

# MyCat 中间件

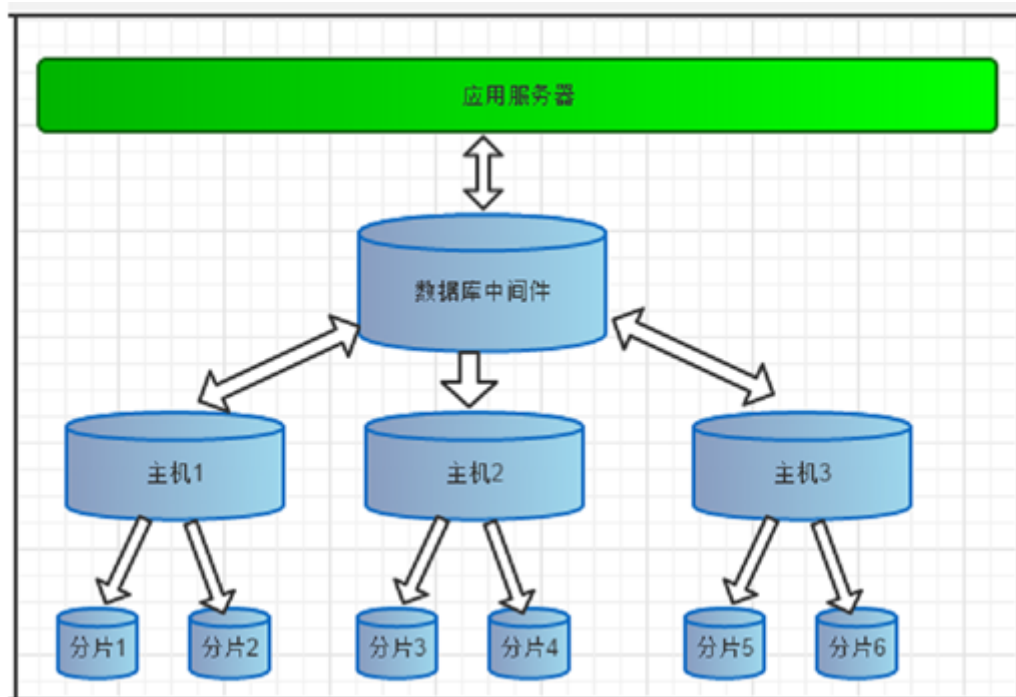
## 1. MyCat简介

Mycat 背后是阿里曾经开源的知名产品——Cobar。Cobar 的核心功能和优势是 MySQL 数据库分片，此产品曾经广为流传，据说最早的发起者对 Mysql 很精通，后来从阿里跳槽了，阿里随后开源的 Cobar，并维持到 2013 年年初，然后，就没有然后了。

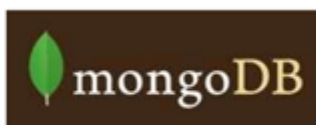
Cobar 的思路和实现路径的确不错。基于 Java 开发的，实现了 MySQL 公开的二进制传输协议，巧妙地将自己伪装成一个 MySQL Server，目前市面上绝大多数 MySQL 客户端工具和应用都能兼容。比自己实现一个新的数据库协议要明智的多，因为生态环境在哪里摆着。

Mycat 是基于 cobar 演变而来，对 cobar 的代码进行了彻底的重构，使用 NIO 重构了网络模块，并且优化了 Buffer 内核，增强了聚合，Join 等基本特性，同时兼容绝大多数数据库成为通用的数据库中间件。

简单的说，MyCAT就是：一个新颖的数据库中间件产品支持mysql集群，或者mariadb cluster，提供高可用性数据分片集群。你可以像使用mysql一样使用mycat。对于开发人员来说根本感觉不到mycat的存在。



MyCat支持的数据库：



MyCat 官网:

<http://www.mycat.org.cn/>

下载地址:

<https://github.com/MyCATapache/Mycat-download>

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

## 2. MyCat安装


### 2.1 版本要求

JDK: 要求jdk必须是1.7及以上版本


MySQL: 推荐mysql是5.5以上版本

### 2.2 MySQL 的安装及启动

1). 将MySQL的服务端和客户端安装包 (RPM) 上传到服务器



MySQL-client-5.5.49-1.linux2.6.i386.rpm



MySQL-server-5.5.49-1.linux2.6.i386.rpm

2). 查询之前是否安装过MySQL

```
rpm -qa | grep -i mysql
```

3). 卸载旧版本的MySQL

```
rpm -e --nodeps 查询到的软件名称
```

4). 安装服务端

```
rpm -ivh MySQL-server-5.5.49-1.linux2.6.i386.rpm
```

5). 安装客户端

```
rpm -ivh MySQL-client-5.5.49-1.linux2.6.i386.rpm
```

6). 启动MySQL服务

```
service mysql start
```

7). 登录MySQL

```
mysql -u root -p
```

8). 设置远程登录权限

```
GRANT ALL PRIVILEGES ON . TO 'root'@'%' IDENTIFIED BY 'itcast' WITH GRANT OPTION;
```

9). 在本地SQLyog 连接远程MySQL进行测试

### 2.3 MyCat 的安装及启动

1). 将 Mycat-server-1.6-RELEASE-20161028204710-linux.tar.gz 上传至服务器

2). 将压缩包解压缩。建议将mycat放到/usr/local/mycat目录下。

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

```
mv mycat /usr/local
```

3). 进入mycat目录的bin目录，启动mycat

```
./mycat start
```

4). 停止MyCat

```
./mycat stop
```

5). Mycat的默认端口号为：8066

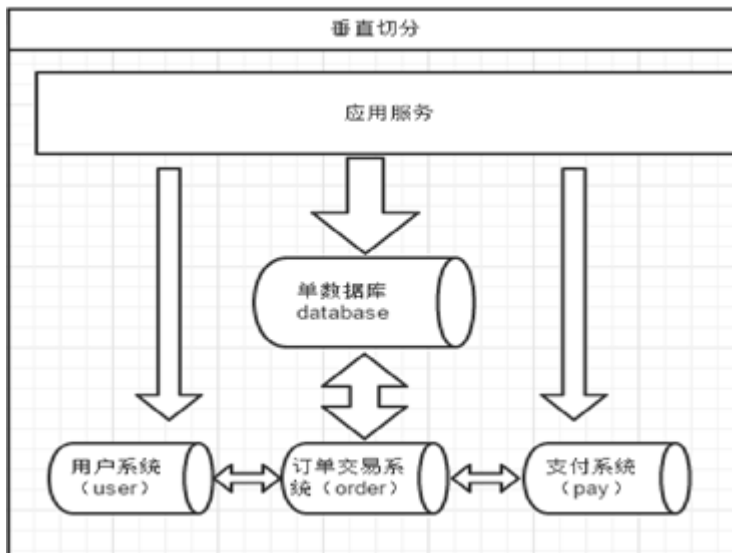
## 3. MyCat分片

### 3.1 什么是分片

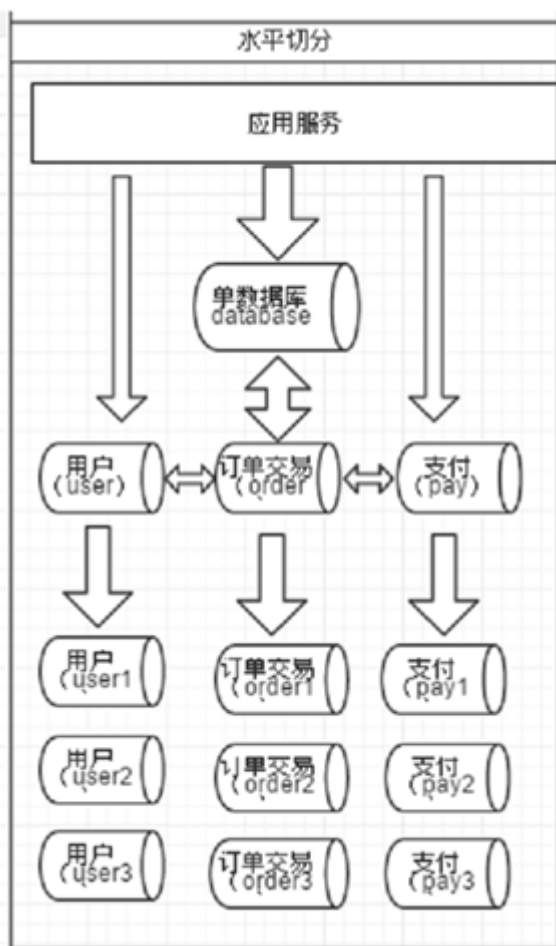
简单来说，就是指通过某种特定的条件，将我们存放在同一个数据库中的数据分散存放到多个数据库（主机）上面，以达到分散单台设备负载的效果。

