阿里云
对象存储

最佳实践

文档版本：20190917

法律声明

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

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

通用约定

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

1 简介

阿里云对象存储 OSS 最佳实践主要介绍数据迁移、数据备份和容灾、数据直传 OSS、数据处理与分析、音视频转码、使用 Terraform 管理 OSS 等操作，帮助您更加高效地使用 OSS，满足您的业务需求。

数据迁移

- OSS 之间数据迁移
- 第三方数据源迁移到 OSS
- 从AWS S3上的应用无缝切换至OSS
- #unique_7

数据备份和容灾

- #unique_8
- 数据库备份到OSS

数据直传 OSS

- Web 端直传实践，包括通过 OSS Browser.js SDK 直传数据到 OSS、表单上传（PostObject）、小程序直传实践
- 移动端直传实践

数据处理与分析

- 基于OSS+MaxCompute构建数据仓库
- EMR+OSS：离线计算的存储与计算分离
- 使用OSS中的数据作为机器学习的训练样本
- DataLakeAnalytics+OSS：基于OSS的Severless的交互式查询分析
- 通过HDP2.6 Hadoop读取和写入OSS数据
- 通过CDH5 Hadoop读取和写入OSS数据
- Apache Impala（CDH6）查询OSS数据
- Spark使用OSS Select加速数据查询

音视频转码

- #unique_23
- #unique_24

性能与扩展性

OSS性能与扩展性最佳实践

使用 ECS 实例反向代理 OSS

- 基于CentOS的ECS实例实现OSS反向代理
- 基于Ubuntu的ECS实例实现OSS反向代理

Terraform

- 使用Terraform管理OSS

2 数据迁移

2.1 OSS 之间数据迁移

您可以使用阿里云在线迁移服务在阿里云对象存储 OSS 之间进行跨账号、跨地域、以及同地域内的数据迁移。

OSS 之间数据迁移包含以下场景：

- 同地域数据迁移，即同地域 Bucket 之间数据迁移
- 跨地域数据迁移，即不同地域 Bucket 之间数据迁移
- 跨账号数据迁移，即不同阿里云账号之间 Bucket 数据迁移

使用在线迁移服务，您只需在控制台填写源数据地址和目标 OSS 地址信息，并创建迁移任务即可。启动迁移后，您可以通过控制台管理迁移任务，查看迁移进度、流量等信息；也可以生成迁移报告，查看迁移文件列表、错误文件列表。详情请参见阿里云 OSS 之间迁移教程。

2.2 第三方数据源迁移到 OSS

您可以使用阿里云在线迁移服务将第三方数据源轻松迁移至阿里云对象存储 OSS。

使用在线迁移服务，您只需在控制台填写源数据地址和目标 OSS 地址信息，并创建迁移任务即可。启动迁移后，您可以通过控制台管理迁移任务，查看迁移进度、流量等信息；也可以生成迁移报告，查看迁移文件列表、错误文件列表。详情请参见：

- HTTP/HTTPS 源迁移教程
- 腾讯云 COS 迁移教程
- AWS S3 迁移教程
- 七牛云迁移教程
- Azure Blob 迁移教程
- 又拍云迁移教程
- 百度云 BOS 迁移教程
- 金山云 KS3 迁移教程
- ECS 数据迁移至 OSS 教程
- NAS 迁移至 OSS 教程

2.3 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。
	WRITE	Bucket下的Object可写入或覆盖	- 对于Bucket下不存在的Object，可写入。 - 对于Bucket下存在的Object，如果该Object没有设置Object权限，则该Object可被覆盖。 - 可以初始化分片上传（InitiateMultipartUpload）。

级别	AWS S3权限	AWS S3	阿里云OSS
	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

2.4 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_44。

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。

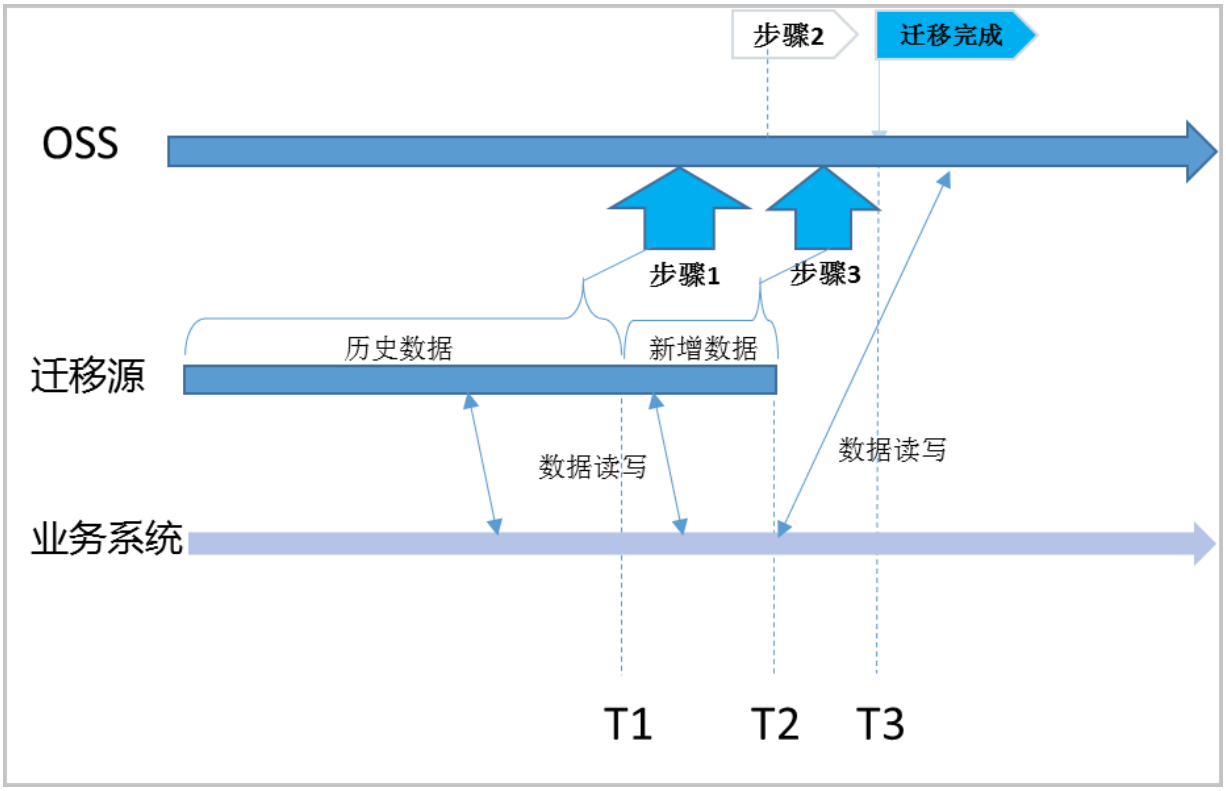
  ECS所需数量的计算公式为：X/Y/(Z/100)台。其中X为需要迁移的数据量、Y为要求迁移完成的时间（天）、Z为单台ECS迁移速度Z Mbps（每天迁移约Z/100 TB数据）。假设单台ECS迁移速度达到200Mbps（即每天约迁移2TB数据），则上述示例中需购买ECS 36台（即500/7/2）。
- 配置ossimport

  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

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

   系统将启动一个Hadoop MapReduce任务（DistCp）。

4. 运行完毕之后会显示本次数据迁移的信息。信息内容类似如下所示。

   ```
   17/05/04 22:35:08 INFO mapreduce.Job: Job job_1493800598643_0009
   completed successfully
   17/05/04 22:35:08 INFO mapreduce.Job: Counters: 38
    File System Counters
            FILE: Number of bytes read=0
            FILE: Number of bytes written=859530
            FILE: Number of read operations=0
            FILE: Number of large read operations=0
            FILE: Number of write operations=0
            HDFS: Number of bytes read=263114
            HDFS: Number of bytes written=0
            HDFS: Number of read operations=70
            HDFS: Number of large read operations=0
            HDFS: Number of write operations=14
            OSS: Number of bytes read=0
   ```

```
        OSS: Number of bytes written=258660
        OSS: Number of read operations=0
        OSS: Number of large read operations=0
        OSS: Number of write operations=0
Job Counters
        Launched map tasks=7
        Other local map tasks=7
        Total time spent by all maps in occupied slots (ms)=60020
        Total time spent by all reduces in occupied slots (ms)=0
        Total time spent by all map tasks (ms)=30010
        Total vcore-milliseconds taken by all map tasks=30010
        Total megabyte-milliseconds taken by all map tasks=45015000
Map-Reduce Framework
        Map input records=10
        Map output records=0
        Input split bytes=952
        Spilled Records=0
        Failed Shuffles=0
        Merged Map outputs=0
        GC time elapsed (ms)=542
        CPU time spent (ms)=14290
        Physical memory (bytes) snapshot=1562365952
        Virtual memory (bytes) snapshot=17317421056
        Total committed heap usage (bytes)=1167589376
File Input Format Counters
        Bytes Read=3502
File Output Format Counters
        Bytes Written=0
org.apache.hadoop.tools.mapred.CopyMapper$Counter
        BYTESCOPIED=258660
        BYTESEXPECTED=258660
        COPY=10
copy from /path/on/hdfs to oss://accessKeyId:accessKeySecret@bucket-
name.oss-cn-hangzhou.aliyuncs.com/path/on/oss does succeed !!!
```

5. 您可以用osscmd等工具查看OSS上数据情况。

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

OSS数据迁移到Hadoop

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

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

更多使用场景

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

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

1. 使用emr-tools迁移数据到OSS上。
2. 在VPC环境中新建一个集群（自建或使用E-MapReduce服务）。

3. 将数据从OSS上迁移到新的HDFS集群中。

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

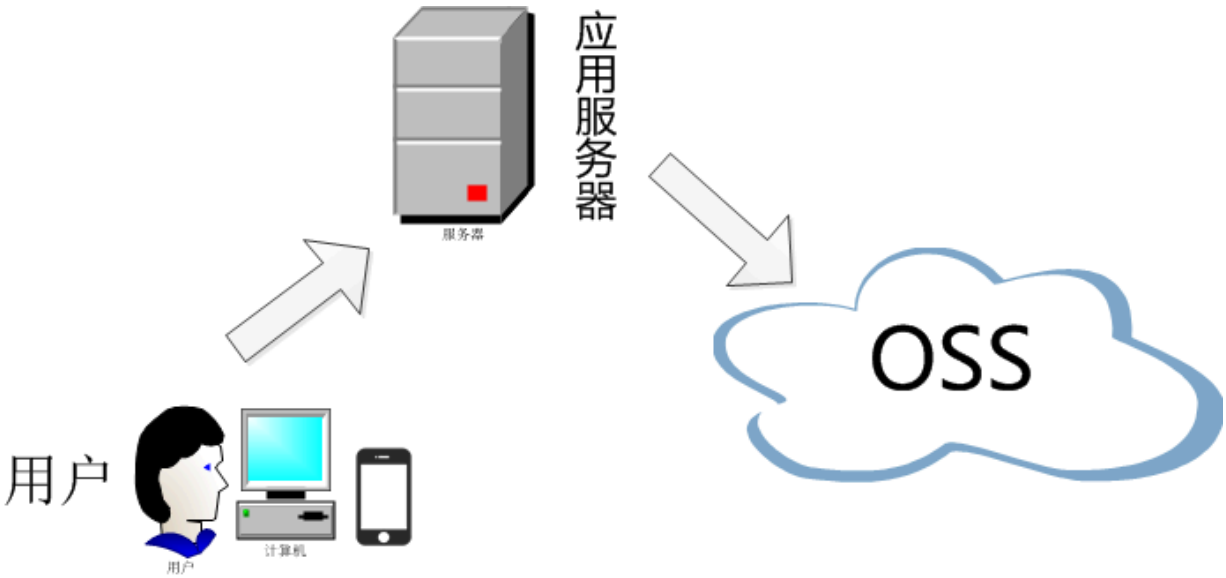
3 Web端上传数据至OSS

3.1 Web端上传介绍

本教程介绍如何利用 Web 前端技术将文件上传到 OSS。

背景信息

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，应用服务器再把文件上传到 OSS。具体流程如下图所示。



和数据直传到 OSS 相比，以上方法有三个缺点：

- 上传慢：用户数据需先上传到应用服务器，之后再上传到OSS。网络传输时间比直传到OSS多一倍。如果用户数据不通过应用服务器中转，而是直传到OSS，速度将大大提升。而且OSS采用BGP带宽，能保证各地各运营商之间的传输速度。
- 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。

技术方案

目前通过 Web 前端技术上传文件到 OSS，有三种技术方案：

- 利用OSS Browser.js SDK 将文件上传到 OSS

  该方案通过OSS Browser.js SDK直传数据到 OSS，详细的 SDK Demo 请参考#unique_52。在网络条件不好的状况下可以通过断点续传的方式上传大文件。该方案在个别浏览器上有兼容性

问题，目前兼容 IE10 及以上版本浏览器，主流版本的 Edge、Chrome、Firefox、Safari 浏览器，以及大部分的 Android、iOS、WindowsPhone 手机上的浏览器。更多信息请参见安装 Browser.js SDK。

> 说明:
> Browser.js SDK 已经支持大部分的 OSS API 接口，包含文件管理、自定义域名设置、图片处理等。

- 使用表单上传方式，将文件上传到 OSS

  利用 OSS 提供的 PostObject 接口，使用表单上传方式将文件上传到 OSS。该方案兼容大部分浏览器，但在网络状况不好的时候，如果单个文件上传失败，只能重试上传。操作方法请参见 PostObject 上传方案。

  > 说明:
  > 关于 PostObject 的详细介绍请参见 PostObject 。

- 通过小程序上传文件到 OSS

  通过小程序，如微信小程序、支付宝小程序等，利用 OSS 提供的 PostObject 接口来实现表单上传。操作方式请参见小程序上传实践。

3.2 Web端PostObject直传实践

3.2.1 Web端PostObject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。

每个OSS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，应用服务器再把文件上传到OSS。这种方式需通过应用服务器中转，传输效率明显低于数据直转至OSS的方式。

数据直转至OSS的方式是利用OSS提供的PostObject接口，使用表单上传方式将文件上传到OSS。您可以通过以下案例了解如何通过表单上传的方式，直传数据到OSS：

- 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。详情请参见JavaScript客户端签名直传。
- 在服务端完成签名，然后通过表单直传数据到OSS。详情请参见服务端签名后直传。
- 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。详情请参见服务端签名直传并设置上传回调。

3.2.2 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。

步骤 1：下载应用服务器代码

下载地址

本示例采用Plupload直接提交表单数据（即PostObject）到OSS，可以运行于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。

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';
```

- accessid：您的AccessKeyId。
- accesskey：您的AcessKeySecret。
- token：您的STS token。使用STS方式验证时，您需要通过STS API获取STS AccessKeyId、STS AcessKeySecret、SecurityToken，详情请参见#unique_62。
- host：您的OSS访问域名，格式为`BucketName.Endpoint`，例如`post-test.oss-cn-hangzhou.aliyuncs.com`。关于OSS访问域名的介绍请参见#unique_63。

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。操作步骤请参见设置跨域访问，配置如下图所示。

说明:

实际使用时，为了您的数据安全，来源栏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。

3.2.3 服务端签名后直传

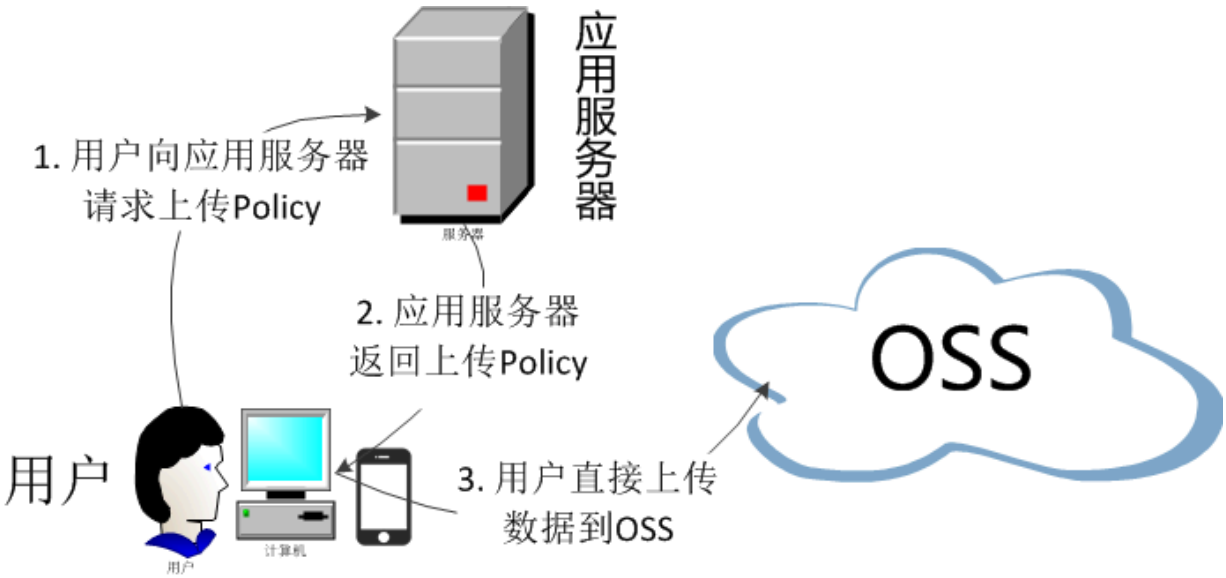
本示例介绍如何在服务端完成签名并通过表单直传数据到OSS。

> 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 。

代码示例

请参考服务端签名直传并设置上传回调中的各语言版本示例：

- Java
- Python
- PHP
- Go
- Ruby

本场景是服务端签名后直传，并没有上传回调。与服务端签名直传并设置上传回调示例中的区别在于，在下载的客户端源码中打开upload.js文件，找到如下代码片段：

```
{
  'key' : g_object_name,
  'policy': policyBase64,
  'OSSAccessKeyId': accessid,
  'success_action_status' : '200', //让服务端返回200,不然，默认会返回204
  'callback' : callbackbody,
  'signature': signature,
}
```

将'callback' : callback注释掉，这样就关闭了上传回调的开关，只提供服务端签名后直传的功能。

说明:
如果要开启回调，请保证callbackbody计算正确。

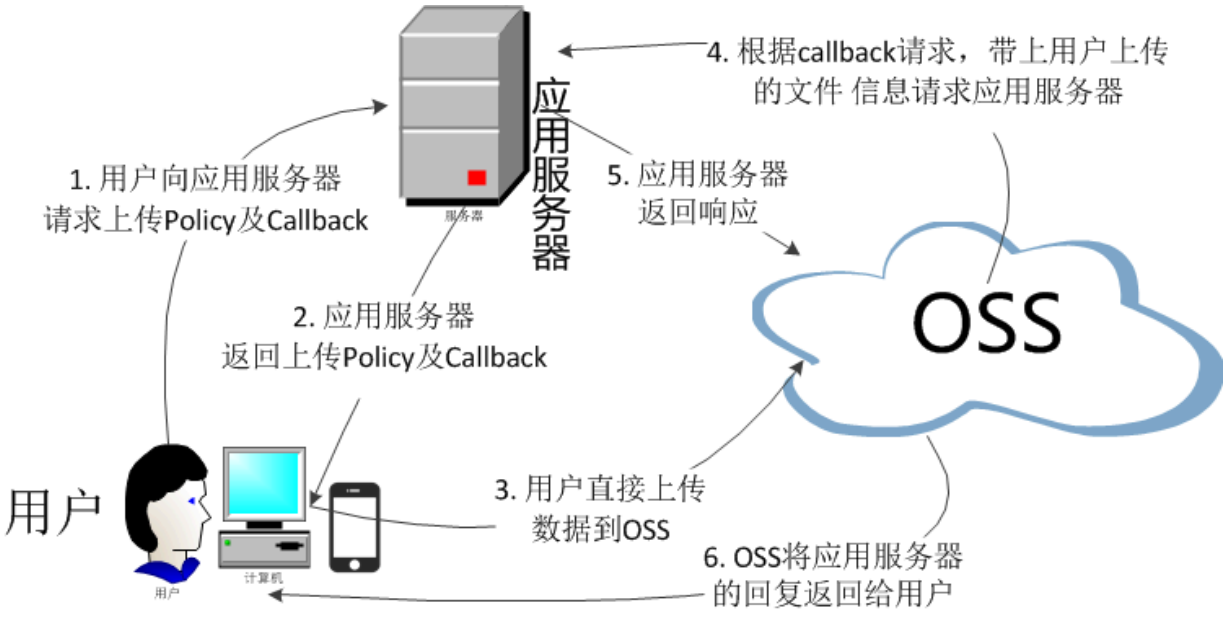
3.2.4 服务端签名直传并设置上传回调

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。

操作示例

服务端签名直传并设置上传回调提供了以下语言的操作示例：

- PHP
- Java
- Python
- Go
- Ruby

流程解析

以下根据流程讲解核心代码和消息内容。

1. 用户向应用服务器请求上传Policy和回调。

   在客户端源码中的`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'
   ```

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。

   下面是签名直传服务返回给客户端消息body内容的示例，这个body的内容将作为客户端上传文件的重要参数。

   ```
   {
   "accessid":"6MKO******4AUk44",
   ```

```
"host":"http://post-test.oss-cn-hangzhou.aliyuncs.com",
"policy":"eyJleHBpcmF0aW9uIjoiMjAxNS0xMS0wNVQyMDo1Mjoy******
Jjdb25kaXRpb25zIjpbWyJjdb250ZW50LWxlbmd0aC1yYW5nZSIsMCwxMDQ4
NTc2MDAwXSxbInN0YXJ0cy13aXRoIiwiJGtleSIsInVzZXItZGlyXC8iXV19",
"signature":"VsxOcOudx******z93CLaXPz+4s=",
"expire":1446727949,
"callback":"eyJjYWxsYmFja1VybCI6Imh0dHA6Ly9vc3MtZGVtby5hbGl5dW
5jcy5jdb206MjM0NTAiLCJjYWxsYmFja0hvc3QiOiJvc3MtZGVtby5hbGl5d
W5jcy5jdb20iLCJjYWxsYmFja0JvZHkiOiJmaWxlbmFtZT0ke29iamVjdH0m
c2l6ZT0ke3NpemV9Jm1pbWVUeXBlPSR7bWltZVR5cGV9JmhlaWdodD0ke2lt
YWdlSW5mby5oZWlnaHR9JndpZHRoPSR7aW1hZ2VJdbmZvLndpZHRofSIsImN
hbGxiYWNrQm9keVR5cGUiOiJhcHBsaWNhdGlvbi94LXd3dy1mb3JtLXVybGV
uY29kZWQifQ==",
"dir":"user-dirs/"
}
```

上述示例的callback内容采用的是base64编码。经过base64解码后的内容如下：

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

说明：

回调方式并不是固定的，以上仅为示例，您可以修改服务端代码自行设置回调。

内容解析如下：

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

3. 用户直接向OSS发送文件上传请求。

   在客户端源码`upload.js`文件中，`callbackbody`的值是步骤2中应用服务器返回给客户端消息body中callback的内容。

```
new_multipart_params = {
     'key' : 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
     'callback':   callbackbody,
     'signature': signature,
```

```
};
```

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。消息内容大致如下：

```
Hypertext Transfer Protocol
    POST / HTTP/1.1\r\n
    Host: 47.97.168.53\r\n
    Connection: close\r\n
    Content-Length: 76\r\n
    Authorization: fsNxFF0w******MNAoFb//a8x6v2lI1******h3nFUDALgk
u9bhC+cWQsnxuCo******tBUmnDI6k1PofggA4g==\r\n
    Content-MD5: eiEMyp7lbL8KStPBzMdr9w==\r\n
    Content-Type: application/x-www-form-urlencoded\r\n
    Date: Sat, 15 Sep 2018 10:24:12 GMT\r\n
    User-Agent: aliyun-oss-callback\r\n
    x-oss-additional-headers: \r\n
    x-oss-bucket: signedcallback\r\n
    x-oss-owner: 15447******20439\r\n
    x-oss-pub-key-url: aHR0cHM6Ly9nb3NzcHVibGljLm******bi5jb20vY2
FsbGJhY2tfcHViX2tleV92MS5wZW0=\r\n
    x-oss-request-id: 5B9CDDCC9******88AE2BEA1\r\n
    x-oss-requester: 15447******20439\r\n
    x-oss-signature-version: 1.0\r\n
    x-oss-tag: CALLBACK\r\n
    eagleeye-rpcid: 0.1\r\n
    \r\n
    [Full request URI: http://47.xx.xx.53/]
    [HTTP request 1/1]
    [Response in frame: 39]
    File Data: 76 bytes
HTML Form URL Encoded: application/x-www-form-urlencoded
    Form item: "filename" = ".snappython.png"
    Form item: "size" = "6014"
    Form item: "mimeType" = "image/png"
    Form item: "height" = "221"
```

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
Jjxb25kaXRpb25zIjpbWyJjcb250ZW50LWxlbmd0aC1yYW5nZSIsMCwxMDQ4
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'
```

3.2.5 服务端签名直传并设置上传回调

3.2.5.1 Go

本文以Golang语言为例，讲解在服务端通过Golang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。

前提条件

- 应用服务器对应的域名可通过公网访问。
- 应用服务器已经安装Golang 1.6以上版本（执行命令`go version`进行验证）。
- PC端浏览器支持JavaScript。

步骤 1：配置应用服务器

1. 下载应用服务器源码（Go版本）到应用服务器的目录下。下载地址： aliyun-oss-appserver-go-master.zip
2. 以Ubuntu 16.04为例，将源码放置到`/home/aliyun/aliyun-oss-appserver-go`目录下。
3. 进入该目录，打开源码文件`appserver.go`，修改以下代码片段：

```
// 请填写您的AccessKeyId。
var accessKeyId string = "<yourAccessKeyId>"

// 请填写您的AccessKeySecret。
var accessKeySecret string = "<yourAccessKeySecret>"

// host的格式为 bucketname.endpoint，请替换为您的真实信息。
var host string = "http://bucket-name.oss-cn-hangzhou.aliyuncs.com'"

// callbackUrl 为上传回调服务器的URL，请将下面的IP和Port配置为您自己的真实信息。
var callbackUrl string = "http://88.88.88.88:8888";

// 上传文件时指定的前缀。
var upload_dir string = "user-dir-prefix/"

// 上传策略Policy的失效时间，单位为秒。
var expire_time int64 = 30
```

- accessKeyId：设置您的AccessKeyId。
- accessKeySecret：设置您的AessKeySecret。
- host：格式为`bucketname.endpoint`，例如`bucket-name.oss-cn-hangzhou.aliyuncs.com`。关于Endpoint的介绍，请参见Endpoint访问域名。
- callbackUrl：设置上传回调URL，即回调服务器地址，用于处理应用服务器与OSS之前的通信。OSS会在文件上传完成后，把文件上传信息通过此回调URL发送给应用服务器。本例中修改为：

  `var callbackUrl string="http://11.22.33.44:1234";`
- upload_dir：指定上传文件的前缀，以免与其他文件发生冲突，您也可以填写空值。

步骤 2：配置客户端

1. 下载客户端源码到PC侧的本地目录。下载地址：aliyun-oss-appserver-js-master.zip
2. 以D:\aliyun\aliyun-oss-appserver-js目录为例。进入该目录，打开upload.js文件，找到下面的代码语句：

```
// 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. 将变量severUrl修改为应用服务器的地址。如本例中修改为：

```
// 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://11.22.33.44:1234'
```

步骤 3： 修改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：体验上传回调

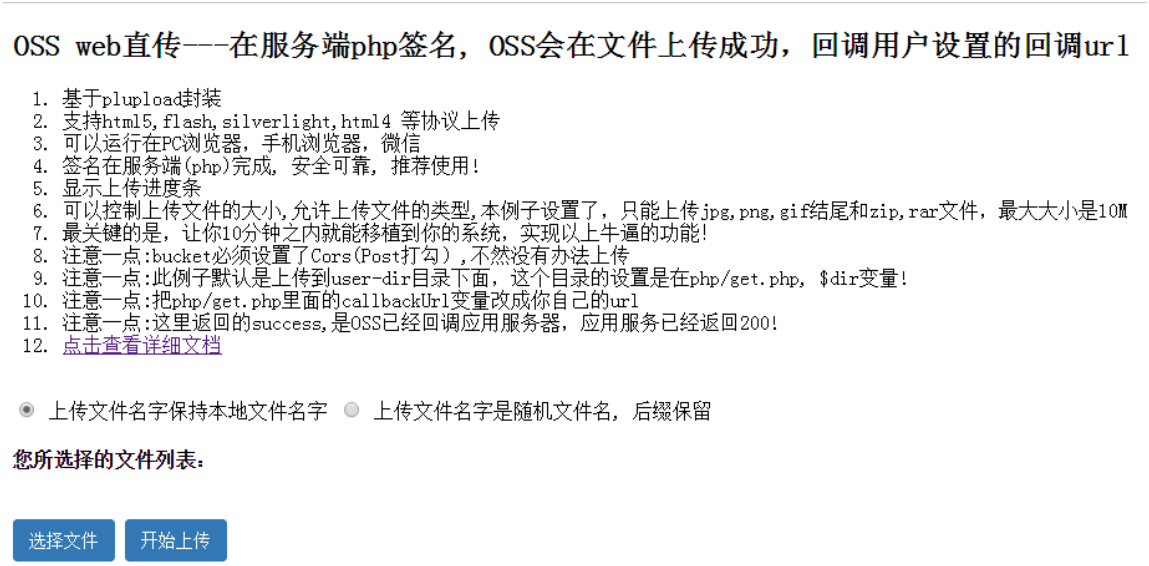
1. **启动应用服务器。**

   在`/home/aliyun/aliyun-oss-appserver-go`目录下，执行Go命令：

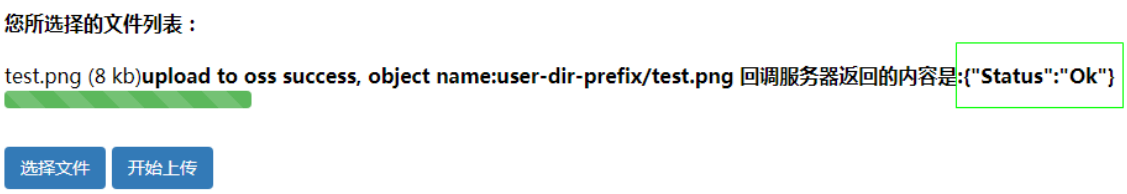
   ```
   go run appserver.go 11.22.33.44 1234
   ```

2. 启动客户端。

a. 在PC侧的客户端源码目录中，打开`index.html`文件。



b. 单击选择文件，选择指定类型的文件之后，单击开始上传。上传成功后，显示回调服务器返回的内容。



应用服务器核心代码解析

应用服务器源码包含了签名直传服务和上传回调服务两个功能。

- 签名直传服务

签名直传服务响应客户端发送给应用服务器的GET消息，代码片段如下：

```
func handlerRequest(w http.ResponseWriter, r *http.Request) {
        if (r.Method == "GET") {
                response := get_policy_token()
                w.Header().Set("Access-Control-Allow-Methods", "POST
")
                w.Header().Set("Access-Control-Allow-Origin", "*")
                io.WriteString(w, response)
  }
```

- 上传回调服务

上传回调服务响应OSS发送给应用服务器的POST消息，代码片段如下：

```
if (r.Method == "POST") {
                fmt.Println("\nHandle Post Request ... ")
```

```
                // Get PublicKey bytes
                bytePublicKey, err := getPublicKey(r)
                if (err != nil) {
                        responseFailed(w)
                        return
                }

                // Get Authorization bytes : decode from Base64Stri
ng
                byteAuthorization, err := getAuthorization(r)
                if (err != nil) {
                        responseFailed(w)
                        return
                }

                // Get MD5 bytes from Newly Constructed Authrization
 String.
                byteMD5, err := getMD5FromNewAuthString(r)
                if (err != nil) {
                        responseFailed(w)
                        return
                }

                // verifySignature and response to client
                if (verifySignature(bytePublicKey, byteMD5,
byteAuthorization)) {
                        // do something you want accoding to
callback_body ...

                        responseSuccess(w)  // response OK : 200
                } else {
                        responseFailed(w)   // response FAILED : 400

                }
        }
```

详情请参见API文档Callback-回调签名。

3.2.5.2 PHP

本文以PHP语言为例，讲解在服务端通过PHP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。

前提条件

- Web服务器已部署。
- Web服务器对应的域名可通过公网访问。
- Web服务器能够解析PHP（执行命令`php -v`进行查看）。
- PC端浏览器支持JavaScript。

步骤 1：配置Web服务器

本文档以Ubuntu16.04和Apache2.4.18为例。本示例中的环境配置如下：

- Web服务器外网IP地址为`11.22.33.44`。您可以在配置文件`/etc/apache2/apache2.conf`中增加`ServerName 11.22.33.44`来进行修改。

- Web服务器的监听端口为8080。您可以在配置文件/etc/apache2/ports.conf中进行修改相关内容Listen 8080。
- 确保Apache能够解析PHP文件：sudo apt-get install libapache2-mod-php5（其他平台请根据实际情况进行安装配置）。

您可以根据自己的实际环境修改IP地址和监听端口。更新配置后，需要重启Apache服务器，重启命令为/etc/init.d/apache2 restart。

步骤 2：配置应用服务器

- 下载部署应用服务器源码

  下载应用服务器源码（PHP版本）：aliyun-oss-appserver-php-master.zip

  下载应用服务器源码后，将其部署到应用服务器的相应目录。本文示例中需要部署到Ubuntu16.04的/var/www/html/aliyun-oss-appserver-php目录下。

  在PC侧浏览器中访问应用服务器URL http://11.22.33.44:8080/aliyun-oss-appserver-php/index.html， 显示的页面内容与测试样例主页相同则验证通过。

- 开启Apache捕获HTTP头部Authorization字段的功能

  您的应用服务器收到的回调请求有可能没有Authotization头，这是因为有些Web应用服务器会将Authorization头自行解析掉。比如Apache2，需要设置成不解析头部。

  - 打开Apache2配置文件/etc/apache2/apache2.conf，找到如下片段，进行相应修改：

    ```
    <Directory /var/www/>
            Options Indexes FollowSymLinks
            AllowOverride All
            Require all granted
    </Directory>
    ```

  - 在/var/www/html/aliyun-oss-appserver-php目录下，新建一个.htaccess文件，填写如下内容：

    ```
    <IfModule mod_rewrite.c>
    RewriteEngine on
    RewriteCond %{HTTP:Authorization} .
    RewriteRule .* - [E=HTTP_AUTHORIZATION:%{HTTP:Authorization}]
    </IfModule>
    ```

  其他Web服务器或其他Apache版本，请根据实际情况进行配置。

- 修改配置应用服务器

  在/var/www/html/aliyun-oss-appserver-php/php目录下打开文件get.php， 修改如下代码片段：

  ```
  $id= '<yourAccessKeyId>';           // 请填写您的AccessKeyId。
  ```

```
    $key= '<yourAccessKeySecret>';      // 请填写您的AccessKeySecret。

    // $host的格式为 bucketname.endpointx， 请替换为您的真实信息。
    $host = 'http://bucket-name.oss-cn-hangzhou.aliyuncs.com';

    // $callbackUrl为上传回调服务器的URL， 请将下面的IP和Port配置为您自己的
真实URL信息。
    $callbackUrl = 'http://88.88.88.88:8888/aliyun-oss-appserver-php
/php/callback.php';

    $dir = 'user-dir-prefix/';           // 上传文件时指定的前缀。
```

- $id： 设置您的AccessKeyId。
- $key： 设置您的AccessKeySecret。
- $host： 格式为`BucketName.Endpoint`，例如`bucket-name.oss-cn-hangzhou.aliyuncs.com`。关于Endpoint的介绍，请参见Endpoint访问域名。
- $callbackUrl：设置上传回调URL，即回调服务器地址，用于处理应用服务器与OSS之前的通信。OSS会在文件上传完成后，把文件上传信息通过此回调URL发送给应用服务器。本例中修改为：

  `$callbackUrl='http://11.22.33.44:8080/aliyun-oss-appserver-php/php/callback.php';`
- $dir： 设置上传到OSS文件的前缀，以便区别于其他文件从而避免冲突，您也可以填写空值。

步骤 3：配置客户端

在应用服务器的`/var/www/html/aliyun-oss-appserver-php`目录下修改文件`upload.js`：

对于PHP版本的应用服务器源码，一般不需要修改文件`upload.js`内容的，因为相对路径也是可以正常工作的。

如果确实需要修改，请找到如下的代码片段：

```
serverUrl ='./php/get.php'
```

将变量`severUrl`改成服务器部署的地址，用于处理浏览器和应用服务器之间的通信。

比如本示例中可以如下修改：

```
serverUrl ='http://11.22.33.44:8080/aliyun-oss-appserver-php/php/get.php
'
```

步骤 4：修改CORS

从浏览器向OSS发出的请求消息带有`Origin`的消息头，OSS对带有`Origin`头的请求消息会首先进行跨域规则的验证。

即客户端进行表单直接上传到OSS会产生跨域请求，需要为Bucket设置跨域规则（CORS），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 5：体验上传回调

在PC侧的Web浏览器中输入`http://11.22.33.44:8080/aliyun-oss-appserver-php/index.html`。

单击选择文件，选择指定类型的文件后，单击开始上传。上传成功后，显示回调服务器返回的内容。

应用服务器核心代码解析

应用服务器源码包含了签名直传服务和上传回调服务两个功能。

- 签名直传服务

  签名直传服务响应客户端发送给应用服务器的GET消息，代码文件是aliyun-oss-appserver-php/php/get.php。代码片段如下：

```
$response = array();
$response['accessid'] = $id;
$response['host'] = $host;
$response['policy'] = $base64_policy;
$response['signature'] = $signature;
$response['expire'] = $end;
$response['callback'] = $base64_callback_body;
$response['dir'] = $dir;
```

- 上传回调服务

  上传回调服务响应OSS发送给应用服务器的POST消息，代码文件是aliyun-oss-appserver-php/php/callback.php。

  代码片段如下：

```
// 6.验证签名
$ok = openssl_verify($authStr, $authorization, $pubKey, OPENSSL_ALGO_MD5);
if ($ok == 1)
{
    header("Content-Type: application/json");
    $data = array("Status"=>"Ok");
    echo json_encode($data);
}
```

  详情请参见API文档Callback-回调签名。

3.2.5.3 Java

本文以Java语言为例，讲解在服务端通过Java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。

前提条件

- 应用服务器对应的域名可通过公网访问。
- 确保应用服务器已经安装Java 1.6以上版本（执行命令java -version进行查看）。
- 确保PC端浏览器支持JavaScript。

步骤 1：配置应用服务器

下载应用服务器源码(Java版本)：aliyun-oss-appserver-java-master.zip

将源码下载到应用服务器的硬盘，本示例中以Ubuntu 16.04为例，放置到/home/aliyun/aliyun-oss-appserver-java目录下。进入该目录，找到并打开源码文件CallbackServer.java，修改如下的代码片段：

```
String accessId = "<yourAccessKeyId>";       // 请填写您的AccessKeyId。
String accessKey = "<yourAccessKeySecret>"; // 请填写您的AccessKeySecret。
String endpoint = "oss-cn-hangzhou.aliyuncs.com"; // 请填写您的 endpoint。
String bucket = "bucket-name";                      // 请填写您的 bucketname 。
String host = "https://" + bucket + "." + endpoint; // host的格式为 bucketname.endpoint

// callbackUrl为上传回调服务器的URL，请将下面的IP和Port配置为您自己的真实信息。
String callbackUrl = "http://88.88.88.88:8888";
String dir = "user-dir-prefix/"; // 用户上传文件时指定的前缀。
```

- accessId： 设置您的AccessKeyId。
- accessKey： 设置您的AessKeySecret。
- host：格式为bucketname.endpoint，例如bucket-name.oss-cn-hangzhou.aliyuncs.com。关于Endpoint的介绍，请参见Endpoint访问域名。
- callbackUrl：设置上传回调URL，即回调服务器地址，用于处理应用服务器与OSS之前的通信。OSS会在文件上传完成后，把文件上传信息通过此回调URL发送给应用服务器。本例中修改为：

  `String callbackUrl ="http://11.22.33.44:1234";`
- dir： 设置上传到OSS文件的前缀，以便区别于其他文件从而避免冲突，您也可以填写空值。

步骤 2：配置客户端

下载客户端源码：aliyun-oss-appserver-js-master.zip

下载客户端源码到PC侧的本地硬盘。本例中以D:\aliyun\aliyun-oss-appserver-js目录为例。

进入该目录，打开upload.js文件，找到下面的代码语句：

```
// 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'
```

将变量severUrl改成应用服务器的地址，客户端可以通过它获取签名直传Policy等信息。如本例中可修改为：

```
// 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://11.22.33.44:1234'
```

步骤 3： 修改CORS

从浏览器向OSS发出的请求消息带有Origin的消息头，OSS对带有Origin头的请求消息会首先进行跨域规则的验证。

即客户端进行表单直接上传到OSS会产生跨域请求，需要为Bucket设置跨域规则（CORS），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：体验上传回调

- 启动应用服务器

  在/home/aliyun/aliyun-oss-appserver-java目录下，执行mvn package命令编译打包，然后执行命令启动应用服务器：

  ```
  java -jar target/appservermaven-1.0.0.jar 1234
  ```

  > 说明：
  > 请将 1234 改成配置应用服务器时的Port。

  您也可以在PC端使用Eclipse/Intellij IDEA等IDE工具导出jar包，然后将jar包拷贝到应用服务器，再执行jar包启动应用服务器。

- 启动客户端

  在PC侧的客户端源码目录中，打开index.html 文件。

  单击选择文件，选择指定类型的文件，单击开始上传。上传成功后，显示回调服务器返回的内容。

应用服务器核心代码解析

应用服务器源码包含了签名直传服务和上传回调服务两个功能。

- 签名直传服务

  签名直传服务响应客户端发送给应用服务器的GET消息，代码片段如下：

```
protected void doGet(HttpServletRequest request, HttpServletResponse
 response)
   throws ServletException, IOException {

  String accessId = "<yourAccessKeyId>"; // 请填写您的AccessKeyId。
  String accessKey = "<yourAccessKeySecret>"; // 请填写您的AccessKeyS
ecret。
  String endpoint = "oss-cn-hangzhou.aliyuncs.com"; // 请填写您的
endpoint。
  String bucket = "bucket-name"; // 请填写您的 bucketname 。
  String host = "https://" + bucket + "." + endpoint; // host的格式
为 bucketname.endpoint
  // callbackUrl为 上传回调服务器的URL，请将下面的IP和Port配置为您自己的真实
信息。
  String callbackUrl = "http://88.88.88.88:8888";
  String dir = "user-dir-prefix/"; // 用户上传文件时指定的前缀。

  OSSClient client = new OSSClient(endpoint, accessId, accessKey);
  try {
   long expireTime = 30;
   long expireEndTime = System.currentTimeMillis() + expireTime *
1000;
   Date expiration = new Date(expireEndTime);
   PolicyConditions policyConds = new PolicyConditions();
   policyConds.addConditionItem(PolicyConditions.COND_CONTE
NT_LENGTH_RANGE, 0, 1048576000);
   policyConds.addConditionItem(MatchMode.StartWith, PolicyCond
itions.COND_KEY, dir);

   String postPolicy = client.generatePostPolicy(expiration,
policyConds);
   byte[] binaryData = postPolicy.getBytes("utf-8");
   String encodedPolicy = BinaryUtil.toBase64String(binaryData);
   String postSignature = client.calculatePostSignature(postPolicy);

   Map<String, String> respMap = new LinkedHashMap<String, String
>();
   respMap.put("accessid", accessId);
   respMap.put("policy", encodedPolicy);
   respMap.put("signature", postSignature);
   respMap.put("dir", dir);
   respMap.put("host", host);
   respMap.put("expire", String.valueOf(expireEndTime / 1000));
   // respMap.put("expire", formatISO8601Date(expiration));

   JSONObject jasonCallback = new JSONObject();
   jasonCallback.put("callbackUrl", callbackUrl);
   jasonCallback.put("callbackBody",
     "filename=${object}&size=${size}&mimeType=${mimeType}&height=${
imageInfo.height}&width=${imageInfo.width}");
   jasonCallback.put("callbackBodyType", "application/x-www-form-
urlencoded");
   String base64CallbackBody = BinaryUtil.toBase64String(jasonCallb
ack.toString().getBytes());
   respMap.put("callback", base64CallbackBody);

   JSONObject ja1 = JSONObject.fromObject(respMap);
   // System.out.println(ja1.toString());
```

```
    response.setHeader("Access-Control-Allow-Origin", "*");
    response.setHeader("Access-Control-Allow-Methods", "GET, POST");
    response(request, response, ja1.toString());

  } catch (Exception e) {
    // Assert.fail(e.getMessage());
    System.out.println(e.getMessage());
  }
 }
```

- **上传回调服务**

  上传回调服务响应OSS发送给应用服务器的POST消息，代码片段如下：

```
protected boolean VerifyOSSCallbackRequest(HttpServletRequest
request, String ossCallbackBody)
    throws NumberFormatException, IOException {
  boolean ret = false;
  String autorizationInput = new String(request.getHeader("
Authorization"));
  String pubKeyInput = request.getHeader("x-oss-pub-key-url");
  byte[] authorization = BinaryUtil.fromBase64String(autorizati
onInput);
  byte[] pubKey = BinaryUtil.fromBase64String(pubKeyInput);
  String pubKeyAddr = new String(pubKey);
  if (!pubKeyAddr.startsWith("https://gosspublic.alicdn.com/")
    && !pubKeyAddr.startsWith("https://gosspublic.alicdn.com/")) {
   System.out.println("pub key addr must be oss addrss");
   return false;
  }
  String retString = executeGet(pubKeyAddr);
  retString = retString.replace("-----BEGIN PUBLIC KEY-----", "");
  retString = retString.replace("-----END PUBLIC KEY-----", "");
  String queryString = request.getQueryString();
  String uri = request.getRequestURI();
  String decodeUri = java.net.URLDecoder.decode(uri, "UTF-8");
  String authStr = decodeUri;
  if (queryString != null && !queryString.equals("")) {
   authStr += "?" + queryString;
  }
  authStr += "\n" + ossCallbackBody;
  ret = doCheck(authStr, authorization, retString);
  return ret;
 }

 protected void doPost(HttpServletRequest request, HttpServle
tResponse response)
    throws ServletException, IOException {
  String ossCallbackBody = GetPostBody(request.getInputStream(),
     Integer.parseInt(request.getHeader("content-length")));
  boolean ret = VerifyOSSCallbackRequest(request, ossCallbackBody);
  System.out.println("verify result : " + ret);
  // System.out.println("OSS Callback Body:" + ossCallbackBody);
  if (ret) {
   response(request, response, "{\"Status\":\"OK\"}", HttpServle
tResponse.SC_OK);
  } else {
   response(request, response, "{\"Status\":\"verdify not ok\"}",
HttpServletResponse.SC_BAD_REQUEST);
  }
```

```
}
```

详情请参见API文档Callback–回调签名。

3.2.5.4 Python

本文以Python语言为例，讲解在服务端通过Python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。

前提条件

- 应用服务器对应的域名可通过公网访问。
- 确保应用服务器已经安装Python 2.6以上版本（执行命令python --version进行查看）。
- 确保PC端浏览器支持JavaScript。

步骤 1：配置应用服务器

下载应用服务器源码(Python版本)：aliyun-oss-appserver-python-master.zip 。

将源码下载到应用服务器的硬盘，本示例中以Ubuntu 16.04为例，放置到/home/aliyun/aliyun-oss-appserver-python目录下。进入该目录，打开源码文件appserver.py，修改如下的代码片段：

```
# 请填写您的AccessKeyId。
access_key_id = '<yourAccessKeyId>'
# 请填写您的AccessKeySecret。
access_key_secret = '<yourAccessKeySecret>'
# host的格式为 bucketname.endpoint ，请替换为您的真实信息。
host = 'http://bucket-name.oss-cn-hangzhou.aliyuncs.com';
# callback_url为 上传回调服务器的URL，请将下面的IP和Port配置为您自己的真实URL信息。
callback_url = "http://88.88.88.88:8888";
# 用户上传文件时指定的前缀。
upload_dir = 'user-dir-prefix/'
```

- access_key_id： 设置您的AccessKeyId。
- access_key_secret： 设置您的AessKeySecret。
- host： 格式为bucketname.endpoint，例如bucket-name.oss-cn-hangzhou.aliyuncs.com。关于Endpoint的介绍，请参见Endpoint访问域名。
- callback_url： 设置上传回调URL，即回调服务器地址，用于处理应用服务器与OSS之前的通信。OSS会在文件上传完成后，把文件上传信息通过此回调URL发送给应用服务器。本例中修改为：

  ```
  var callbackUrl string ="http://11.22.33.44:1234";
  ```

- upload_dir： 指定上传文件的前缀，以便区别于其他文件以免发生冲突，您也可以填写空值。

步骤 2：配置客户端

下载客户端源码：aliyun-oss-appserver-js-master.zip 。

下载客户端源码到PC侧的本地硬盘。本例中以D:\aliyun\aliyun-oss-appserver-js目录为例。

进入该目录，打开upload.js文件，找到下面的代码语句：

```
// 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'
```

将变量severUrl改成应用服务器的地址，客户端可以通过它可以获取签名直传Policy等信息。如本例中可修改为：

```
// 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://11.22.33.44:1234'
```

