查看系统内核

[root@localhost opt]# cat /etc/redhat-release

CentOS Linux release 7.2.1511 (Core)

查看版本

[root@localhost opt]# uname -r

3.10.0-327.el7.x86\_64

查看系统内核位数

[root@localhost opt]# uname -m

x86\_64

mysql设置密码:

mysql登陆后

set password =password('root');

flush privileges;

# mysql的安装

### 如果使用sqlyog登陆mysql报使用root无法登陆mysql的错误:

解决方法:

GRANT ALL PRIVILEGES ON \*.\* TO 'root'@'%' IDENTIFIED BY 'root' WITH GRANT OPTION;

### 数据库服务器的的 selinux 都要 disable

走一遍

systemctl stop firewalld.service #停止firewall

systemctl disable firewalld.service #禁止firewall开机启动

firewall-cmd --state #查看默认防火墙状态（关闭后显示notrunning，开启后显示running）

（永久关闭 selinux，请修改/etc/selinux/config，将 SELINUX 改为 disabled）

# vi /etc/selinux/config

SELINUX=disabled

然后重启

reboot;

### cmake安装方式

### 安装编译 MySQL 需要的依赖包：

查看是否安装过了cmake

[root@localhost mysql]# which cmake

/usr/bin/cmake

（mysql 从 5.5 版本开始，不再使用./configure 编译，而是使用 cmake 编译器，具体的cmake 编译参数可以参考 mysql 官网文档

http://dev.mysql.com/doc/refman/5.5/en/source-configuration-options.html，安装基本依赖包，先用 yum 安装 cmake、automake 、autoconf ，另 MySQL 5.5.x 需要最少安装的包有：bison,gcc、gcc-c++、ncurses-devel）：

# yum install make cmake gcc gcc-c++ bison bison-devel ncurses ncurses-devel autoconf automake

### 开始编译安装 mysql-5.6.26：

新增 mysql 用户组：

# groupadd mysql

新增 mysql 用户，并添加到 mysql 用户组：

# useradd -r -g mysql mysql

解压缩源码包：

# tar -zxvf mysql-5.6.26.tar.gz

进入解压缩源码目录：

# cd mysql-5.6.26

使用 cmake 源码安装 mysql（如果你打算安装到不同的路径，注意修改下面语句中/usr/local/mysql 和/home/mysql/data 路径！）

[root@edu-mysql-01 mysql-5.6.26]# cmake \

-DCMAKE\_INSTALL\_PREFIX=/usr/local/mysql \

-DMYSQL\_UNIX\_ADDR=/usr/local/mysql/mysql.sock \

-DDEFAULT\_CHARSET=utf8 \

-DDEFAULT\_COLLATION=utf8\_general\_ci \

-DWITH\_MYISAM\_STORAGE\_ENGINE=1 \

-DWITH\_INNOBASE\_STORAGE\_ENGINE=1 \

-DWITH\_ARCHIVE\_STORAGE\_ENGINE=1 \

-DWITH\_BLACKHOLE\_STORAGE\_ENGINE=1 \

-DWITH\_MEMORY\_STORAGE\_ENGINE=1 \

-DWITH\_READLINE=1 \

-DENABLED\_LOCAL\_INFILE=1 \

-**DMYSQL\_DATADIR**=/home/mysql/data \

-DMYSQL\_USER=mysql \

-DMYSQL\_TCP\_PORT=3306 \

-DENABLE\_DOWNLOADS=1

上面的这些复制完，回车，然后就开始 cmake 的过程，一般时间不会很长。

### 安装编译好的程序：

cmake 结束后开始编译源码，这一步时间会较长，请耐心等待：

# make

# make install

（注意：如果需要重装 mysql，在/usr/local/src/mysql-5.6.26 在执行下 make install 就可以了，不需要再 cmake 和 make）

清除安装临时文件 ：

# make clean

做一个链接

[root@localhost mysql-5.5.32]# ln -s /usr/local/mysql/ /opt/mysql-5.5.32

ln -s /usr/local/mysql/bin/mysql /usr/bin

修改 mysql 目录拥有者为 mysql 用户：

# chown -Rf mysql:mysql /usr/local/mysql

# chown -Rf mysql:mysql /home/mysql

### 4:配置环境变量

# vi /etc/profile

##在 profile 文件末尾增加两行

# mysql env param

PATH=/usr/local/mysql/bin:/usr/local/mysql/lib:$PATH

export PATH

使 PATH 搜索路径立即生效：

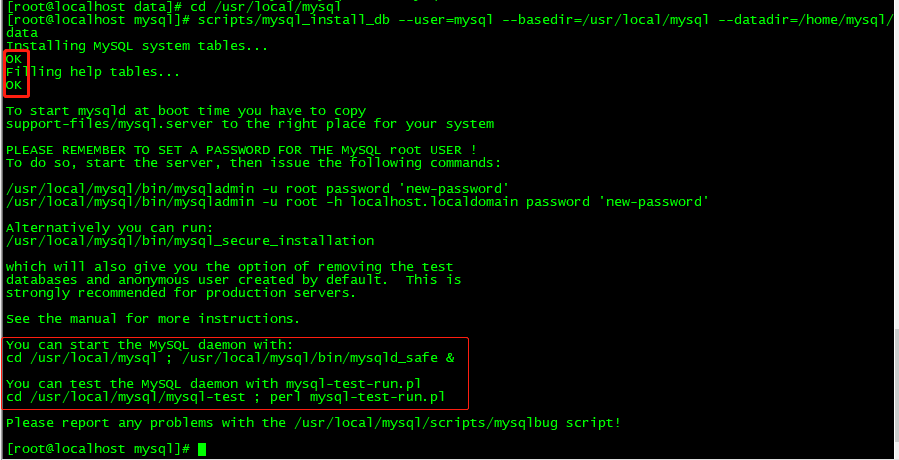
# source /etc/profile

### 初始化mysql脚本

执行初始化配置脚本，创建系统自带的数据库和表（注意：路径/home/mysql/data 需要换成你自定定义的数据库存放路径）：

[root@localhost data]# cd /usr/local/mysql

# scripts/mysql\_install\_db --user=mysql --basedir=/usr/local/mysql --datadir=/home/mysql/data



2个ok表示初始化成功

选择配置文件

mv /etc/my.cnf /etc/my.cnf.bak

# cp /usr/local/mysql/support-files/my-small.cnf /etc/my.cnf

### 给数据目录权限

[root@localhost data]# sudo su

[root@localhost data]# chown -R mysql.mysql /home/mysql/data

[root@localhost data]# chmod -R 1777 /tmp/

### 复制服务启动脚本：

# cp /usr/local/mysql/support-files/mysql.server /etc/init.d/mysql

给执行权限

[root@localhost mysql]# chmod +x /etc/init.d/mysql

### 启动 MySQL 服务：

# service mysql start

Starting MySQL.. SUCCESS!

（初次启动会在/usr/local/mysql 目录下生成 mysql.sock 文件）

设置 MySQL 开机自动启动服务：

# chkconfig mysql on

设置 MySQL 数据库 root 用户的本地登录密码（初始用户没有密码）：

# mysqladmin -u root password 'roncoo'

# mysql主从复制

### Mysql复制特点：

* 异步复制：主节点中一个用户请求一个写操作时，主接点不需要把写的数据在本地操作完成同时发送给从服务器并等待从服务器反馈写入完成，在响应用户，主机点只需要把写入操作在本地完成，就响应用户。但是，从节点中的数据有可能会落后主服务，可以使用（很多软件来检查是否落后）
* 主从数据不一致。

### 主从复制配置过程：

#### 主节点：

启用二进制日志。

为当前节点设置一个全局唯一的server\_id。

创建有复制权限的用户账号 REPLIACTION SLAVE ,REPLIATION CLIENT。

#### 从节点：

启动中继日志。

为当前节点设置一个全局唯一的server\_id。

使用有复制权限的用户账号连接至主节点，并启动复制线程。

### 环境:

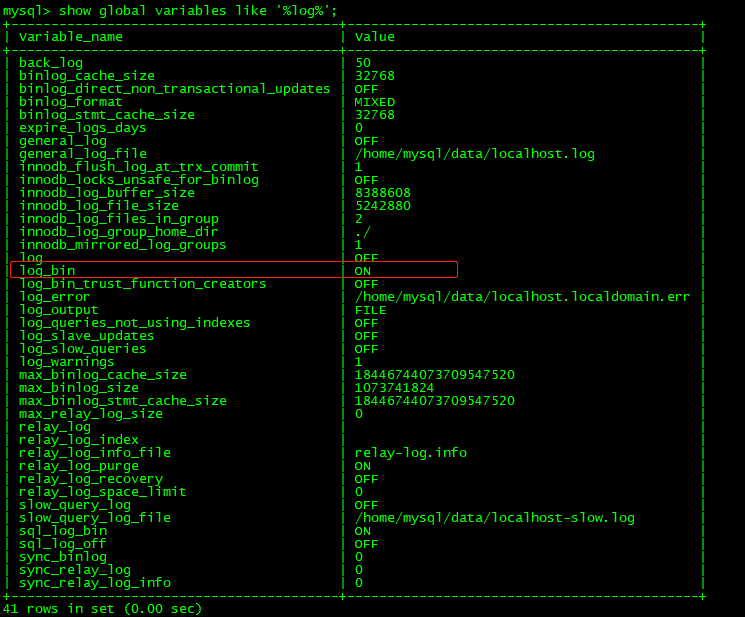
主 192.168.95.21

从 192.168.95.34

### 可能需要用到的命令:

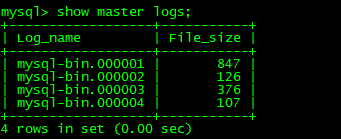
#### 查看二进制日志是否开启?

mysql> show global variables like '%log%';



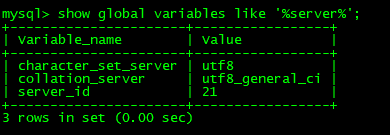
#### 查看主节点二进制日志列表

mysql> show master logs;



#### 查看主节点的server id

mysql> show global variables like '%server%';



### 编辑主节点my.conf:

[root@localhost ~]# vi /etc/my.cnf

server-id = 21

log-bin=mysql-bin

binlog\_format=mixed

innodb-file-per-table =ON

skip\_name\_resolve=ON

重启mysql

### 编辑从节点my.conf:

[root@localhost ~]# vi /etc/my.cnf

server-id = 34

relay-log-index=slave-relay-bin.index

relay-log=slave-relay-bin

重启mysql

### 在master上192.168.95.21中创建一个192.168.95.34主机中可以登录的MySQL用户

#### 创建用户repluser;

create user repluser;

mysql> grant replication slave, replication client on \*.\* to 'repluser'@'192.168.106.34' identified by 'replpass';

Query OK, 0 rows affected (0.00 sec)

## 刷新授权表信息

mysql> flush privileges;

## 查看 position 号，记下 position 号（从机上需要用到这个 position 号和现在的日志文件)

mysql> show master status;

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

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

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

| mysql-bin.000005 | 623 | | |

### 告知二进制文件名与位置

在slave 192.168.106.34中执行：

mysql>

change master to

master\_host='192.168.106.21',

master\_user='repluser',

master\_password='replpass',

master\_port=3306,

master\_log\_file='mysql-bin.000005',

master\_log\_pos=623;

Query OK, 0 rows affected (0.04 sec)

完成主从复制配置!

