阿里云
对象存储 OSS

最佳实践

文档版本：20180806

法律声明

阿里云提醒您在阅读或使用本文档之前仔细阅读、充分理解本法律声明各条款的内容。如果您阅读或使用本文档，您的阅读或使用行为将被视为对本声明全部内容的认可。

1. 您应当通过阿里云网站或阿里云提供的其他授权通道下载、获取本文档，且仅能用于自身的合法合规的业务活动。本文档的内容视为阿里云的保密信息，您应当严格遵守保密义务；未经阿里云事先书面同意，您不得向任何第三方披露本手册内容或提供给任何第三方使用。
2. 未经阿里云事先书面许可，任何单位、公司或个人不得擅自摘抄、翻译、复制本文档内容的部分或全部，不得以任何方式或途径进行传播和宣传。
3. 由于产品版本升级、调整或其他原因，本文档内容有可能变更。阿里云保留在没有任何通知或者提示下对本文档的内容进行修改的权利，并在阿里云授权通道中不时发布更新后的用户文档。您应当实时关注用户文档的版本变更并通过阿里云授权渠道下载、获取最新版的用户文档。
4. 本文档仅作为用户使用阿里云产品及服务的参考性指引，阿里云以产品及服务的”现状“、“有缺陷”和“当前功能”的状态提供本文档。阿里云在现有技术的基础上尽最大努力提供相应的介绍及操作指引，但阿里云在此明确声明对本文档内容的准确性、完整性、适用性、可靠性等不作任何明示或暗示的保证。任何单位、公司或个人因为下载、使用或信赖本文档而发生任何差错或经济损失的，阿里云不承担任何法律责任。在任何情况下，阿里云均不对任何间接性、后果性、惩戒性、偶然性、特殊性或刑罚性的损害，包括用户使用或信赖本文档而遭受的利润损失，承担责任（即使阿里云已被告知该等损失的可能性）。
5. 阿里云网站上所有内容，包括但不限于著作、产品、图片、档案、资讯、资料、网站架构、网站画面的安排、网页设计，均由阿里云和/或其关联公司依法拥有其知识产权，包括但不限于商标权、专利权、著作权、商业秘密等。非经阿里云和/或其关联公司书面同意，任何人不得擅自使用、修改、复制、公开传播、改变、散布、发行或公开发表阿里云网站、产品程序或内容。此外，未经阿里云事先书面同意，任何人不得为了任何营销、广告、促销或其他目的使用、公布或复制阿里云的名称（包括但不限于单独为或以组合形式包含”阿里云”、Aliyun”、“万网”等阿里云和/或其关联公司品牌，上述品牌的附属标志及图案或任何类似公司名称、商号、商标、产品或服务名称、域名、图案标示、标志、标识或通过特定描述使第三方能够识别阿里云和/或其关联公司）。
6. 如若发现本文档存在任何错误，请与阿里云取得直接联系。

通用约定

格式	说明	样例
	该类警示信息将导致系统重大变更甚至故障，或者导致人身伤害等结果。	 **禁止：** 重置操作将丢失用户配置数据。
	该类警示信息可能导致系统重大变更甚至故障，或者导致人身伤害等结果。	 **警告：** 重启操作将导致业务中断，恢复业务所需时间约10分钟。
	用于补充说明、最佳实践、窍门等，不是用户必须了解的内容。	 **说明：** 您也可以通过按**Ctrl** + **A**选中全部文件。
>	多级菜单递进。	设置 > 网络 > 设置网络类型
粗体	表示按键、菜单、页面名称等UI元素。	单击 **确定**。
`courier` 字体	命令。	执行 `cd /d C:/windows` 命令，进入Windows系统文件夹。
斜体	表示参数、变量。	`bae log list --instanceid` *`Instance_ID`*
[]或者[a|b]	表示可选项，至多选择一个。	`ipconfig` *`[-all|-t]`*
{}或者{a|b}	表示必选项，至多选择一个。	`swich` *`{stand | slave}`*

目录

法律声明.....I
通用约定.....I
1 移动应用端直传实践.....1
1.1 快速搭建移动应用直传服务.....1
1.2 权限控制.....13
1.3 快速搭建移动应用上传回调服务.....19
2 Web端直传实践.....25
2.1 Web端直传实践简介.....25
2.2 JavaScript客户端签名直传.....26
2.3 服务端签名后直传.....29
2.4 服务端签名直传并设置上传回调.....36
3 数据迁移到OSS.....42
3.1 如何将HDFS容灾备份到OSS.....42
3.2 使用OssImport迁移数据.....44
3.3 Amazon S3数据迁移到OSS.....49
4 数据备份.....53
4.1 备份存储空间.....53
5 存储空间管理.....55
5.1 CDN加速OSS.....55
5.2 存储类型转换.....58
5.3 防盗链.....61
5.4 跨域资源共享（CORS）.....71
5.5 音视频.....89
5.6 静态网站托管.....89
6 权限管理.....93
6.1 权限管理概述.....93
6.2 RAM和STS介绍.....93
6.3 不使用主账号.....96
6.4 读写权限分离.....97
6.5 Bucket权限分离.....98
6.6 STS临时授权访问.....101
6.7 OSS子账号设置常见问题.....113
7 音视频.....115
8 数据安全.....116
8.1 通过crc64校验数据传输的完整性.....116

8.2 通过客户端加密保护数据 118
9 OSS资源的监控与报警 122
10 OSS性能与扩展性最佳实践 125
11 安卓应用示例 127
11.1 OssDemo简介 127
11.2 使用已经搭建好的应用服务器 127
11.3 上传文件 129
11.4 图片处理 132

1 移动应用端直传实践

1.1 快速搭建移动应用直传服务

背景

在移动互联的时代，手机APP上传的数据越来越多，我们可以把数据存储的问题交给OSS，让开发者更加专注于自己的应用逻辑。

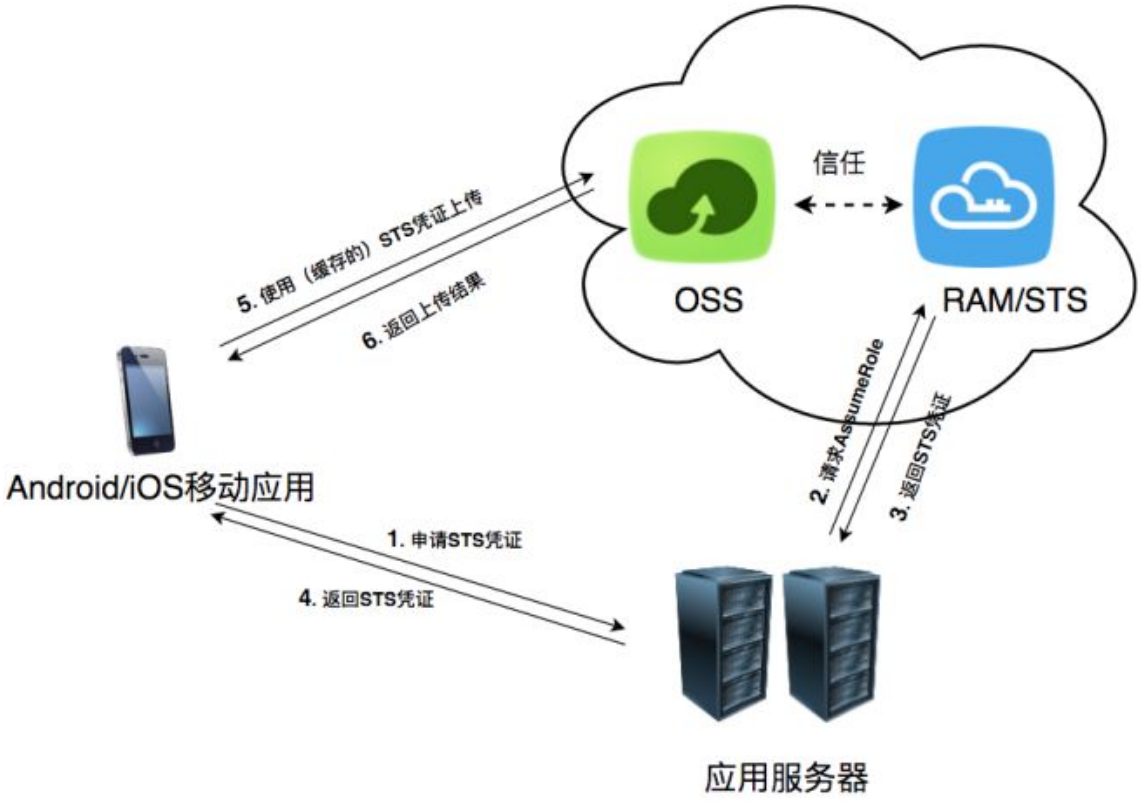
本文主要介绍如何在30分钟内搭建一个基于OSS的移动应用数据直传服务。所谓直传就是移动应用数据的上传和下载直接连接OSS，只有控制流走用户自己的服务器。

优势

搭建一个基于OSS的移动应用数据直传服务，具有以下优势：

- 上传下载方式更加安全（临时、灵活的赋权鉴权）。
- 成本低，用户不需要准备很多服务器。移动应用直联云存储，只有控制流走用户自己的应用服务器。
- 高并发，支持海量用户（OSS有海量的上传和下载带宽）。
- 弹性（OSS有无限扩容的存储空间）。
- 方便，可以方便的对接到媒体转码服务-视频多端适配，图片处理服务，CDN加速下载等。

架构图如下所示：



解析：

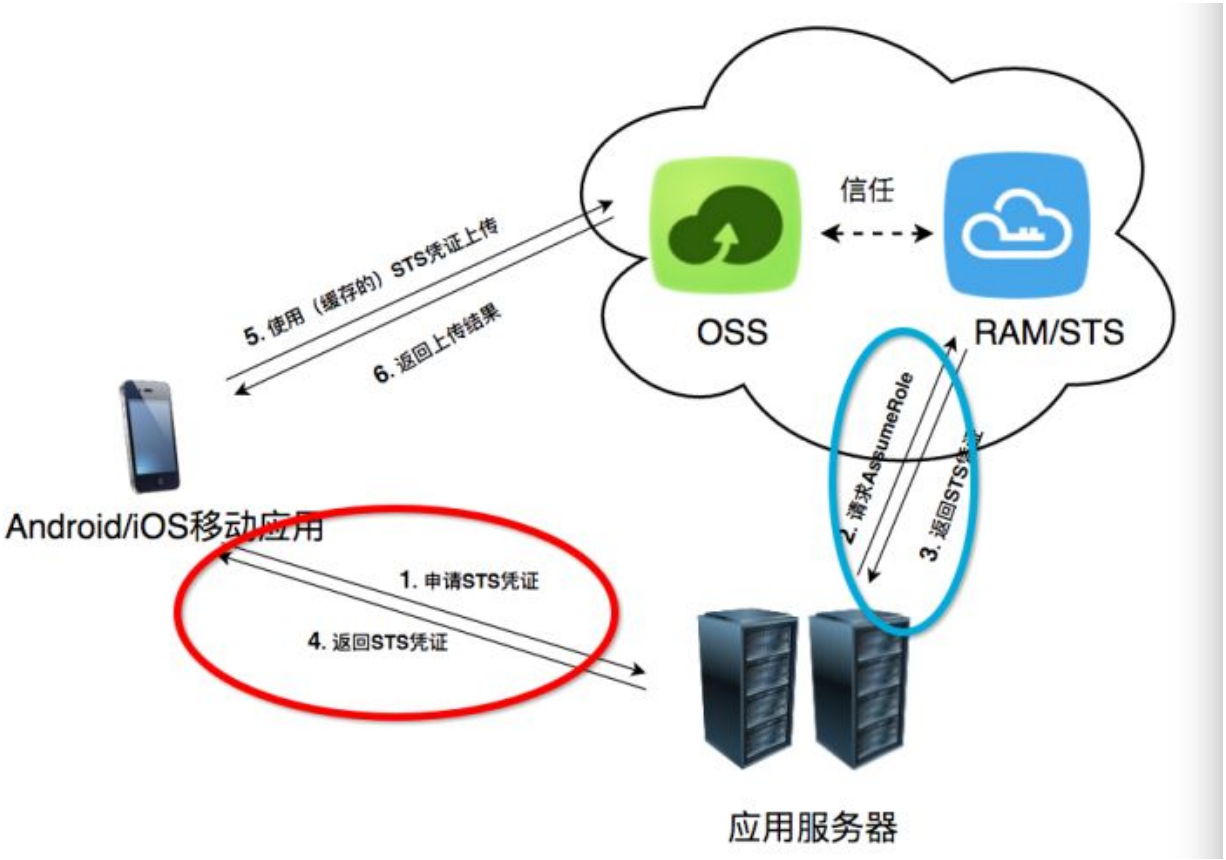
- Android/iOS 移动应用，即最终用户手机上的APP。
- OSS，即阿里云对象存储，负责存储APP上传的数据，可以参考官网介绍。
- RAM/STS负责生成临时上传凭证。
- 用户应用服务器，即提供该Android/iOS应用的开发者开发的APP后台服务，管理APP上传和下载的Token，以及用户在APP上传数据的元数据信息。

实现步骤

1. 应用向用户的应用服务器申请一个临时上传凭证。

   Android/iOS应用不能直接存储AccessKeyID/AccessKeySecret，这样会存在泄密的风险。所以应用必须向用户的应用服务器申请一个临时上传凭证（下文将此临时上传凭证称为Token）。这个Token是有时效性的，如果这个Token的过期时间是30分钟（这个时间可以由应用服务器指定），那么在这30分钟里面，该Android/iOS应用可以使用这个Token从OSS上传和下载数据，30分钟后再重新获取。
2. 用户的应用服务器检测上述请求的合法性，然后返回Token给应用。
3. 手机拿到这个Token后就可以将数据上传到OSS，或者从OSS下载数据了。

本文档主要介绍下图中红色和蓝色框的内容。



- 蓝色框：应用服务器如何生成这个Token
- 红色框：Android/iOS应用如何获取Token

实现效果：

如下图所示，您可以扫描二维码，安装示例APP程序。此工具是用Android开发的，本文档的应用服务器也适用于iOS。

示例应用的最终效果图如下：

说明：

应用服务器中的地址是示例地址。您可以参考文章结尾的STS应用服务器代码，自己部署应用服务器。

- 应用服务器：该移动应用对应的后台应用服务器。
- 上传Bucket：该移动应用要把数据上传到哪个Bucket。
- 区域：上传Bucket对应的区域。

示例APP的使用步骤：

- 单击选择图片，然后把文件上传到OSS。
- 上传的方法支持普通上传和断点上传。

> **说明：**
> 在一些网络环境差的情况下，最好用断点上传。可以利用图片处理服务，对将要上传的图片进行缩略和加水印处理。初始使用请暂时先不要更改应用服务器地址和Bucket名字。

搭建直传服务前提条件

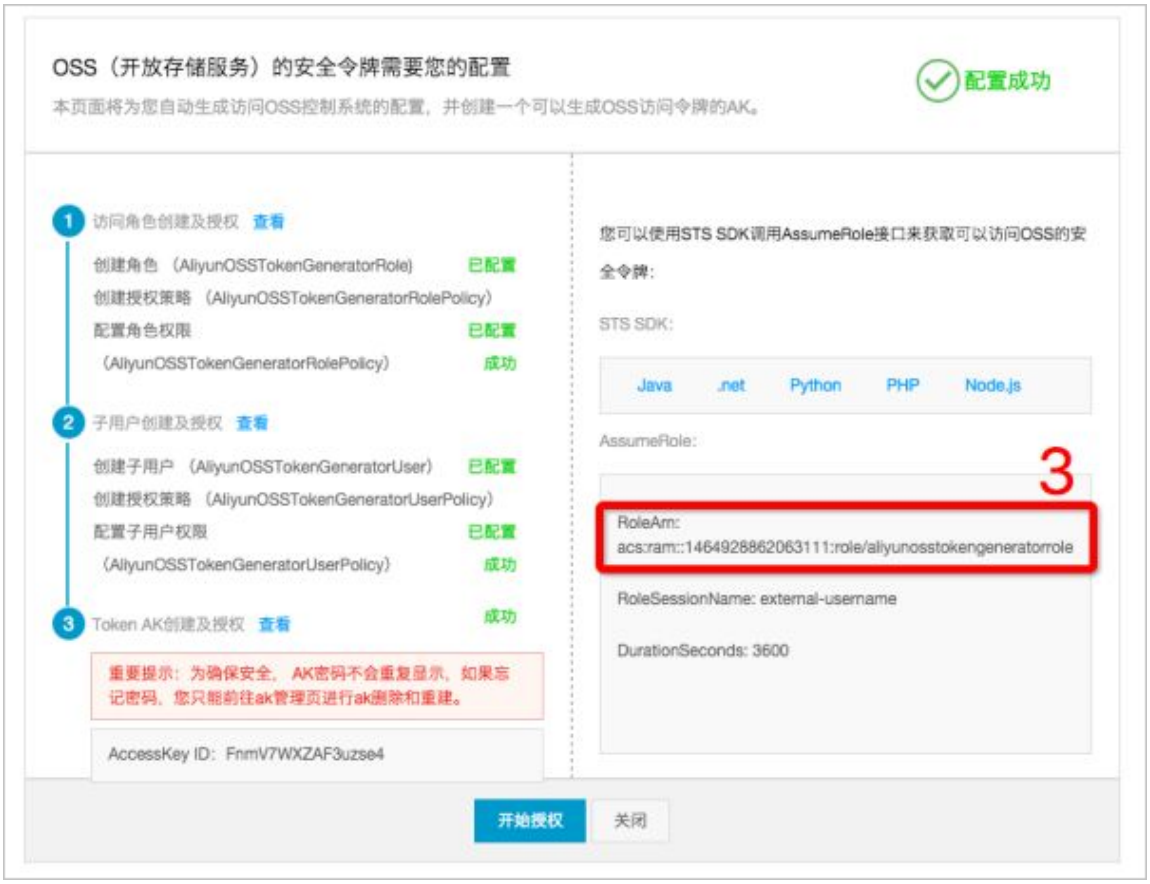
搭建直传服务需要完成以下准备工作：

1. 开通*OSS*，并且创建*Bucket*。
2. 开通STS服务。
   a. 登录 *OSS*管理控制台。
   b. 在OSS概览页中找到基础配置区域，单击 安全令牌，如下图所示：

   c. 进入到 安全令牌快捷配置 页面。

> 说明：
> 如果没有开通RAM，会弹出开通的对话框。直接单击 开通，并进行实名验证。完成后跳到本页面，并单击 开始授权。

d. 系统进行自动授权，请务必保存下图中三个红框内的参数。单击保存**AK**信息后，对话框会关闭，STS的开通完成。

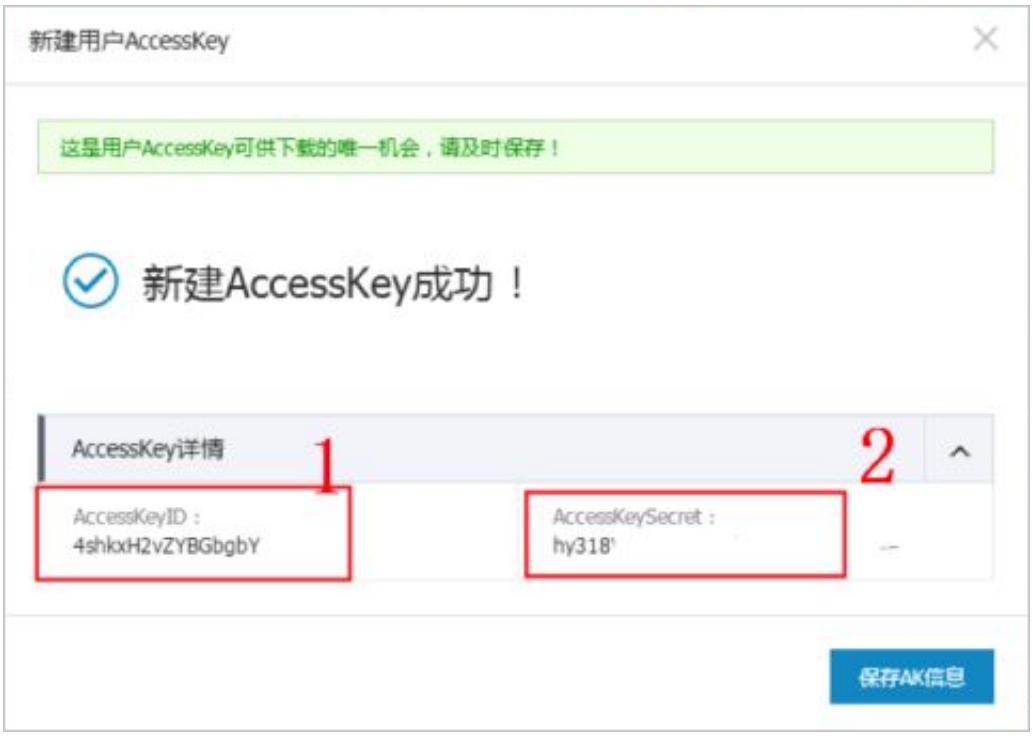


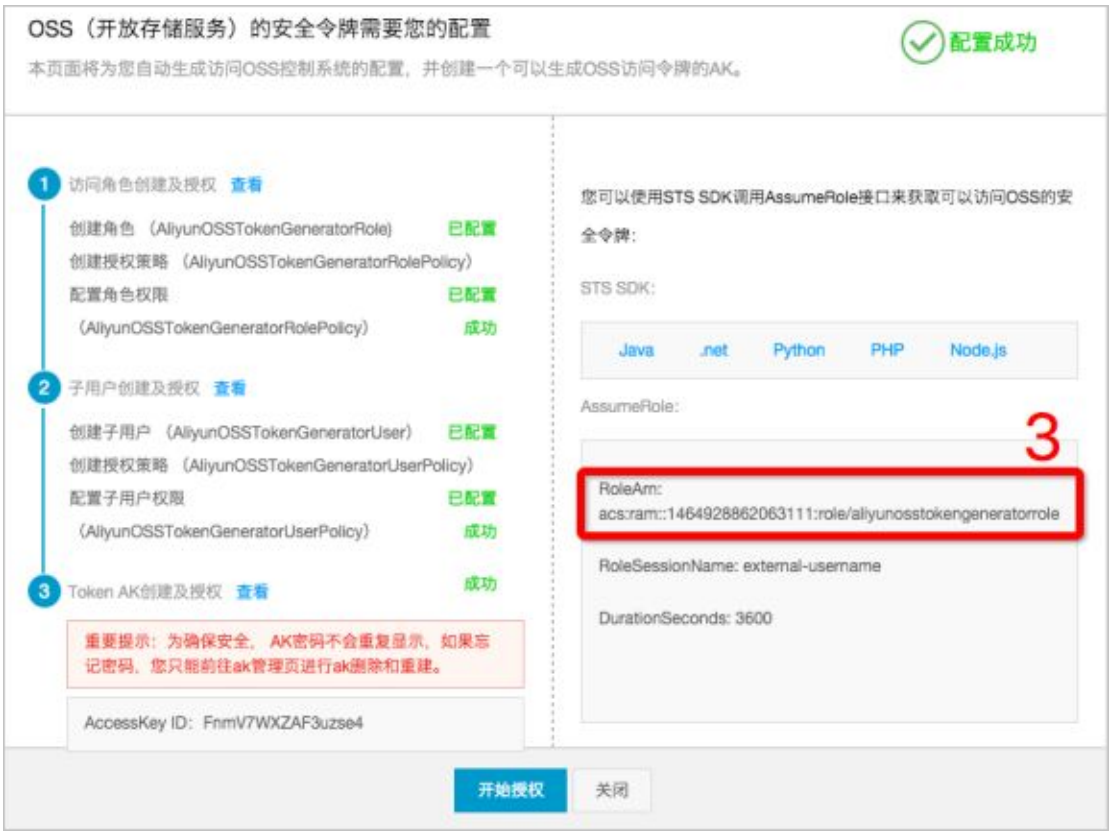
e. 如果您之前已经创建了AccessKeyId/AccessKeySecret， 打开的页面如下：

- 您可以单击如下图所示的查看。

- 单击如下图所示的创建**AccessKey**。

- 记下如下参数1、2、3。





- 保存这三个参数后，STS的开通已经完成了。

搭建一个应用服务器

应用服务器示例的配置

说明：

这个例子是采用PHP编写的，用户也可以选择自己喜欢的语言进行编写，如Java 、Python、Go、Ruby、Node.js、C#等。

为了方便开发，本教程提供了多个语言的版本示例程序供您下载，下载地址见文章最底部。

每个语言包下载下来后，都会有一个配置文件config.json如下所示：

```
{
"AccessKeyID" : "",
"AccessKeySecret" : "",
"RoleArn" : "",
"TokenExpireTime" : "900",
"PolicyFile": "policy/all_policy.txt"
}
```

说明：

- 1. AccessKeyID：填写上述图标红的参数1的内容。
  2. AccessKeySecret：填写上述图标红的参数2的内容。
  3. RoleArn：填写上述图标红的参数3的内容。
  4. TokenExpireTime：指Android/iOS应用取到这个Token的失效时间，注意，最少是900s，默认值可以不修改。
  5. PolicyFile：填写的是该Token所要拥有的权限列表的文件， 默认值可以不修改。

本文档准备了三种最常用token权限文件，放于policy目录下面。分别是:

- all_policy.txt：指定了该token拥有对该账号下创建Bucket、删除Bucket、上传文件、下载文件、删除文件的权限 。
- bucket_read_policy.txt：指定了该token拥有该账号下对指定Bucket的读权限。
- bucket_read_write_policy.txt：指定了该token拥有该账号下对指定Bucket的读写权限。

如果您想要指定这个Token只能对指定的bucket有读写权限， 请把（bucket_read_policy.txt、bucket_read_write_policy.txt）这些文件里面$BUCKET_NAME直接替换成指定的bucket名字。

- 返回的数据格式解析：

```
//正确返回
{
    "StatusCode":200,
    "AccessKeyId":"STS.3p***dgagdasdg",
    "AccessKeySecret":"rpnwO9***tGdrddgsR2YrTtI",
```

```
   "SecurityToken":"CAES+wMIARKAAZhjH0EUOIhJMQBMjRywXq7MQ/cjLYg80Aho
1ek0Jm63XMhr9Oc5s˙∂˙∂3qaPer8p1YaX1NTDiCFZWFkvlHf1pQhuxfKBc+mRR9KAbHUe
fqH+rdjZqjTF7p2m1wJXP8S6k+G2MpHrUe6TYBkJ43GhhTVFMuM3BZajY3VjZWOXBI
ODRIR1FKZjIiEjMzMzE0MjY0NzM5MTE4NjkxMSoLY2xpZGSSDgSDGAGESGTE
TqOio6c2RrLWRlbW8vKgoUYWNzOm9zczoqOio6c2RrLWRlbW9KEDExNDg5Mz
AxMDcyNDY4MThSBTI2ODQyWg9Bc3N1bWVkUm9sZVVzZXJgAGoSMzMzMTQyNj
Q3MzkxMTg2OTExcglzZGstZGVtbzI=",
   "Expiration":"2015-12-12T07:49:09Z",
}
//错误返回
{
    "StatusCode":500,
    "ErrorCode":"InvalidAccessKeyId.NotFound",
    "ErrorMessage":"Specified access key is not found."
}
```

正确返回说明：（下面五个变量将构成了一个Token）

- StatusCode：表示获取Token的状态，获取成功时，返回值是200。
- AccessKeyId：表示Android/iOS应用初始化OSSClient获取的 AccessKeyId。
- AccessKeySecret：表示Android/iOS应用初始化OSSClient获取AccessKeySecret。
- SecurityToken：表示Android/iOS应用初始化的Token。
- Expiration：表示该Token失效的时间。主要在Android SDK会自动判断是否失效，自动获取Token。

错误返回说明：

- StatusCode：表示获取Token的状态，获取失败时，返回值是500。
- ErrorCode：表示错误原因。
- ErrorMessage：表示错误的具体信息描述。

- 代码示例的运行方法：
  - 对于PHP版本，将包下载解压后，修改config.json这个文件，直接运行php sts.php 即能生成Token，将程序部署到指定的地址。
  - 对于JAVA版本 (依赖于java 1.7)，将包下载解压后

    运行方法：java -jar oss-token-server.jar (port)。如果不指定port（端口），直接运行java -jar oss-token-server.jar，程序会监听7080端口。如果想让程序执行在9000端口，运行java -jar app-token-server.jar 9000，其他端口也类似。

用户如何体验自己的APP上传应用服务器

1. 把程序部署起来后，记下应用服务器地址如`http://abc.com:8080` 将示例程序里面的应用服务器修改成上述地址。

2. 选择数据要上传到哪个Bcuket及区域，修改示例APP程序里面相应Bucket及区域。
3. 单击设置按钮，将配置加载。
4. 选择图片，设置上传OSS文件名，上传。这样就可以在Android上体验OSS服务了，就能通Android示例程序将数据直接上传到OSS了。
5. 上传成功后，可以看一下数据是否在OSS上。

核心代码解析

OSS初始化

- Android版本

```
// 推荐使用OSSAuthCredentialsProvider，token过期后会自动刷新。
String stsServer = "应用服务器地址，例如http://abc.com:8080"
OSSCredentialProvider credentialProvider = new OSSAuthCredentialsPr
ovider(stsServer);
//config
ClientConfiguration conf = new ClientConfiguration();
conf.setConnectionTimeout(15 * 1000); // 连接超时，默认15秒
conf.setSocketTimeout(15 * 1000); // socket超时，默认15秒
conf.setMaxConcurrentRequest(5); // 最大并发请求数，默认5个
conf.setMaxErrorRetry(2); // 失败后最大重试次数，默认2次
OSS oss = new OSSClient(getApplicationContext(), endpoint,
credentialProvider, conf);
```

- iOS版本

```
OSSClient * client;
...
// 推荐使用OSSAuthCredentialProvider，token过期后会自动刷新。
id<OSSCredentialProvider> credential = [[OSSAuthCredentialProvider
alloc] initWithAuthServerUrl:@"应用服务器地址，例如http://abc.com:8080
"];
client = [[OSSClient alloc] initWithEndpoint:endPoint credential
Provider:credential];
```

源码下载

示例程序

- Android示例程序的源码下载地址
- iOS示例程序的源码下载地址

应用服务器代码示例的下载

- PHP：下载地址
- Java：下载地址
- Ruby：下载地址

- node.js：下载地址

1.2 权限控制

本文基于*30分钟快速搭建移动应用直传服务* 中提到的应用服务器，以上海的Bucket app-base-oss为例，讲解如何配置不同的Policy实现不同的权限控制。

> 说明：
>
> - 以下说明中假设您已经开通了STS，并完全阅读了*30分钟快速搭建移动应用直传服务*。
> - 以下提到的Policy都是上文提到的config.json中指定的Policy文件的内容。
> - 以下讲述的获取STS Token 后对OSS的操作是指为应用服务器指定Policy，从STS获取临时凭证后，应用通过临时凭证访问OSS。

常见Policy

- 完全授权的Policy

  为了演示方便，默认Policy如下，表示允许应用可以对OSS进行任何操作。

  > 说明：
  > 这对移动应用来说也是不安全的授权，不推荐。

```json
{
  "Statement": [
    {
      "Action": [
        "oss:*"
      ],
      "Effect": "Allow",
      "Resource": ["acs:oss:*:*:*"]
    }
  ],
  "Version": "1"
}
```

获取**STS Token** 后对**OSS**操作	结果
列出所有创建的Bucket	成功
上传不带前缀的Object，test.txt	成功
下载不带前缀的Object, test.txt	成功
上传带前缀的Object, user1/test.txt	成功

获取**STS Token** 后对**OSS**操作	结果
下载带前缀的Object, user1/test.txt	成功
列出Object, test.txt	成功
带前缀的Object, user1/test.txt	成功

- 不限制前缀的只读不写Policy

  这个Policy表示，应用可以对Bucket app-base-oss下所有的Object可列举，可下载。

```
```json
  {
    "Statement": [
      {
        "Action": [
          "oss:GetObject",
          "oss:ListObjects"
        ],
        "Effect": "Allow",
        "Resource": ["acs:oss:*:*:app-base-oss/*", "acs:oss:*:*:app-
base-oss"]
      }
    ],
    "Version": "1"
  }
  ```
```

获取**STS Token** 后对**OSS**操作	结果
列出所有创建的Bucket	失败
上传不带前缀的Object，test.txt	失败
下载不带前缀的Object, test.txt	失败
上传带前缀的Object, user1/test.txt	失败
下载带前缀的Object, user1/test.txt	失败
列出Object, test.txt	失败
带前缀的Object, user1/test.txt	失败

- 限制前缀的只读不写Policy

  这个Policy表示，应用可以对Bucket app-base-oss下带有前缀user1/的Object可列举，可下载。但无法下载其他前缀的Object。这样不同的应用如果对应不同的前缀，就可以达到在同一个bucket中空间隔离的效果。

```
```json
  {
    "Statement": [
```
```

```
      {
        "Action": [
          "oss:GetObject",
          "oss:ListObjects"
        ],
        "Effect": "Allow",
        "Resource": ["acs:oss:*:*:app-base-oss/user1/*", "acs:oss
:*:*:app-base-oss"]
      }
    ],
    "Version": "1"
  }
  ```
```

| 获取**STS Token** 后对**OSS**操作 | 结果 |
| --- | --- |
| 列出所有创建的Bucket | 失败 |
| 上传不带前缀的Object，test.txt | 失败 |
| 下载不带前缀的Object, test.txt | 失败 |
| 上传带前缀的Object, user1/test.txt | 失败 |
| 下载带前缀的Object, user1/test.txt | 成功 |
| 列出Object, test.txt | 成功 |
| 带前缀的Object, user1/test.txt | 成功 |

- 不限制前缀的只写不读Policy

  这个Policy表示，应用可以对Bucket app-base-oss 下所有的Object进行上传。

```
  ```json
  {
    "Statement": [
      {
        "Action": [
          "oss:PutObject"
        ],
        "Effect": "Allow",
        "Resource": ["acs:oss:*:*:app-base-oss/*", "acs:oss:*:*:app-
base-oss"]
      }
    ],
    "Version": "1"
  }
  ```
```

| 获取**STS Token** 后对**OSS**操作 | 结果 |
| --- | --- |
| 列出所有创建的Bucket | 失败 |
| 上传不带前缀的Object，test.txt | 成功 |
| 下载不带前缀的Object, test.txt | 失败 |
```

获取**STS Token** 后对**OSS**操作	结果
上传带前缀的Object, user1/test.txt	成功
下载带前缀的Object, user1/test.txt	成功
列出Object, test.txt	成功
带前缀的Object, user1/test.txt	成功

