天翼云
State Cloud

对象存储（经典版）I 型

(Object-Oriented Storage,OOS)

Python SDK

天翼云科技有限公司

目录

1 开始使用.....1
1.1 要求.....1
1.2 使用方式.....1
1.3 options 配置项.....2
2 SDK 列表.....3
2.1 关于 Service 操作.....5
2.1.1 Get Service (ListBucket).....5
2.2 关于 Bucket 操作.....6
2.2.1 Put Bucket.....6
2.2.2 GET Bucket Acl.....6
2.2.3 GET Bucket.....7
2.2.4 DELETE Bucket.....9
2.2.5 PUT Bucket Policy.....10
2.2.6 GET Bucket Policy.....11
2.2.7 DELETE Bucket Policy.....11
2.2.8 PUT Bucket WebSite.....12
2.2.9 GET Bucket WebSite.....13
2.2.10 DELETE Bucket WebSite.....14
2.2.11 List Multipart Uploads.....14
2.2.12 PUT Bucket Logging.....20
2.2.13 GET Bucket Logging.....21
2.2.14 Head Bucket.....22

2.2.15 PUT Bucket Trigger 23
2.2.16 GET BucketTrigger 25
2.2.17 DELETE Bucket Trigger 26
2.2.18 PUT Bucket Lifecycle 27
2.2.19 GET Bucket Lifecycle 29
2.2.20 DELETE Bucket Lifecycle 31
2.3 关于 Object 的操作 32
2.3.1 PUT Object 32
2.3.2 GET Object 33
2.3.3 Head Object 34
2.3.4 DELETE Object 35
2.3.5 PUT Object - Copy 36
2.3.6 Initial Multipart Upload 37
2.3.7 Upload Part 39
2.3.8 Complete Multipart Upload 41
2.3.9 Abort Multipart Upload 42
2.3.10 List Parts 43
2.3.11 Copy Part 46
2.3.12 DELETE Multiple Objects 48
2.4 关于 AccessKey 的操作 50
2.4.1 CreateAccessKey 50
2.4.2 DeleteAccessKey 51

2.4.3 UpdateAccessKey 51

2.4.4 ListAccessKey 52

1 开始使用

1.1 要求

已注册天翼云账户，开通 OOS 服务。

创建 AccessKeyId 和 AccessSecretKey。AccessKeyId 和 AccessSecretKey 是您访问 OOS 的密钥，OOS 会通过它来验证您的资源请求，请妥善保管。

下载 python sdk 包，解压后执行标准 python 包安装命令：

```
# python setup.py install
```

如果是在 python3 环境下使用，请确保已安装 urllib3。

安装完成后在 examples 目录下有 examples.py 的详细示例，各个接口在示例中均有方法对应，请参照使用。

如果使用 Python2，参数中有中文，请确认参数为 unicode。

```
response = client.put_bucket_trigger(
    Bucket=BUCKET,
    TriggerConfiguration={
        'Triggers': [
            {
                'TriggerName': u'测试 Trigger',
                ...
            }
        ]
    }
)
```

1.2 使用方式

```
# 导入 oos 包
import oos
```

使用 oos.client 创建 oos client 对象

```
config = Config(signature_version='s3')
client = oos.client('s3', …)
```

```
buckets = client.list_buckets()
```

1.3 options 配置项

OOS options 介绍

参数	描述	是否必须
access_key_id	通过 OOS 控制台创建的 access key。	是
secret_access_key	通过 OOS 控制台创建的 secret access key。	是
endpoint_url	OOS 域名，如 http://oos-hz.ctyunapi.cn。	是
signature_version	计算签名版本，目前支持的版本为“s3”。	是
use_ssl	是否使用 https，布尔型，只能选 False。	是

示例

```
config = Config(signature_version='s3')
client = oos.client('s3', use_ssl=False, endpoint_url=ENDPOINT,
                    access_key_id=ACCESS_KEY,
                    secret_access_key=SECRET_KEY,
                    config=config)
```

2 SDK 列表

该部分主要介绍的内容是 OOS Python-SDK 对支持的所有功能，当用户发送请求给 OOS 时，可以通过签名认证的方式请求，也可以匿名访问。

一个完整的操作包括 API 接口调用和异常处理，基本用法：

```
try:
    response = client.delete_bucket(
        Bucket='NoSuchBucket'
    )
    pretty_print(response)
except exceptions.ClientError as e:
    print('Response code: {0}, request id {1}'
          .format(e.response['Error']['Code'],
            e.response['ResponseMetadata']['RequestId']))
```

其中 request id 为唯一标识每个请求，方便问题定位。返回结果 response 为一个 python dictionary，其中包含 http 的返回 header、code 等。

一个完整的返回结果：

```
{
    "Buckets": [
        {
            "CreationDate": "2018-09-20 03:05:26.223000+00:00",
            "Name": "wxs-test"
        },
        {
            "CreationDate": "2018-11-29 03:21:12.908000+00:00",
            "Name": "test-jf-logging"
        }
    ],
    "Owner": {
        "DisplayName": "",
        "ID": "yanxiao@ctyun.cn"
    },
    "ResponseMetadata": {
        "HTTPHeaders": {
            "content-length": "862",
            "content-type": "application/xml;charset=UTF-8",
            "date": "Mon, 10 Dec 2018 06:12:21 GMT",
            "server": "CTYUN",
            "x-amz-request-id": "560f0de60e5f49d0"
```

```
        },
        "HTTPStatusCode": 200,
        "HostId": "",
        "RequestId": "560f0de60e5f49d0",
        "RetryAttempts": 0
    }
}
```

2.1 关于 Service 操作

2.1.1 Get Service (ListBucket)

请求方法

```
buckets = client.list_buckets()
for bucket in buckets['Buckets']:
    print('Bucket: {0}'.format(bucket['Name']))
```

请求参数

该操作不使用请求参数。

返回参数

名称	描述
Bucket	存储 Bucket 信息的容器。
Buckets	存储一个或多个 Bucket 的容器。
CreationDate	Bucket 的创建日期。
DisplayName	Bucket 拥有者的用户显示姓名。
ID	Bucket 拥有者的用户 ID。
Name	Bucket 的名称。
Owner	Bucket 的拥有者的信息。

2.2 关于 Bucket 操作

2.2.1 Put Bucket

Put 操作用来创建一个新的 Bucket。只有在 OOS 中注册的用户才能创建一个 新的 bucekt，匿名请求无效，创建 Bucket 的用户将是 Bucket 的拥有者。

Bucket 的命名方式中并不是支持所有的字符，OOS 中 Bucket 的 name 长度 为 63 个字符以内，只支持小写字母数字，点(.)，以及横杠（-）。

请求方法

```
response = client.create_bucket(
    ACL='public-read-write',
    Bucket=BUCKET
)
```

