ZOS对象存储Nodejs_SDK使用手册

环境配置

下载与安装

1. 相关资源和环境依赖
    - SDK 压缩包快速下载地址:见帮助中心
    - Nodejs版本依赖：>= 18.0

SDK安装

- **安装 AWS SDK：** 复制SDK文件在项目文件夹处，使用 npm或者yarn安装nodejs-sdk，在终端中运行以下命令：

  Npm

  ```
  npm install ./aws-client-s3-v3.886.0.tgz
  npm install ./aws-s3-presigned-post-v3.886.0.tgz
  npm install ./aws-s3-request-presigner-v3.886.0.tgz
  ```

  Yarn

  ```
  yarn add ./aws-client-s3-v3.886.0.tgz
  yarn add ./aws-s3-presigned-post-v3.886.0.tgz
  yarn add ./aws-s3-request-presigner-v3.886.0.tgz
  ```

连接ZOS

首先，在控制台获取AccessKey、SecretKey以及外网访问的endpoint。

- 创建connect_zos.js，连接后端ZOS，示例如下：
- 在connect_zos.js中执行具体操作，示例如下：

```
// 使用ES
import { S3Client} from "@aws-sdk/client-s3";
// 配置 ZOS
const zos_client = new S3Client({
    region: "us-east-1",//填写region标识，固定为us-east-1
    endpoint: "https://jiangsu-10.zos.ctyun.cn", // 填写zos地域下的endpoint
    credentials: {
        accessKeyId: "", // 填写你的ZOS账号的AccessKeyID
        secretAccessKey: "", // 填写你的ZOS账号的SecretKeyID
    },
    // logger: console, //打印日志
});
```

代码执行方式

```
//基于示例新建js文件，如Create Bucket-->create_bucket.js
//create_bucket.js中
import { zos_client } from './connect_zos.js';
```

执行方式：

```
node create_bucket.js
```

全局错误码定义

请求可能会返回相关错误，具体错误码编号及信息请参考下表。同一个错误码可能对应不同的错误码描述，具体由接口来决定。

错误码	错误码描述
100	Continue
200	Success
201	Created
202	Accepted
204	NoContent
206	Partial content
304	NotModified
400	InvalidArgument
400	InvalidDigest
400	BadDigest
400	InvalidBucketName
400	InvalidObjectName
400	UnresolvableGrantByEmailAddress
400	InvalidPart
400	InvalidPartOrder
400	RequestTimeout
400	EntityTooLarge
403	AccessDenied
403	UserSuspended

错误码	错误码描述
403	RequestTimeTooSkewed
404	NoSuchKey
404	NoSuchBucket
404	NoSuchUpload
405	MethodNotAllowed
408	RequestTimeout
409	BucketAlreadyExists
409	BucketNotEmpty
411	MissingContentLength
412	PreconditionFailed
416	InvalidRange
422	UnprocessableEntity
500	InternalError

Bucket操作

Create Bucket

功能说明

Create Bucket 请求可以在指定账号下创建一个新的Bucket。单个用户在每个资源池内默认创建的桶最大数量为100个桶。

注意: 当前通过SDK创建的桶，默认在控制台上无法上传文件; 需要在跨域复制中创建跨域规则，配置之后才可以上传。

方法原型

```
const input = {
    ACL: "private" || "public-read" || "public-read-write" || "authenticated-read",
    Bucket: "STRING_VALUE", // required,
    ObjectLockEnabledForBucket: true || false
}
const command = new CreateBucketCommand(input);
const response = await client.send(command);
```

参数说明

参数名称	参数描述	类型	是否必须
ACL	Bucket的ACL配置，配置参考Put Bucket ACL 可选参数 private, public-read, public-read-write, authenticated-read	String	否
Bucket	Bucket的名称。长度范围为3到63个字符，支持小写字母、数字、中划线（-）。禁止以中划线（-）开头或结尾。	String	是
ObjectLockEnabledForBucket	开启关闭新创建Bucket的Object锁定功能，具体锁定规则参考Put Bucket Object Lock Configuration	Boolean	否
StorageClass	存储类型，可选参数 STANDARD, STANDARD_IA, GLACIER	String	否

返回结果及说明

```
{
  "'$metadata'": {
    "httpStatusCode": 200,
    "requestId": 'tx00000000000000249b20e-0068ca6f87-9f683575-js10',
    "extendedRequestId": undefined,
    "cfId": undefined,
    "attempts": 1,
    "totalRetryDelay": 0
  }
}
```

返回结果	描述	类型
metadata	http请求返回数据	Dict

示例

```
import { CreateBucketCommand } from "@aws-sdk/client-s3";
/**
 * 创建一个新的 S3 存储桶
 * @param {string} bucketName - 要创建的存储桶的名称
 */
const createBucket = async (bucketName) => {
    try {
        const command = new CreateBucketCommand({
            Bucket: bucketName
        });
        const response = await zos_client.send(command);
        console.log(`Success! Bucket "${bucketName}" created successfully.`);
    } catch (error) {
```

```
        console.error("An error occurred:", error.name);
        if (error.name === 'BucketAlreadyOwnedByYou') {
            console.error(`Error: Bucket "${bucketName}" already exists and is owned by
you.`);
        } else if (error.name === 'BucketAlreadyExists') {
            console.error(`Error: Bucket name "${bucketName}" is already taken by another
account.`);
        } else if (error.name === 'InvalidBucketName') {
            console.error(`Error: The bucket name "${bucketName}" is invalid.`);
        } else {
            console.error("Error message:", error.message);
        }
    }
};
createBucket("bucket_to_create");
```

Delete Bucket

功能说明

Delete Bucket 请求可以在指定账号下删除 Bucket，删除之前要求Bucket为空且不存在碎片。

方法原型

```
const input = {
    Bucket: "STRING_VALUE", // required,
};
const command = new DeleteBucketCommand(input);
const response = await client.send(command);
```

参数说明

参数名称	参数描述	类型	是否必须
Bucket	需要删除的Bucket名称	String	是

返回结果说明

```
{
  "'$metadata'": {
    "httpStatusCode": 200,
    "requestId": 'tx0000000000000249b20e-0068ca6f87-9f683575-js10',
    "extendedRequestId": undefined,
    "cfId": undefined,
    "attempts": 1,
    "totalRetryDelay": 0
  }
}
```

返回结果	描述	类型
metadata	http请求返回数据	Dict

示例

```
import { DeleteBucketCommand } from "@aws-sdk/client-s3";
/**
 * 删除指定的空存储桶
 * @param {string} bucketName - 桶的名称
 */
const deleteBucket = async (bucketName) => {
    try {
        const command = new DeleteBucketCommand({
            Bucket: bucketName
        });
        await zos_client.send(command);
        console.log(`Success! Bucket "${bucketName}" has been deleted.`);
    } catch (error) {
        console.error("An error occurred:", error.name);
        if (error.name === 'NoSuchBucket') {
            console.error(`Error: Bucket "${bucketName}" does not exist.`);
        } else {
            console.error("Error message:", error.message);
        }
    }
};
deleteBucket("bucket-to-delete");
```

Head Bucket

功能说明

Head Bucket 请求可以判断某个Bucket是否存在，或者是否有权限访问该Bucket(其他用户名下Bucket)。

方法原型

```
const input = {
    Bucket: "STRING_VALUE", // required,
}
const command = new HeadBucketCommand(input);
const response = await client.send(command);
```

参数说明

参数名称	参数描述	类型	是否必须
Bucket	Bucket的名称	String	是

返回结果说明

```
{
  "'$metadata'": {
    "httpStatusCode": 200,
    "requestId": 'tx0000000000000000249b20e-0068ca6f87-9f683575-js10',
    "extendedRequestId": undefined,
    "cfId": undefined,
    "attempts": 1,
    "totalRetryDelay": 0
  }
}
```

返回结果	描述	类型
metadata	http请求返回数据	Dict
HTTPStatusCode	http请求状态码	Int

示例

```
import { HeadBucketCommand } from "@aws-sdk/client-s3";
/**
 * 检查指定的S3存储桶是否存在且可访问
 * @param {string} bucketName - 要检查的存储桶的名称
 */
const headBucket = async (bucketName) => {
    try {
        const command = new HeadBucketCommand({
            Bucket: bucketName, // 指定要检查的存储桶名称
        });
        const data = await zos_client.send(command);

        // 如果命令成功执行没有抛出错误，说明存储桶存在
        // 成功的响应元数据中 httpStatusCode 会是 200
        console.log(`Success! Bucket "${bucketName}" exists and is accessible.`);
        console.log("Response Status Code:", data.$metadata.httpStatusCode);
    } catch (error) {
        // 错误处理是 headBucket 的关键部分
        console.error("An error occurred:", error.name);

        // 常见错误判断
        if (error.name === 'NotFound') {
            console.error(`Result: Bucket "${bucketName}" does not exist.`);
        } else if (error.$metadata?.httpStatusCode === 403) {
            // 当存储桶存在但你没有权限时，通常会返回 403 Forbidden
            console.error(`Result: Access denied. You may not have permission for bucket
"${bucketName}".`);
        } else {
            console.error("Error message:", error.message);
        }
    }
```

```
};
headBucket("your-bucket-name");
```

List Bucket

功能说明

List Bucket 可以列出请求用户拥有的所有的bucket列表。

方法原型

```
const command = new ListBucketsCommand(input);
const response = await client.send(command);
```

参数说明

无

返回结果说明


```
{
   "Owner": {
      "DisplayName": "用户名",
      "ID": "uid"
   },
   "Buckets": [
      {
         "Name": 'bucket-d464',
         "CreationDate": "2025-07-29T16:41:31.789Z"
      },
      {
         "Name": 'bucket-d633',
         "CreationDate": "2024-07-18T07:48:39.635Z"
      },
      {
         "Name": 'bucket-f468',
         "CreationDate": "2025-01-17T01:17:34.204Z"
      },
   ]
  "'$metadata'": {
      "httpStatusCode": 200,
      "requestId": 'tx0000000000000249b20e-0068ca6f87-9f683575-js10',
      "extendedRequestId": undefined,
      "cfId": undefined,
      "attempts": 1,
      "totalRetryDelay": 0
   }
}
```

返回结果	描述	类型
metadate	http请求返回数据	dict
Owner	列出的bucket的属主	dict
DisplayName	所有者用户名	str
ID	所有者用户ID	str
Buckets	bucket的列表	list
Name	桶名称	str

示例

```
import { ListBucketsCommand } from "@aws-sdk/client-s3";
const listBuckets = async () => {
    try {
        const command = new ListBucketsCommand({});
        const data = await zos_client.send(command);
        console.log("data:", data)
        console.log("Successfully connected. Buckets:", data.Buckets);
    } catch (error) {
        console.error("An error occurred:", error.name);
        console.error("Error message:", error.message);
    }
};
listBuckets();
```

List Bucket Object

功能说明

List Bucket Object请求可以列出该 Bucket 下部分或者所有Object，发起该请求需要拥有 Read 权限。

方法原型

```
const input = { // ListObjectsV2Request
 Bucket: "STRING_VALUE", // required
 Delimiter: "STRING_VALUE",
 MaxKeys: int,
 Prefix: "STRING_VALUE",
 ContinuationToken: "STRING_VALUE",
 ],
};
const command = new ListObjectsV2Command(input);
const response = await client.send(command);
```

参数说明

参数名称	参数描述	类型	是否必须
Bucket	Bucket的名称	String	是
Delimiter	对文件名称进行分组的字符	String	否
ContinuationToken	继续列出桶的起始token	String	否
MaxKeys	一次返回keys的最大数目（默认值和上限为1000）	Integer	否
Prefix	设置返回的key的前缀	String	否

返回结果说明


```
{
  "$metadata": {
    "httpStatusCode": 200,
    "requestId": "tx000000000000000024962ae-0068ca7afd-9f6712ff-js10",
    "extendedRequestId": null,
    "cfId": null,
    "attempts": 1,
    "totalRetryDelay": 0
  },
  "Contents": [
    {
      "Key": "1.txt",
      "LastModified": "2025-07-27T08:05:37.370Z",
      "ETag": "59adb24ef3cdbe0297f05b395827453f-1",
      "Size": 0,
      "StorageClass": "STANDARD"
    }
  ],
  "IsTruncated": false,
  "KeyCount": 1,
  "MaxKeys": 100,
  "Name": "bucket-9356",
  "Prefix": "1"
}
```


返回结果	描述	类型
metadate	http请求返回数据	dict
Name	Bucket 的名称	String
IsTruncated	返回结果是否截断	Bool
Contents	返回对象的列表	List
KeyCount	返回对象数量大小	Int

返回结果	描述	类型
Prefix	请求对象前缀	String
MaxKeys	请求的单次列出桶数量	Int

示例

```
import { ListObjectsV2Command } from "@aws-sdk/client-s3";
/**
 * 列出指定S3存储桶中的对象 (第一页)
 * @param {string} bucketName - 存储桶的名称
 * @param {string} [prefix] - (可选) 只列出以此前缀开头的对象，可用于模拟文件夹
 */
const listObjects = async (bucketName, prefix) => {
    try {
        const command = new ListObjectsV2Command({
            Bucket: bucketName,
            // (可选) MaxKeys 指定单次请求最多返回的对象数量，默认1000
            MaxKeys: 100,
            // (可选) Prefix 用于筛选特定“文件夹”下的对象
            Prefix: prefix,
        });

        const data = await zos_client.send(command);
        console.log("data:", data)
        if (data.Contents) {
            console.log(`Objects in "${bucketName}":`);
            // data.Contents 是一个包含对象信息的数组
            data.Contents.forEach(obj => {
                console.log(`  - Key: ${obj.Key}, Size: ${obj.Size} bytes`);
            });
            console.log("\nTotal objects returned in this request:", data.KeyCount);

            // 如果返回的对象被截断（即桶里还有更多对象），IsTruncated会为true
            if (data.IsTruncated) {
                console.log("\nNote: There are more objects in the bucket. This is just the
first page.");
            }
        } else {
            console.log(`Bucket "${bucketName}" is empty or no objects match the prefix.`);
        }

    } catch (error) {
        console.error("An error occurred:", error.name);
        // 如果存储桶不存在，会抛出 NoSuchBucket 错误
        if (error.name === 'NoSuchBucket') {
            console.error(`Error: Bucket "${bucketName}" does not exist.`);
        } else {
            console.error("Error message:", error.message);
        }
    }
```

```
};
listObjects("my-unique-bucket", "images/");
```

Put Bucket Policy

功能说明

Put Bucket Policy请求用于为ZOS S3 bucket设置桶策略。

方法原型

```
const input = { // PutBucketPolicyRequest
  Bucket: "STRING_VALUE", // required
  Policy: "STRING_VALUE", // required, json str
};
const command = new PutBucketPolicyCommand(input);
const response = await client.send(command);
```

参数说明

参数名称	参数描述	类型	是否必须
Bucket	Bucket的名称	String	是
Policy	设置在bucket上的策略	String	是

Bucket Policy各字段描述如下：

字段	描述	类型	是否必须
Version	保持与Amazon S3一致，当前支持"2012-10-17"和"2006-03-01"	string	否
Id	桶策略ID，桶策略的唯一标识	string	否
Statement	桶策略描述，定义完整的权限控制。每条桶策略的Statement可由多条描述组成，每条描述是一个dict，每条描述可包含以下字段： Sid Effect Principal Action ReSource Condition	list	是
Sid	本条桶策略描述的ID	string	否
Effect	桶策略的效果，即指定本条桶策略描述的权限是接受请求还是拒绝请求。 接受请求：配置为“Allow”， 拒绝请求：配置为“Deny”	string	是
Principal	被授权人，即指定本条桶策略描述所作用的用户，支持通配符“*”，表示所有用户。当对某个user进行授权时，Principal格式为"AWS": "arn:aws:s3:::user/userId"	dict	否
Action	操作，即指定本条桶策略描述所作用的ZOS操作。以列表形式表示，可配置多条操作，以逗号间隔。支持通配符“*”，表示该资源能进行的所有操作。常用的Action有"s3:GetObject"，"s3:GetObjectAcl"，"s3:PutObject"，"s3:PutObjectAcl"等	list	否
Condition	条件语句，指定本条桶策略所限制的条件。可以通过Condition对ZOS资源设置防盗链，形如： "Condition": {"StringEquals":{"aws:Referer": ["`www.ctyun.cn`"]}，此时如果Effect为“Allow”，则允许来自"`www.ctyun.cn`"的请求；如果为“Deny”，则拒绝。	dict	否

