定义元数据。
- 在上面设置对象自定义元数据示例代码中，用户自定义了一个名称为“property1”，值为“property-value1”的元数据和一个名称为“property2”，值为“property-value2”的元数据。
- 一个对象可以有多个元数据，总大小不能超过8KB。
- 对象的自定义元数据可以通过ObsClient.getObjectMetadata获取，参见**获取对象元数据**。
- 使用ObsClient.getObject下载对象时，对象的自定义元数据也会同时下载。

6.7 分段上传

须知

开发过程中，您有任何问题可以在github上**提交issue**。**接口参考文档**详细介绍了每个接口的参数和使用方法。

对于较大文件上传，可以切分成段上传。用户可以在如下的应用场景内（但不仅限于此），使用分段上传的模式：

- 上传超过100MB大小的文件。
- 网络条件较差，和OBS服务端之间的链接经常断开。
- 上传前无法确定将要上传文件的大小。

分段上传分为如下3个步骤：

步骤1 初始化分段上传任务（ObsClient.initiateMultipartUpload）。

步骤2 逐个或并行上传段（ObsClient.uploadPart）。

步骤3 合并段（ObsClient.completeMultipartUpload）或取消分段上传任务(ObsClient.abortMultipartUpload)。

----结束

初始化分段上传任务

使用分段上传方式传输数据前，必须先通知OBS初始化一个分段上传任务。该操作会返回一个OBS服务端创建的全局唯一标识（Upload ID），用于标识本次分段上传任务。您可以根据这个唯一标识来发起相关的操作，如取消分段上传任务、列举分段上传任务、列举已上传的段等。

您可以通过ObsClient.initiateMultipartUpload初始化一个分段上传任务：

```
// 引入obs库
// 使用npm安装
var ObsClient = require('esdk-obs-nodejs');
// 使用源码安装
// var ObsClient = require('./lib/obs');

// 创建ObsClient实例
var obsClient = new ObsClient({
    //推荐通过环境变量获取AKSK，这里也可以使用其他外部引入方式传入，如果使用硬编码可能会存在泄露风险。
    //您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    access_key_id: process.env.ACCESS_KEY_ID,
    secret_access_key: process.env.SECRET_ACCESS_KEY,
    server : 'https://your-endpoint'
});

obsClient.initiateMultipartUpload({
    Bucket : 'bucketname',
    Key : 'objectname',
    ContentType : 'text/plain',
```

```
    Metadata : {'property' : 'property-value'}
}, (err, result) => {
    if(err){
        console.error('Error-->' + err);
    }else{
        console.log('Status-->' + result.CommonMsg.Status);
        if(result.CommonMsg.Status < 300 && result.InterfaceResult){
            console.log('UploadId-->' + result.InterfaceResult.UploadId);
        }
    }
});
```

说明

- 初始化分段上传任务时，除了指定上传对象的名称和所属桶外，您还可以使用**ContentType参数**和**Metadata参数**分别指定对象MIME类型和对象自定义元数据。
- 调用初始化分段上传任务接口成功后，会返回分段上传任务的全局唯一标识（Upload ID），在后面的操作中将用到它。

上传段

初始化一个分段上传任务之后，可以根据指定的对象名和Upload ID来分段上传数据。每一个上传的段都有一个标识它的号码——分段号（Part Number，范围是1~10000）。对于同一个Upload ID，该分段号不但唯一标识这一段数据，也标识了这段数据在整个对象内的相对位置。如果您用同一个分段号上传了新的数据，那么OBS上已有的这个段号的数据将被覆盖。**除了最后一段以外，其他段的大小范围是100KB~5GB**；最后段大小范围是0~5GB。每个段不需要按顺序上传，甚至可以在不同进程、不同机器上上传，OBS会按照分段号排序组成最终对象。

您可以通过ObsClient.uploadPart上传段：

```
// 引入obs库
// 使用npm安装
var ObsClient = require('esdk-obs-nodejs');
// 使用源码安装
// var ObsClient = require('./lib/obs');

// 创建ObsClient实例
var obsClient = new ObsClient({
    //推荐通过环境变量获取AKSK，这里也可以使用其他外部引入方式传入，如果使用硬编码可能会存在泄露风险。
    //您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    access_key_id: process.env.ACCESS_KEY_ID,
    secret_access_key: process.env.SECRET_ACCESS_KEY,
    server : 'https://your-endpoint'
});

obsClient.uploadPart({
    Bucket:'bucketname',
    Key:'objectname',
    // 设置分段号，范围是1~10000
    PartNumber:1,
    // 设置Upload ID
    UploadId:'upload id from initiateMultipartUpload',
    // 设置将要上传的大文件，其中localfile为待上传的本地文件路径，需要指定到具体的文件名
    SourceFile: 'localfile',
    // 设置分段大小
    PartSize: 5 * 1024 * 1024,
    // 设置分段的起始偏移大小
    Offset: 0
}, (err, result) => {
    if(err){
        console.log('Error-->' + err);
    }else{
```



```
        console.log('Status-->' + result.CommonMsg.Status);
        if(result.CommonMsg.Status < 300 && result.InterfaceResult){
            console.log('ETag-->' + result.InterfaceResult.ETag);
        }
    }
});
```

说明

- 使用**PartNumber参数**指定分段号；使用**UploadId参数**指定分段上传任务的全局唯一标识；使用**SourceFile参数**指定待上传的文件；使用**PartSize参数**指定分段大小；使用**Offset参数**指定待上传文件的起始偏移大小。
- 上传段接口要求除最后一段以外，其他的段大小都要大于100KB。但是上传段接口并不会立即校验上传段的大小（因为不知道是否为最后一块）；只有调用合并段接口时才会校验。
- OBS会将服务端收到段数据的ETag值（段数据的MD5值）返回给用户。
- 可以通过**ContentMD5参数**设置上传数据的MD5值，提供给OBS服务端用于校验数据完整性。
- 分段号的范围是1~10000。如果超出这个范围，OBS将返回400 Bad Request错误。
- OBS 3.0的桶支持最小段的大小为100KB，OBS 2.0的桶支持最小段的大小为5MB。请在OBS 3.0的桶上执行分段上传操作。

合并段

所有分段上传完成后，需要调用合并段接口来在OBS服务端生成最终对象。在执行该操作时，需要提供所有有效的分段列表（包括分段号和分段ETag值）；OBS收到提交的分段列表后，会逐一验证每个段的有效性。当所有段验证通过后，OBS将把这些分段组合成最终的对象。

您可以通过ObsClient.completeMultipartUpload合并段：

```
// 引入obs库
// 使用npm安装
var ObsClient = require('esdk-obs-nodejs');
// 使用源码安装
// var ObsClient = require('./lib/obs');

// 创建ObsClient实例
var obsClient = new ObsClient({
    //推荐通过环境变量获取AKSK，这里也可以使用其他外部引入方式传入，如果使用硬编码可能会存在泄露风险。
    //您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    access_key_id: process.env.ACCESS_KEY_ID,
    secret_access_key: process.env.SECRET_ACCESS_KEY,
    server : 'https://your-endpoint'
});

obsClient.completeMultipartUpload({
    Bucket:'bucketname',
    Key:'objectname',
    // 设置Upload ID
    UploadId:'upload id from initiateMultipartUpload',
    Parts: [{'PartNumber':1,'ETag':'etag value from uploadPart'}]
}, (err, result) => {
    if(err){
        console.log('Error-->' + err);
    }else{
        console.log('Status-->' + result.CommonMsg.Status);
    }
});
```


⚠ 注意

- 如果最后一个段之外的其它段尺寸过小（小于100KB），OBS返回400 Bad Request。

说明

- 使用**UploadId参数**指定分段上传任务的全局唯一标识；使用**Parts参数**指定分段号与分段ETag值的列表，该列表必须按分段号升序排列。
- 分段可以是不连续的。

并发分段上传

分段上传的主要目的是解决大文件上传或网络条件较差的情况。下面的示例代码展示了如何使用分段上传并发上传大文件：

```
// 引入obs库
// 使用npm安装
const ObsClient = require('esdk-obs-nodejs');
// 使用源码安装
// var ObsClient = require('./lib/obs');
const fs = require('fs');

// 创建ObsClient实例
var obsClient = new ObsClient({
        //推荐通过环境变量获取AKSK，这里也可以使用其他外部引入方式传入，如果使用硬编码可能会存在泄露风险。
        //您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
        access_key_id: process.env.ACCESS_KEY_ID,
        secret_access_key: process.env.SECRET_ACCESS_KEY,
        server : 'https://your-endpoint'
});

const Bucket = 'bucketname'
const Key = 'objectname'

// 指定分段大小
const DEFAULT_PART_SIZE = 9 * 1024 * 1024;
// 指定并发数为20
const CONCURRENCY = 20

// 准备分段上传的参数
const preparePartParams = (Bucket, Key, UploadId) => (sampleFile, partSize = DEFAULT_PART_SIZE) => {
    try {
        const fileSize = fs.lstatSync(sampleFile).size;
        const partCount = fileSize % partSize === 0 ? Math.floor(fileSize / partSize) : Math.floor(fileSize / partSize) + 1;

        const uploadPartParams = [];
        // 指定并发上传
        for (let i = 0; i < partCount; i++) {
            // 分段在文件中的起始位置
            let Offset = i * partSize;
            // 分段大小
            let currPartSize = (i + 1 === partCount) ? fileSize - Offset : partSize;
            // 分段号
            let PartNumber = i + 1;
            uploadPartParams.push({
                Bucket,
                Key,
                PartNumber,
                UploadId,
                Offset,
                SourceFile: sampleFile,
```



```
                PartSize: currPartSize,
            });
        }

        return ({ uploadPartParams, fileSize });
    } catch (error) {
        console.log(error)
    }
}


/**
 * uploadSuccessSize：已经上传成功的大小
 * uploadSuccessCount：已经上传成功的段数量
 * concurrency：当前并发数
 */
let uploadSuccessSize = 0;
let uploadSuccessCount = 0;
let concurrency = 0

const parts = [];

const uploadPart = (uploadPartParam, otherUploadPartInfo) => {
    const partCount = otherUploadPartInfo.partCount;
    const fileSize = otherUploadPartInfo.fileSize;
    concurrency++;
    return obsClient
        .uploadPart(uploadPartParam)
        .then(result => {
            const { PartNumber, PartSize } = uploadPartParam;
            if (result.CommonMsg.Status < 300) {
                uploadSuccessCount++;
                uploadSuccessSize += PartSize;
                console.log(`the current concurrent count is ${concurrency} | uploaded segment: $
{uploadSuccessCount}/${partCount}. the progress is ${((uploadSuccessSize / fileSize) * 100).toFixed(2)}% |
the partNumber ${PartNumber} upload successed.`)
                parts.push({ PartNumber, ETag: result.InterfaceResult.ETag });
            } else {
                console.log(result.CommonMsg.Code, parts)
            }
            concurrency--;
        }).catch(function (err) {
            console.log(err);
            throw err;
        })
}

const uploadFile = (sourceFile) => {
    obsClient.initiateMultipartUpload({
        Bucket,
        Key
    }).then(res => {
        const Status = res.CommonMsg.Status;
        const UploadId = res.InterfaceResult.UploadId;

        if (typeof Status === 'number' && Status > 300) {
            console.log(`initiateMultipartUpload failed! Status:${Status}`);
            return;
        }

        const partParams = preparePartParams(Bucket, Key, UploadId)(sourceFile)
        const uploadPartParams = partParams.uploadPartParams;
        const fileSize = partParams.fileSize;
        const partCount = uploadPartParams.length;
        const otherUploadPartInfo = { fileSize, partCount }

        // 调用并行上传函数
        parallelFunc(uploadPartParams, (param) => uploadPart(param, otherUploadPartInfo),
CONCURRENCY)
```

```
            .then(() => {
                obsClient.completeMultipartUpload({
                    Bucket,
                    Key,
                    UploadId,
                    Parts: parts.sort((a, b) => a.PartNumber - b.PartNumber)
                }, (err, result) => {
                    if (err) {
                        console.log('Error-->' + err);
                    } else {
                        console.log('Status-->' + result.CommonMsg.Status);
                    }
                });
            })

    }).catch(function (err) {
        console.log(err)
    })
}


/**
 * 实现函数并行执行
 * @param {Array} params 回调函数的参数数组
 * @param {Promise} promiseFn 回调函数
 * @param {number} limit 并行数
 */
const parallelFunc = (params, promiseFn, limit) => {
    return new Promise((resolve) => {
        let concurrency = 0;
        let finished = 0;
        const count = params.length;
        const run = (param) => {
            concurrency++;
            promiseFn(param)
                .then(() => {
                    concurrency--;
                    drainQueue();
                    finished++
                    if (finished === count) {
                        resolve()
                    }
                });
        }
        const drainQueue = () => {
            while (params.length > 0 && concurrency < limit) {
                var param = params.shift();
                run(param);
            }
        }
        drainQueue();
    })

}

uploadFile('localfile');
```

说明

大文件分段上传时，使用**Offset参数**和**PartSize参数**配合指定每段数据在文件中的起始结束位置。

注意

并发数过大，可能会因为网络不稳定等原因，产生Timeout错误，需要限制并发数。

取消分段上传任务

分段上传任务可以被取消，当一个分段上传任务被取消后，就不能再使用其Upload ID做任何操作，已经上传段也会被OBS删除。

采用分段上传方式上传对象过程中或上传对象失败后会在桶内产生段，这些段会占用您的存储空间，您可以通过取消该分段上传任务来清理掉不需要的段，节约存储空间。

您可以通过ObsClient.abortMultipartUpload取消分段上传任务：

```
// 引入obs库
// 使用npm安装
var ObsClient = require('esdk-obs-nodejs');
// 使用源码安装
// var ObsClient = require('./lib/obs');

// 创建ObsClient实例
var obsClient = new ObsClient({
    //推荐通过环境变量获取AKSK，这里也可以使用其他外部引入方式传入，如果使用硬编码可能会存在泄露风险。
    //您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    access_key_id: process.env.ACCESS_KEY_ID,
    secret_access_key: process.env.SECRET_ACCESS_KEY,
    server : 'https://your-endpoint'
});

obsClient.abortMultipartUpload({
    Bucket:'bucketname',
    Key:'objectname',
    // 设置Upload ID
    UploadId:'upload id from initiateMultipartUpload',
}, (err, result) => {
    if(err){
        console.log('Error-->' + err);
    }else{
        console.log('Status-->' + result.CommonMsg.Status);
    }
});
```

列举已上传的段

您可使用ObsClient.listParts列举出某一分段上传任务所有已经上传成功的段。

该接口可设置的参数如下：

参数	作用
UploadId	分段上传任务全局唯一标识，从ObsClient.initiateMultipartUpload返回的结果获取。
MaxParts	表示列举已上传段的返回结果最大段数目，即分页时每一页中段数目。
PartNumberMarker	表示列举已上传段的起始位置，只有Part Number大于该参数的段会被列出。

- 简单列举

```
// 引入obs库
// 使用npm安装
```



```
var ObsClient = require('esdk-obs-nodejs');
// 使用源码安装
// var ObsClient = require('./lib/obs');

// 创建ObsClient实例
var obsClient = new ObsClient({
    //推荐通过环境变量获取AKSK，这里也可以使用其他外部引入方式传入，如果使用硬编码可能会存在泄露风险。
    //您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    access_key_id: process.env.ACCESS_KEY_ID,
    secret_access_key: process.env.SECRET_ACCESS_KEY,
    server : 'https://your-endpoint'
});

// 列举已上传的段，其中uploadId来自于initiateMultipartUpload
obsClient.listParts({
    Bucket : 'bucketname',
    Key: 'objectname',
    UploadId : 'upload id from initiateMultipartUpload'
}, (err, result) => {
    if(err){
        console.log('Error-->' + err);
    }else{
        console.log('Status-->' + result.CommonMsg.Status);
        if(result.CommonMsg.Status < 300 && result.InterfaceResult){
            for(let i=0;i<result.InterfaceResult.Parts.length;i++){
                console.log('Part['+ i +']:');
                // 分段号，上传时候指定
                console.log('PartNumber-->' + result.InterfaceResult.Parts[i]['PartNumber']);
                // 段的最后上传时间
                console.log('LastModified-->' + result.InterfaceResult.Parts[i]['LastModified']);
                // 分段的ETag值
                console.log('ETag-->' + result.InterfaceResult.Parts[i]['ETag']);
                // 段数据大小
                console.log('Size-->' + result.InterfaceResult.Parts[i]['Size']);
            }
        }
    }
});
```

说明

- 列举段至多返回1000个段信息，如果指定的Upload ID包含的段数量大于1000，则返回结果中InterfaceResult.IsTruncated为true表明本次没有返回全部段，并可通过InterfaceResult.NextPartNumberMarker获取下次列举的起始位置。
- 如果想获取指定Upload ID包含的所有分段，可以采用分页列举的方式。

- 列举所有段

由于ObsClient.listParts只能列举至多1000个段，如果段数量大于1000，列举所有分段请参考如下示例：

```
// 引入obs库
// 使用npm安装
var ObsClient = require('esdk-obs-nodejs');
// 使用源码安装
// var ObsClient = require('./lib/obs');

// 创建ObsClient实例
var obsClient = new ObsClient({
    //推荐通过环境变量获取AKSK，这里也可以使用其他外部引入方式传入，如果使用硬编码可能会存在泄露风险。
    //您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    access_key_id: process.env.ACCESS_KEY_ID,
    secret_access_key: process.env.SECRET_ACCESS_KEY,
    server : 'https://your-endpoint'
});
```

```
var listAll = (partNumberMarker) => {
    // 列举已上传的段，其中uploadId来自于initiateMultipartUpload
    obsClient.listParts({
        Bucket : 'bucketname',
        Key: 'objectname',
        UploadId : 'upload id from initiateMultipartUpload',
        PartNumberMarker : partNumberMarker
    }, (err, result) => {
        if(err){
            console.log('Error-->' + err);
        }else{
            console.log('Status-->' + result.CommonMsg.Status);
            if(result.CommonMsg.Status < 300 && result.InterfaceResult){
                for(let i=0;i<result.InterfaceResult.Parts.length;i++){
                    console.log('Part['+ i +']:');
                    // 分段号，上传时候指定
                    console.log('PartNumber-->' + result.InterfaceResult.Parts[i]['PartNumber']);
                    // 段的最后上传时间
                    console.log('LastModified-->' + result.InterfaceResult.Parts[i]['LastModified']);
                    // 分段的ETag值
                    console.log('ETag-->' + result.InterfaceResult.Parts[i]['ETag']);
                    // 段数据大小
                    console.log('Size-->' + result.InterfaceResult.Parts[i]['Size']);
                }
                if(result.InterfaceResult.IsTruncated === 'true'){
                    listAll(result.InterfaceResult.NextPartNumberMarker);
                }
            }
        }
    });
};

listAll();
```

列举分段上传任务

您可以通过ObsClient.listMultipartUploads列举分段上传任务。列举分段上传任务可设置的参数如下：

参数	作用
Prefix	限定返回的分段上传任务中的对象名必须带有Prefix前缀。
Delimiter	用于对分段上传任务中的对象名进行分组的字符。对于对象名中包含Delimiter的任务，其对象名（如果请求中指定了Prefix，则此处的对象名需要去掉Prefix）中从首字符至第一个Delimiter之间的字符串将作为一个分组并作为CommonPrefix返回。
MaxUploads	列举分段上传任务的最大数目，取值范围为1~1000，当超出范围时，按照默认的1000进行处理。
KeyMarker	表示列举时返回指定的KeyMarker之后的分段上传任务。
UploadIdMarker	只有与KeyMarker参数一起使用时才有意义，用于指定返回结果的起始位置，即列举时返回指定KeyMarker的UploadIdMarker之后的分段上传任务。

- 简单列举分段上传任务

```
// 引入obs库
// 使用npm安装
```



```
var ObsClient = require('esdk-obs-nodejs');
// 使用源码安装
// var ObsClient = require('./lib/obs');

// 创建ObsClient实例
var obsClient = new ObsClient({
    //推荐通过环境变量获取AKSK，这里也可以使用其他外部引入方式传入，如果使用硬编码可能会存在泄露风险。
    //您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    access_key_id: process.env.ACCESS_KEY_ID,
    secret_access_key: process.env.SECRET_ACCESS_KEY,
    server : 'https://your-endpoint'
});

obsClient.listMultipartUploads({
    Bucket : 'bucketname'
}, (err, result) => {
    if(err){
        console.log('Error-->' + err);
    }else{
        console.log('Status-->' + result.CommonMsg.Status);
        if(result.CommonMsg.Status < 300 && result.InterfaceResult){
            for(let i=0;i<result.InterfaceResult.Uploads.length;i++){
                console.log('Uploads[' + i + ']');
                console.log('UploadId-->' + result.InterfaceResult.Uploads[i]['UploadId']);
                console.log('Key-->' + result.InterfaceResult.Uploads[i]['Key']);
                console.log('Initiated-->' + result.InterfaceResult.Uploads[i]['Initiated']);
            }
        }
    }
});
```

说明

- 列举分段上传任务至多返回1000个任务信息，如果指定的桶包含的分段上传任务数量大于1000，则返回结果中InterfaceResult.IsTruncated为true表明本次没有返回全部结果，并可通过InterfaceResult.NextKeyMarker和InterfaceResult.NextUploadIdMarker获取下次列举的起点。
- 如果想获取指定桶包含的所有分段上传任务，可以采用分页列举的方式。

- 列举全部分段上传任务

```
// 引入obs库
// 使用npm安装
var ObsClient = require('esdk-obs-nodejs');
// 使用源码安装
// var ObsClient = require('./lib/obs');

// 创建ObsClient实例
var obsClient = new ObsClient({
    //推荐通过环境变量获取AKSK，这里也可以使用其他外部引入方式传入，如果使用硬编码可能会存在泄露风险。
    //您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    access_key_id: process.env.ACCESS_KEY_ID,
    secret_access_key: process.env.SECRET_ACCESS_KEY,
    server : 'https://your-endpoint'
});

var listAll = (keyMarker, uploadIdMarker) => {
    obsClient.listMultipartUploads({
        Bucket : 'bucketname',
        KeyMarker : keyMarker,
        UploadIdMarker : uploadIdMarker
    }, (err, result) => {
        if(err){
            console.log('Error-->' + err);
        }else{
```



```
                console.log('Status-->' + result.CommonMsg.Status);
                if(result.CommonMsg.Status < 300 && result.InterfaceResult){
                    for(let i=0;i<result.InterfaceResult.Uploads.length;i++){
                        console.log('Uploads[' + i + ']');
                        console.log('UploadId-->' + result.InterfaceResult.Uploads[i]['UploadId']);
                        console.log('Key-->' + result.InterfaceResult.Uploads[i]['Key']);
                        console.log('Initiated-->' + result.InterfaceResult.Uploads[i]['Initiated']);
                    }

                    if(result.InterfaceResult.IsTruncated === 'true'){
                        listAll(result.InterfaceResult.NextKeyMarker,
result.InterfaceResult.NextUploadIdMarker);
                    }
                }
            }
        });
}

listAll();
```

6.8 设置对象生命周期

须知

开发过程中，您有任何问题可以在github上**提交issue**。**接口参考文档**详细介绍了每个接口的参数和使用方法。

上传对象或者初始化分段上传任务时，您可以直接指定对象的过期时间。示例代码如下：

```
// 引入obs库
// 使用npm安装
var ObsClient = require('esdk-obs-nodejs');
// 使用源码安装
// var ObsClient = require('./lib/obs');

// 创建ObsClient实例
var obsClient = new ObsClient({
        //推荐通过环境变量获取AKSK，这里也可以使用其他外部引入方式传入，如果使用硬编码可能会存在泄露风
险。
        //您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/
intl/zh-cn/usermanual-ca/ca_01_0003.html
        access_key_id: process.env.ACCESS_KEY_ID,
        secret_access_key: process.env.SECRET_ACCESS_KEY,
        server : 'https://your-endpoint'
});

// 上传对象时，设置对象30天后过期
obsClient.putObject({
        Bucket : 'bucketname',
        Key : 'objectname',
        Body : 'Hello OBS',
        Expires : 30
}, (err, result) => {
        if(err){
            console.error('Error-->' + err);
        }else{
            console.log('Status-->' + result.CommonMsg.Status);
        }
});

// 初始化分段上传任务时，设置合并段后生成的对象60天后过期
obsClient.initiateMultipartUpload({
```



```
    Bucket : 'bucketname',
    Key : 'objectname',
    ContentType : 'text/plain',
    Expires : 60
}, (err, result) => {
    if(err){
        console.error('Error-->' + err);
    }else{
        console.log('Status-->' + result.CommonMsg.Status);
        if(result.CommonMsg.Status < 300 && result.InterfaceResult){
            console.log('UploadId-->' + result.InterfaceResult.UploadId);
        }
    }
});
```

