ALIBABA CLOUD

阿里云

对象存储 OSS

最佳实践

文档版本：20220708

阿里云

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
危险	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。	危险 重置操作将丢失用户配置数据。
警告	该类警示信息可能会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，恢复业务时间约十分钟。
注意	用于警示信息、补充说明等，是用户必须了解的内容。	注意 权重设置为0，该服务器不会再接受新请求。
说明	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。	单击**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。	在**结果确认**页面，单击**确定**。
Courier字体	命令或代码。	执行 `cd /d C:/window` 命令，进入Windows系统文件夹。
斜体	表示参数、变量。	`bae log list --instanceid` *Instance_ID*
[] 或者 [a|b]	表示可选项，至多选择一个。	`ipconfig [-all|-t]`
{} 或者 {a|b}	表示必选项，至多选择一个。	`switch {active|stand}`

目录

1.简介 08
2.网站与移动应用 09
2.1. Web端上传数据至OSS 09
2.1.1. Web端上传介绍 09
2.1.2. 小程序直传实践 10
2.1.2.1. 微信小程序直传实践 10
2.1.2.2. 支付宝小程序直传实践 17
2.1.3. Web端PostObject直传实践简介 22
2.1.4. JavaScript客户端签名直传 23
2.1.5. 服务端签名后直传 28
2.1.6. 服务端签名直传并设置上传回调 30
2.1.6.1. 概述 30
2.1.6.2. Python 36
2.1.6.3. Java 42
2.1.6.4. Go 50
2.1.6.5. Node.js 54
2.1.6.6. .NET 59
2.1.6.7. PHP 65
2.1.6.8. Ruby 69
2.2. 移动应用端直传实践 74
2.2.1. 快速搭建移动应用直传服务 74
2.2.2. 快速搭建移动应用上传回调服务 79
2.3. 使用ECS实例反向代理OSS 83
2.3.1. 基于Ubuntu的ECS实例实现OSS反向代理 83
2.3.2. 基于CentOS的ECS实例实现OSS反向代理 86
2.3.3. 基于Windows的ECS实例实现OSS反向代理 88

2.4. 通过crc64校验数据传输的完整性 91
3.数据湖 94
3.1. 阿里云生态 94
3.1.1. 通过SLS完成日志数据入湖OSS 94
3.1.2. 在EMR Hive或Spark中访问OSS-HDFS 98
3.1.3. 通过EMR集群配置Ranger鉴权方案 101
3.1.4. 使用JindoFuse访问OSS-HDFS服务 104
3.1.5. 基于OSS+MaxCompute构建数据仓库 110
3.1.6. 使用OSS中的数据作为机器学习的训练样本 114
3.1.7. DataLakeAnalytics+OSS：基于OSS的Severless的交互式查询分.. 121
3.2. 开源生态 126
3.2.1. 使用自建Hadoop访问全托管OSS-HDFS服务 126
3.2.2. 通过HDP 2.6 Hadoop读取和写入OSS数据 133
3.2.3. Flink使用JindoSDK处理OSS-HDFS服务的数据 138
3.2.4. HBase使用OSS-HDFS服务作为底层存储 140
3.2.5. Hive使用JindoSDK处理OSS-HDFS服务中的数据 141
3.2.6. Impala使用JindoSDK查询OSS-HDFS服务中的数据 144
3.2.7. Spark使用JindoSDK处理OSS-HDFS服务中的数据 147
3.2.8. Presto使用JindoSDK查询OSS-HDFS服务中的数据 150
3.2.9. Sqoop使用Kite SDK读写OSS-HDFS服务的数据 152
4.内容分发与数据处理 154
4.1. 音视频 154
4.1.1. 短视频 154
4.1.2. 音视频转码 156
4.2. 基于OSS构建HLS流 159
4.3. 使用CDN加速OSS访问 163
4.4. Java SDK的LiveChannel常见操作 167
4.5. Python SDK 的 LiveChannel 常见操作 174

5.数据分析 180
5.1. 使用 Java SDK 的 SelectObject 查询 CSV 和 JSON 文件 180
5.2. 使用Python SDK的SelectObject查询CSV和JSON文件 184
6.数据备份和容灾 188
6.1. 备份存储空间 188
6.2. 数据库备份到OSS 188
7.成本管理 193
7.1. 使用生命周期管理文件版本 193
8.数据迁移 197
8.1. OSS之间数据迁移 197
8.1.1. 概述 197
8.1.2. 使用数据复制功能迁移同账号下的OSS数据 197
8.1.3. 使用在线迁移服务跨账号迁移OSS数据 198
8.2. 第三方数据源迁移到 OSS 202
8.3. 从AWS S3上的应用无缝切换至OSS 202
8.4. 使用ossimport迁移数据 203
8.5. 从HDFS迁移数据到OSS 205
8.6. 从HDFS迁移数据到OSS-HDFS 213
8.7. 在OSS-HDFS服务不同Bucket之间迁移数据 215
8.8. 将半托管JindoFS集群迁移到OSS-HDFS服务 219
9.OSS安全 221
9.1. 降低因账号密码泄露带来的未授权访问风险 221
9.2. 降低因恶意访问流量导致大额资金损失的风险 223
9.3. 降低因操作失误等原因导致数据丢失的风险 226
9.4. 敏感数据安全防护方案 228
10.IaC自动化 231
10.1. Terraform 231
10.1.1. Terraform简介 231

10.1.2. 使用Terraform管理OSS 231
10.2. 通过OSS和ROS部署应用 236
10.2.1. 通过OSS和ROS创建Nginx 236
10.2.2. 通过OSS和ROS创建Sharepoint 2013 243
11.其他 254
11.1. 基于Jenkins+OOS+OSS自动化构建和部署WordPress 254
11.2. 使用函数计算打包下载OSS文件 254
11.3. 使用日志服务告警为您的OSS保驾护航 256
11.4. 通过云监控服务实时监控OSS流控信息 258
11.5. OSS性能与扩展性最佳实践 262

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
- 使用ossimport迁移数据

数据备份和容灾

- 备份存储空间
- 数据库备份到OSS

数据直传OSS

- Web端直传实践
- 移动端直传实践

数据处理与分析

- 基于OSS+MaxCompute构建数据仓库
- 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HDP 2.6 Hadoop读取和写入OSS数据

音视频转码

- 短视频
- 音视频转码

性能与扩展性

- OSS性能与扩展性最佳实践
- 使用CDN加速OSS访问

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.网站与移动应用

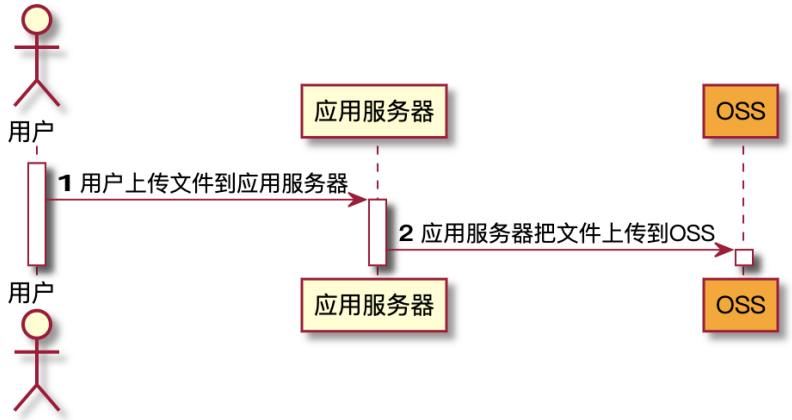
2.1. Web端上传数据至OSS

2.1.1. Web端上传介绍

本文介绍如何通过Web端直传文件（Object）到OSS。

背景信息

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，以上方法存在以下缺点：

- 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：如果后续用户数量逐渐增加，则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上行流量是免费的，如果数据直传到OSS，将节省多台应用服务器的费用。

技术方案

目前通过Web端将文件上传到OSS，有以下三种方案：

- 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请参见概述。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。

- 使用表单上传方式将文件上传到OSS

  利用OSS提供的PostObject接口，通过表单上传的方式将文件上传到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。

- 通过小程序上传文件到OSS

  通过小程序，如微信小程序和支付宝小程序，利用OSS提供的PostObject接口来实现表单上传。操作方式请参见微信小程序直传实践和支付宝小程序直传实践。

2.1.2. 小程序直传实践

2.1.2.1. 微信小程序直传实践

本文介绍如何在微信小程序环境下将文件上传到OSS。

背景信息

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，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PostObject。

步骤1：配置Bucket跨域访问

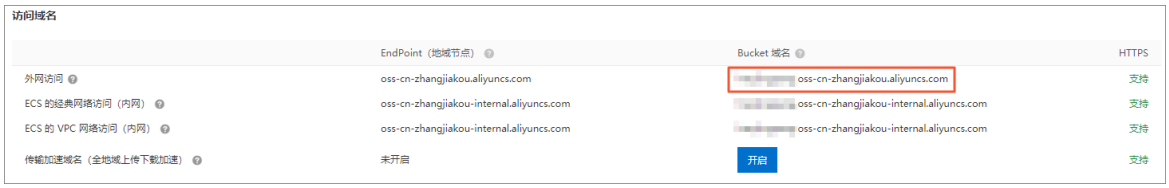
客户端进行表单直传到OSS时，会从浏览器向OSS发送带有Origin的请求消息。OSS对带有Origin头的请求消息会进行跨域规则（CORS）的验证。因此需要为Bucket设置跨域规则以支持Post方法。

1. 登录OSS管理控制台。
2. 单击**Bucket列表**，然后单击目标Bucket名称。
3. 在左侧导航栏，选择**权限管理 > 跨域设置**，然后在**跨域设置**区域，单击**设置**。
4. 单击**创建规则**，配置如下图所示。

> **说明** 为了您的数据安全，实际使用时，来源建议填写实际允许访问的域名。更多配置信息请参见设置跨域访问。

步骤2：微信小程序配置域名白名单

1. 登录OSS管理控制台。
2. 单击**Bucket列表**，然后单击目标Bucket名称。
3. 在存储空间**概览**页的**访问域名**区域查看**Bucket域名**。



4. 登录微信小程序平台，将上传和下载的合法域名填写为Bucket的外网访问域名。

> **说明** 实际业务中，建议您将OSS提供的外网域名和您自己的域名进行绑定。配置步骤请参见绑定自定义域名。

步骤3：获取签名

为了您的数据安全，建议使用签名方式上传文件。OSS提供以下两种签名方式的代码：

- 服务端签名

  使用服务端签名时，您需要先搭建一个签名服务，之后由客户端调用签名服务生成签名。

  - 服务端签名源码

    uploadOssHelper.js代码如下：

```
const crypto = require("crypto-js");
class MpUploadOssHelper {
  constructor(options) {
    this.accessKeyId = options.accessKeyId;
    this.accessKeySecret = options.accessKeySecret;
    // 限制参数的生效时间，单位为小时，默认值为1。
    this.timeout = options.timeout || 1;
    // 限制上传文件的大小，单位为MB，默认值为10。
    this.maxSize = options.maxSize || 10;
  }
  createUploadParams() {
    const policy = this.getPolicyBase64();
    const signature = this.signature(policy);
    return {
      OSSAccessKeyId: this.accessKeyId,
      policy: policy,
      signature: signature,
    };
  }
  getPolicyBase64() {
    let date = new Date();
    // 设置policy过期时间。
    date.setHours(date.getHours() + this.timeout);
    let srcT = date.toISOString();
    const policyText = {
      expiration: srcT,
      conditions: [
        // 限制上传文件大小。
        ["content-length-range", 0, this.maxSize * 1024 * 1024],
      ],
    };
    const buffer = Buffer.from(JSON.stringify(policyText));
    return buffer.toString("base64");
  }
  signature(policy) {
    return crypto.enc.Base64.stringify(
      crypto.HmacSHA1(policy, this.accessKeySecret)
    );
  }
}
module.exports = MpUploadOssHelper;
```

- 服务端接口示例

  以Express为例，接口代码如下：

```
const express = require('express');
const app = express();
const MpUploadOssHelper = require("./uploadOssHelper.js");
app.get('/getPostObjectParams', (req, res) => {
  const mpHelper = new MpUploadOssHelper({
    accessKeyId: '<Your AccessKeyId>',
    accessKeySecret: '<Your AccessKeySecret>',
    // 限制参数的生效时间，单位为小时，默认值为1。
    timeout: 1,
    // 限制上传文件大小，单位为MB，默认值为10。
    maxSize: 10,
  });
  // 生成参数。
  const params = mpHelper.createUploadParams();
  res.json(params);
})
```

  相关参数如下：

  - accessKeyId： 填写您的AccessKey ID。获取方式请参见获取AccessKey。
  - accessKeySecret： 填写您的AccessKey Secret。获取方式请参见获取AccessKey。

- 客户端签名

  使用客户端签名时，您需要先在服务端搭建一个STS服务，之后由客户端获取STS临时授权账号并生成签名。

  - 服务端搭建STS服务

```
const STS = require('ali-oss').STS;
const express = require('express');
const app = express();
const stsClient = new STS({
  accessKeyId: '<Your AccessKeyId>',
  accessKeySecret: '<Your AccessKeySecret>',
  bucket: '<Your bucket name>'
});
async function getToken() {
    const STS_ROLE = '<STS_ROLE>'  // 指定角色的ARN。格式为acs:ram::$accountID:role/$role
Name。
    const STSpolicy = {
      Statement: [
        {
          Action: ['oss:*'],
          Effect: 'Allow',
          Resource: ['acs:oss:*:*:*']
        }
      ],
      Version: '1'
    };
    const result = await stsClient.assumeRole(
      STS_ROLE,
      STSpolicy,
      3600 // STS过期时间，单位：秒。
    );
    const { credentials } = result;
    return credentials;
}
app.get('/getToken', async (req, res) => {
  // 获取STS。
  const credentials = await getToken()
  console.log(credentials.AccessKeyId)
  console.log(credentials.AccessKeySecret)
  console.log(credentials.SecurityToken)
  res.json(credentials);
})
```

相关参数如下：

- accessKeyId： 填写您的AccessKey ID。获取方式请参见获取AccessKey。
- accessKeySecret： 填写您的AccessKey Secret。获取方式请参见获取AccessKey。
- bucket：填写您的OSS存储空间名称。
- STS_ROLE：填写用于授权的RAM角色的ARN值。详情请参见查看RAM角色基本信息。
- STSpolicy：STSpolicy包含版本号（Version）和授权语句（Statement）。每条授权语句又包含授权效力（Effect）、操作（Action）、资源（Resource）。有关policy中各元素的定义及用法，请参见RAM Policy概述。

- 客户端获取STS临时账号并生成签名

```
import crypto from 'crypto-js';
import { Base64 } from 'js-base64';
// 计算签名。
function computeSignature(accessKeySecret, canonicalString) {
  return crypto.enc.Base64.stringify(crypto.HmacSHA1(canonicalString, accessKeySecret))
;
}
const date = new Date();
date.setHours(date.getHours() + 1);
const policyText = {
  expiration: date.toISOString(), // 设置policy过期时间。
  conditions: [
    // 限制上传大小。
    ["content-length-range", 0, 1024 * 1024 * 1024],
  ],
};
async function getFormDataParams() {
  const credentials = await axios.get('/getToken')
  const policy = Base64.encode(JSON.stringify(policyText)) // policy必须为base64的string
。
  const signature = computeSignature(credentials.AccessKeySecret, policy)
  const formData = {
    OSSAccessKeyId: credentials.AccessKeyId,
    signature,
    policy,
    'x-oss-security-token': credentials.SecurityToken
  }
  return formData
}
```

步骤4：使用微信小程序上传

使用uploadFile接口上传文件，示例代码如下：

```
const host = '<host>';
const signature = '<signatureString>';
const ossAccessKeyId = '<accessKey>';
const policy = '<policyBase64Str>';
const key = '<object name>';
const securityToken = '<x-oss-security-token>';
const filePath = '<filePath>'; // 待上传文件的文件路径。
wx.uploadFile({
  url: host, // 开发者服务器的URL。
  filePath: filePath,
  name: 'file', // 必须填file。
  formData: {
    key,
    policy,
    OSSAccessKeyId: ossAccessKeyId,
    signature,
    // 'x-oss-security-token': securityToken // 使用STS签名时必传。
  },
  success: (res) => {
    if (res.statusCode === 204) {
      console.log('上传成功');
    }
  },
  fail: err => {
    console.log(err);
  }
});
```

相关参数如下：

- host：填写存储空间的访问域名，例如 `https://test.oss-cn-zhangjiakou.aliyuncs.com` 。若您的存储空间已绑定自定义域名，建议填写您的自定义域名。
- signature：填写步骤3中获取到的signature信息。
- ossAccessKeyId：填写您的AccessKey ID，若是通过STS获取的临时用户，则填写临时用户的AccessKey ID。
- policy：填写步骤3中获取到的policy信息。
- key：设置文件上传至OSS后的文件路径。例如需要将myphoto.jpg上传至test文件夹下，此处填写 `test/myphoto.jpg` 。
- securityToken：若使用STS认证，此项填写步骤3中，使用客户端签名获取到的SecurityToken。
- filePath：填写待上传文件的文件路径，例如： `D:\example.txt` 。

常见问题

- iOS系统使用微信小程序直传后，返回的结果中，为什么ETag参数名称会变为Etag?

  问题原因：使用微信小程序直传后，OSS会返回标准的ETag。但是iOS系统的微信小程序会将返回的ETag转成了Etag，导致客户端调用ETag参数的业务出现问题。

  解决方案：若您的业务涉及多个系统的微信小程序直传场景，建议您将Header中的所有参数名都转成小写进行兼容。

- 上传成功后，如何获取文件URL?

具体操作，请参见上传Object后如何获取访问URL?

- PostObject请求时，报错 `The bucket POST must contain the specified 'key'. If it is specified, please check the order of the fields.`

  问题原因：没有指定key表单域。

  解决方案：PostObject请求时必须指定key表单域。详情请参见PostObject。

2.1.2.2. 支付宝小程序直传实践

本文介绍如何在支付宝小程序环境下将文件上传到OSS。

背景信息

小程序是当下比较流行的移动应用，例如微信小程序、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，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PostObject。

步骤1：配置Bucket跨域访问

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。

1. 登录OSS管理控制台。
2.
3. 在左侧导航栏，选择**权限管理 > 跨域设置**，然后在**跨域设置**区域，单击**设置**。
4. 单击**创建规则**，配置如下图所示。

> **说明** 为了您的数据安全，实际使用时，来源建议填写实际允许访问的域名。更多配置信息请参见设置跨域访问。

步骤2：获取签名

为了您的数据安全，建议使用签名方式上传文件。OSS提供以下两种签名方式的代码：

- 服务端签名

  使用服务端签名时，您需要先搭建一个签名服务，之后由客户端调用签名服务生成签名。

  - 服务端签名源码

    uploadOssHelper.js代码如下：

```
const crypto = require("crypto-js");
class MpUploadOssHelper {
  constructor(options) {
    this.accessKeyId = options.accessKeyId;
    this.accessKeySecret = options.accessKeySecret;
    this.timeOut = options.timeout || 1; // 限制参数的生效时间(单位：小时)。
    this.maxSize = options.maxSize || 10; // 限制上传文件大小(单位：Mb)。
  }
  createUploadParams() {
    const policy = this.getPolicyBase64();
    const signature = this.signature(policy);
    return {
      OSSAccessKeyId: this.accessKeyId,
      policy: policy,
      signature: signature,
    };
  }
  getPolicyBase64() {
    let date = new Date();
    // 设置Policy过期时间。
    date.setHours(date.getHours() + this.timeOut);
    let srcT = date.toISOString();
    const policyText = {
      expiration: srcT,
      conditions: [
        // 限制上传文件大小。
        ["content-length-range", 0, this.maxSize * 1024 * 1024],
      ],
    };
    const buffer = new Buffer(JSON.stringify(policyText));
    return buffer.toString("base64");
  }
  signature(policy) {
    return crypto.enc.Base64.stringify(
      crypto.HmacSHA1(policy, this.accessKeySecret)
    );
  }
}
module.exports = MpUploadOssHelper;
```

- 服务端接口示例

  以Express为例，接口代码如下：

```
const express = require('express');
const app = express();
const MpUploadOssHelper = require("./uploadOssHelper.js");
app.get('/getPostObjectParams', (req, res) => {
  const mpHelper = new MpUploadOssHelper({
    // 填写AccessKey ID。
    accessKeyId: '<Your AccessKeyId>',
    // 填写AccessKey Secret。
    accessKeySecret: '<Your AccessKeySecret>',
    // 限制参数的生效时间(单位：小时)。
    timeout: 1,
    // 限制上传文件大小(单位：Mb)。
    maxSize: 10,
  });
  // 生成参数。
  const params = mpHelper.createUploadParams();
  res.json(params);
})
```

- 客户端签名

  使用客户端签名时，您需要先在服务端搭建一个STS服务，之后由客户端获取STS临时授权账号并生成签名。

  - 服务端搭建STS服务

```
const OSS = require('ali-oss')
const STS = OSS.STS
const express = require('express');
const app = express();
const stsClient = new STS({
  // 填写您的AccessKey ID。
  accessKeyId: '<Your AccessKeyId>',
  // 填写您的AccessKey Secret。
  accessKeySecret: '<Your AccessKeySecret>',
  // 填写Bucket名称。
  bucket: '<Your bucket name>'
});
async function getToken() {
    // 指定角色的ARN。格式为acs:ram::$accountID:role/$roleName。
    const STS_ROLE = '<STS_ROLE>'
    const STSpolicy = {
      Statement: [
        {
          Action: ['oss:*'],
          Effect: 'Allow',
          Resource: ['acs:oss:*:*:*']
        }
      ],
      Version: '1'
    };
    const result = await stsClient.assumeRole(
      // 填写用于授权的RAM角色的ARN值。详情请参见查看RAM角色基本信息。
      STS_ROLE,
      STSpolicy,
      // 指定STS过期时间，单位为秒。
      3600
    );
    const { credentials } = result;
    return credentials;
}
app.get('/getToken', async (req, res) => {
  // 获取STS。
  const credentials = await getToken()
  console.log(credentials.AccessKeyId)
  console.log(credentials.AccessKeySecret)
  console.log(credentials.SecurityToken)
  res.json(credentials);
})
```

- 客户端获取STS临时账号并生成签名

```
import crypto from 'crypto-js';
import { Base64 } from 'js-base64';
// 计算签名。
function computeSignature(accessKeySecret, canonicalString) {
  return crypto.enc.Base64.stringify(crypto.HmacSHA1(canonicalString, accessKeySecret))
;
}
const date = new Date();
date.setHours(date.getHours() + 1);
const policyText = {
  // 设置policy过期时间。
  expiration: date.toISOString(),
  conditions: [
  // 限制上传的文件大小。
    ["content-length-range", 0, 1024 * 1024 * 1024],
  ],
};
async function getFormDataParams() {
  const credentials = await axios.get('/getToken')
  const policy = Base64.encode(JSON.stringify(policyText)) // policy必须为base64的string
。
  const signature = computeSignature(credentials.AccessKeySecret, policy)
  const formData = {
    OSSAccessKeyId: credentials.AccessKeyId,
    signature,
    policy,
    'x-oss-security-token': credentials.SecurityToken
  }
  return formData
}
```

步骤3：使用支付宝小程序上传

支付宝小程序上传文件参考代码如下：

```
const host = '<host>';
const signature = '<signatureString>';
const ossAccessKeyId = '<accessKey>';
const policy = '<policyBase64Str>';
const securityToken = '<x-oss-security-token>';
const key = '<object name>'
my.chooseImage({
  chooseImage: 1,
  success: res => {
    const path = res.apFilePaths[0];
    my.uploadFile({
        url: host,
        fileType: 'image',
        fileName: 'file',
        filePath: path,
        formData: {
            key,
            policy,
            OSSAccessKeyId: ossAccessKeyId,
            signature,
            success_action_status: '200',
            // 使用STS签名时传入securityToken。
            // 'x-oss-security-token': securityToken
        },
        success: (res) => {
          // 将上传成功状态码设置为200，默认状态码为204。
          if (res.statusCode === 200) {
            console.log('上传成功');
          }
          my.alert({ content: 'success info: ' + res.data });
        },
        fail: err => {
          console.log(err);
        }
    });
  }
});
```

相关参数如下：

- host：填写存储空间的访问域名，例如https://test.oss-cn-zhangjiakou.aliyuncs.com。若您的存储空间已绑定自定义域名，建议填写您的自定义域名。
- signature：填写步骤2中获取到的signature信息。
- ossAccessKeyId：填写您的AccessKey ID，若是通过STS获取的临时用户，则填写临时用户的AccessKey ID。
- policy：填写步骤2中获取到的policy信息。
- key：设置文件上传至OSS后的文件路径。例如需要将myphoto.jpg上传至test文件夹下，此处填写test/myphoto.jpg。
- securityToken：若使用STS认证，此项填写客户端签名时获取到的SecurityToken。

2.1.3. 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。

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传至OSS的方式。

数据直传至OSS是利用OSS的 PostObject接口，使用表单上传方式上传文件至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。

2.1.4. 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。

注意事项

- 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AccesssKey ID和AccessKey Secret写在代码里面有泄露的风险，强烈建议使用服务端签名后直传或者STS临时授权访问OSS。
- 本文档提供的应用服务器代码支持html5、flash、silverlight、html4等协议，请保证您的浏览器支持以上协议。如果提示“你的浏览器不支持flash,Silverlight或者HTML5！”，请升级您的浏览器版本。

步骤1：下载浏览器客户端代码

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。

1. 下载浏览器客户端代码。
2. 将下载的文件解压。

> **说明** 示例中使用的前端插件是Plupload，您可以根据实际需求更换插件。

步骤2：修改配置文件

打开upload.js文件，修改访问配置。

```
// 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维，请登录RAM控制台创建RAM用户。
accessid= '<yourAccessKeyId>';
accesskey= '<yourAccessKeySecret>';
host= '<yourHost>';
.....
new_multipart_params = {
        ....
        'OSSAccessKeyId': accessid,
        ....
    };
//如果是STS方式临时授权访问OSS，请参见如下代码。
accessid= 'accessid';
accesskey= 'accesskey';
host = 'http://post-test.oss-cn-hangzhou.aliyuncs.com';
STS_ROLE = ''; // eg: acs:ram::1069test7698:role/all
.....
new_multipart_params = {
        ....
        'OSSAccessKeyId': STS.accessID,
        'x-oss-security-token':STSToken,
        ....
    };
//===========
host = 'http://post-test.oss-cn-hangzhou.aliyuncs.com';
```

- accessid：您的RAM用户AccessKeyId。
- accesskey：您的RAM用户AcessKeySecret。
- STS_ROLE：表示指定角色的ARN，详情请参见AssumeRole中请求参数RoleArn的说明。
- STSToken：您的STS token。使用STS方式验证时，您需要通过STS API获取STS AccessKey ID、STS AcessKey Secret、SecurityToken，详情请参见使用STS临时访问凭证访问OSS。如果您的AccessKey ID和AccessKey Secret为阿里云账号或拥有永久权限的RAM用户AK，此项可不填。
- 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OSS访问域名使用规则。

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。

1. 登录OSS管理控制台。
2.
3. 在左侧导航栏，选择**权限管理 > 跨域设置**，然后在**跨域设置**区域，单击**设置**。
4. 单击**创建规则**，配置如下图所示。

> **说明** 为了您的数据安全，实际使用时，来源建议填写实际允许访问的域名。更多配置信息请参见设置跨域访问。

步骤4：体验JavaScript客户端签名直传

1. 在解压的客户端代码文件夹中打开*index.html*文件。
2. 单击**选择文件**，之后选择一个或多个文件，并选择上传后的文件名命名规则及上传后文件所在目录。

3. 单击**开始上传**，并等待上传完成。
4. 上传成功后，您可以登录OSS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：

```
function set_upload_param(up, filename, ret)
{
  g_object_name = g_dirname;
  if (filename != '') {
      suffix = get_suffix(filename)
      calculate_object_name(filename)
  }
  new_multipart_params = {
      'key' : g_object_name,
      'policy': policyBase64,
      'OSSAccessKeyId': accessid,
      'success_action_status' : '200', //如果不设置success_action_status为200，文件上传成功后则返回204状态码。
      'signature': signature,
  };
  up.setOption({
      'url': host,
      'multipart_params': new_multipart_params
  });
  up.start();
}
  ....
```

上述代码中，`'key': g_object_name` 表示上传后的文件路径。如果您希望上传后保持原来的文件名，请将该字段改为 `'key': '${filename}'` 。

如果您希望上传到特定目录，例如abc下，且文件名不变，请修改代码如下：

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

- 设置成随机文件名

  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": "setDurationSeconds", // 为Policy指定合理的有效时长，格式为UTC时间。Policy失效后，则无法通过此Policy上传文件。
    "conditions": [
    ["content-length-range", 0, 1048576000] // 设置上传文件的大小限制。如果超过此限制，文件上传到OSS会报错。
    ]
}
```

常见问题

- 如何限制上传文件的格式？

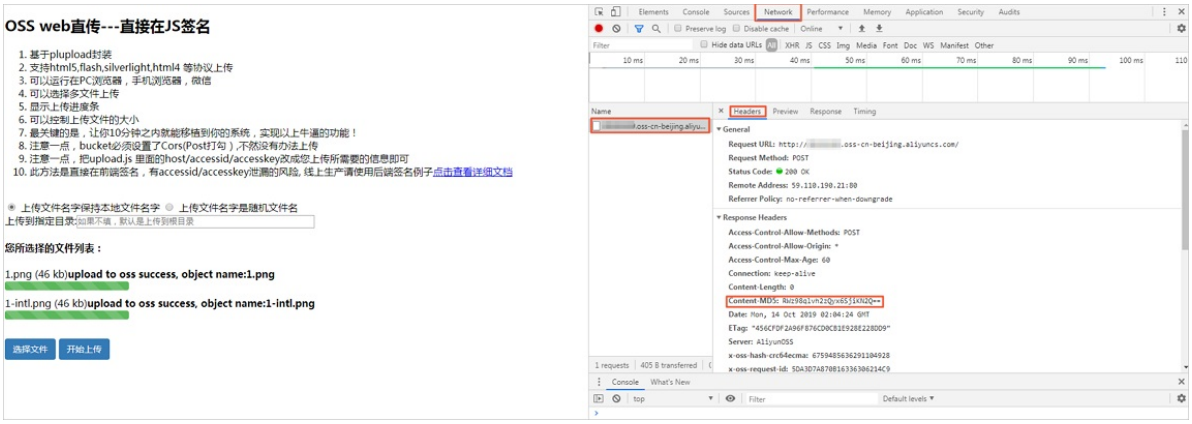
  您可以利用Plupload的filters属性设置上传的过滤条件，例如设置上传的图片类型、上传的文件的大小等。详细代码请参见设置上传过滤条件。

- 上传后如何获取文件URL？

  您可以根据Bucket访问域名及文件访问路径获取文件的URL，详情请参见上传Object后如何获取访问URL？

- 如何获取已上传文件的MD5值？

  您需要按F12打开开发者模式，之后开始上传文件。文件上传成功后在响应头中可以查看文件的MD5，如下图所示。

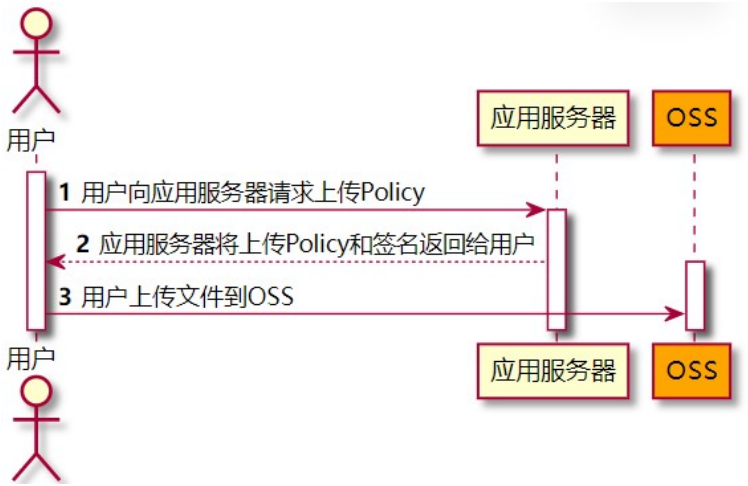


如果您还有更多问题，请提交工单寻求解答。

2.1.5. 服务端签名后直传

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，然后通过表单直传数据到OSS。由于服务端签名直传无需将AccessKey暴露在前端页面，相比JavaScript客户端签名直传具有更高的安全性。

流程和源码解析



1. 用户向应用服务器请求上传Policy和回调。

   请将客户端源码中的 `upload.js` 文件的如下代码片段的变量 `serverUrl` 的值设置为应用服务器的URL。

   ```
   // 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'
   ```

   设置完成后，客户端会向该 `serverUrl` 发送Get请求来获取需要的信息。客户端源码下载地址，请参见aliyun-oss-appserver-js-master.zip。

   本场景为服务端签名后直传，不涉及上传回调。因此，您需要注释客户端源码的 `upload.js` 文件内的 `'callback' : callbackbody` 字段，以关闭上传回调功能。

   ```
   {
     'key' : key + '${filename}',
     'policy': policyBase64,
     'OSSAccessKeyId': accessid,
      // 设置服务端返回200状态码，默认返回204。
     'success_action_status' : '200',
     'callback' : callbackbody,
     'signature': signature,
   }
   ```

2. 应用服务器返回上传Policy和签名给用户。

   应用服务器侧的签名直传服务会处理客户端发送的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。

   以下是签名直传服务返回给客户端消息Body内容的示例：

```
{
"accessid":"LTAI5tBDFVar1hoq****",
"host":"http://post-test.oss-cn-hangzhou.aliyuncs.com",
"policy":"eyJleHBpcmF0aW9uIjoiMjAxNS0xMS0wNVQyMDoyMzoyM1oiLCJjxb25kaXRpb25zIjpbWyJjcb250ZW50LWxlbmd0aC1yYW5nZSIsMCwxMDQ4NTc2MDAwXSxbInN0YXJ0cy13aXRoIiwiJGtleSIsInVzZXItZGlyXC8i****",
"signature":"VsxOcOudx******z93CLaXPz+4s=",
"expire":1446727949,
"dir":"user-dirs/"
}
```

Body中的各字段说明如下：

字段	描述
accessid	用户请求的AccessKey ID。
host	用户发送上传请求的域名。
policy	用户表单上传的策略（Policy），Policy为经过Base64编码过的字符串。详情请参见Post Policy。
signature	对Policy签名后的字符串。详情请参见Post Signature。
expire	由服务器端指定的Policy过期时间，格式为Unix时间戳（自UTC时间1970年01月01号开始的秒数）。
dir	限制上传的文件前缀。

3. 用户使用Post方法向OSS发送文件上传请求。

```
new_multipart_params = {
     // key表示上传到Bucket内的Object的完整路径，例如exampledir/exampleobject.txtObject，完整路径中不能包含Bucket名称。
     // filename表示待上传的本地文件名称。
     'key' : key + '${filename}',
     'policy': policyBase64,
     'OSSAccessKeyId': accessid,
     // 设置服务端返回状态码为200，不设置则默认返回状态码204。
     'success_action_status' : '200',
     'signature': signature,
 };
```

代码示例

服务端签名直传并设置上传回调的各语言版本示例如下：

- Java
- Python
- PHP
- Go
- Node.js
- Ruby

- .NET

更多参考

大多数情况下，应用服务器需要了解用户上传了哪些文件以及对应的文件名称等信息。如果上传的是图片，还希望获取图片大小等。此时，您可以通过上传回调方案实现该需求。更多信息，请参见服务器端签名直传并设置上传回调。

2.1.6. 服务端签名直传并设置上传回调

2.1.6.1. 概述

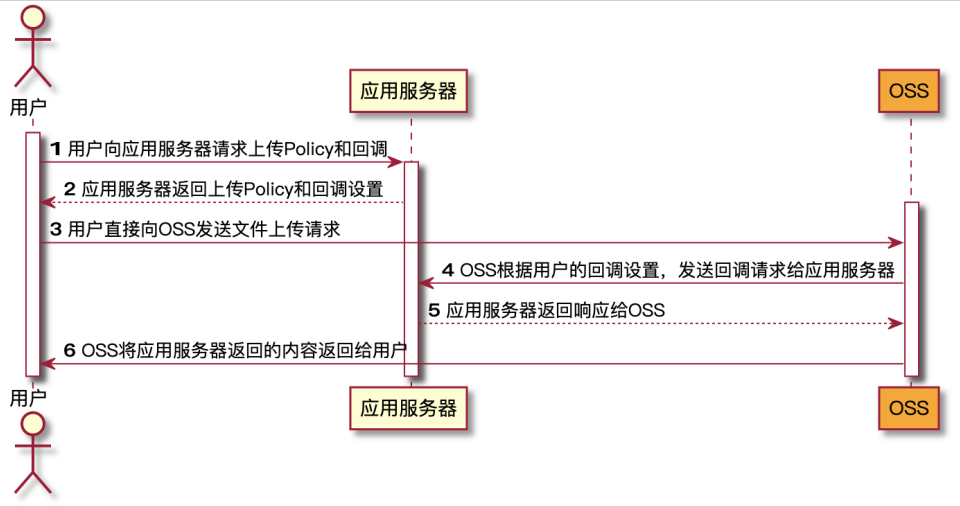
本文主要介绍如何基于Post Policy的使用规则在服务端通过各种语言代码完成签名，并且设置上传回调，然后通过表单直传数据到OSS。

背景介绍

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。

流程介绍

流程如下图所示：



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时，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
- Node.js
- .NET

- 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` 的值可以用来设置应用服务器的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。

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

   ```
   {
   "accessid":"LTAI5tAzivUnv4ZF1azP****",
   "host":"http://post-test.oss-cn-hangzhou.aliyuncs.com",
   "policy":"eyJleHBpcmF0aW9uIjoiMjAxNS0xMS0wNVQyMDoyMzoyM1oiLCJjxb25kaXRpb25zIjpbWyJjcb250ZW50LWxlbmd0aC1yYW5nZSIsMCwxMDQ4NTc2MDAwXSxbInN0YXJ0cy13aXRoIiwiJGtleSIsInVzZXItZGlyXC8i****",
   "signature":"I2u57FWjTKqX/AE6doIdyff1****",
   "expire":1446727949,
   "callback":"eyJjYWxsYmFja1VybCI6Imh0dHA6Ly9vc3MtZGVtby5hbGl5dW5jcy5jb206MjM0NTAiLAoiY2FsbGJhY2tCb2R5IjoiZmlsZW5hbWU9JHtvYmplY3R9JnNpemU9JHtzaXplfSZtaW1lVHlwZT0ke21pbWVUeXBlfSZoZWlnaHQ9JHtpbWFnZUluZm8uaGVpZ2h0fSZ3aWR0aD0ke2ltYWdlSW5mby53aWR0aH0iLAoiY2FsbGJhY2tCb2R5VHlwZSI6ImFwcGxpY2F0aW9uL3gtd3d3LWZvcm0tdXJsZW5jb2RlZCJ9",
   "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",
   "callbackBody":"filename=${object}&size=${size}&mimeType=${mimeType}&height=${imageInfo.height}&width=${imageInfo.width}",
   "callbackBodyType":"application/x-www-form-urlencoded"}
   ```

   > **说明** 以上仅为回调示例，您可以通过修改服务端代码自行设置回调。

参数	说明
callbackUrl	OSS向服务器发送的URL请求。
callbackHost	OSS发送该请求时，请求头部所带的Host头。
callbackBody	OSS发送给应用服务器的内容。如果是文件，可以是文件的名称、大小、类型等。如果是图片，可以是图片的高度、宽度等。

参数	说明
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,
     // 设置服务端返回状态码为200，不设置则默认返回状态码204。
     'success_action_status' : '200',
     '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。消息内容示例如下：

```
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******h3nFUDALgku9bhC+cWQsnxuCo******
tBUmnDI6klPofggA4g==\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: 137918634953****\r\n
    x-oss-pub-key-url: aHR0cDovL2dvc3NwdWJsaWMuYWxpY2RuLmNvbS9jYWxsYmFja19wdWJfa2V5X3Yx
LnaH****\r\n
    x-oss-request-id: 534B371674E88A4D8906****\r\n
    x-oss-requester: 137918634953****\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HTML、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。

以下为常见功能的代码示例：

- 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" }
        ],
        // 最大只能上传400KB的文件。
        max_file_size : '400kb',
        // 不允许选取重复文件。
        prevent_duplicates : true
    },
```

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

可以利用 `get_uploaded_object_name(file.name)` 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。

- 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，则重新获取签名，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，但必须包含accessid、policy、signature三个值。

```
{"accessid":"LTAI5tAzivUnv4ZF1azP****",
"host":"http://post-test.oss-cn-hangzhou.aliyuncs.com",
"policy":"eyJleHBpcmF0aW9uIjoiMjAxNS0xMS0wNVQyMDoyMzoyM1oiLCJjxb25kaXRpb25zIjpbWyJjcb250ZW50LWxlbmd0aC1yYW5nZSIsMCwxMDQ4NTc2MDAwXSxbInN0YXJ0cy13aXRoIiwiJGtleSIsInVzZXItZGlyXC8i****",
"signature":"I2u57FWjTKqX/AE6doIdyff1****",
"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，初始化上传时，每上传一个文件，获取一次签名。再次上传文件时，对当前时间与签名时间进行比较，并查看签名时间是否失效。如果签名已失效，则重新获取一次签名，如果签名未失效，则使用之前的签名。

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，用户也可以自行指定。增加starts-with的原因是，在众多场景下，一个应用对应一个Bucket。为了防止数据覆盖，用户上传到OSS的每个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，从而上传到其他用户的目录下。为解决该问题，可在应用服务器端指定用户上传文件的前缀。这样即便用户获取了Policy也没有办法上传到其他用户的目录，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` 的值可以用来设置应用服务器的URL，设置完成后，客户端会向该 `serverUrl` 发送Get请求来获取信息。

```
// 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'
```

常见问题

前端如何实现批量上传文件？

OSS没有开放批量上传接口，如果需要批量上传，您可以使用一个循环去上传所有文件，示例与上传单个文件一致。

2.1.6.2. 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**python --version**命令进行查看）。
- 确保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（Python版本）。

2. 本示例中以Ubuntu 16.04为例，将源码解压到*/home/aliyun/aliyun-oss-appserver-python*目录下。
3. 进入该目录，打开源码文件*appserver.py*，修改如下的代码片段：

```
# 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维，请登录RAM控制台创建RAM用户。
access_key_id = 'yourAccessKeyId'
access_key_secret = 'yourAccessKeySecret'
# 填写Host地址，格式为https://bucketname.endpoint。
host = 'https://examplebucket.oss-cn-hangzhou.aliyuncs.comm';
# 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之间的通信。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。
callback_url = "https://192.168.0.0:8888";
# 设置上传到OSS文件的前缀，可置空此项。置空后，文件将上传至Bucket的根目录下。
upload_dir = 'exampledir/'
```

步骤2：配置客户端

1. 下载客户端源码。
2. 解压客户端源码文件，本示例解压到*D:\aliyun\aliyun-oss-appserver-js*目录。
3. 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，请对应替换以下IP地址和Port端口信息。
serverUrl = 'http://192.0.2.0:8888'
```

4. 将 `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 = 'https://192.168.0.0:8888'` 。

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。

1. 登录OSS管理控制台。
2. 单击**Bucket列表**，然后单击目标Bucket名称。
3. 在左侧导航栏，选择**权限管理 > 跨域设置**，然后在**跨域设置**区域，单击**设置**。
4. 单击**创建规则**，配置如下图所示。

> **说明** 为了您的数据安全，实际使用时，**来源**建议填写实际允许访问的域名。更多配置信息请参见设置跨域访问。

步骤4：体验上传回调

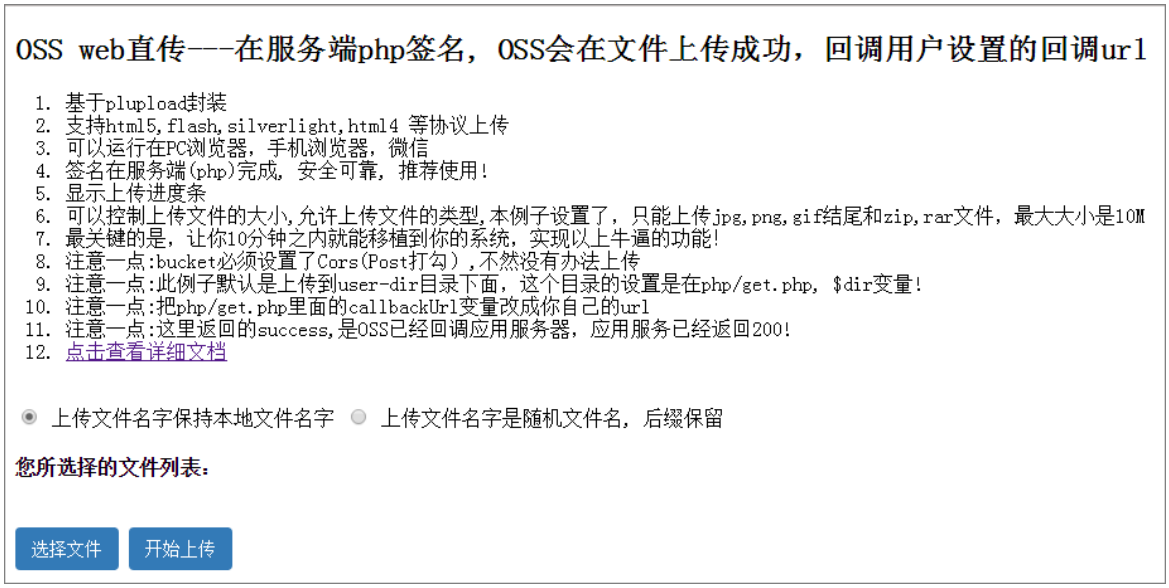
1. 启动应用服务器。

   在*/home/aliyun-oss-appserver-python*目录下，执行**python appserver.py 11.22.33.44 1234**命令启动应用服务器。

   > **说明** 请将IP地址和端口改成您配置的应用服务器的IP地址和端口。

2. 启动客户端。

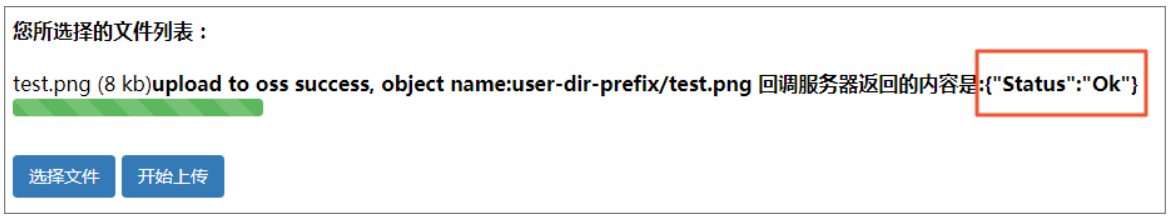
   在PC侧的客户端源码目录中，打开*index.html*文件。



> **注意** index.html文件不保证兼容IE 10以下版本浏览器，若使用IE 10以下版本浏览器出现问题时，您需要自行调试。

3. 上传文件。

   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以及上传回调服务，完整示例代码如下：

```
# -*- coding: UTF-8 -*-
import socket
import base64
import sys
import time
import datetime
import json
import hmac
from hashlib import sha1 as sha
import httpserver
# 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维，请登录RAM控制台创建RAM用户。
access_key_id = 'yourAccessKeyId'
access_key_secret = 'yourAccessKeySecret'
# 填写Host地址，格式为https://bucketname.endpoint。
host = 'https://examplebucket.oss-cn-hangzhou.aliyuncs.com';
# 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之间的通信。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。
callback_url = "https://192.168.0.0:8888";
```

```
# 设置上传到OSS文件的前缀，可置空此项。置空后，文件将上传至Bucket的根目录下。
upload_dir = 'exampledir/'
expire_time = 30
def get_iso_8601(expire):
    gmt = datetime.datetime.utcfromtimestamp(expire).isoformat()
    gmt += 'Z'
    return gmt
def get_token():
    now = int(time.time())
    expire_syncpoint = now + expire_time
    expire_syncpoint = 1612345678
    expire = get_iso_8601(expire_syncpoint)
    policy_dict = {}
    policy_dict['expiration'] = expire
    condition_array = []
    array_item = []
    array_item.append('starts-with');
    array_item.append('$key');
    array_item.append(upload_dir);
    condition_array.append(array_item)
    policy_dict['conditions'] = condition_array
    policy = json.dumps(policy_dict).strip()
    policy_encode = base64.b64encode(policy.encode())
    h = hmac.new(access_key_secret.encode(), policy_encode, sha)
    sign_result = base64.encodestring(h.digest()).strip()
    callback_dict = {}
    callback_dict['callbackUrl'] = callback_url;
    callback_dict['callbackBody'] = 'filename=${object}&size=${size}&mimeType=${mimeType}'
\
                                    '&height=${imageInfo.height}&width=${imageInfo.width}';
    callback_dict['callbackBodyType'] = 'application/x-www-form-urlencoded';
    callback_param = json.dumps(callback_dict).strip()
    base64_callback_body = base64.b64encode(callback_param.encode());
    token_dict = {}
    token_dict['accessid'] = access_key_id
    token_dict['host'] = host
    token_dict['policy'] = policy_encode.decode()
    token_dict['signature'] = sign_result.decode()
    token_dict['expire'] = expire_syncpoint
    token_dict['dir'] = upload_dir
    token_dict['callback'] = base64_callback_body.decode()
    result = json.dumps(token_dict)
    return result
def get_local_ip():
    """
    获取本机IPV4地址。
    :return: 成功获取，则返回本机IP地址。获取失败，则返回为空。
    """
    try:
        csocket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
        csocket.connect(('198.51.100.0', 80))
        (addr, port) = csocket.getsockname()
        csocket.close()
        return addr
```

```
    except socket.error:
        return ""
def do_POST(server):
    """
    启用POST调用处理逻辑。
    :param server: Web HTTP Server服务
    :return:
    """
    print("******************** do_POST ")
    # get public key
    pub_key_url = ''
    try:
        pub_key_url_base64 = server.headers['x-oss-pub-key-url']
        pub_key = httpserver.get_pub_key(pub_key_url_base64)
    except Exception as e:
        print(str(e))
        print('Get pub key failed! pub_key_url : ' + pub_key_url)
        server.send_response(400)
        server.end_headers()
        return
    # get authorization
    authorization_base64 = server.headers['authorization']
    # get callback body
    content_length = server.headers['content-length']
    callback_body = server.rfile.read(int(content_length))
    # compose authorization string
    auth_str = ''
    pos = server.path.find('?')
    if -1 == pos:
        auth_str = server.path + '\n' + callback_body.decode()
    else:
        auth_str = httpserver.get_http_request_unquote(server.path[0:pos]) + server.path[po
s:] + '\n' + callback_body
    result = httpserver.verrify(auth_str, authorization_base64, pub_key)
    if not result:
        print('Authorization verify failed!')
        print('Public key : %s' % (pub_key))
        print('Auth string : %s' % (auth_str))
        server.send_response(400)
        server.end_headers()
        return
    # response to OSS
    resp_body = '{"Status":"OK"}'
    server.send_response(200)
    server.send_header('Content-Type', 'application/json')
    server.send_header('Content-Length', str(len(resp_body)))
    server.end_headers()
    server.wfile.write(resp_body.encode())
def do_GET(server):
    """
    启用Get调用处理逻辑
    :param server: Web HTTP Server服务
    :return:
    """
```

```
    print("******************** do_GET ")
    token = get_token()
    server.send_response(200)
    server.send_header('Access-Control-Allow-Methods', 'POST')
    server.send_header('Access-Control-Allow-Origin', '*')
    server.send_header('Content-Type', 'text/html; charset=UTF-8')
    server.end_headers()
    server.wfile.write(token.encode())
if '__main__' == __name__:
    # 在服务器中，0.0.0.0指的是本机上的所有IPV4地址。
    # 如果一个主机有两个IP地址，例如192.0.2.0和192.0.2.254，并且该主机上的一个服务监听的地址是0.0.0.0，则通过这两个IP地址均能够访问该服务。
    # server_ip = get_local_ip() 如果您希望监听本机外网IPV4地址，则采用本行代码并注释掉下一行代码。
    server_ip = "0.0.0.0"
    server_port = 8080
    if len(sys.argv) == 2:
        server_port = int(sys.argv[1])
    if len(sys.argv) == 3:
        server_ip = sys.argv[1]
        server_port = int(sys.argv[2])
    print("App server is running on http://%s:%s " % (server_ip, server_port))
    server = httpserver.MyHTTPServer(server_ip, server_port)
    server.serve_forever()
```

关于上传回调的API接口说明，请参见Callback。

2.1.6.3. 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

1. 下载应用服务器源码（Java版本）。
2. 本示例中以 `Ubuntu 16.04` 为例，将下载的文件解压到*/home/aliyun/aliyun-oss-appserver-java*目录下。
3. 进入该目录，打开源码文件*CallbackServer.java*，然后结合实际情况修改源码文件。源码文件修改示例如下：

```
// 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维，请登录RAM控制台创建RAM用户。
String accessId = "yourAccessKeyId";
String accessKey = "yourAccessKeySecret";
// Endpoint以华东1（杭州）为例，其它Region请按实际情况填写。
String endpoint = "oss-cn-hangzhou.aliyuncs.com";
// 填写Bucket名称，例如examplebucket。
String bucket = "examplebucket";
// 填写Host地址，格式为https://bucketname.endpoint。
String host = "https://examplebucket.oss-cn-hangzhou.aliyuncs.com";
// 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之间的通信。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。
String callbackUrl = "https://192.168.0.0:8888";
// 设置上传到OSS文件的前缀，可置空此项。置空后，文件将上传至Bucket的根目录下。
String dir = "exampledir/";
```

步骤2：配置客户端

1. 下载客户端源码。
2. 将文件解压，本例中以解压到 `D:\aliyun\aliyun-oss-appserver-js` 目录为例。
3. 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，请对应替换以下IP地址和Port端口信息。
   serverUrl = 'http://192.0.2.0:8888'
   ```

4. 将 `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 = 'https://192.168.0.0:8888'` 。

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。

1. 登录OSS管理控制台。
2. 单击**Bucket列表**，然后单击目标Bucket名称。
3. 在左侧导航栏，选择**权限管理 > 跨域设置**，然后在**跨域设置**区域，单击**设置**。
4. 单击**创建规则**，配置如下图所示。

> **说明** 为了您的数据安全，请在实际使用时将**来源**配置为实际允许访问的域名。关于各配置项的更多信息，请参见设置跨域访问。

步骤4：体验上传回调

1. 启动应用服务器。

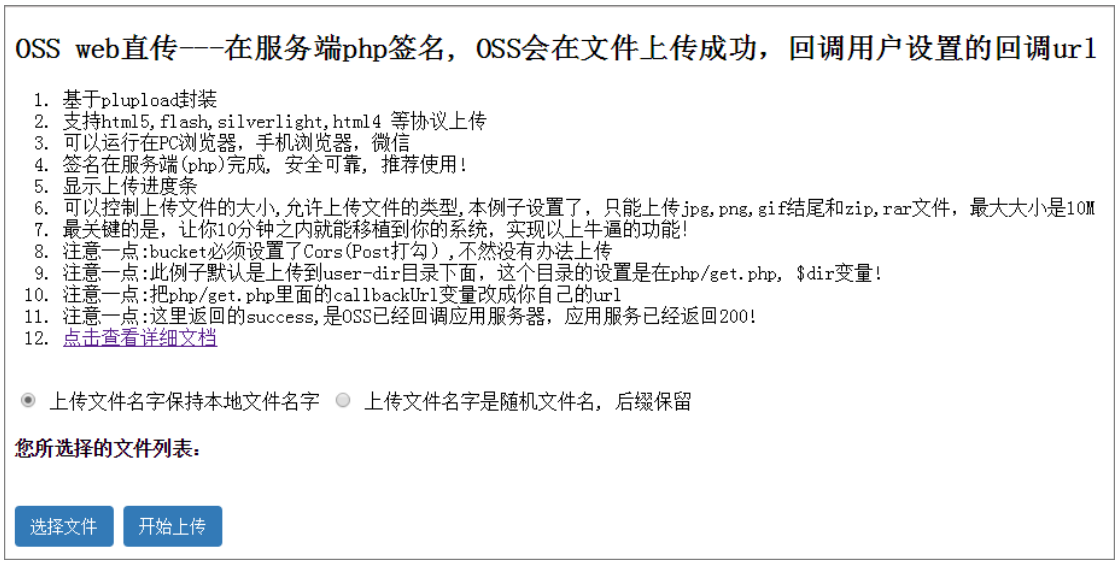
   在*/home/aliyun/aliyun-oss-appserver-java*目录下，执行**mvn package**命令编译打包，然后执行**java -jar target/appservermaven-1.0.0.jar 1234**命令启动应用服务器。

