# 阿里云
# 对象存储

## 最佳实践

文档版本：20190906

# 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:**<br>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:**<br>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:**<br>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。 |
| *##* | 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 数据迁移..... 1
1.1 从AWS S3上的应用无缝切换至OSS.....1
1.2 使用ossimport迁移数据..... 4
1.3 Hadoop文件系统HDFS数据迁移到OSS..... 6
2 Web端上传数据至OSS..... 10
2.1 Web端PostObject直传实践..... 10
2.1.1 Web端直传实践简介..... 10
2.1.2 JavaScript客户端签名直传..... 11
2.1.3 服务端签名后直传.....14
2.1.4 服务端签名直传并设置上传回调.....15
3 移动应用端直传实践.....20
3.1 快速搭建移动应用直传服务..... 20
3.2 权限控制.....28
3.3 快速搭建移动应用上传回调服务..... 33
4 数据处理与分析.....39
4.1 基于OSS+MaxCompute构建数据仓库..... 39
4.2 EMR+OSS：离线计算的存储与计算分离..... 42
5 数据备份和容灾.....46
5.1 备份存储空间.....46
6 存储空间管理.....48
6.1 跨域资源共享（CORS）.....48
6.2 防盗链.....63
6.3 静态网站托管.....71
7 数据安全.....76
7.1 通过crc64校验数据传输的完整性.....76
7.2 通过客户端加密保护数据..... 78
7.3 通过服务器端加密保护数据..... 81
8 OSS资源的监控与报警..... 85
9 OSS性能与扩展性最佳实践..... 88
10 安卓应用示例..... 90
10.1 OssDemo简介..... 90
10.2 使用已经搭建好的应用服务器..... 90
10.3 上传文件..... 91
10.4 图片处理..... 94
11 Terraform..... 97

11.1 Terraform简介..... 97
11.2 使用Terraform管理OSS.....98

# 1 数据迁移

## 1.1 从AWS S3上的应用无缝切换至OSS

OSS提供了S3 API的兼容性，可以让您的数据从AWS S3无缝迁移到阿里云OSS上。

从AWS 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。

### 迁移教程

您可以使用阿里云在线迁移服务将AWS S3 数据轻松迁移至阿里云对象存储 OSS。详情请参见 AWS S3 迁移教程。

### 迁移后通过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。

- ACL权限定义

  OSS对ACL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。

| 级别 | AWS S3权限 | AWS 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。 |

| 级别 | AWS S3权限 | AWS S3 | 阿里云OSS |
| --- | --- | --- | --- |
| | WRITE | Bucket下的Object可写入或覆盖 | - 对于Bucket下不存在的Object，可写入。<br>- 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。<br>- 可以初始化分片上传（InitiateMultipartUpload）。 |
| | 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。 |
| | 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。 |

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。

- 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AWSmazon S3中的STANDARD、STANDARD_IA和GLACIER。您可以根据需要转换OSS Object的存储类型。

  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AWS 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

# 1.2 使用ossimport迁移数据

本文介绍如何使用ossimport将数据从第三方存储（或OSS）迁移到OSS。

## 环境配置

ossimport有单机模式和分布式模式两种部署方式：

- 对于小于30TB的小规模数据迁移，单机模式即可完成。
- 对于大规模的数据迁移，请使用分布式模式。

ossimport部署方式详细介绍请参考#unique_9。

假设您需要将迁移源腾讯云COS华南1（深圳）区域的500TB数据，于一周内迁移至OSS华东1（杭州）区域，您需要进行ossimport分布式环境配置：

- 开通OSS
  1. 使用您的账号创建华东1（杭州）区域的OSS Bucket。
  2. 在RAM控制台创建子帐号，并授权该子账号访问OSS的权限，并保存AccessKeyID和AccessKeySecret。
- 购买ECS

  购买OSS同区域华东1（杭州）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。

  > 说明:
  > 若分布式部署所需的计算机数量较少时，您可以直接在本地部署；若所需计算机数量较多时，建议在ECS实例上部署。

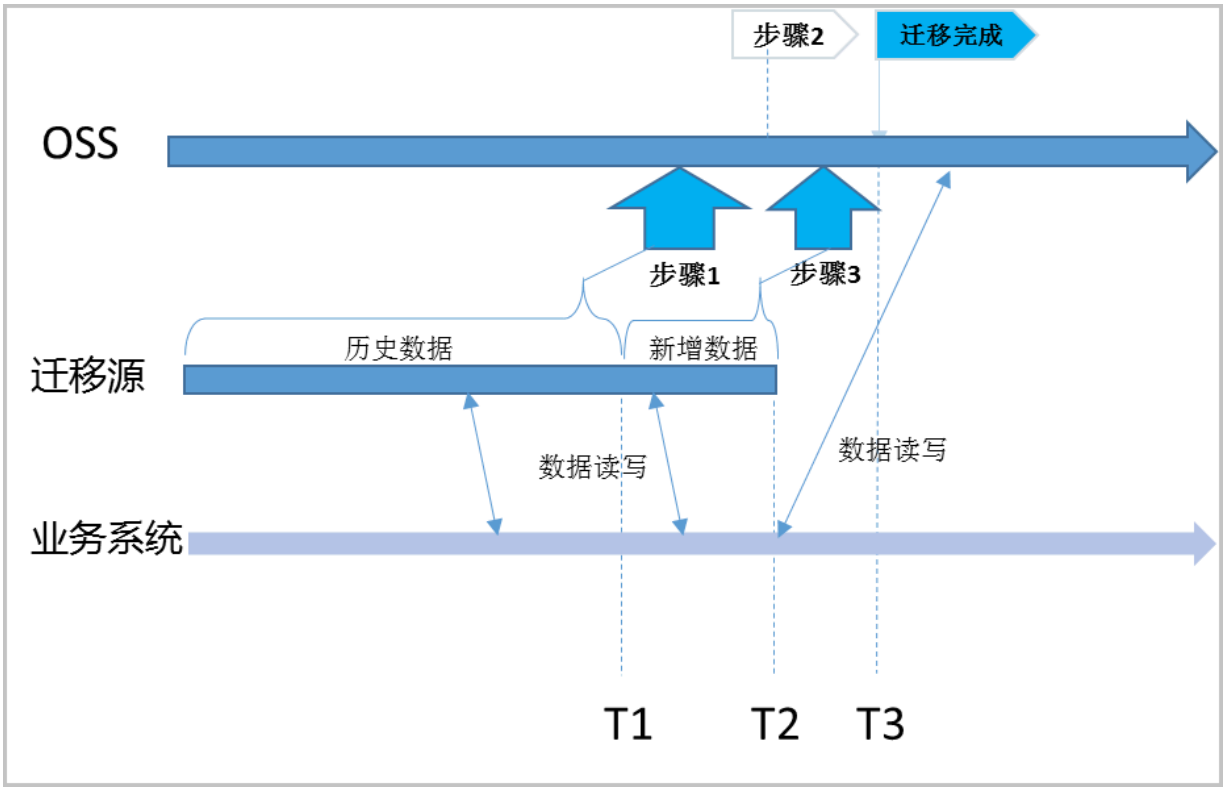
  ECS所需数量的计算公式为：X/Y/(Z/100)台。其中X为需要迁移的数据量、Y为要求迁移完成的时间（天）、Z为单台ECS迁移速度Z Mbps（每天迁移约Z/100 TB数据）。假设单台ECS迁移速度达到200Mbps（即每天约迁移2TB数据），则上述示例中需购买ECS 36台（即500/7/2）。
- 配置ossimport

  结合本示例中的大规模迁移需求，您需要在ECS上搭建ossimport分布式模式。有关分布式部署的配置定义信息，如`conf/job.cfg`、`conf/sys.properties`、并发控制等配置，请参考说明及配置。有关分布式部署的相关操作，如ossimport下载、配置过程的常见错误及排除，请参考分布式部署。

### 迁移步骤

使用分布式模式将第三方存储迁移至OSS的过程如下：

> **说明:**
> 在ECS上搭建ossimport分布式环境后，ossimport从腾讯云COS华南1（深圳）区域下载数据到ECS华东1（杭州），建议使用外网。使用ossimport从ECS华东1（杭州）将数据上传到OSS华东1（杭州），建议使用内网。




1. 全量迁移第三方存储T1前的历史数据，详细步骤请参考分布式部署的运行。

   > **注意:**
   > T1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。

2. 在OSS控制台打开OSS镜像回源，回源地址设置为迁移源（第三方存储）。
3. 将业务系统读写切换至OSS，此时业务系统记录的时间为T2。

   T1前的数据从OSS读取，T1后的数据则通过OSS镜像回源从第三方存储读取，新数据完全写入OSS。

   T2后不再有新数据写入迁移源。

4. 修改配置文件`job.cfg`的配置项`importSince=T1`，重新发起迁移任务，进行T1~T2的增量数据迁移。

> 说明:
>
> - 步骤4完成后，您业务系统的所有的读写都在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只负责数据的迁移和校验，不会删除任何数据。

费用说明

迁移过程涉及到的成本包含：源和目的存储空间访问费用、源存储空间的流出流量费用、ECS实例费用、数据存储费用、时间成本。如果数据超过TB级别，存储成本和迁移时间成正比，且相对流量、存储费用，ECS费用较小，增加ECS数量，会减少迁移时间。

参考文档

有关ossimport的相关说明，请参考以下文档：

分布式部署

说明及配置

常见问题

## 1.3 Hadoop文件系统HDFS数据迁移到OSS

本文介绍如何快速地将 Hadoop 文件系统（HDFS）上的数据迁移到 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ossimport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。

### 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 /
```

### 实施步骤

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。

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。<br>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。

# 2 Web端上传数据至OSS

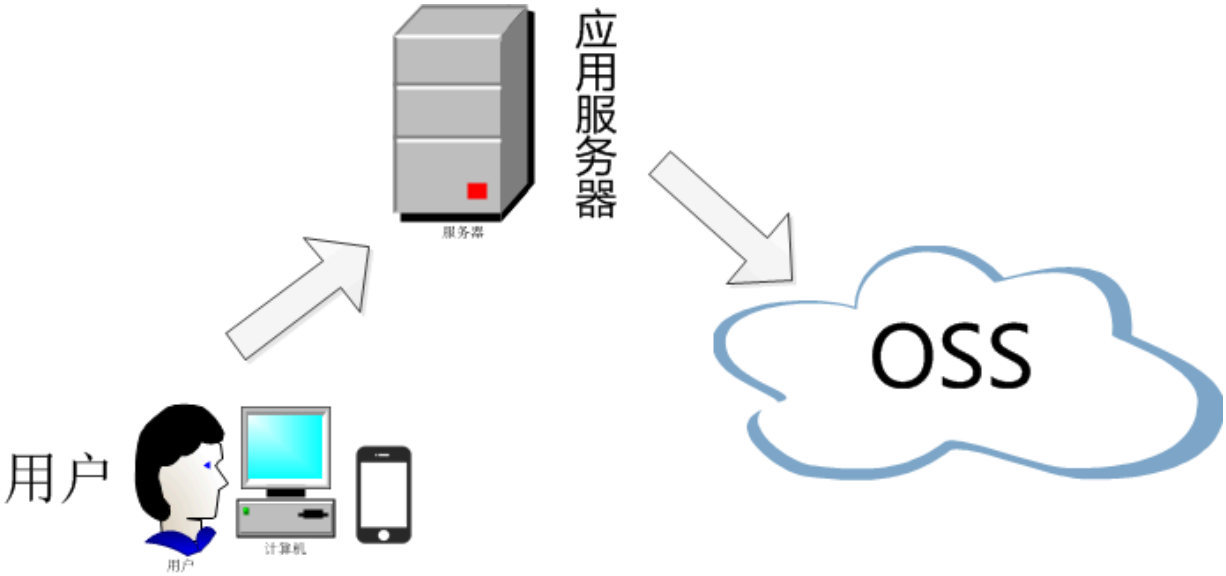
## 2.1 Web端PostObject直传实践

### 2.1.1 Web端直传实践简介

本教程介绍如何在Web端通过表单上传方式直接上传数据到OSS。

背景

每个OSS的用户都会用到上传服务。Web端常见的上传方法是用户在浏览器或app端上传文件到应用服务器，然后应用服务器再把文件上传到OSS，如下图所示：




和数据直传到OSS相比，以上方法有三个缺点：

- 上传慢。先上传到应用服务器，再上传到OSS，网络传送比直传到OSS多了一倍。如果直传到OSS，不通过应用服务器，速度将大大提升，而且OSS采用BGP带宽，能保证各地各运营商的速度。
- 扩展性差。如果后续用户多了，应用服务器会成为瓶颈。
- 费用高。需要准备多台应用服务器。由于OSS上传流量是免费的，如果数据直传到OSS，不通过应用服务器，那么将能省下几台应用服务器。

目的

本教程的目的是通过以下三个例子介绍如何通过表单直传数据到OSS：

- JavaScript客户端签名直传讲解在客户端通过JavaScript代码完成签名，然后通过表单直传数据到OSS。

- 服务端签名后直传讲解在服务端完成签名，然后通过表单直传数据到OSS。
- 服务端签名直传并设置上传回调讲解在服务端完成签名，并且服务端设置了上传后回调，然后通过表单直传数据到OSS。OSS回调完成后，再将应用服务器响应结果返回给客户端。

## 2.1.2 JavaScript客户端签名直传

本文主要介绍如何基于POST Policy的使用规则在客户端通过JavaScript代码完成签名，然后通过表单直传数据到OSS。

### Demo

PC浏览器测试样例

本示例采用Plupload 直接提交表单数据（即PostObject）到OSS，可以运行在PC浏览器、手机浏览器、微信等。您可以同时选择多个文件上传，并设置上传到指定目录和设置上传文件名称是随机文件名还是本地文件名。您还可以通过进度条查看上传进度。

说明:
文件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。

### 步骤 1：下载应用服务器代码

下载地址

### 步骤 2：修改配置文件

将下载包解压后，修改upload.js文件：

```
accessid= '<yourAccessKeyId>';
accesskey= <yourAccessKeySecrety>';
.....
new_multipart_params = {
        ....
        'OSSAccessKeyId': accessid,
        ....
    };

