S3 Python SDK 文档

SDK说明

对象存储 Python SDK 使用开源的S3 Python SDK boto3。本文档介绍用户如何使用boto3 来使用对象存储服务。更加详细的接口参数说明，请在使用时参照boto3 API官方说明boto3。

安装 Python S3 SDK

- 通过pip安装， pip install boto3
- 通过源码安装。
  git clone https://github.com/boto/boto3.git && cd boto3 && sudo python setup.py install

卸载 Python S3 SDK

```
pip uninstall boto3
```

快速入门

> 确认您已经理解 对象存储 基本概念，如 `Bucket`、`Object`、`Endpoint`、`AccessKey` 和 `SecretKey` 等。

下面介绍如何使用Python S3 boto3 SDK来访问对象存储服务，包括查看Bucket列表，上传文件，下载文件，查看文件列表等。 默认这些程序是写在以py后缀的脚本文件里，通过Python程序可以执行。

> 注意: 请不要用生产Bucket试验本文档中的例子

查看Bucket列表

```
from boto3.session import Session
import boto3
#Client初始化
access_key = "您的AccessKey"
secret_key = "您的SecretKey"
url = "您的Endpoint" #例如: 'http://IP:PORT' 或者 'http://eos-beijing-
1.cmecloud.cn' 或者 'http://eos-beijing-2.cmecloud.cn'
session = Session(access_key, secret_key)
s3_client = session.client('s3', endpoint_url=url)
#Client初始化结束
#列出该用户拥有的桶
print [bucket['Name'] for bucket in s3_client.list_buckets()['Buckets']]
```

- 代码中，access_key 与 secret_key 是在对象存储服务中创建用户后，在页面上申请到的给用户用于访问对象存储的认证信息。而url是对象存储服务提供的服务地址及端口。
  在初始化之后，就可以利用s3和s3_client进行各种操作了。
- Session对象承载了用户的认证信息
- session.client()对象用于服务相关的操作，目前就是用来列举Bucket;
- s3_client.list_buckets()['Buckets']对象是一个可以遍历用户Bucket信息的迭代器

> 获取用户所拥有的bucket列表。用户必须有有效的Access Key。匿名请求将不允许此操作。

新建Bucket

创建桶。桶用于对象存储用户上传的对象。桶的名字必须是唯一的，如果bucket已经被其他用户使用，则会创建失败。

```
s3_client.create_bucket(Bucket="您的bucket名") #默认是私有的桶
#创建公开可读的桶
s3_client.create_bucket(Bucket="您的bucket名", ACL='public-read')
#ACL有如下几种"private","public-read","public-read-write","authenticated-read"
```

上传文件

把当前目录下的local.txt文件上传到对象存储服务器

```
resp = s3_client.put_object(Bucket="您的已经存在且配额可用的bucket名", Key="您上传文件
后对象存储服务器端处存储的名字", Body=open("local.txt", 'rb').read())
```

下载文件

把对象存储服务器上的Object下载到本地文件`local-backup.bin`:

```
resp = s3_client.get_object(Bucket="您的已经存在的bucket名", Key="您该bucket中的对象
名")
with open('local-backup.bin','wb') as f:
    f.write(resp['Body'].read())
```

> 如果桶开启了多版本，s3_client.get_object需要参数`VersionId='对应的版本号'`

列举文件

列举Bucket下的文件：

```
resp = s3_client.list_objects(Bucket="您的已经存在的bucket名")
for obj in resp['Contents']:
    print obj['Key']
```

删除文件

删除对象存储服务器端桶下的对象

```
resp = s3_client.delete_object(Bucket="您的已经存在的bucket名", Key="您要删除的对象名")
```

初始化

Python SDK几乎所有的操作都是通过session.client()进行的。这里，我们会详细说明如何初始化上述类。

确定EndPoint

请先阅读理解对象存储中的AccessKey，SecretKey，Endpoint相关的概念。
Endpoint是对象存储的服务地址，详细信息请参考https://ecloud.10086.cn/op-help-center/show/6CCB02D26C4AEA8B：

创建桶的时候指定桶要创建在的区域，比如要创建的桶要在beijing2数据中心的话，桶的请求的Endpoint要写`http://eos-beijing-2.cmecloud.cn`,CreateBucketConfiguration= {'LocationConstraint': 'beijing2'},
设置桶的属性如`多版本，ACL，静态网站，防盗链，上传数据到桶，下载桶内对象`的时候，桶在`哪个数据中心`，请求的`Endpoint就写哪个数据中心`。不管在哪个数据中心，用户都能获得用户的桶列表（list_buckets）和桶所在的数据中心(get_bucket_location)

下面的代码设置访问数据中心beijing2：

```
from boto3.session import Session
import boto3
access_key = "您的AccessKey"
secret_key = "您的SecretKey"
url = "http://eos-beijing-2.cmecloud.cn"
session = Session(access_key, secret_key)
s3_client = session.client('s3', endpoint_url=url)
```