数据的切分（Sharding）根据其切分规则的类型，可以分为两种切分模式。

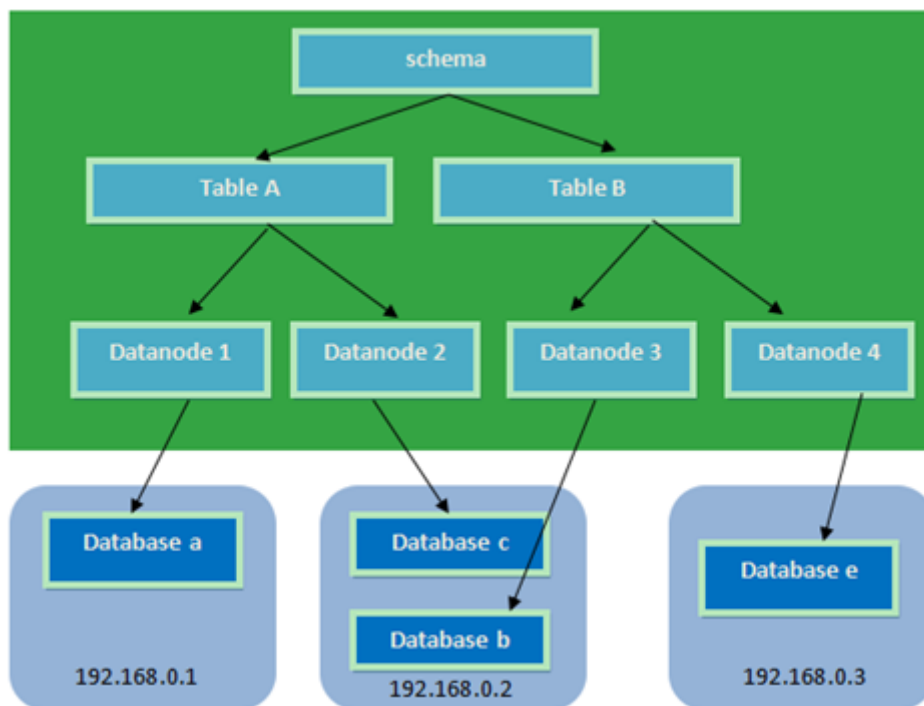
1). 一种是按照不同的表（或者Schema）来切分到不同的数据库（主机）之上，这种切分可以称之为数据的垂直（纵向）切分。



2). 另外一种则是根据表中的数据的逻辑关系，将同一个表中的数据按照某种条件拆分到多台数据库（主机）上面，这种切分称之为数据的水平（横向）切分。



MyCat 分片策略：



### 3.2 分片相关概念

#### 1). 逻辑库(schema):

前面一节讲了数据库中间件，通常对实际应用来说，并不需要知道中间件的存在，业务开发人员只需要知道数据库的概念，所以数据库中间件可以被看做是一个或多个数据库集群构成的逻辑库。

#### 2). 逻辑表 (table) :

既然有逻辑库，那么就会有逻辑表，分布式数据库中，对应用来说，读写数据的表就是逻辑表。逻辑表，可以是数据切分后，分布在一个或多个分片库中，也可以不做数据切分，不分片，只有一个表构成。

