

Table of Contents

Introduction	1.1
简介	1.2
常见问题 FAQ	1.3
对象存储 Object Storage Service	1.4
实例 兼容Amazon SDK	1.4.1
Python	1.4.1.1
PHP	1.4.1.2
PHPV2	1.4.1.3
Browser	1.4.1.4
Nodejs	1.4.1.5
Java	1.4.1.6
Go	1.4.1.7
Dotnet	1.4.1.8
约束与限制	1.4.2
签名算法	1.4.3
ACL	1.4.4
Service	1.4.5
LIST Buckets	1.4.5.1
Bucket	1.4.6
GET Bucket or List Objects	1.4.6.1
PUT Bucket	1.4.6.2
DELETE Bucket	1.4.6.3
PUT Bucket ACL	1.4.6.4
GET Bucket ACL	1.4.6.5
Object	1.4.7
HEAD Object	1.4.7.1
GET Object	1.4.7.2
PUT Object	1.4.7.3
APPEND Object	1.4.7.4
POST Object	1.4.7.5
PUT Object - Copy	1.4.7.6
DELETE Object	1.4.7.7

GET Object ACL	1.4.7.8
PUT Object ACL	1.4.7.9
Initiate Multipart Upload	1.4.7.10
Upload Part	1.4.7.11
Upload Part copy	1.4.7.12
Complete Multipart Upload	1.4.7.13
List Parts	1.4.7.14
临时签名:STS	1.4.8
离线下载 Offline Downloader	1.5
图片实时处理API imgx	1.6
视音频处理	1.7
请求签名	1.7.1
管道相关接口	1.7.2
创建管道	1.7.2.1
获取管道	1.7.2.2
列管道	1.7.2.3
删除管道	1.7.2.4
更新管道	1.7.2.5
任务相关接口	1.7.3
创建任务	1.7.3.1
获取任务	1.7.3.2
列出指定管道的任务	1.7.3.3
取消任务	1.7.3.4
音视频元信息	1.7.4
工具	1.8
windows 图形界面存储管理工具 s3-browser	1.8.1
控制台使用手册	1.9
白山云存储控制台操作手册	1.9.1
All-in-One	1.10

概述

我们提供了一套兼容AmazonS3的RESTful API，可以使您更加自由地开发出灵活的功能。

白山云存储服务主要提供以下三类API：

- Service操作
- Bucket操作
- Object操作

与此同时，为提高用户使用的安全性，白山云存储服务还通过使用签名来验证请求者的身份。

如需了解签名算法的详细信息，请参考 [《签名算法》](#)。

注意：以下接口中所使用的示例都是在需要使用签名情况下；如果相关访问资源已设置为可匿名（所有用户）访问，则可不带签名。



白山云存储服务(BaishanCloud storage service)

白山云存储服务介绍

白山云存储服务：是以白山云强大的CDN-X服务优势为基础，延展出的多数据中心、多副本、分布式架构、对象云存储服务。

功能性概述：

1. 高可靠、高性能：通过多数据中心、多副本、分布式架构、多设备冗余、跨区域复制、异地容灾、系统内部智能调度保障用户数据稳定存储、持久可靠；通过CDN-X边缘节点进行加速上传、通过全网CDN进行加速下载。
2. 高安全、高防护：系统内部有多重安全性、可靠性、用户资源隔离等机制；精确的数据权限管理、多级分权、多种鉴权、多种授权管理机制，防止盗链、保护数据安全。
3. 弹性扩容、注重服务：无限制扩容、无需迁移、满足非结构化数据增长趋势、分布式和分析功能；配置专属技术服务接口人深入了解用户需求、跟进用户服务。
4. 功能齐备、简单易用：研发团队核心成员7年以上对象云存储服务经验积累，全自主知识产权，自研核心组件，高并发生产环境验证；用户编程接口简洁易用、稳定可靠

技术优势：

1. 强大的CDN优势：通过白山云自主的CDN-X边缘节点进行加速上传、通过全网CDN进行加速下载。

2. 遵循和兼容规范：遵循和兼容Amazon S3 REST API接口，推荐开发人员直接使用Amazon S3的官方SDK，简洁易用、稳定可靠，用户可以使用各种开源客户端、工具等。
 3. 音视频转码和图片处理：提供离线音视频转码功能，丰富、便捷的图片处理接口（格式转换、旋转、缩放、剪裁、压缩、水印、模糊等滤镜操作），用户可以高效在线处理静态图片，避免图片重复上传/下载。
 4. 强大的对象云存储服务架构设计：实现数据温度测量、自组边缘存储、对等多活、自激质量控制。
-

4项技术创新：

1. 数据温度测量：

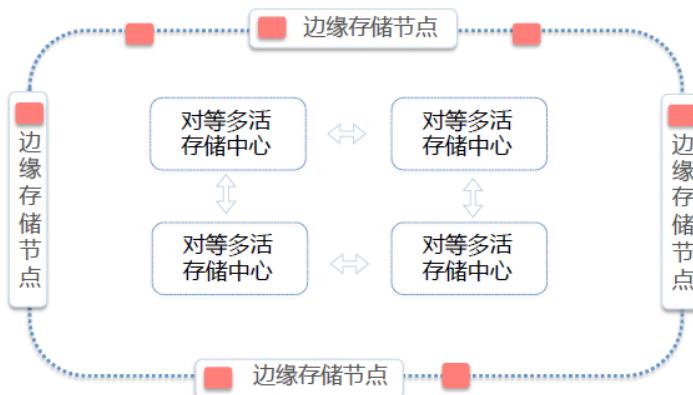
数据温度测量技术



1. 定义数据温度：从数据大小、类型、来源、访问频次、一度关系、可信度共六个维度，来定义数据的热度。内部机制调度数据冷热分级。热数据用高IO存储，热数据自动迁移。
2. 数据分层存储：分层冷热数据后，对热数据用3副本策略提供更高 IO，对冷数据则提供更高可靠性。通过存储端的数据统计，将承载系统 90% 的 IO 的数据定义为热数据，其他定义为冷数据，并在系统内部定期将3副本保存的热数据，转定义成为冷数据。
3. 异步定期合并：冷热分离中使用分层的存储方式，将较新的数据和较老的数据设定为不同的层次，定期将上层的新数据和下层的老数据合并到下层数据的层次中，这种异步定期合并的方式，将冷数据的更新的IO消耗降低了1到2个数量级。

2. 自组边缘节点

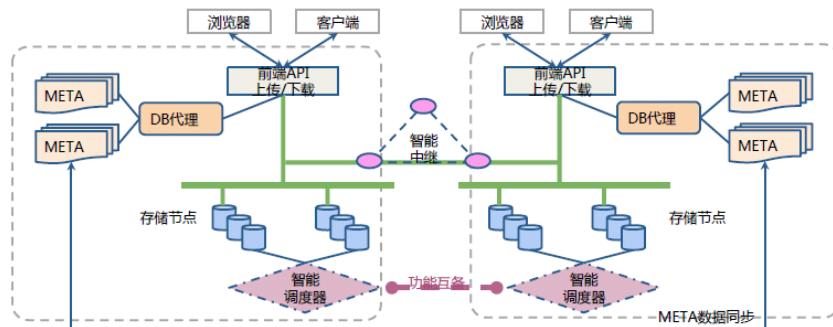
自组边缘存储技术



1. 统一规划网络：通过p2p 的广播网络, 将128个边缘节点连接成一个网络, 在1000ms 以内将用户上传文件的消息通知到全国的每个机房, 以此达到用户数据低延迟的最终一致性。让用户能快速读取到远距离用户上传的文件。
2. 内容就近写入：自组边缘节点部署在全国128个距离用户最近的机房中, 当用户上传或下载文件时, 自动将用户定位到距离最近的机房, 实现用户的告诉上传和下载。
3. 光纤链路接入：边缘节点同时与5个中心核心存储机房保持通信, 通过数据传输加速网络, 在几秒钟的时间内, 异步将用户上传的文件传输到中心存储, 保证数据最终的可靠性达到13个9。
4. 隔离网络故障：同时, 当中心存储出现故障或者边缘存储节点无法与中心存储通信时, 边缘节点也可以在一定时间内支持用户的上传和下载请求, 以达到对用户隔离网络故障的效果。

3. 对等多活

对等多活存储技术



1. 高质量多数据中心互联：多中心存储光纤互联，大文件秒级同步。实现数据库异地多写多读：完全规避了机房故障造成的服务可用性问题。通过数据库在多个机房内部署多主复制，实现多地的同时读取和写入，完成异地数据备份功能。
2. 秒级跨机房同步：保证任意一个机房出现故障的时候，都不会影响业务写入和读取，真正做到全网跨多机房冗余和灾备。
3. 秒级反应故障：故障发现机制提供了秒级的故障反应速度。自动隔离不可用的数据库服务，变更读写到可用服务节点。
4. 秒级数据库探测：每秒定时连接数据库实例，探测是否可用，当连接不可用时，立即向跨机房的配置中心服务发送切换数据库的指令，更新配置信息。每个数据库代理实时监视配置中心的配置变化，当配置发生变化时，在10毫秒级别实现配置重新加载和生效。达到高可用的数据连接。
5. 将配置去中心化：将备份的配置中心服务部署到多个机房，使得任何一个机房出现故障时，系统仍可以正常工作，进行正常的配置变化，故障发现。

4. 自激质量控制

自激质量控制技术



1. paxos算法与自动化数据修复：通过paxos算法，用于实现多副本的自动管理，数据自我修复，多副本之间保持一致性等。将整个存储系统的集群划分成10万个自我维护、自我修复的组，每个组内定时探测其他节点的存活状态，数据正确性，以及以分钟为粒度来进行数据的修复。
2. 对paxos的改进：首先paxos算法保证了自动化数据修复的正确性。paxos本身没有包含对paxos组管理的算法描述。线上环境使用paxos算法时必须考虑到组成员变化的处理。白山云存储系统引入了自行设计和实现的paxos组管理算法，通过2个阶段的paxos实例的运行，可以实现组成员的变更。在变更过程中我们的成员变更算法保证数据一致性，变更结果不会因为网络，磁盘故障而影响最终结果。

3. 保证运行过程的安全性: 成员变更的过程中任何网络或磁盘故障不会造成系统处于不完整的状态, 服务重启之后仍可以继续运行。
4. 全面质量检测: 60秒根据磁盘状态切换写入点; 5分钟检查所有副本完备性; 24小时完成副本数据对比和自我修复; 1个月全量数据扫描纠错。

服务的规模和可扩展性:

可扩展服务项	可扩展至数量级
服务可承载用户数量	100万用户
服务数据存储量	100PB级, 千亿文件
数据写入能力(上传)	24小时上传超过500TB (目前为40TB)
服务整体吞吐量(下载)	24小时下载10PB(目前为1PB)
物理服务器数量	3000 (多数据中心)

服务等级协议 (Service-Level Agreement, SLA) :

1. 可靠性: (对象存储服务的持久性) 为99.9999999999% (13个9)。

互联网主干节点各运营商平均上传/下载速度:

传输文件大小	上传速度	下载速度
50KB	300KB/秒	900KB/秒
2MB	750KB/秒	2MB/秒
4MB	1MB/秒	2.5MB/秒

遵循和兼容Amazon S3的接口，推荐开发人员可直接使用Amazon S3的官方SDK：

说明文档	SDK包下载
Android文档: AWS Mobile SDK for Android Documentation	SDK下载: aws-android-sdk
Browser文档: Getting Started with the SDK in the Browser	SDK下载: Bower
IOS文档: AWS Mobile SDK iOS Developer Guide	SDK下载: aws-ios-sdk
Java文档: AWS SDK for Java Developer Guide	SDK下载: aws-java-sdk
.NET文档: AWS SDK for .NET Documentation	SDK下载: AWSToolsAndSDKForNet.msi
Node.js文档: AWS SDK for JavaScript Documentation	SDK下载: Getting Started with the SDK in Node.js
PHP文档: AWS SDK for PHP Documentation	SDK下载: Installing via Composer
Python文档: Boto 3 Documentation	SDK下载地址及说明: Boto 3 - The AWS SDK for Python
Ruby文档: AWS SDK for Ruby	SDK下载地址: AWS SDK for Ruby Developer Guide v2.16.8
Go文档: AWS SDK for Go	SDK下载地址: aws-sdk-go
C++文档: AWS SDK for C++ Documentation	SDK下载地址: aws-sdk-cpp

进阶技术讲解

1.白山云存储服务 & 白山云CDN-X

CDN-X加速支持:

- CDN上传加速。
- CDN虚拟域名回源。
- CDN签名回源，保护用户的数据安全。
- 第三方CDN回源。

CDN边缘节点上传:

白山云存储通过和CDN的边缘节点配合使用，提升上传的速度。
用户请求加速上传域名，由DNS服务器智能将距离用户最近的CDN边缘节点的ip返给用户。
用户将文件上传到最近的边缘节点，边缘节点通过最优路径，将数据传输到存储中心。

通过边缘节点加速上传和直接上传到存储中心的对比，平均速度提升一倍多，极大地改善了上传的体验。
配置使用加速上传域名即可上传加速功能。

CDN-X和云存储的配置方法:

如果要测试通过CDN进行下载加速的情况，需要将用户要访问的已备案的域名CNAME到CDN给出的域名。例如www.qq.com.i.qingcdn.com，CDN回源至ss.bscstorage.com。

创建bucket的建议：

1. 以用户域名创建同名的bucket。例如用户网站域名
www.baishancloud.com，那么创建www.baishancloud.com这样一个bucket，这时CDN配置加速时，无需修改host。
2. 用户自定义bucket的名称：例如用户网站域名
www.baishancloud.com，用户要创建一个qq的bucket，那么CDN在配置加速时，用户访问
www.baishancloud.com.s2.i.baishancloud.com，回源时，需要将host修改成qq.ss.bscstorage.com。

404回源的方法及如何拉取文件:

如果白山云存储上不存在该文件，可以配置指定去某个域名拉取文件。 -
文件的拉取是异步的，拉取完成时间和拉取的文件大小相关。同时该请求会返回302，location响应头的值是文件拉取的地址。

使用拉取功能时，需要提供：

- 回源的域名domain。
- 请求uri到回源拉取文件uri的映射规则。
- 回源请求的校验方式（可选）。
- 是否启用HEAD请求回源拉取文件。

源存储如何设置缓存时间:

- 白山云存储可以设置所存储文件的Cache-Control等HTTP请求头的设置，并将缓存设置与HTTP访问请求一起下发。

- 白山云CDN可以按源站返回HTTP请求头里的Cache-Control设置执行缓存策略。

2. 对象存储的对象概念、目录结构、权限设置、传输导入

Bucket和Object的概念

- Bucket是桶的意思，这个桶里面就可以存放很多文件。
- Object是对象的意思，我们可以把每个文件都作为一个对象。
- 用户可以创建桶、删除桶、查询桶里面的对象、可以设置桶的属性等。
- 用户可以创建一个对象（这个对象就是用户上传的文件）、删除对象、下载对象、设置对象的访问权限等。
- 小例子：
 - 创建一个桶，桶名叫：baishan_yun，请求的url：PUT /baishan_yun。
 - 上传文件到baishan_yun的桶里，对象名叫：hello_world，每个文件通过桶名和对象名来唯一标识，请求的url：PUT /baishan_yun/hello_world。
 - 要下载这个文件，请求的url：GET /baishan_yun/hello_world
 - 删除这个文件，请求的url：DELETE /baishan_yun/hello_world
 - 删除桶，请求的url：DELETE /baishan_yun

对象存储的目录结构：

白山云存储的Object不是传统意义上的一个文件，存储方式也不是以文件系统保存的。每个Object看上去是一个目录下的文件，但对于云存储只是一个类似目录的字符串。但是这并不影响用户想从指定目录列文件的方式：

当发送列文件的请求时，设置prefix参数为目录前缀的名字，则可以只列出这个目录下的文件，比如：

- bucket/1.mp4
- bucket/cc/2.mp4
- bucket/cc/3.mp4
- bucket/dd/ee/ff/3.mp4

当请求的 prefix 指定为 cc 时，将会列出：

- 2.mp4
- 3.mp4

跟从目录列文件的体验是一样的。除了模拟目录列文件，还提供指定列出最大的文件数目、指定从某个文件名之后列出文件(按照字母顺序进行排序)、将文件按公共前缀进行合并等功能，详细可参照API文档。

使用的时候需要注意的问题：

- Object 名字最好不要用 '/' 开头 比如： 需要上传一个叫file.txt文件， bucket 是 `sandbox` ，
 - object 设置为 `file.txt` , 访问的时候应该是
`http://ss.bscstorage.com/sandbox/file.txt` (推荐方式)
 - object 如果设置为 `/file.txt` , 访问的时候应该是
`http://ss.bscstorage.com/sandbox//file.txt` (强烈不推荐)
- Object 名字中最好不要有两个连续的 '/' 比如： 需要在`folder`目录下上传一个叫file.txt文件， bucket 是 `sandbox` ,
 - object 设置为 `folder/file.txt` , 访问的时候应该是
`http://ss.bscstorage.com/sandbox/folder/file.txt` (推荐方式)
 - object 如果设置为 `folder//file.txt` , 访问的时候应该是
`http://ss.bscstorage.com/sandbox/folder//file.txt` (强烈不推荐)
 - 为什么白山云存储不建议在URL中带有2个'/'：某些CDN的cache server可能会把URL中两个连续的'/'合并为一个'/'来回源，这个时候就会出现验证失败或者404

如何删除白山云存储中的目录：

白山云存储删除目录的方法为：白山云存储不支持直接删除一个目录，因为 Amazon S3 API 没有严格意义上的目录的概念(没有目录层级的结构,但习惯上把 Object 按照"/"分隔当做目录来使用),这种情况下,对多个文件操作的原子性方面没法保证，所以这方面都是写脚本实现的。

可以通过以下方式完成该操作：

- 通过List Objects接口列出该目录的所有文件。
- 通过Delete Object接口逐一进行删除。
- 删除目录。

请注意：

- List Objects接口返回的文件列表不全部是该目录下的文件 (列表是根据文件名字符排序的) ， 可能包含下一个或几个目录；所以需要根据实际情况调整`max-keys`参数的值， 并且需要对每一个文件进行筛选，确认是该目录的文件; 如果设置 `prefix` 参数的话，就可以严格的列出以该 `prefix` 下所有文件.

- 一次删除（列出所有文件，逐一删除）不能保障该目录的文件都被清理完成 在删除的过程中可能会有新上传，所以整个过程不能保持原生性。

访问控制权限详解：

被授权者可以是某个用户、某个组、所有用户

1.对于Bucket的控制权限：

可读：允许被授权者列出bucket中所有的对象。
可写：允许被授权者创建、覆盖和删除存储桶中的对象。
可读ACL：允许被授权者读取存储桶的ACL。
可写ACL：允许被授权者修改存储桶的ACL。
全部权限：允许被授权者对存储桶的可读、可写、可读ACL、可写ACL。

2.对于Object的控制权限：

可读：允许被授权者读取对象的数据和meta信息。
可读ACL：允许被授权者读取对象的ACL。
可写ACL：允许被授权者修改对象的ACL。
全部权限：允许被授权者在对象上的可读、可读ACL、可写ACL。

如何使用不用签名操作操作bucket和object：

正常的操作都需要签名进行用户权限的验证，但是也允许设置共有属性，让用户不用签名来操作。

- 将bucket权限设置成public-read之后，任何用户都可以列出这个bucket下的文件，请求不需要签名。
- 将bucket权限设置成public-read-write之后，任何用户都可以列出这个bucket下的文件，并且可以上传文件到该bucket下，请求不需要签名。
- 当上传文件时，指定Object的权限是public-read之后，任何用户都可以下载该文件，不需要签名，即所谓的匿名下载。
- 当用户通过CDN回源来访问存储时，因为给CDN配置了超级权限，所以请求不需要签名也可以访问存储上的文件。

4种数据导入方式：

1. 用户使用SDK上传文件接口，将文件上传到存储上。
2. 用户使用SDK生成离线下载任务，由存储的离线下载子系统负责将数据从源站进行下载，同步到存储上。
3. 用户配置404回源的规则，当用户访问该用户的bucket，会自动去源站将数据抓取下来，同步到存储上。

4. 用户提供文件的下载列表，白山云存储通过文件列表，将数据下载并同步到存储上。

https设置方法：

1. 如果使用用户域名，首先用户域名需要备案，用户将申请域名的SSL证书提交白山云，白山云将SSL证书加入用户配置，即可使用。
2. 如果使用白山云存储提供的域名，例如：
<https://ss.bscstorage.com>, **注：必须将bucket放在url请求中，而不是放在域名中。**

对ftp、rsync上传的支持：

白山云存储服务基于文件权限的控制和接口简洁可靠的原因，不直接支持ftp和rsync协议。

但用户可以使用如下工具，实现ftp、rsync上传。

- 图形界面的ftp替代品可以使用Mac下的[cyberduck](#)或Windows下的[s3browser](#)。
- 命令行方式的ftp或rsync的替代品可以使用[s3cmd](#)。

大文件上传的3个步骤：

1. 首先调用分片上传初始化接口，请求返回一个uploadId，后续2步需要这个uploadId标识每个分片上传。
2. 将大文件分成若干片，调用分片上传接口，上传每个分片，请求中通过partNumber参数标识这个块的顺序。
3. 上传完成后，调用分片完成接口，通知存储，文件分片上传完成。

白山云存储的总数据容量和数据对象数量不受限制。各个数据对象的大小可在1字节至1TB之间。可在单个PUT中上传的最大数据对象为1TB。

建议使用分片上传接口进行大文件上传，分片上传的规范是：

1. 每个分片最大为512MB。
2. 最多分片数为2000。
3. 单个文件最大为1TB。

如何使用离线下载同步数据：

离线下载服务支持HTTP协议、BT协议、磁力链接等。

通过提交离线下载任务，离线下载服务会根据任务中提供的文件地址，从网站下载文件，并同步到存储上。同步完成后，给用户发送操作结果。

支持下载任务的状态查询、任务取消、下载数据损坏检查，消息秒传等功能高性能，性能达到2000多rps/s。

如何使用：

- 使用SDK，向offline.bscstorage.com发送请求，请求的url：
http://offline_domain/，bucket是要存放文件的桶。
- 发送请求的body体为离线的任务，例如：

```
{
    "Url": "http://www.abc.com/download/movie.mp4",
    "Key": "movies/2016/movie.mp4",
    "SHA1": "abcdeabcdeabcdeabcdeabcdeabcdeabcde",
    "MD5": "abcdabcdabcdabcdabcdabcdabcd",
    "CRC32": "abcdef",
    "SuccessCallbackUrl": "http://website/succ/",
    "FailureCallbackUrl": "http://website/fail/",
}
```

- 用户需要起一个服务用于接收离线下载服务发送的离线下载结果。

3. 音视频转码、静态图片处理、自助管理控制台

音视频转码：

支持flv、mp4、mov等格式之间的转换 支持降低码率、分辨率、修改gop(关键帧间隔)等功能 支持视频截图。

白山存储的静态文件处理：

- 支持静态图片的格式转换、旋转、缩放、剪切、图文混合水印及自定义图片处理，详见imgx_manual文档。
- 处理后的图片生成缓存，下次请求不在重复生成，默认缓存7天。
- 如果配置了CDN，处理后的图片会自动推送到CDN节点。
- 如果修改了原图，可以使用v指令重新生成图片和链接。

开通账户和测试环境：

目前为账户提供企业信息和联系人方式，人工审核方式开通。

- 白山云创建user，生成user对应的acceseskey和secretkey，创建user对应的bucket。
- 用户使用acceseskey、secretkey，根据SDK中给出的demo进行编码，即可使用。

用户可提供进一步需求：

- 上传文件类型，图片 视频 文本。
- 平均文件的大小范围。
- 大概存储空间需求。
- 是否需要用CDN。
- 是否需要上传加速。

用户可以通过，自助管理控制台即时查看以下信息？

- 用户空间的信息，包括使用量、使用文件数。
- 用户使用的带宽、流量、请求次数等历史记录。
- 可以查看文件大小分布、文件类型分布等。
- 提供账单信息，包括使用的流量、空间、请求数的计费详情。
- 白山云存储服务用户帮助手册、音视频转换、图片处理等文档。
- 秘钥管理、密码管理。

[TOC]

基础知识普及

Bucket和Object的概念

- Bucket是桶的意思，这个桶里面就可以存放很多文件。
- Object是对象的意思，我们可以把每个文件都作为一个对象。
- 用户可以创建桶、删除桶、查询桶里面的对象、可以设置桶的属性等。
- 用户可以创建一个对象（这个对象就是用户上传的文件）、删除对象、下载对象、设置对象的访问权限等。
- 小栗子：

1. 创建一个桶，桶名叫：baishan_yun，请求的url: PUT /baishan_yun。
2. 上传文件到baishan_yun的桶里，对象名叫：hello_world，每个文件通过PUT方法上传。
3. 要下载这个文件，请求的url: GET /baishan_yun/hello_world。
4. 删除这个文件，请求的url: DELETE /baishan_yun/hello_world
5. 删除桶，请求的url: DELETE /baishan_yun

编程接口相关, 用户使用相关

白山云存储有哪些技术优势？

- 白山云存储是CDN与云存储紧密结合的产品，白山云存储通过CDN边缘节点进行加速上传、通过全网CDN进行加速下载。但白山不强制将两者绑定，用户也可以单独使用云存储服务。
- 白山云存储是多副本、多IDC、分布式架构，保证高可靠性和可用性。
- 白山云存储提供丰富和便捷的图片处理功能，用户可以高效使用，且帮助客户避免重复上传/下载带来的费用。
- 白山云存储是完全自研开发的系统，老版本系统经历了长达7年的稳定运行，支持各种场景的存储需求，新版本更是提供了更高的性能，更丰富的功能。
- 白山云存储坚持以用户为主的原则，可以根据用户的需求进行定制开发，帮助用户更方便的使用存储，并且提供7*24小时的技术人员在线服务，帮助用户迅速解决问题。
- 白山云存储提供多种用户数据导入机制，方便用户将数据迁移到存储上。
- 白山云存储遵循和兼容Amazon S3的接口；可以随意使用Amazon S3的官方SDK，功能强大、性能高，并且用户也可以使用各种开源客

户端、工具等。

- 白山云遵守Amazon S3 REST API接口，并严格遵守HTTP协议，避免接口不一致、标准不一致带来的各种问题，Amazon S3 REST API接口作为行业标准，已经有10多年没有修改了，非常稳定。
- 白山云支持Amazon S3的ACL访问控制，可以管理bucke和object的访问权限。每个bucke和object都有一个附加的ACL子资源。它定义了哪些用户或则组将被授予访问权限。

白山云存储支持的语言和SDK列表？

- 白山云存储兼容Amazon S3的接口。包括所有主流开发语言，例如：Python、Php、Browser、Node.js等。
- 在文档baishancloud_storage_api，有详细的对Python、Php、Browser、Node.js等语言的SDK的实例说明，能帮助开发人员快速入手。其余规则，可参考Amazon S3的API文档。
- Android 文档：[AWS Mobile SDK for Android Documentation](#) SDK下载:aws-android-sdk
- Browser 文档：[Getting Started with the SDK in the Browser](#) SDK下载：Bower
- ISO 文档：[AWS Mobile SDK iOS Developer Guid](#) SDK下载:aws-ios-sdk
- Java 文档：[AWS SDK for Java Developer Guide](#) SDK下载：aws-java-sdk
- .NET 文档：[AWS SDK for .NET Documentation](#) SDK下载:AWSToolsAndSDKForNet.msi
- Node.js 文档：[AWS SDK for JavaScript Documentation](#) SDK下载:Getting Started with the SDK in Node.js
- PHP 文档：[AWS SDK for PHP Documentation](#) SDK下载:Installing via Composer
- Python 文档：[Boto 3 Documentation](#) SDK下载地址及说明：Boto 3 - The AWS SDK for Python
- Ruby 文档：[AWS SDK for Ruby](#) SDK下载地址：[AWS SDK for Ruby Developer Guide v2.16.8](#)
- Go 文档：[AWS SDK for Go](#) SDK下载地址: aws-sdkgo
- C++ 文档：[AWS SDK for C++ Documentation](#) SDK下载地址: aws-sdk-cpp

云存储的Object是以目录结构存储的吗？

白山云存储的Object不是传统意义上的一个文件，存储方式也不是以文件系统保存的。每个Object看上去是一个目录下的文件，但对于云存储只是一个类似目录的字符串。但是这并不影响用户想从指定目录列文件的方

式：

当发送列文件的请求时，设置prefix参数为目录前缀的名字，则可以只列出这个目录下的文件，比如：

- bucket/1.mp4
- bucket/cc/2.mp4
- bucket/cc/3.mp4
- bucket/dd/ee/ff/3.mp4

当请求的 prefix 指定为 cc 时，将会列出：

- 2.mp4
- 3.mp4

跟从目录列文件的体验是一样的。除了模拟目录列文件，还提供指定列出最大的文件数目、指定从某个文件名之后列出文件(按照字母顺序进行排序)、将文件按公共前缀进行合并等功能，详细可参照API文档。

使用的时候需要注意的问题：

- Object 名字最好不要用 '/' 开头 比如： 需要上传一个叫file.txt文件，bucket 是 sandbox ，
 - object 设置为 file.txt , 访问的时候应该是
`http://ss.bscstorage.com/sandbox/file.txt` (推荐方式)
 - object 如果设置为 /file.txt , 访问的时候应该是
`http://ss.bscstorage.com/sandbox//file.txt` (强烈不推荐)
- Object 名字中最好不要有两个连续的 '/' 比如： 需要在folder目录下上传一个叫file.txt文件，bucket 是 sandbox ，
 - object 设置为 folder/file.txt , 访问的时候应该是
`http://ss.bscstorage.com/sandbox/folder/file.txt` (推荐方式)
 - object 如果设置为 folder//file.txt , 访问的时候应该是
`http://ss.bscstorage.com/sandbox/folder//file.txt` (强烈不推荐)
 - 为什么白山云存储不建议在URL中带有2个'/'：某些CDN的cache server可能会把URL中两个连续的'/'合并为一个'/'来回源，这个时候就会出现验证失败或者404

如何删除一个目录？

白山云存储删除目录的方法为： 白山云存储不支持直接删除一个目录，因为 Amazon S3 API 没有严格意义上的目录的概念(没有目录层级的结构,但习惯上把 Object 按照"/"分隔当做目录来使用),这种情况下,对多个文件操作的原子性方面没法保证, 所以这方面都是写脚本做的.

可以通过以下方式完成该操作：

- 通过List Objects接口列出该目录的所有文件。
- 通过Delete Object接口逐一进行删除。
- 删除目录。

请注意：

- List Objects接口返回的文件列表不全部是该目录下的文件（列表是根据文件名字符排序的），可能包含下一个或几个目录；所以需要根据实际情况调整max-keys参数的值，并且需要对每一个文件进行筛选，确认是该目录的文件；如果设置 prefix 参数的话，就可以严格的列出以该 prefix 下所有文件。
- 一次删除（列出所有文件，逐一删除）不能保障该目录的文件都被清理完成 在删除的过程中可能会有新上传，所以整个过程不能保持原子性。

是否可以使用白山云存储做整站的备份？

白山云CDN提供整站备份服务，但目前尚未与云存储整合。

用户如何导入数据？

白山云存储提供多种方式，方便用户导入数据。

- 用户使用SDK上传文件接口，将文件上传到存储上。
- 用户使用SDK生成离线下载任务，由存储的离线下载子系统负责将数据从源站进行下载，同步到存储上。
- 用户配置404回源的规则，当用户访问该用户的bucket，会自动去源站将数据抓取下来，同步到存储上。
- 用户提供文件的下载列表，白山云存储通过文件列表，将数据下载并同步到存储上。
- 用户邮寄硬盘的方式。

404回源，如果存储上文件不存在，是否可以自动让存储去特定的地方拉取？

如果白山云存储上不存在该文件，可以配置指定去某个域名拉取文件。

文件的拉取是异步的，拉取完成时间和拉取的文件大小相关。

同时该请求会返回302,location 响应头的值是文件拉取的地址。

使用拉取功能时，需要提供：

- 回源的域名domain。
- 请求uri到回源拉取文件uri的映射规则。
- 回源请求的校验方式（可选）。
- 是否启用HEAD请求回源拉取文件。

如何使用离线下载同步数据？

离线下载服务支持HTTP协议、BT协议、磁力链接等。

通过提交离线下载任务，离线下载服务会根据任务中提供的文件地址，从网站下载文件，并同步到存储上。同步完成后，给用户发送操作结果。

支持下载任务的状态查询、任务取消、下载数据损坏检查，消息秒传等功能高性能，性能达到2000多rps/s。

如何使用：

- 使用SDK，向offline.bscstorage.com发送请求，请求的url：
http://offline_domain/，bucket是要存放文件的桶。
- 发送请求的body体为离线的任务，比如：

```
{
    "Url": "http://www.abc.com/download/movie.mp4",
    "Key": "movies/2016/movie.mp4",
    "SHA1": "abcdeabcdeabcdeabcdeabcdeabcdeabcde",
    "MD5": "abcdabcdabcdabcdabcdabcdabcd",
    "CRC32": "abcdef",
    "SuccessCallbackUrl": "http://website/succ/",
    "FailureCallbackUrl": "http://website/fail/",
}
```

- 用户需要起一个服务用于接收离线下载服务发送的离线下载结果。

多文件上传是否支持？

不支持一次请求提交多个文件。

如果用户是为了效率考虑，想一个请求提交多个文件，建议客户使用TCP的长连接，多次发送上传文件的请求，这样避免建立多次连接，效率和一次请求提交多个文件差不多。

大文件如何上传，有什么限制？

- 白山云存储的总数据容量和数据对象数量不受限制。各个数据对象的大小可在 1 字节至 1 TB 之间。
- 可在单个PUT中上传的最大数据对象为 1TB。
- 对于大文件建议使用分片上传接口对大文件分片上传，分片上传的限制是：
 - 每个分片最大 512MB。
 - 最多分片数是2000。
 - 最大上传文件大小是1TB。
- 大文件上传主要分为3步：
 - 首先调用分片上传初始化接口，请求返回一个uploadId, 后续2步需要这个uploadId标识每个分片上传。

- 将大文件分成若干片，调用分片上传接口，上传每个分片，请求中通过partNuber参数标识这个块的顺序。
- 上传完成后，调用分片完成接口，通知存储，文件分片上传完成。

访问控制权限有哪些？

- 被授权者可以是某个用户、某个组、所有用户
- 对于Bucket的控制权限：
 - 可读：允许被授权者列出bucket中所有的对象。
 - 可写：允许被授权者创建、覆盖和删除存储桶中的对象。
 - 可读ACL：允许被授权者读取存储桶的ACL。
 - 可写ACL：允许被授权者修改存储桶的ACL。
 - 全部权限：允许被授权者对存储桶的可读、可写、可读ACL、可写ACL。
- 对于Object的控制权限：
 - 可读：允许被授权者读取对象的数据和meta信息。
 - 可读ACL：允许被授权者读取对象的ACL。
 - 可写ACL：允许被授权者修改对象的ACL。
 - 全部权限：允许被授权者在对象上的可读、可读ACL、可写ACL。

是否可以不用签名操作操作bucket和object(匿名访问)？

正常的操作都需要签名进行用户权限的验证，但是也允许设置共有属性，让用户不用签名来操作。

- 将bucket权限设置成public-read之后，任何用户都可以列出这个bucket下的文件，请求不需要签名。
- 将bucket权限设置成public-read-write之后，任何用户都可以列出这个bucket下的文件，并且可以上传文件到该bucket下，请求不需要签名。
- 当上传文件时，指定Object的权限是public-read之后，任何用户都可以下载该文件，不需要签名，即所谓的匿名下载。
- 当用户通过CDN回源来访问存储时，因为给CDN配置了超级权限，所以请求不需要签名也可以访问存储上的文件。