- 限制前缀的只写不读Policy

  这个Policy表示，应用可以对Bucket app-base-oss 下带有前缀user1/的Object进行上传。但无法上传其他前缀的Object。这样不同的应用如果对应不同的前缀，就可以达到在同一个bucket中空间隔离的效果。

```
```json
  {
    "Statement": [
      {
        "Action": [
          "oss:PutObject"
        ],
        "Effect": "Allow",
        "Resource": ["acs:oss:*:*:app-base-oss/user1/*", "acs:oss
:*:*:app-base-oss"]
      }
    ],
    "Version": "1"
  }
  ```
```

获取**STS Token** 后对**OSS**操作	结果
列出所有创建的Bucket	失败
上传不带前缀的Object，test.txt	失败
下载不带前缀的Object, test.txt	失败
上传带前缀的Object, user1/test.txt	失败
下载带前缀的Object, user1/test.txt	成功
列出Object, test.txt	失败
带前缀的Object, user1/test.txt	失败

- 不限制前缀的读写Policy

这个Policy表示，应用可以对Bucket `app-base-oss`下所有的Object进行列举、下载、上传和删除。

```
```json
  {
    "Statement": [
      {
        "Action": [
          "oss:GetObject",
          "oss:PutObject",
          "oss:DeleteObject",
          "oss:ListParts",
          "oss:AbortMultipartUpload",
          "oss:ListObjects"
        ],
        "Effect": "Allow",
        "Resource": ["acs:oss:*:*:app-base-oss/*", "acs:oss:*:*:app-
base-oss"]
      }
    ],
    "Version": "1"
  }
  ```
```

获取**STS Token** 后对**OSS**操作	结果
列出所有创建的Bucket	失败
上传不带前缀的Object，test.txt	成功
下载不带前缀的Object, test.txt	成功
上传带前缀的Object, user1/test.txt	成功
下载带前缀的Object, user1/test.txt	成功
列出Object, test.txt	成功
带前缀的Object, user1/test.txt	成功

- 限制前缀的读写Policy

  这个Policy表示，应用可以对Bucket `app-base-oss`下带有前缀`user1/`的Object进行列举、下载、上传和删除。但无法对其他前缀的Object进行读写。这样不同的应用如果对应不同的前缀，就可以达到在同一个bucket中空间隔离的效果

```
{
    "Statement": [
      {
        "Action": [
          "oss:GetObject",
          "oss:PutObject",
          "oss:DeleteObject",
          "oss:ListParts",
          "oss:AbortMultipartUpload",
```

```
            "oss:ListObjects"
          ],
          "Effect": "Allow",
          "Resource": ["acs:oss:*:*:app-base-oss/user1/*", "acs:oss
:*:*:app-base-oss"]
        }
      ],
      "Version": "1"
    }
```

获取**STS Token** 后对**OSS**操作	结果
列出所有创建的Bucket	失败
上传不带前缀的Object，test.txt	失败
下载不带前缀的Object, test.txt	失败
上传带前缀的Object, user1/test.txt	成功
下载带前缀的Object, user1/test.txt	成功
列出Object, test.txt	成功
带前缀的Object, user1/test.txt	成功

小结

从上面的例子可以看出：

- 可以根据不同的应用场景制定不同的Policy，然后对应用服务器稍作修改就可以实现对不用的应用用户实现不同的权限控制。
- 也可以在应用端做优化，在STS Token过期之前不需要向应用服务器再次请求。
- Token实际是由STS颁发的，应用服务器相当于定制了Policy，向STS请求Token，然后将Token转发给应用。这里的Token是一个简略的说法。实际上包含了“AccessKeyId”、“AccessKeySecret”、“Expiration”、“SecurityToken”，这些在OSS提供给应用的SDK中会用到。详细参见各个SDK的实现。

更多参考资料

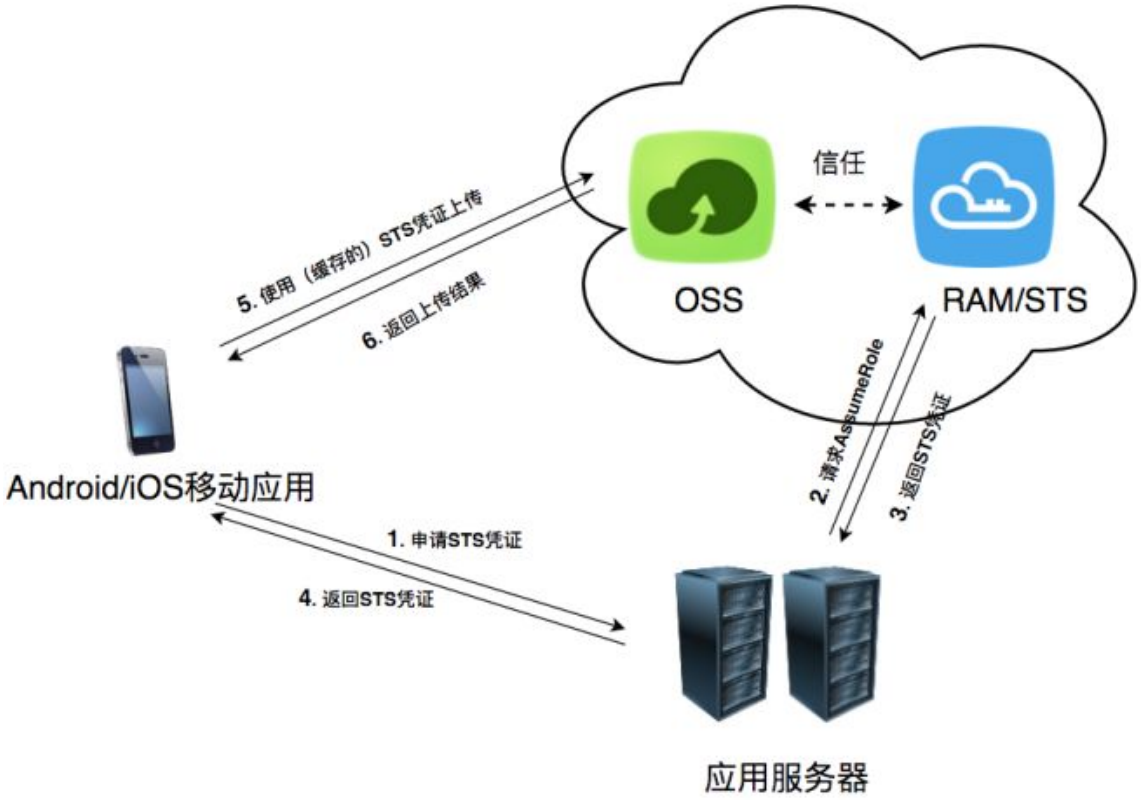
- *RAM*和*STS*在*OSS*中的使用指南
- *RAM*官方文档和*STS*官方文档

1.3 快速搭建移动应用上传回调服务

背景

快速搭建移动应用直传服务介绍了如何在30分钟内中搭建一个基于OSS的移动应用数据直传服务。

移动端开发场景流程图如下：

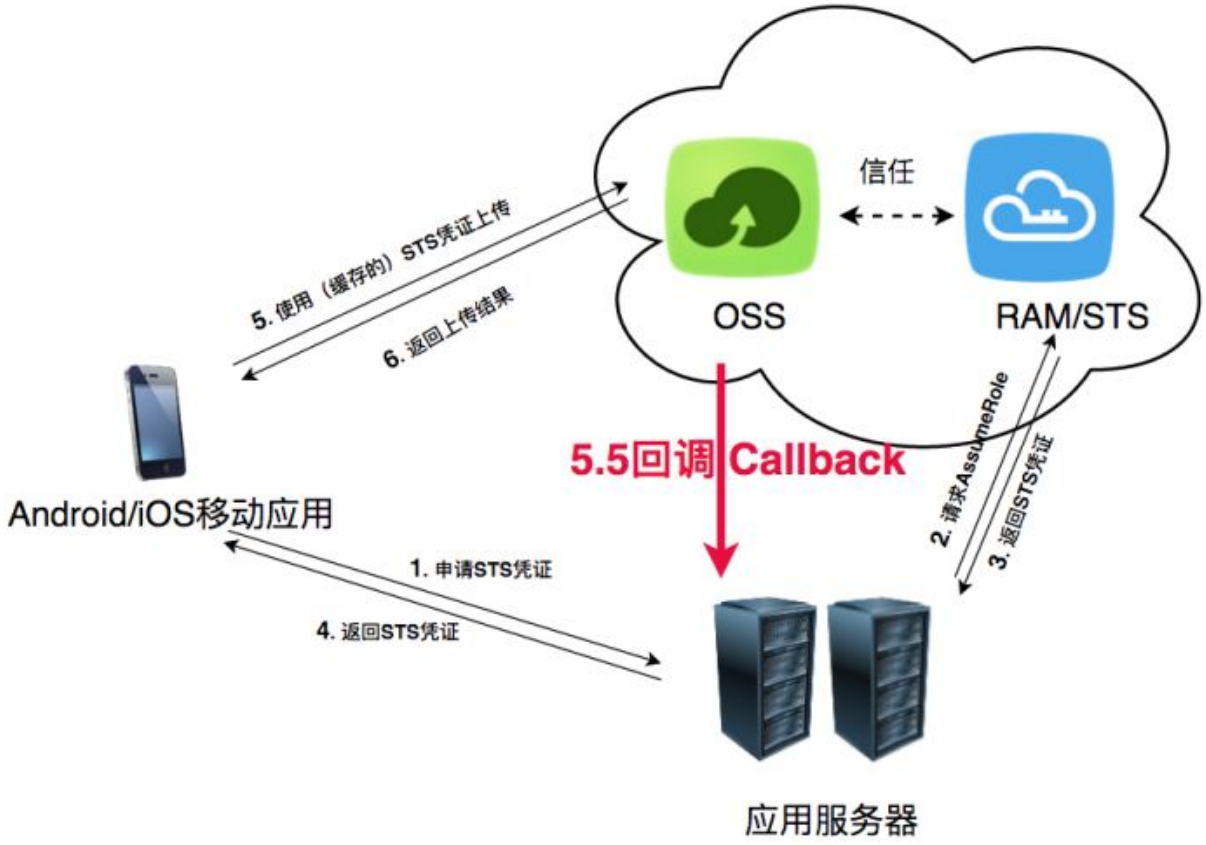


角色分析如下所示：

- 应用服务器负责为Android/iOS移动应用生成STS凭证。
- Android/iOS移动应用负责从应用服务器申请及使用STS凭证。
- OSS负责处理移动应用的数据请求。

对于Android/iOS移动应来说，移动应用只需要执行上图中操作1（申请STS凭证），就能调用多次5(使用该STS凭证上传数据到OSS)，导致应用服务器根本不知道用户都上传了哪些数据 ，作为该APP的开发者，就没法对应用上传数据进行管理。有什么问题能让应用服务器感知到Android/iOS移动应用上传的数据呢？

您可以通过使用OSS的上传回调服务，就能解决上述问题，如下图所示：



OSS在收到Android/iOS移动的数据（上图中操作5）和在返回用户上传结果（上图中操作6）之间，触发一个上传回调工作。即第上图中操作5.5，先回调用户服务器，得到应用服务器返回的内容，将这个内容返回给Android/iOS移动应用。可以参考*Callback API*文档。

上传回调的作用

- 通过上传回调可以让用户应用服务器知道当前上传文件的基本信息。

  基本信息如下表。返回下述变量的一个或者多个，返回内容格式形式在Android/iOS上传时指定。

系统变量	含义
bucket	移动应用上传到哪个存储空间
object	移动应用上传到OSS保存的文件名
etag	该上传的文件的etag，即返回给用户的etag字段
size	该上传的文件的大小
mimeType	资源类型

系统变量	含义
imageInfo.height	图片高度
imageInfo.width	图片宽度
imageInfo.format	图片格式，如jpg、png，只以识别图片

- 通过上传回调设定自定义参数，达到信息传递目的。

  假如您是一个开发者，您想知道当前用户所使用的APP版本、当前用户所在的操作系统版本、用户的GPS信息、用户的手机型号。您可以在Android/iOS端上传文件时，指定上述自定义参数，如下所示：

  - x:version指定APP版本
  - x:system指定操作系统版本
  - x:gps指定GPS信息
  - x:phone指定手机型号

    上述这些值会在Android/iOS移动应用上传到OSS时附带上，OSS会把这些值放到CallbackBody里面一起发给应用服务器。这样应用服务器就能收到这些信息，达到信息传递的目的。

在移动应用端设定上传回调

要让OSS在接收上传请求时，触发上传回调，移动应用在构造上传请求时必须把如下两个内容指定到上传请求里面：

- 要回调到哪个服务器callbackUrl，如 `http://abc.com/callback.php`，这个地址必须是公网能够访问的。
- 上传回调给应用服务器的内容callbackBody，可以是上述OSS返回应用服务器系统变量的一个或者多个。

假如您的用户服务器上传回调地是`http://abc.com/callback.php`。您想获取手机上传的文件名字、文件的大小，并且定义了photo变量是指手机型号，system是指操作系统版本。

上传回调示例分以下两种：

- iOS指定上传回调示例：

```
OSSPutObjectRequest * request = [OSSPutObjectRequest new];
request.bucketName = @"<bucketName>";
request.objectKey = @"<objectKey>";
request.uploadingFileURL = [NSURL fileURLWithPath:@<filepath>"];
// 设置回调参数
request.callbackParam = @{
```

```
                            @"callbackUrl": @"http://abc.com/callback.
php",
                            @"callbackBody": @"filename=${object}&size
=${size}&photo=${x:photo}&system=${x:system}"
                            };
// 设置自定义变量
request.callbackVar = @{
                        @"x:photo": @"iphone6s",
                        @"x:system": @"ios9.1"
                        };
```

- Android指定上传回调示例：

```
PutObjectRequest put = new PutObjectRequest(testBucket, testObject,
uploadFilePath);
ObjectMetadata metadata = new ObjectMetadata();
metadata.setContentType("application/octet-stream");
put.setMetadata(metadata);
put.setCallbackParam(new HashMap<String, String>() {
    {
        put("callbackUrl", "http://abc.com/callback.php");
        put("callbackBody", "filename=${object}&size=${size}&photo=
${x:photo}&system=${x:system}");
    }
});
put.setCallbackVars(new HashMap<String, String>() {
     {
         put("x:photo", "IPOHE6S");
         put("x:system", "YunOS5.0");
     }
});
```

上传回调对应用服务器的要求

- 您必须部署一个可以接收POST请求的服务，这个服务必须有公网地址如`www.abc.com/callback.php`（或者外网IP也可以），不然OSS没有办法访问到这个地址。
- 您要给OSS正确的返回，返回格式必须是JSON格式，内容自定义。因为OSS会把应用服务器返回的内容，原封不动地返回给Android/iOS移动应用。（切记，返回给OSS的Response Header一定要加上Content-Length这个头部）。

本教程在最后为大家准备了多个语言版本的示例、下载及运行方法。

应用服务器收到的回调请求

应用服务器收到OSS的请求，抓包的请求如下（这个结果根据设定的不同URL和回调内容会有不同）：

```
POST /index.html HTTP/1.0
Host: 121.43.113.8
Connection: close
Content-Length: 81
Content-Type: application/x-www-form-urlencoded
User-Agent: ehttp-client/0.0.1
```

```
authorization: kKQeGTRccDKyHB3H9vF+xYMSrmhMZjzzl2/kdD1ktNVgbWEfYTQG0G2
SU/RaHBovRCE8OkQDjC3uG33esH2txA==
x-oss-pub-key-url: aHR0cDovL2dvc3NwdWJsaWMuYWxpY2RuLmNvbS9jYWxsYmFja1
9wdWJfa2V5X3YxLnBlbQ==
filename=test.txt&size=5&photo=iphone6s&system=ios9.1
```

更多内容请参考*Callback API*文档。

应用服务器判断回调请求是否来自OSS

如果您的回调服务器被人恶意攻击了，例如恶意回调您的应用服务器，导致应用服务器收到一些非法的请求，影响正常逻辑，此时您就需要判断回调请求是否来自OSS。

判断的方法主要是利用OSS给应用服务器返回的头部内容中，`x-oss-pub-key-url`和`authorization`这两个参数进行RSA校验。只有通过RSA校验才能说明这个请求是来自OSS，本教程提供的示例程序都有实现的示例供您参考。

应用服务器收到回调请求后的处理

应用服务器在校验这个请求是来自OSS后，指定回调给应用服务器的内容格式，如

```
filename=test.txt&size=5&photo=iphone6s&system=ios9.1
```

应用服务器就可以根据OSS的返回内容，解析得到自己想要得到的数据。得到这个数据后，应用服务器可以把数据存放起来，方便后续管理。

应用服务器收到回调请求后如何返回给OSS

- 返回状态码是200；
- 返回必须是json格式的内容；
- 返回的头部必须带有Content-Length这个头部。

OSS如何处理应用服务器的返回内容

有两种情况：

- OSS将回调请求发送给应用服务器，但是应用服务器接收失败或者访问不通，OSS会返回给Android/iOS移动应用203的状态码，但是数据已经存放到OSS上了。
- 应用服务器接收到OSS的回调请求，并且正确返回了，OSS会返回给Android/iOS移动应用状态码是200, 并把应用服务器给OSS的内容，原封不动地返回给Android/iOS移动应用。

上传回调服务器示例程序下载

示例程序只是完成了如何检查应用服务器收到的签名， 用户要自行增加对应用服务器收到回调的内容的格式解析 。

- Java版本：
  - 下载地址：单击这里。
  - 运行方法，解压包运行`java -jar oss-callback-server-demo.jar 9000`（9000是运行的端口，可以自己指定）。

    说明：

    这个jar例子在java 1.7运行通过，如果有问题可以自己依据提供的代码进行修改。这是一个maven项目。
- PHP版本：
  - 下载地址：单击这里
  - 运行方法：部署到Apache环境下，因为PHP本身语言的特点，取一些数据头部会依赖于环境。所以可以参考例子根据自己所在环境进行修改。
- Python版本：
  - 下载地址：单击这里。
  - 运行方法：解压包直接运行python callback_app_server.py即可，程序自实现了一个简单的http server，运行该程序可能需要安装rsa的依赖。
- Ruby版本：
  - 下载地址：单击这里。
  - 运行方法： ruby aliyun_oss_callback_server.rb。

2 Web端直传实践

2.1 Web端直传实践简介

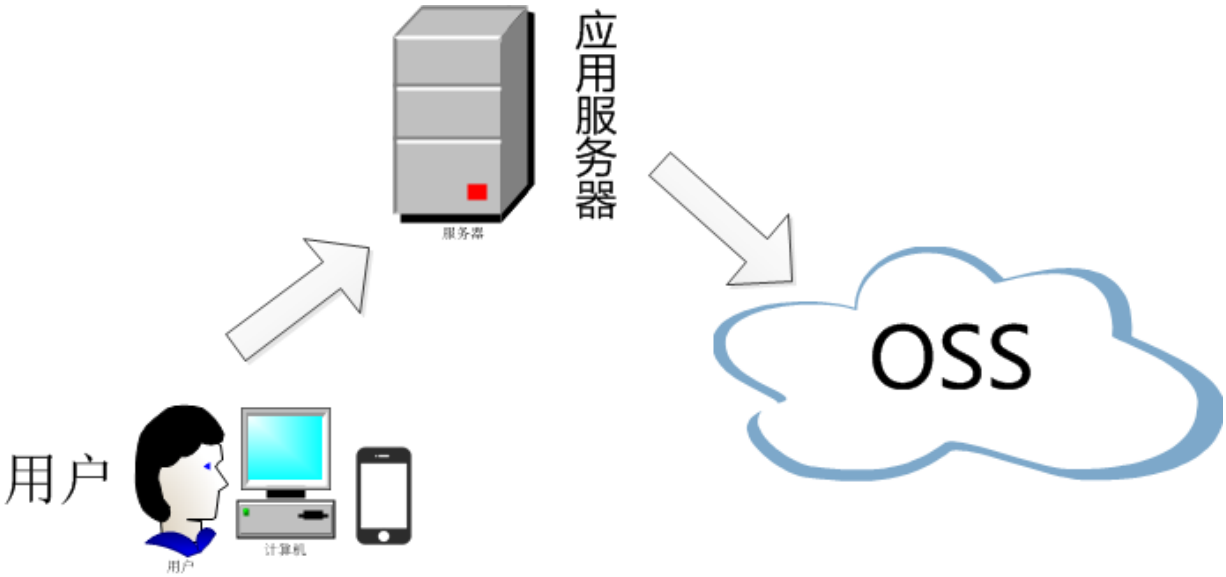
目的

本教程的目的是通过两个例子介绍如何在Html表单提交直传OSS。

- 第一个例子：讲解签名在服务端（php）完成，然后直接通过表单上传到OSS。
- 第二个例子：讲解签名在服务端（php）完成，并且服务端设置了上传后回调。然后直接通过表单上传到OSS，OSS回调完应用服务器再返回给用户。

背景

每个OSS的用户，都会用到上传。由于是网页上传，其中包括一些APP里面的h5页面，对上传的需求很强烈，很多人采用的做法是用户在浏览器/APP上传到应用服务器，然后应用服务器再把文件上传到OSS。



这种方法有三个缺点：

- 第一：上传慢。先上传到应用服务器，再上传到OSS，网络传送多了一倍。如果数据直传到OSS，不走应用服务器，速度将大大提升，而且OSS是采用BGP带宽，能保证各地各运营商的速度。
- 第二：扩展性不好。如果后续用户多了，应用服务器会成为瓶颈。
- 第三：费用高。由于OSS上传流量是免费的。如果数据直传到OSS，不走应用服务器，那么将能省下几台应用服务器。

基础篇：应用服务器**php**返回签名

单击这里打开示例

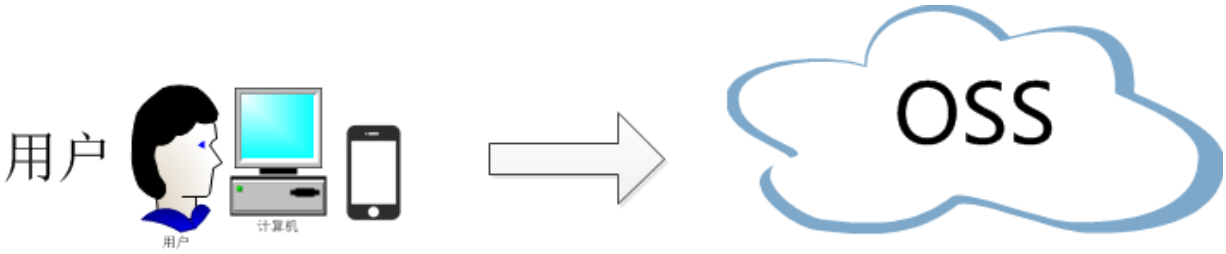
进阶篇：应用服务器**php**返回签名及采用上传回调

单击这里打开示例

2.2 JavaScript客户端签名直传

背景

客户端用JavaScript直接签名，然后上传到OSS。请参考 *Web端直传实践* 中的背景介绍。



示例

下面将介绍用*plupload* 在JavaScript端签名然后直传数据到OSS的例子。用户电脑浏览器测试样例：点击这里打开示例

用手机测试该上传是否有效。二维码：可以用手机（微信，QQ，手机浏览器等）扫一扫试试（这个不是广告，只是上述网址的二维码，为了让大家看一下这个实现能在手机端完美运行。）

文件上传是上传到一个测试的公共 bucket，会定时清理，所以不要传一些敏感及重要数据。

代码下载

点击这里：*oss-h5-upload-js-direct.zip*

原理

- 本例子的功能
  - 采用plupload 直接提交表单数据（即PostObject）到OSS；
  - 支持html5,flash,silverlight,html4 等协议上传；
  - 可以运行在PC浏览器、手机浏览器、微信等；
  - 可以选择多文件上传；
  - 显示上传进度条；
  - 可以控制上传文件的大小；

— 可以设置上传到指定目录和设置上传文件名字是随机文件名还是本地文件名。

OSS的PostObject API细节可以参照*PostObject*。

- plupload

  *plupload*是一款简单易用且功能强大的文件上传工具， 支持多种上传方式，包括html5, flash, silverlight, html4。会智能检测当前环境，选择最适合的方式，并且会优先采用Html5。

- 关键代码

  因为OSS支持POST协议。所以只要将plupload在发送POST请求时，带上OSS签名即可。核心代码如下：

```
var uploader = new plupload.Uploader({
    runtimes : 'html5,flash,silverlight,html4',
    browse_button : 'selectfiles',
    //runtimes : 'flash',
    container: document.getElementById('container'),
    flash_swf_url : 'lib/plupload-2.1.2/js/Moxie.swf',
    silverlight_xap_url : 'lib/plupload-2.1.2/js/Moxie.xap',
    url : host,
    multipart_params: {
        'Filename': '${filename}',
        'key' : '${filename}',
        'policy': policyBase64,
        'OSSAccessKeyId': accessid,
        'success_action_status' : '200', //让服务端返回200,不然，默认会
返回204
        'signature': signature,
    },
  ....
}
```

  在这里有一点请注意一下，就是’Filename’: ‘${filename}’， 这一段代码的作用是表示上传后保持原来的文件文字。如果您想上传到特定目录如abc下，文件名保持成原来的文件名，那么应该这样写：

```
multipart_params: {
        'Filename': 'abc/' + '${filename}',
        'key' : '${filename}',
        'policy': policyBase64,
        'OSSAccessKeyId': accessid,
        'success_action_status' : '200', //让服务端返回200，不然，默认会
返回204
        'signature': signature,
    },
```

- 设置成随机文件名

有时候要把用户上传的文件，设置成随机文件名，后缀保持跟客户端文件一致。例子里面，通过两个radio来区分, 如果想在上传时就固定设置成随机文件名，可以将函数改成如下：

```
function check_object_radio() {
    g_object_name_type = 'random_name';
}
```

如果想在上传时，固定设置成用户的文件，可以将函数改成：

```
function check_object_radio() {
    g_object_name_type = 'local_name';
}
```

- 设置上传目录

  可以将文件上传到指定目录下面， 目录相关设置可以在例子中体验，如果想让代码上传到固定目录如abc, 可以按如下代码改造，注意要以’/‘ 结尾。

```
function get_dirname()
{
    g_dirname = "abc/";
}
```

- 上传签名

  签名signature主要是对policyText进行签名，最简单的例子如下：

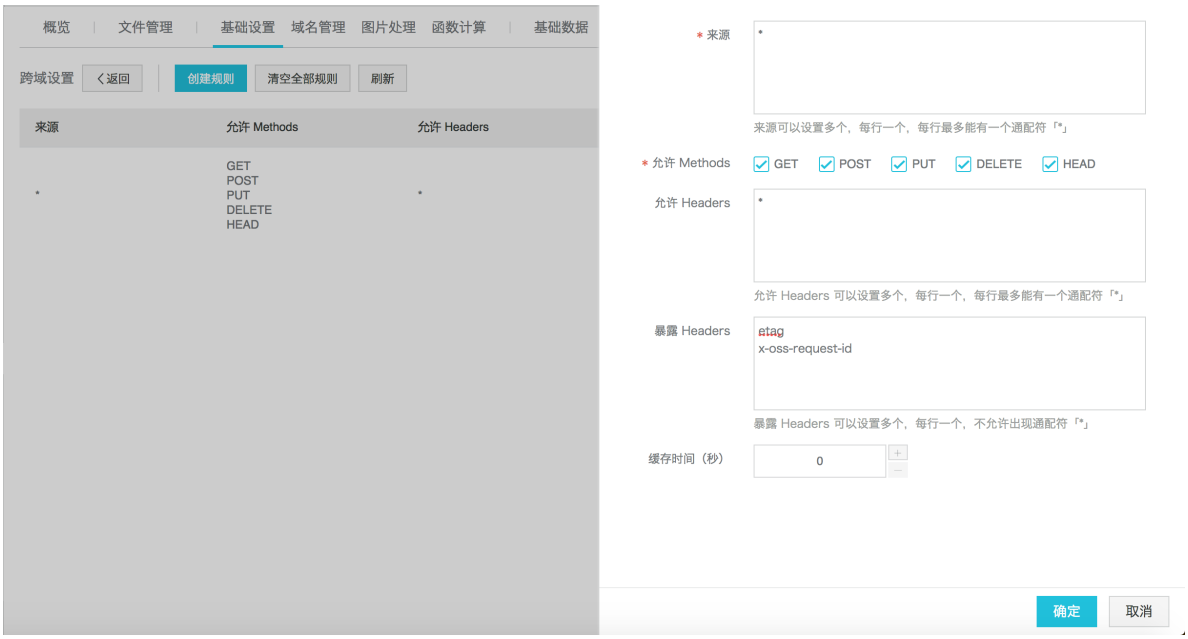
```
var policyText = {
    "expiration": "2020-01-01T12:00:00.000Z", //        设置该Policy的失效时间，超过这个失效时间之后，就没有办法通过这个policy上传文件了
    "conditions": [
    ["content-length-range", 0, 1048576000] // 设置上传文件的大小限制,如果超过了这个大小，文件上传到OSS会报错的
    ]
}
```

- 跨域CORS

  说明：

  一定要保证bucket属性CORS设置支持POST方法。因为这个HTML直接上传到OSS，会产生跨域请求。必须在bucket属性里面设置允许跨域。

  设置如下图：



说明：

在IE低版本浏览器，plupload会以flash方式执行。必须设置crossdomain.xml ，设置方法可以参考：点击这里

- 注意

  把AccesssKeyID 和AccessKeySecret写在代码里面有泄露的风险。建议采用后端签名上传的方案：服务端签名后网页直传

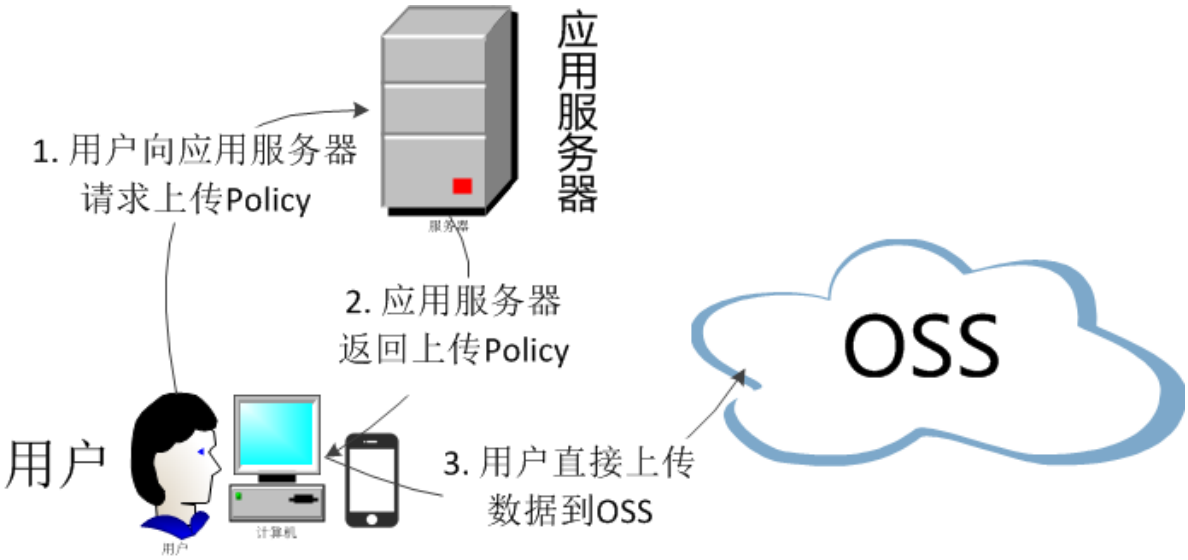
2.3 服务端签名后直传

背景

采用JS客户端直接签名有一个很严重的安全隐患，OSS AccessId/AccessKey暴露在前端页面，其他人可以随意拿到AccessId/AccessKey，这是非常不安全的做法。 本文将讲述如何从后端php代码中取到签名及上传policy。

后端上传签名逻辑图如下：

1. 客户端要上传图片时，到应用服务器取上传的policy及签名。
2. 客户端拿到签名直接上传到OSS。



后端上传签名示例

示例下载：

- 电脑浏览器测试样例：单击下载
- 手机测试该上传是否有效。可以用手机（微信、QQ、手机浏览器等）扫下图二维码（这个不是广告，只是上述网址的二维码。这为了让大家看一下这个实现能在手机端完美运行）。

代码下载：

单击下载：*oss-h5-upload-js-php.zip*

例子采用后端签名，语言是用PHP。

- Java后端的例子：单击这里
- Python的例子：单击这里
- Go的例子： 单击这里

其他语言的用法：

1. 下载对应的语言示例。
2. 修改示例代码，如设置监听的端口等，然后运行。
3. 在oss-h5-upload-js-php.zip里面的upload.js， 将里面的变量serverUrl改成第二步部署的地址。如serverUrl = `http://1.2.3.4:8080`或者serverUrl=`http://abc.com/post/`

服务端构造Post签名原理

上传采用OSS PostObject方法， 用plupload在浏览器构造PostObject请求，发往OSS。签名在服务端实现，即在（PHP）完成，相同道理，服务端可以以JAVA、.NET、Ruby、GO、python等语言编写，核心逻辑就是构造Post签名。本例子提供了JAVA、PHP例子，需要以下步骤：

1. 网页通过JS向服务端请求签名。
2. JS获取到签名后，通过plupload 上传到OSS。

实现方法如下：

1. 设置成自己的id、key、bucket。

   修改php/get.php：

   - 将变量$id设成AccessKeyId
   - $key设置成AccessKeySecret
   - $host设置成：bucket+endpoint

   > 说明：
   > 关于endpoint，请参见 基本概念介绍。

   ```
   $id= 'xxxxxx';
     $key= 'xxxxx';
     $host = 'http://post-test.oss-cn-hangzhou.aliyuncs.com
   ```

2. 为了浏览安全，必须为bucket设置CORS。

   > 说明：
   > 一定要保证bucket属性CORS设置支持POST方法。因为这个HTML直接上传到OSS，会产生跨域请求。必须在bucket属性里面设置允许跨域。

   具体操作步骤请参见 设置跨域访问。设置如下图：

说明：

在IE低版本浏览器，plupload会以flash方式执行。必须设置crossdomain.xml ，设置方法可以参考：单击这里

核心逻辑详解

设置随机文件名

有时候要把用户上传的文件，设置成随机文件名，后缀保持跟客户端文件一致。例子里面，通过两个radio来区分, 如果想在上传时，就固定设置成随机文件名，可以将函数改成如下：

```
function check_object_radio() {
    g_object_name_type = 'random_name';
```

```
}
```

如果想在上传时，固定设置成用户的文件，可以将函数改成：

```
function check_object_radio() {
    g_object_name_type = 'local_name';
}
```

设置上传目录

上传的目录是由服务端（即PHP）指定的，这样的好处就是安全。 每个客户端只能上传到指定的目录，实现安全隔离。下面的代码是将上传目录地址改成abc/(必须以'/'结尾)。

```
$dir = 'abc/';
```