### 测试主从复制

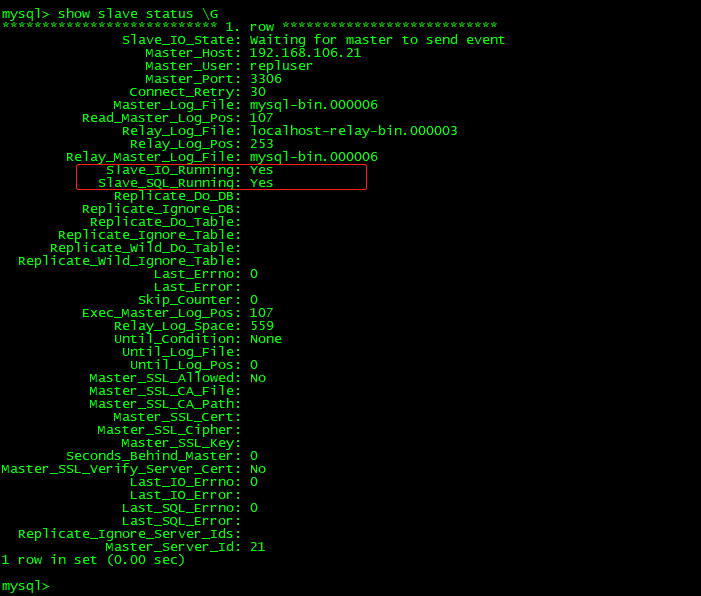
在从mysql客户端中开启复制

mysql> start slave;

Query OK, 0 rows affected (0.01 sec)

#查看主从复制是否配置成功

mysql> show slave status \G



当看到

Slave\_IO\_Running: YES、

Slave\_SQL\_Running: YES

才表明状态正常

### 实际测试：

--登陆192.168.106.21主MySQL

在主节点创建数据库，并查看主节点二进制日志信息

create database aa;

mysql> show master status;

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

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

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

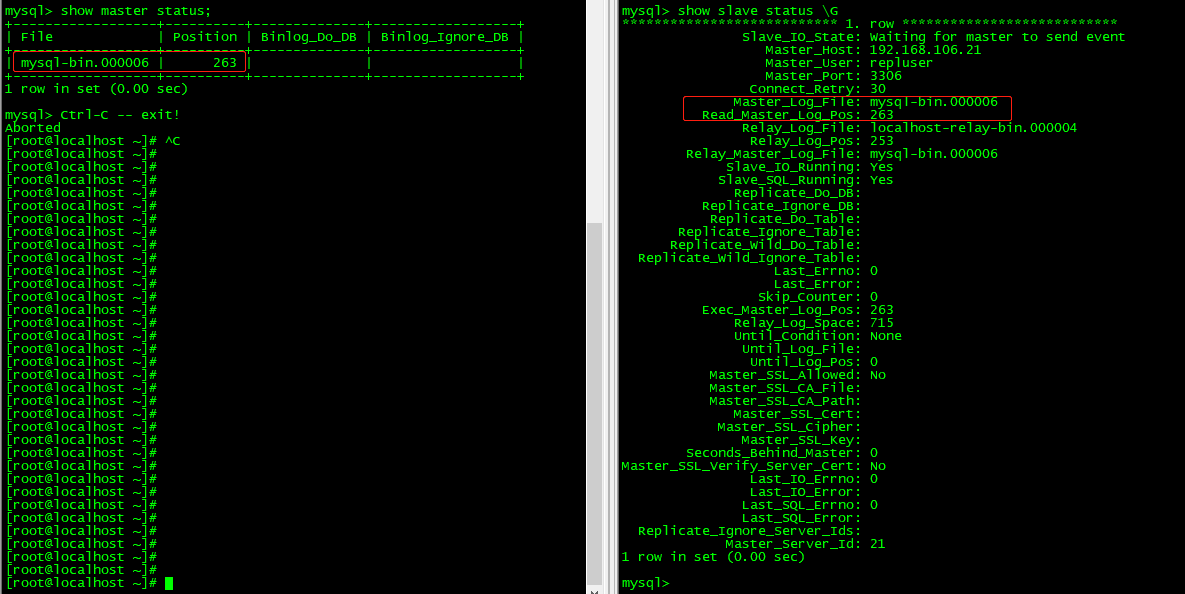
| mysql-bin.000006 | 263 | | |

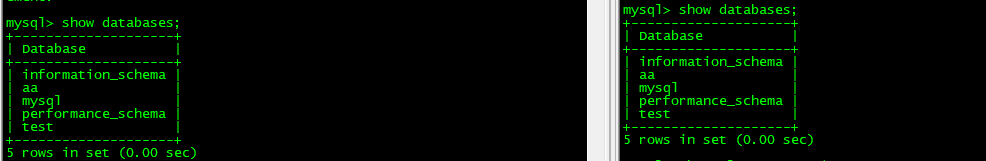
+------------------+----------+--------------+------------------+

1 row in set (0.00 sec)

在从节点查找二进制日志信息，并查看mydb数据库是否复制成功

mysql> show slave status \G





### 可能出现的问题:

#### MySQL同步故障：" Slave\_SQL\_Running:No" 两种解决办法

一般是事务回滚造成的：  
解决办法：  
mysql> stop slave ;  
mysql> set GLOBAL SQL\_SLAVE\_SKIP\_COUNTER=1;  
mysql> start slave ;

# mysql主主复制

### 修改/etc/my.cnf文件

--192.168.106.37

server-id=37   #开启的位置不能变,不然会失效

log-bin=mysql-bin #开启的位置不能变,不然会失效

auto-increment-increment = 2

auto-increment-offset = 1000

replicate-do-db=aa   #要同步的数据库，默认所有库

--192.168.106.41

server-id=41

log-bin=mysql-bin

auto-increment-increment = 2

auto-increment-offset = 1001

replicate-do-db=aa   #要同步的数据库，默认所有库

配置好后重启MySQL

### 开始构建主主复制

master1: 192.168.106.37

master2: 192.168.106.41

1:

现在master1: 192.168.106.37中

create user repluser;

mysql> grant replication slave, replication client on \*.\* to 'repluser'@'192.168.106.41' identified by 'replpass';

mysql> FLUSH PRIVILEGES;

2:

然后在master2:192.168.106.41中创建一个192.168.106.37主机中可以登录的MySQL用户

mysql> grant replication slave, replication client on \*.\* to 'repluser'@'192.168.106.37' identified by 'replpass';

mysql> FLUSH PRIVILEGES;

### 在主2中查看

mysql> show master status;

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

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

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

| mysql-bin.000001 | 501 | | |

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

1 row in set (0.00 sec)

告知二进制文件名与位置

在主1中执行:

change master to

master\_host='192.168.106.41',

master\_user='repluser',

master\_password='replpass',

master\_port=3306,

master\_log\_file='mysql-bin.000001',

master\_log\_pos=501;

### 在主1中查看

mysql> show master status;

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

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

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

| mysql-bin.000001 | 757 | | |

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

1 row in set (0.00 sec)

告知二进制文件名与位置

在主2中执行:

change master to

master\_host='192.168.106.37',

master\_user='repluser',

master\_password='replpass',

master\_port=3306,

master\_log\_file='mysql-bin.000001',

master\_log\_pos=757;

### 在主1中新增数据库

mysql> create database aa;

Query OK, 1 row affected (0.00 sec)

mysql> show databases;

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

| Database |

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

| information\_schema |

| aa |

| mysql |

| performance\_schema |

| test |

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

5 rows in set (0.00 sec)

mysql> use aa;

Database changed

mysql> create table tab1(id int auto\_increment,name varchar(10),primary key(id));

Query OK, 0 rows affected (0.00 sec)

mysql> show tables;

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

| Tables\_in\_aa |

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

| tab1 |

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

1 row in set (0.00 sec)

### 在主1中插入数据

insert into tab1 (name) value('11'),( '11'),( '11');

mysql> select \* from tab1;

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

| id | name |

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

| 1 | 11 |

| 2 | 11 |

| 3 | 11 |

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

3 rows in set (0.00 sec)

在主2中可以查到数据

### 在主2中插入数据

insert into tab1 (name) value('22'),( '22'),( '22');

在主1中可以查到数据

完工!