字段	描述	类型	是否必须
Resource	指定策略作用的一组资源, 支持用通配 符"*"表示所有资源	string	可选, Resource 与 NotResource 选其一
NotResource	指定策略不作用的一组资源, 策略将作用于 除此之外的其他资源, 取值同 Resource	string	可选, Resource 与 NotResource 选其一

返回结果说明

```
{
  "'$metadata'": {
    "httpStatusCode": 200,
    "requestId": 'tx00000000000000249b20e-0068ca6f87-9f683575-js10',
    "extendedRequestId": undefined,
    "cfId": undefined,
    "attempts": 1,
    "totalRetryDelay": 0
  }
}
```

返回结果	描述	类型
ResponseMetadata	http请求返回数据	Dict
HTTPStatusCode	http请求状态码	Int

示例

```
import { PutBucketPolicyCommand } from "@aws-sdk/client-s3"
/**
 * 为指定的存储桶附加一个资源策略
 * @param {string} bucketName - 桶的名称
 * @param {string} policy - JSON 格式的策略字符串
 */
const putBucketPolicy = async (bucketName, policy) => {
    try {
        const command = new PutBucketPolicyCommand({
            Bucket: bucketName,
            Policy: policy, // The policy must be a JSON string
        });

        await zos_client.send(command);
        console.log(`Success! Policy has been applied to bucket "${bucketName}".`);

    } catch (error) {
        console.error("An error occurred:", error.name);
        if (error.name === 'NoSuchBucket') {
            console.error(`Error: Bucket "${bucketName}" does not exist.`);
        } else if (error.name === 'MalformedPolicy') {
```

```
            console.error("Error: The provided policy is not valid JSON or is structured
incorrectly.");
        } else {
            console.error("Error message:", error.message);
        }
    }
};
// 设置桶策略
let myBucket = "your-public-read-bucket"
//    这是一个常见的示例：授予对桶中所有对象的公共读取权限。
const publicReadPolicy = {
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "PublicReadGetObject",
            "Effect": "Allow",
            "Principal": "*",
            "Action": ["s3:GetObject"],
            "Resource": `arn:aws:s3:::${myBucket}/*` // 注意 `/*` 表示桶内的所有对象
        }
    ]
};
// // 3. 将策略对象转换为 JSON 字符串。这是必需的步骤。
const policyString = JSON.stringify(publicReadPolicy);
// // 4. 调用函数来应用策略
putBucketPolicy(myBucket, policyString);
```

Get Bucket Policy

功能说明

Get Bucket Policy请求为获取设置在一个bucket上的策略

方法原型

```
const input = { // GetBucketPolicyRequest
  Bucket: "STRING_VALUE", // required
};
const command = new GetBucketPolicyCommand(input);
const response = await client.send(command);
```

参数说明

参数名称	参数描述	类型	是否必须
Bucket	Bucket的名称	String	是

返回结果说明


```
{
  "'$metadata'": {
    "httpStatusCode": 200,
    "requestId": 'tx0000000000000000249b20e-0068ca6f87-9f683575-js10',
    "extendedRequestId": undefined,
    "cfId": undefined,
    "attempts": 1,
    "totalRetryDelay": 0
  }
    "Policy": "{"Version":"2012-10-17","Statement":
[{"Sid":"PublicReadGetObject","Effect":"Allow","Principal":"*","Action":
["s3:GetObject"],"Resource":"arn:aws:s3:::bucket-9356/*"}]}""
}
```


返回结果	描述	类型
metadata	http请求返回数据	Dict
Policy	返回bucket的policy，解析后为json串	str

示例

```
import { GetBucketPolicyCommand } from "@aws-sdk/client-s3";
/**
 * 获取指定存储桶的资源策略
 * @param {string} bucketName - 桶的名称
 */
const getBucketPolicy = async (bucketName) => {
    try {
        const command = new GetBucketPolicyCommand({
            Bucket: bucketName,
        });

        const response = await zos_client.send(command);
        console.log(`Success! Policy for bucket "${bucketName}" retrieved.`);

        // The policy is returned as a JSON string. Parse and print it for readability.
        if (response.Policy) {
            const policyObject = JSON.parse(response.Policy);
            console.log("Policy JSON:");
            console.log(JSON.stringify(policyObject, null, 2)); // Pretty-print the JSON
        }

    } catch (error) {
        console.error("An error occurred:", error.name);
        // This specific error means the bucket exists but has no policy attached.
        if (error.name === 'NoSuchBucketPolicy') {
            console.log(`Bucket "${bucketName}" does not have a policy.`);
        } else if (error.name === 'NoSuchBucket') {
            console.error(`Error: Bucket "${bucketName}" does not exist.`);
```

```
        } else {
            console.error("Error message:", error.message);
        }
    }
};
getBucketPolicy("your-bucket-with-a-policy");
```

Delete Bucket Policy

功能说明

Delete Bucket Policy请求为删除设置在某个bucket上的策略

方法原型

```
const input = { // DeleteBucketPolicyRequest
  Bucket: "STRING_VALUE" // required
};
const command = new DeleteBucketPolicyCommand(input);
const response = await client.send(command);
```

参数说明

参数名称	参数描述	类型	是否必须
Bucket	Bucket的名称	String	是

返回结果说明

```
{
  "'$metadata'": {
    "httpStatusCode": 200,
    "requestId": 'tx00000000000000249b20e-0068ca6f87-9f683575-js10',
    "extendedRequestId": undefined,
    "cfId": undefined,
    "attempts": 1,
    "totalRetryDelay": 0
  }
}
```

返回结果	描述	类型
metadata	http请求返回数据	Dict

示例

```
import { DeleteBucketPolicyCommand } from "@aws-sdk/client-s3";
/**
 * 删除指定存储桶的资源策略
 * @param {string} bucketName - 桶的名称
```

```
 */
const deleteBucketPolicy = async (bucketName) => {
    try {
        const command = new DeleteBucketPolicyCommand({
            Bucket: bucketName,
        });

        await zos_client.send(command);
        console.log(`Success! Policy has been deleted from bucket "${bucketName}".`);

    } catch (error) {
        console.error("An error occurred:", error.name);
        if (error.name === 'NoSuchBucket') {
            console.error(`Error: Bucket "${bucketName}" does not exist.`);
        } else {
            console.error("Error message:", error.message);
        }
    }
};
deleteBucketPolicy("your-bucket-with-a-policy-to-delete");
```

Put Bucket ACL

功能说明

设置Bucket的ACL，控制对Bucket的访问权限。该操作需要用户具有WRITE_ACP权限。
有三种方式设置ACL，三种方式不可同时使用，每次只能给一种参数赋值。其中，通过ACL参数方式进行操作，是设置预定义的固定的ACL，不能针对特定用户进行授权，且该参数实现的效果，也可以借由另外两种方式实现，该参数使用请求头进行传递；AccessControlPolicy参数方式和Grant参数方式则可以针对特定用户进行授权，AccessControlPolicy方式通过请求体传递，而Grant方式通过请求头传递。三种方式都会覆盖原有ACL属性，包括桶所有者自身的权限，如需保留原有ACL属性，应将需要保留的原ACL添加到本次操作的授权中（ACL参数方式会默认将桶所有者权限设为FULL_CONTROL，而另外两种方式则不会保留任何原ACL属性）。

方法原型

参数说明

参数名称	参数描述	类型	是否必须
Bucket	Bucket的名称	String	是
ACL	预定义的固定ACL，private, public-read, public-read-write, authenticated-read />	String	ACL参数方式则必须，其他两种方式，则不能使用
AccessControlPolicy	包含多个授权列表和一个桶所有者参数	dict	该方式下必须，其他两种方式下则不能使用
Grants	授权列表	list	该方式下必须
Grantee	被授权用户	dict	该方式下必须
Type	被授权用户类型， 取值范围 'CanonicalUser' 'AmazonCustomerByEmail"Group'	String	该方式下必须
ID(Grantee)	被授权用户ID	String	Type为'CanonicalUser'，则该字段必须
EmailAddress	被授权用户邮箱（注: 当前在集群互联的资源池，emailAddress方式不支持，不推荐使用emailAddress方式）	String	如果Type为'AmazonCustomerByEmail'，则该字段必须

参数名称	参数描述	类型	是否必须
URI	被授权组URI 取值范围为 所有用户：`http://acs.amazonaws.com/groups/global/AllUsers` 所有认证用户：`http://acs.amazonaws.com/groups/global/AuthenticatedUsers`	String	如果Type为'Group'，则该字段必须
Permission	向被授权用户授予的权限， 取值范围 'FULL_CONTROL' 'WRITE''WRITE_ACP' 'READ''READ_ACP'	String	该方式下必须
Owner	Bucket所有者	dict	该方式下必须
ID(Owner)	Bucket所有者ID	String	该方式下必须
GrantFullControl	被授权用户可以对桶进行read, write, read ACP, and write ACP操作 以下Grant*参数，格式都是"id=xxxx"或"emailAddress=xxxx"或者"uri=xxxx"以及他们的组合（用逗号连接）（注: 当前在集群互联的资源池，emailAddress方式不支持，不推荐使用emailAddress方式）	String	否
GrantRead	被授权用户可以对桶进行读操作，即list object	String	否
GrantWrite	被授权用户可以对桶进行写操作，创建新的对象，删除或覆盖写属于自己的对象	String	否
GrantReadACP	被授权用户可以读取桶的ACL	String	否
GrantWriteACP	被授权用户可以修改桶的ACL	String	否

返回结果说明

```
{
  "'$metadata'": {
    "httpStatusCode": 200,
    "requestId": 'tx00000000000000249b20e-0068ca6f87-9f683575-js10',
    "extendedRequestId": undefined,
    "cfId": undefined,
    "attempts": 1,
    "totalRetryDelay": 0
  }
}
```

返回结果	描述	类型
metadata	http请求返回数据	Dict

示例

```
import { PutBucketAclCommand } from "@aws-sdk/client-s3";
/**
 * 设置或替换指定存储桶的访问控制列表 (ACL)
 * @param {string} bucketName - 桶的名称
 * @param {Object} accessControlPolicy - 访问控制策略对象
 */
const putBucketAcl = async (bucketName, accessControlPolicy) => {
    try {
        const command = new PutBucketAclCommand({
            Bucket: bucketName,
            AccessControlPolicy: accessControlPolicy,
        });
```

```
        await zos_client.send(command);
        console.log(`Success! ACL has been set for bucket "${bucketName}".`);

    } catch (error){
        console.error("An error occurred:", error.name);
        if (error.name === 'NoSuchBucket') {
            console.error(`Error: Bucket "${bucketName}" does not exist.`);
        } else {
            console.error("Error message:", error.message);
        }
    }
};
// 1. 定义 ACL 对象。
//    您必须指定所有者 (Owner) 和授权列表 (Grants)。
//    这是一个示例，将完全控制权授予存储桶所有者，并授予另一个 AWS 账户读取权限。
const myAcl = {
    Owner: {
        ID: "YOUR_CANONICAL_USER_ID",
        DisplayName: "your-display-name"
    },
    Grants: [
        {
            Grantee: {
                ID: "YOUR_CANONICAL_USER_ID", // 再次指定所有者
                Type: "CanonicalUser",
            },
            Permission: "FULL_CONTROL",
        },
        {
            Grantee: {
                // 这是您希望授予权限的另一个 AWS 账户的 Canonical ID
                ID: "OTHER_ACCOUNT_CANONICAL_ID",
                Type: "CanonicalUser",
            },
            Permission: "READ",
        },
    ],
};
// 2. 调用函数来应用 ACL
putBucketAcl("your-bucket-name", myAcl);
```

Get Bucket ACL

功能说明

Get Bucket ACL 接口用来获取 Bucket 的 ACL， 即存储桶（Bucket）的访问权限控制列表。该操作需要READ_ACP权限。该功能返回的结果与Put Bucket ACL参数一致，但是需要注意的是，如果以邮箱类型授权，返回结果中将会以对应被授权用户ID形式出现，即Type不会是AmazonCustomerByEmail，而是CanonicalUser。

方法原型

```
const input = { // GetBucketAclRequest
  Bucket: "STRING_VALUE" // required
};
const command = new GetBucketAclCommand(input);
const response = await client.send(command);
```

参数名称	参数描述	类型	是否必须
Bucket	Bucket的名称	String	是

返回结果说明


```
{
   "Owner": {
      "DisplayName": "test-1",
      "ID": "test-1"
   },
   "Grants": [
      {
         "Grantee": {
            "Type": "Group",
            "URI": "http://acs.amazonaws.com/groups/global/AllUsers"
         },
         "Permission": "READ"
      },
      {
         "Grantee": {
            "EmailAddress": "",
            "Type": "CanonicalUser",
            "DisplayName": "test-1",
            "ID": "test-1"
         },
         "Permission": "FULL_CONTROL"
      }
   ],
   "'$metadata'": {
      "httpStatusCode": 200,
      "requestId": "tx00000000000000249b20e-0068ca6f87-9f683575-js10",
      "extendedRequestId": undefined,
      "cfId": undefined,
      "attempts": 1,
      "totalRetryDelay": 0
   }
}
```


返回结果	描述	类型
Owner	Bucket所有者	Dict

返回结果	描述	类型
DisplayName(Owner)	Bucket所有者的展示名	String
ID(Owner)	Bucket所有者的用户ID	String
Grants	授权列表	list
Grantee	被授权用户	dict
Type	被授权用户类型	string
URI	被授权组URI	string
ID(Grantee)	被授权用户ID	string
DisplayName(Grantee)	被授权用户展示名	string
EmailAddress	被授权用户邮箱	string
Permission	被授权权限	string

示例

```
import {GetBucketAclCommand } from "@aws-sdk/client-s3"
/**
 * 获取指定存储桶的访问控制列表 (ACL)
 * @param {string} bucketName - 桶的名称
 */
const getBucketAcl = async (bucketName) => {
    try {
        const command = new GetBucketAclCommand({
            Bucket: bucketName
        });
        const response = await zos_client.send(command);
        console.log(`Success! ACL for bucket "${bucketName}" retrieved.`);
        // 打印存储桶所有者信息
        console.log("Owner:", response.Owner.DisplayName);

        // 遍历并打印授权列表
        console.log("Grants:");
        if (response.Grants && response.Grants.length > 0) {
            response.Grants.forEach(grant => {
                console.log(
                    `  - Grantee: ${grant.Grantee.DisplayName || grant.Grantee.ID ||
grant.Grantee.URI} (${grant.Grantee.Type}), Permission: ${grant.Permission}`
                );
            });
        } else {
            console.log("  No grants found.");
        }

    } catch (error) {
        console.error("An error occurred:", error.name);
```

```
        if (error.name === 'NoSuchBucket') {
            console.error(`Error: Bucket "${bucketName}" does not exist.`);
        } else {
            console.error("Error message:", error.message);
        }
    }
};
getBucketAcl("your-bucket-name");
```

Put Bucket Tagging

功能说明

为指定的Bucket设置标签。一个Bucket最多设置50个标签。该操作需要s3:PutBucketTagging权限，桶的所有者默认拥有该权限。该操作会覆盖原有标签。

