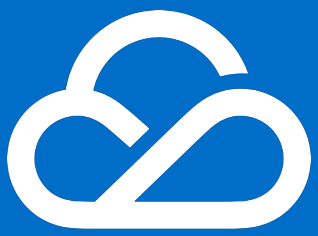
腾讯云无服务器云函数

代码实操

产品文档



腾讯云

【版权声明】

【商标声明】

【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

代码实操

以WordCount为示例的MapReduce方法

示例说明

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。

步骤一：准备COS Bucket

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) 点击【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

，点击【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。）

步骤二：创建部署程序包

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'lLkxx40kIfuyqW0IOI0WqyueCYjlgZEE' # 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

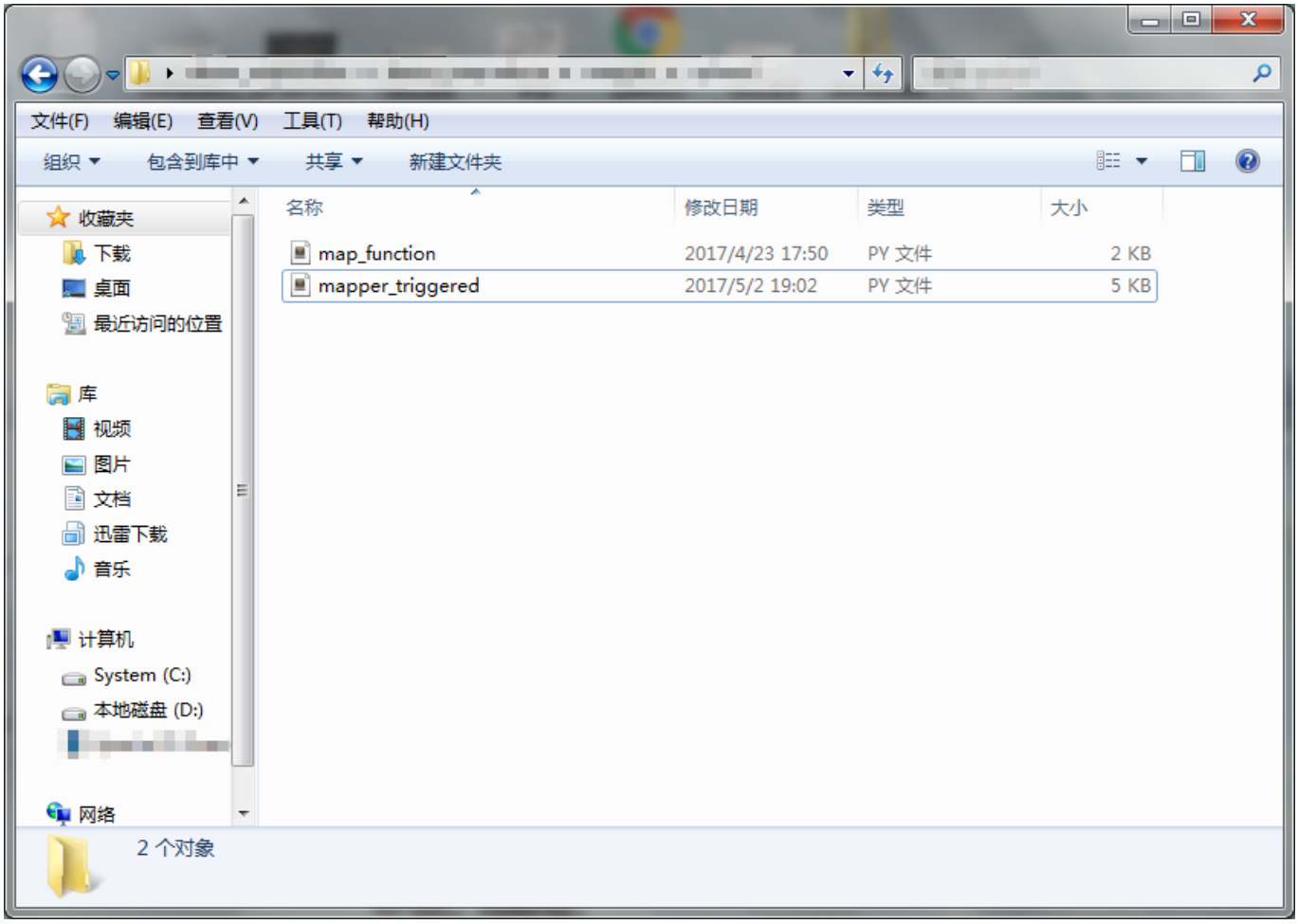
文件，写入下面的代码并保存：

```
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

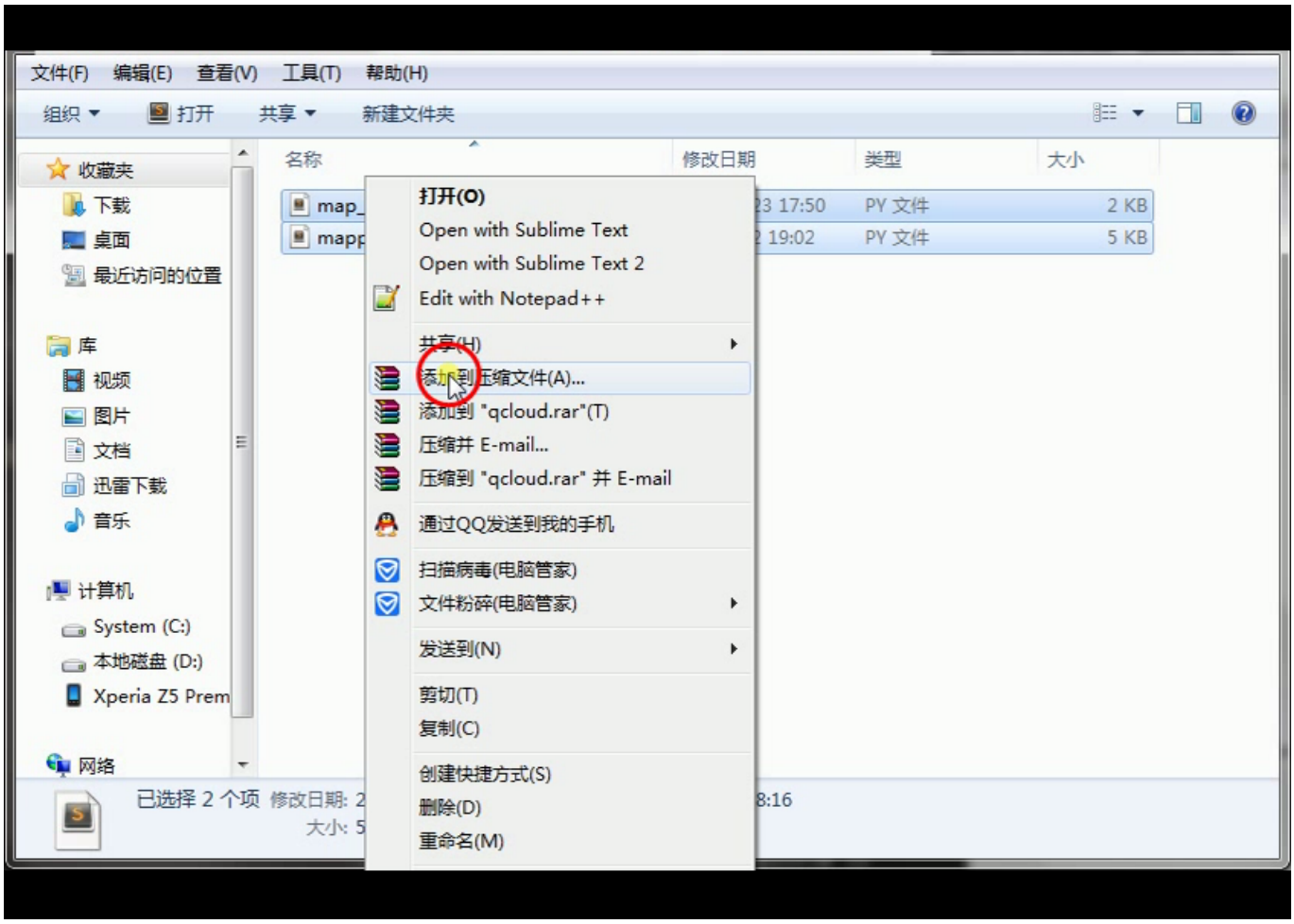
mapper（特别注意：是将文件直接压缩，而不是压缩文件所在的文件夹）。

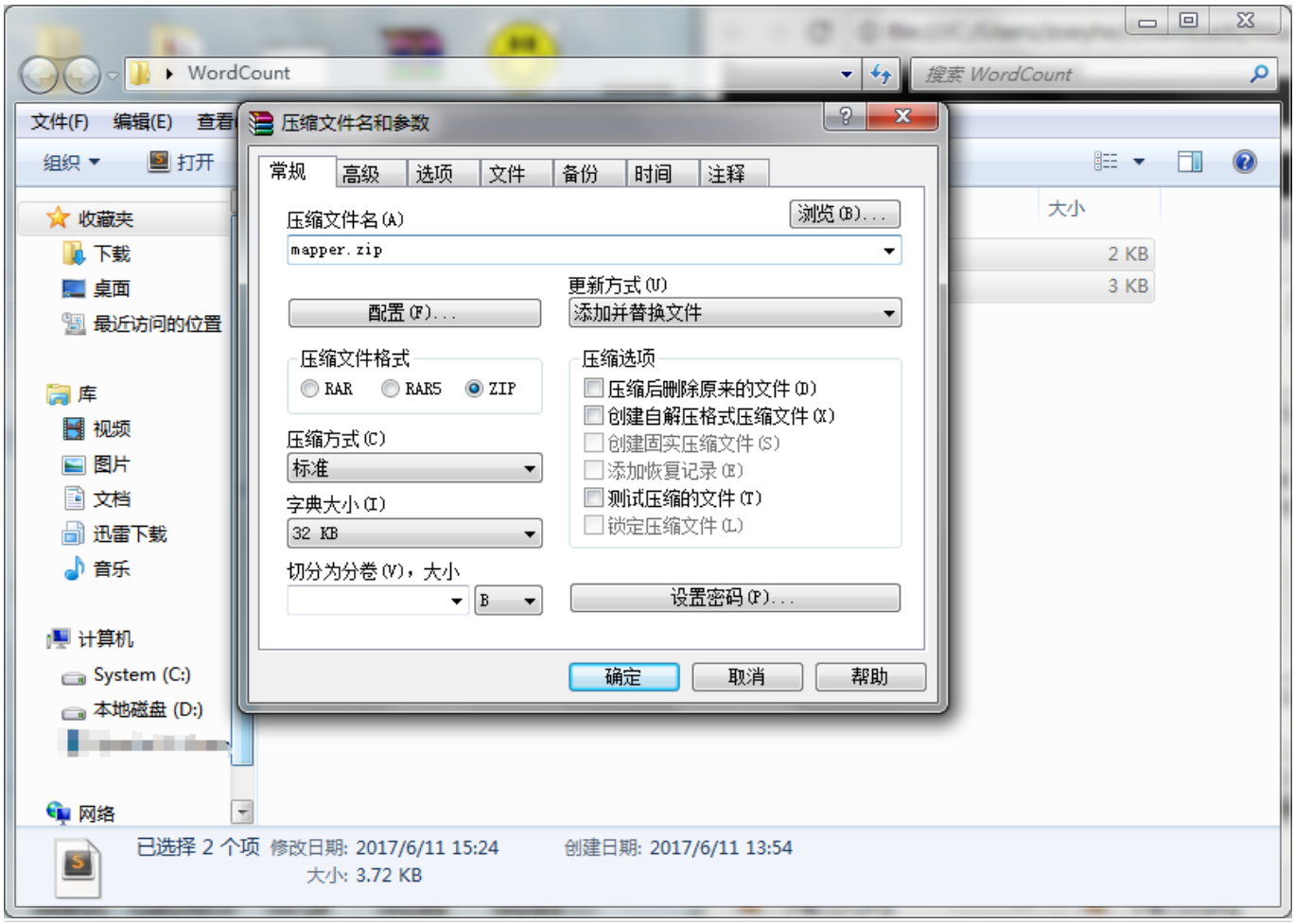
Windows 环境下：

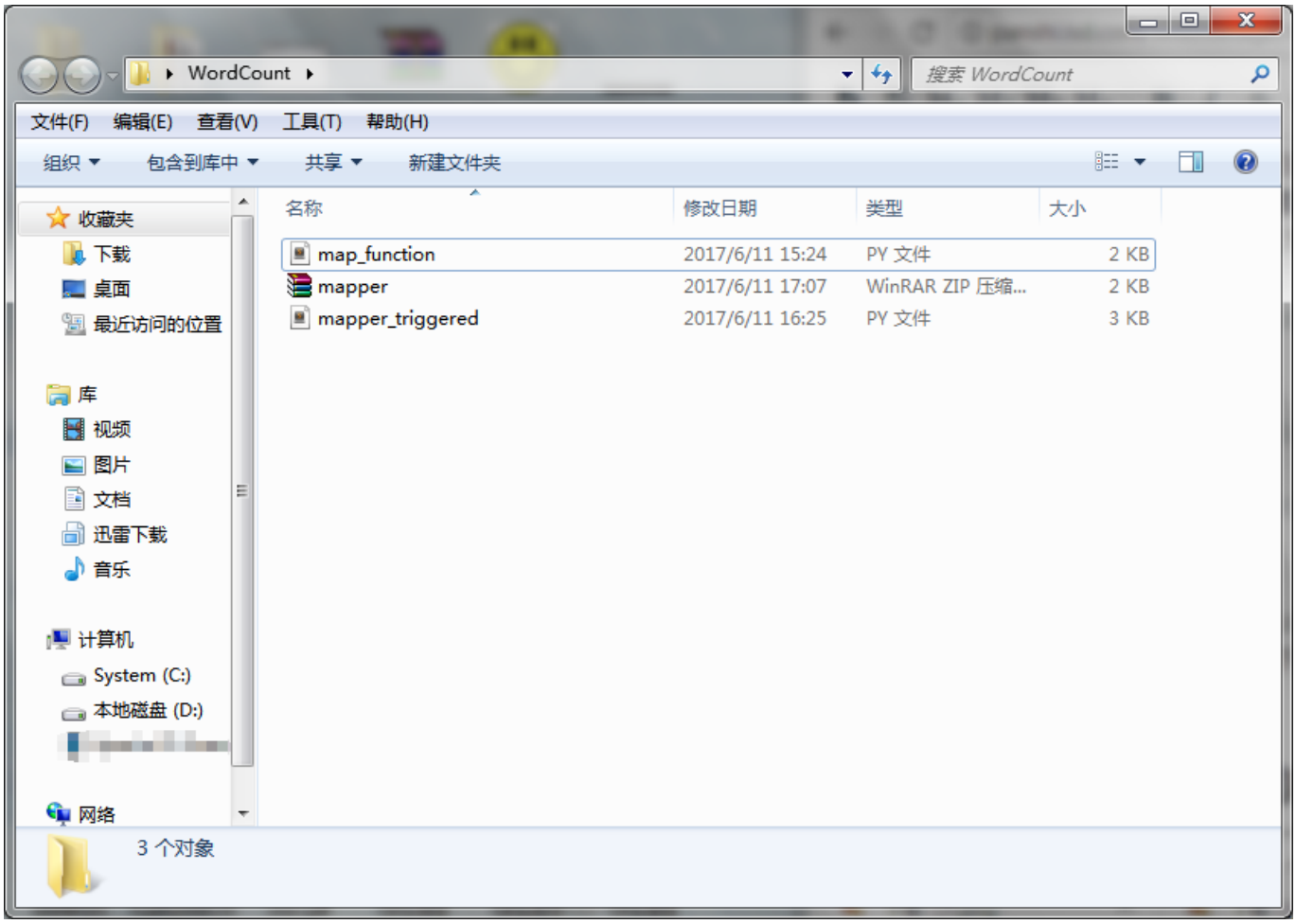
选中这两个文件，点击右键，选择您的压缩工具如winrar，点击【添加到压缩文件...】，将压缩文件设置为

zip

格式，点击【确定】按钮，将生成一个zip包。


文件(F) 编辑(E) 查看(V) 工具(T) 帮助(H)
组织 打开 共享 新建文件夹
收藏夹
下载
桌面
最近访问的位置
库
视频
图片
文档
迅雷下载
音乐
计算机
System (C:)
本地磁盘 (D:)
Xperia Z5 Prem
网络
名称
修改日期
类型
大小
map_
mapp
23 17:50
19:02
PY 文件
2 KB
5 KB
打开(O)
Open with Sublime Text