管理存储空间

存储空间（Bucket）是对象存储服务器上的命名空间，也是计费、权限控制、日志记录等高级功能的管理实体。

查看所有Bucket

s3_client.list_buckets()['Buckets'] 是可以遍历所有的Bucket的对象：

```
from boto3.session import Session
import boto3
session = Session("您的AccessKey", "您的SecretKey")
s3_client = session.client('s3', endpoint_url="数据中心Endpoint")
print [bucket['Name'] for bucket in s3_client.list_buckets()['Buckets']]
```

创建Bucket

```
s3_client.create_bucket(Bucket="您的bucket名") #默认是私有的桶

#s3_client.create_bucket(Bucket="您的bucket名",CreateBucketConfiguration=
{'LocationConstraint': 'beijing2'}) #创建在数据中心beijing2的私有桶，注意发送请求前需要
修改Endpoint为beijing2数据中心的

#创建公开可读的桶
s3_client.create_bucket(Bucket="您的bucket名", ACL='public-read')

#ACL有如下几种"private","public-read","public-read-write","authenticated-read"
```

查询桶所在的数据中心

该请求可以发送到数据中心beijing1和beijing2都可以查询到桶所在数据中心的信息

```
from boto3.session import Session
import boto3
session = Session("您的AccessKey", "您的SecretKey")
s3_client = session.client('s3', endpoint_url="数据中心beijing1或者beijing2的
Endpoint")
print s3_client.get_bucket_location(Bucket="您的bucket名")['LocationConstraint']
```

删除Bucket

删除指定的桶。只有该桶中所有的对象被删除了，该桶才能被删除。另外，只有该桶的拥有者才能删除它，与桶的访问控制权限无关。

```
from botocore.exceptions import ClientError
try:
    resp = s3_client.delete_bucket(Bucket="要删除的桶名")
except ClientError as e:
    print e.response['Error']['Code']
```

查看Bucket访问权限

```
resp = s3_client.get_bucket_acl(Bucket="要查询访问权限的桶")
print resp['Grants']
```

设置Bucket访问权限

```
resp = s3_client.put_bucket_acl(Bucket="要设置访问权限的桶", ACL='public-read')
```

> ACL有如下几种"private","public-read","public-read-write","authenticated-read"

上传文件

对象存储有多种上传方式，不同的上传方式能够上传的数据大小也不一样。普通上传（PutObject）最多只能上传小于或等于5GB的文件；而分片上传（MultipartUpload）每个分片可以达到5GB，合并后的文件能够达到5TB。

首先介绍普通上传，我们会详细展示提供数据的各种方式，即方法中的 data 参数。其他上传接口有类似的data参数，不再赘述。

上传字符串

上传内存中的字符串：

```
resp = s3_client.put_object(Bucket="您的已经存在且配额可用的bucket名", Key="您上传文件后对象存储服务器端处存储的名字", Body="上传的字符串内容", StorageClass='STANDARD')
#如果指定StorageClass='STANDARD_IA'，则上传的文件为低频类型
```

也可以指定上传的是bytes：

```
resp = s3_client.put_object(Bucket="您的已经存在且配额可用的bucket名", Key="您上传文件后对象存储服务器端处存储的名字", Body=b"上传的bytes", StorageClass='STANDARD')
#如果指定StorageClass='STANDARD_IA'，则上传的文件为低频类型
```

或是指定为unicode：

```
resp = s3_client.put_object(Bucket="您的已经存在且配额可用的bucket名", Key="您上传文件后对象存储服务器端处存储的名字", Body=u"上传的unicode", StorageClass='STANDARD')
#如果指定StorageClass='STANDARD_IA'，则上传的文件为低频类型
```

上传本地文件

把当前目录下的local.txt文件上传到对象存储服务器

```
resp = s3_client.put_object(Bucket="您的已经存在且配额可用的bucket名", Key="您上传文件后对象存储服务器端处存储的名字", Body=open("local.txt", 'rb').read(),
StorageClass='STANDARD')
#如果指定StorageClass='STANDARD_IA'，则上传的文件为低频类型
```

断点续传

当需要上传的本地文件很大，或网络状况不够理想，往往会出现上传到中途就失败了。此时，如果对已经上传的数据重新上传，既浪费时间，又占用了网络资源。Python SDK提供了分片上传接口，用于断点续传本地文件或者并发上传文件不同的区域块来加速上传。

分片上传

采用分片上传，用户可以对上传做更为精细的控制。这适用于诸如预先不知道文件大小、并发上传、自定义断点续传等场景。一次分片上传可以分为三个步骤：

1. 初始化（createMultipartUpload）：获得Upload ID
2. 上传分片（uploadPart）：这一步可以并发进行
3. 完成上传（completeMultipartUpload）：合并分片，生成文件

```
mpu = s3_client.create_multipart_upload(Bucket="您的已经存在且配额可用的bucket
名",Key="您上传文件后对象存储服务器端处存储的名字", StorageClass='STANDARD')#step1.初始
化
#如果指定StorageClass='STANDARD_IA'，则上传的对象类型为低频
part_info = {
    'Parts': []
}
i = 1
while 1:
    data = file.read(10 * 1024 * 1024)#每个分块10MiB大小，可调整
    if data == b'':
        break
    response = s3_client.upload_part(Bucket="您的已经存在且配额可用的bucket名",
Key="您上传文件后对象存储服务器端处存储的名字", PartNumber=i,
UploadId=mpu["UploadId"],Body=data)#step2.上传分片 #可改用多线程
    part_info['Parts'].append({
        'PartNumber': i,
        'ETag': response['ETag']
    })
    i += 1
s3_client.complete_multipart_upload(Bucket="您的已经存在且配额可用的bucket名",Key="您
上传文件后对象存储服务器端处存储的名
字",UploadId=mpu["UploadId"],MultipartUpload=part_info)#step3.完成上传
```

