天翼云媒体存储

.NET SDK 使用指导书

天翼云科技有限公司

2026年3月27日

天翼云媒体存储.NET SDK使用指导书

快速使用

创建一个.NET项目：

打开命令提示符或者终端，执行以下命令创建一个.NET项目。

```
dotnet new console --name DotNetSDK
cd DotNetSDK
```

安装SDK：

在天翼云官网下载，下载地址： xos-dotnet-sdk.zip

修改项目的csproj文件，在<PropertyGroup>中增加以下内容。

```
<PropertyGroup>
    <RestoreSources>$(RestoreSources);filePathToPackage</RestoreSources>
</PropertyGroup>
```

其中filePathToPackage指的是XOS_DOTNET_SDK.zip解压后的径。然后在项目csproj文件所在目录下执行dotnet命令安装依赖包：

```
dotnet add package AWSSDK.Core --version 3.7.0.18
dotnet add package AWSSDK.S3 --version 3.7.0.18
# 使用sts服务需要添加以下依赖
dotnet add package AWSSDK.SecurityToken --version 3.7.1.6
dotnet restore
```

创建代码：

注意：直接在客户端上使用主账号存在账号泄露的风险，在客户端上必须使用sts功能生成的临时账号，此初始化流程只能用于测试。如何使用sts初始化参考 本文的安全凭证服务(STS)章节。

修改DotNetSDK文件夹中的Program.cs文件，用以下代码替换内容并保存文件。

```
using System;
using System.Threading.Tasks;
using Amazon.Runtime;
using Amazon.S3;

namespace DotNetSDK
{
    class Program
    {
        static async Task Main(string[] args)
        {
            var accessKey = "<your-access-key>";
            var secretKey = "<your-secret-access-key>";
            var endpoint = "<your-endpoint>"; // e.g. http://endpoint or
https://endpoint
```

```
            try
            {
                var credentials = new BasicAWSCredentials(accessKey, secretKey);
                var conf = new AmazonS3Config
                {
                    ServiceURL = endpoint
                };
                var s3Client = new AmazonS3Client(credentials, conf);
                //创建一个bucket
                await s3Client.PutBucketAsync("<your-bucket-name>");
                //列出bucket
                var result = await s3Client.ListBucketsAsync();
                Console.WriteLine("the buckets of {0} are:",
result.Owner.DisplayName);
                result.Buckets.ForEach(b => { Console.WriteLine(b.BucketName);
});
            }
            catch (Exception e)
            {
                Console.WriteLine("e.Message");
                Console.WriteLine(e.Message);
            }
        }
    }
}
```

参数	说明
accessKey	用户账号 access key
secretKey	用户账号 secret key
endpoint	天翼云资源池的地址，必须指定http或https前缀

执行以下命令运行代码。

```
dotnet run
```

环境配置

使用XOS .NET SDK需要做以下准备：

安装.NET

支持.NET Core 3.1及以上版本，.NET Framework 3.5及以上版本。

获取访问密钥

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

获取Endpoint

EndPoint的获取方式请查阅天翼云媒体存储 基础信息查看。

桶相关接口

创建桶

功能说明

PutBucket操作用于创建桶（bucket），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所有者，对桶拥有FULL_CONTROL权限，可以通过设置参数的方式为其他用户配置创建桶的权限。桶的命名规范如下：

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

代码示例

```
using System;
using System.Threading.Tasks;
using Amazon.Runtime;
using Amazon.S3;
using Amazon.S3.Model;

namespace DotNetSDK.BucketOperation
{
    public class PutBucketExample
    {
        public static async Task PutBucket()
        {
            var accessKey = "<your-access-key>";
            var secretKey = "<your-secret-access-key>";
            var endpoint = "<your-endpoint>";
            var bucketName = "<your-bucket-name>";
            try
            {
                var credentials = new BasicAWSCredentials(accessKey, secretKey);
                var conf = new AmazonS3Config
                {
                    ServiceURL = endpoint
                };
                var s3Client = new AmazonS3Client(credentials, conf);
                var putBucketRequest = new PutBucketRequest()
                {
                    BucketName = bucketName
                };

                var result = await s3Client.PutBucketAsync(putBucketRequest);
                if (result.HttpStatusCode != System.Net.HttpStatusCode.OK)
                {
                    Console.WriteLine("fail to create bucket {0},
HttpStatusCode:{1}, ErrorCode:{2}.", bucketName, (int) result.HttpStatusCode,
result.HttpStatusCode);
                    return;
                }

                Console.WriteLine("create bucket {0} success.", bucketName);
```

```
            }
            catch (Exception e)
            {
                Console.WriteLine(e.Message);
            }
        }
    }
}
```

请求参数

PutBucket可设置的参数如下：

参数	类型	说明	是否必要
BucketName	string	创建桶的名称	是

获取桶列表

功能说明

桶（Bucket），是用于存储对象（Object）的容器，所有的对象都必须隶属于某个桶。用户可以设置和修改存储空间属性用来设置访问权限、生命周期等，这些属性设置直接作用于该存储空间内所有对象，因此可以通过灵活创建不同的存储空间来完成不同的管理功能。用户需通过身份验证来查询自己创建的桶，且无法匿名发送请求。

ListBuckets操作列出用户创建的桶。

代码示例

```
using System;
using System.Threading.Tasks;
using Amazon.Runtime;
using Amazon.S3;

namespace DotNetSDK.BucketOperation
{
    public class ListBucketExample
    {
        public static async Task ListBuckets()
        {
            var accessKey = "<your-access-key>";
            var secretKey = "<your-secret-access-key>";
            var endpoint = "<your-endpoint>";
            try
            {
                var credentials = new BasicAWSCredentials(accessKey, secretKey);
                var conf = new AmazonS3Config
                {
                    ServiceURL = endpoint
                };

                var s3Client = new AmazonS3Client(credentials, conf);
                var result = await s3Client.ListBucketsAsync();
                if (result.HttpStatusCode != System.Net.HttpStatusCode.OK)
```

```
                {
                    Console.WriteLine("fail to list buckets, HttpStatusCode:{0},
ErrorCode:{1}.", (int) result.HttpStatusCode, result.HttpStatusCode);
                    return;
                }

                Console.WriteLine("the buckets of {0} are:",
result.Owner.DisplayName);
                result.Buckets.ForEach(b => { Console.WriteLine(b.BucketName);
});
            }
            catch (Exception e)
            {
                Console.WriteLine(e.Message);
            }
        }
    }
}
```

返回结果

ListBuckets操作返回的结果如下：

属性名	类型	说明
Buckets	List	桶信息的数组，包含了每个桶的名字和创建时间
Owner	Owner	桶的所有者信息

判断桶是否存在

功能说明

可以使用AmazonS3Util.DoesS3BucketExistAsync接口判断桶是否存在。

代码示例

```
using System;
using System.Threading.Tasks;
using Amazon.Runtime;
using Amazon.S3;
using Amazon.S3.Util;

namespace DotNetSDK.BucketOperation
{
    public class DoesBucketExistExample
    {
        public static async Task DoesBucketExist()
        {
            var accessKey = "<your-access-key>";
            var secretKey = "<your-secret-access-key>";
            var endpoint = "<your-endpoint>";
            var bucketName = "<your-bucket-name>";
            try
            {
```

```
                var credentials = new BasicAWSCredentials(accessKey, secretKey);
                var conf = new AmazonS3Config
                {
                    ServiceURL = endpoint
                };
                var s3Client = new AmazonS3Client(credentials, conf);
                var exist = AmazonS3Util.DoesS3BucketExistAsync(s3Client,
bucketName);
                if (exist.Result)
                {
                    Console.Out.WriteLine("bucket exist");
                } else
                {
                    Console.Out.WriteLine("bucket not exist");
                }
            }
            catch (Exception e)
            {
                Console.WriteLine(e.Message);
            }
        }
    }
}
```

请求参数

参数	类型	说明	是否必要
bucketName	string	桶名称	是

返回结果

DoesS3BucketExistAsync操作返回的结果如下：

属性名	类型	说明
Result	bool	true表示桶存在，false表示桶不存在

删除桶

功能说明

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

代码示例

```
using System;
using System.Threading.Tasks;
using Amazon.Runtime;
using Amazon.S3;
using Amazon.S3.Model;

namespace DotNetSDK.BucketOperation
```

```
{
    public class DeleteBucketExample
    {
        public static async Task DeleteBucket()
        {
            var accessKey = "<your-access-key>";
            var secretKey = "<your-secret-access-key>";
            var endpoint = "<your-endpoint>";
            var bucketName = "<your-bucket-name>";
            try
            {
                var credentials = new BasicAWSCredentials(accessKey, secretKey);
                var conf = new AmazonS3Config
                {
                    ServiceURL = endpoint
                };
                var s3Client = new AmazonS3Client(credentials, conf);
                var deleteBucketRequest = new DeleteBucketRequest()
                {
                    BucketName = bucketName
                };

                var result = await
s3Client.DeleteBucketAsync(deleteBucketRequest);
                if (result.HttpStatusCode !=
System.Net.HttpStatusCode.NoContent)
                {
                    Console.WriteLine("fail to delete bucket {0},
HttpStatusCode:{1}, ErrorCode:{2}.", bucketName, (int) result.HttpStatusCode,
result.HttpStatusCode);
                    return;
                }

                Console.WriteLine("delete bucket {0} success.", bucketName);
            }
            catch (Exception e)
            {
                Console.WriteLine(e.Message);
            }
        }
    }
}
```

请求参数

DeleteBucket可设置的参数如下：

参数	类型	说明	是否必要
BucketName	string	桶的名称	是

设置桶访问权限

功能说明

PutACL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。

桶的访问权限说明：

权限类型	说明
READ	可以对桶进行list操作
READ_ACP	可以读取桶的ACL信息。桶的所有者默认具有桶的READ_ACP权限
WRITE	可以在桶中创建对象，修改原有对象数据和删除对象
WRITE_ACP	可以修改桶的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桶的任意权限。桶的所有者默认具有桶的WRITE_ACP权限
FULL_CONTROL	同时授予READ、READ_ACP、WRITE和WRITE_ACP权限

代码示例

```
using System;
using System.Threading.Tasks;
using Amazon.Runtime;
using Amazon.S3;
using Amazon.S3.Model;

namespace DotNetSDK.BucketOperation
{
    public class PutBucketACLExample
    {
        public static async Task PutBucketACL()
        {
            var accessKey = "<your-access-key>";
            var secretKey = "<your-secret-access-key>";
            var endpoint = "<your-endpoint>";
            var bucketName = "<your-bucket-name>";
            try
            {
                var credentials = new BasicAWSCredentials(accessKey, secretKey);
                var conf = new AmazonS3Config
                {
                    ServiceURL = endpoint
                };
                var s3Client = new AmazonS3Client(credentials, conf);
                var putACLRequest = new PutACLRequest()
                {
                    BucketName = bucketName,
                    // 添加一个公共读权限
                    CannedACL = S3CannedACL.PublicRead
                };

                var result = await s3Client.PutACLAsync(putACLRequest);
                if (result.HttpStatusCode != System.Net.HttpStatusCode.OK)
                {
```

```
                    Console.WriteLine("fail to put bucket ACL to bucket {0},
HttpStatusCode:{1}, ErrorCode:{2}.", bucketName, (int) result.HttpStatusCode,
result.HttpStatusCode);
                    return;
                }

                Console.WriteLine("set {0} to bucket {1}",
putACLRequest.CannedACL.Value, bucketName);
            }
            catch (Exception e)
            {
                Console.WriteLine(e.Message);
            }
        }
    }
}
```

请求参数

PutACL可设置的参数如下：

参数	类型	说明	是否必要
CannedACL	S3CannedACL	配置预定义的标准ACL信息	否
AccessControlList	S3AccessControlList	配置自定义的ACL信息	否
BucketName	string	桶的名称	是

S3CannedACL包含了一组预定义的访问控制权限，可以应用于桶的访问权限如下：

权限	说明
NoACL	默认访问权限，桶的所有者拥有FULL_CONTROL权限，其他用户没有访问权限
Private	桶的所有者拥有FULL_CONTROL权限，其他用户没有访问权限
PublicRead	桶的所有者拥有FULL_CONTROL权限，其他用户拥有READ权限
PublicReadWrite	桶的所有者拥有FULL_CONTROL权限，其他用户拥有READ和WRITE权限