存储支持解压客户的文件包吗？

- 现在不支持，如果项目规模足够大，可以定制开发。

存储下载支持断点续传吗？

- 支持，使用HTTP1.1协议(RFC2616)中Range和Content-Range头域来实现断点续传下载，具体头域设置参考[rfc2616](#)
-

视频处理

转码对源文件格式有要求吗？

一般是没有要求的，基本所有格式都是支持的。

转码模版是描述输入文件还是输出文件？

转码模版是配置输出文件的格式等信息，输入文件无需配置任何东西。

转码模版都配置了什么？

- 封装格式，flv,mp4,ts等
- 视频编码格式，h265, h264, mpeg2等
- 视频码率
- 视频分辨率
- 视频关键帧间隔
- 视频帧率
- 音频编码格式，aac, mp2, mp3等
- 音频码率
- 音频声道个数

转码支持的输出格式？

- flv
- fmp4
- mp4
- ts
- mp3
- wma
- wmv
- mp2
- aac
- gif

是否支持视频切片？

支持，创建转码任务指定切片片段时长即可，前提是模版配置的封装格式ts。

是否支持视频水印？

支持，水印位置信息在转码模版中配置，创建转码任务添加水印图片即可。

是否支持视频剪辑？

支持，创建转码任务指定开始时间和持续时间即可。

hls切片是否支持AES128加密？

支持，创建转码提供加密key、解密url、加密iv向量（可选）即可。

是否支持4K转码？

支持，模版中配置4k分辨率即可自动转码自动转码表示上传文件的同时针对这个文件系统自动创建转码任务。输出文件在存储的路径，比如源文件是a/b/xx.mp4 若配置和源文件路径保持一致，文件后缀配置为foo.ts，在存储中的key就是a/b/xxfoo.ts 若没有配置和源文件路径保持一致，文件后缀配置为foo.ts，在存储中key就是xxfoo.ts

时长1h的视频转码需要多长时间？

转码需要45min-60min左右，如果视频编码和音频编码不变（称为转封装，可以理解为只是后缀名字变了），需要3min-5min，这个只是预估时间，高分辨率和特殊格式时间需要更长。

视频切片生成m3u8文件可否直接通过云存储直接播放？

正常是无法播放的。m3u8文件存放的是一堆小ts文件的列表，播放的时候播放器需要从存储下载小ts文件，但是下载是需要签名的。将小ts文件全部设置为匿名可访问，是可以播放的。通过cdn，云存储作为cdn的源站，是可以播放的。

手动创建了转码任务，完成后找不到目标文件？

有可能没有指定输出文件前缀，目标文件放到bucket根目录了

图片处理

白山存储支持实时图片转换,旋转,缩放,剪切,打水印吗？

- 支持，详见文档。
- 处理后的图片生成缓存，下次请求不在重复生成，默认缓存7天。
- 如果配置了CDN，处理后的图片会自动推送到CDN节点。
- 如果修改了原图，可以使用v指令重新生成图片和链接。

是否可以让存储自动判断user-agent触发特定的图片剪裁输出？

暂不支持，如有需求可以开发

是否有视频转码功能？

- 支持flv、mp4、mov等格式之间的转换
- 支持降低码率、分辨率、修改gop(关键帧间隔)等功能
- 支持视频截图。

该服务需要和厦门同事配合，目前正在开发中，后续会提供服务。

用户管理存储资源相关，管理控制台

存储用户可以通过后台查看到哪些信息？

后台可以查看到的信息包括：

- 用户空间的信息，包括使用量、使用文件数。
- 可以查看用户的带宽、流量、请求次数等。
- 可以查看文件大小分布、文件类型分布等。
- 提供账单信息，包括使用的流量、空间、请求数的计费详情。
- 提供图片处理等文档，帮助用户使用。
- 提供秘钥管理。

存储控制台有没有搜索文件功能，如某个bucket下文件全路径是abc/def/123.png，在不知道此全路径的情况下，能否使用文件名123.png搜索该文件？

目前在控制台暂时没有开发搜索功能，可以在每个bucket下面使用head命令查找文件。

能否统计某bucket下面的文件数量？

文件数量可以从查看数据库中的bucket的文件数，但是有误差。

白山云存储是否提供日志下载？

暂不支持，如有需求，后续可开发。

性能, 可靠性, 可用性相关

白山云存储是否做了上传加速？

白山云存储通过和CDN的边缘节点配合使用，提升上传的速度。

用户请求加速上传域名，由DNS服务器智能将距离用户最近的CDN边缘节点的ip返给用户。

用户将文件上传到最近的边缘节点，边缘节点通过最优路径，将数据传输到存储中心。

通过边缘节点加速上传和直接上传到存储中心的对比，平均速度提升一倍多，极大地改善了上传的体验。

使用上传加速很简单，只需要使用加速上传域名即可。

白山云存储是否整合了cdn支持？

白山云存储和白山CDN是完美整合，申请存储和CDN可以同时进行，有完善的配置流程

- CDN上传加速
- CDN支持虚拟域名回源
- CDN支持签名回源，保护用户的数据安全
- 同时白山云存储还支持第三方CDN回源
- CDN和云存储的配置方法：

如果要测试通过CDN进行下载加速的情况，需要将用户要访问的已备案的域名CN，例如www.qq.com.i.qingcdn.com，CDN回源至ss.bscstorage.com。

创建bucket的建议：

1、以用户域名创建同名的bucket。

例如用户网站域名www.qq.com，那么创建www.qq.com这样一个bucket，这时

2、用户自定义bucket的名字：

例如用户网站域名www.qq.com，用户要创建一个qq的bucket，那么CDN在配置

https是否支持？如何使用？

- 支持https方式。
- 如果是用户域名，首先用户域名需要备案，其次用户需要申请域名的SSL证书，然后将证书发给白山云，白山云配置完成后即可使用。

- 如果是使用白山云存储提供的域名，例如：
<https://ss.bscstorage.com>，注：必须将bucket放在url请求中，不要放在域名中。

加密存储是否支持？

暂不支持

可靠性是多少？

- 白山云存储通过多机房、多副本、快速的故障恢复等机制，保证可靠性达到13个9。

售前支持, 测试等

快速测试的环境? 申请流程是什么?

目前为人工审核及开通。

- 申请客户的公司全称。
- 需要用户提供邮箱（登陆邮箱）。
- 用户登录存储控制台的初始密码。
- 为更好的配合测试，需要用户提供如下信息：
 - 上传文件类型，图片 视频 文本。
 - 平均文件的大小范围。
 - 大概存储空间需求。
 - 是否需要用CDN。
 - 是否需要上传加速。
- 研发同事创建user，生成user对应的accseskey和secretkey，创建user对应的bucket即可。
- 用户使用accseskey、secretkey，根据SDK中给出的demo进行编码，即可使用。

如何申请使用上传加速?

使用上传加速域名、调用上传文件的SDK即可。

如何同时申请存储和cdn服务?

- CDN用户申请地址：<http://www.qingcdn.com/account/index/login>
- 云存储用户，人工申请。

源存储如何设置缓存时间?

白山云存储可以设置所存储文件的Cache-Control等HTTP请求头的设置，并将缓存设置与HTTP访问请求一起下发。

白山云CDN可以按源站返回HTTP请求头里的Cache-Control设置执行缓存策略。

与第三方服务整合相关

是否有整合文本编辑器的上传图片?

有客户提出整网页上传功能，暂不支持此功能。

日常使用相关

白山云存储是否提供web管理界面让用户管理自己的文件?

提供，管理界面登陆地址：<https://cwn-ss.portal.baishancloud.com>

如何上传文件？是否可以通过 ftp 或 rsync 的方式上传？是否有客户端？

白山云存储遵循和兼容Amazon S3 REST API。

社区中有非常多成熟可靠的SDK和命令行工具，兼容Amazon S3 REST API，所以可以直接使用，可完全省去接口层的开发工作。例如流行的：

- [s3cmd](#)
- [boto](#)

白山云存储不支持 ftp 或 rsync 协议。因为amazon s3对文件权限的控制粒度上更精细。图形界面的ftp替代品可以使用Mac下的cyberduck和crossftp或Windows下的s3browser和crossftp。命令行方式的ftp 或 rsync 的替代品可以使用 [s3cmd](#) 或[aws cli](#)

存储是否支持将bucket挂载到linux主机上，可以像访问本地文件一样访问bucket中的文件？可以使用第三方工具

- [s3fs](#)
 - [riofs](#)
 - [googys](#)
-

SDK 相关

在使用java SDK上传文件时报错：Unable to verify integrity of data upload. Client calculated content hash (contentMD5: E+6KS0B2pNPJ272XbG92fw== in base 64) didn't match hash (etag: 359740e5918c395b6e35a2c612582e42 in hex) calculated by Amazon S3

Java SDK 在上传文件时， 默认使用“ChunkedEncoding”的方式， 我们暂不支持这种方式， 您可以通过参数设置来禁用这种上传方式， 可参考下面的例子。

```
BasicAWSCredentials awsCreds = new BasicAWSCredentials(  
    accessKey, secretKey);  
AmazonS3 s3 = new AmazonS3Client(awsCreds);  
  
s3.setS3ClientOptions(S3ClientOptions.builder().disableChur
```

在使用android SDK上传文件时报错：Unable to verify integrity of data upload. Client calculated content hash didn't match hash calculated by Amazon S3. You may need to delete the data stored in Amazon S3

android SDK 在上传文件时， 默认使用“ChunkedEncoding”的方式， 我们暂不支持这种方式， 虽然android SDK提供了接口禁用这种方式， 但是设置为禁用后， 仍然不能生效， 因此需要通过使用v2签名来绕开这个问题。“ChunkedEncoding”是只有v4签名的时候才会使用的。 设置使用v2签名的方法可以参考以下代码。

```
AWSCredentials credentials = new BasicAWSCredentials(  
    accessKey, secretKey);  
  
ClientConfiguration configuration = new ClientConfiguration();  
configuration.setSignerOverride("S3SignerType");  
  
AmazonS3 s3 = new AmazonS3Client(credentials, configuration);
```

在使用ios SDK上传,下载文件时报错: Code: 403; Error Code: SignatureDoesNotMatch; Request ID: 23e338d9-1706-1410-3759-a0369fd80cca

ios SDK 会根据endpoint指定的域名来判断服务类型, 如如果域名以's3'开头 (s3.amazonaws.com)表示请求的是s3服务, 则 使用对应的签名方式, 因此, 当指定非AWS的标准域名时, 会导致使用不正确签名方式。另外ios SDK可配置参数较少, 只能 通过修改源代码改变SDK行为。源代码github地址: '<https://github.com/aws/aws-sdk-ios.git>'。需在文件 `AWSCore/Authentication/AWSSignature.m` 第319行处插入新的一行: '`[request setValue:contentSha256 forHTTPHeaderField:@"x-amz-content-sha256"];`'。然后在源代码根目录执行以下命令编译源代码:

```
chmod +x Scripts/SdkPackage.sh
sh Scripts/Package.sh
```

由于只需要重新编译AWSCore, 您可以编辑Scripts/Package.sh文件, 删掉编译其他模块的代码, 删掉后文件末尾如下:

```
if [ -x "Scripts/SdkPackage.sh" ]; then
    Scripts/SdkPackage.sh AWSCore
fi
```

编译好的framework位于builtFramework/framework目录, 使用新编译得到的AWSCore.framework替换项目中原来的即可。

从Maven Repository下载AWS SDK时速度太慢, 且容易断线。可以在pom.xml中加入以下配置, 更换镜像。

```
<repositories>
  <repository>
    <id>my-repo1</id>
    <name>your custom repo</name>
    <url>http://mirrors.redv.com/maven2</url>
  </repository>
</repositories>
```

如何让SDK使用指定版本的签名 (V2签名或V4签名) 。

java SDK

java SDK 默认使用V4签名，如果要使用V2签名，可参考下面的代码进行设置

```
import com.amazonaws.ClientConfiguration;

ClientConfiguration clientconfiguration = new ClientConfiguration();
clientconfiguration.setSignerOverride("S3SignerType");
AmazonS3 client = new AmazonS3Client(awsCreds, clientconfigurati
```

python SDK

python SDK(boto3) 默认使用V2签名，如果要使用V4签名，可参考下面的代码进行设置

```
from botocore.client import Config

config = Config(signature_version='s3v4')
client = boto3.client(
    's3',
    aws_access_key_id = access_key,
    aws_secret_access_key = secret_key,
    config = config,
    region_name = 'us-east-1',
    endpoint_url = 'http://bscstorage.com',
)
```

php SDK

php SDK 只支持V4签名，不能使用V2签名

go SDK

go SDK 只支持V4签名，不能使用V2签名

JavaScript SDK

JavaScript SDK 默认使用V2签名，如果要使用V4签名，可添加下面的代码

```
AWS.config.signatureVersion = 'v4';
```

对象存储

实例（兼容Amazon SDK）

Python Demo

安装AWS Python 客户端boto3

```
pip install boto3
```

初始化，设置帐号信息和域名

```
import boto3
from boto3.s3.transfer import TransferConfig

cli = boto3.client(
    's3',
    aws_access_key_id='ziw5dp1alvty9n47qksu', #请替换为您自己
    aws_secret_access_key='V+TZ5u5wNvXb+KP5g0dMNzhMeWe372,
    endpoint_url='http://ss.bscstorage.com'
)
```

文件操作接口

上传文件

使用put_object接口上传

ACL可设置为：'private' 或 'public-read' 或 'public-read-write' 或
'authenticated-read'

```
resp = cli.put_object(
    ACL='public-read',
    Bucket='test-bucket-xxx',
    Key='test-key-xxx',
    ContentType='image/jpeg', # 请替换为合适的文件类型
    Body='the content of the file as a string'
)
```

使用upload_file接口上传（适合大文件上传，支持自动分块，多块并行上传）

```

config = TransferConfig(
    multipart_threshold=30 * 1024 * 1024,
    multipart_chunksize=8 * 1024 * 1024,
    max_concurrency=10
)
resp = cli.upload_file(
    '/root/test.mp4',
    'test-bucket-xxx',
    'test-key-xxx',
    ExtraArgs={
        'ContentType': 'image/jpeg', # 请替换为合适的文件类型
        'ACL': 'private',
    },
    Config=config
)

```

copy文件

copy源bucket中所有以 aa 前缀的文件到目标bucket中

```

marker = ''

while True:
    resp = s3.list_objects(
        Bucket='src-bucket',
        Prefix='aa',

        Marker=marker,
    )

    if 'Contents' not in resp:
        break

    for content in resp['Contents']:
        s3.copy_object(Bucket='dst-bucket', Key=content['Key'])

    marker = resp['Contents'][-1]['Key']

```

下载文件

```

resp = cli.get_object(
    Bucket='test-bucket-xxx',
    Key='test-key-xxx'
)

```

获取文件的URL

获取已签名的URL用来下载文件，可通过参数ExpiresIn设置签名过期时间。

```
url = cli.generate_presigned_url(  
    'get_object',  
    Params={  
        'Bucket': 'test-bucket-xxx',  
        'Key': 'test-key-xxx'  
    },  
    ExpiresIn=60  
)  
print url
```

删除文件

```
resp = cli.delete_object(  
    Bucket='test-bucket-xxx',  
    Key='test-key-xxx'  
)
```

获取文件的ACL

```
resp = cli.get_object_acl(  
    Bucket='test-bucket-xxx',  
    Key='test-key-xxx'  
)
```

设置文件的ACL

使用预定义的ACL

支持的预定义ACL有：'private', 'public-read', 'public-read-write' 或 'authenticated-read'

```
resp = cli.put_object_acl(  
    ACL='public-read',  
    Bucket='test-bucket-xxx',  
    Key='test-key-xxx'  
)
```

使用自定义的ACL

可指定的Permission包括: 'FULL_CONTROL', 'WRITE', 'WRITE_ACP', 'READ', 'READ_ACP'

```
resp = cli.put_object_acl(
    AccessControlPolicy={
        'Grants': [
            {
                'Grantee': {
                    'ID': 'user_foo', # 请替换为真实存在的用户
                    'Type': 'CanonicalUser',
                },
                'Permission': 'WRITE',
            },
            {
                'Grantee': {
                    'ID': 'your-user-name',
                    'Type': 'CanonicalUser',
                },
                'Permission': 'FULL_CONTROL',
            },
        ],
        'Owner': {
            'ID': 'your-user-name',
        },
    },
    Bucket='test-bucket-xxx',
    Key='test-key-xxx'
)
```

桶操作接口

创建桶

ACL可设置为: 'private' 或 'public-read' 或 'public-read-write' 或 'authenticated-read'

```
resp = cli.create_bucket(
    ACL='public-read',
    Bucket='test-bucket-xxx'
)
```

列出桶中所包含的文件, 每次最多可以返回1000个文件

```
resp = cli.list_objects(  
    Bucket='test-bucket-xxx',  
    Prefix='',  
    Marker='',  
)
```

列出桶中所包含的所有文件

```
marker = ''  
  
while True:  
    resp = s3.list_objects(  
        Bucket='test-bucket-xxx',  
        Marker=marker,  
    )  
  
    if 'Contents' not in resp:  
        break  
  
    for content in resp['Contents']:  
        print 'key: %s, size: %d' % (content['Key'], content['Size'])  
  
    marker = resp['Contents'][~-1]['Key']
```

删除桶

```
resp = cli.delete_bucket(  
    Bucket='test-bucket-xxx'  
)
```

获取桶的ACL

```
resp = cli.get_bucket_acl(  
    Bucket='test-bucket-xxx'  
)
```

设置桶的ACL

使用预定义的ACL

支持的预定义ACL有: 'private', 'public-read', 'public-read-write' 或 'authenticated-read'

```
resp = cli.put_bucket_acl(
    ACL='public-read',
    Bucket='test-bucket-xxx',
)
```

使用自定义的ACL

可指定的Permission包括: 'FULL_CONTROL', 'WRITE', 'WRITE_ACP', 'READ', 'READ_ACP'

```
resp = cli.put_bucket_acl(
    AccessControlPolicy={
        'Grants': [
            {
                'Grantee': {
                    'ID': 'user_foo', # 请替换为真实存在的用户
                    'Type': 'CanonicalUser',
                },
                'Permission': 'WRITE',
            },
            {
                'Grantee': {
                    'ID': 'your-user-name',
                    'Type': 'CanonicalUser',
                },
                'Permission': 'FULL_CONTROL',
            },
        ],
        'Owner': {
            'ID': 'your-user-name',
        },
    },
    Bucket='test-bucket-xxx'
)
```

服务操作接口

列出所有的桶

```
resp = cli.list_buckets()
```

AWS 官方 SDK [aws-sdk-python](#) 接口详细文档 [api-reference](#)

PHP Demo

系统需求：

- PHP >= 5.5.0

安装AWS SDK for PHP

```
curl -O http://docs.aws.amazon.com/aws-sdk-php/v3/download,
```

初始化，设置帐号信息和域名

```
require 'aws.phar';
$cli = new Aws\S3\S3Client([
    'version' => 'latest',
    'region' => 'us-east-1',
    'credentials' => [
        'key' => 'z2qutjf718d0i9gw6skc', //请替换为您自己的a
        'secret' => 'SEQgcc1ppH7uXPG4ZPIcrCv2cWz8grcReMfFAE
    ],
    'endpoint' => 'http://ss.bscstorage.com',
]);

```

文件操作接口

上传文件

ACL可设置为：'private' 或 'public-read' 或 'public-read-write' 或 'authenticated-read' 如果需要上传的文件内容已经在内存中,可以直接使用'Body'指定。如果文件在磁盘上，可以通过'SourceFile'指定文件名，'Body'和'SourceFile'不能同时使用。

```
$resp = $cli->putObject([
    'ACL' => 'public-read',
    'Bucket' => 'test-bucket-xxx',
    'Key' => 'test-key-xxx',
    'ContentType' => 'image/jpeg', //请替换为合适的文件类型
    'Body' => 'file content as a string',
    //'SourceFile' => '/root/test.jpg',
]);

```

下载文件

```
$resp = $cli->getObject([
    'Bucket' => 'test-bucket-xxx',
    'Key' => 'test-key-xxx',
]);
```

获取文件的URL

获取已签名的URL用来下载文件，可设置签名过期时间。

```
$cmd = $cli->getCommand('GetObject', [
    'Bucket' => 'test-bucket-xxx',
    'Key' => 'test-key-xxx',
]);
$req = $cli->createPresignedRequest($cmd, '+10 seconds');
$url = (string) $req->getUri();
```

删除文件

```
$resp = $cli->deleteObject([
    'Bucket' => 'test-bucket-xxx',
    'Key' => 'test-key-xxx',
]);
```

获取文件的ACL

```
$resp = $cli->getObjectAcl([
    'Bucket' => 'test-bucket-xxx',
    'Key' => 'test-key-xxx',
]);
```

设置文件的ACL

使用预定义的ACL

支持的预定义ACL有：'private', 'public-read', 'public-read-write' 或
'authenticated-read'

```
$resp = $cli->putObjectAcl([
    'ACL' => 'public-read-write',
    'Bucket' => 'test-bucket-xxx',
    'Key' => 'test-key-xxx',
]);
```

使用自定义的ACL

可指定的Permission包括: 'FULL_CONTROL', 'WRITE', 'WRITE_ACP', 'READ', 'READ_ACP'

```
$resp = $cli->putObjectAcl([
    'AccessControlPolicy' => [
        'Grants' => [
            [
                'Grantee' => [
                    'ID' => 'user_foo', //请替换为真实存在的用
                    'Type' => 'CanonicalUser',
                ],
                'Permission' => 'WRITE',
            ],
            [
                'Grantee' => [
                    'ID' => 'your-user-name',
                    'Type' => 'CanonicalUser',
                ],
                'Permission' => 'FULL_CONTROL',
            ],
        ],
        'Owner' => [
            'ID' => 'your-user-name',
        ],
    ],
    'Bucket' => 'test-bucket-xxx',
    'Key' => 'test-key-xxx',
]);

```

桶操作接口

创建桶

ACL可设置为: 'private' 或 'public-read' 或 'public-read-write' 或 'authenticated-read'

```
$resp = $cli->createBucket([
    'Bucket' => 'test-bucket-xxx',
    'ACL' => 'public-read',
]);

```

列出桶中所包含的文件, 每次最多返回1000个文件

```
$resp = $cli->listObjects([
    'Bucket' => 'test-bucket-xxx',
    'Prefix' => '',
    'Marker' => '',
]);
```

列出桶中所包含的所有文件

```
$marker = '';
while (true):
    $resp = $cli->listObjects([
        'Bucket' => 'test-bucket-xxx',
        'Marker' => $marker,
    ]);

    if($resp['Contents'] == NULL)
    {
        break;
    }

    foreach($resp['Contents'] as $content)
    {
        var_dump($content['Key']);
        $marker = $content['Key'];
    }
endwhile;
```

删除桶

```
$resp = $cli->deleteBucket([
    'Bucket' => 'test-bucket-xxx',
]);
```

获取桶的ACL

```
$resp = $cli->getBucketAcl([
    'Bucket' => 'test-bucket-xxx',
]);
```

设置桶的ACL

使用预定义的ACL

支持的预定义ACL有: 'private', 'public-read', 'public-read-write' 或
'authenticated-read'

```
$resp = $cli->putBucketAcl([
    'ACL' => 'public-read-write',
    'Bucket' => 'test-bucket-xxx',
]);
```

使用自定义的ACL:

可指定的Permission包括: 'FULL_CONTROL', 'WRITE', 'WRITE_ACP',
'READ', 'READ_ACP'

```
$resp = $cli->putBucketAcl([
    'AccessControlPolicy' => [
        'Grants' => [
            [
                'Grantee' => [
                    'ID' => 'user_foo', //请替换为真实存在的用
                    'Type' => 'CanonicalUser',
                ],
                'Permission' => 'WRITE',
            ],
            [
                'Grantee' => [
                    'ID' => 'your-user-name',
                    'Type' => 'CanonicalUser',
                ],
                'Permission' => 'FULL_CONTROL',
            ],
            [
                'Owner' => [
                    'ID' => 'your-user-name',
                ],
            ],
            'Bucket' => 'test-bucket-xxx',
        ]);
]);
```

服务操作接口

列出所有的桶

```
$resp = $cli->listBuckets([
]);
```

Python

AWS 官方 SDK [aws-sdk-php](#) 接口详细文档 [api-reference](#)

PHP Demo (v2)

系统需求：

- PHP 5.3.3+ compiled with the cURL, JSON, and XML extensions
- A recent version of cURL 7.16.2+ compiled with OpenSSL and zlib

安装AWS SDK for PHP

```
curl -O http://docs.aws.amazon.com/aws-sdk-php/v2/download,
```

初始化，设置帐号信息和域名

```
require 'aws.phar';

$cli = Aws\S3\S3Client::factory(array(
    'endpoint' => 'http://ss.bscstorage.com',
    'credentials' => array(
        'key' => 'your access key',
        'secret' => 'your secret key',
    ),
    'region' => 'us-east-1',
));
```

文件操作接口

上传文件

ACL可设置为：'private' 或 'public-read' 或 'public-read-write' 或
'authenticated-read' Body 为文件的内容

```
$resp = $cli->putObject(array(
    'ACL' => 'public-read',
    'Bucket' => 'test-bucket-xxx',
    'Key' => 'test-key-xxx',
    'ContentType' => 'image/jpeg',
    'Body' => 'file content as a string'
));
```

使用upload接口上传，适合大文件上传，支持自动分片上传

```
$resp = $cli->upload('test-bucket-xxx',
    'test-key-xxx', fopen('path/to/my/file/test.txt', 'r'))
```

上传整个目录, 参数分别为需要上传的本地目录, **bucket**的名字, 添加的前缀

```
$resp = $cli->uploadDirectory('/root/mydata_dir',
    'test-bucket-xxx',
    ''
);
```

下载文件

```
$resp = $cli->getObject(array(
    'Bucket' => 'test-bucket-xxx',
    'Key' => 'test-key-xxx',
));
```

下载文件到本地目录, 参数分别是文件保存目录, **bucket**的名字, 要下载的文件的前缀

```
$resp = $cli->downloadBucket('/root/download_dir',
    'test-bucket-xxx',
    ''
);
```

获取文件的URL

获取已签名的URL用来下载文件, 可设置签名过期时间。

```
$cmd = $cli->getCommand('GetObject', array(
    'Bucket' => 'test-bucket-xxx',
    'Key' => 'test-key-xxx',
));

$url = $cmd->createPresignedUrl('+100 seconds');
```

删除文件

```
$resp = $cli->deleteObject(array(  
    'Bucket' => 'test-bucket-xxx',  
    'Key' => 'test-key-xxx',  

```

获取文件的ACL

```
$resp = $cli->getObjectAcl(array(  
    'Bucket' => 'test-bucket-xxx',  
    'Key' => 'test-key-xxx',  

```

设置文件的ACL

使用预定义的ACL

支持的预定义ACL有: 'private', 'public-read', 'public-read-write' 或
'authenticated-read'

```
$resp = $cli->putObjectAcl(array(  
    'ACL' => 'public-read-write',  
    'Bucket' => 'test-bucket-xxx',  
    'Key' => 'test-key-xxx',  

```

使用自定义的ACL

可指定的Permission包括: 'FULL_CONTROL', 'WRITE', 'WRITE_ACP',
'READ', 'READ_ACP'

```
$resp = $cli->putObjectAcl(array(
    'Grants' => array(
        array(
            'Grantee' => array(
                'ID' => 'user_foo', //请替换为真实存在的用户
                'Type' => 'CanonicalUser',
            ),
            'Permission' => 'FULL_CONTROL',
        ),
    ),
    'Owner' => array(
        'ID' => 'your-user-name',
    ),
    'Bucket' => 'test-bucket-xxx',
    'Key' => 'test-key-xxx',
));
});
```

桶操作接口

创建桶

ACL可设置为：'private' 或 'public-read' 或 'public-read-write' 或
'authenticated-read'

```
$resp = $cli->createBucket(array(
    'Bucket' => 'test-bucket-xxx',
    'ACL' => 'public-read',
));
});
```

列出桶中所包含的文件, 每次最多返回1000个文件

```
$resp = $cli->listObjects(array(
    'Bucket' => 'test-bucket-xxx',
    'Prefix' => '',
    'Marker' => '',
));
});
```

列出桶中所包含的所有文件

```
$marker = '';
while (true):
    $resp = $cli->listObjects(array(
        'Bucket' => 'test-bucket-xxx',
        'Marker' => $marker,
    ));

    if($resp['Contents'] == NULL)
    {
        break;
    }

    foreach($resp['Contents'] as $content)
    {
        var_dump($content['Key']);
        $marker = $content['Key'];
    }
endwhile;
```

删除桶

```
$resp = $cli->deleteBucket(array(
    'Bucket' => 'test-bucket-xxx',
));
```

获取桶的ACL

```
$resp = $cli->getBucketAcl(array(
    'Bucket' => 'test-bucket-xxx',
));
```

设置桶的ACL

使用预定义的ACL

支持的预定义ACL有: 'private', 'public-read', 'public-read-write' 或
'authenticated-read'

```
$resp = $cli->putBucketAcl(array(
    'ACL' => 'public-read-write',
    'Bucket' => 'test-bucket-xxx',
));
```

使用自定义的ACL:

可指定的Permission包括: 'FULL_CONTROL', 'WRITE', 'WRITE_ACP',
'READ', 'READ_ACP'

```
$resp = $cli->putBucketAcl(array(
    'Grants' => array(
        array(
            'Grantee' => array(
                'ID' => 'user_foo', //请替换为真实存在的用户
                'Type' => 'CanonicalUser',
            ),
            'Permission' => 'FULL_CONTROL',
        ),
    ),
    'Owner' => array(
        'ID' => 'your-user-name',
    ),
    'Bucket' => 'test-bucket-xxx',
));
});
```

服务操作接口

列出所有的桶

```
$resp = $cli->listBuckets();
```

AWS 官方 SDK [aws-sdk-php](#) 接口详细文档 [api-reference](#)

Python

JavaScript Demo

```

<html>
<!--将输入文本框中的数据作为文件上传，列出已经上传的文件，下载文件的示例
--不建议将帐号信息放到浏览器端代码中--&gt;
&lt;textarea id="data"&gt;&lt;/textarea&gt;
&lt;button id="upload-button"&gt;Upload to S3&lt;/button&gt;
&lt;div id="results"&gt;&lt;/div&gt;
&lt;script src='https://sdk.amazonaws.com/js/aws-sdk-2.5.0.min.js'&gt;
&lt;script type="text/javascript"&gt;
    AWS.config.update({
        accessKeyId: 'ziw5dp1alvty9n47qksu',      &lt;!--请替换为您自己的Access Key ID
        secretAccessKey: 'V+TZ5u5wNvXb+KP5g0dMNzhMeWe372/'&gt;
    });
    AWS.config.region = 'us-west-1';
    AWS.config.endpoint = 'http://ss.bscstorage.com';
    AWS.config.s3ForcePathStyle = true

    var bucket_name = 'test-bucket'      &lt;!--请替换为您自己的Bucket名称

    var s3 = new AWS.S3({
        params: {
            Bucket: bucket_name
        }
    });

    var textarea = document.getElementById('data');
    var button = document.getElementById('upload-button');
    var results = document.getElementById('results');

    button.addEventListener('click', function() {
        results.innerHTML = '';

        var params = {
            Key: 'data.txt',
            Body: textarea.value
        };
        s3.upload(params, function(err, data) {
            results.innerHTML = err ? 'ERROR!' : 'SAVED.';

            s3.listObjects({
                Bucket: bucket_name
            }, function(err, data) {
                if (err) {
                    console.log(err);
                } else {
                    console.log(data);
                }
            });
        });
    });
&lt;/script&gt;
</pre>

```

Python

```
});  
s3.getObject({  
    Bucket: bucket_name,  
    Key: 'data.txt'  
, function(err, data) {  
    if (err) {  
        console.log(err);  
    } else {  
        console.log(data);  
    }  
});  
  
, false);  
</script>  
</html>
```

```
<html>
<!--上传用户本地文件，并生成文件的URL的示例-->
<!--不建议将帐号信息放到浏览器端代码中-->
<input type="file" id="file-chooser" />
<button id="upload-button">Upload to S3</button>
<div id="results"></div>
<div id="signed_url"></div>

<script src='https://sdk.amazonaws.com/js/aws-sdk-2.5.0.min.js'>
<script type="text/javascript">

    <!--请替换为您自己的access key 和secret key-->
    AWS.config.update({accessKeyId: 'ziw5dp1alvty9n47qksu', secretAccessKey: 'PQJLcZCzXGKUWVfDyfHkRgjBxLJLcZCzXGKUWVfDyfHkRgjBxLJ', region: 'us-west-1', endpoint: 'http://ss.bscstorage.com'}, true);

    var bucket_name = 'test-bucket'      <!--请替换为您自己的bucket name-->
    var s3 = new AWS.S3({params: {Bucket: bucket_name}});

    var fileChooser = document.getElementById('file-chooser');
    var button = document.getElementById('upload-button');
    var results = document.getElementById('results');
    var signed_url = document.getElementById('signed_url');

    button.addEventListener('click', function() {
        var file = fileChooser.files[0];
        if (file) {
            results.innerHTML = '';
            signed_url.innerHTML = '';

            var params = {Key: file.name, ContentType: file.type};
            s3.upload(params, function (err, data) {
                var params_get = { Bucket: bucket_name, Key: file.name };
                var url = s3.getSignedUrl('getObject', params_get, function () {
                    signed_url.innerHTML = err ? 'ERROR!' : 'RETRIEVED!';
                });
                results.innerHTML = err ? 'ERROR!' : 'UPLOADED.';
            });
        } else {
            results.innerHTML = 'Nothing to upload.';
        }
    }, false);
</script>
</html>
```

Python

AWS 官方 SDK [aws-sdk-browser](#)

Node.js Demo

安装AWS SDK for JavaScript in Node.js

```
npm install aws-sdk
```

初始化，设置帐号信息和域名

```
var AWS = require('aws-sdk');
var fs = require('fs');

AWS.config.accessKeyId = 'ziw5dp1alvty9n47qksu'; //请替换为您自己的
AWS.config.secretAccessKey = 'V+TZ5u5wNvXb+KP5g0dMNzhMeWeE';
AWS.config.region = 'us-west-1';
AWS.config.endpoint = 'http://ss.bscstorage.com';
AWS.config.s3ForcePathStyle = true

var s3 = new AWS.S3();
```

文件操作接口

上传文件

使用putObject接口上传

ACL可设置为：'private' 或 'public-read' 或 'public-read-write' 或
'authenticated-read'

```
var params = {
  Bucket: 'test-bucket-xxx',
  Key: 'test-key-xxx',
  ACL: 'public-read',
  Body: new Buffer('blablabla')
};
s3.putObject(params, function(err, data) {
  if (err) console.log(err, err.stack);
  else console.log(data);
});
```

使用upload接口上传（适合大文件上传，支持自动分块，多块并行上传）

```
var file_stream = fs.createReadStream('/root/test.mp4')
var params = {
  Bucket: 'test-bucket-xxx',
  Key: 'test-key-xxx',
  Body: file_stream,
  ACL: 'public-read'
}
var options = {partSize: 10 * 1024 * 1024, queueSize: 10}
s3.upload(params, options, function(err, data) {
  if (err) console.log(err, err.stack);
  else console.log(data);
});
```

下载文件

```
var params = {
  Bucket: 'test-bucket-xxx',
  Key: 'test-key-xxx'
};
s3.getObject(params, function(err, data) {
  if (err) console.log(err, err.stack);
  else console.log(data);
});
```

获取文件的URL

获取已签名的URL用来下载文件，可通过参数Expires设置签名过期时间。

```
var params = {
  Bucket: 'test-bucket-xxx',
  Key: 'test-key-xxx',
  Expires: 60
};
var url = s3.getSignedUrl('getObject', params);
console.log('The URL is', url);
```

删除文件

```
var params = {
    Bucket: 'test-bucket-xxx',
    Key: 'test-key-xxx'
};
s3.deleteObject(params, function(err, data) {
    if (err) console.log(err, err.stack);
    else console.log(data);
});
```

获取文件的ACL

```
var params = {
    Bucket: 'test-bucket-xxx',
    Key: 'test-key-xxx'
};
s3.getObjectAcl(params, function(err, data) {
    if (err) console.log(err, err.stack);
    else console.log(data);
});
```

设置文件的ACL

使用预定义的ACL

支持的预定义ACL有: 'private', 'public-read', 'public-read-write' 或 'authenticated-read'

```
var params = {
    Bucket: 'test-bucket-xxx',
    Key: 'test-key-xxx',
    ACL: 'public-read'
};
s3.putObjectAcl(params, function(err, data) {
    if (err) console.log(err, err.stack);
    else console.log(data);
});
```