说明

- 使用**Expires参数**指定对象的过期时间。
- 上述方式仅支持设置以天为单位的对象过期时间，过期后的对象会被OBS服务端自动清理。
- 上述方式设置的对象过期时间，其优先级高于桶生命周期规则。

6.9 追加上传

须知

开发过程中，您有任何问题可以在github上**提交issue**。**接口参考文档**详细介绍了每个接口的参数和使用方法。

追加上传可实现对同一个对象追加数据内容的功能。您可以通过ObsClient.appendObject进行追加上传。示例代码如下：

```
// 引入obs库
// 使用npm安装
var ObsClient = require('esdk-obs-nodejs');
// 使用源码安装
// var ObsClient = require('./lib/obs');

// 创建ObsClient实例
var obsClient = new ObsClient({
    //推荐通过环境变量获取AKSK，这里也可以使用其他外部引入方式传入，如果使用硬编码可能会存在泄露风险。
    //您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    access_key_id: process.env.ACCESS_KEY_ID,
    secret_access_key: process.env.SECRET_ACCESS_KEY,
    server : 'https://your-endpoint'
});

// 第一次追加上传，起始位置必须为0
obsClient.appendObject({
    Bucket:'bucketname',
    Key:'objectname',
    Position : 0,
    Body : 'Hello OBS'
}).then(function(result){
    console.log('Status-->' + result.CommonMsg.Status);
    if(result.CommonMsg.Status < 300 && result.InterfaceResult){
        console.log('NextPosition-->' + result.InterfaceResult.NextPosition);
    }
    // 第二次追加上传
    obsClient.appendObject({
        Bucket:'bucketname',
```



```
            Key:'objectname',
            Position : result.InterfaceResult.NextPosition,
            Body : 'Hello OBS Again'
        }, function(err, result2){
            if(err){
                console.error('Error-->' + err);
            }else{
                console.log('Status-->' + result2.CommonMsg.Status);
                if(result2.CommonMsg.Status < 300 && result2.InterfaceResult){
                    console.log('NextPosition-->' + result2.InterfaceResult.NextPosition);
                }
            }
        });

        // 通过获取对象属性接口获取下次追加上传的位置
        obsClient.getObjectMetadata({
            Bucket:'bucketname',
            Key:'objectname',
        }).then(function(result3){
            console.log('Status-->' + result3.CommonMsg.Status);
            if(result3.CommonMsg.Status < 300 && result3.InterfaceResult){
                console.log('RequestId-->' + result3.InterfaceResult.RequestId);
                console.log('NextPosition-->' + result3.InterfaceResult.NextPosition);
            }
        }).catch(function(err){
            console.error('err:' + err);
        });

}).catch(function(err){
    console.error('err:' + err);
});
```

说明

- 使用**Position参数**指定追加上传的位置，且第一次追加上传的位置必须为0。
- ObsClient.putObject上传的对象可覆盖ObsClient.appendObject上传的对象，覆盖后对象变为普通对象，不可再进行追加上传。
- 第一次调用追加上传时，如果已存在同名的普通对象，则会抛出异常（HTTP状态码为409）。
- 追加上传返回的ETag是当次追加数据内容的ETag，不是完整对象的ETag。
- 单次追加上传的内容不能超过5GB，且最多支持10000次追加上传。
- 追加上传成功后，可通过返回结果中InterfaceResult.NextPosition获取下次追加上传的位置；或者通过ObsClient.getObjectMetadata接口获取下次追加上传的位置。

6.10 分段复制

须知

开发过程中，您有任何问题可以在github上**提交issue**。**接口参考文档**详细介绍了每个接口的参数和使用方法。

分段复制是分段上传的一种特殊情况，即分段上传任务中的段通过复制OBS指定桶中现有对象（或对象的一部分）来实现。您可以通过ObsClient.copyPart来复制段。以下代码展示了如何使用分段复制模式复制大对象：

```
// 引入obs库
// 使用npm安装
var ObsClient = require('esdk-obs-nodejs');
// 使用源码安装
```



```
// var ObsClient = require('./lib/obs');

// 创建ObsClient实例
var obsClient = new ObsClient({
    //推荐通过环境变量获取AKSK，这里也可以使用其他外部引入方式传入，如果使用硬编码可能会存在泄露风险。
    //您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    access_key_id: process.env.ACCESS_KEY_ID,
    secret_access_key: process.env.SECRET_ACCESS_KEY,
    server : 'https://your-endpoint'
});

var destBucketName = 'destbucketname';
var destObjectKey = 'destobjectname';
var sourceBucketName = 'sourcebucketname';
var sourceObjectKey = 'sourceobjectname';

// 初始化分段上传任务
obsClient.initiateMultipartUpload({
    Bucket : destBucketName,
    Key : destObjectKey
}, (err, result) => {
    if(!err && result.CommonMsg.Status < 300){
        var uploadId = result.InterfaceResult.UploadId;
        console.log('\t' + uploadId + '\n');

        // 获取大对象信息
        obsClient.getObjectMetadata({
            Bucket : sourceBucketName,
            Key : sourceObjectKey
        }, (err, result) => {
            if(!err && result.CommonMsg.Status < 300){
                // 每段复制100MB
                var partSize = 100 * 1024 * 1024;
                var objectSize = Number(result.InterfaceResult.ContentLength);

                // 计算需要复制的段数
                var partCount = objectSize % partSize === 0 ? Math.floor(objectSize / partSize) : Math.floor(objectSize / partSize) + 1;

                var events = require('events');
                var eventEmitter = new events.EventEmitter();
                var parts = [];

                // 执行并发复制段
                for(let i=0;i<partCount;i++){
                    let rangeStart = i * partSize;
                    let rangeEnd = (i + 1) === partCount ? objectSize - 1 : rangeStart + partSize - 1;
                    let partNumber = i + 1;
                    obsClient.copyPart({
                        Bucket: destBucketName,
                        Key: destObjectKey,
                        PartNumber : partNumber,
                        UploadId : uploadId,
                        CopySource: sourceBucketName + '/' + sourceObjectKey,
                        CopySourceRange : 'bytes=' + rangeStart + '-' + rangeEnd
                    }, (err, result) => {
                        if(!err && result.CommonMsg.Status < 300){
                            parts.push({PartNumber : partNumber, ETag : result.InterfaceResult.ETag});
                            if(parts.length === partCount){
                                var _parts = parts.sort((a, b) => {
                                    if(a.PartNumber >= b.PartNumber){
                                        return 1;
                                    }
                                    return -1;
                                });
                                eventEmitter.emit('copy part finished', _parts);
                            }
```

```
                        }else{
                            throw new Error(err || result.CommonMsg.Code);
                        }
                    });
                }

                // 等待复制完成
                eventEmitter.on('copy part finished', (parts) => {
                    // 合并段
                    obsClient.completeMultipartUpload({
                        Bucket: destBucketName,
                        Key: destObjectKey,
                        UploadId: uploadId,
                        Parts: parts
                    }, (err, result) => {
                        if(!err && result.CommonMsg.Status < 300){
                            console.log('Complete to upload multiparts finished.\n');
                        }
                    });
                });
            }
        });
    }
});
```

说明

复制段时，使用**PartNumber参数**指定分段号；使用**UploadId参数**指定分段上传任务的全局唯一标识；使用**CopySource参数**指定复制时的源对象信息；使用**CopySourceRange参数**指定待复制的源对象的字节范围。

6.11 断点续传上传

须知

开发过程中，您有任何问题可以在github上**提交issue**。**接口参考文档**详细介绍了每个接口的参数和使用方法。

当上传大文件时，经常出现因网络不稳定或程序崩溃导致上传失败的情况。失败后再次重新上传不仅浪费资源，而且当网络不稳定时仍然有上传失败的风险。断点续传上传接口能有效地解决此类问题引起的上传失败，其原理是将待上传的文件分成若干个分段分别上传，并实时地将每段上传结果统一记录在checkpoint文件中，仅当所有分段都上传成功时返回上传成功的结果，否则在回调函数中返回错误码提醒用户再次调用接口进行重新上传（重新上传时因为有checkpoint文件记录当前的上传进度，避免重新上传所有分段，从而节省资源提高效率）。

注意

- 断点续传上传接口传入的文件总大小至少要100K以上。

您可以通过ObsClient.uploadFile进行断点续传上传。该接口可设置的主要参数如下：

参数	作用
Bucket	桶名，必选参数。

参数	作用
Key	对象名，必选参数。
UploadFile	待上传的本地文件，必选参数。
PartSize	分段大小，单位字节，取值范围是100KB~5GB，默认为5MB。
TaskNum	分段上传时的最大并发数，默认为20。
EnableCheckpoint	是否开启断点续传模式，默认为false，表示不开启。
CheckpointFile	记录上传进度的文件，只在断点续传模式下有效。当该值为空时，默认与待上传的本地文件同目录。
EnableCheckSum	是否校验待上传文件的内容，只在断点续传模式下有效。默认为false，表示不校验。

以下代码展示了如何使用断点续传上传接口上传文件：

```
// 引入obs库
// 使用npm安装
var ObsClient = require('esdk-obs-nodejs');
// 使用源码安装
// var ObsClient = require('./lib/obs');

// 创建ObsClient实例
var obsClient = new ObsClient({
      //推荐通过环境变量获取AKSK，这里也可以使用其他外部引入方式传入，如果使用硬编码可能会存在泄露风险。
      //您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
      access_key_id: process.env.ACCESS_KEY_ID,
      secret_access_key: process.env.SECRET_ACCESS_KEY,
      server : 'https://your-endpoint'
});

obsClient.uploadFile({
      Bucket : 'bucketname',
      Key : 'objectname',
      // 设置待上传的本地文件，localfile为待上传的本地文件路径，需要指定到具体的文件名
      UploadFile : 'localfile',
      // 设置分段大小为10MB
      PartSize : 10 * 1024 * 1024,
      // 开启断点续传模式
      EnableCheckpoint : true
}, (err, result) => {
   if(err){
       console.error('Error-->' + err);
   }else{
       console.log('RequestId-->' + result.InterfaceResult.RequestId);
       console.log('Bucket-->' + result.InterfaceResult.Bucket);
       console.log('Key-->' + result.InterfaceResult.Key);
       console.log('Location-->' + result.InterfaceResult.Location);
   }
});
```


说明

- 断点续传上传接口是利用**分段上传**特性实现的，是对分段上传的封装和加强。
- 断点续传上传接口不仅能在失败重传时节省资源提高效率，还因其对分段进行并发上传的机制能加快上传速度，能帮助用户快速完成上传业务；且其对用户透明，用户不用关心checkpoint文件的创建和删除、分段任务的切分、并发上传的实现等内部细节。
- **EnableCheckpoint参数**默认是false，代表不启用断点续传模式，此时断点续传上传接口退化成对分段上传的简单封装，不会产生checkpoint文件。
- **CheckpointFile参数**和**EnableCheckSum参数**仅在**EnableCheckpoint参数**为true时有效。

6.12 基于表单上传

须知

开发过程中，您有任何问题可以在github上**提交issue**。**接口参考文档**详细介绍了每个接口的参数和使用方法。

基于表单上传是使用HTML表单形式上传对象到指定桶中，对象最大不能超过5GB。

您可以通过ObsClient.createPostSignatureSync生成基于表单上传的请求参数。使用Node.js代码模拟表单上传的完整代码示例，参见**post-object-sample**。您也可以通过如下步骤进行表单上传：

步骤1 使用ObsClient.createPostSignatureSync生成用于鉴权的请求参数。

步骤2 准备表单HTML页面。

步骤3 将生成的请求参数填入HTML页面。

步骤4 选择本地文件，进行表单上传。

----结束

说明

使用SDK生成用于鉴权的请求参数包括两个：

- Policy，对应表单中policy字段。
- Signature，对应表单中的signature字段。

以下代码展示了如何生成基于表单上传的请求参数：

```
// 引入obs库
// 使用npm安装
var ObsClient = require('esdk-obs-nodejs');
// 使用源码安装
// var ObsClient = require('./lib/obs');

// 创建ObsClient实例
var obsClient = new ObsClient({
      //推荐通过环境变量获取AKSK，这里也可以使用其他外部引入方式传入，如果使用硬编码可能会存在泄露风险。
      //您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
      access_key_id: process.env.ACCESS_KEY_ID,
      secret_access_key: process.env.SECRET_ACCESS_KEY,
      server : 'https://your-endpoint',
      signature : 'obs'
```



```
});

// 设置表单参数
var formParams = {
            // 设置对象访问权限为公共读
            'x-obs-acl': obsClient.enums.AclPublicRead,
            // 设置对象MIME类型
            'content-type': 'text/plain',
};

// 设置表单上传请求有效期，单位：秒
var expires = 3600;

var res = obsClient.createPostSignatureSync({Expires:expires, FormParams: formParams});

// 获取表单上传请求参数
console.log('\t' + res.Policy);
console.log('\t' + res.Signature);
```

示例表单HTML代码如下：

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
</head>
<body>

<form action="http://bucketname.your-endpoint/" method="post" enctype="multipart/form-data">
Object key
<!-- 对象名 -->
<input type="text" name="key" value="objectname" />
<p>
ACL
<!-- 对象ACL权限 -->
<input type="text" name="x-obs-acl" value="public-read" />
<p>
Content-Type
<!-- 对象MIME类型 -->
<input type="text" name="content-type" value="text/plain" />
<p>
<!-- policy的base64编码值 -->
<input type="hidden" name="policy" value="*** Provide your policy ***" />
<!-- AK -->
<input type="hidden" name="AccessKeyId" value="*** Provide your access key ***"/>
<!-- 签名串信息 -->
<input type="hidden" name="signature" value="*** Provide your signature ***"/>

<input name="file" type="file" />
<input name="submit" value="Upload" type="submit" />
</form>
</body>
</html>
```

说明

- HTML表单中的policy，signature的值均是从ObsClient.createPostSignatureSync的返回结果中获取。
- 您可以直接下载表单HTML示例**PostDemo**。

7 下载对象

7.1 对象下载简介

须知

开发过程中，您有任何问题可以在github上**提交issue**。**接口参考文档**详细介绍了每个接口的参数和使用方法。

OBS Node.js SDK提供了丰富的对象下载接口，可以通过以下方式下载对象：

- **文本下载**
- **流式下载**
- **文件下载**
- **范围下载**
- **断点续传下载**

您可以通过ObsClient.getObject下载对象。

7.2 文本下载

须知

开发过程中，您有任何问题可以在github上**提交issue**。**接口参考文档**详细介绍了每个接口的参数和使用方法。

以下代码展示了如何进行文本下载：

```
// 引入obs库
// 使用npm安装
var ObsClient = require('esdk-obs-nodejs');
// 使用源码安装
// var ObsClient = require('./lib/obs');
```

```
// 创建ObsClient实例
var obsClient = new ObsClient({
    //推荐通过环境变量获取AKSK，这里也可以使用其他外部引入方式传入，如果使用硬编码可能会存在泄露风险。
    //您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    access_key_id: process.env.ACCESS_KEY_ID,
    secret_access_key: process.env.SECRET_ACCESS_KEY,
    server : 'https://your-endpoint'
});

obsClient.getObject({
    Bucket : 'bucketname',
    Key : 'objectname'
}, (err, result) => {
    if(err){
        console.error('Error-->' + err);
    }else{
        console.log('Status-->' + result.CommonMsg.Status);
        if(result.CommonMsg.Status < 300 && result.InterfaceResult){
            // 读取对象内容
            console.log('Object Content:');
            console.log(result.InterfaceResult.Content);
        }
    }
});
```

说明

文本下载方式下返回结果中的InterfaceResult.Content是一个String对象。

7.3 流式下载

须知

开发过程中，您有任何问题可以在github上**提交issue**。**接口参考文档**详细介绍了每个接口的参数和使用方法。

以下代码展示了如何进行流式下载：

```
// 引入obs库
// 使用npm安装
var ObsClient = require('esdk-obs-nodejs');
// 使用源码安装
// var ObsClient = require('./lib/obs');

// 创建ObsClient实例
var obsClient = new ObsClient({
    //推荐通过环境变量获取AKSK，这里也可以使用其他外部引入方式传入，如果使用硬编码可能会存在泄露风险。
    //您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    access_key_id: process.env.ACCESS_KEY_ID,
    secret_access_key: process.env.SECRET_ACCESS_KEY,
    server : 'https://your-endpoint'
});

obsClient.getObject({
    Bucket : 'bucketname',
    Key : 'objectname',
    SaveAsStream : true
}, (err, result) => {
    if(err){
```

```
        console.error('Error-->' + err);
    }else{
        console.log('Status-->' + result.CommonMsg.Status);
        if(result.CommonMsg.Status < 300 && result.InterfaceResult){
            // 读取对象内容
            console.log('Object Content:\n');
            result.InterfaceResult.Content.on('data', (data) => {
                console.log(data.toString());
            });
        }
    }
});
```

说明

- 使用**SaveAsStream参数**指定使用流式下载。
- InterfaceResult.Content是stream.Readable的实例，可将对象的内容读取到本地文件或者内存中。

7.4 文件下载

须知

开发过程中，您有任何问题可以在github上**提交issue**。**接口参考文档**详细介绍了每个接口的参数和使用方法。

以下代码展示了如何进行文件下载：

```
// 引入obs库
// 使用npm安装
var ObsClient = require('esdk-obs-nodejs');
// 使用源码安装
// var ObsClient = require('./lib/obs');

// 创建ObsClient实例
var obsClient = new ObsClient({
    //推荐通过环境变量获取AKSK，这里也可以使用其他外部引入方式传入，如果使用硬编码可能会存在泄露风险。
    //您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    access_key_id: process.env.ACCESS_KEY_ID,
    secret_access_key: process.env.SECRET_ACCESS_KEY,
    server : 'https://your-endpoint'
});

obsClient.getObject({
    Bucket : 'bucketname',
    Key : 'objectname',
    SaveAsFile : 'localfile'
}, (err, result) => {
    if(err){
        console.error('Error-->' + err);
    }else{
        console.log('Status-->' + result.CommonMsg.Status);
    }
});
```

说明

使用**SaveAsFile参数**指定文件下载的路径。

7.5 范围下载

须知

开发过程中，您有任何问题可以在github上**提交issue**。**接口参考文档**详细介绍了每个接口的参数和使用方法。

如果只需要下载对象的其中一部分数据，可以使用范围下载，下载指定范围的数据。如果指定的下载范围是0~1000，则返回第0到第1000个字节的数据，包括第1000个，共1001字节的数据，即[0，1000]。如果指定的范围无效，则返回整个对象的数据。以下代码展示了如何进行范围下载：

```
// 引入obs库
// 使用npm安装
var ObsClient = require('esdk-obs-nodejs');
// 使用源码安装
// var ObsClient = require('./lib/obs');

// 创建ObsClient实例
var obsClient = new ObsClient({
    //推荐通过环境变量获取AKSK，这里也可以使用其他外部引入方式传入，如果使用硬编码可能会存在泄露风险。
    //您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    access_key_id: process.env.ACCESS_KEY_ID,
    secret_access_key: process.env.SECRET_ACCESS_KEY,
    server : 'https://your-endpoint'
});

obsClient.getObject({
    Bucket : 'bucketname',
    Key : 'objectname',
    // 指定下载范围
    Range : 'bytes=0-1000'
}, (err, result) => {
    if(err){
        console.error('Error-->' + err);
    }else{
        console.log('Status-->' + result.CommonMsg.Status);
        if(result.CommonMsg.Status < 300 && result.InterfaceResult){
            // 读取对象内容
            console.log('Object Content:');
            console.log(result.InterfaceResult.Content.toString());
        }
    }
});
```

说明

- 使用**Range参数**指定下载范围，格式为“bytes=x-y”。
- 如果指定的范围无效（比如开始位置、结束位置为负数，大于文件大小），则会返回整个对象。
- 可以利用范围下载并发下载大对象，详细代码示例请参考**concurrent-download-object-sample**。

7.6 限定条件下载

须知

开发过程中，您有任何问题可以在github上**提交issue**。**接口参考文档**详细介绍了每个接口的参数和使用方法。

下载对象时，可以指定一个或多个限定条件，满足限定条件时则进行下载，否则返回异常码，下载对象失败。

您可以使用的限定条件如下：

参数	作用	格式
IfModifiedSince	如果对象在指定的时间后有修改，则返回对象内容，否则返回错误。	符合http://www.ietf.org/rfc/rfc2616.txt规定格式的HTTP时间字符串。
IfUnmodifiedSince	如果对象在指定的时间后没有修改，则返回对象内容，否则返回错误。	符合http://www.ietf.org/rfc/rfc2616.txt规定格式的HTTP时间字符串。
IfMatch	如果对象的ETag值与该参数值相同，则返回对象内容，否则返回异常码。	字符串。
IfNoneMatch	如果对象的ETag值与该参数值不相同，则返回对象内容，否则返回异常码。	字符串。

说明

- 对象的ETag值是指对象数据内容的MD5校验值。
- 如果包含IfUnmodifiedSince并且不符合或者包含IfMatch并且不符合，则下载对象失败，返回异常码：412 precondition failed。
- 如果包含IfModifiedSince并且不符合或者包含IfNoneMatch并且不符合，则下载对象失败，返回异常码：304 Not Modified。

以下代码展示了如何进行限定条件下载：

```
// 引入obs库
// 使用npm安装
var ObsClient = require('esdk-obs-nodejs');
// 使用源码安装
// var ObsClient = require('./lib/obs');

// 创建ObsClient实例
var obsClient = new ObsClient({
    //推荐通过环境变量获取AKSK，这里也可以使用其他外部引入方式传入，如果使用硬编码可能会存在泄露风
```

```
险。
    //您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/
intl/zh-cn/usermanual-ca/ca_01_0003.html
    access_key_id: process.env.ACCESS_KEY_ID,
    secret_access_key: process.env.SECRET_ACCESS_KEY,
    server : 'https://your-endpoint'
});

obsClient.getObject({
    Bucket : 'bucketname',
    Key : 'objectname',
    IfModifiedSince : 'Wed, 04 Jul 2018 08:54:53 GMT'
}, (err, result) => {
    if(err){
        console.error('Error-->' + err);
    }else{
        console.log('Status-->' + result.CommonMsg.Status);
        if(result.CommonMsg.Status < 300 && result.InterfaceResult){
            // 读取对象内容
            console.log('Object Content:');
            console.log(result.InterfaceResult.Content.toString());
        }
    }
});
```

7.7 重写响应头

须知

开发过程中，您有任何问题可以在github上**提交issue**。**接口参考文档**详细介绍了每个接口的参数和使用方法。

下载对象时，可以重写部分HTTP/HTTPS响应头信息。可重写的响应头信息见下表：

参数	作用
ResponseContentType	重写HTTP/HTTPS响应中的Content-Type
ResponseContentLanguage	重写HTTP/HTTPS响应中的Content-Language
ResponseExpires	重写HTTP/HTTPS响应中的Expires
ResponseCacheControl	重写HTTP/HTTPS响应中的Cache-Control
ResponseContentDisposition	重写HTTP/HTTPS响应中的Content-Disposition
ResponseContentEncoding	重写HTTP/HTTPS响应中的Content-Encoding

以下代码展示了如何重写响应头：

```
// 引入obs库
// 使用npm安装
var ObsClient = require('esdk-obs-nodejs');
// 使用源码安装
// var ObsClient = require('./lib/obs');


// 创建ObsClient实例
var obsClient = new ObsClient({
```

```
    //推荐通过环境变量获取AKSK，这里也可以使用其他外部引入方式传入，如果使用硬编码可能会存在泄露风险。
    //您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    access_key_id: process.env.ACCESS_KEY_ID,
    secret_access_key: process.env.SECRET_ACCESS_KEY,
    server : 'https://your-endpoint'
});

obsClient.getObject({
    Bucket : 'bucketname',
    Key : 'objectname',
    ResponseContentType : 'image/jpeg'
}, (err, result) => {
    if(err){
        console.error('Error-->' + err);
    }else{
        console.log('Status-->' + result.CommonMsg.Status);
        if(result.CommonMsg.Status < 300 && result.InterfaceResult){
            // 获取重写后的响应头
            console.log(result.InterfaceResult.ContentType);
        }
    }
});
```

7.8 获取自定义元数据

须知

开发过程中，您有任何问题可以在github上**提交issue**。**接口参考文档**详细介绍了每个接口的参数和使用方法。

下载对象成功后会返回对象的自定义元数据。以下代码展示了如何获取自定义元数据：