方法原型

```
const input = { // PutBucketTaggingRequest
  Bucket: "STRING_VALUE", // required
  Tagging: { // Tagging
    TagSet: [ // TagSet // required
      { // Tag
        Key: "STRING_VALUE", // required
        Value: "STRING_VALUE", // required
      },
    ],
  }
};
const command = new PutBucketTaggingCommand(input);
const response = await client.send(command);
```

参数说明

参数名称	参数描述	类型	是否必须
Bucket	Bucket的名称	String	是
Tagging	标签集容器，内部为标签列表，最多50个标签，每个标签都是键值对，key最大128字节，value最大256字节，value可以为空，key和value均为utf-8编码	dict	是

返回结果说明


```
{
  "'$metadata'": {
      "httpStatusCode": 200,
      "requestId": "tx00000000000000249b20e-0068ca6f87-9f683575-js10",
      "extendedRequestId": undefined,
      "cfId": undefined,
      "attempts": 1,
      "totalRetryDelay": 0
   }
}
```


示例

```
import { PutBucketTaggingCommand } from "@aws-sdk/client-s3";
/**
 * 设置或替换指定存储桶的标签集
 * @param {string} bucketName - 桶的名称
 * @param {Array<Object>} tags - 标签对象的数组, e.g., [{ Key: 'project', Value: 'blue' }]
 */
const putBucketTagging = async (bucketName, tags) => {
    try {
        const command = new PutBucketTaggingCommand({
            Bucket: bucketName,
            Tagging: {
                TagSet: tags,
            },
        });

        await zos_client.send(command);
        console.log(`Success! Tags have been set for bucket "${bucketName}".`);

    } catch (error) {
        console.error("An error occurred:", error.name);
        if (error.name === 'NoSuchBucket') {
            console.error(`Error: Bucket "${bucketName}" does not exist.`);
        } else {
            console.error("Error message:", error.message);
        }
    }
};

// --- Usage Example ---

// 1. 定义您想要应用的标签数组
const myTags = [
    {
        Key: "Project",
        Value: "Phoenix",
    },
    {
```

```
        Key: "Environment",
        Value: "Production",
    },
    {
        Key: "Owner",
        Value: "DataScienceTeam",
    }
];
putBucketTagging("your-production-bucket", myTags);
```

Get Bucket Tagging

功能说明

获取指定BUCKET的标签。该操作需要s3:GetBucketTagging权限，桶的拥有者默认具有该权限。

方法原型

```
const input = { // GetBucketTaggingRequest
  Bucket: "STRING_VALUE" // required
};
const command = new GetBucketTaggingCommand(input);
const response = await client.send(command);
```

参数说明

参数名称	参数描述	类型	是否必须
Bucket	Bucket的名称	String	是

返回结果说明

```
{
  "TagSet": [
    {
      "Value": "val-1",
      "Key": "key-1"
    }
  ],
  "'$metadata'": {
      "httpStatusCode": 200,
      "requestId": "tx00000000000000249b20e-0068ca6f87-9f683575-js10",
      "extendedRequestId": undefined,
      "cfId": undefined,
      "attempts": 1,
      "totalRetryDelay": 0
   }
}
```

返回结果	描述	类型
TagSet	标签集	list
Key	标签key	string
Value	标签value	string

示例

```
import { GetBucketTaggingCommand } from "@aws-sdk/client-s3"
/**
 * 获取指定存储桶的所有标签
 * @param {string} bucketName - 桶的名称
 */
const getBucketTagging = async (bucketName) => {
    try {
        const command = new GetBucketTaggingCommand({
            Bucket: bucketName,
        });

        const response = await zos_client.send(command);
        console.log(`Success! Tags for bucket "${bucketName}" retrieved.`);

        if (response.TagSet && response.TagSet.length > 0) {
            console.log("Tags:");
            // 遍历并打印每一个标签的键和值
            response.TagSet.forEach(tag => {
                console.log(`  - ${tag.Key}: ${tag.Value}`);
            });
        } else {
            // 理论上，如果没有任何标签，API会抛出NoSuchTagSet错误
            // 但为了代码健壮性，保留此检查
            console.log("This bucket has no tags.");
        }

    } catch (error) {
        console.error("An error occurred:", error.name);
        // 如果存储桶存在但没有任何标签，S3会返回一个特定的错误
        if (error.name === 'NoSuchTagSet') {
            console.log(`Bucket "${bucketName}" exists but has no tags.`);
        } else if (error.name === 'NoSuchBucket') {
            console.error(`Error: Bucket "${bucketName}" does not exist.`);
        } else {
            console.error("Error message:", error.message);
        }
    }
};
getBucketTagging("your-bucket-with-tags");
```

Delete Bucket Tagging

功能说明

删除Bucket上的标签。该操作需要s3:PutBucketTagging权限，桶的拥有者默认具有该权限。

方法原型

```
const input = { // DeleteBucketTaggingRequest
  Bucket: "STRING_VALUE" // required
}
const command = new DeleteBucketTaggingCommand(input);
const response = await client.send(command);
```

参数说明

参数名称	参数描述	类型	是否必须
Bucket	Bucket的名称	String	是

返回结果说明

```
{
    "'$metadata'": {
      "httpStatusCode": 200,
      "requestId": "tx0000000000000000249b20e-0068ca6f87-9f683575-js10",
      "extendedRequestId": undefined,
      "cfId": undefined,
      "attempts": 1,
      "totalRetryDelay": 0
   }
}
```

示例

```
import { DeleteBucketTaggingCommand } from "@aws-sdk/client-s3";
/**
 * 删除指定存储桶的所有标签
 * @param {string} bucketName - 桶的名称
 */
const deleteBucketTagging = async (bucketName) => {
    try {
        const command = new DeleteBucketTaggingCommand({
            Bucket: bucketName,
        });

        await zos_client.send(command);
        console.log(`Success! All tags have been deleted from bucket "${bucketName}".`);

    } catch (error) {
        console.error("An error occurred:", error.name);
```

```
        if (error.name === 'NoSuchBucket') {
            console.error(`Error: Bucket "${bucketName}" does not exist.`);
        } else {
            console.error("Error message:", error.message);
        }
    }
};
deleteBucketTagging("your-bucket-with-tags-to-delete");
```

Put Bucket Logging

功能说明

put bucket logging请求设置日志转存参数。所有的日志将会保留到和源存储桶属于同一拥有者的目标存储桶中。桶的拥有者可以设置桶的日志状态。桶的拥有者对所有的日志具有FULL_CONTROL权限，可以通过Grantee授权其他用户，其中Permissions参数指定了用户对日志的访问权限。

方法原型

```
const input = { // PutBucketLoggingRequest
  Bucket: "STRING_VALUE", // required
  BucketLoggingStatus: { // BucketLoggingStatus
    LoggingEnabled: { // LoggingEnabled
      TargetBucket: "STRING_VALUE", // required
      TargetGrants: [ // TargetGrants
        { // TargetGrant
          Grantee: { // Grantee
            DisplayName: "STRING_VALUE",
            EmailAddress: "STRING_VALUE",
            ID: "STRING_VALUE",
            URI: "STRING_VALUE",
            Type: "CanonicalUser" || "AmazonCustomerByEmail" || "Group", // required
          },
          Permission: "FULL_CONTROL" || "READ" || "WRITE",
        },
      ],
      TargetPrefix: "STRING_VALUE", // required
        }
    }
}
const command = new PutBucketLoggingCommand(input);
const response = await client.send(command);
```

参数说明

参数名称	参数描述	类型	是否必须
Bucket	Bucket的名称	String	是

参数名称	参数描述	类型	是否必须
BucketLoggingStatus	日志状态信息。若此参数置为空，则表示关闭日志转存功能	dict	是
LoggingEnabled	描述日志存储位置和日志对象前缀	dict	否
TargetBucket	日志存储位置。可以将日志存放到任意用户拥有的桶中，包含源存储桶。用户可以配置多个源桶的日志均投放到同一个目标存储桶中，在这种情况下，用户可以使用TargetPrefix区分日志来自哪个源存储桶。	string	是
TargetGrants	授权信息	list	否
Grantee	授权许可	dict	否
DisplayName	展示名字	string	否
EmailAddress	邮件地址	integer	否
ID	授权用户ID	string	否
Type	授权类型	string	是
URI	授权组URI	string	否
Permission	日志访问许可	string	否
TargetPrefix	日志对象前缀。若多个源存储桶的日志均写到同一个目标存储桶中，则可以通过目标前缀来区分日志来自哪一个源存储桶	string	是
metadata	http请求返回数据，包括请求返回状态码及RequestID等等	Dict	

返回结果说明

```
{
     "'$metadata'": {
      "httpStatusCode": 200,
      "requestId": "tx0000000000000249b20e-0068ca6f87-9f683575-js10",
      "extendedRequestId": undefined,
      "cfId": undefined,
      "attempts": 1,
      "totalRetryDelay": 0
   }
}
```

返回结果	描述	类型
metadata	http请求返回数据，包括请求返回状态码及RequestID等等	Dict

示例

```
import { PutBucketLoggingCommand } from "@aws-sdk/client-s3";
/**
 * 为指定的源存储桶启用服务器访问日志记录
 * @param {string} sourceBucketName - 需要开启日志记录的桶的名称
 * @param {string} targetBucketName - 用于存储日志文件的目标桶的名称
 * @param {string} logPrefix - 日志文件在目标桶中的前缀 (例如 "logs/")
 */
const putBucketLogging = async (sourceBucketName, targetBucketName, logPrefix) => {
    try {
        const command = new PutBucketLoggingCommand({
            Bucket: sourceBucketName,
            BucketLoggingStatus: {
                LoggingEnabled: {
                    TargetBucket: targetBucketName,
                    TargetPrefix: logPrefix,
                },
            },
        });

        await zos_client.send(command);
        console.log(`Success! Server access logging has been enabled for bucket
"${sourceBucketName}".`);
        console.log(`Logs will be delivered to "${targetBucketName}/${logPrefix}".`);

    } catch (error) {
        console.error("An error occurred:", error.name);
        if (error.name === 'NoSuchBucket') {
            console.error(`Error: Bucket "${sourceBucketName}" does not exist.`);
        } else {
            console.error("Error message:", error.message);
        }
    }
};
const sourceBucket = "my-source-app-bucket";
const targetBucket = "my-centralized-logs-bucket";
const prefix = "access-logs/my-app/";
putBucketLogging(sourceBucket, targetBucket, prefix);
```

Get Bucket Logging

功能说明

get bucket logging请求获取存储桶的日志转存配置

方法原型

```
const input = { // GetBucketLoggingRequest
  Bucket: "STRING_VALUE" // required
};
const command = new GetBucketLoggingCommand(input);
const response = await client.send(command);
```

参数说明

参数名称	参数描述	类型	是否必须
Bucket	Bucket的名称	String	是

返回结果说明

```
{
    "LoggingEnabled": {
      "TargetPrefix": "string",
      "TargetBucket": "string",
      "TargetGrants": [
        {
          "Grantee": {
            "DisplayName": "string",
            "EmailAddress": "string",
            "ID": "string",
            "Type": "CanonicalUser"|"AmazonCustomerByEmail"|"Group",
            "URI": "string"
          },
          "Permission": "FULL_CONTROL"|"READ"|"WRITE"
        }
      ]
    },
    "'$metadata'": {
      "httpStatusCode": 200,
      "requestId": "tx0000000000000000249b20e-0068ca6f87-9f683575-js10",
      "extendedRequestId": undefined,
      "cfId": undefined,
      "attempts": 1,
      "totalRetryDelay": 0
   }
}
```

返回结果	描述	类型
LoggingEnabled	描述日志存储位置和日志对象前缀	dict
TargetBucket	日志存储位置。可以将日志存放到任意用户拥有的桶中，包含源存储桶。用户可以配置多个源桶的日志均投放到同一个目标存储桶中，在这种情况下，用户可以使用TargetPrefix区分日志来自哪个源存储桶。	string

返回结果	描述	类型
TargetGrants	授权信息	list
Grantee	授权许可	dict
DisplayName	展示名字	string
EmailAddress	邮件地址	integer
ID	授权用户ID	string
Type	授权类型	string
URI	授权组URI	string
Permission	日志访问许可	string
TargetPrefix	日志对象前缀。若多个源存储桶的日志均写到同一个目标存储桶中，则可以通过目标前缀来区分日志来自哪一个源存储桶	string
metadata	http请求返回数据，包括请求返回状态码及RequestID等等	Dict

示例

```
import { GetBucketLoggingCommand } from "@aws-sdk/client-s3";
/**
 * 获取指定存储桶的服务器访问日志记录配置
 * @param {string} bucketName - 桶的名称
 */
const getBucketLogging = async (bucketName) => {
    try {
        const command = new GetBucketLoggingCommand({
            Bucket: bucketName
        });

        const response = await zos_client.send(command);
        console.log(`Success! Logging status for bucket "${bucketName}" retrieved.`);

        // 检查 LoggingEnabled 字段是否存在。如果存在，说明已启用日志记录。
        if (response.LoggingEnabled) {
            console.log("Logging is ENABLED.");
            console.log(`  -> Target Bucket: ${response.LoggingEnabled.TargetBucket}`);
            console.log(`  -> Target Prefix: ${response.LoggingEnabled.TargetPrefix}`);
        } else {
            // 如果 LoggingEnabled 字段不存在，说明未启用日志记录。
            console.log("Logging is DISABLED for this bucket.");
        }

    } catch (error) {
        console.error("An error occurred:", error.name);
        if (error.name === 'NoSuchBucket') {
            console.error(`Error: Bucket "${bucketName}" does not exist.`);
        } else {
```

```
            console.error("Error message:", error.message);
        }
    }
};
getBucketLogging("your-source-bucket-with-logging-settings");
```

Put Bucket Versioning

功能说明

Put Bucket Versioning 接口实现启用或者暂停Bucket的版本控制功能。

方法原型

```
const client = new S3Client(config);
const input = { // PutBucketVersioningRequest
  Bucket: "STRING_VALUE", // required
  VersioningConfiguration: { // VersioningConfiguration
    MFADelete: "Enabled" || "Disabled",
    Status: "Enabled" || "Suspended",
  }
};
const command = new PutBucketVersioningCommand(input);
const response = await client.send(command);
```

参数说明

参数名称	参数描述	类型	是否必须
Bucket	Bucket的名称	String	是
VersioningConfiguration	是否开启Bucket的版本控制功能，枚举值：Suspended，Enabled。	Dict	是

返回结果说明

```
{
   "'$metadata'": {
        "httpStatusCode": 200,
        "requestId": "tx0000000000000249b20e-0068ca6f87-9f683575-js10",
        "extendedRequestId": undefined,
        "cfId": undefined,
        "attempts": 1,
        "totalRetryDelay": 0
    }
}
```

返回结果	描述	类型
metadata	http请求返回数据	Dict

示例

```
import { PutBucketVersioningCommand } from "@aws-sdk/client-s3";
/**
 * 设置指定存储桶的版本控制配置 (启用或暂停)
 * @param {string} bucketName - 桶的名称
 * @param {string} versioningStatus - 版本控制状态, 必须是 'Enabled' 或 'Suspended'
 */
const putBucketVersioning = async (bucketName, versioningStatus) => {
    // Basic validation for the status parameter
    if (versioningStatus !== 'Enabled' && versioningStatus !== 'Suspended') {
        console.error("Invalid status. Must be 'Enabled' or 'Suspended'.");
        return;
    }

    try {
        const command = new PutBucketVersioningCommand({
            Bucket: bucketName,
            VersioningConfiguration: {
                Status: versioningStatus,
                // MFADelete: "Enabled" // (Optional) Uncomment to require MFA for permanent
deletes
            },
        });

        await zos_client.send(command);
        console.log(`Success! Versioning for bucket "${bucketName}" has been set to
"${versioningStatus}".`);

    } catch (error) {
        console.error("An error occurred:", error.name);
        if (error.name === 'NoSuchBucket') {
            console.error(`Error: Bucket "${bucketName}" does not exist.`);
        } else {
            console.error("Error message:", error.message);
        }
    }
};
putBucketVersioning("my-bucket-to-version", "Enabled");
```