使用自定义的ACL

可指定的Permission包括: 'FULL_CONTROL', 'WRITE', 'WRITE_ACP', 'READ', 'READ_ACP'

```

var params = {
    Bucket: 'test-bucket-xxx',
    Key: 'test-key-xxx',
    AccessControlPolicy: {
        Grants: [
            {
                Grantee: {
                    Type: 'CanonicalUser',
                    ID: 'user-foo'
                },
                Permission: 'WRITE'
            },
            {
                Grantee: {
                    Type: 'CanonicalUser',
                    ID: 'your-user-name'
                },
                Permission: 'FULL_CONTROL'
            },
        ],
        Owner: {
            ID: 'your-user-name'
        }
    }
};

s3.putObjectAcl(params, function(err, data) {
    if (err) console.log(err, err.stack);
    else console.log(data);
});

```

桶操作接口

创建桶

ACL可设置为：'private' 或 'public-read' 或 'public-read-write' 或 'authenticated-read'

```

var params = {
    Bucket: 'test-bucket-xxx',
    ACL: 'public-read'
};

s3.createBucket(params, function(err, data) {
    if (err) console.log(err, err.stack);
    else console.log(data);
});

```

列出桶中所包含的文件

```
var params = {
    Bucket: 'test-bucket-xxx',
};
s3.listObjects(params, function(err, data) {
    if (err) console.log(err, err.stack);
    else console.log(data);
});
```

删除桶

```
var params = {
    Bucket: 'test-bucket-xxx',
};
s3.deleteBucket(params, function(err, data) {
    if (err) console.log(err, err.stack);
    else console.log(data);
});
```

获取桶的ACL

```
var params = {
    Bucket: 'test-bucket-xxx',
};
s3.getBucketAcl(params, function(err, data) {
    if (err) console.log(err, err.stack);
    else console.log(data);
});
```

设置桶的ACL

使用预定义的ACL

支持的预定义ACL有: 'private', 'public-read', 'public-read-write' 或
'authenticated-read'

```
var params = {
    Bucket: 'test-bucket-xxx',
    ACL: 'public-read'
};
s3.putBucketAcl(params, function(err, data) {
    if (err) console.log(err, err.stack);
    else console.log(data);
});
```

使用自定义的ACL

可指定的Permission包括: 'FULL_CONTROL', 'WRITE', 'WRITE_ACP', 'READ', 'READ_ACP'

```
var params = {
    Bucket: 'test-bucket-xxx',
    AccessControlPolicy: {
        Grants: [
            {
                Grantee: {
                    Type: 'CanonicalUser',
                    ID: 'user-foo'
                },
                Permission: 'WRITE'
            },
            {
                Grantee: {
                    Type: 'CanonicalUser',
                    ID: 'your-user-name'
                },
                Permission: 'FULL_CONTROL'
            },
        ],
        Owner: {
            ID: 'your-user-name'
        }
    }
};

s3.putBucketAcl(params, function(err, data) {
    if (err) console.log(err, err.stack);
    else console.log(data);
});
```

服务操作接口

列出所有的桶

```
s3.listBuckets(function(err, data) {
    if (err) console.log(err, err.stack);
    else console.log(data);
});
```

AWS 官方 SDK [aws-sdk-node.js](#) 接口详细文档 [api-reference](#)

Java Demo

安装AWS Java SDK

使用maven, 在pom.xml中加入如下内容

```
<dependencyManagement>
    <dependencies>
        <dependency>
            <groupId>com.amazonaws</groupId>
            <artifactId>aws-java-sdk-bom</artifactId>
            <version>1.12.95</version>
            <type>pom</type>
            <scope>import</scope>
        </dependency>
    </dependencies>
</dependencyManagement>

<dependencies>
    <dependency>
        <groupId>com.amazonaws</groupId>
        <artifactId>aws-java-sdk-s3</artifactId>
    </dependency>
</dependencies>
```

初始化，设置帐号信息和域名

```

package com.baishancloud.sdk_test;

import java.io.*;
import java.util.ArrayList;
import java.util.List;
import com.google.gson.*;
import com.amazonaws.auth.BasicAWSCredentials;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3Client;
import com.amazonaws.regions.Region;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.s3.S3ClientOptions;
import com.amazonaws.services.s3.model.*;
import com.amazonaws.ClientConfiguration;

public class App
{
    private static String endPoint = "http://ss.bscstorage.
    private static String accessKey = "ziw5dp1alvty9n47qksu
    private static String secretKey = "V+TZ5u5wNvXb+KP5g0c

    public static void main( String[] args )
    {
        BasicAWSCredentials awsCreds = new BasicAWSCredential:
            accessKey, secretKey);

        ClientConfiguration clientconfiguration = new Client
        clientconfiguration.setSocketTimeout(60 * 60 * 1000);
        clientconfiguration.setConnectionTimeout(60 * 60 *

        AmazonS3 s3 = new AmazonS3Client(awsCreds, clientco
            s3.setRegion(Region.getRegion(Regions.US_EAST_1));
            s3.setEndpoint(endPoint);
            s3.setS3ClientOptions(S3ClientOptions.builder().set
                .disableChunkedEncoding().build()); //暂不支持
    }
}

```

文件操作接口

上传文件

```

String file_content = "bla bla";
InputStream inputStream = new ByteArrayInputStream(file_content.getBytes());

ObjectMetadata objectMetadata = new ObjectMetadata();
objectMetadata.setContentType("image/jpeg"); //请替换为合适的
objectMetadata.addUserMetadata("key-foo", "value_bar");

PutObjectRequest putObjectrequest = new PutObjectRequest(
    "test-bucket", "test-key", inputStream, objectMetadata);

//您可以使用下面两种方式为上传的文件指定ACL，后一种已被注释
putObjectrequest.setCannedAcl(CannedAccessControlList.Public);

//AccessControlList accessControlList = new AccessControlList();
//accessControlList.setOwner(new Owner("your_user_id", ""));
//accessControlList.grantPermission(GroupGrantee.AllUsers,
//accessControlList.grantPermission(GroupGrantee.AllUsers,
//accessControlList.grantPermission(GroupGrantee.AuthenticatedUsers));
//putObjectrequest.setAccessControlList(accessControlList);

PutObjectResult putObjectResult = s3.putObject(putObjectrequest);

```

copy文件

```
CopyObjectResult res = s3.copyObject("srcBucketName", "srcKey", "destBucketName", "destKey");
```

下载文件

```

GetObjectRequest get0bjectRequest = new GetObjectRequest("test-bucket", "test-key");
S3Object s3object = s3.getObject(get0bjectRequest);

System.out.println(s3object.getObjectMetadata().getUserMetadata());
byte[] buf = new byte[1024];
try {
    int n = s3object.getObjectContent().read(buf, 0, 1024);
    System.out.println(new String(buf, "UTF-8"));
} catch (IOException e) {
    System.out.println("errr");
}

```

下载到本地文件

```

GetObjectRequest get0bjectRequest = new GetObjectRequest("test-bucket", "test-key");
ObjectMetadata meta = s3.getObject(get0bjectRequest, new File("localfile"));

```

获取文件的URL

获取已签名的URL用来下载文件，可通过参数设置签名过期时间。

```
URL url = s3.generatePresignedUrl("test-bucket", "test-key",
                                    new Date(116, 11, 17));
System.out.println(url.toString());
```

删除文件

```
s3.deleteObject("test-bucket", "test_key");
```

获取文件的ACL

```
AccessControlList accessControlList = s3.getObjectAcl("test-bucket", "test-key");
```

设置文件的ACL

使用预定义的ACL

```
s3.setObjectAcl(bucketName, "test-key", CannedAccessControlList.BUCKET_OWNER_FULL_CONTROL);
```

使用自定义的ACL

```
AccessControlList accessControlList = new AccessControlList();
accessControlList.setOwner(new Owner("your_user_id", ""));

accessControlList.grantPermission(GroupGrantee.AuthenticatedUsers, "READ");
accessControlList.grantPermission(GroupGrantee.LogDelivery, "READ");
accessControlList.grantPermission(new CanonicalGrantee("some-user-id"), "READ");
accessControlList.grantPermission(new EmailAddressGrantee("user@example.com"), "READ");

s3.setObjectAcl("test-bucket", "test-key", accessControlList);
```

桶操作接口

创建桶

```
CreateBucketRequest createBucketRequest = new CreateBucketRequest("test-bucket");
createBucketRequest.withCannedAcl(CannedAccessControlList.BUCKET_OWNER_FULL_CONTROL);

Bucket bucket = s3.createBucket(createBucketRequest);
```

列出桶中所包含的文件, 每次最多返回1000个文件

```
ListObjectsRequest listObjectsRequest = new ListObjectsRequest();
listObjectsRequest.setBucketName("test-bucket");
listObjectsRequest.setMarker("foo"); //设置从哪个key开始列
listObjectsRequest.setPrefix("foo"); //只返回以“foo”为前缀的k
listObjectsRequest.setDelimiter("/"); //对有公共部分的keys进行
listObjectsRequest.setMaxKeys(200); //最多返回200个

ObjectListing objectListing = s3.listObjects(listObjectsRe
```

列出桶中所包含的所有文件

```
String marker = "";

ListObjectsRequest listObjectsRequest = new ListObjectsRequest();
listObjectsRequest.setBucketName("test-bucket");
listObjectsRequest.setMarker(marker);

while (true)
{
    ObjectListing objectListing = s3.listObjects(listObjectsRequest);

    List<S3ObjectSummary> contents = new ArrayList<S3ObjectSummary>();
    contents = objectListing.getObjectSummaries();

    if (contents.size() == 0)
    {
        break;
    }

    for (S3ObjectSummary content: contents)
    {

        String key = content.getKey();
        long size = content.getSize();
        System.out.format("key: %s, size: %d\n", key, size);
        listObjectsRequest.setMarker(key);
    }
}
```

删除桶

```
s3.deleteBucket("test-bucket");
```

获取桶的ACL

```
AccessControlList accessControlList = s3.getBucketAcl("test-bucket");
```

设置桶的ACL

使用预定义的ACL

```
s3.setBucketAcl("test-bucket", CannedAccessControlList.AuthenticatedRead);
```

使用自定义的ACL

```
AccessControlList accessControlList = new AccessControlList();
accessControlList.setOwner(new Owner("your_user_id", ""));
accessControlList.grantPermission(GroupGrantee.AuthenticatedRead);
accessControlList.grantPermission(GroupGrantee.LogDelivery);
accessControlList.grantPermission(new CanonicalGrantee("some-user-id"));
accessControlList.grantPermission(new EmailAddressGrantee("some-user-id"));

s3.setBucketAcl("test-bucket", accessControlList);
```

服务操作接口

列出所有的桶

```
List<Bucket> allBuckets = new ArrayList<Bucket>();
allBuckets = s3.listBuckets();
```

AWS 官方 SDK [aws-sdk-java](#) 接口详细文档 [api-reference](#)

Go Demo

安装AWS Go SDK

```
go get -u github.com/aws/aws-sdk-go/...
```

初始化，设置帐号信息和域名

```
package main

import "fmt"
import "time"
import "bytes"
import "os"
import "github.com/aws/aws-sdk-go/aws"
import "github.com/aws/aws-sdk-go/service/s3"
import "github.com/aws/aws-sdk-go/service/s3/s3manager"
import "github.com/aws/aws-sdk-go/aws/session"
import "github.com/aws/aws-sdk-go/aws/credentials"

var access_key = "ziw5dp1alvty9n47qksu" //请替换为您自己的access key
var secret_key = "V+TZ5u5wNvXb+KP5g0dMNzhMeWe372/yRKx4hZV"
var token = ""
var end_point = "http://ss.bscstorage.com"

func main() {
    credential := credentials.NewStaticCredentials(access_|
}

config := aws.NewConfig().WithRegion("us-east-1").
    WithEndpoint(end_point).
    WithCredentials(credential).WithS3ForcePathStyle(true)

sess := session.New(config)
svc := s3.New(sess)
uploader := s3manager.NewUploader(sess)
downloader := s3manager.NewDownloader(sess)
}
```

文件操作接口

上传文件

ACL可设置为：'private' 或 'public-read' 或 'public-read-write' 或
'authenticated-read'

```
params := &s3.PutObjectInput{
    Bucket: aws.String("test-bucket"),
    Key: aws.String("test-key"),
    ACL: aws.String("public-read"),
    ContentType: aws.String("image/jpeg"), //请替换为合适的文件类型
    Body: bytes.NewReader([]byte("bla bla")),
    Metadata: map[string]*string{
        "key-foo": aws.String("value-bar"),
    },
}

resp, err := svc.PutObject(params)
if err != nil {
    fmt.Println(err.Error())
    return
}
fmt.Println(resp)
```

使用uploader上传本地文件

```
f, err := os.Open("/root/test.txt")
if err != nil {
    fmt.Println("open file error")
    return
}

params := &s3manager.UploadInput{
    Bucket: aws.String("test-bucket"),
    Key: aws.String("test-key"),
    Body: f,
}

result, err := uploader.Upload(params)
if err != nil {
    fmt.Println("upload file error")
    return
}
fmt.Printf("file uploaded to: %s\n", result.Location)
```

下载文件

```

params := &s3.GetObjectInput{
    Bucket: aws.String("test-bucket"),
    Key: aws.String("test-key"),
}

resp, err := svc.GetObject(params)
if err != nil {
    fmt.Println(err.Error())
    return
}
buf := new(bytes.Buffer)
buf.ReadFrom(resp.Body)
fmt.Println(buf.String())

```

使用downloader下载到本地文件

```

f, err := os.Create("/root/test.txt.download")
if err != nil {
    fmt.Println("create file error")
    return
}

params := &s3.GetObjectInput{
    Bucket: aws.String("test-bucket"),
    Key: aws.String("test-key"),
}

n, err := downloader.Download(f, params)
if err != nil {
    fmt.Println("download file error")
    return
}
fmt.Printf("file download %d bytes\n", n)

```

获取文件的URL

获取已签名的URL用来下载文件，可通过参数设置签名过期时间

```

params := &s3.GetObjectInput{
    Bucket: aws.String("test-bucket"),
    Key: aws.String("test-key"),
}

req, _ := svc.GetObjectRequest(params)
url, _ := req.Presign(300 * time.Second) //300秒后过期
fmt.Println(url)

```

删除文件

```

params := &s3.DeleteObjectInput{
    Bucket: aws.String("test-bucket"),
    Key: aws.String("test-key"),
}
resp, err := svc.DeleteObject(params)
if err != nil {
    fmt.Println(err.Error())
    return
}
fmt.Println(resp)

```

获取文件的ACL

```

params := &s3.GetObjectAclInput{
    Bucket: aws.String("test-bucket"),
    Key: aws.String("test-key"),
}

resp, err := svc.GetObjectAcl(params)
if err != nil {
    fmt.Println(err.Error())
    return
}
fmt.Println(resp)

```

设置文件的ACL

使用预定义的ACL

支持的预定义ACL有: 'private', 'public-read', 'public-read-write' 和 'authenticated-read'

```

params := &s3.PutObjectAclInput{
    Bucket: aws.String("test-bucket"),
    Key: aws.String("test-key"),
    ACL: aws.String("private"),
}

resp, err := svc.PutObjectAcl(params)
if err != nil {
    fmt.Println(err.Error())
    return
}
fmt.Println(resp)

```

使用Grant头设置ACL

```
params := &s3.PutObjectAclInput{
    Bucket: aws.String("test-bucket"),
    Key: aws.String("test-key"),

    GrantFullControl: aws.String("id=your_user_id"),
    GrantRead: aws.String("uri=\"http://acs.amazonaws.com/c",
    GrantReadACP: aws.String("id=some_user_id, id=another_u",
    GrantWrite: aws.String("uri=\"http://acs.amazonaws.com",
    GrantWriteACP: aws.String("emailAddress=\"some_email@so
}

resp, err := svc.PutObjectAcl(params)
if err != nil {
    fmt.Println(err.Error())
    return
}
fmt.Println(resp)
```

使用自定义的ACL

可指定的Permission包括: 'FULL_CONTROL', 'WRITE', 'WRITE_ACP',
'READ', 'READ_ACP'

```

params := &s3.PutObjectAclInput{
    Bucket: aws.String("test-bucket"),
    Key: aws.String("test-key"),

    AccessControlPolicy: &s3.AccessControlPolicy{
        Grants: []*s3.Grant{
            {
                Grantee: &s3.Grantee{
                    Type: aws.String("CanonicalUser"),
                    DisplayName: aws.String(""),
                    ID: aws.String("some_user_id"),
                },
                Permission: aws.String("FULL_CONTROL"),
            },
            {
                Grantee: &s3.Grantee{
                    Type: aws.String("Group"),
                    URI: aws.String("http://acs.amazonaws.com/groups/global/AllUsers"),
                },
                Permission: aws.String("READ"),
            },
            {
                Grantee: &s3.Grantee{
                    Type: aws.String("AmazonCustomerByEmail"),
                    DisplayName: aws.String(""),
                    EmailAddress: aws.String("some_email@sample.com"),
                },
                Permission: aws.String("READ"),
            },
        },
        Owner: &s3.Owner{
            DisplayName: aws.String(""),
            ID: aws.String("your_user_id"),
        },
    },
}

resp, err := svc.PutObjectAcl(params)
if err != nil {
    fmt.Println(err.Error())
    return
}
fmt.Println(resp)

```

桶操作接口

创建桶

ACL可设置为：'private' 或 'public-read' 或 'public-read-write' 或
'authenticated-read'

```
params := &s3.CreateBucketInput{
    Bucket: aws.String("test-bucket"),
    ACL: aws.String("public-read"),
}

resp, err := svc.CreateBucket(params)
if err != nil {
    fmt.Println(err.Error())
    return
}
fmt.Println(resp)
```

列出桶中所包含的文件, 每次最多可以返回1000个文件

```
params := &s3.ListObjectsInput{
    Bucket: aws.String("test-bucket"),
    Marker: aws.String("foo"), //设置从哪个key开始列
    Prefix: aws.String("foo"), //只返回以“foo”为前缀的key
    Delimiter: aws.String("/"), //对含有公共部分的keys进行合并
    MaxKeys: aws.Int64(200), //最多返回200个
}

resp, err := svc.ListObjects(params)
if err != nil {
    fmt.Println(err.Error())
    return
}
fmt.Println(resp)
```

列出桶中所用的文件

```

marker := ""

for {
    params := &s3.ListObjectsInput{
        Bucket: aws.String("test-bucket"),
        Marker: aws.String(marker),
    }

    resp, err := svc.ListObjects(params)
    if err != nil {
        fmt.Println(err.Error())
        return
    }

    if len(resp.Contents) == 0 {
        break;
    }

    for _, content := range resp.Contents {
        fmt.Printf("key:%s, size:%d\n", *content.Key, *content.Size)
        marker = *content.Key
    }
}

```

删除桶

```

params := &s3.DeleteBucketInput{
    Bucket: aws.String("test-bucket"),
}
resp, err := svc.DeleteBucket(params)
if err != nil {
    fmt.Println(err.Error())
    return
}
fmt.Println(resp)

```

获取桶的ACL

```
params := &s3.GetBucketAclInput{
    Bucket: aws.String("test-bucket"),
}
resp, err := svc.GetBucketAcl(params)
if err != nil {
    fmt.Println(err.Error())
    return
}
fmt.Println(resp)
```

设置桶的ACL

使用预定义的ACL

支持的预定义ACL有: 'private', 'public-read', 'public-read-write' 和 'authenticated-read'

```
params := &s3.PutBucketAclInput{
    Bucket: aws.String("test-bucket"),
    ACL: aws.String("public-read-write"),
}

resp, err := svc.PutBucketAcl(params)
if err != nil {
    fmt.Println(err.Error())
    return
}
fmt.Println(resp)
```

使用Grant头设置ACL

```
params := &s3.PutBucketAclInput{
    Bucket: aws.String("test-bucket"),

    GrantFullControl: aws.String("id=your_user_id"),
    GrantRead: aws.String("uri=\"http://acs.amazonaws.com/c",
    GrantReadACP: aws.String("id=some_user_id, id=another_u
    GrantWrite: aws.String("uri=\"http://acs.amazonaws.com,
    GrantWriteACP: aws.String("emailAddress=\"some_email@so
}

resp, err := svc.PutBucketAcl(params)
if err != nil {
    fmt.Println(err.Error())
    return
}
fmt.Println(resp)
```

使用自定义的ACL

可指定的Permission包括: 'FULL_CONTROL', 'WRITE', 'WRITE_ACP',
'READ', 'READ_ACP'

```

params := &s3.PutBucketAclInput{
    Bucket: aws.String("test-bucket"),

    AccessControlPolicy: &s3.AccessControlPolicy{
        Grants: []*s3.Grant{
            {
                Grantee: &s3.Grantee{
                    Type: aws.String("CanonicalUser"),
                    DisplayName: aws.String(""),
                    ID: aws.String("some_user_id"),
                },
                Permission: aws.String("FULL_CONTROL"),
            },
            {
                Grantee: &s3.Grantee{
                    Type: aws.String("Group"),
                    URI: aws.String("http://acs.amazonaws.com/groups/global/AllUsers"),
                },
                Permission: aws.String("READ"),
            },
            {
                Grantee: &s3.Grantee{
                    Type: aws.String("AmazonCustomerByEmail"),
                    DisplayName: aws.String(""),
                    EmailAddress: aws.String("some_email@sample.com"),
                },
                Permission: aws.String("READ"),
            },
        },
        Owner: &s3.Owner{
            DisplayName: aws.String(""),
            ID: aws.String("your_user_id"),
        },
    },
}

resp, err := svc.PutBucketAcl(params)
if err != nil {
    fmt.Println(err.Error())
    return
}
fmt.Println(resp)

```

服务操作接口

列出所有的桶

```
var params *s3.ListBucketsInput
resp, err := svc.ListBuckets(params)
if err != nil {
    fmt.Println(err.Error())
    return
}
fmt.Println(resp)
```

AWS 官方 SDK [aws-sdk-go](#) 接口详细文档 [api-reference](#)

Dotnet Demo

安装AWS Dotnet SDK

```
dotnet add package AWSSDK.S3
```

初始化，设置帐号信息和域名

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

namespace dnet
{
    class Program
    {
        private static IAmazonS3 client;

        public static void Main()
        {
            # 请替换为您自己的access key
            string accessKey = "accessKey";

            # 请替换为您自己的secret key
            string secretKey = "secretKey";

            AmazonS3Config config = new AmazonS3Config();
            config.ServiceURL = "http://ss.bscstorage.com";
            client = new AmazonS3Client(accessKey, secretKey);
        }
    }
}
```

文件操作接口

上传文件

```
static async Task WritingAnObjectAsync()
{
    try
    {
        // 1. Put object-specify only key name for the new
        var putRequest1 = new PutObjectRequest
        {
            BucketName = "test-bucket-xxx",
            Key = "test-key-xxx",
            ContentBody = "sample text"
        };

        PutObjectResponse response1 = await client.PutObjectAsync(
            putRequest1);

        // 2. Put the object-set ContentType and add metadata
        var putRequest2 = new PutObjectRequest
        {
            BucketName = "test-bucket-xxx",
            Key = "test-key-xxx",
            FilePath = "/root/test.txt",
            ContentType = "text/plain"
        };
        putRequest2.Metadata.Add("x-amz-meta-title", "some title");
        PutObjectResponse response2 = await client.PutObjectAsync(
            putRequest2);
    }
    catch (AmazonS3Exception e)
    {
        Console.WriteLine(
            "Error encountered ***. Message:{0}' when
            , e.Message);
    }
    catch (Exception e)
    {
        Console.WriteLine(
            "Unknown encountered on server. Message:{0}' \n
            , e.Message);
    }
}
```

下载文件

```
static async Task ReadObjectDataAsync()
{
    string responseBody = "";
    try
    {
        GetObjectRequest request = new GetObjectRequest
        {
            BucketName = bucketName,
            Key = keyName
        };
        using (GetObjectResponse response = await client.Get
        using (Stream responseStream = response.ResponseSt
        using (StreamReader reader = new StreamReader(respo
        {
            string title = response.Metadata["x-amz-meta-t
            string contentType = response.Headers["Content-
            Console.WriteLine("Object metadata, Title: {0}"
            Console.WriteLine("Content type: {0}", contentT

            responseBody = reader.ReadToEnd(); // Now you ha
        }
    }
    catch (AmazonS3Exception e)
    {
        Console.WriteLine("Error encountered ***. Message:
    }
    catch (Exception e)
    {
        Console.WriteLine("Unknown encountered on server. N
    }
}
```

删除文件

```
private static async Task DeleteObjectAsync()
{
    try
    {
        // Delete the object
        DeleteObjectRequest request = new DeleteObjectRequest
        {
            BucketName = bucketName,
            Key = keyName,
        };
        Console.WriteLine("Deleting an object");
        await client.DeleteObjectAsync(request);
    }
    catch (AmazonS3Exception e)
    {
        Console.WriteLine("Error encountered on server. Message:");
    }
    catch (Exception e)
    {
        Console.WriteLine("Unknown encountered on server. Message:");
    }
}
```

文件的ACL

```
private static async Task TestObjectACLAAsync()
{
    try
    {
        // Retrieve the ACL for the object.
        GetACLResponse aclResponse = await client.GetACLA
        {
            BucketName = bucketName,
            Key = keyName
        });

        S3AccessControlList acl = aclResponse.AccessContro

        // Retrieve the owner (we use this to re-add permis
        Owner owner = acl.Owner;

        // Clear existing grants.
        acl.Grants.Clear();

        // Add a grant to reset the owner's full permission
        S3Grant fullControlGrant = new S3Grant
        {
            Grantee = new S3Grantee { CanonicalUser = owner
            Permission = S3Permission.FULL_CONTROL
        };

        // Describe the grant for the permission using an e
        S3Grant grantUsingEmail = new S3Grant
        {
            Grantee = new S3Grantee { EmailAddress = email,
            Permission = S3Permission.WRITE_ACP
        };
        acl.Grants.AddRange(new List<S3Grant> { fullContro

        // Set a new ACL.
        PutACLResponse response = await client.PutACLA
        {
            BucketName = bucketName,
            Key = keyName,
            AccessControlList = acl
        });
    }
    catch (AmazonS3Exception amazonS3Exception)
    {
        Console.WriteLine("An AmazonS3Exception was thrown:
    }
    catch (Exception e)
    {
```

```
        Console.WriteLine("Exception: " + e.ToString());
    }
}
```

桶操作接口

创建桶

```
static async Task CreateBucketAsync()
{
    try
    {
        PutBucketResponse response = await client.PutBucket(
    }
    catch (AmazonS3Exception e)
    {
        Console.WriteLine("Error encountered on server. Message: " +
    }
    catch (Exception e)
    {
        Console.WriteLine("Unknown encountered on server. Message: " +
    }
}
```

列出桶中所包含的文件

```
static async Task ListObjectsAsync()
{
    try
    {
        // List all objects
        ListObjectsRequest listRequest = new ListObjectsRequest
        {
            BucketName = "test-bucket-xxx",
            MaxKeys = 10,
        };

        ListObjectsResponse listResponse;
        do
        {
            // Get a list of objects
            listResponse = client.ListObjectsAsync(listRequest);
            foreach (S3Object obj in listResponse.S3Objects)
            {
                Console.WriteLine("key = {0} size = {1}", obj.Key, obj.Size);
            }

            // Set the marker property
            listRequest.Marker = listResponse.NextMarker;
        } while (listResponse.IsTruncated);
    }
    catch (AmazonS3Exception e)
    {
        Console.WriteLine("Error encountered on server. Message: {0}", e.Message);
    }
    catch (Exception e)
    {
        Console.WriteLine("Unknown encountered on server. Message: {0}", e.Message);
    }
}
```

删除桶

```

static async Task DeleteBucketAsync()
{
    try
    {
        var response = await client.DeleteBucketAsync("test-bucket-xxx");
    }
    catch (AmazonS3Exception e)
    {
        Console.WriteLine("Error encountered on server. Message: {0}", e.Message);
    }
    catch (Exception e)
    {
        Console.WriteLine("Unknown encountered on server. Message: {0}", e.Message);
    }
}

```

获取桶的ACL

```

static async Task GetBucketACLAync(string bucketName)
{
    try
    {
        GetACLResponse response = await client.GetACLAync(bucketName);
        {
            BucketName = "test-bucket-xxx";
        });
        S3AccessControlList accessControlList = response.AccessControlList;
        Console.WriteLine("ID: {0}", accessControlList.Owner.Id);
        foreach(var g in accessControlList.Grants)
        {
            Console.WriteLine("Permission: {0}, {1}", g.Permission, g.Grantee);
            Console.WriteLine("Grantee: {0}", g.Grantee.CallerIdentity);
        }
    }
    catch (AmazonS3Exception e)
    {
        Console.WriteLine("Error encountered on server. Message: {0}", e.Message);
    }
    catch (Exception e)
    {
        Console.WriteLine("Unknown encountered on server. Message: {0}", e.Message);
    }
}

```

设置桶的ACL

```
// Set Canned ACL (PublicRead)
client.PutACL(new PutACLRequest
{
    BucketName = "SampleBucket",
    CannedACL = S3CannedACL.PublicRead
});

// Set Canned ACL (PublicRead)
client.PutACL(new PutACLRequest
{
    BucketName = "SampleBucket",
    CannedACL = S3CannedACL.Private
});
```

列出所有的桶

```
static async Task ListBucketsAsync(string bucketName)
{
    var response = client.ListBucketsAsync().Result;
    foreach (var bucket in response.Buckets)
    {
        Console.WriteLine(bucket.BucketName);
    }
}
```

AWS 官方 SDK [aws-sdk-dotnet](#) 接口详细文档 [api-reference](#)

约束与限制

- 编码限制

- bucket名称、object key、meta、ACL内容等只支持UTF-8编码=
- 访问资源的url要进行rawurlencode编码
- object key除了“/”以外需要进行rawurlencode编码

- 命名规则

API中的bucket和object的命名需符合以下规则：

bucket命名：

- 云存储内全局唯一；
- 由小写字母、数字及“-”组成，长度为6~63位；
- 不能以数字、‘-’开头；
- 不能以‘-’开头或结尾；
- 或者使用域名的名命名规则，例如：xxx.foo.com.cn，方便绑定您的域名

object命名：

- key长度不超过512个字节；
- key除了“/”以外需要进行rawurlencode编码

签名算法

- 如果http请求中没有携带身份信息（AccessKey），则改该请求为匿名请求，会被认为是来自匿名用户的访问。
- 如果http请求中携带了身份信息（AccessKey），则认为访问来自该AccessKey所对应的用户。由于AccessKey是可以被他人获取到的，为了防止其他人冒用您的AccessKey来访问服务，请求中还必须携带您的签名。在申请帐号以后，您将得到AccessKey和SecretKey，SecretKey是需要保密的。签名是由此次http请求的相关信息和您的SecretKey计算得到的，其他人因为不知道您的SecretKey，将不能计算出正确的签名。
- 身份信息与签名可以放到请求头（Authorization）中，也可以放到请求参数中。
- 签名的方式与Amazon S3的签名方式兼容，支持signature version 2 和 signature version 4。

添加签名

由于计算签名的过程比较繁琐且容易出错，不建议自己计算签名，推荐使用[SDKs](#)，SDK可以自动为请求计算签名。

添加version 2 签名

身份信息与签名的携带方式：

通过Authorization请求头：

请求头格式：

```
Authorization: AWS AWSAccessKeyId:Signature
```

- AWSAccessKeyId: 您的AccessKey
- Signature: 计算得到的签名

例子：

```
Authorization: AWS ziw5dp1alvty9n47qksu:frJIUN8DYpKDt0LCwo,
```

通过请求参数：

需要在请求中包含以下三个参数：

- AWSAccessKeyId: 指定您的AccessKey
- Signature: 计算得到的签名
- Expires: 指定签名的过期时间

例子:

```
GET /yourbucket/yourkey?AWSAccessKeyId=ziw5dp1alvty9n47qks1
```

签名的计算方法:

```
Signature = Base64( HMAC-SHA1( YourSecretKey, UTF-8-Encoder(StringToSign)) )  
  
StringToSign = HTTP-Verb + "\n" +  
Content-MD5 + "\n" +  
Content-Type + "\n" +  
Date|Expires + "\n" +  
CanonicalizedAmzHeaders +  
CanonicalizedResource
```

- YourSecretKey: 您的SecretKey
- HTTP-Verb: 请求的方法, 如: PUT , GET , DELETE , POST
- Content-MD5: 请求头Content-MD5的内容, 如果没有这个头, 由空字符串代替
- Content-Type: 请求头Content-Type的内容, 如果没有这个头, 由空字符串代替
- Date|Expires: 如果使用Authorization头携带签名信息, 为Date头的内容, 如果没有Date头, 由空字符串代替; 如果使用请求参数携带签名信息, 为参数Expires的内容
- CanonicalizedAmzHeaders: 请求中所有以x-amz-开始的头所组成的字符串,如果没有这样的头, 由空字符串代替
- CanonicalizedResource: 请求所对应的资源

计算CanonicalizedAmzHeaders的例子:

原始请求头:

```
Date: Tue, 27 Mar 2007 19:36:42 +0000
X-Amz-b: Bar
x-amz-a: foob
x-Amz-a: fooa
Host: johnsmith.s3.amazonaws.com
```

对应的CanonicalizedAmzHeaders为:

```
x-amz-a:fooa,foob
x-amz-b:Bar
```

注意: 1. 请求头的名字全部转换为小写, 按转换后的头名字排序。
2, 如果出现重复的头, 需要合并, 各个值之间用逗号分隔, 并排序。3, 去掉值前后的空格。

计算CanonicalizedResource的例子:

```
GET /?foo=bar
GET /yourbucket/yourkey?foo=bar
GET /yourbucket/yourkey?acl&foo=bar
```

对应的nicalizedResource分别为:

```
/
/yourbucket/yourkey
/yourbucket/yourkey?acl
```

如果您需要了解完整和详细的签名计算过程, 请参考以下连接:

- <http://docs.aws.amazon.com/AmazonS3/latest/dev/RESTAuthentication.html>
- <http://docs.aws.amazon.com/general/latest/gr/signature-version-2.html>

添加version 4 签名:

身份信息与签名的携带方式:

通过Authorization头:

例子:

```
Authorization: AWS4-HMAC-SHA256 Credential=ziw5dp1alvty9n4:
```

- Credential由AccessKey, 请求的日期, region, 服务名, aws4_request 五部分组成, 各部分之间用斜线分隔

- SignedHeaders: 表示那些头参与了签名的计算, 未包含在这里的头不会影响到签名的生成
- Signature: 计算得到的签名

通过请求参数:

需要在请求中加入以下参数:

- X-Amz-Algorithm: 计算签名时使用的Hash算法, 指定为 AWS4-HMAC-SHA256
- X-Amz-Credential: 包含了AccessKey, 日期, region, 服务名的信息
- X-Amz-Date: 请求的时间
- X-Amz-Expires: 指定签名在多长时间内有效
- X-Amz-SignedHeaders: 计算签名时用到的头
- X-Amz-Signature: 计算得的到签名

例子:

```
GET /yourbucket/test.mp4?X-Amz-Algorithm=AWS4-HMAC-SHA256&
```

签名的计算方法:

计算CanonicalRequest

```
CanonicalRequest =
    HTTPRequestMethod + '\n' +
    CanonicalURI + '\n' +
    CanonicalQueryString + '\n' +
    CanonicalHeaders + '\n' +
    SignedHeaders + '\n' +
    HexEncode(Hash(RequestPayload))
```

- HTTPRequestMethod: 如 PUT, GET, DELETE, POST
- CanonicalURI: 请求的uri
- CanonicalQueryString: 请求参数排序后组成的字符串
- CanonicalHeaders: 需要加入签名计算的头排序后组成的字符串
- SignedHeaders: 加入到签名计算的头的名字的列表, 各个名字之间用逗号分隔

- HexEncode(Hash(RequestPayload)): 请求body的hash的16进制编码,如果通过请求参数携带签名, 此处应由字符串 UNSIGNED-PAYLOAD 代替

计算StringToSign