分片表：是指那些原有的很大数据的表，需要切分到多个数据库的表，这样，每个分片都有一部分数据，所有分片构成了完整的数据。总而言之就是需要进行分片的表。

#### 3). 非分片表:

一个数据库中并不是所有的表都很大，某些表是可以不用进行切分的，非分片是相对分片表来说的，就是那些不需要进行数据切分的表。

#### 4). 分片节点(dataNode):

数据切分后，一个大表被分到不同的分片数据库上面，每个表分片所在的数据库就是分片节点 (dataNode) 。

#### 5). 节点主机(dataHost):

数据切分后，每个分片节点 (dataNode) 不一定会独占一台机器，同一机器上面可以有多个分片数据库，这样一个或多个分片节点 (dataNode) 所在的机器就是节点主机 (dataHost) ,为了规避单节点主机并发数限制，尽量将读写压力高的分片节点 (dataNode) 均衡的放在不同的节点主机 (dataHost) 。

#### 6). 分片规则(rule):

前面讲了数据切分，一个大表被分成若干个分片表，就需要一定的规则，这样按照某种业务规则把数据分到某个分片的规则就是分片规则，数据切分选择合适的分片规则非常重要，将极大的避免后续数据处理的难度。

### 3.3 分片配置

#### 1). 配置schema.xml

schema.xml 作为MyCat中重要的配置文件之一，管理着MyCat的逻辑库、逻辑表以及对应的分片规则、DataNode以及DataSource。弄懂这些配置，是正确使用MyCat的前提。这里就一层层对该文件进行解析。

schema 标签用于定义MyCat实例中的逻辑库

Table 标签定义了MyCat中的逻辑表 rule用于指定分片规则，auto-sharding-long的分片规则是按ID值的范围进行分片 1-5000000 为第1片 5000001-10000000 为第2片.... 具体设置我们会在第5小节中讲解。

dataNode 标签定义了MyCat中的数据节点，也就是我们通常所说的数据分片。

dataHost标签在mycat逻辑库中也是作为最底层的标签存在，直接定义了具体的数据库实例、读写分离配置和心跳语句。

在服务器上创建3个数据库，分别是db1 db2 db3

修改schema.xml如下：

```
<?xml version="1.0"?>
<!DOCTYPE mycat:schema SYSTEM "schema.dtd">
<mycat:schema xmlns:mycat="http://org.opencloudb/">
  <schema name="ITCAST" checkSQLschema="false" sqlMaxLimit="100">
```

```

        <table name="tb_test" dataNode="dn1,dn2,dn3" rule="auto-sharding-long"
/>
    </schema>

    <dataNode name="dn1" dataHost="localhost1" database="db1" />
    <dataNode name="dn2" dataHost="localhost1" database="db2" />
    <dataNode name="dn3" dataHost="localhost1" database="db3" />

    <dataHost name="localhost1" maxCon="1000" minCon="10" balance="0"
        writeType="0" dbType="mysql" dbDriver="native" switchType="1"
slaveThreshold="100">
        <heartbeat>select user()</heartbeat>
        <writeHost host="hostM1" url="192.168.192.128:3306" user="root"
password="itcast"></writeHost>
    </dataHost>

</mycat:schema>

```

## 2). 配置 server.xml

server.xml几乎保存了所有mycat需要的系统配置信息。最常用的是在此配置用户名、密码及权限。在system中添加UTF-8字符集设置，否则存储中文会出现问号

```

<property name="charset">utf8</property>

```

修改user的设置，我们这里为ITCAST设置了两个用户

```

<user name="test">
    <property name="password">test</property>
    <property name="schemas">ITCAST</property>
</user>
<user name="root">
    <property name="password">123456</property>
    <property name="schemas">ITCAST</property>
</user>

```

## 3.4 MyCat分片测试

进入mycat，执行下列语句创建一个表

```

CREATE TABLE tb_test (
    id BIGINT(20) NOT NULL,
    title VARCHAR(100) NOT NULL ,
    PRIMARY KEY (id)
) ENGINE=INNODB DEFAULT CHARSET=utf8