//如果是STS方式====
accessid = 'STS.ACCESSKEYID';
accesskey = 'STS.ACCESSSECRET';
token = 'STS.token';

.....
new_multipart_params = {
        ....
        'OSSAccessKeyId': accessid,
        'x-oss-security-token':token
        ....
    };
//============
```

```
host = 'http://post-test.oss-cn-hangzhou.aliyuncs.com';
```

- $id：您的AccessKeyId
- $key：您的AcessKeySecret
- $host：格式为BucketName.Endpoint，例如post-test.oss-cn-hangzhou.aliyuncs.com

**说明:**

关于Endpoint的介绍，请参见Endpoint（访问域名）。

步骤 3：设置CORS

客户端进行表单直传到OSS时，会从浏览器向OSS发送带有Origin的请求消息。OSS对带有Origin头的请求消息会进行跨域规则（CORS）的验证。因此需要为Bucket设置跨域规则以支持Post方法。

具体操作步骤请参见设置跨域访问。

**说明:**

来源设置为 * 是为了使用方便，不确保安全性。建议您填写自己需要的域名。

步骤 4：体验JavaScript客户端签名直传

1. 将应用服务器代码zip包解压到Web根目录下。
2. 在Web浏览器中输入<Web应用服务器地址>/oss-h5-upload-js-direct/index.html，例如http://abc.com:8080/oss-h5-upload-js-direct/index.html。
3. 选择一个或多个文件进行上传。
4. 上传成功后，通过控制台查看上传结果。

核心代码解析

因为OSS支持POST协议，所以只要在Plupload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，不设置则默认返
回204
        'signature': signature,
        'x-oss-security-token':token
    },
  ....
}
```

上述代码中，'Filename': '${filename}'表示上传后保持原来的文件名。如果您想上传到特定目录如abc下，且文件名不变，请修改代码如下：

```
multipart_params: {
        'Filename': 'abc/' + '${filename}',
        'key' : '${filename}',
        'policy': policyBase64,
        'OSSAccessKeyId': accessid,
        'success_action_status' : '200', //让服务端返回200，不设置则默认返
回204
        'signature': signature,
    },
```

- 设置成随机文件名

  如果想在上传时固定设置成随机文件名，后缀保持跟客户端文件一致，可以将函数改为：

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

- 设置成用户的文件名

  如果想在上传时固定设置成用户的文件名，可以将函数改为：

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

- 设置上传目录

  您可以将文件上传到指定目录下。下面的代码是将上传目录改成abc/，注意目录必须以正斜线（/）结尾。

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

- 上传签名

  上传签名主要是对policyText进行签名，最简单的例子如下：

  ```
  var policyText = {
      "expiration": "2020-01-01T12:00:00.000Z", // 设置Policy的失效时
  间，如果超过失效时间，就无法通过此Policy上传文件
      "conditions": [
  ```

```
    ["content-length-range", 0, 1048576000] // 设置上传文件的大小限
制，如果超过限制，文件上传到OSS会报错
    ]
}
```

总结

在客户端通过JavaScript代码完成签名，无需过多配置，即可实现直传，非常方便。但是客户端通过JavaScript把AccesssKeyID 和AccessKeySecret写在代码里面有泄露的风险，建议采用服务端签名后直传。

## 2.1.3 服务端签名后直传

本示例介绍如何在服务端完成签名并通过表单直传数据到OSS。

> 说明:
> 本示例无法实现分片上传与断点续传。

背景

采用JavaScript客户端直接签名（参见JavaScript客户端签名直传）时，AccessKeyID和AcessKeySecret会暴露在前端页面，因此存在严重的安全隐患。因此，OSS提供了服务端签名后直传的方案。

原理介绍

服务端签名后直传的原理如下：

1. 用户发送上传Policy请求到应用服务器。
2. 应用服务器返回上传Policy和签名给用户。
3. 用户直接上传数据到OSS。

本示例中，Web端向服务端请求签名，然后直接上传，不会对服务端产生压力，而且安全可靠。但本示例中的服务端无法实时了解用户上传了多少文件，上传了什么文件。如果想实时了解用户上传了什么文件，可以采用服务端签名直传并设置上传回调。

流程和源码解析

服务端签名后直传的源码流程和服务端签名直传并设置上传回调类似，请参考服务端签名直传并设置上传回调–原理介绍 。

## 2.1.4 服务端签名直传并设置上传回调

本文主要介绍如何基于POST Policy的使用规则在服务端通过各种语言代码完成签名，并且设置上传回调，然后通过表单直传数据到OSS。

### 背景

采用服务端签名后直传方案有个问题：大多数情况下，用户上传数据后，应用服务器需要知道用户上传了哪些文件以及文件名；如果上传了图片，还需要知道图片的大小等。为此OSS提供了上传回调方案。OSS回调完成后，应用服务器再将结果返回给客户端，以便服务端实时了解用户上传了什么文件。

### 流程介绍

流程如下：

1. 用户向应用服务器请求上传Policy和回调。
2. 应用服务器返回上传Policy和回调设置。
3. 用户直接向OSS发送文件上传请求。
4. OSS根据用户的回调设置，发送回调请求给应用服务器。
5. 应用服务器返回响应给OSS。
6. OSS将应用服务器返回的内容返回给用户。

当用户要上传一个文件到OSS，而且希望将上传的结果返回给应用服务器，这时就需要设置一个回调函数，将请求告知应用服务器。用户上传完文件后，不会直接得到返回结果，而是先通知应用服务器，再把结果转达给用户。

### 流程解析

服务端签名直传并设置上传回调提供了PHP、Java、Python、Go、Ruby语言版本，流程如下：

1. 用户向应用服务器请求上传Policy和回调。
2. 应用服务器返回上传Policy和回调设置代码。

   应用服务器侧的签名直传服务会处理客户端发过来的GET请求消息，您可以设置对应的代码让应用服务器能够给客户端返回正确的消息。各个语言版本的配置文档中都有明确的说明供您参考。
3. 用户直接向OSS发送文件上传请求。
4. OSS根据用户的回调设置，发送回调请求给应用服务器。

   客户端上传文件到OSS结束后，OSS解析客户端的上传回调设置，发送POST回调请求给应用服务器。

5. 应用服务器返回响应给OSS。

   应用服务器根据OSS发送消息中的`authorization`来进行验证，如果验证通过，则向OSS返回如下json格式的成功消息。

   ```
   String value: OK
   Key: Status
   ```

6. OSS将应用服务器返回的消息返回给用户。

客户端源码解析

客户端源码下载地址：aliyun-oss-appserver-js-master.zip

**说明:**

客户端JavaScript代码使用的是Plupload组件。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。它会智能检测当前环境，选择最适合的上传方式，并且会优先采用html5方式。详情请参见Plupload官网。

下面举例介绍几个关键功能的代码实现：

- 设置成随机文件名

  若上传时采用固定格式的随机文件名，且后缀跟客户端文件名保持一致，可以将函数改为：

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

- 设置成用户的文件名

  如果想在上传时设置成用户的文件名，可以将函数改为：

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

- 设置上传目录

  上传的目录由服务端指定，每个客户端只能上传到指定的目录，实现安全隔离。下面的代码以PHP为例，将上传目录改成abc/，注意目录必须以正斜线（/）结尾。

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

- 设置上传过滤条件

  您可以利用Plupload的属性filters设置上传的过滤条件，如设置只能上传图片、上传文件的大小、不能有重复上传等。

  ```
  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
    },
```

- 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。

· 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，如下所示：

```
FileUploaded: function(up, file, info) {
            if (info.status == 200)
            {
                document.getElementById(file.id).getElement
sByTagName('b')[0].innerHTML = 'upload to oss success, object name:'
 + get_uploaded_object_name(file.name);
            }
            else
            {
                document.getElementById(file.id).getElement
sByTagName('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记录了本地文件上传的名称。

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

· 上传签名

JavaScript可以从服务端获取policyBase64、accessid、signature这三个变量，核心代码如下：

```
function get_signature()
{
    // 判断expire的值是否超过了当前时间，如果超过了当前时间，就重新获取签名，缓
冲时间为3秒。
    now = timestamp = Date.parse(new Date()) / 1000;
    if (expire < now + 3)
    {
        body = send_request()
        var obj = eval ("(" + body + ")");
        host = obj['host']
        policyBase64 = obj['policy']
        accessid = obj['accessid']
        signature = obj['signature']
        expire = parseInt(obj['expire'])
        callbackbody = obj['callback']
        key = obj['dir']
        return true;
    }
```

```
    return false;
};
```

从服务端返回的消息解析如下：

**说明：**

以下仅为示例，并不要求相同的格式，但必须有accessid、policy、signature三个值。

```
{"accessid":"6MKO******4AUk44",
"host":"http://post-test.oss-cn-hangzhou.aliyuncs.com",
"policy":"eyJleHBpcmF0aW9uIjoiMjAxNS0xMS0wNVQyMDoyMzoyM1oiLC
JjxbS5kaXRpb25zIjpbWyJjcb250ZW50LWxlbmd0aC1yYW5nZSIsMCwxMDQ4
NTc2MDAwXSxbInN0YXJ0cy13aXRoIiwiJGtleSIsInVzZXItZGlyXC8iXV19",
"signature":"I2u57FWjTKqX/AE6doIdyff151E=",
"expire":1446726203,"dir":"user-dir/"}
```

- accessid：用户请求的accessid。
- host：用户要往哪个域名发送上传请求。
- policy：用户表单上传的策略（Policy），是经过base64编码过的字符串。详情请参见Post Policy。
- signature：对变量policy签名后的字符串。
- expire：上传策略Policy失效时间，在服务端指定。失效时间之前都可以利用此Policy上传文件，无需每次上传都去服务端获取签名。

**说明：**

为了减少服务端的压力，设计思路是：初始化上传时，每上传一个文件，获取一次签名。再上传时，比较当前时间与签名时间，看签名时间是否失效。如果失效，就重新获取一次签名，如果没有失效，就使用之前的签名。

解析Policy的内容如下：

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

上面Policy中增加了starts-with，用来指定此次上传的文件名必须以user-dir开头，用户也可自行指定。增加starts-with的原因是：在很多场景下，一个应用对应一个Bucket，为了防止数据覆盖，每个用户上传到OSS的文件都可以有特定的前缀。但这样存在一个问题，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。解决方法为，在应用服务器端就指定用户上传文件的前缀。如果用户获取了Policy也没有办法上传到别人的目录，从而保证了数据的安全性。

- 设置应用服务器的地址

  在客户端源码upload.js文件中，如下代码片段的变量serverUrl的值可以用来设置签名服务器的URL，设置好之后，客户端会向该serverUrl发送GET请求来获取需要的信息。

