

# Java 课程系列之 Mycat

尚硅谷 Java 研究院

版本：V1.6

## 第一章 入门概述

### 1.1 是什么

Mycat 是数据库中间件。

#### 1、数据库中间件

中间件：是一类连接软件组件和应用的计算机软件，以便于软件各部件之间的沟通。

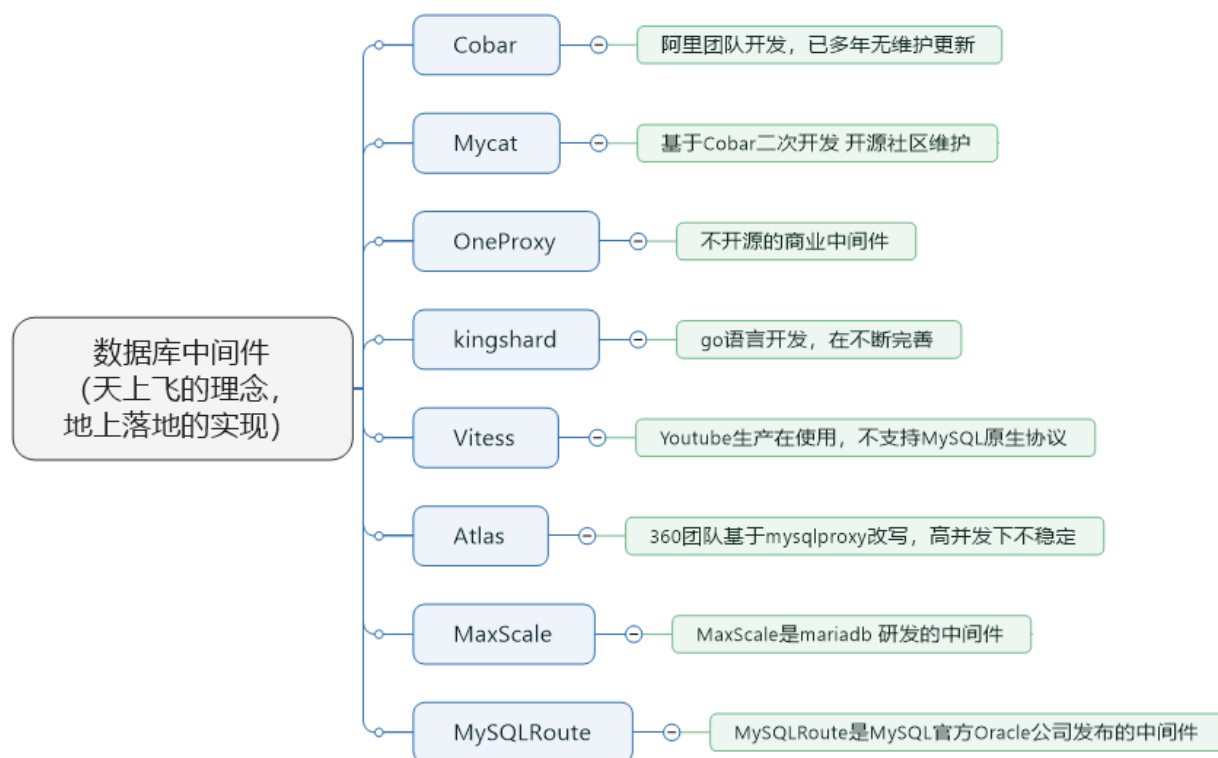
例子：Tomcat，web中间件。

数据库中间件：连接java应用程序和数据库

#### 2、为什么要用Mycat?

- ① Java与数据库紧耦合。
- ② 高访问量高并发对数据库的压力。
- ③ 读写请求数据不一致

#### 3、数据库中间件对比



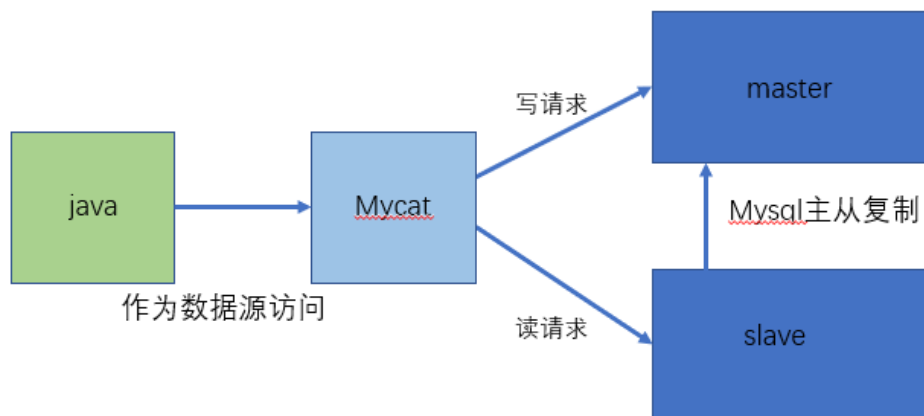
- ① Cobar属于阿里B2B事业群, 始于2008年, 在阿里服役3年多, 接管3000+个MySQL数据库的schema, 集群日处理在线SQL请求50亿次以上。由于Cobar发起人的离职, Cobar停止维护。
- ② Mycat是开源社区在阿里cobar基础上进行二次开发, 解决了cobar存在的问题, 并且加入了许多新的功能在其中。青出于蓝而胜于蓝。
- ③ OneProxy基于MySQL官方的proxy思想利用c进行开发的, OneProxy是一款商业收费的中间件。舍弃了一些功能, 专注在性能和稳定性上。
- ④ kingshard由小团队用go语言开发, 还需要发展, 需要不断完善。
- ⑤ Vitess是Youtube生产在使用, 架构很复杂。不支持MySQL原生协议, 使用需要大量改造成本。
- ⑥ Atlas是360团队基于mysql proxy改写, 功能还需完善, 高并发下不稳定。
- ⑦ MaxScale是mariadb (MySQL原作者维护的一个版本) 研发的中间件
- ⑧ MySQLRoute是MySQL官方Oracle公司发布的中间件

### 3、Mycat的官网

<http://www.mycat.io/>

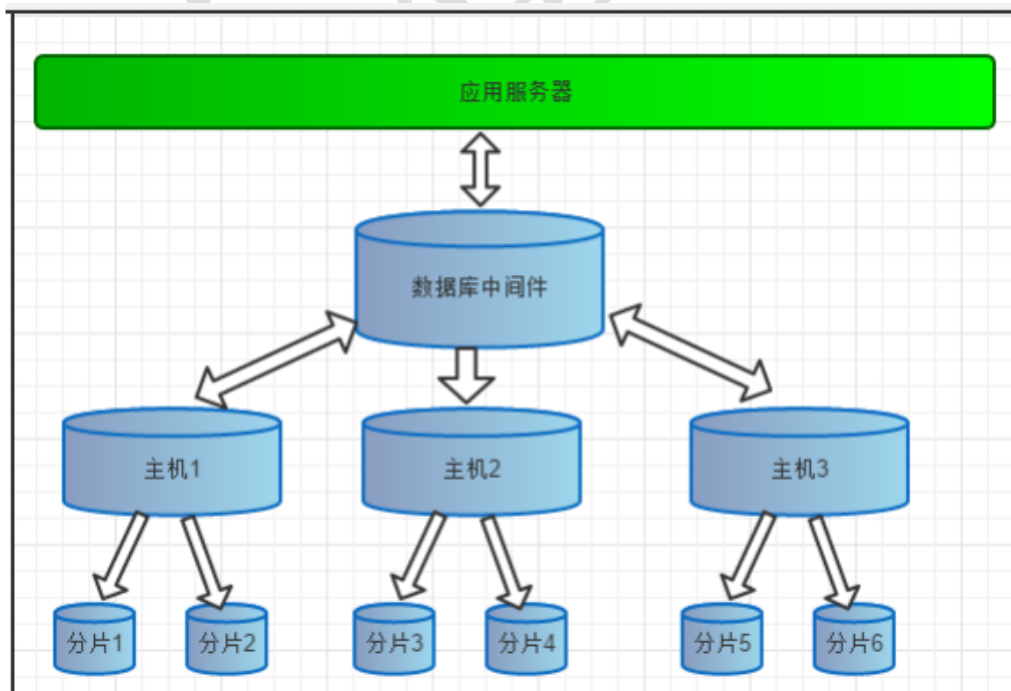
## 1.2 干什么

### 1、读写分离

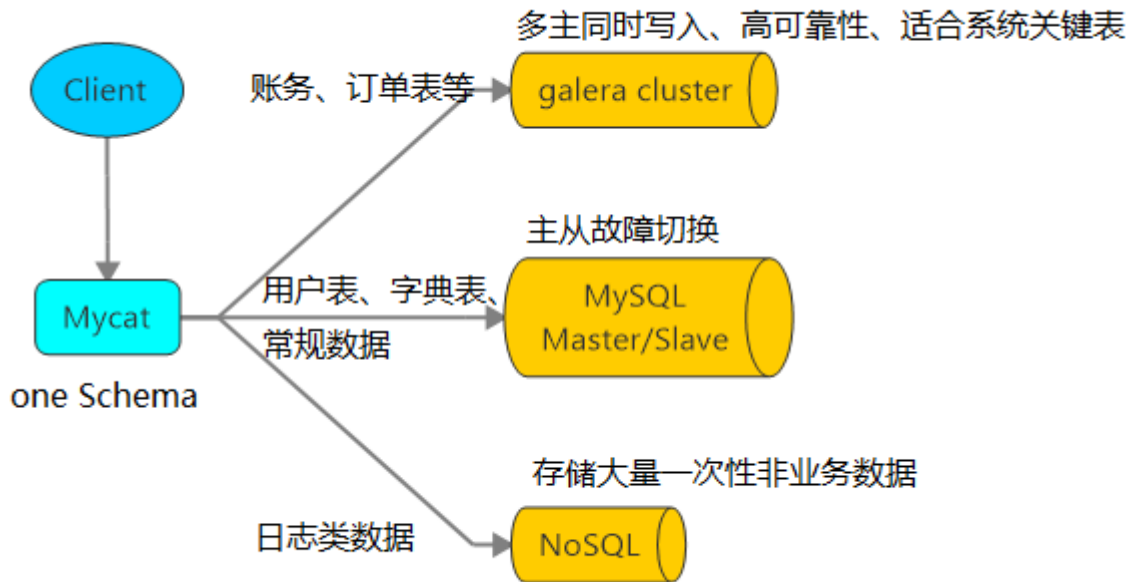


### 2、数据分片

垂直拆分（分库）、水平拆分（分表）、垂直+水平拆分（分库分表）

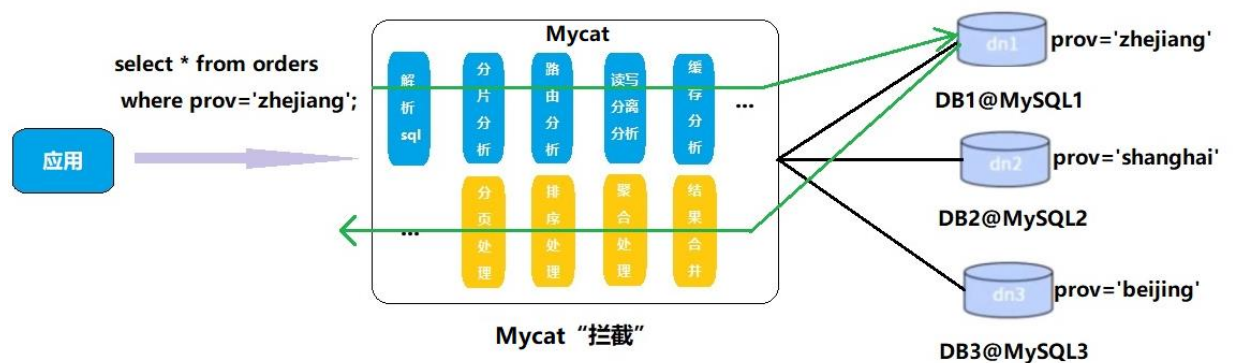


### 3、多数据源整合



## 1.3 原理

Mycat 的原理中最重要的一个动词是“拦截”，它拦截了用户发送过来的 SQL 语句，首先对 SQL 语句做了一些特定的分析：如分片分析、路由分析、读写分离分析、缓存分析等，然后将此 SQL 发往后端的真实数据库，并将返回的结果做适当的处理，最终再返回给用户。



这种方式把数据库的分布式从代码中解耦出来，程序员察觉不出来后台使用 Mycat 还是 MySQL。

## 第二章 安装启动

### 2.1 安装

#### 1、解压后即可使用

解压缩文件拷贝到 linux 下 /usr/local/

#### 2、三个配置文件

①schema.xml: 定义逻辑库, 表、分片节点等内容

②rule.xml: 定义分片规则

③server.xml: 定义用户以及系统相关变量, 如端口等

### 2.2 启动

#### 1、修改配置文件server.xml

修改用户信息, 与MySQL区分, 如下:

```
...  
<user name="mycat">  
    <property name="password">123456</property>  
    <property name="schemas">TESTDB</property>  
</user>  
...
```

```
<user name="mycat">  
    <property name="password">123456</property>  
    <property name="schemas">TESTDB</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>
```

#### 2、修改配置文件 schema.xml

更多 Java -大数据 -前端 -python 人工智能资料下载, 可百度访问: 尚硅谷官网

删除<schema>标签间的表信息, <dataNode>标签只留一个, <dataHost>标签只留一个, <writeHost>  
<readHost>只留一对

```
<?xml version="1.0"?>
<!DOCTYPE mycat:schema SYSTEM "schema.dtd">
<mycat:schema xmlns:mycat="http://io.mycat/">

    <schema name="TESTDB" checkSQLschema="false" sqlMaxLimit="100" dataNode="dn1">
    </schema>
    <dataNode name="dn1" dataHost="host1" database="testdb" />
    <dataHost name="host1" maxCon="1000" minCon="10" balance="0"
        writeType="0" dbType="mysql" dbDriver="native" switchType="1"
slaveThreshold="100">
        <heartbeat>select user()</heartbeat>
        <!-- can have multi write hosts -->
        <writeHost host="hostM1" url="192.168.140.128:3306" user="root"
            password="123123">
        <!-- can have multi read hosts -->
        <readHost host="hostS1" url="192.168.140.127:3306" user="root"
password="123123" />
        </writeHost>
    </dataHost>
</mycat:schema>
```

```
<?xml version="1.0"?>
<!DOCTYPE mycat:schema SYSTEM "schema.dtd">
<mycat:schema xmlns:mycat="http://io.mycat/">

    <schema name="TESTDB" checkSQLschema="false" sqlMaxLimit="100" dataNode="dn1">
    </schema>
    <dataNode name="dn1" dataHost="host1" database="testdb" />
    <dataHost name="host1" maxCon="1000" minCon="10" balance="0"
        writeType="0" dbType="mysql" dbDriver="native" switchType="1" slaveThreshold="100">
        <heartbeat>select user()</heartbeat>
        <!-- can have multi write hosts -->
        <writeHost host="hostM1" url="192.168.140.128:3306" user="root"
            password="123123">
        <!-- can have multi read hosts -->
        <readHost host="hostS1" url="192.168.140.127:3306" user="root" password="123123" />
        </writeHost>
    </dataHost>
</mycat:schema>
```

### 3、验证数据库访问情况

更多 Java -大数据 -前端 -python 人工智能资料下载, 可百度访问: 尚硅谷官网

Mycat 作为数据库中间件要和数据库部署在不同机器上，所以要验证远程访问情况。

```
mysql -uroot -p123123 -h 192.168.140.128 -P 3306
mysql -uroot -p123123 -h 192.168.140.127 -P 3306

#如远程访问报错，请建对应用户
grant all privileges on *.* to root@'缺少的host' identified by '123123';
```

#### 4、启动程序

①控制台启动：去 mycat/bin 目录下执行 `./mycat console`

②后台启动：去 mycat/bin 目录下 `./mycat start`

为了能第一时间看到启动日志，方便定位问题，我们选择①控制台启动。

#### 5、启动时可能出现报错

如果操作系统是 CentOS6.8，可能会出现域名解析失败错误，如下图

```
root@jack bin)# ./mycat console
Running Mycat-server...
wrapper | --> Wrapper Started as Console
wrapper | Launching a JVM...
wrapper | JVM exited while loading the application.
vm 1 | 错误: 代理抛出异常错误: java.net.MalformedURLException: Local host name unknown: java.net.UnknownHostException: jack.atguigu: jack.atguigu: 域名解析暂时失败
wrapper | Launching a JVM...
```

可以按照以下步骤解决

① 用 vim 修改 `/etc/hosts` 文件，在 `127.0.0.1` 后面增加你的机器名

```
127.0.0.1    localhost localhost.localdomain localhost4 localhost4.localdomain4 jack.atguigu
::1         localhost localhost.localdomain localhost6 localhost6.localdomain6
```

② 修改后重新启动网络服务

```
[root@jack ~]# service network restart
正在关闭接口 eth0: 设备状态: 3 (断开连接)
关闭环回接口: [确定]
弹出环回接口: [确定]
弹出界面 eth0: 活跃连接状态: 激活中 [确定]
活跃连接路径: /org/freedesktop/NetworkManager/ActiveConnection/1
状态: 激活的
连接被激活
[确定]
[root@jack ~]#
```

## 2.3 登录

### 1、登录后台管理窗口

此登录方式用于管理维护 Mycat

```
mysql -umycat -p123456 -P 9066 -h 192.168.140.128
```

#常用命令如下:

```
show database
```

```
mysql> show database;
+-----+
| DATABASE |
+-----+
| TESTDB   |
+-----+
1 row in set (0.01 sec)
```

```
show @@help
```

```
mysql> show @@help;
+-----+-----+
| STATEMENT | DESCRIPTION |
+-----+-----+
| show @@time.current | Report current timestamp |
| show @@time.startup | Report startup timestamp |
| show @@version | Report Mycat Server version |
| show @@server | Report server status |
| show @@threadpool | Report threadPool status |
| show @@database | Report databases |
| show @@datanode | Report dataNodes |
| show @@datanode where schema = ? | Report dataNodes |
| show @@datasource | Report dataSources |
| show @@datasource where dataNode = ? | Report dataSources |
| show @@datasource.synstatus | Report datasource data synchronous |
| show @@datasource.syndetail where name=? | Report datasource data synchronous detail |
| show @@datasource.cluster | Report datasource galary cluster variables |
| show @@processor | Report processor status |
| show @@command | Report commands status |
| show @@connection | Report connection status |
| show @@cache | Report system cache usage |
| show @@backend | Report backend connection status |
| show @@session | Report front session details |
| show @@connection.sql | Report connection sql |
| show @@sql.execute | Report execute status |
| show @@sql.detail where id = ? | Report execute detail status |
| show @@sql | Report SQL list |
| show @@sql.high | Report Hight Frequency SQL |
| show @@sql.slow | Report slow SQL |
| show @@sql.resultset | Report BIG RESULTSET SQL |
```

### 2、登录数据窗口

此登录方式用于通过 Mycat 查询数据, 我们选择这种方式访问 Mycat

```
mysql -umycat -p123456 -P 8066 -h 192.168.140.128
```