请求参数

名称	描述	是否必须
Bucket	Bucket 名称。	是
ACL	设置创建 Bucket 的 ACL，private，public-read 或 public-read-write	否

返回参数

名称	描述
Location	Bucket 名称。

2.2.2 GET Bucket Acl

这个 Get 操作用来获取 Bucket 的 ACL 信息，用户必须对改 Bucket 有读权限。

请求方法

```
response = client.get_bucket_acl(
```

```
    Bucket=BUCKET
)
```

请求参数

名称	描述	是否必须
Bucket	Bucket 名称。	是

返回参数

名称	描述
DisplayName	Bucket 拥有者的显示名称。
Grant	存储 Permission 和 Grantee 的容器。
Grantee	用来存储 Display 和拥有 ID 的用户被承认的许可的容器。
ID	Bucket 拥有者的 ID 信息。
Owner	存储 Bucket 的拥有者信息的容器。
Permission	对一个 Bucket 认可的许可信息。

2.2.3 GET Bucket

这个 Get 操作返回 Bucket 中部分或者全部（最多 1000）的 object 信息。用户可以在请求元素中设置选择条件来获取 Bucket 中的 object 的子集。

要执行该操作，需要对操作的 Bucket 拥有读权限

请求方法

```
response = client.list_objects(
    Bucket=BUCKET,
    Marker='test',
    MaxKeys=10,
)
```

请求参数

名称	描述	是否必须
Bucket	Bucket 名称。	是
Delimiter	一个 delimiter 是一个用来对关键字们进行分组的字符。所有的关键字都包含 delimiter 和 prefix 间的相同子串，prefix 之后第一个遇到 delimiter 的字符串都会加到一个叫 CommonPrefix 的组中。如果没有特别定义 prefix，所有的关键字都会被返回，但不会有 CommonPrefix。Delimiter 只支持”/”，不支持其他分隔符。	否
Marker	指明在 Bucket 中 list object 的起始位置，Amazon S3 中 list object 按照阿拉伯字母顺序	否
MaxKeys	设置返回结果中最多显示的数量，如果查询结果超过最大数量，另一个变量是否翻页会标记为 true<IsTruncated>True<IsTruncated>,用来返回剩余的查询结果	否
Prefix	前缀用来限制返回的结果必须以这个前缀开始，可以通过前缀将 Bucket 分为若干个组。	否

返回元素

名称	描述
Contents	每个 object 返回的元数据。
CommonPrefixes	当定义 delimiter 之后，返回结果中会包含 CommonPrefixes，CommonPrefix 中包含以 prefix 开头，delimiter 结束的左右字符串组合，比如 prefix 是 note/,同时 delimiter 是斜杠（/），结果中的 note/summer/ju 将返回 note/summer/,其余结果将按照 maxkey 要求返回。
Delimiter	将 prefix 和第一次出现 delimiter 的所有查询结果滚动的存入 CommonPrefix 组中。
DisplayName	Object 的所有者显示名称。
ETag	通过 MD5 的方式计算出标签，ETag 主要用来反映对象内容的信息发生了改变，并不反映元数据的变化。
ID	对象拥有者的 ID。

IsTruncated	通过是（True）或否（false）来表示返回的结果是否为所有要求的结果。
Key	对象的关键字。
LastModified	记录的对象最后一个被修改的日期和时间。
Marker	标记从哪个位置开始罗列出 Bucket 中的 object。
Name	Bucket 的名称。
Owner	Bucket 的拥有者。
Prefix	关键字以特定的前缀开始。
Size	记录对象 object 的大小。
StorageClass	存储类型。

2.2.4 DELETE Bucket

该操作用来执行删除 Bucket 的操作，但要求所有 Bucket 中的 object 都必须被删除。

请求方法

```
response = client.delete_bucket(
    Bucket=BUCKET
)
```

请求参数

名称	描述	是否必须
Bucket	Bucket 名称。	是

返回元素

成功无返回值。

2.2.5 PUT Bucket Policy

policy 的添加或修改的操作。如果 Bucket 已经存在了 Policy，此操作会替换原有 Policy。只有 Bucket 的 owner 才能执行此操作，否则会返回 403 AccessDenied 错误。

请求方法

```
policy = {
    "Version": "2012-10-17",
    "Id": "aaaa-bbbb-cccc-dddd",
    "Statement": [
        {
            "Effect": "Allow",
            "Sid": "1",
            "Principal": {
                "AWS": "*"
            },
            "Action": ["s3:*"],
            "Resource": "arn:aws:s3:::test-jf/*"
        }
    ]
}
response = client.put_bucket_policy(
    Bucket=BUCKET,
    Policy=json.dumps(policy)
)
```

请求参数

名称	描述	是否必须
Bucket	Bucket 名称。	是
Policy	Policy 规则的字符串。	是

返回元素

成功没有返回值。

2.2.6 GET Bucket Policy

可以获得指定 Bucket 的 policy。只有 Bucket 的 owner 才能执行此操作，否则会返回 403 AccessDenied 错误。如果 Bucket 没有 policy，返回 404，NoSuchPolicy 错误

请求方法

```
response = client.get_bucket_policy(
    Bucket=BUCKET
)
```

请求参数

名称	描述	是否必须
Bucket	Bucket 名称。	是

返回元素

名称	描述
Policy	policy 的 String 格式。

2.2.7 DELETE Bucket Policy

可以删除指定 Bucket 的 policy。只有 Bucket 的 owner 才能执行此操作，否则会返回 403 AccessDenied 错误。如果 Bucket 没有 policy，返回 204 NoContent。

请求方法

```
response = client.delete_bucket_policy(
    Bucket=BUCKET
)
```

请求参数

名称	描述	是否必须

Bucket	Bucket 名称。	是

返回元素

成功没有返回值。

2.2.8 PUT Bucket WebSite

配置 Bucket 的 website，如果 Bucket 已经存在了 website，此操作会替换原有 website。只有 Bucket 的 owner 才能执行此操作，否则会返回 403 AccessDenied 错误。

WebSite 功能可以让用户将静态网站存放到 OOS 上。对于已经设置了 WebSite 的 Bucket，当用户访问 http://bucketName.oos-website-cn.oos.ctyunapi.cn 时，会跳转到用户指定的主页，当出现 4**错误时，会跳转到用户指定的出错页面。

请求方法

```
response = client.put_bucket_website(
    Bucket=BUCKET,
    WebsiteConfiguration={
        'ErrorDocument': {
            'Key': 'error.html'
        },
        'IndexDocument': {
            'Suffix': 'index.html'
        }
    }
)
```

请求参数

名称	描述	是否必须
Bucket	Bucket 名称。	是
WebsiteConfiguration	请求的容器。	是

IndexDocument	Suffix 元素的容器。	是
Suffix	在请求 website endpoint 上的路径时，Suffix 会被加在请求的后面。例如，如果 suffix 是 Index.html，而你请求的是 bucket/images/，那么返回的响应是名为 images/index.html 的 object。	是
ErrorDocument	Key 的容器。	否
Key	如果出现 4XX 错误，会返回指定的 Object。	条件