获取桶访问权限

功能说明

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

代码示例

```
using System;
using System.Threading.Tasks;
using Amazon.Runtime;
using Amazon.S3;
```

```
using Amazon.S3.Model;

namespace DotNetSDK.BucketOperation
{
    public class GetBucketACLExample
    {
        public static async Task GetBucketACL()
        {
            var accessKey = "<your-access-key>";
            var secretKey = "<your-secret-access-key>";
            var endpoint = "<your-endpoint>";
            var bucketName = "<your-bucket-name>";
            try
            {
                var credentials = new BasicAWSCredentials(accessKey, secretKey);
                var conf = new AmazonS3Config
                {
                    ServiceURL = endpoint
                };
                var s3Client = new AmazonS3Client(credentials, conf);
                var getAclRequest = new GetACLRequest()
                {
                    BucketName = bucketName
                };

                var result = await s3Client.GetACLAsync(getAclRequest);
                if (result.HttpStatusCode != System.Net.HttpStatusCode.OK)
                {
                    Console.WriteLine("fail to get ACL of bucket {0},
HttpStatusCode:{1}, ErrorCode:{2}.", bucketName, (int) result.HttpStatusCode,
result.HttpStatusCode);
                    return;
                }

                foreach (var grant in result.AccessControlList.Grants)
                {
                    Console.WriteLine("Type:{0}, CanonicalUser:{1}, DisplayName:
{2}, EmailAddress:{3}, URI:{4}, Permission:{5}",
                        grant.Grantee.Type, grant.Grantee.CanonicalUser,
grant.Grantee.DisplayName, grant.Grantee.EmailAddress, grant.Grantee.URI,
grant.Permission.Value);
                }

                Console.WriteLine("OwnerId:{0}, OwnerDisplayName:{1}.",
result.AccessControlList.Owner.Id, result.AccessControlList.Owner.DisplayName);
            }
            catch (Exception e)
            {
                Console.WriteLine(e.Message);
            }
        }
    }
}
```

请求参数

GetACL可设置的参数如下：

参数	类型	说明	是否必要
BucketName	string	桶的名称	是

返回结果

GetACL返回的结果如下：

属性名	类型	说明
Grants	List<S3Grant>	Grant信息的数组，包含了每项授权和被授予人的信息
Owner	Owner	桶的所有者信息

设置桶策略

功能说明

桶策略（bukcet policy）可以灵活地配置用户各种操作和访问资源的权限。访问控制列表（access control lists，ACL）只能对单一对象设置权限，而桶策略可以基于各种条件对一个桶内的全部或者一组对象配置权限。桶的所有者拥有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。

PutBucketPolicy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字段为媒体存储支持的所有操作集合，以字符串形式表示，不区分大小写。支持通配符“*”，表示该资源能进行的所有操作。例如："Action":["s3:List*", "s3: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，指定本条statement的权限是允许还是拒绝，Effect的值必须为Allow或者Deny	必选
Resource	可选关键字，指定statement起作用的一组资源，支持通配符“*”，表示所有资源	可选，Resource与NotResource选其一
NotResource	可选关键字，指定一组资源，statement匹配除该组资源之外的其他资源。 取值同Resource	可选，NotResource与Resource选其一
Condition	可选关键字，本条statement生效的条件	可选

代码示例

```
using System;
using System.Threading.Tasks;
using Amazon.Runtime;
using Amazon.S3;
using Amazon.S3.Model;

namespace DotNetSDK.BucketOperation
{
    public class PutBucketPolicyExample
    {
        public static async Task PutBucketPolicy()
        {
```

```
            var accessKey = "<your-access-key>";
            var secretKey = "<your-secret-access-key>";
            var endpoint = "<your-endpoint>";
            var bucketName = "<your-bucket-name>";
            var policyJsonStr = "<policy-json>";
            try
            {
                var credentials = new BasicAWSCredentials(accessKey, secretKey);
                var conf = new AmazonS3Config
                {
                    ServiceURL = endpoint
                };
                var s3Client = new AmazonS3Client(credentials, conf);
                var putBucketPolicyRequest = new PutBucketPolicyRequest()
                {
                    BucketName = bucketName,
                    Policy = policyJsonStr
                };

                var result = await
s3Client.PutBucketPolicyAsync(putBucketPolicyRequest);
                if (result.HttpStatusCode != System.Net.HttpStatusCode.OK)
                {
                    Console.WriteLine("fail to put policy to bucket {0},
HttpStatusCode:{1}, ErrorCode:{2}.", bucketName, (int) result.HttpStatusCode,
result.HttpStatusCode);
                }
            }
            catch (Exception e)
            {
                Console.WriteLine(e.Message);
            }
        }
    }
}
```

请求参数

PutBucketPolicy可以设置的参数如下：

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

获取桶策略

功能说明

GetBucketPolicy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。桶的所有者可以查看桶的policy信息。

代码示例

```
using System;
using System.Threading.Tasks;
using Amazon.Runtime;
using Amazon.S3;
using Amazon.S3.Model;

namespace DotNetSDK.BucketOperation
{
    public class GetBucketPolicyExample
    {
        public static async Task GetBucketPolicy()
        {
            var accessKey = "<your-access-key>";
            var secretKey = "<your-secret-access-key>";
            var endpoint = "<your-endpoint>";
            var bucketName = "<your-bucket-name>";
            try
            {
                var credentials = new BasicAWSCredentials(accessKey, secretKey);
                var conf = new AmazonS3Config
                {
                    ServiceURL = endpoint
                };
                var s3Client = new AmazonS3Client(credentials, conf);
                var getBucketPolicyRequest = new GetBucketPolicyRequest()
                {
                    BucketName = bucketName
                };

                var result = await
s3Client.GetBucketPolicyAsync(getBucketPolicyRequest);
                if (result.HttpStatusCode != System.Net.HttpStatusCode.OK)
                {
                    Console.WriteLine("fail to get policy of bucket {0},
HttpStatusCode:{1}, ErrorCode:{2}.", bucketName, (int) result.HttpStatusCode,
result.HttpStatusCode);
                    return;
                }

                Console.WriteLine("the policy of bucket {0} is: {1}.",
bucketName, result.Policy);
            }
            catch (Exception e)
            {
                Console.WriteLine(e.Message);
            }
        }
    }
}
```

请求参数

GetBucketPolicy可设置的参数如下：

参数	类型	说明	是否必要
BucketName	string	bucekt的名称	是

返回结果

GetBucketPolicy返回的结果如下：

属性名	类型	说明
Policy	string	JSON格式的bucekt策略信息

删除桶策略

功能说明

DeleteBucketPolicy操作可以删除桶已经配置的策略，桶的所有者默认拥有删除桶策略的权限。

代码示例

```
using System;
using System.Threading.Tasks;
using Amazon.Runtime;
using Amazon.S3;
using Amazon.S3.Model;

namespace DotNetSDK.BucketOperation
{
    public class DeleteBucketPolicyExample
    {
        public static async Task DeleteBucketPolicy()
        {
            var accessKey = "<your-access-key>";
            var secretKey = "<your-secret-access-key>";
            var endpoint = "<your-endpoint>";
            var bucketName = "<your-bucket-name>";
            try
            {
                var credentials = new BasicAWSCredentials(accessKey, secretKey);
                var conf = new AmazonS3Config
                {
                    ServiceURL = endpoint
                };
                var s3Client = new AmazonS3Client(credentials, conf);
                var deleteBucketPolicyRequest = new DeleteBucketPolicyRequest()
                {
                    BucketName = bucketName
                };

                var result = await
s3Client.DeleteBucketPolicyAsync(deleteBucketPolicyRequest);
                if (result.HttpStatusCode != System.Net.HttpStatusCode.OK)
                {
```

```
                    Console.WriteLine("fail to delete policy of bucket {0},
HttpStatusCode:{1}, ErrorCode:{2}.", bucketName, (int) result.HttpStatusCode,
result.HttpStatusCode);
                }
            }
            catch (Exception e)
            {
                Console.WriteLine(e.Message);
            }
        }
    }
}
```

请求参数

DeleteBucketPolicy可以设置的参数如下：

参数	类型	说明	是否必要
BucketName	string	bucekt的名称	是

设置桶生命周期配置

功能说明

PutLifecycleConfiguration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、标签匹配等方式获取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PutBucketLifecycleConfiguration操作会覆盖桶中已存在的生命周期规则。

代码示例

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Amazon.Runtime;
using Amazon.S3;
using Amazon.S3.Model;

namespace DotNetSDK.BucketOperation
{
    public class PutBucketLifeCycleConfigurationExample
    {
        public static async Task PutBucketLifecycleConfiguration()
        {
            var accessKey = "<your-access-key>";
            var secretKey = "<your-secret-access-key>";
            var endpoint = "<your-endpoint>";
            var bucketName = "<your-bucket-name>";
            try
            {
                var credentials = new BasicAWSCredentials(accessKey, secretKey);
                var conf = new AmazonS3Config
                {
                    ServiceURL = endpoint
```

```
                };
                var s3Client = new AmazonS3Client(credentials, conf);

                List<LifecycleRule> rules = new List<LifecycleRule>();
                // rule1:设置符合指定前缀的对象一天后过期
                var rule1 = new LifecycleRule()
                {
                    Id = "expireAfterOneDay",
                    Filter = new LifecycleFilter()
                    {
                        LifecycleFilterPredicate = new
LifecyclePrefixPredicate()
                        {
                            Prefix = "expireAfterOneDay/"
                        }
                    },
                    Status = LifecycleRuleStatus.Enabled,
                    Expiration = new LifecycleRuleExpiration()
                    {
                        Days = 1
                    }
                };
                rules.Add(rule1);
                // rule2: 设置符合指定前缀的对象的历史版本一天后过期
                var rule2 = new LifecycleRule()
                {
                    Id = "noncurrentVersionExpireAfterOneDay",
                    Status = LifecycleRuleStatus.Enabled,
                    Filter = new LifecycleFilter()
                    {
                        LifecycleFilterPredicate = new
LifecyclePrefixPredicate()
                        {
                            Prefix = "noncurrentVersionExpireAfterOneDay/"
                        }
                    },
                    NoncurrentVersionExpiration = new
LifecycleRuleNoncurrentVersionExpiration()
                    {
                        NoncurrentDays = 1
                    }
                };
                rules.Add(rule2);
                // rule3: 设置匹配指定标签信息的对象一天后过期
                var rule3 = new LifecycleRule()
                {
                    Id = "withTagsExpireAfterOneDay",
                    Status = LifecycleRuleStatus.Enabled,
                    Expiration = new LifecycleRuleExpiration()
                    {
                        Days = 1
                    },
                    Filter = new LifecycleFilter()
                    {
                        LifecycleFilterPredicate = new LifecycleTagPredicate()
                        {
                            Tag = new Tag()
                            {
```

```
                                Key = "<key1>",
                                Value = "<value1>"
                            }
                        }
                    },
                };
                rules.Add(rule3);
                LifecycleConfiguration configuration = new
LifecycleConfiguration()
                {
                    Rules = rules
                };
                var putLifecycleConfigurationRequest = new
PutLifecycleConfigurationRequest()
                {
                    BucketName = bucketName,
                    Configuration = configuration
                };

                var result = await
s3Client.PutLifecycleConfigurationAsync(putLifecycleConfigurationRequest);
                if (result.HttpStatusCode != System.Net.HttpStatusCode.OK)
                {
                    Console.WriteLine("fail to put lifecycle configuration to
bucket {0}, HttpStatusCode:{1}, ErrorCode:{2}.", bucketName, (int)
result.HttpStatusCode, result.HttpStatusCode);
                }
            }
            catch (Exception e)
            {
                Console.WriteLine(e.Message);
            }
        }
    }
}
```

请求参数

PutLifecycleConfiguration可设置的参数如下：

参数	类型	说明	是否必要
BucketName	string	桶的名称	是
Configuration	LifecycleConfiguration	封装了生命周期规则的数组，最多可以包含1000条规则	是

获取桶生命周期配置

功能说明

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

代码示例

```
using System;
using System.Threading.Tasks;
using Amazon.Runtime;
using Amazon.S3;
using Amazon.S3.Model;

namespace DotNetSDK.BucketOperation
{
    public class GetBucketLifecycleConfigurationExample
    {
        public static async Task GetBucketLifecycleConfiguration()
        {
            var accessKey = "<your-access-key>";
            var secretKey = "<your-secret-access-key>";
            var endpoint = "<your-endpoint>";
            var bucketName = "<your-bucket-name>";
            try
            {
                var credentials = new BasicAWSCredentials(accessKey, secretKey);
                var conf = new AmazonS3Config
                {
                    ServiceURL = endpoint
                };
                var s3Client = new AmazonS3Client(credentials, conf);
                var getLifecycleConfigurationRequest = new
GetLifecycleConfigurationRequest()
                {
                    BucketName = bucketName
                };

                var result = await
s3Client.GetLifecycleConfigurationAsync(getLifecycleConfigurationRequest);
                if (result.HttpStatusCode != System.Net.HttpStatusCode.OK)
                {
                    Console.WriteLine("fail to get lifecycle configuration of
bucket {0}, HttpStatusCode:{1}, ErrorCode:{2}.", bucketName, (int)
result.HttpStatusCode, result.HttpStatusCode);
                    return;
                }

