

无服务器云函数

最佳实践

产品文档



腾讯云

【版权声明】

©2013-2018 腾讯云版权所有

本文档著作权归腾讯云单独所有，未经腾讯云事先书面许可，任何主体不得以任何形式复制、修改、抄袭、传播全部或部分本文档内容。

【商标声明】

及其它腾讯云服务相关的商标均为腾讯云计算（北京）有限责任公司及其关联公司所有。本文档涉及的第三方主体的商标，依法由权利人所有。

【服务声明】

本文档意在向客户介绍腾讯云全部或部分产品、服务的当时的整体概况，部分产品、服务的内容可能有所调整。您所购买的腾讯云产品、服务的种类、服务标准等应由您与腾讯云之间的商业合同约定，除非双方另有约定，否则，腾讯云对本文档内容不做任何明示或模式的承诺或保证。

文档目录

最佳实践

最佳实践概述

使用 API 网关提供 API 服务

示例说明

步骤 1. 创建并测试 blogArticle 函数

步骤 2. 创建并测试 API 服务

步骤 3. 发布 API 服务并在线验证

获取 COS 上的图像并创建缩略图

示例说明

步骤 1. 准备 COS Bucket

步骤 2. 创建部署程序包

步骤 3. 创建 CreateThumbnailDemo 函数并测试

步骤 4. 添加触发器

使用 SCF 定时备份数据库到 COS

示例说明

步骤 1. 准备 msyqlDump

步骤 2. 创建及部署云函数

步骤 3. 测试及启动云函数

MapReduce 方法

示例说明

步骤 1. 准备 COS Bucket

步骤 2. 创建部署程序包

步骤 3. 创建 Mapper 和 Reducer 函数并测试

步骤 4. 添加触发器

根据 CMQ 中的消息发送邮件

示例说明

步骤 1. 创建 CMQ Topic 主题模式队列

步骤 2. 创建并测试 sendEmail 函数

步骤 3. 添加触发器并测试

使用 AI 接口实现身份证识别

使用 SCF 定时拨测并通过邮件发送告警

步骤 1. 创建邮件发送函数并测试

步骤 2. 创建拨测函数并测试

使用 SCF 实现日志分析写数据库

最佳实践

最佳实践概述

最近更新时间：2018-08-28 18:03:01

根据无服务器云函数的特点，我们推荐您这样使用：

- 以无状态的风格编写函数代码，确保您的代码不会进行任何状态维护。本地存储和内存结果都是可能丢失的，应当使用COS、Redis/Memcached等服务缓存中间信息并落地最终计算结果
- 在执行方法外实例化任何可能复用的对象（如数据库连接等）
- 请务必已在上传的 ZIP 中设置对您的文件的 +rx（可读及执行）权限，以确保代码能够执行
- 如果用户希望尽量降低启动延时，可以尽可能少地使用与处理当前事件无直接关系的启动代码，以降低开销并提高性能。并且，底层计算资源有一定程度的复用可能，如果用户希望尽可能降低启动延时，可定时执行函数来使得后续的调用都是“热启动”。
- 在代码中尽可能多地使用 log/print 语句，给调试工作带来充足的信息。
- 用户可以使用外部的代码管理服务（Git等）进行核心代码的版本和审计管理，保证代码的完备性（版本管理功能将后续加入）

使用 API 网关提供 API 服务 示例说明

最近更新时间：2018-08-28 16:04:29

本教程假设以下情况：

- 您希望使用云函数来实现 Web 后端服务，例如提供博客内的文章查询和文章内容。
- 您希望使用 API 来对外提供服务供网页和 APP 使用。

实现概要

下面是该服务的实现流程：

- 创建函数，在 API 网关中配置 API 规则并且后端服务指向函数。
- 用户请求 API 时带有文章编号。
- 云函数根据请求参数，查询编号对应内容，并使用 json 格式响应请求。
- 用户可获取到 json 格式响应后进行后续处理。

请注意，完成本教程后，您的账户中将具有以下资源：

- 一个由 API 网关触发的 SCF 云函数。
- 一个 API 网关中的 API 服务及下属的 API 规则。

本教程分为了三个主要部分：

- 完成函数代码编写、函数创建和测试。
- 完成 API 服务和 API 规则的设计，创建及配置。
- 通过浏览器或 HTTP 请求工具测试验证 API 接口工作的正确性。

API 设计

现代应用的 API 设计通常遵守 Restful 规范，因此，在此示例中，我们设计获取博客文章的 API 为以下形式

- /article GET
返回文章列表

- /article/{articleId} GET
根据文章 id，返回文章内容

步骤 1. 创建并测试 blogArticle 函数

最近更新时间：2018-09-03 15:26:36

在此部分中，将创建一个函数来实现博客文章的 API 响应，并通过控制台调用来测试函数。

创建 blogArticle 云函数

1. 登录[无服务器云函数控制台](#)，在【广州】地域下单击【新建】按钮。
2. 进入函数配置部分，函数名称填写 `blogArticle`，剩余项保持默认，单击【下一步】。
3. 进入函数代码部分，执行方法填写 `index.main_handler`，代码窗口内贴入如下代码，单击【下一步】。

```
# -*- coding: utf8 -*-
import json

testArticleInfo=[
{"id":1,"category":"blog","title":"hello world","content":"first blog! hello world!","time":"2017-12-05 13:45"},
{"id":2,"category":"blog","title":"record info","content":"record work and study!","time":"2017-12-06 08:22"},
{"id":3,"category":"python","title":"python study","content":"python study for 2.7","time":"2017-12-06 18:32"},
]

def main_handler(event,content):
if "requestContext" not in event.keys():
return {"errorCode":410,"errorMsg":"event is not come from api gateway"}
if event["requestContext"]["path"] != "/article/{articleId}" and event["requestContext"]["path"] != "/article":
return {"errorCode":411,"errorMsg":"request is not from setting api path"}
if event["requestContext"]["path"] == "/article" and event["requestContext"]["httpMethod"] == "GET": #获取文章列表
retList = []
for article in testArticleInfo:
retItem = {}
retItem["id"] = article["id"]
retItem["category"] = article["category"]
retItem["title"] = article["title"]
retItem["time"] = article["time"]
retList.append(retItem)
return retList
```

```
if event["requestContext"]["path"] == "/article/{articleId}" and event["requestContext"]["httpMethod"] == "GET": #获取文章内容
    articleId = int(event["pathParameters"]["articleId"])
    for article in testArticleInfo:
        if article["id"] == articleId:
            return article
    return {"errorCode":412,"errorMsg":"article is not found"}
    return {"errorCode":413,"errorMsg":"request is not correctly execute"}
```

4. 进入触发方式部分，由于 API 网关触发的配置位于 API 网关中，此处暂时不添加任何触发方式，单击【完成】按钮。

注意

保存文章的数据结构使用 testArticleInfo 变量进行保存和模拟，此处在实际应用中通常为从数据库中或者文件中读取。

测试 blogArticle 云函数

在创建函数时，通常会使用控制台或 API 先进行测试，确保函数输出符合预期后再绑定触发器进行实际应用。

1. 在刚刚创建的函数详情页中，单击【测试】按钮；
2. 在测试模版内选择【API Gateway 测试模版】，并修改模版成为如下内容，此内容为测试获取文章列表的 API。

```
{
  "requestContext": {
    "serviceName": "testsvc",
    "path": "/article",
    "httpMethod": "GET",
    "requestId": "c6af9ac6-7b61-11e6-9a41-93e8deadbeef",
    "identity": {
      "secretId": "abdcxxxxxxxxsdfs"
    },
    "sourceIp": "10.0.2.14",
    "stage": "prod"
  },
  "headers": {
    "Accept-Language": "en-US,en,cn",
    "Accept": "text/html,application/xml,application/json",
    "Host": "service-3ei3tii4-251000691.ap-guangzhou.apigateway.myqcloud.com",
    "User-Agent": "User Agent String"
  },
  "pathParameters": {
```

```
},
"queryStringParameters": {
},
"headerParameters": {
  "Refer": "10.0.2.14"
},
"path": "/article",
"httpMethod": "GET"
}
```

其中 `requestContext` 内的 `path` , `httpMethod` 字段, 外围的 `path` , `httpMethod` 字段, 均修改为我们设计的 API 路径 `/article` 和方法 `GET`。

3. 单击【运行】按钮, 观察运行结果。运行结果应该为成功, 且返回内容应该为如下所示的文章概要内容。

```
[{"category": "blog", "time": "2017-12-05 13:45", "id": 1, "title": "hello world"}, {"category": "blog", "time": "2017-12-06 08:22", "id": 2, "title": "record info"}, {"category": "python", "time": "2017-12-06 18:32", "id": 3, "title": "python study"}]
```

4. 修改测试模版成为如下内容, 此内容为测试获取文章内容的 API。

```
{
  "requestContext": {
    "serviceName": "testsvc",
    "path": "/article/{articleId}",
    "httpMethod": "GET",
    "requestId": "c6af9ac6-7b61-11e6-9a41-93e8deadbeef",
    "identity": {
      "secretId": "abdcxxxxxxxxsdfs"
    },
    "sourceIp": "10.0.2.14",
    "stage": "prod"
  },
  "headers": {
    "Accept-Language": "en-US,en,cn",
    "Accept": "text/html,application/xml,application/json",
    "Host": "service-3ei3tii4-251000691.ap-guangzhou.apigateway.myqcloud.com",
    "User-Agent": "User Agent String"
  },
  "pathParameters": {
    "articleId": "1"
  },
  "queryStringParameters": {
  },
  "headerParameters": {

```

```
"Refer": "10.0.2.14",
},
"path": "/article/1",
"httpMethod": "GET"
}
```

其中 `requestContext` 内的 `path` , `httpMethod` 字段, 外围的 `path` , `httpMethod` 字段, 均修改为我们设计的 API 路径 `/article/{articleId}` 和实际请求路径 `/article/1` , 方法为 `GET` , `pathParameters` 字段内应该为 API 网关内抽取出来的参数和实际值 `"articleId": "1"` 。

5. 单击【运行】按钮, 观察运行结果。运行结果应该为成功, 且返回内容应该为如下所示的文章详细内容。

```
{"category": "blog", "content": "first blog! hello world!", "time": "2017-12-05 13:45", "id": 1, "title": "hello world"}
```

步骤 2. 创建并测试 API 服务

最近更新时间：2018-09-03 15:27:06

在此部分中，将创建一个 API 网关中的服务和相关的 API 规则，对接在步骤一中创建的 SCF 云函数，并通过控制台的 API 测试，来测试 API 的正确性。

注意：

API 服务和函数必须位于同一个地域下。在本教程中，将使用广州区域来创建 API 服务。

创建 API 服务和 API 规则

1. 登录[腾讯云控制台](#)，从云产品中选择【互联网中间件】-【API 网关】。
2. 单击【服务】选项卡，并切换地域为【广州】。
3. 单击【新建】按钮以新建 API 服务，在弹出窗口中写入服务名 `blogAPI`，单击提交创建。
4. 完成服务创建后，进入创建的 `blogAPI` 服务，选择【API 管理】选项卡。
5. 单击【新建】创建 API，路径为 `/article`，请求方法为 GET，为了方便后面的测试，在这里勾选上免鉴权，无需输入参数配置，单击【下一步】。
6. 后端类型选择为【cloud function】，选择函数为步骤一中创建的 `blogArticle`，单击【完成】。
7. 再次在【API 管理】选项卡中单击【新建】创建 API，路径为 `/article/{articleId}`，请求方法为 GET，勾选上免鉴权，参数配置中输入名称为 `articleId` 的参数，参数位置为 Path，类型为 int，默认值为 1，单击【下一步】。
8. 后端类型选择为【cloud function】，选择函数为步骤一中创建的 `blogArticle`，单击【完成】。

对 API 规则进行调试

1. 针对前面第 5 步创建的 `/article` API，单击【API 调试】，在调试页面发送请求，确保返回结果内的响应 Body，为如下内容：

```
[{"category": "blog", "time": "2017-12-05 13:45", "id": 1, "title": "hello world"}, {"category": "blog", "time": "2017-12-06 08:22", "id": 2, "title": "record info"}, {"category": "python", "time": "2017-12-06 18:32", "id": 3, "title": "python study"}]
```

2. 针对前面第 7 步创建的 `/article/{articleId}` API，单击【API 调试】，在调试页面将请求参数修改为 1 后发送请求，确保返回结果内的响应 Body，为如下内容：

```
{"category": "blog", "content": "first blog! hello world!", "time": "2017-12-05 13:45", "id": 1, "title": "hello world"}
```

3. 也可以修改第 2 步中的请求参数 `articleId` 的值为其他数字，查看响应内容。

步骤 3. 发布 API 服务并在线验证

最近更新时间：2018-09-03 15:27:20

如果您完成了步骤二：创建并测试 API 服务，且测试结果符合预期，则接下来可以对外发布此服务并从浏览器发起请求来验证接口的工作情况。

API 服务发布

1. 在 API 网关控制台的【服务】列表页中，找到在步骤二创建的 blogAPI 服务，并单击服务操作中的【发布】按钮。
2. 在发布服务的弹窗中，发布环境选择 发布，备注内填入 发布API，单击【提交】。

API 在线验证

通过发布动作，完成了 API 服务的发布，使得 API 可以被外部所访问到，接下来通过浏览器发起请求来查看 API 是否能正确响应。

1. 在 blogAPI 服务中，进入【环境管理】选项卡，复制其中 发布 环境的访问路径，例如 service-kzeed206-1251762227.ap-guangzhou.apigateway.myqcloud.com/release。
注意 这里由于每个服务的域名均不相同，您的服务所分配到的域名将与本文中的服务域名有差别，请勿直接拷贝本文中的地址访问。
2. 在此路径后增加创建的 API 规则的路径，形成如下路径。

```
service-kzeed206-1251762227.ap-guangzhou.apigateway.myqcloud.com/release/article  
service-kzeed206-1251762227.ap-guangzhou.apigateway.myqcloud.com/release/article/1  
service-kzeed206-1251762227.ap-guangzhou.apigateway.myqcloud.com/release/article/2
```

3. 将第2步中的路径复制到浏览器中访问，确定输出内容与测试 API 时的输出相同。
4. 可进一步修改请求中的文章编号并查看输出，查看代码是否能正确处理错误的文章编号。

至此完成了通过 SCF 云函数实现服务，通过 API 对外提供服务。后续可以通过继续修改代码，增加功能并增加 API 规则，使其完善成为一个更丰富的应用模块。

获取 COS 上的图像并创建缩略图 示例说明

最近更新时间：2018-08-28 15:55:13

本教程假设以下情况：

- 您的用户将上传照片至某个特定的COS Bucket
- 您要为用户上传的每个图像创建一个缩略图
- 创建完缩略图后保存至另一个 COS Bucket

注意：

1. 必须使用两个COS Bucket。如果使用同一个存储桶作为源和目标，上传到源存储桶的每个缩略图都会触发另一个对象创建事件，该事件将再次调用函数，从而产生无限的循环。
2. 请保证函数和 COS Bucket位于同一个地域下

实现概要

下面是该函数的实现流程：

- 创建函数与COS Bucket的事件源映射
- 用户将对象上传到 COS 中的源存储桶（对象创建事件）。
- COS Bucket检测到对象创建事件。
- COS 调用函数并将事件数据作为参数传递给函数，由此将 `cos:ObjectCreated:*` 事件发布给函数。
- SCF 平台接收到调用请求，执行函数。
- 函数通过收到的事件数据获得了 Bucket 名称和文件名称，从该源 Bucket中获取该文件，使用图形库创建缩略图，然后将其保存到目标Bucket上。

请注意，完成本教程后，您的账户中将具有以下资源：

- 一个创建缩略图的 SCF 函数。
- 两个COS Bucket：和（您自行指定的COS Bucket名称。例如：如果您将名为 `example` 的 Bucket 作为源，您将创建 `examplesized` 作为目标 Bucket）
- 源 Bucket 上的通知配置：将 SCF 函数和 COS Bucket绑定将在该 Bucket 的通知配置上添加新项，用来标识 COS 将触发函数的事件类型（文件创建/删除）及要调用的函数名称。有关 COS 通知功能的更多信息，请参阅 [PutBucketNotification](#) 接口。

本教程分为了两个主要部分：

-
- 完成创建函数的必要设置步骤，并使用 COS 示例事件数据手动调用该函数。旨在验证函数能够正常工作。
 - 向源 Bucket 添加通知配置，使得 COS 在检测到文件创建事件时能够调用函数。

步骤 1. 准备 COS Bucket

最近更新时间：2018-09-03 15:27:41

注意：

1. 源 Bucket、目标 Bucket 和函数必须位于同一个地域下。在本教程中，我们将使用华南（广州）区域。
2. 必须使用两个COS Bucket。如果使用同一个 Bucket 作为源和目标，上传到源存储桶的每个缩略图都会再次触发函数，从而产生不必要的递归。

1) 登录腾讯云控制台，选择【对象存储服务】。

2) 单击【Bucket列表】选项卡下的【创建Bucket】按钮，新建源 COS Bucket。

3) 设置COS Bucket的名称如 mybucket，选择地域为 华南，设置访问权限为默认值 公有读私有写 并设置CDN加速为默认值 关闭，单击【保存】按钮新建一个COS Bucket。

4) 按照相同的方式创建目标 Bucket mybucketresized

5) 在源 Bucket（即mybucket）中，上传任意一个图片文件，本示例中使用了一个 HappyFace.png 的图片作为演示。（在实际关联 COS 前手动调用函数进行测试验证时，您要将包含该文件的示例数据传递给 SCF 函数，且 SCF 函数将根据该数据寻找相应的文件。因此您需要先创建此示例对象。）



