# Learning Swift

Email: wu.wenxiang@99cloud.net

Email: maodouzi@gmail.com

Sina Weibo: @wu_wenxiang

---

## 概述

Swift是OpenStack的组件之一，是一个分布式的对象存储系统。

## 安装

### 安装基本系统

请参见《Learning DevStack》篇中，“安装基本系统”章节。

### 安装Swift

1. 安装步骤
   1. 系统更新

      ```
      sudo apt-get update && sudo apt-get dist-upgrade -y
      ```

   2. 安装依赖的软件包

      ```
      sudo apt-get install build-essential python-dev python-setupt
          python-pip libxml2-dev libxslt-dev memcached xfsprogs -y
      ```

   3. 下载swift源码包并安装配置

      ```
      cd ~ && git clone git://github.com/openstack/swift.git
      cd ~/swift && sudo pip install -r tools/pip-requires
      cd ~/swift && sudo python setup.py install
      sudo groupadd swift
      sudo useradd -g swift swift
      sudo mkdir /etc/swift
      sudo touch /etc/swift/swift.conf
      sudo chmod o+w /etc/swift/swift.conf
      echo -e "[swift-hash]\nswift_hash_path_suffix = 9ce957bfb8fbf
          > /etc/swift/swift.conf
      sudo chmod o-w /etc/swift/swift.conf
      sudo mkdir /var/cache/swift
      ```

```
sudo chown -R swift:swift /var/cache/swift
```

4. SSL Key

```
openssl version && cd /etc/swift
sudo openssl req -new -x509 -nodes -out cert.crt -keyout cert
```

5. 配置和启动memcached

```
sudo sed -i 's/127.0.0.1/0.0.0.0/g' /etc/memcached.conf
sudo service memcached restart
```

6. Keystone服务器上添加swift的services和endpoint

   参见《Learning Keystone》中“Keystone安装”篇

7. 配置文件proxy-server.conf

```
[DEFAULT]
bind_ip = 0.0.0.0
bind_port = 8080
swift_dir = /etc/swift
workers = 1
user = swift
cert_file = /etc/swift/cert.crt
key_file = /etc/swift/cert.key
log_name = swift
log_facility = LOG_LOCAL0
log_level = DEBUG

[pipeline:main]
pipeline = catch_errors healthcheck cache ratelimit authtoken
keystone proxy-server

[app:proxy-server]
use = egg:swift#proxy
account_autocreate = true
log_level = DEBUG

[filter:authtoken]
paste.filter_factory = keystone.middleware.auth_token:filter_
#the host must point to your keystone server
auth_host = localhost
auth_port = 35357
auth_token = ADMIN
auth_protocol=http
service_host = localhost
service_port = 5000
admin_token = ADMIN
admin_user = admin
admin_password = openstack
admin_tenant_name = adminTenant
delay_auth_decision = 1

[filter:keystone]
```

```
paste.filter_factory = keystone.middleware.swift_auth:filter_
operator_roles = adminRole, swiftoperator
is_admin = true
#reseller_prefix=AUTH

[filter:healthcheck]
use = egg:swift#healthcheck

[filter:cache]
use = egg:swift#memcache
memcache_servers = 127.0.0.1:11211

[filter:ratelimit]
use = egg:swift#ratelimit

[filter:domain_remap]
use = egg:swift#domain_remap

[filter:catch_errors]
use = egg:swift#catch_errors
```

8. 存储区的创建

本文为Swift安排了3块VirtIO disk，大小为5G，用fdisk分区：

```
sudo fdisk /dev/vdb
    n/p/默认/默认/默认/w
sudo fdisk /dev/vdc
    n/p/默认/默认/默认/w
sudo fdisk /dev/vdd
    n/p/默认/默认/默认/w
```

格式化：

```
sudo mkfs.xfs -i size=1024 /dev/vdb1
sudo mkfs.xfs -i size=1024 /dev/vdc1
sudo mkfs.xfs -i size=1024 /dev/vdd1
```

开机自动mount分区：

```
sudo -s
echo "/dev/vdb1 /srv/1/node/vdb1 xfs noatime,nodiratime,\
    nobarrier,logbufs=8 0 0" >> /etc/fstab
echo "/dev/vdc1 /srv/2/node/vdc1 xfs noatime,nodiratime,\
    nobarrier,logbufs=8 0 0" >> /etc/fstab
echo "/dev/vdd1 /srv/3/node/vdd1 xfs noatime,nodiratime,\
    nobarrier,logbufs=8 0 0" >> /etc/fstab
exit
```

9. 创建Node

```
sudo mkdir -p /srv/1/node/vdb1
sudo mkdir -p /srv/2/node/vdc1
sudo mkdir -p /srv/3/node/vdd1
sudo mount -a
sudo chown -R swift:swift /srv/1/node
sudo chown -R swift:swift /srv/2/node
sudo chown -R swift:swift /srv/3/node
```

