天翼云媒体存储

Node.js SDK 使用指导书

2025-09-28

天翼云科技有限公司

天翼云媒体存储Node.js SDK使用指导书

SDK安装

开发环境

- 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，或者任何您喜欢的js开发工具。

安装SDK

方式一

在天翼云官网下载 `xos-nodejs-sdk.js`，下载地址： xos-nodejs-sdk.zip

使用xos-nodejs-sdk之前，加载dist目录里的xos-nodejs-sdk.js文件即可。

在您的项目中引入xos库：

```
// 引入xos库，使用源码安装
var s3Client = require('./dist/xos-nodejs-sdk');
```

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：

```
var AWS = require('aws-sdk');
```

初始化

使用SDK功能前，需要新建Client，代码如下：

```
let S3Demo = {
    credentials: {
        accessKeyId: "<your-access-key>",
        secretAccessKey: "<your-secret-key>",
    },
    s3Client: null,

    init: function() {
        let config = {
            credentials: this.credentials,
            endpoint: "<your-endpoint>",  // e.g. http://endpoint or
https://endpoint
            //signatureVersion: 'v4', // 预签名使用v4版本签名，默认为v2
```

```
            //httpOptions: { timeout: 120000 }, // 设置请求超时，default 120000，单位毫秒
        };
        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

桶相关接口

创建桶

功能说明

您可以使用 createBucket 接口创建桶，创建桶时可以指定桶的名称、访问权限和区域位置等。

代码示例

```
// Create the parameters for calling createBucket
var bucketParams = {
    Bucket: "<your-bucket-name>",
    ACL: 'private',
    CreateBucketConfiguration: {
        LocationConstraint: ''
    }
};

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

请求参数

参数	类型	说明	是否必要
Bucket	String	桶名称	是
ACL	String	设置桶访问权限为私有读写	否
CreateBucketConfiguration	Object	创建桶的基本配置	是
LocationConstraint	String	设置桶区域位置	是

返回结果

根据返回码判断是否操作成功。

获取桶列表

功能说明

您可以使用 listBuckets 接口获取桶列表，以下代码展示如何获取桶列表。

代码示例

```
// Call s3Client to list the buckets
s3Client.listBuckets(function (err, data) {
    if (err) {
        console.log("Error", err);
    } else {
        console.log("Success", data.Buckets);
    }
});
```

请求参数

无

返回结果

参数	类型	说明
Buckets	Array	桶列表
Owner	Object	bucket的拥有者信息

关于Buckets一些说明

参数	说明
Name	桶名
CreationDate	桶创建时间

判断桶是否存在

功能说明

您可以使用 headBucket接口判断桶是否存在，以下代码展示如何判断一个桶是否存在。

代码示例

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

请求参数

参数	类型	说明	是否必要
Bucket	String	桶名称	是

返回结果

根据返回码判断是否操作成功。

删除桶

功能说明

您可以使用 deleteBucket接口删除桶，以下代码展示如何删除桶。注意：在删除桶前，必须先确保桶为空，否则会导致BucketNotEmpty错误。

代码示例

```
// Create params for s3Client.deleteBucket
var bucketParams = {
    Bucket: "<your-bucket-name>",
};

// Call s3Client to delete the bucket
s3Client.deleteBucket(bucketParams, function (err, data) {
    if (err) {
        console.log("Error", err);
    } else {
        console.log("Success", data);
    }
});
```

请求参数

参数	类型	说明	是否必要
Bucket	String	桶名称	是

返回结果

根据返回码判断是否操作成功。

设置桶访问权限

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
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

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AccesssControlList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 obfuscated
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);
    }
});
```

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。

返回结果

根据返回码判断是否操作成功。

获取桶访问权限

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。

代码示例