                Console.WriteLine("the lifecycle configuration of bucket {0}
are:", bucketName);
                foreach (var lifecycleRule in result.Configuration.Rules)
                {
                    Console.WriteLine("Lifecycle rule id: {0}",
lifecycleRule.Id);
                    Console.WriteLine("Lifecycle rule status: {0}",
lifecycleRule.Status);
                    if (null != lifecycleRule.Expiration)
                    {
                        Console.WriteLine("expiration days: {0}",
lifecycleRule.Expiration.Days);
                    }

                    if (null != lifecycleRule.NoncurrentVersionExpiration)
```

```
                    {
                        Console.WriteLine("NoncurrentVersionExpiration
NoncurrentDays: {0}", lifecycleRule.NoncurrentVersionExpiration.NoncurrentDays);
                    }

                    if (null != lifecycleRule.Transitions)
                    {
                        foreach (var transition in lifecycleRule.Transitions)
                        {
                            Console.WriteLine("Transition Days: {0}",
transition.Days.ToString());
                        }
                    }

                    if (null != lifecycleRule.NoncurrentVersionTransitions)
                    {
                        foreach (var nontransition in
lifecycleRule.NoncurrentVersionTransitions)
                        {
                            Console.WriteLine("NoncurrentVersionTransition
NoncurrentDays: {0}", nontransition.NoncurrentDays.ToString());
                        }
                    }
                }
            }
            catch (Exception e)
            {
                Console.WriteLine(e.Message);
            }
        }
    }
}
```

请求参数

GetBucketLifecycleConfiguration可设置的参数如下：

参数	类型	说明	是否必要
BucketName	string	桶的名称	是

返回结果

GetBucketLifecycleConfiguration返回的结果如下：

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

删除桶生命周期配置

功能说明

DeleteLifecycleConfiguration操作可以删除桶中的全部生命周期规则。

代码示例

```
using System;
using System.Threading.Tasks;
using Amazon.Runtime;
using Amazon.S3;
using Amazon.S3.Model;

namespace DotNetSDK.BucketOperation
{
    public class DeleteBucketLifecycleConfigurationExample
    {
        public static async Task DeleteBucketLifeConfiguration()
        {
            var accessKey = "<your-access-key>";
            var secretKey = "<your-secret-access-key>";
            var endpoint = "<your-endpoint>";
            var bucketName = "<your-bucket-name>";
            try
            {
                var credentials = new BasicAWSCredentials(accessKey, secretKey);
                var conf = new AmazonS3Config
                {
                    ServiceURL = endpoint
                };
                var s3Client = new AmazonS3Client(credentials, conf);
                var deleteLifecycleConfigurationRequest = new
DeleteLifecycleConfigurationRequest()
                {
                    BucketName = bucketName
                };
                var result = await
s3Client.DeleteLifecycleConfigurationAsync(deleteLifecycleConfigurationRequest);
                if (result.HttpStatusCode !=
System.Net.HttpStatusCode.NoContent)
                {
                    Console.WriteLine("fail to delete lifecycle configuration of
bucket {0}, HttpStatusCode:{1}, ErrorCode:{2}.", bucketName, (int)
result.HttpStatusCode, result.HttpStatusCode);
                }
            }
            catch (Exception e)
            {
                Console.WriteLine(e.Message);
            }
        }
    }
}
```

请求参数

DeleteLifecycleConfiguration可设置的参数如下：

参数	类型	说明	是否必要
BucketName	string	桶的名称	是

设置桶跨域访问配置

功能说明

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

您可以通过PutCORSConfiguration接口设置桶的跨域访问配置。

代码示例

```
using System;
using System.Threading.Tasks;
using Amazon.Runtime;
using Amazon.S3;
using Amazon.S3.Model;

namespace DotNetSDK.BucketOperation
{
    public class PutBucketCORSExample
    {
        public static async Task PutCORSConfiguration()
        {
            var accessKey = "<your-access-key>";
            var secretKey = "<your-secret-access-key>";
            var endpoint = "<your-endpoint>";
            var bucketName = "<your-bucket-name>";
            try
            {
                var credentials = new BasicAWSCredentials(accessKey, secretKey);
                var conf = new AmazonS3Config
                {
                    ServiceURL = endpoint
                };
                var s3Client = new AmazonS3Client(credentials, conf);
                var rule = new CORSRule();
                rule.AllowedMethods.Add("PUT");
                rule.AllowedMethods.Add("GET");
                rule.AllowedMethods.Add("HEAD");
                rule.AllowedMethods.Add("POST");
                rule.AllowedMethods.Add("DELETE");
                rule.AllowedHeaders.Add("*");
                rule.AllowedOrigins.Add("*"); // 可以使用http://domain:port
                rule.ExposeHeaders.Add("ETag");
                rule.MaxAgeSeconds = 3600;

                var req = new PutCORSConfigurationRequest()
                {
                    BucketName = bucketName,
                    Configuration = new CORSConfiguration()
                };
```

```
                req.Configuration.Rules.Add(rule);

                var result = await s3Client.PutCORSConfigurationAsync(req);
                if (result.HttpStatusCode != System.Net.HttpStatusCode.OK)
                {
                    Console.WriteLine("fail to put bucket cors {0},
HttpStatusCode:{1}, ErrorCode:{2}.", bucketName, (int) result.HttpStatusCode,
result.HttpStatusCode);
                }
            }
            catch (Exception e)
            {
                Console.WriteLine(e.Message);
            }
        }
    }
}
```

请求参数

参数	类型	说明	是否必要
BucketName	string	桶名称	是
Configuration	CORSConfiguration	跨域访问规则	是

关于CORSConfiguration一些说明

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

获取桶跨域访问配置

功能说明

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

您可以通过GetCORSConfiguration接口设置桶的跨域访问配置。

代码示例

```
using System;
using System.Threading.Tasks;
using Amazon.Runtime;
```

```
using Amazon.S3;

namespace DotNetSDK.BucketOperation
{
    public class GetBucketCORSExample
    {
        public static async Task GetCORSConfiguration()
        {
            var accessKey = "<your-access-key>";
            var secretKey = "<your-secret-access-key>";
            var endpoint = "<your-endpoint>";
            var bucketName = "<your-bucket-name>";
            try
            {
                var credentials = new BasicAWSCredentials(accessKey, secretKey);
                var conf = new AmazonS3Config
                {
                    ServiceURL = endpoint
                };
                var s3Client = new AmazonS3Client(credentials, conf);

                var result = await
s3Client.GetCORSConfigurationAsync(bucketName);
                if (result.HttpStatusCode != System.Net.HttpStatusCode.OK)
                {
                    Console.WriteLine("fail to get bucket cors {0},
HttpStatusCode:{1}, ErrorCode:{2}.", bucketName, (int) result.HttpStatusCode,
result.HttpStatusCode);
                    return;
                }

                foreach (var corsRule in result.Configuration.Rules)
                {
                    Console.WriteLine("cors rule's methods: {0}", string.Join(",
", corsRule.AllowedMethods));
                    Console.WriteLine("cors rule's headers: {0}", string.Join(",
", corsRule.AllowedHeaders));
                    Console.WriteLine("cors rule's origins: {0}", string.Join(",
", corsRule.AllowedOrigins));
                    Console.WriteLine("cors rule's expose headers: {0}",
string.Join(", ", corsRule.ExposeHeaders));
                    Console.WriteLine("cors rule's expired seconds: {0}",
corsRule.MaxAgeSeconds);
                }
            }
            catch (Exception e)
            {
                Console.WriteLine(e.Message);
            }
        }
    }
}
```

请求参数

参数	类型	说明	是否必要
bucketName	string	桶名称	是

返回结果

参数	类型	说明
Configuration	CORSConfiguration	跨域访问规则，包含规则id，请求方法，请求源等信息

关于CORSConfiguration一些说明

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

删除桶跨域访问配置

功能说明

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

您可以通过DeleteCORSConfiguration接口删除桶跨域访问配置。

代码示例

```
using System;
using System.Threading.Tasks;
using Amazon.Runtime;
using Amazon.S3;

namespace DotNetSDK.BucketOperation
{
    public class DeleteBucketCORSExample
    {
        public static async Task DeleteCORSConfiguration()
        {
            var accessKey = "<your-access-key>";
            var secretKey = "<your-secret-access-key>";
            var endpoint = "<your-endpoint>";
            var bucketName = "<your-bucket-name>";
            try
            {
                var credentials = new BasicAWSCredentials(accessKey, secretKey);
```

```
                var conf = new AmazonS3Config
                {
                    ServiceURL = endpoint
                };
                var s3Client = new AmazonS3Client(credentials, conf);

                var result = await
s3Client.DeleteCORSConfigurationAsync(bucketName);
                if (result.HttpStatusCode !=
System.Net.HttpStatusCode.NoContent)
                {
                    Console.WriteLine("fail to get bucket cors {0},
HttpStatusCode:{1}, ErrorCode:{2}.", bucketName,
                        (int)result.HttpStatusCode, result.HttpStatusCode);
                }
            }
            catch (Exception e)
            {
                Console.WriteLine(e.Message);
            }
        }
    }
}
```

请求参数

参数	类型	说明	是否必要
Bucket	string	桶名称	是

设置桶版本控制状态

请求参数

PutBucketVersioning操作可以设置桶版本控制状态。桶的版本控制状态可以设置为以下的值：

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

代码示例

```
using System;
using System.Threading.Tasks;
using Amazon.Runtime;
using Amazon.S3;
using Amazon.S3.Model;

namespace DotNetSDK.BucketOperation
{
    public class PutBucketVersioningExample
    {
        public static async Task PutBucketVersioning()
        {
            var accessKey = "<your-access-key>";
```

```
            var secretKey = "<your-secret-access-key>";
            var endpoint = "<your-endpoint>";
            var bucketName = "<your-bucket-name>";
            try
            {
                var credentials = new BasicAWSCredentials(accessKey, secretKey);
                var conf = new AmazonS3Config
                {
                    ServiceURL = endpoint
                };
                var s3Client = new AmazonS3Client(credentials, conf);
                var putBucketVersioningRequest = new
PutBucketVersioningRequest()
                {
                    BucketName = bucketName,
                    VersioningConfig = new S3BucketVersioningConfig()
                    {
                        Status = VersionStatus.Enabled
                    }
                };

                var result = await
s3Client.PutBucketVersioningAsync(putBucketVersioningRequest);
                if (result.HttpStatusCode != System.Net.HttpStatusCode.OK)
                {
                    Console.WriteLine("fail to put versioning config to bucket
{0}, HttpStatusCode:{1}, ErrorCode:{2}.", bucketName, (int)
result.HttpStatusCode, result.HttpStatusCode);
                }
            }
            catch (Exception e)
            {
                Console.WriteLine(e.Message);
            }
        }
    }
}
```

请求参数

PutBucketVersioning可设置的参数如下：

参数	类型	说明	是否必要
BucketName	string	桶的名称	是
VersioningConfig	S3BucketVersioningConfig	封装了设置版本控制状态的参数	是

获取桶版本控制状态

功能说明

GetBucketVersioning操作可以获取桶的版本控制状态信息。桶的所有者默认拥有获取到桶的版本控制信息的权限。

每个桶的版本控制有三个状态：未开启（Off）、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。

代码示例

```
using System;
using System.Threading.Tasks;
using Amazon.Runtime;
using Amazon.S3;
using Amazon.S3.Model;

namespace DotNetSDK.BucketOperation
{
    public class GetBucketVersioningExample
    {
        public static async Task GetBucketVersioning()
        {
            var accessKey = "<your-access-key>";
            var secretKey = "<your-secret-access-key>";
            var endpoint = "<your-endpoint>";
            var bucketName = "<your-bucket-name>";
            try
            {
                var credentials = new BasicAWSCredentials(accessKey, secretKey);
                var conf = new AmazonS3Config
                {
                    ServiceURL = endpoint
                };
                var s3Client = new AmazonS3Client(credentials, conf);
                var getBucketVersioningRequest = new
GetBucketVersioningRequest()
                {
                    BucketName = bucketName
                };

                var result = await
s3Client.GetBucketVersioningAsync(getBucketVersioningRequest);
                if (result.HttpStatusCode != System.Net.HttpStatusCode.OK)
                {
                    Console.WriteLine("fail to get versioning config of bucket
{0}, HttpStatusCode: {1}, ErrorCode:{2}.", bucketName, (int)
result.HttpStatusCode, result.HttpStatusCode);
                    return;
                }

                Console.WriteLine("the versioning config of bucket {0} is:
{1}.", bucketName,
                    result.VersioningConfig.Status.Value);
            }
            catch (Exception e)
            {
                Console.WriteLine(e.Message);
            }
        }
```

```
    }
}
```

请求参数

GetBucketVersioning可设置的参数如下：

参数	类型	说明	是否必要
BucketName	string	桶的名称	是

返回结果

GetBucketVersioning返回的结果如下：

属性名	类型	说明
VersioningConfig	S3BucketVersioningConfig	封装桶的版本控制状态信息的类，其中的Status属性描述了桶的版本控制设置状态

对象相关接口

获取对象列表

功能说明

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

代码示例

```
using System;
using System.Threading.Tasks;
using Amazon.Runtime;
using Amazon.S3;
using Amazon.S3.Model;

namespace DotNetSDK.ObjectOperation
{
    public class ListObjectsExample
    {
        public static async Task ListObjects()
        {
            var accessKey = "<your-access-key>";
            var secretKey = "<your-secret-access-key>";
            var endpoint = "<your-endpoint>";
            var bucketName = "<your-bucket-name>";
            var credentials = new BasicAWSCredentials(accessKey, secretKey);
            try
            {
                var conf = new AmazonS3Config
                {
                    ServiceURL = endpoint
```

