

OSS运维 基础实战手册

云运维工程师从入门到精通

作者：韩笠



OSS入门必知核心问题精解
阿里云OSS运维实战十二心法



阿里云开发者电子书系列



 **阿里云** 开发者社区



云服务技术大学
云产品干货高频分享



云服务技术课堂
和大牛零距离沟通



阿里云开发者“藏经阁”
海量免费电子书下载

目录

OSS 核心必知概念 4

OSS 的五个核心优势	4
简单易用，API 调配全掌握	7
6 个基础名词带你入门 OSS	18
实操前须知的 10 项使用限制	21
两大经典错误场景的镜像回溯	25
学会自动刷新配置，解决几十万用户存储问题	29
5 个场景全面了解跨域配置	32
不同场景下的 OSS 事件通知功能实现	45

OSS 典型解决方案与经典案例 52

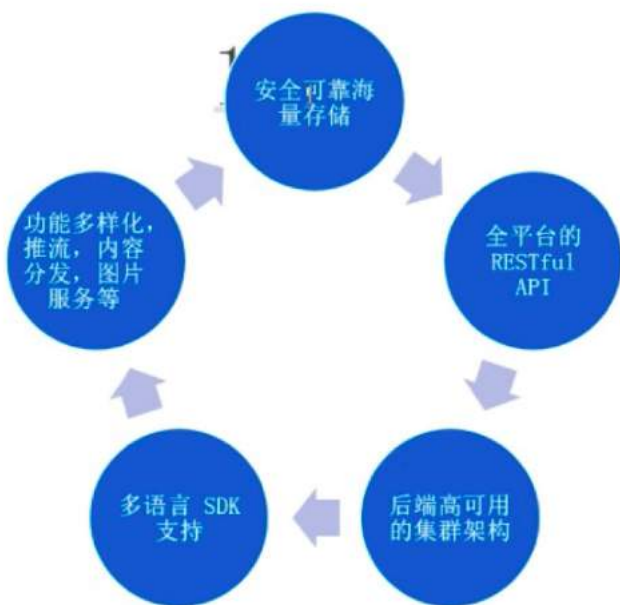
低价高效！OSS 如何处理视频	52
动静分离——实现全站加速	59
直播如何实时截图？两种配置方法实现	66
3 种方式快速完成 OSS 迁移数据	73
如何加强数据安全？3 种方式搞定	80
深度剖析鉴权 API，验签不再是难题	84
C SDK 安装这些坑，你踩到了吗？	94
3 条线索揭秘 OSS C SDK 问题核心	97
Andorid SDK 搭建还有疑问？6 步流程搞定	104
手把手教你 iOS SDK 搭建问题	122
如何安装 browsers JS SDK？这么做事半功倍	126
OSS pyhon SDK 问题排查攻略	131

OSS 核心必知概念

OSS 的五个核心优势

产品定义

阿里云对象存储服务 (Object Storage Service, 简称 OSS), OSS 具有与平台无关的 RESTful API 接口, 您可以在任何应用、任何时间、任何地点存储和访问任意类型的数据。兼容 AWS S3 协议以及 API, 提供了多样化语言的 SDK 支撑; 以及 RESTFUL 形式 API 调用。



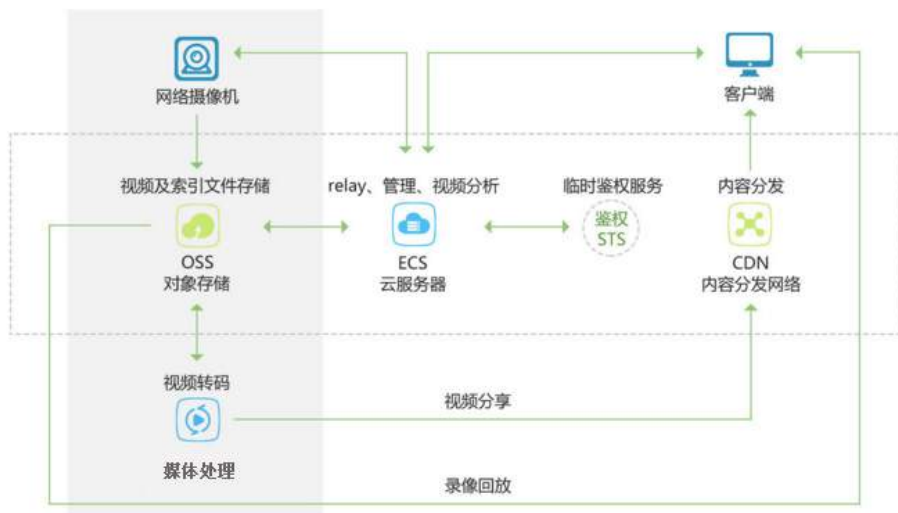
存储类型划分

数据存储到阿里云 OSS 以后，您可以选择标准存储（Standard）作为移动应用、大型网站、图片分享或热点音视频的主要存储方式，也可以选择成本更低、存储期限更长的低频访问存储（Infrequent Access）和归档存储（Archive）作为不经常访问数据的存储方式。

应用场景

图片和音视频等应用的海量存储

- OSS 可以结合媒体处理进行转码，OSS 作为输入输出的 bucket。
- OSS 可以结合视频直播使用作为文件录制的存储。
- OSS 可以作为直播推流来使用。



网页或者移动应用的静态和动态资源分离

OSS 可以实现海量数据的互联网并发下载，OSS 作为客户的存储伪源，面向下游的 CDN 产品提供静态内容存储、分发到边缘节点的解决方案。利用 CDN 边缘节

点缓存的数据，提升同一个文件，被同一地区客户大量重复并发下载的体验。同时可以结合自动刷新功能对同名文件上传进行刷新操作。



云端数据处理

上传文件到 OSS 后，可以通过 OSS 自身的功能或者 媒体处理，对存储的媒体内容进行处理，例如图片处理，视频截帧等。



简单易用，API 调配全掌握

开发者 API 调用

OSS 支持 RESTFUL API 形式调用，基本上服务端控制台上的功能配置，都可以通过 API 完成配置操作。也可以通过 OpenAPI 对文件进行集群的管理，结果用户访问控制台（RAM policy）加强客户的安全屏蔽，目前支持的 API 如下。

- GetService (ListBuckets)

对服务地址做 Get 请求可以返回请求者拥有的所有存储空间（Bucket），其中正斜线（/）表示根目录。

- PutBucket

PutBucket 接口用于创建存储空间（Bucket）

- DeleteBucket

DeleteBucket 用于删除某个存储空间（Bucket）。

- GetBucket (ListObjects)

GetBucket 接口用于列举存储空间（Bucket）中所有文件（Object）的信息。

- GetBucketInfo

GetBucketInfo 接口用于查看存储空间（Bucket）的相关信息。只有 Bucket 的拥有者才能查看 Bucket 的信息。

- GetBucketLocation

GetBucketLocation 接口用于查看存储空间（Bucket）的位置信息。只有 Bucket 的拥有者才能查看 Bucket 的位置信息。

- PutBucketACL

PutBucketACL 接口用于修改存储空间 (Bucket) 的访问权限。只有该 Bucket 的创建者有权限执行此操作。

- GetBucketAcl

GetBucketAcl 接口用于获取某个存储空间 (Bucket) 的访问权限 (ACL)。只有 Bucket 的拥有者才能获取 Bucket 的访问权限。

- PutBucketLifecycle

PutBucketLifecycle 接口用于设置存储空间 (Bucket) 的生命周期规则。生命周期规则开启后, OSS 将按照配置规则, 定期自动删除与规则相匹配的文件 (Object)。只有 Bucket 的拥有者才能发起此请求。

- GetBucketLifecycle

GetBucketLifecycle 接口用于查看存储空间 (Bucket) 的生命周期规则 (Lifecycle)。只有 Bucket 的拥有者才有权查看 Bucket 的生命周期规则。

- DeleteBucketLifecycle

DeleteBucketLifecycle 接口用于删除指定存储空间 (Bucket) 的生命周期规则。使用 DeleteBucketLifecycle 接口删除指定 Bucket 所有的生命周期规则后, 该 Bucket 中的文件 (Object) 不会被自动删除。只有 Bucket 的拥有者才能删除该 Bucket 的生命周期规则。

- PutBucketVersioning

PutBucketVersioning 用于设置指定 Bucket 的版本控制状态。只有 Bucket 所有者有权限执行此操作。

- GetBucketVersioning

GetBucketVersioning 接口用于获取指定 Bucket 的版本控制状态。

- GetBucketVersions(ListObjectVersions)

GetBucketVersions 接口用于列出 Bucket 中包括删除标记 (Delete Marker) 在内的所有 Object 的版本信息。

- PutBucketPolicy

PutBucketPolicy 接口用于为指定的存储空间 (Bucket) 设置授权策略 (Policy)。

- GetBucketPolicy

GetBucketPolicy 用于获取指定存储空间 (Bucket) 的权限策略 (Policy)。

- DeleteBucketPolicy

DeleteBucketPolicy 用于删除指定存储空间 (Bucket) 的权限策略 (Policy)。

- GetBucketLogging

GetBucketLogging 接口用于查看存储空间 (Bucket) 的访问日志配置。只有 Bucket 的拥有者才能查看 Bucket 的访问日志配置。

- DeleteBucketLogging

DeleteBucketLogging 用于关闭存储空间 (Bucket) 的访问日志记录功能。只有 Bucket 的拥有者才有权限关闭 Bucket 访问日志记录功能。

- PutBucketLogging

PutBucketLogging 接口用于为存储空间 (Bucket) 开启访问日志记录功能。访问日志记录功能开启后，OSS 将自动记录访问 Bucket 请求的详细信息，并以小时为单位将访问日志作为一个文件 (Object) 写入指定的 Bucket。

- PutBucketWebsite

PutBucketWebsite 接口用于将一个 bucket 设置成静态网站托管模式，以及设置跳转规则。

- GetBucketWebsite

GetBucketWebsite 接口用于查看存储空间 (Bucket) 的静态网站托管状态以

及跳转规则。

- DeleteBucketWebsite

DeleteBucketWebsite 接口用于关闭存储空间 (Bucket) 的静态网站托管模式以及跳转规则。只有 Bucket 的拥有者才能关闭 Bucket 的静态网站托管模式。

- GetBucketReferer

GetBucketReferer 接口用于查看存储空间 (Bucket) 的防盗链 (Referer) 相关配置。

- PutBucketReferer

PutBucketReferer 接口用于设置存储空间 (Bucket) 的防盗链 (Referer)。您可以使用此接口设置 Referer 的访问白名单以及是否允许 Referer 字段为空。

- PutBucketReferer

PutBucketTags 接口用来给某个 Bucket 添加或修改标签。

- GetBucketTags

GetBucketTags 用于获取存储空间 (Bucket) 的标签信息。

- DeleteBucketTags

DeleteBucketTags 接口用于删除存储空间 (Bucket) 标签。

- PutBucketEncryption

PutBucketEncryption 接口用于配置 Bucket 的加密规则。

- GetBucketEncryption

GetBucketEncryption 接口用于获取 Bucket 加密规则。

- DeleteBucketEncryption

DeleteBucketEncryption 接口用于删除 Bucket 加密规则。

- PutBucketRequestPayment

PutBucketRequestPayment 接口用于设置请求者付费模式。

- GetBucketRequestPayment

GetBucketRequestPayment 接口用于获取请求者付费模式配置信息。

- PutBucketCORS

PutBucketcors 接口用于为指定的存储空间 (Bucket) 设定一个跨域资源共享 (CORS) 规则。如果 Bucket 已有 CORS 规则，使用此接口会覆盖原有规则。Bucket 默认不开启 CORS 功能，所有跨域请求的 Origin 都不被允许。

- GetBucketCORS

GetBucketCORS 接口用于获取指定存储空间 (Bucket) 当前的跨域资源共享 (CORS) 规则。

- DeleteBucketCORS

DeleteBucketcors 用于关闭指定存储空间 (Bucket) 对应的跨域资源共享 (CORS) 功能并清空所有规则。

- PutObject

上传文件到 OSS

- GetObject

下载 OSS 文件

- CopyObject

CopyObject 接口用于在存储空间 (Bucket) 内或同地域的 Bucket 之间拷贝文件 (Object)。使用 CopyObject 接口发送一个 Put 请求给 OSS，OSS 会自动判断为拷贝操作，并直接在服务器端执行该操作。

- AppendObject

AppendObject 接口用于以追加写的方式上传文件 (Object)。通过 Append-

Object 操作创建的 Object 类型为 Appendable Object, 而通过 PutObject 上传的 Object 是 Normal Object。

- DeleteObject

DeleteObject 用于删除某个文件 (Object)。使用 DeleteObject 需要对该 Object 有写权限。

- DeleteMultipleObjects

DeleteMultipleObjects 接口用于删除同一个存储空间 (Bucket) 中的多个文件 (Object)。

- HeadObject

HeadObject 接口用于获取某个文件 (Object) 的元信息。使用此接口不会返回文件内容。

- GetObjectMeta

GetObjectMeta 接口用于获取一个文件 (Object) 的元数据信息, 包括该 Object 的 ETag、Size、LastModified 信息, 并且不返回该 Object 的内容。

- PostObject

PostObject 使用 HTML 表单上传 Object 到指定 Bucket。

- Callback

用户只需要在发送给 OSS 的请求中携带相应的 Callback 参数, 即能实现回调。本文详细介绍 Callback 的实现原理。

- RestoreObject

RestoreObject 接口用于解冻归档类型 (Archive) 的文件 (Object)。

- SelectObject

SelectObject 用于对目标文件执行 SQL 语句, 返回执行结果。

- `InitiateMultipartUpload`

使用 Multipart Upload 模式传输数据前，必须先调用该接口来通知 OSS 初始化一个 Multipart Upload 事件。

- `UploadPart`

初始化一个 MultipartUpload 之后，可以根据指定的 Object 名和 Upload ID 来分块 (Part) 上传数据。

- `UploadPartCopy`

`UploadPartCopy` 通过从一个已存在的 Object 中拷贝数据来上传一个 Part。

- `CompleteMultipartUpload`

在将所有数据 Part 都上传完成后，必须调用 `CompleteMultipartUpload` 接口来完成整个文件的 MultipartUpload。

- `AbortMultipartUpload`

`AbortMultipartUpload` 接口用于终止 MultipartUpload 事件。您需要提供 MultipartUpload 事件相应的 Upload ID。

- `ListMultipartUploads`

`ListMultipartUploads` 用来列举所有执行中的 Multipart Upload 事件，即已经初始化但还未 Complete 或者 Abort 的 Multipart Upload 事件。

- `ListParts`

`ListParts` 接口用于列举指定 Upload ID 所属的所有已经上传成功 Part。

- `PutObjectACL`

`PutObjectACL` 接口用于修改文件 (Object) 的访问权限 (ACL)。此操作只有 Bucket Owner 有权限执行，且需对 Object 有读写权限。

- `GetObjectACL`

`GetObjectACL` 接口用来获取某个存储空间 (Bucket) 下的某个文件 (Object)

的访问权限 (ACL)。

- GetSymlink

GetSymlink 接口用于获取软链接。此操作需要您对该软链接有读权限。

- PutSymlink

PutSymlink 接口用于为 OSS 的 TargetObject 创建软链接 (Symlink)，您可以通过该软链接访问 TargetObject。

- PutObjectTagging

您可以通过 PutObjectTagging 接口设置或更新对象的标签 (Object Tagging)。

- GetObjectTagging

您可以通过 GetObjectTagging 接口获取对象的标签信息。

- DeleteObjectTagging

您可以通过 DeleteObjectTagging 删除指定对象的标签。

- PutLiveChannel

通过 RTMP 协议上传音视频数据前，必须先调用该接口创建一个 LiveChannel。调用 PutLiveChannel 接口会返回 RTMP 推流地址，以及对应的播放地址。

- ListLiveChannel

ListLiveChannel 接口用于列举指定的 LiveChannel。

- DeleteLiveChannel

DeleteLiveChannel 接口用于删除指定的 LiveChannel。

- PutLiveChannelStatus

LiveChannel 分为 enabled 和 disabled 两种状态。您可以使用 PutLiveChan-

nelStatus 接口在两种状态之间进行切换。

- GetLiveChannelInfo

GetLiveChannelInfo 接口用于获取指定 LiveChannel 的配置信息。

- GetLiveChannelStat

GetLiveChannelStat 接口用于获取指定 LiveChannel 的推流状态信息。

- GetLiveChannelHistory

GetLiveChannelHistory 接口用于获取指定 LiveChannel 的推流记录。使用 GetLiveChannelHistory 接口最多会返回指定 LiveChannel 最近的 10 次推流记录。

- PostVodPlaylist

PostVodPlaylist 接口用于为指定的 LiveChannel 生成一个点播用的播放列表。OSS 会查询指定时间范围内由该 LiveChannel 推流生成的 ts 文件，并将其拼装为一个 m3u8 播放列表。

- GetVodPlaylist

GetVodPlaylist 接口用于查看指定 LiveChannel 在指定时间段内推流生成的播放列表。

开发者 SDK 调用

如果用户端不想自己计算鉴权，直接利用 SDK 封装的 API 工具集，免去鉴权的复杂计算。

支持 JAVA、python、PHP、GO、C、C#、C++、Ruby 等。

热点存储优化

OSS 后端采用的不是 hash key 的方式存储，而是采用了 LSM Tree 结构，系

统性能是靠分裂分区扩展的。这种场景要求用户在存储 OSS 内容时,在文件命名上要避开热点前缀。用户如果前缀带了类似日期信息热点前缀,可能导致在切换月份的时候从 N 个分区跳变到压力集中到一个分区。

细节约定

OSS 按照文件名 UTF-8 编码的顺序对用户数据进行自动分区,从而能够处理海量文件,以及承载高速率的客户请求。不过,如果您在上传大量文件时,在命名上使用了顺序前缀(如时间戳或字母顺序),可能会导致大量文件索引集中存储于某个特定分区。这样,当您的请求速率超过 2000 次/秒时(下载、上传、删除、拷贝、获取元数据信息等操作算 1 次操作,批量删除 N 个文件、列举 N 个文件等操作算 N 次操作),会带来如下后果:

- 该分区成为热点分区,导致分区的 I/O 能力被耗尽,或被系统自动限制请求速率。
- 热点分区的存在会触发系统进行持续的分区数据再均衡,这个过程可能会延长请求处理时间。

以上情况会降低 OSS 的水平扩展效果,导致客户的请求速率受限。

要解决这个问题,就要消除文件名中的顺序前缀。您可以在文件名前缀中引入某种随机性,这样文件索引(以及 I/O 负载)就会均匀分布在多个分区。

下面提供了几个将顺序前缀改为随机性前缀的方法示例。

```
sample-bucket-01/2017-11-11/customer-1/file1
sample-bucket-01/2017-11-11/customer-2/file2
sample-bucket-01/2017-11-11/customer-3/file3
...
sample-bucket-01/2017-11-12/customer-2/file4
sample-bucket-01/2017-11-12/customer-5/file5
sample-bucket-01/2017-11-12/customer-7/file6
...
```


针对这种情况, 可以对客户 ID 计算哈希, 即 MD5 (customer-id), 并取若干字符的哈希前缀作为文件名的前缀。假如取 4 个字符的哈希前缀, 结果如下:

```
sample-bucket-01/2c99/2017-11-11/customer-1/file1
sample-bucket-01/7a01/2017-11-11/customer-2/file2
sample-bucket-01/1dbd/2017-11-11/customer-3/file3
...
sample-bucket-01/7a01/2017-11-12/customer-2/file4
sample-bucket-01/b1fc/2017-11-12/customer-5/file5
sample-bucket-01/2bb7/2017-11-12/customer-7/file6
...
```

6 个基础名词带你入门 OSS

存储空间 (Bucket)

存储空间是用户用于存储对象 (Object) 的容器，所有的对象都必须隶属于某个存储空间。存储空间具有各种配置属性，包括地域、访问权限、存储类型等。用户可以根据实际需求，创建不同类型的存储空间来存储不同的数据。

- 同一个存储空间的内部是扁平的，没有文件系统的目录等概念，所有的对象都直接隶属于其对应的存储空间。
- 每个用户可以拥有多个存储空间。
- 存储空间的名称在 OSS 范围内必须是全局唯一的，一旦创建之后无法修改名称。
- 存储空间内部的对象数目没有限制。

存储空间的命名规范如下：

- 只能包括小写字母、数字和短横线 (-)。
- 必须以小写字母或者数字开头和结尾。
- 长度必须在 3 - 63 字节之间。

对象 / 文件 (Object)

对象是 OSS 存储数据的基本单元，也被称为 OSS 的文件。对象由元信息 (Object Meta)，用户数据 (Data) 和文件名 (Key) 组成。对象由存储空间内部唯一的 Key 来标识。对象元信息是一组键值对，表示了对象的一些属性，比如最后修改时间、大小等信息，同时用户也可以在元信息中存储一些自定义的信息。

对象的生命周期是从上传成功到被删除为止。在整个生命周期内，只有通过追加上传的 Object 可以继续通过追加上传写入数据，其他上传方式上传的 Object 内容无法编辑，您可以通过重复上传同名的对象来覆盖之前的对象。

对象的命名规范如下：

如果上传 `objectname` 命名含有一些特殊字符，比如中文、空格、特殊符号等，需要客户本地上传或者下载时将文件的 `objectname` 做一个 URL encoder 编码

- 使用 UTF-8 编码。
- 长度必须在 1 - 1023 字节之间。
- 不能以正斜线 (/) 或者反斜线 (\) 开头。

Region (地域)

Region 表示 OSS 的数据中心所在物理位置。用户可以根据费用、请求来源等选择合适的地域创建 Bucket。一般来说，距离用户更近的 Region 访问速度更快。详情请查看 OSS 已经开通的 Region。

Region 是在创建 Bucket 的时候指定的，一旦指定之后就不允许更改。该 Bucket 下所有的 Object 都存储在对应的数据中心，目前不支持 Object 级别的 Region 设置。

region 格式一般是 oss- 国家 - 地域，比如 oss-cn-beijing，就是代表存储的物理位置是在北京。

Endpoint (访问域名)

Endpoint 表示 OSS 对外服务的访问域名。OSS 以 HTTP RESTful API 的形式对外提供服务，当访问不同的 Region 的时候，需要不同的域名。通过内网和外网访问同一个 Region 所需要的 Endpoint 也是不同的。例如杭州 Region 的外网

Endpoint 是 `oss-cn-hangzhou.aliyuncs.com`，内网 Endpoint 是 `oss-cn-hangzhou-internal.aliyuncs.com`。具体的内容请参见各个 Region 对应的 Endpoint。

bucket 公网域名

指的是 bucket 和 endpoint 结合在一起，让用户访问的公网域名。完整的格式是 **bucket**.endpoint，比如我的 bucket 是 xiaoming，endpoint 在杭州 (`oss-cn-hangzhou.aliyuncs.com`)，那么我的 bucket 公网域名就是 `xiaoming.oss-cn-hangzhou.aliyuncs.com`。

AccessKey (访问密钥)

AccessKey (简称 AK) 指的是访问身份验证中用到的 AccessKeyId 和 AccessKeySecret。OSS 通过使用 AccessKeyId 和 AccessKeySecret 对称加密的方法来验证某个请求的发送者身份。AccessKeyId 用于标识用户；AccessKeySecret 是用户用于加密签名字符串和 OSS 用来验证签名字符串的密钥，必须保密。对于 OSS 来说，AccessKey 的来源有：

- Bucket 的拥有者申请的 AccessKey。
- 被 Bucket 的拥有者通过 RAM 授权给第三方请求者的 AccessKey。
- 被 Bucket 的拥有者通过 STS 授权给第三方请求者的 AccessKey。关于 AccessKey 可以在 [访问控制](#) 详细分析。

实操前须知的 10 项使用限制

归档存储

已经存储的数据从冷冻状态恢复到可读取状态需要 1 分钟的等待时间。

存储空间 (bucket)

- 同一阿里云账号在同一地域内创建的存储空间总数不能超过 100 个。
- 存储空间一旦创建成功，其名称、所处地域、存储类型不能修改。
- 单个存储空间的容量不限制。

