天翼云媒体存储

go SDK 使用指导书

天翼云科技有限公司

2026年3月27日

天翼云媒体存储go SDK使用指导书

环境配置

Go语言环境安装

需要安装1.5或更新的Go语言版本，可以终端或者命令窗口执行命令`go version`查看当前安装的Go语言版本。推荐使用1.11或更新的Go语言版本。如果需要升级安装最新的Go语言版本，请访问 https://golang.google.cn/dl/ 。

配置go mod

Windows

打开命令窗口执行

```
`$ go env -w GO111MODULE=on`
`$ go env -w GOPROXY=https://goproxy.cn,direct`
```

macOS或Linux

打开终端并执行

```
`$ export GO111MODULE=on`
`$ export GOPROXY=https://goproxy.cn`
```

或者

```
`$ echo "export GO111MODULE=on" >> ~/.profile`
`$ echo "export GOPROXY=https://goproxy.cn" >> ~/.profile`
`$ source ~/.profile`
```

初始化SDK

下载SDK

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

获取访问密钥

AccessKey（AK）和SecretAccessKey（SK）是用户访问媒体存储服务的密钥，密钥的管理和获取方式请查阅 天翼云媒体存储 密钥管理 。

获取Endpoint

EndPoint的获取方式请查阅天翼云媒体存储 基础信息查看。

创建工程

下面过程以实现列出媒体存储服务中的bucket功能为例，说明了如何使用媒体存储go-sdk进行应用开发。

1. 初始化go mod

   新建一个项目文件夹go-sdk-demo，在该文件夹路径下执行命令`go mod init {moduleName}`生成go.mod文件，例如：

   ```
   go mod init sdk-demo
   ```

2. 导入sdk代码

   将下载的媒体存储go-sdk代码解压，获得vendor文件夹。在项目文件夹下执行命令导入依赖：

   ```
   go mod edit -require=github.com/aws/aws-sdk-go@v1.35.5
   ```

   执行命令将依赖替换为本地包：

   ```
   go mod edit -replace=github.com/aws/aws-sdk-go@v1.35.5={path of
   vendor}/github.com/aws/aws-sdk-go
   ```

   其中{path of vendor}是vendor文件夹在本地的路径，例如：

   ```
   go mod edit -replace=github.com/aws/aws-sdk-
   go@v1.35.5=C:/Users/admin/Desktop/vendor/github.com/aws/aws-sdk-go
   ```

   下载sdk所需依赖：

   ```
   go mod download github.com/jmespath/go-jmespath
   ```

3. 新建一个demo.go文件，其内容为：

   ```
   package main

   import (
       "fmt"
       "github.com/aws/aws-sdk-go/aws"
       "github.com/aws/aws-sdk-go/aws/credentials"
       "github.com/aws/aws-sdk-go/aws/session"
       "github.com/aws/aws-sdk-go/service/s3"
   )

   const (
       Endpoint        = "<your-endpoint>"
       AccessKey       = "<your-access-key>"
       SecretAccessKey = "<your-secret-access-key>"
   )

   func BuildClient() *s3.S3 {
       conf := &aws.Config{
           Endpoint:         aws.String(Endpoint),
           S3ForcePathStyle: aws.Bool(false),
   ```

```
        DisableSSL:       aws.Bool(true),
        Credentials:      credentials.NewStaticCredentials(AccessKey,
SecretAccessKey, ""),
        LogLevel:         aws.LogLevel(aws.LogDebug)}
    sess :=
session.Must(session.NewSessionWithOptions(session.Options{Config: *conf}))
    svc := s3.New(sess)
    return svc
}

func main() {
    svc := BuildClient()
    result, err := svc.ListBuckets(&s3.ListBucketsInput{})
    if err != nil {
        fmt.Printf("fail to list buckets. %v\n", err)
    } else {
        fmt.Println(result)
    }
}
```

4. 执行命令以下运行程序并查看结果。

```
go mod tidy
go mod vendor
go run ./demo.go
```

整个项目的内容层次如下：

```
go-sdk-demo/
    ├── demo.go
    ├── go.mod
    ├── go.sum
    └── vendor
```

设置service_client

使用SDK访问对象存储服务，首先需要提供正确的AccessKey和SecretAccessKey以及服务端地址EndPoint，用于设置一个service client。

创建session

配置service client首先需要创建一个session， session包含了service client的配置信息，例如AccessKey、SecretAccessKey以及发送请求的附加信息等。一个session可以被用于创建多个service client。

代码示例：

```
import (
    "github.com/aws/aws-sdk-go/aws"
    "github.com/aws/aws-sdk-go/aws/credentials"
    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/s3"
)
```

```
const (
    Endpoint        = "<your-endpoint>"
    AccessKey       = "<your-access-key>"
    SecretAccessKey = "<your-secret-access-key>"
)

// BuildSession 创建并返回一个session
func BuildSession() *session.Session {
    conf := &aws.Config{
        Endpoint:         aws.String(Endpoint),
        // Set this to `true` to force the request to use path-style addressing,
        // i.e., `http://gzoss.xstore.ctyun.cn/BUCKET/KEY`. By default, the S3
client
        // will use virtual hosted bucket addressing when possible
        // (`http://BUCKET.gdoss.xstore.ctyun.cn/KEY`)
        S3ForcePathStyle: aws.Bool(false),
        //Set this to `true` to disable SSL when sending requests. Defaults to
`false`
        DisableSSL:       aws.Bool(true),
        Credentials:      credentials.NewStaticCredentials(AccessKey,
SecretAccessKey, ""),
        LogLevel:         aws.LogLevel(aws.LogDebug)}
    sess := session.Must(session.NewSessionWithOptions(session.Options{Config:
*conf}))
    return sess
}
```

创建service client

代码示例：

```
// BuildClient 创建并返回一个service client
func BuildClient() *s3.S3 {
    conf := &aws.Config{
        Endpoint:         aws.String(Endpoint),
        S3ForcePathStyle: aws.Bool(false),
        DisableSSL:       aws.Bool(true),
        Credentials:      credentials.NewStaticCredentials(AccessKey,
SecretAccessKey, ""),
        LogLevel:         aws.LogLevel(aws.LogDebug)}
    sess := session.Must(session.NewSessionWithOptions(session.Options{Config:
*conf}))
    svc := s3.New(sess)
    return svc
}
```

桶相关接口

创建桶

功能说明

用户通过创建桶操作创建桶（bucket）后才能访问存储资源，每个用户可以拥有多个桶。桶的名称在媒体存储范围内必须是全局唯一的，一旦创建之后无法修改名称。桶的创建者默认是桶的拥有者，对桶拥有FULL_CONTROL权限，可以通过设置参数的方式为其他用户配置创建桶的权限。桶的命名规范如下：

- 使用字母、数字和短横线（-）；
- 以小写字母或者数字开头和结尾；
- 长度在3-63字节之间。

代码示例

```
func CreateBucket(svc *s3.S3) {
    bucketName := "<your-bucket-name>"
    // 创建桶
    createBucketInput := &s3.CreateBucketInput{
        Bucket:           aws.String(bucketName),
        GrantFullControl: aws.String("emailAddress=<user@example.com>"),
    }
    createBucketOutput, err := svc.CreateBucket(createBucketInput)
    if err != nil {
        fmt.Printf("Unable to create bucket %s, %v\n", bucketName, err)
    }
    // 等待桶创建
    fmt.Printf("Waiting for bucket %s to be created...\n", bucketName)
    err = svc.WaitUntilBucketExists(&s3.HeadBucketInput{
        Bucket: aws.String(bucketName),
    })
    if err != nil {
        fmt.Printf("Error occurred while waiting for bucket to be created, %v",
createBucketOutput)
    }

    fmt.Printf("Bucket %s successfully created\n", bucketName)
}
```

通过CreateBucketRequest操作：

CreateBucketRequest操作首先生成一个"request.Request"对象，该对象是一个执行CreateBucket操作的请求。通过调用Request对象的Send方法完成创建bucket的操作。该方法可以生成定制化的请求，例如自定义请求头部请求超时重试设置。

```
func CreateBucketRequest(svc *s3.S3) {
    createBucketInput := &s3.CreateBucketInput{
        Bucket: aws.String("<your-bucket-name>"),
    }
    req, createBucketOutput := svc.CreateBucketRequest(createBucketInput)

    err := req.Send()
    if err != nil {
        fmt.Printf("fail to create bucket. %v\n", err)
    } else {
        fmt.Println(createBucketOutput)
    }
}
```

请求参数

CreateBucketInput可设置的参数如下：

参数	类型	说明	是否必要
ACL	*string	配置创建bucket预定义的标准ACL信息，例如private，public-read等	否
Bucket	*string	创建bucket的名称	是
GrantFullControl	*string	用于自定义用户对此bucket的READ、WRITE、READ_ACP、WRITE_ACP权限信息	否
GrantRead	*string	用于自定义用户对此bucket的READ权限信息	否
GrantReadACP	*string	用于自定义用户对此bucket的READ_ACP权限信息	否
GrantWrite	*string	用于自定义用户对此bucket的WRITE权限信息	否
GrantWriteACP	*string	用于自定义用户对此bucket的WRITE_ACP权限信息	否

获取桶列表

功能说明

获取桶列表可以显示用户全部可用的桶。

代码示例

```
func ListBuckets(svc *s3.S3) {
    listBucketsInput := &s3.ListBucketsInput{}
    listBucketsOutput, err := svc.ListBuckets(listBucketsInput)
    if err != nil {
        fmt.Println("Failed to list buckets", err)
    }
    for _, b := range listBucketsOutput.Buckets {
        fmt.Printf("* %s created on %s\n", aws.StringValue(b.Name),
aws.TimeValue(b.CreationDate))
    }
}
```

通过ListBucketsRequest操作：

ListBucketsRequest操作首先生成一个"request.Request"对象，该对象是一个执行ListBucket操作的请求。通过调用Request对象的Send方法完成列出可用桶的操作。该方法可以生成定制化的请求，例如自定义请求头部请求超时重试设置。

```
func ListBucketRequest(svc *s3.S3) {
    listBucketsInput := &s3.ListBucketsInput{}
    req, listBucketsOutput := svc.ListBucketsRequest(listBucketsInput)

    err := req.Send()
    if err != nil {
        fmt.Printf("fail to list bucket. %v\n", err)
    } else {
        fmt.Println(listBucketsOutput)
    }
}
```

返回结果

ListBucketsOutput返回的属性如下：

属性名	类型	说明
Buckets	[]*Bucket	bucket信息的数组，包含了每个bucket的名字和创建时间
Owner	*Owner	bucket的拥有者信息

判断桶是否存在

功能说明

通过请求返回结果中的HTTP状态码快速判断桶是否存在，返回状态码为200表示桶存在，返回状态码为404表示桶不存在。

代码示例

```
func HeadBucket(svc *s3.S3) {
    headBucketInput := &s3.HeadBucketInput{
        Bucket: aws.String("<your-bucket-name>"),
    }
    _, err := svc.HeadBucket(headBucketInput)

    if err != nil {
        fmt.Printf("fail to head bucket, %v\n", err)
        return
    }
}
```

通过HeadBucketRequest操作：

HeadBucketRequest操作首先生成一个"request.Request"对象，该对象是一个执行HeadBucket操作的请求。通过调用Request对象的Send方法完成判断桶是否存在的操作。该方法可以生成定制化的请求，例如自定义请求头部请求超时重试设置。

```
func HeadBucketRequest(svc *s3.S3) {
    headBucketInput := &s3.HeadBucketInput{
        Bucket: aws.String("<your-bucket-name>"),
    }
    req, _ := svc.HeadBucketRequest(headBucketInput)

    err := req.Send()
    if err != nil {
        fmt.Printf("fail to head bucket. %v\n", err)
    }
}
```

请求参数

HeadBucketInput可设置的参数如下：

参数	类型	说明	是否必要
Bucket	*string	bucket的名称	是

删除桶

功能说明

用于删除桶（bucket），删除一个桶前，需要先删除该桶中的全部对象（包括object versions和delete markers）。

代码示例

```
func DeleteBucket(svc *s3.S3) {
    deleteBucketInput := &s3.DeleteBucketInput{
        Bucket: aws.String("<your-bucket-name>"),
    }

    _, err := svc.DeleteBucket(deleteBucketInput)
    if err != nil {
        fmt.Printf("fail to delete bucket. %v\n", err)
    }
}
```

通过DeleteBucketRequest操作：

DeleteBucketRequest操作首先生成一个"request.Request"对象，该对象是一个执行DeleteBucket操作的请求。通过调用Request对象的Send方法完成删除bucket的操作。该方法可以生成定制化的请求，例如自定义请求头部请求超时重试设置。

```
func DeleteBucketRequest(svc *s3.S3) {
    deleteBucketInput := &s3.DeleteBucketInput{
        Bucket: aws.String("<your-bucket-name>"),
    }

    req, _ := svc.DeleteBucketRequest(deleteBucketInput)

    err := req.Send()
    if err != nil {
        fmt.Printf("fail to delete bucket. %v\n", err)
    }
}
```

请求参数

DeleteBucketInput可设置的参数如下：

参数	类型	说明	是否必要
Bucket	*string	bucket的名称	是

列举桶内对象

功能说明

列举桶内对象操作用于列出桶中的全部对象，该操作返回最多1000个对象信息，可以通过设置过滤条件来列出桶中符合特定条件的对象信息。

代码示例

```
func ListObjects(svc *s3.S3) {
    listObjectsInput := &s3.ListObjectsInput{
        Bucket: aws.String("<your-bucket-name>"),
    }

    ListObjectsOutput, err := svc.ListObjects(listObjectsInput)
    if err != nil {
        fmt.Printf("fail to list objects of bucket. %v\n", err)
    }
    for _, object := range ListObjectsOutput.Contents {
        fmt.Println(*object.Key)
    }
}
```

通过ListObjectsRequest操作：

ListObjectsRequest操作首先生成一个"request.Request"对象，该对象是一个执行ListObjects操作的请求。通过调用Request对象的Send方法完成列出bucket中对象的操作。该方法可以生成定制化的请求，例如自定义请求头部请求超时重试设置。

```
func ListObjectRequest(svc *s3.S3) {
    listObjectsInput := &s3.ListObjectsInput{
        Bucket: aws.String("<your-bucket-name>"),
    }
    req, listObjectsOutput := svc.ListObjectsRequest(listObjectsInput)

    err := req.Send()
    if err != nil {
        fmt.Printf("fail to list objects. %v\n", err)
    } else {
        fmt.Println(listObjectsOutput)
    }
}
```

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

```
func ListAllObjects(svc *s3.S3) {
    var nextMarker *string

    for {
        listObjectsInput := &s3.ListObjectsInput{
            Bucket: aws.String("<your-bucket-name>"),
            Marker: nextMarker,
        }

        req, listObjectsOutput := svc.ListObjectsRequest(listObjectsInput)
        err := req.Send()
        if err != nil {
            fmt.Printf("Failed to list objects: %v\n", err)
            return
        }

        if listObjectsOutput.Contents != nil {
            for _, obj := range listObjectsOutput.Contents {
                fmt.Println("Object Key:", *obj.Key)
            }
        }

        if listObjectsOutput.IsTruncated != nil &&
*listObjectsOutput.IsTruncated {
            nextMarker = listObjectsOutput.NextMarker
        } else {
            break
        }
    }
}
```

请求参数

ListObjectsInput可设置的参数如下：

参数	类型	说明	是否必要
Bucket	*string	bucket的名称	是
Delimiter	*string	与Prefix参数一起用于对对象key进行分组的字符。所有key包含指定的Prefix且第一次出现Delimiter字符的对象作为一组。如果没有指定Prefix参数，按Delimiter对所有对象key进行分割，多个对象分割后从对象key开始到第一个Delimiter之间相同的部分形成一组	否
Marker	*string	指定一个标识符，返回的对象的key将是按照字典顺序排序后位于该标识符之后的所有对象	否
MaxKeys	*int64	设置response中返回object key的数量，默认值和最大值均为1000	否
Prefix	*string	限定返回对象的key必须以Prefix作为前缀	否

返回结果

ListObjectsOutput返回的属性如下：

属性名	类型	说明
CommonPrefixes	[]*CommonPrefix	当请求中设置了Delimiter和Prefix属性时，所有包含指定的Prefix且第一次出现Delimiter字符的对象key作为一组
Contents	[]*Object	对象数据，每个对象包含了Entity Tag、Key、LastModifiedTime、Owner和Size等信息
Delimiter	*string	与请求中设置的Delimiter一致
IsTruncated	*bool	当为false时表示返回结果中包含了全部符合本次请求查询条件的对象信息，否则只返回了数量为MaxKeys个的对象信息
Marker	*string	与请求中设置的Marker一致
MaxKeys	*int64	本次返回结果中包含的对象数量的最大值
Name	*string	执行本操作的桶名称
NextMarker	*string	当返回结果中的IsTruncated为true时，可以使用NextMarker作为下次查询的Marker，继续查询出下一部分的对象信息

属性名	类型	说明
Prefix	*string	与请求中设置的Prefix一致

列举桶内多版本对象

功能说明

列举桶内多版本对象操作可以获取关于对象版本信息的元数据，执行该操作需要对桶有READ权限。

代码示例

```
// 列出对象的版本信息
func ListObjectVersion(svc *s3.S3) {
    listObjectVersionsInput := &s3.ListObjectVersionsInput{
        Bucket: aws.String("<your-bucket-name>"),
        Prefix: aws.String("<your-object-key>"),
    }

    listObjectVersionsOutput, err :=
svc.ListObjectVersions(listObjectVersionsInput)
    if err != nil {
        fmt.Printf("fail to list object versions. %v\n", err)
        return
    }
    fmt.Println(listObjectVersionsOutput)
}
```

通过ListObjectVersionsRequest操作

ListObjectVersionsRequest操作首先生成一个"request.Request"对象，该对象是一个执行ListObjectVersions操作的请求。通过调用Request对象的Send方法来获取对象版本信息 。该方法可以生成定制化的请求，例如自定义请求头部请求超时重试设置。

```
func ListObjectVersionsRequest(svc *s3.S3) {
    listObjectVersionsInput := &s3.ListObjectVersionsInput{
        Bucket:    aws.String("<your-bucket-name>"),
        KeyMarker: aws.String("<your-object-key>"),
    }
    req, listObjectVersionsOutput :=
svc.ListObjectVersionsRequest(listObjectVersionsInput)

    err := req.Send()
    if err != nil {
        fmt.Printf("fail to list object versions. %v\n", err)
    } else {
        fmt.Println(listObjectVersionsOutput)
    }
}
```

请求参数

ListObjectVersionsInput可设置的参数如下：

参数	类型	说明	是否必要
Bucket	*string	查询版本信息的对象所在的桶的名称	是
Delimiter	*string	与Prefix参数一起用于对对象key进行分组的字符。所有key包含指定的Prefix且第一次出现Delimiter字符之间的对象作为一组。如果没有指定Prefix参数，按Delimiter对所有对象key进行分割，多个对象分割后从对象key开始到第一个Delimiter之间相同的部分形成一组	否
KeyMarker	*string	指定一个标识符，返回的对象的key将是按照字典顺序排序后位于该标识符之后的所有对象	否
MaxKeys	*int64	设置查询结果中返回对象key的数量，默认值和最大值均为1000	否
Prefix	*string	限定返回对象的key必须以Prefix作为前缀	否
VersionIdMarker	*string	列举多版本时指定对象的起始版本Id	否