   > **说明** 请将IP和端口改成您配置的应用服务器的IP和端口。

   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。

2. 启动客户端。

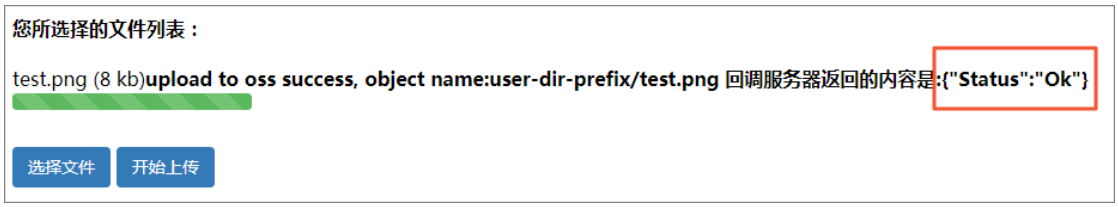
i. 在PC端的客户端源码目录中，打开*index.html*文件。



> **注意** index.html文件不保证兼容IE 10以下版本浏览器，若使用IE 10以下版本浏览器出现问题时，您需要自行调试。

ii. 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以及上传回调服务，完整示例代码如下：

```
package com.aliyun.oss.appservermaven;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.net.URI;
import java.security.KeyFactory;
import java.security.PublicKey;
import java.security.spec.X509EncodedKeySpec;
import java.sql.Date;
import java.util.LinkedHashMap;
import java.util.Map;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.apache.http.HttpResponse;
```

```
import org.apache.http.client.methods.HttpGet;
import org.apache.http.impl.client.DefaultHttpClient;
import com.aliyun.oss.OSSClient;
import com.aliyun.oss.common.utils.BinaryUtil;
import com.aliyun.oss.model.MatchMode;
import com.aliyun.oss.model.PolicyConditions;
//import org.junit.Assert;
import net.sf.json.JSONObject;
@SuppressWarnings("deprecation")
@WebServlet(asyncSupported = true)
public class CallbackServer extends HttpServlet {
    /**
     *
     */
    private static final long serialVersionUID = 5522372203700422672L;
    /**
     * Get请求
     */
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        // 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维，请登录RAM控制台创建RAM用户。
        String accessId = "yourAccessKeyId";
        String accessKey = "yourAccessKeySecret";
        // Endpoint以华东1（杭州）为例，其它Region请按实际情况填写。
        String endpoint = "oss-cn-hangzhou.aliyuncs.com";
        // 填写Bucket名称，例如examplebucket。
        String bucket = "examplebucket";
        // 填写Host地址，格式为https://bucketname.endpoint。
        String host = "https://examplebucket.oss-cn-hangzhou.aliyuncs.com";
        // 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之间的通信。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。
        String callbackUrl = "https://192.168.0.0:8888";
        // 设置上传到OSS文件的前缀，可置空此项。置空后，文件将上传至Bucket的根目录下。
        String dir = "exampledir/";
        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_CONTENT_LENGTH_RANGE, 0, 104
8576000);
            policyConds.addConditionItem(MatchMode.StartWith, PolicyConditions.COND_KEY, di
r);
            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=${mimeType}&height=${imageInf
o.height}&width=${imageInfo.width}");
            jasonCallback.put("callbackBodyType", "application/x-www-form-urlencoded");
            String base64CallbackBody = BinaryUtil.toBase64String(jasonCallback.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());
        }
    }
    /**
     * 获取public key
     *
     * @param url
     * @return
     */
    @SuppressWarnings({ "finally" })
    public String executeGet(String url) {
        BufferedReader in = null;
        String content = null;
        try {
            // 定义HttpClient。
            @SuppressWarnings("resource")
            DefaultHttpClient client = new DefaultHttpClient();
            // 实例化HTTP方法。
            HttpGet request = new HttpGet();
            request.setURI(new URI(url));
            HttpResponse response = client.execute(request);
            in = new BufferedReader(new InputStreamReader(response.getEntity().getContent()
));
            StringBuffer sb = new StringBuffer("");
            String line = "";
            String NL = System.getProperty("line.separator");
            while ((line = in.readLine()) != null) {
                sb.append(line + NL);
            }
            in.close();
            content = sb.toString();
        } catch (Exception e) {
        } finally {
            if (in != null) {
                try {
                    in.close();// 关闭BufferedReader。
```

```
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
            return content;
        }
    }
    /**
     * 获取Post消息体
     *
     * @param is
     * @param contentLen
     * @return
     */
    public String GetPostBody(InputStream is, int contentLen) {
        if (contentLen > 0) {
            int readLen = 0;
            int readLengthThisTime = 0;
            byte[] message = new byte[contentLen];
            try {
                while (readLen != contentLen) {
                    readLengthThisTime = is.read(message, readLen, contentLen - readLen);
                    if (readLengthThisTime == -1) {// Should not happen.
                        break;
                    }
                    readLen += readLengthThisTime;
                }
                return new String(message);
            } catch (IOException e) {
            }
        }
        return "";
    }
    /**
     * 验证上传回调的Request
     *
     * @param request
     * @param ossCallbackBody
     * @return
     * @throws NumberFormatException
     * @throws IOException
     */
    protected boolean VerifyOSSCallbackRequest(HttpServletRequest request, String ossCallba
ckBody)
            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(autorizationInput);
        byte[] pubKey = BinaryUtil.fromBase64String(pubKeyInput);
        String pubKeyAddr = new String(pubKey);
        if (!pubKeyAddr.startsWith("http://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;
    }
    /**
     * Post请求
     */
    protected void doPost(HttpServletRequest request, HttpServletResponse 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\"}", HttpServletResponse.SC_OK);
        } else {
            response(request, response, "{\"Status\":\"verdify not ok\"}", HttpServletRespo
nse.SC_BAD_REQUEST);
        }
    }
    /**
     * 验证RSA
     *
     * @param content
     * @param sign
     * @param publicKey
     * @return
     */
    public static boolean doCheck(String content, byte[] sign, String publicKey) {
        try {
            KeyFactory keyFactory = KeyFactory.getInstance("RSA");
            byte[] encodedKey = BinaryUtil.fromBase64String(publicKey);
            PublicKey pubKey = keyFactory.generatePublic(new X509EncodedKeySpec(encodedKey)
);
            java.security.Signature signature = java.security.Signature.getInstance("MD5wit
hRSA");
            signature.initVerify(pubKey);
            signature.update(content.getBytes());
            boolean bverify = signature.verify(sign);
            return bverify;
        } catch (Exception e) {
```

```
            e.printStackTrace();
        }
        return false;
    }
    /**
     * 服务器响应结果
     *
     * @param request
     * @param response
     * @param results
     * @param status
     * @throws IOException
     */
    private void response(HttpServletRequest request, HttpServletResponse response, String
results, int status)
            throws IOException {
        String callbackFunName = request.getParameter("callback");
        response.addHeader("Content-Length", String.valueOf(results.length()));
        if (callbackFunName == null || callbackFunName.equalsIgnoreCase(""))
            response.getWriter().println(results);
        else
            response.getWriter().println(callbackFunName + "( " + results + " )");
        response.setStatus(status);
        response.flushBuffer();
    }
    /**
     * 服务器响应结果
     */
    private void response(HttpServletRequest request, HttpServletResponse response, String
results) throws IOException {
        String callbackFunName = request.getParameter("callback");
        if (callbackFunName == null || callbackFunName.equalsIgnoreCase(""))
            response.getWriter().println(results);
        else
            response.getWriter().println(callbackFunName + "( " + results + " )");
        response.setStatus(HttpServletResponse.SC_OK);
        response.flushBuffer();
    }
}
```

关于上传回调的API接口说明，请参见Callback。

2.1.6.4. Go

本文以Go语言为例，讲解在服务端通过Go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Go 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。
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 = "https://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 `https://bucketname.endpoint` ，例如 `https://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之间的通信。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";` 。
   - dir：若要设置上传到OSS文件的前缀则需要配置此项，否则置空即可。

步骤2：配置客户端

1. 下载客户端源码到PC端的本地目录。
2. 将以文件解压，并打开*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severUrl修改为应用服务器的地址。本示例中修改为 `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。

1. 登录OSS管理控制台。
2.
3. 在左侧导航栏，选择**权限管理 > 跨域设置**，然后在**跨域设置**区域，单击**设置**。
4. 单击**创建规则**，配置如下图所示。

> **说明** 为了您的数据安全，实际使用时，来源建议填写实际允许访问的域名。更多配置信息请参见设置跨域访问。

步骤4：体验上传回调

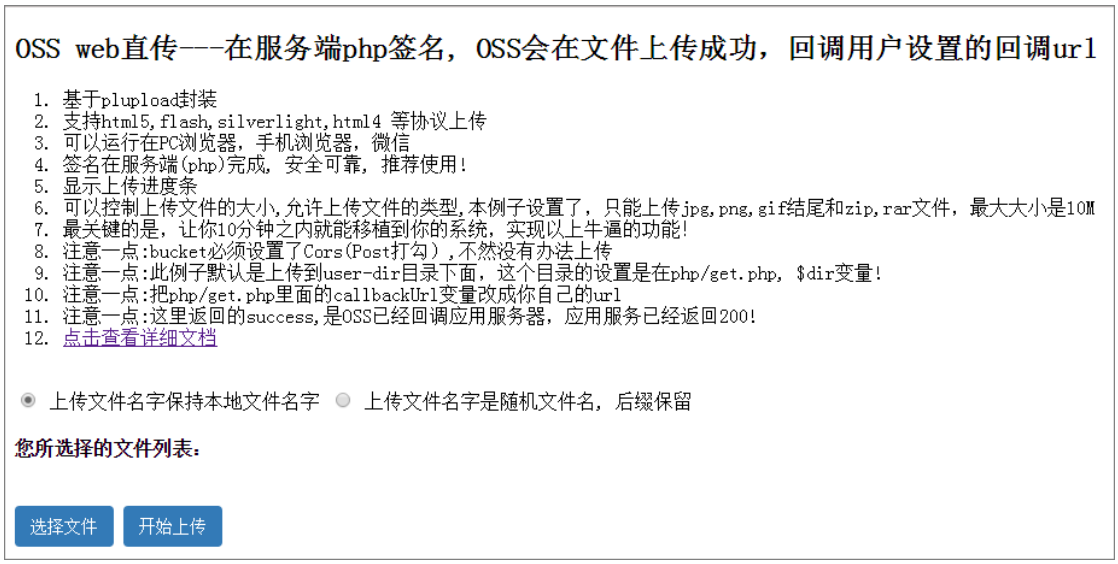
1. 启动应用服务器。

   在 `/home/aliyun/aliyun-oss-appserver-go` 目录下，执行Go命令：**go run appserver.go 11.22.33.44 1234**。

   > **说明** 请将IP和端口改成您配置的应用服务器的IP和端口。

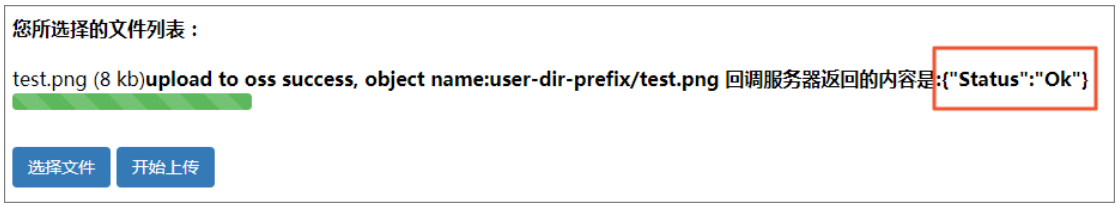
2. 启动客户端。

i. 在PC端的客户端源码目录中，打开*index.html*文件。



> 注意 index.html文件不保证兼容IE 10以下版本浏览器，若使用IE 10以下版本浏览器出现问题时，您需要自行调试。

ii. 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 Base64String
                byteAuthorization, err := getAuthorization(r)
                if (err != nil) {
                        responseFailed(w)
                        return
                }
                // Get MD5 bytes from Newly Constructed Authorization 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 according to callback_body ...
                        responseSuccess(w)  // response OK : 200
                } else {
                        responseFailed(w)   // response FAILED : 400
                }
        }
```

更多信息，请参见API文档Callback。

2.1.6.5. Node.js

本文以Node.js语言为例，介绍如何通过Node.js代码在服务端先完成签名，并设置上传回调，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Node.js 8.0以上版本。您可以执行**node -v**命令验证Node.js版本。
- 客户端运行需要浏览器支持HTML4、HTML5、Flash、Silverlight。

步骤一：配置应用服务器

1. 下载应用服务器源码。
2. 本示例中以Ubuntu 16.04为例，将源码解压到*/home/aliyun/aliyun-oss-appserver-node.js*目录下。
3. 在项目的根目录下执行**npm install**。
4. 在项目的根目录下找到源码文件 `app.js` ，修改以下代码片段：

```
const config = {
  // 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维，请登录RAM控制台创建RAM用户。
  accessKeyId: 'yourAccessKeyId',
  accessKeySecret: 'yourAccessKeySecret',
  // 填写Bucket名称。
  bucket: 'yourBucket',
  // 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之间的通信。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。例如callbackUrl填写为https://oss-demo.aliyuncs.com:23450。
  callbackUrl: 'yourCallBackUrl',
  // 当您需要设置上传到OSS文件的前缀时，请配置此项，否则置空即可。
  dir: 'yourPrefix'
}
```

步骤2：配置客户端

1. 下载客户端源码。
2. 解压文件。本例以解压到 `D:\aliyun\aliyun-oss-appserver-js` 目录为例。
3. 在该目录下打开 `upload.js` 文件，找到以下代码语句：

   ```
   // serverUrl是用户用来获取签名直传和Policy等信息的应用服务器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'
   ```

4. 将 `severUrl` 改为应用服务器的地址。例如，您可以将其修改为 `serverUrl = 'https://oss-demo.aliyuncs.com:23450'` ，客户端可通过该地址获取签名直传和Policy等信息。

步骤三：配置跨域规则

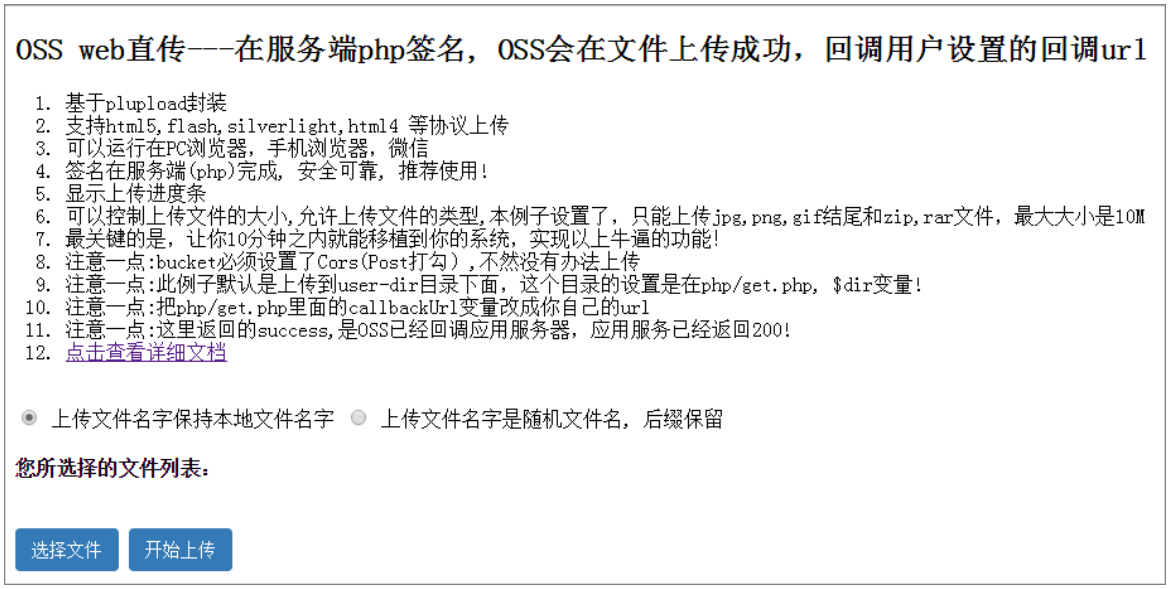
客户端进行表单直传到OSS时，会从浏览器向OSS发送带有 `Origin` 的请求消息。OSS对带有 `Origin` 头的请求消息会进行跨域规则（CORS）的验证。因此需要为Bucket设置跨域规则以支持Post方法。

1. 登录OSS管理控制台。
2.
3. 在左侧导航栏，选择**权限管理 > 跨域设置**，然后在**跨域设置**区域，单击**设置**。
4. 单击**创建规则**，配置如下图所示。

> **说明** 为了您的数据安全，请在实际使用时将**来源**填写实际允许访问的域名。各配置参数的更多信息，请参见设置跨域访问。

步骤四：体验上传回调

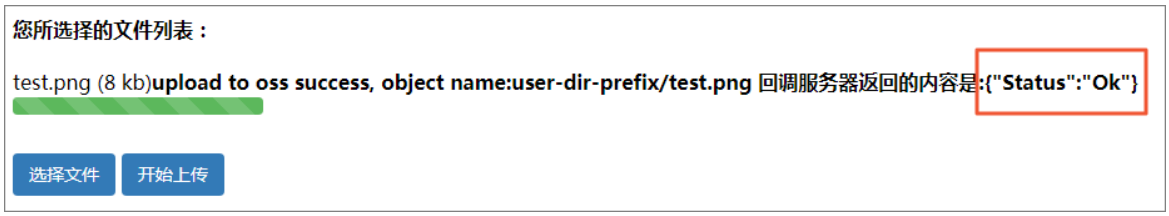
1. 在应用服务器根目录下，执行npm run server命令启动应用服务器。
2. 在PC端的客户端源码目录中，打开*index.html*文件。



> **注意** index.html文件不保证兼容IE 10以下版本浏览器。如果使用IE 10以下版本浏览器出现问题时，您需要自行调试。

3. 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。



附录：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以及上传回调服务的完整示例代码。以下仅提供核心代码片段，如需了解这两个功能的完整实现，请参见应用服务器源码（Node.js版本）。

- 签名直传服务

  获取应用服务器签名参数和回调服务器地址的代码片段如下：

```
app.get("/", async (req, res) => {
  const client = new OSS(config);
  const date = new Date();
  date.setDate(date.getDate() + 1);
  const policy = {
    expiration: date.toISOString(), //设置Unix时间戳（自UTC时间1970年01月01号开始的秒数），用于标识该请求的超时时间。
    conditions: [
      ["content-length-range", 0, 1048576000], //设置上传文件的大小限制。
    ],
  };
  // 设置跨域资源共享规则CORS。
  res.set({
    "Access-Control-Allow-Origin": req.headers.origin || "*",
    "Access-Control-Allow-Methods": "PUT,POST,GET",
  });
  // 调用SDK获取签名。
  const formData = await client.calculatePostSignature(policy);
  // 填写Bucket外网域名。
  const host = `http://${config.bucket}.${
    (await client.getBucketLocation()).location
  }.aliyuncs.com`.toString();
  // 上传回调。
  const callback = {
    callbackUrl: config.callbackUrl,// 设置回调请求的服务器地址，例如http://oss-demo.aliyuncs.com:23450。
    callbackBody:// 设置回调的内容，例如文件ETag、资源类型mimeType等。
      "filename=${object}&size=${size}&mimeType=${mimeType}&height=${imageInfo.height}&width=${imageInfo.width}",
    callbackBodyType: "application/x-www-form-urlencoded",//  设置回调的内容类型。
  };
  // 返回参数。
  const params = {
    expire: moment().add(1, "days").unix().toString(),
    policy: formData.policy, // 从OSS服务器获取到的Policy。
    signature: formData.Signature, // 从OSS服务器获取到的Signature。
    accessid: formData.OSSAccessKeyId,// 从OSS服务器获取到的OSS AccessKeyId。
    host, // 格式为https://bucketname.endpoint，例如https://bucket-name.oss-cn-hangzhou.aliyuncs.com。
    callback: Buffer.from(JSON.stringify(callback)).toString("base64"),// 通过Buffer.from对JSON进行Base64编码。
    dir: config.dir,// 获取到的文件前缀。
  };
  res.json(params);
});
```

- 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：

```
// 监听/result路径下的POST请求。
app.post("/result", (req, res) => {
  // 通过Base64解码公钥地址。
  const pubKeyAddr = Buffer.from(
    req.headers["x-oss-pub-key-url"],
    "base64"
  ).toString("ascii");
  //  判断请求头中的x-oss-pub-key-url是否来源于OSS服务器。
  if (
    !pubKeyAddr.startsWith("https://gosspublic.alicdn.com/") &&
    !pubKeyAddr.startsWith("https://gosspublic.alicdn.com/")
  ) {
    System.out.println("pub key addr must be oss addrss");
    // 如果x-oss-pub-key-url不是来源于OSS服务器，则返回“verify not ok”，表明回调失败。
    res.json({ Status: "verify not ok" });
  }
  // 如果x-oss-pub-key-url来源于OSS服务器，则返回“Ok”，表明回调成功。
  res.json({ Status: "Ok" });
});
```

有关上传回调的更多信息，请参见Callback。

2.1.6.6. .NET

本文以.NET语言为例，介绍如何在服务端完成签名，并设置上传回调，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。
- 应用服务器已安装Visual Studio 2012或更高版本。
- 应用服务器已经安装.Net Framework 4.5及以上版本。

步骤1：配置应用服务器

1. 下载应用服务器源码（.NET版本）。
2. 本示例中以Windows环境为例，将下载的文件解压到*C:\callback-server-dotnet*目录下。
3. 进入该目录，找到并打开源码文件*TinyHttpServer.cs*，修改如下的代码片段：

```
// 请填写您的AccessKeyId。
public static string accessKeyId = "<yourAccessKeyId>";
// 请填写您的AccessKeySecret。
public static string accessKeySecret = "<yourAccessKeySecret>";
// host的格式为https://bucketname.endpoint，请替换为您的真实信息。
public static string host = "https://bucketname.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。
public static string callbackUrl = "http://192.168.0.1:8888";
// 用户上传文件时指定的前缀。
public static string uploadDir = "user-dir-prefix/";
```

   - accessKeyId：设置您的AccessKeyId。
   - accessKeySecret：设置您的AessKeySecret。
   - host：格式为https://bucketname.endpoint，例如https://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之间的通信。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://10.10.10.10:1234";`。
- uploadDir：设置上传到OSS文件的前缀，以便于区分文件。您也可以填写空值。

4. 使用Visual Studio打开*aliyun-oss-net-callback-server.sln*工程文件，单击**启动**，生成执行程序*aliyun-oss-net-callback-server.exe*。

步骤2：配置客户端

1. 下载客户端源码。
2. 将文件解压，本例中以解压到`D:\aliyun\aliyun-oss-appserver-js`目录为例。
3. 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://192.168.0.1:8888'
```

4. 将`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。例如本示例中可修改为：`serverUrl = 'http://10.10.10.10:1234'`。

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。

1. 登录OSS管理控制台。
2.
3. 在左侧导航栏，选择**权限管理 > 跨域设置**，然后在**跨域设置**区域，单击**设置**。
4. 单击**创建规则**，配置如下图所示。

> **说明** 为了您的数据安全，实际使用时，来源建议填写实际允许访问的域名。更多配置信息请参见设置跨域访问。

步骤4：体验上传回调

1. 启动应用服务器。

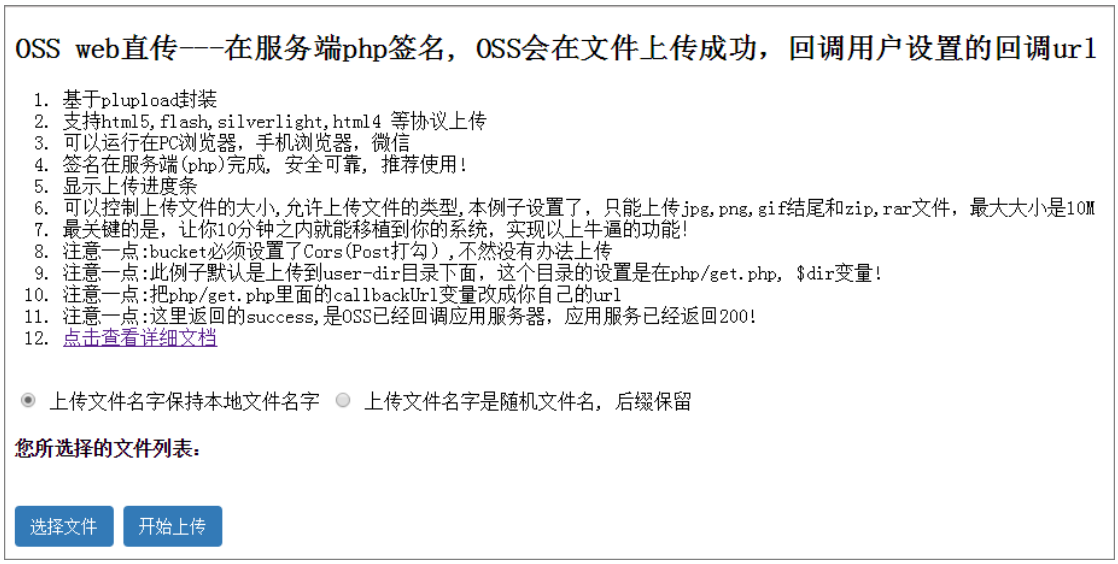
   在应用服务器的命令行窗口下，进入到*aliyun-oss-net-callback-server.exe*执行程序生成的目录下，执行命令**aliyun-oss-net-callback-server.exe ${ip} ${port}**启动应用服务器。

   ```
   cd C:\Users\Desktop\callback-server-dotnet\aliyun-oss-net-callback-server\bin\Debug\
   aliyun-oss-net-callback-server.exe 10.10.10.10 1234
   ```

   > **说明**
   > - 程序目录以实际环境为准。您在使用Visual Studio生成*aliyun-oss-net-callback-server.exe*程序时，可以看到程序目录。
   > - ${ip}和${port} 修改成配置应用服务器的IP和端口。例如本示例中为**aliyun-oss-net-callback-server.exe 10.10.10.10 1234**。

2. 启动客户端。

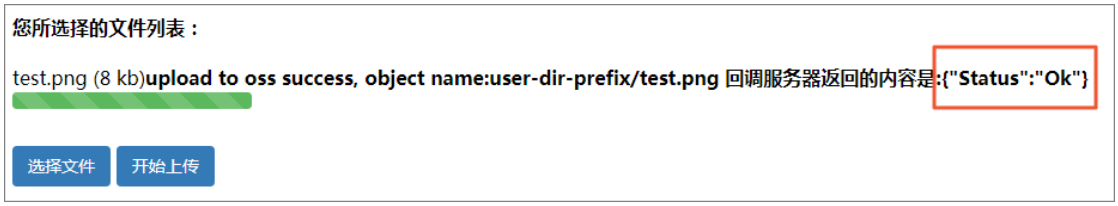
   i. 在PC端的客户端源码目录中，打开*index.html*文件。

   > **注意** index.html文件不保证兼容IE 10以下版本浏览器，若使用IE 10以下版本浏览器出现问题时，您需要自行调试。

ii. 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。

- 签名直传服务响应客户端发送给应用服务器的GET消息，代码片段如下：

```
private static string GetPolicyToken()
{
  //expireTime
  var expireDateTime = DateTime.Now.AddSeconds(expireTime);
  // example of policy
  //{
  //  "expiration": "2020-05-01T12:00:00.000Z",
  //  "conditions": [
  //    ["content-length-range", 0, 1048576000]
  //    ["starts-with", "$key", "user-dir-prefix/"]
  //  ]
  //}
  //policy
  var config = new PolicyConfig();
  config.expiration = FormatIso8601Date(expireDateTime);
  config.conditions = new List<List<Object>>();
  config.conditions.Add(new List<Object>());
  config.conditions[0].Add("content-length-range");
  config.conditions[0].Add(0);
  config.conditions[0].Add(1048576000);
  config.conditions.Add(new List<Object>());
  config.conditions[1].Add("starts-with");
  config.conditions[1].Add("$key");
  config.conditions[1].Add(uploadDir);
  var policy = JsonConvert.SerializeObject(config);
  var policy_base64 = EncodeBase64("utf-8", policy);
  var signature = ComputeSignature(accessKeySecret, policy_base64);
  //callback
  var callback = new CallbackParam();
  callback.callbackUrl = callbackUrl;
  callback.callbackBody = "filename=${object}&size=${size}&mimeType=${mimeType}&height=${
imageInfo.height}&width=${imageInfo.width}";
  callback.callbackBodyType = "application/x-www-form-urlencoded";
  var callback_string = JsonConvert.SerializeObject(callback);
  var callback_string_base64 = EncodeBase64("utf-8", callback_string);
  var policyToken = new PolicyToken();
  policyToken.accessid = accessKeyId;
  policyToken.host = host;
  policyToken.policy = policy_base64;
  policyToken.signature = signature;
```

```
  policyToken.signature = signature;
  policyToken.expire = ToUnixTime(expireDateTime);
  policyToken.callback = callback_string_base64;
  policyToken.dir = uploadDir;
  return JsonConvert.SerializeObject(policyToken);
}
public void DoGet()
{
  Console.WriteLine("DoGet request: {0}", this.httpURL);
  var content = GetPolicyToken();
  this.swResponse.WriteLine("HTTP/1.0 200 OK");
  this.swResponse.WriteLine("Content-Type: application/json");
  this.swResponse.WriteLine("Access-Control-Allow-Origin: *");
  this.swResponse.WriteLine("Access-Control-Allow-Method: GET, POST");
  this.swResponse.WriteLine($"Content-Length: {content.Length.ToString()}");
  this.swResponse.WriteLine("Connection: close");
  this.swResponse.WriteLine("");
  this.swResponse.WriteLine(content);
}
```

- 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：

```
public bool VerifySignature()
{
  // Get the Authorization Base64 from Request
  if (this.httpHeadersDict["authorization"] != null)
  {
    this.strAuthorizationRequestBase64 = this.httpHeadersDict["authorization"].ToString()
;
  } else if (this.httpHeadersDict["Authorization"] != null) {
    this.strAuthorizationRequestBase64 = this.httpHeadersDict["Authorization"].ToString()
;
  }
  if (this.strAuthorizationRequestBase64 == "")
  {
    Console.WriteLine("authorization property in the http request header is null. ");
    return false;
  }
  // Decode the Authorization from Request
  this.byteAuthorizationRequest = Convert.FromBase64String(this.strAuthorizationRequestBa
se64);
  // Decode the URL of PublicKey
  this.strPublicKeyURLBase64 = this.httpHeadersDict["x-oss-pub-key-url"].ToString();
  var bytePublicKeyURL = Convert.FromBase64String(this.strPublicKeyURLBase64);
  var strAsciiPublickeyURL = System.Text.Encoding.ASCII.GetString(bytePublicKeyURL);
  // Get PublicKey from the URL
  ServicePointManager.ServerCertificateValidationCallback = new RemoteCertificateValidati
onCallback(validateServerCertificate);
  HttpWebRequest request = (HttpWebRequest)WebRequest.Create(strAsciiPublickeyURL);
  HttpWebResponse response = (HttpWebResponse)request.GetResponse();
  StreamReader srPublicKey = new StreamReader(response.GetResponseStream(), Encoding.UTF8
);
  this.strPublicKeyBase64 = srPublicKey.ReadToEnd();
  response.Close();
  srPublicKey.Close();
```

```
  this.strPublicKeyContentBase64 = this.strPublicKeyBase64.Replace("-----BEGIN PUBLIC KEY
-----\n", "").Replace("-----END PUBLIC KEY-----", "").Replace("\n", "");
  this.strPublicKeyContentXML = this.RSAPublicKeyString2XML(this.strPublicKeyContentBase6
4);
  // Generate the New Authorization String according to the HttpRequest
  String[] arrURL;
  if (this.httpURL.Contains('?'))
  {
    arrURL = this.httpURL.Split('?');
    this.strAuthSourceForMD5 = String.Format("{0}?{1}\n{2}", System.Web.HttpUtility.UrlDe
code(arrURL[0]), arrURL[1], this.httpBody);
  }
  else
  {
    this.strAuthSourceForMD5 = String.Format("{0}\n{1}", System.Web.HttpUtility.UrlDecode
(this.httpURL), this.httpBody);
  }
  // MD5 hash bytes from the New Authorization String
  var byteAuthMD5 = byteMD5Encrypt32(this.strAuthSourceForMD5);
  // Verify Signature
  System.Security.Cryptography.RSACryptoServiceProvider RSA = new System.Security.Cryptog
raphy.RSACryptoServiceProvider();
  try
  {
    RSA.FromXmlString(this.strPublicKeyContentXML);
  }
  catch (System.ArgumentNullException e)
  {
    throw new ArgumentNullException(String.Format("VerifySignature Failed : RSADeformatte
r.VerifySignature get null argument : {0} .", e));
  }
  catch (System.Security.Cryptography.CryptographicException e)
  {
    throw new System.Security.Cryptography.CryptographicException(String.Format("VerifySi
gnature Failed : RSA.FromXmlString Exception : {0} .", e));
  }
  System.Security.Cryptography.RSAPKCS1SignatureDeformatter RSADeformatter = new System.S
ecurity.Cryptography.RSAPKCS1SignatureDeformatter(RSA);
  RSADeformatter.SetHashAlgorithm("MD5");
  var bVerifyResult = false;
  try
  {
    bVerifyResult = RSADeformatter.VerifySignature(byteAuthMD5, this.byteAuthorizationReq
uest);
  }
  catch (System.ArgumentNullException e)
  {
    throw new ArgumentNullException(String.Format("VerifySignature Failed : RSADeformatte
r.VerifySignature get null argument : {0} .", e));
  }
  catch (System.Security.Cryptography.CryptographicUnexpectedOperationException e)
  {
    throw new System.Security.Cryptography.CryptographicUnexpectedOperationException(Stri
ng.Format("VerifySignature Failed : RSADeformatter.VerifySignature Exception : {0} .", e)
```

```
);
  }
  return bVerifyResult;
}
public void DoPost()
{
  this.GetPostBody();
  // Verify Signature
  try
  {
    if (this.VerifySignature())
    {
      Console.WriteLine("\nVerifySignature Successful . \n");
      // do something accoding to callback_body ...
      this.HttpResponseSuccess();
    }
    else
    {
      Console.WriteLine("\nVerifySignature Failed . \n");
      this.HttpResponseFailure();
    }
  }
  catch
  {
    Console.WriteLine("\nVerifySignature Failed . \n");
    this.HttpResponseFailure();
  }
}
```

2.1.6.7. 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

本文以Ubuntu16.04为例介绍使用不同Web服务器时的环境配置，请根据实际选择。

- 当使用Apache作为Web服务器时，请进行如下环境配置，此处以Apache2.4.18为例说明。
  - Web服务器外网IP地址为 `192.0.2.11` 。您可以在配置文件 `/etc/apache2/apache2.conf` 中增加 `ServerName 192.0.2.11` 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。

- 当使用nginx作为Web服务器时，请进行如下环境配置，此处以nginx1.19.7为例说明。

  Web服务器外网IP地址为 `192.0.2.11` ，监听端口为 `8080` 。您可以在配置文件 `/etc/nginx/nginx.conf` 中修改外网IP地址和端口。配置文件示例如下：

```
server {
    listen 8080;
    server_name 192.0.2.11;
    root /var/www/html;
    index index.html index.php;
    location ~* \.php$ {
            fastcgi_pass 127.0.0.1:9000;
            fastcgi_index index.php;
            fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_name;
            include fastcgi_params;
        }
}
```

  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nginx服务器。

步骤2：配置应用服务器

1. 下载应用服务器源码（PHP版本）。
2. 将应用服务器源码解压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。
3. PC端浏览器中访问应用服务器URL `http://192.0.2.11:8080/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。
4. 如果使用Apache作为Web服务器，请开启Apache捕获HTTP头部Authorization字段的功能。如果使用nginx作为Web服务器，请跳过此步骤。

   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。例如Apache2，需要设置成不解析头部。

   i. 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>
```

   ii. 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。

5. 修改应用服务器配置。

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>';          // 填写您的AccessKey ID。
$key= '<yourAccessKeySecret>';     // 填写您的AccessKey Secret。
// $host的格式为https://bucketname.endpointx，请替换为您的真实信息。
$host = 'https://bucket-name.oss-cn-hangzhou.aliyuncs.com';
// $callbackUrl为上传回调服务器的URL，请将以下IP和Port配置为您自己的真实URL信息。
$callbackUrl = 'http://192.0.2.11:8080/aliyun-oss-appserver-php/php/callback.php';
$dir = 'user-dir-prefix/';          // 上传文件时指定的前缀。
```

配置项	是否必填	示例值	描述
id	是	LTAn**********	阿里云账号或者RAM用户的AccessKey ID和AccessKey Secret。具体操作，请参见获取AccessKey。
key	是	zbnK**********	
host	是	https://bucket-name.oss-cn-hangzhou.aliyuncs.com	访问地址，格式为 `https://BucketName.Endpoint` 。关于Endpoint的更多信息，请参见访问域名和数据中心。
callbackUrl	是	http://192.0.2.11:8080/aliyun-oss-appserver-php/php/callback.php	上传回调URL，即回调服务器地址，用于处理应用服务器与OSS之间的通信。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。
dir	否	exampledir/	上传到OSS的文件前缀。请根据实际需要配置此项。 如果不需要设置文件前缀，设置为空即可。

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，因为相对路径也是可以正常工作的。如果确实需要修改，请找到此段代码片段 `serverUrl ='./php/get.php'` ，将 `serverUrl` 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。例如本示例中可以修改为 `serverUrl ='http://192.0.2.11:8080/aliyun-oss-appserver-php/php/get.php'` 。

步骤4：修改CORS

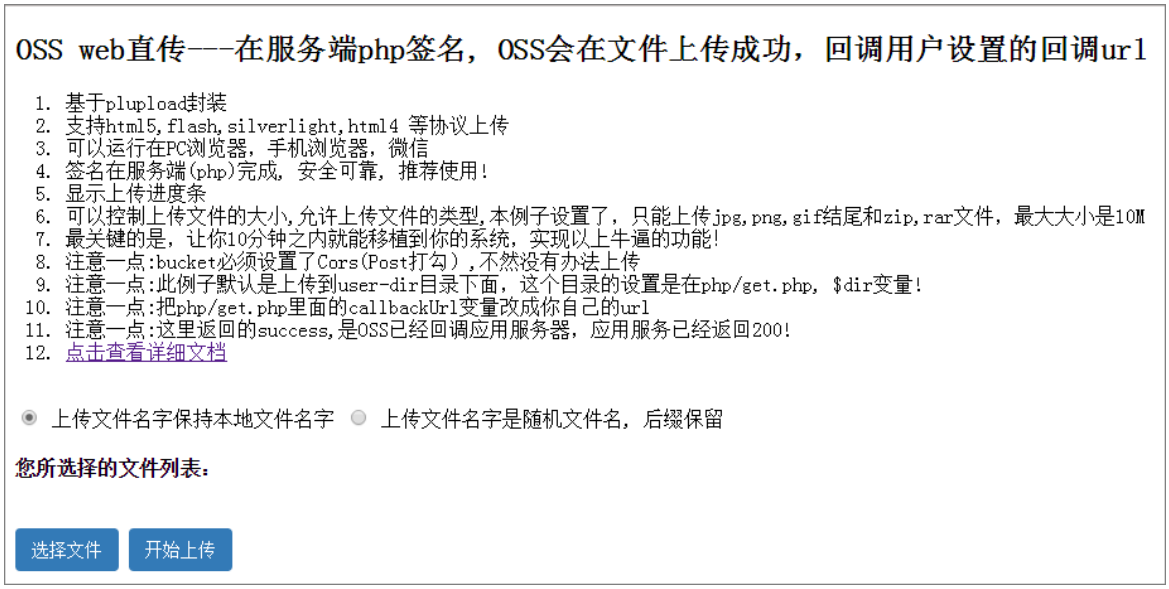
客户端进行表单直传到OSS时，会从浏览器向OSS发送带有 `Origin` 的请求消息。OSS对带有 `Origin` 头的请求消息会进行跨域规则（CORS）的验证。因此需要为Bucket设置跨域规则以支持Post方法。

1. 登录OSS管理控制台。
2.
3. 在左侧导航栏，选择**权限管理 > 跨域设置**，然后在**跨域设置**区域，单击**设置**。
4. 单击**创建规则**，配置如下图所示。

> **说明** 为了您的数据安全，实际使用时，**来源**建议填写实际允许访问的域名。更多配置信息请参见设置跨域访问。

步骤5：体验上传回调

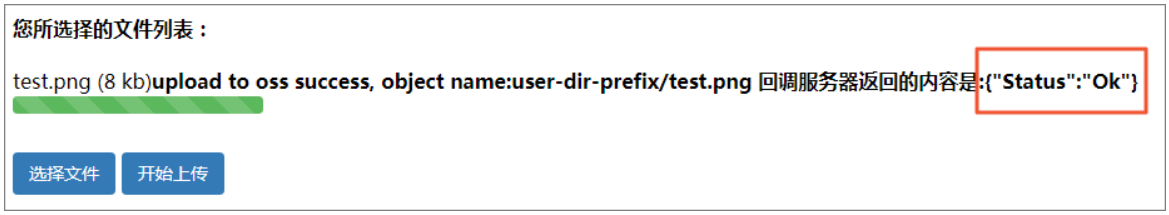
1. 在PC端的Web浏览器中输入http://192.0.2.11:8080/aliyun-oss-appserver-php/index.html。



> **注意** index.html文件不保证兼容IE 10以下版本浏览器，若使用IE 10以下版本浏览器出现问题时，您需要自行调试。

2. 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);
}
```

  更多信息，请参见API参考中的Callback。

2.1.6.8. 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**ruby -v**命令进行查看）。
- 确保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（Ruby版本）。
2. 以Ubuntu 16.04为例，将文件解压到*/home/aliyun/aliyun-oss-appserver-ruby*目录下。
3. 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 = 'https://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https://bucketname.endpoint，例如https://bucket-name.oss-cn-hangzhou.aliyuncs.com。关于Endpoint的介绍，请参见Endpoint访问域名。
   - $callback_url：设置上传回调URL，即回调服务器地址，用于处理应用服务器与OSS之间的通信。OSS会在文件上传完成后，把文件上传信息通过此回调URL发送给应用服务器。本例中修改为：`$callback_url="http://11.22.33.44:1234";`。
   - $upload_dir：若要设置上传到OSS文件的前缀则需要配置此项，否则置空即可。

步骤2：配置客户端

1. 下载客户端源码。
2. 将文件解压，本示例解压至*D:\aliyun\aliyun-oss-appserver-js*目录。
3. 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'
```

4. 将`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 = '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。

1. 登录OSS管理控制台。
2.
3. 在左侧导航栏，选择**权限管理 > 跨域设置**，然后在**跨域设置**区域，单击**设置**。
4. 单击**创建规则**，配置如下图所示。

> **说明** 为了您的数据安全，实际使用时，**来源**建议填写实际允许访问的域名。更多配置信息请参见设置跨域访问。

步骤 4：体验上传回调

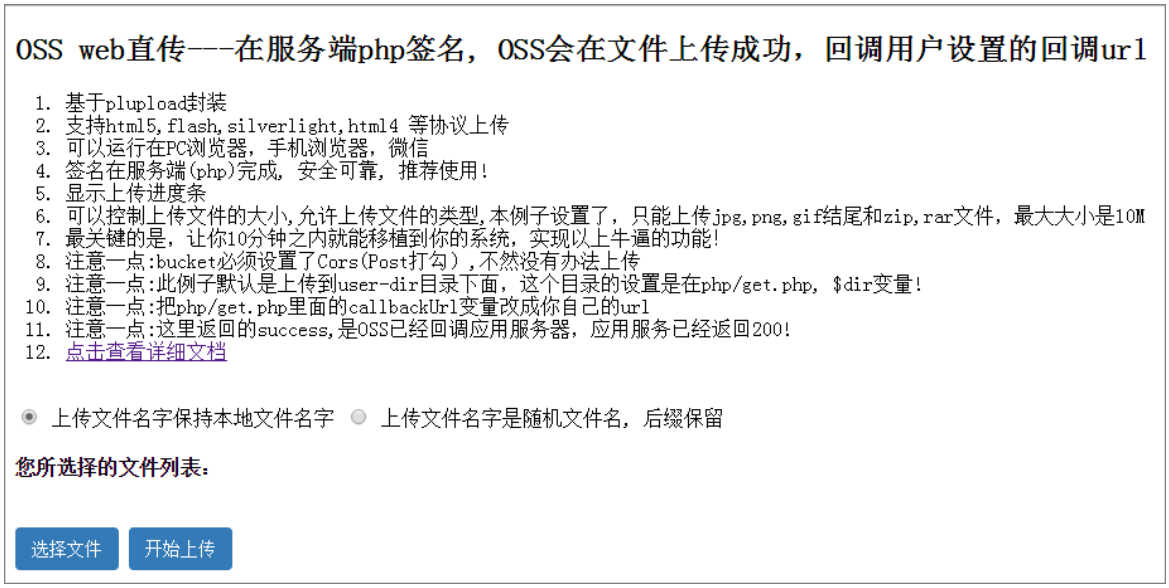
1. 启动应用服务器。

   在*/home/aliyun/aliyun-oss-appserver-ruby*目录下，执行**ruby appserver.rb 11.22.33.44 1234**命令启动应用服务器。

   > **说明** 请将IP和端口改成您配置的应用服务器的IP和端口。

2. 启动客户端。

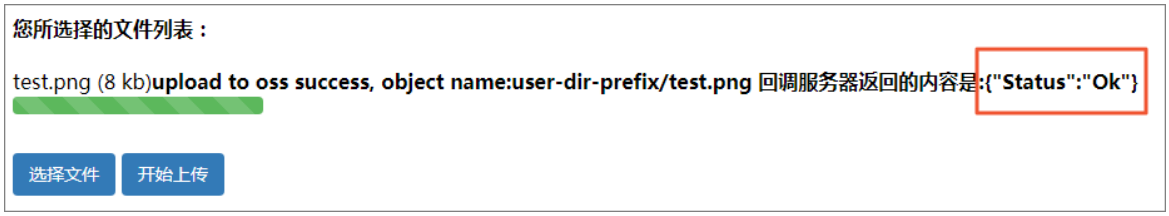
   在PC端的客户端源码目录中，打开*index.html*文件。



> 注意 index.html文件不保证兼容IE 10以下版本浏览器，若使用IE 10以下版本浏览器出现问题时，您需要自行调试。

3. 上传文件。

   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', $access_key_secret, policy_encode)
    hs = Digest::MD5.hexdigest(h)
    sign_result = Base64.strict_encode64(h).strip()
    callback_dict = {}
    callback_dict['callbackBodyType'] = 'application/x-www-form-urlencoded';
    callback_dict['callbackBody'] = 'filename=${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_string + "\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
```

更多详情请参见API文档Callback。

2.2. 移动应用端直传实践

2.2.1. 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。

前提条件

- 已开通OSS服务。详情请参见开通OSS服务。
- 已创建Bucket。详情请参见创建Bucket。

背景信息

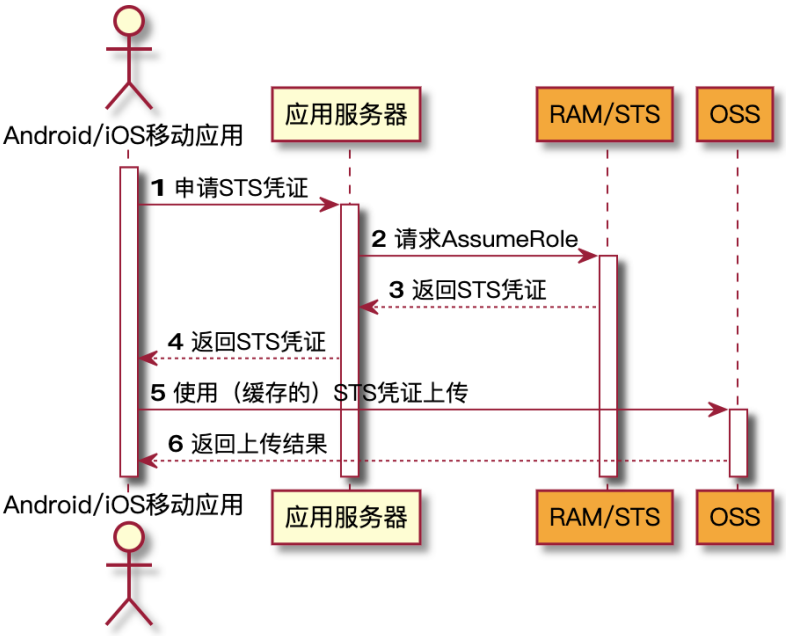
在移动互联的时代，手机App上传的数据越来越多。作为开发者，您可以利用OSS处理各种数据存储需求，从而更加专注于自己的应用逻辑。

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。

流程介绍

移动应用直传服务的开发流程如下：



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。

   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。

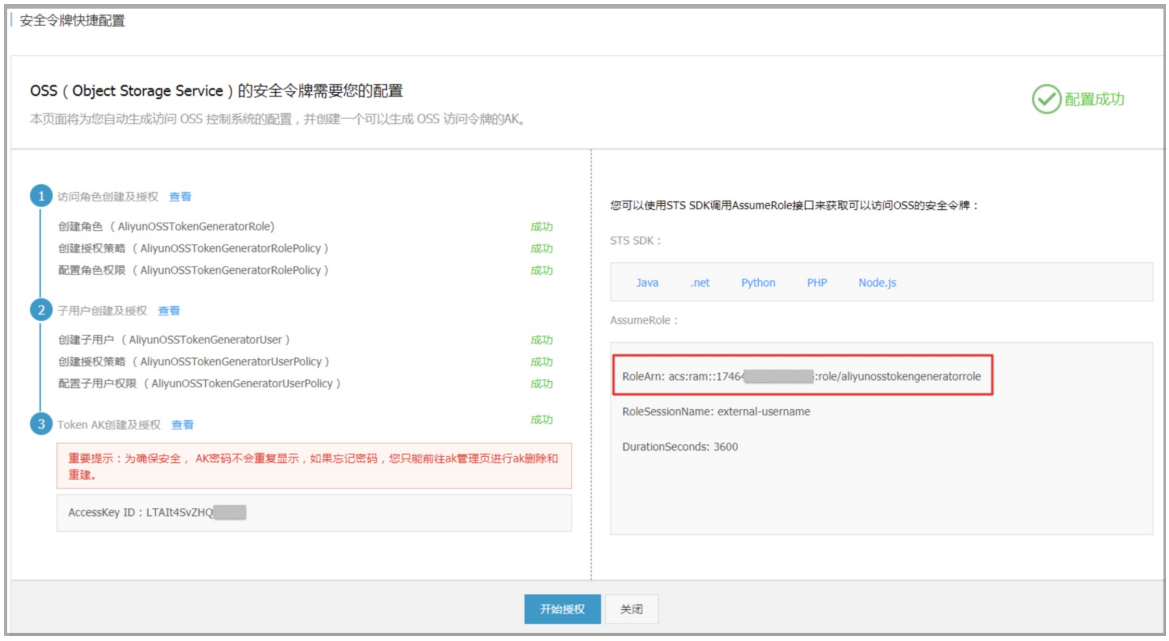
2. 应用服务器检测上述请求的合法性，然后返回Token给移动应用。
3. Android、iOS移动应用使用此Token将数据上传到OSS，或者从OSS下载数据。

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：开通STS服务

1. 登录OSS管理控制台。
2. 单击左侧导航栏的**概览**。
3. 在**大家都在用**区域，单击**安全令牌（子账号授权）** > **前往RAM控制台**。
4. 单击**开始授权**。并按照提示完成授权。

   授权完成后，保存RAM用户的AccessKey ID、AccessKey Secret和RoleArn三个参数。



步骤2：配置应用服务器

1. 下载应用服务器代码。

语言	下载地址
PHP	sts-server.zip
Java	AppTokenServerDemo.zip
Ruby	sts-app-server-master.zip
Node.js	sts-app-server-node.zip
Go	test_token_server.zip

2. 修改配置文件。

   以下例子采用PHP语言编写。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/bucket_write_policy.txt"
}
```

   参数描述如下：

   - AccessKeyID：填写之前记录的AccessKey ID。
   - AccessKeySecret：填写之前记录的AccessKey Secret。
   - 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。

代码示例中提供了以下两种最常用的Token权限文件，位于policy目录下。使用时，需要把相应的权限文件里的 `$BUCKET_NAME` （Bucket名称）和 `$OBJECT_PREFIX` （文件前缀）替换成指定的值。

- bucket_read_policy.txt：指定该Token拥有该账号下指定Bucket及指定前缀的读权限。
- bucket_write_policy.txt：指定该Token拥有该账号下指定Bucket及指定前缀的写权限。

如果需要更多权限设置，请参见RAM Policy概述中的Policy示例和构建方法。 注意请务必根据业务需要，按照最小权限原则进行授权。如果您直接授予所有资源（resource:*），或者所有操作（action:*）权限，则存在由于权限范围过大导致的数据安全风险。

> **警告** 本代码示例仅做参考，实际业务务必根据不同用户或设备进行权限隔离，生成带有不同权限的Token。

3. 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），默认监听7080端口。支持自定义需要监听的端口，但不允许与已有的端口重复。

     您需要先执行**java -jar app-token-server.jar**启动服务，然后调用AssumeRole返回结果。

   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/cjLYg80Aho1ek0Jm63XMhr9Oc5s·∂
·∂3qaPer8p1YaX1NTDiCFZWFkvlHf1pQhuxfKBc+mRR9KAbHUefqH+rdjZqjTF7p2m1wJXP8S6k+G2MpHrUe6TY
BkJ43GhhTVFMuM3BZajY3VjZWOXBIODRIR1FKZjIiEjMzMzE0MjY0NzM5MTE4NjkxMSoLY2xpZGSSDgSDGAGESG
TETqOio6c2RrLWRlbW8vKgoUYWNzOm9zczoqOio6c2RrLWRlbW9KEDExNDg5MzAxMDcyNDY4MThSBTI2ODQyWg9
Bc3N1bWVkUm9sZVVzZXJgAGoSMzMzMTQyNjQ3MzkxMTg2OTExcglzZGstZGVtbzI=",
   "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 AccessKey ID。
   - AccessKeySecret：Android、iOS移动应用初始化OSSClient获取AccessKey Secret。

- 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：获取Token的状态。获取失败时，返回值是500。
- ErrorCode：错误原因。
- ErrorMessage：错误描述。

步骤3：下载并安装移动应用

1. 下载应用源码。

   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。

2. 打开移动应用，配置应用参数。

   - 应用服务器：填写步骤2：配置应用服务器中部署的应用服务器地址。
   - 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。
   - OSS文件名：需要包含应用服务器Policy配置文件里指定的前缀。

3. 单击**设置**。

步骤4：体验移动应用直传服务

1. 打开移动应用。
2. 单击**选择图片**，选择需要上传的图片，并设置OSS文件名。

3. 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https://example.com:8080"
OSSCredentialProvider credentialProvider = new OSSAuthCredentialsProvider(stsServer);
// 完成以下配置项。
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] initWithAuthSer
verUrl:@"应用服务器地址，例如https://example.com:8080"];
client = [[OSSClient alloc] initWithEndpoint:endPoint credentialProvider:credential];
```

2.2.2. 快速搭建移动应用上传回调服务

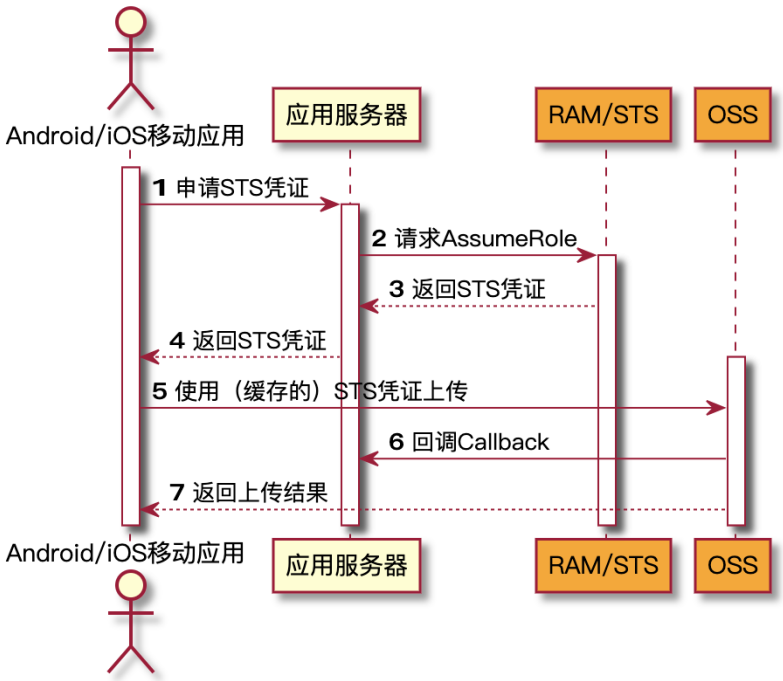
本文讲解如何搭建一个基于OSS的移动应用数据直传服务并设置上传回调。

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。但该方案有个问题：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，就无法对应用上传数据进行管理。为此OSS提供了上传回调方案。

流程介绍

上传回调的开发流程如下：



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7）之间，触发一个上传回调任务，即第上图中步骤6，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
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，指定以下自定义参数：

- 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

- 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，例如 `http://example.com/callback.php` 。
- 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再返回给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），例如 `http://example.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://example.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://example.com/callback.php",
                          @"callbackBody": @"filename=${object}&size=${size}&phone=${x:ph
one}&system=${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://example.com/callback.php");
        put("callbackBody", "filename=${object}&size=${size}&phone=${x:phone}&system=${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/kdD1ktNVgbWE****G0G2SU/RaHBovRCE8OkQDjC3u
G33esH2txA==
x-oss-pub-key-url: aHR0cDovL2dvc3NwdWJsaWMuYWxpY2RuLmNv****YWxsYmFja19wdWJfa2V5X3YxLnBlbQ==
filename=test.txt&size=5&phone=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&phone=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。
  - 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，如果有问题请依据提供的代码自行修改。这是一个Maven项目。

- PHP
  - 下载地址
  - 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。
  - 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。
  - 运行方法： 运行**ruby aliyun_oss_callback_server.rb**。

2.3. 使用ECS实例反向代理OSS

2.3.1. 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。

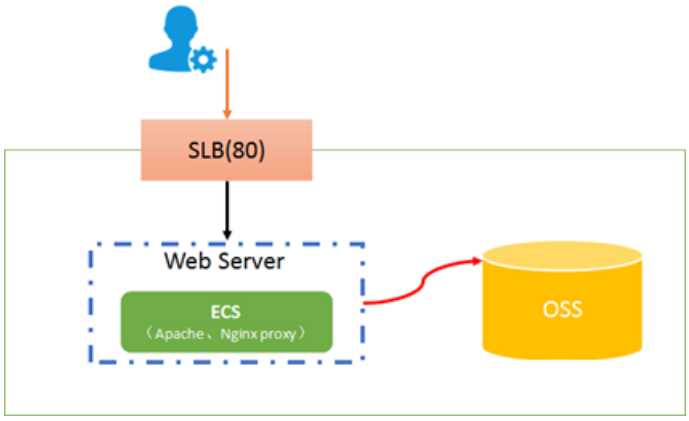
背景信息

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，创建过程请参见创建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;
           location / {
               proxy_pass http://bucketname.oss-cn-beijing-internal.aliyuncs.com;
               #proxy_set_header Host $host;
       }
   }
   ```