步骤 3： 修改CORS

从浏览器向OSS发出的请求消息带有Origin的消息头，OSS对带有Origin头的请求消息会首先进行跨域规则的验证。

即客户端进行表单直接上传到OSS会产生跨域请求，需要为Bucket设置跨域规则（CORS），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：体验上传回调

- 启动应用服务器

  在/home/aliyun-oss-appserver-python目录下，执行命令启动应用服务器：

```
python appserver.py 11.22.33.44 1234
```

- 启动客户端

  在PC侧的客户端源码目录中，打开index.html 文件。

单击选择文件，选择指定类型的文件后，单击开始上传。上传成功后，显示回调服务器返回的内容。

应用服务器核心代码解析

应用服务器源码包含了签名直传服务和上传回调服务两个功能。

- 签名直传服务

  签名直传服务响应客户端发送给应用服务器的GET消息，代码片段如下：

```
def do_GET(self):
        print "********************* do_GET "
        token = get_token()
        self.send_response(200)
        self.send_header('Access-Control-Allow-Methods', 'POST')
```

```
        self.send_header('Access-Control-Allow-Origin', '*')
        self.send_header('Content-Type', 'text/html; charset=UTF-8')
        self.end_headers()
        self.wfile.write(token)
```

- **上传回调服务**

  上传回调服务响应OSS发送给应用服务器的POST消息，代码片段如下：

```
def do_POST(self):
        print "********************* do_POST "
        # get public key
        pub_key_url = ''
        try:
            pub_key_url_base64 = self.headers['x-oss-pub-key-url']
            pub_key_url = pub_key_url_base64.decode('base64')
            url_reader = urllib2.urlopen(pub_key_url)
            pub_key = url_reader.read()
        except:
            print 'pub_key_url : ' + pub_key_url
            print 'Get pub key failed!'
            self.send_response(400)
            self.end_headers()
            return

        # get authorization
        authorization_base64 = self.headers['authorization']
        authorization = authorization_base64.decode('base64')

        # get callback body
        content_length = self.headers['content-length']
        callback_body = self.rfile.read(int(content_length))

        # compose authorization string
        auth_str = ''
        pos = self.path.find('?')
        if -1 == pos:
            auth_str = self.path + '\n' + callback_body
        else:
            auth_str = urllib2.unquote(self.path[0:pos]) + self.path
[pos:] + '\n' + callback_body

        # verify authorization
        auth_md5 = md5.new(auth_str).digest()
        bio = BIO.MemoryBuffer(pub_key)
        rsa_pub = RSA.load_pub_key_bio(bio)
        try:
            result = rsa_pub.verify(auth_md5, authorization, 'md5')
        except e:
            result = False

        if not result:
            print 'Authorization verify failed!'
            print 'Public key : %s' % (pub_key)
            print 'Auth string : %s' % (auth_str)
            self.send_response(400)
            self.end_headers()
            return

        # do something accoding to callback_body

        # response to OSS
        resp_body = '{"Status":"OK"}'
```

```
        self.send_response(200)
        self.send_header('Content-Type', 'application/json')
        self.send_header('Content-Length', str(len(resp_body)))
        self.end_headers()
        self.wfile.write(resp_body)
```

详情请参见API文档Callback-回调签名。

3.2.5.5 Ruby

本文以Ruby语言为例，讲解在服务端通过Ruby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。

前提条件

- 应用服务器对应的域名可通过公网访问。
- 确保应用服务器已经安装Ruby 2.0以上版本（执行命令ruby -v进行查看）。
- 确保PC端浏览器支持JavaScript。

步骤 1：配置应用服务器

下载应用服务器源码(Ruby版本)：aliyun-oss-appserver-ruby-master.zip

将源码下载到应用服务器的硬盘，本示例中以Ubuntu 16.04为例，放置到/home/aliyun/aliyun-oss-appserver-ruby目录下。进入该目录，打开源码文件appserver.rb，修改如下代码片段：

```
# 请填写您的AccessKeyId。
$access_key_id = '<yourAccessKeyId>'

# 请填写您的AccessKeySecret。
$access_key_secret = '<yourAccessKeySecret>'

# $host的格式为 bucketname.endpoint ，请替换为您的真实信息。
$host = 'http://bucket-name.oss-cn-hangzhou.aliyuncs.com';

# $callbackUrl为 上传回调服务器的URL，请将下面的IP和Port配置为您自己的真实信息。
$callback_url = "http://88.88.88.88:8888";

# 用户上传文件时指定的前缀。
$upload_dir = 'user-dir-prefix/'
```

- $access_key_id： 设置您的AccessKeyId。
- $access_key_secret： 设置您的AessKeySecret。
- $host： 格式为bucketname.endpoint，例如bucket-name.oss-cn-hangzhou.aliyuncs.com。关于Endpoint的介绍，请参见Endpoint访问域名。

- $callback_url： 设置上传回调URL，即回调服务器地址，用于处理应用服务器与OSS之前的通信。OSS会在文件上传完成后，把文件上传信息通过此回调URL发送给应用服务器。本例中修改为：

  ```
  $callback_url="http://11.22.33.44:1234";
  ```

- $upload_dir： 设置上传到OSS文件的前缀，以便区别于其他文件从而避免冲突，您也可以填写空值。

步骤 2：配置客户端

下载客户端源码：aliyun-oss-appserver-js-master.zip

下载客户端源码到PC侧的本地硬盘。本例中以`D:\aliyun\aliyun-oss-appserver-js`目录为例。

进入该目录，打开`upload.js`文件，找到下面的代码语句：

```
// 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'
```

将变量`severUrl`改成应用服务器的地址，客户端可以通过它获取签名直传Policy等信息。如本例中可修改为：

```
// 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://11.22.33.44:1234'
```

步骤 3： 修改CORS

从浏览器向OSS发出的请求消息带有`Origin`的消息头，OSS对带有`Origin`头的请求消息会首先进行跨域规则的验证。

即客户端进行表单直接上传到OSS会产生跨域请求，需要为Bucket设置跨域规则（CORS），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：体验上传回调

- 启动应用服务器

  在/home/aliyun/aliyun-oss-appserver-ruby目录下，执行以下命令启动应用服务器：

```
ruby appserver.rb 11.22.33.44 1234
```

- 启动客户端

  在PC侧的客户端源码目录中，打开index.html 文件。

  单击选择文件，选择指定类型的文件后，单击开始上传。上传成功后，显示回调服务器返回的内容。

应用服务器核心代码解析

应用服务器源码包含了签名直传服务和上传回调服务两个功能。

- 签名直传服务

  签名直传服务响应客户端发送给应用服务器的GET消息，代码片段如下：

```
def get_token()
    expire_syncpoint = Time.now.to_i + $expire_time

    expire = Time.at(expire_syncpoint).utc.iso8601()
    response.headers['expire'] = expire
```

```
    policy_dict = {}
    condition_arrary = Array.new
    array_item = Array.new
    array_item.push('starts-with')
    array_item.push('$key')
    array_item.push($upload_dir)
    condition_arrary.push(array_item)
    policy_dict["conditions"] = condition_arrary
    policy_dict["expiration"] = expire
    policy = hash_to_jason(policy_dict)
    policy_encode = Base64.strict_encode64(policy).chomp;
    h = OpenSSL::HMAC.digest('sha1', $access_key_secret, policy_enc
ode)
    hs = Digest::MD5.hexdigest(h)
    sign_result = Base64.strict_encode64(h).strip()

    callback_dict = {}
    callback_dict['callbackBodyType'] = 'application/x-www-form-
urlencoded';
    callback_dict['callbackBody'] = 'filename=${object}&size=${size
}&mimeType=${mimeType}&height=${imageInfo.height}&width=${imageInfo.
width}';
    callback_dict['callbackUrl'] = $callback_url;
    callback_param = hash_to_jason(callback_dict)
    base64_callback_body = Base64.strict_encode64(callback_param);

    token_dict = {}
    token_dict['accessid'] = $access_key_id
    token_dict['host'] = $host
    token_dict['policy'] = policy_encode
    token_dict['signature'] = sign_result
    token_dict['expire'] = expire_syncpoint
    token_dict['dir'] = $upload_dir
    token_dict['callback'] = base64_callback_body
    response.headers["Access-Control-Allow-Methods"] = "POST"
    response.headers["Access-Control-Allow-Origin"] = "*"
    result = hash_to_jason(token_dict)

    result
end

get '/*' do
  puts "********************* GET "
  get_token()
end
```

- **上传回调服务**

  上传回调服务响应OSS发送给应用服务器的POST消息，代码片段如下：

```
post '/*' do
  puts "********************* POST"
  pub_key_url = Base64.decode64(get_header('x-oss-pub-key-url'))
  pub_key = get_public_key(pub_key_url)
  rsa = OpenSSL::PKey::RSA.new(pub_key)

  authorization = Base64.decode64(get_header('authorization'))
  req_body = request.body.read
  if request.query_string.empty? then
    auth_str = CGI.unescape(request.path) + "\n" + req_body
  else
```

```
    auth_str = CGI.unescape(request.path) + '?' + request.query_stri
ng + "\n" + req_body
  end

  valid = rsa.public_key.verify(
    OpenSSL::Digest::MD5.new, authorization, auth_str)

  if valid
    #body({'Status' => 'OK'}.to_json)
    body(hash_to_jason({'Status' => 'OK'}))
  else
    halt 400, "Authorization failed!"
  end
end
```

详情请参见API文档Callback-回调签名。

3.3 小程序直传实践

本文介绍如何在微信小程序环境下将文件上传到 OSS。

说明:

在支付宝小程序环境下上传文件到 OSS，请参见#unique_81。

背景

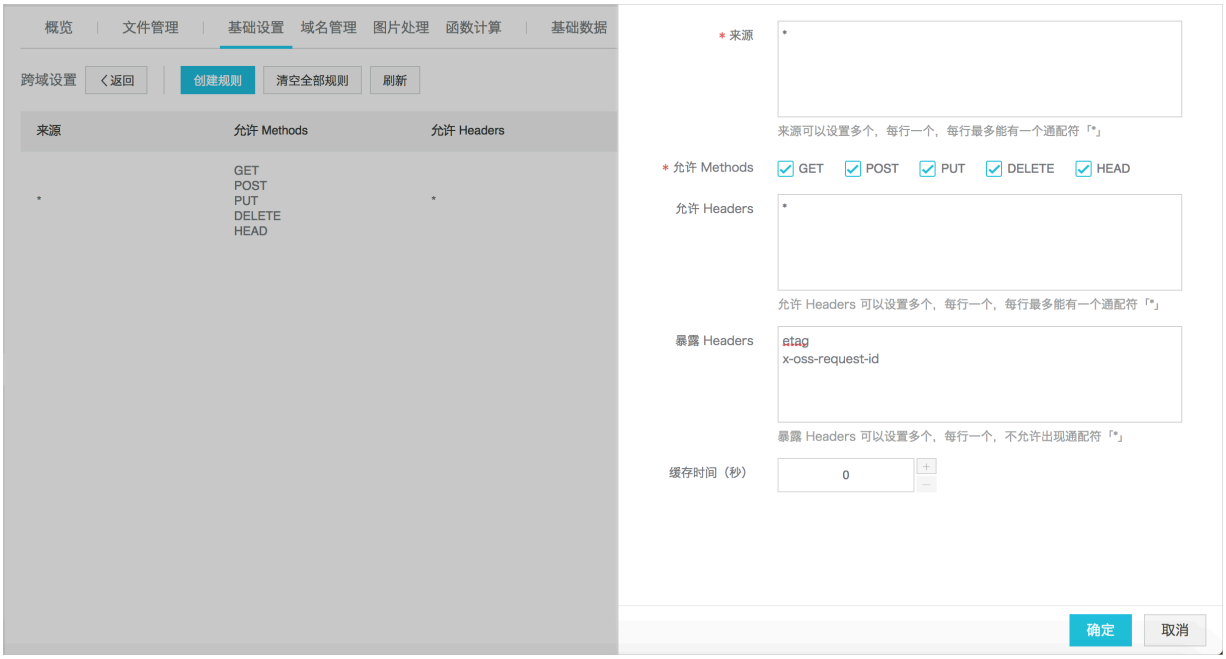
小程序是当下比较流行的移动应用，例如大家熟知的微信小程序、支付宝小程序等。它是一种全新的开发模式，无需下载和安装，因而可以被便捷地获取和传播，为终端用户提供更优的用户体验。如何在小程序环境下上传文件到 OSS 也成为开发者比较关心的一个问题。

与JavaScript客户端直传实践的原理相同，小程序上传文件到 OSS 也是利用 OSS 提供的 PostObject 接口来实现表单文件上传到 OSS。关于 PostObject 的详细介绍请参见 API 文档 PostObject。

步骤 1： 配置 Bucket 跨域

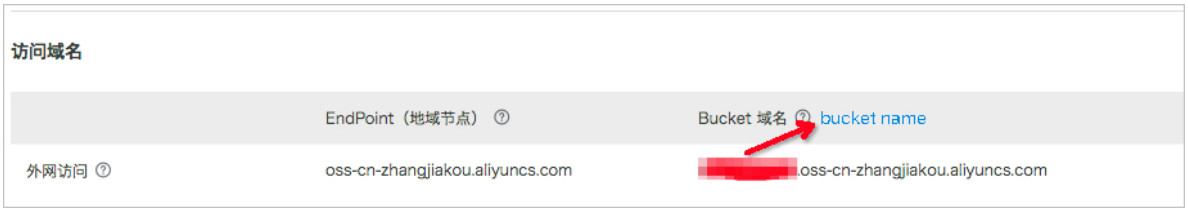
客户端进行表单直传到 OSS 时，会从浏览器向 OSS 发送带有 Origin 的请求消息。OSS 对带有 Origin 头的请求消息会进行跨域规则（CORS）的验证。因此需要为 Bucket 设置跨域规则以支持 Post 方法。

具体操作步骤请参见设置跨域访问。

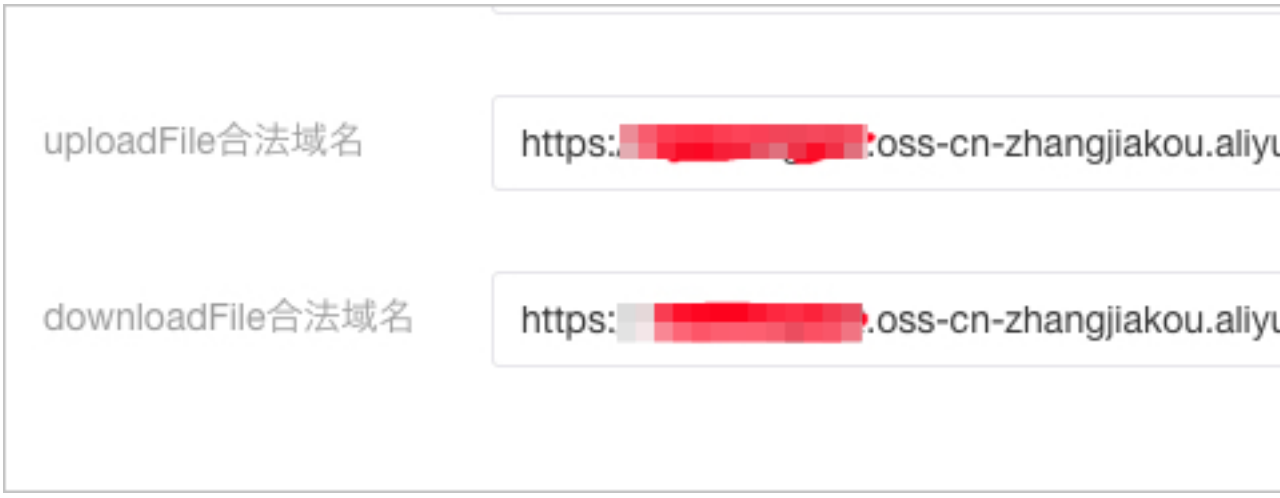


步骤 2：配置外网域名到小程序的上传域名白名单中

1. **通过 OSS 控制台查看外网域名。**



2. **登录微信小程序平台，配置小程序的上传域名白名单。**

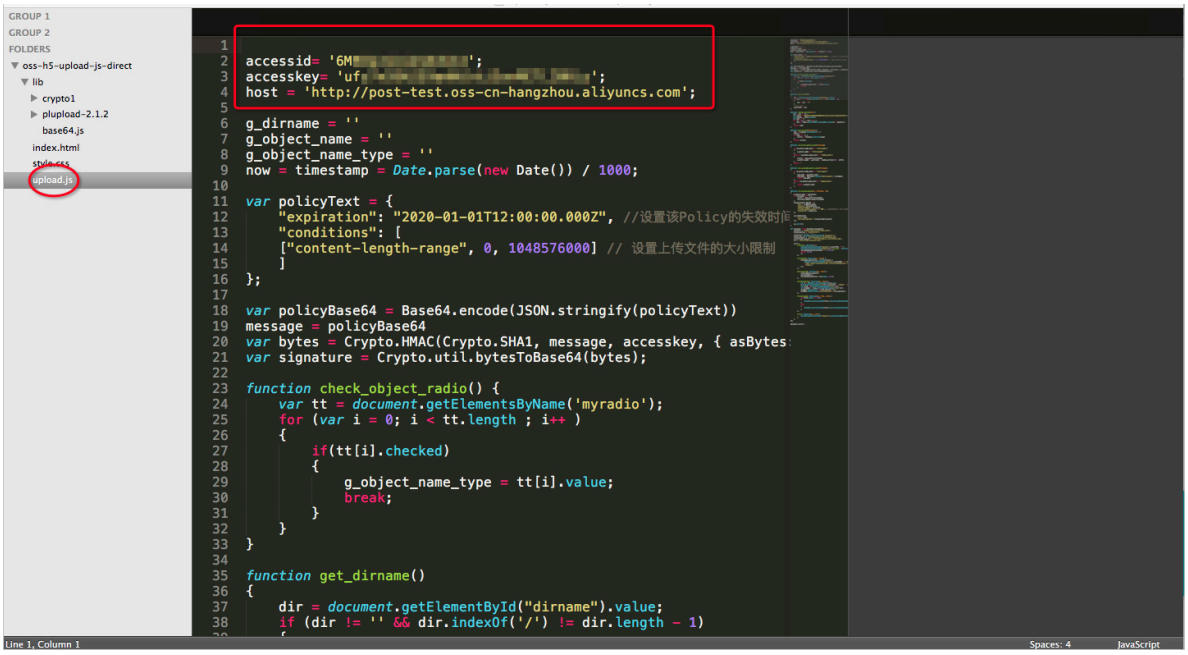


说明：

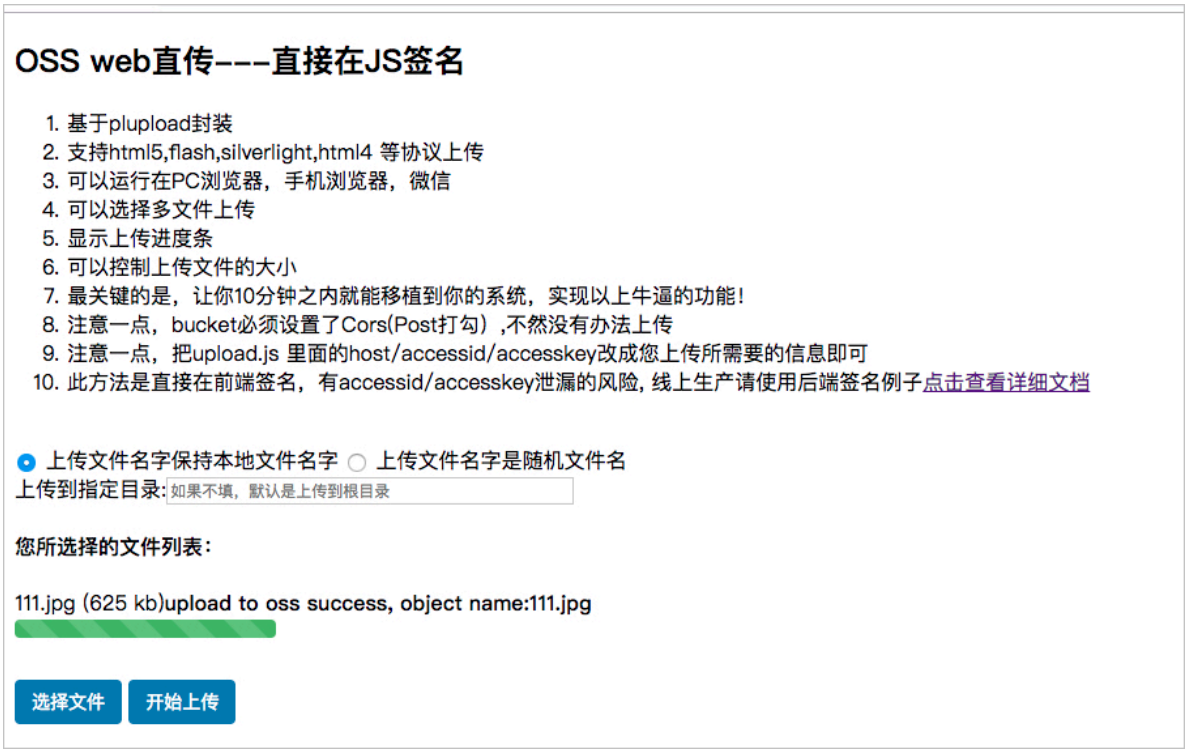
实际业务中，您需要将 OSS 提供的外网域名和您自己的域名进行绑定。具体操作请参见 OSS 域名绑定。

步骤 3：使用 Web 端直传实践方案 Demo 进行上传测试

1. 下载应用服务器代码。下载地址
2. 修改 Demo 中 upload.js 的密钥和地址。



3. 进行上传测试。



4. 获取上传需要的签名（signature）和加密策略（policy）。

   关于 signature 的获取可参考 upload.js 的代码；关于 policy 的格式可参考 upload.js代码 policyText。

说明:

小程序上传主要是计算signature和policy，具体实践中建议用户在服务端进行相关签名的计算后返回给小程序，现提供node版本计算signature和policy的参考代码。

步骤 4：使用微信小程序上传图片

使用 chooseImage API 进行图片选择，然后调用 uploadFile 进行文件上传。参考代码如下：

```
wx.chooseImage({
  count:1,
  sizeType: ['compressed'],
  sourceType: ['album'],
  success: function(res) {
    var imageSrc = res.tempFilePaths[0];
    wx.uploadFile({
      url: uploadFileUrl,
      filePath: imageSrc,
      name: 'file',
      formData: {
        name: res.tempFilePaths[0],
        key: "${filename}",
        policy: "xxxxxxxxxxxxxxxxxx",
        OSSAccessKeyId: "xxxxxxxxxxxxxxxx",
        success_action_status: "200",
        signature: "xxxxxxxxxxxxxxxxxxxxxx"
      },
      success: function(res) {
        wx.showToast({
          title: '上传成功',
          icon: 'success',
          duration: 1000
        })

        self.setData({
          imageSrc
        })
      },
      fail: function(errMsg) {
        console.log(errMsg)
      }
    })
  }
})
```

说明:

- 上述示例中，`name`字段为`file`字段。
- 更多内容请参见#unique_83。

4 移动应用端直传实践

4.1 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。

背景信息

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：

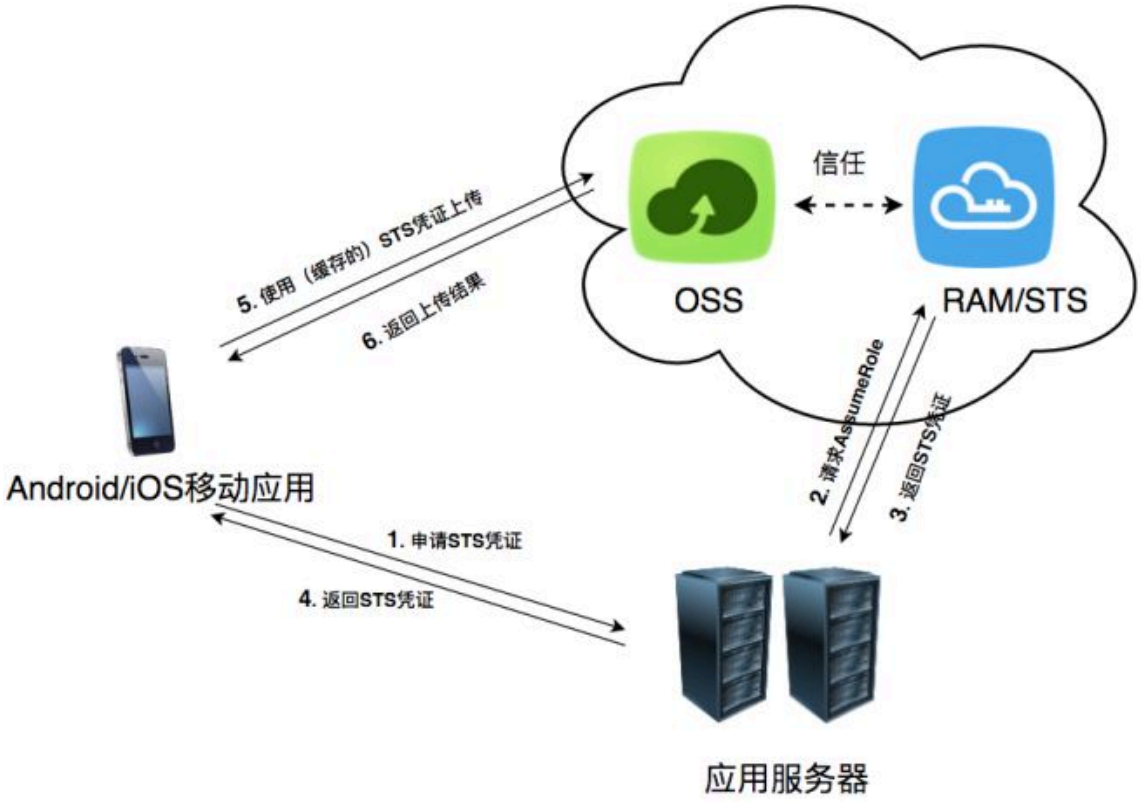
- Android：下载地址
- iOS：下载地址

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。

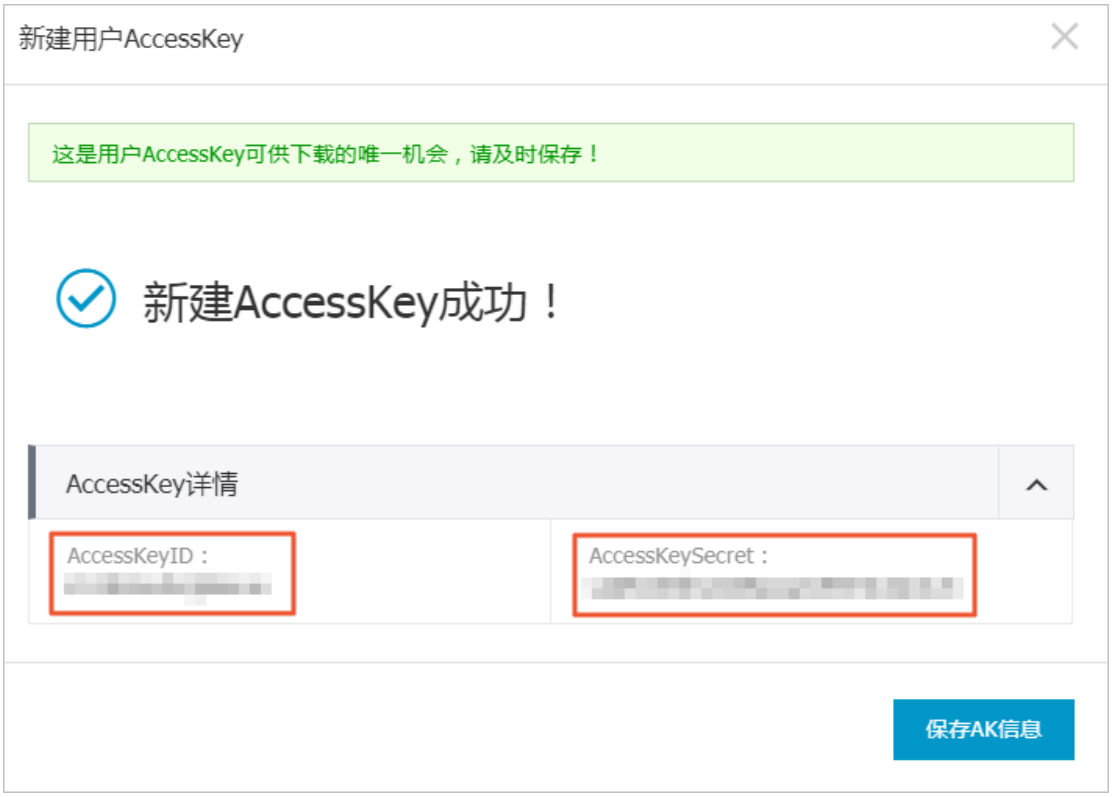
   c. 进入到安全令牌快捷配置页面，单击开始授权。

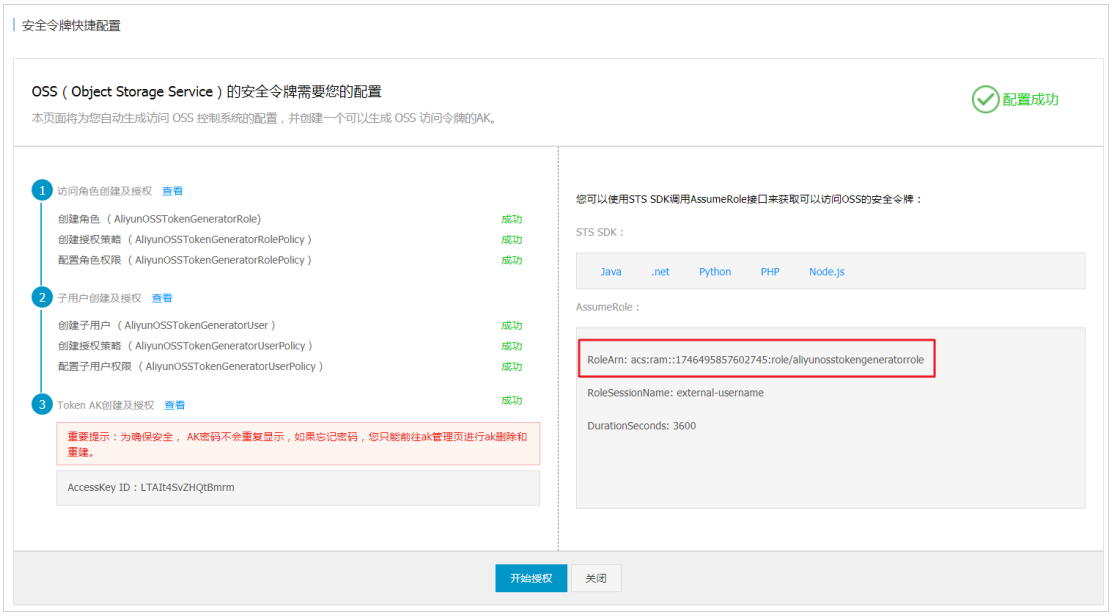
> **说明:**

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。

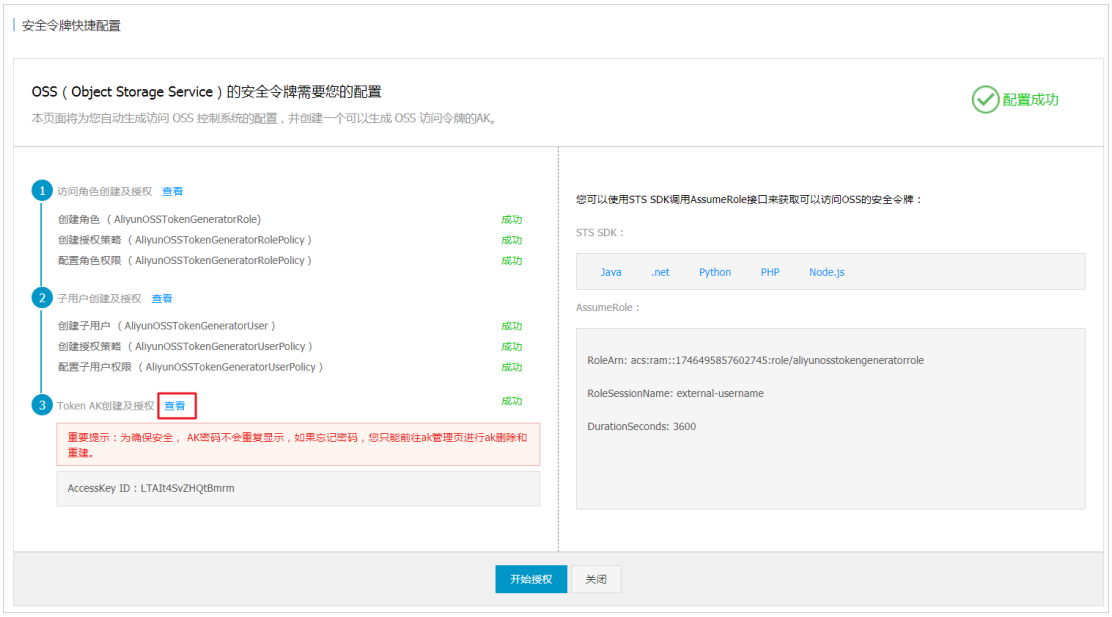
d. 系统进行自动授权。请保存AccessKeyID、AccessKeySecret和RoleArn三个参数。

- 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˙ə˙ə3qaPer8p1YaX1NTDiCFZWFkvlHf1pQhuxfKBc+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];
```

4.2 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

4.3 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。

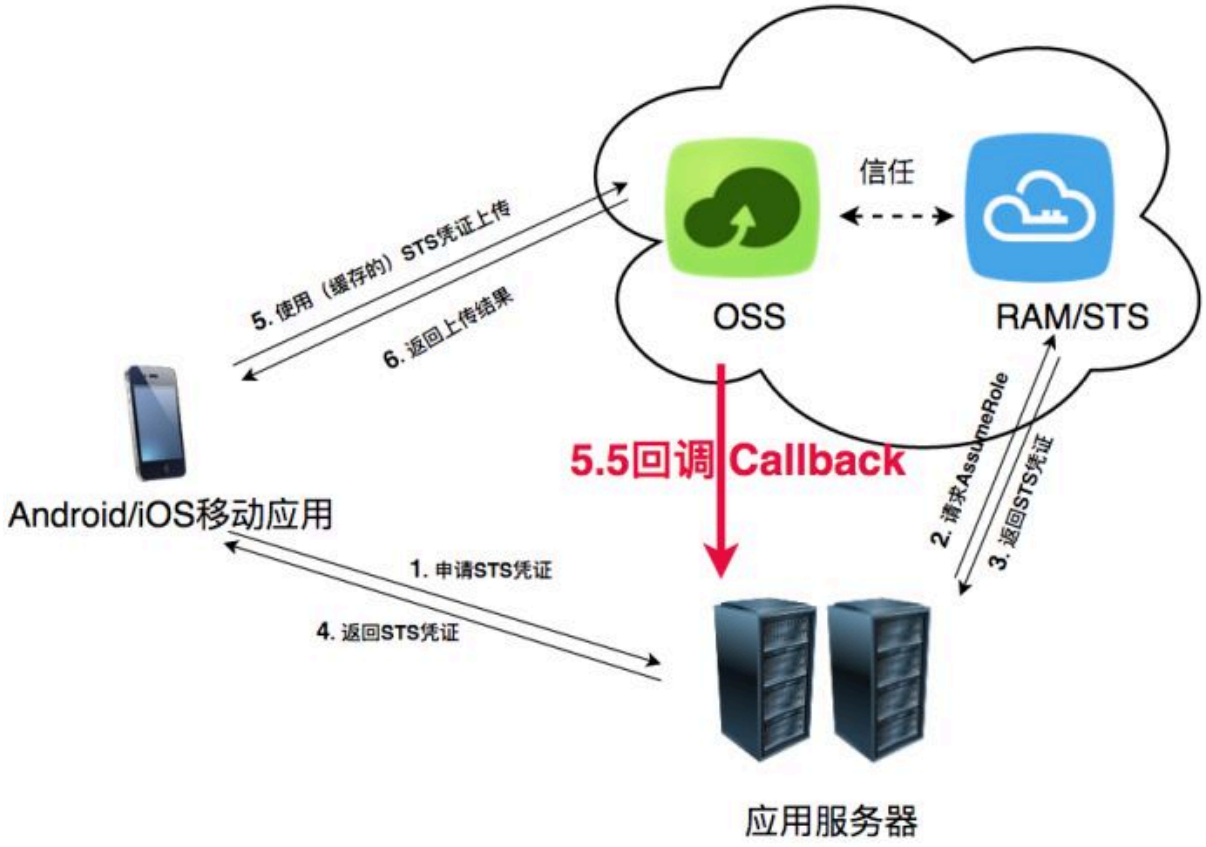
背景信息

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。

5 数据处理与分析

5.1 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_94。
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_96。
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_99/unique_99_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         |
+------------+------------+-----------+
```

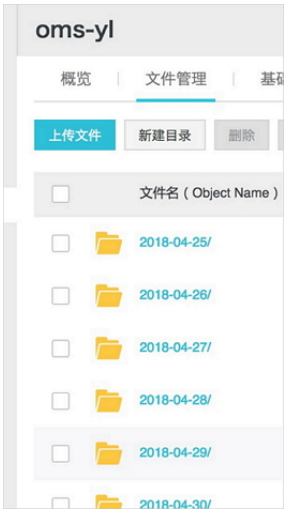
   更多关于OSS外部表使用方法，请参考#unique_100。

操作示例：阿里云产品消费账单分析

1. 开通OSS和MaxCompute服务，创建OSS bucket、MaxCompute Project，并授权MaxCompute访问OSS，详细步骤请参考上个示例中的1、2、3、4步。
2. 在阿里云控制台中，单击费用 > 消费记录 > 存储到OSS，输入用于存储文件的OSS bucket名称，本示例中为：*oms-yl*，如下图所示：

   服务开通后，每天会将增量的实例消费明细数据生成文件，并同步存储到指定的bucket中，如下图所示：

3. 安装MaxCompute客户端，并下载自定义代码。
4. 运行以下命令，将自定义代码编译打包，并上传到MaxCompute。

   ```
   add jar odps-udf-example-0.30.0-SNAPSHOT-jar-with-dependencies.jar
   ```

5. 通过MaxCompute创建外部表，详细步骤请参考创建表，

   以创建5月4日的账单消费表为例，外部表如下：

   ```
   CREATE EXTERNAL TABLE IF NOT EXISTS oms_oss_0504
   (
   月份 string,
   资源拥有者 string,
   消费时间 string,
   消费类型 string,
   账单编号 string,
   商品 string,
   计费方式 string,
   服务开始时间 string,
   服务结束时间 string,
   服务时长 string,
   ```

```
财务核算单元 string,
资源id string,
资源昵称 string,
TAG string,
地域 string,
可用区 string,
公网ip string,
内网ip string,
资源配置 string,
原价 string,
优惠金额 string,
应付金额 string,
计费项1 string,
使用量1 string,
资源包扣除1 string,
原价1 string ,
应付金额1 string,
计费项2 string,
使用量2 string,
资源包扣除2 string,
原价2 string,
应付金额2 string,
计费项3 string,
使用量3 string,
资源包扣除3 string,
原价3 string,
应付金额3 string,
计费项4 string,
使用量4 string,
资源包扣除4 string,
原价4 string,
应付金额4 string,
计费项5 string,
使用量5 string,
资源包扣除5 string,
原价5 string,
应付金额5 string,
计费项6 string,
使用量6 string,
资源包扣除6 string,
原价6 string,
应付金额6 string,
计费项7 string,
使用量7 string,
资源包扣除7 string,
原价7 string,
应付金额7 string,
计费项8 string,
使用量8 string,
资源包扣除8 string,
原价8 string,
应付金额8 string,
计费项9 string,
使用量9 string,
资源包扣除9 string,
原价9 string,
应付金额9 string
)
STORED BY 'com.aliyun.odps.udf.example.text.TextStorageHandler' --
STORED BY 指定自定义 StorageHandler 的类名。
with SERDEPROPERTIES (
'odps.text.option.complex.text.enabled'='true',
'odps.text.option.strict.mode'='false'
```

```
--遇到列数不一致的情况不会抛异常，如果实际列数少于schema列数，将所有列按顺序匹配，剩下的不足的列补NULL。
)
LOCATION 'oss://oss-cn-beijing-internal.aliyuncs.com/oms-yl/2018-05-04/'
USING 'text_oss.jar'; --同时需要指定账单中的文本处理类定义所在的 jar 包。
```

6. 通过MaxCompute查询外部表，查询步骤可参考 #unique_99/unique_99_Connect_42_section_ynz_3mq_kgb

   以查询ECS快照消费账单明细为例，命令如下：

```
select  月份,原价,优惠金额,应付金额,计费项1,使用量 from oms_oss where  商品=快照(SNAPSHOT);
```

5.2 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。

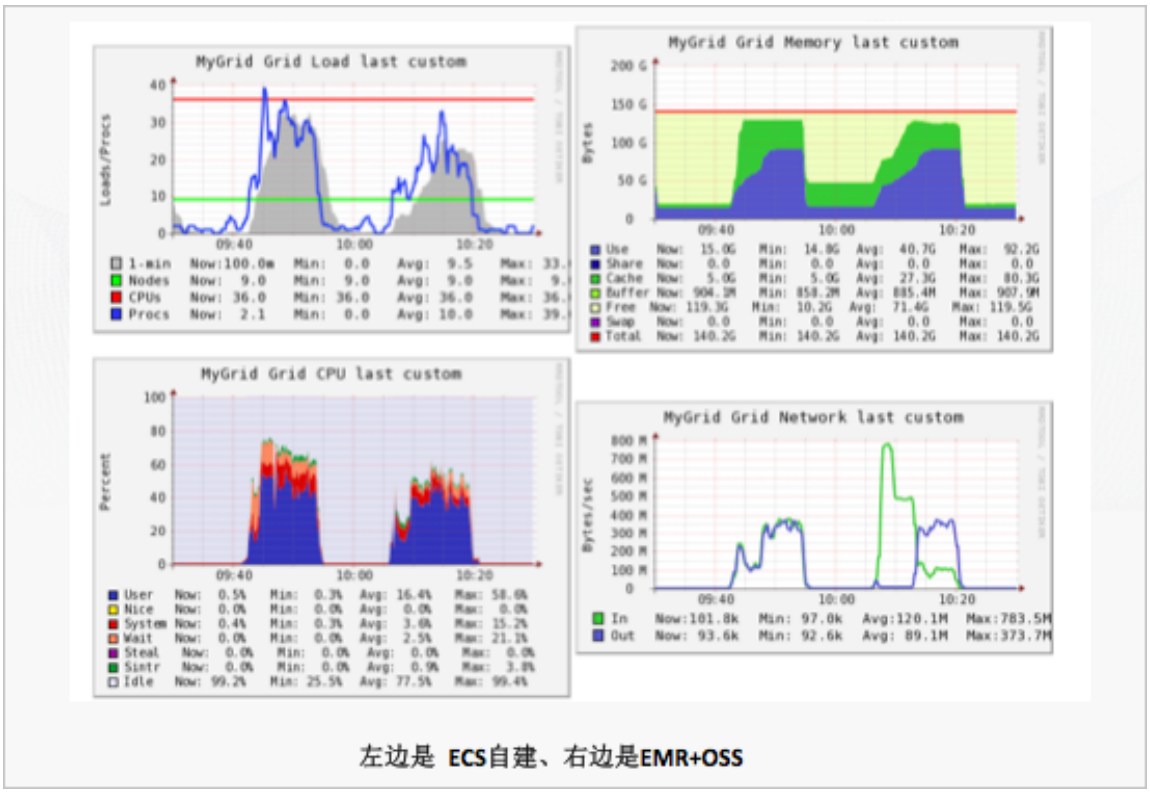
  集群规模：1 master 4cpu 16g、8 Slave 4cpu 16g、每个slave节点250G*4 高效云盘