```
                };
                var s3Client = new AmazonS3Client(credentials, conf);
                var listObjectsRequest = new ListObjectsRequest()
                {
                    BucketName = bucketName
                };

                var result = await
s3Client.ListObjectsAsync(listObjectsRequest);
                if (result.HttpStatusCode != System.Net.HttpStatusCode.OK)
                {
                    Console.WriteLine("fail to list objects in bucket {0},
HttpStatusCode:{1}, ErrorCode:{2}.", bucketName, (int) result.HttpStatusCode,
result.HttpStatusCode);
                    return;
                }

                Console.WriteLine("the objects in bucket {0} are: ",
bucketName);
                foreach (var obj in result.S3Objects)
                {
                    Console.WriteLine(obj.Key);
                }
            }
            catch (Exception e)
            {
                Console.WriteLine(e.Message);
            }
        }
    }
}
```

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

```
using System;
using System.Threading.Tasks;
using Amazon.Runtime;
using Amazon.S3;
using Amazon.S3.Model;

namespace DotNetSDK.ObjectOperation
{
    public class ListObjectsExample
    {
        public static async Task ListObjects2()
        {
            var accessKey = "<your-access-key>";
            var secretKey = "<your-secret-access-key>";
            var endpoint = "<your-endpoint>";
            var bucketName = "<your-bucket-name>";
            var credentials = new BasicAWSCredentials(accessKey, secretKey);
            try
            {
                var conf = new AmazonS3Config
                {
                    ServiceURL = endpoint
```

```
                };
                var s3Client = new AmazonS3Client(credentials, conf);

                string marker = string.Empty;
                bool isTruncated;

                do
                {
                    var listObjectsRequest = new ListObjectsRequest()
                    {
                        BucketName = bucketName,
                        Marker = marker
                    };

                    var result = await
s3Client.ListObjectsAsync(listObjectsRequest);

                    if (result.HttpStatusCode != System.Net.HttpStatusCode.OK)
                    {
                        Console.WriteLine("fail to list objects in bucket {0},
HttpStatusCode:{1}, ErrorCode:{2}.",
                            bucketName, (int)result.HttpStatusCode,
result.HttpStatusCode);
                        return;
                    }

                    foreach (var obj in result.S3Objects)
                    {
                        Console.WriteLine(obj.Key);
                    }

                    isTruncated = result.IsTruncated;
                    marker = result.NextMarker ?? string.Empty;

                } while (isTruncated && !string.IsNullOrEmpty(marker));

            }
            catch (Exception e)
            {
                Console.WriteLine(e.Message);
            }
        }
    }
}
```

请求参数

ListObjects可以设置的参数如下：

参数	类型	说明	是否必要
BucketName	string	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	int	设置response中返回对象的数量，默认值和最大值均为1000	否
Prefix	string	限定返回对象的key必须以Prefix作为前缀	否

返回结果

ListObjects返回的结果如下：

属性名	类型	说明
CommonPrefixes	List<string>	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key作为一组
Contents	List<S3Object>	对象数据，每个对象包含了Etag、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
MaxKeys	int	本次返回结果包含的对象数量上限
Name	string	桶的名字
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

上传对象

功能说明

PutObject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。

代码示例

```
using System;
using System.Threading.Tasks;
using Amazon.Runtime;
using Amazon.S3;
using Amazon.S3.Model;

namespace DotNetSDK.ObjectOperation
{
    public class PutObjectExample
    {
        public static async Task PutObject()
        {
            var accessKey = "<your-access-key>";
            var secretKey = "<your-secret-access-key>";
            var endpoint = "<your-endpoint>";
            var bucketName = "<your-bucket-name>";
            var key = "<your-object-key>";
            var filePath = "<file-path>";
            try
            {
                var credentials = new BasicAWSCredentials(accessKey, secretKey);
                var conf = new AmazonS3Config
                {
                    ServiceURL = endpoint
                };
                var s3Client = new AmazonS3Client(credentials, conf);
                var putObjectRequest = new PutObjectRequest()
                {
                    BucketName = bucketName,
                    Key = key,
                    FilePath = filePath
                };

                var result = await s3Client.PutObjectAsync(putObjectRequest);
                if (result.HttpStatusCode != System.Net.HttpStatusCode.OK)
                {
                    Console.WriteLine("fail to put object {0}, HttpStatusCode:
{1}, ErrorCode:{2}.", key, (int) result.HttpStatusCode, result.HttpStatusCode);
                    return;
                }

                Console.WriteLine("put obejct {0}, ETag: {1}, versionId:{2}",
key, result.ETag, result.VersionId);
            }
            catch (Exception e)
```

```
            {
                Console.WriteLine(e.Message);
            }
        }
    }
}
```

请求参数

PutObject可设置的参数如下：

参数	类型	说明	是否必要
CannedACL	S3CannedACL	配置上传对象的预定义的标准ACL信息	否
ContentBody	string	上传对象的数据内容，与FilePath、InputStream任选其一作为上传对象数据的来源	否
ContentType	string	描述上传文件格式的标准MIME类型	否
FilePath	string	上传文件的本地路径，与ContentBody、InputStream任选其一作为上传对象数据的来源	否
InputStream	Stream	对象的数据，与ContentBody、FilePath任选其一作为上传对象数据的来源	否
Bucket	string	桶的名称	是
MD5Digest	string	上传对象数据的base64编码的128位MD5值，不包含请求头部的信息	否
Key	string	上传文件到媒体存储服务后对应的key	是
Tagset	List<Tag>	对象的标签信息	否
WebsiteRedirectLocation	string	如果桶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桶下的其他对象或者外部的URL	否

返回结果

PutObject返回的结果如下：

参数	类型	说明
ETag	string	上传对象后的对应的Entity Tag
VersionId	string	上传对象后相应的版本id

下载对象

功能说明

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

代码示例

```
using System;
using System.IO;
using System.Threading;
using System.Threading.Tasks;
using Amazon.Runtime;
using Amazon.S3;
using Amazon.S3.Model;

namespace DotNetSDK.ObjectOperation
{
    public class GetObjectExample
    {
        public static async Task GetObject()
        {
            var accessKey = "<your-access-key>";
            var secretKey = "<your-secret-access-key>";
            var endpoint = "<your-endpoint>";
            var bucketName = "<your-bucket-name>";
            var key = "<your-object-key>";
            var filePath = "<file-path>";
            try
            {
                var credentials = new BasicAWSCredentials(accessKey, secretKey);
                var conf = new AmazonS3Config
                {
                    ServiceURL = endpoint
                };
                var s3Client = new AmazonS3Client(credentials, conf);
                var getObjectRequest = new GetObjectRequest()
                {
                    BucketName = bucketName,
                    Key = key
                };

                using (var result = await
s3Client.GetObjectAsync(getObjectRequest))
                using (Stream stream = result.ResponseStream)
                {
                    var fileStream = File.Create(filePath);
                    await stream.CopyToAsync(fileStream);
                    fileStream.Close();
                }
            }
            catch (Exception e)
            {
                Console.WriteLine(e.Message);
            }
```

```
        }
    }
}
```

请求参数

GetObject可设置的参数如下：

参数	类型	说明	是否必要
BucketName	string	桶的名称	是
EtagToMatch	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
ModifiedSinceDateUtc	DateTime	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
EtagToNotMatch	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
UnmodifiedSinceDateUtc	DateTime	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	int	读取对象指定的分片，该参数大于等于1，小于等于10000	否
ByteRange	ByteRange	下载对象指定范围内的数据（单位：字节）	否
VersionId	string	当桶开启版本控制的时候，用于指定获取指定版本的对象数据，当不指定该参数的时候，默认获取最新版本的对象数据	否

返回结果

GetObject返回的结果如下：

参数	类型	说明
BucketName	string	对象所在桶的名称
ContentLength	long	对象数据的长度，单位为字节
ETag	string	对象的Entity Tag
LastModified	DateTime	最后修改对象的时间
TagCount	int	对象标签的数量
VersionId	string	对象的version id

复制对象

功能说明

CopyObject操作用于根据一个已存在的对象创建新的对象。使用CopyOoject可以复制单个最大为5GB的对象，如果需要复制更大的对象，可以使用UploadPartCopy操作。执行CopyObject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，也可以才CopyObject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CopyObject操作的用户私有的。

CopyObject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CopySource中加入version id来拷贝指定的Object版本，如果原对象的version id为删除标记，则不会被拷贝。

代码示例

```
using System;
using System.Threading.Tasks;
using Amazon.Runtime;
using Amazon.S3;
using Amazon.S3.Model;

namespace DotNetSDK.ObjectOperation
{
    public class CopyObjectExample
    {
        public static async Task CopyObject()
        {
            var accessKey = "<your-access-key>";
            var secretKey = "<your-secret-access-key>";
            var endpoint = "<your-endpoint>";
            var sourceBucket = "<source-bucket>";
            var sourceKey = "<source-key>";
            var destinationBucket = "<destination-bucket>";
            var destinationKey = "<destination-key>";
            try
            {
                var credentials = new BasicAWSCredentials(accessKey, secretKey);
                var conf = new AmazonS3Config
                {
                    ServiceURL = endpoint
                };
                var s3Client = new AmazonS3Client(credentials, conf);
                var copyObjectRequest = new CopyObjectRequest()
                {
                    SourceBucket = sourceBucket,
                    SourceKey = sourceKey,
                    DestinationBucket = destinationBucket,
                    DestinationKey = destinationKey
                };
                var result = await s3Client.CopyObjectAsync(copyObjectRequest);
                if (result.HttpStatusCode != System.Net.HttpStatusCode.OK)
                {
                    Console.WriteLine("fail to copy object, HttpStatusCode:{0},
ErrorCode:{1}.", (int) result.HttpStatusCode, result.HttpStatusCode);
```

```
                    return;
                }

                Console.WriteLine("copy object from {0}/{1} to {2}/{3}.",
sourceBucket, sourceKey, destinationBucket, destinationKey);
            }
            catch (Exception e)
            {
                Console.WriteLine(e.Message);
            }
        }
    }
}
```

请求参数

CopyObject可设置的参数如下：

参数	类型	说明	是否必要
CannedACL	S3CannedACL	拷贝后对象的预定义的标准ACL信息	否
DestinationBucket	string	放置拷贝生成对象的桶名称	是
DestinationKey	string	拷贝生成对象的key	是
SourceBucket	string	放置被拷贝对象的桶的名称	是
SourceKey	string	被拷贝对象的key	是
Metadata	MetadataCollection	拷贝生成对象的元数据信息	否

参数	类型	说明	是否必要
MetadataDirective	S3MetadataDirective	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
TagSet	List<Tag>	拷贝生成对象的标签信息	否
SourceVersionId	string	指定被拷贝对象的版本信息，如果不指定，默认拷贝对象的当前版本	否
WebsiteRedirectLocation	string	如果桶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桶下的其他对象或者外部的URL	否

返回结果

CopyObject返回的结果如下：

参数	类型	说明
ETag	string	拷贝生成对象的ETag
LastModified	string	拷贝生成对象的最新修改时间

删除对象

功能说明

DeleteObject操作用于删除一个对象。当桶未开启多版本时会直接删除对象。对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DeleteObject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。当执行GetObject操作时，会检测到当前版本为删除标记，并返回404 NotFound。

代码示例

```
using System;
using System.Threading.Tasks;
using Amazon.Runtime;
using Amazon.S3;
using Amazon.S3.Model;

namespace DotNetSDK.ObjectOperation
{
    public class DeleteObjectExample
    {
        public static async Task DeleteObject()
```

```
        {
            var accessKey = "<your-access-key>";
            var secretKey = "<your-secret-access-key>";
            var endpoint = "<your-endpoint>";
            var bucketName = "<your-bucket-name>";
            var key = "<your-object-key>";
            var credentials = new BasicAWSCredentials(accessKey, secretKey);
            try
            {
                var conf = new AmazonS3Config
                {
                    ServiceURL = endpoint
                };
                var s3Client = new AmazonS3Client(credentials, conf);

                var deleteObjectRequest = new DeleteObjectRequest()
                {
                    BucketName = bucketName,
                    Key = key
                };
                var result = await
s3Client.DeleteObjectAsync(deleteObjectRequest);
                if (result.HttpStatusCode !=
System.Net.HttpStatusCode.NoContent)
                {
                    Console.WriteLine("fail to delete object {0},
HttpStatusCode:{1}, ErrorCode:{2}.", key, (int) result.HttpStatusCode,
result.HttpStatusCode);
                }

