



# 天翼云媒体存储 (Browser.js) SDK 使用指导书

2023-07-07

天翼云科技有限公司

# 目 录

1.	<u>SDK 安装</u> .....	1
1.1.	<u>开发环境</u> .....	1
1.2.	<u>安装 SDK</u> .....	1
2.	<u>初始化</u> .....	3
3.	<u>桶相关接口</u> .....	5
3.1.	<u>获取桶访问权限</u> .....	5
3.2.	<u>设置桶访问权限</u> .....	6
3.3.	<u>获取桶策略</u> .....	9
3.4.	<u>设置桶生命周期配置</u> .....	10
3.5.	<u>获取桶生命周期配置</u> .....	13
3.6.	<u>删除桶生命周期配置</u> .....	14
3.7.	<u>设置桶版本控制状态</u> .....	15
3.8.	<u>获取桶版本控制状态</u> .....	17
4.	<u>对象相关接口</u> .....	19
4.1.	<u>获取对象列表</u> .....	19
4.2.	<u>上传对象</u> .....	20
4.3.	<u>下载对象</u> .....	22
4.4.	<u>复制对象</u> .....	23
4.5.	<u>删除对象</u> .....	24
4.6.	<u>批量删除对象</u> .....	25
4.7.	<u>设置对象访问权限</u> .....	26
4.8.	<u>获取对象元数据</u> .....	29
4.9.	<u>获取对象访问权限</u> .....	30
4.10.	<u>设置对象标签</u> .....	31
4.11.	<u>获取对象标签</u> .....	32
4.12.	<u>删除对象标签</u> .....	33
4.13.	<u>获取多版本对象列表</u> .....	34
4.14.	<u>服务端加密</u> .....	36
5.	<u>分片上传接口</u> .....	38
5.1.	<u>融合接口</u> .....	38
5.2.	<u>初始化分片上传任务</u> .....	40
5.3.	<u>上传分片</u> .....	41
5.4.	<u>合并分片</u> .....	42
5.5.	<u>列举分片上传任务</u> .....	44
5.6.	<u>列举已上传的分片</u> .....	46
5.7.	<u>复制分片</u> .....	47
5.8.	<u>取消分片上传任务</u> .....	49
6.	<u>安全凭证服务(STS)</u> .....	51
6.1.	<u>获取临时 token</u> .....	51
6.2.	<u>使用临时 token</u> .....	51



# 1. SDK 安装

## 1.1. 开发环境

- JavaScript SDK 需浏览器支持基本的 HTML5 特性（支持 IE10 以上浏览器）。
- 登录访问管理控制台，获取您的项目 accessKey 和 secretKey。
- 配置存储桶的 CORS 规则，AllowHeader 需配成\*，根据项目的需要设置 AllowedMethods、AllowedOrigins、ExposeHeaders、MaxAgeSeconds 等。
- 从 [Eclipse 官网](#) 下载并安装 Eclipse IDE for JavaScript and Web Developer，或者任何您喜欢的 js 开发工具。

## 1.2. 安装 SDK

- 方式一

直接通过在线的方式加载托管的开发工具包 SDK。使用<script>标签元素添加到您的 HTML 页面：

```
<script src="https://help.qhoss.xstore.ctyun.cn/sdk/javascript/1.0/xos-js-sdk.js"></script>
```

在 script 标签引用 SDK 后，通过它的构造函数可以创建 SDK 实例对象，从而使用相应的功能接口。

- 方式二

天翼云媒体存储兼容 AWS s3Client 接口，您可以通过 AWS s3Client 接口使用天翼云媒体存储。若您需要使用 AWS s3Client 接口，在您的项目中可以引入aws-sdk：

```
<script src="https://sdk.amazonaws.com/js/aws-sdk-SDK_VERSION_NUMBER.min.js"></script>
```



SDK\_VERSION\_NUMBER 版本信息可以参考：

<https://docs.aws.amazon.com/AWSJavaScriptSDK/latest/>

页面中加载开发工具包之后，可从全局变量 `AWS`（或 `window.AWS`）使用开发工具包。如果您使用 [\*browserify\*](#) 打包代码和模块依赖项，则使用 `require` 加载开发工具包，就像在 `Node.js` 中的操作一样。

## 2. 初始化

注意：直接在 browser js 上使用主账号存在账号泄露的风险，在 browser js 上必须使用 sts 功能生成的临时账号，此初始化流程只能用于测试。如何使用 sts 初始化参考 [安全凭证服务\(STS\)](#)。

使用 SDK 功能前，需要新建 Client，代码如下：

```
let S3Demo = {
  credentials: {
    accessKeyId: S3Config.accessKeyId,
    secretAccessKey: S3Config.secretAccessKey,
  },
  s3Client: null,

  // 初始化 s3 对象
  init: function() {
    let config = {
      credentials: this.credentials,
      endpoint: S3Config.endpoint,
      // httpOptions: { timeout: 50000 }, // 设置超时时间，默认 12
      // 0000，单位毫秒
      // maxRetries: 3, // 设置请求重试次数
      // signatureVersion: "v4", // 设置签名类型，js 预签名默认是 v
      2
    };
    this.s3Client = new AWS.S3(config);
  }
}
```



参数	说明
credentials	用户账号信息，包含 accessKeyId 和 secretAccessKey
endpoint	天翼云资源池的地址，必须指定 http 或 https 前缀

## 3. 桶相关接口

### 3.1. 获取桶访问权限

#### 3.1.1. 功能说明

您可以使用 `getBucketAcl` 接口获取桶的访问权限(ACL)，以下代码展示如何获取桶的访问权限。

在调用 `getBucketAcl` 方法时，您需要传递的唯一参数是所选存储桶的名称。当前访问控制列表配置由在传递到回调函数的 `data` 参数中返回。

#### 3.1.2. 代码示例

```
var bucketParams = {  
    // 您需要传递的唯一参数是所选存储桶的名称  
    Bucket: "<your-bucket-name>",  
};  
  
// call s3Client to retrieve policy for selected bucket  
s3Client.getBucketAcl(bucketParams, function (err, data) {  
    if (err) {  
        console.log("Error", err);  
    } else if (data) {  
        console.log("Success", data.Grants);  
    }  
});
```