返回元素

成功没有返回值。

2.2.9 GET Bucket WebSite

此操作用来获得指定 Bucket 的 website。

请求方法

```
response = client.get_bucket_website(
    Bucket=BUCKET
)
```

请求参数

名称	描述	是否必须
Bucket	Bucket 名称。	是

返回元素

```
{
    "ErrorDocument": {
```

```
        "Key": "error1.html"
    },
    "IndexDocument": {
        "Suffix": "index1.html"
    },
}
```

2.2.10 DELETE Bucket WebSite

删除指定 Bucket 的 website。只有 bucket 的 owner 才能执行此操作，否则会返回 403 AccessDenied 错误。如果 bucket 没有 website，返回 200 OK。

请求方法

```
response = client.delete_bucket_website(
    Bucket=BUCKET
)
```

请求参数

名称	描述	是否必须
Bucket	Bucket 名称。	是

返回元素

成功没有返回值。

2.2.11 List Multipart Uploads

列出所有已经通过 Initiate Multipart Upload 请求初始化，但未完成或未终止的分片上传过程。响应中最多返回 1000 个分片上传过程的信息，它既是响应能返回的最大分片上传过程数目，也是请求的默认值。用户也可以通过设置 max-uploads 参数来限制响应中的分片上传过程数目。

如果当前的分片上传过程数超出了这个值，则响应中会包含一个值为 true 的 IsTruncated 元素。如果用户要列出多于这个值的分片上传过程信息，则需要继续调用 List Multipart Uploads 请求，并在请求中设置 key-marker 和 upload-id-marker 参数。

在响应体中，分片上传过程的信息通过 key 来排序。如果用户的应用程序中启动了多个使用同一 key 对象开头的分片上传过程，那么响应体中分片上传过程首先是通过 key 来排序，在相同 key 的分片上传内部则是按上传启动的起始时间的升序来进行排列。

请求方法

```
response = client.list_multipart_uploads(
    Bucket=BUCKET,
    MaxUploads=100
)
```

请求参数

名称	描述	是否必须
Bucket	Bucket 名称。	是
Delimiter	delimiter 是一个用来对关键字们进行分组的字符。所有的关键字都包含 delimiter 和 prefix 间的相同子串，prefix 之后第一个遇到 delimiter 的字符串都会加到一个叫 CommonPrefix 的组中。如果没有特别定义 prefix，所有的关键字都会被返回，但不会有 CommonPrefix。 类型：String	否
MaxUploads	设置返回的分片上传过程的最大数目，范围从 1 到 1000，1000 是响应体中能返回的分片上传信息的最大值。 类型：Integer 默认值：1000	否
KeyMarker	与 upload-id-marker 参数一起，该参数指定列表操作从什么位置后面开始。如果 upload-id-marker 参数没有被指定，那么只有比 key-marker 参数指定的 key 更大的 key 会被列出。如果	否

	upload-id-marker 参数被指定，任何包含与 key-marker 指定值相等或大于 key 的分片上传过程都会被包括，假设这些分片上传过程包含了比 upload-id-marker 指定值更大的 ID。 类型：String	
Prefix	该参数用于列出那些以 prefix 为前缀的正在进行的上传过程，用户可以使用多个 prefix 来把一个 Bucket 分成不同的组（可以考虑采用类似文件系统中的文件夹的作用那样，使用 prefix 来对 key 进行分组）。	否
UploadIdMarker	与 key-marker 参数一起，指定列表操作从什么位置后面开始。如果 key-marker 没有被指定，upload-id-marker 参数也会被忽略。否则，任何 key 值等于或大于 key-marker 的上传过程都会被包含在列表中，只要 ID 大于 upload-id-marker 指定的值。	否

返回元素

名称	描述
ListMultipartUploadsResult	包含整个响应的容器。 类型：容器 子节点：Bucket, KeyMarker, UploadIdMarker, NextKeyMarker,NextUploadIdMarker,Maxuploads,Delimiter,Prefix,Commonfixes,IsTruncated 父节点：无
Bucket	分片上传对应的对象名称。 类型：String 父节点：ListMultipartUploadsResult
KeyMarker	指定 key 值，在这个 key 当前位置或它之后开始列表操作。 类型：String 父节点：ListMultipartUploadsResult

UploadIdMarker	指定分片上传 ID，在这个 ID 所在位置之后开始列表操作。 类型：String 父节点：ListMultipartUploadsResult
NextKeyMarker	当此次列表不能将所有正在执行的分片上传过程列举完成时，NextKeyMarker 作为下一次列表请求的 key-marker 参数的值。 类型：String 父节点：ListMultipartUploadsResult
NextUploadIdMarker	当此次列表不能将所有正在执行的分片上传过程列举完成时，NextKeyMarker 作为下一次列表请求的 upload-id-marker 参数的值。 类型：String 父节点：ListMultipartUploadsResult
MaxUploads	响应中包含的上传过程的最大数目。 类型：String 父节点：ListMultipartUploadsResult
IsTruncated	标识此次分片上传过程中的所有片段是否全部被列出，如果为 true 则表示没有全部列出。如果分片上传过程的片段数超过了 MaxParts 元素指定的最大数，则会导致一次列表请求无法将所有片段数列出。 类型：Boolean 父节点：ListMultipartUploadsResult
Upload	某个分片上传过程的容器，响应体中可能包含 0 个或多个 Upload 元素。 类型：容器 子节点: Key, UploadId, InitiatorOwner, StorageClass,Initiated

	父节点: ListMultipartUploadsResult
Key	分片上传过程起始位置对象的 Key 值。 类型: Integer 父节点: Upload
UploadId	分片上传过程的 ID 号。 类型: Integer 父节点: Upload
Initiator	指定执行此次分片上传过程的用户账号。 子节点：ID，DisplayName 类型：容器 父节点：Upload
ID	OOS 账号的 ID 号。 类型：String 父节点：Initiator，Owner
DisplayName	OOS 账号的账户名。 类型：String 父节点：Initiator，Owner
Owner	用来标识对象的拥有者。 子节点：ID，DisplayName 类型：容器 父节点：Upload
StorageClass	对象的存储类型。 类型：String 父节点：Upload
Initiated	分片上传过程启动的时间。 类型：Date 父节点：Upload
ListMultipartUploadsResult.Prefix	如果请求中包含了 Prefix 参数，则这个字段会包含 Prefix