```
StringToSign =  
    Algorithm + '\n' +  
    RequestDate + '\n' +  
    CredentialScope + '\n' +  
    HashedCanonicalRequest
```

- Algorithm: 指定为 AWS4-HMAC-SHA256
- RequestDate: ISO8601 basic 格式的请求时间, 如:
20160830T123600Z
- CredentialScope: 日期, region, 服务名等组成的字符串, 如:
20160830/us-east-1/s3/aws4_request
- HashedCanonicalRequest: Hex(SHA256Hash(CanonicalRequest)), 即CanonicalRequest的hash的16进制编码

计算signing key

```
kSecret = YourSecretKey  
kDate = HMAC("AWS4" + kSecret, Date)  
kRegion = HMAC(kDate, Region)  
kService = HMAC(kRegion, Service)  
kSigning = HMAC(kService, "aws4_request")
```

- YourSecretKey: 您的SecretKey
- Date: 8位数的日期, 应与Credential中的Date部分一样
- Region: 应与Credential中的region部分一样
- Service: 应与Credential中的服务名部分一样
- kSigning即为用于计算签名的signing key

计算签名

```
signature = HexEncode(HMAC-SHA256(kSigning, StringToSign))
```

如果您需要了解完整和详细的签名计算过程, 请参考以下连接

- <http://docs.aws.amazon.com/AmazonS3/latest/API/sig-v4-authenticating-requests.html>

Python

- <http://docs.aws.amazon.com/general/latest/gr/signature-version-4.html>

ACL(Access Control List)

访问控制列表(ACL)使您可以管理存储桶(bucket)和对象(object)的访问权限。每个存储桶和对象都都有一个附加的ACL子资源。它定义了哪些用户或则组将被授予访问权限, 收到某个资源(存储桶或或者对象)的请求后, S2将检查相应的ACL以验证请求者是否拥有所需的访问权限。

创建存储桶或对象时, S2将创建一个默认ACL, 以授予资源拥有者对资源的完全控制权限, 如下面的示例存储桶ACL中所示。

```
<?xml version="1.0" encoding="UTF-8"?>
<AccessControlPolicy xmlns="http://s3.amazonaws.com/doc/2006-03-01">
    <Owner>
        <ID>*** Owner-Canonical-User-ID ***</ID>
        <DisplayName>owner-display-name</DisplayName>
    </Owner>
    <AccessControlList>
        <Grant>
            <Grantee xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="String">
                <ID>*** Owner-Canonical-User-ID ***</ID>
                <DisplayName>display-name</DisplayName>
            </Grantee>
            <Permission>FULL_CONTROL</Permission>
        </Grant>
    </AccessControlList>
</AccessControlPolicy>
```

示例ACL包含一个Owner元素表示资源拥有者的用户名。Grant元素表示被授权者（用户或预定义的组）和所授予的权限。您可以通过添加Grant元素授予权限。

被授权者(Grantee)

被授权者可以是S2用户或某个预定义的S2组。您可以通过email地址或用户名将权限授予某个用户。但是, 如果您在授权请求设置的是email地址, S2将查找该email对应的用户名并将它添加到ACL。生成的ACL将始终包含的是S2用户的用户名, 而不是用户的email地址。

S2预定义的组(Group)

S2拥有一系列预定义的组。将用户访问权限授予某个组时, 您可以指定一个URI, 而不是用户名。我们提供以下预定义的组:

- 经身份验证的用户组(Authenticated Users group), 由 <http://acs.amazonaws.com/groups/global/AuthenticatedUsers> 标识, 该组代表了所有的S2用户。该组的访问权限允许任何S2用户访问资源, 但是, 所有的请求必须是已签名的 (经身份验证)。
- 所有的用户组(All Users group), 由 <http://acs.amazonaws.com/groups/global/AllUsers> 标识。该组的访问权限允许任何人访问资源。请求可以是已签名的 (经身份验证), 也可以是未签名的 (匿名)。
- 日志传输组(Log Delivery group), 由 <http://acs.amazonaws.com/groups/s3/LogDelivery> 标识, 存储桶上的 WRITE 许可使该组可以将服务器访问日志写入存储桶。

可以指定被授权者的权限(Permissions)

下表列出了S2在ACL中支持的权限集及在操作资源语境中的含义。

Permission	当在存储桶上授权	当在对象上授权
READ	允许授权者列出 bucket 中所有的对象	允许被授权者读取对象的数据和meta信息
WRITE	允许被授权者创建、覆盖和删除存储桶中的对象	不可用
READ_ACP	允许被授权者读取存储桶ACL	允许被授权者读取对象的ACL
WRITE_ACP	允许被授权者修改存储桶的ACL	允许被授权者修改对象的ACL
FULL_CONTROL	允许被授权者在存储桶上的READ、WRITE、READ_ACP 和WRITE_ACP权限	允许被授权者在对象上的READ、READ_ACP 和WRITE_ACP权限

Canned ACL

S2支持一系列预定义的授权, 称为标准ACL。每个标准ACL都有一组预定义的被授权者和权限。下表列出了一系列标准ACL和相关联的预定义授权。

Canned ACL	适用于	对应的ACL权限
private	存储桶和对象	拥有者将获得FULL_CONTROL。其他人没有访问权限（默认）。
public-read	存储桶和对象	拥有者将获得FULL_CONTROL。匿名用户有READ的权限
public-read-write	存储桶和对象	拥有者将获得FULL_CONTROL。 AllUsers组用户有READ, WRITE的权限， 对象没有WRITE权限
authenticated-read	存储桶和对象	拥有者将获得FULL_CONTROL。 AuthenticatedUsers组用户有READ

如何指定ACL

S2 API使您可以在创建存储桶或对象时设置ACL。S2还提供API在现有存储桶或对象上设置ACL。这些API向您提供了以下方法来设置ACL：

- 使用请求header设置ACL – 发送创建资源（存储桶或对象）的请求时，您可以使用x-amz-acl头设置Canned acl，或者使用x-amz-grant-*头设置某个用户或者则组具有的权限。
- 使用ACL API设置ACL – 当您发送在现有资源上设置ACL的请求时，您可以在请求标头或body中设置ACL，如put_object_acl API

注意：

- x-amz-acl和x-amz-grant-*不能同时指定，同时指定是返回400错误
- 在put_object_acl API中如果，指定 x-amz-acl或者x-amz-grant-*请求头，则忽略body中的ACL

x-amz-grant-*请求头类型

x-amz-grant-*请求头的值为type=value对的形式, 每个type=value以逗号隔开, 同一请求中可以指定多个x-amz-grant-*请求头, 允许的type为:

- emailAddress, 用户的email地址
- id, 用户名
- uri, 预定义用户组的uri

Name	Description
x-amz-grant-read	允许授权者READ的权限
x-amz-grant-write	允许授权者WRITE的权限
x-amz-grant-read-acp	允许授权者READ_ACP的权限
x-amz-grant-write-acp	允许授权者WRITE_ACP的权限
x-amz-grant-full-control	允许授权者READ, WRITE, READ_ACP, WRITE_ACP的权限

例子:

```
x-amz-grant-read: emailAddress="xyz@amazon.com", emailA
```

Service操作

GET Service (List Buckets)

- 描述：获得当前Owner下所有Bucket的列表。
- 请求格式：

```
GET /?formatter=json HTTP/1.1
Host: ss.bscstorage.com
Date: <date>
Authorization: <authorization string> #请参照《签名算法》
```

或者

```
GET /<Your-Bucket-Name>/?formatter=json HTTP/1.1
Host: ss.bscstorage.com
Date: <date>
Authorization: <authorization string> #请参照《签名算法》
```

- 响应格式 (HTTP Body) :

```
{
    "Owner": {
        "DisplayName": "",
        "ID": "Baishan0000001234567890"
    },
    "Buckets": {
        "Bucket": [
            {
                "CreationDate": "Fri, 21 Mar 2014 01:13:42",
                "Name": "bucket_name_0"
            },
            {
                "CreationDate": "Fri, 12 Mar 2013 02:25:22",
                "Name": "bucket_name_1"
            },
            ...
        ]
    }
}
```

- 返回值说明：

Name	Description
Owner	所有者
DisplayName	所有者的显示名字
ID	所有者的UserId
Buckets	多个Buckets的容器
Bucket	Bucket信息的容器
CreationDate	当前Bucket创建时间
Name	Bucket名称

- 请求示例：

```
curl -v -H "Date: Sat, 20 Nov 2286 17:46:39 GMT" -H "Authorization: Bearer <access_key>" -X GET "http://ss.bscstorage.com/?KID=baishan<access_key>"
```

或者

```
curl -v "http://ss.bscstorage.com/?KID=baishan,<access_key>"
```

- 响应示例：

```
HTTP/1.1 200 OK
Server: openresty/1.9.7.4
Date: Mon, 08 Aug 2016 04:04:52 GMT
Content-Type: application/json
Connection: keep-alive
Content-Length: 155
x-amz-s2-requester: your user id
x-amz-request-id: 000011e5-1608-0812-0452-00163e0069ec

{
    "Owner": {
        "DisplayName": "",
        "ID": "Baishan0000001234567890"
    },
    "Buckets": [
        "Bucket": [
            {
                "CreationDate": "Mon, 08 Aug 2016 03:15:40",
                "Name": "bucket_name_0"
            },
            {
                "CreationDate": "Mon, 08 Aug 2016 03:15:40",
                "Name": "bucket_name_1"
            },
            ...
        ]
    }
}
```

Bucket操作

GET Bucket (List Objects)

- 描述：获取bucket下所有object。
- 请求格式：

```
GET /?formatter=json HTTP/1.1
Host: <Your-Bucket-Name>.ss.bscstorage.com
Date: <date>
Authorization: <authorization string> #请参照《签名算法》
```

或者

```
GET /<Your-Bucket-Name>/?formatter=json HTTP/1.1
Host: ss.bscstorage.com
Date: <date>
Authorization: <authorization string> #请参照《签名算法》
```

- 请求参数：

Parameter	Description	Required
delimiter	折叠显示字符。通常使用：‘/’	No
marker	Key的初始位置，系统将列出比Key大的值，通常用作‘分页’的场景	No
max-keys	返回值的最大Key的数量。默认为400	No
prefix	列出以指定字符为开头的Key	No

响应格式举例（HTTP Body）：

```
{  
  
    Delimiter: "/",  
  
    Prefix: "html/",  
  
    CommonPrefixes: [  
  
        {  
            Prefix: "html/assets/"  
        },  
  
        {  
            Prefix: "html/attribution/"  
        },  
  
        {  
            Prefix: "html/documentation/"  
        },  
  
        ...  
    ],  
  
    Marker: null,  
  
    ContentsQuantity: 5,  
  
    CommonPrefixesQuantity: 3,  
  
    NextMarker: null,  
  
    IsTruncated: false,  
  
    Contents: [  
  
        {  
            SHA1: "9fc710aa89efbe42020b0310d16a07449bf0613",  
            Name: "html/coming-soon.html",  
            Expiration-Time: null,  
            Last-Modified: "Fri, 21 Mar 2014 01:50:46 UTC",  
            Owner: "Baishan0000000000000001",  
            MD5: "934d922cac80449ee361cefefeb3276b3e",  
            Content-Type: "text/html",  
            Size: 8781  
        },  
  
        {  
            SHA1: "a9625a128263f05e331f6d78add9bd15911c356",  
            Name: "html/coming-soon.html",  
            Expiration-Time: null,  
            Last-Modified: "Fri, 21 Mar 2014 01:50:46 UTC",  
            Owner: "Baishan0000000000000001",  
            MD5: "934d922cac80449ee361cefefeb3276b3e",  
            Content-Type: "text/html",  
            Size: 8781  
        }  
    ]  
}
```

```

        Name: "html/ebook.html",
        Expiration-Time: null,
        Last-Modified: "Fri, 21 Mar 2014 01:50:47 UTC",
        Owner: "Baishan0000000000000001",
        MD5: "cb7ed943ead4aeb513aa8c0b76865a8b",
        Content-Type: "text/html",
        Size: 18734
    },
    ...
]
}

```

- 返回值说明：

Name	Description
Contents	Object的Metadata数组
CommonPrefixes	折叠以后的Prefix，下一级是Prefix数组
Delimiter	当前使用的折叠字符
Prefix	当前使用的前缀
Marker	当前使用的Marker
ContentsQuantity	Contents中元素个数
CommonPrefixesQuantity	CommonPrefixes中元素个数
NextMarker	下一页的Marker
IsTruncated	是否还有下一页
SHA1	文件内容的sha1值
Name	Object的Key(文件名)
Last-Modified	Object的最后修改时间
Owner	Object的拥有者
MD5	文件内容的md5值
Content-Type	文件的mime type
Size	文件的大小(字节)

- 应用举例： 假设某Bucket下有如下文件(为方便说明，没有显示为json格式，仅表现其中的一些有用信息，以下同)：

```
join/mailaddresss.txt  
join/mycodelist.txt  
join/personalfiles/connects.docx  
join/personalfiles/myphoto.jpg  
join/readme.txt  
join/userlist.txt  
join/zero.txt  
mary/personalfiles/mary.jpg  
mary/readme.txt  
sai/readme.txt
```

使用prefix指定以join/为开头的文件：

```
GET /?prefix=join/&formatter=json HTTP/1.1  
Host: <Your-Bucket-Name>.ss.bscstorage.com  
Date: <date>  
Authorization: <authorization string> #请参照《签名算法》
```

返回：

```
Contents:  
join/mailaddresss.txt  
join/mycodelist.txt  
join/personalfiles/connects.docx  
join/personalfiles/myphoto.jpg  
join/readme.txt  
join/userlist.txt  
join/zero.txt
```

使用delimiter指定折叠方式为'/'：

```
GET /?delimiter=/&formatter=json HTTP/1.1  
Host: <Your-Bucket-Name>.ss.bscstorage.com  
Date: <date>  
Authorization: <authorization string> #请参照《签名算法》
```

返回：

```
Contents:  
  
CommonPrefix:  
join  
mary  
sai
```

使用prefix指定以join/为开头的文件，同时使用delimiter指定折叠方式为'/'：

```
GET /?prefix=join/&delimiter=/&formatter=json HTTP/1.1
Host: <Your-Bucket-Name>.ss.bscstorage.com
Date: <date>
Authorization: <authorization string> #请参照《签名算法》
```

返回：

```
Contents:
join/mailaddresss.txt
join/mycodelist.txt
join/readme.txt
join/userlist.txt
join/zero.txt
CommonPrefix:
join/personalfiles/
```

使用max-keys做最大值列表长度限制：

```
GET /?prefix=join/&delimiter=/&max-keys=4&formatter=json HTTP/1.1
Host: <Your-Bucket-Name>.ss.bscstorage.com

Date: <date>
Authorization: <authorization string> #请参照《签名算法》
```

返回：

```
IsTruncated : true
Next-Marker : join/userlist.txt
Contents:
join/mailaddresss.txt
join/mycodelist.txt
join/readme.txt
CommonPrefix:
join/personalfiles/
```

使用marker继续获得之前的列操作的后续结果：

```
GET /?prefix=join/&delimiter=/&max-keys=4&marker=join/userlist.txt&marker=join/userlist.txt
Host: <Your-Bucket-Name>.ss.bscstorage.com
Date: <date>
Authorization: <authorization string> #请参照《签名算法》
```

Python

返回：

```
IsTruncated : false
Contents:
    join/userlist.txt
    join/zero.txt
```

PUT Bucket

- 描述：创建一个Bucket。
- 请求格式：

```
PUT /?formatter=json HTTP/1.1
Host: <Your-Bucket-Name>.ss.bscstorage.com
Date: <date>
Authorization: <authorization string> #请参照《签名算法》
x-amz-acl: <Canned-ACL> #请参照[《ACL》](../acl/acl.md)
```

或者

```
PUT /<Your-Bucket-Name>/?formatter=json HTTP/1.1
Host: ss.bscstorage.com
Date: <date>
Authorization: <authorization string> #请参照《签名算法》
x-amz-acl: <Canned-ACL> #请参照[《ACL》](../acl/acl.md)
```

- Request Header (请求头) :

Name	Description	Required
x-amz-acl	创建Bucket的同时，设置一个ACL。请参照《ACL》	No

- 响应 (无HTTP Body) :

```
HTTP/1.1 200 OK
Date: Tue, 08 Apr 2014 02:59:47 GMT
Content-Length: 0
Connection: keep-alive
X-RequestId: 00078d50-1404-0810-5947-782bcb10b128
X-Requester: Your UserId
```

- 请求示例：

```
curl -v -X PUT -H "Date: Sat, 20 Nov 2286 17:46:39 GMT" -H
```

或者

```
curl -v -X PUT "http://<Your-Bucket-Name>.ss.bscstorage.cor
```

Python

或者

```
curl -v -X PUT "http://ss.bscstorage.com/<Your-Bucket-Name>
```

DELETE Bucket

- 描述：删除指定Bucket。
- 注意：不能删除非空Bucket。
- 请求格式：

```
DELETE /?formatter=json HTTP/1.1
Host: <Your-Bucket-Name>.ss.bscstorage.com
Date: <date>
Authorization: <authorization string> #请参照《签名算法》
```

或者

```
DELETE /<Your-Bucket-Name>/?formatter=json HTTP/1.1
Host: ss.bscstorage.com
Date: <date>
Authorization: <authorization string> #请参照《签名算法》
```

- 响应（无HTTP Body）：

```
HTTP/1.1 204 No Content
Date: Tue, 08 Apr 2014 02:59:47 GMT
Content-Length: 0
Connection: keep-alive
X-RequestId: 00078d50-1404-0810-5947-782bcb10b128
X-Requester: Your UserId
```

- 请求示例：

```
curl -v -X DELETE -H "Date: Sat, 20 Nov 2286 17:46:39 GMT"
```

或者

```
curl -v -X DELETE "http://<Your-Bucket-Name>.ss.bscstorage.
```

或者

```
curl -v -X DELETE "http://ss.bscstorage.com/<Your-Bucket-Name>".
```

PUT Bucket ACL

- 描述：给指定Bucket设置ACL规则。更多信息请参照：[《ACL》](#)
- 请求格式：

```
PUT /?acl&formatter=json HTTP/1.1
Host: <Your-Bucket-Name>.ss.bscstorage.com
Date: <date>
Authorization: <authorization string> #请参照《签名算法》

#ACL规则
{
    'Baishan0000000000000001' : [ "read", "read_acp" , "wri
    'GRPS00000ANONYMOUSE' : [ "read", "read_acp" , "write"
    'GRPS000000CANONICAL' : [ "read", "read_acp" , "write"
}
```

- 响应：

```
HTTP/1.1 200 OK
Date: Tue, 08 Apr 2014 02:59:47 GMT
Connection: keep-alive
X-RequestId: 00078d50-1404-0810-5947-782bcb10b128
X-Requester: Your UserId

{
    "Owner": {
        "DisplayName": "",
        "ID": "user_authed"
    },
    "AccessControlList": [
        "Grant": [
            {
                "Grantee": {
                    "DisplayName": "",
                    "ID": "Baishan0000000000000001"
                },
                "Permission": "READ,READ_ACP,WRITE,WRITE_ACP"
            },
            ...
        ]
    }
}
```

- 请求格式说明请参照: [《ACL》](#)
- 请求示例:

```
curl -v -T "acl.txt" -H "Date: Sat, 20 Nov 2286 17:46:39 GMT"
```

GET Bucket ACL

- 描述：获得指定Bucket的ACL信息。更多信息请参照：[《ACL》](#))
- 请求格式：

```
GET /?acl&formatter=json HTTP/1.1
Host: <Your-Bucket-Name>.ss.bscstorage.com
Date: <date>
Authorization: <authorization string> #请参照《签名算法》
```

或者

```
GET /<Your-Bucket-Name>/?acl&formatter=json HTTP/1.1
Host: ss.bscstorage.com
Date: <date>
Authorization: <authorization string> #请参照《签名算法》
```

响应：

```

HTTP/1.1 200 OK
Date: Tue, 08 Apr 2014 02:59:47 GMT
Content-Length: 123
Connection: keep-alive
X-RequestId: 00078d50-1404-0810-5947-782bcb10b128
X-Requester: Your UserId

{
    "Owner": {
        "DisplayName": "",
        "ID": "Baishan0000000000000001"
    },
    "AccessControlList": {
        "Grant": [
            {
                "Grantee": {
                    "DisplayName": "",
                    "ID": "GRPS00000ANONYMOUSE"
                },
                "Permission": "READ"
            },
            {
                "Grantee": {
                    "DisplayName": "",
                    "ID": "Baishan000001001NHT3M7"
                },
                "Permission": "READ, READ_ACP, WRITE, WRITE_ACP"
            },
            {
                "Grantee": {
                    "DisplayName": "",
                    "ID": "Baishan0000000000000001"
                },
                "Permission": "READ, WRITE"
            },
            ...
        ]
    }
}

```

- 响应格式说明请参照：《[ACL](#)》）
- 请求示例：

```
curl -v -H "Date: Sat, 20 Nov 2286 17:46:39 GMT" -H "Authorization: Bearer Baishan0000000000000001" -X GET https://api-test.baidubce.com/v1/datalake/objects?path=1&acl
```

Python

或者

```
curl -v "http://<Your-Bucket-Name>.ss.bscstorage.com/?acl&t=
```

或者

```
curl -v "http://ss.bscstorage.com/<Your-Bucket-Name>/?acl&t=
```

Object操作

HEAD Object

- 描述：使用HEAD请求方式获取Object的Metadata。
- 请求格式：

```
HEAD /<ObjectName> HTTP/1.1
Host: <Your-Bucket-Name>.ss.bscstorage.com
Date: <date>
Authorization: <authorization string> #请参照《签名算法》
```

- 响应（无HTTP Body）：

```
HTTP/1.1 200 OK
Date: Tue, 08 Apr 2014 02:59:47 GMT
Connection: keep-alive
Content-Type: <object-mime-type>
Content-Length: <object-file-bytes>
ETag: "<文件的MD5值>"
Last-Modified: <最后修改时间>
x-amz-request-id: 0000106c-1608-0810-4621-00163e000064
x-amz-s2-requester: <Your UserName>
x-amz-meta-foo1: <value1> #自定义meta: foo1
x-amz-meta-foo2: <value2> #自定义meta: foo2
```

- Request Headers（请求头）：

Name	Description	Req
Range	<p>Downloads the specified range bytes of an object. For more information about the HTTP Range header, go to http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html#sec14.35.</p> <ul style="list-style-type: none"> • Type: String • Default: None • Constraints: None 	No
If-Modified-Since	<p>Return the object only if it has been modified since the specified time, otherwise return a 304 (not modified).</p> <ul style="list-style-type: none"> • Type: String • Default: None • Constraints: None 	No
If-Unmodified-Since	<p>Return the object only if it has not been modified since the specified time, otherwise return a 412 (precondition failed).</p> <ul style="list-style-type: none"> • Type: String • Default: None • Constraints: None 	No
If-Match	<p>Return the object only if its entity tag (ETag) is the same as the one specified; otherwise, return a 412 (precondition failed).</p> <ul style="list-style-type: none"> • Type: String • Default: None • Constraints: None 	No
If-None-Match	<p>Return the object only if its entity tag (ETag) is different from the one specified; otherwise, return a 304 (not modified).</p> <ul style="list-style-type: none"> • Type: String • Default: None • Constraints: None 	No

- Response Headers (响应头) :

Name	Description
Content-Type	Object的mime-type
Content-Length	Object的Size(bytes)
ETag	Object的hash值，一般是md5值
Last-Modified	Object的最后修改时间
x-amz-meta-*	用户可自定义文件属性信息，读取时原值返回。 例如： x-amz-meta-UploadLocation: My Home X-amz-meta-ReviewedBy: test@test.net X-amz-meta-FileChecksum: 0x02661779 X-amz-meta-CheckSumAlgorithm: crc32
x-amz-meta-crc32	Object的CRC32值

- 请求示例：

```
curl -v -X HEAD -H "Date: Sat, 20 Nov 2286 17:46:39 GMT" -t
```

GET Object

- 描述：获取一个Object（下载）。
- 请求格式：

```
GET /<ObjectName> HTTP/1.1
Host: <Your-Bucket-Name>.ss.bscstorage.com
Date: <date>
Authorization: <authorization string> #请参照《签名算法》
Range: bytes=<byte_range> #支持断点下载
```

- 响应：

```
HTTP/1.1 200 OK
Date: Tue, 08 Apr 2014 02:59:47 GMT
Connection: keep-alive
Content-Type: <object-mime-type>
Content-Length: <object-file-bytes>
ETag: "<文件的MD5值>"
Last-Modified: <最后修改时间>
x-amz-request-id: 0000106c-1608-0810-4621-00163e000064
x-amz-s2-requester: <Your UserName>
x-amz-meta-foo1: <value1> #自定义meta: foo1
x-amz-meta-foo2: <value2> #自定义meta: foo2

#文件内容
<BODY>
```

- Request Headers（请求头）：

Name	Description	Req
Range	<p>Downloads the specified range bytes of an object. For more information about the HTTP Range header, go to http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html#sec14.35.</p> <ul style="list-style-type: none"> • Type: String • Default: None • Constraints: None 	No
If-Modified-Since	<p>Return the object only if it has been modified since the specified time, otherwise return a 304 (not modified).</p> <ul style="list-style-type: none"> • Type: String • Default: None • Constraints: None 	No
If-Unmodified-Since	<p>Return the object only if it has not been modified since the specified time, otherwise return a 412 (precondition failed).</p> <ul style="list-style-type: none"> • Type: String • Default: None • Constraints: None 	No
If-Match	<p>Return the object only if its entity tag (ETag) is the same as the one specified; otherwise, return a 412 (precondition failed).</p> <ul style="list-style-type: none"> • Type: String • Default: None • Constraints: None 	No
If-None-Match	<p>Return the object only if its entity tag (ETag) is different from the one specified; otherwise, return a 304 (not modified).</p> <ul style="list-style-type: none"> • Type: String • Default: None • Constraints: None 	No

- Response Headers (响应头) :

Name	Description
Content-Type	Object的mime-type
Content-Length	Object的Size(bytes)
ETag	Object的hash值, 一般是md5值
Last-Modified	Object的最后修改时间
x-amz-meta-*	用户可自定义文件属性信息, 读取时原值返回。 例如: x-amz-meta-UploadLocation: My Home X-amz-meta-ReviewedBy: test@test.net X-amz-meta-FileChecksum: 0x02661779 X-amz-meta-CheckSumAlgorithm: crc32
x-amz-meta-crc32	Object的CRC32值

- 请求示例:

```
curl -v -H "Range: bytes=0-1024" -H "Date: Sat, 20 Nov 2286 11:53:46 GMT" -H "Authorization: AWS AccessKey:1234567890123456789012345678901234567890" -H "Content-Type: application/octet-stream" -H "Content-Length: 1025" -H "Content-MD5: 0x02661779" -H "Content-CRC32: 0x02661779" -H "ETag: 1234567890123456789012345678901234567890" -H "Last-Modified: 2023-11-20T11:53:46Z" -H "x-amz-meta-UploadLocation: My Home" -H "x-amz-meta-ReviewedBy: test@test.net" -H "x-amz-meta-FileChecksum: 0x02661779" -H "x-amz-meta-CheckSumAlgorithm: crc32" -H "User-Agent: curl/7.64.1" -H "Host: ss.bscstorage.com" -H "Accept: */*"
```

- 应用举例:

- 标准示例:

```
GET /my_bucket/path/to/my/file.txt HTTP/1.1
Host: ss.bscstorage.com
Date: Sun, 1 Jan 2006 12:00:00 GMT
Authorization: AWS AccessKey:ssig
Range: bytes=100-2048
```

- 响应:

```
HTTP/1.1 206 Partial Content
Server: openresty/1.9.7.4
Content-Type: application/xml
Connection: keep-alive
x-amz-request-id: 0000106c-1608-0810-4621-00163e000064
x-amz-s2-requester: GRPS00000ANONYMOUSE
Date: Mon, 08 Aug 2016 02:46:21 GMT
Last-Modified: Mon, 08 Aug 2016 02:45:55 GMT
ETag: "21b1a992d1cbf49729fc4461e55dd94f"
x-amz-meta-s2-size: 109051904
x-amz-meta-s2-crc32: 9422bc32
Cache-Control: max-age=31536000
Content-Length: 109051904
```

```
...
file_content
...
```

- 使用各种验证措施的下载方式：

```
GET /path/to/my/file.txt?AWSAccessKeyId=<AccessKey>&Expires=<Expire>
Host: my_bucket.ss.bscstorage.com
Date: date
Range: bytes=byte_range
```

- 响应：

```
HTTP/1.1 206 Partial Content
Server: openresty/1.9.7.4
Content-Type: application/xml
Connection: keep-alive
x-amz-request-id: 0000106c-1608-0810-4621-00163e000064
x-amz-s2-requester: GRPS00000ANONYMOUSE
Date: Mon, 08 Aug 2016 02:46:21 GMT
Last-Modified: Mon, 08 Aug 2016 02:45:55 GMT
ETag: "21b1a992d1cbf49729fc4461e55dd94f"
x-amz-meta-s2-size: 109051904
x-amz-meta-s2-crc32: 9422bc32
Cache-Control: max-age=31536000
Content-Length: 109051904
```

```
...
file_content
...
```

以上示例中*QueryString*的含义请参照[《签名算法》][1]中的关于 认证方式 的说明。

PUT Object

- 描述：PUT方式上传一个文件（同时可以设置meta和acl）。
- 请求格式：

```
PUT /<ObjectName> HTTP/1.1
Host: <Your-Bucket-Name>.ss.bscstorage.com
Date: <date>
Content-Length: <object data length>
Content-Type: <mime-type>
Authorization: <authorization string> #请参照《签名算法》

[object data]
```

- 响应：

```
HTTP/1.1 200 OK
Date: Tue, 08 Apr 2014 02:59:47 GMT
Connection: keep-alive
ETag: "<文件的MD5值>"
x-amz-request-id: 0000106c-1608-0810-4621-00163e000064
x-amz-s2-requester: <Your UserName>
```

- Request Headers（请求头）：

Name	Description	Req
Expires	文件过期时间，到期系统将自动清除文件 (非即时，清除时间不定期)，格式参考： http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html#sec14.21	No
Cache-Control	文件Cache，标准HTTP协议，更多内容参见： http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html#sec14.9	No
Content-Type	文件mime type，读取时原值返回	No
Content-Length	文件大小，读取时原值返回	Yes
Content-MD5	base64编码的文件MD5（与传送内容不符时失败），注意：字符串格式为rfc标准使用base64编码的值	No
Content-Disposition	HTTP标准文件属性信息，读取时原值返回。 参见： http://www.w3.org/Protocols/rfc2616/rfc2616-sec19.html#sec19.5.1	No
Content-Encoding	文件编码，HTTP标准文件属性信息，读取时原值返回。参见： http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html#sec14.11	No
x-amz-acl	文件ACL：创建文件的同时，设置一个ACL。 请参照《ACL》	No
x-amz-meta-*	用户自定义MetaData，header以x-amz-meta-开头，所有meta以key:value的形式存储，最大限制64KB，HEAD或者GET时原样返回	No

- 请求示例：

```
curl -v -T "myfile.txt" -H "x-amz-meta-UploadLocation: My F
```

Append Object

- Append Object以追加写的方式上传文件，被追加的文件可以是bucket中任意方式上传的文件(除开分片上传的分片文件)，如: put Object, copy Object, 分片上传且已经合并的文件。

Append上传和put Object接口一致，除开以下两点：

- x-amz-meta-s2-append-position请求头，必选，用于标识append上传和上传文件追加在当前文件的位置
 - position的值为0时，如果当前追加的文件不存在或者文件的size等于0，则将数据追加到文件尾部后返回成功，否则返回409错误；
 - position的值大于0时，如果和当前文件的size相等，将数据追加到size开始的位置后返回成功，且修改文件的修改时间为当前时间；否则，返回409错误，且在response的x-amz-meta-append-position header中设置当前文件的size。
- x-amz-meta-s2-directive请求头，取值：COPY, REPLACE, 可选(则为COPY)，用于标识append文件是否覆盖文件的file meta信息

注意：

- 同一个position只能有一个请求能追加成功，且由于并发，设置正确的position也可能会返回失败，
- 同一文件最多只能追加2000次请求
- 单次请求最大文件大小为20G
- 追加文件的总大小最大1T

- 请求格式：

```
PUT /<ObjectName> HTTP/1.1
Host: <Your-Bucket-Name>.ss.bscstorage.com
Date: <date>
Content-Length: <object data length>
Content-Type: <mime-type>
x-amz-meta-s2-append-position: <position>
Authorization: <authorization string> #请参照《签名算法》

[object data]
```

- 响应：

```
HTTP/1.1 200 OK
Date: Tue, 08 Apr 2014 02:59:47 GMT
Connection: keep-alive
ETag: "<文件的MD5值>"
x-amz-request-id: 0000106c-1608-0810-4621-00163e000064
x-amz-s2-requester: <Your UserName>
x-amz-meta-s2-append-position: <position>
```

- Request Headers (请求头) :

Name	Description	
x-amz-meta-append-position	用于标识append上传和上传文件追加在当前文件的位置	、
x-amz-meta-s2-directive	指定是否使用被append文件的file meta, 取值: COPY,REPALCE, 默认为:COPY, 如果设置为COPY, 则使用append文件的file meta, 如果设置为REPLACE, 则使用当前append请求中携带的file meta	↑
Expires	文件过期时间, 到期系统将自动清除文件(非即时, 清除时间不定期), 格式参考: http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html#sec14.21 .	↑
Cache-Control	文件Cache, 标准HTTP协议, 更多内容参见: http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html#sec14.9	↑
Content-Type	文件mime type, 读取时原值返回	↑
Content-Length	文件大小, 读取时原值返回	、
Content-MD5	base64编码的文件MD5 (与传送内容不符时失败), 注意: 字符串格式为rfc标准使用base64编码的值	↑
Content-Disposition	HTTP标准文件属性信息, 读取时原值返回。 参见: http://www.w3.org/Protocols/rfc2616/rfc2616-sec19.html#sec19.5.1	↑
Content-Encoding	文件编码, HTTP标准文件属性信息, 读取时原值返回。参见: http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html#sec14.11	↑
x-amz-acl	文件ACL: 创建文件的同时, 设置一个ACL。 请参照《ACL》	↑
x-amz-meta-*	用户自定义MetaData, header以x-amz-meta-开头, 所有meta以key:value的形式存储, 最大限制64KB, HEAD或者GET时原样返回	↑

POST Object

- 描述：POST方式上传一个文件（基于浏览器表单的上传方式）。
 - 请求格式：

- #### • 表单元素：

Name	Description	Required
AWSAccessKeyId	就是AccessKey, 可以到控制台获取	Yes
key	object上传后的key (路径), 例如: angel/\${filename}, 变量 \${filename} 将被自动替换为上传文件的文件名; 当然也可以直接指定被上传文件存储在存储中的文件名。如: angel/path/to/myfile.txt, 变量名可以为: filename, sha1, md5, size	Yes
acl	文件的ACL: 创建文件的同时, 设置一个 ACL。请参照《ACL》	No
success_action_status	上传成功后的响应码, 可以设置的值为: 200, 201, or 204(defalut), 若设置为200或者204, 则返回body为空, status为200或者204, 若设置为201, 则返回xml格式的body, stats为201, 如果设置为非法的值, 则忽略该值, 使用默认值204 注: 如果设置 success_action_redirect 或者 redirect, 则忽略该设置	No
success_action_redirect, redirect	上传成功后客户端重定向的URL, 实际返回的location会在原来的URL加上bucket, key和etag querystring	No
Policy	文件的策略, json格式字符串, 并使用base64进行编码。后面详细介绍	Yes
Signature	使用SecretKey签名后的字符串。后面详细介绍	Yes
file	type=file的input表单	Yes

Name	Description	Required
x-amz-meta-*	用户自定义MetaData, header以x-amz-meta-开头, 所有meta以key:value的形式存储, 最大限制64KB, HEAD或者GET时原样返回	No
Cache-Control, Content-Type, Content-Disposition, Content-Encoding, Expires	和put_file一样, 参考put_file接口	No

Policy的构建:

策略是使用 UTF-8 和 Base64 编码的 JSON 文档, 它指定了请求必须满足的条件并且用于对内容进行身份验证。根据您设计策略文档的方式, 您可以对每次上传、每个用户、所有上传或根据其他能够满足您需要的设计来使用它们。