#### 3.1.3. 请求参数

参数	类型	说明	是否必要
Bucket	String	桶名称	是



### 3.1.4. 返回结果

参数	类型	说明
Grantee	Object	桶访问权限配置
Permission	String	桶访问权限

## 3.2. 设置桶访问权限

### 3.2.1. 功能说明

您可以使用 setBucketAcl 接口设置桶的访问权限(ACL)，以下代码展示如何设置桶的访问权限。

使用访问控制列表（ACL）在存储桶上设置桶访问权限。要设置存储桶的 ACL，您必须具有 WRITE\_ACP 权限。您可以使用以下两种方式之一来设置存储桶的权限：在请求正文中指定 ACL、使用请求标头指定权限。注意：您不能同时使用正文和请求标头指定访问权限。

- xos 支持一组预定义的 ACL，称为 canned ACL。每个 canned ACL 都有一组预定义的被许可人和权限，预定义的访问策略使用请求标头指定权限，见下表：

权限	描述	SDK 对应值
私有读写	所有者获得 FULL_CONTROL。其他人没有访问权限（默认）	private
公共读 私有写	所有者获得 FULL_CONTROL。AllUsers 组获得 READ 访问权限	public-read
公共读 写	所有者获得 FULL_CONTROL。AllUsers 组具有“读取”和“写入”访问权限。通常不建议在存储桶上授予此权限	public-read-write

- 指定显式访问权限和被授权用户。这些参数映射到 s3Client 在 ACL 中支持的权限集。使用 AccessControlList 设置桶访问权限时，可以设置特定用户对桶的访问权限。桶的 AccessControlList 权限如下表：

权限	SDK 对应值
可读，可列出桶下的对象。	READ

权限	SDK 对应值
可写，可创建/删除/覆盖写该桶下的对象。	WRITE
可读取桶的 acl 规则。	READ_ACP
可修改桶的 acl 规则。	WRITE_ACP
完全权限。拥有以上所有权限。	FULL_CONTROL

### 3.2.2. 代码示例

使用 canned ACL 设置桶的访问权限示例代码如下：用预定义访问策略设置桶权限。

```
s3Client.putBucketAcl({
  Bucket: "<your-bucket-name>",
  // 设置桶访问权限
  ACL: private | public-read | public-read-write,
}, (err, data) => {
  if (err) {
    console.error(err);
  } else {
    console.log(data);
  }
});
```

使用 AccessControlList 设置桶的访问权限示例代码如下：以下示例替换了存储桶上的现有 ACL。授予存储桶所有者（使用所有者 ID 指定）FullControl 权限，并向用户组授予写入权限。

```
var params = {
  Bucket: "<your-bucket-name>",
  AccessControlPolicy: {
    Grants: [
      {
        Grantee: {
```

```

        // The canonical ID of the user. This ID is an obfu
scated form of your XOS account number.
        ID: "<grantee-canonical-id>",
        Type: "CanonicalUser",
    },
    Permission: "FULL_CONTROL",
},
],
Owner: {
    ID: "<owner-canonical-id>",
},
});
s3Client.putBucketAcl(params, function (err, data) {
    if (err) {
        console.error(err);
    } else {
        console.log(data);
    }
});

```

### 3.2.3. 请求参数

- 预定义的 ACL

参数	类型	说明	是否必要
Bucket	String	桶名称	是
ACL	String	桶访问权限，参考上面的 canned ACL	是

- 指定显式访问权限和被授权用户 AccessControlList

参数	类型	说明	是否必要
Bucket	String	桶名称	是
AccessControlPolicy	Object	访问权限和被授权用户对象	是

关于 AccessControlPolicy 的说明:

Owner 参数用来指定桶的所有者信息, Grants 参数用来指定被授权的用户信息。其中和权限有关的参数是 Permission, 您可以选择 *FULL\_CONTROL / READ / WRITE / READ\_ACP / WRITE\_ACP* 来设定相应的权限。

### 3.2.4. 返回结果

根据返回码判断是否操作成功。

## 3.3. 获取桶策略

### 3.3.1. 功能说明

在调用 `getBucketPolicy` 方法时, 您需要传递的唯一参数是所选存储桶的名称。如果存储桶当前具有策略, 该策略在由传递到回调函数的 `data` 参数中返回, 如果所选存储桶没有策略, 该信息将在 `error` 参数中返回给回调函数。

### 3.3.2. 代码示例

```
var bucketParams = {
  Bucket: "<your-bucket-name>",
};
// call s3Client to retrieve policy for selected bucket
s3Client.getBucketPolicy(bucketParams, function(err, data) {
  if (err) {
    console.log("Error", err);
  } else if (data) {
    console.log("Success", data.Policy);
  }
});
```

```
}
});
```

### 3.3.3. 请求参数

参数	类型	说明	是否必要
Bucket	String	桶名称	是

### 3.3.4. 返回结果

参数	类型	说明
Policy	Object	桶策略对象

关于 Policy 一些说明

参数	说明
Resource	指定 statement 起作用的一组资源，支持通配符“*”，表示所有资源
Action	指定本条 statement 作用的操作，Action 字段为 <b>媒体存储</b> 支持的所有操作集合，以字符串形式表示，不区分大小写。支持通配符“*”，表示该资源能进行的所有操作
Effect	指定本条 statement 的权限是允许还是拒绝，Effect 的值必须为 Allow 或者 Deny
Principal	被授权人，指定本条 statement 权限针对的 Domain 以及 User，支持通配符“*”，表示所有用户（匿名用户）。当对 Domain 下所有用户授权时，Principal 格式为 arn:aws:iam:::user/*。当对某个 User 进行授权时，Principal 格式为 arn:aws:iam:::user/<your-user-name>

## 3.4. 设置桶生命周期配置

### 3.4.1. 功能说明

生命周期管理可以通过设置规则实现自动清理过期的对象，优化存储空间。本文介绍如何设置桶（Bucket）生命周期配置。您可以使用 putBucketLifecycleConfiguration 接口设置桶的生命周期配置，配置规则可以通过匹配对象 key 前缀、标签的方法设置当前版本或者历史版本对象的过期时间，对象过期后会被自动删除。