mysql多实例

共用一套mysql安装程序,

使用不同(也可以相同)的my.cnf配置文件,启动程序,数据文件

# MyCat 在 MySQL 主从复制基础上实现读写分离

### 因为 MyCat 是用 Java 开发的，因此 MyCat 运行需要安装 JDK（准确来说是 JRE 就够了），并且需要 JDK1.7 或以上版本

### 上传文件,解压

Mycat-server-1.6-RELEASE-20161028204710-linux.tar.gz

[root@localhost local]# cd mycat/

[root@localhost mycat]# ll

total 16

drwxr-xr-x 2 root root 4096 Oct 14 20:52 bin

drwxrwxrwx 2 root root 6 Feb 29 2016 catlet

drwxrwxrwx 4 root root 4096 Oct 14 20:52 conf

drwxr-xr-x 2 root root 4096 Oct 14 20:52 lib

drwxrwxrwx 2 root root 6 Oct 28 2016 logs

-rwxrwxrwx 1 root root 217 Oct 28 2016 version.txt

### 设置 MyCat 的环境变量

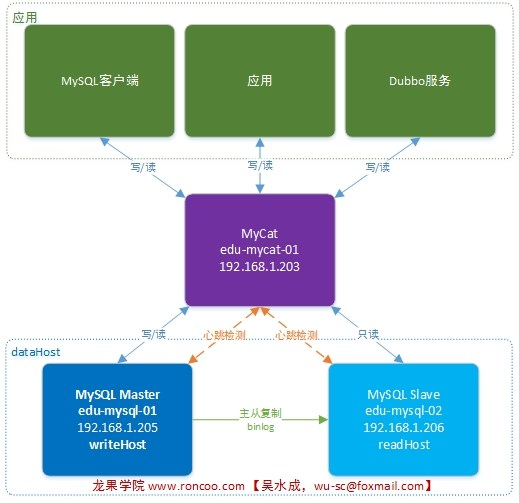
# vi /etc/profile

## mycat env

export MYCAT\_HOME=/usr/local/mycat

export PATH=$PATH:$MYCAT\_HOME/bin

# source /etc/profile



### 配置 MyCat

1、在配置 MyCat 前，请确认 MySQL 的主从复制安装配置已完成并正常运行。MySQL 主从数据的同步在 MySQL 中配置，MyCat 不负责数据同步的问题。

补充：

(1) MySQL 主从复制配置中，如果涉及到函数或存储过程的同步复制，需要在/etc/my.cnf 中的[mysqld]段中增加配置 log\_bin\_trust\_function\_creators=true

或在客户端中设置 set global log\_bin\_trust\_function\_creators = 1;

(2) 如果要做读写分离下的主从切换，那么从节点也有可能会变为写节点，因此从节点就不能设置为只读 read\_only=1 。

(3) Linux 版本的 MySQL，需要设置为 MySQL 大小写不敏感，否则可能会发生找不到表的问题。

可在/etc/my.cnf 的[mysqld]段中增加 lower\_case\_table\_names=1 。

2、配置 MyCat 的 schema.xml

schema.xml 是 MyCat 最重要的配置文件之一，用于设置 MyCat 的逻辑库、表、数据节点、dataHost 等内容，

[mycat@edu-mycat-01 conf]$ cd /usr/local/mycat/conf/

[mycat@edu-mycat-01 conf]$ vi schema.xml

*<?***xml version**="1.0"*?>***<!DOCTYPE mycat:schema SYSTEM** "schema.dtd"***>***<*mycat***:schema xmlns:***mycat*="http://io.mycat/">  
 <!-- 定义MyCat的逻辑库，逻辑库的概念与MySQL中的 database 概念相同 -->  
 <!-- schema name="rc\_schema1" checkSQLschema="false" sqlMaxLimit="100" dataNode="rc\_dn1"></schema -->  
 <!--schema name="pay\_schema1" checkSQLschema="false" sqlMaxLimit="100" dataNode="pay\_dn1"></schema-->  
 <**schema name**="aa" **checkSQLschema**="false" **sqlMaxLimit**="100" **dataNode**="dn1"></**schema**>  
 <!-- 其中checkSQLschema表明是否检查并过滤SQL中包含schema的情况，如逻辑库为 TESTDB，则可能写为select \* from  
 TESTDB.edu\_user，此时会自动过滤TESTDB，SQL变为select \* from edu\_user，若不会出现上述写法，则可以关闭属性为false -->  
 <!--sqlMaxLimit默认返回的最大记录数限制，MyCat1.4版本里面，用户的Limit参数会覆盖掉MyCat的sqlMaxLimit默认设置-->  
 <!-- 定义MyCat的数据节点 -->  
 <!-- dataNode name="rc\_dn1" dataHost="dtHost1" database="roncoo" / -->  
 <!-- dataNode name="pay\_dn1" dataHost="dtHost1" database="edu\_simple\_pay" / -->  
 <**dataNode name**="dn1" **dataHost**="dthost" **database**="aa"/>  
 <!-- dataNode 中的 name 数据表示节点名称， dataHost表示数据主机名称， database表示该节点要路由的数据库的名称 -->  
 <!-- dataHost配置的是实际的后端数据库集群（当然，也可以是非集群） -->  
 <!-- 注意：schema中的每一个dataHost中的host属性值必须唯一，否则会出现主从在所有dataHost中全部切换的现象 -->  
 <!-- 定义数据主机dtHost1，只连接到MySQL读写分离集群中的Master节点，不使用MyCat托管MySQL主从切换 -->  
 <!--  
 <dataHost name="dtHost1" maxCon="500" minCon="20" balance="0"  
 writeType="0" dbType="mysql" dbDriver="native" switchType="1" slaveThreshold="100">  
 <heartbeat>select user()</heartbeat>  
 <writeHost host="hostM1" url="192.168.1.205:3306" user="root" password="www.roncoo.com" />  
 </dataHost>  
 -->  
 <!-- 使用MyCat托管MySQL主从切换 -->  
 <!-- 定义数据主机dtHost2，连接到MySQL读写分离集群，并配置了读写分离和主从切换 -->  
 <!-- 通过show slave status检测主从状态，当主宕机以后，发生切换，从变为主，原来的主变为从，这时候show slave  
 status就会发生错误，因为原来的主没有开启slave，不建议直接使用switch操作，而是在DB中做主从对调。 -->  
 <!-- can have multi write hosts -->  
 <**dataHost name**="dthost" **maxCon**="1000" **minCon**="10" **balance**="0" **writeType**="0" **dbType**="mysql" **dbDriver**="native" **switchType**="2" **slaveThreshold**="100">  
 <**heartbeat**>show slave status</**heartbeat**>  
 <**writeHost host**="hostM1" **url**="192.168.106.21:3306" **user**="root" **password**="root"/>  
 <**writeHost host**="hostS1" **url**="192.168.106.34:3306" **user**="root" **password**="root"/>  
 </**dataHost**>  
 <!-- 参数balance决定了哪些MySQL服务器参与到读SQL的负载均衡中 -->  
 <!-- balance="0"，为不开启读写分离，所有读操作都发送到当前可用的writeHost上-->  
 <!-- balance="1"，全部的readHost与stand by writeHost参与select语句的负载均衡-->  
 <!-- balance="2"，所有读操作都随机的在writeHost、readHost上分发-->  
 <!-- MyCat1.4版本中，若想支持MySQL一主一从的标准配置，并且在主节点宕机的情况下，从节点还能读取数据，则需要在MyCat里  
 配置为两个writeHost并设置balance="1" -->  
 <!-- writeType="0"，所有写操作都发送到可用的writeHost上 -->  
 <!-- writeType="1"，仅仅对于galera for mysql集群这种多主多节点都能写入的集群起效，此时Mycat会随机选择一个  
 writeHost并写入数据，对于非galera for mysql集群，请不要配置writeType=1，会导致数据库不一致的严重问题 -->  
</*mycat***:schema**>

MyCat1.4 开始支持 MySQL 主从复制状态绑定的读写分离机制，让读更加安全可靠，配置如下：

MyCat 心跳检查语句配置为 show slave status ，dataHost 上定义两个新属性：

switchType="2" 与 slaveThreshold="100"，

此时意味着开启 MySQL 主从复制状态绑定的读写分离与切换机制，MyCat 心跳机制通过检测 show slave status 中的

"Seconds\_Behind\_Master", "Slave\_IO\_Running", "Slave\_SQL\_Running" 三 个 字段 来 确定 当 前主 从同 步 的 状态 以及Seconds\_Behind\_Master 主从复制时延，

当 Seconds\_Behind\_Master 大于 slaveThreshold 时，读写分离筛选器会过滤掉此 Slave机器，防止读到很久之前的旧数据，

而当主节点宕机后，切换逻辑会检查 Slave 上的 Seconds\_Behind\_Master 是否为 0，为 0 时则表示主从同步，可以安全切换，

否则不会切换。

### 配置 server.xml

server.xml 主要用于设置系统变量、管理用户、设置用户权限等。

[wusc@edu-mycat-01 conf]$ vi server.xml

*<?***xml version**="3.0" **encoding**="UTFa  
<!-- - - Licensed under the Apache License, Version 2.0 (the "License");  
 - you may not use this file except in compliance with the License. - You  
 may obtain a copy of the License at - - http://www.apache.org/licenses/LICENSE-2.0  
 - - Unless required by applicable law or agreed to in writing, software -  
 distributed under the License is distributed on an "AS IS" BASIS, - WITHOUT  
 WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. - See the  
 License for the specific language governing permissions and - limitations  
 under the License. -->  
**<!DOCTYPE mycat:server SYSTEM** "server.dtd"***>***<*mycat***:server xmlns:***mycat*="http://io.mycat/">  
 <**system**>  
 <**property name**="useSqlStat">0</**property**> <!-- 1为开启实时统计、0为关闭 -->  
 <**property name**="useGlobleTableCheck">0</**property**> <!-- 1为开启全加班一致性检测、0为关闭 -->  
  
 <**property name**="sequnceHandlerType">2</**property**>  
 <!-- <property name="useCompression">1</property>--> <!--1为开启mysql压缩协议-->  
 <!-- <property name="fakeMySQLVersion">5.6.20</property>--> <!--设置模拟的MySQL版本号-->  
 <!-- <property name="processorBufferChunk">40960</property> -->  
 <!--  
 <property name="processors">1</property>  
 <property name="processorExecutor">32</property>  
 -->  
 <!--默认为type 0: DirectByteBufferPool | type 1 ByteBufferArena-->  
 <**property name**="processorBufferPoolType">0</**property**>  
 <!--默认是65535 64K 用于sql解析时最大文本长度 -->  
 <!--<property name="maxStringLiteralLength">65535</property>-->  
 <!--<property name="sequnceHandlerType">0</property>-->  
 <!--<property name="backSocketNoDelay">1</property>-->  
 <!--<property name="frontSocketNoDelay">1</property>-->  
 <!--<property name="processorExecutor">16</property>-->  
 <!--  
 <property name="serverPort">8066</property> <property name="managerPort">9066</property>  
 <property name="idleTimeout">300000</property> <property name="bindIp">0.0.0.0</property>  
 <property name="frontWriteQueueSize">4096</property> <property name="processors">32</property> -->  
 <!--分布式事务开关，0为不过滤分布式事务，1为过滤分布式事务（如果分布式事务内只涉及全局表，则不过滤），2为不过滤分布式事务,但是记录分布式事务日志-->  
 <**property name**="handleDistributedTransactions">0</**property**>  
  