使用分段上传 API 复制对象[6.2版本开始支持]

直接拷贝source桶下的大文件10GiB到dest桶下,dest桶下命名为copyed-10GiB

```
# -*- coding: utf-8 -*-
import math
from boto3.session import Session

access_key = "您的AccessKey"
secret_key = "您的SecretKey"
url = "您的Endpoint"
session = Session(access_key, secret_key)
s3_client = session.client('s3', endpoint_url=url)

objectSize = s3_client.head_object(Bucket="source", Key="10GiB")
['ContentLength']
mpu = s3_client.create_multipart_upload(Bucket="dest", Key="copyed-10GiB",
StorageClass='STANDARD')
#如果指定StorageClass='STANDARD_IA'，则上传的对象类型为低频
psize = 10 * 1024 * 1024 #每个分片大小为10MiB
part_info = {
    'Parts': []
}
bytePosition = 0
i = 1
while bytePosition < objectSize:
    lastbyte = bytePosition + psize - 1
    if lastbyte >= objectSize:
        lastbyte = objectSize - 1
    print "mp.copy_part_from_key part %d (%d %d)" % (i, bytePosition, lastbyte)
    res = s3_client.upload_part_copy(Bucket="dest", CopySource="source/10GiB",
                                     CopySourceRange='bytes=%d-%d'%
(bytePosition,lastbyte),
```

```
                                    Key="copyed-10GiB",
                                    PartNumber=i, UploadId=mpu["UploadId"])
    part_info['Parts'].append({
        'PartNumber': i,
        'ETag': res['CopyPartResult']['ETag']
    })
    i = i + 1
    bytePosition += psize
s3_client.complete_multipart_upload(Bucket="dest", Key="copyed-10GiB",
UploadId=mpu["UploadId"], MultipartUpload=part_info)
```

设置自定义元数据

```
#上传对象并且设置元数据
s3_client.put_object(Bucket="您的bucket",Key= '您bucket下的对象',Metadata={'自定义元数据1': '元数据1值','自定义元数据2': '元数据2值'},Body="上传的内容")

#修改已有对象的元数据
#python sdk 目前对象没有post方法，因此建议在上传对象时就设置下面使用copy来实现对已有的对象的元数据更新
s3_client.copy_object(Bucket="您的bucket", Key="您bucket下的已有对象",CopySource=str('您的bucket/您bucket下的已有对象'),
MetadataDirective='REPLACE',Metadata={'自定义元数据1': '元数据1值','自定义元数据2': '元数据2值'})
```

下载文件

把对象存储服务器上的Object下载到本地文件`local-backup.bin`：

```
resp = s3_client.get_object(Bucket="您的已经存在的bucket名", Key="您该bucket中的对象名")
with open('local-backup.bin','wb') as f:
    f.write(resp['Body'].read())
```

下载文件

下载对象存储服务器上的Object的前10个字节：

```
resp = s3_client.get_object(Bucket="您的已经存在的bucket名",Key="您该bucket中的对象名", Range="bytes=0-10")
print resp['Body'].read()
```

断点续传

当需要下载的文件很大，或网络状况不够理想，往往下载到中途就失败了。如果下次重试，还需要重新下载，就会浪费时间和带宽。
断点续传的过程大致如下：

1. 在本地创建一个临时文件，文件名由原始文件名加上一个随机的后缀组成；
2. 通过指定HTTP请求的 Range 头，按照范围读取对象存储服务器上的文件，并写入到临时文件里相应的位置；
3. 下载完成之后，把临时文件重名为目标文件。

管理文件

通过Python SDK，用户可以罗列、删除、拷贝文件，也可以查看文件信息，更改文件元信息等。

罗列文件

Python SDK提供了一系列的迭代器，用于列举文件、分片上传等。

简单罗列

列举Bucket里的文件：

```
resp = s3_client.list_objects(Bucket="您的已经存在的bucket名")
for obj in resp['Contents']:
    print obj['Key']
```

按前缀罗列

只列举前缀为"img-"的所有文件：

```
resp = s3_client.list_objects(Bucket="您的已经存在的bucket名",Prefix='img-')
for obj in resp['Contents']:
    print obj['Key']
```