```
// 引入obs库
// 使用npm安装
var ObsClient = require('esdk-obs-nodejs');
// 使用源码安装
// var ObsClient = require('./lib/obs');

// 创建ObsClient实例
var obsClient = new ObsClient({
    //推荐通过环境变量获取AKSK，这里也可以使用其他外部引入方式传入，如果使用硬编码可能会存在泄露风险。
    //您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    access_key_id: process.env.ACCESS_KEY_ID,
    secret_access_key: process.env.SECRET_ACCESS_KEY,
    server : 'https://your-endpoint'
});

// 下载对象，获取对象自定义元数据
obsClient.getObject({
    Bucket : 'bucketname',
    Key : 'objectname',
}, (err, result) => {
    if(err){
        console.error('Error-->' + err);
    }else{
        console.log('Status-->' + result.CommonMsg.Status);
        if(result.CommonMsg.Status < 300 && result.InterfaceResult){
            console.log('Metadata-->' + JSON.stringify(result.InterfaceResult.Metadata['property']));
```



```
        }
    }
});
```

7.9 下载归档存储对象

须知

开发过程中，您有任何问题可以在github上**提交issue**。**接口参考文档**详细介绍了每个接口的参数和使用方法。

如果要下载归档存储对象，需要先将归档存储对象恢复。恢复归档存储对象的恢复选项可支持两类，见下表：

选项	说明	OBS Node.js SDK对应值
快速恢复	恢复耗时1~5分钟。	ObsClient.enums.RestoreTierExpedited
标准恢复	恢复耗时3~5小时。默认值。	ObsClient.enums.RestoreTierStandard

⚠ 注意

重复恢复归档存储数据时在延长恢复有效期的同时，也将会对恢复时产生的恢复费用进行重复收取。产生的标准存储类别的对象副本有效期将会延长，并且收取延长时间段产生的标准存储副本费用。

您可以通过ObsClient.restoreObject恢复归档存储对象。以下代码展示了如何下载归档存储对象：

```
// 引入obs库
// 使用npm安装
var ObsClient = require('esdk-obs-nodejs');
// 使用源码安装
// var ObsClient = require('./lib/obs');

// 创建ObsClient实例
var obsClient = new ObsClient({
    //推荐通过环境变量获取AKSK，这里也可以使用其他外部引入方式传入，如果使用硬编码可能会存在泄露风险。
    //您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    access_key_id: process.env.ACCESS_KEY_ID,
    secret_access_key: process.env.SECRET_ACCESS_KEY,
    server : 'https://your-endpoint'
});

// 恢复归档存储对象
obsClient.restoreObject({
    Bucket : 'bucketname',
    Key : 'objectname',
    Days : 1,
    Tier : obsClient.enums.RestoreTierExpedited
}, (err, result) => {
```

```
    if(err){
        console.error('Error-->' + err);
    }else{
        console.log('Status-->' + result.CommonMsg.Status);

        // 等待对象恢复
        setTimeout(()=>{

            // 下载对象, 获取对象内容
            obsClient.getObject({
                Bucket : 'bucketname',
                Key : 'objectname'
            }, (err, result) => {
                if(err){
                    console.error('Error-->' + err);
                }else{
                    console.log('Status-->' + result.CommonMsg.Status);
                    if(result.CommonMsg.Status < 300 && result.InterfaceResult){
                        // 读取对象内容
                        console.log('Object Content:');
                        console.log(result.InterfaceResult.Content.toString());
                    }
                }
            });
        }, 6 * 60 * 1000);
    }
});
```

说明

- ObsClient.restoreObject中指定的对象必须是归档存储类型，否则调用该接口会报错。
- 使用**Days参数**指定恢复对象保存的时间，取值范围是1~30；使用**Tier参数**指定恢复选项，表示恢复对象所耗的时间。

7.10 断点续传下载

须知

开发过程中，您有任何问题可以在github上**提交issue**。**接口参考文档**详细介绍了每个接口的参数和使用方法。

当下载大对象到本地文件时，经常出现因网络不稳定或程序崩溃导致下载失败的情况。失败后再次重新下载不仅浪费资源，而且当网络不稳定时仍然有下载失败的风险。断点续传下载接口能有效地解决此类问题引起的下载失败，其原理是将待下载的对象分成若干个分段分别下载，并实时地将每段下载结果统一记录在checkpoint文件中，仅当所有分段都下载成功时返回下载成功的结果，否则在回调函数中返回错误码提醒用户再次调用接口进行重新下载（重新下载时因为有checkpoint文件记录当前的下载进度，避免重新下载所有分段，从而节省资源提高效率）。

您可以通过ObsClient.downloadFile进行断点续传下载。该接口可设置的主要参数如下：

参数	作用
Bucket	桶名，必选参数。
Key	对象名，必选参数。

参数	作用
DownloadFile	下载对象的本地文件全路径。当该值为空时，默认为当前程序的运行目录。
PartSize	分段大小，单位字节，取值范围是100KB~5GB，默认为5MB。
TaskNum	分段下载时的最大并发数，默认为20。
EnableCheckpoint	是否开启断点续传模式，默认为false，表示不开启。
CheckpointFile	记录下载进度的文件，只在断点续传模式下有效。当该值为空时，默认与下载对象的本地文件路径同目录。
VersionId	对象的版本号。
IfModifiedSince	如果对象在指定的时间后有修改，则返回对象内容，否则返回错误。
IfUnmodifiedSince	如果对象在指定的时间后没有修改，则返回对象内容，否则返回错误。
IfMatch	如果对象的ETag值与该参数值相同，则返回对象内容，否则返回异常码。
IfNoneMatch	如果对象的ETag值与该参数值不相同，则返回对象内容，否则返回异常码。

以下代码展示了如何使用断点续传下载接口下载对象到本地文件：

```
// 引入obs库
// 使用npm安装
var ObsClient = require('esdk-obs-nodejs');
// 使用源码安装
// var ObsClient = require('./lib/obs');

// 创建ObsClient实例
var obsClient = new ObsClient({
        //推荐通过环境变量获取AKSK，这里也可以使用其他外部引入方式传入，如果使用硬编码可能会存在泄露风险。
        //您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
        access_key_id: process.env.ACCESS_KEY_ID,
        secret_access_key: process.env.SECRET_ACCESS_KEY,
        server : 'https://your-endpoint'
});

obsClient.downloadFile({
        Bucket : 'bucketname',
        Key : 'objectname',
        // 设置下载对象的本地文件路径
        DownloadFile : 'localfile',
        // 设置分段大小为10MB
        PartSize : 10 * 1024 * 1024,
        // 开启断点续传模式
        EnableCheckpoint : true
}, (err, result) => {
    if(err){
        console.error('Error-->' + err);
    }else{
        console.log('RequestId-->' + result.InterfaceResult.RequestId);
```



```
        console.log('LastModified-->' + result.InterfaceResult.LastModified);
        console.log('Metadata-->' + JSON.stringify(result.InterfaceResult.Metadata));
    }
});
```

说明

- 断点续传下载接口是利用**范围下载**特性实现的，是对范围下载的封装和加强。
- 断点续传下载接口不仅能在失败重下时节省资源提高效率，还因其对分段进行并发下载的机制能加快下载速度，能帮助用户快速完成下载业务；且其对用户透明，用户不用关心checkpoint文件的创建和删除、分段任务的切分、并发下载的实现等内部细节。
- **EnableCheckpoint参数**默认是false，代表不启用断点续传模式，此时断点续传下载接口退化成对范围下载的简单封装，不会产生checkpoint文件。
- **CheckpointFile参数**仅在**EnableCheckpoint参数**为true时有效。

7.11 图片处理

须知

开发过程中，您有任何问题可以在github上**提交issue**。**接口参考文档**详细介绍了每个接口的参数和使用方法。

OBS为用户提供了稳定、安全、高效、易用、低成本的图片处理服务。当要下载的对象是图片文件时，您可以通过传入图片处理参数对图片文件进行图片剪切、图片缩放、图片水印、格式转换等处理。

以下代码展示了如何使用下载对象接口实现图片处理：

```
// 引入obs库
// 使用npm安装
var ObsClient = require('esdk-obs-nodejs');
// 使用源码安装
// var ObsClient = require('./lib/obs');

// 创建ObsClient实例
var obsClient = new ObsClient({
    //推荐通过环境变量获取AKSK，这里也可以使用其他外部引入方式传入，如果使用硬编码可能会存在泄露风险。
    //您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    access_key_id: process.env.ACCESS_KEY_ID,
    secret_access_key: process.env.SECRET_ACCESS_KEY,
    server : 'https://your-endpoint'
});

obsClient.getObject({
    Bucket : 'bucketname',
    Key : 'objectname.jpg',
    // 对图片依次进行缩放、旋转
    ImageProcess : 'image/resize,m_fixed,w_100,h_100/rotate,90'
}, (err, result) => {
    if(err){
        console.error('Error-->' + err);
    }else{
        console.log('Status-->' + result.CommonMsg.Status);
    }
});
```


说明

- 使用**ImageProcess参数**指定图片处理参数。
- 图片处理参数支持级联处理，可对图片文件依次实施多条命令

8 管理对象

8.1 获取对象属性

须知

开发过程中，您有任何问题可以在github上**提交issue**。**接口参考文档**详细介绍了每个接口的参数和使用方法。

您可以通过ObsClient.getObjectMetadata来获取对象属性，包括对象长度，对象MIME类型，对象自定义元数据等信息。以下代码展示了如何获取对象属性：

```
// 引入obs库
// 使用npm安装
var ObsClient = require('esdk-obs-nodejs');
// 使用源码安装
// var ObsClient = require('./lib/obs');

// 创建ObsClient实例
var obsClient = new ObsClient({
    //推荐通过环境变量获取AKSK，这里也可以使用其他外部引入方式传入，如果使用硬编码可能会存在泄露风险。
    //您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    access_key_id: process.env.ACCESS_KEY_ID,
    secret_access_key: process.env.SECRET_ACCESS_KEY,
    server : 'https://your-endpoint'
});

obsClient.getObjectMetadata({
    Bucket : 'bucketname',
    // prefix：可选，为对象前文件夹的名称
    Key : '[prefix/]objectname'
}, (err, result) => {
    if(err){
      console.error('Error-->' + err);
    }else{
      console.log('Status-->' + result.CommonMsg.Status);
      if(result.CommonMsg.Status < 300 && result.InterfaceResult){
        console.log(result.InterfaceResult.ContentType);
        console.log(result.InterfaceResult.ContentLength);
        console.log(result.InterfaceResult.Metadata['property']);
      }
```

```
    }
});
```

8.2 管理对象访问权限

须知

开发过程中，您有任何问题可以在github上**提交issue**。**接口参考文档**详细介绍了每个接口的参数和使用方法。

对象访问权限与桶访问权限类似，也可支持预定义访问策略（参见**桶访问权限**）或直接设置。

对象访问权限（**ACL**）可以通过三种方式设置：

1. 上传对象时指定预定义访问策略。
2. 调用ObsClient.setObjectAcl指定预定义访问策略。
3. 调用ObsClient.setObjectAcl直接设置。

上传对象时指定预定义访问策略

以下代码展示如何在上传对象时指定预定义访问策略：

```
// 引入obs库
// 使用npm安装
var ObsClient = require('esdk-obs-nodejs');
// 使用源码安装
// var ObsClient = require('./lib/obs');

// 创建ObsClient实例
var obsClient = new ObsClient({
    //推荐通过环境变量获取AKSK，这里也可以使用其他外部引入方式传入，如果使用硬编码可能会存在泄露风险。
    //您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    access_key_id: process.env.ACCESS_KEY_ID,
    secret_access_key: process.env.SECRET_ACCESS_KEY,
    server : 'https://your-endpoint'
});

obsClient.putObject({
    Bucket : 'bucketname',
    Key : 'objectname',
    Body : 'Hello OBS',
    // 设置对象访问权限为公共读
    ACL : obsClient.enums.AclPublicRead
}, (err, result) => {
    if(err){
        console.error('Error-->' + err);
    }else{
        console.log('Status-->' + result.CommonMsg.Status);
    }
});
```

为对象设置预定义访问策略

以下代码展示如何为对象设置预定义访问策略：


```
// 引入obs库
// 使用npm安装
var ObsClient = require('esdk-obs-nodejs');
// 使用源码安装
// var ObsClient = require('./lib/obs');

// 创建ObsClient实例
var obsClient = new ObsClient({
    //推荐通过环境变量获取AKSK，这里也可以使用其他外部引入方式传入，如果使用硬编码可能会存在泄露风险。
    //您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    access_key_id: process.env.ACCESS_KEY_ID,
    secret_access_key: process.env.SECRET_ACCESS_KEY,
    server : 'https://your-endpoint'
});

obsClient.setObjectAcl({
    Bucket : 'bucketname',
    Key : 'objectname',
    // 设置对象访问权限为私有读写
    ACL : obsClient.enums.AclPrivate
}, (err, result) => {
    if(err){
      console.error('Error-->' + err);
    }else{
      console.log('Status-->' + result.CommonMsg.Status);
    }
});
```

说明

使用**ACL参数**指定对象的访问权限。

直接设置对象访问权限

以下代码展示如何直接设置对象访问权限：

```
// 引入obs库
// 使用npm安装
var ObsClient = require('esdk-obs-nodejs');
// 使用源码安装
// var ObsClient = require('./lib/obs');

// 创建ObsClient实例
var obsClient = new ObsClient({
    //推荐通过环境变量获取AKSK，这里也可以使用其他外部引入方式传入，如果使用硬编码可能会存在泄露风险。
    //您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    access_key_id: process.env.ACCESS_KEY_ID,
    secret_access_key: process.env.SECRET_ACCESS_KEY,
    server : 'https://your-endpoint'
});

obsClient.setObjectAcl({
    Bucket : 'bucketname',
    Key : 'objectname',
    // 设置对象所有者
    Owner:{'ID':'ownerid'},
    Grants:[
        // 为指定用户设置完全控制权限
       { Grantee : {Type : 'CanonicalUser',ID : 'userid'}, Permission : obsClient.enums.PermissionFullControl},
        // 为所有用户设置读权限
       { Grantee: {Type : 'Group', URI : obsClient.enums.GroupAllUsers}, Permission : obsClient.enums.PermissionRead},
    ]
}, (err, result) => {
    if(err){
```

```
        console.error('Error-->' + err);
    }else{
        console.log('Status-->' + result.CommonMsg.Status);
    }
});
```

说明

- 使用**Owner参数**指定对象的所有者信息；使用**Grants参数**指定被授权的用户信息。
- ACL中需要填写的所有者（Owner）或者被授权用户（Grantee）的ID，是指用户的账户ID，可通过OBS控制台“我的凭证”页面查看。
- 当前OBS对象支持的可被授权的用户组为：
  - 所有用户：ObsClient.enums.GroupAllUsers

获取对象访问权限

您可以通过ObsClient.getObjectAcl获取对象的访问权限。以下代码展示如何获取对象访问权限：

```
// 引入obs库
// 使用npm安装
var ObsClient = require('esdk-obs-nodejs');
// 使用源码安装
// var ObsClient = require('./lib/obs');

// 创建ObsClient实例
var obsClient = new ObsClient({
    //推荐通过环境变量获取AKSK，这里也可以使用其他外部引入方式传入，如果使用硬编码可能会存在泄露风险。
    //您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    access_key_id: process.env.ACCESS_KEY_ID,
    secret_access_key: process.env.SECRET_ACCESS_KEY,
    server : 'https://your-endpoint'
});

obsClient.getObjectAcl({
    Bucket : 'bucketname',
    Key : 'objectname'
}, (err, result) => {
    if(err){
        console.error('Error-->' + err);
    }else{
        console.log('Status-->' + result.CommonMsg.Status);
            if(result.CommonMsg.Status < 300 && result.InterfaceResult){
                console.log('Owner[ID]-->' + result.InterfaceResult.Owner.ID);
                console.log('Owner[Name]-->' + result.InterfaceResult.Owner.Name);
                for(let i=0;i<result.InterfaceResult.Grants.length;i++){
                    console.log('Grant[' + i + ']:');
                    console.log('Grantee[ID]-->' + result.InterfaceResult.Grants[i]['Grantee']['ID']);
                    console.log('Grantee[URI]-->' + result.InterfaceResult.Grants[i]['Grantee']['URI']);
                    console.log('Permission-->' + result.InterfaceResult.Grants[i]['Permission']);
                }
            }
    }
});
```


8.3 列举对象

须知

开发过程中，您有任何问题可以在github上**提交issue**。**接口参考文档**详细介绍了每个接口的参数和使用方法。

您可以通过ObsClient.listObjects列举出桶里的对象。

该接口可设置的参数如下：

参数	作用
Prefix	限定返回的对象名必须带有Prefix前缀。
Marker	列举对象的起始位置，返回的对象列表将是对象名按照字典序排序后该参数以后的所有对象。
MaxKeys	列举对象的最大数目，取值范围为1~1000，当超出范围时，按照默认的1000进行处理。
Delimiter	用于对对象名进行分组的字符。对于对象名中包含Delimiter的对象，其对象名（如果请求中指定了Prefix，则此处的对象名需要去掉Prefix）中从首字符至第一个Delimiter之间的字符串将作为一个分组并作为CommonPrefix返回。 对于并行文件系统，不携带此参数时默认列举是递归列举此目录下所有内容，会列举子目录。在大数据场景下（目录层级深、目录下文件多）的列举，建议设置[delimiter='/']，只列举当前目录下的内容，不列举子目录，提高列举效率。

简单列举

以下代码展示如何简单列举对象，最多返回1000个对象：

```
// 引入obs库
// 使用npm安装
var ObsClient = require('esdk-obs-nodejs');
// 使用源码安装
// var ObsClient = require('./lib/obs');

// 创建ObsClient实例
var obsClient = new ObsClient({
        //推荐通过环境变量获取AKSK，这里也可以使用其他外部引入方式传入，如果使用硬编码可能会存在泄露风险。
        //您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
        access_key_id: process.env.ACCESS_KEY_ID,
        secret_access_key: process.env.SECRET_ACCESS_KEY,
        server : 'https://your-endpoint'
});

obsClient.listObjects({
        Bucket : 'bucketname'
```

```
}, (err, result) => {
    if(err){
      console.error('Error-->' + err);
    }else{
      console.log('Status-->' + result.CommonMsg.Status);
        if(result.CommonMsg.Status < 300 && result.InterfaceResult){
            for(let j=0;j<result.InterfaceResult.Contents.length;j++){
                console.log('Contents[' + j +  ']:');
                console.log('Key-->' + result.InterfaceResult.Contents[j]['Key']);
                console.log('Owner[ID]-->' + result.InterfaceResult.Contents[j]['Owner']['ID']);
              }
        }
    }
});
```

说明

- 每次至多返回1000个对象，如果指定桶包含的对象数量大于1000，则返回结果中InterfaceResult.IsTruncated为true表明本次没有返回全部对象，并可通过InterfaceResult.NextMarker获取下次列举的起始位置。
- 如果想获取指定桶包含的所有对象，可以采用分页列举的方式。

指定数目列举

以下代码展示如何指定数目列举对象：

```
// 引入obs库
// 使用npm安装
var ObsClient = require('esdk-obs-nodejs');
// 使用源码安装
// var ObsClient = require('./lib/obs');

// 创建ObsClient实例
var obsClient = new ObsClient({
    //推荐通过环境变量获取AKSK，这里也可以使用其他外部引入方式传入，如果使用硬编码可能会存在泄露风险。
    //您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    access_key_id: process.env.ACCESS_KEY_ID,
    secret_access_key: process.env.SECRET_ACCESS_KEY,
    server : 'https://your-endpoint'
});

obsClient.listObjects({
    Bucket : 'bucketname',
    // 只列举100个对象
    MaxKeys : 100
}, (err, result) => {
    if(err){
      console.error('Error-->' + err);
    }else{
      console.log('Status-->' + result.CommonMsg.Status);
        if(result.CommonMsg.Status < 300 && result.InterfaceResult){
            for(let j=0;j<result.InterfaceResult.Contents.length;j++){
                console.log('Contents[' + j +  ']:');
                console.log('Key-->' + result.InterfaceResult.Contents[j]['Key']);
                console.log('Owner[ID]-->' + result.InterfaceResult.Contents[j]['Owner']['ID']);
            }
        }
    }
});
```

指定前缀列举

以下代码展示如何指定前缀列举对象：

```
// 引入obs库
// 使用npm安装
var ObsClient = require('esdk-obs-nodejs');
// 使用源码安装
// var ObsClient = require('./lib/obs');

// 创建ObsClient实例
var obsClient = new ObsClient({
    //推荐通过环境变量获取AKSK，这里也可以使用其他外部引入方式传入，如果使用硬编码可能会存在泄露风险。
    //您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    access_key_id: process.env.ACCESS_KEY_ID,
    secret_access_key: process.env.SECRET_ACCESS_KEY,
    server : 'https://your-endpoint'
});

obsClient.listObjects({
    Bucket : 'bucketname',
    // 设置列举带有prefix前缀的100个对象
    MaxKeys : 100,
    Prefix : 'prefix'
}, (err, result) => {
    if(err){
      console.error('Error-->' + err);
    }else{
      console.log('Status-->' + result.CommonMsg.Status);
        if(result.CommonMsg.Status < 300 && result.InterfaceResult){
          for(let j=0;j<result.InterfaceResult.Contents.length;j++){
            console.log('Contents[' + j +  ']:');
            console.log('Key-->' + result.InterfaceResult.Contents[j]['Key']);
            console.log('Owner[ID]-->' + result.InterfaceResult.Contents[j]['Owner']['ID']);
          }
        }
    }
});
```

指定起始位置列举

以下代码展示如何指定起始位置列举对象：

```
// 引入obs库
// 使用npm安装
var ObsClient = require('esdk-obs-nodejs');
// 使用源码安装
// var ObsClient = require('./lib/obs');

// 创建ObsClient实例
var obsClient = new ObsClient({
    //推荐通过环境变量获取AKSK，这里也可以使用其他外部引入方式传入，如果使用硬编码可能会存在泄露风险。
    //您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    access_key_id: process.env.ACCESS_KEY_ID,
    secret_access_key: process.env.SECRET_ACCESS_KEY,
    server : 'https://your-endpoint'
});

obsClient.listObjects({
    Bucket : 'bucketname',
    // 设置列举对象名字典序在"test"之后的100个对象
    MaxKeys : 100,
    Marker : 'test'
}, (err, result) => {
    if(err){
      console.error('Error-->' + err);
    }else{
      console.log('Status-->' + result.CommonMsg.Status);
        if(result.CommonMsg.Status < 300 && result.InterfaceResult){
```

```
                for(let j=0;j<result.InterfaceResult.Contents.length;j++){
                    console.log('Contents[' + j +  ']:');
                    console.log('Key-->' + result.InterfaceResult.Contents[j]['Key']);
                    console.log('Owner[ID]-->' + result.InterfaceResult.Contents[j]['Owner']['ID']);
                }
            }
        }
});
```

分页列举全部对象

以下代码展示分页列举全部对象：

```
// 引入obs库
// 使用npm安装
var ObsClient = require('esdk-obs-nodejs');
// 使用源码安装
// var ObsClient = require('./lib/obs');

// 创建ObsClient实例
var obsClient = new ObsClient({
    //推荐通过环境变量获取AKSK，这里也可以使用其他外部引入方式传入，如果使用硬编码可能会存在泄露风险。
    //您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    access_key_id: process.env.ACCESS_KEY_ID,
    secret_access_key: process.env.SECRET_ACCESS_KEY,
    server : 'https://your-endpoint'
});

var listAll = (marker) => {
    obsClient.listObjects({
        Bucket : 'bucketname',
        // 设置每页100个对象
        MaxKeys : 100,
        Marker : marker
    }, (err, result) => {
        if(err){
            console.error('Error-->' + err);
        }else{
            console.log('Status-->' + result.CommonMsg.Status);
            if(result.CommonMsg.Status < 300 && result.InterfaceResult){
                for(let j=0;j<result.InterfaceResult.Contents.length;j++){
                    console.log('Contents[' + j +  ']:');
                    console.log('Key-->' + result.InterfaceResult.Contents[j]['Key']);
                    console.log('Owner[ID]-->' + result.InterfaceResult.Contents[j]['Owner']['ID']);