   - server_name：对外提供反向代理服务的IP，即ECS实例的外网地址。
   - proxy_pass：填写跳转的域名。

- 当ECS实例与Bucket在同一地域时，填写目标Bucket的内网访问域名。访问域名介绍请参见OSS访问域名使用规则。
- 当ECS实例与Bucket不在同一地域时，填写目标Bucket的外网访问域名。
- 因OSS的安全设置，当使用默认域名通过浏览器访问OSS中的图片或网页文件时，会直接下载。所以，若您的用户需通过浏览器预览Bucket中的图片或网页文件，需为Bucket绑定自定义域名，并在此项中添加已绑定的域名。绑定自定义域名操作请参见绑定自定义域名。

- proxy_set_header Host $host：添加此项时，Nginx会在向OSS请求的时候，将host替换为ECS的访问地址。遇到以下情况时，您需要添加此项。
  - 遇到签名错误问题。
  - 如果您的域名已解析到ECS实例的外网上，且您的用户需要通过浏览器预览Bucket中的图片或网页文件。您可以将您的域名绑定到ECS实例代理的Bucket上，不配置CNAME。这种情况下，proxy_pass项可直接配置Bucket的内网或外网访问地址。绑定自定义域名操作请参见绑定自定义域名。

> 注意
> - 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，配置方法请参见反向代理配置。
> - 此种配置方式只能代理一个Bucket的访问。

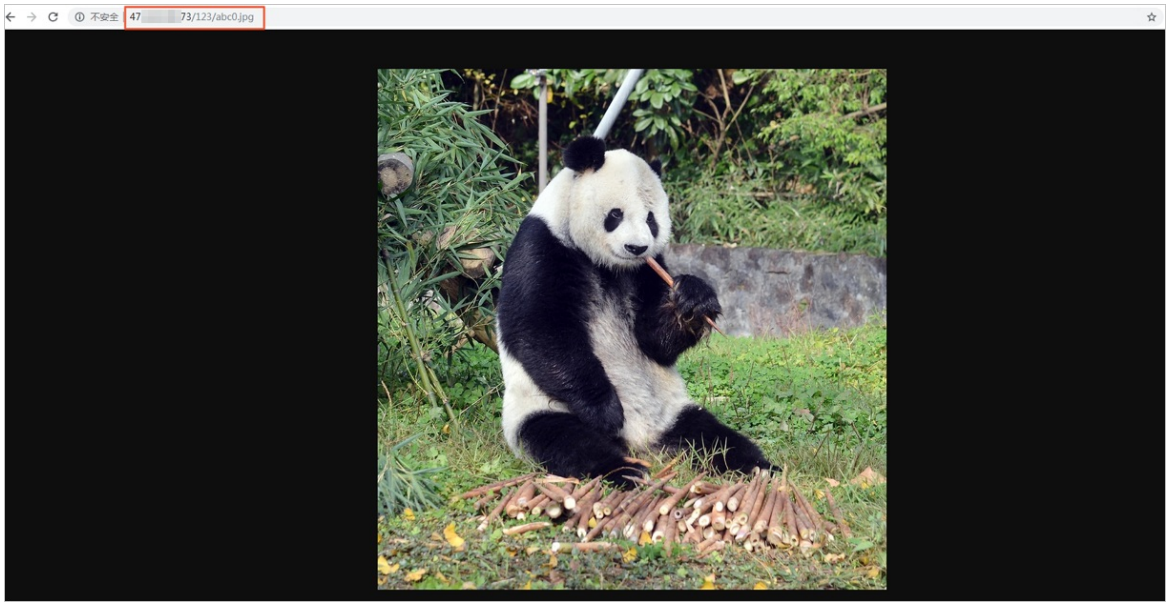
6. 进入Nginx主程序文件夹，启动Nginx。

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

7. 开放ECS实例的TCP 80端口。

   Nginx默认使用80端口，需在ECS的安全组配置中，允许用户访问TCP 80端口。配置方式请参见添加安全组规则。

8. 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

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

2.3.2. 基于CentOS的ECS实例实现OSS反向代理

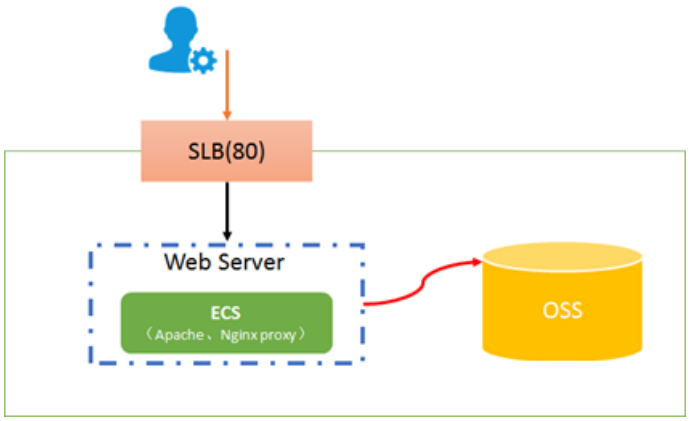
阿里云OSS的存储空间（Bucket）访问地址会随机变换，您可以通过在ECS实例上配置OSS的反向代理，实现通过固定IP地址访问OSS的存储空间。

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。创建过程请参见创建ECS实例。

2. 使用root用户登录ECS实例并安装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;
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;
proxy_set_header Host $host;
}
```

- server_name：对外提供反向代理服务的IP，即ECS实例的外网地址。
- proxy_pass：填写跳转的域名。
  - 当ECS实例与Bucket在同一地域时，填写目标Bucket的内网访问域名。访问域名介绍请参见OSS访问域名使用规则。
  - 当ECS实例与Bucket不在同一地域时，填写目标Bucket的外网访问域名。
  - 因OSS的安全设置，当使用默认域名通过浏览器访问OSS中的图片或网页文件时，会直接下载。所以，若您的用户需通过浏览器预览Bucket中的图片或网页文件，需为Bucket绑定自定义域名，并在此项中添加已绑定的域名。绑定自定义域名操作请参见绑定自定义域名。
- proxy_set_header Host $host：添加此项时，Nginx会在向OSS请求的时候，将host替换为ECS的访问地址。遇到以下情况时，您需要添加此项。
  - 遇到签名错误问题。
  - 如果您的域名已解析到ECS实例的外网上，且您的用户需要通过浏览器预览Bucket中的图片或网页文件。您可以将您的域名绑定到ECS实例代理的Bucket上，不配置CNAME。这种情况下，proxy_pass项可直接配置Bucket的内网或外网访问地址。绑定自定义域名操作请参见绑定自定义域名。

> **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，配置方法请参见反向代理配置。

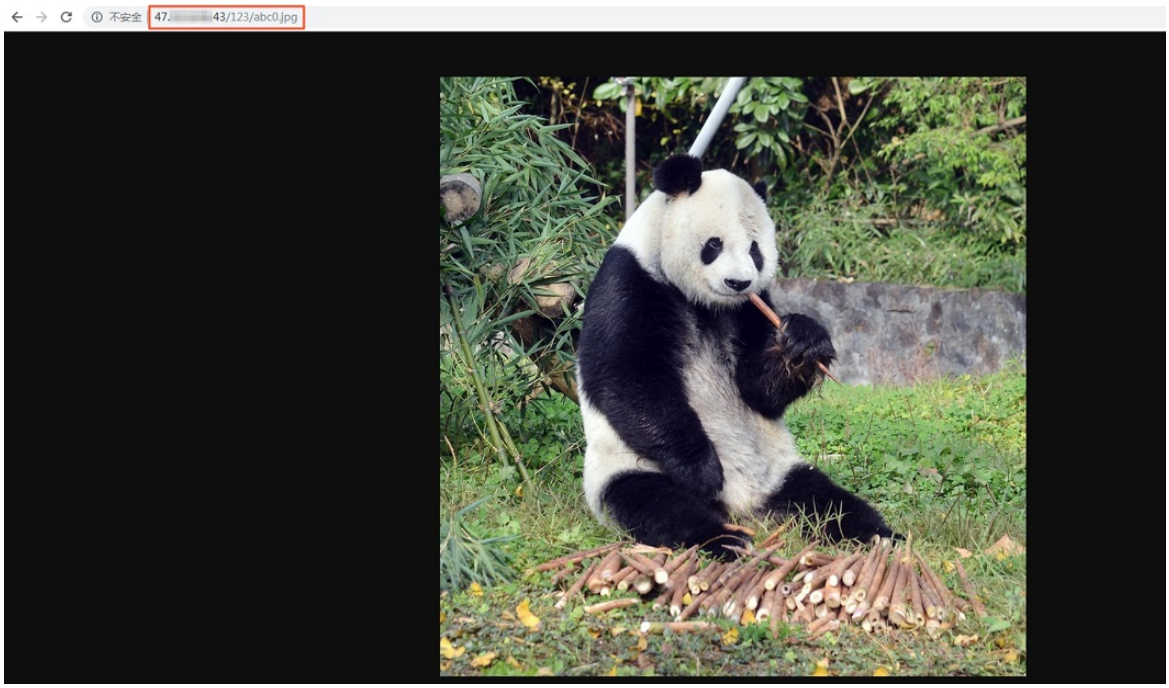
5. 进入Nginx主程序文件夹，启动Nginx。

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

6. 开放ECS实例的TCP 80端口。

   Nginx默认使用80端口，需在ECS的安全组配置中，允许用户访问TCP 80端口。配置方式请参见添加安全组规则。

7. 测试使用ECS外网地址加文件访问路径访问OSS资源。



如果访问的文件读写权限为私有，文件URL中还需要包含签名信息。详情请参见在URL中包含签名。

更多参考

- 基于Windows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

2.3.3. 基于Windows的ECS实例实现OSS反向代理

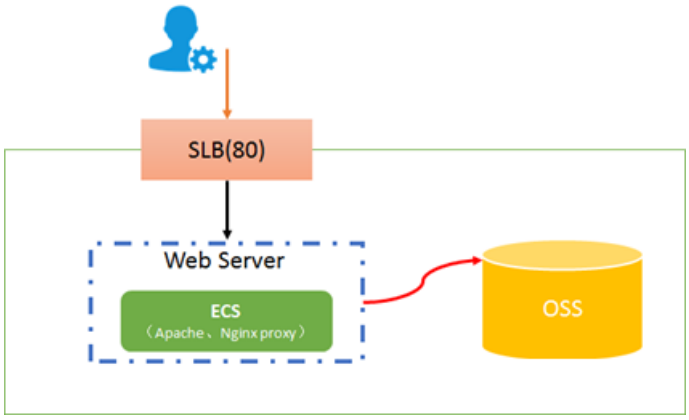
阿里云OSS的存储空间（Bucket）访问地址会随机变换，您可以通过在ECS实例上配置OSS的反向代理，实现通过固定IP地址访问OSS的存储空间。

背景信息

阿里云OSS为用户提供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，金融云的Bucket只能在金融云内部访问。

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. 创建一个和指定Bucket相同地域的Windows Server系统的ECS实例并登录。

   本文演示系统为Windows Server 2019数据中心版64位中文版系统，创建和登录过程请参见Windows系统实例快速入门。

2. 下载Nginx并解压。

   本文以Nginx-1.19.2版本为例，下载地址请参见Nginx。

3. 修改配置文件。

   i. 进入conf文件夹，使用记事本打开*nginx.conf*文件。

ii. 修改配置文件内容。

```
worker_processes  1;
events {
    worker_connections  1024;
}
http {
    include       mime.types;
    default_type  application/octet-stream;
    keepalive_timeout  65;
    server {
        listen       80;
        server_name  47.**.**.43;
        error_page   500 502 503 504  /50x.html;
        location = /50x.html {
            root   html;
        }
       location / {
            proxy_pass  http://bucketname.oss-cn-hangzhou-internal.aliyuncs.com;
            #proxy_set_header Host $host;
        }
   }
}
```

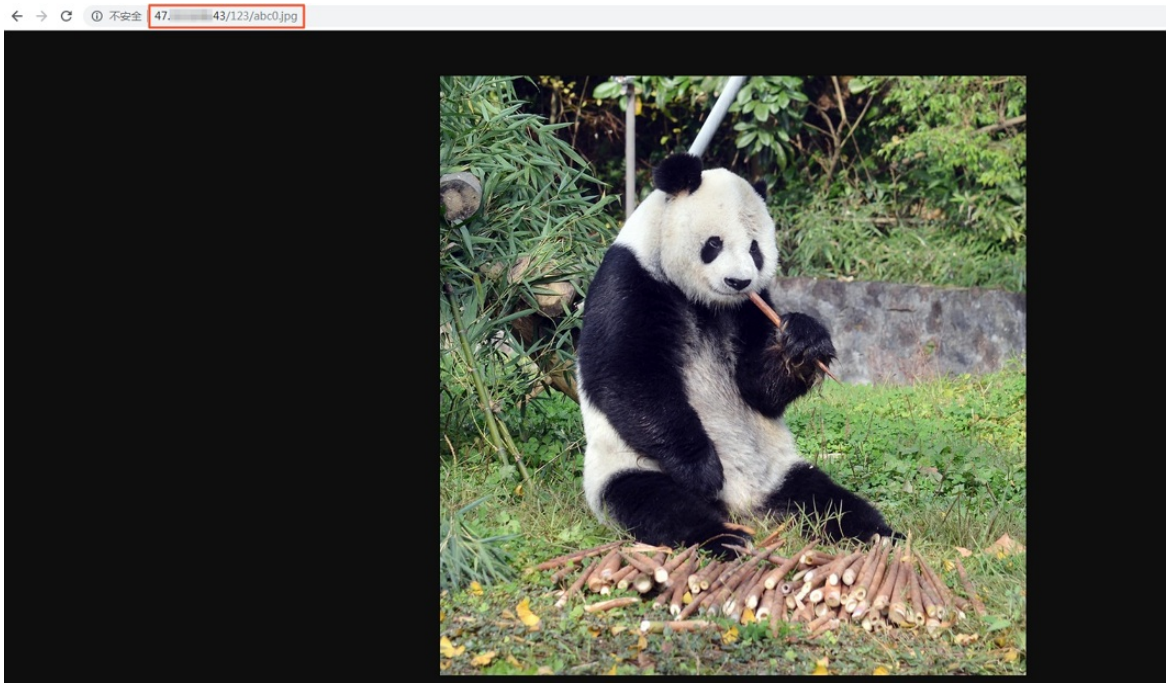
- server_name：对外提供反向代理服务的IP，即ECS实例的外网地址。
- proxy_pass：填写跳转的域名。
  - 当ECS实例与Bucket在同一地域时，填写目标Bucket的内网访问域名。访问域名介绍请参见OSS访问域名使用规则。
  - 当ECS实例与Bucket不在同一地域时，填写目标Bucket的外网访问域名。
  - 因OSS的安全设置，当使用默认域名通过浏览器访问OSS中的图片或网页文件时，会直接下载。所以，若您的用户需通过浏览器预览Bucket中的图片或网页文件，需为Bucket绑定自定义域名，并在此项中添加已绑定的域名。绑定自定义域名操作请参见绑定自定义域名。
- proxy_set_header Host $host：添加此项时，Nginx会在向OSS请求的时候，将host替换为ECS的访问地址。遇到以下情况时，您需要添加此项。
  - 遇到签名错误问题。
  - 如果您的域名已解析到ECS实例的外网上，且您的用户需要通过浏览器预览Bucket中的图片或网页文件。您可以将您的域名绑定到ECS实例代理的Bucket上，不配置CNAME。这种情况下，proxy_pass项可直接配置Bucket的内网或外网访问地址。绑定自定义域名操作请参见绑定自定义域名。

> 注意
>
> - 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。
> - 上述配置方式只能代理访问一个Bucket。

4. 返回Nginx主程序文件夹，双击nginx.exe启动Nginx。
5. 开放ECS实例的TCP 80端口。

   Nginx默认使用TCP 80端口，所以需在ECS的安全组配置中，允许用户访问TCP 80端口。配置方式请参见添加安全组规则。

6. 使用ECS外网地址加文件访问路径访问OSS资源。



更多参考

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

2.4. 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。

背景信息

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；若某些Part没有crc64值，则不返回整个Object的crc64值。例如某个Part在crc64上线之前就已经上传，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
import crcmod
import random
import string
do_crc64 = crcmod.mkCrcFun(0x142F0E1EBA9EA3693, initCrc=0, xorOut=0xffffffffffffffff, r
ev=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, os
s_crc64))
        return False
    else:
        print("{0} check crc64 ok.".format(msg))
        return True
def random_string(length):
    return ''.join(random.choice(string.ascii_lowercase) for i in range(length))
bucket = oss2.Bucket(oss2.Auth('yourAccessKeyId', 'yourAccessKeySecret'),
                     'https://oss-cn-hangzhou.aliyuncs.com', 'yourBucketName')
```

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(oss2.to_bytes(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(oss2.to_bytes(part_1)))
#检查上传的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(oss2.to_bytes(part_2)))
#检查上传的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(oss2.to_bytes(part_2), do_crc64(oss2.to_bytes(part_1))))
#检查最终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

3.数据湖

3.1. 阿里云生态

3.1.1. 通过SLS完成日志数据入湖OSS

数据湖支持在低存储成本的情况下，更好地满足数据长期存储、查询、分析、读取等需求。本文介绍通过日志服务SLS完成日志数据入湖的操作方式。

前提条件

已在日志服务Project所在的地域创建Bucket。具体操作，请参见创建存储空间。

背景信息

数据湖是一个集中式存储库，允许您以任意规模存储所有结构化、半结构化以及非结构化数据。这些数据包括来源于关系型数据库中的结构化数据（行和列）、半结构化数据（例如CSV、日志、XML、JSON等）、非结构化数据（例如email、文档、PDF等）和二进制数据（例如图像、音频、视频等）。

日志服务集成了数据投递到OSS的功能，以Serverless的方式助力日志数据入湖，支持丰富的日志数据源，一站式的分析查询，多种投递格式，压缩类型，无需运维。

步骤一：选择投递的数据源

1. 登录OSS管理控制台。
2. 在左侧导航栏，选择**数据导入 > 日志导入 > 前往控制台**。
3. 选择投递方式。
   - 如果您在SLS中已采集数据：
     a. 选择**SLS现有数据投递搭到OSS**。
     b. 在**选择已存储数据的Project/Logstore创建OSS投递任务**对话框，下拉选择已采集数据所在的**项目Project**和**日志库Logstore**。
   - 如果您在SLS中未采集数据：

     选择**数据采集并投递OSS**，完成数据采集后再将数据投递到OSS。

     关于采集数据的更多信息，请参见数据采集概述。
4. 选择是否对数据进行加工。
   - 如果选择**投递OSS**，则跳过步骤二，直接执行步骤步骤三。
   - 如果选择**加工后投递OSS**，需要执行步骤二。

     当您对日志数据有以下使用需求时，可以选择对日志数据加工后再投递到OSS：
     - 数据规整：针对混乱格式的日志进行字段提取、格式转换，获取结构化数据以支持后续的流处理、数据仓库计算。
     - 数据富化：对日志（例如订单日志）和维表（例如用户信息表）进行字段连接（JOIN），为日志添加更多维度的信息，用于数据分析。
     - 数据流转：通过跨地域加速功能将海外地域的日志传输到中心地域，实现全球日志集中化管理。
     - 数据脱敏：对数据中包含的密码、手机号、地址等敏感信息进行脱敏。
     - 数据过滤：过滤出关键服务的日志，用于重点分析。

（可选）步骤二：加工数据

以下以过滤OSS访问日志为例，介绍过滤后仅保留OSS访问日志中GetObject方法的操作步骤。

1. 单击目标Logstore左侧的>，选择**数据处理 > 加工**。
2. 在页面右上角，选择数据的时间范围。

   请确保在**原始日志**页签中有日志数据。
3. 在编辑框中，输入以下数据加工语句。

```
e_keep(e_search("operation=GetObject"))
```

4. 预览数据。
   i. 单击**快速**。

      日志服务支持快速预览和高级预览。更多信息，请参见预览调试概述。
   ii. 单击**预览数据**，查看预览结果。

   - 如果加工语句错误或者权限配置错误，导致数据加工失败，请根据页面提示处理。
   - 如果确认数据加工结果无误，请执行步骤。
5. 创建数据加工任务。
   i. 单击**保存数据加工**。

ii. 在**创建数据加工规则**页面，按如下说明配置相关参数，其他参数保留默认配置，然后单击**确定**。

参数	说明
规则名称	定义数据加工规则的名称。
授权方式	选择**默认角色**，表示授予数据加工任务使用阿里云系统角色AliyunLogETLRole来读取源Logstore中的数据。如果您还没有生成默认角色，需单击**授权系统角色AliyunLogETLRole**，并根据页面提示完成授权。更多信息，请参见通过默认角色访问数据。
存储目标	
目标名称	定义存储目标的名称。
目标Project	定义用于存储数据加工结果的目标Project名称。
目标库	定义用于存储数据加工结果的目标Logstore名称。
授权方式	选择**默认角色**，即授予数据加工任务使用阿里云系统角色AliyunLogETLRole将数据加工结果写入目标Logstore。如果您还没有生成默认角色，需单击**授权系统角色AliyunLogETLRole**，并根据页面提示完成授权。更多信息，请参见通过默认角色访问数据。
加工范围	
时间范围	指定数据加工任务的时间范围，详细说明如下： **说明** 此处的时间范围依赖日志的接收时间。 ■ **所有**：从Logstore接收到第一条日志的时间点开始数据加工作业，直到加工作业被手动停止。 ■ **某时间开始**：指定数据加工作业的开始时间，从该时间点开始加工，直到加工作业被手动停止。 ■ **特定时间范围**：指定数据加工作业的起止时间，加工作业执行到指定时间后自动停止。

关于创建数据加工任务涉及的各参数详细说明，请参见创建数据加工作业。

步骤三：投递数据到OSS

日志服务投递数据到OSS为同地域投递，即日志服务Project所在的地域和OSS Bucket所在地域相同。您可以选择新版或旧版投递数据到OSS的方式完成日志数据入湖。旧版投递数据到OSS支持日志服务所有已开服地域，而新版投递数据到OSS（新版）仅支持以下地域：

注意 目前只支持华东1（杭州）、华东2（上海）、华北2（北京）、华北3（张家口）、西南1（成都）、华南1（深圳）、中国（香港）。

关于新旧版投递数据到OSS的更多信息，请参见OSS投递新旧版本对比。

1. 创建OSS投递作业。

在**OSS投递功能**面板，按如下说明配置相关参数，其他参数保留默认配置，然后单击**确定**。

参数	说明
OSS投递名称	投递作业的名称。
OSS Bucket	OSS Bucket名称。 **注意** 必须是已存在的OSS Bucket名称，且该OSS Bucket与日志服务Project位于相同地域。
文件投递目录	指定OSS Bucket中的目录。目录名不能以正斜线（/）或者反斜线（\）开头。如果未指定该目录，则生成的文件将保存在Bucket的根目录下。 创建OSS投递作业后，Logstore中的数据将投递到目标OSS Bucket的此目录中。
文件后缀	指定生成文件的后缀。如果未设置文件后缀，则日志服务会根据存储格式和压缩类型自动生成后缀，例如suffix。
写OSS RAM角色	选择**默认角色**，表示授权OSS投递作业使用阿里云系统角色AliyunLogDefaultRole将数据写入到OSS Bucket中。如果您还没有生成默认角色，需手动输入AliyunLogDefaultRole的ARN。如何获取ARN，请参见通过默认角色访问数据。
读Logstore RAM角色	选择**默认角色**，表示授权OSS投递作业使用阿里云系统角色AliyunLogDefaultRole来读取Logstore中的数据。如果您还没有生成默认角色，需手动输入AliyunLogDefaultRole的ARN。如何获取ARN，请参见通过默认角色访问数据。
起始时间	选择投递作业开始拉取Logstore中数据的起始时间。

关于创建投递作业涉及的各参数详细说明，请参见创建OSS投递作业（新版）。

2. （可选）查看生成的文件。

将日志投递到OSS成功后，您可以通过OSS控制台、API、SDK或ossutil查看生成的文件。文件格式为：

```
oss://OSS-BUCKET/OSS-PREFIX/PARTITION-FORMAT_RANDOM-ID
```

参数说明如下表所示：

参数	说明	示例值
OSS-BUCKET	OSS Bucket名称。	examplebucket
OSS-PREFIX	文件投递目录。	exampledir
PARTITION-FORMAT	分区格式，通过strptime API计算得到的投递作业创建时间。	2022/01/20/19/50_1484913043351525351
RANDOM-ID	随机生成的一次投递行为的唯一标识。	2850008

结合以上示例值，则投递作业创建时间为2022/01/20 19:50:43生成的OSS文件路径为：

```
oss://examplebucket/exampledir/2022/01/20/19/50_1484913043351525351_2850008.suffix
```

> **说明** OSS文件路径以投递作业的创建时间动态生成。假设您在2022-01-20 00:00:00创建投递作业，5分钟投递一次数据到OSS Bucket，则此次投递任务投递的是2022-01-19 23:55后写入日志服务的数据。由于写入日志服务的数据可能存在延时，因此当您希望分析2022-01-19全天日志，除了查看*2022/01/19*目录下的全部Object以外，还需要检查*2022/01/20/00/*目录下前十分钟的Object是否包含2022-01-22的日志。

3.1.2. 在EMR Hive或Spark中访问OSS-HDFS

EMR-3.40及后续版本或EMR-5.6.0及后续版本的集群，支持OSS-HDFS（JindoFS服务）作为数据存储，提供缓存加速服务和Ranger鉴权功能，使得在Hive或Spark等大数据ETL场景将获得更好的性能和HDFS平迁能力。本文为您介绍E-MapReduce（简称EMR）Hive或Spark如何操作OSS-HDFS。

背景信息

OSS-HDFS服务是一款云原生数据湖存储产品，基于统一的元数据管理能力，在完全兼容HDFS文件系统接口的同时，提供充分的POSIX能力支持，能更好的满足大数据和AI领域丰富多样的数据湖计算场景，详细信息请参见OSS-HDFS服务概述。

前提条件

已在EMR上创建EMR-3.40及后续版本或EMR 5.6.0及后续版本的集群，具体操作请参见创建集群。

开通并授权访问OSS-HDFS服务

- 创建Bucket时开通并授权访问OSS-HDFS服务

具体操作，请参见创建Bucket。

- 对已创建的Bucket开通并授权访问OSS-HDFS服务

i. 完成RAM授权

a. 登录OSS管理控制台。

b. 单击左侧导航栏的**Bucket列表**，然后单击待开通OSS-HDFS服务的Bucket名称。

c. 在左侧导航栏，选择**数据湖管理 > HDFS服务**。

d. 在**HDFS服务**页签，单击**前往授权**，然后按照页面指引完成授权操作。

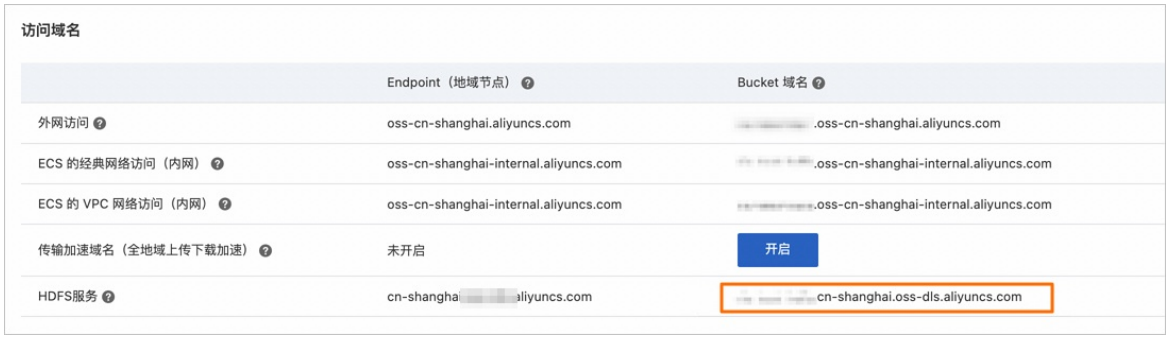
ii. 开通HDFS服务

a. 在**HDFS服务**页签，单击**开通HDFS服务**。

b. 在弹出的对话框，单击**确定**。

步骤二：获取OSS-HDFS服务域名

1. 单击左侧导航栏的**Bucket列表**，然后单击已开通OSS-HDFS服务的Bucket名称。
2. 在左侧导航栏，单击**概览**。
3. 在**访问域名**区域，记录HDFS服务的Bucket域名。



步骤三：在EMR集群中使用OSS-HDFS

> **说明** 本示例以Hive操作OSS-HDFS为例介绍。您也可以参照此方式使用Spark操作OSS-HDFS。

1. 登录集群，具体操作请参见登录集群。
2. 创建指向OSS-HDFS的Hive表。
   i. 执行以下命令，进入Hive命令行。

   ```
   hive
   ```

   ii. 执行以下命令，创建指向OSS-HDFS的数据库。

   ```
   CREATE DATABASE if not exists dw LOCATION 'oss://<yourBucketName>.<yourBucketEndpoint>/<path>';
   ```

   > **说明**
   > - 上述命令中的 `dw` 为数据库名，`<path>` 为任意路径，`<yourBucketName>.<yourBucketEndpoint>` 为步骤二获取到的HDFS服务的域名。
   > - 本示例使用OSS-HDFS的域名作为路径的前缀。如果您希望只使用Bucket名称来指向OSS-HDFS，则可以配置Bucket级别的Endpoint或全局Endpoint，具体操作请参见附录：配置Endpoint的其他方式。

   iii. 执行以下命令，使用新创建的数据库。

   ```
   use dw;
   ```

   iv. 执行以下命令，在新建的数据库下创建表。

   ```
   CREATE TABLE IF NOT EXISTS employee(eid int, name String,salary String,destination String)
   COMMENT 'Employee details';
   ```

   v. 执行以下命令，进入Hive命令行。

   ```
   hive
   ```

3. 向表中插入数据。

   使用INSERT INTO语句向表写入数据，该语句会产生MapReduce作业。

   ```
   INSERT INTO employee(eid, name, salary, destination) values(1, 'liu hua', '100.0', '');
   ```

4. 验证表数据。

   ```
   SELECT * FROM employee WHERE eid = 1;
   ```

   返回信息中会包含插入的数据。

   ```
   OK
   1       liu hua 100.0
   Time taken: 12.379 seconds, Fetched: 1 row(s)
   ```

（可选）为EMR集群授权

当前EMR的实例角色AliyunECSInstanceForEMRRole，其关联的AliyunECSInstanceForEMRRolePolicy默认已经添加了oss:PostDataLakeStorageFileOperation策略，因此默认情况下，您无需重新对EMR授权访问OSS-HDFS服务。

如果用于访问OSS-HDFS服务的EMR集群未使用默认的AliyunECSInstanceForEMRRole实例角色，则需要为该EMR集群授权。具体步骤，请参见角色授权。

3.1.3. 通过EMR集群配置Ranger鉴权方案

Apache Ranger提供集中式的权限管理框架，支持对Hadoop生态中的多个组件进行细粒度的权限控制。本文介绍如何集成Ranger以及如何配置OSS-HDFS服务的访问权限。

前提条件

- 已创建EMR-5.6.0及以上版本的高安全集群。

  创建集群过程中，选择Ranger服务、并选中Kerberos集群模式。关于创建集群的具体步骤，请参见创建集群。

- 已开通并授权访问OSS-HDFS服务。具体步骤，请参见开通并授权访问OSS-HDFS服务。

集成Ranger

1. 进入集群详情页面。
   i. 登录阿里云E-MapReduce控制台。
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   iii. 单击上方的**集群管理**页签。
   iv. 在**集群管理**页面，单击相应集群所在行的**详情**。
2. 启用OSS。
   i. 在左侧导航栏中，选择**集群服务 > RANGER**。
   ii. 在RANGER服务页面，选择**操作 > 启用OSS**。
   iii. 在**执行集群操作**对话框中，填写**执行原因**，单击**确定**。
   iv. 在**确认**对话框，单击**确定**。
3. 部署客户端配置。
   i. 在左侧导航栏中，选择**集群服务 > HDFS**。
   ii. 在HDFS服务页面，单击**配置**，然后单击右上角的**部署客户端配置**。
   iii. 在**执行集群操作**中，输入执行原因，单击**确定**。
   iv. 在**确认**对话框中，单击**确定**。

   您可以单击上方的**查看操作历史**，查看执行状态和进度。

4. 重启Jindofsx Namespace Service。
   i. 在**JindoData**服务页面的右上角，选择**操作 > 重启Jindofsx Namespace Service**。
   ii. 在**执行集群操作**对话框中，输入执行原因，单击**确定**。
   iii. 在弹出的**确认**对话框中，单击**确定**。
5. 重启HiveServer2。
   i. 在左侧导航栏中，选择**集群服务 > Hive**。
   ii. 在Hive服务页面的右上角，选择**操作 > 重启HiveServer2**。
   iii. 在**执行集群操作**对话框中，输入执行原因，单击**确定**。
   iv. 在弹出的**确认**对话框中，单击**确定**。
6. 创建用户Principal。
   i. 通过SSH方式连接集群的emr-header-1节点，详情请参见登录集群。
   ii. 执行如下命令，进入Kerberos的admin工具。

   ```
   sh /usr/lib/has-current/bin/admin-local.sh /etc/ecm/has-conf -k /etc/ecm/has-conf/admin.keytab
   ```

   iii. 执行如下命令，创建用户名为test的Principal。

   本示例密码设置为123456。

   ```
   addprinc -pw 123456 test
   ```

   > **说明** 需要记录用户名和密码，在创建TGT时会用到。如果您不想记录用户名和密码，则可以执行下一步，将Principal的用户名和密码导入到keytab文件中。

   iv. （可选）执行如下命令，生成keytab文件。

   ```
   ktadd -k /root/test.keytab test
   ```

   执行`quit`命令，可以退出Kerberos的admin工具。
7. 创建TGT。

   创建TGT的机器，可以是任意一台需要运行Hive Client的机器。
   i. 使用root用户执行以下命令，创建test用户。

   ```
   useradd test
   ```

   ii. 执行以下命令，切换为test用户。

   ```
   su test
   ```

iii. 生成TGT。

- 方式一：使用用户名和密码方式，创建TGT。

  执行 `kinit` 命令，回车后输入test的密码123456。

```
[root@emr-header-1 ~]# su test
[test@emr-header-1 root]$ kinit
Password for test@EMR.238075.COM:
[test@emr-header-1 root]$
```

- 方式二：使用keytab文件，创建TGT。

  在步骤6中的*test.keytab*文件，已经保存在emr-header-1机器的*/root/*目录下，需要先使用 `cp /root/test.keytab /home/test/` 命令拷贝到当前机器的*/home/test/*目录下，再执行以下命令创建TGT。

```
kinit -kt /home/test/test.keytab test
```

iv. 查看TGT。

使用 `klist` 命令，返回如下信息。

```
Ticket cache: FILE:/tmp/krb5cc_1012
Default principal: test@EMR.23****.COM
Valid starting       Expires              Service principal
07/24/2021 13:20:44  07/25/2021 13:20:44  krbtgt/EMR.23****.COM@EMR.23****.COM
        renew until 07/25/2021 13:20:44
```

配置OSS-HDFS访问权限

配置test用户拥有访问*oss://examplebucket/dir/*目录的访问权限。

1. 进入Ranger UI页面，详情请参见概述。
2. 在Ranger UI页面，单击已有的**emr-oss**。

3. 配置*dir/*目录的Execute权限。
   i. 单击右上角的**Add New Policy**。

ii. 在Edit Policy页面，配置下表参数。

参数	描述
Policy Name	策略名称，可以自定义。
Path	填写为*examplebucket/dir*。
Select User	指定添加此策略的用户。 本示例设置为test用户。
Permissions	选择授予的权限。 本示例设置访问权限为Execute。

iii. 单击Add。

4. 访问OSS-HDFS。

i. 通过SSH方式连接集群的emr-header-1节点，详情请参见登录集群。

ii. 执行以下命令，切换为本文示例创建的test用户。

```
su test
```

iii. 执行以下命令，访问OSS-HDFS服务。

```
hadoop fs -ls oss://examplebucket/dir/
```

当您访问Ranger没有授权的路径时，提示以下错误信息。

```
org.apache.hadoop.security.AccessControlException: Permission denied: user=test, access=READ_EXECUTE, resourcePath="examplebucket/dir/"
```

3.1.4. 使用JindoFuse访问OSS-HDFS服务

阿里云OSS-HDFS服务（JindoFS服务）通过JindoFuse提供POSIX支持。JindoFuse可以把JindoFS服务上的文件挂载到本地文件系统中，让您能够像操作本地文件系统一样操作JindoFS服务中的文件。

步骤一：安装Fuse3依赖

- CentOS

```
yum install -y fuse3 fuse3-devel
```

- Debian

```
apt install -y fuse3 libfuse3-dev
```

步骤二：配置客户端

1. 配置目录。

解压下载的安装包，以安装包内容解压在*/usr/lib/jindosdk-4.4.0/conf*目录为例：

```
export JINDOSDK_CONF_DIR=/usr/lib/jindosdk-4.4.0/conf
```

2. 配置文件。

   使用 `.ini` 风格配置文件，配置文件的文件名为*jindosdk.cfg*，示例如下：

```
[common]
logger.dir = /tmp/fuse-log
[jindosdk]
# 已开启HDFS服务的Bucket对应的Endpoint。以华东1（杭州）为例，填写为cn-hangzhou.oss-dls.aliyuncs.com。
fs.oss.endpoint = <your_endpoint>
# 用于访问JindoFS服务的AccessKey ID和AccessKey Secret。阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维，请登录RAM控制台创建RAM用户。
fs.oss.accessKeyId = <your_key_id>
fs.oss.accessKeySecret = <your_key_secret>
```

步骤三：挂载JindoFuse

1. 创建挂载点。

```
mkdir -p <mount-point>
```

2. 挂载JindoFuse。

```
jindo-fuse <mount_point> -ouri=[<oss_path>]
```

   *-our*需配置为待映射的dls路径，路径可以为Bucket根目录或者子目录。执行该命令会启动后台的守护进程，将指定的<oss_path>挂载到本地文件系统的<mount_point>。

   关于挂载Fuse过程中可以配置的挂载选项的更多信息，请参见挂载选项 。

步骤四：访问JindoFuse

例如，当您将JindoFuse挂载到本地路径*/mnt/jindodls/*后，通过以下命令执行JindoFuse的基础操作：

- 创建目录

```
mkdir /mnt/oss/dir1
```

- 列举*/mnt/oss/*下的所有目录：

```
ls /mnt/oss/
```

- 写入文件：

```
echo "hello world" > /mnt/oss/dir1/hello.txt
```

- 读取文件：

```
cat /mnt/oss/dir1/hello.txt
```

- 删除目录：

```
rm -rf /mnt/oss/dir1/
```

步骤五：取消挂载JindoFuse

您可以使用以下两种方式取消挂载JindoFuse：

方式一：手动取消挂载JindoFuse

```
umount <mount_point>
```

方式二：自动取消挂载JindoFuse

```
-oauto_unmount
```

使用以上命令支持通过 `killall -9 jindo-fuse` 发送SIGINT到jindo-fuse进程，进程退出前会自动取消挂载JindoFuse。

常见问题

如何排查JindoFuse错误？

与JindoSDK调用API过程中可获取到详细的错误信息ErrorMsg不同，JindoFuse只显示操作系统预设的错误信息：

```
ls: /mnt/oss/: Input/output error
```

如果您需要定位具体的错误原因，请前往JindoSDK配置项*logger.dir*指定路径下的*jindosdk.log*文件。如下为使用JindoFuse过程中常见的鉴权错误信息：

```
EMMDD HH:mm:ss jindofs_connectivity.cpp:13] Please check your Endpoint/Bucket/RoleArn.
Failed test connectivity, operation: mkdir, errMsg: [RequestId]: 618B8183343EA53531C62B74 [
HostId]: oss-cn-shanghai-internal.aliyuncs.com [ErrorMessage]: [E1010]HTTP/1.1 403 Forbidde
n ...
```

收到以上报错信息后，请自行排查Endpoint、Bucket以及RoleArn配置信息是否正确。关于Endpoint、Bucket、RoleArn的配置详情，请参OSS-HDFS服务快速入门。

如果遇到程序类报错，请提交工单申请处理。

附录一：支持特性

目前JindoFuse支持以下POSIX API:

特性	说明
getattr()	查询文件属性，类似**ls**。
mkdir()	创建目录，类似**mkdir**。
rmdir()	删除目录，类似**rm -rf**。
unlink()	删除文件，类似**unlink**。
rename()	重命名文件或目录，类似**mv**。
read()	顺序读取。
pread()	随机读。
write()	顺序写。

特性	说明
pwrite()	随机写。
flush()	刷新内存到内核缓冲区。
fsync()	刷新内存到磁盘。
release()	关闭文件。
readdir()	读取目录。
create()	创建文件。
open() O_APPEND	通过追加写的方式打开文件。
open() O_TRUNC	通过覆盖写的方式打开文件。
ftruncate()	截断已打开的文件。
truncate()	截断未打开的文件，类似**truncate -s**。
lseek()	指定打开文件中的读写位置。
chmod()	修改文件权限，类似**chmod**。
access()	查询文件权限。
utimes()	修改文件的存储时间和更改时间。
setxattr()	修改文件的xattr属性。
getxattr()	获取文件的xattr属性。
listxattr()	列举文件的xattr属性。
removexattr()	删除文件的xattr属性。
lock()	支持posix锁，类似**fcntl**。
fallocate()	为文件预分配物理空间。
symlink()	创建软连接，目前仅支持在OSS-HDFS服务中使用，且不支持缓存加速。
readlink()	读取软连接。

附录二：挂载选项

挂载JindoFuse过程中可以配置的挂载选项如下表所示：

名称	是否必选	参数说明	使用示例

名称	是否必选	参数说明	使用示例
uri	是	配置需要映射的dls路径。路径可以是Bucket根目录-*ouri=oss://bucket.endpoint/*，也可以是子目录-*ouri=oss://bucket.endpoint/subdir*。	-ouri=oss://examplebucket.cn-beijing.oss-dls.aliyuncs.com/
f	否	启动进程。默认使用守护进程方式后台启动。使用该参数时，推荐开启终端日志。	-f
d	否	使用Debug模式，在前台启动进程。使用该参数时，推荐开启终端日志。	-d
auto_unmount	否	fuse进程退出后自动卸载挂载点。	-oauto_unmount
ro	否	只读挂载，启用参数后不允许写操作。	-oro
direct_io	否	开启该选项后，读写文件可以绕过页高速缓冲存储器（Page Cache）。	-odirect_io
kernel_cache	否	开启后该选项后，通过内核缓存优化读性能。	-okernel_cache
auto_cache	否	默认开启自动缓存。与`kernel_cache`不同的是，如果文件大小或修改时间发生变化，则缓存失效。	-oauto_cache
entry_timeout	否	文件名读取成功缓存保留时间，单位为秒。该选项用于性能优化。0表示不缓存。默认值为0.1。	-oentry_timeout=60
attr_timeout	否	文件属性缓存保留时间，单位为秒。该选项用于性能优化。0表示不缓存。默认值为0.1。	-oattr_timeout=60
negative_timeout	否	文件名读取失败缓存保留时间，单位为秒。该选项用于性能优化。0表示不缓存。默认值为0.1。	-onegative_timeout=0
jindo_entry_size	否	目录条目缓存数量，用于优化readdir性能。0表示不缓存。默认值为5000。	-ojindo_entry_size=5000
jindo_attr_size	否	文件属性缓存数量，用于优化getattr性能。0表示不缓存。默认值为50000。	-ojindo_attr_sizet=50000
max_idle_threads	否	最大空闲线程数。默认值为10。	-omax_idle_threads=10

名称	是否必选	参数说明	使用示例
metrics_port	否	开启HTTP端口，用于输出metrics，例如*http://localhost:9090/brpc_metrics*。默认值为9090。	-ometrics_port=9090
enable_pread	否	使用pread接口读取文件。	-oenable_pread

附录三：配置选项

配置项	配置节点	说明
logger.dir	common	日志目录。默认值为*/tmp/jindodata-log*。
logger.sync	common	输出日志的方式。取值如下： • *true*：同步输出日志。 • *false*（默认值）：异步输出日志。
logger.consolelogger	common	是否打印日志。取值如下： • *true*：打印日志到终端。 • *false*（默认值）：不打印日志。
logger.level	common	输出大于等于该等级的日志。 • 开启终端日志 日志等级范围为0~6，日志级别对应关系如下： ○ *0*：TRACE ○ *1*：DEBUG ○ *2*（默认值）：INFO ○ *3*：WARN ○ *4*：ERROR ○ *5*：CRITICAL ○ *6*：OFF • 关闭终端日志 使用文件日志时，如果日志等级<=1，表示WARN。日志等级>1，表示INFO。
logger.verbose	common	输出大于等于该等级的VERBOSE日志，等级范围为0~99，默认值为0，0表示不输出。
logger.cleaner.enable	common	是否开启日志清理。取值如下： • *true*：开启日志清理。 • *false*（默认值）：不开启日志清理。
fs.oss.endpoint	jindosdk	用于访问JindoFS服务的地址，例如`cn-hangzhou.oss-dls.aliyuncs.com`

配置项	配置节点	说明
fs.oss.accessKeyId	jindosdk	用于访问JindoFS服务的AccessKey ID。
fs.oss.accessKeySecret	jindosdk	用于访问JindoFS服务的AccessKey Secret。

3.1.5. 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。

前提条件

- 已开通OSS服务，并已创建Bucket。
  - 开通OSS服务请参见开通OSS服务。
  - 创建Bucket请参见创建Bucket。
- 已开通MaxCompute服务，并已授权MaxCompute访问OSS。
  - 开通MaxCompute服务请参见开通MaxCompute和DataWorks。
  - 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。

背景信息

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. 将物联网数据上传到OSS。具体操作，请参见上传文件。

   您可以使用任何数据集执行测试。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*。

2. 创建MaxCompute Project。

   操作步骤请参见创建MaxCompute Project。

3. 通过MaxCompute创建外部表。

   操作步骤请参考创建表，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/';
```

4. 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。查询步骤请参见运行SQL命令并导出结果数据。

   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_external 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，请参见概述。

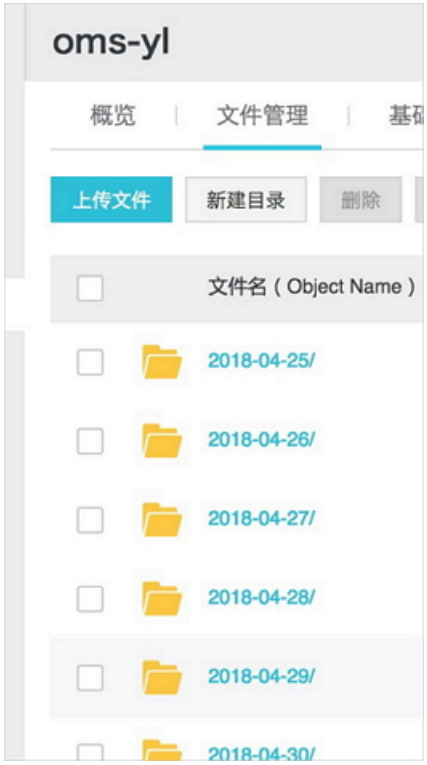
操作示例：阿里云产品消费账单分析

1. 创建MaxCompute Project。具体操作，请参见创建MaxCompute Project。

2. 在阿里云控制台单击**费用**，之后单击**返回旧版**。
3. 单击**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*。

   如果您之前未开通过此服务，您需要单击**进入授权**，按流程完成授权后再配置。

   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。



4. 安装MaxCompute客户端（odpscmd），并下载自定义代码。
5. 运行**add jar odps-udf-example-0.30.0-SNAPSHOT-jar-with-dependencies.jar**命令，将自定义代码编译打包，并上传到MaxCompute。
6. 通过MaxCompute创建外部表。

   详细步骤请参考创建表。以创建05月04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 S
                                                                  的类名
```

```
torageHandler 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。
```

7. 通过MaxCompute查询外部表。

   查询步骤可参见运行SQL命令并导出结果数据。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)';
```

3.1.6. 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。

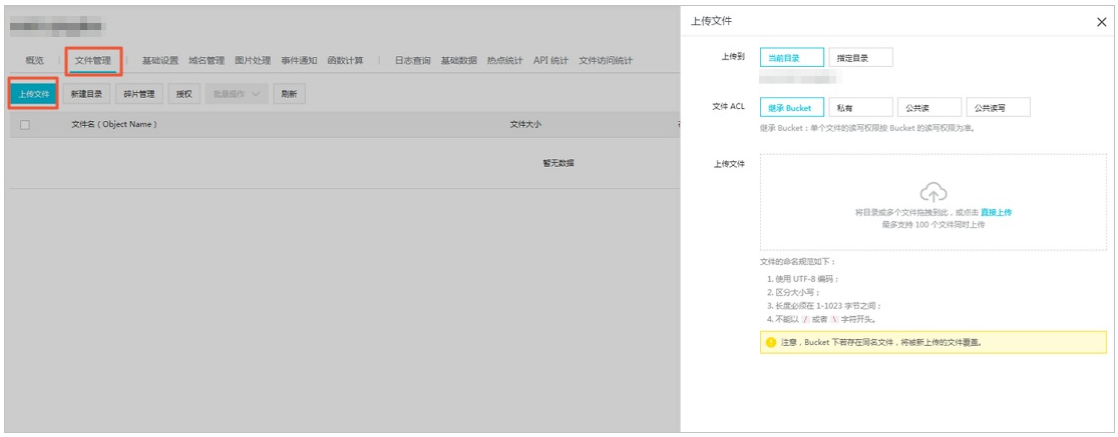
操作步骤

1. 数据导入OSS
   i. 新建一个名为“oss-pai-sample”的Bucket。
   ii. 记录其Endpoint为 oss-cn-shanghai.aliyuncs.com。

iii. 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。

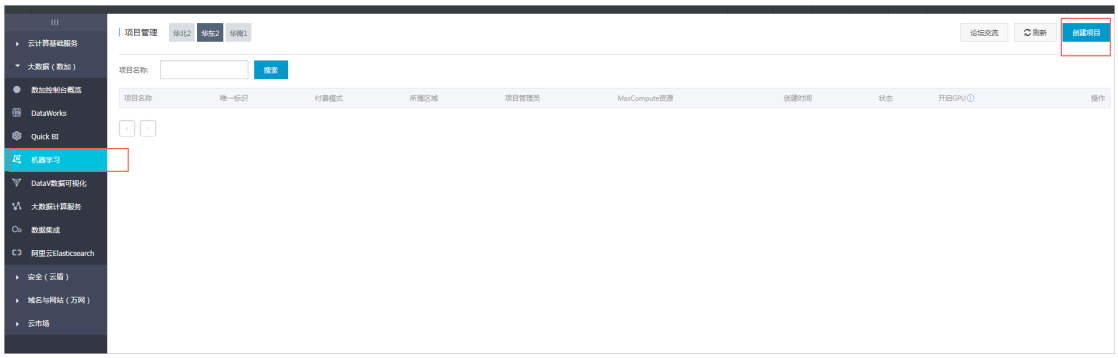
iv. 单击Bucket名称（oss-pai-sample），然后依次单击**文件管理 > 上传文件**，将订单数据Sample_superstore.csv上传至OSS。



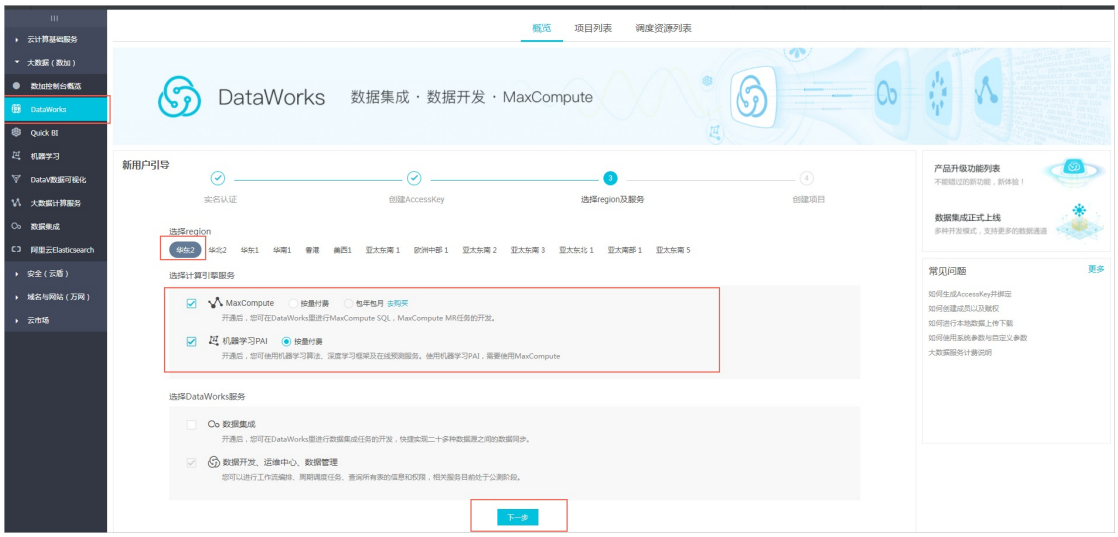
v. 上传成功，界面如下图所示：

2. 创建机器学习项目

i. 在控制台页面左侧选择**机器学习**，单击右上角的**创建项目**。



ii. 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**。

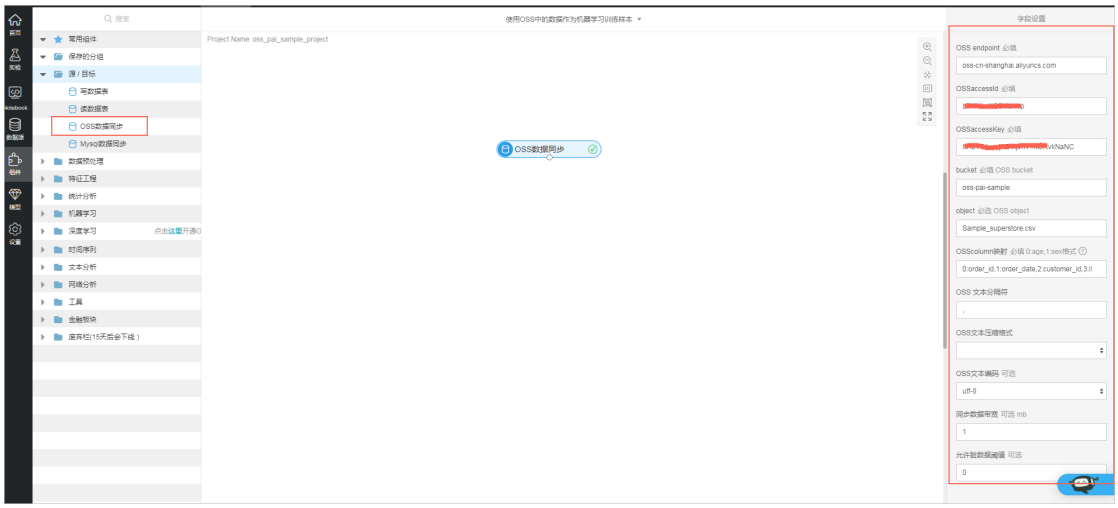


iii. 项目创建成功后，开通服务列中会显示MaxCompute和机器学习PAI两个图标，如下图所示：

iv. 回到机器学习页面，点击**进入机器学习**。

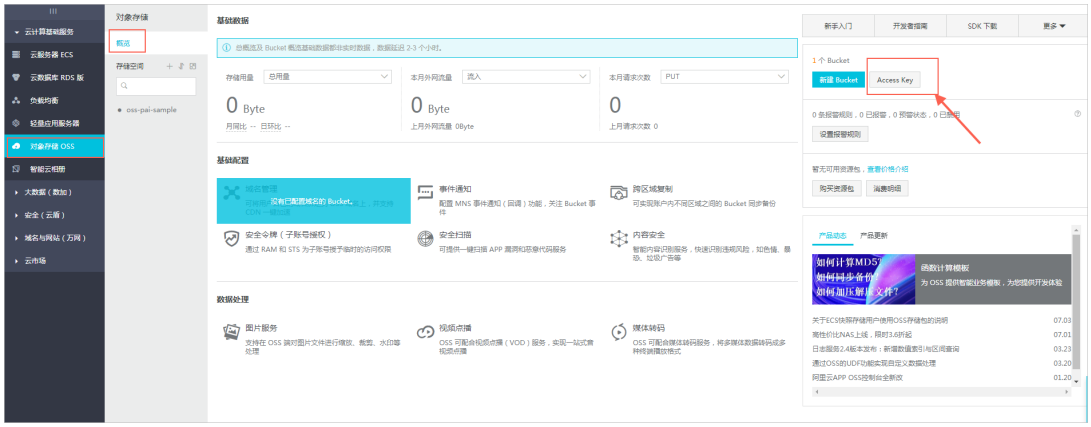
3. 连通OSS与PAI。

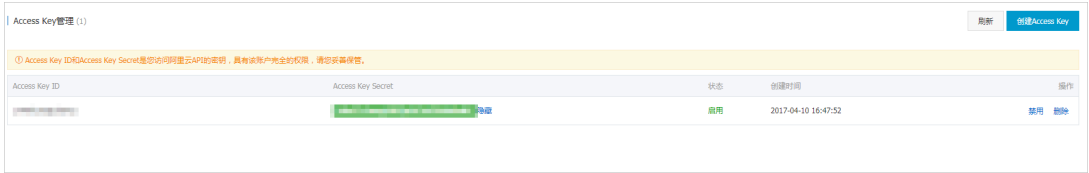
i. 在机器学习界面左侧选择**组件**，并将**OSS数据同步**组件拖拽至画布。



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：





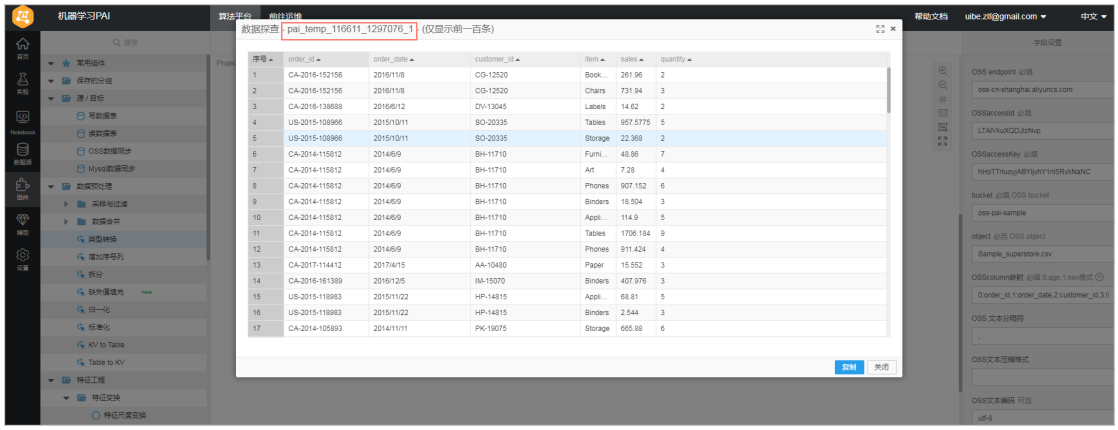
- OSSbucket 和 object 分别为 oss-pai-sample 和 Sample_superstore.csv。

- 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

ii. 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：



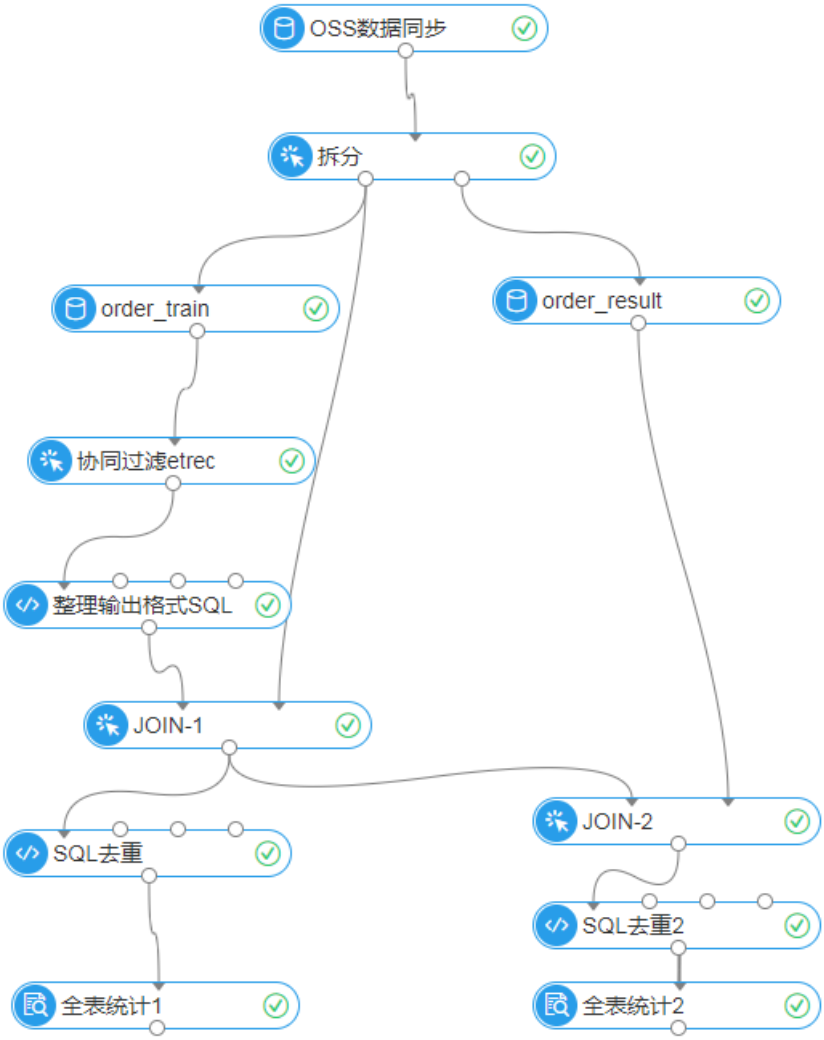
此时OSS中的csv文件已经在MaxCompute中生成一张临时表：pai_temp_116611_1297076_1

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。

3.1.7. 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。

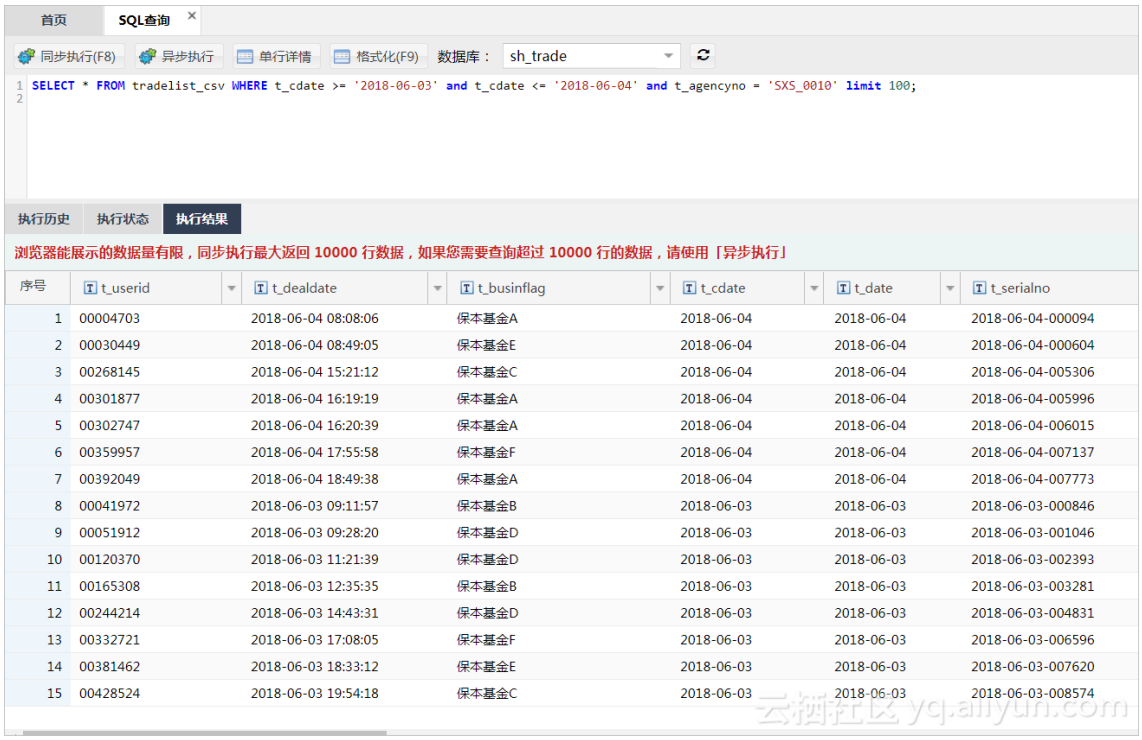
SQL查询（同步执行）

- 查询交易机构SXS_0010，在0603至0604的100条交易记录
  i. 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;
```

