天翼云媒体存储（Go）SDK

使用指导书

2023-07-07

天翼云科技有限公司

目 录

1. 环境配置

1.1. 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/ 。

1.2. 配置 go mod

1.3. Windows

打开命令窗口执行

```
`$ go env -w GO111MODULE=on`

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

1.4. 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`
```

2. 初始化 SDK

2.1. 下载 SDK

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

2.2. 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*。

2.3. 获取 Endpoint

EndPoint 的获取方式请查阅 *天翼云**媒体存储**用户使用指南*。

2.4. 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(Acce
ssKey, 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
```

3. 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 a
ddressing,
        // i.e., `http://gzoss.xstore.ctyun.cn/BUCKET/KEY`. By defaul
```

```
t, 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. Def
aults to `false`
        DisableSSL:       aws.Bool(true),
        Credentials:      credentials.NewStaticCredentials(AccessKey,
 SecretAccessKey, ""),
        LogLevel:         aws.LogLevel(aws.LogDebug)}
    sess := session.Must(session.NewSessionWithOptions(session.Option
s{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.Option
s{Config: *conf}))
```

```
    svc := s3.New(sess)

    return svc
}
```

4. 桶相关接口

4.1. 创建桶

4.1.1. 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。

4.1.2. 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, er
r)
    }
    // 等待桶创建
```

```
    fmt.Printf("Waiting for bucket %s to be created...\n", bucketNam
e)
    err = svc.WaitUntilBucketExists(&s3.HeadBucketInput{
        Bucket: aws.String(bucketName),
    })
    if err != nil {
        fmt.Printf("Error occurred while waiting for bucket to be crea
ted, %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(createBucketIn
put)

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

```
        fmt.Println(createBucketOutput)
    }
}
```

4.1.3. 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

4.2. 获取桶列表

4.2.1. 功能说明

获取桶列表可以显示用户全部可用的桶。

4.2.2. 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), a
ws.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(listBucketsInpu
t)

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

4.2.3. 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

4.3. 判断桶是否存在

4.3.1. 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。

4.3.2. 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)
    }
}
```

4.3.3. 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

4.4. 删除桶

4.4.1. 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）。

4.4.2. 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)
    }
}
```

4.4.3. 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

4.5. 列举桶内对象

4.5.1. 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。

4.5.2. 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)
    }
}
```

4.5.3. 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

4.5.4. 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

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

4.6. 列举桶内多版本对象

4.6.1. 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。

4.6.2. 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(listObjec
```

```
tVersionsInput)
    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(li
stObjectVersionsInput)

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

4.6.3. 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

4.6.4. 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

属性名	类型	说明
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
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

4.7. 设置桶 ACL

4.7.1. 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

4.7.2. 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: &permis
sion}

    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 buc
```

```
ket ", 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: &permis
```

```
sion}

    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)
    }
}
```

4.7.3. 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
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。

4.8. 获取桶 ACL

4.8.1. 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。

4.8.2. 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, *owne
r.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(getBucketAclIn
put)
```

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

4.8.3. 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

4.8.4. 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

4.9. 设置桶策略

4.9.1. 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

4.9.2. 代码示例

```
// 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。

```
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)
```

```
    }
}
```

4.9.3. 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

4.10. 获取桶策略

4.10.1. 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。

4.10.2. 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(getBucketPolicy
Input)
    if err != nil {
        fmt.Printf("fail to get policy of bucket. %v\n", err)
        return
    }
    fmt.Printf("policy of bucket: %v\n", *getBucketPolicyOutput.Polic
```

```
y)
}
```

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(getBucke
tPolicyInput)

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

4.10.3. 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

4.10.4. 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

4.11. 删除桶策略

4.11.1. 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。

4.11.2. 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)
    }
}
```