                }
                if(result.InterfaceResult.IsTruncated === 'true'){
                    listAll(result.InterfaceResult.NextMarker);
                }
            }
        }
    });
};

listAll();
```

列举文件夹中的所有对象

OBS本身是没有文件夹的概念的，桶中存储的元素只有对象。文件夹对象实际上是一个大小为0且对象名以“/”结尾的对象，将这个文件夹对象名作为前缀，即可模拟列举文件夹中对象的功能。以下代码展示如何列举文件夹中的对象：

```
// 引入obs库
// 使用npm安装
var ObsClient = require('esdk-obs-nodejs');
```



```
// 使用源码安装
// var ObsClient = require('./lib/obs');

// 创建ObsClient实例
var obsClient = new ObsClient({
    //推荐通过环境变量获取AKSK，这里也可以使用其他外部引入方式传入，如果使用硬编码可能会存在泄露风险。
    //您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    access_key_id: process.env.ACCESS_KEY_ID,
    secret_access_key: process.env.SECRET_ACCESS_KEY,
    server : 'https://your-endpoint'
});

var listAll = (marker) => {
    obsClient.listObjects({
        Bucket : 'bucketname',
        MaxKeys : 1000,
        // 设置文件夹对象名"dir/"为前缀
        Prefix : 'dir/'
    }, (err, result) => {
        if(err){
            console.error('Error-->' + err);
        }else{
            console.log('Status-->' + result.CommonMsg.Status);
            if(result.CommonMsg.Status < 300 && result.InterfaceResult){
                for(let j=0;j<result.InterfaceResult.Contents.length;j++){
                    console.log('Contents[' + j +  ']:');
                    console.log('Key-->' + result.InterfaceResult.Contents[j]['Key']);
                    console.log('Owner[ID]-->' + result.InterfaceResult.Contents[j]['Owner']['ID']);

                }
                if(result.InterfaceResult.IsTruncated === 'true'){
                    listAll(result.InterfaceResult.NextMarker);
                }
            }
        }
    });
};

listAll();
```

按文件夹分组列举所有对象

以下代码展示如何按文件夹分组，列举桶内所有对象：

```
// 引入obs库
// 使用npm安装
var ObsClient = require('esdk-obs-nodejs');
// 使用源码安装
// var ObsClient = require('./lib/obs');

// 创建ObsClient实例
var obsClient = new ObsClient({
    //推荐通过环境变量获取AKSK，这里也可以使用其他外部引入方式传入，如果使用硬编码可能会存在泄露风险。
    //您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    access_key_id: process.env.ACCESS_KEY_ID,
    secret_access_key: process.env.SECRET_ACCESS_KEY,
    server : 'https://your-endpoint'
});

obsClient.listObjects({
    Bucket: 'bucketname',
    // 设置文件夹分隔符"/"
    Delimiter: '/'
}, (err, result) => {
    if(!err && result.CommonMsg.Status < 300){
```

```
        console.log('Objects in the root directory:');
        for(let j=0;j<result.InterfaceResult.Contents.length;j++){
            console.log('\tKey-->' + result.InterfaceResult.Contents[j]['Key']);
            console.log('Owner[ID]-->' + result.InterfaceResult.Contents[j]['Owner']
['ID']);
            console.log('Owner[Name]-->' + result.InterfaceResult.Contents[j]['Owner']['Name']);
        }
        var listObjectsByPrefix = (commonPrefixes) => {
            for(let i=0;i<commonPrefixes.length;i++){
                obsClient.listObjects({
                    Bucket: 'bucketname',
                    Delimiter: '/',
                    Prefix: commonPrefixes[i]['Prefix']
                }, (err, result)=>{
                    if(!err && result.CommonMsg.Status < 300){
                        console.log('Objects in folder [' + commonPrefixes[i]['Prefix'] + ']:');
                        for(let j=0;j<result.InterfaceResult.Contents.length;j++){
                            console.log('\tKey-->' + result.InterfaceResult.Contents[j]['Key']);
                            console.log('Owner[ID]-->' + result.InterfaceResult.Contents[j]['Owner']
['ID']);
                        }
                        console.log('\n');
                        if(result.InterfaceResult.CommonPrefixes &&
result.InterfaceResult.CommonPrefixes.length > 0){
                            listObjectsByPrefix(result.InterfaceResult.CommonPrefixes);
                        }
                    }
                });
            }
        };
        listObjectsByPrefix(result.InterfaceResult.CommonPrefixes);
    }
});
```

说明

- 以上代码示例没有考虑文件夹中对象数超过1000个的情况。
- 由于是需要列举出文件夹中的对象和子文件夹，且文件夹对象总是以“/”结尾，因此Delimiter总是为“/”。
- 每次递归的返回结果中InterfaceResult.Contents包含的是文件夹中的对象；InterfaceResult.CommonPrefixes包含的是文件夹的子文件夹。

8.4 删除对象

须知

开发过程中，您有任何问题可以在github上**提交issue**。**接口参考文档**详细介绍了每个接口的参数和使用方法。

说明

请您谨慎使用删除操作，如果对象所在的桶未开启多版本控制功能，该对象一旦删除将无法恢复。

删除单个对象

您可以通过ObsClient.deleteObject删除单个对象。以下代码展示如何删除单个对象：

```
// 引入obs库
// 使用npm安装
```

```
var ObsClient = require('esdk-obs-nodejs');
// 使用源码安装
// var ObsClient = require('./lib/obs');

// 创建ObsClient实例
var obsClient = new ObsClient({
    //推荐通过环境变量获取AKSK，这里也可以使用其他外部引入方式传入，如果使用硬编码可能会存在泄露风险。
    //您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    access_key_id: process.env.ACCESS_KEY_ID,
    secret_access_key: process.env.SECRET_ACCESS_KEY,
    server : 'https://your-endpoint'
});

obsClient.deleteObject({
    Bucket: 'bucketname',
    Key : 'objectname'
}, (err, result) => {
    if(err){
        console.log('Error-->' + err);
    }else{
        console.log('Status-->' + result.CommonMsg.Status);
    }
});
```

批量删除对象

您可以通过ObsClient.deleteObjects批量删除对象。

每次最多删除1000个对象，并支持两种响应模式：详细（verbose）模式和简单（quiet）模式。

- 详细模式：返回的删除成功和删除失败的所有结果，默认模式。
- 简单模式：只返回的删除过程中出错的结果。

以下代码展示了如何进行批量删除对象：

```
// 引入obs库
// 使用npm安装
var ObsClient = require('esdk-obs-nodejs');
// 使用源码安装
// var ObsClient = require('./lib/obs');

// 创建ObsClient实例
var obsClient = new ObsClient({
    //推荐通过环境变量获取AKSK，这里也可以使用其他外部引入方式传入，如果使用硬编码可能会存在泄露风险。
    //您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    access_key_id: process.env.ACCESS_KEY_ID,
    secret_access_key: process.env.SECRET_ACCESS_KEY,
    server : 'https://your-endpoint'
});

obsClient.deleteObjects({
    Bucket: 'bucketname',
    // 设置为verbose模式
    Quiet : false,
    Objects : [{Key:'objectname1'},{Key:'objectname2'}, {Key : 'objectname3'}]
}, (err, result) => {
    if(err){
        console.log('Error-->' + err);
    }else{
        console.log('Status-->' + result.CommonMsg.Status);
        if(result.CommonMsg.Status < 300 && result.InterfaceResult){
```

```
            // 获取删除成功的对象
            console.log('Deleteds:');
            for(let i=0;i<result.InterfaceResult.Deleteds.length;i++){
                console.log('Deleted[' + i + ']:');
                console.log('Key-->'+result.InterfaceResult.Deleteds[i]['Key']);
                console.log('VersionId-->' + result.InterfaceResult.Deleteds[i]['VersionId']);
            }
            // 获取删除失败的对象
            console.log('Errors:');
            for(let i=0;i<result.InterfaceResult.Errors.length;i++){
                console.log('Error[' + i + ']:');
                console.log('Key-->' + result.InterfaceResult.Errors[i]['Key']);
                console.log('VersionId-->' + result.InterfaceResult.Errors[i]['VersionId']);
            }
        }
    }
});
```

说明

使用**Quiet参数**指定响应模式；使用**Objects参数**指定待删除的对象列表。

8.5 复制对象

须知

开发过程中，您有任何问题可以在github上**提交issue**。**接口参考文档**详细介绍了每个接口的参数和使用方法。

复制对象特性用来为OBS上已经存在的对象创建一个副本。

您可以通过ObsClient.copyObject来复制对象。复制对象时，可重新指定新对象的属性和设置对象权限，且支持条件复制。

说明

- 如果待复制的源对象是归档存储类型，则必须先恢复源对象才能进行复制。

简单复制

以下代码展示了如何进行简单复制：

```
// 引入obs库
// 使用npm安装
var ObsClient = require('esdk-obs-nodejs');
// 使用源码安装
// var ObsClient = require('./lib/obs');

// 创建ObsClient实例
var obsClient = new ObsClient({
    //推荐通过环境变量获取AKSK，这里也可以使用其他外部引入方式传入，如果使用硬编码可能会存在泄露风险。
    //您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    access_key_id: process.env.ACCESS_KEY_ID,
    secret_access_key: process.env.SECRET_ACCESS_KEY,
    server : 'https://your-endpoint'
});

obsClient.copyObject({
```



```
    Bucket: 'destbucketname',
    Key : 'destobjectname',
    CopySource:'sourcebucketname/sourceobjectname'
}, (err, result) => {
    if(err){
        console.log('Error-->' + err);
    }else{
        console.log('Status-->' + result.CommonMsg.Status);
    }
});
```

说明

使用**CopySource参数**指定复制时的源对象信息。

重写对象属性

以下代码展示了如何在复制对象时重写对象属性：

```
// 引入obs库
// 使用npm安装
var ObsClient = require('esdk-obs-nodejs');
// 使用源码安装
// var ObsClient = require('./lib/obs');

// 创建ObsClient实例
var obsClient = new ObsClient({
    //推荐通过环境变量获取AKSK，这里也可以使用其他外部引入方式传入，如果使用硬编码可能会存在泄露风险。
    //您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    access_key_id: process.env.ACCESS_KEY_ID,
    secret_access_key: process.env.SECRET_ACCESS_KEY,
    server : 'https://your-endpoint'
});

obsClient.copyObject({
    Bucket: 'destbucketname',
    Key : 'destobjectname',
    CopySource:'sourcebucketname/sourceobjectname',
    ContentType : 'image/jpeg',
    StorageClass : obsClient.enums.StorageClassWarm,
    Metadata:{'property':'property-value'},
    MetadataDirective : ObsClient.enums.ReplaceMetadata
}, (err, result) => {
    if(err){
        console.log('Error-->' + err);
    }else{
        console.log('Status-->' + result.CommonMsg.Status);
    }
});
```

说明

使用**Metadata参数**指定待重写的自定义对象属性；使用**MetadataDirective参数**指定重写选项，支持ObsClient.enums.CopyMetadata（从源对象复制）和ObsClient.enums.ReplaceMetadata（重写）两个值。

限定条件复制

复制对象时，可以指定一个或多个限定条件，满足限定条件时则进行复制，否则返回异常码，复制对象失败。

您可以使用的限定条件如下：

参数	作用	格式
CopySourceIfModifiedSince	如果源对象在指定的时间后有修改，则进行复制，否则抛出异常。	符合http://www.ietf.org/rfc/rfc2616.txt规定格式的HTTP时间字符串。
CopySourceIfUnmodifiedSince	如果源对象在指定的时间后没有修改，则进行复制，否则抛出异常。	符合http://www.ietf.org/rfc/rfc2616.txt规定格式的HTTP时间字符串。
CopySourceIfMatch	如果源对象的ETag值与该参数值相同，则进行复制，否则返回异常码。	字符串。
CopySourceIfNoneMatch	如果源对象的ETag值与该参数值不相同，则进行复制，否则返回异常码。	字符串。

说明

- 源对象的ETag值是指源对象数据的MD5校验值。
- 如果包含CopySourceIfUnmodifiedSince并且不符合，或者包含CopySourceIfMatch并且不符合，或者包含CopySourceIfModifiedSince并且不符合，或者包含CopySourceIfNoneMatch并且不符合，则复制失败，返回异常码：412 precondition failed。
- CopySourceIfModifiedSince和CopySourceIfNoneMatch可以一起使用；CopySourceIfUnmodifiedSince和CopySourceIfMatch可以一起使用。

以下代码展示了如何进行限定条件复制：

```
// 引入obs库
// 使用npm安装
var ObsClient = require('esdk-obs-nodejs');
// 使用源码安装
// var ObsClient = require('./lib/obs');

// 创建ObsClient实例
var obsClient = new ObsClient({
      //推荐通过环境变量获取AKSK，这里也可以使用其他外部引入方式传入，如果使用硬编码可能会存在泄露风险。
      //您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
      access_key_id: process.env.ACCESS_KEY_ID,
      secret_access_key: process.env.SECRET_ACCESS_KEY,
      server : 'https://your-endpoint'
});

obsClient.copyObject({
    Bucket: 'destbucketname',
    Key : 'destobjectname',
    CopySource:'sourcebucketname/sourceobjectname',
    CopySourceIfModifiedSince : 'Thu, 31 Dec 2015 16:00:00 GMT',
    CopySourceIfNoneMatch : 'none-match-etag'
}, (err, result) => {
      if(err){
            console.log('Error-->' + err);
      }else{
            console.log('Status-->' + result.CommonMsg.Status);
```



```
    }
});
```

重写对象访问权限

以下代码展示了如何在复制对象时重写对象访问权限：

```
// 引入obs库
// 使用npm安装
var ObsClient = require('esdk-obs-nodejs');
// 使用源码安装
// var ObsClient = require('./lib/obs');

// 创建ObsClient实例
var obsClient = new ObsClient({
    //推荐通过环境变量获取AKSK，这里也可以使用其他外部引入方式传入，如果使用硬编码可能会存在泄露风险。
    //您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    access_key_id: process.env.ACCESS_KEY_ID,
    secret_access_key: process.env.SECRET_ACCESS_KEY,
    server : 'https://your-endpoint'
});

obsClient.copyObject({
    Bucket: 'destbucketname',
    Key : 'destobjectname',
    CopySource:'sourcebucketname/sourceobjectname',
    // 复制时重写对象访问权限为公共读
    ACL : obsClient.enums.AclPublicRead
}, (err, result) => {
    if(err){
        console.log('Error-->' + err);
    }else{
        console.log('Status-->' + result.CommonMsg.Status);
    }
});
```

说明

使用**ACL参数**重写对象访问权限。

9 临时授权访问

9.1 使用临时 URL 进行授权访问

须知

开发过程中，您有任何问题可以在github上**提交issue**。**接口参考文档**详细介绍了每个接口的参数和使用方法。

说明

使用**Method参数**指定HTTP请求方法类型；使用**Expires参数**（单位：秒）指定生成的URL有效期；使用**Headers参数**指定请求的头信息；使用**SpecialParam参数**指定特殊操作符；使用**QueryParams参数**指定请求的查询参数。

OBS客户端支持通过访问密钥、请求方法类型、请求参数等信息生成一个在Query参数中携带鉴权信息的URL，可将该URL提供给其他用户进行临时访问。在生成URL时，您需要指定URL的有效期来限制访客用户的访问时长。

如果您想授予其他用户对桶或对象临时进行其他操作的权限（例如上传或下载对象），则需要生成带对应请求的URL后（例如使用生成PUT请求的URL上传对象），将该URL提供给其他用户。

通过该方式可支持的操作以及相关信息见下表：

操作名	HTTP请求方法	特殊操作符（子资源）	是否需要桶名	是否需要对象名
创建桶	PUT	N/A	是	否
获取桶列表	GET	N/A	否	否
删除桶	DELETE	N/A	是	否
列举桶内对象	GET	N/A	是	否
列举桶内多版本对象	GET	versions	是	否

操作名	HTTP请求方法	特殊操作符（子资源）	是否需要桶名	是否需要对象名
列举分段上传任务	GET	uploads	是	否
获取桶元数据	HEAD	N/A	是	否
获取桶区域位置	GET	location	是	否
获取桶存量信息	GET	storageinfo	是	否
设置桶配额	PUT	quota	是	否
获取桶配额	GET	quota	是	否
设置桶存储类型	PUT	storagePolicy	是	否
获取桶存储类型	GET	storagePolicy	是	否
设置桶访问权限	PUT	acl	是	否
获取桶访问权限	GET	acl	是	否
开启/关闭桶日志	PUT	logging	是	否
查看桶日志	GET	logging	是	否
设置桶策略	PUT	policy	是	否
查看桶策略	GET	policy	是	否
删除桶策略	DELETE	policy	是	否
设置生命周期规则	PUT	lifecycle	是	否
查看生命周期规则	GET	lifecycle	是	否
删除生命周期规则	DELETE	lifecycle	是	否
设置托管配置	PUT	website	是	否
查看托管配置	GET	website	是	否
清除托管配置	DELETE	website	是	否
设置桶多版本状态	PUT	versioning	是	否
查看桶多版本状态	GET	versioning	是	否
设置跨域规则	PUT	cors	是	否
查看跨域规则	GET	cors	是	否
删除跨域规则	DELETE	cors	是	否
设置桶标签	PUT	tagging	是	否
查看桶标签	GET	tagging	是	否
删除桶标签	DELETE	tagging	是	否

操作名	HTTP请求方法	特殊操作符（子资源）	是否需要桶名	是否需要对象名
上传对象	PUT	N/A	是	是
追加上传	POST	append	是	是
下载对象	GET	N/A	是	是
复制对象	PUT	N/A	是	是
删除对象	DELETE	N/A	是	是
批量删除对象	POST	delete	是	是
获取对象属性	HEAD	N/A	是	是
设置对象访问权限	PUT	acl	是	是
查看对象访问权限	GET	acl	是	是
初始化分段上传任务	POST	uploads	是	是
上传段	PUT	N/A	是	是
复制段	PUT	N/A	是	是
列举已上传的段	GET	N/A	是	是
合并段	POST	N/A	是	是
取消分段上传任务	DELETE	N/A	是	是
恢复归档存储对象	POST	restore	是	是

通过OBS Node.js SDK生成临时URL访问OBS的步骤如下：

步骤1 通过ObsClient.createSignedUrlSync生成带签名信息的URL。

步骤2 使用任意HTTP库发送HTTP/HTTPS请求，访问OBS服务。

----结束

注意

如果遇到跨域报错、签名不匹配问题，请参考以下步骤排查问题：

1. 未配置跨域，需要在控制台配置CORS规则，请参考**配置桶允许跨域请求**。
2. 签名计算问题，请参考**URL中携带签名**排查签名参数是否正确；比如上传对象功能，后端将Content-Type参与计算签名生成授权URL，但是前端使用授权URL时没有设置Content-Type字段或者传入错误的值，此时会出现跨域错误。解决方案为：Content-Type字段前后端保持一致。

以下代码展示了如何使用临时URL进行授权访问，包括：创建桶、上传对象、下载对象、列举对象、删除对象。

创建桶

```
// 引入obs库
// 使用npm安装
var ObsClient = require('esdk-obs-nodejs');
// 使用源码安装
// var ObsClient = require('./lib/obs');
var https = require('https');
var urlLib = require('url');
var crypto = require('crypto');

// 创建ObsClient实例
var obsClient = new ObsClient({
    //推荐通过环境变量获取AKSK，这里也可以使用其他外部引入方式传入，如果使用硬编码可能会存在泄露风险。
    //您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    access_key_id: process.env.ACCESS_KEY_ID,
    secret_access_key: process.env.SECRET_ACCESS_KEY,
    server : 'https://your-endpoint'
});

let bucketName = 'bucketname';
let method = 'PUT';
let res = obsClient.createSignedUrlSync({Method : method, Bucket : bucketName});
let location = 'your-location';
let content  = `<CreateBucketConfiguration><LocationConstraint>${location}</LocationConstraint></CreateBucketConfiguration>`;

// 使用PUT请求创建桶
var url = urlLib.parse(res.SignedUrl);
var req = https.request({
    method : method,
    host : url.hostname,
    port : url.port,
    path : url.path,
    rejectUnauthorized : false,
    headers : res.ActualSignedRequestHeaders || {}
});


console.log('Creating bucket using url:' + res.SignedUrl);

req.on('response',   (serverback) => {
    var buffers = [];
    serverback.on('data', (data) => {
        buffers.push(data);
    }).on('end', () => {

        if(serverback.statusCode < 300){
            console.log('Creating bucket using temporary signature succeed.');
        }else{
            console.log('Creating bucket using temporary signature failed!');
            console.log('status:' + serverback.statusCode);
            console.log('\n');
        }
        buffers = Buffer.concat(buffers);
        if(buffers.length > 0){
            console.log(buffers.toString());
        }
        console.log('\n');
    });
}).on('error',(err) => {
    console.log('Creating bucket using temporary signature failed!');
    console.log(err);
    console.log('\n');
});

if(content){
    req.write(content);
```



```
}
req.end();
```

上传对象

```
// 引入obs库
// 使用npm安装
var ObsClient = require('esdk-obs-nodejs');
// 使用源码安装
// var ObsClient = require('./lib/obs');
var https = require('https');
var urlLib = require('url');
var crypto = require('crypto');

// 创建ObsClient实例
var obsClient = new ObsClient({
    //推荐通过环境变量获取AKSK，这里也可以使用其他外部引入方式传入，如果使用硬编码可能会存在泄露风险。
    //您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    access_key_id: process.env.ACCESS_KEY_ID,
    secret_access_key: process.env.SECRET_ACCESS_KEY,
    server : 'https://your-endpoint'
});

let bucketName = 'bucketname';
let objectKey = 'objectname';
let method = 'PUT';
let res = obsClient.createSignedUrlSync({Method : method, Bucket : bucketName, Key: objectKey, Expires: 3600});
let content  = 'Hello OBS';

// 使用PUT请求上传对象
var url = urlLib.parse(res.SignedUrl);
var req = https.request({
    method : method,
    host : url.hostname,
    port : url.port,
    path : url.path,
    rejectUnauthorized : false,
    headers : res.ActualSignedRequestHeaders || {}
});


console.log('Creating object using url:' + res.SignedUrl);

req.on('response',   (serverback) => {
    var buffers = [];
    serverback.on('data', (data) => {
        buffers.push(data);
    }).on('end', () => {

        if(serverback.statusCode < 300){
            console.log('Creating object using temporary signature succeed.');
        }else{
            console.log('Creating object using temporary signature failed!');
            console.log('status:' + serverback.statusCode);
            console.log('\n');
        }
        buffers = Buffer.concat(buffers);
        if(buffers.length > 0){
            console.log(buffers.toString());
        }
        console.log('\n');
    });
}).on('error',(err) => {
    console.log('Creating object using temporary signature failed!');
    console.log(err);
    console.log('\n');
```



```
});

if(content){
    req.write(content);
}
req.end();
```

下载对象

```
// 引入obs库
// 使用npm安装
var ObsClient = require('esdk-obs-nodejs');
// 使用源码安装
// var ObsClient = require('./lib/obs');
var https = require('https');
var urlLib = require('url');
var crypto = require('crypto');

// 创建ObsClient实例
var obsClient = new ObsClient({
    //推荐通过环境变量获取AKSK，这里也可以使用其他外部引入方式传入，如果使用硬编码可能会存在泄露风险。
    //您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    access_key_id: process.env.ACCESS_KEY_ID,
    secret_access_key: process.env.SECRET_ACCESS_KEY,
    server : 'https://your-endpoint'
});

let bucketName = 'bucketname';
let objectKey = 'objectname';
let method = 'GET';
let res = obsClient.createSignedUrlSync({Method : method, Bucket : bucketName, Key: objectKey, Expires: 3600});

// 使用GET请求下载对象
var url = urlLib.parse(res.SignedUrl);
var req = https.request({
    method : method,
    host : url.hostname,
    port : url.port,
    path : url.path,
    rejectUnauthorized : false,
    headers : res.ActualSignedRequestHeaders || {}
});


console.log('Creating object using url:' + res.SignedUrl);

req.on('response',  (serverback) => {
    var buffers = [];
    serverback.on('data', (data) => {
        buffers.push(data);
    }).on('end', () => {

        if(serverback.statusCode < 300){
            console.log('Getting object using temporary signature succeed.');
        }else{
            console.log('Getting object using temporary signature failed!');
            console.log('status:' + serverback.statusCode);
            console.log('\n');
        }
        buffers = Buffer.concat(buffers);
        if(buffers.length > 0){
            console.log(buffers.toString());
        }
        console.log('\n');
    });
}).on('error',(err) => {
```