设置上传文件过滤条件

有时候需要设置上传的过滤条件，如可以设置只能上传图片、上传文件的大小、不能有重复上传等。这时可以利用filters参数。

```
var uploader = new plupload.Uploader({
    ......
    filters: {
        mime_types : [ //只允许上传图片和zip文件
        { title : "Image files", extensions : "jpg,gif,png,bmp" },
        { title : "Zip files", extensions : "zip" }
        ],
        max_file_size : '400kb', //最大只能上传400kb的文件
        prevent_duplicates : true //不允许选取重复文件
    },
```

设置过滤条件原理是利用plupload 的属性filters来设置。

上述值的设置含义：

- mime_types：限制上传的文件后缀
- max_file_size: 限制上传的文件大小
- prevent_duplicates： 限制不能重复上传

> **说明：**
> filters过滤条件不是必须的。如果不想设置过滤条件，只要把该项注释即可。

获取上传后的文件名

如果要知道文件上传成功后的文件名，可以用plupload调用FileUploaded事件获取，如下：

```
FileUploaded: function(up, file, info) {
            if (info.status == 200)
```

```
            {
                document.getElementById(file.id).getElementsByTagName
('b')[0].innerHTML = 'upload to oss success, object name:' +
get_uploaded_object_name(file.name);
            }
            else
            {
                document.getElementById(file.id).getElementsByTagName
('b')[0].innerHTML = info.response;
            }
    }
```

可以利用如下函数，得到上传到oss的文件名，其中file.name记录了上传本地文件的名字。

```
get_uploaded_object_name(file.name)
```

上传签名

javaScript可以从后端取到policyBase64、accessid、signature这三个变量，取这三个变量核心代码如下：

```
phpUrl = './php/get.php'
        xmlhttp.open( "GET", phpUrl, false );
        xmlhttp.send( null );
        var obj = eval ("(" + xmlhttp.responseText+ ")");
        host = obj['host']
        policyBase64 = obj['policy']
        accessid = obj['accessid']
        signature = obj['signature']
        expire = parseInt(obj['expire'])
        key = obj['dir']
```

解析xmlhttp.responseText（以下仅为示例，并不一定要求是以下的格式，但是必须有signature、accessid、policy这三个值）。

```
{"accessid":"6MKOqxGiGU4AUk44",
"host":"http://post-test.oss-cn-hangzhou.aliyuncs.com",
"policy":"eyJleHBpcmF0aW9uIjoiMjAxNS0xMS0wNVQyMDoyMzoyM1oiLCJjxb25kaXR
pb25zIjpbWyJjcb250ZW50LWxlbmd0aC1yYW5nZSIsMCwxMDQ4NTc2MDAwXSxbInN0YXJ0
cy13aXRoIiwiJGtleSIsInVzZXItZGlyXC8iXV19",
"signature":"I2u57FWjTKqX/AE6doIdyff151E=",
"expire":1446726203,"dir":"user-dir/"}
```

- accessid: 指的用户请求的accessid。注意仅知道accessid, 对数据不会有影响。
- host: 指的是用户要往哪个域名发往上传请求。
- policy：指的是用户表单上传的策略policy，是经过base64编码过的字符串。
- signature：是对上述第三个变量policy签名后的字符串。
- expire：指的是当前上传策略失效时间，这个变量并不会发送到OSS，因为这个已经指定在policy里面，这个变量的含义，后面讲

解析policy的内容，将其解码后的内容是：

```
{"expiration":"2015-11-05T20:23:23Z",
"conditions":[["content-length-range",0,1048576000],
["starts-with","$key","user-dir/"]]
```

这里有一个关键的地方，PolicyText指定了该Policy上传失效的最终时间。即在这个失效时间之前，都可以利用这个policy上传文件，所以没有必要每次上传，都去后端取签名。

为了减少后端的压力，设计思路是：初始化上传时，每上传一个文件后，取一次签名。然后再上传时，将当前时间跟签名时间对比，看签名时间是否失效了。如果失效了，就重新取一次签名，如果没有失效就不取。这里就用到了变量expire。 核心代码如下：

```
now = timestamp = Date.parse(new Date()) / 1000;
[color=#000000]//可以判断当前expire是否超过了当前时间，如果超过了当前时间，就重新取一下，3s 做为缓冲[/color]
    if (expire < now + 3)
{
      .....
      phpUrl = './php/get.php'
      xmlhttp.open( "GET", phpUrl, false );
      xmlhttp.send( null );
      ......
}
return .
```

上面policy的内容增加了starts-with，用来指定此次上传的文件名必须是user-dir开头（这个字符串，用户可以自己指定）。

增加这个内容的背景是：在很多场景下，一个应用一个bucket，不同用户的数据，为了防止数字覆盖，每个用户上传到OSS的文件都可以有特定的前缀。那么问题来了，用户获取到这个policy后，在失效期内都能修改上传前缀，从而上传到别人的目录下。为了解决这个问题，可以设置应用服务器在上传时就指定用户上传的文件必须是某个前缀。这样如果用户拿到了policy也没有办法上传到别人的前缀上。保证了数据的安全性。

总结

本文档示例主要讲述网页端上传时，网页端向服务端请求签名，然后直接上传，不用对服务端产生压力。而且安全可靠。但是这个例子有一个特点，就是用户上传了多少文件，用户上传了什么文件，用户后端程序并不能马上知道，如果想实时知晓用户上传了什么文件，可以采用上传回调。本例子无法实现分片与断点。

相关文档

- 基本概念

- 设置跨域访问
- *Web*端直传实践简介
- *JavaScript*客户端签名直传
- 进阶篇#应用服务器*php*返回签名及采用上传回调
- 移动应用端直传实践

2.4 服务端签名直传并设置上传回调

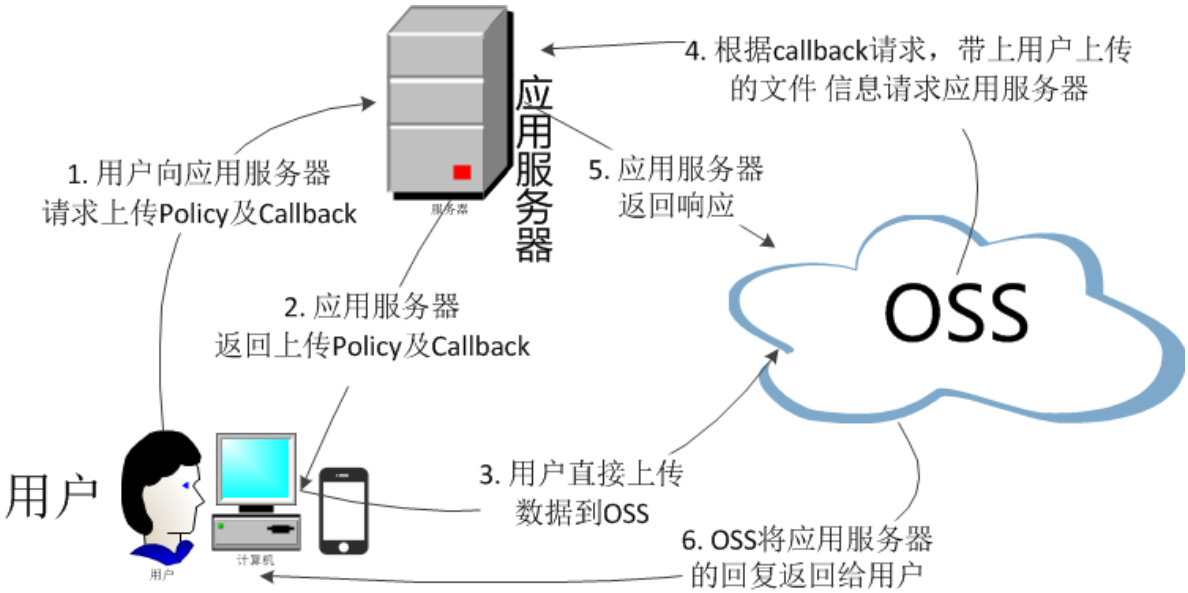
背景

请参考 *Web*端直传实践 里的背景介绍。

当采用服务端签名后直传方案后，问题来了，用户上传数据后，很多场景下，应用服务器都要知道用户上传了哪些文件，文件名字，甚至如果是图片的话，图片的大小等。为此OSS开发了上传回调功能。

用户的请求逻辑

1. 用户向应用服务器取到上传policy和回调设置。
2. 应用服务器返回上传policy和回调。
3. 用户直接向OSS发送文件上传请求。
4. 等文件数据上传完，OSS给用户Response前，OSS会根据用户的回调设置，请求用户的服务器。
5. 如果应用服务器返回成功，那么就返回用户成功，如果应用服务器返回失败，那么OSS也返回给用户失败。这样确保了用户上传成功的照片，应用服务器都已经收到通知了。
6. 应用服务器给OSS返回。
7. OSS将应用服务器返回的内容返回给用户。



简单讲，就是用户要上载一个文件到OSS服务器，而且希望上载完毕的时候自己的应用服务能够知道这件事，这时就需要设置一个回调函数，把这件事告知用户的应用服务器。这样当OSS收到用户的上传请求之后，开始上传，传完之后不会直接给用户返回结果，而是先通知用户的应用服务器：“我上传完毕了”，然后应用服务器告诉OSS：“我知道啦，你帮我转达给我的主人吧”，于是OSS就把结果转达给用户了。

示例

用户电脑浏览器测试样例：点击这里体验上传回调示例

用手机测试该上传是否有效。可以用手机(微信、QQ、手机浏览器等）扫一扫二维码试试（这个不是广告，只是上述网址的二维码，为了让大家看一下这个实现能在手机端完美运行）。

代码下载

点击这里：*oss-h5-upload-js-php-callback.zip*

例子是采用后端签名，语言是用PHP。

- 采用Java语言后端签名的示例：点击这里
- Go语言示例： 点击这里
- Python语言示例： 点击这里

其他语言的用法：

1. 下载对应的语言示例。
2. 然后修改示例代码，如设置监听的端口等，然后运行。

3. 在oss-h5-upload-js-php-callback.zip里面的upload.js， 将里面的变量severUrl改成第2步部署的地址。如severUrl = http://1.2.3.4:8080 或者serverUrl= http://abc.com/post/

快速使用

只要以下三步，就能实现文件快速通过网页上传到OSS，并且OSS会回调通知到用户设置的应用服务器。

1. 设置成自己的 id、key、bucket。

   设置方法：修改php/get.php，将变量$id设成AccessKeyId，$key设置成AccessKeySecret，$host设置成 bucket+endpoint。

```
$id= 'xxxxxx';
 $key= 'xxxxx';
 $host = 'http://post-test.oss-cn-hangzhou.aliyuncs.com
```

2. 为了浏览安全，必须为bucket设置Cors, 参照下文。
3. 设置自己的回调URL，如http://abc.com/test.html (必须公网访问得通),即自己的回调服务器地址，OSS会在文件上传完成后，把文件上传信息，通过自己设置的回调URL(http://abc.com/test.html)发送给应用服务器。设置方法：修改php/get.php，（这个回调服务端代码实例参考下文）

```
$callbackUrl = "http://abc.com/test.html";
```

这个例子里面更多细节 ，如上传签名，设置随机文件名等更多细节可以参照：点击这里#了解上传更多细节。

下面讲解一下核心逻辑。

核心代码解析

代码要添加的内容如下：

```
new_multipart_params = {
      'key' : key + '${filename}',
      'policy': policyBase64,
      'OSSAccessKeyId': accessid,
      'success_action_status' : '200', //让服务端返回200,不然，默认会返回204
      'callback':   callbackbody,
      'signature': signature,
 };
```

上述的callbackbody 是php服务端返回的。在本例中，从后端php取到的内容如下：

```
{"accessid":"6MKOqxGiGU4AUk44",
```

```
"host":"http://post-test.oss-cn-hangzhou.aliyuncs.com",
"policy":"eyJleHBpcmF0aW9uIjoiMjAxNS0xMS0wNVQyMDo1MjoyOVoiLCJjdb25kaXR
pb25zIjpbWyJjdb250ZW50LWxlbmd0aC1yYW5nZSIsMCwxMDQ4NTc2MDAwXSxbInN0YXJ0
cy13aXRoIiwiJGtleSIsInVzZXItZGlyXC8iXV19",
"signature":"VsxOcOudxDbtNSvz93CLaXPz+4s=",
"expire":1446727949,
"callback":"eyJjYWxsYmFja1VybCI6Imh0dHA6Ly9vc3MtZGVtby5hbGl5dW
5jcy5jdb206MjM0NTAiLCJjYWxsYmFja0hvc3QiOiJvc3MtZGVtby5hbGl5dW5jcy5jdb2
0iLCJjYWxsYmFja0JvZHkiOiJmaWxlbmFtZT0ke29iamVjdH0mc2l6ZT0ke3NpemV9Jm1p
bWVUeXBlPSR7bWltZVR5cGV9JmhlaWdodD0ke2ltYWdlSW5mby5oZWlnaHR9JndpZHRoPS
R7aW1hZ2VJdbmZvLndpZHRofSIsImNhbGxiYWNrQm9keVR5cGUiOiJhcHBsaWNhdGlvbi9
4LXd3dy1mb3JtLXVybGVuY29kZWQifQ==","dir":"user-dirs/"}
```

上面提到callbackbody，就是上述返回结果里面的callback内容，经过base64编码后生成的。

解码后的内容如下：

```
{"callbackUrl":"http://oss-demo.aliyuncs.com:23450",
"callbackHost":"oss-demo.aliyuncs.com",
"callbackBody":"filename=${object}&size=${size}&mimeType=${mimeType}&
height=${imageInfo.height}&width=${imageInfo.width}",
"callbackBodyType":"application/x-www-form-urlencoded"}
```

内容的解析如下：

- CallbackUrl: OSS往这个机器发送的url请求。
- callbackHost: OSS发送这个请求时，请求头部所带的Host头。
- callbackBody: OSS请求时，发送给应用服务器的内容，可以包括文件的名字、大小、类型，如果是图片可以是图片的高度、宽度。
- callbackBodyType: 请求发送的Content-Type。

回调应用服务器

在用户的请求逻辑中，很重要的地方就是第4步和第5步，OSS与应用服务器交互的时候，用户可能会有以下疑问：

- 问题1：如果我是开发者，那么我要怎么样确认请求是从OSS发送过来的呢？

  答案：OSS发送请求时，会跟应用服务器构造签名。两者通过签名保证。

- 问题2：这个签名是怎么做的？或者有示例代码吗？

  答案：有的。上面的例子里面是Callback应用服务器的例子：`http://oss-demo.aliyuncs.com:23450`（目前只支持Linux）。

  上面运行的代码是：*callback_app_server.py.zip*

  运行方案：在Linux下面直接执行里面的文件 `python callback_app_server.py`即可，程序自实现了一个简单的http server，运行该程序可能需要安装rsa的依赖。

- 问题3: 为何我的应用服务器收到的回调请求没有Authotization头？

  答案：有些 Web server会将Authorization头自行解析掉，比如apache2，因此需要设置成不解析这个头部。以apache2为例，具体设置方法为：

  1. 打开rewrite模块，执行命令：a2enmod rewrite；
  2. 修改配置文件 `/etc/apache2/apache2.conf`（apache2的安装路径不同会有不一样）。将Allow Override设置成All，然后添加下面两条配置：
     - RewriteEngine on
     - RewriteRule .* - [env=HTTP_AUTHORIZATION:%{HTTP:Authorization},last]

示例程序只是完成了如何检查应用服务器收到的签名， 用户要自行增加对应用服务器收到回调的内容的格式解析 。

其他语言的回调应用服务器版本

- Java版本：
  - 下载地址：点击这里
  - 运行方法：解压包运行`java -jar oss-callback-server-demo.jar 9000`（9000就运行的端口，可以自己指定）

    说明：
    注意这个jar例子在java 1.7运行通过，如果有问题可以自己依据提供的代码进行修改。这是一个maven项目。
- PHP版本：
  - 下载地址：点击这里
  - 运行方法：部署到Apache环境下，因为PHP本身语言的特点，取一些数据头部会依赖于环境。所以可以参考例子根据自己所在环境进行修改。
- Python版本：
  - 下载地址：点击这里
  - 运行方法：解压包直接运行python callback_app_server.py即可，程序自实现了一个简单的http server，运行该程序可能需要安装rsa的依赖。
- Ruby版本：
  - 下载地址：点击这里

- 运行方法： ruby aliyun_oss_callback_server.rb

总结

- 第一个例子：讲解如何在JavaScript直接签名，直接表单上传到OSS。 *oss-h5-upload-js-direct.tar.gz*
- 第二个例子：讲解如何在从后端PHP获取签名，然后直接表单上传到OSS。 *oss-h5-upload-js-php.tar.gz*
- 第三个例子：讲解如何在从后端PHP获取签名及上传后回调。然后直接表单上传到OSS，OSS回调完应用服务器再返回给用户。 *oss-h5-upload-js-php-callback.tar.gz*

3 数据迁移到OSS

3.1 如何将HDFS容灾备份到OSS

背景

当前业界有很多公司是以Hadoop技术构建数据中心，而越来越多的公司和企业希望将业务顺畅地迁移到云上。

在阿里云上使用最广泛的存储服务是对象存储OSS。OSS的数据迁移工具ossimport2可以将您本地或第三方云存储服务上的文件同步到OSS上，但这个工具无法读取Hadoop文件系统的数据，从而发挥Hadoop分布式的特点。并且，该工具只支持本地文件，需要将HDFS上的文件先下载到本地，再通过工具上传，整个过程耗时又耗力。

阿里云E-MapReduce团队开发的Hadoop数据迁移工具**emr-tools**，能让您从Hadoop集群直接迁移数据到OSS上。

本文介绍如何快速地将Hadoop文件系统（HDFS）上的数据迁移到OSS。

前提条件

确保当前机器可以正常访问您的Hadoop集群，即能够用Hadoop命令访问HDFS。

```
hadoop fs -ls /
```

Hadoop数据迁移到OSS

1. 下载*emr-tools*。

   说明：

   emr-tools兼容Hadoop 2.4.x、2.5.x、2.6.x、2.7.x版本，如果有其他Hadoop版本兼容性的需求，请提交工单。

2. 解压缩工具到本地目录。

   ```
   tar jxf emr-tools.tar.bz2
   ```

3. 复制HDFS数据到OSS上。

   ```
   cd emr-tools
   ./hdfs2oss4emr.sh /path/on/hdfs oss://accessKeyId:accessKeySecret@
   bucket-name.oss-cn-hangzhou.aliyuncs.com/path/on/oss
   ```

   参数说明如下。

参数	说明
accessKeyId	访问OSS API的密钥。 获取方式请参见如何获取如何获取*AccessKeyId*和*AccessKeySecret*。
accessKeySecret	
bucket-name.oss-cn-hangzhou.aliyuncs.com	OSS的访问域名，包括bucket名称和endpoint地址。

系统将启动一个Hadoop MapReduce任务（DistCp）。

4. 运行完毕之后会显示本次数据迁移的信息。信息内容类似如下所示。

```
17/05/04 22:35:08 INFO mapreduce.Job: Job job_1493800598643_0009
completed successfully
17/05/04 22:35:08 INFO mapreduce.Job: Counters: 38
 File System Counters
        FILE: Number of bytes read=0
        FILE: Number of bytes written=859530
        FILE: Number of read operations=0
        FILE: Number of large read operations=0
        FILE: Number of write operations=0
        HDFS: Number of bytes read=263114
        HDFS: Number of bytes written=0
        HDFS: Number of read operations=70
        HDFS: Number of large read operations=0
        HDFS: Number of write operations=14
        OSS: Number of bytes read=0
        OSS: Number of bytes written=258660
        OSS: Number of read operations=0
        OSS: Number of large read operations=0
        OSS: Number of write operations=0
 Job Counters
        Launched map tasks=7
        Other local map tasks=7
        Total time spent by all maps in occupied slots (ms)=60020
        Total time spent by all reduces in occupied slots (ms)=0
        Total time spent by all map tasks (ms)=30010
        Total vcore-milliseconds taken by all map tasks=30010
        Total megabyte-milliseconds taken by all map tasks=45015000
 Map-Reduce Framework
        Map input records=10
        Map output records=0
        Input split bytes=952
        Spilled Records=0
        Failed Shuffles=0
        Merged Map outputs=0
        GC time elapsed (ms)=542
        CPU time spent (ms)=14290
        Physical memory (bytes) snapshot=1562365952
        Virtual memory (bytes) snapshot=17317421056
        Total committed heap usage (bytes)=1167589376
 File Input Format Counters
        Bytes Read=3502
 File Output Format Counters
        Bytes Written=0
 org.apache.hadoop.tools.mapred.CopyMapper$Counter
```

```
          BYTESCOPIED=258660
          BYTESEXPECTED=258660
          COPY=10
copy from /path/on/hdfs to oss://accessKeyId:accessKeySecret@bucket-
name.oss-cn-hangzhou.aliyuncs.com/path/on/oss does succeed !!!
```

5. 您可以用osscmd等工具查看OSS上数据情况。

```
osscmd ls oss://bucket-name/path/on/oss
```

OSS数据迁移到Hadoop

如果您已经在阿里云上搭建了Hadoop集群，可以使用如下命令把数据从OSS上迁移到新的Hadoop集群。

```
./hdfs2oss4emr.sh oss://accessKeyId:accessKeySecret@bucket-name.oss-cn
-hangzhou.aliyuncs.com/path/on/oss /path/on/new-hdfs
```

更多使用场景

除了线下的集群，在ECS上搭建的Hadoop集群也可以使用emr-tools，将自建集群迅速地迁移到*E-MapReduce*服务上。

如果您的集群已经在ECS上，但是在经典网络中，无法和VPC中的服务做很好的互操作，所以想把集群迁移到VPC中。可以按照如下步骤迁移：

1. 使用emr-tools迁移数据到OSS上。
2. 在VPC环境中新建一个集群（自建或使用E-MapReduce服务）。
3. 将数据从OSS上迁移到新的HDFS集群中。

如果你使用E-MapReduce服务，还可以直接在Hadoop集群中通过*Spark*、*MapReduce*和*Hive*等组件访问OSS，这样不仅可以减少一次数据复制（从OSS到HDFS），还可以极大的降低存储成本。有关降低成本的详细信息，请参见*EMR+OSS#*计算与存储分离。

3.2 使用OssImport迁移数据

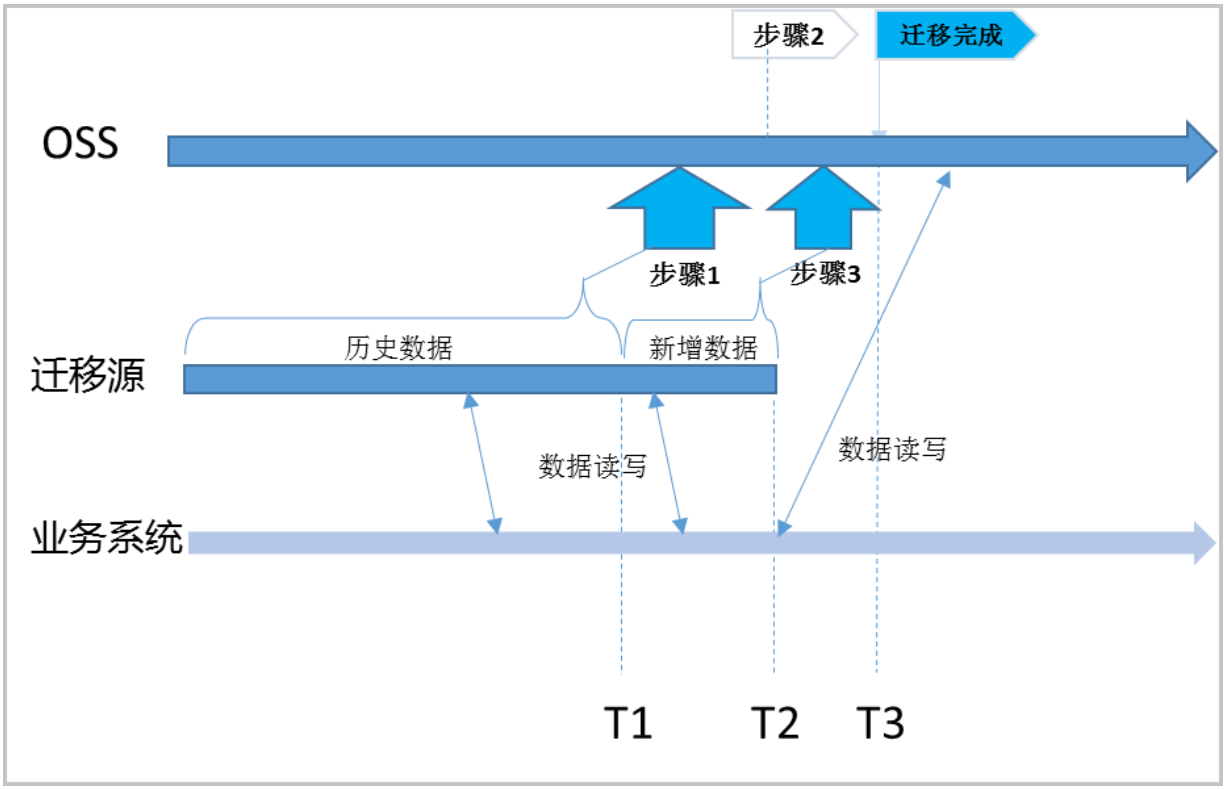
本文向您介绍如何使用OssImport将数据从第三方存储（或OSS）迁移到OSS。

工具选择：单机模式和分布式模式

OssImport有单机模式和分布式模式两种部署方式。一般建议使用分布式模式。您参考OssImport官网指导文档，即可完成迁移过程。本文介绍您在整体迁移方案中可能会关注的内容，以及可以参考的官网文档资源。

迁移方案（从第三方存储迁移到OSS）

以下步骤可以完成从其它存储到OSS的无缝切换（参考官网支持的第三方存储类型*OssImport* 架构及配置）：



具体步骤如下：

1. 全量迁移T1前的历史数据，请参考：*OssImport* 架构及配置。
   - 记录迁移开始时间T1（注意为Unix时间戳，即自1970年1月1日UTC零点以来的秒数，通过命令`date +%s`获取）。
   - 迁移指导说明参考OssImport官网文档，请参考迁移工具-分布式。
2. 打开OSS镜像回源，并将读写切换到OSS，迁移源不再新增数据。
   - 步骤1迁移完成后，在OSS控制台打开*OSS*镜像回源功能，回源地址为迁移源（第三方存储）。
   - 在业务系统读写切换到OSS，假设业务系统修改好的时间为T2。
   - 此时T1前的数据从OSS读取，T1后的数据，OSS利用镜像回源从第三方服务读取，而新数据完全写入OSS。
3. 快速迁移T1~T2到数据。
   - 在步骤2完成后，第三方存储不会再新增数据，数据读写已切到OSS。

- 修改配置文件`job.cfg`的配置项`importSince=T1`，新发起迁移job，迁移T1~T2数据。

4. 步骤3完成后，即完成迁移全过程。

- 步骤3完成后，您业务的所有的读写都在OSS上，第三方存储只是一份历史数据，您可以根据需要决定保留或删除。
- OssImport负责数据的迁移和校验，不会删除任何数据。

迁移成本

迁移过程涉及到成本一般有ECS费用、流量费用、存储费用、时间成本。其中，多数情况下，比如数据超过TB级别，存储成本和迁移时间成正比，而ECS费用相对流量、存储费用较小。

环境准备

假设您有如下迁移需求：

需求项	需求情况
迁移源	AWS S3东京
迁移目的	OSS香港
数据量	500TB
迁移时间要求	1周内完成迁移

您需要准备的环境：

环境项	说明
开通OSS	开通OSS步骤如下： 1. 使用您的账号创建香港地域的OSS Bucket。 2. 在RAM控制台创建子帐号，并授权该子账号可访问OSS。保存AccessKeyID和AccessKeySecret。
购买ECS	购买OSS同区域(香港)的ECS，一般普通的2核4G机型即可，如果迁移后ECS需释放，建议按需购买ECS。正式迁移时的ECS数量，请参考关于迁移用的*ECS*数量。
配置OssImport	说明： *destDomain*参数，即OSS目的endpoint，建议设置为OSS内网的endpoint，避免从ECS数

环境项	说明
	据上传到OSS时产生外网流量费用。具体的endpoint，请参考访问域名和数据中心。
迁移过程	1. 在ECS上搭建OssImport分布式环境。 2. 使用OssImport从东京AWS S3下载数据到ECS（香港），建议使用外网。 3. 使用OssImport从ECS（香港）将数据上传到OSS（香港），建议使用内网。

关于迁移用的ECS数量

您需根据迁移需求，计算您需要用来做迁移的ECS数量：

- 假设您需要迁移的数据量是X TB，要求迁移完成时间Y天，单台迁移速度Z Mbps（每天迁移约Z/100 TB数据）。
- 则正式迁移时，ECS大致需要X/Y/(Z/100)台。

假设单台ECS迁移速度达到200Mbps(即每天约迁移2TB数据)。则上面Case中，ECS约需36台（500/7/2）。

OssImport迁移步骤

配置参考

您可以阅读官网指导文档，了解配置定义*OssImport* 架构及配置、操作步骤分布式部署，在开始前请关注如下信息：

- OssImport下载：在Master下载OssImport分布式版，且master、workers都使用同样的ssh账号、密码。（worker上不用单独下载OssImport，运行`deploy`命令时，OssImport会分发到worker上，请参考分布式部署。）
- Java环境：确认Master和worker都已安装。

> 说明：
> 作为worker的ECS也需要安装Java。

- 设置workdir：通过`conf/sys.properties`的配置项`workingDir`指定，请参考分布式部署。

> 说明：

workdir的设置请参考官网文档，不要设置成OssImport包所在的路径，同时尽量不要设置为已有内容的目录。

- 并发控制：`conf/job.cfg`的配置项`taskObjectCountLimit`，`conf/sys.properties`的配置项`workerTaskThreadNum`，请参考*OssImport* 架构及配置。

  如果小文件比较多，单台ECS迁移速度上不去且CPU load不高，可以参考调高`workerTaskThreadNum`、调低`taskObjectCountLimit`查看效果。
- 其他操作过程遇到问题，一般都是配置问题，请参考分布式部署，并可以查看master和worker上`workdir/Logs`的日志文件。

前期测试

建议您先搭建小型环境（比如2~3台ECS），迁移少量数据以验证配置正确与否。单台ECS迁移带宽能否达到预期，比如200Mbps，如您对迁移时间明确无要求，可不关注。

带宽查看：您可使用`iftop`或`Nload`（需先安装，如`yum install ***`），ECS控制台带宽统计有延时。

说明：

如果测试时文件数目较少，可能无法验证并发性，可以减少参数`taskObjectCountLimit`，比如减少到：文件数/workerTaskThreadNum/worker总数。

迁移步骤

1. 迁移历史数据。
   - 清任务、清配置。
   - 操作过程请参考分布式部署。
   - 开始迁移。
   - 您可在*OSS*控制台OSS Bucket中查看实际迁移完成的数据。
2. 设置镜像回源，客户业务系统读写切到OSS。
   - 在*OSS*控制台打开*OSS*镜像回源，回源地址为迁移源（第三方存储）。
   - 客户业务系统读写切换到OSS，假设业务系统修改好的时间为T2，则T2后不再有新数据写到迁移源。
3. 迁移剩余的数据。

修改job.cfg配置项importSince=T1，请参考*OssImport* 架构及配置，迁移剩余数据（T1~T2）特殊情况下，也可以使用job.cfg中importSince = 0, isSkipExistFile=true进行再次迁移，请参考*OssImport* 架构及配置。

关于迁移速度

- 单台迁移速度：如单台迁移速度不理想（比如小于200Mbps、且CPU load不高时），可参考官网文档和上文，优化并发控制参数，即conf/job.cfg的配置项taskObjectCountLimit，conf/sys.properties的配置项workerTaskThreadNum。
- 迁移ECS数量：参考*ECS*数量估算（相对于流量、存储、时间成本，ECS费用，在迁移总成本中占比较少）。加大ECS数量，会减少迁移时间。

OssImport分布式相关官网文档

序号	官网文档
1	*OssImport* 分布式部署
2	*OssImport* 架构及配置
3	*OssImport* 最佳实践
4	*OssImport* 常见问题

3.3 Amazon S3数据迁移到OSS

OSS提供了S3 API的兼容性，可以让您的数据从Amazon S3无缝迁移到阿里云OSS上。从Amazon S3迁移到OSS后，您仍然可以使用S3 API访问OSS，仅需要对S3的客户端应用进行如下改动：

1. 获取OSS主账号或子账号的AccessKeyId和AccessKeySecret，并在您使用的客户端和SDK中配置您申请的AccessKeyId与AccessKeySecret。
2. 设置客户端连接的endpoint为OSS endpoint。OSS endpoint列表请参见访问域名和数据中心。

迁移步骤

从第三方存储迁移到OSS的具体操作步骤，请您参见使用*OssImport*迁移数据。

迁移后用S3 API访问OSS

从S3迁移到OSS后，您使用S3 API访问OSS时，需要注意以下几点：

- Path style和Virtual hosted style

*Virtual hosted style*是指将Bucket放入host header的访问方式。OSS基于安全考虑，仅支持virtual hosted访问方式，所以在S3到OSS迁移后，客户端应用需要进行相应设置。部分S3工具默认使用Path style，也需要进行相应配置，否则可能导致OSS报错禁止访问。

- OSS对权限的定义与S3不完全一致，迁移后如有需要，可对权限进行相应调整。二者的主要区别可参考下表。

说明：

- 更详细的区别请参见*ACL*验证流程。
- OSS仅支持S3中的私有、公共读和公共读写三种ACL模式。

对象	Amazon S3权限	Amazon S3	阿里云OSS
Bucket	READ	拥有Bucket的list权限	对于Bucket下的所有Object，如果某Object没有设置Object权限，则该Object可读
	WRITE	Bucket下的Object可写入或覆盖	• 对于Bucket下不存在的Object，可写入 • 对于Bucket下存在的Object，如果该Object没有设置Object权限，则该Object可被覆盖 • 可以initiate multipart upload
	READ_ACP	读取Bucket ACL	读取Bucket ACL，仅Bucket owner和授权子账号拥有此权限
	WRITE_ACP	设置Bucket ACL	设置Bucket ACL，仅Bucket owner和授权子账号拥有此权限
Object	READ	Object可读	Object可读
	WRITE	N/A	Object可以被覆盖

对象	Amazon S3权限	Amazon S3	阿里云OSS
	READ_ACP	读取Object ACL	读取Object ACL，仅Bucket owner和授权子账号拥有此权限
	WRITE_ACP	设置Object ACL	设置Object ACL，仅Bucket owner和授权子账号拥有此权限

- 存储类型

  OSS支持标准（Standard）、低频访问（IA）和归档存储（Archive）三种存储类型，分别对应Amazon S3中的STANDARD、STANDARD_IA和GLACIER。