步骤 2. 创建部署程序包

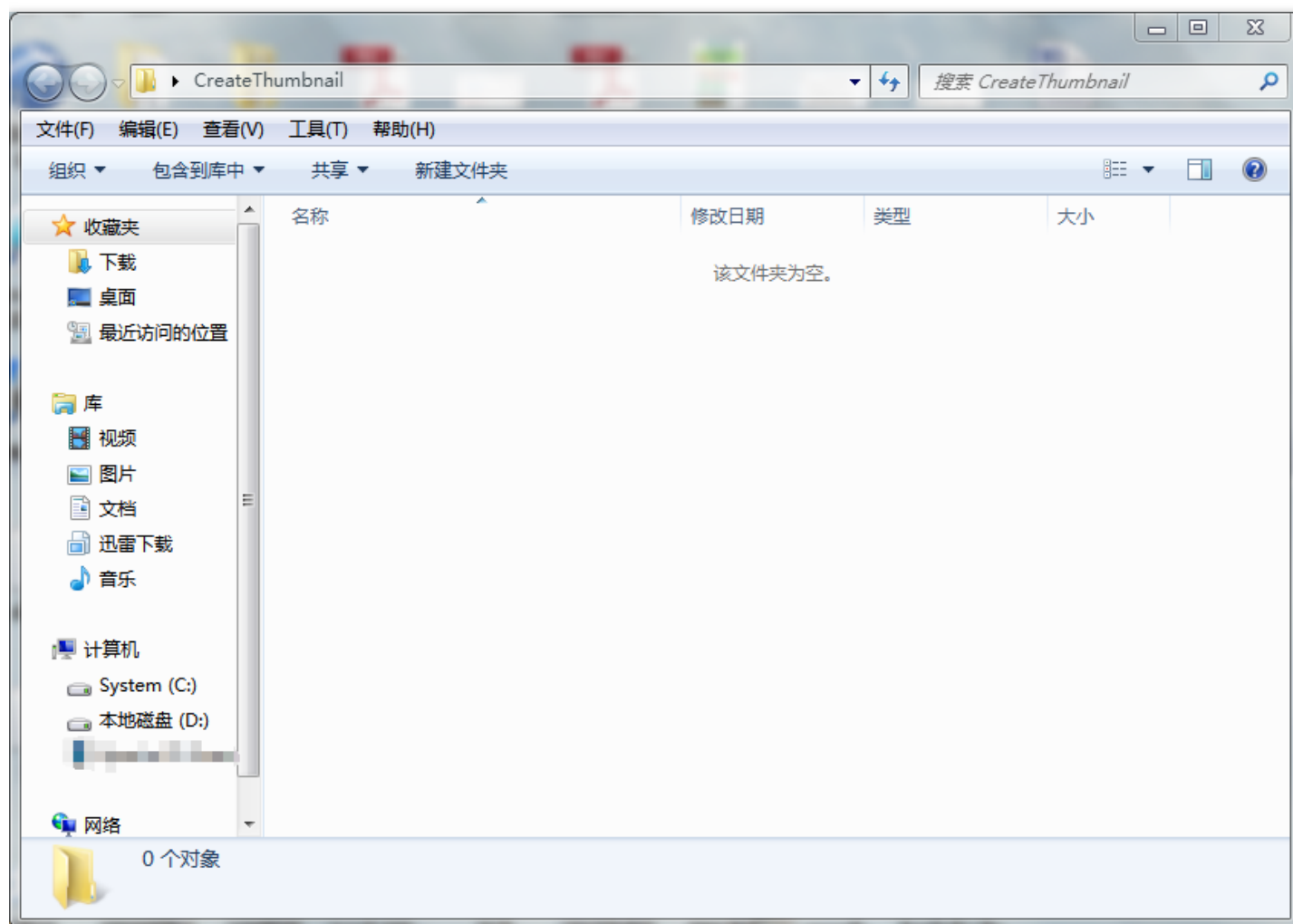
最近更新时间：2018-09-03 15:27:54

由于 SCF 的底层容器环境使用了 CentOS 7.2 版本的镜像。因此本示例中的 Linux 示例部分是在 CentOS 7.2 环境下的操作，如果您本地环境为其他 Linux 发行版，请适当调整操作

创建图片处理代码

1) 新建目录

如果您本地环境为 Windows，请在任意位置新建一个名为 CreateThumbnail 的文件夹，如下图所示：



如果您本地环境为 Linux，请在任意位置新建一个名为 CreateThumbnail 的文件夹，如下图所示：

```
[root@localhost ~]# mkdir /CreateThumbnail
```

2) 打开文本编辑器，输入以下代码

请特别注意，将参数 `appid`, `secret_id`, `secret_key`, `region` 改为您的实际数据，其中：

- `appid`可在控制台【账户信息】中获得



- `secret_id` 和 `secret_key`可在控制台【云API密钥】中获得



- `region` 为函数和COS Bucket所在地域，支持 `sh`、`gz`、`bj` 三个值。请注意，必须保持和上一步骤中创建的COS Bucket在同一个地域。此处由于步骤一：准备COS Bucket中创建的存储桶位于华南（广州），因此代码中的 `region` 值必须为 `gz`

```
import uuid
import json
import os
import logging
from PIL import Image
import PIL.Image
import commands
```

```
import datetime
import urllib
from qcloud_cos import CosClient
from qcloud_cos import DownloadFileRequest
from qcloud_cos import UploadFileRequest

print('Loading function')
appid = 1251762222 #please change to your appid. Find it in Account Info
secret_id = u'AKIDYDh085xQp48161uOn2CKKVbeebvDu6j2' #please change to your API secret id. Find it in API secret key pair
secret_key = u'ILkxx40klfuyqW0lOI0WqyueCYjlgZQ2' #please change to your API secret key. Find it in API secret key pair
region = u'gz'

cos_client = CosClient(appid, secret_id, secret_key, region)
logger = logging.getLogger()

def resize_image(image_path, resized_path):
    with Image.open(image_path) as image:
        image.thumbnail(tuple(x / 2 for x in image.size))
        image.save(resized_path)

def delete_local_file(src):
    logger.info("delete files and folders")
    if os.path.isfile(src):
        try:
            os.remove(src)
        except:
            pass
    elif os.path.isdir(src):
        for item in os.listdir(src):
            itemsrc=os.path.join(src,item)
            delete_file_folder(itemsrc)
        try:
            os.rmdir(src)
        except:
            pass

def main_handler(event, context):
    logger.info("start main handler")
    for record in event['Records']:
        try:
            bucket = record['cos']['cosBucket']['name']
            cosobj = record['cos']['cosObject']['key']
```

```
cosobj = cosobj.replace("/"+str(appid)+"/"+bucket,"")
key = urllib.unquote_plus(cosobj.encode('utf8'))
download_path = '/tmp/{}'.format(uuid.uuid4(), key.strip('/'))
upload_path = '/tmp/resized-{}'.format(key.strip('/'))
print("Get from [%s] to download file [%s]" %(bucket,key))

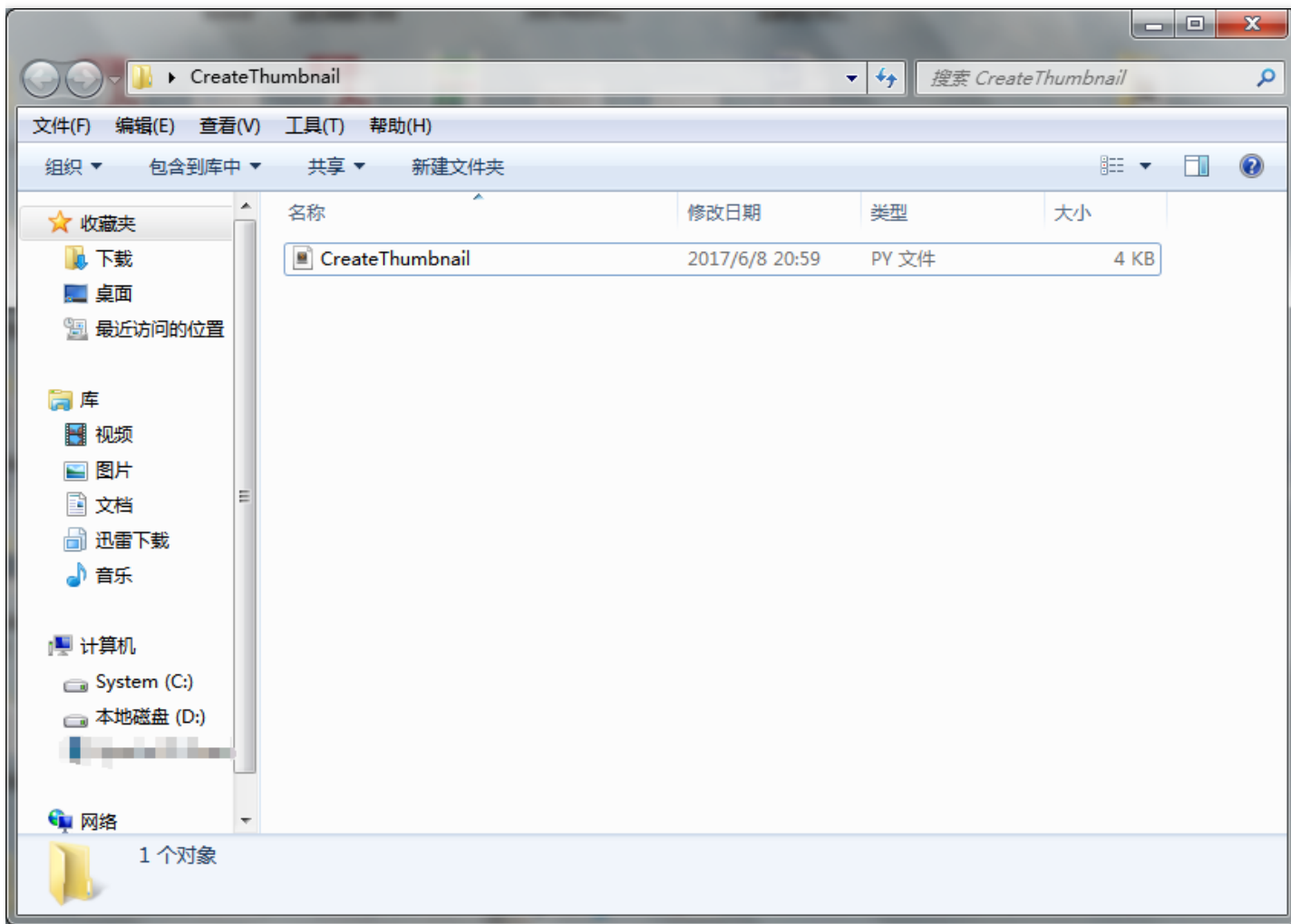
# download image from cos
request = DownloadFileRequest(bucket, key, download_path)
download_file_ret = cos_client.download_file(request)
if download_file_ret['code'] == 0:
    logger.info("Download file [%s] Success" % key)
    logger.info("Image compress function start")
    starttime = datetime.datetime.now()

#compress image here
resize_image(download_path, upload_path)
endtime = datetime.datetime.now()
logger.info("compress image take " + str((endtime-starttime).microseconds/1000) + "ms")

#upload the compressed image to resized bucket
request = UploadFileRequest(u'%sresized' % bucket, key.decode('utf-8'), upload_path.decode('utf-8'))
upload_file_ret = cos_client.upload_file(request)
logger.info("upload image, return message: " + str(upload_file_ret))

#delete local file
delete_local_file(str(download_path))
delete_local_file(str(upload_path))
else:
    logger.error("Download file [%s] Failed, err: %s" % (key, download_file_ret['message']))
    return -1
except Exception as e:
    print(e)
    print('Error getting object {} from bucket {}. Make sure the object exists and your bucket is in the same region as this function.'.format(key, bucket))
    raise e
```

3) 将文件另存为 `CreateThumbnail.py`，保存在刚刚创建的目录下：



```
[root@UM_67_49_centos ~]# cd /CreateThumbnail
[root@UM_67_49_centos CreateThumbnail]# vi CreateThumbnail.py
```

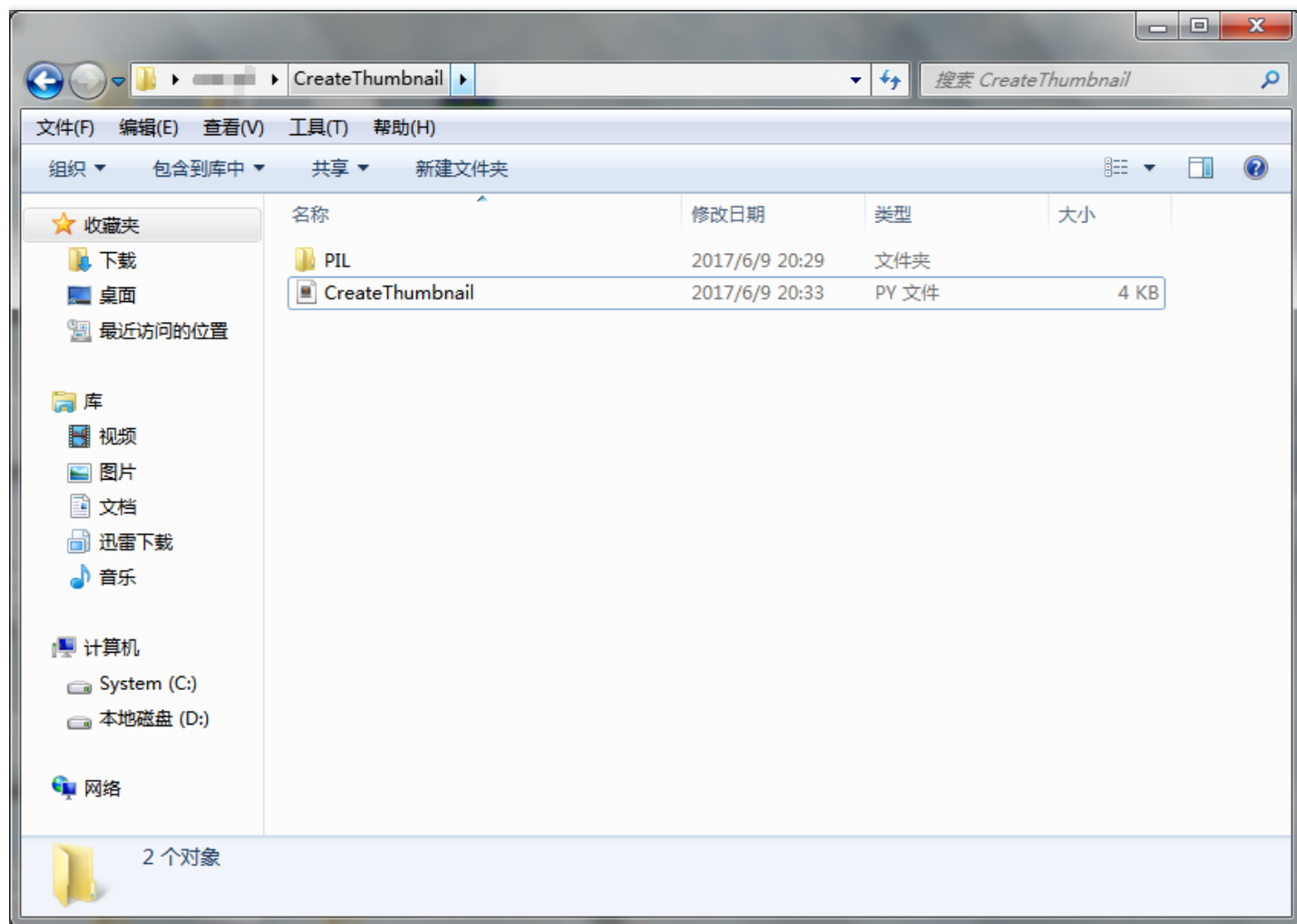
```
[root@UM_67_49_centos CreateThumbnail]# ls
CreateThumbnail.py
```