```
    console.log('Getting object using temporary signature failed!');
    console.log(err);
    console.log('\n');
});

req.end();
```

列举对象

```
// 引入obs库
// 使用npm安装
var ObsClient = require('esdk-obs-nodejs');
// 使用源码安装
// var ObsClient = require('./lib/obs');
var https = require('https');
var urlLib = require('url');
var crypto = require('crypto');

// 创建ObsClient实例
var obsClient = new ObsClient({
    //推荐通过环境变量获取AKSK，这里也可以使用其他外部引入方式传入，如果使用硬编码可能会存在泄露风险。
    //您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    access_key_id: process.env.ACCESS_KEY_ID,
    secret_access_key: process.env.SECRET_ACCESS_KEY,
    server : 'https://your-endpoint'
});

let bucketName = 'bucketname';
let method = 'GET';
let res = obsClient.createSignedUrlSync({Method : method, Bucket : bucketName, Expires: 3600});

// 使用GET请求获取对象列表
var url = urlLib.parse(res.SignedUrl);
var req = https.request({
    method : method,
    host : url.hostname,
    port : url.port,
    path : url.path,
    rejectUnauthorized : false,
    headers : res.ActualSignedRequestHeaders || {}
});


console.log('Listing object using url:' + res.SignedUrl);

req.on('response',   (serverback) => {
    var buffers = [];
    serverback.on('data', (data) => {
        buffers.push(data);
    }).on('end', () => {

        if(serverback.statusCode < 300){
            console.log('Listing object using temporary signature succeed.');
        }else{
            console.log('Listing object using temporary signature failed!');
            console.log('status:' + serverback.statusCode);
            console.log('\n');
        }
        buffers = Buffer.concat(buffers);
        if(buffers.length > 0){
            console.log(buffers.toString());
        }
        console.log('\n');
    });
}).on('error',(err) => {
    console.log('Listing object using temporary signature failed!');
    console.log(err);
```



```
    console.log('\n');
});

req.end();
```

删除对象

```
// 引入obs库
// 使用npm安装
var ObsClient = require('esdk-obs-nodejs');
// 使用源码安装
// var ObsClient = require('./lib/obs');
var https = require('https');
var urlLib = require('url');
var crypto = require('crypto');

// 创建ObsClient实例
var obsClient = new ObsClient({
    //推荐通过环境变量获取AKSK，这里也可以使用其他外部引入方式传入，如果使用硬编码可能会存在泄露风险。
    //您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    access_key_id: process.env.ACCESS_KEY_ID,
    secret_access_key: process.env.SECRET_ACCESS_KEY,
    server : 'https://your-endpoint'
});

let bucketName = 'bucketname';
let objectKey = 'objectname';
let method = 'DELETE';
let res = obsClient.createSignedUrlSync({Method : method, Bucket : bucketName, Key: objectKey, Expires: 3600});

// 使用DELETE请求删除对象
var url = urlLib.parse(res.SignedUrl);
var req = https.request({
    method : method,
    host : url.hostname,
    port : url.port,
    path : url.path,
    rejectUnauthorized : false,
    headers : res.ActualSignedRequestHeaders || {}
});


console.log('Deleting object using url:' + res.SignedUrl);

req.on('response',  (serverback) => {
    var buffers = [];
    serverback.on('data', (data) => {
        buffers.push(data);
    }).on('end', () => {

        if(serverback.statusCode < 300){
            console.log('Deleting object using temporary signature succeed.');
        }else{
            console.log('Deleting object using temporary signature failed!');
            console.log('status:' + serverback.statusCode);
            console.log('\n');
        }
        buffers = Buffer.concat(buffers);
        if(buffers.length > 0){
            console.log(buffers.toString());
        }
        console.log('\n');
    });
}).on('error',(err) => {
    console.log('Deleting object using temporary signature failed!');
    console.log(err);
```



```
    console.log('\n');
});

req.end();
```


10 多版本控制

10.1 多版本控制简介

须知

开发过程中，您有任何问题可以在github上**提交issue**。**接口参考文档**详细介绍了每个接口的参数和使用方法。

OBS支持保存一个对象的多个版本，使您更方便地检索和还原各个版本，在意外操作或应用程序故障时快速恢复数据。

更多关于多版本控制的内容请参见**多版本控制**。

10.2 设置桶多版本状态

须知

开发过程中，您有任何问题可以在github上**提交issue**。**接口参考文档**详细介绍了每个接口的参数和使用方法。

您可以通过ObsClient.setBucketVersioning设置桶的多版本状态。OBS中的桶支持两种多版本状态：

多版本状态	说明	OBS服务端对应值
启用状态	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（ObsClient.listObjects）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（ObsClient.listVersions）。 5. 除了删除标记外，每个版本的对象存储均需计费。	Enabled
暂停状态	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. 除了删除标记外，每个版本的对象存储均需计费。	Suspended

以下代码展示了如何设置桶的多版本状态：

```
// 引入obs库
// 使用npm安装
var ObsClient = require('esdk-obs-nodejs');
// 使用源码安装
// var ObsClient = require('./lib/obs');

// 创建ObsClient实例
var obsClient = new ObsClient({
    //推荐通过环境变量获取AKSK，这里也可以使用其他外部引入方式传入，如果使用硬编码可能会存在泄露风险。
    //您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    access_key_id: process.env.ACCESS_KEY_ID,
    secret_access_key: process.env.SECRET_ACCESS_KEY,
    server : 'https://your-endpoint'
});

// 启用桶多版本状态
obsClient.setBucketVersioning({
    Bucket : 'bucketname',
    VersionStatus : 'Enabled'
}, (err, result) => {
    if(err){
        console.log('Error-->' + err);
    }else{
        console.log('Status-->' + result.CommonMsg.Status);
```



```
    }
});

// 暂停桶多版本状态
obsClient.setBucketVersioning({
    Bucket : 'bucketname',
    VersionStatus : 'Suspended'
}, (err, result) => {
    if(err){
        console.log('Error-->' + err);
    }else{
        console.log('Status-->' + result.CommonMsg.Status);
    }
});
```

说明

使用**VersionStatus参数**指定桶的多版本状态。

10.3 查看桶多版本状态

须知

开发过程中，您有任何问题可以在github上**提交issue**。**接口参考文档**详细介绍了每个接口的参数和使用方法。

您可以通过ObsClient.getBucketVersioning查看桶的多版本状态。以下代码展示了如何查看桶的多版本状态：

```
// 引入obs库
// 使用npm安装
var ObsClient = require('esdk-obs-nodejs');
// 使用源码安装
// var ObsClient = require('./lib/obs');

// 创建ObsClient实例
var obsClient = new ObsClient({
    //推荐通过环境变量获取AKSK，这里也可以使用其他外部引入方式传入，如果使用硬编码可能会存在泄露风险。
    //您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    access_key_id: process.env.ACCESS_KEY_ID,
    secret_access_key: process.env.SECRET_ACCESS_KEY,
    server : 'https://your-endpoint'
});

// 启用桶多版本
obsClient.getBucketVersioning({
    Bucket : 'bucketname'
}, (err, result) => {
    if(err){
        console.log('Error-->' + err);
    }else{
        console.log('Status-->' + result.CommonMsg.Status);
        if(result.CommonMsg.Status < 300 && result.InterfaceResult){
            console.log('VersionStatus-->' + result.InterfaceResult.VersionStatus);
        }
    }
});
```


10.4 获取多版本对象

须知

开发过程中，您有任何问题可以在github上**提交issue**。**接口参考文档**详细介绍了每个接口的参数和使用方法。

您可以通过ObsClient.getObject接口指定**VersionId参数**来获取多版本对象，示例代码如下：

```
// 引入obs库
// 使用npm安装
var ObsClient = require('esdk-obs-nodejs');
// 使用源码安装
// var ObsClient = require('./lib/obs');

// 创建ObsClient实例
var obsClient = new ObsClient({
    //推荐通过环境变量获取AKSK，这里也可以使用其他外部引入方式传入，如果使用硬编码可能会存在泄露风险。
    //您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    access_key_id: process.env.ACCESS_KEY_ID,
    secret_access_key: process.env.SECRET_ACCESS_KEY,
    server : 'https://your-endpoint'
});

// 设置versionId获取多版本对象
obsClient.getObject({
    Bucket : 'bucketname',
    Key : 'objectname',
    VersionId : 'versionid'
}, (err, result) => {
    if(err){
        console.log('Error-->' + err);
    }else{
        console.log('Status-->' + result.CommonMsg.Status);
        if(result.CommonMsg.Status < 300 && result.InterfaceResult){
            console.log('Content-->' + result.InterfaceResult.Content.toString());
        }
    }
});
```

说明

如果版本号为空则默认下载最新版本的对象。

10.5 复制多版本对象

须知

开发过程中，您有任何问题可以在github上**提交issue**。**接口参考文档**详细介绍了每个接口的参数和使用方法。

您可以通过ObsClient.copyObject接口在**CopySource参数**中指定待复制对象的versionId来复制多版本对象，示例代码如下：

```
// 引入obs库
// 使用npm安装
var ObsClient = require('esdk-obs-nodejs');
// 使用源码安装
// var ObsClient = require('./lib/obs');

// 创建ObsClient实例
var obsClient = new ObsClient({
    //推荐通过环境变量获取AKSK，这里也可以使用其他外部引入方式传入，如果使用硬编码可能会存在泄露风险。
    //您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    access_key_id: process.env.ACCESS_KEY_ID,
    secret_access_key: process.env.SECRET_ACCESS_KEY,
    server : 'https://your-endpoint'
});

obsClient.copyObject({
    Bucket : 'destbucketname',
    Key : 'destobjectname',
    // 设置要复制对象的版本号
    CopySource : 'sourcebucket/sourceobjectname?versionId=versionid'
}, (err, result) => {
    if(err){
        console.log('Error-->' + err);
    }else{
        console.log('Status-->' + result.CommonMsg.Status);
    }
});
```

10.6 恢复多版本归档存储对象

须知

开发过程中，您有任何问题可以在github上**提交issue**。**接口参考文档**详细介绍了每个接口的参数和使用方法。

您可以通过ObsClient.restoreObject接口指定**VersionId参数**来恢复多版本归档存储对象，示例代码如下：

```
// 引入obs库
// 使用npm安装
var ObsClient = require('esdk-obs-nodejs');
// 使用源码安装
// var ObsClient = require('./lib/obs');

// 创建ObsClient实例
var obsClient = new ObsClient({
    //推荐通过环境变量获取AKSK，这里也可以使用其他外部引入方式传入，如果使用硬编码可能会存在泄露风险。
    //您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    access_key_id: process.env.ACCESS_KEY_ID,
    secret_access_key: process.env.SECRET_ACCESS_KEY,
    server : 'https://your-endpoint'
});

obsClient.restoreObject({
    Bucket : 'bucketname',
    Key : 'objectname',
    VersionId : 'versionid',
    Days : 1,
    // 使用快速恢复方式，恢复多版本对象
```



```
    Tier : obsClient.enums.RestoreTierExpedited
}, (err, result) => {
    if(err){
        console.log('Error-->' + err);
    }else{
        console.log('Status-->' + result.CommonMsg.Status);
    }
});
```

⚠ 注意

重复恢复归档存储数据时在延长恢复有效期的同时，也将会对恢复时产生的恢复费用进行重复收取。产生的标准存储类别的对象副本有效期将会延长，并且收取延长时间段产生的标准存储副本费用。

10.7 列举多版本对象

须知

开发过程中，您有任何问题可以在github上**提交issue**。**接口参考文档**详细介绍了每个接口的参数和使用方法。

您可以通过ObsClient.listVersions列举多版本对象。

该接口可设置的参数如下：

参数	作用
Prefix	限定返回的对象名必须带有Prefix前缀。
KeyMarker	列举多版本对象的起始位置，返回的对象列表将是对象名按照字典序排序后该参数以后的所有对象。
MaxKeys	列举多版本对象的最大数目，取值范围为1~1000，当超出范围时，按照默认的1000进行处理。
Delimiter	用于对对象名进行分组的字符。对于对象名中包含Delimiter的对象，其对象名（如果请求中指定了Prefix，则此处的对象名需要去掉Prefix）中从首字符至第一个Delimiter之间的字符串将作为一个分组并作为CommonPrefix返回。
VersionIdMarker	与KeyMarker配合使用，返回的对象列表将是对象名和版本号按照字典序排序后该参数以后的所有对象。

说明

- 如果VersionIdMarker不是KeyMarker的一个版本号，则该参数无效。
- ObsClient.listVersions返回结果包含多版本对象和对象删除标记。

简单列举

以下代码展示如何简单列举多版本对象，最多返回1000个对象：

```
// 引入obs库
// 使用npm安装
var ObsClient = require('esdk-obs-nodejs');
// 使用源码安装
// var ObsClient = require('./lib/obs');

// 创建ObsClient实例
var obsClient = new ObsClient({
    //推荐通过环境变量获取AKSK，这里也可以使用其他外部引入方式传入，如果使用硬编码可能会存在泄露风险。
    //您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    access_key_id: process.env.ACCESS_KEY_ID,
    secret_access_key: process.env.SECRET_ACCESS_KEY,
    server : 'https://your-endpoint'
});

obsClient.listVersions({
    Bucket : 'bucketname'
}, (err, result) => {
    if(err){
        console.log('Error-->' + err);
    }else{
        console.log('Status-->' + result.CommonMsg.Status);
        if(result.CommonMsg.Status < 300 && result.InterfaceResult){
            // 获取多版本对象
            for(let j=0;j<result.InterfaceResult.Versions.length;j++){
                console.log('Version[' + j +  ']:');
                console.log('Key-->' + result.InterfaceResult.Versions[j]['Key']);
                console.log('VersionId-->' + result.InterfaceResult.Versions[j]['VersionId']);
                console.log('Owner[ID]-->' + result.InterfaceResult.Versions[j]['Owner']['ID']);
            }
            // 获取对象删除标记
            for(let i=0;i<result.InterfaceResult.DeleteMarkers.length;i++){
                console.log('DeleteMarker[' + i +  ']:');
                console.log('Key-->' + result.InterfaceResult.DeleteMarkers[i]['Key']);
                console.log('VersionId-->' + result.InterfaceResult.DeleteMarkers[i]['VersionId']);
                console.log('Owner[ID]-->' + result.InterfaceResult.DeleteMarkers[i]['Owner']['ID']);
            }
        }
    }
});
```

说明

- 每次至多返回1000个多版本对象，如果指定桶包含的对象数量大于1000，则返回结果中InterfaceResult.IsTruncated为true表明本次没有返回全部对象，并可通过InterfaceResult.NextKeyMarker和InterfaceResult.NextVersionIdMarker获取下次列举的起始位置。
- 如果想获取指定桶包含的所有多版本对象，可以采用分页列举的方式。

指定数目列举

以下代码展示如何指定数目列举多版本对象：

```
// 引入obs库
// 使用npm安装
var ObsClient = require('esdk-obs-nodejs');
// 使用源码安装
// var ObsClient = require('./lib/obs');

// 创建ObsClient实例
var obsClient = new ObsClient({
```

```
    //推荐通过环境变量获取AKSK，这里也可以使用其他外部引入方式传入，如果使用硬编码可能会存在泄露风险。
    //您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    access_key_id: process.env.ACCESS_KEY_ID,
    secret_access_key: process.env.SECRET_ACCESS_KEY,
    server : 'https://your-endpoint'
});

obsClient.listVersions({
    Bucket : 'bucketname',
    MaxKeys : 100
}, (err, result) => {
    if(err){
        console.log('Error-->' + err);
    }else{
        console.log('Status-->' + result.CommonMsg.Status);
        if(result.CommonMsg.Status < 300 && result.InterfaceResult){
            // 获取多版本对象
            for(let j=0;j<result.InterfaceResult.Versions.length;j++){
                console.log('Version[' + j +  ']:');
                console.log('Key-->' + result.InterfaceResult.Versions[j]['Key']);
                console.log('VersionId-->' + result.InterfaceResult.Versions[j]['VersionId']);
                console.log('Owner[ID]-->' + result.InterfaceResult.Versions[j]['Owner']['ID']);
            }
            // 获取对象删除标记
            for(let i=0;i<result.InterfaceResult.DeleteMarkers.length;i++){
                console.log('DeleteMarker[' + i +  ']:');
                console.log('Key-->' + result.InterfaceResult.DeleteMarkers[i]['Key']);
                console.log('VersionId-->' + result.InterfaceResult.DeleteMarkers[i]['VersionId']);
                console.log('Owner[ID]-->' + result.InterfaceResult.DeleteMarkers[i]['Owner']['ID']);
            }
        }
    }
});
```

指定前缀列举

以下代码展示如何指定前缀列举多版本对象：

```
// 引入obs库
// 使用npm安装
var ObsClient = require('esdk-obs-nodejs');
// 使用源码安装
// var ObsClient = require('./lib/obs');

// 创建ObsClient实例
var obsClient = new ObsClient({
    //推荐通过环境变量获取AKSK，这里也可以使用其他外部引入方式传入，如果使用硬编码可能会存在泄露风险。
    //您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    access_key_id: process.env.ACCESS_KEY_ID,
    secret_access_key: process.env.SECRET_ACCESS_KEY,
    server : 'https://your-endpoint'
});
;

// 设置列举带有prefix前缀的100个多版本对象
obsClient.listVersions({
    Bucket : 'bucketname',
    MaxKeys : 100,
    Prefix : 'prefix'
}, (err, result) => {
    if(err){
        console.log('Error-->' + err);
    }else{
        console.log('Status-->' + result.CommonMsg.Status);
        if(result.CommonMsg.Status < 300 && result.InterfaceResult){
```

```
            // 获取多版本对象
            for(let j=0;j<result.InterfaceResult.Versions.length;j++){
                console.log('Version[' + j +  ']:');
                console.log('Key-->' + result.InterfaceResult.Versions[j]['Key']);
                console.log('VersionId-->' + result.InterfaceResult.Versions[j]['VersionId']);
                console.log('Owner[ID]-->' + result.InterfaceResult.Versions[j]['Owner']['ID']);
            }
            // 获取对象删除标记
            for(let i=0;i<result.InterfaceResult.DeleteMarkers.length;i++){
                console.log('DeleteMarker[' + i +  ']:');
                console.log('Key-->' + result.InterfaceResult.DeleteMarkers[i]['Key']);
                console.log('VersionId-->' + result.InterfaceResult.DeleteMarkers[i]['VersionId']);
                console.log('Owner[ID]-->' + result.InterfaceResult.DeleteMarkers[i]['Owner']['ID']);
            }
        }
    }
});
```

指定起始位置列举

以下代码展示如何指定起始位置列举多版本对象：

```
// 引入obs库
// 使用npm安装
var ObsClient = require('esdk-obs-nodejs');
// 使用源码安装
// var ObsClient = require('./lib/obs');

// 创建ObsClient实例
var obsClient = new ObsClient({
    //推荐通过环境变量获取AKSK，这里也可以使用其他外部引入方式传入，如果使用硬编码可能会存在泄露风险。
    //您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    access_key_id: process.env.ACCESS_KEY_ID,
    secret_access_key: process.env.SECRET_ACCESS_KEY,
    server : 'https://your-endpoint'
});

// 设置列举对象名字典序在"test"之后的100个多版本对象
obsClient.listVersions({
    Bucket : 'bucketname',
    MaxKeys : 100,
    KeyMarker : 'test'
}, (err, result) => {
    if(err){
        console.log('Error-->' + err);
    }else{
        console.log('Status-->' + result.CommonMsg.Status);
        if(result.CommonMsg.Status < 300 && result.InterfaceResult){
            // 获取多版本对象
            for(let j=0;j<result.InterfaceResult.Versions.length;j++){
                console.log('Version[' + j +  ']:');
                console.log('Key-->' + result.InterfaceResult.Versions[j]['Key']);
                console.log('VersionId-->' + result.InterfaceResult.Versions[j]['VersionId']);
                console.log('Owner[ID]-->' + result.InterfaceResult.Versions[j]['Owner']['ID']);
            }
            // 获取对象删除标记
            for(let i=0;i<result.InterfaceResult.DeleteMarkers.length;i++){
                console.log('DeleteMarker[' + i +  ']:');
                console.log('Key-->' + result.InterfaceResult.DeleteMarkers[i]['Key']);
                console.log('VersionId-->' + result.InterfaceResult.DeleteMarkers[i]['VersionId']);
                console.log('Owner[ID]-->' + result.InterfaceResult.DeleteMarkers[i]['Owner']['ID']);
            }
        }
    }
});
```

分页列举全部多版本对象

以下代码展示分页列举全部多版本对象：

```
// 引入obs库
// 使用npm安装
var ObsClient = require('esdk-obs-nodejs');
// 使用源码安装
// var ObsClient = require('./lib/obs');

// 创建ObsClient实例
var obsClient = new ObsClient({
    //推荐通过环境变量获取AKSK，这里也可以使用其他外部引入方式传入，如果使用硬编码可能会存在泄露风险。
    //您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    access_key_id: process.env.ACCESS_KEY_ID,
    secret_access_key: process.env.SECRET_ACCESS_KEY,
    server : 'https://your-endpoint'
});

var listAll = (keyMarker, versionIdMarker) => {
    obsClient.listVersions({
        Bucket : 'bucketname',
        MaxKeys : 100,
        KeyMarker : keyMarker,
        VersionIdMarker : versionIdMarker
    }, (err, result) => {
        if(err){
            console.log('Error-->' + err);
        }else{
            console.log('Status-->' + result.CommonMsg.Status);
            if(result.CommonMsg.Status < 300 && result.InterfaceResult){
                // 获取多版本对象
                for(let j=0;j<result.InterfaceResult.Versions.length;j++){
                    console.log('Version[' + j +  ']:');
                    console.log('Key-->' + result.InterfaceResult.Versions[j]['Key']);
                    console.log('VersionId-->' + result.InterfaceResult.Versions[j]['VersionId']);
                    console.log('Owner[ID]-->' + result.InterfaceResult.Versions[j]['Owner']['ID']);
                }
                // 获取对象删除标记
                for(let i=0;i<result.InterfaceResult.DeleteMarkers.length;i++){
                    console.log('DeleteMarker[' + i +  ']:');
                    console.log('Key-->' + result.InterfaceResult.DeleteMarkers[i]['Key']);
                    console.log('VersionId-->' + result.InterfaceResult.DeleteMarkers[i]['VersionId']);
                    console.log('Owner[ID]-->' + result.InterfaceResult.DeleteMarkers[i]['Owner']['ID']);
                }
                if(result.InterfaceResult.IsTruncated === 'true'){
                    listAll(result.InterfaceResult.NextKeyMarker, result.InterfaceResult.NextVersionIdMarker);
                }
            }
        }
    });
};

listAll();
```

列举文件夹中的所有多版本对象

OBS本身是没有文件夹的概念的，桶中存储的元素只有对象。文件夹对象实际上是一个大小为0且对象名以“/”结尾的对象，将这个文件夹对象名作为前缀，即可模拟列举文件夹中对象的功能。以下代码展示如何列举文件夹中的多版本对象：

```
// 引入obs库
// 使用npm安装
var ObsClient = require('esdk-obs-nodejs');
// 使用源码安装
```



```
// var ObsClient = require('./lib/obs');

