# 单证查询系统搭建说明

## 文档版本

版本：1.1

修改日期：2013年4月1日

## 准备工作

### 系统架构描述

全部网络分为三层：

第一层：swift存储仓库，包括5个存储节点和1个存储代理（proxy）服务器

第二层：包括查询服务器、和存储代理服务器，存储代理服务器跨第一层和第二层

第三层：查询服务器，对外提供web服务。

总共需要安装7个服务器

### Pxe安装服务器描述

所有服务器安装使用PXE方式，通过一个PC进行安装。

安装PC系统环境使用ubuntu12.04 Server LTS+dnsmasq+tftp+nginx+apt-cacher。

## 安装PC环境的配置

1. 安装操作系统ubuntu12.04 server LTS

设定：英文系统，hostname：PXEServer，用户：pxe，密码：123456

附带安装openssh-server。

1. 安装dnsmasq、tftpserver

安装：apt-get install dnsmasq tftpserver

修改配置文件：/etc/dnsmasq.conf

bogus-priv

filterwin2k

interface=eth0

dhcp-range=192.168.33.50,192.168.33.150,12h

dhcp-boot=pxelinux.0

enable-tftp

tftp-root=/var/lib/tftpboot

dhcp-authoritative

1. 安装nginx

首先安装gem，再安装passenger，最后通过passenger安装nginx

安装命令：

apt-get install rubygems libcurl4-openssl-dev

gem install passenger

passenger-install-nginx-module 期间都使用默认参数安装

1. 配置pxe的MAC地址检测文件

网络下载10.04的网络启动文件：

cd /var/lib/tftpboot/

wget <http://cn.archive.ubuntu.com/ubuntu/dists/lucid/main/installer-amd64/current/images/netboot/netboot.tar.gz>

各版本的文件不同，11.10为：

<http://cn.archive.ubuntu.com/ubuntu/dists/oneiric/main/installer-amd64/current/images/netboot/netboot.tar.gz>

12.04LTS为：

<http://cn.archive.ubuntu.com/ubuntu/dists/precise/main/installer-amd64/current/images/netboot/netboot.tar.gz>

解压缩文件：

tar -xzvf netboot.tar.gz

复制安装的系统文件到/opt/nginx/html/ubuntu目录下

假设安装目标的服务器MAC地址为d4-ae-52-64-b3-1d，

建立文件/var/lib/tftpboot/pxelinux.cfg/01-d4-ae-52-64-b3-1d，内容：

default install

label install

menu label ^Install

menu default

kernel ubuntu-installer/amd64/linux

append ks=http://192.168.33.1/ks.cfg preseed/url=http://192.168.33.1/ubuntu/preseed/ubuntu-server.seed vga=normal initrd=ubuntu-installer/amd64/initrd.gz – quiet

建立文件/opt/nginx/html/ks.cfg，内容：

lang en\_US.UTF-8

langsupport en\_US.UTF-8

keyboard us

mouse

timezone Asia/Shanghai

rootpw --disabled

#明文密码,不能少于6位,不然会中断自动应答,问你要强密码

#密文可以执行echo 1234567890 | openssl passwd -1 -stdin

#user www --fullname="www" --iscrypted --password $1$YKmaOIb5$/13bs7gCjaoH./ohFT0A7/

user ubuntu --fullname="ubuntu" --password ubuntu

install

url --url http://192.168.33.1/ubuntu

bootloader --location=mbr

zerombr yes

clearpart --all --initlabel --drives=sda

#size单位为M

part / --fstype ext4 --size 51200

part swap --size 4048

network --bootproto=dhcp --device=eth0

#静态ip

#network --bootproto=static --ip=192.168..168 --netmask=255.255.255.0 --gateway=192.168.5.112 \ nameserver=221.5.88.88 --device=eth0

firewall --disabled

skipx

%packages

@openssh-server

1. 安装apt-cacher

apt-get install apt-cacher

修改配置文件，/etc/apt-cacher/apt-cacher.conf

修改内容：

cache\_dir = /var/cache/apt-cacher

log\_dir = /var/log/apt-cacher

daemon\_port = 3142

group = www-data

user = www-data

allowed\_hosts = \*

path\_map = ubuntu cn.archive.ubuntu.com/ubuntu ; ubuntu-updates cn.archive.ubuntu.com/ubuntu ; ubuntu-security cn.archive.ubuntu.com/ubuntu security.debian.org/debian-security ; swift ppa.launchpad.net/swift-core/release/ubuntu ; swauth gholt.github.com/swauth/lucid