4.11.3. 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

4.12. 设置桶生命周期规则

4.12.1. 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。

4.12.2. 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.PutBucketLifecycleCon
figurationInput{
        Bucket: aws.String("<your-bucket-name>"),
        LifecycleConfiguration: &s3.BucketLifecycleConfiguration{
            Rules: []*s3.LifecycleRule{rule1, rule2, rule3},
        },
    }
    _, err := svc.PutBucketLifecycleConfiguration(putBucketLifecycleC
onfigurationInput)
    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)
    }
}
```

4.12.3. 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

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

4.13. 查看桶生命周期规则

4.13.1. 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。

4.13.2. 代码示例

```
func GetBucketLifecycleConfiguration(svc *s3.S3) {
    getBucketLifecycleConfigurationInput := &s3.GetBucketLifecycleCon
figurationInput{
        Bucket: aws.String("<your-bucket-name>"),
    }
    getBucketLifecycleConfigurationOutput, err := svc.GetBucketLifecy
cleConfiguration(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)
    }
}
```

4.13.3. 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

4.13.4. 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

4.14. 删除桶生命周期规则

4.14.1. 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。

4.14.2. 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(deleteBucketLifecycleI
nput)

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

4.14.3. 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

4.15. 配置桶的 website 配置

4.15.1. 功能说明

设置桶的静态网站配置。

4.15.2. 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)
    }
}
```

4.15.3. 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

4.16. 获取桶的 website 配置

4.16.1. 功能说明

获取桶的静态网站配置。

4.16.2. 代码示例

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


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

4.16.3. 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

4.16.4. 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

4.17. 删除桶的 website 配置

4.17.1. 功能说明

删除桶的静态网站配置。

4.17.2. 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)
    }
}
```

4.17.3. 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

4.18. 设置通的多版本状态

4.18.1. 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。

4.18.2. 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(putBucketVersioningInpu
t)

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

4.18.3. 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

4.19. 获取通的多版本状态

4.19.1. 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。

4.19.2. 代码示例

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

    getBucketVersioningOutput, err := svc.GetBucketVersioning(getBuck
etVersioningInput)
    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)
    }
}
```

4.19.3. 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

4.19.4. 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

4.20. 设置桶的 CORS 配置

4.20.1. 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。

4.20.2. 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)
    }
}
```

4.20.3. 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

4.21. 获取桶的 CORS 配置

4.21.1. 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。

4.21.2. 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)
    }
}
```

4.21.3. 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

4.21.4. 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

4.22. 删除桶的 CORS 配置

4.22.1. 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。

4.22.2. 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)
    }
}
```

4.22.3. 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

4.23. 设置桶标签

4.23.1. 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。

4.23.2. 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)
    }
}
```

4.23.3. 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

4.24. 获取桶标签

4.24.1. 功能说明

获取指定桶的标签信息。

4.24.2. 代码示例

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

```
    getBucketTaggingOutput, err := svc.GetBucketTagging(getBucketTagg
ingInput)
    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(getBuc
ketTaggingInput)

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

4.24.3. 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

4.24.4. 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

4.25. 删除桶标签

4.25.1. 功能说明

删除指定桶的全部标签。

4.25.2. 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)
    }
}
```

4.25.3. 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

4.26. 设置桶加密配置

4.26.1. 功能说明

通过设置桶的加密配置开启服务端加密功能，可以对重要数据进行加密保护。设置桶的加密配置后，所有上传至该桶的对象数据都会自动进行加密处理，目前支持 AES256 和国密 SM4 加密算法。