上传 / 下载文件

- 通过控制台上传、简单上传、表单上传、追加上传的文件大小不能超过 5GB，要上传大小超过 5GB 的文件必须使用断点续传方式。
- 断点续传方式上传的文件大小不能超过 48.8TB。
- 同一账号在同一地域内的上传或下载的带宽缺省阈值为：中国内地各地域 10Gbit/s、其他地域 5Gbit/s。如达到该阈值，请求的 latency 会升高。如果您的业务（如大数据离线处理等）有更大的带宽需求（如 10Gbit/s~100Gbit/s），请工单升级到阿里云进行支持。建议客户端程序如果上传批量的但文件可以采用断点上传，可以将文件进行切片，降低网络拥塞。
- OSS 支持上传同名文件，但会覆盖已有文件。

删除文件

- 文件删除后无法恢复。

- 控制台批量删除文件的上限为 100 个，更大批量的删除必须通过 API 或 SDK 实现。

这里要注意下如果客户端没有开启 OSS bucket 版本控制，客户直接删除了文件（相当于物理性删除）那么文件无法找回。

域名绑定

- 账号必须在阿里云官网完成实名认证。
- 中国内地各地域绑定的域名必须在工信部备案，其他地域的域名绑定不需要在工信部备案。
- 每个存储空间最多可以绑定 100 个域名；一个域名只能绑定在一个存储空间上；每个账号可绑定的域名个数无限制。

生命周期

- 每个存储空间最多可配置 1000 条生命周期规则。

图片处理

- 图片限制

- 原图

图片格式只能是：jpg、png、bmp、gif、webp、tiff。

文件大小不能超过 20MB。

使用图片旋转时图片的宽或者高不能超过 4096px。

原图单边大小不能超过 30000px。

原图总像素不能超过 2.5 亿 px。

- 缩略后的图

宽与高的乘积不能超过 4096px*4096px。

单边长度不能超过 4096px。

- 样式限制

每个存储空间下最多能创建 50 个样式。如您的业务有更多的样式需求，请提交工单申请。

资源包

- 地域资源包只支持归属地域使用；中国大陆通用资源包仅支持在中国内地使用。
- 已购资源包不支持更换地域。
- 存储包不支持叠加购买，但您可以对已购存储包进行升级。
- 传输加速包和回源流量包支持叠加购买，但不支持升级和续费。
- 下行流量包支持叠加购买和续费，但不支持升级。
- 请求费用、数据处理费和跨区域复制流量费用暂时不支持包年包月付费。
- 资源包只能抵扣 OSS 的，不能抵扣 CDN 产品的流量费用。

文件压缩

OSS gzip 支持：

- 如果 GetObject 请求的 header 中添加了 Accept-Encoding: gzip，一般使用浏览器下载浏览器会协助增加。如果是使用 sdk 那么需要您在请求 header 头中设置。
- 文件大小必须大于或等于 1KB。
- Content-Type 必须是 "text/cache-manifest" "text/xml" "text/plain" "text/css" "application/javascript" "application/x-javascript" "application/rss+xml" 中的一个。

- 如果 object 的 " 设置 http 头 " 中手动设置 "content-Encoding: gzip", 如上面附件截图中的地方。那么 object 内容需要被 gzip 压缩后上传上来, 否则部分浏览器会不识别。
- gzip 是对下载文件的压缩 list object 不会压缩, 而且每次 list object 最多只会返回 1000 个 object 的名字。

文件访问限制

用户存储到 OSS 如果是 html 文件或者 图片文件, 不能直接在浏览器中访问, 会被增加一个强制下载头 (Content-Disposition: attachment; filename={filename})。需要用户绑定一个已经备案的域名后, 通过绑定的域名去访问 OSS 文件才可以。

绑定域名时, 这些二级域是保留域名, 不能直接和 OSS 绑定。

```
"aliyuncs.com",  
"aliyun.com",  
"aliyun-inc.com",  
"tbcn.cn",  
"tbcn.com",  
"aliimg.com",  
"taobaocdn.com",  
"zmx.com.cn",  
"alipay.com",  
"sinopec.com",  
"gzdata.com.cn",  
"nxcloud.com.cn",  
"paytm.com",  
"mybank.cn",  
"gzdata.gov.cn",  
"taobao.com"
```


两大经典错误场景的镜像回溯

浅谈

OSS 回源功能，也称镜像回源，可以类比 Nginx 的 PCRE 满足一定条件下触发的 rewrite 功能，但是 OSS 的功能更加丰富。大致分为了两种主要功能，一种是 404 回源，另外是重定向回源。

使用的场景，一般是数据迁移、主备 OSS、CDN 结合 OSS、等多个场景。

功能说明

404

当客户请求到 OSS 获取资源时，如果 OSS 咨询不存在，那么会根据客户配置的镜像回源地址去拉去对应的资源，如果真正的源站有数据就会响应 OSS 200，OSS 会将文件一同存储到本地，并且透传给客户端 200 的状态码。

在这个过程中，如果 OSS 回源没有回去到资源，源站响应了 404，那么 OSS 也直接将 404 透传给用户。

如果 OSS 回源获取资源失败，源站返回了一个非 200 404 的状态码，比如 502，那 OSS 会记录 424，然后透传给源站的状态码 502 给客户端。

重定向

回源另外一个功能就是重定向，重定向，分为 301、302、307。常用的就是 302 301，配置重定向功能后，客户端请求到 bucket A 后，会直接 302 301 到用户

配置的 bucket B，客户端直接根据 301 302 follow 到目的 bucket B。

配置功能

创建规则 ×

回源类型

☒ 镜像 ☐ 重定向

使用镜像方式配置回源规则，当请求在 OSS 没有找到文件，会自动到源站抓取对应文件保存到 OSS，并将内容直接返回给用户

回源条件

☒ HTTP 状态码 404 ☐ 文件名前缀

回源地址

:// / /文件名

例如：

OSS 访问地址：
bucketname.oss-endpoint.com/image.jpg
回源获取文件地址：
<http://回源域名/image.jpg>

回源参数

☐ 携带请求字符串 ①

3xx 请求响应策略

☒ 跟随源站重定向请求 ①

设置 HTTP header 传递规则 ①

允许 ☐ 传递所有 HTTP header ☐ 传递指定 HTTP header

禁止 ☐ 禁止传递指定 HTTP header

设置 ☐ 设置指定 HTTP header 参数

云栖社区 yq.aliyun.com

这里不会给大家说全部的功能，因为详细使用官网有，我们着重说下标红的位置。

- 文件前缀：如果勾选了文件前缀，择访问出现 404 后，只会针对匹配到的 OSS 前缀触发镜像回源，否则不会触发。
- 回源参数：如果是 CDN 回源时会携带很多参数，想要保留这些参数透传到源站，可以开启此参数。

- 允许 header：如果配置了允许所有 header，默认将客户端请求过来的 header 全部透传给源站。
- 禁止 header：可以自定义哪些 header 是不透传回源的。
- 设置：触发回源后，可以自定义一些请求头透传给源站。

案例介绍

一、配置了镜像回源但是响应 404

- 首先固定真正的源站访问同样的 URL 测试是否 200，只有源站响应了 200 的情况 OSS 回源才能获取到资源。
- 看是否配置了透传所有 header 头选项，如果配置了该选项需要同时配置禁止 host 头透传回去，否则会出现回源时 404 的问题。因为镜像回源时会携带当前 bucket host 回源，所以目标源站肯定也响应 404。

二、配置了镜像回源但是响应 502

- 出现这种问题，都是客户源站响应超时导致，可以先固定源站的 URL 测试一下。
- 分析对应时间点源站 error log 或者 access log 是否有 502 的情况，看下源站的网络链路负载等。

三、镜像回源的源站和访问的域名一致

- 访问 www.a.com 资源 404 回源到 B.oss-cn-beijing.aliyuncs.com，B bucket 上设置回源到 www.a.com，这种情况是不可以的，会造成死循环的问题。
- 访问 A.oss-cn-beijing.aliyuncs.com 资源 404 回源到 B.oss-cn-beijing.aliyuncs.com，B bucket 设置的源站是 A bucket 也不行。

四、镜像回源报错 "Error status: 0 from mirror host ,should return 200 ok"

```
com.aliyun.oss.OSSException: Error status : 0 from mirror host, should return 200 OK.  
[ErrorCode]: MirrorFailed  
[RequestId]: 5C1CE79F6787AD8B56989810  
[HostId]: .oss-cn-hangzhou.aliyuncs.com  
[ResponseError]:  
<?xml version="1.0" encoding="UTF-8"?>  
<Error>  
  <Code>MirrorFailed</Code>  
  <Message>Error status : 0 from mirror host, should return 200 OK.</Message>  
  <RequestId>5C1CE79F6787AD8B56989810</RequestId>  
  <HostId>.oss-cn-hangzhou.aliyuncs.com</HostId>  
</Error>
```

云栖社区 yq.aliyun.com

出现类似报错就是 OSS 在触发镜像回用户原站拉资源时，源站没有返回 200 的内容，而是返回了其他非 200 的状态码，导致拉取资源失败。

这种场景，用户端只要固定源站测试以下就可以知道问题。

学会自动刷新配置，解决几十万用户存储问题

场景描述

对象存储 (OSS) 主要场景是在文件存储以及对文件管理的云产品。不带有文件下行或者上行加速的功能，很多用户也会通过 CDN + OSS 的方式结合，通过 CDN 的优势加速文件访问速度。那么怎么能保证 CDN 缓存下的文件在原站 (OSS) 更新的情况下自动更新呢？今天就说下自动刷新的功能。

what Refresh

使用过 CDN 都知道 它本身带有刷新接口功能，在调用刷新时被动的将 CDN 缓存住的旧文件内容刷新掉，但用户需要预知有哪些文件，或者这些文件在哪些目录；如果量级少还可以，如果几十万量级的 URL 用户端很一次性搜集完，而且调用刷新接口也需要大量的时间等待。如果用户的文件是存在 OSS 上，OSS 提供了这个自动刷新的功能，替代用户手动搜集 URL 提交刷新的过程。

开启 OSS 自动刷新

CDN 控制台

在 CDN 控制台，域名管理界面将原站类型改为用户要绑定的 OSS 地址即可。（这里要注意如果你的 OSS 是私有的，要把 CDN 私有 bucket 回源功能开启，否则会导致 CDN 回源到 OSS 失败。如果客户端请求 URL 中带了 OSS 签名，那可以不用开启此功能）



阿里云OSS私有Bucket回源

阿里云OSS私有Bucket回源



支持权限为Private的OSS源站的内容加速，有效防止资源盗链，源站为非OSS时，无法开启此功能。 [什么是私有Bucket回源?](#)

子账号开启私有Bucket回源前需要获得ListRoles授权，操作见 [文档](#)



OSS 控制台

当 CDN 已经配置好，在 OSS 控制台的域名管理界面可以看到已经出现“自动刷新”开关，点击按钮即可开启。





开启效果

1. 用户上传同名文件到 OSS 后，OSS 会主动调用 CDN 刷新接口将旧文件清理掉。
2. OSS 刷新的任务优先级比主动掉 CDN 刷新接口的优先级低，这是正常现象，因为 OSS 操作文件的量突发性会大。
3. 当用户文件几十万或者百万，同时在线更新时，OSS 调用 CDN 的刷新任务会出现排队情况，因为为了保护接口稳定性，每个账号下的任务并发数量是有限制的。所以建议用户端能分批去更新 OSS 文件。
4. OSS 的刷新会占用 CDN 的刷新配额。
5. 如果 OSS 的文件被删除后，也会调用 CDN 刷新将文件 cache 清理掉。

5 个场景全面了解跨域配置

简介：经常遇到有跨域的问题，老生长谈，却又屡禁不止，谈到跨域我们就了解下它是什么？（以下数据均为模拟数据，屏蔽了真实用户数据）

什么是“跨域”

一句话简单说明

一个资源请求一个其它域名的资源时会发起一个跨域 HTTP 请求 (cross-origin HTTP request)。比如说，域名 A `http://domaina.example` 的某 Web 应用通过 `` 标签引入了域名 B: `http://domainb.foo` 的某图片资源 `http://domainb.foo/image.jpg`，域名 A 的 Web 应用会触发浏览器发起一个跨域 HTTP 请求。

demo

```
http://www.123.com/index.html 调用 http://www.123.com/server.php (非跨域)

http://www.123.com/index.html 调用 http://www.456.com/server.php (主域名不同
:123/456, 跨域)

http://abc.123.com/index.html 调用 http://def.123.com/server.php (子域名不同
:abc/def, 跨域)

http://www.123.com:8080/index.html 调用 http://www.123.com:8081/server.php (端
口不同 :8080/8081, 跨域)

http://www.123.com/index.html 调用 https://www.123.com/server.php (协议不同
:http/https, 跨域)
```

请注意：localhost 和 127.0.0.1 虽然都指向本机，但也属于跨域。

跨域请求标识

origin，当浏览器识别出 client 发起的请求需要转到另外一个域名上处理是，会在请求的 request header 中增加一个 origin 标识，如下我用 curl 测试了一个域名。

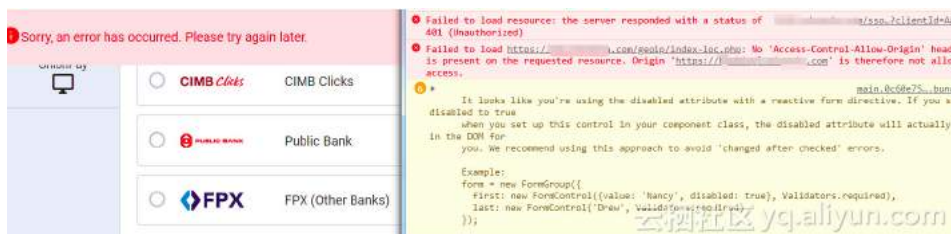
```
curl -voa http://mo-im.oss-cn-beijing.aliyuncs.com/stu_avatar/010/personal.
jpg -H "Origin:www.
mobby.cn"
  % Total    % Received % Xferd  Average   Speed    Time     Time     Time  Current
           Dload    Upload   Total   Spent    Left     Speed
  0      0      0      0      0      0      0      0  --:--:--  --:--:--  --:--:--
0*   Trying 59.110.190.173...
*   TCP_NODELAY set
*   Connected to mo-im.oss-cn-beijing.aliyuncs.com (59.110.190.173) port 80
(#0)
> GET /stu_avatar/010/personal.jpg HTTP/1.1
> Host: mo-im.oss-cn-beijing.aliyuncs.com
> User-Agent: curl/7.54.0
> Accept: */*
> Origin:www.mo.cn
>
< HTTP/1.1 200 OK
< Server: AliyunOSS
< Date: Sun, 09 Sep 2018 12:30:28 GMT
< Content-Type: image/jpeg
< Content-Length: 8407
< Connection: keep-alive
< x-oss-request-id:
< Access-Control-Allow-Origin: www.mobby.cn
< Access-Control-Allow-Credentials: true
< Access-Control-Allow-Methods: GET, POST, HEAD
< Access-Control-Max-Age: 0
< Accept-Ranges: bytes
```

可以看到挡我发起 Origin 的请求头后，如果目标的网页服务允许来源的域名访问，就会在响应的 Response header 中带上跨域的响应头。（以下 header 目标域名如果设置了才会有响应）

```
< Access-Control-Allow-Origin: www.mobby.cn (允许的跨域来源，可以写 *, 或者绝对域名)
< Access-Control-Allow-Headers: * (允许跨域时携带哪些 header )
< Access-Control-Allow-Methods: GET, POST, HEAD (允许哪些跨域请求方法, origin 是默认支持的)
```

常见案例分析

场景一：CDN 访问 CDN 跨域被拦截



通过报错可以看出 发起跨区域请求的源头 是 bo3.ai.com 加载了 www.ai.com 网站的资源，这两个域名都在 阿里云 cdn 加速。既然找到了请求目的 www.ai.com，那么直接检查下目的域名上是否新增了跨域头。这种情况基本都是目的域名没有加上允许的跨域头导致。

场景二：直传 OSS 引用 CDN 资源被拦截

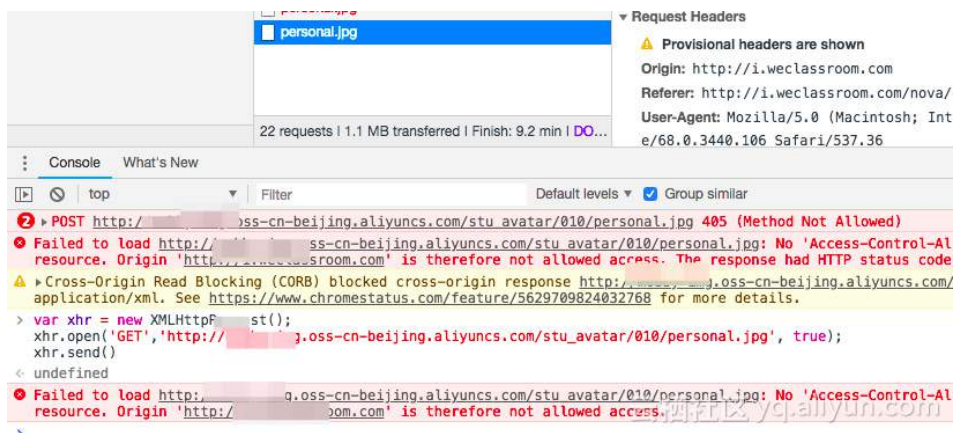


用户直接上传到 OSS，但是应用了 CDN 的域名时出现的跨域的报错，出现这种情况因为引用的 CDN 上没有配置跨域的属性所以报错，在 CDN 上配置好跨域参数后问题解决。

找到阿里 CDN 控制台对应的 CDN 域名，配置 http header 头，增加三个属性，如下：



场景三：CDN 回源到 OSS



这个问题比较特殊，拆分两部分说明。

出现这种情况，通过截图我们发现用户有两种请求，分别是 `GET` 和 `POST` 两种，由于 `GET` 好测试，我们先说 `GET`。

GET:

出现跨域错误，首先就要检查原是否添加了跨域头，于是我们使用 curl 测试一下，如果最简单的 get 测试成功返回跨域头，说明目的 域名设置了跨域响应，如果测试失败说明原没有添加跨域响应。通过如下图很显然看到了目的添加了跨域头。

```
curl -voa http://mo-im.oss-cn-beijing.aliyuncs.com/stu_avatar/010/personal.
jpg -H "Origin:www.
```

```

mo.cn"
  % Total    % Received % Xferd  Average Speed   Time    Time     Time
Current                                  Dload  Upload   Total   Spent    Left  Speed
  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--
0*   Trying 59.110.190.173...
*   TCP_NODELAY set
*   Connected to mo-im.oss-cn-beijing.aliyuncs.com () port 80 (#0)
> GET /stu_avatar/010/personal.jpg HTTP/1.1
> Host: mo-im.oss-cn-beijing.aliyuncs.com
> User-Agent: curl/7.54.0
> Accept: */*
> Origin:www.mo.cn
>
< HTTP/1.1 200 OK
< Server: AliyunOSS
< Date: Sun, 09 Sep 2018 12:30:28 GMT
< Content-Type: image/jpeg
< Content-Length: 8407
< Connection: keep-alive
< x-oss-request-id: 5B951264980F8FDB749972B3
< Access-Control-Allow-Origin: www.mo.cn
< Access-Control-Allow-Credentials: true
< Access-Control-Allow-Methods: GET, POST, HEAD
< Access-Control-Max-Age: 0

```

POST

通过 `GET` 测试发现 oss 是加了跨域头的，但是为什么 `POST` 请求就返回 405 呢？没有任何跨域头呢？用户反馈为什么手动 curl 测试也是失败。

```

curl -v -X POST -d '{"user":"xxx"}' http://mo-im.oss-cn-beijing.aliyuncs.com/
stu_avatar/010/personal.
jpg -H "Origin:www.mo.cn"
Note: Unnecessary use of -X or --request, POST is already inferred.
*   Trying 59.110.190.173...
*   TCP_NODELAY set
*   Connected to mo-im.oss-cn-beijing.aliyuncs.com (59.110.190.173) port 80
(#0)
> POST /stu_avatar/010/personal.jpg HTTP/1.1
> Host: mo-im.oss-cn-beijing.aliyuncs.com
> User-Agent: curl/7.54.0
> Accept: */*
> Origin:www.mo.cn
> Content-Length: 14
> Content-Type: application/x-www-form-urlencoded

```

```
>
* upload completely sent off: 14 out of 14 bytes
< HTTP/1.1 405 Method Not Allowed
< Server: AliyunOSS
< Date: Sun, 09 Sep 2018 13:06:28 GMT
< Content-Type: application/xml
< Content-Length: 337
< Connection: keep-alive
< x-oss-request-id:
< Allow: GET DELETE HEAD PUT POST OPTIONS
<
<?xml version="1.0" encoding="UTF-8"?>
<Error>
  <Code>MethodNotAllowed</Code>
  <Message>The specified method is not allowed against this resource.</
Message>
  <RequestId></RequestId>
  <HostId>mo-im.oss-cn-beijing.aliyuncs.com</HostId>
  <Method>POST</Method>
  <ResourceType>Object</ResourceType>
</Error>
```

- 首先我们看 oss 对于 `POST` 的要求，看完我们就找到原因了。

https://help.aliyun.com/document_detail/31988.html?spm=a2c4g.11186623.6.1008.5bb84b4e1JEoA4

结论:

- 请求的格式不是 RFC 标准规定的 `content-type: multipart/form-data`。
- 请求头不是内容不是 RFC 规定的表单域提交。
- 既然不是表单域，那么 OSS API 要求的 `filename` 参数肯定也不是放在最后一个选项。

JAVA 跨域请求源码

```
package com.alibaba.edas.carshop.OSS;

import javax.activation.MimetypesFileTypeMap;
import javax.crypto.Mac;
import javax.crypto.spec.SecretKeySpec;
```

```

import org.apache.commons.codec.binary.Base64;

import java.io.*;
import java.net.HttpURLConnection;
import java.net.URL;
import java.security.InvalidKeyException;
import java.security.NoSuchAlgorithmException;
import java.util.Iterator;
import java.util.LinkedHashMap;
import java.util.Map;
import java.util.Map.Entry;

public class OSSPostFile {
    // The local file path to upload.
    private String localFilePath = "C:\\T\\1.txt";
    // OSS domain, such as http://oss-cn-hangzhou.aliyuncs.com
    private String endpoint = "http://oss-cn-beijing.aliyuncs.com";
    // Access key Id. Please get it from https://ak-console.aliyun.com
    private String accessKeyId = "";
    private String accessKeySecret = "";
    // The existing bucket name
    private String bucketName = "您自己的bucket名称";
    // The key name for the file to upload.
    private String key = "1.txt";

    public void PostObject() throws Exception {
        // append the 'bucketname.' prior to the domain, such as
        // http://bucket1.oss-cn-hangzhou.aliyuncs.com.
        String urlStr = endpoint.replace("http://", "http://" + bucketName +
            ".");

        // form fields
        Map<String, String> formFields = new LinkedHashMap<String, String>();

        // key
        formFields.put("key", this.key);
        // Content-Disposition
        formFields.put("Content-Disposition", "attachment;filename=" +
            localFilePath);
        // OSSAccessKeyId
        formFields.put("OSSAccessKeyId", accessKeyId);
        // policy
        String policy = "{\\"expiration\\":
            "\2120-01-01T12:00:00.000Z\\",\\"conditions\\": [[\\"content-length-
            range\\", 0, 104857600000]]}";
        String encodePolicy = new String(Base64.encodeBase64(policy.
            getBytes()));
        formFields.put("policy", encodePolicy);
        // Signature