10. 配置rsyncd：/etc/rsyncd.conf

```
uid = swift
gid = swift
log file = /var/log/rsyncd.log
pid file = /var/run/rsyncd.pid
address = 127.0.0.1

[account6012]
max connections = 25
path = /srv/1/node/
read only = false
lock file = /var/lock/account6012.lock

[account6022]
max connections = 25
path = /srv/2/node/
read only = false
lock file = /var/lock/account6022.lock

[account6032]
max connections = 25
path = /srv/3/node/
read only = false
lock file = /var/lock/account6032.lock

[container6011]
max connections = 25
path = /srv/1/node/
read only = false
lock file = /var/lock/container6011.lock

[container6021]
max connections = 25
path = /srv/2/node/
read only = false
lock file = /var/lock/container6021.lock

[container6031]
max connections = 25
path = /srv/3/node/
read only = false
lock file = /var/lock/container6031.lock

[object6010]
max connections = 25
path = /srv/1/node/
read only = false
lock file = /var/lock/object6010.lock

[object6020]
```

```
max connections = 25
path = /srv/2/node/
read only = false
lock file = /var/lock/object6020.lock

[object6030]
max connections = 25
path = /srv/3/node/
read only = false
lock file = /var/lock/object6030.lock
```

11. 配置和启动rsync

```
sudo sed -i 's/RSYNC_ENABLE=false/RSYNC_ENABLE=true/g' \
    /etc/default/rsync
sudo service rsync start
```

12. 建Ring File

三个Ring，18表示分区将被处理为2 ^ 18th，2表示3个zone，1表示1小时，是限制分区数据转移的时间。

```
sudo swift-ring-builder account.builder create 18 2 1
sudo swift-ring-builder container.builder create 18 2 1
sudo swift-ring-builder object.builder create 18 2 1

sudo swift-ring-builder account.builder add \
    z1-192.168.122.139:6012/vdb1 100
sudo swift-ring-builder container.builder add \
    z1-192.168.122.139:6011/vdb1 100
sudo swift-ring-builder object.builder add \
    z1-192.168.122.139:6010/vdb1 100
sudo swift-ring-builder account.builder add \
    z2-192.168.122.139:6022/vdc1 100
sudo swift-ring-builder container.builder add \
    z2-192.168.122.139:6021/vdc1 100
sudo swift-ring-builder object.builder add \
    z2-192.168.122.139:6020/vdc1 100
sudo swift-ring-builder account.builder add \
    z3-192.168.122.139:6032/vdd1 100
sudo swift-ring-builder container.builder add \
    z3-192.168.122.139:6031/vdd1 100
sudo swift-ring-builder object.builder add \
    z3-192.168.122.139:6030/vdd1 100
```

当创建好了Ring文件， 你可以通过下面的命令来验证刚才添加的内容是否正确。

```
swift-ring-builder account.builder
swift-ring-builder container.builder
swift-ring-builder object.builder
```

如果没有问题，创建最终的ring：

```
sudo swift-ring-builder account.builder rebalance
sudo swift-ring-builder container.builder rebalance
sudo swift-ring-builder object.builder rebalance
```

13. 配置account-server, container-server和object-server

    创建目录：

```
sudo mkdir -p /etc/swift/account-server /etc/swift/container-
    /etc/swift/object-server
```

account-server配置文件：

```
/etc/swift/account-server/1.conf
[DEFAULT]
devices = /srv/1/node
mount_check = false
disable_fallocate = true
bind_port = 6012
user = swift
log_facility = LOG_LOCAL2
recon_cache_path = /var/cache/swift

[pipeline:main]
pipeline = recon account-server

[app:account-server]
use = egg:swift#account

[filter:recon]
use = egg:swift#recon

[account-replicator]
vm_test_mode = yes

[account-auditor]

[account-reaper]
```

建立另外两个同类的配置文件：

```
cd /etc/swift/account-server/
sudo cp 1.conf 2.conf
sudo sed -i "s/srv\/1\/node/srv\/2\/node/" 2.conf
sudo sed -i "s/6012/6022/" 2.conf
sudo cp 1.conf 3.conf
sudo sed -i "s/srv\/1\/node/srv\/3\/node/" 3.conf
sudo sed -i "s/6012/6032/" 3.conf
```

container-server配置文件：/etc/swift/container-server/1.conf

```
[DEFAULT]
devices = /srv/1/node
mount_check = false
disable_fallocate = true
```

```
bind_port = 6011
user = swift
log_facility = LOG_LOCAL2
recon_cache_path = /var/cache/swift

[pipeline:main]
pipeline = recon container-server

[app:container-server]
use = egg:swift#container

[filter:recon]
use = egg:swift#recon

[container-replicator]
vm_test_mode = yes

[container-updater]

[container-auditor]

[container-sync]
```