返回结果

ListObjectVersionsOutput返回的属性如下：

属性名	类型	说明
CommonPrefixes	[]*CommonPrefix	当请求中设置了Delimiter和Prefix属性时，所有包含指定的Prefix且第一次出现Delimiter字符的对象key作为一组
DeleteMarkers	[]*DeleteMarkerEntry	对象删除标记信息数组，数组中每一项包含了是否为最新版本、对象key、最新修改时间、拥有者和版本Id等信息
Delimiter	*string	与请求中设置的Delimiter一致
IsTruncated	*bool	当为false时表示返回结果中包含了全部符合本次请求查询条件的对象版本信息，否则只返回了MaxKeys个对象版本信息
KeyMarker	*string	与请求中设置的KeyMarker一致
MaxKeys	*int64	本次返回结果中包含的对象版本信息数量的最大值
Name	*string	执行本操作的桶名称
NextKeyMarker	*string	当返回结果中的IsTruncated为true时，可以使用NextKeyMarker作为下次查询请求中的KeyMarker，继续查询出下一部分的对象版本信息

属性名	类型	说明
NextVersionIdMarker	*string	当返回结果中的IsTruncated为true时，可以使用NextVersionIdMarker作为下次查询请求中的VersionIdMarker，继续查询出下一部分的对象版本信息
Prefix	*string	与请求中设置的Prefix一致
VersionIdMarker	*string	表示列举多版本对象的起始版本Id，与请求中的该参数对应
Versions	[]*ObjectVersion	对象版本信息的数组，数组中每一项包含了对象的Entity Tag、是否为最新版本、对象key、最新修改时间、拥有者、大小、存储类型和版本Id的信息

设置桶ACL

功能说明

设置桶ACL操作可以通过access control list（ACL）设置一个桶的访问权限。用户在设置桶的ACL之前需要具备WRITE_ACP 权限。

Bucket的权限说明：

权限类型	说明
READ	可以对bucket进行list操作
READ_ACP	可以读取bucket的ACL信息。bucket的拥有者默认具有bucket的READ_ACP权限
WRITE	可以在bucket中创建对象，修改原有对象数据和删除对象
WRITE_ACP	可以修改bucket的ACL信息，授予该权限相当于授予FULL_CONTROL权限，因为具有WRITE_ACP权限的用户可以配置bucket的任意权限。bucket的拥有者默认具有bucket的WRITE_ACP权限
FULL_CONTROL	同时授予READ、READ_ACP、WRITE和WRITE_ACP权限

代码示例

```
func PutBucketAcl(svc *s3.S3) {
    bucket := "<your-bucket-name>"
    permission := "READ_ACP" // FULL_CONTROL、WRITE、WRITE_ACP、READ、READ_ACP
    granteeDisplayName := "<your-display-name>"
    granteeId := "<your-user-id>"
    userType := "CanonicalUser"
    // 获取当前ACL
    currentACL, err := svc.GetBucketAcl(&s3.GetBucketAclInput{Bucket:
aws.String(bucket)})
    if err != nil {
        fmt.Printf("fail to get acl of bucket, %v\n", err)
        os.Exit(1)
```

```
    }
    // 创建一个新的授权信息
    var newGrantee = s3.Grantee{
        Type:         aws.String(userType),
        DisplayName: aws.String(granteeDisplayName),
        ID:           aws.String(granteeId),
    }
    var newGrant = s3.Grant{Grantee: &newGrantee, Permission: &permission}

    grants := currentACL.Grants
    owner := *currentACL.Owner.DisplayName
    ownerId := *currentACL.Owner.ID
    grants = append(grants, &newGrant)
    // 添加一个授权信息
    putBucketAclInput := &s3.PutBucketAclInput{
        Bucket: &bucket,
        AccessControlPolicy: &s3.AccessControlPolicy{
            Grants: grants,
            Owner: &s3.Owner{
                DisplayName: &owner,
                ID:           &ownerId,
            },
        },
    }

    _, err = svc.PutBucketAcl(putBucketAclInput)
    if err != nil {
        fmt.Printf("fail to put acl to bucket. %v\n", err)
        os.Exit(1)
    }
    fmt.Println("You gave user with ", permission, "permission to bucket ",
bucket)
}
```

通过PutBucketAclRequest操作：

PutBucketAclRequest操作首先生成一个"request.Request"对象，该对象是一个执行PutBucketAcl操作的请求。通过调用Request对象的Send方法完成设置bucket ACL信息的操作。该方法可以生成定制化的请求，例如自定义请求头部请求超时重试设置。

```
func PutBucketAclRequest(svc *s3.S3) {
    bucket := "<your-bucket-name>"
    permission := "READ_ACP" // FULL_CONTROL、WRITE、WRITE_ACP、READ、READ_ACP
    granteeDisplayName := "<your-display-name>"
    granteeId := "<your-user-id>"
    userType := "CanonicalUser"
    // 获取当前ACL
    currentACL, err := svc.GetBucketAcl(&s3.GetBucketAclInput{Bucket:
aws.String(bucket)})
    if err != nil {
        fmt.Printf("fail to get acl of bucket, %v\n", err)
        os.Exit(1)
    }
    // 创建一个新的授权信息
    var newGrantee = s3.Grantee{
        Type:         aws.String(userType),
        DisplayName: aws.String(granteeDisplayName),
```

```
        ID:          aws.String(granteeId),
    }
    var newGrant = s3.Grant{Grantee: &newGrantee, Permission: &permission}

    grants := currentACL.Grants
    owner := *currentACL.Owner.DisplayName
    ownerId := *currentACL.Owner.ID
    grants = append(grants, &newGrant)
    // 添加一个授权信息
    putBucketAclInput := &s3.PutBucketAclInput{
        Bucket: &bucket,
        AccessControlPolicy: &s3.AccessControlPolicy{
            Grants: grants,
            Owner: &s3.Owner{
                DisplayName: &owner,
                ID:          &ownerId,
            },
        },
    }

    req, _ := svc.PutBucketAclRequest(putBucketAclInput)

    err = req.Send()
    if err != nil {
        fmt.Printf("fail to put bucket acl. %v\n", err)
    }
}
```

请求参数

PutBucketAclInput可设置的参数如下：

参数	类型	说明	是否必要
ACL	*string	配置此bucket预定义的标准ACL信息，例如private，public-read等	否
AccessControlPolicy	*AccessControlPolicy	配置该bucket对于每个用户的ACL授权信息	否
Bucket	*string	bucket的名称	是
GrantFullControl	*string	用于自定义用户对此bucket的FULL_CONTROL权限信息	否
GrantRead	*string	用于自定义用户对此bucket的READ权限信息	否
GrantReadACP	*string	用于自定义用户对此bucket ACL的READ权限信息	否
GrantWrite	*string	用于自定义用户对此bucket的WRITE权限信息	否

参数	类型	说明	是否必要
GrantWriteACP	*string	用于自定义用户对此bucket ACL的WRITE权限信息	否

用户可以通过ACL参数设置bucket的访问权限，也可以通过GrantFullControl、GrantRead、GrantReadACP、GrantWrite、GrantWriteACP等参数设置bucket的访问权限，但二者只能同时使用一种方式。

获取桶ACL

功能说明

获取桶ACL操作用户获取bucket的access control list（ACL）信息。桶的ACL可以在创建的时候设置并且通过API查看，用户需要具有READ_ACP（读取桶ACL信息）权限才可以查询桶的ACL信息。

代码示例

```
// 获取桶的ACL信息
func GetBucketAcl(svc *s3.S3) {
    getBucketAclInput := &s3.GetBucketAclInput{
        Bucket: aws.String("<your-bucket-name>"),
    }

    getBucketAclOutput, err := svc.GetBucketAcl(getBucketAclInput)
    if err != nil {
        fmt.Printf("fail to get bucekt acl. %v\n", err)
        return
    }
    owner := getBucketAclOutput.Owner
    fmt.Printf("owner is %v with ID %v\n", *owner.DisplayName, *owner.ID)
    for _, grant := range getBucketAclOutput.Grants {
        fmt.Println(grant.String())
    }
}
```

通过GetBucketAclRequest操作：

GetBucketAclRequest操作首先生成一个"request.Request"对象，该对象是一个执行GetBucketAcl操作的请求。通过调用Request对象的Send方法完成获取bucket ACL信息的操作。该方法可以生成定制化的请求，例如自定义请求头部请求超时重试设置。

```
func GetBucketAclRequest(svc *s3.S3) {
    getBucketAclInput := &s3.GetBucketAclInput{
        Bucket: aws.String("<your-bucket-name>"),
    }
    req, getBucketAclOutput := svc.GetBucketAclRequest(getBucketAclInput)

    err := req.Send()
    if err != nil {
        fmt.Printf("fail to get bucket acl. %v\n", err)
    } else {
        fmt.Println(getBucketAclOutput)
    }
}
```

请求参数

GetBucketAclInput可设置的参数如下：

参数	类型	说明	是否必要
Bucket	*string	bucket的名称	是

返回结果

GetBucketAclOutput返回的属性如下：

属性名	类型	说明
Grants	[]*Grant	Grant信息的数组，包含了每项授权和被授予人的信息
Owner	*Owner	bucket的拥有者信息

设置桶策略

功能说明

桶策略（bukcet policy）可以灵活地配置用户各种操作和访问资源的权限。访问控制列表（access control lists，ACL）只能对单一对象设置权限，而桶策略可以基于各种条件对一个桶内的全部或者一组对象配置权限。桶的拥有者拥有PutBucketPolicy操作的权限，如果桶已经被设置了policy，则新的policy会覆盖原有的policy。

设置桶策略操作可以设置桶策略，描述桶策略的信息以JSON格式的字符串形式通过Policy参数传入。一个policy的示例如下：

```
{
    "Id":"PolicyId",
    "Version":"2012-10-17",
    "Statement":[
        {
            "Sid":"ExampleStatementID1",
            "Principal":{
                "AWS":[
                    "arn:aws:iam:::user/userId",
                    "arn:aws:iam:::user/userName"
                ]
            },
            "Effect":"Allow",
            "Action":[
                "s3:ListBucket",
                "s3:CreateBucket"
            ],
            "Resource":[
                "arn:aws:iam:::exampleBucket"
            ],
            "Condition":"some conditions"
        },
        ......
    ]
}
```

Statement的内容说明如下：

元素	描述	是否必要
Sid	statement Id，可选关键字，描述statement的字符串	否
Principal	可选关键字，被授权人，指定本条statement权限针对的Domain以及User，支持通配符“*”，表示所有用户（匿名用户）。当对Domain下所有用户授权时，Principal格式为arn:aws:iam:::user/*。当对某个User进行授权时，Principal格式为arn:aws:iam:::user/userId或者arn:aws:iam:::user/userName。	可选，Principal与NotPrincipal选其一
NotPrincipal	可选关键字，不被授权人，statement匹配除此之外的其他人。取值同Principal	可选，NotPrincipal与Principal选其一
Action	可选关键字，指定本条statement作用的操作，Action字段为对象存储支持的所有操作集合，以字符串形式表示，不区分大小写。支持通配符“*”，表示该资源能进行的所有操作。例如："Action":["List*", "Get*"]	可选，Action与NotAction选其一
NotAction	可选关键字，指定一组操作，statement匹配除该组操作之外的其他操作。 取值同Action	可选，NotAction与Action选其一
Effect	必选关键字，效果，指定本条statement的权限是允许还是拒绝，Effect的值必须为Allow或者Deny	必选
Resource	可选关键字，指定statement起作用的一组资源，支持通配符“*”，表示所有资源	可选，Resource与NotResource选其一
NotResource	可选关键字，指定一组资源，statement匹配除该组资源之外的其他资源。 取值同Resource	可选，NotResource与Resource选其一
Condition	可选关键字，本条statement生效的条件	可选

代码示例

```go
// 设置桶策略
func PutBucketPolicy(svc *s3.S3) {
    putBucketPolicyInput := &s3.PutBucketPolicyInput{
        Bucket: aws.String("<your-bucket-name>"),
        Policy: aws.String("<example-policy>"),
    }
    _, err := svc.PutBucketPolicy(putBucketPolicyInput)
    if err != nil {
        fmt.Printf("fail to put bucket policy. %v\n", err)
    }
}
```

通过PutBucketPolicyRequest操作：

PutBucketPolicyRequest操作首先生成一个"request.Request"对象，该对象是一个执行PutBucketPolicy操作的请求。通过调用Request对象的Send方法完成设置桶策略的操作。该方法可以生成定制化的请求，例如自定义请求头部请求超时重试设置。

```go
func PutBucketPolicyRequest(svc *s3.S3) {
    putBucketPolicyInput := &s3.PutBucketPolicyInput{
        Bucket: aws.String("<your-bucket-name>"),
        Policy: aws.String("<example-policy>"),
    }
    req, _ := svc.PutBucketPolicyRequest(putBucketPolicyInput)

    err := req.Send()
    if err != nil {
        fmt.Printf("fail to put bucket policy. %v\n", err)
    }
}
```

请求参数

PutBucketPolicyInput可以设置的参数如下：

参数	类型	说明	是否必要
Bucket	*string	桶的名称	是
Policy	*string	JSON格式的桶策略信息	是

获取桶策略

功能说明

获取桶策略操作用于获取桶的policy，policy配置功能可以使用户根据需求更精确地定义桶的访问策略。桶的拥有者可以查看桶的policy信息。

代码示例

```
// 获取桶的策略信息
func GetBucketPolicy(svc *s3.S3) {
    getBucketPolicyInput := &s3.GetBucketPolicyInput{
        Bucket: aws.String("<your-bucket-name>"),
    }

    getBucketPolicyOutput, err := svc.GetBucketPolicy(getBucketPolicyInput)
    if err != nil {
        fmt.Printf("fail to get policy of bucket. %v\n", err)
        return
    }
    fmt.Printf("policy of bucket: %v\n", *getBucketPolicyOutput.Policy)
}
```

通过GetBucketPolicyRequest操作：

GetBucketPolicyRequest操作首先生成一个"request.Request"对象，该对象是一个执行GetBucketPolicy操作的请求。通过调用Request对象的Send方法完成获取bucket policy信息的操作。该方法可以生成定制化的请求，例如自定义请求头部请求超时重试设置。

```
func GetBucketPolicyRequest(svc *s3.S3) {
    getBucketPolicyInput := &s3.GetBucketPolicyInput{
        Bucket: aws.String("<your-bucket-name>"),
    }
    req, getBucketPolicyOutput :=
svc.GetBucketPolicyRequest(getBucketPolicyInput)

    err := req.Send()
    if err != nil {
        fmt.Printf("fail to get bucket policy. %v\n", err)
    } else {
        fmt.Println(getBucketPolicyOutput)
    }
}
```

请求参数

GetBucketPolicyInput可设置的参数如下：

参数	类型	说明	是否必要
Bucket	*string	桶的名称	是

返回结果

GetBucketPolicyOutput返回的属性如下：

属性名	类型	说明
Policy	*string	JSON格式的桶策略信息

删除桶策略

功能说明

删除桶策略操作可以删除桶已经配置的策略，桶的创建者默认拥有删除桶策略的权限。

代码示例

```
func DeleteBucketPolicy(svc *s3.S3) {
    deleteBucketPolicyInput := &s3.DeleteBucketPolicyInput{
        Bucket: aws.String("<your-bucket-name>"),
    }

    _, err := svc.DeleteBucketPolicy(deleteBucketPolicyInput)
    if err != nil {
        fmt.Printf("fail to delete bucket policy. %v\n", err)
        return
    }
}
```

通过DeleteBucketPolicyRequest操作：

DeleteBucketPolicyRequest操作首先生成一个"request.Request"对象，该对象是一个执行DeleteBucketPolicy操作的请求。通过调用Request对象的Send方法完成删除桶策略的操作。该方法可以生成定制化的请求，例如自定义请求头部请求超时重试设置。

```
func DeleteBucketPolicyRequest(svc *s3.S3) {
    deleteBucketPolicyInput := &s3.DeleteBucketPolicyInput{
        Bucket: aws.String("<your-bucket-name>"),
    }
    req, _ := svc.DeleteBucketPolicyRequest(deleteBucketPolicyInput)

    err := req.Send()
    if err != nil {
        fmt.Printf("fail to delete bucket policy. %v\n", err)
    }
}
```

请求参数

DeleteBucketPolicyInput可以设置的参数如下：

参数	类型	说明	是否必要
Bucket	*string	桶的名称	是

设置桶生命周期规则

功能说明

设置桶生命周期规则操作可以设置桶的生命周期规则，规则可以通过匹配对象key前缀、标签的方法设置当前版本或者历史版本对象的过期时间，对象过期后会被自动删除。桶的版本控制状态必须处于Enabled或者Suspended，历史版本对象过期时间配置才能生效。每次执行操作会覆盖桶中已存在的生命周期规则。

代码示例

```
// 设置桶生命周期规则
func PutBucketLifecycleConfiguration(svc *s3.S3) {
    // rule1:设置匹配指定前缀的对象一天后过期
    rule1 := &s3.LifecycleRule{
        ID:     aws.String("expireAfterOneDay"),
        Status: aws.String("Enabled"),
        Filter: &s3.LifecycleRuleFilter{
            Prefix: aws.String("expireAfterOneDay/"),
        },
        Expiration: &s3.LifecycleExpiration{
            Days: aws.Int64(1),
        },
    }
    // rule2: 设置匹配指定前缀的对象的历史版本一天后过期
    rule2 := &s3.LifecycleRule{
        ID:     aws.String("noncurrentVersionExpireAfterOneDay"),
        Status: aws.String("Enabled"),
        Filter: &s3.LifecycleRuleFilter{
            Prefix: aws.String("noncurrentVersionExpireAfterOneDay/"),
        },
        NoncurrentVersionExpiration: &s3.NoncurrentVersionExpiration{
            NoncurrentDays: aws.Int64(1),
        },
    }
    // rule3: 设置匹配指定标签信息的对象一天后过期
    rule3 := &s3.LifecycleRule{
        ID:     aws.String("withTagsExpireAfterOneDay"),
        Status: aws.String("Enabled"),
        Expiration: &s3.LifecycleExpiration{
            Days: aws.Int64(1),
        },
        Filter: &s3.LifecycleRuleFilter{
            Tag: &s3.Tag{
                Key:   aws.String("<key1>"),
                Value: aws.String("<value1>"),
            },
        },
    }
    putBucketLifecycleConfigurationInput :=
&s3.PutBucketLifecycleConfigurationInput{
        Bucket: aws.String("<your-bucket-name>"),
        LifecycleConfiguration: &s3.BucketLifecycleConfiguration{
            Rules: []*s3.LifecycleRule{rule1, rule2, rule3},
        },
    }
    _, err :=
svc.PutBucketLifecycleConfiguration(putBucketLifecycleConfigurationInput)
    if err != nil {
        fmt.Printf("fail to put bucket lifecycle configuration. %v\n", err)
    }
}
```