```

```

        String signaturecom = computeSignature(accessKeySecret,
        encodePolicy);
        formFields.put("Signature", signaturecom);

        String ret = formUpload(urlStr, formFields, localFilePath);

        System.out.println("Post Object [" + this.key + "] to bucket [" +
        bucketName + "]);
        System.out.println("post reponse:" + ret);
    }

    private static String computeSignature(String accessKeySecret, String
    encodePolicy)
        throws UnsupportedEncodingException, NoSuchAlgorithmException,
        InvalidKeyException {
        // convert to UTF-8
        byte[] key = accessKeySecret.getBytes("UTF-8");
        byte[] data = encodePolicy.getBytes("UTF-8");

        // hmac-sha1
        Mac mac = Mac.getInstance("HmacSHA1");
        mac.init(new SecretKeySpec(key, "HmacSHA1"));
        byte[] sha = mac.doFinal(data);

        // base64
        return new String(Base64.encodeBase64(sha));
    }

    private static String formUpload(String urlStr, Map<String, String>
    formFields, String localFile)
        throws Exception {
        String res = "";
        HttpURLConnection conn = null;
        String boundary = "9431149156168";

        try {
            URL url = new URL(urlStr);
            conn = (HttpURLConnection) url.openConnection();
            conn.setConnectTimeout(5000);
            conn.setReadTimeout(30000);
            conn.setDoOutput(true);
            conn.setDoInput(true);
            conn.setRequestMethod("POST");
            conn.setRequestProperty("User-Agent", "Mozilla/5.0 (Windows; U;
            Windows NT 6.1; zh-CN;
            rv:1.9.2.6)");
            conn.setRequestProperty("Content-Type", "multipart/form-data;
            boundary=" + boundary);
            OutputStream out = new DataOutputStream(conn.getOutputStream());

```

```

// text
if (formFields != null) {
    StringBuffer strBuf = new StringBuffer();
    Iterator<Entry<String, String>> iter = formFields.entrySet().
iterator();

    int i = 0;

    while (iter.hasNext()) {
        Entry<String, String> entry = iter.next();
        String inputName = entry.getKey();
        String inputValue = entry.getValue();

        if (inputValue == null) {
            continue;
        }

        if (i == 0) {
            strBuf.append("--").append(boundary).append("\r\n");
            strBuf.append("Content-Disposition: form-data;
name=\"\" + inputName + "\"\r\n\r\n");
            strBuf.append(inputValue);
        } else {
            strBuf.append("\r\n").append("--").append(boundary).
append("\r\n");
            strBuf.append("Content-Disposition: form-data;
name=\"\" + inputName + "\"\r\n\r\n");
            strBuf.append(inputValue);
        }

        i++;
    }
    out.write(strBuf.toString().getBytes());
}

StringBuffer strBuf1 = new StringBuffer();
String callback = "{\"callbackUrl\":\"http://47.93.116.168/Revice.
ashx\", \"callbackBody\":\"{\\"bucket\\"=\"$${bucket}\",\\"size\\"=\"$${size}\"\"}\"";

byte[] textByte = callback.getBytes("UTF-8");
strBuf1.append("\r\n").append("--").append(boundary).append("\r\
n");

String callbackstr = new String(Base64.encodeBase64(textByte));
strBuf1.append("Content-Disposition: form-data;
name=\"callback\"\r\n\r\n" + callbackstr + "\r\
n\r\n");

out.write(strBuf1.toString().getBytes());

```



```

        // file
        File file = new File(localFile);
        String filename = file.getName();
        String contentType = new MimetypesFileTypeMap().
getContentType(file);
        if (contentType == null || contentType.equals("")) {
            contentType = "application/octet-stream";
        }

        StringBuffer strBuf = new StringBuffer();
        strBuf.append("\r\n").append("--").append(boundary).append("\r\n");

        strBuf.append("Content-Disposition: form-data; name=\"file\"; " +
"filename=\"" + filename +
"\r\n");
        strBuf.append("Content-Type: " + contentType + "\r\n\r\n");
        out.write(strBuf.toString().getBytes());

        DataInputStream in = new DataInputStream(new
FileInputStream(file));
        int bytes = 0;
        byte[] bufferOut = new byte[1024];
        while ((bytes = in.read(bufferOut)) != -1) {
            out.write(bufferOut, 0, bytes);
        }
        in.close();

        byte[] endData = ("\r\n--" + boundary + "--\r\n").getBytes();
        out.write(endData);
        out.flush();
        out.close();

        // Gets the file data
        strBuf = new StringBuffer();
        BufferedReader reader = new BufferedReader(new
InputStreamReader(conn.getInputStream()));
        String line = null;
        while ((line = reader.readLine()) != null) {
            strBuf.append(line).append("\n");
        }
        res = strBuf.toString();
        reader.close();
        reader = null;
    } catch (Exception e) {
        System.err.println("Send post request exception: " +
e.getLocalizedMessage());
        throw e;
    } finally {
        if (conn != null) {

```


初步分析：出现的原因是因为用户之前历史配置过的规则中含有特殊字符导致控制台拉取是否有历史配置时失败，响应了 invalidresponse。

解决方法：用户通过 SDK 修改跨域规则，通过 SDK 配置的规则将历史规则覆盖掉。

场景五：客户端使用 cavens 测试图片的跨域访问被 403

```
248ms html2canvas: Added image data:image/png;base64,iVBORw0KGgoAAAANSUHEUgAAAMMAAACICAYAAAD8
131VAAAAAXNSR-wxASjkCAAEKJjNk6vu/IRmK+me7skcPUNV79cz3XW++le9eIwVl.iJpJAISAYmAREAi4IyAdv
255ms html2canvas: Added image http://localhost:3000/static/media/share-t-bg.6a0616f9.png Logger.js:36
256ms html2canvas: Added image http://teacher-center-avator.oss-cn-beijing.aliyuncs.com/imag
e/2018062...jpg Logger.js:36
260ms html2canvas: Added image https://chn-fs.oss-cn-beijing.aliyuncs.com/711c7e6...jpg Logger.js:36
262ms html2canvas: Finished parsing node tree Logger.js:36
❶ Access to image at 'http://teacher-center-avator.oss-cn-beijing.aliyuncs.com/image/2018062...
pg' from origin 'http://localhost:3000' has been blocked by CORS policy: No 'Access-Control-Allow-Origin'
header is present on the requested resource.
❷ GET http://teacher-center-avator.oss-cn-beijing.aliyuncs.com/image/2018062...jpg ResourceLoader.js:153
net::ERR_FAILED
❸ Access to image at 'https://chn-fs.oss-cn-beijing.aliyuncs.com/711c7e6...jpg' from origin 'htt
p://localhost:3000' has been blocked by CORS policy: No 'Access-Control-Allow-Origin' header is present on
the requested resource.
❹ GET https://chn-fs.oss-cn-beijing.aliyuncs.com/711c7e6...jpg net::ERR_FAILED ResourceLoader.js:153
272ms html2canvas: Unable to load image Logger.js:36
> Event {isTrusted: true, type: "error", target: img, currentTarget: img, eventPhase: 2, ...}
276ms html2canvas: Unable to load image Logger.js:36
> Event {isTrusted: true, type: "error", target: img, currentTarget: img, eventPhase: 2, ...}
278ms html2canvas: Finished loading 5 images > (5) [img, img, img, null, null] Logger.js:36
281ms html2canvas: Starting renderer Logger.js:36
```

1. 先确认 OSS 是否非配置了跨域头，配置的是否正确。



2. 出现类似问题可以使用 postman 或者 curl 工具测试，看下是否同样出现问题。

```
[root@edas02 python]# curl -I "https://chn-fs.oss-cn-beijing.aliyuncs.com/711c7e68de084e148cf2356a3c54f51f.jpg" -H "Origin:www.taobao.com"
HTTP/1.1 200 OK
Server: AliyunOSS
Date: Mon, 31 Dec 2018 16:20:28 GMT
Content-Type: image/jpeg
Content-Length: 37499
Connection: keep-alive
x-oss-request-id: 5C2A41CC2BFA718B4D7A8A78
Access-Control-Allow-Origin: *
Access-Control-Allow-Methods: GET, POST, PUT, HEAD
Access-Control-Max-Age: 0
Accept-Ranges: bytes
ETag: "06A3D89A3F07CC8968FCCA17CD649A14"
Last-Modified: Sat, 29 Dec 2018 06:31:14 GMT
x-oss-object-type: Normal
x-oss-hash-crc64ecma: 15342513831561008656
x-oss-storage-class: Standard
Content-MD5: BqPTmj8Hz0lo/MoXaWSaFA==
x-oss-server-time: 66
```

云栖社区 yq.aliyun.com

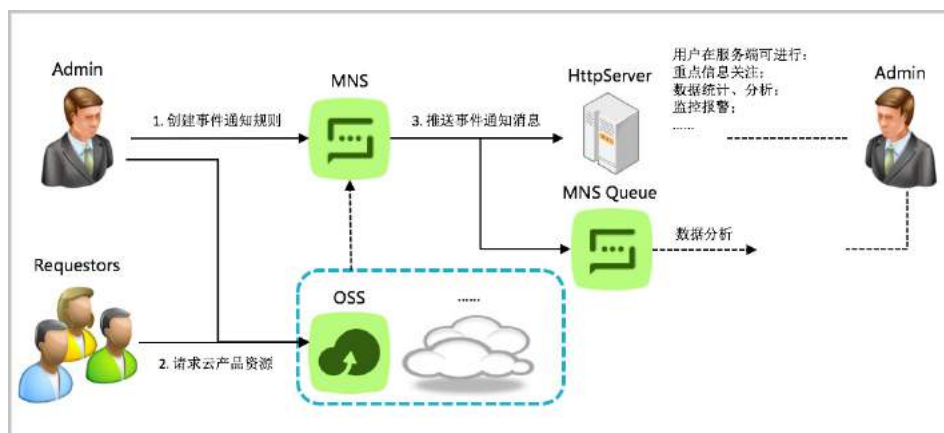
3. 如果发现本地测试跨域头都是正常的，只有客户端的浏览器测试异常，请用
户清除浏览器缓存，开启隐私模式进行测试。

不同场景下的 OSS 事件通知功能实现

功能描述

您可以在创建事件通知规则的时候，自定义您关注的 Object 信息，当这些资源发生变化后，您可以第一时间收到通知。例如：

- 有新数据从图片内容分享平台、音视频平台上传到 OSS。
- OSS 上的相关内容发生了更新。
- OSS 上的重要文件被删除。
- OSS 上数据同步已经完成。



开通须知

1. 通知方式分为 HTTP 通知和队列通知。

- HTTP 是用户填入一个服务器的 URL 地址，OSS 在监测到客户操作完成后，

去 POST URL 地址回调用户。

- 队列的方式需要用户开通阿里云的 MNS 服务，配置好队列后在 OSS 控制台填入队列名称，OSS 通过 MNS 回调用户。

创建规则

事件通知规则须知:

1. 系统会自动为新建的规则创建主题，主题实例可能产生费用，详见[消息服务价格](#)。

2. 删除规则后，主题不会自动删除，可以登录[消息服务控制台](#)进行删除。

规则名称

0/128

事件类型

请至少选择一个事件类型

资源描述

全名

//

添加

1 / 5

接收终端

HTTP

http:// 开头的协议格式

添加

4 / 5

确定

取消

2. 系统会自动为新建的规则创建主题，主题实例可能产生费用，详见 [消息服务价格](#)。
3. 删除规则后，主题不会自动删除，可以登录 [消息服务控制台](#) 进行删除。

4. 规则配置分为两种，根据资源描述分为：全名、前后缀。很多用户在这里遇到了坑，需要注意。

- 全名：用户上传的文件必须和规则配置的绝对一致才会触发，比如全名是 `bucket/image/1.png`，那用户上传也要是 `image/1.png`，如果上传的是 `image/2.png` 就不会通知。
- 前后缀：用户上传的文件前缀或者后缀满足条件即可触发。比如前缀为 `buck-et/202003-`，这种规则可以匹配到 `202003-01`，`202003-02....`，如果配置的后缀为 `bucket/202003-/jpg`，可以匹配到 `202003-01/1.jpg`，`202003-02/2.jpg`。

常见问题分析

场景一：控制台上配置事件通知报错，`configurationcountlimit-exception`

出现这种问题异常后说明默认的，mns 产品的事件通知已经达到上限（默认是 10），如果想要提升数量请提交工单联系 mns 支持人员提升，非 OSS 故障。



场景二: ossutil 上传文件成功, 但是客户配置的 mns 队列没有收到通知

可能导致问题原因如下:

- 用户上传失败, 有可能被劫持虽然反馈了状态码, 但是没有返回 `x-oss-request-id` 标识, 建议用户把判断成功标准改为 `httpcode == 200`, 并且 `requestID != null`。
- mns 产品可能出现消息堆积, 需要提交工单确认。
- 用户配置的 oss 事件类型不全, 比如 OSS 控制台用的是 `PostObject`, ossutil 上传大文件使用的分片传输。如果规则没配置全就不会触发。
- 用户上传文件的 `prefix` 和事件通知配置的不符, 或者用户使用的是全名匹配但上传用的是前后缀。
- 用户配置的是前后缀规则, 但是前缀并不是单独创建的 object, 而且上传文件时和 object 一起创建的, 比如, 用户如过是通过这种方式 创建的 `imageprefix/1.png`, 那 `imageprefix` 不在是一个独立的前缀, 而是和 `imageprefix/1.png` 绑定在一起。

如何兼得 bucket policy 简单易操作和精细权限粒度

简介：bucket policy 相比用户自己定义 ram policy 更为简单，界面操作清晰，但是权限粒度不如 ram policy 精细。

功能描述

- 用户想针对子账号授权访问某个 bucket，将只读、读写、完全控制分开授权。
- 用户想针对跨账号进行授权。
- 用户想针对匿名用户进行管理。
- 在授权基础上增加条件限制，从 IP 纬度限制用户的访问。

bucket policy 和 RAM policy 区别

- 二者同时配置的情况下，可以并行存在，但如果策略出现冲突，将以范围粒度更大的优先。
- RAM policy 更适合开发者通过 AccesskeyID，AccesskeySecret 进行调用。bucket policy 用在控制台登录或者匿名用户访问的场景较多。
- RAM policy 定义的 Action (API) 细化的场景更多，bucket policy 粒度较粗。

授权场景

先了解完全控制和读写有什么区别，完全控制是说对 bucket 属性本身进行控制，包括 bucket 的配置权限。读写只是能上传、下载 OSS 的文件。

bucket 是私有的，只开放某个目录公开访问

新增授权 ×

授权资源

整个 Bucket

指定资源

资源路径 ?

/ 123/456/

授权用户 ?

子账号

其他账号

匿名账号 (*)

*

授权操作

只读

读/写

完全控制

拒绝访问

条件

请选择

▼

bucket 是私有的，只想某个 IP 匿名能访问

新增授权 ×

授权资源

整个 Bucket

指定资源

资源路径 ?

/

授权用户 ?

子账号

其他账号

匿名账号 (*)

*

授权操作

只读

读/写

完全控制

拒绝访问

条件

IP =

▼

1.1.1.1

支持输入 IP 地址和 IP 地址段。如有多个 IP 地址，请用英文逗号分隔。

只想让子账号拥有某个目录的管理权限，其他目录要写具体权限

1. 先在 OSS 控制台上增加一个具体目录的操作权限。
2. 然后通过这个工具自动编排一个细粒度的 RAM 权限即可。

自助工具: <https://ali.zhangyb.mobi/robot>

RAM Policy Editor v1.2.0

[English](#)[Star](#)

添加规则

Effect Actions Resources

- 每添加一个Resource后按回车确认
- 例子: my-bucket, my-bucket/dir/
- [More...](#)

EnablePath ☐ 自动设置父目录权限 ?Conditions
(Optional)

授权策略

```
{
  "Version": "1",
  "Statement": []
}
```

OSS 典型解决方案与经典案例

低价高效！OSS 如何处理视频

简介：当用户需要对 OSS 存储的音视频文件做专业内容处理，并且希望保留源文件将处理后的内容再存储到 OSS 上，可以使用 OSS + MPS 方案解决。

先了解 MPS

媒体处理 (ApsaraVideo Media Processing, 原 MTS) 是一种多媒体数据处理服务。它以经济、弹性和高可扩展的音视频转换方法，帮助您将存储于 OSS 的音视频转码成适合在 PC、TV 以及移动终端上播放的格式。并基于海量数据深度学习，对音视频的内容、文字、语音、场景多模态分析，实现智能审核、内容理解、智能编辑。

支持格式

输入格式

- 容器格式: 3GP、AVI、FLV、MP4、M3U8、MPG、ASF、WMV、MKV、MOV、TS、WebM、MXF。
- 视频编码格式: H.264/AVC、H.263、H.263+、H.265、MPEG-1、MPEG-2、MPEG-4、MJPEG、VP8、VP9、Quicktime、RealVideo、Windows Media Video。

- 音频编码格式：AAC、AC-3、ADPCM、AMR、DSD、MP1、MP2、MP3、PCM、RealAudio、Windows Media Audio。

输出格式

- 容器格式：
 - 视频：FLV、MP4、HLS (m3u8+ts)、MPEG-DASH (MPD+fMP4)。
 - 音频：MP3、MP4、OGG、FLAC、m4a。
 - 图片：GIF、WEBP。
- 视频编码格式：H.264/AVC、H.265/HEVC。
- 音频编码格式：MP3、AAC、VORBIS、FLAC。

MPS 转码方式

API 提交转码作业

通过 API 根据 OSS 文件存储地址的方式，用 MPS 预置系统的转码模版进行转码。将源文件下载后进行转码然后在回传到 OSS。

提交转码作业

通过 API 根据 OSS 文件存储地址的方式，用自定义的 MPS 模版进行转码。将源文件下载后进行转码然后在回传到 OSS，这种方式比较灵活，能自定义转码模版适合自由度高，对音视频编解码深度了解的客户。

自定义转码模版

提交转码作业

workflow 自动触发

如果用户有大量文件新传到 OSS 需要批量触发转码可以通过工作来完成。这种模式是 OSS 通过 mns 事件通知的方式告知 MPS，然后触发 workflow，开始按照配置

的转码规则进行转码。

工作的特点可以大批量文件自动触发执行，通过管道的方式设置消息通知来回调客户端，异步非阻塞的模式降低用户的代码成本。

工作流可以配置多个，每个工作流可以使用监听不同的 OSS prefix，使用不同的媒体转码管道，将用户的不同业务隔离开。



工作流配置方法

1. 首先用户要先开通媒体处理产品功能，并且绑定好媒体处理所在地区的 bucket 信息，这里媒体处理的输入输出 bucket region 需要和媒体处理开通的区域一致。

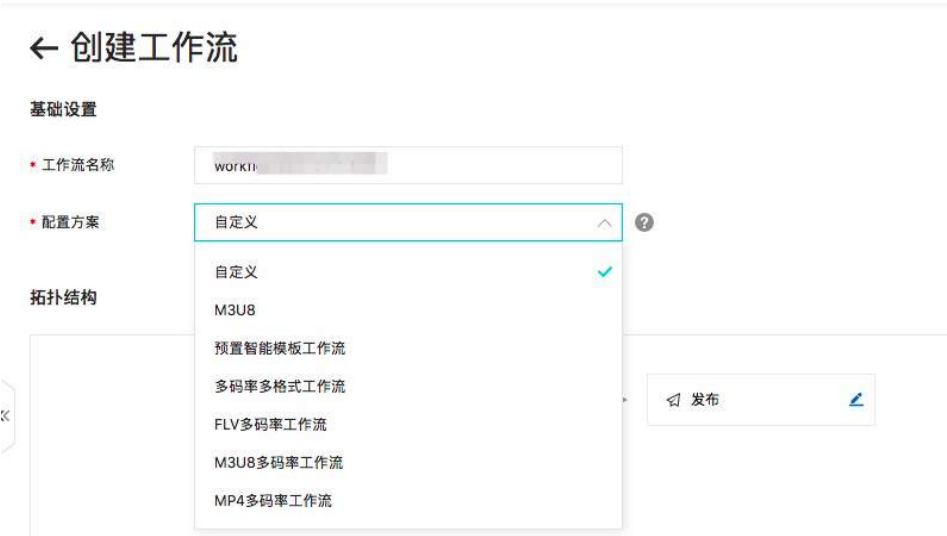
而且需要注意，媒体处理绑定 bucket 是要有授权的，所以需要子账号尽量具备 OSS 管理权限。



2. 创建工作流。

创建工作流选用方式很多种，可以自己定义一个规则，选择自己想要转码还是截图，灵活度很高。

如果用户不是对编解码很懂，也可以使用系统推荐的一些模版。但这里要注意尽量不要用预置智能模版，智能模版需要对源文件进行音视频文件分析然后和你转码输出的规则匹配，如果匹配失败则不会触发转码，尽量不要使用，如果很清楚自己业务输入视频和输出视频的内容规则可以使用。



3. 如果选择自定工作流，可以在 + 号的位置，灵活定义你需要的功能，不需要的不用引入；但是需要注意下，如果转码后的文件最后发布时没有选择自动发布，那么 OSS 转码后的文件还不能被公开访问到，需要手动发布下，建议都用自动发布，如果有内容鉴黄的需求可以改成手动发布。



4. 如果需要工作流转码完成后回调用户，可以对管道设置消息通知，这里用的是 mns 消息服务产品，涉及到消息服务产品的计费请先了解好。



- 配置好的工作流方式是监听到 OSS 的事件后自动触发，但很多用户时 API 或者 SDK 调用工作流，希望能控制工作流的触发模式，可以参考[更新媒体工作流触发模式](#)。

触发模式。（范围：OssAutoTrigger 自动执行、NotInAuto 非自动执行）

FQA

如果获取 OSS 视频文件编码信息

用户可以调用 [查询媒体](#) 接口来获取 OSS 存储的视频文件信息（视频宽高、码率、容器格式等信息）。

如何加速大文件的转码效率

用户基本上都是 1G 以上的大文件，一次性提交了多个文件，类似场景可以提交工单申请倍速转码管道来提高批量大文件的转码效率，但是如果是几百兆或者以下的文件不推荐用倍速转码，并无太多提升效果。

如果文件较多有上百个或者几十个大文件并发转码时，也可以申请新的转码管

道, 将 OSS 下不同 prefix 的视频文件按业务分管道转码也可以提高效率。

取消大量转码中的作业

如果用户提交太多的大文件, 转码需要很长时间才能完成这是正常情况, 如果用户等不及需要取消转码中的文件, 分多个管道完成也可以的。

需要先调用 [列出转码作业](#) 找到需要取消的 jobid , 然后调用 [取消转码作业](#)。

OSS 视频文件能宽高自适应吗

用户可以选择一边固定, 比如宽, 然后另外一边自适应不用填写。

1 基础设置

2 音频参数

3 高级参数

基础参数

转码模版名称

封装格式

mp4

视频参数

禁用视频

编码格式

H.264

编码级别

适合高分辨率设备

码率

质量控制因子

视频原始宽度值范围: [128, 4096]

视频宽度(px)

720

视频高度(px)

横竖屏自适应

帧率(fps)

动静分离——实现全站加速

简介：全站加速（DCDN）与 OSS 是常见的站点动静分离的方式，可以实现将静态资源存储在 OSS 上，并通过 DCDN 加速 OSS 实现静态资源的访问加速效果。如果是动态业务，可以通过最短路由上传回传。

功能描述

首先阐述如果能解决动静态分析的场景

DCDN 支持动态的文件加速规则，用户可以配置配置哪些是静态文件需要缓存，哪些是动态文件不需要缓存。

开启动态加速

- 开启：可自定义动静态资源加速规则，静态内容使用边缘缓存，动态内容采用最优路由回源，支持三种静态文件匹配方式。（URI、后缀、路径）
- 关闭：无动态内容加速效果，仅保留静态边缘缓存功能。

动态加速

动态加速



开启：可自定义动静资源加速规则，静态内容使用边缘缓存，动态内容采用最优路由回源 [动态请求计费详情](#)
关闭：无动态内容加速效果，仅保留静态边缘缓存功能

静态文件类型

静态URI

静态路径

协议跟随回源

静态文件类型

.gif	.png
.bmp	.jpg
.jpeg	.mp3
.wma	.flv
.mp4	.wmv
.ogg	.avi

指定需要边缘缓存的文件类型。通常为静态资源设置边缘缓存，动态资源通过最优路由加速 [如何配置静态文件类型？](#)

[修改配置](#)