```
var bucketParams = {
    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);
    }
});
```

请求参数

参数	类型	说明	是否必要
Bucket	String	桶名称	是

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

设置桶策略

功能说明

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。您可以在 putBucketPolicy 接口中使用 JSON格式的文本字符串直接指定策略。

存储桶的策略主要包含以下元素：

参数	说明
Resource	可选关键字，指定statement起作用的一组资源，支持通配符“*”，表示所有资源
Action	可选关键字，指定本条statement作用的操作，Action字段为媒体存储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	必选关键字，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	可选关键字，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>

代码示例

```
let policy = {
    Version: "2012-10-17",
    Statement: [{
        Sid: "AddPerm",
        Effect: "Allow",
        Principal: "*",
        Action: [
            "s3:GetObject"
        ],
        Resource: [
            "arn:aws:s3:::" + this.bucket + "/*"
        ]
    }]
};
let params = {
    Bucket: this.bucket,
    Policy: JSON.stringify(policy),
};
this.s3Client.putBucketPolicy(params, function(err, data) {
    if (err) {
        console.log("Error", err);
    } else {
        console.log("Success", data);
    }
});
```

请求参数

参数	类型	说明	是否必要
Bucket	String	桶名称	是
Policy	String	桶策略	是

返回结果

根据返回码判断是否操作成功。

获取桶策略

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。

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);
    }
});
```

请求参数

参数	类型	说明	是否必要
Bucket	String	桶名称	是

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字段为媒体存储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>

删除桶策略

功能说明

您可以使用 deleteBucketPolicy接口删除桶策略，以下代码展示如何删除一个桶策略。

代码示例

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

请求参数

参数	类型	说明	是否必要
Bucket	String	桶名称	是

返回结果

根据返回码判断是否操作成功。

设置桶生命周期配置

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。

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);
    }
});
```

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

关于生命周期规则Rule一些说明

参数	类型	说明	是否必要
ID	String	规则ID	否
Status	String	是否启用规则（Enabled|Disabled）	是
Expiration	Expiration	文件过期时间	否
AbortIncompleteMultipartUpload	AbortIncompleteMultipartUpload	未完成上传的分片过期时间	否
Transitions	Transition数组	文件转换到低频存储规则（距离修改时间）	否
Filter	Filter	应用范围，可以指定前缀或对象标签	否

关于Expiration的说明：

参数	类型	说明
Days	Int	过期天数

关于AbortIncompleteMultipartUpload的说明：

参数	类型	说明
DaysAfterInitiation	Int	过期天数

关于Transition的说明：

参数	类型	说明
Days	Int	转换过期天数
StorageClass	StorageClass	要转换的存储类型

关于Filter的说明：

参数	类型	说明
Prefix	string	需要过滤的前缀

返回结果

根据返回码判断是否操作成功。

获取桶生命周期配置

功能说明

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

代码示例

```
var params = {
    Bucket: "<your-bucket-name>",
};

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

请求参数

参数	类型	说明	是否必要
Bucket	String	桶名称	是

返回结果

参数	类型	说明
Rules	Rule 数组	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

关于生命周期规则Rule一些说明

参数	类型	说明
ID	String	规则ID
Status	String	是否启用规则（Enabled|Disabled）
Expiration	Expiration	文件过期时间
AbortIncompleteMultipartUpload	AbortIncompleteMultipartUpload	未完成上传的分片过期时间
Transitions	Transition数组	文件转换到低频存储规则（距离修改时间）
Filter	Filter	应用范围，可以指定前缀或对象标签

关于Expiration的说明：

参数	类型	说明
Days	Int	过期天数

关于AbortIncompleteMultipartUpload的说明：

参数	类型	说明
DaysAfterInitiation	Int	过期天数

关于Transition的说明：

参数	类型	说明
Days	Int	转换过期天数
StorageClass	StorageClass	要转换的存储类型

关于Filter的说明：

参数	类型	说明
Prefix	String	需要过滤的前缀

删除桶生命周期配置

功能说明

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

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));
  }
});
```

请求参数

参数	类型	说明	是否必要
Bucket	String	桶名称	是

返回结果

根据返回码判断是否操作成功。

设置桶跨域访问配置

功能说明

跨域资源共享 (CORS) 定义了在一个域中加载的客户端 Web 应用程序与另一个域中的资源交互的方式。利用 CORS 支持，您可以构建丰富的客户端 Web 应用程序，同时可以选择性地允许跨源访问您的资源。

您可以通过 putBucketCors接口设置桶的跨域访问设置，以下为示例代码。

代码示例