### 3.4.2. 代码示例

```
var params = {  
  Bucket: "<your-bucket-name>",  
  LifecycleConfiguration: {  
    Rules: [  
      {  
        Expiration: {  
          Days: 3650,  
        },  
        Filter: {           //required  
          Prefix: '',  
        },  
        ID: 'TestOnly',  
        Status: 'Enabled', //required  
        Transitions: [  
          {  
            Days: 365,  
            StorageClass: 'GLACIER',  
          },  
        ],  
        NoncurrentVersionExpiration: {  
          NoncurrentDays: 123,  
        },  
        AbortIncompleteMultipartUpload: {  
          DaysAfterInitiation: 123  
        }  
      },  
    ],  
  },  
}
```

```

    }
};

s3Client.putBucketLifecycleConfiguration(params, function (err, data) {
    if (err) {
        console.log("Error", err);
    } else {
        console.log("Success", data);
    }
});

```

### 3.4.3. 请求参数

参数	类型	说明	是否必要
Bucket	String	桶名称	是
LifecycleConfiguration	LifecycleConfiguration	封装了生命周期规则的数组，最多包含 1000 条规则	是

关于 LifecycleConfiguration 一些说明

参数	说明
Prefix	需要过滤的前缀
Tag	需要过滤的 Tag 组包含 key 和 value
Transitions	存储类型自动转换规则
NoncurrentVersionExpiration	针对历史版本过期
AbortIncompleteMultipartUpload	清理过期未完成分片

### 3.4.4. 返回结果

根据返回码判断是否操作成功。

## 3.5. 获取桶生命周期配置

### 3.5.1. 功能说明

您可以使用 `getBucketLifecycleConfiguration` 接口获取桶的生命周期配置。

### 3.5.2. 代码示例

```
var params = {  
    Bucket: "<your-bucket-name>",  
};  
  
s3Client.getBucketLifecycleConfiguration(params, function (err, data)  
a) {  
    if (err) {  
        console.log("Error", err);  
    } else {  
        console.log("Success", data);  
    }  
});
```

### 3.5.3. 请求参数

参数	类型	说明	是否必要
Bucket	String	桶名称	是

### 3.5.4. 返回结果

参数	类型	说明
LifecycleConfiguration	LifecycleConfiguration	一个描述生命周期管理的规则数组，一条规则包含了规则 ID、匹配的对象 key 前缀、匹配的对象标签信息、当前版本



参数	类型	说明
		对象过期时间、历史版本对象过期时间和是否生效标识等信息

关于 LifecycleConfiguration 一些说明

参数	说明
Prefix	需要过滤的前缀
Tag	需要过滤的 Tag 组包含 key 和 value
Transitions	存储类型自动转换规则
NoncurrentVersionExpiration	针对历史版本过期
AbortIncompleteMultipartUpload	清理过期未完成分片

## 3.6. 删除桶生命周期配置

### 3.6.1. 功能说明

您可以使用 `deleteBucketLifecycle` 接口删除桶的生命周期配置。

### 3.6.2. 代码示例

```
// Set the parameters for s3Client.deleteBucketLifecycle
var bucketParams = {
    Bucket: "<your-bucket-name>",
};

s3Client.deleteBucketLifecycle(bucketParams, function (err, data) {
    if (err) {
        console.log("Error", err);
    } else if (data) {
        console.log("Success", JSON.stringify(data.CORSRules));
    }
});
```

### 3.6.3. 请求参数

参数	类型	说明	是否必要
Bucket	String	桶名称	是

### 3.6.4. 返回结果

根据返回码判断是否操作成功。

## 3.7. 设置桶版本控制状态

### 3.7.1. 功能说明

通过媒体存储提供的版本控制，您可以在一个桶中保留多个对象版本。例如，image.jpg (版本 1)和 image.jpg(版本 2)。如果您希望防止自己意外覆盖和删除版本，或存档对象，以便您可以检索早期版本的对象，您可以开启版本控制功能。您必须在您的存储桶上显式启用版本控制。默认情况下，版本控制处于禁用状态。无论您是否已启用版本控制，您的存储桶中的每个对象都具有版本 ID。如果未启用版本控制，则版本 ID 值被设置为空。如果已启用版本控制，则对象会被指定唯一版本 ID 值。在存储桶上启用版本控制时，该存储桶中的现有对象（如果有）不会发生更改：版本 ID（空）、内容和权限保持为相同。

在开启版本控制功能后，上传同名对象将不再删除旧对象，而是添加一个新的对象。普通的删除操作也不会将对象彻底删除，而是添加一个 Delete Marker 作为标识。容器开启版本控制功能之后，无法再关闭该功能，只能暂停。

### 3.7.2. 代码示例

```
//开启版本控制
var params = {
    Bucket: "<your-bucket-name>",
    VersioningConfiguration: {
        Status: 'Enabled'
    },
};
```

```
s3Client.putBucketVersioning(params, function (err, data) {  
  if (err) {  
    console.Log("Error", err);  
  } else {  
    console.Log("Success", data);  
  }  
});  
  
//暂停版本控制  
var params = {  
  Bucket: "<your-bucket-name>",  
  VersioningConfiguration: {  
    Status: 'Suspended'  
  },  
};  
s3Client.putBucketVersioning(params, function (err, data) {  
  if (err) {  
    console.Log("Error", err);  
  } else {  
    console.Log("Success", data);  
  }  
});
```

### 3.7.3. 请求参数

参数	类型	说明	是否必要
Bucket	String	桶名称	是
VersioningConfiguration	Object	桶版本控制配置对象	是
Status	String	桶版本	

### 3.7.4. 返回结果

根据返回码判断是否操作成功。

## 3.8. 获取桶版本控制状态

### 3.8.1. 功能说明

媒体存储支持设置和存储一个对象的多个版本，使您更方便地检索和还原各个版本，以便可以快速恢复数据。您可以使用 `getBucketVersioning` 接口获取版本控制状态，代码示例：

### 3.8.2. 代码示例

```
// Create the parameters for calling getBucketVersioning
var bucketParams = {
    Bucket: "<your-bucket-name>",
};

// call S3 to getBucketVersioning
s3Client.getBucketVersioning(bucketParams, function (err, data) {
    if (err) {
        console.log("Error", err);
    } else {
        console.log("Success", data);
    }
});
```

### 3.8.3. 请求参数

参数	类型	说明	是否必要
Bucket	String	桶名称	是

### 3.8.4. 返回结果

参数	类型	说明
Status	String	桶版本控制状态

## 4. 对象相关接口

### 4.1. 获取对象列表

#### 4.1.1. 功能说明

您可以使用 `listObjects` 接口列举对象，以下代码展示如何简单列举对象。

#### 4.1.2. 代码示例

```
var params = {  
    Bucket: "<your-bucket-name>",  
};  
s3Client.listObjects(params, function (err, data) {  
    if (err)  
        console.log(err, err.stack);  
    else  
        console.log(data);  
});
```