```
// serverUrl是 用户获取签名和Policy等信息的应用服务器的URL，请将下面的IP和Port配置为您自己的真实信息。
serverUrl = 'http://88.88.88.88:8888'
```

# 3 移动应用端直传实践

## 3.1 快速搭建移动应用直传服务

本文主要介绍如何基于STS Policy的使用规则在30分钟内搭建一个移动应用数据直传服务。直传指的是移动应用数据的上传和下载直接连接OSS，只有控制流连接自己的服务器。

### 背景

在移动互联的时代，手机app上传的数据越来越多。作为开发者，您可以利用OSS处理各种数据存储需求，从而更加专注于自己的应用逻辑。

基于OSS的移动应用数据直传服务具有以下优势：

- 数据安全：使用灵活的授权和鉴权方式进行数据的上传和下载，更加安全。
- 成本低廉：您不需要准备很多服务器。移动应用直接连接云存储OSS，只有控制流连接应用服务器。
- 高并发：支持海量用户并发访问。
- 弹性扩容：无限扩容的存储空间。
- 数据处理：和图片处理以及音视频转码搭配使用，方便灵活地进行数据处理。

### 下载并安装移动应用

移动应用源码的下载地址如下：

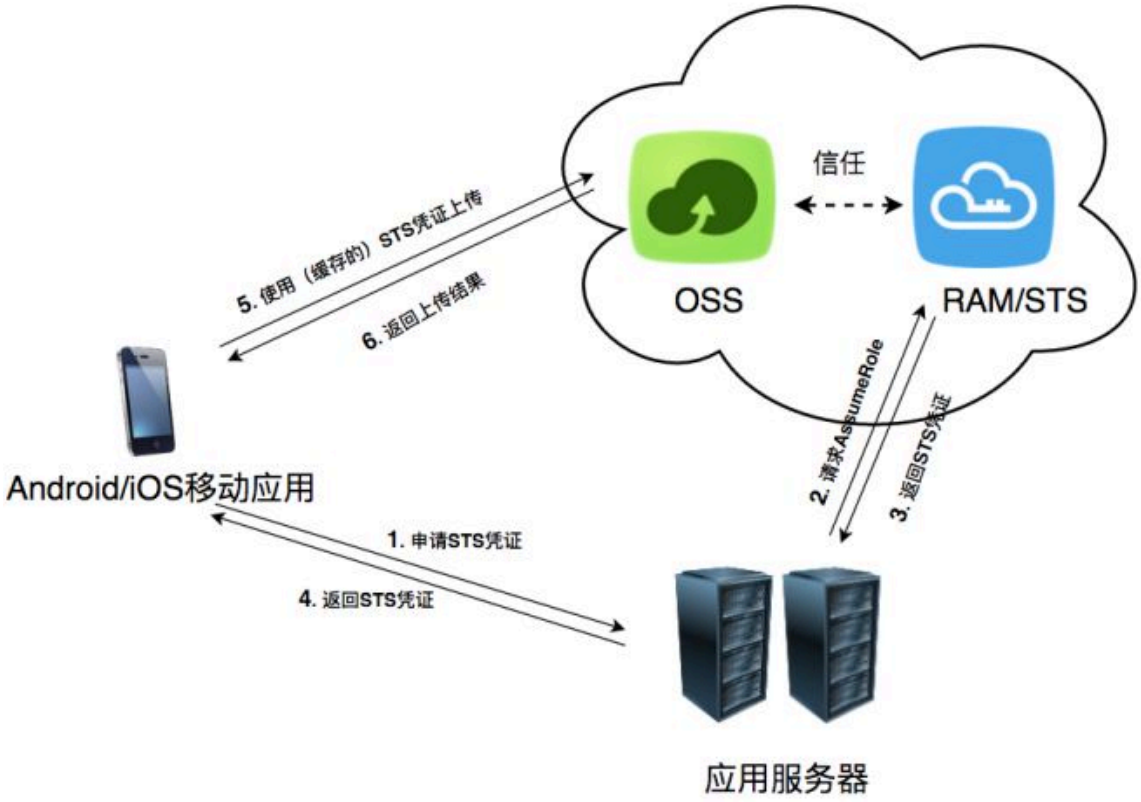
- Android：下载地址
- iOS：下载地址

您可以通过此移动应用上传图片到OSS。上传的方法支持普通上传和断点续传上传。在网络环境差的情况下，推荐使用断点续传上传。您还可以利用图片处理服务，对要上传的图片进行缩略和加水印处理。示例应用的最终效果图如下：

- 应用服务器：该移动应用对应的后台应用服务器。请使用步骤2：下载应用服务器代码示例中提供的应用服务器代码示例，部署自己的应用服务器。
- 上传Bucket：该移动应用要把数据上传到哪个Bucket。
- 区域：上传Bucket所在的地域。

### 原理介绍

移动应用直传服务的开发流程图如下：




角色分析如下：

- Android/iOS 移动应用：即最终用户手机上的app，负责从应用服务器申请及使用STS凭证。
- OSS：即阿里云对象存储，负责处理移动应用的数据请求。
- RAM/STS：负责生成临时上传凭证，即Token。
- 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，用于管理app上传和下载的Token，以及用户通过app上传数据的元信息。

实现步骤如下：

1. 移动应用向应用服务器申请一个临时上传凭证。

> 说明：
> Android和iOS应用不能直接存储AccessKey，这样会存在数据泄露的风险。所以应用必须向用户的应用服务器申请一个Token。这个Token是有时效性的，如果Token的过期时间是30分钟（由应用服务器指定），那么在这30分钟里，该Android/iOS应用可以使用此Token从OSS上传和下载数据， 30分钟后需要重新获取Token。

2. 应用服务器检测上述请求的合法性，然后返回Token给移动应用。
3. Android/iOS移动应用使用此Token将数据上传到OSS，或者从OSS下载数据。

以下介绍应用服务器如何生成Token以及Android/iOS移动应用如何获取Token。

步骤1：创建Bucket并开通STS服务

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。

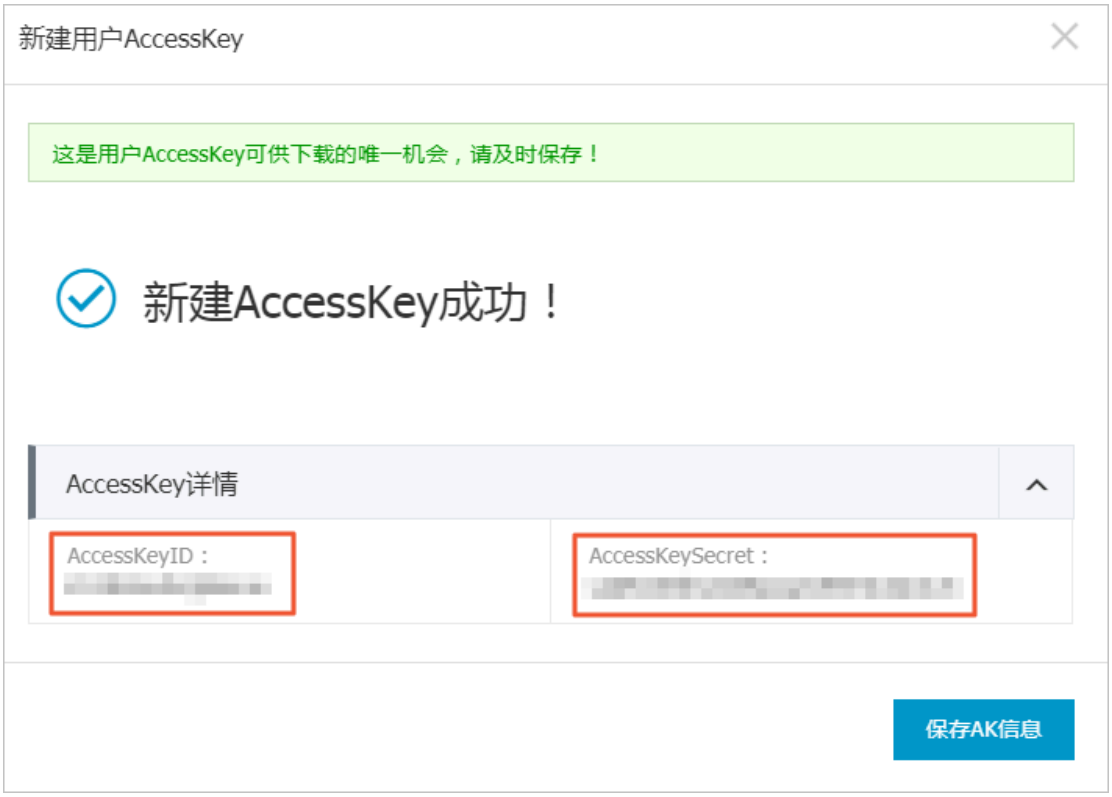
   c. 进入到安全令牌快捷配置页面，单击开始授权。

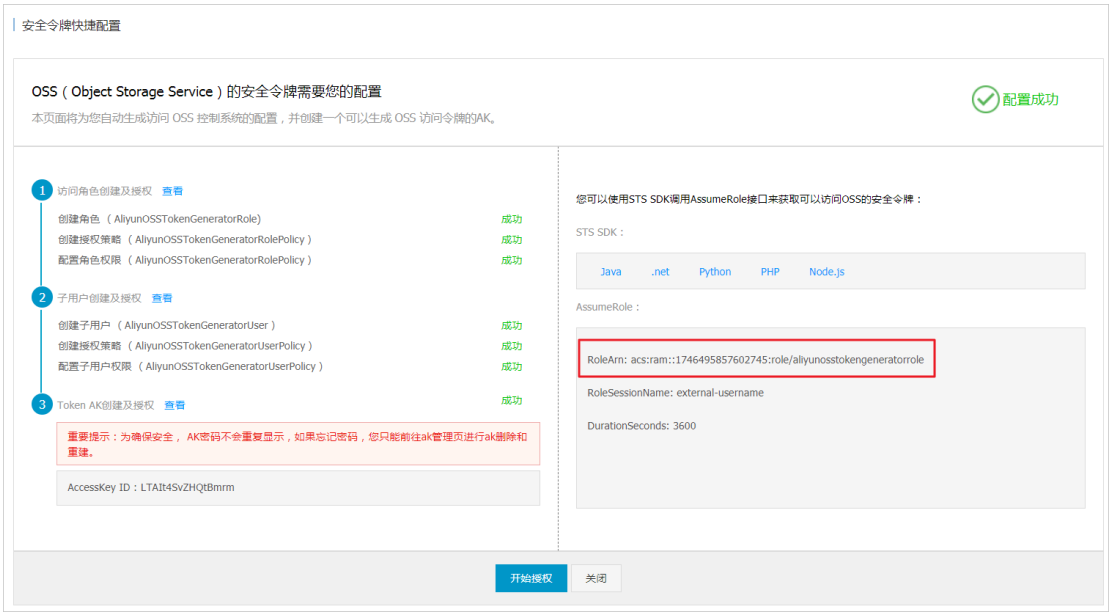
> **说明:**

如果没有开通RAM，会弹出开通的对话框。单击开通。开通完成后单击开始授权。

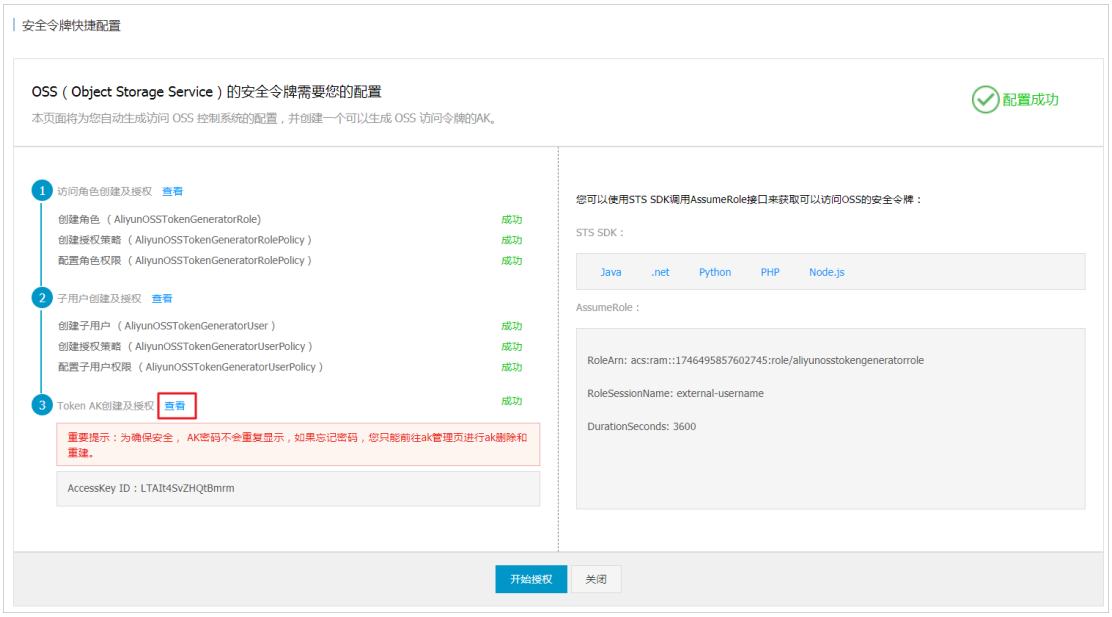
d. 系统进行自动授权。请保存AccessKeyID、AccessKeySecret和RoleArn三个参数。

- 如果您未创建过AccessKey，系统自动创建AccessKey。请保存AccessKeyId、AccessKeySecret和RoleArn，如下图所示。




- 如果您之前已经创建过AccessKey，也可以单击如下图所示的查看创建新的AccessKey。




### 步骤2：下载应用服务器代码示例

- PHP：下载地址
- Java：下载地址
- Ruby：下载地址
- Node.js：下载地址

### 步骤3：修改配置文件

以下例子采用PHP编写。每个语言的示例代码下载后，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"
}
```

相关参数说明如下：

- AccessKeyID：填写之前记录的AccessKeyID。
- AccessKeySecret：填写之前记录的AccessKeySecret。
- RoleArn：填写之前记录的RoleArn。
- TokenExpireTime：指Android/iOS应用获取到的Token的失效时间，最小值和默认值均为900s。

- PolicyFile：该Token所拥有的权限列表的文件，可使用默认值。

> 说明：
>
> 代码示例中提供了三种最常用的Token权限文件，位于policy目录下面。分别是：
>
> - all_policy.txt：指定该Token拥有该账号下的所有权限，包括：
>   - 创建Bucket
>   - 删除Bucket
>   - 上传文件
>   - 下载文件
>   - 删除文件
> - bucket_read_policy.txt：指定该Token拥有该账号下指定Bucket的读权限。
> - bucket_read_write_policy.txt：指定了该Token拥有该账号下指定Bucket的读写权限。
>
> 如果您想要指定该Token只对指定的Bucket有读写权限， 请把bucket_read_policy.txt和bucket_read_write_policy.txt文件里的$BUCKET_NAME替换成指定的Bucket名称。

步骤4：运行示例代码

代码示例的运行方法如下：

- 对于PHP版本，将包下载解压后，修改config.json文件，直接运行php sts.php即能生成Token，将程序部署到指定的应用服务器地址。
- 对于Java版本（依赖于java 1.7），将包下载解压后，修改config.json文件，重新打包并运行java -jar app-token-server.jar (port)。如果不指定port（端口），直接运行java -jar app-token-server.jar，程序会监听7080端口。

返回的数据格式解析如下：

```
//正确返回
{
    "StatusCode":200,
    "AccessKeyId":"STS.3p***dgagdasdg",
    "AccessKeySecret":"rpnwO9***tGdrddgsR2YrTtI",
   "SecurityToken":"CAES+wMIARKAAZhjH0EUOIhJMQBMjRywXq7MQ/cjLYg80Aho
1ek0Jm63XMhr9Oc5s˙ð˙ð3qaPer8p1YaX1NTDiCFZWFkvlHf1pQhuxfKBc+mRR9KAbHUe
fqH+rdjZqjTF7p2m1wJXP8S6k+G2MpHrUe6TYBkJ43GhhTVFMuM3BZajY3VjZWOXBI
ODRIR1FKZjIiEjMzMzE0MjY0NzM5MTE4NjkxMSoLY2xpZGSSDgSDGAGESGTETqOio6c2Rr
LWRlbW8vKgoUYWNzOm9zczoqOio6c2RrLWRlbW9KEDExNDg5MzAxMDcyNDY4MThSBTI2OD
QyWg9Bc3N1bWVkUm9sZVVzZXJgAGoSMzMzMTQyNjQ3MzkxMTg2OTExcglzZGstZGVtbzI
=",
   "Expiration":"2017-12-12T07:49:09Z",
}
```

```
//错误返回
{
    "StatusCode":500,
    "ErrorCode":"InvalidAccessKeyId.NotFound",
    "ErrorMessage":"Specified access key is not found."
}
```

正确返回的五个变量构成一个Token：

- StatusCode：获取Token的状态，获取成功时，返回值是200。
- AccessKeyId：Android/iOS移动应用初始化OSSClient获取的 AccessKeyId。
- AccessKeySecret：Android/iOS移动应用初始化OSSClient获取AccessKeySecret。
- SecurityToken：Android/iOS移动应用初始化的Token。
- Expiration：该Token失效的时间。Android SDK会自动判断Token是否失效，如果失效，则自动获取Token。

错误返回说明如下：

- 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。

步骤5：体验移动应用直传服务

1. 部署程序后，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及地域，修改示例程序里相应的上传Bucket及区域。
3. 单击设置，加载配置。
4. 选择图片，设置上传OSS文件名，上传图片。
5. 上传成功后，通过控制台查看上传结果。