创建部署程序包

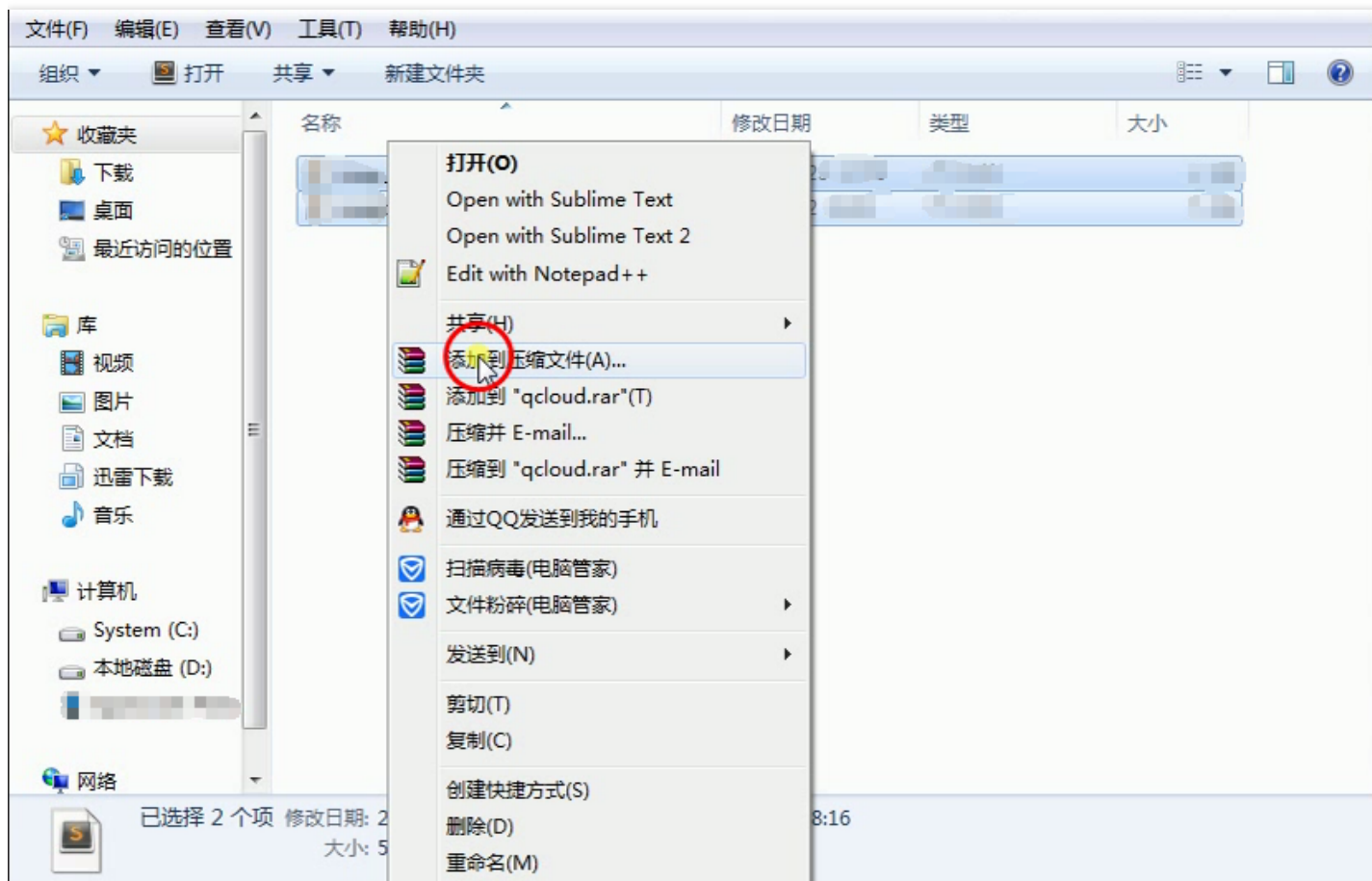
如果您本地环境是 Windows

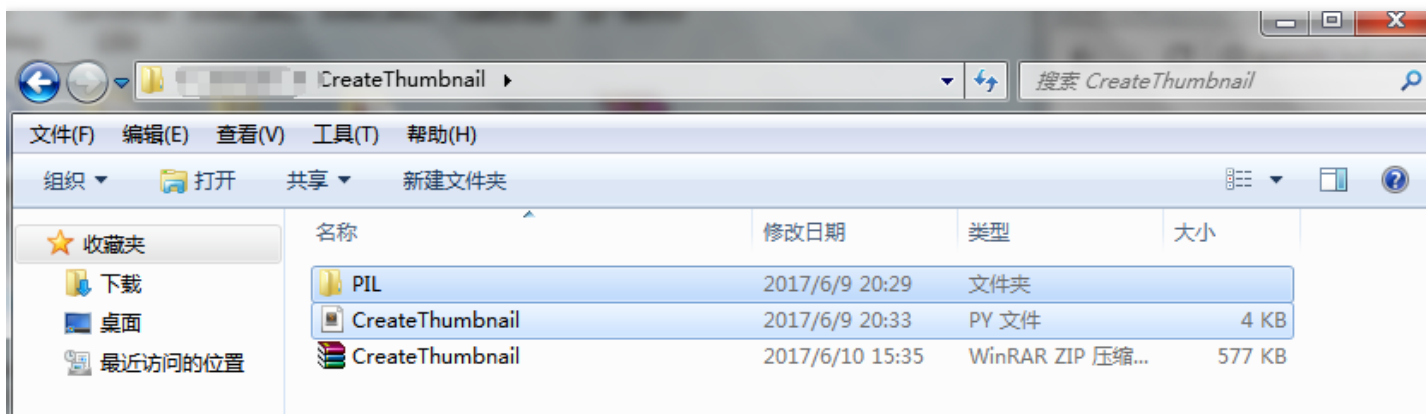
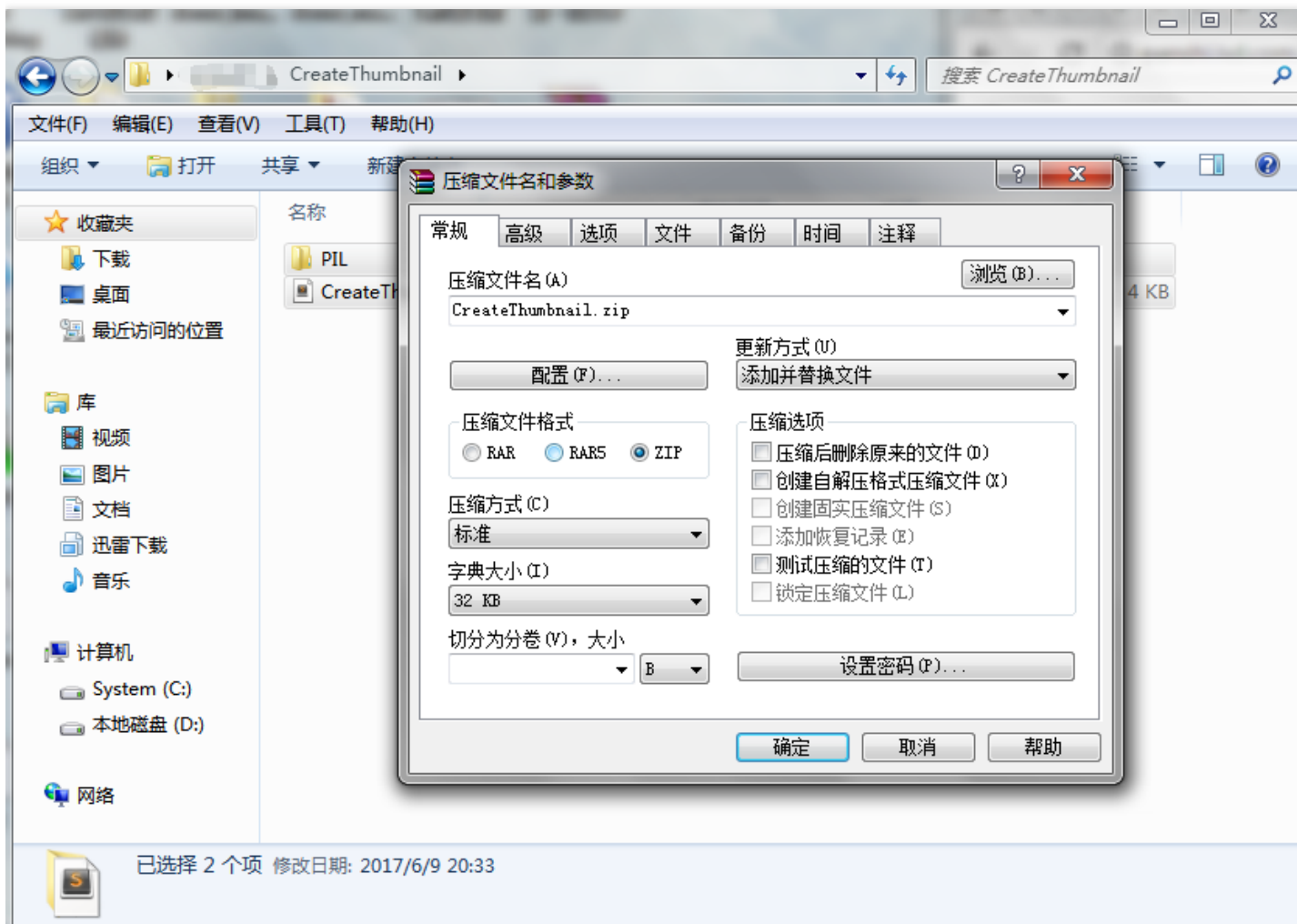
由于本示例程序依赖与 Python 的 Pillow 依赖库，为了避免您本地 Windows 环境下安装的依赖库与平台冲突，我们建议您：

直接单击链接下载 [Pillow 库](#)，并将该zip包解压至刚刚创建的CreateThumbnail 文件夹内：



压缩该文件夹下的所有内容至一个名为 CreateThumbnailDemo.zip 的压缩包中（注意不是压缩文件夹本身！）：选中所有文件，单击右键，选择您的压缩工具如winrar，单击【添加到压缩文件...】，将压缩文件设置为zip格式，单击【确定】按钮，将生成一个zip包（此处命名为 CreateThumbnailDemo.zip）。





如果您本地环境是 Linux

请注意：以下步骤是假设在 **CentOS 7.2** 环境下，如果您的环境是其他 **Linux** 发行版，请根据该版本的相关方法修改命令，并确保 **Python** 版本为 2.7

1) 安装 python 环境

```
sudo yum install python
```

2) 请确保您当前的 Linux 环境安装了必要的依赖

```
sudo yum install python-devel python-pip gcc libjpeg-devel zlib-devel python-virtualenv
```

3) 创建和激活虚拟环境

```
virtualenv ~/shrink_venv  
source ~/shrink_venv/bin/activate
```

4) 在虚拟环境下安装 Pillow 库

```
pip install Pillow
```

5) 将 lib 和 lib64 的相关内容添加至一个 .zip 文件中 (假定路径为 /CreateThumbnailDemo.zip)

```
cd $VIRTUAL_ENV/lib/python2.7/site-packages  
zip -r /CreateThumbnailDemo.zip *  
cd $VIRTUAL_ENV/lib64/python2.7/site-packages  
zip -r /CreateThumbnailDemo.zip *
```

6) 将第一步中创建的 PY 文件也添加至 .zip文件中

```
cd /CreateThumbnail  
zip -g /CreateThumbnailDemo.zip CreateThumbnail.py
```

步骤 3. 创建 CreateThumbnailDemo 函数并测试

最近更新时间：2018-09-03 15:28:00

在此部分中，用户将创建一个函数来实现缩略图程序，并通过控制台/API调用来测试函数。

创建 CreateThumbnailDemo SCF 函数

- 1) 登录[无服务器云函数控制台](#)，在【广州】地域下单击【新建】按钮；
- 2) 进入函数配置部分，函数名称填写 `CreateThumbnailDemo`，剩余项保持默认，单击【下一步】；
- 3) 进入函数代码部分，选择【本地上传zip包】。执行方法填写 `CreateThumbnail.main_handler`，选择步骤二：创建部署程序包中创建的 `CreateThumbnailDemo.zip`，单击【下一步】；
- 4) 进入触发方式部分，此时由于需要先手动测试函数，暂时不添加任何触发方式，单击【完成】按钮。

测试 CreateThumbnailDemo SCF 函数

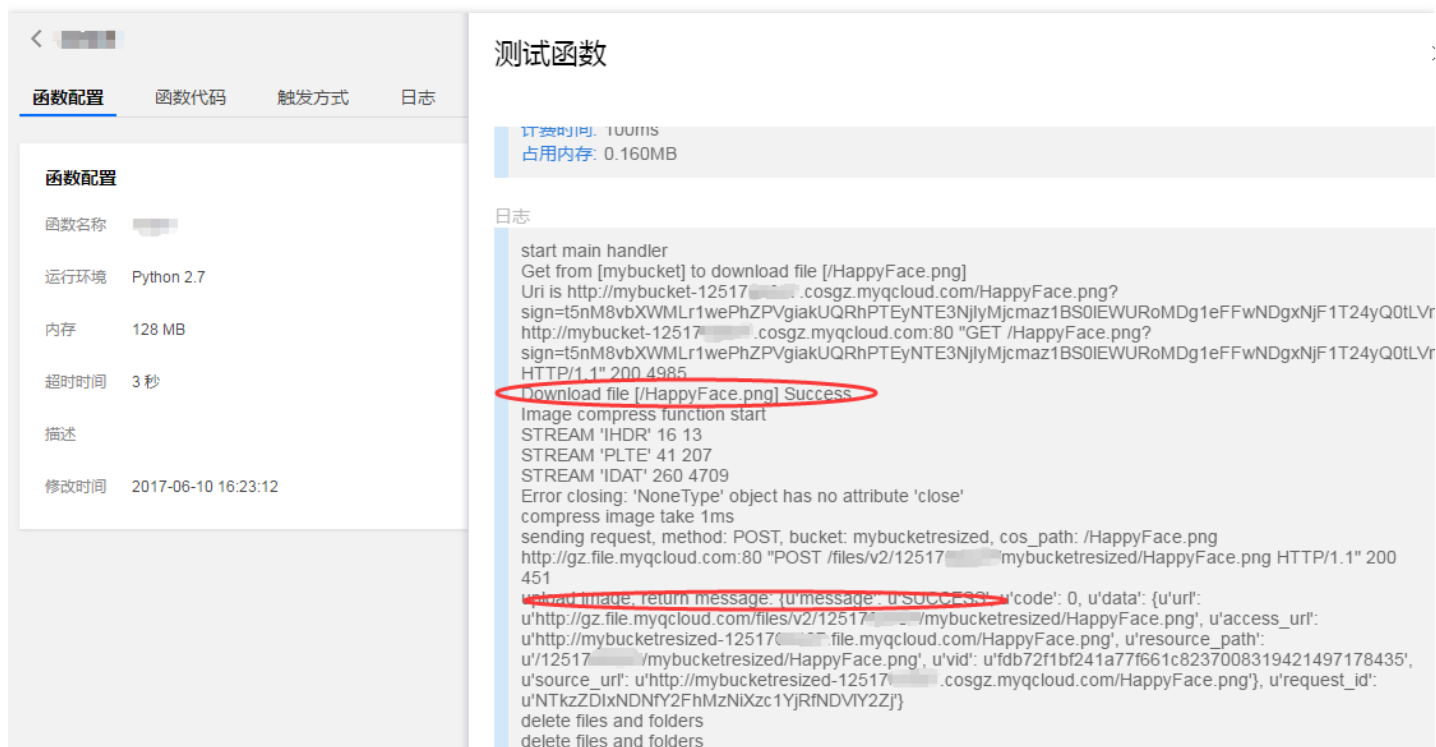
在创建函数时，通常会使用控制台或 API 先进行测试，确保函数输出符合预期后再绑定触发器进行实际应用。

- 1) 在刚刚创建的 `CreateThumbnailDemo` 函数详情页中，单击【测试】按钮；
- 2) 在测试模版下拉列表中选择【COS 上传/删除文件测试代码】
- 3) 轻微改动测试代码，将 `name` 设置为步骤一：准备 COS Bucket 中创建的 `mybucket` 存储桶名称，将 `key` 设置为步骤一：准备 COS Bucket 中上传的 `/HappyFace.jpg` 键值，如下面示例：

```
{
  "Records": [
    {
      "event": {
        "eventVersion": "1.0",
        "eventSource": "qcs::cos",
        "eventName": "event-type",
        "eventTime": "Unix 时间戳",
        "eventQueue": "qcs:0:cos:gz:1251111111:cos",
        "requestParameters": {
          "requestSourceIP": "111.111.111.111",
          "requestHeaders": {
```

```
"Authorization": "上传的鉴权信息"
}
}
},
"cos":{
  "cosSchemaVersion":"1.0",
  "cosNotificationId":"设置的或返回的 ID",
  "cosBucket":{
    "name":"mybucket", #set to demo bucket here
    "appid":"appld",
    "region":"gz"
  },
  "cosObject":{
    "key":"/HappyFace.png", #set to demo file here
    "size":"1024",
    "meta":{
      "Content-Type": "text/plain",
      "x-cos-meta-test": "自定义的 meta",
      "x-image-test": "自定义的 meta"
    },
    "url": "访问文件的源站url"
  }
}
}
}
]
```

4) 单击【运行】按钮，观察运行结果。如果在结果中发现下载和上传均成功，则此程序运行正常：



测试函数

执行时间: 100ms
占用内存: 0.160MB

日志

```
start main handler
Get from [mybucket] to download file [/HappyFace.png]
Uri is http://mybucket-12517-12517-12517.cosgz.myqcloud.com/HappyFace.png?
sign=t5nM8vbXWMLr1wePhZPVgiakUQRhPTEyNTE3NjlyMjcmaZ1BS0IEWURoMDg1eFFwNDgxNjF1T24yQ0tLVr
http://mybucket-12517-12517-12517.cosgz.myqcloud.com:80 "GET /HappyFace.png?
sign=t5nM8vbXWMLr1wePhZPVgiakUQRhPTEyNTE3NjlyMjcmaZ1BS0IEWURoMDg1eFFwNDgxNjF1T24yQ0tLVr
HTTP/1.1" 200 4985
Download file [/HappyFace.png] Success
Image compress function start
STREAM 'IHDR' 16 13
STREAM 'PLTE' 41 207
STREAM 'IDAT' 260 4709
Error closing: 'NoneType' object has no attribute 'close'
compress image take 1ms
sending request, method: POST, bucket: mybucketresized, cos_path: /HappyFace.png
http://gz.file.myqcloud.com:80 "POST /files/v2/12517-12517-12517/mybucketresized/HappyFace.png HTTP/1.1" 200
451
upload image, return message: {'u:message': 'u:SUCCESS', 'u:code': 0, 'u:data': {'u:url':
u'http://gz.file.myqcloud.com/files/v2/12517-12517-12517/mybucketresized/HappyFace.png', 'u:access_url':
u'http://mybucketresized-12517-12517-12517.file.myqcloud.com/HappyFace.png', 'u:resource_path':
u'/12517-12517-12517/mybucketresized/HappyFace.png', 'u:vid': 'u:fdb72f1bf241a77f661c8237008319421497178435',
u'source_url': u'http://mybucketresized-12517-12517-12517.cosgz.myqcloud.com/HappyFace.png'}, 'u:request_id':
u'NTkzZDlxNDNFY2FhMzNlXzc1YjRlNDVlY2Zj'}
delete files and folders
delete files and folders
```

5) 前往[对象存储控制台](#)，单击步骤一：准备 COS Bucket 中创建的 mybucketresized，观察是否有名为 HappyFace.png 的缩略图产生。



< 返回 | mybucketresized

文件列表 基础配置 域名管理

+ 上传文件 创建文件夹

文件名	大小
HappyFace.png	2.30KB

6) 下载该图片，对比观察它和原图片的大小。

步骤 4. 添加触发器

最近更新时间：2018-09-03 15:28:05

如果您完成了步骤三：创建 CreateThumbnailDemo 函数并测试，且测试结果符合预期，则您可以添加 COS 配置以便 COS 能向 SCF 发布事件并调用函数。

1) 在刚刚创建的 CreateThumbnailDemo 函数详情页中，选择【触发方式】选项卡，单击【添加触发方式】按钮；

2) 选择 COS 触发，COS Bucket 选择步骤一：准备 COS Bucket 中创建的 mybucket，事件类型选择“文件上传”，单击【保存】按钮；

此时本示例全部完成！现在您可以按以下方式测试设置：

1. 前往[对象存储控制台](#)，选择 mybucket，上传任意的 .jpg 或 .png 图片，一段时间后观察 mybucketresized 中是否有同名文件；
2. 您可以在[无服务器云函数控制台](#)中监控函数的活动，选择【日志】选项来查看函数被调用的日志记录。

使用 SCF 定时备份数据库到 COS 示例说明

最近更新时间：2018-08-28 15:59:27

本代码实操教程假设以下情况：

- 您需要对数据库定期的导出备份；
- 您希望把备份结果放到 COS 以便于随时下载使用。

数据库备份通常是 DBA 每天要进行的工作。对数据库进行备份，可以在数据错误，数据库异常等有需要时及时进行数据回滚。最常用的方式，就是使用 crontab 定时任务，每日调用备份脚本进行数据库备份。而在备份脚本中，通常最方便使用的，就是 mysqldump 工具，导出表结构及表数据。

接下来，我们将利用云函数，实现数据库备份能力，然后通过配置定时触发器，确保备份函数可以按需每天、或按指定间隔时间运行。

实现概要

下面是该函数的实现流程：

1. 准备 msyqldump。
2. 安装云函数相关的依赖库。
3. 创建及部署云函数。
4. 根据需要配置定时触发器并进行测试。

步骤 1. 准备 mysqldump

最近更新时间：2018-09-03 15:28:26

mysqldump 工具常用来导出数据库备份数据，在云函数中也同样能使用；但是由于云函数环境并未内置 mysqldump，因此我们要自行打包工具。

参考下载地址：

[MySQL 社区版下载地址](#)

以操作系统 **Linux - geneic**，操作系统版本 **x86，64-bit** 为例，下载 tar.gz 压缩包并存储在本地。

下载完成后，解压压缩包，可以看到解压后的文件夹内包含有 bin、lib 等目录；而我们要找到的 mysqldump 工具就在 bin 目录下。

同时可以看到 bin 目录下还有 libcrypto.so.1.0.0 和 libssl.so.1.0.0 两个动态库。这两个库也是工具在运行时所要依赖的库，但在 bin 下的这两个文件实际为文件链接，实际指向分别是 `../lib/libcrypto.so.1.0.0` 和 `../lib/libssl.so.1.0.0`，因此这两个真实文件是在 lib 目录下。