配置方法

创建 DCDN 域名时，选择 OSS 域名作为原站，在下来框里面可以自动检索出客户的 OSS bucket。

回源端口建议使用 443 回源避免被劫持。



回源规则

这个规则默认不用更改，但当原站如果是私有 bucket 权限，客户端不想每次自己计算 OSS signature 签名，那可以把“私有 bucket 回源”功能开启，这样用户不用自己计算签名，通过 DCDN 来自动算签。



缓存规则配置

用户如果之前配置来静态路由规则后，在缓存规则里面可以配置对应的缓存时间，那动态的文件就直接回源，静态的走本地 cache。



跨域配置

当 DCDN 域名回源到 OSS 时会触发跨域，因为 DCDN 和 OSS 是两个完全不同的主站，所以需要在 OSS 上配置跨域。





如果客户端请求 DCDN 也是两个不同的主站域名那么在 DCDN 上也要配置。

Access-Control-Allow-Origin

Access-Control-Allow-Headers

Access-Control-Allow-Methods



开启 OSS 自动刷新功能

用户如果更新 OSS 静态文件后，希望 DCDN 将同名的旧文件从缓存中清理掉，可以开启自动刷新功能。



FQA

使用 DCDN 和 OSS 后静态文件为强制下载

由于 OSS 的默认策略在访问 3 级域名时，会给文件添加 attachment 属性，导致文件为强制下载。

需要修改 DCDN 的回源 HOST，配置为加速域名而不是 Bucket 域名（即不是为 aliyuncs.com 结尾的域名，此域名为 OSS 的默认域名）。

访问 OSS 静态文件大小和 DCDN 缓存文件大小不一致

OSS 通过 putObject 等上传方式都是会在 response 头中记录 content-length 和 content-MD5 的信息返回给客户端，用户可以根据该信息确定本地下载得到的文件是否与 OSS 服务器端存储的数据是否一致。


```
< HTTP/1.1 200 OK
< Server: AliyunOSS
< Date: Thu, 30 Nov 2017 01:40:55 GMT
< Content-Type: image/png
< Content-Length: 105207
< Connection: keep-alive
< x-oss-request-id: 5A1F61A696C0068918762B94
< Accept-Ranges: bytes
< ETag: "65209C627AB8B2B6E59B5074105EF385"
< Last-Modified: Tue, 28 Nov 2017 11:05:41 GMT
< x-oss-object-type: Normal
< x-oss-hash-crc64ecma: 9348850889578336387
< x-oss-storage-class: Standard
< Content-MD5: ZSCcYnq4srblm1B0EF7zhQ==
< x-oss-server-time: 124
```

但使用 DCDN 加速 OSS 出现文件大小和 OSS 不一致时可以从一下方面进行排查：

- 获取 DCDN 上的历史缓存。由于 OSS 上的文件更新而 DCDN 上仍然缓存着历史的旧数据导致的该问题，可以通过刷新 DCDN 缓存解决。
- DCDN 的智能压缩功能。DCDN 会对满足特定条件的文件自动做 gzip 压缩，当客户端发送的 Request 头有 Accept-Encoding : gzip，即表示客户端支持 gzip 压缩并且满足 CDN 智能压缩就会进行压缩，而压缩后就会导致该文件更改为 chunked 编码，将无法获取得到 content-length。
- DCDN 的页面优化功能。DCDN 针对于 html 文件提供了 trim 的功能，即 DCDN 在开启页面优化功能后可以帮助用户自动去掉 html 页面中的空格以及注释，这样可以减少下行流量。但是这就会导致客户端接收到的 content-length 或者 content-MD5 发生变化。
- HTTP 劫持问题。当如果客户端到 DCDN 的边缘节点或者 DCDN 父层节点回源到源站使用 HTTP 协议时数据传输是非加密的，因此是有可能出现在网络传输的过程中包内容被篡改的情况。这种情况就会导致客户端接收与 OSS 存储内容不一致。该问题可以通过修改为 HTTPS 协议规避该问题。

直播如何实时截图？两种配置方法实现

简介：OSS 作为多媒体的使用，不仅能结合媒体处理使用，也可以结合视频直播，作为 录制、截图的存储原站。

功能描述

- 视频直播是将推流端的数据流实时的通过播流地址进行播放，因此后续如果需要再对播放历史直播数据就必须要使用视频直播的录制功能。视频直播的录制功能就是将视频中心接收到的推流端推流的数据进行录制，并将其保存成 hls 协议的封装格式存储到 OSS 中。
- 用户常需要对视频直播的内容进行分析。例如，需要对直播中的某一帧的内容作为封面图片或者对于直播内容进行鉴黄以查看内容合法性等需求。因此，视频直播提供了实时截图功能满足用户的截图需求。
- 视频直播的录制功能可以将直播数据录制成 HLS 协议文件，详情参考 直播录制功能介绍。视频直播录制在直播过程中会生成 TS 文件，但是仅在推流结束 180 秒后才会生成 m3u8 索引文件，无法在直播过程中生成，并且该索引文件时间为推流开始到结束，用户无法根据业务需要自定义索引范围。因此，视频直播提供录制索引管理功能供用户对录制索引 m3u8 文件的管理功能。

录制功能介绍

配置视频直播的录制功能

- 通过控制台创建：控制台配置方法请参考录制存储至 OSS、录制存储至 VOD。

- 通过 API/SDK 创建: API/SDK 提供了创建、删除以及查询的功能，创建录制配置请参考 添加 APP 录制配置，删除录制配置请参考 删除 APP 录制配置，查看录制配置可以分别查询整个直播域名和单个 AppName 的录制配置，请参考 查询域名录制配置 和 查询 APP 录制配置。下面提供使用 Java SDK 添加 APP 录制配置的示例代码。

录制代码如下











```
IClientProfile profile = DefaultProfile.getProfile("cn-
hangzhou", "<AccessKeyId>",
"<AccessKeyScret>");
IAcsClient client = new DefaultAcsClient(profile);

AddLiveAppRecordConfigRequest addLiveAppRecordConfigRequest = new
AddLiveAppRecordConfigRequest();
addLiveAppRecordConfigRequest.setDomainName("<DomainName>");
addLiveAppRecordConfigRequest.setAppName("<AppName>");
addLiveAppRecordConfigRequest.setOssEndpoint("<Endpoint>");
addLiveAppRecordConfigRequest.setOssBucket("<BucketName>");
addLiveAppRecordConfigRequest.setOssObjectPrefix("<ObjectPrefix>");
try {
    AddLiveAppRecordConfigResponse addLiveAppRecordConfigResponse =
client.
getAcsResponse(addLiveAppRecordConfigRequest);
    System.out.println(addLiveAppRecordConfigResponse.
getRequestId());
    // todo something.
} catch (ServerException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
} catch (ClientException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}
```

查看录制文件

录制配置后新发起的推流在满足 AppName 录制配置时即会自动录制推流的数据。推流过程中即会生成对应的 TS 文件到配置目录中。而对应的索引 m3u8 文件需要在推流 180 秒结束后生成（180 秒为兼容直播推流过程由于网络抖动等

问题导致的推流短时间中断)。其默认命名规则为 {AppName}/{StreamName}/{EscapedStartTime}_{EscapedEndTime}.m3u8，其中，AppName 为直播流所属应用名称，StreamName 为流名称，EscapedStartTime 为录制开始时间，EscapedEndTime 为录制结束时间，下图即是一组录制后的 ts 和 m3u8 文件列表示意图。

<input type="checkbox"/>		1574409784_33.ts
<input type="checkbox"/>		1574409817_34.ts
<input type="checkbox"/>		1574409851_35.ts
<input type="checkbox"/>		1574409884_36.ts
<input type="checkbox"/>		1574409917_37.ts
<input type="checkbox"/>		1574409951_38.ts
<input type="checkbox"/>		1574409984_39.ts
<input type="checkbox"/>		1574410017_40.ts
<input type="checkbox"/>		1574410054_41.ts
<input type="checkbox"/>		1574410065_42.ts

截图功能

用户常需要对视频直播的内容进行分析。例如，需要对直播中的某一帧的内容作为封面图片或者对于直播内容进行鉴黄以查看内容合法性等需求。因此，视频直播提

供了实时截图功能满足用户的截图需求。

截图功能配置

配置视频直播的截图功能可以通过两种方法：

- 通过控制台创建。参见 [配置截图](#)。
- 通过 API/SDK: API/SDK 提供了添加、删除、查询和更新截图配置的接口。

详情参见 [添加截图配置](#)、[删除截图配置](#)、[查询截图配置](#)、[更新截图配置](#)、[查询截图信息](#)。下面提供使用 Java SDK 添加截图配置的示例代码。

```
IClientProfile profile = DefaultProfile.getProfile("cn-
hangzhou", "<AccessKeyId>",
"<AccessKeySecret>");
IAcsClient client = new DefaultAcsClient(profile);

AddLiveAppSnapshotConfigRequest addLiveAppSnapshotConfigRequest = new
AddLiveAppSnapshotConfigRequest();
addLiveAppSnapshotConfigRequest.setDomainName("<DomainName>");
addLiveAppSnapshotConfigRequest.setAppName("<AppName>");
addLiveAppSnapshotConfigRequest.setTimeInterval(5);
addLiveAppSnapshotConfigRequest.setOssEndpoint("<Endpoint>");
addLiveAppSnapshotConfigRequest.setOssBucket("<BucketName>");
addLiveAppSnapshotConfigRequest.setOverwriteOssObject("{AppName}/
{StreamName}.jpg");
try {
    AddLiveAppSnapshotConfigResponse addLiveAppSnapshotConfigResponse =
client.
getAcsResponse(addLiveAppSnapshotConfigRequest);
    System.out.println(addLiveAppSnapshotConfigResponse.getRequestId());
    // todo something.
} catch (ServerException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
} catch (ClientException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}
```

查看截图文件

上述截图配置完成后新发起的推流即可生成截图，查看截图的方法包括以下三种：

OSS 控制台提供截图管理功能。参见 删除截图。

API/SDK 提供了查询截图信息的接口。参见 查询截图信息。

在知晓截图录制地址时可以直接在该媒体 Bucket 的对应目录查看生成结果，请参考下图。



文件名 (Object Name)	文件大小	存储类型	更新时间
返回上一步 (AppName/StreamName/)			
<input type="checkbox"/> 1.jpg	37.976KB	标准存储	2017-11-07 14:55:42
<input type="checkbox"/> 10.jpg	36.527KB	标准存储	2017-11-07 14:56:28
<input type="checkbox"/> 100.jpg	38.13KB	标准存储	2017-11-07 15:03:58
<input type="checkbox"/> 101.jpg	36.834KB	标准存储	2017-11-07 15:04:03
<input type="checkbox"/> 102.jpg	36.634KB	标准存储	2017-11-07 15:04:08
<input type="checkbox"/> 103.jpg	36.433KB	标准存储	2017-11-07 15:04:13

FQA

录制常见问题

- 直播录制是针对于推流的 AppName 为粒度设置的，用户可以设置某个 AppName 下的所有 StreamName 的录制规则。并且 AppName 支持 “*” 通配符，表示该录制规则是针对于该直播域名下的所有推流均生效的，录制生效需要注意推流的 AppName 需要与配置的 AppName 匹配。
- 使用直播录制功能是需要开通视频点播服务的，并且录制生成的 m3u8 和 TS 文件都将存放在视频点播的输入媒体 Bucket 中，详情参考 媒体 Bucket 的增删改操作。

3. 录制设置中可以设置变量，默认的录制 TS 文件存放地址为：record/{Date}/{AppName}/{StreamName}/{UnixTimestamp}_{Sequence}，其中的变量均是使用“{}”引起的，用户可以自行修改或者变更为常量。各变量的意义请参考控制台配置录制。
4. 录制设置仅会对配置完成后的新发起的推流记录生效，当前的推流需中断 180 秒后重新推流方可生效。
5. 录制配置中 ObjectName 是包括了前缀的文件名称，OSS 中的目录是逻辑概念，目录是 Object 的 key 值的前缀。详情参考 OSS 目录 / 文件夹概念。
6. 录制自动生成的索引 m3u8 文件仅有在断流 180 秒后才可以正常生成，如果用户需要在没有断流时生成索引文件提供播流访问，请参考 直播录制索引 创建。
7. 同样的 AppName 和 StreamName 不能同时存储至 VOD 和 OSS，只能二者选其一，不能重复添加、冲突。

截图常见问题

1. 直播截图是针对于推流的 AppName 为粒度设置的，用户可以设置某个 AppName 下的所有 StreamName 的截图规则。并且 AppName 支持 “*” 通配符，表示该截图规则是针对于该直播域名下的所有推流均生效的，截图生效需要注意推流的 AppName 需要与配置的 AppName 匹配。
2. 与直播录制功能类似，截图功能也需要将截图结果存放在媒体 Bucket 中。因此，截图功能也需要开通 OSS 服务，并且在媒体 Bucket 中设置输入媒体 Bucket。详情参见 媒体 Bucket 的增删改操作。
3. 配置截图功能时，ObjectName 可以定义为覆盖和非覆盖两种类型，其中覆盖是按照截图频率每次新生成的截图将覆盖之前生成的截图文件，因此仅会得到一张截图文件；而不覆盖则是每次新产生的截图将不覆盖之前生成

的截图文件，各截图文件使用 {Sequence} 区分，{Sequence} 将通过 1, 2, …, n 的方式表示。

4. 使用 API/SDK 配置截图配置是需要输入 OSS 的 Endpoint，OSS 的 Endpoint 为不包括 Bucket 名称的 OSS 访问地址，详情参见 OSS Endpoint 设置，并且这里请不要加 http:// 或者 https:// 协议头。
5. 使用 API/SDK 配置截图时 OverwriteOssObject 参数和 SequenceOss-Object 参数必须二者选一进行配置，暂时没有默认值设置。

3 种方式快速完成 OSS 迁移数据

简介：OSS 迁移数据有三种方式（ossutil、ossimport、在线迁移服务）用户可以根据场景自由选择。

功能描述

ossutil：是命令行迁移的工具，配置简单，可以适用在本地域名迁移到 OSS 上，支持多线程以及超时重试功能。但支持场景少可以用在本地迁移、OSS 之前迁移，而且迁移日志不太友好，分析起来比较麻烦，没有详细的任务进度。

ossimport：也是需要命令行进行配置，支持多个目录同时进行迁移，适合第三方云迁移到 OSS，本地数据迁移到 OSS、OSS 之前迁移。而且支持主备多机器同时迁移，增加迁移的带宽吞吐，可以有效的提高迁移效率。缺点是配置复杂，多个不同目录迁移时需要配置多个迁移文件。

在线迁移服务：这种方式几乎兼容了以上的所有优先，还可以限制每天迁移高峰期的流量，但缺点是客户数据如果放在本地的 IDC 机房那就无法使用了，要用 ossutil 或者 ossimport，本文主要介绍在线迁移的方式。

在线迁移

- 目前由于 OSS 数据迁移服务涉及到对目标的 OSS 要有很多 action 的 API 授权，为避免用户产生过多的学习成本，我们直接强制使用主账号进行迁移。
- 该服务正在公测中，目前仍在免费使用阶段；服务使用需要提前工单申请账号 UID 加入白名单。

在线迁移分类

离线迁移

这里是指的闪电立方硬盘数据 copy 的方式迁移到 OSS；适合用在专有云，以及海量 PB 级别数据想要快速迁移的需求。



闪电立方(海量数据迁移服务)

售卖模式

按量付费实例规格

按迁移数据量

仅支持数据迁移至阿里云

闪电立方型号

闪电立方mini

闪电立方1U(机架式)

闪电立方1U(6U机架式)

容量40TB，提供10天可用期，超出后按260元/天收取

云端存储类型

对象存储OSS

文件存储NAS

云上存储的区域

华东2（杭州）

华东1（杭州）

华东2（上海）

华南1（深圳）

美国（硅谷）

美国（弗吉尼亚）

马来西亚（吉隆坡）

澳大利亚（悉尼）

您的数据源类型

本地物理块设备

本地共享存储NAS

Hadoop(HDFS-NAS)

fastDFS

专有云OSS

Lustre（NAS协议）

您的交换机类型

千兆交换机

万兆交换机

交换机可用端口数

1

交换机接口类型

10G

电口

在线迁移

包含了第三方存储迁移到 OSS 以及 ECS 数据迁移到 OSS，具体配置方法如下。

创建迁移任务

如需更多帮助请参考产品手册

×

任务配置

性能调优

迁移数据地址

★ 任务名称

oss-test-lansfer

16/63

如果无可用数据（源/目的）地址，请您先[创建数据地址](#)

★ 源地址 ?

[oss] 迁移测试

▼

http://oss-cn-hangzhou-internal.aliyuncs.com:zyb-zhaou/

源地址选internal时，目的地址只能选同一个地域。如果要选不同地域的目的地址，请先选public的源地址。

★ 目的地址 ?

[oss] 目标bucket

▼

http://oss-cn-hangzhou-internal.aliyuncs.com:hanli-hangzhou/

迁移策略

迁移方式 ?

全量迁移

增量迁移

数据同步

全量数据迁移完成后任务将立即停止，不再对增量数据进行迁移。同任务多次提交全量迁移，仅迁移更新的数据

取消

下一步

联系我们

3. 任务类型说明

- 全量迁移：上传所有的源文件到 OSS。
- 增量迁移：上传前先 list 所有的源文件，比对哪些是已经上传过的，将不再重复上传。
- 数据同步：这里注意下，当部署的是 OSS 之间迁移的任务时，可以在同区域进行数据同步操作，定期的同步源 bucket 数据到目标 bucket。

同时还可以针对指定文件的时间进行数据迁移。

迁移策略

迁移方式 ?

全量迁移

增量迁移

数据同步

全量数据迁移完成后任务将立即停止，不再对增量数据进行迁移。同任务多次提交全量迁移，仅迁移更新的数据

迁移文件起点时间 ?

迁移全部

指定时间

4. 调优设置

用户配置好迁移体量和文件大致数量后，可以根据设置自动计算分配的工作线程数，同时用户也可以灵活的按时间段限流，或者不限流。

创建迁移任务

如需更多帮助请参考 [产品手册](#)

✕

数据预估



为保障顺利完成迁移任务，准确统计迁移进度和成功率，请尽量准确评估您的迁移存储量和迁移文件个数。[如何评估迁移数据量](#)

待迁移存储量

GB



待迁移文件个数

↑



流量控制

(每天)限流时间段

0点 3点 6点 9点 12点 15点 18点 21点 24点

最大流量(MB/s)

5

添加

联系我们

开始

结束

限流

操作

不设置限流

5. 使用注意

在创建迁移任务后，OSS 会去源拉个别文件进行测试，如果恰好源文件含有非法命名的 object 就会导致整个任务失败。

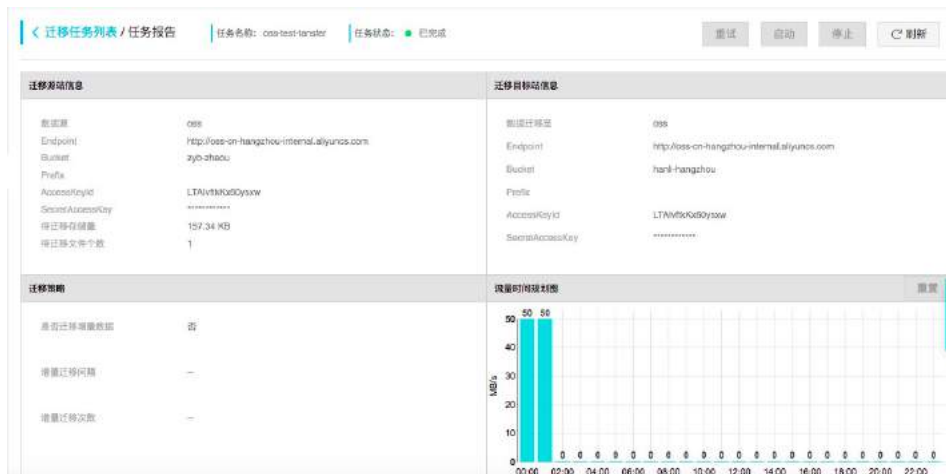
非法字符包含 "/" .. " 以及长度不能超过 1024 字节。



在线迁移管理

已经创建好的迁移任务，用户可以对其进行管理和监控。

- 进入到管理界面后我们可以看到迁移任务的整体监控。（流量，任务状态，迁移进展）



- 任务迁移过程中如果出现失败，用户可以进行重试。



- 任务迁移完成后，用户生成迁移报告，包含来整体的迁移实际数量，是否有报错，以及报错原因等信息；报告是保存在 OSS 上的。



FQA

1. 如果要使用迁移服务，子账号需要具备 OSS 管理权限 以及 MGW 的管理权限，授权地址。

服务开启方式

https://mgw.console.aliyun.com/?spm=a2c4g.11186623.2.12.5bf-6614cWHZzGe#/job?_k=6w2hbo

2. 跨域迁移的场景，需要提交功能单独申请，个人要提供内容合法性的生命。
3. 排查失败任务原因可以在控制台迁移任务内，生成对应任务报告。
4. 迁移文件量大、文件多时，OSS 要对源文件进行 list 后才会启动迁移，在 list 过程中比较耗时，如果看到迁移进度 0 的情况正常，只要没有迁移失败用户不用担心。

如何加强数据安全？3 种方式搞定

简介：加强 OSS 信息安全管控，多种解决方案实现数据安全。

OSS 数据安全

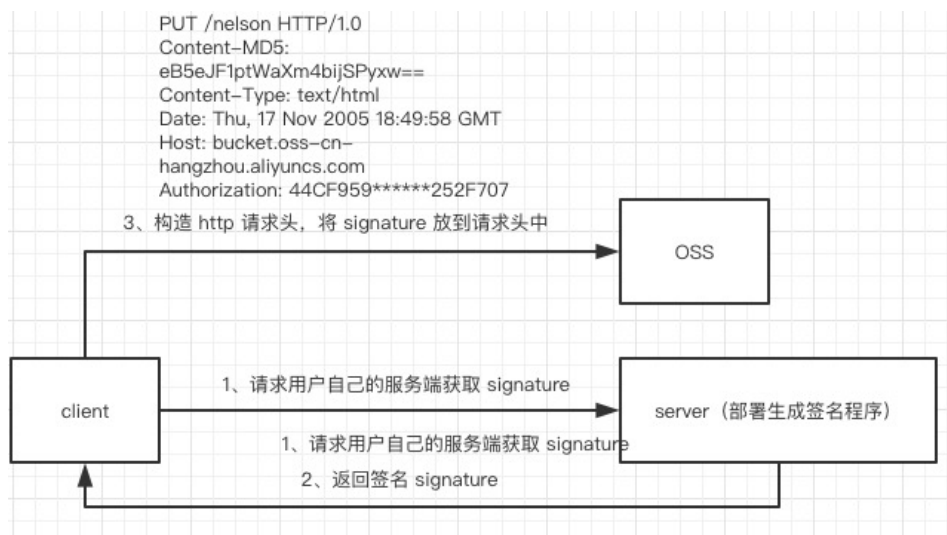
当前 OSS 保证数据安全的方式参考的方式有如下几种方式。

1. OSS 要设置为私有的避免公共读，或者公共读写。
2. 尽量不要使用传统的 AccesskeyID (AK)、AccesskeySecret (SK)，改用 STS token 的方式替代原来的教研方式。
3. 可以采用 OSS 内容加密，在鉴权的基础上双重加密，使用 KMS 对内容进行加密，但操作过程略微复杂。

使用场景

服务端存储 AKSK

用户自己有服务器，部署一套签名代码，使用 AK SK 生成鉴权的 signature 返回给客户端，端上利用 signature 构造 http 的请求头的方式来验签。