模拟文件夹功能

对象存储的存储空间（Bucket）本身是扁平结构的，并没有文件夹或目录的概念。用户可以通过在文件名里加入"/"来模拟文件夹。在列举的时候，则要设置delimiter参数（目录分隔符）为"/"，并通过是否"在CommonPrefixes"来判断是否为文件夹。
例如：websitebucket1下结构如下：

```
.
├── about.html
├── blog
|   ├── angry-post.html
|   ├── atom.xml
|   ├── excerpts.xml
|   ├── happy-post.html
|   ├── index.html
|   ├── sad-post.html
|   └── tags
|       ├── angry.html
|       ├── happy.html
|       ├── sad.html
|       └── thoughts.html
├── index.html
├── media
|   ├── css
|   |   ├── site.css
|   |   └── syntax.css
|   ├── images
|   |   ├── airport.png
|   |   ├── apple-touch-icon.png
|   |   ├── dark.png
```

```
|   |   └── favicon.ico
|   └── js
|       └── libs
|           ├── dd_belatedpng.js
|           ├── jquery-1.5.1.min.js
|           └── modernizr-1.7.min.js
└── portfolio
    └── index.html
```

```
#列举websitebucket1下的文件夹和文件（不递归列出子文件夹下的文件和子子文件夹）
resp = s3_client.list_objects(Bucket="websitebucket1", Delimiter='/')
print "==============DIRS FOLLOWS============"
for o in resp.get('CommonPrefixes'):
    print(o.get('Prefix'))
print "==============FILES FOLLOWS============"
for o in resp.get('Contents'):
    print(o.get('Key'))
```

结果

```
==============DIRS FOLLOWS============
blog/
media/
portfolio/
==============FILES FOLLOWS============
about.html
copy3
data4
index.html
```

```
#列举websitebucket1下的文件夹blog下的文件夹和文件（不递归列出子文件夹下的文件和子子文件夹）
resp = s3_client.list_objects(Bucket="websitebucket1",
Delimiter='/',Prefix='blog/')
print "==============DIRS FOLLOWS============"
for o in resp.get('CommonPrefixes'):
    print(o.get('Prefix'))
print "==============FILES FOLLOWS============"
for o in resp.get('Contents'):
    print(o.get('Key'))
```

结果

```
==============DIRS FOLLOWS============
blog/tags/
==============FILES FOLLOWS============
blog/
blog/angry-post.html
blog/atom.xml
blog/excerpts.xml
blog/happy-post.html
blog/index.html
blog/sad-post.html
```

判断文件是否存在

可以使用head接口来判断对象是否存在

```
try:
    s3_client.head_object(Bucket="您的已经存在的bucket名", Key="要查询的对象名字")
    print "EXIST"
except:
    print "NOT FOUND"
```

删除文件

删除单个文件

```
resp = s3_client.delete_object(Bucket="您的已经存在的bucket名", Key="您要删除的对象名")
```

删除多个文件

批量删除bucket1下的以2017-05为前缀的对象

```
objects_to_delete = [{'Key':'待删除对象1'}, {'Key':'待删除对象2'}]
s3_client.delete_objects(
    Bucket="您的桶名",
    Delete={
        'Objects': objects_to_delete
    }
)
```

拷贝文件

把Bucket名为src-bucket下的source.txt拷贝到dst_bucket下且命名为的target.txt文件。

```
resp = s3_client.copy_object(Bucket=dst_bucket, Key="target.txt",
CopySource=str(src_bucket+'/'+"source.txt"))
```

查看文件访问权限

查看文件的访问权限

```
resp = s3_client.get_object_acl(Bucket="用户存在的bucket", Key="bucket下的对象名字")
print resp['Grants']
print resp['Owner']
```

设置文件访问权限

设置文件的访问权限

```
resp = s3_client.put_object_acl(Bucket="用户存在的bucket", Key="bucket下的对象名字",
ACL='public-read')
```

使用私有链接下载

对于私有Bucket，可以生成私有链接（又称为“签名URL”）供用户访问，下面是生成私有链接下载，该链接在3600秒后失效

```
print s3_client.generate_presigned_url(
  ClientMethod = 'get_object',
  Params = {'Bucket' : "您的bucket", 'Key' : "您的bucket下的要生成私有下载链接的对象"},
  ExpiresIn = 3600,
  HttpMethod = 'GET')
```

使用私有链接上传

对于私有Bucket，可以生成私有链接（又称为“签名URL”）供用户访问，下面是生成私有链接上传，该链接在3600秒后失效

```
print s3_client.generate_presigned_url(
  ClientMethod = 'put_object',
  Params = {'Bucket' : "您的bucket", 'Key' : "您要存储到bucket下的key名字"},
  ExpiresIn = 3600,
  HttpMethod = 'PUT')

#假设生成的签名链接为 http://10.139.4.136/newbucket/newone?AWSAccessKeyId=yly&Expires=1496206092&Signature=5gP1zbR%2BtbHa8OXiUBxkKily6YU%3D


#直接使用curl和生成的签名连接上传文件
export URL="http://10.139.4.136/newbucket/newone?AWSAccessKeyId=yly&Expires=1496206092&Signature=5gP1zbR%2BtbHa8OXiUBxkKily6YU%3D"
echo abcde > test.txt  #上传的本地文件为test.txt
curl -D - -X PUT --upload-file test.txt $URL -H "Host:10.139.4.136"
```