## Swift安装

### 所有节点安装

建立文件/etc/apt/sources.list.d/swift-core-release-lucid.list内容为：

deb http://192.168.33.1:3142/swift lucid main

apt-get update

apt-get install swift

建立swift配置目录

mkdir -p /etc/swift

chown -R swift:swift /etc/swift/

在第一个节点，建立配置文件

cat >/etc/swift/swift.conf <<EOF

[swift-hash]

# random unique string that can never change (DO NOT LOSE)

swift\_hash\_path\_suffix = `od -t x8 -N 8 -A n </dev/random`

EOF

在其他节点拷贝配置文件

scp Ubuntu@192.168.33.201:/etc/swift/swift.conf /etc/swift/

同时设置系统参数

export STORAGE\_LOCAL\_NET\_IP=192.168.33.201~205

export PROXY\_LOCAL\_NET\_IP=192.168.33.206

### 转到存储代理服务器

apt-get install swift-proxy memcached

生成密钥Key文件

cd /etc/swift

openssl req -new -x509 -nodes -out cert.crt -keyout cert.key

设置系统系统参数

export PROXY\_LOCAL\_NET\_IP=192.168.33.206

设置服务器地址

perl -pi -e "s/-l 127.0.0.1/-l $PROXY\_LOCAL\_NET\_IP/" /etc/memcached.conf

启动memcached服务

service memcached restart

建立配置文件/etc/swift/proxy-server.conf，内容：

cat >/etc/swift/proxy-server.conf <<EOF

[DEFAULT]

cert\_file = /etc/swift/cert.crt

key\_file = /etc/swift/cert.key

bind\_port = 443

workers = 8

user = swift

[pipeline:main]

pipeline = healthcheck cache tempauth proxy-server

[app:proxy-server]

use = egg:swift#proxy

allow\_account\_management = true

account\_autocreate = true

[filter:tempauth]

use = egg:swift#tempauth

user\_system\_root = testpass .admin https://$PROXY\_LOCAL\_NET\_IP/v1/AUTH\_system

[filter:healthcheck]

use = egg:swift#healthcheck

[filter:cache]

use = egg:swift#memcache

memcache\_servers = $PROXY\_LOCAL\_NET\_IP:11211

EOF

建立配置文件，创建account, container and object rings。生成器命令基本上是创建一个生成器文件与几个参数。参数的值：18代表2 ^第十八，价值的分区大小。这个“分区”的价值基础上的存储总量你期望你的整个环使用。价值3的复制品的数量与每个对象，最后被数个小时的限制移动分区超过一次。

cd /etc/swift

swift-ring-builder account.builder create 18 3 1

swift-ring-builder container.builder create 18 3 1

swift-ring-builder object.builder create 18 3 1

在代理节点，操作配置swift object rings操作如下

export ZONE= # 设置ZONE序号

export STORAGE\_LOCAL\_NET\_IP= # 存储节点的IP地址

export WEIGHT=100 # 存储的配重

export DEVICE=sdb1

swift-ring-builder account.builder add z$ZONE-$STORAGE\_LOCAL\_NET\_IP:6002/$DEVICE $WEIGHT

swift-ring-builder container.builder add z$ZONE-$STORAGE\_LOCAL\_NET\_IP:6001/$DEVICE $WEIGHT

swift-ring-builder object.builder add z$ZONE-$STORAGE\_LOCAL\_NET\_IP:6000/$DEVICE $WEIGHT