  ii. 显示执行结果

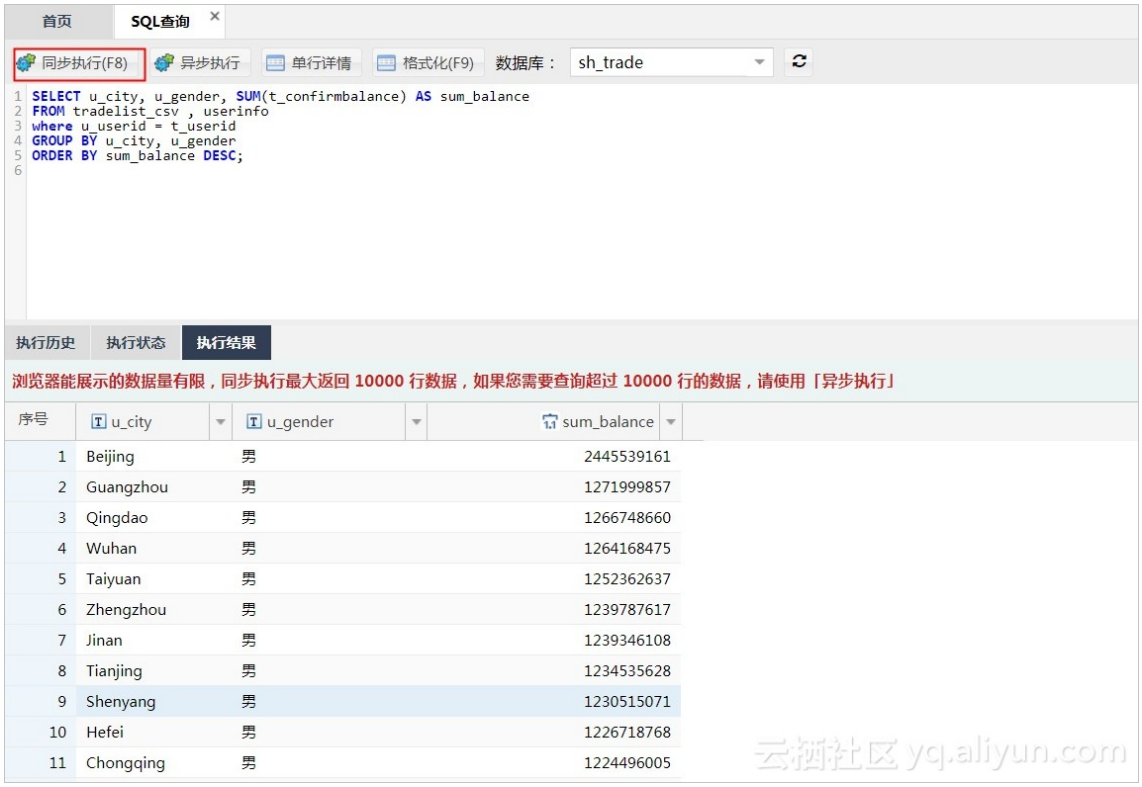


序号	t_userid	t_dealdate	t_businflag	t_cdate	t_date	t_serialno
1	00004703	2018-06-04 08:08:06	保本基金A	2018-06-04	2018-06-04	2018-06-04-000094
2	00030449	2018-06-04 08:49:05	保本基金E	2018-06-04	2018-06-04	2018-06-04-000604
3	00268145	2018-06-04 15:21:12	保本基金C	2018-06-04	2018-06-04	2018-06-04-005306
4	00301877	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）
  i. 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;
```

  ii. 显示执行结果

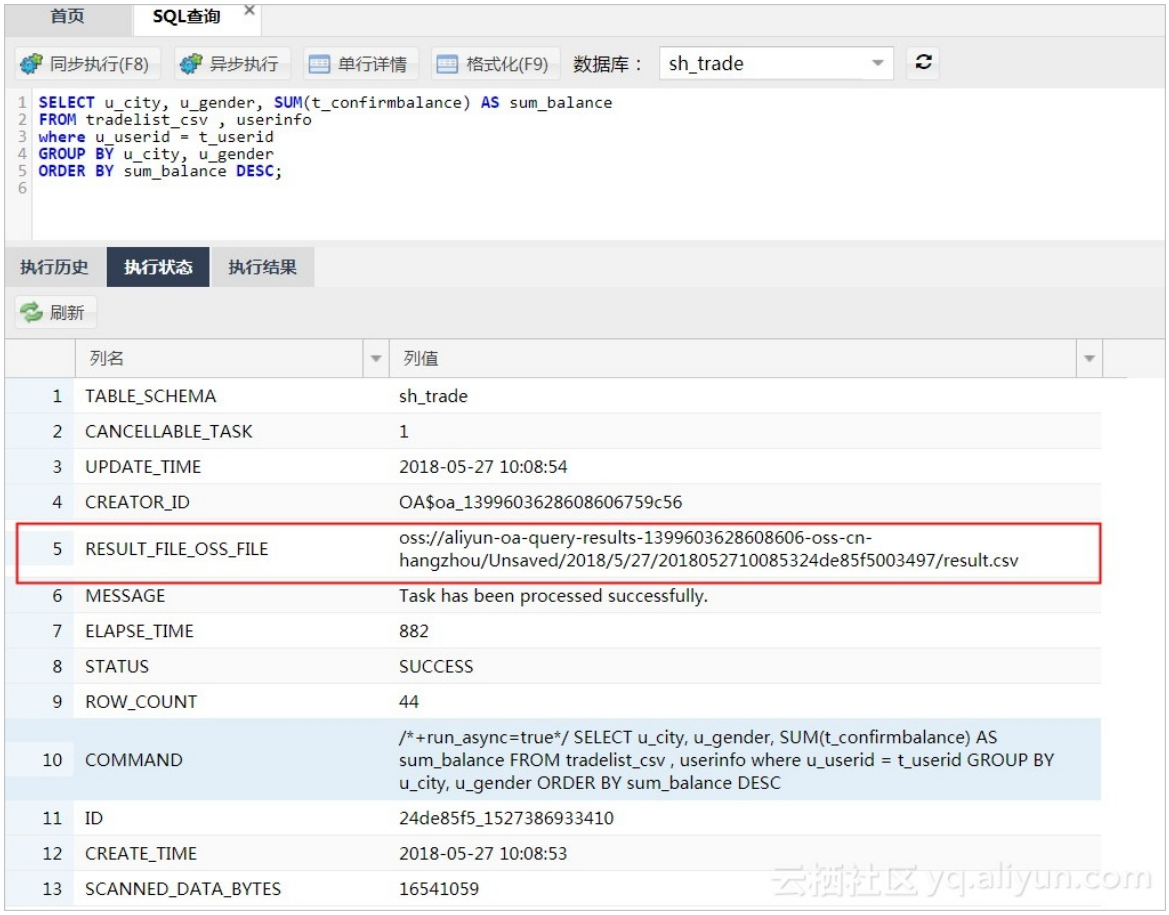


SQL查询（异步执行）

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。

3.2. 开源生态

3.2.1. 使用自建Hadoop访问全托管OSS-HDFS服务

OSS-HDFS服务（JindoFS服务）是一款云原生数据湖存储产品。基于统一的元数据管理能力，在完全兼容HDFS文件系统接口的同时，提供充分的POSIX能力支持，能更好地满足大数据和AI等领域的数据湖计算场景。本文介绍使用自建Hadoop访问全托管OSS-HDFS服务的操作步骤。

前提条件

已开通并授权访问OSS-HDFS服务。具体操作，请参见开通并授权访问OSS-HDFS服务。

什么是OSS-HDFS服务

通过OSS-HDFS服务，无需对现有的Hadoop、Spark大数据分析应用做任何修改。通过简单的配置即可像在原生HDFS中那样管理和访问数据，同时获得OSS无限容量、弹性扩展、更高的安全性、可靠性和可用性支撑。

作为云原生数据湖基础，OSS-HDFS在满足EB级数据分析、亿级文件管理服务、TB级吞吐量的同时，全面融合大数据存储生态，除提供对象存储扁平命名空间之外，还提供了分层命名空间服务。分层命名空间支持将对象组织到一个目录层次结构中进行管理，并能通过统一元数据管理能力进行内部自动转换。对Hadoop用户而言，无需做数据复制或转换就可以实现像访问本地HDFS一样高效的数据访问，极大提升整体作业性能，降低了维护成本。

关于OSS-HDFS服务的应用场景、服务特性、功能特性等更多信息，请参见OSS-HDFS服务概述。

步骤一：创建专有网络VPC并添加云服务器ECS实例

1. 创建允许内网访问OSS-HDFS服务的专有网络VPC。
   i. 登录专有网络管理控制台。
   ii. 在**专有网络**页面，单击**创建专有网络**。

   创建专有网络VPC时，需确保创建的VPC与待开启OSS-HDFS服务的Bucket位于相同的地域（Region）。创建VPC的具体操作，请参见创建专有网络和交换机。
2. 添加云服务器ECS实例。
   i. 单击已创建的VPC ID，然后单击**资源管理**页签。
   ii. 在**包含基础云资源**区域，单击云服务器（ECS）右侧的⊕。
   iii. 在**实例**页面，单击**创建实例**。

   创建ECS实例时，需确保该ECS实例与已创建的专有网络VPC位于相同地域。创建ECS实例的具体操作，请参见选购ECS实例。

步骤二：创建Hadoop运行环境

1. 安装Java环境。
   i. 在已创建的ECS示例右侧，单击**远程连接**。

   关于远程连接ECS实例的具体操作，请参见连接方式概述。
   ii. 检查JDK版本。

   ```
   java -version
   ```

   iii. （可选）如果JDK为1.8.0以下版本，请卸载已有的JDK。如果JDK为1.8.0或以上版本，请跳过此步骤。

   ```
   rpm -qa | grep java | xargs rpm -e --nodeps
   ```

   iv. 安装Java。

   ```
   yum install java-1.8.0-openjdk* -y
   ```

   v. 配置环境变量。

   ```
   vim /etc/profile
   ```

vi. 添加环境变量。

如果提示当前JDK Path不存在，请前往*/usr/lib/jvm/*查找*java-1.8.0-openjdk*。

```
export JAVA_HOME=/usr/lib/jvm/java-1.8.0-openjdk
export CLASSPATH=.:$JAVA_HOME/lib/dt.jar:$JAVA_HOME/lib/tools.jar:$JAVA_HOME/jre/lib/rt.jar
export PATH=$PATH:$JAVA_HOME/bin
```

2. 启用SSH服务。

i. 安装SSH服务。

```
yum install -y openssh-clients openssh-server
```

ii. 启用SSH服务。

```
systemctl enable sshd && systemctl start sshd
```

iii. 生成SSH密钥，并将生成的密钥添加到信任列表。

```
ssh-keygen -t rsa -P '' -f ~/.ssh/id_rsa
cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
chmod 0600 ~/.ssh/authorized_keys
```

3. 安装Hadoop。

i. 下载Hadoop安装包。

```
wget https://mirrors.sonic.net/apache/hadoop/common/hadoop-3.3.1/hadoop-3.3.1.tar.gz
```

ii. 解压安装包。

```
tar xzf hadoop-3.3.1.tar.gz
```

iii. 将安装包移动到常用位置。

```
mv hadoop-3.3.1 /usr/local/hadoop
```

iv. 配置环境变量。

a. 配置Hadoop环境变量。

```
vim /etc/profile
export HADOOP_HOME=/usr/local/hadoop
export PATH=$HADOOP_HOME/bin:$PATH
```

b. 更新Hadoop配置文件中的*HADOOP_HOME*。

```
cd $HADOOP_HOME
vim etc/hadoop/hadoop-env.sh
```

c. 将*${JAVA_HOME}*替换为实际路径。

```
export JAVA_HOME=/usr/lib/jvm/java-1.8.0-openjdk
```

v. （可选）如果提示目录不存在，请执行以下命令，使环境变量生效。

```
cd $HADOOP_HOME/etc/hadoop
```

vi. 更新配置文件*core-site.xml*以及*hdfs-site.xml*。

- 更新配置文件*core-site.xml*并添加属性。

```
<configuration>
  <!-- 指定HDFS中NameNode的地址。-->
  <property>
      <name>fs.defaultFS</name>
      <!--替换为主机名或localhost。-->
      <value>hdfs://localhost:9000</value>
  </property>
  <!--将Hadoop临时目录修改为自定义目录。-->
  <property>
      <name>hadoop.tmp.dir</name>
      <!--admin操作时完成目录授权sudo chown -R admin:admin /opt/module/hadoop-3.3.1
-->
      <value>/opt/module/hadoop-3.3.1/data/tmp</value>
  </property>
</configuration>
```

- 更新配置文件*hdfs-site.xml*并添加属性。

```
<configuration>
  <!-- 指定HDFS副本的数量。-->
  <property>
      <name>dfs.replication</name>
      <value>1</value>
  </property>
</configuration>
```

vii. 格式化文件结构。

```
hdfs namenode -format
```

viii. 启动HDFS。

启动HDFS分为启动NameNode、DataNode和Secondary NameNode三个步骤。

a. 启动HDFS。

```
cd /usr/local/hadoop/
sbin/start-dfs.sh
```

b. 查看进程。

```
jps
```

返回结果如下：

```
[hadoop@02497b hadoop]$ sbin/start-dfs.sh
Starting namenodes on [02497b]
02497b: Warning: Permanently added '02497b' (ECDSA) to the list of known hosts.
Starting datanodes
Starting secondary namenodes [02497b378f76]
[hadoop@02497 hadoop]$ jps
3239 SecondaryNameNode
3402 Jps
3034 DataNode
2908 NameNode
```

完成上述步骤后，即可建立HDFS守护进程。由于HDFS本身具备HTTP面板，您可以通过浏览器访问*http://{ip}:9870*，查看HDFS面板以及详细信息。

4. 测试Hadoop是否安装成功。

   执行**hadoop version**命令，如果正常返回版本信息，表明安装成功。

```
[hadoop@02497b workspace]$ hadoop version
Hadoop 3.3.1
Source code repository https://github.com/apache/hadoop.git -r a3b9c37a397ad4188041dd80621bdeefc46885f2
Compiled by ubuntu on 2021-06-15T05:13Z
Compiled with protoc 3.7.1
From source with checksum 88a4ddb2299aca054416d6b7f81ca55
This command was run using /usr/local/hadoop/share/hadoop/common/hadoop-common-3.3.1.jar
```

步骤三：切换本地HDFS到云上OSS-HDFS服务

1. 创建Bucket时开启HDFS服务。

   具体操作，请参见创建Bucket。

2. 授权访问。

   i. 授权服务端管理已开通HDFS服务的Bucket。

   首次使用HDFS功能时，需要先在RAM控制台完成以下授权，以便OSS服务账号能够管理Bucket中的数据。

   a. 创建名为AliyunOSSDlsDefaultRole的角色。

   b. 新建名为AliyunOSSDlsRolePolicy的自定义权限策略。

   c. 为角色授予自定义权限策略。

ii. 授权RAM用户访问已开通HDFS服务的Bucket。

a. 创建RAM用户。具体操作，请参见创建RAM用户。

b. 创建自定义策略，策略内容如下：

```
{
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "oss:*",
            "Resource": [
                "acs:oss:*:*:*/.dlsdata",
                "acs:oss:*:*:*/.dlsdata*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "oss:GetBucketInfo",
                "oss:PostDataLakeStorageFileOperation"
            ],
            "Resource": "*"
        }
    ],
    "Version": "1"
}
```

创建自定义策略的具体操作，请参见创建自定义权限策略。

c. 为RAM用户授权已创建的自定义策略。具体步骤，请参见为RAM用户授权。

如果您使用服务角色（例如，EMR服务角色 `AliyunEMRDefaultRole` ）访问已开通HDFS服务的Bucket，请参见上述为RAM用户授权的步骤完成服务角色授权。

3. 下载JindoSDK JAR包。

i. 切换至目标目录。

```
cd /usr/lib/
```

ii. 下载JindoSDK JAR包。

```
wget https://jindodata-binary.oss-cn-shanghai.aliyuncs.com/release/4.1.0/jindosdk-4.1.0.tar.gz
```

iii. 解压JindoSDK JAR包。

```
tar xzf jindosdk-4.1.0.tar.gz
```

4. 配置环境变量。

i. 切换至目标目录。

```
vim /etc/profile
```

ii. 解压下载的安装包，以安装包内容解压在*/usr/lib/jindosdk-4.0.0*目录为例。

```
export JINDOSDK_HOME=/usr/lib/jindosdk-4.1.0
```

iii. 配置*HADOOP_CLASSPATH*。

```
export HADOOP_CLASSPATH=$HADOOP_CLASSPATH:${JINDOSDK_HOME}/lib/*
```

iv. 执行以下命令使环境变量配置生效。

```
. /etc/profile
```

5. 配置JindoFS服务实现类及AccessKey。

i. 将JindoSDK DLS实现类配置到Hadoop的core-site.xml中。

```
<configuration>
    <property>
        <name>fs.AbstractFileSystem.oss.impl</name>
        <value>com.aliyun.jindodata.oss.JindoOSS</value>
    </property>
    <property>
        <name>fs.oss.impl</name>
        <value>com.aliyun.jindodata.oss.JindoOssFileSystem</value>
    </property>
</configuration>
```

ii. 将已开启HDFS服务的Bucket对应的accessKeyId、accessKeySecret预先配置在Hadoop的core-site.xml中。

```
<configuration>
    <property>
        <name>fs.oss.accessKeyId</name>
        <value>xxx</value>
    </property>
    <property>
        <name>fs.oss.accessKeySecret</name>
        <value>xxx</value>
    </property>
</configuration>
```

6. 配置OSS-HDFS服务的Endpoint。

访问OSS-HDFS服务时需要配置Endpoint。推荐访问路径格式为 `oss://<Bucket>.<Endpoint>/<Object>` ，例如 `oss://examplebucket.cn-shanghai.oss-dls.aliyuncs.com/exampleobject.txt` 。配置完成后，JindoSDK会根据访问路径中的Endpoint访问对应的OSS-HDFS服务接口。

除上述提到的在访问路径中指定Endpoint的方式以外，您还可以通过其他配置OSS-HDFS服务的Endpoint。更多信息，请参见配置Endpoint的其他方式。

步骤四：访问OSS-HDFS服务

- 新建目录

在目标存储空间examplebucket下创建名为dir/的目录，示例如下：

```
hdfs dfs -mkdir oss://examplebucket.cn-hangzhou.oss-dls.aliyuncs.com/dir/
```

- 上传文件

将本地examplefile.txt文件上传至目标存储空间examplebucket，示例如下：

```
hdfs dfs -put /root/workspace/examplefile.txt oss://examplebucket.cn-hangzhou.oss-dls.aliyuncs.com/examplefile.txt
```

- 查看目录信息

  查看目标存储空间examplebucket下目录dir/的信息，示例如下：

```
hdfs dfs -ls oss://examplebucket.oss-dls.aliyuncs.com/dir/
```

- 查看文件信息

  查看目标存储空间examplebucket下文件examplefile.txt的信息，示例如下：

```
hdfs dfs -ls oss://examplebucket.oss-dls.aliyuncs.com/examplefile.txt
```

- 查看文件内容

  查看目标存储空间examplebucket下文件examplefile.txt的内容，示例如下：

  > **注意** 执行以下命令后，文件内容将以纯文本形式打印在屏幕上。如果文件存在特定格式的编码，请使用HDFS的Java API读取文件内容，然后进行解码操作后即可获取对应的文件内容。

```
hdfs dfs -cat oss://examplebucket.oss-dls.aliyuncs.com/examplefile.txt
```

- 拷贝目录或文件

  将目标存储空间examplebucket下根目录subdir1拷贝到目录subdir2下，且根目录subdir1所在的位置、根目录下的文件和子目录结构和内容保持不变，示例如下：

```
hdfs dfs -cp oss://examplebucket.oss-dls.aliyuncs.com/subdir1/  oss://examplebucket.oss-dls.aliyuncs.com/subdir2/subdir1/
```

- 移动目录或文件

  将目标存储空间根目录srcdir及其包含的文件或者子目录移动至另一个根目录destdir下，示例如下：

```
hdfs dfs -mv oss://examplebucket.oss-dls.aliyuncs.com/srcdir/  oss://examplebucket.oss-dls.aliyuncs.com/destdir/
```

- 下载文件

  将目标存储空间examplebucket下的exampleobject.txt下载到本地根目录文件夹/tmp下，示例如下：

```
hdfs dfs -get oss://examplebucket.oss-dls.aliyuncs.com/exampleobject.txt  /tmp/
```

- 删除目录或文件

  删除目标存储空间examplebucket下目录destfolder/及其目录下的所有文件，示例如下：

```
hdfs dfs -rm oss://examplebucket.oss-dls.aliyuncs.com/destfolder/
```

3.2.2. 通过HDP 2.6 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Hadoop 3.1.1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。

前提条件

您需要拥有一个已搭建好的HDP 2.6.1.0的集群。若没有已搭建好的HDP 2.6.1.0集群，您可以通过以下方式搭建：

- 查找参考文档利用Ambari搭建HDP 2.6.1.0的集群。
- 不使用Ambari，自行搭建HDP 2.6.1.0集群。

配置步骤

1. 下载HDP 2.6.1.0版本支持OSS的支持包。

   此支持包是根据HDP 2.6.1.0中Hadoop的版本，打了Apache Hadoop对OSS支持的补丁后编译得到的，其他HDP 2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
```

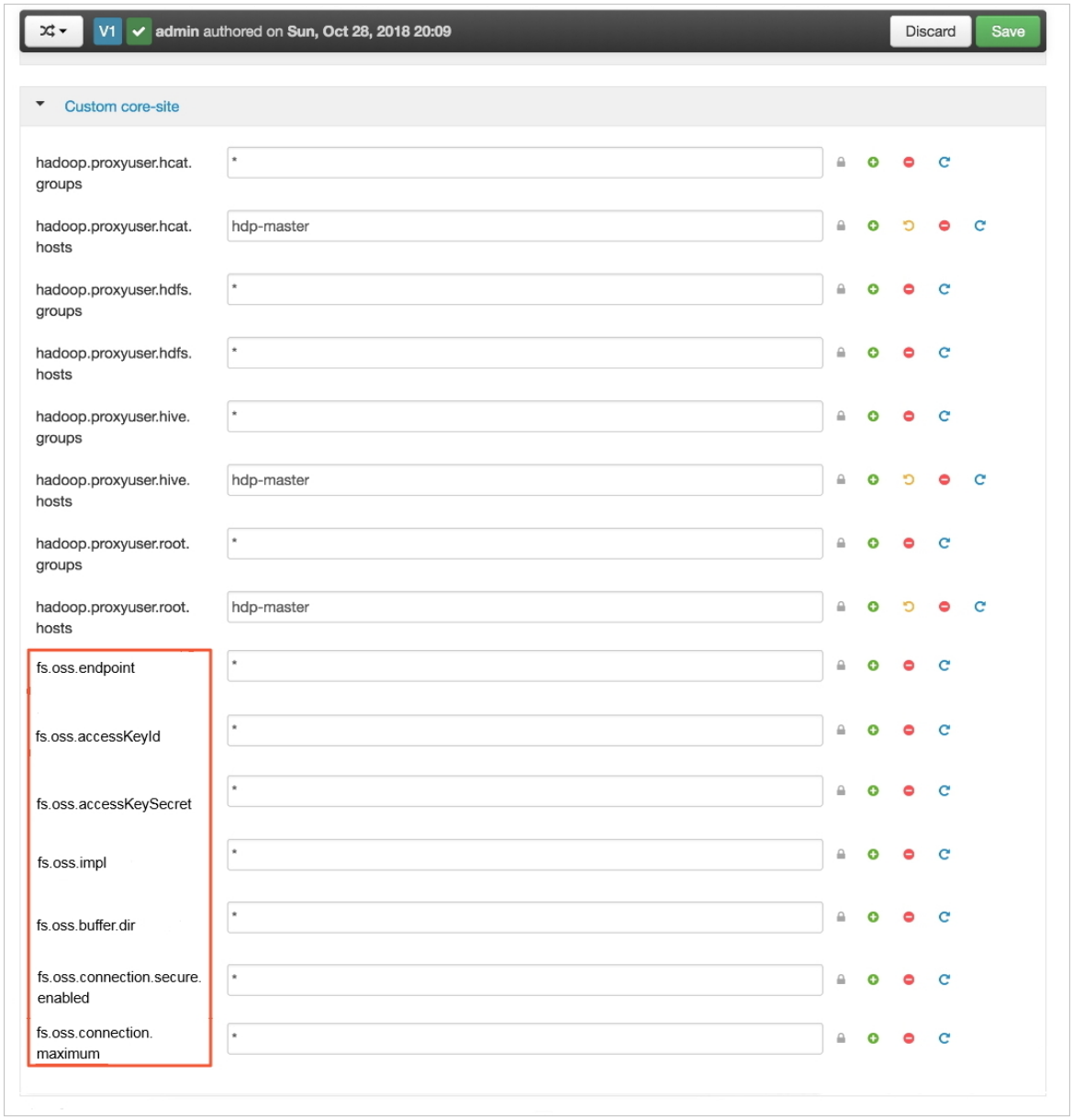
3. 将*hadoop-aliyun-2.7.3.2.6.1.0-129.jar*移至*${/usr/hdp/current}/hadoop-client/*目录内，其余的*jar*文件移至*${/usr/hdp/current}/hadoop-client/lib/*目录内。

   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

配置项	值
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

更多参数解释请参见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/192.16
8.0.161:8050
18/10/28 21:32:38 INFO client.AHSProxy: Connecting to Application History server at cdh-m
aster/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_15407289865
31_0005
18/10/28 21:32:39 INFO impl.YarnClientImpl: Submitted application application_15407289865
31_0005
18/10/28 21:32:39 INFO mapreduce.Job: The url to track the job: http://cdh-master:8088/pr
oxy/application_1540728986531_0005/
18/10/28 21:32:39 INFO mapreduce.Job: Running job: job_1540728986531_0005
18/10/28 21:32:49 INFO mapreduce.Job: Job job_1540728986531_0005 running in uber mode : f
alse
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_15407289865
31_0006
18/10/28 21:39:02 INFO impl.YarnClientImpl: Submitted application application_15407289865
31_0006
18/10/28 21:39:02 INFO mapreduce.Job: The url to track the job: http://cdh-master:8088/pr
oxy/application_1540728986531_0006/
18/10/28 21:39:02 INFO mapreduce.Job: Running job: job_1540728986531_0006
18/10/28 21:39:09 INFO mapreduce.Job: Job job_1540728986531_0006 running in uber mode : f
alse
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。

3.2.3. Flink使用JindoSDK处理OSS-HDFS服务的数据

开源版本Flink不支持流式写入OSS-HDFS（JindoFS）服务，也不支持以EXACTLY_ONCE语义写入存储介质。当您希望开源版本Flink以EXACTLY_ONCE语义流式写入OSS-HDFS服务，需要结合JindoSDK。

前提条件

已在集群中部署开源版本Flink，且版本不低于1.10.1。

步骤一：部署JindoSDK

在所有Flink节点根目录下的*lib*文件夹下放置*.jar*文件，命令如下：

```
jindo-flink-${version}-full.jar
```

*.jar*文件包含在*jindosdk-${version}.tar.gz*内，解压缩后可在*plugins/flink/*目录下查看。

步骤二：在Flink作业中的用法

- 通用配置

  为了支持EXACTLY_ONCE语义写入OSS，您需要执行如下配置：

i. 打开Flink的检查点（Checkpoint）。

a. 通过如下方式建立的StreamExecutionEnvironment。

```
StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();
```

b. 执行如下命令，启动Checkpoint。

```
env.enableCheckpointing(<userDefinedCheckpointInterval>, CheckpointingMode.EXACTLY_ONCE);
```

ii. 使用可以重发的数据源，例如Kafka。

- 便捷使用

  您无需额外引入依赖，只需使用带有*oss://*前缀的路径，即可使用该功能。JindoFS可以自动识别*oss://*前缀，并启用该功能。

  以将DataStream<String>的对象OutputStream写入OSS为例。

  i. 添加Sink。

  ```
  String outputPath = "oss://<user-defined-oss-bucket>/<user-defined-oss-dir>"
  StreamingFileSink<String> sink = StreamingFileSink.forRowFormat(
          new Path(outputPath),
          new SimpleStringEncoder<String>("UTF-8")
  ).build();
  outputStream.addSink(sink);
  ```

  ii. 使用`env.execute()`执行Flink作业。

（可选）步骤三：自定义配置

您在提交Flink作业时，可以自定义参数，以开启或控制特定功能。

例如，以yarn-cluster模式提交Flink作业时，通过`-yD`配置。示例如下。

```
<flink_home>/bin/flink run -m yarn-cluster -yD key1=value1 -yD key2=value2 ...
```

目前，JindoSDK支持配置开启熵注入（Entropy Injection）功能或控制分片上传（Multipart Upload）的并行度。

- 熵注入

  该功能可以匹配写入路径的一段特定字符串，用一段随机的字符串进行替换，以削弱所谓片区效应，提高写入效率。

  当写入场景为OSS-HDFS时，需要完成下列配置。

  ```
  oss.entropy.key=<user-defined-key>
  oss.entropy.length=<user-defined-length>
  ```

  写入新文件时，路径中与`<user-defined-key>`相同的字符串会被替换为一个随机字符串，随机串的长度为`<user-defined-length>`，且`<user-defined-length>`必须大于零。

- 分片上传

当写入场景为OSS-HDFS时，可恢复性读写功能会自动调用高效的分片上传机制，将待上传的文件分为多个数据块分别上传，最后组合。目前支持配置参数*oss.upload.max.concurrent.uploads*，用来控制上传数据块的并行度，如果设置较高的数值则可能会提高写入效率（但也会占用更多资源）。默认情况下，该值为当前可用的处理器数量。

3.2.4. HBase使用OSS-HDFS服务作为底层存储

HBase是Hadoop生态中的实时数据库，有较高的写入性能。 OSS-HDFS服务（JindoFS服务）是阿里云新推出的存储空间类型，并兼容HDFS接口。JindoSDK支持HBase使用OSS-HDFS服务作为底层存储，同时支持存储WAL文件，实现存储与计算分离。相对于本地HDFS存储，OSS-HDFS服务使用更加灵活，且一定程度减少了运维成本。

步骤一：下载和安装JAR包

1. 下载最新版本的JindoSDK JAR包。下载地址，请参见GitHub。

   > **说明** 4.3.0及以上版本包含Kerberos和SASL支持。

2. 如果您的环境中未包含Kerberos和SASL相关依赖，则需要在部署JindoSDK的所有节点安装以下依赖。
   - Ubuntu或Debian

     ```
     sudo apt-get install libkrb5-dev krb5-admin-server krb5-kdc krb5-user libsasl2-dev li
     bsasl2-modules libsasl2-modules-gssapi-mit
     ```

   - Red Hat Enterprise Linux或CentOS

     ```
     sudo yum install krb5-server krb5-workstation cyrus-sasl-devel cyrus-sasl-gssapi cyru
     s-sasl-plain
     ```

   - macOS

     ```
     brew install krb5
     ```

3. 解压下载的安装包。

   以下以安装包解压到*/usr/lib*路径为例：

   ```
   tar -zxvf jindosdk-4.3.0.tar.gz -C /usr/lib
   ```

4. 配置`JINDOSDK_HOME`。

   ```
   export JINDOSDK_HOME=/usr/lib/jindosdk-4.3.0
   export PATH=$JINDOSDK_HOME/bin:$PATH
   ```

5. 配置`HADOOP_CLASSPATH`。

   > **注意** 请将安装目录和环境变量部署到所有所需节点上。

6. 安装JAR包。

   以下以将JAR包安装到`HADOOP_CLASSPATH`路径为例：

   ```
   cp ./jindosdk-4.3.0.jar <HADOOP_CLASSPATH>/share/hadoop/hdfs/lib/jindofs-sdk.jar
   ```

步骤二：配置OSS-HDFS服务实现类及AccessKey

1. 将JindoSDK OSS-HDFS服务实现类配置到HBase的*core-site.xml*文件中。

   配置内容如下：

```
<configuration>
    <property>
        <name>fs.AbstractFileSystem.oss.impl</name>
        <value>com.aliyun.jindodata.oss.OSS</value>
    </property>
    <property>
        <name>fs.oss.impl</name>
        <value>com.aliyun.jindodata.oss.JindoOssFileSystem</value>
    </property>
</configuration>
```

2. 将已开启OSS-HDFS服务的Bucket对应的AccessKey ID、AccessKey Secret预先配置在HBase的*core-site.xml*文件中。

```
<configuration>
    <property>
        <name>fs.oss.accessKeyId</name>
        <value>LTAI5t7h6SgiLSganP2m****</value>
    </property>
    <property>
        <name>fs.oss.accessKeySecret</name>
        <value>KZo149BD9GLPNiDIEmdQ7d****</value>
    </property>
</configuration>
```

步骤三：配置OSS-HDFS服务Endpoint

访问OSS-HDFS服务时需要配置Endpoint。推荐访问路径格式为 `oss://<Bucket>.<Endpoint>/<Object>`，例如 `oss://examplebucket.cn-shanghai.oss-dls.aliyuncs.com/exampleobject.txt`。配置完成后，JindoSDK会根据访问路径中的Endpoint访问对应的OSS-HDFS服务接口。

您还可以通过其他方式配置OSS-HDFS服务Endpoint，且不同方式配置的Endpoint存在生效优先级。更多信息，请参见配置Endpoint的其他方式。

步骤四：指定HBase的存储路径

您可以通过将*hbase-site*配置文件中的参数hbase.rootdir的值修改为OSS地址（格式为 `oss://bucket.endpoint/hbase-root-dir`）的方式，指定HBase和WAL文件的存储路径。

> **注意** 如果要释放集群，需要先禁用table，确保WAL文件已全量更新到存储文件HFile。

3.2.5. Hive使用JindoSDK处理OSS-HDFS服务中的数据

使用Hive搭建离线数仓时，随着数据量的不断增长，传统的基于HDFS存储的数仓可能无法以较低成本满足用户的需求。在这种情况下，您可以使用OSS-HDFS服务（JindoFS服务）作为Hive数仓的底层存储，并通过JindoSDK获得更好的读写性能。

步骤一：在Hive客户端或服务所在结点安装JindoSDK

1. 下载最新版本的JindoSDK JAR包。下载地址，请参见GitHub。

> 说明 4.3.0及以上版本包含Kerberos和SASL支持。

2. 如果您的环境中未包含Kerberos和SASL相关依赖，则需要在部署JindoSDK的所有节点安装以下依赖。
   - Ubuntu或Debian

     ```
     sudo apt-get install libkrb5-dev krb5-admin-server krb5-kdc krb5-user libsasl2-dev libsasl2-modules libsasl2-modules-gssapi-mit
     ```

   - Red Hat Enterprise Linux或CentOS

     ```
     sudo yum install krb5-server krb5-workstation cyrus-sasl-devel cyrus-sasl-gssapi cyrus-sasl-plain
     ```

   - macOS

     ```
     brew install krb5
     ```

3. 将已下载的JindoSDK JAR包安装到Hive的classpath路径下。

   安装命令如下：

   ```
   cp jindosdk-4.3.0/lib/*.jar  $HIVE_HOME/lib/
   ```

步骤二：配置OSS-HDFS服务实现类及AccessKey

1. 将JindoSDK OSS-HDFS服务实现类配置到Hive的*core-site.xml*文件中。

   配置内容如下：

   ```
   <configuration>
       <property>
           <name>fs.AbstractFileSystem.oss.impl</name>
           <value>com.aliyun.jindodata.oss.OSS</value>
       </property>
       <property>
           <name>fs.oss.impl</name>
           <value>com.aliyun.jindodata.oss.JindoOssFileSystem</value>
       </property>
   </configuration>
   ```

2. 将已开启OSS-HDFS服务的Bucket对应的AccessKey ID、AccessKey Secret配置在Hive的*core-site.xml*文件中。

   ```
   <configuration>
       <property>
           <name>fs.oss.accessKeyId</name>
           <value>LTAI5t7h6SgiLSganP2m****</value>
       </property>
       <property>
           <name>fs.oss.accessKeySecret</name>
           <value>KZo149BD9GLPNiDIEmdQ7d****</value>
       </property>
   </configuration>
   ```

步骤三：配置OSS-HDFS服务Endpoint

访问OSS-HDFS服务时需要配置Endpoint。推荐访问路径格式为 `oss://<Bucket>.<Endpoint>/<Object>` ，例如 `oss://examplebucket.cn-shanghai.oss-dls.aliyuncs.com/exampleobject.txt` 。配置完成后，JindoSDK会根据访问路径中的Endpoint访问对应的OSS-HDFS服务接口。

您还可以通过其他方式配置OSS-HDFS服务Endpoint，且不同方式配置的Endpoint存在生效优先级。更多信息，请参见配置Endpoint的其他方式。

完成以上配置后，您需要重启Hive服务，使配置生效。

步骤四：通过OSS-HDFS服务存储数据

创建数据库和表时，您可以通过以下两种方式指定OSS-HDFS服务路径，将数据库或表的数据保存到OSS-HDFS服务中。

- 方式一：在命令示例中指定OSS-HDFS服务路径
  - 创建数据库时指定OSS-HDFS服务路径

    ```
    CREATE DATABASE db_on_oss1 LOCATION 'oss://bucket_name.endpoint_name/path/to/db1';
    ```

  - 创建表时指定OSS-HDFS服务路径

    ```
    CREATE TABLE db2.table_on_oss ... LOCATION 'oss://bucket_name.endpoint_name/path/to/db2/tablepath';
    ```

- 方式二：在配置文件中指定OSS-HDFS服务路径

  您可以在Hive Metastore的*hive-site.xml*配置文件中设置*hive.metastore.warehouse.dir*到OSS-HDFS服务路径，然后重启Hive Metastore，则后续创建的数据库和数据库下的表均默认存储于OSS-HDFS服务路径中。

  配置示例如下：

  ```
  <configuration>
      <property>
          <name>hive.metastore.warehouse.dir</name>
          <value>oss://bucket_name.endpoint_name/path/to/warehouse</value>
      </property>
  </configuration>
  ```

步骤五：为已有表添加分区

您可以为已创建的表添加分区，从而将其分成较小的存储单元。根据查询条件，只扫描满足条件的分区而避免全表扫描，从而显著提升查询性能。

- 命令格式

  ```
  ALTER TABLE <table_name> ADD [IF NOT EXISTS] PARTITION <pt_spec> [PARTITION <pt_spec> PARTITION <pt_spec>...] LOCATION 'location';
  ```

  参数说明如下：

参数	是否可选	说明
table_name	必选	待添加分区的表名称。
IF NOT EXISTS	可选	未指定IF NOT EXISTS时，如果同名的分区已存在，会执行失败并返回报错。

参数	是否可选	说明
pt_spec	必选	新增的分区，格式为 `(partition_col1 = partition_col_value1, partition_col2 = partition_col_value2, ...)` 。其中，`partition_col` 表示分区字段，`partition_col_value` 表示分区值。分区字段不区分大小写，分区值区分大小写。
location	必选	指定存储分区的OSS路径。

- 使用示例

  以下示例用于为表sale_detail添加一个分区，用于存储2013年12月华东1（杭州）地域的销售记录，并将分区存储于指定的OSS路径。

```
ALTER TABLE sale_detail ADD IF NOT EXISTS PARTITION (sale_date='201312', region='hangzhou
') LOCATION 'oss://examplebucket.cn-hangzhou.oss-dls.aliyuncs.com/path/2013/';
```

3.2.6. Impala使用JindoSDK查询OSS-HDFS服务中的数据

JindoSDK是一个面向Hadoop、Spark生态且简单易用的OSS客户端，为OSS提供高度优化的Hadoop FileSystem实现。相对于Hadoop社区OSS客户端，Impala使用JindoSDK查询OSS-HDFS服务（JindoFS服务）中的数据时，可以获得更好的性能。

前提条件

已安装Hadoop。具体操作，请参见创建Hadoop运行环境。

步骤一：在Impala所有节点安装JindoSDK

1. 下载最新版本的JindoSDK JAR包。下载地址，请参见GitHub。

   > **说明** 4.3.0及以上版本包含Kerberos和SASL支持。

2. 如果您的环境中未包含Kerberos和SASL相关依赖，则需要在部署JindoSDK的所有节点安装以下依赖。
   - Ubuntu或Debian

     ```
     sudo apt-get install libkrb5-dev krb5-admin-server krb5-kdc krb5-user libsasl2-dev li
     bsasl2-modules libsasl2-modules-gssapi-mit
     ```

   - Red Hat Enterprise Linux或CentOS

     ```
     sudo yum install krb5-server krb5-workstation cyrus-sasl-devel cyrus-sasl-gssapi cyru
     s-sasl-plain
     ```

   - macOS

     ```
     brew install krb5
     ```

3. 将已下载的JindoSDK JAR包安装到Impala的classpath路径下。

```
cp jindosdk-4.3.0/lib/*.jar  $IMPALA_HOME/lib/
```

步骤二：配置OSS-HDFS服务实现类及AccessKey

1. 将JindoSDK OSS-HDFS服务实现类配置到Impala的*core-site.xml*文件中。

   配置内容如下：

```
<configuration>
    <property>
        <name>fs.AbstractFileSystem.oss.impl</name>
        <value>com.aliyun.jindodata.oss.OSS</value>
    </property>
    <property>
        <name>fs.oss.impl</name>
        <value>com.aliyun.jindodata.oss.JindoOssFileSystem</value>
    </property>
</configuration>
```

2. 将已开启OSS-HDFS服务的Bucket对应的AccessKey ID、AccessKey Secret配置在Impara的*core-site.xml*文件中。

```
<configuration>
    <property>
        <name>fs.oss.accessKeyId</name>
        <value>LTAI5t7h6SgiLSganP2m****</value>
    </property>
    <property>
        <name>fs.oss.accessKeySecret</name>
        <value>KZo149BD9GLPNiDIEmdQ7d****</value>
    </property>
</configuration>
```

步骤三：配置OSS-HDFS服务Endpoint

访问OSS-HDFS服务时需要配置Endpoint。推荐访问路径格式为 `oss://<Bucket>.<Endpoint>/<Object>`，例如 `oss://examplebucket.cn-shanghai.oss-dls.aliyuncs.com/exampleobject.txt`。配置完成后，JindoSDK会根据访问路径中的Endpoint访问对应的OSS-HDFS服务接口。

您还可以通过其他方式配置OSS-HDFS服务Endpoint，且不同方式配置的Endpoint存在生效优先级。更多信息，请参见配置Endpoint的其他方式。

步骤四：使用Impala访问OSS

1. 创建表。

```
CREATE EXTERNAL TABLE customer_demographics (
 `cd_demo_sk` INT,
 `cd_gender` STRING,
 `cd_marital_status` STRING,
 `cd_education_status` STRING,
 `cd_purchase_estimate` INT,
 `cd_credit_rating` STRING,
 `cd_dep_count` INT,
 `cd_dep_employed_count` INT,
 `cd_dep_college_count` INT)
STORED AS PARQUET
LOCATION 'oss://bucket.endpoint/dir';
```

2. 查询表数据。

```
select * from customer_demographics;
```

（可选）步骤五：JindoSDK性能调优

您可以结合实际业务需求，将以下配置项添加到Hadoop的core-site.xml中。仅JindoSDK 4.0及以上版本支持以下配置项。

```
<configuration>
    <property>
          <!-- 客户端写入的临时文件目录，可配置多个，每个临时文件目录需以逗号隔开。多用户环境需配置可读写权限 -->
        <name>fs.oss.tmp.data.dirs</name>
        <value>/tmp/</value>
    </property>
    <property>
          <!-- 访问OSS失败重试次数 -->
        <name>fs.oss.retry.count</name>
        <value>5</value>
    </property>
    <property>
          <!-- 请求OSS超时时间（毫秒） -->
        <name>fs.oss.timeout.millisecond</name>
        <value>30000</value>
    </property>
    <property>
          <!-- 连接OSS超时时间（毫秒） -->
        <name>fs.oss.connection.timeout.millisecond</name>
        <value>3000</value>
    </property>
    <property>
          <!-- OSS单个文件并发上传线程数 -->
        <name>fs.oss.upload.thread.concurrency</name>
        <value>5</value>
    </property>
    <property>
          <!-- OSS并发上传任务队列大小 -->
        <name>fs.oss.upload.queue.size</name>
        <value>5</value>
    </property>
```

```
    <property>
        <!-- 进程内OSS最大并发上传任务数 -->
        <name>fs.oss.upload.max.pending.tasks.per.stream</name>
        <value>16</value>
    </property>
    <property>
        <!-- OSS并发下载任务队列大小 -->
        <name>fs.oss.download.queue.size</name>
        <value>5</value>
    </property>
    <property>
        <!-- 进程内OSS最大并发下载任务数 -->
        <name>fs.oss.download.thread.concurrency</name>
        <value>16</value>
    </property>
    <property>
        <!-- 预读OSS的buffer大小 -->
        <name>fs.oss.read.readahead.buffer.size</name>
        <value>1048576</value>
    </property>
    <property>
        <!-- 同时预读OSS的buffer个数 -->
        <name>fs.oss.read.readahead.buffer.count</name>
        <value>4</value>
    </property>
</configuration>
```

3.2.7. Spark使用JindoSDK处理OSS-HDFS服务中的数据

JindoSDK是一个面向Hadoop、Spark生态且简单易用的OSS客户端，为OSS提供高度优化的Hadoop FileSystem实现。相对于Hadoop社区OSS客户端，Spark使用JindoSDK查询OSS-HDFS服务（JindoFS服务）中的数据时，可以获得更好的性能。

前提条件

已安装Hadoop。具体操作，请参见创建Hadoop运行环境。

步骤一：在Spark所有节点安装JindoSDK

1. 下载最新版本的JindoSDK JAR包。下载地址，请参见GitHub。

   > **说明** 4.3.0及以上版本包含Kerberos和SASL支持。

2. 如果您的环境中未包含Kerberos和SASL相关依赖，则需要在部署JindoSDK的所有节点安装以下依赖。
   - Ubuntu或Debian

     ```
     sudo apt-get install libkrb5-dev krb5-admin-server krb5-kdc krb5-user libsasl2-dev libsasl2-modules libsasl2-modules-gssapi-mit
     ```

   - Red Hat Enterprise Linux或CentOS

```
sudo yum install krb5-server krb5-workstation cyrus-sasl-devel cyrus-sasl-gssapi cyru
s-sasl-plain
```

- macOS

```
brew install krb5
```

3. 将已下载的JindoSDK JAR包安装到Spark的classpath路径下。

```
cp jindosdk-4.3.0/lib/*.jar  $SPARK_HOME/jars/
```

步骤二：配置OSS-HDFS服务实现类及AccessKey

- 在*core-site.xml*文件中配置
  i. 将JindoSDK OSS-HDFS服务实现类配置到Spark的*core-site.xml*配置文件中。

  配置示例如下：

```
<configuration>
    <property>
        <name>fs.AbstractFileSystem.oss.impl</name>
        <value>com.aliyun.jindodata.oss.OSS</value>
    </property>
    <property>
        <name>fs.oss.impl</name>
        <value>com.aliyun.jindodata.oss.JindoOssFileSystem</value>
    </property>
</configuration>
```

  ii. 将已开启OSS-HDFS服务的Bucket对应的AccessKey ID、AccessKey Secret配置在Spark的*core-site.xml*配置文件中。

```
<configuration>
    <property>
        <name>fs.oss.accessKeyId</name>
        <value>LTAI5t7h6SgiLSganP2m****</value>
    </property>
    <property>
        <name>fs.oss.accessKeySecret</name>
        <value>KZo149BD9GLPNiDIEmdQ7d****</value>
    </property>
</configuration>
```

- 在提交任务时配置

  在提交Spark任务时配置OSS-HDFS服务实现类及AccessKey，示例如下：

```
spark-submit --conf spark.hadoop.fs.AbstractFileSystem.oss.impl=com.aliyun.jindodata.oss.
OSS --conf spark.hadoop.fs.oss.impl=com.aliyun.jindodata.oss.JindoOssFileSystem --conf sp
ark.hadoop.fs.oss.accessKeyId=LTAI5t7h6SgiLSganP2m****  --conf spark.hadoop.fs.oss.access
KeySecret=KZo149BD9GLPNiDIEmdQ7d****
```

步骤三：配置OSS-HDFS服务Endpoint

访问OSS-HDFS服务时需要配置Endpoint。推荐访问路径格式为 `oss://<Bucket>.<Endpoint>/<Object>`，例如 `oss://examplebucket.cn-shanghai.oss-dls.aliyuncs.com/exampleobject.txt`。配置完成后，JindoSDK会根据访问路径中的Endpoint访问对应的OSS-HDFS服务接口。

您还可以通过其他方式配置OSS-HDFS服务Endpoint，且不同方式配置的Endpoint存在生效优先级。更多信息，请参见配置Endpoint的其他方式。

步骤四：使用Spark访问OSS

1. 创建表。

```
create table test_oss (c1 string) location "oss://examplebucket.cn-hangzhou.oss-dls.aliyuncs.com/dir/";
```

2. 往表中插入数据。

```
insert into table test_oss values ("testdata");
```

3. 查询表。

```
select * from test_oss;
```

（可选）步骤五：JindoSDK性能调优

您可以结合实际业务需求，将以下配置项添加到Hadoop的core-site.xml中。仅JindoSDK 4.0及以上版本支持以下配置项。

```
<configuration>
    <property>
          <!-- 客户端写入的临时文件目录，可配置多个，每个临时文件目录需以逗号隔开。多用户环境需配置可读写权限 -->
        <name>fs.oss.tmp.data.dirs</name>
        <value>/tmp/</value>
    </property>
    <property>
          <!-- 访问OSS失败重试次数 -->
        <name>fs.oss.retry.count</name>
        <value>5</value>
    </property>
    <property>
          <!-- 请求OSS超时时间（毫秒） -->
        <name>fs.oss.timeout.millisecond</name>
        <value>30000</value>
    </property>
    <property>
          <!-- 连接OSS超时时间（毫秒） -->
        <name>fs.oss.connection.timeout.millisecond</name>
        <value>3000</value>
    </property>
    <property>
          <!-- OSS单个文件并发上传线程数 -->
        <name>fs.oss.upload.thread.concurrency</name>
        <value>5</value>
    </property>
    <property>
          <!-- OSS并发上传任务队列大小 -->
```

```
        <name>fs.oss.upload.queue.size</name>
        <value>5</value>
    </property>
    <property>
        <!-- 进程内OSS最大并发上传任务数 -->
        <name>fs.oss.upload.max.pending.tasks.per.stream</name>
        <value>16</value>
    </property>
    <property>
        <!-- OSS并发下载任务队列大小 -->
        <name>fs.oss.download.queue.size</name>
        <value>5</value>
    </property>
    <property>
        <!-- 进程内OSS最大并发下载任务数 -->
        <name>fs.oss.download.thread.concurrency</name>
        <value>16</value>
    </property>
    <property>
        <!-- 预读OSS的buffer大小 -->
        <name>fs.oss.read.readahead.buffer.size</name>
        <value>1048576</value>
    </property>
    <property>
        <!-- 同时预读OSS的buffer个数 -->
        <name>fs.oss.read.readahead.buffer.count</name>
        <value>4</value>
    </property>
</configuration>
```

3.2.8. Presto使用JindoSDK查询OSS-HDFS服务中的数据

Presto是一个开源的分布式SQL查询引擎，适用于交互式分析查询。本文介绍Presto如何使用JindoSDK查询OSS-HDFS服务（JindoFS服务）中的数据。