使用私有链接删除

对于私有Bucket，可以生成私有链接（又称为“签名URL”）供用户访问，下面是生成私有链接删除对象，该链接在3600秒后失效

```
print s3_client.generate_presigned_url(
  ClientMethod = 'delete_object',
  Params = {'Bucket' : "您的bucket", 'Key' : "您要删除的对象名字"},
  ExpiresIn = 3600,
  HttpMethod = 'DELETE')

#直接用curl和生成的签名连接删除对象
export URL="http://10.139.4.136/newbucket/newone?AWSAccessKeyId=yly&Expires=1496209233&Signature=cpvl%2BXbFb1O6C%2Fp36bJGu6Gmta8%3D"
```

静态网站配置

获取静态网站配置

```
response = s3_client.get_bucket_website(
    Bucket='您的bucket名字'
)
```

开启静态网站配置

```
response = s3_client.put_bucket_website(
    Bucket='您的bucket名字',
    WebsiteConfiguration={
        'ErrorDocument': {
            'Key': '您的ErrorDocument文件，比如404.html或者error.html'
        },
        'IndexDocument': {
            'Suffix': '您的IndexDocument文件，比如index.html'
        }
    }
)
```

关闭静态网站

```
response = s3_client.delete_bucket_website(
    Bucket='您的bucket名字'
)
```

查询桶内对象个数和使用空间

```
resp = s3_client.head_bucket(Bucket="hahahaha")
print "使用空间（字节）：",resp['ResponseMetadata']['HTTPHeaders']['x-rgw-bytes-used']
print "桶内对象个数:",resp['ResponseMetadata']['HTTPHeaders']['x-rgw-object-count']
```

查询对象元数据

```
resp = s3_client.head_object(Bucket="您的bucket",Key="您bucket下的对象")
```

查询桶内使用分块上传但未完成的分块上传

```
response = s3_client.list_multipart_uploads(Bucket="您的bucket")
```

查询桶内使用分块上传但未完成的分块上传的对象的UploadId

```
response = s3_client.list_multipart_uploads(Bucket="您的bucket",Prefix="您的对象名")
print response[u'Uploads'][0][u'UploadId']
```

终止桶内使用分块上传但未完成的分块上传

```
response = s3_client.list_multipart_uploads(Bucket="您的bucket")
s3_client.abort_multipart_upload(Bucket = "您的bucket", Key = "您分块上传的对象
名",UploadId = response[u'Uploads'][0][u'UploadId'])
```

列出桶内使用分块上传接口但未完成的分块上传的对象的分块

```
response = s3_client.list_multipart_uploads(Bucket="您的bucket", Prefix="您的对象
名")
response = s3_client.list_parts(Bucket="您的bucket",Key="您的对象
名",UploadId=response[u'Uploads'][0][u'UploadId'])
for part in response['Parts']:
    print part['ETag'],part['PartNumber'],part['Size']
```

设置桶的多版本

设置容器的多版本状态。容器的多版本状态有以下 三种：

- Off: 关闭
- Enabled：开启
- Suspended：暂停

默认情况下，新建容器处于关闭多版本状态。一旦其多版本被开启，就只能暂停多版本。
开启桶的多版本，注意如果要对多版本桶下的非当前版本对象进行操作的话需要带上对象的版本号。

```
response =
s3_client.put_bucket_versioning(Bucket="hahahaha",VersioningConfiguration={
        'Status': 'Enabled'
    })
```

关闭桶的多版本

```
response =
s3_client.put_bucket_versioning(Bucket="hahahaha",VersioningConfiguration={
        'Status': 'Suspended'
    })
```

获取桶的多版本状态

```
response = s3_client.get_bucket_versioning(Bucket="您的bucket")
print response['Status']
```

列出多版本对象的所有版本

```
resp = s3_client.list_object_versions(Bucket="您的bucket",Prefix="您的对象")
for i in resp['Versions']:
    print i['Key'],i['VersionId'],i['ETag']
```

设置防盗链

如下桶设置对来自http://www.example.com/域名下的浏览器请求通过的防盗链设置

```
#单域名
```

```
response = client.put_bucket_policy(
    Bucket='您要设置防盗链的桶名字',
    Policy='''{
    "Version": "2012-10-17",
    "Statement": [{
    "Effect": "Allow",
    "Principal": "*",
    "Action": "s3:GetObject",
    "Resource": [
      "arn:aws:s3:::您要设置防盗链的桶名字/*"
    ],
    "Condition": {
        "StringLike": {
          "aws:Referer": "http://www.example.com/*"
        }
      }
  }]
}'''
)

#多域名
response = client.put_bucket_policy(
    Bucket='您要设置防盗链的桶名字',
    Policy='''{
    "Version": "2012-10-17",
    "Statement": [{
    "Effect": "Allow",
    "Principal": "*",
    "Action": "s3:GetObject",
    "Resource": [
      "arn:aws:s3:::您要设置防盗链的桶名字/*"
    ],
    "Condition": {
        "StringLike": {
          "aws:Referer":
["http://www.example.com/*","http://www.example2.com/*"]
        }
      }
  }]
}'''
)
```