Get Bucket Versioning

功能说明

Get Bucket Versioning 接口实现获得Bucket的版本控制配置。

方法原型

```
const input = { // GetBucketVersioningRequest
  Bucket: "STRING_VALUE" // required
}
const command = new GetBucketVersioningCommand(input);
const response = await client.send(command);
```

参数说明

参数名称	参数描述	类型	是否必须
Bucket	Bucket的名称	String	是

返回结果说明

```
{
    "Status": "Enabled",
    "MFADelete": "Disabled",
    "'$metadata'": {
        "httpStatusCode": 200,
        "requestId": "tx00000000000000249b20e-0068ca6f87-9f683575-js10",
        "extendedRequestId": undefined,
        "cfId": undefined,
        "attempts": 1,
        "totalRetryDelay": 0
    }
}
```

返回结果	返回结果描述	类型
Status	Bucket的版本控制配置，假如没有设置过该配置，该字段不会返回	String
MFADelete	是否启用MFADelete删除	String
metadata	http请求返回数据	Dict

示例

```
import { GetBucketVersioningCommand } from "@aws-sdk/client-s3"
/**
 * 获取指定存储桶的版本控制状态
 * @param {string} bucketName - 桶的名称
 */
const getBucketVersioning = async (bucketName) => {
```

```
    try {
        const command = new GetBucketVersioningCommand({
            Bucket: bucketName,
        });

        const response = await zos_client.send(command);
        console.log(`Success! Versioning status for bucket "${bucketName}" retrieved.`);

        // The response.Status will be 'Enabled', 'Suspended', or undefined
        const status = response.Status || 'Disabled';

        console.log(`  -> Versioning Status: ${status}`);

        // Additionally, check if MFA Delete is enabled
        if (response.MFADelete === 'Enabled') {
            console.log(`  -> MFA Delete: Enabled`);
        } else {
            console.log(`  -> MFA Delete: Disabled`);
        }

    } catch (error) {
        console.error("An error occurred:", error.name);
        if (error.name === 'NoSuchBucket') {
            console.error(`Error: Bucket "${bucketName}" does not exist.`);
        } else {
            console.error("Error message:", error.message);
        }
    }
};
getBucketVersioning("your-versioned-bucket");
```

Object 操作

Get Object

功能说明

Get Object 请求可以将一个文件（Object）下载至本地。该操作需要对目标 Object 具有读权限或目标 Object 对所有人都开放了读权限（公有读）。

方法原型

```
const input = { // GetObjectRequest
  Bucket: "STRING_VALUE", // required
  IfMatch: "STRING_VALUE",
  IfModifiedSince: new Date("TIMESTAMP"),
  IfNoneMatch: "STRING_VALUE",
  IfUnmodifiedSince: new Date("TIMESTAMP"),
  Key: "STRING_VALUE", // required
  Range: "STRING_VALUE",
  VersionId: "STRING_VALUE"
};
const command = new GetObjectCommand(input);
const response = await client.send(command);
```

参数说明

参数名称	参数描述	类型	是否必须
Bucket	Bucket的名称	String	是
IfMatch	当对象的Etag和IfMatch中的值一致时返回对象，否则返回412错误	String	否
IfModifiedSince	只有指定时间之后有修改记录的对象才会返回对象，否则返回304错误	String	否
IfNoneMatch	只有对象的Etag不满足指定字符串时才会返回对象，否则返回304错误	String	否
IfUnmodifiedSince	只有指定时间之后没有修改记录的对象才会返回，否则返回412错误	String	否
Key	Object的名称	String	是
Range	指定下载对象的字节范围，例如 bytes=0-9 表示下载开头10个byte	String	否
VersionId	对象的某个特定版本的版本号，没有该版本则返回404错误	String	否

返回结果说明

```
{
    "Body": IncomingMessage,
    "AcceptRanges": "bytes",
    "ContentType": "application/octet-stream",
    "'$metadata'": {
        "httpStatusCode": 200,
        "requestId": "tx0000000000000249b20e-0068ca6f87-9f683575-js10",
        "extendedRequestId": undefined,
        "cfId": undefined,
```

```
        "attempts": 1,
        "totalRetryDelay": 0
    },
    "LastModified": "2025-03-21T00: 55: 48.000Z",
    "ContentRange": "bytes 20-40/20971520",
    "ETag": "7fa986abc03d636c14dea69df48a7090-2",
    "VersionId": "PS9l7ww77taWqBwqcTuXmgv7rvDdKb6",
    "ContentLength": 21,
    "Metadata": {},
    "ContentDisposition": "attachment"
}
```


参数名称	参数描述	类型
Body	返回对象的内容	StreamingBody
metadata	http请求返回数据	Dict
LastModified	对象的创建时间	Datetime
ContentRange	指明对象在返回的响应中包含的部分	String
ETag	对象的ETag	String
ContentLength	响应体的长度	Integer
Metadata	和对象一起存储的元数据信息	Dict
ContentDisposition	设置响应的 Content-Disposition 头部	String
VersionId	版本标识	String

示例

```
import { GetObjectCommand } from "@aws-sdk/client-s3";
import { createWriteStream } from "fs";
import { Readable } from "stream";

/**
 * 从指定的存储桶中获取一个对象并将其保存到本地文件
 * @param {string} bucketName - 存储桶的名称
 * @param {string} objectKey - 要获取的对象的键（名称/路径）
 * @param {string} localFilePath - 要保存文件的本地路径（包括文件名）
 */
const getObjectAndSaveToFile = async (bucketName, objectKey, localFilePath) => {
    try {
        const command = new GetObjectCommand({
            Bucket: bucketName,
            Key: objectKey,
        });
        const response = await zos_client.send(command);
        if (response.Body instanceof Readable) {
            // 创建一个可写流到本地文件
```

```
            const fileStream = createWriteStream(localFilePath);
            // 将S3对象的可读流通过管道传输到本地文件流
            response.Body.pipe(fileStream);
            // 监听'finish'事件以确认文件已完全写入
            fileStream.on('finish', () => {
                console.log(`Success! Object "${objectKey}" from bucket "${bucketName}" has
been downloaded to "${localFilePath}".`);
            });
            // 监听错误事件
            fileStream.on('error', (err) => {
                 console.error(`Error writing to file "${localFilePath}":`, err);
            });
        } else {
             throw new Error("Response body is not a readable stream.");
        }

    } catch (error) {
        console.error("An error occurred:", error.name);
        if (error.name === 'NoSuchKey') {
            console.error(`Error: The object "${objectKey}" does not exist in bucket
"${bucketName}".`);
        } else if (error.name === 'NoSuchBucket') {
            console.error(`Error: Bucket "${bucketName}" does not exist.`);
        } else {
            // 捕获其他所有错误
            console.error("Error message:", error.message);
        }
    }
};
const BUCKET_NAME = "your-bucket-name";
const OBJECT_KEY = "path/to/your/file.zip";
const LOCAL_FILE_PATH = "./downloads/downloaded-file.zip";
getObjectAndSaveToFile(BUCKET_NAME, OBJECT_KEY, LOCAL_FILE_PATH);
```

Head Object

功能说明

Head Object 请求可以获取对应 Object 的元数据，Head 的权限与 Get 的权限一致。

方法原型

```
const input = { // HeadObjectRequest
  Bucket: "STRING_VALUE", // required
  IfMatch: "STRING_VALUE",
  IfModifiedSince: new Date("TIMESTAMP"),
  IfNoneMatch: "STRING_VALUE",
  IfUnmodifiedSince: new Date("TIMESTAMP"),
  Key: "STRING_VALUE", // required
  VersionId: "STRING_VALUE"
};
const command = new HeadObjectCommand(input);
const response = await client.send(command);
```

参数说明

参数名称	参数描述	类型	是否必须
Bucket	Bucket的名称	String	是
IfMatch	当文件的Etag和IfMatch中的值一致时返回文件元数据，否则返回412错误	String	否
IfModifiedSince	只有指定时间之后有修改记录的文件才会返回元数据，否则返回304错误	String	否
IfNoneMatch	只有文件的Etag不满足指定字符串时才会返回元数据，否则返回304错误	String	否
IfUnmodifiedSince	只有指定时间之后没有修改记录的文件才会返回元数据，否则返回412错误	String	否
Key	Object的名称	String	是
VersionId	文件的某个特定版本的版本号，没有该版本则返回404错误	String	否

返回结果说明

```
{
  "$metadata": {
    "httpStatusCode": 200,
    "requestId": "tx0000000000000298b6cd-0068cc0304-a0170eb5-js10",
    "extendedRequestId": undefined,
    "cfId": undefined,
    "attempts": 1,
    "totalRetryDelay": 0
  },
  "AcceptRanges": "bytes",
  "LastModified": "2025-03-21T00:55:48.000Z",
  "ContentLength": 536,
  "ETag": "\"bb3841421734dbc2a0d6b4ac7272ad33-1\"",
```

```
  "VersionId": "PS9l7ww77taWqBwqcTuXmgv7rvDdKb6",
  "ContentDisposition": "attachment",
  "ContentType": "binary/octet-stream",
  "StorageClass": "STANDARD_IA",
  "Metadata": {}
}
```

返回结果	描述	类型
metadata	http请求返回数据	Dict
HTTPStatusCode	http请求状态码	Int
content-length	文件大小，单位字节	Int
ETag	文件最新版本的etag	String
StorageClass	文件的存储类型	String
VersionId	如果存储桶启用了版本控制，则返回对象的版本 ID。	String
ContentDisposition	指定内容的呈现形式，例如 attachment 表示应下载文件。	String
ContentType	对象的标准 MIME 类型。	String

示例

```
import { HeadObjectCommand } from "@aws-sdk/client-s3";

/**
 * 获取指定对象的元数据，而不下载对象本身
 * @param {string} bucketName - 存储桶的名称
 * @param {string} objectKey - 要查询的对象的键（名称/路径）
 */
const headObject = async (bucketName, objectKey) => {
    try {
        const command = new HeadObjectCommand({
            Bucket: bucketName,
            Key: objectKey,
        });

        const response = await zos_client.send(command);
        console.log(response);
        console.log(`Success! Metadata for object "${objectKey}" from bucket "${bucketName}"
retrieved.`);
        console.log(` -> ETag: ${response.ETag}`);
        console.log(` -> Content-Type: ${response.ContentType}`);
        console.log(` -> Content-Length: ${response.ContentLength} bytes`);
        console.log(` -> Last Modified: ${response.LastModified}`);
        if (response.Metadata && Object.keys(response.Metadata).length > 0) {
             console.log(" -> Custom Metadata:", response.Metadata);
        }
```

```
    } catch (error) {
        // 对于 HeadObject，当对象不存在时，SDK v3 会抛出一个 'NotFound' 错误。
        if (error.name === 'NotFound') {
             console.error(`Error: The object "${objectKey}" was not found in bucket
"${bucketName}".`);
        } else if (error.name === 'NoSuchBucket') {
            console.error(`Error: Bucket "${bucketName}" does not exist.`);
        }
        else {
             console.error("An error occurred:", error.name);
             console.error("Error message:",
error.message);
    }
    }
};
const BUCKET_NAME = "your-bucket-name";
const OBJECT_KEY = "path/to/your/object.txt";
headObject(BUCKET_NAME, OBJECT_KEY);
```

Put Object

功能说明

Put Object 请求可以将一个文件（Object）上传至指定 Bucket。对象权限默认为私有权限。原始数据格式支持字节数据，字符串，文件对象，内存流对象等。

方法原型

```
const input = { // PutObjectRequest
  ACL: "private" || "public-read" || "public-read-write" || "authenticated-read",
  Body: "String" || "Buffer",
  Bucket: "STRING_VALUE", // required
  Key: "STRING_VALUE", // required
  ContentMD5: "STRING_VALUE",
  Metadata: { // Metadata
    "<keys>": "STRING_VALUE",
  },
  StorageClass: 'GLACIER'|'STANDARD_IA'|'STANDARD',
};
const command = new PutObjectCommand(input);
const response = await client.send(command);
```

参数说明

参数名称	参数描述	类型	是否必须
ACL	Object的访问控制;传空默认为私有权限,可选值: private, public-read, public-read-write, authenticated-read	String	否
Body	对象的数据	Bytes	否

参数名称	参数描述	类型	是否必须
Bucket	存入bucket的名称	String	是
ContentMD5	对象数据的MD5	String	否
Key	对象的Key	String	是
Metadata	Object 的元数据	Dict	否
StorageClass	Object 的存储类型，现在支持'GLACIER'|'STANDARD_IA'|'STANDARD'	String	否

返回结果说明

```
{
    "ETag": "21fd59a7339f2e5f73c97039254fd784",
    "VersionId": "2Nm5PNw5vKUcCrpPqTxy3csej7ihMny",
    "$metadata": {
        "httpStatusCode": 200,
        "requestId": "tx00000000000000298b6cd-0068cc0304-a0170eb5-js10",
        "extendedRequestId": undefined,
        "cfId": undefined,
        "attempts": 1,
        "totalRetryDelay": 0
    }
}
```

返回结果	描述	类型
ETag	Object的ETag	String
VersionId	上传对象的版本ID	String
metadata	http请求返回数据	Dict

示例

```
import { PutObjectCommand } from "@aws-sdk/client-s3";
import fs from "fs";
import path from "path";

/**
 * 将对象（例如，文件或文本）上传到指定的存储桶
 * @param {string} bucketName - 目标存储桶的名称
 * @param {string} objectKey - 在存储桶中为对象指定的键（名称/路径）
 * @param {string | Buffer } body - 要上传的对象内容
 */
const putObject = async (bucketName, objectKey, body) => {
    try {
        const command = new PutObjectCommand({
            Bucket: bucketName,
```

```
            Key: objectKey,
            Body: body,
            // (可选) 您还可以设置其他参数，例如:
            // ContentType: "image/jpeg",
            // ACL: "public-read",
            // Metadata: {
            //     "my-custom-key": "my-custom-value"
            // }
        });

        const response = await zos_client.send(command);
        console.log(response);
        console.log(`Success! Object "${objectKey}" uploaded to bucket "${bucketName}".`);
        console.log(` -> ETag: ${response.ETag}`);
        if(response.VersionId) {
            console.log(` -> Version ID: ${response.VersionId}`);
        }

    } catch (error) {
        console.error("An error occurred:", error.name);
        if (error.name === 'NoSuchBucket') {
            console.error(`Error: Bucket "${bucketName}" does not exist.`);
        } else {
            // 捕获其他所有错误
            console.error("Error message:", error.message);
        }
    }
};
const BUCKET_NAME = "your-bucket-name";
const LOCAL_FILE_PATH = "./path/to/your/local-file.txt";
const OBJECT_KEY_FILE = `uploads/${path.basename(LOCAL_FILE_PATH)}`;
const fileStream = fs.readFileSync(LOCAL_FILE_PATH);
putObject(BUCKET_NAME, OBJECT_KEY_FILE, fileStream);
```

Delete Object

功能说明

Delete Object 请求可以将一个文件（Object）删除。

方法原型

```
const input = { // DeleteObjectRequest
  Bucket: "STRING_VALUE", // required
  Key: "STRING_VALUE", // required
  VersionId: "STRING_VALUE"
};
const command = new DeleteObjectCommand(input);
const response = await client.send(command);
```

参数说明

参数名称	参数描述	类型	是否必须
Bucket	指定Object所在的Bucket	String	是
Key	指定要删除的Key	String	是
VersionId	指定要删除的Object的VersionId	String	否