Open with Sublime Text 2
Edit with Notepad++
共享(H)
添加到压缩文件(A)...
添加到 "qcloud.rar"(T)
压缩并 E-mail...
压缩到 "qcloud.rar" 并 E-mail
通过QQ发送到我的手机
扫描病毒(电脑管家)
文件粉碎(电脑管家)
发送到(N)
剪切(T)
复制(C)
创建快捷方式(S)
删除(D)
重命名(M)
已选择 2 个项
修改日期: 2
大小: 5
8:16


WordCount
搜索 WordCount
文件(F) 编辑(E) 查看
组织 打开
压缩文件名和参数
常规 高级 选项 文件 备份 时间 注释
压缩文件名(A)
浏览(B)...
mapper.zip
更新方式(U)
配置(F)...
添加并替换文件
压缩文件格式
RAR RAR5 ZIP
压缩选项
压缩后删除原来的文件(D)
创建自解压格式压缩文件(X)
创建固实压缩文件(S)
添加恢复记录(E)
测试压缩的文件(T)
锁定压缩文件(L)
压缩方式(C)
标准
字典大小(I)
32 KB
切分为分卷(V)，大小
B
设置密码(P)...
确定
取消
帮助
收藏夹
下载
桌面
最近访问的位置
库
视频
图片
文档
迅雷下载
音乐
计算机
System (C:)
本地磁盘 (D:)
网络
大小
2 KB
3 KB
已选择 2 个项
修改日期: 2017/6/11 15:24
创建日期: 2017/6/11 13:54
大小: 3.72 KB



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'lLkxx40kIfuyqW0IOI0WqyueCYjlgZEE' # 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）。

步骤三：创建 Mapper 和 Reducer 函数并测试

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】地域下点击【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，点击【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

，点击【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，点击【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】地域下点击【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，点击【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