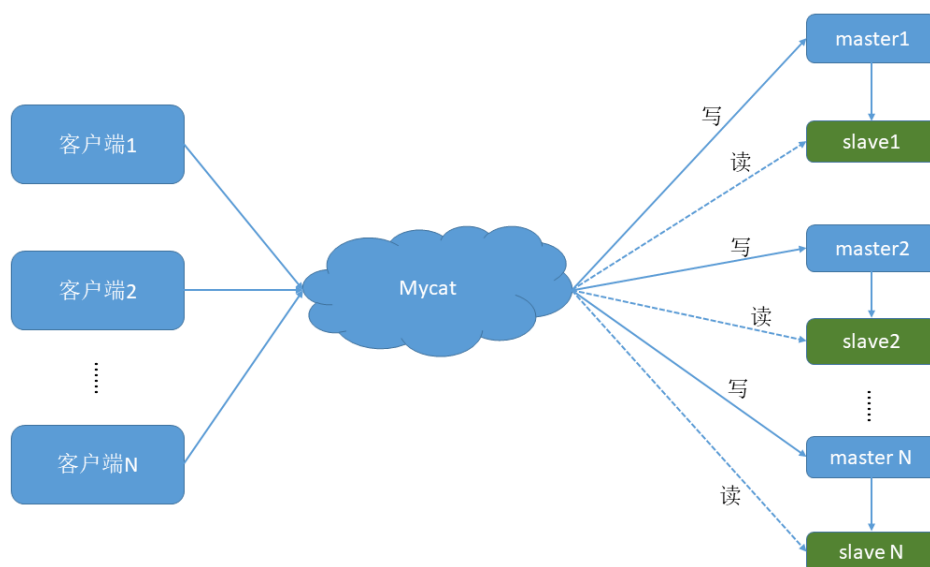


## 第三章 搭建读写分离

我们通过 Mycat 和 MySQL 的主从复制配合搭建数据库的读写分离，实现 MySQL 的高可用性。  
我们将搭建：一主一从、双主双从两种读写分离模式。

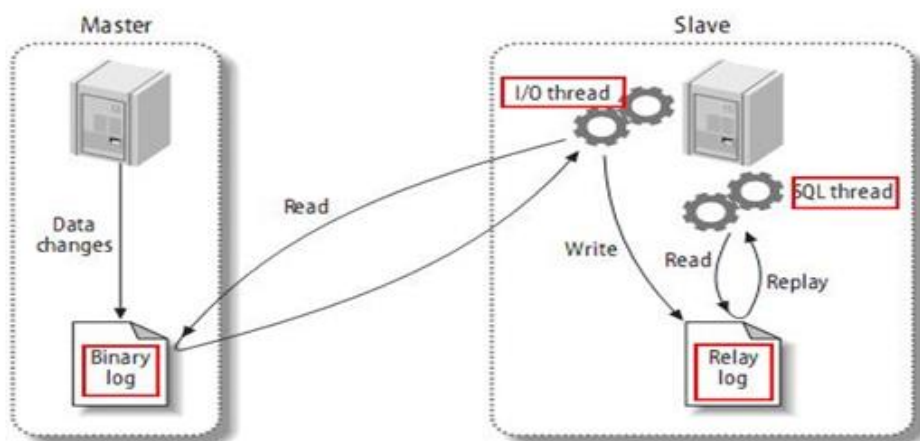
### 3.1 搭建一主一从

一个主机用于处理所有写请求，一台从机负责所有读请求，架构图如下



#### 1、搭建 MySQL 数据库主从复制

##### ① MySQL 主从复制原理



## ② 主机配置(host79)

修改配置文件: vim /etc/my.cnf

#主服务器唯一ID

**server-id=1**

#启用二进制日志

**log-bin=mysql-bin**

# 设置不要复制的数据库(可设置多个)

binlog-ignore-db=mysql

binlog-ignore-db=information\_schema

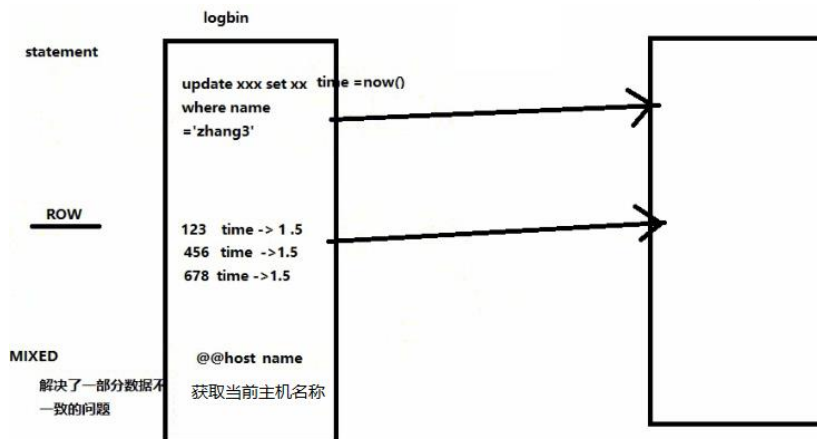
#设置需要复制的数据库

**binlog-do-db=需要复制的主数据库名字**

#设置logbin格式

**binlog\_format=STATEMENT**

binlog 日志三种格式



## ③ 从机配置(host80)

修改配置文件: vim /etc/my.cnf

#从服务器唯一ID

**server-id=2**

#启用中继日志

**relay-log=mysql-relay**

## ④ 主机、从机重启 MySQL 服务

## ⑤ 主机从机都关闭防火墙

## ⑥ 在主机上建立帐户并授权 slave

#在主机MySQL里执行授权命令

**GRANT REPLICATION SLAVE ON \*.\* TO 'slave'@'%' IDENTIFIED BY '123123';**

#查询master的状态

show master status;

```
mysql> show master status;
+-----+-----+-----+-----+-----+
| File           | Position | Binlog_Do_DB | Binlog_Ignore_DB | Executed_Gtid_Set |
+-----+-----+-----+-----+-----+
| mysql-bin.000007 | 154      | testdb       | mysql             |                    |
+-----+-----+-----+-----+-----+
1 row in set (0.01 sec)
```

#记录下File和Position的值

#执行完此步骤后不要再操作主服务器MySQL，防止主服务器状态值变化

#### ⑦ 在从机上配置需要复制的主机

#复制主机的命令

CHANGE MASTER TO MASTER\_HOST='主机的IP地址';

MASTER\_USER='slave';

MASTER\_PASSWORD='123123';

MASTER\_LOG\_FILE='mysql-bin.具体数字',MASTER\_LOG\_POS=具体值;

```
mysql> CHANGE MASTER TO MASTER_HOST='192.168.140.128',
-> MASTER_USER='slave',
-> MASTER_PASSWORD='123123',
-> MASTER_LOG_FILE='mysql-bin.000007',MASTER_LOG_POS=154;
Query OK, 0 rows affected, 2 warnings (0.00 sec)
```

#启动从服务器复制功能

start slave;

#查看从服务器状态

show slave status\G;

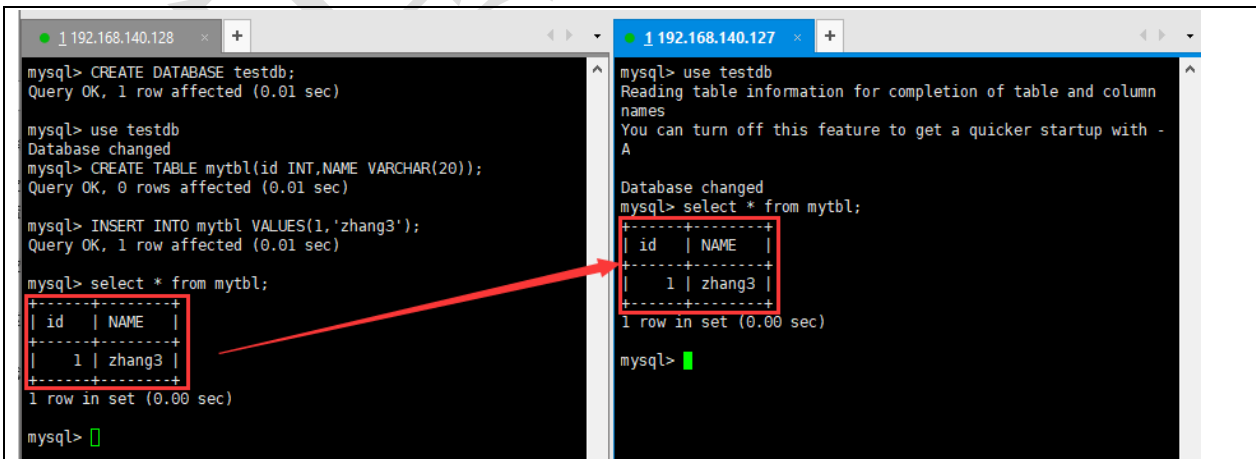
```
mysql> show slave status\G;
***** 1. row *****
      Slave_IO_State: Waiting for master to send event
      Master_Host: 192.168.140.128
      Master_User: slave
      Master_Port: 3306
      Connect_Retry: 60
      Master_Log_File: mysql-bin.000007
      Read_Master_Log_Pos: 154
      Relay_Log_File: mysql-relay.000002
      Relay_Log_Pos: 320
      Relay_Master_Log_File: mysql-bin.000007
      Slave_IO_Running: Yes
      Slave_SQL_Running: Yes
      Replicate_Do_DB:
      Replicate_Ignore_DB:
      Replicate_Do_Table:
      Replicate_Ignore_Table:
      Replicate_Wild_Do_Table:
      Replicate_Wild_Ignore_Table:
      Last_Errno: 0
      Last_Error:
      Skip_Counter: 0
      Exec_Master_Log_Pos: 154
      Relay_Log_Space: 523
      Until_Condition: None
      Until_Log_File:
      Until_Log_Pos: 0
      Master_SSL_Allowed: No
      Master_SSL_CA_File:
      Master_SSL_CA_Path:
```

#下面两个参数都是Yes，则说明主从配置成功！

# Slave\_IO\_Running: Yes

# Slave\_SQL\_Running: Yes

### ⑧ 主机新建库、新建表、insert 记录，从机复制



```
mysql> CREATE DATABASE testdb;
Query OK, 1 row affected (0.01 sec)

mysql> use testdb
Database changed
mysql> CREATE TABLE mytbl(id INT,NAME VARCHAR(20));
Query OK, 0 rows affected (0.01 sec)

mysql> INSERT INTO mytbl VALUES(1,'zhang3');
Query OK, 1 row affected (0.01 sec)

mysql> select * from mytbl;
+----+-----+
| id | NAME |
+----+-----+
| 1  | zhang3 |
+----+-----+
1 row in set (0.00 sec)

mysql>
```

```
mysql> use testdb
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> select * from mytbl;
+----+-----+
| id | NAME |
+----+-----+
| 1  | zhang3 |
+----+-----+
1 row in set (0.00 sec)

mysql>
```

## ⑨ 如何停止从服务复制功能

```
stop slave;
```

## ⑩ 如何重新配置主从

```
stop slave;  
  
reset master;
```

## 2、修改 Mycat 的配置文件 schema.xml

之前的配置已分配了读写主机，是否已实现读写分离？

验证读写分离

- (1) 在写主机插入: `insert into mytbl values (1, @@hostname);`  
主从主机数据不一致了
- (2) 在Mycat里查询: `select * from mytbl;`

修改<dataHost>的balance属性，通过此属性配置读写分离的类型

负载均衡类型，目前的取值有4种：

- (1) `balance="0"`，不开启读写分离机制，所有读操作都发送到当前可用的 `writeHost` 上。
- (2) `balance="1"`，全部的 `readHost` 与 `stand by writeHost` 参与 `select` 语句的负载均衡，简单的说，当双主双从模式(M1->S1, M2->S2, 并且 M1 与 M2 互为备用)，正常情况下，M2,S1,S2 都参与 `select` 语句的负载均衡。
- (3) `balance="2"`，所有读操作都随机的在 `writeHost`、`readhost` 上分发。
- (4) `balance="3"`，所有读请求随机的分发到 `readhost` 执行，`writerHost` 不负担读压力

为了能看到读写分离的效果，把balance设置成2，会在两个主机间切换查询

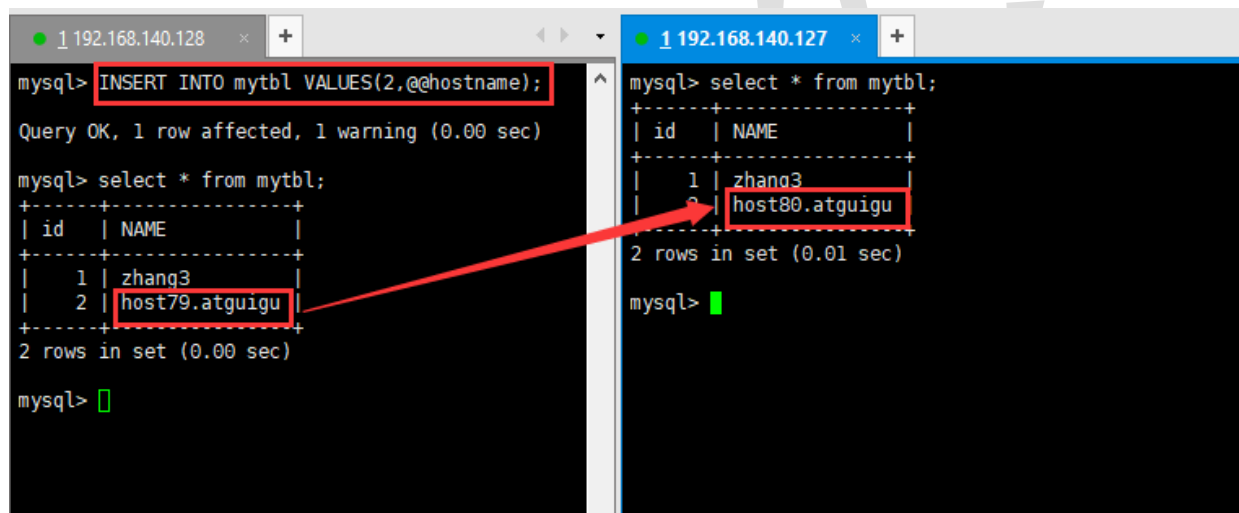
```
...  
<dataHost name="host1" maxCon="1000" minCon="10" balance="2"  
                writeType="0" dbType="mysql" dbDriver="native" switchType="1"  
slaveThreshold="100">  
...  
...
```

```
<dataNode name="dn1" dataHost="host1" database="testdb" />
<dataHost name="host1" maxCon="1000" minCon="10" balance="2"
  writeType="0" dbType="mysql" dbDriver="native" switchType="1" slaveThreshold="100">
```

### 3、启动 Mycat

### 4、验证读写分离

# (1) 在写主机数据库表mytbl中插入带系统变量数据，造成主从数据不一致  
INSERT INTO mytbl VALUES(2,@hostname);



Left terminal (192.168.140.128):

```
mysql> INSERT INTO mytbl VALUES(2,@hostname);
Query OK, 1 row affected, 1 warning (0.00 sec)

mysql> select * from mytbl;
+----+-----+
| id | NAME          |
+----+-----+
| 1  | zhang3        |
| 2  | host79.atguigu |
+----+-----+
2 rows in set (0.00 sec)

mysql>
```

Right terminal (192.168.140.127):

```
mysql> select * from mytbl;
+----+-----+
| id | NAME          |
+----+-----+
| 1  | zhang3        |
| 2  | host80.atguigu |
+----+-----+
2 rows in set (0.01 sec)

mysql>
```

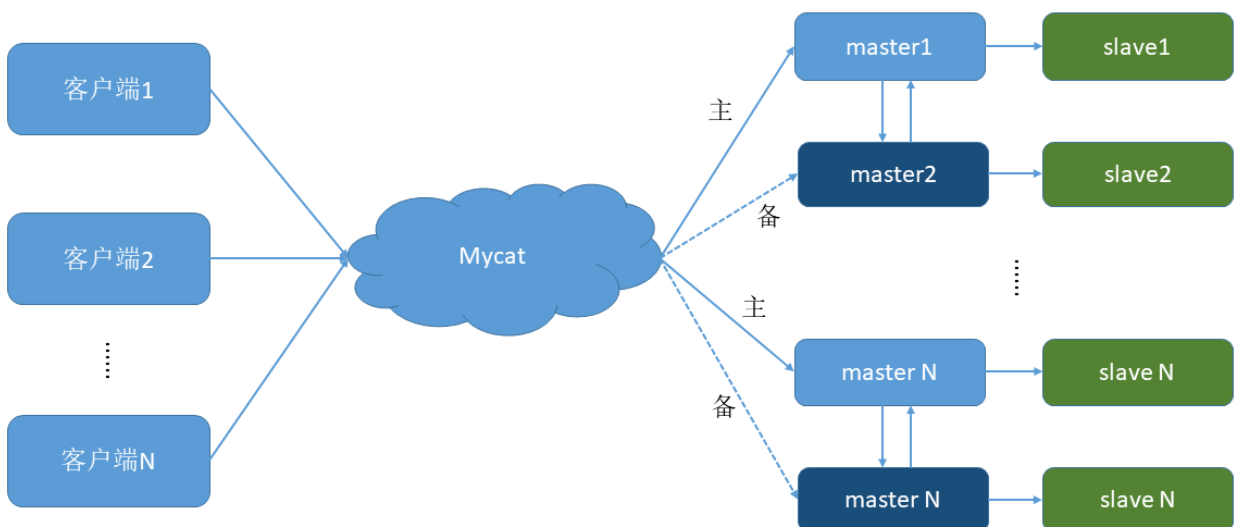
# (2) 在Mycat里查询mytbl表,可以看到查询语句在主从两个主机间切换

```
1 mycat x 2 conf x 3 bin x +
mysql> select * from mytbl;
+-----+-----+
| id | NAME |
+-----+-----+
| 1 | zhang3 |
| 2 | host79.atguigu |
+-----+-----+
2 rows in set (0.00 sec)

mysql>
mysql> select * from mytbl;
+-----+-----+
| id | NAME |
+-----+-----+
| 1 | zhang3 |
| 2 | host80.atguigu |
+-----+-----+
2 rows in set (0.00 sec)
```

### 3.2 搭建双主双从

一个主机 m1 用于处理所有写请求，它的从机 s1 和另一台主机 m2 还有它的从机 s2 负责所有读请求。当 m1 主机宕机后，m2 主机负责写请求，m1、m2 互为备机。架构图如下



编号	角色	IP 地址	机器名
1	Master1	192.168.140.128	host79.atguigu
2	Slave1	192.168.140.127	host80.atguigu
3	Master2	192.168.140.126	host81.atguigu
4	Slave2	192.168.140.125	host82.atguigu