返回结果说明


```
{
  "$metadata": {
    "httpStatusCode": 204,
    "requestId": "tx000000000000005e13ce3-0068cd1953-8de19637-js10",
    "extendedRequestId": null,
    "cfId": null,
    "attempts": 1,
    "totalRetryDelay": 0
  },
  "DeleteMarker": true,
  "VersionId": "ZOl5HJrvIFO..V-B5Dh79zFEawycqPU"
}
```


返回结果	描述	类型
metadata	http请求返回数据	Dict
DeleteMarker	为true时，该存储桶开启了版本控制 (Versioning)	Bool
VersionId	若开启版本控制，则返回最新生成的删除标记的版本 ID	String

示例

```
import { DeleteObjectCommand } from "@aws-sdk/client-s3";
/**
 * 从指定的存储桶中删除一个对象
 * @param {string} bucketName - 目标存储桶的名称
 * @param {string} objectKey - 要删除的对象的键（名称/路径）
 */
const deleteObject = async (bucketName, objectKey) => {
    try {
        const command = new DeleteObjectCommand({
            Bucket: bucketName,
            Key: objectKey,
        });

        let response = await zos_client.send(command);
        console.log(response);
        console.log(`Success! Object "${objectKey}" has been deleted from bucket
"${bucketName}".`);
```

```
    } catch (error) {
        console.error("An error occurred during deletion:", error.name);
        if (error.name === 'NoSuchBucket') {
            console.error(`Error: Bucket "${bucketName}" does not exist.`);
        } else {
            console.error("Error message:", error.message);
        }
    }
};
const BUCKET_NAME = "your-bucket-name";
const OBJECT_KEY_TO_DELETE = "uploads/local-file.txt";
deleteObject(BUCKET_NAME, OBJECT_KEY_TO_DELETE);
```

Delete Multiple Object

功能说明

Delete Multiple Object 请求实现批量删除文件，最大支持单次删除 1000 个文件。对于返回结果，提供 Verbose 和 Quiet 两种结果模式。Verbose 模式将返回每个 Object 的删除结果，默认为Verbose 模式；Quiet 模式只返回报错的 Object 信息。

方法原型

```
const input = { // DeleteObjectsRequest
  Bucket: "STRING_VALUE", // required
  Delete: {
    Objects: [ // ObjectIdentifierList // required
      { // ObjectIdentifier
        Key: "STRING_VALUE", // required
        VersionId: "STRING_VALUE"
      },
    ],
    Quiet: true || false,
  }
};
const command = new DeleteObjectsCommand(input);
const response = await client.send(command);
```

参数说明

参数名称	参数描述	类型	是否必须
Bucket	指定Object所在的Bucket	String	是
Delete	其中Delete['Objects'] 指定要删除的对象数组， Delete['Quiet']为bool变量，指定Verbose模式的关和开	Dict	是
Objects	删除的对象，key为键名，VersionId为标本号	List	是
Key	对象的键	String	是

参数名称	参数描述	类型	是否必须
VersionId	对象的版本ID	String	否
Quiet	是否启用静默模式	Bool	否

返回结果说明

Verbose模式开

```
{
 "Deleted": [
    {
      "Key": "test_put1"
    },
    {
      "Key": "test_put2"
    }
  ],
  "$metadata": {
    "httpStatusCode": 204,
    "requestId": "tx000000000000005e13ce3-0068cd1953-8de19637-js10",
    "extendedRequestId": null,
    "cfId": null,
    "attempts": 1,
    "totalRetryDelay": 0
  }
}
```

Verbose模式关

```
{
    "$metadata": {
    "httpStatusCode": 204,
    "requestId": "tx000000000000005e13ce3-0068cd1953-8de19637-js10",
    "extendedRequestId": null,
    "cfId": null,
    "attempts": 1,
    "totalRetryDelay": 0
  }
}
```

返回结果	描述	类型
metadata	http请求返回数据	Dict
Deleted	删除成功的对象列表	List

示例

```
import { DeleteObjectsCommand } from "@aws-sdk/client-s3";
/**
 * 从指定的存储桶中删除多个对象
 * @param {string} bucketName - 目标存储桶的名称
 * @param {Array<Object>} objectsToDelete - 要删除的对象数组，格式为 [{ Key: "key1" }, { Key:
"key2" }]
 */
const deleteMultipleObjects = async (bucketName, objectsToDelete, isQuiet = true) => {
    // 确保至少有一个要删除的对象
    if (!objectsToDelete || objectsToDelete.length === 0) {
        console.log("没有提供要删除的对象。");
        return;
    }
    try {
        const command = new DeleteObjectsCommand({
            Bucket: bucketName,
            Delete: {
                Objects: objectsToDelete,
                Quiet: isQuiet, // 设置为 false 以便在响应中获取所有成功删除的对象列表
            },
        });

        const response = await zos_client.send(command);
        console.log(response);
        console.log("批量删除操作已完成。");

        // 打印成功删除的对象
        if (response.Deleted && response.Deleted.length > 0) {
            console.log("成功删除的对象:");
            response.Deleted.forEach(deleted => console.log(` -> ${deleted.Key}`));
        }

        // 打印删除失败的对象及其错误信息
        if (response.Errors && response.Errors.length > 0) {
            console.error("删除失败的对象:");
            response.Errors.forEach(error => console.error(` -> ${error.Key}:
${error.Message} (Code: ${error.Code})`));
        }

    } catch (error) {
        console.error("执行批量删除时发生严重错误:", error.name);
        if (error.name === 'NoSuchBucket') {
            console.error(`错误: 存储桶 "${bucketName}" 不存在。`);
        } else {
            console.error("错误信息:", error.message);
        }
    }
};
const BUCKET_NAME = "your-bucket-name";
const OBJECTS_TO_DELETE = [
    { Key: "uploads/file-to-delete-1.txt" },
```

```
    { Key: "images/archive/photo.jpg" },
    { Key: "logs/non-existent-log.log" },
];
deleteMultipleObjects(BUCKET_NAME, OBJECTS_TO_DELETE);
```

Put Object ACL

功能说明

设置Object的ACL，控制对Object的访问权限。该操作需要用户具有WRITE_ACP权限。
有三种方式设置ACL，三种方式不可同时使用，每次只能给一种参数赋值。其中，通过ACL参数方式进行操作，是设置预定义的固定的ACL，不能针对特定用户进行授权，且该参数实现的效果，也可以借由另外两种方式实现，该参数使用请求头进行传递；AccessControlPolicy参数方式和Grant*参数方式则可以针对特定用户进行授权，AccessControlPolicy方式通过请求体传递，而Grant*方式通过请求头传递。三种方式都会覆盖原有ACL属性，包括对象所有者自身的权限，如需保留原有ACL属性，应将需要保留的原ACL添加到本次操作的授权中（ACL参数方式会默认将对象所有者权限设为FULL_CONTROL，而另外两种方式则不会保留任何原ACL属性）。

方法原型

```
const input = { // PutObjectAclRequest
  ACL: "private" || "public-read" || "public-read-write" || "authenticated-read",
  AccessControlPolicy: { // AccessControlPolicy
    Grants: [ // Grants
      { // Grant
        Grantee: { // Grantee
          DisplayName: "STRING_VALUE",
          ID: "STRING_VALUE",
          URI: "STRING_VALUE",
          Type: "CanonicalUser" || "Group", // required
        },
        Permission: "FULL_CONTROL" || "WRITE" || "WRITE_ACP" || "READ" || "READ_ACP",
      },
    ],
    Owner: { // Owner
      ID: "STRING_VALUE",
    },
  },
  Bucket: "STRING_VALUE", // required
  GrantRead: "STRING_VALUE",
  GrantReadACP: "STRING_VALUE",
  GrantWrite: "STRING_VALUE",
  GrantWriteACP: "STRING_VALUE",
  Key: "STRING_VALUE", // required
  VersionId: "STRING_VALUE"
};
const command = new PutObjectAclCommand(input);
const response = await client.send(command);
```

参数说明

参数名称	参数描述	类型	是否必须
Bucket	Bucket的名称	String	是
Key	对象名	String	是
VersionId	多版本场景下，指定对象的特定版本	String	否
ACL	预定义的固定ACL， 取值范围 'private' 'public-read'	String	ACL参数方式则必须，其他两种方式，则不能使用
AccessControlPolicy	包含多个授权列表和一个所有者参数	dict	该方式下必须，其他两种方式下则不能使用
Grants	授权列表	list	该方式下必须
Grantee	被授权用户	dict	该方式下必须
Type	被授权用户类型， 取值范围 'CanonicalUser' Group'	String	该方式下必须
ID(Grantee)	被授权用户ID	String	Type为'CanonicalUser'，则该字段必须
URI	被授权组URI 取值范围 为所有用户：`http://acs.amazonaws.com/groups/global/AllUsers` 所有认证用户：`http://acs.amazonaws.com/groups/global/AuthenticatedUsers`	String	如果Type为'Group'，则该字段必须
Permission	向被授权用户授予的权限， 取值范围 'FULL_CONTROL' 'WRITE' 'WRITE_ACP' 'READ''READ_ACP'	String	该方式下必须
Owner	对象所有者	dict	该方式下必须
ID(Owner)	对象所有者ID	String	该方式下必须
GrantFullControl	被授权用户可以对桶进行read, write, read ACP, and write ACP操作 以下Grant*参数，格式都是"id=xxxx"或"emailAddress=xxxx"或者"uri=xxxx"以及他们的组合（用逗号连接）（注: 当前在集群互联的资源池，emailAddress方式不支持，不推荐使用emailAddress方式）	String	否
GrantRead	被授权用户可以对对象进行读操作	String	否
GrantWrite	被授权用户可以对对象进行写操作，删除或覆盖写该对象	String	否
GrantReadACP	被授权用户可以读取对象的ACL	String	否
GrantWriteACP	被授权用户可以修改对象的ACL	String	否

返回结果说明

```
{
    "$metadata": {
    "httpStatusCode": 204,
    "requestId": "tx000000000000005e13ce3-0068cd1953-8de19637-js10",
    "extendedRequestId": null,
    "cfId": null,
    "attempts": 1,
    "totalRetryDelay": 0
  }
}
```

返回结果	描述	类型
metadata	http请求返回数据	Dict

示例

```
import { PutObjectAclCommand } from "@aws-sdk/client-s3";

/**
 * 为指定的对象设置访问控制列表 (ACL)
 * @param {string} bucketName - 目标存储桶的名称
 * @param {string} objectKey - 目标对象的键（名称/路径）
 * @param {object} aclConfig - ACL 配置对象，可以包含 ACL (canned) 或 AccessControlPolicy
 */
const putObjectAcl = async (bucketName, objectKey, aclConfig) => {
    try {
        const command = new PutObjectAclCommand({
            Bucket: bucketName,
            Key: objectKey,
            ...aclConfig,
        });

        await zos_client.send(command);

        console.log(`Success! ACL for object "${objectKey}" has been updated in bucket
"${bucketName}".`);

    } catch (error) {
        console.error("An error occurred while setting ACL:", error.name);
        if (error.name === 'NoSuchKey') {
            console.error(`Error: Object "${objectKey}" not found in bucket
"${bucketName}".`);
        } else if (error.name === 'NoSuchBucket') {
            console.error(`Error: Bucket "${bucketName}" does not exist.`);
        } else {
            console.error("Error message:", error.message);
        }
    }
```

```
};
const detailedAclConfig = {
    AccessControlPolicy: {
        Grants: [
            {
                Grantee: {
                    // 替换为被授权用户的 Canonical User ID
                    ID: "被授权用户的CanonicalUserID",
                    Type: "CanonicalUser",
                },
                Permission: "READ",
            },
        ],
        // 必须指定对象的所有者
        Owner: {
            ID: "对象所有者的CanonicalUserID",
        },
    },
};
const BUCKET_NAME = "your-bucket-name";
const OBJECT_KEY = "uploads/your-file.txt";
putObjectAcl(BUCKET_NAME, OBJECT_KEY, detailedAclConfig);
```

Get Object ACL

功能说明

获取指定Object的 ACL。该操作需要READ_ACP权限。该功能返回的结果与Put Object ACL参数一致，但是需要注意的是，如果以邮箱类型授权，返回结果中将会以对应被授权用户ID形式出现，即Type不会是AmazonCustomerByEmail，而是CanonicalUser。

方法原型

```
const input = { // GetObjectAclRequest
  Bucket: "STRING_VALUE", // required
  Key: "STRING_VALUE", // required
  VersionId: "STRING_VALUE"
};
const command = new GetObjectAclCommand(input);
const response = await client.send(command);
```

参数说明

参数名称	参数描述	类型	是否必须
Bucket	Bucket的名称	String	是
Key	对象名	String	是
VersionId	多版本场景下，指定对象的特定版本	String	否

返回结果说明


```
{
    "Owner": {
        "DisplayName": "test-1",
        "ID": "test-1"
    },
    "Grants": [
        {
            "Grantee": {
                "Type": "Group",
                "URI": "http://acs.amazonaws.com/groups/global/AllUsers"
            },
            "Permission": "READ"
        },
        {
            "Grantee": {
                "EmailAddress": "",
                "Type": "CanonicalUser",
                "DisplayName": "test-1",
                "ID": "test-1"
            },
            "Permission": "FULL_CONTROL"
        }
    ],
    "$metadata": {
        "httpStatusCode": 204,
        "requestId": "tx000000000000005e13ce3-0068cd1953-8de19637-js10",
        "extendedRequestId": null,
        "cfId": null,
        "attempts": 1,
        "totalRetryDelay": 0
    }
}
```


返回结果	描述	类型
Owner	对象所有者	Dict
DisplayName(Owner)	对象所有者的展示名	String
ID(Owner)	对象所有者的用户ID	String
Grants	授权列表	list
Grantee	被授权用户	dict
Type	被授权用户类型	string
URI	被授权组URI	string
ID(Grantee)	被授权用户ID	string

返回结果	描述	类型
DisplayName(Grantee)	被授权用户展示名	string
Permission	被授权权限	string
metadata	http请求返回数据	Dict

示例

```
import { GetObjectAclCommand } from "@aws-sdk/client-s3";

/**
 * 获取指定对象的访问控制列表 (ACL)
 * @param {string} bucketName - 目标存储桶的名称
 * @param {string} objectKey - 目标对象的键（名称/路径）
 */
const getObjectAcl = async (bucketName, objectKey) => {
    try {
        const command = new GetObjectAclCommand({
            Bucket: bucketName,
            Key: objectKey,
        });

        const response = await zos_client.send(command);
        console.log(`Success! ACL for object "${objectKey}" in bucket "${bucketName}" has
been retrieved.`);
        // 打印所有者信息
        console.log(" -> Owner:", JSON.stringify(response.Owner, null, 2));
        // 打印授权列表
        console.log(" -> Grants:", JSON.stringify(response.Grants, null, 2));

    } catch (error) {
        console.error("An error occurred while getting ACL:", error.name);
        if (error.name === 'NoSuchKey') {
            console.error(`Error: Object "${objectKey}" not found in bucket
"${bucketName}".`);
        } else if (error.name === 'NoSuchBucket') {
            console.error(`Error: Bucket "${bucketName}" does not exist.`);
        } else {
            console.error("Error message:", error.message);
        }
    }
};
const BUCKET_NAME = "your-bucket-name";
const OBJECT_KEY = "uploads/your-file.txt";
getObjectAcl(BUCKET_NAME, OBJECT_KEY);
```

Put Object Tagging

功能说明

将提供的标签集设置为存储桶中已存在的对象。标签是一个键值对。请注意，标签的最大数量限制为每个对象 10 个标签。要使用此操作，您必须具有执行 s3:PutObjectTagging 操作的权限。 默认情况下，Bucket 拥有者拥有此权限，并且可以将此权限授予其他人。要放置任何其他版本的标签，请使用 versionId 查询参数。 您还需要 s3:PutObjectVersionTagging 操作的权限。