                Console.WriteLine("deleted object {0}.", key);
            }
            catch (Exception e)
            {
                Console.WriteLine(e.Message);
            }
        }
    }
}
```

请求参数

DeleteObject可设置的参数如下：

参数	类型	说明	是否必要
BucketName	string	桶的名称	是
Key	string	对象的key	是
VersionId	string	用于指定要删除对象的版本，即version id	否

返回结果

DeleteObject返回的结果如下：

参数	类型	说明
DeleteMarker	string	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，会创建删除标记，此时DeleteMarker为true。如果指定对象的版本来永久删除指定对象版本时，若该版本是删除标记，则DeleteMarker也为true。其他情况下DeleteMarker的值为空
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。 如果指定版本来永久删除对象指定版本时，返回的VersionId为对象的版本

批量删除对象

功能说明

（本接口目前仅支持部分资源池使用，如需使用，请联系天翼云客服确认。）

DeleteObjects操作可以实现通过一个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个对象的删除请求，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，会当作已删除处理。

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

代码示例

```
using System;
using System.Threading.Tasks;
using Amazon.Runtime;
using Amazon.S3;
using Amazon.S3.Model;

namespace DotNetSDK.ObjectOperation
{
    public class DeleteObjectsExample
    {
        public static async Task DeleteObjects()
        {
            var accessKey = "<your-access-key>";
            var secretKey = "<your-secret-access-key>";
            var endpoint = "<your-endpoint>";
            var bucketName = "<your-bucket-name>";
            try
            {
                var credentials = new BasicAWSCredentials(accessKey, secretKey);
                var conf = new AmazonS3Config
                {
                    ServiceURL = endpoint
                };
                var s3Client = new AmazonS3Client(credentials, conf);
                DeleteObjectsRequest deleteObjectsRequest = new
DeleteObjectsRequest()
```

```
                {
                    BucketName = bucketName,
                    Quiet = false
                };
                deleteObjectsRequest.AddKey("<object-key1>");
                deleteObjectsRequest.AddKey("<object-key2>");

                var result = await
s3Client.DeleteObjectsAsync(deleteObjectsRequest);
                Console.WriteLine("HttpStatusCode:{0}, {1}", (int)
result.HttpStatusCode, result.HttpStatusCode);
                //打印删除成功的对象信息
                foreach (var deletedObject in result.DeletedObjects)
                {
                    Console.WriteLine("Key:{0}, DeleteMarker:{1}, VersionId:{2},
DeleteMarkerVersionId:{3}.", deletedObject.Key, deletedObject.DeleteMarker,
deletedObject.VersionId,
                        deletedObject.DeleteMarkerVersionId);
                }

                //打印删除失败的对象信息
                foreach (var deleteError in result.DeleteErrors)
                {
                    Console.WriteLine("Code:{0}, Key:{1}, Message:{2},
VersionId:{3}", deleteError.Code, deleteError.Key, deleteError.Message,
deleteError.VersionId);
                }
            }
            catch (Exception e)
            {
                Console.WriteLine(e.Message);
            }
        }
    }
}
```

请求参数

DeleteObjects可设置的参数如下：

参数	类型	说明	是否必要
BucketName	string	桶的名称	是
Quiet	bool	设置为ture表示采用返回结果为quiet response模式，设置为false表示返回结果为verbose response模式	否

返回结果

DeleteObjects返回的结果如下：

参数	类型	说明
Deleted	List<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、对象名称和版本等信息
Errors	List<DeleteError>	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、对象名称和版本等信息

获取对象元数据

功能说明

GetObjectMetadata操作用于获取对象的元数据信息。执行HeadObject操作需要具有对该对象的READ权限。GetObjectMetadata操作可以用于判断对象是否存在。

代码示例

```
using System;
using System.Threading.Tasks;
using Amazon.Runtime;
using Amazon.S3;
using Amazon.S3.Model;

namespace DotNetSDK.ObjectOperation
{
    public class GetObjectMetadataExample
    {
        public static async Task GetObjectMetadata()
        {
            var accessKey = "<your-access-key>";
            var secretKey = "<your-secret-access-key>";
            var endpoint = "<your-endpoint>";
            var bucketName = "<your-bucket-name>";
            var key = "<your-object-key>";
            try
            {
                var credentials = new BasicAWSCredentials(accessKey, secretKey);
                var conf = new AmazonS3Config
                {
                    ServiceURL = endpoint
                };
                var s3Client = new AmazonS3Client(credentials, conf);
                var getObjectMetadataRequest = new GetObjectMetadataRequest()
                {
                    BucketName = bucketName,
                    Key = key
                };

                var result = await
s3Client.GetObjectMetadataAsync(getObjectMetadataRequest);
                if (result.HttpStatusCode != System.Net.HttpStatusCode.OK)
                {
```

```
                    Console.WriteLine("fail to get metadata of object {0},
HttpStatusCode:{1}, ErrorCode:{2}.", key, (int) result.HttpStatusCode,
result.HttpStatusCode);
                    return;
                }

                Console.WriteLine("get object metadata response: {0}", (int)
result.HttpStatusCode);
                Console.WriteLine("object ETag:{0}, object versionId:{1}",
result.ETag, result.VersionId);
                Console.WriteLine("object storage class:{0}",
result.StorageClass);
                Console.WriteLine("object content length:{0}",
result.ContentLength);
                Console.WriteLine("object last modified time:{0}",
result.LastModified);
            }
            catch (Exception e)
            {
                Console.WriteLine(e.Message);
            }
        }
    }
}
```

请求参数

GetObjectMetadata可设置的参数如下：

参数	类型	说明	是否必要
BucketName	string	桶的名称	是
EtagToMatch	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
ModifiedSinceDateUtc	DateTime	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
EtagToNotMatch	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
UnmodifiedSinceDateUtc	DateTime	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	int	读取对象指定的分片，该参数大于等于1，小于等于10000	否
VersionId	string	当桶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	否

		候，默认获取最新版本的对象数据	
返回结果 GetObjectMetadata返回的结果如下：	**类型**	**说明**	**是否必要**

参数	类型	说明
ContentLength	long	本次请求返回数据的大小（单位：字节）
ETag	string	对象的Entity Ttag
LastModified	DateTime	最近一次修改对象的时间
VersionId	string	对象最新的版本ID
StorageClass	string	对象的存储类型

设置对象访问权限

功能说明

PutACL操作可以通过access control list（ACL）设置一个对象的访问权限。用户在设置对象的ACL之前需要具备WRITE_ACP 权限。

对象的访问权限说明：

权限类型	说明
READ	可以对桶进行list操作
READ_ACP	可以读取桶的ACL信息。桶的所有者默认具有桶的READ_ACP权限
WRITE	可以在桶中创建对象，修改原有对象数据和删除对象
WRITE_ACP	可以修改桶的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桶的任意权限。桶的所有者默认具有桶的WRITE_ACP权限
FULL_CONTROL	同时授予READ、READ_ACP、WRITE和WRITE_ACP权限

代码示例

```
using System;
using System.Threading.Tasks;
using Amazon.Runtime;
using Amazon.S3;
using Amazon.S3.Model;

namespace DotNetSDK.ObjectOperation
{
    public class PutObjectACLExample
    {
        public static async Task PubObjectACL()
        {
            var accessKey = "<your-access-key>";
```

```
            var secretKey = "<your-secret-access-key>";
            var endpoint = "<your-endpoint>";
            var bucketName = "<your-bucket-name>";
            var key = "<your-object-key>";
            try
            {
                var credentials = new BasicAWSCredentials(accessKey, secretKey);
                var conf = new AmazonS3Config
                {
                    ServiceURL = endpoint
                };
                var s3Client = new AmazonS3Client(credentials, conf);
                var putACLRequest = new PutACLRequest()
                {
                    BucketName = bucketName,
                    Key = key,
                    // 添加一个公共读权限
                    CannedACL = S3CannedACL.PublicRead
                };

                var result = await s3Client.PutACLAsync(putACLRequest);
                if (result.HttpStatusCode != System.Net.HttpStatusCode.OK)
                {
                    Console.WriteLine("fail to put ACL to object {0},
HttpStatusCode:{1}, ErrorCode:{2}.", key, (int) result.HttpStatusCode,
result.HttpStatusCode);
                    return;
                }

                Console.WriteLine("set {0} to object {1}",
putACLRequest.CannedACL.Value, key);
            }
            catch (Exception e)
            {
                Console.WriteLine(e.Message);
            }
        }
    }
}
```

请求参数

PutACL可设置的参数如下：

参数	类型	说明	是否必要
CannedACL	S3CannedACL	配置对象的预定义的标准ACL信息	否
AccessControlList	S3AccessControlList	配置自定义的ACL信息	否
BucketName	string	桶的名称	是
Key	string	对象的key	是
VersionId	string	设置对象的version id	否

S3CannedACL包含了一组预定义的访问控制权限，可以应用于对象的访问权限如下：

权限	说明
NoACL	默认访问权限，对象的所有者拥有FULL_CONTROL权限，其他用户没有访问权限
Private	对象的所有者拥有FULL_CONTROL权限，其他用户没有访问权限
PublicRead	对象的所有者拥有FULL_CONTROL权限，其他用户拥有READ权限
PublicReadWrite	对象的所有者拥有FULL_CONTROL权限，其他用户拥有READ和WRITE权限
BucketOwnerRead	对象的所有者拥有FULL_CONTROL权限，桶的所有者拥有READ权限
BucketOwnerFullControl	对象的所有者和桶的所有者拥有FULL_CONTROL权限

获取对象访问权限

功能说明

GetACL操作可以获取对象的access control list（ACL）信息。对象的ACL可以在上传对象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）权限才可以查询对象的ACL信息。

代码示例

```
using System;
using System.Threading.Tasks;
using Amazon.Runtime;
using Amazon.S3;
using Amazon.S3.Model;

namespace DotNetSDK.ObjectOperation
{
    public class GetObjectACLExample
    {
        public static async Task GetObjectACL()
        {
            var accessKey = "<your-access-key>";
            var secretKey = "<your-secret-access-key>";
            var endpoint = "<your-endpoint>";
            var bucketName = "<your-bucket-name>";
            var key = "<your-object-key>";
            try
            {
                var credentials = new BasicAWSCredentials(accessKey, secretKey);
                var conf = new AmazonS3Config
                {
                    ServiceURL = endpoint
                };
                var s3Client = new AmazonS3Client(credentials, conf);
                var getAclRequest = new GetACLRequest()
                {
                    BucketName = bucketName,
                    Key = key
                };
```

```
                var result = await s3Client.GetACLAsync(getAclRequest);
                if (result.HttpStatusCode != System.Net.HttpStatusCode.OK)
                {
                    Console.WriteLine("fail to get ACL of bucket {0},
HttpStatusCode:{1}, ErrorCode:{2}.", bucketName, (int) result.HttpStatusCode,
result.HttpStatusCode);
                    return;
                }

                foreach (var grant in result.AccessControlList.Grants)
                {
                    Console.WriteLine("Type:{0}, CanonicalUser:{1}, DisplayName:
{2}, EmailAddress:{3}, URI:{4}, Permission:{5}",
                        grant.Grantee.Type, grant.Grantee.CanonicalUser,
grant.Grantee.DisplayName, grant.Grantee.EmailAddress, grant.Grantee.URI,
grant.Permission.Value);
                }

                Console.WriteLine("OwnerId:{0}, OwnerDisplayName:{1}.",
result.AccessControlList.Owner.Id, result.AccessControlList.Owner.DisplayName);
            }
            catch (Exception e)
            {
                Console.WriteLine(e.Message);
            }
        }
    }
}
```

请求参数

GetACL可设置的参数如下：

参数	类型	说明	是否必要
BucketName	string	桶的名称	是
Key	string	对象的key	是
VersionId	string	设置对象的版本	否

返回结果

GetACL返回的结果如下：

属性名	类型	说明
Grants	List<S3Grant>	Grant信息的数组，包含了每项授权和被授予人的信息
Owner	Owner	对象的所有者信息

获取对象标签

功能说明

GetObjectTagging操作可以查询对象的标签信息，对象标签以key-value的形式设置。桶的所有者默认拥有查询桶中对象的标签的权限，并且可以将权限授予其他用户。

代码示例

```
using System;
using System.Threading.Tasks;
using Amazon.Runtime;
using Amazon.S3;
using Amazon.S3.Model;

namespace DotNetSDK.ObjectOperation
{
    public class GetObjectTaggingExample
    {
        public static async Task GetObjectTagging()
        {
            var accessKey = "<your-access-key>";
            var secretKey = "<your-secret-access-key>";
            var endpoint = "<your-endpoint>";
            var bucketName = "<your-bucket-name>";
            var key = "<your-object-key>";
            var credentials = new BasicAWSCredentials(accessKey, secretKey);
            try
            {
                var conf = new AmazonS3Config
                {
                    ServiceURL = endpoint
                };
                var s3Client = new AmazonS3Client(credentials, conf);
                var getObjectTaggingRequest = new GetObjectTaggingRequest()
                {
                    BucketName = bucketName,
                    Key = key
                };
                var result = await
s3Client.GetObjectTaggingAsync(getObjectTaggingRequest);
                if (result.HttpStatusCode != System.Net.HttpStatusCode.OK)
                {
                    Console.WriteLine("fail to get tags of object {0},
HttpStatusCode:{1}, ErrorCode:{2}.", key, (int) result.HttpStatusCode,
result.HttpStatusCode);
                    return;
                }