```
{
    "expiration": "2014-04-10T08:55:34.000Z",

    "conditions": [
        {
            "bucket": "my-bucket-name"
        },
        {
            "acl": "private"
        },
        [
            "starts-with", "$key", "my_prefix/"
        ],
        [
            "content-length-range", 0, 52428800
        ]
    ]
}
```

- 以上示例的说明:
 - 上传必须在"2014-04-10T08:55:34.000Z"之前。
 - 文件上传到名为"my-bucket-name"的bucket。

- starts-with: \$key必须以"my_prefix/"开始 (Policy中"\$key"前必须带"\$")。若\$key值为空, 文件名前无前缀。
- content-length-range: 文件大小必须在指定范围内。
- 最终将policy进行base64编码设置到表单Policy的value中。

expiration:

expiration用于指定policy的过期时间, 采用ISO 8601 UTC日期格式来指定策略的过期日期。例如, “2007-12-01T12:00:00.000Z”指定, 在策略中过期是必需的。

Condition:

Condition用于验证上传的对象的内容与表单冲填写的域

Condition 类型

Condition	Description
精确匹配	1. 精确匹配将验证字段是否匹配特定的值。此示例指示ACL 必须设置为公共读取: {"acl": "public-read" } 2.ACL 必须设置为公共读取的替代方法: ["eq", "\$acl", "public-read"]
Starts With	如果值必须从某个特定的值开始, 请使用starts-with。本示例指示密钥必须从user/betty 开始: ["starts-with", "\$key", "user/betty"]
指定范围	对于接受范围的字段, 请使用逗号来分隔上限和下限值。本示例允许1 到 10 MB 的文件大小: ["content-length-range", 1048579, 10485760], 单位字节

Conditions 字段

策略文档中的条件验证上传的对象的内容。您在表单中指定的每个表单字段 (AWSAccessKeyId、Signature、file、Policy除外) 都可以作为条件

Signature的构建:

- 用UTF-8对policy进行编码
- 用Base64对UTF-8形式的policy进行编码
- 用HMAC SHA-1和你的Secret Key将你的policy进行转换。最后进行base64编码。如: 使用php时, base64_encode(hash_hmac("sha1", \$policy, \$SECRETKEY, true));
- 将最终的值设置到表单Signature的value中。

- 最终生成的html表单:

- 注意事项：
 - POST请求后的uri只能是“/”
 - success_action_redirect：指定上传成功后客户端重定向的URL。
 - key：变量\${filename}将被自动替换成被上传文件的文件名；

PUT Object - Copy

- 描述：通过拷贝方式创建Object（不上传具体的文件内容。而是通过COPY方式对系统内另一文件进行复制）。
- 请求格式：

```
PUT /<ObjectName> HTTP/1.1
Host: <Your-Bucket-Name>.ss.bscstorage.com
Date: <date>
x-amz-copy-source: </source-bucket/source-object>
Authorization: <authorization string> #请参照《签名算法》
```

- 响应：

```
HTTP/1.1 200 OK
Date: Tue, 08 Apr 2014 02:59:47 GMT
Connection: keep-alive
ETag: "<文件的MD5值>"
x-amz-request-id: 0000106c-1608-0810-4621-00163e000064
x-amz-s2-requester: <Your UserName>

<?xml version="1.0" encoding="UTF-8"?>
<CopyObjectResult xmlns="http://s3.amazonaws.com/doc/2006-03-01">
    <LastModified>Mon, 08 Aug 2016 05:04:10 GMT</LastModified>
    <ETag>870c06c00566c4fb1861bb10f34d1904</ETag>
</CopyObjectResult>
```

- Request Headers (请求头) :

Name	Description	Req
x-amz-copy-source	被copy的文件地址。格式：/source-bucket/source-object，需要整体进行urlencode编码。	Yes
Cache-Control	文件Cache，标准HTTP协议，更多内容参见： http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html#sec14.9	No
Expires	文件在客户端或浏览器的缓存过期时间，允许客户端在这个时间之前不去服务端检查，读取时原值返回。格式为：Sun, 29 Jul 2018 20:36:14 UTC	No
Content-Type	文件mime type，读取时原值返回	No
Content-Length	必须是0	=0
Content-Disposition	HTTP标准文件属性信息，读取时原值返回。 参见： http://www.w3.org/Protocols/rfc2616/rfc2616-sec19.html#sec19.5.1	No
Content-Encoding	文件编码，HTTP标准文件属性信息，读取时原值返回。参见： http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html#sec14.11	No
x-amz-acl	文件ACL：创建文件的同时，设置一个ACL。 请参照《ACL》	No
x-amz-meta-*	用户自定义MetaData, header以x-amz-meta-开头，所有meta以key:value的形式存储,最大限制64KB, HEAD或者GET时原样返回	No
x-amz-copy-source-if-match	如果指定的ETag与源文件的ETag匹配，则可copy源文件，否则返回412(PreconditionFailed)	No
x-amz-copy-source-if-nonmatch	如果指定的ETag与源文件的ETag不匹配，则可copy源文件，否则返回412(PreconditionFailed)	No
x-amz-copy-source-if-unmodified-since	如果指定的时间开始源文件没有修改，则可copy源文件，否则返回412(PreconditionFailed)	No

Name	Description	Reqd
x-amz-copy-source-if-modified-since	如果指定的时间开始源文件有修改, 则可copy源文件, 否则返回412(PreconditionFailed)	No

- Response Body (响应XML Body) :

Name	Description
CopyObjectResult	包含ETag和LastModified元素
ETag	文件的ETag
CopyObjectResult	文件的最后修改时间

- 请求示例:

```
curl -v -X PUT -H "x-amz-copy-source: /bucket-123/path/to/source.txt" -H "Date: Sat, 20 Nov 2286 17:46:39 GMT" -H "Authorization: AWS <access-key>" https://s3.amazonaws.com/bucket-123/dest.txt
```

DELETE Object

- 描述：删除指定Object。
- 请求格式：

```
DELETE /<ObjectName> HTTP/1.1
Host: <Your-Bucket-Name>.ss.bscstorage.com
Date: <date>
Authorization: <authorization string> #请参照《签名算法》
```

- 响应（无HTTP Body）：

```
HTTP/1.1 204 No Content
Date: Tue, 08 Apr 2014 02:59:47 GMT
Content-Length: 0
Connection: keep-alive
x-amz-request-id: 0000106c-1608-0810-4621-00163e000064
x-amz-s2-requester: <Your UserName>
```

- 请求示例：

```
curl -v -X DELETE -H "Date: Sat, 20 Nov 2286 17:46:39 GMT"
```

GET Object ACL

- 描述：获得指定Object的ACL信息。更多信息请参照：[《ACL》](#)
- 请求格式：

```
GET /<ObjectName>?acl HTTP/1.1
Host: <Your-Bucket-Name>.ss.bscstorage.com
Date: <date>
Authorization: <authorization string> #请参照《签名算法》
```

- 响应：

```
HTTP/1.1 200 OK
Date: Tue, 08 Apr 2014 02:59:47 GMT
Content-Length: 123
Connection: keep-alive
x-amz-request-id: 0000106c-1608-0810-4621-00163e000064
x-amz-s2-requester: <Your UserName>

Json format:
Note: You can specify the XML with formatter request parameter
{
    "Owner": "Baishan0000000000000001",

    "ACL": {

        "GRPS00000ANONYMOUSE": [
            "read"
        ],

        "Baishan000001001NHT3M7": [
            "read",
            "write",
            "read_acp",
            "write_acp"
        ],

        "Baishan0000000000000001": [
            "read",
            "write",
            "read_acp",
            "write_acp"
        ],

        "Baishan000001001HBK3UT": [
            "read",
            "write",
            "read_acp",
            "write_acp"
        ],

        "GRPS0000000CANONICAL": [
            "read",
            "write",
            "read_acp",
            "write_acp"
        ],
        ...
    }
}
```

```
    }  
}
```

- 响应格式说明请参照： [《ACL》](#)

- 请求示例：

```
curl -v -H "Date: Sat, 20 Nov 2286 17:46:39 GMT" -H "Authorizat
```

PUT Object ACL

- 描述：给指定Object设置ACL规则。更多信息请参照：[《ACL》](#)
- 请求格式：

```
PUT /<ObjectName>?acl HTTP/1.1
Host: <Your-Bucket-Name>.ss.bscstorage.com
Date: <date>
Authorization: <authorization string> #请参照《签名算法》

#ACL规则: XML or Json
{
    "Baishan0000000000000001" : [ "read", "read_acp" , "write",
    "GRPS00000ANONYMOUSE" : [ "read", "read_acp" , "write",
    "GRPS000000CANONICAL" : [ "read", "read_acp" , "write"
}
```

- Request Headers (请求头) :

Name	Description	Required
x-amz-acl	文件ACL: 请参照《ACL》	No

```
HTTP/1.1 200 OK
Date: Tue, 08 Apr 2014 02:59:47 GMT
Connection: keep-alive
x-amz-request-id: 000106c-1608-0810-4621-00163e000064
x-amz-s2-requester: <Your UserName>
```

- Request Body (requestBody) : XML or JSON format acl body

Note: 假如指定了x-amz-acl header将忽略body中acl, body中的xml格式由请求参数formatter参数决定, 默认为XML

- Response (无HTTP Body) :
- 请求格式说明请参照：[《ACL》](#)
- 请求示例：

```
curl -v -T "acl.txt" -H "Date: Sat, 20 Nov 2286 17:46:39 GMT"
AWS <access_key>:<ssig>" "http://<Your-Bucket-Name>.ss.bscs
```

Initiate Multipart Upload

- 描述：大文件分片上传初始化接口
- 注意：在初始化上传接口中要求必须进行用户认证，匿名用户无法使用该接口。在初始化上传时可以给定文件上传所需要的meta绑定信息，在后续的上传中该信息将被保留，并在最终完成时写入云存储系统。
- 请求格式：

```
POST /<ObjectName>?uploads HTTP/1.1
Host: <Your-Bucket-Name>.ss.bscstorage.com
Date: <date>
Content-Type: <mime-type>
x-amz-meta-foo1: <value1> #自定义meta: foo1
x-amz-meta-foo2: <value2> #自定义meta: foo2
Authorization: <authorization string> #请参照《签名算法》
```

- 响应：

```
HTTP/1.1 200 OK
Date: Tue, 08 Apr 2014 02:59:47 GMT
Connection: keep-alive
x-amz-request-id: 0000106c-1608-0810-4621-00163e000064
x-amz-s2-requester: <Your UserName>

<?xml version="1.0" encoding="UTF-8"?>
<InitiateMultipartUploadResult xmlns="http://s3.amazonaws.com/doc/2006-03-01">
    <Bucket>your-bucket</Bucket>
    <Key>ObjectName</Key>
    <UploadId>VXBsb2FkIElEIGZvcIA2aWlpbmNcyBteS1tb3ZpZS5tH
</InitiateMultipartUploadResult>
```

- Request Headers (请求头) : 参考put_object
- Response Body (响应XML Body) :

Name	Description
InitiateMultipartUploadResult	包含Bucket, Key和UploadId元素
Bucket	bucket name
Key	objectName
UploadId	标识该分块上传的ID, 之后上传分片时需要携带该参数

- 请求示例:

```
curl -v -X POST "Date: Sat, 20 Nov 2286 17:46:39 GMT" -H "/"
```

Upload Part

- 描述：上传分片接口
- 请求格式：

```
PUT /<ObjectName>?partNumber=<PartNumber>&uploadId=<UploadId>
Host: <Your-Bucket-Name>.ss.bscstorage.com
Date: <date>
Content-Length: <Content-Length>
Content-MD5: <Content-MD5>
Authorization: <authorization string> #请参照《签名算法》

...
file_content
...
```

- 响应：

```
HTTP/1.1 200 OK
Date: Tue, 08 Apr 2014 02:59:47 GMT
Connection: keep-alive
x-amz-request-id: 0000106c-1608-0810-4621-00163e000064
x-amz-s2-requester: <Your UserName>
Etag: <Etag>
```

- Request Parameters(请求参数)：

Parameter	Description	Required
partNumber	文件分片的序号，从1开始	Yes
uploadId	通过Initiate Multipart Upload (大文件分片上传初始化接口) 获得的uploadId值	Yes

- Request Headers(请求头)：

Name	Description	Required
Content-Length	文件大小，读取时原值返回	Yes
Content-MD5	base64编码的文件MD5 (与传送内容不符时失败)，注意：字符串格式为rfc标准使用base64编码的值	No

- 注意：

Python

- 分片数不能超过2000。

Upload Part - Copy

- 描述：通过拷贝方式创建上传一个分片（不上传具体的文件内容。而是通过COPY方式对系统内另一文件进行复制）。
- 请求格式：

```
PUT /<ObjectName>?partNumber=PartNumber&uploadId=UploadId
Host: <Your-Bucket-Name>.ss.bscstorage.com
Date: <date>
x-amz-copy-source: </source-bucket/source-object>
Authorization: <authorization string> #请参照《签名算法》
```

- 响应：

```
HTTP/1.1 200 OK
Date: Tue, 08 Apr 2014 02:59:47 GMT
Connection: keep-alive
ETag: "<文件的MD5值>"
x-amz-request-id: 0000106c-1608-0810-4621-00163e000064
x-amz-s2-requester: <Your UserName>

<?xml version="1.0" encoding="UTF-8"?>
<CopyObjectResult xmlns="http://s3.amazonaws.com/doc/2006-03-01">
    <LastModified>Mon, 08 Aug 2016 05:04:10 GMT</LastModified>
    <ETag>870c06c00566c4fb1861bb10f34d1904</ETag>
</CopyObjectResult>
```

- Request Parameters(请求参数)：

Parameter	Description	Required
partNumber	文件分片的序号，从1开始	Yes
uploadId	通过Initiate Multipart Upload (大文件分片上传初始化接口) 获得的uploadId值	Yes

- Request Headers (请求头)：

Name	Description	Required
x-amz-copy-source	被copy的文件地址。格式: /source-bucket/source-object, 需要整体进行urlencode编码.	Yes
Content-Length	必须是0	=0
x-amz-copy-source-if-match	如果指定的ETag与源文件的ETag匹配, 则可copy源文件, 否则返回412(PreconditionFailed)	No
x-amz-copy-source-if-nonmatch	如果指定的ETag与源文件的ETag不匹配, 则可copy源文件, 否则返回412(PreconditionFailed)	No
x-amz-copy-source-if-unmodified-since	如果指定的时间开始源文件没有修改, 则可copy源文件, 否则返回412(PreconditionFailed)	No
x-amz-copy-source-if-modified-since	如果指定的时间开始源文件有修改, 则可copy源文件, 否则返回412(PreconditionFailed)	No

- Response Body (响应XML Body) :

Name	Description
CopyObjectResult	包含ETag和LastModified元素
ETag	文件的ETag
CopyObjectResult	文件的最后修改时间

- 请求示例:

```
curl -v -X PUT -H "x-amz-copy-source: /bucket-123/path/to/" -H "Date: Sat, 20 Nov 2286 17:46:39 GMT" -H "Authorization: AWS <access-key>" https://s3.amazonaws.com/bucket-123/path/to/
```

Complete Multipart Upload

- 描述：大文件分片上传拼接完成接口（合并接口）
- 请求格式：

```
POST /<ObjectName>?uploadId=<UploadId> HTTP/1.1
Host: <Your-Bucket-Name>.ss.bscstorage.com
Date: <date>
Content-Type: text/json
Authorization: <authorization string> #请参照《签名算法》

<CompleteMultipartUpload>
    <Part>
        <PartNumber>1</PartNumber>
        <ETag>"a54357aff0632cce46d942af68356b38"</ETag>
    </Part>
    <Part>
        <PartNumber>2</PartNumber>
        <ETag>"0c78aef83f66abc1fa1e8477f296d394"</ETag>
    </Part>
    <Part>
        <PartNumber>3</PartNumber>
        <ETag>"acbd18db4cc2f85cedef654fcc4a4d8"</ETag>
    </Part>
</CompleteMultipartUpload>
```

- 响应：

```
HTTP/1.1 200 OK
Date: Tue, 08 Apr 2014 02:59:47 GMT
Connection: keep-alive
x-amz-request-id: 000106c-1608-0810-4621-00163e000064
x-amz-s2-requester: <Your UserName>

<?xml version="1.0" encoding="UTF-8"?>
<CompleteMultipartUploadResult xmlns="http://s3.amazonaws.com/doc/2006-03-01">
    <Location>http://Example-Bucket.ss.bscstorage.com/Example-Object</Location>
    <Bucket>Example-Bucket</Bucket>
    <Key>Example-Object</Key>
    <ETag>"3858f62230ac3c915f300c664312c11f-9"</ETag>
</CompleteMultipartUploadResult>
```

- Request Parameters(请求参数)：

Parameter	Description	Required
uploadId	通过Initiate Multipart Upload（大文件分片上传初始化接口）获得的uploadId值	Yes

- Request Body（请求Body）：

Name	Description	Required
PartNumber	文件分片的序号	Yes
ETag	通过Upload Part（上传分片接口）上传成功后返回的响应头中的Etag值	Yes

- Response Body（响应XML Body）：

Name	Description	Required
CompleteMultipartUploadResult	包含Location, Bucket, Key 和ETag元素	Yes
Location	URI标识新上传的文件	Yes
Bucket	bucket name	Yes
Key	objectName	Yes
ETag	文件的ETag, 该ETag和put_object的ETag不同, 并不是文件的md5值	Yes

List Parts

- 描述：列出已经上传的所有分块
- 请求格式：

```
GET /<ObjectName>?uploadId=<UploadId> HTTP/1.1
Host: <Your-Bucket-Name>.ss.bscstorage.com
Date: <date>
Authorization: <authorization string> #请参照《签名算法》
```

- 响应：

```
HTTP/1.1 200 OK
Date: Tue, 08 Apr 2014 02:59:47 GMT
Connection: keep-alive
x-amz-request-id: 0000106c-1608-0810-4621-00163e000064
x-amz-s2-requester: <Your UserName>

<?xml version="1.0" encoding="UTF-8"?>
<ListPartsResult xmlns="http://s3.amazonaws.com/doc/2006-03-01">
<Bucket>example-bucket</Bucket>
<Key>example-object</Key>
<UploadId>XXBsb2FkIElEIGZvcIBlbHZpbmcncyVcdS1tb3ZpZS5tMnRzf
<Initiator>
    <ID>initiator</ID>
    <DisplayName></DisplayName>
</Initiator>
<Owner>
    <ID>owner</ID>
    <DisplayName></DisplayName>
</Owner>
<StorageClass>STANDARD</StorageClass>
<PartNumberMarker>1</PartNumberMarker>
<NextPartNumberMarker>3</NextPartNumberMarker>
<MaxParts>2</MaxParts>
<IsTruncated>true</IsTruncated>
<Part>
    <PartNumber>2</PartNumber>
    <LastModified>2010-11-10T20:48:34.000Z</LastModified>
    <ETag>"7778aef83f66abc1fa1e8477f296d394"</ETag>
    <Size>10485760</Size>
</Part>
<Part>
    <PartNumber>3</PartNumber>
    <LastModified>2010-11-10T20:48:33.000Z</LastModified>
    <ETag>"aaaa18db4cc2f85cedef654fcc4a4x8"</ETag>
    <Size>10485760</Size>
</Part>
</ListPartsResult>
```

- Request Parameters (请求参数) :

Parameter	Description	Required
uploadId	通过Initiate Multipart Upload (大文件分片上传初始化接口) 获得的 uploadId值	Yes
max-parts	返回最大的part数量, 默认返回2048个	Yes
part-number-marker	列出该partNumber之后的parts, 不包括该partNUmber指定的part, partNUmber从1开始	Yes

- Request Headers (请求头) :
- Response Body (响应XML Body) :

Name	Description
ListPartsResult	包含Bucket, Key, UploadId, Initiator, Owner, StorageClass, PartNumberMarker, NextPartNumberMarker, MaxParts, IsTruncated, Part元素
Bucket	bucket name
Key	objectName
UploadId	标识该分块上传的ID, 之后上传分片时需要携带该参数
Initiator	包含ID, DisplayName元素
ID	init UserName
DisplayName	
Owner	包含ID, DisplayName元素, 指定文件的owner
StorageClass	存储类别
PartNumberMarker	list开始的partNumber, 不包含该partNumber
NextPartNumberMarker	下次开始list的partNumber
MaxParts	最大part数量
IsTruncated	是否被truncated
Part	包含PartNumber, LastModified, ETag, Size元素, 表示一个part的信息
PartNumber	part的number
LastModified	part的最后修改时间
ETag	part的md5
Size	part的size

临时凭证使用流程

当在白山云存储开通存储账号以后可以得到访问存储服务需要的access key和secret key，这是永久访问凭证，而且具有任何操作权限，因此为了防止泄露该永久凭证，不适合将其存放在移动客户端等不安全的地方。

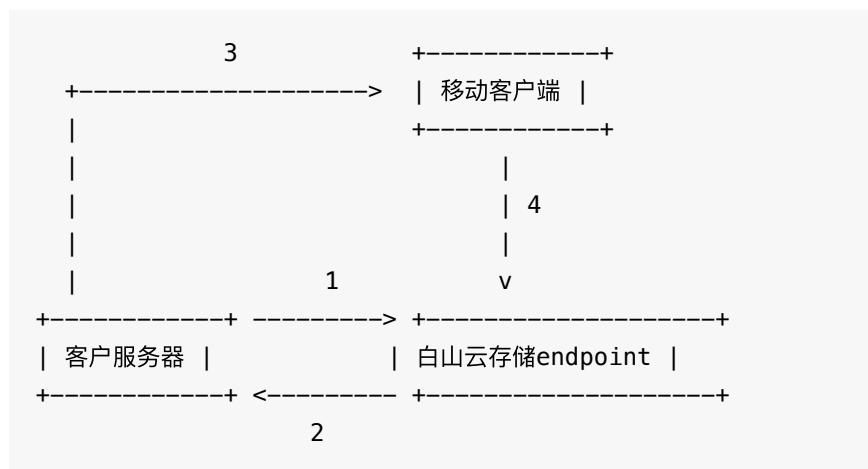
当需要从移动客户端访问存储服务时，就可以使用临时凭证，此时，永久凭证存放在客户服务器上，客户服务器使用永久凭证向白山云存储节点索取临时凭证，然后将得到的临时凭证发送给移动客户端，移动客户端再使用临时凭证访问存储服务。

临时凭证包括三部分：临时 access_key，临时 secret_key，token。使用临时凭证和使用永久凭证的方法类似，只是需要多加一个 token，如果使用SDK，这个token可以通过参数传入。

临时凭证是有有效期的，可以在获取临时凭证时设置有效期的长短，可以设置为15分钟到60分钟。

临时凭证的权限也可以在获取的时候设置，如可以设置为只能上传，或只能下载，或只能上传到指定目录等。

STS使用流程



1. 客户服务器访问白山云存储节点提供的STS服务获取临时凭证。
2. 云存储STS服务返回临时凭证。
3. 客户服务器将临时凭证发送给移动客户端。
4. 移动客户端使用临时凭证访问云存储服务。

示例

使用STS服务的SDK访问白山云存储STS服务， endpoint为

`http://sts.ss.bscstorage.com`。

申请临时凭证常用到下面三个参数：

- Policy: 用来描述分配的权限, 它是一个json字符串, 只需要填充里面的 Statement 字段。
 - Effect : Allow 表示允许, Deny 表示拒绝。
 - Resource : 用来描述哪些资源会被分配权限, * 代表所有。
 - Action : 表示被拒绝或者允许的动作, * 代表所有。
 - s3:GetObject
 - s3:GetObjectAcl
 - s3:PutObject
 - s3:DeleteObject
 - s3:PutObjectAcl
 - s3>ListMultipartUploadParts
 - s3:AbortMultipartUpload
 - s3>ListAllMyBuckets
 - s3>CreateBucket
 - s3>DeleteBucket
 - s3:GetBucketVersioning
 - s3:GetBucketCORS
 - s3:PutBucketAcl
 - s3:PutBucketVersioning
 - s3:PutBucketCORS
 - s3>ListBucket
 - s3>ListBucketMultipartUploads
 - s3:GetBucketPolicy
 - s3:PutBucketPolicy
 - s3:DeleteBucketPolicy
 - s3:GetBucketAcl

Eg: `{"Statement": [{"Resource": "\\", "Action": "\\\\"s3:\\", "Effect": "Allow"}]}` 表示允许所有方法访问所有资源。

- RoleArn: 资源名称, 字符串类型, 用户自定义, 长度必须大于20。
- RoleSessionName: session名称, 字符串类型, 用户自定义。

使用Java SDK的示例如下：

```

// 通过永久凭证生成临时凭证
String stsEndPoint = "http://sts.ss.bscstorage.com";
String accessKey = "ziwxxxxxxxxxxxxxxx";
String secretKey = "V+ZTZ5xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx";
BasicAWSCredentials awsCreds = new BasicAWSCredentials(
    accessKey, secretKey);

AWSecurityTokenService sts = AWSecurityTokenServiceClient
    .builder()
    .credentialsProvider(new AWSStaticCredentialsProvider(awsCreds))
    .endpointConfiguration(new EndpointConfiguration(stsEndPoint))

String policy = "{\"Statement\": [{\"Resource\":\"*\", \"Action\": \"*\"}]}";

AssumeRoleRequest assumeRoleRequest = new AssumeRoleRequest();
assumeRoleRequest.setPolicy(policy);
assumeRoleRequest.setRoleArn("arn:aws:iam::123456789:role/test-role");
assumeRoleRequest.setRoleSessionName("test-session");

AssumeRoleResult assumeRoleResult = sts.assumeRole(assumeRoleRequest);

String tmp_access_key = assumeRoleResult.getCredentials().getAccessKeyId();
String tmp_secret_key = assumeRoleResult.getCredentials().getSecretAccessKey();
String token = assumeRoleResult.getCredentials().getSessionToken();

// 使用服务器返回的临时凭证
String s3EndPoint = "http://ss.bscstorage.com";

BasicSessionCredentials sessionCredentials = new BasicSessionCredentials(
    tmp_access_key, tmp_secret_key, token);

AmazonS3 s3 = AmazonS3ClientBuilder.standard().withCredentials(
    new AWSStaticCredentialsProvider(sessionCredentials))
    .endpointConfiguration(new EndpointConfiguration(s3EndPoint, ''));

S3Object obj = s3.getObject("bucket", "key");

```

使用Python SDK示例如下：

```

import boto3

cli = boto3.client(
    'sts',
    region_name="bj",
    aws_access_key_id="accesskey",
    aws_secret_access_key="secretkey",
    endpoint_url='http://sts.ss.bscstorage.com'
)

policy = "{\"Statement\": [{\"Resource\":\"*\", \"Action\": ["
# 使用永久凭证生成临时凭证
resp = cli.assume_role(
    DurationSeconds=3600,
    RoleArn='arn:aws:iam::123456789012:role/demo',
    RoleSessionName="test",
    Policy=policy,
)

print("tmp accesskey: %s" % resp['Credentials']['AccessKeyId'])
print("tmp secretkey: %s" % resp['Credentials']['SecretAccessKey'])
print("expired time : %s" % resp['Credentials']['Expiration'])
print("session token: %s" % resp['Credentials']['SessionToken'])

# 使用临时凭证创建client
tmpcli = boto3.client(
    "s3",
    region_name="bj",
    aws_access_key_id=resp['Credentials']['AccessKeyId'],
    aws_secret_access_key=resp['Credentials']['SecretAccessKey'],
    aws_session_token=resp['Credentials']['SessionToken'],
    endpoint_url='http://ss.bscstorage.com',
)

print tmpcli.head_object(Bucket='bucket', Key='key')

```

使用PHP SDK示例如下：

```

<?php

require 'aws.phar';

try {
    $cli = new Aws\Sts\StsClient([
        'version' => 'latest',
        'region' => 'us-east-1',
        'credentials' => [
            'key' => 'accesskey',
            'secret' => 'secretkey',
        ],
        'endpoint' => 'http://sts.ss.bscstorage.com',
    ]);
    $result = $cli->assumeRole([
        'DurationSeconds' => 3600,
        'Policy' => "{\"Statement\": [{\"Resource\":\"*\", \"Action\": \"sts:AssumeRole\", \"Condition\": {}}]}",
        'RoleArn' => 'arn:aws:iam::123456789012:role/demo',
        'RoleSessionName' => 'test', // REQUIRED
    ]);

    $tmp_accesskey = $result['Credentials']['AccessKeyId'];
    $tmp_secretkey = $result['Credentials']['SecretAccessKey'];
    $tmp_sesstoken = $result['Credentials']['SessionToken'];

    echo "tmp accesskey:", $tmp_accesskey, "\n";
    echo "tmp secretkey:", $tmp_secretkey, "\n";
    echo "session token:", $tmp_sesstoken, "\n";

    $s3 = new Aws\S3\S3Client([
        'version' => 'latest',
        'region' => 'us-east-1',
        'credentials' => [
            'key' => $tmp_accesskey,
            'secret' => $tmp_secretkey,
            'token' => $tmp_sesstoken,
        ],
        'endpoint' => 'http://ss.bscstorage.com',
    ]);

    $resp = $s3->getObject([
        'Bucket' => 'bucket',
        'Key' => 'key',
    ]);

    echo $resp;
} catch (\Exception $e) {

```

```
    echo "Exception: " . $e->getMessage() . "\n";
}
```

使用dotnet SDK示例如下：

```
static async Task StsDemo()
{
    AmazonSecurityTokenServiceConfig config = new AmazonSecurityTokenServiceConfig();
    config.ServiceURL = "http://sts.ss.bscstorage.com";
    config.SignatureVersion = "4";
    AmazonSecurityTokenServiceClient sts = new AmazonSecurityTokenServiceClient(config);
    var response = await sts.AssumeRoleAsync(new AssumeRoleRequest()
    {
        Policy = "{\"Statement\": [{\"Resource\":\"*\", \"Action\": \"sts:AssumeRole\", \"Effect\": \"Allow\"}], \"Version\": \"2012-10-17\"}",
        RoleArn = "arn:aws:iam::123456789012:role/demo",
        RoleSessionName = "testAssumeRoleSession",
    });

    Credentials credentials = response.Credentials;
    Console.WriteLine("tmp accesskey:" + credentials.AccessKey);
    Console.WriteLine("tmp secretkey:" + credentials.SecretKey);
    Console.WriteLine("tmp sesstoken:" + credentials.SessionToken);

    AmazonS3Config s3config = new AmazonS3Config();
    s3config.ServiceURL = "http://ss.bscstorage.com";
    s3config.SignatureVersion = "4";
    AmazonS3Client s3 = new AmazonS3Client(credentials.AccessKey, credentials.SecretKey, s3config);
    var resp = await s3.GetObjectMetadataAsync("bucket", "file");
    Console.WriteLine(resp.HttpStatusCode);
}
```

Offline Download Interface

Authorization

Use AccessKey and SecretKey.

Please refer to [REST-Authentication](#).

Error Response Status Code and Error

Following is a list of error response definitions, including:

Status code Error code and its description.

- **400** InvalidURI
Couldn't parse the specified URI.
- **400** DownloadSourceException
Download source exception can not download file.
- **404** NoSuchDownloadSource
The specified download source does not exist.
- **500** InternalError
We encountered an internal error. Please try again.
- **501** NotImplemented
Current request not implemented.

Other status code is same as Amazon S3. Please refer to [ErrorResponses](#).

Signature of all requests is same as Amazon S3.

Commit Offline Download Task

Request URI

`http://offline_domain/<bucket>`

Request Method

PUT

Request Body

Data of JSON format.

Following is a list of request content, including:

Element Name Element Type and its description.

- Elements for ALL requests:

- **Url** String

Download resource URL (including BT, Magnet, HTTP).

- **SuccessCallbackUrl** String

URL callback success.

- **FailureCallbackUrl** String

URL callback failure.

- **Target** String *optional*

Prefix of **Key** for the file to store.

- **ACLs** Dict *optional*

Access control lists. you can see ACL part in s2 doc for detail

Examples:

- Specify the upload file's ACL:

```
task['ACLs'] = {'acl': 'private'} ,
task['ACLs'] = {'acl': 'public-read'} ,
task['ACLs'] = {'acl': 'authenticated-read'}
```

- Authorize the specified user's access to this file:

```
task['ACLs'] = {'grant-read': 'id=user_name',
'grant-write': 'id=user_name', 'grant-read-acp':
'id=user_name', 'grant-write-acp':
'id=user_name'}
```

- Authorize the specified multi users' access to this file:

```
task['ACLs'] = {'grant-full-control':
'id=user_name'}
```

```
task['ACLs'] = {'grant-read': 'user_name1',
'grant-write': 'user_name2'}
```

```
task['ACLs'] = {'grant-read-acp':
'emailAddress=user_email', 'grant-write-acp':
'id=user_email'}
```

- Elements for BT request:

- **Files** Array

Contains files which user wanted.

- Elements for HTTP request:

- **Key** `String optional`

Specify put object url, if this field is set, **Target** field will be ignored.

- **SHA1** `String optional`

SHA1 hash of this file. Check the file's accuracy.

- **MD5** `String optional`

MD5 hash for this file. Check the file's accuracy.

- **CRC32** `String optional`

CRC32 hash of this file. Check the file's accuracy.

- **Size** `Int optional`

Size of this file. Check the file's accuracy.

- **OnKeyExist** `String optional`

specify this file's ignore rule.

Ignore rules (If the same `key file` exists):

- `ignore` : Always ignore.
- `ignore-same-md5` : This key just work with `MD5` key. If this file with the same MD5 value exists, ignore this file.
- `ignore-same-sha1` : This key just work with `SHA1` key. If this file with the same SHA1 value exists, ignore this file.
- `ignore-same-crc32` : This key just work with `CRC32` key. If this file with the same CRC32 value exists, ignore this file.
- `ignore-same-checksum` : This key just work with `MD5` or `SHA1` or `CRC32` key. If this file with the same value of any one of the three exists, ignore this file.
- `ignore-same-size` : This just work with `Size` key. If this file with the same size exists, ignore this file.
- `override` : Always override.

Request Example

```
//BT method.  
{  
    "Files": ["movie1.txt", "movie2.jpg"],  
    "Url": "http://www.abc.com/download/movie.torrent",  
    "Target": "movie/2016",  
    "SuccessCallbackUrl": "http://website/succ/",  
    "FailureCallbackUrl": "http://website/fail/",  
}
```

or

```
//HTTP method.  
{  
    "Url": "http://www.abc.com/download/movie.mp4",  
    "Key": "movies/2016/movie.mp4",  
    "ACLs": {'acl': 'private'},  
    "SHA1": "abcdeabcdeabcdeabcdeabcdeabcdeabcde",  
    "MD5": "abcdabcdabcdabcdabcdabcdabcd",  
    "CRC32": "abcdef",  
    "SuccessCallbackUrl": "http://website/succ/",  
    "FailureCallbackUrl": "http://website/fail/",  
}
```

or

```
//Magnet method.  
{  
    "Url": "magnet:?xt=urn:btih:88594AAACBDE40EF3E2510C47374",  
    "Target": "movie/2016",  
    "SuccessCallbackUrl": "http://website/succ/",  
    "FailureCallbackUrl": "http://website/fail/",  
}
```

Request Response

- **Success**

Return status code 200.

- **Failure**

Same as Amazon, refer to [ErrorResponses](#).

Request Response Header

After request handled successful, it generated a `x-amz-s2-offline-task-id` header, contains this offline download task ID. The subsequent query task and cancel task both can be done by looking up this ID.

Query Status Of Offline Download Task

Request URI

```
http://offline_domain/<bucket>/<taskId>
```

Request Method

GET

Request Response

- Success

Return status code 200 and body.

Body contains **State** and **Percent** (total download progress percent). (Json type)

Example:

```
{  
    "State": "TASK_UNDO",  
    "Percent": "0.00"  
}
```

or

```
{  
    "State": "TASK_DOING",  
    "Percent": "80.00"  
}
```

or

```
{  
    "State": "TASK_SUCCESS",  
    "Percent": "100.00"  
}
```

Python

or

```
{  
    "State":"TASK_FAILED"  
}
```

or

```
{  
    "State":"TASK_CANCELLED"  
}
```

- **Failure**

Same as Amazon, refer to [ErrorResponses](#).

Delete Offline Download Task

Request URI

