对象存储 zos-sdk-go 使用手册

环境依赖

Golang 版本　1.14 及以上，可访问 Golang 官网下载页面进行下载。

需配置 `go env` 中的 `GO111MODULE` 选项为 `on`。

项目配置

项目初始化

下面过程以"获取桶列表"功能为例，说明如何从零开始创建并运行一个 zos-sdk-go 项目。

首先在项目目录依次运行下列 go 命令初始化项目：

```
# 生成 go.mod 文件
go mod init zos-sdk-go-example

# 配置本地依赖
go mod edit -require="github.com/aws/aws-sdk-go@v1.38.63"
go mod edit -require="github.com/jmespath/go-jmespath@v0.4.0"
go mod edit -replace="github.com/aws/aws-sdk-go@v1.38.63"="{path of sdk}/aws-sdk-go-1.38.63"
go mod edit -replace="github.com/jmespath/go-jmespath@v0.4.0"="{path of sdk}/go-jmespath-0.4.0"
```

配置后的 `go.mod` 文件内容类似如下示例：

```
module zos-sdk-go-example

go 1.14

require (
    github.com/aws/aws-sdk-go v1.38.63
    github.com/jmespath/go-jmespath v0.4.0
)

replace github.com/aws/aws-sdk-go v1.38.63 => {path of sdk}/aws-sdk-go-1.38.63

replace github.com/jmespath/go-jmespath v0.4.0 => {path of sdk}/go-jmespath-0.4.0
```

获取访问凭证

`AccessKey`，`SecretAccessKey` 和 `Endpoint` 是调用ZOS服务必要的三个参数，具体获取方式如下：

- `AccessKey` 和 `SecretAccessKey`：可在控制台查看或通过 `OpenAPI` 的 get-keys 接口查询
- `Endpoint`：可在控制台查看或通过 `OpenAPI` 的 get-endpoint 接口查询

项目功能验证

在项目目录下创建 `main.go` 文件，内容如下：

```
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/request"
    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/s3"
    "os"
)

var AccessKey string
var SecretAccessKey string
var Endpoint string
var bucket string
var key string
var SessionToken string
var s3c *s3.S3

func init() {
    // 初始化账号信息 & 用于测试的桶名、对象名等
    AccessKey = os.Getenv("zosAk")
    SecretAccessKey = os.Getenv("zosSk")
    Endpoint = os.Getenv("zosEp")
    SessionToken = ``
    // 构建客户端
    s3c = BuildClient()
}

func main() {
    ListBuckets()
}

// 创建并返回一个对象存储 service client
func BuildClient() *s3.S3 {
    conf := &aws.Config{
        Endpoint:         aws.String(Endpoint),
        S3ForcePathStyle: aws.Bool(false),
        Credentials:      credentials.NewStaticCredentials(AccessKey,
SecretAccessKey, SessionToken),
        Region:           aws.String("us-east-1"),
    }
    sess := session.Must(session.NewSessionWithOptions(session.Options{Config:
*conf}))
    svc := s3.New(sess)
    return svc
}

// 发送 request 请求，输出响应
func send(req *request.Request) {
    err := req.Send()
    fmt.Printf(`@ ======================== `)
```

```
    fmt.Printf("%v", req.Operation.Name)
    fmt.Println(` ======================== @`)
    fmt.Printf("* Request URL: %v\n", req.HTTPRequest.URL)
    fmt.Printf("* Request Headers: \n{\n")
    for k, v := range req.HTTPRequest.Header {
        fmt.Printf("   %v: %v\n", k, v)
    }
    fmt.Println("}")
    fmt.Printf("* Response Headers: \n{\n")
    for k, v := range req.HTTPResponse.Header {
        fmt.Printf("   %v: %v\n", k, v)
    }
    fmt.Println("}")
    fmt.Printf("* Response Data:\n%v\n", req.Data)
    if err != nil {
        fmt.Printf("* Request Error:\n%v\n", err)
    }
}

func ListBuckets() {
    req, _ := s3c.ListBucketsRequest(&s3.ListBucketsInput{})
    send(req)
}
```

完成上述操作之后，可执行 `go run main.go` 查看运行结果，响应结果示例如下：

```
@ ======================== ListBuckets ======================== @
* Request URL: https://jiangsu-10.zos.ctyun.cn/
* Request Headers:
{
   X-Amz-Date: [20010801T083006Z]
   X-Amz-Content-Sha256:
[k9bxxxxxxxxxc149afbf4c890000000000001e4649xxxxxxxxxxxxxxxxxxxx]
   Authorization: [AWS4-HMAC-SHA256 Credential=Q2KXXXXXXXXY82W19AU/20010801/us-
east-1/s3/aws4_request, SignedHeaders=host;x-amz-content-sha256;x-amz-date,
Signature=xxxxxxxxxxxxxx98ea7924efekf2997dj64l60k6f15e6aa0000000000000000]
   User-Agent: [aws-sdk-go/1.38.63 (go1.14; windows; amd64)]
}
* Response Headers:
{
   Server: [ct-zos/1.22.1]
   Date: [Sat, 01 Aug 2001 08:30:07 GMT]
   Content-Type: [application/xml]
   Connection: [keep-alive]
   Vary: [Accept-Encoding]
   X-Amz-Request-Id: [l2000000000000003ekk290-2226akea8f-324c0222-js10]
}
* Response Data:
{
  Buckets: [
    {
      CreationDate: 2000-11-09 06:54:20.7 +0000 UTC,
      Name: "test"
    }
  ],
  Owner: {
    DisplayName: "张三",
```

```
    ID: "xxxxxxxx-0000-xxxx-0000-xxxxxxxxxxxx"
  }
}
```

桶操作

创建桶

功能介绍

在指定账号下创建一个新的桶。

方法原型

```
func (c *S3) CreateBucketRequest(input *CreateBucketInput) (req *request.Request, output *CreateBucketOutput)
```

输入参数

属性名	类型	说明	是否必要
ACL	*string	配置创建桶预定义的标准ACL信息，例如 `private`、`public-read` 等	否
Bucket	*string	创建桶的名称	是
AZPolicy	*string	桶的数据冗余策略，可选值为 `single-az` 和 `multi-az`，分别表示单 AZ 存储和多 AZ 存储。默认为单 AZ 存储。	否
StorageClass	*string	桶的存储类型，可选值为 `STANDARD`、`STANDARD_IA` 和 `GLACIER`，分别表示标准、低频、归档。默认为标准存储。	否

输出结果

无

代码示例

```
func CreateBucket() {
    input := &s3.CreateBucketInput{
        Bucket: aws.String(bucket),
    }
    req, _ := s3c.CreateBucketRequest(input)
    send(req)
}
```

获取桶列表

功能介绍

查询请求用户拥有的所有桶的列表。

方法原型

```
func (c *S3) ListBucketsRequest(input *ListBucketsInput) (req *request.Request, output *ListBucketsOutput)
```

输入参数

无

输出结果

属性名	类型	说明
Buckets	[]*Bucket	桶信息数组，包含了每个桶的名字与创建时间
Owner	*Owner	桶的拥有者信息

代码示例

```
func ListBuckets() {
    req, _ := s3c.ListBucketsRequest(&s3.ListBucketsInput{})
    send(req)
}
```

获取桶信息

功能介绍

查询用户账号下某个桶是否存在并且用户是否有权访问。

方法原型

```
func (c *S3) HeadBucketRequest(input *HeadBucketInput) (req *request.Request, output *HeadBucketOutput)
```

输入参数

属性名	类型	说明	是否必要
Bucket	*string	桶的名称	是

代码示例

```
func HeadBucket() {
    input := &s3.HeadBucketInput{
        Bucket: aws.String(bucket),
    }
    req, _ := s3c.HeadBucketRequest(input)
    send(req)
}
```

删除桶

功能介绍

删除指定桶。删除一个桶前，需要先删除该桶中的全部对象。

方法原型

```
func (c *S3) DeleteBucketRequest(input *DeleteBucketInput) (req *request.Request, output *DeleteBucketOutput)
```

输入参数

属性名	类型	说明	是否必要
Bucket	*string	桶的名称	是

输出结果

无

代码示例

```
func DeleteBucket() {
    input := &s3.DeleteBucketInput{
        Bucket: aws.String(bucket),
    }
    req, _ := s3c.DeleteBucketRequest(input)
    send(req)
}
```

获取桶访问权限

功能介绍

获取桶的 ACL，用户在获取桶的 ACL 之前需要具备 `READ_ACP` 权限。

方法原型

```
func (c *S3) GetBucketAclRequest(input *GetBucketAclInput) (req *request.Request, output *GetBucketAclOutput)
```

输入参数

属性名	类型	说明	是否必要
Bucket	*string	桶的名称	是

输出结果

属性名	类型	说明
Grants	[]*Grant	授权信息，包含了每项授权和被授权人的信息
Owner	*Owner	桶的拥有者信息

代码示例

```
func GetBucketAcl() {
    input := &s3.GetBucketAclInput{
        Bucket: aws.String(bucket),
    }
    req, _ := s3c.GetBucketAclRequest(input)
    send(req)
}
```

设置桶访问权限

功能介绍

设置桶的 ACL，用户在设置桶的 ACL 之前需要具备 `WRITE_ACP` 权限。

方法原型

```
func (c *S3) PutBucketAclRequest(input *PutBucketAclInput) (req *request.Request, output *PutBucketAclOutput)
```