步骤一：在Presto所有节点安装JindoSDK

1. 下载最新版本的JindoSDK JAR包。下载地址，请参见GitHub。

   > **说明** 4.3.0及以上版本包含Kerberos和SASL支持。

2. 如果您的环境中未包含Kerberos和SASL相关依赖，则需要在部署JindoSDK的所有节点安装以下依赖。
   - Ubuntu或Debian

     ```
     sudo apt-get install libkrb5-dev krb5-admin-server krb5-kdc krb5-user libsasl2-dev libsasl2-modules libsasl2-modules-gssapi-mit
     ```

   - Red Hat Enterprise Linux或CentOS

```
sudo yum install krb5-server krb5-workstation cyrus-sasl-devel cyrus-sasl-gssapi cyrus-sasl-plain
```

- macOS

```
brew install krb5
```

3. 将已下载的JindoSDK JAR包安装到Presto的classpath路径下。

```
cp jindosdk-4.3.0/lib/*.jar  $PRESTO_HOME/plugin/hive-hadoop2/
```

步骤二：配置OSS-HDFS服务实现类及AccessKey

1. 将JindoSDK OSS-HDFS服务实现类配置到Presto所有节点上的Hadoop配置文件*core-site.xml*中。

   配置内容如下：

```
<configuration>
    <property>
        <name>fs.AbstractFileSystem.oss.impl</name>
        <value>com.aliyun.jindodata.oss.OSS</value>
    </property>
    <property>
        <name>fs.oss.impl</name>
        <value>com.aliyun.jindodata.oss.JindoOssFileSystem</value>
    </property>
</configuration>
```

2. 将已开启OSS-HDFS服务的Bucket对应的AccessKey ID、AccessKey Secret配置到Presto所有节点上的Hadoop配置文件*core-site.xml*文件中。

```
<configuration>
    <property>
        <name>fs.oss.accessKeyId</name>
        <value>LTAI5t7h6SgiLSganP2m****</value>
    </property>
    <property>
        <name>fs.oss.accessKeySecret</name>
        <value>KZo149BD9GLPNiDIEmdQ7d****</value>
    </property>
</configuration>
```

步骤三：配置OSS-HDFS服务Endpoint

访问OSS-HDFS服务时需要配置Endpoint。推荐访问路径格式为 `oss://<Bucket>.<Endpoint>/<Object>`，例如 `oss://examplebucket.cn-shanghai.oss-dls.aliyuncs.com/exampleobject.txt`。配置完成后，JindoSDK会根据访问路径中的Endpoint访问对应的OSS-HDFS服务接口。

您还可以通过其他方式配置OSS-HDFS服务Endpoint，且不同方式配置的Endpoint存在生效优先级。更多信息，请参见配置Endpoint的其他方式。

完成以上配置后，您需要重启Presto服务，使配置生效。

步骤四：查询OSS-HDFS服务中的数据

以下以常用的Hive catalog为例，使用Presto创建一个OSS中的schema，并执行简单的SQL查询示例。由于Presto依赖Hive Metastore，因此Hive服务也需要安装并部署JindoSDK。具体操作，请参见Hive使用JindoSDK处理OSS-HDFS服务中的数据。

1. 登录Presto控制台。

```
presto --server <presto_server_address>:<presto_server_port> --catalog hive
```

2. 创建OSS中的schema。

```
create schema testDB with (location='oss://<Bucket>.<Endpoint>/<schema_dir>');
use testDB;
```

3. 创建表。

```
create table tbl (key int, val int);
```

4. 往表中插入数据。

```
insert into tbl values (1,666);
```

5. 查询表。

```
select * from tbl;
```

3.2.9. Sqoop使用Kite SDK读写OSS-HDFS服务的数据

Sqoop是一款Apache社区的开源软件，支持在Hadoop生态软件和结构化数据集（例如数据库）之间进行高效的批量数据传输。Sqoop本身并不支持对OSS-HDFS服务的数据进行读写操作，您可以通过第三方Kite SDK进行URI转换的方式实现数据的交互。

前提条件

- 在集群上有开源版本Sqoop软件，且版本不低于1.4.7。下载地址，请参见Apache Sqoop。
- Sqoop依赖的Hadoop环境可访问OSS-HDFS服务。具体操作，请参见OSS-HDFS服务快速入门。

操作步骤

1. 安装JindoSDK JAR包。
   i. 下载kite-data-oss-3.4.0.jar。
   ii. 将已下载的JAR包安装在Sqoop的classpath路径下。

```
cp ./kite-data-oss-3.4.0.jar <SQOOP_HOME>/lib/kite-data-oss-3.4.0.jar
```

2. 授权指定用户或用户组对JAR包的读写权限。

```
sudo chmod 755 kite-data-oss-3.4.0.jar
```

3. 将OSS数据导入MySQL。

```
sqoop import --connect  <dburi>/<dbname> --username <username> --password <password> --
table <tablename> --target-dir <oss-dir>  --temporary-rootdir <oss-tmpdir> --check-colu
mn <col> --incremental <mode> --last-value <value> -as <format> -m <count>
```

参数说明如下：

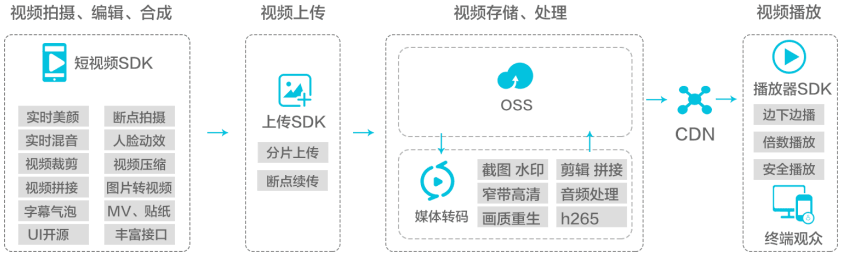
参数	是否必选	说明
dburi	必选	数据库的访问链接，例如 `jdbc:mysql://192.168.xxx.xxx:3306/` 。
dbname	必选	数据库的名称。
username	必选	数据库登录用户名。
password	必选	数据库登录密码。
tablename	必选	MySQL表的名称。
oss-dir	必选	读取或写入OSS-HDFS服务指定路径下的数据，例如 `oss://examplebucket.cn-hangzhou.oss-dls.aliyuncs.com/dir/` 。
oss-tmpdir	可选	临时写入目录。指定mode为append模式时，需要指定该参数。 采用append模式后，Sqoop会先将数据导入临时目录，然后将文件重命名为正常目标目录。如果目标目录已经存在于HDFS中，则Sqoop拒绝导入并覆盖该目录的内容。
col	可选	增量导入场景的检查列。
mode	可选	增量导入模式，支持append和lastmodified两种模式。 ◦ append模式：基于递增列的增量数据导入。 ◦ lastmodified模式：基于时间列的增量数据导入。
value	可选	指定上次增量导入的检查列的最大值。
format	可选	文件存储的格式。取值为*avrodatafile*、*sequencefile*、*textfile*（默认值）、*parquetfile*。
count	可选	指定MapReduce的任务数。

4.内容分发与数据处理

4.1. 音视频

4.1.1. 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，助您快速低成本实现视频萌拍效果。

- 接口简单易用，开放性强

  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。

专业版SDK功能说明

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。

功能	说明
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

- 如需下载阿里云短视频SDK专业版，请参见SDK简介与下载。
- 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）、阿里云账号（如果没有账号请注册）至videosdk@service.aliyun.com，申请开通试用，试用期为一个月。

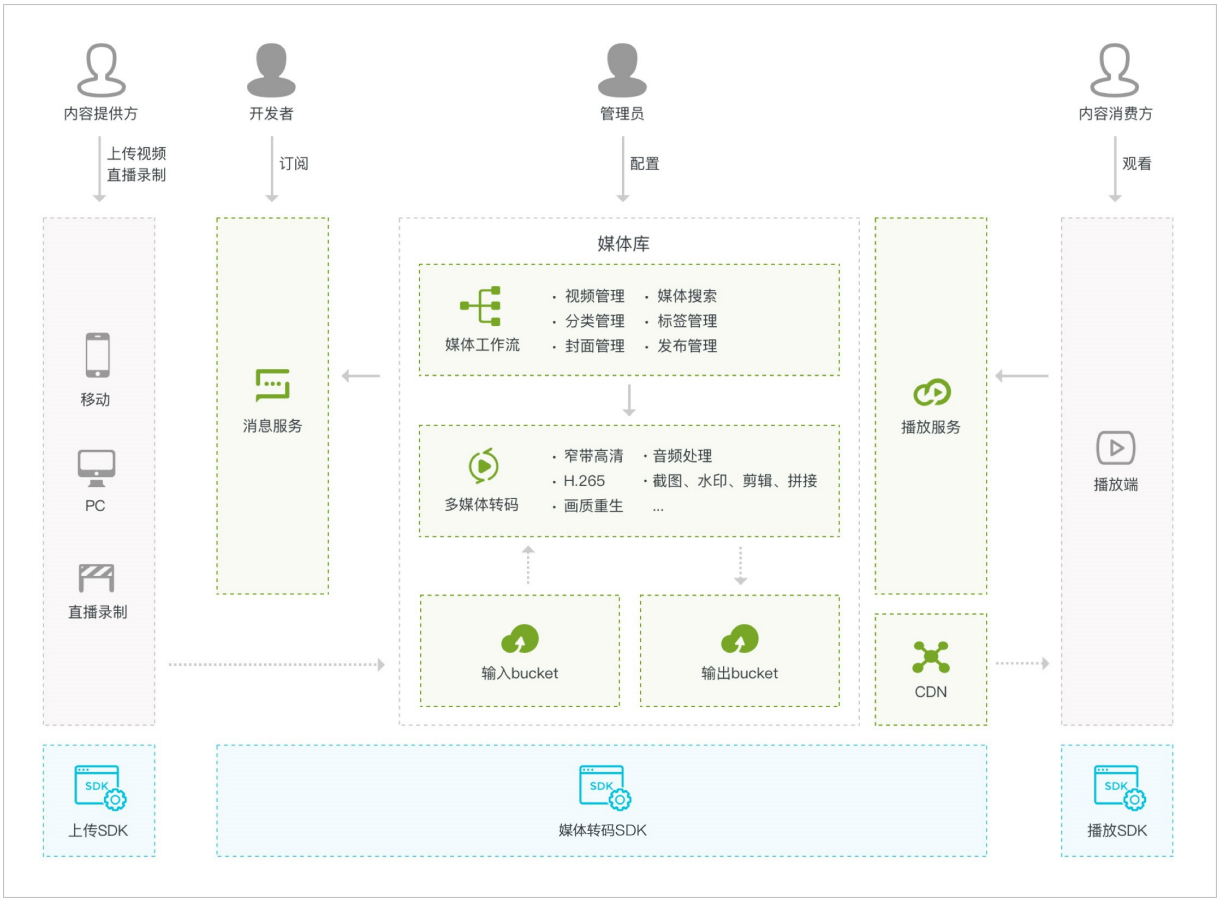
4.1.2. 音视频转码

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。

> **说明** 更多信息请参见功能特性。



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。
华数 wasu	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。

客户	业务介绍
触手	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，详情请参见功能特性。
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上的音频、视频文件的编码和内容信息。
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	阿里云窄带高清技术，以更低的带宽成本，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，4 K大屏播放也无顿挫感。 • 2 K转4 K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，生成高品质4 K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。

功能点	说明
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。
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。

操作步骤

关于音视频转码的具体操作，请参见任务管理。

更多内容

- 媒体处理
- 窄带高清2.0

4.2. 基于OSS构建HLS流

OSS支持以RTMP协议推流音视频至存储空间（Bucket），并转储为HLS协议格式，同时提供了丰富的鉴权、授权机制实现更细颗粒度的音视频数据访问控制。

前提条件

已创建了存储空间。具体操作，请参见创建存储空间。

基础操作

OSS支持使用RTMP推流协议上传H264格式的视频数据和AAC格式的音频数据，并通过访问PlayURL地址的方式获取音视频数据。

- 上传音视频数据
  i. 调用PutLiveChannel接口创建LiveChannel。

调用该接口会返回RTMP推流地址PublishURL和播放地址PlayURL。您还可以使用SDK的方式创建LiveChannel。更多信息，请参见Python SDK 的 LiveChannel 常见操作。

ii. 通过PublishURL推流音视频文件。

OSS将上传的音视频文件按HLS协议转储，即转储为一个m3u8格式的索引文件和多个ts格式的视频文件。具体操作，请参见RTMP推流上传。

- 获取音视频数据

您可以通过浏览器直接访问PlayURL地址的方式，即访问m3u8索引文件的方式来获取音视频数据。

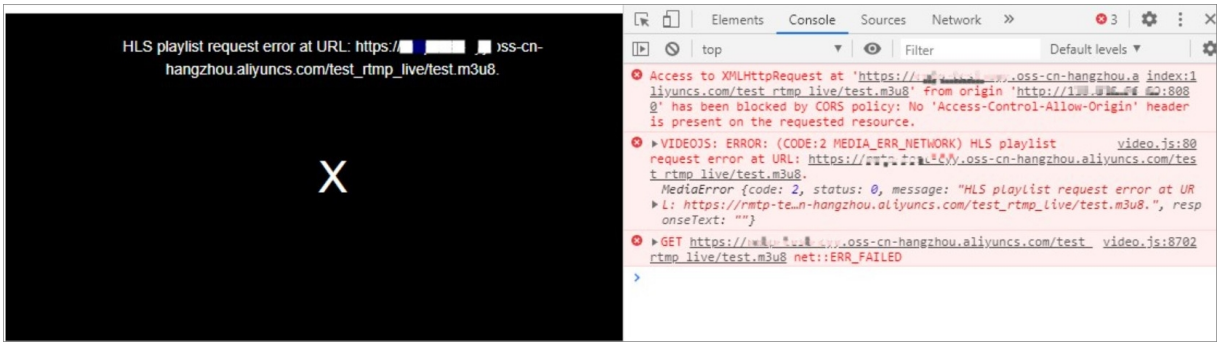
Android和iOS等移动平台，以及PC端的少数浏览器例如Microsoft Edge和Safari，支持通过访问m3u8文件的方式播放视频。Chrome等浏览器需要嵌入Video.js等JavaScript脚本才能正常播放视频。

您在公共读写的Bucket中通过OSS上传和获取音视频，意味着任何人都有权限读写您的音视频数据，从而造成不必要的数据泄露和流量计费等问题。默认情况下，OSS Bucket的访问权限为私有，且拒绝任何来源的跨域请求。推荐您根据自身的使用场景，结合跨域资源共享CORS、防盗链、私有Bucket签名机制中的一种或多种方式，从而有效保护您的数据安全。

跨域资源共享CORS

如果不是通过浏览器直接访问OSS的音视频，而是先访问第三方的网页并在网页中嵌入了OSS的音视频，则很有可能遇到跨域问题，导致视频无法播放。这是因为当Web服务器和OSS不属于同一个域时，将违反同源策略，浏览器默认会拒绝该连接。

例如，Web服务器地址为*http://192.168.xx.xx:8080*，浏览器访问该地址获取了网页，网页的JavaScript脚本里嵌入了视频，视频来源指向OSS Bucket。此时，浏览器需要再次向OSS发送请求，试图访问OSS中的视频数据。但是，浏览器识别到OSS Bucket的访问地址与*http://192.168.xx.xx:8080*的网页地址不属于同一个域，则会先向OSS Bucket询问是否允许跨域请求。OSS Bucket默认不开启CORS配置，因此会拒绝浏览器的跨域请求，使得浏览器不能正常播放音视频。如下图所示：



您可以通过OSS的跨域资源共享，解决因跨域限制导致音视频无法正常播放的问题。

1. 登录OSS管理控制台。
2. 单击Bucket列表，之后单击目标Bucket名称。
3. 单击**权限管理** > **跨域设置**，在**跨域设置**区域单击**设置**。
4. 单击**创建规则**。
5. 在**创建跨域规则**页面配置各项参数。

   结合本文档示例场景，请将**来源**设置为*http://192.168.xx.xx:8080*，其他参数设置请参见设置跨域资源共享。

   - 如果来源为具体的域名，请填写完整的域名，例如*www.example.com*，不可省略为*example.com*。
   - 如果来源为精准的IP地址，请填写包括协议类型和端口号在内的完整的IP地址，例如*http://xx.xx.xx.xx:80*，不可省略为*xx.xx.xx.xx*。

浏览器对跨域配置通常会有数十秒至几分钟的缓存时间。如果希望跨域配置立即生效，建议清空浏览器缓存后刷新网页。

防盗链

CORS规则可以有效阻止其他网站服务器在网页中嵌入您的音视频资源。但如果您通过直接访问OSS Bucket的方式获取音视频，则CORS规则将失效。在这种情况下，您可以通过OSS防盗链对Bucket设置Referer白名单的机制，避免其他人盗取您的音视频资源。

默认情况下，Bucket允许空Referer，通过本地浏览器访问PlayURL的方式可以直接观看视频。为了防止音视频资源被其他人盗用，您可以将Bucket设置为不允许空Referer，并将您信任的域名或IP加入Referer白名单。此时，除Referer白名单以外的第三方访问音视频资源时均会被拒绝，并返回403 Forbidden错误。

1.
2. 单击Bucket列表，之后单击目标Bucket名称。
3. 单击**权限管理 > 防盗链**。
4. 在**防盗链**区域，单击**设置**。
   - 在**Referer**框中，填写域名或IP地址，例如*.aliyun.com。

     您可以结合实际使用场景设置不同的Referer字段。Referer配置示例详情请参见设置防盗链。
   - 在**空Referer**框中，选择不允许空Referer。

     > **说明**
     > - 选择不允许空Referer且设置了Referer白名单，则只有HTTP或HTTPS header中包含Referer字段的请求才能访问OSS资源。
     > - 选择不允许空Referer但未设置Referer白名单，效果等同于允许空Referer，即防盗链设置无效。
5. 单击**保存**。

私有Bucket签名机制

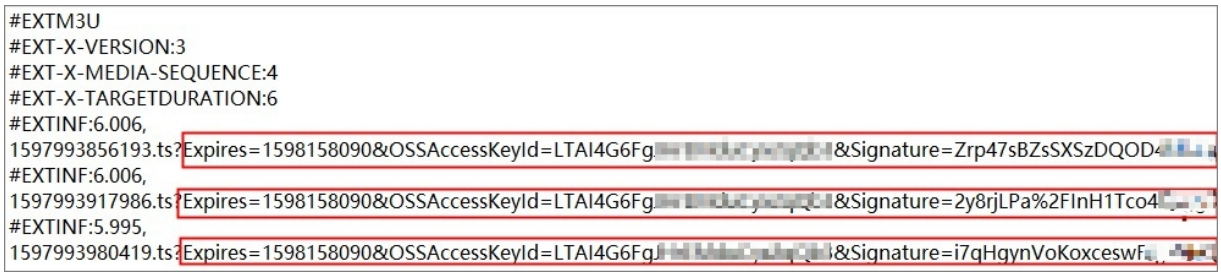
为了保护您的数据安全，OSS默认Bucket的读写权限为私有。因此当您需要向私有Bucket执行读取或写入操作时，需要使用签名机制向OSS声明您的操作权限。

当您向私有Bucket推流时，需要对推流地址进行签名后才能上传音视频文件。具体操作，请参见RTMP推流地址及签名。

使用Python SDK获取签名的推流地址示例如下：

```
import os
import oss2
access_key_id = "your_access_key_id"
access_key_secret = "your_access_key_secret"
bucket_name = "your_bucket_name"
endpoint = "your_endpoint"
# 创建Bucket实例。
bucket = oss2.Bucket(oss2.Auth(access_key_id, access_key_secret), endpoint, bucket_name)
# 创建并配置流频道。
# 该索引文件包含3个ts文件，每个ts文件的时长为5秒。此示例中的5 s时常为建议值，具体的时长取决于关键帧。
channel_name = "your_channel_name"
playlist_name = "your_playlist_name.m3u8"
frag_count_config = 3
frag_duration_config = 5
create_result = bucket.create_live_channel(
        channel_name,
        oss2.models.LiveChannelInfo(
            status = 'enabled',
            description = 'your description here',
            target = oss2.models.LiveChannelInfoTarget(
                playlist_name = playlist_name,
                frag_count = frag_count_config,
                frag_duration = frag_duration_config)))
# 获取RTMP推流签名地址。
# 示例中的expires是一个相对时间，表示从现在开始此次推流过期的秒数。
# 获取签名后的signed_url后即可使用推流工具直接进行推流。一旦连接上OSS，即使超出上面设置的expires也不会断流，OSS仅在每次推流连接时检查expires是否合法。
signed_rtmp_url = bucket.sign_rtmp_url(channel_name, playlist_name, expires=3600)
print(signed_rtmp_url)
```

OSS访问私有Bucket中的文件时，需要在URL中加上签名。HLS的访问机制为动态访问m3u8索引文件，根据索引文件的内容多次请求下载最新的ts文件，此过程中的每一次请求都需要在URL中加上签名。



为此，OSS对音视频数据的访问提供了动态签名机制，即只需在首次访问m3u8文件时在URL中添加`x-oss-process=hls/sign`，OSS将对返回的播放列表中的所有ts地址自动按照与m3u8完全相同的方式进行签名。

使用Python SDK动态签名机制访问音视频的示例如下：

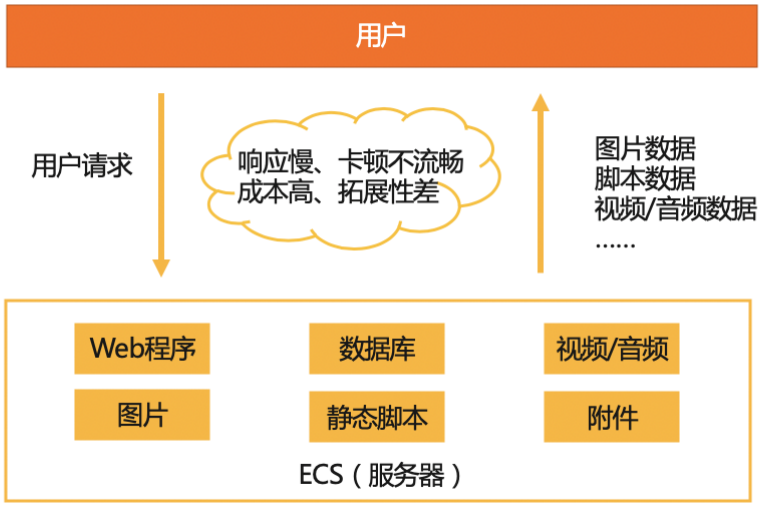
```
# 获取观流的动态签名地址。
your_object_name = "test_rtmp_live/test.m3u8"
style = "hls/sign"
# 生成签名URL时，OSS默认会对Object完整路径中的正斜线（/）进行转义，从而导致生成的签名URL无法直接使用。
# 设置slash_safe为True，OSS不会对Object完整路径中的正斜线（/）进行转义，此时生成的签名URL可以直接使用。
signed_download_url = bucket.sign_url('GET', your_object_name, 3600, params={'x-oss-process': style}, slash_safe=True)
print(signed_download_url)
```

4.3. 使用CDN加速OSS访问

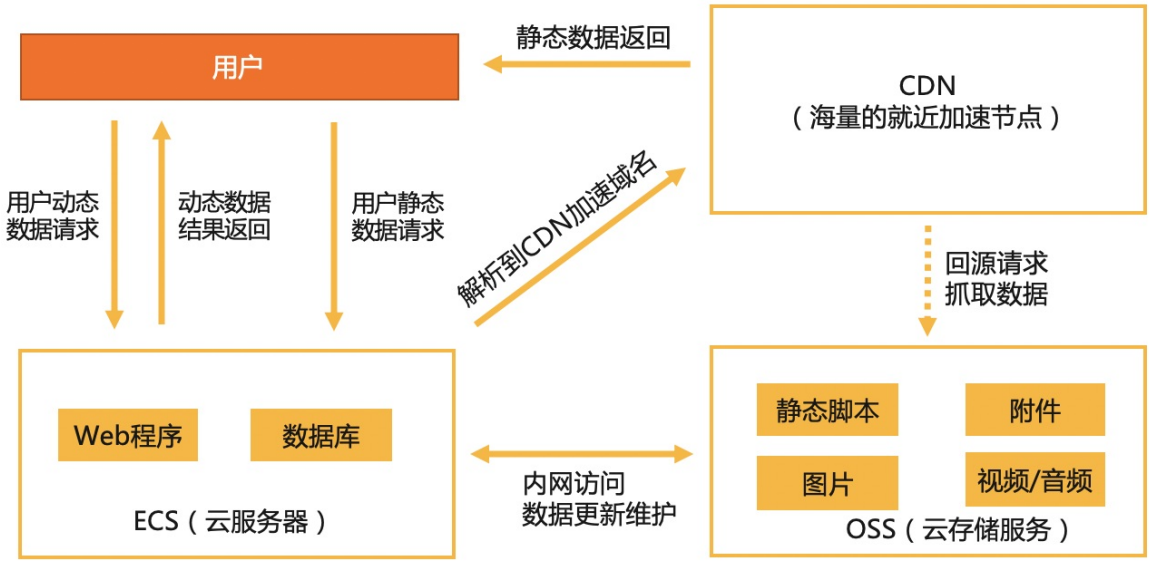
用户直接访问OSS资源，访问速度会受到OSS的下行带宽以及Bucket地域的限制。如果通过CDN来访问OSS资源，带宽上限更高，并且可以将OSS的资源缓存至就近的CDN节点，通过CDN节点进行分发，访问速度更快，且费用更低。本文介绍如何使用CDN来加速OSS的访问。

背景信息

传统网站架构下，动态资源和静态资源不分离，随着访问量的增长，性能会成为瓶颈，如下图所示：



如果采用动静分离的网站架构，就能够解决海量用户访问的性能瓶颈问题，如下图所示：



该架构的要点如下：

- 将动态资源如Web程序、数据库等存放在云服务器ECS上。
- 将静态资源如图片、音视频、静态脚本等存放在对象存储OSS上。
- 将OSS作为CDN的源站，通过CDN加速分发，使用户通过CDN节点就近获得文件。

该架构有以下优势：

- 降低了Web服务器负载。

  OSS的资源缓存至就近的CDN节点，通过CDN节点进行分发，缩短了网络传输距离，加快了用户的调用速度。

- 支持海量存储。

  OSS的存储空间弹性无限扩展，您无需考虑存储架构升级。

- 降低了存储费用和流量费用。

  使用该架构会产生OSS的存储费用、CDN的下行流量费用，以及极少量的回源流量费用。其中OSS的存储费用仅为ECS云盘费用的一半，而CDN流量的单价约为OSS外网流量单价的30%～40%。

前提条件

- 已创建一个OSS Bucket，且上传了相关资源。详情请参见上传文件。
- 已开通阿里云CDN服务。详情请参见开通CDN服务。

操作步骤

以下步骤以域名example.com为例，加速域名以oss.example.com为例。您可以根据自己的实际情况来选择加速域名，包括主域名、二级域名、泛域名等。

1. 添加域名。

   i. 登录CDN管理控制台，选择**域名管理**。

ii. 单击**添加域名**，设置以下参数：

- **加速域名**：输入加速域名，该示例为**oss.example.com**。
- **业务类型**：选择**图片小文件**。
- **加速区域**：选择**仅中国内地**。
- **源站信息**：单击**新增源站信息**，然后选择**OSS域名**和需要加速的OSS域名（即之前创建的OSS Bucket对应的域名），其他参数保持默认值。单击**确认**。

iii. 单击**下一步**，然后单击**返回域名列表**。

iv. 等到域名状态为**正常运行**时，复制CNAME值，该示例为**oss.example.com.w.kunluncan.com**。

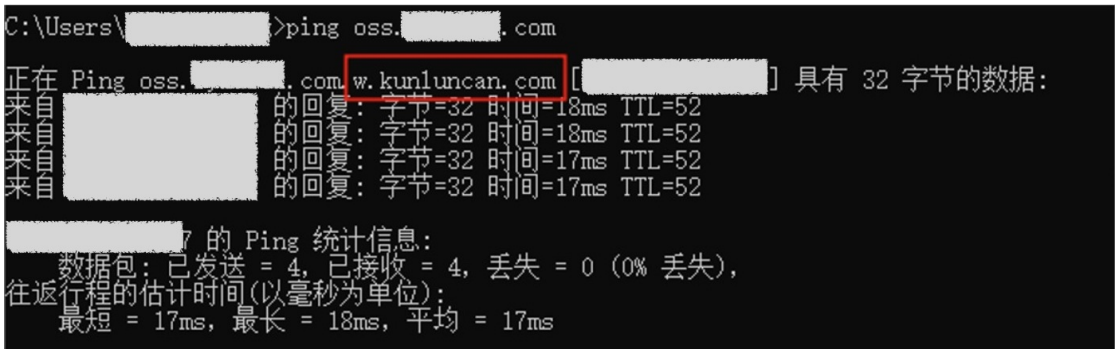
2. 解析域名。

i. 进入域名控制台，找到域名**example.com**，单击**解析**。

ii. 在**添加记录**页面，配置以下参数：

- **记录类型**：选择**CNAME**。
- **主机记录**：输入**oss**。
- **记录值**：输入之前复制的CNAME值**oss.example.com.w.kunluncan.com**。
- 其他参数：保留默认值。

iii. 单击**确认**。等待几分钟后，使用ping命令查看加速域名是否生效。下图表示已生效。



3. 开启CDN缓存自动刷新。

i. 进入OSS控制台，单击左侧导航栏的**Bucket列表**，然后选择对应的Bucket。

ii. 选择**传输管理**，然后选择**域名管理**。

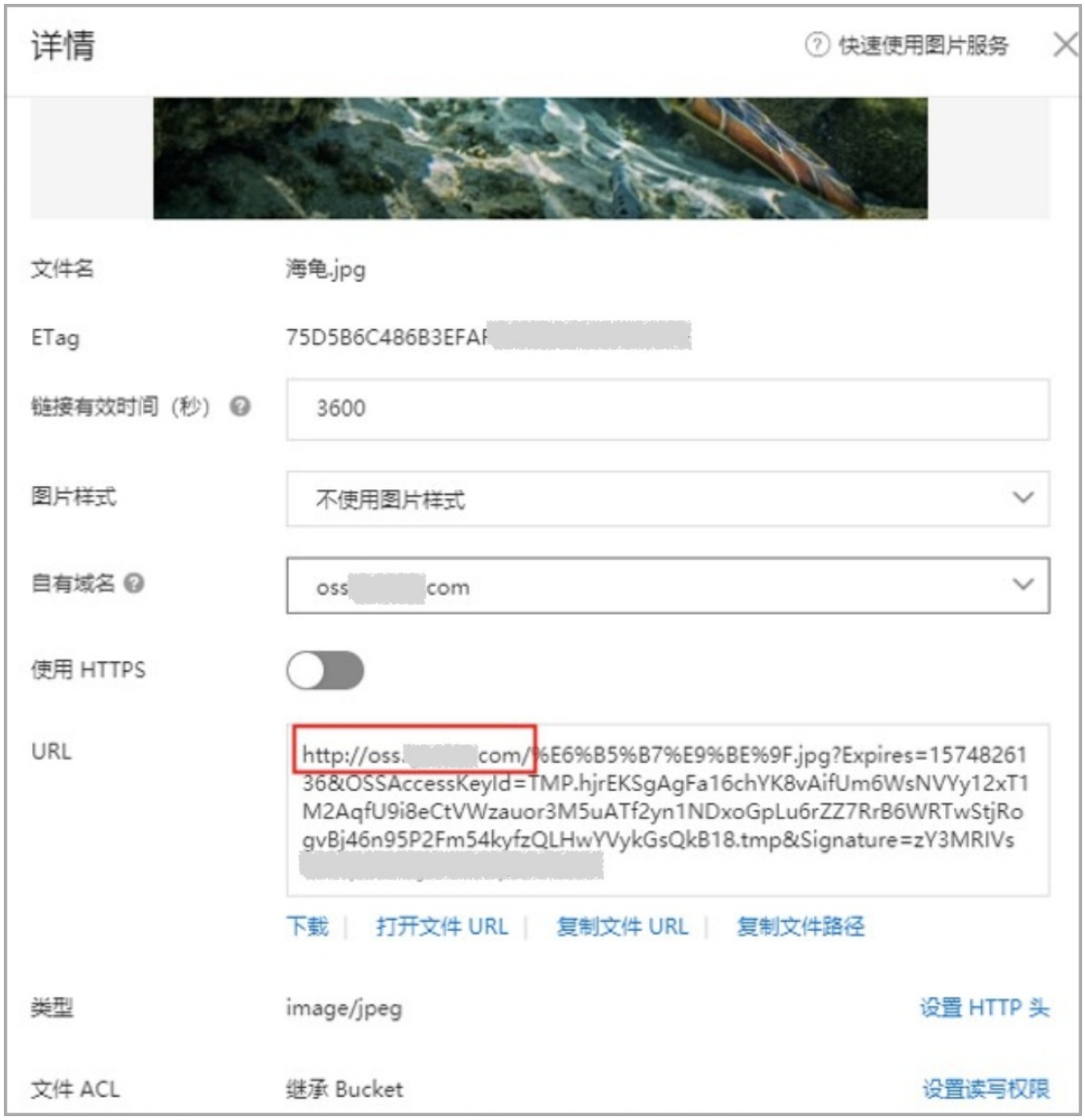
iii. 开启加速域名对应的**CDN缓存自动刷新**。

4. 查看文件的URL。

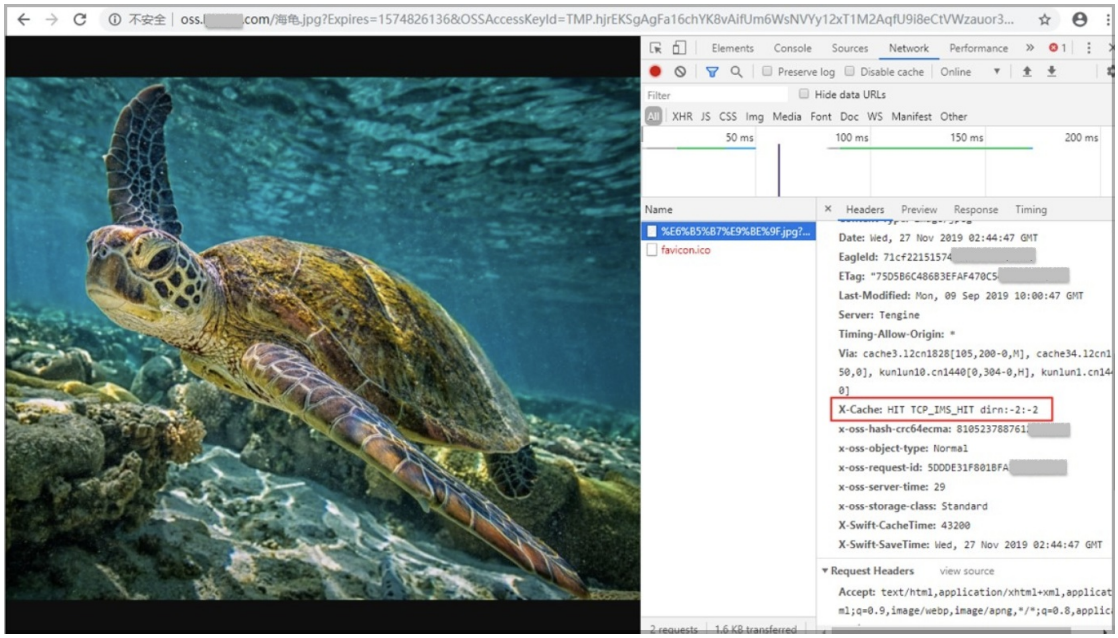
i. 进入OSS控制台，单击左侧导航栏的**Bucket列表**，然后选择对应的Bucket。

ii. 进入**文件管理**，然后单击文件对应的**详情**，进入文件的**详情**页面。

iii. 在文件的**详情**页面，从**自有域名**列表中选择加速域名，该示例为**oss.example.com**。可以看到文件的URL已经变为加速域名开头的URL。



iv. 直接访问上述的URL，通过开发者工具检查可以发现，CDN已经生效并成功缓存了这张图片。



5. 使文件的URL长期有效。
   i. 在文件的**详情**页面，单击**设置读写权限**。
   ii. 选择**公共读**，然后单击**确定**。
6. （可选）配置证书加密访问。
   i. 在文件的**详情**页面，开启**使用HTTPS**。
   ii. 在CDN管理控制台，选择**域名管理**，然后单击加速域名。
   iii. 在左侧导航栏，单击**HTTPS配置**，然后在**HTTPS证书**区域单击**修改配置**。
   iv. 完成配置后即可通过HTTPS加密访问，具体步骤请参见配置HTTPS证书。

购买链接

要进一步降低费用，请单击OSS资源套餐包和CDN资源套餐包购买相关折扣套餐。

4.4. Java SDK的LiveChannel常见操作

本文介绍Java SDK的LiveChannel常见操作，如创建LiveChannel、列举LiveChannel及删除LiveChannel等。

> **说明**
>
> 本文示例由阿里云用户bin提供，仅供参考。

创建LiveChannel

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

> **说明** 您可以使用返回的地址进行推流、播放，您还可以根据该LiveChannel的名称来发起相关的操作，如查询推流状态、查询推流记录、禁止推流等。

以下代码用于创建 LiveChannel:

```
public static void createLiveChannel() {
        String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
        // 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维，请登录RAM控制台创建RAM用户。
        String accessKeyId = "<yourAccessKeyId>";
        String accessKeySecret = "<yourAccessKeySecret>";
        String liveChannelName = "<yourLiveChannelName>";
        // 创建OSSClient实例。
        OSS oss = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
        CreateLiveChannelRequest request = new CreateLiveChannelRequest(bucketName,
                liveChannelName, "desc", LiveChannelStatus.Enabled, new LiveChannelTarget()
);
        CreateLiveChannelResult result = oss.createLiveChannel(request);
        //获取推流地址。
        List<String> publishUrls = result.getPublishUrls();
        for (String item : publishUrls) {
            System.out.println(item);
        }
        //获取播放地址。
        List<String> playUrls = result.getPlayUrls();
        for (String item : playUrls) {
            System.out.println(item);
        }
        oss.shutdown();
    }
```

创建LiveChannel详情，请参见PutLiveChannel。

列举LiveChannel

以下代码用于列举指定的LiveChannel:

```
public static void listLiveChannels() {
        String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
        // 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维，请登录RAM控制台创建RAM用户。
        String accessKeyId = "<yourAccessKeyId>";
        String accessKeySecret = "<yourAccessKeySecret>";
        String bucketName = "<yourBucketName>";
        // 创建OSSClient实例。
        OSS oss = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
        ListLiveChannelsRequest request = new ListLiveChannelsRequest(bucketName);
        LiveChannelListing liveChannelListing = oss.listLiveChannels(request);
        System.out.println(JSON.toJSONString(liveChannelListing));
        oss.shutdown();
    }
```

列举LiveChannel详情，请参见ListLiveChannel。

删除LiveChannel

> **说明**
>
> - 当有客户端正在向LiveChannel推流时，删除请求会失败。
> - DeleteLiveChannel接口只会删除LiveChannel本身，不会删除推流生成的文件。

以下代码用于删除指定的LiveChannel：

```
public static void deleteLiveChannel() {
        String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
        // 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维，请登录RAM控制台创建RAM用户。
        String accessKeyId = "<yourAccessKeyId>";
        String accessKeySecret = "<yourAccessKeySecret>";
        String bucketName = "<yourBucketName>";
        // 创建OSSClient实例。
        OSS oss = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
        LiveChannelGenericRequest request = new LiveChannelGenericRequest(bucketName, liveC
hannelName);
        try {
            oss.deleteLiveChannel(request);
        } catch (OSSException ex) {
            ex.printStackTrace();
        } catch (ClientException ex) {
            ex.printStackTrace();
        } finally {
            oss.shutdown();
        }
    }
```

删除LiveChannel详情，请参见DeleteLiveChannel。

设置LiveChannel状态

LiveChannel有enabled和disabled两种状态供您选择。

以下代码用于设置LiveChannel状态：

```
public static void setLiveChannelStatus() {
        String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
        // 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维，请登录RAM控制台创建RAM用户。
        String accessKeyId = "<yourAccessKeyId>";
        String accessKeySecret = "<yourAccessKeySecret>";
        String liveChannelName = "<yourLiveChannelName>";
        String bucketName = "<yourBucketName>";
        // 创建OSSClient实例。
        OSS oss = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
        try {
           oss.setLiveChannelStatus(bucketName, liveChannelName, LiveChannelStatus.Enabled);
        } catch (OSSException ex) {
            System.out.println(ex.getErrorCode().concat(",").concat(ex.getErrorMessage()));
        } catch (ClientException ex) {
            System.out.println(ex.getErrorCode().concat(",").concat(ex.getErrorMessage()));
        } finally {
            oss.shutdown();
        }
    }
```

设置LiveChannel状态详情，请参见PutLiveChannelStatus。

获取LiveChannel状态信息

以下代码用于获取指定LiveChannel的推流状态信息。

```
public static void getLiveChannelStat() {
        String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
        // 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维，请登录RAM控制台创建RAM用户。
        String accessKeyId = "<yourAccessKeyId>";
        String accessKeySecret = "<yourAccessKeySecret>";
        String liveChannelName = "<yourLiveChannelName>";
        String bucketName = "<yourBucketName>";
         // 创建OSSClient实例。
        OSS oss = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
        LiveChannelStat liveChannelStat = oss.getLiveChannelStat(bucketName, liveChannelName);
        System.out.println(liveChannelStat.toString());
        oss.shutdown();
    }
```

获取LiveChannel状态信息详情，请参见GetLiveChannelStat。

获取LiveChannel配置信息

以下代码用于获取指定LiveChannel 的配置信息：

```
public static void getLiveChannelInfo() {
        String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
        // 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维，请登录RAM控制台创建RAM用户。
        String accessKeyId = "<yourAccessKeyId>";
        String accessKeySecret = "<yourAccessKeySecret>";
        String bucketName = "<yourBucketName>";
        String liveChannelName = "<yourLiveChannelName>";
        // 创建OSSClient实例。
        OSS oss = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
        LiveChannelInfo liveChannelInfo = oss.getLiveChannelInfo(bucketName, liveChannelName);
        System.out.println(JSON.toJSONString(liveChannelInfo));
        oss.shutdown();
    }
```

获取LiveChannel配置信息详情，请参见GetLiveChannelInfo。

生成LiveChannel播放列表

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

以下代码用于生成LiveChannel播放列表：

```
public static void postVodPlaylist() {
        String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
        // 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维，请登录RAM控制台创建RAM用户。
        String accessKeyId = "<yourAccessKeyId>";
        String accessKeySecret = "<yourAccessKeySecret>";
        String liveChannelName = "<yourLiveChannelName>";
        String bucketName = "<yourBucketName>";
        String playListName = "<yourPlayListName>";
        // 创建OSSClient实例。
        OSS oss = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
        long startTime = getUnixTimestamp("2019-06-27 23:00:00");
        long endTIme = getUnixTimestamp("2019-06-28 22:00:00");
        try {
            oss.generateVodPlaylist(bucketName, liveChannelName, playListName, startTime, endTIme);
        } catch (OSSException ex) {
            System.out.println(ex.getErrorCode().concat(",").concat(ex.getErrorMessage()));
        } catch (ClientException ex) {
            System.out.println(ex.getErrorCode().concat(",").concat(ex.getErrorMessage()));
        } finally {
            oss.shutdown();
        }
    }
    private static long getUnixTimestamp(String time) {
        DateFormat format = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
        try {
            Date date = format.parse(time);
            return date.getTime() / 1000;
        } catch (ParseException e) {
            e.printStackTrace();
            return 0;
        }
    }
```

生成LiveChannel播放列表详情，请参见PostVodPlaylist。

查看LiveChannel播放列表

以下代码用于查看指定LiveChannel推流生成的、且指定时间段内的播放列表：

```
public static void getVodPlaylist() {
        String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
        // 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维，请登录RAM控制台创建RAM用户。
        String accessKeyId = "<yourAccessKeyId>";
        String accessKeySecret = "<yourAccessKeySecret>";
        String liveChannelName = "<yourLiveChannelName>";
        String bucketName = "<yourBucketName>";
        // 创建OSSClient实例。
        OSS oss = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
        long startTime = getUnixTimestamp("2019-06-27 23:00:00");
        long endTIme = getUnixTimestamp("2019-06-28 22:00:00");
        try {
            OSSObject ossObject = oss.getVodPlaylist(bucketName, liveChannelName, startTime, endTIme);
            System.out.println(ossObject.toString());
        } catch (OSSException ex) {
            System.out.println(ex.getErrorCode().concat(",").concat(ex.getErrorMessage()));
        } catch (ClientException ex) {
            System.out.println(ex.getErrorCode().concat(",").concat(ex.getErrorMessage()));
        } finally {
            oss.shutdown();
        }
    }
    private static long getUnixTimestamp(String time) {
        DateFormat format = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
        try {
            Date date = format.parse(time);
            return date.getTime() / 1000;
        } catch (ParseException e) {
            e.printStackTrace();
            return 0;
        }
    }
```

查看LiveChannel播放列表详情，请参见GetVodPlaylist。

获取LiveChannel推流记录

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

以下代码用于获取LiveChannel推流记录：

```
public void getLiveChannelHistory() {
        String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
        // 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维，请登录RAM控制台创建RAM用户。
        String accessKeyId = "<yourAccessKeyId>";
        String accessKeySecret = "<yourAccessKeySecret>";
        String bucketName = "<yourBucketName>";
        String liveChannelName = "<yourLiveChannelName>";
        // 创建OSSClient实例。
        OSS oss = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
        List<LiveRecord> list = oss.getLiveChannelHistory(bucketName, liveChannelName);
        System.out.println(JSON.toJSONString(list));
        oss.shutdown();
    }
```

获取LiveChannel推流记录详情，请参见GetLiveChannelHistory。

4.5. Python SDK 的 LiveChannel 常见操作

本文介绍 Python SDK 的 LiveChannel 常见操作，如创建 LiveChannel、列举 LiveChannel 及删除 LiveChannel 等。

> **说明**
>
> 本文示例由阿里云用户 fralychen 提供，仅供参考。

环境准备

- Python 3.6

  > **说明** 本文档示例基于Python 3.6版本编写，同样适用于Python 2.6、2.7、3.3、3.4、3.5版本。

- aliyun-oss-python-sdk 2.9.0
- OBS Studio推流工具
- IDE

有关 LiveChannel 详情，请参见LiveChannel简介、live_channel.py以及api.py。

创建 LiveChannel

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

> **说明** 您可以使用返回的地址进行推流、播放，您还可以根据该LiveChannel的名称来发起相关的操作，如查询推流状态、查询推流记录、禁止推流等。

以下代码用于创建 LiveChannel:

```
import os
import oss2
access_key_id = os.getenv('OSS_TEST_ACCESS_KEY_ID', '**')
access_key_secret = os.getenv('OSS_TEST_ACCESS_KEY_SECRET', '***')
bucket_name = os.getenv('OSS_TEST_BUCKET', '********')
endpoint = os.getenv('OSS_TEST_ENDPOINT', '***')
# 创建Bucket实例。
bucket = oss2.Bucket(oss2.Auth(access_key_id, access_key_secret), endpoint, bucket_name)
# 创建并配置流频道。
# 频道的名称是test_rtmp_live。直播生成的m3u8文件叫做test.m3u8，该索引文件包含3片ts文件，每片ts文件的时长为5秒（这只是一个建议值，具体的时长取决于关键帧）。
channel_name = "test_rtmp_live"
playlist_name = "test.m3u8"
create_result = bucket.create_live_channel(
        channel_name,
        oss2.models.LiveChannelInfo(
            status = 'enabled',
            description = '测试使用的直播频道',
            target = oss2.models.LiveChannelInfoTarget(
                playlist_name = playlist_name,
                frag_count = 3,
                frag_duration = 5)))
```

创建LiveChannel详情，请参见PutLiveChannel。

列举和删除 LiveChannel

以下代码用于列举和删除 LiveChannel：

```
import os
import oss2
access_key_id = os.getenv('OSS_TEST_ACCESS_KEY_ID', '**')
access_key_secret = os.getenv('OSS_TEST_ACCESS_KEY_SECRET', '***')
bucket_name = os.getenv('OSS_TEST_BUCKET', '********')
endpoint = os.getenv('OSS_TEST_ENDPOINT', '***')
# 创建Bucket实例。
bucket = oss2.Bucket(oss2.Auth(access_key_id, access_key_secret), endpoint, bucket_name)
# 列举符合规则的LiveChannel。
# 列举出Bucket下所有符合条件的livechannel。
# param: prefix (类型: str)表示要列举的livechannel名称的前缀，不指定则列举所有的livechannel。
# return: class:`ListLiveChannelResult <oss2.models.ListLiveChannelResult>`
for info in oss2.LiveChannelIterator(bucket, prefix="test"):
    print(info.name)
# 删除LiveChannel。
bucket.delete_live_channel(info.name)
```

列举 LiveChannel 的更多详情，请参见ListLiveChannel。删除 LiveChannel 的更多详情，请参见DeleteLiveChannel。

设置 LiveChannel 状态

LiveChannel有enabled和disabled两种状态供您选择。

以下代码用于设置LiveChannel状态：

```
import os
import oss2
access_key_id = os.getenv('OSS_TEST_ACCESS_KEY_ID', '**')
access_key_secret = os.getenv('OSS_TEST_ACCESS_KEY_SECRET', '***')
bucket_name = os.getenv('OSS_TEST_BUCKET', '********')
endpoint = os.getenv('OSS_TEST_ENDPOINT', '***')
# 创建Bucket实例。
bucket = oss2.Bucket(oss2.Auth(access_key_id, access_key_secret), endpoint, bucket_name)
# 打开或关闭流频道。
bucket.put_live_channel_status(channel_name, 'enabled')
bucket.put_live_channel_status(channel_name, 'disabled')
```

设置LiveChannel状态详情，请参见PutLiveChannelStatus。

获取 RTMP 推流地址及签名

以下代码用于获取 RTMP 推流地址及签名：

```
import os
import oss2
access_key_id = os.getenv('OSS_TEST_ACCESS_KEY_ID', '**')
access_key_secret = os.getenv('OSS_TEST_ACCESS_KEY_SECRET', '***')
bucket_name = os.getenv('OSS_TEST_BUCKET', '********')
endpoint = os.getenv('OSS_TEST_ENDPOINT', '***')
# 创建Bucket实例。
bucket = oss2.Bucket(oss2.Auth(access_key_id, access_key_secret), endpoint, bucket_name)
# 获取推流和观流地址。
# 创建直播频道之后拿到推流用的play_url（rtmp推流的url，如果Bucket ACL为非公共读写，则需要带上签名，见下文示例）和观流用的publish_url（推流产生的m3u8文件的url）。
# 通过create_live_channel()获取create_result。
publish_url = create_result.publish_url
play_url = create_result.play_url
print("推流地址:",publish_url)
print("观流地址:",play_url)
# 拿到推流地址和观流地址之后就可以向OSS推流和观流。如果Bucket ACL为非公共读写，则需要对推流做签名，如果Bucket ACL为公共读写，则可以直接用publish_url推流。
# 这里的expires是一个相对时间，表示从现在开始此次推流过期的秒数。
# 所有的参数都会参与签名。
# 拿到签名后的signed_url就可以使用推流工具直接进行推流。一旦连接上OSS之后即使超出上面设置的expires也不会断流，OSS仅在每次推流连接的时候检查expires是否合法。
signed_url = bucket.sign_rtmp_url(channel_name, playlist_name, expires=3600)
print(signed_url)
```

获取 LiveChannel 状态信息

以下代码用于获取指定LiveChannel的推流状态信息。

```
import os
import oss2
access_key_id = os.getenv('OSS_TEST_ACCESS_KEY_ID', '**')
access_key_secret = os.getenv('OSS_TEST_ACCESS_KEY_SECRET', '***')
bucket_name = os.getenv('OSS_TEST_BUCKET', '********')
endpoint = os.getenv('OSS_TEST_ENDPOINT', '***')
# 创建Bucket实例。
bucket = oss2.Bucket(oss2.Auth(access_key_id, access_key_secret), endpoint, bucket_name)
# 查看当前流的状态信息。
get_status = bucket.get_live_channel_stat(channel_name)
print("连接时间:",get_status.connected_time)
print("推流客户端的IP:",get_status.remote_addr )
print("推流状态:",get_status.status )
```

获取LiveChannel状态信息详情，请参见GetLiveChannelStat。

生成并查看播放列表

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

以下代码用于生成并查看播放列表：

```
import os
import oss2
import time
access_key_id = os.getenv('OSS_TEST_ACCESS_KEY_ID', '**')
access_key_secret = os.getenv('OSS_TEST_ACCESS_KEY_SECRET', '***')
bucket_name = os.getenv('OSS_TEST_BUCKET', '********')
endpoint = os.getenv('OSS_TEST_ENDPOINT', '***')
# 创建Bucket实例
bucket = oss2.Bucket(oss2.Auth(access_key_id, access_key_secret), endpoint, bucket_name)
end_time = int(time.time())
start_time = end_time - 3600
generate_playlist = "my_vod_list.m3u8"
# 生成点播列表。
# 如果希望利用直播推流产生的ts文件生成一个点播列表，可以使用post_vod_playlist方法。
# 本示例指定起始时间为当前时间减去3600秒，结束时间为当前时间，即表示将最近一个小时的推流生成一个播放列表。
# 这个接口调用成功之后会在OSS上生成一个名为“my_vod_list.m3u8”的播放列表文件。
bucket.post_vod_playlist(
                channel_name,
                playlist_name,
                start_time = start_time,
                end_time = end_time)
# 如果想查看指定时间段内的播放列表内容，可以使用get_vod_playlist。
result = bucket.get_vod_playlist(channel_name, start_time=start_time, end_time=end_time)
print("playlist:", result.playlist)
```

生成LiveChannel播放列表详情，请参见PostVodPlaylist。

查看LiveChannel播放列表详情，请参见GetVodPlaylist。

获取 LiveChannel 配置信息

以下代码用于获取指定LiveChannel 的配置信息：

```
import os
import oss2
access_key_id = os.getenv('OSS_TEST_ACCESS_KEY_ID', '**')
access_key_secret = os.getenv('OSS_TEST_ACCESS_KEY_SECRET', '***')
bucket_name = os.getenv('OSS_TEST_BUCKET', '********')
endpoint = os.getenv('OSS_TEST_ENDPOINT', '***')
# 创建Bucket实例。
bucket = oss2.Bucket(oss2.Auth(access_key_id, access_key_secret), endpoint, bucket_name)
#获取LiveChannel的配置信息。
get_result = bucket.get_live_channel(channel_name)
print("-------------------")
print("推流配置信息:")
print(get_result.description)
print(get_result.status)
print(get_result.target.type)
print(get_result.target.frag_count)
print(get_result.target.frag_duration)
print(get_result.target.playlist_name)
print("-------------------")
```

获取LiveChannel配置信息详情，请参见GetLiveChannelInfo。

获取 LiveChannel 推流记录

使用 get_live_channel_history 接口最多会返回指定 LiveChannel 最近的 10 次推流记录。以下代码用于获取 LiveChannel 推流记录：

```
import os
import oss2
access_key_id = os.getenv('OSS_TEST_ACCESS_KEY_ID', '**')
access_key_secret = os.getenv('OSS_TEST_ACCESS_KEY_SECRET', '***')
bucket_name = os.getenv('OSS_TEST_BUCKET', '********')
endpoint = os.getenv('OSS_TEST_ENDPOINT', '***')
# 创建Bucket实例。
bucket = oss2.Bucket(oss2.Auth(access_key_id, access_key_secret), endpoint, bucket_name)
# 查看一个频道历史推流记录。
history_result = bucket.get_live_channel_history(channel_name)
print("推流历史次数:",len(history_result.records))
```

获取LiveChannel推流记录详情，请参见GetLiveChannelHistory。

FAQ

- 推流状态、客户端 IP、连接时间等信息为何获取不了？

  get_live_channel_stat 获取推流状态信息要求对应的频道（channel_name）处于 Live 状态，即连接上推流地址后客户端正处于推流中的状态。

- .get_live_channel_history能否获取历史推流的起止时间和远端地址？

  可以。详情请参见GetLiveChannelHistory。

- 通过 list_live_channel 获取到的频道信息是什么类型的？

  字符串。详情请参见ListLiveChannel。

- 生成点播列表 post_vod_playlist 函数中 end_time 参数所需要的格式是什么？

  整数。详情请参见PostVodPlaylist。

- 报错【'Code': 'InvalidArgument', 'Message': 'No ts file found in specified time span.'】。