 <!--  
 off heap for merge/order/group/limit 1开启 0关闭  
-->  
 <**property name**="useOffHeapForMerge">1</**property**>  
  
 <!--  
 单位为m  
 -->  
 <**property name**="memoryPageSize">1m</**property**>  
  
 <!--  
 单位为k  
 -->  
 <**property name**="spillsFileBufferSize">1k</**property**>  
  
 <**property name**="useStreamOutput">0</**property**>  
  
 <!--  
 单位为m  
 -->  
 <**property name**="systemReserveMemorySize">384m</**property**>  
  
  
 <!--是否采用zookeeper协调切换 -->  
 <**property name**="useZKSwitch">true</**property**>  
  
  
 </**system**>  
  
 <!-- 全局SQL防火墙设置 -->  
 <!--  
 <firewall>  
 <whitehost>  
 <host host="127.0.0.1" user="mycat"/>  
 <host host="127.0.0.2" user="mycat"/>  
 </whitehost>  
 <blacklist check="false">  
 </blacklist>  
 </firewall>  
 -->  
  
 <**user name**="root">  
 <**property name**="password">root</**property**>  
 <**property name**="schemas">aa</**property**>  
  
 <!-- 表级 DML 权限设置 -->  
 <!--  
 <privileges check="false">  
 <schema name="TESTDB" dml="0110" >  
 <table name="tb01" dml="0000"></table>  
 <table name="tb02" dml="1111"></table>  
 </schema>  
 </privileges>  
 -->  
 </**user**>  
  
 <**user name**="user">  
 <**property name**="password">user</**property**>  
 <**property name**="schemas">aa</**property**>  
 <**property name**="readOnly">true</**property**>  
 </**user**>  
  
</*mycat***:server**>

### 防火墙中打开 8066 和 9066 端口

MyCat 的默认数据端口为 8066，mycat 通过这个端口接收数据库客户端的访问请求。

管理端口为 9066，用来接收 mycat 监控命令、查询 mycat 运行状况、重新加载配置文件等。

[root@edu-mycat-01 mycat]# vi /etc/sysconfig/iptables

增加：

## MyCat

-A INPUT -m state --state NEW -m tcp -p tcp --dport 8066 -j ACCEPT

-A INPUT -m state --state NEW -m tcp -p tcp --dport 9066 -j ACCEPT

重启防火墙：

[root@edu-mycat-01 mycat]# service iptables restart

关闭防火墙

走一遍

systemctl stop firewalld.service #停止firewall

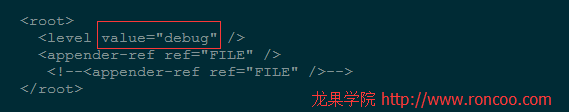
systemctl disable firewalld.service #禁止firewall开机启动

firewall-cmd --state #查看默认防火墙状态（关闭后显示notrunning，开启后显示running）

### **修改 log 日志级别为 debug**

，以便通过日志确认基于 MyCat 的 MySQL 数据库集群读写分离的数据操作状态（可以在正式上生产前改成 info 级别）

[mycat@edu-mycat-01 conf]$ vi /usr/local/mycat/conf/log4j.xml



### 在mycat服务器下启动 MyCat

[mycat@edu-mycat-01 bin]$ cd /usr/local/mycat/bin/

(1) 控制台启动，这种启动方式在控制台关闭后，MyCat 服务也将关闭，适合调试使用：

[mycat@edu-mycat-01 bin]$ ./mycat console

(2) 可以采用以下后台启动的方式：

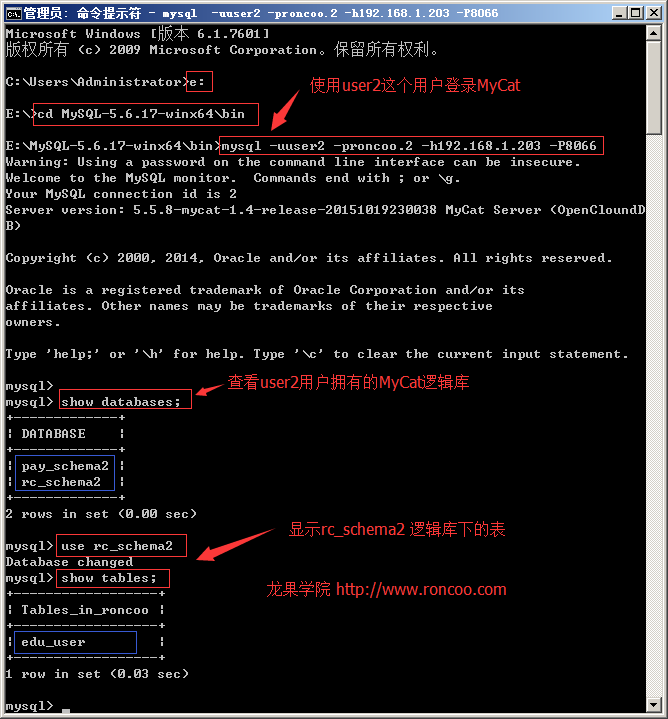
[mycat@edu-mycat-01 bin]$ ./mycat start

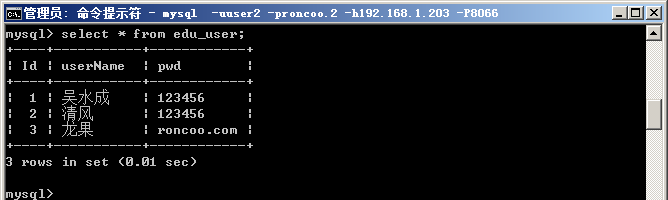
Starting Mycat-server...

（对应的，重启: mycat restart ， 关闭: mycat stop ）

### MyCat 连接测试

(1) 如果本地 Windows 安装有 MySQL，可以使用已有的 mysql 客户端远程操作 MyCat





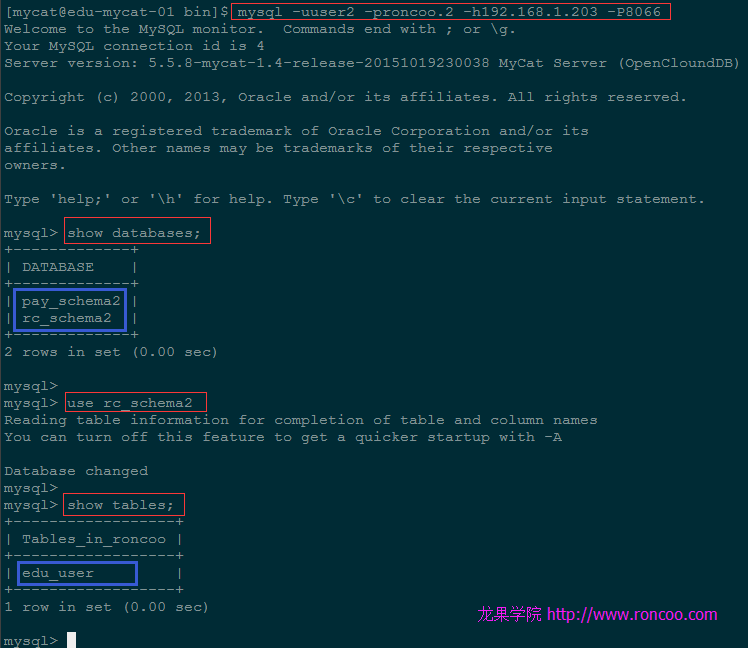
(2) 如果为了方便，需要在 MyCat 主机上对 MyCat 进行操作（把 MyCat 当是本地 MySQL 来操作），可以在 MyCat 节点主机上安装 MySQL 客户端：

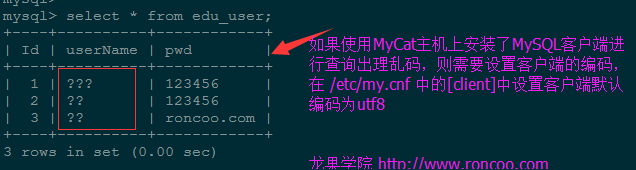
[mycat@edu-mycat-01 bin]$ su root

[root@edu-mycat-01 bin]# yum install mysql

使用安装好的 mysql 客户端登录 MyCat

[root@localhost bin]# mysql -uroot -proot -h192.168.106.82 -P8066





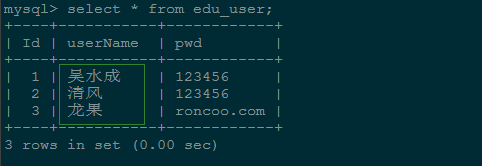
[root@edu-mycat-01 mycat]# vi /etc/my.cnf

增加：

[client]

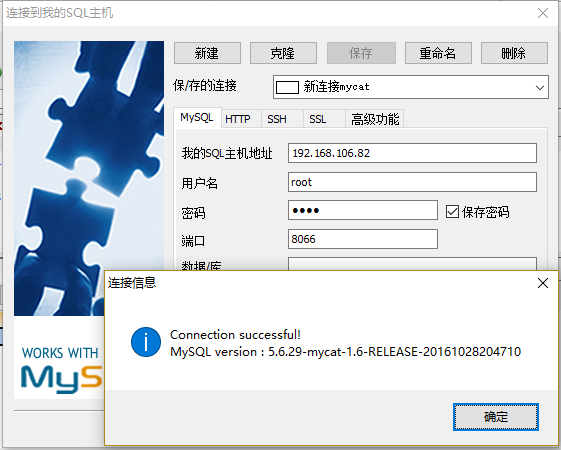
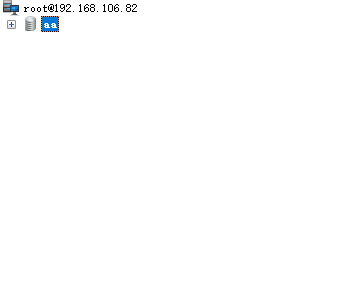
default-character-set=utf8

保存后再查询，乱码问题解决，如下：



(3) 使用第三方 MySQL 管理客户端连接 MyCat 测试（navicat 支持，MySQL-Front 兼容性不太

好），以 navicat 为例：

mysql> select \* from tb\_a;

mysql> insert into tb\_a (id) values(5);