```
http://offline_domain/<bucket>/<taskId>
```

Request Method

DELETE

Request Response

- **Success**

Return status code 204.

- **Failure**

Same as Amazon, refer to [ErrorResponses](#).

notice: current delete download task is not open to user, if you want to use this feature contact us.

Success Callback

Request URL

Determined by argument of `SuccessCallbackUrl` Request.

Request Method

POST

Request Body

Data of JSON format.

Following is a list of request content, including:

Element Name Element Type and its description.

Not null for ALL of these elements.

- **ETag** String

MD5 hash of this file. This md5 value must be enclosed in double quotes.

- **MD5** String

MD5 hash of this file.

- **SHA1** String

SHA1 hash of this file.

- **CRC32** String

CRC32 hash of this file.

- **Key** String

Name of this file. (e.g. <Target>/<File> or <Key>)

- **Size** Int

Size of this file.

- **Content-Type** String

MIME type of this file (Converted to lowercase letters)

- **LastModified** String

Upload time. (Unit: ms)

- **Task** String

Name of this offline download task. (e.g. <bucket>/<taskId>)

- **Concrete-Upload** bool

Concrete upload file to storage, (offline will ignore upload file if it already existed in storage)

- **Success-IDC** String

file has uploaded succeed idc list, use comma to split idcs. if Concrete-Upload value is false, this field is null

- **Failure-IDC** String

file has uploaded failed idc list, use comma to split idcs. if Concrete-Upload value is false, this field is null

Request Example

```
{
  "movie1": {
    "ETag": "\"abcdabcdabcdabcdabcdabcd\"",
    "MD5": "abcdabcdabcdabcdabcdabcd",
    "SHA1": "abcdeabcdeabcdeabcdeabcdeabcdeabcde",
    "CRC32": "fasdfasdf",
    "Key": "offline/prefix/movie1",
    "Size": "123456",
    "Content-Type": "video\\mp4",
    "LastModified": "1467355438043.852",
    "Task": "bucket/0000d1f0010000000062",
    "Concrete-Upload": true,
    "Success-IDC": "idc1,idc2",
    "Failure-IDC": ""
  },
}

or

{
  "movie1": {
    "ETag": "\"abcdabcdabcdabcdabcdabcd\"",
    "MD5": "abcdabcdabcdabcdabcdabcd",
    "SHA1": "abcdeabcdeabcdeabcdeabcdeabcde",
    "CRC32": "fasdfasdf",
    "Key": "offline/prefix/movie1",
    "Size": "123456",
    "Content-Type": "video\\mp4",
    "LastModified": "1467355438043.852",
    "Task": "bucket/0000d1f0010000000062",
    "Concrete-Upload": false,
    "Success-IDC": null,
    "Failure-IDC": null
  },
}
```

Request Response

- **Success**

Return status code 200.

- **Failure**

Same as Amazon, refer to [ErrorResponses](#).

Failure Callback

Request URL

Determined by argument of FailureCallbackUrl Response.

Request Method

POST

Request Body

Data of JSON format.

Following is a list of request content, including:

Element Name Element Type and its description.

Not null for ALL of these elements EXCEPT **Files**.

- **Code** String

Error code.

- **Message** String

Error description information.

- **Resource** String

Resource name, contains URL of this offline request task.

- **RequestId** String

Request ID.

- **Task** String

Name of this offline download task. (e.g. <bucket>/<taskId>)

- **Files** dict optional

Contains information of files which has been uploaded. With this element, users can pick up information of files which has been uploaded sucessful.

Request Example

```
{  
    "Code": "NoSuchDownloadSource",  
    "Message": "The specified download source does not exist",  
    "Resource": "http://www.abc.com/download/movie.torrent",  
    "RequestId": "4442587FB7D0A2F9",  
    "Task": "bucket/0000d1f001000000062",  
}
```

or

Request Response

- Success

Return status code 200.

- Failure

Same as Amazon, refer to [ErrorResponses](#).



Imgx(图片处理服务)使用说明

标签： 图片处理 imgx 白山云 存储服务

URL格式

```
http://imgx-ss.bscstorage.com/<bucket>/<处理指令>/<文件路径>?
```

或者

```
http://<bucket>.imgx-ss.bscstorage.com/<处理指令>/<文件路径>?
```

- `bucket` : 您在云存储服务中的bucket名称;
- `处理指令` : 对原始图片的处理指令, 下面有详细介绍;
- `文件路径` : 原始图片文件在云存储服务中的存储路径;
- `签名保护` : 主要是签名相关的参数;

举例

例如: 原始图片在云存储中的url为: <http://ss.bscstorage.com/imgx-test/demo/1.jpg>

`AWSAccessKeyId=acc_drdrxp&Expires=2464773061&Signature=jFVHR SrOLeg5e3nlR00UE2vik0A%3D` 可以看出:

- `bucket`为: `imgx-test`
- 在`bucket`中的路径为: `demo/1.jpg`

下面我们进行下列操作:

- 裁剪人脸区域, 并将图片缩略:
`400x400: c_thumb,g_face,w_400,h_400`
- 将图片变成圆角, 半径为最大(正圆): `r_max`
- 将图片亮度增加8%: `e_brightness:8`
- 将图片格式转换为png格式: `f_png`

处理指令

为: `c_thumb,g_face,w_400,h_400,r_max,e_brightness:8,f_png`

我们可以通过URL直接进行访问:

```
http://imgx-ss.bscstorage.com/imgx-test/c_thumb,g_face,w_400,h_400,r_max,e_brightness:8,f_png?
```

或者

```
http://imgx-test.imgx-ss.bscstorage.com/c_thumb,g_face,w_400,h_400,l_text:my_font:hello+world,w_100,h_40
```

或者创建一个json文件(如果您不想将处理指令暴露在URL中), 内容为:

```
[  
  {  
    "crop" : "thumb",  
    "gravity" : "face",  
    "width" : 400,  
    "height" : 400,  
    "radius" : "max",  
    "effect" : "brightness:8",  
    "format" : "png"  
  }  
]
```

将文件保存到您对应的bucket下的路径:

```
imgx/cmd_template/my_thumb.json      #其中文件名(my_thumb)是您自定义的文件名
```

然后就可以通过下面的URL进行访问 (也就是说, 处理指令也可以不用放在URL中, 直接隐藏到您自定义的json文件中) :

```
http://imgx-ss.bscstorage.com/imgx-test/t_my_thumb/demo/1.:
```

指令格式

- 指令名和指令值之间用“_”连接, 如: c_fit
- 相关指令使用逗号隔开“,”如: c_fit,w_100,h_100,g_face
- 多组指令之间用“--”隔开, 如:
c_fit,w_100,h_100,g_face,r_max--
l_text:my_font:hello+word,w_100,h_40

签名

为了保护您的原图不被盗取以及处理指令不被恶意滥用, 所以这里必须使用签名保护, 签名形式完全兼容AWS-V2认证, 以下提供一个生成代签名URL的php函数:

```

/**
 * Get a query string authenticated URL
 *
 * @param string $accessKey AccessKey
 * @param string $secretKey SecretKey
 * @param string $bucket Bucket name
 * @param string $uri Object URI
 * @param integer $lifetime Lifetime in seconds
 * @param boolean $hostBucket Use the bucket name as the host
 * @param boolean $https Use HTTPS ($hostBucket should be false)
 * @return string
 */
function getAuthenticatedURL($accessKey, $secretKey, $bucket, $uri, $lifetime, $hostBucket, $https) {
    $expires = time() + $lifetime;
    $uri = str_replace(array('%2F', '%2B', '%2C', '%3A', '%3D'), array('/', '+', ',', '=', '='), $uri);
    return sprintf(($https ? 'https' : 'http').'://%s/%s?AWSAccessKeyId=%s&Expires=%d&Signature=%s', $hostBucket ? $bucket : $endpoint.'/'.$bucket, $uri, $accessKey, $expires, urlencode(base64_encode(hash_hmac('sha1', "GET\n\n\n{$uri}")));
}

//用法很简单
echo getAuthenticatedURL('您的accessKey', '您的secretKey', 'http://oss-cn-hangzhou.aliyuncs.com/oss/hello/world.png', 3600);
# 建议$lifetime设置比较长, 例如: 1000000000

```

缓存 & CDN

- 处理后的图片生成缓存，下次请求不在重复生成， 默认缓存2天
- 如果配置了CDN， 处理后的图片会自动推送到CDN节点

图片处理指令

以下介绍具体处理指令，用下列几张原图为例，方便您对照：

1. [demo/charles.png](#)
2. [demo/1.jpg](#)
3. [demo/3.png](#)
4. [demo/4.png](#)
5. [demo/sheep.png](#)
6. [demo/horses.png](#)
7. [avatar.png](#)

指令名	指 令	Value	示 例	
crop	c	mode		裁剪方式,
		scale		改变图像的大小 有原始图像的倍 而变形。 c_scale,h_80
		fill		裁剪图像, 同时 c_fill,h_80
		lfill		同 fill 模式, 不 c_lfill,h_80
		fit		改变图像的大小 时保留原有比例 的。等比放缩, c_fit,h_80,1
		mfit		同 fit 模式, 不 c_mfit,h_80
		limit		同 fit, 不同的是 不会超过原图。 c_limit,h_80
		pad		指定图像的尺寸 比例不满足指定 c_pad,h_80,1
		lpad		同 pad 模式, > 图, 将不扩大原 c_lpad,h_64
		mpad		同 pad 模式, > c_mpad,h_80
		crop		指定尺寸和位置 出一部分。 c_crop,h_21
		thumb		定位人脸 (结合 缩略图, 常用于 c_thumb,h_21

指令名	指 令	Value	示 例
crop	c	<i>mode</i>	裁剪方式,
width	w	像素或者百分比	宽度参数, :
		80	 调整宽度为80像 w_80
		0.1	 调整图像到其原 w_0.1
height	h	像素或者百分比	高度参数, :
		80	 调整高度为80像 h_80
		0.1	 调整图像到其原 h_0.1
gravity	g	用于指定位置 或者方向	1. 用于 'cr 2. 用于:
		north_west	 左上位置 c_crop,g_no
		north	 正上位置, 水平 c_crop,g_no
		north_east	 右上位置 c_crop,g_no
		west	 左边, 垂直方 c_crop,g_we
		center	 正中 c_crop,g_ce
		east	 右边, 垂直方 c_crop,g_ea
		south_west	 左下位置 c_crop,g_so

指令名	指 令	Value	示 例
crop	c	mode	裁剪方式,
		south	正下位置, 水平  c_crop,g_sou
		south_east	右下位置  c_crop,g_sou
		xy_center	指定的x,y坐标,  c_crop,g_xy h_400,w_400,:)
		face	自动定位人脸的 易识别的一个  c_crop,g_fa
		face (thumb)	自动定位人脸的 缩略图。如果有  c_thumb,g_fa
		faces	自动定位多张人 脸  c_thumb,g_fa e_brightness
		face:center	自动定位人脸的 位到原图的中心  c_thumb,g_fa
		faces:center	自动定位多张人 脸 动定位到原图的  c_thumb,g_fa h_120,w_330,:)
x	x	像素	用于指定图
		110	裁剪图像180x1  c_crop,h_180
y	y	像素	用于指定图
		230	裁剪图像180x1  c_crop,h_180
quality	q	百分比	控制JPG或者W 值

指令名	指 令	Value	示 例
crop	c	<i>mode</i>	裁剪方式,  图片质量为100 c_thumb,g_f;
		100	
		10	 图片质量为10% c_thumb,g_f;
radius	r	像素值或 者'max'	指定半径, 生 成模糊效果  生成30像素半径 c_thumb,g_f;
		30	
		max	 使用最大半径生 成模糊效果 c_thumb,g_f;
angle	a	角度或者翻转 模式	
		90	 顺时针旋转90度 c_fill,h_80
		10	 顺时针旋转10度 c_fill,h_80
		-20	 逆时针旋转20度 c_fill,h_80
		vflip	 垂直翻转 c_fill,h_80
		hflip	 水平翻转 c_fill,h_80
effect	e	<i>name and value</i>	
		grayscale	 灰度 c_fill,h_380
		oil_paint	 油画效果 c_fill,h_380

指令名	指 令	Value	示 例
crop	c	mode	裁剪方式,
		<i>oil_paint:2</i>	使用油画效果, 围1到8, 默认值  c_fill,h_38(
		negate	反色  c_fill,h_38(
		<i>brightness:28</i>	调整图片的亮度 值范围-100到1  c_fill,h_38(
		<i>brightness:-28</i>	调整图片的亮度 取值范围-100至  c_fill,h_38(
		blur	模糊效果  c_fill,h_38(
		<i>blur:300</i>	使用模糊效果, 范围1到2000,  c_fill,h_38(
		pixelate	像素化  c_fill,h_38(
		<i>pixelate:40</i>	使用像素化效果 值为5  c_fill,h_38(
		sharpen	锐化  c_fill,h_38(
		<i>sharpen:400</i>	使用锐化效果, 范围1到2000,  c_fill,h_38(
		auto_contrast	自动对比度  c_fill,h_38(

指令名	指 令	Value	示 例
crop	c	<i>mode</i>	裁剪方式,
		improve	 自动调整图像色 c_fill,h_380
		sepia	 增加褐色, 实现 c_fill,h_380
		<i>sepia:60</i>	 增加褐色, 实现 为60。取值范围 c_fill,h_380
		<i>red:40</i>	 增加红色 c_fill,h_380
		<i>green:40</i>	 增加绿色 c_fill,h_380
		<i>blue:40</i>	 增加蓝色 c_fill,h_380
		<i>yellow:40</i>	 增加黄色 c_fill,h_380
		<i>cyan:40</i>	 增加青色 c_fill,h_380
		<i>magenta:40</i>	 增加粉色 c_fill,h_380
opacity	o	百分比	控制PNG或者 JPG的透明度
		25	 不透明度为25% h_330,w_330
border	bo	style	
		<i>10_0000004a</i>	 设置一个边框宽 度为4a (16进制) h_330,w_330

指令名	指 令	Value	示 例
crop	c	<i>mode</i>	裁剪方式, 对圆角图像设置 bbbbbb h_330,w_330
background	b	<i>color</i>	设置背景颜色为 设置背景颜色为 ffffdef0 设置背景颜色为 dbeced
overlay	l	水印图片名, 或者字体描述 文件名称等	在原图上增加 重力参数控制: 参
		<i>superman</i>	在图片的右下角 部区域) ; 首先 (式) 保存到您的 imgx/overlay/m l_my_name指 c_fill.w 50(l_superman.q w 250.x -120 l_scs_logo,x
		<i>text:font_me:</i> 你好，白山云	文字水印。关 绍。 l_text:font_ g_north_east
format	f	图片格式	
		<i>png</i>	f_png
		<i>webp</i>	f_webp
		<i>jpeg</i>	f_jpeg
		<i>jpg</i>	f_jpg
version	v	版本	缓存未过期的

指令名	指 令	Value	示 例	
crop	c	<i>mode</i>		裁剪方式,
		1.21		小数 v_1.21
		13		整数 v_13
transformation	t	名称		"指令集", 指令式的文
		<i>my_thumbs</i>		自定义名称, 在 <i>imgx/cmd_temp</i> t_my_thumbs
information	i	信息类型		按照#
		<i>exif</i>	<i>exif</i>	i_exif, 返回json
		<i>iptc</i>	<i>iptc</i>	i_iptc, 返回json
		<i>all</i>	<i>all</i>	i_all, 返回json#

使用AWS-SDK访问图片处理服务

可以使用标准的AWS-SDK直接访问图片处理服务，注意imgx服务只接受标准的 `get_object` 操作，其他操作都是非法的。

下面的示例代码使用python boto3 sdk：

```

import boto3

s2_imgx_domain_name = 'http://imgx-ss.bscstorage.com'    # ;
s2_imgx_access_key = 'xxxxxxxx'                         # ;
s2_imgx_secret_key = 'xxxxxxxx'                         # ;

imgx_cmd = 'c_scale,w_100,h_100' # 图片处理命令, 参考 图片处理插
imgx_bucket = 'my_imgx_bucket'   # 用户在存储上的bucket名称
imgx_key = 'my_img.jpg'         # 用户想要操作的图片的key名称

# 创建s2 client
imgx_cli = boto3.client('s3',
                        endpoint_url=s2_imgx_domain_name,
                        aws_access_key_id=s2_imgx_access_key,
                        aws_secret_access_key=s2_imgx_secret_key)

imgx_cmd_key = u'{cmd}/{key}'.format(cmd=imgx_cmd, key=imgx_key)
resp = imgx_cli.get_object(Bucket=imgx_bucket, Key=imgx_cmd_key)

print resp

```

水印功能详细介绍

1. 水印分为： 图片水印 和 文字水印 ；
2. 如果您使用了水印 `l` 指令，则和 `l` 一组的其他指令将针对 水印
图片或者文字 进行处理
(如：`w`、`h`、`g`、`x`、`y`、`o`、`bo`、`e`、`a`、`r` 等)，后面举例介绍。

图片水印

您需要预先将水印贴图保存到对应的bucket下

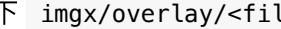
`imgx/overlay/<filename>.png`，图片必须是png格式，下面两张图为
例：

水印贴图	对应路径	对应指令
	<code>imgx/overlay/icon_v.png</code>	<code>l_icon_v</code>
	<code>imgx/overlay/ws_logo.png</code>	<code>l_ws_logo</code>

下面举例介绍具体功能

示例	描述	指令
	<p>1. 将原图等比放缩； 2. 添加一个图片水印到原图的左上角，并且微调坐标(x_43,y_20)，设置水印的宽度为120px，不透明度为35%，逆时针旋转水印10度</p>	<pre>c fit,w 300,f png--l bs logo,q north west, w_120,o_35,x_43,y_20,a_-10</pre>
	<p>1. 将原图处理成一个圆角头像； 2. 在右下角添加一个水印图片，并且向外微调水印坐标(x_-1,y_-5)，设置水印图片的宽度为60px</p>	<pre>c thumb,q face,w 200,h 200, r max,bo 6 ffffff80,f png--l icon_v, g_south_east,w_60,x_-1,y_-5</pre>
	<p>1. 同上 2. 将水印图片反色</p>	<pre>c thumb,q face,w 200,h 200, r max,bo 6 ffffff80,f png--l icon_v, g_south_east,w_60,x_-1,y_-5,e_navigate</pre>

文字水印

您需要预先将文字水印的字体配置(json格式的文件)保存到对应的bucket下，例如：

```
{  
    "font_family": "Xingkai SC",  
    "font_style": "bold",  
    "font_size": 30,  
    "font_color": "000000",  
    "background": "ff0000cc",  
    "padding": 10,  
    "word_spacing": 1,  
    "kerning": 1,  
    "line_spacing": 2,  
    "pierced": false,  
    "tile": false,  
    "text": "默认值",  
}
```

对应路径: imgx/overlay/my_font.json

字体参数介绍

所有参数非必填项

参数名	介绍	默认值
font_family	字体家族，我们支持的所有 字体	Songti SC
font_style	字体样式（如果字体本身支持以下样式才有效果，否则默认使用normal）： normal（常规） <i>italic</i> （斜体） bold （粗体） light （细体）	normal
font_size	字号（单位：px）	14
font_color	16进制rgba（前6为rgb，最后2位alpha，都是0到f），省略最后两位则认为不透明	黑色 (000000)
background	16进制rgba（前6为rgb，最后2位alpha，都是0到f），省略最后两位则认为不透明	透明 (ffffff00)
padding	文字周围填充的宽度（单位：px）	6
word_spacing	词间距（单位：px）	0
kerning	字间距（单位：px）	0
line_spacing	行间距（单位：px）	0
pierced	镂空字效果（bool值）	false
tile	是否平铺（bool值）	false
text	默认字符串	空字符串

示例1：

imgx/overlay/simple_font.json :

```
{
    "font_family" : "Microsoft YaHei",
    "font_size" : 40,
    "font_color" : "fffffff"
}
```

一个最简单的文字水印 

指令：

```
w_800,f_png--l_text:simple_font:Hello+Bai+Shan!!,x_20,y_20,
```

示例2：

```
imgx/overlay/subtitles.json :
```

```
{  
    "font_size" : 22,  
    "font_color" : "ffffff"  
}
```

```
imgx/overlay/subtitles_s.json :
```

```
{  
    "font_size" : 16,  
    "font_color" : "ffffff"  
}
```

咱们也搞一个大片效果, 应用到两张图上: 

指令:

```
f_png,c_fill,w_800,h_400,e_brightness:-8--c_pad,w_800,h_550
```

示例3：

```
imgx/overlay/my_font.json :
```

```
{  
    "font_family" : "Xingkai SC",  
    "font_style" : "bold",  
    "font_size" : 40,  
    "font_color" : "000000",  
    "background" : "ff0000cc",  
    "padding" : 18  
}
```

咱们做一个对联: 

指令:

```
w_800,f_png--l_text:my_font:马驰大道征途远,g_south_west,w_40,
```

示例4：

imgx/overlay/font_me.json :

```
{  
    "font_family": "Microsoft YaHei",  
    "font_size": 30,  
    "font_color": "fffffff",  
    "font_style": "bold",  
    "background": "0000008f",  
    "pierced": true,  
    "tile": false,  
    "padding": 12,  
    "word_spacing": 2.2,  
    "line_spacing": 1.2,  
    "kerning": 1.5  
}
```

镂空字效果 

指令：

```
l_text:font_me:你好, 白山云,g_north_west,x_20,y_20--w_800
```

示例5：

imgx/overlay/tile.json :

```
{  
    "font_family" : "Microsoft YaHei",  
    "font_style" : "bold",  
    "font_size" : 40,  
    "font_color" : "000000",  
    "background" : "ff000066",  
    "padding" : 18,  
    "tile": true  
}
```

平铺效果 

指令：

```
w_800,f_png--l_text:title:Hello BaiShan!!,g_south
```

示例6：

```
imgx/overlay/badge.json :
```

```
{  
    "font_style" : "bold",  
    "font_size" : 30,  
    "font_color" : "fffffff",  
    "background" : "ff0000cc",  
    "padding" : 15  
}
```

增加一个badge(数字徽章)

指令：

```
c_thumb,g_face,w_200,h_200,r_max,bo_6_ffffff80,f_png--l_te>
```

使用“指令集”

如果您不希望把指令暴露在URL中，很简单：

下面举例说明，自动识别照片上的人脸位置并做一张带有边框的圆形缩略图：

- 先创建一个json文件，保存到对应的bucket中：路径：

```
imgx/cmd_template/avatar.json :
```

```
[  
    {  
        "crop" : "thumb",  
        "gravity" : "face:center",  
        "width" : 200,  
        "height" : 200,  
        "radius" : "max",  
        "border" : "6_ffffff80",  
        "format" : "png"  
    },  
    {  
        "overlay" : "icon_v",  
        "gravity" : "south_east",  
        "width" : 60,  
        "x" : -1,  
        "y" : -5  
    }  
]
```

- 通过下面的方式访问：

```
http://imgx-ss.bscstorage.com/imgx-test/t_avatar/demo/1.jpg  
http://imgx-ss.bscstorage.com/imgx-test/t_avatar/demo/3.jpg
```

- 效果: 

字体 (font_family)

查询字体的API: <http://imgx-ss.bscstorage.com/fonts>

如何使用视音频转码服务

功能列表

支持的输入格式

- 容器格式
3GP、AVI、FLV、MP4、M3U8、MPG、WMV、MKV、MOV、TS、DIV、GIF、AMR、MP3
- 视频编码格式
H.264、H.265、MPEG-1、MPEG-2、MPEG-4、VP8、VP9、Windows Media Video等
- 音频编码格式
AAC、AC-3、MP2、MP3、SPEEX、Windows Media Audio

支持的输出格式

- 容器格式
FLV、MP4、TS、M3U8、MPG、TS、GIF、MP3、MP2、AAC、WMV、WMA、AAC、DASH
- 视频编码格式
H.264、H.265、GIF、MPEG-2、MS-MPEG4
- 音频编码格式
AAC、MP2、MP3、Windows Media Video
- 编码格式兼容关系

编码格式	支持的容器
H.264	FLV, MP4, TS
H.265	FLV, MP4, TS
GIF	GIF
MPEG2	MPG
AAC	FLV, MP4, TS, AAC
MP2	MPG, MP2
MP3	MP3, FLV, MP4, TS
MP3	MP3, FLV, MP4, TS
MS-MPEG4	WMV
Windows Media Audio	WMV, WMA

转码方式

- 自动转码：

通过提前配置转码模板和转码规则，对上传至某 Bucket 中的视频文件进行自动转码

- 主动转码：

用户通过创建转码任务（需提前创建转码模板和管道）主动对单个文件进行转码

转码功能

- 视频截图：对视频文件某一个时间点或者按照时间间隔截取 JPG/PNG 图片。
- 视频剪辑：截取视频中的一段时间进行转码。（正在开发中）
- 转分辨率
- 转视频码率
- 转音频码率
- 转音频采样率
- 转视频帧率
- 设置GOP（关键帧固定间距）
- HLS转码
- DASH转码
- FastStart：通过将 MP4 文件的原信息从文件尾部移到头部，实现播放快速启动。

名词解释

转码模板 (Preset)

转码模板用于预定义可以复用的转码规则。

任务 (Job)

任务负责按照预先创建的模板对已经存储在Bucket中的多媒体文件进行转码。一个转码任务可以将一个输出转码成多种输出格式。为了提高并行效率，建议为每个输出创建一个任务。

管道 (Pipeline)

管道是用于管理任务的队列。当创建任务时，您需要选择一个管道用于存放将要创建的任务。您可以创建多个管道，用于执行不同任务。比如您可以创建两个管道，一个用于处理普通任务，另一个只负责处理紧急任务。每个管道中的任务可以并发执行，但并发数量有限（目前为5个任务）。转码系统按照每个管道中任务创建的顺序来依次执行。

如何使用转码系统

创建转码模板

在存储控制台的[视频处理->转码模板](#)模块中，通过 UI 来创建转码模板。

创建管道

使用[创建管道接口](#)创建管道。

创建任务

使用[创建任务接口](#)创建管道。

Transcoder 请求签名方式

Transcoder API 中的所有请求均需要签名验证。 Transcoder 服务目前支持 AWS V4 签名方式。包括下列四个步骤：

1. 创建 Canonical Request
2. 创建 String to Sign (请注意使用 `elastictranscoder` 服务名)
3. 使用 HMAC 方法生成签名。

具体 V4 签名步骤请参考：[签名算法](#)

创建管道

描述

发送 POST 请求到 `/2012-09-25/pipelines/` 以创建管道。

请求

```
POST /2012-09-25/pipelines HTTP/1.1
Content-Type: application/json; charset=UTF-8
Accept: */*
Host: transcoder-ss.bscstorage.com
x-amz-date: 20170726T174952Z
Authorization: AWS4-HMAC-SHA256
    Credential=AccessKeyId/request-date/Transco
    SignedHeaders=host;x-amz-date;x-amz-target,
    Signature=calculated-signature
Content-Length: number of characters in the JSON string
{
    "Name": "管道名称",
    "InputBucket": "用于存放转码源文件的 Bucket 名称",
    "OutputBucket": "用于存放输出文件的 Bucket 名称",
    "ContentConfig": {
        "Permissions": [
            {
                "GranteeType": "Canonical|Email|Group",
                "Grantee": "用户名(Canonical)|用户邮箱>Email)|AllUsers|AuthenticatedUsers|LogDeliveryUser",
                "Access": [
                    "Read|ReadAcp|WriteAcp|FullControl",
                    ...
                ]
            },
            {...}
        ],
    },
    "SuccessCallbackUrl": "接受任务成功回调的 URL",
    "FailureCallbackUrl": "接受任务失败回调的 URL",
}
}
```

请求参数

- Name

管道名称。为了便于您的管理，我们推荐对不同的管道设置不同的名称，但并不强迫。管道名称最长40个字符。

- InputBucket

用于存放转码源文件的 Bucket 名称。请确认输入的 Bucket 存在于您的账号中。

- OutputBucket

用于存放转码后的输出文件的 Bucket 名称。请确认输入的 Bucket 存在于您的账号中。

- (可选的) ContentConfig: Permissions

定义了哪些用户或者预定义的用户组可以访问或修改转码后的输入文件。如果您设置了 Permissions，那么系统将对转码后的文件仅授予您指定的权限，并不会对所有者授予完全控制权限。如果您忽略了此项，那么系统将对所有者授予完全控制权限。

- (可选的) SuccessCallbackUrl

接受任务成功回调的 URL。仅支持 HTTP 协议的 URL。转码任务成功后系统会向该 URL 发送一个 POST 请求，请求 JSON 内容如下：

```
{  
    "result": "success",  
    "input_bucket": "in_bucket_name",  
    "input_key": "test.mp4",  
    "output_bucket": "out_bucket_name",  
    "output_keys": ["test_300K.mp4",  
  
        # HLS 转码相关  
        "output_playlists": ["test_300K.m3u8"],  
  
        # 转码后输出文件元信息  
        "metadata": {  
            "test_300K.mp4": {  
                'etag': "xxxx",  
                'video_streams': [  
                    {  
                        "index": 0,  
                        "codec_name": "h264",  
                        "width": 560,  
                        "height": 320,  
                        "duration_ts": 498000,  
                        "duration": "5.533333",  
                        "bit_rate": "300000",  
                        ...  
                    }, {}, ...  
                ],  
                'audio_stream': [  
                    {  
                        "index": 1,  
                        "codec_name": "aac",  
                        "sample_fmt": "fltp",  
                        "sample_rate": "48000",  
                        "channels": 1,  
                        "channel_layout": "mono",  
                        "bits_per_sample": 0,  
                        "bit_rate": "83050",  
                        ...  
                    }  
                ],  
                'format': {  
                    "nb_streams": 2,  
                    "nb_programs": 0,  
                    "format_name": "mov,mp4,m4a,3gp,3g2,mj2",  
                    "format_long_name": "QuickTime / MOV",  
                    "start_time": "0.000000",  
                    "duration": "5.568000",  
                    "size": "383631",  
                    "bit_rate": "323000",  
                }  
            }  
        }  
    }  
}
```

```
        },
        },
        "test_300K.m3u8": { ... }
    }
}
```

- (可选的) FailureCallbackUrl

接受任务成功回调的 URL。仅支持 HTTP 协议的 URL。转码任务失败后系统会向该 URL 发送一个 POST 请求，请求 JSON 内容如下：

```
{
    "result": "failure",
    "input_bucket": "in_bucket_name",
    "input_key": "test.mp4",
    "error_type": "InternalError",
    "error_message": "...",
}
```

[返回](#)

```
Status: 201 Created
Content-Type: application/json
Content-Length: number of characters in the response
Date: Mon, 14 Jun 2017 06:01:47 GMT

{
    "Pipeline": {
        "Id": "123456789012345678",
        "Name": " my_pipeline",
        "InputBucket": "input_bucket",
        "OutputBucket": "output_bucket",
        "ContentConfig": {
            "Permissions": [
                {
                    "GranteeType": "Group",
                    "Grantee": "AllUsers",
                    "Access": ["Read"]
                }
            ],
            "SuccessCallbackUrl": "http://mydomain.com/cb",
            "FailureCallbackUrl": "http://mydomain.com/cb",
            "Status": "Active|Paused",
        }
    }
}
```

查看管道

描述

发送 GET 请求到 `/2012-09-25/pipelines/pipelineId` 以获取指定管道。

请求

```
GET /2012-09-25/pipelines/pipelineId HTTP/1.1
Content-Type: charset=UTF-8
Accept: */*
Host: transcoder-ss.bscstorage.com
x-amz-date: 20170726T174952Z
Authorization: AWS4-HMAC-SHA256
    Credential=AccessKeyID/request-date/Transco...
    SignedHeaders=host;x-amz-date;x-amz-target,
    Signature=calculated-signature
```

返回

```
Status: 200 OK
Content-Type: application/json
Content-Length: number of characters in the response
Date: Mon, 14 Jun 2017 06:01:47 GMT

{
    "Pipeline": {
        "Id": "123456789012345678",
        "Name": " my_pipeline",
        "InputBucket": "input_bucket",
        "OutputBucket": "output_bucket",
        "ContentConfig": {
            "Permissions": [
                {
                    "GranteeType": "Group",
                    "Grantee": "AllUsers",
                    "Access": ["Read"]
                }
            ],
            "SuccessCallbackUrl": "http://mydomain.com/cb",
            "FailureCallbackUrl": "http://mydomain.com/cb",
            "Status": "Active|Paused",
        }
    }
}
```

列管道

描述

发送 GET 请求到 `/2012-09-25/pipelines/` 以获取属于您账号的管道。

请求

```
GET /2012-09-25/pipelines?PageToken=value HTTP/1.1
Content-Type: charset=UTF-8
Accept: */*
Host: transcoder-ss.bscstorage.com
x-amz-date: 20170726T174952Z
Authorization: AWS4-HMAC-SHA256
    Credential=AccessKeyId/request-date/Transco
    SignedHeaders=host;x-amz-date;x-amz-target,
    Signature=calculated-signature
```

请求参数

- `PageToken`

当 Transcoder 接口返回超过一页的结果时（返回中出现 `NextPageToken`），请使用 `PageToken` 来获取下一页的结果。

返回

```
Status: 200 OK
Content-Type: application/json
Content-Length: number of characters in the response
Date: Mon, 14 Jun 2017 06:01:47 GMT

{
    "Pipelines": [
        {
            "Id": "123456789012345678",
            "Name": "my_pipeline",
            "InputBucket": "input_bucket",
            "OutputBucket": "output_bucket",
            "SuccessCallbackUrl": "http://mydomain.com/cb",
            "FailureCallbackUrl": "http://mydomain.com/cb",
            "ContentConfig": {...},
            "Status": "Active|Paused",
        },
        { ... }
    ],
    "NextPageToken": "123456789012345679"
}
```

删除管道

描述

发送 DELETE 请求到 `/2012-09-25/pipelines/pipelineId` 以获取指定管道。

请求

```
DELETE /2012-09-25/pipelines/pipelineId HTTP/1.1
Content-Type: charset=UTF-8
Accept: */*
Host: transcoder-ss.bscstorage.com
x-amz-date: 20170726T174952Z
Authorization: AWS4-HMAC-SHA256
    Credential=AccessKeyId/request-date/Transco...
    SignedHeaders=host;x-amz-date;x-amz-target,
    Signature=calculated-signature
```

返回

```
Status: 202 Accepted
Content-Type: application/json
Content-Length: number of characters in the response
Date: Mon, 14 Jun 2017 06:01:47 GMT

{
    "Success": "true"
}
```

更新管道

描述

发送 PUT 请求到 `/2012-09-25/pipelines/pipelineId` 以更新已创建的管道。

如何更新队列的参数

请在请求的 Body 中包含更新后的所有的队列参数（不变的参数也需要包含在其中）。

请求

```
PUT /2012-09-25/pipelines/pipelineId HTTP/1.1
Content-Type: application/json; charset=UTF-8
Accept: */*
Host: transcoder-ss.bscstorage.com
x-amz-date: 20170726T174952Z
Authorization: AWS4-HMAC-SHA256
    Credential=AccessKeyId/request-date/Transco
    SignedHeaders=host;x-amz-date;x-amz-target,
    Signature=calculated-signature
Content-Length: number of characters in the JSON string
{
    "Name": "管道名称",
    "InputBucket": "用于存放转码源文件的 Bucket 名称",
    "OutputBucket": "用于存放输出文件的 Bucket 名称",
    "ContentConfig": {
        "Permissions": [
            {
                "GranteeType": "Canonical|Email|Group",
                "Grantee": "用户名(Canonical)|用户邮箱>Email)|AllUsers|AuthenticatedUsers|LogDeliveryUser",
                "Access": [
                    "Read|ReadAcp|WriteAcp|FullControl",
                    ...
                ]
            },
            {...}
        ],
    },
    "SuccessCallbackUrl": "接受任务成功回调的 URL",
    "FailureCallbackUrl": "接受任务失败回调的 URL",
}
}
```

请求参数

- Name

管道名称。为了便于您的管理，我们推荐对不同的管道设置不同的名称，但并不强迫。管道名称最长40个字符。

- InputBucket

用于存放转码源文件的 Bucket 名称。请确认输入的 Bucket 存在于您的账号中。

- OutputBucket

用于存放转码后的输出文件的 Bucket 名称。请确认输入的 Bucket 存在于您的账号中。

- (可选的) ContentConfig: Permissions

定义了哪些用户或者预定义的用户组可以访问或修改转码后的输入文件。如果您设置了 Permissions，那么系统将对转码后的文件仅授予您指定的权限，并不会对所有者授予完全控制权限。如果您忽略了此项，那么系统将对所有者授予完全控制权限。

- (可选的) SuccessCallbackUrl

接受任务成功回调的 URL。仅支持 HTTP 协议的 URL。转码任务成功后系统会向该 URL 发送一个 POST 请求。

- (可选的) FailureCallbackUrl

接受任务成功回调的 URL。仅支持 HTTP 协议的 URL。转码任务失败后系统会向该 URL 发送一个 POST 请求。

返回