输入参数

属性名	类型	说明	是否必要
ACL	*string	配置创建桶预定义的标准 ACL 信息，可选值为 `private` 和 `public-read`	否
AccessControlPolicy	*AccessControlPolicy	配置该桶对于每个用户的 ACL 授权信息	否
Bucket	*string	桶的名称	是

`AccessControlPolicy` 属性表

属性名	类型	说明	是否必要
Grants	[]*Grant	授权信息	是
Owner	*Owner	所有者	是

`Grants` 属性表

属性名	类型	说明	是否必要
Grantee	*Grantee	被授权者	是
Permission	*string	授予权限，可选值为 `READ`，`READ_ACP`、 `WRITE`、`WRITE_ACP` 和 `FULL_CONTROL`	是

`Owner` 属性表

属性名	类型	说明	是否必要
DisplayName	*string	用户名称	是
ID	*string	用户 ID	是

`Grantee` 属性表

属性名	类型	说明	是否必要
DisplayName	*string	用户名称	否
EmailAddress	*string	用户邮箱	否
Type	*string	用户类型，可选值为 `CanonicalUser`、`AmazonCustomerByEmail` 和 `Group`。 `CanonicalUser` 则 `ID` 为必填项； `AmazonCustomerByEmail` 则 `EmailAddress` 为必填项，并且将会保存其指向到的 `CanonicalUser` 类型的用户； `Group` 则 `URI` 为必填项。	是
ID	*string	标准用户 ID	否
URI	*string	授权组的 URI	否

输出结果

无

代码示例

```
func PutBucketAcl() {
    input := &s3.PutBucketAclInput{
        Bucket: aws.String(bucket),
        ACL:    aws.String("private"),
    }
    req, _ := s3c.PutBucketAclRequest(input)
    send(req)
}
```

获取桶策略

功能介绍

获取桶的桶策略。

方法原型

```
func (c *S3) GetBucketPolicyRequest(input *GetBucketPolicyInput) (req
*request.Request, output *GetBucketPolicyOutput)
```

输入参数

属性名	类型	说明	是否必要
Bucket	*string	桶的名称	是

输出结果

属性名	类型	说明
Policy	*string	JSON 格式的桶策略信息

代码示例

```
func GetBucketPolicy() {
    input := &s3.GetBucketPolicyInput{
        Bucket: aws.String(bucket),
    }
    req, _ := s3c.GetBucketPolicyRequest(input)
    send(req)
}
```

设置桶策略

功能介绍

设置桶的桶策略。描述桶策略的信息以 JSON 格式的字符串通过 `Policy` 参数传入。

方法原型

```
func (c *S3) PutBucketPolicyRequest(input *PutBucketPolicyInput) (req
*request.Request, output *PutBucketPolicyOutput)
```

输入参数

属性名	类型	说明	是否必要
Bucket	*string	桶的名称	是
Policy	*string	JSON 格式的桶策略信息字符串，详细结构见下方补充表格	是

`Policy` 的内部结构如下：

属性名	类型	说明
Version	string	当前支持 "2012-10-17"
Id	string	桶策略 Id
Statement	list	桶策略描述列表，定义完整的权限控制。每条桶策略的 `Statement` 可由多条描述组成，每条描述均为一个 dict

`Statement` 的内部结构如下：

属性名	类型	说明
Sid	string	本条桶策略描述的ID
Effect	string	桶策略的效果，即指定本条桶策略描述的权限是接受请求还是拒绝请求。接受请求：`"Allow"`， 拒绝请求：`"Deny"`
Principal	dict	指定本条桶策略描述所作用的用户。以字典形式表示，支持直接使用通配符 `"*"` 表示所有用户。当对特定用户进行授权时，格式为 `{"AWS": "arn:aws:s3:::userId"}`
Action	list	指定本条桶策略描述所作用的操作。以列表形式表示，支持直接使用通配符 `"*"` 表示该资源能进行的所有操作。常用的Action有 `"s3:GetObject"`，`"s3:PutObject"` 等
Resource	list	此条策略所作用的资源，如桶、对象等
Conditon	dict	条件语句，指定本条桶策略所限制的条件。可以用于对资源配置各种策略，比如设置防盗链等

输出结果

无

代码示例

```
func PutBucketPolicy() {
    // 一个允许任意用户读取指定桶内对象的桶策略
    policy := `
    {
        "Version": "2012-10-17",
        "Id": "policy id",
        "Statement": [
            {
                "Sid": "statement id",
                "Effect": "Allow",
                "Principal": "*",
                "Action": "*",
                "Resource": [
                    "arn:aws:s3:::YOUR-BUCKET-NAME/*"
                ]
            }
        ]
    }`
    input := &s3.PutBucketPolicyInput{
        Bucket: aws.String(bucket),
        Policy: aws.String(policy),
    }
    req, _ := s3c.PutBucketPolicyRequest(input)
    send(req)
}
```

删除桶策略

功能介绍

删除桶的桶策略。

方法原型

```
func (c *S3) DeleteBucketPolicyRequest(input *DeleteBucketPolicyInput) (req
*request.Request, output *DeleteBucketPolicyOutput)
```

输入参数

属性名	类型	说明	是否必要
Bucket	*string	桶的名称	是

输出结果

无

代码示例

```
func DeleteBucketPolicy() {
    input := &s3.DeleteBucketPolicyInput{
        Bucket: aws.String(bucket),
    }
    req, _ := s3c.DeleteBucketPolicyRequest(input)
    send(req)
}
```

获取版本控制信息

功能介绍

获取桶的版本控制状态信息，只有桶的拥有者才能获取到桶的版本控制信息。

桶的版本控制有三种状态：未开启/开启（Enabled）/暂停（Suspended）。桶在被设置版本状态前默认为未开启状态，此时调用本接口将无法获得任何信息。

方法原型

```
func (c *S3) GetBucketVersioningRequest(input *GetBucketVersioningInput) (req
*request.Request, output *GetBucketVersioningOutput)
```

输入参数

属性名	类型	说明	是否必要
Bucket	*string	桶的名称	是

输出结果

属性名	类型	说明
Status	*string	桶的版本控制设置状态，可选值为 `Enabled` 和 `Suspended`

代码示例

```
func GetBucketVersioning() {
    input := &s3.GetBucketVersioningInput{
        Bucket: aws.String(bucket),
    }
    req, _ := s3c.GetBucketVersioningRequest(input)
    send(req)
}
```

设置版本控制

功能介绍

设置桶的版本控制状态。

方法原型

```
func (c *S3) PutBucketVersioningRequest(input *PutBucketVersioningInput) (req
*request.Request, output *PutBucketVersioningOutput)
```

输入参数

属性名	类型	说明	是否必要
Bucket	*string	桶的名称	是
VersioningConfiguration	*VersioningConfiguration	设置版本控制状态的参数	是

`VersioningConfiguration` 属性表

属性名	类型	说明	是否必要
Status	*string	桶的版本控制设置状态，可选值为 `Enabled` 和 `Suspended`	是

输出结果

无

代码示例

```
func PutBucketVersioning() {
    input := &s3.PutBucketVersioningInput{
        Bucket: aws.String(bucket),
        VersioningConfiguration: &s3.VersioningConfiguration{
            Status: aws.String("Enabled"),
        },
    }
    req, _ := s3c.PutBucketVersioningRequest(input)
    send(req)
}
```

获取桶生命周期规则

功能介绍

获取指定桶的生命周期规则。

方法原型

```
func (c *S3) GetBucketLifecycleConfigurationRequest(input
*GetBucketLifecycleConfigurationInput) (req *request.Request, output
*GetBucketLifecycleConfigurationOutput)
```

输入参数

属性名	类型	说明	是否必要
Bucket	*string	桶的名称	是

输出结果

属性名	类型	说明
Rules	[]*LifecycleRule	封装生命周期的数组

`LifecycleRule` 属性表

属性名	类型	说明
AbortIncompleteMultipartUpload	*AbortIncompleteMultipartUpload	指定未完成的分段上传的保留天数
Expiration	*LifecycleExpiration	用日期或天数指定的对象过期时间
Filter	*LifecycleRuleFilter	过滤规则，可为空。若非空则必须为 `Prefix`、`Tag` 或 `And` 之一。
ID	*string	生命周期规则的唯一标识
NoncurrentVersionExpiration	*NoncurrentVersionExpiration	指定历史版本的过期规则
NoncurrentVersionTransitions	[]*NoncurrentVersionTransition	指定历史版本的转存储规则
Status	*string	标识本生命周期规则是否启用
Transitions	[]*Transition	指定桶内对象何时转存储到指定的存储类型

代码示例

```
func GetBucketLifecycleConfiguration() {
    input := &s3.GetBucketLifecycleConfigurationInput{
        Bucket: aws.String(bucket),
    }
    req, _ := s3c.GetBucketLifecycleConfigurationRequest(input)
    send(req)
}
```

设置桶生命周期规则

功能介绍

为指定桶设置生命周期规则。

方法原型

```
func (c *S3) PutBucketLifecycleConfigurationRequest(input
*PutBucketLifecycleConfigurationInput) (req *request.Request, output
*PutBucketLifecycleConfigurationOutput)
```

输入参数

属性名	类型	说明	是否必要
Bucket	*string	桶的名称	是
LifecycleConfiguration	*BucketLifecycleConfiguration	封装生命周期的数组，最多可包含1000条规则	否