，点击【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，点击【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，点击【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":"appId",
"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) 点击【运行】按钮，观察运行结果。

5) 前往对象存储控制台，点击步骤一：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

步骤四：添加触发器

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，点击【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”，点击【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。

获取COS上的图像并创建缩略图

示例说明

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 exampleresized 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。

步骤一：准备COS Bucket

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) 点击【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

，点击【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。）

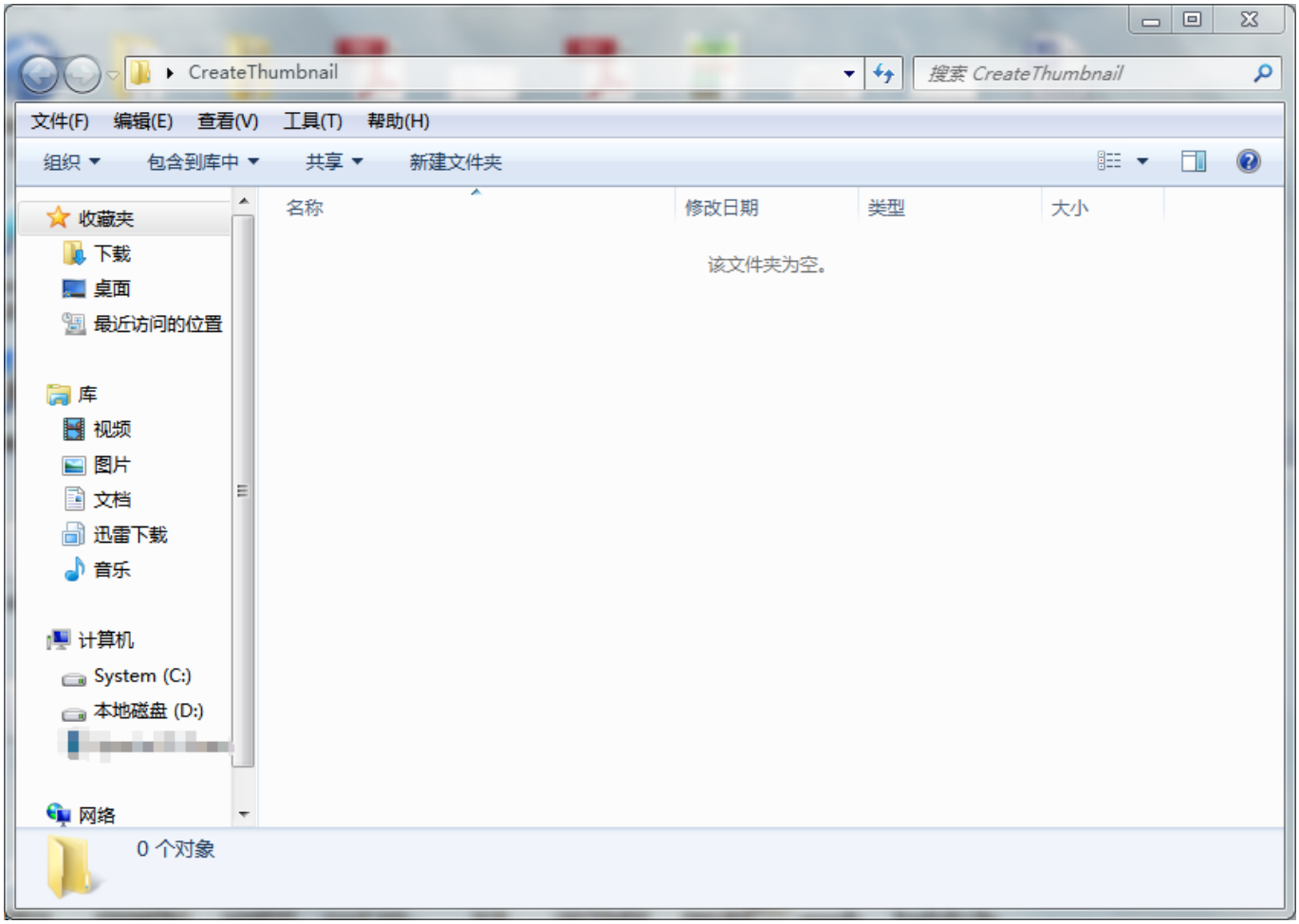
步骤二：创建部署程序包

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@[redacted]_centos /]# 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'lLkxx40kIfuyqW0IOI0WqyueCYjlgZQ2' #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)
```