通过PutBucketLifecycleConfigurationRequest操作:

PutBucketLifecycleConfigurationRequest操作首先生成一个"request.Request"对象，该对象是一个执行PutBucketLifecycleConfiguration操作的请求。通过调用Request对象的Send方法完成获设置桶生命周期规则的操作。该方法可以生成定制化的请求，例如自定义请求头部请求超时重试设置。

```
func PutBucketLifecycleConfigurationRequest(svc *s3.S3) {
    // rule1:设置匹配指定前缀的对象一天后过期
    rule1 := &s3.LifecycleRule{
        ID:     aws.String("expireAfterOneDay"),
        Status: aws.String("Enabled"),
        Filter: &s3.LifecycleRuleFilter{
            Prefix: aws.String("expireAfterOneDay/"),
        },
        Expiration: &s3.LifecycleExpiration{
            Days: aws.Int64(1),
        },
    }
    // rule2: 设置匹配指定前缀的对象的历史版本一天后过期
    rule2 := &s3.LifecycleRule{
        ID:     aws.String("noncurrentVersionExpireAfterOneDay"),
        Status: aws.String("Enabled"),
        Filter: &s3.LifecycleRuleFilter{
            Prefix: aws.String("noncurrentVersionExpireAfterOneDay/"),
        },
        NoncurrentVersionExpiration: &s3.NoncurrentVersionExpiration{
            NoncurrentDays: aws.Int64(1),
        },
    }
    // rule3: 设置匹配指定标签信息的对象一天后过期
    rule3 := &s3.LifecycleRule{
        ID:     aws.String("withTagsExpireAfterOneDay"),
        Status: aws.String("Enabled"),
        Expiration: &s3.LifecycleExpiration{
            Days: aws.Int64(1),
        },
        Filter: &s3.LifecycleRuleFilter{
            Tag: &s3.Tag{
                Key:   aws.String("<key1>"),
                Value: aws.String("<value1>"),
            },
        },
    }
    putBucketLifecycleConfigurationInput :=
&s3.PutBucketLifecycleConfigurationInput{
        Bucket: aws.String("<your-bucket-name>"),
        LifecycleConfiguration: &s3.BucketLifecycleConfiguration{
            Rules: []*s3.LifecycleRule{rule1, rule2, rule3},
        },
    }

    req, _ :=
svc.PutBucketLifecycleConfigurationRequest(putBucketLifecycleConfigurationInput)

    err := req.Send()
    if err != nil {
        fmt.Printf("fail to put bucket policy. %v\n", err)
    }
}
```

请求参数

PutBucketLifecycleConfigurationInput可设置的参数如下：

参数	类型	说明	是否必要
Bucket	*string	bucket的名称	是
LifecycleConfiguration	*BucketLifecycleConfiguration	封装了生命周期规则的数组	是

查看桶生命周期规则

功能说明

查看桶生命周期规则操作可以查看桶当前的生命周期规则。生命周期管理可以通过设置规则实现自动清理过期的对象，优化存储空间。

代码示例

```
func GetBucketLifecycleConfiguration(svc *s3.S3) {
    getBucketLifecycleConfigurationInput :=
&s3.GetBucketLifecycleConfigurationInput{
        Bucket: aws.String("<your-bucket-name>"),
    }
    getBucketLifecycleConfigurationOutput, err :=
svc.GetBucketLifecycleConfiguration(getBucketLifecycleConfigurationInput)
    if err != nil {
        fmt.Printf("fail to get bucket lifecycle. %v\n", err)

        return
    }
    fmt.Println(getBucketLifecycleConfigurationOutput)
}
```

通过GetBucketLifecycleConfigurationRequest操作：

GetBucketLifecycleConfigurationRequest操作首先生成一个"request.Request"对象，该对象是一个执行GetBucketLifecycleConfiguration操作的请求。通过调用Request对象的Send方法完成获取桶当前生命周期规则的操作。该方法可以生成定制化的请求，例如自定义请求头部请求超时重试设置。

```
func GetBucketLifecycleConfigurationRequest(svc *s3.S3) {
    getBucketLifecycleConfigurationInput :=
&s3.GetBucketLifecycleConfigurationInput{
        Bucket: aws.String("<your-bucket-name>"),
    }
    req, getBucketLifecycleConfigurationOutput :=
svc.GetBucketLifecycleConfigurationRequest(getBucketLifecycleConfigurationInput)

    err := req.Send()
    if err != nil {
        fmt.Printf("fail to put bucket policy. %v\n", err)
    } else {
        fmt.Println(getBucketLifecycleConfigurationOutput)
    }
}
```

请求参数

GetBucketLifecycleConfigurationInput可设置的参数如下：

参数	类型	说明	是否必要
Bucket	*string	bucket的名称	是

返回结果

GetBucketLifecycleConfigurationOutput返回的属性如下：

属性名	类型	说明
Rules	[]*LifecycleRule	一个描述生命周期管理的规则数组，一条规则包含了规则ID、匹配的对象key前缀、匹配的对象标签信息、当前版本对象过期时间、历史版本对象过期时间和是否生效标识等信息

删除桶生命周期规则

功能说明

删除桶生命周期操作可以删除桶中的全部生命周期规则。

代码示例

```
func DeleteBucketLifeCycle(svc *s3.S3) {
    deleteBucketLifecycleInput := &s3.DeleteBucketLifecycleInput{
        Bucket: aws.String("<your-bucket-name>"),
    }

    _, err := svc.DeleteBucketLifecycle(deleteBucketLifecycleInput)
    if err != nil {
        fmt.Printf("fail to delete bucket lifecycle. %v\n", err)
    }
}
```

通过DeleteBucketLifecycleRequest操作：

DeleteBucketLifecycleRequest操作首先生成一个"request.Request"对象，该对象是一个执行DeleteBucketLifecycle操作的请求。通过调用Request对象的Send方法完成删除桶生命周期规则的操作。该方法可以生成定制化的请求，例如自定义请求头部请求超时重试设置。

```
func DeleteBucketLifeCycleRequest(svc *s3.S3) {
    deleteBucketLifecycleInput := &s3.DeleteBucketLifecycleInput{
        Bucket: aws.String("<your-bucket-name>"),
    }

    req, _ := svc.DeleteBucketLifecycleRequest(deleteBucketLifecycleInput)

    err := req.Send()
    if err != nil {
        fmt.Printf("fail to put bucket policy. %v\n", err)
    }
}
```

请求参数

DeleteBucketLifecycleInput可设置的参数如下：

参数	类型	说明	是否必要
Bucket	*string	bucket的名称	是

配置桶的website配置

功能说明

设置桶的静态网站配置。

代码示例

```
func PutBucketWebsite(svc *s3.S3) {
    putBucketWebsiteInput := &s3.PutBucketWebsiteInput{
        Bucket: aws.String("<your-bucket-name>"),
        WebsiteConfiguration: &s3.WebsiteConfiguration{
            ErrorDocument: &s3.ErrorDocument{
                Key: aws.String("error.html"),
            },
            IndexDocument: &s3.IndexDocument{
                Suffix: aws.String("index.html"),
            },
        },
    }

    _, err := svc.PutBucketWebsite(putBucketWebsiteInput)
    if err != nil {
        fmt.Printf("fail to put bucket website. %v\n", err)
    }
}
```

通过PutBucketWebsiteRequest操作设置桶网站配置：

PutBucketWebsiteRequest操作首先生成一个"request.Request"对象，该对象是一个执行PutBucketWebsite操作的请求。通过调用Request对象的Send方法完成设置桶静态网站配置的操作。该方法可以生成定制化的请求，例如自定义请求头部请求超时重试设置。

```
func PutBucketWebsiteRequest(svc *s3.S3) {
    putBucketWebsiteInput := &s3.PutBucketWebsiteInput{
        Bucket: aws.String("<your-bucket-name>"),
        WebsiteConfiguration: &s3.WebsiteConfiguration{
            ErrorDocument: &s3.ErrorDocument{
                Key: aws.String("error.html"),
            },
            IndexDocument: &s3.IndexDocument{
                Suffix: aws.String("index.html"),
            },
        },
    }

    req, _ := svc.PutBucketWebsiteRequest(putBucketWebsiteInput)
    err := req.Send()
    if err != nil {
        fmt.Printf("fail to put bucket website. %v\n", err)
    }
}
```

请求参数

PutBucketWebsiteInput可设置的参数如下：

参数	类型	说明	是否必要
Bucket	*string	bucket的名称	是
WebsiteConfiguration	*WebsiteConfiguration	描述桶网站配置的配置项	是

获取桶的website配置

功能说明

获取桶的静态网站配置。

代码示例

```
func GetBucketWebsite(svc *s3.S3) {
    getBucketWebsiteInput := &s3.GetBucketWebsiteInput{
        Bucket: aws.String("<your-bucket-name>"),
    }

    getBucketWebsiteOutput, err := svc.GetBucketWebsite(getBucketWebsiteInput)
    if err != nil {
        fmt.Printf("fail to get bucket website. %v\n", err)
    } else {
        fmt.Println(getBucketWebsiteOutput)
    }
}
```

通过GetBucketWebsiteRequest操作获取桶网站配置：

GetBucketWebsiteRequest操作首先生成一个"request.Request"对象，该对象是一个执行GetBucketWebsite操作的请求。通过调用Request对象的Send方法完成获取桶静态网站配置的操作。该方法可以生成定制化的请求，例如自定义请求头部请求超时重试设置。

```
func GetBucketWebsiteRequest(svc *s3.S3) {
    getBucketWebsiteInput := &s3.GetBucketWebsiteInput{
        Bucket: aws.String("<your-bucket-name>"),
    }

    req, getBucketWebsiteOutput :=
svc.GetBucketWebsiteRequest(getBucketWebsiteInput)
    err := req.Send()
    if err != nil {
        fmt.Printf("fail to get bucket website. %v\n", err)
    } else {
        fmt.Println(getBucketWebsiteOutput)
    }
}
```

请求参数

GetBucketWebsiteInput可设置的参数如下：

参数	类型	说明	是否必要
Bucket	*string	bucket的名称	是

返回结果

GetBucketWebsiteOutput返回的属性如下：

属性名	类型	说明
ErrorDocument	*ErrorDocument	错误页面信息
IndexDocument	*IndexDocument	默认主页页面信息
RedirectAllRequestsTo	*RedirectAllRequestsTo	重定向全部请求配置
RoutingRules	[]*RoutingRule	重定向规则配置

删除桶的website配置

功能说明

删除桶的静态网站配置。

代码示例

```
func DeleteBucketWebsite(svc *s3.S3) {
    deleteBucketWebsiteInput := &s3.DeleteBucketWebsiteInput{
        Bucket: aws.String("<your-bucket-name>"),
    }

    _, err := svc.DeleteBucketWebsite(deleteBucketWebsiteInput)
    if err != nil {
        fmt.Printf("fail to delete bucket website. %v\n", err)
    }
}
```

通过DeleteBucketWebsiteRequest操作：

DeleteBucketWebsiteRequest操作首先生成一个"request.Request"对象，该对象是一个执行DeleteBucketWebsite操作的请求。通过调用Request对象的Send方法完成删除桶静态网站配置的操作。该方法可以生成定制化的请求，例如自定义请求头部请求超时重试设置。

```
func DeleteBucketWebsiteRequest(svc *s3.S3) {
    deleteBucketWebsiteInput := &s3.DeleteBucketWebsiteInput{
        Bucket: aws.String("<your-bucket-name>"),
    }

    req, _ := svc.DeleteBucketWebsiteRequest(deleteBucketWebsiteInput)
    err := req.Send()
    if err != nil {
        fmt.Printf("fail to delete bucket website. %v\n", err)
    }
}
```

请求参数

DeleteBucketWebsiteInput可设置的参数如下：

参数	类型	说明	是否必要
Bucket	*string	bucket的名称	是

设置桶的多版本状态

功能说明

设置桶多版本状态操作可以设置版本控制状态。桶的版本控制状态可以设置为以下的值：

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

- Suspended：关闭bucket的版本控制，之后每个添加到bucket中的对象的version ID会被设置为null。

代码示例

```
func PutBucketVersioning(svc *s3.S3) {
    putBucketVersioningInput := &s3.PutBucketVersioningInput{
        Bucket: aws.String("<your-bucket-name>"),
        VersioningConfiguration: &s3.VersioningConfiguration{
            Status:    aws.String("Enabled"), //启用版本控制：Enabled，暂停版本控制：
Suspended
        },
    }

    _, err := svc.PutBucketVersioning(putBucketVersioningInput)
    if err != nil {
        fmt.Printf("fail to put bucket versioning. %v\n", err)
        return
    }
}
```

通过PutBucketVersioningRequest操作：

PutBucketVersioningRequest操作首先生成一个"request.Request"对象，该对象是一个执行PutBucketVersioning操作的请求。通过调用Request对象的Send方法完成配置bucket版本控制信息的操作。该方法可以生成定制化的请求，例如自定义请求头部请求超时重试设置。

```
func PutBucketVersioningRequest(svc *s3.S3) {
    putBucketVersioningInput := &s3.PutBucketVersioningInput{
        Bucket: aws.String("<your-bucket-name>"),
        VersioningConfiguration: &s3.VersioningConfiguration{
            Status: aws.String("Enabled"), //启用版本控制
        },
    }
    req, _ := svc.PutBucketVersioningRequest(putBucketVersioningInput)

    err := req.Send()
    if err != nil {
        fmt.Printf("fail to put bucket versioning request. %v\n", err)
    }
}
```

请求参数

PutBucketVersioningInput可设置的参数如下：

参数	类型	说明	是否必要
Bucket	*string	bucket的名称	是
VersioningConfiguration	*VersioningConfiguration	封装了设置版本控制状态的参数	是

获取桶的多版本状态

功能说明

获取桶的多版本状态操作可以获取桶的版本控制状态信息。只有bucket的拥有者才能获取到桶的版本控制信息。

每个桶的版本控制有三个状态：未开启、开启（Enabled）和暂停（Suspended）版本控制，如果桶从来没有被设置过版本控制状态，那么该桶默认为未开启版本控制状态，执行GetBucketVersioning操作不能获取任何版本控制信息。

代码示例

```
func GetBucketVersioning(svc *s3.S3){
    getBucketVersioningInput := &s3.GetBucketVersioningInput{
        Bucket: aws.String("<your-bucket-name>"),
    }

    getBucketVersioningOutput, err :=
svc.GetBucketVersioning(getBucketVersioningInput)
    if err != nil {
        fmt.Printf("fail to get bucket versioning. %v\n", err)
        return
    }
    fmt.Println(getBucketVersioningOutput)
}
```

通过GetBucketVersioningRequest操作：

GetBucketVersioningRequest操作首先生成一个"request.Request"对象，该对象是一个执行GetBucketVersioning操作的请求。通过调用Request对象的Send方法完成获取bucket版本控制配置信息的操作。该方法可以生成定制化的请求，例如自定义请求头部请求超时重试设置。

```
func GetBucketVersioningRequest(svc *s3.S3) {
    getBucketVersioningInput := &s3.GetBucketVersioningInput{
        Bucket: aws.String("<your-bucket-name>"),
    }
    req, getBucketVersioningOutput :=
svc.GetBucketVersioningRequest(getBucketVersioningInput)

    err := req.Send()
    if err != nil {
        fmt.Printf("fail to get bucket versioning. %v\n", err)
    } else {
        fmt.Println(getBucketVersioningOutput)
    }
}
```

请求参数

GetBucketVersioningInput可设置的参数如下：

参数	类型	说明	是否必要
Bucket	*string	bucket的名称	是

返回结果

GetBucketVersioningOutput返回的属性如下：

属性名	类型	说明
Status	*string	桶的版本控制设置状态

设置桶的CORS配置

功能说明

设置桶的跨域资源共享CORS（Cross-Origin Resource Sharing）规则。桶默认不开启跨域资源共享规则，设置桶的跨域资源共享规则时，新配置的规则会覆盖已有的规则。

代码示例

```
func PutBucketCors(svc *s3.S3) {
    putBucketCorsInput := &s3.PutBucketCorsInput{
        Bucket: aws.String("<your-bucket-name>"),
        CORSConfiguration: &s3.CORSConfiguration{
            CORSRules: []*s3.CORSRule{
                {
                    AllowedHeaders: []*string{
                        aws.String("*"),
                    },
                    AllowedMethods: []*string{
                        aws.String("PUT"),
                        aws.String("POST"),
                        aws.String("DELETE"),
                    },
                    AllowedOrigins: []*string{
                        aws.String("http://www.example.com"),
                    },
                    ExposeHeaders: []*string{
                        aws.String("x-amz-server-side-encryption"),
                    },
                    MaxAgeSeconds: aws.Int64(3000),
                },
                {
                    AllowedHeaders: []*string{
                        aws.String("Authorization"),
                    },
                    AllowedMethods: []*string{
                        aws.String("GET"),
                    },
                    AllowedOrigins: []*string{
                        aws.String("*"),
                    },
                    MaxAgeSeconds: aws.Int64(3000),
                },
```

```
            },
        },
    }

    _, err := svc.PutBucketCors(putBucketCorsInput)
    if err != nil {
        fmt.Printf("fail to put bucket cors. %v\n", err)
    }
}
```

通过PutBucketCorsRequest操作：

PutBucketCorsRequest操作首先生成一个"request.Request"对象，该对象是一个执行PutBucketCors操作的请求。通过调用Request对象的Send方法完成设置桶CORS配置的操作。该方法可以生成定制化的请求，例如自定义请求头部请求超时重试设置。

```
func PutBucketCorsRequest(svc *s3.S3) {
    putBucketCorsInput := &s3.PutBucketCorsInput{
        Bucket: aws.String("<your-bucket-name>"),
        CORSConfiguration: &s3.CORSConfiguration{
            CORSRules: []*s3.CORSRule{
                {
                    AllowedHeaders: []*string{
                        aws.String("*"),
                    },
                    AllowedMethods: []*string{
                        aws.String("PUT"),
                        aws.String("POST"),
                        aws.String("DELETE"),
                    },
                    AllowedOrigins: []*string{
                        aws.String("http://www.example.com"),
                    },
                    ExposeHeaders: []*string{
                        aws.String("x-amz-server-side-encryption"),
                    },
                    MaxAgeSeconds: aws.Int64(3000),
                },
                {
                    AllowedHeaders: []*string{
                        aws.String("Authorization"),
                    },
                    AllowedMethods: []*string{
                        aws.String("GET"),
                    },
                    AllowedOrigins: []*string{
                        aws.String("*"),
                    },
                    MaxAgeSeconds: aws.Int64(3000),
                },
            },
        },
    }

    req, _ := svc.PutBucketCorsRequest(putBucketCorsInput)
    err := req.Send()
    if err != nil {
```

```
        fmt.Printf("fail to put bucket cors. %v\n", err)
    }
}
```

请求参数

PutBucketCorsInput可设置的参数如下：

参数	类型	说明	是否必要
Bucket	*string	bucket的名称	是
CORSConfiguration	*CORSConfiguration	描述桶跨域配置的配置信息	是