4.26.2. 代码示例

```
func PutBucketEncryption(svc *s3.S3) {
    putBucketEncryptionInput := &s3.PutBucketEncryptionInput{
        Bucket: aws.String("<your-bucket-name>"),
        ServerSideEncryptionConfiguration: &s3.ServerSideEncryptionConfiguration{
            Rules: []*s3.ServerSideEncryptionRule{
                {
                    ApplyServerSideEncryptionByDefault: &s3.ServerSideEncryptionByDefault{
                        SSEAlgorithm: aws.String("AES256"), //AES256 算法：AES256；国密 SM4 算法：SM4
                    },
                },
            },
        },
    }

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

通过 PutBucketEncryptionRequest 操作：

PutBucketEncryptionRequest 操作首先生成一个"request.Request"对象，该对象是一个执行 PutBucketEncryption 操作的请求。通过调用 Request 对象的 Send 方法完成设置桶加密配置的操作。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 PutBucketEncryptionRequest(svc *s3.S3) {
    putBucketEncryptionInput := &s3.PutBucketEncryptionInput{
        Bucket: aws.String("<your-bucket-name>"),
        ServerSideEncryptionConfiguration: &s3.ServerSideEncryptionCo
nfiguration{
            Rules: []*s3.ServerSideEncryptionRule{
                {
                    ApplyServerSideEncryptionByDefault: &s3.ServerSide
EncryptionByDefault{
                        SSEAlgorithm: aws.String("AES256"), //AES256 算
法：AES256；国密 SM4 算法：SM4
                    },
                },
            },
        },
    }

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

4.26.3. 请求参数

PutBucketEncryptionInput 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
ServerSideEncryptionConfiguration	*ServerSideEncryptionConfiguration	描述桶默认加密配置信息，包含设置的加密算法	是

4.27. 获取桶加密配置

4.27.1. 功能说明

获取指定桶的加密配置信息。

4.27.2. 代码示例

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

    GetBucketEncryptionOutput, err := svc.GetBucketEncryption(getBuck
etEncryptionInput)
    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)
    }
}
```

4.27.3. 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

4.27.4. 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

4.28. 删除桶加密配置

4.28.1. 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。

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

4.28.3. 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

5. 对象相关接口

5.1. 上传对象

5.1.1. 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。

5.1.2. 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)
    }
}
```

5.1.3. 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

参数	类型	说明	是否必要
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

5.1.4. 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。

5.2. 下载对象

5.2.1. 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。

5.2.2. 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
    }
}
```

5.2.3. 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

参数	类型	说明	是否必要
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	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	否

参数	类型	说明	是否必要
		该参数的时候，默认获取最新版本的对象数据。	

5.2.4. 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。

5.3. 下载对象-范围下载

5.3.1. 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。

5.3.2. 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
        }
    }
}
```

5.3.3. 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
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

参数	类型	说明	是否必要
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

5.3.4. 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。

参数	类型	说明
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。

5.4. 下载对象-限定条件下载

5.4.1. 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。

5.4.2. 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
        }
    }
}
```

5.4.3. 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
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，	否

参数	类型	说明	是否必要
		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。	
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

5.4.4. 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。

参数	类型	说明
VersionId	*string	对象的版本 ID。
StorageClass	*string	对象的存储类型。

5.5. 复制对象

5.5.1. 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。

5.5.2. 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)
    }
}
```

5.5.3. 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
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

参数	类型	说明	是否必要
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

5.5.4. 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。

5.6. 删除对象

5.6.1. 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。

5.6.2. 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(deleteObjectIn
put)

    err := req.Send()
```

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

5.6.3. 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

5.6.4. 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。

5.7. 批量删除对象

5.7.1. 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。

5.7.2. 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(deleteObject
sInput)
```

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

5.7.3. 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

5.7.4. 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、

5.8. 获取对象元数据

5.8.1. 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。

5.8.2. 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)
    }
}
```

5.8.3. 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

参数	类型	说明	是否必要
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

5.8.4. 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。

5.9. 设置对象 ACL

5.9.1. 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。

5.9.2. 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: &permis
sion}
    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: &permis
sion}
    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(putObjectAclIn
put)

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

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

5.9.3. 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
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

5.10. 获取对象 ACL

5.10.1. 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。

5.10.2. 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(getObjectAclIn
put)

    err := req.Send()
```

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

5.10.3. 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

5.10.4. 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。

5.11. 设置对象属性

5.11.1. 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。

5.11.2. 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)
    }
}
```