方法原型

```
const input = { // PutObjectTaggingRequest
  Bucket: "STRING_VALUE", // required
  Key: "STRING_VALUE", // required
  VersionId: "STRING_VALUE",
  Tagging: { // Tagging
    TagSet: [ // TagSet // required
      { // Tag
        Key: "STRING_VALUE", // required
        Value: "STRING_VALUE", // required
      },
    ],
  }
};
const command = new PutObjectTaggingCommand(input);
const response = await client.send(command);
```

参数说明

参数名称	参数描述	类型	是否必须
Bucket	Bucket的名称	String	是
Key	Object的名称	String	是
VersionId	多版本场景下，指定对象的特定版本。不指定时，默认为最新版本的Id，如存在null版本号，并需对其操作，则需指定VersionId='null'。VersionId=''等效于指定最新版本的Id。	String	否
Tagging	标签集容器，内部为标签列表，最多50个标签，每个标签都是键值对，key最大128字节，value最大256字节，value可以为空，key和value均为utf-8编码	dict	是

返回结果说明

```
{
    "$metadata": {
        "httpStatusCode": 204,
        "requestId": "tx000000000000005e13ce3-0068cd1953-8de19637-js10",
        "extendedRequestId": null,
        "cfId": null,
        "attempts": 1,
        "totalRetryDelay": 0
    }
}
```

示例

```
import { PutObjectTaggingCommand } from "@aws-sdk/client-s3";

/**
 * 为指定的对象设置标签
 * @param {string} bucketName - 目标存储桶的名称
 * @param {string} objectKey - 目标对象的键（名称/路径）
 * @param {Array<{Key: string, Value: string}>} tagSet - 要应用的标签集数组
 */
const putObjectTagging = async (bucketName, objectKey, tagSet) => {
    try {
        const command = new PutObjectTaggingCommand({
            Bucket: bucketName,
            Key: objectKey,
            Tagging: {
                TagSet: tagSet,
            },
        });

        await zos_client.send(command);

        console.log(`Success! Tags for object "${objectKey}" in bucket "${bucketName}" have
been set.`);

    } catch (error) {
        console.error("An error occurred while setting tags:", error.name);
        if (error.name === 'NoSuchKey') {
            console.error(`Error: Object "${objectKey}" not found in bucket
"${bucketName}".`);
        } else if (error.name === 'NoSuchBucket') {
            console.error(`Error: Bucket "${bucketName}" does not exist.`);
        } else {
            console.error("Error message:", error.message);
        }
    }
};
const BUCKET_NAME = "your-bucket-name";
const OBJECT_KEY = "uploads/your-file.txt";
```

```
// 定义要应用的标签集
const ObjectTags = [
    {
        Key: "Project",
        Value: "Blue",
    },
    {
        Key: "Status",
        Value: "PendingReview",
    },
];
putObjectTagging(BUCKET_NAME, OBJECT_KEY, ObjectTags);
```

Get Object Tagging

功能说明

返回对象的标签集。要使用此操作，您必须具有执行s3:GetObjectTagging操作的权限。默认情况下，操作返回有关对象当前版本的信息。对于多版本的存储桶，您的存储桶中可以有一个对象的多个版本。此时，要检索任何其他版本的标签，请使用 versionId 查询参数。同时，您还需要s3:GetObjectVersionTagging操作的权限。
默认情况下，存储桶拥有者具有此权限，并且可以将此权限授予其他人。

方法原型

```
const input = { // GetObjectTaggingRequest
  Bucket: "STRING_VALUE", // required
  Key: "STRING_VALUE", // required
  VersionId: "STRING_VALUE"
};
const command = new GetObjectTaggingCommand(input);
const response = await client.send(command);
```

参数说明

参数名称	参数描述	类型	是否必须
Bucket	Bucket的名称	String	是
Key	Object的名称	String	是
VersionId	多版本场景下，指定对象的特定版本。不指定时，默认为最新版本的Id，如存在null版本号，并需对其操作，则需指定VersionId='null'。VersionId=''等效于指定最新版本的Id。	String	否

返回结果说明

```
{
    "$metadata": {
        "httpStatusCode": 204,
        "requestId": "tx000000000000005e13ce3-0068cd1953-8de19637-js10",
```

```
        "extendedRequestId": null,
        "cfId": null,
        "attempts": 1,
        "totalRetryDelay": 0
    }
    "TagSet": [{
        "Key": "key-1",
        "Value": "val-1"
    }]
}
```

示例

```
import { GetObjectTaggingCommand } from "@aws-sdk/client-s3";
/**
 * 获取指定对象的标签集
 * @param {string} bucketName - 目标存储桶的名称
 * @param {string} objectKey - 目标对象的键（名称/路径）
 */
const getObjectTagging = async (bucketName, objectKey) => {
    try {
        const command = new GetObjectTaggingCommand({
            Bucket: bucketName,
            Key: objectKey,
        });

        const response = await zos_client.send(command);
        console.log(`Success! Tags for object "${objectKey}" in bucket "${bucketName}" have
been retrieved.`);
        console.log(" -> Tags:", JSON.stringify(response.TagSet, null, 2));

    } catch (error) {
        console.error("An error occurred while getting tags:", error.name);
        if (error.name === 'NoSuchKey') {
            console.error(`Error: Object "${objectKey}" not found in bucket
"${bucketName}".`);
        } else if (error.name === 'NoSuchBucket') {
            console.error(`Error: Bucket "${bucketName}" does not exist.`);
        } else {
            console.error("Error message:", error.message);
        }
    }
};
const BUCKET_NAME = "your-bucket-name";
const OBJECT_KEY = "uploads/your-file.txt";
getObjectTagging(BUCKET_NAME, OBJECT_KEY);
```

Delete Object Tagging

功能说明

从指定的对象中删除整个标记集。要使用此操作，您必须具有执行s3:DeleteObjectTagging操作的权限。要删除特定对象版本的标签，请在请求中添加versionId查询参数。您将需要s3:DeleteObjectVersionTagging操作的权限。

方法原型

```
const input = { // DeleteObjectTaggingRequest
  Bucket: "STRING_VALUE", // required
  Key: "STRING_VALUE", // required
  VersionId: "STRING_VALUE"
};
const command = new DeleteObjectTaggingCommand(input);
const response = await client.send(command);
```

参数说明

参数名称	参数描述	类型	是否必须
Bucket	Bucket的名称	String	是
Key	Object的名称	String	是
VersionId	多版本场景下，指定对象的特定版本。不指定时，默认为最新版本的Id，如存在null版本号，并需对其操作，则需指定VersionId='null'。VersionId=''等效于指定最新版本的Id。	String	否

返回结果说明

```
{
    "$metadata": {
        "httpStatusCode": 204,
        "requestId": "tx000000000000005e13ce3-0068cd1953-8de19637-js10",
        "extendedRequestId": null,
        "cfId": null,
        "attempts": 1,
        "totalRetryDelay": 0
    }
}
```

示例

```
import { DeleteObjectTaggingCommand } from "@aws-sdk/client-s3";
/**
 * 删除指定对象的所有标签
 * @param {string} bucketName - 目标存储桶的名称
 * @param {string} objectKey - 目标对象的键（名称/路径）
 */
```

```
const deleteObjectTagging = async (bucketName, objectKey) => {
    try {
        const command = new DeleteObjectTaggingCommand({
            Bucket: bucketName,
            Key: objectKey,
        });

        await zos_client.send(command);

        console.log(`Success! All tags for object "${objectKey}" in bucket "${bucketName}"
have been deleted.`);

    } catch (error) {
        console.error("An error occurred while deleting tags:", error.name);
        if (error.name === 'NoSuchKey') {
            console.error(`Error: Object "${objectKey}" not found in bucket
"${bucketName}".`);
        } else if (error.name === 'NoSuchBucket') {
            console.error(`Error: Bucket "${bucketName}" does not exist.`);
        } else {
            console.error("Error message:", error.message);
        }
    }
};
const BUCKET_NAME = "your-bucket-name";
const OBJECT_KEY = "uploads/your-file.txt";
deleteObjectTagging(BUCKET_NAME, OBJECT_KEY);
```

Generate Presigned Url

功能说明

Generate Presigned Url请求生成一个临时的预签名的Url，没有权限访问集群的用户可以通过该Url访问集群，包括上传、下载文件等等。

方法原型

```
// get
import { getSignedUrl } from "@aws-sdk/s3-request-presigner";
const command = new GetObjectCommand(getObjectParams);
const url = await getSignedUrl(client, command, { expiresIn: expiration});
// put

import { getSignedUrl } from "@aws-sdk/s3-request-presigner";
const command = new PutObjectCommand(putObjectParams);
const presigned = getSignedUrl(client, command, {expiresIn: expiration});
```

参数说明

参数名称	参数描述	类型	是否必须
client	已初始化的 S3Client 实例	S3Client	是
command	希望为其生成预签名 URL 的具体操作命令实例。例如，要生成一个下载链接，您会传入一个 GetObjectCommand 实例；要生成一个上传链接，则传入 PutObjectCommand 实例。	Command	是
options	一个可选的配置对象，用于控制 URL 的行为。常用属性expiresIn，单位为秒	Dict	否

返回结果说明

```
// 签名地址
https://bucket-endpoint/key-name?AWSAccessKeyId=/*your
accesskey&Expires=1622796987&Signature=EW8brpCcL5E36UfqDXCbF%2FBuHTk%3D
```

返回结果	描述	类型
Url	集群生成的临时的Url	Dict

示例

```
import { GetObjectCommand } from "@aws-sdk/client-s3";
import { getSignedUrl } from "@aws-sdk/s3-request-presigner";
/**
 * 为指定的对象生成一个临时的、有时间限制的下载链接 (预签名 URL)
 * @param {string} bucketName - 目标存储桶的名称
 * @param {string} objectKey - 目标对象的键（名称/路径）
 * @param {number} expiresInSeconds - URL 的有效时间（单位：秒）
 */
const generatePresignedUrl = async (bucketName, objectKey, expiresInSeconds) => {
    try {
        // 创建一个 GetObject 命令，指定要访问的对象
        const command = new GetObjectCommand({
            Bucket: bucketName,
            Key: objectKey,
        });

        // 使用 getSignedUrl 方法创建预签名 URL
        const signedUrl = await getSignedUrl(zos_client, command, {
            expiresIn: expiresInSeconds,
        });

        console.log(`Success! Generated a presigned URL for "${objectKey}".`);
        console.log(` -> URL: ${signedUrl}`);
```

```
        console.log(` -> This URL is valid for ${expiresInSeconds} seconds.`);
        return signedUrl;
    } catch (error) {
        console.error("An error occurred while generating the presigned URL:", error.name);
        if (error.name === 'NoSuchKey') {
            console.error(`Error: Object "${objectKey}" not found in bucket
"${bucketName}".`);
        } else if (error.name === 'NoSuchBucket') {
            console.error(`Error: Bucket "${bucketName}" does not exist.`);
        } else {
            console.error("Error message:", error.message);
        }
    }
};
const BUCKET_NAME = "your-bucket-name";
const OBJECT_KEY = "documents/user-guide.pdf";
const EXPIRATION_IN_SECONDS = 3600; // URL 在 1 小时后过期
generatePresignedUrl(BUCKET_NAME, OBJECT_KEY, EXPIRATION_IN_SECONDS);
```

Generate Presigned Post

功能说明

Generate Presigned Post请求生成一个临时的预签名的Url，没有权限访问集群的用户可以通过该Url上传文件到集群(通过http的post方式)。预签名上传的对象，权限默认为私有。

方法原型

```
import { createPresignedPost } from "@aws-sdk/s3-presigned-post";
const { url, fields } = await createPresignedPost(client, {
  Bucket,
  Key,
  Expires: 600, //Seconds before the presigned post expires. 3600 by default.
});
```

参数说明

参数名称	参数描述	类型	是否必须
client	已初始化的 S3Client 实例	S3Client	是
options	一个配置对象，用于定义预签名 POST 的所有参数和约束	PresignedPostOptions	是
Bucket	上传到的目标存储桶的名称	String	是
Key	文件上传后在存储桶中保存的对象键（即文件名或路径）	String	是
Expires	预签名 POST 的有效时间，单位为秒	Int	是

返回结果说明


```
{
  "url": "https://bucket-test.jiangsu-10.zos.ctyun.cn/",
  "fields": {
    "bucket": "bucket-test",
    "X-Amz-Algorithm": "AWS4-HMAC-SHA256",
    "X-Amz-Credential": "L2XOUYJPWYLW31EFR9DO/20250920/us-east-1/s3/aws4_request",
    "X-Amz-Date": "20250920T081241Z",
    "key": "console.log.txt",
    "Policy": "********************",
    "X-Amz-Signature": "********************"
  },
  "$metadata": {
    "httpStatusCode": 204,
    "requestId": "tx000000000000005e13ce3-0068cd1953-8de19637-js10",
    "extendedRequestId": null,
    "cfId": null,
    "attempts": 1,
    "totalRetryDelay": 0
  }
}
```


返回结果	描述	类型
url	集群生成的临时的url	String
fields	服务端返回的数据，包含签名、policy等数据，需要在post文件上传的时候发送到服务端	Dict

示例

```
import { createPresignedPost } from "@aws-sdk/s3-presigned-post";

/**
 * 为 HTML 表单上传生成预签名 POST 数据
 * @param {string} bucketName - 目标存储桶的名称
 * @param {string} objectKey - 目标对象的键（名称/路径）
 * @param {number} expiresInSeconds - 预签名 POST 的有效时间（单位：秒）
 */
const generatePresignedPost = async (bucketName, objectKey, expiresInSeconds) => {
    try {
        const { url, fields } = await createPresignedPost(zos_client, {
            Bucket: bucketName,
            Key: objectKey,
            Expires: expiresInSeconds,
        });
        console.log(`Success! Generated presigned POST data for "${objectKey}".`);
        console.log(" -> The client should POST to this URL:");
        console.log(`    ${url}`);
```

```
        console.log(" -> The POST request must include these fields (as hidden form
inputs):");
        console.log(fields);
        return { url, fields };

    } catch (error) {
        console.error("An error occurred while generating the presigned POST:", error.name);
        if (error.name === 'NoSuchBucket') {
            console.error(`Error: Bucket "${bucketName}" does not exist.`);
        } else {
            console.error("Error message:", error.message);
        }
    }
};
const BUCKET_NAME = "your-upload-bucket-name";
const OBJECT_KEY = "user-uploads/form-upload.txt";
const EXPIRATION_IN_SECONDS = 300;
const result = generatePresignedPost(BUCKET_NAME, OBJECT_KEY, EXPIRATION_IN_SECONDS);
```

Put Object Copy

功能说明

Put Object Copy 请求实现将一个文件从源路径复制到目标路径。

方法原型

```
const input = { // CopyObjectRequest
  ACL: "private" || "public-read",
  Bucket: "STRING_VALUE", // required
  CopySource: "STRING_VALUE", // required
  Key: "STRING_VALUE", // required
  Metadata: { // Metadata
    "<keys>": "STRING_VALUE",
  },
  MetadataDirective: "COPY" || "REPLACE",
  TaggingDirective: "COPY" || "REPLACE",
  Tagging: "STRING_VALUE"
};
const command = new CopyObjectCommand(input);
const response = await client.send(command);
```

参数说明

参数名称	参数描述	类型	是否必须
ACL	object的ACL	String	否
Bucket	复制的目的Bucket	String	是

参数名称	参数描述	类型	是否必须
CopySource	复制的源 String格式：{bucket}/{key} or {bucket}/{key}?versionId={versionId}， Dict格式 {'Bucket': 'bucket', 'Key': 'key', 'VersionId': 'id'}	String 或 Dict	是
Key	复制的目的Key	String	是
Metadata	复制的目的Metadata	Dict	否
MetadataDirective	Metadata复制选项，'COPY' 复制源的Metadata或 'REPLACE'使用新指定的Metadata覆盖	String	否
TaggingDirective	Tagging 复制选项 'COPY' 复制源的Tag或 'REPLACE'使用新指定的Tag覆盖	String	否
Tagging	指定目的Object的Tag	String	否