#### 4.1.3. 请求参数

参数	类型	说明	是否必要
Bucket	String	桶名称	是
MaxKeys	Int	设置响应中返回的最大键数。默认值和可设置最大值均为1000	否
Prefix	String	指定列出对象的键名需要包含的前缀	否
Marker	String	用于在某一个具体的键名后列出对象，可指定存储桶中的任一个键名	否

#### 4.1.4. 返回结果

`listObjects` 中可设置的列举相关参数如下：

参数	描述
Bucket	包含对象的存储桶的名称
Delimiter	分隔符是用于对对象 objectKey 进行分组的字符。所有名字包含指定的 prefix (可以不指定) 且第一次出现 delimiter 字符之间的对象作为一组返回对象
Marker	指定列出存储桶中的对象时以其开头的键
MaxKeys	设置响应中返回的最大键数。默认情况下, 该操作最多返回 1,000 个键名
Prefix	将响应限制为以指定前缀开头的键

## 4.2. 上传对象

### 4.2.1. 功能说明

您可以使用 putObject 接口上传对象, 可以上传最大不超过 5GB 的文件, 超过 5GB 的文件可以通过分片上传操作上传到媒体存储服务, 对象 key 的命名使用 UTF-8 编码, 长度必须在 1~1023 字节之间, 不能反斜线 (\) 开头, 以下代码展示如何上传一个对象。

### 4.2.2. 代码示例

简单文本上传:

```
var params = {
  Bucket: "<your-bucket-name>",
  Key: "<your-key-name>",
  Body: 'STRING_VALUE',
  ACL: "private", // 初始化 acl 权限, 默认为 private, "private"|"public-read"|"public-read-write"
  ContentType: "text/plain", // 设置 contentType, 默认是 application/octet-stream
};
s3Client.putObject(params, function (err, data) {
  if (err)
    console.log(err, err.stack);
});
```

```
    else
        console.log(data);
});
```

文件上传，指定待上传的文件 File 对象，例如在 HTML 页面中使用类型为“file”的 input 标签：

```
var params = {
    Bucket: "<your-bucket-name>",
    Key: "<your-key-name>",
    Body: document.getElementById('input-file').files[0],
    ACL: "private", // 初始化 acl 权限，默认为 private, "private"|"public-read"|"public-read-write"
    ContentType: "text/plain", // 设置 contentType, 默认是 application/octet-stream
};
s3Client.putObject(params, function (err, data) {
    if (err)
        console.log(err, err.stack);
    else
        console.log(data);
});
```

### 4.2.3. 请求参数

参数	类型	说明	是否必要
Bucket	String	桶名称	是
Key	String	对象名称	是
Body	String	上传的对象数据流	是
ACL	String	对象访问控制权限	否
ContentType	String	对象类型	是



#### 4.2.4. 返回结果

参数	类型	说明
ETag	String	上传对象后对应的 Entity Tag

### 4.3. 下载对象

#### 4.3.1. 功能说明

您可以使用 getObject 接口下载对象，以下代码展示如何下载一个对象。

#### 4.3.2. 代码示例

```
var params = {  
  Bucket: "<your-bucket-name>",  
  Key: "<your-key-name>",  
};  
s3Client.getObject(params, function (err, data) {  
  if (err)  
    console.log(err, err.stack);  
  else  
    console.log(data);  
});
```

#### 4.3.3. 请求参数

参数	类型	说明	是否必要
Bucket	String	桶名称	是
Key	String	对象名称	是

#### 4.3.4. 返回结果

参数	类型	说明
Body	Array	对象数据内容字节数组

参数	类型	说明
ContentLength	Int	对象数据的长度，单位为字节
ContentType	String	数据的标准 MIME 类型
Metadata	Object	自定义元数据
ETag	String	对象的 Entity Tag
LastModified	Time	最后修改对象的时。
StorageClass	String	对象的存储类型

## 4.4. 复制对象

### 4.4.1. 功能说明

您可以使用 `copyObject` 接口复制对象，您需要设置复制的对象名，所在的桶以及目标桶和对象名。

### 4.4.2. 代码示例

```
var params = {
  Bucket: "<dst-bucket-name>",
  Key: "<dst-bucket-name>",
  CopySource: "<source-bucket-name>" + "/" + "<source-object-key>"
// 必须加上桶名前缀，
};
s3Client.copyObject(params, function (err, data) {
  if (err) {
    console.Log("Error", err);
  } else {
    console.Log("Success", data);
  }
});
```

注意事项：对象 key 会放到请求头 header 来发送，如果其中包含中文的话会出现编码格式问题，解决方法是可以将 header 中的中文字符进行 URL 编码。### 请求参数

参数	类型	说明	是否必要
Bucket	String	目的桶名称	是
Key	String	目的桶的对象 key	是
CopySource	String	源对象地址 (bucket+key)	是

### 4.4.3. 返回结果

参数	类型	说明
CopyObjectResult	Object	包含拷贝生成对象的 Entity Tag 和最后修改时间等信息

## 4.5. 删除对象

### 4.5.1. 功能说明

您可以使用 `deleteObject` 接口删除单个对象，以下代码展示如何删除一个对象。

### 4.5.2. 代码示例

```
var params = {  
  Bucket: "<your-bucket-name>",  
  Key: "<your-key-name>",  
};  
s3Client.deleteObject(params, function (err, data) {  
  if (err)  
    console.log(err, err.stack);  
  else  
    console.log(data);  
});
```

### 4.5.3. 请求参数

参数	类型	说明	是否必要
Bucket	String	桶名称	是
Key	String	对象名称	是

### 4.5.4. 返回结果

根据返回码判断是否操作成功。

## 4.6. 批量删除对象

### 4.6.1. 功能说明

您可以使用 `deleteObjects` 接口批量删除多个对象，代码示例如下。

### 4.6.2. 代码示例

```
var params = {
  Bucket: "<your-bucket-name>",
  Delete: {
    Objects: [
      {
        'Key': 'ExampleObject.txt',
      },
      {
        'Key': 'ExampleObject1.txt',
      },
    ]
  },
};

s3Client.deleteObjects(params, function (err, data) {
  if (err) {
```

```
        console.Log("Error", err);
    } else {
        console.Log("Success");
    }
});
```

