Alibaba Cloud
对象存储

最佳實践

檔案版本：20191022

目錄

1 資料移轉

1.1 Amazon S3資料移轉到OSS

OSS提供了S3 API的相容性，可以讓您的數據從Amazon S3無縫遷移到阿里雲OSS上。從Amazon S3遷移到OSS後，您仍然可以使用S3 API訪問OSS，僅需要對S3的客戶端應用進行如下改動：

1. **獲取OSS主帳號或子帳號的AccessKeyId和AccessKeySecret，並在您使用的客戶端和SDK中配置您申請的AccessKeyId與AccessKeySecret。**
2. **設定客戶端串連的endpoint為OSS endpoint。OSS endpoint列表請參見訪問網域名稱和資料中心。**

遷移步驟

從第三方儲存遷移到OSS的具體操作步驟，請您參見使用*OssImport*遷移數據。

遷移後用S3 API訪問OSS

從S3遷移到OSS後，您使用S3 API訪問OSS時，需要注意以下幾點：

- **Path style和Virtual hosted style**

  *Virtual hosted style*是指將Bucket放入host header的訪問方式。OSS基於安全考慮，僅支援virtual hosted訪問方式，所以在S3到OSS遷移後，客戶端應用需要進行相應設定。部分S3工具預設使用Path style，也需要進行相應配置，否則可能導致OSS報錯禁止訪問。
- **OSS對許可權的定義與S3不完全一致，遷移後如有需要，可對許可權進行相應調整。二者的主要區別可參考下表。**

  说明：

  - **更詳細的區別請參見*ACL*驗證流程。**
  - **OSS僅支援S3中的私有、公共讀和公共讀寫三種ACL模式。**

對象	Amazon S3許可權	Amazon S3	阿里雲OSS
Bucket	READ	擁有Bucket的list許可權	對於Bucket下的所有Object，如果某Object沒有設定Object許可權，則該Object可讀

對象	Amazon S3許可權	Amazon S3	阿里雲OSS
	WRITE	Bucket下的Object可寫入或覆蓋	- 對於Bucket下不存在的Object，可寫入 - 對於Bucket下存在的Object，如果該Object沒有設定Object許可權，則該Object可被覆蓋 - 可以initiate multipart upload
	READ_ACP	讀取Bucket ACL	讀取Bucket ACL，僅Bucket owner和授權子帳號擁有此許可權
	WRITE_ACP	設定Bucket ACL	設定Bucket ACL，僅Bucket owner和授權子帳號擁有此許可權
Object	READ	Object可讀	Object可讀
	WRITE	N/A	Object可以被覆蓋
	READ_ACP	讀取Object ACL	讀取Object ACL，僅Bucket owner和授權子帳號擁有此許可權
	WRITE_ACP	設定Object ACL	設定Object ACL，僅Bucket owner和授權子帳號擁有此許可權

- **儲存類型**

  OSS支援標準（Standard）、低頻訪問（IA）和Archive Storage（Archive）三種儲存類型，分別對應Amazon S3中的STANDARD、STANDARD_IA和GLACIER。

  與Amazon S3不同的是，OSS不支援在上傳object時直接指定儲存類型，object的儲存類型由bucket的儲存類型指定。OSS支援標準、低頻訪問和歸檔三種Bucket類型，Object的儲存類型還可以通過lifecycle進行轉換。

  Archive Storage類型的object在讀取之前，要先使用Restore請求進行解凍操作。與S3不同，OSS會忽略S3 API中的解凍天數設定，解凍狀態預設持續1天，用戶可以延長到最多7天，之後，Object又回到初始時的冷凍狀態。

- **ETag**
  - **對於PUT方式上傳的Object，OSS Object的ETag與Amazon S3在大小寫上有區別。OSS為大寫，而S3為小寫。如果客戶端有關於ETag的校驗，請忽略大小寫。**
  - **對於分區上傳的Object，OSS的ETag計算方式與S3不同。**

相容的S3 API

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

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

1.2 使用OssImport遷移數據

本文向您介紹如何使用OssImport將數據從第三方儲存（或OSS）遷移到OSS。

工具選擇：單機模式和分布式模式

OssImport有單機模式和分布式模式兩種部署方式。一般建議使用分布式模式。您參考OssImport官網指導文檔，即可完成遷移過程。本文介紹您在整體遷移方案中可能會關注的內容，以及可以參考的官網文檔資源。

遷移方案（從第三方儲存遷移到OSS）

以下步驟可以完成從其它儲存到OSS的無縫切換（參考官網支援的第三方儲存類型*OssImport* 架構及配置）：

具體步驟如下：

1. 全量遷移T1前的曆史數據，請參考：*OssImport* 架構及配置。
   - 記錄遷移開始時間T1（注意為Unix時間戳記，即自1970年1月1日UTC零點以來的秒數，通過命令`date +%s`獲取）。
   - 遷移指導說明參考OssImport官網文檔，請參考遷移工具-分布式。
2. 開啟OSS鏡像回源，並將讀寫切換到OSS，遷移源不再新增數據。
   - 步驟1遷移完成後，在OSS控制台開啟*OSS*鏡像回源功能，回源地址為遷移源（第三方儲存）。
   - 在業務系統讀寫切換到OSS，假設業務系統修改好的時間為T2。
   - 此時T1前的數據從OSS讀取，T1後的數據，OSS利用鏡像回源從第三方服務讀取，而新數據完全寫入OSS。
3. 快速遷移T1~T2到數據。
   - 在步驟2完成後，第三方儲存不會再新增數據，數據讀寫已切到OSS。
   - 修改設定檔`job.cfg`的配置項`importSince=T1`，新發起遷移job，遷移T1~T2數據。
4. 步驟3完成後，即完成遷移全過程。
   - 步驟3完成後，您業務的所有的讀寫都在OSS上，第三方儲存只是一份曆史數據，您可以根據需要決定保留或刪除。
   - OssImport負責數據的遷移和校驗，不會刪除任何數據。

遷移成本

遷移過程涉及到成本一般有ECS費用、流量費用、儲存費用、時間成本。其中，多數情況下，比如數據超過TB等級，儲存成本和遷移時間成正比，而ECS費用相對流量、儲存費用較小。

環境準備

假設您有如下遷移需求：

需求項	需求情況
遷移源	AWS S3東京
遷移目的	OSS香港
數據量	500TB
遷移時間要求	1周內完成遷移

您需要準備的環境：

環境項	說明
開通OSS	開通OSS步驟如下： 1. 使用您的帳號建立香港地域的OSS Bucket。 2. 在RAM控制台建立子帳號，並授權該子帳號可訪問OSS。保存AccessKeyID和AccessKeySecret。
購買ECS	購買OSS同區域(香港)的ECS，一般普通的2核4G機型即可，如果遷移後ECS需釋放，建議按需購買ECS。正式遷移時的ECS數量，請參考關於遷移用的*ECS*數量。
配置OssImport	說明： *destDomain*參數，即OSS目的endpoint，建議設定為OSS內網的endpoint，避免從ECS數據上傳到OSS時產生外網流量費用。具體的endpoint，請參考訪問網域名稱和資料中心。
遷移過程	1. 在ECS上搭建OssImport分布式環境。 2. 使用OssImport從東京AWS S3下載數據到ECS（香港），建議使用外網。 3. 使用OssImport從ECS（香港）將數據上傳到OSS（香港），建議使用內網。

關於遷移用的ECS數量

您需根據遷移需求，計算您需要用來做遷移的ECS數量：

- **假設您需要遷移的數據量是X TB，要求遷移完成時間Y天，單台遷移速度Z Mbps（每天遷移約Z/100 TB數據）。**
- **則正式遷移時，ECS大致需要X/Y/(Z/100)台。**

假設單台ECS遷移速度達到200Mbps(即每天約遷移2TB數據)。則上面Case中，ECS約需36台（500/7/2）。

OssImport遷移步驟

配置參考

您可以閱讀官網指導文檔，了解配置定義*OssImport*架構及配置、操作步驟分布式部署，在開始前請關注如下資訊：

- OssImport下載：在Master下載OssImport分布式版，且master、workers都使用同樣的ssh帳號、密碼。（worker上不用單獨下載OssImport，運行`deploy`命令時，OssImport會分發到worker上，請參考分布式部署。）
- Java環境：確認Master和worker都已安裝。

  
  说明:
  作為worker的ECS也需要安裝Java。
- 設定workdir：通過`conf/sys.properties`的配置項`workingDir`指定，請參考分布式部署。

  
  说明:
  workdir的設定請參考官網文檔，不要設定成OssImport包所在的路徑，同時盡量不要設定為已有內容的目錄。
- 並發控制：`conf/job.cfg`的配置項`taskObjectCountLimit`，`conf/sys.properties`的配置項`workerTaskThreadNum`，請參考*OssImport* 架構及配置。

  如果小檔案比較多，單台ECS遷移速度上不去且CPU load不高，可以參考調高`workerTaskThreadNum`、調低`taskObjectCountLimit`查看效果。
- 其他動作過程遇到問題，一般都是配置問題，請參考分布式部署，並可以查看master和worker上`workdir/Logs`的記錄檔。

前期測試

建議您先搭建小型環境（比如2~3台ECS），遷移少量數據以驗證配置正確與否。單台ECS遷移頻寬能否達到預期，比如200Mbps，如您對遷移時間明確無要求，可不關注。

頻寬查看：您可使用`iftop`或`Nload`（需先安裝，如`yum install ***`），ECS控制台頻寬統計有延時。


说明:
如果測試時檔案數目較少，可能無法驗證並發性，可以減少參數`taskObjectCountLimit`，比如減少到：檔案數/workerTaskThreadNum/worker總數。

遷移步驟

1. 遷移歷史數據。
   - 清任務、清配置。
   - 操作過程請參考分布式部署。
   - 開始遷移。
   - 您可在OSS控制台OSS Bucket中查看實際遷移完成的數據。
2. 設定鏡像回源，客戶業務系統讀寫切到OSS。
   - 在OSS控制台開啟OSS鏡像回源，回源地址為遷移源（第三方儲存）。
   - 客戶業務系統讀寫切換到OSS，假設業務系統修改好的時間為T2，則T2後不再有新數據寫到遷移源。
3. 遷移剩餘的數據。

   修改`job.cfg`配置項`importSince=T1`，請參考*OssImport* 架構及配置，遷移剩餘數據（T1~T2）特殊情況下，也可以使用`job.cfg`中`importSince = 0, isSkipExistFile=true`進行再次遷移，請參考*OssImport* 架構及配置。

關於遷移速度

- 單台遷移速度：如單台遷移速度不理想（比如小於200Mbps、且CPU load不高時），可參考官網文檔和上文，優化並發控制參數，即`conf/job.cfg`的配置項`taskObjectCountLimit`，`conf/sys.properties`的配置項`workerTaskThreadNum`。
- 遷移ECS數量：參考ECS數量估算（相對於流量、儲存、時間成本，ECS費用，在遷移總成本中佔比較少）。加大ECS數量，會減少遷移時間。

OssImport分布式相關官網文檔

序號	官網文檔
1	*OssImport* 分布式部署
2	*OssImport* 架構及配置
3	*OssImport* 最佳實踐
4	*OssImport* 常見問題

1.3 如何將HDFS容災備份到OSS

背景

當前業界有很多公司是以Hadoop技術構建資料中心，而越來越多的公司和企業希望將業務順暢地遷移到雲上。

在阿里雲上使用最廣泛的儲存服務是Object Storage Service。OSS的資料移轉工具ossimport2可以將您本地或第三方雲端儲存體服務上的檔案同步到OSS上，但這個工具無法讀取Hadoop檔案系統的數據，從而發揮Hadoop分布式的特點。並且，該工具只支援本地檔案，需要將HDFS上的檔案先下載到本地，再通過工具上傳，整個過程耗時又耗力。

阿里雲E-MapReduce團隊開發的Hadoop資料移轉工具emr-tools，能讓您從Hadoop叢集隨即轉移數據到OSS上。

本文介紹如何快速地將Hadoop檔案系統（HDFS）上的資料移轉到OSS。

前提條件

確保當前機器可以正常訪問您的Hadoop叢集，即能夠用Hadoop命令訪問HDFS。

```
hadoop fs -ls /
```

Hadoop資料移轉到OSS

1. 下載*emr-tools*。

   

   说明:
   emr-tools相容Hadoop 2.4.x、2.5.x、2.6.x、2.7.x版本，如果有其他Hadoop版本相容性的需求，請**提交工單**。

2. 解壓縮工具到本地目錄。

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

3. 複製HDFS數據到OSS上。

   ```
   cd emr-tools
   ```

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

參數說明如下。

參數	說明
accessKeyId	訪問OSS API的密鑰。
accessKeySecret	獲取方式請參見如何獲取如何獲取*AccessKeyId*和*AccessKeySecret*。
bucket-name.oss-cn-hangzhou.aliyuncs.com	OSS的訪問網域名稱，包括bucket名稱和endpoint地址。

系統將啟動一個Hadoop MapReduce任務（DistCp）。

4. 運行完畢之後會顯示本次資料移轉的資訊。資訊內容類別似如下所示。

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

```
            Bytes Read=3502
    File Output Format Counters
            Bytes Written=0
    org.apache.hadoop.tools.mapred.CopyMapper$Counter
            BYTESCOPIED=258660
            BYTESEXPECTED=258660
            COPY=10
copy from /path/on/hdfs to oss://accessKeyId:accessKeySecret@bucket-
name.oss-cn-hangzhou.aliyuncs.com/path/on/oss does succeed !!!
```

5. 您可以用osscmd等工具查看OSS上數據情況。

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

OSS資料移轉到Hadoop

如果您已經在阿里雲上搭建了Hadoop叢集，可以使用如下命令把數據從OSS上遷移到新的Hadoop叢集。

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

更多使用場景

除了線下的叢集，在ECS上搭建的Hadoop叢集也可以使用emr-tools，將自建叢集迅速地遷移到*E-MapReduce*服務上。

如果您的叢集已經在ECS上，但是在經典網路中，無法和VPC中的服務做很好的互操作，所以想把叢集遷移到VPC中。可以按照如下步驟遷移：

1. 使用emr-tools遷移數據到OSS上。
2. 在VPC環境中新建一個叢集（自建或使用E-MapReduce服務）。
3. 將數據從OSS上遷移到新的HDFS叢集中。