获取防盗链

```
response = client.get_bucket_policy(
    Bucket='设置了防盗链的桶名字'
)
print response['Policy']
```

删除防盗链

```
response = client.delete_bucket_policy(
    Bucket='设置了防盗链的桶名字'
)
```

> 同一个桶上不建议防盗链和静态网站同时开启，否则访问静态网站会失败。最佳实践：桶1开启静态网站，桶2存放静态网站需要的图片和视频资源，然后开启这个桶2的防盗链，设置允许桶1的域名访问桶2的资源。

设置跨域访问

```
s3_client.put_bucket_cors(
            Bucket="桶的名字",
            CORSConfiguration={
                'CORSRules': [
                    {
                        'AllowedMethods': ['POST', 'GET', 'PUT', 'DELETE',
'HEAD'],
                        'AllowedOrigins': ['*'],
                        'AllowedHeaders': ['*'],
                        'MaxAgeSeconds':100,
                    },
                ],
            },
        )
```

获取跨域访问

```
s3_client.get_bucket_cors(
            Bucket="桶的名字",
        )
```

删除跨域访问

```
s3_client.delete_bucket_cors(
            Bucket="桶的名字",
        )
```

回调功能

回调功能需要使用修改代码后的botocore

```
#卸载原有的botocore
pip uninstall botocore
#安装支持回调的botocore
pip install https://github.com/joke-lee/botocore/archive/1.0.zip
```

put_object回调

```
import base64
import json
```

```
callback_dict = {}
callback_dict['callbackUrl'] = 'http://67.218.159.42:23450/index.php?
id=1&index=2'
callback_dict['callbackBody'] =
'bucket=${bucket}&filename=${object}&size=${size}&mimeType=${mimeType}&my_var=${
x:var1}'
callback_dict['callbackBodyType'] = 'application/x-www-form-urlencoded'

callback_var = {}
callback_var["x:var1"]='value1'
callback_var["x:var2"]='value2'


callback_param = json.dumps(callback_dict).strip()
base64_callback_body = base64.b64encode(callback_param)
headers={'x-amz-meta-callback': base64_callback_body}

callback_var_param = json.dumps(callback_var).strip()
base64_callback_var = base64.b64encode(callback_var_param)
headers['x-amz-meta-callback-var'] = base64_callback_var

response = s3_client.put_object(Bucket="桶名字",Key="对象名字", Metadata=
{'callback': base64_callback_body, 'callback-var': base64_callback_var },
Body="上传对象内容")
print response
```

complete_multipart_upload回调

```
import base64
import json

callback_dict = {}
callback_dict['callbackUrl'] = 'http://67.218.159.42:23450/index.php?
id=1&index=2'
callback_dict['callbackBody'] =
'bucket=${bucket}&filename=${object}&size=${size}&mimeType=${mimeType}&my_var=${
x:var1}'
callback_dict['callbackBodyType'] = 'application/x-www-form-urlencoded'

callback_var = {}
callback_var["x:var1"]='value1'
callback_var["x:var2"]='value2'

callback_param = json.dumps(callback_dict).strip()
base64_callback_body = base64.b64encode(callback_param)
headers={'x-amz-meta-callback': base64_callback_body}

callback_var_param = json.dumps(callback_var).strip()
base64_callback_var = base64.b64encode(callback_var_param)
headers['x-amz-meta-callback-var'] = base64_callback_var


s3_client.complete_multipart_upload(Bucket="桶的名字", Key="对象名字",
UploadId=mpu["UploadId"], MultipartUpload=part_info, Metadata={'callback':
base64_callback_body,'callback-var': base64_callback_var} )
```

设置生命周期

```
s3_client.put_bucket_lifecycle(Bucket='桶的名字', LifecycleConfiguration={
'Rules': [
{
'Expiration': {
'Days': 1,
},
'ID': 'test',
'Prefix': '',
'Status': 'Enabled',
},
]
})
```