  Spark测试脚本如下所示。

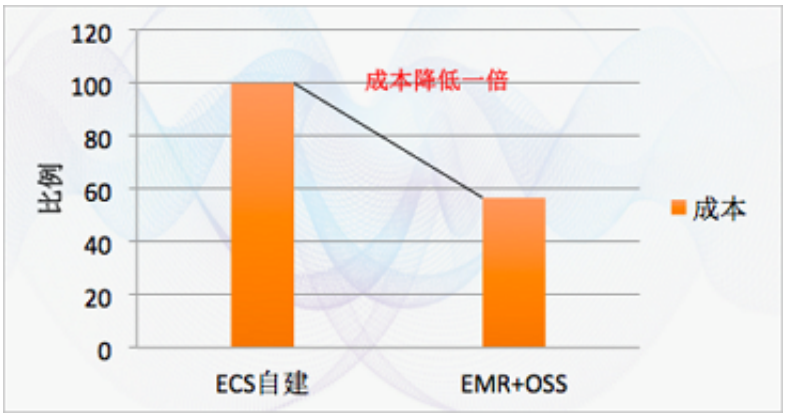
```
/opt/apps/spark-1.6.1-bin-hadoop2.7/bin/spark-submit  --master yarn
 --deploy-mode cluster --executor-memory 3G --num-executors 30
 --conf spark.default.parallelism=800   --class  com.github.ehiggs
```

```
.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。

5.3 使用OSS中的数据作为机器学习的训练样本

本文介绍如何将对象存储OSS里面的数据作为机器学习PAI的训练样本。

说明:

本文由 龙临@阿里云 提供，仅供参考。

背景信息

本文通过OSS与PAI的结合，为一家传统的文具零售店提供决策支持。本文涉及的具体业务场景（场景与数据均为虚拟）如下：

一家传统的线下文具零售店，希望通过数据挖掘寻找强相关的文具品类，帮助合理调整文具店的货架布局。但由于收银设备陈旧，是一台使用XP系统的POS收银机，可用的销售数据仅有一份从POS收银机导出的订单记录（csv格式）。本文介绍如何将此csv文件导入OSS，并连通OSS与PAI，实现商品的关联推荐。

操作步骤

1. 数据导入OSS

   a. 新建一个名为“oss-pai-sample”的Bucket。

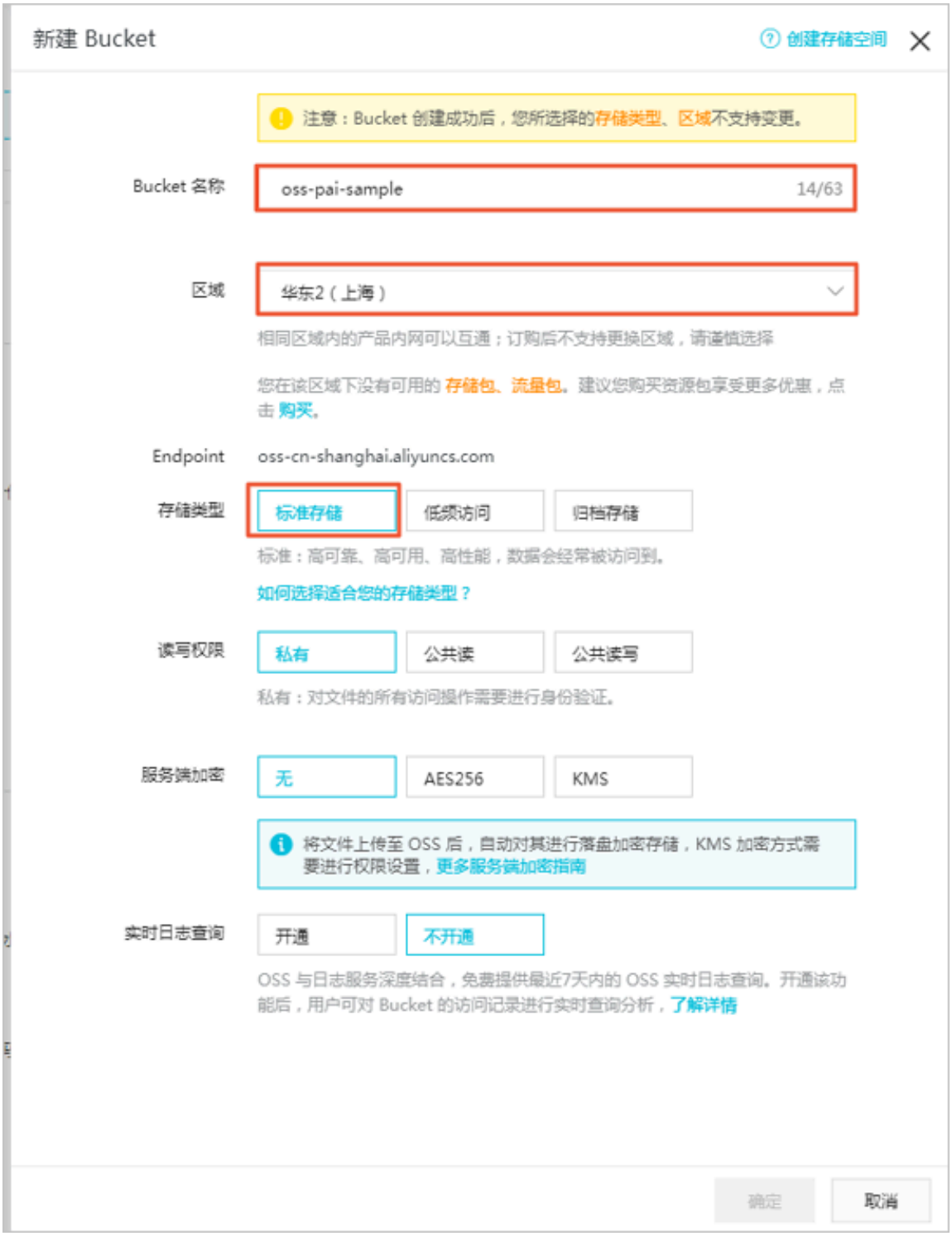
   b. 记录其Endpoint为 oss-cn-shanghai.aliyuncs.com。

   c. 选择存储类型为标准存储。

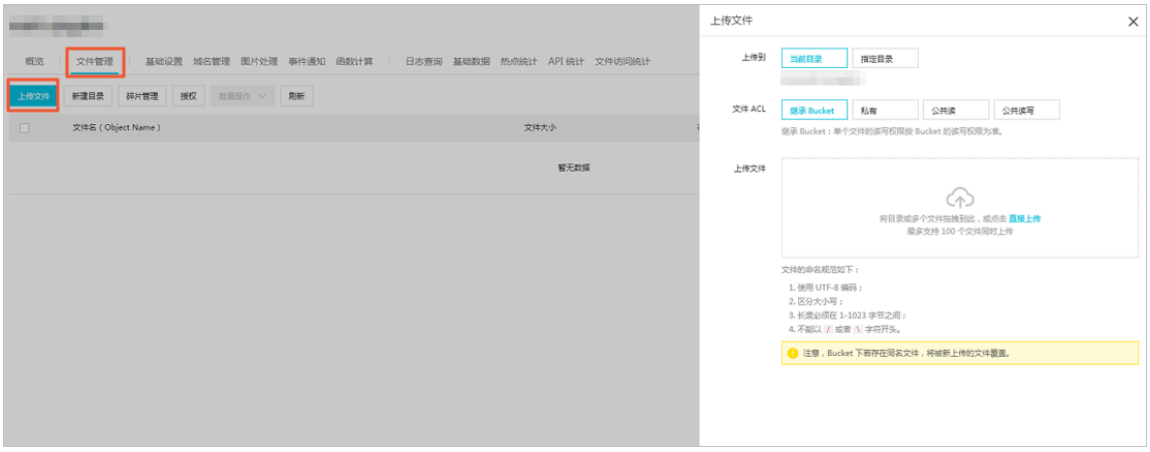
   

   说明:

OSS中有三种存储类型，相关介绍请参见存储类型介绍。



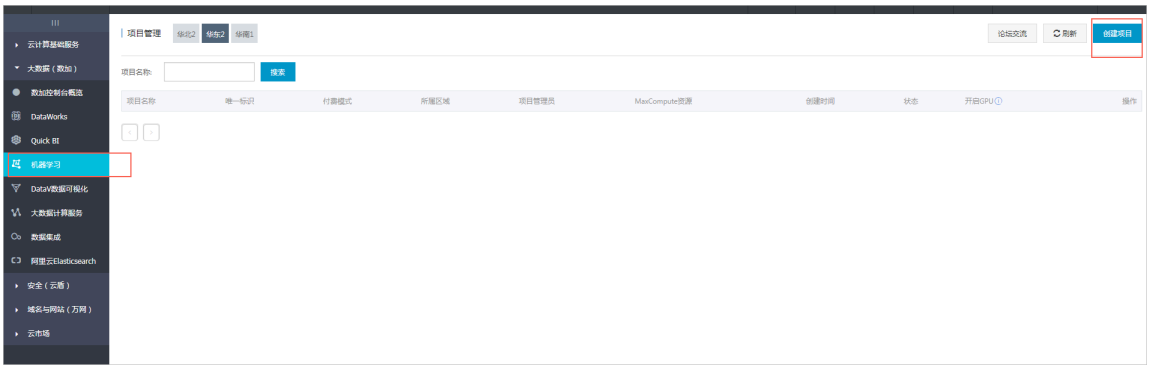
d. 单击Bucket名称（oss-pai-sample），然后依次单击文件管理 > 上传文件，将订单数据Sample_superstore.csv上传至OSS。



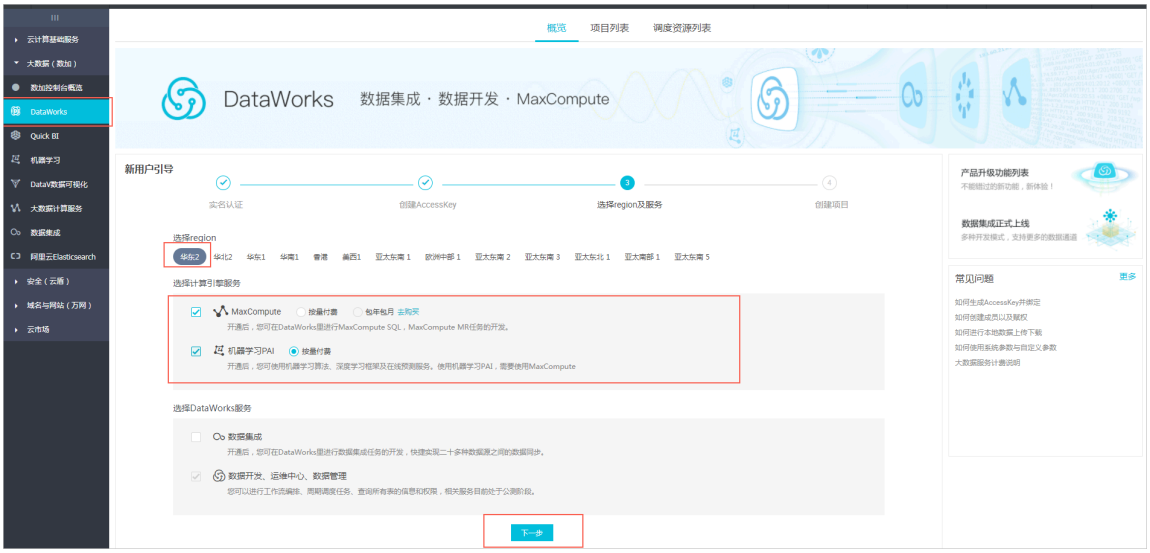
e. 上传成功，界面如下图所示：

2. 创建机器学习项目

a. 在控制台页面左侧选择机器学习，单击右上角的创建项目。

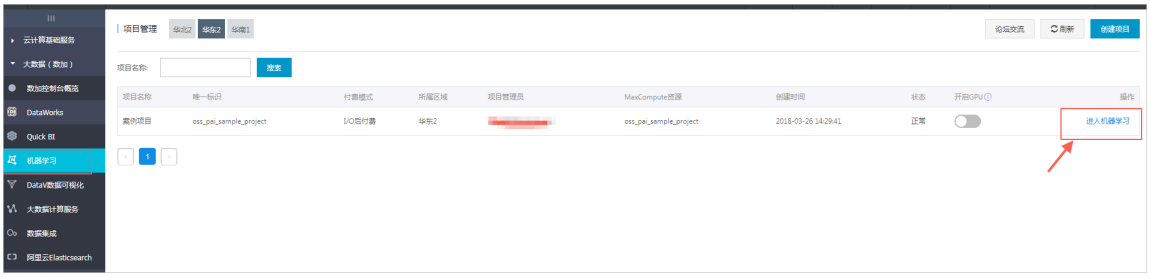


b. 在显示的DataWorks新用户引导界面中，勾选region（本文中选择与OSS相同的region：华东2），并勾选计算引擎服务机器学习PAI，然后单击下一步。



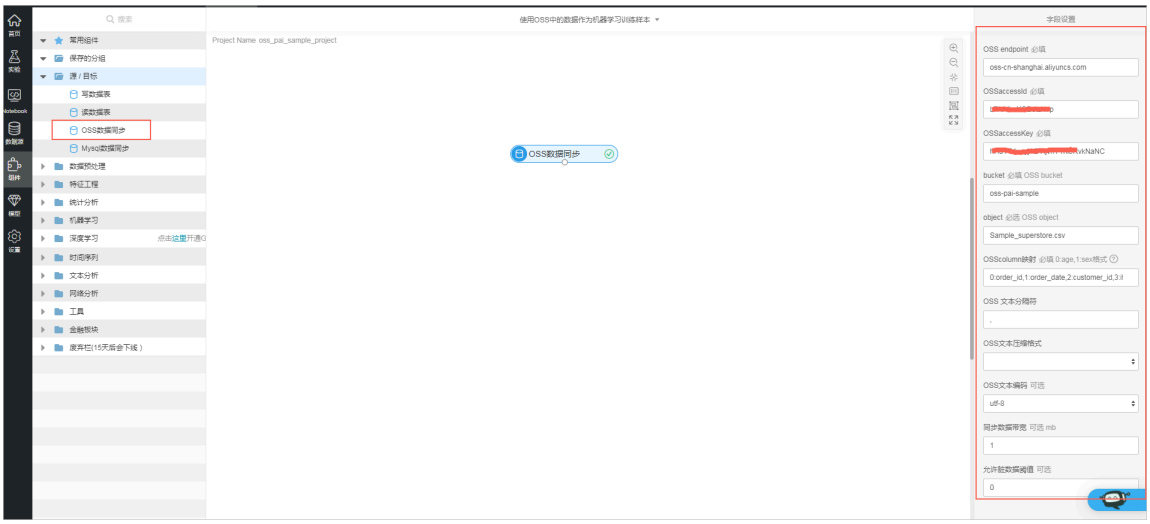
c. 项目创建成功后，开通服务列中会显示MaxCompute和机器学习PAI两个图标，如下图所示：

d. 回到机器学习页面，点击进入机器学习。


项目管理
华北2
华东2
华南1
论坛交流
刷新
创建项目
项目名称:
搜索
项目名称
唯一标识
付费模式
所属区域
项目管理员
MaxCompute资源
创建时间
状态
开启GPU
操作
oss_pai_sample_project
I/O后付费
华东2
2018-03-26 14:29:41
正常
进入机器学习
云计算基础服务
大数据（数加）
数加控制台概览
DataWorks
Quick BI
机器学习
DataV数据可视化
大数据计算服务
数据集成
阿里云Elasticsearch

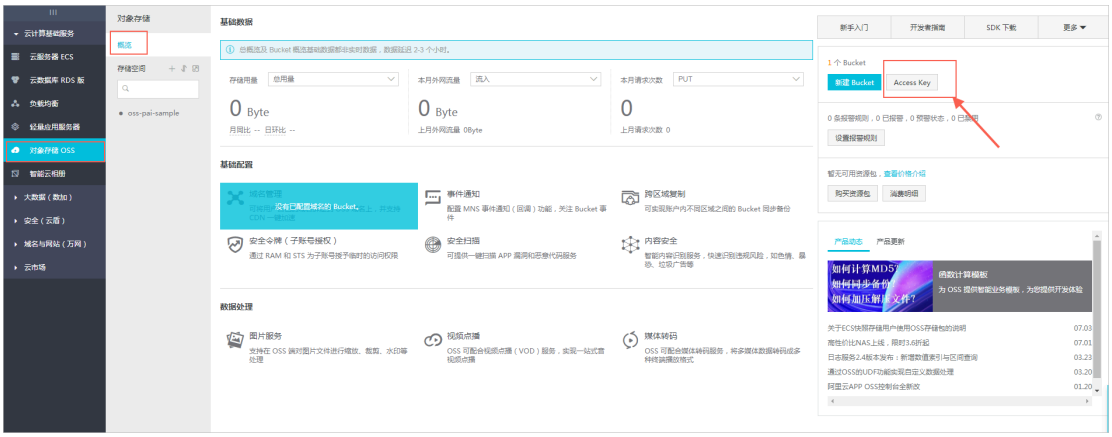
3. 连通OSS与PAI。

a. 在机器学习界面左侧选择组件，并将OSS数据同步组件拖拽至画布。



界面右侧会提示填入组件需要的以下信息：

- OSS endpoint：根据步骤一中记录的信息，endpoint 为 oss-cn-shanghai.aliyuncs.com。
- OSSAccessId 和 OSSAccessKey 可以在对象存储OSS的界面中获取，如下图所示：

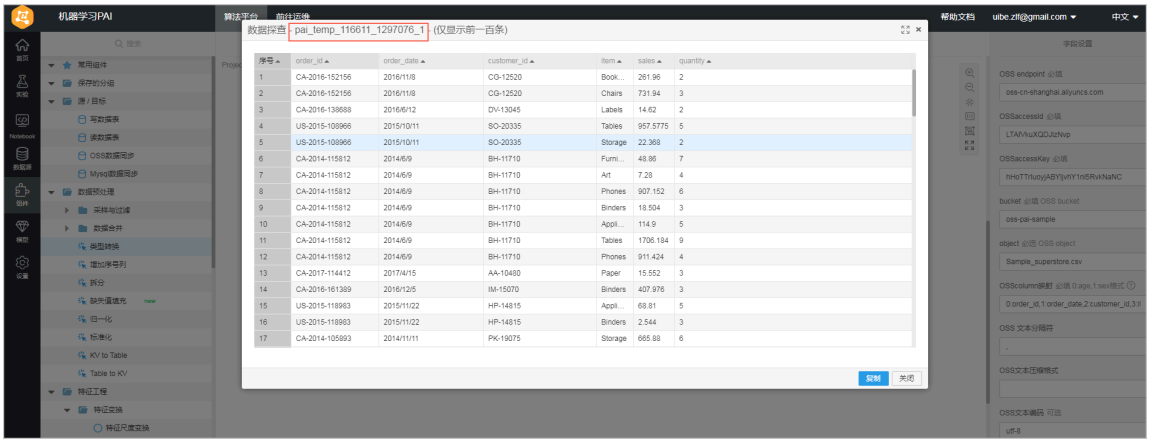


- OSSbucket 和 object 分别为 oss-pai-sample 和 Sample_superstore.csv。
- OSScolumn 映射的作用是为OSS中的csv文件增加列名。例如，虚拟数据Sample_superstore.csv共有如下6列：

序号	字段名称	字段类型	字段备注
1	order_id	string	订单号
2	order_date	string	订单日期
3	customer_id	string	用户编号
4	item	string	产品
5	sales	string	销售额
6	quantity	string	销售数量

则OSScolumn映射应该填入：0:order_id,1:order_date,2:customer_id,3:item,4:sales,5:quantity

b. 单击运行，成功后右键查看组件，可观察前100条数据，如下图所示：



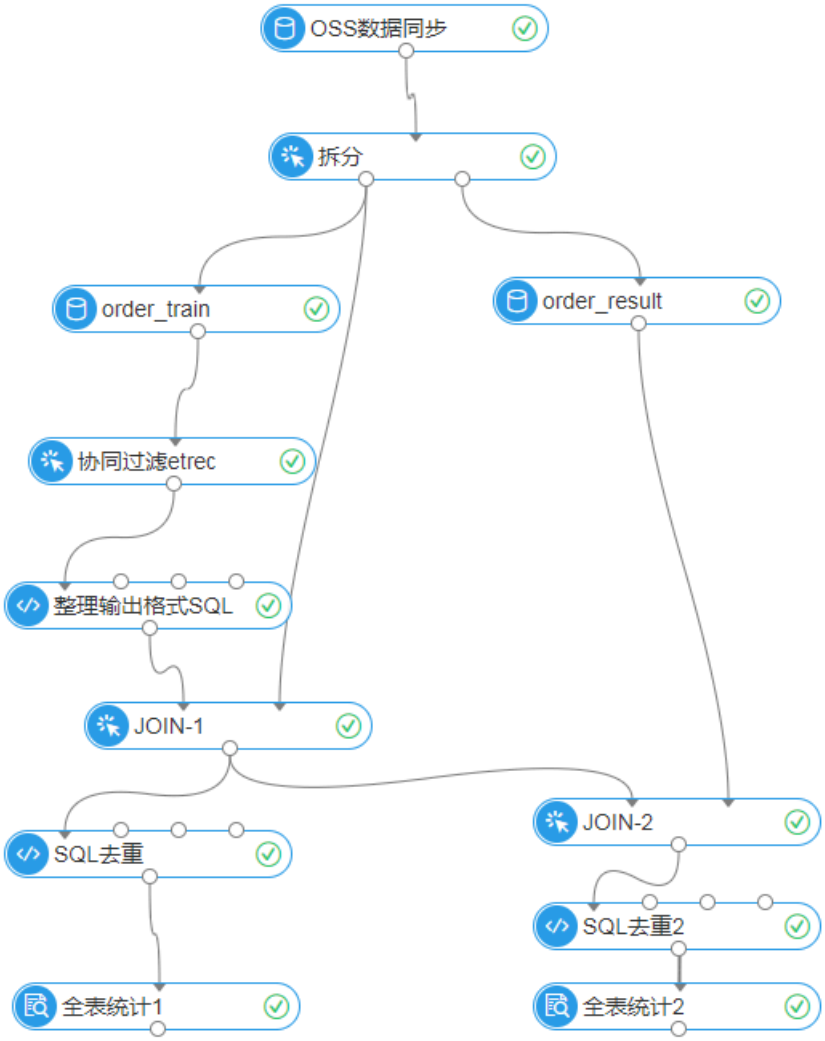
此时OSS中的csv文件已经在MaxCompute中生成一张临时表：pai_temp_116611_1297076_1

至此，本案例最关键的步骤已经完成，OSS中的数据已经与PAI连通，可以作为机器学习的样本进行训练。

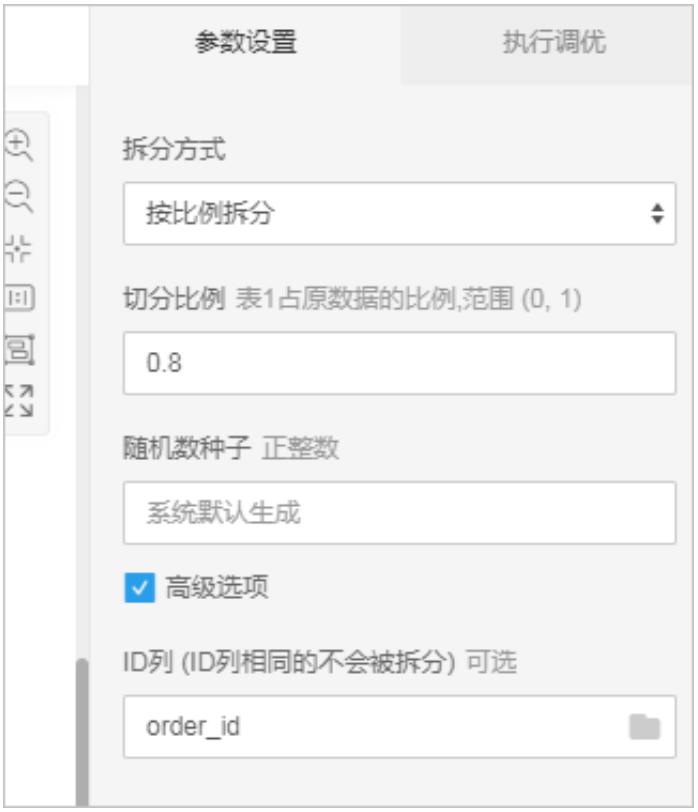
数据探索流程

本文所用的主要算法组件为协同过滤。有关该组件的详细用法，请参见协同过滤做商品推荐。

本案例中的数据探索流程如下：



本案例按8:2的比例将源数据拆分为训练集和测试集，其中一个订单中可能有多个item，故ID列选择order_id，保证含有多个item的订单不会被拆分，如下图所示：



本案例中共有17个产品item。通过协同过滤算法组件，取相似度最高的item，结果如下表：

序号	itemid	similar_item	similarity
1	Binders	Paper	0.621813
2	Paper	Binders	0.621813
3	Furnishings	Paper	0.538344
4	Phones	Paper	0.513804
5	Storage	Paper	0.511521
6	Accessories	Paper	0.498457
7	Art	Binders	0.497823
8	Chairs	Paper	0.431782
9	Appliances	Binders	0.375580
10	Labels	Binders	0.303599
11	Tables	Binders	0.276730
12	Envelopes	Storage	0.232794
13	Fasteners	Storage	0.232218
14	Bookcases	Storage	0.229209
15	Supplies	Accessories	0.199566
16	Machines	Fasteners	0.139423
17	Copiers	Machines	0.088710

结论

通过机器学习，我们发现“纸张”与“订书器”二者的相似度较高，且与其它产品也有较高的相似度。

对于这家文具零售店来说，根据此数据发现可以有两种布局货架的方式：

- 纸张和订书器货架放在最中间，其它产品货架呈环形围绕二者摆放，这样无论顾客从哪个货架步入，都可以快速找到关联程度较高的纸张和订书器。
- 将纸张和订书器两个货架分别摆放在文具店的两端，顾客需要横穿整个文具店才可以购买到另外一样，中途路过其他产品的货架可以提高交叉购买率。当然，此布局方式牺牲了用户购物的便利性，实际操作中应保持慎重。

5.4 DataLakeAnalytics+OSS：基于OSS的Severless的交互式查询分析

本文以基金交易数据处理为例，介绍将数据存储在OSS，使用DataLakeAnalytics进行Severless的交互式查询分析。

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。

用户可以打造基于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。

DataLakeAnalytics

Data Lake Analytics是无服务器（Serverless）化的云上交互式查询分析服务。无需ETL，就可通过此服务在云上通过标准JDBC直接对阿里云OSS、TableStore的数据轻松进行查询和分析，以及无缝集成商业分析工具。

服务开通

- OSS服务：

  开通OSS服务

- DataLakeAnalytics服务：

  开通DataLakeAnalytics服务

数据导出到OSS

以基金交易数据为例：

假设在OSS上存储了以下trade\user文件夹，并存储相应的交易数据和开户信息：

下载模拟数据（该数据为以下实验中使用的模拟数据。）

登录Data Lake Analytics控制台

点击登录数据库，使用方法可以参考 DataLakeAnalytics 产品帮助文档。

创建Schema和Table

- 创建Schema

  输入创建SCHEMA的语句，点击同步执行。

```
CREATE SCHEMA sh_trade
```

  CREATE SCHEMA sh_trade（注意：同一个阿里云region，schema名全局唯一，建议根据业务定义。如有重名schema，在创建时会提示报错，则需换另一个schema名。）

- 创建Table

  在数据库的下拉框中，选择刚刚创建的schema。

> 说明:
> 创建交易记录表及创建开户信息表时：
> - LOCATION ‘oss://Bucket名称/交易记录表目录/‘
> - Location替换为您的Bucket和测试数据的路径

  - 创建交易记录表：

    在SQL文本框中输入建表语句如下：

```
CREATE EXTERNAL TABLE tradelist_csv (
    t_userid STRING COMMENT '用户ID',
    t_dealdate STRING COMMENT '申请时间',
    t_businflag STRING COMMENT '业务代码',
    t_cdate STRING COMMENT '确认日期',
    t_date STRING COMMENT '申请日期',
    t_serialno STRING COMMENT'申请序号',
    t_agencyno STRING COMMENT'销售商编号',
    t_netno STRING  COMMENT'网点编号',
    t_fundacco STRING COMMENT'基金账号',
    t_tradeacco STRING COMMENT'交易账号',
    t_fundcode STRING  COMMENT'基金代码',
    t_sharetype STRING COMMENT'份额类别',
    t_confirmbalance DOUBLE  COMMENT'确认金额',
    t_tradefare DOUBLE COMMENT'交易费',
    t_backfare DOUBLE COMMENT'后收手续费',
    t_otherfare1 DOUBLE COMMENT'其他费用1',
    t_remark STRING COMMENT'备注'
    )
    ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
```

```
    STORED AS TEXTFIlE
    LOCATION 'oss://testdatasample/workshop_sh/trade/';
```

- 创建开户信息表：

```
CREATE EXTERNAL TABLE userinfo (
    u_userid STRING COMMENT '用户ID',
    u_accountdate STRING COMMENT '开户时间',
    u_gender STRING COMMENT '性别',
    u_age INT COMMENT '年龄',
    u_risk_tolerance INT COMMENT '风险承受能力，1-10，10为最高级',
    u_city STRING COMMENT'所在城市',
    u_job STRING COMMENT'工作类别，  A-K',
    u_income DOUBLE  COMMENT'年收入(万)'
    )
    ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
    STORED AS TEXTFIlE
    LOCATION 'oss://testdatasample/workshop_sh/user/';
```

建表完毕后，刷新页面，在左边导航条中能看到schema下的2张表。

SQL查询(同步执行)

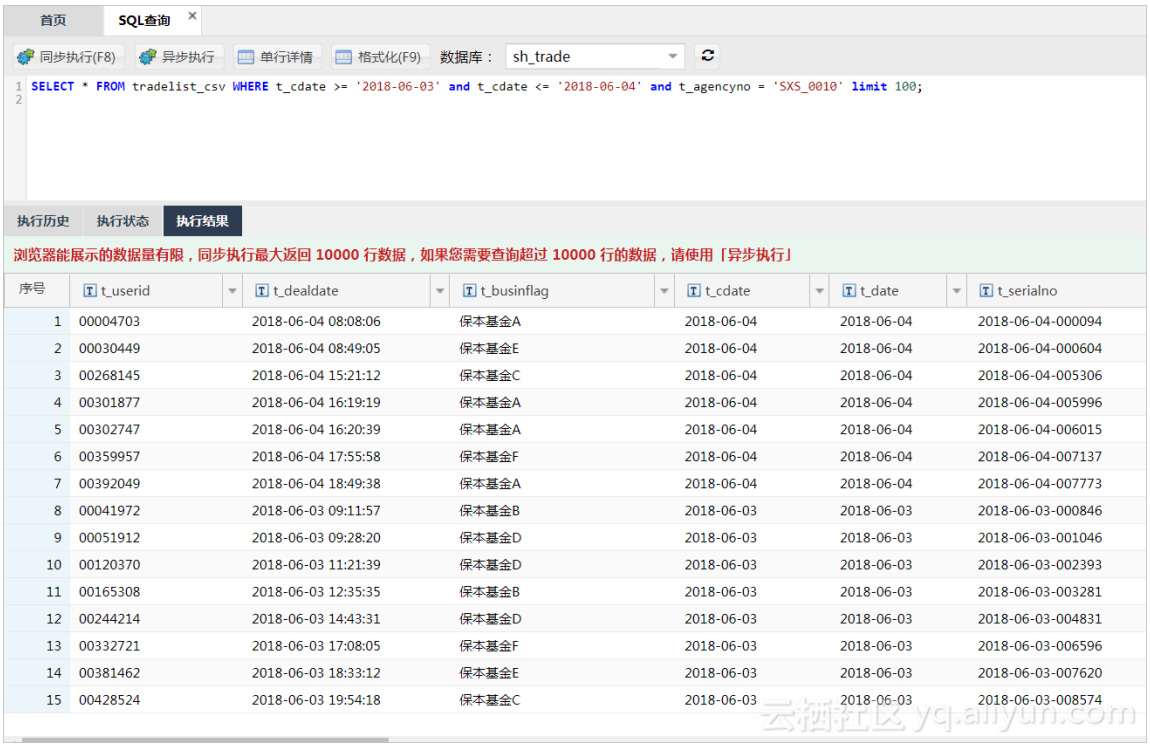
- 查询交易机构SXS_0010，在0603至0604的100条交易记录

1. 查询SQL

```
SELECT * FROM tradelist_csv
WHERE t_cdate >= '2018-06-03' and t_cdate <= '2018-06-04' and
t_agencyno = 'SXS_0010'
```

```
limit 100;
```

2. 显示执行结果



序号	t_userid	t_dealdate	t_businflag	t_cdate	t_date	t_serialno
1	00004703	2018-06-04 08:08:06	保本基金A	2018-06-04	2018-06-04	2018-06-04-000094
2	00030449	2018-06-04 08:49:05	保本基金E	2018-06-04	2018-06-04	2018-06-04-000604
3	00268145	2018-06-04 15:21:12	保本基金C	2018-06-04	2018-06-04	2018-06-04-005306
4	00301877	2018-06-04 16:19:19	保本基金A	2018-06-04	2018-06-04	2018-06-04-005996
5	00302747	2018-06-04 16:20:39	保本基金A	2018-06-04	2018-06-04	2018-06-04-006015
6	00359957	2018-06-04 17:55:58	保本基金F	2018-06-04	2018-06-04	2018-06-04-007137
7	00392049	2018-06-04 18:49:38	保本基金A	2018-06-04	2018-06-04	2018-06-04-007773
8	00041972	2018-06-03 09:11:57	保本基金B	2018-06-03	2018-06-03	2018-06-03-000846
9	00051912	2018-06-03 09:28:20	保本基金D	2018-06-03	2018-06-03	2018-06-03-001046
10	00120370	2018-06-03 11:21:39	保本基金D	2018-06-03	2018-06-03	2018-06-03-002393
11	00165308	2018-06-03 12:35:35	保本基金B	2018-06-03	2018-06-03	2018-06-03-003281
12	00244214	2018-06-03 14:43:31	保本基金D	2018-06-03	2018-06-03	2018-06-03-004831
13	00332721	2018-06-03 17:08:05	保本基金F	2018-06-03	2018-06-03	2018-06-03-006596
14	00381462	2018-06-03 18:33:12	保本基金E	2018-06-03	2018-06-03	2018-06-03-007620
15	00428524	2018-06-03 19:54:18	保本基金C	2018-06-03	2018-06-03	2018-06-03-008574

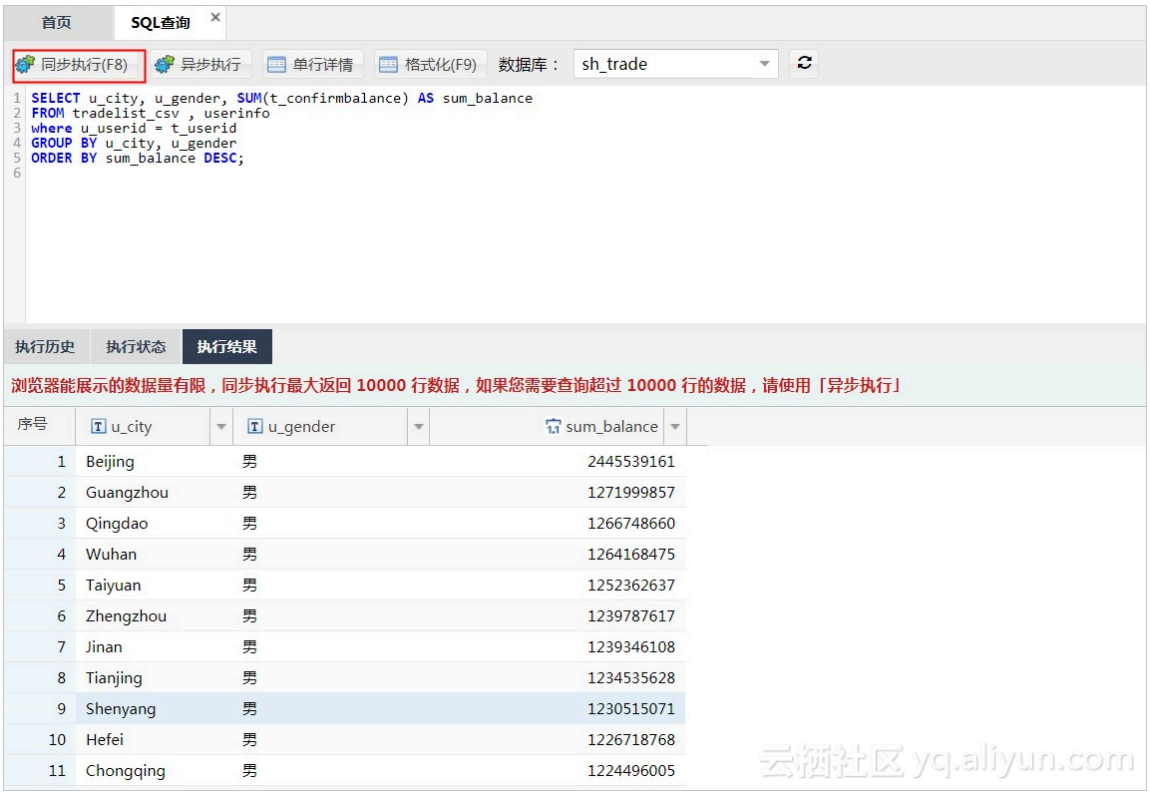
- 查询各城市、男性女性人群，购买的基金总额（多表Join查询）

1. 查询SQL

```
SELECT u_city, u_gender, SUM(t_confirmbalance) AS sum_balance
FROM tradelist_csv , userinfo
where u_userid = t_userid
GROUP BY u_city, u_gender
```

```
ORDER BY sum_balance DESC;
```

2. 显示执行结果



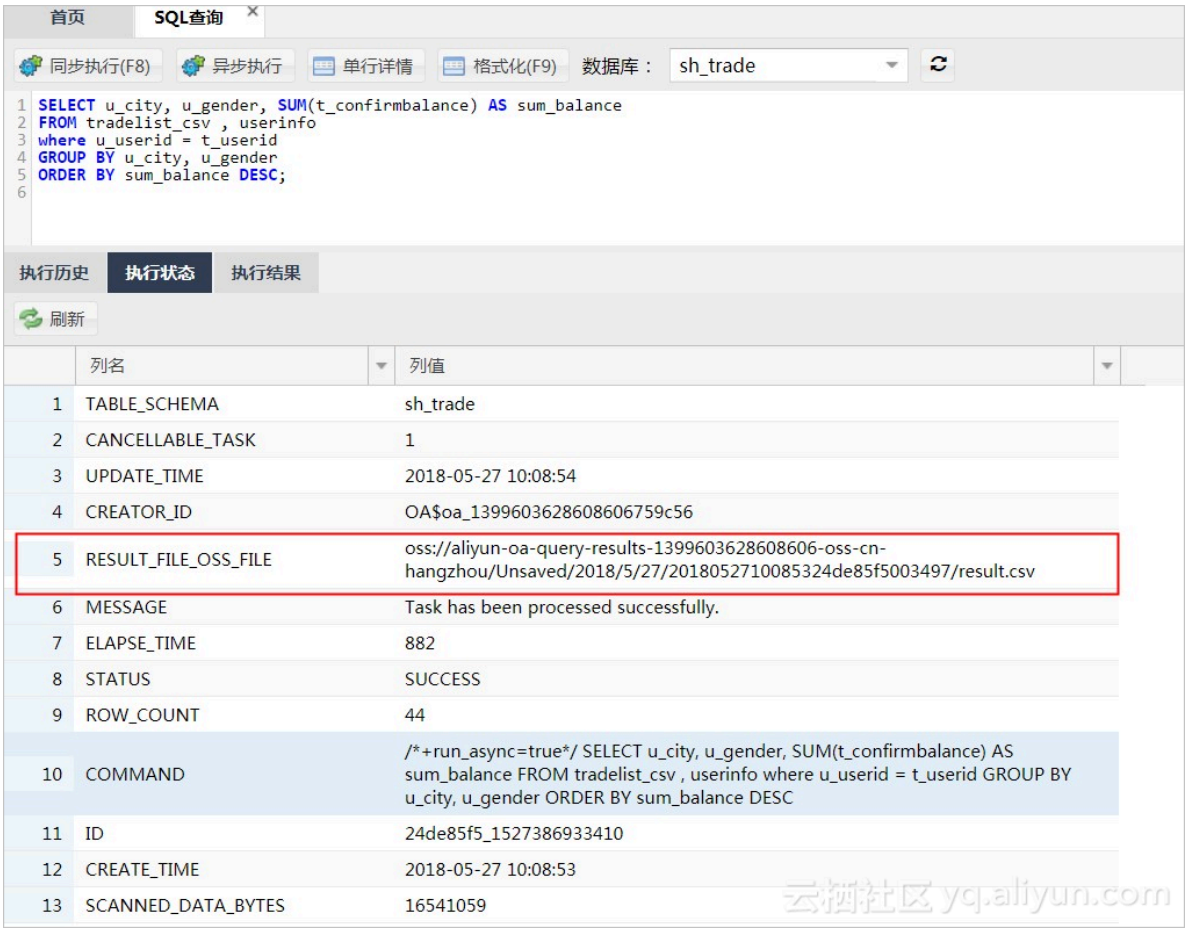
SQL查询(异步执行)

1. 异步执行查询，将查询结果以CSV格式输出到OSS。

2. 点击执行状态，可查看该异步查询任务的执行状态。

说明:

执行状态主要分为RUNNING、SUCCESS 和 FAILURE三种。点击刷新，当STATUS变为SUCCESS时，可以查看查询结果输出到OSS的文件路径。



3. 查看导出OSS的结果文件。

5.5 通过HDP2.6 Hadoop读取和写入OSS数据

HDP（Hortonworks Data Platform） 是由 Hortonworks 发行的大数据平台，包含了 Hadoop、Hive、HBase 等开源组件。HDP 最新版本3.0.1 中的 Hadoop3.1.1 版本已经支持 OSS，但是低版本的 HDP 不支持 OSS。本文以 HDP2.6.1.0 版本为例，介绍如何配置 HDP2.6 版本支持读写 OSS。

准备工作

您需要拥有一个已搭建好的 HDP2.6.1.0 的集群。若没有已搭建好的 HDP2.6.1.0 集群，您可以通过以下方式搭建：

- 查找参考文档利用 Ambari 搭建 HDP2.6.1.0 的集群。
- 不使用 Ambari，自行搭建 HDP2.6.1.0 集群。

配置步骤

1. 下载 HDP2.6.1.0 版本支持 OSS的支持包。

   说明：

   此支持包是根据HDP2.6.1.0中 Hadoop 的版本，打了 Apache Hadoop 对 OSS 支持的补丁后编译得到的，其他 HDP2 的小版本对 OSS 的支持将陆续提供。

2. 将下载的支持包解压：

```
 [root@hdp-master ~]# tar -xvf hadoop-oss-hdp-2.6.1.0-129.tar
hadoop-oss-hdp-2.6.1.0-129/
hadoop-oss-hdp-2.6.1.0-129/aliyun-java-sdk-ram-3.0.0.jar
hadoop-oss-hdp-2.6.1.0-129/aliyun-java-sdk-core-3.4.0.jar
hadoop-oss-hdp-2.6.1.0-129/aliyun-java-sdk-ecs-4.2.0.jar
hadoop-oss-hdp-2.6.1.0-129/aliyun-java-sdk-sts-3.0.0.jar
hadoop-oss-hdp-2.6.1.0-129/jdom-1.1.jar
hadoop-oss-hdp-2.6.1.0-129/aliyun-sdk-oss-3.4.1.jar
hadoop-oss-hdp-2.6.1.0-129/hadoop-aliyun-2.7.3.2.6.1.0-129.jar
```

3. 将 *hadoop-aliyun-2.7.3.2.6.1.0-129.jar* 移至 *${/usr/hdp/current}/hadoop-client/* 目录内，其余的*jar*文件移至*${/usr/hdp/current}/hadoop-client/lib/* 目录内。调整后，目录结构如下：

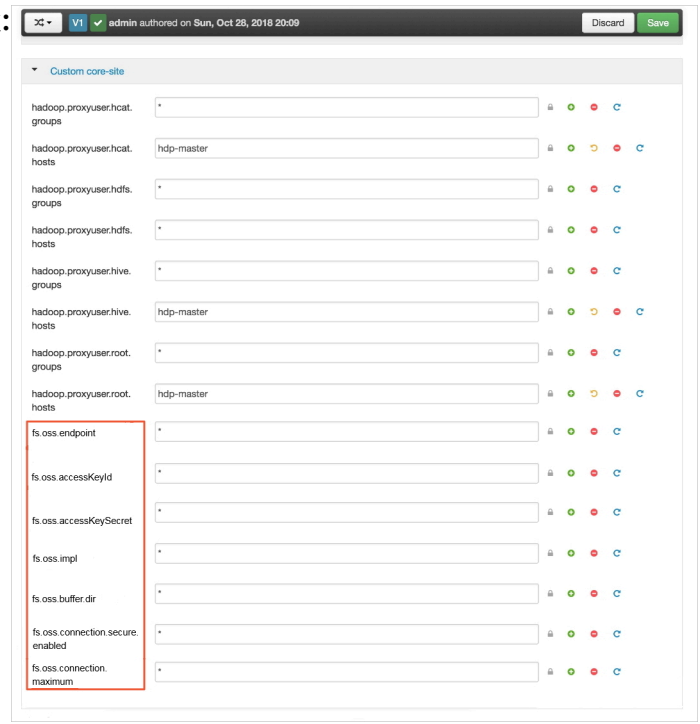
```
[root@hdp-master ~]# ls -lh /usr/hdp/current/hadoop-client/hadoop-
aliyun-2.7.3.2.6.1.0-129.jar
-rw-r--r-- 1 root root 64K Oct 28 20:56 /usr/hdp/current/hadoop-
client/hadoop-aliyun-2.7.3.2.6.1.0-129.jar

[root@hdp-master ~]# ls -ltrh /usr/hdp/current/hadoop-client/lib
total 27M
......
drwxr-xr-x 2 root root 4.0K Oct 28 20:10 ranger-hdfs-plugin-impl
drwxr-xr-x 2 root root 4.0K Oct 28 20:10 ranger-yarn-plugin-impl
drwxr-xr-x 2 root root 4.0K Oct 28 20:10 native
-rw-r--r-- 1 root root 114K Oct 28 20:56 aliyun-java-sdk-core-3.4.0.
jar
-rw-r--r-- 1 root root 513K Oct 28 20:56 aliyun-sdk-oss-3.4.1.jar
-rw-r--r-- 1 root root  13K Oct 28 20:56 aliyun-java-sdk-sts-3.0.0.
jar
-rw-r--r-- 1 root root 211K Oct 28 20:56 aliyun-java-sdk-ram-3.0.0.
jar
-rw-r--r-- 1 root root 770K Oct 28 20:56 aliyun-java-sdk-ecs-4.2.0.
jar
```

```
-rw-r--r-- 1 root root 150K Oct 28 20:56 jdom-1.1.jar
```

说明:

本文中所有 `${}` 的内容为环境变量，请根据您实际的环境修改。

4. **在所有的 HDP 节点执行以上操作。**
5. **通过 Ambari 来增加配置。没有使用 Ambari 管理的集群，可以修改`core-site.xml`。这里以 Ambari 为例，需要增加如下配置:**



配置项	值
fs.oss.endpoint	填写需要连接的 OSS 的 Endpoint。 例如：oss-cn-zhangjiakou-internal.aliyuncs.com
fs.oss.accessKeyId	填写 OSS 的 AccessKeyId。
fs.oss.accessKeySecret	填写 OSS 的 AccessKeySecret。
fs.oss.impl	Hadoop OSS 文件系统实现类。目前固定为：org.apache.hadoop.fs.aliyun.oss.AliyunOSSFileSystem
fs.oss.buffer.dir	填写临时文件目录。 建议值：/tmp/oss

配置项	值
fs.oss.connection.secure.enabled	是否开启 HTTPS。开启 HTTPS 会影响性能。 建议值：false
fs.oss.connection.maximum	与 OSS 的连接数 建议值：2048

更多参数解释可参考 Hadoop-Aliyun module。

6. 根据 Ambari 提示重启集群。
7. 测试读写 OSS：
   - 测试读：

     ```
     hadoop fs -ls oss://${your-bucket-name}/
     ```

   - 测试写：

     ```
     hadoop fs -mkdir oss://${your-bucket-name}/hadoop-test
     ```

     若测试可以读写 OSS，则配置成功；若无法读写 OSS，请检查配置。
8. 为了能够运行 MAPREDUCE 任务，还需更改*hdfs://hdp-master:8020/hdp/apps/2.6.1.0-129/mapreduce/mapreduce.tar.gz*包的内容，如果是 TEZ 类型的作业，则修改*hdfs://hdp-master:8020/hdp/apps/2.6.1.0-129/tez/tez.tar.gz*，其他类型的作业以此类推。将 OSS 的支持包放如上述压缩包，执行如下命令：

```
[root@hdp-master ~]# sudo su hdfs
[hdfs@hdp-master root]$ cd
[hdfs@hdp-master ~]$ hadoop fs -copyToLocal /hdp/apps/2.6.1.0-129/
mapreduce/mapreduce.tar.gz .
[hdfs@hdp-master ~]$ hadoop fs -rm /hdp/apps/2.6.1.0-129/mapreduce/
mapreduce.tar.gz
[hdfs@hdp-master ~]$ cp mapreduce.tar.gz mapreduce.tar.gz.bak
[hdfs@hdp-master ~]$ tar zxf mapreduce.tar.gz
[hdfs@hdp-master ~]$ cp /usr/hdp/current/hadoop-client/hadoop-aliyun
-2.7.3.2.6.1.0-129.jar hadoop/share/hadoop/tools/lib/
[hdfs@hdp-master ~]$ cp /usr/hdp/current/hadoop-client/lib/aliyun-*
hadoop/share/hadoop/tools/lib/
[hdfs@hdp-master ~]$ cp /usr/hdp/current/hadoop-client/lib/jdom-1.1.
jar hadoop/share/hadoop/tools/lib/
[hdfs@hdp-master ~]$ tar zcf mapreduce.tar.gz hadoop
[hdfs@hdp-master ~]$ hadoop fs -copyFromLocal mapreduce.tar.gz /hdp/
apps/2.6.1.0-129/mapreduce/
```

验证配置

可通过测试 teragen 和 terasort，来检测配置是否生效。

- 测试 teragen：

```
[hdfs@hdp-master ~]$ hadoop jar /usr/hdp/current/hadoop-mapreduce-
client/hadoop-mapreduce-examples.jar teragen -Dmapred.map.tasks=100
10995116 oss://{bucket-name}/1G-input
18/10/28 21:32:38 INFO client.RMProxy: Connecting to ResourceManager
 at cdh-master/192.168.0.161:8050
18/10/28 21:32:38 INFO client.AHSProxy: Connecting to Application
History server at cdh-master/192.168.0.161:10200
18/10/28 21:32:38 INFO aliyun.oss: [Server]Unable to execute HTTP
request: Not Found
[ErrorCode]: NoSuchKey
[RequestId]: 5BD5BA7641FCE369BC1D052C
[HostId]: null
18/10/28 21:32:38 INFO aliyun.oss: [Server]Unable to execute HTTP
request: Not Found
[ErrorCode]: NoSuchKey
[RequestId]: 5BD5BA7641FCE369BC1D052F
[HostId]: null
18/10/28 21:32:39 INFO terasort.TeraSort: Generating 10995116 using
100
18/10/28 21:32:39 INFO mapreduce.JobSubmitter: number of splits:100
18/10/28 21:32:39 INFO mapreduce.JobSubmitter: Submitting tokens for
 job: job_1540728986531_0005
18/10/28 21:32:39 INFO impl.YarnClientImpl: Submitted application
application_1540728986531_0005
18/10/28 21:32:39 INFO mapreduce.Job: The url to track the job: http
://cdh-master:8088/proxy/application_1540728986531_0005/
18/10/28 21:32:39 INFO mapreduce.Job: Running job: job_154072
8986531_0005
18/10/28 21:32:49 INFO mapreduce.Job: Job job_1540728986531_0005
running in uber mode : false
18/10/28 21:32:49 INFO mapreduce.Job:  map 0% reduce 0%
18/10/28 21:32:55 INFO mapreduce.Job:  map 1% reduce 0%
18/10/28 21:32:57 INFO mapreduce.Job:  map 2% reduce 0%
18/10/28 21:32:58 INFO mapreduce.Job:  map 4% reduce 0%
...
18/10/28 21:34:40 INFO mapreduce.Job:  map 99% reduce 0%
18/10/28 21:34:42 INFO mapreduce.Job:  map 100% reduce 0%
18/10/28 21:35:15 INFO mapreduce.Job: Job job_1540728986531_0005
completed successfully
18/10/28 21:35:15 INFO mapreduce.Job: Counters: 36
...
```

- 测试 terasort：

```
[hdfs@hdp-master ~]$ hadoop jar /usr/hdp/current/hadoop-mapreduce-
client/hadoop-mapreduce-examples.jar terasort -Dmapred.map.tasks=100
 oss://{bucket-name}/1G-input oss://{bucket-name}/1G-output
18/10/28 21:39:00 INFO terasort.TeraSort: starting
...
18/10/28 21:39:02 INFO mapreduce.JobSubmitter: number of splits:100
18/10/28 21:39:02 INFO mapreduce.JobSubmitter: Submitting tokens for
 job: job_1540728986531_0006
18/10/28 21:39:02 INFO impl.YarnClientImpl: Submitted application
application_1540728986531_0006
18/10/28 21:39:02 INFO mapreduce.Job: The url to track the job: http
://cdh-master:8088/proxy/application_1540728986531_0006/
18/10/28 21:39:02 INFO mapreduce.Job: Running job: job_154072
8986531_0006
18/10/28 21:39:09 INFO mapreduce.Job: Job job_1540728986531_0006
running in uber mode : false
```