```
var corsParams = {
    Bucket: "<your-bucket-name>",
    CORSConfiguration: {
        CORSRules: [{
            AllowedHeaders: ["*"],
            AllowedMethods: ["POST", "GET", "PUT", "DELETE", "HEAD"],
            AllowedOrigins: ["*"],
            ExposeHeaders: ["ETag"],
            MaxAgeSeconds: 3600,
        }],
    },
};
s3Client.putBucketCors(corsParams, function (err, data) {
    if (err) {
        console.log("Error", err);
    } else {
        console.log("Success", data);
    }
});
```

请求参数

参数	类型	说明	是否必要
Bucket	String	桶名称	是
CORSConfiguration	Object	CORS配置对象	是

返回结果

根据返回码判断是否操作成功。

获取桶跨域访问配置

功能说明

您可以使用getBucketCors接口获取桶的跨域访问配置，以下代码展示如何获取桶的跨域访问配置。

在调用 getBucketCors 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 CORS 配置，该配置由传递到回调函数的 data 参数的 CORSRules 属性返回。如果所选存储桶没有 CORS 配置，该信息将在 error 参数中返回到回调函数。

代码示例

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

// call s3Client to retrieve CORS configuration for selected bucket
s3Client.getBucketCors(bucketParams, function (err, data) {
    if (err) {
        console.log("Error", err);
    } else if (data) {
        console.log("Success", JSON.stringify(data.CORSRules));
    }
});
```

请求参数

参数	类型	说明	是否必要
Bucket	String	桶名称	是

返回结果

参数	类型	说明
CORSRules	Array	跨域访问规则数组

关于CORSRules一些说明

参数	说明
AllowedMethods	允许的请求方法
AllowedOrigins	允许的请求源
AllowedHeaders	允许的请求头
ExposedHeaders	允许返回的Response Header
MaxAgeSeconds	跨域请求结果的缓存时间

删除桶跨域访问配置

功能说明

您可以通过 deleteBucketCors接口删除桶跨域访问设置，以下为示例代码。

在调用 deleteBucketCors 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跨域访问设置，该信息将在error参数中返回给回调函数。

代码示例

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

// call s3Client to retrieve CORS configuration for selected bucket
s3Client.deleteBucketCors(bucketParams, function (err, data) {
    if (err) {
        console.log("Error", err);
    } else if (data) {
        console.log("Success", JSON.stringify(data.CORSRules));
    }
});
```

请求参数

参数	类型	说明	是否必要
Bucket	String	桶名称	是

返回结果

根据返回码判断是否操作成功。

设置桶版本控制状态

功能说明

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。
桶的版本控制状态可以设置为以下的值：

- Enabled：对bucket中的所有对象启用版本控制，之后每个添加到bucket中的对象都会被设置一个唯一的version id。
- Suspended：关闭bucket的版本控制，之后每个添加到bucket中的对象的version ID会被设置为null。

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);
    }
});
```

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	桶版本控制开关状态	是

返回结果

根据返回码判断是否操作成功。

获取桶版本控制状态

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：

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);
    }
});
```

请求参数

参数	类型	说明	是否必要
Bucket	String	桶名称	是

返回结果

参数	类型	说明
Status	String	桶版本控制状态

对象相关接口

获取对象列表

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。

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);
});
```