具体的步骤是例如有5个zone，每个zone有一台服务器，每台服务器的第二块硬盘作为存储之用，在存储代理中依次添加五个zone的配置。

首先是第一个zone

export ZONE=1 # 设置ZONE序号

export STORAGE\_LOCAL\_NET\_IP=192.168.33.201 # 存储节点的IP地址

export WEIGHT=100 # 存储的配重

export DEVICE=sdb1

swift-ring-builder account.builder add z$ZONE-$STORAGE\_LOCAL\_NET\_IP:6002/$DEVICE $WEIGHT

swift-ring-builder container.builder add z$ZONE-$STORAGE\_LOCAL\_NET\_IP:6001/$DEVICE $WEIGHT

swift-ring-builder object.builder add z$ZONE-$STORAGE\_LOCAL\_NET\_IP:6000/$DEVICE $WEIGHT

第二个zone

export ZONE=2 # 设置ZONE序号

export STORAGE\_LOCAL\_NET\_IP=192.168.33.202 # 存储节点的IP地址

export WEIGHT=100 # 存储的配重

export DEVICE=sdb1

swift-ring-builder account.builder add z$ZONE-$STORAGE\_LOCAL\_NET\_IP:6002/$DEVICE $WEIGHT

swift-ring-builder container.builder add z$ZONE-$STORAGE\_LOCAL\_NET\_IP:6001/$DEVICE $WEIGHT

swift-ring-builder object.builder add z$ZONE-$STORAGE\_LOCAL\_NET\_IP:6000/$DEVICE $WEIGHT

第三个zone

export ZONE=3 # 设置ZONE序号

export STORAGE\_LOCAL\_NET\_IP=192.168.33.203 # 存储节点的IP地址

export WEIGHT=100 # 存储的配重

export DEVICE=sdb1

swift-ring-builder account.builder add z$ZONE-$STORAGE\_LOCAL\_NET\_IP:6002/$DEVICE $WEIGHT

swift-ring-builder container.builder add z$ZONE-$STORAGE\_LOCAL\_NET\_IP:6001/$DEVICE $WEIGHT

swift-ring-builder object.builder add z$ZONE-$STORAGE\_LOCAL\_NET\_IP:6000/$DEVICE $WEIGHT

第四个zone

export ZONE=4 # 设置ZONE序号

export STORAGE\_LOCAL\_NET\_IP=192.168.33.204 # 存储节点的IP地址

export WEIGHT=100 # 存储的配重

export DEVICE=sdb1

swift-ring-builder account.builder add z$ZONE-$STORAGE\_LOCAL\_NET\_IP:6002/$DEVICE $WEIGHT

swift-ring-builder container.builder add z$ZONE-$STORAGE\_LOCAL\_NET\_IP:6001/$DEVICE $WEIGHT

swift-ring-builder object.builder add z$ZONE-$STORAGE\_LOCAL\_NET\_IP:6000/$DEVICE $WEIGHT

第五个zone

export ZONE=5 # 设置ZONE序号

export STORAGE\_LOCAL\_NET\_IP=192.168.33.205 # 存储节点的IP地址

export WEIGHT=100 # 存储的配重

export DEVICE=sdb1

swift-ring-builder account.builder add z$ZONE-$STORAGE\_LOCAL\_NET\_IP:6002/$DEVICE $WEIGHT

swift-ring-builder container.builder add z$ZONE-$STORAGE\_LOCAL\_NET\_IP:6001/$DEVICE $WEIGHT

swift-ring-builder object.builder add z$ZONE-$STORAGE\_LOCAL\_NET\_IP:6000/$DEVICE $WEIGHT