// 创建ObsClient实例
var obsClient = new ObsClient({
    //推荐通过环境变量获取AKSK，这里也可以使用其他外部引入方式传入，如果使用硬编码可能会存在泄露风险。
    //您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    access_key_id: process.env.ACCESS_KEY_ID,
    secret_access_key: process.env.SECRET_ACCESS_KEY,
    server : 'https://your-endpoint'
});

var listAll = (keyMarker, versionIdMarker) => {
    obsClient.listVersions({
        Bucket : 'bucketname',
        MaxKeys : 100,
        KeyMarker : keyMarker,
        VersionIdMarker : versionIdMarker,
        // 设置文件夹对象名"dir/"为前缀
        Prefix : 'dir/'
    }, (err, result) => {
        if(err){
            console.log('Error-->' + err);
        }else{
            console.log('Status-->' + result.CommonMsg.Status);
            if(result.CommonMsg.Status < 300 && result.InterfaceResult){
                // 获取多版本对象
                for(let j=0;j<result.InterfaceResult.Versions.length;j++){
                    console.log('Version[' + j +  ']:');
                    console.log('Key-->' + result.InterfaceResult.Versions[j]['Key']);
                    console.log('VersionId-->' + result.InterfaceResult.Versions[j]['VersionId']);
                    console.log('Owner[ID]-->' + result.InterfaceResult.Versions[j]['Owner']['ID']);
                }
                // 获取对象删除标记
                for(let i=0;i<result.InterfaceResult.DeleteMarkers.length;i++){
                    console.log('DeleteMarker[' + i +  ']:');
                    console.log('Key-->' + result.InterfaceResult.DeleteMarkers[i]['Key']);
                    console.log('VersionId-->' + result.InterfaceResult.DeleteMarkers[i]['VersionId']);
                    console.log('Owner[ID]-->' + result.InterfaceResult.DeleteMarkers[i]['Owner']['ID']);
                }
                if(result.InterfaceResult.IsTruncated === 'true'){
                    listAll(result.InterfaceResult.NextKeyMarker, result.InterfaceResult.NextVersionIdMarker);
                }
            }
        }
    });
};

listAll();
```

按文件夹分组列举所有多版本对象

以下代码展示如何按文件夹分组，列举桶内所有多版本对象：

```
// 引入obs库
// 使用npm安装
var ObsClient = require('esdk-obs-nodejs');
// 使用源码安装
// var ObsClient = require('./lib/obs');

// 创建ObsClient实例
var obsClient = new ObsClient({
    //推荐通过环境变量获取AKSK，这里也可以使用其他外部引入方式传入，如果使用硬编码可能会存在泄露风险。
    //您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    access_key_id: process.env.ACCESS_KEY_ID,
```

```
    secret_access_key: process.env.SECRET_ACCESS_KEY,
    server : 'https://your-endpoint'
});

obsClient.listVersions({
    Bucket : 'bucketname',
    // 设置文件夹分隔符"/"
    Delimiter : '/'
}, (err, result) => {
    if(err){
        console.log('Error-->' + err);
    }else{
        console.log('Objects in the root directory:');
        if(result.CommonMsg.Status < 300 && result.InterfaceResult){
            // 获取多版本对象.
            for(let j=0;j<result.InterfaceResult.Versions.length;j++){
                console.log('Version[' + j +  ']:');
                console.log('Key-->' + result.InterfaceResult.Versions[j]['Key']);
                console.log('VersionId-->' + result.InterfaceResult.Versions[j]['VersionId']);
                console.log('Owner[ID]-->' + result.InterfaceResult.Versions[j]['Owner']['ID']);
            }
            // 获取对象删除标记.
            for(let i=0;i<result.InterfaceResult.DeleteMarkers.length;i++){
                console.log('DeleteMarker[' + i +  ']:');
                console.log('Key-->' + result.InterfaceResult.DeleteMarkers[i]['Key']);
                console.log('VersionId-->'+ result.InterfaceResult.DeleteMarkers[i]['VersionId']);
                console.log('Owner[ID]-->' + result.InterfaceResult.DeleteMarkers[i]['Owner']['ID']);
            }
        }
        var listVersionsByPrefix = (commonPrefixes) => {
            for(let n=0;n<commonPrefixes.length;n++){
                obsClient.listVersions({
                    Bucket : 'bucketname',
                    Delimiter : '/',
                    Prefix: commonPrefixes[n]['Prefix']
                }, (err, result) => {
                    console.log('Objects in folder [' + commonPrefixes[n]['Prefix'] + ']:');
                    if(result.CommonMsg.Status < 300 && result.InterfaceResult){
                      // 获取多版本对象
                      for(let j=0;j<result.InterfaceResult.Versions.length;j++){
                          console.log('Version[' + j +  ']:');
                          console.log('Key-->' + result.InterfaceResult.Versions[j]['Key']);
                          console.log('VersionId-->' + result.InterfaceResult.Versions[j]['VersionId']);
                          console.log('Owner[ID]-->' + result.InterfaceResult.Versions[j]['Owner']['ID']);
                      }
                      // 获取对象删除标记
                      for(let i=0;i<result.InterfaceResult.DeleteMarkers.length;i++){
                          console.log('DeleteMarker[' + i +  ']:');
                          console.log('Key-->' + result.InterfaceResult.DeleteMarkers[i]['Key']);
                          console.log('VersionId-->' + result.InterfaceResult.DeleteMarkers[i]
['VersionId']);
                          console.log('Owner[ID]-->' + result.InterfaceResult.DeleteMarkers[i]['Owner']
['ID']);
                      }
                      console.log('\n');
                      if(result.InterfaceResult.CommonPrefixes &&
result.InterfaceResult.CommonPrefixes.length > 0){
                          listVersionsByPrefix(result.InterfaceResult.CommonPrefixes);
                      }
                    }
                });
            }
        };
        listVersionsByPrefix(result.InterfaceResult.CommonPrefixes);
    }
});
```


说明

- 以上代码示例没有考虑文件夹中多版本对象数超过1000个的情况。
- 由于是需要列举出文件夹中的对象和子文件夹，且文件夹对象总是以“/”结尾，因此Delimiter总是为“/”。
- 每次递归的返回结果中InterfaceResult.Versions包含的是文件夹中的对象；InterfaceResult.DeleteMarkers包含的是文件夹中的删除标记；InterfaceResult.CommonPrefixes包含的是文件夹的子文件夹。

10.8 多版本对象权限

须知

开发过程中，您有任何问题可以在github上**提交issue**。**接口参考文档**详细介绍了每个接口的参数和使用方法。

设置多版本对象访问权限

您可以通过ObsClient.setObjectAcl接口指定**VersionId参数**设置多版本对象的访问权限，示例代码如下：

```
// 引入obs库
// 使用npm安装
var ObsClient = require('esdk-obs-nodejs');
// 使用源码安装
// var ObsClient = require('./lib/obs');

// 创建ObsClient实例
var obsClient = new ObsClient({
    //推荐通过环境变量获取AKSK，这里也可以使用其他外部引入方式传入，如果使用硬编码可能会存在泄露风险。
    //您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    access_key_id: process.env.ACCESS_KEY_ID,
    secret_access_key: process.env.SECRET_ACCESS_KEY,
    server : 'https://your-endpoint'
});

obsClient.setObjectAcl({
    Bucket : 'bucketname',
    Key : 'objectname',
    VersionId : 'versionid',
    // 通过预定义访问策略设置多版本对象访问权限为公共读
    ACL : obsClient.enums.AclPublicRead
}, (err, result) => {
    if(err){
        console.log('Error-->' + err);
    }else{
        console.log('Status-->' + result.CommonMsg.Status);
    }
});

obsClient.setObjectAcl({
    Bucket : 'bucketname',
    Key : 'objectname',
    VersionId : 'versionid',
    // 设置对象所有者
    Owner:{'ID':'ownerid'},
    Grants:{
      Grant:[
```

```
            // 为所有用户设置读权限
            { Grantee:{Type : 'Group', URI : obsClient.enums.GroupAllUsers}, Permission :
obsClient.enums.PermissionRead},
            // 为所有用户设置写ACP权限
            { Grantee:{Type : 'Group', URI : obsClient.enums.GroupAllUsers}, Permission :
obsClient.enums.PermissionWriteAcp}
        ]
    }
}, (err, result) => {
    if(err){
        console.log('Error-->' + err);
    }else{
        console.log('Status-->' + result.CommonMsg.Status);
    }
});
```

说明

- 使用**Owner参数**指定对象的所有者信息；使用**Grants参数**指定被授权的用户信息。
- ACL中需要填写的所有者（Owner）或者被授权用户（Grantee）的ID，是指用户的账户ID，可通过OBS控制台“我的凭证”页面查看。
- 当前OBS对象支持的可被授权的用户组为：
  - 所有用户：ObsClient.enums.GroupAllUsers

获取多版本对象访问权限

您可以通过ObsClient.getObjectAcl接口指定**VersionId参数**获取多版本对象的访问权限，示例代码如下：

```
// 引入obs库
// 使用npm安装
var ObsClient = require('esdk-obs-nodejs');
// 使用源码安装
// var ObsClient = require('./lib/obs');

// 创建ObsClient实例
var obsClient = new ObsClient({
    //推荐通过环境变量获取AKSK，这里也可以使用其他外部引入方式传入，如果使用硬编码可能会存在泄露风险。
    //您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    access_key_id: process.env.ACCESS_KEY_ID,
    secret_access_key: process.env.SECRET_ACCESS_KEY,
    server : 'https://your-endpoint'
});

obsClient.getObjectAcl({
    Bucket : 'bucketname',
    Key : 'objectname',
    VersionId : 'versionid'
}, (err, result) => {
    if(err){
        console.log('Error-->' + err);
    }else{
        console.log('Status-->' + result.CommonMsg.Status);
        if(result.CommonMsg.Status < 300 && result.InterfaceResult){
            console.log('Owner[ID]-->' + result.InterfaceResult.Owner.ID);
            for(let i=0;i<result.InterfaceResult.Grants.Grant.length;i++){
                console.log('Grant[' + i + ']:');
                console.log('Grantee[ID]-->' + result.InterfaceResult.Grants.Grant[i]['Grantee']['ID']);
                console.log('Grantee[URI]-->' + result.InterfaceResult.Grants.Grant[i]['Grantee']['URI']);
                console.log('Permission-->'+ result.InterfaceResult.Grants.Grant[i]['Permission']);
            }
        }
    }
});
```

10.9 删除多版本对象

须知

开发过程中，您有任何问题可以在github上**提交issue**。**接口参考文档**详细介绍了每个接口的参数和使用方法。

删除单个多版本对象

您可以通过ObsClient.deleteObject接口指定**VersionId参数**删除多版本对象，示例代码如下：

```
// 引入obs库
// 使用npm安装
var ObsClient = require('esdk-obs-nodejs');
// 使用源码安装
// var ObsClient = require('./lib/obs');

// 创建ObsClient实例
var obsClient = new ObsClient({
    //推荐通过环境变量获取AKSK，这里也可以使用其他外部引入方式传入，如果使用硬编码可能会存在泄露风险。
    //您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    access_key_id: process.env.ACCESS_KEY_ID,
    secret_access_key: process.env.SECRET_ACCESS_KEY,
    server : 'https://your-endpoint'
});

obsClient.deleteObject({
    Bucket : 'bucketname',
    Key : 'objectname',
    VersionId : 'versionid'
}, (err, result) => {
    if(err){
        console.log('Error-->' + err);
    }else{
        console.log('Status-->' + result.CommonMsg.Status);
    }
});
```

批量删除多版本对象

您可以通过ObsClient.deleteObjects接口传入每个待删除对象的**VersionId参数**批量删除多版本对象，示例代码如下：

```
// 引入obs库
// 使用npm安装
var ObsClient = require('esdk-obs-nodejs');
// 使用源码安装
// var ObsClient = require('./lib/obs');

// 创建ObsClient实例
var obsClient = new ObsClient({
    //推荐通过环境变量获取AKSK，这里也可以使用其他外部引入方式传入，如果使用硬编码可能会存在泄露风险。
    //您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    access_key_id: process.env.ACCESS_KEY_ID,
    secret_access_key: process.env.SECRET_ACCESS_KEY,
```

```
    server : 'https://your-endpoint'
});

obsClient.deleteObjects({
    Bucket: 'bucketname',
    // 设置为verbose模式
    Quiet : false,
    Objects : [{Key:'objectname1', VersionId : 'version1'},{Key:'objectname2', VersionId : 'version2'}, {Key :
'objectname3', VersionId : 'version3'}]
}, (err, result) => {
    if(err){
        console.log('Error-->' + err);
    }else{
        console.log('Status-->' + result.CommonMsg.Status);
        if(result.CommonMsg.Status < 300 && result.InterfaceResult){
            // 获取删除成功的对象
            console.log('Deleteds:');
            for(let i=0;i<result.InterfaceResult.Deleteds.length;i++){
                console.log('Deleted[' + i + ']:');
                console.log('Key-->'+result.InterfaceResult.Deleteds[i]['Key']);
                console.log('VersionId-->' + result.InterfaceResult.Deleteds[i]['VersionId']);
            }
            // 获取删除失败的对象
            console.log('Errors:');
            for(let j=0;j<result.InterfaceResult.Errors.length;j++){
                console.log('Error[' + j + ']:');
                console.log('Key-->' + result.InterfaceResult.Errors[j]['Key']);
                console.log('VersionId-->' + result.InterfaceResult.Errors[j]['VersionId']);
            }
        }
    }
});
```


11 生命周期管理

11.1 生命周期管理简介

须知

开发过程中，您有任何问题可以在github上**提交issue**。**接口参考文档**详细介绍了每个接口的参数和使用方法。

OBS允许您对桶设置生命周期规则，实现自动转换对象的存储类型、自动淘汰过期的对象，以有效利用存储特性，优化存储空间。针对不同前缀的对象，您可以同时设置多条规则。一条规则包含：

- 规则ID，用于标识一条规则，不能重复。
- 受影响的对象前缀，此规则只作用于符合前缀的对象。
- 最新版本对象的转换策略，指定方式为：
  a. 指定满足前缀的对象创建后第几天时转换为指定的存储类型。
  b. 直接指定满足前缀的对象转换为指定的存储类型的日期。
- 最新版本对象过期时间，指定方式为：
  a. 指定满足前缀的对象创建后第几天时过期。
  b. 直接指定满足前缀的对象过期日期。
- 历史版本对象转换策略，指定方式为：
  - 指定满足前缀的对象成为历史版本后第几天时转换为指定的存储类型。
- 历史版本对象过期时间，指定方式为：
  - 指定满足前缀的对象成为历史版本后第几天时过期。
- 是否生效标识。

更多关于生命周期的内容请参考**生命周期管理**。

说明

- 对象过期后会被OBS服务端自动删除。
- 对象转换策略中的时间必须早于对象过期时间；历史版本对象转换策略中的时间也必须早于历史版本对象的过期时间。
- 桶的多版本状态必须处于Enabled或者Suspended，历史版本对象转换策略和历史版本对象过期时间配置才能生效。

11.2 设置生命周期规则

须知

开发过程中，您有任何问题可以在github上**提交issue**。**接口参考文档**详细介绍了每个接口的参数和使用方法。

您可以通过ObsClient.setBucketLifecycle设置桶的生命周期规则。

设置对象转换策略

以下代码展示了如何设置最新版本对象和历史版本对象的转换策略：

```
// 引入obs库
// 使用npm安装
var ObsClient = require('esdk-obs-nodejs');
// 使用源码安装
// var ObsClient = require('./lib/obs');

// 创建ObsClient实例
var obsClient = new ObsClient({
    //推荐通过环境变量获取AKSK，这里也可以使用其他外部引入方式传入，如果使用硬编码可能会存在泄露风险。
    //您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    access_key_id: process.env.ACCESS_KEY_ID,
    secret_access_key: process.env.SECRET_ACCESS_KEY,
    server : 'https://your-endpoint'
});

obsClient.setBucketLifecycle({
    Bucket: 'bucketname',
    Rules:[
        {
            ID:'rule1',Prefix:'prefix1',Status:'Enabled',
            // 指定满足前缀的对象创建30天后过期转换为低频访问存储

            Transitions:[{StorageClass: obsClient.enums.StorageClassWarm, Days:30}],
            // 指定满足前缀的对象成为历史版本30天后过期转换为归档存储

            NoncurrentVersionTransitions:[{StorageClass: obsClient.enums.StorageClassCold, NoncurrentDays : 30}]
        },
        {
            ID:'rule2',Prefix:'prefix2',Status:'Enabled',
            // 直接指定满足前缀的对象转换为低频访问存储的时间

            Transitions:[{StorageClass: obsClient.enums.StorageClassWarm, Date: '2018-10-31T00:00:00Z'}],
        }
    ]
}, (err, result) => {
    if(err){
```

```
        console.log('Error-->' + err);
    }else{
        console.log('Status-->' + result.CommonMsg.Status);
    }
});
```

设置对象过期时间

以下代码展示了如何设置最新版本对象和历史版本对象的过期时间：

```
// 引入obs库
// 使用npm安装
var ObsClient = require('esdk-obs-nodejs');
// 使用源码安装
// var ObsClient = require('./lib/obs');

// 创建ObsClient实例
var obsClient = new ObsClient({
    //推荐通过环境变量获取AKSK，这里也可以使用其他外部引入方式传入，如果使用硬编码可能会存在泄露风险。
    //您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    access_key_id: process.env.ACCESS_KEY_ID,
    secret_access_key: process.env.SECRET_ACCESS_KEY,
    server : 'https://your-endpoint'
});

obsClient.setBucketLifecycle({
    Bucket: 'bucketname',
    Rules:[
        {
            ID:'rule1',Prefix:'prefix1',Status:'Enabled',
            // 指定满足前缀的对象创建60天后过期
            Expiration:{Days:60},
            // 指定满足前缀的对象成为历史版本60天后过期
            NoncurrentVersionExpiration:{NoncurrentDays : 60}
        },
        {
            ID:'rule2',Prefix:'prefix2',Status:'Enabled',
            // 直接指定满足前缀的对象过期时间，该值必须符合ISO8601格式，而且必须是UTC午夜0点
            Expiration:{Date: '2018-12-31T00:00:00Z'},
        }
    ]
}, (err, result) => {
    if(err){
        console.log('Error-->' + err);
    }else{
        console.log('Status-->' + result.CommonMsg.Status);
    }
});
```

说明

使用**Rules参数**指定桶的生命周期规则。

11.3 查看生命周期规则

须知

开发过程中，您有任何问题可以在github上**提交issue**。**接口参考文档**详细介绍了每个接口的参数和使用方法。

您可以通过ObsClient.getBucketLifecycle查看桶的生命周期规则。以下代码展示了如何查看桶的生命周期规则：

```
// 引入obs库
// 使用npm安装
var ObsClient = require('esdk-obs-nodejs');
// 使用源码安装
// var ObsClient = require('./lib/obs');

// 创建ObsClient实例
var obsClient = new ObsClient({
    //推荐通过环境变量获取AKSK，这里也可以使用其他外部引入方式传入，如果使用硬编码可能会存在泄露风险。
    //您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    access_key_id: process.env.ACCESS_KEY_ID,
    secret_access_key: process.env.SECRET_ACCESS_KEY,
    server : 'https://your-endpoint'
});

obsClient.getBucketLifecycle({
    Bucket: 'bucketname'
}, (err, result) => {
    if(err){
        console.log('Error-->' + err);
    }else{
        console.log('Status-->' + result.CommonMsg.Status);
        if(result.CommonMsg.Status < 300 && result.InterfaceResult){
            for(let i=0;i<result.InterfaceResult.Rules.length;i++){
                console.log('Rule[' + i + ']:');
                console.log('ID-->' + result.InterfaceResult.Rules[i]['ID']);
                console.log('Prefix-->' + result.InterfaceResult.Rules[i]['Prefix']);
                console.log('Status-->' + result.InterfaceResult.Rules[i]['Status']);
                for(let j=0;j<result.InterfaceResult.Rules[i]['Transitions'].length;j++){
                    console.log('Transition[' + j + ']:');
                    console.log('Transition[StorageClass]-->' + result.InterfaceResult.Rules[i]['Transitions'][j]['StorageClass']);
                    console.log('Transition[Date]-->' + result.InterfaceResult.Rules[i]['Transitions'][j]['Date']);
                    console.log('Transition[Days]-->' + result.InterfaceResult.Rules[i]['Transitions'][j]['Days']);
                }
                console.log('Expiration[Date]-->' + result.InterfaceResult.Rules[i]['Expiration']['Date']);
                console.log('Expiration[Days]-->' + result.InterfaceResult.Rules[i]['Expiration']['Days']);
                for(let k=0;k<result.InterfaceResult.Rules[i]['NoncurrentVersionTransitions'].length;k++){
                    console.log('NoncurrentVersionTransition[' + k + ']:');
                    console.log('NoncurrentVersionTransition[StorageClass]-->' + result.InterfaceResult.Rules[i]['NoncurrentVersionTransitions'][k]['StorageClass']);
                    console.log('NoncurrentVersionTransition[NoncurrentDays]-->' + result.InterfaceResult.Rules[i]['NoncurrentVersionTransitions'][k]['NoncurrentDays']);
                }
                console.log('NoncurrentVersionExpiration[NoncurrentDays]-->' + result.InterfaceResult.Rules[i]['NoncurrentVersionExpiration']['NoncurrentDays']);
            }
        }
    }
});
```

11.4 删除生命周期规则

须知

开发过程中，您有任何问题可以在github上**提交issue**。**接口参考文档**详细介绍了每个接口的参数和使用方法。

您可以通过ObsClient.deleteBucketLifecycle删除桶的生命周期规则。以下代码展示了如何删除桶的生命周期规则：

```
// 引入obs库
// 使用npm安装
var ObsClient = require('esdk-obs-nodejs');
// 使用源码安装
// var ObsClient = require('./lib/obs');

// 创建ObsClient实例
var obsClient = new ObsClient({
    //推荐通过环境变量获取AKSK，这里也可以使用其他外部引入方式传入，如果使用硬编码可能会存在泄露风险。
    //您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    access_key_id: process.env.ACCESS_KEY_ID,
    secret_access_key: process.env.SECRET_ACCESS_KEY,
    server : 'https://your-endpoint'
});

obsClient.deleteBucketLifecycle({
    Bucket: 'bucketname'
}, (err, result) => {
    if(err){
        console.log('Error-->' + err);
    }else{
        console.log('Status-->' + result.CommonMsg.Status);
    }
});
```


12 跨域资源共享

12.1 跨域资源共享简介

须知

开发过程中，您有任何问题可以在github上**提交issue**。**接口参考文档**详细介绍了每个接口的参数和使用方法。

跨域资源共享（CORS）允许Web端的应用程序访问不属于本域的资源。OBS提供接口方便开发者控制跨域访问的权限。

更多关于跨域资源共享的内容请参考**跨域资源访问**。

12.2 设置跨域规则

须知

开发过程中，您有任何问题可以在github上**提交issue**。**接口参考文档**详细介绍了每个接口的参数和使用方法。

您可以通过ObsClient.setBucketCors设置桶的跨域规则，如果原规则存在则覆盖原规则。以下代码展示了如何设置跨域规则：

```
// 引入obs库
// 使用npm安装
var ObsClient = require('esdk-obs-nodejs');
// 使用源码安装
// var ObsClient = require('./lib/obs');

// 创建ObsClient实例
var obsClient = new ObsClient({
    //推荐通过环境变量获取AKSK，这里也可以使用其他外部引入方式传入，如果使用硬编码可能会存在泄露风险。
    //您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
```

```
    access_key_id: process.env.ACCESS_KEY_ID,
    secret_access_key: process.env.SECRET_ACCESS_KEY,
    server : 'https://your-endpoint'
});

obsClient.setBucketCors({
    Bucket:'bucketname',
    CorsRules:[
      {
            // 指定允许的跨域请求方法(GET/PUT/DELETE/POST/HEAD)
            AllowedMethod: ['GET','HEAD','PUT'],
            // 指定允许跨域请求的来源
            AllowedOrigin: ['http://www.a.com','http://www.b.com'],
            // 控制在OPTIONS预取指令中Access-Control-Request-Headers头中指定的header是否被允许使
用
            AllowedHeader: ['x-obs-header'],
            // 指定允许用户从应用程序中访问的header
            ExposeHeader: ['x-obs-expose-header'],
            // 指定浏览器对特定资源的预取(OPTIONS)请求返回结果的缓存时间,单位为秒
            MaxAgeSeconds: 10
        }
    ]
}, (err, result) => {
    if(err){
        console.log('Error-->' + err);
    }else{
        console.log('Status-->' + result.CommonMsg.Status);
    }
});
```

说明

- 使用**CorsRules参数**指定桶的CORS配置信息。
- AllowedOrigin、AllowedHeader都能够最多支持一个“*”通配符。“*”表示对于所有的域来源或者头域都满足。

12.3 查看跨域规则

须知

开发过程中，您有任何问题可以在github上**提交issue**。**接口参考文档**详细介绍了每个接口的参数和使用方法。

您可以通过ObsClient.getBucketCors查看桶的跨域规则。以下代码展示了如何查看跨域规则：

```
// 引入obs库
// 使用npm安装
var ObsClient = require('esdk-obs-nodejs');
// 使用源码安装
// var ObsClient = require('./lib/obs');

// 创建ObsClient实例
var obsClient = new ObsClient({
    //推荐通过环境变量获取AKSK，这里也可以使用其他外部引入方式传入，如果使用硬编码可能会存在泄露风
险。
    //您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/
intl/zh-cn/usermanual-ca/ca_01_0003.html
    access_key_id: process.env.ACCESS_KEY_ID,
    secret_access_key: process.env.SECRET_ACCESS_KEY,
    server : 'https://your-endpoint'
});
```