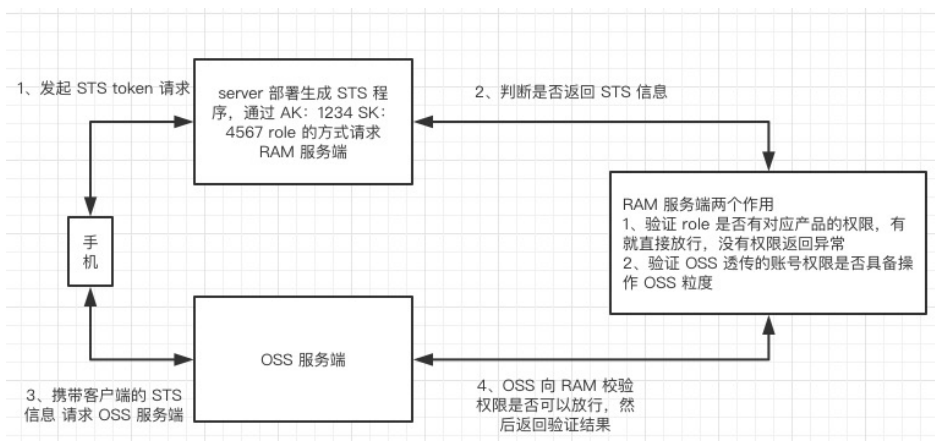
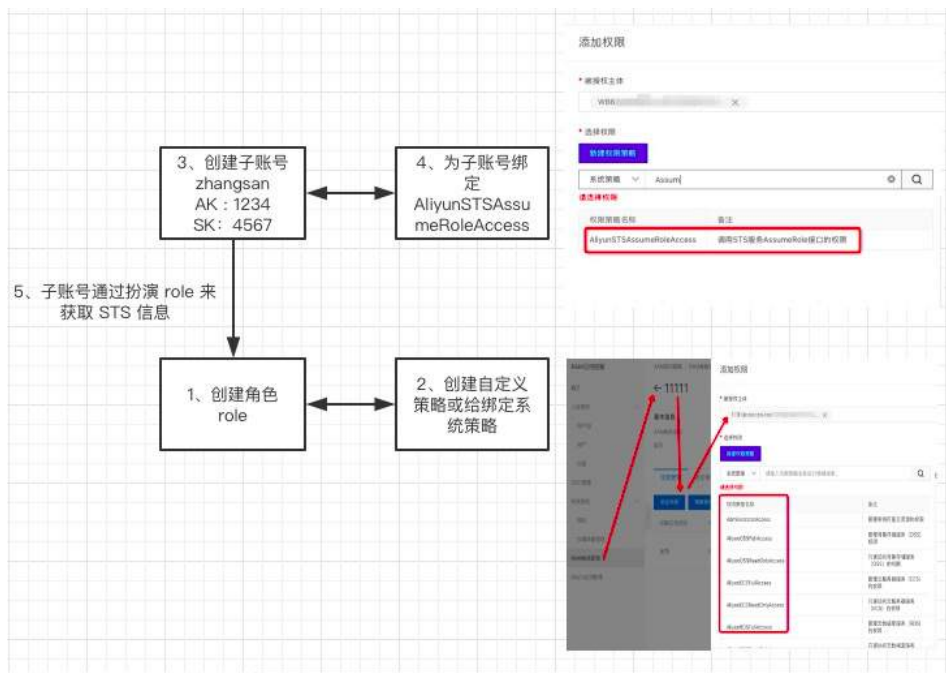


服务端生成 STS token

此类场景多用在移动端使用，客户端通过一个 https 地址请求到用户服务器，服务器上部署一个生成 STS token 的程序，收到移动端请求后请求 RAM 服务端生成 STS 信息。获取到 RAM 服务端返回的 STS.AK STS.SK STS.token 信息后，再返回给移动端使用。

需要注意用户服务器上也是用 AK SK 去申请的 STS，所以需要用户将生成一个子账号，然后配置好角色，将角色和权限绑定后，再通过子账号进行调用角色去生成 STS 信息。

移动端请求用户服务器尽量使用 HTTPS 协议，避免明文被劫持；服务端也可以针对移动端的请求做二次校验，比如客户端请求时携带一个 token，服务端校验通过再返回 STS 信息。



客户端拿到鉴权签名如何防泄露

header AK SK 签名

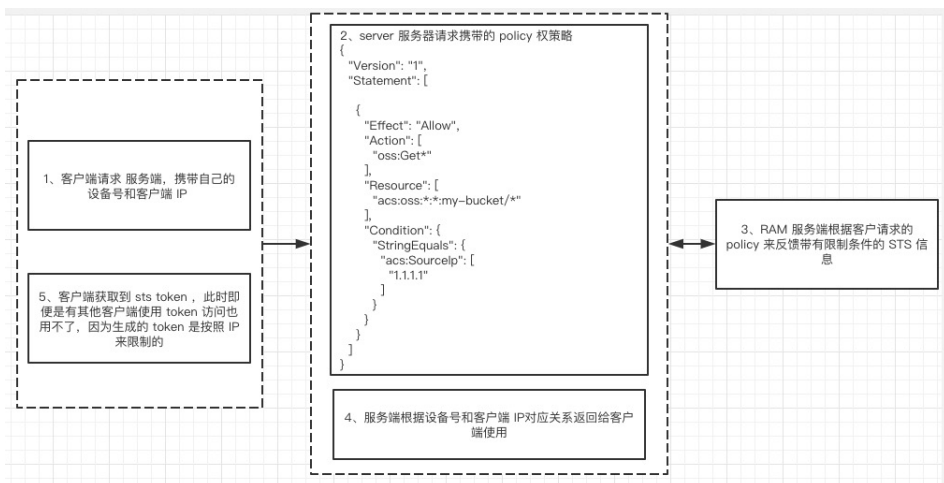
1. 服务端通过 header + AK SK 签名方式生成 signature 后，可以对 header

签名做二次加密。比如客户端请求服务端时携带设备号和时间戳信息，服务端通过设备号和时间戳对 signature 做加密，再返回给客户端。

2. 客户端拿到加密后的 signature，通过约定的解密算法将 signature 解析出来，这样可以避免 signature 被别人抓到，或者被反编译出来。
3. 通过客户端和服务端的传输可以用 https 方式避免被旁路劫持干扰。
4. header 签名拿到后，有效期是 15min，超过 15min 没有使用，signature 失效无法继续使用。

header STS 签名

1. 服务端通过 header + STS 签名方式生成 signature 后，在生成 STS 时可以限制只允许当前携带了设备号的 IP 来获取 signature，如果设备号和 IP 不匹配的情况，即便是其他客户获取到了 STS token 也无法上传。
2. 当 STS 暴露后，用户可以在 RAM 产品上将这个 STS 用的角色删除掉，重新配置一个角色即可，可以最小化降低影响。
3. 生成 STS 时的有效时间尽量不要那么长，控制在 900-3600s 之内，如果每个 STS token 都是 3600s 失效时间会带来一定的业务风险。



深度剖析鉴权 API，验签不再是难题

简介：出现 signature 一般出现客户端自签名调 API 的操作中，signature 的计算稍微复杂点，建议最好用 SDK 来替代计算的过程和多样性。如果业务强需求，先要读懂如果计算 signature。

背景

使用过阿里云 OSS 存储 API 的用户都知道，如果 OSS 是私有的权限，需要进行验签才能访问。验签过程要求客户端请求的 http request header 中有一个 Authorization (鉴权) 的 header，计算的复杂性和带来的很多问题让客户头痛不已，尤其 OSS Authorization 头的计算，今天带这大家剖析下鉴权的 API。

使用规范预热

官网鉴权文章：

[header 签名](#)

[URL 签名](URL 中携带签名。

https://help.aliyun.com/document_detail/31952.html

[PutObject 规范](#)

签名区别

Header	URL
不支持设置 expires，但是要求请求时间不能超过 15min	支持设置 expires
常用 method GET、POST、PUT	常用 method GET、PUT
Date 时间是 GMT 格式	Date 替换成 expires 变成时间戳
signature 不需要 URL encode	signature 需要 URL encode

鉴权名词

AccessKey

简称 AK，访问云产品的凭证，类似宝藏的锁，可以是主账号或者 RAM 子账号的。

Access Key Secret

简称 SK，访问云产品的密钥，类似宝藏的钥匙，可以是主账号或者 RAM 子账号的。

Authorization

Header 来包含签名 (Signature) 信息，表明这个消息已被授权。

Signature

经过各种计算得到的鉴权指纹信息。

CanonicalizedOSSHeaders

访问 OSS 时，用户想要加的自定义头，必须以 "x-oss-" 为头的前缀。如果使用，也必须要加到 Signature 中计算。

CanonicalizedResource

访问 OSS 的资源 object，结构是 /bucket/ object，如下：

- /zhangyibo/Japan/video/tokhot.avi 将视频上传到 bucket 为 zhangyibo 的虚拟目录 Japan 下面，命名为 tokhot.avi，可以是 PUT / GET 等操作。
- /zhangyibo/ 针对 bucket 是 zhangyibo 进行的操作，可以是 PUT / GET 等操作。

主账号 AK SK 获取方式



子账号 AK SK 获取方式

进入到 RAM 访问控制台，找到对应的子账号。



计算鉴权

当客户通过 header 或者 URL 中自签名计算 signature 时，经常会遇到计算签名失败 “The request signature we calculated does not match the signature you provided”，可以参考以下 demo 演示了如何调用 API 自签名时上传 Object 到 OSS，注意签名和 header 加入的内容。

使用方法

```
$PSA1#: python Signature.py -h
Usage: beiwo.py [options]
```

Options:

-h, --help	show this help message and exit	
-i AK	Must fill in Accesskey	访问云产品的 Accesskey
-k SK	Must fill in AccessKeySecret	访问云产品的 Accesskey Secret
-e ED	Must fill in endpoint	OSS 的 endpoint 地理信息
-b BK	Must fill in bucket	OSS bucket
-o OBJECTS	File name uploaded to oss	上传的 object 名称
-f FI	Must fill localfile path	本地文件的名称

```
#!/usr/bin/env python
#Author: hanli
#Update: 2018-09-29

from optparse import OptionParser
import urllib, urllib2
import datetime
import base64
import hmac
import sha
import os
import sys
import time

class Main():

    # Initial input parse

    def __init__(self,options):

        self.ak = options.ak
        self.sk = options.sk
        self.ed = options.ed
        self.bk = options.bk
        self.fi = options.fi
        self.oj = options.objects
        self.left = '\033[1;31;40m'
        self.right = '\033[0m'
        self.types = "application/x-www-form-urlencoded"
        self.url = 'http://{0}.{1}/{2}'.format(self.bk,self.ed,self.oj)

    # Check client input parse

    def CheckParse(self):

        if (self.ak and self.sk and self.ed and self.bk and self.oj and self.fi) !=
None:
            if str(self.ak and self.sk and self.ed and self.bk and self.oj and self.
```



```
fi):
    self.PutObject()
else:
    self.ConsoleLog("error", "Input parameters cannot be empty")

# GET local GMT time

def GetGMT(self):

    SRM = datetime.datetime.utcnow()
    GMT = SRM.strftime('%a, %d %b %Y %H:%M:%S GMT')

    return GMT

# GET Signature

def GetSignature(self):

    mac = hmac.new("{0}".format(self.sk), "PUT\n\n{0}\n{1}\n/{2}/{3}".
format(self.types, self.GetGMT(), self.
bk, self.oj), sha)
    Signature = base64.b64encode(mac.digest())

    return Signature

# PutObject

def PutObject(self):

    try:
        with open(self.fi) as fd:
            files = fd.read()
    except Exception as e:
        self.ConsoleLog("error", e)

    try:
        request = urllib2.Request(self.url, files)
        request.add_header('Host', '{0}.{1}'.format(self.bk, self.ed))
        request.add_header('Date', '{0}'.format(self.GetGMT()))
        request.add_header('Authorization', 'OSS {0}:{1}'.format(self.ak, self.
GetSignature()))
        request.get_method = lambda: 'PUT'
        response = urllib2.urlopen(request, timeout=10)
        fd.close()
        self.ConsoleLog(response.code, response.headers)
    except Exception, e:
        self.ConsoleLog("error", e)

# output error log
```

```
def ConsoleLog(self, level=None, mess=None):

    if level == "error":
        sys.exit('{0} [ERROR:] {1}{2}'.format(self.left, self.right, mess))
    else:
        sys.exit('\nHTTP/1.1 {0} OK\n{1}'.format(level, mess))

if __name__ == "__main__":

    parser = OptionParser()
    parser.add_option("-i", dest="ak", help="Must fill in Accesskey")
    parser.add_option("-k", dest="sk", help="Must fill in AccessKeySecrety")
    parser.add_option("-e", dest="ed", help="Must fill in endpoint")
    parser.add_option("-b", dest="bk", help="Must fill in bucket")
    parser.add_option("-o", dest="objects", help="File name uploaded to oss")
    parser.add_option("-f", dest="fi", help="Must fill localfile path")

    (options, args) = parser.parse_args()
    handler = Main(options)
    handler.CheckParse()
```

请求头

```
PUT /yuntest HTTP/1.1
Accept-Encoding: identity
Content-Length: 147
Connection: close
User-Agent: Python-urllib/2.7
Date: Sat, 22 Sep 2018 04:36:52 GMT
Host: yourBucket.oss-cn-shanghai.aliyuncs.com
Content-Type: application/x-www-form-urlencoded
Authorization: OSS B0g3mdt:1NCA4L0P43Ax
```

响应头

```
HTTP/1.1 200 OK
Server: AliyunOSS
Date: Sat, 22 Sep 2018 04:36:52 GMT
Content-Length: 0
Connection: close
x-oss-request-id: 5BA5C6E4059A3C2F
ETag: "D0CAA153941AAA1CBDA38AF"
x-oss-hash-crc64ecma: 8478734191999037841
Content-MD5: 0MqhU5QbIp3Ujqghy9o4rw==
x-oss-server-time: 15
```

注意事项

1. Signature 中所有加入计算的参数都要放在 header 中，保持 header 和 Signature 一致。
2. PUT 上传时，Signature 计算的 Content-Type 必须是 application/x-www-form-urlencoded。
3. 通过 header 方式进行签名认证时无法设置过期时间。目前只有 SDK、URL 签名支持设置过期时间。
4. 用户想要保证文件一致性，可以在请求头增加 Content-MD5，但是不注意的人就会忘记加了，补充如下：根据协议 RFC 1864 对消息内容（不包括头部）计算 MD5 值获得 128 比特位数字，对该数字进行 base64 编码为一个消息的 Content-MD5 值，并且 MD5 是大写。
5. 如果用户想要单独加项目 CanonicalizedOSSHeaders 一定要记得不仅在 Header 中加，你的 hmac 计算时也要加。

```
hmac.new("5Lic5Lqs5LiA54K56Y095LiN540t", "PUT\n\napplication/x-www-form-urlencoded\nSun, 02 Sep 2018 03:20:05 GMT\nx-oss-video:tokhot.avi/zhangyibo/tokhot.avi", sha)````
```

6. 如果遇到 client 计算的 MD5 和 Server 不一致的情况请直接使用 HTTPS 传输，很可能中间的网络设置有故障或者劫持时导致内存被篡改，只要将 url 改为 https:// 就是启动 HTTPS 协议上传 / 下载了。

常见案例

通过微信小程序请求 OSS 返回签名失败，通过浏览器正常

1. 只要通过浏览器访问，鉴权通过就证明 OSS 的签名校验是正常的没有问题，可以先排除掉 OSS 端。

2. 客户端一定要在微信小程序上部署 HTTP 抓包，对后续分析很重要，抓包中可以看到所有的请求头和请求参数。
3. 通过浏览器访问时的 HTTP 抓包。



```
GET /1530903286-055AccessKeyId=...&Expires=1540915205&Signature=XDdcOn8%... HTTP/1.1
Accept: */*
C-Seq: 123
Cache-Control: no-cache
Connection: Keep-Alive
Content-Type: application/octet-stream
Host: ...
Range: bytes=0-262143
User-Agent: MicroMessenger Client

HTTP/1.1 403 Forbidden
Server: Tengine
Content-Type: application/xml
Content-Length: 785
Connection: keep-alive
Date: Fri, 12 Oct 2018 09:47:19 GMT
x-oss-request-id: 5BC06DA78E...
x-oss-server-time: 0
Via: cache2.l2cm10-l[27,403-1280,M], cache27.l2cm10-l[29,0], cache2.cn1453[239,403-1280,M], cache8.cn1453[243,0]
X-Swift-Error: orig response 4XX error
All-Swift-Global-Savetime: 1539337639
X-Cache: MISS TCP_MISS dirn:-2:-2
X-Swift-Savetime: Fri, 12 Oct 2018 09:47:19 GMT
X-Swift-CacheTime: 0
X-Swift-Error: orig response 4XX error
Timing-Allow-Origin: *
EagleId: 7488861c15393376395951725e
```

结论：

1. 通过 403 和 200 的抓包反复对比发现，通过小程序发出的 HTTP 请求和浏览器发起的 HTTP 请求的 URL、signature、expires 都一样，唯一的区别就是微信小程序携带了 Content-type，而通过 Chrom 的请求是没有携带 Content-type，怀疑矛头指向了这里。
2. 经过代码确认，发现 signature 计算时是没有包含 Content-tpye 头的，而小程序发起的请求携带的 Content-tpye，OSS 收到后会按照携带了 Content-tpye 去计算 signature，所以每次计算都不一样。

结尾：

遇到类似问题，抓包是最能快速看到问题的。同时也必须要了解下 OSS 请求 header 中携带了 Content-tpye，那么 signature 计算就要加上 Content-tpye，保持一致。

多个 OSS SDK 测试，在 CDN 结合 OSS 场景时，客户端使用 CDN 域名计算 signature，发起 HEAD 请求，OSS 收到后返回 403

```

Loaded assembly: Microsoft.GeneratedCode [External]
System.Net.WebException: The remote server returned an error: (403) Forbidden.
  at Aliyun.OSS.Common.Communication.ServiceResponse.EnsureSuccessful() [0x00021] in <d8db38433d62487181c
  at Aliyun.OSS.Common.Handlers.ErrorResponseHandler.ErrorHandle (Aliyun.OSS.Common.Communication.ServiceR
  at Aliyun.OSS.Common.Handlers.ErrorResponseHandler.Handle (Aliyun.OSS.Common.Communication.ServiceRespon
  at Aliyun.OSS.Common.Communication.ServiceClient.HandleResponse (Aliyun.OSS.Common.Communication.Service
  at Aliyun.OSS.Common.Communication.ServiceClient.Send (Aliyun.OSS.Common.Communication.ServiceRequest re
  at Aliyun.OSS.Common.Communication.RetryableServiceClient.SendImpl (Aliyun.OSS.Common.Communication.Serv
  at Aliyun.OSS.Common.Communication.RetryableServiceClient.Send (Aliyun.OSS.Common.Communication.ServiceR
  at Aliyun.OSS.Commands.OssCommand.Execute () [0x00014] in <d8db38433d62487181cd2ba5c760defd>:0
  at Aliyun.OSS.Commands.OssCommand`1[T].Execute () [0x00000] in <d8db38433d62487181cd2ba5c760defd>:0
  at Aliyun.OSS.OssClient.GetObjectMetadata (System.String bucketName, System.String key) [0x0001f] in <d8
  at Aliyun.OSS.Samples.Program.Main (System.String[] args) [0x00073] in <774abbba5f5443709aa5fc54070462a5

```

出现这个问题不区分什么 SDK 都会出现，问题原因是由于客户端发起的 HEAD 请求在通过 CDN 回原到 OSS 时，CDN 回原是用 GET 请求，而 OSS 收到时就用 GET 请求方式去计算签名，得到的结果肯定和客户端计算不一致，可以升级到阿里云 CDN 处理。以上分析只适合上述场景。

问题可以通过 tcpdump 抓包或者 Wireshark 对比一下即可知道。

C SDK 安装这些坑，你踩到了吗？

背景

由于很多人对 SDK 的安装和系统依赖的环境变量不是很熟悉，导致很熟悉，浪费不必要的时间，而且导致环境变量引入也出现异常。特此写了一篇从安装到遇坑的过程给大家。

操作环境

Linux Centos 6.9 64 位系统。

预先安装好的库

1. glibc-2.14 (mxml 库需要依赖这个库):
 - 先看下 `strings /lib64/libc.so.6 | grep GLIBC` 是否有 GLIBC-2.14 或者以上。
如果没有的话，请先编译 GLIBC-2.14 的安装包。

```
[root@i2wz91s6lw79vfn3fgzlazZ oss_c_sdk]# strings /lib64/libc.so.6 | grep GLIBC
GLIBC_2.2.5
GLIBC_2.2.6
GLIBC_2.3
GLIBC_2.3.2
GLIBC_2.3.3
GLIBC_2.3.4
GLIBC_2.4
GLIBC_2.5
GLIBC_2.6
GLIBC_2.7
GLIBC_2.8
GLIBC_2.9
GLIBC_2.10
GLIBC_2.11
GLIBC_2.12
GLIBC_2.13
GLIBC_2.14
GLIBC_PRIVATE
[root@i2wz91s6lw79vfn3fgzlazZ oss_c_sdk]#
```

- 下载 glibc <https://zhangyb.oss-cn-shanghai.aliyuncs.com/OSS-C-SDK/glibc-2.14.tar.gz> 下载不了请联系本人。
- `tar xvf glibc-2.14.tar.gz && cd glibc-2.14 && mkdir build && cd build && ../configure --prefix=/usr/local/glibc-2.14 && make -j4 && make install && cp /usr/local/glibc-2.14/lib/libc-2.14.so /lib64/libc-2.14.so && mv /lib64/libc.so.6 /lib64/libc.so.6.bak && LD_PRELOAD=/lib64/libc-2.14.so ln -s /lib64/libc-2.14.so /lib64/libc.so.6。`

2. 安装 cmake 库:

- `sudo yum install cmake。`

3. 安装第三方库文件:

- `sudo yum install curl-devel apr-devel apr-util-devel。`

4. 安装 mxml 库:

- http://docs.aliyun.cn-hangzhou.oss.aliyun-inc.com/assets/attach/32132/cn_zh/1501596081318/mxml-2.9-1.x86_64.rpm?spm=a2c4g.11186623.2.6.I37Y6M&file=mxml-2.9-1.x86_64.rpm。
- `rpm -ivh mxml-2.9-1.x86_64.rpm --nodeps --force。`

```
[root@iZwz91s6lw79vfn3fgzlazZ oss_c_sdk]# rpm -qF mxml
mxml-2.9-1.x86_64
[root@iZwz91s6lw79vfn3fgzlazZ oss_c_sdk]#
```

安装 OSS-C-SDK

- SDK 3.5，官网随时更新，请以官网为准 http://docs.aliyun.cn-hangzhou.oss.aliyun-inc.com/assets/attach/32131/cn_zh/1501595738954/aliyun-oss-c-sdk-3.5.0.tar.gz?spm=a2c4g.11186623.2.4.Bf6aUL&file=aliyun-oss-c-sdk-3.5.0.tar.gz。

- `tar -xf SDK.tar.gz && cd SDK && cmake . && make && make install` 安装好 SDK 后，会自动 `/usr/local/include/` 下创建好 `oss_c_sdk` 的目录，并引入需要依赖的 SDK 库文件。

```
[root@iZwz91s6lw79vfn3fgzlazZ ~]# cd /usr/local/include/oss_c_sdk/
[root@iZwz91s6lw79vfn3fgzlazZ oss_c_sdk]# ls
aos_buf.h      aos_fstack.h  aos_log.h      aos_transport.h  oss_auth.h      oss_util.h
aos_crc64.h    aos_http_io.h aos_status.h    aos_util.h        oss_define.h     oss_xml.h
aos_define.h   aos_list.h    aos_string.h    oss_api.h         oss_resumable.h
[root@iZwz91s6lw79vfn3fgzlazZ oss_c_sdk]#
```

测试 SDK demo

注意 demo 中的动态链接库都是查找的默认安装路径，请先用命令搜索一下是否包含了以下路径。

- `g++ -print-prog-name=ccl -v`。
- `gcc -print-prog-name=ccl -v`。
- 下载 demo，解压后找到对应的系统的路径，进行测试 <http://docs-alibaba.com/oss/oss-c-sdk-demo.tar.gz?spm=a2c4g.11186623.2.14.AsMiZ9&file=aliyun-oss-c-sdk-demo.tar.gz>。
- 编译 `gcc -Wall -O -g -I /usr/local/include/oss_c_sdk -I /usr/include/apr-1 -c main.c -o main.o && gcc main.o -o main -lpthread -L /usr/local/lib -loss_c_sdk -lcurl -lxml -L /usr/local/apr/lib -lapr-1 -laprutil-1`。
- `./main`。