获取生命周期

```
s3_client.get_bucket_lifecycle(Bucket='桶的名字')
```

删除生命周期

```
s3_client.delete_bucket_lifecycle(Bucket='桶的名字')
```

设置转低频生命周期

```
# -*- coding: utf-8 -*-
import boto.s3.connection
boto.set_stream_logger('boto')
host = '数据中心endpoint'
access_key = '您的access-key'
secret_key = '您的secret-key'
conn = boto.connect_s3(
    access_key,
    secret_key,
    host=host,
    is_secure=False,
    port=80,
    calling_format=boto.s3.connection.OrdinaryCallingFormat(),
)

lcxml = '''<LifecycleConfiguration>
    <Rule>
        <ID>任意取id号</ID>
        <Prefix>将被转低频对象的前缀</Prefix>
        <Status>Enabled</Status>
        <Transition>
            <Days>30</Days>
            <StorageClass>STANDARD_IA</StorageClass>
        </Transition>
        <NoncurrentVersionTransition>
            <NoncurrentDays>60</NoncurrentDays>
            <StorageClass>STANDARD_IA</StorageClass>
        </NoncurrentVersionTransition>
    </Rule>
```

```
</LifecycleConfiguration>'''

import StringIO
StringIO = StringIO.StringIO
fp = StringIO(lcxml)
md5 = boto.utils.compute_md5(fp)
headers = {'Content-Md5': md5[1]}
bucketname = '要设置的桶名'
bucket = conn.get_bucket(bucketname)
req = bucket.connection.make_request("PUT", bucket.name , data = lcxml, headers
= headers, query_args='lifecycle')
```

> 以上设置设置了桶当前版本对象在30天后转为低频类型，非当前版本的对象在60天后转为低频类型,只有华南节点guangzhou1数据中心支持

设置桶日志

授权LogDeliver用户对目标桶有权限

```
s3cmd setacl s3://{存日志桶名字} --acl-grant=write:LogDeliver
s3cmd setacl s3://{存日志桶名字} --acl-grant=read_acp:LogDeliver
```

```
bl =  {
        'LoggingEnabled': {
            'TargetBucket': '存日志桶名字',
            'TargetPrefix': '目录前缀/'
        }
    }
s3_client.put_bucket_logging(Bucket="桶名字", BucketLoggingStatus = bl)
```

获取桶日志配置

```
s3_client.get_bucket_logging(Bucket="桶名字")
```

停止桶日志

设置为空就是停止

```
bl =  {
    }
s3_client.put_bucket_logging(Bucket="桶名字", BucketLoggingStatus = bl)
```

追加写

追加写使用boto库

```
import boto.s3.connection
boto.set_stream_logger('boto')
host = '您的Endpoint域名' #例如eos-beijing-1.cmecloud.cn
access_key = "您的AccessKey"
secret_key = "您的SecretKey"
conn = boto.connect_s3(
                access_key,
```

```
                secret_key,
                host = host,
                is_secure = False,
                port=80,
                calling_format = boto.s3.connection.OrdinaryCallingFormat(),
                )
bucket = conn.get_bucket("桶的名字")
obj = "对象名字"
response = bucket.connection.make_request("PUT", bucket.name , obj, None,
"abcd", query_args='append&position=0')

response = bucket.connection.make_request("PUT", bucket.name , obj, None, "ef",
query_args='append&position=4')  #abcd长度为4

#上传后的对象为abcdef
```

查询追加对象的position

```
import boto.s3.connection
boto.set_stream_logger('boto')
host = '您的Endpoint域名'  #例如eos-beijing-1.cmecloud.cn
access_key = "您的AccessKey"
secret_key = "您的SecretKey"
conn = boto.connect_s3(
                access_key,
                secret_key,
                host = host,
                is_secure = False,
                port=80,
                calling_format = boto.s3.connection.OrdinaryCallingFormat(),
                )
bucket = conn.get_bucket("桶的名字")
obj = "对象名字"
response = bucket.connection.make_request("HEAD", bucket.name , obj)
print response.msg.dict['x-rgw-next-append-position']
```

设置软链接

```
import boto.s3.connection
boto.set_stream_logger('boto')
host = '您的Endpoint域名' #例如eos-beijing-1.cmecloud.cn
access_key = "您的AccessKey"
secret_key = "您的SecretKey"
conn = boto.connect_s3(
                access_key,
                secret_key,
                host = host,
                is_secure = False,
                port=80,
                calling_format = boto.s3.connection.OrdinaryCallingFormat(),
                )
bucket = conn.get_bucket("桶的名字")
obj = "对象名字"
headers = {'x-amz-meta-symlink':'目标对象名字'}
response = bucket.connection.make_request(method = "PUT", bucket = bucket.name ,
key = obj, headers = headers, data="", query_args='symlink')
```

查询软连接

```
import boto.s3.connection
boto.set_stream_logger('boto')
host = '您的Endpoint域名' #例如eos-beijing-1.cmecloud.cn
access_key = "您的AccessKey"
secret_key = "您的SecretKey"
conn = boto.connect_s3(
                access_key,
                secret_key,
                host = host,
                is_secure = False,
                port=80,
                calling_format = boto.s3.connection.OrdinaryCallingFormat(),
                )
bucket = conn.get_bucket("桶的名字")
obj = "对象名字"
response = bucket.connection.make_request(method = "HEAD", bucket = bucket.name
, key = obj, headers = None, data="", query_args='symlink')
print response.msg.dict['x-amz-meta-symlink']
```