`BucketLifecycleConfiguration` 属性表

属性名	类型	说明	是否必要
Rules	[]*LifecycleRule	封装生命周期的数组	是

`LifecycleRule` 属性表

属性名	类型	说明	是否必要
AbortIncompleteMultipartUpload	*AbortIncompleteMultipartUpload	指定未完成的分段上传的保留天数	否
Expiration	*LifecycleExpiration	用日期或天数指定的对象过期时间	否
Filter	*LifecycleRuleFilter	过滤规则，可为空。若非空则必须为 `Prefix`、`Tag` 或 `And` 之一。	否
ID	*string	生命周期规则的唯一标识	否
NoncurrentVersionExpiration	*NoncurrentVersionExpiration	指定历史版本的过期规则	否
NoncurrentVersionTransitions	[]*NoncurrentVersionTransition	指定历史版本的转存储规则	否
Status	*string	标识本生命周期规则是否启用	是
Transitions	[]*Transition	指定桶内对象何时转存储到指定的存储类型	否

输出结果

无

代码示例

```
func PutBucketLifecycleConfiguration() {
    // 设置匹配指定前缀的对象一天后过期
    rule := &s3.LifecycleRule{
        ID:     aws.String("expireAfterOneDay"),
        Status: aws.String("Enabled"),
        Filter: &s3.LifecycleRuleFilter{
            Prefix: aws.String("expireAfterOneDay/"),
        },
        Expiration: &s3.LifecycleExpiration{
            Days: aws.Int64(1),
        },
    }
    input := &s3.PutBucketLifecycleConfigurationInput{
        Bucket: aws.String(bucket),
        LifecycleConfiguration: &s3.BucketLifecycleConfiguration{
            Rules: []*s3.LifecycleRule{rule},
        },
    }
    req, _ := s3c.PutBucketLifecycleConfigurationRequest(input)
    send(req)
}
```

删除生命周期规则

功能介绍

删除指定桶的生命周期规则。

方法原型

```
func (c *S3) DeleteBucketLifecycleRequest(input *DeleteBucketLifecycleInput) (req
*request.Request, output *DeleteBucketLifecycleOutput)
```

输入参数

属性名	类型	说明	是否必要
Bucket	*string	桶的名称	是

输出结果

无

代码示例

```
func DeleteBucketLifecycle() {
    input := &s3.DeleteBucketLifecycleInput{
        Bucket: aws.String(bucket),
    }
    req, _ := s3c.DeleteBucketLifecycleRequest(input)
    send(req)
}
```

获取桶标签

功能介绍

获取指定桶的标签信息。

方法原型

```
func (c *S3) GetBucketTaggingRequest(input *GetBucketTaggingInput) (req
*request.Request, output *GetBucketTaggingOutput)
```

输入参数

属性名	类型	说明	是否必要
Bucket	*string	桶的名称	是

输出结果

属性名	类型	说明
TagSet	[]*Tag	标签集合

`Tag` 属性表

属性名	类型	说明
Key	*string	键
Value	*string	值

代码示例

```
func GetBucketTagging() {
    input := &s3.GetBucketTaggingInput{
        Bucket: aws.String(bucket),
    }
    req, _ := s3c.GetBucketTaggingRequest(input)
    send(req)
}
```

设置桶标签

功能介绍

设置指定桶的标签信息。

方法原型

```
func (c *S3) PutBucketTaggingRequest(input *PutBucketTaggingInput) (req
*request.Request, output *PutBucketTaggingOutput)
```

输入参数

属性名	类型	说明	是否必要
Bucket	*string	桶的名称	是
Tagging	*Tagging	封装标签键值对的数组	是

`Tagging` 属性表

属性名	类型	说明	是否必要
TagSet	[]*Tag	标签集合	是

`Tag` 属性表

属性名	类型	说明	是否必要
Key	*string	键	是
Value	*string	值	是

输出结果

无

代码示例

```
func PutBucketTagging() {
    tagging := &s3.Tagging{
        TagSet: []*s3.Tag{
            {
                Key:   aws.String("key1"),
                Value: aws.String("value1"),
            },
        },
    }
    input := &s3.PutBucketTaggingInput{
        Bucket:  aws.String(bucket),
        Tagging: tagging,
    }
    req, _ := s3c.PutBucketTaggingRequest(input)
    send(req)
}
```

删除桶标签

功能介绍

删除指定桶的标签信息。

方法原型

```
func (c *S3) DeleteBucketTaggingRequest(input *DeleteBucketTaggingInput) (req
*request.Request, output *DeleteBucketTaggingOutput)
```

输入参数

属性名	类型	说明	是否必要
Bucket	*string	桶的名称	是

输出结果

无

代码示例

```
func DeleteBucketTagging() {
    input := &s3.DeleteBucketTaggingInput{
        Bucket: aws.String(bucket),
    }
    req, _ := s3c.DeleteBucketTaggingRequest(input)
    send(req)
}
```

获取桶加密

功能介绍

获取指定桶的加密信息。

方法原型

```
func (c *S3) GetBucketEncryptionRequest(input *GetBucketEncryptionInput) (req
*request.Request, output *GetBucketEncryptionOutput)
```

输入参数

属性名	类型	说明	是否必要
Bucket	*string	桶的名称	是

输出结果

属性名	类型	说明
ServerSideEncryptionConfiguration	*ServerSideEncryptionConfiguration	服务端加密配置信息

`ServerSideEncryptionConfiguration` 属性表

属性名	类型	说明
Rules	[]*ServerSideEncryptionRule	服务端加密配置规则的容器

`ServerSideEncryptionRule` 属性表

属性名	类型	说明
ApplyServerSideEncryptionByDefault	*ServerSideEncryptionByDefault	默认的服务端加密配置，未指定加密方式的对象上传操作将默认使用此配置

`ServerSideEncryptionByDefault` 属性表

属性名	类型	说明
KMSMasterKeyID	*string	使用 `aws:kms` 加密算法时的 KMS key ID
SSEAlgorithm	*string	服务端加密算法

代码示例

```
func GetBucketEncryption() {
    input := &s3.GetBucketEncryptionInput{
        Bucket: aws.String(bucket),
    }
    req, _ := s3c.GetBucketEncryptionRequest(input)
    send(req)
}
```

设置桶加密

功能介绍

设置指定桶的加密方式。

方法原型

```
func (c *S3) PutBucketEncryptionRequest(input *PutBucketEncryptionInput) (req
*request.Request, output *PutBucketEncryptionOutput)
```

输入参数

属性名	类型	说明	是否必要
Bucket	*string	桶的名称	是
ServerSideEncryptionConfiguration	*ServerSideEncryptionConfiguration	服务端加密配置信息	是

`ServerSideEncryptionConfiguration` 属性表

属性名	类型	说明	是否必要
Rules	[]*ServerSideEncryptionRule	服务端加密配置规则的容器	是

`ServerSideEncryptionRule` 属性表

属性名	类型	说明	是否必要
ApplyServerSideEncryptionByDefault	*ServerSideEncryptionByDefault	默认的服务端加密配置，未指定加密方式的对象上传操作将默认使用此配置	是

`ServerSideEncryptionByDefault` 属性表

属性名	类型	说明	是否必要
KMSMasterKeyID	*string	使用 `aws:kms` 加密算法时的 KMS key ID，参数格式为 `{密钥管理服务处的密钥ID}:::{regionID}:{userID}`	否
SSEAlgorithm	*string	服务端加密算法，仅支持 `AES256` 或 `aws:kms`。若传入 `AES256`，将自动生成 `KMSMasterKeyID`，若传入 `aws:kms`，需用户预先通过密钥管理服务创建密钥	是

输出结果

无

代码示例

```
func PutBucketEncryption() {
    serverSideEncryptionConfiguration := &s3.ServerSideEncryptionConfiguration{
        Rules: []*s3.ServerSideEncryptionRule{
            {
                ApplyServerSideEncryptionByDefault:
&s3.ServerSideEncryptionByDefault{
                    KMSMasterKeyID: aws.String(`xxxxxxxx-xxxx-xxxx-xxxx-
xxxxxxxxxxxx:::xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx:xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx`
),
                    SSEAlgorithm:   aws.String(`aws:kms`),
                },
            },
        },
    }
    input := &s3.PutBucketEncryptionInput{
        Bucket:                            aws.String(bucket),
        ServerSideEncryptionConfiguration: serverSideEncryptionConfiguration,
    }
    req, _ := s3c.PutBucketEncryptionRequest(input)
```

```
    send(req)
}
```

删除桶加密

功能介绍

删除指定桶的加密配置。

方法原型

```
func (c *S3) DeleteBucketEncryptionRequest(input *DeleteBucketEncryptionInput)
(req *request.Request, output *DeleteBucketEncryptionOutput)
```

输入参数

属性名	类型	说明	是否必要
Bucket	*string	桶的名称	是

输出结果

无

代码示例

```
func DeleteBucketEncryption() {
    input := &s3.DeleteBucketEncryptionInput{
        Bucket: aws.String(bucket),
    }
    req, _ := s3c.DeleteBucketEncryptionRequest(input)
    send(req)
}
```

对象操作

获取对象

功能介绍

下载对象。

方法原型

```
func (c *S3) GetObjectRequest(input *GetObjectInput) (req *request.Request,
output *GetObjectOutput)
```