## 1、搭建 MySQL 数据库主从复制（双主双从）

### ① 双主机配置

#### Master1配置

修改配置文件：vim /etc/my.cnf

#主服务器唯一ID

**server-id=1**

#启用二进制日志

**log-bin=mysql-bin**

# 设置不要复制的数据库(可设置多个)

binlog-ignore-db=mysql

binlog-ignore-db=information\_schema

#设置需要复制的数据库

**binlog-do-db=需要复制的主数据库名字**

#设置logbin格式

binlog\_format=STATEMENT

# 在作为从数据库的时候，有写入操作也要更新二进制日志文件

**log-slave-updates**

#表示自增长字段每次递增的量，指自增字段的起始值，其默认值是1，取值范围是1 .. 65535

**auto-increment-increment=2**

# 表示自增长字段从哪个数开始，指字段一次递增多少，他的取值范围是1 .. 65535

**auto-increment-offset=1**

#### Master2配置

修改配置文件：vim /etc/my.cnf

#主服务器唯一ID

**server-id=3**

#启用二进制日志

**log-bin=mysql-bin**

# 设置不要复制的数据库(可设置多个)



```
binlog-ignore-db=mysql
binlog-ignore-db=information_schema
#设置需要复制的数据库
binlog-do-db=需要复制的主数据库名字
#设置logbin格式
binlog_format=STATEMENT
# 在作为从数据库的时候，有写入操作也要更新二进制日志文件
log-slave-updates
#表示自增长字段每次递增的量，指自增字段的起始值，其默认值是1，取值范围是1 .. 65535
auto-increment-increment=2
# 表示自增长字段从哪个数开始，指字段一次递增多少，他的取值范围是1 .. 65535
auto-increment-offset=2
```

## ② 双从机配置

### Slave1配置

```
修改配置文件：vim /etc/my.cnf
#从服务器唯一ID
server-id=2
#启用中继日志
relay-log=mysql-relay
```

### Slave2配置

```
修改配置文件：vim /etc/my.cnf
#从服务器唯一ID
server-id=4
#启用中继日志
relay-log=mysql-relay
```

- ③ 双主机、双从机重启 mysql 服务
- ④ 主机从机都关闭防火墙
- ⑤ 在两台主机上建立帐户并授权 slave

```
#在主机MySQL里执行授权命令
GRANT REPLICATION SLAVE ON *.* TO 'slave'@'%' IDENTIFIED BY '123123';
#查询Master1的状态
show master status;
```

```
mysql> show master status;
+-----+-----+-----+-----+-----+
| File           | Position | Binlog_Do_DB | Binlog_Ignore_DB | Executed_Gtid_Set |
+-----+-----+-----+-----+-----+
| mysql-bin.000008 |      154 | testdb       | mysql             |                    |
+-----+-----+-----+-----+-----+
1 row in set (0.00 sec)
```

#查询Master2的状态

show master status;

```
mysql> show master status;
+-----+-----+-----+-----+-----+
| File           | Position | Binlog_Do_DB | Binlog_Ignore_DB | Executed_Gtid_Set |
+-----+-----+-----+-----+-----+
| mysql-bin.000001 |      154 | testdb       | mysql             |                    |
+-----+-----+-----+-----+-----+
1 row in set (0.01 sec)
```

#分别记录下File和Position的值

#执行完此步骤后不要再操作主服务器MYSQL，防止主服务器状态值变化

⑥ 在从机上配置需要复制的主机

Slava1 复制 Master1, Slava2 复制 Master2

#复制主机的命令

```
CHANGE MASTER TO MASTER_HOST='主机的IP地址',
MASTER_USER='slave',
MASTER_PASSWORD='123123',
MASTER_LOG_FILE='mysql-bin.具体数字',MASTER_LOG_POS=具体值;
```

Slava1的复制命令

```
mysql> CHANGE MASTER TO MASTER_HOST='192.168.140.128',
-> MASTER_USER='slave',
-> MASTER_PASSWORD='123123',
-> MASTER_LOG_FILE='mysql-bin.000008',MASTER_LOG_POS=154;
Query OK, 0 rows affected, 2 warnings (0.01 sec)
```

Slava2的复制命令

```
mysql> CHANGE MASTER TO MASTER_HOST='192.168.140.126',
-> MASTER_USER='slave',
-> MASTER_PASSWORD='123123',
-> MASTER_LOG_FILE='mysql-bin.000001',MASTER_LOG_POS=154;
Query OK, 0 rows affected, 2 warnings (0.02 sec)
```

#启动两台从服务器复制功能

start slave;

#查看从服务器状态

show slave status\G;

#Slave1的复制Master1

```
mysql> show slave status\G;
***** 1. row *****
      Slave_IO_State: Waiting for master to send event
      Master_Host: 192.168.140.128
      Master_User: slave
      Master_Port: 3306
      Connect_Retry: 60
      Master_Log_File: mysql-bin.000008
      Read_Master_Log_Pos: 154
      Relay_Log_File: mysql-relay.000002
      Relay_Log_Pos: 320
      Relay_Master_Log_File: mysql-bin.000008
      Slave_IO_Running: Yes
      Slave_SQL_Running: Yes
      Replicate_Do_DB:
      Replicate_Ignore_DB:
      Replicate_Do_Table:
      Replicate_Ignore_Table:
      Replicate_Wild_Do_Table:
```

#Slave2的复制Master2

```
mysql> show slave status\G;
***** 1. row *****
      Slave_IO_State: Waiting for master to send event
      Master_Host: 192.168.140.126
      Master_User: slave
      Master_Port: 3306
      Connect_Retry: 60
      Master_Log_File: mysql-bin.000001
      Read_Master_Log_Pos: 154
      Relay_Log_File: mysql-relay.000002
      Relay_Log_Pos: 320
      Relay_Master_Log_File: mysql-bin.000001
      Slave_IO_Running: Yes
      Slave_SQL_Running: Yes
      Replicate_Do_DB:
```

#下面两个参数都是Yes，则说明主从配置成功！

# Slave\_IO\_Running: Yes

# Slave\_SQL\_Running: Yes

⑦ 两个主机互相复制

Master2 复制 Master1，Master1 复制 Master2

# Master2的复制命令

```
mysql> CHANGE MASTER TO MASTER_HOST='192.168.140.128',  
-> MASTER_USER='slave',  
-> MASTER_PASSWORD='123123',  
-> MASTER_LOG_FILE='mysql-bin.000008',MASTER_LOG_POS=154;  
Query OK, 0 rows affected, 2 warnings (0.01 sec)
```

# Master1的复制命令

```
mysql> CHANGE MASTER TO MASTER_HOST='192.168.140.126',  
-> MASTER_USER='slave',  
-> MASTER_PASSWORD='123123',  
-> MASTER_LOG_FILE='mysql-bin.000001',MASTER_LOG_POS=154;  
Query OK, 0 rows affected, 2 warnings (0.02 sec)
```

#启动两台主服务器复制功能

start slave;

#查看从服务器状态

show slave status\G;

Master2的复制Master1

```
mysql> show slave status\G;  
***** 1. row *****  
Slave_IO_State: Waiting for master to send event  
Master_Host: 192.168.140.128  
Master_User: slave  
Master_Port: 3306  
Connect_Retry: 60  
Master_Log_File: mysql-bin.000008  
Read_Master_Log_Pos: 154  
Relay_Log_File: host81-relay-bin.000002  
Relay_Log_Pos: 320  
Relay_Master_Log_File: mysql-bin.000008  
Slave_IO_Running: Yes  
Slave_SQL_Running: Yes
```

Master1的复制Master2

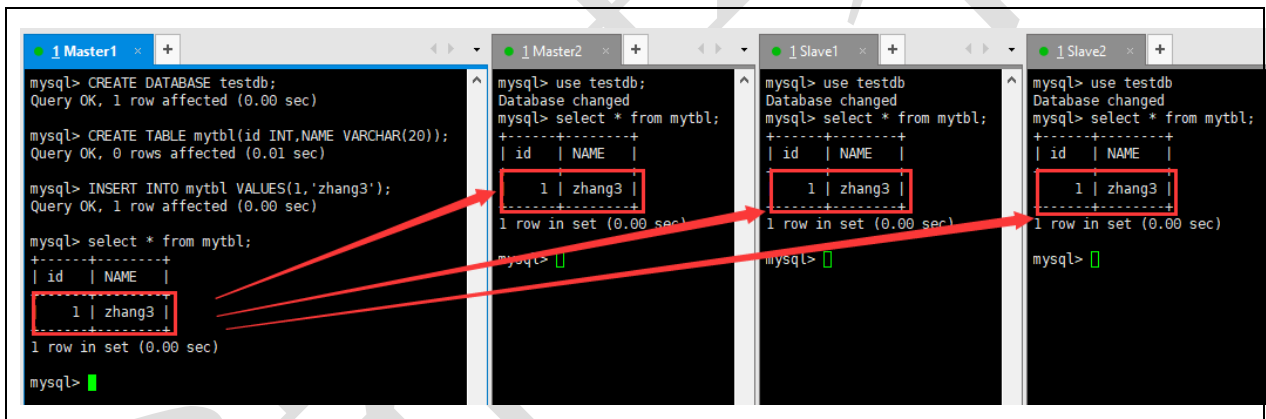
```
mysql> show slave status\G;
***** 1. row *****
Slave_IO_State: Waiting for master to send event
Master_Host: 192.168.140.126
Master_User: slave
Master_Port: 3306
Connect_Retry: 60
Master_Log_File: mysql-bin.000001
Read_Master_Log_Pos: 154
Relay_Log_File: host79-relay-bin.000002
Relay_Log_Pos: 320
Relay_Master_Log_File: mysql-bin.000001
Slave_IO_Running: Yes
Slave_SQL_Running: Yes
```

#下面两个参数都是Yes，则说明主从配置成功！

# Slave\_IO\_Running: Yes

# Slave\_SQL\_Running: Yes

⑧ Master1 主机新建库、新建表、insert 记录，Master2 和从机复制



The screenshot displays four MySQL terminal windows side-by-side, illustrating the replication process:

- Master1:** Executes `CREATE DATABASE testdb;`, `CREATE TABLE mytbl(id INT, NAME VARCHAR(20));`, and `INSERT INTO mytbl VALUES(1, 'zhang3');`. The final query result shows one row with id 1 and name zhang3.
- Master2:** Executes `use testdb;` and `select * from mytbl;`. The result shows the replicated row (1, zhang3).
- Slave1:** Executes `use testdb;` and `select * from mytbl;`. The result shows the replicated row (1, zhang3).
- Slave2:** Executes `use testdb;` and `select * from mytbl;`. The result shows the replicated row (1, zhang3).

Red boxes highlight the table structure and the inserted data in each terminal, with red arrows indicating the flow of replication from Master1 to Master2 and then to the slaves.

⑨ 如何停止从服务复制功能

stop slave;

⑩ 如何重新配置主从

stop slave;

reset master;

## 2、修改 Mycat 的配置文件 schema.xml

修改<dataHost>的balance属性，通过此属性配置读写分离的类型

更多 Java -大数据 -前端 -python 人工智能资料下载，可百度访问：尚硅谷官网

负载均衡类型，目前的取值有4种：

(1) `balance="0"`，不开启读写分离机制，所有读操作都发送到当前可用的 `writeHost` 上。

(2) `balance="1"`，全部的 `readHost` 与 `stand by writeHost` 参与 `select` 语句的负载均衡，简单的说，当双主双从模式(M1->S1, M2->S2, 并且 M1 与 M2 互为主备)，正常情况下，M2,S1,S2 都参与 `select` 语句的负载均衡。

(3) `balance="2"`，所有读操作都随机的在 `writeHost`、`readhost` 上分发。

(4) `balance="3"`，所有读请求随机的分发到 `readhost` 执行，`writerHost` 不负担读压力

为了双主双从读写分离`balance`设置为1

```
...
<dataNode name="dn1" dataHost="host1" database="testdb" />
  <dataHost name="host1" maxCon="1000" minCon="10" balance="1"
    writeType="0" dbType="mysql" dbDriver="native" switchType="1"
    slaveThreshold="100" >
    <heartbeat>select user()</heartbeat>
    <!-- can have multi write hosts -->
    <writeHost host="hostM1" url="192.168.140.128:3306" user="root"
      password="123123">
      <!-- can have multi read hosts -->
      <readHost host="hostS1" url="192.168.140.127:3306" user="root"
        password="123123" />
    </writeHost>
    <writeHost host="hostM2" url="192.168.140.126:3306" user="root"
      password="123123">
      <!-- can have multi read hosts -->
      <readHost host="hostS2" url="192.168.140.125:3306" user="root"
        password="123123" />
    </writeHost>
  </dataHost>
...
```

#`balance="1"`: 全部的`readHost`与`stand by writeHost`参与`select`语句的负载均衡。

#`writeType="0"`: 所有写操作发送到配置的第一个`writeHost`，第一个挂了切到还生存的第二个

#`writeType="1"`，所有写操作都随机的发送到配置的 `writeHost`，1.5 以后废弃不推荐

#`writeHost`，重新启动后以切换后的为准，切换记录在配置文件中: `dnindex.properties`。

#`switchType="1"`: 1 默认值，自动切换。

# -1 表示不自动切换

# 2 基于 MySQL 主从同步的状态决定是否切换。

```
<?xml version="1.0"?>
<!DOCTYPE mycat:schema SYSTEM "schema.dtd">
<mycat:schema xmlns:mycat="http://io.mycat/">

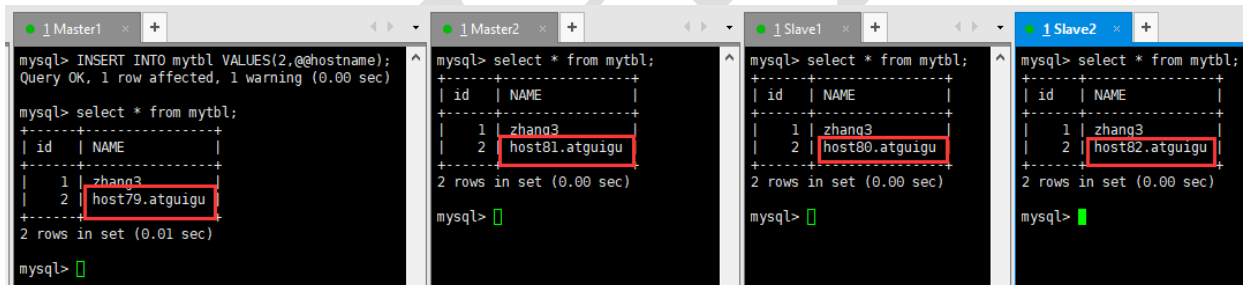
  <schema name="TESTDB" checkSQLschema="false" sqlMaxLimit="100" dataNode="dn1">
    </schema>
    <dataNode name="dn1" dataHost="host1" database="testdb" />
    <dataHost name="host1" maxCon="1000" minCon="10" balance="1"
      writeType="0" dbType="mysql" dbDriver="native" switchType="1" slaveThreshold="100">
      <heartbeat>select user()</heartbeat>
      <!-- can have multi write hosts -->
      <writeHost host="hostM1" url="192.168.140.128:3306" user="root"
        password="123123">
        <!-- can have multi read hosts -->
        <readHost host="hostS1" url="192.168.140.127:3306" user="root" password="123123" />
      </writeHost>
      <writeHost host="hostM2" url="192.168.140.126:3306" user="root"
        password="123123">
        <!-- can have multi read hosts -->
        <readHost host="hostS2" url="192.168.140.125:3306" user="root" password="123123" />
      </writeHost>
    </dataHost>
  </mycat:schema>
```

### 3、启动 Mycat

### 4、验证读写分离

#在写主机Master1数据库表mytbl中插入带系统变量数据，造成主从数据不一致

INSERT INTO mytbl VALUES(3,@hostname);



The image shows four terminal windows representing different MySQL instances in a Mycat setup:

- Master1:** Executes `INSERT INTO mytbl VALUES(2,@hostname);` and `select * from mytbl;`. The result shows two rows: (1, zhang3) and (2, host79.atguigu).
- Master2:** Executes `select * from mytbl;`. The result shows two rows: (1, zhang3) and (2, host81.atguigu).
- Slave1:** Executes `select * from mytbl;`. The result shows two rows: (1, zhang3) and (2, host80.atguigu).
- Slave2:** Executes `select * from mytbl;`. The result shows two rows: (1, zhang3) and (2, host82.atguigu).

#在Mycat里查询mytbl表,可以看到查询语句在Master2 (host81)、Slave1 (host80)、Slave2 (host82) 主从三个主机间切换

```
1 mycat x 2 conf x 3 bin x +
mysql> select * from mytbl;
+-----+-----+
| id  | NAME          |
+-----+-----+
| 1   | zhang3        |
| 2   | host81.atguigu|
+-----+-----+
2 rows in set (0.00 sec)

mysql> select * from mytbl;
+-----+-----+
| id  | NAME          |
+-----+-----+
| 1   | zhang3        |
| 2   | host80.atguigu|
+-----+-----+
2 rows in set (0.01 sec)

mysql> select * from mytbl;
+-----+-----+
| id  | NAME          |
+-----+-----+
| 1   | zhang3        |
| 2   | host82.atguigu|
+-----+-----+
2 rows in set (0.01 sec)
```

## 5、抗风险能力

#停止数据库Master1

```
[root@host79 ~]# systemctl stop mysqld
[root@host79 ~]# systemctl status mysqld
● mysqld.service - MySQL Server
   Loaded: loaded (/usr/lib/systemd/system/mysqld.service; enabled;
   vendor preset: disabled)
   Active: inactive (dead) since 五 2019-08-02 00:04:54 CST; 8s ago
     Process: 5253 ExecStart=/usr/sbin/mysqld --daemonize --pid-file=/
var/run/mysqld/mysqld.pid $MYSQLD_OPTS (code=exited, status=0/SUCCESS)
```

#在Mycat里插入数据依然成功，Master2自动切换为写主机

```
INSERT INTO mytbl VALUES(3,@hostname);
```



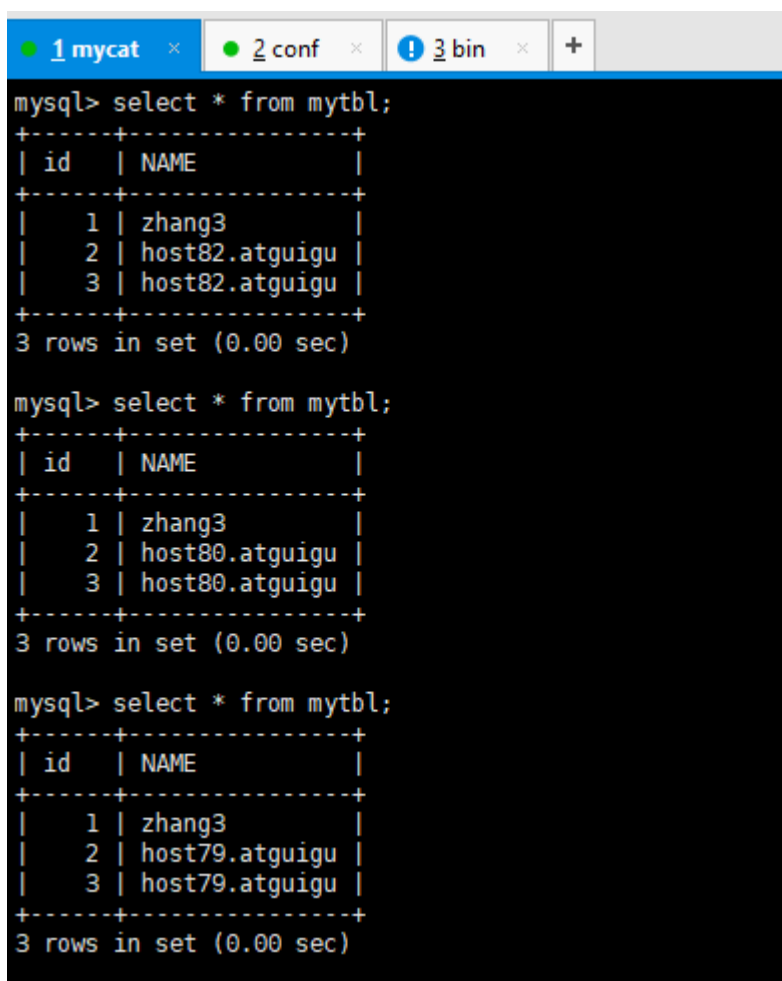
```
1 mycat x 2 conf x 3 bin x +
mysql> INSERT INTO mytbl VALUES(3,@hostname);
Query OK, 1 row affected, 1 warning (0.01 sec)

mysql> select * from mytbl;
+-----+-----+
| id  | NAME          |
+-----+-----+
| 1   | zhang3        |
| 2   | host82.atguigu |
| 3   | host82.atguigu |
+-----+-----+
3 rows in set (0.01 sec)
```