因此，为了确保 mysqldump 工具可以运行成功，我们将 bin 目录下的 mysqldump 文件拷贝到我们提前准备的项目根目录下，同时将 lib 目录下的 libcrypto.so，libcrypto.so.1.0.0，libssl.so，libssl.so.1.0.0 四个文件也拷贝到项目根目录下。

由于拷贝出来的 mysqldump 和 so 动态库文件是 Linux 版本，如果需要验证可用性，我们可以将准备好的项目目录拷贝到一台 Linux 服务器上，通过运行 mysqldump 命令验证工具的可用性。

```
./mysqldump -h {host} -P {port} -u{user} -p{password} {dbName} > dump.sql
```

使用如上命令，就可以将数据库内某一个具体的库导出到对应的 sql 文件内，我们可以通过命令运行时的输出，和导出文件的内容，判断是否运行成功。

步骤 2. 创建及部署云函数

最近更新时间：2018-09-03 15:28:31

云函数准备

云函数的主要功能，就是在每一次运行时，调用上一步骤中我们准备好的 `mysqldump` 工具，来连接远程数据库并在本地生成 dump 文件。由于云函数的本地环境中不提供持久存储，生成的 dump 文件，需要上传到对象存储中做持久化，并在所需要的时候可以下载使用。

我们在这里使用 Python 2.7 作为开发语言，在项目根目录下创建 `index.py` 文件，并在文件内输入如下内容并保存。

```
#coding=utf-8

import os
from qcloud_cos import CosConfig
from qcloud_cos import CosS3Client
import sys
import logging
import time

logging.basicConfig(level=logging.INFO, stream=sys.stdout)

DB_HOST = os.getenv('dbhost') #'sh-cdb-irye027y.sql.tencentcdb.com'
DB_PORT = os.getenv('dbport') #'63374'
DB_USER = os.getenv('dbuser') #'root'
DB_USER_PASSWORD = os.getenv('dbpwd') #'abc123!@#'
DB_NAME = os.getenv('dbname') #'cAuth'

BACKUP_PATH = '/tmp'

SECRET_ID = os.getenv('secretid') #'AKIDQm6iUh2NJ6jL41tVUis9KpY5Rgv49zyC'
SECRET_KEY = os.getenv('secretkey') #'xxlbnYqo4EMJ5jPGA7sUvHWaSAIxxxxx'
REGION = os.getenv('cosregion') #'ap-shanghai'
BACKUP_BUCKET = os.getenv('cosbucket') #'dbbackup-1253970226'

config = CosConfig(Secret_id=SECRET_ID, Secret_key=SECRET_KEY, Region=REGION, Token='')
client = CosS3Client(config)

def backup2cos(file, bucket, key):
    response = client.put_object_from_local_file(
        Bucket=bucket,
```

```
LocalFilePath=file,
Key=key,
)
print(response)
return response

def main_handler(event, context):
    timestr = time.strftime('%Y%m%d-%H%M%S', time.localtime(time.time()))
    db = DB_NAME
    filename = db+"-"+timestr+".sql"
    filepath = BACKUP_PATH+os.sep+filename
    print "Start Backup"
    dumpcmd = "./mysqldump -h " + DB_HOST + " -P " + DB_PORT + " -u " + DB_USER + " -p'" + DB_USER_PASSWORD + "' " + db + " > " + filepath
    print dumpcmd
    print os.popen(dumpcmd).read()
    print "Backup script completed"
    print "Your backups has been created in '" + filepath + "' file"
    print os.popen('ls -l /tmp').read()
    print "finish backup"
    print "start send to cos"
    backup2cos(filepath, BACKUP_BUCKET, "/" + filename)
    print "finish send to cos"

if __name__ == "__main__":
    main_handler("", "")
```

在代码中，我们使用到了 COS 的 v5 版本 sdk 进行备份后的文件上传操作，因此我们需要安装好依赖便于脚本调用。我们可以在项目根目录通过执行 `pip install cos-python-sdk-v5 -t .` 命令，来将 COS 的 Python sdk 以及相关依赖安装到项目目录中。

云函数打包及创建

在完成以上几部操作以后，项目目录里就有了 `mysqldump` 工具及依赖的 so 库，有了 cos sdk 及依赖包，以及我们的入口函数文件 `index.py`。接下来就需要将项目文件打包为 zip 格式以便上传至云函数，完成函数创建。

注意：

由于项目目录下的 `mysqldump` 工具作为二进制程序，需要在云函数的环境中运行，因此需要具有可执行权限，需要在 Linux 或 Mac 环境下为此文件赋予可执行权限后再打包，因此建议在 Linux 或 Mac 环境下执行打包。

我们可以将项目目录放置到 Linux 或 Mac 环境下后，通过在项目目录下执行 `chmod +x mysqldump` 命令，为 `mysqldump` 工具附加上可执行权限。

完成附加权限后，可以继续在根目录下通过执行 `zip mysqldump.zip *` 命令，将所有文件打包到 `mysqldump.zip` 内，生成可以用于创建函数的 zip 包。

创建及配置函数

通过如上步骤创建的 zip 包，由于体积稍大，需要通过对象存储 COS 的方式上传。我们需要在特定地域创建两个存储桶，一个用于上传及更新函数代码使用，一个用于存储备份的 mysql dump 文件。两个存储桶分别命名为 codefile 和 dbbackup。

然后将上一步创建的 zip 包，上传到 codefile 根目录中，作为函数创建时的代码来源。

接下来，我们开始创建函数，在特定地域下创建名为 mysqldump 的函数，运行环境选择为 Python2.7，超时时间可配置为 60 秒。接下来上传代码的位置，选择为从 COS 上传代码，选择 codefile 存储桶，并指定代码文件为 /mysqldump.zip，同时入口函数为 index.main_handler，与 index.py 文件中的 main_handler 函数对应。

同时，由于我们代码中的数据库相关配置，COS 读写相关配置，均从环境变量中读取的，因此我们也需要对函数配置上所需的环境变量，配置包括 dbhost，dbport，dbuser，dbpwd，dbname，secretid，secretkey，cosregion，cosbucket 的环境变量对应的值，使用具体的数据连接信息以及对应的 COS 读写认证相关的信息。

配置完成后，我们就完成了函数的创建。

步骤 3. 测试及启动云函数

最近更新时间：2018-09-03 15:28:35

完成函数创建后，我们就可以测试函数的运行情况，并完成最终的定时触发配置。

- 通过控制台右上角的测试按键，可以直接触发函数的运行。
- 通过函数的输出日志，可以查看代码的运行情况，检查 dump 文件是否生成正常，是否成功上传到 COS 存储桶中。
- 同时也可以到对应的备份存储桶中，查看生成的文件，检查是否数据正确，备份正常。

确认函数测试运行正常后，可以在触发器中，为函数新增一个定时触发器。我们可以根据自身需要，配置为每天，或每 12 小时，或每月的指定时间运行。定时触发器可以按 crontab 格式编写触发时间，既可以按一定时间周期，也可以指定具体时间运行。

总结

在本代码实操示例教程中，我们通过使用 mysqldump 工具，以及对象存储 COS 的 SDK，实现了数据库的按时备份能力。通过使用云函数，我们无需使用云服务器以及配置 crontab 脚本，就可以实现高可靠的定时运维能力。云函数，以及云函数的定时触发器，在运维过程中可以被广泛使用，实现例如备份、检查、告警、同步等各种能力。本教程，仅为大家提供了一种实现的参考思路，欢迎大家在这个思路继续扩展，并分享在运维过程中使用云函数的经验。

MapReduce 方法 示例说明

最近更新时间：2018-08-28 15:49:18

本教程假设以下情况：

- 您将不定时上传一些文本文件（如日志等）至某个特定的COS Bucket
- 您要对这些文本文件进行字数统计

注意：

1. 必须使用两个COS Bucket。如果使用同一个存储桶作为源和目标，上传到源存储桶的每个缩略图都会触发另一个对象创建事件，该事件将再次调用函数，从而产生无限的循环。
2. 请保证函数和 COS Bucket位于同一个地域下

实现概要

下面是该函数的实现流程：

- 创建函数与COS Bucket的事件源映射
- 用户将对象上传到 COS 中的源存储桶（对象创建事件）。
- COS Bucket检测到对象创建事件。
- COS 调用函数并将事件数据作为参数传递给函数，由此将 `cos:ObjectCreated:*` 事件发布给函数。
- SCF 平台接收到调用请求，执行函数。
- 函数通过收到的事件数据获得了 Bucket 名称和文件名称，从该源 Bucket中获取该文件，根据代码中实现的 wordcount 进行字数统计，然后将其保存到目标Bucket上。

请注意，完成本教程后，您的账户中将具有以下资源：

- 两个 SCF 函数，分别为 Mapper 和 Reducer
- 三个COS Bucket：srcmr、middlestagebucket 和 destmr
- 源 Bucket 上的通知配置：将 SCF 函数和 COS Bucket绑定将在该 Bucket 的通知配置上添加新项，用来标识 COS 将触发函数的事件类型（文件创建/删除）及要调用的函数名称。有关 COS 通知功能的更多信息，请参阅 [PutBucketNotification](#) 接口。

本教程分为了两个主要部分：

- 完成创建函数的必要设置步骤，并使用 COS 示例事件数据手动调用该函数。旨在验证函数能够正常工作。

- 向源 Bucket 添加通知配置，使得 COS 在检测到文件创建事件时能够调用函数。

步骤 1. 准备 COS Bucket

最近更新时间：2018-09-03 15:28:43

请确保您在执行此示例时，已经获得了 SCF 使用权限。

- 1) 登录腾讯云控制台，选择【对象存储服务】。
- 2) 单击【Bucket 列表】选项卡下的【创建Bucket】按钮，新建源 COS Bucket。
- 3) 设置COS Bucket的名称如 `srcmr`，选择地域为 `华南`，设置访问权限为默认值 `公有读私有写` 并设置CDN加速为默认值 `关闭`，单击【保存】按钮新建一个COS Bucket。
- 4) 按照相同的方式创建中间阶段 Bucket `middlestagebucket` 和目标 Bucket `destmr`
- 5) 在源 Bucket (即 `srcmr`) 中，上传一个文本文件，本示例中使用了一个 `Serverless.txt` 文本文件作为演示。（在实际关联 COS 前手动调用函数进行测试验证时，您要将包含该文件的示例数据传递给 SCF 函数，且 SCF 函数将根据该数据寻找相应的文件。因此您需要先创建此示例文件。）

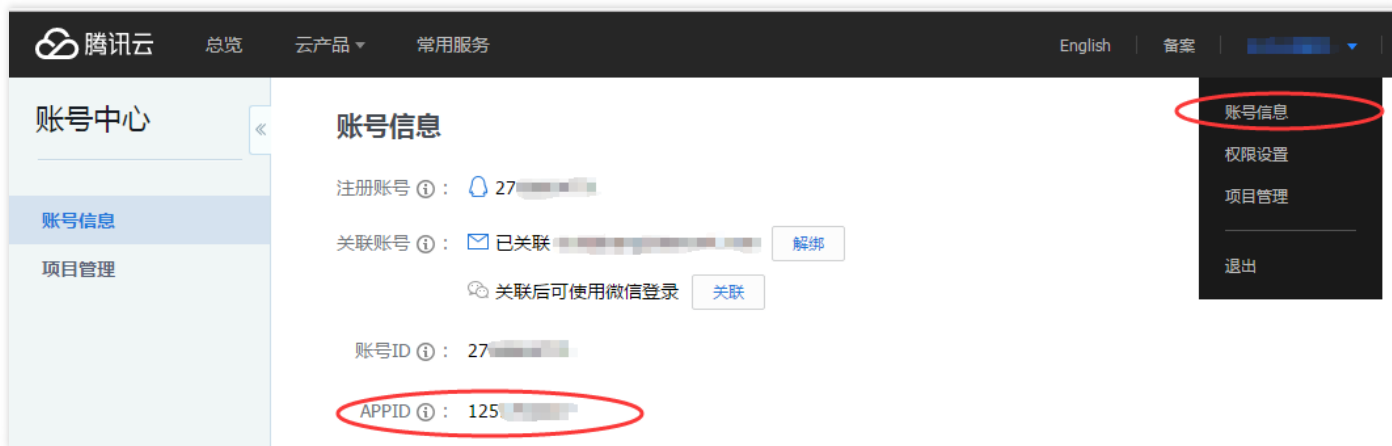


步骤 2. 创建部署程序包

最近更新时间：2018-09-03 15:28:48

创建 Mapper 部署程序包

- 1) 请在任意位置新建一个名为 WordCount 的文件夹
- 2) 新建一个名为 map_function 的 .py 文件，写入下面的代码并保存。请特别注意，将参数 appid, secret_id, secret_key, region 改为您的实际数据，其中：
 - appid可在控制台【账户信息】中获得



- secret_id 和 secret_key可在控制台【云 API 密钥】中获得



- region 为 函数 和 COS Bucket 所在地域，支持 sh、gz、bj 三个值。请注意，必须保持和上一步骤中创建的 COS Bucket 在同一个地域。此处由于步骤一：准备COS Bucket 中创建的存储桶位于华南（广州），因此代码中

的 region 值必须为 gz :

```
import mapper_triggered as Mapper
import datetime
from qcloud_cos import CosClient

def map_caller(event, context):
    appid = 1251762222 # change to user's appid
    secret_id = u'AKIDYDh085xQp48161uOn2CKKVbeebvDu6EE' # change to user's secret_id
    secret_key = u'ILkxx40kIfuyqW0IOI0WqyueCYjlgZEE' # change to user's secret_key
    region = u'gz' # change to user's region
    cos_client = CosClient(appid, secret_id, secret_key, region)

    bucket = event['Records'][0]['cos']['cosBucket']['name']
    key = event['Records'][0]['cos']['cosObject']['key']
    middle_stage_bucket = u'middlestagebucket'
    middle_file_key = '/' + 'middle_' + key.split('/')[-1]

    return Mapper.do_mapping(cos_client, bucket, key, middle_stage_bucket, middle_file_key)

def main_handler(event, context):
    start_time = datetime.datetime.now()
    res = map_caller(event, context)
    end_time = datetime.datetime.now()
    print("data mapping duration: " + str((end_time-start_time).microseconds/1000) + "ms")
    if res == 0:
        return "Data mapping SUCCESS"
    else:
        return "Data mapping FAILED"
```

创建完成后，在 相同路径下，创建名为 mapper_triggered 的 .py 文件，写入下面的代码并保存：

```
from qcloud_cos import UploadFileRequest
from qcloud_cos import DownloadFileRequest
import re
import os
import logging

logger = logging.getLogger()

#delete folders and files
def delete_file_folder(src):
    if os.path.isfile(src):
```

```
try:
    os.remove(src)
except:
    pass
elif os.path.isdir(src):
    for item in os.listdir(src):
        itemsrc=os.path.join(src,item)
        delete_file_folder(itemsrc)
try:
    os.rmdir(src)
except:
    pass

# Download files
def download_file(cos_client, bucket, key, local_file_path):
    request = DownloadFileRequest(bucket, key, local_file_path)
    download_file_ret = cos_client.download_file(request)
    if download_file_ret['code'] == 0:
        logger.info("Download file [%s] Success" % key)
        return 0
    else:
        logger.error("Download file [%s] Failed, err: %s" % (key, download_file_ret['message']))
        return -1

# Upload file to bucket
def upload_file(cos_client, bucket, key, local_file_path):
    request = UploadFileRequest(bucket.decode('utf-8'), key.decode('utf-8'), local_file_path.decode('utf-8'))
    upload_file_ret = cos_client.upload_file(request)
    if upload_file_ret['code'] == 0:
        logger.info("Upload data map file [%s] Success" % key)
        return 0
    else:
        logger.error("Upload data map file [%s] Failed, err: %s" % (key, upload_file_ret['message']))
        return -1

# domapping
def do_mapping(cos_client, bucket, key, middle_stage_bucket, middle_file_key):
    src_file_path = u'/tmp/' + key.split('/')[1]
    middle_file_path = u'/tmp/' + u'mapped_' + key.split('/')[1]
    download_ret = download_file(cos_client, bucket, key, src_file_path) #download src file
    if download_ret == 0:
        inputfile = open(src_file_path, 'r') #open local /tmp file
        mapfile = open(middle_file_path, 'w') #open a new file write stream
```

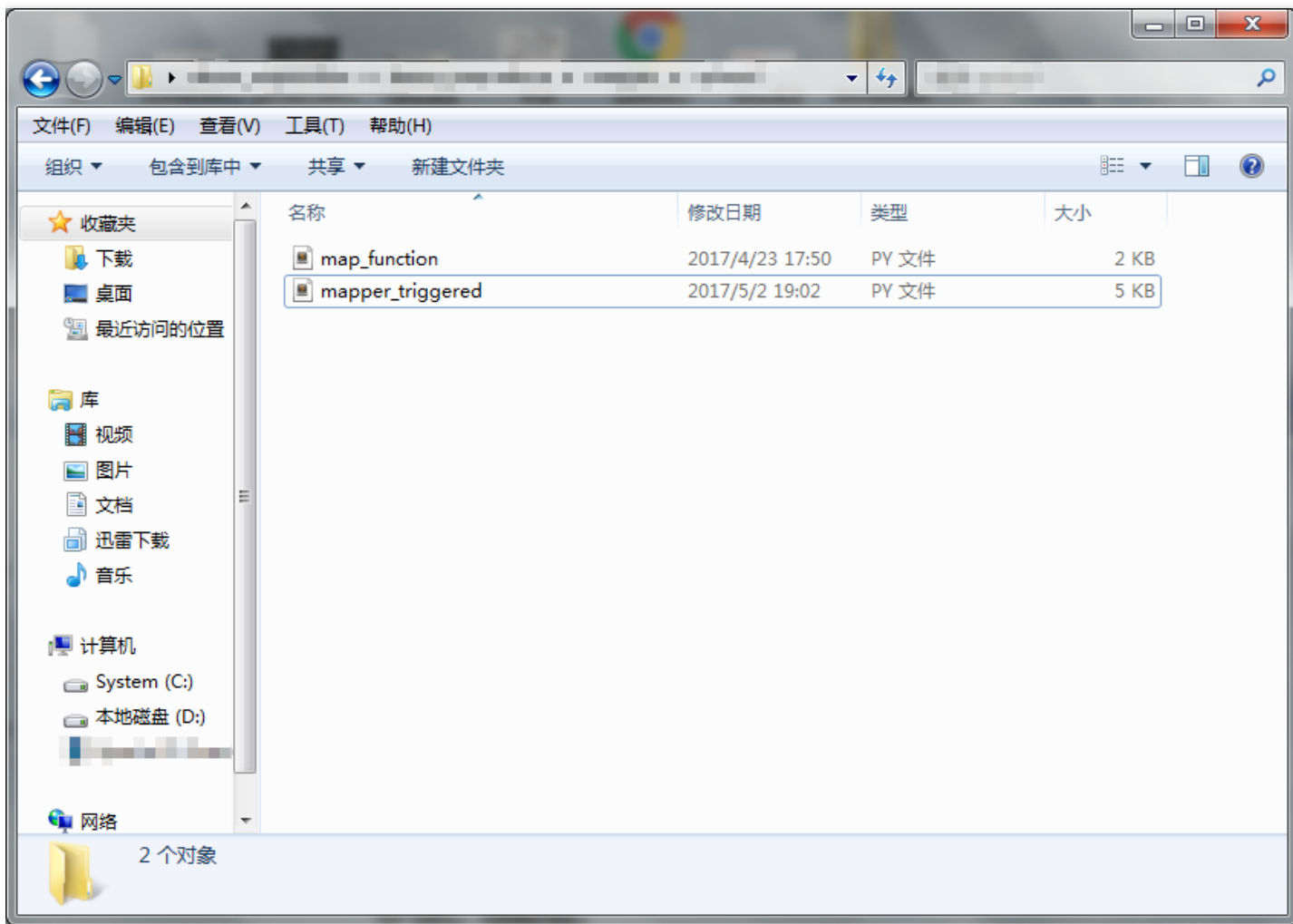
```
for line in inputfile:
line = re.sub('[^a-zA-Z0-9]', ' ', line) #replace non-alphabetic/number characters
words = line.split()
for word in words:
mapfile.write('%s\t%s' % (word, 1)) #count for 1
mapfile.write('\n')

inputfile.close()
mapfile.close()

upload_ret = upload_file(cos_client, middle_stage_bucket, middle_file_key, middle_file_path) #upload
the file's each word

delete_file_folder(src_file_path)
delete_file_folder(middle_file_path)
return upload_ret
else:
return -1
```