多次执行以上插入语句，发现新增数据都是从 Master 节点（192.168.106.21）插进入的，并

且 Slave 节点通过 Binlog 同步了 Master 节点中的数据。

综上，基于 MyCat 的读写分离集群配置成功。

接下来计划课程：

MyCat 读写分离集群的主从容错（切换）、恢复；

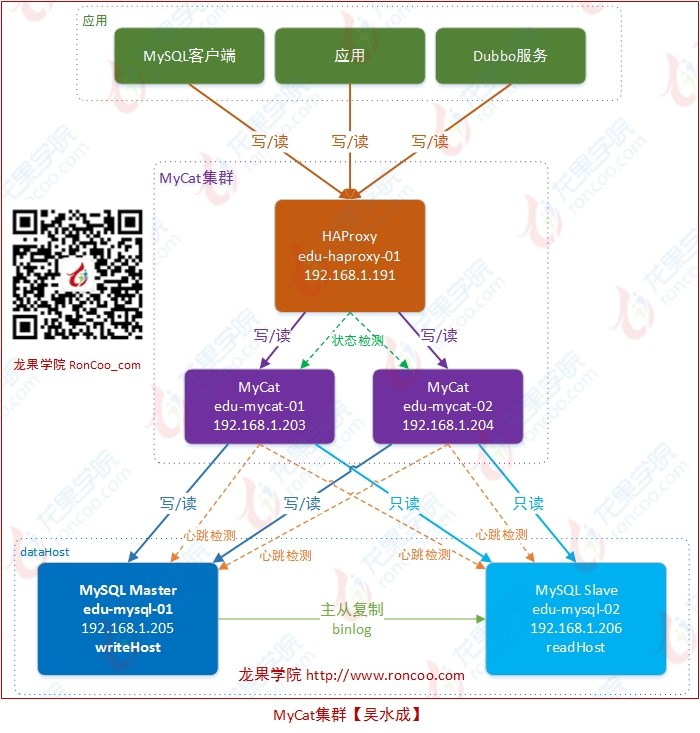
MyCat 的高可用集群 HAProxy + Keepalived + MyCat；

# MyCat 集 群部署 （ HAProxy + MyCat ）

<https://www.haproxy.org/download/1.8/src/>

haproxy下载





图解说明：

HAProxy 负责将请求分发到 MyCat 上，起到负载均衡的作用，同时 HAProxy 也能检测到 MyCat 是否存活，HAProxy 只会将请求转发到存活的 MyCat 上。如果一台 MyCat 服务器宕机，HAPorxy 转发请求时不会转发到宕机的 MyCat 上，所以 MyCat 依然可用。

### MyCat节点2的部署

MyCat主机 2（edu-mycat-02，192.168.1.204）请参考《高可用架构篇--第 15 节--MyCat 在 MySQL 主

从复制基础上实现读写分离》进行对等部署和做相应配置。

### **配置 MyCat 状态 检查服务**

**（在 MyCat 节点主机上配置 xinetd）**

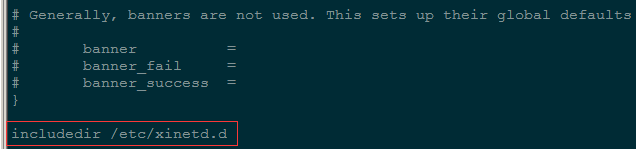
MyCat 服务主机（edu-mycat-01、edu-mycat-02）上需要增加 mycat 服务的状态检测脚本，并开放相应的检测端口，以提供给HAProxy对MyCat的服务状态进行检测判断。可以使用xinetd来实现，通过xinetd，HAProxy 可以用 httpchk 来检测 MyCat 的存活状态。（xinetd 即 extended internet daemon，xinetd 是新一代的网络守护进程服务程序，又叫超级 Internet 服务器。经常用来管理多种轻量级 Internet 服务。xinetd 提供类似于 inetd+tcp\_wrapper 的功能，但是更加强大和安全。xinetd 为 linux 系统的基础服务）

1、如果 xinetd 还没有安装，可使用如下命令安装：

# yum install xinetd

2、检查/etc/xinetd.conf 的末尾是否有 includedir /etc/xinetd.d ，没有就加上

# vi /etc/xinetd.conf



3、检查 /etc/xinetd.d 目录是否存在，不存在刚创建

# mkdir /etc/xinetd.d/

4、增加 MyCat 存活状态检测服务配置

# touch /etc/xinetd.d/mycat\_status

# vi /etc/xinetd.d/mycat\_status

增加以下内容：

service mycat\_status

{

flags = REUSE

## 使用该标记的 socket\_type 为 stream，需要设置 wait 为 no

socket\_type = stream ## 封包处理方式，Stream 为 TCP 数据包

port = 48700 ## 服务监听端口

wait = no ## 表示不需等待，即服务将以多线程的方式运行

user = root ## 执行此服务进程的用户

server =/usr/local/bin/mycat\_status ## 需要启动的服务脚本

log\_on\_failure += USERID ## 登录失败记录的内容

disable = no ## 要启动服务，将此参数设置为 no

}

5、添加 /usr/local/bin/mycat\_status 服务脚本

# touch /usr/local/bin/mycat\_status

# vi /usr/local/bin/mycat\_status

增加以下内容：

#!/bin/bash

#/usr/local/bin/mycat\_status.sh

# This script checks if a mycat server is healthy running on localhost.

# It will return:

# "HTTP/1.x 200 OK\r" (if mycat is running smoothly)

# "HTTP/1.x 503 Internal Server Error\r" (else)

mycat=`/usr/local/mycat/bin/mycat status | grep 'not running' | wc -l`

if [ "$mycat" = "0" ]; then

/bin/echo -e "HTTP/1.1 200 OK\r\n"

else

/bin/echo -e "HTTP/1.1 503 Service Unavailable\r\n"