输入参数

属性名	类型	说明	是否必要
Bucket	*string	桶的名称	是
Key	*string	对象的 `Key`	是
VersionId	*string	用于获取对象的特定版本	否

输出结果

属性名	类型	说明
AcceptRanges	*string	指出一段字节的范围
Body	io.ReadClosed	对象数据
ContentLength	*int64	以字节为单位的对象数据长度
ContentType	*string	描述对象数据格式的 MIME 类型
ETag	*string	对象的 `ETag`
LastModified	*time.Time	最后修改对象的时间
TagCount	*int64	对象上标签的数量(如果有)
VersionId	*string	对象版本

代码示例

```
func GetObject() {
    input := &s3.GetObjectInput{
        Bucket: aws.String(bucket),
        Key:    aws.String(key),
    }
    req, _ := s3c.GetObjectRequest(input)
    send(req)
}
```

上传对象

功能介绍

上传对象。单次上传最大不能超过5GB，超过5GB的文件需要使用分段上传操作，请参考“分段上传操作”章节。

方法原型

```
func (c *S3) PutObjectRequest(input *PutObjectInput) (req *request.Request,
output *PutObjectOutput)
```

输入参数

属性名	类型	说明	是否必要
ACL	*string	配置上传对象的预定义 ACL 信息，如`private`、`public-read`、`public-read-write`等。不传该参数时默认为`private`	否
Body	io.ReadSeeker	对象的数据	是
Bucket	*string	执行本操作的桶名称	是
Key	*string	对象的名称，如需上传到`/folder`目录下，只需设置`Key`为`/folder/{Key}`即可	是
Tagging	*string	对象的标签信息，必须是 URL 请求参数的形式。如`Key1=Value1`	否

输出结果

属性名	类型	说明
ETag	*string	上传对象后对应的`Entity Tag`
VersionId	*string	上传对象后对应的版本号

代码示例

```
func PutObject() {
    // 以当前文件夹的指定文件作为输入
    file, err := os.Open("./go.mod")
    if err != nil {
        panic(err)
    }
    defer file.Close()
    input := &s3.PutObjectInput{
        Bucket: aws.String(bucket),
        Key:    aws.String(key),
        Body:   file,
    }
    req, _ := s3c.PutObjectRequest(input)
    send(req)
}
```

获取对象列表

功能介绍

列出指定桶中的全部对象，该操作最多返回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。

方法原型

```
func (c *S3) ListObjectsRequest(input *ListObjectsInput) (req *request.Request, output *ListObjectsOutput)
```

输入参数

属性名	类型	说明	是否必要
Bucket	*string	桶的名称	是
Delimiter	*string	一个分隔符，用于分组键	否
Maker	*string	指定开始列出对象的分隔 `key`（按字典序）	否
MaxKeys	*int64	最大返回数量，默认和最大值均为1000	否
Prefix	*string	返回以指定前缀开头的 `key`	否

输出结果

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 `Prefix` 且第一次出现 `Delimiter` 字符的对象 `key` 作为一组
Contents	[]*Object	对象的元数据
Delimiter	*string	与请求中设置的 `Delimiter` 相同
IsTruncated	*bool	表示响应结果是否被截断
Maker	*string	与请求中设置的 `Maker` 相同
MaxKeys	*int64	返回结果的对象数量的最大值
Name	*string	执行本操作的桶名称
NextMaker	*string	当响应被截断时（`IsTruncated` 为 true），可以在下次查询请求时用该字段作为 `Marker` 来获取下一组对象
Prefix	*string	与请求中设置的 `Prefix` 一致

代码示例

```
func ListObjects() {
    input := &s3.ListObjectsInput{
        Bucket: aws.String(bucket),
    }
    req, _ := s3c.ListObjectsRequest(input)
    send(req)
}
```

删除对象

功能介绍

删除指定对象。对于开启了版本控制的桶执行该操作时，若指定版本号，则删除该版本的对象; 若未指定版本号，则将保留对象的当前版本并插入删除标记（Delete Marker），`DeleteObjectOutput` 中 `DeleteMarker` 将为 true，后续若执行 `GetObject` 操作将检测到该标记并返回 `404` 错误。

方法原型

```
func (c *S3) DeleteObjectRequest(input *DeleteObjectInput) (req *request.Request, output *DeleteObjectOutput)
```

输入参数

属性名	类型	说明	是否必要
Bucket	*string	桶的名称	是
Key	*string	对象的 `key`	是
VersionId	*string	指定要删除的对象的版本号	否

输出结果

属性名	类型	说明
DeleteMarker	*bool	桶开启版本控制下，未指定版本号将得到该值为 true 的返回值
VersionId	*string	删除标记所在的版本号

代码示例

```
func DeleteObject() {
    input := &s3.DeleteObjectInput{
        Bucket: aws.String(bucket),
        Key:    aws.String(key),
    }
    req, _ := s3c.DeleteObjectRequest(input)
    send(req)
}
```

批量删除对象

功能介绍

批量删除多个对象，可以减少多次请求的重复花销。一次 `DeleteObjects` 操作最多可包含1000个 key 的删除请求。

方法原型

```
func (c *S3) DeleteObjectsRequest(input *DeleteObjectsInput) (req
*request.Request, output *DeleteObjectsOutput)
```

输入参数

属性名	类型	说明	是否必要
Bucket	*string	桶的名称	是
Delete	*Delete	封装了要删除的的对象和返回模式的属性	是

输出结果

属性名	类型	说明
Deleted	[]*DeletedObject	被删除的对象的信息数组，每项都包含了一个被成功删除的对象的删除标记、`key` 和版本号等信息
Errors	[]*Error	删除失败的对象的信息数组，每项都包含了一个被成功删除的对象的信息和错误码

代码示例

```
func DeleteObjects() {
    input := &s3.DeleteObjectsInput{
        Bucket: aws.String(bucket),
        Delete: &s3.Delete{
            Objects: []*s3.ObjectIdentifier{
                {
                    Key: aws.String("exampleKey1"),
                },
                {
                    Key: aws.String("exampleKey2"),
                },
            },
        },
    }
    req, _ := s3c.DeleteObjectsRequest(input)
    send(req)
}
```

复制对象

功能介绍

将指定对象拷贝到指定位置。

方法原型

```
func (c *S3) CopyObjectRequest(input *CopyObjectInput) (req *request.Request, output *CopyObjectOutput)
```

输入参数

属性名	类型	说明	是否必要
ACL	*string	预定义的 ACL 信息	否
Bucket	*string	目标桶的名称	是
CopySource	*string	URL 格式的原对象路径，可以用 URL 请求参数的形式指定版本号，如 `{bucketName}/{objectName}?versionId={versionId}`	否
Key	*string	目标对象的 key	是

输出结果

属性名	类型	说明
CopyObjectResult	*CopyObjectResult	包含目标对象的 `ETag` 和最后修改时间等信息

代码示例

```
func CopyObject() {
    input := &s3.CopyObjectInput{
        Bucket:     aws.String(bucket),
        Key:        aws.String(key + `2`),
        CopySource: aws.String(bucket + `/` + key),
    }
    req, _ := s3c.CopyObjectRequest(input)
    send(req)
}
```

获取对象元数据

功能介绍

获取指定对象的元数据信息。

方法原型

```
func (c *S3) HeadObjectRequest(input *HeadObjectInput) (req *request.Request, output *HeadObjectOutput)
```

输入参数

属性名	类型	说明	是否必要
Bucket	*string	桶的名称	是
Key	*string	对象的 `key`	是
VersionId	*string	用于获取对象的特定版本	否

输出结果

属性名	类型	说明
ContentLength	*int64	以字节为单位的对象数据长度
ContentType	*string	描述对象数据格式的MIME类型
ETag	*string	对象的 `ETag`
LastModified	*time.Time	最后修改对象的时间
VersionId	*string	对象版本
Metadata	map[string]*string	元数据字典

代码示例

```
func HeadObject() {
    input := &s3.HeadObjectInput{
        Bucket: aws.String(bucket),
        Key:    aws.String(key),
    }
    req, _ := s3c.HeadObjectRequest(input)
    send(req)
}
```

获取对象标签

功能介绍

查询对象的标签信息。

方法原型

```
func (c *S3) GetObjectTaggingRequest(input *GetObjectTaggingInput) (req
*request.Request, output *GetObjectTaggingOutput)
```

输入参数

属性名	类型	说明	是否必要
Bucket	*string	桶的名称	是
Key	*string	对象的 `key`	是
VersionId	*string	对象的版本号	否

输出结果

属性名	类型	说明
TagSet	[]*Tag	标签集合

`Tag` 属性表

属性名	类型	说明
Key	*string	键
Value	*string	值

代码示例

```
func GetObjectTagging() {
    input := &s3.GetObjectTaggingInput{
        Bucket: aws.String(bucket),
        Key:    aws.String(key),
    }
    req, _ := s3c.GetObjectTaggingRequest(input)
    send(req)
}
```

设置对象标签

功能介绍

为对象设置标签。

方法原型

```
func (c *S3) PutObjectTaggingRequest(input *PutObjectTaggingInput) (req
*request.Request, output *PutObjectTaggingOutput)
```

输入参数

属性名	类型	说明	是否必要
Bucket	*string	桶的名称	是
Key	*string	对象的key	是
Tagging	*Tagging	封装标签键值对的数组	是
VersionId	*string	对象的版本号	否