#启动数据库Master1

```
[root@host79 ~]# systemctl start mysqld
[root@host79 ~]# systemctl status mysqld
● mysqld.service - MySQL Server
   Loaded: loaded (/usr/lib/systemd/system/mysqld.service; enabled; preset: disabled)
   Active: active (running) since 五 2019-08-02 00:09:08 CST;
   Process: 6424 ExecStart=/usr/sbin/mysqld --daemonize --pid-file=/usr/lib/mysql/mysql.pid $MYSQLD_OPTS (code=exited, status=0/SUCCESS)
   Process: 6400 ExecStartPre=/usr/bin/mysqld_pre_systemd (code=exited, status=0/SUCCESS)
```

#在Mycat里查询mytbl表,可以看到查询语句在Master1 (host79) 、Slava1 (host80) 、Slava2 (host82) 主从三个主机间切换



```
mysql> select * from mytbl;
+-----+-----+
| id  | NAME          |
+-----+-----+
| 1   | zhang3        |
| 2   | host82.atguigu |
| 3   | host82.atguigu |
+-----+-----+
3 rows in set (0.00 sec)

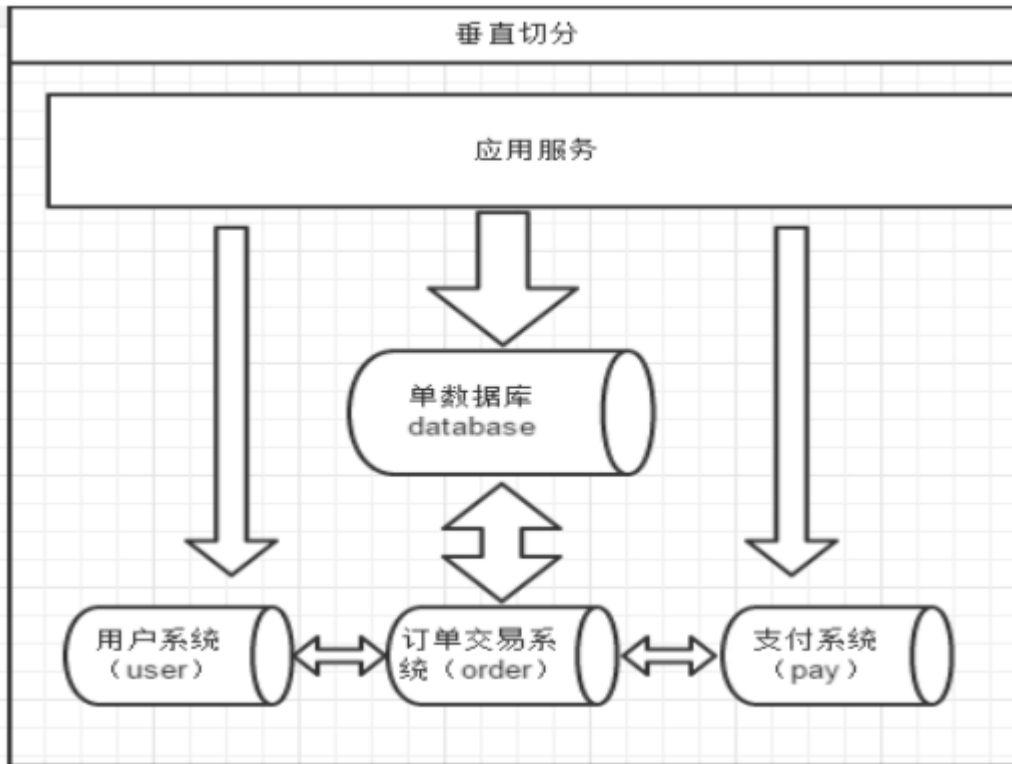
mysql> select * from mytbl;
+-----+-----+
| id  | NAME          |
+-----+-----+
| 1   | zhang3        |
| 2   | host80.atguigu |
| 3   | host80.atguigu |
+-----+-----+
3 rows in set (0.00 sec)

mysql> select * from mytbl;
+-----+-----+
| id  | NAME          |
+-----+-----+
| 1   | zhang3        |
| 2   | host79.atguigu |
| 3   | host79.atguigu |
+-----+-----+
3 rows in set (0.00 sec)
```

**Master1、Master2** 互做备机，负责写的主机宕机，备机切换负责写操作，保证数据库读写分离高可用性。

## 第四章 垂直拆分——分库

一个数据库由很多表的构成，每个表对应着不同的业务，垂直切分是指按照业务将表进行分类，分布到不同的数据库上面，这样也就将数据或者说压力分担到不同的库上面，如下图：



系统被切分成了，用户，订单交易，支付几个模块。

## 4.1 如何划分表

一个问题：在两台主机上的两个数据库中的表，能否关联查询？

答案：不可以关联查询。

分库的原则：有紧密关联关系的表应该在一个库里，相互没有关联关系的表可以分到不同的库里。

```
#客户表 rows:20万
CREATE TABLE customer(
    id INT AUTO_INCREMENT,
    NAME VARCHAR(200),
    PRIMARY KEY(id)
);
#订单表 rows:600万
CREATE TABLE orders(
    id INT AUTO_INCREMENT,
    order_type INT,
    customer_id INT,
    amount DECIMAL(10,2),
    PRIMARY KEY(id)
```

```
);  
#订单详细表      rows:600万  
CREATE TABLE orders_detail(  
    id INT AUTO_INCREMENT,  
    detail VARCHAR(2000),  
    order_id INT,  
    PRIMARY KEY(id)  
);  
#订单状态字典表  rows:20  
CREATE TABLE dict_order_type(  
    id INT AUTO_INCREMENT,  
    order_type VARCHAR(200),  
    PRIMARY KEY(id)  
);
```

以上四个表如何分库？客户表分在一个数据库，另外三张都需要关联查询，分在另外一个数据库。

## 4.2 实现分库

### 1、修改 schema 配置文件

```
...  
<schema name="TESTDB" checkSQLschema="false" sqlMaxLimit="100" dataNode="dn1">  
    <table name="customer" dataNode="dn2" ></table>  
</schema>  
<dataNode name="dn1" dataHost="host1" database="orders" />  
<dataNode name="dn2" dataHost="host2" database="orders" />  
<dataHost name="host1" maxCon="1000" minCon="10" balance="0"  
    writeType="0" dbType="mysql" dbDriver="native" switchType="1"  
slaveThreshold="100">  
    <heartbeat>select user()</heartbeat>  
    <!-- can have multi write hosts -->  
    <writeHost host="hostM1" url="192.168.140.128:3306" user="root"  
        password="123123">  
    </writeHost>  
</dataHost>  
<dataHost name="host2" maxCon="1000" minCon="10" balance="0"  
    writeType="0" dbType="mysql" dbDriver="native" switchType="1"  
slaveThreshold="100">
```

```
<heartbeat>select user()</heartbeat>
<!-- can have multi write hosts -->
<writeHost host="hostM2" url="192.168.140.127:3306" user="root"
           password="123123">

</writeHost>
</dataHost>
```

...

#如下图

```
<?xml version="1.0"?>
<!DOCTYPE mycat:schema SYSTEM "schema.dtd">
<mycat:schema xmlns:mycat="http://io.mycat/">

  <schema name="TESTDB" checkSQLschema="false" sqlMaxLimit="100" dataNode="dn1">
    <table name="customer" dataNode="dn2" ></table>
  </schema>
  <dataNode name="dn1" dataHost="host1" database="orders" />
  <dataNode name="dn2" dataHost="host2" database="orders" />
  <dataHost name="host1" maxCon="1000" minCon="10" balance="0"
            writeType="0" dbType="mysql" dbDriver="native" switchType="1" slaveThreshold="100">
    <heartbeat>select user()</heartbeat>
    <!-- can have multi write hosts -->
    <writeHost host="hostM1" url="192.168.140.128:3306" user="root"
              password="123123">

    </writeHost>
  </dataHost>
  <dataHost name="host2" maxCon="1000" minCon="10" balance="0"
            writeType="0" dbType="mysql" dbDriver="native" switchType="1" slaveThreshold="100">
    <heartbeat>select user()</heartbeat>
    <!-- can have multi write hosts -->
    <writeHost host="hostM2" url="192.168.140.127:3306" user="root"
              password="123123">

    </writeHost>
  </dataHost>
</mycat:schema>
```

## 2、新增两个空白库

分库操作不是在原来的老数据库上进行操作，需要准备两台机器分别安装新的数据库

#在数据节点 dn1、dn2 上分别创建数据库 orders

```
CREATE DATABASE orders;
```

## 3、启动 Mycat

```
./mycat console
```

```
[root@host79 bin]# ./mycat console
Running Mycat-server...
wrapper | --> Wrapper Started as Console
wrapper | Launching a JVM...
jvm 1   | Java HotSpot(TM) 64-Bit Server VM warning: ignoring option MaxPermSize
jvm 1   | Wrapper (Version 3.2.3) http://wrapper.tanukisoftware.org
jvm 1   | Copyright 1999-2006 Tanuki Software, Inc. All Rights Reserved.
jvm 1   |
jvm 1   | 2019-08-04 17:29:07,534 [INFO ][WrapperSimpleAppMain] total resources o
e.PhysicalDBPool:PhysicalDBPool.java:100)
```

#### 4、访问 Mycat 进行分库

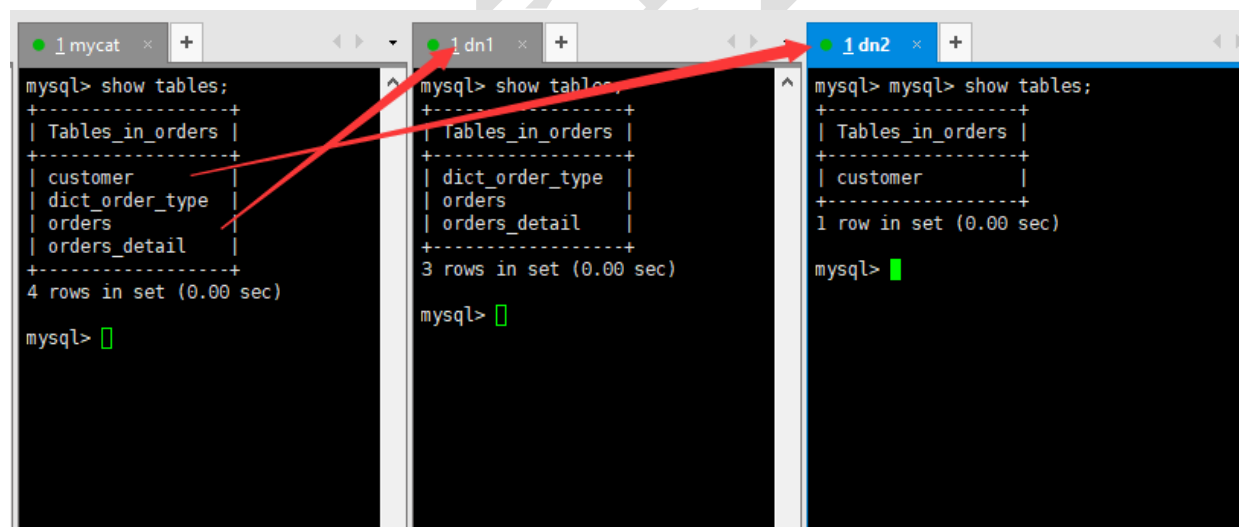
#访问 Mycat

mysql -umycat -p123456 -h 192.168.140.128 -P 8066

#切换到 TESTDB

#创建 4 张表

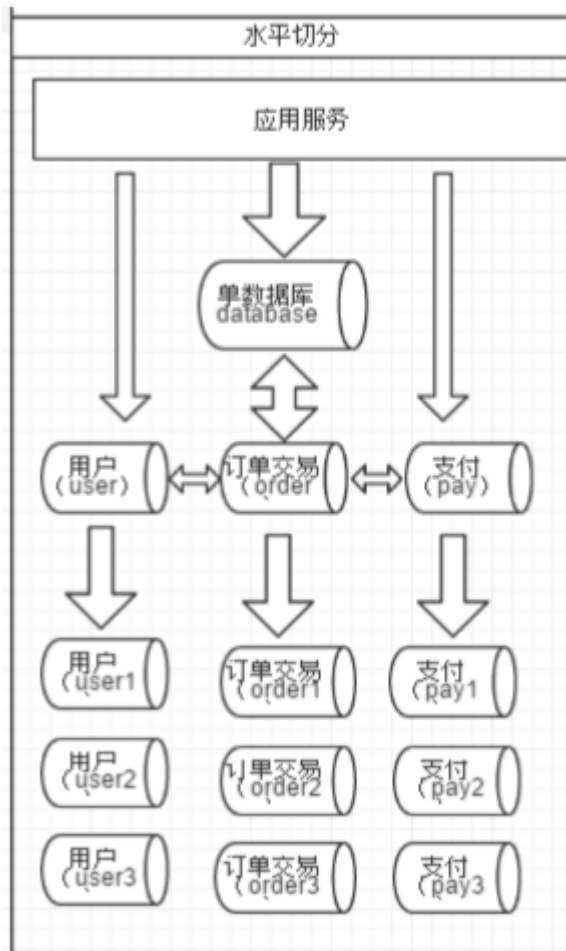
#查看表信息，可以看到成功分库



## 第五章 水平拆分——分表

相对于垂直拆分，水平拆分不是将表做分类，而是按照某个字段的某种规则来分散到多个库之中，每个表中 包含一部分数据。简单来说，我们可以将数据的水平切分理解为是按照数据行的切分，就是将表中的某些行切分 到一个数据库，而另外的某些行又切分到其他的数据库中，如图：

更多 [Java](#) -[大数据](#) -[前端](#) -[python](#) 人工智能资料下载，可[百度访问](#)：尚硅谷官网



## 5.1 实现分表

### 1、选择要拆分的表

MySQL 单表存储数据条数是有瓶颈的，单表达到 1000 万条数据就达到了瓶颈，会影响查询效率，需要进行水平拆分（分表）进行优化。

例如：例子中的 orders、orders\_detail 都已经达到 600 万行数据，需要进行分表优化。

### 2、分表字段

以 orders 表为例，可以根据不同自字段进行分表

编号	分表字段	效果
1	id（主键、或创建时间）	查询订单注重时效，历史订单被查询的次数少，如此

		分片会造成一个节点访问多，一个访问少，不平均。
2	customer_id (客户 id)	根据客户 id 去分，两个节点访问平均，一个客户的所有订单都在同一个节点

### 3、修改配置文件 schema.xml

#为 orders 表设置数据节点为 dn1、dn2，并指定分片规则为 mod\_rule（自定义的名字）

```
<table name="orders" dataNode="dn1,dn2" rule="mod_rule" ></table>
```

#如下图

```
<schema name="TESTDB" checkSQLschema="false" sqlMaxLimit="100" dataNode="dn1">
  <table name="customer" dataNode="dn2" ></table>
  <table name="orders" dataNode="dn1,dn2" rule="mod_rule" ></table>
</schema>
```

### 4、修改配置文件 rule.xml

#在 rule 配置文件里新增分片规则 mod\_rule，并指定规则适用字段为 customer\_id，

#还有选择分片算法 mod-long（对字段求模运算），customer\_id 对两个节点求模，根据结果分片

#配置算法 mod-long 参数 count 为 2，两个节点

```
<tableRule name="mod_rule">
  <rule>
    <columns>customer_id</columns>
    <algorithm>mod-long</algorithm>
  </rule>
</tableRule>
...
<function name="mod-long" class="io.mycat.route.function.PartitionByMod">
  <!-- how many data nodes -->
  <property name="count">2</property>
</function>
```

#如下图：



```
<tableRule name="mod_rule">
  <rule>
    <columns>customer_id</columns>
    <algorithm>mod-long</algorithm>
  </rule>
</tableRule>
```

```
<function name="mod-long" class="io.mycat.route.function.PartitionByMod">
  <!-- how many data nodes -->
  <property name="count">2</property>
</function>
```

5、在数据节点 dn2 上建 orders 表

6、重启 Mycat，让配置生效

7、访问 Mycat 实现分片

#在 mycat 里向 orders 表插入数据，INSERT 字段不能省略

```
INSERT INTO orders(id,order_type,customer_id,amount) VALUES (1,101,100,100100);
```

```
INSERT INTO orders(id,order_type,customer_id,amount) VALUES(2,101,100,100300);
```

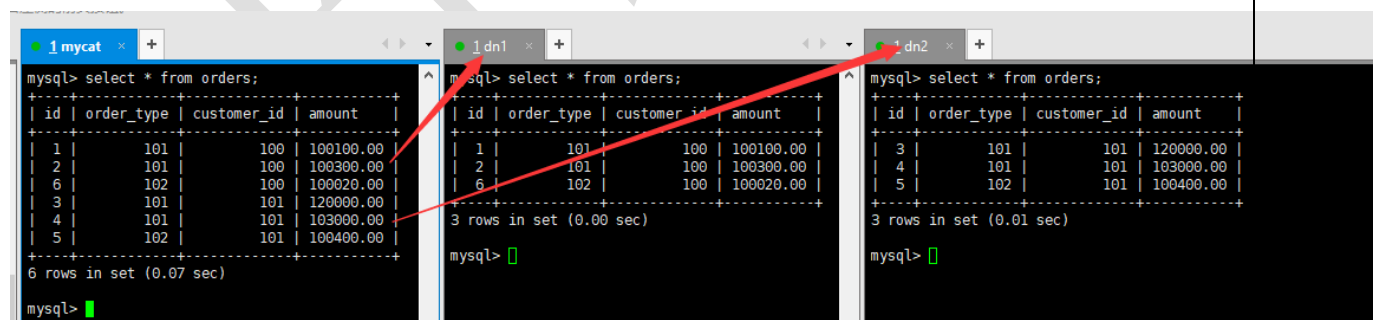
```
INSERT INTO orders(id,order_type,customer_id,amount) VALUES(3,101,101,120000);
```

```
INSERT INTO orders(id,order_type,customer_id,amount) VALUES(4,101,101,103000);
```

```
INSERT INTO orders(id,order_type,customer_id,amount) VALUES(5,102,101,100400);
```

```
INSERT INTO orders(id,order_type,customer_id,amount) VALUES(6,102,100,100020);
```

#在mycat、dn1、dn2中查看orders表数据，分表成功



The screenshot shows three terminal windows with the following data:

id	order_type	customer_id	amount
1	101	100	100100.00
2	101	100	100300.00
6	102	100	100020.00
3	101	101	120000.00
4	101	101	103000.00
5	102	101	100400.00