	的值。返回的结果只包含那些 key 值以 Prefix 开头的对象。 类型：String 父节点：ListMultipartUploadsResult
Delimiter	一个 delimiter 是一个用来对关键字们进行分组的字符。所有的关键字都包含 delimiter 和 prefix 间的相同子串，prefix 之后第一个遇到 delimiter 的字符串都会加到一个叫 CommonPrefix 的组中。如果没有特别定义 prefix，所有的关键字都会被返回，但不会有 CommonPrefix。 类型：String 父节点：ListMultipartUploadsResult
CommonPrefixes	当定义 delimiter 之后，返回结果中会包含 CommonPrefixes，CommonPrefix 中包含以 prefix 开头，delimiter 结束的左右字符串组合，比如 prefix 是 note/, 同时 delimiter 是斜杠（/），结果中的 note/summer/ju 将返回 note/summer/,其余结果将按照 maxkey 要求返回。 类型：String 父节点：ListMultipartUploadsResult
CommonPrefixes.Prefix	如果请求中不包含 Prefix 参数，那么这个元素只显示那些在 delimiter 字符第一次出现之前的 key 的子字符串，且这些 key 不在响应的其它位置出现。 如果请求中包含 Prefix 参数，那么这个元素显示在 prefix 之后，到第一次出现 delimiter 之间的子串。 类型：String 父节点：CommonPrefixes

2.2.12 PUT Bucket Logging

进行添加/修改/删除 logging 的操作,如果 Bucket 已经存在了 logging,此操作会替换原有 logging。只有 Bucket 的 owner 才能执行此操作，否则会返回 403 AccessDenied 错误。

请求方法

```
response = client.put_bucket_logging(
    Bucket=BUCKET,
    BucketLoggingStatus={
        'LoggingEnabled': {
            'TargetBucket': '{0}-logging'.format(BUCKET),
            'TargetPrefix': '{0}_log'.format(BUCKET)
        }
    }
)
```

请求参数

名称	描述	是否必须
Bucket	Bucket 名称。	是
BucketLoggingStatus	请求的容器。	是
LoggingEnabled	日志信息的容器，当启动日志时，需要包含这个元素。	否
TargetBucket	指定要保存 log 的 Bucket，OOS 会向此 Bucket 存储日志。可以设置任意一个你拥有的 Bucket 作为 TargetBucket，包括启动日志的 Bucket 本身。你也可以设置将多个 Bucket 的日志存放到一个 TargetBucket 中，在这种情况下，你需要为每个源 Bucket 设置不同的 TargetPrefix，以便不同 Bucket 的 log 可以被区分出来。	否
TargetPrefix	生成的 log 文件将以此为前缀命名。	否
TriggerTargetBucket	在异地互备过程中，目标资源池的 log 会保存到	否

	此 Bucket 中，oos 会向此 Bucket 存储日志。	
TriggerTargetPrefix	在异地互备过程中，目标资源池的 log 文件将以此为前缀命名。	否
TriggerSourceBucket	在异地互备过程中，源资源池的 log 会保存到此 Bucket 中，oos 会向此 Bucket 存储日志。	否
TriggerSourcePrefix	在异地互备过程中，源资源池的 log 文件将以此为前缀命名。	否

参数格式

```
Bucket=BUCKET,
BucketLoggingStatus={
    'LoggingEnabled': {
        'TargetBucket': '{0}-logging'.format(BUCKET),
        'TargetPrefix': '{0}_log'.format(BUCKET)
    }
}
```

返回元素

成功没有返回值。

2.2.13 GET Bucket Logging

获得指定 Bucket 的 logging，只有 Bucket 的 owner 才能执行此操作，否则会返回 403 AccessDenied 错误。

请求方法

```
response = client.get_bucket_logging(
    Bucket=BUCKET
)
```

请求参数

名称	描述	是否必须
Bucket	Bucket 名称。	是

返回元素

名称	描述
BucketLoggingStatus	响应的容器。
LoggingEnabled	日志信息的容器，当启动日志时，包含这个元素；否则此元素及其子元素都不显示。
TargetBucket	保存 log 的 Bucket，OOS 会向此 Bucket 存储日志。
TargetPrefix	生成的 log 文件将以此为前缀命名。
TriggerTargetBucket	在异地互备过程中，目标资源池的 log 会保存到此 Bucket 中，oos 会向此 Bucket 存储日志。
TriggerTargetPrefix	在异地互备过程中，目标资源池的 log 文件将以此为前缀命名。
TriggerSourceBucket	在异地互备过程中，源资源池的 log 会保存到此 Bucket 中，oos 会向此 Bucket 存储日志。
TriggerSourcePrefix	在异地互备过程中，源资源池的 log 文件将以此为前缀命名。

2.2.14 Head Bucket

判断 Bucket 是否存在，而且用户是否有权限访问。如果 Bucket 存在，而且用户有权限访问时，此操作返回 200 OK。否则，返回 404 不存在，或者 403 没有权限。

请求方法

```
response = client.head_bucket(
    Bucket=BUCKET
)
```

请求参数

名称	描述	是否必须
Bucket	Bucket 名称。	是

返回元素

成功没有返回值。

2.2.15 PUT Bucket Trigger

进行添加 trigger 的操作，即添加一个向异地资源池同步的触发器。当客户端向本地资源池的 Bucket 上传对象时，OOS 可以根据配置的策略，自动将对象同步到异地资源池中。一个 Bucket 可以配置多个触发器，但只能有一个是默认的触发器。如果客户端要使用非默认的触发器上传对象，需要在 put object 时，加上参数 Trigger，值是指定的 TriggerName，目前部分资源池支持此功能，使用前请与技术支持确认。只有 Bucket 的 owner 才能执行此操作，否则会返回 403 AccessDenied 错误。

请求方法

```
response = client.put_bucket_trigger(
    Bucket=BUCKET,
    TriggerConfiguration={
        'Triggers': [
            {
                'TriggerName': 'jftrigger01',
                'IsDefault': True,
                'RemoteSite': {
                    'RemontEndPoint': 'http://oos-hz.ctyunapi.cn',
                    'ReplicaMode': '',
                    'RemoteBucketName': 'test-jf-logging',
                    'RemoteAK': '2510935f75d781385b35',
                    'RemoteSK': '567b89b544a89bb3d3ccef4be8507e6d30d75579'
                }
```

```
            }
        ]
    }
)
```

请求参数

名称	描述	是否必须
Bucket	Bucket 名称。	是
TriggerName	Trigger 的名称，是字母或数字，不能包含特殊符号，最多 20 个字符。	是
IsDefault	是否是默认的 trigger。	是
RemoteSite	异地资源池的相关配置，可以配置向多个异地资源池同步数据，每个异地资源池对应一个 RemoteSite。如果不配置 RemoteSite，即不复制到其他资源池。	否
RemontEndPoint	异地资源池的 endpoint。	是
ReplicaMode	同步到异地资源池的副本模式，如果不填写，会使用动态副本模式。	否
RemoteBucketName	异地资源池的 bucketName。	是
RemoteAK	异地资源池的 AccessKey。	是
RemoteSK	异地资源池的 secretKey。	是