                Console.WriteLine("the tags of {0} are: ", key);
                foreach (var tag in result.Tagging)
                {
                    Console.WriteLine("key={0}, value={1}", tag.Key, tag.Value);
                }
            }
            catch (Exception e)
            {
                Console.WriteLine(e.Message);
            }
        }
    }
```

```
}
```

请求参数

GetObjectTagging可设置的参数如下：

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

返回结果

GetObjectTagging返回的结果如下：

参数	类型	说明
Tagging	List<Tag>	对象标签信息数组，数组中的每一项包含了标签Key和Value的值

删除对象标签

功能说明

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

代码示例

```
using System;
using System.Threading.Tasks;
using Amazon.Runtime;
using Amazon.S3;
using Amazon.S3.Model;

namespace DotNetSDK.ObjectOperation
{
    public class DeleteObjectTaggingExample
    {
        public static async Task DeleteObjectTagging()
        {
            var accessKey = "<your-access-key>";
            var secretKey = "<your-secret-access-key>";
            var endpoint = "<your-endpoint>";
            var bucketName = "<your-bucket-name>";
            var key = "<your-object-key>";
            try
            {
                var credentials = new BasicAWSCredentials(accessKey, secretKey);
                var conf = new AmazonS3Config
                {
                    ServiceURL = endpoint
```

```
                };
                var s3Client = new AmazonS3Client(credentials, conf);
                var deleteObjectTaggingRequest = new
DeleteObjectTaggingRequest()
                {
                    BucketName = bucketName,
                    Key = key
                };
                var result = await
s3Client.DeleteObjectTaggingAsync(deleteObjectTaggingRequest);
                if (result.HttpStatusCode !=
System.Net.HttpStatusCode.NoContent)
                {
                    Console.WriteLine("fail to delete tags of object {0},
HttpStatusCode:{1}, ErrorCode:{2}.", key, (int) result.HttpStatusCode,
result.HttpStatusCode);
                    return;
                }

                Console.WriteLine("deleted the tags of object {0} ", key);
            }
            catch (Exception e)
            {
                Console.WriteLine(e.Message);
            }
        }
    }
}
```

请求参数

DeleteObjectTagging可设置的参数如下：

参数	类型	说明	是否必要
BucketName	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version id	否

设置对象标签

功能说明

PutObjectTagging操作可以为对象设置标签，对象标签以key-value的形式设置，每个对象最多可以有10个标签。桶的所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。

代码示例

```
using System;
using System.Threading.Tasks;
using Amazon.Runtime;
using Amazon.S3;
using Amazon.S3.Model;
```

```
namespace DotNetSDK.ObjectOperation
{
    public class PutObjectTaggingExample
    {
        public static async Task PutObjectTagging()
        {
            var accessKey = "<your-access-key>";
            var secretKey = "<your-secret-access-key>";
            var endpoint = "<your-endpoint>";
            var bucketName = "<your-bucket-name>";
            var key = "<your-object-key>";
            try
            {
                var credentials = new BasicAWSCredentials(accessKey, secretKey);
                var conf = new AmazonS3Config
                {
                    ServiceURL = endpoint
                };
                var s3Client = new AmazonS3Client(credentials, conf);
                var putObjectTaggingRequest = new PutObjectTaggingRequest()
                {
                    BucketName = bucketName,
                    Key = key,
                };
                putObjectTaggingRequest.Tagging.TagSet.Add(new Tag()
                {
                    Key = "<key1>",
                    Value = "<value1>"
                });
                putObjectTaggingRequest.Tagging.TagSet.Add(new Tag()
                {
                    Key = "<key2>",
                    Value = "<value2>"
                });
                var result = await
s3Client.PutObjectTaggingAsync(putObjectTaggingRequest);
                if (result.HttpStatusCode != System.Net.HttpStatusCode.OK)
                {
                    Console.WriteLine("fail to put tags of object {0},
HttpStatusCode:{1}, ErrorCode:{2}.", key, (int) result.HttpStatusCode,
result.HttpStatusCode);
                }
            }
            catch (Exception e)
            {
                Console.WriteLine(e.Message);
            }
        }
    }
}
```

请求参数

PutObjectTagging可设置的参数如下：

参数	类型	说明	是否必要
BucketName	string	设置标签信息的对象所在桶的名称	是
Key	string	设置标签信息的对象的key	是
VersionId	string	设置标签信息的对象的version id	否

生成下载预签名URL

功能说明

PutObjectTagging操作可以为对象设置标签，对象标签以key-value的形式设置，每个对象最多可以有10个标签。桶的所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。

代码示例

```
using System;
using System.Threading.Tasks;
using Amazon.Runtime;
using Amazon.S3;
using Amazon.S3.Model;

namespace DotNetSDK.ObjectOperation
{
    public class PutObjectTaggingExample
    {
        public static async Task PutObjectTagging()
        {
            var accessKey = "<your-access-key>";
            var secretKey = "<your-secret-access-key>";
            var endpoint = "<your-endpoint>";
            var bucketName = "<your-bucket-name>";
            var key = "<your-object-key>";
            try
            {
                var credentials = new BasicAWSCredentials(accessKey, secretKey);
                var conf = new AmazonS3Config
                {
                    ServiceURL = endpoint
                };
                var s3Client = new AmazonS3Client(credentials, conf);
                var putObjectTaggingRequest = new PutObjectTaggingRequest()
                {
                    BucketName = bucketName,
                    Key = key,
                };
                putObjectTaggingRequest.Tagging.TagSet.Add(new Tag()
                {
                    Key = "<key1>",
                    Value = "<value1>"
                });
                putObjectTaggingRequest.Tagging.TagSet.Add(new Tag()
                {
                    Key = "<key2>",
                    Value = "<value2>"
```

```
                });
                var result = await
s3Client.PutObjectTaggingAsync(putObjectTaggingRequest);
                if (result.HttpStatusCode != System.Net.HttpStatusCode.OK)
                {
                    Console.WriteLine("fail to put tags of object {0},
HttpStatusCode:{1}, ErrorCode:{2}.", key, (int) result.HttpStatusCode,
result.HttpStatusCode);
                }
            }
            catch (Exception e)
            {
                Console.WriteLine(e.Message);
            }
        }
    }
}
```

请求参数

PutObjectTagging可设置的参数如下：

参数	类型	说明	是否必要
BucketName	string	设置标签信息的对象所在桶的名称	是
Key	string	设置标签信息的对象的key	是
VersionId	string	设置标签信息的对象的version id	否

生成上传预签名URL

功能说明

GetPreSignedURL接口为一个指定对象生成一个预签名的上传链接，访问该链接可以直接上传对象。

代码示例

生成上传预签名URL：

```
public string GeneratePutUrl()
{
    Console.Out.WriteLine("generateUploadUrl");

    GetPreSignedUrlRequest request = new GetPreSignedUrlRequest
    {
        BucketName = "<your-bucket-name>",
        Key = "<your-object-key>",
        Expires = DateTime.Now.AddMinutes(5),
        Protocol = Protocol.HTTP,  //Protocol.HTTP or Protocol.HTTPS
        Verb = HttpVerb.PUT, // 设置 HTTP 方法为 PUT，用于上传
        ContentType = "application/octet-stream"  // 指定 Content-Type
    };

    string url = this.s3Client.GetPreSignedURL(request);
    Console.Out.WriteLine("generateUploadUrl: {0}", url);
```

```
    return url;
}
```

通过该预签名URL，可以直接将文件上传到指定的桶：

```
public void PutObjUsingPresignedUrl(string url, string filePath)
{
    try
    {
        using (WebClient client = new WebClient())
        {
            client.Headers.Add("Content-Type", "application/octet-stream");
            client.UploadFile(url, "PUT", filePath);
        }
        Console.WriteLine("Upload successful. File uploaded from: " + filePath);
    }
    catch (Exception ex)
    {
        Console.WriteLine("Upload failed: " + ex.ToString());
    }
}
```

请求参数

参数	类型	说明	是否必要
BucketName	string	桶的名称	是
Key	string	对象的key	是
Expires	DateTime	超时的时间戳	否
Protocol	Protocol	指定生成的URL使用http协议还是https协议	否
Verb	HttpVerb	HTTP 方法，设置为 PUT 以生成上传 URL，默认为GET	否
ContentType	String	对象的ContentType	否。若生成预签名URL时指定了，则通过预签名链接上传时也需要指定为相同的ContentType

返回结果

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

服务端加密

功能说明

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

代码示例

```
using System;
using System.Threading.Tasks;
using Amazon.Runtime;
using Amazon.S3;
using Amazon.S3.Model;

namespace DotNetSDK.ObjectOperation
{
    public class PutObjectWithEncryptionExample
    {
        public static async Task PutObjectWithEncryption()
        {
            var accessKey = "<your-access-key>";
            var secretKey = "<your-secret-access-key>";
            var endpoint = "<your-endpoint>";
            var bucketName = "<your-bucket-name>";
            var key = "<your-object-key>";
            var filePath = "<file-path>";
            try
            {
                var credentials = new BasicAWSCredentials(accessKey, secretKey);
                var conf = new AmazonS3Config
                {
                    ServiceURL = endpoint,
                    UseHttp = false
                };
                var s3Client = new AmazonS3Client(credentials, conf);
                var putObjectRequest = new PutObjectRequest()
                {
                    BucketName = bucketName,
                    Key = key,
                    FilePath = filePath,
                    ServerSideEncryptionMethod = new
ServerSideEncryptionMethod("AES256")
                };

                var result = await s3Client.PutObjectAsync(putObjectRequest);
                if (result.HttpStatusCode != System.Net.HttpStatusCode.OK)
                {
                    Console.WriteLine("fail to put object {0}, HttpStatusCode:
{1}, ErrorCode:{2}.", key, (int) result.HttpStatusCode, result.HttpStatusCode);
                    return;
                }
                Console.WriteLine("object ETag:{0}, object versionId:{1}",
result.ETag, result.VersionId);
                Console.WriteLine("encryption method:{0}",
result.ServerSideEncryptionMethod.Value);
```

```
            }
            catch (Exception e)
            {
                Console.WriteLine(e.Message);
            }
        }
    }
}
```

请求参数

加密参数说明如下：

参数	类型	说明	是否必要
ServerSideEncryptionMethod	ServerSideEncryptionMethod	指定服务端采用的加密算法，如AES256、SM4	否

返回结果

PutObject返回的结果如下：

参数	类型	说明
ETag	string	上传对象后的对应的Entity Tag
VersionId	string	上传对象后相应的版本id
ServerSideEncryptionMethod	ServerSideEncryptionMethod	返回对象数据加密的算法名称

获取多版本对象列表

功能说明

ListVersions操作可以获取关于对象版本信息的元数据，执行该操作需要对桶有READ权限。

代码示例

```
using System;
using System.Threading.Tasks;
using Amazon.Runtime;
using Amazon.S3;
using Amazon.S3.Model;

namespace DotNetSDK.ObjectOperation
```

```
{
    public class ListVersionsExample
    {
        public static async Task ListVersions()
        {
            var accessKey = "<your-access-key>";
            var secretKey = "<your-secret-access-key>";
            var endpoint = "<your-endpoint>";
            var bucketName = "<your-bucket-name>";
            var key = "<your-object-key>";
            try
            {
                var credentials = new BasicAWSCredentials(accessKey, secretKey);
                var conf = new AmazonS3Config
                {
                    ServiceURL = endpoint
                };
                var s3Client = new AmazonS3Client(credentials, conf);
                var listVersionsRequest = new ListVersionsRequest()
                {
                    BucketName = bucketName,
                    Prefix = key
                };
                var result = await
s3Client.ListVersionsAsync(listVersionsRequest);
                if (result.HttpStatusCode != System.Net.HttpStatusCode.OK)
                {
                    Console.WriteLine("fail to list versions, HttpStatusCode:
{0}, ErrorCode:{1}.", (int) result.HttpStatusCode, result.HttpStatusCode);
                    return;
                }

                foreach (var s3ObjectVersion in result.Versions)
                {
                    Console.WriteLine("key: {0}, versionId: {1}.",
s3ObjectVersion.Key, s3ObjectVersion.VersionId);
                }
            }
            catch (Exception e)
            {
                Console.WriteLine(e.Message);
            }
        }
    }
}
```

请求参数

ListVersions可以设置的参数如下：

参数	类型	说明	是否必要
BucketName	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	int	设置response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

返回结果

ListVersions返回的结果如下：

属性名	类型	说明
CommonPrefixes	List<string>	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	int	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	本次查询返回的最后一个的对象version id

属性名	类型	说明
Prefix	string	与请求中设置的Prefix一致
Versions	List<S3ObjectVersion>	对象版本信息的数组，数组中每一项包含了对象的Entity Tag、是否为最新版本以及DeleteMarker、对象key、最新修改时间、拥有者、大小、存储类型和对象版本的信息

分片上传接口

融合接口

SDK提供封装好的融合接口，方便用户实现分片上传的功能。

接口定义