获取桶的CORS配置

功能说明

获取指定桶当前生效的跨域资源共享CORS（Cross-Origin Resource Sharing）规则。

代码示例

```
func GetBucketCors(svc *s3.S3) {
    getBucketCorsInput := &s3.GetBucketCorsInput{
        Bucket: aws.String("<your-bucket-name>"),
    }

    getBucketCorsOutput, err := svc.GetBucketCors(getBucketCorsInput)
    if err != nil {
        fmt.Printf("fail to get bucket cors. %v\n", err)
    } else {
        fmt.Println(getBucketCorsOutput)
    }
}
```

通过GetBucketCorsRequest操作：

GetBucketCorsRequest操作首先生成一个"request.Request"对象，该对象是一个执行GetBucketCors操作的请求。通过调用Request对象的Send方法完成获取桶CORS配置的操作。该方法可以生成定制化的请求，例如自定义请求头部请求超时重试设置。

```
func GetBucketCorsRequest(svc *s3.S3) {
    getBucketCorsInput := &s3.GetBucketCorsInput{
        Bucket: aws.String("<your-bucket-name>"),
    }

    req, getBucketCorsOutput := svc.GetBucketCorsRequest(getBucketCorsInput)
    err := req.Send()
    if err != nil {
        fmt.Printf("fail to get bucket cors. %v\n", err)
    } else {
        fmt.Println(getBucketCorsOutput)
    }
}
```

请求参数

GetBucketCorsInput可设置的参数如下：

参数	类型	说明	是否必要
Bucket	*string	bucket的名称	是

返回结果

GetBucketCorsOutput返回的属性如下：

属性名	类型	说明
CORSRules	[]*CORSRule	桶设置的跨域资源共享规则的集合

删除桶的CORS配置

功能说明

删除指定桶的跨域资源共享CORS（Cross-Origin Resource Sharing）所有规则并关闭跨域资源共享功能。

代码示例

```
func DeleteBucketCors(svc *s3.S3) {
    deleteBucketCorsInput := &s3.DeleteBucketCorsInput{
        Bucket: aws.String("<your-bucket-name>"),
    }

    _, err := svc.DeleteBucketCors(deleteBucketCorsInput)
    if err != nil {
        fmt.Printf("fail to delete bucket website. %v\n", err)
    }
}
```

通过DeleteBucketCorsRequest操作：

DeleteBucketCorsRequest操作首先生成一个"request.Request"对象，该对象是一个执行DeleteBucketCors操作的请求。通过调用Request对象的Send方法完成删除桶CORS配置的操作。该方法可以生成定制化的请求，例如自定义请求头部请求超时重试设置。

```
func DeleteBucketCorsRequest(svc *s3.S3) {
    deleteBucketCorsInput := &s3.DeleteBucketCorsInput{
        Bucket: aws.String("<your-bucket-name>"),
    }

    req, _ := svc.DeleteBucketCorsRequest(deleteBucketCorsInput)
    err := req.Send()
    if err != nil {
        fmt.Printf("fail to delete bucket website. %v\n", err)
    }
}
```

请求参数

DeleteBucketCorsInput可设置的参数如下：

参数	类型	说明	是否必要
Bucket	*string	bucket的名称	是

设置桶标签

功能说明

以key-value的形式为桶设置标签，通过设置通标签可以标记桶的用途，方便对其进行分类和管理。

代码示例

```
func PutBucketTagging(svc *s3.S3) {
    putBucketTaggingInput := &s3.PutBucketTaggingInput{
        Bucket: aws.String("<your-bucket-name>"),
        Tagging: &s3.Tagging{
            TagSet: []*s3.Tag{
                {
                    Key:   aws.String("<key1>"),
                    Value: aws.String("<value1>"),
                },
                {
                    Key:   aws.String("<key2>"),
                    Value: aws.String("<value2>"),
                },
            },
        },
    }

    _, err := svc.PutBucketTagging(putBucketTaggingInput)
    if err != nil {
        fmt.Printf("fail to put bucket tagging. %v\n", err)
    }
}
```

通过PutBucketTaggingRequest操作：

PutBucketTaggingRequest操作首先生成一个"request.Request"对象，该对象是一个执行PutBucketTagging操作的请求。通过调用Request对象的Send方法完成设置桶标签的操作。该方法可以生成定制化的请求，例如自定义请求头部请求超时重试设置。

```
func PutBucketTaggingRequest(svc *s3.S3) {
    putBucketTaggingInput := &s3.PutBucketTaggingInput{
        Bucket: aws.String("<your-bucket-name>"),
        Tagging: &s3.Tagging{
            TagSet: []*s3.Tag{
                {
                    Key:   aws.String("<key1>"),
                    Value: aws.String("<value1>"),
                },
                {
                    Key:   aws.String("<key2>"),
                    Value: aws.String("<value2>"),
                },
            },
        },
    }

    req, _ := svc.PutBucketTaggingRequest(putBucketTaggingInput)
    err := req.Send()
    if err != nil {
        fmt.Printf("fail to put bucket tagging. %v\n", err)
    }
}
```

请求参数

PutBucketTaggingInput可设置的参数如下：

参数	类型	说明	是否必要
Bucket	*string	bucket的名称	是
Tagging	*Tagging	描述桶标签的信息	是

获取桶标签

功能说明

获取指定桶的标签信息。

代码示例

```
func GetBucketTagging(svc *s3.S3) {
    getBucketTaggingInput := &s3.GetBucketTaggingInput{
        Bucket: aws.String("<your-bucket-name>"),
    }

    getBucketTaggingOutput, err := svc.GetBucketTagging(getBucketTaggingInput)
    if err != nil {
        fmt.Printf("fail to get bucket tagging. %v\n", err)
        return
    }
    fmt.Println(getBucketTaggingOutput)
}
```

通过GetBucketTaggingRequest操作：

GetBucketTaggingRequest操作首先生成一个"request.Request"对象，该对象是一个执行GetBucketTagging操作的请求。通过调用Request对象的Send方法完成获取桶标签的操作。该方法可以生成定制化的请求，例如自定义请求头部请求超时重试设置。

```
func GetBucketTaggingRequest(svc *s3.S3) {
    getBucketTaggingInput := &s3.GetBucketTaggingInput{
        Bucket: aws.String("<your-bucket-name>"),
    }
    req, getBucketTaggingOutput :=
svc.GetBucketTaggingRequest(getBucketTaggingInput)

    err := req.Send()
    if err != nil {
        fmt.Printf("fail to get bucket tagging. %v\n", err)
    } else {
        fmt.Println(getBucketTaggingOutput)
    }
}
```

请求参数

GetBucketTaggingInput可设置的参数如下：

参数	类型	说明	是否必要
Bucket	*string	bucket的名称	是

返回结果

GetBucketTaggingOutput返回的属性如下：

属性名	类型	说明
TagSet	[]*Tag	桶的标签集合，以key-value的形式描述桶标签信息

删除桶标签

功能说明

删除指定桶的全部标签。

代码示例

```
func DeleteBucketTagging(svc *s3.S3) {
    deleteBucketTaggingInput := &s3.DeleteBucketTaggingInput{
        Bucket: aws.String("<your-bucket-name>"),
    }

    _, err := svc.DeleteBucketTagging(deleteBucketTaggingInput)
    if err != nil {
        fmt.Printf("fail to delete bucket tagging. %v\n", err)
        return
    }
}
```

通过DeleteBucketTaggingRequest操作：

DeleteBucketTaggingRequest操作首先生成一个"request.Request"对象，该对象是一个执行DeleteBucketTagging操作的请求。通过调用Request对象的Send方法完成删除桶标签的操作。该方法可以生成定制化的请求，例如自定义请求头部请求超时重试设置。

```
func DeleteBucketTaggingRequest(svc *s3.S3) {
    deleteBucketTaggingInput := &s3.DeleteBucketTaggingInput{
        Bucket: aws.String("<your-bucket-name>"),
    }
    req, _ := svc.DeleteBucketTaggingRequest(deleteBucketTaggingInput)

    err := req.Send()
    if err != nil {
        fmt.Printf("fail to delete bucket tagging. %v\n", err)
    }
}
```

请求参数

DeleteBucketTaggingInput可设置的参数如下：

参数	类型	说明	是否必要
Bucket	*string	bucket的名称	是

设置桶加密配置

功能说明

获取指定桶的加密配置信息。

代码示例

```
func GetBucketEncryption(svc *s3.S3) {
    getBucketEncryptionInput := &s3.GetBucketEncryptionInput{
        Bucket: aws.String("<your-bucket-name>"),
    }

    GetBucketEncryptionOutput, err :=
svc.GetBucketEncryption(getBucketEncryptionInput)
    if err != nil {
        fmt.Printf("fail to get bucket encryption. %v\n", err)
    } else {
        fmt.Println(GetBucketEncryptionOutput)
    }
}
```

通过GetBucketEncryptionRequest操作：

GetBucketEncryptionRequest操作首先生成一个"request.Request"对象，该对象是一个执行GetBucketEncryption操作的请求。通过调用Request对象的Send方法完成获取桶加密配置的操作。该方法可以生成定制化的请求，例如自定义请求头部请求超时重试设置。

```
func GetBucketEncryptionRequest(svc *s3.S3) {
    GetBucketEncryptionInput := &s3.GetBucketEncryptionInput{
        Bucket: aws.String("<your-bucket-name>"),
    }

    req, GetBucketEncryptionOutput :=
svc.GetBucketEncryptionRequest(GetBucketEncryptionInput)
    err := req.Send()
    if err != nil {
        fmt.Printf("fail to get bucket encryption. %v\n", err)
    } else {
        fmt.Println(GetBucketEncryptionOutput)
    }
}
```

请求参数

GetBucketEncryptionInput可设置的参数如下：

参数	类型	说明	是否必要
Bucket	*string	bucket的名称	是

返回结果

GetBucketEncryptionOutput返回的属性如下：

属性名	类型	说明
ServerSideEncryptionConfiguration	*ServerSideEncryptionConfiguration	桶设置的加密配置信息

获取桶加密配置

功能说明

获取指定桶的加密配置信息。

代码示例

```
func GetBucketEncryption(svc *s3.S3) {
    getBucketEncryptionInput := &s3.GetBucketEncryptionInput{
        Bucket: aws.String("<your-bucket-name>"),
    }

    GetBucketEncryptionOutput, err :=
svc.GetBucketEncryption(getBucketEncryptionInput)
    if err != nil {
        fmt.Printf("fail to get bucket encryption. %v\n", err)
    } else {
        fmt.Println(GetBucketEncryptionOutput)
    }
}
```

通过GetBucketEncryptionRequest操作：

GetBucketEncryptionRequest操作首先生成一个"request.Request"对象，该对象是一个执行GetBucketEncryption操作的请求。通过调用Request对象的Send方法完成获取桶加密配置的操作。该方法可以生成定制化的请求，例如自定义请求头部请求超时重试设置。

```
func GetBucketEncryptionRequest(svc *s3.S3) {
    GetBucketEncryptionInput := &s3.GetBucketEncryptionInput{
        Bucket: aws.String("<your-bucket-name>"),
    }

    req, GetBucketEncryptionOutput :=
svc.GetBucketEncryptionRequest(GetBucketEncryptionInput)
    err := req.Send()
    if err != nil {
        fmt.Printf("fail to get bucket encryption. %v\n", err)
    } else {
        fmt.Println(GetBucketEncryptionOutput)
    }
}
```

请求参数

GetBucketEncryptionInput可设置的参数如下：

参数	类型	说明	是否必要
Bucket	*string	bucket的名称	是

返回结果

GetBucketEncryptionOutput返回的属性如下：

属性名	类型	说明

属性名SideEncryptionConfiguration	类型verSideEncryptionConfiguration	桶设置的加密配置信息

删除桶加密配置

功能说明

删除指定桶的全部加密配置，停用上传对象时自动加密功能。

代码示例

```
func DeleteBucketEncryption(svc *s3.S3) {
    deleteBucketEncryptionInput := &s3.DeleteBucketEncryptionInput{
        Bucket: aws.String("<your-bucket-name>"),
    }

    _, err := svc.DeleteBucketEncryption(deleteBucketEncryptionInput)
    if err != nil {
        fmt.Printf("fail to delete bucket encryption. %v\n", err)
    }
}
```

通过DeleteBucketEncryptionRequest操作：

DeleteBucketEncryptionRequest操作首先生成一个"request.Request"对象，该对象是一个执行DeleteBucketEncryption操作的请求。通过调用Request对象的Send方法完成删除桶加密配置的操作。该方法可以生成定制化的请求，例如自定义请求头部请求超时重试设置。

```
func DeleteBucketEncryptionRequest(svc *s3.S3) {
    deleteBucketEncryptionInput := &s3.DeleteBucketEncryptionInput{
        Bucket: aws.String("<your-bucket-name>"),
    }

    req, _ := svc.DeleteBucketEncryptionRequest(deleteBucketEncryptionInput)
    err := req.Send()
    if err != nil {
        fmt.Printf("fail to delete bucket encryption. %v\n", err)
    }
}
```

请求参数

DeleteBucketEncryptionInput可设置的参数如下：

参数	类型	说明	是否必要
Bucket	*string	bucket的名称	是

对象相关接口

上传对象

功能说明

上传对象操作用于上传对象。如果对同一个对象同时发起多个上传请求，最后一次完成的请求将覆盖之前所有请求的上传的对象。可以通过设置请求头部中的Content-MD5字段来保证数据被完整上传，如果上传的数据不能通过MD5校验，该操作将返回一个错误提示。用户可以通过比较上传对象后获得的ETag 与原文件的MD5值是否相等来确认上传操作是否成功。

上传对象操作在上传对象时可以在请求里携带HTTP协议规定的6个请求头：Cache-Control、Expires、Content-Encoding、Content-Disposition、Content-Type、Content-Language。如果上传对象的请求设置了这些请求头，服务端会直接将这些头域的值保存下来。这6个值也可以通过修改对象元数据操作进行修改。在该对象被下载或者执行HeadObject操作的时候，这些保存的值将会被设置到对应的HTTP头域中返回客户端。

上传对象操作可以上传最大不超过5GB的文件，超过5GB的文件可以通过分段上传操作上传到对象存储服务，对象key的命名使用UTF-8编码，长度必须在1~1023字节之间，不能反斜线（\）开头。

代码示例

```
func PutObject(svc *s3.S3) {
    file, err := os.Open("<file-path>")
    if err != nil {
        fmt.Printf("Unable to open file:%v", err)
        os.Exit(1)
    }
    defer func() {
        err := file.Close()
        if err != nil {
            fmt.Printf("fail to close file. %v\n", err)
        }
    }()

    putObjectInput := &s3.PutObjectInput{
        Body:   file,
        Bucket: aws.String("<your-bucket-name>"),
        Key:    aws.String("<your-object-key>"),
    }
    putObjectOutput, err := svc.PutObject(putObjectInput)
    if err != nil {
        fmt.Printf("Failed to put object. %v", err)
    } else {
        fmt.Println(putObjectOutput)
    }
}
```

通过PutObjectRequest操作：

PutObjectRequest操作首先生成一个"request.Request"对象，该对象是一个执行PutObject操作的请求。通过调用Request对象的Send方法完成上传对象的操作。该方法可以生成定制化的请求，例如自定义请求头部请求超时重试设置。

```
func PutObjectRequest(svc *s3.S3) {
    file, err := os.Open("exampleFile")
    if err != nil {
        fmt.Printf("Unable to open file:%v", err)
        os.Exit(1)
    }
```

```
    defer func() {
        err := file.Close()
        if err != nil {
            fmt.Printf("fail to close file. %v\n", err)
        }
    }()

    putObjectInput := &s3.PutObjectInput{
        Body:   file,
        Bucket: aws.String("<your-bucket-name>"),
        Key:    aws.String("<your-object-key>"),
    }
    req, putObjectOutput := svc.PutObjectRequest(putObjectInput)

    err = req.Send()
    if err != nil {
        fmt.Printf("fail to put object. %v\n", err)
    } else {
        fmt.Println(putObjectOutput)
    }
}
```

请求参数

PutObjectInput可设置的参数如下：

参数	类型	说明	是否必要
ACL	*string	配置上传对象的预定义的标准ACL信息，例如private，public-read，public-read-write等。	否
Body	io.ReadSeeker	对象的数据。	是
Bucket	*string	执行本操作的桶名称。	是
ContentLength	*int64	说明请求body的长度（单位：字节），该参数可以在body长度不能被自动识别的情况下设置。	否
ContentMD5	*string	base64编码的128位MD5值，不包含请求头部的信息	否
GrantFullControl	*string	用于自定义用户对此对象的FULL_CONTROL权限信息。	否
GrantRead	*string	用于自定义用户对此对象的READ权限信息。	否
GrantReadACP	*string	用于自定义用户对此对象的READ_ACP权限信息。	否
GrantWrite	*string	用于自定义用户对此对象的WRITE权限信息。	否
GrantWriteACP	*string	用于自定义用户对此对象的WRITE_ACP权限信息。	否
Key	*string	上传文件到对象存储服务后对应的key。PutObject操作支持将文件上传至文件夹，如需要将对象上传至"/folder"文件下，只需要设置Key="/folder/{exampleKey}"即可。	是
Metadata	map[string]*string	对象的元数据信息。	否
Tagging	*string	对象的标签信息，必须是URL请求参数的形式。例如，"Key1=Value1"。	否
WebsiteRedirectLocation	*string	如果bucket被配置用于提供网站的静态数据，该参数可以用于设置访问对象时候重定向到当前bucket下的其他对象或者外部的URL。	否

返回结果

PutObjectOutput返回的属性如下：

参数	类型	说明
ETag	*string	上传对象后对应的Entity Tag
VersionId	*string	上传对象后相应的版本Id。
ServerSideEncryption	*string	返回对象数据加密的算法名称。

下载对象

功能说明

下载对象操作可以获取对象数据，并且保存为本地文件。执行GetObject操作必须对目标Object具有READ权限。

代码示例

```
func GetObject(svc *s3.S3) {
    getObjectInput := &s3.GetObjectInput{
        Bucket: aws.String("<your-bucket-name>"),
        Key:    aws.String("<your-object-key>"),
    }

    getObjectOutput, err := svc.GetObject(getObjectInput)
    if err != nil {
        fmt.Printf("Failed to get object. %v\n", err)
        return
    }
    defer func() {
        err := getObjectOutput.Body.Close()
        if err != nil {
            fmt.Printf("fail to close result body. %v\n", err)
        }
    }()
    fmt.Println(getObjectOutput)
    // 将对象保存为本地文件
    p := make([]byte, 1024)
    filepath := "<file-path>"
    file, err := os.Create(filepath)
    if err != nil {
        fmt.Println("fail to create file.", err)
        return
    }
    defer func() {
        err := file.Close()
        if err != nil {
            fmt.Printf("fail to close file. %v\n", err)
        }
    }()
    for {
        readCount, readErr := getObjectOutput.Body.Read(p)
        _, _ = file.Write(p[:readCount])
        if readErr != nil {
            break
        }
    }
```

```
}
```