3) 如果本地是 Windows 环境，此时应该在该路径下看到两个 py 文件，形如下图：



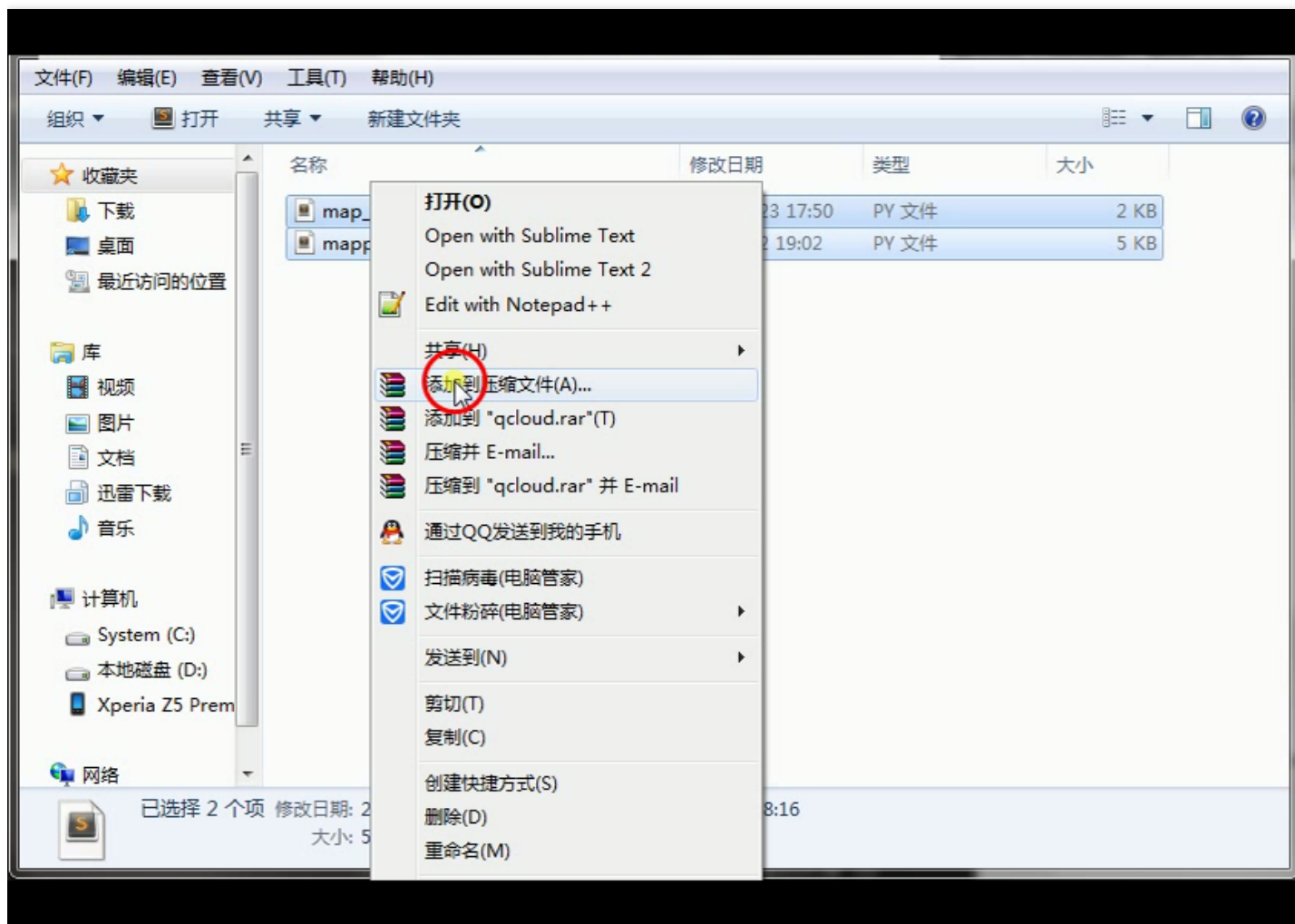
如果本地是 Linux 环境，此时应该在该路径下看到两个 py 文件，形如下图：

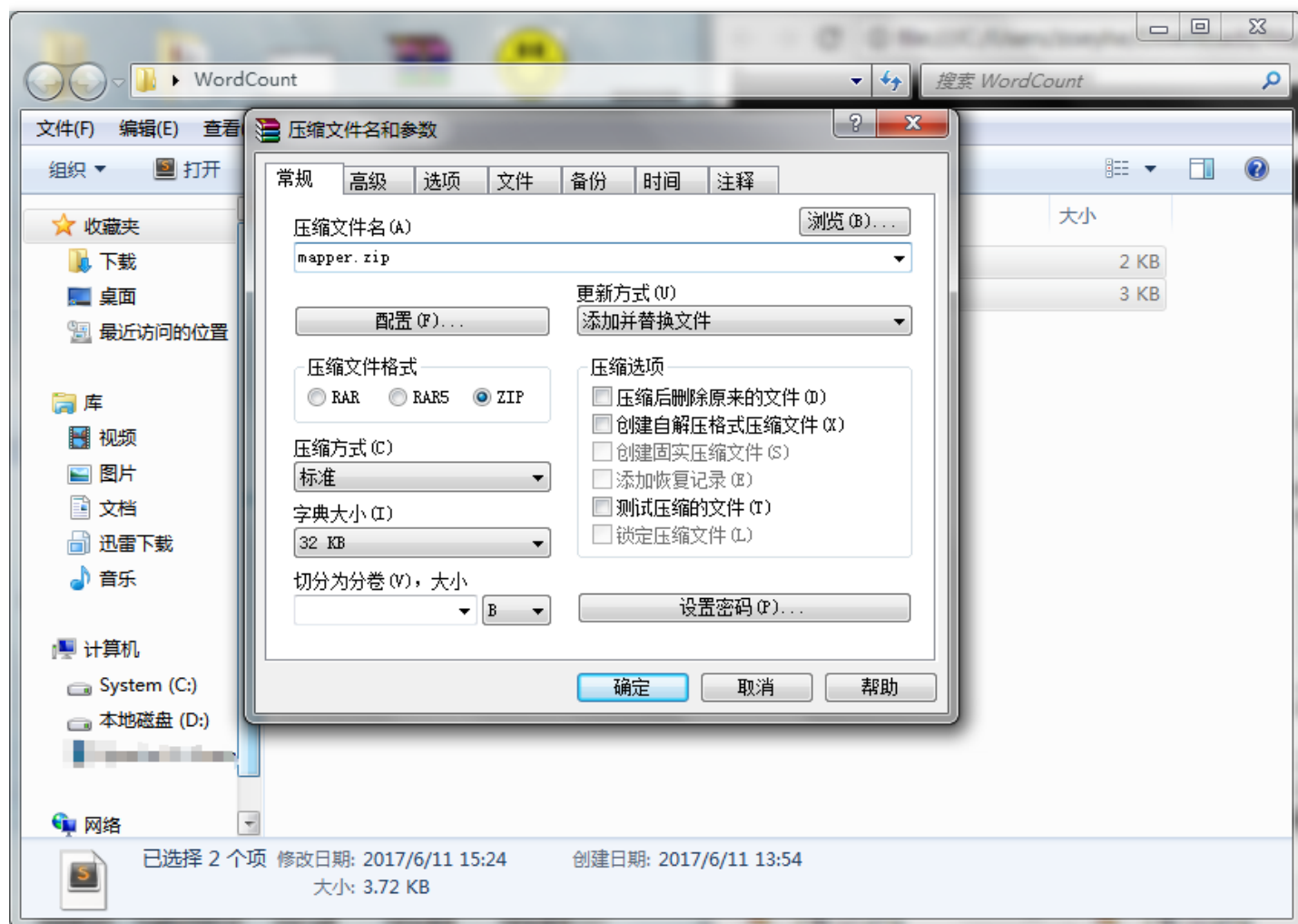
```
[root@..._centos ...test]# ls
map_function.py  mapper_triggered.py
```

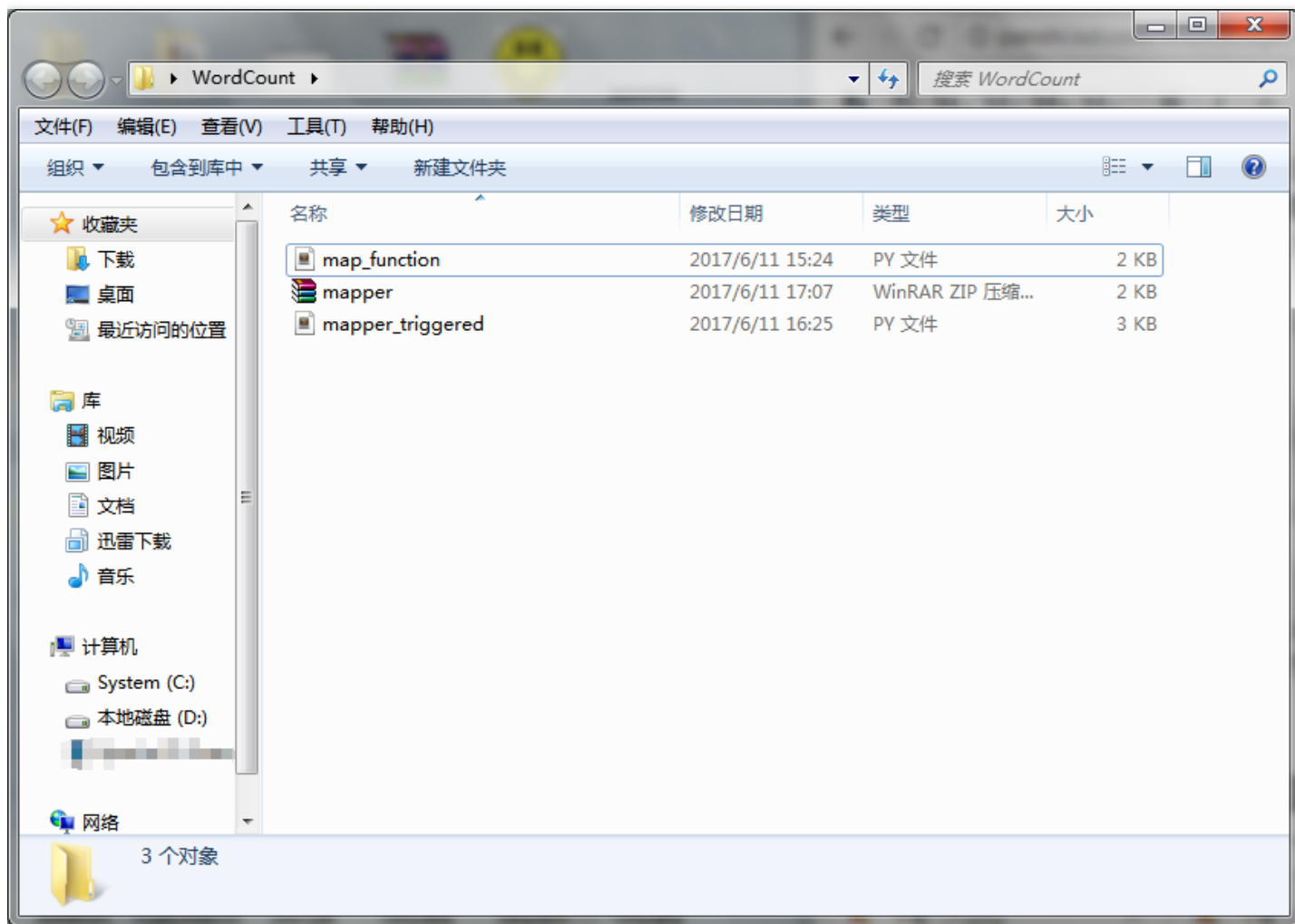
将这两个文件压缩成 zip 包，命名为 mapper（特别注意：是将文件直接压缩，而不是压缩文件所在的文件夹）。

Windows 环境下：

选中这两个文件，单击右键，选择您的压缩工具如 winrar，单击【添加到压缩文件...】，将压缩文件设置为 zip 格式，单击【确定】按钮，将生成一个 zip 包。







Linux 环境下：

直接进入该目录，运行命令

```
cd /WordCount
zip mapper.zip map_function.py mapper_triggered.py
```

创建 Reducer 部署程序包

1) 同样地，在 WordCount 目录下新建一个名为 `reduce_function` 的 .py 文件，写入下面的代码并保存。请特别注意，将参数 `appid`, `secret_id`, `secret_key`, `region` 改为您的实际数据，其中：

- appid 可在控制台【账户信息】中获得



- secret_id 和 secret_key 可在控制台【云 API 密钥】中获得



- region 为 函数 和 COS Bucket 所在地域，支持 sh、gz、bj 三个值。请注意，必须保持和上一步骤中创建的 COS Bucket 在同一个地域。此处由于步骤一：准备 COS Bucket 中创建的存储桶位于华南（广州），因此代码中的 region 值必须为 gz：

```
import reducer_triggered as Reducer
import datetime
from qcloud_cos import CosClient

def reduce_caller(event, context):
    appid = 1251762222 # change to user's appid
    secret_id = u'AKIDYDh085xQp48161uOn2CKKVbeebvDu6EE' # change to user's secret_id
    secret_key = u'ILkxx40klfuyqW0IOI0WqyueCYjlgZEE' # change to user's secret_key
    region = u'gz' # change to user's region
```

```
cos_client = CosClient(appid, secret_id, secret_key, region)

bucket = event['Records'][0]['cos']['cosBucket']['name']
key = event['Records'][0]['cos']['cosObject']['key']
result_bucket = u'destmr'
result_key = '/' + 'result_' + key.split('/')[-1]

return Reducer.qcloud_reducer(cos_client, bucket, key, result_bucket, result_key)

def main_handler(event, context):
    start_time = datetime.datetime.now()
    res = reduce_caller(event, context)
    end_time = datetime.datetime.now()
    print("data reducing duration: " + str((end_time-start_time).microseconds/1000) + "ms")
    if res == 0:
        return "Data reducing SUCCESS"
    else:
        return "Data reducing FAILED"
```

创建完成后，在 相同路径下，创建名为 `reducer_triggered` 的 .py 文件，写入下面的代码并保存：

```
from qcloud_cos import UploadFileRequest
from qcloud_cos import DownloadFileRequest
import os
import logging
from operator import itemgetter

logger = logging.getLogger()

#delete folders and files
def delete_file_folder(src):
    if os.path.isfile(src):
        try:
            os.remove(src)
        except:
            pass
    elif os.path.isdir(src):
        for item in os.listdir(src):
            itemsrc=os.path.join(src,item)
            delete_file_folder(itemsrc)
        try:
            os.rmdir(src)
        except:
            pass
```

Download files

```
def download_file(cos_client, bucket, key, local_file_path):
    request = DownloadFileRequest(bucket, key, local_file_path)
    download_file_ret = cos_client.download_file(request)
    if download_file_ret['code'] == 0:
        logger.info("Download file [%s] Success" % key)
        return 0
    else:
        logger.error("Download file [%s] Failed, err: %s" % (key, download_file_ret['message']))
        return -1
```

Upload file to bucket

```
def upload_file(cos_client, bucket, key, local_file_path):
    request = UploadFileRequest(bucket.decode('utf-8'), key.decode('utf-8'), local_file_path.decode('utf-8'))
    upload_file_ret = cos_client.upload_file(request)
    if upload_file_ret['code'] == 0:
        logger.info("Upload data map file [%s] Success" % key)
        return 0
    else:
        logger.error("Upload data map file [%s] Failed, err: %s" % (key, upload_file_ret['message']))
        return -1
```

doreducing

```
def qcloud_reducer(cos_client, bucket, key, result_bucket, result_key):
    word2count = {}
    src_file_path = u'/tmp/' + key.split('/')[1]
    result_file_path = u'/tmp/' + u'result_' + key.split('/')[1]
    download_ret = download_file(cos_client, bucket, key, src_file_path)
    if download_ret == 0:
        map_file = open(src_file_path, 'r')
        result_file = open(result_file_path, 'w')

        for line in map_file:
            line = line.strip()
            word, count = line.split('\t', 1)
            try:
                count = int(count)
                word2count[word] = word2count.get(word, 0) + count
            except ValueError:
                logger.error("error value: %s, current line: %s" % (ValueError, line))
                continue
        map_file.close()
        delete_file_folder(src_file_path)
```

```
sorted_word2count = sorted(word2count.items(), key=itemgetter(1))[:-1]
for wordcount in sorted_word2count:
    res = '%s\t%s' % (wordcount[0], wordcount[1])
    result_file.write(res)
    result_file.write('\n')
    result_file.close()

upload_ret = upload_file(cos_client, result_bucket, result_key, result_file_path)
delete_file_folder(result_file_path)
return upload_ret
```