  与Amazon S3不同的是，OSS不支持在上传object时直接指定存储类型，object的存储类型由bucket的存储类型指定。OSS支持标准、低频访问和归档三种Bucket类型，Object的存储类型还可以通过lifecycle进行转换。

  归档存储类型的object在读取之前，要先使用Restore请求进行解冻操作。与S3不同，OSS会忽略S3 API中的解冻天数设置，解冻状态默认持续1天，用户可以延长到最多7天，之后，Object又回到初始时的冷冻状态。

- ETag
  - 对于PUT方式上传的Object，OSS Object的ETag与Amazon S3在大小写上有区别。OSS为大写，而S3为小写。如果客户端有关于ETag的校验，请忽略大小写。
  - 对于分片上传的Object，OSS的ETag计算方式与S3不同。

兼容的S3 API

- Bucket操作：
  - Delete Bucket
  - Get Bucket（list objects）
  - Get Bucket ACL
  - Get Bucket lifecycle
  - Get Bucket location
  - Get Bucket logging
  - Head Bucket
  - Put Bucket

- Put Bucket ACL
- Put Bucket lifecycle
- Put Bucket logging

- Object操作：
  - Delete Object
  - Delete Objects
  - Get Object
  - Get Object ACL
  - Head Object
  - Post Object
  - Put Object
  - Put Object Copy
  - Put Object ACL
- Multipart操作：
  - Abort Multipart Upload
  - Complete Multipart Upload
  - Initiate Multipart Upload
  - List Parts
  - Upload Part
  - Upload Part Copy

4 数据备份

4.1 备份存储空间

针对存放在OSS上的数据，阿里云提供多种数据备份方式，以满足不同场景的备份需求。

OSS云上备份方式有如下2种：

- 跨区域复制（提供控制台配置操作以及基于API/SDK两种模式）
- 基于OssImport工具

通过跨区域复制进行备份

使用场景

参见管理跨区域复制开发指南。

控制台设置操作

参见设置跨区域复制操作指南。

常见问题

参见*Bucket*之间数据同步常见问题。

特别说明

- 源Bucket和目标Bucket属于同一个用户，且分属不同的区域。
- 源Bucket、目标Bucket存储类型都不是归档类型。
- 若要在同一区域的Bucket之间进行数据同步，则可以通过OSS的SDK/API编码实现。

通过使用OssImport工具进行备份

OssImport工具可以将本地、其它云存储的数据迁移到OSS，它有以下特点：

- 支持丰富的数据源，有本地、七牛、百度BOS、AWS S3、Azure Blob、又拍云、腾讯云COS、金山KS3、HTTP、OSS等，并可根据需要扩展。
- 支持断点续传。
- 支持流量控制。
- 支持迁移指定时间后的文件、特定前缀的文件。
- 支持并行数据下载、上传。
- 支持单机模式和分布式模式。单机模式部署简单使用方便，分布式模式适合大规模数据迁移

使用场景

参见数据迁移。

安装部署

参见说明及配置、单机部署、分布式部署。

常见问题

参见常见问题。

特别说明

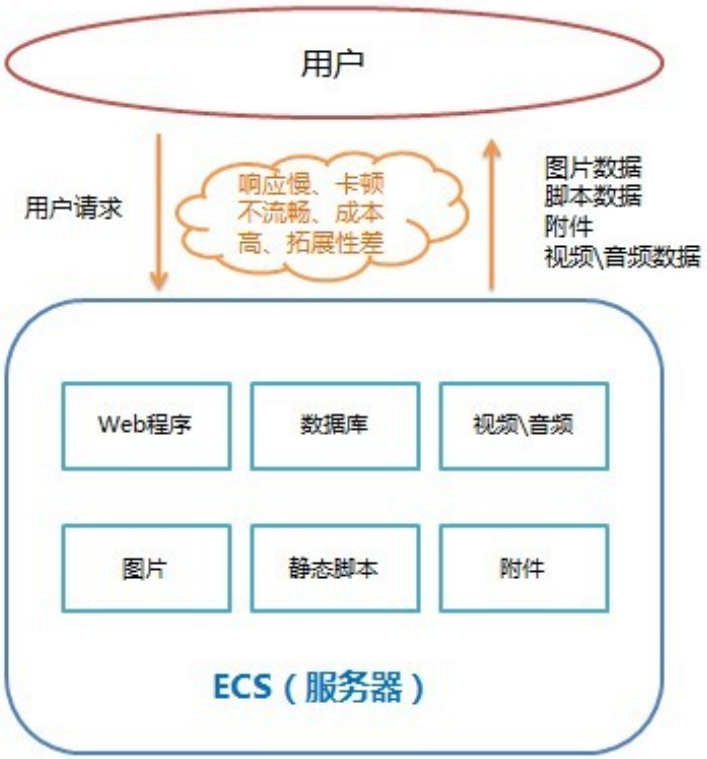
- 不同账户的不同Bucket之间，数据量很大，10TB级别以上的情况，建议使用分布式的版本。
- 利用增量模式在OSS之间同步文件修改操作，OssImport只能同步文件的修改操作(put/apppend/multipart)，不能同步读取和删除操作，数据同步的及时性没有具体的 SLA 保证， 慎重选择。
- 如果开通了跨区域复制的地域，则推荐使用跨区域复制进行地域之间的数据同步。

5 存储空间管理

5.1 CDN加速OSS

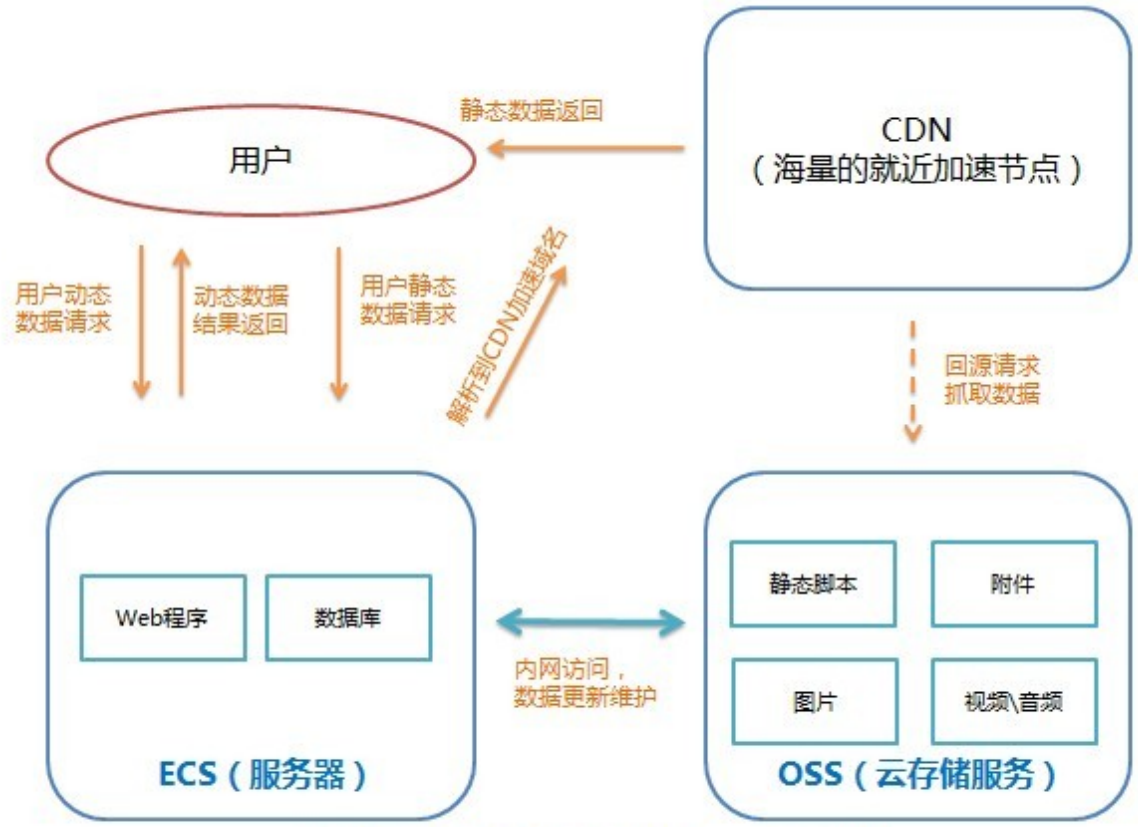
背景介绍

传统动静不分离的产品架构，其性能会随着系统访问量的增长而受到限制甚至遭遇瓶颈。产品架构如下图所示：



传统网站架构示意

利用CDN和OSS实现动静分离，灵活的架构可以支持海量用户访问。产品架构如下图所示：



网站动静分离架构示意图

适用场景

- 静态文件访问量大，服务器负载高，I/O问题导致用户访问卡顿。
- 静态文件数量大，服务器存储空间不够。
- 静态文件用户访问量大，且分布在各地。
- 移动更新包在某个时间段需要高速下载，且并发下载量高。

架构描述

OSS作为海量文件存储源，静态图片、视频文件、下载包、app更新包等均放在OSS上。OSS作为CDN的源站，通过CDN加速分发，用户通过CDN节点就近获得文件。架构优势：

- 降低Web服务器负载，静态文件访问负载全部通过CDN。
- 存储费用最低，OSS的存储费用仅为ECS磁盘费用的50%。
- 海量存储空间，无需考虑存储架构升级。
- 流量费用低，相比直接通过OSS访问，除极少额外增加的回源流量外，主要流量使用CDN流量，单价远远低于OSS直接访问的外网流量单价。

实战案例

以一个常见的Web站点为例。www.acar.com是一个刚建立的汽车资讯车友交流网站。主站用Php搭建，有10GB的图片素材和部分JS文件。目前购买一台ECS放置所有程序代码，并在ECS上安装MySQL数据库。随着用户访问量的不断增长，不少用户反映，访问网站的速度越来越慢，图片加载慢，网站响应慢。网站技术人员也发现用户上传的图片越来越多，快超过 1TB了。

对于以上案例我们可以利用OSS和CDN对网站进行架构优化，实现动静分离的产品架构，提升用户访问体验，同时成本也在可控的范围内。

解决方案及步骤如下：

1. 对ECS上的网站程序进行整理，把动态程序部分和静态部分分不同的目录进行管理：
   - 建立Images目录，放置所有网站高清素材图片。
   - 建立Javascript目录，放置所有的JS脚本。
   - 建立Attachment目录，放置所有用户上传的图片和附件。
2. 新建一个Bucket。

   根据您的ECS所在的区域选择bucket所在区域，权限选择 公共读，bucket名称与ECS上新建的目录的名称对应，比如“acar-image-bucket”。具体操作请参见 创建存储空间。
3. 绑定域名**image.acar.com** 并进行CDN加速。具体操作请参见 管理域名。
4. 上传文件，体验加速效果。
   a. 把您在第1步中建立在ECS上的Images目录下的所有图片文件上传到**acar-image-bucket**下。具体操作请参见 上传文件。您也可以使用OSS客户端工具更加方便灵活的完成图片的上传。
   b. 获取该文件的CDN加速的访问地址，通常为您输入的加速域名+'/'+'文件名'的格式。具体操作请参见 获取文件访问地址。
   c. 逐一完成图片文件的上传。
5. 按照前4步的示意，把其他两个文件也通过**CDN**加速的方式上传，分别建立**acar-js-bucket**和**acar-csimages-bucket**两个使用CDN加速的OSS bucket。
6. 在原本ECS系统中，找到原本访问静态文件的代码，把访问URL修改为加速访问的地址。以后用户访问您的网站的静态文件就全部通过OSS+CDN的方式访问，不再占用您ECS的资源。

> **说明：**
> 如果您想把用户上传的文件自动同步到**acar-csimages-bucket**中，您可以参考OSS相关SDK和API的 PutObjcet部分，实现代码级别自动上传。

CDN缓存自动刷新

如果您使用了阿里云的CDN并绑定了加速域名回源到OSS，您就可以使用OSS的CDN缓存自动刷新功能，此功能在覆盖写的情况下（包括覆盖一个已有的文件、删除一个已有的文件），OSS会主动刷新CDN，回源到OSS获取覆盖后的文件，用户不需要显式调用CDN的刷新接口。刷新的URL规则如下：加速域名 `+ / + Object`

例如您绑定的加速域名是`image.acar.com`，如果这个域名绑定的bucket覆盖上传了一个文件`test.jpg`，则OSS会刷新掉`image.acar.com/test.jpg`这个URL，刷新生效的时间以CDN保证的刷新生效时间为准，一般在十分钟以内。

开通方法：在Bucket的 域名管理 页面，打开 **CDN缓存自动刷新** 功能即可。

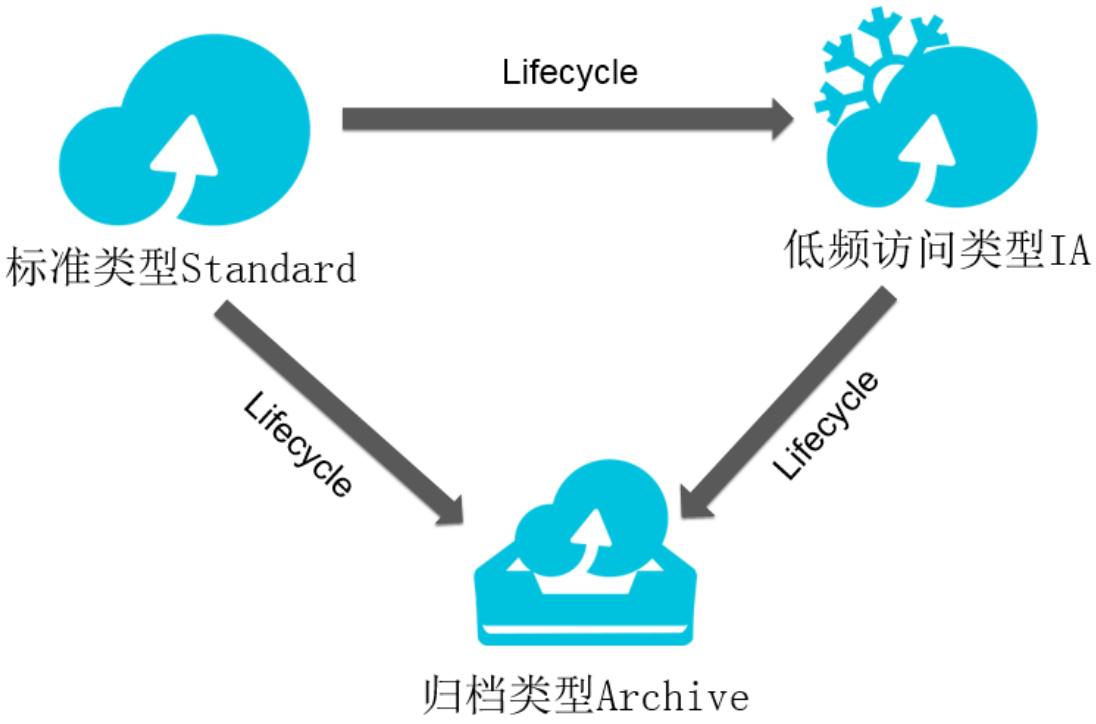
5.2 存储类型转换

通过生命周期设置存储类型自动转换

OSS支持三种存储类型：标准类型、低频访问类型、归档类型。

OSS 生命周期管理（Lifecycle）提供Object Transition机制，支持以下存储类型的自动转换：

- 标准类型转换为低频访问类型
- 标准类型转换为归档类型
- 低频访问类型转换为归档类型



举例

对一个Bucket里特定前缀的Object配置Lifecycle策略如下：

- 存储30天后，自动转换到低频访问类型。
- 180天后，自动转换到归档类型。
- 超过360天后，文件会被自动删除。

您可以通过控制台完成以上Lifecycle策略的配置。具体操作请参见设置生命周期。

说明：

如果一个Bucket同时配置了**Object保留指定天数后转换到低频访问**、**Object保留指定天数后转换到归档冷备**和**Object保留指定天数后删除**，各规则设置的天数必须满足以下规则：

转换到低频的天数 < 转换到归档的天数 < 指定天数后删除

注意事项

Object类型转换后，会按照转换后的存储类型的存储单价计算存储费用。低频访问类型和归档存储类型需要特别注意：

- 最小计量空间

对于小于128KB的Object，会按照128KB计算空间大小。

- 最短存储周期

  存储的数据需要至少保存30天，删除短于30天的文件，会收取提前删除费用。

- 归档类型的Restore时间

  归档类型Object恢复到可读取状态（Restore）需要1分钟的解冻时间。所以如果业务场景上需要实时读取，建议只转换成低频访问存储类型，而不应转换成归档存储类型，避免转换成归档类型后，数据无法实时读取。

- 数据获取费用

  低频访问类型和归档类型会收取数据获取费用，与流出流量是独立计费项，如果每个Object平均访问频率高于每月1次，不建议转换成低频访问或者归档类型。

其他方式的存储类型转换

对于需要将归档类型重新转换为标准类型或低频类型，低频类型转换为标准类型，可以通过读取Object重新写入到相对应存储类型Bucket来实现存储类型转换，Object的缺省存储类型由Bucket决定。

例如，用户需要对在标准类型Bucket中已经存储成低频访问类型的Object重新转换成标准类型Object，可以通过读取此Object后重新写入，根据Bucket的类型，新写入的Object是标准存储类型。

对于已经转储成归档类型的Object，需要先执行Restore操作，解冻成可读取状态后，才能被读取。详情请参见使用归档存储类型。

使用归档存储类型

5.3 防盗链

背景

A是某一网站站长，A网站中的图片和音频视频链接等静态资源都保存在阿里云对象存储*OSS*上。以图片为例，A在OSS上存放的URL为`http://referer-test.oss-cn-hangzhou.aliyuncs.com/aliyun-logo.png`

OSS资源外链地址见*OSS* 地址，这样的URL（不带签名）要求用户的Bucket权限为公开读权限。

B是另一网站的站长，B在未经A允许的情况下使用A网站的图片资源，放置在自己网站的网页中，通过这种方法盗取空间和流量。这样的情况下，第三方网站用户看到的是B网站，但并不清楚

网站里的图片来源。由于OSS是按使用量来收费，这样用户A在没有获取任何收益的情况下，反而承担了资源使用费用。

本文适用于在网页中使用了OSS资源作为外链的用户，并介绍类似A用户将资源存放在OSS上后，如何通过设置防盗链的方法避免承担不必要的资源使用费用。

实现方法

目前OSS提供的防盗链方法主要有以下两种：

- 设置Referer。该操作通过控制台和SDK均可进行，用户可根据自身需求进行选择。
- 签名URL，适合习惯开发的用户。

本文将提供如下两个示例：

- 通过控制台设置Referer防盗链
- 基于PHP SDK动态生成签名URL防盗链

设置Referer

该部分主要说明什么是Referer，以及OSS如何利用Referer做防盗链。

- Referer是什么

  Referer是HTTP Header的一部分，当浏览器向网站Web服务器发送请求的时候，通常会带上Referer，告诉服务器此次请求的链接来源。从以上示例来看，假如用户B的网站为userdomain-steal，想盗链用户A的图片链接`http://referer-test.oss-cn-hangzhou.aliyuncs.com/aliyun-logo.png`。A的网站域名为`userdomain`。

  假设盗链网站userdomain-steal的网页如下：

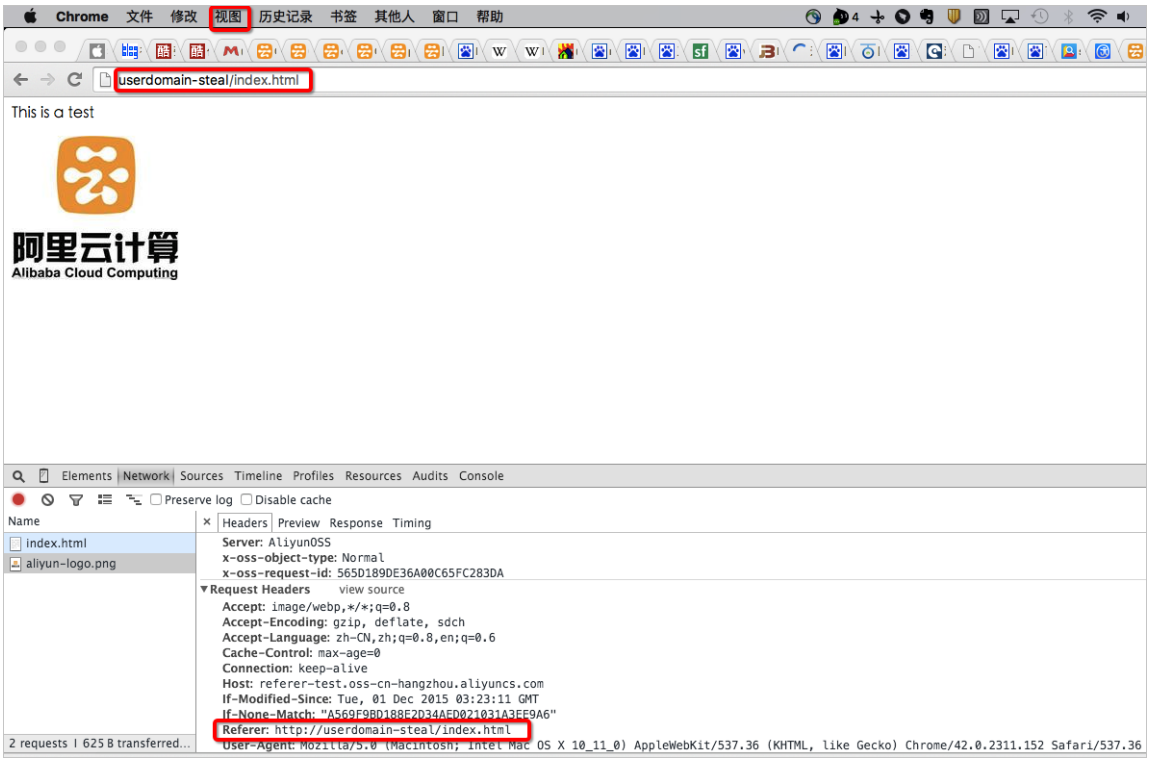
```
<html>
    <p>This is a test</p>
    <img src="http://referer-test.oss-cn-hangzhou.aliyuncs.com/
aliyun-logo.png" />
</html>
```

  假设源站为userdomain的网页如下：

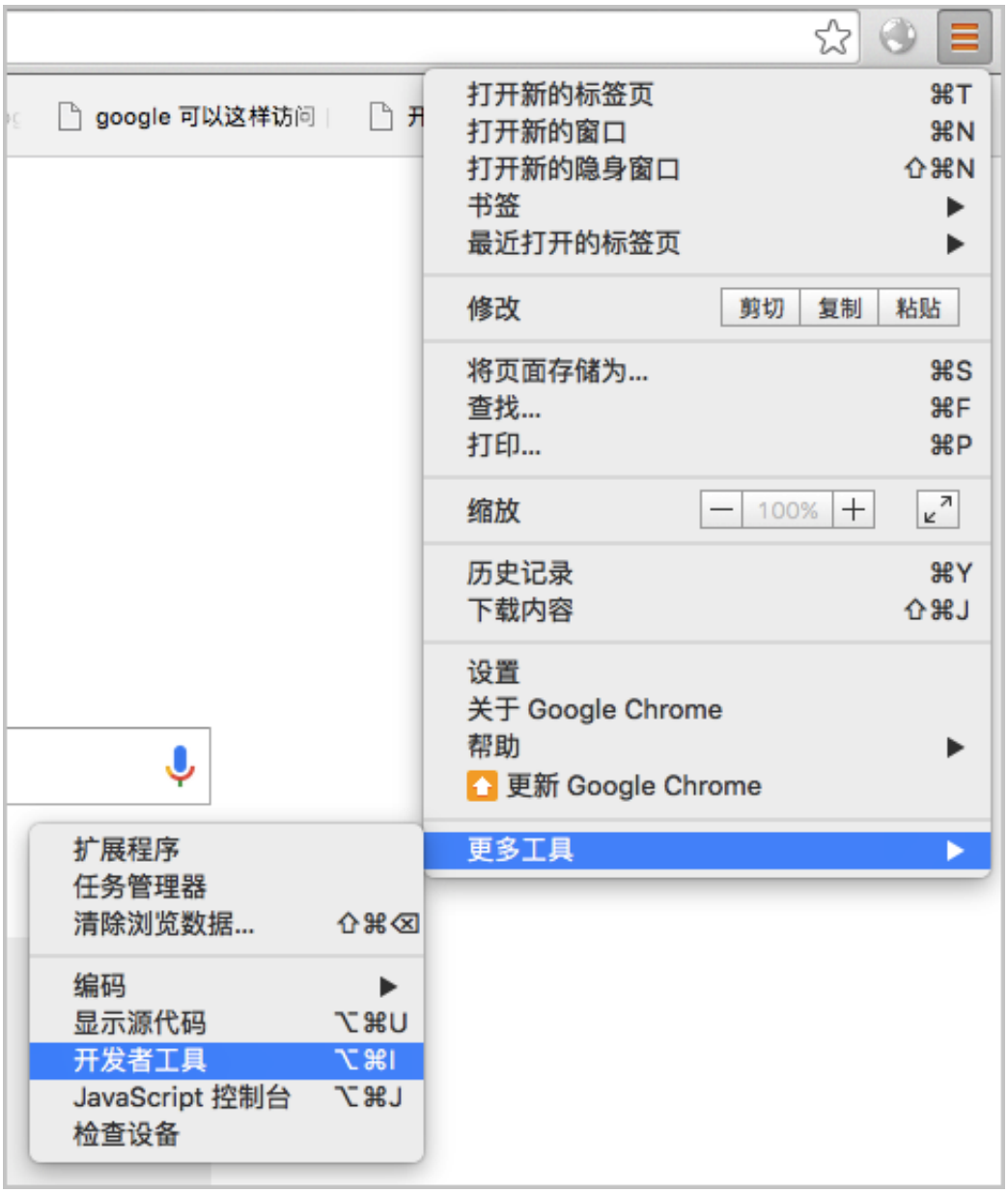
```
<html>
    <p>This is my test link from OSS URL</p>
    <img src="http://referer-test.oss-cn-hangzhou.aliyuncs.com/
aliyun-logo.png" />
```

```
</html>
```

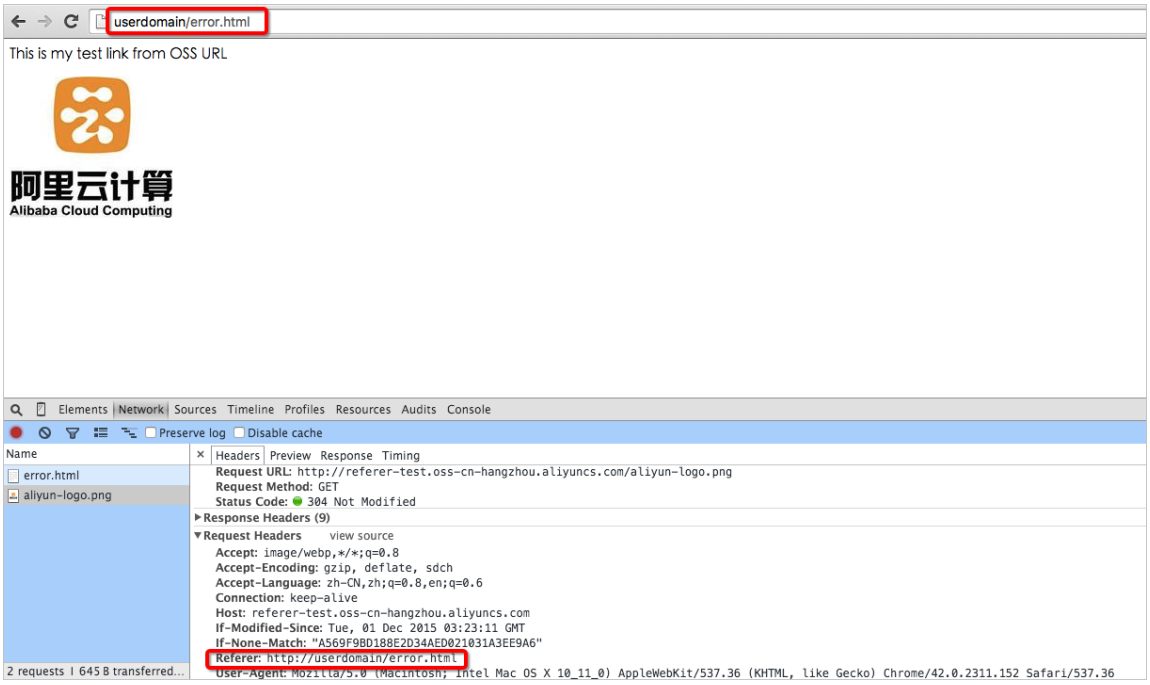
- 当互联网用户用浏览器访问B的网站页面`http://userdomain-steal/index.html`，网页里链接是A的网站的图片。由于从一个域名(userdomain-steal)请求跳到了另一个域名(oss-cn-hangzhou.aliyuncs.com)，浏览器就会在HTTP请求的Header中带上Referer，如图所示：



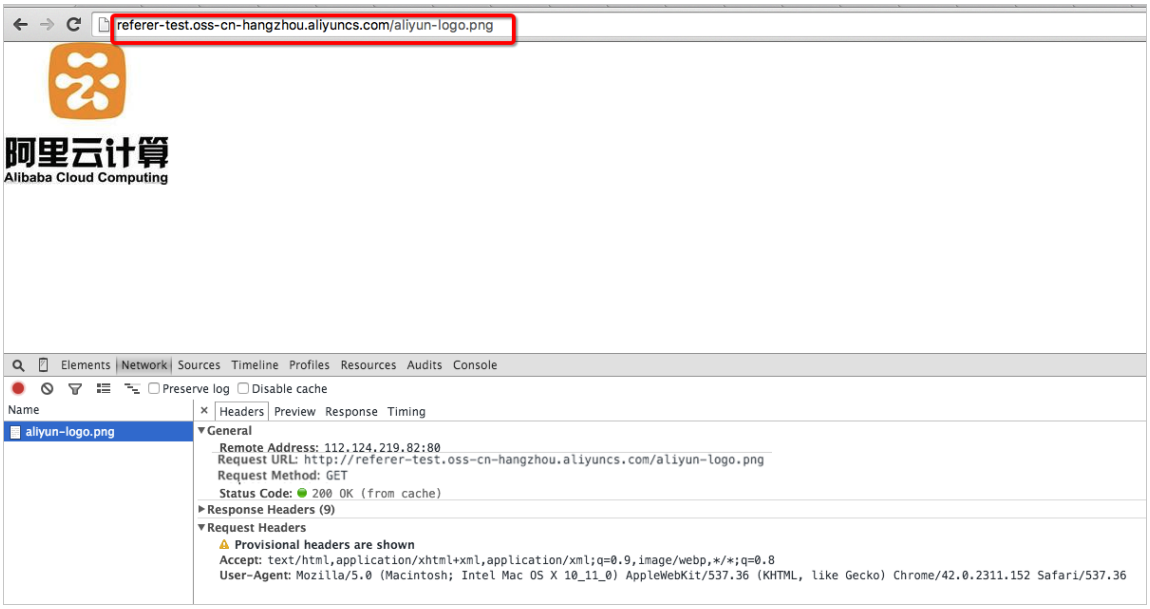
可以看到浏览器在HTTP请求中的Referer为`http://userdomain-steal/index.html`。本文主要是使用chrome的开发者模式来查看网页请求的，如图：



- 同样浏览器访问`http://userdomain/error.html`，也可以看到浏览器的Referer为`http://userdomain/error.html`。



— 如果浏览器直接输入地址，可以看到，请求中的Referer为空。



如果A没有在OSS进行任何Referer相关设置，以上三种情况都是可以访问用户A的图片链接。

- OSS通过Referer防盗链的原理

  由此可见，浏览器在请求OSS资源时，如果发生页面跳转，浏览器会在请求中带入Referer，此时Referer的值为上一页面的URL，有的时候Referer也会为空。

  针对这两种情况，OSS的Referer功能提供两种选择：

  — 设置是否允许空Referer访问。不能单独设置，需要配合Referer白名单一起使用。

- 设置Referer白名单。

细节分析如下：

- 用户只有通过签名URL或者匿名访问object时，才会做防盗链验证。请求的Header中有“Authorization”字段的，不会做防盗链验证。
- 一个Bucket可以支持多个Referer参数。
- Referer参数支持通配符“*”和“？”。
- 用户可以设置允许空Referer的请求访问。
- 白名单为空时，不会检查Referer字段是否为空（不然所有的请求都会被拒绝，因为空Referer会被拒绝，对于非空Referer OSS在Referer白名单里也找不到）。
- 白名单不为空，且设置了”不允许Referer字段为空”的规则。则只有Referer属于白名单的请求被允许，其他请求（包括Referer为空的请求）会被拒绝。
- 白名单不为空，但设置了”允许Referer字段为空”的规则。则Referer为空的请求和符合白名单的请求会被允许，其他请求都会被拒绝。
- Bucket的三种权限（private，public-read，public-read-write）都会检查Referer字段。

通配符详解：

- 星号“*”：可以使用星号代替0个或多个字符。如果正在查找以AEW开头的一个文件，但不记得文件名其余部分，可以输入AEW#查找以*AEW*开头的所有文件类型的文件#如*AEWT.txt*、*AEWU.EXE*、*AEWI.dll*等。要缩小范围可以输入*AEW*.txt，查找以AEW开头的所有文件类型并.txt为扩展名的文件如AEWIP.txt、AEWDF.txt。
- 问号“?”：可以使用问号代替一个字符。如果输入love?，查找以love开头的一个字符结尾文件类型的文件，如lovey、lovei等。要缩小范围可以输入love?.doc，查找以love开头的一个字符结尾文件类型并.doc为扩展名的文件如lovey.doc、loveh.doc。

- 不同的Referer设置和防盗链效果

Referer 设置的效果如下：

- 只设置“不允许referer为空”

从控制台中来设置不允许referer为空，如图所示：

直接访问：发现可以访问，是防盗链失效了吗？不是的，因为”白名单为空时，不会检查Referer字段是否为空”，所以白名单为空的时候，这个设置无效。因此需要设置Referer白名单。

— 设置“不允许referer为空”的同时也设置Referer白名单

从前面例子中我们可以看到在浏览器的请求中Referer为当前页面的URL，所以需要知道网站会从哪几个URL跳转过来，然后进行配置。

Referfer白名单的设置规则：

- 例子中的Referer为`http://userdomain/error.html`，所以Referer白名单可以设置为`http://userdomain/error.html`，但由于OSS的Referer检查是通过前缀匹配的，假如有其他网页比如`http://userdomain/index.html`就访问不了，所以Referer白名单可以配置成`http://userdomain/`。
- 假如还有其他域名比如`http://img.userdomain/index.html`也需要访问，那么Referer白名单应该加上`http://*.userdomain/`。

这里两个都配置，如图所示：

做测试可以得到如下结果：

浏览器输入	预期	结果
http://referer-test.oss-cn-hangzhou.aliyuncs.com/aliyun-logo.png	直接访问，Referer为空，预期：不允许空Referer的请求，返回403。	符合预期