6 rows in set (0.07 sec)

id	order_type	customer_id	amount
1	101	100	100100.00
2	101	100	100300.00
6	102	100	100020.00

3 rows in set (0.00 sec)

id	order_type	customer_id	amount
3	101	101	120000.00
4	101	101	103000.00
5	102	101	100400.00

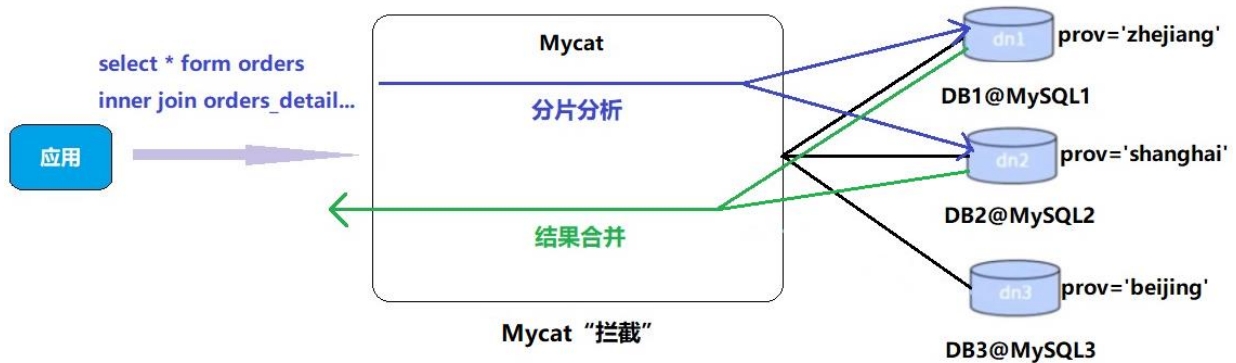
3 rows in set (0.01 sec)

## 5.2 Mycat 的分片 “join”

Orders 订单表已经进行分表操作了，和它关联的 orders\_detail 订单详情表如何进行 join 查询。

更多 Java -大数据 -前端 -python 人工智能资料下载，可百度访问：尚硅谷官网

我们要对 orders\_detail 也要进行分片操作。Join 的原理如下图：



## 1、ER 表

Mycat 借鉴了 NewSQL 领域的新秀 Foundation DB 的设计思路，Foundation DB 创新性的提出了 Table Group 的概念，其将子表的存储位置依赖于主表，并且物理上紧邻存放，因此彻底解决了 JOIN 的效率和性能问题，根据这一思路，提出了基于 E-R 关系的数据分片策略，子表的记录与所关联的父表记录存放在同一个数据分片上。

#修改 schema.xml 配置文件

```
...
<table name="orders" dataNode="dn1,dn2" rule="mod_rule" >
  <childTable name="orders_detail" primaryKey="id" joinKey="order_id" parentKey="id" />
</table>
...
```

```
<table name="orders" dataNode="dn1,dn2" rule="mod_rule" >
  <childTable name="orders_detail" primaryKey="id" joinKey="order_id" parentKey="id" />
</table>
```

#在 dn2 创建 orders\_detail 表

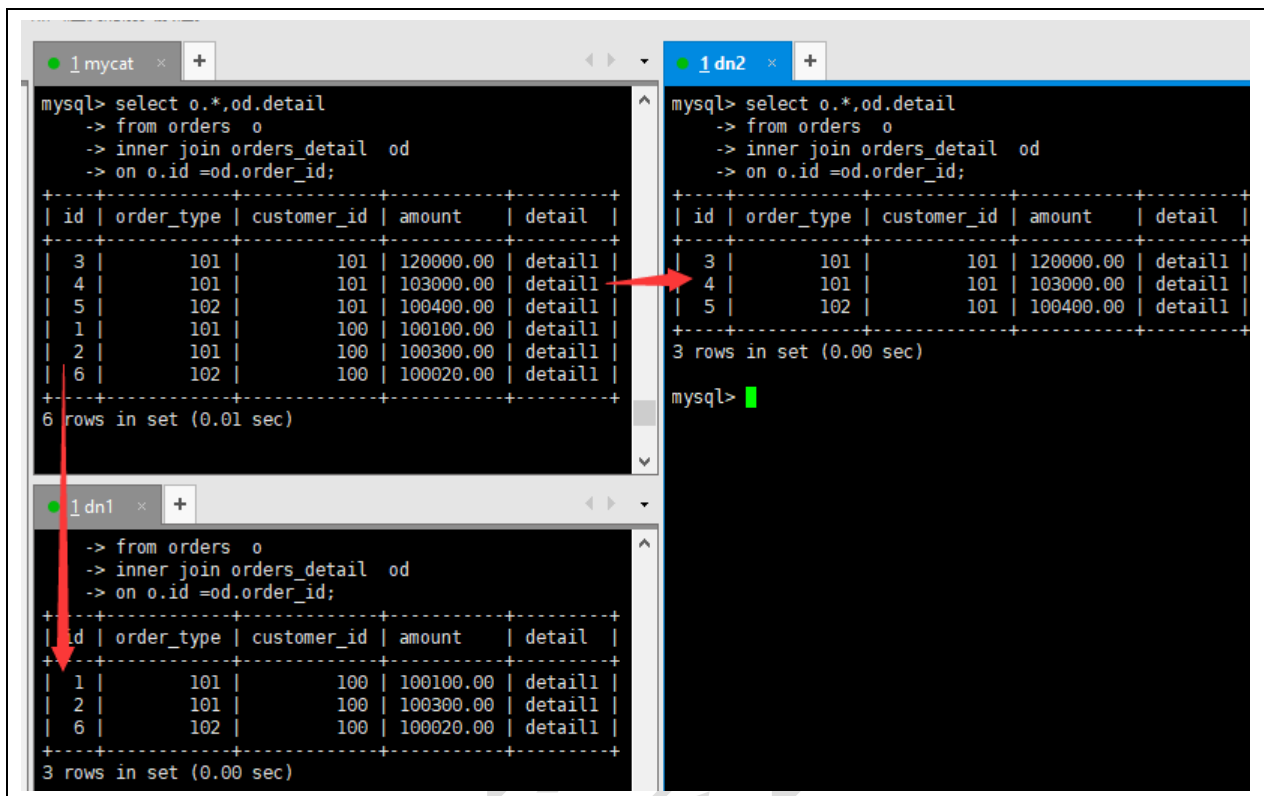
#重启 Mycat

#访问 Mycat 向 orders\_detail 表插入数据

```
INSERT INTO orders_detail(id,detail,order_id) values(1,'detail1',1);
INSERT INTO orders_detail(id,detail,order_id) VALUES(2,'detail1',2);
INSERT INTO orders_detail(id,detail,order_id) VALUES(3,'detail1',3);
INSERT INTO orders_detail(id,detail,order_id) VALUES(4,'detail1',4);
INSERT INTO orders_detail(id,detail,order_id) VALUES(5,'detail1',5);
INSERT INTO orders_detail(id,detail,order_id) VALUES(6,'detail1',6);
```

#在mycat、dn1、dn2中运行两个表join语句

```
Select o.*,od.detail from orders o inner join orders_detail od on o.id=od.order_id;
```



```
mysql> select o.*,od.detail
-> from orders o
-> inner join orders_detail od
-> on o.id =od.order_id;
+-----+-----+-----+-----+-----+
| id | order_type | customer_id | amount | detail |
+-----+-----+-----+-----+-----+
| 3 | 101 | 101 | 120000.00 | detail1 |
| 4 | 101 | 101 | 103000.00 | detail1 |
| 5 | 102 | 101 | 100400.00 | detail1 |
| 1 | 101 | 100 | 100100.00 | detail1 |
| 2 | 101 | 100 | 100300.00 | detail1 |
| 6 | 102 | 100 | 100020.00 | detail1 |
+-----+-----+-----+-----+-----+
6 rows in set (0.01 sec)
```

```
mysql> select o.*,od.detail
-> from orders o
-> inner join orders_detail od
-> on o.id =od.order_id;
+-----+-----+-----+-----+-----+
| id | order_type | customer_id | amount | detail |
+-----+-----+-----+-----+-----+
| 3 | 101 | 101 | 120000.00 | detail1 |
| 4 | 101 | 101 | 103000.00 | detail1 |
| 5 | 102 | 101 | 100400.00 | detail1 |
+-----+-----+-----+-----+-----+
3 rows in set (0.00 sec)
```

```
mysql>
mysql>
```

```
mysql> select o.*,od.detail
-> from orders o
-> inner join orders_detail od
-> on o.id =od.order_id;
+-----+-----+-----+-----+-----+
| id | order_type | customer_id | amount | detail |
+-----+-----+-----+-----+-----+
| 1 | 101 | 100 | 100100.00 | detail1 |
| 2 | 101 | 100 | 100300.00 | detail1 |
| 6 | 102 | 100 | 100020.00 | detail1 |
+-----+-----+-----+-----+-----+
3 rows in set (0.00 sec)
```

## 2、全局表

在分片的情况下，当业务表因为规模而进行分片以后，业务表与这些附属的字典表之间的关联，就成了比较棘手的问题，考虑到字典表具有以下几个特性：

- ① 变动不频繁
- ② 数据量总体变化不大
- ③ 数据规模不大，很少有超过数十万条记录

鉴于此，Mycat 定义了一种特殊的表，称之为“全局表”，全局表具有以下特性：

- ① 全局表的插入、更新操作会实时在所有节点上执行，保持各个分片的数据一致性
- ② 全局表的查询操作，只从一个节点获取
- ③ 全局表可以跟任何一个表进行 JOIN 操作

将字典表或者符合字典表特性的一些表定义为全局表，则从另外一个方面，很好的解决了数据 JOIN 的难题。通过全局表+基于 E-R 关系的分片策略，Mycat 可以满足 80% 以上的企业应用开发

#修改 schema.xml 配置文件

```
...  
<table name="orders" dataNode="dn1,dn2" rule="mod_rule" >  
    <childTable name="orders_detail" primaryKey="id" joinKey="order_id" parentKey="id" />  
</table>  
<table name="dict_order_type" dataNode="dn1,dn2" type="global" ></table>  
...
```

```
<schema name="TESTDB" checkSQLSchema="false" sqlMaxLimit="100" dataNode="dn1">  
    <table name="customer" dataNode="dn2" ></table>  
    <table name="orders" dataNode="dn1,dn2" rule="mod_rule" >  
        <childTable name="orders_detail" primaryKey="id" joinKey="order_id" parentKey="id" />  
    </table>  
    <table name="dict_order_type" dataNode="dn1,dn2" type="global" ></table>  
</schema>
```

#在 dn2 创建 dict\_order\_type 表

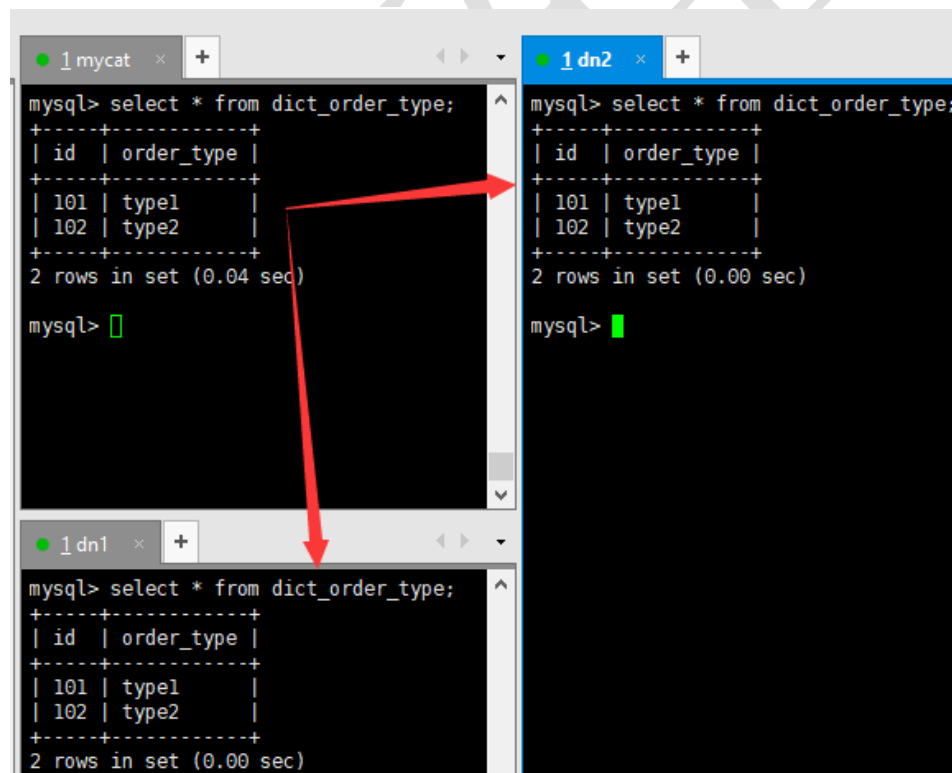
#重启 Mycat

#访问 Mycat 向 dict\_order\_type 表插入数据

```
INSERT INTO dict_order_type(id,order_type) VALUES(101,'type1');
```

```
INSERT INTO dict_order_type(id,order_type) VALUES(102,'type2');
```

#在Mycat、dn1、dn2中查询表数据



The screenshot shows three terminal windows, each with a MySQL prompt. The top-left window is titled '1 mycat' and shows a query 'select \* from dict\_order\_type;' with results for id 101 (type1) and id 102 (type2). The top-right window is titled '1 dn2' and shows the same query with the same results. The bottom window is titled '1 dn1' and also shows the same query with the same results. A red arrow points from the '1 dn2' window to the '1 dn1' window, indicating data consistency or replication.

```
mysql> select * from dict_order_type;  
+----+-----+  
| id | order_type |  
+----+-----+  
| 101 | type1      |  
| 102 | type2      |  
+----+-----+  
2 rows in set (0.04 sec)  
  
mysql>   
  
mysql> select * from dict_order_type;  
+----+-----+  
| id | order_type |  
+----+-----+  
| 101 | type1      |  
| 102 | type2      |  
+----+-----+  
2 rows in set (0.00 sec)  
  
mysql>   
  
mysql> select * from dict_order_type;  
+----+-----+  
| id | order_type |  
+----+-----+  
| 101 | type1      |  
| 102 | type2      |  
+----+-----+  
2 rows in set (0.00 sec)
```

## 5.3 常用分片规则

### 1、取模

此规则为对分片字段求摸运算。也是水平分表最常用规则。5.1 配置分表中，orders 表采用了此规则。

### 2、分片枚举

通过在配置文件中配置可能的枚举 id，自己配置分片，本规则适用于特定的场景，比如有些业务需要按照省份或区县来做保存，而全国省份区县固定的，这类业务使用本条规则。

# (1) 修改schema.xml配置文件

```
<table name="orders_ware_info" dataNode="dn1,dn2" rule="sharding_by_intfile"></table>
```

# (2) 修改rule.xml配置文件

```
<tableRule name="sharding_by_intfile">
    <rule>
        <columns>areacode</columns>
        <algorithm>hash-int</algorithm>
    </rule>
</tableRule>
```

...

```
<function name="hash-int"
    class="io.mycat.route.function.PartitionByFileMap">
    <property name="mapFile">partition-hash-int.txt</property>
    <property name="type">1</property>
    <property name="defaultNode">0</property>
</function>
```

# columns: 分片字段，algorithm: 分片函数

# mapFile: 标识配置文件名称，type: 0为int型、非0为String，

#defaultNode: 默认节点:小于 0 表示不设置默认节点，大于等于 0 表示设置默认节点，

# 设置默认节点如果碰到不识别的枚举值，就让它路由到默认节点，如不设置不识别就报错

# (3) 修改partition-hash-int.txt配置文件

110=0

120=1

# (4) 重启 Mycat

# (5) 访问Mycat创建表

#订单归属区域信息表

```
CREATE TABLE orders_ware_info
```

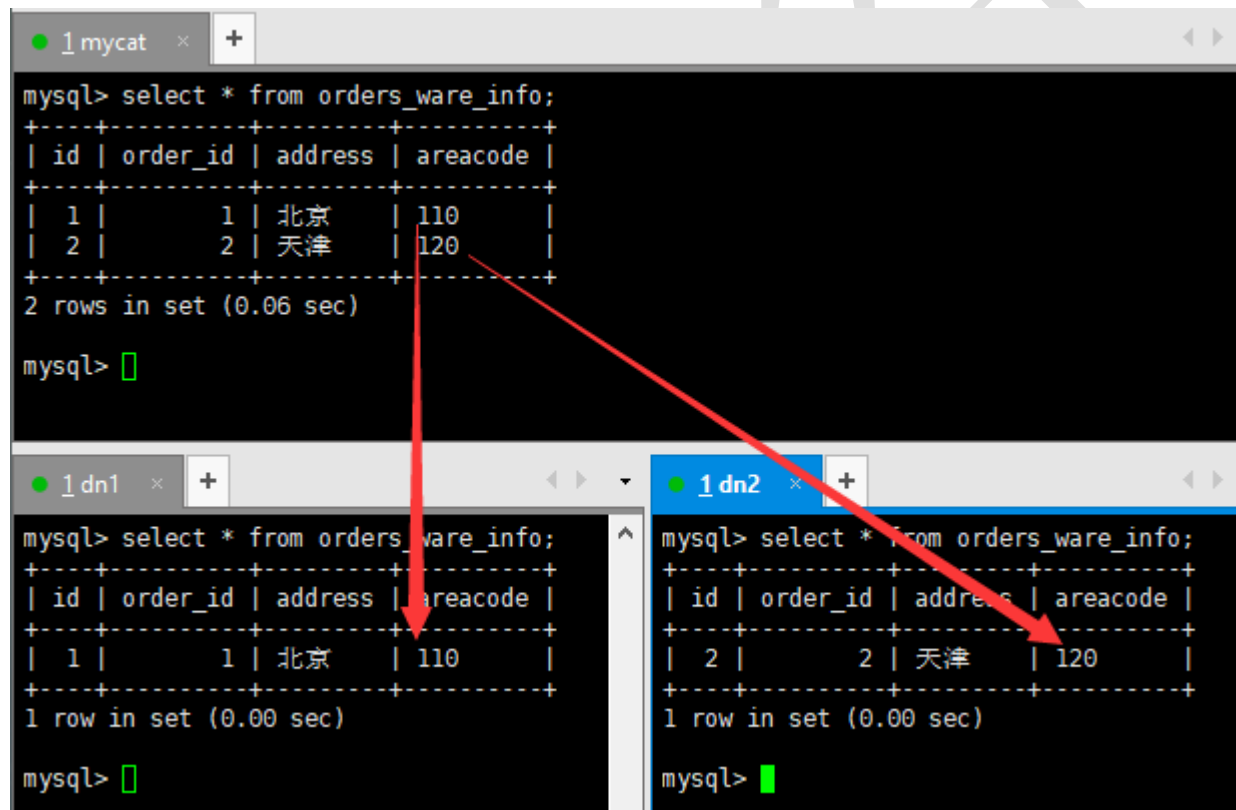
```
(  
  `id`          INT AUTO_INCREMENT comment '编号',  
  `order_id`    INT comment '订单编号',  
  `address`     VARCHAR(200) comment '地址',  
  `areacode`    VARCHAR(20) comment '区域编号',  
  PRIMARY KEY(id)  
);
```