fi

6、给新增脚本赋予可执行权限

# chmod a+x /usr/local/bin/mycat\_status

7、在 /etc/services 中加入 mycat\_status 服务

# vi /etc/services

在末尾加入：

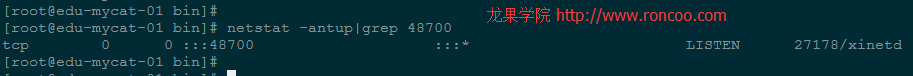
mycat\_status 48700/tcp # mycat\_status

保存后，重启 xinetd 服务

# service xinetd restart

8、验证 mycat\_status 服务是否成功启动

# netstat -antup|grep 48700



能看到上图这样的信息，说明服务配置成功。

9、MyCat 服务主机的防火墙上打开 48700 端口

# vi /etc/sysconfig/iptables

增加：

-A INPUT -m state --state NEW -m tcp -p tcp --dport 48700 -j ACCEPT

保存后重启防火墙

# service iptables restart

systemctl stop firewalld.service #停止firewall

systemctl disable firewalld.service #禁止firewall开机启动

firewall-cmd --state #查看默认防火墙状态（关闭后显示notrunning，开启后显示running）

### 脚本测试：

# /usr/local/bin/mycat\_status



### HAProxy 的 安装

解压haproxy，并安装

tar xf haproxy-1.7.5.tar.gz

cd haproxy-1.7.5

make TARGET=linux2628 PREFIX=/usr/local/haproxy

make install PREFIX=/usr/local/haproxy

安装成功后，查看版本

/usr/local/haproxy/sbin/haproxy -v

复制haproxy文件到/usr/sbin下   
因为下面的haproxy.init启动脚本默认会去/usr/sbin下找，当然你也可以修改，不过比较麻烦。

cp /usr/local/haproxy/sbin/haproxy /usr/sbin/

复制haproxy脚本，到/etc/init.d下

cp ./examples/haproxy.init /etc/init.d/haproxy

chmod 755 /etc/init.d/haproxy

创建系统账号

useradd -r haproxy

创建配置文件

mkdir /etc/haproxy

vi /etc/haproxy/haproxy.cfg

在配置文件haproxy.cfg中添加如下设置：

#全局配置

global

#设置日志

log 127.0.0.1 local3 info

chroot /usr/local/haproxy

#用户与用户组

user haproxy

group haproxy

#守护进程启动

daemon

#最大连接数

maxconn 4000

#默认配置

defaults

log global

mode http

option httplog

option dontlognull

timeout connect 5000

timeout client 50000

timeout server 50000

#前端配置，http\_front名称可自定义

frontend http\_front

# 发起http请求道80端口，会被转发到设置的ip及端口

bind \*:80

#haproxy的状态管理页面，通过/haproxy?stats来访问

stats uri /haproxy?stats

default\_backend http\_back

#后端配置，http\_back名称可自定义

backend http\_back

#负载均衡方式

#source 根据请求源IP

#static-rr 根据权重

#leastconn 最少连接者先处理

#uri 根据请求的uri

#url\_param 根据请求的url参数

#rdp-cookie 据据cookie(name)来锁定并哈希每一次请求

#hdr(name) 根据HTTP请求头来锁定每一次HTTP请求

#roundrobin 轮询方式

balance roundrobin

#设置健康检查页面

option httpchk GET /index.html

#传递客户端真实IP

option forwardfor header X-Forwarded-For

# inter 2000 健康检查时间间隔2秒

# rise 3 检测多少次才认为是正常的

# fall 3 失败多少次才认为是不可用的

# weight 30 权重

# 需要转发的ip及端口

server node1 192.168.106.82:8066 check inter 2000 rise 3 fall 3 weight 30

server node2 192.168.106.84:8066 check inter 2000 rise 3 fall 3 weight 30

---------------------

listen admin\_stats

bind 0.0.0.0:8888

mode http

stats uri /haproxy

stats realm Global\ statistics

stats auth admin:admin

# yum install rsyslog ## 没安装的情况下执行安装

# vi /etc/rsyslog.conf

把 $ModLoad imudp 和 $UDPServerRun 514 前面的 # 去掉

$ModLoad imudp ## 是模块名，支持 UDP 协议

$UDPServerRun 514

##允许 514 端口接收使用 UDP 和 TCP 协议转发过来的日志，

##而 rsyslog 在默认情况下，正是在 514 端口监听 UDP

并添加下面一行

local3.\* /var/log/haproxy.log

重启rsyslog

systemctl restart rsyslog

# centos 6 中木有systemctl命令，可以使用下面的命令

# service rsyslog restart

#### 启动 HAProxy

# service haproxy start

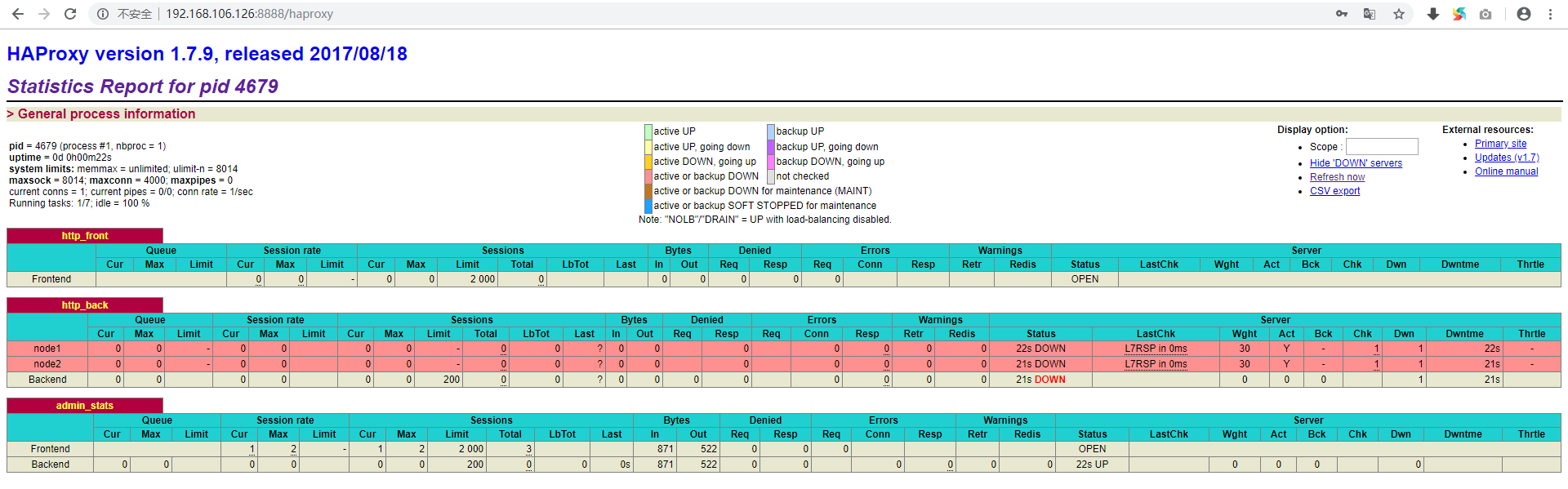
# ps -ef | grep haproxy

haproxy 23921 1 0 23:27 ? 00:00:00 /usr/sbin/haproxy -D -f /etc/haproxy/haproxy.cfg –p /var/run/haproxy.pid

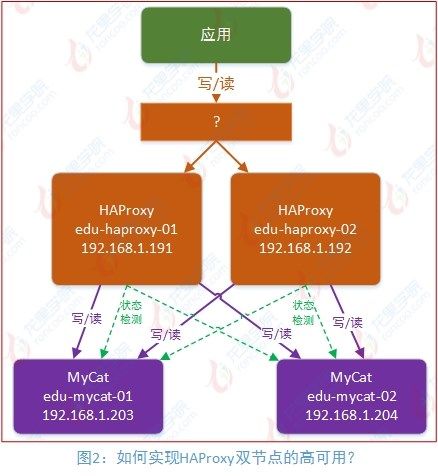
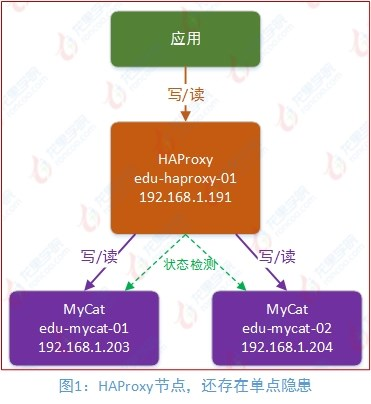
root 23924 23179 0 23:27 pts/1 00:00:00 grep haproxy

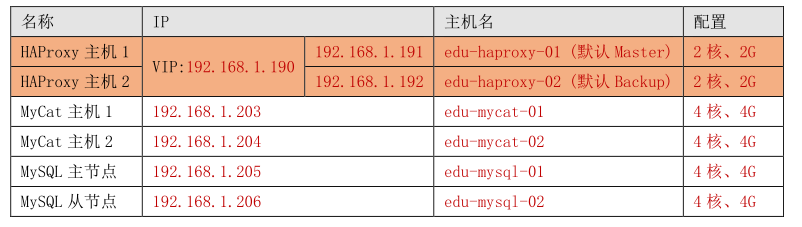
#### 使用 MySQL 客户端通过 HAProxy 连接 MyCat

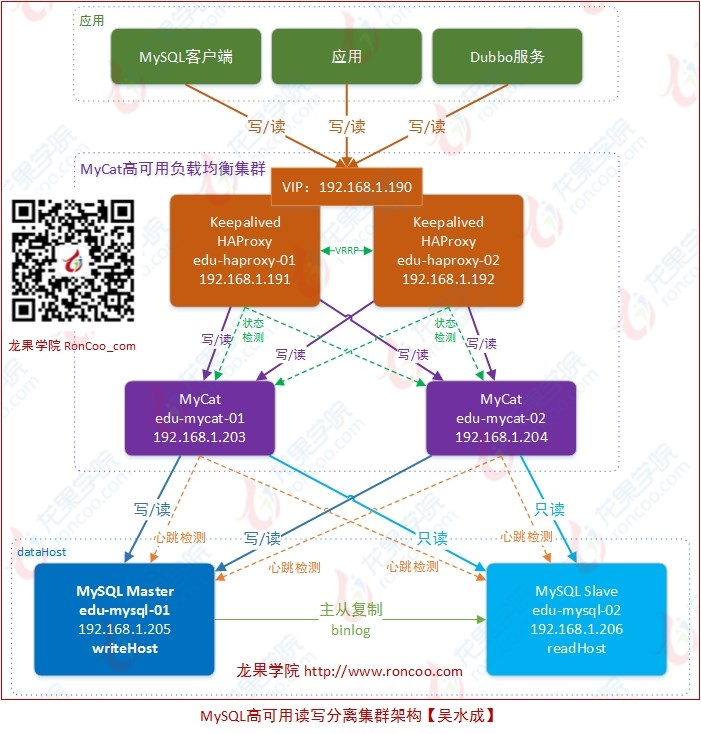
E:\MySQL-5.6.17-winx64\bin>mysql -uroot -proot -h192.168.106.126 -P3306