核心代码解析

OSS初始化的代码解析如下：

- 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); // 连接超时时间，默认15秒
conf.setSocketTimeout(15 * 1000); // Socket超时时间，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];
```

## 3.2 权限控制

本文主要介绍如何基于STS Policy的使用规则进行权限控制。

本文基于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:**
>
> - 本文提到的 Policy 是指快速搭建移动应用直传服务提到的config.json中指定的Policy文件内容。
> - 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

  完全授权的Policy表示允许应用可以对OSS进行任何操作。

> **警告:**
>
> 完全授权的Policy对移动应用来说是不安全的授权，不推荐使用。

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

```
}
```

| 获取STS Token后对OSS的操作 | 结果 |
| --- | --- |
| 列举所有创建的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 |
| 下载带前缀的Object，user1/test.txt | 成功 |
| 列举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

  此Policy表示应用对Bucket`app-base-oss`下所有的Object可列举，可下载。

```
{
    "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的操作 | 结果 |
| --- | --- |
| 列举所有创建的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 | 成功 |
| 上传带前缀的Object，user1/test.txt | 失败 |
| 下载带前缀的Object，user1/test.txt | 成功 |
| 列举不带前缀的Object，test.txt | 成功 |
| 列举带前缀的Object，user1/test.txt | 成功 |

- 限制前缀的只读不写Policy

  此Policy表示应用对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。采用此种Policy，如果不同的应用对应不同的前缀，就可以达到在同一个Bucket中空间隔离的效果。

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

| 获取STS Token后对OSS的操作 | 结果 |
| --- | --- |
| 列举所有创建的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 |
| 列举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

  此Policy表示应用可以对Bucket`app-base-oss`下所有的Object进行上传。

```
{
    "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 |
| --- | --- |
| 列举所有创建的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 |
| 上传带前缀的Object，user1/test.txt | 成功 |
| 下载带前缀的Object，user1/test.txt | 成功 |
| 列举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

  此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。采用此种Policy，如果不同的应用对应不同的前缀，就可以达到在同一个Bucket中空间隔离的效果。

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

| 获取STS Token后对OSS的操作 | 结果 |
| --- | --- |
| 列举所有创建的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 | 失败 |
| 列举Object，test.txt | 失败 |
| 带前缀的Object，user1/test.txt | 失败 |

- 不限制前缀的读写Policy

  此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。

```
{
    "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的操作 | 结果 |
| --- | --- |
| 列举所有创建的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 |
| 列举Object，test.txt | 成功 |
| 带前缀的Object，user1/test.txt | 成功 |

- 限制前缀的读写Policy

  此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。采用此种Policy，如果不同的应用对应不同的前缀，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的操作 | 结果 |
| --- | --- |
| 列举所有创建的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 |
| 列举Object，test.txt | 成功 |
| 带前缀的Object，user1/test.txt | 成功 |

总结

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同的用户实现不同的权限控制。
- 可以在应用端做优化，使得STS Token过期之前不需要向应用服务器再次请求。
- Token由STS颁发。应用服务器只是定制了Policy，向STS请求Token，然后将Token转发给应用。这里的Token包含了AccessKeyId、AccessKeySecret、Expiration、SecurityToken，这些参数在OSS提供给应用的SDK中会用到。详情请参见各语言SDK参考中的授权访问章节。

参考文档

- RAM和STS在OSS中的使用指南
- 阿里云RAM官方文档
- 阿里云STS官方文档

## 3.3 快速搭建移动应用上传回调服务

本文讲解如何搭建一个基于OSS的移动应用数据直传服务并设置上传回调。

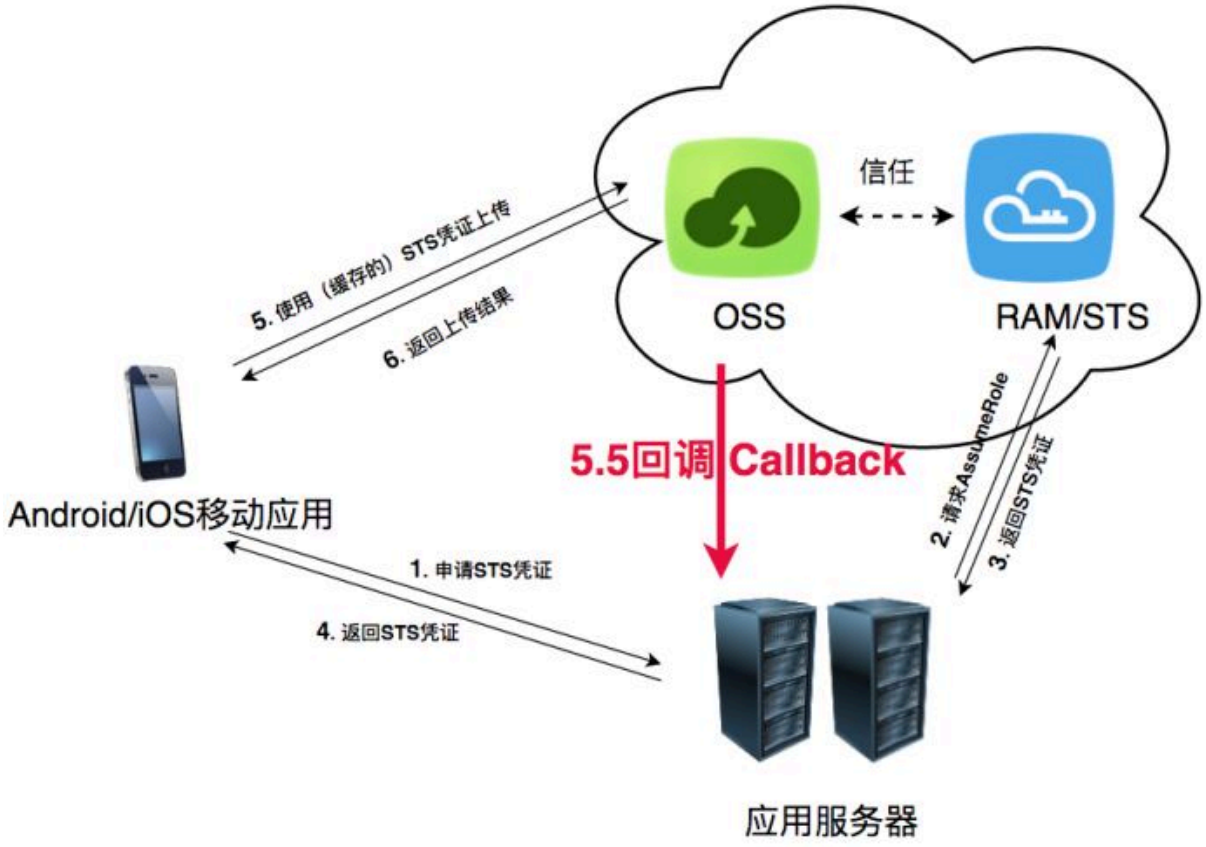
背景

快速搭建移动应用直传服务介绍了如何快速搭建一个基于OSS的移动应用数据直传服务。但这一方案有个问题：对于Android/iOS移动应用来说，只需要申请一次STS凭证，就能多次使用该STS凭

证上传数据到OSS。这就导致应用服务器无法得知用户上传了哪些数据，作为该app的开发者，就没法对应用上传数据进行管理。为此OSS提供了上传回调方案。

### 原理介绍

上传回调的原理如下图所示：




OSS在收到Android/iOS移动应用的数据（上图中步骤5）和在返回用户上传结果（上图中步骤6）之间，触发一个上传回调任务，即第上图中步骤5.5，先回调用户服务器，得到应用服务器返回的内容，然后将此内容返回给Android/iOS移动应用。详情请参见Callback API文档。

### 上传回调的作用

- 通过上传回调让用户应用服务器知道当前上传文件的基本信息。

  返回的基本信息可以包含下表中一个或多个变量，返回内容的格式在Android/iOS上传时指定。

| 系统变量 | 含义 |
| --- | --- |
| bucket | 移动应用上传文件到OSS的哪个存储空间 |
| object | 移动应用上传文件到OSS后保存的文件名 |

| 系统变量 | 含义 |
| --- | --- |
| etag | 该上传的文件的ETag，即返回给用户的etag字段 |
| size | 上传文件的大小 |
| mimeType | 资源类型 |
| imageInfo.height | 图片高度 |
| imageInfo.width | 图片宽度 |
| imageInfo.format | 图片格式，如jpg、png等 |

- 通过上传回调设定自定义参数，达到信息传递的目的。

  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

  Android/iOS移动应用上传文件到OSS时附带上述参数，然后OSS把这些参数放到CallbackBody里发给应用服务器。这样应用服务器就能收到这些信息，达到信息传递的目的。

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地址。
- 您必须给OSS正确的返回，返回格式必须是JSON格式，内容自定义。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。详情请参见Callback API文档。

在移动应用端设置上传回调

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址是http://abc.com/callback.php。您想获取手机上传的文件名称、文件的大小，并且定义了x:phone变量是指手机型号，x:system变量是指操作系统版本。

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:phone": @"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:phone", "IPOHE6S");
         put("x:system", "YunOS5.0");
     }
});
```

应用服务器收到的回调请求

根据设定的不同URL和回调内容，应用服务器收到的回调请求会有所不同，示例如下：

```
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
```

更多内容请参见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。

判断的方法主要是根据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。本文提供的示例程序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。

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200，并把应用服务器返回给OSS的内容原封不动地返回给Android/iOS移动应用。

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， 您需要自行增加对应用服务器收到回调内容的格式解析 。

- Java：
  - 下载地址：单击这里。
  - 运行方法：解压后运行`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：
  - 下载地址：单击这里
  - 运行方法：将解压包部署到Apache环境下，因为PHP本身语言的特点，某些数据头部的获取会依赖于环境。请参考例子根据实际环境进行修改。
- Python：
  - 下载地址：单击这里。
  - 运行方法：解压后直接运行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。

# 4 数据处理与分析

## 4.1 基于OSS+MaxCompute构建数据仓库

本文介绍如何基于OSS并使用MaxCompute构建PB级数据仓库。通过MaxCompute对OSS上的海量数据进行分析，将您的大数据分析工作效率提升至分钟级，帮助您更高效、更低成本的挖掘海量数据价值。

功能介绍

- 对象存储OSS

  对象存储OSS提供标准、低频、归档存储类型，能够覆盖从热到冷的不同存储场景。同时，OSS能够与Hadoop开源社区及EMR、批量计算、MaxCompute、机器学习PAI、DatalakeAnalytics、函数计算等阿里云计算产品进行深度结合。

  您可以打造基于OSS的数据分析应用，如MapReduce、HIVE/Pig/Spark等批处理（如日志离线计算）、交互式查询分析（Imapla、Presto、DataLakeAnalytics）、深度学习训练（阿里云PAI）、基因渲染计算交付（批量计算）、大数据应用（MaxCompute）及流式处理（函数计算）等。

- MaxCompute

  MaxCompute是一项大数据计算服务，能够提供快速且完全托管的数据仓库解决方案，并可以与OSS结合，高效并经济地分析处理海量数据。MaxCompute的处理性能达到了全球领先水平，被Forrester评为全球云端数据仓库领导者。

- OSS外部表查询功能

  MaxCompute重磅推出了一项重要特性：OSS外表查询功能。该功能可以帮助您直接对OSS中的海量文件进行查询，而不必将数据加载到MaxCompute表中，既节约了数据搬迁的时间和人力，也节省了多地存储的成本。

  使用MaxCompute+OSS的方案，您可以获得以下优势：

  - MaxCompute是一个无服务器的分布式计算架构，无需用户对服务器基础设施进行额外的维护和管理，能够根据OSS用户的需求方便及时地提供临时查询服务，帮助企业大幅节省成本。
  - OSS为海量的对象存储服务。用户将数据统一存储在OSS，可以做到计算和存储分离，多种计算应用和业务都可以访问使用OSS的数据，数据只需要存储一份。
  - 支持处理OSS上开源格式的结构化文件，包括：Avro、CSV、ORC、Parquet、RCFile、RegexSerDe、SequenceFile和TextFile，同时支持gzip压缩格式。

应用场景

互联网金融应用每天都需要将大量的金融数据交换文件存放在OSS上，并需要进行超大文本文件的结构化分析。通过MaxCompute的OSS外部表查询功能，用户可以直接用外部表的方式将OSS上的大文件加载到MaxCompute进行分析，从而大幅提升整个链路的效率。

操作示例：物联网采集数据分析

1. 开通OSS服务，详情请参见#unique_40。
2. 创建OSS Bucket，详情请参见创建Bucket 。
3. 开通MaxCompute服务，详情请参见#unique_42。
4. 创建MaxCompute Project，详情请参见创建MaxCompute Project。
5. 授权MaxCompute访问OSS。

   MaxCompute需要直接访问OSS的数据，因此需要将OSS的数据相关权限赋给MaxCompute的访问账号。您可以在直接登录阿里云账号后，单击此处完成一键授权。
6. 将物联网数据上传到OSS。

   说明：

   您可以使用任何数据集来执行测试，以验证我们在这篇文章中概述的最佳实践。本示例在OSS上准备了一批 CSV 数据，endpoint为oss-cn-beijing-internal.aliyuncs.com，bucket为oss-odps-test，数据文件的存放路径为 `/demo/vehicle.csv`。

7. 通过MaxCompute创建外部表，详细步骤请参考创建表，语句如下：

```
CREATE EXTERNAL TABLE IF NOT EXISTS ambulance_data_csv_external
(
    vehicleId int,
    recordId int,
    patientId int,
    calls int,
    locationLatitute double,
    locationLongtitue double,
    recordTime string,
    direction string
    )
    STORED BY 'com.aliyun.odps.CsvStorageHandler'
    LOCATION 'oss://oss-cn-beijing-internal.aliyuncs.com/oss-odps-
test/Demo/';
```

8. 通过MaxCompute查询外部表。成功创建外部表后，便可如普通表一样使用该外部表。查询步骤可参考#unique_45/unique_45_Connect_42_section_ynz_3mq_kgb。

   假设*/demo/vehicle.csv*的数据如下：