通过GetObjectRequest操作获取对象：

GetObjectRequest操作首先生成一个"request.Request"对象，该对象是一个执行GetObject操作的请求。通过调用Request对象的Send方法完成下载对象的操作。该方法可以生成定制化的请求，例如自定义请求头部请求超时重试设置。

```
func GetObjectRequest(svc *s3.S3) {
    getObjectInput := &s3.GetObjectInput{
        Bucket: aws.String("<your-bucket-name>"),
        Key:    aws.String("<your-object-key>"),
    }
    req, getObjectOutput := svc.GetObjectRequest(getObjectInput)

    err := req.Send()
    if err != nil {
        fmt.Printf("fail to get object. %v\n", err)
        return
    }
    // 将对象保存为本地文件
    p := make([]byte, 1024)
    filepath := "<file-path>"
    file, err := os.Create(filepath)
    if err != nil {
        fmt.Println("fail to create file.", err)
        return
    }
    defer func() {
        err := file.Close()
        if err != nil {
            fmt.Printf("fail to close file. %v\n", err)
        }
    }()
    for {
        readCount, readErr := getObjectOutput.Body.Read(p)
        _, _ = file.Write(p[:readCount])
        if readErr != nil {
            break
        }
    }
}
```

通过downloader下载对象：

```
func DownloadObject(sess *session.Session) {
    filePath := "<file-path>"
    file, err := os.Create(filePath)
    if err != nil {
        fmt.Println("fail to create file.", err)
        return
    }
    defer func() {
        err := file.Close()
        if err != nil {
            fmt.Printf("fail to close file. %v\n", err)
```

```
        }
    }()

    downloader := s3manager.NewDownloader(sess)
    _, err = downloader.Download(file,
        &s3.GetObjectInput{
            Bucket: aws.String("<your-bucket-name>"),
            Key:    aws.String("<your-object-key>"),
        })
    if err != nil {
        fmt.Printf("fail to download file. %v\n", err)
        return
    }
}
```

请求参数

GetObjectInput可设置的参数如下：

参数	类型	说明	是否必要
Bucket	*string	执行本操作的桶名称。	是
IfMatch	*string	用于指定只有在对象的ETag和该参数值匹配的情况下才返回对象数据，否则返回412错误码。	否
IfModifiedSince	*time.Time	用于只有当对象在指定时间后被修改的情况下才返回该对象，否则返回304错误码。	否
IfNoneMatch	*string	用于指定只有在对象的ETag和该参数值不匹配的情况下才返回对象数据，否则返回304错误码	否

参数	类型	说明	是否必要
IfUnmodifiedSince	*time.Time	用于仅当对象自指定时间以来未被修改的情况下才返回对象数据，否则返回412错误码。	否
Key	*string	对象的key。	是
PartNumber	*int64	读取对象指定的片段，该参数大于等于1，小于等于10000。	否
Range	*string	下载对象指定范围内的数据（单位：字节），必须是"bytes=first-last"的格式，例如"bytes=0-9"表示前10字节的数据，详情请参见详情请参见RFC2616。	否
ResponseCacheControl	*string	用于设置response的Cache-Control头部字段信息。	否
ResponseContentDisposition	*string	用于设置response的Content-Disposition头部字段信息。	否
ResponseContentEncoding	*string	用于设置response的Content-Encoding头部字段信息。	否
ResponseContentLanguage	*string	用于设置response的Content-Language头部字段信息。	否
ResponseContentType	*string	用于设置response的Content-Type头部字段信息。	否
ResponseExpires	*time.Time	用于设置response的Expires头部字段信息。	否
VersionId	*string	当bucket开启版本控制的时候，用于指定获取指定版本的对象数据，当不指定该参数的时候，默认获取最新版本的对象数据。	否

返回结果

GetObjectOutput返回的属性如下：

参数	类型	说明
Body	io.ReadCloser	对象的数据。
ContentLength	*int64	对象数据的长度，单位为字节。
ContentType	*string	数据的标准MIME类型。
ETag	*string	对象的Entity Tag。
LastModified	*time.Time	最后修改对象的时间。
TagCount	*int64	对象标签的数量。
VersionId	*string	对象的版本ID。
StorageClass	*string	对象的存储类型。

范围下载

功能说明

范围下载可以下载对象指定范围的数据，通过Range参数指定需要下载的数据范围，如果指定的下载范围为0-9，则返回对象的第1至第10字节共10字节的数据。如果指定的起始位置大于对象的大小，或者起始位置大于结束位置，则返回{416: InvalidRange}错误码。如果指定的结束位置大于对象大小，则返回从对象从起始位置开始的全部数据。

代码示例

```
func GetObject(svc *s3.S3) {
    getObjectInput := &s3.GetObjectInput{
        Bucket: aws.String("<your-bucket-name>"),
        Key:    aws.String("<your-object-key>"),
        // 请求对象第1至第10字节的数据
        Range:  aws.String("bytes=0-9"),
    }

    getObjectOutput, err := svc.GetObject(getObjectInput)
    if err != nil {
        fmt.Printf("Failed to get object. %v\n", err)
        return
    }
    defer func() {
        err := getObjectOutput.Body.Close()
        if err != nil {
            fmt.Printf("fail to close result body. %v\n", err)
        }
    }()
    // 将对象保存为本地文件
    p := make([]byte, 1024)
    filepath := "<file-path>"
    file, err := os.Create(filepath)
    if err != nil {
        fmt.Println("fail to create file.", err)
        return
    }
```

```
    defer func() {
        err := file.Close()
        if err != nil {
            fmt.Printf("fail to close file. %v\n", err)
        }
    }()
    for {
        readCount, readErr := getObjectOutput.Body.Read(p)
        _, _ = file.Write(p[:readCount])
        if readErr != nil {
            break
        }
    }
}
```

请求参数

GetObjectInput可设置的参数如下：

参数	类型	说明	是否必要
Bucket	*string	执行本操作的桶名称。	是
IfMatch	*string	用于指定只有在对象的ETag和该参数值匹配的情况下才返回对象数据，否则返回412错误码。	否
IfModifiedSince	*time.Time	用于只有当对象在指定时间后被修改的情况下才返回该对象，否则返回304错误码。	否
IfNoneMatch	*string	用于指定只有在对象的ETag和该参数值不匹配的情况下才返回对象数据，否则返回304错误码	否

参数	类型	说明	是否必要
IfUnmodifiedSince	*time.Time	用于仅当对象自指定时间以来未被修改的情况下才返回对象数据，否则返回412错误码。	否
Key	*string	对象的key。	是
PartNumber	*int64	读取对象指定的片段，该参数大于等于1，小于等于10000。	否
Range	*string	下载对象指定范围内的数据（单位：字节），必须是"bytes=first-last"的格式，例如"bytes=0-9"表示前10字节的数据，详情请参见详情请参见RFC2616。	否
ResponseCacheControl	*string	用于设置response的Cache-Control头部字段信息。	否
ResponseContentDisposition	*string	用于设置response的Content-Disposition头部字段信息。	否
ResponseContentEncoding	*string	用于设置response的Content-Encoding头部字段信息。	否
ResponseContentLanguage	*string	用于设置response的Content-Language头部字段信息。	否
ResponseContentType	*string	用于设置response的Content-Type头部字段信息。	否
ResponseExpires	*time.Time	用于设置response的Expires头部字段信息。	否
VersionId	*string	当bucket开启版本控制的时候，用于指定获取指定版本的对象数据，当不指定该参数的时候，默认获取最新版本的对象数据。	否

返回结果

GetObjectOutput返回的属性如下：

参数	类型	说明
Body	io.ReadCloser	对象的数据。
ContentLength	*int64	对象数据的长度，单位为字节。
ContentType	*string	数据的标准MIME类型。
ETag	*string	对象的Entity Tag。
LastModified	*time.Time	最后修改对象的时间。
TagCount	*int64	对象标签的数量。
VersionId	*string	对象的版本ID。
StorageClass	*string	对象的存储类型。

限定条件下载

功能说明

下载对象时可以设置对象的Etag和修改时间是否匹配指定值等条件，当满足限定条件时则进行下载，否则返回错误码。

代码示例

```
func GetObject(svc *s3.S3) {
    format := "2006/01/02 15:04:05"
    mtime, _ := time.Parse(format, "2023/06/15 00:00:00")

    getObjectInput := &s3.GetObjectInput{
        Bucket: aws.String("<your-bucket-name>"),
        Key:    aws.String("<your-object-key>"),
        // 当对象Etag等于指定值时才下载
        IfMatch: aws.String("781e5e245d69b566979b86e28d23f2c7"),
        // 当对象的修改时间大于指定值时才下载
        IfModifiedSince: &mtime,
    }

    getObjectOutput, err := svc.GetObject(getObjectInput)
    if err != nil {
        fmt.Printf("Failed to get object. %v\n", err)
        return
    }
    defer func() {
        err := getObjectOutput.Body.Close()
        if err != nil {
            fmt.Printf("fail to close result body. %v\n", err)
        }
    }()

    // 将对象保存为本地文件
    p := make([]byte, 1024)
    filepath := "<file-path>"
    file, err := os.Create(filepath)
    if err != nil {
```

```
        fmt.Println("fail to create file.", err)
        return
    }
    defer func() {
        err := file.Close()
        if err != nil {
            fmt.Printf("fail to close file. %v\n", err)
        }
    }()
    for {
        readCount, readErr := getObjectOutput.Body.Read(p)
        _, _ = file.Write(p[:readCount])
        if readErr != nil {
            break
        }
    }
}
```

请求参数

GetObjectInput可设置的参数如下：

参数	类型	说明	是否必要
Bucket	*string	执行本操作的桶名称。	是
IfMatch	*string	用于指定只有在对象的ETag和该参数值匹配的情况下才返回对象数据，否则返回412错误码。	否
IfModifiedSince	*time.Time	用于只有当对象在指定时间后被修改的情况下才返回该对象，否则返回304错误码。	否
IfNoneMatch	*string	用于指定只有在对象的ETag和该参数值不匹配的情况下才返回对象数据，否则返回304错误码	否

参数	类型	说明	是否必要
IfUnmodifiedSince	*time.Time	用于仅当对象自指定时间以来未被修改的情况下才返回对象数据，否则返回412错误码。	否
Key	*string	对象的key。	是
PartNumber	*int64	读取对象指定的片段，该参数大于等于1，小于等于10000。	否
Range	*string	下载对象指定范围内的数据（单位：字节），必须是"bytes=first-last"的格式，例如"bytes=0-9"表示前10字节的数据，详情请参见详情请参见RFC2616。	否
ResponseCacheControl	*string	用于设置response的Cache-Control头部字段信息。	否
ResponseContentDisposition	*string	用于设置response的Content-Disposition头部字段信息。	否
ResponseContentEncoding	*string	用于设置response的Content-Encoding头部字段信息。	否
ResponseContentLanguage	*string	用于设置response的Content-Language头部字段信息。	否
ResponseContentType	*string	用于设置response的Content-Type头部字段信息。	否
ResponseExpires	*time.Time	用于设置response的Expires头部字段信息。	否
VersionId	*string	当bucket开启版本控制的时候，用于指定获取指定版本的对象数据，当不指定该参数的时候，默认获取最新版本的对象数据。	否

返回结果

GetObjectOutput返回的属性如下：

参数	类型	说明
Body	io.ReadCloser	对象的数据。
ContentLength	*int64	对象数据的长度，单位为字节。
ContentType	*string	数据的标准MIME类型。
ETag	*string	对象的Entity Tag。
LastModified	*time.Time	最后修改对象的时间。
TagCount	*int64	对象标签的数量。
VersionId	*string	对象的版本ID。
StorageClass	*string	对象的存储类型。

复制对象

功能说明

复制对象操作用于创建一个已经在对象存储中的对象。复制对象可以拷贝单个最大为5GB的对象，如果需要拷贝更大的对象，可以使用复制段操作。执行复制对象操作，必须具有对被拷贝对象的READ权限和对目标桶的WRITE权限。

拷贝生成的对象默认保留原对象的元数据信息，也可以在复制对象操作中指定新的元数据。拷贝生成的对象不会保留原来的ACL信息，该对象默认是发起复制对象操作的用户私有的。

复制对象操作默认拷贝原对象的当前版本数据，如果需要拷贝原对象的指定版本，可以在复制对象时加入version id来拷贝指定的对象版本，如果原对象的version id为删除标记，则不会被拷贝。如果目标bucket开启了版本控制，那么拷贝生成的对象会有一个与原对象不同的唯一的version id，并且会在响应头部字段 x-amz-version-id中返回该version id。

代码示例

```
func CopyObject(svc *s3.S3) {
    copyObjectInput := &s3.CopyObjectInput{
        Bucket:     aws.String("<your-bucket-name>"),
        CopySource: aws.String("<copy-source>"),
        Key:        aws.String("<your-object-key>"),
    }

    copyObjectOutput, err := svc.CopyObject(copyObjectInput)
    if err != nil {
        fmt.Printf("fail to copy object. %v\n", err)
    }
    fmt.Println(copyObjectOutput)
}
```

通过CopyObjectRequest操作：

CopyObjectRequest操作首先生成一个"request.Request"对象，该对象是一个执行CopyObject操作的请求。通过调用Request对象的Send方法来完成拷贝对象操作。该方法可以生成定制化的请求，例如自定义请求头部请求超时重试设置。

```
func CopyObjectRequest(svc *s3.S3) {
    copyObjectInput := &s3.CopyObjectInput{
        Bucket:     aws.String("<your-bucket-name>"),
        CopySource: aws.String("<copy-source>"),
        Key:        aws.String("<your-object-key>"),
    }
    req, copyObjectOutput := svc.CopyObjectRequest(copyObjectInput)

    err := req.Send()
    if err != nil {
        fmt.Printf("fail to copy object. %v\n", err)
    } else {
        fmt.Println(copyObjectOutput)
    }
}
```

请求参数

CopyObjectInput可设置的参数如下：

参数	类型	说明	是否必要
ACL	*string	拷贝后对象的预定义的标准ACL信息，例如private，public-read，public-read-write等。	否
Bucket	*string	存放拷贝生成对象的桶名称。	是

参数	类型	说明	是否必要
CopySource	*string	URL格式的拷贝对象数据来源，包含了桶名称和对象key的信息，二者之间使用正斜杆（/）分割，versionId可选参数用于指定原对象的版本。例如，"foo/boo?versionId=11111"表示拷贝foo桶中的boo对象，其版本id为11111。如果不指定versionId参数，则默认拷贝当前版本的对象数据。	是
GrantFullControl	*string	用于为用户配置被拷贝对象的FULL_CONTROL权限信息。	否
GrantRead	*string	用于为用户配置被拷贝对象的READ权限信息。	否
GrantReadACP	*string	用于为用户配置被拷贝对象的READ_ACP权限信息。	否
GrantWriteACP	*string	用于为用户配置被拷贝对象的WRITE_ACP权限信息。	否
Key	*string	拷贝生成对象对应的key。	是
Metadata	map[string]*string	拷贝生成对象的元数据信息。	否
MetadataDirective	*string	指定拷贝生成的对象的元数据信息来自被拷贝对象，还是来自请求中附带的信息。	否
Tagging	*string	拷贝生成对象的标签信息，必须是URL请求参数的形式。例如，"Key1=Value1"。	否
TaggingDirective	*string	用于说明拷贝生成对象的标签信息是来自被拷贝对象，还是来自请求中附带的信息。	否
WebsiteRedirectLocation	*string	如果bucket被配置用于提供网站的静态数据，该参数可以用于设置访问对象时候重定向到当前bucket下的其他对象或者外部的URL。	否

返回结果

CopyObjectOutput返回的属性如下：

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

删除对象

功能说明

删除对象操作用于删除指定的一个对象。对于开启版本控制的桶执行 删除对象操作时，如果未指定version id，则保留对象的当前版本，并插入删除标记（Delete Marker）。如果指定version id，则永久删除该指定版本的对象。如果在未指定version id的情况下执行删除对象操作时，默认仅作用于对象的当前版本，但不会直接删除该对象的当前版本，而是插入一个删除标记（Delete Marker），并保留原来的当前版本。当访问对象时，服务端会检测到当前版本为删除标记，并返回404 Not Found。

代码示例

```
func DeleteObject(svc *s3.S3) {
    deleteObjectInput := &s3.DeleteObjectInput{
        Bucket: aws.String("<your-bucket-name>"),
        Key:    aws.String("<your-object-key>"),
    }

    deleteObjectOutput, err := svc.DeleteObject(deleteObjectInput)
    if err != nil {
        fmt.Printf("Failed to delete object. %v\n", err)
    } else {
        fmt.Println(deleteObjectOutput)
    }
}
```

通过DeleteObjectRequest操作：

DeleteObjectRequest操作首先生成一个"request.Request"对象，该对象是一个执行DeleteObject操作的请求。通过调用Request对象的Send方法完成删除对象的操作。该方法可以生成定制化的请求，例如自定义请求头部请求超时重试设置。

```
func DeleteObjectRequest(svc *s3.S3) {
    deleteObjectInput := &s3.DeleteObjectInput{
        Bucket: aws.String("<your-bucket-name>"),
        Key:    aws.String("<your-object-key>"),
    }
    req, deleteObjectOutput := svc.DeleteObjectRequest(deleteObjectInput)

    err := req.Send()
    if err != nil {
        fmt.Printf("fail to delete object. %v\n", err)
    } else {
        fmt.Println(deleteObjectOutput)
    }
}
```

请求参数

DeleteObjectInput可设置的参数如下：

参数	类型	说明	是否必要
Bucket	*string	执行本操作的桶名称。	是
Key	*string	对象的key。	是
VersionId	*string	用于指定要删除对象的versionId	否

返回结果

DeleteObjectOutput返回的属性如下：

参数	类型	说明
DeleteMarker	*bool	有效值为true。在桶开启版本控制的情况下，执行DeleteObject 操作时如果未指定对象的版本Id，会创建删除标记，此时DeleteMarker为true。如果指定对象的版本Id来永久删除指定对象版本时，若该版本Id是删除标记，则DeleteMarker为true。
VersionId	*string	执行DeleteObject操作时如果未指定对象的版本Id，会创建删除标记，VersionId为删除标记的版本Id。 如果指定版本Id来永久删除对象指定版本时，返回的VersionId为对象的版本Id。

批量删除对象

功能说明

（本接口目前仅支持部分资源池使用；如需使用，请联系天翼云客服确认。）

DeleteObjects操作可以实现通过一个HTTP请求批量删除多个对象的功能，可以减少发起多起请求去删除大量对象的花销。DeleteObjects操作发起一个包含了最多1000个key的删除请求，对象存储服务会对相应的对象逐个进行删除，并且将删除成功或者失败的结果通过response返回。如果请求删除的对象不存在，会返回已删除的结果。

DeleteObjects操作返回verbose 和quiet两种response模式。 verbose response是默认的返回模式，该模式的返回结果包含了每个key的删除结果。quiet response返回模式返回的结果仅包含了删除失败的key，对于一个完全成功的删除操作，该返回模式不在相应消息体中返回任何信息。

代码示例

```
func DeleteObjects(svc *s3.S3) {
    deleteObjectsInput := &s3.DeleteObjectsInput{
        Bucket: aws.String("<your-bucket-name>"),
        Delete: &s3.Delete{
            Objects: []*s3.ObjectIdentifier{
                {
                    Key: aws.String("exampleKey1"),
                },
                {
                    Key: aws.String("exampleKey2"),
                },
            },
            Quiet: aws.Bool(false),
```

```
        },
    }

    deleteObjectsOutput, err := svc.DeleteObjects(deleteObjectsInput)
    if err != nil {
        fmt.Printf("Failed to delete objects. %v\n", err)
    } else {
        fmt.Println(deleteObjectsOutput)
    }
}
```