参数格式

```
Bucket=BUCKET,
TriggerConfiguration={
    'Triggers': [
        {
            'TriggerName': 'trigger01',
            'IsDefault': True,
            'RemoteSite': {
                'RemontEndPoint': 'http://oos-hz.ctyunapi.cn',
                'ReplicaMode': '',
```

```
                'RemoteBucketName': 'test-jf-logging',
                'RemoteAK': '2510935f75d781385b35',
                'RemoteSK': '567b89b544a89bb3d3ccef4be8507e6d30d75579'
            }
        }
    ]
}
```

返回元素

成功没有返回值。

2.2.16 GET BucketTrigger

查询某 Bucket 中配置的所有 trigger。只有 Bucket 的 owner 才能执行此操作，否则会返回 403 AccessDenied 错误。

请求方法

```
response = client.get_bucket_trigger(
    Bucket=BUCKET
)
```

请求参数

名称	描述	是否必须
Bucket	Bucket 名称。	是

返回元素

名称	描述	是否必须
TriggerName	Trigger 的名称。	是
IsDefault	是否是默认的 trigger。	是
RemoteSite	异地资源池的相关配置。	否

RemontEndPoint	异地资源池的 endpoint。	是
ReplicaMode	同步到异地资源池的副本模式。	否
RemoteBucketName	异地资源池的 bucketName。	是
RemoteAK	异地资源池的 AccessKey。	是
RemoteSK	异地资源池的 SecretKey。	是

2.2.17 DELETE Bucket Trigger

删除某 Bucket 中配置的所有 trigger，只有 Bucket 的 owner 才能执行此操作，否则会返回 403 AccessDenied 错误。

请求方法

```
response = client.delete_bucket_trigger(
    Bucket=BUCKET,
    TriggerName='trigger01'
)
```

请求参数

名称	描述	是否必须
Bucket	Bucket 名称。	是
TriggerName	Trigger 的名称。	是

返回元素

成功没有返回值。

2.2.18 PUT Bucket Lifecycle

Put Bucket Lifecycle 接口用于设置 Bucket 的生命周期，如果生命周期的配置已经存在，将会被替换。用户可以通过设置生命周期，来让 OOS 删除过期的对象。

请求方法

```
response = client.put_bucket_lifecycle(
    Bucket=BUCKET,
    LifecycleConfiguration={
        'Rules': [
            {
                'Expiration': {
                    'Days': 100
                },
                'ID': 'lifecycle-123',
                'Prefix': 'test',
                'Status': 'Enabled'
            }
        ]
    }
)
```

请求参数

名称	描述	是否必须
Bucket	Bucket 名称。	是
LifecycleConfiguration	容器，最多包含 100 个规则。 类型：容器 子节点：Rule 父节点：无	是
Rule	生命周期规则的容器。 类型：object 类型 Array 父节点：LifecycleConfiguration	是
ID	规则的唯一标识，最长 255 个字符。 类型：字符串	否

	父节点：Rule	
Prefix	指明要使用规则的对象前缀，最长 1024 个字符。 类型：字符串 父节点：Filter	是
Status	如果是 Enabled，那么规则立即生效。如果是 Disabled，那么规则不会生效。 类型：字符串 父节点：Rule	是
Expiration	描述过期动作的容器。 类型：容器 子节点：Days 父节点：Rule	是
Days	以天数来描述生命周期，值是正整数。 类型：整数 父节点：Expiration	Days 和 Date 二选一
Date	生成时间早于此时间的对象将被认为是过期对象。 日期必需服从 ISO8601 的格式，并且总是 UTC 的零点。 例如：2002-10-11T00:00:00.000Z。 类型：String 父节点：Expiration	Days 和 Date 二选一

参数格式

```
Bucket=BUCKET,
LifecycleConfiguration={
    'Rules': [
        {
            'Expiration': {
                'Days': 100
            },
            'ID': 'lifecycle-123',
```

```
            'Prefix': 'test',
            'Status': 'Enabled'
        }
    ]
}
```

返回元素

成功没有返回值。

2.2.19 GET Bucket Lifecycle

此接口用于返回配置的 Bucket 生命周期。

请求方法

```
response = client.get_bucket_lifecycle(
    Bucket=BUCKET
)
```

请求参数

名称	描述	是否必须
Bucket	Bucket 名称。	是

返回元素

名称	描述
LifecycleConfiguration	容器，最多包含 100 个规则。 类型：容器 子节点：Rule 父节点：无
Rule	生命周期规则的容器。 类型：容器 父节点：LifecycleConfiguration

ID	规则的唯一标示，最长 255 个字符。 类型：字符串 父节点：Rule
Prefix	指明要使用规则的对象前缀。 类型：字符串 父节点：Rule
Status	如果是 Enabled，那么规则立即生效。如果是 Disabled，那么规则不会生效。 类型：字符串 父节点：Rule
Expiration	描述过期动作的容器。 类型：容器 子节点：Days 父节点：Rule
Days	以天数来描述生命周期，值是正整数。 类型：整数 父节点：Expiration
Date	生成时间早于此时间的对象将被认为是过期对象 日期必需服从 ISO8601 的格式，并且总是 UTC 的零点。例如：2002-10-11T00:00:00.000Z。 类型：String 父节点：Expiration

2.2.20 DELETE Bucket Lifecycle

删除配置的 Bucket 生命周期，OOS 将会删除指定 Bucket 的所有生命周期配置规则。用户的对象将永远不会到期，OOS 也不会再自动删除对象

请求方法

```
response = client.get_bucket_lifecycle(
    Bucket=BUCKET
)
```

请求参数

名称	描述	是否必须
Bucket	Bucket 名称。	是

返回元素

成功没有返回值。

2.3 关于 Object 的操作

2.3.1 PUT Object

Put 操作用来向指定 Bucket 中添加一个对象，要求发送请求者对该 Bucket 有写权限，用户必须添加完整的对象。

请求方法

```
with open(UPLOAD_FILE, 'rb') as data:
    client.put_object(
        Bucket=BUCKET,
        Key=KEY,
        Body=data,
        StorageClass='STANDARD'
    )
```

请求参数

名称	描述	是否必须
Bucket	Bucket 名称。	是
Body	File 内容。	是
Key	文件名。	是
CacheControl	按照请求/回应的方式用来定义缓存行为。	否
ContentDisposition	指出对象的描述性的信息。	否
ContentEncoding	指出对象所使用的编码格式。	否
ContentMD5	按照 RFC 1864，使用 base64 编码格式生成信息的 128 位 MD5 值。	否
ContentType	标准的 MIME 类型用来描述内容格式。	否
Expires	new Date，对象不再被缓存的时间。 类型：String	否
Metadata	任何头以这个前缀开始都会被认为是用户的元数据，当用户检索时，它将会和对象一起被存储并返回。PUT 请	否