浏览器输入	预期	结果
http://userdomain/error.html	请求来自于源站，预期：访问成功。	符合预期
http://userdomain-steal/index.html	请求来自于盗链网站，预期：OSS返回403，防盗链成功。	符合预期
http://img.userdomain/error.html	请求来自于源站三级域名。	符合预期

说明：

- 测试中提到的域名是为了测试而假设的，和您实际的域名不一样，请注意区分。
- 如果Referer白名单只有`http://userdomain/`，浏览器模拟三级域名访问`http://img.userdomain/error.html`的时候，三级域名无法匹配Referer白名单，OSS就会返回403。

- 设置“允许referer为空”的同时也设置Referer白名单

  Referer白名单有`http://*.userdomain/`和`http://userdomain`。

  如图所示：

  测试得出以下结果：

浏览器输入	预期	结果
http://referer-test.oss-cn-hangzhou.aliyuncs.com/aliyun-logo.png	直接访问，Referer为空，预期：访问成功。	符合预期
http://userdomain/error.html	请求来自于源站，预期：访问成功。	符合预期

浏览器输入	预期	结果
http://userdomain-steal/index.html	请求来自于盗链网站，预期：OSS返回403，防盗链成功。	符合预期
http://img.userdomain/error.html	请求来自于源站三级域名。	符合预期

- 如何在OSS里设置Referer

  功能使用参考：

  - API：*Put Bucket Referer*
  - 控制台：防盗链设置

- Referer防盗链的缺点

  Referer防盗链的优点是设置简单，控制台即可操作。最大的缺点就是无法防止恶意伪造Referer，如果盗链是通过应用程序模拟HTTP请求，伪造Referer，则会绕过用户防盗链设置。如果对防盗链有更高要求，请参考以下签名URL防盗链的描述。

签名URL

签名URL的原理和实现方法见OSS开发人员指南授权第三方下载。签名URL的实现步骤如下：

1. 将Bucket的权限设置为私有读。
2. 根据期望的超时时间（签名URL失效的时间）生成签名。

具体实现方法如下：

1. 安装PHP最新代码，参考*PHP SDK文档*。
2. 实现生成签名URL并将其放在网页中，作为外链使用的简单示例如下：

```
<?php
 require 'vendor/autoload.php';
 #最新PHP提供的自动加载
 use OSS\OssClient;
 #表示命名空间的使用
 $accessKeyId="a5etodit71tlznjt3pdx7lch";
 #AccessKeyId，需要使用用户自己的
 $accessKeySecret="secret_key";
 #AccessKeySecret，需要用用户自己的
 $endpoint="oss-cn-hangzhou.aliyuncs.com";
 #Endpoint，根据Bucket创建的区域来选择，本文中是杭州
 $bucket = 'referer-test';
 #Bucket，需要用用户自己的
 $ossClient = new OssClient($accessKeyId, $accessKeySecret, $
endpoint);
```

```
$object = "aliyun-logo.png";
#需要签名的Object
$timeout = 300;
#期望链接失效的时间，这里表示从代码运行到这一行开始的当前时间往后300秒
$signedUrl = $ossClient->signUrl($bucket, $object, $timeout); #签名
URL实现的函数
$img= $signedUrl;
#将签名URL动态放到图片资源中并打印出来
$my_html = "<html>";
$my_html .= "<img src=\"".$img. "\" />";
$my_html .= "<p>".$img."</p>";
$my_html .= "</html>";
echo $my_html;
?>
```

3. 通过浏览器访问多请求几次会发现签名的URL会变，这是正常的。主要是因为过期时间的改变导致的。这个过期时间是链接失效的时间，是以Unix time的形式展示的。例如：Expires=1448991693，可以将这个时间转换成本地时间。在Linux下的命令为`date -d@ 1448991693`，也可以在网络上找工具自行转换。

特别说明

签名URL可以和Referer白名单功能一起使用。

如果签名URL失效的时间限制在分钟内，盗链用户即使伪造了Referer也必须拿到签名的URL，且必须在有效的时间内才能盗链成功。相比只使用Referer来说，增加了盗链的难度。也就是说签名URL配合Referer白名单功能，可以增加防盗链的效果。

防盗链总结

基于OSS的防盗链最佳实践点如下：

- 使用三级域名URL，例如`referer-test.oss-cn-hangzhou.aliyuncs.com/aliyun-logo.png`，安全性比绑定二级域名更高。三级域名方式能够提供Bucket级别的清洗和隔离，能够应对被盗链后的流量暴涨的情况，也能避免不同Bucket间的互相影响，最终提高业务可用性。
- 如果使用自定义域名作为连接，CNAME也请绑定到三级域名，规则是bucket + endpoint。假如您的bucket名为test，三级域名则为`test.oss-cn-hangzhou.aliyuncs.com`。
- 对Bucket设定尽可能严格的权限类别。例如提供公网服务的Bucket设置为public-read或private，禁止设置为public-read-write。Bucket权限参见访问控制。
- 对访问来源进行验证，根据需要设置合适的Referer白名单。
- 如果需要更严格的防盗链方案，请参考签名的URL方案。
- 记录Bucket访问日志，能够及时发现盗链活动和验证防盗链方案的有效性。访问日志参见设置访问日志记录。

常见问题及解决方案

- 在OSS控制台设置了防盗链，但一直不生效，页面可以反而播放器不可以，请问为什么？怎么解决？

  目前设置防盗链不生效的主要问题集中于视频和音频文件，在使用诸如Windows Media Player、Flash Player等播放器后，在请求OSS资源的时候传递的Referer为空，这就造成防盗链的失效。针对这种情况，可以参考上面提到的签名URL防盗链的方法。

- Referer是什么？如果遇到HTTPS怎么办？

  Referer是HTTP协议中的请求头，在跨页面访问的时候会带上。需要看看浏览器请求的Referer是`http://`还是`https://`，通常是`http://`。

- 如何生成签名URL？AccessKeySecret放在客户端里的安全性如何？

  签名URL的方法参见各个SDK文档。AccessKeySecret不建议直接放在客户端，RAM提供了*STS服务*可以解决这个问题，详情参见*RAM*和*STS*指南。

- 例如要写`a.baidu.com`和`b.baidu.com`，这两个用通配符(*,?) 如何写？

  可以写成`http://*.baidu.com`，对于这种单字符，也可以写成`http://?.baidu.com`。

- *.domain.com 可以匹配二级域名，但无法匹配 domain.com，另外添加一行 domain.com 也没效果，如何配置？

  注意一般的referer中会带http这样的参数，可以通过chrome的开发者模式观察下请求的Referer是什么，然后再具体设置。这里可能是忘了写`http://`，应该为`http://domain.com`。

- 如果防盗链没有生效怎么办？

  推荐使用chrome来查看。打开开发者模式，点击网页，查看HTTP请求中的`Referer`具体值。并确认是否与OSS中设置的`Referer`匹配。如果还是解决不了，请提工单。

5.4 跨域资源共享（CORS）

同源策略

跨域访问，或者说 JavaScript 的跨域访问问题，是浏览器出于安全考虑而设置的一个限制，即同源策略。举例说明，当 A，B 两个网站属于不同的域时，如果来自于 A 网站的页面中的 JavaScript 代码希望访问 B 网站的时候，浏览器会拒绝该访问。

然而，在实际应用中，经常会有跨域访问的需求。比如用户的网站 www.a.com，后端使用了 OSS，在网页中提供了使用 JavaScript 实现的上传功能，但是在该页面中，只能向 www.a.com 发送请求，向其他网站发送的请求都会被浏览器拒绝。这样会导致用户上传的数据必须从www.a.com 中转。如果设置了跨域访问的话，用户就可以直接上传到 OSS 而无需从 www.a.com 中转。

CORS 介绍

跨域资源共享（Cross-Origin Resource Sharing，简称 CORS），是 HTML5 提供的标准跨域解决方案，具体的CORS规则可以参考 *W3C CORS规范*。

CORS 是一个由浏览器共同遵循的一套控制策略，通过HTTP的Header来进行交互。浏览器在识别到发起的请求是跨域请求的时候，会将Origin的Header加入HTTP请求发送给服务器，比如上面那个例子，Origin的Header就是www.a.com。服务器端接收到这个请求之后，会根据一定的规则判断是否允许该来源域的请求，如果允许的话，服务器在返回的响应中会附带上Access-Control-Allow-Origin这个Header，内容为www.a.com来表示允许该次跨域访问。如果服务器允许所有的跨域请求的话，将Access-Control-Allow-Origin的Header设置为*即可，浏览器根据是否返回了对应的Header来决定该跨域请求是否成功，如果没有附加对应的Header，浏览器将会拦截该请求。

以上描述的仅仅是简单情况，CORS规范将请求分为两种类型，一种是简单请求，另外一种是带预检的请求。预检机制是一种保护机制，防止资源被本来没有权限的请求修改。浏览器会在发送实际请求之前先发送一个OPTIONS的HTTP请求来判断服务器是否能接受该跨域请求。如果不能接受的话，浏览器会直接阻止接下来实际请求的发生。

只有同时满足以下两个条件才不需要发送预检请求：

- 请求方法是如下之一：
  - GET
  - HEAD
  - POST
- 所有的Header都在如下列表中：
  - Cache-Control
  - Content-Language
  - Content-Type
  - Expires
  - Last-Modified

— Pragma

预检请求会附带一些关于接下来的请求的信息给服务器，主要有以下几种：

- Origin：请求的源域信息
- Access-Control-Request-Method：接下来的请求类型，如POST、GET等
- Access-Control-Request-Headers：接下来的请求中包含的用户显式设置的Header列表

服务器端收到请求之后，会根据附带的信息来判断是否允许该跨域请求，信息返回同样是通过Header完成的：

- Access-Control-Allow-Origin：允许跨域的Origin列表
- Access-Control-Allow-Methods：允许跨域的方法列表
- Access-Control-Allow-Headers：允许跨域的Header列表
- Access-Control-Expose-Headers：允许暴露给JavaScript代码的Header列表
- Access-Control-Max-Age：最大的浏览器缓存时间，单位s。

浏览器会根据以上的返回信息综合判断是否继续发送实际请求。当然，如果没有这些Header浏览器会直接阻止接下来的请求。

说明：

这里需要再次强调的一点是，以上行为都是浏览器自动完成的，用户无需关注这些细节。如果服务器配置正确，那么和正常非跨域请求是一样的。

CORS主要使用场景

CORS使用一定是在使用浏览器的情况下，因为控制访问权限的是浏览器而非服务器。因此使用其它的客户端的时候无需关心任何跨域问题。

使用CORS的主要应用就是在浏览器端使用Ajax直接访问OSS的数据，而无需走用户本身的应用服务器中转。无论上传或者下载。对于同时使用OSS和使用Ajax技术的网站来说，都建议使用CORS来实现与OSS的直接通信。

OSS对CORS的支持

OSS提供了CORS规则的配置从而根据需求允许或者拒绝相应的跨域请求。该规则是配置在Bucket级别的。详情可以参考*PutBucketCORS*。

CORS请求的通过与否和OSS的身份验证等是完全独立的，即OSS的CORS规则仅仅是用来决定是否附加CORS相关的Header的一个规则。是否拦截该请求完全由浏览器决定。

使用跨域请求的时候需要关注浏览器是否开启了Cache功能。当运行在同一个浏览器上分别来源于www.a.com和www.b.com的两个页面都同时请求同一个跨域资源的时候，如果www.a.com的请求先到达服务器，服务器将资源带上Access-Control-Allow-Origin的Header为www.a.com返回给用户。这个时候www.b.com又发起了请求，浏览器会将Cache的上一次请求返回给用户，这个时候Header的内容和CORS的要求不匹配，就会导致后面的请求失败。

说明：

目前OSS的所有Object相关的接口都提供了CORS相关的验证，另外Multipart相关的接口目前也已经完全支持CORS验证。

GET请求跨域示例

这里举一个使用Ajax从OSS获取数据的例子。为了简单起见，使用的Bucket都是public，访问私有的Bucket的CORS配置是完全一样的，只需要在请求中附加签名即可。

简单开始

首先创建一个Bucket，这里举例为oss-cors-test，将权限设置为public-read，然后这里创建一个test.txt的文本文档，上传到这个Bucket。

单击此处，获取test.txt的访问地址。

说明：

请将下文中的地址换成自己试验的地址。

首先用curl直接访问：

```
curl http://oss-cors-test.oss-cn-hangzhou.aliyuncs.com/test.txt
just for test
```

可见现在已经能正常访问了。

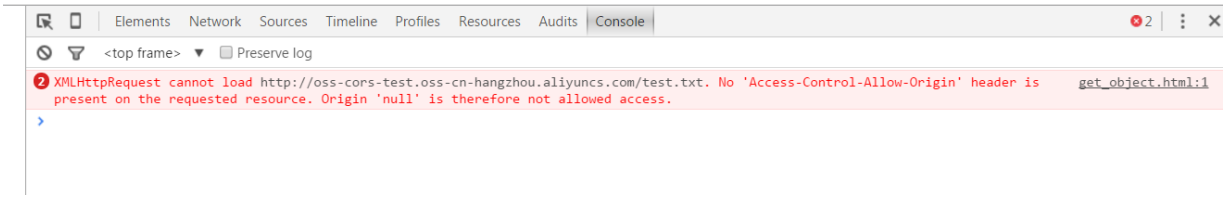
那么我们现在再试着使用Ajax技术来直接访问这个网站。我们写一个最简单的访问的HTML，可以将以下代码复制到本地保存成html文件然后使用浏览器打开。因为没有设置自定义的头，因此这个请求是不需要预检的。

```
<!DOCTYPE html>
<html>
<head>
<script type="text/javascript" src="./functions.js"></script>
</head>
<body>
<script type="text/javascript">
// Create the XHR object.
```

```
function createCORSRequest(method, url) {
  var xhr = new XMLHttpRequest();
  if ("withCredentials" in xhr) {
    // XHR for Chrome/Firefox/Opera/Safari.
    xhr.open(method, url, true);
  } else if (typeof XDomainRequest != "undefined") {
    // XDomainRequest for IE.
    xhr = new XDomainRequest();
    xhr.open(method, url);
  } else {
    // CORS not supported.
    xhr = null;
  }
  return xhr;
}
// Make the actual CORS request.
function makeCorsRequest() {
  // All HTML5 Rocks properties support CORS.
  var url = 'http://oss-cors-test.oss-cn-hangzhou.aliyuncs.com/test.
txt';
  var xhr = createCORSRequest('GET', url);
  if (!xhr) {
    alert('CORS not supported');
    return;
  }
  // Response handlers.
  xhr.onload = function() {
    var text = xhr.responseText;
    var title = text;
    alert('Response from CORS request to ' + url + ': ' + title);
  };
  xhr.onerror = function() {
    alert('Woops, there was an error making the request.');
  };
  xhr.send();
}
</script>
<p align="center" style="font-size: 20px;">
<a href="#" onclick="makeCorsRequest(); return false;">Run Sample</a>
</p>
</body>
</html>
```

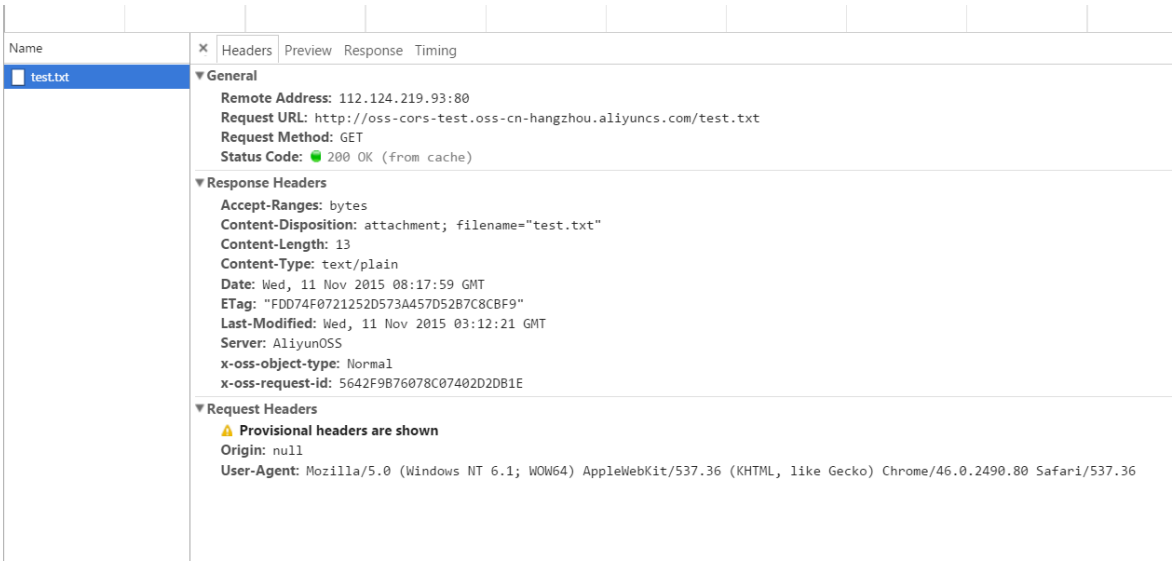
打开之后点击链接（这里以Chrome为例），当然，肯定会失败：

利用Chrome的开发者工具来查看错误原因。



这里告诉我们是因为没有找到Access-Control-Allow-Origin这个头。显然，这就是因为服务器没有配置CORS。

再进入Header界面，可见浏览器发送了带Origin的Request，因此是一个跨域请求，在Chrome上因为是本地文件所以Origin是null。



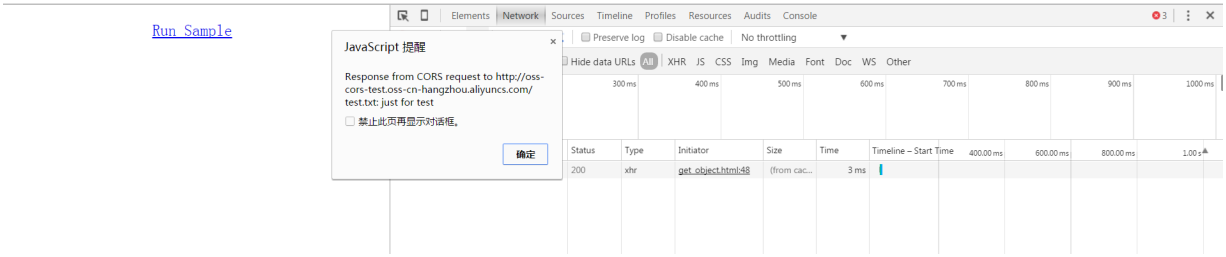
设置 Bucket CORS

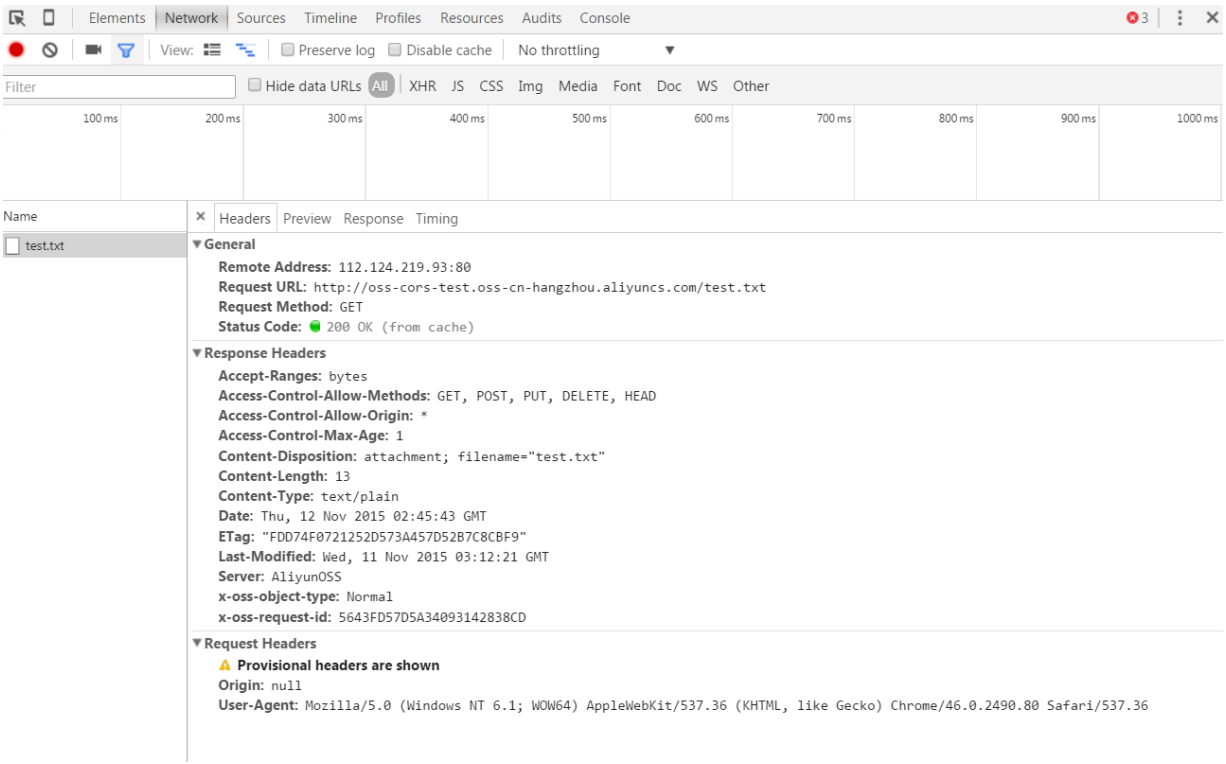
知道了上文的问题，那么现在可以设置Bucket相关的CORS来使上文的例子能够执行成功。为了直观起见，这里使用控制台来完成CORS的设置。如果用户的CORS设置不是很复杂，也建议使用控制台来完成CORS设置。CORS设置的界面如下：

CORS设置是由一条一条规则组成的，真正匹配的时候会从第一条开始逐条匹配，以最早匹配上的规则为准。现在添加第一条规则，使用最宽松的配置：

这里表示的意思就是所有的Origin都允许访问，所有的请求类型都允许访问，所有的Header都允许，最大的缓存时间为1s。

配置完成之后重新测试，结果如下：



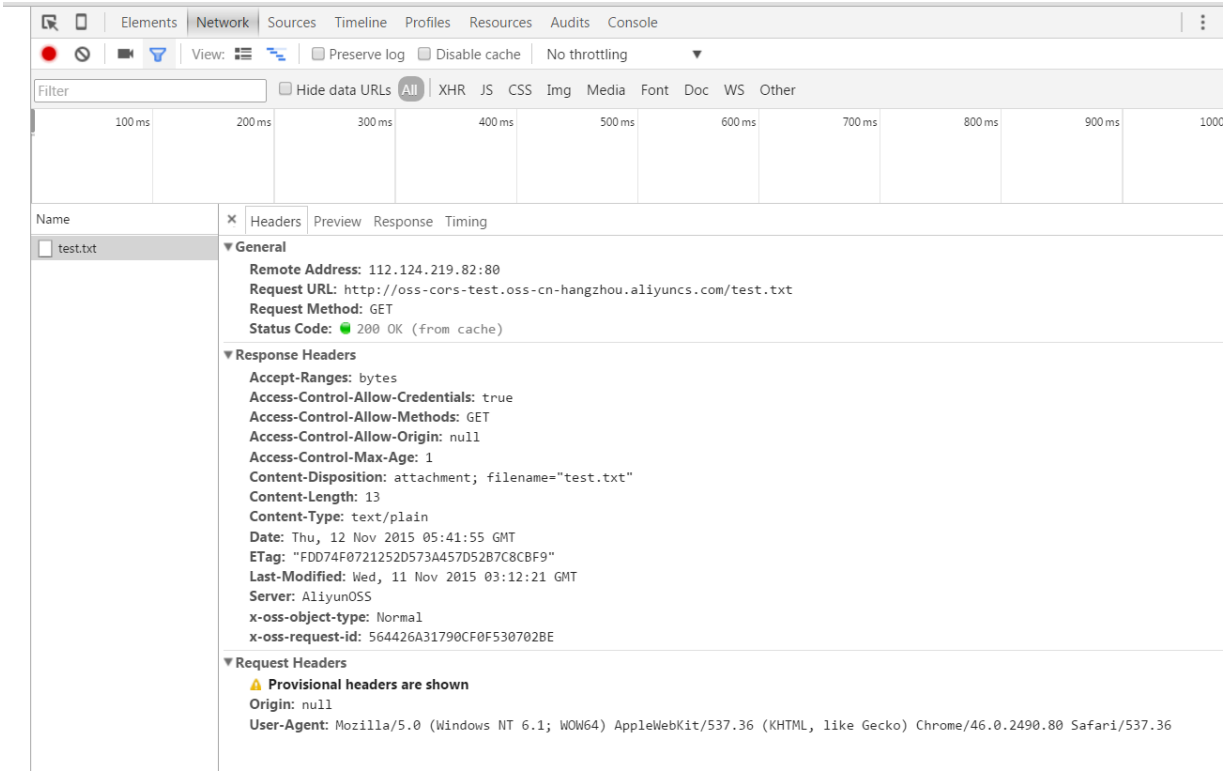


可见现在已经可以正常访问请求了。

因此排查问题来说，如果想要排除跨域带来的访问问题，可以将CORS设置为上面这种配置。这种配置是允许所有的跨域请求的一种配置，因此如果这种配置下依然出错，那么就表明错误出现在其他的部分而不是CORS。

除了最宽松的配置之外，还可以配置更精细的控制机制来实现针对性的控制。比如对于这个场景可以使用如下最小的配置匹配成功：


设定跨域规则
* 来源：
null
来源可以设置多个，每行一个，每行最多能有一个"*"符号
* Allowed Methods：
GET
POST
PUT
DELETE
HEAD
Allowed Headers：
Allowed Headers可以设置多个，每行一个，每行最多能有一个"*"符号
Exposed Headers：
Exposed Headers可以设置多个，每行一个，每行最多能有一个"*"符号
缓存时间：
1
取消
确定



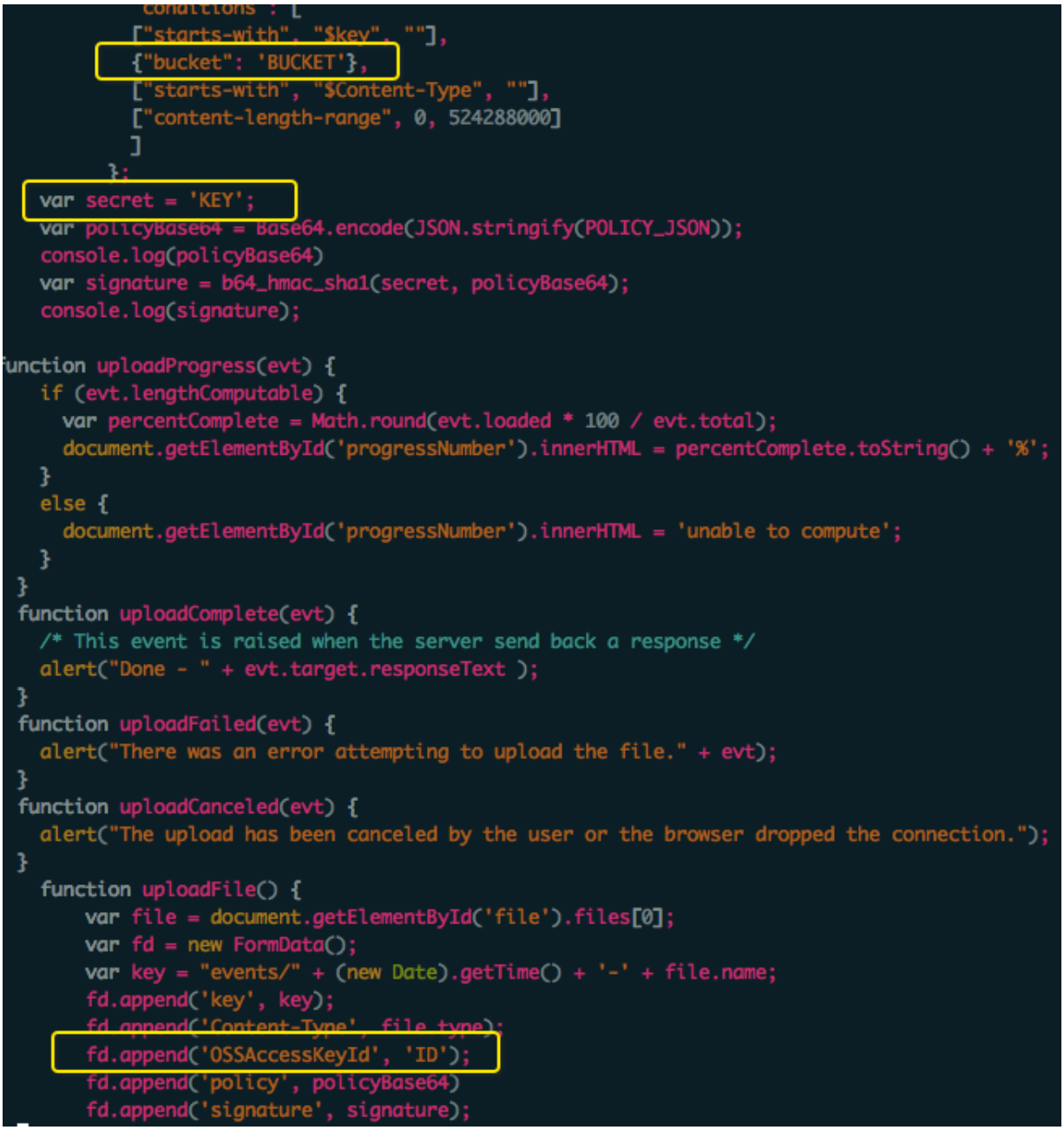
因此对于大部分场景来说，用户最好根据自己的使用场景来使用最小的配置以保证安全性。

利用跨域实现POST上传

这里提供一个更复杂的例子，这里使用了带签名的POST请求，而且需要发送预检请求。

PostObjectSample

> **说明：**
>
> 将上面的代码下载下来之后，将以下对应的部分全部修改成自己对应的内容，然后使用自己的服务器运行起来。



```
        conditions : [
          ["starts-with", "$key", ""],
          {"bucket": 'BUCKET'},
          ["starts-with", "$Content-Type", ""],
          ["content-length-range", 0, 524288000]
          ]
        };
  var secret = 'KEY';
  var policyBase64 = Base64.encode(JSON.stringify(POLICY_JSON));
  console.log(policyBase64)
  var signature = b64_hmac_sha1(secret, policyBase64);
  console.log(signature);

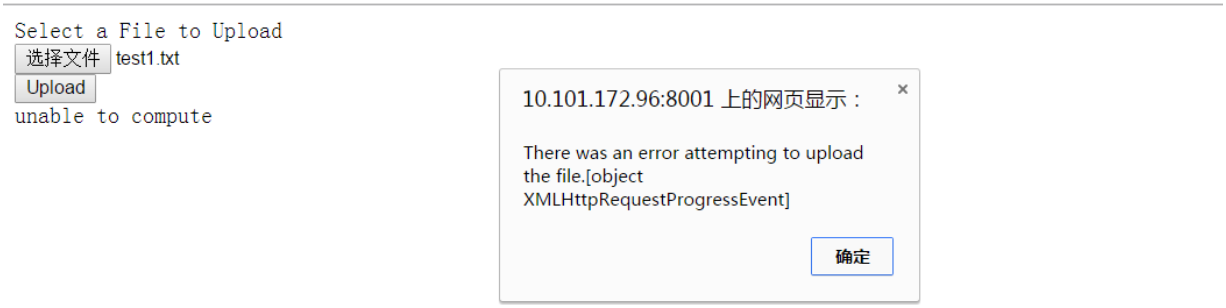
function uploadProgress(evt) {
  if (evt.lengthComputable) {
    var percentComplete = Math.round(evt.loaded * 100 / evt.total);
    document.getElementById('progressNumber').innerHTML = percentComplete.toString() + '%';
  }
  else {
    document.getElementById('progressNumber').innerHTML = 'unable to compute';
  }
}
function uploadComplete(evt) {
  /* This event is raised when the server send back a response */
  alert("Done - " + evt.target.responseText );
}
function uploadFailed(evt) {
  alert("There was an error attempting to upload the file." + evt);
}
function uploadCanceled(evt) {
  alert("The upload has been canceled by the user or the browser dropped the connection.");
}
  function uploadFile() {
      var file = document.getElementById('file').files[0];
      var fd = new FormData();
      var key = "events/" + (new Date).getTime() + '-' + file.name;
      fd.append('key', key);
      fd.append('Content-Type', file.type);
      fd.append('OSSAccessKeyId', 'ID');
      fd.append('policy', policyBase64)
      fd.append('signature', signature);
```

```
    function uploadFile() {
        var file = document.getElementById('file').files[0];
        var fd = new FormData();
        var key = "events/" + (new Date).getTime() + '-' + file.name;
        fd.append('key', key);
        fd.append('Content-Type', file.type);
        fd.append('OSSAccessKeyId', 'ID');
        fd.append('policy', policyBase64)
        fd.append('signature', signature);
        fd.append("file",file);
        var xhr = createXmlHttpRequest()
        xhr.upload.addEventListener("progress", uploadProgress, false);
        xhr.addEventListener("load", uploadComplete, false);
        xhr.addEventListener("error", uploadFailed, false);
        xhr.addEventListener("abort", uploadCanceled, false);

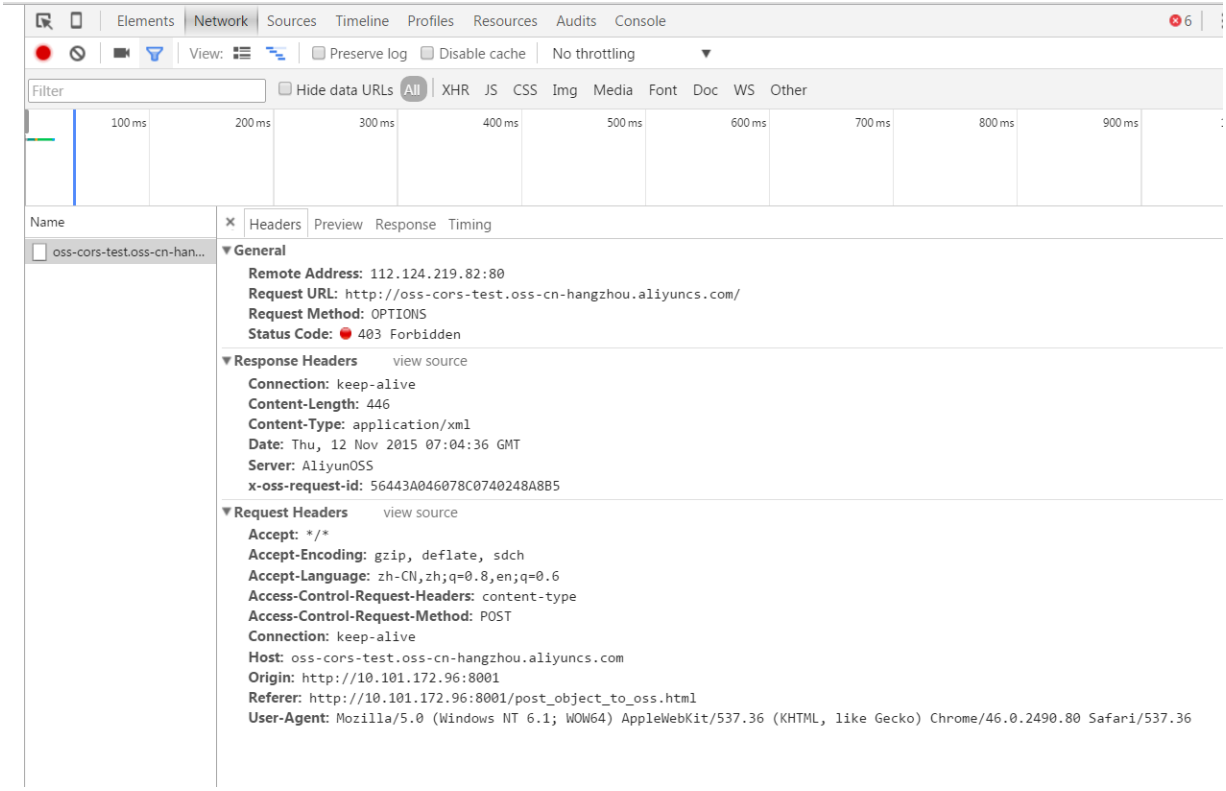
        xhr.open('POST', 'http://BUCKET.HOST', true); //MUST BE LAST LINE BEFORE YOU SEND
        xhr.send(fd);
      }
</script>
```