通过DeleteObjectsRequest操作：

DeleteObjectsRequest操作首先生成一个"request.Request"对象，该对象是一个执行DeleteObjects操作的请求。通过调用Request对象的Send方法完成批量删除对象的操作。该方法可以生成定制化的请求，例如自定义请求头部请求超时重试设置。

```
func DeleteObjectsRequest(svc *s3.S3) {
    deleteObjectsInput := &s3.DeleteObjectsInput{
        Bucket: aws.String("<your-bucket-name>"),
        Delete: &s3.Delete{
            Objects: []*s3.ObjectIdentifier{
                {
                    Key: aws.String("exampleKey1"),
                },
                {
                    Key: aws.String("exampleKey2"),
                },
            },
            Quiet: aws.Bool(false),
        },
    }
    req, deleteObjectsOutput := svc.DeleteObjectsRequest(deleteObjectsInput)

    err := req.Send()
    if err != nil {
        fmt.Printf("fail to delete objects. %v\n", err)
    } else {
        fmt.Println(deleteObjectsOutput)
    }
}
```

请求参数

DeleteObjectsInput可设置的参数如下：

参数	类型	说明	是否必要
Bucket	*string	执行本操作的桶名称。	是
Delete	*Delete	封装了要删除对象信息（Key、VersionId）和删除模式等信息	是

返回结果

DeleteObjectsOutput返回的属性如下：

参数	类型	说明
Deleted	[]*DeletedObject	被删除对象信息的数组，数组中每一项包含了一个被成功删除的对象的信息，包括删除标记、对象key和版本id等信息。
Errors	[]*Error	删除失败的对象信息的数组，数组中每一项包含了一个删除失败的对象的信息，包括错误码、

获取对象元数据

功能说明

获取对象元数据操作用于获取对象的元数据信息。执行该操作需要具有对该对象的READ权限，请求参数与下载对象操作一样，区别在于获取对象元数据操作响应体中没有body部分。

代码示例

```
func HeadObject(svc *s3.S3) {
    headObjectInput := &s3.HeadObjectInput{
        Bucket: aws.String("<your-bucket-name>"),
        Key:    aws.String("<your-object-key>"),
    }

    headObjectOutput, err := svc.HeadObject(headObjectInput)
    if err != nil {
        fmt.Printf("fail to head object. %v\n", err)
        return
    }
    fmt.Printf("object info: %v\n", headObjectOutput)
}
```

通过HeadObjectRequest操作：

HeadObjectRequest操作首先生成一个"request.Request"对象，该对象是一个执行HeadObject操作的请求。通过调用Request对象的Send方法完成获取对象元数据信息的操作。该方法可以生成定制化的请求，例如自定义请求头部请求超时重试设置。

```
func HeadObjectRequest(svc *s3.S3) {
    headObjectInput := &s3.HeadObjectInput{
        Bucket: aws.String("<your-bucket-name>"),
        Key:    aws.String("<your-object-key>"),
    }
    req, headObjectOutput := svc.HeadObjectRequest(headObjectInput)

    err := req.Send()
    if err != nil {
        fmt.Printf("fail to head object. %v\n", err)
    } else {
        fmt.Println(headObjectOutput)
    }
}
```

请求参数

HeadObjectInput可设置的参数如下：

参数	类型	说明	是否必要
Bucket	*string	执行本操作的桶名称。	是
IfMatch	*string	用于指定只有在对象的ETag和该参数值匹配的情况下才返回对象数据，否则返回412错误码。	否
IfModifiedSince	*time.Time	用于只有当对象在指定时间后被修改的情况下才返回该对象，否则返回304错误码。	否
IfNoneMatch	*string	用于指定只有在对象的ETag和该参数值不匹配的情况下才返回对象数据，否则返回304错误码	否
IfUnmodifiedSince	*time.Time	用于仅当对象自指定时间以来未被修改的情况下才返回对象数据，否则返回412错误码。	否
Key	*string	对象的key。	是
PartNumber	*int64	读取对象指定的片段，该参数大于等于1，小于等于10000。	否
Range	*string	指定对象的数据范围（单位：字节），必须是"bytes=first-last"的格式，例如"bytes=0-9"表示前10字节的数据，详情请参见RFC2616。	否
VersionId	*string	当bucket开启版本控制的时候，用于指定获取指定版本的对象数据，当不指定该参数的时候，默认获取最新版本的对象数据。	否

返回结果

HeadObjectOutput返回的属性如下：

参数	类型	说明
ContentLength	*int64	本次请求返回对象数据的大小（单位：字节）。
ContentType	*string	对象文件格式的标准MIME类型
ETag	*string	对象的Entity Ttag
LastModified	*time.Time	最近一次修改对象的时间。
VersionId	*string	对象最新的版本ID。
StorageClass	*string	对象的存储类型。

设置对象ACL

功能说明

设置对象ACL操作可以为对象存储服务中的对象设置访问权限，设置对象ACL操作需要具有对象的WRITE_ACP权限，执行操作的时候，要么使用封装好的ACL数据类型传入ACL信息，要么以字符串的形式详细描述ACL信息。

对象的访问权限说明：

权限类型	说明
READ	可以读取对象的数据和元数据信息。
READ_ACP	可以读取对象的ACL信息。对象的拥有者默认具有对象的READ_ACP权限。
WRITE	可以修改原有对象数据和删除对象。
WRITE_ACP	可以修改对象的ACL信息，授予该权限相当于授予FULL_CONTROL权限，因为具有WRITE_ACP权限的用户可以配置对象的任意权限。对象的拥有者默认具有WRITE_ACP权限。
FULL_CONTROL	同时授予READ、READ_ACP、WRITE和WRITE_ACP。

代码示例

```
func PutObjectAcl(svc *s3.S3) {
    bucket := "<your-bucket-name>"
    key := "<your-object-key>"
    permission := "READ" // FULL_CONTROL、WRITE、WRITE_ACP、READ、READ_ACP
    granteeDisplayName := "<your-display-name>"
    granteeId := "<your-user-id>"
    userType := "CanonicalUser"
    // 获取当前acl
    getObjectAclOutput, err := svc.GetObjectAcl(&s3.GetObjectAclInput{
        Bucket: aws.String(bucket),
        Key:    aws.String(key),
    })
    if err != nil {
        fmt.Printf("fail to get acl of object %v, %v\n", key, err)
        os.Exit(1)
    }
    // 创建一个新的授权信息
    var newGrantee = s3.Grantee{
        Type:        aws.String(userType),
        DisplayName: aws.String(granteeDisplayName),
        ID:          aws.String(granteeId),
    }
    var newGrant = s3.Grant{Grantee: &newGrantee, Permission: &permission}
    grants := getObjectAclOutput.Grants
    owner := *getObjectAclOutput.Owner.DisplayName
    ownerId := *getObjectAclOutput.Owner.ID
    grants = append(grants, &newGrant)
    // 设置对象的ACL
    putObjectAclInput := &s3.PutObjectAclInput{
        AccessControlPolicy: &s3.AccessControlPolicy{
            Grants: grants,
            Owner: &s3.Owner{
                DisplayName: &owner,
```

```
                ID:          &ownerId,
            },
        },
        Bucket: aws.String(bucket),
        Key:    aws.String(key),
    }

    putObjectAclOutput, err := svc.PutObjectAcl(putObjectAclInput)
    if err != nil {
        fmt.Printf("fail to put objec acl. %v\n", err)
        return
    }
    fmt.Println(putObjectAclOutput)
}
```

通过PutObjectAclRequest操作：

PutObjectAclRequest操作首先生成一个"request.Request"对象，该对象是一个执行PutObjectAcl操作的请求。通过调用Request对象的Send方法来设置对象的ACL信息。该方法可以生成定制化的请求，例如自定义请求头部请求超时重试设置。

```
func PutObjectAclRequest(svc *s3.S3) {
    bucket := "<your-bucket-name>"
    key := "<your-object-key>"
    permission := "READ_ACP" // FULL_CONTROL、WRITE、WRITE_ACP、READ、READ_ACP
    granteeDisplayName := "<your-display-name>"
    granteeId := "<your-user-id>"
    userType := "CanonicalUser"
    // 获取当前acl
    currentACL, err := svc.GetObjectAcl(&s3.GetObjectAclInput{
        Bucket: aws.String(bucket),
        Key:    aws.String(key),
    })
    if err != nil {
        fmt.Printf("fail to get acl of object %v, %v\n", key, err)
        os.Exit(1)
    }
    // 创建一个新的授权信息
    var newGrantee = s3.Grantee{
        Type:         aws.String(userType),
        DisplayName: aws.String(granteeDisplayName),
        ID:          aws.String(granteeId),
    }
    var newGrant = s3.Grant{Grantee: &newGrantee, Permission: &permission}
    grants := currentACL.Grants
    owner := *currentACL.Owner.DisplayName
    ownerId := *currentACL.Owner.ID
    grants = append(grants, &newGrant)
    // 设置对象的ACL
    putObjectAclInput := &s3.PutObjectAclInput{
        AccessControlPolicy: &s3.AccessControlPolicy{
            Grants: grants,
            Owner: &s3.Owner{
                DisplayName: &owner,
                ID:          &ownerId,
            },
        },
```

```
        Bucket: aws.String(bucket),
        Key:    aws.String(key),
    }
    req, putObjectAclOutput := svc.PutObjectAclRequest(putObjectAclInput)

    err = req.Send()
    if err != nil {
        fmt.Printf("fail to put object ACL. %v\n", err)
    } else {
        fmt.Println(putObjectAclOutput)
    }
}
```

请求参数

PutObjectAclInput可设置的参数如下：

参数	类型	说明	是否必要
ACL	*string	配置对象预定义的标准ACL信息，例如private，public-read，public-read-write等。	否
AccessControlPolicy	*AccessControlPolicy	配置该对象对于每个用户的ACL授权信息。	否
Bucket	*string	执行本操作的桶名称。	是
GrantFullControl	*string	配置对象的FULL_CONTROL权限信息。	否

参数	类型	说明	是否必要
GrantRead	*string	配置对象的READ权限信息。	否
GrantReadACP	*string	配置对象ACL的READ权限信息。	否
GrantWrite	*string	配置对象的WRITE权限信息。	否
GrantWriteACP	*string	配置对象ACL的WRITE权限信息。	否
Key	*string	设置ACL信息的对象的key。	是
VersionId	*string	设置ACL信息的对象的versionId	否

获取对象ACL

功能说明

获取对象ACL操作可以获取对象的access control list（ACL）信息。对一个对象执行获取ACL操作需要具有READ_ACP权限。

代码示例

```
func GetObjectAcl(svc *s3.S3) {
    getObjectAclInput := &s3.GetObjectAclInput{
        Bucket: aws.String("<your-bucket-name>"),
        Key:    aws.String("<your-object-key>"),
    }

    getObjectAclOutput, err := svc.GetObjectAcl(getObjectAclInput)
    if err != nil {

        return
    }
    fmt.Println(getObjectAclOutput)
}
```

通过GetObjectAclRequest操作：

GetObjectAclRequest操作首先生成一个"request.Request"对象，该对象是一个执行GetObjectAcl操作的请求。通过调用Request对象的Send方法来获取对象的ACL信息。该方法可以生成定制化的请求，例如自定义请求头部请求超时重试设置。

```
func GetObjectAclRequest(svc *s3.S3) {
    getObjectAclInput := &s3.GetObjectAclInput{
        Bucket: aws.String("<your-bucket-name>"),
        Key:    aws.String("<your-object-key>"),
    }
    req, getObjectAclOutput := svc.GetObjectAclRequest(getObjectAclInput)

    err := req.Send()
    if err != nil {
```

```
        fmt.Printf("fail to get object ACL. %v\n", err)
    } else {
        fmt.Println(getObjectAclOutput)
    }
}
```

请求参数

GetObjectAclInput可设置的参数如下：

参数	类型	说明	是否必要
Bucket	*string	对象所在桶的名称。	是
Key	*string	对象的key。	是
VersionId	*string	对象的版本Id	否

返回结果

GetObjectAclOutput返回的属性如下：

参数	类型	说明
Grants	[]*Grant	授权信息数组，数组中每一项描述了权限被授予者的用户类型、用户名、邮箱地址、用户Id、用户组和授予权限类型等信息。
Owner	*Owner	对象所在桶的拥有者信息，包含了用户名和用户Id等信息。

设置对象属性

功能说明

在上传对象时可以设置对象的属性，对象属性以key-value的形式描描述。

代码示例

```
func PutObjectWithMetadata(svc *s3.S3) {
    putObjectInput := &s3.PutObjectInput{
        Body:   strings.NewReader("<object-data>"),
        Bucket: aws.String("<your-bucket-name>"),
        Key:    aws.String("<your-object-key>"),
        Metadata: map[string]*string{
            "<metadata1>": aws.String("<value1>"),
            "<metadata2>": aws.String("<value2>"),
        },
    }
    putObjectOutput, err := svc.PutObject(putObjectInput)
    if err != nil {
        fmt.Printf("Failed to put object. %v", err)
    } else {
        fmt.Println(putObjectOutput)
    }
}
```

请求参数

PutObjectInput可设置的参数如下：

参数	类型	说明	是否必要
ACL	*string	配置上传对象的预定义的标准ACL信息，例如private，public-read，public-read-write等。	否
Body	io.ReadSeeker	对象的数据。	是
Bucket	*string	执行本操作的桶名称。	是
ContentLength	*int64	说明请求body的长度（单位：字节），该参数可以在body长度不能被自动识别的情况下设置。	否
ContentMD5	*string	base64编码的128位MD5值，不包含请求头部的信息	否
GrantFullControl	*string	用于自定义用户对此对象的FULL_CONTROL权限信息。	否
GrantRead	*string	用于自定义用户对此对象的READ权限信息。	否
GrantReadACP	*string	用于自定义用户对此对象的READ_ACP权限信息。	否
GrantWrite	*string	用于自定义用户对此对象的WRITE权限信息。	否
GrantWriteACP	*string	用于自定义用户对此对象的WRITE_ACP权限信息。	否
Key	*string	上传文件到对象存储服务后对应的key。PutObject操作支持将文件上传至文件夹，如需要将对象上传至"/folder"文件下，只需要设置Key="/folder/{exampleKey}"即可。	是
Metadata	map[string]*string	对象的元数据信息。	否
Tagging	*string	对象的标签信息，必须是URL请求参数的形式。例如，"Key1=Value1"。	否
WebsiteRedirectLocation	*string	如果bucket被配置用于提供网站的静态数据，该参数可以用于设置访问对象时候重定向到当前bucket下的其他对象或者外部的URL。	否

返回结果

PutObjectOutput返回的属性如下：

参数	类型	说明
ETag	*string	上传对象后对应的Entity Tag
VersionId	*string	上传对象后相应的版本Id。

设置对象标签

功能说明

设置对象标签操作可以为对象设置标签，标签是一个键值对，每个对象最多可以有10个标签。桶的拥有者默认拥有给桶中的对象设置标签的权限，并且可以将权限授予其他用户。

每次执行设置对象标签操作会覆盖对象已有的标签信息。每个对象最多可以设置10个标签，标签Key和Value区分大小写，并且Key不可重复。每个标签的Key长度不超过128字节，Value长度不超过255字节。通过HTTP header的方式设置标签且标签中包含任意字符时，需要对标签的Key和Value做URL编码。设置对象标签信息不会更新对象的最新更改时间。

代码示例

```
func PutObjectTagging(svc *s3.S3) {
    putObjectTaggingInput := &s3.PutObjectTaggingInput{
        Bucket: aws.String("<your-bucket-name>"),
        Key:    aws.String("<your-object-key>"),
        Tagging: &s3.Tagging{
            TagSet: []*s3.Tag{
                {
                    Key:   aws.String("<key1>"),
                    Value: aws.String("<value1>"),
                },
                {
                    Key:   aws.String("<key2>"),
                    Value: aws.String("<value2>"),
                },
            },
        },
    }
    putObjectTaggingOutput, err := svc.PutObjectTagging(putObjectTaggingInput)
    if err != nil {
        fmt.Printf("fail to put object tagging. %v\n", err)
        return
    }
    fmt.Println(putObjectTaggingOutput)
}
```

通过PutObjectTaggingRequest操作：

PutObjectTaggingRequest操作首先生成一个"request.Request"对象，该对象是一个执行PutObjectTaggingRequest操作的请求。通过调用Request对象的Send方法来设置对象的标签信息。该方法可以生成定制化的请求，例如自定义请求头部请求超时重试设置。

```
func PutObjectTaggingRequest(svc *s3.S3) {
    putObjectTaggingInput := &s3.PutObjectTaggingInput{
        Bucket: aws.String("<your-bucket-name>"),
        Key:    aws.String("<your-object-key>"),
```

```
        Tagging: &s3.Tagging{
            TagSet: []*s3.Tag{
                {
                    Key:   aws.String("<key1>"),
                    Value: aws.String("<value1>"),
                },
                {
                    Key:   aws.String("<key2>"),
                    Value: aws.String("<value2>"),
                },
            },
        },
    }
    req, putObjectTaggingOutput :=
svc.PutObjectTaggingRequest(putObjectTaggingInput)

    err := req.Send()
    if err != nil {
        fmt.Printf("fail to put object tagging. %v\n", err)
    } else {
        fmt.Println(putObjectTaggingOutput)
    }
}
```

请求参数

PutObjectTaggingInput可设置的参数如下：

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

获取对象标签

功能说明

获取对象标签操作可以查询对象的标签信息，标签是一个键值对，每个对象最多可以有10个标签。桶的拥有者默认拥有查询桶中对象的标签的权限，并且可以将权限授予其他用户。

代码示例

```
func GetObjectTagging(svc *s3.S3) {
    getObjectTaggingInput := &s3.GetObjectTaggingInput{
        Bucket: aws.String("<your-bucket-name>"),
        Key:    aws.String("<your-object-key>"),
    }

    getObjectTaggingOutput, err := svc.GetObjectTagging(getObjectTaggingInput)
    if err != nil {
        fmt.Printf("fail to get object tagging. %v\n", err)
        return
    }
    fmt.Println(getObjectTaggingOutput)
}
```

通过GetObjectTaggingRequest操作：

GetObjectTaggingRequest操作首先生成一个"request.Request"对象，该对象是一个执行GetObjectTaggingRequest操作的请求。通过调用Request对象的Send方法来完成获取对象标签信息的操作。该方法可以生成定制化的请求，例如自定义请求头部请求超时重试设置。

```
func GetObjectTaggingRequest(svc *s3.S3) {
    getObjectTaggingInput := &s3.GetObjectTaggingInput{
        Bucket: aws.String("<your-bucket-name>"),
        Key:    aws.String("<your-object-key>"),
    }
    req, getObjectTaggingOutput :=
svc.GetObjectTaggingRequest(getObjectTaggingInput)

    err := req.Send()
    if err != nil {
        fmt.Printf("fail to get object tagging. %v\n", err)
    } else {
        fmt.Println(getObjectTaggingOutput)
    }
}
```