如果你使用E-MapReduce服務，還可以直接在Hadoop叢集中通過*Spark*、*MapReduce*和*Hive*等組件訪問OSS，這樣不僅可以減少一次數據複製（從OSS到HDFS），還可以極大的降低儲存成本。有關降低成本的詳細資料，請參見*EMR*+OSS：計算與儲存分離。

2 Web端直傳實踐

3 行動裝置 App端直傳實踐

3.1 快速搭建行動裝置 App直傳服務

本文介紹如何在30分鐘內搭建一個基於OSS的行動裝置 App數據直傳服務。直傳指的是行動裝置App數據的上傳和下載直接連接OSS，只有控制流程串連自己的伺服器。

背景

在移動互聯的時代，手機app上傳的數據越來越多。作為開發人員，您可以利用OSS處理各種資料存放區需求，從而更加專註於自己的應用邏輯。

基於OSS的行動裝置 App數據直傳服務具有以下優勢：

- **數據安全：使用靈活的授權和鑒權方式進行數據的上傳和下載，更加安全。**
- **成本低廉：您不需要準備很多伺服器。行動裝置 App直接連接雲端儲存OSS，只有控制流程串連應用伺服器。**
- **高並發：支援海量用戶並發訪問。**
- **彈性擴容：無限擴容的儲存空間。**
- **資料處理：和圖片處理以及音視頻轉碼搭配使用，方便靈活地進行資料處理。**

下載並安裝行動裝置 App

行動裝置 App源碼的下載地址如下：

- **Android：下載地址**
- **iOS：下載地址**

您可以通過此行動裝置 App上傳圖片到OSS。上傳的方法支援普通上傳和斷點續傳上傳。在網路環境差的情況下，推薦使用斷點續傳上傳。您還可以利用圖片處理服務，對要上傳的圖片進行縮略和加浮水印處理。樣本應用的最終效果圖如下：

- **應用伺服器：該行動裝置 App對應的後台應用伺服器。請使用步驟2：下載應用伺服器程式碼範例中提供的應用伺服器程式碼範例，部署自己的應用伺服器。**
- **上傳Bucket：該行動裝置 App要把數據上傳到哪個Bucket。**
- **區域：上傳Bucket所在的地域。**

原理介紹

行動裝置 App直傳服務的架構圖如下所示：

- Android/iOS 行動裝置 App：即終端使用者手機上的app。
- OSS：即阿里雲對象儲存，負責儲存app上傳的數據。
- RAM/STS：負責生成臨時上傳憑證，即Token。
- 應用伺服器：即提供該Android/iOS應用的開發人員開發的app後台服務，用於管理app上傳和下載的Token，以及用戶通過app上傳數據的元資訊。

實現步驟如下：

1. 行動裝置 App嚮應用伺服器申請一個臨時上傳憑證。

    说明:
   Android和iOS應用不能直接儲存AccessKey，這樣會存在數據泄露的風險。所以應用必須向用戶的應用伺服器申請一個Token。這個Token是有時效性的，如果Token的過期時間是30分鐘（由應用伺服器指定），那麼在這30分鐘裡，該Android/iOS應用可以使用此Token從OSS上傳和下載數據， 30分鐘後需要重新獲取Token。
2. 應用伺服器檢測上述請求的合法性，然後返回Token給行動裝置 App。
3. Android/iOS行動裝置 App使用此Token將數據上傳到OSS，或者從OSS下載數據。

以下介紹應用伺服器如何生成Token以及Android/iOS行動裝置 App如何獲取Token。

步驟1：建立Bucket並開通STS服務

1. 開通*OSS*，並且建立*Bucket*。
2. 開通STS服務。
   a. 登入*OSS*管理主控台。
   b. 在OSS概覽頁中找到基礎配置區域，單擊安全性權杖。
   c. 進入到安全性權杖快捷配置頁面，單擊開始授權。

       说明:

如果沒有開通RAM，會彈出開通的對話方塊。單擊開通。開通完成後單擊開始授權。

d. 系統進行自動授權。請保存AccessKeyID、AccessKeySecret和RoleArn三個參數。

- 如果您未建立過AccessKey，系統自動建立AccessKey。請保存AccessKeyId、AccessKeySecret和RoleArn，如下圖所示。

- 如果您之前已經建立過AccessKey，也可以單擊如下圖所示的查看建立新的AccessKey。

步驟2：下載應用伺服器程式碼範例

- PHP：下載地址
- Java：下載地址
- Ruby：下載地址
- Node.js：下載地址

步驟3：修改設定檔

以下例子採用PHP編寫。每個語言的範例程式碼下載後，都會有一個設定檔config.json，如下所示：

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

相關參數說明如下：

- AccessKeyID：填寫之前記錄的AccessKeyID。
- AccessKeySecret：填寫之前記錄的AccessKeySecret。
- RoleArn：填寫之前記錄的RoleArn。
- TokenExpireTime：指Android/iOS應用獲取到的Token的失效時間，最小值和預設值均為900s。
- PolicyFile：該Token所擁有的許可權列表的檔案，可使用預設值。

說明：

程式碼範例中提供了三種最常用的Token許可權檔案，位於policy目錄下面。分別是：

- **all_policy.txt：指定該Token擁有該帳號下的所有許可權，包括：**
  - **■ 建立Bucket**
  - **■ 刪除Bucket**
  - **■ 上傳檔案**
  - **■ 下載檔案**
  - **■ 刪除檔案**
- **bucket_read_policy.txt：指定該Token擁有該帳號下指定Bucket的讀許可權。**
- **bucket_read_write_policy.txt：指定了該Token擁有該帳號下指定Bucket的讀寫權限。**

如果您想要指定該Token只對指定的Bucket有讀寫權限， 請把bucket_read_policy.txt和bucket_read_write_policy.txt檔案裡的$BUCKET_NAME替換成指定的Bucket名稱。

步驟4：運行範例程式碼

程式碼範例的運行方法如下：

- **對於PHP版本，將包下載解壓後，修改config.json檔案，直接運行php sts.php即能生成Token，將程式部署到指定的應用伺服器位址。**
- **對於Java版本（依賴於java 1.7），將包下載解壓後，修改config.json檔案，運行java -jar oss-token-server.jar (port)。如果不指定port（通信埠），直接運行java –jar oss-token-server.jar，程式會監聽7080通信埠。**

返回的資料格式解析如下：

```
//正確返回
{
    "StatusCode":200,
    "AccessKeyId":"STS.3p***dgagdasdg",
    "AccessKeySecret":"rpnwO9***tGdrddgsR2YrTtI",
   "SecurityToken":"CAES+wMIARKAAZhjH0EUOIhJMQBMjRywXq7MQ/cjLYg80Aho
1ek0Jm63XMhr9Oc5s˙ə˙ə3qaPer8p1YaX1NTDiCFZWFkvlHf1pQhuxfKBc+mRR9KAbHUe
fqH+rdjZqjTF7p2m1wJXP8S6k+G2MpHrUe6TYBkJ43GhhTVFMuM3BZajY3VjZWOXBI
ODRIR1FKZjIiEjMzMzE0MjY0NzM5MTE4NjkxMSoLY2xpZGSSDgSDGAGESGTETqOio6c2Rr
LWRlbW8vKgoUYWNzOm9zczoqOio6c2RrLWRlbW9KEDExNDg5MzAxMDcyNDY4MThSBTI2OD
QyWg9Bc3N1bWVkUm9sZVVzZXJgAGoSMzMzMTQyNjQ3MzkxMTg2OTExcglzZGstZGVtbzI
=",
   "Expiration":"2017-12-12T07:49:09Z",
}
//錯誤返回
{
    "StatusCode":500,
    "ErrorCode":"InvalidAccessKeyId.NotFound",
    "ErrorMessage":"Specified access key is not found."
}
```

正確返回的五個變數構成一個Token：

- StatusCode：獲取Token的狀態，獲取成功時，傳回值是200。
- AccessKeyId：Android/iOS行動裝置 App初始化OSSClient獲取的 AccessKeyId。
- AccessKeySecret：Android/iOS行動裝置 App初始化OSSClient獲取AccessKeySecret。
- SecurityToken：Android/iOS行動裝置 App初始化的Token。
- Expiration：該Token失效的時間。Android SDK會自動判斷Token是否失效，如果失效，則自動獲取Token。

錯誤返回說明如下：

- StatusCode：表示獲取Token的狀態，獲取失敗時，傳回值是500。
- ErrorCode：表示錯誤原因。
- ErrorMessage：表示錯誤的具體資訊描述。

步驟5：體驗行動裝置 App直傳服務

1. 部署程式後，記下應用伺服器位址如`http://abc.com:8080`，將樣本程式裡面的應用伺服器修改成上述地址。
2. 選擇數據要上傳到哪個Bcuket及地域，修改樣本程式裡相應的上傳Bucket及區域。
3. 單擊設定，載入配置。
4. 選擇圖片，設定上傳OSS檔案名，上傳圖片。
5. 上傳成功後，通過控制台查看上傳結果。

核心代碼解析

OSS初始化的代碼解析如下：

- Android版本

```
// 推薦使用OSSAuthCredentialsProvider，token過期後會自動刷新。
String stsServer = "應用伺服器位址，例如http://abc.com:8080"
OSSCredentialProvider credentialProvider = new OSSAuthCredentialsPr
ovider(stsServer);
//config
ClientConfiguration conf = new ClientConfiguration();
conf.setConnectionTimeout(15 * 1000); // 連接逾時時間，預設15秒
conf.setSocketTimeout(15 * 1000); // Socket逾時時間，預設15秒
conf.setMaxConcurrentRequest(5); // 最大並發請求數，預設5個
conf.setMaxErrorRetry(2); // 失敗後最大重試次數，預設2次
OSS oss = new OSSClient(getApplicationContext(), endpoint,
credentialProvider, conf);
```

- iOS版本

```
OSSClient * client;
...
// 推薦使用OSSAuthCredentialProvider，token過期後會自動刷新。
```

```
id<OSSCredentialProvider> credential = [[OSSAuthCredentialProvider
alloc] initWithAuthServerUrl:@"應用伺服器位址，例如http://abc.com:8080
"];
client = [[OSSClient alloc] initWithEndpoint:endPoint credential
Provider:credential];
```

3.2 許可權控制

本文基於快速搭建行動裝置 App直傳服務中提到的應用伺服器，以上海的Bucket app-base-oss為例，講解如何配置不同的Policy實現不同的許可權控制。

说明:

- **本文提到的Policy是指快速搭建行動裝置 App直傳服務提到的config.json中指定的Policy檔案內容。**
- **以下講述的獲取STS Token 後對OSS的操作是指為應用伺服器指定Policy。從STS獲取臨時憑證後，應用通過臨時憑證訪問OSS。**

常見Policy

- **完全授權的Policy**

  完全授權的Policy表示允許應用可以對OSS進行任何操作。

警告:

完全授權的Policy對行動裝置 App來說是不安全的授權，不推薦使用。

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

獲取STS Token 後對OSS的操作	結果
列出所有建立的Bucket	成功
上傳不帶首碼的Object，test.txt	成功
下載不帶首碼的Object，test.txt	成功
上傳帶首碼的Object，user1/test.txt	成功
下載帶首碼的Object，user1/test.txt	成功

獲取STS Token 後對OSS的操作	結果
列出Object，test.txt	成功
帶首碼的Object，user1/test.txt	成功

- 不限制首碼的只讀不寫Policy

  此Policy表示應用對Bucket`app-base-oss`下所有的Object可列舉，可下載。

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

獲取STS Token 後對OSS的操作	結果
列出所有建立的Bucket	失敗
上傳不帶首碼的Object，test.txt	失敗
下載不帶首碼的Object，test.txt	成功
上傳帶首碼的Object，user1/test.txt	失敗
下載帶首碼的Object，user1/test.txt	成功
列出不帶首碼的Object，test.txt	成功
列出帶首碼的Object，user1/test.txt	成功

- 限制首碼的只讀不寫Policy

  此Policy表示應用對Bucket`app-base-oss`下帶有首碼`user1/`的Object可列舉、可下載，但無法下載其他首碼的Object。採用此種Policy，如果不同的應用對應不同的首碼，就可以達到在同一個Bucket中空間隔離的效果。

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

```
    ],
    "Version": "1"
  }
```

獲取STS Token 後對OSS的操作	結果
列出所有建立的Bucket	失敗
上傳不帶首碼的Object，test.txt	失敗
下載不帶首碼的Object，test.txt	失敗
上傳帶首碼的Object，user1/test.txt	失敗
下載帶首碼的Object，user1/test.txt	成功
列出Object，test.txt	成功
帶首碼的Object，user1/test.txt	成功

- 不限制首碼的唯寫不讀Policy

  此Policy表示應用可以對Bucket`app-base-oss`下所有的Object進行上傳。

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

獲取STS Token 後對OSS操作	結果
列出所有建立的Bucket	失敗
上傳不帶首碼的Object，test.txt	成功
下載不帶首碼的Object，test.txt	失敗
上傳帶首碼的Object，user1/test.txt	成功
下載帶首碼的Object，user1/test.txt	成功
列出Object，test.txt	成功
帶首碼的Object，user1/test.txt	成功

- **限制首碼的唯寫不讀Policy**

  此Policy表示應用可以對Bucket`app-base-oss`下帶有首碼`user1/`的Object進行上傳。但無法上傳其他首碼的Object。採用此種Policy，如果不同的應用對應不同的首碼，就可以達到在同一個Bucket中空間隔離的效果。

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

獲取STS Token 後對OSS的操作	結果
列出所有建立的Bucket	失敗
上傳不帶首碼的Object，test.txt	失敗
下載不帶首碼的Object，test.txt	失敗
上傳帶首碼的Object，user1/test.txt	失敗
下載帶首碼的Object，user1/test.txt	成功
列出Object，test.txt	失敗
帶首碼的Object，user1/test.txt	失敗

- **不限制首碼的讀寫Policy**

  此Policy表示應用可以對Bucket`app-base-oss`下所有的Object進行列舉、下載、上傳和刪除。

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

```
  }
```

獲取STS Token 後對OSS的操作	結果
列出所有建立的Bucket	失敗
上傳不帶首碼的Object，test.txt	成功
下載不帶首碼的Object，test.txt	成功
上傳帶首碼的Object，user1/test.txt	成功
下載帶首碼的Object，user1/test.txt	成功
列出Object，test.txt	成功
帶首碼的Object，user1/test.txt	成功