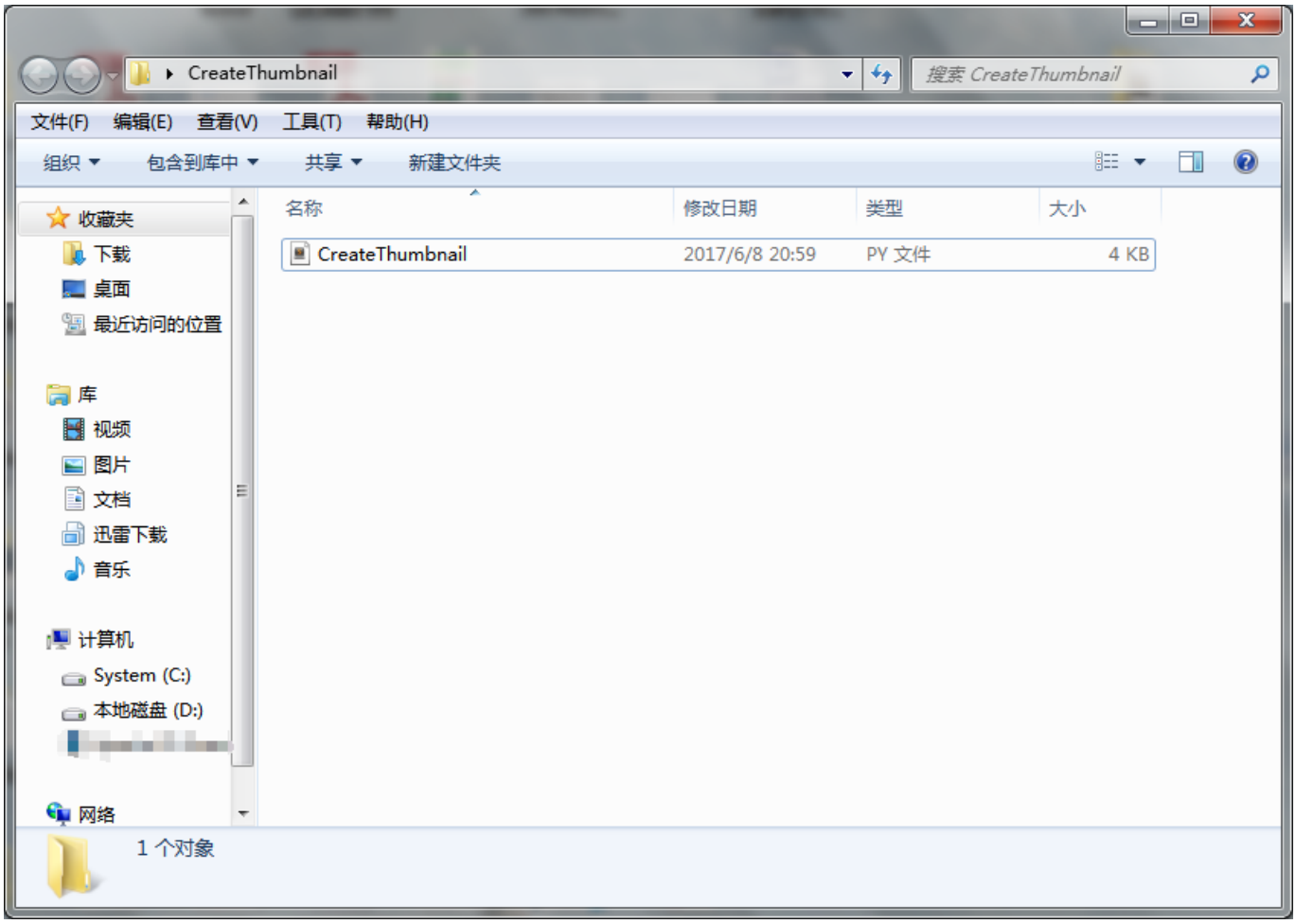
```
  except:
  pass
  elif os.path.isdir(src):
  for item in os.listdir(src):
  itemsrc=os.path.join(src,item)
  delete_file_folder(itemsrc)
  try:
  os.rmdir(src)
  except:
  pass

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@VM_67_49_centos /]# cd /CreateThumbnail
[root@VM_67_49_centos CreateThumbnail]# vi CreateThumbnail.py
```