```
public void Upload(string filePath, string bucketName, string key);

public void Upload(Stream stream, string bucketName, string key);

public void Upload(TransferUtilityUploadRequest request);

public Task UploadAsync(string filePath, string bucketName, string key,
CancellationToken cancellationToken = default);

public Task UploadAsync(Stream stream, string bucketName, string key,
CancellationToken cancellationToken = default);

public Task UploadAsync(TransferUtilityUploadRequest request, CancellationToken
cancellationToken = default);
```

代码示例

```
class TransDemo
{
    private readonly AmazonS3Client s3Client;
    private readonly TransferUtility utility;
    private string bucket = "<your-bucket-name>";

    public TransDemo()
    {
        var credentials = new BasicAWSCredentials("<your-access-key>", "<your-
secret-key>");
        var conf = new AmazonS3Config
        {
            ServiceURL = "<your-endpoint>",
        };
        this.s3Client = new AmazonS3Client(credentials, conf);
        this.utility = new TransferUtility(this.s3Client);
    }
```

```
    public void UploadFile()
    {
        Console.Out.WriteLine("UploadFile");
        var key = "<your-object-key>";
        var filePath = "<file-path>";
        this.utility.Upload(filePath, bucket, key);
        Console.Out.WriteLine("UploadFile success");
    }

    public void UploadFileRequest()
    {
        Console.Out.WriteLine("UploadFileRequest");
        var key = "<your-object-key>";
        var filePath = "<file-path>";
        TransferUtilityUploadRequest req = new TransferUtilityUploadRequest();
        req.BucketName = this.bucket;
        req.Key = key;
        //req.FilePath = filePath;
        req.InputStream = new FileStream(filePath, FileMode.Open,
FileAccess.Read);
        req.PartSize = 5 * 1024 * 1024;
        req.CannedACL = S3CannedACL.PublicRead;

        this.utility.Upload(req);
        Console.Out.WriteLine("UploadFileRequest success");
    }
}
```

请求参数

参数名	类型	说明	是否必要
bucketName	string	桶名	是
key	string	要上传的对象名称	是
filePath	string	上传的文件路径（和stream二选一）	否
stream	Stream	上传的文件流（和filePath二选一）	否

TransferUtilityUploadRequest主要参数：

参数名	类型	说明	是否必要
BucketName	string	桶名	是
Key	string	要上传的对象名称	是
FilePath	string	上传的文件路径（和InputStream二选一）	否
InputStream	Stream	上传的文件流（和FilePath二选一）	否
PartSize	long	分片大小，默认5MB	否
ContentType	string	描述上传文件格式的标准MIME类型	否
CannedACL	S3CannedACL	标准ACL信息（Private|PublicRead）	否

关于Content-Type的配置

Content-Type用于标识文件的资源类型，比如`image/png`， `image/jpg`是图片类型，`video/mpeg`， `video/mp4`是视频类型，`text/plain`， `text/html`是文本类型， 浏览器针对不同的Content-Type会有不同的操作，比如图片类型可以预览，视频类型可以播放，文本类型可以直接打开。`application/octet-stream`类型会直接打开下载窗口。

在dotnet sdk中，如果用户没有设置Content-Type，会根据对象的key后缀扩展名自动生成Content-Type。

上传分片

功能说明

分片上传操作可以将超过5GB的大文件分割后上传，一共包含三个步骤：首先，发起分片上传请求获取一个upload id。然后，将大文件分割成分片后上传，除了最后一个分片，每个分片的数据大小为5MB~5GB，每个分片上传的时候附带upload id。最后，发送一个带有upload id的请求，完成分片上传操作。

代码示例

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Threading.Tasks;
using Amazon.Runtime;
using Amazon.S3;
using Amazon.S3.Model;

namespace DotNetSDK.ObjectOperation
{
    public class UploadPartExample
    {
        public static async Task UploadPart()
        {
            var accessKey = "<your-access-key>";
            var secretKey = "<your-secret-access-key>";
            var endpoint = "<your-endpoint>";
            var bucketName = "<your-bucket-name>";
            var key = "<your-object-key>";
            var filePath = "<file-path>";
            try
            {
                var credentials = new BasicAWSCredentials(accessKey, secretKey);
                var conf = new AmazonS3Config
                {
                    ServiceURL = endpoint
                };
                var s3Client = new AmazonS3Client(credentials, conf);
                //1. 发起一个分片上传操作请求，获取upload id
                var initiateMultipartUploadRequest = new
InitiateMultipartUploadRequest()
                {
                    BucketName = bucketName,
```

```
                    Key = key
                };
                var initiateMultipartUploadResponse = await
s3Client.InitiateMultipartUploadAsync(initiateMultipartUploadRequest);
                var uploadId = initiateMultipartUploadResponse.UploadId;
                Console.WriteLine("upload id: {0}", uploadId);
                // 2. 分割大文件然后分片上传
                var partSize = 1024 * 1024 * 16;
                var fileInfo = new FileInfo(filePath);
                var fileLen = fileInfo.Length;
                var partNumber = fileLen / partSize;
                if (fileLen % partSize != 0)
                {
                    partNumber++;
                }

                var etagList = new List<PartETag>();
                using (var fs = File.Open(filePath, FileMode.Open))
                {
                    for (var i = 0; i < partNumber; i++)
                    {
                        var seekBytes = (long) partSize * i;
                        fs.Seek(seekBytes, 0);
                        var size = (partSize < fileLen - seekBytes) ? partSize :
(fileLen - seekBytes);
                        var uploadPartRequest = new UploadPartRequest()
                        {
                            BucketName = bucketName,
                            Key = key,
                            UploadId = uploadId,
                            InputStream = fs,
                            PartSize = size,
                            PartNumber = i + 1
                        };
                        // 分片上传
                        var uploadPartResponse = await
s3Client.UploadPartAsync(uploadPartRequest);
                        etagList.Add(new PartETag(uploadPartResponse.PartNumber,
uploadPartResponse.ETag));
                        Console.WriteLine("finish upload part {0}/{1}",
etagList.Count, partNumber);
                    }
                }

                // 3. 完成分片上传
                var completeMultipartUploadRequest = new
CompleteMultipartUploadRequest()
                {
                    BucketName = bucketName,
                    Key = key,
                    UploadId = uploadId
                };
                foreach (var etag in etagList)
                {
                    completeMultipartUploadRequest.PartETags.Add(etag);
                }
```

```
                var completeMultipartUploadResponse = await
s3Client.CompleteMultipartUploadAsync(completeMultipartUploadRequest);
                if (completeMultipartUploadResponse.HttpStatusCode !=
System.Net.HttpStatusCode.OK)
                {
                    Console.WriteLine("fail to complete multipart upload,
HttpStatusCode:{0}, ErrorCode:{1}.", (int)
completeMultipartUploadResponse.HttpStatusCode,
                        completeMultipartUploadResponse.HttpStatusCode);
                }
                else
                {
                    Console.WriteLine("complete multipart upload.");
                }
            }
            catch (Exception e)
            {
                Console.WriteLine(e.Message);
            }
        }
    }
}
```

创建分片上传任务

InitiateMultipartUpload 用于创建分片上传任务，返回分片上传任务的ID。

可设置的参数如下：

参数	类型	说明	是否必要
CannedACL	S3CannedACL	配置上传对象的预定义的标准ACL信息	否
BucketName	string	桶的名称	是
ContentType	string	描述上传文件格式的标准MIME类型	否
Key	string	上传文件到媒体存储服务后对应的key	是
Metadata	MetadataCollection	本次请求附带的元数据信息	否
TagSet	List<Tag>	对象的标签信息	否
WebsiteRedirectLocation	string	如果桶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桶下的其他对象或者外部的URL	否

返回的结果如下：

参数	类型	说明
BucketName	string	执行分片上传的桶的名称
Key	string	本次分片上传对象的名称
UploadId	string	本次分片上传操作Id

上传一个分片

UploadPart用于获取到分片任务upload id之后，通过upload id来上传分片数据到指定的分片上传任务对应的对象。

可设置的参数如下：

参数	类型	说明	是否必要
InputStream	Stream	对象的数据	是
BucketName	string	桶的名称	是
PartSize	long	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
MD5Digest	string	上传对象数据的base64编码的128位MD5值，不包含请求头部的信息	否
Key	string	上传文件到媒体存储服务后对应的key	是
PartNumber	int	说明当前数据在文件中所属的分片，大于等于1，小于等于10000	是
UploadId	string	通过InitiateMultipartUpload操作获取的UploadId，与一个分片上传的对象对应	是

返回的结果如下：

参数	类型	说明
ETag	string	本次上传分片后对应的Entity Tag
PartNumber	int	分片上传数据在文件中所属的分片

完成分片上传任务

完成所有分片的上传之后，调用完成接口CompleteMultipartUpload，服务端会把所有分片合并成对象保存。

可设置的参数如下：

参数	类型	说明	是否必要
BucketName	string	执行分片上传的桶的名称	是
Key	string	上传文件到媒体存储服务后对应的key	是
PartETags	List<PartETag>	包含了每个已上传的分片的ETag和PartNUmber等信息	否
UploadId	string	通过CreateMultipartUpload操作获取的UploadId，与一个对象的分片上传对应	是

返回的结果如下：

参数	类型	说明
BucketName	string	执行分片上传的桶的名称
ETag	string	本次上传对象后对应的Entity Tag
Key	string	上传文件到媒体存储服务后对应的key
Location	string	上传对象后对应的URI
VersionId	string	上传对象后相应的版本id

列举分片上传任务

功能说明

ListMultipartUploads操作可以列出一个桶中正在进行的分片上传任务，这些分片上传任务的请求已经发起，但是还没完成或者被中止。

代码示例

```
using System;
using System.Threading.Tasks;
using Amazon.Runtime;
using Amazon.S3;
using Amazon.S3.Model;

namespace DotNetSDK.ObjectOperation
{
    public class ListMultipartUploadsExample
    {
        public static async Task ListMultipartUploads()
        {
            var accessKey = "<your-access-key>";
```

```
            var secretKey = "<your-secret-access-key>";
            var endpoint = "<your-endpoint>";
            var bucketName = "<your-bucket-name>";
            var credentials = new BasicAWSCredentials(accessKey, secretKey);
            try
            {
                var conf = new AmazonS3Config
                {
                    ServiceURL = endpoint
                };
                var s3Client = new AmazonS3Client(credentials, conf);
                var listMultipartUploadsRequest = new
ListMultipartUploadsRequest()
                {
                    BucketName = bucketName
                };

                var result = await
s3Client.ListMultipartUploadsAsync(listMultipartUploadsRequest);
                if (result.HttpStatusCode != System.Net.HttpStatusCode.OK)
                {
                    Console.WriteLine("fail to list multipart uploads in bucket
{0}, HttpStatusCode:{1}, ErrorCode:{2}.", bucketName, (int)
result.HttpStatusCode, result.HttpStatusCode);
                    return;
                }

                foreach (var multipartUpload in result.MultipartUploads)
                {
                    Console.WriteLine("key: {0}, uploadId: {1}",
multipartUpload.Key, multipartUpload.UploadId);
                }
            }
            catch (Exception e)
            {
                Console.WriteLine(e.Message);
            }
        }
    }
}
```

请求参数

ListMultipartUploads可设置的参数如下：

参数	类型	说明	是否必要
BucketName	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片上传的信息。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标识符之后的分片信息。如果设置了UploadIdMarker参数，则会返回对象key等于KeyMarker且UploadId大于UploadIdMarker的分片信息	否
MaxUploads	int64	用于指定相应消息体中正在进行的分片上传任务的最大数量，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片上传的信息，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的分片信息	否

返回结果

ListMultipartUploads返回的结果如下：

参数	类型	说明
BucketName	string	执行本操作的桶名称
CommonPrefixes	List<string>	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分片上传任务信息，否则只返回了数量为MaxUploads个的任务信息
KeyMarker	string	返回分片上传任务列表中的起始对象的key
MaxUploads	int	本次返回结果中包含的分片上传任务数量的最大值
NextKeyMarker	string	当IsTruncated为true时，NextKeyMarker可以作为后续查询已初始化的分片上传任务请求中的KeyMarker的值
NextUploadIdMarker	string	当IsTruncated为true时，NextKeyMarker可以作为后续查询已初始化的分片上传任务请求中的UploadIdMarker的值
Prefix	string	限定返回分片中对应对象的key必须以Prefix作为前缀
UploadIdMarker	string	返回分片上传任务列表中的起始UploadId
MultipartUploads	List<MultipartUpload>	包含了零个或多个已初始化的分片上传任务信息的数组。数组中的每一项包含了分片上传初始化时间、分片上传操作发起者、对象key、对象所有者、存储类型和UploadId等信息

列举已上传的分片

功能说明

ListParts操作可以列出一个分片上传操作中已经上传完毕但是还未合并的分片信息。使用ListParts操作需要提供object key和upload id，返回的结果最多包含1000个已上传的分片信息，默认返回1000个，可以通过设置MaxParts参数的值指定返回结果中分片信息的数量。