### 4.6.3. 请求参数

参数	类型	说明	是否必要
Bucket	String	桶名称	是
Delete	Delete	要删除的对象 key 列表	是

### 4.6.4. 返回结果

根据返回码判断是否操作成功。

## 4.7. 设置对象访问权限

### 4.7.1. 功能说明

与桶访问权限类似，对象访问权限设置方式具有 Canned ACL 与 AccessControlPolicy 两种。需要注意的是，对象的访问优先级要高于桶访问权限。比如桶访问权限是 private，但是对象访问权限是 public read，则所有用户都可以访问该对象。默认情况下，只有对象的拥有者才能访问该对象，即对象的访问权限默认是 private。

### 4.7.2. 代码示例

使用 canned ACL 设置桶的访问权限，对象访问权限包含了：private(私有读写)，public-read (公共读私有写)，public-read-write (公共读写)，示例代码如下：

```
var params = {
    Bucket: "<your-bucket-name>",
    Key: "<your-key-name>",
    ACL: "public-read-write",
```

```
};

s3Client.putObjectAcl(params, function (err, data) {
  if (err)
    console.log(err, err.stack);
  else
    console.log(data);
});
```

使用 ACL 对象授予权限，指定显式访问权限和被授权用户。这些参数映射到 s3Client 在 ACL 中支持的权限集。使用 AccessControlList 设置桶访问权限时，可以设置特定用户对桶的访问权限：

```
var params = {
  Bucket: "<your-bucket-name>",
  Key: "<your-key-name>",
  AccessControlPolicy: {
    Grants: [
      {
        Grantee: {
          Type: "CanonicalUser",
          ID: "<grantee-canonical-id>"
        },
        Permission: "FULL_CONTROL"
      },
    ],
  },
  Owner: {
    ID: "<owner-canonical-id>",
  }
}
```

```

    }
};
s3Client.putObjectAcl(params, function (err, data) {
    if (err) console.log(err, err.stack);
    else console.log(data);
});

```

### 4.7.3. 请求参数

- 预定义的 ACL

参数	类型	说明	是否必要
Bucket	String	桶名称	是
Key	String	对象名称	是
ACL	String	桶访问权限，参考上面的 Canned ACL	是

每个 Canned ACL 都有一组预定义的被授权者和权限，下表列出了相关的预定义授权

含义:

ACL	权限	描述
private	私有读写	对象拥有者有读写权限，其他用户没有访问权限
public-read	公共读私有写	对象拥有者有读写权限，其他用户只有该对象的读权限
public-read-write	公共读写	所有用户都有该对象的读写权限

- 指定显式访问权限和被授权用户 AccessControlPolicy

参数	类型	说明	是否必要
Bucket	String	桶名称	是
Key	String	对象名称	是
AccessControlPolicy	Object	访问权限和被授权用户对象	是

关于 AccessControlPolicy 的说明:

Owner 参数用来指定桶的所有者信息，Grants 参数用来指定被授权的用户信息。其中和权限有关的参数是 Permission，您可以选择 `FULL_CONTROL` / `READ` / `WRITE` / `READ_ACP` / `WRITE_ACP` 来设定相应的权限。

#### 4.7.4. 返回结果

根据返回码判断是否操作成功。

### 4.8. 获取对象元数据

#### 4.8.1. 功能说明

您可以使用 `headObject` 接口获取对象元数据。`headObject` 操作的请求参数与 `getObject` 类似，但是 `headObject` 返回的 http 响应中没有对象数据。

#### 4.8.2. 代码示例

```
var params = {
  Bucket: "<your-bucket-name>",
  Key: "<your-key-name>",
};
s3Client.headObject(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success: ", data.ContentType);
  }
});
```

#### 4.8.3. 请求参数

参数	类型	说明	是否必要
Bucket	String	桶名称	是



参数	类型	说明	是否必要
Key	String	对象 key	是

#### 4.8.4. 返回结果

参数	类型	说明
AcceptRanges	String	跨域访问规则
LastModified	Date	最后修改时间
ContentLength	Long	对象大小
ETag	String	资源标识符
ContentType	String	对象类型
Metadata	Object	对象元数据
StorageClass	String	对象存储类型

### 4.9. 获取对象访问权限

#### 4.9.1. 功能说明

您可以使用 getObjectAcl 接口获取对象访问的权限。以下代码展示如何获取对象的访问权限。

#### 4.9.2. 代码示例

```
var params = {
  Bucket: "<your-bucket-name>",
  Key: "<your-key-name>",
};
s3Client.getObjectAcl(params, function (err, data) {
  if (err)
    console.log(err, err.stack);
  else
    console.log(data);
});
```

### 4.9.3. 请求参数

参数	类型	说明	是否必要
Bucket	String	桶名称	是
Key	String	对象名	是

### 4.9.4. 返回结果

参数	类型	说明
Owner	Object	对象的 owner 信息
Grants	Object	grants 授权信息，包含了每个用户与其权限 Permission

## 4.10. 设置对象标签

### 4.10.1. 功能说明

您可以使用 `putObjectTagging` 接口为对象设置标签。标签是一个键值对，每个对象最多可以有 10 个标签。bucket 的拥有者默认拥有给 bucket 中的对象设置标签的权限，并且可以将权限授予其他用户。每次执行 `PutObjectTagging` 操作会覆盖对象已有的标签信息。每个对象最多可以设置 10 个标签，标签 Key 和 Value 区分大小写，并且 Key 不可重复。每个标签的 Key 长度不超过 128 字节，Value 长度不超过 256 字节。SDK 通过 HTTP header 的方式设置标签且标签中包含任意字符时，需要对标签的 Key 和 Value 做 URL 编码。设置对象标签信息不会更新对象的最新更改时间。

### 4.10.2. 代码示例

```
var params = {  
    Bucket: "<your-bucket-name>",  
    Key: "<your-key-name>",  
    Tagging: {  
        TagSet: [  
            {  
                Key: 'key1',
```

```
        Value: 'value1'
      },
    ]
  }
};

s3Client.putObjectTagging(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

### 4.10.3. 请求参数

参数	类型	说明	是否必要
Bucket	String	桶名称	是
Key	String	对象 key	是
Tagging	Object	设置的标签信息，包含了一个 Tag 结构体的数组，每个 Tag 以 Key-Value 的形式说明了标签的内容	是
VersionId	String	设置标签信息的对象的版本 Id	否

### 4.10.4. 返回结果

根据返回码判断是否操作成功。

## 4.11. 获取对象标签

### 4.11.1. 功能说明

您可以使用 getObjectTagging 接口获取对象标签。

### 4.11.2. 代码示例