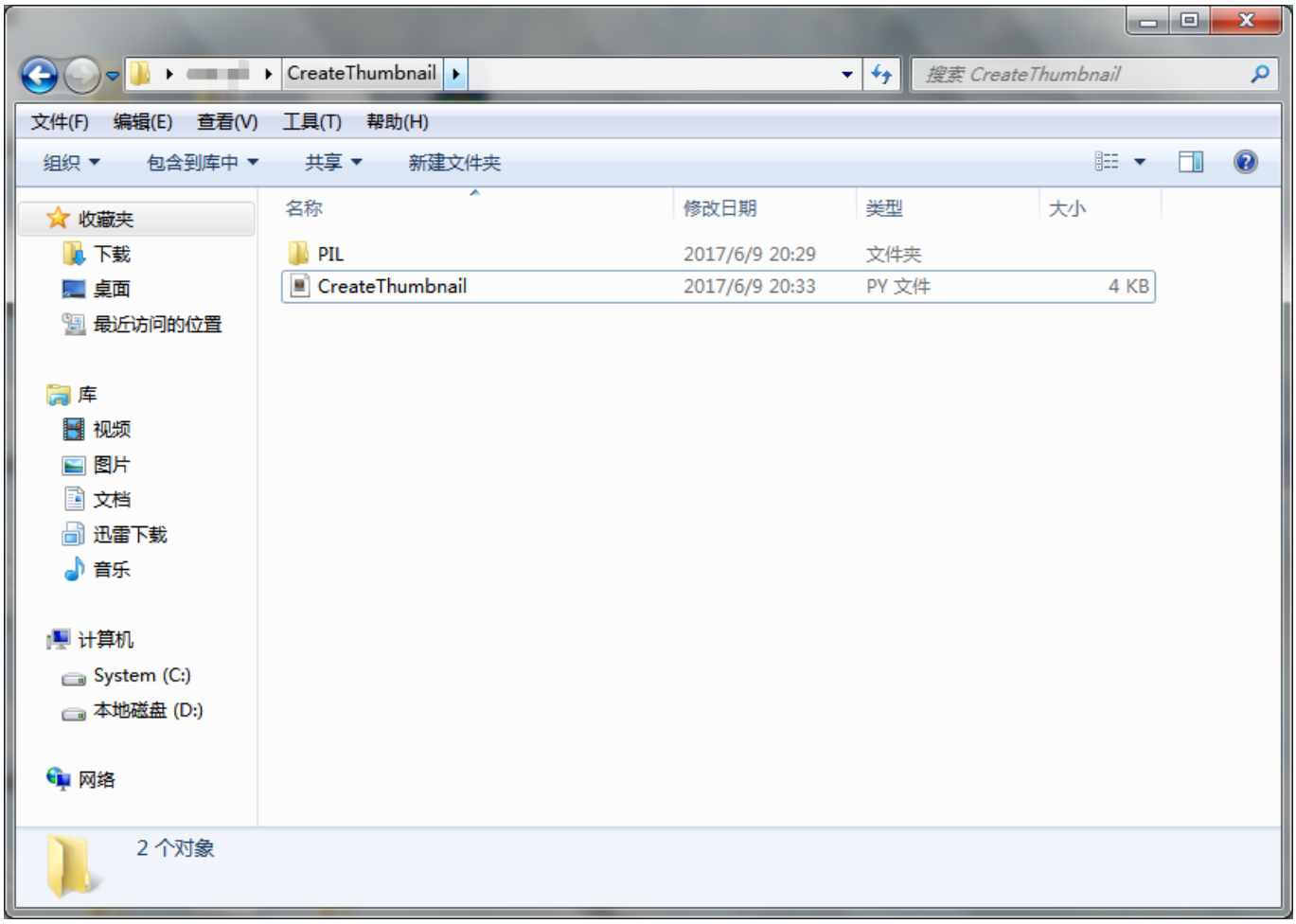
```
[root@VM_67_49_centos CreateThumbnail]# ls
CreateThumbnail.py
```