这里继续使用上文oss-cors-test这个Bucket来进行测试，在测试之前先删除所有的CORS相关的规则，恢复初始状态。

访问这个网页，选择一个文件上传，结果如下：

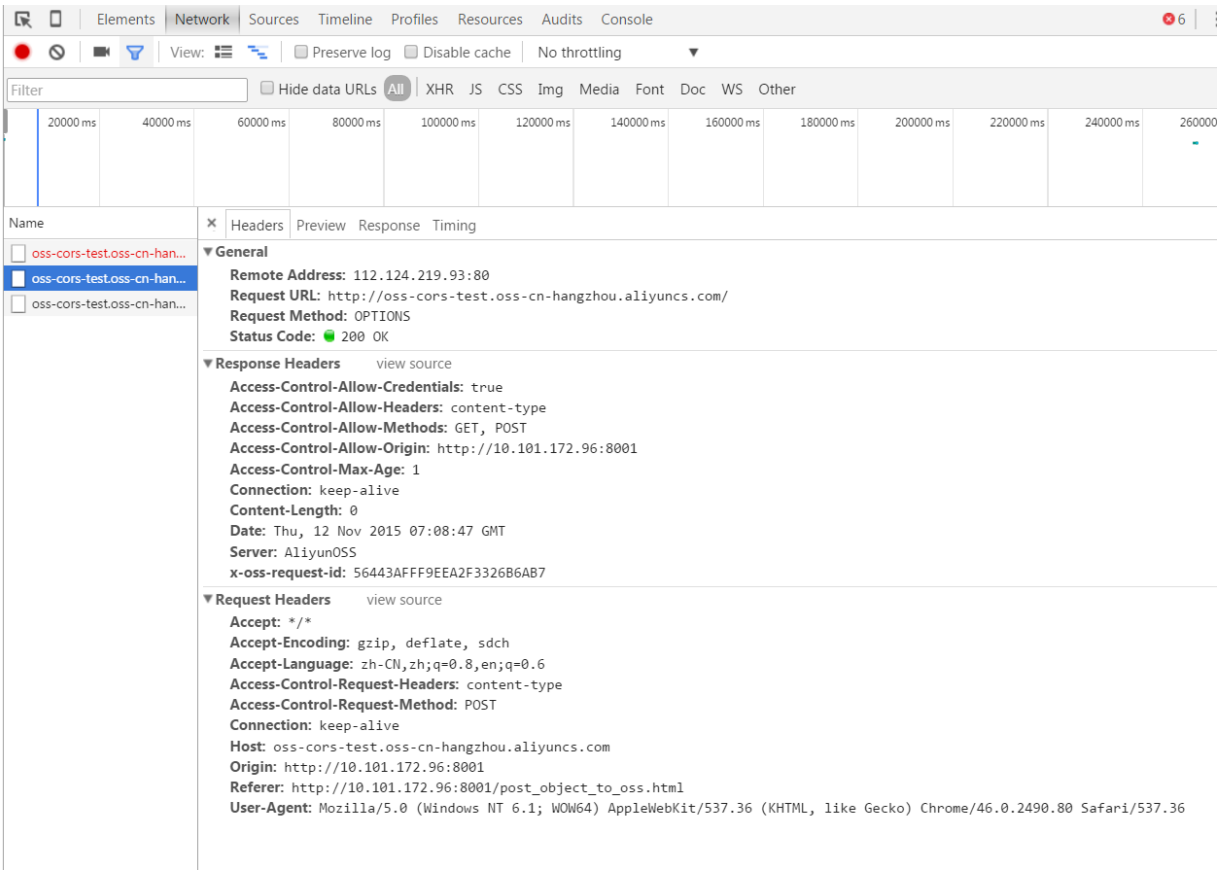


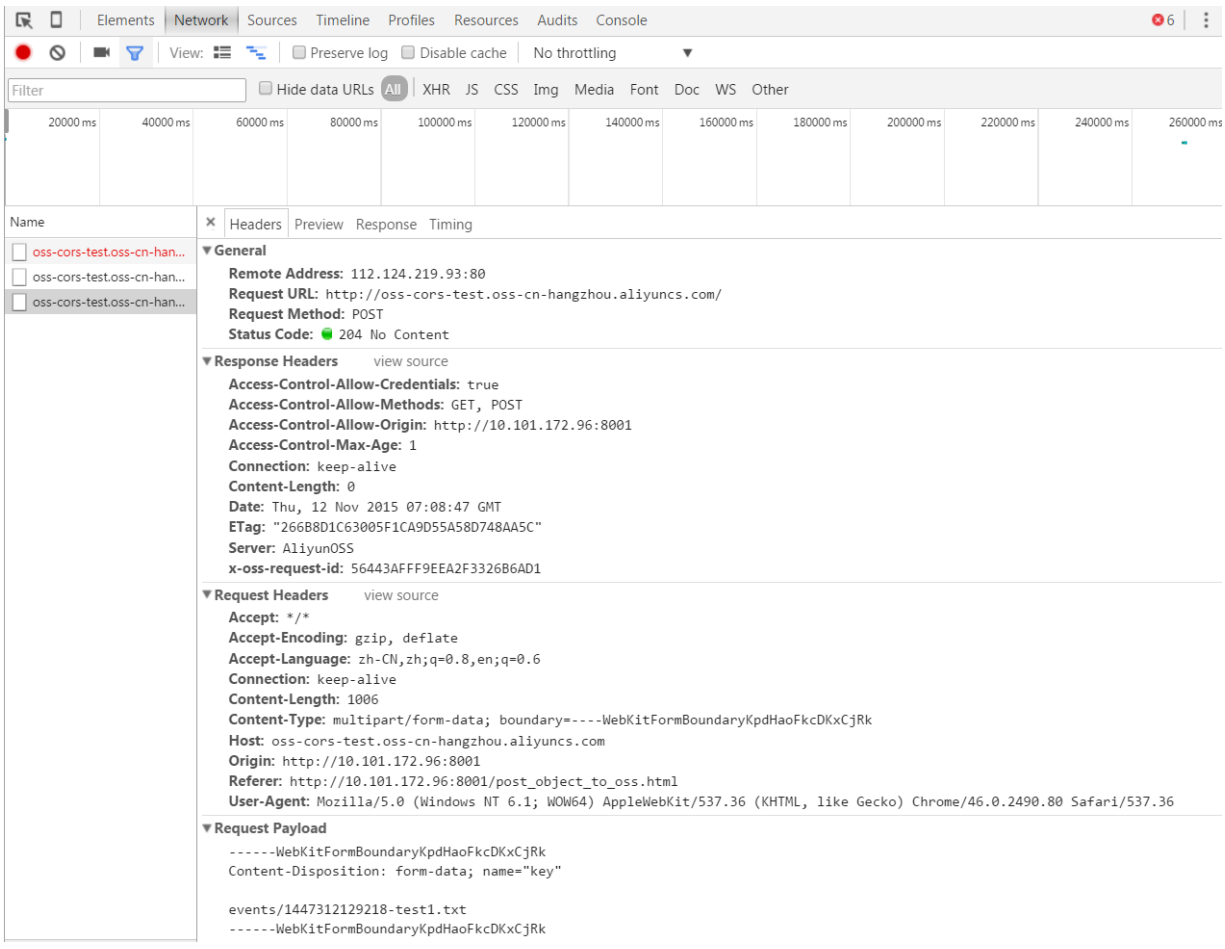
同样打开开发者工具，可以看到以下内容，根据上面的Get的例子，很容易明白同样发生了跨域的错误。不过这里与Get请求的区别在于这是一个带预检的请求，所以可以从图中看到是OPTIONS的Response没有携带CORS相关的Header然后失败了。



那么我们根据对应的信息修改Bucket的CORS配置：

再重新运行，可见已经成功了，从控制台中也可以看到新上传的文件。


Elements Network Sources Timeline Profiles Resources Audits Console
Preserve log Disable cache No throttling
Filter Hide data URLs All XHR JS CSS Img Media Font Doc WS Other
Name
oss-cors-test.oss-cn-han...
Headers Preview Response Timing
General
Remote Address: 112.124.219.93:80
Request URL: http://oss-cors-test.oss-cn-hangzhou.aliyuncs.com/
Request Method: OPTIONS
Status Code: 200 OK
Response Headers view source
Access-Control-Allow-Credentials: true
Access-Control-Allow-Headers: content-type
Access-Control-Allow-Methods: GET, POST
Access-Control-Allow-Origin: http://10.101.172.96:8001
Access-Control-Max-Age: 1
Connection: keep-alive
Content-Length: 0
Date: Thu, 12 Nov 2015 07:08:47 GMT
Server: AliyunOSS
x-oss-request-id: 56443AFFF9EEA2F3326B6AB7
Request Headers view source
Accept: */*
Accept-Encoding: gzip, deflate, sdch
Accept-Language: zh-CN,zh;q=0.8,en;q=0.6
Access-Control-Request-Headers: content-type
Access-Control-Request-Method: POST
Connection: keep-alive
Host: oss-cors-test.oss-cn-hangzhou.aliyuncs.com
Origin: http://10.101.172.96:8001
Referer: http://10.101.172.96:8001/post_object_to_oss.html
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/46.0.2490.80 Safari/537.36


Elements Network Sources Timeline Profiles Resources Audits Console
Preserve log Disable cache No throttling
Filter Hide data URLs All XHR JS CSS Img Media Font Doc WS Other
Name
oss-cors-test.oss-cn-han...
Headers Preview Response Timing
General
Remote Address: 112.124.219.93:80
Request URL: http://oss-cors-test.oss-cn-hangzhou.aliyuncs.com/
Request Method: POST
Status Code: 204 No Content
Response Headers view source
Access-Control-Allow-Credentials: true
Access-Control-Allow-Methods: GET, POST
Access-Control-Allow-Origin: http://10.101.172.96:8001
Access-Control-Max-Age: 1
Connection: keep-alive
Content-Length: 0
Date: Thu, 12 Nov 2015 07:08:47 GMT
ETag: "266B8D1C63005F1CA9D55A58D748AA5C"
Server: AliyunOSS
x-oss-request-id: 56443AFFF9EEA2F3326B6AD1
Request Headers view source
Accept: */*
Accept-Encoding: gzip, deflate
Accept-Language: zh-CN,zh;q=0.8,en;q=0.6
Connection: keep-alive
Content-Length: 1006
Content-Type: multipart/form-data; boundary=----WebKitFormBoundaryKpdHaoFkcDKxCjRk
Host: oss-cors-test.oss-cn-hangzhou.aliyuncs.com
Origin: http://10.101.172.96:8001
Referer: http://10.101.172.96:8001/post_object_to_oss.html
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/46.0.2490.80 Safari/537.36
Request Payload
------WebKitFormBoundaryKpdHaoFkcDKxCjRk
Content-Disposition: form-data; name="key"
events/1447312129218-test1.txt
------WebKitFormBoundaryKpdHaoFkcDKxCjRk

测试一下内容：

```
$curl http://oss-cors-test.oss-cn-hangzhou.aliyuncs.com/events/
1447312129218-test1.txt
post object test
```

CORS配置的一些注意事项

CORS配置一共有以下几项：

- 来源：配置的时候要带上完整的域信息，比如示例中 `http://10.101.172.96:8001`.

  注意不要遗漏了协议名如http，如果端口号不是默认的还要带上端口号之类的。如果不确定的话，可以打开浏览器的调试功能查看Origin头。这一项支持使用通配符*，但是只支持一个，可以根据实际需要灵活配置。
- Method：按照需求开通对应的方法即可。
- Allow Header：允许通过的Header列表，因为这里容易遗漏Header，因此建议没有特殊需求的情况下设置为*。大小写不敏感。
- Expose Header：暴露给浏览器的Header列表，不允许使用通配符。具体的配置需要根据应用的需求来选择，只暴露需要使用的Header，如ETag等。如果不需要暴露这些信息，这里可以不填。如果有特殊需求可以单独指定，大小写不敏感。
- 缓存时间：如果没有特殊情况可以酌情设置的大一点，比如60s。

因此，CORS的一般配置方法就是针对每个可能的访问来源单独配置规则，尽量不要将多个来源混在一个规则中，多个规则之间最好不要有覆盖冲突。其他的权限只开放需要的权限即可。
一些错误排查经验

在调试类似程序的时候，很容易发生一种情况就是将其他错误和CORS错误弄混淆了。

举个例子，当用户因为签名错误访问被拒绝的时候，因为权限验证发生在CORS验证之前，因此返回的结果中并没有带上CORS的Header信息，有些浏览器就会直接报CORS出错，但是实际服务器端的CORS配置是正确的。对于这种情况，我们有两种方式：

- 查看HTTP请求的返回值，因为CORS验证是一个独立的验证，并不影响主干流程，因此像403之类的返回值不可能是由CORS导致的，需要先排除程序原因。如果之前发送了预检请求，那么可

以查看预检请求的结果，如果预检请求中返回了正确的CORS相关的Header，那么表示真实请求应该是被服务器端所允许的，因此错误只可能出现在其他的方面。

- 将服务器端的CORS设置按照如上文所示，改成允许所有来源所有类型的请求都通过的配置，再重新验证。如果还是验证失败，表明有其他的错误发生。

5.5 音视频

5.6 静态网站托管

用户可以基于OSS搭建一个静态网站。本文介绍了如何从申请域名开始，基于OSS搭建一个简单的静态网站。主要的步骤是：

1. 申请一个域名。
2. 开通OSS并创建Bucket。
3. 开通OSS的静态网站托管功能。
4. 使用自定义域名访问OSS。

最后本文会介绍一些常见的问题

静态网站托管功能介绍

简单说就是用户可以基于OSS搭建一个简单的静态网页。用户开启此功能后，OSS提供了一个默认的首页和默认的404页面功能。具体参见开发人员指南中静态网站托管的介绍。

具体实现步骤

1. 申请域名
2. 开通OSS并创建Bucket
   a. 登录OSS控制台，创建一个Bucket为imgleo23，创建在上海，Endpoint为`oss-cn-shanghai.aliyuncs.com`。操作步骤请参见 创建存储空间。
   b. 将Bucket的权限设置为公共读。操作步骤请参见 设置存储空间读写权限。
   c. 上传index.html、error.html、aliyun-logo.png 文件。操作步骤请参见 上传文件。
      - index.html 的内容为：

```
<html>
  <head>
      <title>Hello OSS!</title>
      <meta charset="utf-8">
  </head>
  <body>
```

```
        <p>欢迎使用OSS静态网站的功能</p>
        <p>这是首页</p>
    </body>
</html>
```

- error.html 的内容为：

```
<html>
    <head>
        <title>Hello OSS!</title>
        <meta charset="utf-8">
    </head>
    <body>
        <p>这是OSS静态网站托管的错误首页</p>
    </body>
</html>
```

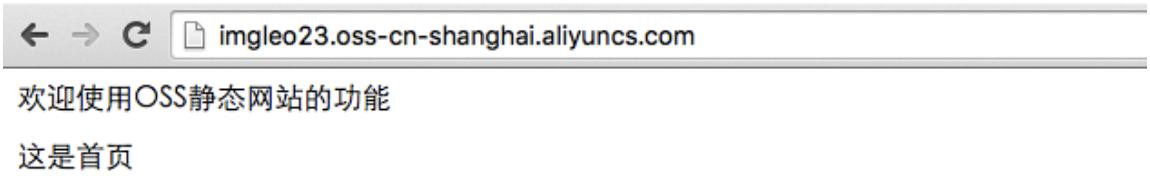
- aliyun-logo.png 是一张图片。

3. 开通OSS的静态网站托管功能

如下图所示，登录控制台后，将 **默认首页** 设置为上文中的 `index.html`，将 **默认404页** 设置为上文中的 `error.html`。具体操作请参见 设置静态网站托管。

检验静态网站托管功能，输入如图所示的URL地址：

- 显示默认的首页



可以看到输入类似URL的时候，会显示开通时指定的index.html中的内容。

- 显示正常的文件

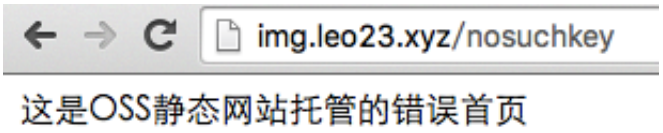
可以看到输入的URL有对应的文件时，会读取成功。

4. 使用自定义域名访问 OSS

关于如何实现自定义域名访问 OSS，请参见开发人员指南中的自定义域名访问 *OSS*。

- 显示默认的首页

- 显示默认的404页



- 显示正常的文件

常见问题及解决方案

- OSS静态网站托管对客户来说有什么好处?

  在用户需求比较简单的时候，且访问量比较小的时候，可以省掉一台ECS。如果访问量大一点，可以考虑结合CDN来使用。

- 价格怎么样？如何和CDN结合？

  价格可以参考官方网站OSS的价格，CDN的价格也可以参考官方网站CND的价格。和CDN结合的例子可以参考*CDN加速OSS实践*。

- 默认的首页和默认的404页面都需要设置吗？

  默认首页需要设置，但默认404页面可以不用设置。

- 为什么输入的URL在浏览器上返回403？

  有可能Bucket的权限不是公开读。也有可能是因为欠费被停止使用。

6 权限管理

6.1 权限管理概述

阿里云权限管理机制包括访问控制（Resource Access Management，简称 RAM）和安全凭证管理（Security Token Service，简称 STS），可以根据需求使用不同权限的子账号来访问OSS，也支持为用户提供临时的访问授权。灵活使用RAM和STS能极大的提高管理的灵活性和安全性。

本文主要讲解了如下的内容：

- *RAM*和*STS*介绍
- 不使用主账号
- 读写权限分离
- *Bucket*权限分离
- 访问控制
- *STS*临时授权访问
- *OSS*权限问题及排查
- *STS*常见问题及排查
- *OSS*子账号设置常见问题

单击 *RAM Policy Editor* 进行在线编辑，即可生成授权策略。

6.2 RAM和STS介绍

RAM和STS是阿里云提供的权限管理系统。

RAM主要的作用是控制账号系统的权限。通过使用RAM可以将在主账号的权限范围内创建子用户，给不同的子用户分配不同的权限从而达到授权管理的目的。

STS是一个安全凭证（Token）的管理系统，用来授予临时的访问权限，这样就可以通过STS来完成对于临时用户的访问授权。

为什么要使用RAM和STS

RAM和STS需要解决的一个核心问题是如何在不暴露主账号的AccessKey的情况下安全的授权别人访问。因为一旦主账号的AccessKey暴露出去的话会带来极大的安全风险，别人可以随意操作该账号下所有的资源，盗取重要信息等。

RAM提供的实际上是一种长期有效的权限控制机制，通过分出不同权限的子账号，将不同的权限分给不同的用户，这样一旦子账号泄露也不会造成全局的信息泄露。但是，由于子账号在一般情况下是长期有效的。

说明：

因此，子账号的AccessKey也是不能泄露的。

相对于RAM提供的长效控制机制，STS提供的是一种临时访问授权，通过STS可以返回临时的AccessKey和Token，这些信息是可以直接发给临时用户用来访问OSS。一般来说从STS获取的权限会受到更加严格的限制，并且拥有时间限制，因此这些信息泄露之后对于系统的影响也很小。

这些功能在下文中会以实际的例子来说明。

基本概念

以下是一些基本概念的简单解释：

- 子账号（RAM account）：从阿里云的主账号中创建出来的子账号，在创建的时候可以分配独立的密码和权限，每个子账号拥有自己AccessKey，可以和阿里云主账号一样正常的完成有权限的操作。一般来说，这里的子账号可以理解为具有某种权限的用户，可以被认为是一个具有某些权限的操作发起者。
- 角色（Role）：表示某种操作权限的虚拟概念，但是没有独立的登录密码和AccessKey。

  说明：

  子账号可以扮演角色，扮演角色的时候的权限是该角色自身的权限。
- 授权策略（Policy）：用来定义权限的规则，比如允许用户读取、或者写入某些资源。
- 资源（Resource）：代表用户可访问的云资源，比如OSS所有的Bucket、或者OSS的某个Bucket、或者OSS的某个Bucket下面的某个Object等。

子账号和角色可以类比为某个个人和其身份的关系，某人在公司的角色是员工，在家里的角色是父亲，在不同的场景扮演不同的角色，但是还是同一个人。在扮演不同的角色的时候也就拥有对应角色的权限。单独的员工或者父亲概念并不能作为一个操作的实体，只有有人扮演了之后才是一个完整的概念。这里还可以体现一个重要的概念，那就是角色可以被多个不同的个人同时扮演。

说明：

完成角色扮演之后，该个人就自动拥有该角色的所有权限。

这里再用一个例子解释一下。

- 某个阿里云用户，名为Alice，其在OSS下有两个私有的Bucket，alice_a和alice_b。alice对这两个Bucket都拥有完全的权限。
- 为了避免阿里云账号的AccessKey泄露导致安全风险，Alice使用RAM创建了两个子账号bob和carol，bob对alice_a拥有读写权限，carol对alice_b拥有读写权限。bob和carol都拥有独立的AccessKey，这样万一泄露了也只会影响其中一个Bucket，而且Alice可以很方便的在控制台取消泄露用户的权限。
- 现在因为某些原因，需要授权给别人读取alice_a中的Object，这种情况下不应该直接把bob的AccessKey透露出去，那么，这个时候可以新建一个角色，比如AliceAReader，给这个角色赋予读取alice_a的权限。但是请注意，这个时候AliceAReader还是没法直接用的，因为并不存在对应AliceAReader的AccessKey，AliceAReader现在仅仅表示一个拥有访问alice_a权限的一个虚拟实体。
- 为了能获取临时授权，这个时候可以调用STS的AssumeRole接口，告诉STS，bob将要扮演AliceAReader这个角色，如果成功，STS会返回一个临时的AccessKeyId、AccessKeySecret还有SecurityToken作为访问凭证。将这个凭证发给需要访问的临时用户就可以获得访问alice_a的临时权限了。凭证过期的时间在调用AssumeRole的时候指定。

为什么RAM和STS这么复杂

乍一看RAM和STS的概念是很复杂的，但这是为了权限控制的灵活性而牺牲了部分的易用性。

将子账号和角色分开，主要是为了将执行操作的实体和代表权限集合的虚拟实体分开。如果用户本身需要的权限很多，比如读写权限，但是实际上每次操作只需要其中的一部分权限，那么我们就可以创建两个角色，分别具有读写权限，然后创建一个没有任何权限但是可以拥有扮演这两个角色权限的用户。当用户需要读的时候就可以临时扮演其中拥有读权限的角色，写的时候同理，将每次操作中权限泄露的风险降低。而且通过扮演角色可以将权限授予其他的阿里云用户，更加方便了协同使用。

当然，提供了灵活性并不代表一定要使用全部的功能，应该根据需求来使用其中的一个子集即可。比如不需要带过期时间的临时访问凭证的话，完全可以只使用RAM的子账号功能而无需使用STS。

下面会用范例提供一些RAM和STS的使用指南，以及使用上的建议。示例在操作上会尽量使用控制台和命令行等操作方式，减少实际代码使用。如果需要使用代码来实现建议参看RAM和STS的API手册。

测试工具

在测试中使用到了osscmd，这是OSS的pythonSDK中的一个工具，可以直接在命令行操作OSS。获取的地址为*PythonSDK*。

osscmd的典型使用方法如下：

```
下载文件
./osscmd get oss://BUCKET/OBJECT LOCALFILE --host=Endpoint -i
AccessKeyId -k AccessKeySecret
这里的BUCKET和OBJECT替换成自己的，Endpoint的格式类似oss-cn-hangzhou.
aliyuncs.com。AccessKeyId和AccessKeySecret使用自己账号对应的
上传文件
./osscmd put LOCALFILE oss://BUCKET/OBJECT  --host=Endpoint -i
AccessKeyId -k AccessKeySecret
各个字段含义和下载一致
```

6.3 不使用主账号

假定目前只有一个ram-test-dev的Bucket用于开发测试。建议不要使用主账号访问这个Bucket，这样可以规避AccessKey或者密码泄露导致的问题。以下内容中出现的AccessKey请替换成用户自己对应的AccessKey。具体操作步骤如下：

1. 在控制台上操作请单击进入**访问控制**。

   说明：

   如果之前没有使用过的需要先开通服务。

2. 在左侧导航栏，单击**用户管理**进入管理页面。
3. 当前显示还没有任何用户。单击右上角的**新建用户**，创建一个拥有和主账号一样可以访问OSS权限的子账号，并勾选为该用户自动生成AccessKey的选项。
4. 生成该账号的AccessKey，务必保存该AK信息便于后续的访问。
5. 返回**用户管理**界面，这里显示一个名为ram_test的账号已创建。创建完成之后，该子账号还是没有任何权限，单击右侧的**授权**，赋予该账号OSS的完全访问权限。

授权完成之后，如果该账号需要控制台登录等权限也可以单击右侧的管理链接来完成操作。

现在可以来测试一下上传下载的操作。这里的AccessKey为ram_test的AccessKey，试验过程中请替换成自己获取到的AccessKey。

```
$./osscmd get
oss://ram-test-dev/test.txt test.txt --host=oss-cn-hangzhou.aliyuncs.
com -i oOhue******Frogv -k OmVwFJO3qcT0******FhOYpg3p0KnA
100%  The object test.txt is downloaded to test.txt, please check.
```

```
0.069(s) elapsed
```

```
$./osscmd put test.txt oss://ram-test-dev/test.txt  --host=oss-cn
-hangzhou.aliyuncs.com -i oOhue******Frogv -k OmVwFJO3qcT0******
FhOYpg3p0KnA
100%
Object URL is: http://ram-test-dev.oss-cn-hangzhou.aliyuncs.com/test.
txt
Object abstract path is: oss://ram-test-dev/test.txt
ETag is "E27172376D49FC609E7F46995E1F808F"
0.108(s) elapsed
```

可见一般的操作已经完全可以使用这个账号来完成，避免了泄露主账号的AccessKey。

6.4 读写权限分离

当用户要使用应用服务器对外服务的时候，OSS可以作为后端静态资源的存储。这个时候应用服务器建议只获取OSS的读权限以降低被攻击的风险。可以设置读写权限分离，给应用服务器一个只读权限的用户即可。

1. 创建一个ram_test_pub的账号，在授权管理的地方选择ReadOnly即可。
2. 使用该账号的AccessKey测试一下上传下载的权限。该AccessKey仅作为示例账号ram_test_pub的AccessKey，实际操作中AccessKey请替换成用户自己对应的AccessKey。

```
$./osscmd get oss://ram-test-dev/test.txt test.txt --host=oss-cn
-hangzhou.aliyuncs.com -i oOhue******Frogv -k OmVwFJO3qcT0******
FhOYpg3p0KnA
100%  The object test.txt is downloaded to test.txt, please check.
0.070(s) elapsed
```

```
$./osscmd put test.txt oss://ram-test-dev/test.txt  --host=oss-cn
-hangzhou.aliyuncs.com -i oOhue******Frogv -k OmVwFJO3qcT0******
FhOYpg3p0KnA
100%  Error Headers:
[('content-length', '229'), ('server', 'AliyunOSS'), ('connection', '
keep-alive'), ('x-oss-request-id', '5646E49C1790CF0F531BAE0D'), ('date
', 'Sat, 14 Nov 2015 07:37:00 GMT'), ('content-type', 'application/xml
')]
Error Body:
<?xml version="1.0" encoding="UTF-8"?>
<Error>
  <Code>AccessDenied</Code>
  <Message>AccessDenied</Message>
  <RequestId>5646E49C1790CF0F531BAE0D</RequestId>
  <HostId>ram-test-dev.oss-cn-hangzhou.aliyuncs.com</HostId>
</Error>
Error Status:
403
put Failed!
```

返回的错误信息提示：使用ram_test_pub账号上传文件失败。

6.5 Bucket权限分离

再来考虑一种场景，已经有别的用户在使用开发的App，那么可以考虑用单独的Bucket来存储用户的App数据，假定为ram-test-app。那么考虑到权限隔离的问题，就不应该让应用服务器访问到ram-test-app，即仅授予ram_test_pub这个账号ram-test-dev的读权限。这也是可以通过RAM的权限系统来完成的。具体操作步骤如下：

1. 由于系统中没有Bucket级别的默认策略，因此需要自定义策略，如下图示：

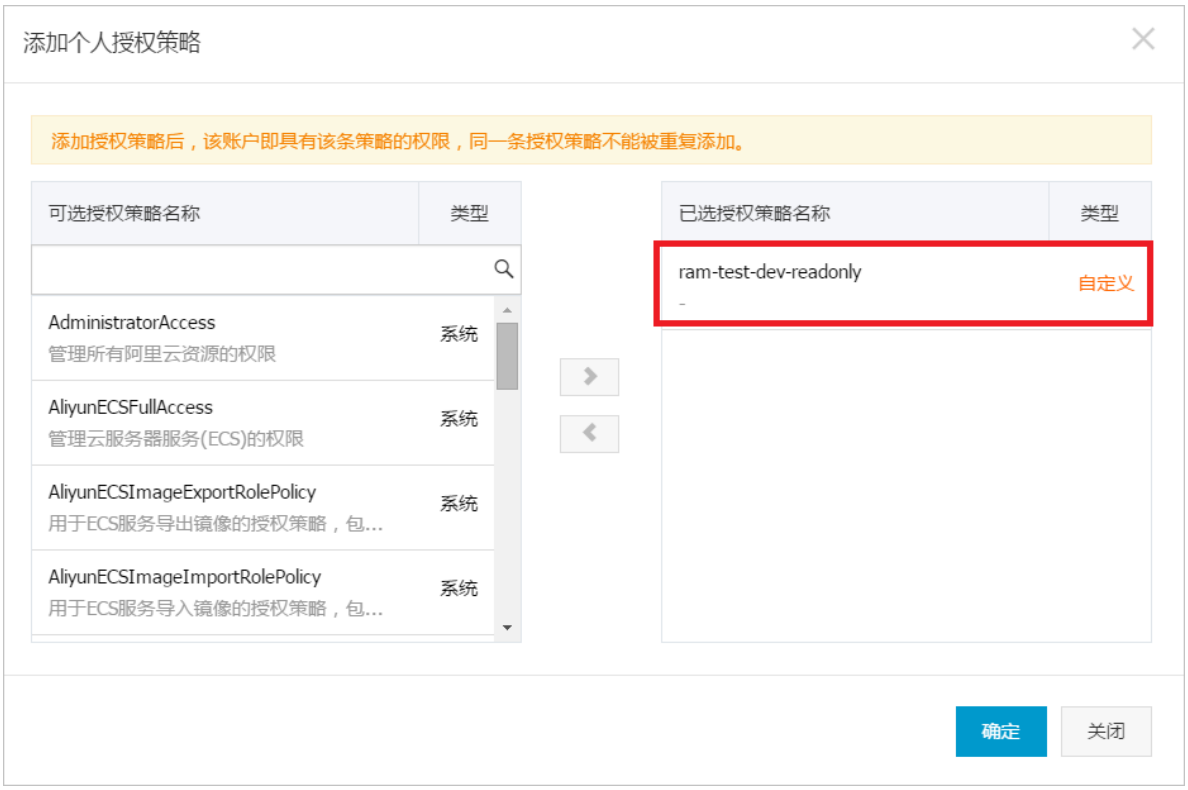
这里Bucket访问的策略如下，其他更详细的可以参考 *RAM策略的说明*和 *OSS授权问题FAQ*。

```
{
   "Version": "1",
   "Statement": [
     {
       "Effect": "Allow",
       "Action": [
         "oss:ListObjects",
         "oss:GetObject"
       ],
       "Resource": [
         "acs:oss:*:*:ram-test-dev",
         "acs:oss:*:*:ram-test-dev/*"
       ]
     }
   ]
```

```
}
```

设置完成之后可以在自定义授权策略列表中查看。

2. 在用户的授权管理中将该策略加入授权范围，并且在用户管理 > 管理 > 用户授权策略中将之前授予的OSS全部可读权限解除。



3. 测试权限设置的有效性。

   - 访问ram-test-dev的Object成功：

```
$./osscmd get oss://ram-test-dev/test.txt test.txt --host=oss-cn
-hangzhou.aliyuncs.com -i oOhue******Frogv -k OmVwFJO3qcT0******
FhOYpg3p0KnA
100%  The object test.txt is downloaded to test.txt, please check.
0.047(s) elapsed
```

   - 访问ram-test-app的Object失败：

```
$./osscmd get oss://ram-test-app/test.txt test.txt --host=oss-cn
-hangzhou.aliyuncs.com -i oOhue******Frogv -k OmVwFJO3qcT0******
FhOYpg3p0KnA
 Error Headers:
 [('content-length', '229'), ('server', 'AliyunOSS'), ('connection
', 'keep-alive'), ('x-oss-request-id', '5646ED53F9EEA2F3324191A2
'), ('date', 'Sat, 14 Nov 2015 08:14:11 GMT'), ('content-type', '
application/xml')]
 Error Body:
 <?xml version="1.0" encoding="UTF-8"?>
 <Error>
   <Code>AccessDenied</Code>
```

```
    <Message>AccessDenied</Message>
    <RequestId>5646ED53F9EEA2F3324191A2</RequestId>
    <HostId>ram-test-app.oss-cn-hangzhou.aliyuncs.com</HostId>
  </Error>
  Error Status:
  403
  get Failed!
```