```
18/10/28 21:39:09 INFO mapreduce.Job:  map 0% reduce 0%
18/10/28 21:39:17 INFO mapreduce.Job:  map 1% reduce 0%
18/10/28 21:39:19 INFO mapreduce.Job:  map 2% reduce 0%
18/10/28 21:39:20 INFO mapreduce.Job:  map 3% reduce 0%
...
18/10/28 21:42:50 INFO mapreduce.Job:  map 100% reduce 75%
18/10/28 21:42:53 INFO mapreduce.Job:  map 100% reduce 80%
18/10/28 21:42:56 INFO mapreduce.Job:  map 100% reduce 86%
18/10/28 21:42:59 INFO mapreduce.Job:  map 100% reduce 92%
18/10/28 21:43:02 INFO mapreduce.Job:  map 100% reduce 98%
18/10/28 21:43:05 INFO mapreduce.Job:  map 100% reduce 100%
^@18/10/28 21:43:56 INFO mapreduce.Job: Job job_1540728986531_0006
completed successfully
18/10/28 21:43:56 INFO mapreduce.Job: Counters: 54
...
```

测试成功，配置生效。

参考文档

关于 Hadoop 更多内容，可参考：Hadoop 支持集成 OSS。

您也可以通过阿里云 EMR 访问 OSS。阿里云 EMR 基于开源生态，包括 Hadoop、Spark、Kafka、Flink、Storm 等组件，为您提供集群、作业、数据管理等服务的一站式企业大数据平台，并无缝支持 OSS。阿里云 EMR 与 OSS 紧密结合，针对开源生态访问 OSS，有多项技术优化，详情可参考 EMR 产品介绍。

5.6 通过CDH5 Hadoop读取和写入OSS数据

CDH（Cloudera's Distribution, including Apache Hadoop）是众多 Hadoop 发行版本中的一种，最新版本 CDH6.0.1 中的 Hadoop3.0.0 版本已经支持 OSS。CDH5 中的 Hadoop2.6 版本不支持 OSS。本文介绍如何配置 CDH5 支持 OSS 读写。

由于 CDH5 的 httpclient 和 httpcore 这两个组件版本较低（4.2.5），Resource Manager 要求的 httpclient 和 httpcore 必须是低版本，而 OSS SDK 要求这两个组件的版本较高，因此，下面提供了一个 workaround 方案。

准备工作

您需要拥有一个已搭建好的 CDH5 集群。若没有已搭建好的 CDH5 集群，您需要参考官方文档，先搭建一个 CDH5 集群。本文以 CDH5.14.4 版本为例。

步骤一：增加 OSS 配置

您需要在所有的 CDH 节点执行以下操作：

1. 查看 CDH5 安装目录*${CDH_HOME}*的结构：

```
[root@cdh-master CDH-5.14.4-1.cdh5.14.4.p0.3]# ls -lh
总用量 100K
drwxr-xr-x  2 root root 4.0K 6月  12 21:03 bin
```

```
drwxr-xr-x 27 root root 4.0K 6月  12 20:57 etc
drwxr-xr-x  5 root root 4.0K 6月  12 20:57 include
drwxr-xr-x  2 root root  68K 6月  12 21:09 jars
drwxr-xr-x 38 root root 4.0K 6月  12 21:03 lib
drwxr-xr-x  3 root root 4.0K 6月  12 20:57 lib64
drwxr-xr-x  3 root root 4.0K 6月  12 20:51 libexec
drwxr-xr-x  2 root root 4.0K 6月  12 21:02 meta
drwxr-xr-x  4 root root 4.0K 6月  12 21:03 share
```

说明:

本文中所有 *${}* 的内容为环境变量，请根据您实际的环境修改。

2. **下载 CDH 5.14.4 版本支持 OSS 的支持包至 CDH5 的安装目录中的*jars*文件夹中。**

说明:

这个支持包是根据 CDH5.14.4 中 Hadoop 的版本，并打了 Apache Hadoop 对 OSS 支持的补丁后编译得到的，其他 CDH5 的小版本对 OSS 的支持将陆续提供。

3. **解压支持包：**

```
[root@cdh-master CDH-5.14.4-1.cdh5.14.4.p0.3]# tar -tvf hadoop-oss-
cdh-5.14.4.tar.gz
drwxr-xr-x root/root 0 2018-10-08 18:16 hadoop-oss-cdh-5.14.4/
-rw-r--r-- root/root 13277 2018-10-08 17:36 hadoop-oss-cdh-5.14.4/
aliyun-java-sdk-sts-3.0.0.jar
-rw-r--r-- root/root 326724 2018-10-08 18:16 hadoop-oss-cdh-5.14.4/
httpcore-4.4.4.jar
-rw-r--r-- root/root 524927 2018-10-08 17:36 hadoop-oss-cdh-5.14.4/
aliyun-sdk-oss-3.4.1.jar
-rw-r--r-- root/root 116337 2018-10-08 17:36 hadoop-oss-cdh-5.14.4/
aliyun-java-sdk-core-3.4.0.jar
-rw-r--r-- root/root 215492 2018-10-08 17:36 hadoop-oss-cdh-5.14.4/
aliyun-java-sdk-ram-3.0.0.jar
-rw-r--r-- root/root 788137 2018-10-08 17:36 hadoop-oss-cdh-5.14.4/
aliyun-java-sdk-ecs-4.2.0.jar
-rw-r--r-- root/root 70017 2018-10-08 17:36 hadoop-oss-cdh-5.14.4/
hadoop-aliyun-2.6.0-cdh5.14.4.jar
-rw-r--r-- root/root 736658 2018-10-08 18:16 hadoop-oss-cdh-5.14.4/
httpclient-4.5.2.jar
```

4. **进入*${CDH_HOME}/lib/hadoop*目录，执行如下命令：**

```
[root@cdh-master hadoop]# rm -f lib/httpclient-4.2.5.jar
[root@cdh-master hadoop]# rm -f lib/httpcore-4.2.5.jar
[root@cdh-master hadoop]# ln -s ../../jars/hadoop-aliyun-2.6.0-cdh5.
14.4.jar hadoop-aliyun-2.6.0-cdh5.14.4.jar
[root@cdh-master hadoop]# ln -s hadoop-aliyun-2.6.0-cdh5.14.4.jar
hadoop-aliyun.jar
[root@cdh-master hadoop]# cd lib
[root@cdh-master lib]# ln -s ../../../jars/aliyun-java-sdk-core-3.4.
0.jar aliyun-java-sdk-core-3.4.0.jar
[root@cdh-master lib]# ln -s ../../../jars/aliyun-java-sdk-ecs-4.2.0
.jar aliyun-java-sdk-ecs-4.2.0.jar
[root@cdh-master lib]# ln -s ../../../jars/aliyun-java-sdk-ram-3.0.0
.jar aliyun-java-sdk-ram-3.0.0.jar
```

```
[root@cdh-master lib]# ln -s ../../../jars/aliyun-java-sdk-sts-3.0.0
.jar aliyun-java-sdk-sts-3.0.0.jar
[root@cdh-master lib]# ln -s ../../../jars/aliyun-sdk-oss-3.4.1.jar
aliyun-sdk-oss-3.4.1.jar
[root@cdh-master lib]# ln -s ../../../jars/httpclient-4.5.2.jar
httpclient-4.5.2.jar
[root@cdh-master lib]# ln -s ../../../jars/httpcore-4.4.4.jar
httpcore-4.4.4.jar
[root@cdh-master lib]# ln -s ../../../jars/jdom-1.1.jar jdom-1.1.jar
```

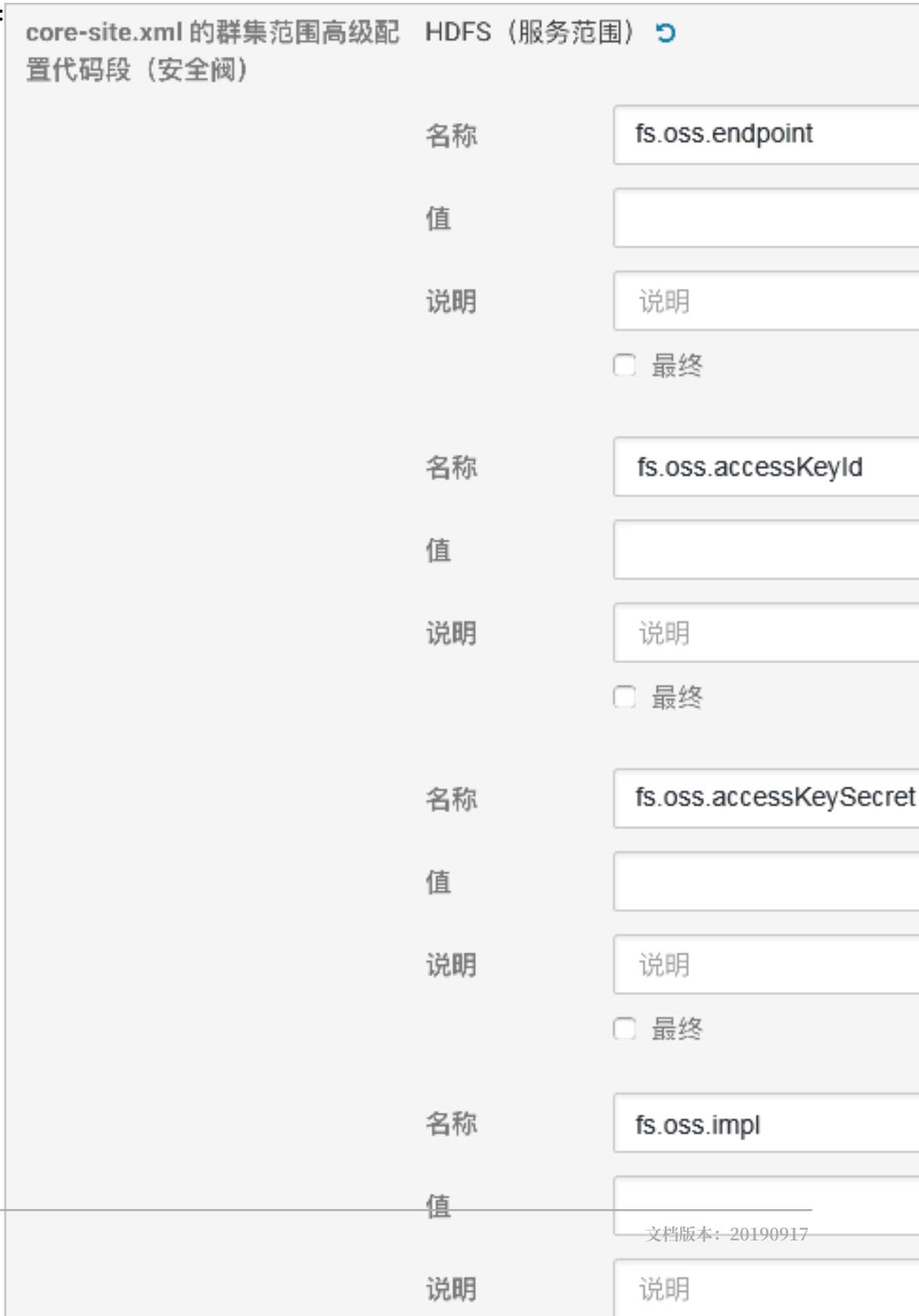
5. **进入 Resurce Manager 部署节点的***${CDH_HOME}/lib/hadoop-yarn/bin/***目录，将***yarn* **中的***CLASSPATH=${CLASSPATH}:$HADOOP_YARN_HOME/${YARN_DIR}/* CLASSPATH=${CLASSPATH}:$HADOOP_YARN_HOME/${YARN_LIB_JARS_DIR}/****替换为***CLASSPATH=$HADOOP_YARN_HOME/${YARN_DIR}/*:${CLASSPATH}CLASSPATH=$HADOOP_YARN_HOME/${YARN_LIB_JARS_DIR}/*:${CLASSPATH}#*

6. **进入到 Resurce Manager 部署节点的***${CDH_HOME}/lib/hadoop-yarn/lib***目录，执行如下命令：**

```
[root@cdh-master lib]# ln -s ../../../jars/httpclient-4.2.5.jar
httpclient-4.2.5.jar
```

```
[root@cdh-master lib]# ln -s ../../../jars/httpcore-4.2.5.jar
httpcore-4.2.5.jar
```

7. **通过集群管理工具 CM 来增加配置。若没有 CM 管理的集群，可以修改*core-site.xml*。以CM为例，需要增加如下配置:**



配置项	值
fs.oss.endpoint	填写需要连接的 OSS 的 Endpoint。 例如：oss-cn-zhangjiakou-internal.aliyuncs.com
fs.oss.accessKeyId	填写 OSS 的 AccessKeyId。
fs.oss.accessKeySecret	填写 OSS 的 AccessKeySecret。
fs.oss.impl	Hadoop OSS 文件系统实现类。目前固定为：org.apache.hadoop.fs.aliyun.oss.AliyunOSSFileSystem
fs.oss.buffer.dir	填写临时文件目录。 建议值：/tmp/oss
fs.oss.connection.secure.enabled	是否开启 HTTPS。开启 HTTPS 会影响性能。 建议值：false
fs.oss.connection.maximum	与 OSS 的连接数 建议值：2048

更多参数解释可参考 Hadoop-Aliyun module。

8. 根据 CM 提示重启集群。
9. 测试读写 OSS：
   - 测试读：

     ```
     hadoop fs -ls oss://${your-bucket-name}/
     ```

   - 测试写：

     ```
     hadoop fs -mkdir oss://${your-bucket-name}/hadoop-test
     ```

     若测试可以读写 OSS，则配置成功；若无法读写 OSS，请检查配置。

步骤二：配置 Impala 对 OSS 的支持

Impala 可以直接查询存储在 HDFS 的数据，在 CDH5 支持 OSS 后，就可以直接查询存储在 OSS 的数据。OSS SDK 要求这两个组件的版本较高，所以需要在所有部署 Impala 的节点执行以下操作：

1. 进入*${CDH_HOME}/lib/impala/lib*，执行如下命令：

```
[root@cdh-master lib]# rm -f httpclient-4.2.5.jar httpcore-4.2.5.jar
[root@cdh-master lib]# ln -s ../../../jars/httpclient-4.5.2.jar
httpclient-4.5.2.jar
[root@cdh-master lib]# ln -s ../../../jars/httpcore-4.4.4.jar
httpcore-4.4.4.jar
[root@cdh-master lib]# ln -s ../../../jars/hadoop-aliyun-2.6.0-cdh5.
14.4.jar hadoop-aliyun.jar
[root@cdh-master lib]# ln -s ../../../jars/aliyun-java-sdk-core-3.4.
0.jar aliyun-java-sdk-core-3.4.0.jar
[root@cdh-master lib]# ln -s ../../../jars/aliyun-java-sdk-ecs-4.2.0
.jar aliyun-java-sdk-ecs-4.2.0.jar
[root@cdh-master lib]# ln -s ../../../jars/aliyun-java-sdk-ram-3.0.0
.jar aliyun-java-sdk-ram-3.0.0.jar
[root@cdh-master lib]# ln -s ../../../jars/aliyun-java-sdk-sts-3.0.0
.jar aliyun-java-sdk-sts-3.0.0.jar
[root@cdh-master lib]# ln -s ../../../jars/aliyun-sdk-oss-3.4.1.jar
aliyun-sdk-oss-3.4.1.jar
[root@cdh-master lib]# ln -s ../../../jars/jdom-1.1.jar jdom-1.1.jar
```

2. 进入到*${CDH_HOME}/bin*目录，修改*impalad*、*statestored*、*catalogd*三个文件，在文件最后一行*exec*命令前，增加如下内容：

```
export CLASSPATH=$CLASSPATH:${IMPALA_HOME}/lib/httpclient-4.5.2.jar
:${IMPALA_HOME}/lib/httpcore-4.4.4.jar:${IMPALA_HOME}/lib/hadoop
-aliyun.jar:${IMPALA_HOME}/lib/aliyun-java-sdk-core-3.4.0.jar:${
IMPALA_HOME}/lib/aliyun-java-sdk-ecs-4.2.0.jar:${IMPALA_HOME}/lib/
aliyun-java-sdk-ram-3.0.0.jar:${IMPALA_HOME}/lib/aliyun-java-sdk-sts
-3.0.0.jar:${IMPALA_HOME}/lib/aliyun-sdk-oss-3.4.1.jar:${IMPALA_HOME
}/lib/jdom-1.1.jar
```

3. 重启所有节点的 impala 相关进程，之后 impala 就可以查询OSS的数据了。

验证配置

TPC-DS 的 benchmark 有一张表为*call_center*，假设存储在 OSS 中，我们可以创建一个外部表指向它，并且查询这张表根据*cc_country*分组分别有多少条记录。

```
[root@cdh-master ~]# impala-shell -i cdh-slave01:21000
Starting Impala Shell without Kerberos authentication
Connected to cdh-slave01:21000
Server version: impalad version 2.11.0-cdh5.14.4 RELEASE (build20e63
5646a13347800fad36a7d0b1da25ab32404)
***********************************************************************
Welcome to the Impala shell.
(Impala Shell v2.11.0-cdh5.14.4 (20e6356) built on Tue Jun 1203:43:08
PDT 2018)

The HISTORY command lists all shell commands in chronological order.
***********************************************************************
[cdh-slave01:21000] > droptableifexists call_center;
Query: droptableifexists call_center
[cdh-slave01:21000] >
[cdh-slave01:21000] > createexternaltable call_center(
                    >       cc_call_center_sk         bigint
                    > ,     cc_call_center_id         string
                    > ,     cc_rec_start_date        string
                    > ,     cc_rec_end_date          string
```

```
                    > ,     cc_closed_date_sk         bigint
                    > ,     cc_open_date_sk           bigint
                    > ,     cc_name                   string
                    > ,     cc_class                  string
                    > ,     cc_employees              int
                    > ,     cc_sq_ft                  int
                    > ,     cc_hours                  string
                    > ,     cc_manager                string
                    > ,     cc_mkt_id                 int
                    > ,     cc_mkt_class              string
                    > ,     cc_mkt_desc               string
                    > ,     cc_market_manager         string
                    > ,     cc_division               int
                    > ,     cc_division_name          string
                    > ,     cc_company                int
                    > ,     cc_company_name           string
                    > ,     cc_street_number          string
                    > ,     cc_street_name            string
                    > ,     cc_street_type            string
                    > ,     cc_suite_number           string
                    > ,     cc_city                   string
                    > ,     cc_county                 string
                    > ,     cc_state                  string
                    > ,     cc_zip                    string
                    > ,     cc_country                string
                    > ,     cc_gmt_offset             double
                    > ,     cc_tax_percentage         double
                    > )
                    > rowformatdelimitedfieldsterminatedby'|'
                    > location 'oss://${your-bucket-name}/call_center
';
Query: createexternaltable call_center(
      cc_call_center_sk         bigint
,     cc_call_center_id         string
,     cc_rec_start_date        string
,     cc_rec_end_date          string
,     cc_closed_date_sk         bigint
,     cc_open_date_sk           bigint
,     cc_name                   string
,     cc_class                  string
,     cc_employees              int
,     cc_sq_ft                  int
,     cc_hours                  string
,     cc_manager                string
,     cc_mkt_id                 int
,     cc_mkt_class              string
,     cc_mkt_desc               string
,     cc_market_manager         string
,     cc_division               int
,     cc_division_name          string
,     cc_company                int
,     cc_company_name           string
,     cc_street_number          string
,     cc_street_name            string
,     cc_street_type            string
,     cc_suite_number           string
,     cc_city                   string
,     cc_county                 string
,     cc_state                  string
,     cc_zip                    string
,     cc_country                string
,     cc_gmt_offset             double
,     cc_tax_percentage         double
)
```

```
rowformatdelimitedfieldsterminatedby'|'
location 'oss://${your-bucket-name}/call_center'
Fetched 0row(s) in0.07s

[cdh-slave01:21000] > select cc_country, count(*) from call_center
groupby cc_country;
Query: select cc_country, count(*) from call_center groupby cc_country
Query submitted at: 2018-10-2816:21:13 (Coordinator: http://cdh-
slave01:25000)
Query progress can be monitored at: http://cdh-slave01:25000/
query_plan?query_id=fb4e09977145f367:3bdfe4d600000000
+---------------+----------+
| cc_country    | count(*) |
+---------------+----------+
| United States | 30       |
+---------------+----------+
Fetched 1 row(s) in 4.71s
```

参考文档

关于 Hadoop 更多内容，可参考：Hadoop 支持集成 OSS。

您也可以通过阿里云 EMR 访问 OSS。阿里云 EMR 基于开源生态，包括 Hadoop、Spark、Kafka、Flink、Storm 等组件，为您提供集群、作业、数据管理等服务的一站式企业大数据平台，并无缝支持 OSS。阿里云 EMR 与 OSS 紧密结合，针对开源生态访问 OSS，有多项技术优化，详情可参考 EMR 产品介绍。

5.7 Apache Impala（CDH6）查询OSS数据

CDH（Cloudera's Distribution, including Apache Hadoop）是众多 Hadoop 发行版本中的一种。CDH的最新版本是6.0.1，支持Hadoop 3.0.0，本文介绍如何使 CDH6 的相关组件（Hadoop、Hive、Spark、Impala 等）能够查询 OSS。

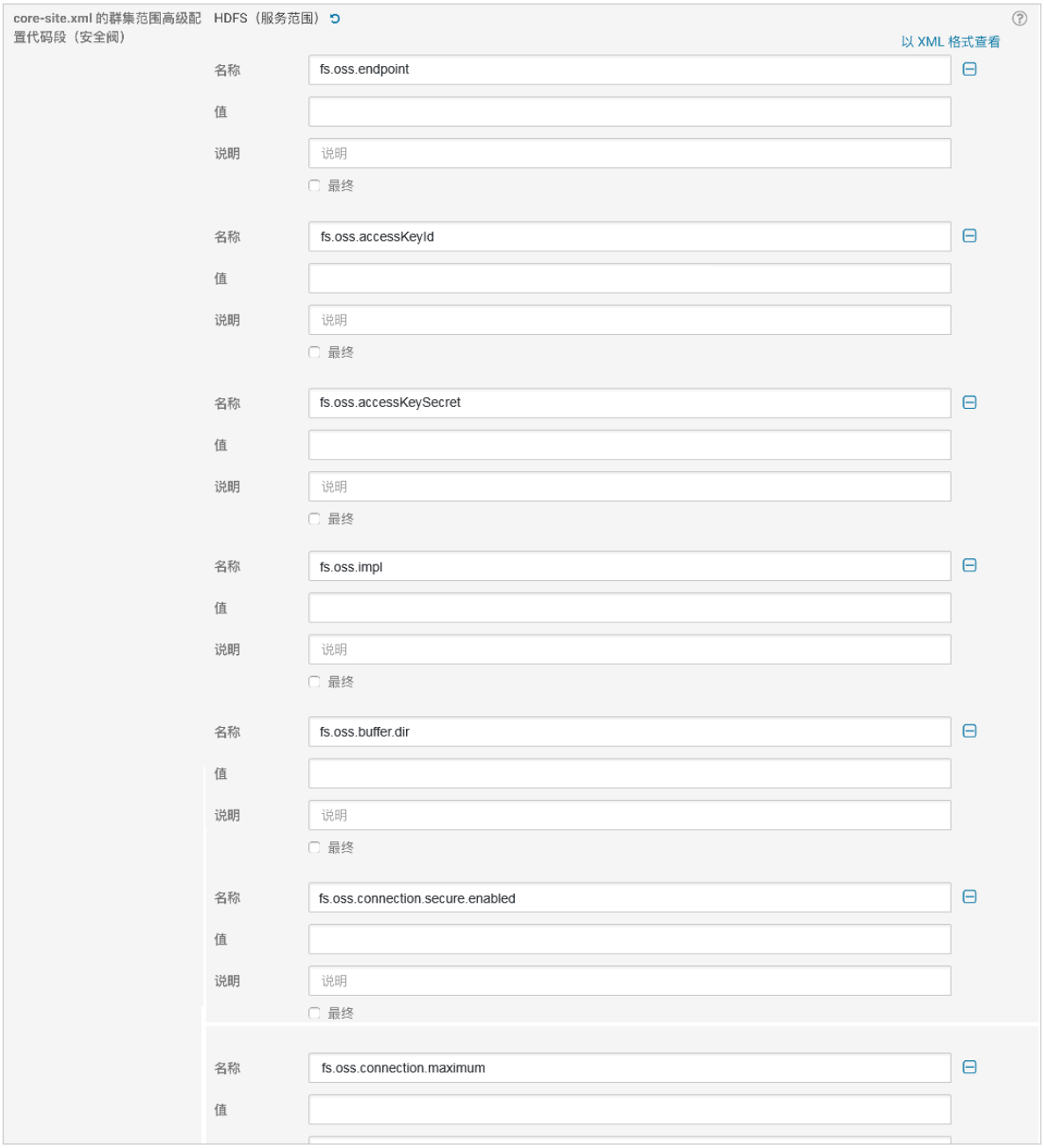
准备工作

您需要拥有一个已搭建好的 CDH6 集群。若没有已搭建好的 CDH6 集群，您需要参考官方文档，先搭建一个 CDH6 集群，本文演示版本为 CDH6.0.1 版本。

步骤一：增加 OSS 配置

1. 通过集群管理工具 CM 来增加配置。若没有 CM 管理的集群，可以修改`core-site.xml`。

   以CM为例，需要增加如下配置：



配置项	值
fs.oss.endpoint	填写需要连接的 OSS 的 Endpoint。 例如：oss-cn-zhangjiakou-internal.aliyuncs.com
fs.oss.accessKeyId	填写OSS的 AccessKeyId。
fs.oss.accessKeySecret	填写OSS的 AccessKeySecret。

配置项	值
fs.oss.impl	Hadoop OSS文件系统实现类。目前固定为：org.apache.hadoop.fs.aliyun.oss.AliyunOSSFileSystem
fs.oss.buffer.dir	填写临时文件目录。 建议值：/tmp/oss
fs.oss.connection.secure.enabled	是否开启HTTPS。开启HTTPS会影响性能。 建议值：false
fs.oss.connection.maximum	与 OSS 的连接数 建议值：2048

更多参数解释可参考 Hadoop-Aliyun module。

2. 根据 CM 提示重启集群。
3. 测试读写 OSS：
   - 测试读：

     ```
     hadoop fs -ls oss://${your-bucket-name}/
     ```

   - 测试写：

     ```
     hadoop fs -mkdir oss://${your-bucket-name}/hadoop-test
     ```

     若测试可以读写 OSS，则配置成功；若无法读写 OSS，请检查配置。

     说明：
     本文中所有 *${}* 的内容为环境变量，请根据您实际的环境修改。

步骤二：配置 Apache Impala 对 OSS 的支持

虽然 CDH6 支持 OSS，但是它里面的 Impala 组件却没有将 OSS 的支持放到它的 CLASSPATH 里面去，因此您需要在所有的 Impala 节点执行如下操作：

1. 进入*${CDH_HOME}/lib/impala*目录，创建符号链接：

```
[root@cdh-master impala]# cd lib/
[root@cdh-master lib]# ln -s ../../../jars/hadoop-aliyun-3.0.0-cdh6.
0.1.jar hadoop-aliyun.jar
[root@cdh-master lib]# ln -s ../../../jars/aliyun-sdk-oss-2.8.3.jar
aliyun-sdk-oss-2.8.3.jar
```

```
[root@cdh-master lib]# ln -s ../../../jars/jdom-1.1.jar jdom-1.1.jar
```

2. 进入*${CDH_HOME}/bin*目录，修改*impalad*、*statestored*、*catalogd*这三个文件，在文件最后一行*exec*命令前增加如下内容：

```
export CLASSPATH=${CLASSPATH}:${IMPALA_HOME}/lib/hadoop-aliyun.jar:
${IMPALA_HOME}/lib/aliyun-sdk-oss-2.8.3.jar:${IMPALA_HOME}/lib/jdom-
1.1.jar
```

3. 重启所有节点的 impala 相关进程。

验证配置

尝试使用 Apache Impala 查询 OSS，并且运行 TPC-DS 的查询语句。相关介绍请参考Apache Impala 的介绍和TPC-DS查询。

TPC-DS 基于 Hive QL，因为 Hive QL与 Impala SQL兼容性比较高，所以我们用这个作为实验案例（有几个 query 因为兼容性问题不能运行）：

1. 生成 sample 数据到 OSS：

```
[root@cdh-master ~]# git clone https://github.com/hortonworks/hive-
testbench.git
[root@cdh-master ~]# cd hive-testbench
[root@cdh-master hive-testbench]# git checkout hive14
[root@cdh-master hive-testbench]# ./tpcds-build.sh
[root@cdh-master hive-testbench]# FORMAT=textfile ./tpcds-setup.sh
50 oss://{your-bucket-name}/
```

2. 建表。这个 benchmark 的建表语句与 Apache Impala 的建表语句兼容，复制*ddl-tpcds/text/alltables.sql*中的建表语句，修改*${LOCATION}*即可，例如：

```
[root@cdh-master hive-testbench]# impala-shell -i cdh-slave01 -d
default
Starting Impala Shell without Kerberos authentication
Connected to cdh-slave01:21000
Server version: impalad version 3.0.0-cdh6.0.1 RELEASE(build9a74a
5053de5f7b8dd983802e6d75e58d31472db)
***********************************************************************
Welcome to the Impala shell.
(Impala Shell v3.0.0-cdh6.0.1(9a74a50) built on Wed Sep 1911:27:37
PDT 2018)

Want to know what versionof Impala you're connected to? Run the
VERSION command to
find out!
***********************************************************************
Query: use `default`
[cdh-slave01:21000] default> create external table call_center(
                           >       cc_call_center_sk         bigint
                           > ,     cc_call_center_id         string
                           > ,     cc_rec_start_date        string
                           > ,     cc_rec_end_date          string
                           > ,     cc_closed_date_sk         bigint
                           > ,     cc_open_date_sk           bigint
                           > ,     cc_name                   string
                           > ,     cc_class                  string
```

```
                            > ,     cc_employees                int
                            > ,     cc_sq_ft                    int
                            > ,     cc_hours                    string
                            > ,     cc_manager                  string
                            > ,     cc_mkt_id                   int
                            > ,     cc_mkt_class                string
                            > ,     cc_mkt_desc                 string
                            > ,     cc_market_manager           string
                            > ,     cc_division                 int
                            > ,     cc_division_name            string
                            > ,     cc_company                  int
                            > ,     cc_company_name             string
                            > ,     cc_street_number            string
                            > ,     cc_street_name              string
                            > ,     cc_street_type              string
                            > ,     cc_suite_number             string
                            > ,     cc_city                     string
                            > ,     cc_county                   string
                            > ,     cc_state                    string
                            > ,     cc_zip                      string
                            > ,     cc_country                  string
                            > ,     cc_gmt_offset               double
                            > ,     cc_tax_percentage           double
                            > )
                            > row format delimited fields terminated
by '|'
                            > location 'oss://{your-bucket-name}/50/
call_center';
Query: create external table call_center(
      cc_call_center_sk         bigint
,     cc_call_center_id         string
,     cc_rec_start_date        string
,     cc_rec_end_date          string
,     cc_closed_date_sk         bigint
,     cc_open_date_sk           bigint
,     cc_name                   string
,     cc_class                  string
,     cc_employees              int
,     cc_sq_ft                  int
,     cc_hours                  string
,     cc_manager                string
,     cc_mkt_id                 int
,     cc_mkt_class              string
,     cc_mkt_desc               string
,     cc_market_manager         string
,     cc_division               int
,     cc_division_name          string
,     cc_company                int
,     cc_company_name           string
,     cc_street_number          string
,     cc_street_name            string
,     cc_street_type            string
,     cc_suite_number           string
,     cc_city                   string
,     cc_county                 string
,     cc_state                  string
,     cc_zip                    string
,     cc_country                string
,     cc_gmt_offset             double
,     cc_tax_percentage         double
)
row format delimited fields terminated by '|'
location 'oss://{your-bucket-name}/50/call_center'
+-------------------------+
```

```
| summary                  |
+--------------------------+
| Table has been created. |
+--------------------------+
Fetched 1 row(s) in 4.10s
```

3. 检查建表内容：

```
[cdh-slave01:21000] default> showtables;
Query: showtables
+------------------------+
| name                   |
+------------------------+
| call_center            |
| catalog_page           |
| catalog_returns        |
| catalog_sales          |
| customer               |
| customer_address       |
| customer_demographics  |
| date_dim               |
| household_demographics |
| income_band            |
| inventory              |
| item                   |
| promotion              |
| reason                 |
| ship_mode              |
| store                  |
| store_returns          |
| store_sales            |
| time_dim               |
| warehouse              |
| web_page               |
| web_returns            |
| web_sales              |
| web_site               |
+------------------------+
Fetched 24 row(s) in 0.03s
```

4. 在 Impala 上执行 TPC-DS 的查询，已经可以正常查询 OSS 的数据。

   a. 查询的 SQL 文件在*sample-queries-tpcds*目录下：

```
[root@cdh-master hive-testbench]# ls sample-queries-tpcds
query12.sql  query20.sql  query27.sql  query39.sql  query46.sql
query54.sql  query64.sql  query71.sql  query7.sql   query87.sql
query93.sql  README.md
query13.sql  query21.sql  query28.sql  query3.sql   query48.sql
query55.sql  query65.sql  query72.sql  query80.sql  query88.sql
query94.sql  testbench.settings
query15.sql  query22.sql  query29.sql  query40.sql  query49.sql
query56.sql  query66.sql  query73.sql  query82.sql  query89.sql
query95.sql  testbench-withATS.settings
query17.sql  query24.sql  query31.sql  query42.sql  query50.sql
query58.sql  query67.sql  query75.sql  query83.sql  query90.sql
query96.sql
query18.sql  query25.sql  query32.sql  query43.sql  query51.sql
query60.sql  query68.sql  query76.sql  query84.sql  query91.sql
query97.sql
```

```
query19.sql  query26.sql  query34.sql  query45.sql  query52.sql
query63.sql  query70.sql  query79.sql  query85.sql  query92.sql
query98.sql
```

b. 执行 TPC-DS 查询*query13.sql*：

```
[root@cdh-master hive-testbench]# impala-shell -i cdh-slave01 -d
default -f sample-queries-tpcds/query13.sql
Starting Impala Shell without Kerberos authentication
Connected to cdh-slave01:21000
Server version: impalad version 3.0.0-cdh6.0.1 RELEASE(build9a74a
5053de5f7b8dd983802e6d75e58d31472db)
Query: use`default`Query: selectavg(ss_quantity)
       ,avg(ss_ext_sales_price)
       ,avg(ss_ext_wholesale_cost)
       ,sum(ss_ext_wholesale_cost)
 from store_sales
     ,store
     ,customer_demographics
     ,household_demographics
     ,customer_address
     ,date_dim
 where store.s_store_sk = store_sales.ss_store_sk
 and  store_sales.ss_sold_date_sk = date_dim.d_date_sk and
date_dim.d_year = 2001and((store_sales.ss_hdemo_sk=household_
demographics.hd_demo_sk
  and customer_demographics.cd_demo_sk = store_sales.ss_cdemo_sk
  and customer_demographics.cd_marital_status = 'M'and customer_d
emographics.cd_education_status = '4 yr Degree'and store_sales.
ss_sales_price between100.00and150.00and household_demographics.
hd_dep_count = 3
     )or
    (store_sales.ss_hdemo_sk=household_demographics.hd_demo_sk
  and customer_demographics.cd_demo_sk = store_sales.ss_cdemo_sk
  and customer_demographics.cd_marital_status = 'D'and customer_d
emographics.cd_education_status = 'Primary'and store_sales.
ss_sales_price between50.00and100.00and household_demographics.
hd_dep_count = 1
     ) or
    (store_sales.ss_hdemo_sk=household_demographics.hd_demo_sk
  and customer_demographics.cd_demo_sk = ss_cdemo_sk
  and customer_demographics.cd_marital_status = 'U'and customer_d
emographics.cd_education_status = 'Advanced Degree'and store_sale
s.ss_sales_price between150.00and200.00and household_demographics.
hd_dep_count = 1
     ))
 and((store_sales.ss_addr_sk = customer_address.ca_address_sk
  and customer_address.ca_country = 'United States'and customer_a
ddress.ca_state in('KY', 'GA', 'NM')
  and store_sales.ss_net_profit between100and200
     ) or
    (store_sales.ss_addr_sk = customer_address.ca_address_sk
  and customer_address.ca_country = 'United States'and customer_a
ddress.ca_state in('MT', 'OR', 'IN')
  and store_sales.ss_net_profit between150and300
     ) or
    (store_sales.ss_addr_sk = customer_address.ca_address_sk
  and customer_address.ca_country = 'United States'and customer_a
ddress.ca_state in('WI', 'MO', 'WV')
  and store_sales.ss_net_profit between50and250
     ))
Query submitted at: 2018-10-3011:44:47(Coordinator: http://cdh-
slave01:25000)
```

```
Query progress can be monitored at: http://cdh-slave01:25000/
query_plan?query_id=ff4b3157eddfc3c4:8a31c6500000000
+-------------------+-----------------------
+-------------------------+--------------------------+
| avg(ss_quantity)  | avg(ss_ext_sales_price) | avg(ss_ext_who
lesale_cost) | sum(ss_ext_wholesale_cost) |
+-------------------+-----------------------
+-------------------------+--------------------------+
| 30.87106918238994 | 2352.642327044025       | 2162.600911949685
         | 687707.09                |
+-------------------+-----------------------
+-------------------------+--------------------------+
Fetched 1 row(s) in 353.16s
```

本次查询涉及到6张表，*store*、*store_sales*、*customer_demographics*、*household_demographics*、*customer_address*、*date_dim*:

```
[cdh-slave01:21000] default> showtable STATS store;
Query: showtable STATS store
+-------+--------+---------+-------------
+-------------------+--------+-------------------
+-----------------------------------------+
| #Rows | #Files | Size    | Bytes Cached | CacheReplication |
Format | Incremental stats | Location
   |
+-------+--------+---------+-------------
+-------------------+--------+-------------------
+-----------------------------------------+
| -1    | 1      | 37.56KB | NOT CACHED   | NOT CACHED        | TEXT
   | false             | oss://{your-bucket-name}/50/store |
+-------+--------+---------+-------------
+-------------------+--------+-------------------
+-----------------------------------------+
Fetched 1 row(s) in 0.01s
[cdh-slave01:21000] default> showtable STATS store_sales;
Query: showtable STATS store_sales
+-------+--------+---------+-------------
+-------------------+--------+-------------------
+-------------------------------------------------+
| #Rows | #Files | Size    | Bytes Cached | CacheReplication |
Format | Incremental stats | Location
         |
+-------+--------+---------+-------------
+-------------------+--------+-------------------
+-------------------------------------------------+
| -1    | 50     | 18.75GB | NOT CACHED   | NOT CACHED        | TEXT
   | false             | oss://{your-bucket-name}/50/store_sales |
+-------+--------+---------+-------------
+-------------------+--------+-------------------
+-------------------------------------------------+
Fetched 1 row(s) in 0.01s
[cdh-slave01:21000] default> showtable STATS customer_demographics;
Query: showtable STATS customer_demographics
+-------+--------+---------+-------------
+-------------------+--------+-------------------
+-----------------------------------------------------------+
| #Rows | #Files | Size    | Bytes Cached | CacheReplication |
Format | Incremental stats | Location
                   |
+-------+--------+---------+-------------
+-------------------+--------+-------------------
+-----------------------------------------------------------+
```

```
| -1    | 50     | 76.92MB | NOT CACHED   | NOT CACHED        |
TEXT   | false             | oss://{your-bucket-name}/50/customer_d
emographics |
+-------+--------+---------+--------------
+-------------------+--------+-------------------
+-----------------------------------------------------------+
Fetched 1 row(s) in 0.01s
[cdh-slave01:21000] default> showtable STATS household_demographics;
Query: showtable STATS household_demographics
+-------+--------+----------+--------------
+-------------------+--------+-------------------
+-----------------------------------------------------------+
| #Rows | #Files | Size     | Bytes Cached | CacheReplication |
Format | Incremental stats | Location
                   |
+-------+--------+----------+--------------
+-------------------+--------+-------------------
+-----------------------------------------------------------+
| -1    | 1      | 148.10KB | NOT CACHED   | NOT CACHED        |
TEXT   | false             | oss://{your-bucket-name}/50/household_
demographics |
+-------+--------+----------+--------------
+-------------------+--------+-------------------
+-----------------------------------------------------------+
Fetched 1 row(s) in 0.01s
[cdh-slave01:21000] default> showtable STATS customer_address;
Query: showtable STATS customer_address
+-------+--------+---------+--------------
+-------------------+--------+-------------------
+-----------------------------------------------------+
| #Rows | #Files | Size    | Bytes Cached | CacheReplication |
Format | Incremental stats | Location
              |
+-------+--------+---------+--------------
+-------------------+--------+-------------------
+-----------------------------------------------------+
| -1    | 1      | 40.54MB | NOT CACHED   | NOT CACHED        |
TEXT   | false             | oss://{your-bucket-name}/50/customer_a
ddress |
+-------+--------+---------+--------------
+-------------------+--------+-------------------
+-----------------------------------------------------+
Fetched 1 row(s) in 0.01s
[cdh-slave01:21000] default> showtable STATS date_dim;
Query: showtable STATS date_dim
+-------+--------+--------+--------------
+-------------------+--------+-------------------
+-----------------------------------------------+
| #Rows | #Files | Size   | Bytes Cached | CacheReplication | Format
 | Incremental stats | Location
 |
+-------+--------+--------+--------------
+-------------------+--------+-------------------
+-----------------------------------------------+
| -1    | 1      | 9.84MB | NOT CACHED   | NOT CACHED        | TEXT
  | false             | oss://{your-bucket-name}/50/date_dim |
+-------+--------+--------+--------------
+-------------------+--------+-------------------
+-----------------------------------------------+
```

```
Fetched 1 row(s) in 0.01s
```

参考文档

更多内容，可参考以下文档：

- Hadoop 支持集成 OSS
- Hadoop
- CDH5 Hadoop如何读写OSS
- HDP2.6 Hadoop如何读写OSS

您也可以通过阿里云 EMR 访问 OSS。阿里云 EMR 基于开源生态，包括 Hadoop、Spark、Kafka、Flink、Storm 等组件，为您提供集群、作业、数据管理等服务的一站式企业大数据平台，并无缝支持 OSS。阿里云 EMR 与 OSS 紧密结合，针对开源生态访问 OSS，有多项技术优化，详情可参考 EMR 产品介绍。

5.8 Spark使用OSS Select加速数据查询

本文基于Apache Impala(CDH6) 处理 OSS 数据搭建的 CDH6 集群及配置，介绍如何配置 Spark 使用 OSS Select 加速数据查询。

说明：

本文中所有 *${}* 的内容为环境变量，请根据您实际的环境修改。

步骤一：配置 Spark 支持读写 OSS

由于默认 Spark 并没有将 OSS 的支持包放到它的 CLASSPATH 里面，所以我们需要配置 Spark 支持读写 OSS。您需要在所有的 CDH 节点执行以下操作：

1. 进入到*${CDH_HOME}/lib/spark*目录，执行如下命令：

```
[root@cdh-master spark]# cd jars/
[root@cdh-master jars]# ln -s ../../../jars/hadoop-aliyun-3.0.0-cdh6
.0.1.jar hadoop-aliyun.jar
[root@cdh-master jars]# ln -s ../../../jars/aliyun-sdk-oss-2.8.3.jar
 aliyun-sdk-oss-2.8.3.jar
[root@cdh-master jars]# ln -s ../../../jars/jdom-1.1.jar jdom-1.1.
jar
```

2. 进入到*${CDH_HOME}/lib/spark*目录，运行一个查询：

```
[root@cdh-master spark]# ./bin/spark-shell
WARNING: User-defined SPARK_HOME (/opt/cloudera/parcels/CDH-6.0.1-
1.cdh6.0.1.p0.590678/lib/spark) overrides detected (/opt/cloudera/
parcels/CDH/lib/spark).
WARNING: Running spark-class from user-defined location.
Setting default log level to "WARN".
```

```
To adjust logging level use sc.setLogLevel(newLevel). For SparkR,
use setLogLevel(newLevel).
Spark context Web UI available at http://x.x.x.x:4040
Spark context available as 'sc' (master = yarn, app id = applicatio
n_1540878848110_0004).
Spark session available as'spark'.
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 2.2.0-cdh6.0.1
      /_/

Using Scala version 2.11.8 (Java HotSpot(TM) 64-Bit Server VM, Java
1.8.0_152)
Type in expressions to have them evaluated.
Type :help for more information.

scala> val myfile = sc.textFile("oss://{your-bucket-name}/50/
store_sales")
myfile: org.apache.spark.rdd.RDD[String] = oss://{your-bucket-name}/
50/store_sales MapPartitionsRDD[1] at textFile at <console>:24

scala> myfile.count()
res0: Long = 144004764

scala> myfile.map(line => line.split('|')).filter(_(0).toInt >=
2451262).take(3)
res15: Array[Array[String]] = Array(Array(2451262, 71079, 20359,
154660, 284233, 6206, 150579, 46, 512, 2160001, 84, 6.94, 11.38, 9.
33, 681.83, 783.72, 582.96, 955.92, 5.09, 681.83, 101.89, 106.98, -
481.07), Array(2451262, 71079, 26863, 154660, 284233, 6206, 150579
, 46, 345, 2160001, 12, 67.82, 115.29, 25.36, 0.00, 304.32, 813.84,
1383.48, 21.30, 0.00, 304.32, 325.62, -509.52), Array(2451262, 71079
, 55852, 154660, 284233, 6206, 150579, 46, 243, 2160001, 74, 32.41,
34.67, 1.38, 0.00, 102.12, 2398.34, 2565.58, 4.08, 0.00, 102.12, 106
.20, -2296.22))

scala> myfile.map(line => line.split('|')).filter(_(0) >= "2451262
").saveAsTextFile("oss://{your-bucket-name}/spark-oss-test.1")
```

可正常执行查询，则部署生效。

步骤二：配置 Spark 支持 OSS Select

OSS Select目前已经在深圳区域实现商业化，以下内容基于 oss-cn-shenzhen.aliyuncs.com 这个 OSS EndPoint 来进行。您需要在所有的 CDH 节点执行以下操作：

1. **下载OSS Select 的 Spark 支持包到*${CDH_HOME}/jars*目录下（目前该支持包还在测试中）。**
2. **解压支持包：**

```
[root@cdh-master jars]# tar -tvf spark-2.2.0-oss-select-0.1.0-
SNAPSHOT.tar.gz
drwxr-xr-x root/root 0 2018-10-30 17:59 spark-2.2.0-oss-select-0.1.0
-SNAPSHOT/
-rw-r--r-- root/root 26514 2018-10-30 16:11 spark-2.2.0-oss-select-0
.1.0-SNAPSHOT/stax-api-1.0.1.jar
```

```
-rw-r--r-- root/root 547584 2018-10-30 16:11 spark-2.2.0-oss-select-
0.1.0-SNAPSHOT/aliyun-sdk-oss-3.3.0.jar
-rw-r--r-- root/root 13277 2018-10-30 16:11 spark-2.2.0-oss-select-0
.1.0-SNAPSHOT/aliyun-java-sdk-sts-3.0.0.jar
-rw-r--r-- root/root 116337 2018-10-30 16:11 spark-2.2.0-oss-select-
0.1.0-SNAPSHOT/aliyun-java-sdk-core-3.4.0.jar
-rw-r--r-- root/root 215492 2018-10-30 16:11 spark-2.2.0-oss-select-
0.1.0-SNAPSHOT/aliyun-java-sdk-ram-3.0.0.jar
-rw-r--r-- root/root 67758 2018-10-30 16:11 spark-2.2.0-oss-select-0
.1.0-SNAPSHOT/jettison-1.1.jar
-rw-r--r-- root/root 57264 2018-10-30 16:11 spark-2.2.0-oss-select-0
.1.0-SNAPSHOT/json-20170516.jar
-rw-r--r-- root/root 890168 2018-10-30 16:11 spark-2.2.0-oss-select-
0.1.0-SNAPSHOT/jaxb-impl-2.2.3-1.jar
-rw-r--r-- root/root 458739 2018-10-30 16:11 spark-2.2.0-oss-select-
0.1.0-SNAPSHOT/jersey-core-1.9.jar
-rw-r--r-- root/root 147952 2018-10-30 16:11 spark-2.2.0-oss-select-
0.1.0-SNAPSHOT/jersey-json-1.9.jar
-rw-r--r-- root/root 788137 2018-10-30 16:11 spark-2.2.0-oss-select-
0.1.0-SNAPSHOT/aliyun-java-sdk-ecs-4.2.0.jar
-rw-r--r-- root/root 153115 2018-10-30 16:11 spark-2.2.0-oss-select-
0.1.0-SNAPSHOT/jdom-1.1.jar
-rw-r--r-- root/root 65437 2018-10-31 14:41 spark-2.2.0-oss-select-0
.1.0-SNAPSHOT/aliyun-oss-select-spark_2.11-0.1.0-SNAPSHOT.jar
```

3. 进入*${CDH_HOME}/lib/spark/jars*目录，执行如下命令：

```
[root@cdh-master jars]# pwd
/opt/cloudera/parcels/CDH/lib/spark/jars
[root@cdh-master jars]# rm -f aliyun-sdk-oss-2.8.3.jar
[root@cdh-master jars]# ln -s ../../../jars/aliyun-oss-select-
spark_2.11-0.1.0-SNAPSHOT.jar aliyun-oss-select-spark_2.11-0.1.0-
SNAPSHOT.jar
[root@cdh-master jars]# ln -s ../../../jars/aliyun-java-sdk-core-3.4
.0.jar aliyun-java-sdk-core-3.4.0.jar
[root@cdh-master jars]# ln -s ../../../jars/aliyun-java-sdk-ecs-4.2.
0.jar aliyun-java-sdk-ecs-4.2.0.jar
[root@cdh-master jars]# ln -s ../../../jars/aliyun-java-sdk-ram-3.0.
0.jar aliyun-java-sdk-ram-3.0.0.jar
[root@cdh-master jars]# ln -s ../../../jars/aliyun-java-sdk-sts-3.0.
0.jar aliyun-java-sdk-sts-3.0.0.jar
[root@cdh-master jars]# ln -s ../../../jars/aliyun-sdk-oss-3.3.0.jar
 aliyun-sdk-oss-3.3.0.jar
[root@cdh-master jars]# ln -s ../../../jars/jdom-1.1.jar jdom-1.1.
jar
```

对比测试

测试环境：使用 spark on yarn 进行对比测试，其中 Node Manager 节点是4个，每个节点最多可以运行4个 container，每个 container 配备的资源是 1 核 2GB 内存。

测试数据：共 630MB，包含 3 列，分别是姓名、公司和年龄。