- 限制首碼的讀寫Policy

  此Policy表示應用可以對Bucket`app-base-oss`下帶有首碼`user1/`的Object進行列舉、下載、上傳和刪除，但無法對其他首碼的Object進行讀寫。採用此種Policy，如果不同的應用對應不同的首碼，就可以達到在同一個Bucket中空間隔離的效果。

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

獲取STS Token 後對OSS的操作	結果
列出所有建立的Bucket	失敗
上傳不帶首碼的Object，test.txt	失敗
下載不帶首碼的Object，test.txt	失敗
上傳帶首碼的Object，user1/test.txt	成功
下載帶首碼的Object，user1/test.txt	成功
列出Object，test.txt	成功
帶首碼的Object，user1/test.txt	成功

總結

從上面的例子可以看出：

- **可以根據不同的應用場景制定不同的Policy，然後對應用伺服器稍作修改就可以實現對不用的應用用戶實現不同的許可權控制。**
- **可以在應用端做優化，使得STS Token過期之前不需要嚮應用伺服器再次請求。**
- **Token由STS頒發。應用伺服器只是定製了Policy，向STS請求Token，然後將Token轉寄給應用。這裡的Token包含了AccessKeyId、AccessKeySecret、Expiration、SecurityToken，這些參數在OSS提供給應用的SDK中會用到。詳情請參見各語言SDK參考中的授權訪問章節。**

參考文檔

- ***RAM*和*STS*在*OSS*中的使用指南**
- **阿里雲*RAM*官方文檔和阿里雲*STS*官方文檔**

3.3 快速搭建行動裝置 App上傳回調服務

背景

快速搭建行動裝置*App*直傳服務介紹了如何在30分鐘內中搭建一個基於OSS的行動裝置 App數據直傳服務。移動端開發場景流程圖如下：

角色分析如下所示：

- **應用伺服器負責為Android/iOS行動裝置 App生成STS憑證。**
- **Android/iOS行動裝置 App負責從應用伺服器申請及使用STS憑證。**
- **OSS負責處理行動裝置 App的數據請求。**

對於Android/iOS移動應來說，行動裝置 App只需要執行上圖中操作1（申請STS憑證），就能調用多次5(使用該STS憑證上傳數據到OSS)，導致應用伺服器根本不知道用戶都上傳了哪些數據，作為該APP的開發人員，就沒法對應用上傳數據進行管理。有什麼問題能讓應用伺服器感知到Android/iOS行動裝置 App上傳的數據呢?

您可以通過使用OSS的上傳回調服務，就能解決上述問題，如下圖所示：

OSS在收到Android/iOS移動的數據（上圖中操作5）和在返回用戶上傳結果（上圖中操作6）之間，觸發一個上傳回調工作。即第上圖中操作5.5，先回調使用者服務器，得到應用伺服器返回的內容，將這個內容返回給Android/iOS行動裝置 App。可以參考*Callback API*文檔。

上傳回調的作用

- 通過上傳回調可以讓用戶應用伺服器知道當前上傳檔案的基本資料。

  基本資料如下表。返回下述變數的一個或者多個，返回內容格式形式在Android/iOS上傳時指定。

系統變數	含義
bucket	行動裝置 App上傳到哪個儲存空間
object	行動裝置 App上傳到OSS保存的檔案名
etag	該上傳的檔案的etag，即返回給用戶的etag欄位
size	該上傳的檔案的大小
mimeType	資源類型
imageInfo.height	圖片高度
imageInfo.width	圖片寬度
imageInfo.format	圖片格式，如jpg、png，只以識別圖片

- 通過上傳回調設定自訂參數，達到資訊傳遞目的。

  假如您是一個開發人員，您想知道目前使用者所使用的APP版本、目前使用者所在的作業系統版本、用戶的GPS資訊、用戶的手機型號。您可以在Android/iOS端上傳檔案時，指定上述自訂參數，如下所示：

  - x:version指定APP版本
  - x:system指定作業系統版本
  - x:gps指定GPS資訊
  - x:phone指定手機型號

    上述這些值會在Android/iOS行動裝置 App上傳到OSS時附帶上，OSS會把這些值放到CallbackBody裡面一起發給應用伺服器。這樣應用伺服器就能收到這些資訊，達到資訊傳遞的目的。

在行動裝置 App端設定上傳回調

要讓OSS在接收上傳請求時，觸發上傳回調，行動裝置 App在構造上傳請求時必須把如下兩個內容指定到上傳請求裡面：

- 要回調到哪個伺服器callbackUrl，如 http://abc.com/callback.php，這個地址必須是公網能夠訪問的。
- 上傳回調給應用伺服器的內容callbackBody，可以是上述OSS返回應用伺服器系統變數的一個或者多個。

假如您的使用者服務器上傳回調地是http://abc.com/callback.php。您想獲取手機上傳的檔案名字、檔案的大小，並且定義了photo變數是指手機型號，system是指作業系統版本。

上傳回調樣本分以下兩種：

- iOS指定上傳回調樣本：

```
OSSPutObjectRequest * request = [OSSPutObjectRequest new];
request.bucketName = @"<bucketName>";
request.objectKey = @"<objectKey>";
request.uploadingFileURL = [NSURL fileURLWithPath:@<filepath>"];
// 設定回調參數
request.callbackParam = @{
                          @"callbackUrl": @"http://abc.com/callback.
php",
                          @"callbackBody": @"filename=${object}&size
=${size}&photo=${x:photo}&system=${x:system}"
                          };
// 設定自訂變數
request.callbackVar = @{
                        @"x:photo": @"iphone6s",
                        @"x:system": @"ios9.1"
                        };
```

- Android指定上傳回調樣本：

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

上傳回調對應用伺服器的要求

- 您必須部署一個可以接收POST請求的服務，這個服務必須有公網地址如www.abc.com/callback.php（或者外網IP也可以），不然OSS沒有辦法訪問到這個地址。

- **您要給OSS正確的返回，返回格式必須是JSON格式，內容自訂。因為OSS會把應用伺服器返回的內容，原封不動地返回給Android/iOS行動裝置 App。（切記，返回給OSS的Response Header一定要加上Content-Length這個頭部）。**

本教程在最後為大家準備了多個語言版本的樣本、下載及運行方法。

應用伺服器收到的回調請求

應用伺服器收到OSS的請求，抓包的請求如下（這個結果根據設定的不同URL和回調內容會有不同）：

```
POST /index.html HTTP/1.0
Host: 121.43.113.8
Connection: close
Content-Length: 81
Content-Type: application/x-www-form-urlencoded
User-Agent: ehttp-client/0.0.1
authorization: kKQeGTRccDKyHB3H9vF+xYMSrmhMZjzzl2/kdD1ktNVgbWEfYTQG0G2
SU/RaHBovRCE8OkQDjC3uG33esH2txA==
x-oss-pub-key-url: aHR0cDovL2dvc3NwdWJsaWMuYWxpY2RuLmNvbS9jYWxsYmFja1
9wdWJfa2V5X3YxLnBlbQ==
filename=test.txt&size=5&photo=iphone6s&system=ios9.1
```

更多內容請參考*Callback API*文檔。

應用伺服器判斷回調請求是否來自OSS

如果您的回調伺服器被人惡意攻擊了，例如惡意回調您的應用伺服器，導致應用伺服器收到一些非法的請求，影響正常邏輯，此時您就需要判斷回調請求是否來自OSS。

判斷的方法主要是利用OSS給應用伺服器返回的頭部內容中，`x-oss-pub-key-url`和`authorization`這兩個參數進行RSA校驗。只有通過RSA校驗才能說明這個請求是來自OSS，本教程提供的樣本程式都有實現的樣本供您參考。

應用伺服器收到回調請求後的處理

應用伺服器在校驗這個請求是來自OSS後，指定回調給應用伺服器的內容格式，如

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

應用伺服器就可以根據OSS的返回內容，解析得到自己想要得到的數據。得到這個數據後，應用伺服器可以把數據存放起來，方便後續管理。

應用伺服器收到回調請求後如何返回給OSS

- **返回狀態碼是200；**
- **返回必須是json格式的內容；**
- **返回的頭部必須帶有Content-Length這個頭部。**

OSS如何處理應用伺服器的返回內容

有兩種情況：

- **OSS將回調請求發送給應用伺服器，但是應用伺服器接收失敗或者訪問不通，OSS會返回給Android/iOS行動裝置 App203的狀態碼，但是數據已經存放到OSS上了。**
- **應用伺服器接收到OSS的回調請求，並且正確返回了，OSS會返回給Android/iOS行動裝置App狀態碼是200, 並把應用伺服器給OSS的內容，原封不動地返回給Android/iOS行動裝置App。**

上傳回調伺服器樣本程式下載

樣本程式只是完成了如何檢查應用伺服器收到的簽名， 用戶要自行增加對應用伺服器收到回調的內容的格式解析 。

- **Java版本：**
  - **下載地址：單擊這裡。**
  - **運行方法，解壓包運行`java -jar oss-callback-server-demo.jar 9000`（9000是啟動並執行通信埠，可以自己指定）。**

    说明:
    這個jar例子在java 1.7運行通過，如果有問題可以自己依據提供的代碼進行修改。這是一個maven項目。
- **PHP版本：**
  - **下載地址：單擊這裡**
  - **運行方法：部署到Apache環境下，因為PHP本身語言的特點，取一些數據頭部會依賴於環境。所以可以參考例子根據自己所在環境進行修改。**
- **Python版本：**
  - **下載地址：單擊這裡。**
  - **運行方法：解壓包直接運行python callback_app_server.py即可，程式自實現了一個簡單的http server，運行該程式可能需要安裝rsa的依賴。**
- **Ruby版本：**
  - **下載地址：單擊這裡。**
  - **運行方法： ruby aliyun_oss_callback_server.rb。**

4 資料處理與分析

4.1 基於OSS+MaxCompute構建資料倉儲

背景

本文重點介紹基於OSS並使用MaxCompute構建PB資料倉儲的方法。

- Object Storage Service

  Object Storage Service提供標準、低頻、Archive Storage類型，能夠覆蓋從熱到冷的不同儲存場景。同時，OSS能夠與Hadoop開源社區及EMR、批次運算、MaxCompute、機器學習服務PAI、DatalakeAnalytics、Function Compute等阿里雲計算產品進行深度結合。

  用戶可以打造基於OSS的資料分析應用，如MapReduce、HIVE/Pig/Spark等批處理（如日誌離線計算）、互動式查詢分析（Imapla、Presto、DataLakeAnalytics）、深度學習訓練（阿里雲PAI）、基因渲染計算交付（批次運算）、大數據應用（MaxCompute）及串流（Function Compute）等。
- MaxCompute

  MaxCompute是一項大數據計算服務，能夠提供快速且完全託管的資料倉儲解決方案，並可以與OSS結合，高效並經濟地分析處理海量數據。MaxCompute的處理性能達到了全球領先水平，被Forrester評為全球雲端資料倉儲領導者。
- OSS外部表格查詢功能

  MaxCompute重磅推出了一項重要特性：OSS外表查詢功能。該功能可以幫助您直接對OSS中的海量檔案進行查詢，而不必將數據載入到MaxCompute表中，既節約了數據搬遷的時間和人力，也節省了多地儲存的成本。

  使用MaxCompute+OSS的方案，您可以獲得以下優勢：

  - MaxCompute是一個無伺服器的分散式運算架構，無需用戶對伺服器基礎設施進行額外的維護和管理，能夠根據OSS用戶的需求方便及時地提供臨時查詢服務，幫助企業大幅節省成本。
  - OSS為海量的對象儲存服務。用戶將數據統一儲存在OSS，可以做到計算和儲存分離，多種計算應用和業務都可以訪問使用OSS的數據，數據只需要儲存一份。
  - 支援處理OSS上開源格式的結構化檔案，包括：Avro、CSV、ORC、Parquet、RCFile、RegexSerDe、SequenceFile和TextFile，同時支援gzip壓縮格式。

典型應用場景

互連網金融應用每天都需要將大量的金融數據分頁檔存放在OSS上，並需要進行超大文字檔的結構化分析。通過MaxCompute的OSS外部表格查詢功能，用戶可以直接用外部表格的方式將OSS上的大檔案載入到MaxCompute進行分析，從而大幅提升整個鏈路的效率。

操作步驟

下面我們通過兩個簡單的樣本，介紹如何通過MaxCompute外表功能實現對OSS數據的分析和處理。

- **場景一：物聯網採集資料分析**

  1. **步驟1：準備工作**

     a. **開通OSS和MaxCompute服務。**

        您可以通過官網分別開通OSS、MaxCompute服務，並建立OSS bucket、MaxCompute Project。

     b. **採集數據到OSS。**

        您可以使用任何資料集來執行測試，以驗證我們在這篇文章中概述的最佳實踐。本樣本在OSS上準備了一批 CSV 數據，endpoint為oss-cn-beijing-internal.aliyuncs.com，bucket為oss-odps-test，資料檔案的存放路徑為 /demo/vehicle.csv。

     c. **授權MaxCompute訪問OSS。**

        MaxCompute需要直接存取OSS的數據，因此需要將OSS的數據相關許可權賦給MaxCompute的訪問帳號。您可以在直接登入阿里雲帳號後，點擊完成授權。

  2. **步驟2：通過MaxCompute建立外部表格**

     建立外部表格，語句如下：

```
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/';
```

3. **步驟3：通過MaxCompute查詢外部表格**

   成功建立外部表格後，便可如普通表一樣使用該外部表格。

   假設 /demo/vehicle.csv 的數據如下：

```
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
```