# MyCat 高 可用负载均衡集群的实现 （ HAProxy + Keepalived + MyCat ）







图解说明：

(1) HAProxy 实现了 MyCat 多节点的集群高可用和负载均衡，而 HAProxy 自身的高可用则可以通过

Keepalived 来实现。因此，HAProxy 主机上要同时安装 HAProxy 和 Keepalived，Keepalived 负责为该服务

器抢占 vip（虚拟 ip，图中的 192.168.1.190），抢占到 vip 后，对该主机的访问可以通过原来的 ip

（192.168.1.191）访问，也可以直接通过 vip（192.168.1.190）访问。

(2) Keepalived 抢占 vip 有优先级，在 keepalived.conf 配置中的 priority 属性决定。但是一般哪台主

机上的Keepalived服务先启动就会抢占到vip，即使是slave，只要先启动也能抢到（要注意避免Keepalived

的资源抢占问题）。

(3) HAProxy 负责将对 vip 的请求分发到 MyCat 集群节点上，起到负载均衡的作用。同时 HAProxy 也能检

测到 MyCat 是否存活，HAProxy 只会将请求转发到存活的 MyCat 上。

(4) 如果 Keepalived+HAProxy 高可用集群中的一台服务器宕机，集群中另外一台服务器上的 Keepalived

会立刻抢占 vip 并接管服务，此时抢占了 vip 的 HAProxy 节点可以继续提供服务。

(5) 如果一台 MyCat 服务器宕机，HAPorxy 转发请求时不会转发到宕机的 MyCat 上，所以 MyCat 依然可用。

综上：MyCat 的高可用及负载均衡由 HAProxy 来实现，而 HAProxy 的高可用，由 Keepalived 来实现。

### Keepalived 介绍 （官网：http://www.haproxy.org/ ）

Keepalived 是一种高性能的服务器高可用或热备解决方案，Keepalived 可以用来防止服务器单点故

障的发生，通过配合 Haproxy 可以实现 web 前端服务的高可用。

Keepalived 以 VRRP 协议为实现基础，用 VRRP 协议来实现高可用性(HA)。VRRP(Virtual Router

Redundancy Protocol)协议是用于实现路由器冗余的协议，VRRP 协议将两台或多台路由器设备虚拟成一个

设备，对外提供虚拟路由器 IP(一个或多个)，而在路由器组内部，如果实际拥有这个对外 IP 的路由器如果

工作正常的话就是 MASTER，或者是通过算法选举产生。MASTER 实现针对虚拟路由器 IP 的各种网络功能，

如 ARP 请求，ICMP，以及数据的转发等；其他设备不拥有该虚拟 IP，状态是 BACKUP，除了接收 MASTER 的

VRRP 状态通告信息外，不执行对外的网络功能。当主机失效时，BACKUP 将接管原先 MASTER 的网络功能。

VRRP 协议使用多播数据来传输 VRRP 数据，VRRP 数据使用特殊的虚拟源 MAC 地址发送数据而不是自身

网卡的 MAC 地址，VRRP 运行时只有 MASTER 路由器定时发送 VRRP 通告信息，表示 MASTER 工作正常以及虚

拟路由器 IP(组)，BACKUP 只接收 VRRP 数据，不发送数据，如果一定时间内没有接收到 MASTER 的通告信

息，各 BACKUP 将宣告自己成为 MASTER，发送通告信息，重新进行 MASTER 选举状态。

### **Keepalived 的 安装**

（ 192 .168.1.191 、 192 .168.1.192 ）

Keepalived （ http://www.keepalived.org/download.html ）

<http://www.keepalived.org/software/>

1、上传或下载 keepalived（keepalived-1.2.18.tar.gz）到 /usr/local/src 目录

2、解压安装

安装 keepalived 需要用到 openssl

# yum install gcc gcc-c++ openssl openssl-devel

# cd /usr/local/src

# tar -zxvf keepalived-1.2.18.tar.gz

# cd keepalived-1.2.18

# ./configure --prefix=/usr/local/keepalived

# make && make install

3、将 keepalived 安装成 Linux 系统服务：

因为没有使用 keepalived 的默认路径安装（默认是/usr/local）,安装完成之后，需要做一些工作

复制默认配置文件到默认路径

# mkdir /etc/keepalived

# cp /usr/local/keepalived/etc/keepalived/keepalived.conf /etc/keepalived/

复制 keepalived 服务脚本到默认的地址

# cp /usr/local/keepalived/etc/rc.d/init.d/keepalived /etc/init.d/

# cp /usr/local/keepalived/etc/sysconfig/keepalived /etc/sysconfig/

# ln -s /usr/local/keepalived/sbin/keepalived /usr/sbin/

# ln -s /usr/local/keepalived/sbin/keepalived /sbin/

设置 keepalived 服务开机启动

# chkconfig keepalived on

4、修改 Keepalived 配置文件

(1) MASTER 节点配置文件（192.168.1.191）

# vi /etc/keepalived/keepalived.conf

! Configuration File for keepalived

global\_defs {

## keepalived 自带的邮件提醒需要开启 sendmail 服务。建议用独立的监控或第三方 SMTP

router\_id edu-haproxy-01 ## 标识本节点的字条串，通常为 hostname

}

## keepalived 会定时执行脚本并对脚本执行的结果进行分析，动态调整 vrrp\_instance 的优先级。

## 如果脚本执行结果为 0，并且 weight 配置的值大于 0，则优先级相应的增加。

## 如果脚本执行结果非 0，并且 weight 配置的值小于 0，则优先级相应的减少。

## 其他情况，维持原本配置的优先级，即配置文件中 priority 对应的值。

vrrp\_script chk\_haproxy {

script "/etc/keepalived/haproxy\_check.sh" ## 检测 haproxy 状态的脚本路径

interval 2 ## 检测时间间隔

weight 2 ## 如果条件成立，权重+2

}

## 定义虚拟路由，VI\_1 为虚拟路由的标示符，自己定义名称

vrrp\_instance VI\_1 {

state BACKUP ## 默认主设备（priority 值大的）和备用设备（priority 值小的）都设置为 BACKUP，

## 由 priority 来控制同时启动情况下的默认主备，否则先启动的为主设备

interface eth1 ## 绑定虚拟 IP 的网络接口，与本机 IP 地址所在的网络接口相同，我的是 eth1

virtual\_router\_id 91 ## 虚拟路由的 ID 号，两个节点设置必须一样，可选 IP 最后一段使用,

## 相同的 VRID 为一个组，他将决定多播的 MAC 地址

priority 120 ## 节点优先级，值范围 0-254，MASTER 要比 BACKUP 高

nopreempt ## 主设备（priority 值大的）配置一定要加上 nopreempt，否则非抢占也不起作用

advert\_int 1 ## 组播信息发送间隔，两个节点设置必须一样，默认 1s

## 设置验证信息，两个节点必须一致

authentication {

auth\_type PASS

auth\_pass 1111 ## 真实生产，按需求对应该过来

}

## 将 track\_script 块加入 instance 配置块

track\_script {

chk\_haproxy ## 检查 HAProxy 服务是否存活

}

## 虚拟 IP 池, 两个节点设置必须一样

virtual\_ipaddress {

192.168.1.190 ## 虚拟 ip，可以定义多个，每行一个

}

}

(2)BACKUP 节点配置文件（192.168.1.192）:

# vi /etc/keepalived/keepalived.conf

! Configuration File for keepalived

global\_defs {

router\_id edu-haproxy-02

}

vrrp\_script chk\_haproxy {

script "/etc/keepalived/haproxy\_check.sh"

interval 2

weight 2

}

vrrp\_instance VI\_1 {

state BACKUP

interface eth1

virtual\_router\_id 91

priority 110

advert\_int 1

authentication {

auth\_type PASS

auth\_pass 1111

}

track\_script {

chk\_haproxy

}

virtual\_ipaddress {

192.168.1.190

}

}

特别 注意： 如 果 非 抢占模式不生效，在 在 d Keepalived 的 故障 节 点恢复后会再次导 抢占 vip ，从 而因 因 p vip 切换

而 闪断 带来 的风 险 （视频） 解说） 。 按以 上配置 ，配置了Keepalived 非 抢占模式 ， 配置及 注意点如下：

(1) 主设备、从设备中的 state 都设置为 BACKUP

(2) 主设备、从设备中都不要配置 mcast\_src\_ip （本机 IP 地址）

(3) 默认主设备（priority 值大的 Keepalived 节点）配置一定要加上 nopreempt，否则非抢占不起作用

(4) 防火墙配置允许组播（主、备两台设备上都需要配置，keepalived 使用 224.0.0.18 作为 Master 和

Backup 健康检查的通信 IP）

# iptables -I INPUT -i eth1 -d 224.0.0.0/8 -p vrrp -j ACCEPT

# iptables -I OUTPUT -o eth1 -d 224.0.0.0/8 -p vrrp -j ACCEPT

（eth1 为主机的网卡设备名称，生产环境服务器可以用独立网卡来处理组播和心跳检测等）

# service iptables save

重启防火墙：

# service iptables restart

5、编写 Haproxy 状态检测脚本 /etc/keepalived/haproxy\_check.sh (已在 keepalived.conf 中配置)

脚本要求：如果 haproxy 停止运行，尝试启动，如果无法启动则杀死本机的 keepalived 进程，keepalied

将虚拟 ip 绑定到 BACKUP 机器上。内容如下：

# mkdir -p /usr/local/keepalived/log

# vi /etc/keepalived/haproxy\_check.sh

#!/bin/bash

START\_HAPROXY="/etc/rc.d/init.d/haproxy start"

STOP\_HAPROXY="/etc/rc.d/init.d/haproxy stop"

LOG\_FILE="/usr/local/keepalived/log/haproxy-check.log"

HAPS=`ps -C haproxy --no-header |wc -l`

date "+%Y-%m-%d %H:%M:%S" >> $LOG\_FILE

echo "check haproxy status" >> $LOG\_FILE

if [ $HAPS -eq 0 ];then

echo $START\_HAPROXY >> $LOG\_FILE

$START\_HAPROXY >> $LOG\_FILE 2>&1

sleep 3

if [ `ps -C haproxy --no-header |wc -l` -eq 0 ];then

echo "start haproxy failed, killall keepalived" >> $LOG\_FILE

killall keepalived

fi

fi

保存后，给脚本赋执行权限：

# chmod +x /etc/keepalived/haproxy\_check.sh

6、启动 Keepalived

# service keepalived start

Starting keepalived: [ OK ]

Keepalived 服务管理命令：

停止：service keepalived stop

启动：service keepalived start

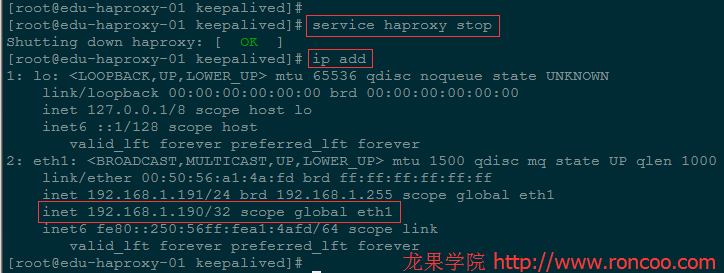
重启：service keepalived restart

查看状态：service keepalived status

八 、 Keepalived + + y Haproxy 的 高可用测试

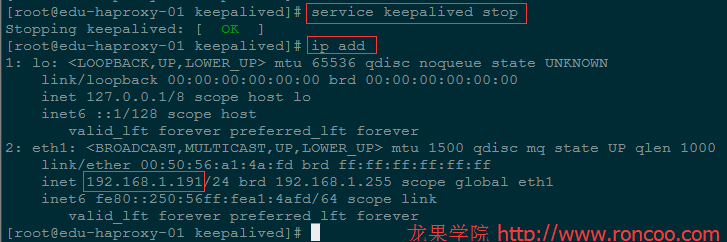
1、关闭 192.168.1.191 中的 Haproxy，Keepalived 会将它重新启动