请求参数

GetObjectTaggingInput可设置的参数如下：

参数	类型	说明	是否必要
Bucket	*string	执行本操作的桶名称。	是
Key	*string	想获取标签信息的对象的key。	是
VersionId	*string	想获取标签信息的对象的版本Id	否

返回结果

GetObjectTaggingOutput返回的属性如下：

参数	类型	说明
TagSet	[]*Tag	对象标签信息数组，数组中的每一项包含了标签Key和Value的值。

删除对象标签

功能说明

删除对象标签操作可以删除一个对象的全部标签信息，删除时可以设置版本Id参数删除指定版本对象的标签信息，如果不设置版本Id，则默认删除当前版本的对象标签信息。

代码示例

```
func DeleteObjectTagging(svc *s3.S3) {
    deleteObjectTaggingInput := &s3.DeleteObjectTaggingInput{
        Bucket: aws.String("<your-bucket-name>"),
        Key:    aws.String("<your-object-key>"),
    }

    deleteObjectTaggingOutput, err :=
svc.DeleteObjectTagging(deleteObjectTaggingInput)
    if err != nil {
        fmt.Printf("fail to delete object tagging. %v\n", err)
        return
    }
    fmt.Println(deleteObjectTaggingOutput)
}
```

通过DeleteObjectTaggingRequest操作：

DeleteObjectTaggingRequest操作首先生成一个"request.Request"对象，该对象是一个执行DeleteObjectTagging操作的请求。通过调用Request对象的Send方法来删除对象的标签信息。该方法可以生成定制化的请求，例如自定义请求头部请求超时重试设置。

```
func DeleteObjectTaggingRequest(svc *s3.S3) {
    deleteObjectTaggingInput := &s3.DeleteObjectTaggingInput{
        Bucket: aws.String("<your-bucket-name>"),
        Key:    aws.String("<your-object-key>"),
    }
    req, deleteObjectTaggingOutput :=
svc.DeleteObjectTaggingRequest(deleteObjectTaggingInput)

    err := req.Send()
    if err != nil {
        fmt.Printf("fail to delete object tagging. %v\n", err)
    } else {
        fmt.Println(deleteObjectTaggingOutput)
    }
}
```

请求参数

DeleteObjectTaggingInput可设置的参数如下：

参数	类型	说明	是否必要
Bucket	*string	执行本操作的桶名称。	是
Key	*string	被删除标签信息的对象的key。	是
VersionId	*string	被删除标签信息的对象的versionId	否

服务端加密

功能说明

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

代码示例

```
func PutObjectWithEncryption(svc *s3.S3) {
    putObjectInput := &s3.PutObjectInput{
        Body:                 strings.NewReader("<object-data>"),
        Bucket:               aws.String("<your-bucket-name>"),
        Key:                  aws.String("<your-object-key>"),
        ServerSideEncryption: aws.String("AES256"),
    }
    putObjectOutput, err := svc.PutObject(putObjectInput)
    if err != nil {
        fmt.Printf("Failed to put object. %v", err)
    } else {
        fmt.Println(putObjectOutput)
    }
}
```

请求参数

PutObjectInput可设置的参数如下：

参数	类型	说明	是否必要
ACL	*string	配置上传对象的预定义的标准ACL信息，例如private，public-read，public-read-write等。	否
Body	io.ReadSeeker	对象的数据。	是
Bucket	*string	执行本操作的桶名称。	是
ContentLength	*int64	说明请求body的长度（单位：字节），该参数可以在body长度不能被自动识别的情况下设置。	否
ContentMD5	*string	base64编码的128位MD5值，不包含请求头部的信息	否
GrantFullControl	*string	用于自定义用户对此对象的FULL_CONTROL权限信息。	否
GrantRead	*string	用于自定义用户对此对象的READ权限信息。	否
GrantReadACP	*string	用于自定义用户对此对象的READ_ACP权限信息。	否
GrantWrite	*string	用于自定义用户对此对象的WRITE权限信息。	否
GrantWriteACP	*string	用于自定义用户对此对象的WRITE_ACP权限信息。	否
Key	*string	上传文件到对象存储服务后对应的key。PutObject操作支持将文件上传至文件夹，如需要将对象上传至"/folder"文件下，只需要设置Key="/folder/{exampleKey}"即可。	是
Metadata	map[string]*string	对象的元数据信息。	否
ServerSideEncryption	*string	指定服务端采用的加密算法，如AES256、SM4。	否
Tagging	*string	对象的标签信息，必须是URL请求参数的形式。例如，"Key1=Value1"。	否
WebsiteRedirectLocation	*string	如果bucket被配置用于提供网站的静态数据，该参数可以用于设置访问对象时候重定向到当前bucket下的其他对象或者外部的URL。	否

返回结果

PutObjectOutput返回的属性如下：

参数	类型	说明
ETag	*string	上传对象后对应的Entity Tag
VersionId	*string	上传对象后相应的版本Id。
ServerSideEncryption	*string	返回对象数据加密的算法名称。

生成预签名上传链接

功能说明

本接口能够为一个指定对象生成一个预签名的上传链接，访问该链接可以直接上传该对象。

代码示例

以下代码演示如何为一个指定对象生成一个预签名的上传链接。

```
func generatePresignedPut(svc *s3.S3) {
    req, _ := p.svc.PutObjectRequest(&s3.PutObjectInput{
        Bucket: aws.String("<your-bucket-name>"),
        Key:    aws.String("<your-object-key>"),
    })
    expires := 5 * time.Minute
    urlStr, err := req.Presign(expires)
    if err != nil {
        fmt.Println("Failed to generate presigned URL:", err)
    }else {
        fmt.Println("Generated upload URL:", urlStr)
    }
}
```

通过该预签名URL，可以直接将文件上传到指定的桶：

```
func (p *S3Demo) putObjUsingPresignedUrl(url string, filePath string) error {
    file, err := os.Open(filePath)
    if err != nil {
        return fmt.Errorf("failed to open file %s: %w", filePath, err)
    }
    defer file.Close()

    // 读取文件内容并获取长度
    fileData, err := ioutil.ReadAll(file)
    if err != nil {
        return fmt.Errorf("failed to read file data: %w", err)
    }
    contentLength := len(fileData)

    // 创建 PUT 请求
    req, err := http.NewRequest("PUT", url, bytes.NewReader(fileData))
    if err != nil {
        return fmt.Errorf("failed to create request: %w", err)
    }

    // 设置 Content-Length
```

```
    req.Header.Set("Content-Length", fmt.Sprintf("%d", contentLength))
    req.ContentLength = int64(contentLength)

    client := &http.Client{}
    resp, err := client.Do(req)
    if err != nil {
        return fmt.Errorf("failed to upload file: %w", err)
    }
    defer resp.Body.Close()

    if resp.StatusCode != http.StatusOK {
        bodyBytes, _ := io.ReadAll(resp.Body)
        return fmt.Errorf("failed to upload file, status: %s, response: %s",
resp.Status, string(bodyBytes))
    }

    fmt.Println("Successfully uploaded file to:", url)
    return nil
}
```

请求参数

参数	类型	说明	是否必要
Bucket	*string	bucket的名称	是
Key	*string	对象的key	是
Expires	time.Duration	超时时间	是

返回结果

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

生成预签名下载链接

功能说明

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

代码示例

以下代码演示如何为一个指定对象生成一个预签名的下载链接。

```
func GetObjectPresignedUrl(svc *s3.S3) {
    getObjectInput := &s3.GetObjectInput{
        Bucket: aws.String("<your-bucket-name>"),
        Key:    aws.String("<your-object-key>"),
    }
    req, _ := svc.GetObjectRequest(getObjectInput)
    expires := 10 * time.Minute
    urlStr, err := req.Presign(expires)
    if err != nil {
        fmt.Println("err, ", err)
        return
```

```
    }
    fmt.Println("success, ", urlStr)
}
```

通过该URL，可以直接下载对象。

```
func (p *S3Demo) getObjectUsingPresignedUrl(url string, downloadPath string)
error {
    // 发送 GET 请求以下载对象
    resp, err := http.Get(url)
    if err != nil {
        return fmt.Errorf("failed to download file: %w", err)
    }
    defer resp.Body.Close()

    // 检查响应状态是否成功
    if resp.StatusCode != http.StatusOK {
        bodyBytes, _ := io.ReadAll(resp.Body)
        return fmt.Errorf("failed to download file, status: %s, response: %s",
resp.Status, string(bodyBytes))
    }

    // 创建目标文件以保存下载内容
    outFile, err := os.Create(downloadPath)
    if err != nil {
        return fmt.Errorf("failed to create file %s: %w", downloadPath, err)
    }
    defer outFile.Close()

    // 将响应内容写入文件
    _, err = io.Copy(outFile, resp.Body)
    if err != nil {
        return fmt.Errorf("failed to save file: %w", err)
    }

    fmt.Println("Successfully downloaded file to:", downloadPath)
    return nil
}
```

请求参数

参数	类型	说明	是否必要
Bucket	*string	bucket的名称	是
Key	*string	对象的key	是
Expires	time.Duration	超时时间	是

返回结果

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

分片上传接口

初始化分段上传任务

功能说明

分段上传操作可以将超过5GB的大文件分割后上传，分段上传对象首先需要发起分段上传请求获取一个upload id。

代码示例

```
func CreateMultipartUpload(svc *s3.S3) {
    createMultipartUploadInput := &s3.CreateMultipartUploadInput{
        Bucket: aws.String("<your-bucket-name>"),
        Key:    aws.String("<your-object-key>"),
    }
    createMultipartUploadOutput, err :=
svc.CreateMultipartUpload(createMultipartUploadInput)
    if err != nil {
        fmt.Printf("fail to create multipart upload. %v", err)
    } else {
        fmt.Printf("upload id: %v\n", *createMultipartUploadOutput.UploadId)
    }
}
```

通过CreateMultipartUploadRequest操作获取对象：

CreateMultipartUploadRequest操作首先生成一个"request.Request"对象，该对象是一个执行CreateMultipartUpload操作的请求。通过调用Request对象的Send方法完成初始化分段上传的操作。该方法可以生成定制化的请求，例如自定义请求头部请求超时重试设置。

```
func CreateMultipartUploadRequest(svc *s3.S3) {
    createMultipartUploadInput := &s3.CreateMultipartUploadInput{
        Bucket: aws.String("<your-bucket-name>"),
        Key:    aws.String("<your-object-key>"),
    }
    req, createMultipartUploadOutput :=
svc.CreateMultipartUploadRequest(createMultipartUploadInput)
    err := req.Send()
    if err != nil {
        fmt.Printf("fail to create multipart upload. %v", err)
    } else {
        fmt.Printf("upload id: %v\n", *createMultipartUploadOutput.UploadId)
    }
}
```

请求参数

CreateMultipartUploadInput可设置的参数如下：

参数	类型	说明	是否必要
ACL	*string	配置上传对象的预定义的标准ACL信息，例如private，public-read，public-read-write等	否
Bucket	*string	bucket的名称	是
ContentType	*string	描述上传文件格式的标准MIME类型	否
GrantFullControl	*string	用于自定义用户对此对象的FULL_CONTROL权限信息	否
GrantRead	*string	用于自定义用户对此对象的READ权限信息	否
GrantReadACP	*string	用于自定义用户对此对象的READ_ACP权限信息	否
GrantWrite	*string	用于自定义用户对此对象的WRITE权限信息	否
GrantWriteACP	*string	用于自定义用户对此对象的WRITE_ACP权限信息	否
Key	*string	上传文件到对象存储服务后对应的key	是
Metadata	map[string]*string	对象的元数据信息	否
Tagging	*string	对象的标签信息，必须是URL请求参数的形式。例如，"Key1=Value1"	否
WebsiteRedirectLocation	*string	如果bucket被配置用于提供网站的静态数据，该参数可以用于设置访问对象时候重定向到当前bucket下的其他对象或者外部的URL	否

返回结果

CreateMultipartUploadOutput返回的属性如下：

参数	类型	说明
Bucket	*string	执行分段上传的桶的名称
Key	*string	本次分段上传对象的名称
UploadId	*string	本次生成分段上传任务的id

上传段

功能说明

初始化分段上传任务后，指定分段上传任务的id可以上传分段数据，可以将大文件分割成片段后上传，除了最后一个片段，每个片段的数据大小为5MB~5GB，每个分段上传任务最多上传10000个分段。

代码示例

```
func UploadPart(svc *s3.S3) {
    file, err := os.Open("<file-path>")
    if err != nil {
        fmt.Printf("Unable to open file:%v", err)
        os.Exit(1)
    }
    defer func() {
        err := file.Close()
        if err != nil {
            fmt.Printf("fail to close file. %v\n", err)
        }
    }()

    uploadPartInput := &s3.UploadPartInput{
        Body:       file,
        Bucket:     aws.String("<your-bucket-name>"),
        Key:        aws.String("<your-object-key>"),
        PartNumber: aws.Int64(1),
        UploadId:   aws.String("<your-upload-id>"),
    }
    uploadPartOutput, err := svc.UploadPart(uploadPartInput)
    if err != nil {
        fmt.Printf("fail to upload part. %v\n", err)
    } else {
        fmt.Println(uploadPartOutput)
    }
}
```

通过UploadPartRequest操作获取对象：

UploadPartRequest操作首先生成一个"request.Request"对象，该对象是一个执行UploadPart操作的请求。通过调用Request对象的Send方法完成上传分段的操作。该方法可以生成定制化的请求，例如自定义请求头部请求超时重试设置。

```
func UploadPartRequest(svc *s3.S3) {
    file, err := os.Open("<file-path>")
    if err != nil {
        fmt.Printf("Unable to open file:%v", err)
```

```
        os.Exit(1)
    }
    defer func() {
        err := file.Close()
        if err != nil {
            fmt.Printf("fail to close file. %v\n", err)
        }
    }()

    uploadPartInput := &s3.UploadPartInput{
        Body:       file,
        Bucket:     aws.String("<your-bucket-name>"),
        Key:        aws.String("<your-object-key>"),
        PartNumber: aws.Int64(1),
        UploadId:   aws.String("<your-upload-id>"),
    }
    req, uploadPartOutput := svc.UploadPartRequest(uploadPartInput)
    err = req.Send()
    if err != nil {
        fmt.Printf("fail to upload part. %v\n", err)
    } else {
        fmt.Println(uploadPartOutput)
    }
}
```

请求参数

UploadPartInput可设置的参数如下：

参数	类型	说明	是否必要
Body	io.ReadSeeker	对象的数据	是
Bucket	*string	执行分段上传的桶的名称	是
ContentLength	*int64	说明请求body的长度（单位：字节），该参数可以在body长度不能被自动识别的情况下设置	否
ContentMD5	*string	base64编码的128位MD5值，不包含请求头部的信息	否
Key	*string	上传文件到对象存储服务后对应的key	是
PartNumber	*int64	说明当前数据在文件中所属的片段，大于等于1，小于等于10000	是
UploadId	*string	通过CreateMultipartUpload操作获取的UploadId，与一个分段上传的对象对应	是

返回结果

UploadPartOutput返回的属性如下：

参数	类型	说明
ETag	*string	本次上传片段对应的Entity Tag

合并段

功能说明

合并指定分段上传任务id对应任务中已上传的对象分片，使之成为一个完整的文件。

代码示例

```
func CompleteMultipartUpload(svc *s3.S3) {
    completeMultipartUploadInput := &s3.CompleteMultipartUploadInput{
        Bucket: aws.String("<your-bucket-name>"),
        Key:    aws.String("<your-object-key>"),
        MultipartUpload: &s3.CompletedMultipartUpload{
            Parts: []*s3.CompletedPart{
                {
                    ETag:       aws.String("<etag1>"),
                    PartNumber: aws.Int64(1),
                },
                {
                    ETag:       aws.String("<etag2>"),
                    PartNumber: aws.Int64(2),
                },
            },
        },
        UploadId: aws.String("<your-upload-id>"),
    }

    completeMultipartUploadOutput, err :=
svc.CompleteMultipartUpload(completeMultipartUploadInput)
    if err != nil {
        fmt.Printf("fail to complete multipart upload.%v\n", err)
    } else {
        fmt.Println(completeMultipartUploadOutput)
    }
}
```

通过CompleteMultipartUploadRequest操作获取对象：

CompleteMultipartUploadRequest操作首先生成一个"request.Request"对象，该对象是一个执行CompleteMultipartUpload操作的请求。通过调用Request对象的Send方法完成合并分段的操作。该方法可以生成定制化的请求，例如自定义请求头部请求超时重试设置。

```
func CompleteMultipartUploadRequest(svc *s3.S3) {
    completeMultipartUploadInput := &s3.CompleteMultipartUploadInput{
        Bucket: aws.String("<your-bucket-name>"),
        Key:    aws.String("<your-object-key>"),
        MultipartUpload: &s3.CompletedMultipartUpload{
            Parts: []*s3.CompletedPart{
                {
                    ETag:       aws.String("<etag1>"),
                    PartNumber: aws.Int64(1),
```

```
                },
                {
                    ETag:       aws.String("<etag2>"),
                    PartNumber: aws.Int64(2),
                },
            },
        },
        UploadId: aws.String("<your-upload-id>"),
    }

    req, completeMultipartUploadOutput :=
svc.CompleteMultipartUploadRequest(completeMultipartUploadInput)
    err := req.Send()
    if err != nil {
        fmt.Printf("fail to complete multipart upload. %v\n", err)
    } else {
        fmt.Println(completeMultipartUploadOutput)
    }
}
```

请求参数

CompleteMultipartUploadInput可设置的参数如下：

参数	类型	说明	是否必要
Bucket	*string	执行分段上传的桶的名称	是
Key	*string	上传文件到对象存储服务后对应的key	是
MultipartUpload	*CompletedMultipartUpload	包含了每个已上传的片段的ETag和PartNUmber等信息	否
UploadId	*string	通过CreateMultipartUpload操作获取的UploadId，与一个对象的分段上传对应	是

返回结果

CompleteMultipartUploadOutput返回的属性如下：

参数	类型	说明
Bucket	*string	执行分段上传的桶的名称
ETag	*string	本次上传对象后对应的Entity Tag
Key	*string	上传文件到对象存储服务后对应的key
Location	*string	合并生成对象的URI信息
VersionId	*string	上传对象后相应的版本ID

列举分段上传任务

功能说明

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

代码示例

```
func ListMultipartUploads(svc *s3.S3) {
    listMultipartUploadsInput := &s3.ListMultipartUploadsInput{
        Bucket: aws.String("<your-bucket-name>"),
    }
    listMultipartUploadsOutput, err :=
svc.ListMultipartUploads(listMultipartUploadsInput)
    if err != nil {
        fmt.Printf("fail to list multipart uploads. %v\n", err)
        return
    }
    fmt.Println(listMultipartUploadsOutput)
}
```