   執行如下 SQL 語句：

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

   輸出結果如下：

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

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

- 場景二：阿里雲產品消費賬單分析

1. 步驟1：準備工作

   a. 開通OSS和MaxCompute服務。

   您可以通過官網分別開通OSS和MaxCompute服務，並建立OSS bucket、MaxCompute Project。

   b. 授權MaxCompute訪問OSS。

   MaxCompute需要直接存取OSS的數據，因此需要將OSS的數據相關許可權賦給MaxCompute的訪問帳號。您可以在直接登入阿里雲帳號後，點擊完成授權。

2. 步驟2：通過費用中心同步賬單數據到OSS

   a. 服務開通後，每天會將增量的實例消費詳細資料組建檔案，並同步儲存到指定的bucket中，如下圖所示：

3. 步驟3：通過MaxCompute註冊賬單處理類

   a. 點擊以下連結下載自訂代碼：*odps-udf-example-0.30.0-SNAPSHOT-jar-with-dependencies.jar*

   b. 運行以下命令，將自訂代碼編譯打包，並上傳到 MaxCompute。

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

4. 步驟4：通過MaxCompute建立外部表格

   以建立5月4日的賬單消費表為例，外部表格如下：

   ```
   CREATE EXTERNAL TABLE IF NOT EXISTS oms_oss_0504
   (
   月份 string,
   資源擁有者 string,
   消費時間 string,
   消費類型 string,
   賬單編號 string,
   商品 string,
   計費方式 string,
   服務開始時間 string,
   服務結束時間 string,
   服務時長 string,
   財務核算單元 string,
   資源id string,
   資源暱稱 string,
   TAG string,
   地域 string,
   可用區 string,
   公網ip string,
   內網ip string,
   資源配置 string,
   原價 string,
   優惠金額 string,
   應付金額 string,
   ```

```
計費項1 string,
使用量1 string,
資源套件扣除1 string,
原價1 string ,
應付金額1 string,
計費項2 string,
使用量2 string,
資源套件扣除2 string,
原價2 string,
應付金額2 string,
計費項3 string,
使用量3 string,
資源套件扣除3 string,
原價3 string,
應付金額3 string,
計費項4 string,
使用量4 string,
資源套件扣除4 string,
原價4 string,
應付金額4 string,
計費項5 string,
使用量5 string,
資源套件扣除5 string,
原價5 string,
應付金額5 string,
計費項6 string,
使用量6 string,
資源套件扣除6 string,
原價6 string,
應付金額6 string,
計費項7 string,
使用量7 string,
資源套件扣除7 string,
原價7 string,
應付金額7 string,
計費項8 string,
使用量8 string,
資源套件扣除8 string,
原價8 string,
應付金額8 string,
計費項9 string,
使用量9 string,
資源套件扣除9 string,
原價9 string,
應付金額9 string
)
STORED BY 'com.aliyun.odps.udf.example.text.TextStorageHandler' --
STORED BY 指定自訂 StorageHandler 的類名。
with SERDEPROPERTIES (
'odps.text.option.complex.text.enabled'='true',
'odps.text.option.strict.mode'='false'
--遇到列數不一致的情況不會拋異常，如果實際列數少於schema列數，將所有列按順序匹配，剩下的不足的列補NULL。
)
LOCATION 'oss://oss-cn-beijing-internal.aliyuncs.com/oms-yl/2018-05-04/'
```

```
USING 'text_oss.jar'; --同時需要指定賬單中的文本處理類定義所在的 jar 包。
```

5. **步驟5：通過MaxCompute查詢外部表格**

   以查詢ECS快照消費賬單明細為例，命令如下：

```
select  月份,原價,優惠金額,應付金額,計費項1,使用量 from oms_oss where  商品=快照(SNAPSHOT);
```

總結

上述樣本說明了如何通過MaxCompute對OSS上的海量數據進行分析，將您的大資料分析工作效率提升至分鐘級，幫助您更高效、更低成本的挖掘海量數據價值。

4.2 EMR+OSS：離線計算的儲存與計算分離

背景

在傳統Hadoop的使用中，儲存與計算密不可分，而隨著業務的發展，叢集的規模常常不能滿足業務的需求。例如，數據規模超過了叢集儲存能力，業務上對數據產出的周期提出新的要求導致計算能力跟不上。這就要求我們能隨時應對叢集儲存空間不足或者計算能力不足的挑戰。

如果將計算和儲存混合部署，常常會因為為了擴儲存而帶來額外的計算擴容，這其實就是一種浪費；同理，只為了提升計算能力，也會帶來一段時期的儲存浪費。

將離線計算的計算和儲存分離，可以更好地應對單方面的不足。把數據全部放在OSS中，再通過無狀態的E-MapReduce分析。E-MapReduce只需進行純粹的計算，不存在儲存跟計算搭配來適應業務了，這樣最為靈活。

架構

離線計算的儲存和計算分離架構簡單，如下圖所示。OSS作為預設的儲存，Hadoop/Spark作為計算引擎直接分析OSS儲存的數據。

優勢

因素	計算和儲存不分離	計算和儲存分離
靈活性	不靈活	計算與儲存分離後，叢集規劃簡單靈活，基本不需要估算未來業務的規模，做到按需使用。

因素	計算和儲存不分離	計算和儲存分離
成本	高	在ECS自建的磁碟選擇高效雲盤，以1 master 8 cpu32g/6 slave 8 cpu32g/10T數據量為例進行估算，儲存與計算分離後，成本下降一倍。
性能	較高	至多下降10%。

案例測試

- 測試條件

  詳細的測試代碼請參見*GitHub*。

  叢集規模：1 master 4cpu 16g、8 Slave 4cpu 16g、每個slave節點250G*4 高效雲盤

  Spark測試指令碼如下所示。

```
/opt/apps/spark-1.6.1-bin-hadoop2.7/bin/spark-submit  --master yarn
 --deploy-mode cluster --executor-memory 3G --num-executors 30
 --conf spark.default.parallelism=800   --class  com.github.ehiggs
.spark.terasort.TeraSort  spark-terasort-1.0-jar-with-dependencies.
jar /data/teragen_100g /data/terasort_out_100g
```

- 測試結果

  - 性能

  - 成本

  - 時間

- 結果分析

  從性能圖上看，EMR+OSS相較於ECS自建Hadoop，有如下優勢：

  - 整體的負載更低。
  - 記憶體利用率基本一樣。
  - CPU使用低，其中iowait與sys低很多。因為ECS自建有datanode及磁碟操作，需要佔一些資源，增加CPU的開銷。
  - 從網路看，因為sortbenchmark有兩次讀取數據，第一次是採樣，第二次是真正的讀取數據，開始網路比較高，隨後shuffle+輸出結果階段，網路比ECS自建Hadoop低一半左右。因此從網路來看，整體使用量基本持平。

  綜上所述，EMR+OSS比自建ECS使用更少的資源，成本節約了一半，但是性能下降基本可以忽略不計。並且，如果提高EMR+OSS的並發度，則時間上有可能比ECS自建Hadoop叢集更有優勢。

不適用的場景

以下場景不建議使用EMR+OSS：

- 小檔案過多的場景。

  如果檔案小於10M時，請合并小檔案。當數據量在128M以上時，使用EMR+OSS的性能最佳。
- 頻繁操作OSS元數據的場景。

4.3 使用OSS中的數據作為機器學習服務的訓練樣本

背景

- 對象儲存服務OSS

  阿里雲對象儲存服務（Object Storage Service，以下簡稱 OSS），是阿里雲提供的海量、安全、低成本、高可靠的雲端儲存體服務。
- 雲機器學習服務平台PAI

  阿里雲機器學習服務PAI（Platform of Artificial Intelligence，以下簡稱PAI）是一款一站式的機器學習服務平台，包含大量封裝的演算法組件和視覺化檢視，用戶上手容易。
- 業務場景

  本文通過OSS與PAI的結合，為一家傳統的文具零售店提供決策支援。本文涉及的具體業務場景（場景與數據均為虛擬）如下：

  一家傳統的線下文具零售店，希望通過資料採礦尋找強相關的文具品類，幫助合理調整文具店的貨架布局。但由於收銀裝置陳舊，是一台使用XP系統的POS收銀機，可用的銷售數據僅有一份

從POS收銀機匯出的訂單記錄（csv格式）。本文介紹如何將此csv檔案匯入OSS，並連通OSS與PAI，實現商品的關聯推薦。

步驟

1. 數據匯入OSS
   a. 新建一個名為“oss-pai-sample”的Bucket。
   b. 記錄其endpoint為 oss-cn-shanghai.aliyuncs.com。
   c. 選擇儲存類型為標準儲存。

      

      说明:
      OSS中有三種儲存類型，相關介紹請參見儲存類型介紹。
   d. 單擊Bucket名稱（oss-pai-sample），然後依次單擊檔案管理 > 上傳檔案，將訂單數據Sample_superstore.csv上傳至OSS。
   e. 上傳成功，介面如下圖所示：
2. 建立機器學習服務項目
   a. 在控制台頁面左側選擇機器學習服務，單擊右上方的建立項目。
   b. 在顯示的DataWorks新用戶引導介面中，勾選region（本文中選擇與OSS相同的region：華東2），並勾選計算引擎服務機器學習服務PAI，然後單擊下一步。
   c. 項目建立成功後，開通服務列中會顯示MaxCompute和機器學習服務PAI兩個表徵圖，如下圖所示：
   d. 回到機器學習服務頁面，點擊進入機器學習服務。

3. 連通OSS與PAI

a. 在機器學習服務介面左側選擇組件，並將OSS資料同步組件拖拽至畫布。

介面右側會提示填入組件需要的以下資訊：

- OSS endpoint：根據步驟一中記錄的資訊，endpoint 為 oss-cn-shanghai.aliyuncs.com。
- OSSaccessId 和 OSSaccessKey 可以在Object Storage Service的介面中獲取，如下圖所示：
- OSSbucket 和 object 分別為 oss-pai-sample 和 Sample_superstore.csv。
- OSScolumn 映射的作用是為OSS中的csv檔案增加列名。例如，虛擬數據 Sample_superstore.csv共有如下6列：

則OSScolumn映射應該填入：0:order_id,1:order_date,2:customer_id,3:item,4:sales,5:quantity

b. 單擊運行，成功後右鍵查看組件，可觀察前100條數據，如下圖所示：

此時OSS中的csv檔案已經在MaxCompute中生成一張暫存資料表：pai_temp_116611_1297076_1

至此，本案例最關鍵的步驟已經完成，OSS中的數據已經與PAI連通，可以作為機器學習服務的樣本進行訓練。

數據探索流程

本案例中的數據探索流程如下：

本案例按8:2的比例將來源資料拆分為訓練集和測試集，其中一個訂單中可能有多個item，故ID列選擇order_id，保證含有多個item的訂單不會被拆分，如下圖所示：

本案例中共有17個產品item。通過協同過濾演算法組件，取相似性最高的item，結果如下表：

結論

通過機器學習服務，我們發現“紙張”與“訂書器”二者的相似性較高，且與其它產品也有較高的相似性。

對於這家文具零售店來說，根據此數據發現可以有兩種布局貨架的方式：

- 紙張和訂書器貨架放在最中間，其它產品貨架呈環形圍繞二者擺放，這樣無論顧客從哪個貨架步入，都可以快速找到關聯程度較高的紙張和訂書器。
- 將紙張和訂書器兩個貨架分別擺放在文具店的兩端，顧客需要橫穿整個文具店才可以購買到另外一樣，中途路過其他產品的貨架可以提高交叉購買率。當然，此布局方式犧牲了用戶購物的便利性，實際操作中應保持慎重。

4.4 DataLakeAnalytics+OSS：基於OSS的Severless的互動式查詢分析

背景

本文以基金交易資料處理為例，介紹將資料存放區在OSS，使用DataLakeAnalytics進行Severless的互動式查詢分析。

Object Storage Service

Object Storage Service提供標準、低頻、Archive Storage類型，能夠覆蓋從熱到冷的不同儲存場景。同時，OSS能夠與Hadoop開源社區及EMR、批次運算、MaxCompute、機器學習服務PAI、DatalakeAnalytics、Function Compute等阿里雲計算產品進行深度結合。

用戶可以打造基於OSS的資料分析應用，如MapReduce、HIVE/Pig/Spark等批處理（如日誌離線計算）、互動式查詢分析（Imapla、Presto、DataLakeAnalytics）、深度學習訓練（阿里雲PAI）、基因渲染計算交付（批次運算）、大數據應用（MaxCompute）及串流（Function Compute）等。

DataLakeAnalytics

Data Lake Analytics是無伺服器（Serverless）化的雲上互動式查詢分析服務。無需ETL，就可通過此服務在雲上通過標準JDBC直接對阿里雲OSS、TableStore的數據輕鬆進行查詢和分析，以及無縫整合商務分析工具。

服務開通

- **OSS服務：**
- **DataLakeAnalytics服務：**

數據匯出到OSS

以基金交易數據為例：

假設在OSS上儲存了以下trade\user檔案夾，並儲存相應的交易數據和開戶資訊：

下載類比資料（該數據為以下實驗中使用的類比資料。）

登入Data Lake Analytics控制台

點擊登入資料庫，使用方法可以參考 DataLakeAnalytics 產品幫助文檔。

建立Schema和Table

- **建立Schema**

  輸入建立SCHEMA的語句，點擊同步執行。

  ```
  CREATE SCHEMA sh_trade
  ```

  CREATE SCHEMA sh_trade（注意：同一個阿里雲region，schema名全域唯一，建議根據業務定義。如有重名schema，在建立時會提示報錯，則需換另一個schema名。）

- **建立Table**

  在資料庫的下拉框中，選擇剛剛建立的schema。

  说明:
  建立交易記錄表及建立開戶資訊表時：

  - **LOCATION ‘oss://Bucket名稱/交易記錄表目錄/‘**
  - **Location替換為您的Bucket和測試數據的路徑**

  - **建立交易記錄表：**

    在SQL文字框中輸入建表語句如下：

    ```
    CREATE EXTERNAL TABLE tradelist_csv (
        t_userid STRING COMMENT '用戶ID',
        t_dealdate STRING COMMENT '申請時間',
        t_businflag STRING COMMENT '業務代碼',
        t_cdate STRING COMMENT '確認日期',
        t_date STRING COMMENT '申請日期',
        t_serialno STRING COMMENT'申請序號',
        t_agencyno STRING COMMENT'銷售商編號',
        t_netno STRING  COMMENT'網點編號',
        t_fundacco STRING COMMENT'基金帳號',
        t_tradeacco STRING COMMENT'交易帳號',
        t_fundcode STRING  COMMENT'基金代碼',
        t_sharetype STRING COMMENT'份額類別',
        t_confirmbalance DOUBLE  COMMENT'確認金額',
        t_tradefare DOUBLE COMMENT'交易費',
        t_backfare DOUBLE COMMENT'後收手續費',
        t_otherfare1 DOUBLE COMMENT'其他費用1',
        t_remark STRING COMMENT'備忘'
        )
        ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
        STORED AS TEXTFILE
        LOCATION 'oss://testdatasample/workshop_sh/trade/';
    ```