返回结果说明

```
{
  "CopyObjectResult": {
    "LastModified": "2025-09-20T08:35:38.193Z",
    "ETag": "21fd59a7339f2e5f73c97039254fd784"
  },
  "$metadata": {
    "httpStatusCode": 204,
    "requestId": "tx000000000000005e13ce3-0068cd1953-8de19637-js10",
    "extendedRequestId": null,
    "cfId": null,
    "attempts": 1,
    "totalRetryDelay": 0
  }
}
```

返回结果	描述	类型
CopyObjectResult	复制的结果，包含复制的Object的ETAG和创建时间	Dict
ResponseMetadata	http请求返回数据，包括请求返回状态码及RequestID等等	Dict

示例

```
import { CopyObjectCommand } from "@aws-sdk/client-s3";
/**
 * 将一个对象从源位置复制到目标位置
 * @param {string} sourceBucket - 源存储桶的名称
 * @param {string} sourceKey - 源对象的键（名称/路径）
 * @param {string} destinationBucket - 目标存储桶的名称
 * @param {string} destinationKey - 目标对象的键（名称/路径）
 */
```

```
const copyObject = async (sourceBucket, sourceKey, destinationBucket, destinationKey) => {
    try {
        // **重要**: CopySource 的格式必须是 '源存储桶名称/源对象键'
        const copySource = `${sourceBucket}/${sourceKey}`;

        const command = new CopyObjectCommand({
            Bucket: destinationBucket,
            Key: destinationKey,
            CopySource: copySource,
            // (可选) 您还可以控制元数据和标签的复制方式
            // MetadataDirective: "REPLACE", // 使用新的元数据替换，而不是复制
            // TaggingDirective: "COPY", // 复制源对象的标签
        });

        const response = await zos_client.send(command);
        console.log(response);
        console.log(`Success! Object copied successfully.`);
        console.log(` -> Source: s3://${sourceBucket}/${sourceKey}`);
        console.log(` -> Destination: s3://${destinationBucket}/${destinationKey}`);
        console.log(` -> ETag of new object: ${response.CopyObjectResult?.ETag}`);

    } catch (error) {
        console.error("An error occurred during copy operation:", error.name);
        // 常见的错误包括源或目标存储桶/对象不存在
        if (error.name === 'NoSuchBucket') {
            console.error(`Error: One of the buckets does not exist.`);
        } else if (error.name === 'NoSuchKey') {
            console.error(`Error: The source object "${sourceKey}" was not found in bucket
"${sourceBucket}".`);
        } else {
            console.error("Error message:", error.message);
        }
    }
};
const SOURCE_BUCKET = "source-bucket-name";
const SOURCE_KEY = "path/to/original-file.txt";
const DESTINATION_BUCKET = "destination-bucket-name";
const DESTINATION_KEY = "backups/copied-file.txt";
copyObject(SOURCE_BUCKET, SOURCE_KEY, DESTINATION_BUCKET, DESTINATION_KEY);
```

Restore Object

功能说明

用于解冻归档对象数据。

方法原型

```
const input = { // RestoreObjectRequest
  Bucket: "STRING_VALUE", // required
  Key: "STRING_VALUE", // required
  VersionId: "STRING_VALUE",
  RestoreRequest: { // RestoreRequest
    Days: Number("int"),
    GlacierJobParameters: { // GlacierJobParameters
      Tier: "Standard" // required
    },
  }
};
const command = new RestoreObjectCommand(input);
const response = await client.send(command);
```

参数说明

参数名称	参数描述	类型	是否必须
Bucket	Bucket的名称	string	是
Key	要执行解冻操作的对象的名称	string	是
VersionId	对象的版本号	string	否
RestoreRequest	解冻操作的相关参数	Dict	是
Days	解冻之后副本的保留时间，目前支持范围为1-31天	integer	是
Tier	解冻模式，目前只支持Standard	String	是

返回结果说明

```
{
    "$metadata": {
    "httpStatusCode": 204,
    "requestId": "tx000000000000005e13ce3-0068cd1953-8de19637-js10",
    "extendedRequestId": null,
    "cfId": null,
    "attempts": 1,
    "totalRetryDelay": 0
  }
}
```

返回结果	描述	类型
metadata	http请求返回数据	Dict

示例

```
import { RestoreObjectCommand } from "@aws-sdk/client-s3";
/**
 * 启动一个归档对象的恢复请求
 * @param {string} bucketName - 存储桶的名称
 * @param {string} objectKey - 要恢复的归档对象的键（名称/路径）
 * @param {number} restorationDays - 恢复后的临时副本的有效期（天数）
 * @param {string} tier - 恢复类型。目前仅可选值为 'Standard';
 */
const restoreObject = async (bucketName, objectKey, restorationDays, tier = 'Standard') => {
    try {
        const command = new RestoreObjectCommand({
            Bucket: bucketName,
            Key: objectKey,
            RestoreRequest: {
                Days: restorationDays,
                GlacierJobParameters: {
                    Tier: tier,
                },
            },
        });
        const response = await zos_client.send(command);
        console.log(`Success! Restoration request for object "${objectKey}" has been
initiated.`);
        console.log(` -> HTTP Status Code: ${response.$metadata.httpStatusCode}`);
    } catch (error) {
        console.error("An error occurred during restoration request:", error.name);
        if (error.name === 'NoSuchKey') {
            console.error(`Error: Object "${objectKey}" not found in bucket
"${bucketName}".`);
        } else if (error.name === 'ObjectAlreadyInActiveTierError') {
            // 这是一个特殊的错误，表示对象已经是可访问状态（未归档），无需恢复。
            console.error(`Info: Object "${objectKey}" is already in an active tier and does
not need to be restored.`);
        } else {
            console.error("Error message:", error.message);
        }
    }
};
const BUCKET_NAME = "my-archive-bucket";
const ARCHIVED_OBJECT_KEY = "archive/very-old-file.zip";
const RESTORATION_DAYS = 7; // 将临时副本保留 7 天
const TIER = "Standard"; // 目标仅支持Standard
restoreObject(BUCKET_NAME, ARCHIVED_OBJECT_KEY, RESTORATION_DAYS, TIER);
```

分块上传操作

Create Multipart Upload

功能说明

Create Multipart Upload 请求实现初始化分片上传，成功执行此请求以后会返回 Upload ID 用于后续的 Upload Part 请求。对象权限默认为私有权限。

方法原型

```
const input = { // CreateMultipartUploadRequest
  ACL: "private" || "public-read" || "public-read-write" || "authenticated-read",
  Bucket: "STRING_VALUE", // required
  Key: "STRING_VALUE", // required
  Metadata: { // Metadata
    "<keys>": "STRING_VALUE",
  },
  StorageClass: "STRING_VALUE", // 'GLACIER'|'STANDARD_IA'|'STANDARD'
  Tagging: "STRING_VALUE",
  ContentType: "STRING_VALUE",
};
const command = new CreateMultipartUploadCommand(input);
const response = await client.send(command);
```

参数说明

参数名称	参数描述	类型	是否必须
ACL	要上传文件的ACL规则，支持private, public-read, public-read-write, authenticated-read	String	否
Bucket	文件要上传的Bucket名称	String	是
Key	要上传的文件名称	String	是
StorageClass	文件的存储级别，现在支持'GLACIER'|'STANDARD_IA'|'STANDARD'	String	否
Tagging	文件的标签，字符串要类似于这种Key1=Value1，而且只能有一个=符号，=之前为key，之后为value，所有即使有多个=也会解析为value	String	否
Metadata	与 S3 中的对象一起存储的元数据映射	Dict	否
ContentType	一种描述对象数据格式的标准 MIME 类型。	String	否

返回结果说明


```
{
  "$metadata": {
    "httpStatusCode": 200,
    "requestId": "tx000000000000000032439b1-0068ce7244-92cee370-js10",
    "extendedRequestId": null,
    "cfId": null,
    "attempts": 1,
    "totalRetryDelay": 0
  },
  "Bucket": "bucket-test",
  "Key": "videos/large-video-file.mp4",
  "UploadId": "2~n5kVNKNX_pqr7SeLWunnXupPlnsV4WA"
}
```


返回结果	描述	类型
Bucket	文件保存的Bucket名称	String
UploadId	分段上传ID	String
Key	文件在Bucket中保存的名称	String
metadata	http请求返回数据	Dict

示例

```
import { CreateMultipartUploadCommand } from "@aws-sdk/client-s3";
/**
 * 初始化一个分段上传任务
 * @param {string} bucketName - 存储桶的名称
 * @param {string} objectKey - 准备上传的对象的键（名称/路径）
 * @returns {Promise<string|undefined>} 成功时返回 UploadId，失败时返回 undefined
 */
const createMultipartUpload = async (bucketName, objectKey) => {
    try {
        const command = new CreateMultipartUploadCommand({
            Bucket: bucketName,
            Key: objectKey,
            // (可选) 您可以在这里指定 ACL, ContentType 等元数据
            // ACL: "public-read",
            // ContentType: "video/mp4",
        });
        const response = await zos_client.send(command);
        console.log(`Success! Multipart upload initiated for object "${objectKey}".`);
        console.log(` -> Upload ID: ${response.UploadId}`);
        // 返回 UploadId，以便后续操作使用
        return response.UploadId;
    } catch (error) {
        console.error("An error occurred during multipart upload initiation:", error.name);
        if (error.name === 'NoSuchBucket') {
```

```
            console.error(`Error: Bucket "${bucketName}" does not exist.`);
        } else {
            console.error("Error message:", error.message);
        }
    }
};
const BUCKET_NAME = "my-large-files-bucket";
const OBJECT_KEY = "videos/large-video-file.mp4";
const uploadId = await createMultipartUpload(BUCKET_NAME, OBJECT_KEY);
```

Upload Part

功能说明

Upload Part 请求实现在初始化以后的分块上传，支持的块的数量为 1 到 10000，除了最后一块，其他每块大小都必须大于或等于5M。在每次请求 Upload Part 时，需要携带 partNumber 和 uploadID，partNumber 为块的编号，支持乱序上传。partNumber 的取值需要从1开始。

方法原型

```
const input = { // UploadPartRequest
  Body: 'Buffer' | 'String', // required
  Bucket: "STRING_VALUE", // required
  Key: "STRING_VALUE", // required
  PartNumber: Number("int"), // required
  UploadId: "STRING_VALUE", // required
};
const command = new UploadPartCommand(input);
const response = await client.send(command);
```

参数说明

参数名称	参数描述	类型	是否必须
Body	要上传的文件数据	String	是
Bucket	文件要上传的Bucket名称	String	是
Key	要上传的文件名称	String	是
PartNumber	当前分段上传的编号	String	是
UploadId	分段上传的ID，在Create Multipart Upload中返回的UploadId	String	是

返回结果说明


```
{
   "$metadata": {
        "httpStatusCode": 200,
        "requestId": "tx0000000000000031ea432-0068ce7912-a0175bbc-js10",
        "extendedRequestId": null,
        "cfId": null,
        "attempts": 1,
        "totalRetryDelay": 0
    },
   "ETag": "b383264ada8e1932d7863560e79dcfbf"
}
```


返回结果	描述	类型
Etag	当前分段数据的Etag	String
ResponseMetadata	http请求返回数据	Dict
HTTPStatusCode	http请求状态码	Int

示例

```
import { UploadPartCommand } from "@aws-sdk/client-s3";
/**
 * 上传一个分段（Part）
 * @param {string} bucketName - 存储桶的名称
 * @param {string} objectKey - 对象的键（名称/路径）
 * @param {string} uploadId - 从 CreateMultipartUpload 命令获取的上传 ID
 * @param {number} partNumber - 当前分段的编号 (从 1 开始)
 * @param Buffer | String} body - 当前分段的数据内容
 * @returns {Promise<{ETag: string, PartNumber: number}|undefined>} 成功时返回包含 ETag 和
PartNumber 的对象，失败时返回 undefined
 */
const uploadPart = async (bucketName, objectKey, uploadId, partNumber, body) => {
    try {
        const command = new UploadPartCommand({
            Bucket: bucketName,
            Key: objectKey,
            UploadId: uploadId,
            PartNumber: partNumber,
            Body: body,
        });
        const response = await zos_client.send(command);
        console.log(`Success! Part #${partNumber} uploaded successfully.`);
        console.log(` -> ETag of part: ${response.ETag}`);
        // 返回 ETag 和 PartNumber，用于最后的 CompleteMultipartUpload 操作
        return { ETag: response.ETag, PartNumber: partNumber };
    } catch (error) {
        console.error(`An error occurred during part #${partNumber} upload:`, error.name);
        if (error.name === 'NoSuchUpload') {
```

```
            console.error(`Error: The multipart upload with ID "${uploadId}" does not
exist.`);
        } else if (error.name === 'NoSuchBucket') {
            console.error(`Error: Bucket "${bucketName}" does not exist.`);
        } else {
            console.error("Error message:", error.message);
        }
    }
};
import fs from "fs";
const UPLOAD_ID = "your-long-upload-id-string";
const BUCKET_NAME = "my-large-files-bucket";
const OBJECT_KEY = "videos/large-video-file.mp4";
const fileChunk = fs.readFileSync("my-local-file.txt"); // 临时替代一个chunk文件，作为分段
const partInfo = await uploadPart(BUCKET_NAME, OBJECT_KEY, UPLOAD_ID, 1, fileChunk);
```

Complete Multipart Upload

功能说明

Complete Multipart Upload 用来实现完成整个分块上传。当您已经使用 Upload Parts 上传所有块以后，你可以用该 API 完成上传。在使用该 API 时，您必须在 Body 中给出每一个块的 PartNumber 和 ETag，用来校验块的准确性。

方法原型

```
const input = { // CompleteMultipartUploadRequest
  Bucket: "STRING_VALUE", // required
  Key: "STRING_VALUE", // required
  MultipartUpload: { // CompletedMultipartUpload
    Parts: [ // CompletedPartList
      { // CompletedPart
        ETag: "STRING_VALUE",
        PartNumber: Number("int"),
      },
    ],
  },
  UploadId: "STRING_VALUE", // required
};
const command = new CompleteMultipartUploadCommand(input);
const response = await client.send(command);
```

参数说明

参数名称	参数描述	类型	是否必须
Bucket	文件要上传的Bucket名称	String	是
Key	要上传的文件名称	String	是
MultipartUpload	所有成功上传的分段的分段信息，包括分段对应Etag和分段号	Dict	是
Etag	分段对应的Etag	String	是

参数名称	参数描述	类型	是否必须
PartNumber	分段对应的分段编号	String	是
UploadId	分段上传的ID，在Create Multipart Upload中返回的UploadId	String	是

返回结果说明

```
{
  '$metadata': {
    httpStatusCode: 200,
    requestId: 'tx00000000000000000005-0068d0f145-73bfb-default',
    extendedRequestId: undefined,
    cfId: undefined,
    attempts: 1,
    totalRetryDelay: 0
  },
  Bucket: 'bucket-nodejs',
  ETag: '05676c14107010071925230430c37457-3',
  Key: 'largefile',
  Location: 'endpoint/bucket-nodejs/largefile'
}
```

返回结果	描述	类型
Etag	整体文件的Etag	String
Bucket	保存文件的Bucket名称	String
Location	文件具体位置	String
ResponseMetadata	http请求返回数据	Dict
HTTPStatusCode	http请求状态码	Int