```
1,1,51,1,46.81006,-92.08174,9/14/2014 0:00,S
1,2,13,1,46.81006,-92.08174,9/14/2014 0:00,NE
1,3,48,1,46.81006,-92.08174,9/14/2014 0:00,NE
1,4,30,1,46.81006,-92.08174,9/14/2014 0:00,W
1,5,47,1,46.81006,-92.08174,9/14/2014 0:00,S
1,6,9,1,46.81006,-92.08174,9/14/2014 0:00,S
1,7,53,1,46.81006,-92.08174,9/14/2014 0:00,N
1,8,63,1,46.81006,-92.08174,9/14/2014 0:00,SW
1,9,4,1,46.81006,-92.08174,9/14/2014 0:00,NE
1,10,31,1,46.81006,-92.08174,9/14/2014 0:00,N
```

   执行如下 SQL 语句：

```
select recordId, patientId, direction from ambulance_data_csv_e
xternal where patientId > 25;
```

   输出结果如下：

```
+------------+------------+-----------+
| recordId   | patientId  | direction |
+------------+------------+-----------+
| 1          | 51         | S         |
| 3          | 48         | NE        |
| 4          | 30         | W         |
| 5          | 47         | S         |
| 7          | 53         | N         |
| 8          | 63         | SW        |
| 10         | 31         | N         |
+------------+------------+-----------+
```

   更多关于OSS外部表使用方法，请参考#unique_46。

# 4.2 EMR+OSS：离线计算的存储与计算分离

本文主要介绍通过EMR+OSS实现离线计算的存储与计算分离的方案。

## 背景信息

在传统Hadoop的使用中，存储与计算密不可分，而随着业务的发展，集群的规模常常不能满足业务的需求。例如，数据规模超过了集群存储能力，业务上对数据产出的周期提出新的要求导致计算能力跟不上。这就要求我们能随时应对集群存储空间不足或者计算能力不足的挑战。

如果将计算和存储混合部署，常常会因为为了扩存储而带来额外的计算扩容，这其实就是一种浪费；同理，只为了提升计算能力，也会带来一段时期的存储浪费。

将离线计算的计算和存储分离，可以更好地应对单方面的不足。把数据全部放在OSS中，再通过无状态的E-MapReduce分析。E-MapReduce只需进行纯粹的计算，不存在存储跟计算搭配来适应业务了，这样最为灵活。

## 架构

离线计算的存储和计算分离架构简单，如下图所示。OSS作为默认的存储，Hadoop/Spark作为计算引擎直接分析OSS存储的数据。

## 优势

| 因素 | 计算和存储不分离 | 计算和存储分离 |
| --- | --- | --- |
| 灵活性 | 不灵活 | 计算与存储分离后，集群规划简单灵活，基本不需要估算未来业务的规模，做到按需使用。 |
| 成本 | 高 | 在ECS自建的磁盘选择高效云盘，以1 master 8 cpu32g/6 slave 8 cpu32g/10T数据量为例进行估算，存储与计算分离后，成本降低50%。 |
| 性能 | 较高 | 至多下降10%。 |

### 案例测试

- 测试条件

  详细的测试代码请参见GitHub。

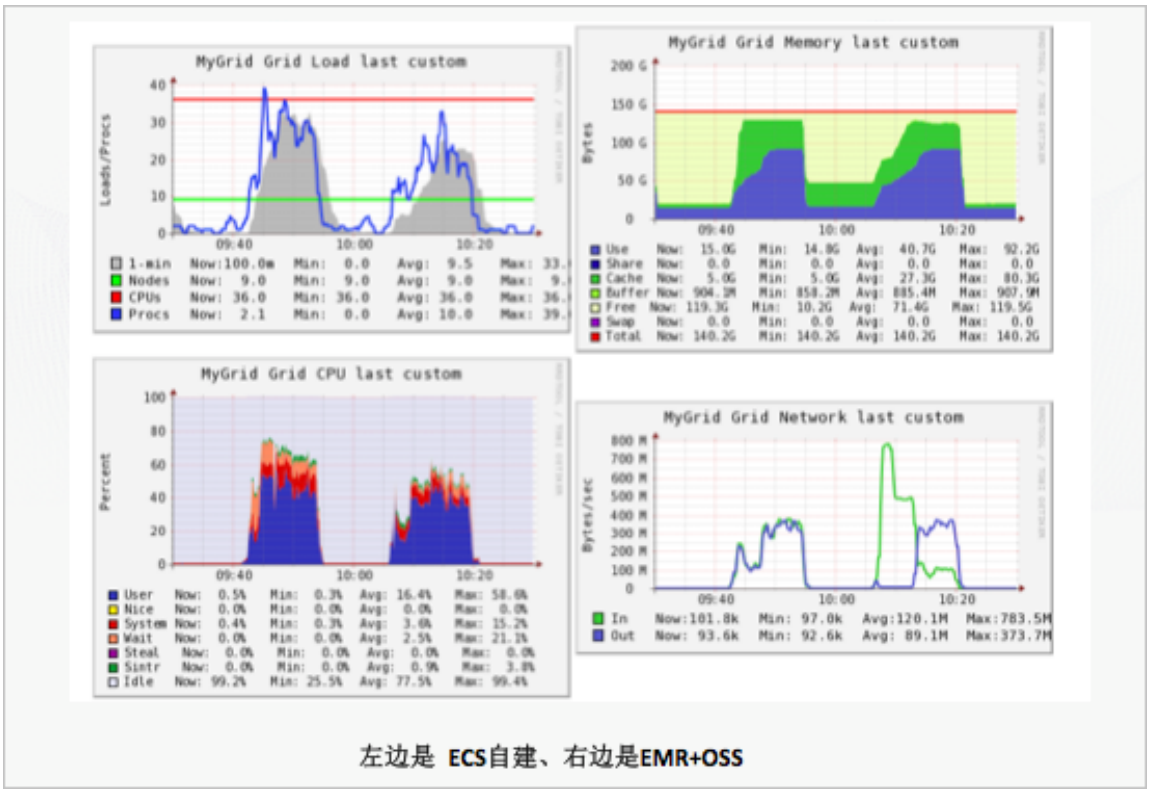
  集群规模：1 master 4cpu 16g、8 Slave 4cpu 16g、每个slave节点250G*4 高效云盘

  Spark测试脚本如下所示。

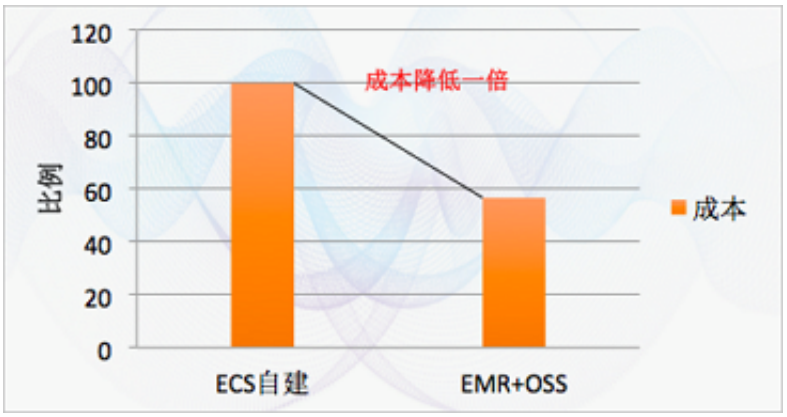
```
/opt/apps/spark-1.6.1-bin-hadoop2.7/bin/spark-submit  --master yarn
 --deploy-mode cluster --executor-memory 3G --num-executors 30
 --conf spark.default.parallelism=800   --class  com.github.ehiggs
```

```
.spark.terasort.TeraSort  spark-terasort-1.0-jar-with-dependencies.
jar /data/teragen_100g /data/terasort_out_100g
```

- **测试结果**
  - **性能**




  - **成本**




  - **时间**

- 结果分析

  从性能图上看，EMR+OSS相较于ECS自建Hadoop，有如下优势：

  - 整体的负载更低。
  - 内存利用率基本一样。
  - CPU使用低，其中iowait与sys低很多。因为ECS自建有datanode及磁盘操作，需要占一些资源，增加CPU的开销。
  - 从网络看，因为sortbenchmark有两次读取数据，第一次是采样，第二次是真正的读取数据，开始网络比较高，随后shuffle+输出结果阶段，网络比ECS自建Hadoop低一半左右。因此从网络来看，整体使用量基本持平。

  综上所述，EMR+OSS比自建ECS使用更少的资源，成本节约了一半，但是性能下降基本可以忽略不计。并且，如果提高EMR+OSS的并发度，时间上有可能比ECS自建Hadoop集群更有优势。

不适用的场景

以下场景不建议使用EMR+OSS：

- 小文件过多的场景

  如果文件小于10M时，请合并小文件。当数据量在128M以上时，使用EMR+OSS的性能最佳。

- 不停操作OSS元数据的场景

  如果是在不停操作元数据的场景中，不建议使用EMR+OSS。建议每次操作OSS元数据时间隔一定的时间。

# 5 数据备份和容灾

## 5.1 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云上备份有以下2种方式：

- 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

- 安装部署

  参见说明及配置 、单机部署 、分布式部署。

- 产品优势

  ossimport工具可以将本地、其它云存储的数据迁移到OSS，它有以下优势：

  - 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
- 常见问题

  参见常见问题。
- 特别说明

  - 不同账户的不同Bucket之间，数据量很大时，例如10TB级别以上的情况，建议使用分布式模式。
  - 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。

# 6 存储空间管理

## 6.1 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，单位为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
```