# (6) 插入数据

```
INSERT INTO orders_ware_info(id, order_id, address, areacode) VALUES (1,1,'北京','110');
```

```
INSERT INTO orders_ware_info(id, order_id, address, areacode) VALUES (2,2,'天津','120');
```

# (7) 查询Mycat、dn1、dn2可以看到数据分片效果



```
mysql> select * from orders_ware_info;  
+-----+-----+-----+-----+  
| id | order_id | address | areacode |  
+-----+-----+-----+-----+  
| 1 | 1 | 北京 | 110 |  
| 2 | 2 | 天津 | 120 |  
+-----+-----+-----+-----+  
2 rows in set (0.06 sec)  
  
mysql>   
  
mysql> select * from orders_ware_info;  
+-----+-----+-----+-----+  
| id | order_id | address | areacode |  
+-----+-----+-----+-----+  
| 1 | 1 | 北京 | 110 |  
+-----+-----+-----+-----+  
1 row in set (0.00 sec)  
  
mysql>   
  
mysql> select * from orders_ware_info;  
+-----+-----+-----+-----+  
| id | order_id | address | areacode |  
+-----+-----+-----+-----+  
| 2 | 2 | 天津 | 120 |  
+-----+-----+-----+-----+  
1 row in set (0.00 sec)  
  
mysql> 
```

### 3、范围约定

此分片适用于，提前规划好分片字段某个范围属于哪个分片。

# (1) 修改schema.xml配置文件

```
<table name="payment_info" dataNode="dn1,dn2" rule="auto_sharding_long" ></table>
```

# (2) 修改rule.xml配置文件

```
<tableRule name="auto_sharding_long">
```

```
<rule>
```

```
<columns>order_id</columns>
```

```
<algorithm>rang-long</algorithm>
```

```
</rule>
```

```
</tableRule>
```

...

```
<function name="rang-long"
```

```
class="io.mycat.route.function.AutoPartitionByLong">
```

```
<property name="mapFile">autopartition-long.txt</property>
```

```
<property name="defaultNode">0</property>
```

```
</function>
```

# columns: 分片字段, algorithm: 分片函数

# mapFile: 标识配置文件名称

# defaultNode: 默认节点: 小于 0 表示不设置默认节点, 大于等于 0 表示设置默认节点,

# 设置默认节点如果碰到不识别的枚举值, 就让它路由到默认节点, 如不设置不识别就报错

# (3) 修改autopartition-long.txt配置文件

0-102=0

103-200=1

# (4) 重启 Mycat

# (5) 访问Mycat创建表

# 支付信息表

```
CREATE TABLE payment_info
```

```
(
```

```
`id` INT AUTO_INCREMENT comment '编号',
```

```
`order_id` INT comment '订单编号',
```

```
`payment_status` INT comment '支付状态',
```

```
PRIMARY KEY(id)
```

```
);
```

# (6) 插入数据

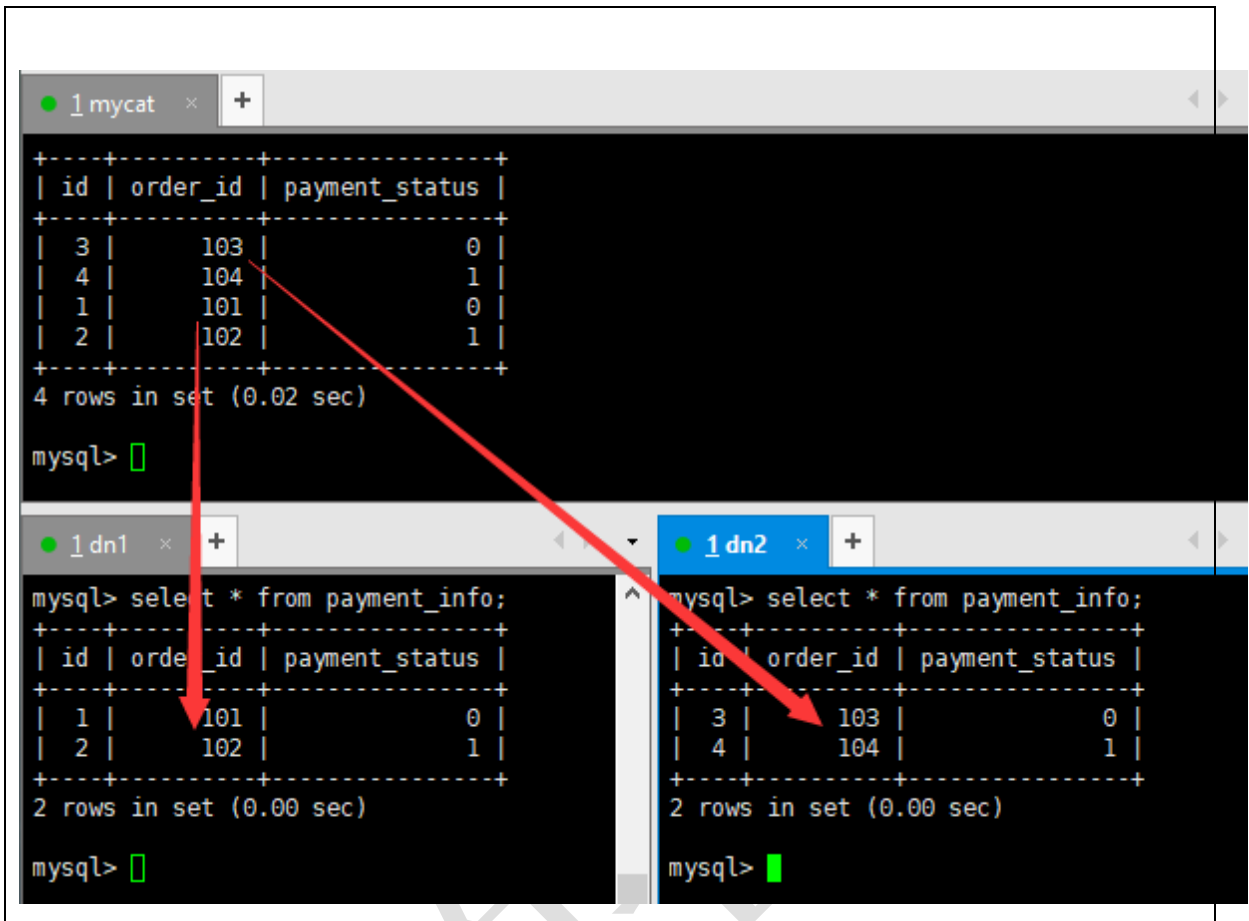
```
INSERT INTO payment_info (id,order_id,payment_status) VALUES (1,101,0);
```

```
INSERT INTO payment_info (id,order_id,payment_status) VALUES (2,102,1);
```

```
INSERT INTO payment_info (id,order_id,payment_status) VALUES (3,103,0);
```

```
INSERT INTO payment_info (id,order_id,payment_status) VALUES (4,104,1);
```

# (7) 查询Mycat、dn1、dn2可以看到数据分片效果



```
mysql> select * from payment_info;
+----+-----+-----+
| id | order_id | payment_status |
+----+-----+-----+
| 3 | 103 | 0 |
| 4 | 104 | 1 |
| 1 | 101 | 0 |
| 2 | 102 | 1 |
+----+-----+-----+
4 rows in set (0.02 sec)

mysql>

mysql> select * from payment_info;
+----+-----+-----+
| id | order_id | payment_status |
+----+-----+-----+
| 1 | 101 | 0 |
| 2 | 102 | 1 |
+----+-----+-----+
2 rows in set (0.00 sec)

mysql>

mysql> select * from payment_info;
+----+-----+-----+
| id | order_id | payment_status |
+----+-----+-----+
| 3 | 103 | 0 |
| 4 | 104 | 1 |
+----+-----+-----+
2 rows in set (0.00 sec)

mysql>
```

#### 4、按日期（天）分片

此规则为按天分片。设定时间格式、范围

# (1) 修改schema.xml配置文件

```
<table name="login_info" dataNode="dn1,dn2" rule="sharding_by_date" ></table>
```

# (2) 修改rule.xml配置文件

```
<tableRule name="sharding_by_date">
```

```
<rule>
```

```
<columns>login_date</columns>
```

```
<algorithm>shardingByDate</algorithm>
```

```
</rule>
```

```
</tableRule>
```

...

```
<function name="shardingByDate" class="io.mycat.route.function.PartitionByDate">
```

```
<property name="dateFormat">yyyy-MM-dd</property>
```

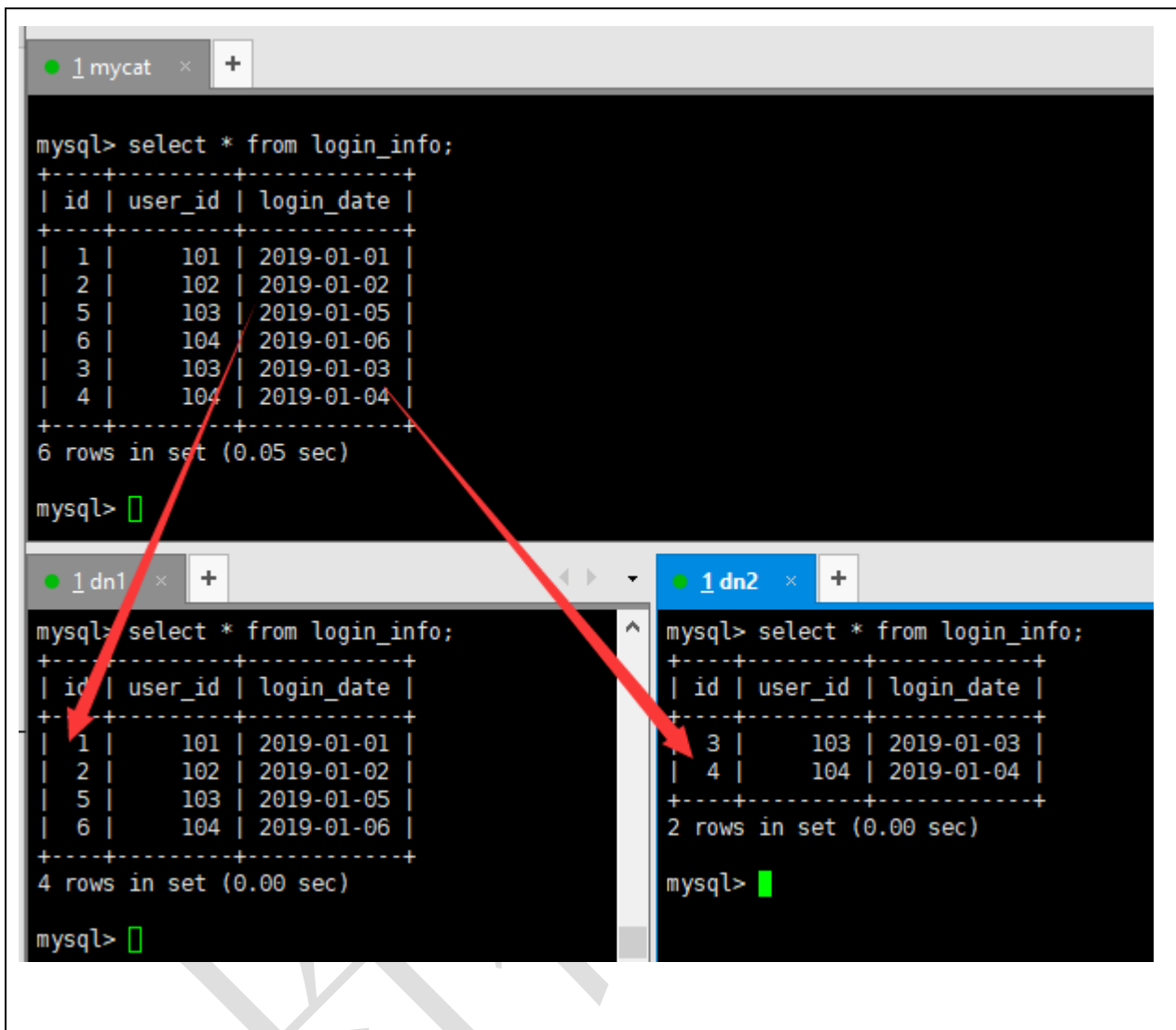
```
<property name="sBeginDate">2019-01-01</property>
```

```
<property name="sEndDate">2019-01-04</property>
```



```
<property name="sPartionDay">2</property>
</function>
# columns: 分片字段, algorithm: 分片函数
#dateFormat : 日期格式
#sBeginDate : 开始日期
#sEndDate: 结束日期,则代表数据达到了这个日期的分片后循环从开始分片插入
#sPartionDay : 分区天数, 即默认从开始日期算起, 分隔 2 天一个分区

# (3) 重启 Mycat
# (4) 访问Mycat创建表
# 用户信息表
CREATE TABLE login_info
(
    `id`          INT AUTO_INCREMENT comment '编号',
    `user_id`     INT comment '用户编号',
    `login_date`  date comment '登录日期',
    PRIMARY KEY(id)
);
# (6) 插入数据
INSERT INTO login_info(id,user_id,login_date) VALUES (1,101,'2019-01-01');
INSERT INTO login_info(id,user_id,login_date) VALUES (2,102,'2019-01-02');
INSERT INTO login_info(id,user_id,login_date) VALUES (3,103,'2019-01-03');
INSERT INTO login_info(id,user_id,login_date) VALUES (4,104,'2019-01-04');
INSERT INTO login_info(id,user_id,login_date) VALUES (5,103,'2019-01-05');
INSERT INTO login_info(id,user_id,login_date) VALUES (6,104,'2019-01-06');
# (7) 查询Mycat、dn1、dn2可以看到数据分片效果
```



## 5.4 全局序列

在实现分库分表的情况下，数据库自增主键已无法保证自增主键的全局唯一。为此，Mycat 提供了全局 sequence，并且提供了包含本地配置和数据库配置等多种实现方式

### 1、本地文件

此方式 Mycat 将 sequence 配置到文件中，当使用到 sequence 中的配置后，Mycat 会更改 classpath 中的 sequence\_conf.properties 文件中 sequence 当前的值。

① 优点：本地加载，读取速度较快

更多 [Java](#) -[大数据](#) -[前端](#) -[python](#) 人工智能资料下载，可百度访问：[尚硅谷官网](#)

② 缺点：抗风险能力差，Mycat 所在主机宕机后，无法读取本地文件。

## 2、数据库方式

利用数据库一个表 来进行计数累加。但是并不是每次生成序列都读写数据库，这样效率太低。

Mycat 会预加载一部分号段到 Mycat 的内存中，这样大部分读写序列都是在内存中完成的。

如果内存中的号段用完了 Mycat 会再向数据库要一次。

问：那如果 Mycat 崩溃了，那内存中的序列岂不是都没了？

是的。如果是这样，那么 Mycat 启动后会向数据库申请新的号段，原有号段会弃用。

也就是说如果 Mycat 重启，那么损失是当前的号段没用完的号码，但是不会因此出现主键重复

### ① 建库序列脚本

```
#在 dn1 上创建全局序列表
CREATE TABLE MYCAT_SEQUENCE (NAME VARCHAR(50) NOT NULL,current_value INT NOT
NULL,increment INT NOT NULL DEFAULT 100, PRIMARY KEY(NAME)) ENGINE=INNODB;

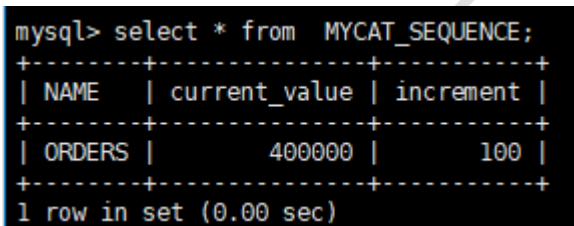
#创建全局序列所需函数
DELIMITER $$
CREATE FUNCTION mycat_seq_currval(seq_name VARCHAR(50)) RETURNS VARCHAR(64)
DETERMINISTIC
BEGIN
DECLARE retval VARCHAR(64);
SET retval="-999999999,null";
SELECT CONCAT(CAST(current_value AS CHAR),",",CAST(increment AS CHAR)) INTO retval FROM
MYCAT_SEQUENCE WHERE NAME = seq_name;
RETURN retval;
END $$
DELIMITER ;

DELIMITER $$
CREATE FUNCTION mycat_seq_setval(seq_name VARCHAR(50),VALUE INTEGER) RETURNS
VARCHAR(64)
DETERMINISTIC
BEGIN
UPDATE MYCAT_SEQUENCE
```

```
SET current_value = VALUE
WHERE NAME = seq_name;
RETURN mycat_seq_currval(seq_name);
END $$
DELIMITER ;

DELIMITER $$
CREATE FUNCTION mycat_seq_nextval(seq_name VARCHAR(50)) RETURNS VARCHAR(64)
DETERMINISTIC
BEGIN
UPDATE MYCAT_SEQUENCE
SET current_value = current_value + increment WHERE NAME = seq_name;
RETURN mycat_seq_currval(seq_name);
END $$
DELIMITER ;

#初始化序列表记录
INSERT INTO MYCAT_SEQUENCE(NAME,current_value,increment) VALUES ('ORDERS', 400000,
100);
```



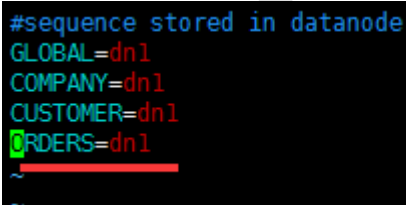
```
mysql> select * from MYCAT_SEQUENCE;
+-----+-----+-----+
| NAME   | current_value | increment |
+-----+-----+-----+
| ORDERS |          400000 |          100 |
+-----+-----+-----+
1 row in set (0.00 sec)
```

## ② 修改 Mycat 配置

#修改sequence\_db\_conf.properties

vim sequence\_db\_conf.properties

#意思是 ORDERS这个序列在dn1这个节点上，具体dn1节点是哪台机器，请参考schema.xml



```
#sequence stored in datanode
GLOBAL=dn1
COMPANY=dn1
CUSTOMER=dn1
ORDERS=dn1
~
~
```

#修改server.xml

vim server.xml

#全局序列类型：0-本地文件，1-数据库方式，2-时间戳方式。此处应该修改成1。

```
<property name="sequenceHandlerType">1</property>  
<property name="useCompression">1</property>--> <!--1为开启mys
```

#重启Mycat

### ③ 验证全局序列

#登录 Mycat，插入数据

```
insert into orders(id,amount,customer_id,order_type) values(next value for  
MYCATSEQ_ORDERS,1000,101,102);
```

#查询数据

```
mysql> insert into orders(id,amount,customer_id,order_type) values(next  
t value for MYCATSEQ_ORDERS,1000,101,102);  
Query OK, 1 row affected (0.00 sec)  
  
mysql> select * from orders;  
+-----+-----+-----+-----+  
| id      | order_type | customer_id | amount      |  
+-----+-----+-----+-----+  
| 3       | 101       | 101         | 120000.00   |  
| 4       | 101       | 101         | 103000.00   |  
| 5       | 102       | 101         | 100400.00   |  
| 400100  | 102       | 101         | 1000.00     |  
| 400101  | 102       | 101         | 1000.00     |  
| 1       | 101       | 100         | 100100.00   |  
| 2       | 101       | 100         | 100300.00   |  
| 6       | 102       | 100         | 100020.00   |  
+-----+-----+-----+-----+  
8 rows in set (0.08 sec)
```