示例

```
import { CompleteMultipartUploadCommand } from "@aws-sdk/client-s3";
import { zos_client } from './app.js';

/**
 * 完成分段上传
 * @param {string} bucketName - 存储桶的名称
 * @param {string} objectKey - 对象的键（名称/路径）
 * @param {string} uploadId - 从 CreateMultipartUpload 获取的上传 ID
 * @param {Array} parts - 已上传的分段信息，包含每个分段的 ETag 和 PartNumber
 * @returns {Promise<object|undefined>} 成功时返回 CompleteMultipartUpload 命令的响应，失败时返回
undefined
 */
const completeMultipartUpload = async (bucketName, objectKey, uploadId, parts) => {
    try {
        // 构建完成上传请求的参数
```

```
        const input = {
            Bucket: bucketName, // 存储桶名称
            Key: objectKey, // 对象的键
            UploadId: uploadId, // 上传 ID
            MultipartUpload: {
                Parts: parts.map(part => ({
                    ETag: part.ETag, // 分段的 ETag
                    PartNumber: part.PartNumber, // 分段的编号
                })),
            },
        };
        // 创建 CompleteMultipartUploadCommand 对象
        const command = new CompleteMultipartUploadCommand(input);
        // 发送请求并等待响应
        const response = await zos_client.send(command); // zos_client 是你的 S3 客户端实例
        console.log(`Success! Multipart upload completed for ${objectKey}.`);
        console.log(` -> Upload complete response:`, response);
        // 返回响应信息
        return response;
    } catch (error) {
        console.error(`An error occurred while completing the multipart upload:`,
error.name);
        if (error.name === 'NoSuchUpload') {
            console.error(`Error: The multipart upload with ID "${uploadId}" does not
exist.`);
        } else if (error.name === 'NoSuchBucket') {
            console.error(`Error: Bucket "${bucketName}" does not exist.`);
        } else {
            console.error("Error message:", error.message);
        }
    }
};
// 使用示例：完成上传过程
const UPLOAD_ID = "your-long-upload-id-string"; // 从 CreateMultipartUpload 获取的上传 ID
const BUCKET_NAME = "your-bucket-name"; // 存储桶名称
const OBJECT_KEY = "your-object-name"; // 文件对象键
// 假设我们有多个分段的 ETag 和 PartNumber
const uploadedParts = [
    { ETag: '"etag-for-part-1"', PartNumber: 1 },
    { ETag: '"etag-for-part-2"', PartNumber: 2 },
    { ETag: '"etag-for-part-3"', PartNumber: 3 },
];
// 调用完成上传的函数
const response = await completeMultipartUpload(BUCKET_NAME, OBJECT_KEY, UPLOAD_ID,
uploadedParts);
```

List Parts

功能说明

List Parts 用来查询特定分段上传中的已上传的分段的信息。

方法原型

```
const input = { // ListPartsRequest
  Bucket: "STRING_VALUE", // required
  Key: "STRING_VALUE", // required
  MaxParts: Number("int"),
  PartNumberMarker: "STRING_VALUE",
  UploadId: "STRING_VALUE", // required
};
const command = new ListPartsCommand(input);
const response = await client.send(command);
```

参数说明

参数名称	参数描述	类型	是否必须
Bucket	文件要上传的Bucket名称	String	是
Key	上传文件在集群中保存的文件名称	String	是
MaxParts	最多返回的分段数目	Int	否
PartNumberMarker	list的分段编号的起始编号，只有分段编号大于这个数字的分段信息才会返回--例："1"	String	否
UploadId	分段上传的ID，在Create Multipart Upload中返回的UploadId	String	是

返回结果说明

```
{
  '$metadata': {
    httpStatusCode: 200,
    requestId: 'tx000000000000000000000e-0068d0fd7a-73bfb-default',
    extendedRequestId: undefined,
    cfId: undefined,
    attempts: 1,
    totalRetryDelay: 0
  },
  Bucket: 'bucket-nodejs',
  IsTruncated: false,
  Key: 'largefile',
  MaxParts: 1000,
  NextPartNumberMarker: '3',
  Owner: { DisplayName: '', ID: '' },
  PartNumberMarker: '0',
```

```
  Parts: [
    {
      PartNumber: 1,
      LastModified: 2025-09-22T07:35:51.224Z,
      ETag: '"98c95dbb16e9f096305d7faf4115d7dd"',
      Size: 5242880
    },
    {
      PartNumber: 2,
      LastModified: 2025-09-22T07:35:51.331Z,
      ETag: '"474d889ecc10387016fb01810623913b"',
      Size: 5242880
    },
    {
      PartNumber: 3,
      LastModified: 2025-09-22T07:35:51.368Z,
      ETag: '"532601539f5804ef58326d39ea00575d"',
      Size: 225240
    }
  ],
  StorageClass: 'STANDARD',
  UploadId: '2~VtaZ1t4TJUkzqAMLd79nhm5BnuvBx-i'
}
```

返回结果	描述	类型
ResponseMetadata	http请求返回数据	Dict
HTTPStatusCode	http请求状态码	Int
Bucket	文件上传的Bcuket名称	String
NextPartNumberMarker	下一次list的时候的分段起始编号，主要用于截断返回时(也就是已上传的分段数目大于当前返回的分段数目)，作为下一次list的分段起始编号	Int
Parts	已经上传的分段信息	Dict
LastModified	该分段上次被修改的时间	Datetime
PartNumber	分段编号	Int
Etag	该分段数据对应Etag	String
Size	该分段数据大小，单位字节	Int
UploadId	分段上传的ID，在Create Multipart Upload中返回	String
StorageClass	分段上传的文件对应的存储级别	String
Key	分段上传的文件在集群中保存的名字	String
Owner	分段上传的文件所属用户	Dict
MaxParts	当前list最多返回的分段数目	Int

返回结果	描述	类型
IsTruncated	是否截断	Boolean
PartNumberMarker	当前list的分段起始编号	Int

示例

```
import { ListPartsCommand } from "@aws-sdk/client-s3";
/**
 * 列出指定上传任务的分段信息
 * @param {string} bucketName - 存储桶的名称
 * @param {string} objectKey - 对象的键（名称/路径）
 * @param {string} uploadId - 分段上传的上传 ID
 * @param {number} [maxParts=1000] - 每次请求返回的最大分段数量，默认为1000
 * @param {string} [partNumberMarker] - 用于分页的分段编号标记
 * @returns {Promise<object|undefined>} 返回 ListParts 命令的响应，失败时返回 undefined
 */
const listMultipartParts = async (bucketName, objectKey, uploadId, maxParts = 1000,
partNumberMarker = null) => {
    try {
        // 构建列出分段请求的参数
        const input = {
            Bucket: bucketName, // 存储桶名称
            Key: objectKey, // 对象的键
            UploadId: uploadId, // 上传 ID
            MaxParts: maxParts, // 每次请求返回的最大分段数量
            PartNumberMarker: partNumberMarker, // 分段编号标记，用于分页
        };

        // 创建 ListPartsCommand 对象
        const command = new ListPartsCommand(input);

        // 发送请求并等待响应
        const response = await zos_client.send(command); // zos_client 是你的 S3 客户端实例

        // 输出列出分段的响应
        console.log(`Successfully listed parts for ${objectKey}:`);
        console.log(response);

        // 返回响应信息
        return response;
    } catch (error) {
        console.error(`An error occurred while listing multipart parts:`, error.name);
        if (error.name === 'NoSuchUpload') {
            console.error(`Error: The multipart upload with ID "${uploadId}" does not
exist.`);
        } else if (error.name === 'NoSuchBucket') {
            console.error(`Error: Bucket "${bucketName}" does not exist.`);
        } else {
            console.error("Error message:", error.message);
        }
```

```
    }
};

// 使用示例：列出分段上传的分段信息
const UPLOAD_ID = "your-upload-id"; // 从 CreateMultipartUpload 获取的上传 ID
const BUCKET_NAME = "your-bucket-name"; // 存储桶名称
const OBJECT_KEY = "your-object-name"; // 文件对象键

// 调用列出分段的函数
const response = await listMultipartParts(BUCKET_NAME, OBJECT_KEY, UPLOAD_ID, 1000, "1");
```

Abort Multipart Upload

功能说明

Abort Multipart Upload 用来实现舍弃一个分块上传并删除已上传的块。当您调用 Abort Multipart Upload 时，如果有正在使用这个 Upload Parts 上传块的请求，则 Upload Parts 会返回失败。

方法原型

```
const input = { // AbortMultipartUploadRequest
  Bucket: "STRING_VALUE", // required
  Key: "STRING_VALUE", // required
  UploadId: "STRING_VALUE", // required
};
const command = new AbortMultipartUploadCommand(input);
const response = await client.send(command);
```

参数说明

参数名称	参数描述	类型	是否必须
Bucket	文件要上传的Bucket名称	String	是
Key	上传文件在集群中保存的文件名称	String	是
UploadId	分段上传的ID，在Create Multipart Upload中返回的UploadId	String	是

返回结果说明

```
{
  '$metadata': {
    httpStatusCode: 204,
    requestId: 'tx00000000000000000011-0068d0ffad-73bfb-default',
    extendedRequestId: undefined,
    cfId: undefined,
    attempts: 1,
    totalRetryDelay: 0
  }
}
```

返回结果	描述	类型
ResponseMetadata	http请求返回数据	Dict

示例

```
import { AbortMultipartUploadCommand } from "@aws-sdk/client-s3";
/**
 * 中止分段上传
 * @param {string} bucketName - 存储桶的名称
 * @param {string} objectKey - 对象的键（名称/路径）
 * @param {string} uploadId - 分段上传的上传 ID
 * @returns {Promise<object|undefined>} 成功时返回 AbortMultipartUpload 命令的响应，失败时返回
undefined
 */
const abortMultipartUpload = async (bucketName, objectKey, uploadId) => {
    try {
        // 构建中止上传请求的参数
        const input = {
            Bucket: bucketName, // 存储桶名称
            Key: objectKey, // 对象的键
            UploadId: uploadId, // 上传 ID
        };

        // 创建 AbortMultipartUploadCommand 对象
        const command = new AbortMultipartUploadCommand(input);

        // 发送请求并等待响应
        const response = await zos_client.send(command); // zos_client 是你的 S3 客户端实例

        console.log(`Successfully aborted multipart upload for ${objectKey}.`);
        console.log(` -> Abort response:`, response);

        // 返回响应信息
        return response;
    } catch (error) {
        console.error(`An error occurred while aborting the multipart upload:`, error.name);
        if (error.name === 'NoSuchUpload') {
            console.error(`Error: The multipart upload with ID "${uploadId}" does not
exist.`);
        } else if (error.name === 'NoSuchBucket') {
            console.error(`Error: Bucket "${bucketName}" does not exist.`);
        } else {
            console.error("Error message:", error.message);
        }
    }
};

// 使用示例：中止分段上传
const UPLOAD_ID = "your-upload-id"; // 从 CreateMultipartUpload 获取的上传 ID
const BUCKET_NAME = "your-bucket-name"; // 存储桶名称
const OBJECT_KEY = "your-object-name"; // 文件对象键
```

```
// 调用中止上传的函数
const response = await abortMultipartUpload(BUCKET_NAME, OBJECT_KEY, UPLOAD_ID);
```

List Multipart Uploads

功能说明

List Multiparts Uploads 用来查询正在进行中的分段上传，也就是已经 Created但是还没有Aaborted或者Completed的分段上传数据，单次最多列出 1000 个正在进行中的分段上传。

方法原型

```
const input = { // ListMultipartUploadsRequest
  Bucket: "STRING_VALUE", // required
  KeyMarker: "STRING_VALUE",
  MaxUploads: Number("int"),
  Prefix: "STRING_VALUE",
};
const command = new ListMultipartUploadsCommand(input);
const response = await client.send(command);
```

参数说明

参数名称	参数描述	类型	是否必须
Bucket	文件要上传的Bucket名称	String	是
KeyMarker	只有Key大于KeyMarker的分段上传数据才会返回	String	否
MaxUploads	单次最多返回的分段上传数据，大小是1-1000，超过1000的数据会被视为1000	Int	否
Prefix	Key的前缀，只有以Prefix为开头的Key才会被返回	String	否

返回结果说明

```
{
  '$metadata': {
    httpStatusCode: 200,
    requestId: 'tx00000000000000000016-0068d101d9-73bfb-default',
    extendedRequestId: undefined,
    cfId: undefined,
    attempts: 1,
    totalRetryDelay: 0
  },
  Bucket: 'bucket-nodejs',
  IsTruncated: false,
  MaxUploads: 1000,
  NextKeyMarker: 'small/file',
  NextUploadIdMarker: '2~JbiBiA1FtyNomxtx4J_pFgbECvmbOGn',
```

```
  Uploads: [
    {
      UploadId: '2~jIKM3eXPsEMjKjOHXb2v-vmd9RQv7mB',
      Key: 'largefile1',
      Initiated: 2025-09-22T07:58:19.082Z,
      StorageClass: 'STANDARD',
      Owner: [Object],
      Initiator: [Object]
    },
    {
      UploadId: '2~v4kozWqIyX2F_sKDmP6UPsCTmO9f-In',
      Key: 'largefile2',
      Initiated: 2025-09-22T07:58:27.368Z,
      StorageClass: 'STANDARD',
      Owner: [Object],
      Initiator: [Object]
    },
    {
      UploadId: '2~JbiBiA1FtyNomxtx4J_pFgbECvmbOGn',
      Key: 'small/file',
      Initiated: 2025-09-22T07:58:52.857Z,
      StorageClass: 'STANDARD',
      Owner: [Object],
      Initiator: [Object]
    }
  ]
}
```

返回结果	描述	类型
ResponseMetadata	http请求返回数据	Dict
Uploads	分段上传数据	Dict
Initiator	该分段上传的创建者	Dict
Inittited	该分段上传创建的时间	datetime
UploadId	该分段上传的UploadId	String
StorageClass	存储级别	String
Key	分段上传的Key	String
Owner	分段上传的所有者	Dict

示例

```
import { ListMultipartUploadsCommand } from "@aws-sdk/client-s3";
/**
 * 列出所有分段上传任务
 * @param {string} bucketName - 存储桶的名称
 * @param {string} prefix - 文件前缀（可选）
```

```
 * @param {string} keyMarker - 键标记（可选）
 * @param {number} maxUploads - 最大上传数（可选）
 * @returns {Promise<object|undefined>} 成功时返回 ListMultipartUploads 命令的响应，失败时返回 undefined
 */
const listMultipartUploads = async (bucketName, prefix = '', keyMarker = '', maxUploads = 1000) => {
    try {
        // 构建列出上传任务的请求参数
        const input = {
            Bucket: bucketName, // 存储桶名称
            Prefix: prefix, // 文件前缀
            KeyMarker: keyMarker, // 键标记
            MaxUploads: maxUploads, // 最大上传数
        };

        // 创建 ListMultipartUploadsCommand 对象
        const command = new ListMultipartUploadsCommand(input);
        // 发送请求并等待响应
        const response = await zos_client.send(command);
        console.log(`Successfully listed multipart uploads for ${bucketName}.`);
        console.log(` -> List response:`, response);
        // 返回响应信息
        return response;
    } catch (error) {
        console.error(`An error occurred while listing multipart uploads:`, error.name);
        if (error.name === 'NoSuchBucket') {
            console.error(`Error: Bucket "${bucketName}" does not exist.`);
        } else {
            console.error("Error message:", error.message);
        }
    }
};

// 使用示例：列出分段上传任务
const BUCKET_NAME = "your-bucket-name"; // 存储桶名称
const PREFIX = ""; // 可选，文件前缀（如："documents/"）
const KEY_MARKER = ""; // 可选，键标记，分页标记
const MAX_UPLOADS = 1000; // 每次列出的最大上传任务数

// 调用列出分段上传任务的函数
const response = await listMultipartUploads(BUCKET_NAME, PREFIX, KEY_MARKER, MAX_UPLOADS);
```