	求头大小限制为 8KB。在 PUT 请求头中，用户定义的元数据大小限制为 2KB。	
StorageClass	数据的存储类型，默认采用动态副本模式存储。用户也可选择使用标准模式进行存储。对于那些不太重要，可以重复生成的数据，用户可以选择减少冗余策略来降低成本。 类型: String 默认值: STANDARD 可选值: STANDARD。	否

参数格式

```
Bucket=BUCKET,
Key=KEY,
Body=data,
StorageClass='STANDARD'
```

返回元素

成功没有返回值。

2.3.2 GET Object

GET 操作用来检索在 OOS 中的对象信息，执行 GET 操作，用户必须对 object 所在的 Bucket 有读权限。如果 Bucket 是 public read 的权限，匿名用户也可以通过非授权的方式进行读操作。

请求方法

```
response = client.get_object(
    Bucket=BUCKET,
    Key=KEY
)
body = response['Body']
```

```
with open(DOWNLOAD_FILE_PATH, 'wb') as fd:
    for chunk in iter(lambda: body.read(4096), b''):
        fd.write(chunk)
```

请求参数

名称	描述	是否必须
Bucket	Bucket 名称。	是
Key	文件名。	是

返回元素

名称	描述
Body	OOS 中的对象信息。
ContentType	对象类型。
LastModified	最后更改时间。
ContentLength	对象大小。
Metadata	对象的元数据。

2.3.3 Head Object

HEAD 操作返回对象的属性信息，但不会返回对象的数据内容。当用户只想 获取对象的属性信息时，可以调用此接口。执行 HEAD 操作，用户必须对 object 所在的 Bucket 有读权限。如果 Bucket 是 public read 的权限，匿名用户也可以通 过非授权的方式进行读操作。

请求方法

```
response = client.head_object(
    Bucket=BUCKET,
    Key=KEY
)
```

请求参数

名称	描述	是否必须
Bucket	Bucket 名称。	是
Key	文件名。	是

返回元素

名称	描述
ContentLength	对象大小。
ContentType	对象类型。
ETag	通过 MD5 的方式计算出标签，ETag 主要用来反映对象内容的信息发生了改变，并不反映元数据的变化。
Expiration	对象到期时间。
LastModified	最后更改时间。
Metadata	对象的元数据。

2.3.4 DELETE Object

Delete 操作移除指定的对象，要求用户要对对象所在的 Bucket 拥有写权限。

请求方法

```
response = client.delete_object(
    Bucket=BUCKET,
    Key=KEY
)
```

请求参数

名称	描述	是否必须
Bucket	Bucket 名称。	是
Key	文件名。	是

返回元素

成功没有返回值。

2.3.5 PUT Object - Copy

通过 PUT 操作创建一个存储在 OOS 里的对象的拷贝。PUT 操作类似于执行一个 GET 然后在执行一次 PUT。增加参数 CopySource，使用 PUT 操作将源对象存入指定 Bucket。要执行拷贝请求，用户需要对源对象有读权限，对目标 Bucket 有写权限。

请求方法

```
response = client.copy_object(
    Bucket=BUCKET,
    CopySource={'Bucket': BUCKET, 'Key': KEY},
    Key='{0}-Copy'.format(KEY)
)
```

请求参数

名称	描述	是否必须
Bucket	目标 Bucket 名称。	是
CopySource	复制的源 Bucket/key。	是
Key	目标文件名。	是
CacheControl	按照请求/回应的方式用来定义缓存行为。	否
ContentDisposition	指出对象的描述性的信息。	否
ContentEncoding	指出对象所使用的编码格式。	否
ContentMD5	按照 RFC 1864，使用 base64 编码格式生成信息的 128 位 MD5 值。	否
ContentType	标准的 MIME 类型用来描述内容格式。	否
Expires	new Date，对象不再被缓存的时间。	否

	类型：String	
Metadata	任何头以这个前缀开始都会被认为是用户的元数据，当用户检索时，它将会和对象一起被存储并返回。PUT 请求头大小限制为 8KB。在 PUT 请求头中，用户定义的元数据大小限制为 2KB。	否
StorageClass	数据的存储类型，默认采用动态副本模式存储。用户也可选择使用标准模式进行存储。对于那些不太重要，可以重复生成的数据，用户可以选择减少冗余策略来降低成本。 类型: String 默认值: STANDARD 可选值: STANDARD。	否

返回元素

成功没有返回值。

2.3.6 Initial Multipart Upload

本接口初始化一个分片上传（Multipart Upload）操作，并返回一个上传 ID，此 ID 用来将此次分片上传操作中上传的所有片段合并成一个对象。用户在执行每一次子上传请求（见 Upload Part）时都应该指定该 ID。用户也可以在表示整个分片上传完成的最后一个请求中指定该 ID。或者在用户放弃该分片上传操作时指定该 ID。

请求方法

```
upload = client.create_multipart_upload(
    Bucket=BUCKET,
    Key=MULTIPART_UPLOAD_KEY
)
```

请求参数

名称	描述	是否必须
Bucket	Bucket 名称。	是
Key	文件名。	是
CacheControl	按照请求/回应的方式用来定义缓存行为。	否
ContentDisposition	指出对象的描述性的信息。	否
ContentEncoding	指出对象所使用的编码格式。	否
ContentType	标准的 MIME 类型用来描述内容格式。	否
Expires	new Date，对象不再被缓存的时间。 类型：String	否
Metadata	任何头以这个前缀开始都会被认为是用户的元数据，当用户检索时，它将会和对象一起被存储并返回。PUT 请求头大小限制为 8KB。在 PUT 请求头中，用户定义的元数据大小限制为 2KB。	否
StorageClass	数据的存储类型，默认采用动态副本模式存储。用户也可选择使用标准模式进行存储。对于那些不太重要，可以重复生成的数据，用户可以选择减少冗余策略来降低成本。 类型: String 默认值: STANDARD 可选值: STANDARD。	否

返回元素

名称	描述
InitiateMultipartUploadResult	包含所有返回元素的容器。 类型：容器 子节点：Bucket，Key，UploadId 父节点：无
Bucket	分片上传对应的 Bucket 的名称。

	类型：Sring 父节点：InitiateMultipartUploadResult
Key	分片上传对应的对象名称。 类型：String 父节点：InitiateMultipartUploadResult
UploadId	分片上传 ID。 类型：String 父节点：InitiateMultipartUploadResult

2.3.7 Upload Part

该接口用于实现分片上传操作中片段的上传。

在上传任何一个分片之前，必须执行 Initial Multipart Upload 操作来初始化分片上传操作，初始化成功后，OOS 会返回一个上传 ID，这是一个唯一的标识，用户必须在调用 Upload Part 接口时加入该 ID。