```
ot@cdh-master jars]# hadoop fs -ls oss://select-test-sz/people/
Found 10 items
-rw-rw-rw-   1   63079930 2018-10-30 17:03 oss://select-test-sz/people
/part-00000
-rw-rw-rw-   1   63079930 2018-10-30 17:03 oss://select-test-sz/people
/part-00001
-rw-rw-rw-   1   63079930 2018-10-30 17:05 oss://select-test-sz/people
/part-00002
```

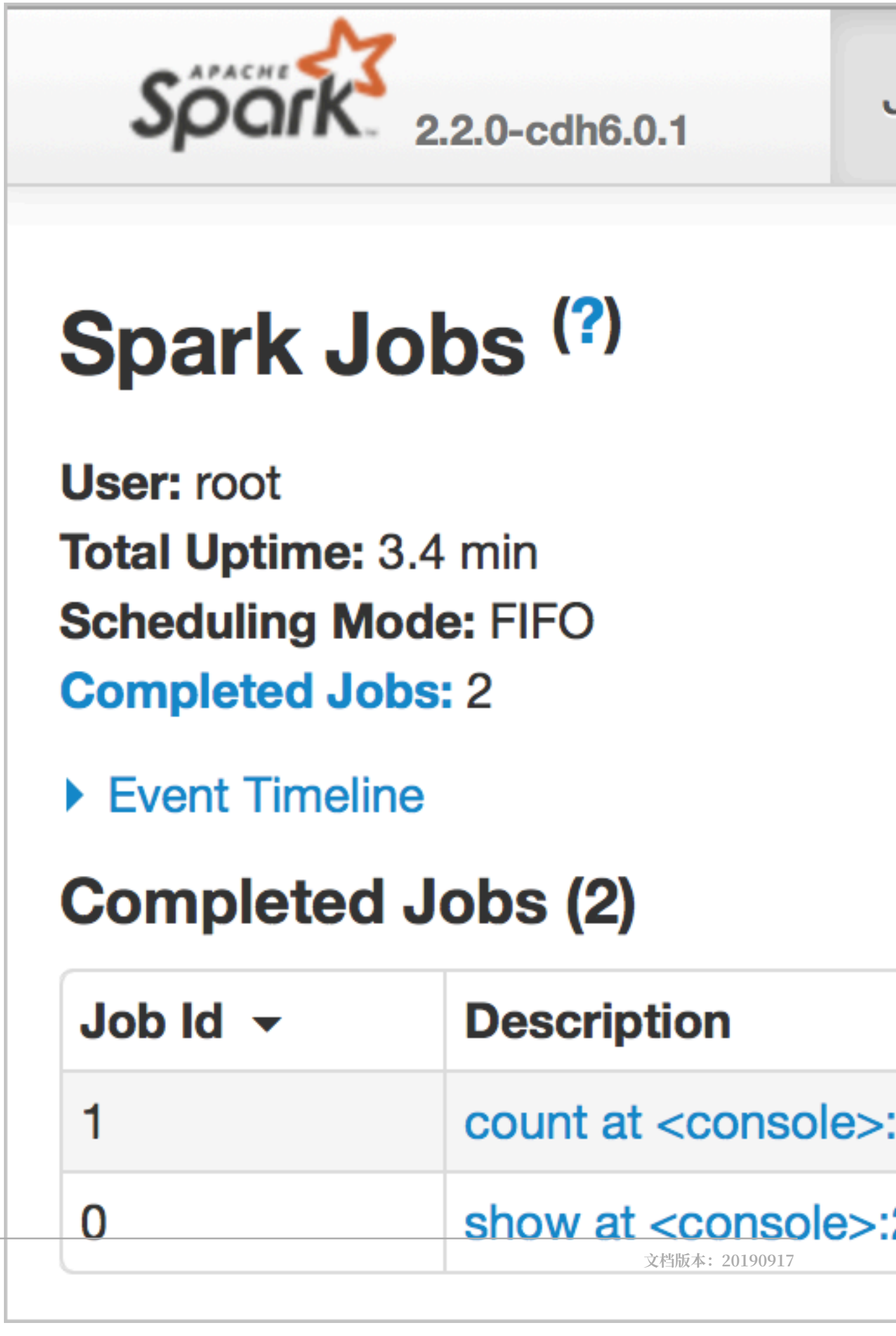
```
-rw-rw-rw-   1   63079930 2018-10-30 17:05 oss://select-test-sz/people
/part-00003
-rw-rw-rw-   1   63079930 2018-10-30 17:06 oss://select-test-sz/people
/part-00004
-rw-rw-rw-   1   63079930 2018-10-30 17:12 oss://select-test-sz/people
/part-00005
-rw-rw-rw-   1   63079930 2018-10-30 17:14 oss://select-test-sz/people
/part-00006
-rw-rw-rw-   1   63079930 2018-10-30 17:14 oss://select-test-sz/people
/part-00007
-rw-rw-rw-   1   63079930 2018-10-30 17:15 oss://select-test-sz/people
/part-00008
-rw-rw-rw-   1   63079930 2018-10-30 17:16 oss://select-test-sz/people
/part-00009
```

进入到*${CDH_HOME}/lib/spark/*，启动 spark-shell，分别测试使用 OSS Select 查询数据和不使用 OSS Select 查询数据：

```
[root@cdh-master spark]# ./bin/spark-shell
WARNING: User-defined SPARK_HOME (/opt/cloudera/parcels/CDH-6.0.1-
1.cdh6.0.1.p0.590678/lib/spark) overrides detected (/opt/cloudera/
parcels/CDH/lib/spark).
WARNING: Running spark-class from user-defined location.
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use
setLogLevel(newLevel).
Spark context Web UI available at http://x.x.x.x:4040
Spark context available as 'sc' (master = yarn, app id = applicatio
n_1540887123331_0008).
Spark session available as 'spark'.
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 2.2.0-cdh6.0.1
      /_/

Using Scala version 2.11.8 (Java HotSpot(TM) 64-Bit Server VM, Java 1.
8.0_152)
Type in expressions to have them evaluated.
Type :help for more information.

scala> val sqlContext = spark.sqlContext
sqlContext: org.apache.spark.sql.SQLContext = org.apache.spark.sql.
SQLContext@4bdef487

scala> sqlContext.sql("CREATE TEMPORARY VIEW people USING com.aliyun.
oss " +
     |   "OPTIONS (" +
     |   "oss.bucket 'select-test-sz', " +
     |   "oss.prefix 'people', " + // objects with this prefix belong
to this table
     |   "oss.schema 'name string, company string, age long'," + //
like 'column_a long, column_b string'
     |   "oss.data.format 'csv'," + // we only support csv now
     |   "oss.input.csv.header 'None'," +
     |   "oss.input.csv.recordDelimiter '\r\n'," +
     |   "oss.input.csv.fieldDelimiter ','," +
     |   "oss.input.csv.commentChar '#'," +
     |   "oss.input.csv.quoteChar '\"'," +
     |   "oss.output.csv.recordDelimiter '\n'," +
     |   "oss.output.csv.fieldDelimiter ','," +
```

```
     |   "oss.output.csv.quoteChar '\"'," +
     |   "oss.endpoint 'oss-cn-shenzhen.aliyuncs.com', " +
     |   "oss.accessKeyId 'Your Access Key Id', " +
     |   "oss.accessKeySecret 'Your Access Key Secret')")
res0: org.apache.spark.sql.DataFrame = []

scala>   val sql: String = "select count(*) from people where name
like 'Lora%'"
sql: String = select count(*) from people where name like 'Lora%'

scala>   sqlContext.sql(sql).show()
+--------+
|count(1)|
+--------+
|   31770|
+--------+

scala> val textFile = sc.textFile("oss://select-test-sz/people/")
textFile: org.apache.spark.rdd.RDD[String] = oss://select-test-sz/
people/ MapPartitionsRDD[8] at textFile at <console>:24

scala> textFile.map(line => line.split(',')).filter(_(0).startsWith("
Lora")).count()
res3: Long = 31770
```

从下图可看到：使用 OSS Select 查询数据耗时为 15s，不使用 OSS Select 查询数据耗时为 54s，使用 OSS Select 能大幅度加快查询速度。


APACHE Spark
2.2.0-cdh6.0.1
Spark Jobs (?)
User: root
Total Uptime: 3.4 min
Scheduling Mode: FIFO
Completed Jobs: 2
Event Timeline
Completed Jobs (2)
Job Id
Description
1
count at <console>:
0
show at <console>:

Spark 对接 OSS Select 支持包的实现(Preview)

通过扩展 Spark 的DataSource API可以实现 Spark 对接 OSS Select。通过实现 PrunedFilteredScan，可以把需要的列和过滤条件下推到 OSS Select 执行。目前这个支持包还在开发中。

- 定义的规范：

```
scala> sqlContext.sql("CREATE TEMPORARY VIEW people USING com.aliyun
.oss " +
     |   "OPTIONS (" +
     |   "oss.bucket 'select-test-sz', " +
     |   "oss.prefix 'people', " + // objects with this prefix
belong to this table
     |   "oss.schema 'name string, company string, age long'," + //
like 'column_a long, column_b string'
     |   "oss.data.format 'csv'," + // we only support csv now
     |   "oss.input.csv.header 'None'," +
     |   "oss.input.csv.recordDelimiter '\r\n'," +
     |   "oss.input.csv.fieldDelimiter ','," +
     |   "oss.input.csv.commentChar '#'," +
     |   "oss.input.csv.quoteChar '\"'," +
     |   "oss.output.csv.recordDelimiter '\n'," +
     |   "oss.output.csv.fieldDelimiter ','," +
     |   "oss.output.csv.quoteChar '\"'," +
     |   "oss.endpoint 'oss-cn-shenzhen.aliyuncs.com', " +
     |   "oss.accessKeyId 'Your Access Key Id', " +
     |   "oss.accessKeySecret 'Your Access Key Secret')")
```

字段	说明
oss.bucket	数据所在的 bucket
oss.prefix	拥有这个前缀的 Object 都属于定义的这个 TEMPORARY VIEW。
oss.schema	这个 TEMPORARY VIEW 的 schema，目前通过字符串指定，后续会通过一个文件来指定 schema。
oss.data.format	数据内容的格式，目前支持 CSV 格式，其他格式也会陆续支持。
oss.input.csv.*	定义 CSV 输入格式参数。
oss.output.csv.*	定义 CSV 输出格式参数。
oss.endpoint	bucket 所在的 Endpoint。
oss.accessKeyId	填写 AccessKeyId。
oss.accessKeySecret	填写 AccessKeySecret。

 说明：

目前只定义了基本参数，可以参考OSS Select API 文档，其余参数将陆续支持。

- 支持的过滤条件：`=,<,>,<=, >=,||,or,not,and,in,like(StringStartsWith, StringEndsWith,StringContains)`。对于不能下推的过滤条件，例如：算术运算、字符串拼接等通过 PrunedFilteredScan 获取不到的条件，则只下推需要的列到 OSS Select。

 说明:

OSS Select 还支持其他过滤条件，可以参考OSS Select API 文档。

对比TPC-H的查询

通过测试 TPC-H 中 query1.sql 对于 lineitem 这个 table 的查询性能，来检验配置效果。为了能使 OSS Select 过滤更多的数据，我们将 where 条件由 l_shipdate <= '1998-09-16' 改为 where l_shipdate > '1997-09-16'，测试数据大小为 2.27GB。

- 仅使用 Spark SQL 查询：

```
[root@cdh-master ~]# hadoop fs -ls oss://select-test-sz/data/
lineitem.csv
-rw-rw-rw-   1 2441079322 2018-10-31 11:18 oss://select-test-sz/data
/lineitem.csv
```

- 在 Spark SQL 上使用 OSS Select 查询：

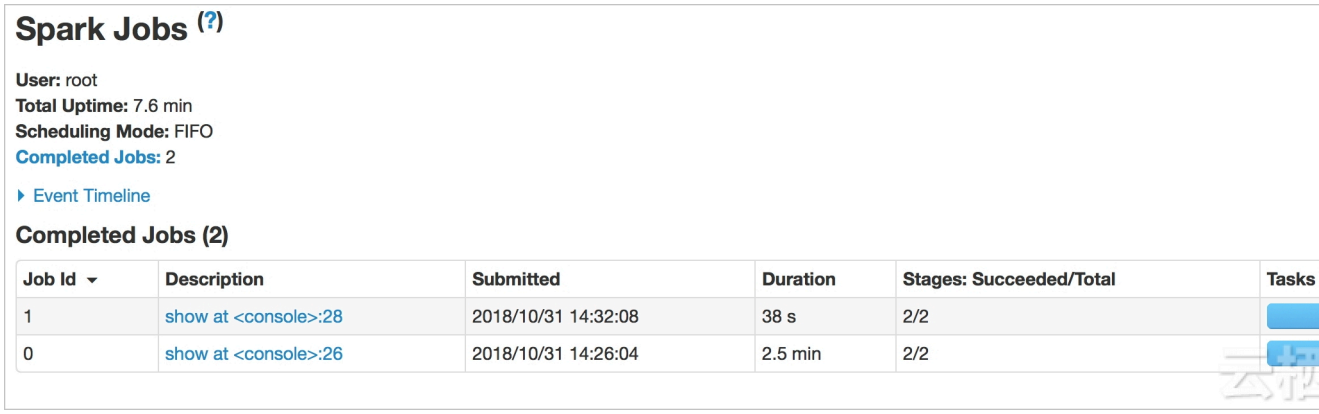
```
scala> import org.apache.spark.sql.types.{IntegerType, LongType,
StringType, StructField, StructType, DoubleType}
import org.apache.spark.sql.types.{IntegerType, LongType, StringType
, StructField, StructType, DoubleType}

scala> import org.apache.spark.sql.{Row, SQLContext}
import org.apache.spark.sql.{Row, SQLContext}

scala> val sqlContext = spark.sqlContext
sqlContext: org.apache.spark.sql.SQLContext = org.apache.spark.sql.
SQLContext@74e2cfc5

scala> val textFile = sc.textFile("oss://select-test-sz/data/
lineitem.csv")
textFile: org.apache.spark.rdd.RDD[String] = oss://select-test-sz/
data/lineitem.csv MapPartitionsRDD[1] at textFile at <console>:26

scala> val dataRdd = textFile.map(_.split('|'))
dataRdd: org.apache.spark.rdd.RDD[Array[String]] = MapPartitionsRDD[
2] at map at <console>:28

scala> val schema = StructType(
     |     List(
     |         StructField("L_ORDERKEY",LongType,true),
     |         StructField("L_PARTKEY",LongType,true),
     |         StructField("L_SUPPKEY",LongType,true),
     |         StructField("L_LINENUMBER",IntegerType,true),
     |         StructField("L_QUANTITY",DoubleType,true),
     |         StructField("L_EXTENDEDPRICE",DoubleType,true),
     |         StructField("L_DISCOUNT",DoubleType,true),
     |         StructField("L_TAX",DoubleType,true),
```

```
     |          StructField("L_RETURNFLAG",StringType,true),
     |          StructField("L_LINESTATUS",StringType,true),
     |          StructField("L_SHIPDATE",StringType,true),
     |          StructField("L_COMMITDATE",StringType,true),
     |          StructField("L_RECEIPTDATE",StringType,true),
     |          StructField("L_SHIPINSTRUCT",StringType,true),
     |          StructField("L_SHIPMODE",StringType,true),
     |          StructField("L_COMMENT",StringType,true)
     |      )
     | )
schema: org.apache.spark.sql.types.StructType = StructType(
StructField(L_ORDERKEY,LongType,true), StructField(L_PARTKEY,
LongType,true), StructField(L_SUPPKEY,LongType,true), StructField
(L_LINENUMBER,IntegerType,true), StructField(L_QUANTITY,DoubleType
,true), StructField(L_EXTENDEDPRICE,DoubleType,true), StructField
(L_DISCOUNT,DoubleType,true), StructField(L_TAX,DoubleType,true),
StructField(L_RETURNFLAG,StringType,true), StructField(L_LINESTAT
US,StringType,true), StructField(L_SHIPDATE,StringType,true),
StructField(L_COMMITDATE,StringType,true), StructField(L_RECEIPTD
ATE,StringType,true), StructField(L_SHIPINSTRUCT,StringType,true
), StructField(L_SHIPMODE,StringType,true), StructField(L_COMMENT,
StringType,true))

scala> val dataRowRdd = dataRdd.map(p => Row(p(0).toLong, p(1).
toLong, p(2).toLong, p(3).toInt, p(4).toDouble, p(5).toDouble, p(6).
toDouble, p(7).toDouble, p(8), p(9), p(10), p(11), p(12), p(13), p(
14), p(15)))
dataRowRdd: org.apache.spark.rdd.RDD[org.apache.spark.sql.Row] =
MapPartitionsRDD[3] at map at <console>:30

scala> val dataFrame = sqlContext.createDataFrame(dataRowRdd, schema
)
dataFrame: org.apache.spark.sql.DataFrame = [L_ORDERKEY: bigint,
L_PARTKEY: bigint ... 14 more fields]

scala> dataFrame.createOrReplaceTempView("lineitem")

scala> spark.sql("select l_returnflag, l_linestatus, sum(l_quantity
) as sum_qty, sum(l_extendedprice) as sum_base_price, sum(l_extended
price * (1 - l_discount)) as sum_disc_price, sum(l_extendedprice *
 (1 - l_discount) * (1 + l_tax)) as sum_charge, avg(l_quantity)
as avg_qty, avg(l_extendedprice) as avg_price, avg(l_discount) as
avg_disc, count(*) as count_order from lineitem where l_shipdate > '
1997-09-16' group by l_returnflag, l_linestatus order by l_returnfl
ag, l_linestatus").show()
+------------+------------+-----------+--------------------
+--------------------+--------------------+------------------
+------------------+--------------------+-----------+
|l_returnflag|l_linestatus|    sum_qty|      sum_base_price|
  sum_disc_price|          sum_charge|           avg_qty|
avg_price|          avg_disc|count_order|
+------------+------------+-----------+--------------------
+--------------------+--------------------+------------------
+------------------+--------------------+-----------+
|           N|           O|7.5697385E7|1.135107538838699...|1.
078345555027154...|1.121504616321447...|25.501957856643052|38241.
036487881756|0.04999335309103123|    2968297|
+------------+------------+-----------+--------------------
+--------------------+--------------------+------------------
+------------------+--------------------+-----------+

scala> sqlContext.sql("CREATE TEMPORARY VIEW item USING com.aliyun.
oss " +
     |   "OPTIONS (" +
```

```
     |   "oss.bucket 'select-test-sz', " +
     |   "oss.prefix 'data', " +
     |   "oss.schema 'L_ORDERKEY long, L_PARTKEY long, L_SUPPKEY
long, L_LINENUMBER int, L_QUANTITY double, L_EXTENDEDPRICE double,
L_DISCOUNT double, L_TAX double, L_RETURNFLAG string, L_LINESTATUS
string, L_SHIPDATE string, L_COMMITDATE string, L_RECEIPTDATE string
, L_SHIPINSTRUCT string, L_SHIPMODE string, L_COMMENT string'," +
     |   "oss.data.format 'csv'," + // we only support csv now
     |   "oss.input.csv.header 'None'," +
     |   "oss.input.csv.recordDelimiter '\n'," +
     |   "oss.input.csv.fieldDelimiter '|'," +
     |   "oss.input.csv.commentChar '#'," +
     |   "oss.input.csv.quoteChar '\"'," +
     |   "oss.output.csv.recordDelimiter '\n'," +
     |   "oss.output.csv.fieldDelimiter ','," +
     |   "oss.output.csv.commentChar '#'," +
     |   "oss.output.csv.quoteChar '\"'," +
     |   "oss.endpoint 'oss-cn-shenzhen.aliyuncs.com', " +
     |   "oss.accessKeyId 'Your Access Key Id', " +
     |   "oss.accessKeySecret 'Your Access Key Secret')")
res2: org.apache.spark.sql.DataFrame = []

scala> sqlContext.sql("select l_returnflag, l_linestatus, sum(
l_quantity) as sum_qty, sum(l_extendedprice) as sum_base_price,
sum(l_extendedprice * (1 - l_discount)) as sum_disc_price, sum(
l_extendedprice * (1 - l_discount) * (1 + l_tax)) as sum_charge,
avg(l_quantity) as avg_qty, avg(l_extendedprice) as avg_price, avg
(l_discount) as avg_disc, count(*) as count_order from item where
l_shipdate > '1997-09-16' group by l_returnflag, l_linestatus order
by l_returnflag, l_linestatus").show()

scala> sqlContext.sql("select l_returnflag, l_linestatus, sum(
l_quantity) as sum_qty, sum(l_extendedprice) as sum_base_price,
sum(l_extendedprice * (1 - l_discount)) as sum_disc_price, sum(
l_extendedprice * (1 - l_discount) * (1 + l_tax)) as sum_charge,
avg(l_quantity) as avg_qty, avg(l_extendedprice) as avg_price, avg
(l_discount) as avg_disc, count(*) as count_order from item where
l_shipdate > '1997-09-16' group by l_returnflag, l_linestatus order
by l_returnflag, l_linestatus").show()
+------------+------------+-----------+--------------------
+--------------------+--------------------+------------------
+------------------+--------------------+-----------+
|l_returnflag|l_linestatus|    sum_qty|      sum_base_price|
  sum_disc_price|          sum_charge|           avg_qty|
avg_price|           avg_disc|count_order|
+------------+------------+-----------+--------------------
+--------------------+--------------------+------------------
+------------------+--------------------+-----------+
|           N|           O|7.5697385E7|1.135107538838701E11|1.
078345555027154...|1.121504616321447...|25.501957856643052|38241.
03648788181|0.04999335309103024|    2968297|
```

```
+------------+-----------+----------+-------------------
+--------------------+--------------------+-----------------
+------------------+--------------------+-----------+
```

从下图可以看出：在 Spark SQL 上使用 OSS Select 查询数据耗时为 38s，在 Spark SQL 上不使用 OSS Select 查询数据耗时为2.5min，使用 OSS Select 可大幅度加快查询速度。



参考文档

更多内容，可参考以下文档：

- 使用 SQL 选取 OSS 文件
- Apache Impala(CDH6) 处理 OSS 数据
- CDH5 Hadoop 如何读写 OSS
- Spark Data Source API

6 数据备份和容灾

6.1 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。

6.2 数据库备份到OSS

本文介绍如何通过数据库备份DBS将本地IDC、公网、第三方云数据库、阿里云RDS和阿里云ECS自建数据库实时备份到OSS上。

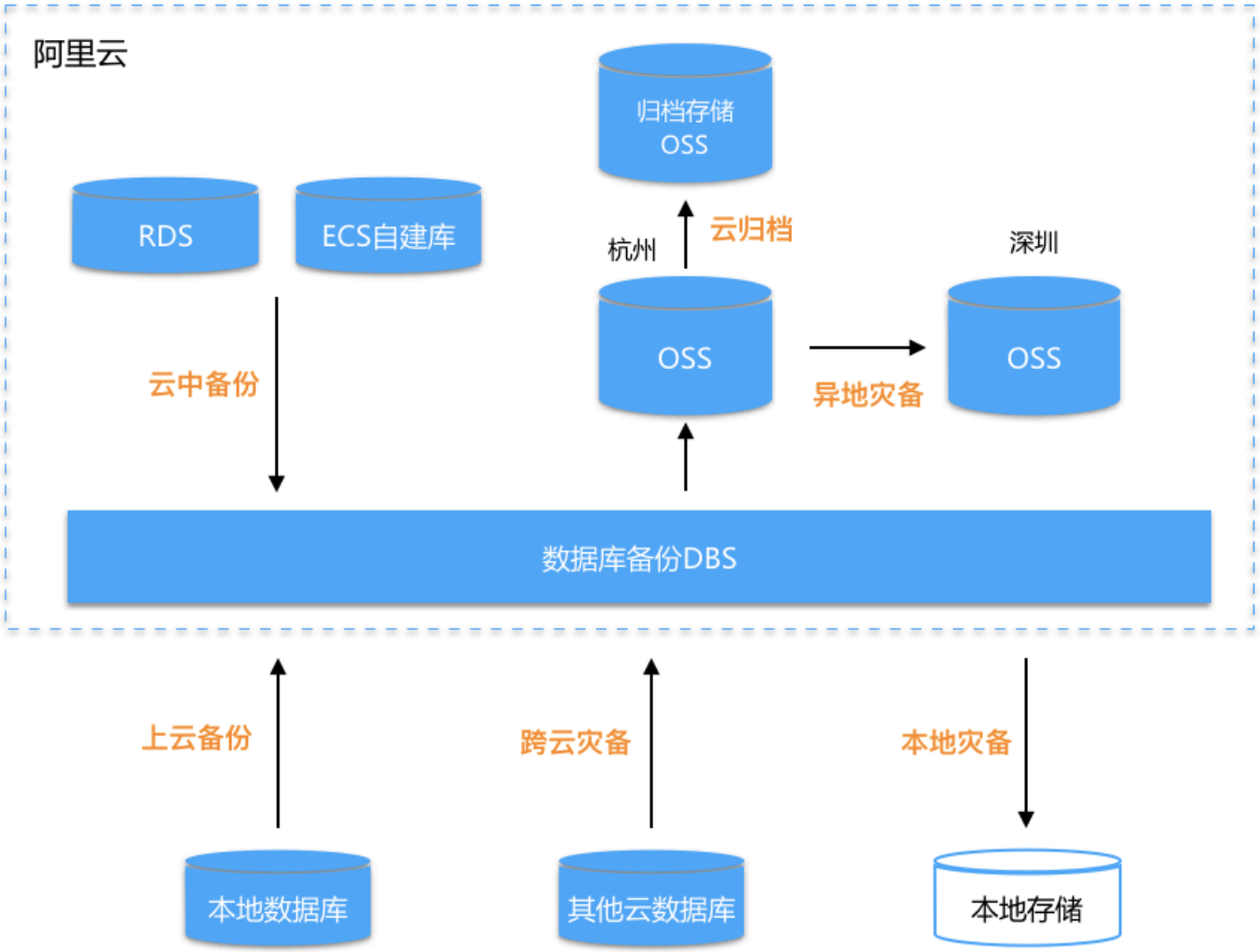
背景

- 对象存储OSS

  对象存储OSS提供了标准类型存储，作为移动应用、大型网站、图片分享或热点音视频的主要存储方式，也提供了成本更低、存储期限更长的低频访问类型存储和归档类型存储，作为不经常访问数据的备份和归档。对象存储OSS非常适合作为数据库备份的存储介质。
- 数据库备份DBS

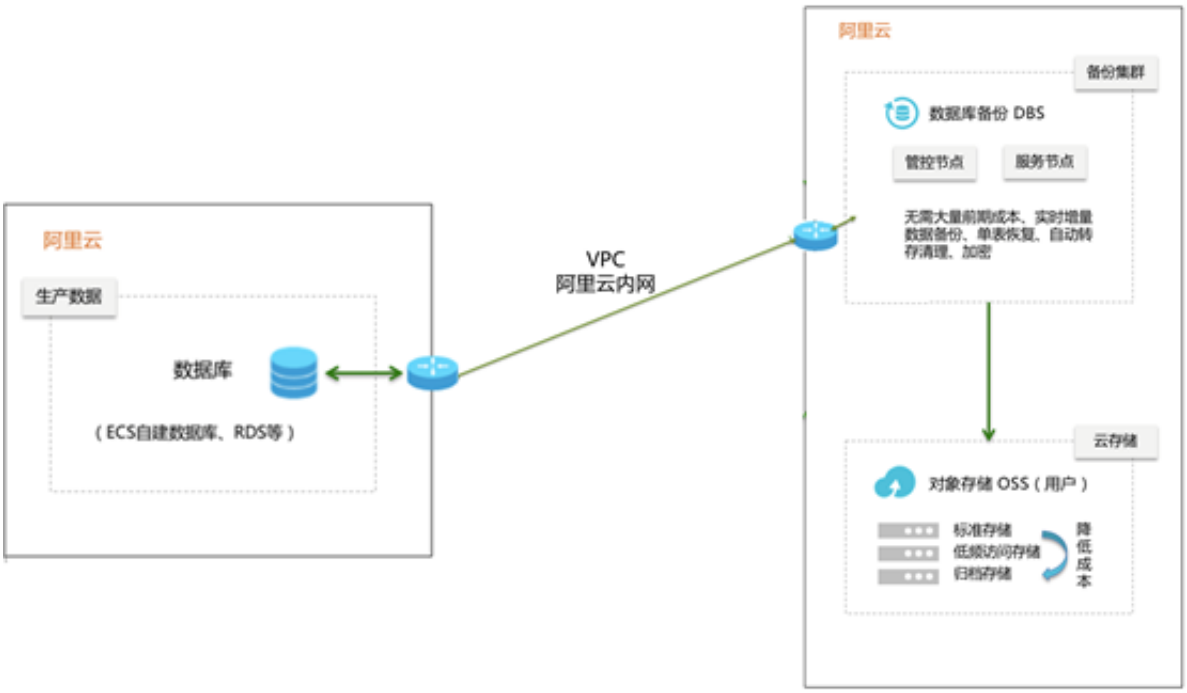
  数据库备份DBS是为数据库提供连续数据保护、低成本的备份服务。它可以为多种环境的数据提供强有力的保护，包括企业数据中心、其他云厂商及公共云。数据库备份提供数据备份和操作恢复的整体方案，具备实时增量备份、精确到秒级的数据恢复能力。

应用场景

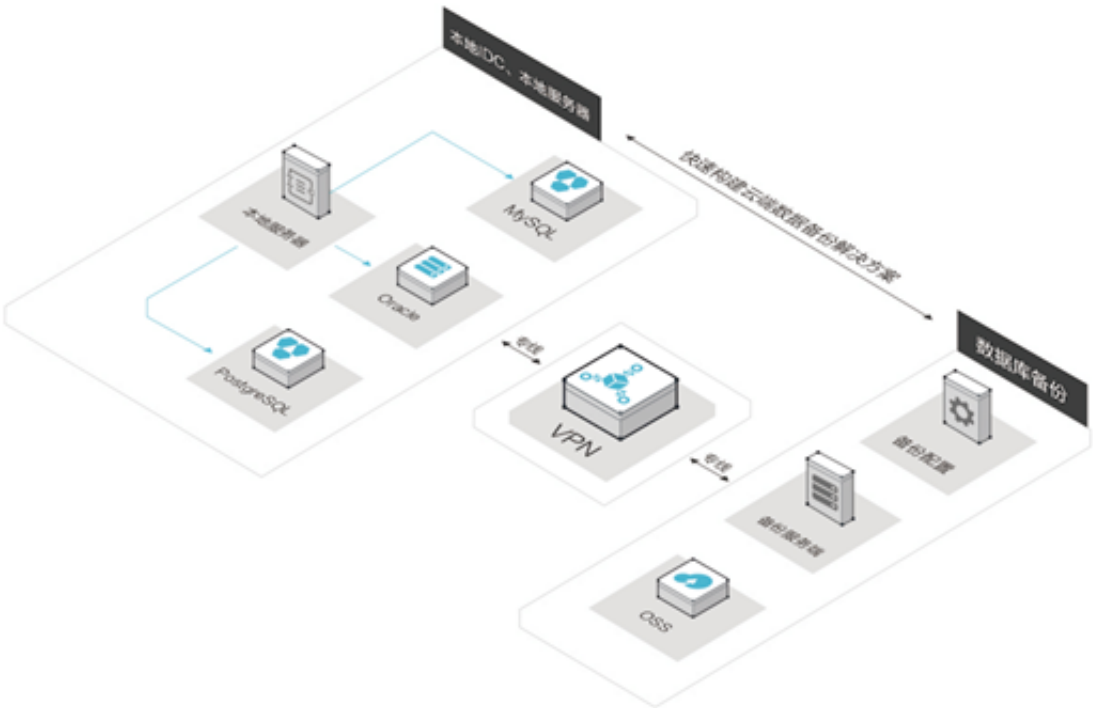


数据库备份到OSS的方案应用场景如下：

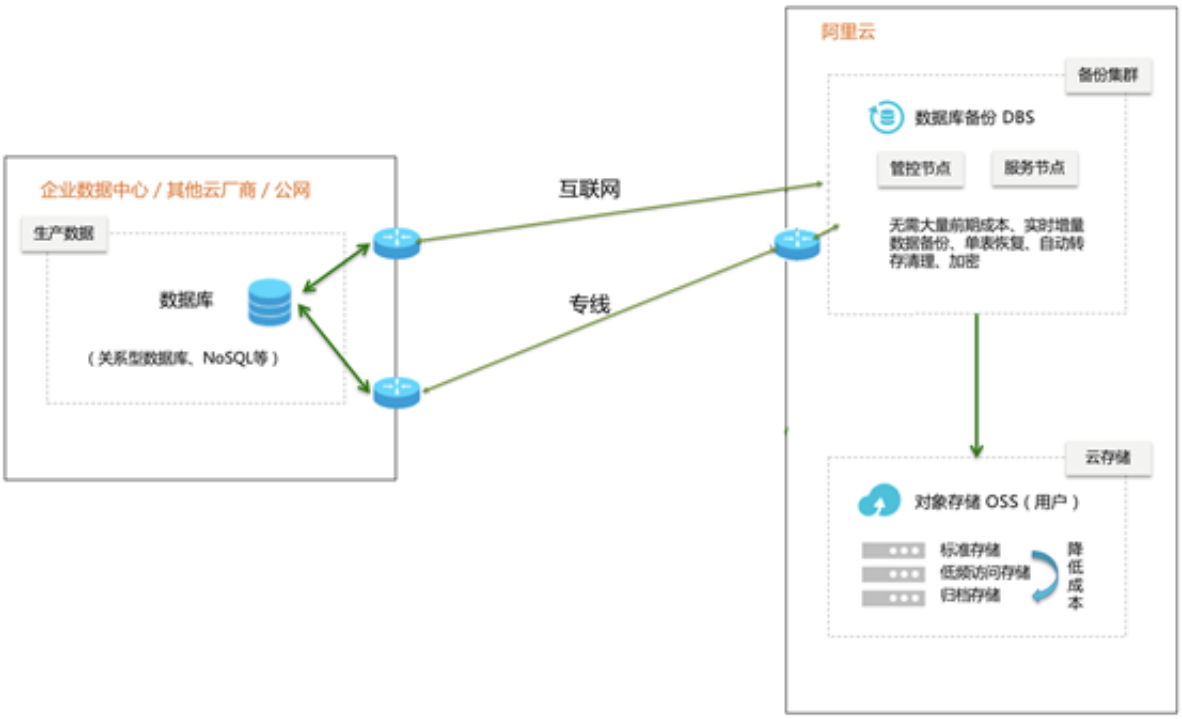
- 阿里云RDS或阿里云ECS自建数据库在OSS上备份或长期归档



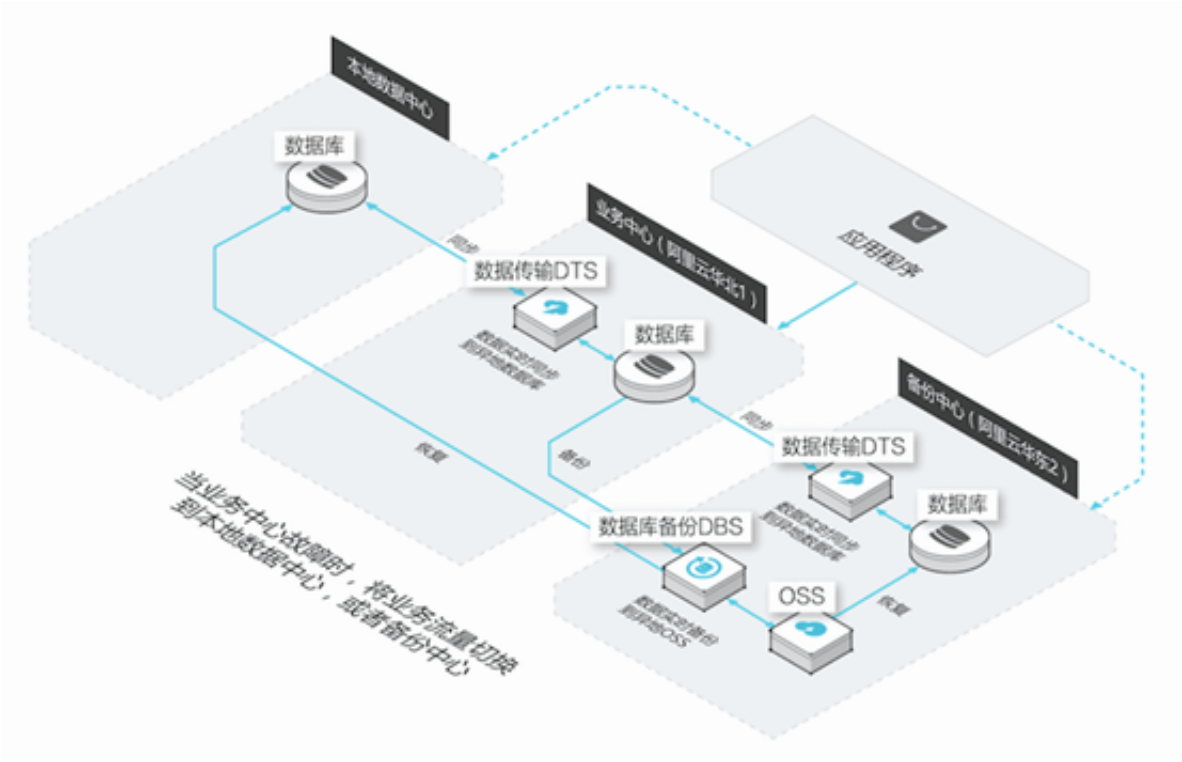
- 自建IDC数据备份上云



- 其他云数据库在阿里云OSS上做多云备份

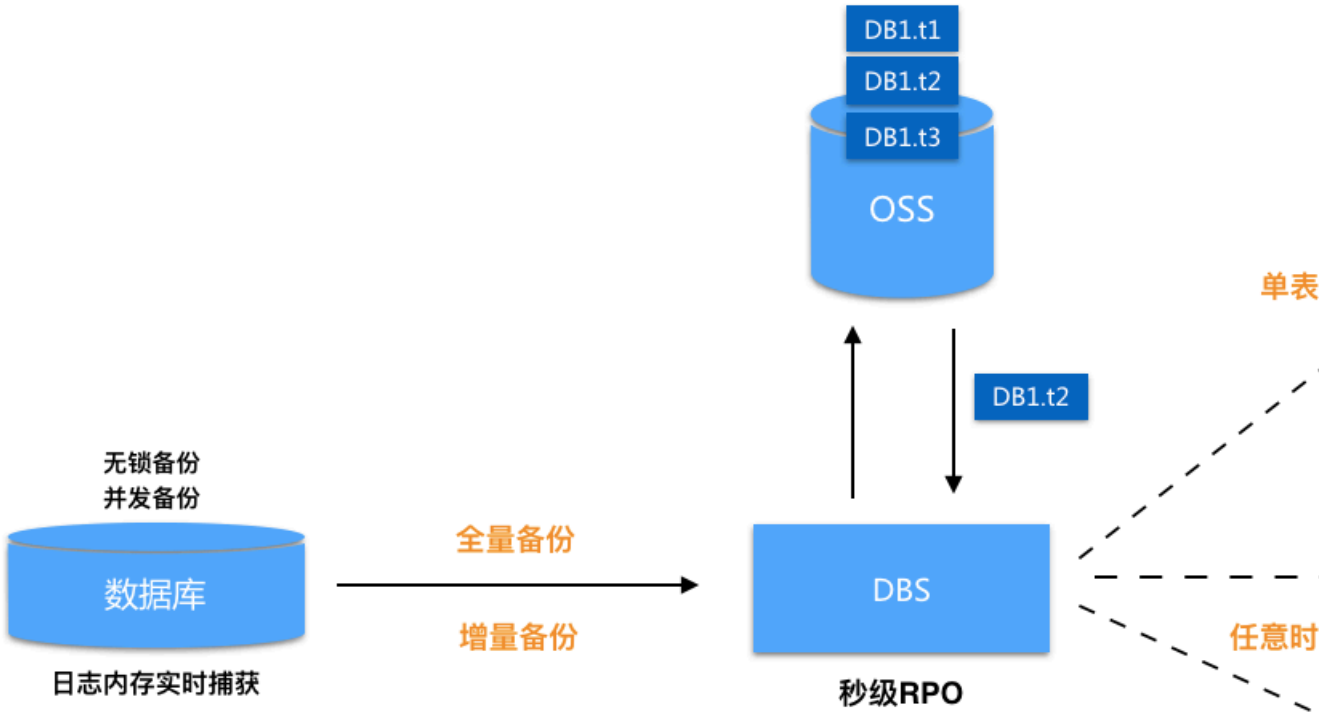


- 数据库做异地备份灾备



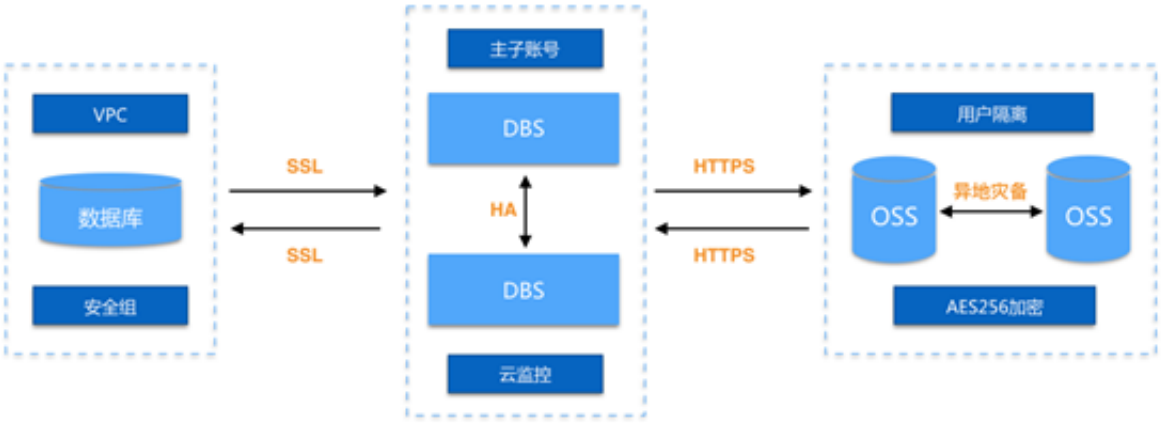
方案优势

- 支持全量或实时增量备份



- 秒级RPO：日志内存实时捕获，CDP实时备份，RPO达到秒级。
- 无锁并发：全程无锁备份、并发备份、数据拉取自适应分片。
- 精准恢复：恢复对象精准匹配，单表恢复，从而大幅降低RTO。
- 灵活恢复：提供可恢复日历及时间轴选择，实现任意时间点恢复。

· 数据强安全高可靠



- 异地灾备：利用OSS的跨区域复制功能，做备份数据的异地灾备，提升数据保护级别。
- 数据传输加密：在传输过程中进行SSL加密，保障数据安全性。
- 数据存储加密：对备份到OSS的数据进行加密存储，保障数据隐私性。
- 随时验证：随时验证数据库备份的有效性。

· 低成本

- 按需付费：OSS存储空间按需付费，避免一次性投入大量资产。
- 自动存储分级：OSS提供标准、低频、归档多种类型，全面优化存储成本。
- 弹性扩展：OSS存储容量弹性扩展，无缝支撑企业在不同发展阶段的性能要求。

- 支持多环境多数据源

  - 支持MySQL、Oracle、SQL Server、MongoDB多种数据库。
  - 支持IDC、第三方云数据库、阿里云RDS、阿里云ECS自建数据库。

方案实施流程

数据库备份到OSS的方案实施流程如下：

1. 创建备份计划。详情请参见数据库备份快速入门中的创建备份计划。
2. 配置备份计划。详情请参见数据库备份快速入门中的配置备份计划。
3. 查看备份计划。详情请参见数据库备份快速入门中的查看备份计划。
4. 恢复数据库。详情请参见数据库备份快速入门中的恢复数据库。

7 通过云存储网关使用OSS服务

7.1 应用场景

本文介绍OSS与云存储网关结合的几种应用场景。

对象存储OSS是海量分布式对象存储服务，提供标准、低频、归档存储类型，覆盖从热到冷的不同存储场景。OSS提供RESTful API，您可以从任何位置访问OSS，存储容量和处理能力弹性扩展。

阿里云云存储网关是一款帮助客户在现有本地应用程序、基础设施和数据存储与阿里云的存储服务之间实现无缝集成的数据服务。通过可在本地和云上部署的兼容行业标准存储协议的虚拟设备，云存储网关将现有的存储应用程序和工作负载无缝对接阿里云的存储和计算服务。有关云存储网关更多详情，请参见云存储网关产品详情页。

云存储网关与OSS结合，主要有以下几种应用场景：

云端扩容

云存储网关以阿里云OSS为后端存储。OSS是阿里云提供的海量、安全、低成本、高可靠、高可用的云存储服务。相比阿里云的EBS和NAS服务，OSS可以提供更高的容量以及更低的存储成本，这对于有存放海量数据需求的客户来说是一个很好的选择。OSS默认支持RESTful的对象接口，或者通过Hadoop来访问OSS。

您还可以使用云存储网关通过NFS、SMB、iSCSI来访问OSS，使得传统应用程序可以像使用文件夹或者块设备一样使用OSS。您无需改造现有应用，只需开通和简单配置云存储网关即可。主要有以下几种应用场景：

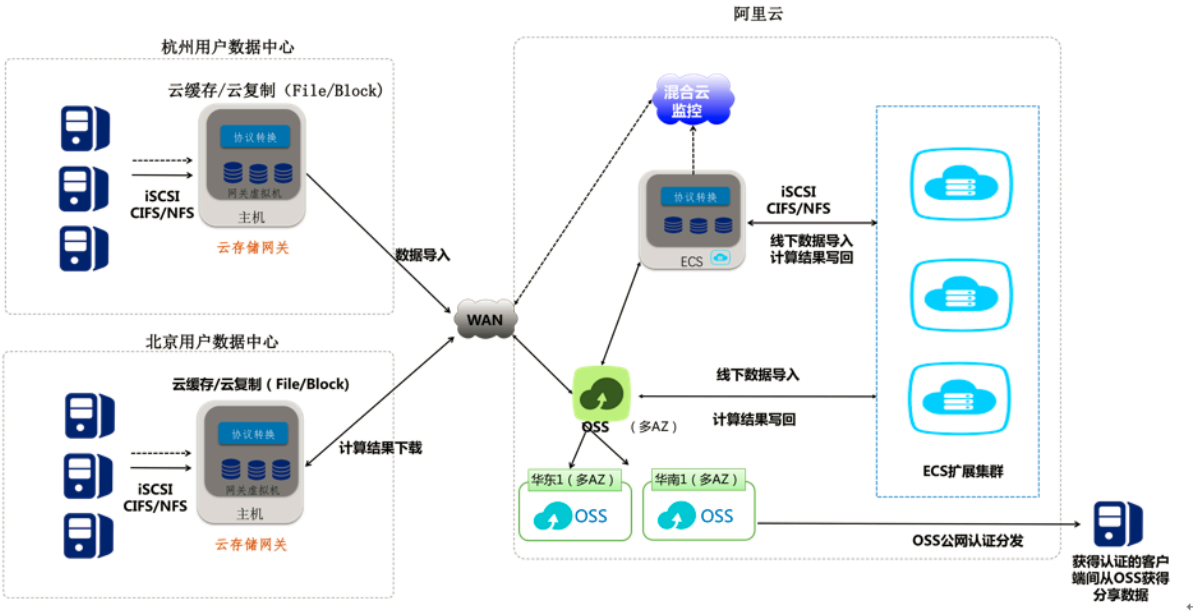
- 共享文件池：在不同计算集群之间共享文件和数据。
- 数据备份：通过类似Veaam、NBU等备份软件，将应用数据按一定的策略通过云存储网关将数据备份到阿里云的OSS存储服务。
- 冷数据归档：通过云存储网关将冷数据从线下或者ECS实例中写入OSS的低频和归档存储类型，从而释放本地空间，并减少存储成本。

跨地域共享和数据分发

云存储网关对接的OSS存储服务可以通过网络实现线下和线上多地访问。通过云存储网关，您可以轻松实现一处写入多地读取的数据共享服务。

在下图的示例场景中，杭州的用户数据中心产生大量数据，但因本地计算能力不足，您可以在阿里云上弹性扩展ECS集群。您在阿里云ECS集群里的应用既可以直接读取网关上传到OSS的对象数据，也可以部署和线下一样的应用程序，通过云存储网关读取OSS共享的数据。

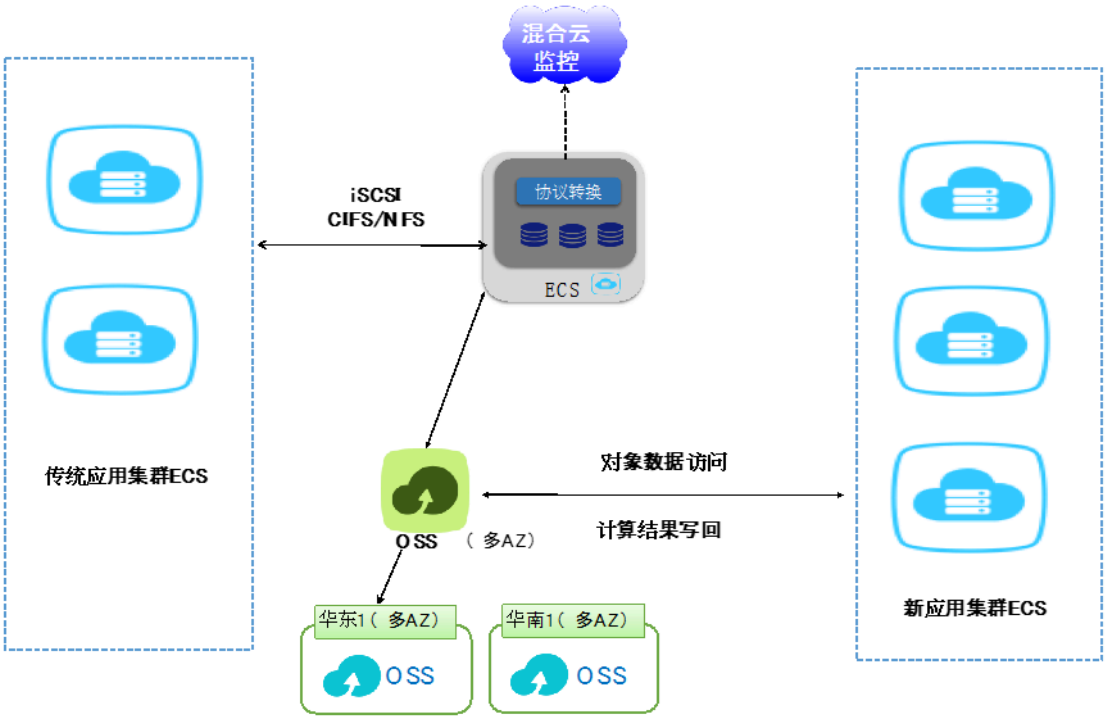
计算任务完成后，最终的结果写回到OSS。如果需要将数据发布到北京，您可以通过存储网关把数据反向同步到北京的数据中心，也可以直接通过OSS的外链功能把数据开放给认证的客户端下载。



适配传统应用

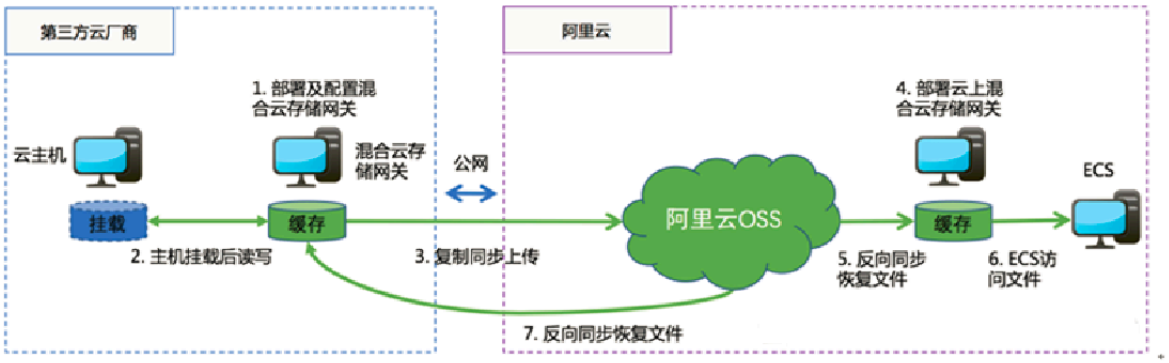
很多云上业务都是新旧业务的组合。其中，从数据中心迁移过来的旧业务使用的是标准的存储协议，如NFS、CIFS和iSCSI；新业务一般采用比较新的技术，支持对象访问的协议。要实现新旧业务之间的数据通讯需要大量的改造工作。而云存储网关恰好可以起到桥梁作用，便捷地进行新旧业务的数据交换。

在下图的示例场景中，传统应用集群可能是文件处理、邮件服务器、图片处理等传统的应用，它们只支持文件或者块访问协议。此时云存储网关的在线版本可以兼容不同的数据访问协议，让这些传统应用集群可以直接访问OSS中的数据。传统应用集群产生的中间数据存储在OSS后，新应用集群可以直接从OSS中读取。反之，新应用集群收集了数据并存放在OSS中，传统应用集群通过云存储网关也可以无缝访问和处理OSS中的数据。



云容灾

随着云计算的普及，越来越多的用户选择将业务部署到云上。为确保数据安全性和业务连续性，跨云容灾提供了高效可靠的本地备份恢复和云上故障转移。借助云存储网关对虚拟化的全面支持，可以轻松应对各种第三方云厂商对接阿里云的数据容灾。整体架构如下图所示。



通过在第三方云厂商导入阿里云的云储存网关（OVA、VHD格式），并配置公网IP，即可搭建阿里云OSS的数据通路。完成数据复制后，部署在阿里云上的ECS即可通过云上部署的阿里云存储网关获取容灾数据，并恢复服务。

注意事项

#unique_118

7.2 使用指南

7.2.1 简介

OSS和云存储网关集成，让您可以像使用本地文件夹和磁盘一样使用OSS存储服务。

阿里云的云主机主要运行的操作系统分两大类，Linux和Windows。针对不同的操作系统，云存储网关提供了两种文件访问协议：NFS和SMB（CIFS），从而实现本地共享文件夹访问。详情请参见#unique_121。

此外，云存储网关还提供了iSCSI协议，将海量的OSS存储空间映射为本地磁盘，并提供高性价比的存储扩容方案以及超过EBS单盘32TB的容量，适用于对接冷数据和归档数据。详情请参见#unique_122。

通过云存储网关提供的上述三种协议，OSS存储资源会以Bucket为基础映射成本地文件夹或者磁盘。您可以通过文件读写操作访问OSS资源，无缝衔接基于POSIX和块访问协议的应用，降低应用改造和学习成本。

7.2.2 本地共享文件夹访问

针对Linux和Windows两种不同的操作系统，云存储网关提供了两大类文件访问协议：NFS和SMB（CIFS）。本文主要介绍两种不同协议的共享文件夹访问方式。

Linux下的NFS协议共享文件夹访问

NFS协议是Linux/Unix系统下的主流共享文件访问协议。通过简单的挂载，NFS协议共享的OSS存储可以像一个本地文件夹一样进行读写和访问。

访问存储网关的客户端必须安装NFS，不同的linux操作系统安装方法也不一样。这里介绍ubuntu操作系统和centos操作系统的安装命令，其它操作系统上安装NFS，请查阅官方文档。

- 在ubuntu操作系统上执行以下安装命令：