```
Status: 202 Accepted
Content-Type: application/json
Content-Length: number of characters in the response
Date: Mon, 14 Jun 2017 06:01:47 GMT

{
    "Pipeline": {
        "Id": "123456789012345678",
        "Name": "my_pipeline",
        "InputBucket": "input_bucket",
        "OutputBucket": "output_bucket",
        "SuccessCallbackUrl": "http://mydomain.com/cb",
        "FailureCallbackUrl": "http://mydomain.com/cb",
        "ContentConfig": {...},
        "Status": "Active|Paused",
    }
}
```

创建任务

描述

发送 POST 请求到 `/2012-09-25/jobs` 以创建任务。

请求

```
POST /2012-09-25/jobs HTTP/1.1
Content-Type: application/json; charset=UTF-8
Accept: */*
Host: transcoder-ss.bscstorage.com
x-amz-date: 20170726T174952Z
Authorization: AWS4-HMAC-SHA256
    Credential=AccessKeyId/request-date/Transco
    SignedHeaders=host;x-amz-date;x-amz-target,
    Signature=calculated-signature
Content-Length: number of characters in the JSON string
{
    "Inputs": [
        {
            "Key": "转码源文件名"
        }
    ],
    "OutputKeyPrefix": "转码后的输出文件名前缀",
    "Outputs": [
        {
            "Key": "转码后的输出文件名",
            "PresetId": "转码模板 ID",
            "SegmentDuration": "[1,60]",
            "Watermarks": [
                {
                    "InputKey": "string",
                }
            ],
            "Encryption": {
                "Mode": "string",
                "Key": "string",
                "KeyMd5": "string",
                "InitializationVector": "string"
            },
            "Composition": [
                {
                    "TimeSpan": {
                        "StartTime": "string",
                        "Duration": "string"
                    }
                }
            ],
            {...},
            {...}
        },
        {
            "S
napshots": [
                {
                    "Key": "转码后的输出文件名",
                    "Format": "jpg|png",
                    "Time": "截图开始时间点",
                    "Interval": "截图间隔时间",
                    "Number": "截图数量",
                }
            ],
            {...}
        },
        {
            "Playlists": [
                {
                    "Format": "HLSv3|HLSv4",
                    "Name": "播放列表文件名",
                    "OutputKeys": [
                }
            ]
        }
    ]
}
```

```

        "包含在输出文件列表中的输出文件名",
        ...
    ],
},
{...}],
"PipelineId":"本转码任务使用的管道 ID"
}

```

请求参数

- Inputs

转码源文件的信息。接受类型：数组。注意目前只支持单一 input。如：

```

"Inputs": [
    "Key": "path/to/orginal.mp4"
]

```

- Inputs: Key

转码原文件名。

- OutputKeyPrefix

转码后输出文件名前缀。通常用于将转码输出存入单独目录中。比如设置 OutputKeyPrefix 为 transcode_output/，从而将转码后的文件统一存放在 transcode_ouput 目录下。如不需要设置前缀，则使用空字符串。

- (可选) Outputs

转码输出文件信息。Outputs 和 Snapshots 必须至少存在一个。

- Outputs: Key

转码输出文件名。输出的文件将会保存在管道中设置的 OutputBucket 中。注意：

- 如果输出文件名已经存在于存储中，原文件将被本次转码输出文件覆盖。
- 如果模板中的 Container 是 ts，且 SegmentDuration 不为空，则最终输出文件名会在所提供的输出文件名的基础上增加 uuid0000.ts 和 .m3u8。所以如果所提供的输出文件名已经包含后缀 .ts，则最终的结果会是 OutputKeyPrefixKey.tsuuid0000.ts，uuid 表示本次转码的唯一标识，所有的 ts 文件都会携带，类似 471c1c6e620c45debe9e3b3388749017。

- Outputs: PresetId

本输出应用的转码模板 ID。

- (可选、TS 输出 Only) Outputs: SegmentDuration

HLS 切片时长。可选值：1 - 60。单位为秒。注意 SegmentDuration 仅对 Container 为 ts 有效。对于其他 Container，则自动忽略此项。

- (可选) Outputs: Watermarks

视频水印。如果配置了此选项，转码模版中视频编码方式不能选择“不变”，否则无法创建任务。目前仅支持单一水印，例如：

```
"Watermarks": [{}  
    "InputKey": "abc.png",  
]
```

水印位置信息在转码模版中配置，默认是右上角。图片格式仅支持jpg, png。“InputKey”表示在input bucket（配置在管道中）下的图片文件key，该图片文件必须存在。

- (可选, hls加密) Outputs: Encryption

hls切片AES-128加密信息。该选项只有在hls切片时起作用。

```
"Encryption": {  
    "Mode": "AES128",  
    "Key": "string",  
    "KeyMd5": "string",  
    "InitializationVector": "string"  
}
```

"Mode"选项：目前仅支持"AES128"。"Key"选项：json格式的字符串，包括AES128加密视频的秘钥（必须是16个字节）和密钥的访问url。其必须经过base64编码。比如：'{"Url": "<http://ss.bscstorage.com/file.key>", "Key": "1111111122222222"}'，base64编码之后为：'eyJVcmwiOiAiaHR0cDovL3NzLmJzY3N0b3JhZ2UuY29tL2ZpbGUua2V5liwgIktleSI6IClxDTExMTExMTIyMjlyIn0=' "KeyMd5"选项：未经base64编码的key的md5值，其必须经过base64编码。

"InitializationVector"选项：可选配置，初始化向量，长度为16个字节，其必须经过base64编码。

- (可选, 视频裁剪) Outputs: Composition

视频裁剪配置，目前仅支持单一裁剪。

```

"Composition": [
    "TimeSpan": {
        "StartTime": "string",
        "Duration": "string"
    }
]

```

"StartTime"选项：裁剪的开始时间，单位为毫秒。 "Duration"选项：裁剪的时长，单位为毫秒。

- (可选) Snapshots

截图输出信息。Outputs 和 Snapshots 必须至少存在一个。

- Snapshots: Format

截图文件格式。可选值：png, jpg

- Snapshots: Key

输出截图文件名。注意扩展名会自动根据 Format 添加。

- Snapshots: Time

截图开始时间。格式为 s.S (s 为秒，S 为浮点型毫秒)。

- (可选) Snapshots: Interval

截图间隔。格式为 s.S (s 为秒，S 为浮点型毫秒)。只有当 Number > 1 时才会起作用。如果设置截图间隔，则会在输出文件名中自动添加图片索引，比如 snapshot0000.png – snapshot0010.png。

- (可选) Snapshots: Number

截图最大数量。默认为1。

- (可选) Snapshots: Resolution

截图分辨率。格式为 [width]x[height]。比如：1920x1080。默认为原视频分辨率。

- (可选) Snapshots: AspectRatio

截图宽高比。可选值为 ["1:1", "4:3", "3:2", "16:9"]。默认为原视频宽高比。

- (可选) Playlists

自适应 HLS 转码输出信息。

- Playlists: Format

自适应转码的格式。可选值：HLSv3、HLSv4。对于自适应HLS播放列表中的每一个子HLS转码

- HLSv3转码会生成ts切片文件（video00000.ts, video00001.ts, ...）, 以及一个m3u8播放列表。
- HLSv4转码会生成一个ts文件和一个m3u8播放列表。
- Playlists: Name

自适应HLS播放列表的文件名。注意系统会自动添加 .m3u8 扩展名。

Playlists: OutputKeys

包含在自适应 HLS 播放列表中的输出文件名。每个被包含的输出必须是 HLS 输出（Container 为 ts，且提供 SegmentDuration），而且 SegmentDuration 必须相同。

- PipelineId

本转码任务使用的管道 ID

返回

```
Status: 201 Created
Content-Type: application/json
Content-Length: number of characters in the response
Date: Mon, 14 Jan 2013 06:01:47 GMT

{
    "Job": {
        "Id": "任务 ID",
        "Inputs": [
            {
                "Key": "转码源文件名"
            }
        ],
        "OutputKeyPrefix": "转码后的输出文件名前缀",
        "Outputs": [
            {
                "Key": "转码后的输出文件名",
                "PresetId": "转码模板 ID",
                "SegmentDuration": "[1,60]",
            },
            {...}
        ],
        "Snapshots": [
            {
                "Key": "转码后的输出文件名",
                "Format": "jpg|png",
                "Time": "截图开始时间点",
                "Interval": "截图间隔时间",
                "Number": "截图数量",
            },
            {...}
        ],
        "Playlists": [
            {
                "Format": "HLSv3|HLSv4",
                "Name": "播放列表文件名",
                "OutputKeys": [
                    "包含在输出文件列表中的输出文件名",
                    ...
                ],
                ...
            },
            {...}
        ],
        "PipelineId": "本转码任务使用的管道 ID",
        "Status": "Submitted|Progressing|Complete|Canceled|Error",
        "Timing": {
            "SubmitTimeMillis": "任务创建的时间。单位: epoch millisecond",
            "FinishTimeMillis": "任务完成的时间。单位: epoch millisecond"
        }
    }
}
```

例子

```
{
    "Inputs": [
        {
            "Key": "input/test.ts"
        }
    ],
    "OutputKeyPrefix": "",
    "Outputs": [
        {
            "Key": "test.mp4",
            "PresetId": "123",
        }
    ],
    "PipelineId": "10000000000000000001"
}
```

通过SDK创建任务

python

```
import boto3
from botocore.client import Config

config = Config(signature_version='s3v4')
cli = boto3.client('elastictranscoder',
                    config=config,
                    region_name='us-west-2',
                    endpoint_url='http://transcoder-ss.bscst
aws_access_key_id="accesskey",
aws_secret_access_key="secretkey")

inputs = [
    {
        'Key': "sample.mp4",
    }
]

outputs = [
    {
        'Key': "sample.m3u8",
        # 转码模板id, 需要预先配置
        'PresetId': "1000",
        'SegmentDuration': "10",
    }
]

print cli.create_job(
    # 管道id, 请预先创建
    PipelineId='pipelineid',
    Inputs=inputs,
    Outputs=outputs,
)
```

```
java
```

```
public static void create_job() {
    BasicAWSCredentials awsCreds = new BasicAWSCredentials('

        ClientConfiguration clientconfiguration = new ClientConfigu
        clientconfiguration.setSocketTimeout(60 * 60 * 1000); /
        clientconfiguration.setConnectionTimeout(60 * 60 * 1000)

        AmazonElasticTranscoder client = new AmazonElasticTransc
        client.setEndpoint("http://transcoder-ss.bscstorage.com"

        JobInput input = new JobInput().withKey("sample.mp4");

        List<JobInput> inputs = Arrays.asList(input);

        CreateJobOutput out1 = new CreateJobOutput().withKey("o

        List<CreateJobOutput> outputs = Arrays.asList(out1);

        CreateJobRequest createJobRequest = new CreateJobRequest(
            .withInputs(inputs).withOutputs(outputs);

        Job j = client.createJob(createJobRequest).getJob();
        System.out.printf("task id: %s\n", j.getId());
    }
```

获取任务

描述

发送 GET 请求到 `/2012-09-25/jobs/jobId` 来获取详细任务信息。

请求

```
GET /2012-09-25/jobs/jobId HTTP/1.1
Content-Type: charset=UTF-8
Accept: /*
Host: transcoder-ss.bscstorage.com
x-amz-date: 20170726T174952Z
Authorization: AWS4-HMAC-SHA256
    Credential=AccessKeyId/request-date/Transco...
    SignedHeaders=host;x-amz-date;x-amz-target,
    Signature=calculated-signature
```

返回

```
Status: 200 OK
Content-Type: application/json
Content-Length: number of characters in the response
Date: Mon, 14 Jan 2013 06:01:47 GMT

{
    "Job": {
        "Id": "任务 ID",
        "Inputs": [
            {
                "Key": "转码源文件名"
            }
        ],
        "OutputKeyPrefix": "转码后的输出文件名前缀",
        "Outputs": [
            {
                "Key": "转码后的输出文件名",
                "PresetId": "转码模板 ID",
                "SegmentDuration": "[1,60]",
            },
            {...},
            ...
        ],
        "Snapshots": [
            {
                "Key": "转码后的输出文件名",
                "Format": "jpg|png",
                "Time": "截图开始时间点",
                "Interval": "截图间隔时间",
                "Number": "截图数量",
            },
            {...}
        ],
        "Playlists": [
            {
                "Format": "HLSv3|HLSv4",
                "Name": "播放列表文件名",
                "OutputKeys": [
                    "包含在输出文件列表中的输出文件名",
                    ...
                ],
                ...
            },
            {...}
        ],
        "PipelineId": "本转码任务使用的管道 ID",
        "Status": "Submitted|Progressing|Complete|Canceled|Failed",
        "Timing": {
            "SubmitTimeMillis": "任务创建的时间。单位: epoch millis",
            "FinishTimeMillis": "任务完成的时间。单位: epoch millis"
        }
    }
}
```

列出管道中的任务

描述

发送 GET 请求到 `/2012-09-25/jobsByPipeline/pipelineId` 以列出分配在指定管道中的任务。

请求

```
GET /2012-09-25/jobsByPipeline/pipelineId?PageToken=1234567890
Content-Type: charset=UTF-8
Accept: */*
Host: transcoder-ss.bscstorage.com
x-amz-date: 20170726T174952Z
Authorization: AWS4-HMAC-SHA256
    Credential=AccessKeyId/request-date/TranscoderSS/jobsByPipeline/pipelineId/PageToken/1234567890,
    SignedHeaders=host;x-amz-date;x-amz-target,
    Signature=calculated-signature
```

请求参数

- `PageToken`

如果结果中出现 `NextPageToken`, 则说明结果没有列完全。需要在下一次请求中使用上一次结果中的 `NextPageToken` 做为 `PageToken`, 来继续请求结果。

返回

```
Status: 200 OK
Content-Type: application/json
Content-Length: number of characters in the response
Date: Mon, 14 Jan 2013 06:01:47 GMT

{
    "Jobs": [
        {
            "Id": "任务 ID",
            "Inputs": [
                {
                    "Key": "转码源文件名"
                }
            ],
            "OutputKeyPrefix": "转码后的输出文件名前缀",
            "Outputs": [
                {
                    "Key": "转码后的输出文件名",
                    "PresetId": "转码模板 ID",
                    "SegmentDuration": "[1,60]",
                },
                ...
            ],
            "Schemas": [
                {
                    "Key": "转码后的输出文件名",
                    "Format": "jpg|png",
                    "Time": "截图开始时间点",
                    "Interval": "截图间隔时间",
                    "Number": "截图数量",
                },
                ...
            ]
        },
        ...
    ],
    "Playlists": [
        {
            "Format": "HLSv3|HLSv4",
            "Name": "播放列表文件名",
            "OutputKeys": [
                "包含在输出文件列表中的输出文件名",
                ...
            ],
            ...
        },
        ...
    ],
    "PipelineId": "本转码任务使用的管道 ID",
    "Status": "Submitted|Progressing|Complete|Canceled|Error",
    "Timing": {
        "SubmitTimeMillis": "任务创建的时间。单位: epoch milli",
        "FinishTimeMillis": "任务完成的时间。单位: epoch milli"
    }
},
{
    ...
},
"NextPageToken": "1234567890"
}
```

取消任务

描述

发送 DELETE 请求到 /2012-09-25/jobs/jobId 以取消已创建的任务。注意已经开始执行的任务无法取消。

请求

```
DELETE /2012-09-25/jobs/jobId HTTP/1.1
Content-Type: charset=UTF-8
Accept: */*
Host: transcoder-ss.bscstorage.com
x-amz-date: 20170726T174952Z
Authorization: AWS4-HMAC-SHA256
    Credential=AccessKeyId/request-date/Transco...
    SignedHeaders=host;x-amz-date;x-amz-target,
    Signature=calculated-signature
```

返回

```
Status: 202 Accepted
Content-Type: application/json
Content-Length: number of characters in the response
Date: Mon, 14 Jan 2013 06:01:47 GMT

{
    "Success":"true"
}
```

获取音视频元信息

描述

发送 GET 请求到 `/metadata/<bucket>/<key>` 以获取元信息。

请求

```
GET /metadata/test_bucket/test.mp4 HTTP/1.1
Content-Type: charset=UTF-8
Accept: */*
Host: transcoder-ss.bscstorage.com
x-amz-content-sha256: UNSIGNED-PAYLOAD
x-amz-date: 20170925T093131Z
Authorization: AWS4-HMAC-SHA256
    Credential=acc_yanhui/20170925/us-east-1/ela
    SignedHeaders=host;x-amz-content-sha256;x-am
    Signature=e9147d9d1461d7ca92c0805735490be7fc
```

请求参数

- bucket

bucket名字

- key

获取元信息的音视频文件的key

如何使用SDK接口

python

Python

```
import boto3
from botocore.client import Config
import json

cli = boto3.client(
    's3',
    use_ssl=False,
    aws_access_key_id='accesskey',
    aws_secret_access_key='secretkey',
    endpoint_url='http://transcoder-ss.bscstorage.com/meta',
    config=Config(s3={'addressing_style': 'path'})
)

res = cli.get_object(Bucket='your bucket', Key="sample.mp4")
res_json = json.loads(res['Body'].read())
print repr(res_json)
```

java

```
public static void get_meta() {
    BasicAWSCredentials awsCreds = new BasicAWSCredentials("accesskey", "secretkey");
    ClientConfiguration clientconfiguration = new ClientConfiguration();
    clientconfiguration.setSocketTimeout(60 * 60 * 1000);
    clientconfiguration.setConnectionTimeout(60 * 60 * 1000);

    AmazonS3 client = new AmazonS3Client(awsCreds, clientconfiguration);
    client.setEndpoint("http://transcoder-ss.bscstorage.com");

    S3ClientOptions opt = S3ClientOptions.builder().setPathStyle(true).build();
    client.setS3ClientOptions(opt);

    S3Object s3object = client.getObject("your bucket", "sample.mp4");
    byte[] buf = new byte[10240];
    try {
        s3object.getObjectContent().read(buf, 0, 10240);
        System.out.println(new String(buf, "UTF-8"));
    } catch (IOException e) {
        System.out.println("errr");
    }
}
```

返回

```

Status:200

Content-length: '2253'
x-amz-s2-requester: test_bucket
server: openresty/1.11.2.4
connection: keep-alive
x-amz-request-id: 00361611-1709-2517-4651-00163e0630f7
date: Mon, 25 Sep 2017 09:46:51 GMT
content-type: application/json


{
    "audio_streams":      音频信息
    [
        {
            "sample_fmt": "fltp",    采样格式
            "codec_tag": "0x6134706d", 编码器标签
            "codec_type": "audio",   编码器类型
            "channels": 2,          音频数
            "bit_rate": "96001",    码率
            "codec_name": "aac",    编码器名
            "duration": "716.885011", 文件总时长
            "nb_frames": "30876",   帧数
            "codec_time_base": "1/44100", 编码器每帧时
            "index": 1,             流索引号
            "start_pts": 0,          时间戳
            "profile": "LC",         编码的profile
            "tags": 标签信息
            {
                "handler_name": "SoundHandler", 处理
                "language": "und"     语言
            },
            "r_frame_rate": "0/0",   真实基础帧率
            "start_time": "0.000000", 首帧时间
            "time_base": "1/44100", 每帧时长
            "codec_tag_string": "mp4a", 编码器标签名
            "duration_ts": 31614629, 单位为time_base的时
            "codec_long_name": "AAC (Advanced Audio Cod
            "bits_per_sample": 0,    采样码率
            "avg_frame_rate": "0/0", 平均帧率
            "channel_layout": "stereo", 声道
            "max_bit_rate": "96001", 最大码率
            "sample_rate": "44100"   采样率
        }
    ],
    "video_streams":      视频信息
    [
        {

```