如果 list 大于1000，则返回的结果中 isTruncated 为true，通过返回的 nextMarker 标记可以作为下次读取的起点。列举所有对象示例代码如下：

```
let marker = '';
const params = {
    Bucket: "<your-bucket-name>",
    MaxKeys: 1000,
    Marker: marker
};

const fetchObjects = () => {
    s3Client.listObjects(params, function (err, data) {
        if (err) {
            console.error("Error listing objects:", err);
            return;
        }

        if (data.Contents && data.Contents.length > 0) {
            data.Contents.forEach(obj => {
                console.log("Object Key:", obj.Key);
            });
        }
```

```
        // 更新 marker
        if (data.IsTruncated) {
            params.Marker = data.NextMarker || '';
            fetchObjects();
        }
    });
};

fetchObjects();
```

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

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

上传对象

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。

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，文件上传使用本地文件作为对象的数据源，指定待上传的文件File对象，以字节流的方式进行上传：

```
var params = {
    Bucket: "<your-bucket-name>",
    Key: "<your-key-name>",
    Body: fs.createReadStream('<your-file-path>'),
    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);
});
```

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

返回结果

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

下载对象

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。

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);
});
```

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

返回结果

参数	类型	说明
Body	Array	对象数据内容字节数组
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

复制对象

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。

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

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

删除对象

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。

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);
});
```

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

返回结果

根据返回码判断是否操作成功。

批量删除对象

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。

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");
    }
});
```

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

返回结果

根据返回码判断是否操作成功。

获取对象元数据

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。

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);
    }
});
```

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

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

设置对象访问权限

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。

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);
});
```

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
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。

返回结果

根据返回码判断是否操作成功。

获取对象访问权限

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。

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);
});
```

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

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

获取对象标签

功能说明

您可以使用getObjectTagging接口获取对象标签。

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]);
    }
});
```

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

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

删除对象标签

功能说明

您可以使用deleteObjectTagging接口删除对象标签。

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);
    }
});
```

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

返回结果

根据返回码判断是否操作成功。

设置对象标签

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。

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);
    }
});
```

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

返回结果

根据返回码判断是否操作成功。

生成上传对象预签名URL

功能说明

getSignedUrl接口为一个指定对象生成一个预签名的上传链接，访问该链接可以直接上传对象到指定的媒体存储存储桶。

代码示例

生成预签名上传链接

```
var params = {
    Bucket: "<your-bucket-name>",
    Key: "<your-key-name>",
    Expires: 900,
};
s3Client.getSignedUrl("putObject", params, function (err, url) {
    if (err) {
        console.log("Error", err);
    } else {
        console.log("Success: ", url);
    }
});
```

生成上传对象的预签名URL后，可以通过该URL将文件上传至媒体存储

```
putObjUsingPresignedUrl: function (presignedUrl, filePath) {
    const http = require('http');
    const fs = require('fs');
    const url = require('url');
    const parsedUrl = url.parse(presignedUrl);
    const fileData = fs.readFileSync(filePath);

    const options = {
        hostname: parsedUrl.hostname,
        port: parsedUrl.port,
        path: parsedUrl.path,
        method: 'PUT',
        headers: {
            'Content-Length': fileData.length
        }
    };

    const req = http.request(options, (res) => {
        if (res.statusCode === 200) {
            console.log("File uploaded successfully");
        } else {
            console.log(`Failed to upload. Status Code: ${res.statusCode}`);
        }
    });

    req.on('error', (e) => {
        console.error(`Problem with request: ${e.message}`);
    });

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

请求参数

参数	类型	说明	是否必要
Bucket	String	bucket的名称	是
Key	String	对象的key	是

参数	类型	说明	是否必要
Expires	Int	超时时间（秒）	否，默认900秒
ContentType	String	对象的ContentType	否。若生成预签名URL时指定了，则通过预签名链接上传时也需要指定为相同的ContentType

返回结果

生成对应的预签名上传 URL，该链接允许用户在指定的时间内直接将对象上传到媒体存储存储桶。

生成下载对象预签名URL

功能说明

getSignedUrl接口为一个指定对象生成一个预签名的下载链接，访问该链接可以直接下载该对象。

代码示例

生成预签名下载链接

```
var params = {
    Bucket: "<your-bucket-name>",
    Key: "<your-key-name>",
    Expires: 900,
};
s3Client.getSignedUrl("getObject", params, function (err, url) {
    if (err) {
        console.log("Error", err);
    } else {
        console.log("Success: ", url);
    }
});
```