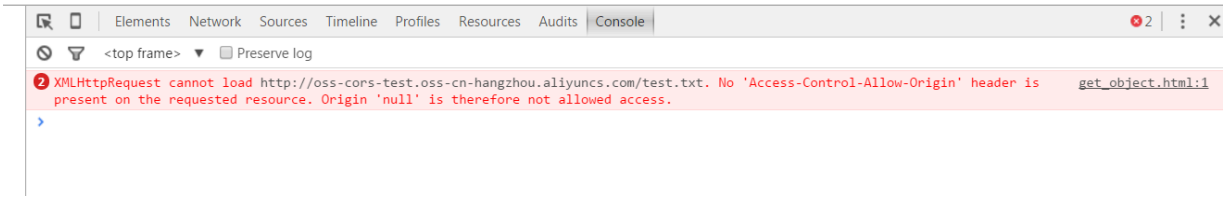
可见现在已经能正常访问了。

**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>
```

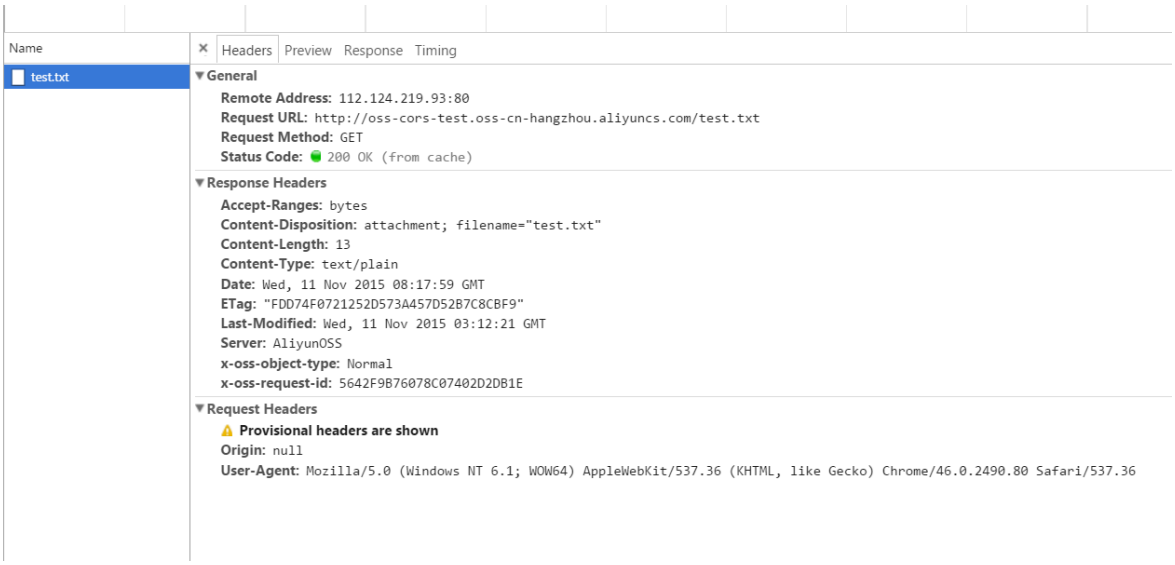
**打开文件后点击链接（以Chrome为例），提示该链接无法访问。**

**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。

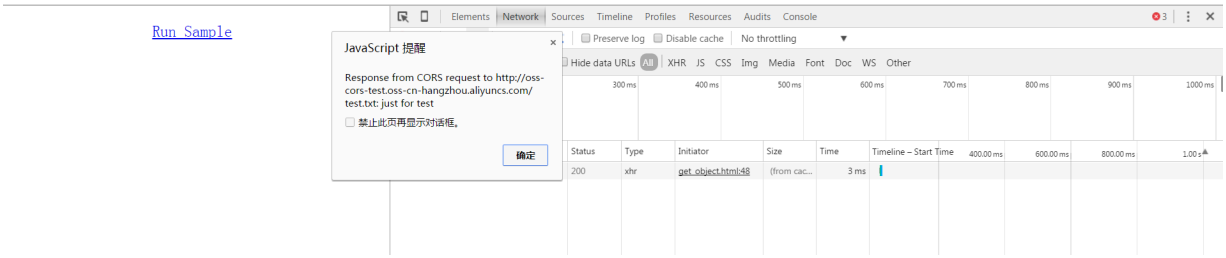


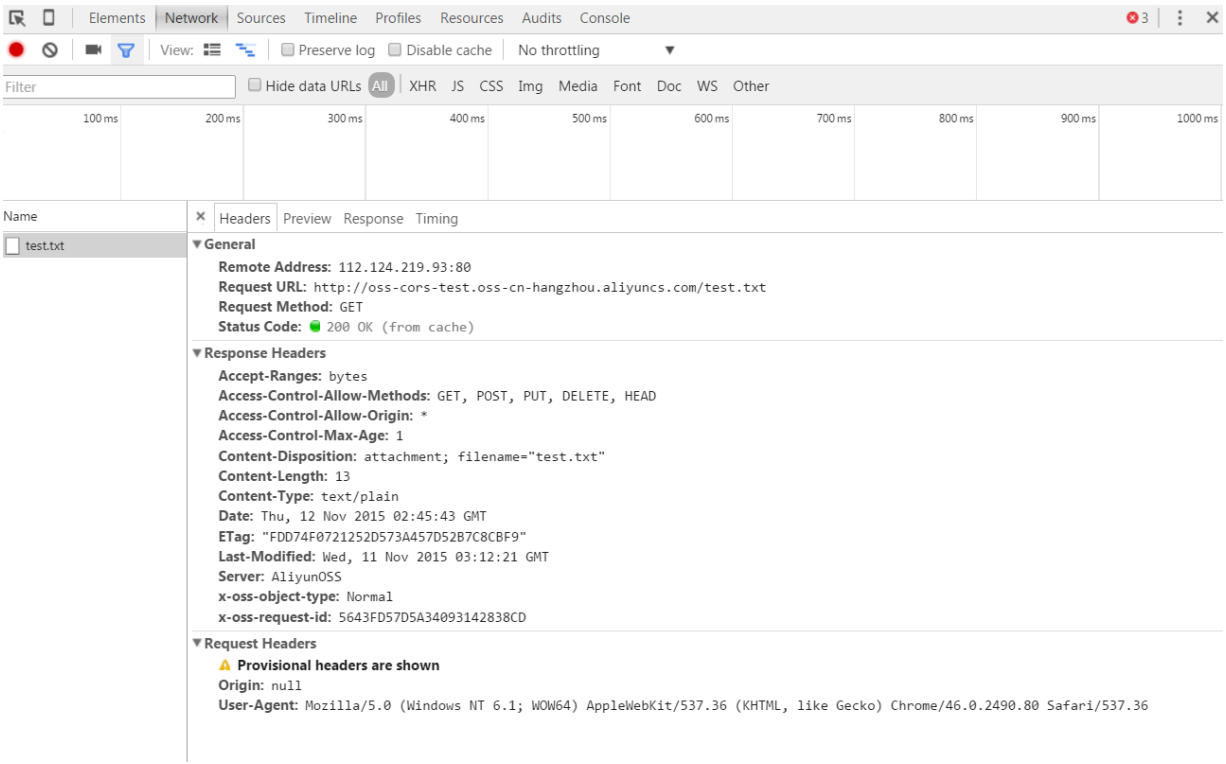

设置 Bucket CORS知道了上文的问题，那么现在可以设置Bucket相关的CORS来使上文的例子能够执行成功。为了直观起见，这里使用控制台来完成CORS的设置。如果用户的CORS设置不是很复杂，也建议使用控制台来完成CORS设置。CORS设置的界面如下：

**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：**



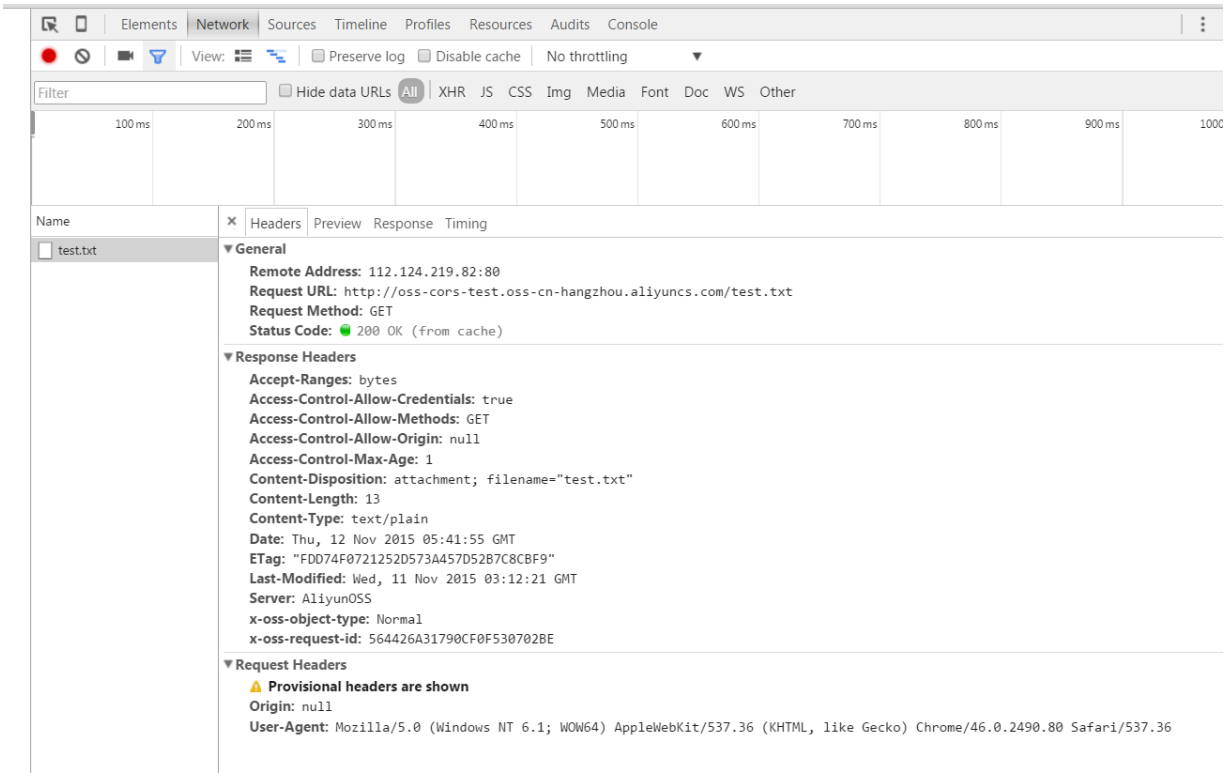




可见现在已经可以正常访问请求了。

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。详情请参见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);
```

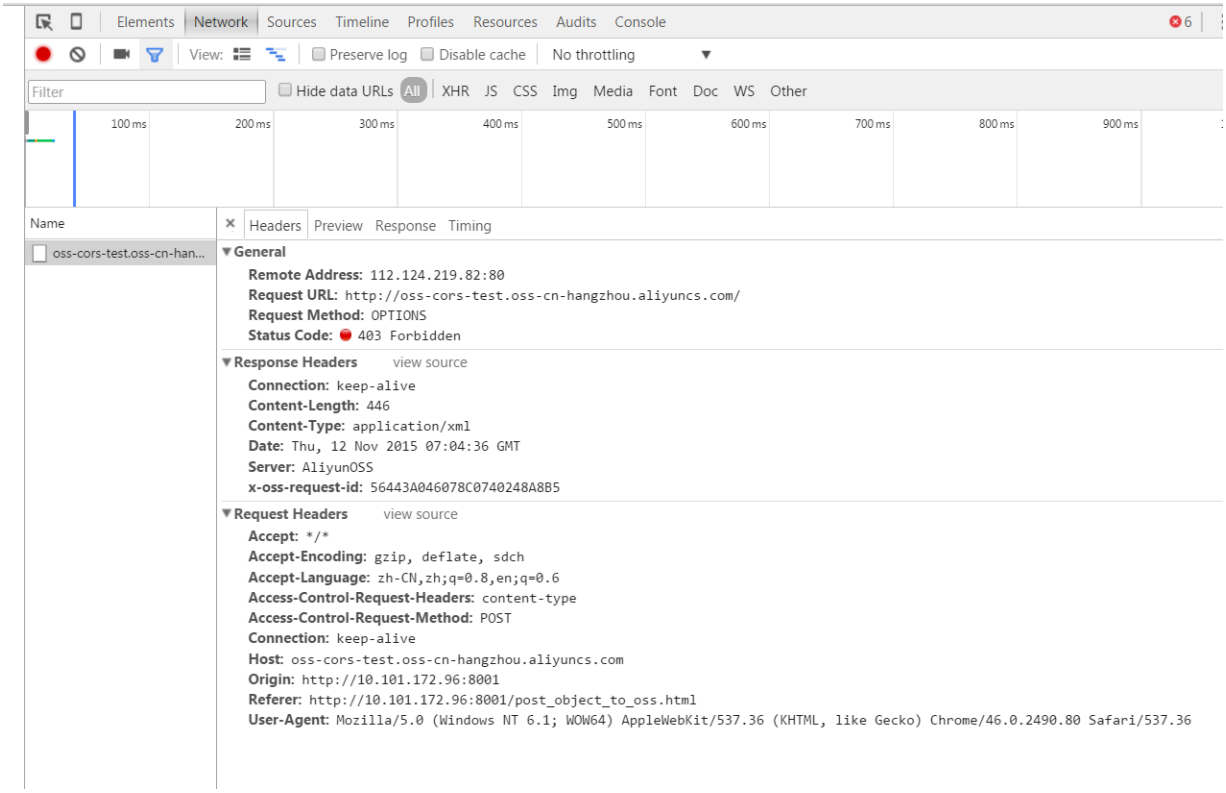
```
    function uploadFile() {
        var file = document.getElementById('file').files[0];
        var fd = new FormData();
        var key = "events/" + (new Date).getTime() + '-' + file.name;
        fd.append('key', key);
        fd.append('Content-Type', file.type);
        fd.append('OSSAccessKeyId', '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。

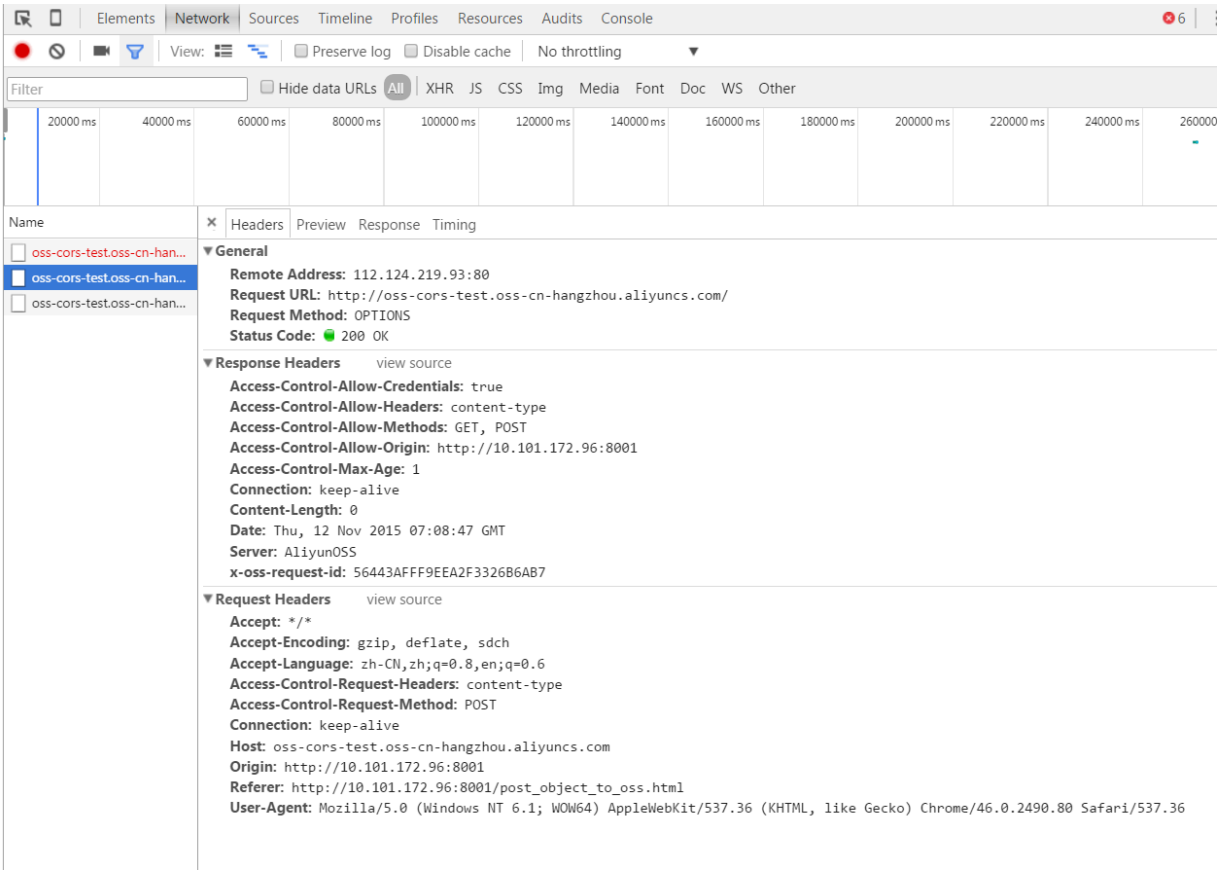
访问这个网页，选择一个文件上传。

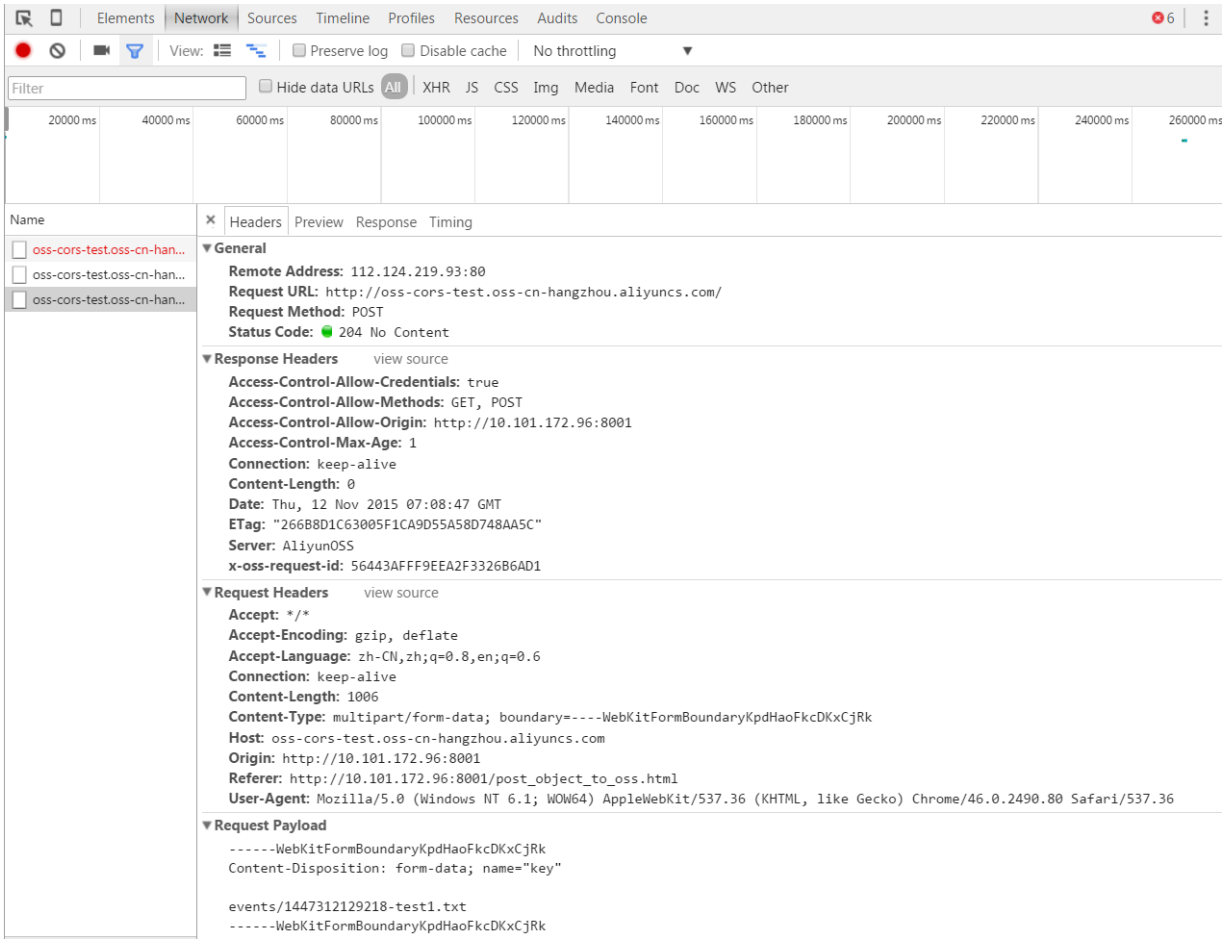
同样打开开发者工具，可以看到以下内容，根据上面的Get的例子，很容易明白同样发生了跨域的错误。不过这里与Get请求的区别在于这是一个带预检的请求，所以可以从图中看到是OPTIONS的Response没有携带CORS相关的Header然后失败了。



**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。

## 6.2 防盗链

### 背景信息

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`。这样的URL（不带签名）要求用户的Bucket权限为公开读权限。

**说明:**

OSS资源外链地址说明请参见OSS访问域名使用规则。

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，如图所示。
- 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 |
| 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 |
| 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：PutBucketReferer
  - 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。
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="a5etodit71tlznjt3pdx****";
 #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。

### 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。Referer同样适用于HTTPS。
- 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。

## 6.3 静态网站托管

本文介绍如何从申请域名开始，基于OSS搭建一个简单的静态网站。

基于OSS搭建一个简单的静态网站的主要步骤是:

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。

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>
```