`Tagging` 属性表

属性名	类型	说明	是否必要
TagSet	[]*Tag	标签集合	是

`Tag` 属性表

属性名	类型	说明	是否必要
Key	*string	键	是
Value	*string	值	是

输出结果

无

代码示例

```
func PutObjectTagging() {
    tagging := &s3.Tagging{
        TagSet: []*s3.Tag{
            {
                Key:   aws.String("key1"),
                Value: aws.String("value1"),
            },
        },
    }
    input := &s3.PutObjectTaggingInput{
        Bucket:  aws.String(bucket),
        Key:     aws.String(key),
        Tagging: tagging,
    }
    req, _ := s3c.PutObjectTaggingRequest(input)
    send(req)
}
```

删除对象标签

功能介绍

删除指定对象的全部标签信息。删除时可以指定版本号参数来删除指定版本的对象的标签信息，不指定时默认操作于当前版本。

方法原型

```
func (c *S3) DeleteObjectTaggingRequest(input *DeleteObjectTaggingInput) (req *request.Request, output *DeleteObjectTaggingOutput)
```

输入参数

属性名	类型	说明	是否必要
Bucket	*string	桶的名称	是
Key	*string	对象的 `key`	是
VersionId	*string	对象的版本号	否

输出结果

无

代码示例

```
func DeleteObjectTagging() {
    input := &s3.DeleteObjectTaggingInput{
        Bucket: aws.String(bucket),
        Key:    aws.String(key),
    }
    req, _ := s3c.DeleteObjectTaggingRequest(input)
    send(req)
}
```

获取对象访问权限配置信息

功能介绍

获取指定对象的访问权限配置信息。

方法原型

```
func (c *S3) GetObjectAclRequest(input *GetObjectAclInput) (req *request.Request, output *GetObjectAclOutput)
```

输入参数

属性名	类型	说明	是否必要
Bucket	*string	桶的名称	是
Key	*string	对象的 `key`	是
VersionId	*string	对象的版本号	否

输出结果

属性名	类型	说明
Grants	[]*Grant	授权信息的数组，包含了每项授权和被授权人的信息
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

代码示例

```
func GetObjectAcl() {
    input := &s3.GetObjectAclInput{
        Bucket: aws.String(bucket),
        Key:    aws.String(key),
    }
    req, _ := s3c.GetObjectAclRequest(input)
    send(req)
}
```

设置对象访问权限

功能介绍

为指定对象设置访问权限。

方法原型

```
func (c *S3) PutObjectAclRequest(input *PutObjectAclInput) (req *request.Request, output *PutObjectAclOutput)
```

输入参数

属性名	类型	说明	是否必要
ACL	*string	预定义的标准ACL信息，常用的有 `private` 私有读写、`public-read` 公共读、`public-read-write` 公共读写等	否
AccessControlPolicy	*AccessControlPolicy	配置该对象对于每个用户的 ACL 授权信息	否
Bucket	*string	桶的名称	是
Key	*string	对象的 `key`	是
VersionId	*string	对象的版本号	否

`AccessControlPolicy` 属性表

属性名	类型	说明	是否必要
Grants	[]*Grant	授权信息	是
Owner	*Owner	所有者	是

`Grants` 属性表

属性名	类型	说明	是否必要
Grantee	*Grantee	被授权者	是
Permission	*string	授予权限，可选值为 `READ`，`READ_ACP`、 `WRITE`、`WRITE_ACP` 和 `FULL_CONTROL`	是

`Owner` 属性表

属性名	类型	说明	是否必要
DisplayName	*string	所有者的用户名称	是
ID	*string	所有者的用户 ID	是

`Grantee` 属性表

属性名	类型	说明	是否必要
DisplayName	*string	用户名称	否
EmailAddress	*string	用户邮箱	否
Type	*string	用户类型，可选值为 `CanonicalUser`、`AmazonCustomerByEmail` 和 `Group`。`CanonicalUser` 则 `ID` 为必填项；`AmazonCustomerByEmail` 则 `EmailAddress` 为必填项，并且将会保存其指向到的 `CanonicalUser` 类型的用户；`Group` 则 `URI` 为必填项。	是
ID	*string	标准用户 ID	否
URI	*string	授权组的 URI	否

输出结果

无

代码示例

```
func PutObjectAcl() {
    input := &s3.PutObjectAclInput{
        Bucket: aws.String(bucket),
        Key:    aws.String(key),
        ACL:    aws.String("private"),
    }
    req, _ := s3c.PutObjectAclRequest(input)
    send(req)
}
```

获取对象版本信息

功能介绍

列出指定桶中的全部对象的版本信息，该操作最多返回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。

方法原型

```
func (c *S3) ListObjectVersionsRequest(input *ListObjectVersionsInput) (req
*request.Request, output *ListObjectVersionsOutput)
```

输入参数

属性名	类型	说明	是否必要
Bucket	*string	桶的名称	是
Delimiter	*string	若设置了 `Prefix`，所有包含指定 `Prefix` 且第一次出现 `Delimiter` 字符的对象 `key` 作为一组。若未指定 `Prefix` 参数，按 `Delimiter` 对所有对象 `key` 进行分割	否
KeyMarker	*string	指示从哪里开始列出，`ZOS` 会在这个指定的键之后开始列出，即列出名称大于 `keyMarker` 的对象。标记可以是桶中的任何键。默认值为空字符串 ""。	否
MaxKeys	*int64	响应中返回的对象 `key` 的数量，最大值和默认值均为1000	否
Prefix	*string	返回的 `key` 的前缀，注意 `key` 与 `Prefix` 完全相同的对象不会返回。默认值为空字符串 ""。	否

输出结果

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 `Prefix` 且第一次出现 `Delimiter` 字符的对象 `key` 作为一组
DeleteMarkers	[]*DeleteMarkerEntry	对象删除标记数组，每一项包含了是否为最新版本，对象 `key`，最新修改时间，拥有者和版本号等信息
Delimiter	*string	与请求中设置的 `Delimiter` 相同
EncodingType	*string	返回对象 `key` 的字符编码类型
IsTruncated	*bool	表示是否返回满足搜索条件的所有结果的标志
KeyMarker	*string	与请求中设置的 `KeyMarker` 相同
MaxKeys	*int64	返回结果的对象数量的最大值
Name	*string	执行本操作的桶名称
NextKeyMaker	*string	当响应被截断时（`IsTruncated` 为 `true`），可以在下次查询请求时用该字段作为 `KeyMarker` 来获取下一部分的对象版本信息
NextVersionIdMarker	*string	本次查询返回的最后一个对象的版本号
Prefix	*string	与请求中设置的 `Prefix` 一致
Versions	[]*ObjectVersion	对象版本信息数组

代码示例

```
func ListObjectVersions() {
    input := &s3.ListObjectVersionsInput{
        Bucket: aws.String(bucket),
    }
    req, _ := s3c.ListObjectVersionsRequest(input)
    send(req)
}
```

解冻对象

功能介绍

解冻归档存储类型的对象。

方法原型

```
func (c *S3) RestoreObjectRequest(input *RestoreObjectInput) (req
*request.Request, output *RestoreObjectOutput)
```

输入参数

属性名	类型	说明	是否必要
Bucket	*string	桶的名称	是
Key	*string	对象的名称	是
VersionId	*string	对象的版本号	否
RestoreRequest	*RestoreRequest	解冻操作相关参数	是

`RestoreRequest` 属性表

属性名	类型	说明	是否必要
Days	*int64	解冻后副本的保留时间，支持范围为 [1, 31]	是

输出结果

无

代码示例

```
func RestoreObject() {
    input := &s3.RestoreObjectInput{
        Bucket: aws.String(bucket),
        Key:    aws.String(key),
        RestoreRequest: &s3.RestoreRequest{
            Days: aws.Int64(1),
        },
    }
    req, _ := s3c.RestoreObjectRequest(input)
    send(req)
}
```

分段上传操作

初始化分段上传

功能介绍

初始化分段上传，该请求返回的 `UploadId` 将用于本次分段上传的其他请求。

方法原型

```
func (c *S3) CreateMultipartUploadRequest(input *CreateMultipartUploadInput) (req *request.Request, output *CreateMultipartUploadOutput)
```

输入参数

属性名	类型	说明	是否必要
ACL	*string	配置上传对象的预定义 ACL 信息，如 `private`、`public-read`、`public-read-write` 等。不传该参数时默认为 `private`	否
Bucket	*string	桶的名称	是
Key	*string	对象的名称	是
StorageClass	*Tagging	存储类型，可选的有 `STANDARD`，`STANDARD_IA`，`GLACIER`，默认为 `STANDARD`	否
Tagging	*Tagging	封装标签键值对的数组	否

`Tagging` 属性表

属性名	类型	说明
TagSet	[]*Tag	标签集合

`Tag` 属性表

属性名	类型	说明
Key	*string	键
Value	*string	值

输出结果

属性名	类型	说明
Bucket	*string	桶的名称
Key	*string	对象的名称
UploadId	*string	分段上传的唯一标识

代码示例

```
func CreateMultipartUpload() {
    input := &s3.CreateMultipartUploadInput{
        Bucket: aws.String(bucket),
        Key:    aws.String(key),
    }
    req, _ := s3c.CreateMultipartUploadRequest(input)
    send(req)
}
```