zone的值要从1开始，每次加1

确认一下zone的配置

swift-ring-builder account.builder

swift-ring-builder container.builder

swift-ring-builder object.builder

调整一下zone的负载配置

swift-ring-builder account.builder rebalance

swift-ring-builder container.builder rebalance

swift-ring-builder object.builder rebalance

复制存储代理节点account.ring.gz、container.ring.gz和 object.ring.gz到每个存储节点的/etc/swift目录下。

确认目录的权限

chown -R swift:swift /etc/swift

启动存储代理节点的代理进程

swift-init proxy start

### 配置存储节点

在每个节点做一下操作，以节点1为例：

安装相应程序

apt-get install swift-account swift-container swift-object xfsprogs

将第二块硬盘格式化为XFS volume格式，挂载到/srv/node目录下

fdisk /dev/sdb (set up a single partition)

mkfs.xfs -i size=1024 /dev/sdb1

echo "/dev/sdb1 /srv/node/sdb1 xfs noatime,nodiratime,nobarrier,logbufs=8 0 0" >> /etc/fstab

mkdir -p /srv/node/sdb1

mount /srv/node/sdb1

chown -R swift:swift /srv/node

建立rsyncd配置文件/etc/rsyncd.conf：

cat >/etc/rsyncd.conf <<EOF

uid = swift

gid = swift

log file = /var/log/rsyncd.log

pid file = /var/run/rsyncd.pid

address = $STORAGE\_LOCAL\_NET\_IP

[account]

max connections = 2

path = /srv/node/

read only = false

lock file = /var/lock/account.lock

[container]

max connections = 2

path = /srv/node/

read only = false

lock file = /var/lock/container.lock

[object]

max connections = 2

path = /srv/node/

read only = false

lock file = /var/lock/object.lock

EOF

修改/etc/default/rsync文件内容

perl -pi -e 's/RSYNC\_ENABLE=false/RSYNC\_ENABLE=true/' /etc/default/rsync

启动rsync服务

service rsync start

建立配置文件/etc/swift/account-server.conf:

cat >/etc/swift/account-server.conf <<EOF

[DEFAULT]

bind\_ip = $STORAGE\_LOCAL\_NET\_IP

workers = 2

[pipeline:main]

pipeline = account-server

[app:account-server]

use = egg:swift#account

[account-replicator]

[account-auditor]

[account-reaper]

EOF

以及/etc/swift/container-server.conf:

cat >/etc/swift/container-server.conf <<EOF

[DEFAULT]

bind\_ip = $STORAGE\_LOCAL\_NET\_IP

workers = 2

[pipeline:main]

pipeline = container-server

[app:container-server]

use = egg:swift#container

[container-replicator]

[container-updater]

[container-auditor]

EOF

最后 /etc/swift/object-server.conf:

cat >/etc/swift/object-server.conf <<EOF

[DEFAULT]

bind\_ip = $STORAGE\_LOCAL\_NET\_IP

workers = 2

[pipeline:main]

pipeline = object-server

[app:object-server]

use = egg:swift#object

[object-replicator]

[object-updater]

[object-auditor]

EOF

都完成后启动服务

swift-init all start

### 建立管理员权限

进入存储代理服务器，建立管理员，用户名是system:root，密码testpass

curl -k -v -H 'X-Storage-User: system:root' -H 'X-Storage-Pass: testpass' https://$PROXY\_LOCAL\_NET\_IP:8080/auth/v1.0

会得到用户的X-Auth-Token和X-Storage-Url

检查用户权限

curl -k -v -H 'X-Auth-Token: <token-from-x-auth-token-above>' <url-from-x-storage-url-above>

### 安装swauth认证

下载 swauth 包，解压缩

运行 *python setup.py install* 命令安装

编辑配置文件 /etc/swift/proxy-server.conf

修改 [pipeline:main] 节中的

pipeline = healthcheck cache **tempauth** proxy-server

将 **tempauth** 修改为 **swauth** 并添加

[filter:swauth]

use = egg:swauth#swauth

set log\_name = swauth

super\_admin\_key = swauthkey

default\_swift\_cluster = local#https://$PROXY\_LOCAL\_NET\_IP/v1

到配置文件内

运行 *swift-init proxy reload* 命令使配置生效

检测swauth：

swauth-prep -K swauthkey -A <https://13.141.43.80/auth>，##没有错误输出表示正常。

添加swauth用户：

swauth-add-user -K swauthkey -A <https://13.141.43.80/auth> -a test tester testing