```
apt-get install nfs-utils
```

- 在centos操作系统上执行以下安装命令：

```
yum install -y nfs-utils
```

命令安装完后，请执行以下步骤：

1. 创建NFS。

   a. 在云存储网关/NFS页面，单击右上角的创建按钮，打开创建NFS对话框，如下图所示。

   说明:
   仅需填写带*的必选项即可创建NFS。

   b. 在共享名称框中，填写NFS的虚拟路径（NFS v4）。

   c. 在用户映射框中，下拉选择None、root_squash、all_squash、all_anomnymous四种方式中的其中一种。

   d. 在模式框中，选择相应的模式。

      · 同步模式：所有数据都会保存两份拷贝，一份保存在本地缓存，另一份保存在OSS。
      · 缓存模式：本地缓存全量元数据和经常访问的用户数据。OSS保存全量数据。

   e. 在云资源框中，单击选择，选择相应的云资源后单击确认。

   f. 在缓存硬盘框中，单击选择，选择缓存硬盘后单击确认。

   其他更多参数详情，请参见#unique_124。

2. 客户端挂载

在客户端打开命令行终端输入以下挂载命令：

```
mount.nfs x.x.x.x:/shares local-directory
```

其中，

- x.x.x.x:/shares：指的是您的存储网关上的IP地址和共享目录。
- local-directory：指的是客户端的本地目录，可以是任意有读写权限的目录，不能指定不存在的文件目录。

挂载成功后使用命令df -h查看系统的磁盘信息。

```
[root@centos7cb ~]# df -h
Filesystem          Size  Used Avail Use% Mounted on
/dev/vda1            99G  1.6G   92G   2% /
devtmpfs             24G     0   24G   0% /dev
tmpfs                24G     0   24G   0% /dev/shm
tmpfs                24G  424K   24G   1% /run
tmpfs                24G     0   24G   0% /sys/fs/cgroup
tmpfs               4.8G     0  4.8G   0% /run/user/0
192.168.0.68:/nfs2  256T     0  256T   0% /mnt/nfs172cent7.4
[root@centos7cb ~]#
```

说明：

NFS客户端挂载后容量是云端OSS的容量大小。256TB是文件系统的最大容量，目前OSS的存储空间无大小限制。

挂载后的文件夹就是一个云盘，所以读写的数据都会存放在OSS。NFS共享可以被多个客户端主机挂载以实现NAS服务。更多详情，请参见#unique_118。

Windows下的SMB（CIFS）协议共享文件夹访问

SMB（CIFS）协议是Windows系统下的主流共享文件访问协议。通过网络文件夹映射，把OSS存储映射成Windows下的一个网络驱动器，提供类似硬盘的访问。具体操作步骤如下：

1. 创建SMB（CIFS）共享

a. 在云存储网关/CIFS页面，单击右上角的创建按钮，打开创建CIFS对话框，如下图所示。

b. 在共享名称框中，填写CIFS的网络共享名称。

c. 在模式框中，选择相应的模式。

- 同步模式：所有数据都会保存两份拷贝，一份保存在本地缓存，另一份保存在OSS。
- 缓存模式：本地缓存全量元数据和经常访问的用户数据。OSS保存全量数据。

d. 反向同步：将OSS上的元数据同步到本地，适用于网关容灾和数据恢复、共享场景。

e. 在云资源框中，单击选择，选择相应的云资源后单击确认。

f. 在缓存硬盘框中，单击选择，选择缓存硬盘后单击确认。

2. 添加SMB （CIFS）用户

a. 在云存储网关/CIFS页面，选择CIFS用户一栏。

b. 单击右上角的创建按钮，打开添加CIFS用户对话框，如下图所示。

c. 填写名称、密码后，单击确认。

> 说明:
> 添加CIFS用户时，不支持使用单个英文字母作为用户名称。

3. 访问共享目录

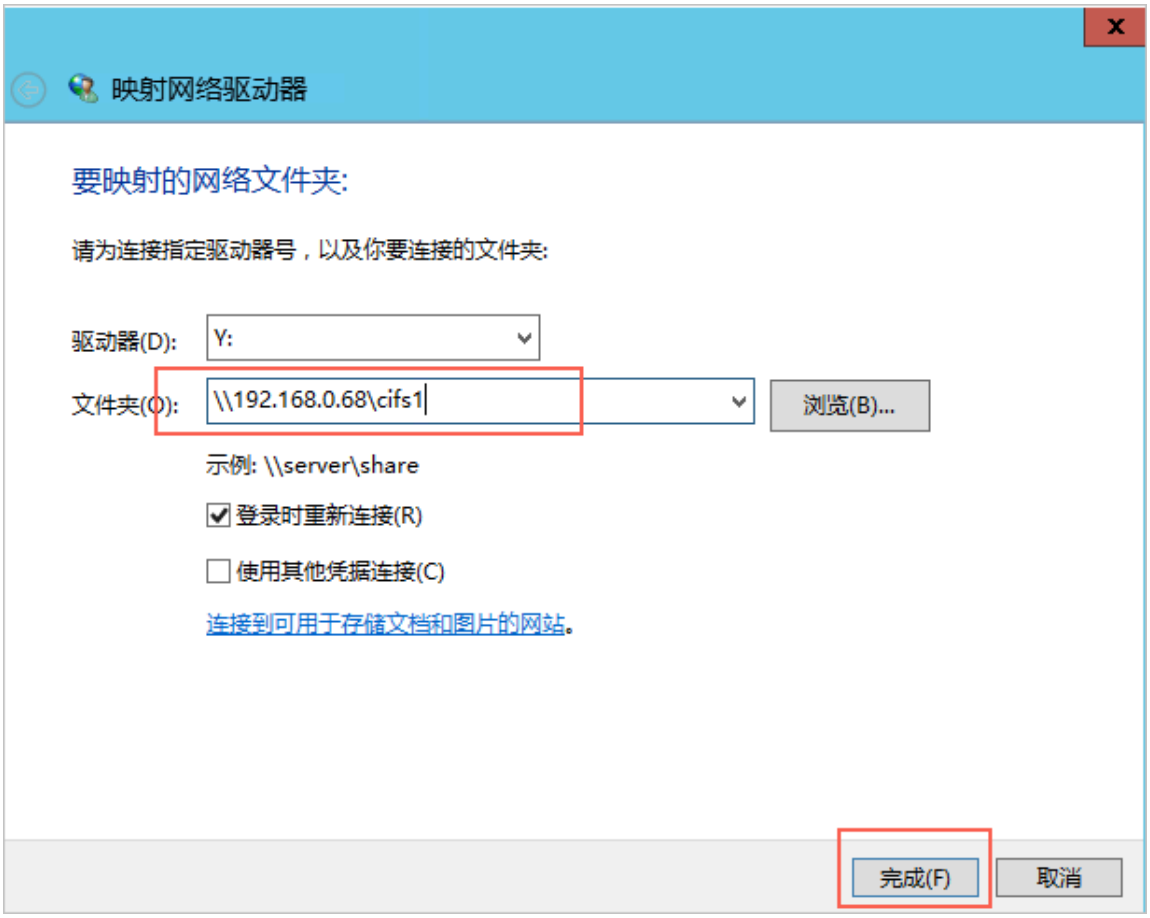
Windows操作系统访问存储网关，首先需要将存储网关的共享目录添加到本地的映射网络驱动器，添加成功后会在本地的目录和存储网关上的共享目录间建立映射。您通过操作本地的磁盘目录实现对远端OSS存储的操作。

> 说明:
> · CIFS共享最大允许个数为16个。

- CIFS客户端挂载后容量是云端OSS的容量大小。256TB是文件系统的最大容量，目前OSS的存储空间无大小限制。

具体操作步骤如下：

a. 打开计算机，单击映射网络驱动器，填写文件夹一栏，如下图所示。

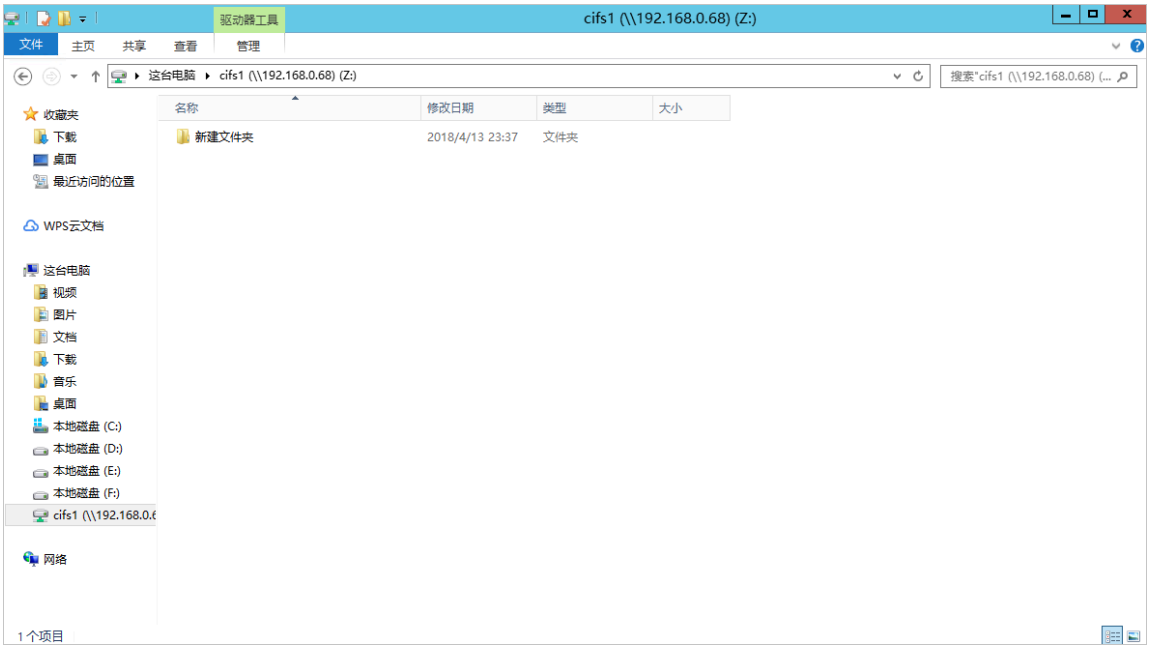


说明:

文件夹一栏填写格式为：\\文件网关与客户端互通的ip\共享目录名。

b. 单击完成，然后输入CIFS用户名密码。

c. Windows客户端添加CIFS共享完成后，在客户端访问该共享网络驱动器情况如下。



7.2.3 本地磁盘访问

云存储网关提供了iSCSI协议，将海量的OSS存储空间映射为本地磁盘。通过集成云存储网关的在线服务，可以像使用本地磁盘一样使用OSS存储服务。

配置iSCSI Target

通过云存储网关导航列iSCSI卷选项进入块云网关/iSCSI卷界面，您可以在这个界面上创建卷、启用/禁用iSCSI功能、设置CHAP授权、删除逻辑卷等。

创建卷的具体步骤如下：

1. 单击创建，进行卷的创建。

说明:

- 卷最大创建个数：128个。
- iSCSI目标最大允许个数：128个。

2. 填写卷名称、是否启用恢复、容量、选择云资源、是否启用iSCSI、选择模式、缓存磁盘等信息。

说明:

- 在恢复选项框中：
  - 选择否（默认）：系统会直接使用云资源对应的OSS Bucket创建新的卷。
  - 选择是：如果云资源对应的OSS Bucket已经被用作卷的云存储，系统会尝试使用其中的元数据（例如卷的容量等）进行卷的恢复。

- 在模式选择框中：
  - 缓存模式：在缓存模式下，热数据会缓存在本地，读写优先访问本地的缓存盘。通常iSCSI网关的读写性能在缓存模式下更优。
  - 写透模式：在写透模式下，客户文件会透传到云上的OSS Bucket，读取时从云端直接读取，适用于冷数据备份归档场景。

3. 信息填写完成后，单击确认。

卷创建成功后会在导航页显示创建的列表：

在块云网关/iSCSI卷的卷列表中，单击左侧>按钮查看卷属性。

Linux客户端上使用卷

1. 安装软件

   请执行以下步骤通过Linux客户端连接到卷：

   a. 使用iscsi-initiator-utils RPM 包连接到网关iSCSI目标。

   说明:

使用sudo yum install iscsi-initiator-utils 命令安装该包，如果您已完成安装，请跳过此步骤。

b. 使用如下命令验证iSCSI守护进程是否正在运行。

- sudo /etc/init.d/iscsi status //：适用于RHEL 5或 RHEL 6。
- sudo service iscsid status //：适用于RHEL7。

如果使用上述命令检查未返回running状态，请使用如下命令运行程序：

```
sudo service iscsid start //
```

2. 挂载卷

a. 发现卷，访问端口是3260。

格式：

sudo iscsiadm -m discovery -t st –p < GATEWAY_IP >:3260

示例：

iscsiadm -m discovery -t st -p 10.0.100.51:3260

说明：

其中，10.0.100.51是网关IP，可通过控制台的关于 > 网络信息 > IP信息获取。

b. 挂载卷

注意：

由于 iSCSI 协议限制，请勿将一个卷挂载到多个客户端上。

请使用如下命令挂载发现的卷。

格式：

iscsiadm --mode node --targetname <TargetName> --portal <GATEWAY_IP> -l

说明：

其中 TargetName 替换为需要挂载的卷的 TargetName，GATEWAY_IP 替换为您的网关 IP。

示例：

iscsiadm -m node -T iqn.2009-09.com.aliyun.iscsi-sgw:test97-block95 -p 10.0.100.51:3260 -l

```
[root@localhost ~]# iscsiadm -m discovery -t st -p 10.0.100.51:3260
10.0.100.51:3260,1 iqn.2009-09.com.aliyun.iscsi-sgw:testVolume95-block95
10.0.100.51:3260,1 iqn.2009-09.com.aliyun.iscsi-sgw:test96-block95
10.0.100.51:3260,1 iqn.2009-09.com.aliyun.iscsi-sgw:test961-block95
10.0.100.51:3260,1 iqn.2009-09.com.aliyun.iscsi-sgw:test97-block95
[root@localhost ~]# iscsiadm -m node  -T iqn.2009-09.com.aliyun.iscsi-sgw:test97-block95 -p 10.0.100.51:3260 -l
Logging in to [iface: default, target: iqn.2009-09.com.aliyun.iscsi-sgw:test97-block95, portal: 10.0.100.51,3260] (multiple)
Login to [iface: default, target: iqn.2009-09.com.aliyun.iscsi-sgw:test97-block95, portal: 10.0.100.51,3260] successful.
[root@localhost ~]#
```

c. 查看卷

您可以使用`fdisk -l`、`lsblk`等命令查看已经挂载的卷。当前状态下，卷已经是一个可用的裸磁盘。

使用`fdisk -l`查看如图所示：

```
[root@client2 ~]# fdisk -l

Disk /dev/sda: 214.7 GB, 214748364800 bytes, 419430400 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 4096 bytes
I/O size (minimum/optimal): 4096 bytes / 4096 bytes
Disk label type: dos
Disk identifier: 0x000e1179

   Device Boot      Start         End      Blocks   Id  System
/dev/sda1   *        2048     2099199     1048576   83  Linux
/dev/sda2         2099200   419430399   208665600   8e  Linux LVM

Disk /dev/mapper/centos-root: 53.7 GB, 53687091200 bytes, 104857600 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 4096 bytes
I/O size (minimum/optimal): 4096 bytes / 4096 bytes


Disk /dev/mapper/centos-swap: 8455 MB, 8455716864 bytes, 16515072 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 4096 bytes
I/O size (minimum/optimal): 4096 bytes / 4096 bytes


Disk /dev/mapper/centos-home: 151.5 GB, 151523426304 bytes, 295944192 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 4096 bytes
I/O size (minimum/optimal): 4096 bytes / 4096 bytes


Disk /dev/sdd: 1099.5 GB, 1099511627776 bytes, 2147483648 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 4096 bytes / 524288 bytes
```

Windows客户端上使用卷

1. 启动iSCSI发起程序

使用 Microsoft Windows 客户端连接到卷，您需要使用 Microsoft Windows iSCSI 启动程序来连接到卷，将卷作为 Windows 客户端上的本地设备。

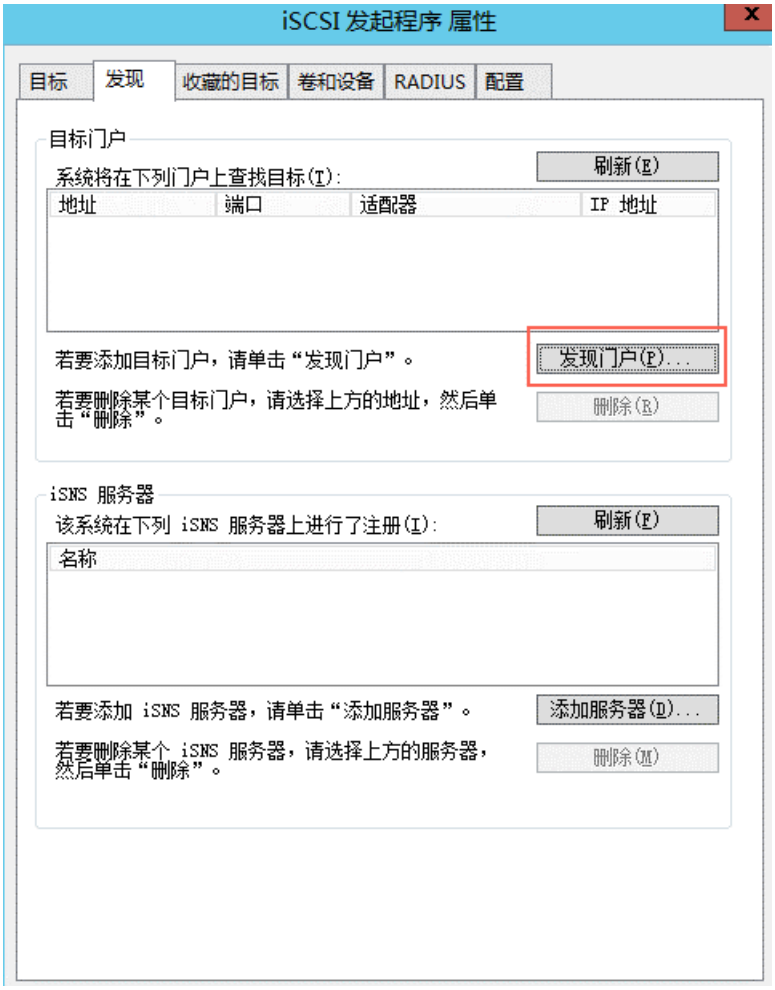
> **注意:**
> 由于 iSCSI 协议限制，不支持将多个主机连接到同一个 iSCSI 目标。

以下用Windows 2012R2为例说明如何启动iSCSI发起程序：

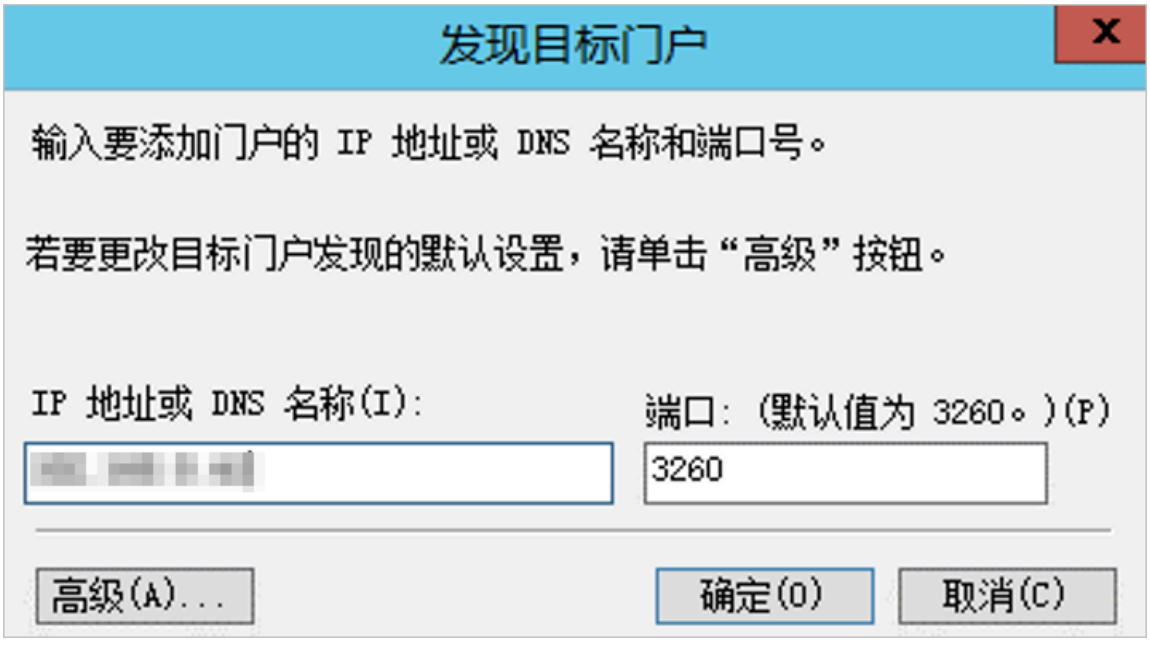
2. 设置iSCSI门户

a. 在弹出的 iSCSI 发起程序 属性对话框中，单击发现。

b. 在发现选项卡中单击发现门户，如下图所示：



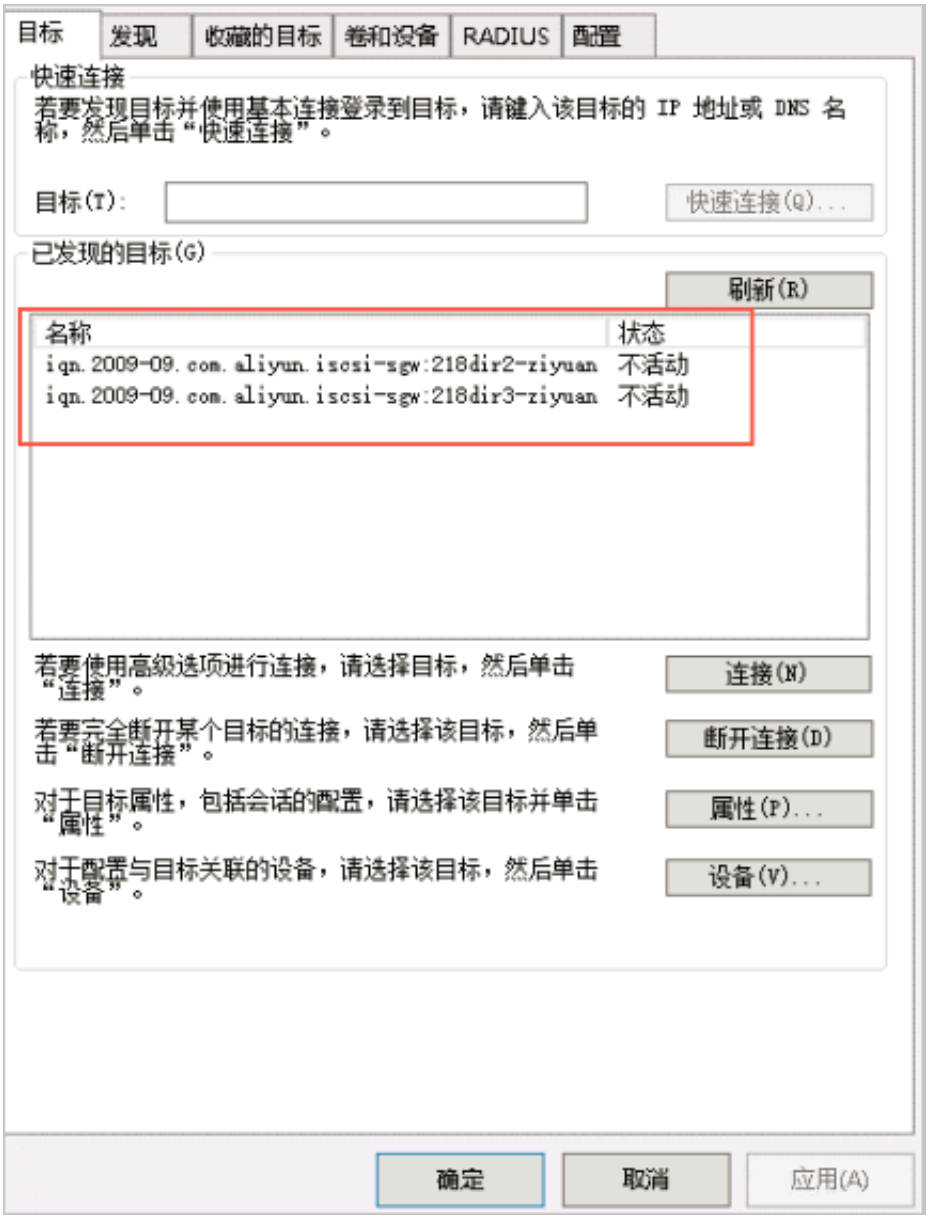
c. 在弹出的窗口中输入目标的 IP 地址，然后单击确认添加该目标门户。



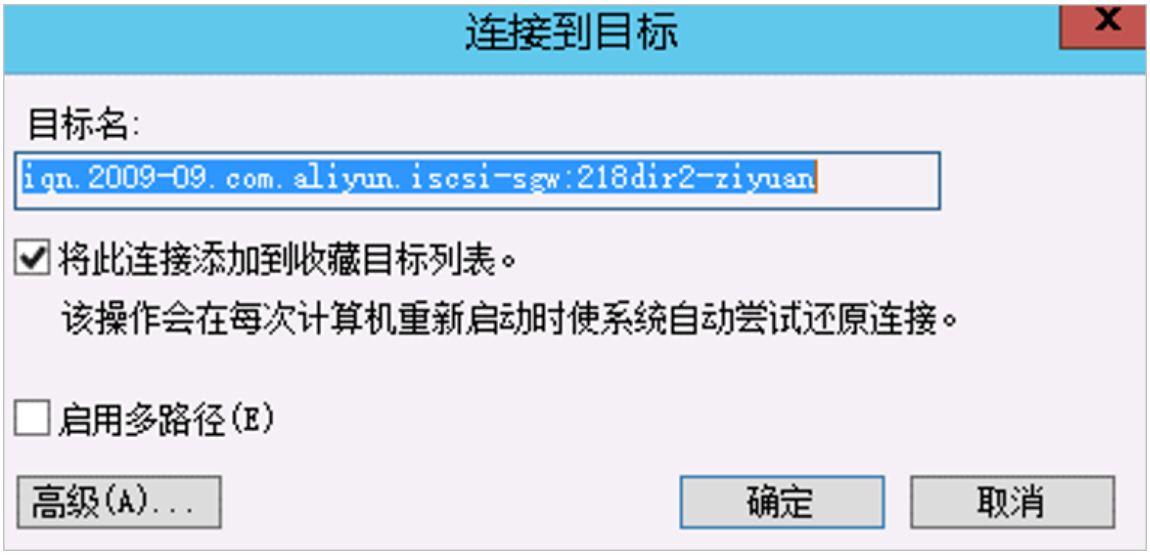
3. iSCSI Target 连接

a. 单击目标。

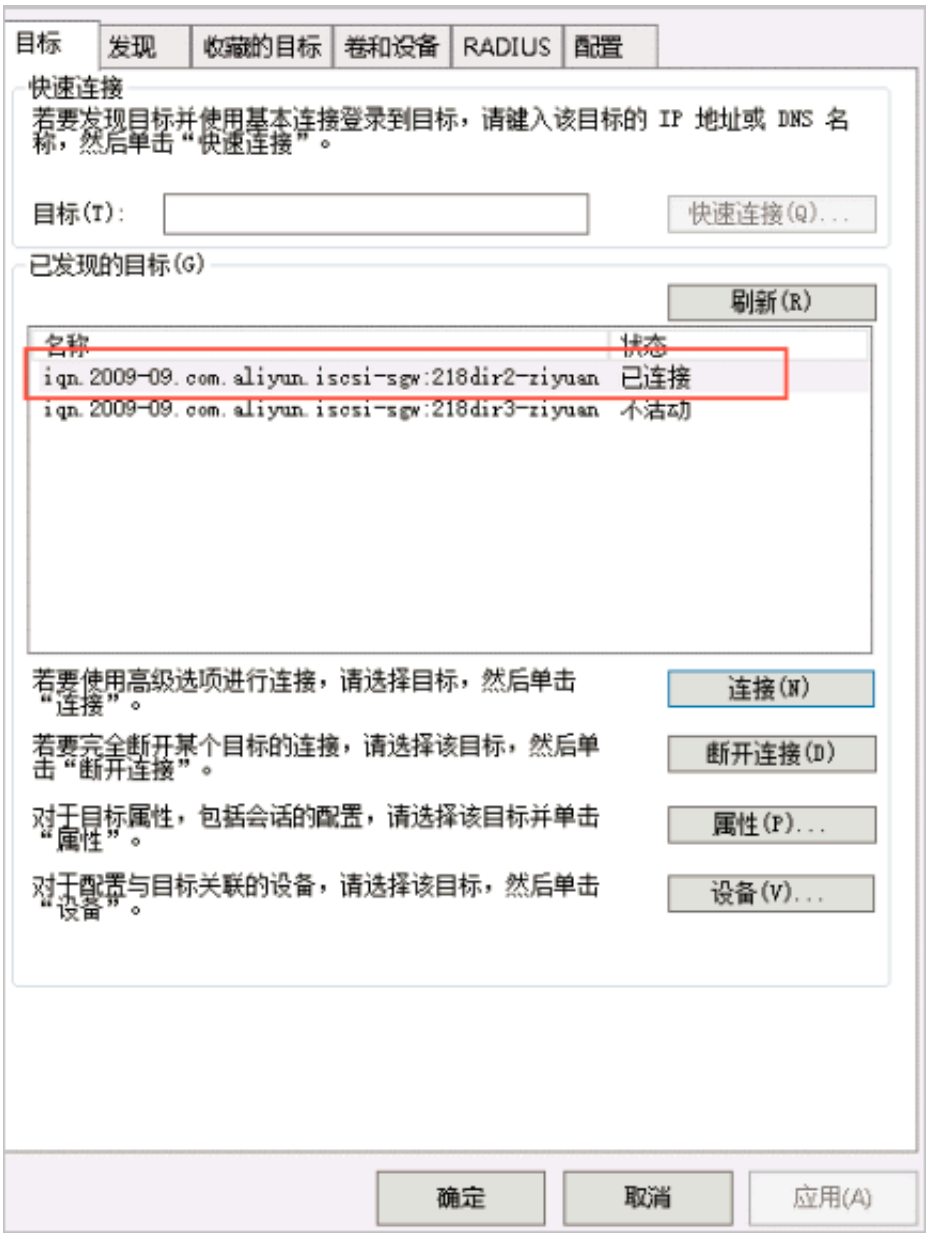
b. 在目标选项卡中，选中上一步骤中处于未激活状态的目标门户，然后单击连接按钮。



c. 在弹出的对话窗中确认目标名，并勾选将此连接添加到收藏目标列表，然后单击确定。

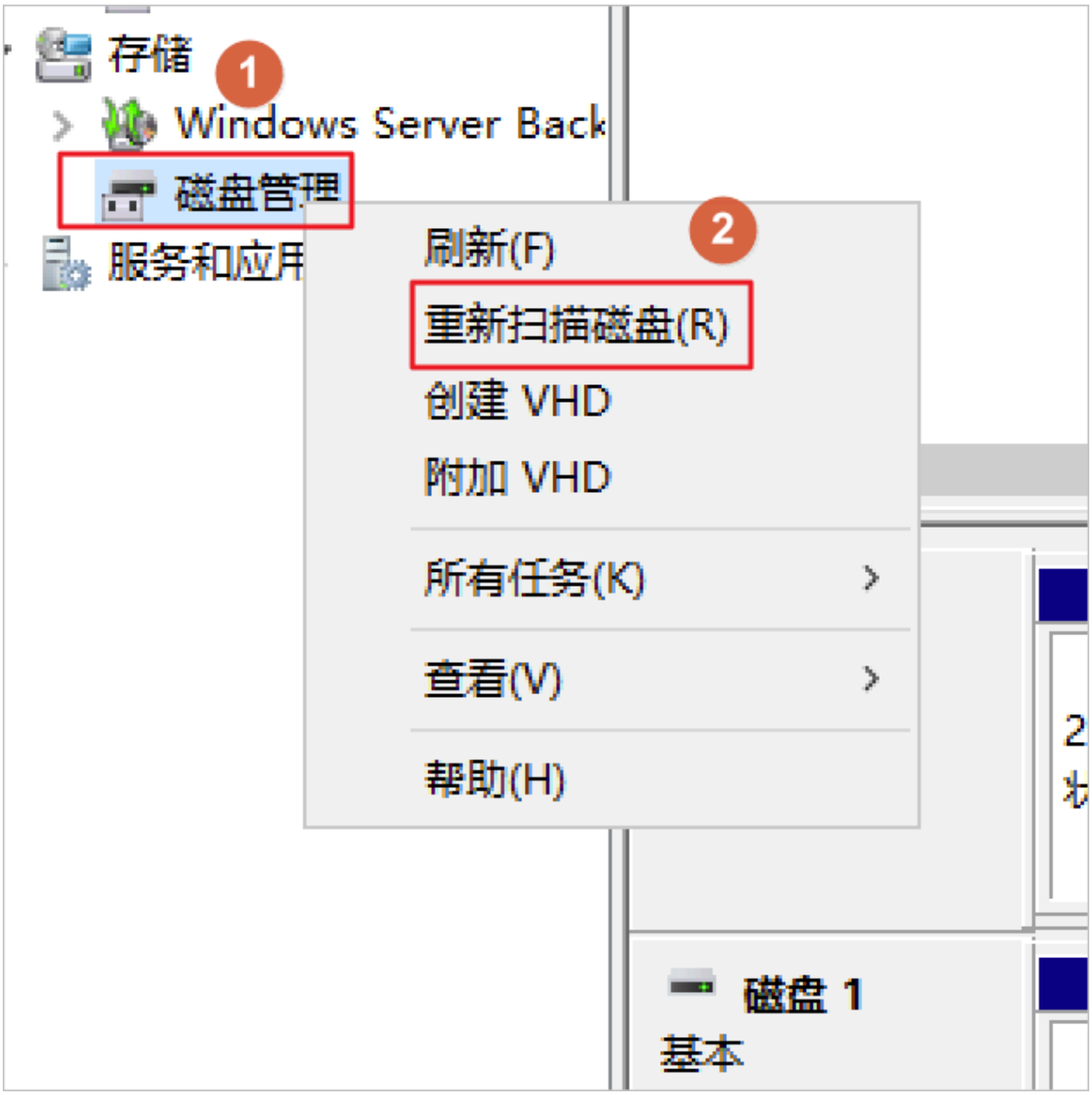


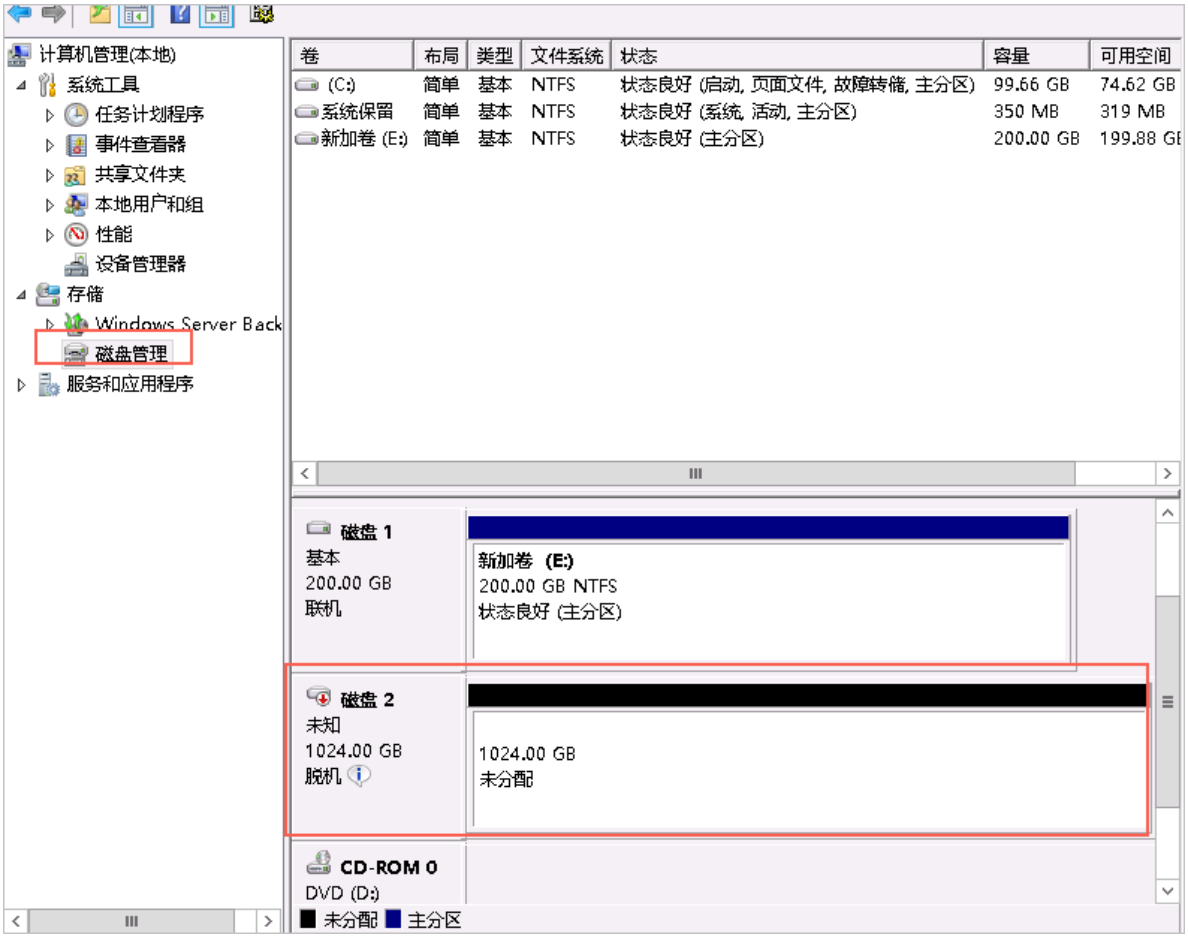
d. 确定状态为已连接后，单击确定按钮。



4. 查看磁盘

右键单击磁盘管理选项，选择重新扫描磁盘，即能发现新添加的磁盘。




计算机管理(本地)
系统工具
任务计划程序
事件查看器
共享文件夹
本地用户和组
性能
设备管理器
存储
Windows Server Back
磁盘管理
服务和应用程序
卷 布局 类型 文件系统 状态 容量 可用空间
(C:) 简单 基本 NTFS 状态良好 (启动, 页面文件, 故障转储, 主分区) 99.66 GB 74.62 GB
系统保留 简单 基本 NTFS 状态良好 (系统, 活动, 主分区) 350 MB 319 MB
新加卷 (E:) 简单 基本 NTFS 状态良好 (主分区) 200.00 GB 199.88 GB
磁盘 1
基本
200.00 GB
联机
新加卷 (E:)
200.00 GB NTFS
状态良好 (主分区)
磁盘 2
未知
1024.00 GB
脱机
1024.00 GB
未分配
CD-ROM 0
DVD (D:)
未分配 主分区

8 存储空间管理

8.1 跨域资源共享（CORS）

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，OSS 支持 CORS 标准来实现跨域访问。本文通过两个示例介绍如何设置 CORS。

说明:

具体的 CORS 规则可以参考W3C CORS规范。

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

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。

使用场景

CORS一定是在使用浏览器的情况下使用，因为控制访问权限的是浏览器而非服务器。因此使用其它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。

1. 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。该文件的访问地址为`http://oss-cors-test.oss-cn-hangzhou.aliyuncs.com/test.txt`。
2. 尝试使用Ajax技术来直接访问这个网站。

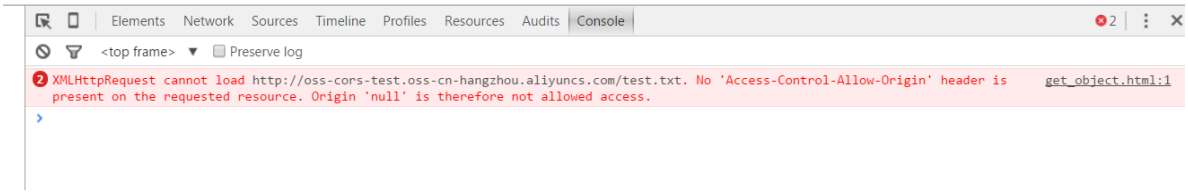
   写一个最简单的访问的HTML。可以将以下代码复制到本地保存成html文件，之后使用浏览器打开。因为没有设置自定义的头，因此这个请求是不需要预检的。