```
"profile": "High", 编码的profile
"sample_aspect_ratio": "0:1",
"codec_tag": "0x31637661", 编码器标签
"refs": 1,
"codec_type": "video", 编码器类型
"coded_height": 720, 图像高度
"bit_rate": "1500443", 码率
"codec_name": "h264", 编码器名
"duration": "716.800000", 时长
"is_avc": "true",
"nb_frames": "17920", 帧数
"codec_time_base": "1/50", 编码器每帧时长
"index": 0, 流索引号
"start_pts": 0,
"width": 1280, 帧宽度
"coded_width": 1280, 图像宽度
"pix_fmt": "yuv420p", 像素个数
"chroma_location": "left",
"tags":
{
    "handler_name": "VideoHandler", 处理器名
    "language": "und" 语言
},
"r_frame_rate": "25/1", 真实基础帧率
"start_time": "0.000000", 首帧时间
"time_base": "1/12800", 每帧时长
"codec_tag_string": "avc1", 编码器标签名
"duration_ts": 9175040, 单位为time_base的时长
"codec_long_name": "H.264 / AVC / MPEG-4 AVC/H.264
"display_aspect_ratio": "0:1",
"height": 720, 帧高度
"avg_frame_rate": "25/1", 平均帧率
"level": 40, 级别
"bits_per_raw_sample": "8",
"has_b_frames": 2, 记录缓存帧大小
"nal_length_size": "4"
}
],
"format":
{
    "tags": 标签信息
    {
        "major_brand": "isom",
        "minor_version": "512",
        "compatible_brands": "isomiso2avc1mp41",
        "encoder": "Lavf56.15.102"
    },
}
```

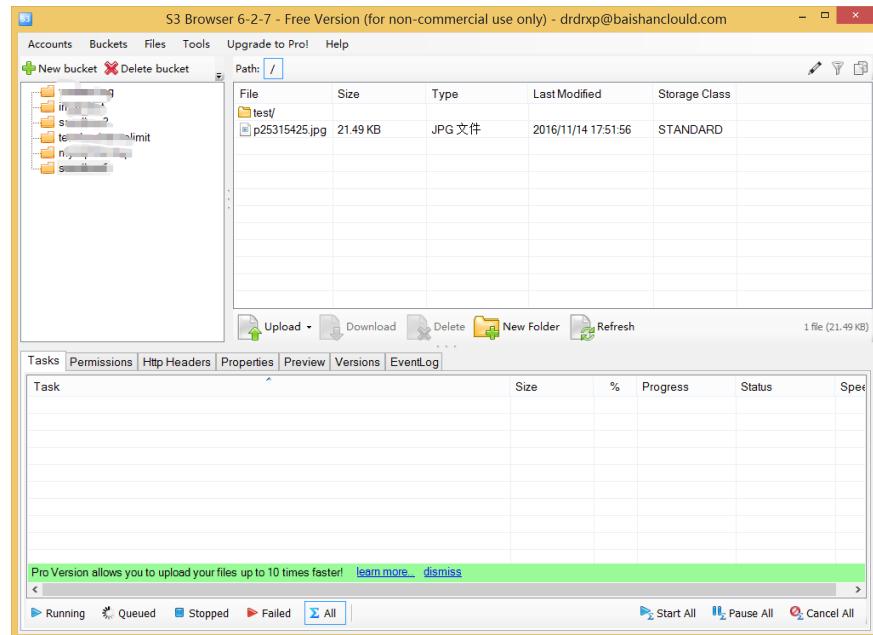
```
        "nb_streams": 2,      流的个数
        "start_time": "0.000000",    首帧时间
        "format_long_name": "QuickTime / MOV",  格式名全
        "format_name": "mov,mp4,m4a,3gp,3g2,mj2",   格式
        "bit_rate": "1607943",   码率
        "nb_programs": 0,
        "duration": "716.932000",   时长
        "size": "144098269"  文件大小
    },
    "other_streams": []
}
```

存储管理工具等

Windows下利用S3 Browser对接白山存储上传文件的步骤

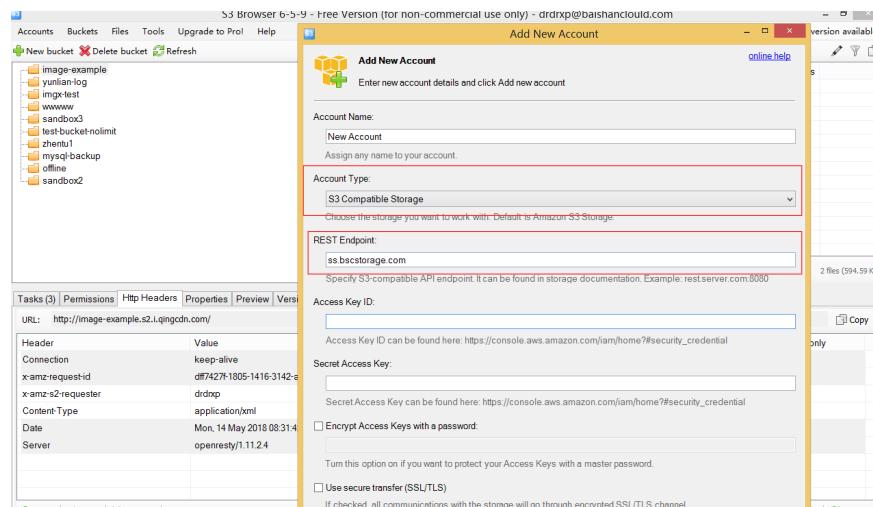
第一步

打开S3 Browser软件，界面如下图所示：



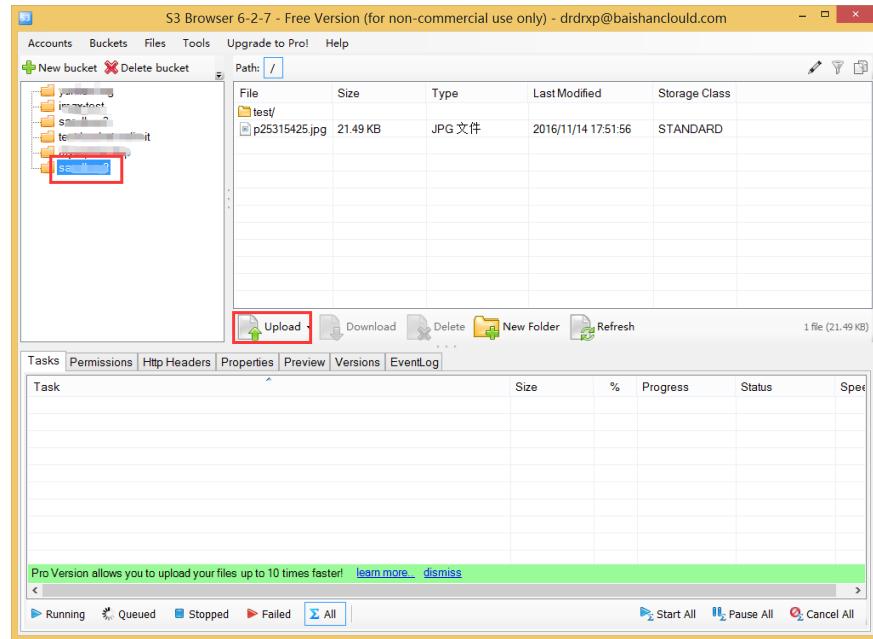
第二步

点击Accounts下的Add New Account，弹出如下界面，注意Storage Type选择S3 Compatible Storage，REST Endpoint填ss.bscstorage.com。输入账号、AccessKey和SecretKey就可以点击添加一个新账户了。



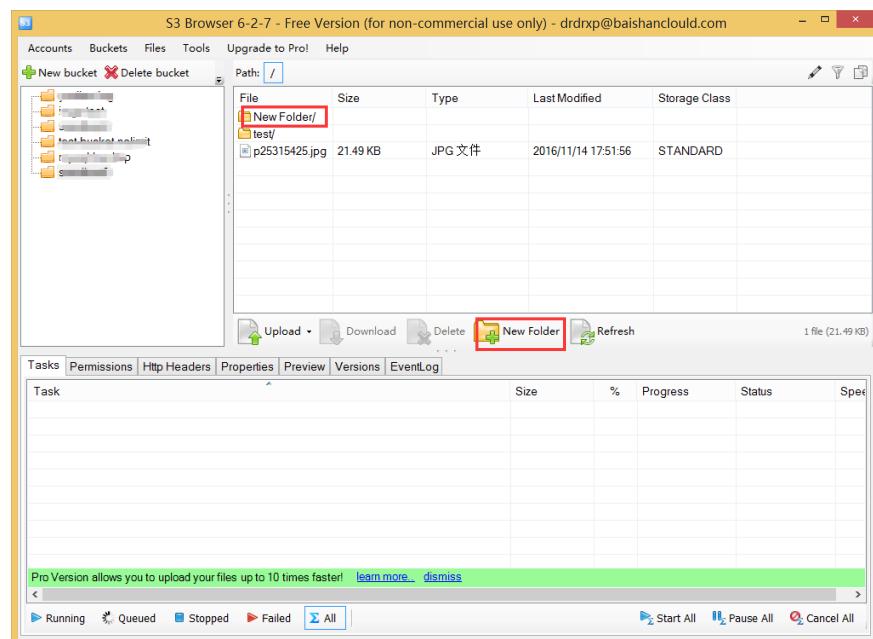
第三步

添加新账户之后就可以看到Storage里面的内容了。选中左边的某个bucket，就可以点击Upload向这个bucket里上传文件。



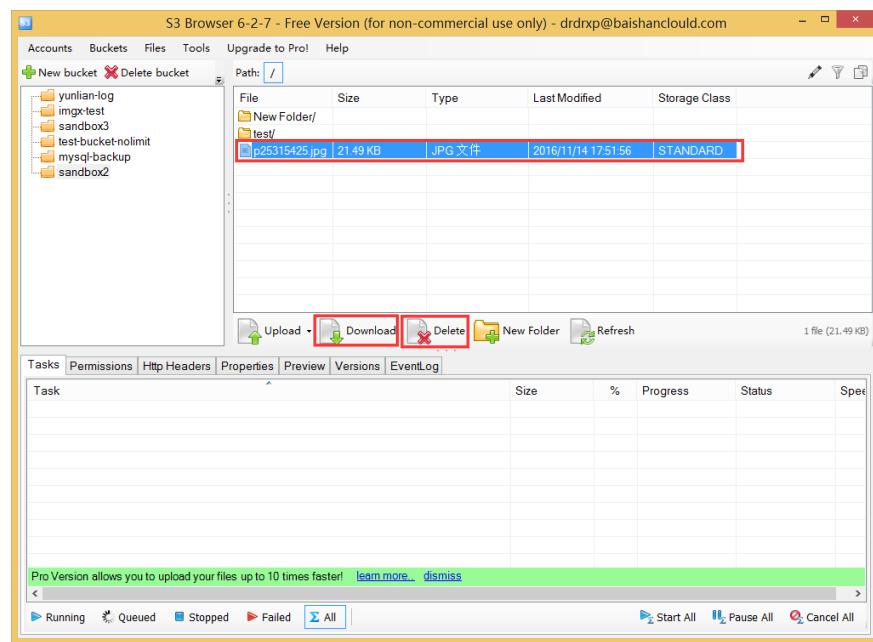
第四步

选中左边的某个bucket，点击New Folder可以向这个bucket里新建文件夹。



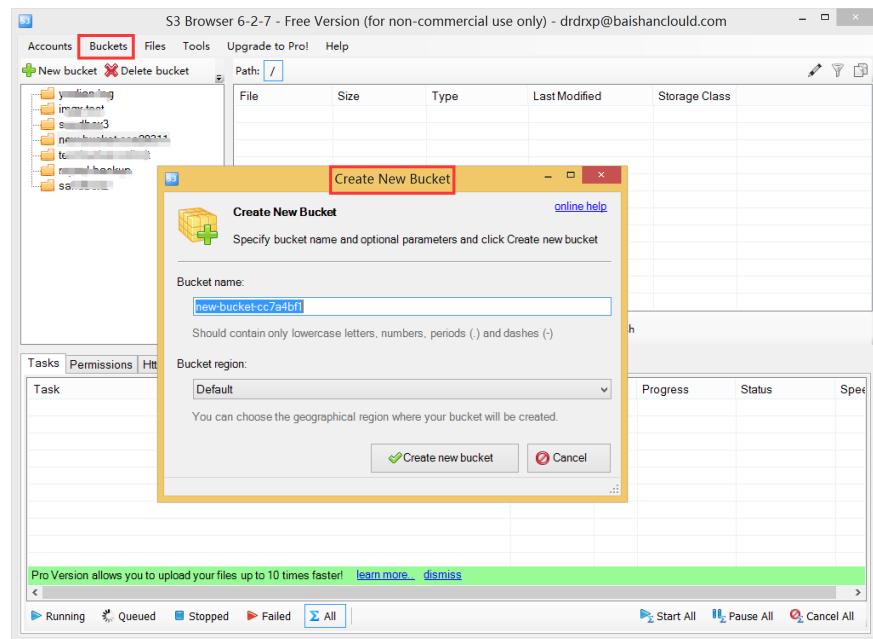
第五步

选中某个文件，点击Download和Delete可以下载和删除文件。



第六步

您还可以点击Buckets菜单下做新增Bucket等一系列操作。至此，您就学完了利用S3 Browser对接白山存储上传文件的所有过程啦。



S3 Browser下载地址: [s3-browser](#)

控制台使用手册

白山云存储控制台操作手册

1. 概述

白山云存储控制台系统是面向客户在线使用云存储的管理平台。该管理平台主要包括以下功能模块：

我的存储：

- 展示用户所有的Bucket。
- 对Bucket进行配置管理。
- 配置图片处理样式。
- Bucket中的对象进行操作。

统计分析：以图表的方式展示各时间段带宽、流量、请求次数、存储空间等数据信息。

视频处理：用户可以创建转码模板、配置自动或主动转码的规则以及查看视频处理的统计分析数据。

秘钥管理：展示用户当前账户的Accesskey和SecretKey。

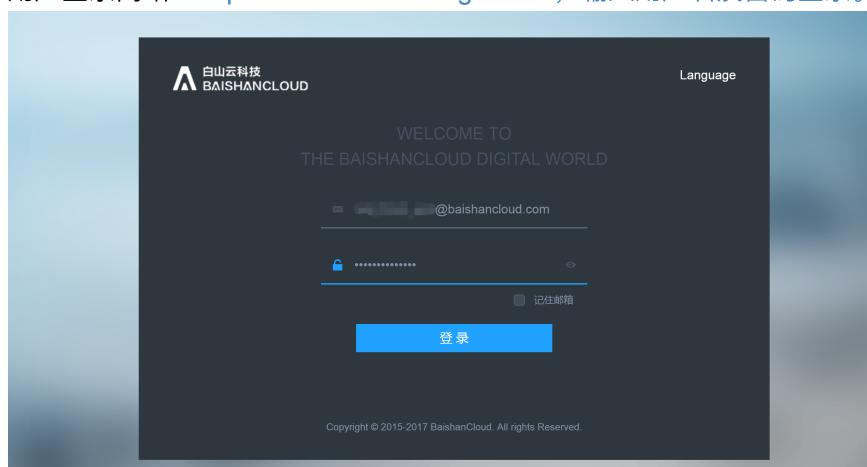
用户管理：超级用户可以创建子账号并对子帐号赋予相应的权限。

帮助文档：帮助用户使用白山云存储系统的文档。

1. 功能特性

1. 用户登录

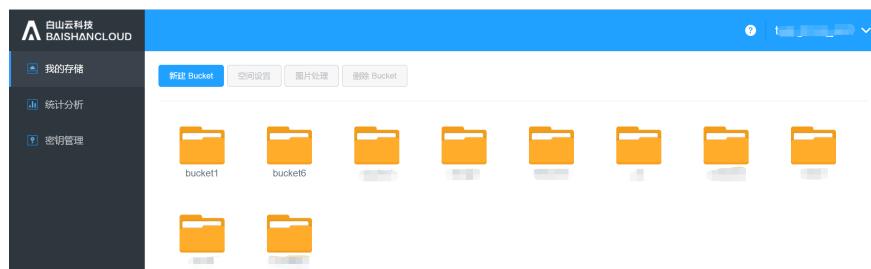
用户登录网站：<http://cwn-ss.bscstorage.com/>，输入用户名及密码登录。



在登录操作界面后，右上角可以选择退出登入。



1. 我的存储



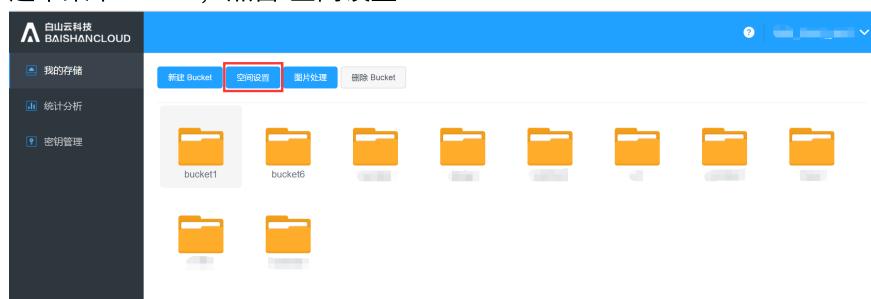
1. 新增Bucket

用户新建Bucket并命名此Bucket。Bucket命名规则:云存储内全局唯一;由小写字母、数字及“-”组成, 长度为3~63位;不能以数字、‘-’开头;不能以‘-’开头或结尾。



1. 空间设置

选中某个Bucket, 点击“空间设置”:



空间设置下面有“权限设置”、“图片鉴黄”等; 这里的权限设置是对存储空

间Bucket进行的权限设置。权限设置可以对某些用户添加权限。

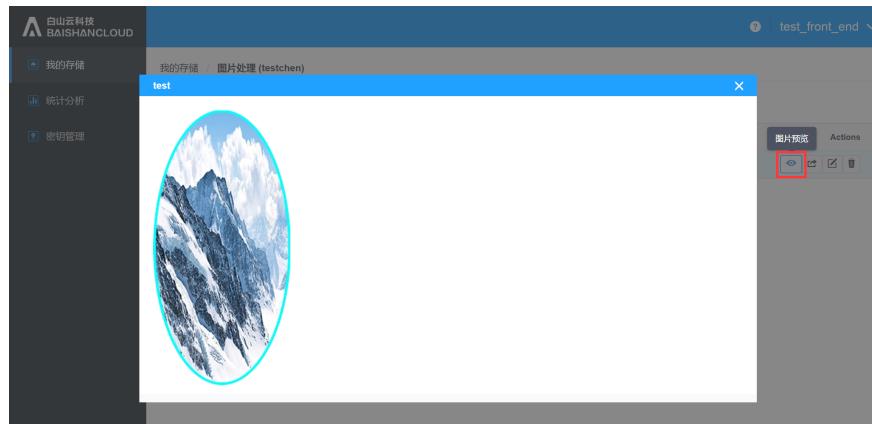
The screenshot shows two views of the BaiShanCloud storage management interface. The top view is titled '空间设置 (bucket1)' and displays '权限控制' (Access Control) settings. It includes sections for '用户组权限' (Group Permissions) and '用户权限' (User Permissions), both of which show '对象访问' (Object Access) and '权限访问' (Permission Access) checkboxes for 'All Users' and 'Auth Users'. The bottom view is also titled '空间设置 (bucket1)' and shows a message 'We've got somethings special for you'.

1. 图片处理

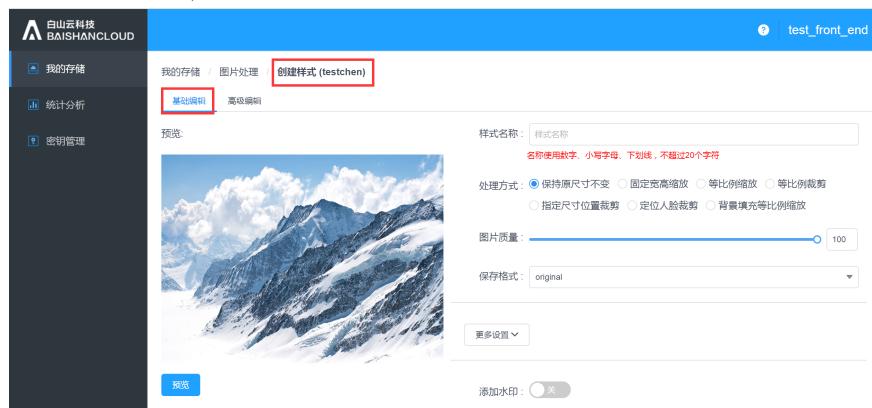
选中某个Bucket，点击“图片处理”进入到图片处理的管理页面。

The screenshot shows two views of the BaiShanCloud storage management interface. The top view is titled '空间设置 (testchen)' and displays '图片处理' (Image Processing) settings. It shows a list of buckets: 'bucket1', 'bucket0', 'myytest', 'design', 'testchen' (which is highlighted with a red box), 'yii', 'wolutest', and 'dasd'. The bottom view is also titled '空间设置 (testchen)' and shows a table of image processing styles:

Name	Styles	Quality	Format	Actions
test	c_scale,w_200,h_400,q_100,f_webp,a_hflip,o_auto,c...	100	webp	



点击“创建样式”，可以创建新的图片处理样式。



更多设置：

更多设置 ^

翻转模式： 旋转角度 垂直翻转 水平翻转

角度：

滤镜与特效：

设置边框： 关

生成圆角： 0

不透明度：

添加水印： 开

水印类型： 文字水印 图片水印

文字内容：
请输入水印文字内容

文字样式： px #ff0000

背景： #ff0000

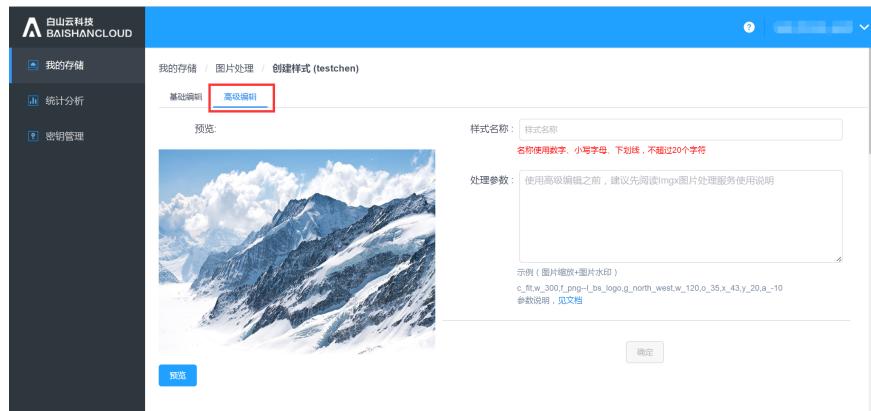
水印位置：

左边距离： px

顶边距离： px

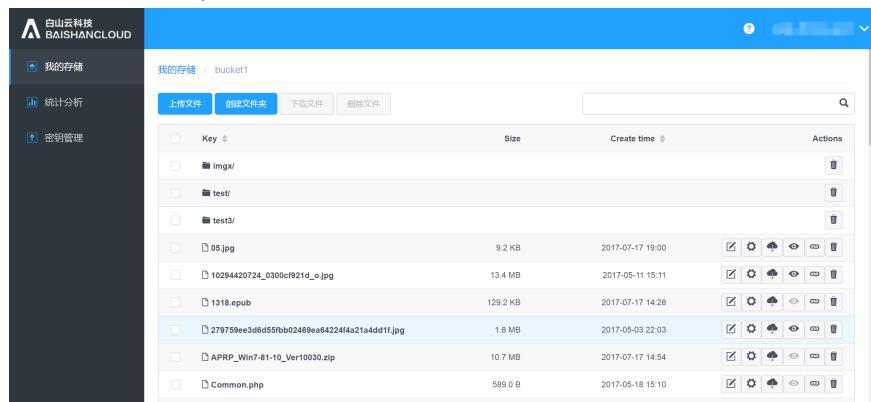
不透明度：

高级编辑：

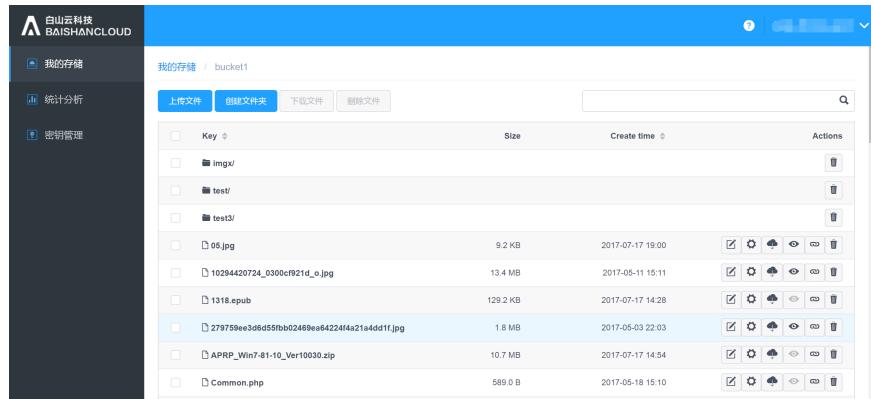


1. 文件管理

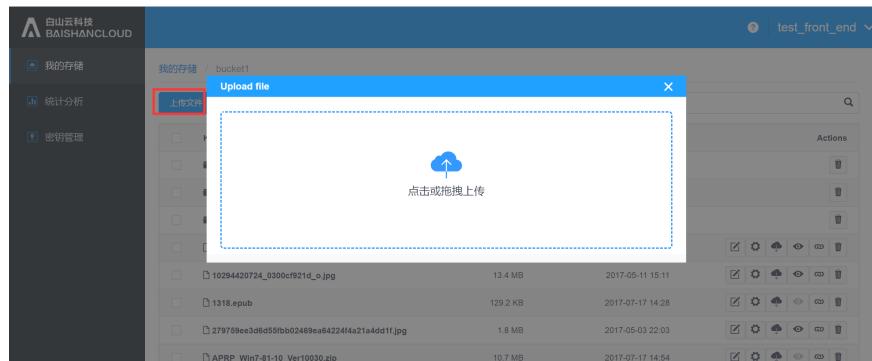
双击某个Bucket，进入到管理某个Bucket中的文件或文件夹。



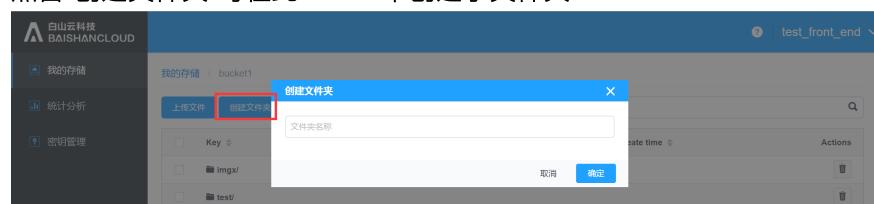
下图为文件管理界面：此界面内，用户可以在此Bucket中创建/删除文件夹、上传/删除文件，并对文件进行权限管理、下载、获取地址、预览文件等操作。



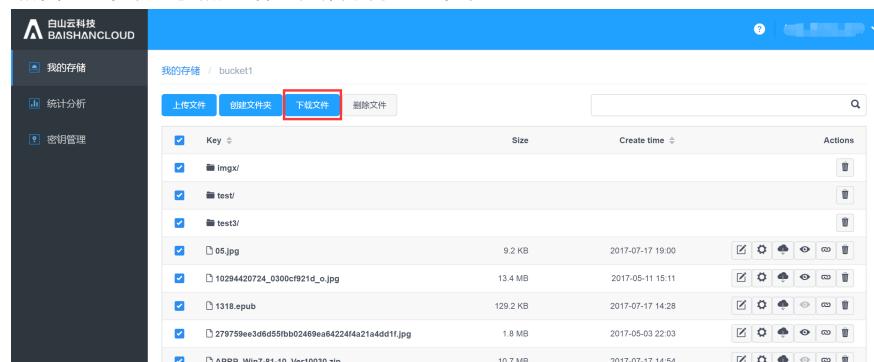
点击“上传文件”可上传本地文件：



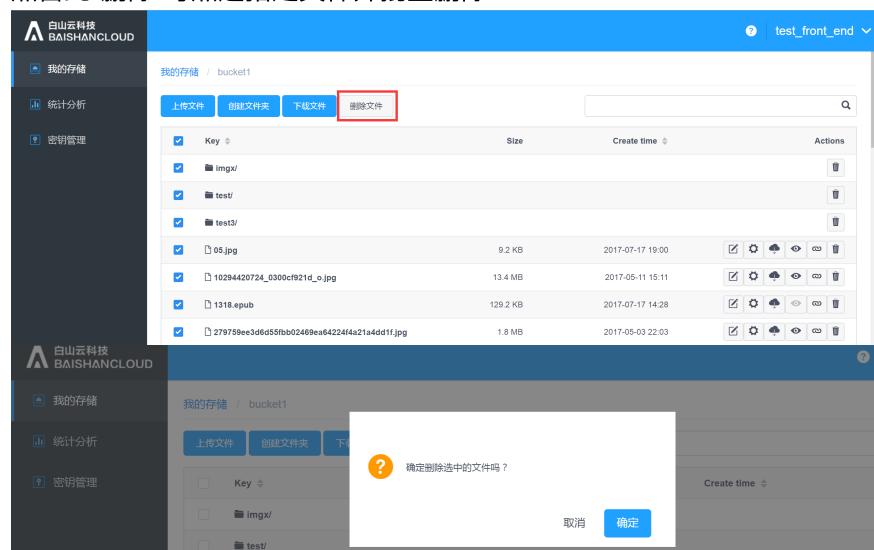
点击“创建文件夹”可在此Bucket中创建子文件夹：



点击此“下载”可点选指定文件并批量下载：



点击此“删除”可点选指定文件并批量删除：



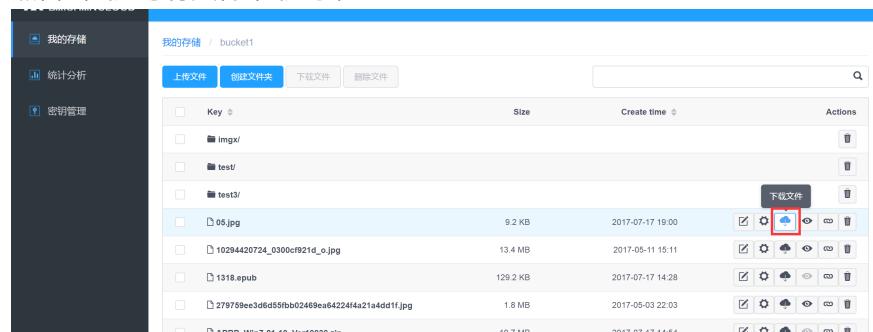
点击“文件重命名”可给文件重命名：



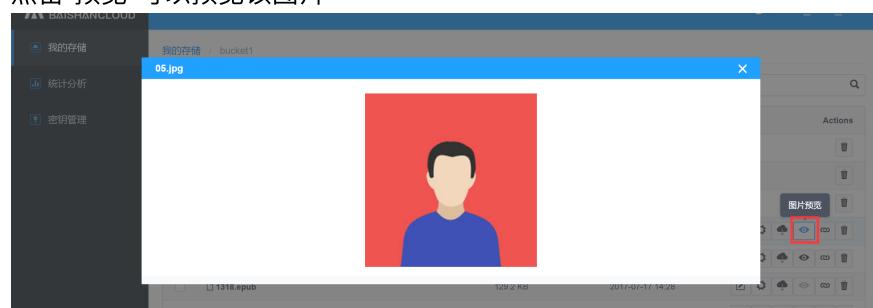
下图为权限设置界面：允许用户对不同的访问者开放不同的访问/编辑权限。



点击“下载”可将文件下载到本地：



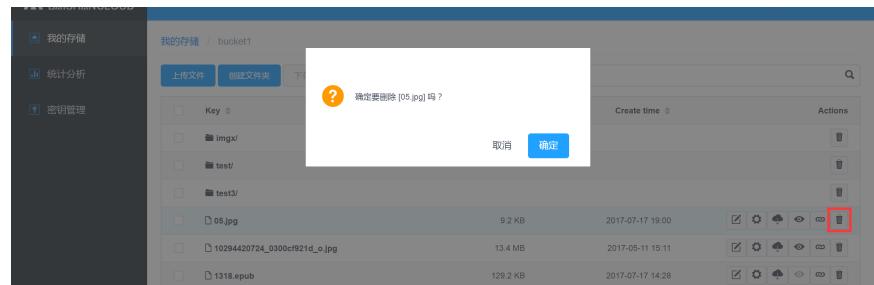
点击“预览”可以预览该图片：



点击“获取地址”，可获取此文件的url地址：

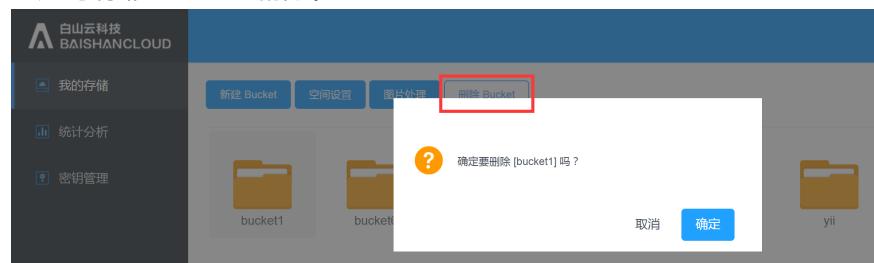


点击“删除”可删除选中文件：



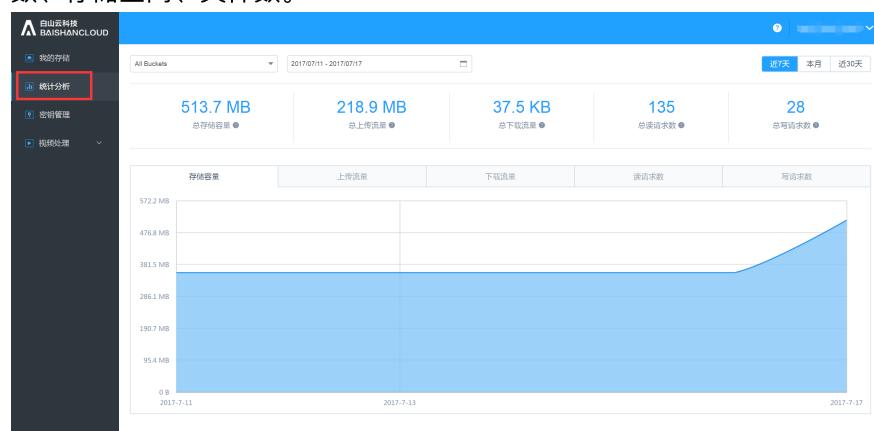
1. 删除Bucket

用户可将非空Bucket删除。

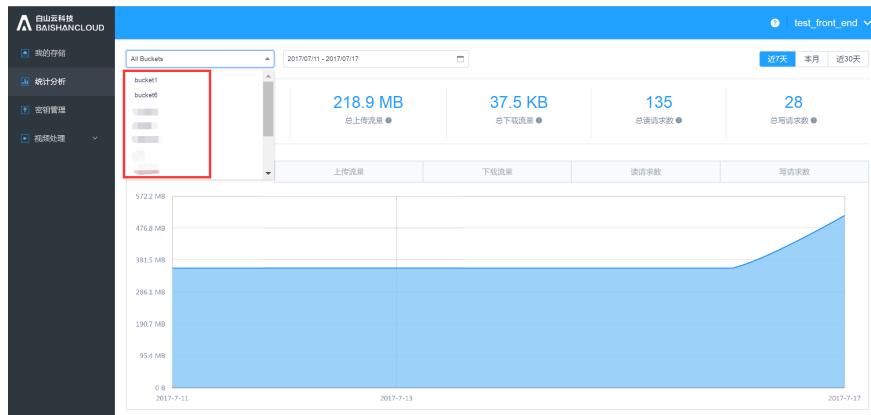


1. 统计分析

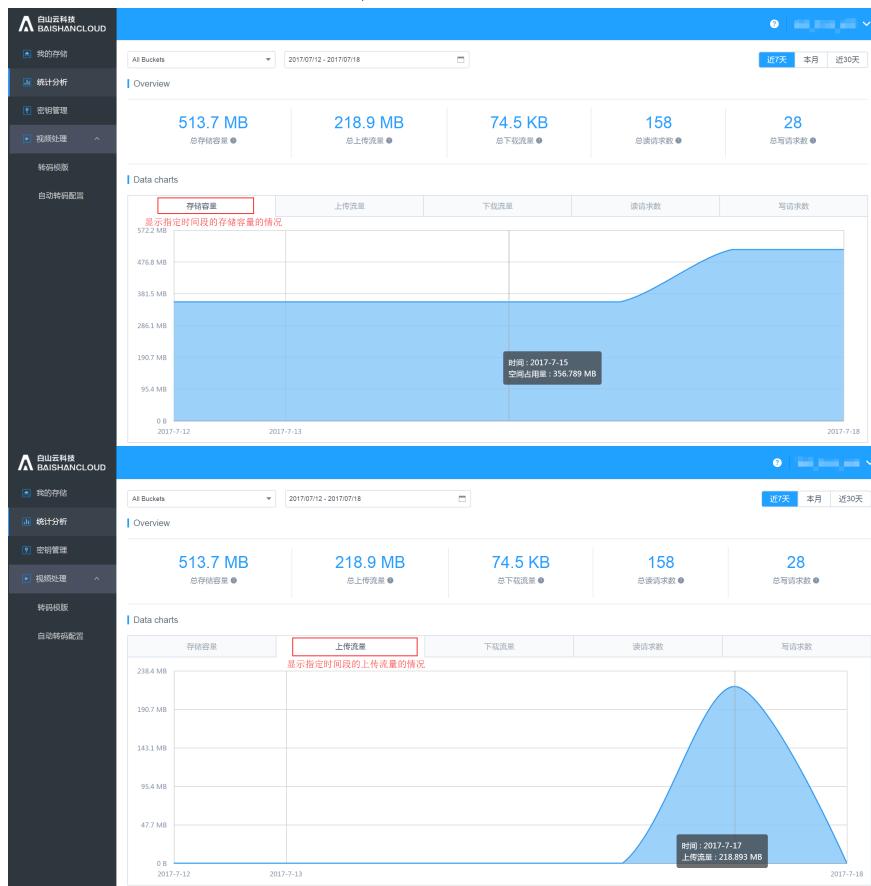
统计分析：显示的是用户各时段的存储情况，包括带宽、流量、请求次数、存储空间、文件数。

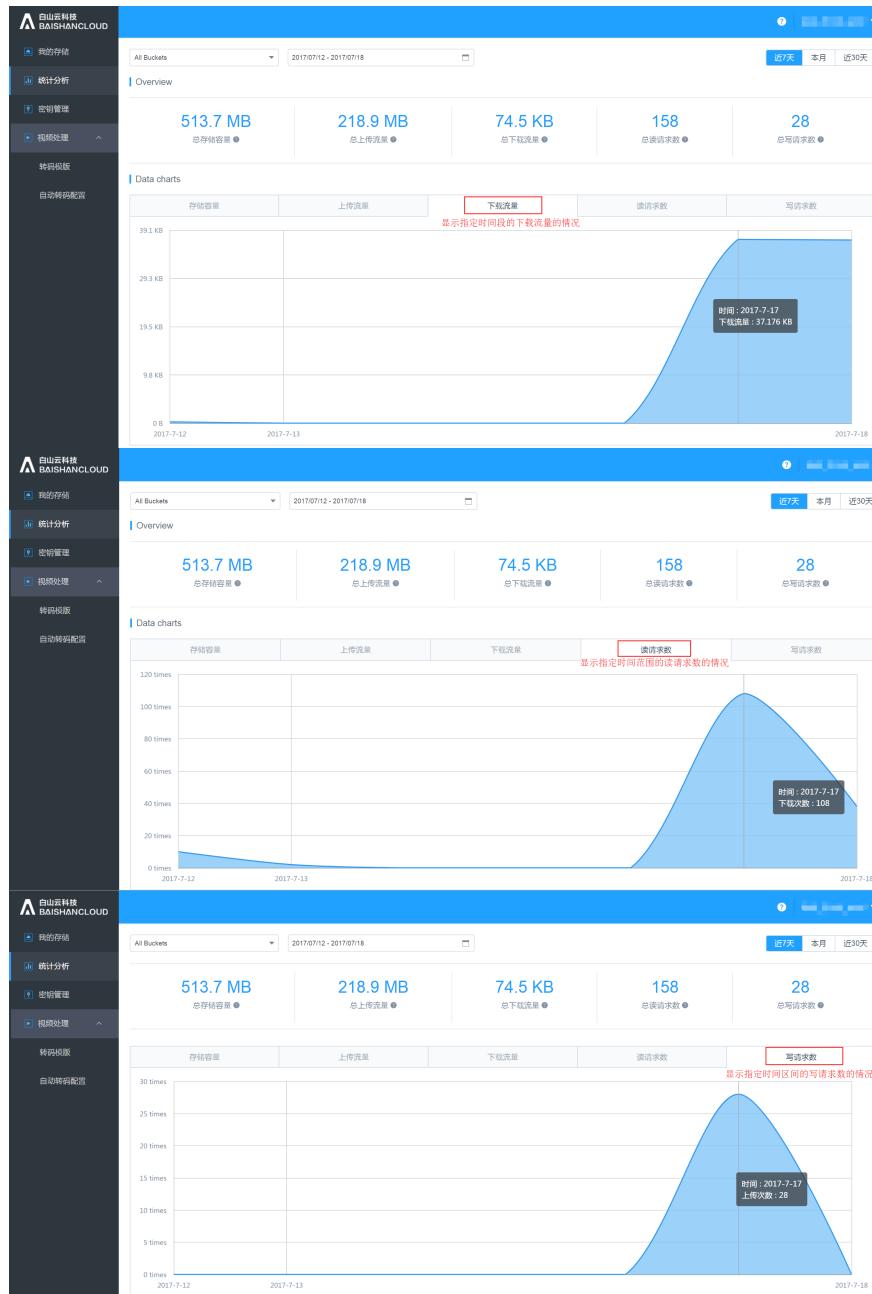


用户可选择查看不同空间的存储情况，如下图：



用户可根据需求查看不同数据，如下图：





1. 视频处理

1. 转码模板

视频模板是由用户预定义的可以重复使用的转码规则。其中主要规定了转码后的封装类型、编码方式、分辨率、码率等多媒体参数。

1. 转码模板管理

您可以在控制台创建转码模板。点击“转码模板”，进入到转码模板的管理页面。转码模板管理页面上列出了视频转码的基本参数。将鼠标移动至视频、音频区域则可以查看详细参数。在此页面上，您还可以新建转码模板，点击“新建模板”即可。

Name	Container	Video	Audio	Action
①	mp4	宽高比:1:1 帧率:23.97	码率:auto 分辨率:auto 编码方式:H.264 声道数:auto 编码方式:AAC 码率:140 采样率:22050	<input type="button" value="Delete"/>
st②	mp3	宽高比:1:1 帧率:auto	码率:auto 分辨率:auto 编码方式:auto 声道数:auto 编码方式:mp3 码率:auto 采样率:22050	<input type="button" value="Delete"/>
st③	mp4	宽高比:auto 帧率:auto	码率:auto 分辨率:auto 编码方式:H.264 声道数:auto 编码方式:AAC 码率:auto 采样率:44100	<input type="button" value="Delete"/>
④	mp4	宽高比:auto 帧率:auto	码率:auto 分辨率:auto 编码方式:auto 声道数:auto 编码方式:auto 码率:auto 采样率:auto	<input type="button" value="Delete"/>
	ts	宽高比:auto 帧率:auto	码率:auto 分辨率:auto 编码方式:H.265 声道数:auto 编码方式:auto 码率:auto 采样率:auto	<input type="button" value="Delete"/>

1. 创建转码模板

点击“新建模板”，

Name	Container	Video	Audio	Action
⑤	mp4	宽高比:1:1 帧率:23.97	码率:auto 分辨率:auto 编码方式:H.264 声道数:auto 编码方式:AAC 码率:140 采样率:22050	<input type="button" value="Delete"/>
⑥	mp3	宽高比:1:1 帧率:auto	码率:auto 分辨率:auto 编码方式:auto 声道数:auto 编码方式:mp3 码率:auto 采样率:22050	<input type="button" value="Delete"/>
⑦	mp4	宽高比:auto 帧率:auto	码率:auto 分辨率:auto 编码方式:H.264 声道数:auto 编码方式:AAC 码率:auto 采样率:44100	<input type="button" value="Delete"/>
⑧	mp4	宽高比:auto 帧率:auto	码率:auto 分辨率:auto 编码方式:auto 声道数:auto 编码方式:auto 码率:auto 采样率:auto	<input type="button" value="Delete"/>
⑨	ts	宽高比:auto 帧率:auto	码率:auto 分辨率:auto 编码方式:H.265 声道数:auto 编码方式:auto 码率:auto 采样率:auto	<input type="button" value="Delete"/>
⑩	mp4	宽高比:auto 帧率:auto	码率:auto 分辨率:auto 编码方式:auto 声道数:auto 编码方式:auto 码率:auto 采样率:auto	<input type="button" value="Delete"/>

“新建模板”的页面如下：填写“模板名称”、“模板描述”，勾选“输出封装格式”；当输出格式为mp4时，FastStart可以选择开启或关闭，当输出格式不是mp4时，没有FastStart这个选择项。

模板配置 新建模板

模板名称：
不少于1个字符，不能超过20个字符

模板描述：

输出封装格式： flac fiv gif mp4 mpg ts
FastStart

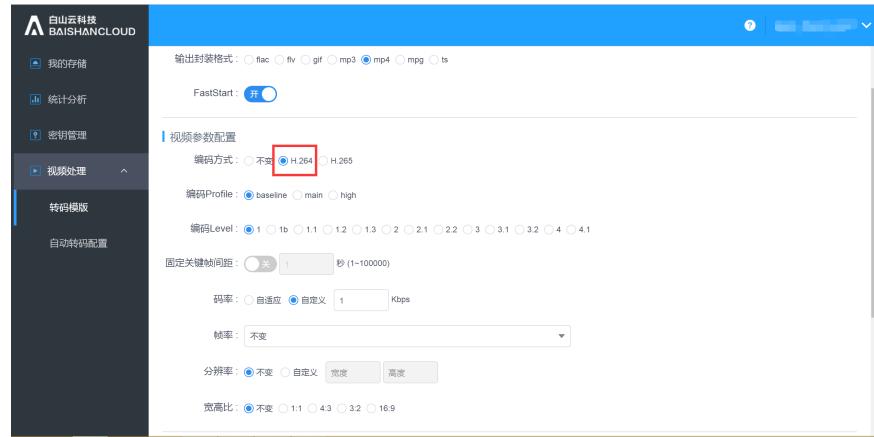
视频参数配置

编码方式： 不变 H.264 H.265
码率： 自适应 自定义 Kbps
帧率：

视频配置参数如下：当选择的编码是“不变”时，只用填“码率”和“帧率”。



当编码方式选择H.264时，需要选择“编码Profile”、“编码Level”、“固定关键帧间距”、“码率”、“帧率”、“分辨率”、“宽高比”。



当编码方式选择H.265时，需要选择“固定关键帧间距”、“码率”、“帧率”、“分辨率”、“宽高比”。



音频配置参数如下：



1. 自动转码配置

自动转码配置适用于以工作流的方式对新上传的文件进行转码。请注意新创建的自动转码配置只对新上传的文件有效，不会对应用于在创建配置之前上传的文件。点击“自动转码配置”，进入到自动转码配置的管理页面。

点击“新建配置”，进入到新建配置的页面。首先您需要配置“允许转码的路径规则”和“输入 Bucket”。对于“允许转码的路径规则”，您可以选择按照文件扩展名或正则表达式的方式输入；按文件扩展名配置路径时，您可以输入多个扩展名（多个扩展名之间使用“|”分割）；路径只能输入一个。

输出配置区：需要填写输出文件名前缀（可以为空）和输出Bucket。还需要填写输出规则和视频截图规则，输出规则和视频截图规则都可以配置多套。

输出规则的配置页面如下：当转码模板是输出的ts格式时，需要选择HLS切片时长；其它普通的输出只需要填写转码模板和输出文件名后缀即可。

视频截图的配置页面如下：需填写“输出文件名后缀”、“截图起始时间点”、“截图时间间隔”、“截图最大数量”、“分辨率”和“宽高比”。



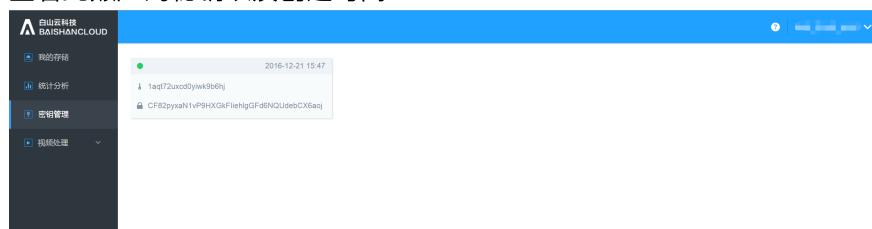
请注意在“输出规则”和“视频截图规则”中至少需要配置其中之一。当转码模板里有输出ts格式的时候，可以选择是否开启HLS自适应转码，如果开启，需要填写切片格式和MasterPlaylist文件名后缀。



在“更多配置”中，您可以配置转码后是否删除原始文件、转码后是否保留原始路径、转码失败回调URL、转码成功回调URL。权限设置：设置转码输出文件的权限。

1. 秘钥管理

查看此账户的秘钥以及创建时间：



1. 用户管理

如果您的账号是超级账号，您可以在控制台创建子帐号，并对子账号赋予相应的权限。

The screenshots illustrate the process of managing users in the BaiShanCloud control panel:

- Screenshot 1: User Management List**
Shows a list of existing users with columns for User name, Type, Email, and Action. One user, 'fetest12@baishancloud.com', is highlighted.
- Screenshot 2: New User Creation (Basic Information)**
Shows the 'New User' creation dialog. Fields include User name, Email, Password, and Company name. A large list of permissions is shown on the right under 'Actions'.
- Screenshot 3: New User Creation (Advanced Permissions)**
Shows the 'New User' dialog with advanced permission settings. It includes a table for Bucket permissions (Read, Write) and File permissions (Read, Write). A specific permission entry for 'fetest12' is highlighted with a red box.

1. 帮助文档

帮助用户快速查阅常用问题、要使用的函数、以及函数的用法及介绍，包括FAQ, SDK使用的例子，调用接口创建bucket、获取bucket列表、上传文件、下载文件等的例子，以及图片处理等子服务的介绍，图形界面工具上传文件的步骤说明等等。

The screenshot shows the BaiShanCloud storage interface with a prominent 'New Bucket' button at the top of the main content area.

下图为帮助文档界面：

概述

我们提供了一套兼容AmazonS3的RESTful API，可以使您更加自由地开发出灵活的功能。

白山云存储服务主要提供以下三类API：

- Service操作
- Bucket操作
- Object操作

与此同时，为提高用户使用的安全性，白山云存储服务通过使用签名来验证请求者的身份。

如需了解签名算法的详细信息，请参考[《签名算法》](#)。

注意：以下接口中所使用的示例都是在需要使用签名情况下；如果相关访问资源已设置为可匿名（所有用户）访问，则可不带签名。

各类SDK的使用例子一一展开详述：

安装AWS Python 客户端boto3

```
pip install boto3
```

初始化，设置帐号信息和域名

```
import boto3
from boto3.s3.transfer import TransferConfig

cli = boto3.client(
    's3',
    aws_access_key_id="z1w5dp1alvty9n47qksu", #请替换为自己的access_key
    aws_secret_access_key="VzTIZSuSwWvXb+KPSg0dWlzhMele372/yRKx4hZV", #请替换为自己的secret_key
    endpoint_url='http://s2.1.qingcdn.com'
)
```

文件操作接口