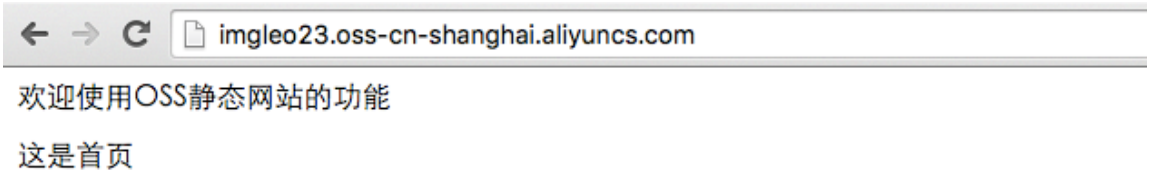
- aliyun-logo.png 是一张图片。

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。

- 显示默认的 404 页

可以看到输入的URL没有对应的文件时，会显示开通时指定的error.html中的内容。

- 显示正常的文件

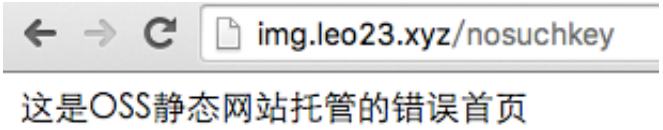
可以看到输入的URL有对应的文件时，会读取成功。

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

说明:

针对中国大陆、中国香港地区的OSS，如果直接使用OSS默认域名从互联网访问OSS上网页类型文件，Response Header中会自动加上 Content-Disposition:'attachment=filename;'。即从浏览器访问网页类型文件时，会以附件形式进行下载。若用户使用自有域名访问OSS的请求，Response Header中不会加上此信息。如何使用自有域名访问OSS，请参考OSS帮助文档绑定自定义域名。

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。

# 7 数据安全

## 7.1 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，并将结果作为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

- PutObject、AppendObject、PostObject、MultipartUploadPart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。
- MultipartComplete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。
- GetObject、HeadObject、GetObjectMeta都会返回对应的crc64值（如有）。客户端可以在GetObject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CopyObject、UploadPartCopy，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PutObject。

```
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GetObject。

```
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UploadPart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 | 是 | OSSCrc64Tests.m |
| Android SDK | 是 | CRC64Test.java |

## 7.2 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。

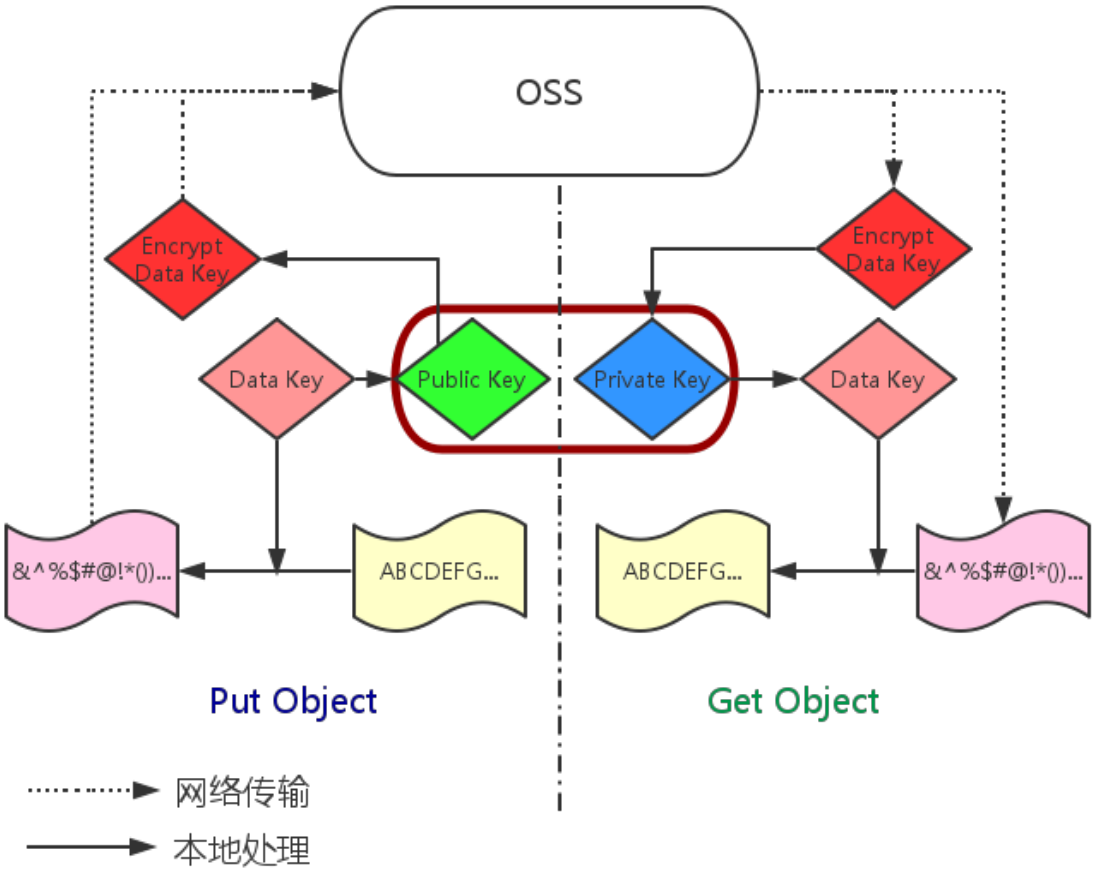
### 原理介绍

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
```

## 7.3 通过服务器端加密保护数据

使用服务器端加密方式保护静态数据，即OSS将用户数据写入数据中心内的磁盘时，会在对象（Object）级别加密数据，并且在访问这些数据时自动解密，用户只需要在请求时验证是否拥有访问权限。

**说明:**

关于服务器端加密的详细介绍可参考开发指南文档#unique_79。

当前OSS支持如下三种服务器端加密方式：

- 使用OSS完全托管的服务器端加密方式（SSE-OSS）

  您可以在上传Object或修改Object的meta信息时，在请求中携带`X-OSS-server-side-encryption`并指定其值为AES256，阿里云OSS服务器端加密使用AES256加密每个对象。OSS会为每个对象使用不同的密钥进行加密，作为额外的保护，它将使用定期轮转的主密钥对加密密钥本身进行加密。

- 使用OSS默认托管的KMS密钥的服务器端加密方式（SSE-KMS）

  您可以在上传Object或修改Object的meta信息时，在请求中携带`X-OSS-server-side-encryption`并指定其值为KMS且不指定具体的CMS ID。OSS将使用默认托管的CMK生成不同的密钥来加密不同的对象，并且在下载时自动解密。

- 使用用户指定的KMS密钥的服务器端加密方式（SSE-KMS）

  您可以在上传Object或修改Object的meta信息时，在请求中携带`X-OSS-server-side-encryption`，指定其值为KMS，并指定`X-OSS-server-side-encryption-key-id`为具体的CMK ID。OSS将使用指定的CMK生成不同的密钥来加密不同的对象，并将加密Object的CMK ID记录到对象的元数据中，因此具有解密权限的用户下载对象时会自动解密。您可以使用系统自动生成的密钥材料，也可以导入外部密钥材料。

  > 注意:
  >
  > - 使用指定的KMS密钥加密的方式目前处于公测中，请联系售后技术支持开通试用。
  > - 同一对象同一时间仅可以使用一种服务器端加密方式。
  > - 使用KMS密钥加密时，原数据中用于加密数据的数据密钥也会被加密，并且作为Object的元数据信息一并存储。
  > - KMS托管密钥的服务器端加密方式仅加密对象数据，不会加密任何对象的元数据。
  > - 使用KMS密钥功能时会产生少量的KMS密钥API调用费用。
  > - 使用子账号对数据使用指定的KMS密钥加密时，子账号需取得KMS相关权限，详情请参考#unique_80。

使用OSS完全托管的服务器端加密方式

1. 登录OSS控制台，创建一个Bucket。配置步骤请参考创建Bucket。
2. 上传1个明文对象至OSS，详情请参考#unique_81。
3. 对已上传的文件进行加密，Python脚本如下：

```
# -*- coding: utf-8 -*-
import oss2

# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以香港为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hongkong.aliyuncs.com',
 'test-hongkong-2025')
```

```
bucket.update_object_meta('01.txt',{'x-oss-server-side-
encryption':'AES256'})
```

4. 验证加密结果。

   使用ossutil工具查看加密前后文件变化。

   - 加密前：

```
D:\5-AK账号\ossutil64>ossutil64.exe stat  oss://test-hongkong-2025
/01.txt
ACL                             : default
Accept-Ranges                   : bytes
Content-Length                  : 62
Content-Md5                     : k2GA4LeqHvVpQvBfnleNOg==
Content-Type                    : text/plain
Etag                            : 936180E0B7AA1EF56942F05F9E578D3A
Last-Modified                   : 2018-10-2420:41:54 +0800 CST
Owner                           : 14166xxxxxx36597
X-Oss-Hash-Crc64ecma            : 9888192182077127097
X-Oss-Object-Type               : Normal
X-Oss-Storage-Class             : Standard
```

   - 加密后：

```
D:\5-AK账号\ossutil64>ossutil64.exe stat  oss://test-hongkong-2025
/01.txt
ACL                             : default
Accept-Ranges                   : bytes
Content-Length                  : 62
Content-Md5                     : k2GA4LeqHvVpQvBfnleNOg==
Content-Type                    : text/plain
Etag                            : 936180E0B7AA1EF56942F05F9E578D3A
Last-Modified                   : 2018-10-2420:46:39 +0800 CST
Owner                           : 14166xxxxxx36597
X-Oss-Hash-Crc64ecma            : 9888192182077127097
X-Oss-Object-Type               : Normal
X-Oss-Server-Side-Encryption: AES256
X-Oss-Storage-Class             : Standard
```

使用OSS默认托管的KMS密钥的服务器端加密方式

1. 登录OSS控制台，创建一个Bucket。配置步骤请参考创建Bucket。
2. 上传1个明文对象至OSS，详情请参考#unique_81。
3. 在云产品管理页，开通KMS服务。
4. 对已上传的文件进行加密，Python脚本如下：

```
# -*- coding: utf-8 -*-
import oss2

# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使
用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>
')
# Endpoint以香港为例，其它Region请按实际情况填写。
```

```
bucket = oss2.Bucket(auth, 'http://oss-cn-hongkong.aliyuncs.com',
 'test-hongkong-2025')

bucket.update_object_meta('01.txt',{'x-oss-server-side-
encryption':'KMS'})
```

使用用户指定的KMS密钥的服务器端加密方式

1. 登录OSS控制台，创建一个Bucket。配置步骤请参考创建Bucket。
2. 上传1个明文对象至OSS，详情请参考#unique_81。
3. 在云产品管理页，开通KMS服务。
4. 登录KMS管理控制台，单击创建密钥，创建一个和Bucket同区域的密钥。
   - 描述：自定义。
   - 高级选项内，密钥材料来源选择：阿里云KMS。

   **说明:**

   您也可以导入外部密钥，详情请参考#unique_83。
5. 使用指定的CMK ID加密已上传的对象，Python脚本如下：