进行分段上传

功能介绍

进行分段上传，支持的分段数范围为 [1, 10000]。每次请求时均需要携带 `PartNumber` 和 `UploadId` 参数，支持乱序上传。

注：每个分段大小不得大于 5GB，且除最后一个分段外，每个分段的大小不得小于 5MB。

方法原型

```
func (c *S3) UploadPartRequest(input *UploadPartInput) (req *request.Request, output *UploadPartOutput)
```

输入参数

属性名	类型	说明	是否必要
Bucket	*string	桶的名称	是
Key	*string	初始化分段上传所用的对象名称	是
Body	io.ReadSeeker	分段的数据	是
UploadId	*string	分段上传的唯一标识	是
PartNumber	*int64	分段编号，可用范围： [1, 10000]	是

输出结果

属性名	类型	说明
ETag	*string	分段的 `Entity Tag`

代码示例

```
func UploadPart() {
    data := bytes.NewReader([]byte("data"))
    input := &s3.UploadPartInput{
        Bucket:     aws.String(bucket),
        Key:        aws.String(key),
        UploadId:   aws.String("xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"),
        Body:       data,
        PartNumber: aws.Int64(2),
    }
    req, _ := s3c.UploadPartRequest(input)
    send(req)
}
```

列出分段

功能介绍

查询指定分段上传中已成功上传的分段的信息。

方法原型

```
func (c *S3) ListPartsRequest(input *ListPartsInput) (req *request.Request, output *ListPartsOutput)
```

输入参数

属性名	类型	说明	是否必要
Bucket	*string	桶的名称	是
Key	*string	初始化分段上传所用的对象名称	是
MaxParts	*int64	设置返回分段数量的最大值，默认值为 1000，可选范围：[1, 1000]	否
PartNumberMarker	*int64	指定返回结果的分段编号起始位置，只有分段编号大于该值的分段会被返回	否
UploadId	*string	分段上传的唯一标识	是

输出结果

属性名	类型	说明
Bucket	*string	桶的名称
IsTruncated	*bool	表示本次响应是否被截断
Key	*string	初始化分段上传所用的对象名称
MaxParts	*int64	本次响应可返回分段数量的最大值
NextPartNumberMarker	*int64	当响应被截断时返回，可作为下次请求的 `PartNumberMarker` 参数以进行连续查询
Owner	*Owner	分段上传所属的用户
PartNumberMarker	*int64	当前响应中分段的起始编号
Parts	[]*Part	分段信息数组

`Owner` 属性表

属性名	类型	说明
DisplayName	*string	用户名称
ID	*string	用户 ID

`Part` 属性表

属性名	类型	说明
ETag	*string	分段的 `Entity Tag`
LastModified	*time.Time	该分段完成上传的时间点
PartNumber	*int64	分段编号
Size	*int64	分段大小，单位为字节

代码示例

```
func ListParts() {
    input := &s3.ListPartsInput{
        Bucket:   aws.String(bucket),
        Key:      aws.String(key),
        UploadId: aws.String("xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"),
    }
    req, _ := s3c.ListPartsRequest(input)
    send(req)
}
```

中止分段上传

功能介绍

中止分段上传。将弃用对应的 `UploadId` 并舍弃之前在该分段上传中上传的所有分段。

方法原型

```
func (c *S3) AbortMultipartUploadRequest(input *AbortMultipartUploadInput) (req
*request.Request, output *AbortMultipartUploadOutput)
```

输入参数

属性名	类型	说明	是否必要
Bucket	*string	桶的名称	是
Key	*string	初始化分段上传所用的对象名称	是
UploadId	*string	分段上传的唯一标识	是

输出结果

无

代码示例

```
func AbortMultipartUpload() {
    input := &s3.AbortMultipartUploadInput{
        Bucket:   aws.String(bucket),
        Key:      aws.String(key),
        UploadId: aws.String("xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"),
    }
    req, _ := s3c.AbortMultipartUploadRequest(input)
    send(req)
}
```

完成分段上传

功能介绍

完成分段上传。当指定分段上传所需的所有分段均已成功上传之后，可通过该接口完成分段上传。

该请求需要提供分段上传过程中每个成功上传分段的 `PartNumber` 和 `ETag` 作为参数以校验分段的有效性。

方法原型

```
func (c *S3) CompleteMultipartUploadRequest(input *CompleteMultipartUploadInput)
(req *request.Request, output *CompleteMultipartUploadOutput)
```

输入参数

属性名	类型	说明	是否必要
Bucket	*string	桶的名称	是
Key	*string	初始化分段上传所用的对象名称	是
UploadId	*string	分段上传的唯一标识	是
MultipartUpload	*CompletedMultipartUpload	分段上传相关信息	是

`CompletedMultipartUpload` 属性表

属性名	类型	说明	是否必要
Parts	[]*CompletedPart	已完成的分段上传信息数组	是

`CompletedPart` 属性表

属性名	类型	说明
ETag	*string	分段的 `Entity Tag`
PartNumber	*int64	分段编号

输出结果

属性名	类型	说明
Bucket	*string	桶的名称
Key	*string	初始化分段上传所用的对象名称
ETag	*string	最终上传对象的 `Entity Tag`
Location	*string	最终上传对象的具体位置路径
VersionId	*string	最终上传对象的的版本号

代码示例

```
func CompletedMultipartUpload() {
    multipartUpload := &s3.CompletedMultipartUpload{
        Parts: []*s3.CompletedPart{
            {
                ETag:       aws.String("\"xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx\""),
                PartNumber: aws.Int64(1),
            },
            {
                ETag:       aws.String("\"xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx\""),
                PartNumber: aws.Int64(2),
            },
```

```
        },
    }
    input := &s3.CompleteMultipartUploadInput{
        Bucket:          aws.String(bucket),
        Key:             aws.String(key),
        UploadId:        aws.String("xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"),
        MultipartUpload: multipartUpload,
    }
    req, _ := s3c.CompleteMultipartUploadRequest(input)
    send(req)
}
```

查询正在进行的分段上传

功能介绍

查询正在进行的分段上传。

方法原型

```
func (c *S3) ListMultipartUploadsRequest(input *ListMultipartUploadsInput) (req
*request.Request, output *ListMultipartUploadsOutput)
```

输入参数

属性名	类型	说明	是否必要
Bucket	*string	桶的名称	是
MaxUploads	*int64	设置返回分段上传数量的最大值，默认值为 1000，可选范围：[1, 1000]	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。 若未设置 `UploadIdMarker` 参数，则返回对象 `Key` 按字典序大于等于 `KeyMarker` 的分段上传信息。 若设置了 `UploadIdMarker` 参数，则返回对象 `Key` 大于等于 `KeyMarker` 且 `UploadId` 大于 `UploadIdMarker` 的片段信息	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，使用方法参考 `KeyMarker` 参数说明	否
Prefix	*string	筛选参数，只有 `Key` 以 `Prefix` 为前缀的分段上传会被返回	否

输出结果

属性名	类型	说明
Bucket	*string	桶的名称
IsTruncated	*bool	表示本次响应是否被截断
MaxUploads	*int64	本次响应可返回分段上传数量的最大值
NextKeyMarker	*string	当响应被截断时返回，可作为下次请求的 `KeyMarker` 参数以进行连续查询
NextUploadIdMarker	*string	当响应被截断时返回，可作为下次请求的 `UploadIdMarker` 参数以进行连续查询
Uploads	[]*MultipartUpload	分段上传信息数据
KeyMarker	*string	与请求中设置的 `KeyMarker` 一致
Prefix	*string	与请求中设置的 `Prefix` 一致
UploadIdMarker	*string	与请求中设置的 `UploadIdMarker` 一致

`MultipartUpload` 属性表

属性名	类型	说明
Initiated	*time.Time	分段上传初始化的时间点
Initiator	*Initiator	分段上传初始化的创建者用户
Key	*string	初始化分段上传所用的对象名称
Owner	*Owner	分段上传对象的所有者用户
StorageClass	*string	分段上传对象的存储类型
UploadId	*string	分段上传的唯一标识

`Initiator` 属性表

属性名	类型	说明
DisplayName	*string	用户名称
ID	*string	用户 ID

`Owner` 属性表

属性名	类型	说明
DisplayName	*string	用户名称
ID	*string	用户 ID

代码示例

```
func ListMultipartUploads() {
    input := &s3.ListMultipartUploadsInput{
        Bucket: aws.String(bucket),
    }
    req, _ := s3c.ListMultipartUploadsRequest(input)
    send(req)
}
```

图片操作

处理图片并输出至桶中

功能介绍

对指定的图片进行处理并输出到桶中。

支持的处理操作：裁剪、旋转、水印、缩放、格式转换

支持处理的图片格式为：JPG、PNG、WEBP、BMP、TIFF

方法原型

```
func (c *S3) ProcessObjectRequest(input *ProcessObjectInput) (req
*request.Request, output *ProcessObjectOutput)
```

输入参数

属性名	类型	说明	是否必要
Bucket	*string	保存处理结果的桶的名称	是
Key	*string	处理结果的对象名	是
ProcessSource	*string	待处理的图片路径，格式为 `{Bucket}/{Object}[?versionid=]`	是
ZosProcess	*string	图片的处理方式，多种处理可直接拼接，例如 `image/crop,xxx/resize,xxx`	是