生成下载对象的预签名URL后，可以通过该URL下载文件

```
getObjUsingPresignedUrl: function (presignedUrl, saveFilePath) {
    const http = require('http');
    const fs = require('fs');
    const url = require('url');
    const parsedUrl = url.parse(presignedUrl);

    const options = {
        hostname: parsedUrl.hostname,
        port: parsedUrl.port,
        path: parsedUrl.path,
        method: 'GET'
    };

    const file = fs.createWriteStream(saveFilePath);

    const req = http.request(options, (res) => {
        if (res.statusCode === 200) {
            console.log("Downloading file...");
            res.pipe(file);
```

```
            file.on('finish', () => {
                file.close();
                console.log("File downloaded successfully");
            });
        } else {
            console.log(`Failed to download. Status Code: ${res.statusCode}`);
        }
    });

    req.on('error', (e) => {
        console.error(`Problem with request: ${e.message}`);
    });

    req.end();
}
```

请求参数

参数	类型	说明	是否必要
Bucket	String	bucket的名称	是
Key	String	对象的key	是
Expires	Int	超时时间（秒）	否，默认900秒

返回结果

生成对应的预签名下载 URL，该链接允许用户在指定的时间内直接从媒体存储下载对象。

服务端加密

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。

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);
    }
```

```
});
```

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

返回结果

根据返回码判断是否操作成功。

Post上传

功能说明

createPresignedPost接口为一个指定对象生成一个支持post方式上传文件的参数集合，可以在前端使用post form-data的方式上传文件。

代码示例

```
var params = {
    Bucket: "<your-bucket-name>",
    Fields: {
        key: "<your-key-name>",
    },
    Conditions: [
        ['starts-with', '$key', key],
        //{"acl": "public-read" },                          // 设置acl
        //["starts-with", "$Content-Type", "image/jpeg"]    // 设置content-type
        //["content-length-range", 10, 1024],               // 限制上传的文件大小
    ],
    Expires: 900,
};
s3Client.createPresignedPost(params, function (err, data) {
    if (err) {
        console.error('Presigning post data encountered an error', err);
    } else {
        console.log('The post data is', data);
    }
});
```

请求参数

参数	类型	说明	是否必要
Bucket	String	bucket的名称	是
Key	String	对象的key	是
Expires	Int	超时时间（秒）	否，默认900秒

返回结果

参数	类型	说明
url	String	请求上传的url
Policy	String	服务端用于校验的policy
X-Amz-Algorithm	String	v4签名，哈希算法
X-Amz-Signature	String	v4签名，请求的参数签名
X-Amz-Date	String	v4签名，日期信息
X-Amz-Credential	String	v4签名，ak信息

前端使用方式如下：

```
<form action="<data.url>" method="POST" enctype="multipart/form-data">
    <input type="hidden" name="Policy" value="<data.fields['Policy']>" />
    <input type="hidden" name="X-Amz-Algorithm" value="<data.fields['X-Amz-
Algorithm']>" />
    <input type="hidden" name="X-Amz-Credential" value="<data.fields['X-Amz-
Credential']>" />
    <input type="hidden" name="X-Amz-Date" value="<data.fields['X-Amz-Date']>"
/>
    <input type="hidden" name="X-Amz-Signature" value="<data.fields['X-Amz-
Signature']>" />
    <input type="hidden" name="bucket" value="<data.fields['bucket']>" />
    <input type="hidden" name="key" value="<data.fields['key']>" />

    <input type="file" name="file" value="" />
    <input type="submit" value="Submit" />
</form>
```

获取多版本对象列表

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。

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);
    }
});
```

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

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
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

分片上传接口

融合接口

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。

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;
}
```

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

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

初始化分片上传任务

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。

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);
});
```