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:

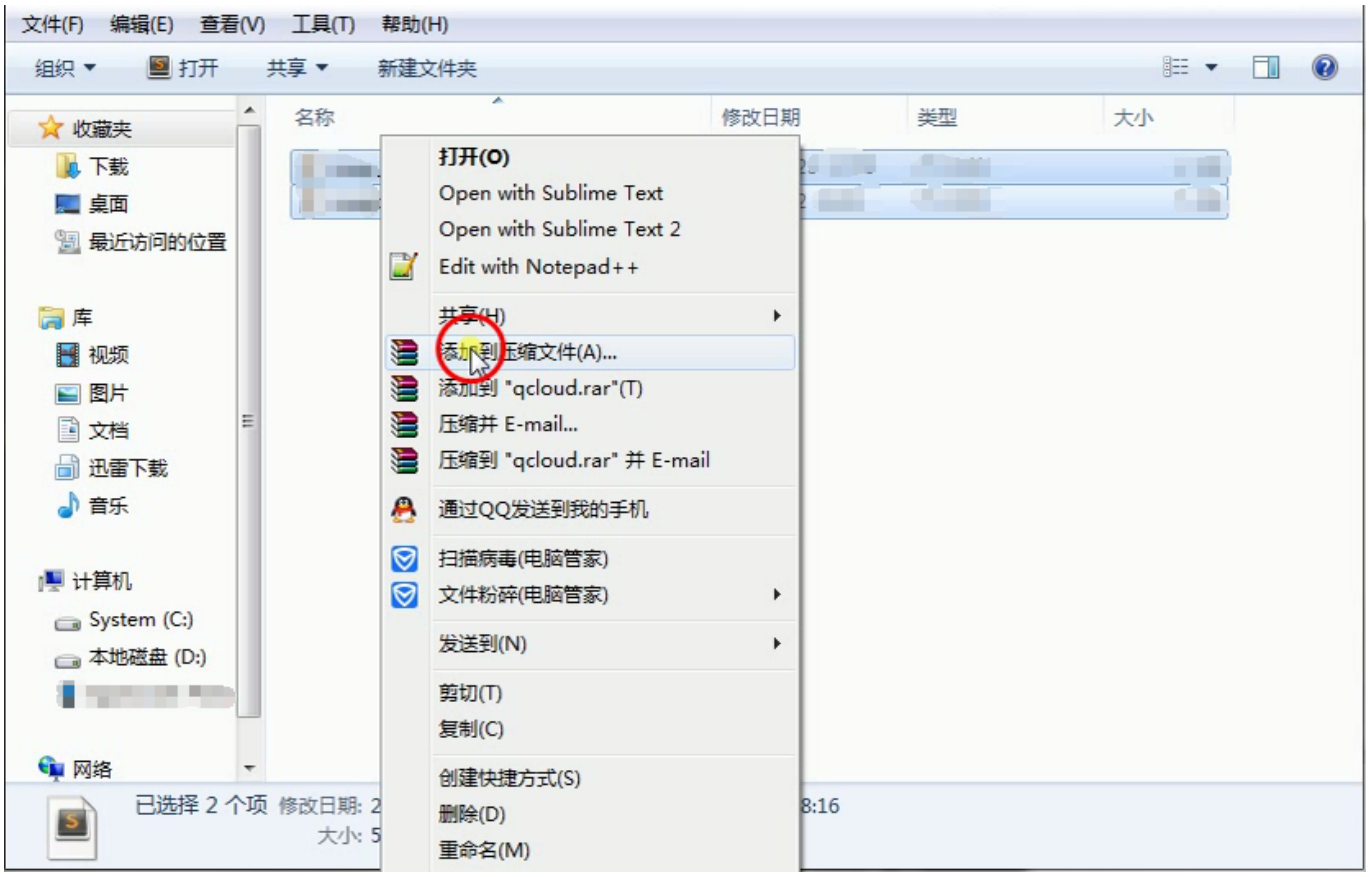
直接点击链接下载 Pillow 库，并将该zip包解压至刚刚创建的CreateThumbnail 文件夹内：