```
<!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) {
```

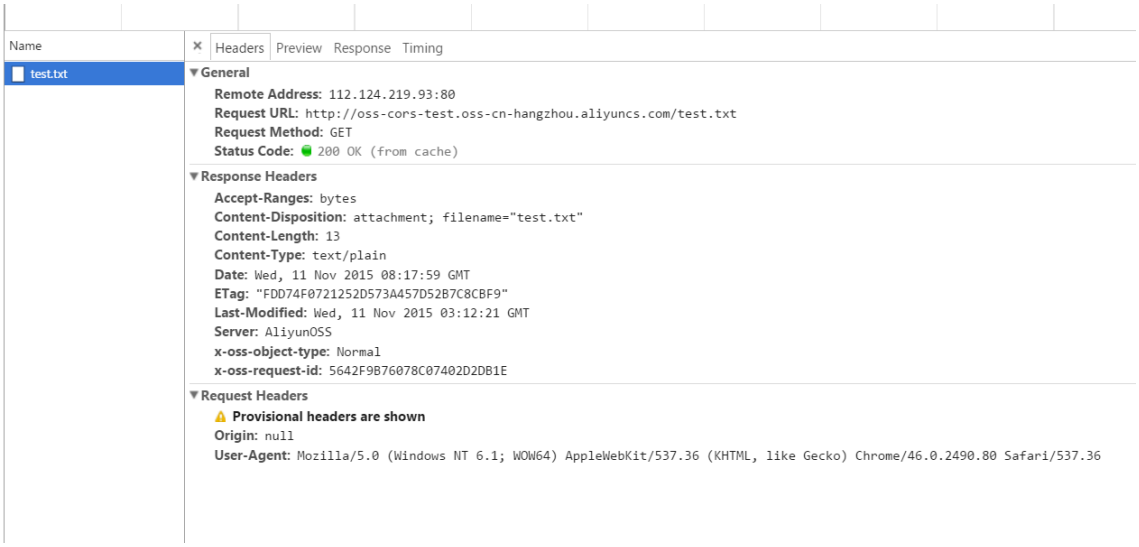
```
    // XHR for Chrome/Firefox/Opera/Safari.
    xhr.open(method, url, true);
  } else if (typeof XDomainRequest != "undefined") {
    // XDomainRequest for IE.
    xhr = new XDomainRequest();
    xhr.open(method, url);
  } else {
    // CORS not supported.
    xhr = null;
  }
  return xhr;
}
// Make 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>
```

3. **打开文件后单击链接（以Chrome浏览器为例），提示该链接无法访问。**
4. **利用Chrome的开发者工具来查看错误原因。**

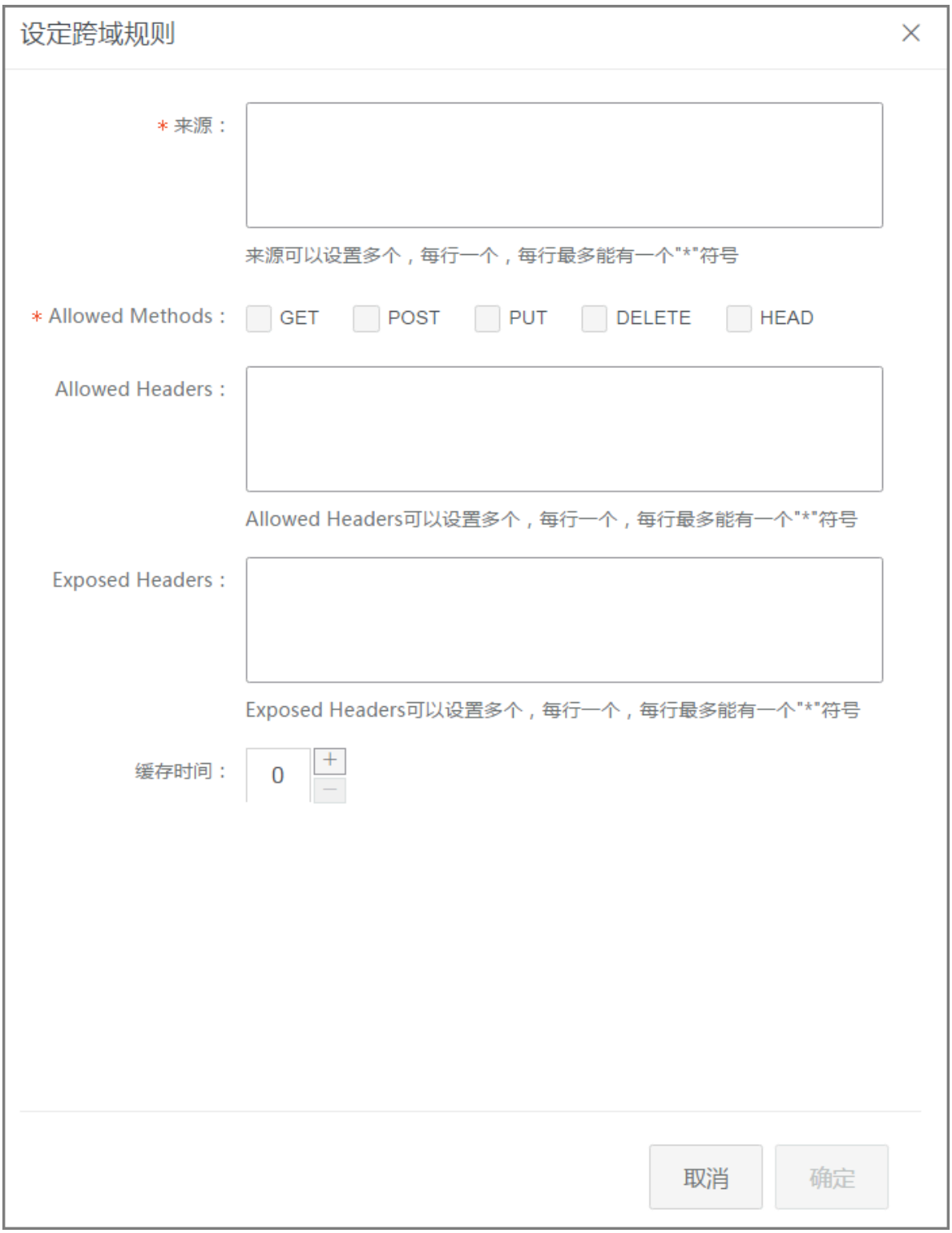
   错误原因是因为没有找到`Access-Control-Allow-Origin`这个头。因为服务器没有配置CORS。



5. 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，因为*test.txt*在Chrome上是本地文件，所以Origin是null。



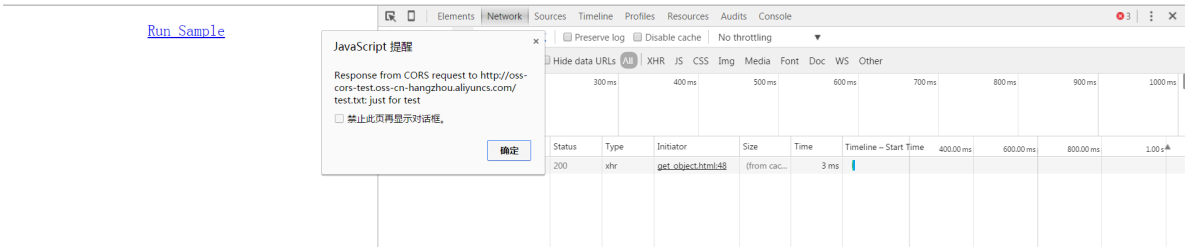
6. 设置CORS，使上文的例子能够执行成功。这里使用控制台来完成CORS的设置。如果用户的CORS设置不是很复杂，也建议使用控制台来完成CORS设置。CORS设置的界面如下。

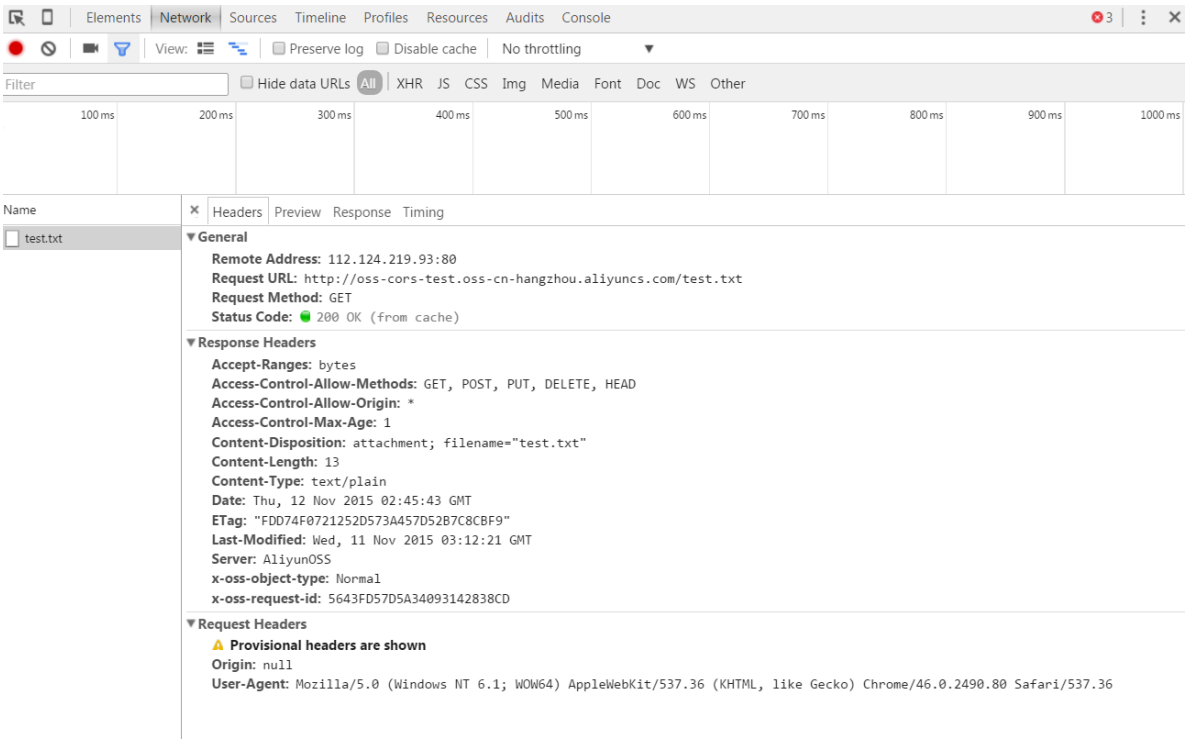


CORS设置是由一条一条规则组成的，真正匹配的时候会从第一条开始逐条匹配，以最早匹配上的规则为准。现在添加第一条规则，使用最宽松的配置，允许所有的Origin、所有的请求类型、所有的Header，最大的缓存时间为1s。


设定跨域规则
* 来源：
*
来源可以设置多个，每行一个，每行最多能有一个"*"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

7. 配置完成之后重新测试，已经可以正常访问。

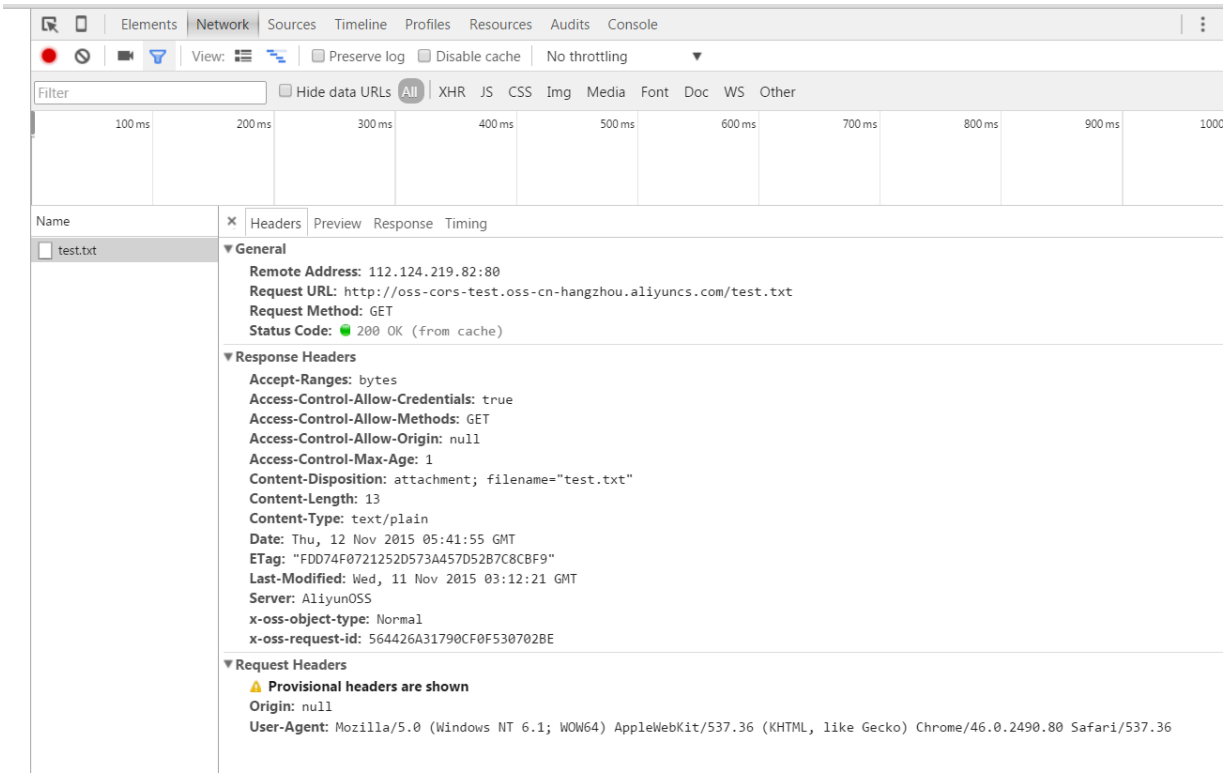




对于排查问题来说，如果想要排除跨域带来的访问问题，可以将CORS设置为最宽松的配置。这种配置是允许所有的跨域请求的一种配置。如果这种配置下依然出错，就表明错误出现在其他的部分，而不是CORS。

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。例如，对于上面的场景，可以使用如下最小的配置进行匹配。


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



说明:

实际使用中，建议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

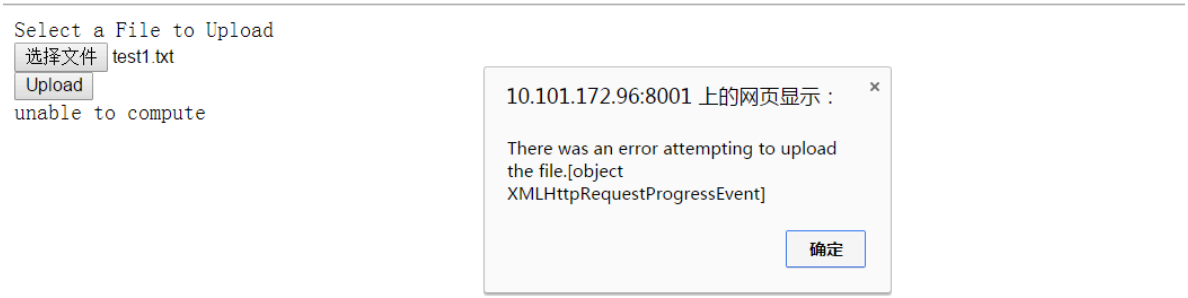
下面提供一个使用了带签名的POST请求，而且需要发送预检请求的例子。

1. 下载PostObjectSample代码，将以下对应的部分修改成根据自己的实际环境修改，之后使用自己的服务器运行此代码。

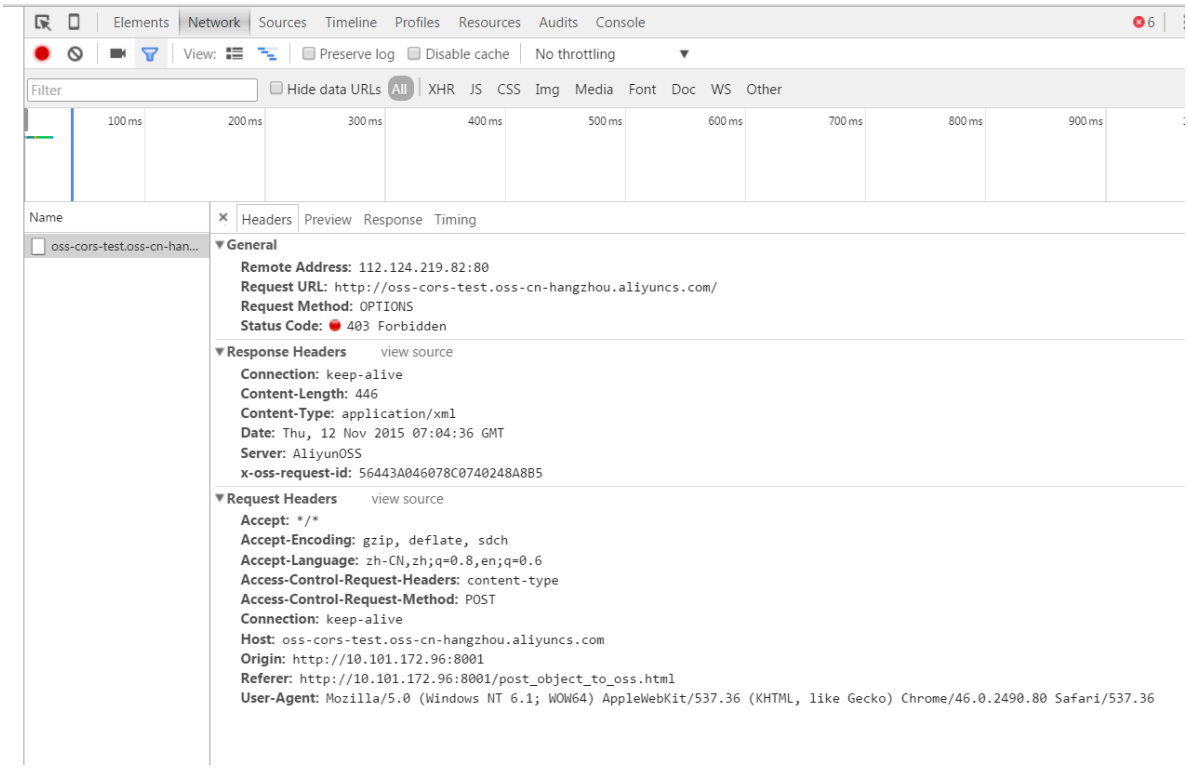
```
            conditions : [
            ["starts-with", "$key", ""],
            {"bucket": 'BUCKET'},
            ["starts-with", "$Content-Type", ""],
            ["content-length-range", 0, 524288000]
            ]
          };
  var secret = 'KEY';
  var policyBase64 = Base64.encode(JSON.stringify(POLICY_JSON));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

2. 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。
3. 访问这个网页，选择一个文件上传。

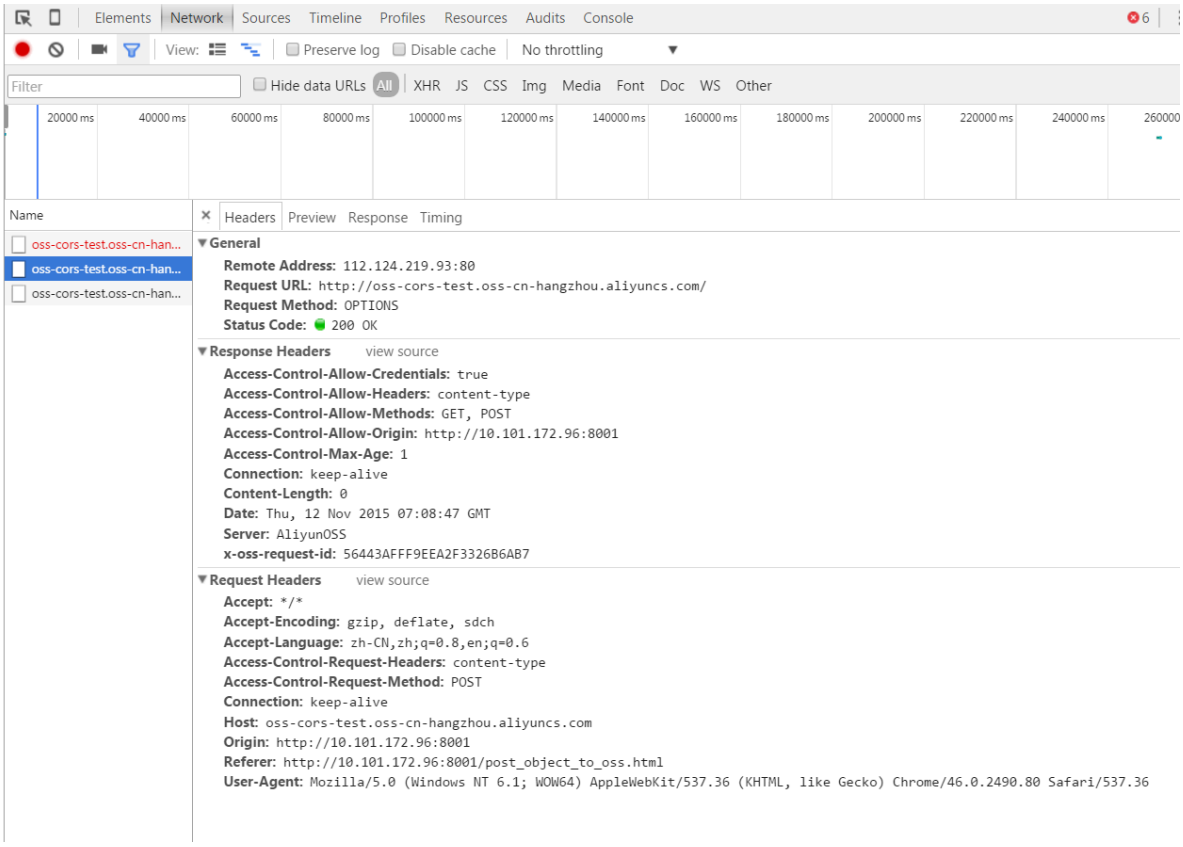


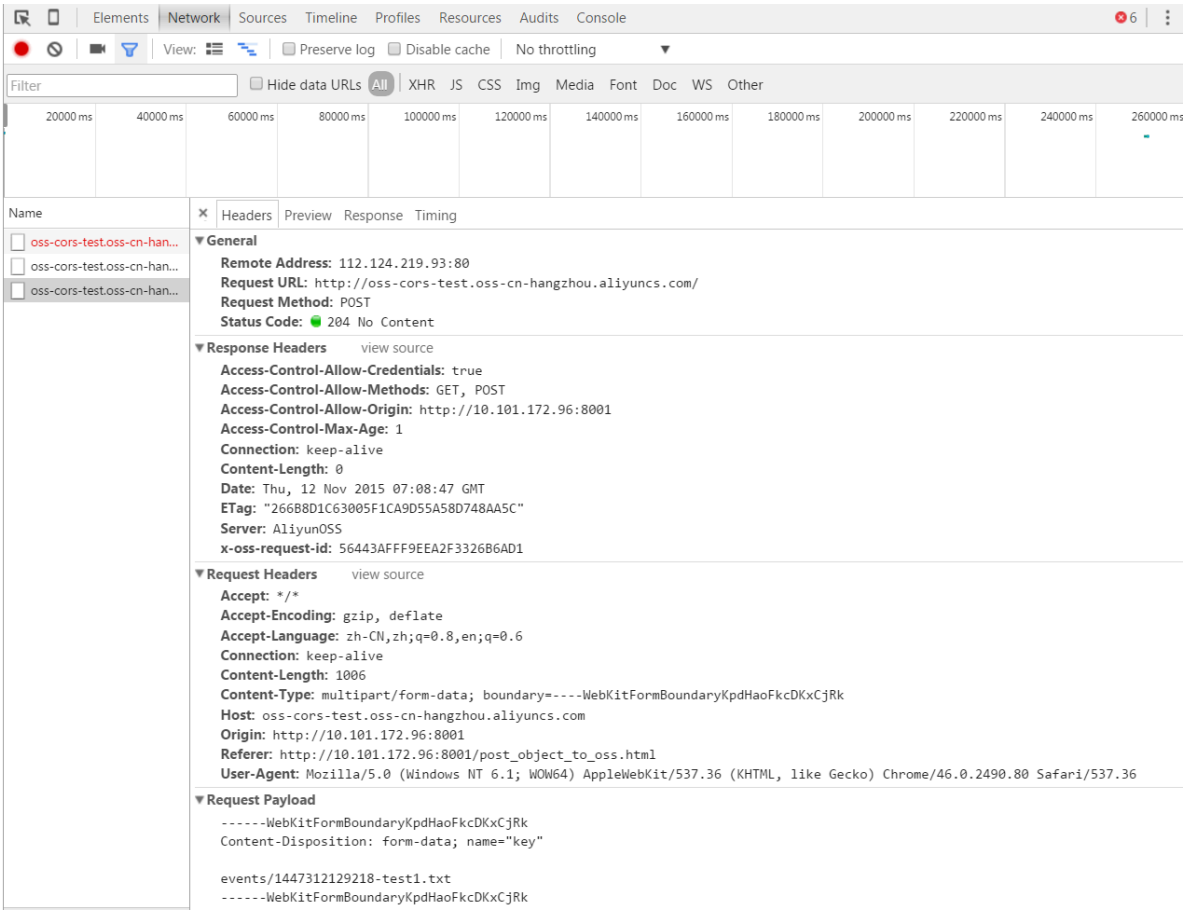
4. 同样打开开发者工具，可以看到以下内容。根据上面的Get的例子确认同样发生了跨域的错误。但是这里与Get请求的区别在于这是一个带预检的请求，可以从图中看到是OPTIONS的Response没有携带CORS相关的Header，所以失败了。



5. 根据对应的信息修改Bucket的CORS配置。

6. 重新运行代码成功，从控制台中也可以看到新上传的文件。





7. 测试访问以下内容正常。

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

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。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。

错误排查

在调试类似程序的时候，容易将其他错误和CORS错误弄混淆。

举个例子，当用户因为签名错误而访问被拒绝的时候，这是由于权限验证发生在CORS验证之前，返回的结果中并没有带上CORS的Header信息，所以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，我们有两种排查方式：

- 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，所以像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。

8.2 防盗链

本文介绍对象存储OSS的防盗链功能原理及实现方法。

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：

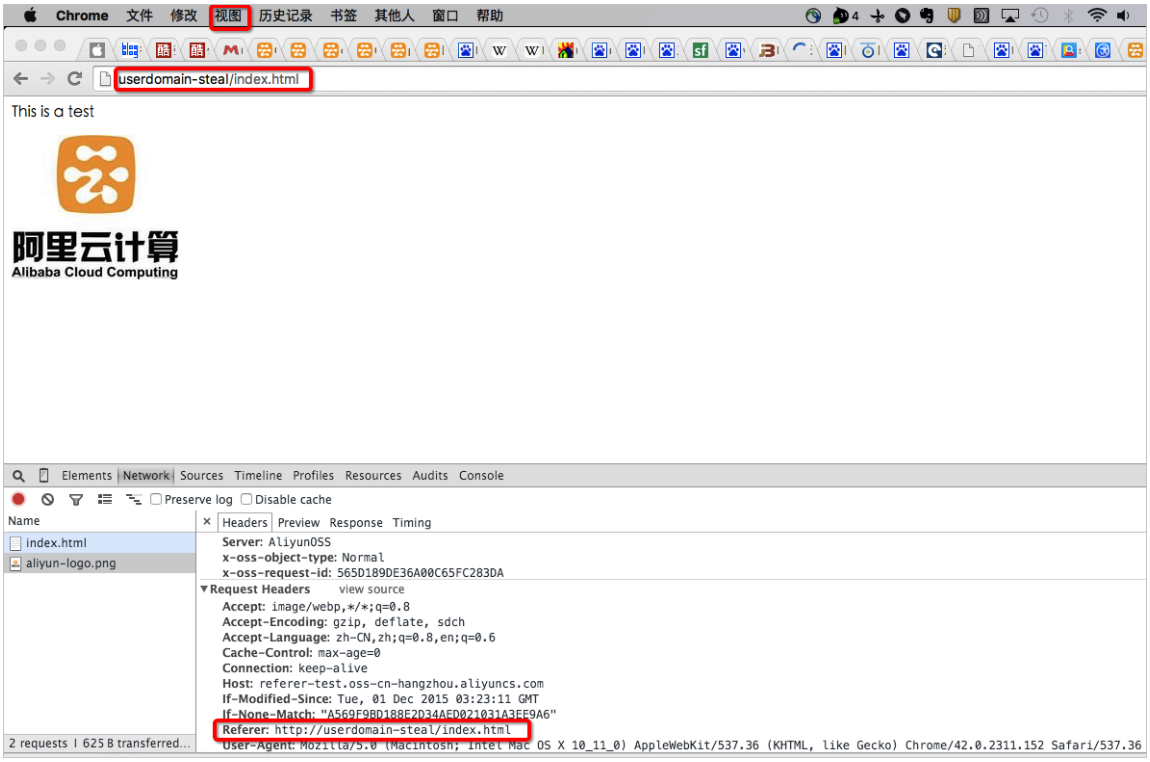
```
<html>
    <p>This is a test</p>
    <img src="http://referer-test.oss-cn-hangzhou.aliyuncs.com/
aliyun-logo.png" />
</html>
```

假设源站为userdomain的网页如下：

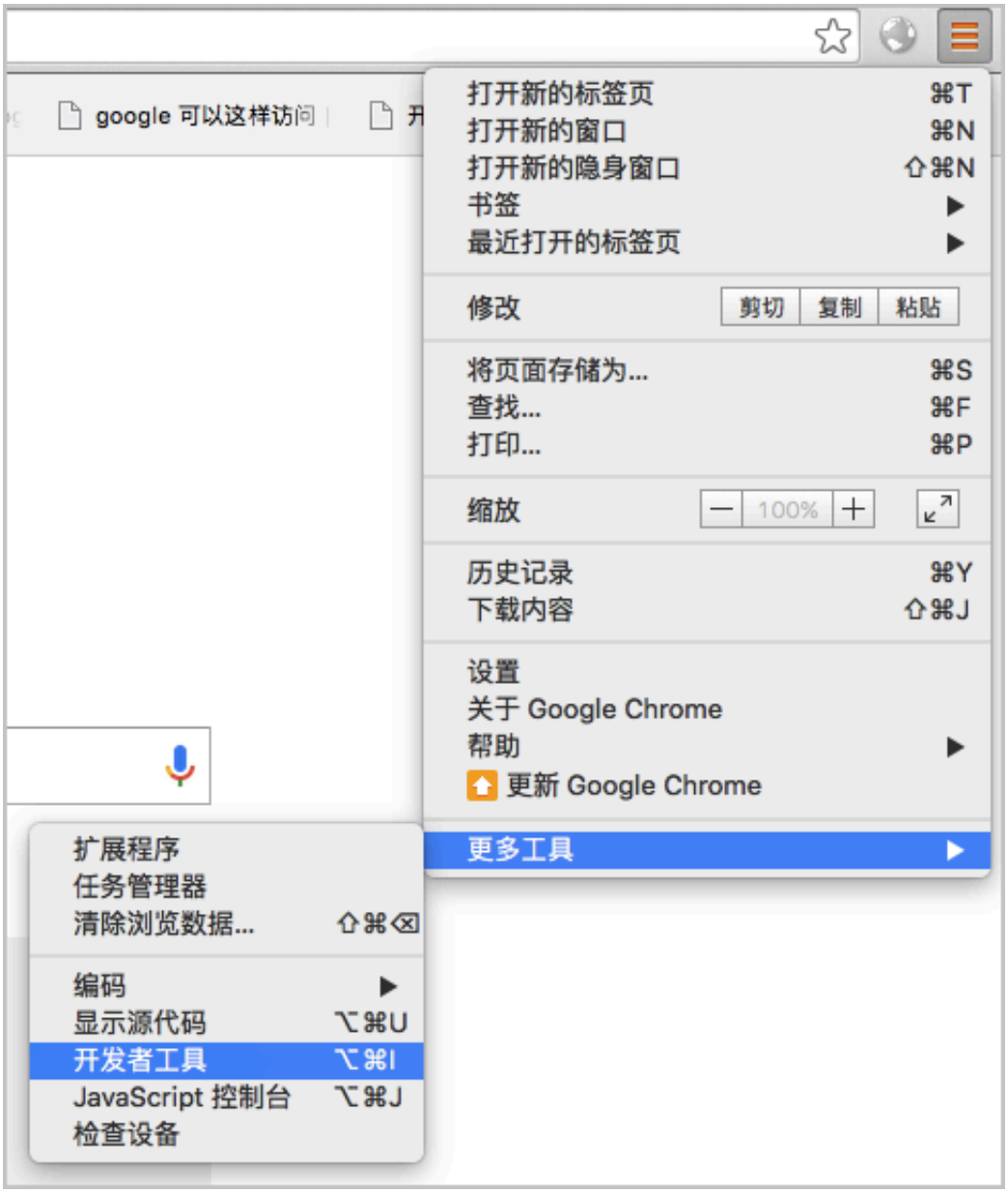
```
<html>
    <p>This is my test link from OSS URL</p>
    <img src="http://referer-test.oss-cn-hangzhou.aliyuncs.com/
aliyun-logo.png" />
</html>
```

- 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

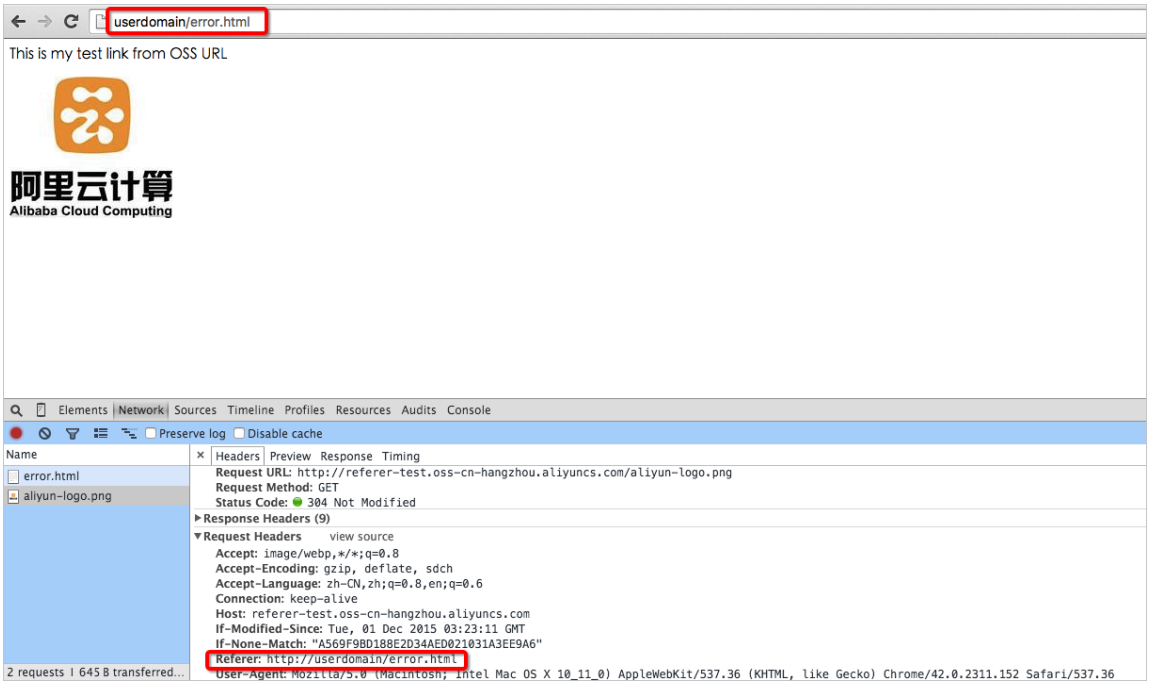
个域名(oss-cn-hangzhou.aliyuncs.com)，浏览器就会在HTTP请求的Header中带上Referer，如图所示。



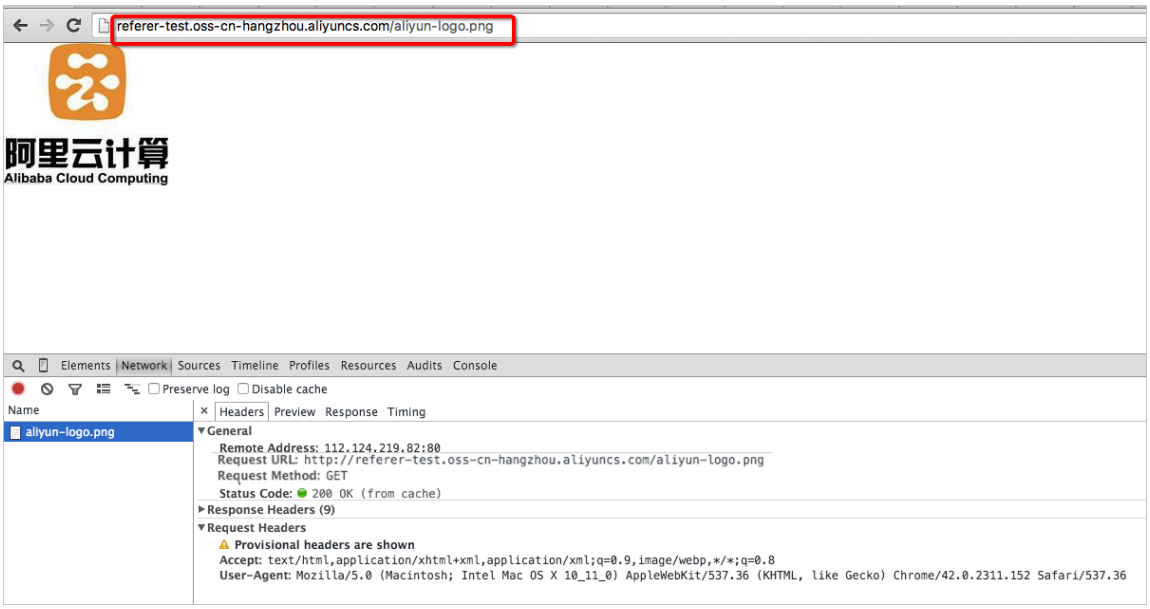
可以看到浏览器在HTTP请求中的Referer为`http://userdomain-steal/index.html`。本文主要是使用chrome的开发者模式来查看网页请求的，如图所示。