分片号 PartNumber 可以唯一标识一个片段并且定义该分片在对象中的位置，范围从 1 到 10000。如果用户用之前上传过的片段的分片号来上传新的分片，之前的分片将会被覆盖。

除了最后一个分片外，所有分片的大小都应该不小于 5M，最后一个分片的大小不受限制。

为了确保数据不会由于网络传输而毁坏，需要在每个分片上传请求中指定 Content-MD5 头，OOS 通过提供的 Content-MD5 值来检查数据的完整性，如果不匹配，则会返回一个错误信息。

请求方法

```
part = client.upload_part(
    Body=chunk,
    Bucket=BUCKET,
    Key=MULTIPART_UPLOAD_KEY,
    PartNumber=num,
    UploadId=upload['UploadId']
)
```

请求参数

名称	描述	是否必须
Bucket	分片所属的 Bucket 名称。	是
Key	分片的名称。	是
PartNumber	分片的 id。	是
UploadId	初始化分片后返回的 id。	是
Body	File 数据。	否
ContentLength	该分片的大小，以字节为单位。 类型：Integer 默认值：None	否
ContentMD5	该分片数据的 128 位采用 base64 编码的 MD5 值。这个头可以用来验证该分片数据是否与原始数据值保持一致。尽管这个值是可选的，我们仍然推荐使用 Content-MD5 机制来执行端到端的一致性校验。 类型：String 默认值：None	否
Expect	如果用户的应用中设置该头为 100-continue，则应用在接收到请求回应之前不会发送请求实体。如果基于头的消息被拒绝，消息的实体也不会被发送。 类型：String 默认值：None 可选值：100-continue	否

返回元素

成功没有返回值。

2.3.8 Complete Multipart Upload

该接口通过合并之前的上传片段来完成一次分片上传过程。

请求方法

```
complete = client.complete_multipart_upload(
    Bucket=BUCKET,
    Key=MULTIPART_UPLOAD_KEY,
    UploadId=upload['UploadId']
)
```

请求参数

名称	描述	是否必须
Bucket	分片 object 所属 Bucket 名称。	是
Key	分片的 object 名称。	是
UploadId	初始化时返回的 uploadID。	是
MultipartUpload	请求的容器。 类型：object 子节点：1 个或多个 Part 元素。	否
Parts	描述 part 的信息。 父容器：MultipartUpload 类型 object 类型数组 包括：ETag: 上传每个分片时返回的 ETag（实体标签）。 PartNumber: 上传的每个分片对应的编号。	否

返回元素

名称	描述
CompleteMultipartUploadResult	包含整个响应的容器。 类型：容器 子节点：Location, Bucket, Key, ETag

	父节点：无
Location	新创建的对象的 URL 地址。 类型：URI 父节点：CompleteMultipartUploadResult
Bucket	分片上传对应的对象容器。 类型：String 父节点：CompleteMultipartUploadResult
Key	新创建的对象的 Key。 类型：String 父节点：CompleteMultipartUploadResult
ETag	Tag 用来标识新创建的对象数据，拥有不同数据的对象，它的 Tag 值也不同。ETag 值是一个不透明的字符串，它可以是也可以不是一个对象数据的 MD5 值，如果 ETag 值不是一个对象的 MD5 值，它将会包含一个或者多个非十六进制字符串，并且/或者包含少于 32 位/多于 32 位的十六进制数字。 类型：String 父节点：CompleteMultipartUploadResult

2.3.9 Abort Multipart Upload

该接口用于终止一次分片上传操作。分片上传操作被终止后，用户不能再通过上传 ID 上传其它片段，之前已上传完成的片段所占用的存储空间将被释放。

请求方法

```
response = client.abort_multipart_upload(
    Bucket=BUCKET,
    Key=MULTIPART_UPLOAD_KEY,
```

```
    UploadId='1544063802527236705'
)
```

请求参数

名称	描述	是否必须
Bucket	分片 object 所属 Bucket 名称。	是
Key	分片的 object 名称。	是
UploadId	初始化时返回的 uploadID。	是

返回元素

成功没有返回值。

2.3.10 List Parts

该操作用于列出一次分片上传过程中已经上传完成的所有片段。

请求方法

```
response = client.list_parts(
    Bucket=BUCKET,
    Key=MULTIPART_UPLOAD_KEY,
    MaxParts=10,
    UploadId='1543479591653868846'
)
```

请求参数

名称	描述	是否必须
Bucket	分片 object 所属 Bucket 名称。	是
Key	分片的 object 名称。	是
UploadId	初始化时返回的 uploadID。	是
MaxParts	Integer 设置返回的最大的分片数量。	否

PartNumberMarker	Intege 设定此值，只会返回分片号大于此值得分片。	否

返回元素

名称	描述
ListPartsResult	包含整个响应的容器。 类型：容器 子节点：Bucket, Key, UploadId, Initiator, Owner, StorageClass,PartNumberMarker, NextPartNumberMarker, MaxParts,IsTruncated, Part 父节点：无
Bucket	分片上传对应的对象名称。 类型：String 父节点：ListPartsResult
Key	新创建的对象的 Key。 类型：String 父节点：ListPartsResult
UploadId	分片上传 ID。 类型：String 父节点：ListPartsResult
Initiator	指定执行此次分片上传过程的用户账号。 子节点：ID，DisplayName 类型：容器 父节点：ListPartsResult
ID	OOS 账号的 ID 号。 类型：String 父节点：Initiator
DisplayName	OOS 账号的账户名。 类型：String

	父节点：Initiator
Owner	用来标识对象的拥有者。 子节点：ID，DisplayName 类型：容器 父节点：ListPartsResult
StorageClass	对象的存储类型。 类型：String 父节点：ListPartsResult
PartNumberMarker	列表起始位置的片段的分片号。 类型：Integer 父节点：ListPartsResult
NextPartNumberMarker	当此次请求没有将所有片段列举完时，此元素指定列表中的最后一个片段的分片号。此分片号用于作为下一次连续列表请求的 part-number-marker 参数的值。 类型：Integer 父节点：ListPartsResult
MaxParts	响应中片段的最大数目。 类型：Integer 父节点：ListPartsResult
IsTruncated	标识此次分片上传过程中的所有片段是否全部被列出，如果为 true 则表示没有全部列出。如果分片上传过程的片段数超过了 MaxParts 元素指定的最大数，则会导致一次列表请求无法将所有片段数列出。 类型：Boolean 父节点：ListPartsResult
Part	与某个片段对应的容器，响应中可能包含 0 个或多个 Part 元素。 子节点：PartNumber, LastModified, ETag, Size

	类型：String 父节点：ListPartsResult
PartNumber	标识片段的分片号。 类型：Integer 父节点：Part
LastModified	片段上传完成的日期。 类型：Date 父节点：Part
ETag	片段上传完成时返回的 ETag 值。 类型：String 父节点：Part
Size	片段的数据大小。 类型：Integer 父节点：Part