#重启Mycat后，再次插入数据，再查询

```
mysql> insert into orders(id,amount,customer_id,order_type) values(next  
t value for MYCATSEQ_ORDERS,1000,101,102);  
Query OK, 1 row affected (0.14 sec)  
  
mysql> select * from orders;  
+-----+-----+-----+-----+  
| id      | order_type | customer_id | amount  |  
+-----+-----+-----+-----+  
| 1       | 101       | 100         | 100100.00 |  
| 2       | 101       | 100         | 100300.00 |  
| 6       | 102       | 100         | 100020.00 |  
| 3       | 101       | 101         | 120000.00 |  
| 4       | 101       | 101         | 103000.00 |  
| 5       | 102       | 101         | 100400.00 |  
| 400100  | 102       | 101         | 1000.00   |  
| 400101  | 102       | 101         | 1000.00   |  
| 400200  | 102       | 101         | 1000.00   |  
+-----+-----+-----+-----+  
9 rows in set (0.07 sec)
```

### 3、时间戳方式

全局序列ID= 64 位二进制 (42(毫秒)+5(机器 ID)+5(业务编码)+12(重复累加) 换算成十进制为 18 位数的 long 类型，每毫秒可以并发 12 位二进制的累加。

- ① 优点：配置简单
- ② 缺点：18 位 ID 过长

### 4、自主生成全局序列

可在 java 项目里自己生成全局序列，如下：

- ① 根据业务逻辑组合
- ② 可以利用 redis 的单线程原子性 incr 来生成序列

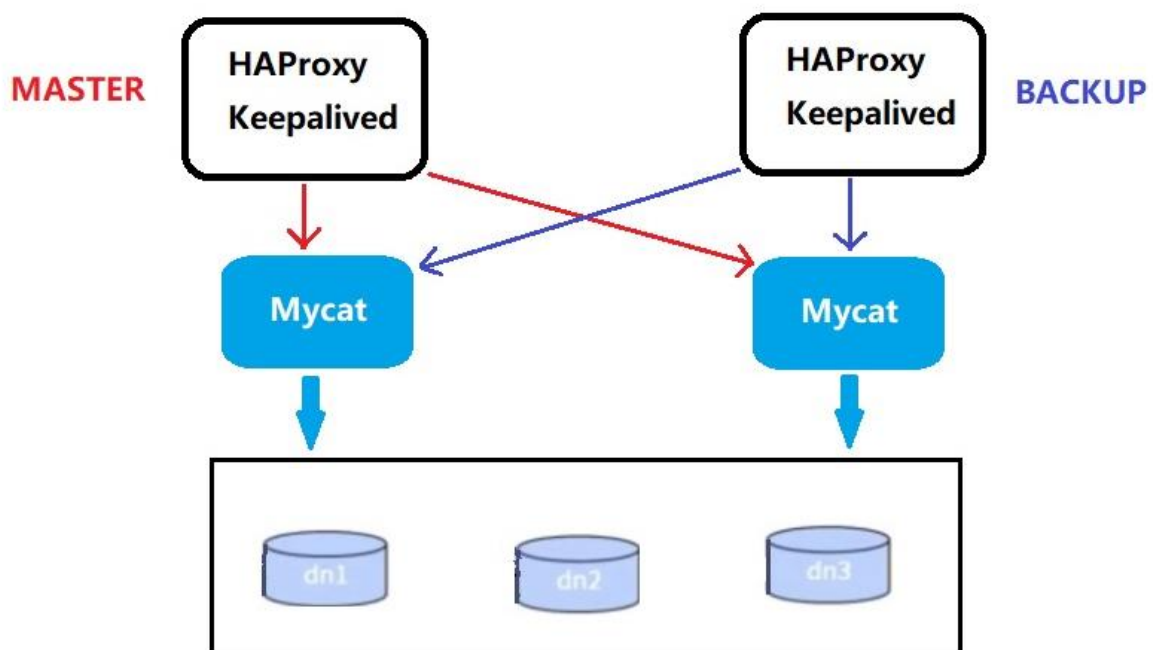
但，自主生成需要单独在工程中用 java 代码实现，还是推荐使用 Mycat 自带全局序列。

## 第六章 基于 HA 机制的 Mycat 高可用

在实际项目中，Mycat 服务也需要考虑高可用性，如果 Mycat 所在服务器出现宕机，或 Mycat 服务故障，需要有备机提供服务，需要考虑 Mycat 集群。

## 6.1 高可用方案

我们可以使用 HAProxy + Keepalived 配合两台 Mycat 搭起 Mycat 集群 实现高可用性。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。



编号	角色	IP 地址	机器名
1	Mycat1	192.168.140.128	host79.atguigu
2	Mycat2	192.168.140.127	host80.atguigu
3	HAProxy (master)	192.168.140.126	host81.atguigu
4	Keepalived (master)	192.168.140.126	host81.atguigu
5	HAProxy (backup)	192.168.140.125	host82.atguigu
6	Keepalived (backup)	192.168.140.125	host82.atguigu

## 6.2 安装配置 HAProxy

### 1、安装 HAProxy

#1准备好HAProxy安装包，传到/opt目录下

#2解压到/usr/local/src

```
tar -zxvf haproxy-1.5.18.tar.gz -C /usr/local/src
```

#3进入解压后的目录，查看内核版本，进行编译

```
cd /usr/local/src/haproxy-1.5.18
```

```
uname -r
```

```
make TARGET=linux310 PREFIX=/usr/local/haproxy ARCH=x86_64
```

# TARGET=linux310，内核版本，使用uname -r查看内核，如：3.10.0-514.el7，此时该参数就为linux310；

#ARCH=x86\_64，系统位数；

#PREFIX=/usr/local/haprpxy #/usr/local/haprpxy，为haprpxy安装路径。

#4编译完成后，进行安装

```
make install PREFIX=/usr/local/haproxy
```

#5安装完成后，创建目录、创建HAProxy配置文件

```
mkdir -p /usr/data/haproxy/
```

```
vim /usr/local/haproxy/haproxy.conf
```

#6向配置文件中插入以下配置信息,并保存

global

```
log 127.0.0.1    local0
#log 127.0.0.1    local1 notice
#log loghost     local0 info
maxconn 4096
chroot /usr/local/haproxy
pidfile /usr/data/haproxy/haproxy.pid
uid 99
gid 99
daemon
#debug
#quiet
```

defaults

```
log        global
```



```
mode    tcp
option  abortonclose
option  redispatch
retries 3
maxconn 2000
timeout connect 5000
timeout client  50000
timeout server  50000

listen proxy_status
    bind :48066
    mode tcp
    balance roundrobin
    server mycat_1 192.168.140.128:8066 check inter 10s
    server mycat_2 192.168.140.127:8066 check inter 10s

frontend admin_stats
    bind :7777
    mode http
    stats enable
    option httplog
    maxconn 10
    stats refresh 30s
    stats uri /admin
    stats auth admin:123123
    stats hide-version
    stats admin if TRUE
```

## 2、启动验证

```
#1启动HAProxy
/usr/local/haproxy/sbin/haproxy -f /usr/local/haproxy/haproxy.conf



#2查看HAProxy进程
ps -ef|grep haproxy

#3打开浏览器访问
http://192.168.140.125:7777/admin
```

#在弹出框输入用户名：admin密码：123123

#如果Mycat主备机均已启动，则可以看到如下图

Note: NULL / UNKNOWN = 0 or with read-balancing disabled.

proxy_status		Queue			Session rate			Sessions				Bytes		Denied		Errors			Warnings		Server									
		Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	Last	In	Out	Req	Conn	Resp	Retr	Redis	Status	LastChk	Wght	Act	Bck	Chk	Dwn	Downtme	Thrtle	
Frontend					0	0	-	0	0	2 000	0					0	0	0	0	0	OPEN									
	mycat_1	0	0	-	0	0	-	0	0	-	0	0	?	0	0	0	0	0	0	0	2m32s UP	L4OK in 0ms	1	Y	-	0	0	0s	-	
	mycat_2	0	0	-	0	0	-	0	0	-	0	0	?	0	0	0	0	0	0	0	2m32s UP	L4OK in 0ms	1	Y	-	0	0	0s	-	
Backend		0	0		0	0		0	0	200	0	0	?	0	0	0	0	0	0	0	2m32s UP		2	2	0		0	0s		

#4验证负载均衡，通过HAProxy访问Mycat

mysql -umycat -p123456 -h 192.168.140.126 -P 48066

## 6.3 配置 Keepalived

### 1、安装 Keepalived

#1准备好Keepalived安装包，传到/opt目录下

#2解压到/usr/local/src

tar -zxvf keepalived-1.4.2.tar.gz -C /usr/local/src

#3安装依赖插件

yum install -y gcc openssl-devel popt-devel

#3进入解压后的目录，进行配置，进行编译

cd /usr/local/src/keepalived-1.4.2

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

#4进行编译，完成后进行安装

make && make install

#5运行前配置

cp /usr/local/src/keepalived-1.4.2/keepalived/etc/init.d/keepalived /etc/init.d/

mkdir /etc/keepalived

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

cp /usr/local/src/keepalived-1.4.2/keepalived/etc/sysconfig/keepalived /etc/sysconfig/

cp /usr/local/keepalived/sbin/keepalived /usr/sbin/

#6修改配置文件

vim /etc/keepalived/keepalived.conf

#修改内容如下

! Configuration File for keepalived

```
global_defs {
    notification_email {
        xlcocoon@foxmail.com
    }
    notification_email_from keepalived@showjoy.com
    smtp_server 127.0.0.1
    smtp_connect_timeout 30
    router_id LVS_DEVEL
    vrrp_skip_check_adv_addr
    vrrp_garp_interval 0
    vrrp_gna_interval 0
}

vrrp_instance VI_1 {
    #主机配MASTER，备机配BACKUP
    state MASTER
    #所在机器网卡
    interface ens33
    virtual_router_id 51
    #数值越大优先级越高
    priority 100
    advert_int 1
    authentication {
        auth_type PASS
        auth_pass 1111
    }
    virtual_ipaddress {
        #虚拟IP
        192.168.140.200
    }
}

virtual_server 192.168.140.200 48066 {
    delay_loop 6
    lb_algo rr
    lb_kind NAT
    persistence_timeout 50
    protocol TCP
}
```

```
real_server 192.168.140.125 48066 {  
    weight 1  
    TCP_CHECK {  
        connect_timeout 3  
        retry 3  
        delay_before_retry 3  
    }  
}  
real_server 192.168.140.126 48600 {  
    weight 1  
    TCP_CHECK {  
        connect_timeout 3  
        nb_get_retry 3  
        delay_before_retry 3  
    }  
}  
}
```

## 2、启动验证

#1启动Keepalived

service keepalived start

#2登录验证

mysql -umycat -p123456 -h 192.168.140.200 -P 48066

## 6.4 测试高可用

### 1、测试步骤

#1关闭mycat

#2通过虚拟ip查询数据

mysql -umycat -p123456 -h 192.168.140.200 -P 48066

## 第七章 Mycat 安全设置

### 7.1 权限配置

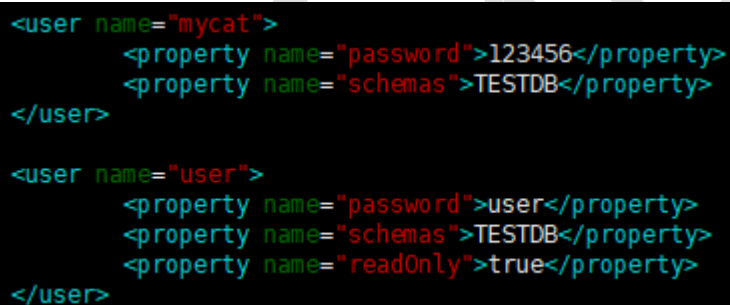
#### 1、user 标签权限控制

目前 Mycat 对于中间件的连接控制并没有做太复杂的控制，目前只做了中间件逻辑库级别的读写权限控制。是通过 server.xml 的 user 标签进行配置。

#server.xml配置文件user部分

```
<user name="mycat">
    <property name="password">123456</property>
    <property name="schemas">TESTDB</property>
</user>
<user name="user">
    <property name="password">user</property>
    <property name="schemas">TESTDB</property>
    <property name="readOnly">true</property>
</user>
```

#如下图



```
<user name="mycat">
  <property name="password">123456</property>
  <property name="schemas">TESTDB</property>
</user>

<user name="user">
  <property name="password">user</property>
  <property name="schemas">TESTDB</property>
  <property name="readOnly">true</property>
</user>
```

配置说明

标签属性	说明
name	应用连接中间件逻辑库的用户名
password	该用户对应的密码
TESTDB	应用当前连接的逻辑库中所对应的逻辑表。schemas 中可以配置一个或多个
readOnly	应用连接中间件逻辑库所具有的权限。true 为只读，false 为读写都有，默认为 false

## 测试案例

## #测试案例一

- # 使用user用户，权限为只读（readOnly: true）
- # 验证是否可以查询出数据，验证是否可以写入数据

#1、用user用户登录，运行命令如下：

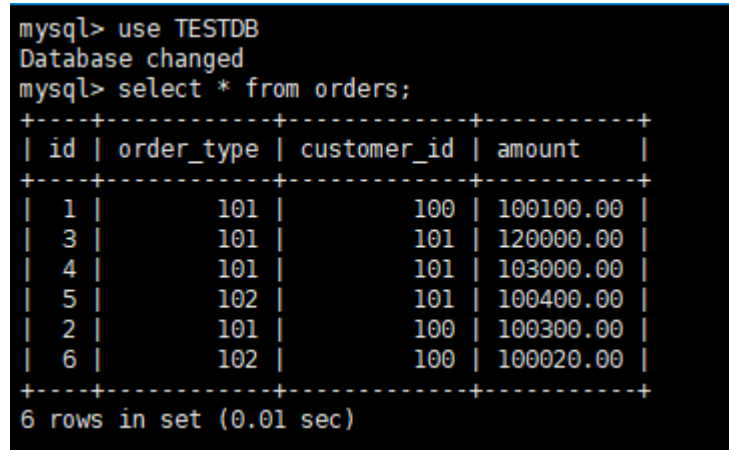
```
mysql -uuser -puser -h 192.168.140.128 -P8066
```

#2、切换到TESTDB数据库，查询orders表数据，如下：

```
use TESTDB
```

```
select * from orders;
```

#3、可以查询到数据，如下图

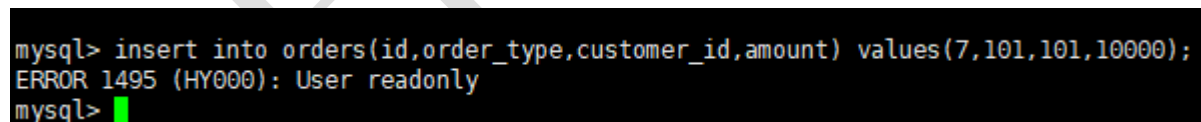


```
mysql> use TESTDB
Database changed
mysql> select * from orders;
+----+-----+-----+-----+
| id | order_type | customer_id | amount |
+----+-----+-----+-----+
| 1  | 101       | 100        | 100100.00 |
| 3  | 101       | 101        | 120000.00 |
| 4  | 101       | 101        | 103000.00 |
| 5  | 102       | 101        | 100400.00 |
| 2  | 101       | 100        | 100300.00 |
| 6  | 102       | 100        | 100020.00 |
+----+-----+-----+-----+
6 rows in set (0.01 sec)
```

#4、执行插入数据sql，如下：

```
insert into orders(id,order_type,customer_id,amount) values(7,101,101,10000);
```

#5、可看到运行结果，插入失败，只有只读权限，如下图：



```
mysql> insert into orders(id,order_type,customer_id,amount) values(7,101,101,10000);
ERROR 1495 (HY000): User readonly
mysql>
```

## #测试案例二

- # 使用mycat用户，权限为可读写（readOnly: false）
- # 验证是否可以查询出数据，验证是否可以写入数据

#1、用mycat用户登录，运行命令如下：

```
mysql -umycat -p123456 -h 192.168.140.128 -P8066
```

#2、切换到TESTDB数据库，查询orders表数据，如下：

```
use TESTDB
```

```
select * from orders;
```

#3、可以查询到数据，如下图

```
mysql> use TESTDB
Database changed
mysql> select * from orders;
+-----+-----+-----+-----+
| id | order_type | customer_id | amount |
+-----+-----+-----+-----+
| 1 | 101 | 100 | 100100.00 |
| 2 | 101 | 100 | 100300.00 |
| 6 | 102 | 100 | 100020.00 |
| 3 | 101 | 101 | 120000.00 |
| 4 | 101 | 101 | 103000.00 |
| 5 | 102 | 101 | 100400.00 |
+-----+-----+-----+-----+
6 rows in set (0.01 sec)
```

#4、执行插入数据sql，如下：

```
insert into orders(id,order_type,customer_id,amount) values(7,101,101,10000);
```

#5、可看到运行结果，插入成功，如下图：

```
mysql> insert into orders(id,order_type,customer_id,amount) values(7,101,101,10000);
Query OK, 1 row affected (0.01 sec)
```

## 2、privileges 标签权限控制

在 user 标签下的 privileges 标签可以对逻辑库（schema）、表（table）进行精细化的 DML 权限控制。

privileges 标签下的 check 属性，如为 true 开启权限检查，为 false 不开启，默认为 false。

由于 Mycat 一个用户的 schemas 属性可配置多个逻辑库（schema），所以 privileges 的下级节点 schema 节点同样可配置多个，对多库多表进行细粒度的 DML 权限控制。

```
#server.xml配置文件privileges部分
#配置orders表没有增删改查权限
<user name="mycat">
  <property name="password">123456</property>
  <property name="schemas">TESTDB</property>
  <!-- 表级 DML 权限设置 -->
  <privileges check="true">
    <schema name="TESTDB" dml="1111" >
      <table name="orders" dml="0000"></table>
      <!--<table name="tb02" dml="1111"></table>-->
    </schema>
  </privileges>
</user>
```

#如下图

```
<user name="mycat">
  <property name="password">123456</property>
  <property name="schemas">TESTDB</property>
  <!-- 表级 DML 权限设置 -->
  <privileges check="true">
    <schema name="TESTDB" dml="1111" >
      <table name="orders" dml="0000"></table>
      <!--<table name="tb02" dml="1111"></table>-->
    </schema>
  </privileges>
</user>
```

配置说明

DML 权限	增加 (insert)	更新 (update)	查询 (select)	删除 (select)
0000	禁止	禁止	禁止	禁止
0010	禁止	禁止	可以	禁止
1110	可以	禁止	禁止	禁止
1111	可以	可以	可以	可以

测试案例

#测试案例一

# 使用mycat用户， privileges配置orders表权限为禁止增删改查 (dml="0000")

# 验证是否可以查询出数据， 验证是否可以写入数据

#1、重启mycat， 用mycat用户登录， 运行命令如下：

mysql -umycat -p123456 -h 192.168.140.128 -P8066

#2、切换到TESTDB数据库， 查询orders表数据， 如下：

use TESTDB

select \* from orders;