  - **建立開戶資訊表：**

    ```
    CREATE EXTERNAL TABLE userinfo (
        u_userid STRING COMMENT '用戶ID',
        u_accountdate STRING COMMENT '開戶時間',
    ```

```
    u_gender STRING COMMENT '性別',
    u_age INT COMMENT '年齡',
    u_risk_tolerance INT COMMENT '風險承受能力，1-10，10為最進階',
    u_city STRING COMMENT'所在城市',
    u_job STRING COMMENT'工作類別，  A-K',
    u_income DOUBLE  COMMENT'年度營收(萬)'
    )
    ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
    STORED AS TEXTFILE
    LOCATION 'oss://testdatasample/workshop_sh/user/';
```

建表完畢後，刷新頁面，在左邊導航條中能看到schema下的2張表。

SQL查詢(同步執行)

- 查詢交易機構SXS_0010，在0603至0604的100條交易記錄

  1. 查詢SQL

```
SELECT * FROM tradelist_csv
WHERE t_cdate >= '2018-06-03' and t_cdate <= '2018-06-04' and
t_agencyno = 'SXS_0010'
limit 100;
```

  2. 顯示執行結果

- 查詢各城市、男性女性人群，購買的基金總額（多表Join查詢）

  1. 查詢SQL

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

  2. 顯示執行結果

SQL查詢(非同步執行)

1. 非同步執行查詢，將查詢結果以CSV格式輸出到OSS。

2. 點擊執行狀態，可查看該非同步查詢任務的執行狀態。

   

   说明:

   執行狀態主要分為RUNNING、SUCCESS 和 FAILURE三種。點擊刷新，當STATUS變為SUCCESS時，可以查看查詢結果輸出到OSS的檔案路徑。

3. 查看匯出OSS的結果檔案。

5 資料備份

5.1 備份儲存空間

針對存放在OSS上的數據，阿里雲提供多種資料備份方式，以滿足不同場景的備份需求。

OSS雲上備份方式有如下2種：

- **跨區域複製（提供控制台配置操作以及基於API/SDK兩種模式）**
- **基於OssImport工具**

通過跨區域複製進行備份

使用場景

參見管理跨區域複製開發指南。

控制台設定作業

參見設定跨區域複製操作指南。

常見問題

參見*Bucket*之間資料同步常見問題。

特別說明

- **源Bucket和目標Bucket屬於同一個用戶，且分屬不同的區域。**
- **源Bucket、目標Bucket儲存類型都不是歸檔類型。**
- **若要在同一區域的Bucket之間進行資料同步，則可以通過OSS的SDK/API編碼實現。**

通過使用OssImport工具進行備份

OssImport工具可以將本地、其它雲端儲存的資料移轉到OSS，它有以下特點：

- **支援豐富的資料來源，有本地、七牛、百度BOS、AWS S3、Azure Blob、又拍雲、騰訊雲COS、金山KS3、HTTP、OSS等，並可根據需要擴充。**
- **支援斷點續傳。**
- **支援流量控制。**
- **支援遷移指定時間後的檔案、特定首碼的檔案。**
- **支援並行數據下載、上傳。**
- **支援單機模式和分布式模式。單機模式部署簡單使用方便，分布式模式適合大規模資料移轉**

使用場景

參見資料移轉。

安裝部署

參見說明及配置、 單機部署、 分布式部署。

常見問題

參見常見問題。

特別說明

- 不同賬戶的不同Bucket之間，數據量很大，10TB等級以上的情況，建議使用分布式的版本。
- 利用增量模式在OSS之間同步檔案修改操作，OssImport只能同步檔案的修改操作(put/apppend/multipart)，不能同步讀取和刪除操作，資料同步的及時性沒有具體的 SLA 保證，慎重選擇。
- 如果開通了跨區域複製的地域，則推薦使用跨區域複製進行地域之間的資料同步。

6 儲存空間管理

6.1 防盜鏈

背景

A是某一網站站長，A網站中的圖片和音頻視頻連結等靜態資源都保存在阿里雲*Object Storage Service*上。以圖片為例，A在OSS上存放的URL為`http://referer-test.oss-cn-hangzhou.aliyuncs.com/aliyun-logo.png`

OSS資源外鏈地址見*OSS* 地址，這樣的URL（不帶簽名）要求用戶的Bucket許可權為公開讀許可權。

B是另一網站的站長，B在未經A允許的情況下使用A網站的圖片資源，放置在自己網站的網頁中，通過這種方法盜取空間和流量。這樣的情況下，第三方網站用戶看到的是B網站，但並不清楚網站裡的圖片來源。由於OSS是按使用量來收費，這樣用戶A在沒有獲取任何收益的情況下，反而承擔了資源使用費用。

本文適用於在網頁中使用了OSS資源作為外鏈的用戶，並介紹類似A用戶將資源存放在OSS上後，如何通過設定防盜鏈的方法避免承擔不必要的資源使用費用。

實現方法

目前OSS提供的防盜鏈方法主要有以下兩種：

- 設定Referer。該操作通過控制台和SDK均可進行，用戶可根據自身需求進行選擇。
- 簽名URL，適合習慣開發的用戶。

本文將提供如下兩個樣本：

- 通過控制台設定Referer防盜鏈
- 基於PHP SDK動態生成簽名URL防盜鏈

設定Referer

該部分主要說明什麼是Referer，以及OSS如何利用Referer做防盜鏈。

- Referer是什麼

  Referer是HTTP Header的一部分，當瀏覽器向網站Web伺服器發送請求的時候，通常會帶上Referer，告訴伺服器此次請求的連結來源。從以上樣本來看，假如用戶B的網站為

userdomain-steal，想盜鏈用戶A的圖片連結`http://referer-test.oss-cn-hangzhou.aliyuncs.com/aliyun-logo.png`**。A的網站網域名稱為**`userdomain`。

假設盜鏈網站userdomain-steal的網頁如下：

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

假設源站為userdomain的網頁如下：

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

- **當互連網用戶用瀏覽器訪問B的網站頁面**`http://userdomain-steal/index.html`**，網頁裡連結是A的網站的圖片。由於從一個網域名稱(userdomain-steal)請求跳到了另一個網域名稱(oss-cn-hangzhou.aliyuncs.com)，瀏覽器就會在HTTP請求的Header中帶上Referer，如圖所示：**

  可以看到瀏覽器在HTTP請求中的Referer為`http://userdomain-steal/index.html`**。本文主要是使用chrome的開發人員模式來查看網頁請求的，如圖：**
- **同樣瀏覽器訪問**`http://userdomain/error.html`**，也可以看到瀏覽器的Referer為**`http://userdomain/error.html`。
- **如果瀏覽器直接輸入地址，可以看到，請求中的Referer為空。**

  如果A沒有在OSS進行任何Referer相關設定，以上三種情況都是可以訪問用戶A的圖片連結。

- **OSS通過Referer防盜鏈的原理**

  由此可見，瀏覽器在請求OSS資源時，如果發生頁面跳轉，瀏覽器會在請求中帶入Referer，此時Referer的值為上一頁面的URL，有的時候Referer也會為空。

  針對這兩種情況，OSS的Referer功能提供兩種選擇：

  - **設定是否允許空Referer訪問。不能單獨設定，需要配合Referer白名單一起使用。**
  - **設定Referer白名單。**

  細節分析如下：

  - **用戶只有通過簽名URL或者匿名訪問object時，才會做防盜鏈驗證。請求的Header中有“Authorization”欄位的，不會做防盜鏈驗證。**
  - **一個Bucket可以支援多個Referer參數。**
  - **Referer參數支援萬用字元“*”和“？”。**
  - **用戶可以設定允許空Referer的請求訪問。**
  - **白名單為空時，不會檢查Referer欄位是否為空（不然所有的請求都會被拒絕，因為空Referer會被拒絕，對於非空Referer OSS在Referer白名單裡也找不到）。**
  - **白名單不為空，且設定了”不允許Referer欄位為空”的規則。則只有Referer屬於白名單的請求被允許，其他請求（包括Referer為空的請求）會被拒絕。**
  - **白名單不為空，但設定了”允許Referer欄位為空”的規則。則Referer為空的請求和符合白名單的請求會被允許，其他請求都會被拒絕。**
  - **Bucket的三種許可權（private，public-read，public-read-write）都會檢查Referer欄位。**

  萬用字元詳解：

  - **星號“*”：可以使用星號代替0個或多個字元。如果正在尋找以AEW開頭的一個檔案，但不記得檔案名其餘部分，可以輸入AEW，尋找以*AEW*開頭的所有檔案類型的檔案，如*AEWT.txt*、*AEWU.EXE*、*AEWI.dll*等。要縮小範圍可以輸入*AEW*.txt，尋找以AEW開頭的所有檔案類型並.txt為副檔名的檔案如AEWIP.txt、AEWDF.txt。**
  - **問號“?”：可以使用問號代替一個字元。如果輸入love?，尋找以love開頭的一個字元結尾檔案類型的檔案，如lovey、lovei等。要縮小範圍可以輸入love?.doc，尋找以love開頭的一個字元結尾檔案類型並.doc為副檔名的檔案如lovey.doc、loveh.doc。**

- 不同的Referer設定和防盜鏈效果

  Referer 設定的效果如下：

  - 只設定“不允許referer為空”

    從控制台中來設定不允許referer為空，如圖所示：

    直接存取：發現可以訪問，是防盜鏈失效了嗎？不是的，因為”白名單為空時，不會檢查Referer欄位是否為空”，所以白名單為空的時候，這個設定無效。因此需要設定Referer白名單。
  - 設定“不允許referer為空”的同時也設定Referer白名單

    從前面例子中我們可以看到在瀏覽器的請求中Referer為當前頁面的URL，所以需要知道網站會從哪幾個URL跳轉過來，然後進行配置。

    Referfer白名單的設定規則：

    ■ 例子中的Referer為`http://userdomain/error.html`，所以Referer白名單可以設定為`http://userdomain/error.html`，但由於OSS的Referer檢查是通過首碼匹配的，假如有其他網頁比如`http://userdomain/index.html`就訪問不了，所以Referer白名單可以配置成`http://userdomain/`。

    ■ 假如還有其他網域名稱比如`http://img.userdomain/index.html`也需要訪問，那麼Referer白名單應該加上`http://*.userdomain/`。

    這裡兩個都配置，如圖所示：

    做測試可以得到如下結果：

瀏覽器輸入	預期	結果
http://referer-test.oss-cn-hangzhou.aliyuncs.com/aliyun-logo.png	直接存取，Referer為空，預期：不允許空Referer的請求，返回403。	符合預期
http://userdomain/error.html	請求來自於源站，預期：訪問成功。	符合預期
http://userdomain-steal/index.html	請求來自於盜鏈網站，預期：OSS返回403，防盜鏈成功。	符合預期

瀏覽器輸入	預期	結果
http://img.userdomain/error.html	請求來自於源站第三層網域名。	符合預期

> **说明:**
>
> ■ 測試中提到的網域名稱是為了測試而假設的，和您實際的網域名稱不一樣，請注意區分。
>
> ■ 如果Referer白名單只有`http://userdomain/`，瀏覽器類比第三層網域名訪問`http://img.userdomain/error.html`的時候，第三層網域名無法匹配Referer白名單，OSS就會返回403。

- 設定“允許referer為空”的同時也設定Referer白名單

  Referer白名單有`http://*.userdomain/`和`http://userdomain`。

  如圖所示：

  測試得出以下結果：

瀏覽器輸入	預期	結果
http://referer-test.oss-cn-hangzhou.aliyuncs.com/aliyun-logo.png	直接存取，Referer為空，預期：訪問成功。	符合預期
http://userdomain/error.html	請求來自於源站，預期：訪問成功。	符合預期
http://userdomain-steal/index.html	請求來自於盜鏈網站，預期：OSS返回403，防盜鏈成功。	符合預期
http://img.userdomain/error.html	請求來自於源站第三層網域名。	符合預期

- 如何在OSS裡設定Referer

  功能使用參考：

  - API：*Put Bucket Referer*
  - 控制台：**防盜鏈設定**

- Referer防盜鏈的缺點

  Referer防盜鏈的優點是設定簡單，控制台即可操作。最大的缺點就是無法防止惡意偽造Referer，如果盜鏈是通過應用程式類比HTTP請求，偽造Referer，則會繞過用戶防盜鏈設定。如果對防盜鏈有更高要求，請參考以下簽名URL防盜鏈的描述。

簽名URL

簽名URL的原理和實現方法見OSS開發人員指南授權第三方下載。簽名URL的實現步驟如下：

1. 將Bucket的使用權限設定為私有讀。
2. 根據期望的逾時時間（簽名URL失效的時間）生成簽名。

具體實現方法如下：

1. 安裝PHP最新代碼，參考*PHP SDK*文檔。
2. 實現生成簽名URL並將其放在網頁中，作為外鏈使用的簡單樣本如下：

```
<?php
 require 'vendor/autoload.php';
 #最新PHP提供的自動載入
 use OSS\OssClient;
 #表示命名空間的使用
 $accessKeyId="a5etodit71tlznjt3pdx****";
 #AccessKeyId，需要使用用戶自己的
 $accessKeySecret="secret_key";
 #AccessKeySecret，需要用用戶自己的
 $endpoint="oss-cn-hangzhou.aliyuncs.com";
 #Endpoint，根據Bucket建立的區域來選擇，本文中是杭州
 $bucket = 'referer-test';
 #Bucket，需要用用戶自己的
 $ossClient = new OssClient($accessKeyId, $accessKeySecret, $
endpoint);
 $object = "aliyun-logo.png";
 #需要簽名的Object
 $timeout = 300;
 #期望連結失效的時間，這裡表示從代碼運行到這一行開始的目前時間往後300秒
 $signedUrl = $ossClient->signUrl($bucket, $object, $timeout); #簽名
URL實現的函數
 $img= $signedUrl;
 #將簽名URL動態放到圖片資源中並列印出來
 $my_html = "<html>";
 $my_html .= "<img src=\"".$img. "\" />";
 $my_html .= "<p>".$img."</p>";
 $my_html .= "</html>";
 echo $my_html;
 ?>
```