3 条线索揭秘 OSS C SDK 问题核心

概述

客户使用 OSS C SDK 3.5 版本，通过 `get_object_to_local_file` 方法直接下载 OSS 文件到本地，测试过程返回 403 签名不对。

搜集信息

挖掘有价值的信息很重要，通过基础信息可以筛选下客户上传日志的详细描述，通过堆栈报错可以找到客户大概原因。

- OSS response header 中包含 `requestID`，记录客户端请求到 OSS 的详细日志。
- 堆栈完整报错信息。

分析 `requestID` 对应的服务端错误日志

服务端返回 4xx 2xx 3xx 500 的状态码都会返回 `requestID`；通过 `requestID` 我们可以过滤到服务的错误日志如下。

```
java
Method:HEAD      Host:oss-cn-shanghai.aliyuncs.com    URI:/
vod%2Fplat%2Fupdate%2Fhaimu_21_9%2FBasketball.zip

StringToSign:HEAD\n\nWed, 06 Nov 2019 07:55:44 GMT\n/vod/plat/update/
haimu_21_9/Basketball.
zip
UserSignature:LIMmOQ/5TLs2Gk24MuNVRl+Lu0Q=
OssServerSignature:P+YZauMD+fRtZjeMWkauuN6A4eU=

ErrorCode:SignatureDoesNotMatch    ErrorMsg:The request signature we
```

```
calculated does not match
the signature you provided. Check your key and signing method.
```

构建CanonicalizedResource的方法

用户发送请求中想访问的OSS目标资源被称为CanonicalizedResource，它的构建方法如下：


1. 将CanonicalizedResource置为空字符串 ""。
2. 设置要访问的OSS资源 /BucketName/ObjectName。如果仅有BucketName而没有ObjectName，则 CanonicalizedResource为"/BucketName/"，如果既没有BucketName也没有ObjectName，则CanonicalizedResource为 "/"。
3. 如果请求的资源包括子资源（SubResource），那么将所有的子资源按照字典序，从小到大排列并以 & 为分隔符生成子资源字符串。在CanonicalizedResource字符串尾添加 ? 和子资源字符串。此时的CanonicalizedResource为 /BucketName/ObjectName?acl&uploadId=UploadId。

- 通过日志可以明显看出客户端签名和服务端签名不一致导致校验失败；
- 根据签名算法可以知道这算计算签名的参数中 /vod/plat/update/haimu_21_9/Basketball.zip 其中 vod 是 bucket 的位置。
- Host: oss-cn-shanghai.aliyuncs.com 这个位置是错误，因为正常的直接访问 OSS，Host 正确写法应该是 bucket.oss-cn-region.aliyuncs.com。（region 替换成 bucket 所在地区）

问题分析

线索一

已经知道问题出现在签名不对，而且 Host 也不对；问题出现在这里，但是如果 Host，但 OSS 还能识别出 bucket，这很奇怪；于是和用户沟通，vod 并不是用户的 bucket，那这个桶是怎么获取到的呢？遂让用户在端上进行 Wireshark 抓包，得到报文。



```
Host: oss-cn-shanghai.aliyuncs.com
Accept: */*
Date: Wed, 06 Nov 2019 09:05:47 GMT
Authorization: OSS LTAIfHQ8Bp2AZEbK:CScXue/UTauZ2jG1/eXMWhj/1+E=

HTTP/1.1 403 Forbidden
Server: AliyunOSS
Date: Wed, 06 Nov 2019 09:05:48 GMT
Content-Type: application/xml
Content-Length: 838
Connection: keep-alive
x-oss-request-id: 5DC28CECD6A1093932626746
x-oss-server-time: 12

GET /vod%2Fplat%2Fupdate%2Fhaimu_21_9%2FBasketball.zip HTTP/1.1
User-Agent: aliyun-sdk-c/3.5.0(Compatible Unknown)
Host: oss-cn-shanghai.aliyuncs.com
Accept: */*
Date: Wed, 06 Nov 2019 09:05:52 GMT
Authorization: OSS LTAIfHQ8Bp2AZEbK:w+BkQQgy26jI5Vv8cxwonRk6llk=

HTTP/1.1 403 Forbidden
Server: AliyunOSS
Date: Wed, 06 Nov 2019 09:05:53 GMT
Content-Type: application/xml
Content-Length: 834
Connection: keep-alive
x-oss-request-id: 5DC28CF1D6A1093932F38F46
x-oss-server-time: 0
```

1. 从报文中可以看到 Host 是客户端传的时候就没有 bucket。
2. 而 vod 是用户 objectkey 文件前缀而已。
3. 通过上述几点可以知道问题一定是用户代码上哪里出现的变量写错或者用法不对。

线索二

请用户在本机 debug，将关键签名的变量信息打印出来看是否完整。

关键信息位置

```

2 const char BUCKET_NAME[] = "hai-";
3
4
5 #if 0
6 const char OSS_ENDPOINT[] = "http://.s";
7 const char ACCESS_KEY_ID[] = "aaa";
8 const char ACCESS_KEY_SECRET[] = "aaa";
9 #else
10 const char OSS_ENDPOINT[] = "oss-cn-.aliyuncs.com";
11 const char ACCESS_KEY_ID[] = "aaa";
12 const char ACCESS_KEY_SECRET[] = "aaa";
13 #endif
14
15 void init_sample_config(oss_config_t *config, int is_cname)
16 {
17     aos_str_set(&config->endpoint, OSS_ENDPOINT);
18     aos_str_set(&config->access_key_id, ACCESS_KEY_ID);
19     aos_str_set(&config->access_key_secret, ACCESS_KEY_SECRET);
20     config->is_cname = is_cname;
21 }
22
23 void init_sample_request_options(oss_request_options_t *options, int is_cname)
24 {
25     options->config = oss_config_create(options->pool);
26     init_sample_config(options->config, is_cname);
27     options->ctl = aos_http_controller_create(options->pool, 0);
28 }
29
30 int get_object_to_local_file(char *bucketname, char *objectname, char *filename)
31 {
32     aos_pool_t *p = NULL;
33     aos_string_t bucket;
34     aos_string_t object;
35     oss_request_options_t *options = NULL;
36     aos_table_t *headers = NULL;
37     aos_table_t *params = NULL;
38     aos_table_t *resp_headers = NULL;
39
40     // ...
41 }

```

Diagram illustrating the mapping of variables to their values in the code:

- `OSS_ENDPOINT` is mapped to `oss-cn-.aliyuncs.com`.
- `bucketname` is mapped to `hai-`.
- `objectname` is mapped to `vod/x/y/z/xxx.zip`.
- `filename` is mapped to `vod/x/y/z/xxx.zip`.

用户自己 debug 出来变量的位置

```

bucketname = ha-
objectname = vod/plat/update/haimu_21_9/Basketball.zip
filename = Basketball.zip

```

通过客户 debug 看到变量都正确，为什么还会出现 bucket “丢失” 的情况呢。

线索三

分析源码

```

int get_object_to_local_file(char *bucketname, char *objectname, char
*filename)
{
    aos_pool_t *p = NULL;

```

```

aos_string_t bucket;
aos_string_t object;
oss_request_options_t *options = NULL;
aos_table_t *headers = NULL;
aos_table_t *params = NULL;
aos_table_t *resp_headers = NULL;
aos_status_t *s = NULL;
aos_string_t file;
int is_cname = 1;

if (bucketname == NULL || objectname == NULL || filename == NULL)
    return -1;

aos_pool_create(&p, NULL);
options = oss_request_options_create(p);
init_sample_request_options(options, is_cname);
aos_str_set(&bucket, bucketname);
aos_str_set(&object, objectname);
headers = aos_table_make(p, 0);
aos_str_set(&file, filename);

s = oss_get_object_to_file(options, &bucket, &object, headers,
    params, &file, &resp_headers);
if (aos_status_is_ok(s))
{
    printf("get object to local file succeeded\n");
}
else
{
    printf("get object to local file failed\n");
    aos_pool_destroy(p);
    return -2;
}
aos_pool_destroy(p);
return 0;
}

```

——> 从源码中可以，getobject 下载时形参都是传进来的，既然客户 debug 的变量都是正确的，那么肯定不是传进来变量，问题一定是方法内的常量导致；对比官方源码：

```

void get_object_to_local_file()
{
    aos_pool_t *p = NULL;
    aos_string_t bucket;
    char *download_filename = "get_object_to_local_file.txt";

```

```

aos_string_t object;
int is_cname = 0;
oss_request_options_t *options = NULL;
aos_table_t *headers = NULL;
aos_table_t *params = NULL;
aos_table_t *resp_headers = NULL;
aos_status_t *s = NULL;
aos_string_t file;

aos_pool_create(&p, NULL);
options = oss_request_options_create(p);
init_sample_request_options(options, is_cname);
aos_str_set(&bucket, BUCKET_NAME);
aos_str_set(&object, OBJECT_NAME);
headers = aos_table_make(p, 0);
aos_str_set(&file, download_filename);

s = oss_get_object_to_file(options, &bucket, &object, headers,
                          params, &file, &resp_headers);
if (aos_status_is_ok(s)) {
    printf("get object to local file succeeded\n");
} else {
    printf("get object to local file failed\n");
}

aos_pool_destroy(p);
}

```

——> 从源码和用户的源码对比发现有个 `is_cname` 的参数，这个参数用户端是 1，官网的源码是 0，参数的含义，是否启用 `is_cname` 方式上传，`is_cname` 简称别名，比如用户给 `bucket` 绑定一个备案的域名后，那个域名就是 `is_cname`；我们追下 `is_cname` 的用法。

```

if (options->config->is_cname ||
    is_valid_ip(raw_endpoint_str))
{
    req->host = apr_psprintf(options->pool, "%.*s",
                            raw_endpoint.len, raw_endpoint.data);
    req->uri = apr_psprintf(options->pool, "%.*s", bucket->len,
                            bucket->data);
} else {
    req->host = apr_psprintf(options->pool, "%.*s.%.*s",
                            bucket->len, bucket->data,
                            raw_endpoint.len, raw_endpoint.data);
    req->uri = apr_psprintf(options->pool, "%s", "");
}

```

——> 从这端代码中可以看到，启用了 cname 后，Host 直接取的 endpoint 值；基本上找到了和用户用了 cname 有关系。

- 经过后台分析发现用户端并没有绑定 oss+ 域名 的映射关系，而 SDK 在启动了 cname 上传，把 endpoint 当做完成的域名，没有把用户的 bucket name 拼到 Host 中，认为启用 cname 后，endpoint 就是完成域名。（设计就是这样并不是缺陷）
- 让用户把 cname 改为 0，采用直接的方式，这样 SDK 会把 bucketname 和 endpoint 拼成一个 完成域名再上传。

总结

该问题核心在于

1. 要对 OSS SDK 的用法熟悉，知道如何进行本地对比用户测试。
2. 要清楚排查思路知道搜集哪些有价值的信息。
3. 层级递进，从签名算法着眼，推测到是代码中的使用。
4. 善用 tcpdump/wireshark 抓包分析请求报文。

Andorid SDK 搭建还有疑问？6 步流程搞定

背景

很多小伙伴对 Android 的 SDK 搭建熟悉，但是对于上云的 OSS Android SDK 使用有些疑问，今天从环境搭建到客户端使用全面给大家梳理一遍个人的使用总结。

分解

- 下载 jdk 和 Android studio。
- 安装 jdk。
- 配置环境变量。
- 安装 Android Studio。
- 配置 Android Studio。

下载工具

- [JDK](#)
- [Android Studio](#)

安装

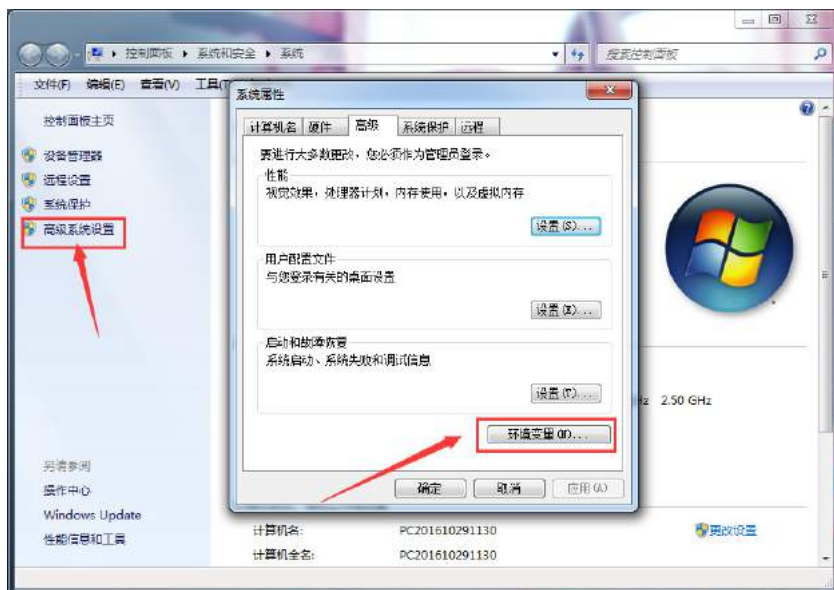
JDK



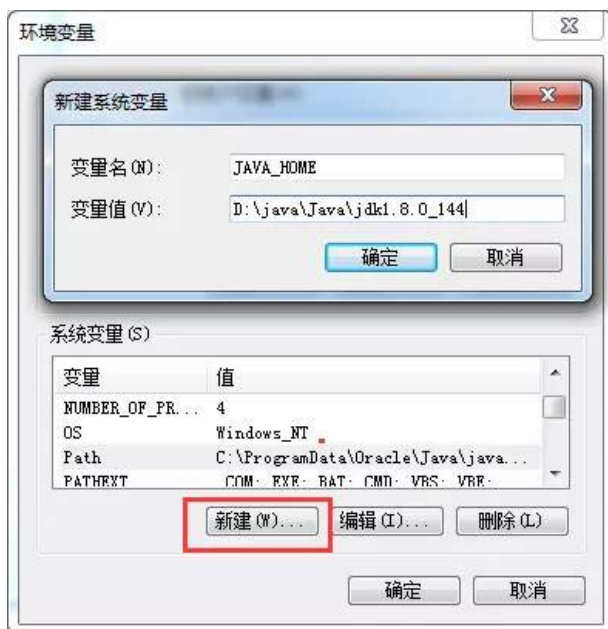
典型下一步的操作



环境变量配置



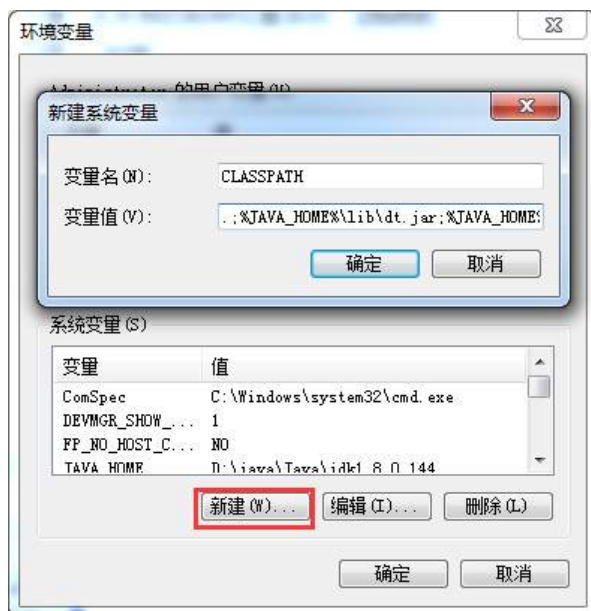
JAVA_HOME



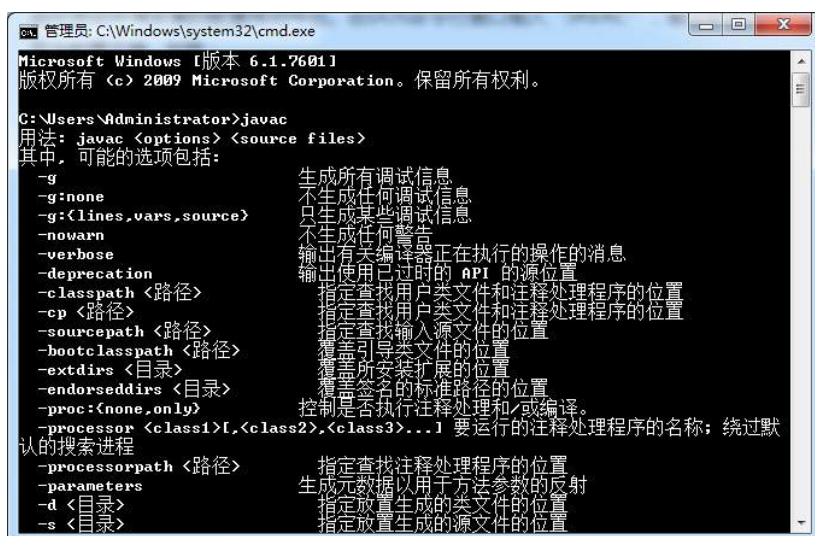
编辑 Path



CLASSPATH 变量



测试环境变量是否配置成功“开始”-》输入 -》“cmd” 打开命令行窗口输入 javac

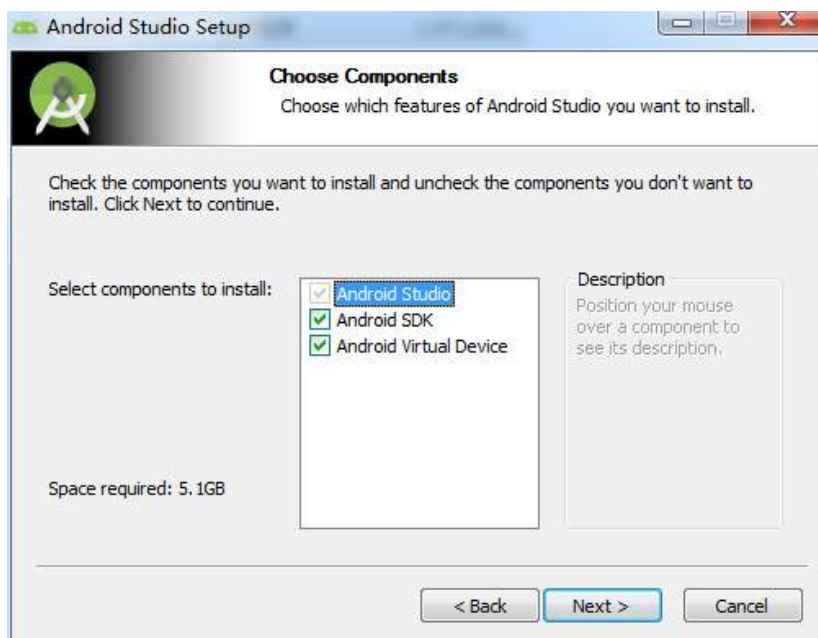


```

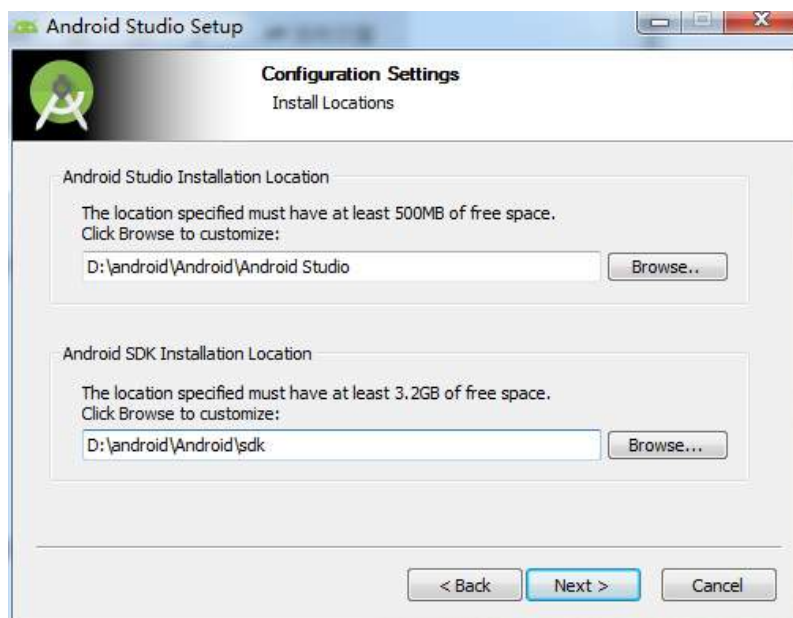
C:\Windows\system32\cmd.exe
Microsoft Windows [版本 6.1.7601]
版权所有 (c) 2009 Microsoft Corporation。保留所有权利。

C:\Users\Administrator>javac
用法: javac <options> <source files>
其中, 可能的选项包括:
-g          生成所有调试信息
-g:none     不生成任何调试信息
-g:<lines,vars,source> 只生成某些调试信息
-nowarn     不生成任何警告
-verbose    输出有关编译器正在执行的操作的消息
-deprecation 输出使用已过时的 API 的源位置
-classpath <路径> 指定查找用户类文件和注释处理程序的位置
-cp <路径> 指定查找用户类文件和注释处理程序的位置
-sourcepath <路径> 指定查找输入源文件的位置
-bootclasspath <路径> 指定类引导类文件的位置
-extdirs <目录> 指定安装扩展的位置
-endorseddirs <目录> 指定签名标准路径的位置
-processor <class1[,<class2>...<class3>...<classN>] 控制是否执行注释处理和/或编译。
              要运行的注释处理程序的名称; 绕过默认
              的搜索进程
-processorpath <路径> 指定查找注释处理程序的位置
-parameters 生成元数据以用于方法参数的反射
-d <目录> 指定放置生成的类文件的位置
-s <目录> 指定放置生成的源文件的位置
    
```