#3、禁止该用户查询数据， 如下图

```
mysql> use TESTDB
Database changed
mysql> select * from orders;
ERROR 3012 (HY000): The statement DML privilege check is not passed, reject for user 'mycat'
mysql>
```

#4、执行插入数据sql， 如下：

insert into orders(id,order\_type,customer\_id,amount) values(8,101,101,10000);

#5、可看到运行结果， 禁止该用户插入数据， 如下图：

```
mysql> insert into orders(id,order_type,customer_id,amount) values(8,101,101,10000);
ERROR 3012 (HY000): The statement DML privilege check is not passed, reject for user 'mycat'
mysql>
```



## #测试案例二

- # 使用mycat用户， privileges配置orders表权限为可以增删改查（dml="1111"）
- # 验证是否可以查询出数据，验证是否可以写入数据

#1、重启mycat，用mycat用户登录，运行命令如下：

```
mysql -umycat -p123456 -h 192.168.140.128 -P8066
```

#2、切换到TESTDB数据库，查询orders表数据，如下：

```
use TESTDB
```

```
select * from orders;
```

#3、可以查询到数据，如下图

```
mysql> use TESTDB
Database changed
mysql> select * from orders;
+----+-----+-----+-----+
| id | order_type | customer_id | amount |
+----+-----+-----+-----+
| 1 | 101 | 100 | 100100.00 |
| 2 | 101 | 100 | 100300.00 |
| 6 | 102 | 100 | 100020.00 |
| 3 | 101 | 101 | 120000.00 |
| 4 | 101 | 101 | 103000.00 |
| 5 | 102 | 101 | 100400.00 |
| 7 | 101 | 101 | 10000.00 |
+----+-----+-----+-----+
7 rows in set (0.00 sec)
```

#4、执行插入数据sql，如下：

```
insert into orders(id,order_type,customer_id,amount) values(8,101,101,10000);
```

#5、可看到运行结果，插入成功，如下图：

```
mysql> insert into orders(id,order_type,customer_id,amount) values(8,101,101,10000);
Query OK, 1 row affected (0.01 sec)
```

#4、执行插入数据sql，如下：

```
delete from orders where id in (7,8);
```

#5、可看到运行结果，插入成功，如下图：

```
mysql> delete from orders where id in (7,8);
Query OK, 2 rows affected (0.00 sec)
```

## 7.2 SQL 拦截

firewall 标签用来定义防火墙；firewall 下 whitehost 标签用来定义 IP 白名单，blacklist 用来定义 SQL 黑名单。

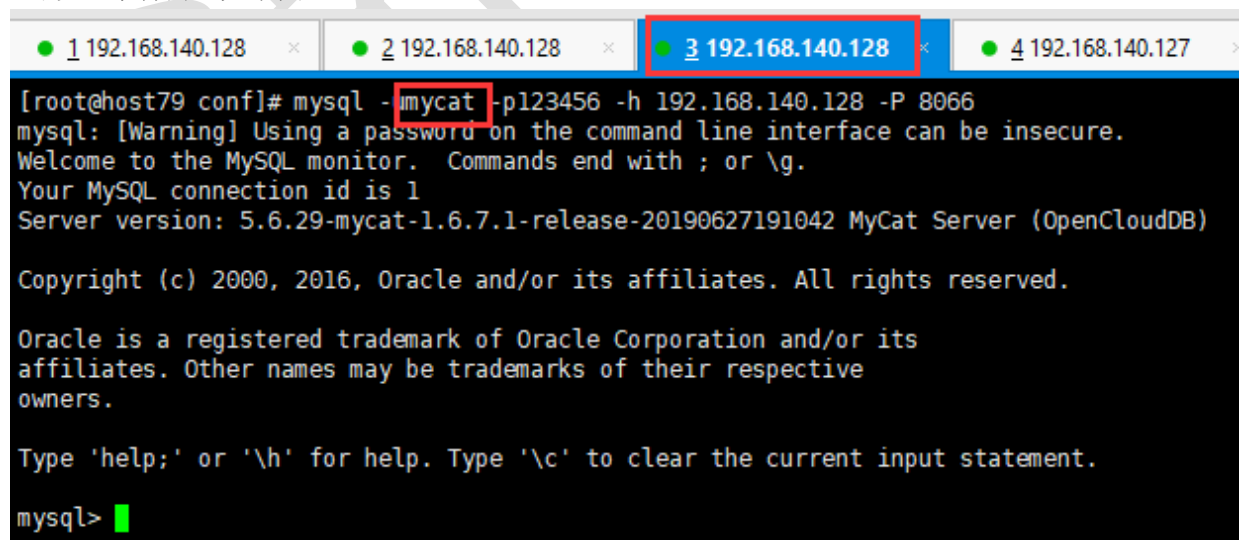
### 1、白名单

可以通过设置白名单，实现某主机某用户可以访问 Mycat，而其他主机用户禁止访问。

```
#设置白名单
#server.xml配置文件firewall标签
#配置只有192.168.140.128主机可以通过mycat用户访问
<firewall>
    <whitehost>
        <host host="192.168.140.128" user="mycat"/>
    </whitehost>
</firewall>
#如下图
```

```
<firewall>
  <whitehost>
    <host host="192.168.140.128" user="mycat"/>
  </whitehost>
</firewall>
```

```
#重启Mycat后，192.168.140.128主机使用mycat用户访问
mysql -umycat -p123456 -h 192.168.140.128 -P 8066
#可以正常访问，如下图
```

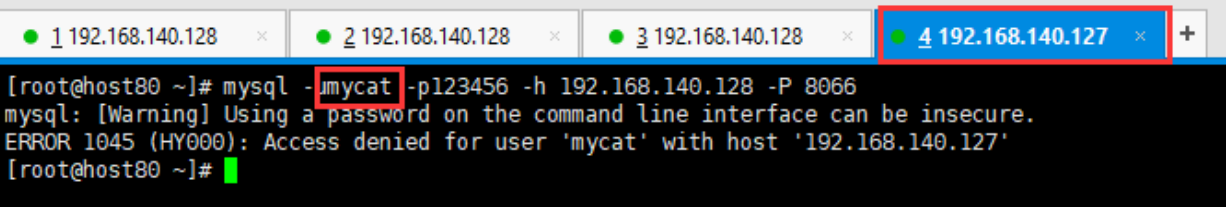


The screenshot shows a terminal window with four tabs at the top, all pointing to the IP 192.168.140.128. The active tab is labeled '3 192.168.140.128'. The terminal content shows the command `mysql -umycat -p123456 -h 192.168.140.128 -P 8066` being executed. The output includes a warning about using passwords on the command line, a welcome message to the MySQL monitor, the connection ID (1), the server version (5.6.29-mycat-1.6.7.1-release-20190627191042 MyCat Server (OpenCloudDB)), and copyright information. The prompt `mysql>` is visible at the bottom.

#在此主机换user用户访问，禁止访问

```
[root@host79 conf]# mysql -user -p123456 -h 192.168.140.128 -P 8066
mysql: [Warning] Using a password on the command line interface can be insecure.
ERROR 1045 (HY000): Access denied for user 'user' with host '192.168.140.128'
[root@host79 conf]#
```

#在192.168.140.127主机用mycat用户访问，禁止访问



```
[root@host80 ~]# mysql -umycat -p123456 -h 192.168.140.128 -P 8066
mysql: [Warning] Using a password on the command line interface can be insecure.
ERROR 1045 (HY000): Access denied for user 'mycat' with host '192.168.140.127'
[root@host80 ~]#
```

## 2、黑名单

可以通过设置黑名单，实现 Mycat 对具体 SQL 操作的拦截，如增删改查等操作的拦截。

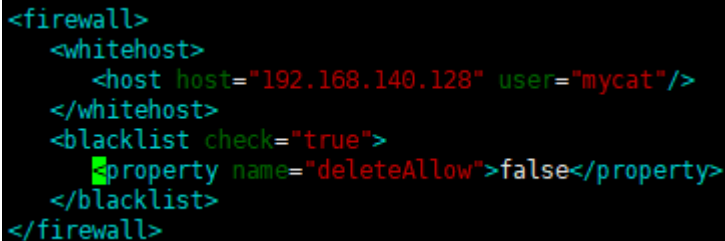
#设置黑名单

#server.xml配置文件firewall标签

#配置禁止mycat用户进行删除操作

```
<firewall>
    <whitehost>
        <host host="192.168.140.128" user="mycat"/>
    </whitehost>
    <blacklist check="true">
        <property name="deleteAllow">false</property>
    </blacklist>
</firewall>
```

#如下图



```
<firewall>
    <whitehost>
        <host host="192.168.140.128" user="mycat"/>
    </whitehost>
    <blacklist check="true">
        <property name="deleteAllow">false</property>
    </blacklist>
</firewall>
```

#重启Mycat后，192.168.140.128主机使用mycat用户访问

mysql -umycat -p123456 -h 192.168.140.128 -P 8066

#可以正常访问，如下图

```
1 192.168.140.128 x 2 192.168.140.128 x 3 192.168.140.128 x 4 192.168.140.127 >
[root@host79 conf]# mysql -mycat -p123456 -h 192.168.140.128 -P 8066
mysql: [Warning] Using a password on the command line interface can be insecure.
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 1
Server version: 5.6.29-mycat-1.6.7.1-release-20190627191042 MyCat Server (OpenCloudDB)

Copyright (c) 2000, 2016, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>
```

#切换TESTDB数据库后，执行删除数据语句  
delete from orders where id=7;  
#运行后发现已禁止删除数据，如下图

```
mysql> delete from orders where id=7;
ERROR 3012 (HY000): The statement is unsafe SQL, reject for user 'mycat'
```

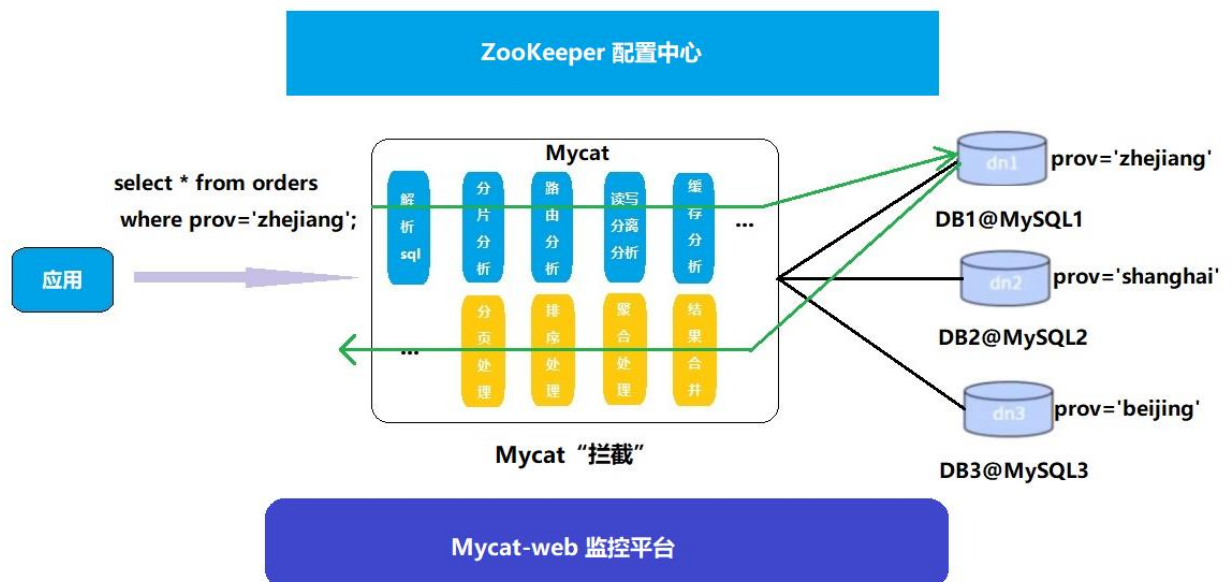
可以设置的黑名单 SQL 拦截功能列表

配置项	缺省值	描述
selectAllow	true	是否允许执行 SELECT 语句
deleteAllow	true	是否允许执行 DELETE 语句
updateAllow	true	是否允许执行 UPDATE 语句
insertAllow	true	是否允许执行 INSERT 语句
createTableAllow	true	是否允许创建表
setAllow	true	是否允许使用 SET 语法
alterTableAllow	true	是否允许执行 Alter Table 语句
dropTableAllow	true	是否允许修改表
commitAllow	true	是否允许执行 commit 操作
rollbackAllow	true	是否允许执行 roll back 操作

## 第八章 Mycat 监控工具

### 8.1 Mycat-web 简介

Mycat-web 是 Mycat 可视化运维的管理和监控平台，弥补了 Mycat 在监控上的空白。帮 Mycat 分担统计任务和配置管理任务。Mycat-web 引入了 ZooKeeper 作为配置中心，可以管理多个节点。Mycat-web 主要管理和监控 Mycat 的流量、连接、活动线程和内存等，具备 IP 白名单、邮件告警等模块，还可以统计 SQL 并分析慢 SQL 和高频 SQL 等。为优化 SQL 提供依据。



### 8.2 Mycat-web 配置使用

#### 1、ZooKeeper 安装

安装步骤如下：

#1 下载安装包<http://zookeeper.apache.org/>

#2 安装包拷贝到Linux系统/opt目录下，并解压

```
tar -zxvf zookeeper-3.4.11.tar.gz
```

```
[root@host79 opt]# tar -zxvf zookeeper-3.4.11.tar.gz
```

#3 进入ZooKeeper解压后的配置目录（conf），复制配置文件并改名

```
cp zoo_sample.cfg zoo.cfg
```

#4 进入ZooKeeper的命令目录（bin），运行启动命令

更多 [Java](#) -[大数据](#) -[前端](#) -[python](#) 人工智能资料下载，可百度访问：尚硅谷官网

```
./zkServer.sh start
```

```
[root@host79 bin]# ./zkServer.sh start
ZooKeeper JMX enabled by default
Using config: /myzookeeper/zookeeper-3.4.11/bin/../conf/zoo.cfg
Starting zookeeper ... STARTED
```

#5 ZooKeeper服务端端口为2181，查看服务已经启动

```
netstat -ant | grep 2181
```

```
[root@host79 bin]# netstat -ant | grep 2181
tcp6      0      0 :::2181          :::*              LISTEN
```

## 2、Mycat-web 安装

安装步骤如下：

#1 下载安装包<http://www.mycat.io/>

#2 安装包拷贝到Linux系统/opt目录下，并解压

```
tar -zxvf Mycat-web-1.0-SNAPSHOT-20170102153329-linux.tar.gz
```

#3 拷贝mycat-web文件夹到/usr/local目录下

```
cp -r mycat-web /usr/local
```

#4 进入mycat-web的目录下运行启动命令

```
cd /usr/local/mycat-web/
```

```
./start.sh &
```

```
[root@host80 mycat-web]# cd /usr/local/mycat-web/
[root@host80 mycat-web]# ll
总用量 40
drwxr-xr-x. 2 root root 4096 12月 18 10:58 etc
drwxr-xr-x. 3 root root 4096 12月 18 10:58 lib
drwxr-xr-x. 7 root root 4096 12月 18 10:58 mycat-web
-rwxr-xr-x. 1 root root 116 12月 18 10:58 readme.txt
-rwxr-xr-x. 1 root root 17125 12月 18 10:58 start.jar
-rwxr-xr-x. 1 root root 381 12月 18 10:58 start.sh
[root@host80 mycat-web]# ./start.sh &
[1] 3229
[root@host80 mycat-web]# nohup: 忽略输入并把输出追加到"nohup.out"
^C
[root@host80 mycat-web]#
```

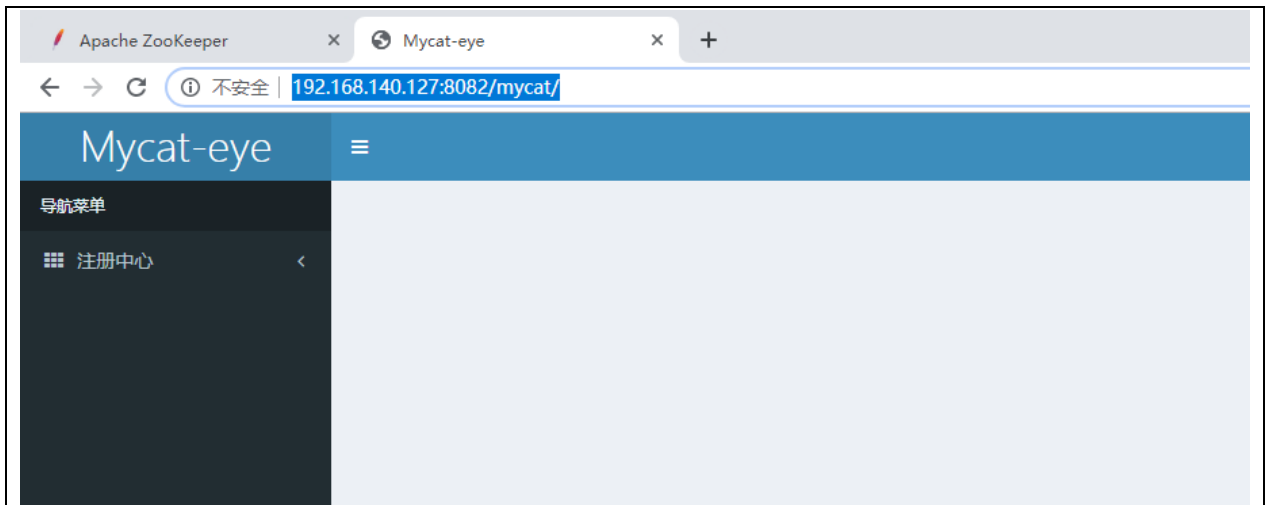
#5 Mycat-web服务端端口为8082，查看服务已经启动

```
netstat -ant | grep 8082
```

```
[root@host80 mycat-web]# netstat -ant | grep 8082
tcp6      0      0 :::8082          :::*              LISTEN
```

#6 通过地址访问服务

```
http://192.168.140.127:8082/mycat/
```



### 3、Mycat-web 配置

安装步骤如下：

#1 先在注册中心配置ZooKeeper地址，配置后刷新页面，可见



#2 新增Mycat监控实例

Mycat-配置

mycat服务管理

mycat-VM管理

mysql管理

mycat系统参数

IP白名单

mycat日志管理

网络拓扑图

邮件告警

Mycat-监控

SQL-监控

SQL-上线

Mycat Zone

### Mycat配置管理

**Mycat名称(必须为英文哦):**

**IP地址:**

**管理端口:**

**服务端口:**

**数据库名称:**

**用户名:**

**密码:**

导航菜单

Mycat-配置

mycat服务管理

mycat-VM管理

mysql管理

mycat系统参数

IP白名单

mycat日志管理

网络拓扑图

邮件告警

Mycat-监控

SQL-监控

SQL-上线

Mycat Zone

### Mycat配置管理

Mycat-配置 > mycat服务管理

查询条件

**Mycat名称:**

查询结果

#	操作	Mycat名称	IP地址	管理端口	服务端口
1	<input type="button" value="修改"/> <input type="button" value="删除"/>	mycatone	192.168.140.128	9066	8066



## 8.3 Mycat 性能监控指标

在 Mycat-web 上可以进行 Mycat 性能监控，例如：内存分析、流量分析、连接分析、活动线程分析等等。