查看用户信息：

swauth-list -K swauthkey -A https://13.141.43.80/auth

### 安装Swift1.8.0

在安装第一步添加源的时候改为，建立文件/etc/apt/sources.list.d/grizzly.list，内容：

deb http://13.141.43.17:3142/grizzly precise-updates/grizzly main

deb http://13.141.43.17:3142/grizzly precise-proposed/grizzly main

之后swauth安装版本1.0.8。

### Swift升级（1.4.6🡪1.8.0）

复制源1.4.6服务器中每个服务器的/etc/swift文件到新的1.8.0服务器中，确认proxy服务器中所有IP地址相关的配置，包括文件：

/etc/swift/proxy-server.conf

/etc/memcached.conf

确认存储节点服务器中所有IP地址相关的配置，包括文件：

/etc/rsyncd.conf

/etc/swift/account-server.conf

/etc/swift/container-server.conf

/etc/swift/object-server.conf

在proxy使用swift-ring-builder的set\_info命令修改ring中的每个存储设备的信息，例如：

swift-ring-builder account.builder set\_info d0 13.141.43.81:6002/sdb1

完成后重新生成ring.gz文件：

swift-ring-builder account.builder write\_ring

swift-ring-builder container.builder write\_ring

swift-ring-builder object.builder write\_ring

重启swift：

swift-init all restart

确认swauth账户test正常

swauth-list -K swauthkey -A <https://13.141.43.80/auth> test

如果proxy修改了IP地址，需要修改swauth用户的IP地址信息：

swauth-set-account-service -K swauthkey -A https://13.141.43.80/auth test storage local https://13.141.43.80:443/v1/AUTH\_121195c0-99a2-4d17-b127-fea144b6a58b

此处的local要和之前proxy-server.conf中default\_swift\_cluster一项中的值相对应。

### 常用的swift命令：

检查swift的状态

swift -A https://$PROXY\_LOCAL\_NET\_IP/auth/v1.0 -U system:root -K testpass stat

复制bigfile1.tgz到名为myfiles的container中

swift -A https://$PROXY\_LOCAL\_NET\_IP:8080/auth/v1.0 -U system:root -K testpass upload myfiles bigfile1.tgz

下载名为myfiles的container中的所有文件

swift -A https://$PROXY\_LOCAL\_NET\_IP:8080/auth/v1.0 -U system:root -K testpass download myfiles

备份swift的配置文件到名为builders的container中

swift -A https://$PROXY\_LOCAL\_NET\_IP:8080/auth/v1.0 -U system:root -K testpass upload builders /etc/swift/\*.builder

遍历swift的所有container

swift -A https://$PROXY\_LOCAL\_NET\_IP:8080/auth/v1.0 -U system:root -K testpass list

遍历名为builders的container中所有的文件

swift -A https://$PROXY\_LOCAL\_NET\_IP:8080/auth/v1.0 -U system:root -K testpass list builders

## 查询服务器的部署

### 操作系统的安装

查询服务器安装ubuntu12.04操作系统。

### 修改sudo不用输入密码

sudo visudo

在文件结尾增加一行：

%david ALL=(ALL) NOPASSWD: ALL

### 安装所需的软件

sudo apt-get update

sudo apt-get upgrade

sudo apt-get install build-essential curl git-core zlib1g-dev libssl-dev imagemagick cifs-utils keyutils smbfs vsftpd makepasswd libaio1 libdbi-perl libnet-daemon-perl libplrpc-perl

sudo apt-get install libcurl4-openssl-dev p7zip-full libtool inoticoming ghostscript unzip