2.3.11 Copy Part

可以将已经存在的 object 作为分段上传的片段，拷贝生成一个新的片段。需要通过参数 CopySource 来定义拷贝源。如果想拷贝源 object 中的一部分，可以加参数 CopySourceRange。

请求方法

```
part = client.upload_part_copy(
    Bucket=BUCKET,
    CopySource={'Bucket': BUCKET, 'Key': KEY},
    #CopySourceRange='bytes=0-9',
    Key=MULTIPART_UPLOAD_KEY,
    PartNumber=1,
    UploadId=upload['UploadId']
)
```

请求参数

名称	描述	是否必须
Bucket	分片 object 所属 Bucket 名称。	是
Key	分片的 object 名称。	是
PartNumber	分片的编号。	是
UploadId	初始化时返回的 uploadID。	是
CopySource	已经存在的 object 地址。	是
CopySourceRange	要从源 object 拷贝的 bytes 范围。Rage 值的格式是 bytes=第一个字节-最后一个字节。第一个字节从 0 开始。例如要拷贝前 10 个字节，bytes=0-9。只允许对大于 5G 的源 object 进行部分拷贝的操作。如果要拷贝整个 object，不需要这个头。 类型：String 默认值：None	否

返回元素

名称	描述
CopyPartResult	包含整个响应的容器。 类型：容器 父节点：无
ETag	新分片的 ETag。 类型：String 父节点：CopyPartResult
LastModified	分片的最后修改时间。 类型：String 父节点：CopyPartResult

2.3.12 DELETE Multiple Objects

支持用一个 HTTP 请求删除一个 Bucket 中的多个 object。如果你知道你想删除的 object 名字，此功能可以批量删除这些 object，而不用发送多个单独的删除请求。

请求方法

```
response = client.delete_objects(
    Bucket=BUCKET,
    Delete={
        'Objects': [
            {
                'Key': 'test'
            },
            {
                'Key': 'test-Copy'
            }
        ]
    }
)
```

请求参数

名称	描述	是否必须
Bucket	需要删除 objects 的 Bucket 名称。	是
Delete	包含整个请求的容器。 类型：容器 父节点：无	是
Quiet	使用简明信息模式来返回响应，当使用此元素时，需要指定 true。 类型：Boolean 父节点：Delete	否
Object	包含被删除 object 的容器。	是

	类型：容器 父节点：Delete	
Key	被删除 object 名。 类型：String 父节点：Object	是

返回元素

名称	描述
DeleteResult	包含整个响应的容器。 类型：容器 父节点：无
Deleted	成功删除的容器，包含成功删除的 object。 类型：容器 父节点：DeleteResult
Key	尝试删除的 object 名。 类型：String 父节点：Deleted，或 Error
Error	删除失败的容器，包含删除失败的 object 信息。 类型：容器 父节点：DeleteResult
Code	删除失败的状态码。 类型：String 父节点：Error 值：AccessDenied, InternalError
Message	错误的描述。 类型：String 父节点：Error

2.4 关于 AccessKey 的操作

由于 AccessKey 的接口采用 Form-Data 的类型，所有参数均需转换为 http body 后传入。

2.4.1 CreateAccessKey

创建一对普通的 AccessKey 和 SecretKey，默认的状态是 Active。只有主 key 才能执行此操作。

为保证账户的安全，SecretKey 只在创建的时候会被显示。请把 key 保存起 来，比如保存到一个文本文件中。如果 SecretKey 丢失了，你可以删除 AccessKey， 并创建一对新的 key。默认情况下，每个账号最多创建 10 个 AccessKey。

请求方法

```
data = {
    "Action": "CreateAccessKey"
}
body = encode_params(data)
response = iam_client.create_access_key(
    Body=body
)
```

请求参数

名称	描述	是否必须
Action	值是 CreateAccessKey。	是

返回元素

名称	描述
UserName	用户名称。
AccessKeyId	生成的 AccessKey。
Status	默认生成的是 Active 状态。
SecretAccessKey	生成的 SecretKey。
IsPrimary	是否是主 key，默认生成的都是普通 key，值是 false。

2.4.2 DeleteAccessKey

删除一对普通的 AccessKey 和 SecretKey。只有主 key 才能执行此操作。

请求方法

```
data = {
    "Action": "DeleteAccessKey",
    "AccessKeyId": "10c2758180defd92e034"
}
body = encode_params(data)
response = iam_client.delete_access_key(
    Body=body
)
```

请求参数

名称	描述	是否必须
Action	值是 DeleteAccessKey。	是
AccessKeyId	要删除的 AccessKey。	是

返回元素

成功没有返回值。

2.4.3 UpdateAccessKey

更新普通的 AccessKey 的状态，或将普通 key 设置成为主 key，反之亦然。 只有主 key 才能执行此操作。

请求方法

```
data = {
    "Action": "UpdateAccessKey",
    "AccessKeyId": "10c2758180defd92e034",
```

```
    "Status": "Inactive",          # Active or Inactive
    "IsPrimary": False
}
body = encode_params(data)
response = iam_client.update_access_key(
    Body=body
)
```

请求参数

名称	描述	是否必须
Action	值是 UpdateAccessKey。	是
AccessKeyId	要更新的 AccessKey。	是
Status	Key 的状态，启用或禁止。值是 Active, Inactive。	是
Isprimary	是否是主 key，值是 true，false。	是

返回元素

成功没有返回值。

2.4.4 ListAccessKey

列出账号下的主 key 和普通 key。只有主 key 才能执行此操作。可以通过 MaxItems 参数指定返回的结果数量，默认返回 100 个 key。可以通过 Marker 参数设置返回的起始位置，该参数可以从前一次请求的响应体中获得。为保证账号 安全，list 操作时，不会返回 SecretKey。

请求方法

```
data = {
    'Action': 'ListAccessKey',
    'MaxItems': 10
}
body = encode_params(data)
response = iam_client.list_access_key(
    Body=body
```

```
)
```

请求参数

名称	描述	是否必须
Action	值是 ListAccessKey。	是
MaxItems	设置返回结果集的最大数量，如果实际的 key 的数量超 过了 MaxItem，那么在响应结果中，IsTruncated 的值是 true。	否
Marker	可以通过 Marker 参数设置返回的起始位置，该参数可 以从前一次请求的响应体中获得。	否

返回元素

名称	描述
AccessKeyMetadata	Access Key 元数据的 list。
IsTruncated	返回的结果是否被截断，如果是 true，表示还有结果 没有返回，可以通过 Marker 参数，继续请求，以便 获得后续的数据。
Marker	如果 IsTruncated 是 true，那么会返回 Marker，可以用做下次请求的参数。