3. 通過瀏覽器訪問多請求幾次會發現簽名的URL會變，這是正常的。主要是因為過期時間的改變導致的。這個過期時間是連結失效的時間，是以Unix time的形式展示的。例如：Expires=1448991693，可以將這個時間轉換成本地時間。在Linux下的命令為`date -d@1448991693`，也可以在網路上找工具自行轉換。

特別說明

簽名URL可以和Referer白名單功能一起使用。

如果簽名URL失效的時間限制在分鐘內，盜鏈用戶即使偽造了Referer也必須拿到簽名的URL，且必須在有效時間內才能盜鏈成功。相比只使用Referer來說，增加了盜鏈的難度。也就是說簽名URL配合Referer白名單功能，可以增加防盜鏈的效果。

防盜鏈總結

基於OSS的防盜鏈最佳實踐點如下：

- 使用第三層網域名URL，例如`referer-test.oss-cn-hangzhou.aliyuncs.com/aliyun-logo.png`，安全性比綁定次層網域更高。第三層網域名方式能夠提供Bucket等級的清洗和隔離，能夠應對被盜鏈後的流量暴漲的情況，也能避免不同Bucket間的互相影響，最終提高業務可用性。
- 如果使用自訂網域名作為串連，CNAME也請綁定到第三層網域名，規則是bucket + endpoint。假如您的bucket名為test，第三層網域名則為`test.oss-cn-hangzhou.aliyuncs.com`。
- 對Bucket設定儘可能嚴格的權限類別別。例如提供公網服務的Bucket設定為public-read或private，禁止設定為public-read-write。Bucket許可權參見**存取控制**。
- 對訪問來源進行驗證，根據需要設定合適的Referer白名單。
- 如果需要更嚴格的防盜鏈方案，請參考簽名的URL方案。
- 記錄Bucket訪問日誌，能夠及時發現盜鏈活動和驗證防盜鏈方案的有效性。訪問日誌參見**設定訪問日誌記錄**。

常見問題及解決方案

- 在OSS控制台設定了防盜鏈，但一直不生效，頁面可以反而播放器不可以，請問為什麼？怎麼解決？

  目前設定防盜鏈不生效的主要問題集中於視頻和音頻檔案，在使用諸如Windows Media Player、Flash Player等播放器後，在請求OSS資源的時候傳遞的Referer為空，這就造成防盜鏈的失效。針對這種情況，可以參考上面提到的簽名URL防盜鏈的方法。
- Referer是什麼？如果遇到HTTPS怎麼辦？

  Referer是HTTP協議中的要求標頭，在跨頁面訪問的時候會帶上。需要看看瀏覽器請求的Referer是`http://`還是`https://`，通常是`http://`。
- 如何生成簽名URL？AccessKeySecret放在客戶端裡的安全性如何？

  簽名URL的方法參見各個SDK文檔。AccessKeySecret不建議直接放在客戶端，RAM提供了*STS*服務可以解決這個問題，詳情參見*RAM*和*STS*指南。

- 例如要寫a.baidu.com和b.baidu.com，這兩個用萬用字元(*,?) 如何寫?

  可以寫成http://*.baidu.com，對於這種單字元，也可以寫成http://?.baidu.com。

- *.domain.com 可以匹配次層網域，但無法匹配 domain.com，另外添加一行 domain.com 也沒效果，如何配置?

  注意一般的referer中會帶http這樣的參數，可以通過chrome的開發人員模式觀察下請求的Referer是什麼，然後再具體設定。這裡可能是忘了寫http://，應該為http://domain.com。

- 如果防盜鏈沒有生效怎麼辦?

  推薦使用chrome來查看。開啟開發人員模式，點擊網頁，查看HTTP請求中的Referer具體值。並確認是否與OSS中設定的Referer匹配。如果還是解決不了，請提工單。

6.2 跨域資源共用（CORS）

同源策略

跨域訪問，或者說 JavaScript 的跨域訪問問題，是瀏覽器出於安全考慮而設定的一個限制，即同源策略。舉例說明，當 A，B 兩個網站屬於不同的域時，如果來自於 A 網站的頁面中的 JavaScript 代碼希望訪問 B 網站的時候，瀏覽器會拒絕該訪問。

然而，在實際應用中，經常會有跨域訪問的需求。比如用戶的網站 www.a.com，後端使用了 OSS，在網頁中提供了使用 JavaScript 實現的上傳功能，但是在該頁面中，只能向 www.a.com 發送請求，向其他網站發送的請求都會被瀏覽器拒絕。這樣會導致用戶上傳的數據必須從www.a.com 中轉。如果設定了跨域訪問的話，用戶就可以直接上傳到 OSS 而無需從 www.a.com 中轉。

CORS 介紹

跨域資源共用（Cross-Origin Resource Sharing，簡稱 CORS），是 HTML5 提供的標準跨域解決方案，具體的CORS規則可以參考*W3C CORS*規範。

CORS 是一個由瀏覽器共同遵循的一套控制策略，通過HTTP的Header來進行互動。瀏覽器在識別到發起的請求是跨域請求的時候，會將Origin的Header加入HTTP請求發送給伺服器，比如上面那個例子，Origin的Header就是www.a.com。伺服器端接收到這個請求之後，會根據一定的規則判斷是否允許該來源域的請求，如果允許的話，伺服器在返回的響應中會附帶上Access-Control-Allow-Origin這個Header，內容為www.a.com來表示允許該次跨域訪問。如果伺服器允許所有的跨域請求的話，將Access-Control-Allow-Origin的Header設定為*即可，瀏覽器根據是否返回了對應的Header來決定該跨域請求是否成功，如果沒有附加對應的Header，瀏覽器將會攔截該請求。

以上描述的僅僅是簡單情況，CORS規範將請求分為兩種類型，一種是簡單請求，另外一種是帶預檢的請求。預檢機制是一種保護機制，防止資源被本來沒有許可權的請求修改。瀏覽器會在發送實際請求之前先發送一個OPTIONS的HTTP請求來判斷伺服器是否能接受該跨域請求。如果不能接受的話，瀏覽器會直接阻止接下來實際請求的發生。

只有同時滿足以下兩個條件才不需要發送預檢請求：

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

預檢請求會附帶一些關於接下來的請求的資訊給伺服器，主要有以下幾種：

- Origin：請求的源域資訊
- Access-Control-Request-Method：接下來的請求類型，如POST、GET等
- Access-Control-Request-Headers：接下來的請求中包含的用戶顯式設定的Header列表

伺服器端收到請求之後，會根據附帶的資訊來判斷是否允許該跨域請求，資訊返回同樣是通過Header完成的：

- Access-Control-Allow-Origin：允許跨域的Origin列表
- Access-Control-Allow-Methods：允許跨域的方法列表
- Access-Control-Allow-Headers：允許跨域的Header列表
- Access-Control-Expose-Headers：允許暴露給JavaScript代碼的Header列表
- Access-Control-Max-Age：最大的瀏覽器緩存時間，單位為s

瀏覽器會根據以上的返回資訊綜合判斷是否繼續發送實際請求。當然，如果沒有這些Header瀏覽器會直接阻止接下來的請求。

 说明：

這裡需要再次強調的一點是，以上行為都是瀏覽器自動完成的，用戶無需關注這些細節。如果伺服器配置正確，那麼和正常非跨域請求是一樣的。

CORS主要使用場景

CORS使用一定是在使用瀏覽器的情況下，因為控制存取權限的是瀏覽器而非伺服器。因此使用其它的客戶端的時候無需關心任何跨域問題。

使用CORS的主要應用就是在瀏覽器端使用Ajax直接存取OSS的數據，而無需走用戶本身的應用伺服器中轉。無論上傳或者下載。對於同時使用OSS和使用Ajax技術的網站來說，都建議使用CORS來實現與OSS的直接通訊。

OSS對CORS的支援

OSS提供了CORS規則的配置從而根據需求允許或者拒絕相應的跨域請求。該規則是配置在Bucket等級的。詳情可以參考*PutBucketCORS*。

CORS請求的通過與否和OSS的身分識別驗證等是完全獨立的，即OSS的CORS規則僅僅是用來決定是否附加CORS相關的Header的一個規則。是否攔截該請求完全由瀏覽器決定。

使用跨域請求的時候需要關注瀏覽器是否開啟了Cache功能。當運行在同一個瀏覽器上分別來源於www.a.com和www.b.com的兩個頁面都同時請求同一個跨域資源的時候，如果www.a.com的請求先到達伺服器，伺服器將資源帶上Access-Control-Allow-Origin的Header為www.a.com返回給用戶。這個時候www.b.com又發起了請求，瀏覽器會將Cache的上一次請求返回給用戶，這個時候Header的內容和CORS的要求不匹配，就會導致後面的請求失敗。

說明:

目前OSS的所有Object相關的介面都提供了CORS相關的驗證，另外Multipart相關的介面目前也已經完全支援CORS驗證。

GET請求跨域樣本

這裡舉一個使用Ajax從OSS獲取數據的例子。為了簡單起見，使用的Bucket都是public，訪問私有的Bucket的CORS配置是完全一樣的，只需要在請求中附加簽名即可。

簡單開始

首先建立一個Bucket，這裡舉例為oss-cors-test，將使用權限設定為public-read，然後這裡建立一個test.txt的文字文件，上傳到這個Bucket。

單擊**此處**，獲取test.txt的訪問地址。

說明:

請將下文中的地址換成自己實驗的地址。

首先用curl直接存取：

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

可見現在已經能正常訪問了。

那麼我們現在再試著使用Ajax技術來直接存取這個網站。我們寫一個最簡單的訪問的HTML，可以將以下代碼複製到本地保存成html檔案然後使用瀏覽器開啟。因為沒有設定自訂的頭，因此這個請求是不需要預檢的。

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

```
</html>
```

開啟檔案後點選連結（以Chrome為例），提示該連結無法訪問。

利用Chrome的開發人員工具來查看錯誤原因。

這裡告訴我們是因為沒有找到Access-Control-Allow-Origin這個頭。顯然，這就是因為伺服器沒有配置CORS。

再進入Header介面，可見瀏覽器發送了帶Origin的Request，因此是一個跨域請求，在Chrome上因為是本地檔案所以Origin是null。

設定 Bucket CORS知道了上文的問題，那麼現在可以設定Bucket相關的CORS來使上文的例子能夠執行成功。為了直觀起見，這裡使用控制台來完成CORS的設定。如果用戶的CORS設定不是很複雜，也建議使用控制台來完成CORS設定。CORS設定的介面如下：

CORS設定是由一條一條規則群組成的，真正匹配的時候會從第一條開始逐條匹配，以最早匹配上的規則為準。現在添加第一條規則，使用最寬鬆的配置：

這裡表示的意思就是所有的Origin都允許訪問，所有的請求類型都允許訪問，所有的Header都允許，最大的緩存時間為1s。

配置完成之後重新測試，結果如下：

可見現在已經可以正常訪問請求了。

因此排查問題來說，如果想要排除跨域帶來的訪問問題，可以將CORS設定為上面這種配置。這種配置是允許所有的跨域請求的一種配置，因此如果這種配置下依然出錯，那麼就表明錯誤出現在其他的部分而不是CORS。

除了最寬鬆的配置之外，還可以配置更精細的控制機制來實現針對性的控制。比如對於這個場景可以使用如下最小的配置匹配成功：

因此對於大部分場景來說，用戶最好根據自己的使用場景來使用最小的配置以保證安全性。

利用跨域實現POST上傳

這裡提供一個更複雜的例子，這裡使用了帶簽名的POST請求，而且需要發送預檢請求。詳情請參見*PostObjectSample*。

说明:

將上面的代碼下載下來之後，將以下對應的部分全部修改成自己對應的內容，然後使用自己的伺服器運行起來。

這裡繼續使用上文oss-cors-test這個Bucket來進行測試，在測試之前先刪除所有的CORS相關的規則，恢複初始狀態。

訪問這個網頁，選擇一個檔案上傳。

同樣開啟開發人員工具，可以看到以下內容，根據上面的Get的例子，很容易明白同樣發生了跨域的錯誤。不過這裡與Get請求的區別在於這是一個帶預檢的請求，所以可以從圖中看到是OPTIONS的Response沒有攜帶CORS相關的Header然後失敗了。

那麼我們根據對應的資訊修改Bucket的CORS配置：

再重新運行，可見已經成功了，從控制台中也可以看到新上傳的檔案。

測試一下內容：

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

CORS配置的一些注意事項

CORS配置一共有以下幾項：

- 來源：配置的時候要帶上完整的域資訊，比如樣本中 `http://10.101.172.96:8001`.