### 安装数据库

系统使用percona封装的mysql，系统使用的是5.1版本，从percona网站下载文件：

mkdir percona

cd percona

wget <http://www.percona.com/redir/downloads/Percona-Server-5.1/LATEST/deb/oneiric/x86_64/libmysqlclient16-dev_5.1.63-rel13.4-443.oneiric_amd64.deb>

wget <http://www.percona.com/redir/downloads/Percona-Server-5.1/LATEST/deb/oneiric/x86_64/libmysqlclient16_5.1.63-rel13.4-443.oneiric_amd64.deb>

wget <http://www.percona.com/redir/downloads/Percona-Server-5.1/LATEST/deb/oneiric/x86_64/percona-server-client-5.1_5.1.63-rel13.4-443.oneiric_amd64.deb>

wget <http://www.percona.com/redir/downloads/Percona-Server-5.1/LATEST/deb/oneiric/x86_64/percona-server-common_5.1.63-rel13.4-443.oneiric_amd64.deb>

wget <http://www.percona.com/redir/downloads/Percona-Server-5.1/LATEST/deb/oneiric/x86_64/percona-server-server-5.1_5.1.63-rel13.4-443.oneiric_amd64.deb>

sudo dpkg –i \*.deb

sudo apt-get -f install

建立数据库，复制数据库文件。

### 在线安装RVM、ruby、rails

安装RVM

bash -s stable < <(curl -s https://raw.github.com/wayneeseguin/rvm/master/binscripts/rvm-installer )

安装ruby1.9.3

rvm install ruby-1.9.3

gem install rails

### 在线安装passenger、nginx

gem install passenger

sudo passenger-install-nginx-module

### 离线安装RVM、ruby

tar –xvf rvm-1.16.13.tar.gz

./install –auto

source ~/.rvm/scripts/rvm

cp ruby-1.9.3-p286.tar.bz2 rubygems-1.8.24.tgz yaml-0.1.4.tar.gz $rvm\_path/archives/

echo rubygems\_version=1.8.24 >> $rvm\_path/user/db

echo "" > ~/.rvm/gemsets/default.gems

echo "" > ~/.rvm/gemsets/global.gems

sh install\_sh.sh

rvm install 1.9.3-p286

cd gems/vendor/cache/

gem install bundler-1.2.1.gem

bundle install --local

### passenger离线安装nginx

首先确认nginx解压的目录，在安装nginx中需要例如：

/home/david/setup-package/irrgrmpn/nginx-1.3.7

rvmsudo passenger-install-nginx-module

在选择1、2时选择2离线安装。

### 安装node

cd node-v0.6.14/

./configure

make

sudo make install

### 安装jpeg支持库

下载文件jpegsrc.v6b.tar.gz

tar –xvf jpegsrc.v6b.tar.gz

/configure

make

sudo make install

### 安装spawning

下载spawning文件。

wget <http://pypi.python.org/packages/source/S/Spawning/Spawning-0.9.7.tar.gz#md5=d9e517ab33bba1961a4733aec8e1c92e>

执行安装命令

sudo apt-get install python-pip python-eventlet

python setup.py install

第二种安装方法：

首先解压tar zxvf install\_spawning.tar.gz，里面包含了许多依赖包，如：

（greenlet-0.3.tar.gz，eventlet-0.9.17.tar.gz，setuptools-0.6c11.tar.gz，Spawning.tar.gz）