# service haproxy stop

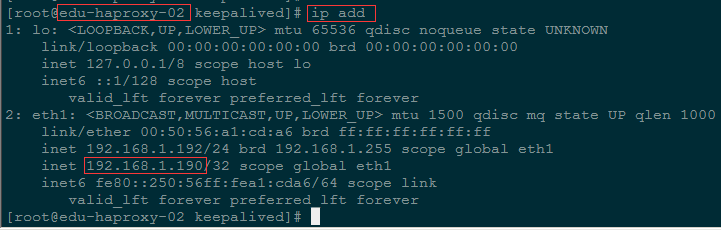


2、关闭 192.168.1.191 中的 Keepalived，VIP（192.168.1.190）会被 192.168.1.192 抢占

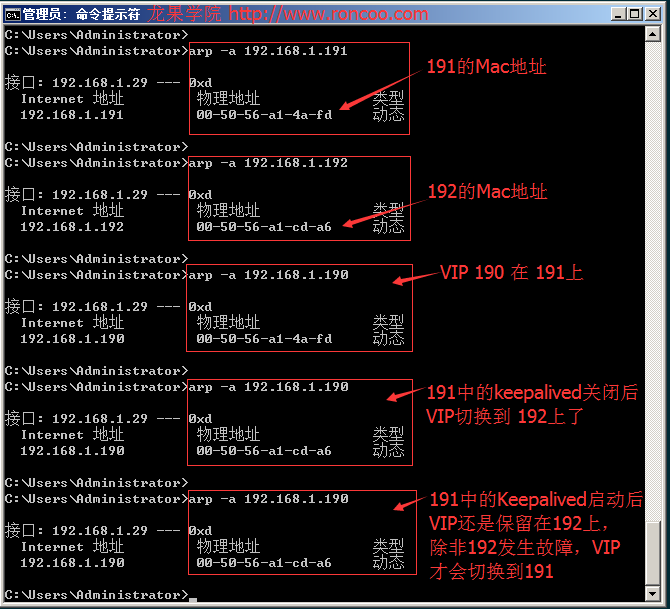
# service keepalived stop



Keepalived 停止后，192.168.1.191 节点的网络接口中的 VIP（192.168.1.190）将消失



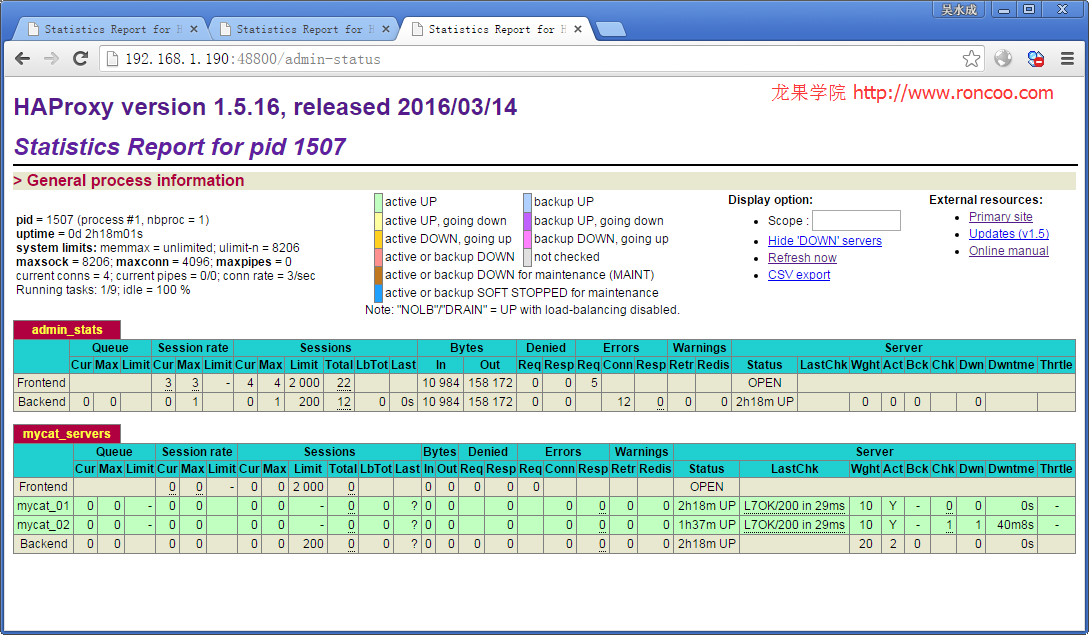
查看此时 VIP 对应的 MAC，Windows 下使用 CMD 命令查看：



说明此时 VIP 已经漂移到物理主机 192.168.1.192 上了

再通过 VIP 来访问 Haproxy 集群，访问到的也是 192.168.1.192

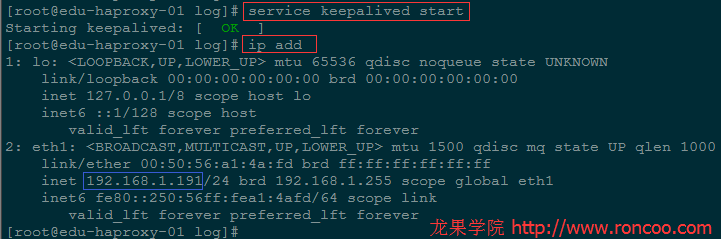
http://192.168.1.190:48800/admin-status



3、重新启动 192.168.1.191 中的 Keepalived，vip（192.168.1.190）保留在 192.168.1.192 主机上，不会

出现 191 启动抢占 vip 的情况。

# service keepalived start

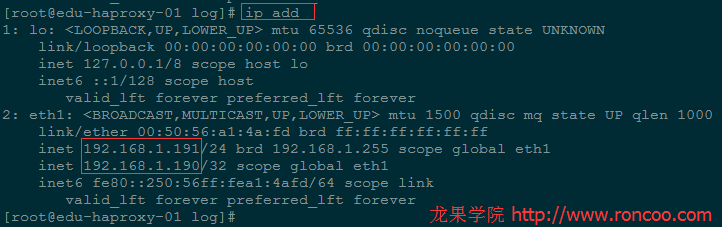


4、模拟抢占了 vip 的节点（192.168.1.192）中的 HAProxy 故障或启动失败。

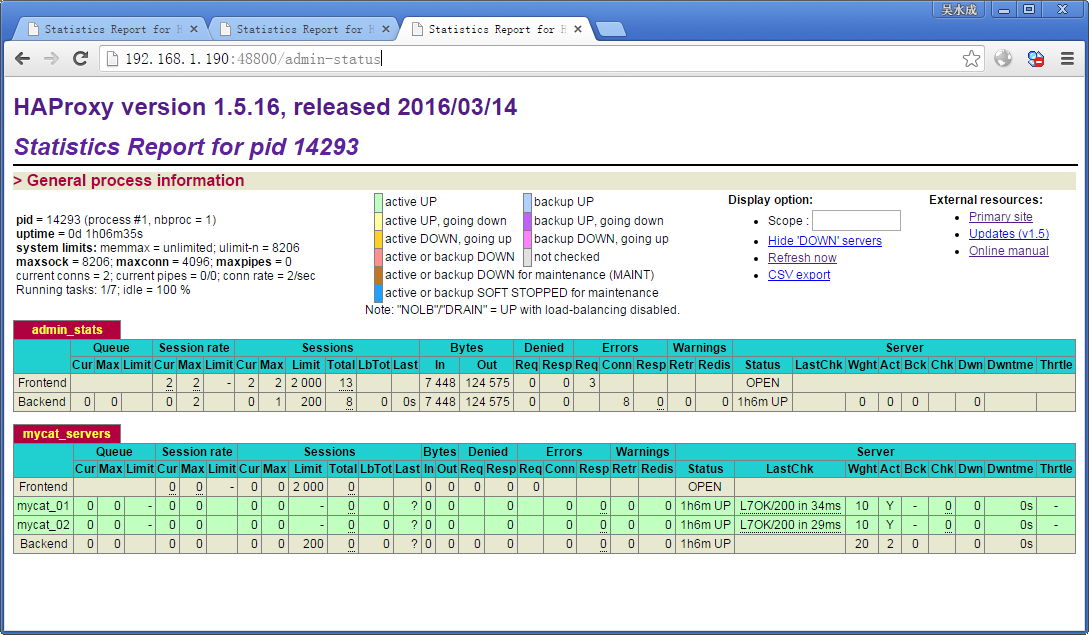
方式：把 192 节点中的 haproxy.cfg 文件重命名为 haproxy.cfg\_bak，并把 haproxy 服务进行 kill 掉，此

时keepalived会尝试去启动haproxy，会由于找不到配置文件而启动失败，此时就会进行haproxy\_check.sh

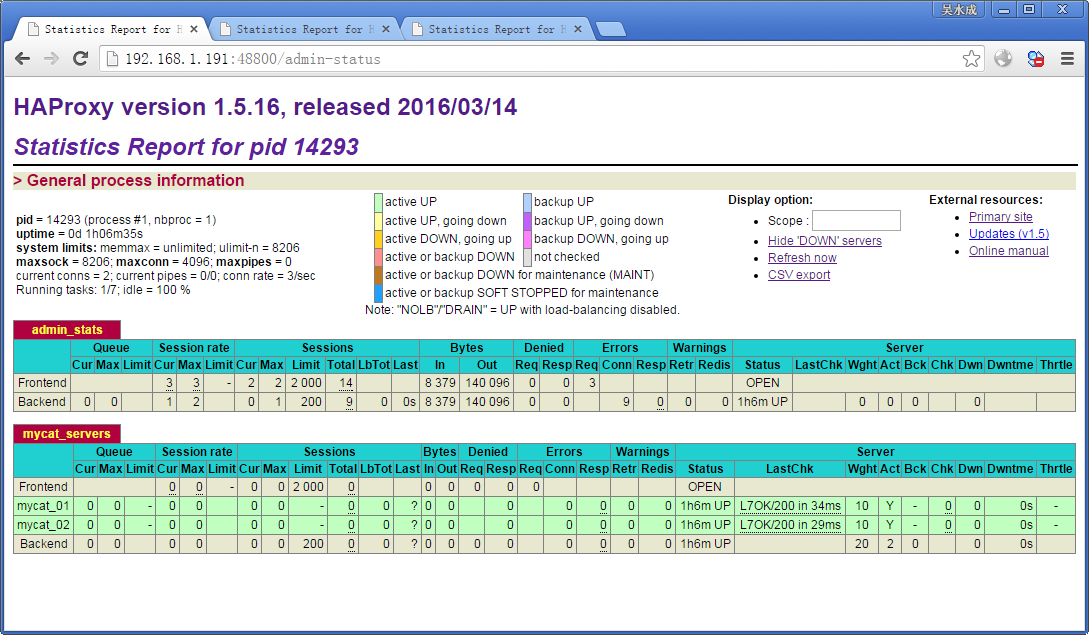
脚本中的 killall keepalived 命令，结束 keepalived 进行。随后就是 192.168.1.191 节点重新抢占 vip



http://192.168.1.190:48800/admin-status



http://192.168.1.191:48800/admin-status



九 、 通过 过 p vip 访问 数据库 、验证 p vip 切换后 的数据库访问



# 分库分表是什么、什么情况下需要分表分库、分表分库后，如何做联合查找

1、什么是分库分表？

顾名思义，即把存于一个库的数据分散到多个库中，把存于一个表的数据分散到多个表中。

2、什么情况下需要分库分表？

当一个数据库被创建之后，随着时间的推移和业务量的增加，数据库中的表以及表中的数据量都会越来越多，就有可能会出现两种弊端：

（1）数据库的存储资源是有限的，其负载能力也是有限的，数据的大量积累肯定会导致其处理数据的能力下降；

（2）数据量越多，那么对数据的增删改查等操作的开销也会越来越大，所以，当出现如上两种情况，分库分表势在必行。

3、分库分表的方式

（1）垂直切分

适用场景：如果是因为表的个数多而让数据多，可以按照功能划分，把联系密切的表切分出来放在同一个库中（分库）；

如果表的字段太多，可以以列为出发点，将字段进行拆分（分表）；

（2）水平切分

适用场景：如果是因为表中的数据量过于庞大，则可以采用水平切分，按照某种约定好的规则将数据切分到不同的数据库中；

必须要根据当前数据库的情况做出合适的选择，也可以将两种情况结合在一起。

4、如何联合查找？

分库分表的结果会使数据分散，不好查询，主要有两种查询方式：

（1）、分步查：先查找主表，然后得到关联表的id，再发起请求得到关联数据；

（2）、联合查：同时发起多个查询请求，然后将所有的结果集合起来。

# Mycat教程---数据库的分库分表

<https://www.cnblogs.com/mracale/p/8989248.html>