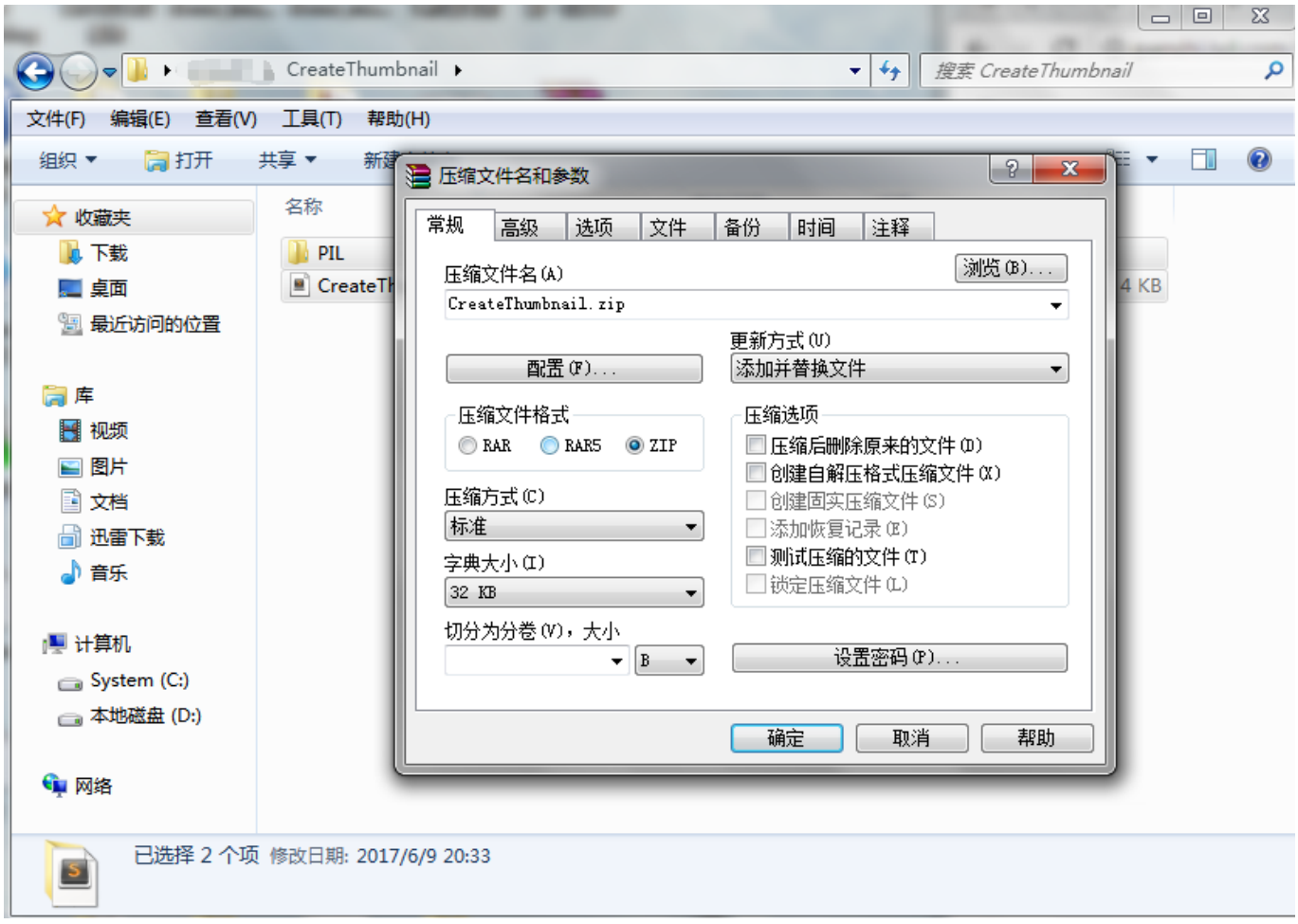


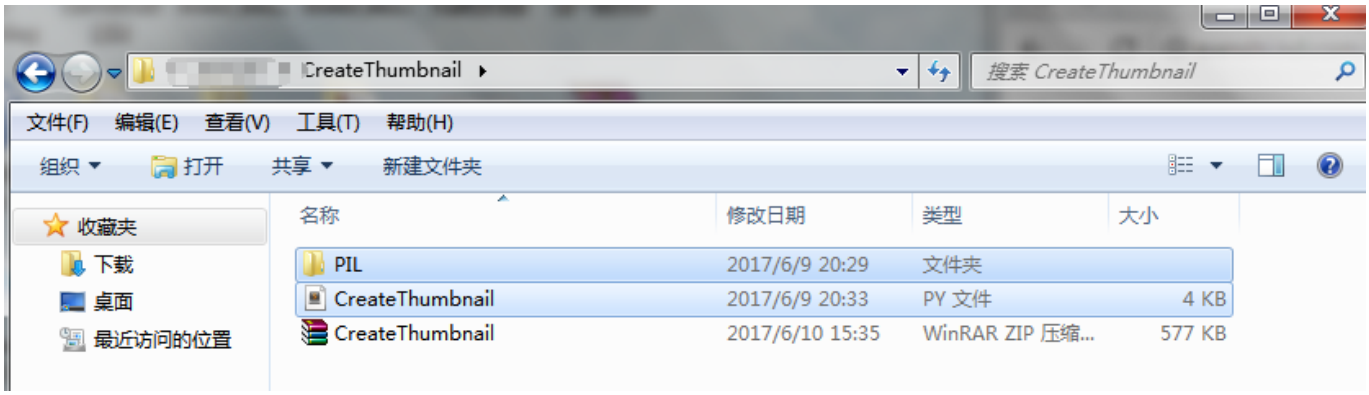
压缩该文件夹下的所有内容至一个名为 CreateThumbnailDemo.zip 的压缩包中（注意不是压缩文件夹本身！）：选中所有文件，点击右键，选择您的压缩工具如winrar，点击【添加到压缩文件...】，将压缩文件设置为zip格式，点击【确定】按钮，将生成一个zip包（此处命名为

CreateThumbnailDemo.zip

）。


文件(F) 编辑(E) 查看(V) 工具(T) 帮助(H)
组织 打开 共享 新建文件夹
收藏夹
下载
桌面
最近访问的位置
库
视频
图片
文档
迅雷下载
音乐
计算机
System (C:)
本地磁盘 (D:)
网络
名称
修改日期
类型
大小
打开(O)
Open with Sublime Text
Open with Sublime Text 2
Edit with Notepad++
共享(H)
添加到压缩文件(A)...
添加到 "qcloud.rar"(T)
压缩并 E-mail...
压缩到 "qcloud.rar" 并 E-mail
通过QQ发送到我的手机
扫描病毒(电脑管家)
文件粉碎(电脑管家)
发送到(N)
剪切(T)
复制(C)
创建快捷方式(S)
删除(D)
重命名(M)
已选择 2 个项 修改日期: 2
大小: 5
8:16