  已上传推流文件后才能生成点播列表。

5.数据分析

5.1. 使用 Java SDK 的 SelectObject 查询 CSV 和 JSON 文件

本文介绍如何使用 Java SDK 的 SelectObject 查询 CSV 和 JSON 文件。

> **说明**
>
> 本文示例由阿里云用户 bin 提供，仅供参考。

以下代码用于查询 CSV 文件和 JSON 文件：

```
 private static final String csvKey = "test.csv";
 // Endpoint 以杭州为例，其它 Region 请按实际情况填写。
 private static final String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
 // 阿里云主账号 AccessKey 拥有所有 API 的访问权限，风险很高。
 // 强烈建议您创建并使用 RAM 账号进行 API 访问或日常运维，请登录 https://ram.console.aliyun.com 创建 RAM 账号。
 private static final String accessKeyId = "<yourAccessKeyId>";
 private static final String accessKeySecret = "<yourAccessKeySecret>";
 private static final String bucketName = "<yourBucketName>";
    /**
    * CreateSelectObjectMeta API 用于获取目标文件总的行数，总的列数（对于 CSV 文件），以及 Splits 个数。
    */
    public static void createCsvSelectObjectMetadata() {
      // 创建 OSSClient 实例。
      OSS client = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
      String content = "name,school,company,age\r\n" +
           "Lora Francis,School A,Staples Inc,27\r\n" +
           "Eleanor Little,School B,\"Conectiv, Inc\",43\r\n" +
           "Rosie Hughes,School C,Western Gas Resources Inc,44\r\n" +
           "Lawrence Ross,School D,MetLife Inc.,24";
      client.putObject(bucketName, csvKey, new ByteArrayInputStream(content.getBytes()));
       SelectObjectMetadata selectObjectMetadata = client.createSelectObjectMetadata(
            new CreateSelectObjectMetadataRequest(bucketName, csvKey)
               .withInputSerialization(
                    new InputSerialization().withCsvInputFormat(
                        new CSVFormat().withHeaderInfo(CSVFormat.Header.Use)
                            .withRecordDelimiter("\r\n"))));
       //获取 csv 的总列数。
       System.out.println(selectObjectMetadata.getCsvObjectMetadata().getTotalLines());
       //获取 csv 的 splits 个数。
       System.out.println(selectObjectMetadata.getCsvObjectMetadata().getSplits());
       client.shutdown();
   }
    /**
    * 查询 csv。
    */
    public static void selectCsv() {
```

```
        // 创建 OSSClient 实例。
        OSS client = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
        try {
            //保存 Select 请求的容器。
            SelectObjectRequest selectObjectRequest =
                new SelectObjectRequest(bucketName, csvKey)
                    .withInputSerialization(
                        new InputSerialization().withCsvInputFormat(
                            new CSVFormat().withHeaderInfo(CSVFormat.Header.Use)
                                .withRecordDelimiter("\r\n")))
                    .withOutputSerialization(new OutputSerialization().withCsvOutputFormat(new
CSVFormat()));
            // ossobject 不可修改，查询第 4 列值大于 40 的数据，会直接进行隐式转换。
            selectObjectRequest.setExpression("select * from ossobject where _4 > 40");
            OSSObject ossObject = client.selectObject(selectObjectRequest);
            writeToFile(ossObject.getObjectContent(), "result.csv");
        } catch (OSSException ex) {
            System.out.println(ex.getErrorCode().concat(",").concat(ex.getErrorMessage()));
        } catch (ClientException ex) {
            System.out.println(ex.getErrorCode().concat(",").concat(ex.getErrorMessage()));
        } finally {
            client.shutdown();
        }
    }
    /**
     * 查询简单 json。
     */
    public static void selectSimpleJson() {
        String key = "simple.json";
        // 创建 OSSClient 实例。
        OSS client = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
        final String content = "{\n" +
                "\t\"name\": \"Lora Francis\",\n" +
                "\t\"age\": 27,\n" +
                "\t\"company\": \"Staples Inc\"\n" +
                "}\n" +
                "{\n" +
                "\t\"name\": \"Eleanor Little\",\n" +
                "\t\"age\": 43,\n" +
                "\t\"company\": \"Conectiv, Inc\"\n" +
                "}\n" +
                "{\n" +
                "\t\"name\": \"Rosie Hughes\",\n" +
                "\t\"age\": 44,\n" +
                "\t\"company\": \"Western Gas Resources Inc\"\n" +
                "}\n" +
                "{\n" +
                "\t\"name\": \"Lawrence Ross\",\n" +
                "\t\"age\": 24,\n" +
                "\t\"company\": \"MetLife Inc.\"\n" +
                "}";
        try {
            client.putObject(bucketName, key, new ByteArrayInputStream(content.getBytes()))
;
```

```
            SelectObjectRequest selectObjectRequest =
                 new SelectObjectRequest(bucketName, key)
                     .withInputSerialization(new InputSerialization()
                          .withCompressionType(CompressionType.NONE)
                          .withJsonInputFormat(new JsonFormat().withJsonType(JsonType.LINES
))))
                          .withOutputSerialization(new OutputSerialization()
                          .withCrcEnabled(true)
                          .withJsonOutputFormat(new JsonFormat()))
                     .withExpression("select * from ossobject as s where s.age > 40");
            OSSObject ossObject = client.selectObject(selectObjectRequest);
            writeToFile(ossObject.getObjectContent(), "result.simple.json");
        } catch (OSSException ex) {
            System.out.println(ex.getErrorCode().concat(",").concat(ex.getErrorMessage()));
        } catch (ClientException ex) {
            System.out.println(ex.getErrorCode().concat(",").concat(ex.getErrorMessage()));
        } finally {
            client.shutdown();
        }
    }
    /**
     * 查询复杂 json。
     */
    public static void selectComplexJson() {
        String key = "complex.json";
        // 创建 OSSClient 实例。
        OSS client = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);
        String content = "{\n" +
             "  \"contacts\":[\n" +
             "{\n" +
             "  \"firstName\": \"John\",\n" +
             "  \"lastName\": \"Smith\",\n" +
             "  \"isAlive\": true,\n" +
             "  \"age\": 27,\n" +
             "  \"address\": {\n" +
             "    \"streetAddress\": \"21 2nd Street\",\n" +
             "    \"city\": \"New York\",\n" +
             "    \"state\": \"NY\",\n" +
             "    \"postalCode\": \"10021-3100\"\n" +
             "  },\n" +
             "  \"phoneNumbers\": [\n" +
             "    {\n" +
             "      \"type\": \"home\",\n" +
             "      \"number\": \"212 555-1234\"\n" +
             "    },\n" +
             "    {\n" +
             "      \"type\": \"office\",\n" +
             "      \"number\": \"646 555-4567\"\n" +
             "    },\n" +
             "    {\n" +
             "      \"type\": \"mobile\",\n" +
             "      \"number\": \"123 456-7890\"\n" +
             "    }\n" +
             "  ],\n" +
```

```
                "  \"children\": [],\n" +
                "  \"spouse\": null\n" +
                "}\n" +
                "]}";
        try {
            client.putObject(bucketName, key, new ByteArrayInputStream(content.getBytes()))
;
            SelectObjectRequest selectObjectRequest =
                new SelectObjectRequest(bucketName, key)
                    .withInputSerialization(new InputSerialization()
                         .withCompressionType(CompressionType.NONE)
                         .withJsonInputFormat(new JsonFormat().withJsonType(JsonType.LINES))
)
                    .withOutputSerialization(new OutputSerialization()
                         .withCrcEnabled(true)
                         .withJsonOutputFormat(new JsonFormat()))
                    //返回所有 age 是 27 的记录，表达式可以根据 json 对象结构进行嵌套调用。
                    .withExpression("select * from ossobject.contacts[*] s where s.age = 27"
);
            OSSObject ossObject = client.selectObject(selectObjectRequest);
            writeToFile(ossObject.getObjectContent(), "result.complex.json");
        } catch (OSSException ex) {
            System.out.println(ex.getErrorCode().concat(",").concat(ex.getErrorMessage()));
        } catch (ClientException ex) {
            System.out.println(ex.getErrorCode().concat(",").concat(ex.getErrorMessage()));
        } finally {
            client.shutdown();
        }
    }
    /**
     * 写入文件。
     *
     * @param in
     * @param file
     */
    private static void writeToFile(InputStream in, String file) {
        try {
            BufferedOutputStream outputStream = new BufferedOutputStream(new FileOutputStre
am(file));
            byte[] buffer = new byte[1024];
            int bytesRead;
            while ((bytesRead = in.read(buffer)) != -1) {
              outputStream.write(buffer, 0, bytesRead);
            }
            outputStream.close();
        } catch (FileNotFoundException ex) {
            ex.printStackTrace();
        } catch (IOException ex) {
            ex.printStackTrace();
        }
    }
```

SelectObject 的更多详情，请参考SelectObject。

5.2. 使用Python SDK的SelectObject查询CSV和JSON文件

本文介绍如何使用Python SDK的SelectObject查询CSV和JSON文件。

> **说明**
>
> 本文示例由阿里云用户fralychen提供，仅供参考。

上传CSV或JSON格式文件

您可以根据业务需求，在OSS管理控制台将CSV或JSON格式文件上传到OSS bucket中。如何将文件上传至OSS bucket，请参见上传文件。

调用测试

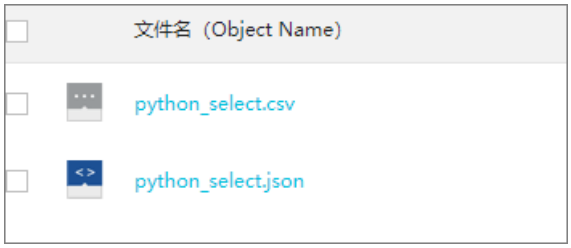
通过put_object中的key、contant参数创建并上传了一个名为python_select的文件。

> **说明** 以下JSON与CSV示例需分开执行。

```
import os
import oss2
# 首先初始化AccessKeyId、AccessKeySecret、Endpoint等信息。
# 通过环境变量获取，或者把诸如"<yourAccessKeyId>"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
access_key_id = os.getenv('OSS_TEST_ACCESS_KEY_ID', '<yourAccessKeyId>')
access_key_secret = os.getenv('OSS_TEST_ACCESS_KEY_SECRET', '<yourAccessKeySecret>')
bucket_name = os.getenv('OSS_TEST_BUCKET', '<yourBucket>')
endpoint = os.getenv('OSS_TEST_ENDPOINT', '<yourEndpoint>')
# 创建存储空间实例，所有文件相关的方法都需要通过存储空间实例来调用。
bucket = oss2.Bucket(oss2.Auth(access_key_id, access_key_secret), endpoint, bucket_name)
### CSV示例
key = 'python_select.csv'
# 文件内容。
content = 'fralychen,China,30\r\nTom,USA,20\r\n'
filename = 'python_select.csv'
# 上传一个名为python_select.csv文件。
bucket.put_object(key, content)
# 通过select_object使用sql语法查询文件。
# def select_object(self, key, sql,
#                   progress_callback=None,
#                   select_params=None,
#                   byte_range=None
#                   )
result = bucket.select_object(key, 'select * from ossobject where _3 > 29')
print('csv select result:')
print(result.read())
### JSON示例
key = 'python_select.json'
#content = "{"contacts":
#        [
#            {
#            "key1":1,
#            "key2":"love china"
#            },
#            {
#            "key1":2,
#            "key2":"fralychen"
#            }
#        ]
#    }"
content = "{\"contacts\":[{\"key1\":1,\"key2\":\"love China\"},{\"key1\":2,\"key2\":\"fraly
chen\"}]}"
# 上传一个名为python_select.json文件。
bucket.put_object(key,content)
select_json_params = {'Json_Type': 'DOCUMENT'}
result = bucket.select_object(key,'select s.key2 from ossobject.contacts[*] s where s.key1
= 1', None, select_json_params)
print('json select result:')
print(result.read())
```

输出示例

您可以在OSS控制台上查看上传后的python_select.csv及python_select.json文件。



CSV及JSON的示例输出结果如下所示。

```
csv select result:
fralychen,China,30

json select result:
{"key2":"love China"}
```

常见SQL语句

常见的SQL应用场景及对应的SQL语句如下表所示：

应用场景	SQL语句
返回前10行数据	select * from ossobject limit 10
返回第1列和第3列的整数，并且第1列大于第3列	select _1, _3 from ossobject where cast(_1 as int) > cast(_3 as int)
返回第1列以'陈'开头的记录的个数	select count(*) from ossobject where _1 like '陈%' **说明** 此处like之后的中文需要用UTF-8编码。
返回所有第2列时间大于2018-08-09 11:30:25且第3列大于200的记录	select * from ossobject where _2 > cast('2018-08-09 11:30:25' as timestamp) and _3 > 200
返回第2列浮点数的平均值、总和、最大值、和最小值	select AVG(cast(_2 as double)), SUM(cast(_2 as double)), MAX(cast(_2 as double)), MIN(cast(_2 as double))
返回第1列和第3列连接的字符串中以'Tom'为开头以’Anderson‘结尾的所有记录	select * from ossobject where (_1 || _3) like 'Tom%Anderson'
返回第1列能被3整除的所有记录	select * from ossobject where (_1 % 3) = 0
返回第1列大小在1995到2012之间的所有记录	select * from ossobject where _1 between 1995 and 2012
返回第5列值为N、M、G、和L的所有记录	select * from ossobject where _5 in ('N', 'M', 'G', 'L')
返回第2列乘以第3列比第5列大100以上的所有记录	select * from ossobject where _2 * _3 > _5 + 100

更多参考

有关Python SDK API的更多信息，请参见GitHub。

有关SelectObject的更多信息，请参见查询文件。

有关使用Python SDK的SelectObject查询CSV和JSON文件的更多信息，请参见查询文件。

6.数据备份和容灾

6.1. 备份存储空间

针对存放在对象存储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。本文介绍备份OSS数据的几种主要方式。

通过定时备份功能进行备份

混合云备份服务可创建备份计划，并按计划将您OSS内的数据备份到混合云备份服务中，当您的数据因误修改、误删除等原因丢失时，可及时恢复。您也可以使用混合云备份服务长期、低成本地保存OSS的历史数据。配置方法请参见定时备份。

通过跨区域复制功能进行备份

跨区域复制是跨不同OSS数据中心（地域）的存储空间（Bucket）自动、异步（近实时）复制文件，它会将文件的创建、更新和删除等操作从源存储空间复制到不同区域的目标存储空间。配置方法请参见设置跨区域复制。

通过在线迁移服务进行备份

阿里云在线迁移服务是阿里云提供的存储产品数据通道。使用在线迁移服务，您可以将第三方数据轻松迁移至OSS，也可以在OSS之间进行灵活的数据迁移。配置方法请参见阿里云OSS之间迁移教程。

通过ossimport工具进行备份

ossimport是一款将数据迁移至OSS的工具。您可以将ossimport部署在本地服务器或云上ECS实例内，轻松将您本地或其它云存储的数据迁移到OSS。配置方法请参见说明及配置。

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

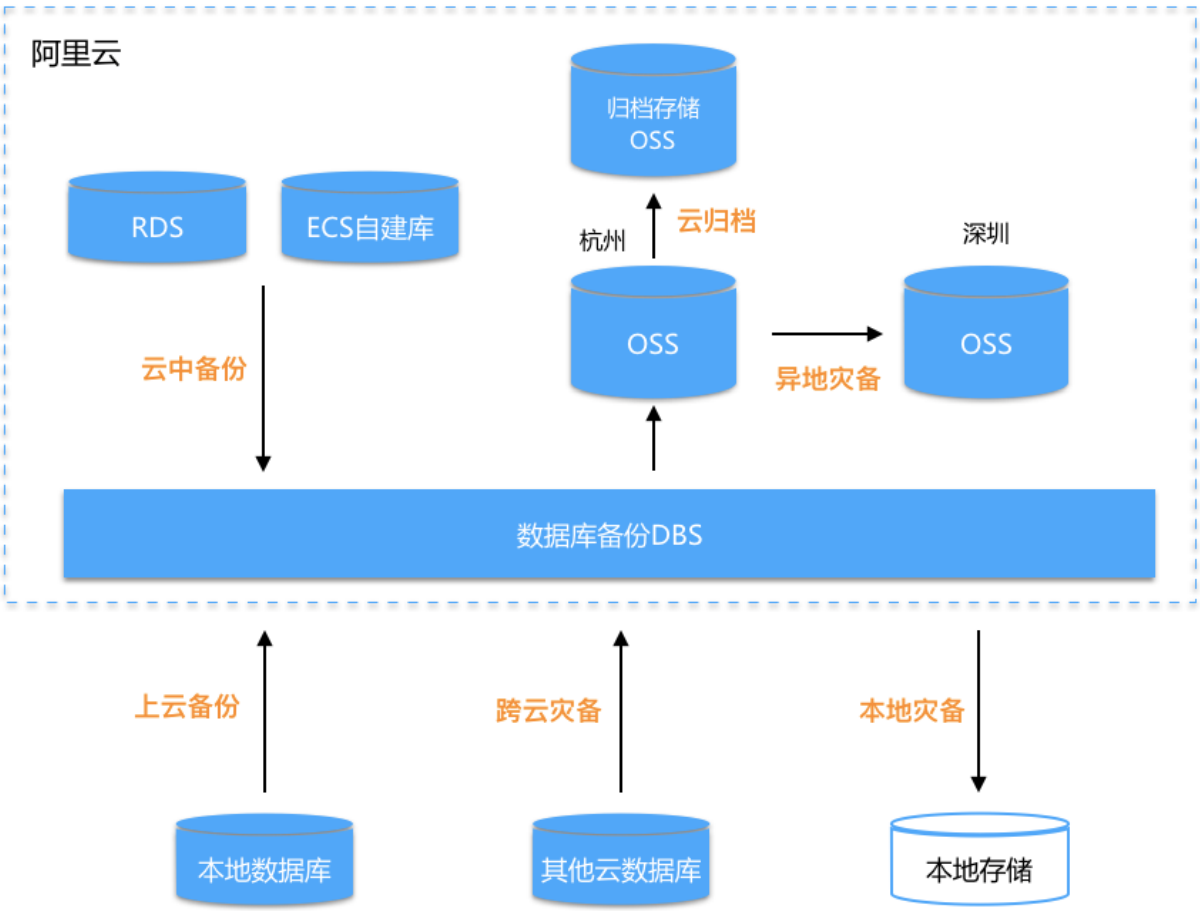
- 对象存储OSS

  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

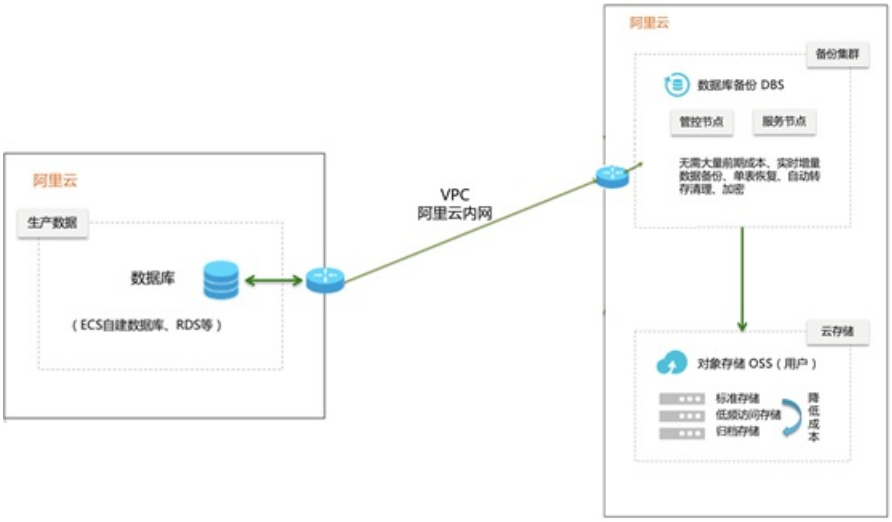
  数据库备份DBS是为数据库提供连续数据保护、低成本的备份服务。它可以为多种环境的数据提供强有力的保护，包括企业数据中心、其他云厂商及公共云。数据库备份提供数据备份和操作恢复的整体方案，具备实时增量备份、精确到秒级的数据恢复能力。

应用场景

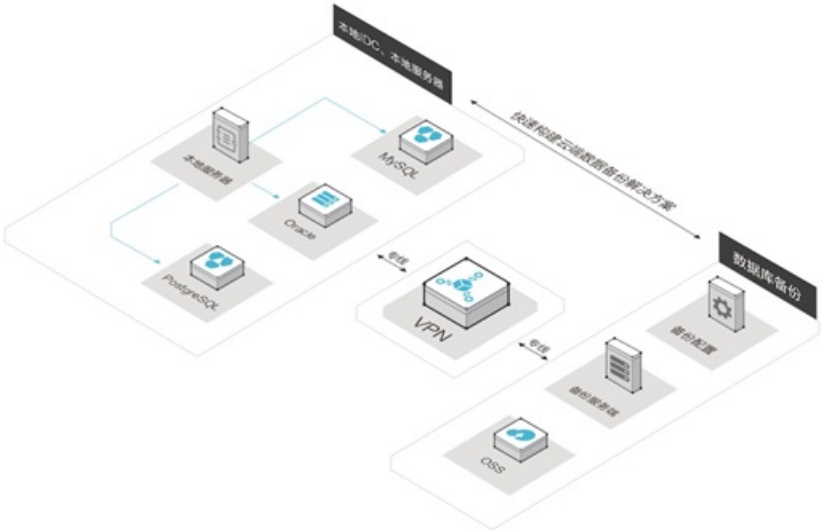


数据库备份到OSS的方案应用场景如下：

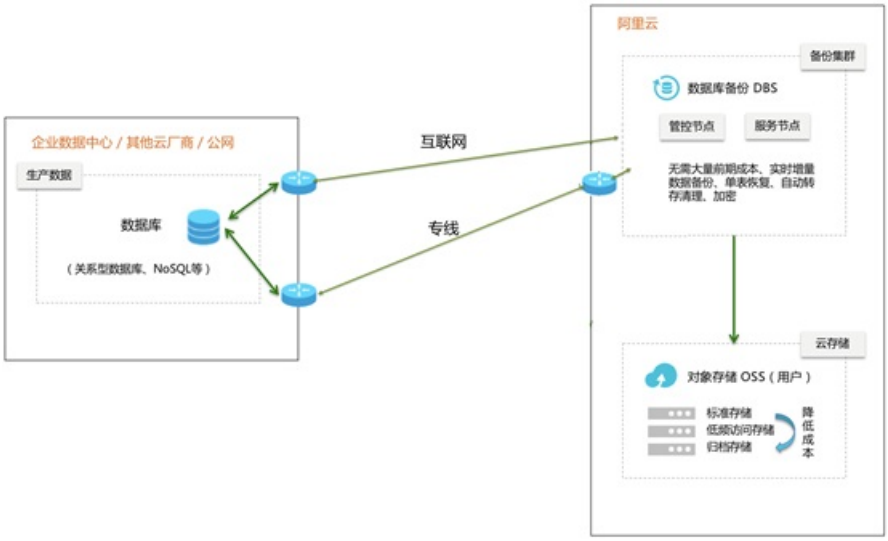
- 阿里云RDS或阿里云ECS自建数据库在OSS上备份或长期归档



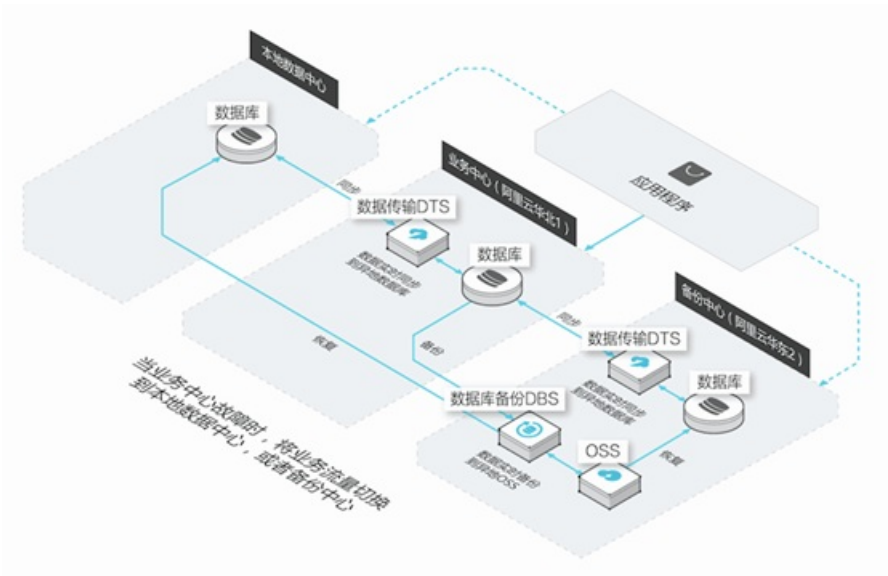
- 自建IDC数据备份上云



- 其他云数据库在阿里云OSS上做多云备份

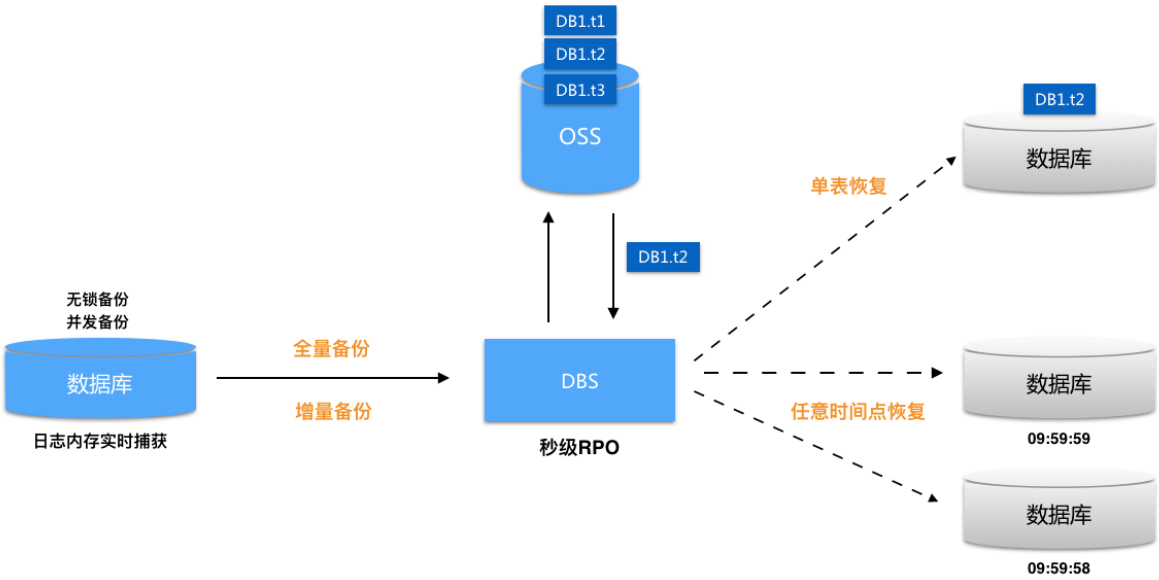


- 数据库做异地备份灾备



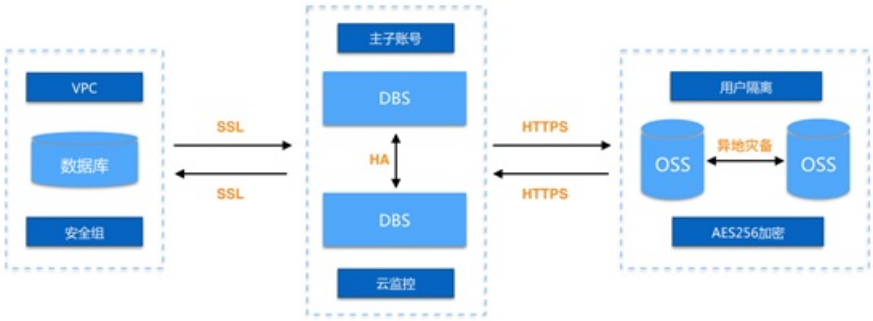
方案优势

- 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。详情请参见创建备份计划。
2. 配置备份计划。详情请参见配置备份计划。
3. 查看备份计划。详情请参见查看备份计划。
4. 恢复数据库。详情请参见恢复数据库。

7.成本管理

7.1. 使用生命周期管理文件版本

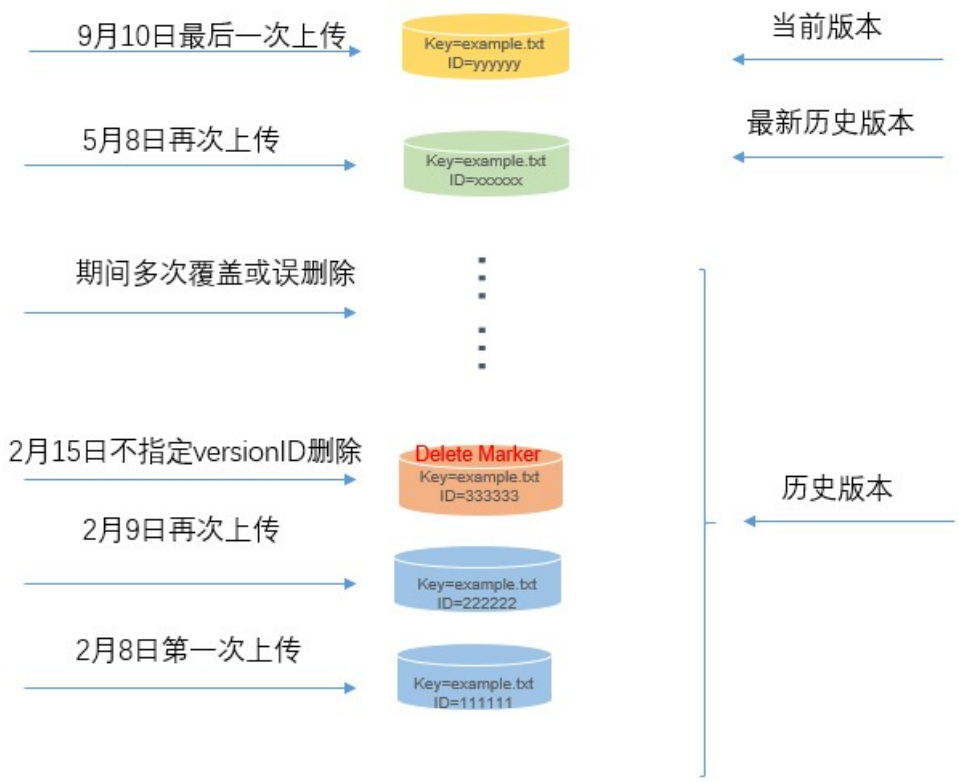
存储空间（Bucket）开启版本控制后，针对数据的覆盖和删除操作将会以历史版本的形式保存下来。当Bucket累积了大量的历史版本或者过期删除标记时，您可以结合生命规则删除不必要的历史版本以及过期删除标记，从而减少存储成本并有效提升列举Object的性能。

前提条件

目标Bucket已开启版本控制。详情请参见开启版本控制。

场景说明

当目标存储空间examplebucket开启版本控制后，王先生在某一年2月8日上传了名为example.txt的文件，此后在同一年份的不同时间内对example.txt文件进行了多次覆盖或不指定versionID的删除操作，OSS对该文件的每一次覆盖和删除操作均生成全局唯一的随机字符串versionID（图示中的versionID均以简易版本号标识，不代表实际versionID），并将文件以历史版本的形式保存在目标Bucket中。



文件经多次覆盖和不指定versionID的删除操作后，结合业务场景的变化，王先生需实现如下需求：

- 仅保留5月8日以及9月10日上传的文件版本。
- 将5月8日生成的最新历史版本文件恢复为当前版本。

注意事项

使用生命周期过期策略管理不同版本Object时，有如下注意事项：

- 当前版本Object过期策略

- 在开启版本控制的情况下，如果生命周期规则中的过期策略作用于当前版本Object，OSS会添加删除标记将当前版本Object作为历史版本Object保留，而不是删除当前版本Object，且删除标记将成为Object的当前版本。
- 在暂停版本控制的情况下，如果生命周期规则中的过期策略作用于当前版本Object，OSS会添加删除标记作为当前版本，且versionID为null。由于OSS保证同一个Object只会有一个versionID为null的版本，因此原versionID为null的版本将被覆盖。

- 历史版本Object过期策略

  在开启或暂停版本控制的情况下，如果生命周期规则中的过期策略作用于历史版本Object，OSS会永久删除历史版本Object，且无法恢复永久删除的历史版本Object。

有关生命周期规则的更多信息，请参见基于最后一次修改时间的生命周期规则介绍。

操作步骤

1. 保留指定版本文件

   假设当前时间为9月10日，则通过配置以下生命周期规则可实现仅保留5月8日以及9月10日上传的文件版本。

   i. 登录OSS管理控制台。

   ii. 单击**Bucket列表**，然后单击examplebucket。

   iii. 单击**基础设置** > **生命周期**，在**生命周期**区域单击**设置**。

   iv. 单击**创建规则**，按如下说明配置生命周期规则。

区域	配置项	配置方法
基础设置	**状态**	选择**启动**。
	策略	选择**配置到整个Bucket**。

区域	配置项	配置方法
当前版本	**文件过期策略**	选择**清理对象删除标记**。
历史版本	**文件过期策略**	选择**过期天数**。
	删除文件	设置为90天，Object会在其被转换为历史版本的90天后过期，并在过期的第二天被删除。
清理碎片	**碎片过期策略**	选择**过期天数**。
	删除碎片	设置为90天，因分片上传产生的碎片90天后过期，并在过期的第二天被删除。

v. 单击**确定**。

2. 恢复指定版本文件

将5月8日生成的最新历史版本文件恢复为当前版本的操作步骤如下：

i. 在examplebucket管理页面，单击**文件管理**。

ii. 找到更新时间为5月8日对应版本的example.txt文件。

iii. 单击目标历史版本右侧的**恢复**。

8.数据迁移

8.1. OSS之间数据迁移

8.1.1. 概述

您可以将同一个阿里云账号下的OSS某个存储空间（Bucket）的数据迁移至另一个Bucket，还可以跨不同阿里云账号迁移OSS Bucket之间的数据。

OSS Bucket之间的数据迁移包含以下场景：

- 同账号下的OSS数据迁移，即同一个阿里云账号下相同或者不同地域Bucket之间的数据迁移。具体步骤，请参见使用数据复制功能迁移同账号下的OSS数据。
- 跨账号下的OSS数据迁移，即不同阿里云账号下相同或不同地域Bucket之间的数据迁移。具体操作，请参见使用在线迁移服务跨账号迁移OSS数据。

8.1.2. 使用数据复制功能迁移同账号下的OSS数据

在同一个阿里云账号下，您可以通过OSS的跨区域复制功能将地域A的某个存储空间（Bucket）数据迁移至地域B下的另一个Bucket。如果您需要将地域A某个Bucket的数据迁移至相同地域的另一个Bucket，请使用OSS的同区域复制功能。

注意事项

- 数据迁移任务会在跨区域复制或者同区域复制规则配置完成的3~5分钟后启动。
- 由于Bucket间的数据复制采用异步（近实时）复制，数据迁移到目标Bucket需要的时间取决于数据的大小，通常几分钟到几小时不等。
- 迁移历史数据时，从源Bucket复制的Object可能会覆盖目标Bucket中同名的Object。为避免同名文件被覆写，建议您对源Bucket和目标Bucket开启版本控制。开启版本控制的具体步骤，请参见版本控制相关操作。
- 如果您希望在数据迁移进度达到100%时，不再继续迁移源Bucket中的增量数据，您可以选择关闭数据同步。此时，已迁移的数据将被保留在目标Bucket中，源Bucket中的增量数据将不再迁移到目标Bucket。

有关跨区域复制的更多信息，请参见跨区域复制。有关同区域复制的更多信息，请参见同区域复制。

不同地域Bucket之间的数据迁移

例如，您需要将华北2（北京）地域的源Bucket A的所有数据迁移到华东1（杭州）的目标Bucket B，具体步骤如下：

1. 登录OSS管理控制台。
2. 在左侧导航栏，单击**Bucket列表**，然后单击Bucket A。
3. 在左侧导航栏，选择**冗余与容错 > 跨区域复制**。
4. 在**跨区域复制**区域，单击**设置**。
5. 单击**跨区域复制**，然后在**跨区域复制**面板配置以下参数。

参数	说明及示例值
源Bucket地域	显示Bucket A所在地域华北2（北京）。

参数	说明及示例值
源Bucket	显示Bucket A名称。
目标地域	选择华东1（杭州）。
目标Bucket	选择Bucket B。
数据同步对象	选择**全部文件进行同步**。
数据同步策略	选择**增/改 同步**。
同步历史数据	选择**同步**。

6. 单击**确定**。
   此时，**跨区域复制**页面将显示数据迁移进度。

相同地域Bucket之间的数据迁移

例如，您需要将华北2（北京）地域的源Bucket C的所有数据迁移到相同地域的目标Bucket D，具体步骤如下：

1. 登录OSS管理控制台。
2. 在左侧导航栏，单击**Bucket列表**，然后单击Bucket C。
3. 在左侧导航栏，选择**冗余与容错 > 同区域复制**。
4. 在**同区域复制**区域，单击**设置**。
5. 单击**同区域复制**。
6. 在**同区域复制**面板，按如下说明配置各项参数。

参数	说明及示例值
源Bucket地域	显示Bucket C所在地域华北2（北京）。
源Bucket	显示Bucket C的名称。
目标Bucket	选择Bucket D。
数据同步对象	选择**全部文件进行同步**。
数据同步策略	选择**增/改 同步**。
同步历史数据	选择**同步**。

7. 单击**确定**。
   此时，**同区域复制**页面将显示数据迁移进度。

更多参考

如果您需要跨账号迁移OSS Bucket之间的数据，请参见使用在线迁移服务跨账号迁移OSS数据。

8.1.3. 使用在线迁移服务跨账号迁移OSS数据

您可以使用阿里云在线迁移服务，将阿里云账号A下的OSS源存储空间Bucket A的数据迁移至另一个阿里云账号B的OSS目标存储空间Bucket B，Bucket A与Bucket B可以位于相同或不同地域。

前提条件

- 已创建RAM用户。

  为阿里云账号A创建RAM用户A，为阿里云账号B创建RAM用户B。具体步骤，请参见创建RAM用户。

- 已创建AccessKey。

  分别为RAM用户A以及RAM用户B创建访问密钥AccessKey，并记录AccessKey信息。具体步骤，请参见为RAM用户创建访问密钥

- 已为RAM用户授权。

  分别为RAM用户A以及RAM用户B授予 `AliyunOSSFullAccess` 以及 `AliyunMGWFullAccess` 的权限。具体步骤，请参见为RAM用户授权。

跨账号跨地域迁移OSS数据

例如，您需要以外网Endpoint的方式，将阿里云账号A下华东2（上海）地域下的OSS Bucket A的数据迁移至阿里云账号B华东1（杭州）地域的Bucket B。具体步骤如下：

> **注意** 跨账号跨地域迁移OSS数据时，仅支持使用外网Endpoint。

1. 创建源地址。
   i. 登录阿里云数据在线迁移控制台。
   ii. 在左侧导航栏，选择**在线迁移服务 > 数据地址**，然后单击右上角的**创建数据地址**。
   iii. 在**创建数据地址**面板，按如下说明配置如各项参数。

参数	说明和示例值
数据类型	选择**OSS**。
数据所在区域	选择**华东2（上海）**。
数据名称	输入*migrationtask1*。
OSS Endpoint	选择**https://oss-cn-shanghai.aliyuncs.com**。有关OSS Endpoint的更多信息，请参见访问域名和数据中心。
AccessKey Id	输入RAM用户A的AccessKey ID。
AccessKey Secret	输入RAM用户A的AccessKey Secret。
OSS Bucket	选择Bucket A。

2. 创建目的地址。
   i. 在左侧导航栏，选择**在线迁移服务 > 数据地址**，然后单击右上角的**创建数据地址**。

ii. 在**创建数据地址**面板，按如下说明配置如各项参数。

参数	说明和示例值
数据类型	选择**OSS**。
数据所在区域	选择**华东1（杭州）**。
数据名称	输入*migrationtask2*。
OSS Endpoint	选择**https://oss-cn-hangzhou.aliyuncs.com**。
AccessKey Id	输入RAM用户B的AccessKey ID。
AccessKey Secret	输入RAM用户B的AccessKey Secret。
OSS Bucket	选择Bucket B。

3. 创建迁移任务。
   i. 选择**在线迁移服务 > 迁移任务**，然后单击**创建迁移任务**。
   ii. 在**创建迁移任务**页面，阅读迁移服务条款协议，选中**我理解如上条款，并开通数据迁移服务**，然后单击**下一步**。
   iii. 在弹出的**费用提示**对话框，单击**确认，继续创建**。
   iv. 在**配置任务**面板，设置以下参数，其他参数保留默认值，然后单击**下一步**。

参数	说明和示例值
任务名称	输入*Task2*。
源地址	选择已创建的源地址**[oss]migrationtask1**。
目的地址	选择已创建的目的地址**[oss]migrationtask2**。
迁移方式	选择**全量迁移**。

   v. 在**性能调优**页签的**数据预估**区域，填写**待迁移存储量**和**待迁移文件个数**。
   vi. 在**性能调优**页签的**流量控制**区域，设置限流时间段和最大流量，然后单击**添加**。
   vii. 单击**创建**。

跨账号同地域迁移OSS数据

例如，您可以通过内网Endpoint的方式，将阿里云账号A下华东2（上海）地域下的OSS Bucket A的数据迁移至阿里云账号B相同地域的Bucket B。具体步骤如下：

> **说明** 跨账号同地域迁移OSS数据时，建议使用内网Endpoint。如果使用外网Endpoint，可能会产生大量的外网流出流量费用。

1. 创建源地址。
   i. 登录阿里云数据在线迁移控制台。
   ii. 在左侧导航栏，选择**在线迁移服务 > 数据地址**，然后单击右上角的**创建数据地址**。

iii. 在**创建数据地址**面板，按如下说明配置如各项参数。

参数	说明和示例值
数据类型	选择**OSS**。
数据所在区域	选择**华东2（上海）**。
数据名称	输入*migrationtask1*。
OSS Endpoint	选择**https://oss-cn-shanghai-internal.aliyuncs.com**。有关OSS Endpoint的更多信息，请参见访问域名和数据中心。
AccessKey Id	输入RAM用户A的AccessKey ID。
AccessKey Secret	输入RAM用户A的AccessKey Secret。
OSS Bucket	选择Bucket A。

2. 创建目的地址。

i. 在左侧导航栏，选择**在线迁移服务 > 数据地址**，然后单击右上角的**创建数据地址**。

ii. 在**创建数据地址**面板，按如下说明配置如各项参数。

参数	说明和示例值
数据类型	选择**OSS**。
数据所在区域	选择**华东2（上海）**。
数据名称	输入*migrationtask2*。
OSS Endpoint	选择**https://oss-cn-shanghai-internal.aliyuncs.com**。
AccessKey Id	输入RAM用户B的AccessKey ID。
AccessKey Secret	输入RAM用户B的AccessKey Secret。
OSS Bucket	选择Bucket B。

3. 创建迁移任务。

i. 选择**在线迁移服务 > 迁移任务**，然后单击**创建迁移任务**。

ii. 在**创建迁移任务**页面，阅读迁移服务条款协议，选中**我理解如上条款，并开通数据迁移服务**，然后单击**下一步**。

iii. 在弹出的**费用提示**对话框，单击**确认，继续创建**。

iv. 在**配置任务**面板，设置以下参数，其他参数保留默认值，然后单击**下一步**。

参数	说明和示例值
任务名称	输入*Task2*。
源地址	选择已创建的源地址**[oss]migrationtask1**。
目的地址	选择已创建的目的地址**[oss]migrationtask2**。
迁移方式	选择**全量迁移**。

v. 在**性能调优**页签的**数据预估**区域，填写**待迁移存储量**和**待迁移文件个数**。

vi. 在**性能调优**页签的**流量控制**区域，设置限流时间段和最大流量，然后单击**添加**。

vii. 单击**创建**。

更多参考

- 迁移指定数据

  以上场景假设了迁移整个Bucket的所有数据，如果您只需要迁移部分数据，例如包含指定前缀Prefix的文件，您可以在创建数据地址时指定OSS Prefix。

- 使用增量迁移

  考虑到一次全量迁移完成后源数据可能有变化，您可以指定增量迁移间隔和增量迁移次数执行增量迁移任务，将源地址从前次迁移任务开始后到下次迁移开始前新增或修改的增量数据迁移至目的地址。

- 选择同名文件的覆盖形式

  如果迁移过程中源地址和目的地址出现同名文件时，您可以选择不进行任何判断直接覆盖同名文件或者直接跳过同名文件，也可以结合文件元数据信息，例如最后修改时间LastModified、文件大小Size和文件类型Content-Type等是否相同进一步判断覆盖或者跳过同名文件。

如果您希望在数据迁移场景中结合以上条件满足更灵活的业务需求，请参见迁移实施。

跨账号数据迁移的更多场景

- 如果您希望在同一个阿里云账号下迁移OSS数据，请参见使用数据复制功能迁移同账号下的OSS数据。

同账号数据迁移

8.2. 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，如亚马逊AWS、谷歌云等数据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。具体各个数据源的迁移操作，请参见在线迁移服务使用教程。

8.3. 从AWS S3上的应用无缝切换至OSS

OSS提供了S3 API的兼容性，可以将您的数据从AWS S3无缝迁移至阿里云OSS。

注意事项

- 使用限制

  由于OSS兼容S3协议，因此您可以通过S3 SDK进行创建Bucket、上传Object等相关操作。执行相关操作过程中其带宽、QPS等限制遵循OSS性能指标，详情请参见使用限制。

- 客户端配置

  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：

  i. 获取阿里云账号或RAM用户的AccessKey ID和AccessKey Secret，并在您使用的客户端和SDK中配置您申请的AccessKey ID与AccessKey Secret。

  ii. 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。

S3兼容性

关于OSS兼容的S3 API以及OSS与S3的差异，请参见AWS S3兼容性。

8.4. 使用ossimport迁移数据

ossimport支持将任意地域的本地存储数据、第三方存储数据、对象存储OSS数据迁移至任意地域的OSS中。本文介绍如何使用ossimport将数据从第三方存储迁移到OSS。

背景信息

某用户的数据存储于腾讯云COS广州（华南）区域，数据大小约500TB。现希望将这些数据，通过ossimport工具，于一周内迁移至OSS华东1（杭州）区域。在迁移的同时，需保证自身业务的正常进行。

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。

此需求需要使用ossimport分布式配置进行数据迁移。

> **说明** 您也可以使用在线迁移服务进行数据的迁移，迁移过程更加简单，详情请参见在线迁移服务。

准备工作

- 开通OSS，并创建华东1（杭州）地域的存储空间（Bucket）
  - 开通OSS步骤请参见开通OSS服务。
  - 创建Bucket步骤请参见创建存储空间。
- 创建RAM用户，并授予访问OSS的权限

  在RAM控制台创建RAM用户，并授权该RAM用户访问OSS的权限，然后保存AccessKey ID和AccessKey Secret。详情请参见创建RAM用户并授予相关权限。

- （可选）购买ECS

  购买与OSS相同地域的ECS实例。有关ECS实例规格的更多信息，请参见通用型。如果迁移后ECS实例需释放，建议按需购买ECS。

> **说明** 如果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；如果所需计算机数量较多时，建议在ECS实例上部署。本示例以ECS实例进行迁移任务。

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）。

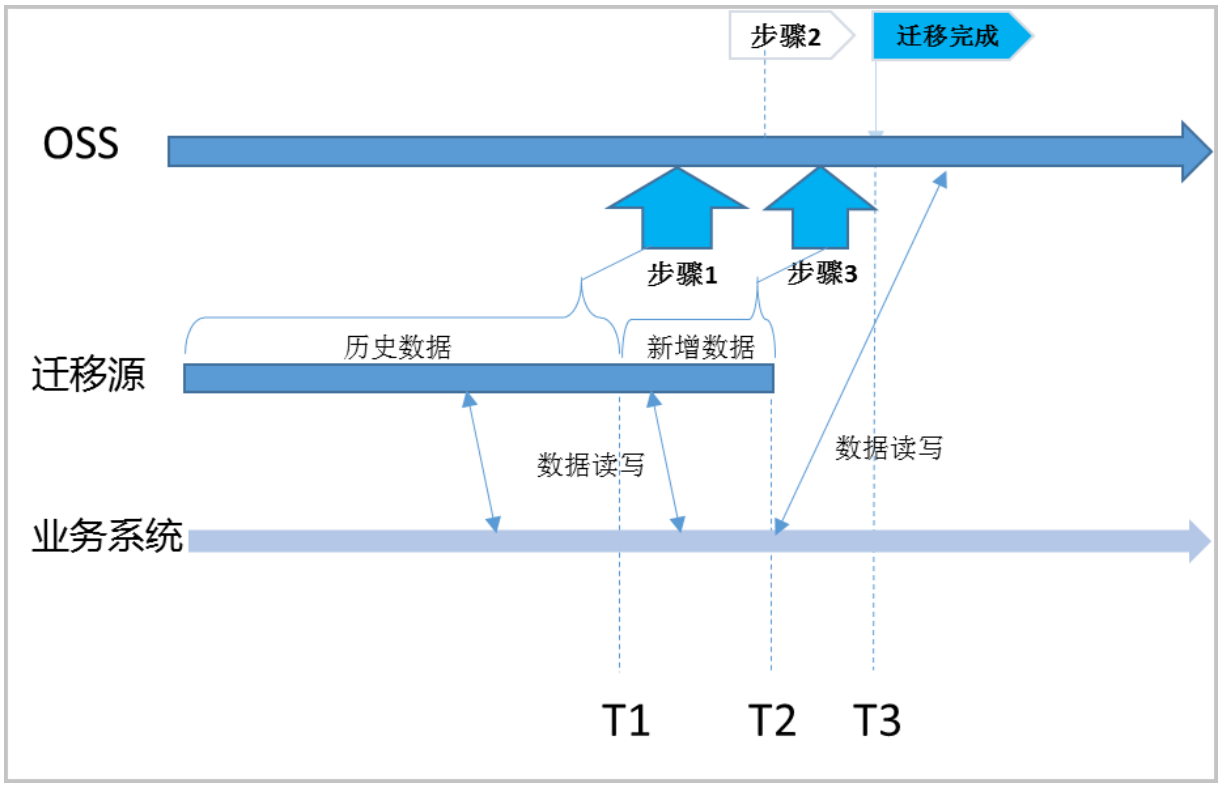
- 配置ossimport

  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，请参见说明及配置。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等，请参见分布式部署。

迁移方案

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广州（华南）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。



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。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。

迁移实施

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01月01日UTC零点以来的秒数，通过命令date +%s获取。

2. 配置镜像回源。

   数据迁移过程中，源站还在不断产生新的数据。为了不中断业务，做到业务无缝切换，还需要配置镜像回源功能。当用户请求的文件在 OSS 中没有找到时，OSS会自动到源站抓取对应文件保存到 OSS，并将内容直接返回给用户。配置步骤请参见OSS镜像回源。

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。
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。

参考文档

有关ossimport的相关说明，请参见以下文档：

分布式部署

说明及配置

常见问题

8.5. 从HDFS迁移数据到OSS

本文介绍如何使用阿里云Jindo DistCp从HDFS迁移数据到OSS。

背景信息

在传统大数据领域，HDFS经常作为大规模数据的底层存储。在进行数据迁移、数据拷贝的场景中，最常用的是Hadoop自带的DistCp工具。但是该工具不能很好利用对象存储OSS的特性，导致效率低下并且不能保证数据一致性。此外，该工具提供的功能选项较单一，无法很好地满足用户的需求。

阿里云Jindo DistCp（分布式文件拷贝工具）用于大规模集群内部或集群之间拷贝文件。Jindo DistCp使用MapReduce实现文件分发，错误处理和恢复，把文件和目录的列表作为MapReduce任务的输入，每个任务会完成源列表中部分文件的拷贝。全量支持HDFS之间、HDFS与OSS之间、以及OSS之间的数据拷贝场景，提供多种个性化拷贝参数和多种拷贝策略。

相对于Hadoop DistCp，使用阿里云Jindo DistCp从HDFS迁移数据到OSS具有以下优势：

- 效率高，在测试场景中最高可达到1.59倍的加速。
- 基本功能丰富，提供多种拷贝方式和场景优化策略。
- 深度结合OSS，对文件提供归档、压缩等操作。
- 实现No-Rename拷贝，保证数据一致性。
- 场景全面，可完全替代Hadoop DistCp，目前支持Hadoop2.7+和Hadoop3.x。

前提条件

- 如果您使用的是自建ECS集群，需要具备Hadoop2.7+或Hadoop3.x环境以及进行MapReduce作业的能

力。

- 如果您使用的是阿里云E-MapReduce：
  - 对于EMR3.28.0/bigboot2.7.0及以上的版本，可以通过Shell命令的方式使用Jindo DistCp。更多信息，请参见Jindo DistCp使用说明。
  - 对于EMR3.28.0/bigboot2.7.0以下的版本，可能会存在一定的兼容性问题，您可以通过提交工单申请处理。

步骤一：下载JAR包

- Hadoop 2.7+的JAR包
- Hadoop 3.x的JAR包

步骤二：配置OSS的AccessKey

您可以通过以下任意方式配置AccessKey：

- 在示例命令中配置AccessKey

  例如，在将HDFS中的目录拷贝到OSS指定路径的示例命令中结合*--ossKey*、*--ossSecret*、*--ossEndPoint*选项配置AccessKey。

```
hadoop jar jindo-distcp-3.7.3.jar --src /data/incoming/examplefile --dest oss://examplebu
cket/example_file --ossKey LTAI5t7h6SgiLSganP2m**** --ossSecret KZo149BD9GLPNiDIEmdQ7dyNK
G**** --ossEndPoint oss-cn-hangzhou.aliyuncs.com
```

- 通过配置文件预先配置AccessKey

  将OSS的--ossKey、--ossSecret、--ossEndPoint预先配置在Hadoop的*core-site.xml*文件里。示例命令如下：

```
<configuration>
    <property>
        <name>fs.oss.accessKeyId</name>
        <value>xxx</value>
    </property>
    <property>
        <name>fs.oss.accessKeySecret</name>
        <value>xxx</value>
    </property>
    <property>
        <name>fs.oss.endpoint</name>
        <!-- 阿里云ECS环境下推荐使用内网OSS Endpoint，即oss-cn-xxx-internal.aliyuncs.com -->
        <value>oss-cn-xxx.aliyuncs.com</value>
    </property>
</configuration>
```

- 配置免密功能

  配置免密功能，避免明文保存AccessKey，提高安全性。具体操作，请参见使用JindoFS SDK免密功能。

步骤三：迁移或拷贝数据

以下以Jindo DistCp 3.7.3版本为例，您可以根据实际环境替换对应的版本号。

- 全量迁移或拷贝数据

将HDFS指定目录*/data/incoming*下的数据全量迁移或拷贝到OSS目标路径*oss://examplebucket/incoming/*，示例命令如下：

```
hadoop jar jindo-distcp-3.7.3.jar --src /data/incoming --dest oss://examplebucket/incoming --ossKey LTAI5t7h6SgiLSganP2m**** --ossSecret KZo149BD9GLPNiDIEmdQ7dyNKG**** --ossEndPoint oss-cn-hangzhou.aliyuncs.com --parallelism 10
```

示例中涉及的各参数或选项说明如下：

参数及选项	说明	示例
--src	待迁移或拷贝的HDFS数据所在的路径。	*/data/incoming*
--dest	OSS中存放迁移或拷贝数据的目标路径。	*oss://examplebucket/incoming*
--ossKey	访问OSS的AccessKey ID。关于获取AccessKey ID的具体操作，请参见获取AccessKey。	LTAI5t7h6SgiLSganP2m****
--ossSecret	访问OSS的AccessKey Secret。关于获取AccessKey Secret的具体操作，请参见获取AccessKey。	KZo149BD9GLPNiDIEmdQ7dyNKG****
--ossEndPoint	Bucket所在地域（Region）对应的访问域名（Endpoint）。关于OSS支持的地域和对应的访问域名列表信息，请参见访问域名和数据中心。 注意 ECS环境下推荐使用内网ossEndPoint，即*oss-cn-xxx-internal.aliyuncs.com*。	oss-cn-hangzhou.aliyuncs.com
--parallelism	根据集群资源调整任务并发数。	10

- 增量迁移或拷贝数据

  如果您仅希望拷贝在上一次全量迁移或拷贝后源路径下新增的数据，此时您可以结合*--update*选项完成数据的增量迁移或拷贝。

  将HDFS指定目录*/data/incoming*下的数据增量迁移或拷贝到OSS目标路径*oss://examplebucket/incoming*，示例命令如下：

  ```
  hadoop jar jindo-distcp-3.7.3.jar --src /data/incoming --dest oss://examplebucket/incoming --ossKey LTAI5t7h6SgiLSganP2m**** --ossSecret KZo149BD9GLPNiDIEmdQ7dyNKG**** --ossEndPoint oss-cn-hangzhou.aliyuncs.com --update --parallelism 10
  ```

  使用*--update*选项时，默认开启校验和Checksum。开启后，DistCp将对源路径和目标路径的文件名称、文件大小以及文件的Checksum进行比较。如果源路径或目标路径下的文件名称、文件大小或者文件的Checksum不一致时，将自动触发增量迁移或拷贝任务。

  如果您不需要对源路径和目标路径的文件的Checksum进行比较，请增加*--disableChecksum*选项关闭Checksum校验，示例命令如下：

```
hadoop jar jindo-distcp-3.7.3.jar --src /data/incoming --dest oss://examplebucket/incomin
g --ossKey LTAI5t7h6SgiLSganP2m**** --ossSecret KZo149BD9GLPNiDIEmdQ7dyNKG**** --ossEndPo
int oss-cn-hangzhou.aliyuncs.com --update --disableChecksum --parallelism 10
```

附录一：Jindo DistCp支持的参数及选项

Jindo DistCp提供一系列的参数及选项。您可以通过以下命令获取各参数及选项的具体用法。

```
hadoop jar jindo-distcp-3.7.3.jar --help
```

各参数及选项的含义及其示例如下表所示。

参数及选项	说明	示例
--src	指定拷贝的源路径。	--src oss://exampleBucket/sourceDir
--dest	指定拷贝的目标路径。	--dest oss://exampleBucket/destDir
--parallelism	指定拷贝的任务并发数，可根据集群资源调节。	--parallelism 10
--policy	指定拷贝到OSS后的文件类型。取值： • *ia*：低频访问 • *archive*：归档存储 • *coldArchive*：冷归档存储	--policy archive
--srcPattern	指定通过正则表达式来选择或者过滤需要拷贝的文件，正则表达式必须为全路径正则匹配。	--srcPattern .*\.log
--deleteOnSuccess	指定在拷贝任务完成后删除源路径下的文件。	--deleteOnSuccess
--outputCodec	指定文件的压缩方式。当前版本支持编解码器gzip、gz、lzo、lzop和snappy，以及关键字none和keep。关键字含义如下： • none：保存为未压缩的文件。如果文件已压缩，则Jindo DistCp会将其解压缩。 • keep（默认值）：不更改文件压缩形态。 **说明** 如果您需要在开源Hadoop集群环境中使用lzo的压缩方式，请确保已安装gplcompression的native库和hadoop-lzo包。如果缺少相关环境，建议使用其他压缩方式进行压缩。	--outputCodec gzip
--srcPrefixesFile	指定需要拷贝的文件列表，列表里文件以src路径作为前缀。	--srcPrefixesFile file:///opt/folders.txt