5.11.3. 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

参数	类型	说明	是否必要
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

5.11.4. 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。

5.12. 设置对象标签

5.12.1. 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。

5.12.2. 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(putObjectTagg
ingInput)
    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(putObj
ectTaggingInput)

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

5.12.3. 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

5.13. 获取对象标签

5.13.1. 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。

5.13.2. 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(getObj
ectTaggingInput)

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

5.13.3. 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

5.13.4. 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。

5.14. 删除对象标签

5.14.1. 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。

5.14.2. 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(deleteO
bjectTaggingInput)
    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)
    }
}
```

5.14.3. 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

5.15. 服务端加密

5.15.1. 功能说明

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

5.15.2. 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)
    }
}
```

5.15.3. 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

参数	类型	说明	是否必要
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	否

参数	类型	说明	是否必要
		象时候重定向到当前 bucket 下的其他对象或者外部的 URL。	

5.15.4. 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。

5.16. 生成预签名下载链接

5.16.1. 功能说明

指定访问密钥、请求类型、请求参数等信息生成一个在 Query 参数中带有鉴权信息的 URL，通过该 URL 可以访问对象存储资源。

5.16.2. 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)
    duration := 10 * time.Minute
    urlStr, err := req.Presign(duration)
```

```
    if err != nil {
        fmt.Println("err, ", err)
        return
    }
    fmt.Println("success, ", urlStr)
}
```

5.16.3. 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

6. 分片上传接口

6.1. 分段上传-初始化分段上传任务

6.1.1. 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。

6.1.2. 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(cre
ateMultipartUploadInput)
    if err != nil {
        fmt.Printf("fail to create multipart upload. %v", err)
    } else {
        fmt.Printf("upload id: %v\n", *createMultipartUploadOutput.Up
loadId)
    }
}
```

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)
    }
}
```

6.1.3. 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

参数	类型	说明	是否必要
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

6.1.4. 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

6.2. 分段上传-上传段

6.2.1. 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。

6.2.2. 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)
    }
}
```

6.2.3. 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

6.2.4. 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

6.3. 分段上传-合并段

6.3.1. 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。

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

6.3.3. 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

参数	类型	说明	是否必要
UploadId	*string	通过 CreateMultipartUpload 操作获取的 UploadId，与一个对象的分段上传对应	是

6.3.4. 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

6.4. 分段上传-列举分段上传任务

6.4.1. 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。

6.4.2. 代码示例

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

```
    }
    listMultipartUploadsOutput, err := svc.ListMultipartUploads(listM
ultipartUploadsInput)
    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.ListMultipartUploadsReques
t(listMultipartUploadsInput)

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

6.4.3. 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
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

6.4.4. 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
UploadIdMarker	*string	返回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

6.5. 分段上传-列举已上传的段

6.5.1. 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。

6.5.2. 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)
    }
}
```

6.5.3. 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

6.5.4. 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

参数	类型	说明
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
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

6.6. 分段上传-复制段

6.6.1. 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。

6.6.2. 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(&s
3.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, uploadI
d); 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: partItem
s},
    })
    if err != nil {
        fmt.Printf("Failed to CompleteMultipartUpload. %v", err)
        fmt.Println(completeMultipartUploadOutput)
    }
}

func copyAUploadPart(svc *s3.S3, start int64, end int64, partNumber i
nt64, 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(uploadPartCopyInp
ut)
    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(&s
3.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(uploadPart
CopyInput)

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

6.6.3. 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	是

参数	类型	说明	是否必要
		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	
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

6.6.4. 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

6.7. 分段上传-取消分段上传任务

6.7.1. 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。

6.7.2. 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(abort
MultipartUploadInput)
    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.AbortMultipartUploadReques
t(abortMultipartUploadInput)
    err := req.Send()
    if err != nil {
        fmt.Printf("fail to AbortMultipartUpload. %v\n", err)
    } else {
        fmt.Println(abortMultipartUploadOutput)
    }
}
```

6.7.3. 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

7. 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。

7.1. 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)
```

7.2. 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","Ac
tion":["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

7.3. 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)
```