- 上传文件到oss-test-app失败：

```
$./osscmd put test.txt oss://ram-test-app/test.txt  --host=oss-cn
-hangzhou.aliyuncs.com -i oOhue******Frogv -k OmVwFJO3qcT0******
FhOYpg3p0KnA
 100%  Error Headers:
 [('content-length', '229'), ('server', 'AliyunOSS'), ('connection
', 'keep-alive'), ('x-oss-request-id', '5646ED7BB8DE437A912DC7A8
'), ('date', 'Sat, 14 Nov 2015 08:14:51 GMT'), ('content-type', '
application/xml')]
 Error Body:
 <?xml version="1.0" encoding="UTF-8"?>
 <Error>
   <Code>AccessDenied</Code>
   <Message>AccessDenied</Message>
   <RequestId>5646ED7BB8DE437A912DC7A8</RequestId>
   <HostId>ram-test-app.oss-cn-hangzhou.aliyuncs.com</HostId>
 </Error>
 Error Status:
 403
 put Failed!
```

通过上文的设置，就成功的将ram-test-dev和ram-test-app的权限完全区分开了。

上面介绍的主要是如何使用子账号的权限控制功能来分割权限，将信息泄露造成的危害降到最低。

如果用户有更复杂的权限控制需求，也可以参考 *RAM*用户指南。

6.6 STS临时授权访问

之前章节只用到了RAM的子账号功能，这些子账号都是可以长期正常使用的，发生泄露之后如果无法及时解除权限的话会很危险。

继续上文的例子，当开发者的app被用户使用之后，用户的数据要上传到OSS的ram-test-app这个Bucket，当app的用户数据很多的时候，需要考虑如何才能安全的授权给众多的app用户上传数据呢，以及如何保证多个用户之间存储的隔离。

类似这种需要临时访问的场景可以使用STS来完成。STS可以指定复杂的策略来对特定的用户进行限制，仅提供最小的权限。

创建角色

继续上一章节的例子，App用户有一个名为ram-test-app的Bucket来保存个人数据。创建角色的步骤如下：

1. 按照上文的流程创建一个子账号ram_test_app，不需要赋予任何权限，因为在扮演角色的时候会自动获得被扮演角色的所有权限。
2. 创建角色。这里创建两个角色，一个用于用户读取等操作，一个用于用户上传文件。
   - 打开访问控制的管理控制台，选择角色管理 > 新建角色。
   - 选择角色类型。这里选择 用户角色。
   - 填写类型信息。因为角色是被阿里云账号使用过的，因此选择默认的即可。
   - 配置角色基本信息。

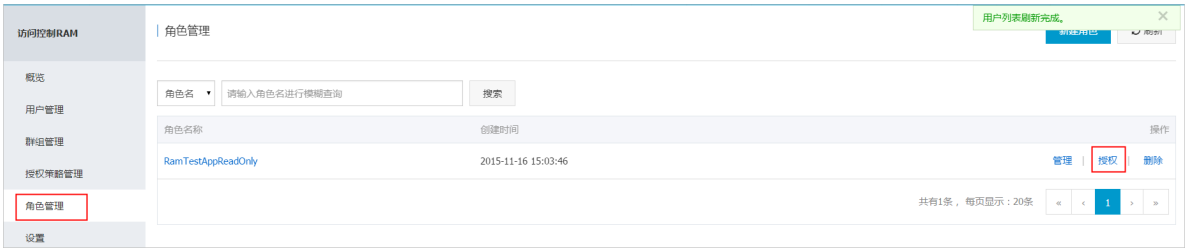
3. 创建完角色之后，角色是没有任何权限的，因此这里和上文所述一样需要新建一个自定义的授权策略。授权策略如下：

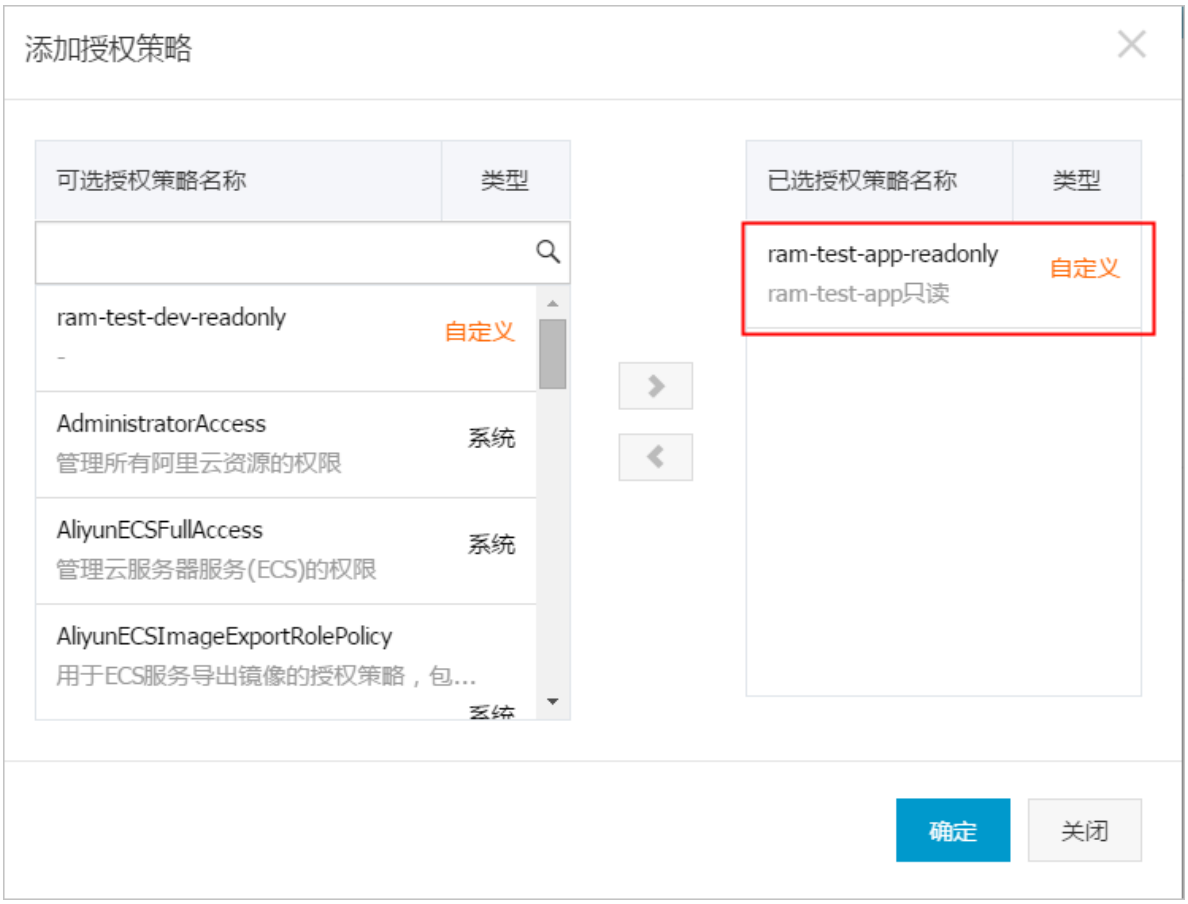
```
{
"Version": "1",
"Statement": [
 {
   "Effect": "Allow",
   "Action": [
     "oss:ListObjects",
     "oss:GetObject"
   ],
   "Resource": [
     "acs:oss:*:*:ram-test-app",
     "acs:oss:*:*:ram-test-app/*"
   ]
 }
]
```

```
}
```

这里表示对ram-test-app拥有只读权限。

4. 建立完成后，即可在角色管理里面给RamTestAppReadOnly添加上ram-test-app的只读授权。

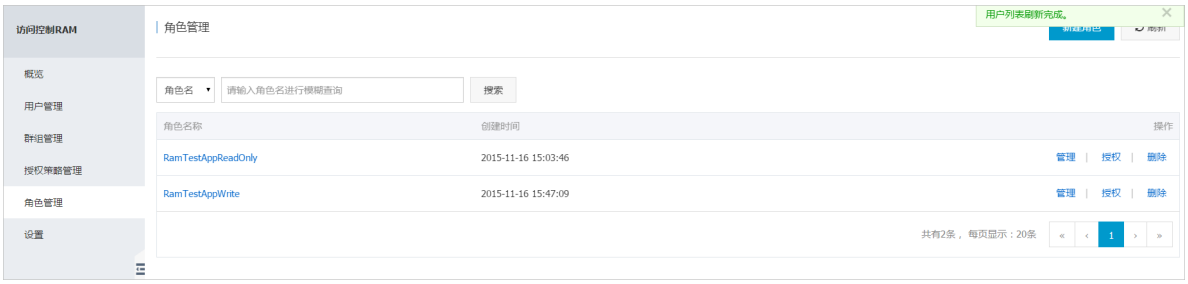




5. 按照上文同样的方法，建立一个RamTestAppWrite的角色，并且赋予写ram-test-app的自定义授权，授权如下：

```
{
"Version": "1",
"Statement": [
 {
   "Effect": "Allow",
   "Action": [
     "oss:DeleteObject",
     "oss:ListParts",
     "oss:AbortMultipartUpload",
     "oss:PutObject"
   ],
   "Resource": [
     "acs:oss:*:*:ram-test-app",
     "acs:oss:*:*:ram-test-app/*"
   ]
 }
]
}
```

现在我们新建好了两个角色：RamTestAppReadOnly和RamTestAppWrite，分别表示了对于ram-test-app的读写权限。



临时授权访问

创建了角色之后，接下来就可以使用临时授权来访问OSS了。

准备工作

在正式使用之前，还有一些工作需要完成。扮演角色也是需要授权的，否则任意子账号都可以扮演这些角色会带来不可预计的风险，因此有扮演对应角色需求的子账号需要显式的配置权限。

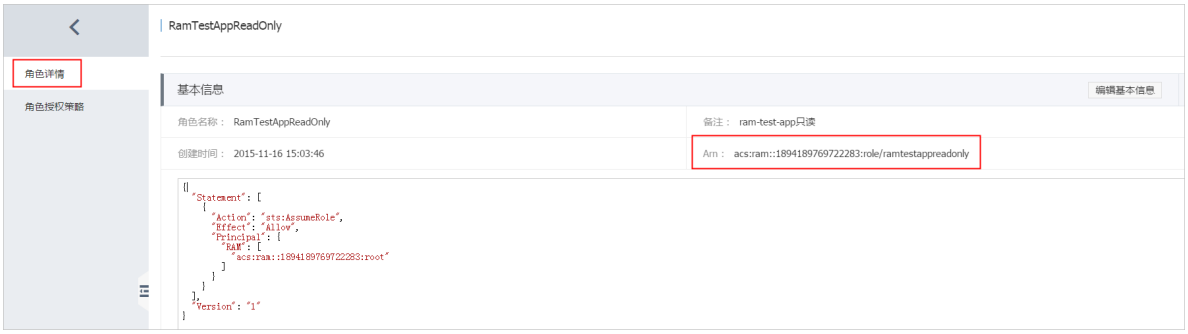
1. 在授权管理策略中新建两个自定义的授权策略，分别如下：


```
{
"Statement": [
 {
   "Action": "sts:AssumeRole",
   "Effect": "Allow",
```

```
    "Resource": "acs:ram::1894189769722283:role/ramtestappreadonly"
  }
],
"Version": "1"
}
```


```
{
"Statement": [
  {
    "Action": "sts:AssumeRole",
    "Effect": "Allow",
    "Resource": "acs:ram::1894189769722283:role/ramtestappwrite"
  }
],
"Version": "1"
}
```

这里Resource后面填写的内容表示某个角色ID，角色的ID可以在角色管理 > 角色详情 中找到。



2. 将这两个授权赋给ram_test_app这个账号。

使用STS授权访问

现在一切准备就绪，可以正式使用STS来授权访问了。

这里使用一个简单的STS的python命令行工具 *sts.py*。 具体的调用方法如下：

```
$python ./sts.py AssumeRole RoleArn=acs:ram::1894189769722283:role
/ramtestappreadonly RoleSessionName=usr001 Policy='{"Version":"1
","Statement":[{"Effect":"Allow","Action":["oss:ListObjects","oss:
GetObject"],"Resource":["acs:oss:*:*:ram-test-app","acs:oss:*:*:ram-
test-app/*"]}]}' DurationSeconds=1000 --id=id --secret=secret
```

- RoleArn表示的是需要扮演的角色ID，角色的ID可以在角色管理 > 角色详情 中找到。
- RoleSessionName是一个用来标示临时凭证的名称，一般来说建议使用不同的应用程序用户来区分。

- Policy表示的是在扮演角色的时候额外加上的一个权限限制。
- DurationSeconds指的是临时凭证的有效期，单位是s，最小为900，最大为3600。
- id和secret表示的是需要扮演角色的子账号的AccessKey。

这里需要解释一下Policy，这里传入的Policy是用来限制扮演角色之后的临时凭证的权限。最后临时凭证获得的权限是角色的权限和这里传入的Policy的交集。

在扮演角色的时候传入Policy的原因是为了灵活性，比如上传文件的时候可以根据不同的用户添加对于上传文件路径的限制，这点会在下面的例子展示。

现在我们可以来实际试验一下STS的作用，作为试验用的Bucket，先在控制台向ram-test-app传入一个test.txt的文本，内容为ststest。

首先使用ram_test_app这个子账号直接来访问。请将下面的AccessKey换成自己试验用的AccessKey。

```
[admin@NGIS-CWWF344M01C /home/admin/oss_test]
$./osscmd get oss://ram-test-app/test.txt test.txt --host=oss-cn
-hangzhou.aliyuncs.com -i oOhue******Frogv -k OmVwFJO3qcT0******
FhOYpg3p0KnA
Error Headers:
[('content-length', '229'), ('server', 'AliyunOSS'), ('connection', '
keep-alive'), ('x-oss-request-id', '564A94D444F4D8B2225E4AFE'), ('date
', 'Tue, 17 Nov 2015 02:45:40 GMT'), ('content-type', 'application/xml
')]
Error Body:
<?xml version="1.0" encoding="UTF-8"?>
<Error>
  <Code>AccessDenied</Code>
  <Message>AccessDenied</Message>
  <RequestId>564A94D444F4D8B2225E4AFE</RequestId>
  <HostId>ram-test-app.oss-cn-hangzhou.aliyuncs.com</HostId>
</Error>
Error Status:
403
get Failed!
[admin@NGIS-CWWF344M01C /home/admin/oss_test]
$./osscmd put test.txt  oss://ram-test-app/test.txt  --host=oss-
cn-hangzhou.aliyuncs.com -i oOhue******Frogv -k OmVwFJO3qcT0******
FhOYpg3p0KnA
100%  Error Headers:
[('content-length', '229'), ('server', 'AliyunOSS'), ('connection', '
keep-alive'), ('x-oss-request-id', '564A94E5B1119B445B9F8C3A'), ('date
', 'Tue, 17 Nov 2015 02:45:57 GMT'), ('content-type', 'application/xml
')]
Error Body:
<?xml version="1.0" encoding="UTF-8"?>
<Error>
  <Code>AccessDenied</Code>
  <Message>AccessDenied</Message>
  <RequestId>564A94E5B1119B445B9F8C3A</RequestId>
  <HostId>ram-test-app.oss-cn-hangzhou.aliyuncs.com</HostId>
</Error>
Error Status:
```

```
403
put Failed!
```

因为ram_test_app这个子账号没有访问权限，因此访问失败。

使用临时授权下载

现在使用STS来下载文件，这里为了简单，传入的Policy和角色的Policy一致，过期时间使用默认的3600s，App的用户假定为usr001。步骤如下：

1. 使用STS来获取临时的凭证。

```
[admin@NGIS-CWWF344M01C /home/admin/oss_test]
 $python ./sts.py AssumeRole RoleArn=acs:ram::1894189769722283:role
/ramtestappreadonly RoleSessionName=usr001 Policy='{"Version":"1
","Statement":[{"Effect":"Allow","Action":["oss:ListObjects","oss:
GetObject"],"Resource":["acs:oss:*:*:ram-test-app","acs:oss:*:*:ram
-test-app/*"]}]}'  --id=oOhue******Frogv --secret=OmVwFJO3qcT0******
FhOYpg3p0KnA
 https://sts.aliyuncs.com/?SignatureVersion=1.0&Format=JSON&
Timestamp=2015-11-17T03%3A07%3A25Z&RoleArn=acs%3Aram%3A%3A18941897
69722283%3Arole%2Framtestappreadonly&RoleSessionName=usr001&
AccessKeyId=oOhuek56i53Frogv&Policy=%7B%22Version%22%3A%221%22%2C%
22Statement%22%3A%5B%7B%22Effect%22%3A%22Allow%22%2C%22Action%22%3A
%5B%22oss%3AListObjects%22%2C%22oss%3AGetObject%22%5D%2C%22Resource
%22%3A%5B%22acs%3Aoss%3A%2A%3A%2A%3Aram-test-app%22%2C%22acs%3Aoss%
3A%2A%3A%2A%3Aram-test-app%2F%2A%22%5D%7D%5D%7D&SignatureMethod=HMAC
-SHA1&Version=2015-04-01&Signature=bshxPZpwRJv5ch3SjaBiXLodwq0%3D&
Action=AssumeRole&SignatureNonce=53e1be9c-8cd8-11e5-9b86-008cfa5e49
38
 {
   "AssumedRoleUser": {
     "Arn": "acs:ram::1894189769722283:role/ramtestappreadonly/
usr001",
     "AssumedRoleId": "317446347657426289:usr001"
   },
   "Credentials": {
     "AccessKeyId": "STS.3mQEbNf******wa180Le",
     "AccessKeySecret": "B1w7rCbR4dzGwNYJ******3PiPqKZ3gjQhAxb6mB",
     "Expiration": "2015-11-17T04:07:25Z",
     "SecurityToken": "CAESvAMIARKAASQQUUTSE+7683CGlhdGsv2/di8uI+
X1BxG7MDxM5FTd0fp5wpPK/7UctYH2MJ///c4yMN1PUCcEHI1zppCINmpDG2XeNA3
OS16JwS6ESmI50sHyWBmsYkCJW15gXnfhz/OK+mSp1bYxlfB33qfgCFe97Ijeuj8RMgq
Fx0Hny2BzGhhTVFMuM21RRWJOZnR5Yzl1T3dhMTgwTGUiEjMxNzQ0NjM0NzY
1NzQyNjI4OSoGdXNyMDAxMJTrgJ2RKjoGUnNhTUQ1QpsBCgExGpUBCgVBbGx
vdxI4CgxBY3Rpb25FcXVhbHMSBkFjdGlvbhogCg9vc3M6TGlzdE9iamVjdHM
KDW9zczpHZXRPYmplY3QSUgoOUmVzb3VyY2VFcXVhbHMSCFJlc291cmNlGjY
KGGFjczpvc3M6KjoqOnJhbS10ZXN0LWFwcAoaYWNzOm9zczoqOio6cmFtLXR
lc3QtYXBwLypKEDE4OTQxODk3Njk3MjIyODNSBTI2ODQyWg9Bc3N1bWVkUm9
sZVVzZXJgAGoSMzE3NDQ2MzQ3NjU3NDI2Mjg5chJyYW10ZXN0YXBwcmVhZG9ubHk="
   },
   "RequestId": "8C009F64-F19D-4EC1-A3AD-7A718CD0B49B"
 }
```

2. 使用临时凭证来下载文件，这里sts_token就是上面STS返回的SecurityToken。

```
[admin@NGIS-CWWF344M01C /home/admin/oss_test]
 $./osscmd get oss://ram-test-app/test.txt test.txt --host=oss-cn
-hangzhou.aliyuncs.com -i STS.3mQEbNf******wa180Le -k B1w7rCbR4d
```

```
zGwNYJ******3PiPqKZ3gjQhAxb6mB --sts_token=CAESvAMIARKAASQQUUTSE+
7683CGlhdGsv2/di8uI+X1BxG7MDxM5FTd0fp5wpPK/7UctYH2MJ///c4yMN1PUCc
EHI1zppCINmpDG2XeNA3OS16JwS6ESmI50sHyWBmsYkCJW15gXnfhz/OK+mSp1bYxlfB
33qfgCFe97Ijeuj8RMgqFx0Hny2BzGhhTVFMuM21RRWJOZnR5Yzl1T3dhMTg
wTGUiEjMxNzQ0NjM0NzY1NzQyNjI4OSoGdXNyMDAxMJTrgJ2RKjoGUnNhTUQ
1QpsBCgExGpUBCgVBbGxvdxI4CgxBY3Rpb25FcXVhbHMSBkFjdGlvbhogCg9
vc3M6TGlzdE9iamVjdHMKDW9zczpHZXRPYmplY3QSUgoOUmVzb3VyY2VFcXV
hbHMSCFJlc291cmNlGjYKGGFjczpvc3M6KjoqOnJhbS10ZXN0LWFwcAoaYWN
zOm9zczoqOio6cmFtLXRlc3QtYXBwLypKEDE4OTQxODk3Njk3MjIyODNSBTI
2ODQyWg9Bc3N1bWVkUm9sZVVzZXJgAGoSMzE3NDQ2MzQ3NjU3NDI2Mjg5chJ
yYW10ZXN0YXBwcmVhZG9ubHk=
 100%  The object test.txt is downloaded to test.txt, please check.
 0.061(s) elapsed
```

3. 可见已经可以使用临时凭证来下载文件了，那再试着使用这个凭证来上传。

```
[admin@NGIS-CWWF344M01C /home/admin/oss_test]
 $./osscmd put test.txt oss://ram-test-app/test.txt  --host=oss-
cn-hangzhou.aliyuncs.com -i STS.3mQEbNf******wa180Le -k B1w7rCbR4d
zGwNYJ******3PiPqKZ3gjQhAxb6mB --sts_token=CAESvAMIARKAASQQUUTSE+
7683CGlhdGsv2/di8uI+X1BxG7MDxM5FTd0fp5wpPK/7UctYH2MJ///c4yMN1PUCc
EHI1zppCINmpDG2XeNA3OS16JwS6ESmI50sHyWBmsYkCJW15gXnfhz/OK+mSp1bYxlfB
33qfgCFe97Ijeuj8RMgqFx0Hny2BzGhhTVFMuM21RRWJOZnR5Yzl1T3dhMTg
wTGUiEjMxNzQ0NjM0NzY1NzQyNjI4OSoGdXNyMDAxMJTrgJ2RKjoGUnNhTUQ
1QpsBCgExGpUBCgVBbGxvdxI4CgxBY3Rpb25FcXVhbHMSBkFjdGlvbhogCg9
vc3M6TGlzdE9iamVjdHMKDW9zczpHZXRPYmplY3QSUgoOUmVzb3VyY2VFcXV
hbHMSCFJlc291cmNlGjYKGGFjczpvc3M6KjoqOnJhbS10ZXN0LWFwcAoaYWN
zOm9zczoqOio6cmFtLXRlc3QtYXBwLypKEDE4OTQxODk3Njk3MjIyODNSBTI
2ODQyWg9Bc3N1bWVkUm9sZVVzZXJgAGoSMzE3NDQ2MzQ3NjU3NDI2Mjg5chJ
yYW10ZXN0YXBwcmVhZG9ubHk=
 100%  Error Headers:
 [('content-length', '254'), ('server', 'AliyunOSS'), ('connection
', 'keep-alive'), ('x-oss-request-id', '564A9A2A1790CF0F53C15C82
'), ('date', 'Tue, 17 Nov 2015 03:08:26 GMT'), ('content-type', '
application/xml')]
 Error Body:
 <?xml version="1.0" encoding="UTF-8"?>
 <Error>
   <Code>AccessDenied</Code>
   <Message>Access denied by authorizer's policy.</Message>
   <RequestId>564A9A2A1790CF0F53C15C82</RequestId>
   <HostId>ram-test-app.oss-cn-hangzhou.aliyuncs.com</HostId>
 </Error>
 Error Status:
 403
 put Failed!
```

由于扮演的角色只有下载的权限，因此上传失败。

使用临时授权上传

现在可以来试验一下使用STS上传。步骤如下：

1. 获取STS的临时凭证，App用户为usr001。

```
[admin@NGIS-CWWF344M01C /home/admin/oss_test]
 $python ./sts.py AssumeRole RoleArn=acs:ram::1894189769722283:role
/ramtestappwrite RoleSessionName=usr001 Policy='{"Version":"1","
Statement":[{"Effect":"Allow","Action":["oss:PutObject"],"Resource":
```

```
["acs:oss:*:*:ram-test-app/usr001/*"]}]}'  --id=oOhue******Frogv --
secret=OmVwFJO3qcT0******FhOYpg3p0KnA
 https://sts.aliyuncs.com/?SignatureVersion=1.0&Format=JSON&
Timestamp=2015-11-17T03%3A16%3A10Z&RoleArn=acs%3Aram%3A%3A18941897
69722283%3Arole%2Framtestappwrite&RoleSessionName=usr001&AccessKeyI
d=oOhuek56i53Frogv&Policy=%7B%22Version%22%3A%221%22%2C%22Statemen
t%22%3A%5B%7B%22Effect%22%3A%22Allow%22%2C%22Action%22%3A%5B%22oss
%3APutObject%22%5D%2C%22Resource%22%3A%5B%22acs%3Aoss%3A%2A%3A%2A%
3Aram-test-app%2Fusr001%2F%2A%22%5D%7D%5D%7D&SignatureMethod=HMAC-
SHA1&Version=2015-04-01&Signature=Y0OPUoL1PrCqX4X6A3%2FJvgXuS6c%3D&
Action=AssumeRole&SignatureNonce=8d0798a8-8cd9-11e5-9f49-008cfa5e49
38
 {
   "AssumedRoleUser": {
     "Arn": "acs:ram::1894189769722283:role/ramtestappwrite/usr001
",
     "AssumedRoleId": "355407847660029428:usr001"
   },
   "Credentials": {
     "AccessKeyId":  "STS.rtfx13******NlIJlS4U",
     "AccessKeySecret": "2fsaM8E2maB2dn******wpsKTyK4ajo7TxFr0zIM",
     "Expiration": "2015-11-17T04:16:10Z",
     "SecurityToken": "CAESkwMIARKAAUh3/Uzcg13YLRBWxy0IZjGew
Mpg31ITxCleBFU1eO/3Sgpudid+GVs+Olvu1vXJn6DLcvPa8azKJKtzV0oKSy+
mwUrxSvUSRVDntrs78CsNfWoOJUMJKjLIxdWnGi1pgxJCBzNZ2YV/6ycTaZySSE
1V6kqQ7A+GPwYoBSnWmLpdGhhTVFMucnRmeDEzRFlNVWJjTmxJSmxTNFUiE
jM1NTQwNzg0NzY2MDAyOTQyOCoGdXNyMDAxMOPzoJ2RKjoGUnNhTUQ1QnYKA
TEacQoFQWxsb3cSJwoMQWN0aW9uRXF1YWxzEgZBY3Rpb24aDwoNb3NzOlB1d
E9iamVjdBI/Cg5SZXNvdXJjZUVxdWFscxIIUmVzb3VyY2UaIwohYWNzOm9zcz
oqOio6cmFtLXRlc3QtYXBwL3VzcjAwMS8qShAxODk0MTg5NzY5NzIyMjgzUg
UyNjg0MloPQXNzdW1lZFJvbGVVc2VyYABqEjM1NTQwNzg0NzY2MDAyOTQyOH
IPcmFtdGVzdGFwcHdyaXRl"
   },
   "RequestId": "19407707-54B2-41AD-AAF0-FE87E8870B0D"
 }
```

2. 试验一下能否使用这个凭证来上传下载。

```
[admin@NGIS-CWWF344M01C /home/admin/oss_test]
 $./osscmd get oss://ram-test-app/test.txt test.txt --host=oss-cn-
hangzhou.aliyuncs.com -i STS.rtfx13******NlIJlS4U -k 2fsaM8E2maB2dn
******wpsKTyK4ajo7TxFr0zIM --sts_token=CAESkwMIARKAAUh3/Uzcg13YLRB
Wxy0IZjGewMpg31ITxCleBFU1eO/3Sgpudid+GVs+Olvu1vXJn6DLcvPa8azK
JKtzV0oKSy+mwUrxSvUSRVDntrs78CsNfWoOJUMJKjLIxdWnGi1pgxJCBzNZ2
YV/6ycTaZySSE1V6kqQ7A+GPwYoBSnWmLpdGhhTVFMucnRmeDEzRFlNVWJjTmx
JSmxTNFUiEjM1NTQwNzg0NzY2MDAyOTQyOCoGdXNyMDAxMOPzoJ2RKjoGUnN
hTUQ1QnYKATEacQoFQWxsb3cSJwoMQWN0aW9uRXF1YWxzEgZBY3Rpb24aDwo
Nb3NzOlB1dE9iamVjdBI/Cg5SZXNvdXJjZUVxdWFscxIIUmVzb3VyY2UaIwoh
YWNzOm9zczoqOio6cmFtLXRlc3QtYXBwL3VzcjAwMS8qShAxODk0MTg5NzY5
NzIyMjgzUgUyNjg0MloPQXNzdW1lZFJvbGVVc2VyYABqEjM1NTQwNzg0NzY2
MDAyOTQyOHIPcmFtdGVzdGFwcHdyaXRl
 Error Headers:
 [('content-length', '254'), ('server', 'AliyunOSS'), ('connection
', 'keep-alive'), ('x-oss-request-id', '564A9C31FFFC811F24B6E7E3
'), ('date', 'Tue, 17 Nov 2015 03:17:05 GMT'), ('content-type', '
application/xml')]
 Error Body:
 <?xml version="1.0" encoding="UTF-8"?>
 <Error>
   <Code>AccessDenied</Code>
   <Message>Access denied by authorizer's policy.</Message>
   <RequestId>564A9C31FFFC811F24B6E7E3</RequestId>
```

```
    <HostId>ram-test-app.oss-cn-hangzhou.aliyuncs.com</HostId>
 </Error>
 Error Status:
 403
 get Failed!
 [admin@NGIS-CWWF344M01C /home/admin/oss_test]
 $./osscmd put test.txt oss://ram-test-app/test.txt  --host=oss-cn-
hangzhou.aliyuncs.com -i STS.rtfx13******NlIJlS4U -k 2fsaM8E2maB2dn
******wpsKTyK4ajo7TxFr0zIM --sts_token=CAESkwMIARKAAUh3/Uzcg13YLRB
Wxy0IZjGewMpg31ITxCleBFU1eO/3Sgpudid+GVs+Olvu1vXJn6DLcvPa8azK
JKtzV0oKSy+mwUrxSvUSRVDntrs78CsNfWoOJUMJKjLIxdWnGi1pgxJCBzNZ2
YV/6ycTaZySSE1V6kqQ7A+GPwYoBSnWmLpdGhhTVFMucnRmeDEzRFlNVWJjTmx
JSmxTNFUiEjM1NTQwNzg0NzY2MDAyOTQyOCoGdXNyMDAxMOPzoJ2RKjoGUnN
hTUQ1QnYKATEacQoFQWxsb3cSJwoMQWN0aW9uRXF1YWxzEgZBY3Rpb24aDwo
Nb3NzOlB1dE9iamVjdBI/Cg5SZXNvdXJjZUVxdWFscxIIUmVzb3VyY2UaIwoh
YWNzOm9zczoqOio6cmFtLXRlc3QtYXBwL3VzcjAwMS8qShAxODk0MTg5NzY5
NzIyMjgzUgUyNjg0MloPQXNzdW1lZFJvbGVVc2VyYABqEjM1NTQwNzg0NzY2
MDAyOTQyOHIPcmFtdGVzdGFwcHdyaXRl
 100%  Error Headers:
 [('content-length', '254'), ('server', 'AliyunOSS'), ('connection
', 'keep-alive'), ('x-oss-request-id', '564A9C3FB8DE437A91B16772
'), ('date', 'Tue, 17 Nov 2015 03:17:19 GMT'), ('content-type', '
application/xml')]
 Error Body:
 <?xml version="1.0" encoding="UTF-8"?>
 <Error>
   <Code>AccessDenied</Code>
   <Message>Access denied by authorizer's policy.</Message>
   <RequestId>564A9C3FB8DE437A91B16772</RequestId>
   <HostId>ram-test-app.oss-cn-hangzhou.aliyuncs.com</HostId>
 </Error>
 Error Status:
 403
 put Failed!
```

这里出现了问题，上传test.txt失败了。将本小节开始的时候传入的Policy格式化之后如下：

```
{
     "Version": "1",
         "Statement": [
         {
             "Effect": "Allow",
             "Action": [
                 "oss:PutObject"
                 ],
             "Resource": [
                 "acs:oss:*:*:ram-test-app/usr001/*"
                 ]
         }
     ]
 }
```

这个Policy的意义是仅允许用户向ram-test-app这个Bucket上传类似usr001/的文件，如果App用户是usr002的时候，就可以修改Policy为仅允许上传类似usr002/这种类型的文件，通过这种对不同的App用户设定不同的Policy的方式，可以做到不同App用户之间拥有独立的存储空间互不干扰的目的。

3. 重新试验，将上传的目标指定为ram-test-app/usr001/test.txt。

```
[admin@NGIS-CWWF344M01C /home/admin/oss_test]
 $./osscmd put test.txt oss://ram-test-app/usr001/test.txt  --
host=oss-cn-hangzhou.aliyuncs.com -i STS.rtfx13******NlIJlS4U -k
 2fsaM8E2maB2dn******wpsKTyK4ajo7TxFr0zIM --sts_token=CAESkwMIAR
KAAUh3/Uzcg13YLRBWxy0IZjGewMpg31ITxCleBFU1eO/3Sgpudid+GVs+Olvu1vXJn6
DLcvPa8azKJKtzV0oKSy+mwUrxSvUSRVDntrs78CsNfWoOJUMJKjLIxdWnGi1
pgxJCBzNZ2YV/6ycTaZySSE1V6kqQ7A+GPwYoBSnWmLpdGhhTVFMucnRmeDEzR
FlNVWJjTmxJSmxTNFUiEjM1NTQwNzg0NzY2MDAyOTQyOCoGdXNyMDAxMOPzo
J2RKjoGUnNhTUQ1QnYKATEacQoFQWxsb3cSJwoMQWN0aW9uRXF1YWxzEgZBY
3Rpb24aDwoNb3NzOlB1dE9iamVjdBI/Cg5SZXNvdXJjZUVxdWFscxIIUmVzb3
VyY2UaIwohYWNzOm9zczoqOio6cmFtLXRlc3QtYXBwL3VzcjAwMS8qShAxOD
k0MTg5NzY5NzIyMjgzUgUyNjg0MloPQXNzdW1lZFJvbGVVc2VyYABqEjM1NT
QwNzg0NzY2MDAyOTQyOHIPcmFtdGVzdGFwcHdyaXRl
 100%
 Object URL is: http://ram-test-app.oss-cn-hangzhou.aliyuncs.com/
usr001%2Ftest.txt
 Object abstract path is: oss://ram-test-app/usr001/test.txt
 ETag is "946A0A1AC8245696B9C6A6F35942690B"
 0.071(s) elapsed
```

可见上传成功了。

总结

本章主要介绍了使用STS来临时授权用户访问OSS。在典型的移动开发场景中，使用STS可以做到不同的App用户需要访问App的时候，可以通过获取到的临时授权来访问OSS。临时授权可以指定过期时间，因此大大降低了泄露的危害。在获取临时授权的时候，可以根据App用户的不同，传入不同的授权策略来限制用户的访问权限，比如限制用户访问的Object路径，从而达到隔离不同App用户的存储空间的目的。

6.7 OSS子账号设置常见问题

如何生成STS临时账户？如何应用STS临时账户访问资源？

请参考：*STS*临时授权访问

子账号授权权限，客户端/控制台登录报错

请参考：子用户已经被授权了某*Bucket*权限#为什么登录*OSS*控制台访问时提示没有操作权限

如何为子账户授权单个bucket权限？

请参考：怎样授权子用户完全管理某个*Bucket*的权限

如何为子账户授权bucket下某个目录的权限？

请参考：*OSS*目录级别的授权

如何为子账户授权某个bucket的只读权限？

请参考：授权一个子用户列出并读取一个 *Bucket* 中的资源

OSS SDK 调用报错：InvalidAccessKeyId

请参考：*STS*常见问题及排查

STS调用报错：Access denied by authorizer's policy

详细错误：ErrorCode: AccessDenied ErrorMessage: Access denied by authorizer's policy.