参数及选项	说明	示例
--outputManifest	指定在dest目录下生成一个gzip压缩的文件，记录已完成拷贝的文件信息。	--outputManifest=manifest-2020-04-17.gz
--requirePreviousManifest	指定本次拷贝操作是否需要读取之前已拷贝的文件信息。取值如下： • *false*：不读取已拷贝的文件信息，直接拷贝全量数据。 • *true*：读取已拷贝的文件信息，仅拷贝增量数据。	--requirePreviousManifest=false
--previousManifest	指定本次拷贝需要读取已拷贝文件信息所在的路径，完成增量更新。	--previousManifest=oss://exampleBucket/manifest-2020-04-16.gz
--copyFromManifest	从已完成的Manifest文件中进行拷贝，通常与*--previousManifest*选项配合使用。	--previousManifest oss://exampleBucket/manifest-2020-04-16.gz --copyFromManifest
--groupBy	通过正则表达式将符合规则的文件进行聚合。	--groupBy='.*/([a-z]+).*.txt'
--targetSize	指定聚合后的文件大小阈值，单位为MB。	--targetSize=10
--enableBalancePlan	适用于拷贝任务中数据量差异不大的场景，例如均为大于10 GB或者均为小于10 KB的文件。	--enableBalancePlan
--enableDynamicPlan	适用于拷贝任务中数据量差异较大的场景，例如10 GB大文件和10 KB小文件混合的场景。	--enableDynamicPlan
--enableTransaction	保证Job级别的一致性，默认是Task级别。	--enableTransaction
--diff	查看本次拷贝是否完成全部文件拷贝，并对未完成拷贝的文件生成文件列表。	--diff
--ossKey	访问OSS的AccessKey ID。	--ossKey LTAI5t7h6SgiLSganP2m****
--ossSecret	访问OSS的AccessKey Secret。	--ossSecret KZo149BD9GLPNiDIEmdQ7dyNKG****
--ossEndPoint	Bucket所在地域对应的Endpoint。	--ossEndPoint oss-cn-hangzhou.aliyuncs.com
--cleanUpPending	清理OSS残留文件，清理过程需要消耗一定的时间。	--cleanUpPending
--queue	Yarn队列名称。	--queue examplequeue1
--bandwidth	指定本次DistCp任务所用的单机带宽，单位为 MB。	--bandwidth 6

参数及选项	说明	示例
--disableChecksum	关闭Checksum校验。	--disableChecksum
--enableCMS	开启云监控告警功能。	--enableCMS
--update	使用增量同步功能，即仅同步上一次全量迁移或拷贝后源路径下新增的数据到目标路径。	--update
--filters	通过filters参数指定一个文件路径。在这个文件中，每一行配置一个正则表达式，对应DistCp任务中不需要拷贝或比对的文件。	--filters /path/to/filterfile.txt
--tmp	指定在使用DistCp工具的过程中，用于存放临时文件的目录。	--tmp /data
--overwrite	使用覆盖同步功能，即使用源路径完全覆盖目标路径。	--overwrite
--ignore	忽略数据迁移期间发生的异常，相关报错不会中断任务，并最终以DistCp Counter的形式透出。如果使用了--enableCMS，也会通过指定方式进行通知。	--ignore

附录二：场景示例

JindoDistCp提供了以下两种方式用于验证数据的完整性。

- 方式一：DistCp Counters

  通过Distcp Counters信息中包含的BYTES_EXPECTED、FILES_EXPECTED等参数验证数据完整性。

  示例

```
JindoDistcpCounter
    BYTES_COPIED=10000
    BYTES_EXPECTED=10000
    FILES_COPIED=11
    FILES_EXPECTED=11
    ...
Shuffle Errors
    BAD_ID=0
    CONNECTION=0
    IO_ERROR=0
    WRONG_LENGTH=0
    WRONG_MAP=0
    WRONG_REDUCE=0
```

  示例中可能包含的Counter参数如下：

参数	说明
BYTES_COPIED	拷贝成功的字节数。

参数	说明
BYTES_EXPECTED	预期拷贝的字节数。
FILES_COPIED	拷贝成功的文件数。
FILES_EXPECTED	预期拷贝的文件数。
FILES_SKIPPED	增量拷贝时跳过的文件数。
BYTES_SKIPPED	增量拷贝时跳过的字节数。
COPY_FAILED	拷贝失败的文件数，不为0时触发告警。
BYTES_FAILED	拷贝失败的字节数。
DIFF_FILES	源路径与目标路径下不相同的文件数，不为0时触发告警。
DIFF_FAILED	文件比较操作异常的文件数，并计入DIFF_FILE。
SRC_MISS	源路径下不存在的文件数，并计入DIFF_FILES。
DST_MISS	目标路径下不存在的文件数，并计入DIFF_FILES。
LENGTH_DIFF	源文件和目标文件大小不一致的数量，并计入DIFF_FILES。
CHECKSUM_DIFF	Checksum校验失败的文件数，并计入COPY_FAILED。
SAME_FILES	源路径与目标路径下完全相同的文件数。

- 方式二：通过*--diff*选项

  您可以在示例中结合*--diff*选项对源路径和目标路径下文件名和文件大小进行比较。

```
hadoop jar jindo-distcp-3.7.3.jar --src /data/incoming --dest oss://examplebucket/incomin
g --ossKey LTAI5t7h6SgiLSganP2m**** --ossSecret KZo149BD9GLPNiDIEmdQ7dyNKG**** --ossEndPo
int oss-cn-hangzhou.aliyuncs.com --diff
```

1. 您可以在示例中结合*--diff*选项查看HDFS指定路径下的文件是否都已迁移至OSS。

```
hadoop jar jindo-distcp-3.7.3.jar --src /data/incoming --dest oss://examplebucket/incom
ing --ossKey LTAI5t7h6SgiLSganP2m**** --ossSecret KZo149BD9GLPNiDIEmdQ7dyNKG**** --ossE
ndPoint oss-cn-hangzhou.aliyuncs.com --diff
```

   如果所有文件都已迁移完成，则提示如下信息，否则在执行目录下会生成一个manifest文件。

```
INFO distcp.JindoDistCp: Jindo DistCp job exit with 0
```

2. 对于生成的manifest文件，您可以使用*--copyFromManifest*和*--previousManifest*选项迁移剩余文件。

```
hadoop jar jindo-distcp-3.7.3.jar --src /data/incoming --dest oss://examplebucket/incom
ing --previousManifest=file:///opt/manifest-2020-04-17.gz --copyFromManifest --parallel
ism 20
```

其中，*--previousManifest*选项后指定的*file:///opt/manifest-2020-04-17.gz*为当前执行命令的本地路径。

您可以在示例中添加*--policy*选项来指定写入OSS文件的存储类型。以下以指定为低频访问类型为例：

```
hadoop jar jindo-distcp-3.7.3.jar --src /data/incoming --dest oss://examplebucket/incoming
--ossKey LTAI5t7h6SgiLSganP2m**** --ossSecret KZo149BD9GLPNiDIEmdQ7dyNKG**** --ossEndPoint
oss-cn-hangzhou.aliyuncs.com --policy ia --parallelism 10
```

如果需要指定为归档存储类型，请将*--policy ia*替换为*--policy archive*。如需指定为冷归档存储类型，请将*--policy ia*替换为*--policy coldArchive*。此外，目前冷归档存储仅支持部分地域，更多信息，请参见冷归档存储（Cold Archive）。

- 小文件较多，大文件较大

  例如HDFS源路径下包含50万个大小为100 KB左右的文件，10个5 TB大小的文件，此时您可以结合*--enableDynamicPlan*选项优化数据传输速度。

  ```
  hadoop jar jindo-distcp-3.7.3.jar --src /data/incoming --dest oss://examplebucket/incomin
  g --ossKey LTAI5t7h6SgiLSganP2m**** --ossSecret KZo149BD9GLPNiDIEmdQ7dyNKG**** --ossEndPo
  int oss-cn-hangzhou.aliyuncs.com --enableDynamicPlan --parallelism 10
  ```

- 文件大小无明显差异

  例如HDFS源路径下包含100个大小为200 KB的文件，此时您可以结合*--enableBalancePlan*选项优化数据传输速度。

  ```
  hadoop jar jindo-distcp-3.7.3.jar --src /data/incoming --dest oss://examplebucket/incomin
  g --ossKey LTAI5t7h6SgiLSganP2m**** --ossSecret KZo149BD9GLPNiDIEmdQ7dyNKG**** --ossEndPo
  int oss-cn-hangzhou.aliyuncs.com --enableBalancePlan --parallelism 10
  ```

> **说明** 不支持在同一个示例中同时使用*--enableDynamicPlan*以及*--enableBalancePlan*选项。

您可以结合*--deleteOnSuccess*选项，在迁移或者拷贝任务完成后，仅保留OSS目标路径下的数据，并删除HDFS源路径下的指定数据。

```
hadoop jar jindo-distcp-3.7.3.jar --src /data/incoming --dest oss://examplebucket/incoming
--ossKey LTAI5t7h6SgiLSganP2m**** --ossSecret KZo149BD9GLPNiDIEmdQ7dyNKG**** --ossEndPoint
oss-cn-hangzhou.aliyuncs.com --deleteOnSuccess --parallelism 10
```

场景一：使用JindoDistCp成功传输数据后，如何验证数据完整性？

场景二：从HDFS迁移数据到OSS过程中，迁移任务可能随时失败，要想支持断点续传，该使用哪些参数？

场景三：如果要以低频访问、归档或者冷归档的方式存储写入OSS的文件，该使用哪些参数？

场景四：了解待迁移或拷贝的源路径下数据的分布情况后，例如大文件和小文件的占比，该使用哪些参数来优化数据传输速度？

场景五：迁移或者拷贝任务完成后，希望仅保留目标路径下的数据，且删除源路径下的指定数据，该使用哪些参数？

8.6. 从HDFS迁移数据到OSS-HDFS

本文介绍如何使用阿里云Jindo DistCp从HDFS迁移数据到OSS-HDFS。

前提条件

- JDK 1.8及以上版本。
- 如果您使用的是自建ECS集群，需要具备Hadoop2.7+或Hadoop3.x环境以及进行MapReduce作业的能力。
- 如果您使用的是阿里云E-MapReduce，需使用EMR-5.6.0及后续版本或EMR-3.40.0及后续版本。

背景信息

阿里云Jindo DistCp（分布式文件拷贝工具）用于大规模集群内部或集群之间拷贝文件。Jindo DistCp使用MapReduce实现文件分发，错误处理和恢复，把文件和目录的列表作为MapReduce任务的输入，每个任务会完成源列表中部分文件的拷贝。Jindo DistCp全量支持HDFS之间、HDFS与OSS之间、HDFS与OSS-HDFS之间以及OSS-HDFS之间数据拷贝场景，提供多种个性化拷贝参数和多种拷贝策略。

使用阿里云Jindo DistCp迁移数据时，有以下优势：

- 效率高，在测试场景中最高可达到1.59倍的加速。
- 基本功能丰富，提供多种拷贝方式和场景优化策略。
- 深度结合OSS，对文件提供归档、压缩等操作。
- 实现No-Rename拷贝，保证数据一致性。
- 场景全面，可完全替代Hadoop DistCp，目前支持Hadoop2.7+和Hadoop3.x。

步骤一：下载JAR包

JindoData 4.x.x版本

步骤二：配置OSS-HDFS服务的AccessKey

您可以通过以下任意方式配置OSS-HDFS服务的AccessKey：

- 在示例命令中配置AccessKey

  例如，在将HDFS中的/data路径的数据迁移到OSS-HDFS指定路径的示例中结合*--hadoopConf*选项配置AccessKey。

  ```
  hadoop jar jindo-distcp-tool-${version}.jar --src data/ --dest oss://destbucket.cn-hangzhou.oss-dls.aliyuncs.com/dir/ --hadoopConf fs.oss.accessKeyId=yourkey --hadoopConf fs.oss.accessKeySecret=yoursecret --parallelism 10
  ```

- 通过配置文件预先配置AccessKey

  将OSS-HDFS的fs.oss.accessKeyId、fs.oss.accessKeySecret预先配置在Hadoop的core-site.xml文件里。示例命令如下：

```
<configuration>
    <property>
        <name>fs.oss.accessKeyId</name>
        <value>LTAI5t7h6SgiLSganP2m****</value>
    </property>
    <property>
        <name>fs.oss.accessKeySecret</name>
        <value>KZo149BD9GLPNiDIEmdQ7dyNKG****</value>
    </property>
</configuration>
```

步骤三：配置OSS-HDFS服务Endpoint

访问OSS-HDFS服务时需要配置Endpoint。推荐访问路径格式为 `oss://<Bucket>.<Endpoint>/<Object>`，例如 `oss://examplebucket.cn-shanghai.oss-dls.aliyuncs.com/exampleobject.txt`。配置完成后，JindoSDK会根据访问路径中的Endpoint访问对应的OSS-HDFS服务接口。

您还可以通过其他方式配置OSS-HDFS服务Endpoint，且不同方式配置的Endpoint存在生效优先级。更多信息，请参见配置Endpoint的其他方式。

步骤四：将HDFS指定路径下的数据全量迁移至OSS-HDFS

以下以Jindo DistCp 4.4.0版本为例，您可以根据实际环境替换对应的版本号。

- 命令格式

```
hadoop jar jindo-distcp-tool-${version}.jar --src path --dest oss://bucketname.region.oss-dls.aliyuncs.com/path --hadoopConf fs.oss.accessKeyId=yourkey --hadoopConf fs.oss.accessKeySecret=yoursecret --parallelism 10
```

参数及选项说明如下：

参数及选项	说明	示例
--src	待迁移或拷贝的HDFS数据所在的路径。	data/
--dest	OSS-HDFS中存放迁移或拷贝数据的目标路径。	oss://destbucket.cn-hangzhou.oss-dls.aliyuncs.com/dir/

--hadoopConf	访问OSS-HDFS服务的AccessKey ID和AccessKey Secret。	◦ AccessKey ID
		`LTAI5t7h6SgiLSganP2m****`
		◦ AccessKey Secret
		`KZo149BD9GLPNiDIEmdQ7dyNKG******`

--parallelism	根据集群资源调整任务并发数。	10

- 使用示例

将HDFS指定目录*data/*下的数据全量迁移或拷贝到OSS-HDFS目标路径*oss://destbucket.cn-hangzhou.oss-dls.aliyuncs.com/dir/*下。

```
hadoop jar jindo-distcp-tool-4.4.0.jar --src data/ --dest oss://destbucket.cn-hangzhou.oss-dls.aliyuncs.com/dir/ --hadoopConf fs.oss.accessKeyId=LTAI5t7h6SgiLSganP2m**** --hadoopConf fs.oss.accessKeySecret=KZo149BD9GLPNiDIEmdQ7dyNKG**** --parallelism 10
```

（可选）将HDFS指定路径下的数据增量迁移至OSS-HDFS

如果您仅希望拷贝在上一次全量迁移或拷贝后源路径下新增的数据，此时您可以结合--update选项完成数据的增量迁移或拷贝。

将HDFS指定目录*data/*下的数据增量迁移或拷贝到OSS-HDFS目标路径*oss://destbucket.cn-hangzhou.oss-dls.aliyuncs.com/dir/*下。

```
hadoop jar jindo-distcp-tool-4.4.0.jar --src data/ --dest oss://destbucket.cn-hangzhou.oss-dls.aliyuncs.com/dir/ --hadoopConf fs.oss.accessKeyId=LTAI5t7h6SgiLSganP2m**** --hadoopConf fs.oss.accessKeySecret=KZo149BD9GLPNiDIEmdQ7dyNKG**** --update --parallelism 10
```

更多参考

关于Jindo DistCp的其他使用场景，请参见Jindo DistCp使用说明。

8.7. 在OSS-HDFS服务不同Bucket之间迁移数据

本文介绍如何使用阿里云Jindo DistCp在OSS-HDFS服务不同Bucket之间迁移数据。

前提条件

- JDK 1.8及以上版本。
- 如果您使用的是自建ECS集群，需要具备Hadoop2.7+或Hadoop3.x环境以及进行MapReduce作业的能力。
- 如果您使用的是阿里云E-MapReduce，需使用EMR-5.6.0及后续版本或EMR-3.40.0及后续版本。

背景信息

阿里云Jindo DistCp（分布式文件拷贝工具）用于大规模集群内部或集群之间拷贝文件。Jindo DistCp使用MapReduce实现文件分发，错误处理和恢复，把文件和目录的列表作为MapReduce任务的输入，每个任务会完成源列表中部分文件的拷贝。Jindo DistCp全量支持HDFS之间、HDFS与OSS之间、HDFS与OSS-HDFS之间以及OSS-HDFS之间数据拷贝场景，提供多种个性化拷贝参数和多种拷贝策略。

使用阿里云Jindo DistCp迁移数据时，有以下优势：

- 效率高，在测试场景中最高可达到1.59倍的加速。
- 基本功能丰富，提供多种拷贝方式和场景优化策略。
- 深度结合OSS，对文件提供归档、压缩等操作。
- 实现No-Rename拷贝，保证数据一致性。
- 场景全面，可完全替代Hadoop DistCp，目前支持Hadoop2.7+和Hadoop3.x。

步骤一：下载JAR包

JindoData 4.x.x版本

步骤二：配置OSS-HDFS服务的AccessKey

您可以通过以下任意方式配置OSS-HDFS服务的AccessKey：

- 在示例命令中配置AccessKey

  例如，在将OSS-HDFS中srcbucket的数据迁移到destbucket的示例中结合*--hadoopConf*选项配置AccessKey。

```
hadoop jar jindo-distcp-tool-${version}.jar --src oss://srcbucket.cn-hangzhou.oss-dls.aliyuncs.com/ --dest oss://destbucket.cn-hangzhou.oss-dls.aliyuncs.com/ --hadoopConf fs.oss.accessKeyId=yourkey --hadoopConf fs.oss.accessKeySecret=yoursecret --parallelism 10
```

- 通过配置文件预先配置AccessKey

  将OSS-HDFS的fs.oss.accessKeyId、fs.oss.accessKeySecret预先配置在Hadoop的core-site.xml文件里。示例命令如下：

```
<configuration>
    <property>
        <name>fs.oss.accessKeyId</name>
        <value>LTAI5t7h6SgiLSganP2m****</value>
    </property>
    <property>
        <name>fs.oss.accessKeySecret</name>
        <value>KZo149BD9GLPNiDIEmdQ7dyNKG****</value>
    </property>
</configuration>
```

步骤三：配置OSS-HDFS服务Endpoint

访问OSS-HDFS服务时需要配置Endpoint。推荐访问路径格式为 `oss://<Bucket>.<Endpoint>/<Object>`，例如 `oss://examplebucket.cn-shanghai.oss-dls.aliyuncs.com/exampleobject.txt`。配置完成后，JindoSDK会根据访问路径中的Endpoint访问对应的OSS-HDFS服务接口。

您还可以通过其他方式配置OSS-HDFS服务Endpoint，且不同方式配置的Endpoint存在生效优先级。更多信息，请参见配置Endpoint的其他方式。

步骤四：在OSS-HDFS服务不同Bucket之间全量迁移数据

以下以Jindo DistCp 4.4.0版本为例，您可以根据实际环境替换对应的版本号。

使用相同的AccessKey将同一个Region下Bucket A的数据迁移至Bucket B

- 命令格式

```
hadoop jar jindo-distcp-tool-${version}.jar --src oss://bucketname.region.oss-dls.aliyuncs.com/ --dest oss://bucketname.region.oss-dls.aliyuncs.com/ --hadoopConf fs.oss.accessKeyId=yourkey --hadoopConf fs.oss.accessKeySecret=yoursecret --parallelism 10
```

参数及选项说明如下：

参数及选项	说明	示例
--src	待迁移或拷贝的OSS-HDFS服务中源Bucket的完整路径。	oss://srcbucket.cn-hangzhou.oss-dls.aliyuncs.com/

参数及选项	说明	示例
--dest	OSS-HDFS中存放迁移或拷贝数据的目标Bucket的完整路径。	oss://destbucket.cn-hangzhou.oss-dls.aliyuncs.com/

--hadoopConf	访问OSS-HDFS服务的AccessKey ID和AccessKey Secret。	◦ AccessKey ID
		`LTAI5t7h6SgiLSganP2m****`
		◦ AccessKey Secret
		`KZo149BD9GLPNiDIEmdQ7dyNKG******`

--parallelism	根据集群资源调整任务并发数。	10

- 使用示例

  使用相同的AccessKey将华东1（杭州）地域下srcbucket的数据迁移至destbucket。

```
hadoop jar jindo-distcp-tool-4.4.0.jar --src oss://srcbucket.cn-hangzhou.oss-dls.aliyuncs.com/ --dest oss://destbucket.cn-hangzhou.oss-dls.aliyuncs.com/ --hadoopConf fs.oss.accessKeyId=LTAI5t7h6SgiLSganP2m**** --hadoopConf fs.oss.accessKeySecret=KZo149BD9GLPNiDIEmdQ7dyNKG**** --parallelism 10
```

使用不同的AccessKey将Region A下Bucket A的数据迁移至另一个Region下的Bucket B

- 命令格式

```
hadoop jar jindo-distcp-tool-${version}.jar --src oss://srcbucketname.region.oss-dls.aliyuncs.com/ --dest oss://destbucketname.region.oss-dls.aliyuncs.com/ --hadoopConf fs.oss.bucket.srcbucketname.accessKeyId=yourkey --hadoopConf fs.oss.bucket.srcbucketname.accessKeySecret=yoursecret --hadoopConf fs.oss.bucket.destbucketname.accessKeyId=yourkey --hadoopConf fs.oss.bucket.destbucketname.accessKeySecret=yoursecret --parallelism 10
```

参数及选项说明如下：

参数及选项	说明	示例
--src	待迁移或拷贝的OSS-HDFS服务中源Bucket的完整路径。	oss://srcbucket.cn-hangzhou.oss-dls.aliyuncs.com/
--dest	OSS-HDFS中存放迁移或拷贝数据的目标Bucket的完整路径。	oss://destbucket.cn-shanghai.oss-dls.aliyuncs.com/

参数及选项	说明	示例

--hadoopConf	访问OSS-HDFS服务中源Bucket以及目标Bucket的AccessKey ID和AccessKey Secret。	○ 访问源Bucket的AccessKey
		AccessKey ID
		`LTAI5t7h6SgiLSganP2m****`
		AccessKey Secret
		`KZo149BD9GLPNiDIEmdQ7dyNKG******`
		○ 访问目标Bucket的AccessKey
		AccessKey ID
		`LTAI5t8K9BtVqPSxjdDX****`
		AccessKey Secret
		`5xwFyoOjgKmt05VCdqzrwBdAwm******`

--parallelism	根据集群资源调整任务并发数。	10

- 使用示例

  将华东1（杭州）地域OSS-HDFS服务中的srcbucket的数据迁移至华东2（上海）地域的destbucket，且这两个Bucket需使用不同的AccessKey进行访问。

```
hadoop jar jindo-distcp-tool-4.4.0.jar --src oss://srcbucket.cn-hangzhou.oss-dls.aliyuncs.com/ --dest oss://destbucket.cn-shanghai.oss-dls.aliyuncs.com/ --hadoopConf fs.oss.bucket.srcbucket.accessKeyId=LTAI5t7h6SgiLSganP2m**** --hadoopConf fs.oss.bucket.srcbucket.accessKeySecret=KZo149BD9GLPNiDIEmdQ7dyNKG**** --hadoopConf --hadoopConf fs.oss.bucket.destbucket.accessKeyId=LTAI5t8K9BtVqPSxjdDX**** --hadoopConf fs.oss.bucket.destbucket.accessKeySecret=5xwFyoOjgKmt05VCdqzrwBdAwm**** --parallelism 10
```

（可选）在OSS-HDFS服务不同Bucket之间增量迁移数据

如果您仅希望拷贝在上一次全量迁移或拷贝后源路径下新增的数据，此时您可以结合--update选项完成数据的增量迁移或拷贝。

例如，您需要使用相同的AccessKey将华东1（杭州）地域下srcbucket的数据增量迁移至destbucket。

```
hadoop jar jindo-distcp-tool-4.4.0.jar --src oss://srcbucket.cn-hangzhou.oss-dls.aliyuncs.com/ --dest oss://destbucket.cn-hangzhou.oss-dls.aliyuncs.com/ --hadoopConf fs.oss.accessKeyId=LTAI5t7h6SgiLSganP2m**** --hadoopConf fs.oss.accessKeySecret=KZo149BD9GLPNiDIEmdQ7dyNKG****** --update --parallelism 10
```

更多参考

关于Jindo DistCp的其他使用场景，请参见Jindo DistCp使用说明。

8.8. 将半托管JindoFS集群迁移到OSS-HDFS服务

本文介绍如何将半托管JindoFS集群迁移到OSS-HDFS服务。

前提条件

- 半托管JindoFS集群对应的OSS Bucket已开通OSS-HDFS服务。
- 半托管JindoFS集群已开启Audit Log。

步骤一：全量迁移

全量迁移模式负责将半托管JindoFS集群中的某个目录中的元数据一次性全量迁移OSS-HDFS服务中的某个目录。目前仅支持迁移到OSS-HDFS服务中的一级子目录。

- 命令格式

```
jindo distjob -migrateImport -srcPath <srcPath> -destPath <desPth> -backendLoc <backendLoc>
```

参数说明如下：

参数	说明
-srcPath	待迁移的半托管JindoFS集群的源路径。
-destPath	OSS-HDFS服务中存放半托管JindoFS集群的目标路径。
-backendLoc	半托管JindoFS集群的源数据块对应的OSS路径。

- 使用示例

将半托管JindoFS集群中某个目录*jfs://mycluster/foo*全量迁移到OSS-HDFS服务的bar目录中，OSS-HDFS服务所在的Bucket名称为examplebucket。

```
jindo distjob -migrateImport -srcPath jfs://mycluster/foo -destPath oss://examplebucket/bar/
```

（可选）步骤二：增量迁移

1. 生成ChangeLog。

   如果需要将半托管JindoFS集群增量迁移至OSS-HDFS服务，您需要通过Jindo工具将半托管JindoFS集群的Audit Log转换成对应目录的变更日志ChangeLog。

   - 命令格式

   ```
   jindo distjob -mkchangelog -auditLogDir <auditLogDir> -changeLogDir <changeLogDir> -startTime <startTime>
   ```

   参数说明如下：

参数	说明
-auditLogDir	半托管JindoFS集群的AuditLog所在路径。
-changeLogDir	变更日志ChangeLog输出路径。
-startTime	处理AuditLog的起始时间。

- 使用示例

  半托管JindoFS集群中AuditLog的路径为*oss://examplebucket/sysinfo/auditlog*，待输出目录的变更日志存放在*oss://examplebucket/sysinfo/changelog*下，且只处理从2022年06月01日开始的AuditLog。

```
jindo distjob -mkchangelog -auditLogDir oss://examplebucket/sysinfo/auditlog -changeL
ogDir oss://examplebucket/sysinfo/changelog -startTime 2022-06-01T12:00:00Z
```

2. 单次增量迁移。

   半托管JindoFS集群产生的元数据的增量更新会通过对应的AuditLog转换为ChangeLog，然后迁移至OSS-HDFS服务。

   - 命令格式

```
jindo distjob -migrateImport -srcPath <srcPath> -destPath <desPth> -changeLogDir <cha
ngeLogDir> -backendLoc <backendLoc> -update
```

   参数说明如下：

参数	说明
-srcPath	待迁移的半托管JindoFS集群的源路径。
-destPath	OSS-HDFS服务中存放半托管JindoFS集群的目标路径。
-changeLogDir	变更日志ChangeLog输出路径。
-backendLoc	半托管JindoFS集群的源数据块对应的OSS路径。
-update	开启增量迁移模式。

   - 使用示例

     将半托管JindoFS集群中某个目录*jfs://mycluster/foo*增量迁移到OSS-HDFS服务的bar目录中，OSS-HDFS服务所在的Bucket名称为examplebucket，变更日志ChangeLog输出路径为*oss://logbucket/logdir/*。

```
jindo distjob -migrateImport -srcPath jfs://mycluster/foo -destPath oss://examplebuck
et/bar/ -changeLogDir oss://logbucket/logdir/ -backendLoc oss://examplebucket/jfsdata
Dir -update
```

3. （可选）多次增量迁移。

   如果需要将半托管JindoFS集群中多次增量迁移到OSS-HDFS服务，您可以通过修改-startTime参数自定义处理Auditlog的起始时间，然后多次重复步骤1和步骤2。

9.OSS安全

9.1. 降低因账号密码泄露带来的未授权访问风险

如果因个人或者企业账号密码泄露引发了未经授权的访问，可能会出现非法用户对OSS资源进行违法操作，或者合法用户以未授权的方式对OSS资源进行各类操作，这将给数据安全带来极大的威胁。为此，OSS提供了在实施数据安全保护时需要考虑的多种安全最佳实践。

> **注意** 以下最佳实践遵循一般准则，并不等同完整的安全解决方案。这些最佳实践可能不适合您的环境或不满足您的环境要求，仅建议将其视为参考因素。请您在日常使用中提高数据安全意识并时刻做好内容安全防范措施。

阻止公共访问权限

除非您明确要求包括匿名访问者在内的任何人都能读写您的OSS资源，包括存储空间（Bucket）以及文件（Object），否则请勿将Bucket或者Object的读写权限ACL设置为公共读（public-read）或者公共读写（public-read-write）。设置公共读或者公共读写权限后，对访问者的权限说明如下：

- 公共读写：任何人（包括匿名访问者）都可以对该Bucket内的Object进行读写操作。

  > **警告** 互联网上任何用户都可以对该Bucket内的Object进行访问，并且向该Bucket写入数据。这有可能造成您数据的外泄以及费用激增，若被人恶意写入违法信息还可能会侵害您的合法权益。除特殊场景外，不建议您配置公共读写权限。

- 公共读：只有该Bucket的拥有者可以对该Bucket内的Object进行写操作，任何人（包括匿名访问者）都可以对该Bucket内的Object进行读操作。

  > **警告** 互联网上任何用户都可以对该Bucket内的Object进行访问，这有可能造成您数据的外泄以及费用激增，请谨慎操作。

鉴于公共读或者公共读写权限对OSS资源带来的数据安全风险考虑，强烈建议您将Bucket或者Object读写权限设置为私有（private）。设置为私有权限后，只有该Bucket拥有者可以对该Bucket以及Bucket内的Object进行读写操作，其他人均无访问权限。

您可以通过多种方式将Bucket或者Object的读写权限设置为私有，具体步骤请参见设置存储空间读写权限ACL以及设置文件读写权限ACL。

避免代码明文使用AccessKey或本地加密存储AccessKey

代码中明文使用AccessKey会由于各种原因的代码泄露导致AccessKey泄露。而本地加密存储AccessKey也并不安全，原因是数据的加解密内容会存放在内存中，而内存中的数据可以被转储。尤其是移动App和PC桌面应用极易出现此类问题，攻击者只需要使用某些注入、API HOOK、动态调试等技术，就可以获取到加解密后的数据。

服务端可以通过阿里云SDK托管凭据插件的方式规避代码明文使用AccessKey，解决因源码或编译产物泄露而导致的AccessKey泄露问题。有关阿里云SDK托管凭据插件的工作原理及使用方式的更多信息，请参见多种阿里云SDK的托管凭据插件。

> **注意** 此方案不适用于客户端，请不要以任何方式在客户端内置任意形式的AccessKey。

以RAM用户的方式访问OSS

阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM用户进行API访问或日常运维。

创建RAM用户后，您可以控制这些RAM用户对资源的操作权限。当您的企业存在多用户协同操作资源的场景时，可以让您避免与其他用户共享阿里云账号密钥，按需为用户分配最小权限，从而降低企业的信息安全风险。创建RAM用户的具体步骤，请参见创建RAM用户。

RAM用户创建完成后，您可以通过RAM Policy为RAM用户授权的方式来集中管理您的用户（例如员工、系统或应用程序），以及控制用户可以访问您名下哪些资源的权限。例如，您可以通过以下RAM Policy阻止指定RAM用户访问目标存储空间examplebucket及examplebucket内的文件或目录。

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

您还可以通过RAM Policy拒绝RAM用户删除某个Bucket下任意文件、或者授权RAM用户仅拥有读取某个Bucket资源的权限等。有关RAM Policy常见场景的授权示例，请参见RAM Policy常见示例。

启用多因素认证

多因素认证MFA（Multi Factor Authentication）是一种简单有效的最佳安全实践。启用MFA后，登录阿里云控制台时需要输入账号密码和MFA设备实时生成的动态验证码，在账号密码泄露时也可以阻止未授权访问，提高账号安全性。

您可以选择为阿里云账号启用MFA，具体步骤请参见为阿里云账号启用多因素认证。您还可以选择为RAM用户启用MFA，具体步骤请参见为RAM用户启用多因素认证。

STS临时授权访问OSS

您可以通过STS服务给其他用户颁发一个临时访问凭证。该用户可使用临时访问凭证在规定时间内访问您的OSS资源。临时访问凭证无需透露您的长期密钥，使您的OSS资源访问更加安全。

有关STS临时授权访问OSS的具体步骤，请参见开发指南中的使用STS临时访问凭证访问OSS。

Bucket Policy

Bucket Policy是阿里云OSS推出的针对Bucket的授权策略，您可以通过Bucket Policy授权其他用户访问您指定的OSS资源。通过Bucket Policy，您可以授权另一个账号访问或管理整个Bucket或Bucket内的部分资源，或者对同账号下的不同RAM用户授予访问或管理Bucket资源的不同权限。

配置Bucket Policy时，请遵循以下权限最小化原则，从而降低数据安全风险。

- 避免授权整个Bucket

  资源授权过大容易导致其它用户数据被非法访问，所以在实际生产业务中请限制指定资源途径，避免授权整个Bucket。

- 不授权匿名访问

  允许匿名账号访问意味着用户只需要知道Endpoint和Bucket名称就可以正常访问OSS，而EndPoint是可以枚举的，Bucket名称也可以从已授权的访问文件URL中提取。由此可见，授权允许匿名账号访问会带来极大的安全风险。

- 设置合理的Action

  通过控制台的图形化配置Bucket Policy时，简单设置中的四类授权操作（Action）仅为用户提供了一种便捷的Policy设置方法，每类Action并不一定完全符合您的业务需求，建议您结合业务需求并通过高级设置的方式给予授权用户最小的权限。例如，生成只读权限往往会包含 `oss:ListObjects` 、 `oss:GetObject` 等。但是一般场景下的文件下载只需要 `oss:GetObject` 权限即可。

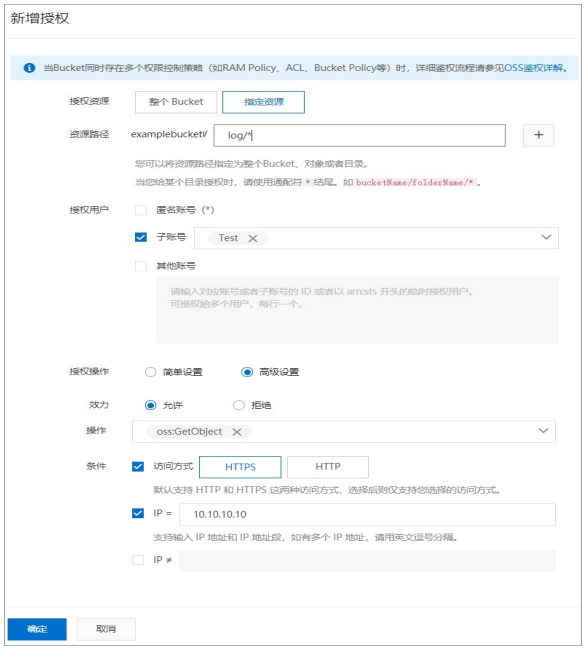
- 启用HTTPS访问

  启用HTTPS访问可以解决网络中间人攻击以及域名劫持等问题。另外Chrome浏览器也对HTTP协议进行干预，例如HTTPS站点默认无法加载HTTP资源。基于以上原因，请务必开启HTTPS，以最低成本解决多种风险问题。

- 限定来源的IP地址

  如果访问OSS资源的IP地址是固定可枚举的，强烈建议配置IP地址。

例如，通过Bucket Policy授予名为Test的RAM用户通过SDK或者命令行工具ossutil下载目标存储空间examplebucket下指定目录log下所有文件的权限：



9.2. 降低因恶意访问流量导致大额资金损失的风险

当您的存储空间（Bucket）被恶意攻击、流量被恶意盗刷时，会出现高带宽或者大流量突发的情况，进而产生高于日常消费金额的账单。如果您希望降低因类似情况带来的大额资金损失的风险，请参考本文提供的多种安全最佳实践。

> **注意** 以下最佳实践遵循一般准则，并不等同完整的安全解决方案。这些最佳实践可能不适合您的环境或不满足您的环境要求，仅建议将其视为参考因素。请您在日常使用中提高数据安全意识并时刻做好内容安全防范措施。

阻止公共访问权限

除非您明确要求包括匿名访问者在内的任何人都能读写您的OSS资源，包括存储空间（Bucket）以及文件（Object），否则请勿将Bucket或者Object的读写权限ACL设置为公共读（public-read）或者公共读写（public-read-write）。设置公共读或者公共读写权限后，对访问者的权限说明如下：

- 公共读写：任何人（包括匿名访问者）都可以对该Bucket内的Object进行读写操作。

  > **警告** 互联网上任何用户都可以对该Bucket内的Object进行访问，并且向该Bucket写入数据。这有可能造成您数据的外泄以及费用激增，若被人恶意写入违法信息还可能会侵害您的合法权益。除特殊场景外，不建议您配置公共读写权限。

- 公共读：只有该Bucket的拥有者可以对该Bucket内的Object进行写操作，任何人（包括匿名访问者）都可以对该Bucket内的Object进行读操作。

  > **警告** 互联网上任何用户都可以对该Bucket内的Object进行访问，这有可能造成您数据的外泄以及费用激增，请谨慎操作。

鉴于公共读或者公共读写权限对OSS资源带来的数据安全风险考虑，强烈建议您将Bucket或者Object读写权限设置为私有（private）。设置为私有权限后，只有该Bucket拥有者可以对该Bucket以及Bucket内的Object进行读写操作，其他人均无访问权限。

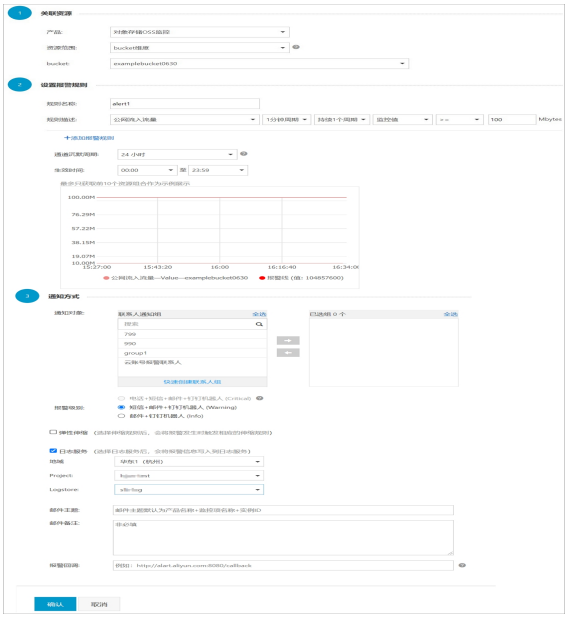
您可以通过多种方式将Bucket或者Object的读写权限设置为私有，具体步骤请参见设置存储空间读写权限ACL以及设置文件读写权限ACL。

通过云监控配置报警规则

当您需要监控OSS资源的使用和运行情况时，可以通过云监控创建阈值报警规则，实现监控指标超过静态阈值或动态阈值后自动发送报警通知的功能，帮助您及时了解监控数据异常并快速进行处理。

例如，您可以为目标存储空间examplebucket配置报警规则，并在报警规则中指定当连续一次出现公网流入流量、公网流出流量、CDN流入流量、CDN流出流量等在其大于或等于100 Mbytes，以短信+邮件+钉钉机器人的方式通知告警，并将告警信息写入日志服务指定的Logstore中。

以公网流入流量大于或者等于100 Mbytes时触发报警为例，其报警规则配置如下：



您可以基于Bucket的维度配置报警规则，您还可以为当前阿里云账号下的所有OSS资源配置报警规则。有关报警规则的配置步骤，请参见创建报警规则。

配置防盗链

OSS支持对Bucket设置防盗链，即通过对访问来源设置白名单的机制，避免OSS资源被其他人盗用。

防盗链通过请求Header中的Referer地址判断访问来源。当浏览器向Web服务器发送请求的时候，请求Header中将包含Referer，用于告知Web服务器该请求的页面链接来源。OSS根据浏览器附带的Referer与用户配置的Referer规则来判断允许或拒绝此请求，如果Referer一致，则OSS将允许该请求的访问；如果Referer不一致，则OSS将拒绝该请求的访问。

如下图所示，Referer设置为 `https://www.example.com` ，且不允许空Referer。

完成上述防盗链配置后，如果用户A在 `https://www.example.com` 嵌入了exampleobject.png图片，当浏览器请求访问此图片时会带上 `https://www.example.com` 的Referer，此场景下OSS将允许该请求的访问。

此时，如果用户B盗用了exampleobject.png的图片并将其嵌入 `https://example.org` ，当浏览器请求访问此图片时会带上 `https://example.org` 的Referer，此场景下OSS将拒绝该请求的访问。

有关防盗链的更多信息，请参见开发指南中的防盗链。

设置跨域资源共享

跨域资源共享CORS（Cross-Origin Resource Sharing）简称跨域访问，是HTML5提供的标准跨域解决方案，允许Web应用服务器进行跨域访问控制，确保跨域数据传输的安全性。跨域访问是浏览器出于安全考虑而设置的限制，是一种用于隔离潜在恶意文件的关键安全机制。当A、B两个网站属于不同域时，来自于A网站页面中的JavaScript代码访问B网站时，浏览器会拒绝该访问。

OSS支持根据需求灵活配置CORS规则，实现允许或者拒绝相应的跨域请求。例如，您希望仅允许来源为 `www.aliyun.com` 、跨域请求方法为 `GET` 的请求，则CORS规则配置如下：

有关CORS的配置详情，请参见设置跨域资源共享。

避免使用顺序前缀的方式命名文件

当您上传大量文件时，如果使用顺序前缀（如时间戳或字母顺序）、日期、数字ID等可以被遍历的方式来命名文件，攻击者可通过总结规律以及编写脚本的方式获取全部的文件，最终造成数据泄露。强烈建议您通过向文件名添加十六进制哈希前缀或以反转文件名的方式命名文件，从而有效降低文件名被遍历的风险。具体操作，请参见数据安全。

9.3. 降低因操作失误等原因导致数据丢失的风险

我们经常遇到客户反馈因不小心的误操作导致本不该删除的数据被误删除，从而导致业务无法正常运转。如果您也遇到过类似的情况或希望尽可能避免类似情况的发生，请结合本文选用合适的规避方式。

> **注意** 以下最佳实践遵循一般准则，并不等同完整的安全解决方案。这些最佳实践可能不适合您的环境或不满足您的环境要求，仅建议将其视为参考因素。请您在日常使用中提高数据安全意识并时刻做好内容安全防范措施。

为Bucket开启版本控制

版本控制是针对存储空间（Bucket）级别的数据保护功能。开启版本控制后，针对数据的覆盖和删除操作将会以历史版本的形式保存下来。在对象（Object）被错误覆盖或者误删除后，您能够将Bucket中存储的Object恢复至任意时刻的历史版本。

开启版本控制后，OSS会为Bucket中所有Object的每个版本指定唯一的versionId。请通过控制台按如下步骤开启版本控制：

- 新建Bucket时开启版本控制。
  i. 登录OSS管理控制台。
  ii. 单击**Bucket列表**，然后单击**创建Bucket**。
  iii. 在**创建Bucket**页面配置各项参数。

其中，**版本控制**区域选择**开通**。其他参数的配置详情，请参见创建存储空间。

iv. 单击**确定**。

- 对已创建的Bucket开启版本控制。
  i. 单击**Bucket列表**，然后单击目标Bucket名称。
  ii. 在左侧导航栏，选择**冗余与容错 > 版本控制**。
  iii. 单击**设置**，然后版本控制状态选择**开通**。
  iv. 单击**保存**。

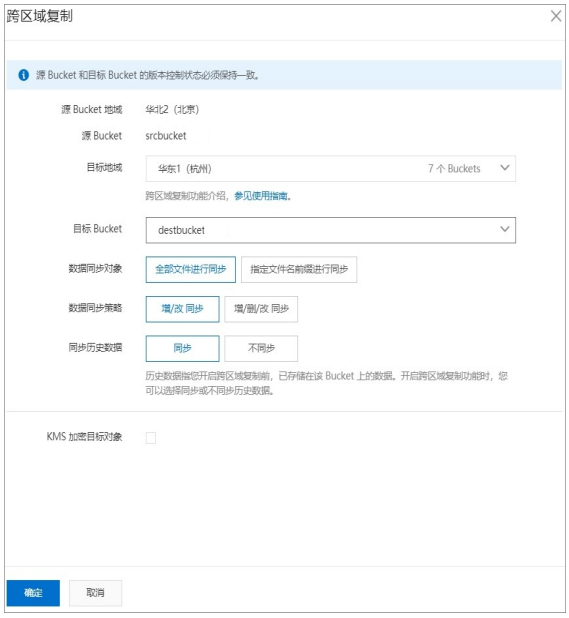
有关版本控制的使用场景及注意事项等信息，请参见版本控制介绍。

为Bucket开启跨区域复制

跨区域复制（Cross-Region Replication）是跨不同OSS数据中心（地域）的Bucket自动、异步（近实时）复制文件（Object），它会将Object的创建、更新等操作从源Bucket复制到不同区域的目标Bucket。

跨区域复制功能满足Bucket跨区域容灾或用户数据复制的需求。您可以通过该功能达到数据的跨区域容灾效果，也可以通过将数据保护策略配置为**增/改同步**来实现误删数据的保护。

如下图中的配置，当您将源存储空间srcbucket的所有数据同步到目标存储空间destbucket后，如果您误删除了srcbucket中的某个Object，此时您仍然可以从destbucket中找回误删除的Object。

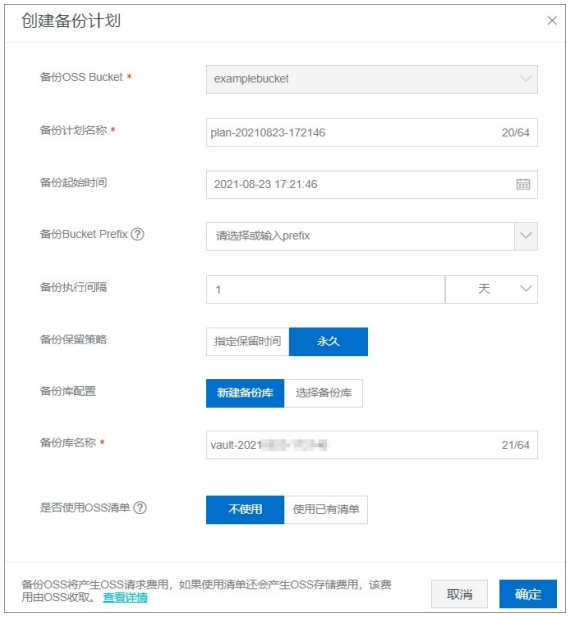


有关跨区域复制的使用场景及注意事项，请参见跨区域复制。

为Bucket设置定时备份

您可以使用OSS的定时备份功能将Bucket内的Object定期备份到混合云备份服务HBR中，当Object意外丢失或受损时，可通过HBR及时恢复。

如下图中的配置，当您为examplebucket配置定时备份后，HBR将按照指定的起始时间备份examplebucket内的所有Object，并按照设置的保留时间策略将备份版本永久保存在目标备份库中。有关创建定时备份的具体步骤，请参见定时备份。



在HBR完成一次全量备份后，如果您误删除了examplebucket内的任意Object，可通过HBR控制台创建恢复任务的方式，将examplebucket内的Object恢复至被删除前的版本。有关创建恢复任务的具体步骤，请参见创建OSS恢复任务。

9.4. 敏感数据安全防护方案

本文介绍如何将阿里云对象存储OSS与阿里云敏感数据保护SDDP（Sensitive Data Discovery & Protection）结合，对敏感数据进行识别、分类、分级和保护。

前提条件

- 已开通SDDP

  开通步骤请参见快速入门。

- 已开通OSS

  开通步骤请参见开通OSS服务。

背景信息

敏感数据主要包括个人隐私信息、密码/密钥、敏感图片等高价值数据，这些数据通常会以不同的格式存储在您的各类存储系统中。如何更好地发现、定位、保护这些数据，对您的企业非常重要。OSS本身提供了细粒度的权限管理和数据加密等数据安全选项，同城冗余存储、跨区域复制、版本控制等数据保护机制，日志转存和实时日志查询等记录监控与审计能力。若您希望更好的针对敏感数据进行识别、分类、分级和保护，可使用OSS+SDDP的方案。

SDDP与OSS结合使用，在您完成数据源识别授权后，从您的海量数据中快速发现和定位敏感数据，对敏感数据分类分级并统一展示，同时追踪敏感数据的使用情况，并根据预先定义的安全策略，对数据进行保护和审计，以便您随时了解数据资产的安全状态。更多详情请参见什么是数据安全中心。

> **说明** 当您完成OSS资产授权后，在OSS数据初次接入扫描时，SDDP会对已授权的数据源执行全量扫描并收取全量扫描费用。初次扫描任务完成后，SDDP仅对该数据源中新增或修改的文件收取扫描费用，因此计费会大幅降低。详细的计费方式请参见包年包月计费。

应用场景

OSS与SDDP结合的常见场景如下：

- 敏感数据识别

  企业拥有大量数据，但无法准确获知这些数据中是否包含敏感信息，以及敏感数据所在的位置。您可以使用OSS结合SDDP的方案，利用SDDP内置算法规则或根据行业特点自定义规则，对存储在OSS中的数据进行整体扫描、分类、分级，并根据结果做进一步的安全防护。例如利用OSS的访问控制和加密等功能，对数据进行保护。

- 数据脱敏

  数据进行对外交换供他人分析或使用时，未进行脱敏处理会导致敏感信息的意外泄漏。OSS与SDDP结合的方案，可以支持灵活多样的内置或自定义脱敏算法，可实现生产类敏感数据脱敏后，供开发、测试等非生产环境使用的场景，并确保脱敏后的数据保真可用。

- 异常检测和审计

  SDDP可通过智能化的检测模型，对访问OSS中敏感数据的行为进行检测和审计。为数据安全管理团队提供相关告警，并基于检测结果完善风险预判和规避方案。

方案优势

- 可视化
  - 提供敏感数据识别结果可视化能力，让企业数据安全状态一目了然。
  - 数据访问监控和异常审计可追溯，降低企业数据安全风险。
  - 提升整体企业数据资产安全透明度，强化企业数据管理理能力。
  - 降低数据安全运维成本，为制定企业数据安全策略提供强有力的数据支撑。
- 智能化
  - 运用大数据和机器学习能力，通过智能化算法，对敏感数据和高风险活动（例如数据异常访问和潜在的泄漏风险）进行有效识别和监控，并提供修复建议。
  - 提供定制化的敏感数据识别能力，便于客户自定义识别标准，实现精准识别和高效防护。
  - 将复杂的数据格式和内容汇总至统一的数据风险模型，并以标准化的方式呈现，实现企业关键数据资产的防御。
- 云原生
  - 生于云，长于云，充分利用云上服务优势，并支持云上多类型数据源。

- 相较于传统软件化部署方式，OSS+SDDP方案服务架构更为健壮、可用性更高、成本也更低，同时系统自身安全性也更好。

操作步骤

1. 登录数据安全中心控制台。
2. 在左侧导航栏，选择**数据保护授权 > 数据资产授权**。
3. 在**OSS**页签下，单击**未授权**。
4. 选中需要授权的OSS Bucket，并单击**批量授权**。

   您也可以单击目标Bucket右侧**授权**，为单个Bucket授权。
5. 在**对于选中资产批量处理**页面，根据您的需求配置以下参数：
   - **识别权限**：开启或关闭SDDP识别选中资产敏感数据的权限。
   - **审计权限**：开启或关闭SDDP对选中资产进行数据审计的权限。
   - **脱敏权限**：开启或关闭SDDP对选中资产进行敏感数据脱敏的权限。
   - **敏感数据采样**：设置SDDP对选中资产进行敏感数据采样的条数。可选取值：0条、5条、10条。
   - **审计日志存档**：设置选中资产的审计日志保存时间。可选取值：30天、90天、180天。

     > **说明** 设置审计日志存档时间无需您额外开通日志服务。
6. 单击**确认**。

   完成资产授权后，DSC将会对开启授权的OSS存储空间中的文件执行敏感数据检测。如果该存储空间是首次开启授权，DSC将会自动触发全量扫描并收取全量数据扫描的费用。更多信息，请参见数据源授权完成后需要多长时间完成扫描。

   已授权资产中的数据可进行编辑或取消授权。取消授权后，DSC不会检测该文件桶中的数据。

   > **说明** DSC仅对已授权的Bucket进行数据扫描和风险分析。
7. 添加安全策略。

   扫描完成后，您可以根据敏感数据扫描结果进行相应的安全加固措施。例如配置数据加密、添加访问权限等。您也可以根据需要，在SDDP控制台开启OSS安全审计功能，实现对OSS中存储的敏感文件的异常行为检测和智能安全审计。详情请参见创建审计规则和自定义泄漏检测模型。

10.IaC自动化

10.1. Terraform

10.1.1. 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

10.1.2. 使用Terraform管理OSS

本文介绍Terraform的安装和配置详情，以及如何使用Terraform来管理OSS。

安装和配置Terraform

使用Terraform前，您需要按照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，如果显示可用的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。

   i. 登录RAM控制台。

   ii. 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。

   iii. 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. 创建测试目录。

   因为每个 Terraform 项目都需要创建1个独立的工作目录，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
variable.tf                 -- 通用变量
resource.tf               -- 资源定义
data.tf                      -- 包文件定义
output.tf                  -- 输出
```

   例如创建*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"
}
```

   更多配置信息请参见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，允许在正式执行配置文件之前，查看将要执行哪些操作。

  例如，您添加了创建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。

  例如您想创建名为*figo-chen-2020*的Bucket，您需要先添加创建 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，如果*figo-chen-2020*这个Bucket不存在，则创建一个Bucket；如果已存在，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Bucket。
- **terraform import** ：如果Bucket不是通过Terraform 创建，可通过命令导入现有的Bucket。

  首先，创建名为*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配置操作示例请参见alicloud_oss_bucket。
- 更多Object相关配置操作示例请参见alicloud_oss_bucket_object。

10.2. 通过OSS和ROS部署应用

10.2.1. 通过OSS和ROS创建Nginx

资源编排ROS支持在模板中定义所需的阿里云资源（例如ECS实例、RDS数据库实例）、资源间的依赖关系等。ROS的编排引擎将根据模板自动完成所有资源的创建和配置，实现自动化部署及运维。本文介绍如何通过OSS以及ROS服务快速创建Nginx。

前提条件

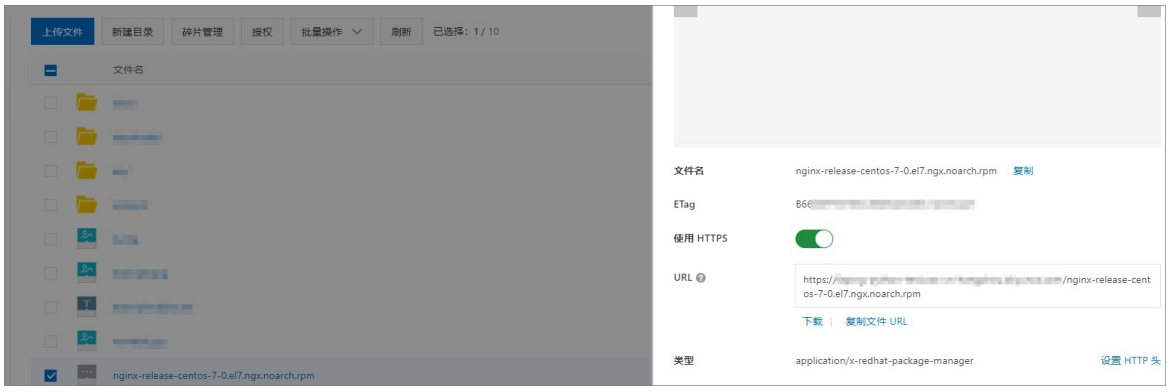
- 已创建读写权限ACL为私有的Bucket。关于创建Bucket的具体步骤，请参见创建存储空间。
- 已下载Nginx安装包。

步骤一：在Bucket中上传Nginx安装包

1. 登录OSS管理控制台。
2. 单击**Bucket列表**，然后单击目标Bucket名称。
3. 在左侧导航栏，选择**文件管理 > 文件管理**。
4. 上传已下载的Nginx安装包。

   上传时，将文件ACL设置为公共读，其他参数保留默认配置，关于上传文件的具体步骤，请参见上传文件。
5. 获取文件URL。

   文件上传完成后，单击Nginx安装包文件右侧的**详情**，然后单击**复制文件URL**。



步骤二：通过ROS新建资源栈

1. 选择模板。
   i. 登录资源编排控制台。
   ii. 在左侧导航栏，单击**资源栈**。
   iii. 在页面左上角的地域下拉列表，选择**华东1（杭州）**。
   iv. 在**资源栈列表**页面，单击**创建资源栈**，然后在下拉列表中选择**使用新资源（标准）**。
   v. 在**选择模板**页面，指定模板为**选择已有模板**，模板录入方式选择**输入模板**，在ROS模板内容