安装 Android Studio



- 建议典型安装, 如果不是很懂安装过程, 不要建议更改配置。



- 上面是 Android Studio 的安装目录 下面是 sdk 的目录, 自行选择即可。

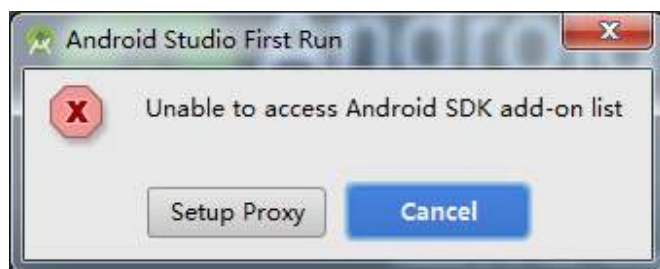


Android Studio 配置

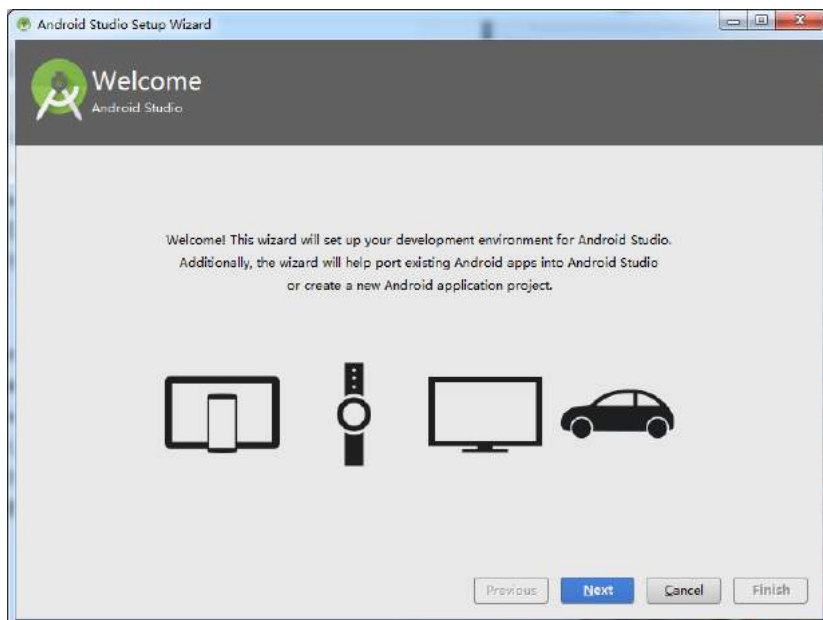
- 加载本地配置信息提示。



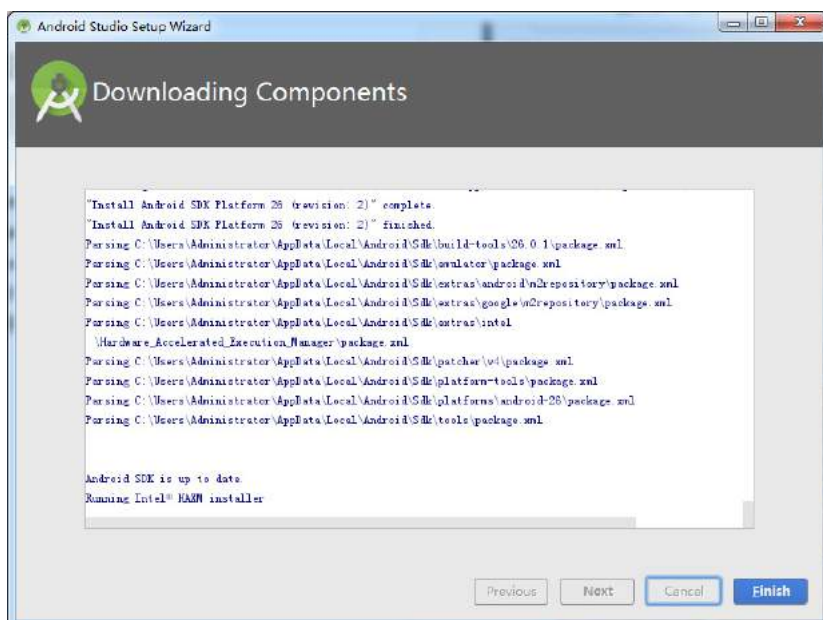
- 刚开始打开的时候会问我们是否加载本地的配置信息，就是 Android Studio 的配置环境信息等，如果你以前没有用过，忽略掉就好了。直接按默认的点 ok 启动 Android Studio。



- 直接点击 Cancel 进入 SDK 配置。

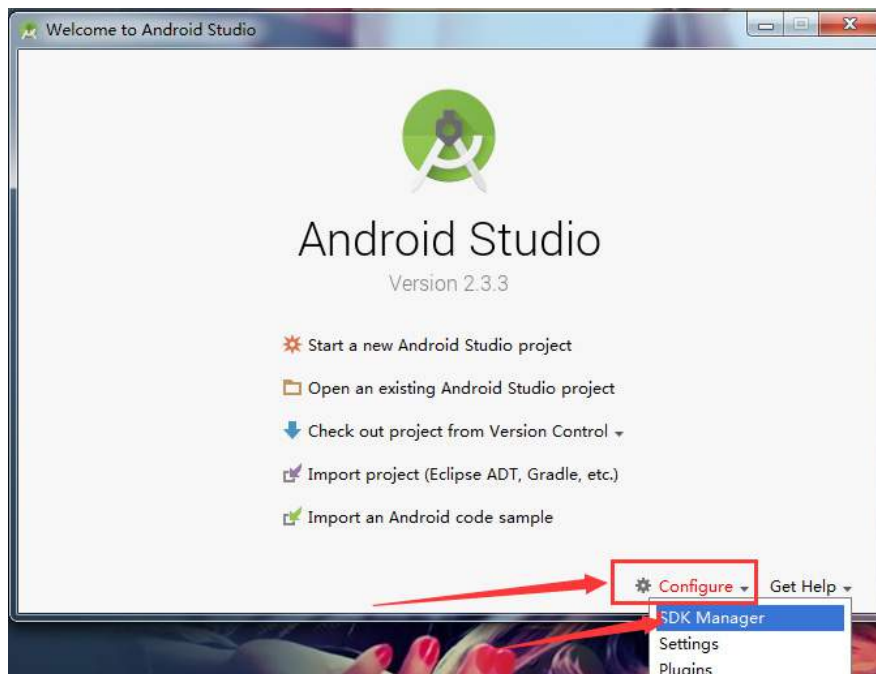


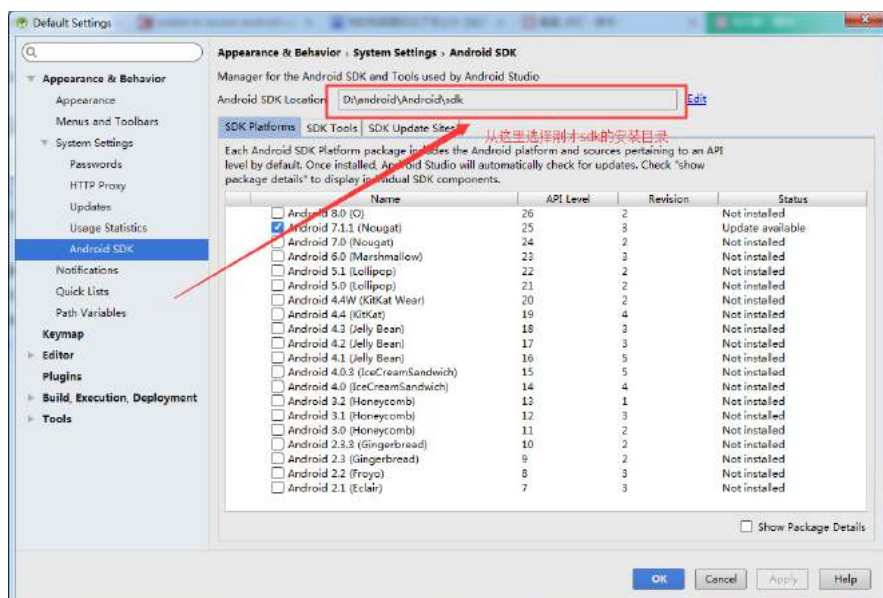
- 然后一路下一步，直至 finish，finish 后会下载一些插件，等一下就好。





配置 SDK

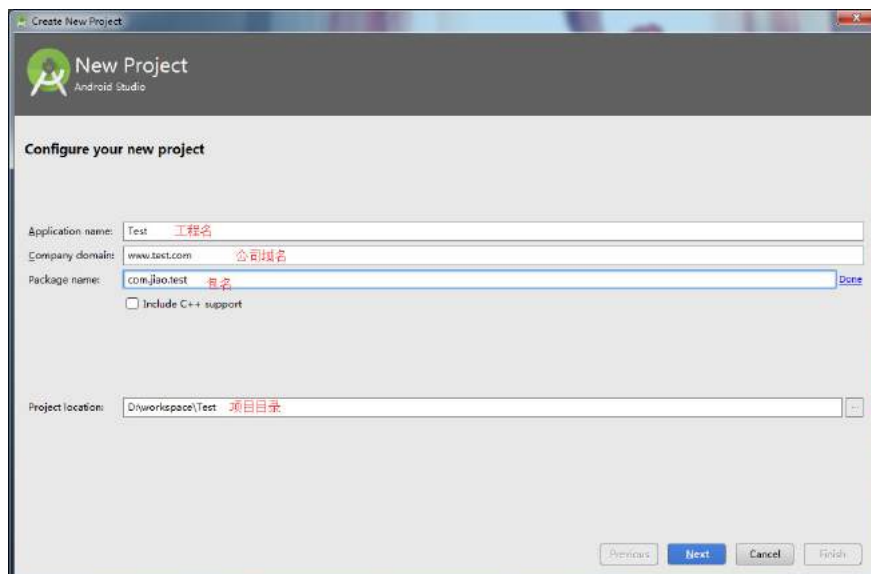




这里我们选择我们刚才 sdk 的安装目录即可。

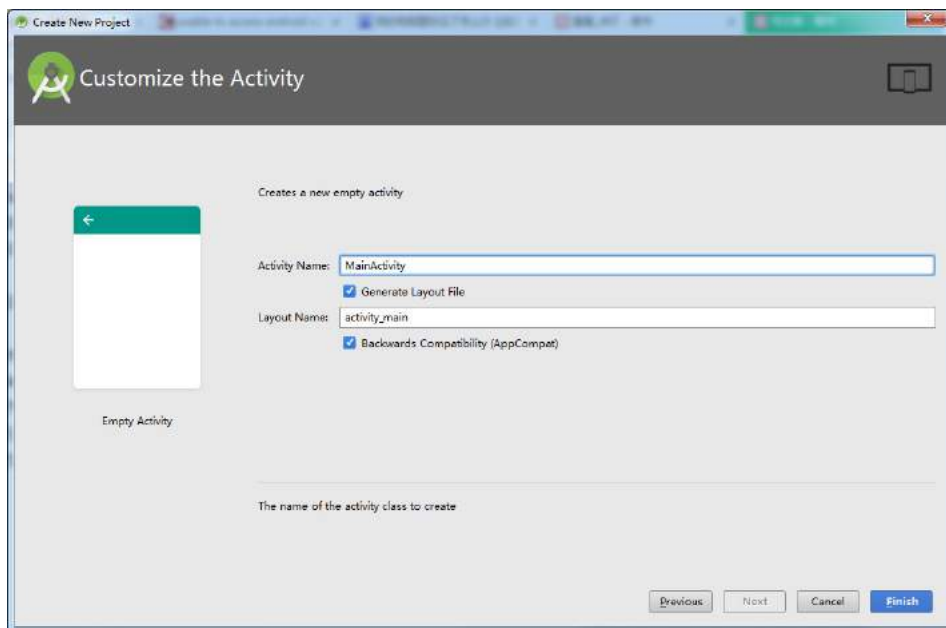
这时候我们就可以点击 Start a new Android Studio project 来新建一个工程了。

弹出新建工程对话框。



新建工程，依次填写项目名称 公司域名 包名 等信息，注意选择自己的工作区不要默认。

然后一路 next 全默认最后到 finish。



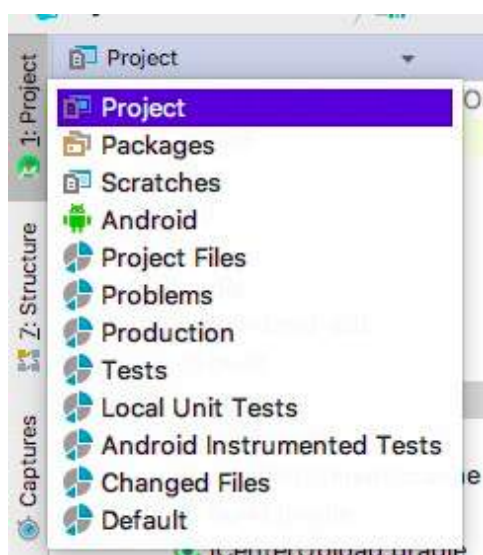
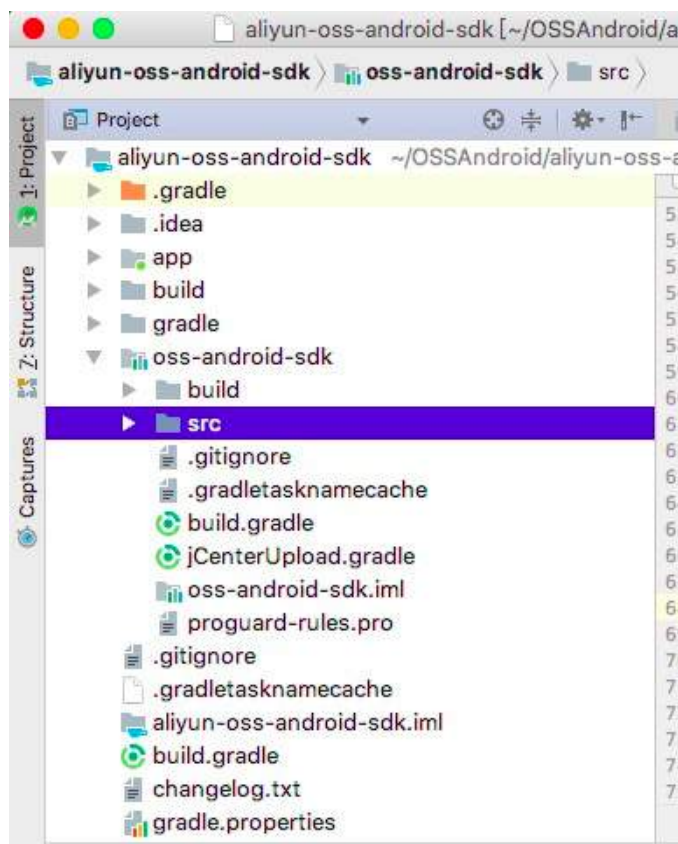
集成 OSS SDK

1. 集成工程到 Android Studio 中

- 直接 git clone 工程。

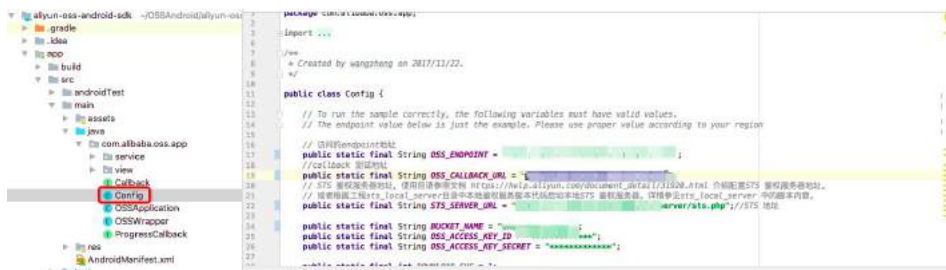
<https://github.com/aliyun/aliyun-oss-android-sdk.git>

- 引入 git 工程，这里要注意下，模式切换为 project。



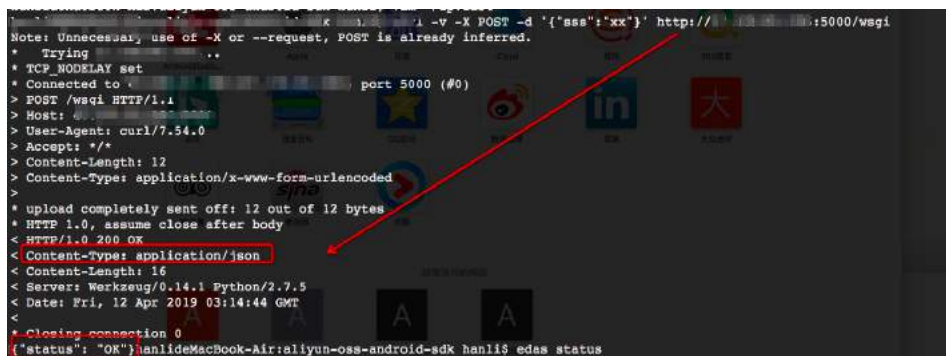
2. 引入成功后，了解主要的 java 类

2.1 Config 是我们主要的配置文件，里面存放的是 OSS 操作的 bucket 配置，以及回调、STS 获取地址、endpoint 等信息。



注意：

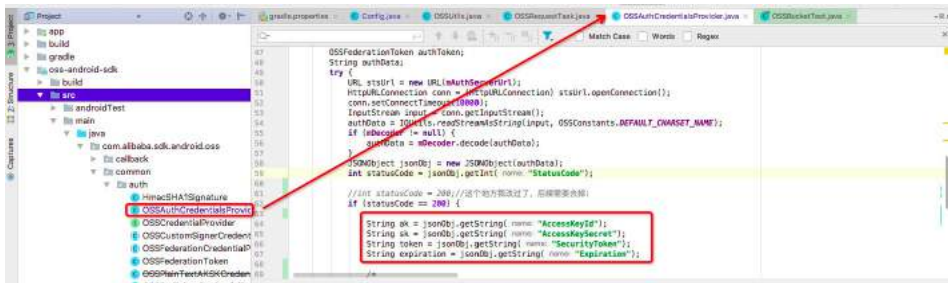
- 现在 SDK 基本都采用 STS 的方式上传，用户需要自己维护一台 STS Server 服务器，目的是用来获取 STS token 临时令牌的信息；STS 的搭建可以参考：快速搭建移动服务。
- callback 是上传成功后回调用户的地址，也是用户自己维护，目的是接收 OSS 上传成功后回调信息，用户的回调服务器必须能返回 200 并且是 JSON 的字符串，类似截图中。



2.2 OSSAuthCredentialsProvider 自动鉴权。

这个类主要功能，是根据用户设置的 STS URL 地址，自动解析出返回的变量，

并且在 STS 过期后自动去获取新的 STS token；STS 返回的鉴权信息格式一定要如下，并且返回 200。



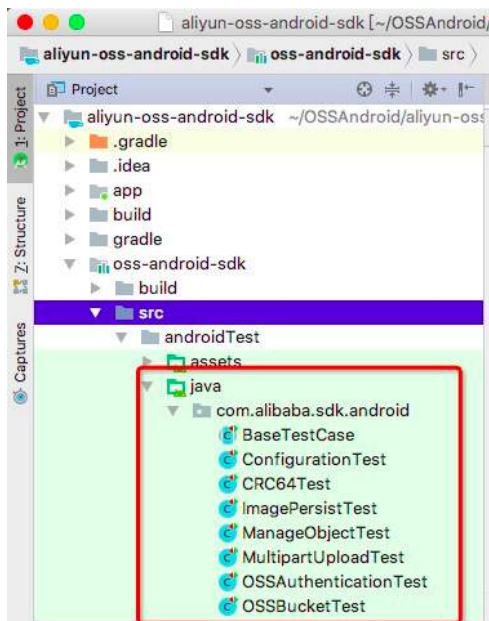
STS 地址返回的鉴权信息格式

```

{"StatusCode":200,"AccessKeyld":"STS.NJxxxxxxxxxx1zMxm6Q","-
AccessKeySecret":"EDLc9CxxxxxxxxPcSDxqqrW1kwCh5z7","Expira-
tion":"2019-04-12T04:24:32Z","SecurityToken":"CAIS9xxxxxxxxxxKH6"}

```

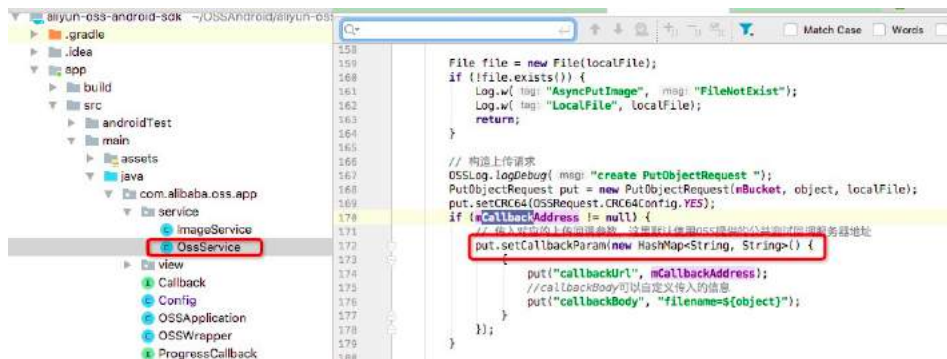
2.3 这个路径下面主要是 Android 测试的类文件。





2.4 demo 的测试集合类。

OssService 中集合 demo 的测试方式 (PutObject , PutImage) 当使用者需要修改设置上传回调时可以通过这个地址进行修改。

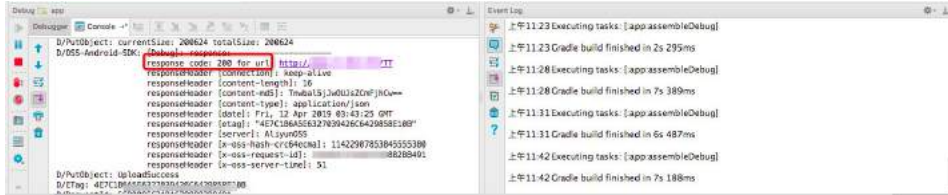


测试上传

按照截图顺序，点击 设置，提示成功后，点击 选择图片，输入 object 名称，点击上传。



通过 DEBUG 结果我们可以看到文件上传成功，导致我们的初步测试已经成功。



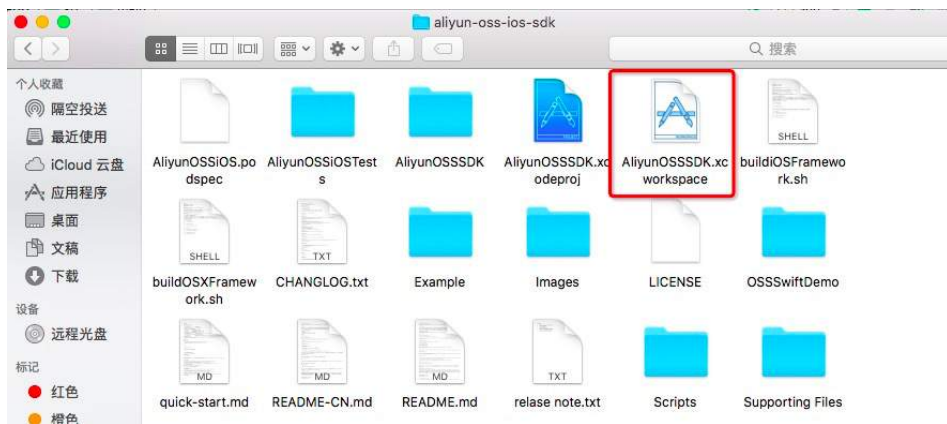
手把手教你 iOS SDK 搭建问题

背景

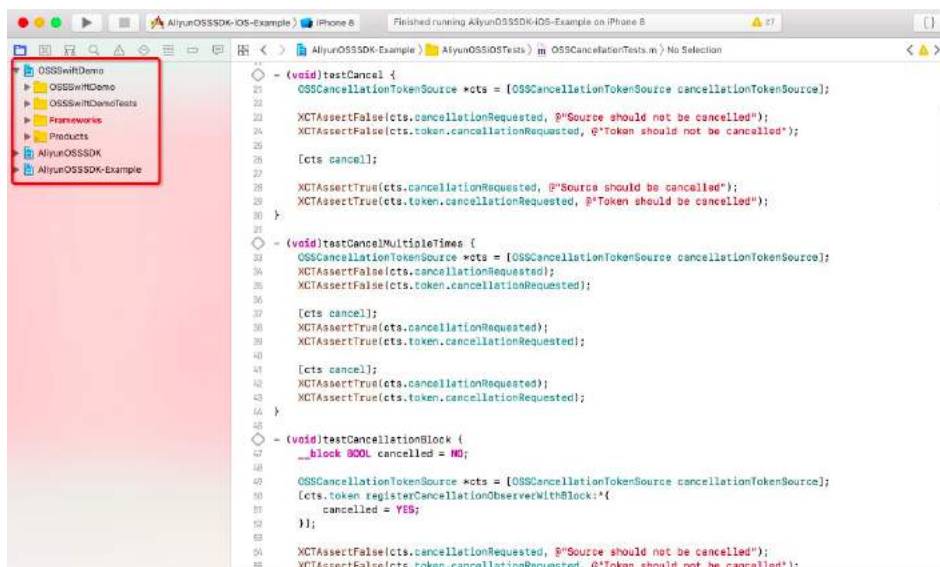
很多小伙伴对 iOS 的 SDK 搭建熟悉，但是对于上云的 OSS iOS SDK 使用有些疑问，今天从环境搭建到客户端使用全面给大家梳理一遍个人的使用总结。

通过 git 拉取项目

地址，拉下的目录结构如下；workspace 就是我们的编译后直接可运行文件，如果本机已经安装好了 xcode，可以直接运行；未安装 xcode 请按照 [install](#)。