代码示例

```
using System;
using System.Threading.Tasks;
using Amazon.Runtime;
using Amazon.S3;
using Amazon.S3.Model;

namespace DotNetSDK.ObjectOperation
{
    public class ListPartsExample
    {
        public static async Task ListParts()
        {
            var accessKey = "<your-access-key>";
            var secretKey = "<your-secret-access-key>";
            var endpoint = "<your-endpoint>";
            var bucketName = "<your-bucket-name>";
            var key = "<your-object-key>";
            var uploadId = "<your-upload-id>";
            var credentials = new BasicAWSCredentials(accessKey, secretKey);
            try
            {
                var conf = new AmazonS3Config
                {
                    ServiceURL = endpoint
                };
                var s3Client = new AmazonS3Client(credentials, conf);
                var listPartsRequest = new ListPartsRequest()
                {
                    BucketName = bucketName,
                    Key = key,
                    UploadId = uploadId
                };

                var result = await s3Client.ListPartsAsync(listPartsRequest);
                if (result.HttpStatusCode != System.Net.HttpStatusCode.OK)
                {
                    Console.WriteLine("fail to list part of uploadId {0},
HttpStatusCode:{1}, ErrorCode:{2}.", uploadId, (int) result.HttpStatusCode,
result.HttpStatusCode);
                    return;
                }

                Console.WriteLine("uploaded parts:");
                foreach (var partDetail in result.Parts)
                {
                    Console.WriteLine("Etag:{0}, PartNumber:{1}, Size:{2}.",
partDetail.ETag, partDetail.PartNumber, partDetail.Size);
                }
            }
            catch (Exception e)
            {
                Console.WriteLine(e.Message);
            }
        }
    }
}
```

请求参数

ListParts可设置的参数如下：

参数	类型	说明	是否必要
BucketName	string	执行本操作的桶名称	是
Key	string	分片上传的对象的key	是
MaxParts	int	指定返回分片信息的数量，默认值和最大值均为1000	否
PartNumberMarker	string	用于指定返回part number大于PartNumberMarker的分片信息	否
UploadId	string	指定返回该id所属的分片上传的分片信息	是

返回结果

ListParts返回的结果如下：

参数	类型	说明
BucketName	string	执行本操作的桶名称
IsTruncated	bool	当为false时表示返回结果中包含了全部符合本次请求查询条件的上传分片信息，否则只返回了数量为MaxParts个的分片信息
Key	string	分片上传对象的名称
MaxParts	int	本次返回结果中包含的上传分片数量的最大值
NextPartNumberMarker	int	当IsTruncated为true时，NextPartNumberMarker可以作为后续查询已上传分片请求中的PartNumberMarker的值
Owner	Owner	分片上传对象的所有者信息，包含了用户名和Id等信息
Parts	List<PartDetail>	包含了已上传分片信息的数组，数组中的每一项包含了该分片的Entity tag、最后修改时间、PartNumber和大小等信息

参数	类型	说明
UploadId	string	分片上传操作的id

复制分片

功能说明

CopyPart操作可以从一个已存在的对象中拷贝指定分片的数据，当拷贝的对象大小超过5GB，必须使用UploadPartCopy操作完成对象的复制。除了最后一个分片外，每个拷贝分片的大小范围是[5MB，5GB]。在一个在拷贝大对象之前，需要使用CompleteMultiPartUpload操作获取一个upload id，在完成拷贝操作之后，需要使用CompleteMultipartUpload操作组装已拷贝的分片成为一个对象。

代码示例

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Amazon.Runtime;
using Amazon.S3;
using Amazon.S3.Model;

namespace DotNetSDK.ObjectOperation
{
    public class CopyPartsExample
    {
        public static async Task CopyParts()
        {
            var accessKey = "<your-access-key>";
            var secretKey = "<your-secret-access-key>";
            var endpoint = "<your-endpoint>";
            var sourceBucket = "<source-bucket>";
            var sourceKey = "<source-key>";
            var destinationBucket = "<destination-bucket>";
            var destinationKey = "<destination-key>";
            try
            {
                var credentials = new BasicAWSCredentials(accessKey, secretKey);
                List<CopyPartResponse> copyResponses = new
List<CopyPartResponse>();
                var conf = new AmazonS3Config
                {
                    ServiceURL = endpoint
                };
                var s3Client = new AmazonS3Client(credentials, conf);
                // 发起一个分片上传操作请求，获取upload id
                Task<InitiateMultipartUploadResponse>
taskInitiateMultipartUploadResp = s3Client.InitiateMultipartUploadAsync(new
InitiateMultipartUploadRequest()
                {
                    BucketName = destinationBucket,
                    Key = destinationKey
                });
                var uploadId = taskInitiateMultipartUploadResp.Result.UploadId;
                Console.WriteLine("upload id: {0}", uploadId);
```

```
                // 获取被拷贝对象的大小
                GetObjectMetadataRequest getObjectMetadataRequest = new
GetObjectMetadataRequest
                {
                    BucketName = sourceBucket,
                    Key = sourceKey
                };
                var getObjectMetadataResponse = await
s3Client.GetObjectMetadataAsync(getObjectMetadataRequest);
                long objectSize = getObjectMetadataResponse.ContentLength;
                // 拷贝分片
                long partSize = 5 * (long) Math.Pow(2, 20); // 5 MB.
                long bytePosition = 0;
                for (int i = 1; bytePosition < objectSize; i++)
                {
                    CopyPartRequest copyPartRequest = new CopyPartRequest
                    {
                        DestinationBucket = destinationBucket,
                        DestinationKey = destinationKey,
                        SourceBucket = sourceBucket,
                        SourceKey = sourceKey,
                        UploadId = uploadId,
                        FirstByte = bytePosition,
                        LastByte = bytePosition + partSize - 1 >= objectSize ?
objectSize - 1 : bytePosition + partSize - 1,
                        PartNumber = i
                    };
                    var copyPartResponse = await
s3Client.CopyPartAsync(copyPartRequest);
                    copyResponses.Add(copyPartResponse);
                    bytePosition += partSize;
                }

                // 完成拷贝分片
                CompleteMultipartUploadRequest completeRequest =
                    new CompleteMultipartUploadRequest
                    {
                        BucketName = destinationBucket,
                        Key = destinationKey,
                        UploadId = uploadId
                    };
                completeRequest.AddPartETags(copyResponses);
                CompleteMultipartUploadResponse completeUploadResponse = await
s3Client.CompleteMultipartUploadAsync(completeRequest);
                if (completeUploadResponse.HttpStatusCode !=
System.Net.HttpStatusCode.OK)
                {
                    Console.WriteLine("fail to get copy parts, HttpStatusCode:
{0}, ErrorCode:{1}.", (int) completeUploadResponse.HttpStatusCode,
completeUploadResponse.HttpStatusCode);
                    return;
                }

                Console.WriteLine("copy object from {0}/{1} to {2}/{3}.",
sourceBucket, sourceKey, destinationBucket, destinationKey);
            }
            catch (Exception e)
            {
```

```
                Console.WriteLine(e.Message);
            }
        }
    }
}
```

请求参数

CopyPart可设置的参数如下：

参数	类型	说明	是否必要
DestinationBucket	string	放置拷贝生成对象的桶名称	是
DestinationKey	string	拷贝生成对象的key	是
SourceBucket	string	放置被拷贝对象的桶名称	是
SourceKey	string	被拷贝对象的key	是
UploadId	string	与本次拷贝操作相应的分片上传Id	是
SourceVersionId	string	指定被拷贝对象的版本信息，如果不指定，默认拷贝对象的当前版本	否

返回结果

CopyPart返回的结果如下：

参数	类型	说明
ContentLength	long	拷贝分片的长度
ETag	string	拷贝分片的ETag
LastModified	string	拷贝分片的最新修改时间
PartNumber	int	拷贝分片的序号

取消分片上传任务

功能说明

AbortMultipartUpload操作用于终止一个分片上传任务。当一个分片上传任务被中止后，不会再有数据通过与之相应的upload id上传，同时已经被上传的分片所占用的空间会被释放。执行AbortMultipartUpload操作后，正在上传的分片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AbortMultipartUpload操作去释放全部上传成功的分片所占用的空间。可以通过执行ListParts操作来确认所有中止分片上传后所有已上传分片的空间是否被被释放。

代码示例

```
using System;
using System.Threading.Tasks;
using Amazon.Runtime;
```

```
using Amazon.S3;
using Amazon.S3.Model;

namespace DotNetSDK.ObjectOperation
{
    public class AbortMultipartUploadExample
    {
        public static async Task AbortMultipartUpload()
        {
            var accessKey = "<your-access-key>";
            var secretKey = "<your-secret-access-key>";
            var endpoint = "<your-endpoint>";
            var bucketName = "<your-bucket-name>";
            var key = "<your-object-key>";
            var uploadId = "<your-upload-id>";
            try
            {
                var credentials = new BasicAWSCredentials(accessKey, secretKey);
                var conf = new AmazonS3Config
                {
                    ServiceURL = endpoint
                };
                var s3Client = new AmazonS3Client(credentials, conf);
                var abortMultipartUploadRequest = new
AbortMultipartUploadRequest()
                {
                    BucketName = bucketName,
                    Key = key,
                    UploadId = uploadId
                };

                var result = await
s3Client.AbortMultipartUploadAsync(abortMultipartUploadRequest);
                if (result.HttpStatusCode !=
System.Net.HttpStatusCode.NoContent)
                {
                    Console.WriteLine("fail to abort multipart upload, uploadId:
{0}, HttpStatusCode:{1}, ErrorCode:{2}.", uploadId, (int) result.HttpStatusCode,
result.HttpStatusCode);
                    return;
                }

                Console.WriteLine("aborted multipart upload, uploadId:{0}.",
uploadId);
            }
            catch (Exception e)
            {
                Console.WriteLine(e.Message);
            }
        }
    }
}
```

请求参数

AbortMultipartUpload可设置的参数如下：

参数	类型	说明	是否必要
BucketName	string	桶的名称	是
Key	string	分片上传的对象的key	是
UploadId	string	指定需要终止的分片上传的id	是

安全凭证服务(STS)

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

初始化STS服务

```
private const string AK = "<your-access-key>";
private const string SK = "<your-secret-access-key>";
private const string ENDPOINT = "<your-endpoint>"; // e.g. http://endpoint or
https://endpoint
private readonly AmazonSecurityTokenServiceClient stsClient;

public S3ClientToCtyun()
{
    var credentials = new BasicAWSCredentials(AK, SK);
    var confSts = new AmazonSecurityTokenServiceConfig
    {
        ServiceURL = ENDPOINT
    };
    this.stsClient = new AmazonSecurityTokenServiceClient(credentials, confSts);
}
```

获取临时token

```
public void AssumeRole()
{
    var bucket = "<your-bucket-name>";
    var roleSessionName = "<your-session-name>";
    var roleArn = "arn:aws:iam:::role/xxxxxx";
    var policy = "{\"Version\":\"2012-10-17\"," + "\"Statement\":" + "
{\"Effect\":\"Allow\","
    + "\"Action\":[\"s3:*\"]," // 允许进行 S3 的所有操作。如果仅需要上传，这里可以设置为
PutObject
    + "\"Resource\":[\"arn:aws:s3:::" + bucket + "\",\"arn:aws:s3:::" + bucket +
"/*\"]"// 允许操作默认桶中的所有文件，可以修改此处来保证操作的文件
    + "}}";

    AssumeRoleRequest req = new AssumeRoleRequest();
    req.Policy = policy;
    req.RoleArn = roleArn;
    req.RoleSessionName = roleSessionName;
    var task = this.stsClient.AssumeRoleAsync(req);
    try
```

```
    {
        var result = task.Result;
        Console.Out.WriteLine("AssumeRole, ak={0}, sk={1}, token={2}",
result.Credentials.AccessKeyId,
            result.Credentials.SecretAccessKey,
result.Credentials.SessionToken);
    }
    catch (Exception ex)
    {
        Console.Out.WriteLine("exception: {0}", ex.Message);
    }
}
```

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

使用临时token

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

```
public class MyCredProvider : AWSCredentials
{
    private readonly object syncRoot = new object();
    private ImmutableCredentials creds;

    public MyCredProvider(string ak, string sk, string token)
    {
        creds = new ImmutableCredentials(ak, sk, token);
    }

    public override ImmutableCredentials GetCredentials()
    {
        lock (syncRoot)
        {
            return creds.Copy();
        }
    }

    // 更新token
    public void UpdateCred(string ak, string sk, string token)
    {
        lock (syncRoot)
        {
            creds = new ImmutableCredentials(ak, sk, token);
        }
    }
```

```
}
```

使用临时token

```
var ak = "<temporary-access-key>";
var sk = "<temporary-secret-access-key>";
var endPoint = "<your-endpoint>"; // e.g. http://endpoint or https://endpoint
var token = "<your-session-token>";
var credentials = new MyCredProvider(ak, sk, token);
var conf = new AmazonS3Config
{
    ServiceURL = endPoint
};
AmazonS3Client s3 = new AmazonS3Client(credentials, conf);
```