`ZosProcess` 支持的处理方法详情如下

1. 图片缩放

示例：`image/resize,w_300,h_200,m_fixed`

`resize` 属性表

属性名	说明	取值	是否必要
w	指定目标缩放图宽度	[1, 4096]	使用按百分比缩放可不指定宽高
h	指定目标缩放图高度	[1, 4096]	使用按百分比缩放可不指定宽高
m	指定缩放模式	参考下文 `resize.m` 属性表	否
color	缩放模式（`m`）为 `pad` 时，指定填充颜色	RGB 颜色值，默认 FFFFFF（白色）	否（仅当 `m` 为 `pad` 时使用）
p	按百分比进行缩放	[1, 1000]，小于100为缩小，大于100为放大	否
limit	指定目标缩放图大于原图时是否缩放	1(默认)：目标缩放图大于原图时返回原图 0: 按指定参数缩放	否

`resize.m` 属性表

属性名	说明
lfit	默认值，表示等比缩放，目标缩放图为指定的 `w` 和 `h` 矩形框内的最大图形
mfit	等比缩放，目标缩放图为延伸出指定的 `w` 和 `h` 矩形框外的最小图形
fill	将原图等比缩放为延伸出指定 `w` 与 `h` 的矩形框外的最小图片，之后将超出的部分进行居中裁剪
pad	将原图等比缩放为指定 `w` 与 `h` 的矩形框内的最大图形，然后使用 `color` 指定的颜色将矩形框内剩余部分填充
fixed	固定宽高，强制缩放

2. 图片裁剪

示例：`image/crop,w_100,h_100,x_10,y_10,g_se`

`crop` 属性表

属性名	说明	取值	是否必要
w	指定裁剪宽度	[0, 图片宽度]，默认为最大值	否
h	指定裁剪高度	[0, 图片高度]，默认为最大值	否
x	指定裁剪起点横坐标（默认左上角为原点）	[0, 图片边界]	否
y	指定裁剪起点纵坐标（默认左上角为原点）	[0, 图片边界]	否
g	设置裁剪的原点位置。原点按九宫格顶点形式分布	nw(默认)表示左上；north 表示中上；ne 表示右上；west 表示左中；center 表示正中；east 表示右中；sw 表示左下；south 表示中下；se 表示右下	否

3. 图片旋转

示例：`image/rotate,45`

`rotate` 属性表

属性名	说明	取值	是否必要
无，直接填写数值即可	图片按顺时针旋转的角度	[0, 360]，默认为0，即不旋转	是

4. 添加水印

添加图片水印示例：
`image/watermark,image_aHVkaWUuanBnP3gtem9zLXByb2Nlc3M9aW1hZ2UvcmVzaXplLHBfMzAvcm90YXRlLDE4MA==,g_north,t_40`

添加文字水印示例：
`image/watermark,text_Q2hpbmFOZWxlY29t,type_heiti,color_FF0000,size_40,g_se,t_80`

`watermark` 属性表

属性名	说明	取值	是否必要
t	水印的透明度	[0, 100]	否
x	文字水印距离图片边界的水平距离	[0, 4096]，默认值为10	否
y	文字水印距离图片边界的垂直距离	[0, 4096]，默认值为10	否
text	指定文字水印内容	Base64 编码后的字符串，编码结果中的 `/` 需替换为 `_`	
color	指定文字水印的颜色	RGB 颜色值。默认为 FFFFFF（白色）	否
size	指定文字水印的字体大小	默认值：40	否
type	指定文字水印的字体类型	如 Airal, Helvetica，支持的中文字体包括 yahei(微软雅黑), heiti(黑体), kaishu(楷书), youyuan(幼圆)	否
rotate	指定文字水印顺时针旋转角度	[0, 360]，默认值为0	否
image	指定图片水印名称，水印图片需和原图存放在相同桶内	水印图片可以被预处理（e.g. 水印图片缩放为 30% 并旋转 180 度：hudie.jpg/?x-zos-process=image/resize,p_30/rotate,180）。 需转换为 Base64 编码，编码结果字符串中的 `/` 替换为 `_`	否
g	指定水印在图片中的位置	nw(默认)表示左上；north 表示中上；ne 表示右上；west 表示左中；center 表示正中；east 表示右中；sw 表示左下；south 表示中下；se 表示右下	否

5. 格式转化

示例：`image/format,png`

`format` 属性表

属性名	说明	取值	是否必要
无，直接填写数值即可	将原图转换成指定格式	jpg、png、webp、bmp、tiff	是

输出结果

属性名	类型	说明
ETag	*string	经过处理的图片对象的 `ETag`
VersionId	*string	经过处理的图片对象的的版本号

代码示例

```
func ProcessObject() {
    input := &s3.ProcessObjectInput{
        Bucket:        aws.String(bucket),
        Key:           aws.String(`after.jpg`),
        ProcessSource: aws.String(bucket + "/example.jpg"),
        ZosProcess:    aws.String(`image/crop,w_100,h_100,x_10,y_10,g_se`),
    }
    req, _ := s3c.ProcessObjectRequest(input)
    send(req)
}
```

处理图片并下载

功能介绍

对指定的图片进行处理并下载，使用前文所述的 `GetObject` 配合 `ZosProcess` 参数即可实现。

支持的处理操作：裁剪、旋转、水印、缩放、格式转换、获取 EXIF 信息

支持处理的图片格式为：JPG、PNG、WEBP、BMP、TIFF

方法原型

```
func (c *S3) GetObject(input *GetObjectInput) (*GetObjectOutput, error)
```

输入参数

`GetObject` 所使用的其他输入参数请参考“对象操作”的“获取对象”小节。这里只描述图片处理所需的相关输入参数。

属性名	类型	说明	是否必要
Bucket	*string	保存处理结果的桶的名称	是
Key	*string	处理结果的对象名	是
ProcessSource	*string	待处理的图片路径，格式为 `{Bucket}/{Object}[?Versionid=]`	是
ZosProcess	*string	图片的处理方式，多种处理可直接拼接，例如 `image/crop,xxx/resize,xxx`	是

`ZosProcess` 支持的处理方法详情如下

1. 图片缩放

示例：`image/resize,w_300,h_200,m_fixed`

`resize` 属性表

属性名	说明	取值	是否必要
w	指定目标缩放图宽度	[1, 4096]	使用按百分比缩放可不指定宽高
h	指定目标缩放图高度	[1, 4096]	使用按百分比缩放可不指定宽高
m	指定缩放模式	参考下文 `resize.m` 属性表	否
color	缩放模式（`m`）为 `pad` 时，指定填充颜色	RGB 颜色值，默认 FFFFFF（白色）	否（仅当 `m` 为 `pad` 时使用）
p	按百分比进行缩放	[1, 1000]，小于100为缩小，大于100为放大	否
limit	指定目标缩放图大于原图时是否缩放	1(默认)：目标缩放图大于原图时返回原图 0: 按指定参数缩放	否

`resize.m` 属性表

属性名	说明
lfit	默认值，表示等比缩放，目标缩放图为指定的 `w` 和 `h` 矩形框内的最大图形
mfit	等比缩放，目标缩放图为延伸出指定的 `w` 和 `h` 矩形框外的最小图形
fill	将原图等比缩放为延伸出指定 `w` 与 `h` 的矩形框外的最小图片，之后将超出的部分进行居中裁剪
pad	将原图等比缩放为指定 `w` 与 `h` 的矩形框内的最大图形，然后使用 `color` 指定的颜色将矩形框内剩余部分填充
fixed	固定宽高，强制缩放

2. 图片裁剪

示例：`image/crop,w_100,h_100,x_10,y_10,g_se`

`crop` 属性表

属性名	说明	取值	是否必要
w	指定裁剪宽度	[0, 图片宽度]，默认为最大值	否
h	指定裁剪高度	[0, 图片高度]，默认为最大值	否
x	指定裁剪起点横坐标（默认左上角为原点）	[0, 图片边界]	否
y	指定裁剪起点纵坐标（默认左上角为原点）	[0, 图片边界]	否
g	设置裁剪的原点位置。原点按九宫格顶点形式分布	nw(默认)表示左上；north 表示中上；ne 表示右上；west 表示左中；center 表示正中；east 表示右中；sw 表示左下；south 表示中下；se 表示右下	否

3. 图片旋转

示例：`image/rotate,45`

`rotate` 属性表

属性名	说明	取值	是否必要
无，直接填写数值即可	图片按顺时针旋转的角度	[0, 360]，默认为0，即不旋转	是

4. 添加水印

添加图片水印示例：
`image/watermark,image_aHVkaWUuanBnP3gtem9zLXByb2Nlc3M9aW1hZ2UvcmVzaXplLHBfMzAvcm90YXRlLDE4MA==,g_north,t_40`

添加文字水印示例：
`image/watermark,text_Q2hpbmFOZWxlY29t,type_heiti,color_FF0000,size_40,g_se,t_80`