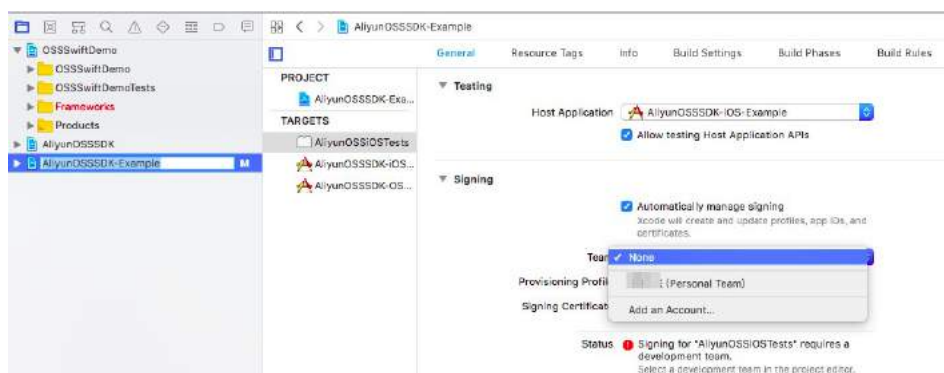


工程的目录如下，其中 Example 是我们运行在 MAC 例子。

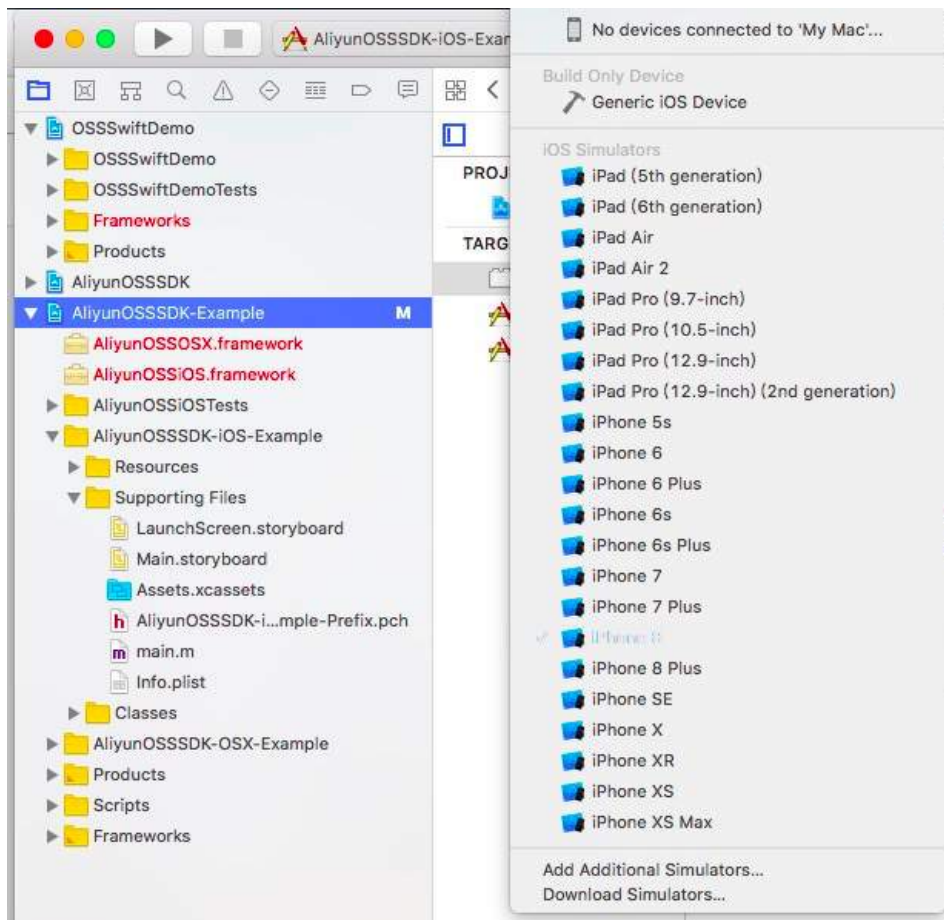


测试配置

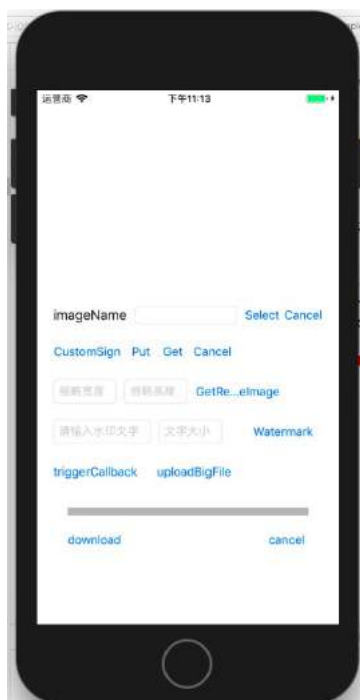
配置开发者账号。



配置运行 simulator。



编译成功后 simulator 就运行出来了。



如何配置 STS 地址

在这个应用代码中可以配置 STS 的地址。

```
1 //
2 //  OSSTestMacros.h
3 //  AliyunOSSiOSTests
4 //
5 //  Created by 乔松 on 2017/12/11.
6 //  Copyright © 2017年 阿里云. All rights reserved.
7 //
8
9 #ifndef OSSTestMacros_h
10 #define OSSTestMacros_h
11
12 #define OSS_ACCESSKEY_ID      @"AccessKeyID"           // 子账号id
13 #define OSS_SECRETKEY_ID     @"AccessKeySecret"        // 子账号secret
14
15 #define OSS_BUCKET_PUBLIC    @"public-bucket"         // bucket名称
16 #define OSS_BUCKET_PRIVATE   @"private-bucket"        // bucket名称
17 #define OSS_ENDPOINT         @"http://oss-cn-region.aliyuncs.com" // 访问阿里云endpoint
18 #define OSS_IMG_ENDPOINT     @"http://img-cn-region.aliyuncs.com" // 旧版本图片服务的endpoint
19 #define OSS_MULTIPART_UPLOADKEY @"multipart_key"       // 分片上传的object key
20 #define OSS_RESUMABLE_UPLOADKEY @"resumable_key"       // 断点续传的object key
21 #define OSS_CALLBACK_URL      @"http://oss-demo.aliyuncs.com:23456" // 对象上传成功时回调的业务服务
22 // 地址
23 #define OSS_CNAME_URL         @"http://www.cname.com/"  // cname, 用于替换
24 // bucket.endpoint.cname
25 #define OSS_STSTOKEN_URL      @"http://*.*.*.*/sts/getsts" // sts授权服务器的地址
26 #define OSS_IMAGE_KEY         @"testImage.png"         // 测试图片的名称
27 #define OSS_DOWNLOAD_FILE_NAME @"OSS_DOWNLOAD_FILE_NAME" // 用于下载的object key
28
29 #endif /* OSSTestMacros_h */
```

如何安装 browsers JS SDK ? 这么做事半功倍

浅谈

今天带来的是 OSS browser js SDK 的安装过程和使用的 demo 测试用例。

环境准备

OSS browser js SDK 是基于 node js 框架上的服务端程序，服务端启动以后，提供客户端的访问地址。

- 准备 node js 安装，最好在 9.x.x 以上版本，我当前的测试版本是 v10.9.0。
- 测试浏览器环境。（IE>=10，主流版本的 Chrome/Firefox/Safari，主流版本的 Android/iOS/WindowsPhone ）

开始安装

下载源码 git 库。

- git clone <https://github.com/ali-sdk/ali-oss.git>。

npm 开始安装。

- cd ali-oss，执行 npm install。
- cd example，执行 npm install。

tips：因为部分浏览器不支持 promise，需要引入 promise 兼容库。例如：IE10 和 IE11 需要引入 promise-polyfill。

修改配置文件

1. OSS browser 自带集成了 STS 生成的功能，其实就是在本地启动了一个小型的 web server，这样用可以通过 STS 的安全方式上传、下载 OSS。如要用这个集成的 STS 生成方式，需要修改：

```
#vim ali-oss/example/server/config.js
module.exports = {
  AccessKeyId: "子账号 accesskey",
  AccessKeySecret: "子账号 accesskeysecret",
  RoleArn: "角色 Arn",
  // 建议 Token 失效时间为 1 小时
  TokenExpireTime: '3600',
  PolicyFile: 'policy/all_policy.txt'
};
```

2. 如果用户不想用这个集成的 STS 生成器，可以自己单独写个生成 STS 代码。那么上面的 1 步忽略，直接执行以下操作。vim ali-oss/example/server/config.js，将 bucket 和 region 替换成自己的信息。

```
// require("babel-polyfill")
require('./style.css')
const $ = require('jquery');
// if use in react , you can use require('ali-oss/dist/aliyun-oss-sdk.js'), or see webpack.prod.js
// import local for test
// const OSS = require('...../lib/browser.js');
const OSS = require('ali-oss');
const crypto = require('crypto');

const appServer = 'http://localhost:9000/sts';
const bucket = 'shenzhen-';
const region = 'oss-cn-';
const { Buffer } = OSS;
```

云栖社区 yq.aliyun.com

3. 如果用户自己单独写了一个 sts 的程序，需要将 main.js 中依赖的 sts 地址换成自己的访问链接。

```
// require("babel-polyfill")
require('./style.css')
const $ = require('jquery');
// if use in react , you can use require('ali-oss/dist/aliyun-oss')
// import local for test
// const OSS = require('../lib/browser.js');
const OSS = require('ali-oss');
const crypto = require('crypto');

const appServer = 'http://localhost:9000/sts';
const bucket = 'aliyun-oss';
const region = 'cn-hangzhou';
const { Buffer } = OSS;
```

云栖社区 yq.aliyun.com

4. 在 OSS 上配置跨域头，避免跨域访问到 OSS 是出现 deny 403 的情况。
如果客户端访问是 <http://192.168.1.102/brower/testindex.html>，在 OSS
跨域来源上 IP 也要加入，最方便的做法是配置为 *。

跨域规则

来源

来源可以设置多个，每行一个，每行最多能有一个通配符「*」

允许 Methods

☒ GET ☒ POST ☒ PUT ☐ DELETE ☒ HEAD

允许 Headers

*

允许 Headers 可以设置多个，每行一个，每行最多能有一个通配符「*」

暴露 Headers

Etag

暴露 Headers 可以设置多个，每行一个，不允许出现通配符「*」

缓存时间 (秒)

0

云栖社区 yq.aliyun.com

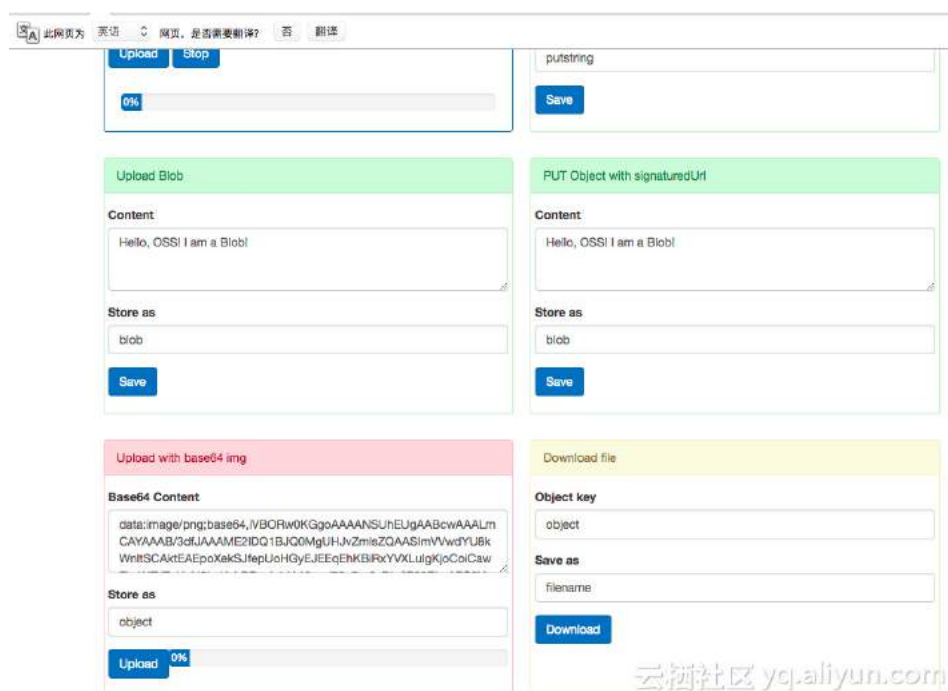
启动

cd ali-oss/example 执行 npm run start

如果用户想要用 https 的方式上传, 在 OSSClient 初始化时加上 secure:true 就是 https 传输了。

```
const client = new OSS({
  region,
  accessKeyId: creds.AccessKeyId,
  accessKeySecret: creds.AccessKeySecret,
  stsToken: creds.SecurityToken,
  bucket,
  secure:true
});
```

下图就是启动后的效果。



使用遇到问题

上传回调如何添加

```
const uploadFile = function uploadFile(client) {
  if (!uploadFileClient || Object.keys(uploadFileClient).length === 0) {
    uploadFileClient = client;
  }

  const file = document.getElementById('file').files[0];
  const key = document.getElementById('object-key-file').value.trim() ||
'object';

  console.log(`${file.name} => ${key}`);
  const options = {
    progress,
    partSize: 500 * 1024,
    timeout: 60000,
    meta: {
      year: 2017,
      people: 'test',
    },
    callback: {
      // 这里是添加上传回调的位置
      url: 'https://uploadtest.xueersi.com/v2/sync',
      /* host: 'oss-cn-shenzhen.aliyuncs.com', */
      /* eslint no-template-curly-in-string: [0] */
      body: 'bucket=${bucket}&object=${object}&var1=${x:var1}',
      contentType: 'application/x-www-form-urlencoded',
      customValue: {
        var1: 'value1',
        var2: 'value2',
      },
    },
  },
```

如何使用 https 传输

```
const client = new OSS({
  region,
  accessKeyId: creds.AccessKeyId,
  accessKeySecret: creds.AccessKeySecret,
  stsToken: creds.SecurityToken,
  bucket,
  secret: true
})
```

OSS python SDK 问题排查攻略

浅谈

很多 oss 使用者在使用 Python SDK 时出现很多问题，不确定是否影响使用，有的安装失败环境有问题，今天说下遇到的几个案例。

官方安装

- pip install oss2。
- 版本最好是 2.7.5 或以上。

oss2 依赖

- 如果要开启 crc64 循环冗余校验，需要先将 crcmod 安装好。
- 安装 python-devel 执行 yum install python-devel。
- 需要循环冗余校验，安装 crcmod 执行 pip install crcmod。

安装遇到的问题

验证 oss2

先判断是 oss2 是否安装成功，在命令行输入 python 并回车，进入 Python 环境，执行以下命令检查 SDK 版本：

```
>>> import oss2
>>> oss2.__version__
'2.x.x'
```

导入 crcmod 失败

```
>>> import crcmod._crcfunext
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ImportError: No module named _crcfunext
```

- 没有安装 python-devel 或者 crcmod，如果已经安装 crcmod，请 uninstall 后，重新安装 python-devel 然后再安装 crcmod。
- crcmod 安装的环境 path 和你本机的 python 环境不一致，可以用 sys path 查看你 python 加载的环境变量路径确认一下。
- 参考一些网上的处理方法，这是个开源的报错参考。

使用遇到问题排查

问：同台机器 ossutil 很快，python SDK 很慢

- ossutil 源码是 go，并发上传的性能确实很好，但是 python SDK 也至于慢很多，一般这种情况基本都是默认开启了 crc64。
- 如果对性能有要求的话，建议把 crc64 关掉，通过在 header 头中增加 Content-md5 头的方式替代 crc64 更好。

```
1. = oss2.Auth('AK', 'SK')
```

```
2. = oss2.Bucket(auth, 'endpoint', 'bucket', enable_crc=False)
```

```
[root@localhost ~]# time python putobject.py
real    0m1.698s
user    0m0.183s
sys     0m0.086s
```

云栖社区 yq.aliyun.com

问：安装 oss2 导入出现 urllib3 不存在提示

```

root@ ~ # pip install oss2
Requirement already satisfied (use --upgrade to upgrade): oss2 in /usr/lib/python2.7/site-packages
Requirement already satisfied (use --upgrade to upgrade): requests<2.9.0 in /usr/lib/python2.7/site-packages (from oss2)
Requirement already satisfied (use --upgrade to upgrade): crcmod<=1.7 in /usr/lib/python2.7/site-packages (from oss2)
Requirement already satisfied (use --upgrade to upgrade): pycryptodome<=3.4.7 in /usr/lib/python2.7/site-packages (from oss2)
Requirement already satisfied (use --upgrade to upgrade): aliyun-python-sdk-kms<=2.4.1 in /usr/lib/python2.7/site-packages (from oss2)
Requirement already satisfied (use --upgrade to upgrade): aliyun-python-sdk-core<=2.6.2 in /usr/lib/python2.7/site-packages (from oss2)
You are using pip version 8.1.2, however version 18.1 is available.
You should consider upgrading via the 'pip install --upgrade pip' command.
root@ ~ # python
Python 2.7.5 (default, Jul 13 2018, 13:06:57)
[GCC 4.8.5 20150623 (Red Hat 4.8.5-28)] on linux2
type help(), __copyright__, "credits" or "license()" for more information.
>>> import oss2
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "/usr/lib/python2.7/site-packages/oss2/_init_.py", line 5, in <module>
    from .api import Service, Bucket, CryptoBucket
  File "/usr/lib/python2.7/site-packages/oss2/api.py", line 119, in <module>
    from . import http
  File "/usr/lib/python2.7/site-packages/oss2/http.py", line 13, in <module>
    import requests
  File "/usr/lib/python2.7/site-packages/requests/_init_.py", line 58, in <module>
    from . import utils
  File "/usr/lib/python2.7/site-packages/requests/utils.py", line 32, in <module>
    from .exceptions import InvalidURL
  File "/usr/lib/python2.7/site-packages/requests/exceptions.py", line 10, in <module>
    from .packages.urllib3.exceptions import HTTPError as BaseHTTPError
  File "/usr/lib/python2.7/site-packages/requests/packages/_init_.py", line 95, in load_module
    raise ImportError("No module named '%s'" % (name,))
ImportError: No module named 'requests.packages.urllib3'

```

云栖社区 yq.aliyun.com

答：

- 和之前说过的一样，这种错误都是 python 自身的问题，看哪个依赖的模块没有，安装即可，oss2 的 http 请求处理依赖 urllib3。
- 确认下 pip install 安装的环境和本地 python 环境是否一致。

问：OSS python SDK 分片上传失败

文件名称	上传 ID	文件碎片
<input type="checkbox"/> accesslog_2018_09.tar.gz	E609B68C0F0647BF902DF6032902BBD5	点击统计
<input type="checkbox"/> accesslog_2018_10.tar.gz	EA9E2EDFBA00450688127E17D2AED4CA	点击统计

云栖社区 yq.aliyun.com

用户通过 python SDK 的分片上传函数上传到 OSS 失败，碎片管理中出现很对碎片。

- 先确认是直接传到 OSS，还是通过其他 proxy 传输到 OSS (类似 CDN)，如果经过 CDN 再上传到 OSS 需要在 OSS 上配置跨域的头，Access-Control-Allow-Origin、Access-Control-Allow-Method、Access-Control-Allow-Header，并且将 Etag 暴露出去。
- 客户端上传失败是因为网络超时，还是捕获到异常上传失败，需要详细看下捕获到的 SDK 异常信息分析，如果是网络超时导致上传失败，建议使用断点续传来替代普通上传。断点续传支持分片，并发，已经弱网的兼容。
- 清理掉上传失败的碎片文件重新上传。
- 当以上操作都解决不了你的问题时，需要提供以下信息升级阿里云便于快速定位：
 - 提供 SDK 异常时返回的 requestID，这个属性是 response header 中携带的记录了完整的 OSS 请求过程。
 - 客户端部署 tcpdump，然后重新运行代码上传，保存抓包。

```
tcpdump -i <网卡出口名称> -s0 host <访问 oss 的域名> -w faild.pcap
```

案例

Centos 机器上执行分片上传 SDK 问题正常，但是 ubuntu 机器上传总是报 403 失败。

分析

首先出现问题后，如果在 ubuntu 机器上，操作 OSS 出现 403，OSS 服务端会返回 403 对应的 OSS requestID，里面包含了 403 的原因，需要提供给阿里云排查。

客户端部署 tcpdump 抓包，可以通过 tcp 报文排查是否由于 header 头信息不对引起的计算签名与服务端不匹配。

```
POST /ttsservice%2Fpasswd?uploadId=D468E486D1D94D90A1AB8885A4E32AE0 HTTP/1.1
Host: rokid.oss-cn-hangzhou.aliyuncs.com
Accept-Encoding: identity
Accept: */*
Content-Length: 137
date: Sat, 29 Dec 2018 07:32:34 GMT
authorization: OSS LTAIknFr:r2KPR0y4E0G5tnU/MYdcvXHPQQ4=
Content-Type: application/x-www-form-urlencoded
User-Agent: aliyun-sdk-python/2.6.0(Linux/4.4.0-31-generic/x86_64;3.4.3)

<CompleteMultipartUpload><Part><PartNumber>1</
PartNumber><ETag>"3195544E19D99658706D51EF5"</ETag></Part></
CompleteMultipartUpload>HTTP/1.1 403 Forbidden

Server: AliyunOSS
Date: Sat, 29 Dec 2018 07:33:43 GMT
Content-Type: application/xml
Content-Length: 1122
Connection: keep-alive
x-oss-request-id: 5C2723573183A12D
x-oss-server-time: 0

<?xml version="1.0" encoding="UTF-8"?>
<Error>
  <Code>SignatureDoesNotMatch</Code>
  <Message>The request signature we calculated does not match the signature
you provided. Check
your key and signing method.</Message>
  <RequestId> 5C2723573183A12D </RequestId>
  <HostId>rokid.oss-cn-hangzhou.aliyuncs.com</HostId>
  <OSSAccessKeyId> LTAIknFr </OSSAccessKeyId>
  <SignatureProvided>r2KPR0y4E0G5tnU/MYdcvXHPQQ4=</SignatureProvided>
  <StringToSign>POST
application/x-www-form-urlencoded
Sat, 29 Dec 2018 07:32:34 GMT
/rokid/ttsservice/passwd?uploadId=D468E486D1D94D90A1AB8885A4E32AE0</
StringToSign>
  <StringToSignBytes>50 4F 53 54 0A 0A 61 70 70 6C 69 63 61 74 69 6F 6E 2F 78
2D 77 77 77 2D 66 6F
72 6D 2D 75 72 6C 65 6E 63 6F 64 65 64 0A 53 61 74 2C 20 32 39 20 44 65 63 20
32 30 31 38 20 30 37 3A
33 32 3A 33 34 20 47 4D 54 0A 2F 72 6F 6B 69 64 2D 6F 70 73 2D 6D 6F 64 65 6C
2F 74 74 73 73 65 72 76
69 63 65 2F 70 61 73 73 77 64 3F 75 70 6C 6F 61 64 49 64 3D 44 34 36 38 45 34
```

```

38 36 44 31 44 39 34 44
39 30 41 31 41 42 38 38 38 35 41 34 45 33 32 41 45 30 </StringToSignBytes>
</Error>

```

- 如是客户端抓到的报文信息，我们主要是分析请求头的信息。拿到请求后，带入到以下脚本中，看下计算出来的结果是否和 SDK 一致。如果一致说明 SDK 的计算是正确的，那么如果服务端收到的和客户端计算的 signature 还不一致说明请求的内容可能被网络中设备改动过，可以使用 https 的方式上传。

```

import base64
import hmac
import sha
mac = hmac.new("<Secretkey>", "POST\n\napplication/x-www-form-urlencoded\nSat,
29 Dec 2018
07:32:34 GMT\n/rokid/ttsservice/
passwd?uploadId=D468E486D1D94D90A1AB8885A4E32AE0", sha)
Signature = base64.b64encode(mac.digest())
print(Signature)

```

- 如果抓包中和脚本中计算的结果不一致，有可能存在 SDK 在 ubuntu 平台编译的适配问题导致 MD5 值不一样。



 **阿里云** 开发者社区



云服务技术大学
云产品干货高频分享



云服务技术课堂
和大牛零距离沟通



阿里云开发者“藏经阁”
海量免费电子书下载