3) 按上面的方法将这两个文件压缩成另一个 zip 包，命名为 reducer（特别注意：是将文件直接压缩，而不是压缩文件所在的文件夹）。

步骤 3. 创建 Mapper 和 Reducer 函数并测试

最近更新时间：2018-09-03 15:29:27

在此部分中，用户将创建两个函数来实现简单的WordCount，并通过控制台/API调用来测试函数。

创建 Mapper 函数

通过控制台创建函数

- 1) 登录[无服务器云函数控制台](#)，在【广州】地域下单击【新建】按钮；
- 2) 进入函数配置部分，函数名称填写 `Mapper`，剩余项保持默认，单击【下一步】；
- 3) 进入函数代码部分，选择【本地上传zip包】。执行方法填写 `map_function.main_handler`，选择步骤二：创建部署程序包中创建的 `mapper.zip`，单击【下一步】；
- 4) 进入触发方式部分，此时由于需要先手动测试函数，暂时不添加任何触发方式，单击【完成】按钮。

通过 API 创建函数

请参考 `CreateFunction` 接口

创建 Reducer 函数

通过控制台创建函数

- 1) 登录[无服务器云函数控制台](#)，在【广州】地域下单击【新建】按钮；
- 2) 进入函数配置部分，函数名称填写 `Reducer`，剩余项保持默认，单击【下一步】；
- 3) 进入函数代码部分，选择【本地上传zip包】。执行方法填写 `reduce_function.main_handler`，选择步骤二：创建部署程序包中创建的 `reducer.zip`，单击【下一步】；
- 4) 进入触发方式部分，此时由于需要先手动测试函数，暂时不添加任何触发方式，单击【完成】按钮。

通过 API 创建函数

请参考 `CreateFunction` 接口

测试函数

在创建函数时，通常会使用控制台或 API 先进行测试，确保函数输出符合预期后再绑定触发器进行实际应用。

使用控制台测试函数

- 1) 在刚刚创建的 Mapper 函数详情页中，单击【测试】按钮；
- 2) 在测试模版下拉列表中选择【COS 上传/删除文件测试代码】
- 3) 轻微改动测试代码，将 `name` 设置为步骤一：准备 COS Bucket 中创建的 `srcmr` 存储桶名称，将 `key` 设置为步骤一：准备 COS Bucket 中上传的 `/serverless.txt` 键值，如下面示例：

```
{
  "Records":[
    {
      "event": {
        "eventVersion":"1.0",
        "eventSource":"qcs::cos",
        "eventName":"event-type",
        "eventTime":"Unix 时间戳",
        "eventQueue":"qcs:0:cos:gz:1251111111:cos",
        "requestParameters":{
          "requestSourceIP": "111.111.111.111",
          "requestHeaders":{
            "Authorization": "上传的鉴权信息"
          }
        }
      },
      "cos":{
        "cosSchemaVersion":"1.0",
        "cosNotificationId":"设置的或返回的 ID",
        "cosBucket":{
          "name":"srcmr", #set to demo bucket here
          "appid":"appld",
          "region":"gz"
        },
        "cosObject":{
          "key":"/serverless.txt", #set to demo file here
          "size":"1024",
          "meta":{
            "Content-Type": "text/plain",
            "x-cos-meta-test": "自定义的 meta",
            "x-image-test": "自定义的 meta"
          },
          "url": "访问文件的源站url"
        }
      }
    }
  ]
}
```

```
}  
]  
}
```

4) 单击【运行】按钮，观察运行结果。

5) 前往[对象存储控制台](#)，单击步骤一：准备 COS Bucket 中创建的 `destmr`，观察该 COS Bucket 中是否有 `result_middle_serverless.txt` 文件生成，该文件中统计了刚刚上传的文本文件中文章各个单词出现的次数：

6) 下载该文件，文件内容应该类似如下：

...

```
the 29  
as 25  
to 20  
of 17  
and 16  
a 16  
code 16  
in 15  
be 14  
or 12  
serverless 10  
is 10  
that 10  
computing 9  
by 9  
for 9  
an 8  
not 8  
cloud 7  
functions 6  
edit 6  
can 6  
with 5  
servers 5  
on 5  
because 5  
at 5  
Serverless 5  
have 5
```

platform 5
such 5
time 4
virtual 4
up 4
function 4
provider 4
autoscaling 4
used 4
it 4
js 4
are 4
also 4
than 4
service 4
written 4
2016 4
1 4
runtime 4
example 4
use 4
Node 4
run 3
does 3
Java 3
execution 3
other 3
model 3
In 3
requests 3
means 3
typically 3
which 3
latency 3
any 3
could 3
This 3
may 3

via 3
resources 3
required 2
10 2
developers 2
application 2
underutilisation 2
hosted 2
was 2
start 2
support 2
microservices 2
per 2
back 2
server 2
source 2
generally 2
It 2
well 2
user 2
running 2
Python 2
handling 2
JSON 2
addition 2
limits 2
both 2
available 2
For 2
request 2
The 2
efficient 2
part 2
2 2
system 2
APIs 2
first 2
completely 2

supports 2
container 2
programmer 2
API 2
name 2
provision 2
now 2
they 2
3 2
its 2
operations 2
machine 2
end 2
more 2
from 2
public 2
officially 2
even 2
cost 2
Functions 2
all 2
requires 1
At 1
2006 1
starting 1
serialized 1
calls 1
assets 1
InterConnect 1
exposure 1
needed 1
included 1
suited 1
includes 1
you 1
5 1
setting 1
when 1

no 1
periods 1
abstract 1
defined 1
called 1
continuously 1
simplifying 1
Infrequently 1
but 1
triggered 1
significantly 1
registration 1
down 1
functional 1
into 1
own 1
software 1
automatically 1
introduced 1
about 1
although 1
processing 1
business 1
However 1
latencies 1
some 1
creation 1
exposed 1
spins 1
either 1
specific 1
necessary 1
using 1
rules 1
usable 1
has 1
Despite 1
4 1

web 1
world 1
being 1
just 1
spend 1
without 1
person 1
task 1
provisioned 1
uses 1
Resource 1
meets 1
development 1
composition 1
units 1
make 1
pay 1
if 1
worry 1
technology 1
launched 1
stopping 1
owns 1
policies 1
will 1
batch 1
unlike 1
case 1
demand 1
ensuring 1
bulk 1
containers 1
offers 1
been 1
billing 1
tuning 1
GitHub 1
sequences 1

simply 1
simple 1
point 1
consume 1
Monitoring 1
s 1
sensitive 1
triggers 1
essentially 1
packages 1
premise 1
Programming 1
initially 1
terms 1
C 1
invoker 1
behind 1
capacity 1
Performance 1
configured 1
wrote 1
where 1
fixed 1
architecture 1
imposed 1
Swift 1
resource 1
infrequent 1
conjunction 1
cheaper 1
do 1
8 1
black 1
only 1
outside 1
renting 1
quantity 1
Cloud 1

Cost 1

traditional 1

considered 1

alpha 1

significant 1

performance 1

online 1

purchasing 1

responsible 1

offering 1

option 1

need 1

rent 1

debugging 1

jobs 1

PaaS 1

11 1

specialists 1

released 1

serve 1

Zimki 1

structures 1

their 1

management 1

given 1

open 1

directly 1

instances 1

number 1

features 1

major 1

launch 1

language 1

multithreading 1

announced 1

7 1

since 1

measure 1

another 1
events 1
fully 1
9 1
introduce 1
Alternatively 1
greater 1
affected 1
non 1
might 1
2014 1
systems 1
production 1
actually 1
group 1
style 1
suffer 1
including 1
order 1
allow 1
Haskell 1
purchase 1
response 1
data 1
designed 1
produce 1
REST 1
provisioning 1
rather 1
involve 1
providers 1
dedicated 1
believed 1
this 1
hour 1
endpoint 1
high 1
billed 1

known 1
expose 1
sharing 1
6 1
Microsoft 1
would 1
environment 1
private 1
followed 1
FaaS 1
involves 1
amount 1
deserialized 1
box 1
groups 1
supporting 1
satisfy 1
new 1
version 1
Disadvantages 1
machines 1
likely 1
cron 1
HTTP 1
workloads 1
small 1
level 1
announcing 1
include 1
languages 1
Docker 1
go 1
manages 1

步骤 4. 添加触发器

最近更新时间：2018-09-03 15:29:32

如果您完成了步骤三：创建 Mapper 和 Reducer 函数并测试，且测试结果符合预期，则您可以添加 COS 配置以便 COS 能向 SCF 发布事件并调用函数。

- 1) 在刚刚创建的 Mapper 函数详情页中，选择【触发方式】选项卡，单击【添加触发方式】按钮；
- 2) 选择 COS 触发，COS Bucket 选择步骤一：准备 COS Bucket 中创建的 srcmr，事件类型选择“文件上传”，单击【保存】按钮；

此时本示例全部完成！现在您可以按以下方式测试设置：

1. 前往[对象存储控制台](#)，选择 src，上传任意的 .txt 文本文件，一段时间后观察 destmr 中是否有类似文件生成；
2. 您可以在[无服务器云函数控制台](#)中监控函数的活动，选择【日志】选项来查看函数被调用的日志记录。

根据 CMQ 中的消息发送邮件 示例说明

最近更新时间：2018-08-29 17:09:24

本教程假设以下情况：

- 您的系统需要在某些情况下发送邮件
- 您希望采用消息队列来收集和传递需要发送的内容和接收者

实现概要

下面是该函数的实现流程：

- 创建函数与CMQ Topic队列的事件源映射。
- 用户将需要发送的邮件内容和邮件接收者以特定数据格式发送到消息队列中。
- CMQ 队列将会触发 SCF 云函数的运行，把消息以事件的格式传递给函数。
- SCF 平台接收到调用请求，执行函数。
- 函数通过收到的事件数据获得需要发送的邮件内容、邮件接收者，调用邮件发送接口，发送邮件。

请注意，完成本教程后，您的账户中将具有以下资源：

- 一个发送邮件的 SCF 函数。
- 一个CMQ Topic 主题队列。
- CMQ 主题队列上的订阅配置。

本教程分为了三个主要部分：

- 完成 CMQ 主题队列的创建。
- 完成创建函数的必要设置步骤，并使用 CMQ 示例事件数据手动调用该函数。旨在验证函数能够正常工作。
- 完成 CMQ 主题队列和函数的绑定，并通过队列的发送消息接口，测试 CMQ 队列和 SCF 云函数的联动能力，使得 CMQ 队列在接收到消息时能够调用函数。

数据结构设计

假设用于发送邮件的数据结构如下所示，此数据结构将在根据需要填充数值后发送至 CMQ 队列中并由 SCF 函数接受并处理，发送邮件。

```
{  
  "fromAddr": "sender@testhost.com",  
  "toAddr": "test@testhost.com",  
  "title": "hello from scf & cmq",  
  "body": "email content to send"  
}
```

步骤 1. 创建 CMQ Topic 主题模式队列

最近更新时间：2018-09-03 15:29:39

注意：

CMQ 队列和函数必须位于同一个地域下。在本教程中，我们将使用华南（广州）区域。

1. 登录[腾讯云控制台](#)，从云产品中选择【消息服务 CMQ】。
2. 单击【主题订阅】选项卡，并切换地域为【华南地区（广州）】。
3. 单击【新建】按钮以新建队列，在弹出窗口中写入主题名 `sendEmailQueue`。
4. 单击【创建】，完成队列创建。

步骤 2. 创建并测试 sendEmail 函数

最近更新时间：2018-09-03 15:29:44

在此部分中，用户将创建一个函数来实现发送邮件程序，并通过控制台/API调用来测试函数。

创建 sendEmail 云函数

1. 登录[无服务器云函数控制台](#)，在【广州】地域下单击【新建】按钮。
2. 进入函数配置部分，函数名称填写 `sendEmail`，剩余项保持默认，单击【下一步】。
3. 进入函数代码部分，执行方法填写 `index.main_handler`，代码窗口内贴入如下代码，单击【下一步】。

```
# -*- coding: utf8 -*-
import json
import smtplib
from email.mime.text import MIMEText
from email.header import Header

# 第三方 SMTP 服务
mail_host="smtp.qq.com" #SMTP服务器
mail_user="3473058547@qq.com" #用户名
mail_pass="xxxxxxx" #口令
mail_port=465 #SMTP服务端口

def sendEmail(fromAddr,toAddr,subject,content):
    sender = fromAddr
    receivers = [toAddr] # 接收邮件，可设置为您的QQ邮箱或者其他邮箱

    message = MIMEText(content, 'plain', 'utf-8')
    message['From'] = Header(fromAddr, 'utf-8')
    message['To'] = Header(toAddr, 'utf-8')
    message['Subject'] = Header(subject, 'utf-8')

    try:
        smtpObj = smtplib.SMTP_SSL(mail_host, mail_port)
        smtpObj.login(mail_user,mail_pass)
        smtpObj.sendmail(sender, receivers, message.as_string())
        print("send success")
    except smtplib.SMTPException as e:
```

```
print(e)
print("Error: send fail")

def main_handler(event, context):
    cmqMsg = None
    if event is not None and "Records" in event.keys():
        if len(event["Records"]) >= 1 and "CMQ" in event["Records"][0].keys():
            cmqMsgStr = event["Records"][0]["CMQ"]["msgBody"]
            cmqMsg = json.loads(cmqMsgStr)
            print cmqMsg
            sendEmail(cmqMsg['fromAddr'], cmqMsg['toAddr'], cmqMsg['title'], cmqMsg['body'])
            return "send email success"
```

4) 进入触发方式部分，此时由于需要先手动测试函数，暂时不添加任何触发方式，单击【完成】按钮。

说明

请特别注意，参数 `mail_host`, `mail_user`, `mail_pass`, `mail_port` 需要您根据所期望发送的邮箱或邮件服务器来配置，这里我们以 QQ 邮箱为例，您可以从 [这里](#) 了解到如何开启 QQ 邮箱的 SMTP 功能。QQ 邮箱的 SMTP 功能开启后，相应的参数如下。

- `mail_host` SMTP服务器地址为 "smtp.qq.com"
- `mail_user` 登录用户名为您的邮箱地址，例如 3473058547@qq.com
- `mail_pass` 为您在开启 SMTP 功能时设置的密码
- `mail_port` 为服务器登录端口，由于 QQ 邮箱强制要求SSL登录，端口固定为 465，同时代码中使用 `smtplib.SMTP_SSL` 创建 SSL 的 SMTP 连接

测试 sendEmail 云函数

在创建函数时，通常会使用控制台或 API 先进行测试，确保函数输出符合预期后再绑定触发器进行实际应用。

1) 在刚刚创建的 sendEmail 函数详情页中，单击【测试】按钮；

2) 在测试模版内输入如下内容：

```
{
  "Records": [
    {
      "CMQ": {
        "type": "topic",
        "topicOwner": "1253970226",
        "topicName": "sendEmailQueue",
        "subscriptionName": "sendEmailFunction",
        "publishTime": "2017-09-25T06:34:00.000Z",
```

```
"msgId": "123345346",
"requestId": "123345346",
"msgBody": "{\n  \"fromAddr\": \"3473058547@qq.com\", \"toAddr\": \"3473058547@qq.com\", \"title\": \"hello from scf & cmq\", \"body\": \"email content to send\"}",
"msgTag": []
}
}
```

其中 msgBody 字段内，fromAddr，toAddr 内的字段，可以根据您自身邮箱地址进行修改，建议可以修改为相同地址，自身邮箱向自身邮箱内发送邮件，以便测试邮件发送的正确性。我们在这里使用了 3473058547@qq.com 这个邮箱来进行测试。

3) 单击【运行】按钮，观察运行结果。如果在结果中发现返回值和日志中均显示 "send email success"，则此程序运行正常。

sendEmail

函数配置 函数代码 触发方式 日志 监控

代码输入种类 ☒ 在线编辑 ☐ 本地上传zip包 ☐ 通过COS上传zip包

执行方法

```
27 except smtplib.SMTPException as e:
28     print(e)
29     print("Error: send fail")
30
31 def main_handler(event, context):
32     cmqMsg = None
33     if event is not None and "Records" in event.keys():
34         if len(event["Records"]) >= 1 and "CMQ" in event["Records"][0].keys():
35             cmqMsgStr = event["Records"][0]["CMQ"]["msgBody"]
36             cmqMsg = json.loads(cmqMsgStr)
37             print cmqMsg
38             sendEmail(cmqMsg['fromAddr'], cmqMsg['toAddr'], cmqMsg['title'], cmqMsg['body'])
39             return "send email success"
```

保存

测试函数

测试事件模板

```
9 successCallback = sendEmailFunction,
10 "publishTime": "2017-09-25T06:34:00.000Z",
11 "msgId": "123345346",
12 "requestId": "123345346",
13 "msgBody": "{\n  \"fromAddr\": \"56179642@qq.com\", \"toAddr\": \"56179642@qq.com\", \"title\": \"hello from scf & cmq\", \"body\": \"email content to send\"}",
14 "msgTag": []
15 }
16 ]
17 }
```

运行

测试结果: 成功

返回结果
send email success

摘要
请求ID: dd4e9e2b-a1c6-11e7-a46f-5254001df6c6
运行时间: 1017.72ms
计费时间: 1100ms
占用内存: 0.473MB

日志
{u'body': u'email content to send', u'fromAddr': u'56179642@qq.com', u'toAddr': u'56179642@qq.com', u'title': u'hello from scf & cmq'}
send success