  注意不要遺漏了協議名如http，如果連接埠號碼不是預設的還要帶上連接埠號碼之類的。如果不確定的話，可以開啟瀏覽器的調試功能查看Origin頭。這一項支援使用萬用字元*，但是只支援一個，可以根據實際需要靈活配置。
- Method：按照需求開通對應的方法即可。
- Allow Header：允許通過的Header列表，因為這裡容易遺漏Header，因此建議沒有特殊需求的情況下設定為*。大小寫不敏感。
- Expose Header：暴露給瀏覽器的Header列表，不允許使用萬用字元。具體的配置需要根據應用的需求來選擇，只暴露需要使用的Header，如ETag等。如果不需要暴露這些資訊，這裡可以不填。如果有特殊需求可以單獨指定，大小寫不敏感。
- 緩存時間：如果沒有特殊情況可以酌情設定的大一點，比如60s。

因此，CORS的一般配置方法就是針對每個可能的訪問來源單獨配置規則，盡量不要將多個來源混在一個規則中，多個規則之間最好不要有覆蓋衝突。其他的許可權只開放需要的許可權即可。

一些錯誤排查經驗

在調試類似程式的時候，很容易發生一種情況就是將其他錯誤和CORS錯誤弄混淆了。

舉個例子，當用戶因為簽名錯誤訪問被拒絕的時候，因為許可權驗證發生在CORS驗證之前，因此返回的結果中並沒有帶上CORS的Header資訊，有些瀏覽器就會直接報CORS出錯，但是實際伺服器端的CORS配置是正確的。對於這種情況，我們有兩種方式：

- 查看HTTP請求的傳回值，因為CORS驗證是一個獨立的驗證，並不影響主幹流程，因此像403之類的傳回值不可能是由CORS導致的，需要先排除程式原因。如果之前發送了預檢請求，那麼可以查看預檢請求的結果，如果預檢請求中返回了正確的CORS相關的Header，那麼表示真實請求應該是被伺服器端所允許的，因此錯誤只可能出現在其他的方面。
- 將伺服器端的CORS設定按照如上文所示，改成允許所有來源所有類型的請求都通過的配置，再重新驗證。如果還是驗證失敗，表明有其他的錯誤發生。

6.3 靜態網站託管

用戶可以基於OSS搭建一個靜態網站。本文介紹了如何從申請網域名稱開始，基於OSS搭建一個簡單的靜態網站。主要的步驟是：

1. 申請一個網域名稱。
2. 開通OSS並建立Bucket。
3. 開通OSS的靜態網站託管功能。
4. 使用自訂網域名訪問OSS。

靜態網站託管功能介紹

簡單說就是用戶可以基於OSS搭建一個簡單的靜態網頁。用戶開啟此功能後，OSS提供了一個預設的首頁和預設的404頁面功能。具體參見開發人員指南中靜態網站託管的介紹。

具體實現步驟

1. 申請網域名稱
2. 開通OSS並建立Bucket
   a. 登入OSS控制台，建立一個Bucket為imgleo23，建立在上海，Endpoint為`oss-cn-shanghai.aliyuncs.com`。操作步驟請參見建立儲存空間。
   b. 將Bucket的使用權限設定為公共讀。操作步驟請參見設定儲存空間讀寫權限。
   c. 上傳index.html、error.html、aliyun-logo.png 檔案。操作步驟請參見上傳檔案。
      - index.html 的內容為：

```
<html>
  <head>
      <title>Hello OSS!</title>
      <meta charset="utf-8">
  </head>
  <body>
      <p>歡迎使用OSS靜態網站的功能</p>
      <p>這是首頁</p>
  </body>
</html>
```

      - error.html 的內容為：

```
<html>
  <head>
      <title>Hello OSS!</title>
      <meta charset="utf-8">
  </head>
  <body>
      <p>這是OSS靜態網站託管的錯誤首頁</p>
  </body>
</html>
```

      - aliyun-logo.png 是一張圖片。

3. 開通OSS的靜態網站託管功能

   如下圖所示，登入控制台後，將預設首頁設定為上文中的index.html，將預設404頁設定為上文中的error.html。具體操作請參見設定靜態網站託管。

   檢驗靜態網站託管功能，輸入如圖所示的URL地址：

   - 顯示預設的首頁

     可以看到輸入類似URL的時候，會顯示開通時指定的index.html中的內容。
   - 顯示預設的 404 頁

     可以看到輸入的URL沒有對應的檔案時，會顯示開通時指定的error.html中的內容。
   - 顯示正常的檔案

     可以看到輸入的URL有對應的檔案時，會讀取成功。
4. 使用自訂網域名訪問 OSS

   關於如何?自訂網域名訪問 OSS，請參見開發人員指南中的自訂網域名訪問 *OSS*。

   - 顯示預設的首頁
   - 顯示預設的404頁
   - 顯示正常的檔案

常見問題及解決方案

- OSS靜態網站託管對客戶來說有什麼好處?

  在用戶需求比較簡單的時候，且訪問量比較小的時候，可以省掉一台ECS。如果訪問量大一點，可以考慮結合CDN來使用。
- 價格怎麼樣？如何和CDN結合?

  價格可以參考官方網站OSS的價格，CDN的價格也可以參考官方網站CND的價格。和CDN結合的例子可以參考*CDN*加速*OSS*實踐。

- **預設的首頁和預設的404頁面都需要設定嗎?**

  預設首頁需要設定，但預設404頁面可以不用設定。

- **為什麼輸入的URL在瀏覽器上返回403?**

  有可能Bucket的許可權不是公開讀。也有可能是因為欠費被停止使用。

7 數據安全

7.1 通過crc64校驗資料轉送的完整性

背景

數據在客戶端和伺服器之間傳輸時有可能會出錯。OSS現在支援對各種方式上傳的Object返回其crc64值，客戶端可以和本地計算的crc64值做對比，從而完成資料完整性的驗證。

- OSS對新上傳的Object進行crc64的計算，並將結果儲存為Object的元資訊儲存，隨後在返回的response header中增加`x-oss-hash-crc64ecma`頭部，表示其crc64值，該64位CRC根據*ECMA-182*標準計算得出。
- 對於crc64上線之前就已經存在於OSS上的Object，OSS不會對其計算crc64值，所以獲取此類Object時不會返回其crc64值。

操作說明

- Put Object / Append Object / Post Object / Multipart upload part 均會返回對應的crc64值，客戶端可以在上傳完成後拿到伺服器返回的crc64值和本地計算的數值進行校驗。
- Multipart Complete時，如果所有的Part都有crc64值，則會返回整個Object的crc64值；否則，比如有crc64上線之前就已經上傳的Part，則不返回crc64值。
- Get Object / Head Object / Get ObjectMeta 都會返回對應的crc64值（如有）。客戶端可以在Get Object完成後，拿到伺服器返回的crc64值和本地計算的數值進行校驗。

  说明:
  range get請求返回的將會是整個Object的crc64值。
- Copy相關的操作，如Copy Object / Upload Part Copy，新生成的Object/Part不保證具有crc64值。

應用樣本

以下為完整的Python範例程式碼，示範如何基於crc64值驗證資料轉送的完整性。

1. 計算crc64。

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

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

2. **驗證Put Object**。

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

3. **驗證Get Object**。

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

4. **驗證Upload Part 和 Complete**。

```
part_info_list = []
 key = "multipart-key"
 result = bucket.init_multipart_upload(key)
 upload_id = result.upload_id
 part_1 = random_string(1024 * 1024)
 result = bucket.upload_part(key, upload_id, 1, part_1)
 oss_crc64 = result.headers.get('x-oss-hash-crc64ecma', '')
 local_crc64 = str(do_crc64(part_1))
 #check 上傳的 part 1數據是否完整
 check_crc64(local_crc64, oss_crc64, "upload_part object 1")
 part_info_list.append(PartInfo(1, result.etag, len(part_1)))
 part_2 = random_string(1024 * 1024)
 result = bucket.upload_part(key, upload_id, 2, part_2)
 oss_crc64 = result.headers.get('x-oss-hash-crc64ecma', '')
 local_crc64 = str(do_crc64(part_2))
 #check 上傳的 part 2數據是否完整
 check_crc64(local_crc64, oss_crc64, "upload_part object 2")
 part_info_list.append(PartInfo(2, result.etag, len(part_2)))
 result = bucket.complete_multipart_upload(key, upload_id,
part_info_list)
 oss_crc64 = result.headers.get('x-oss-hash-crc64ecma', '')
 local_crc64 = str(do_crc64(part_2, do_crc64(part_1)))
 #check 最終oss上的object和本地檔案是否一致
```

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

OSS SDK支援

部分OSS SDK已經支援上傳、下載使用crc64進行數據校驗，用法見下表中的樣本：

SDK	是否支援CRC	樣本
Java SDK	是	*CRCSample.java*
Python SDK	是	*object_check.py*
PHP SDK	否	無
C# SDK	否	無
C SDK	是	*oss_crc_sample.c*
JavaScript SDK	否	無
Go SDK	是	*crc_test.go*
Ruby SDK	否	無
iOS SDK	否	*OSSCrc64Tests.m*
Android SDK	否	*CRC64Test.java*

7.2 通過伺服器端加密保護資料

使用伺服器端加密方式保護待用資料，即OSS將使用者資料寫入資料中心內的磁碟時，會在對象（Object）層級加密資料，並且在訪問這些資料時自動解密，使用者只需要在請求時驗證是否擁有存取權限。

说明:
關於伺服器端加密的詳細介紹可參考開發指南文檔*#unique_51*。

當前OSS支援如下三種伺服器端加密方式：

- **使用OSS完全託管的伺服器端加密方式（SSE-OSS）**

  您可以在上傳Object或修改Object的meta資訊時，在請求中攜帶`X-OSS-server-side-encryption`並指定其值為AES256，阿里雲OSS伺服器端加密使用AES256加密每個對象。OSS會為每個對象使用不同的密鑰進行加密，作為額外的保護，它將使用定期輪轉的主要金鑰對加密金鑰本身進行加密。

- 使用OSS預設託管的KMS密鑰的伺服器端加密方式（SSE-KMS）

  您可以在上傳Object或修改Object的meta資訊時，在請求中攜帶`X-OSS-server-side-encryption`並指定其值為KMS且不指定具體的CMS ID。OSS將使用預設託管的CMK產生不同的密鑰來加密不同的對象，並且在下載時自動解密。

- 使用使用者指定的KMS密鑰的伺服器端加密方式（SSE-KMS）

  您可以在上傳Object或修改Object的meta資訊時，在請求中攜帶`X-OSS-server-side-encryption`，指定其值為KMS，並指定`X-OSS-server-side-encryption-key-id`為具體的CMK ID。OSS將使用指定的CMK產生不同的密鑰來加密不同的對象，並將加密Object的CMK ID記錄到對象的中繼資料中，因此具有解密許可權的使用者下載對象時會自動解密。您可以使用系統自動產生的密鑰材料，也可以匯入外部金鑰材料。

  > **注意:**
  >
  > - 使用指定的KMS祕密金鑰加密的方式目前處於公測中，請聯絡售後支援人員開通試用。
  > - 同一對象同一時間僅可以使用一種伺服器端加密方式。
  > - 使用KMS祕密金鑰加密時，原資料中用於加密資料的資料密鑰也會被加密，並且作為Object的中繼資料資訊一併儲存。
  > - KMS託管密鑰的伺服器端加密方式僅加密對象資料，不會加密任何對象的中繼資料。
  > - 使用KMS密鑰功能時會產生少量的KMS密鑰API調用費用。
  > - 使用子帳號對資料使用指定的KMS祕密金鑰加密時，子帳號需取得KMS相關許可權，詳情請參考*#unique_52*。

使用OSS完全託管的伺服器端加密方式

1. 登入*OSS*控制台，建立一個Bucket。配置步驟請參考建立*Bucket*。
2. 上傳1個明文對象至OSS，詳情請參考*#unique_53*。
3. 對已上傳的檔案進行加密，Python指令碼如下：

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

# 阿里雲主帳號AccessKey擁有所有API的存取權限，風險很高。強烈建議您建立並使用RAM帳號進行API訪問或日常營運，請登入RAM###建立RAM帳號。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint以香港為例，其它Region請按實際情況填寫。
bucket = oss2.Bucket(auth, 'http://oss-cn-hongkong.aliyuncs.com',
 'test-hongkong-2025')
```

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

4. 驗證加密結果。

   使用*ossutil*工具查看加密前後檔案變化。

   - 加密前：

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

   - 加密後：

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

使用OSS預設託管的KMS密鑰的伺服器端加密方式

1. 登入*OSS*控制台，建立一個Bucket。配置步驟請參考建立*Bucket*。
2. 上傳1個明文對象至OSS，詳情請參考*#unique_53*。
3. 在雲產品管理頁，開通KMS服務。
4. 對已上傳的檔案進行加密，Python指令碼如下：

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

# 阿里雲主帳號AccessKey擁有所有API的存取權限，風險很高。強烈建議您建立並使用
RAM帳號進行API訪問或日常營運，請登入RAM###建立RAM帳號。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>
')
# Endpoint以香港為例，其它Region請按實際情況填寫。
```

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

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

使用使用者指定的KMS密鑰的伺服器端加密方式

1. 登入*OSS*控制台，建立一個Bucket。配置步驟請參考建立*Bucket*。
2. 上傳1個明文對象至OSS，詳情請參考*#unique_53*。
3. 在雲產品管理頁，開通KMS服務。
4. 登入*KMS*管理主控台，單擊建立密鑰，建立一個和Bucket同地區的密鑰。
   - 描述：自訂。
   - 進階選項內，密鑰材料來源選擇：阿里雲KMS。

   说明:

   您也可以匯入外部金鑰，詳情請參考*#unique_55*。
5. 使用指定的CMK ID加密已上傳的對象，Python指令碼如下：

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

# 阿里雲主帳號AccessKey擁有所有API的存取權限，風險很高。強烈建議您建立並使用
RAM帳號進行API訪問或日常營運，請登入RAM###建立RAM帳號。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>
')
# Endpoint以香港為例，其它Region請按實際情況填寫。
bucket = oss2.Bucket(auth, 'http://oss-cn-hongkong.aliyuncs.com',
 'test-hongkong-2025')

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

8 OSS資源的監控與報警

Cloud Monitor服務能夠監控OSS服務資源。藉助Cloud Monitor服務，您可以全面了解您在阿里雲上的資源使用方式、性能和健全狀態。藉助報警服務，您可以及時做出反應，保證應用程式順暢運行。本章介紹如何監控OSS資源、設定報警規則以及自訂監控大盤。

前提條件

- 已開通*OSS*服務。
- 已開通*Cloud Monitor*服務。

監控OSS資源

1. 登入*Cloud Monitor*控制台。
2. 在左側導覽列選擇 雲端服務監控 > Object Storage Service，進入OSS監控頁面，如下圖所示。

   在OSS監控頁面，您可以獲取各類監控數據。

   > 说明:
   > 用戶層級指用戶等級的數據，即該用戶下所有bucket的數據。

設定報警規則

1. 在OSS監控頁面的報警規則頁簽下，單擊建立報警規則。
2. 填寫配置項。

   配置項說明參見管理報警規則。
3. 完成配置後，即生成一條報警規則，您可以使用測試數據來檢測該條規則是否生效，確認能否順利接收報警資訊（郵件、SMS、旺旺、釘釘等）。

自訂監控大盤

您可以參考如下步驟，在Cloud Monitor控制台上自訂配置OSS資源監控圖。

1. 登入*Cloud Monitor*控制台。
2. 在左側導覽列單擊Dashboard。
3. 單擊建立監控大盤。

4. 輸入大盤名稱後，單擊添加圖表。

5. 根據需求完成配置，並單擊發布。

   配置項說明參見監控指標參考手冊。

9 OSS性能與擴充性最佳實踐

分區與命名規範

OSS按照檔案名UTF-8編碼的順序對用戶數據進行自動分區，從而能夠處理海量檔案，以及承載高速率的客戶請求。不過，如果客戶在上傳大量對象時，在命名上使用了順序首碼（如時間戳記或字母順序），可能會導致大量檔案索引集中儲存於某個特定分區。這樣，當用戶的請求速率超過2000操作/秒時（下載、上傳、刪除、拷貝、獲取元數據資訊等操作算1次操作，大量刪除N個檔案、列舉N個檔案等操作算N次操作），會帶來如下後果：