```
# -*- coding: utf-8 -*-
import oss2

# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使
用RAM账号进行API访问或日常运维，请登录RAM控制台创建RAM账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>
')
# Endpoint以香港为例，其它Region请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hongkong.aliyuncs.com',
 'test-hongkong-2025')

bucket.update_object_meta('01.txt',{'x-oss-server-side-
encryption':'KMS','x-oss-server-side-encryption-key-id':
 '33701a45-6723-4a04-a367-68c060382652'})
```

# 8 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

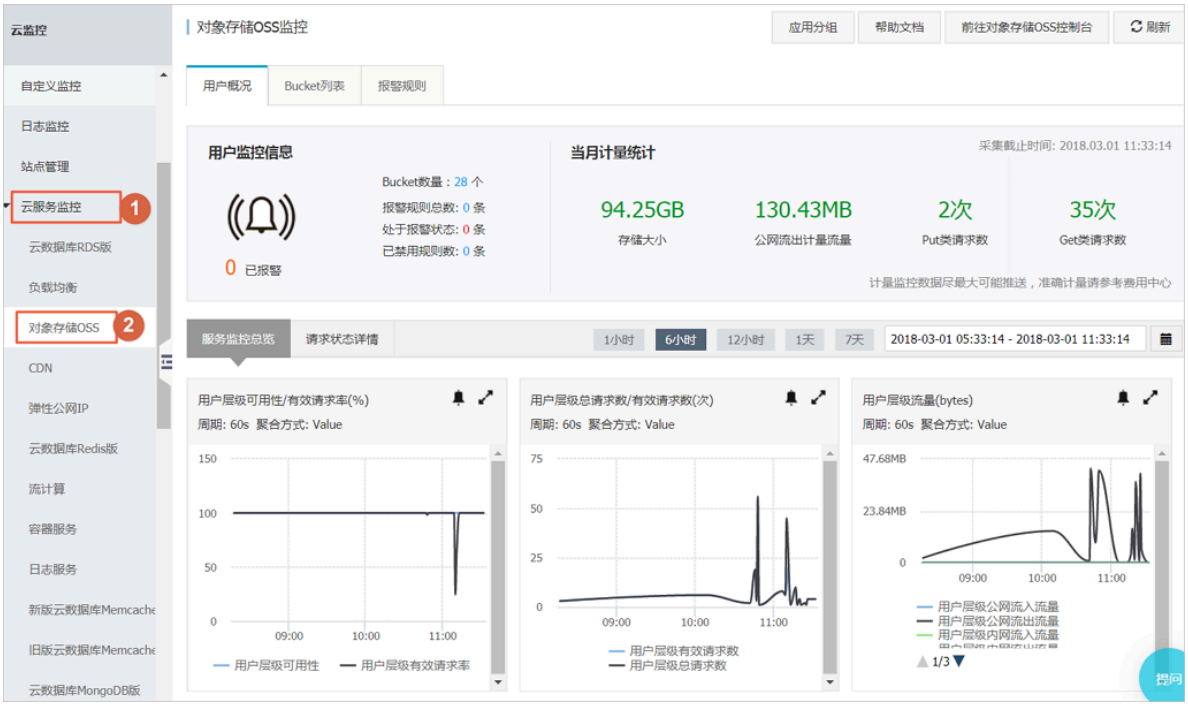
- 已开通OSS服务。
- 已开通云监控服务。

## 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。

# 9 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，如果您在上传大量对象时，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，当您的请求速率超过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，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。

# 10 安卓应用示例

## 10.1 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。

## 10.2 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);
 }
```

## 10.3 上传文件

**简单上传**

**简单上传就是调用OSS API中的PutObject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. 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_CONTENT_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MultipartUpload（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. 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_CONTENT_URI);
         startActivityForResult(i, RESULT_PAUSEABLEUPLOAD_IMAGE);
     }
 }
 );
```

3. 点击继续，完成断点续传。

   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.
asyncMultiPartUpload(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,
PauseableUploadResult> userCallback,
                    final OSSProgressCallback<PauseableUploadReque
st> userProgressCallback) {     if (object.equals("")) {
Log.w("AsyncMultiPartUpload", "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; }
```

## 10.4 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, GetObjectResult 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");
```

# 11 Terraform

## 11.1 Terraform简介

Terraform 是一个开源的自动化的资源编排工具，支持多家云服务提供商。阿里云作为第三大云服务提供商，terraform-alicloud-provider 已经支持了超过 90 多个 Resource 和 Data Source，覆盖 20 多个服务和产品，吸引了越来越多的开发者加入到阿里云 Terraform 生态的建设中。

HashiCorp Terraform 是一个IT基础架构自动化编排工具，可以用代码来管理维护 IT 资源。Terraform 的命令行接口（CLI） 提供一种简单机制，用于将配置文件部署到阿里云或其他任意支持的云上，并对其进行版本控制。它编写了描述云资源拓扑的配置文件中的基础结构，例如虚拟机、存储帐户和网络接口。Terraform 的命令行接口（CLI）提供一种简单机制，用于将配置文件部署到阿里云或任何其他支持的云并对其进行版本控制。Terraform 是一个高度可扩展的工具，通过 Provider 来支持新的基础架构。您可以使用 Terraform 来创建、修改或删除 OSS、ECS、VPC、RDS、SLB 等多种资源。

### OSS Terraform Module 功能

OSS 的 Terraform Module 目前主要提供 Bucket 管理、文件对象管理的功能。例如：

- Bucket 管理功能：
  - 创建 Bucket
  - 设置 Bucket ACL
  - 设置 Bucket CORS
  - 设置 Bucket Logging
  - 设置 Bucket 静态网站托管
  - 设置 Bucket Referer
  - 设置 Bucket Lifecycle
- Object 管理功能：
  - 文件上传
  - 设置文件服务端加密方式
  - 设置 ACL
  - 设置对象元数据信息

### 参考文档

- 安装及使用 Terraform 请参见：使用Terraform管理OSS
- OSS Terraform Module 下载地址请参见：terraform-alicloud-modules
- 更多 OSS Terraform Module 介绍请参见：alicloud_oss_bucket

## 11.2 使用Terraform管理OSS

本文主要介绍如何安装配置 Terraform 及使用 Terraform 管理 OSS。

### 安装和配置 Terraform

使用 Terraform 前，您需要参考以下步骤安装并配置 Terraform：

1. 前往 Terraform 官网 下载适用于您的操作系统的程序包，本文以 Linux 系统为例。
2. 将程序包解压到 */usr/local/bin*。如果将可执行文件解压到其他目录，则需要将路径加入到全局变量。
3. 运行 Terraform 验证路径配置，若显示可用的 Terraform 选项的列表，表示安装完成。

```
[root@test bin]#terraform
Usage: terraform [-version] [-help] <command> [args]
```

4. 创建 RAM 用户，并为其授权。
   a. 登录 RAM 控制台。
   b. 创建名为 *Terraform* 的 RAM 用户，并为该用户创建 AccessKey。具体步骤参见创建 RAM 用户。
   c. 为 RAM 用户授权。您可以根据实际的情况为 *Terraform* 授予合适的管理权限。具体步骤参见为 RAM 用户授权。

**注意:**
请不要使用主账号的 AccessKey 配置 Terraform 工具。

5. 因为每个 Terraform 项目都需要创建 1 个独立的执行目录，所以先创建一个测试目录 *terraform-test*。

```
[root@test bin]#mkdir terraform-test
```

6. 进入 *terraform-test* 目录：

```
[root@test bin]#cd terraform-test
```

```
[root@test terraform-test]#
```

7. 创建配置文件。Terraform 在运行时，会读取该目录空间下所有*.tf和*.tfvars 文件。因此，您可以按照实际用途将配置信息写入到不同的文件中。下面列出几个常用的配置文件：

```
provider.tf                -- provider 配置
terraform.tfvars      -- 配置 provider 要用到的变量
varable.tf                  -- 通用变量
resource.tf                -- 资源定义
data.tf                        -- 包文件定义
output.tf                   -- 输出
```

如创建 *provider.tf* 文件时，您可按以下格式配置您的身份认证信息：

```
[root@test terraform-test]# vim provider.tf
provider "alicloud" {
    region           = "cn-beijing"
    access_key  = "LTA**********NO2"
    secret_key   = "MOk8x0**********************wwff"
```

更多配置信息请参考：alicloud_oss_bucket。

8. 初始化工作目录：

```
[root@test terraform-test]#terraform init


Initializing provider plugins...
- Checking for available provider plugins on https://releases.
hashicorp.com...
- Downloading plugin for provider "alicloud" (1.25.0)...




The following providers do not have any version constraints in
configuration,
so the latest version was installed.


To prevent automatic upgrades to new major versions that may contain
 breaking
changes, it is recommended to add version = "..." constraints to the
corresponding provider blocks in configuration, with the constraint
strings
suggested below.


* provider.alicloud: version = "~> 1.25"


Terraform has been successfully initialized!


You may now begin working with Terraform. Try running "terraform
plan" to see
any changes that are required for your infrastructure. All Terraform
 commands
should now work.
```

```
If you ever set or change modules or backend configuration for
Terraform,
rerun this command to reinitialize your working directory. If you
forget, other
commands will detect it and remind you to do so if necessary.
```

**注意:**

**每个 Terraform 项目在新建 Terraform 工作目录，并创建配置文件后，都需要初始化工作目录。**

**以上操作完成之后，您就可以使用 Terraform 工具了。**

### 使用 Terraform 管理 OSS

**Terraform 安装完成之后，您就可以通过 Terraform 的操作命令管理 OSS 了，下面介绍几个常用的操作命令：**

- `terraform plan`：**预览功能，允许在正式执行之前查看将要执行那些操作。**

  **例如，您添加了一个创建 Bucket 的配置文件 test.tf：**

```
[root@test terraform-test]#vim test.tf
resource "alicloud_oss_bucket" "bucket-acl"{
  bucket = "figo-chen-2020"
  acl = "private"
}
```

  **使用 `terraform plan` 可查看到将会执行的操作：**

```
[root@test terraform-test]# terraform plan
Refreshing Terraform state in-memory prior to plan...
The refreshed state will be used to calculate this plan, but will
not be
persisted to local or remote state storage.


------------------------------------------------------------------------

An execution plan has been generated and is shown below.
Resource actions are indicated with the following symbols:
  + create

Terraform will perform the following actions:

  + alicloud_oss_bucket.bucket-acl
      id:                 <computed>
      acl:                "private"
      bucket:             "figo-chen-2020"
      creation_date:      <computed>
      extranet_endpoint:  <computed>
      intranet_endpoint:  <computed>
      location:           <computed>
      logging_isenable:   "true"
      owner:              <computed>
```

```
      referer_config.#:  <computed>
      storage_class:     <computed>


Plan: 1 to add, 0 to change, 0 to destroy.

------------------------------------------------------------------------

Note: You didn't specify an "-out" parameter to save this plan, so
Terraform
can't guarantee that exactly these actions will be performed if
"terraform apply" is subsequently run.
```

- terraform apply：**执行工作目录中的配置文件**。

  **例如，您想创建一个名为** *figo-chen-2020* **的 Bucket，您需要先添加一个创建 Bucket 的配置文件 test.tf：**

```
[root@test terraform-test]#vim test.tf
resource "alicloud_oss_bucket" "bucket-acl"{
  bucket = "figo-chen-2020"
  acl = "private"
}
```

  **之后使用** terraform apply **命令执行配置文件即可**。

```
[root@test terraform-test]#terraform  apply

An execution plan has been generated and is shown below.
Resource actions are indicated with the following symbols:
  + create

Terraform will perform the following actions:

  + alicloud_oss_bucket.bucket-acl
      id:                <computed>
      acl:               "private"
      bucket:            "figo-chen-2020"
      creation_date:     <computed>
      extranet_endpoint: <computed>
      intranet_endpoint: <computed>
      location:          <computed>
      logging_isenable:  "true"
      owner:             <computed>
      referer_config.#:  <computed>
      storage_class:     <computed>


Plan: 1 to add, 0 to change, 0 to destroy.

Do you want to perform these actions?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.

  Enter a value: yes

alicloud_oss_bucket.bucket-acl: Creating...
  acl:               "" => "private"
  bucket:            "" => "figo-chen-2020"
  creation_date:     "" => "<computed>"
  extranet_endpoint: "" => "<computed>"
```

```
  intranet_endpoint: "" => "<computed>"
  location:          "" => "<computed>"
  logging_isenable:  "" => "true"
  owner:             "" => "<computed>"
  referer_config.#:  "" => "<computed>"
  storage_class:     "" => "<computed>"
alicloud_oss_bucket.bucket-acl: Creation complete after 1s (ID: figo
-chen-2020)

Apply complete! Resources: 1 added, 0 changed, 0 destroyed.
```

**说明:**

**此配置运行后，若 *figo-chen-2020* 这个 Bucket 不存在，则创建一个 Bucket；若已存在，且为 Terraform 创建的空 Bucket，则会删除原有 Bucket 并重新生成。**

- `terraform destroy`：可删除通过 Terraform 创建的空的 Bucket。
- 导入 Bucket：若 Bucket 不是通过 Terraform 创建，可通过命令导入现有的 Bucket。

  首先，创建一个 *main.tf* 的文件，并写入 Bucket 相关信息：

```
[root@test terraform-test]#vim main.tf
resource "alicloud_oss_bucket" "bucket" {
 bucket = "test-hangzhou-2025"
 acl = "private"
}
```

  使用如下命令导入 *test-hangzhou-2025* 这个 Bucket：

```
terraform import alicloud_oss_bucket.bucket test-hangzhou-2025
```

**参考文档**

- 更多 Bucket 配置操作示例请参考：alicloud_oss_bucket
- 更多Object相关配置操作示例请参考：alicloud_oss_bucket_object