的JSON页签输入以下模板内容，然后单击下一步。

```
{
  "ROSTemplateFormatVersion": "2015-09-01",
  "Description": "",
  "Parameters": {
    "NginxDownloadUrl": {
      "Type": "String",
      "Description": {
        "zh-cn": "nginx-*.rpm的下载路径",
        "en": "The download path of nginx-*.rpm"
      },
      "Label": {
        "zh-cn": "Nginx下载地址",
        "en": "Nginx Download Url"
      }
    },
    "ZoneId": {
      "AssociationProperty": "ALIYUN::ECS::Instance:ZoneId",
      "Type": "String",
      "Description": {
        "zh-cn": "可用区ID。<br><b>注： <font color='blue'>选择前请确认该可用区是否支持创建ECS资源的规格，建议选择与其他交换机不同的可用区</font></b>",
        "en": "Availability Zone ID.<br><b>note: <font color='blue'>before selecting, please confirm that the Availability Zone supports the specification of creating ECS resources,which is recommended to be different from other VSwitch Availability Zone</font></b>"
      },
      "Label": {
        "zh-cn": "交换机可用区",
        "en": "VSwitch Availability Zone"
      }
    },
    "ImageId": {
      "Default": "centos_7",
      "Type": "String",
      "Description": {
        "zh-cn": "请使用Centos7的镜像ID。详见：<b><a href='https://help.aliyun.com/document_detail/112977.html' target='_blank'><font color='blue'>查找镜像</font></a></b>",
        "en": "Image ID, Please use Centos7, see detail: <b><a href='https://www.alibabacloud.com/help/en/doc-detail/112977.html' target='_blank'><font color='blue'>Find the mirror</font></a></b>"
      },
      "Label": {
        "zh-cn": "镜像",
        "en": "Image"
      }
    },
    "InstanceType": {
      "AssociationProperty": "ALIYUN::ECS::Instance::InstanceType",
      "AssociationPropertyMetadata": {
        "ZoneId": "ZoneId"
      },
```

```
      "Label": {
        "zh-cn": "实例规格",
        "en": "Instance Type"
      },
      "Type": "String",
      "Description": {
        "zh-cn": "<font color='blue'><b>1.选择机型前请先确认当前可用区下是否存在该机型，部分机型需要提前报备</b></font><br><font color='blue'><b>2.可选机型列表</font><br></b></font>[ecs.c5.large <font color='green'>2vCPU 4GiB 内网带宽1Gbps 内网收发包30万PPS</font>]<br></b>[ecs.c5.xlarge <font color='green'>4vCPU 8GiB 内网带宽1.5Gbps 内网收发包50万PPS</font>]<br></b>[ecs.c5.2xlarge <font color='green'>8vCPU 16GiB 内网带宽2.5Gbps 内网收发包80万PPS</font>]",
        "en": "<font color='blue'><b>1.Before selecting the model please confirm that the current available zone under the model is in stock, some models need to be reported in advance</b></font><br><font color='blue'><b>2.List of optional models</font><br></b></font>[ecs.c5.large <font color='green'>2vCPU 4GiB Intranet bandwidth1Gbps In-grid sending and receiving packages30MillionPPS</font>]<br></b>[ecs.c5.xlarge <font color='green'>4vCPU 8GiB Intranet bandwidth1.5Gbps In-grid sending and receiving packages50MillionPPS</font>]<br></b>[ecs.c5.2xlarge <font color='green'>8vCPU 16GiB Intranet bandwidth2.5Gbps In-grid sending and receiving packages80MillionPPS</font>]"
      }
    },
    "SystemDiskCategory": {
      "Default": "cloud_efficiency",
      "Label": {
        "zh-cn": "系统盘类型",
        "en": "System Disk Type"
      },
      "Type": "String",
      "Description": {
        "en": "<font color='blue'><b>Optional values:</b></font><br>[cloud_efficiency: <font color='green'>Efficient Cloud Disk</font>]<br>[cloud_ssd: <font color='green'>SSD Cloud Disk</font>]<br>[cloud_essd: <font color='green'>ESSD Cloud Disk</font>]<br>[cloud: <font color='green'>Cloud Disk</font>]<br>[ephemeral_ssd: <font color='green'>Local SSD Cloud Disk</font>]",
        "zh-cn": "<font color='blue'><b>可选值：</b></font><br>[cloud_efficiency: <font color='green'>高效云盘</font>]<br>[cloud_ssd: <font color='green'>SSD云盘</font>]<br>[cloud_essd: <font color='green'>ESSD云盘</font>]<br>[cloud: <font color='green'>普通云盘</font>]<br>[ephemeral_ssd: <font color='green'>本地SSD盘</font>]"
      },
      "AllowedValues": [
        "cloud_efficiency",
        "cloud_ssd",
        "cloud",
        "cloud_essd",
        "ephemeral_ssd"
      ]
    },
    "InstancePassword": {
      "Type": "String",
      "Description": {
        "zh-cn": "服务器登录密码。长度为8~30字符，必须包含三项（大写字母、小写字母、数字、 ( )`~!@#$%^&*_-+=|{}[]:;'<>,.?/ 中的特殊符号）。",
        "en": "Server login password. Length 8-30, must contain three(Capital lette
```

```
rs, lowercase letters, numbers, ()`~!@#$%^&*_-+=|{}[]:;'<>,.?/ Special symbol in)."
      },
      "MinLength": 8,
      "Label": {
        "zh-cn": "实例密码",
        "en": "Instance Password"
      },
      "AllowedPattern": "[0-9A-Za-z\\_\\-\\&:;'<>,=%`~!@#\\(\\)\\$\\^\\*\\+\\|\\{\\
}\\[\\]\\.\\?\\/]+$",
      "NoEcho": true,
      "MaxLength": 30,
      "ConstraintDescription": {
        "zh-cn": "长度为8~30字符，必须包含三项（大写字母、小写字母、数字、 ()`~!@#$%^&*_-+
=|{}[]:;'<>,.?/ 中的特殊符号）。",
        "en": "Length 8-30, must contain three(Capital letters, lowercase letters,
numbers, ()`~!@#$%^&*_-+=|{}[]:;'<>,.?/ Special symbol in)."
      }
    },
    "SecurityGroup": {
      "Type": "String",
      "AssociationProperty": "ALIYUN::ECS::SecurityGroup::SecurityGroupId",
      "Description": {
        "zh-cn": "现有业务安全组的实例ID，可通过ECS控制台-网络与安全-安全组进行查询。",
        "en": "Please search the business security group ID starting with(sg-xxx)fr
om console-ECS-Network & Security"
      },
      "Label": {
        "zh-cn": "安全组ID",
        "en": "Security Group ID"
      }
    },
    "VPC": {
      "AssociationProperty": "ALIYUN::ECS::VPC::VPCId",
      "Type": "String",
      "Description": {
        "zh-cn": "现有虚拟专有网络的实例ID，可通过VPC控制台-专有网络进行查询。",
        "en": "Please search the ID starting with (vpc-xxx)from console-Virtual Pri
vate Cloud"
      },
      "Label": {
        "zh-cn": "现有VPC的实例ID",
        "en": "Existing VPC Instance ID"
      }
    },
    "VSwitch": {
      "AssociationProperty": "ALIYUN::ECS::VSwitch::VSwitchId",
      "Type": "String",
      "Description": {
        "zh-cn": "现有业务网络交换机的实例ID，可通过VPC控制台-专有网络-交换机进行查询。",
        "en": "Please search the business vswitch ID starting with(vsw-xxx)from con
sole-Virtual Private Cloud-VSwitches"
      },
      "Label": {
        "zh-cn": "网络交换机ID",
```

```
        "en": "VSwitch ID"
      }
    }
  },
  "Metadata": {
    "ALIYUN::ROS::Interface": {
      "ParameterGroups": [
        {
          "Parameters": [
            "VPC",
            "VSwitch",
            "SecurityGroup"
          ],
          "Label": {
            "default": {
              "zh-cn": "基础资源配置",
              "en": "Infrastructure Configuration"
            }
          }
        },
        {
          "Parameters": [
            "ZoneId",
            "ImageId",
            "InstanceType",
            "SystemDiskCategory",
            "InstancePassword",
            "NginxDownloadUrl"
          ],
          "Label": {
            "default": {
              "zh-cn": "ECS云服务器配置",
              "en": "ECS Configuration"
            }
          }
        }
      ],
      "TemplateTags": [
        "acs:example:Linux应用服务:ROS_OOS创建Nginx应用"
      ]
    }
  },
  "Resources": {
    "WebServer": {
      "Type": "ALIYUN::ECS::Instance",
      "Properties": {
        "InternetMaxBandwidthOut": 80,
        "IoOptimized": "optimized",
        "VpcId": {
          "Ref": "VPC"
        },
        "UserData": {
          "Fn::Join": [
            "",
```

```
            [
              "#!/bin/bash \n",
              "NginxUrl=",
              {
                "Ref": "NginxDownloadUrl"
              },
              "\n",
              "yum -y install aria2 \n",
              "aria2c $NginxUrl \n",
              "rpm -ivh nginx-*.rpm \n",
              "yum -y install nginx \n",
              "systemctl enable nginx.service \n",
              "systemctl restart nginx.service \n"
            ]
          ]
        },
        "SecurityGroupId": {
          "Ref": "SecurityGroup"
        },
        "VSwitchId": {
          "Ref": "VSwitch"
        },
        "ImageId": {
          "Ref": "ImageId"
        },
        "InstanceType": {
          "Ref": "InstanceType"
        },
        "SystemDiskCategory": {
          "Ref": "SystemDiskCategory"
        },
        "Password": {
          "Ref": "InstancePassword"
        }
      }
    }
  },
  "Outputs": {
    "NginxUrl": {
      "Value": {
        "Fn::Join": [
          "",
          [
            "http://",
            {
              "Fn::GetAtt": [
                "WebServer",
                "PublicIp"
              ]
            },
            ":80"
          ]
        ]
      }
```

```
      }
    }
}
```

2. 配置模板。
   i. 在**配置模板参数**页面，填写资源栈名称，然后按以下要求完成各配置项。

区域	参数	说明
基础资源配置	**现有VPC的实例ID**	下拉选择VPC ID。
	网络交换机ID	下拉选择网络交换机ID。
	安全组ID	下拉选择安全组ID。
ECS云服务器配置	**交换机可用区**	按需选择交换机可用区，例如可用区G。
	镜像	填写centos_7。
	实例规格	按需选择实例规格。
	系统盘类型	选择cloud_ssd。
	实例密码	自定义实例密码。
	Nginx下载地址	填写Nginx安装包的文件URL。您可以从步骤一中获取文件URL。

   ii. 单击**创建**。
3. 访问Nginx欢迎页面。
   i. 单击已创建的资源栈。

ii. 单击**输出**页签，然后单击NginxUrl。

单击NginxUrl后，自动跳转至Nginx欢迎页面。

如果遇到无法访问该页面的问题，请为当前安全组ID添加默认端口 `HTTP（80）` 。具体步骤，请参见添加安全组规则。

10.2.2. 通过OSS和ROS创建Sharepoint 2013

资源编排ROS支持在模板中定义所需的阿里云资源（例如ECS实例、RDS数据库实例）、资源间的依赖关系等。ROS的编排引擎将根据模板自动完成所有资源的创建和配置，实现自动化部署及运维。本文介绍如何通过OSS以及ROS服务快速创建Sharepoint 2013。

> **说明** 本文示例由阿里云用户肖伊提供，仅供参考。

前提条件

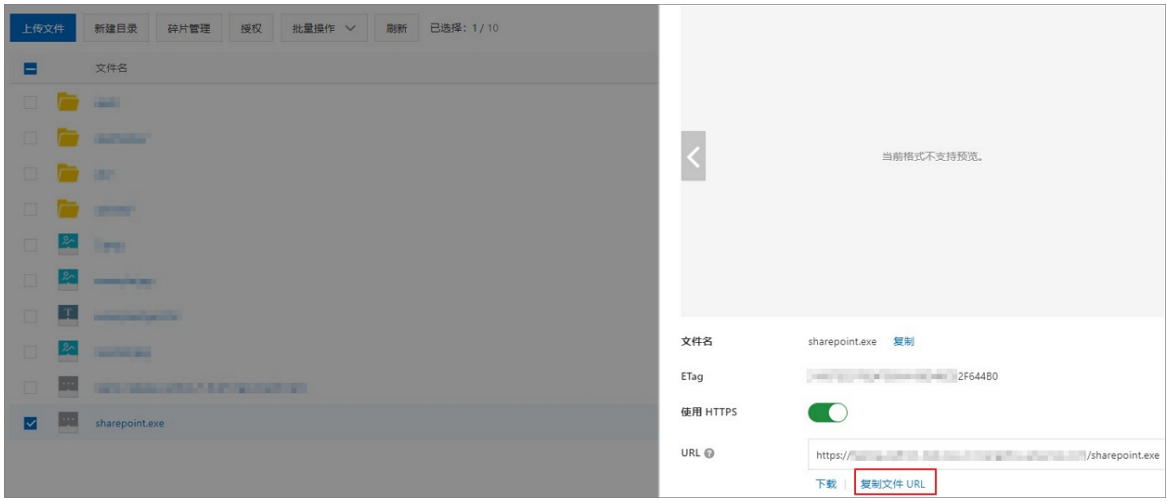
- 已创建读写权限ACL为私有的Bucket。关于创建Bucket的具体步骤，请参见创建存储空间。
- 已下载sharepoint.exe。

步骤一：在Bucket中上传sharepoint.exe文件

1. 登录OSS管理控制台。
2. 单击**Bucket列表**，然后单击目标Bucket名称。
3. 在左侧导航栏，选择**文件管理 > 文件管理**。
4. 上传已下载的sharepoint.exe文件。

   上传时，将文件ACL设置为公共读，其他参数保留默认配置，关于上传文件的具体步骤，请参见上传文件。
5. 获取文件URL。

   文件上传完成后，单击sharepoint.exe文件右侧的**详情**，然后单击**复制文件URL**。



步骤二：通过ROS新建资源栈

1. 选择模板。
   i. 登录资源编排控制台。
   ii. 在左侧导航栏，单击**资源栈**。
   iii. 在页面左上角的地域下拉列表，选择**华东1（杭州）**。
   iv. 在**资源栈列表**页面，单击**创建资源栈**，然后在下拉列表中选择**使用新资源（标准）**。
   v. 在**选择模板**页面，指定模板为**选择已有模板**、模板录入方式选择**输入模板**、在ROS模板内容的**JSON**页签输入以下内容，然后单击**下一步**。

```
{
  "ROSTemplateFormatVersion": "2015-09-01",
  "Description": "One simple ECS instance with SharePoint Foundation2013, default image is win2008r2_sp1_x64_ent_zh-cn_40G_alibase_20200116.vhd,The user needs to specify the instance type ",
  "Parameters": {
    "VpcCidrBlock": {
      "Type": "String",
      "Description": {
        "en": "The IP address range of the VPC in the CIDR Block form; <br>you can use the following IP address ranges: <br><font color='green'>[10.0.0.0/8]</font><br><font color='green'>[172.16.0.0/12]</font><br><font color='green'>[192.168.0.0/16]</font>",
        "zh-cn": "专有网络IP地址段范围，<br>您可以使用以下的IP地址段：<br><font color='green'>[10.0.0.0/8]</font><br><font color='green'>[172.16.0.0/12]</font><br><font color='green'>[192.168.0.0/16]</font>"
      },
      "AllowedValues": [
        "192.168.0.0/16",
        "172.16.0.0/12",
        "10.0.0.0/8"
      ],
      "Label": {
        "en": "VPC CIDR Block",
        "zh-cn": "专有网络网段"
      },
      "Default": "192.168.0.0/16"
```

```
    },
    "VSwitchCidrBlock": {
      "Type": "String",
      "Description": {
        "zh-cn": "创建的虚拟交换机的子网，必须是所属专有网络的子网段，并且没有被其他交换机占用。",
        "en": "Subnet of the created virtual switch, must be a sub-network segment of the proprietary network and is not occupied by other VSwitches."
      },
      "Label": {
        "zh-cn": "交换机网段",
        "en": "VSwitch CIDR Block"
      },
      "Default": "192.168.0.0/16"
    },
    "ZoneId": {
      "Type": "String",
      "AssociationProperty": "ALIYUN::ECS::Instance:ZoneId",
      "Description": {
        "en": "Availability zone ID,<br><b>note: <font color='blue'>Before selecting, please confirm that the Availability Zone supports the specification of creating ECS resources</font></b>",
        "zh-cn": "可用区ID，<br><b>注：<font color='blue'>选择可用区前请确认该可用区是否支持创建ECS资源的规格</font></b>"
      },
      "Label": {
        "zh-cn": "交换机可用区",
        "en": "VSwitch Available Zone"
      }
    },
    "SharePointDownloadPath": {
      "Label": {
        "zh-cn": "SharePoint镜像下载路径",
        "en": "SharePoint Image Download Path"
      },
      "Type": "String",
      "Description": {
        "zh-cn": "SharePoint Foundation2013镜像下载路径，详见： <a href='https://docs.microsoft.com/zh-cn/OfficeUpdates/sharepoint-updates' target='_blank'><b><font color='blue'>SharePoint Foundation2013镜像下载路径</b></font></a>",
        "en": "SharePoint Foundation2013 image download path, see detail: <a href='https://docs.microsoft.com/en-us/OfficeUpdates/sharepoint-updates' target='_blank'><b><font color='blue'>SharePoint Foundation2013 image download path</b></font></a>"
      }
    },
    "ImageId": {
      "Default": "win2008r2_sp1_x64_ent_zh-cn_40G_alibase_20200116.vhd",
      "Label": {
        "zh-cn": "镜像",
        "en": "Image ID"
      },
      "Type": "String",
      "Description": {
        "zh-cn": "镜像ID，关于SharePoint Foundation2013系统要求，详见： <a href='https
```

```
://docs.microsoft.com/zh-cn/sharepoint/install/hardware-and-software-requirements-0
' target='_blank'><b><font color='blue'>SharePoint Foundation2013系统要求</b></font>
</a>",
        "en": "Image ID, About SharePoint Foundation2013 system requirements, see d
etail: <a href='https://docs.microsoft.com/en-us/sharepoint/install/hardware-and-so
ftware-requirements-0' target='_blank'><b><font color='blue'>SharePoint Foundation2
013 system requirements</b></font></a>"
      }
    },
    "InstanceType": {
      "Type": "String",
      "Description": {
        "zh-cn": "填写VSwitch可用区下可使用的规格；<br>通用规格：<font color='red'><b>ec
s.c5.large</b></font><br>注：可用区可能不支持通用规格<br>规格详见：<a href='https://help
.aliyun.com/document_detail/25378.html' target='_blank'><b><font color='blue'>实例规
格族</font></a></b>",
        "en": "Fill in the specifications that can be used under the VSwitch availa
bility zone;</b></font><br>general specifications：<font color='red'><b>ecs.c5.larg
e</b></font><br>note: a few zones do not support general specifications<br>see deta
il: <a href='https://www.alibabacloud.com/help/en/doc-detail/25378.html' target='_b
lank'><b><font color='blue'>Instance Specification Family</font></a></b>"
      },
      "Label": {
        "zh-cn": "实例规格",
        "en": "Instance Type"
      },
      "AssociationProperty": "ALIYUN::ECS::Instance::InstanceType",
      "AssociationPropertyMetadata": {
        "ZoneId": "ZoneId"
      }
    },
    "Password": {
      "NoEcho": true,
      "Type": "String",
      "Description": {
        "en": "Server login password, Length 8-30, must contain three(Capital lette
rs, lowercase letters, numbers, ()`~!@#$%^&*_-+=|{}[]:;'<>,.?/ Special symbol in)."
,
        "zh-cn": "服务器登录密码，长度为8~30个字符，必须包含三项（大写字母、小写字母、数字、
()`~!@#$%^&*_-+=|{}[]:;'<>,.?/ 中的特殊符号）。"
      },
      "AllowedPattern": "[0-9A-Za-z\\_\\-\\&:;'<>,=%`~!@#\\(\\)\\$\\^\\*\\+\\|\\{\\
}\\[\\]\\.\\?\\/]+$",
      "Label": {
        "zh-cn": "实例密码",
        "en": "Instance Password"
      },
      "ConstraintDescription": {
        "en": "Length 8-30, must contain three(Capital letters, lowercase letters,
numbers, ()`~!@#$%^&*_-+=|{}[]:;'<>,.?/ Special symbol in).",
        "zh-cn": "长度为8~30个字符，必须包含三项（大写字母、小写字母、数字、 ()`~!@#$%^&*_
-+=|{}[]:;'<>,.?/ 中的特殊符号）。"
      },
      "MinLength": 8,
```

```
      "MaxLength": 30
    }
  },
  "Metadata": {
    "ALIYUN::ROS::Interface": {
      "ParameterGroups": [
        {
          "Parameters": [
            "VpcCidrBlock",
            "VSwitchCidrBlock"
          ],
          "Label": {
            "default": "VPC"
          }
        },
        {
          "Parameters": [
            "ZoneId",
            "ImageId",
            "SharePointDownloadPath",
            "InstanceType",
            "Password"
          ],
          "Label": {
            "default": "ECS"
          }
        }
      ],
      "TemplateTags": [
        "acs:example:Windows应用服务:ROS_OOS创建SharePoint2013的windows实例"
      ]
    }
  },
  "Resources": {
    "RosWaitCondition": {
      "Type": "ALIYUN::ROS::WaitCondition",
      "Properties": {
        "Timeout": 2700,
        "Count": 2,
        "Handle": {
          "Ref": "RosWaitConditionHandle"
        }
      },
      "Metadata": {
        "ALIYUN::ROS::Designer": {
          "id": "47058b03-d345-4071-bcdb-9bc2888899be"
        }
      }
    },
    "EcsVswitch": {
      "Type": "ALIYUN::ECS::VSwitch",
      "Properties": {
        "VpcId": {
          "Ref": "EcsVpc"
        }
```

```
        },
        "ZoneId": {
          "Ref": "ZoneId"
        },
        "CidrBlock": {
          "Ref": "VSwitchCidrBlock"
        }
      },
      "Metadata": {
        "ALIYUN::ROS::Designer": {
          "id": "3f1d6b20-25eb-4a1c-a40f-1f764b157bd6"
        }
      }
    },
    "SharePointServer": {
      "Type": "ALIYUN::ECS::Instance",
      "Properties": {
        "VpcId": {
          "Ref": "EcsVpc"
        },
        "UserData": {
          "Fn::Replace": [
            {
              "ros-notify": {
                "Fn::GetAtt": [
                  "RosWaitConditionHandle",
                  "PowerShellCurlCli"
                ]
              }
            },
            {
              "Fn::Join": [
                "",
                [
                  "[powershell]\r\n",
                  "New-Item -Path \"C:\\Install_dir\" -Force -type directory \r\n",
                  "$ossName = 'ros-template-resources' \r\n",
                  "$client = new-object System.Net.WebClient \r\n",
                  "$client.DownloadFile(\"",
                  {
                    "Ref": "SharePointDownloadPath"
                  },
                  "\", 'C:\\Install_dir\\SharePoint.exe') \r\n",
                  "$client.DownloadFile(\"https://$ossName.oss-cn-beijing.aliyuncs.
com/SharePoint/AppFabric1.1-RTM-KB2671763-x64-ENU.exe\", 'C:\\Install_dir\\AppFabri
c1.1-RTM-KB2671763-x64-ENU.exe')\r\n",
                  "$client.DownloadFile(\"https://$ossName.oss-cn-beijing.aliyuncs.
com/SharePoint/MicrosoftIdentityExtensions-64.msi\", 'C:\\Install_dir\\MicrosoftIde
ntityExtensions-64.msi')\r\n",
                  "$client.DownloadFile(\"https://$ossName.oss-cn-beijing.aliyuncs.
com/SharePoint/setup_msipc_x64.msi\", 'C:\\Install_dir\\setup_msipc_x64.msi')\r\n",
                  "$client.DownloadFile(\"https://$ossName.oss-cn-beijing.aliyuncs.
com/SharePoint/sqlncli.msi\", 'C:\\Install_dir\\sqlncli.msi')\r\n",
                  "$client.DownloadFile(\"https://$ossName.oss-cn-beijing.aliyuncs.
com/SharePoint/Synchronization.msi\", 'C:\\Install_dir\\Synchronization.msi')\r\n",
```

```
                "$client.DownloadFile(\"https://$ossName.oss-cn-beijing.aliyuncs.
com/SharePoint/WcfDataServices.exe\", 'C:\\Install_dir\\WcfDataServices.exe')\r\n",
                "$client.DownloadFile(\"https://$ossName.oss-cn-beijing.aliyuncs.
com/SharePoint/Windows6.1-KB974405-x64.msu\", 'C:\\Install_dir\\Windows6.1-KB974405
-x64.msu')\r\n",
                "$client.DownloadFile(\"https://$ossName.oss-cn-beijing.aliyuncs.
com/SharePoint/WindowsServerAppFabricSetup_x64.exe\", 'C:\\Install_dir\\WindowsServ
erAppFabricSetup_x64.exe')\r\n",
                "cd C:\\Install_dir \r\n",
                "New-Item \"install_sharepoint.bat\" -type File \r\n",
                "Add-Content install_sharepoint.bat \"C:\" \r\n",
                "Add-Content install_sharepoint.bat 'powershell.exe C:\\Install_d
ir\\Install_SharePoint.ps1' \r\n",
                "New-Item \"Install_SharePoint.ps1\" -type File \r\n",
                "Add-Content Install_SharePoint.ps1 'if(Test-Path \"C:\\Install_d
ir\\finished.log\"){' \r\n",
                "Add-Content Install_SharePoint.ps1 \"cd C:\\Install_dir \" \r\n"
,
                "Add-Content Install_SharePoint.ps1 \"}else{ \" \r\n",
                "Add-Content Install_SharePoint.ps1 'Start-Process C:\\Install_di
r\\2013\\Setup.exe -args \"/config C:\\Install_dir\\2013\\Files\\SetupSilent\\confi
g.xml\" -Wait'\r\n",
                "Add-Content Install_SharePoint.ps1 'cd C:\\Install_dir'\r\n",
                "Add-Content Install_SharePoint.ps1 'New-Item \"finish.log\" -typ
e File'\r\n",
                "Add-Content Install_SharePoint.ps1 'Remove-Item \"C:\\Install_di
r\\SharePoint.exe\"' \r\n",
                "Add-Content Install_SharePoint.ps1 'Remove-Item \"C:\\Install_di
r\\2013\\*\" -recurse' \r\n",
                "Add-Content Install_SharePoint.ps1 'schtasks /delete /TN \"insta
ll_sharepoint\" /f'\r\n",
                "Add-Content Install_SharePoint.ps1 'ros-notify'\r\n",
                "Add-Content Install_SharePoint.ps1 \"}\" \r\n",
                "Start-Process C:\\Install_dir\\SharePoint.exe  -args \"/extract:
C:\\Install_dir\\2013 /quiet\" -Wait \r\n",
                "Start-Process C:\\Install_dir\\2013\\prerequisiteinstaller.exe -
args \"/continue /unattended\" -Wait \r\n",
                "Start-Process C:\\Install_dir\\2013\\prerequisiteinstaller.exe
-args \"/SQLNCli:C:\\Install_dir\\sqlncli.msi /continue /unattended\" -Wait\r\n",
                "Start-Process C:\\Install_dir\\2013\\prerequisiteinstaller.exe
-args \"/IDFX:C:\\Install_dir\\Windows6.1-KB974405-x64.msu /continue /unattended\"
-Wait\r\n",
                "Start-Process C:\\Install_dir\\2013\\prerequisiteinstaller.exe
-args \"/Sync:C:\\Install_dir\\Synchronization.msi /continue /unattended\" -Wait\r\
n",
                "Start-Process C:\\Install_dir\\2013\\prerequisiteinstaller.exe -
args \"/AppFabric:C:\\Install_dir\\WindowsServerAppFabricSetup_x64.exe /continue /u
nattended\" -Wait \r\n",
                "Start-Process C:\\Install_dir\\MicrosoftIdentityExtensions-64.ms
i -args \"/quiet\" -Wait \r\n",
                "Start-Process C:\\Install_dir\\2013\\prerequisiteinstaller.exe -
args \"/continue /unattended\" -Wait \r\n",
                "Start-Process C:\\Install_dir\\2013\\prerequisiteinstaller.exe -
args \"/WCFDataServices:C:\\Install_dir\\WcfDataServices.exe /continue /unattended\
```

```
" -Wait \r\n",
                  "Start-Process C:\\Install_dir\\2013\\prerequisiteinstaller.exe -
args \"/KB2671763:C:\\Install_dir\\AppFabric1.1-RTM-KB2671763-x64-ENU.exe /continue
/unattended\" -Wait\r\n",
                  "schtasks /create /TN \"install_sharepoint\" /RU administrator /R
P \"",
                  {
                    "Ref": "Password"
                  },
                  "\" /SC ONSTART /TR \"C:\\Install_dir\\install_sharepoint.bat\" \
r\n",
                  "ros-notify \r\n",
                  "shutdown /r /f /t 1 \r\n"
                ]
              ]
            }
          ]
        },
        "SecurityGroupId": {
          "Ref": "EcsSecurityGroup"
        },
        "VSwitchId": {
          "Ref": "EcsVswitch"
        },
        "ImageId": {
          "Ref": "ImageId"
        },
        "InstanceType": {
          "Ref": "InstanceType"
        },
        "Password": {
          "Ref": "Password"
        }
      },
      "Metadata": {
        "ALIYUN::ROS::Designer": {
          "id": "62ed3480-152b-40d3-946d-4d19c0de7530"
        }
      }
    },
    "EcsVpc": {
      "Type": "ALIYUN::ECS::VPC",
      "Properties": {
        "CidrBlock": {
          "Ref": "VpcCidrBlock"
        },
        "VpcName": "Vpc-sharepoint"
      },
      "Metadata": {
        "ALIYUN::ROS::Designer": {
          "id": "4dfdcadf-d55e-4085-af95-79d40005c3da"
        }
      }
    },
```

```
    "EcsSecurityGroup": {
      "Type": "ALIYUN::ECS::SecurityGroup",
      "Properties": {
        "SecurityGroupIngress": [
          {
            "Priority": 1,
            "PortRange": "3389/3389",
            "NicType": "intranet",
            "SourceCidrIp": "0.0.0.0/0",
            "IpProtocol": "tcp"
          }
        ],
        "VpcId": {
          "Ref": "EcsVpc"
        }
      },
      "Metadata": {
        "ALIYUN::ROS::Designer": {
          "id": "f1c13fc6-e74d-477c-be14-9e24fd583b55"
        }
      }
    },
    "RosWaitConditionHandle": {
      "Type": "ALIYUN::ROS::WaitConditionHandle",
      "Metadata": {
        "ALIYUN::ROS::Designer": {
          "id": "d3d24118-a044-4f1d-9684-758480e50a31"
        }
      }
    }
  },
  "Outputs": {
    "InstanceId": {
      "Value": {
        "Fn::GetAtt": [
          "SharePointServer",
          "InstanceId"
        ]
      }
    },
    "PublicIp": {
      "Value": {
        "Fn::GetAtt": [
          "SharePointServer",
          "PublicIp"
        ]
      }
    },
    "SecurityGroupId": {
      "Value": {
        "Fn::GetAtt": [
          "EcsSecurityGroup",
          "SecurityGroupId"
        ]
```

```
        }
      },
      "VpcName": {
        "Value": {
          "Fn::GetAtt": [
            "EcsVpc",
            "VpcId"
          ]
        }
      }
    }
}
```

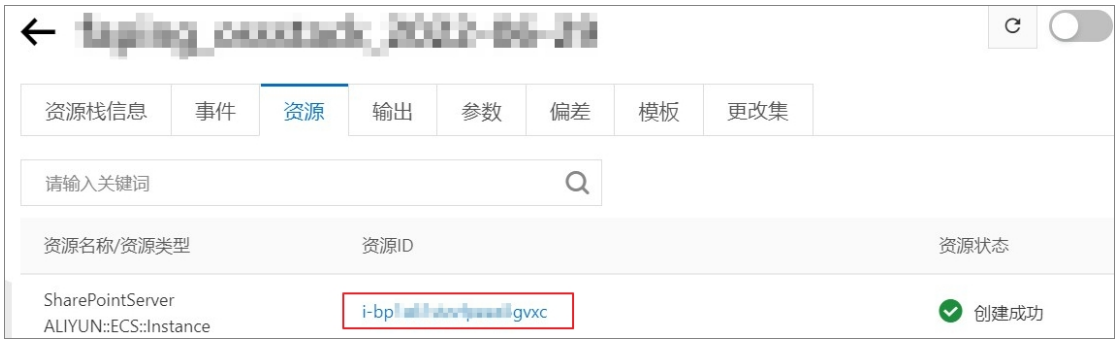
2. 配置模板。
   i. 在**配置模板参数**页面，填写资源栈名称，然后按以下要求完成各配置项。

区域	参数	说明
VPC	**专有网络网段**	选中192.168.0.0/16。
	交换机网段	填写192.168.0.0/24。
ECS	**交换机可用区**	按需选择交换机可用区，例如可用区G。
	镜像	创建ECS使用的镜像ID。
	SharePoint镜像下载路径	填写sharepoint.exe的文件URL。您可以从步骤一种获取文件URL。
	实例规格	按需选择实例规格，例如ecs.c5.large。
	实例密码	自定义ECS实例登录密码。

   ii. 单击**创建**。

   创建资源栈大约需要40分钟。

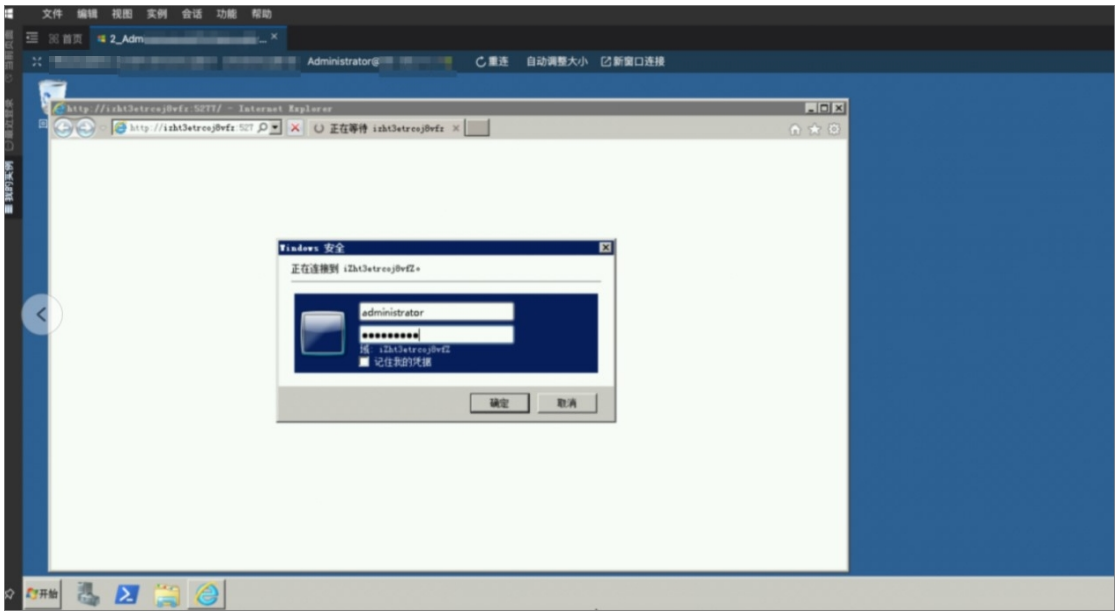
3. 登录ECS实例。
   i. 单击已创建的资源栈。
   ii. 单击**资源**页签，单击SharePointServerALIYUN::ECS::Instance资源ID。



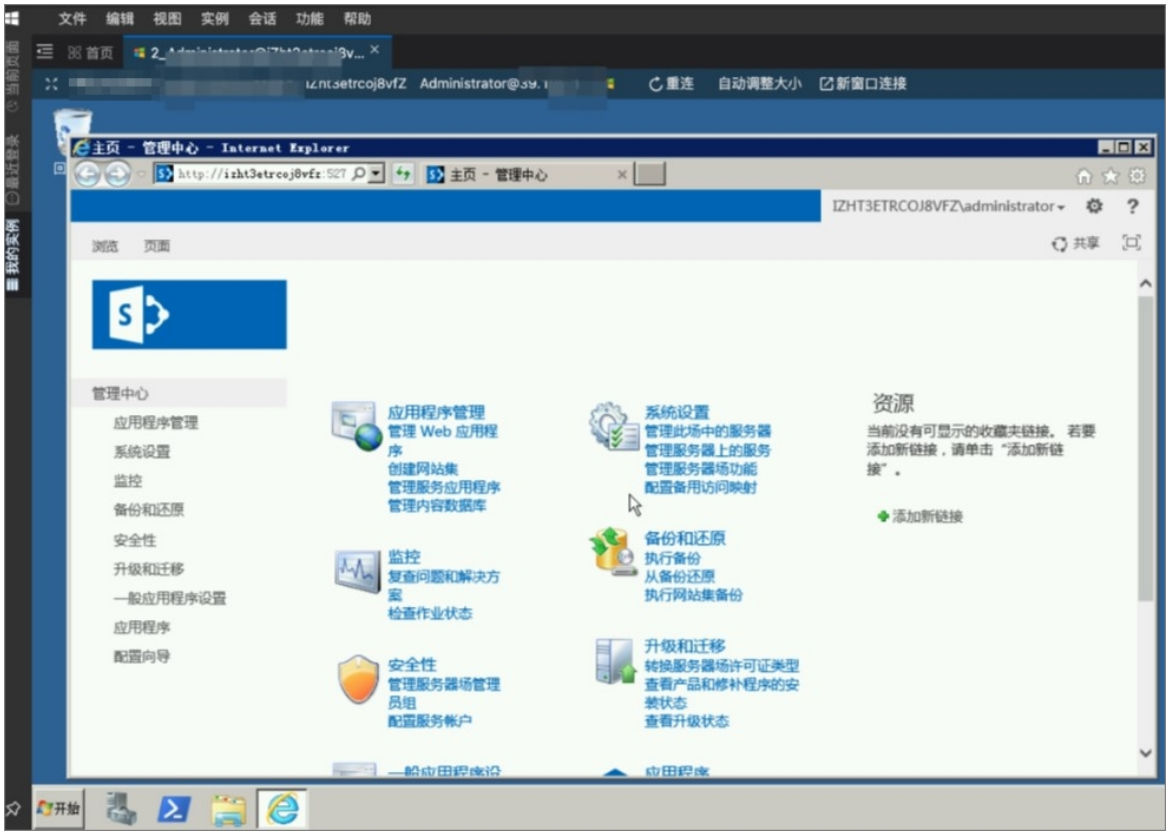
   iii. 在ECS管理控制台，单击该实例右侧的**远程连接**，然后单击Workbench远程连接下的**立即登录**。

4. 体验SharePoint。

i. 打开SharePoint 2013管理中心，输入默认用户名administrator与ECS实例登录密码。



5. 在SharePoint管理站点，体验SharePoint多种功能。



11.其他

11.1. 基于Jenkins+OOS+OSS自动化构建和部署WordPress

您可以将OSS与Jenkins和OOS结合使用，实现自动化构建和部署WordPress博客系统。

Jenkins是一款开源的基于Java的持续集成工具。在该示例中，Jenkins用于自动化构建并触发更新OSS和OOS的软件包；OOS是阿里云提供的运维编排服务，用于管理软件版本和分批发布；OSS用于存储构建的软件包。

该示例的详细部署介绍和最佳实践，请参见基于Jenkins+OOS+OSS的自动化部署。

11.2. 使用函数计算打包下载OSS文件

本文介绍如何使用函数计算将对象存储OSS上多个文件（Object）打包下载到本地。

前提条件

- 已开通函数计算

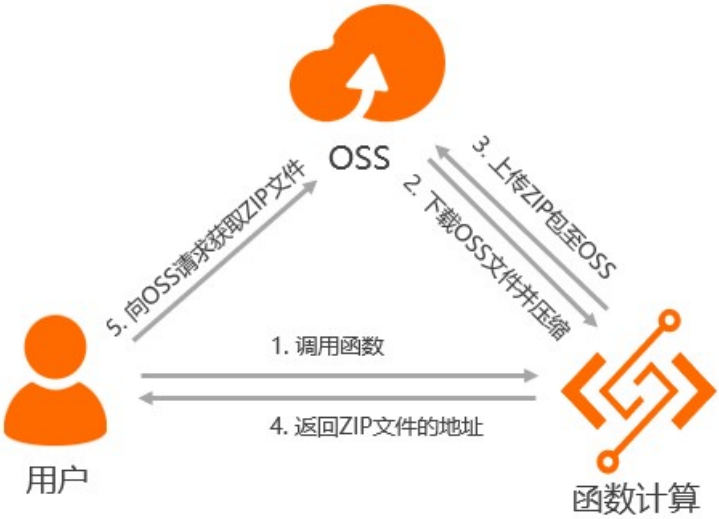
  您可以在产品详情页开通产品。更多信息，请参见函数计算。

- 已授权函数计算访问OSS

  授权操作请参见通过阿里云账号给RAM用户授权。

背景信息

当您从OSS中批量下载Object时可能会遇到批量下载不方便、小文件较多时下载缓慢等问题。通过调用函数计算，可以将OSS上的Object先打包，之后将压缩包下载到本地后再解压，实现快速下载批量文件的目的。使用函数计算打包下载OSS文件的流程如下图所示。



详细流程：

1. 用户调用函数，指定存储空间及待压缩的文件。
2. 函数计算从OSS中获取指定文件，并生成一个随机名称的ZIP压缩包。

3. 函数计算将压缩包上传至OSS。
4. 函数计算将ZIP包的下载地址返回给用户。
5. 用户使用返回的下载地址从OSS中下载文件。

> **说明**
>
> - 函数运行环境的磁盘空间是有限的，所以采用流式下载和上传的方式，只在内存中缓存少量的数据。
> - 为了加快速度，函数计算会一边生成ZIP文件，一边上传到OSS。
> - 上传ZIP文件到OSS时，利用OSS分片上传的特性，多线程并发上传。
> - 使用函数计算压缩文件时，最大处理时间是10分钟（实验数据：57个文件，总大小1.06 GB，处理时间为63s）。

操作步骤

以下步骤均以Linux系统为例，Windows和macOS系统请修改相应命令。

1. 安装funcraft。

   关于安装方法，请参见安装funcraft。

2. 下载函数代码。

   ```
   wget https://codeload.github.com/awesome-fc/zip-oss/zip/master
   ```

3. 解压已下载的函数代码。

   ```
   unzip master.zip
   ```

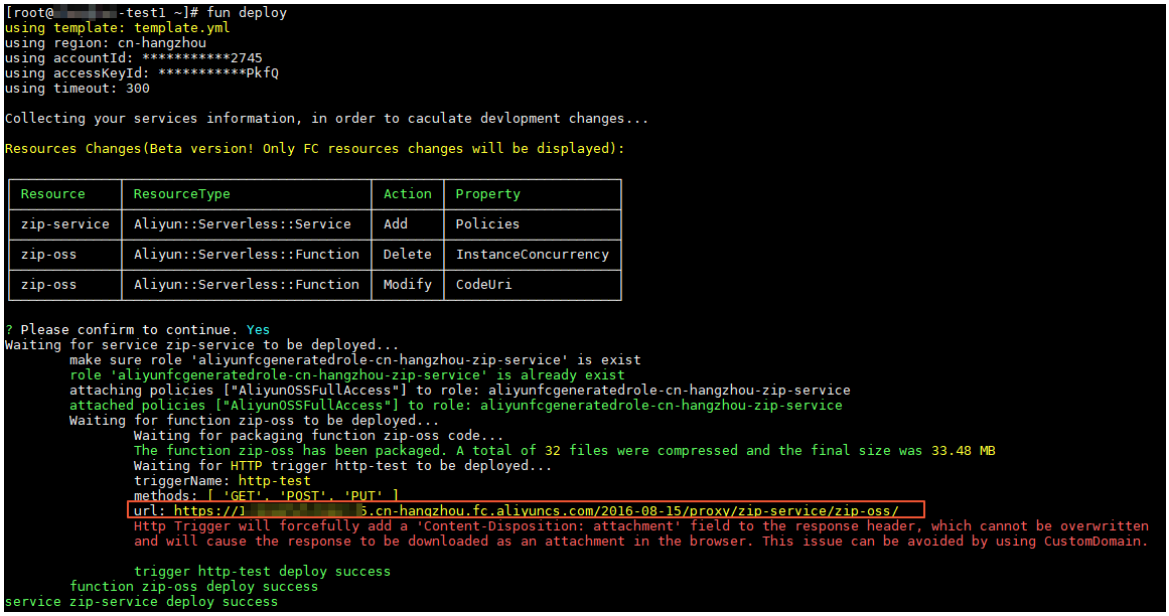
4. 修改*event.json*文件，填写需要压缩的文件所在位置。

   ```
   vi event.json
   {
      "region": "regionid",
      "bucket": "bucketname",
      "source-dir": "path/"
    }
   ```

   各参数说明如下：

   - region：填写Bucket所在地域的regionid，例如杭州填写 `cn-hangzhou` 。
   - bucket：填写Bucket名称。
   - source-dir：填写需要解压的文件目录，例如根目录填写 `./` 。建议将需要压缩的文件统一放在一个文件目录下。

5. 输入**fun deploy**命令部署函数，并记录URL值。



6. 使用curl命令触发函数。

   命令格式如下：

   ```
   curl -v -L -o /localpath -d @./event.json urlvalue
   ```

   本文示例中输入的命令为：

   ```
   curl -v -L -o /test/oss.zip -d @./event.json https://174649585760****.cn-hangzhou.fc.aliyuncs.com/2016-08-15/proxy/zip-service/zip-oss/
   ```

11.3. 使用日志服务告警为您的OSS保驾护航

日志服务SLS告警作为一站式运维告警平台，为OSS的访问提供了定制化的告警规则。您只需要在日志服务控制台进行简单配置，即可完成对OSS访问指标的监控，并在指标出现异常时及时收到告警通知。

场景描述

客户A是一家多媒体公司，主要产品有短视频App。用户可以在上面发布和观看短视频。客户A使用了OSS中名为examplebucket的存储空间来存储用户产生的短视频，App每天的用户活动具有一定的周期性，一般情况下深夜的PV较低，其他时段PV较高。

每天9点~10点之间的PV访问基本持平。例如每月5号9点~10点的PV是50万，6号9点~10点的PV是51万可以理解为正常波动。如果6号9~10点的PV出现陡增至100万或者陡降至20万的情况，您可以通过创建日志服务告警规则，指定在某个时间段内PV陡增或者陡降20%时发出告警，并通过钉钉机器人推送告警信息。

前提条件

已为目标存储空间examplebucket开启实时日志查询并记录生成的Project名称。

> **说明** 开启实时日志查询后，日志服务将自动生成名为 `oss-log-store` 的Logstore，该Logstore保存在格式为 `oss-log-阿里云账号ID-region` 的Project下。请记录生成的Project名称，用于配置后续的通知渠道及告警阈值。开启实时日志查询的具体步骤，请参见查询实时日志。

操作步骤

1. 通过SLS OSS内置行动策略设置告警触发后的通知渠道。
   i. 登录日志服务控制台。
   ii. 在Project列表区域，单击目标Project。
   iii. 在左侧导航栏中，单击 图标。
   iv. 在**告警中心**页面，选择**告警管理 > 行动策略**。
   v. 在**行动策略**页面，单击策略名**SLS OSS内置行动策略**右侧的**修改**。
   vi. 在**第一行动列表**页签，将**请求地址**修改为钉钉群的机器人WebHook地址。其他选项，保持默认配置。

   有关如何获取WebHook地址的具体操作，请参见钉钉-自定义。同时，日志服务支持多种通知渠道，例如通过短信、语音、邮件等形式推送给指定用户或用户组。详情请参见通知渠道说明。
   vii. 单击**确认**。
2. 设置触发告警阈值。
   i. 在Project列表区域，单击目标Project。
   ii. 在左侧导航栏中，单击 图标。
   iii. 在**告警中心**页面下的**规则/事物**页签，单击监控规则**OSS访问PV同比昨日变化率过高告警**右侧的**添加**。
   iv. 在**参数设置**页面，设置以下参数，其他参数保留默认值。

配置项	说明
增长率阈值	20
下降率阈值	20
阿里云账号ID	填写通配符星号（*），表示监控审计服务下配置的所有阿里云账号。
Bucket名称	填写目标存储空间*examplebucket*。

   v. 单击**设置并开启**。
3. 接收告警通知。

   当满足以上触发条件时，将收到如下格式的告警通知。

常见问题

问题描述：告警渠道选择钉钉机器人，但是钉钉通知发送失败，且出现如下错误。

```
{"errcode":310000,"errmsg":"sign not match"}
{"errcode":310000,"errmsg":"keywords not in content"}
```

问题原因：机器人的安全设置有误，导致通知被钉钉拦截。

解决方法：将安全设置配置为自定义关键字，其中一个关键字设置为“告警”，因为通知内容中至少包含其中1个关键字才可以发送成功。配置详情，请参见钉钉-自定义。

11.4. 通过云监控服务实时监控OSS流控信息

当用户的请求量超出OSS使用限制后会触发OSS流控，触发流控会对用户的请求产生一定的影响。您只需要在云监控管理控制台进行简单的配置，即可完成对OSS请求指标的实时监控，并在触发流控时及时收到告警通知。

背景信息

OSS提供了用户级别和Bucket级别的流控，支持的类别主要包括带宽流控和QPS流控。当您访问OSS的QPS、带宽超出OSS使用限制时，访问速度会受到OSS流控的限制。如果触发了带宽流控，则访问OSS的延迟会增加。如果触发了QPS流控，则OSS会丢弃部分请求。关于带宽流控和QPS流控的限制信息，请参见使用限制。

您可以通过云监控管理控制台创建OSS流控事件告警规则，并指定在监测到用户发送到OSS指定类型的请求量触发流控或达到汇报阈值时，以短信、邮件和钉钉机器人的方式向指定联系人组发送报警信息。

前提条件

已创建用于接收流控报警信息的联系人组，并向联系人组添加多个联系人。具体操作，请参见创建报警联系人或报警联系人组。

创建报警规则

1. 登录云监控控制台。
2. 在左侧导航栏，选择**事件监控 > 系统事件**。
3. 单击**事件报警规则**页签。
4. 单击**创建报警规则**。
5. 在**创建/修改事件报警**面板，设置以下参数，其他参数保留默认值，然后单击**完成**。

参数	说明
报警规则名称	设置为*rule1*。
产品类型	选择**对象存储OSS**。
事件类型	选择**全部事件**。
事件等级	选择**警告**和**信息**。
事件名称	选择**全部事件**。关于云监控支持的OSS流控事件的含义及说明，请参见云监控支持的OSS流控事件。
联系人组	选中报警方式下的**报警通知**，然后选择已创建的联系人组。
通知方式	选择**Warning（短信+邮件+钉钉机器人）**。

以上事件告警规则配置完成后，如果请求触发OSS流控或者超过汇报阈值，则云监控会自动向指定的联系人发送报警通知。报警通知中包含报警资源、事件名称、事件类别以及事件详情等信息。关于报警通知的更多信息，请参见报警通知。

> **注意** 流控报警为每分钟一次，一分钟内如果有30s或以上时间触发流控则产生报警。汇报阈值为每10分钟一次，只要1s内触发汇报阈值则产生报警。

报警通知

如果指定联系人收到了流控触发报警通知，请参见以下表格了解各类流控事件触发的原因、影响、对应的解决方法以及事件的详细内容。

> **注意** 如果您希望在收到User级别的报警事件后，查看归属当前用户下所有Bucket的流量使用情况，请提前创建OSS监控大盘。具体步骤，请参见创建系统预置大盘。

报警通知事件名称

> **说明** 下表中的汇报阈值=流控阈值*0.8。

事件名称	触发原因	影响	解决方法
BucketIngressBandwidthThresholdExceeded	当前Bucket的上行带宽之和大于Bucket的上行流控阈值时触发此事件。	上传请求将会被流控且请求延迟会增加。	合理降低上传请求并发数。
BucketEgressBandwidthThresholdExceeded	当前Bucket的下行带宽之和大于Bucket下行带宽流控阈值时触发此事件。	下载请求将会被流控且请求延迟会增加。	合理降低下载请求并发数。
BucketQpsThresholdExceeded	当前Bucket的每秒请求数之和大于Bucket每秒请求数流控阈值时触发此事件。	OSS会拒绝响应部分请求并返回503。	合理降低每秒请求数。
UserIngressBandwidthThresholdExceeded	归属当前用户的所有Bucket的上行带宽之和大于上行带宽流控阈值时触发此事件。	上传请求将会被流控且请求延迟会增加。	合理降低上传请求并发数。
UserEgressBandwidthThresholdExceeded	归属当前用户的所有Bucket的下行带宽之和大于下行带宽流控阈值时触发此事件。	下载请求将会被流控且请求延迟会增加。	合理降低下载请求并发数。
UserQpsThresholdExceeded	归属当前用户的所有Bucket的每秒请求数之和大于每秒请求数流控阈值时触发此事件。	OSS会拒绝响应部分请求。	合理降低每秒请求数。
BucketImageCpuThresholdExceeded	当前Bucket中所有用于处理图片请求的CPU核数之和大于Bucket CPU核数流控阈值时触发此事件。	图片处理类型的请求延迟会增加。	合理降低图片处理请求并发数。
UserImageCpuThresholdExceeded	归属当前用户的所有Bucket中用于处理图片请求的CPU核数之和大于该用户的CPU核数流控阈值时触发此事件。	图片处理类型的请求延迟会增加。	合理降低图片处理请求并发数。
BucketMirrorIngressBandwidthThresholdExceeded	当前Bucket的所有镜像回源类型请求的带宽之和大于流控阈值时触发此事件。	镜像回源请求延迟会增加。	合理降低镜像回源类型请求并发数。
BucketMirrorQpsThresholdExceeded	当前Bucket的所有镜像回源类型的每秒请求数之和大于镜像回源QPS流控阈值时触发此事件。	OSS会拒绝部分镜像回源类型请求。	合理降低镜像回源类型每秒请求数。

事件名称	触发原因	影响	解决方法
UserMirrorIngressBandwidthThresholdExceeded	归属当前用户的所有Bucket内的镜像回源类型上传请求流量之和大于用户镜像回源带宽流控阈值时触发此事件。	镜像回源类的上传请求延迟会增加。	合理降低镜像回源类型请求并发数。
UserMirrorQpsThresholdExceeded	归属当前用户的所有Bucket每秒发出的镜像回源类型请求数之和大于用户镜像回源QPS流控阈值时触发此事件。	OSS将拒绝响应部分镜像回源类型的请求。	合理降低镜像回源类型每秒请求数。
BucketIngressBandwidth	当前Bucket的上行请求带宽之和大于汇报阈值时触发此事件。	Bucket的上行请求延迟会增加。	合理降低上行请求并发数。
BucketEgressBandwidth	当前Bucket的下行请求带宽之和大于汇报阈值时触发此事件。	Bucket的下行请求延迟会增加。	合理降低下行请求并发数。
UserIngressBandwidth	归属当前用户的所有Bucket的上行请求带宽之和大于汇报阈值时触发此事件。	用户的上行请求延迟会增加。	合理降低上行请求并发数。
UserEgressBandwidth	归属当前用户的所有Bucket的下行请求带宽之和大于汇报阈值时触发此事件。	用户的下行请求延迟会增加。	合理降低下行请求并发数。

报警通知详细内容

参数	说明	示例值
AvgSeverity	流控的程度。数值越高代表流控越强，延时越高。取值范围为0~100。	10
QosType	触发的流控类型。取值如下： • *IngressBandwidth*：上行带宽流量。 • *EgressBandWidth*：下行带宽流量。 • *Qps*：每秒请求数。	*IngressBandwidth*
TrafficSource	触发流控的流量来源。取值如下： • *intranet*：内网带宽。 • *extranet*：外网带宽。 • *net_all*：总带宽（同时包含内网带宽和外网带宽）。	*net_all*

如何查看User级别的流量使用情况？

以下以收到报警事件UserEgressBandwidthThresholdExceeded为例，您可以通过以下步骤查看归属当前用户下各个Bucket的流量使用情况。

1. 登录云监控控制台。
2. 在左侧导航栏，选择**企业云监控 > 监控大盘**。
3. 在**大盘管理**页签，单击OSS监控大盘右侧的**查看**。
4. 根据流控报警类型，在**流量监控**区域查看具体哪些Bucket占用了较高的流量。

11.5. OSS性能与扩展性最佳实践

如果您在上传大量文件（Object）时，在命名上使用了顺序前缀（如时间戳或字母顺序），可能会出现大量文件索引集中存储于存储空间（Bucket）中的某个特定分区的情况。此时如果您的请求次数过多，会导致请求速率下降。出现此类问题时，建议您为文件名称增加随机前缀。

背景信息

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并承载高速率的客户请求。但是，OSS限制了在顺序读写模式下，每秒请求数QPS的值为2,000。如果您在上传大量文件时，在命名上使用了顺序前缀（如时间戳或字母顺序），可能会导致大量文件索引集中存储于某个特定分区。关于单个账号QPS的更多信息，请参见使用限制。

例如，当您的请求速率超过2000次/秒时（下载、上传、删除、拷贝、获取元数据信息等操作算1次操作，批量删除N个文件、列举N个文件等操作算N次操作），会带来如下后果：

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

> **说明** 分区数据再均衡是依赖于对当前系统状态、处理能力等信息做各种智能分析后进行的，并不是某个固定的拆分规则。所以分区数据再均衡后，使用了顺序前缀的文件可能还会处于高热点的分区当中。

为解决以上问题，您可以通过将文件名的顺序前缀改为随机性前缀，这样文件索引（以及I/O负载）就会均匀地分布在多个分区。

解决方案

以下提供了两个将顺序前缀改为随机性前缀的方法。

- 向文件名添加十六进制哈希前缀

  如果您使用日期与客户ID生成文件名，则会包含顺序时间戳前缀：

```
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，您可以对客户ID计算哈希（即MD5），并取若干字符的哈希前缀作为文件名的前缀。假如取4个字符的哈希前缀，结果如下：

```
sample-bucket-01/9b11/2017-11-11/customer-1/file1
sample-bucket-01/9fc2/2017-11-11/customer-2/file2
sample-bucket-01/d1b3/2017-11-11/customer-3/file3
...
sample-bucket-01/9fc2/2017-11-12/customer-2/file4
sample-bucket-01/f1ed/2017-11-12/customer-5/file5
sample-bucket-01/0ddc/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，则每个字符有0~9以及a~f共16种取值，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，以每个分区操作2000次/秒的性能瓶颈标准，再结合您业务的请求速率，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ListObject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。

- 反转文件名

  如果您使用了毫秒精度的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
...
```

  这种情况可以考虑通过反转时间戳前缀来避免文件名包含顺序前缀，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。