- **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。访问日志参见实时日志查询。

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。

8.3 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

   本文的域名是从万网购买的，申请了一个leo23.xyz的域名。如果需要更多域名方面的帮助，请参见万网。

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>
```

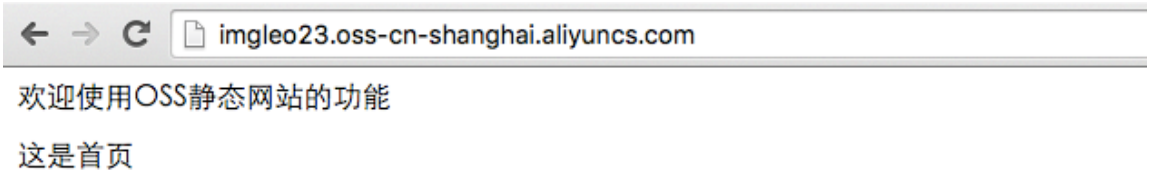
- aliyun-logo.png 是一张图片。

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

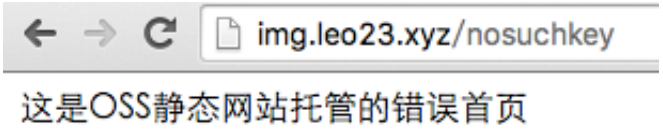
可以看到输入的URL有对应的文件时，会读取成功。

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。

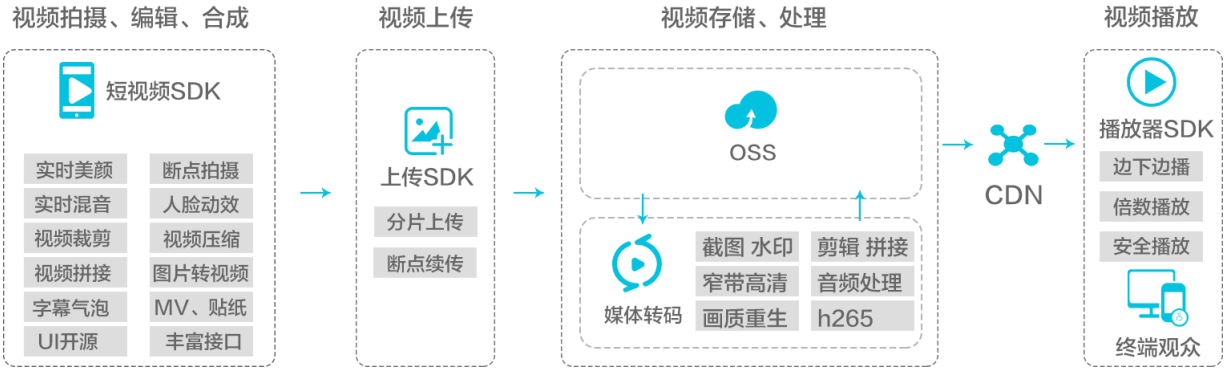
- 为什么输入的URL在浏览器上返回403？

  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。

9 音视频

9.1 短视频

阿里云短视频SDK提供短视频录制、导入、编辑等功能，结合上传SDK、OSS、MTS、CDN及阿里云播放器，可实现短视频的采集、上传、存储、转码、分发、播放的完整功能。



核心优势

- 快速接入，成本经济

  提供产品级SDK，最快2小时接入，节省自行开发耗费的人力物力，帮助您快速实现APP短视频功能。

- 免费人脸检测SDK

  自带由阿里自研的人脸识别技术，快速稳定且高效，助你快速低成本实现视频萌拍效果。

- 接口简单，开放强性

  接口简单易用，开放性强，专业版（UI开源）可以根据业务自由定制UI。

- 功能齐备，应用广泛

  录制功能自带断点录制、实时滤镜、高效美颜、人脸贴图接口功能，支持本地视频导入压缩裁剪，对视频添加MV、动图、字幕、音乐等高级功能。

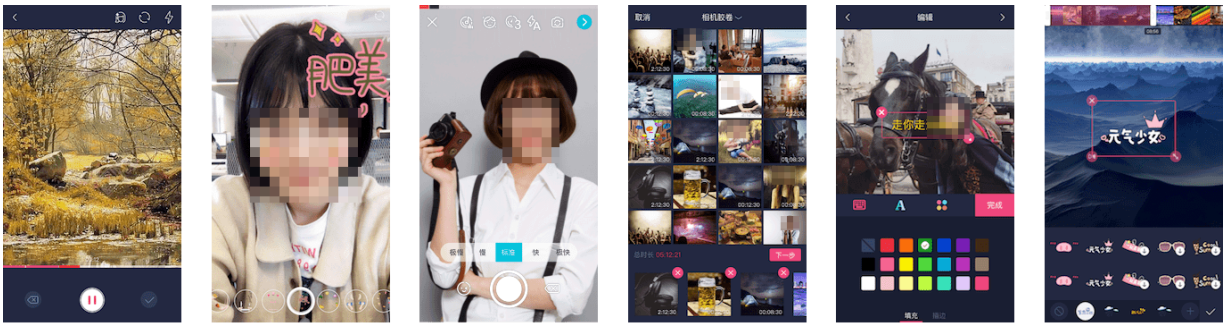
- 迭代打磨，稳定可靠

  视频技术经钉钉、美柚、梨视频、迅雷、贝贝网、宝宝树、蚂蜂窝等1000多家应用商用验证，稳定可靠。

应用场景

短视频目前已成为APP中的标配功能，SDK适合有短视频UGC场景的APP。

专业版SDK界面



断点录制 人脸检测+贴纸 变速录制+音乐 图片、多视频导入 字幕（气泡字幕） 视频动静态贴纸

专业版SDK详细功能

功能点	功能说明
自定义UI	SDK包含一套默认的UI，布局、交互、界面可二次开发，基础版支持图标和背景颜色替换，标准版UI完全自定义。
UI开源	提供完整的UI交互源码，用户可定制UI界面。
多段录制	支持断点拍摄和连续拍摄。
自定义时长	自定义最长和最短拍摄时长。
摄像头切换	可选择使用前后摄像头进行录制。

闪光灯	支持打开、关闭、自动闪光灯模式。
实时水印	支持在录制时添加水印。
焦距调节	录制中可调节画面焦距进行放大缩小。
自定义分辨率及质量	可设定拍摄的画面尺寸、比例和质量，基础版仅支持9:16、3:4、1:1三种比例，标准版、专业版支持任意分辨率。
美颜	录制实时美颜，平滑无极调整强度。
实时滤镜	拍摄预览界面实时切换滤镜。
人脸识别	内置免费人脸检测功能，结合贴纸可实现添加实时追踪人脸挂件功能。
第三方人脸SDK	支持第三方人脸检测SDK。
实时混音和变速	支持录制界面添加音乐和变速功能。变速支持调整倍数。
相册选择	支持从相册过滤视频，也支持视频时长过滤。
照片裁剪	支持照片画面大小的裁剪，同时支持画面填充和画面裁剪。
视频裁剪	支持视频画面大小和时长裁剪，同时支持画面填充和画面裁剪。
原比例裁剪	支持保持原始视频比例裁剪视频时长。
单视频导入	支持单视频导入，跳转进入用户定义的页面。
多照片导入	支持多张照片导入，跳转进入用户定义的页面。
多视频导入	支持多视频导入，进入编辑界面。
视频和照片导入	支持多个视频多张照片混合导入，进入编辑界面。
滤镜	在编辑界面添加滤镜，切换滤镜。
动图	在编辑界面添加动图，可在任意时间点添加并支持时间调整。
MV	在编辑界面添加MV效果，切换MV。
音乐	支持将网络音乐和本地音乐合成到视频中。
静音	支持消除当前视频的原音和音乐声音。
字幕	支持普通文字字幕和气泡效果字幕，并且可更换字体。
片尾	支持在视频末尾添加片尾水印效果，可定义持续时间。

涂鸦	支持画笔尺寸和颜色调整。
播放器SDK	提供主流播放功能，支持秒开、边下边播、SEEK、安全播放、倍数播放等。

专业版SDK下载及试用

- 请参见阿里云短视频SDK专业版下载地址和开发文档。
- 如需试用请发送公司名称、应用名称、申请试用的SDK版本、联系人、联系电话、应用bundleID、包名和签名信息（MD5格式小写无冒号）、阿里云的账号/UID（若没有账号请注册）至videosdk@service.aliyun.com，申请开通试用，试用期为一个月。

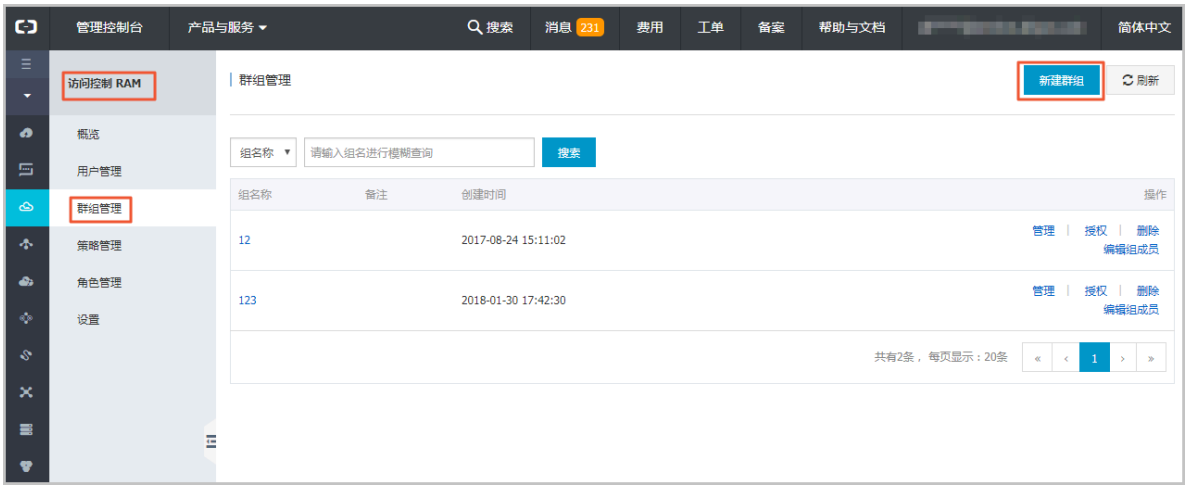
Web端上传视频到OSS

1. 注册阿里云用户，登录阿里云首页，选择对象存储产品，单击立刻开通，如下图所示：

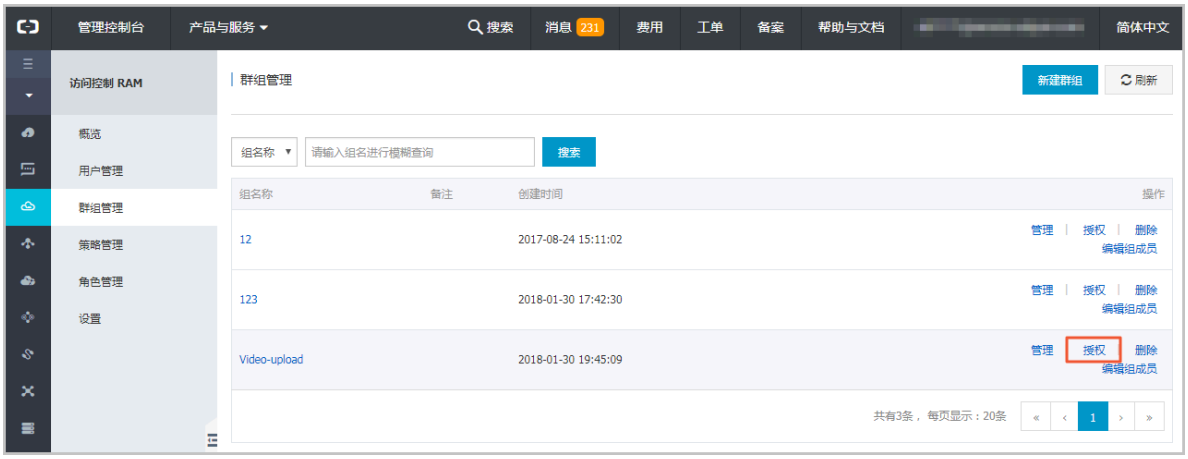
2. 新建Bucket，存储类型选择低频访问，读写权限选择私有，如下图所示：

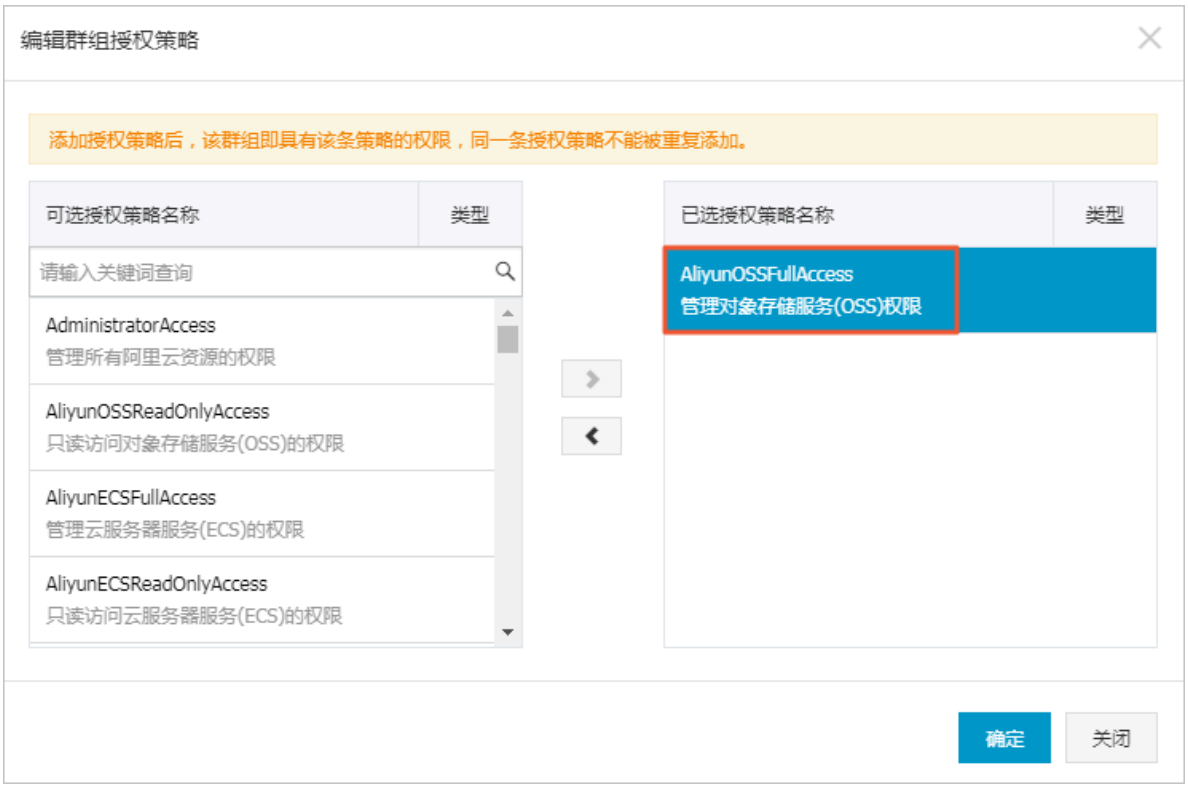
3. 登录OSS管理控制台，单击Access Key。
4. 在弹出的对话框中，选择开始使用子用户AccessKey，创建新的子用户。
5. 在步骤选择权限中，选择AliyunOSSFullAccess。

6. 在OSS管理控制台中，选择访问控制 > 群组管理 > 新建群组，如下图所示：

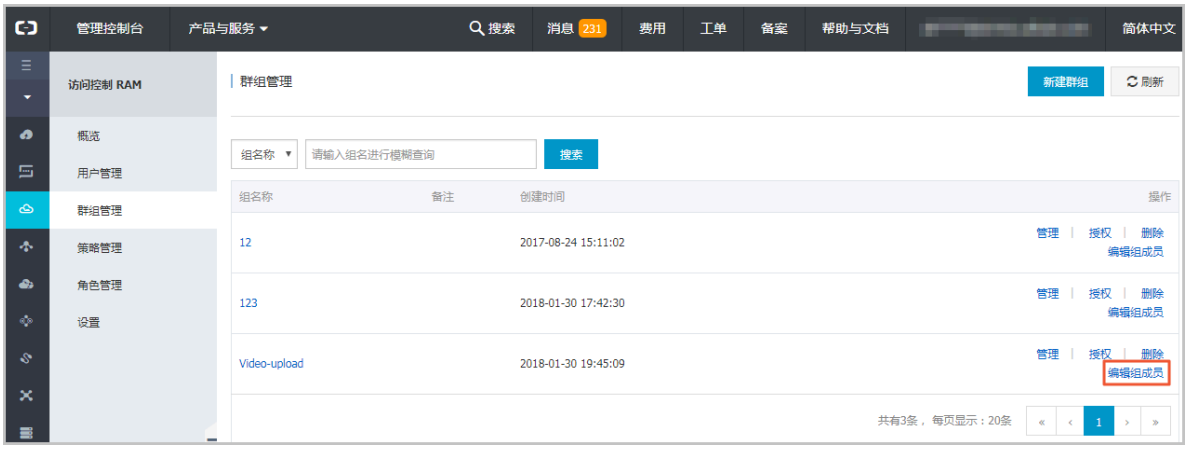


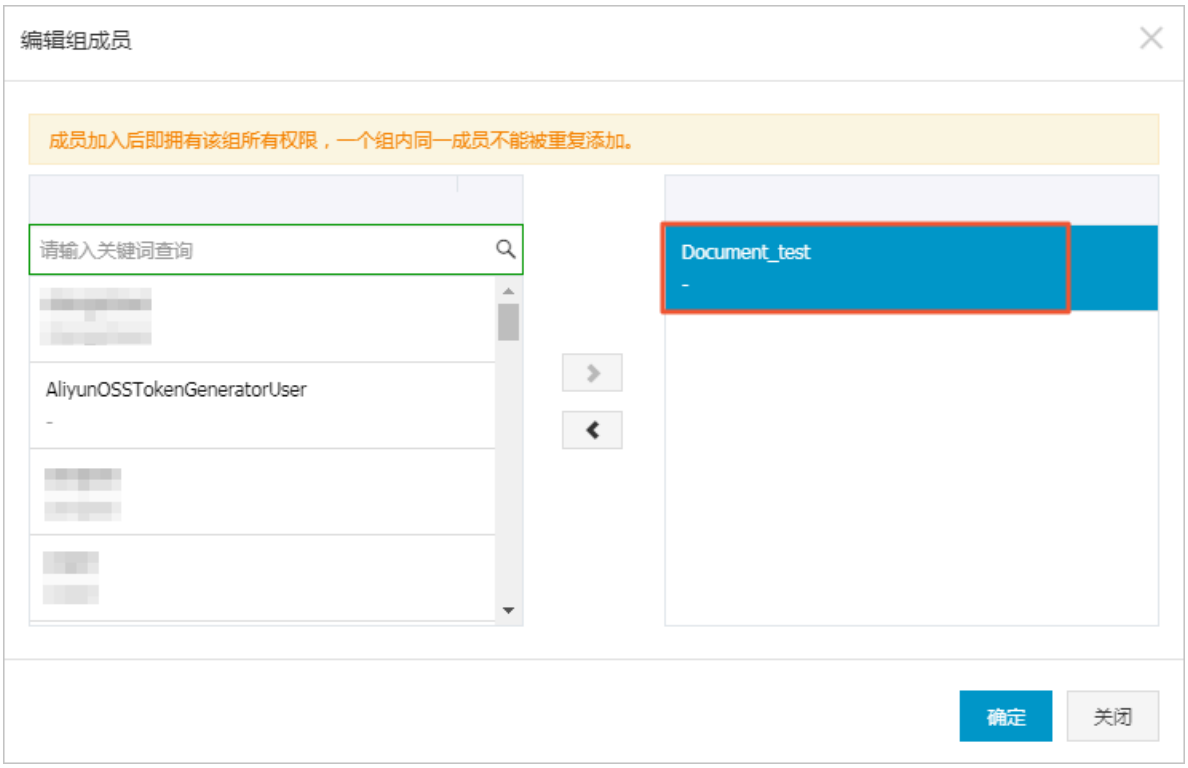
7. 在新建的群组，选择授权，为群组选择AliyunOSSFullAccess权限，如下图所示：





8. 在新建的群组，选择编辑组成员，将刚才创建的子用户设置为组成员，如下图所示：

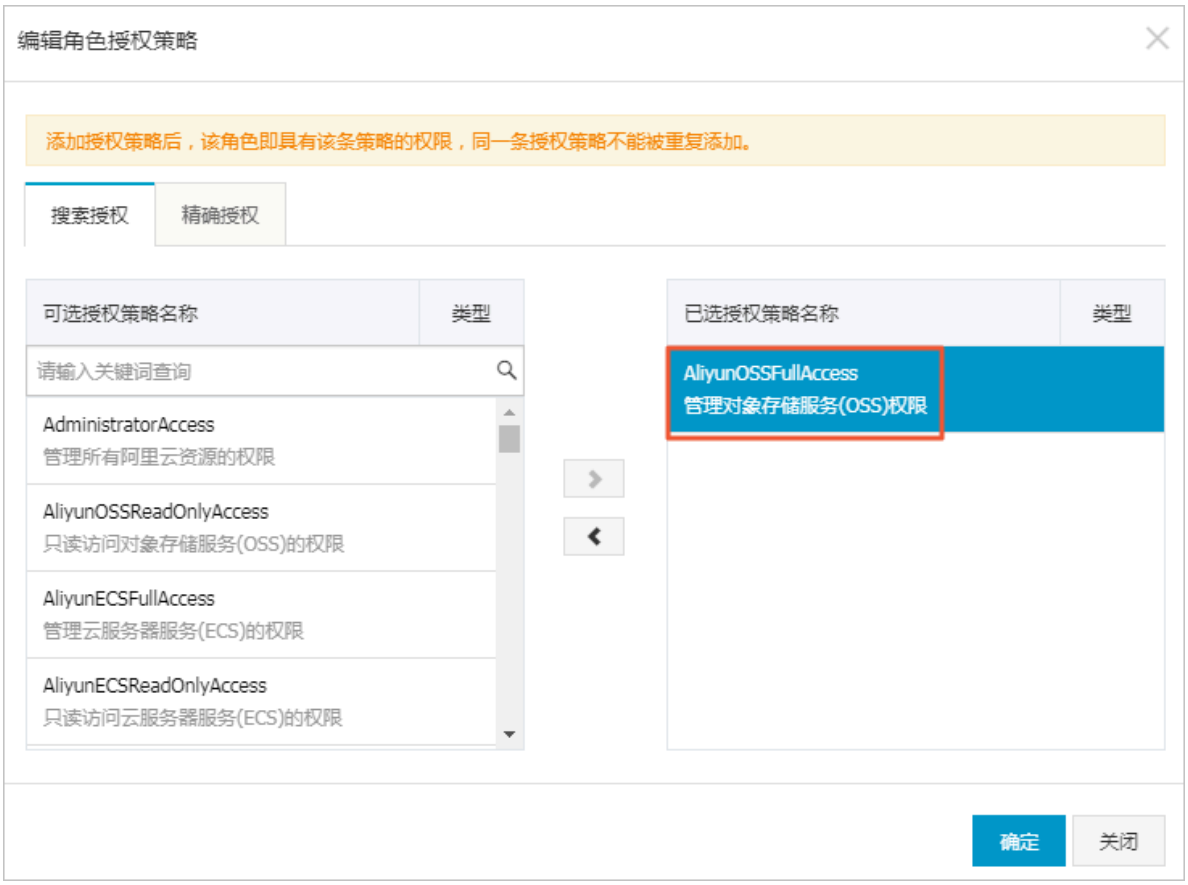




9. 在OSS管理控制台，选择访问控制 RAM > 角色管理 > 新建角色。

10.在步骤选择角色类型中，选择用户角色。在步骤填写类型信息中，选择当前云账号。

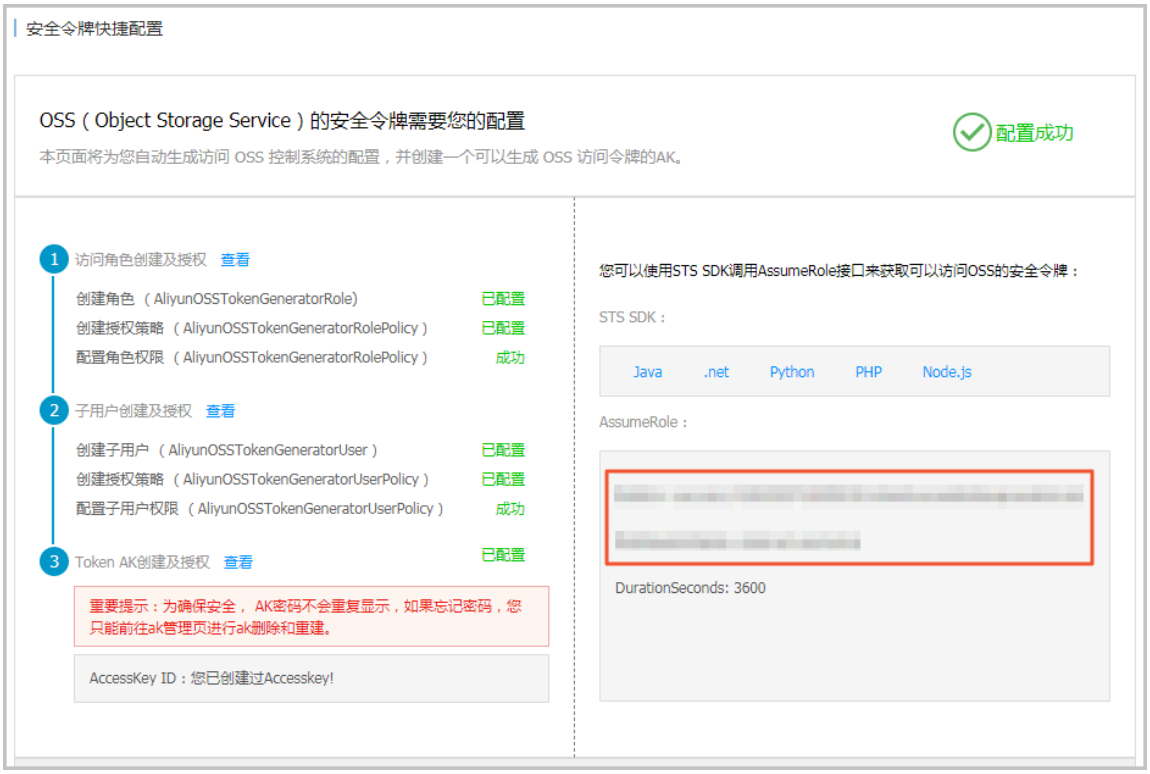
11.选择新建的角色，选择角色授权策略 > 编辑授权策略，为角色选择AliyunOSSFullAccess权限，如下图所示：



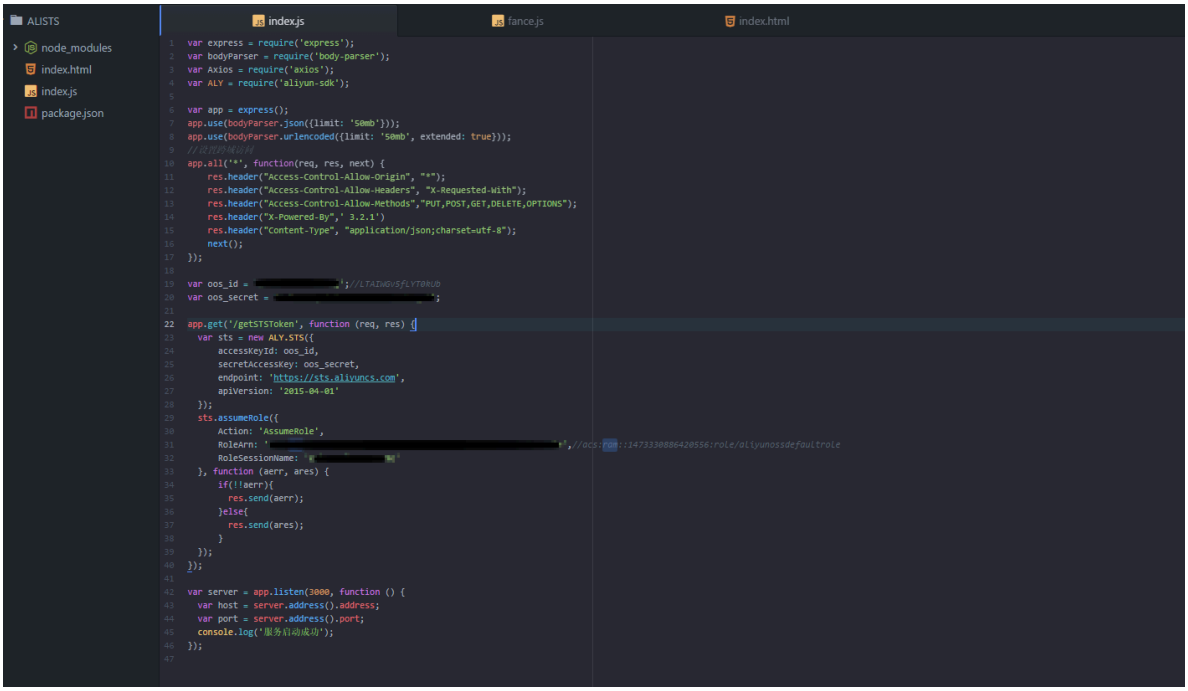
12.返回OSS管理控制台，单击安全令牌 > 开始授权，保存AccessKey、AssumeRole。

13.参考示例，编写获取安全令牌的服务。有关STS权限的介绍，请参考STS权限介绍。

- 所需的 accessKeyId 及 secretAccessKey，请参见之前保存的AcessKey信息。
- 所需的 RoleArn 及 RoleSessionName，可在OSS管理控制台中单击安全令牌，在显示的配置页面中获取，如下图所示：



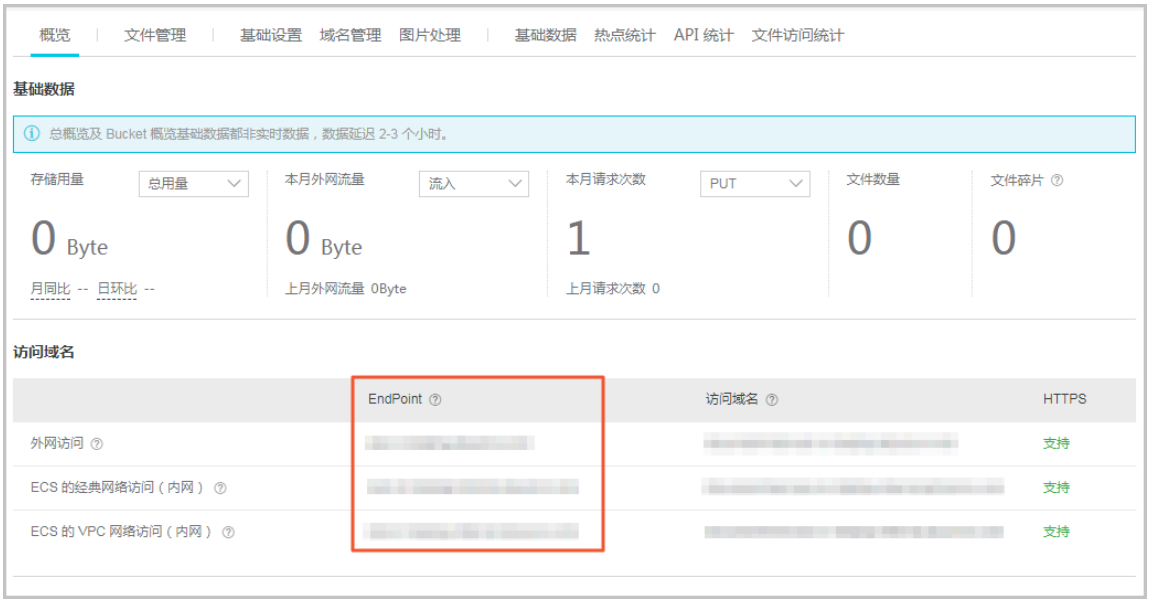
14.通过express搭建node服务器，监听接口、执行请求，并将token返回给客户端，如下图所示：



15.在本地启动node.js，引入`< script src="http://gosspublic.alicdn.com/aliyun-oss-sdk-4.3.0.min.js"></ /script>`，具体代码如下图所示：

```
<input type="file" id="file" />
<script src="http://gosspublic.alicdn.com/aliyun-oss-sdk-4.3.0.min.js"></script>
<script type="text/javascript">
  var client;//上传文件对象实例
  var storeAs = 'upload-file';//上传文件名称,最好唯一
  document.getElementById('file').addEventListener('change', function (e) {
    var file = e.target.files[0];
    OSS.urllib.request("http://localhost:3000/getSTSToken",{method: 'GET'},//获取tocken
        function (err, response) {
          if (err) {
            return alert(err);
          }
          try {
            result = JSON.parse(response);
          } catch (e) {
            return alert('parse sts response info error: ' + e.message);
          }
          client = new OSS.Wrapper({//根据返回的tocken创建上传实例对象
            accessKeyId: result.Credentials.AccessKeyId,
            accessKeySecret: result.Credentials.AccessKeySecret,
            stsToken: result.Credentials.SecurityToken,
            endpoint: '[redacted]',
            bucket: '[redacted]'
          });
          client.multipartUpload(storeAs, file).then(function (result) {//执行上传
            console.log(result);
            getLoadUrl();//获取下载地址
          }).catch(function (err) {
            console.log(err);
          });
      });
  });
  function getLoadUrl(){
    var url = client.signatureUrl(storeAs);
    console.log(url);
  }
```

- 代码中的 bucket 填写最开始创建的空间名称。
- 代码中的 endpoint，可在OSS管理控制台中选择创建的bucket，在页面Endpoint区域获取。

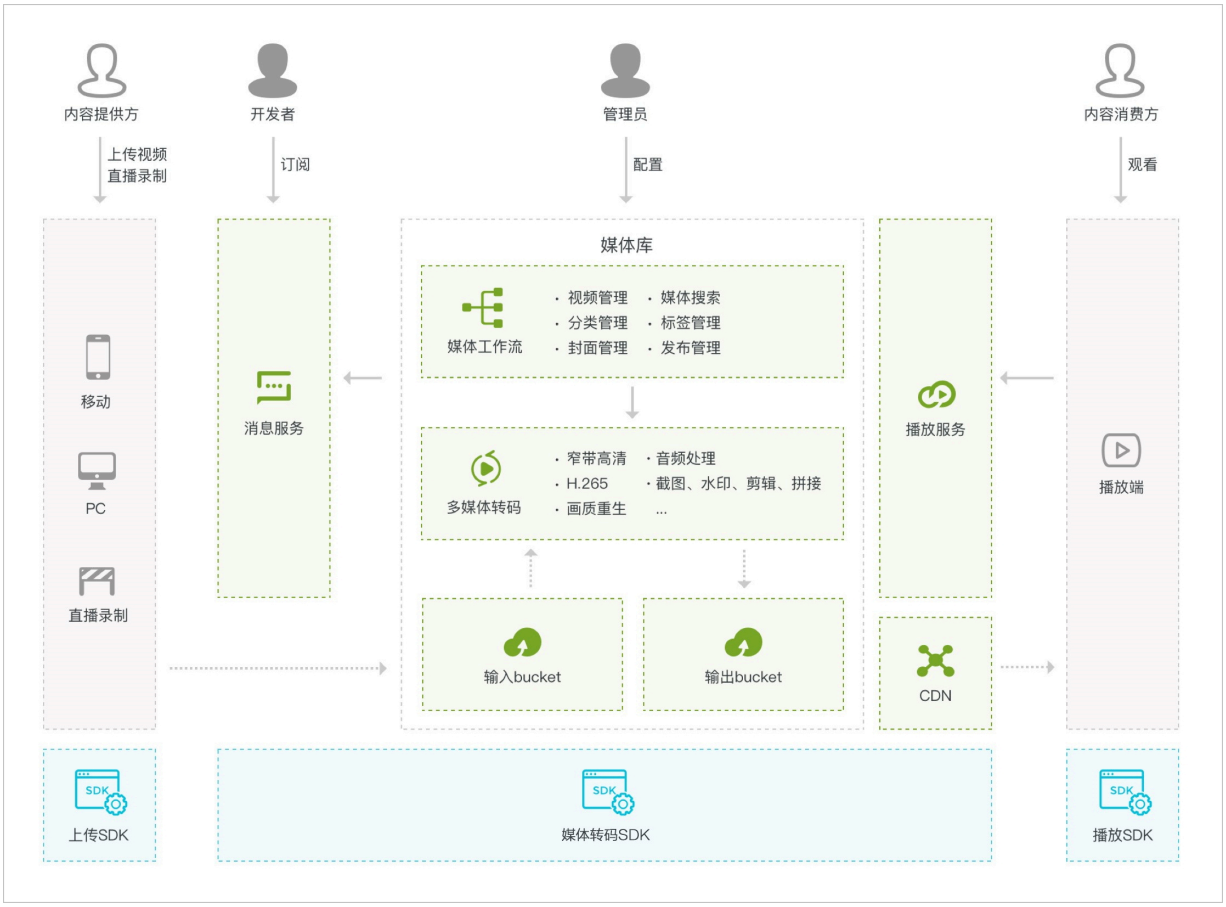


9.2 音视频转码

存储在OSS上的多媒体音视频数据，可以通过经济、弹性、高扩展的阿里云媒体转码服务，转换成适合在移动端、PC、TV上播放的格式。

媒体转码核心能力包括：

- 转换媒体格式，支持多平台播放。
- 保证相同画质质量的前提下，调整视频码率、提高视频压缩效率、减小文件体积，从而减少播放卡顿并节省存储空间和流量费用。
- 添加水印logo，突出品牌，增加产品识别度。
- 对视频进行剪辑/拼接等二次创作。
- 针对画质较差的视频，去除画面中的毛刺、马赛克等，修复为高清晰版本。



核心优势

- 高性价比
  - 无需前期投资，只按实际用量付费。
  - 窄带高清和H.265技术，同等视频质量，文件更小，更省流量。

- 强大的转码能力
  - 高速稳定的并行转码系统，按需动态调整转码资源，自动扩容/缩容，应对高并发转码需求无缝扩展集群资源。
- 专业的转码算法
  - 强大的计算资源，先进的视频处理算法，业界独有的画质重生技术，将现存普通或受损的影视内容重制为超高清或画质修复的版本。
- 功能丰富、高可定制
  - 视频转码、截图、水印、剪辑、拼接等丰富的媒体转码功能满足各种应用场景。
  - 高可扩展的媒体转码模板，支持自定义转码参数，满足多样化转码需求。
- 易用的媒体工作流
  - 自定义媒体工作流，文件上传完毕自动触发执行媒体工作流转码，消息机制实时状态更新，1分钟搭建常见视频处理流程。

客户案例

客户	业务介绍
YOUKU 优酷	视频网站：提供集音视频上传、自动化转码、媒体资源管理、分发加速于一体的一站式音视频点播解决方案。帮助快速搭建安全、弹性、高可定制的点播平台和应用。
智慧树网	在线教育：窄带高清2.0优化录播视频画质，让教学课件更清晰，同时丰富的视频功能助您快速构建自己的教学平台！满足教师分享课程视频的需求，实现对教育资源的统一管理。

客户	业务介绍
	电视传媒：满足传统广电视频技术规范要求的转码服务。倍速转码功能可提供更高转码效率。丰富的院线大片优化处理经验提供更佳画质更流畅播放体验。
	游戏视频：用户可快速搭建一个功能齐备的游戏视频发布平台，为游戏门户提供完善的视频服务。

媒体转码详细功能

功能点	说明
输入格式全覆盖	输入格式全覆盖，请参考功能特性。
支持常用输出格式	· 视频：FLV、MP4、M3U8（TS） · 音频：MP3、MP4、OGG、FLAC · 图片：gif、webp
截图	· 支持对存储于OSS上的视频文件截取指定时间的JPG格式图像。 · 支持单张截图、多张截图、平均截图。
水印	· 静态水印：支持在输出的视频上覆盖最多20个静态图像，水印图片支持PNG格式。 · 动态水印：在视频开头和结尾融入动态logo，突出品牌，增加产品识别度。
媒体信息	支持获取存储于OSS上的音、视频文件的编码和内容信息。

功能点	说明
转码模版	· 预置模版：媒体转码服务为适配一定网络带宽范围的输出视频预设了一系列转码模版。 · 自定义模版：由用户自行定义转码参数的转码模版，它是转码参数（音频、视频、容器等）的集合，可以满足用户个性化的转码需求。
倍速转码	适用于30分钟以上的长视频，通过对视频分片并行转码，大幅提升转码速度，转码速度可提升5倍。
窄带高清	阿里云独家窄带高清技术，更低的带宽成本，享受更清晰的视频体验。
画质重生	· 高帧率重制服务（FRC）：对于30帧/秒以内的普通帧率高清节目，生成60帧/秒甚至120帧/秒的高帧率版本，4K大屏播放也无顿挫感。 · 2K转4K重制服务（2K转4K）：对于1080p影片，利用基于海量视频训练的超分辨率技术，生成独家高品质4K节目源。 · 标清转高清重制服务（SD转HD）：对于标清的经典老片，去除胶片颗粒和压缩噪声，加以超分辨率技术，生成720p甚至1080p的高清版本。 · 片源修复服务（PicRescue）：对于被过度压缩的网络视频，去除画面中的毛刺和马赛克，生成更高清晰度的修复重制版。
视频加密	支持“阿里云私有加密”和“HLS-AES128标准加密”两种加密方式。保护视频内容、防下载，适用于在线教育，付费观看等场景。
剪辑输出	支持指定时间点开始，截取指定时长的媒体剪辑。
视频拼接	最多支持20个视频拼接。
分辨率按比例缩放	转码输出参数中仅指定宽或者高，另一个参数留空，则媒体转码服务会按照原视频的宽高比自动设定另一个参数。
M3U8输出自定义切片时长	支持自定义设置M3U8切片时长，范围从1秒至60秒。有助于用户根据播放端带宽条件来设定切片时长，降低用户首屏加载时间。
音视频抽取	从视频文件中单独分离出音频或视频。

功能点	说明
视频画面旋转	支持输出视频旋转视频画面一定角度。
视频转GIF	支持视频转码为GIF输出。
外挂字幕	转码支持导入外部字幕文件并指定字幕编码格式。
媒资	· 支持媒资库：标题、标签、分类、描述等。 · 媒体工作流：云端自动化处理工作流，音视频上传完毕后自动执行处理流程。

操作指南

1. 登录媒体转码控制台。
2. 提交转码作业。
   a. 单击OSS文件转码管理。
   b. 单击新建转码。
   c. 单击浏览选择待转码文件。
   d. 填写输出文件名。
   e. 单击浏览，选择输出路径，并单击下一步。
   f. 在预置静态模版中选择一个转码模版，单击转码按钮以提交转码作业。

3. 查看转码进度及结果。

单击OSS文件转码管理 > OSS文件转码任务列表，查看转码作业列表、转码进度及转码结果。

- 查看作业列表及转码进度

- 查看作业详情

转码完成后可通过转码作业详情 > 转码输出，单击复制地址可得到输出文件地址。

更多内容

- 媒体转码产品介绍和文档地址
- 窄带高清2.0 让压缩超越极限，了解详情
- 视频安全解决方案，了解详情

10 数据安全

10.1 通过crc64校验数据传输的完整性

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

10.2 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。

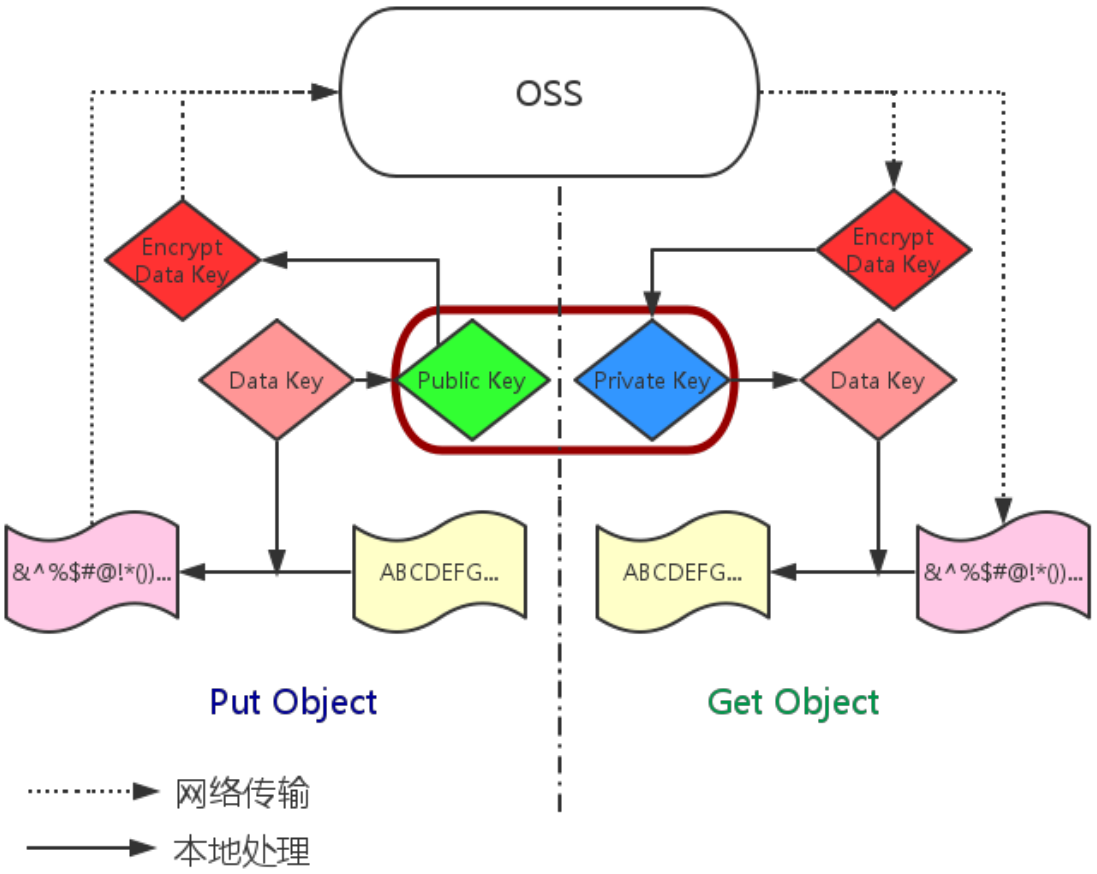
原理介绍

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
```

11 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。本文介绍如何监控OSS资源、设置报警规则以及自定义监控大盘。

前提条件

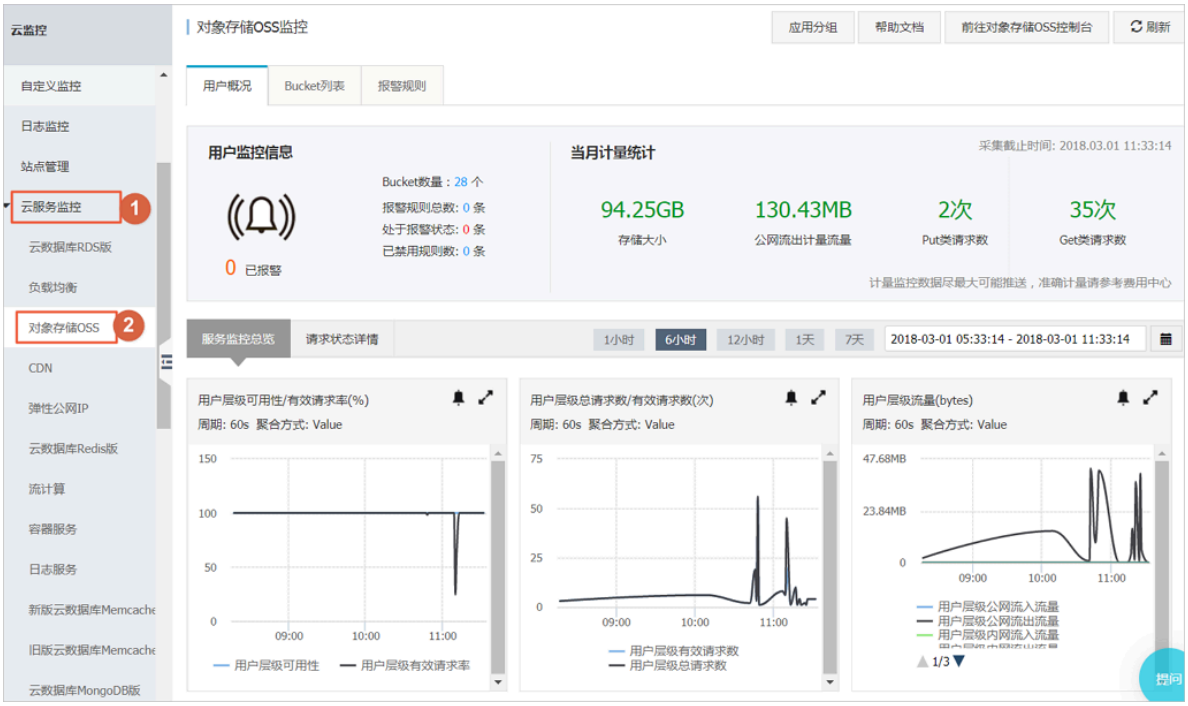
- 已开通OSS服务。
- 已开通云监控服务。

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。

12 OSS性能与扩展性最佳实践

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。当您的请求速率过大时，会导致请求速率下降。

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。

13 安卓应用示例

13.1 OssDemo简介

本文主要是基于移动端开发上传的场景来介绍在Android上如何使用SDK来实现一些常见的操作。

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. OssDemo源码下载后可安装体验，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。

13.2 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);
 }
```

13.3 上传文件

本文介绍在安卓系统中，如何上传文件到OSS。

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; }
```

13.4 图片处理

本文介绍通过OssDemo对已上传的图片添加水印、缩放、裁剪等操作。

和下载不同的是：

- 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. 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，到处理图片页面。
  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API的GetObject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");
```

14 使用ECS实例反向代理OSS

14.1 基于Ubuntu的ECS实例实现OSS反向代理

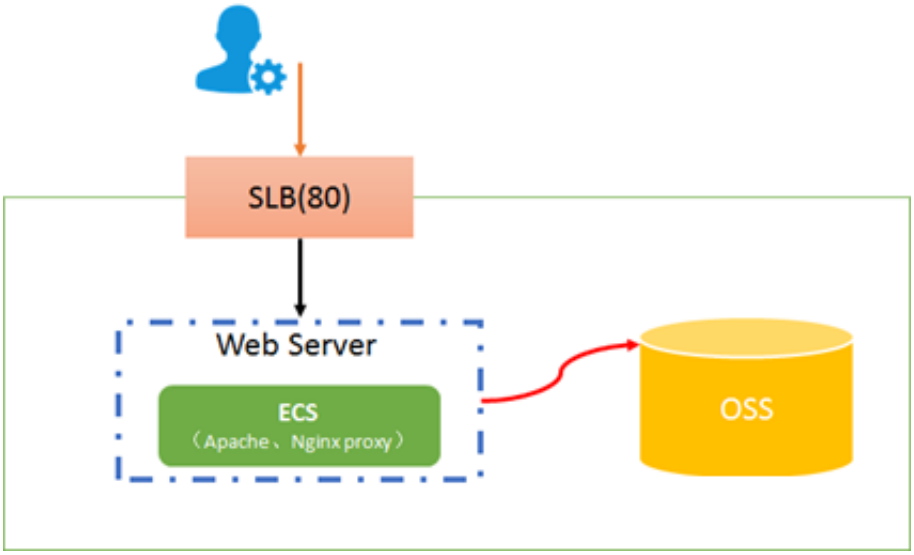
阿里云OSS的存储空间（Bucket）访问地址会随机变换，您可以通过在ECS实例上配置OSS的反向代理，实现通过固定IP地址访问OSS的存储空间。

背景信息

阿里云OSS通过Restful API方式对外提供服务。最终用户通过OSS默认域名或者绑定的自定义域名方式访问，但是在某些场景下，用户需要通过固定的IP地址访问OSS：

- 某些企业由于安全机制，需要在出口防火墙配置策略，以限制内部员工和业务系统只能访问指定的公网IP，但是OSS的Bucket访问IP会随机变换，导致需要经常修改防火墙策略。
- 金融云环境下，因金融云网络架构限制，金融云内网类型的Bucket只能在金融云内部访问，不支持在互联网上直接访问金融云内网类型Bucket。

以上问题可以通过在ECS实例上搭建反向代理的方式访问OSS。



配置步骤

1. 创建一个和对应Bucket相同地域的Ubuntu系统的ECS实例，本文演示系统为Ubuntu 18.04 64位系统。创建过程可参考创建ECS实例。

2. 使用root用户登录ECS实例，并更新apt源：

```
root@test:~# apt-get update
```

3. 安装Nginx：

```
root@test:~# apt-get install nginx
```

说明：

Nginx默认安装位置：

```
/usr/sbin/nginx        主程序
/etc/nginx             存放配置文件
/usr/share/nginx       存放静态文件
/var/log/nginx         存放日志
```

4. 打开Nginx配置文件：

```
root@test:~# vi /etc/nginx/nginx.conf
```

5. 在config文件中的http模块添加如下内容：

```
server {
        listen 80;
        server_name 47.**.**.73; #对外提供反向代理服务的IP，即ECS实例的外网地址；

        location / {
            proxy_pass http://bucketname.oss-cn-beijing-internal.aliyuncs.com; #填写Bucket的内网访问域名，如果ECS实例与Bucket不在同一个地域，需填写外网域名；
        }
```

```
}
```

```
user www-data;
worker_processes auto;
pid /run/nginx.pid;
include /etc/nginx/modules-enabled/*.conf;

events {
        worker_connections 768;
        # multi_accept on;
}

http {
server {
        listen 80;
        server_name 47        73;

       location /{
        proxy_pass http://                              .com;
        }
}
        ##
        # Basic Settings
        ##

                                                        5,0-1         Top
```

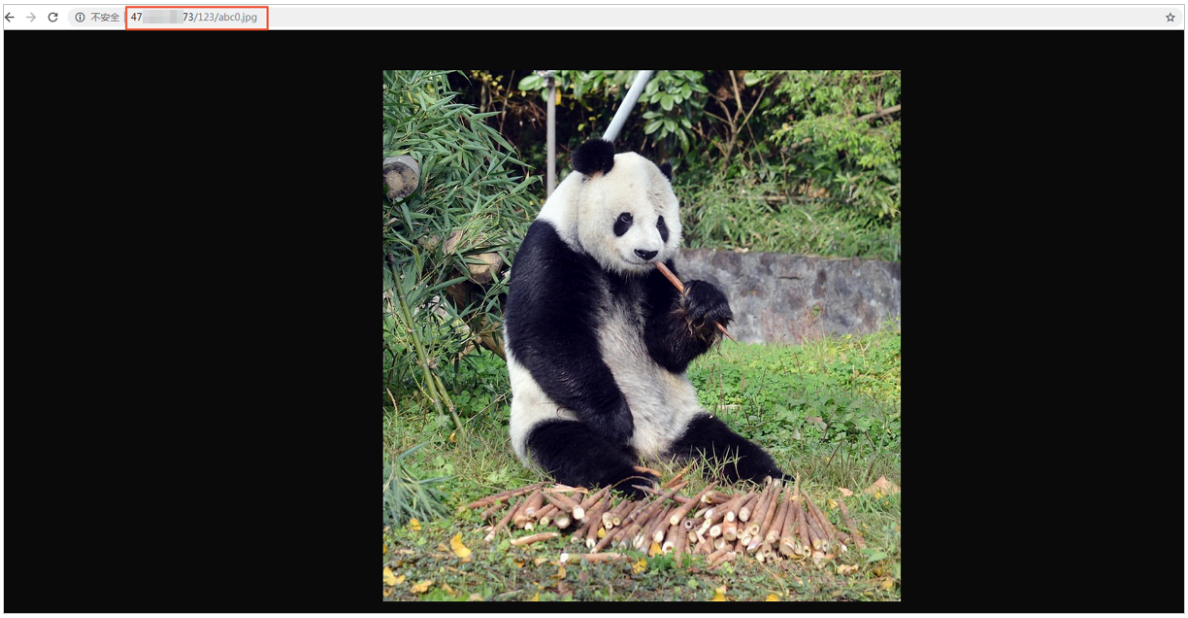
说明:

本文为演示环境，实际环境中，为了您的数据安全，建议配置https模块，配置方法可参考反向代理配置。

6. **进入Nginx主程序文件夹，启动Nginx：**

```
root@test:~# cd /usr/sbin/
root@test:~# ./nginx
```

7. **测试使用ECS外网地址加文件访问路径访问OSS资源。**



更多参考

#unique_164

14.2 基于CentOS的ECS实例实现OSS反向代理

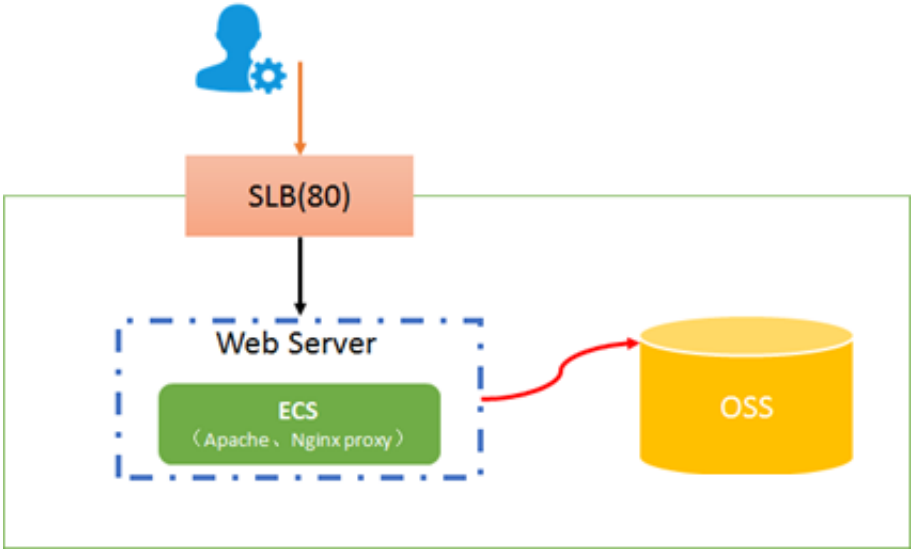
阿里云OSS的存储空间（Bucket）访问地址会随机变换，您可以通过在ECS实例上配置OSS的反向代理，实现通过固定IP地址访问OSS的存储空间。

背景信息

阿里云OSS通过Restful API方式对外提供服务。最终用户通过OSS默认域名或者绑定的自定义域名方式访问，但是在某些场景下，用户需要通过固定的IP地址访问OSS：

- 某些企业由于安全机制，需要在出口防火墙配置策略，以限制内部员工和业务系统只能访问指定的公网IP，但是OSS的Bucket访问IP会随机变换，导致需要经常修改防火墙策略。
- 金融云环境下，因金融云网络架构限制，金融云内网类型的Bucket只能在金融云内部访问，不支持在互联网上直接访问金融云内网类型Bucket。

以上问题可以通过在ECS实例上搭建反向代理的方式访问OSS。



配置步骤

1. 创建一个和对应Bucket相同地域的CentOS系统的ECS实例，本文演示系统为CentOS 7.6 64位系统。创建过程可参考创建ECS实例。
2. 使用root用户登录ECS实例，安装Nginx：

```
root@test:~# yum install -y nginx
```

说明：

Nginx默认安装位置：

```
/usr/sbin/nginx         主程序
/etc/nginx              存放配置文件
/usr/share/nginx        存放静态文件
/var/log/nginx          存放日志
```

3. 打开Nginx配置文件：

```
root@test:~# vi /etc/nginx/nginx.conf
```

4. 在config文件中的http模块中，修改配置如下：

```
server {
listen 80 default_server;
listen [::]:80 default_server;
server_name 47.**.**.43; #对外提供反向代理服务的IP，即ECS实例的外网地址；
root /usr/share/nginx/html;


# Load configuration files for the default server block.
include /etc/nginx/default.d/*.conf;


location / {
proxy_pass https://bucketname.oss-cn-beijing-internal.aliyuncs.com;
 #填写Bucket的内网访问域名，如果ECS实例与Bucket不在同一个地域，需填写外网域名；
}
```

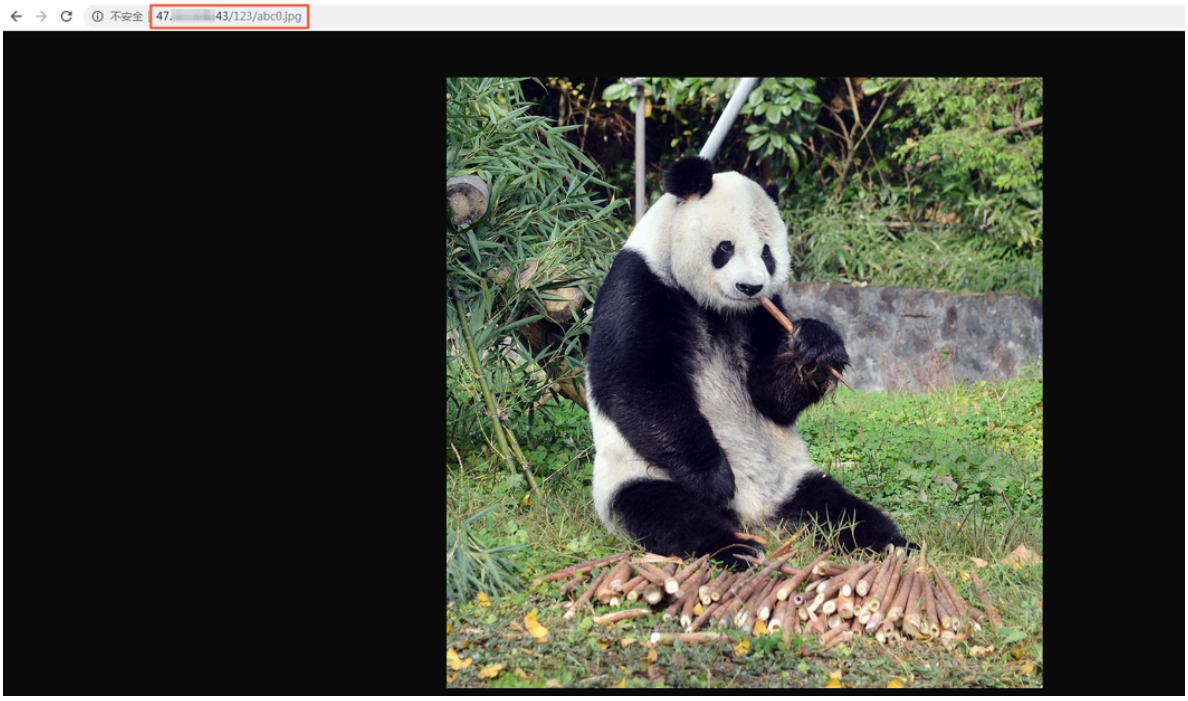
说明：

本文为演示环境，实际环境中，为了您的数据安全，建议配置https模块，配置方法可参考反向代理配置。

5. 进入Nginx主程序文件夹，启动Nginx：

```
root@test:~# cd /usr/sbin/
root@test:~# ./nginx
```

6. 测试使用ECS外网地址加文件访问路径访问OSS资源。



更多参考

#unique_165

15 Terraform

15.1 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 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

15.2 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