```
var params = {  
    Bucket: "<your-bucket-name>",  
    Key: "<your-key-name>",  
};  
s3Client.getObjectTagging(params, function (err, data) {  
    if (err) {  
        console.log("Error", err);  
    } else {  
        console.log("Success", data.TagSet[0]);  
    }  
});
```

### 4.11.3. 请求参数

参数	类型	说明	是否必要
Bucket	String	桶名称	是
Key	String	对象 key	是
VersionId	String	设置标签信息的对象的版本 Id	否

### 4.11.4. 返回结果

参数	类型	说明
TagSet	Array	设置的标签信息，包含了一个 Tag 结构体的数组，每个 Tag 以 Key-Value 的形式说明了标签的内容

## 4.12. 删除对象标签

### 4.12.1. 功能说明

您可以使用 deleteObjectTagging 接口删除对象标签。

## 4.12.2. 代码示例

```
var params = {  
    Bucket: "<your-bucket-name>",  
    Key: "<your-key-name>",  
};  
s3Client.deleteObjectTagging(params, function (err, data) {  
    if (err) {  
        console.log("Error", err);  
    } else {  
        console.log("Success", data);  
    }  
});
```

## 4.12.3. 请求参数

参数	类型	说明	是否必要
Bucket	String	执行本操作的桶名称	是
Key	String	设置标签信息的对象 key	是
VersionId	String	设置标签信息的对象的版本 Id	否

## 4.12.4. 返回结果

根据返回码判断是否操作成功。

## 4.13. 获取多版本对象列表

### 4.13.1. 功能说明

如果桶开启了版本控制，您可以使用 listObjectVersions 接口列举对象的版本，每次 list 操作最多返回 1000 个分片上传事件，简单列举对象版本代码如下。

### 4.13.2. 代码示例

```

var params = {
    Bucket: "<your-bucket-name>",
};
s3Client.ListObjectVersions(params, function (err, data) {
    if (err) {
        console.log("Error", err);
    } else {
        console.log("Success", data);
    }
});

```

### 4.13.3. 请求参数

参数	类型	说明	是否必要
Bucket	String	桶名称	是
MaxKeys	Int	设置响应中返回的最大键数。默认值和可设置最大值均为1000	否
Prefix	String	指定列出对象的键名需要包含的前缀	否
Marker	String	用于在某一个具体的键名后列出对象，可指定存储桶中的任一个键名	否

### 4.13.4. 返回结果

参数	类型	说明
IsTruncated	Boolean	分页判断，指明是否已返回所有结果
KeyMarker	String	设定结果从 KeyMarker 之后按字母序开始返回，与 VersionIdMarker 组合使用
VersionIdMarker	String	设定结果从 KeyMarker 对象的 VersionIdMarker 之后按新旧版本排序开始返回

参数	类型	说明
Versions	Array	保存除删除标记以外的对象多版本的数组
DeleteMarkers	Array	保存删除标记的对象数组
Name	String	桶名
Prefix	String	本次查询结果的前缀
MaxKeys	String	限定此次返回对象的最大个数
CommonPrefixes	Array	公共前缀

## 4.14. 服务端加密

### 4.14.1. 功能说明

上传对象时可以指定对象的加密算法，即使设置桶的加密配置也可以加密请求上传的对象数据，服务端根据指定的加密算法对对象数据进行加密，目前支持 AES256 和国密 SM4 加密算法。

### 4.14.2. 代码示例

上传对象，并开启服务端加密：

```
//上传文件
var params = {
  Bucket: "<your-bucket-name>",
  Key: "<your-key-name>",
  Body: "<your-key-body>",
  ServerSideEncryption: "AES256"
};
s3Client.putObject(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

```
}  
});
```

### 4.14.3. 请求参数

参数	类型	说明	是否必要
Bucket	String	桶名称	是
Key	String	上传对象名	是
Body	String	上传对象流	是
ServerSideEncryption	String	服务端加密算法	是

### 4.14.4. 返回结果

根据返回码判断是否操作成功。



## 5. 分片上传接口

### 5.1. 融合接口

#### 5.1.1. 功能说明

分片上传步骤较多，包括初始化、文件切片、各个分片上传、完成上传。为了简化分片上传，可以使用 AWS.S3.ManagedUpload 接口进行分片上传。

#### 5.1.2. 代码示例

```
let key = "ExampleObject.txt"

let localFile = "E:/ExampleObject.txt"

let f = fs.createReadStream(localFile)

let upload = new AWS.S3.ManagedUpload({
  service: this.s3Client,
  partSize: 10 * 1024 * 1024, // 10M 一片，可以根据需要自己定义，每个文件不能超过 10000 分片
  params: {
    Bucket: this.bucket,
    Key: key,
    Body: f,
    ACL: "private", // 初始化 acl 权限，默认为 private, "private"|"public-read"|"public-read-write"
    ContentType: "text/plain", // 设置 contentType，默认是 application/octet-stream
  },
});

upload.on("httpUploadProgress", progress => console.log(progress))
```

```
upload.send((err, data) => {  
  if (err) {  
    console.log("Error", err);  
  } else {  
    console.log("Success", data);  
  }  
});
```

关于 Content-Type 的配置

Content-Type 用于标识文件的资源类型，比如 `image/png`，`image/jpg` 是图片类型，`video/mpeg`，`video/mp4` 是视频类型，`text/plain`，`text/html` 是文本类型，浏览器针对不同的 Content-Type 会有不同的操作，比如图片类型可以预览，视频类型可以播放，文本类型可以直接打开。`application/octet-stream` 类型会直接打开下载窗口。

有些用户反馈图片和视频无法预览的问题，主要就是 Content-Type 没有正确设置导致的；Content-Type 参数需要用户主动设置，默认是 `application/octet-stream`。在 nodejs 中，可以根据对象 key 值后缀扩展名来决定文件的 Content-Type，参考代码如下：

```
let mime = require("mime-types")  
  
let mimeType = (key) => {  
  let ret = mime.lookup(key);  
  if (ret == false) {  
    return "";  
  }  
  return ret;  
}
```