通过ListMultipartUploadsRequest操作：

ListMultipartUploadsRequest操作首先生成一个"request.Request"对象，该对象是一个执行ListMultipartUploads操作的请求。通过调用Request对象的Send方法来列出进行的分段上传信息。该方法可以生成定制化的请求，例如自定义请求头部请求超时重试设置。

```
func ListMultipartUploadsRequest(svc *s3.S3) {
    listMultipartUploadsInput := &s3.ListMultipartUploadsInput{
        Bucket: aws.String("<your-bucket-name>"),
    }
    req, listMultipartUploadsOutput :=
svc.ListMultipartUploadsRequest(listMultipartUploadsInput)

    err := req.Send()
    if err != nil {
        fmt.Printf("fail to list multipart uploads. %v\n", err)
    } else {
        fmt.Println(listMultipartUploadsOutput)
    }
}
```

请求参数

ListMultipartUploadsInput可设置的参数如下：

参数	类型	说明	是否必要
Bucket	*string	执行本操作的桶名称	是
Delimiter	*string	与Prefix参数一起用于对对象key进行分组的字符。所有key包含指定的Prefix且第一次出现Delimiter字符之间的对象作为一组。如果没有指定Prefix参数，按Delimiter对所有对象key进行分割，多个对象分割后从对象key开始到第一个Delimiter之间相同的部分形成一组	否

参数	类型	说明	是否必要
KeyMarker	*string	和UploadIdMarker参数一起用于指定返回哪部分分段上传的信息。如果没有设置UploadIdMarker参数，则只返回对象key按照字典顺序排序后位于KeyMarker标识符之后的片段信息。如果设置了UploadIdMarker参数，则会返回对象key等于KeyMarker且UploadId大于UploadIdMarker的片段信息	否
MaxUploads	*int64	用于指定相应消息体中正在进行的分段上传信息的最大数量，最小值为1，默认值和最大值都是1000	否
Prefix	*string	与Delimiter参数一起用于对对象key进行分组的字符。所有key包含指定的Prefix且第一次出现Delimiter字符之间的对象作为一组	否
UploadIdMarker	*string	和KeyMarker参数一起用于指定返回哪部分分段上传的信息，仅当设置了KeyMarker参数的时候有效。设置后返回对象key等于KeyMarker且UploadId大于UploadIdMarker的片段信息	否

返回结果

ListMultipartUploadsOutput返回的属性如下：

参数	类型	说明
Bucket	*string	执行本操作的桶名称
CommonPrefixes	[]*CommonPrefix	当请求中设置了Delimiter和Prefix属性时，所有包含指定的Prefix且第一次出现Delimiter字符的对象key作为一组
Delimiter	*string	与请求中设置的Delimiter一致
IsTruncated	*bool	当为false时表示返回结果中包含了全部符合本次请求查询条件的上传片段信息，否则只返回了数量为MaxUploads个的片段信息
KeyMarker	*string	返回上传片段列表中的起始对象的key
MaxUploads	*int64	本次返回结果中包含的上传片段数量的最大值
NextKeyMarker	*string	当IsTruncated为true时，NextKeyMarker可以作为后续查询已初始化的上传片段请求中的KeyMarker的值
NextUploadIdMarker	*string	当IsTruncated为true时，NextKeyMarker可以作为后续查询已初始化的上传片段请求中的UploadIdMarker的值
Prefix	*string	限定返回片段中对应对象的key必须以Prefix作为前缀

参数	类型	说明
UploadIdMarker	String	上传片段列表中的起始UploadId
Uploads	[]*MultipartUpload	包含了零个或多个已初始化的上传片段信息的数组。数组中的每一项包含了片段初始化时间、分段上传操作发起者、对象key、对象拥有者、存储类型和UploadId等信息

列举已上传的段

功能说明

列举已上传段操作可以列出一个分段上传操作中已经上传完毕但是还未合并的片段信息。请求中需要提供object key和upload id，返回的结果最多包含1000个已上传的片段信息，默认返回1000个，可以通过设置MaxParts参数的值指定返回结果中片段信息的数量。如果已上传的片段信息的数量多于1000个，则返回结果中的IsTruncated字段为true，可用通过设置PartNumberMarker参数获取PartNumber大于该参数的片段信息。

代码示例

```
func ListParts(svc *s3.S3) {
    listPartsInput := &s3.ListPartsInput{
        Bucket:   aws.String("<your-bucket-name>"),
        Key:      aws.String("<your-object-key>"),
        UploadId: aws.String("<your-upload-id>"),
    }
    listPartsOutput, err := svc.ListParts(listPartsInput)
    if err != nil {
        fmt.Printf("fail to list parts. %v\n", err)
        return
    }
    fmt.Println(listPartsOutput)
}
```

通过ListPartsRequest操作：

ListPartsRequest操作首先生成一个"request.Request"对象，该对象是一个执行ListParts操作的请求。通过调用Request对象的Send方法来列出一个分段上传操作中已经上传完毕的片段信息。该方法可以生成定制化的请求，例如自定义请求头部请求超时重试设置。

```
func ListPartsRequest(svc *s3.S3) {
    listPartsInput := &s3.ListPartsInput{
        Bucket:   aws.String("<your-bucket-name>"),
        Key:      aws.String("<your-object-key>"),
        UploadId: aws.String("<your-upload-id>"),
    }
    req, listPartsOutput := svc.ListPartsRequest(listPartsInput)

    err := req.Send()
    if err != nil {
        fmt.Printf("fail to list parts. %v\n", err)
    } else {
        fmt.Println(listPartsOutput)
    }
}
```

请求参数

ListPartsInput可设置的参数如下：

参数	类型	说明	是否必要
Bucket	*string	执行本操作的桶名称	是
Key	*string	分段上传的对象的key	是
MaxParts	*int64	指定返回片段信息的数量，默认值和最大值均为1000	否
PartNumberMarker	*int64	用于指定返回part number大于PartNumberMarker的片段信息	否
UploadId	*string	指定返回该id所属的分段上传的片段信息	是

返回结果

ListPartsOutput返回的属性如下：

参数	类型	说明
Bucket	*string	执行本操作的桶名称
IsTruncated	*bool	当为false时表示返回结果中包含了全部符合本次请求查询条件的上传片段信息，否则只返回了数量为MaxParts个的片段信息
Key	*string	本次分段上传对象的名称
MaxParts	*int64	本次返回结果中包含的上传片段数量的最大值
NextPartNumberMarker	*int64	当IsTruncated为true时，NextPartNumberMarker可以作为后续查询已上传片段请求中的PartNumberMarker的值
Owner	*Owner	分段上传对象的拥有者信息，包含了用户名和Id等信息
PartNumberMarker	*int64	如果本次请求未能列出全部片段信息，该返回结果用于指定下次列举片段请求的起始位置

参数	类型	说明
Parts	[]*Part	包含了已上传片段信息的数组，数组中的每一项包含了该片段的Entity tag、最后修改时间、PartNumber和大小等信息
StorageClass	*string	对象的存储类型
UploadId	*string	本次分段上传操作Id

复制段

功能说明

复制段操作可以从一个已存在的对象中拷贝指定片段的数据，当拷贝的对象大小超过5GB，必须使用复制段操作完成对象的复。除了最后一个片段外，每个拷贝片段的大小范围是[5MB，5GB]。在一个在拷贝大对象之前，需要使用初始化分段上传操作获取一个upload id，在完成拷贝操作之后，需要使用CompleteMultipartUpload操作组装已拷贝的片段成为一个对象。

代码示例

```
var (
    sourceBucket      = "<source-bucket-name>"      //拷贝对象的来源
    key               = "<your-object-key>"         //拷贝对象的key
    destinationBucket = "<your-bucket-name>"        //目标桶名字
)

func UploadPartCopy(svc *s3.S3) {
    var MB int64
    MB = 1024 * 1024
    // 获取被拷贝对象大小
    headObjectOutput, err := svc.HeadObject(&s3.HeadObjectInput{
        Bucket: aws.String(sourceBucket),
        Key:    aws.String(key),
    })
    if err != nil {
        fmt.Printf("fail to head object. %v\n", err)
        return
    }
    objectSize := *headObjectOutput.ContentLength
    // UploadPartCopy操作的对象必须大于5MB
    if objectSize <= 5*MB {
        fmt.Printf("The size of the object must be bigger than 5MB.")
        return
    }
    // 发起一个分段上传请求，获取uploadId
    createMultipartUploadOutput, err :=
svc.CreateMultipartUpload(&s3.CreateMultipartUploadInput{
        Bucket: aws.String(destinationBucket),
        Key:    aws.String(key),
    })
    if err != nil {
        fmt.Printf("fail to create multipart upload. %v\n", err)
        return
    }
```

```
    uploadId := createMultipartUploadOutput.UploadId
    // 记录拷贝分段结果
    var uploadPartCopyResults map[int64]string
    uploadPartCopyResults = make(map[int64]string)
    // 拷贝分段
    var start, end, i int64
    fmt.Printf("object size: %v\n", objectSize)
    for i = 0; i*16*MB+i < objectSize; i++ {
        //每个片段大小为16MB
        start = i*16*MB + i
        if start+16*MB < objectSize {
            end = start + 16*MB
        } else {
            end = objectSize - 1
        }
        fmt.Printf("start: %v, end: %v.\n", start, end)
        if eTag, err := copyAUploadPart(svc, start, end, i+1, uploadId); err ==
nil {
            uploadPartCopyResults[i+1] = *eTag
        }
    }
    // 构建分段复制的part信息
    var partItems []*s3.CompletedPart
    for i, eTag := range uploadPartCopyResults {
        partItems = append(partItems, &s3.CompletedPart{
            ETag:       aws.String(eTag),
            PartNumber: aws.Int64(i),
        })
    }
    // 完成分段上传
    completeMultipartUploadOutput, err :=
svc.CompleteMultipartUpload(&s3.CompleteMultipartUploadInput{
        Bucket:          aws.String(destinationBucket),
        Key:             aws.String(key),
        UploadId:        uploadId,
        MultipartUpload: &s3.CompletedMultipartUpload{Parts: partItems},
    })
    if err != nil {
        fmt.Printf("Failed to CompleteMultipartUpload. %v", err)
        fmt.Println(completeMultipartUploadOutput)
    }
}

func copyAUploadPart(svc *s3.S3, start int64, end int64, partNumber int64,
uploadId *string) (*string, error) {
    copySourceRange := "bytes=" + strconv.FormatInt(start, 10) + "-" +
strconv.FormatInt(end, 10)
    uploadPartCopyInput := &s3.UploadPartCopyInput{
        Bucket:          aws.String(destinationBucket),
        CopySource:      aws.String(sourceBucket + "/" + key),
        CopySourceRange: aws.String(copySourceRange),
        Key:             aws.String(key),
        PartNumber:      aws.Int64(partNumber),
        UploadId:        uploadId,
    }
    uploadPartCopyOutput, err := svc.UploadPartCopy(uploadPartCopyInput)
    if err != nil {
        fmt.Printf("fail to copy upload part. %v\n", err)
```

```
        return nil, err
    }
    return uploadPartCopyOutput.CopyPartResult.ETag, nil
}
```

通过UploadPartCopyRequest操作：

UploadPartCopyRequest操作首先生成一个"request.Request"对象，该对象是一个执行UploadPartCopyRequest操作的请求。通过调用Request对象的Send方法来完成分段拷贝大对象的操作。该方法可以生成定制化的请求，例如自定义请求头部请求超时重试设置。

```
func UploadPartCopyRequest(svc *s3.S3) {
    // 发起一个分段上传请求，获取uploadId
    createMultipartUploadOutput, err :=
svc.CreateMultipartUpload(&s3.CreateMultipartUploadInput{
        Bucket: aws.String("<your-bucket-name>"),
        Key:    aws.String("<your-object-key>"),
    })
    if err != nil {
        fmt.Printf("fail to create multipart upload. %v\n", err)
        return
    }
    uploadId := createMultipartUploadOutput.UploadId
    // 分段复制
    uploadPartCopyInput := &s3.UploadPartCopyInput{
        Bucket:     aws.String("<destination-bucket-name>"),
        CopySource: aws.String("<source-bucket/object-Key>"),
        Key:        aws.String("<your-object-key>"),
        PartNumber: aws.Int64(1),
        UploadId:   uploadId,
    }
    req, uploadPartCopyOutput := svc.UploadPartCopyRequest(uploadPartCopyInput)

    err = req.Send()
    if err != nil {
        fmt.Printf("fail to copy upload part. %v\n", err)
    } else {
        fmt.Print(uploadPartCopyOutput)
    }
}
```

请求参数

UploadPartCopyInput可设置的参数如下：

参数	类型	说明	是否必要
Bucket	*string	存放拷贝生成对象的桶名称	是
CopySource	*string	URL格式的拷贝对象数据来源，包含了桶名称和对象key的信息，二者之间使用正斜杆（/）分割，versionId可选参数用于指定原对象的版本。例如，"foo/boo?versionId=11111"表示拷贝foo桶中的boo对象，其版本id为11111。如果不指定versionId参数，则默认拷贝当前版本的对象数据	是
CopySourceIfMatch	*string	用于指定只有在被拷贝对象的ETag和该参数值匹配的情况下才进行拷贝操作	否
CopySourceIfModifiedSince	*time.Time	用于只有当被拷贝对象在指定时间后被修改的情况下才进行拷贝操作。	否
CopySourceIfNoneMatch	*string	用于指定只有在被拷贝对象的ETag和该参数值不匹配的情况下才进行拷贝操作	否
CopySourceIfUnmodifiedSince	*time.Time	用于仅当对象自指定时间以来未被修改的情况下才返回对象数据，否则返回412错误码	否
CopySourceRange	*string	指定本次分段拷贝的数据范围，必须是"bytes=first-last"的格式，例如"bytes=0-9"表示拷贝原对象中前10字节的数据，只有当拷贝的片段大小大于5MB的时候有效	否
Key	*string	拷贝生成对象的key	是
PartNumber	*int64	说明本次分段拷贝的数据在原对象中所属的部分	是
UploadId	*string	与本次拷贝操作相应的分段上传Id	是

返回结果

UploadPartCopyOutput返回的属性如下：

参数	类型	说明
CopyPartResult	*CopyPartResult	包含拷贝片段的Entity Tag和最后修改时间等信息

取消分段上传任务

功能说明

取消分段上传任务操作用于终止一个分段上传。当一个分段上传被中止后，不会再有数据通过与之相应的upload id上传，同时已经被上传的片段所占用的空间会被释放。执行取消分段上传任务操作后，正在上传的片段可能会上传成功也可能会被中止，所以必要的情况下需要执行多次取消分段上传任务操作去释放全部上传成功的片段所占用的空间。可以通过执行列举已上传段操作来确认所有中止分段上传后所有已上传片段的空间是否被被释放。

代码示例

```
func AbortMultipartUpload(svc *s3.S3, bucket, key, uploadId string) {
    abortMultipartUploadInput := &s3.AbortMultipartUploadInput{
        Bucket:   aws.String("<your-bucket-name>"),
        Key:      aws.String("<your-object-key>"),
        UploadId: aws.String("<your-upload-id>"),
    }

    abortMultipartUploadOutput, err :=
svc.AbortMultipartUpload(abortMultipartUploadInput)
    if err != nil {
        fmt.Printf("fail to AbortMultipartUpload. %v\n", err)
        return
    }
    fmt.Println(abortMultipartUploadOutput)
}
```

通过AbortMultipartUploadRequest操作：

AbortMultipartUploadRequest操作首先生成一个"request.Request"对象，该对象是一个执行AbortMultipartUploadRequest操作的请求。通过调用Request对象的Send方法来中止一个分段上传操作。该方法可以生成定制化的请求，例如自定义请求头部请求超时重试设置。

```
func AbortMultipartUploadRequest(svc *s3.S3) {
    abortMultipartUploadInput := &s3.AbortMultipartUploadInput{
        Bucket:   aws.String("<your-bucket-name>"),
        Key:      aws.String("<your-object-key>"),
        UploadId: aws.String("<your-upload-id>"),
    }

    req, abortMultipartUploadOutput :=
svc.AbortMultipartUploadRequest(abortMultipartUploadInput)
    err := req.Send()
    if err != nil {
        fmt.Printf("fail to AbortMultipartUpload. %v\n", err)
```

```
    } else {
        fmt.Println(abortMultipartUploadOutput)
    }
}
```

请求参数

AbortMultipartUploadInput可设置的参数如下：

参数	类型	说明	是否必要
Bucket	*string	执行本操作的桶名称	是
Key	*string	分段上传的对象的key	是
UploadId	*string	指定需要终止的分段上传的id	是

安全凭证服务(STS)

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

初始化STS服务

```
ak := "<your-access-key>"
sk := "<your-secret-access-key>"
endpoint := "<your-endpoint>"
config := &aws.Config{
    Credentials:      credentials.NewStaticCredentials(ak, sk, ""),
    Endpoint:         aws.String(endpoint),
    S3ForcePathStyle: aws.Bool(true),
    DisableSSL:       aws.Bool(true),
    LogLevel:         aws.LogLevel(aws.LogDebug),
}
sess := session.Must(session.NewSession(config))
svc := sts.New(sess)
```

获取临时token

```
bucket := "<your-bucket-name>"
roleSessionName := "<your-session-name>"
arn := "arn:aws:iam:::role/xxxxxx"
policy := `{"Version":"2012-10-17","Statement":{"Effect":"Allow","Action":
["s3:*"],"Resource":["arn:aws:s3:::%s","arn:aws:s3:::%s/*"]}}`
policy = fmt.Sprintf(policy, bucket, bucket)
fmt.Println("policy: ", policy)
out, err := svc.AssumeRole(&sts.AssumeRoleInput{
    Policy:          aws.String(policy),
    RoleArn:         aws.String(arn),
    RoleSessionName: aws.String(roleSessionName),
})
if err != nil {
```

```
    fmt.Println("err, ", err)
    return
}
fmt.Println("assumeRole success, ", out)
```

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

使用临时token

实现一个CredentialsProvider，支持更新ak/sk和token。

```
type MyCredProvider struct {
    sync.Mutex

    value *credentials.Value
}

func NewMyCredProvider(ak, sk, token string) *MyCredProvider{
    p := &MyCredProvider{
        value: &credentials.Value{
            AccessKeyID:     ak,
            SecretAccessKey: sk,
            SessionToken:    token,
            ProviderName:    "MyCredProvider",
        },
    }
    return p
}

func (p *MyCredProvider) Retrieve() (credentials.Value, error) {
    defer p.Unlock()
    p.Lock()

    return *p.value, nil
}

func (p *MyCredProvider) IsExpired() bool {
    return false
}

// 更新token
func (p *MyCredProvider) UpdateCred(ak, sk, token string) {
    defer p.Unlock()
    p.Lock()
```

```
    p.value.AccessKeyID = ak
    p.value.SecretAccessKey = sk
    p.value.SessionToken = token
}
```

使用临时token

```
ak := "<temporary-access-key>"
sk := "<temporary-secret-access-key>"
token := "<your-session-token>"
endpoint := "<your-endpoint>"
credProvider := NewMyCredProvider(ak, sk, token)
config := &aws.Config{
    Credentials:      credentials.NewCredentials(credProvider),
    Endpoint:         aws.String(endpoint),
    S3ForcePathStyle: aws.Bool(true),
    DisableSSL:       aws.Bool(true),
    LogLevel:         aws.LogLevel(aws.LogDebug),
}
sess := session.Must(session.NewSession(config))
svc := s3.New(sess)
```