- **該分區成為熱點分區，導致分區的I/O能力被耗盡，或被系統自動限制請求速率。**
- **熱點分區的存在會觸發系統進行持續的分區數據再均衡，這個過程可能會延長請求處理時間。**

從而降低OSS的水平擴充效果，導致客戶的請求速率受限。

要解決這個問題，根本上，要消除檔案名中的順序首碼。我們可以在檔案名首碼中引入某種隨機性，這樣檔案索引（以及 I/O 負載）就會均勻分布在多個分區。

下面提供了幾個將順序首碼改為隨機性首碼的方法案例。

- **樣本 1：向檔案名添加十六進位雜湊首碼**

  如下例所示，用戶使用了日期與客戶ID組建檔案名，包含了順序時間戳記首碼：

```
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
...
```

  針對這種情況，我們可以對客戶ID計算雜湊，即MD5 (customer-id)，並取若干字元的雜湊首碼作為檔案名的首碼。假如取4個字元的雜湊首碼，結果如下所示：

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

  加入4個字元組成的十六進位雜湊作為首碼，每個字元有0-f共16種取值，因此4個字元共有16^4=65536種可能的字元組合。那麼在儲存系統中，這些數據理論上會被持續劃分至最多65536

個分區，以每個分區2000操作/秒的性能瓶頸標準，再結合您的業務的請求速率，以此您可以評估hash桶的個數是否合適。

如果您想要列出檔案名中帶有特定日期的檔案，例如列出sample-bucket-01裡帶有2017-11-11的檔案，您只要對sample-bucket-01進行列舉(即通過多次調用List Object介面，分批次地獲得sample-bucket-01下的所有檔案)，然後合并帶有該日期的檔案即可。

- 樣本 2：反轉檔案名

如下例所示，用戶使用了毫秒精度的UNIX時間戳記組建檔案名，同樣屬於順序首碼：

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

由前面的分析，我們知道，這種順序首碼命名，在請求速率超過一定閾值時，會引發性能問題。我們可以通過反轉時間戳記首碼來避免，這樣檔案名就不包含順序首碼了。反轉後結果如下：

```
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
...
```

由於檔案名中的前3位元字代表毫秒時間，會有1000種取值。而第4位元字，每1秒鐘就會改變一次。同理第5位元字每10秒鐘就會改變一次…以此類推，反轉檔案名後，極大地增強了首碼的隨機性，從而將負載壓力均勻地分攤在各個分區上，避免出現性能瓶頸。

10 安卓應用樣本

10.1 OssDemo簡介

在 OSS 的開發人員指南中介紹了移動端開發上傳場景。本文主要是基於這個場景來介紹在Android上如何使用SDK來實現一些常見的操作，也就是OssDemo這個工程。主要是以下幾個方面：

- 如何使用已經搭建好的應用伺服器（STS）
- 如何使用SDK來上傳檔案
- 如何使用圖片服務

這裡假設您對OSS的移動開發場景有所了解，並且知道STS（Security Token Service）。

準備工作

由於是基於Android的開發，所以需要做一些準備工作。

1. 開通OSS。請參考快速入門。
2. 搭建應用伺服器。請參考快速搭建行動裝置 App 直傳服務。
3. 準備Android開發環境。這裡主要用的是Android Studio，網上有很多教程，這裡就不再重複。
4. *OssDemo*源碼下載後可安裝體驗，後面源碼分析有詳細介紹上面提到的常見操作的實現。
5. 開啟OSS官方提供的*OSS Android SDK*文檔以供參考。

10.2 使用已經搭建好的應用伺服器

本文主要講解OssDemo這樣的移動APP如何使用應用伺服器，以達到不需要在APP端儲存AccessKeyId和AccessKeySecret也能向OSS上傳的目的。

調用邏輯

1. OssDemo在獲取sts_server的地址後，發送請求。
2. sts_server返回AccessKeyId、AccessKeySecret、SecurityToken、Expiration。
3. OssDemo獲取這些資訊後，調用SDK，構建OssClient。

具體代碼

1. 生成一個EditText控制項。

   位置：

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

2. 從應用伺服器獲取STS相關參數的代碼。

   函數實現:

```
OSSFederationToken getFederationToken()
```

3. 調用STS返回參數，初始化OssClient代碼。

   函數實現:

```
//初始化一個OssService用來上傳下載
 public OssService initOSS(String endpoint, String bucket,
ImageDisplayer displayer) {
     //如果希望直接使用accessKey來訪問的時候，可以直接使用OSSPlainTe
xtAKSKCredentialProvider來鑒權。
     //OSSCredentialProvider credentialProvider = new OSSPlainTe
xtAKSKCredentialProvider(accessKeyId, accessKeySecret);
     //使用自己的獲取STSToken的類
     OSSCredentialProvider credentialProvider = new STSGetter(
stsServer);
     ClientConfiguration conf = new ClientConfiguration();
     conf.setConnectionTimeout(15 * 1000); // 連接逾時，預設15秒
     conf.setSocketTimeout(15 * 1000); // socket逾時，預設15秒
     conf.setMaxConcurrentRequest(5); // 最大並發請求書，預設5個
     conf.setMaxErrorRetry(2); // 失敗後最大重試次數，預設2次
     OSS oss = new OSSClient(getApplicationContext(), endpoint,
credentialProvider, conf);
     return new OssService(oss, bucket, displayer);
 }
```

10.3 上傳檔案

簡單上傳

簡單上傳就是調用OSS API中的Put Object介面，一次性將選擇的檔案上傳到OSS上。

調用邏輯

1. 選擇上傳後，可以選擇需要上傳的本地檔案。
2. 選擇後OssDemo在獲取sts_server的地址後，發送請求。
3. sts_server返回AccessKeyId、AccessKeySecret、SecurityToken、Expiration。

4. OssDemo獲取這些資訊後，調用SDK，構建OssClient，進行簡單上傳。

具體代碼

1. 生成一個Button控制項。

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

2. 點擊上傳，選擇要上傳的檔案。

函數實現片段:

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

3. 調用SDK的上傳介面。

函數實現片段:

```
@Override
 protected void onActivityResult(int requestCode, int resultCode,
Intent data) {
     super.onActivityResult(requestCode, resultCode, data);
     if ((requestCode == RESULT_UPLOAD_IMAGE || requestCode ==
RESULT_PAUSEABLEUPLOAD_IMAGE) && resultCode == RESULT_OK && null !=
data) {
         Uri selectedImage = data.getData();
         String[] filePathColumn = { MediaStore.Images.Media.DATA };
         Cursor cursor = getContentResolver().query(selectedImage,
                 filePathColumn, null, null, null);
         cursor.moveToFirst();
         int columnIndex = cursor.getColumnIndex(filePathColumn[0]);
         String picturePath = cursor.getString(columnIndex);
         Log.d("PickPicture", picturePath);
         cursor.close();
         try {
             Bitmap bm = ImageDisplayer.autoResizeFromLocalFile(
picturePath);
             displayImage(bm);
             File file = new File(picturePath);
             displayInfo("檔案: " + picturePath + "\n大小: " + String.
valueOf(file.length()));
         }
```

```
            catch (IOException e) {
                e.printStackTrace();
                displayInfo(e.toString());
            }
            //根據操作不同完成普通上傳或者斷點上傳
            if (requestCode == RESULT_UPLOAD_IMAGE) {
                final EditText editText = (EditText) findViewById(R.id.
edit_text);
                String objectName = editText.getText().toString();
                //調用簡單上傳介面上傳
                ossService.asyncPutImage(objectName, picturePath,
getPutCallback(), new ProgressCallbackFactory<PutObjectRequest>().
get());
            }
        }
    }
```

這裡省略了對上傳結果的處理，可以參考源碼中的onSuccess和onFailure的處理。

基於分區上傳實現的斷點續傳上傳

調用OSS API中的Multipart Upload（分區上傳）介面實現斷點續傳的效果。

調用邏輯

1. **選擇上傳後，可以選擇需要上傳的本地檔案。**
2. **選擇後OssDemo在獲取sts_server的地址後，發送請求。**
3. **sts_server返回AccessKeyId、AccessKeySecret、SecurityToken、Expiration。**
4. **OssDemo獲取這些資訊後，調用SDK，構建OssClient，進行分區上傳。**
5. **點擊暫停時，如果分區上傳沒有結束，再點擊繼續時，可以接著未完成的地方繼續上傳。以達到斷點續傳的效果。**

具體代碼

1. **生成一個Button控制項。**

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

2. **點擊上傳，選擇要上傳的檔案。**

   函數實現片段:

```
Button multipart_upload = (Button) findViewById(R.id.multipart_
upload);
 multipart_upload.setOnClickListener(new View.OnClickListener() {
     @Override
     public void onClick(View v) {
```

```
            //為了簡單化，這裡只會同時運行一個斷點上傳的任務
            Intent i = new Intent(
                    Intent.ACTION_PICK,
                    android.provider.MediaStore.Images.Media.EXTERNAL_C
ONTENT_URI);
            startActivityForResult(i, RESULT_PAUSEABLEUPLOAD_IMAGE);
        }
    }
    );
```

3. 點擊上傳，繼續時的斷點續傳。

   函數實現片段：

```
點擊上傳： //這裡調用SDK的分區上傳介面來上傳 task = ossService.asyncMulti
PartUpload(objectName, picturePath, getMultiPartCallback().
addCallback(new Runnable() {     @Override     public void run()
 {          pauseTaskStatus = TASK_NONE;          multipart_resume.
setEnabled(false);          multipart_pause.setEnabled(false);
     task = null;     } }}, new ProgressCallbackFactory<PauseableU
ploadRequest>().get()); 底層對SDK的封裝邏輯，可以看到是在multiPartU
ploadManager中的asyncUpload實現的斷點續傳上傳 //斷點上傳，返回的task可以
用來暫停任務 public PauseableUploadTask asyncMultiPartUpload(String
 object,                                                    String
localFile,                                                   @NonNull
 final OSSCompletedCallback<PauseableUploadRequest, PauseableU
ploadResult> userCallback,
        final OSSProgressCallback<PauseableUploadRequest> userProgre
ssCallback) {     if (object.equals("")) {          Log.w("AsyncMulti
PartUpload", "ObjectNull");          return null;     }     File file
 = new File(localFile);     if (!file.exists()) {          Log.w("
AsyncMultiPartUpload", "FileNotExist");          Log.w("LocalFile",
localFile);          return null;     }     Log.d("MultiPartUpload",
localFile);     PauseableUploadTask task = multiPartUploadManager.
asyncUpload(object, localFile, userCallback, userProgressCallback);
    return task; }
```

10.4 圖片處理

在OssDemo中展示了上傳一張圖片後，各種不同的處理。和下載不同的地方是：

- 使用的是圖片處理的Endpoint。
- 在Object後面帶了一些處理參數。

圖片加浮水印

調用邏輯

1. 上傳一張圖片到OSS，在預設的情況下bucket是sdk-demo，object是test，OSS的Endpoint是`oss-cn-hangzhou.aliyuncs.com`。
2. 根據不同的圖片處理方式，在test後面加不同的處理參數，以展示不同的顯示效果。
3. 選擇後OssDemo在獲取sts_server的地址後，發送請求。
4. sts_server返回AccessKeyId、AccessKeySecret、SecurityToken、Expiration。

5. OssDemo獲取這些資訊後，調用SDK，構建OssClient，進行下載操作。呈現的效果就是圖片處理的效果。不過圖片服務的Endpoint是`img-cn-hangzhou.aliyuncs.com`。

具體代碼

1. 點擊更多後，到了圖片服務處理圖片後的頁面。
2. 將之前上傳的圖片，在右下角加浮水印，並且大小為100，獲取這樣的操作命令。

   函數實現片段：

```
在ImageService類中
 提供了這樣的一個方法，主要是在原來的object後增加相應的功能需要的參數
 //給圖片打上文字浮水印，除了大小字型之外其他都是預設值，有需要更改的可以參考圖片服務文檔自行調整
 public String textWatermark(String object, String text, int size) {
     String base64Text = Base64.encodeToString(text.getBytes(),
Base64.URL_SAFE | Base64.NO_WRAP);
     String queryString = "@watermark=2&type=" + font + "&text=" +
base64Text + "&size=" + String.valueOf(size);
     Log.d("TextWatermark", object);
     Log.d("Text", text);
     Log.d("QuerySyring", queryString);
     return (object + queryString);
 }
```

3. 調用SDK的下載介面，進行圖片處理。

   函數實現片段：

```
getImage(imageService.textWatermark(objectName, "OSS測試", 100), 0,
 "右下角文字浮水印，大小100");
 public void getImage(final String object, final Integer index,
final String method) {
     GetObjectRequest get = new GetObjectRequest(bucket, object);
     Log.d("Object", object);
     OSSAsyncTask task = oss.asyncGetObejct(get, new UICallback<
GetObjectRequest, GetObjectResult>(uiDispatcher) {
         @Override
         public void onSuccess(GetObjectRequest request, GetObjectR
esult result) {
             // 請求成功
             InputStream inputStream = result.getObjectContent();
             Log.d("GetImage", object);
             Log.d("Index", String.valueOf(index));
             try {
                 //防止超過顯示的最大限制
                 adapter.getImgMap().put(index, new ImageDisplayer(
1000, 1000).autoResizeFromStream(inputStream));
                 adapter.getTextMap().put(index, method + "\n" +
object);
                 //需要根據對應的View大小來自適應縮放
                 addCallback(new Runnable() {
                     @Override
                     public void run() {
                         adapter.notifyDataSetChanged();
                     }
                 }, null);
             }
             catch (IOException e) {
```

```
                e.printStackTrace();
            }
            super.onSuccess(request,result);
        }
```

這裡省略了對下載結果失敗的處理，可以參考源碼中的onFailure的處理。

圖片縮放、裁剪、旋轉

和加浮水印的過程類似，在ImageService中增加獲取處理命令的函數，以”object + 處理參數“的形式返回，最後調用SDK的Get Object介面來處理。

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