CreateThumbnail
搜索 CreateThumbnail
文件(F) 编辑(E) 查看(V) 工具(T) 帮助(H)
组织 打开 共享
压缩文件名和参数
常规 高级 选项 文件 备份 时间 注释
压缩文件名(A)
CreateThumbnail.zip
浏览(B)...
配置(F)...
更新方式(U)
添加并替换文件
压缩文件格式
RAR RAR5 ZIP
压缩选项
压缩后删除原来的文件(D)
创建自解压格式压缩文件(X)
创建固实压缩文件(S)
添加恢复记录(E)
测试压缩的文件(T)
锁定压缩文件(L)
压缩方式(C)
标准
字典大小(I)
32 KB
切分为分卷(V)，大小
B
设置密码(P)...
确定
取消
帮助
PIL
收藏夹
下载
桌面
最近访问的位置
库
视频
图片
文档
迅雷下载
音乐
计算机
System (C:)
本地磁盘 (D:)
网络
已选择 2 个项 修改日期: 2017/6/9 20:33



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
```

步骤三：创建 CreateThumbnailDemo 函数并测试

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】地域下点击【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，点击【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

，点击【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，点击【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，点击【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

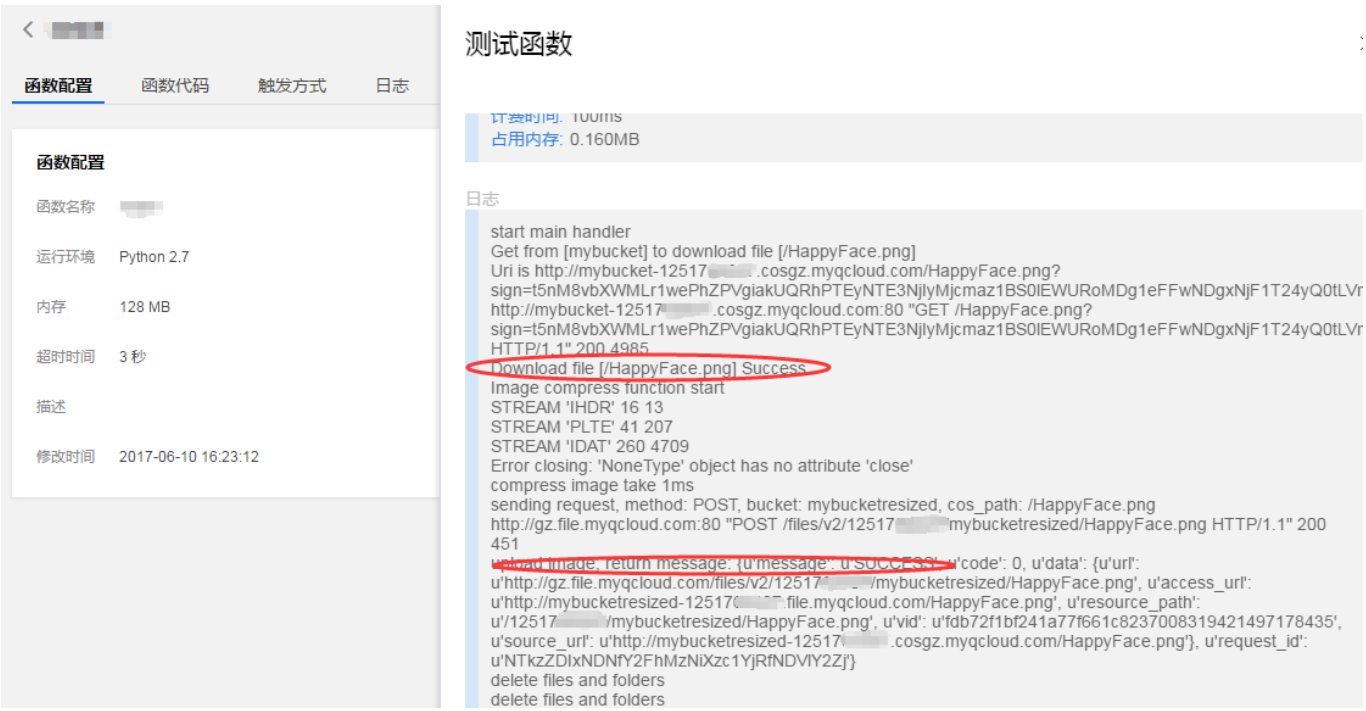
/HappyFace.jpg

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":"appId",
"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) 点击【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：



5) 前往对象存储控制台，点击步骤一：准备 COS Bucket 中创建的

mybucketresized

，观察是否有名为

HappyFace.png

的缩略图产生。

6) 下载该图片，对比观察它和原图片的大小。

步骤四：添加触发器

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，点击【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”，点击【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。

根据CMQ中的消息发送邮件

示例说明

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"
}
```

步骤一：创建 CMQ Topic 主题模式队列

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. 点击【主题订阅】选项卡，并切换地域为【华南地区（广州）】。
3. 点击【新建】按钮以新建队列，在弹出窗口中写入主题名

   sendEmailQueue

   。
4. 点击【创建】，完成队列创建。

步骤二：创建并测试 sendEmail 函数

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】地域下点击【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，点击【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，点击【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，可设置为你的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，点击【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，点击【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":
"{\"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) 点击【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。

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。

步骤三：添加触发器并测试

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，点击【添加触发方式】按钮.

2. 选择 【CMQ主题订阅触发】，CMQ Topoic 选择步骤一：创建 CMQ Topic 主题模式队列中创建的

   sendEmailQueue

   ，点击【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

   ，点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API网关提供API服务

示例说明

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

步骤一：创建并测试 blogArticle 函数

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，点击【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，点击【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 json.dumps({"errorCode":410,"errorMsg":"event is not come from api gateway"})
```

```
  if event["requestContext"]["path"] != "/article/{articleId}" and event["requestContext"]["path"] !=
"/article":
  return json.dumps({"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 json.dumps(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 json.dumps(article)
  return json.dumps({"errorCode":412,"errorMsg":"article is not found"})
  return json.dumps({"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，点击【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": "abdcdxxxxxxxsdfs"
  },
  "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.myqloud.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": "abdcdxxxxxxxsdfs"
  },
  "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.myqloud.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"}

步骤二：创建并测试 API 服务

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

   ，点击提交创建。
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，点击【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】选项卡中点击【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。

步骤三：发布 API 服务并在线验证

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。