`watermark` 属性表

属性名	说明	取值	是否必要
t	水印的透明度	[0, 100]	否
x	文字水印距离图片边界的水平距离	[0, 4096]，默认值为10	否
y	文字水印距离图片边界的垂直距离	[0, 4096]，默认值为10	否
text	指定文字水印内容	Base64 编码后的字符串，编码结果中的 `/` 需替换为 `_`	
color	指定文字水印的颜色	RGB 颜色值。默认为 FFFFFF（白色）	否
size	指定文字水印的字体大小	默认值：40	否
type	指定文字水印的字体类型	如 Airal, Helvetica，支持的中文字体包括 yahei(微软雅黑), heiti(黑体), kaishu(楷书), youyuan(幼圆)	否
rotate	指定文字水印顺时针旋转角度	[0, 360]，默认值为0	否
image	指定图片水印名称，水印图片需和原图存放在相同桶内	水印图片可以被预处理（e.g. 水印图片缩放为 30% 并旋转 180 度：hudie.jpg/?x-zos-process=image/resize,p_30/rotate,180）。 需转换为 Base64 编码，编码结果字符串中的 `/` 替换为 `_`	否
g	指定水印在图片中的位置	nw(默认)表示左上；north 表示中上；ne 表示右上；west 表示左中；center 表示正中；east 表示右中；sw 表示左下；south 表示中下；se 表示右下	否

5. 格式转化

示例：`image/format,png`

`format` 属性表

属性名	说明	取值	是否必要
无，直接填写数值即可	将原图转换成指定格式	jpg、png、webp、bmp、tiff	是

6. 获取图片 EXIF 信息

示例：`image/info`

输出结果

属性名	类型	说明
AcceptRanges	*string	指出一段字节的范围
Body	io.ReadClosed	对象数据
ContentLength	*int64	以字节为单位的对象数据长度
ContentType	*string	描述对象数据格式的 MIME 类型
ETag	*string	对象的 `ETag`
LastModified	*time.Time	最后修改对象的时间
TagCount	*int64	对象上标签的数量(如果有)
VersionId	*string	对象版本

代码示例

```
func GetProcessedObject() {
    input := &s3.GetObjectInput{
        Bucket:     aws.String(bucket),
        Key:        aws.String("example.jpg"),
        ZosProcess:
aws.String(`image/watermark,text_Q2hpbmFOZWxlY29t,type_heiti,color_FF0000,size_40
,g_center,t_80`),
    }
    output, err := s3c.GetObject(input)
    if err != nil {
        fmt.Printf("* Status: Fail\n* Output:\n%v\n* Error:\n%v\n", output, err)
    } else {
        // 将获取到对象写入当前文件夹的 output 文件
        reader := output.Body
        defer reader.Close()
        file, err := os.Create("output")
        if err != nil {
            panic(err)
        }
        defer file.Close()
        if _, err := io.Copy(file, reader); err != nil {
            panic(err)
        }
        fmt.Printf("* Status: Susccess\n* Output:\n%v\n", output)
    }
}
```

其他操作

预签名下载

功能介绍

通过预签名链接下载对象。

使用`Presign`方法可以对请求进行预签名，生成预签名链接。任意用户在预签名链接的有效期内均可通过该链接完成`GetObject`请求。

预签名下载链接的使用方式可参考下面的指令，将文件名和链接替换为自己的参数即可。

```
curl -Lo your_local_file_name 'your_presign_url'
```

方法原型

```
// 用于构造 `GetObject` 请求
func (c *S3) GetObjectRequest(input *GetObjectInput) (req *request.Request,
output *GetObjectOutput)
// 用于预签名请求
func (r *Request) Presign(expire time.Duration) (string, error)
```

输入参数

构造`GetObject`请求所使用的输入参数请参考“对象操作”的“获取对象”小节。这里只描述预签名方法所需的输入参数。

属性名	类型	说明	是否必要
expire	time.Duration	预签名链接的过期时间	是

输出结果

属性名	类型	说明
u	string	预签名链接

代码示例

```
func PresignGetObjectRequest() {
    input := &s3.GetObjectInput{
        Bucket: aws.String(bucket),
        Key:    aws.String(key),
    }
    // 构造请求
    request, _ := s3c.GetObjectRequest(input)
    // 设定预签名链接过期时间
    expire := 15 * time.Minute
    // 预签名请求
    output, err := request.Presign(expire)
    if err != nil {
        fmt.Printf("* Status: Fail\n* Output:\n%v\n* Error:\n%v\n", output, err)
    } else {
        fmt.Printf("* Status: Susccess\n* Output:\n%v\n", output)
    }
```

```
}
```

预签名上传

功能介绍

通过预签名链接上传对象。

使用`Presign`方法可以对请求进行预签名，生成预签名链接。任意用户在预签名链接的有效期内均可通过该链接完成`PutObject`请求。

预签名上传链接的使用方式可参考下面的指令，将文件名和链接替换为自己的参数即可。

```
curl -v -k -X PUT -T your_local_file_name 'your_presign_url'
```

方法原型

```
// 用于构造 `PutObject` 请求
func (c *S3) PutObjectRequest(input *PutObjectInput) (req *request.Request,
output *PutObjectOutput)
// 用于预签名请求
func (r *Request) Presign(expire time.Duration) (string, error)
```

输入参数

构造`PutObject`请求所使用的输入参数请参考“对象操作”的“上传对象”小节。这里只描述预签名方法所需的输入参数。

属性名	类型	说明	是否必要
expire	time.Duration	预签名链接的过期时间	是

输出结果

属性名	类型	说明
u	string	预签名链接

代码示例

```
func PresignPutObjectRequest() {
    input := &s3.PutObjectInput{
        Bucket: aws.String(bucket),
        Key:    aws.String(key),
    }
    // 构造请求
    request, _ := s3c.PutObjectRequest(input)
    // 设定预签名链接过期时间
    expire := 15 * time.Minute
    // 预签名请求
    output, err := request.Presign(expire)
    if err != nil {
        fmt.Printf("* Status: Fail\n* Output:\n%v\n* Error:\n%v\n", output, err)
    } else {
```

```
        fmt.Printf("* Status: Susccess\n* Output:\n%v\n", output)
    }
}
```

获取使用情况统计信息

功能介绍

获取用户维度的使用情况统计信息。

方法原型

```
func (c *S3) GetUsageStatsRequest(input *GetUsageStatsInput) (req
*request.Request, output *GetUsageStatsOutput)
```

输入参数

无

输出结果

属性名	类型	说明
Summary	*UsageStatsSummary	用户维度的总使用情况统计信息

`Summary` 属性表

属性名	类型	说明
TotalBytes	*int64	用户使用的字节数
TotalBytesRounded	*int64	用户使用的使用字节数（按 4K 边界计算）
TotalEntries	*int64	对象实体数量（包括完整对象与碎片）

代码示例

```
func GetUsageStats() {
    req, _ := s3c.GetUsageStatsRequest(&s3.GetUsageStatsInput{})
    send(req)
}
```

附录

错误码总表

请求时返回的错误码含义可参考下表。

HTTP状态码	错误码	说明
403	AccessDenied	访问被拒绝
400	BadDigest	指定的 Content-MD5 与接收到的不匹配
409	BucketAlreadyExists	指定的桶已存在
409	BucketNotEmpty	桶内非空
400	EntityTooSmall	上传对象小于允许的最小大小
400	EntityTooLarge	上传对象超过允许的最大大小
400	ExpiredToken	提供的令牌已过期
400	IncompleteBody	未提供 Content-Length HTTP 头中指定的字节数
400	IncorrectNumberOfFilesInPostRequest	POST 请求每个请求只允许上传一个文件
500	InternalError	内部错误，请重试
403	InvalidAccessKeyId	无效的访问密钥
400	InvalidArgument	无效的参数
400	InvalidBucketName	无效的桶名称
409	InvalidBucketState	请求与桶的当前状态不符
400	InvalidDigest	指定的 Content-MD5 无效
403	InvalidObjectState	请求与对象的当前状态不符
400	InvalidPart	无效的分段
400	InvalidPartOrder	无效的分段顺序
400	InvalidPolicyDocument	无效的策略参数
416	InvalidRange	无法的请求范围
403	InvalidSecurity	无效的秘密密钥
400	InvalidStorageClass	无效的存储类型
400	InvalidTargetBucketForLogging	无效的日志目标存储桶
400	InvalidToken	无效的令牌格式
400	InvalidURI	无法解析指定的URI
400	KeyTooLongError	对象名过长
400	MalformedACLError	错误的 ACL 格式
400	MalformedPOSTRequest	错误的 POST 请求体格式

HTTP状态码	错误码	说明
400	MalformedXML	错误的 XML 格式
400	MaxMessageLengthExceeded	请求信息过长
405	MethodNotAllowed	指定的方法不被允许对此资源进行操作
411	MissingContentLength	未提供 Content-Length HTTP 头
400	MissingRequestBodyError	缺少请求体
404	NoSuchBucket	指定的桶不存在
404	NoSuchBucketPolicy	指定的桶没有桶策略
404	NoSuchKey	指定的对象不存在
404	NoSuchLifecycleConfiguration	指定的桶没有生命周期配置
404	NoSuchUpload	指定的分段上传不存在
404	NoSuchVersion	请求中指定的版本号与现有版本不匹配
409	OperationAborted	操作被中止，请重试
307	Redirect	临时重定向
409	RestoreAlreadyInProgress	解冻进行中
400	RequestIsNotMultiPartContent	POST 请求体不符合 multipart/form-data 格式
400	RequestTimeout	请求超时
403	RequestTimeTooSkewed	请求时间与服务器的时间差值过大
403	SignatureDoesNotMatch	服务计算的请求签名与您提供的签名不匹配
400	TooManyBuckets	桶数量超额
400	TokenRefreshRequired	提供的令牌需要刷新
400	UnresolvableGrantByEmailAddress	提供的电子邮件地址与记录中的任何帐户都不匹配