### 5.1.3. 请求参数

参数	类型	说明	是否必要
Bucket	String	桶名称	是
Key	String	对象名称	是
Body	String	对象内容	是
ACL	String	对象访问控制权限	否
ContentType	String	对象类型	是
partSize	Long	对象分片大小	是

### 5.1.4. 返回结果

参数	类型	说明
Etag	String	本次上传对象对应的 Entity Tag
Location	String	对象的 URL 信息
Key	String	对象名称
Bucket	String	桶名称

## 5.2. 初始化分片上传任务

### 5.2.1. 功能说明

使用 `createMultipartUpload` 接口创建分片上传任务，该接口会返回一个 `UploadId`，客户端使用这个 `UploadId` 来上传分片。

### 5.2.2. 代码示例

```
var createParams = {
  Bucket: "<your-bucket-name>",
  Key: "<your-key-name>"
};
s3Client.createMultipartUpload(createParams, function (err, data) {
  if (err)
    console.log(err, err.stack);
});
```

```
else
    console.log(data);
});
```

### 5.2.3. 请求参数

参数	类型	说明	是否必要
Bucket	String	桶名称	是
Key	String	对象名称	是

### 5.2.4. 返回结果

参数	类型	说明
Bucket	String	执行分片上传的桶的名称。
Key	String	本次分片上传对象的名称。
UploadId	String	本次生成分片上传任务的 id。

## 5.3. 上传分片

### 5.3.1. 功能说明

初始化分片上传任务后，可以根据指定的对象名和 Upload ID 来分片上传数据。将大文件分割成分片后上传，除了最后一个分片，每个分片的数据大小为 5MB~5GB，每个分片上传任务最多上传 10000 个分片。每一个上传的对象分片都对应一个分片号。对于同一个 Upload ID，该分片号不但唯一标识这一段数据，也标识了分片数据在整个对象内的相对位置。

### 5.3.2. 代码示例

```
var partParams = {
    Bucket: "<your-bucket-name>",
    Key: "<your-key-name>",
    // 设置分片号，范围是 1~10000
    PartNumber: "<your-part-name>",
```

```
// 设置 Upload ID
UploadId: "<your-upload-id>",
// 设置将要上传的大文件
Body: document.getElementById('input-file').files[0],
}
s3Client.uploadPart(partParams, function (err, result) {
  if(err){
    console.log('uploadPart Error ' + err);
  }else{
    console.log('uploadPart ETag ' + result.ETag);
  }
});
```

### 5.3.3. 请求参数

参数	类型	说明	是否必要
Bucket	String	桶名称	是
Body	String	对象数据流	是
Key	String	对象名称	是
UploadId	String	分片上传 id	是
PartNumber	Long	分片个数	是

### 5.3.4. 返回结果

参数	类型	说明
Etag	String	本次上传分片对应的 Entity Tag

## 5.4. 合并分片

### 5.4.1. 功能说明

使用 `completeMultipartUpload` 完成分片上传任务。合并指定分片上传任务 id 对应任务中已上传的对象分片，使之成为一个完整的文件对象。

## 5.4.2. 代码示例

```
var params = {
  Bucket: "<your-bucket-name>",
  Key: "<your-key-name>",
  // 设置 Upload ID
  UploadId: "<your-upload-id>",
  MultipartUpload: {
    Parts: [
      {
        ETag: "<your-object-etag>",
        PartNumber: "<your-part-number>"
      },
      {
        ETag: "<your-object-etag>",
        PartNumber: "<your-part-number>"
      },
      {
        ETag: "<your-object-etag>",
        PartNumber: "<your-part-number>"
      },
      ...
    ]
  },
};

s3Client.completeMultipartUpload(params, function (err, data) {
  if (err) {
    console.log('completeMultipartUpload err, ', err);
  } else {
```

```

        console.Log('completeMultipartUpload success');
    }
});

```

### 5.4.3. 请求参数

参数	类型	说明	是否必要
Bucket	String	桶名称	是
Key	String	对象名称	是
UploadId	String	对象分片 id	是
MultipartUpload	String	对象分片列表, 包含了每个已上传的分片的 ETag 和 PartNumber 等信息	是

### 5.4.4. 返回结果

参数	类型	说明
ETag	string	本次上传对象后对应的 Entity Tag
Bucket	String	执行分片上传的桶的名称
Key	String	上传文件到对象存储服务后对应的 key
Location	String	合并生成对象的 URL 信息

## 5.5. 列举分片上传任务

### 5.5.1. 功能说明

列举分片上传操作可以列出一个桶中正在进行的分片上传, 这些分片上传的请求已经发起, 但是还没完成或者被中止。listMultipartUploads 操作可以通过指定 maxUploads 参数来设置返回分片上传信息的数量, maxUploads 参数的最大值和默认值均为 1000。如果返回结果中的 isTruncated 字段为 true, 表示还有符合条件的分片上传信息没有列出, 可以通过设置请求中的 keyMarker 和 uploadIdMarker 参数, 来列出符合筛选条件的正在上传的分片信息。

## 5.5.2. 代码示例

```

var params = {
    Bucket: "<your-bucket-name>"
};
s3Client.listMultipartUploads(params, function (err, data) {
    if (err) {
        console.log('listMultipartUploads err, ', err);
    } else {
        console.log('listMultipartUploads success, ', data);
    }
});

```

## 5.5.3. 请求参数

参数	类型	说明	是否必要
Bucket	String	桶名称	是

## 5.5.4. 返回结果

参数	类型	说明
Bucket	String	执行本操作的桶名称
CommonPrefixes	Array	当请求中设置了 Delimiter 和 Prefix 属性时，所有包含指定的 Prefix 且第一次出现 Delimiter 字符的对象 key 作为一组
IsTruncated	Bool	当为 false 时表示返回结果中包含了全部符合本次请求查询条件的分片上传任务信息，否则只返回了数量为 MaxUploads 个的分片信息
MaxUploads	Int	本次返回结果中包含的分片上传任务数量的最大值
KeyMarker	String	返回分片上传任务列表中的起始对象的 key
NextKeyMarker	String	当 IsTruncated 为 true 时，NextKeyMarker 可以作为后续查询已初始化的分片上传任务请求中的 KeyMarker 的值
UploadIdMarker	String	返回分片上传任务列表中的起始 UploadId。