4) 前往个人配置的接收邮箱，查收是否收取到邮件。打开邮件，查看邮件内容是否为配置的内容。

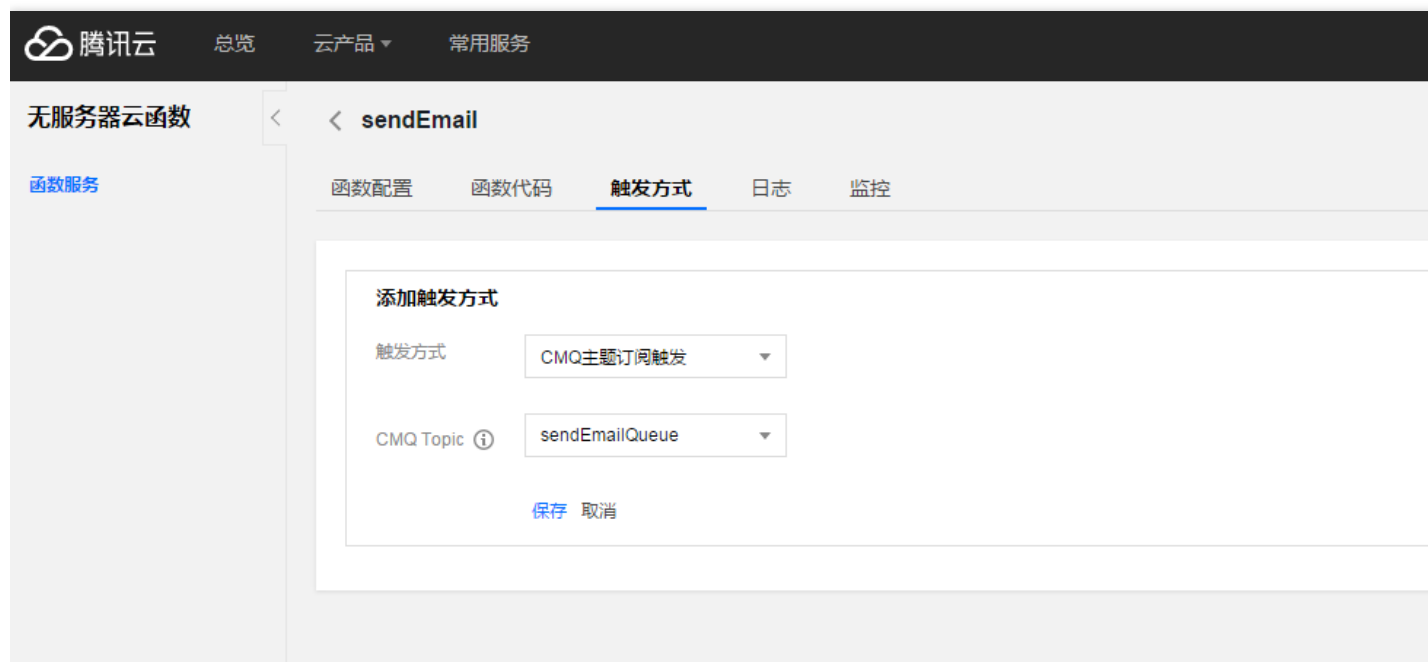


步骤 3. 添加触发器并测试

最近更新时间：2018-09-03 15:29:49

如果您完成了步骤二：创建 sendEmail 函数并测试，且测试结果符合预期，则您可以添加 CMQ Topic 队列触发配置以便 CMQ Topic 队列能向 SCF 传递消息事件并调用函数。

1. 在刚刚创建的 sendEmail 函数详情页中，选择【触发方式】选项卡，单击【添加触发方式】按钮。
2. 选择【CMQ主题订阅触发】，CMQ Topic 选择[步骤一：创建 CMQ Topic 主题模式队列](#)中创建的 sendEmailQueue，单击【保存】按钮。



此时本示例全部完成！现在您可以按以下方式测试设置：

1. 前往[消息服务CMQ](#)，在左侧栏选择【主题订阅】后，从列表中找到创建好的队列 sendEmailQueue，单击此队列提供的操作 发送消息，并在弹出的窗口中输入如下消息。
您需要根据自身情况修改消息内的内容，包括发送邮箱，接收邮箱，邮件标题，邮件内容等信息。

```
{
  "fromAddr": "xxx@qq.com",
  "toAddr": "xxx@qq.com",
  "title": "hello from scf & cmq",
```

```
"body": "email content to send"
}
```

2. 在[无服务器云函数控制台](#)中找到您所创建的 `sendEmail` 函数，监控函数活动，选择【日志】选项来查看函数被调用的日志记录。
3. 登录进入您的收件邮箱，查询是否收到邮件，邮件内容是否正确。

完成测试后，您可以在您的应用代码中嵌入 CMQ SDK，对 `sendEmailQueue` 这个队列发送在示例说明中所定义的消息，来完成邮件发送。

使用 AI 接口实现身份证识别

最近更新时间：2018-09-03 20:04:21

概述

步骤 1. 创建 COS Bucket

登录 [对象存储控制台](#) 创建一个 Bucket，我们命名为 idcard-detect，并选择北京地域，权限选择“公有读-私有写”。

创建存储桶

名称

idcard-detect

-125



仅支持小写字母、数字和 - 的组合，不能超过40字符

所属地域

北京

与相同地域其他腾讯云服务内网互通，创建后不可更改地域

访问权限



私有读写



公有读私有写



公有读写

可对object进行匿名读操作，写操作需要进行身份验证。

请求域名

idcard-detect-125.cos.ap-beijing.myqcloud.com

创建完成后，您可以使用该域名对存储桶进行访问

确定

取消

步骤 2. 开通 AI 接口

前往 [智能图像控制台](#) 开通身份证识别功能，单击【开通服务】即可。

步骤 3. 创建云函数 SCF

创建准备

1. 首先确保在您的系统中已经安装好了 Python 运行环境和 pip 工具，然后在本地创建需要放置函数代码的文件夹，并通过命令行进入该目录下，直接执行命令安装 COS V5 SDK、文字识别 SDK 和 Pillow 库：

```
pip install cos-python-sdk-v5 -t .  
pip install qcloud_image -t .  
pip install Pillow -t .
```

2. 在函数代码文件夹的根目录下，使用下面的示例代码创建 Python 文件，可以命名为：idcard_detect.py。
3. Python 文件保存后，回到根目录下，对所有文件进行打包（**请注意，不是对外层的文件夹打包**）；另外还需要保证：**idcard_detect.py 存在于根目录下，压缩包需要为 zip 格式。**

示例代码

```
# -*- coding: utf-8 -*-  
import json  
import os  
import logging  
import datetime  
from qcloud_cos import CosConfig  
from qcloud_cos import CosS3Client  
from qcloud_cos import CosServiceError  
from qcloud_image import Client  
from qcloud_image import CIUrl, CIFile, CIBuffer, CIUrls, CIFiles, CIBuffers  
from PIL import Image  
import PIL.Image  
import sys  
  
logging.basicConfig(level=logging.INFO, stream=sys.stdout)  
  
appid = **** # 请替换为您的 APPID  
secret_id = u'****' # 请替换为您的 SecretId  
secret_key = u'****' # 请替换为您的 SecretKey  
region = u'ap-beijing'  
token = ''  
  
config = CosConfig(Secret_id=secret_id, Secret_key=secret_key, Region=region, Token=token)  
client = CosS3Client(config)  
logger = logging.getLogger()  
  
def resize_image(image_path, resized_path):  
    with Image.open(image_path) as image:  
        size = float(os.path.getsize(image_path) / 1024 / 1024)  
        print size
```

```
if size >= 1:
    image.thumbnail(tuple(x / 2 for x in image.size))
    # image.resize((1024,1024),Image.BILINEAR)
    image.save(resized_path)

def delete_local_file(src):
    logger.info("delete files and folders")
    if os.path.isfile(src):
        try:
            os.remove(src)
        except:
            pass
    elif os.path.isdir(src):
        for item in os.listdir(src):
            itemsrc = os.path.join(src, item)
            delete_file_folder(itemsrc)
        try:
            os.rmdir(src)
        except:
            pass

def main_handler(event, context):
    logger.info("start main handler")
    for record in event['Records']:
        try:
            bucket = record['cos']['cosBucket']['name'] + '-' + str(appid)
            key = record['cos']['cosObject']['key']
            key = key.replace('/') + str(appid) + '/' + record['cos']['cosBucket']['name'] + '/', "", 1)
            download_path = '/tmp/{}'.format(key)
            tmpload_path = '/tmp/resized-{}'.format(key)
            print("Key is " + key)
            print("Get from [%s] to download file [%s]" % (bucket, key))

            # download image from cos
        try:
            response = client.get_object(Bucket=bucket, Key=key, )
            response['Body'].get_stream_to_file(download_path)
        except CosServiceError as e:
            print(e.get_error_code())
            print(e.get_error_msg())
            print(e.get_resource_location())
        return "Fail"
```

```
logger.info("Download file [%s] Success" % key)
logger.info("Image compress function start")
starttime = datetime.datetime.now()

# compress image here
resize_image(download_path, tmpload_path)
endtime = datetime.datetime.now()
logger.info("compress image take " + str((endtime - starttime).microseconds / 1000) + "ms")

# detect idcard
print("Start Detection")
client_card = Client(appid, secret_id, secret_key, record['cos']['cosBucket']['name'])
client_card.use_http()
client_card.set_timeout(30)
res_up = client_card.idcard_detect(CIFiles([tmpload_path]), 0)
res_down = client_card.idcard_detect(CIFiles([tmpload_path]), 1)
if res_up["result_list"][0]["code"] == 0 and res_down["result_list"][0]["code"] != 0:
    res_up_print = {
        "姓名": res_up["result_list"][0]["data"]["name"],
        "性别": res_up["result_list"][0]["data"]["sex"],
        "出生": res_up["result_list"][0]["data"]["birth"],
        "住址": res_up["result_list"][0]["data"]["address"],
        "民族": res_up["result_list"][0]["data"]["nation"],
        "公民身份证号": res_up["result_list"][0]["data"]["id"]
    }
    print(json.dumps(res_up_print).decode('unicode-escape'))
elif res_up["result_list"][0]["code"] != 0 and res_down["result_list"][0]["code"] == 0:
    res_down_print = {
        "有效期限": res_down["result_list"][0]["data"]["valid_date"],
        "签发机关": res_down["result_list"][0]["data"]["authority"]
    }
    print(json.dumps(res_down_print).decode('unicode-escape'))
else:
    print("err_message: [%s]" % (res_up["result_list"][0]["message"]))
    print("err_code: [%s]" % (res_up["result_list"][0]["code"]))
    print("err_filename: [%s]" % (res_up["result_list"][0]["filename"]))
    delete_local_file(str(download_path))
    delete_local_file(str(tmpload_path))
    return "Detect Fail"

# delete local file
delete_local_file(str(download_path))
delete_local_file(str(tmpload_path))
return "Success"
```

```
except Exception as e:
    print(e)
    raise e
return "Detect Fail"
```

注意：

在使用本段代码的时候，需要把 appid、secret_id 和 secret_key 替换为您自己的 APPID、SecretId 和 SecretKey 方能使用，您可以在 [云 API 密钥控制台](#) 中查看对应信息。

创建步骤

打包完成后，我们就可以前往 [无服务器云函数控制台](#) 进行部署。

1. 选择北京地域，单击【新建】，命名函数为 Demo2_IDCard_Detect，函数超时时间修改为 5 秒，内存默认 128MB 即可，单击【下一步】。
2. 选择 **本地上传zip包**，执行方法填写：idcard_detect.main_handler，保存后单击【下一步】。
3. 添加触发方式为 COS 触发，选择之前创建好的 Bucket：idcard-detect，事件类型为文件上传，单击【保存】>【完成】，完成函数部署。

您也可以直接从 [GitHub下载](#) 项目文件，并打成 zip 包，通过控制台创建函数并完成部署，请注意在函数代码中需修改 appid、secret_id 和 secret_key 并保存。

步骤 4. 测试函数功能

1. 进入对象存储控制台，选择创建好的 Bucket：idcard-detect，单击【上传文件】，选择自己拍好的身份证照片，照片要清晰可读且尽可能大的占满图片，然后上传。
2. 进入无服务器云函数控制台查看执行结果，在 **日志** 中可以看到打印出来的日志信息，包含身份证识别的结果。

使用 SCF 定时拨测并通过邮件发送告警

步骤 1. 创建邮件发送函数并测试

最近更新时间：2018-09-04 16:54:37

概述

在本 Demo 中，我们用到了无服务器云函数 SCF，消息队列 CMQ。其中，云函数 SCF2：PlayCheck 用来进行定时拨测，在拨测失败时写失败信息到 CMQ 主题：SendEmail，CMQ 会自动触发云函数 SCF1：SendEmail 来发送邮件。

发邮件功能和拨测功能可以在一个函数中实现，本示例想要演示消息队列 CMQ 的使用方法，同时使用 CMQ 可以解耦拨测功能和发邮件功能，便于维护。

创建 CMQ 主题

首先请登录 [消息服务CMQ 控制台](#)，新建一个主题。选择北京地域，命名为 SendEmail，主题属性保持默认即可。

创建云函数 SCF1：SendEmail

1. 前往 [无服务器云函数控制台](#)，选择北京地域，单击【新建】，命名函数为 SendEmail，函数超时时间修改为 5 秒，内存默认 128MB 即可，单击【下一步】。
2. 在函数代码的编辑框中，直接复制粘贴下面的代码。单击【下一步】。
3. 添加触发方式为 CMQ主题订阅触发，选择之前创建好的主题 Topic：SendEmail。单击【保存】>【完成】。

代码

```
# -*- coding: utf8 -*-  
import json  
import smtplib  
from email.mime.text import MIMEText  
from email.header import Header  
  
# 第三方 SMTP 服务  
mail_host="smtp.qq.com" #SMTP服务器  
mail_user="xxxxxxxxx" #用户名，如是QQ邮箱，则为****@qq.com  
mail_pass="xxxxxxxxx" #口令,登录smtp服务器的口令密码  
mail_port=465 #SMTP服务端口
```

```
def sendEmail(fromAddr,toAddr,subject,content):
    sender = fromAddr
    receivers = [toAddr] # 接收邮件，可设置为您的QQ邮箱或者其他邮箱

    message = MIMEText(content, 'plain', 'utf-8')
    message['From'] = Header(fromAddr, 'utf-8')
    message['To'] = Header(toAddr, 'utf-8')
    message['Subject'] = Header(subject, 'utf-8')

    try:
        smtpObj = smtplib.SMTP_SSL(mail_host, mail_port)
        smtpObj.login(mail_user,mail_pass)
        smtpObj.sendmail(sender, receivers, message.as_string())
        print("send email success")
    except smtplib.SMTPException as e:
        print(e)
        print("Error: send email fail")
    return False

def main_handler(event, context):
    if event is not None and "Records" in event.keys():
        if len(event["Records"]) >= 1 and "CMQ" in event["Records"][0].keys():
            cmqMsgStr = event["Records"][0]["CMQ"]["msgBody"]
            cmqMsg = json.loads(cmqMsgStr.replace("'", ""))
            print cmqMsg
            if sendEmail(cmqMsg['fromAddr'], cmqMsg['toAddr'], cmqMsg['title'], cmqMsg['body']):
                return "send email success"
            else:
                return "send email fail"
```

注意：

参数 mail_host、mail_user、mail_pass、mail_port 需要您根据期望发送的邮箱或邮件服务器来配置，本示例我们以 QQ 邮箱为例进行说配置：

您可以从 [QQ邮箱帮助中心](#) 了解到如何开启 QQ 邮箱的 SMTP 功能。QQ 邮箱的 SMTP 功能开启后，相应的参数如下。

- mail_host：SMTP 服务器地址，即 smtp.qq.com。
- mail_user：登录用户名为您的邮箱地址，例如 123*****@qq.com。
- mail_pass：您在开启 SMTP 功能时设置的密码。

- mail_port：服务器登录端口，由于 QQ 邮箱强制要求 SSL 登录，端口固定为 465，同时代码中使用 smtpplib.SMTP_SSL 创建 SSL 的 SMTP 连接。

测试 CMQ 主题和云函数的连通性

测试邮件发送的正确性

完成 SendEmail 函数创建后，可以在函数详情页的函数代码界面的右上角，单击【测试】，选择 CMQ Topic 事件模板，并把下面的代码复制粘贴到模板的代码框中，单击【测试运行】查看函数运行日志。

其中 msgBody 字段内的 fromAddr 字段和 toAddr 字段，需要根据您自身邮箱地址进行修改，建议修改为相同地址，自身邮箱向自身邮箱内发送邮件，以便测试邮件发送的正确性。

测试代码

```
{
  "Records": [
    {
      "CMQ": {
        "type": "topic",
        "topicOwner": "1253970226",
        "topicName": "sendEmailQueue",
        "subscriptionName": "sendEmailFunction",
        "publishTime": "2017-09-25T06:34:00.000Z",
        "msgId": "123345346",
        "requestId": "123345346",
        "msgBody": "{\"fromAddr\":\"*****@qq.com\",\"toAddr\":\"*****@qq.com\",\"title\":\"hello from scf & cmq\",\"body\":\"email content to send\"}",
        "msgTag": []
      }
    }
  ]
}
```

测试结果

测试成功结果如下图：

测试运行

测试结果：成功

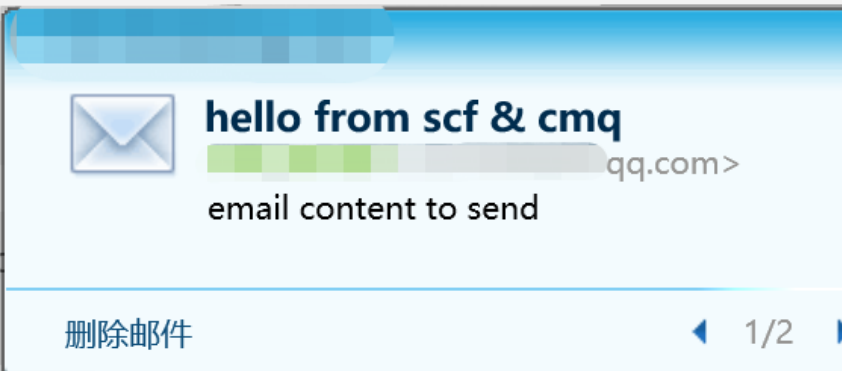
返回结果：
"send email success"

摘要

请求ID: adaa313f-9ab4-11e8-a842-5254005fc4da
运行时间: 1638.432ms
计费时间: 1700ms
占用内存: 0.242MB

日志

{u'body': u'email content to s
u'hello from scf & cmq'}
send email success



测试 CMQ 主题与云函数的联动

邮件测试成功后，继续测试 CMQ 主题和云函数的联动。

1. 请前往 [消息服务CMQ 控制台](#)，如下图所示，在主题订阅页面，单击主题 SendEmail 对应的【发送消息】按钮，在弹出的对话框中，输入如下消息内容后，单击【发送】

主题订阅 华北地区（北京）

+ 新建 API请求地址

请输入主题名称搜索

ID/主题名	监控	消息生命周期	消息最大长度	堆积消息数	创建/修改时间	操作
topic-9vho8vir SendEmail		24小时	64KB	0条	2018-08-07 21:28:40 2018-08-07 21:28:40	订阅 发送消息 置 删除 修改配

其中 msgBody 字段内的 fromAddr 字段和 toAdd 字段，需要根据您自身邮箱地址进行修改。

```
{  
  "fromAddr": "*****@qq.com",  
  "toAddr": "*****@qq.com",  
  "title": "hello from scf & cmq",  
}
```

```
"body": "email content to send"
}
```

2. 回到 [无服务器云函数控制台](#)，查看 SendEmail 函数的执行日志，并前往邮箱查看是否收到了邮件。

步骤 2. 创建拨测函数并测试

最近更新时间：2018-09-04 16:55:20

概述

通过 [步骤 1. 创建邮件发送函数并测试](#)，我们已经成功创建云函数 SCF1：SendEmail 和 CMQ 主题：SendEmail，并成功测试了连通性。