```
obsClient.getBucketCors({
        Bucket:'bucketname'
}, (err, result) => {
        if(err){
                console.log('Error-->' + err);
        }else{
                console.log('Status-->' + result.CommonMsg.Status);
                if(result.CommonMsg.Status < 300 && result.InterfaceResult){
                        for(let k=0;k<result.InterfaceResult.CorsRule.length;k++){
                                console.log('CorsRule[',k,']');
                                console.log('CorsRule[MaxAgeSeconds]-->' + result.InterfaceResult.CorsRules[k]
['MaxAgeSeconds']);
                                for (let i=0;i<result.InterfaceResult.CorsRule[k]['AllowedMethod'].length;i++){
                                        console.log('CorsRule[AllowedMethod][' , i ,']-->'+result.InterfaceResult.CorsRules[k]
['AllowedMethod'][i]);
                                }
                                for(let j=0;j<result.InterfaceResult.CorsRule[k]['AllowedOrigin'].length;j++){
                                        console.log('CorsRule[AllowedOrigin][', j ,']-->'+result.InterfaceResult.CorsRules[k]
['AllowedOrigin'][j]);
                                 }
                                for(let n=0;n<result.InterfaceResult.CorsRule[k]['AllowedHeader'].length;n++){
                                        console.log('CorsRule[AllowedHeader][', n ,']-->'+result.InterfaceResult.CorsRules[k]
['AllowedHeader'][n]);
                                }
                                for(let m=0;m<result.InterfaceResult.CorsRule[k]['ExposeHeader'].length;m++){
                                        console.log('CorsRule[ExposeHeader][', m ,']-->'+result.InterfaceResult.CorsRules[k]
['ExposeHeader'][m]);
                                }
                        }
                }
        }
});
```

12.4 删除跨域规则

须知

开发过程中，您有任何问题可以在github上**提交issue**。**接口参考文档**详细介绍了每个接口的参数和使用方法。

您可以通过ObsClient.deleteBucketCors删除桶的跨域规则。以下代码展示了如何删除跨域规则：

```
// 引入obs库
// 使用npm安装
var ObsClient = require('esdk-obs-nodejs');
// 使用源码安装
// var ObsClient = require('./lib/obs');

// 创建ObsClient实例
var obsClient = new ObsClient({
        //推荐通过环境变量获取AKSK，这里也可以使用其他外部引入方式传入，如果使用硬编码可能会存在泄露风险。
        //您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
        access_key_id: process.env.ACCESS_KEY_ID,
        secret_access_key: process.env.SECRET_ACCESS_KEY,
        server : 'https://your-endpoint'
});

obsClient.deleteBucketCors({
        Bucket:'bucketname'
```



```
}, (err, result) => {
    if(err){
        console.log('Error-->' + err);
    }else{
        console.log('Status-->' + result.CommonMsg.Status);
    }
});
```


13 设置访问日志

13.1 日志简介

须知

开发过程中，您有任何问题可以在github上**提交issue**。**接口参考文档**详细介绍了每个接口的参数和使用方法。

OBS允许您对桶设置访问日志记录，设置之后对于桶的访问会被记录成日志，日志存储在OBS上您指定的目标桶中。

更多关于访问日志的内容请参考**日志记录**。

13.2 开启桶日志

须知

开发过程中，您有任何问题可以在github上**提交issue**。**接口参考文档**详细介绍了每个接口的参数和使用方法。

您可以通过ObsClient.setBucketLogging开启桶日志功能。

须知

- 日志目标桶与源桶必须在同一个区域（region）。
- 在配置桶日志管理前，需要先到**统一身份认证服务**生成一个对OBS服务的委托，获取到委托名。请参考**创建IAM委托**。

说明

- 如果桶的存储类型为低频访问存储或归档存储，则不能作为日志目标桶。

开启桶日志

以下代码展示了如何开启桶日志：

```
// 引入obs库
// 使用npm安装
const ObsClient = require('esdk-obs-nodejs');
// 使用源码安装
// const ObsClient = require('./lib/obs');

// 创建ObsClient实例
const obsClient = new ObsClient({
    //推荐通过环境变量获取AKSK，这里也可以使用其他外部引入方式传入，如果使用硬编码可能会存在泄露风险。
    //您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    access_key_id: process.env.ACCESS_KEY_ID,
    secret_access_key: process.env.SECRET_ACCESS_KEY,
    //这里以中国-香港为例，其他地区请按实际情况填写
    server: 'https://obs.ap-southeast-1.myhuaweicloud.com'

});

// 设置桶的日志配置
obsClient.setBucketLogging({
    Bucket:'bucketname',
    // 通过IAM创建的委托名
    Agency: 'Agency name',
    LoggingEnabled:{
        // 生成日志文件的桶名
        TargetBucket: 'LogBucketName',
        // 指定生成的日志文件的对象名前缀
        TargetPrefix: 'logs/',
    }
}, (err, result) => {
    if(err){
        console.log('Error-->' + err);
    }else{
        console.log('Status-->' + result.CommonMsg.Status);
        console.log('RequestId-->' + result.CommonMsg.RequestId);
    }
});
```

说明

使用**LoggingEnabled参数**指定桶的日志配置。

为日志对象设置权限

以下代码展示了如何为日志对象设置权限：

```
// 引入obs库
// 使用npm安装
var ObsClient = require('esdk-obs-nodejs');
// 使用源码安装
// var ObsClient = require('./lib/obs');

// 创建ObsClient实例
const obsClient = new ObsClient({
    //推荐通过环境变量获取AKSK，这里也可以使用其他外部引入方式传入，如果使用硬编码可能会存在泄露风险。
    //您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-
```

```
cn/usermanual-ca/ca_01_0003.html
    access_key_id: process.env.ACCESS_KEY_ID,
    secret_access_key: process.env.SECRET_ACCESS_KEY,
    //这里以中国-香港为例，其他地区请按实际情况填写
    server: 'https://obs.ap-southeast-1.myhuaweicloud.com'

});

// 设置桶的日志配置
obsClient.setBucketLogging({
    Bucket:'bucketname',
    // 目标桶Owner通过IAM创建对OBS服务的委托名称
    Agency: 'Agency name',
    LoggingEnabled:{
        // 生成日志文件的桶名
        TargetBucket: 'LogBucketName',
        // 指定生成的日志文件的对象名前缀
        TargetPrefix: 'logs/',
        TargetGrants:[
            // 为所有用户设置对日志对象的读权限
           {Grantee:
{Type:'Group',URI:obsClient.enums.GroupAllUsers},Permission:obsClient.enums.PermissionRead},
            // 为所有用户设置对日志文件的写权限
           {Grantee:
{Type:'Group',URI:obsClient.enums.GroupAllUsers},Permission:obsClient.enums.PermissionWrite}
        ]
     }
}, (err, result) => {
    if(err){
        console.log('Error-->' + err);
    }else{
        console.log('Status-->' + result.CommonMsg.Status);
        console.log('RequestId-->' + result.CommonMsg.RequestId);
    }
});
```

13.3 查看桶日志配置

须知

开发过程中，您有任何问题可以在github上**提交issue**。**接口参考文档**详细介绍了每个接口的参数和使用方法。

您可以通过ObsClient.getBucketLogging查看桶日志配置。以下代码展示了如何查看桶日志配置：

```
// 引入obs库
// 使用npm安装
var ObsClient = require('esdk-obs-nodejs');
// 使用源码安装
// var ObsClient = require('./lib/obs');

// 创建ObsClient实例
const obsClient = new ObsClient({
    //推荐通过环境变量获取AKSK，这里也可以使用其他外部引入方式传入，如果使用硬编码可能会存在泄露风险。
    //您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    access_key_id: process.env.ACCESS_KEY_ID,
    secret_access_key: process.env.SECRET_ACCESS_KEY,
    //这里以中国-香港为例，其他地区请按实际情况填写
    server: 'https://obs.ap-southeast-1.myhuaweicloud.com'
```



```
});

obsClient.getBucketLogging({
      Bucket:'bucketname'
}, (err, result) => {
      if(err){
            console.log('Error-->' + err);
      }else{
            console.log('Status-->' + result.CommonMsg.Status);
            if(result.CommonMsg.Status < 300 && result.InterfaceResult){
                  if(result.InterfaceResult.LoggingEnabled){
                        console.log('TargetBucket-->' + result.InterfaceResult.LoggingEnabled.TargetBucket);
                        console.log('TargetPrefix-->' + result.InterfaceResult.LoggingEnabled.TargetPrefix);
                  }
                  for(let i=0;i<result.InterfaceResult.LoggingEnabled.TargetGrants.length;i++){
                        console.log('Grant[' + i + ']:');
                        console.log('Grantee[ID]-->' + result.InterfaceResult.LoggingEnabled.TargetGrants[i]
['Grantee']['ID']);
                        console.log('Grantee[URI]-->' + result.InterfaceResult.LoggingEnabled.TargetGrants[i]
['Grantee']['URI']);
                        console.log('Permission-->' + result.InterfaceResult.LoggingEnabled.TargetGrants[i]
['Permission']);
                  }
            }
      }
});
```

13.4 关闭桶日志

须知

开发过程中，您有任何问题可以在github上**提交issue**。**接口参考文档**详细介绍了每个接口的参数和使用方法。

关闭桶日志功能实际上就是调用ObsClient.setBucketLogging将日志配置清空，以下代码展示了如何关闭桶日志：

```
// 引入obs库
// 使用npm安装
var ObsClient = require('esdk-obs-nodejs');
// 使用源码安装
// var ObsClient = require('./lib/obs');

// 创建ObsClient实例
const obsClient = new ObsClient({
    //推荐通过环境变量获取AKSK，这里也可以使用其他外部引入方式传入，如果使用硬编码可能会存在泄露风
险。
    //您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-
cn/usermanual-ca/ca_01_0003.html
    access_key_id: process.env.ACCESS_KEY_ID,
    secret_access_key: process.env.SECRET_ACCESS_KEY,
    //这里以中国-香港为例，其他地区请按实际情况填写
    server: 'https://obs.ap-southeast-1.myhuaweicloud.com'

});

obsClient.setBucketLogging({
      Bucket:'bucketname',
      LoggingEnabled : {}
}, (err, result) => {
      if(err){
            console.log('Error-->' + err);
      }else{
```



```
            console.log('Status-->' + result.CommonMsg.Status);
        }
});
```


14 静态网站托管

14.1 静态网站托管简介

须知

开发过程中，您有任何问题可以在github上**提交issue**。**接口参考文档**详细介绍了每个接口的参数和使用方法。

您可以将静态网站文件上传至OBS的桶中作为对象，并对这些对象赋予公共读权限，然后将该桶配置成静态网站托管模式，以实现在OBS上托管静态网站的目的。第三方用户在访问您网站的时候，实际上是在访问OBS的桶中的对象。在使用静态网站托管功能时，OBS还支持配置请求重定向，通过重定向配置您可以将特定的请求或所有请求实施重定向。

更多关于静态网站托管的内容请参考**静态网站托管**。

14.2 网站文件托管

须知

开发过程中，您有任何问题可以在github上**提交issue**。**接口参考文档**详细介绍了每个接口的参数和使用方法。

您可通过以下步骤实现网站文件托管：

步骤1 将网站文件上传至OBS的桶中，并设置对象MIME类型。

步骤2 设置对象访问权限为公共读。

步骤3 通过浏览器访问对象。

----结束

以下代码展示了如何实现网站文件托管：

```
// 引入obs库
// 使用npm安装
var ObsClient = require('esdk-obs-nodejs');
// 使用源码安装
// var ObsClient = require('./lib/obs');

// 创建ObsClient实例
const obsClient = new ObsClient({
    //推荐通过环境变量获取AKSK，这里也可以使用其他外部引入方式传入，如果使用硬编码可能会存在泄露风险。
    //您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    access_key_id: process.env.ACCESS_KEY_ID,
    secret_access_key: process.env.SECRET_ACCESS_KEY,
    //这里以中国-香港为例，其他地区请按实际情况填写
    server: 'https://obs.ap-southeast-1.myhuaweicloud.com'

});

// 上传对象
obsClient.putObject({
        Bucket: 'bucketname',
        Key: 'test.html',
        Body: '<html><header></header><body><h1>Hello OBS</h1></body></html>',
        // 设置对象MIME类型
        ContentType: 'text/html',
        // 设置对象访问权限为公共读
        ACL: obsClient.enums.AclPublicRead
}, (err, result) => {
        if(err){
            console.log('Error-->' + err);
        }else{
            console.log('Status-->' + result.CommonMsg.Status);
        }
});
```

说明

上例中您可以使用http://bucketname.your-endpoint/test.html在浏览器直接访问托管的文件。

14.3 设置托管配置

须知

开发过程中，您有任何问题可以在github上**提交issue**。**接口参考文档**详细介绍了每个接口的参数和使用方法。

您可以通过ObsClient.setBucketWebsite设置桶的托管配置。

配置默认主页和错误页面

以下代码展示了如何配置默认主页和错误页面：

```
// 引入obs库
// 使用npm安装
var ObsClient = require('esdk-obs-nodejs');
// 使用源码安装
// var ObsClient = require('./lib/obs');

// 创建ObsClient实例
```



```
const obsClient = new ObsClient({
    //推荐通过环境变量获取AKSK，这里也可以使用其他外部引入方式传入，如果使用硬编码可能会存在泄露风险。
    //您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    access_key_id: process.env.ACCESS_KEY_ID,
    secret_access_key: process.env.SECRET_ACCESS_KEY,
    //这里以中国-香港为例，其他地区请按实际情况填写
    server: 'https://obs.ap-southeast-1.myhuaweicloud.com'

});

obsClient.setBucketWebsite({
        Bucket: 'bucketname',
        // 配置默认主页
        IndexDocument:{Suffix:'index.html'},
        // 配置错误页面
        ErrorDocument:{Key:'error.html'}
}, (err, result) => {
        if(err){
            console.log('Error-->' + err);
        }else{
            console.log('Status-->' + result.CommonMsg.Status);
        }
});
```

配置重定向规则

以下代码展示了如何配置重定向规则：

```
// 引入obs库
// 使用npm安装
var ObsClient = require('esdk-obs-nodejs');
// 使用源码安装
// var ObsClient = require('./lib/obs');

// 创建ObsClient实例
const obsClient = new ObsClient({
    //推荐通过环境变量获取AKSK，这里也可以使用其他外部引入方式传入，如果使用硬编码可能会存在泄露风险。
    //您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    access_key_id: process.env.ACCESS_KEY_ID,
    secret_access_key: process.env.SECRET_ACCESS_KEY,
    //这里以中国-香港为例，其他地区请按实际情况填写
    server: 'https://obs.ap-southeast-1.myhuaweicloud.com'

});

obsClient.setBucketWebsite({
        Bucket: 'bucketname',
        // 配置默认主页
        IndexDocument:{Suffix:'index.html'},
        // 配置错误页面
        ErrorDocument:{Key:'error.html'},
        // 配置重定向规则
        RoutingRules:[
            {
                Condition:{HttpErrorCodeReturnedEquals:'404', KeyPrefixEquals: 'keyprefix'},
                Redirect:{Protocol:'http',ReplaceKeyWith:'replacekeyprefix', HostName: 'www.example.com', HttpRedirectCode: '305'}
            }
        ]
}, (err, result) => {
        if(err){
            console.log('Error-->' + err);
        }else{
            console.log('Status-->' + result.CommonMsg.Status);
```



```
    }
});
```

说明

使用**RoutingRules参数**指定桶的重定向规则。

配置所有请求重定向

以下代码展示了如何配置所有请求重定向：

```
// 引入obs库
// 使用npm安装
var ObsClient = require('esdk-obs-nodejs');
// 使用源码安装
// var ObsClient = require('./lib/obs');

// 创建ObsClient实例
const obsClient = new ObsClient({
    //推荐通过环境变量获取AKSK，这里也可以使用其他外部引入方式传入，如果使用硬编码可能会存在泄露风险。
    //您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    access_key_id: process.env.ACCESS_KEY_ID,
    secret_access_key: process.env.SECRET_ACCESS_KEY,
    //这里以中国-香港为例，其他地区请按实际情况填写
    server: 'https://obs.ap-southeast-1.myhuaweicloud.com'

});

obsClient.setBucketWebsite({
        Bucket: 'bucketname',
        RedirectAllRequestsTo : {HostName : 'www.example.com', Protocol : 'http'}
}, (err, result) => {
        if(err){
                console.log('Error-->' + err);
        }else{
                console.log('Status-->' + result.CommonMsg.Status);
        }
});
```

说明

使用**RedirectAllRequestsTo参数**指定桶的所有请求重定向规则。

14.4 查看托管配置

须知

开发过程中，您有任何问题可以在github上**提交issue**。**接口参考文档**详细介绍了每个接口的参数和使用方法。

您可以通过ObsClient.getBucketWebsite查看桶的托管配置。以下代码展示了如何查看托管配置：

```
// 引入obs库
// 使用npm安装
var ObsClient = require('esdk-obs-nodejs');
// 使用源码安装
// var ObsClient = require('./lib/obs');

// 创建ObsClient实例
```

```
const obsClient = new ObsClient({
    //推荐通过环境变量获取AKSK，这里也可以使用其他外部引入方式传入，如果使用硬编码可能会存在泄露风险。
    //您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    access_key_id: process.env.ACCESS_KEY_ID,
    secret_access_key: process.env.SECRET_ACCESS_KEY,
    //这里以中国-香港为例，其他地区请按实际情况填写
    server: 'https://obs.ap-southeast-1.myhuaweicloud.com'

});

obsClient.getBucketWebsite({
        Bucket: 'bucketname'
}, (err, result) => {
        if(err){
                console.log('Error-->' + err);
        }else{
                console.log('Status-->' + result.CommonMsg.Status);
                if(result.CommonMsg.Status < 300 && result.InterfaceResult){
                        console.log('RedirectAllRequestsTo:');
                        console.log('HostName-->' + result.InterfaceResult.RedirectAllRequestsTo['HostName']);
                        console.log('Protocol-->' + result.InterfaceResult.RedirectAllRequestsTo['Protocol']);
                        console.log('IndexDocument[Suffix]-->' + result.InterfaceResult.IndexDocument['Suffix']);
                        console.log('ErrorDocument[Key]-->' + result.InterfaceResult.ErrorDocument['Key']);
                        console.log('RoutingRules:');
                        for(let i=0;i<result.InterfaceResult.RoutingRules;i++){
                                console.log('RoutingRule[' + i + ']:');
                                let RoutingRule = result.InterfaceResult.RoutingRules[i];
                                console.log('Condition[HttpErrorCodeReturnedEquals]-->' + RoutingRule['Condition']['HttpErrorCodeReturnedEquals']);
                                 console.log('Condition[KeyPrefixEquals]-->' + RoutingRule['Condition']['KeyPrefixEquals']);
                                console.log('Redirect[HostName]-->' + RoutingRule['Redirect']['HostName']);
                                console.log('Redirect[HttpRedirectCode]-->' + RoutingRule['Redirect']['HttpRedirectCode']);
                                console.log('Redirect[Protocol]-->' + RoutingRule['Redirect']['Protocol']);
                                console.log('Redirect[ReplaceKeyPrefixWith]-->' + RoutingRule['Redirect']['ReplaceKeyPrefixWith']);
                                console.log('Redirect[ReplaceKeyWith]-->' + RoutingRule['Redirect']['ReplaceKeyWith']);
                        }
                }
        }
});
```

14.5 清除托管配置

须知

开发过程中，您有任何问题可以在github上**提交issue**。**接口参考文档**详细介绍了每个接口的参数和使用方法。

您可以通过ObsClient.deleteBucketWebsite清除桶的托管配置。以下代码展示了如何清除托管配置：

```
// 引入obs库
// 使用npm安装
var ObsClient = require('esdk-obs-nodejs');
// 使用源码安装
// var ObsClient = require('./lib/obs');

// 创建ObsClient实例
const obsClient = new ObsClient({
```



```
    //推荐通过环境变量获取AKSK，这里也可以使用其他外部引入方式传入，如果使用硬编码可能会存在泄露风险。
    //您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    access_key_id: process.env.ACCESS_KEY_ID,
    secret_access_key: process.env.SECRET_ACCESS_KEY,
    //这里以中国-香港为例，其他地区请按实际情况填写
    server: 'https://obs.ap-southeast-1.myhuaweicloud.com'

});

obsClient.deleteBucketWebsite({
        Bucket: 'bucketname'
}, (err, result) => {
        if(err){
                console.log('Error-->' + err);
        }else{
                console.log('Status-->' + result.CommonMsg.Status);
        }
});
```


15 标签管理

15.1 标签简介

须知

开发过程中，您有任何问题可以在github上**提交issue**。**接口参考文档**详细介绍了每个接口的参数和使用方法。

标签用于标识OBS中的桶，以此来达到对OBS中的桶进行分类的目的。

15.2 设置桶标签

须知

开发过程中，您有任何问题可以在github上**提交issue**。**接口参考文档**详细介绍了每个接口的参数和使用方法。

您可以通过ObsClient.setBucketTagging设置桶标签。以下代码展示了如何设置桶标签：

```
// 引入obs库
// 使用npm安装
var ObsClient = require('esdk-obs-nodejs');
// 使用源码安装
// var ObsClient = require('./lib/obs');

// 创建ObsClient实例
const obsClient = new ObsClient({
    //推荐通过环境变量获取AKSK，这里也可以使用其他外部引入方式传入，如果使用硬编码可能会存在泄露风险。
    //您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    access_key_id: process.env.ACCESS_KEY_ID,
    secret_access_key: process.env.SECRET_ACCESS_KEY,
    //这里以中国-香港为例，其他地区请按实际情况填写
    server: 'https://obs.ap-southeast-1.myhuaweicloud.com'
```

```
});

obsClient.setBucketTagging({
      Bucket: 'bucketname',
      Tags : [
         {Key:'tag1',Value:'value1'},
         {Key:'tag2',Value:'value2'}
      ]
}, (err, result) => {
      if(err){
            console.log('Error-->' + err);
      }else{
            console.log('Status-->' + result.CommonMsg.Status);
      }
});
```

说明

- 使用**Tags参数**指定桶的标签信息。
- 每个桶支持最多10个标签。
- 标签的Key和Value支持Unicode。

15.3 查看桶标签

须知

开发过程中，您有任何问题可以在github上**提交issue**。**接口参考文档**详细介绍了每个接口的参数和使用方法。

您可以通过ObsClient.getBucketTagging查看桶标签。以下代码展示了如何查看桶标签：

```
// 引入obs库
// 使用npm安装
var ObsClient = require('esdk-obs-nodejs');
// 使用源码安装
// var ObsClient = require('./lib/obs');

// 创建ObsClient实例
const obsClient = new ObsClient({
    //推荐通过环境变量获取AKSK，这里也可以使用其他外部引入方式传入，如果使用硬编码可能会存在泄露风险。
    //您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    access_key_id: process.env.ACCESS_KEY_ID,
    secret_access_key: process.env.SECRET_ACCESS_KEY,
    //这里以中国-香港为例，其他地区请按实际情况填写
    server: 'https://obs.ap-southeast-1.myhuaweicloud.com'

});

obsClient.getBucketTagging({
      Bucket: 'bucketname'
}, (err, result) => {
      if(err){
            console.log('Error-->' + err);
      }else{
            console.log('Status-->' + result.CommonMsg.Status);
            if(result.CommonMsg.Status < 300 && result.InterfaceResult){
                  result.InterfaceResult.Tags.forEach((tag) => {
                        console.log('Tag-->' + tag.Key + ':' + tag.Value);
```



```
            });
        }
    }
});
```

15.4 删除桶标签

须知

开发过程中，您有任何问题可以在github上**提交issue**。**接口参考文档**详细介绍了每个接口的参数和使用方法。

您可以通过ObsClient.deleteBucketTagging删除桶标签。以下代码展示了如何删除桶标签：

```
// 引入obs库
// 使用npm安装
var ObsClient = require('esdk-obs-nodejs');
// 使用源码安装
// var ObsClient = require('./lib/obs');

// 创建ObsClient实例
const obsClient = new ObsClient({
    //推荐通过环境变量获取AKSK，这里也可以使用其他外部引入方式传入，如果使用硬编码可能会存在泄露风险。
    //您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    access_key_id: process.env.ACCESS_KEY_ID,
    secret_access_key: process.env.SECRET_ACCESS_KEY,
    //这里以中国-香港为例，其他地区请按实际情况填写
    server: 'https://obs.ap-southeast-1.myhuaweicloud.com'