该错误的原因是：临时用户访问无权限，该临时用户角色扮演指定授权策略，该授权策略无权限。

更多STS错误提示以及对应原因，请参考：*OSS* 权限问题及排查

7 音视频

8 数据安全

8.1 通过crc64校验数据传输的完整性

背景

数据在客户端和服务器之间传输时有可能会出错。OSS现在支持对各种方式上传的Object返回其crc64值，客户端可以和本地计算的crc64值做对比，从而完成数据完整性的验证。

- OSS对新上传的Object进行crc64的计算，并将结果存储为Object的元信息存储，随后在返回的response header中增加`x-oss-hash-crc64ecma`头部，表示其crc64值，该64位CRC根据*ECMA-182*标准 计算得出。
- 对于crc64上线之前就已经存在于OSS上的Object，OSS不会对其计算crc64值，所以获取此类Object时不会返回其crc64值。

操作说明

- Put Object / Append Object / Post Object / Multipart upload part 均会返回对应的crc64值，客户端可以在上传完成后拿到服务器返回的crc64值和本地计算的数值进行校验。
- Multipart Complete时，如果所有的Part都有crc64值，则会返回整个Object的crc64值；否则，比如有crc64上线之前就已经上传的Part，则不返回crc64值。
- Get Object / Head Object / Get ObjectMeta 都会返回对应的crc64值（如有）。客户端可以在Get Object完成后，拿到服务器返回的crc64值和本地计算的数值进行校验。

  说明：

  range get请求返回的将会是整个Object的crc64值。
- Copy相关的操作，如Copy Object / Upload Part Copy，新生成的Object/Part不保证具有crc64值。

应用示例

以下为完整的Python示例代码，演示如何基于crc64值验证数据传输的完整性。

1. 计算crc64。

```
import oss2
from oss2.models import PartInfo
import os
import crcmod
import random
import string
```

```
do_crc64 = crcmod.mkCrcFun(0x142F0E1EBA9EA3693L, initCrc=0L, xorOut=
0xffffffffffffffffL, rev=True)
def check_crc64(local_crc64, oss_crc64, msg="check crc64"):
if local_crc64 != oss_crc64:
print "{0} check crc64 failed. local:{1}, oss:{2}.".format(msg,
local_crc64, oss_crc64)
return False
else:
print "{0} check crc64 ok.".format(msg)
return True
def random_string(length):
return ''.join(random.choice(string.lowercase) for i in range(length
))
bucket = oss2.Bucket(oss2.Auth(access_key_id, access_key_secret),
endpoint, bucket_name)
```

2. 验证Put Object。

```
content = random_string(1024)
 key = 'normal-key'
 result = bucket.put_object(key, content)
 oss_crc64 = result.headers.get('x-oss-hash-crc64ecma', '')
 local_crc64 = str(do_crc64(content))
 check_crc64(local_crc64, oss_crc64, "put object")
```

3. 验证Get Object。

```
result = bucket.get_object(key)
 oss_crc64 = result.headers.get('x-oss-hash-crc64ecma', '')
 local_crc64 = str(do_crc64(result.resp.read()))
 check_crc64(local_crc64, oss_crc64, "get object")
```

4. 验证Upload Part 和 Complete。

```
part_info_list = []
 key = "multipart-key"
 result = bucket.init_multipart_upload(key)
 upload_id = result.upload_id
 part_1 = random_string(1024 * 1024)
 result = bucket.upload_part(key, upload_id, 1, part_1)
 oss_crc64 = result.headers.get('x-oss-hash-crc64ecma', '')
 local_crc64 = str(do_crc64(part_1))
 #check 上传的 part 1数据是否完整
 check_crc64(local_crc64, oss_crc64, "upload_part object 1")
 part_info_list.append(PartInfo(1, result.etag, len(part_1)))
 part_2 = random_string(1024 * 1024)
 result = bucket.upload_part(key, upload_id, 2, part_2)
 oss_crc64 = result.headers.get('x-oss-hash-crc64ecma', '')
 local_crc64 = str(do_crc64(part_2))
 #check 上传的 part 2数据是否完整
 check_crc64(local_crc64, oss_crc64, "upload_part object 2")
 part_info_list.append(PartInfo(2, result.etag, len(part_2)))
 result = bucket.complete_multipart_upload(key, upload_id,
part_info_list)
 oss_crc64 = result.headers.get('x-oss-hash-crc64ecma', '')
 local_crc64 = str(do_crc64(part_2, do_crc64(part_1)))
 #check 最终oss上的object和本地文件是否一致
```

```
check_crc64(local_crc64, oss_crc64, "complete object")
```

OSS SDK支持

部分OSS SDK已经支持上传、下载使用crc64进行数据校验，用法见下表中的示例：

SDK	是否支持CRC	示例
Java SDK	是	*CRCSample.java*
Python SDK	是	*object_check.py*
PHP SDK	否	无
C# SDK	否	无
C SDK	是	*oss_crc_sample.c*
JavaScript SDK	否	无
Go SDK	是	*crc_test.go*
Ruby SDK	否	无
iOS SDK	否	无
Android SDK	否	无

8.2 通过客户端加密保护数据

客户端加密是指用户数据在发送给远端服务器之前就完成加密，而加密所用的密钥的明文只保留在本地，从而可以保证用户数据安全，即使数据泄漏别人也无法解密得到原始数据。

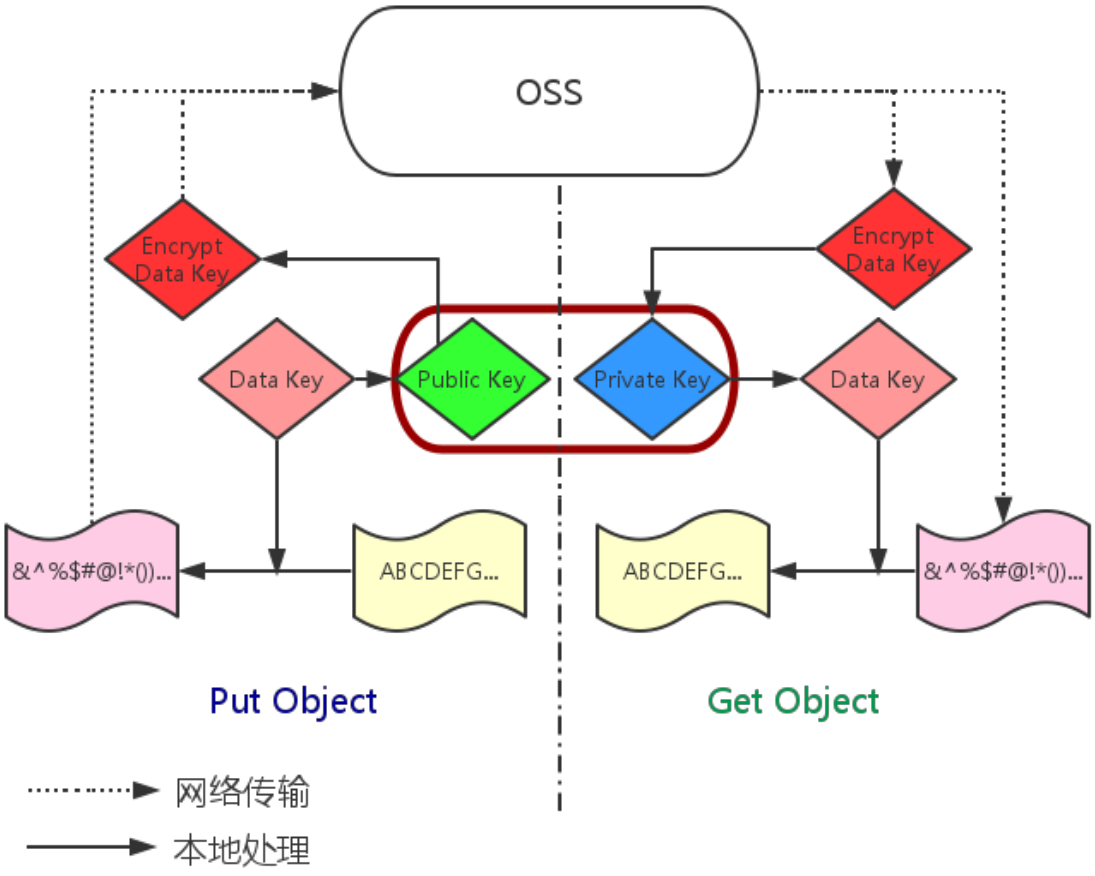
本文介绍如何基于OSS的现有Python SDK版本，通过客户端加密来保护数据。

原理介绍

1. 用户本地维护一对RSA密钥(`rsa_private_key`和`rsa_public_key`)。
2. 每次上传Object时，随机生成一个AES256类型的对称密钥`data_key`，然后用`data_key`加密原始content得到encrypt_content。
3. 用`rsa_public_key`加密`data_key`，得到`encrypt_data_key`，作为用户的自定义meta放入请求头部，和encrypt_content一起发送到OSS。
4. Get Object时，首先得到encrypt_content以及用户自定义meta中的`encrypt_data_key`。
5. 用户使用`rsa_private_key`解密`encrypt_data_key`得到`data_key`，然后用`data_key`解密encrypt_content得到原始content。

> 说明：
> 本文用户的密钥为非对称的RSA密钥，加密Object content时用的AES256-CTR算法，详情可参考 *PyCrypto Document*。本文旨在介绍如何通过Object的自定义meta来实现客户端加密，加密密匙类型及加密算法，用户可以根据自己的需要进行选择。

架构图



准备工作

1. Python SDK的安装和使用，参考 *Python SDK 快速安装*。
2. 安装PyCrypto库。

```
pip install pycrypto
```

完整 **Python** 示例代码

```
# -*- coding: utf-8 -*-
import os
import shutil
```

```
import base64
import random
import oss2
from Crypto.Cipher import PKCS1_OAEP
from Crypto.PublicKey import RSA
from Crypto.Cipher import AES
from Crypto import Random
from Crypto.Util import Counter
# aes 256, key always is 32 bytes
_AES_256_KEY_SIZE = 32
_AES_CTR_COUNTER_BITS_LEN = 8 * 16
class AESCipher:
    def __init__(self, key=None, start=None):
        self.key = key
        self.start = start
        if not self.key:
            self.key = Random.new().read(_AES_256_KEY_SIZE)
        if not self.start:
            self.start = random.randint(1, 10)
        ctr = Counter.new(_AES_CTR_COUNTER_BITS_LEN, initial_value=
self.start)
        self.cipher = AES.new(self.key, AES.MODE_CTR, counter=ctr)
    def encrypt(self, raw):
        return self.cipher.encrypt(raw)
    def decrypt(self, enc):
        return self.cipher.decrypt(enc)
# 首先初始化AccessKeyId、AccessKeySecret、Endpoint等信息。
# 通过环境变量获取，或者把诸如“<您的AccessKeyId>”替换成真实的AccessKeyId等。
#
# 以杭州区域为例，Endpoint可以是：
#   http://oss-cn-hangzhou.aliyuncs.com
#   https://oss-cn-hangzhou.aliyuncs.com
# 分别以HTTP、HTTPS协议访问。
access_key_id = os.getenv('OSS_TEST_ACCESS_KEY_ID', '<您的AccessKeyId
>')
access_key_secret = os.getenv('OSS_TEST_ACCESS_KEY_SECRET', '<您的
AccessKeySecret>')
bucket_name = os.getenv('OSS_TEST_BUCKET', '<您的Bucket>')
endpoint = os.getenv('OSS_TEST_ENDPOINT', '<您的访问域名>')
# 确认上面的参数都填写正确了
for param in (access_key_id, access_key_secret, bucket_name, endpoint
):
    assert '<' not in param, '请设置参数：' + param
##### 0 prepare ########
# 0.1 生成rsa key文件并保存到disk
rsa_private_key_obj = RSA.generate(2048)
rsa_public_key_obj = rsa_private_key_obj.publickey()
encrypt_obj = PKCS1_OAEP.new(rsa_public_key_obj)
decrypt_obj = PKCS1_OAEP.new(rsa_private_key_obj)
# save to local disk
file_out = open("private_key.pem", "w")
file_out.write(rsa_private_key_obj.exportKey())
file_out.close()
file_out = open("public_key.pem", "w")
file_out.write(rsa_public_key_obj.exportKey())
file_out.close()
# 0.2 创建Bucket对象，所有Object相关的接口都可以通过Bucket对象来进行
bucket = oss2.Bucket(oss2.Auth(access_key_id, access_key_secret),
endpoint, bucket_name)
obj_name = 'test-sig-1'
```

```
content = "test content"
#### 1 Put Object  ####
# 1.1 生成加密这个object所用的一次性的对称密钥 encrypt_cipher, 其中的key 和
start为随机生成的value
encrypt_cipher = AESCipher()
# 1.2 将辅助解密的信息用公钥加密后存到object的自定义meta中. 后续当我们get
object时，就可以根据自定义meta，用私钥解密得到原始content
headers = {}
headers['x-oss-meta-x-oss-key'] = base64.b64encode(encrypt_obj.encrypt
(encrypt_cipher.key))
headers['x-oss-meta-x-oss-start'] = base64.b64encode(encrypt_obj.
encrypt(str(encrypt_cipher.start)))
# 1.3. 用 encrypt_cipher 对原始content加密得到encrypt_content
encryt_content = encrypt_cipher.encrypt(content)
# 1.4 上传object
result = bucket.put_object(obj_name, encryt_content, headers)
if result.status / 100 != 2:
    exit(1)
#### 2 Get Object ####
# 2.1 下载得到加密后的object
result = bucket.get_object(obj_name)
if result.status / 100 != 2:
    exit(1)
resp = result.resp
download_encrypt_content = resp.read()
# 2.2 从自定义meta中解析出之前加密这个object所用的key 和 start
download_encrypt_key = base64.b64decode(resp.headers.get('x-oss-meta-x
-oss-key', ''))
key = decrypt_obj.decrypt(download_encrypt_key)
download_encrypt_start = base64.b64decode(resp.headers.get('x-oss-meta
-x-oss-start', ''))
start = int(decrypt_obj.decrypt(download_encrypt_start))
# 2.3 生成解密用的cipher, 并解密得到原始content
decrypt_cipher = AESCipher(key, start)
download_content = decrypt_cipher.decrypt(download_encrypt_content)
if download_content != content:
    print "Error!"
else:
    print "Decrypt ok. Content is: %s" % download_content
```

9 OSS资源的监控与报警

云监控服务能够监控OSS服务资源。借助云监控服务，您可以全面了解您在阿里云上的资源使用情况、性能和运行状况。借助报警服务，您可以及时做出反应，保证应用程序顺畅运行。本章介绍如何监控OSS资源、设置报警规则以及自定义监控大盘。

前提条件

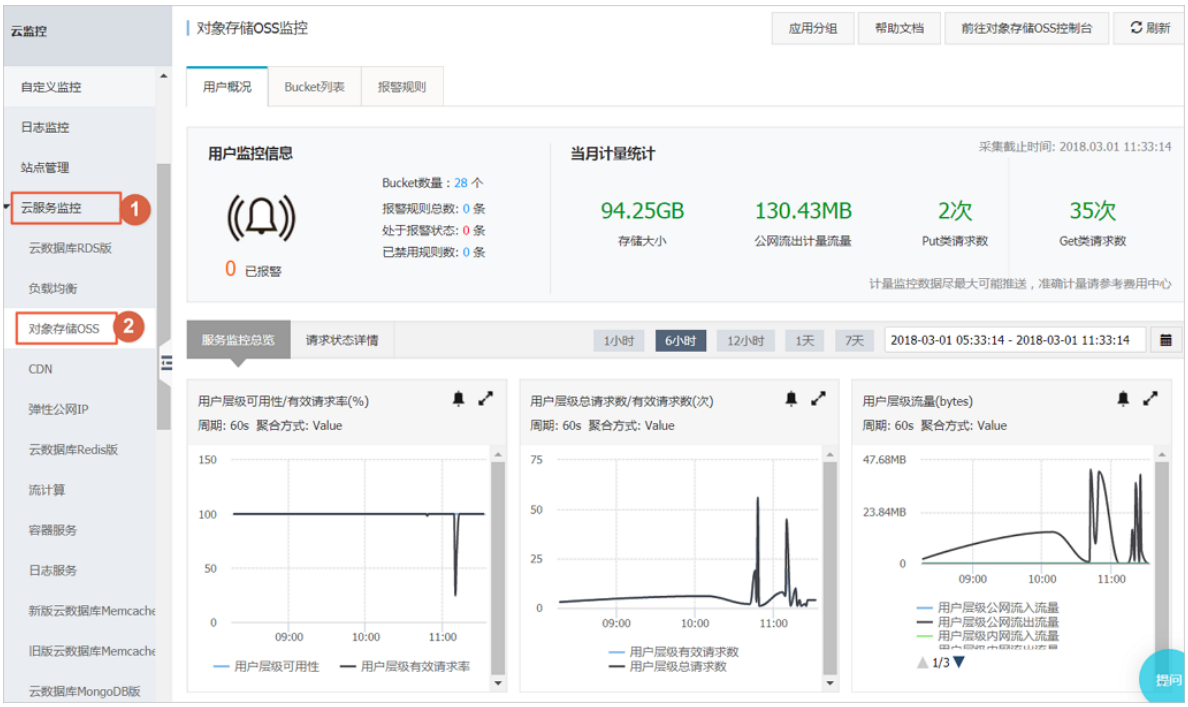
- 已开通OSS服务。
- 已开通云监控服务。

监控OSS资源

1. 登录云监控控制台。
2. 在左侧导航栏选择 **云服务监控 > 对象存储OSS**，进入OSS监控页面，如下图所示。

   在OSS监控页面，您可以获取各类监控数据。

> **说明：**
> 用户层级指用户级别的数据，即该用户下所有bucket的数据。



设置报警规则

1. 在OSS监控页面的报警规则页签下，单击创建报警规则。

2. 填写配置项。

   配置项说明参见管理报警规则。

3. 完成配置后，即生成一条报警规则，您可以使用测试数据来检测该条规则是否生效，确认能否顺利接收报警信息（邮件、短信、旺旺、钉钉等）。

自定义监控大盘

您可以参考如下步骤，在云监控控制台上自定义配置OSS资源监控图。

1. 登录云监控控制台。
2. 在左侧导航栏单击**Dashboard**。
3. 单击创建监控大盘。

4. 输入大盘名称后，单击添加图表。

5. 根据需求完成配置，并单击**发布**。

   配置项说明参见监控指标参考手册。

10 OSS性能与扩展性最佳实践

分区与命名约定

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，如果客户在上传大量对象时，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，当用户的请求速率超过2000操作/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。

从而降低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，根本上，要消除文件名中的顺序前缀。我们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案例。

- 示例 1：向文件名添加十六进制哈希前缀

  如下例所示，用户使用了日期与客户ID生成文件名，包含了顺序时间戳前缀：

```
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，我们可以对客户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，结果如下所示：

```
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
...
```

  加入4个字符组成的十六进制哈希作为前缀，每个字符有0-f共16种取值，因此4个字符共有16^4=65536种可能的字符组合。那么在存储系统中，这些数据理论上会被持续划分至最多65536个

分区，以每个分区2000操作/秒的性能瓶颈标准，再结合您的业务的请求速率，以此您可以评估hash桶的个数是否合适。

如果您想要列出文件名中带有特定日期的文件，例如列出sample-bucket-01里带有2017-11-11的文件，您只要对sample-bucket-01进行列举(即通过多次调用List Object接口，分批次地获得sample-bucket-01下的所有文件)，然后合并带有该日期的文件即可。

- 示例 2：反转文件名

  如下例所示，用户使用了毫秒精度的UNIX时间戳生成文件名，同样属于顺序前缀：

```
sample-bucket-02/1513160001245.log
sample-bucket-02/1513160001722.log
sample-bucket-02/1513160001836.log
sample-bucket-02/1513160001956.log
...
sample-bucket-02/1513160002153.log
sample-bucket-02/1513160002556.log
sample-bucket-02/1513160002859.log
...
```

  由前面的分析，我们知道，这种顺序前缀命名，在请求速率超过一定阈值时，会引发性能问题。我们可以通过反转时间戳前缀来避免，这样文件名就不包含顺序前缀了。反转后结果如下：

```
sample-bucket-02/5421000613151.log
sample-bucket-02/2271000613151.log
sample-bucket-02/6381000613151.log
sample-bucket-02/6591000613151.log
...
sample-bucket-02/3512000613151.log
sample-bucket-02/6552000613151.log
sample-bucket-02/9582000613151.log
...
```

  由于文件名中的前3位数字代表毫秒时间，会有1000种取值。而第4位数字，每1秒钟就会改变一次。同理第5位数字每10秒钟就会改变一次…以此类推，因此，反转文件名后，极大地增强了前缀的随机性，从而将负载压力均匀地分摊在各个分区上，避免出现性能瓶颈。

11 安卓应用示例

11.1 OssDemo简介

在 OSS 的开发人员指南中介绍了移动端开发上传场景。本文主要是基于这个场景来介绍在Android上如何使用SDK来实现一些常见的操作，也就是OssDemo这个工程。主要是以下几个方面：

- 如何使用已经搭建好的应用服务器（STS）
- 如何使用SDK来上传文件
- 如何使用图片服务

这里假设您对OSS的移动开发场景有所了解，并且知道STS（Security Token Service）。

准备工作

由于是基于Android的开发，所以需要做一些准备工作。

1. 开通OSS。请参考 快速入门。
2. 搭建应用服务器。请参考 快速搭建移动应用直传服务。
3. 准备Android开发环境。这里主要用的是Android Studio，网上有很多教程，这里就不再重复。
4. 下载OssDemo的源码。*OssDemo*源码下载地址。可以安装后体验，后面源码分析有详细介绍上面提到的常见操作的实现。
5. 打开OSS官方提供的 *OSS Android SDK*文档 以供参考。

11.2 使用已经搭建好的应用服务器

本文主要讲解OssDemo这样的移动APP如何使用应用服务器，以达到不需要在APP端存储AccessKeyId和AccessKeySecret也能向OSS上传的目的。

调用逻辑

1. OssDemo在获取sts_server的地址后，发送请求。
2. sts_server返回AccessKeyId、AccessKeySecret、SecurityToken、Expiration。
3. OssDemo获取这些信息后，调用SDK，构建OssClient。

具体代码

1. 生成一个EditText控件。

```
位置：
 res/layout/content_main.xml
 内容：
 <EditText
     android:layout_height="wrap_content"
     android:layout_width="0dp"
     android:layout_weight="4"
     android:id="@+id/sts_server"
     android:text="@string/sts_server"
     />
 位置：
 res/values/strings
 内容：
 <string name="sts_server">http://oss-demo.aliyuncs.com/app-server/
sts.php</string>
```

2. 从应用服务器获取STS相关参数的代码。

   函数实现：

```
OSSFederationToken getFederationToken()
```

3. 调用STS返回参数，初始化OssClient代码。

   函数实现：

```
//初始化一个OssService用来上传下载
 public OssService initOSS(String endpoint, String bucket,
ImageDisplayer displayer) {
     //如果希望直接使用accessKey来访问的时候，可以直接使用OSSPlainTe
xtAKSKCredentialProvider来鉴权。
     //OSSCredentialProvider credentialProvider = new OSSPlainTe
xtAKSKCredentialProvider(accessKeyId, accessKeySecret);
     //使用自己的获取STSToken的类
     OSSCredentialProvider credentialProvider = new STSGetter(
stsServer);
     ClientConfiguration conf = new ClientConfiguration();
     conf.setConnectionTimeout(15 * 1000); // 连接超时，默认15秒
     conf.setSocketTimeout(15 * 1000); // socket超时，默认15秒
     conf.setMaxConcurrentRequest(5); // 最大并发请求书，默认5个
     conf.setMaxErrorRetry(2); // 失败后最大重试次数，默认2次
     OSS oss = new OSSClient(getApplicationContext(), endpoint,
credentialProvider, conf);
     return new OssService(oss, bucket, displayer);
```

```
 }
```

11.3 上传文件

简单上传

简单上传就是调用OSS API中的Put Object接口，一次性将选择的文件上传到OSS上。

调用逻辑

1. 选择上传后，可以选择需要上传的本地文件。
2. 选择后OssDemo在获取sts_server的地址后，发送请求。
3. sts_server返回AccessKeyId、AccessKeySecret、SecurityToken、Expiration。
4. OssDemo获取这些信息后，调用SDK，构建OssClient，进行简单上传。

具体代码

1. 生成一个Button控件。

```
位置：
 res/layout/content_main.xml
 内容：
 <Button
     style="?android:attr/buttonStyleSmall"
     android:layout_height="wrap_content"
     android:layout_width="wrap_content"
     android:text="@string/upload"
     android:id="@+id/upload" />
```

2. 点击上传，选择要上传的文件。

   函数实现片段：

```
Button upload = (Button) findViewById(R.id.upload);
 upload.setOnClickListener(new View.OnClickListener() {
     @Override
     public void onClick(View v) {
         Intent i = new Intent(
                 Intent.ACTION_PICK,
                 android.provider.MediaStore.Images.Media.EXTERNAL_C
ONTENT_URI);
         startActivityForResult(i, RESULT_UPLOAD_IMAGE);
     }
 }
```

3. 调用SDK的上传接口。

   函数实现片段：

```
@Override
 protected void onActivityResult(int requestCode, int resultCode,
Intent data) {
```

```
     super.onActivityResult(requestCode, resultCode, data);
     if ((requestCode == RESULT_UPLOAD_IMAGE || requestCode ==
RESULT_PAUSEABLEUPLOAD_IMAGE) && resultCode == RESULT_OK && null !=
data) {
         Uri selectedImage = data.getData();
         String[] filePathColumn = { MediaStore.Images.Media.DATA };
         Cursor cursor = getContentResolver().query(selectedImage,
                 filePathColumn, null, null, null);
         cursor.moveToFirst();
         int columnIndex = cursor.getColumnIndex(filePathColumn[0]);
         String picturePath = cursor.getString(columnIndex);
         Log.d("PickPicture", picturePath);
         cursor.close();
         try {
             Bitmap bm = ImageDisplayer.autoResizeFromLocalFile(
picturePath);
             displayImage(bm);
             File file = new File(picturePath);
             displayInfo("文件: " + picturePath + "\n大小: " + String
.valueOf(file.length()));
         }
         catch (IOException e) {
             e.printStackTrace();
             displayInfo(e.toString());
         }
         //根据操作不同完成普通上传或者断点上传
         if (requestCode == RESULT_UPLOAD_IMAGE) {
             final EditText editText = (EditText) findViewById(R.id.
edit_text);
             String objectName = editText.getText().toString();
             //调用简单上传接口上传
             ossService.asyncPutImage(objectName, picturePath,
getPutCallback(), new ProgressCallbackFactory<PutObjectRequest>().
get());
         }
     }
 }
```

这里省略了对上传结果的处理，可以参考源码中的onSuccess和onFailure的处理。

基于分片上传实现的断点续传上传

调用OSS API中的Multipart Upload（分片上传）接口实现断点续传的效果。

调用逻辑

1. 选择上传后，可以选择需要上传的本地文件。
2. 选择后OssDemo在获取sts_server的地址后，发送请求。
3. sts_server返回AccessKeyId、AccessKeySecret、SecurityToken、Expiration。
4. OssDemo获取这些信息后，调用SDK，构建OssClient，进行分片上传。
5. 点击**暂停**时，如果分片上传没有结束，再点击**继续**时，可以接着未完成的地方继续上传。以达到断点续传的效果。

具体代码

1. 生成一个Button控件。

```
位置：
 res/layout/content_main.xml
 内容：
 <Button
     style="?android:attr/buttonStyleSmall"
     android:layout_height="wrap_content"
     android:layout_width="wrap_content"
     android:text="@string/multipart_upload"
     android:id="@+id/multipart_upload" />
```

2. 点击上传，选择要上传的文件。

   函数实现片段：

```
Button multipart_upload = (Button) findViewById(R.id.multipart_
upload);
 multipart_upload.setOnClickListener(new View.OnClickListener() {
     @Override
     public void onClick(View v) {
         //为了简单化，这里只会同时运行一个断点上传的任务
         Intent i = new Intent(
                 Intent.ACTION_PICK,
                 android.provider.MediaStore.Images.Media.EXTERNAL_C
ONTENT_URI);
         startActivityForResult(i, RESULT_PAUSEABLEUPLOAD_IMAGE);
     }
 }
 );
```

3. 点击上传，继续时的断点续传。

   函数实现片段：

```
点击上传： //这里调用SDK的分片上传接口来上传 task = ossService.asyncMulti
PartUpload(objectName, picturePath, getMultiPartCallback().
addCallback(new Runnable() {     @Override     public void run()
 {         pauseTaskStatus = TASK_NONE;         multipart_resume.
setEnabled(false);         multipart_pause.setEnabled(false);
     task = null;     } }}, new ProgressCallbackFactory<PauseableU
ploadRequest>().get()); 底层对SDK的封装逻辑，可以看到是在multiPartU
ploadManager中的asyncUpload实现的断点续传上传 //断点上传，返回的task可以
用来暂停任务 public PauseableUploadTask asyncMultiPartUpload(String
 object,                                                    String
localFile,                                                   @NonNull
 final OSSCompletedCallback<PauseableUploadRequest, PauseableU
ploadResult> userCallback,
         final OSSProgressCallback<PauseableUploadRequest> userProgre
ssCallback) {     if (object.equals("")) {         Log.w("AsyncMulti
PartUpload", "ObjectNull");         return null;     }     File file
 = new File(localFile);     if (!file.exists()) {         Log.w("
AsyncMultiPartUpload", "FileNotExist");         Log.w("LocalFile",
localFile);         return null;     }     Log.d("MultiPartUpload",
localFile);     PauseableUploadTask task = multiPartUploadManager.
```

```
asyncUpload(object, localFile, userCallback, userProgressCallback);
    return task; }
```

11.4 图片处理

在OssDemo中展示了上传一张图片后，各种不同的处理。和下载不同的地方是：

- 使用的是图片处理的Endpoint。
- 在Object后面带了一些处理参数。

图片加水印

调用逻辑

1. 上传一张图片到OSS，在默认的情况下bucket是sdk-demo，object是test，OSS的Endpoint是`oss-cn-hangzhou.aliyuncs.com`。
2. 根据不同的图片处理方式，在test后面加不同的处理参数，以展示不同的显示效果。
3. 选择后OssDemo在获取sts_server的地址后，发送请求。
4. sts_server返回AccessKeyId、AccessKeySecret、SecurityToken、Expiration。
5. OssDemo获取这些信息后，调用SDK，构建OssClient，进行下载操作。呈现的效果就是图片处理的效果。不过图片服务的Endpoint是`img-cn-hangzhou.aliyuncs.com`。

具体代码

1. 点击更多后，到了图片服务处理图片后的页面。
2. 将之前上传的图片，在右下角加水印，并且大小为100，获取这样的操作命令。

   函数实现片段：

```
在ImageService类中
 提供了这样的一个方法，主要是在原来的object后增加相应的功能需要的参数
 //给图片打上文字水印，除了大小字体之外其他都是默认值，有需要更改的可以参考图片服务文档自行调整
 public String textWatermark(String object, String text, int size) {
     String base64Text = Base64.encodeToString(text.getBytes(), Base64.URL_SAFE | Base64.NO_WRAP);
     String queryString = "@watermark=2&type=" + font + "&text=" + base64Text + "&size=" + String.valueOf(size);
     Log.d("TextWatermark", object);
     Log.d("Text", text);
     Log.d("QuerySyring", queryString);
     return (object + queryString);
 }
```

3. 调用SDK的下载接口，进行图片处理。

函数实现片段：

```
getImage(imageService.textWatermark(objectName, "OSS测试", 100), 0,
 "右下角文字水印，大小100");
 public void getImage(final String object, final Integer index,
final String method) {
     GetObjectRequest get = new GetObjectRequest(bucket, object);
     Log.d("Object", object);
     OSSAsyncTask task = oss.asyncGetObejct(get, new UICallback<
GetObjectRequest, GetObjectResult>(uiDispatcher) {
         @Override
         public void onSuccess(GetObjectRequest request, GetObjectR
esult result) {
             // 请求成功
             InputStream inputStream = result.getObjectContent();
             Log.d("GetImage", object);
             Log.d("Index", String.valueOf(index));
             try {
                 //防止超过显示的最大限制
                 adapter.getImgMap().put(index, new ImageDisplayer(
1000, 1000).autoResizeFromStream(inputStream));
                 adapter.getTextMap().put(index, method + "\n" +
object);
                 //需要根据对应的View大小来自适应缩放
                 addCallback(new Runnable() {
                     @Override
                     public void run() {
                         adapter.notifyDataSetChanged();
                     }
                 }, null);
             }
             catch (IOException e) {
                 e.printStackTrace();
             }
             super.onSuccess(request,result);
         }
```

这里省略了对下载结果失败的处理，可以参考源码中的onFailure的处理。

图片缩放、裁剪、旋转

和加水印的过程类似，在ImageService中增加获取处理命令的函数，以”object + 处理参数“的形式返回，最后调用SDK的Get Object接口来处理。

```
//缩放
getImage(imageService.resize(objectName, 100, 100), 1, "缩放到100*100
");
//裁剪
getImage(imageService.crop(objectName, 100, 100, 9), 2, "右下角裁剪100*
100");
//旋转
getImage(imageService.rotate(objectName, 90), 3, "旋转90度");
```