建立另外两个同类的配置文件:

```
cd /etc/swift/container-server
sudo cp 1.conf 2.conf
sudo sed -i "s/srv\/1\/node/srv\/2\/node/" 2.conf
sudo sed -i "s/6011/6021/" 2.conf
sudo cp 1.conf 3.conf
sudo sed -i "s/srv\/1\/node/srv\/3\/node/" 3.conf
sudo sed -i "s/6011/6031/" 3.conf
```

object-server配置文件：/etc/swift/object-server/1.conf

```
[DEFAULT]
devices = /srv/1/node
mount_check = false
disable_fallocate = true
bind_port = 6010
user = swift
log_facility = LOG_LOCAL2
recon_cache_path = /var/cache/swift

[pipeline:main]
pipeline = recon object-server

[app:object-server]
use = egg:swift#object

[filter:recon]
use = egg:swift#recon

[object-replicator]
vm_test_mode = yes
```

```
[object-updater]

[object-auditor]
```

建立另外两个同类的配置文件：

```
cd /etc/swift/object-server
sudo cp 1.conf 2.conf
sudo sed -i "s/srv\/1\/node/srv\/2\/node/" 2.conf
sudo sed -i "s/6010/6020/" 2.conf
sudo cp 1.conf 3.conf
sudo sed -i "s/srv\/1\/node/srv\/3\/node/" 3.conf
sudo sed -i "s/6010/6030/" 3.conf
```

14. keystone-signing

```
mkdir -p ~/keystone-signing
sudo chown -R swift:swift ~/keystone-signing
sudo mkdir -p /var/cache/swift
sudo chown -R swift:swift /var/cache/swift
```

2. 启动Swift

```
sudo swift-init all start
```

3. 测试

```
# 查看统计
swift -V 2 -A http://localhost:5000/v2.0 -U admin:admin -K 231 sta
# 创建一个container：testDir
swift -V 2 -A http://localhost:5000/v2.0 -U admin:admin -K 231 pos
    testDir
# 在testDir里上传一个object：testFile，命名为testFile1
swift -V 2 -A http://localhost:5000/v2.0 -U admin:admin -K 231 upl
    testDir testFile testFile1
```

## 源代码分析

### SwiftClient

1. 代码结构

Swift Client的主要代码文件如下：

```
./bin/swift # 命令行入口
./swiftclient/client.py # Client核心代码
./tests/test_swiftclient.py # Client单元测试代码
./tests/utils.py # 测试库函数
```

2. bin/swift

bin/swift启动了两个线程：print_thread和err_thread。这两个线程都是QueueFunctionThread类的实例。在这个类从Queue里取任务，然后执行。

执行的命令从命令行参数来，分为'delete', 'download', 'list', 'post', 'stat', 'upload'六类，分别对应于bin/swift module里六个st_前缀的。

以st_download为例，在这个方法里启动了两个线程，也是QueueFunctionThread类的实例，分别用来download container和object。对应的处理函数是st_download函数的闭包函数，_download_container和_download_object。

以_download_container为例，这个方法接受两个参数，一个是container的标识，另一个是swiftclient.client.Connection类的实例conn。_download_container的处理逻辑是调用conn实例中对应的API函数，这里是conn.get_container(container, marker=marker)。

其余类同。

3. swiftclient.client

   swiftclient.client.Connection类用_retry方法封装了定义在swiftclient.client模块中的各个API函数，如get_container。

   _retry(self, reset_func, func, *args,* *kwargs)的处理逻辑是：在有限的次数内，反复尝试向swift发起业务请求。在每一次请求中，函数体会先查找endpoint和token是否存在，如果没有，就向keystone请求验证get_auth(RAW_PATH_INFO = /v2.0/tokens，用到了keystone client的API来向keystone请求token并解析response)，得到了token和swift的endpoint再向swift请求。

   请求中的endpoint就是url，形如http://192.168.122.90:8080/v1/AUTH_f5a836cb571747a691f4dec2eb7af6a4)，token放在{'headers': {'X-Auth-Token': u'95728c28d6c54af7b0fed9929480466d'}里面。Swift收到请求后，会向keystone要求验证(RAW_PATH_INFO = /v2.0/tokens/95728c28d6c54af7b0fed9929480466d)，验证通过后会响应用户的请求，给出回复。

   如果向_retry函数传递了reset_func，那么每一次向swift请求失败，并且没有命中except时，就会执行reset_fun，通常用于触发预设的异常。

   被_retry封装的API也定义在这个模块里，比如get_container。这些API用HttpConnection和json向Swift请求服务。

4. tests.test_swiftclient

   运行方法：

   ```
   pip install tox
   ./run_tests.sh
   ```

## Swift