本文继续实现：创建云函数 SCF2：PlayCheck，用来定时拨测，在拨测失败时写失败信息到 CMQ 主题：SendEmail，进而触发云函数 SCF1：SendEmail 来发送邮件。

创建云函数 SCF2：PlayCheck

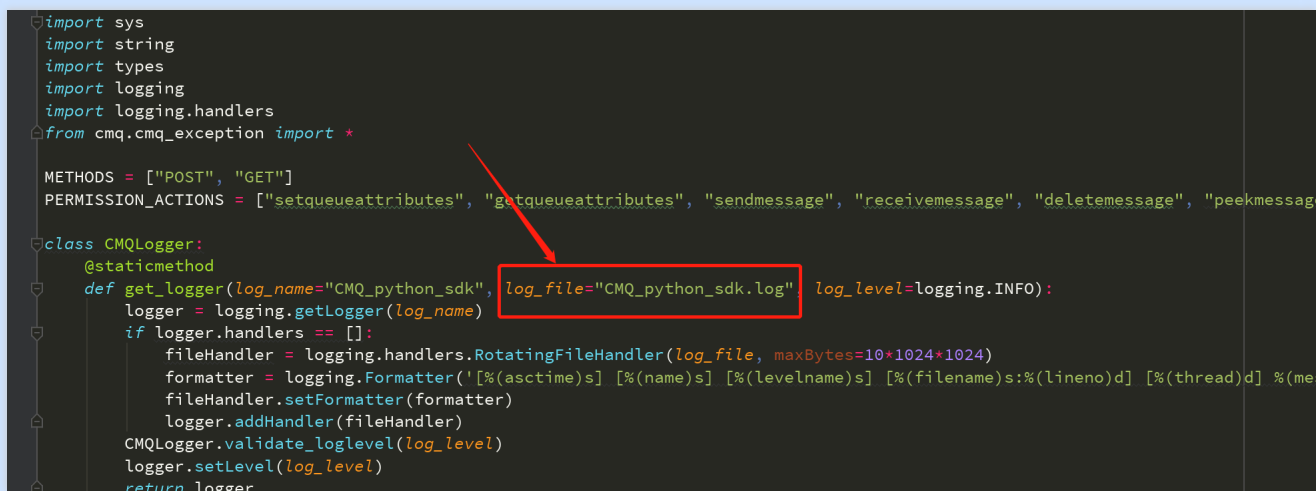
创建准备

1. 首先确保在您的系统中已经安装好了 Python 运行环境，然后在本地创建需要放置函数代码的文件夹。
2. [下载 CMQ Python SDK](#) 并解压，把 **文件夹 cmq** 复制到刚创建好的函数代码文件夹的根目录下。

注意：

因为本次 Demo 会在 SCF 中使用 CMQ SDK，而 CMQ SDK 发送消息时会本地创建 log，所以修改路径为云函数的本地临时存储路径 tmp。

请在 cmq 文件夹中找到 cmq_tool.py 文件，打开后，把 `log_file="CMQ_python_sdk.log"`（如下图所示位置）修改为 `log_file="/tmp/CMQ_python_sdk.log"` 并保存。



```
import sys
import string
import types
import logging
import logging.handlers
from cmq.cmq_exception import *

METHODS = ["POST", "GET"]
PERMISSION_ACTIONS = ["setqueueattributes", "getqueueattributes", "sendmessage", "receivemessage", "deletemessage", "peekmessage"]

class CMQLogger:
    @staticmethod
    def get_logger(log_name="CMQ_python_sdk", log_file="CMQ_python_sdk.log", log_level=logging.INFO):
        logger = logging.getLogger(log_name)
        if logger.handlers == []:
            fileHandler = logging.handlers.RotatingFileHandler(log_file, maxBytes=10*1024*1024)
            formatter = logging.Formatter('%(asctime)s [%(name)s] [%(levelname)s] [%(filename)s:%(lineno)d] [%(thread)d] %(message)s')
            fileHandler.setFormatter(formatter)
            logger.addHandler(fileHandler)
            CMQLogger.validate_loglevel(log_level)
            logger.setLevel(log_level)
        return logger
```

3. 前往函数代码文件夹的根目录下，使用下面的示例代码创建 Python 文件，命名为：Play_Check.py：
4. 保存后，回到根目录下，对 cmq 文件夹和 Play_Check.py 文件进行压缩，打包成 zip 格式，注意不是对外层的文件夹打包。

示例代码

```
# -*- coding: utf8 -*-
import sys
import os

sys.path.insert(0, os.path.dirname(os.path.abspath(__file__)) + "/..")

import json
import logging
from cmq.account import Account
from cmq.cmq_exception import *
from cmq.topic import *
import requests

logger = logging.getLogger()
logger.setLevel(logging.DEBUG)

#使用 cmq 所需的鉴权信息
secret_id = u'*****' # 请替换为您的 SecretId
secret_key = u'*****' # 请替换为您的 SecretKey
topic_name = "SendEmail" # 请替换为您的 Topic 名称
endpoint = "http://cmq-topic-bj.api.qcloud.com"

#拨测失败后，告警邮件需要通知的邮箱列表
email_notify_list = {
    "*****@qq.com",
    "*****@qq.com",
}

#拨测失败后，发出告警邮件的邮箱，请根据您的自身设置的邮箱地址进行修改
fromAddr = "*****@qq.com"

#拨测地址列表
test_url_list = [
    "http://www.baidu.com",
    "http://www.qq.com",
    "http://wrong.tencent.com",
    "http://unkownurl.com"
```

```
]

def Send_CMQ(body):
    # 初始化 my_account
    my_account = Account(endpoint, secret_id, secret_key, debug=True)
    my_account.set_log_level(logging.DEBUG)
    my_topic = my_account.get_topic(topic_name)
    for toAddr in email_notify_list:
        try:
            message = Message()
            sendbody = {
                "fromAddr": fromAddr,
                "toAddr": toAddr,
                "title": u"Please note: PlayCheck Error 拨测地址异常，请检查",
                "body": body
            }
            message.msgBody = json.dumps(sendbody)
            print ("send message [%s] to [%s]" % (body, toAddr))
            my_topic.publish_message(message)

        except CMQExceptionBase, e:
            print "Exception:%s\n" % e

def test_url(url_list):
    errorinfo = []
    for url in url_list:
        resp = None
        try:
            resp = requests.get(url, timeout=3)
            print resp
        except (
            requests.exceptions.Timeout, requests.exceptions.ConnectionError, requests.exceptions.ConnectTimeo
            ut) as e:
            logger.warn("request exceptions:" + str(e))
            errorinfo.append("Access " + url + " timeout")
        else:
            if resp.status_code >= 400:
                logger.warn("response status code fail:" + str(resp.status_code))
                errorinfo.append("Access " + url + " fail, status code:" + str(resp.status_code))
            if len(errorinfo) != 0:
                Send_CMQ("\r\n".join(errorinfo))

def main_handler(event, context):
```

```
test_url(test_url_list)
```

```
if __name__ == '__main__':  
    main_handler("", "")
```

注意：

- 在使用本段代码的时候，需要把 appid、secret_id 和 secret_key 替换为您自己的 APPID、SecretId 和 SecretKey 方能使用，您可以在 [云 API 密钥控制台](#) 中查看对应信息。
- topic_name 需要和您在 [步骤 1. 创建邮件发送函数并测试](#) 中创建的 topic 保持一致。
- 请把 email_notify_list 中的地址替换为您想要通知的邮箱列表，可放置一个，也可放置多个。
- fromAddr 是发出告警邮件的邮箱，还请根据您的自身设置的邮箱地址进行修改。

创建步骤

打包完成后，我们就可以前往 [无服务器云函数控制台](#) 进行部署。

1. 选择北京地域，单击【新建】，命名函数为 PlayCheck，函数超时时间修改为 5 秒，内存默认 128MB 即可，单击【下一步】。
2. 选择 **本地上传zip包**，执行方法填写：Play_Check.main_handler，保存后单击【下一步】。
3. 触发方式暂不设置，单击【完成】，完成函数部署。

您也可以直接从 [GitHub 下载](#) 项目文件，并打成 zip 包，通过控制台创建函数并完成部署，请注意：

1. 将Play_Check 文件夹中的 cmq 文件夹和 Play_Check.py 文件打包为 zip 文件。
2. 在函数代码中需修改 secret_id、secret_key、topic_name、email_notify_list、fromAddr 等字段并保存。

测试函数功能

PlayCheck 函数部署完成后，可以在函数详情页的函数代码界面的右上角，单击【测试】，默认使用“Hello World 事件模板”即可，单击【测试运行】查看运行日志，函数正常执行成功之后，可以收到主题为“Please note: PlayCheck Error 拨测地址异常，请检查”的告警邮件，则表示函数功能正常。

函数测试成功后，您可以给 PlayCheck 函数添加定时触发器，那么一个完整的播测+邮件告警系统就部署完成了。您也可以根据自身业务的需求，DIY 其他的拨测和告警使用方法。

使用 SCF 实现日志分析写数据库

最近更新时间：2018-09-03 20:05:30

概述

在本 Demo 中，我们用到了无服务器云函数 SCF，对象存储 COS，云数据库 MySQL。其中，对象存储 COS 用来存储需要分析的日志文件，无服务器云函数 SCF 实现从 COS 下载日志文件并进行统计分析，把分析的结果写入到 MySQL 数据库中。

步骤 1. 创建 COS Bucket

登录 [对象存储控制台](#) 创建一个 Bucket，我们命名为 loganalysis，并选择北京地域，权限选择“公有读-私有写”。

步骤 2. 创建 MySQL 云数据库

由于数据库需要花钱购买，您可以选择在北京地域购买最便宜的 [云数据库MySQL入门机型](#)，价格为 12 元/月。

步骤 3. 创建云函数 SCF

创建准备

1. 首先确保在您的系统中已经安装好了 Python 运行环境和 pip 工具，然后在本地创建需要放置函数代码的文件夹，并通过命令行进入该目录下，直接执行命令安装数据库操作相关的库：

```
pip install cos-python-sdk-v5 -t .  
pip install pymysql -t .  
pip install cryptography -t .
```

2. 在函数代码文件夹的根目录下，使用下面的示例代码创建 Python 文件，可以命名为：LogAnalysis.py。
3. Python 文件保存后，回到根目录下，对所有文件进行打包（**请注意不是对外层的文件夹打包**）；另外还需要保证：**LogAnalysis.py 存在于根目录下，压缩包需要为 zip 格式。**

示例代码

```
# -*- coding: utf-8 -*-  
from qcloud_cos import CosConfig  
from qcloud_cos import CosS3Client
```

```
from qcloud_cos import CosServiceError

from datetime import datetime
from time import time
import re
import operator
import pymysql.cursors
import logging

# MySql数据库
Host='*****' #如bj-cdb-cj9w3q53.sql.tencentcdb.com
User='xxxxx'
Password='xxxxx'
Port= 63054
DB=u'*****' # 如SCF_Demo

# COS
appid = ***** # 请替换为您的 APPID
secret_id = u'*****' # 请替换为您的 SecretId
secret_key = u'*****' # 请替换为您的 SecretKey
region = u'ap-beijing'
token = ''

config = CosConfig(Secret_id=secret_id, Secret_key=secret_key, Region=region, Token=token)
client = CosS3Client(config)
logger = logging.getLogger()

def main_handler(event, context):
    # Authentication Information
    print("Start Request {}", datetime.fromtimestamp(time()).strftime('%Y-%m-%d %H:%M:%S'))

    # Start downloading from COS
    for record in event['Records']:
        try:
            bucket = record['cos']['cosBucket']['name'] + '-' + str(appid)
            key = record['cos']['cosObject']['key']
            key = key.replace('/') + str(appid) + '/' + record['cos']['cosBucket']['name'] + '/', '', 1)
            download_path = '/tmp/{}'.format(key)
            print("Key is " + key)
            print("Get from [%s] to download file [%s]" % (bucket, key))
            try:
                response = client.get_object(Bucket=bucket, Key=key, )
                response['Body'].get_stream_to_file(download_path)
            except CosServiceError as e:
```

```
print(e.get_error_code())
print(e.get_error_msg())
print(e.get_resource_location())
return "Download log fail"
logger.info("Download file [%s] Success" % key)
except Exception as e:
    print(e)
raise e
return "Access COS Fail"

print("Start analyzing data {}", datetime.fromtimestamp(time()).strftime('%Y-%m-%d %H:%M:%S'))
urlList = {}
statueList = {}
terminalList = {}
timeList = {"24/May/2018 10:00-10:30": 0, "24/May/2018 10:30-11:00": 0}

fileObject = open(download_path, 'rU')
try:
    for line in fileObject:
        # Count URL
        URLstart = re.search("GET", line)
        URLend = re.search("mp4", line)
        if URLstart and URLend:
            url = line[URLstart.end() + 1: URLend.end()]
            if url in urlList:
                urlList[url] += 1
            else:
                urlList[url] = 1

        # Count Statue code
        Statuestart = re.search("HTTP/1.1", line)
        if Statuestart:
            StatueCode = line[Statuestart.end() + 2: Statuestart.end() + 5]
            if StatueCode in statueList:
                statueList[StatueCode] += 1
            else:
                statueList[StatueCode] = 1

        # Count Terminal Device
        Terminalstart = re.search("\-\\", line)
        TerminalEnd = re.search("\-\\ \"-\\", line)
        if Terminalstart and TerminalEnd:
            terminal = line[Terminalstart.end() + 2: TerminalEnd.start() - 2]
            if terminal in terminalList:
                terminalList[terminal] += 1
```

```
else:
    terminalList[terminal] = 1

# Count Timelist
Timestarter = re.search("\[", line)
if Timestarter:
    if int(line[Timestarter.end() + 15: Timestarter.end() + 17]) > 30:
        timeList["24/May/2018 10:30-11:00"] += 1
    else:
        timeList["24/May/2018 10:00-10:30"] += 1

finally:
    fileObject.close()

# Sort Result according to frequency
URL_sorted_res = sorted(urlList.items(), key=operator.itemgetter(1), reverse=True)
Statue_sorted_res = sorted(statueList.items(), key=operator.itemgetter(1), reverse=True)
Terminal_sorted_res = sorted(terminalList.items(), key=operator.itemgetter(1), reverse=True)
Time_sorted_res = sorted(timeList.items(), key=operator.itemgetter(1), reverse=True)

URLres = []
Statueres = []
Terminalres = []
Timeres = []

for i in range(3):
    URLres.append(URL_sorted_res[i])
    Statueres.append(Statue_sorted_res[i])
    Terminalres.append(Terminal_sorted_res[i])

for i in range(2):
    Timeres.append(Time_sorted_res[i])

print("Analyzing Successfully, Start writing to database {}",
      datetime.fromtimestamp(time()).strftime('%Y-%m-%d %H:%M:%S'))

connection = pymysql.connect(host=Host,
                              user=User,
                              password=Password,
                              port=Port,
                              db= DB,
                              charset='utf8',
                              cursorclass=pymysql.cursors.DictCursor)

try:
```

```
with connection.cursor() as cursor:
    # Clean dirty data
    cursor.execute("DROP TABLE IF EXISTS url")
    cursor.execute("DROP TABLE IF EXISTS state")
    cursor.execute("DROP TABLE IF EXISTS terminal")
    cursor.execute("DROP TABLE IF EXISTS time")
    cursor.execute("CREATE TABLE url (URL TEXT NOT NULL, Count INT)")
    cursor.execute("CREATE TABLE state (StateCode TEXT NOT NULL, Count INT)")
    cursor.execute("CREATE TABLE terminal (Terminal TEXT NOT NULL, Count INT)")
    cursor.execute("CREATE TABLE time (Timestatue TEXT NOT NULL, Count INT)")

    sql = "INSERT INTO `url` (`URL`, `Count`) VALUES (%s, %s)"
    for i in range(len(URLres)):
        cursor.execute(sql, (URLres[i][0], URLres[i][1]))

    sql = "INSERT INTO `state` (`StateCode`, `Count`) VALUES (%s, %s)"
    for i in range(len(Statueres)):
        cursor.execute(sql, (Statueres[i][0], Statueres[i][1]))

    sql = "INSERT INTO `terminal` (`Terminal`, `Count`) VALUES (%s, %s)"
    for i in range(len(Terminalres)):
        cursor.execute(sql, (Terminalres[i][0], Terminalres[i][1]))

    sql = "INSERT INTO `time` (`Timestatue`, `Count`) VALUES (%s, %s)"
    for i in range(len(Timeres)):
        cursor.execute(sql, (Timeres[i][0], Timeres[i][1]))

    connection.commit()

    finally:
        connection.close()

    print("Write to database successfully {}", datetime.fromtimestamp(time()).strftime('%Y-%m-%d %H:%M:%S'))
    return "LogAnalysis Success"
```

注意：

在使用本段代码的时候，需要把 appid、secret_id 和 secret_key 替换为您自己的 APPID、SecretId 和 SecretKey 方能使用，您可以在 [云 API 密钥控制台](#) 中查看对应信息。

创建步骤

打包完成后，我们就可以前往 [无服务器云函数控制台](#) 进行部署。

1. 选择北京地域，单击【新建】，命名函数为 Demo3_LogAnalysis，函数超时时间修改为 10 秒，内存默认 128MB 即可，单击【下一步】。
2. 选择 **本地上传 zip 包**，执行方法填写：LogAnalysis.main_handler，保存后单击【下一步】。
3. 添加触发方式为 COS 触发，选择之前创建好的 Bucket：loganalysis，事件类型为文件上传，单击【保存】>【完成】，完成函数部署。

在这里，您也可以直接从 [GitHub 下载](#) 项目文件，并打成 zip 包，通过控制台创建函数并完成部署，请注意：

1. 在打包 zip 文件时，请不要包含“demo-scf1.txt”和“demo-scf2.txt”，否则 zip 包有可能会超过 5MB；
2. 在函数代码中需修改 appid、secret_id 和 secret_key 并保存。

步骤 4. 测试函数功能

1. 进入对象存储控制台，选择创建好的 Bucket：loganalysis，单击【上传文件】，选择 [GitHub](#) 中提供的样例日志文件 demo-scf1.txt，然后上传。
2. 进入无服务器云函数控制台查看执行结果，在 **日志** 中可以看到打印出来的日志信息。
3. 前往 MySQL 管理界面，查看分析结果。
4. 依照前面的三个步骤上传 demo-scf2.txt，然后查看 SCF 的执行结果和 MySQL 中的分析结果。

这里可以根据您自身的日志格式编写具体的处理方法，数据库的写方法也可以改成增量写。