});

obsClient.deleteBucketTagging({
        Bucket: 'bucketname'
}, (err, result) => {
        if(err){
                console.log('Error-->' + err);
        }else{
                console.log('Status-->' + result.CommonMsg.Status);
        }
});
```


16 服务端加密

16.1 服务端加密简介

须知

开发过程中，您有任何问题可以在github上**提交issue**。**接口参考文档**详细介绍了每个接口的参数和使用方法。

OBS支持服务端加密功能，使对象加密的行为在OBS服务端进行。

更多关于服务端加密的内容请参考**服务端加密**。

16.2 加密说明

须知

开发过程中，您有任何问题可以在github上**提交issue**。**接口参考文档**详细介绍了每个接口的参数和使用方法。

OBS Node.js SDK支持服务端加密的接口见下表：

OBS Node.js SDK接口方法	描述	支持加密类型
ObsClient.putObject	上传对象时设置加密算法、密钥，对对象启用服务端加密。	SSE-KMS SSE-C
ObsClient.appendObject	追加上传对象时设置加密算法、密钥，对对象启用服务端加密。	SSE-KMS SSE-C

OBS Node.js SDK接口方法	描述	支持加密类型
ObsClient.getObject	下载对象时设置解密算法、密钥，用于解密对象。	SSE-C
ObsClient.copyObject	1. 复制对象时设置源对象的解密算法、密钥，用于解密源对象。 2. 复制对象时设置目标对象的加密算法、密钥，对目标对象启用加密算法。	SSE-KMS SSE-C
ObsClient.getObjectMetadata	获取对象元数据时设置解密算法、密钥，用于解密对象。	SSE-C
ObsClient.initiateMultipartUpload	初始化分段上传任务时设置加密算法、密钥，对分段上传任务最终生成的对象启用服务端加密。	SSE-KMS SSE-C
ObsClient.uploadPart	上传段时设置加密算法、密钥，对分段数据启用服务端加密。	SSE-C
ObsClient.copyPart	1. 复制段时设置源对象的解密算法、密钥，用于解密源对象。 2. 复制段时设置目标段的加密算法、密钥，对目标段启用加密算法。	SSE-C

OBS Node.js SDK两种加密方式支持的请求参数：

加密类型	OBS Node.js SDK对应请求参数	说明
SSE-KMS	SseKms	表示服务端加密是SSE-KMS方式，目前仅支持：**kms**。
	SseKmsKey	表示SSE-KMS方式下的主密钥，可为空。
SSE-C	SseC	表示服务端加密是SSE-C方式，目前仅支持：**AES256**。
	SseCKey	表示SSE-C方式下的密钥，由AES256算法得到。上传对象时作为加密密钥；下载对象时作为解密密钥。
	CopySourceSseC	适用于ObsClient.copyObject和ObsClient.copyPart，表示以SSE-C方式解密源对象，目前仅支持：**AES256**。
	CopySourceSseCKey	适用于ObsClient.copyObject和ObsClient.copyPart，表示以SSE-C方式解密源对象时使用的密钥，由AES256算法得到。

16.3 加密示例

须知

开发过程中，您有任何问题可以在github上**提交issue**。**接口参考文档**详细介绍了每个接口的参数和使用方法。

上传对象加密

以下代码展示了在上传对象时使用服务端加密功能：

```
// 引入obs库
// 使用npm安装
var ObsClient = require('esdk-obs-nodejs');
// 使用源码安装
// var ObsClient = require('./lib/obs');

// 创建ObsClient实例
const obsClient = new ObsClient({
    //推荐通过环境变量获取AKSK，这里也可以使用其他外部引入方式传入，如果使用硬编码可能会存在泄露风险。
    //您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    access_key_id: process.env.ACCESS_KEY_ID,
    secret_access_key: process.env.SECRET_ACCESS_KEY,
    //这里以中国-香港为例，其他地区请按实际情况填写
    server: 'https://obs.ap-southeast-1.myhuaweicloud.com'

});

obsClient.putObject({
        Bucket: 'bucketname',
        Key: 'objectname',
        SourceFile: 'localfile',
        // 设置SSE-C算法加密对象
        SseC: 'AES256',
        SseCKey: 'your sse-c key generated by AES-256 algorithm'
}, (err, result) => {
        if(err){
            console.log('Error-->' + err);
        }else{
            console.log('Status-->' + result.CommonMsg.Status);
        }
});

obsClient.putObject({
        Bucket: 'bucketname',
        Key: 'objectname2',
        SourceFile: 'localfile2',
        // 设置SSE-KMS算法加密对象
        SseKms: 'kms'
}, (err, result) => {
        if(err){
            console.log('Error-->' + err);
        }else{
            console.log('Status-->' + result.CommonMsg.Status);
        }
});
```

下载对象解密

以下代码展示了在下载对象时使用服务端解密功能：

```
// 引入obs库
// 使用npm安装
var ObsClient = require('esdk-obs-nodejs');
// 使用源码安装
// var ObsClient = require('./lib/obs');

// 创建ObsClient实例
const obsClient = new ObsClient({
    //推荐通过环境变量获取AKSK，这里也可以使用其他外部引入方式传入，如果使用硬编码可能会存在泄露风险。
    //您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    access_key_id: process.env.ACCESS_KEY_ID,
    secret_access_key: process.env.SECRET_ACCESS_KEY,
    //这里以中国-香港为例，其他地区请按实际情况填写
    server: 'https://obs.ap-southeast-1.myhuaweicloud.com'

});

obsClient.getObject({
        Bucket: 'bucketname',
        Key: 'objectname',
        // 设置SSE-C算法解密对象
        SseC: 'AES256',
        // 此处的密钥必须和上传对象加密时使用的密钥一致
        SseCKey: 'your sse-c key generated by AES-256 algorithm'
}, (err, result) => {
        if(err){
            console.log('Error-->' + err);
        }else{
            console.log('Status-->' + result.CommonMsg.Status);
            if(result.CommonMsg.Status < 300 && result.InterfaceResult){
                console.log('Content-->' + result.InterfaceResult.Content.toString());
            }
        }
});
```


17 异常处理

17.1 OBS 服务端错误码

在向OBS服务端发出请求后，如果遇到错误，会在响应中包含响应的错误码描述错误信息。详细的错误码及其对应的描述和HTTP状态码见下表：

错误码	描述	HTTP状态码
AccessDenied	拒绝访问。	403 Forbidden
AccessForbidden	权限不足。	403 Forbidden
AccountProblem	用户的账户出现异常（过期、冻结等），不能成功地完成操作。	403 Forbidden
AllAccessDisabled	用户无权限执行某操作。	403 Forbidden
AmbiguousGrantByEmailAddress	用户提供的Email地址关联的账户超过了1个。	400 Bad Request
BadDigest	客户端指定的对象内容的MD5值与系统接收到的内容MD5值不一致。	400 Bad Request
BadDomainName	域名不合法。	400 Bad Request
BadRequest	请求参数不合法。	400 Bad Request
BucketAlreadyExists	请求的桶名已经存在。桶的命名空间是系统中所有用户共用的，选择一个不同的桶名再重试一次。	409 Conflict
BucketAlreadyOwnedByYou	发起该请求的用户已经创建过了这个名字的桶，并拥有这个桶。	409 Conflict

错误码	描述	HTTP状态码
BucketNotEmpty	用户尝试删除的桶不为空。	409 Conflict
CredentialsNotSupported	该请求不支持证书验证。	400 Bad Request
CustomDomainAreadyExist	配置了已存在的域。	400 Bad Request
CustomDomainNotExist	操作的域不存在。	400 Bad Request
DeregisterUserId	用户已经注销。	403 Forbidden
EntityTooSmall	用户试图上传的对象大小小于系统允许的最小大小。	400 Bad Request
EntityTooLarge	用户试图上传的对象大小超过了系统允许的最大大小。	400 Bad Request
FrozenUserId	用户被冻结。	403 Forbidden
IllegalVersioningConfigurationException	请求中的版本配置无效。	400 Bad Request
IllegalLocationConstraintException	配置了与所在Region不匹配的区域限制。	400 Bad Request
InArrearOrInsufficientBalance	因为ACL而没有权限进行某种操作。	403 Forbidden
IncompleteBody	请求体不完整。	400 Bad Request
IncorrectNumberOfFilesInPostRequest	每个POST请求都需要带一个上传的文件。	400 Bad Request
InlineDataTooLarge	Inline Data超过了允许的最大长度。	400 Bad Request
InsufficientStorageSpace	存储空间不足。	403 Forbidden
InternalError	系统遇到内部错误，请重试。	500 Internal Server Error
InvalidAccessKeyId	系统记录中不存在客户提供的Access Key Id。	403 Forbidden
InvalidAddressingHeader	用户必须指定匿名角色。	N/A
InvalidArgument	无效的参数。	400 Bad Request
InvalidBucketName	请求中指定的桶名无效。	400 Bad Request
InvalidBucket	请求访问的桶已不存在。	400 Bad Request

错误码	描述	HTTP状态码
InvalidBucketState	无效的桶状态。	409 Conflict
InvalidBucketStoragePolicy	修改桶策略时，提供的新策略不合法。	400 Bad Request
InvalidDigest	HTTP头中指定的Content-MD5值无效。	400 Bad Request
InvalidEncryptionAlgorithmError	错误的加密算法。	400 Bad Request
InvalidLocationConstraint	创建桶时，指定的location不合法。	400 Bad Request
InvalidPart	一个或多个指定的段无法找到。这些段可能没有上传，或者指定的entity tag与段的entity tag不一致。	400 Bad Request
InvalidPartOrder	段列表的顺序不是升序，段列表必须按段号升序排列。	400 Bad Request
InvalidPayer	所有对这个对象的访问已经无效了。	403 Forbidden
InvalidPolicyDocument	表单中的内容与策略文档中指定的条件不一致。	400 Bad Request
InvalidRange	请求的range不可获得。	416 Client Requested Range Not Satisfiable
InvalidRedirectLocation	无效的重定向地址。	400 Bad Request
InvalidRequest	无效请求。	400 Bad Request
InvalidRequestBody	POST请求体无效。	400 Bad Request
InvalidSecurity	提供的安全证书无效。	403 Forbidden
InvalidStorageClass	用户指定的Storage Class无效。	400 Bad Request
InvalidTargetBucketForLogging	delivery group对目标桶无ACL权限。	400 Bad Request
InvalidURI	无法解析指定的URI。	400 Bad Request
KeyTooLong	提供的Key过长。	400 Bad Request

错误码	描述	HTTP状态码
MalformedACLError	提供的XML格式错误，或者不符合要求的格式。	400 Bad Request
MalformedError	请求中携带的XML格式不正确。	400 Bad Request
MalformedLoggingStatus	Logging的XML格式不正确。	400 Bad Request
MalformedPolicy	Bucket policy检查不通过。	400 Bad Request
MalformedPOSTRequest	POST请求的请求体不是结构化良好的多段或形式化数据。	400 Bad Request
MalformedQuotaError	Quota的XML格式不正确。	400 Bad Request
MalformedXML	当用户发送了一个配置项的错误格式的XML会出现这样的错误。错误消息是：“The XML you provided was not well-formed or did not validate against our published schema.”。	400 Bad Request
MaxMessageLengthExceeded	请求消息过长。	400 Bad Request
MaxPostPreDataLengthExceededError	在上传文件前面的POST请求域过大。	400 Bad Request
MetadataTooLarge	元数据消息头超过了允许的最大元数据大小。	400 Bad Request
MethodNotAllowed	指定的方法不允许操作在请求的资源上。 对应返回的Message为：Specified method is not supported.	405 Method Not Allowed
MissingContentLength	必须要提供HTTP消息头中的Content-Length字段。	411 Length Required
MissingRegion	请求中缺少Region信息，且系统无默认Region。	400 Bad Request
MissingRequestBodyError	当用户发送一个空的XML文档作为请求时会发生。错误消息是：“Request body is empty.”。	400 Bad Request

错误码	描述	HTTP状态码
MissingRequiredHeader	请求中缺少必要的头域。	400 Bad Request
MissingSecurityHeader	请求缺少一个必须的头。	400 Bad Request
NoSuchBucket	指定的桶不存在。	404 Not Found
NoSuchBucketPolicy	桶policy不存在。	404 Not Found
NoSuchCORSConfiguration	CORS配置不存在。	404 Not Found
NoSuchCustomDomain	请求的用户域不存在。	404 Not Found
NoSuchKey	指定的Key不存在。	404 Not Found
NoSuchLifecycleConfiguration	请求的LifeCycle不存在。	404 Not Found
NoSuchPolicy	给定的policy名字不存在。	404 Not Found
NoSuchUpload	指定的多段上传不存在。Upload ID不存在，或者多段上传已经终止或完成。	404 Not Found
NoSuchVersion	请求中指定的version ID与现存的所有版本都不匹配。	404 Not Found
NoSuchWebsiteConfiguration	请求的Website不存在。	404 Not Found
NotImplemented	用户提供的消息头功能上还没有实现。	501 Not Implemented
NotSignedUp	账户未在系统中注册，必须先在系统中注册了才能使用该账户。	403 Forbidden
OperationAborted	另外一个冲突的操作当前正作用在这个资源上，请重试。	409 Conflict
PermanentRedirect	尝试访问的桶必须使用指定的节点，将以后的请求发送到这个节点。	301 Moved Permanently
PreconditionFailed	用户指定的先决条件中至少有一项没有包含。	412 Precondition Failed
Redirect	临时重定向。	307 Moved Temporarily
RequestIsNotMultiPartContent	桶POST必须是闭式的多段/表单数据。	400 Bad Request

错误码	描述	HTTP状态码
RequestTimeout	用户与Server之间的socket连接在超时时间内没有进行读写操作。	400 Bad Request
RequestTimeTooSkewed	请求的时间与服务器的时间相差太大。	403 Forbidden
RequestTorrentOfBucketError	不允许请求桶的torrent文件。	400 Bad Request
ServiceNotImplemented	请求的方法服务端没有实现。	501 Not Implemented
ServiceNotSupported	请求的方法服务端不支持。	409 Conflict
ServiceUnavailable	服务器过载或者内部错误异常。	503 Service Unavailable
SignatureDoesNotMatch	请求中带的签名与系统计算得到的签名不一致。检查您的访问密钥（AK和SK）和签名计算方法。	403 Forbidden
SlowDown	请降低请求频率。	503 Service Unavailable
System Capacity Not enough	系统空间不足异常。	403 Forbidden
TooManyCustomDomains	配置了过多的用户域。	400 Bad Request
TemporaryRedirect	当DNS更新时，请求将被重定向到桶。	307 Moved Temporarily
TooManyBuckets	用户拥有的桶的数量达到了系统的上限，并且请求试图创建一个新桶。	400 Bad Request
TooManyObjectCopied	用户单个对象被拷贝的数量超过系统上限。	400 Bad Request
TooManyWrongSignature	因高频错误请求被拒绝服务。	400 Bad Request
UnexpectedContent	该请求不支持带内容字段。	400 Bad Request
UnresolvableGrantByEmailAddress	用户提供的Email与记录中任何账户的都不匹配。	400 Bad Request
UserKeyMustBeSpecified	请求中缺少用户的AK信息。	400 Bad Request
WebsiteRedirect	Website请求缺少bucketName。	301 Moved Permanently

错误码	描述	HTTP状态码
KMS.DisabledException	SSE-KMS加密方式下，主密钥被禁用。	400 Bad Request
KMS.NotFoundException	SSE-KMS加密方式下，主密钥不存在。	400 Bad Request
RestoreAlreadyInProgress	对象正在恢复，请求冲突。	409 Conflict
ObjectHasAlreadyRestored	已经恢复的对象，禁止缩短恢复保存时间。	409 Conflict
InvalidObjectState	恢复对象不是归档存储对象。	403 Forbidden
InvalidTagError	配置桶标签时，提供了无效的Tag。	400 Bad Request
NoSuchTagSet	指定的桶没有设置标签。	404 Not Found

17.2 SDK 公共结果对象

调用ObsClient的相关接口完成后，如果未发生异常，则均会返回公共结果对象。该对象包含的内容见下表：

字段名		类型	说明
CommonMsg		Object	接口调用完成后的公共信息，包含HTTP状态码，操作失败的错误码等。
	Status	Number	HTTP状态码，小于300表明操作成功；反之，表明操作失败。
	Code	String	OBS服务端错误码，当Status小于300时为空。
	Message	String	OBS服务端错误描述，当Status小于300时为空。
	HostId	String	请求的服务端ID，当Status小于300时为空。
	RequestId	String	OBS服务端返回的请求ID。
	Id2	String	OBS服务端返回的请求ID2。
	Indicator	String	OBS服务端返回的详细错误码，当Status小于300时为空。
InterfaceResult		Object	操作成功后的结果数据，当Status大于300时为空。
	RequestId	String	OBS服务端返回的请求ID。
	Id2	String	OBS服务端返回的请求ID2。

字段名		类型	说明
	其他字段	请查阅《对象存储服务Node.js SDK API参考》。	

处理公共结果对象的示例代码如下：

```
// 引入obs库
// 使用npm安装
var ObsClient = require('esdk-obs-nodejs');
// 使用源码安装
// var ObsClient = require('./lib/obs');

// 创建ObsClient实例
const obsClient = new ObsClient({
    //推荐通过环境变量获取AKSK，这里也可以使用其他外部引入方式传入，如果使用硬编码可能会存在泄露风险。
    //您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    access_key_id: process.env.ACCESS_KEY_ID,
    secret_access_key: process.env.SECRET_ACCESS_KEY,
    //这里以中国-香港为例，其他地区请按实际情况填写
    server: 'https://obs.ap-southeast-1.myhuaweicloud.com'

});

// 调用接口进行操作，例如下载对象
obsClient.getObject({
        Bucket : 'bucketname',
        Key : 'objectname',
}, (err, result) => {
        if(!err){
            if(result.CommonMsg.Status < 300){
                // 获取RequestId
                console.log('RequestId-->' + result.InterfaceResult.RequestId);
                // 获取其他参数
                console.log('Content-->' + result.InterfaceResult.Content.toString());
            }else{
                // 获取Code和Message
                console.log('Code-->' + result.CommonMsg.Code);
                console.log('Message-->' + result.CommonMsg.Message);
            }
        }
});
```

17.3 日志分析

日志配置

OBS Node.js SDK基于log4js开源库提供了日志功能，您可以通过ObsClient.initLog开启日志功能并进行配置。示例代码如下：

```
obsClient.initLog({
        name: 'test', // 日志名称
        file_full_path:'./logs/OBS-SDK.log', // 配置日志文件路径
        max_log_size:20480, // 配置日志文件大小，单位：字节
        backups:10, // 配置最大可保留的日志文件个数
        level:'warn', // 配置日志级别
        log_to_console:true // 配置是否将日志打印到console
});
```


说明

- 日志功能默认是关闭的，需要主动开启。
- 使用**file_full_path参数**配置日志文件路径，可配置相对路径或绝对路径。

日志内容格式

SDK日志格式为：日志时间|日志级别|调用接口|日志内容。示例如下：

```
2017/10/12 10:21:05 666|INFO |ListBuckets|enter ListBuckets...
2017/10/12 10:21:05 672|INFO |ListBuckets|prepare request parameters ok,then Send request to service start
2017/10/12 10:21:05 715|INFO |ListBuckets|2017-10-12 10:21:05|http cost 34 ms|0|
2017/10/12 10:21:05 716|INFO |ListBuckets|get response start, statusCode:200
```

日志级别

当系统出现问题需要定位且当前的日志无法满足要求时，可以通过修改日志的级别来获取更多的信息。其中debug日志信息最丰富，error日志信息最少。

具体说明如下：

- debug：调试级别，如果设置为这个级别，将打印SDK记录的所有日志。
- info：信息级别，如果设置为这个级别，除了打印warn级别的信息外，还将打印HTTP/HTTPS请求的耗时时间等信息。
- warn：告警级别，如果设置为这个级别，除了打印error级别的信息外，还将打印一些关键事件的信息。
- error：错误级别，如果设置为这个级别，仅打印发生异常时的错误信息。

17.4 缺少模块异常

使用OBS Node.js SDK进行二次开发时如果报缺少模块异常，如“Cannot find module 'xml2js'”，请确保依赖库已正确安装，参见**安装SDK**。

17.5 连接超时异常

如果调用接口时出现“connect ETIMEDOUT”错误，表明连接超时，其原因一般是服务地址（Endpoint）错误或网络不通导致无法连接OBS服务，此时请检查服务地址和网络状况。

17.6 签名不匹配异常

如果从返回结果的CommonMsg.Status中获取到的HTTP状态码为403，CommonMsg.Code中获取到的OBS服务端错误码为SignatureDoesNotMatch，请检查AK/SK是否有误。

18 常见问题

18.1 如何指定 Content-SHA256?

上传对象和上传段支持携带x-obs-content-sha256头域，该头域值为请求消息体256-bit SHA256值转十六进制值，计算方式为Hex(SHA256Hash(<payload>)，服务端会对携带此头域的请求计算其消息体的sha256值做校验（性能会有部分下降，在安全上推荐该算法），上传对象示例代码如下：

```
// 引入obs库
// 使用npm安装
var ObsClient = require('esdk-obs-nodejs');
var crypto = require('crypto');
var fs = require('fs');
// 使用源码安装
// var ObsClient = require('./lib/obs');

// 创建ObsClient实例
const obsClient = new ObsClient({
    //推荐通过环境变量获取AKSK，这里也可以使用其他外部引入方式传入，如果使用硬编码可能会存在泄露风险。
    //您可以登录访问管理控制台获取访问密钥AK/SK，获取方式请参见https://support.huaweicloud.com/intl/zh-cn/usermanual-ca/ca_01_0003.html
    access_key_id: process.env.ACCESS_KEY_ID,
    secret_access_key: process.env.SECRET_ACCESS_KEY,
    //这里以中国-香港为例，其他地区请按实际情况填写
    server: 'https://obs.ap-southeast-1.myhuaweicloud.com'

});

let filePath = 'D:\\example.xml'
let sha256 = crypto.createHash('sha256').update(fs.readFileSync(filePath, 'utf8'), 'utf8').digest('hex')

obsClient.putObject({
        Bucket : 'bucketname',
        Key : 'objectname',
        SourceFile : filePath,  // filePath为待上传的本地文件路径，需要指定到具体的文件名
        ContentSha256 : sha256
}, (err, result) => {
        if(err){
            console.error('Error-->' + err);
        }else{
            console.log('Status-->' + result.CommonMsg.Status);
        }
});
```

须知

Nodejs sdk 同时支持MD5与SHA256校验，在安全上更推荐使用SHA256算法

18.2 为什么 SDK 源码中包含 acs.amazonaws.com 关键字?

OBS SDK 为了兼容访问aws s3服务，在源码中会包含acs.amazonaws.com关键字的固定格式，SDK仅作常量使用，不会向该格式有其他处理，也不会对其访问。SDK会自动兼容该场景，用户可不感知。比如**Nodejs SDK**:

```
Utils.prototype.URIAdapter = function(value, signatureContext){
        value = value || '';
        value = String(value);
        if(signatureContext.signature === 'obs'){
                if(obsAllowedUri.indexOf(value) >= 0){
                        return value;
                }
                if(value === 'AllUsers' || value === 'http://acs.amazonaws.com/groups/global/AllUsers'){
                        return 'Everyone';
                }

                return '';

        }
        if(v2AllowedUri.indexOf(value) >= 0){
                return value;
        }
        if(value === 'Everyone' || value === 'AllUsers'){
                return 'http://acs.amazonaws.com/groups/global/AllUsers';
        }
        if(value === 'AuthenticatedUsers'){
                return 'http://acs.amazonaws.com/groups/global/AuthenticatedUsers';
        }
        if(value === 'LogDelivery'){
                return 'http://acs.amazonaws.com/groups/s3/LogDelivery';
        }
        return '';
};
```

A API 参考

如果您想要了解OBS Node.js SDK各API的所有参数及定义，请参考**《对象存储服务 Node.js SDK API参考》**。

B 修订记录

发布日期	修订记录
2023-11-30	第三次正式发布。
2023-08-26	第二次正式发布。 新增章节**如何指定Content-SHA256?** 新增章节**为什么SDK源码中包含acs.amazonaws.com关键字?**
2020-03-31	第一次正式发布。