设置防盗链v2

```
import boto.s3.connection
boto.set_stream_logger('boto')
host = '您的Endpoint域名' #例如eos-beijing-1.cmecloud.cn
access_key = "您的AccessKey"
secret_key = "您的SecretKey"
conn = boto.connect_s3(
                access_key,
                secret_key,
                host = host,
                is_secure = False,
                port=80,
                calling_format = boto.s3.connection.OrdinaryCallingFormat(),
                )
bucket = conn.get_bucket("桶的名字")

referers = '''<?xml version="1.0" encoding="UTF-8"?>
<RefererConfiguration
    xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
    <AllowEmptyReferer>false</AllowEmptyReferer>
    <RefererList>
        <Referer>http://www.example.com</Referer>
        <Referer>http://www.example?.com</Referer>
        <Referer>https://www.*.onest.com</Referer>
    </RefererList>
</RefererConfiguration>
'''
response = bucket.connection.make_request(method = "PUT", bucket = bucket.name ,
data = referers, query_args='referer' )
```

查询防盗链v2

```
import boto.s3.connection
boto.set_stream_logger('boto')
host = '您的Endpoint域名' #例如eos-beijing-1.cmecloud.cn
access_key = "您的AccessKey"
secret_key = "您的SecretKey"
conn = boto.connect_s3(
                access_key,
                secret_key,
                host = host,
                is_secure = False,
                port=80,
                calling_format = boto.s3.connection.OrdinaryCallingFormat(),
                )
bucket = conn.get_bucket("桶的名字")
response = bucket.connection.make_request(method = "GET", bucket = bucket.name ,
query_args='referer' )
body = response.read()
print body
```

创建桶时指定桶的默认属性为低频

> 只有部分数据中心支持低频存储

```
import boto.s3.connection
boto.set_stream_logger('boto')
host = 'eos-guangzhou-1.cmecloud.cn'
access_key = '您的access-key'
secret_key = '您的secret-key'
conn = boto.connect_s3(
                access_key,
                secret_key,
                host = host,
                is_secure = False,
                port=80,
                calling_format = boto.s3.connection.OrdinaryCallingFormat(),
                )
bucketname = '要创建的桶名'
bucketxml = '''<CreateBucketConfiguration
xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
  <LocationConstraint>guangzhou1</LocationConstraint>
  <StorageClass>STANDARD_IA</StorageClass>
</CreateBucketConfiguration>'''

req = conn.make_request('PUT', bucketname, headers=None,
                data=bucketxml)
print req.read()
print req.status
```

分块上传加密对象

```
mpu = s3_client.create_multipart_upload(
    Bucket="桶名",
    Key="对象名",
    SSECustomerAlgorithm ='AES256',
    SSECustomerKey='dAgbyOHCjBJ+OsusTU2t/XMHOTxMU+zzsIRVo4sWj3s=',
    SSECustomerKeyMD5='2LnIX++18heOZTEGMlmutg==')
```

```
part_info = {
    'Parts': []
}
file = open("本地文件名","rb")
i = 1
while 1:
    data = file.read(5 * 1024 * 1024) #按5M大小切块
    if data == b'':
        break
    response = s3_client.upload_part(Bucket="桶名",
                                     Key="对象名",
                                     PartNumber=i,
                                     UploadId=mpu["UploadId"],
                                     Body=data,
                                     SSECustomerAlgorithm='AES256',
SSECustomerKey='dAgbyOHCjBJ+OsusTU2t/XMHOTxMU+zzsIRVo4sWj3s=',

SSECustomerKeyMD5='2LnIX++18heOZTEGMlmutg=='
                                     )
    part_info['Parts'].append({
        'PartNumber': i,
        'ETag': response['ETag']
    })
    i += 1

s3_client.complete_multipart_upload(Bucket="桶名",Key="对象
名",UploadId=mpu["UploadId"],MultipartUpload=part_info)
```

桶通知

> 目前只支持TopicConfigurations类型的通知

获取桶通知配置

```
conf = s3_client.get_bucket_notification_configuration(Bucket="桶名字")
print json.dumps(conf['TopicConfigurations'], indent=4, sort_keys=True)
```

设置桶通知配置

设置当上传`.jpg`结尾的文件的时候发送通知

```
bucket_notifications_config = {
'TopicConfigurations' : [
        {
            "Events": [
                "s3:ObjectCreated:*"
            ],
            "Filter": {
                "Key": {
                    "FilterRules": [
                        {
                            "Name": "Suffix",
                            "Value": ".jpg"
                        }
```

```
                    ]
                }
            },
            "Id": "image",
            "TopicArn": "arn:aws:sns:us-east-1:***:image"
        },
        {
            "Events": [
                "s3:ObjectCreated:*"
            ],
            "Filter": {
                "Key": {
                    "FilterRules": [
                        {
                            "Name": "Suffix",
                            "Value": ".txt"
                        }
                    ]
                }
            },
            "Id": "png",
            "TopicArn": "arn:aws:sns:us-east-1:***:image"
        }
    ]
}
s3_client.put_bucket_notification_configuration(Bucket="桶名",
NotificationConfiguration=bucket_notifications_config)
```

删除桶通知配置

清空配置即为删除桶通知配置

```
bucket_notifications_config = {
'TopicConfigurations' : [
    ]
}
s3_client.put_bucket_notification_configuration(Bucket="桶名",
NotificationConfiguration=bucket_notifications_config)
```