参数	类型	说明
NextUploadIdMarker	String	当 IsTruncated 为 true 时，NextKeyMarker 可以作为后续查询已初始化的分片上传任务请求中的 UploadIdMarker 的值
Uploads	Array	包含了零个或多个已初始化的分片上传任务信息的数组。数组中的每一项包含了分片初始化时间、分片上传操作发起者、对象 key、对象拥有者、存储类型和 UploadId 等信息

## 5.6. 列举已上传的分片

### 5.6.1. 功能说明

使用 listParts 可以根据 UploadId 列举已完成上传的分片。可以使用此接口实现断点续传，在客户端保存 UploadId 和对应的本地文件路径，重新上传的时候先通过 listParts 获取到已上传的分片，避免重复上传这些分片，从而实现断点续传。

### 5.6.2. 代码示例

```
var params = {  
  Bucket: "<your-bucket-name>",  
  Key: "<your-key-name>",  
  // 设置 Upload ID  
  UploadId: "<your-upload-id>"  
};  
s3Client.listParts(params, function (err, data) {  
  if (err) {  
    console.log('listParts err, ', err);  
  } else {  
    console.log('listParts success, ', data);  
  }  
});
```

### 5.6.3. 请求参数

参数	类型	说明	是否必要
Bucket	String	桶名称	是
Key	String	对象名称	是
UploadId	String	分片任务 id	是

### 5.6.4. 返回结果

参数	类型	说明
Bucket	String	执行本操作的桶名称
Key	String	执行本操作的对象名称
UploadId	String	分片任务 id
IsTruncated	Bool	当为 false 时表示返回结果中包含了全部符合本次请求查询条件的上传分片信息，否则只返回了数量为 MaxParts 个的分片信息
PartNumberMarker	Int	返回分片上传任务分片号
NextPartNumberMarker	Int	当 IsTruncated 为 true 时，NextPartNumberMarker 可以作为后续查询已上传分片请求中的 PartNumberMarker 的值
MaxParts	Int	本次返回结果中包含的上传分片数量的最大值
Owner	Object	分片上传对象的拥有者信息，包含了用户名和 Id 等信息
Parts	Array	包含了已上传分片信息的数组，数组中的每一项包含了该分片的 Entity tag、最后修改时间、PartNumber 和大小等信息
StorageClass	String	对象的存储类型

## 5.7. 复制分片

### 5.7.1. 功能说明

复制分片操作可以从一个已存在的对象中复制指定分片的数据。您可以使用 `uploadPartCopy` 复制分片。在复制分片前，需要使用 `initiateMultipartUpload` 接口获取一

个 upload id，在完成复制和上传分片操作之后，需要使用 completeMultipartUpload 操作组装分片成为一个对象。当复制的对象大小超过 5GB，必须使用复制分片操作完成对象的复制。除了最后一个分片外，每个复制分片的大小范围是[5MB, 5GB]。

### 5.7.2. 代码示例

```
var params = {  
    Bucket: "<dst-bucket-name>",  
    Key: "<dst-bucket-name>",  
    // 设置 Upload ID  
    UploadId: "<your-upload-id>",  
    // 设置分片号，范围是 1~10000  
    PartNumber: "<your-part-name>",  
    CopySource: "<source-bucket-name>" + "/" + "<source-object-key>"  
// 必须加上桶名前缀，  
};  
s3Client.uploadPartCopy(params, function (err, data) {  
    if (err) {  
        console.log("Error", err);  
    } else {  
        console.log("Success", data);  
    }  
});
```

### 5.7.3. 请求参数

参数	类型	说明	是否必要
Bucket	String	目的桶名称	是
Key	String	目的对象 key	是
CopySource	string	源对象地址 (bucket+key)	是

参数	类型	说明	是否必要
UploadId	String	与本次复制操作相应的分片上传任务 Id	是
PartNumber	String	与本次复制操作相应的分片编号	是

#### 5.7.4. 返回结果

参数	类型	说明
CopyPartResult	CopyPartResult	包含拷贝分片的 Entity Tag 和最后修改时间等信息

## 5.8. 取消分片上传任务

### 5.8.1. 功能说明

使用 `abortMultipartUpload` 取消分片上传任务。

### 5.8.2. 代码示例

```
var params = {
  Bucket: "<your-bucket-name>",
  Key: "<your-key-name>",
  // 设置 Upload ID
  UploadId: "<your-upload-id>",
};
s3Client.abortMultipartUpload(params, function (err, data) {
  if (err) {
    console.log('abortMultipartUpload err, ', err);
  } else {
    console.log('abortMultipartUpload success');
  }
});
```

### 5.8.3. 请求参数

参数	类型	说明	是否必要
Bucket	String	桶名称	是
Key	String	对象名称	是
UploadId	String	上传分片 id	是

### 5.8.4. 返回结果

根据返回码判断是否操作成功。

## 6. 安全凭证服务(STS)

STS 即 Secure Token Service 是一种安全凭证服务，可以使用 STS 来完成对于临时用户的访问授权。对于跨用户短期访问对象存储资源时，可以使用 STS 服务。这样就不需要透露主账号 AK/SK，只需要生成一个短期访问凭证给需要的用户使用即可，避免主账号 AK/SK 泄露带来的安全风险。

### 6.1. 获取临时 token

在服务端生成临时 token，可参考 java、python、nodejs SDK 说明，请从 [SDK 概览](#) 页面选择进入对应的开发指南查阅。

### 6.2. 使用临时 token

使用 `assumeRole` 请求申请临时凭证，使用服务端返回的 `accessKeyId`，`secretAccessKey` 和 `sessionToken` 来初始化 s3 对象。使用过程中，需要更新凭证，保证凭证不过期。使用过期凭证的请求，服务端会返回 403 (AccessDenied)。

```
let S3Demo = {  
  
  credentials: {  
  
    accessKeyId: "<your-access-key>",  
  
    secretAccessKey: "<your-secret-access-key>",  
  
    sessionToken: "<your-session-token>",  
  
  },  
  
  s3Client: null,  
  
  // 初始化 s3 对象  
  init: function () {  
  
    let config = {  
  
      credentials: this.credentials,  
  
      endpoint: "<your-endpoint>",  
  
    };  
  
  };  
  
};
```

```
        this.s3Client = new AWS.S3(config);  
    },  
}
```

参数	说明
credentials	用户账号信息，包含 accessKeyId 和 secretAccessKey 和 sessionToken
endpoint	天翼云资源池的地址，必须指定 http 或 https 前缀