1、更新源及依赖库

sudo aptitude update

sudo apt-get install python-dev gcc

2、安装eventlet的依赖包greenlet

tar zxvf greenlet-0.3.tar.gz

cd greenlet-0.3/

sudo python setup.py install

3、安装spawning的依赖包evenlet

tar zxvf eventlet-0.9.17.tar.gz

cd eventlet-0.9.17/

sudo python setup.py install

4、安装python依赖包

tar zxvf setuptools-0.6c11.tar.gz

cd setuptools-0.6c11/

sudo python setup.py install

5、最好安装可以安装spawning了

tar zxvf Spawning.tar.gz

cd Spawning-0.9.7/

sudo python setup.py install

最后，测试是否安装成功：

spawning –version

显示当前spawing的版本号

以上步骤可以在解压的install\_spawning 目录下，执行sh install.sh自动完成，安装过程中，可能需要提供一次管理员的密码。

### 布置查询项目

安装查询项目目录~/query-server/、~/tcollector/和~/bin/。

检录目录：

mkdir ~/var

mkdir ~/var/run

mkdir ~/var/log

mkdir ~/datastore

mkdir ~/incoming\_json

修改项目目录~/query-server/public/docimages/为真实目录。

### MySQL主从配置

搭建环境：主服务器地址13.141.43.230，从服务器地址13.141.43.225

数据库名称dms\_development

主数据库端：

修改配置文件/etc/mysql/my.cnf，注释以下内容：

#bind-address = 127.0.0.1

重启mysql

/etc/init.d/mysql restart

netstat -tap | grep mysql，确认mysql可以监听远程端口调用。

tcp        0      0 \*:mysql                 \*:\*                     LISTEN      -

登陆mysql，建立备份账号

GRANT REPLICATION SLAVE ON \*.\* TO 'slave'@'13.141.43.225' IDENTIFIED BY '123456';

FLUSH PRIVILEGES;

修改mysql配置文件/etc/mysql/my.cnf，修改以下内容

server-id = 1

log\_bin = /var/log/mysql/mysql-bin.log

expire\_logs\_days = 10

max\_binlog\_size = 100M

binlog\_do\_db = dms\_development

重启mysql服务，登陆mysql，将数据库锁住

USE dms\_development;

FLUSH TABLES WITH READ LOCK;   //不要退出这个终端，否则这个锁就不生效了。从服务器的数据库建好后。在主服务器执行解锁

查看主服务器数据库状态

mysql> SHOW MASTER STATUS;

+------------------+----------+--------------+------------------+

| File | Position | Binlog\_Do\_DB | Binlog\_Ignore\_DB |

+------------------+----------+--------------+------------------+

| mysql-bin.000001 | 19467 | dms\_development | |

+------------------+----------+--------------+------------------+

1 row in set (0.00 sec)

不要退出mysql，另外一个进程登陆服务器，备份mysql数据库

mysqldump -uroot -pXXXXXX --opt dms\_development > snapshot.sql

将snapshot.sql复制到从服务器上。

在上面锁住mysql数据库的界面，给数据库解锁

UNLOCK TABLES;

转到从服务器，修改mysql配置文件/etc/mysql/my.cnf

添加以下内容：

server-id=2

#master-connect-retry=60 #加上这句mysql无法启动，先注释掉

replicate-do-db= dms\_development

重启mysql数据库服务，CREATE DATABASE dms\_development;

退出数据库，恢复数据库：

mysqladmin --user=root --password=XXXXXX stop-slave

mysql -uroot -pXXXXX dms\_development < snapshot.sql

登录服务器，设置主服务器信息：

CHANGE MASTER TO MASTER\_HOST='dms-server', MASTER\_USER='slave', MASTER\_PASSWORD='123456', MASTER\_LOG\_FILE='mysql-bin.000001', MASTER\_LOG\_POS=19467;

注意MASTER\_LOG\_FILE和MASTER\_LOG\_POS要和主服务器相同。

启动从服务器模式

START SLAVE;

查看从数据库状态：

SHOW SLAVE STATUS \G，其中存在两项：

Slave\_IO\_Running: Yes

Slave\_SQL\_Running: Yes

### 服务器启动，挂载密钥u盘

首先确认系统挂载了RAM盘，在/etc/fstab中确认存在以下内容：

tmpfs /home/david/bin/conf tmpfs uid= david,gid= david,size=1M 0 0

查看磁盘信息状态

sudo fdisk –l

假设u盘盘符是/dev/sdb1，并且系统存在/media/u目录

sudo mount -o uid=david,gid=david,rw /dev/sdb1 /media/u

cp -r /media/u/conf/\* bin/conf/

将密码本解密

./recover.sh