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

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。

上传分片

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。

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: fs.createReadStream('<your-file-path>'),
}
s3Client.uploadPart(partParams, function (err, result) {
    if(err){
        console.log('uploadPart Error ' + err);
    }else{
        console.log('uploadPart ETag ' + result.ETag);
    }
});
```

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

返回结果

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

合并分片

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。

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');
```

```
    }
});
```

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

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

列举分片上传任务

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。

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);
    }
});
```

请求参数

参数	类型	说明	是否必要
Bucket	String	桶名称	是

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命信息，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
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

列举已上传的分片

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。

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);
    }
});
```

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

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

复制分片

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]。

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);
    }
});
```

请求参数

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

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

取消分片上传任务

功能说明

使用abortMultipartUpload取消分片上传任务。

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');
    }
});
```

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

返回结果

根据返回码判断是否操作成功。

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。

初始化STS服务

```
var config = {
    accessKeyId: "<your-access-key>",
    secretAccessKey: "<your-secret-access-key>",
    endpoint: "<your-endpoint>",
    region: "ctyun",// region固定填ctyun
};
var stsClient = new AWS.STS(config);
```

获取临时token

```
var params = {
    Policy: `{"Version":"2012-10-17","Statement":{"Effect":"Allow","Action":
["s3:*"],"Resource":["arn:aws:s3:::<your-bucket>","arn:aws:s3:::<your-
bucket>/*"]}}`,
    RoleArn: "arn:aws:iam:::role/<your-role>",
    RoleSessionName: "<your-role-session-name>",
    DurationSeconds: 900, // 过期时间
}
stsClient.assumeRole(params, function (err, data) {
    if (err) {
        console.log("Error", err);
    } else {
        console.log("Success", data);
    }
});
```

Policy设置例子

允许所有的操作

```
{"Version":"2012-10-17","Statement":{"Effect":"Allow","Action":
["s3:*"],"Resource":["arn:aws:s3:::<your-bucket-name>","arn:aws:s3:::<your-
bucket-name>/*"]}}
```

限制只能上传和下载

```
{"Version":"2012-10-17","Statement":{"Effect":"Allow","Action":
["s3:PutObject","s3:GetObject"],"Resource":["arn:aws:s3:::<your-bucket-
name>","arn:aws:s3:::<your-bucket-name>/*"]}}
```

使用分片上传

```
{"Version":"2012-10-17","Statement":{"Effect":"Allow","Action":
["s3:PutObject","s3:AbortMultipartUpload","s3:ListBucketMultipartUploads","s3:Li
stMultipartUploadParts"],"Resource":["arn:aws:s3:::<your-bucket-
name>","arn:aws:s3:::<your-bucket-name>/*"]}}
```

其他操作权限

上传权限：s3:PutObject
下载权限：s3:GetObject
删除权限：s3:DeleteObject
获取列表权限：s3:ListBucket

注意：

1.ListObjects 操作是由ListBucket权限控制的
2."Version:2012-10-17"是系统的policy格式的版本号，不能改成其他日期

更多操作权限可以参考：

https://www.ctyun.cn/document/10306929/10136179

参数	类型	描述	是否必要
RoleArn	String	角色的ARN，在控制台创建角色后可以查看	是
Policy	String	角色的policy，需要是json格式，限制长度1~2048	是
RoleSessionName	String	角色会话名称，此字段为用户自定义，限制长度2~64	是
DurationSeconds	Integer	会话有效期时间，默认为3600s，范围15分钟至12小时	否

使用临时token

```
let S3Demo = {
    credentials: {
        accessKeyId: "<your-access-key>",
        secretAccessKey: "<your-secret-access-key>",
        sessionToken: "<your-session-token>",
    },
    s3Client: null,

    // 初始化s3Client
    init: function () {
        let config = {
            credentials: this.credentials,
            endpoint: "<your-endpoint>",
        };
        this.s3Client = new AWS.S3(config);
    },
}
```