```

我们再查看MySQL的3个库，发现表都自动创建好啦。好神奇。

接下来是插入表数据，注意，在写INSERT语句时一定要写把字段列表写出来，否则会出现下列错误提示：

错误代码： 1064

partition table, insert must provide ColumnList

我们试着插入一些数据：

```
INSERT INTO TB_TEST(ID,TITLE) VALUES(1,'goods1');
INSERT INTO TB_TEST(ID,TITLE) VALUES(2,'goods2');
INSERT INTO TB_TEST(ID,TITLE) VALUES(3,'goods3');
```

我们会发现这些数据被写入到第一个节点中了，那什么时候数据会写到第二个节点中呢？

我们插入下面的数据就可以插入第二个节点了

```
INSERT INTO TB_TEST(ID,TITLE) VALUES(5000001,'goods5000001');
```

因为我们采用的分片规则是每节点存储500万条数据，所以当ID大于5000000则会存储到第二个节点上。

目前只设置了两个节点，如果数据大于1000万条，会怎么样呢？执行下列语句测试一下

```
INSERT INTO TB_TEST(ID,TITLE) VALUES(10000001,'goods10000001');
```

### 3.5 MyCat 分片规则

rule.xml用于定义分片规则，我们这里讲解两种最常见的分片规则

1). 按主键范围分片rang-long

```
<tableRule name="auto-sharding-long">
  <rule>
    <columns>id</columns>
    <algorithm>rang-long</algorithm>
  </rule>
</tableRule>
```

tableRule 是定义具体某个表或某一类表的分片规则名称

columns用于定义分片的列

algorithm代表算法名称

我们接着找rang-long的定义

```
<function name="rang-long"
class="org.opencloudb.route.function.AutoPartitionByLong">
  <property name="mapFile">autopartition-long.txt</property>
</function>
```

Function用于定义算法 mapFile 用于定义算法需要的数据，我们打开autopartition-long.txt

```
# range start-end ,data node index
# K=1000,M=10000.
0-500M=0
500M-1000M=1
1000M-1500M=2
```

## 2). 一致性哈希murmur

当我们需要将数据平均分在几个分区中，需要使用一致性hash规则

我们找到function的name为murmur 的定义，将count属性改为3，因为我要将数据分成3片

```
<function name="murmur"
class="org.opencldb.route.function.PartitionByMurmurHash">

    <!-- 默认是0 -->
    <property name="seed">0</property>

    <!-- 要分片的数据库节点数量，必须指定，否则没法分片 -->
    <property name="count">3</property>

    <!-- 一个实际的数据库节点被映射为这么多虚拟节点，默认是160倍，也就是虚拟节点数是物理节点数的160倍 -->
    <property name="virtualBucketTimes">160</property>

    <!-- 节点的权重，没有指定权重的节点默认是1。以properties文件的格式填写，以从0开始到count-1的整数值也就是节点索引为key，以节点权重值为值。所有权重值必须是正整数，否则以1代替 -->
    <!-- <property name="weightMapFile">weightMapFile</property> -->

    <!-- 用于测试时观察各物理节点与虚拟节点的分布情况，如果指定了这个属性，会把虚拟节点的murmur hash值与物理节点的映射按行输出到这个文件，没有默认值，如果不指定，就不会输出任何东西 -->
    <!-- <property name="bucketMapPath">/etc/mycat/bucketMapPath</property> -->
</function>
```

我们再配置文件中可以找到表规则定义

```
<tableRule name="sharding-by-murmur">
    <rule>
        <columns>id</columns>
        <algorithm>murmur</algorithm>
    </rule>
</tableRule>
```

但是这个规则指定的列是id ,如果我们的表主键不是id的话 ,那么我们应该重新定义一个tableRule:

```
<tableRule name="sharding-by-murmur-order">
    <rule>
        <columns>id</columns>
        <algorithm>murmur</algorithm>
    </rule>
</tableRule>
```



在schema.xml中配置逻辑表时，指定规则为sharding-by-murmur-order

```
<table name="tb_order" dataNode="dn1,dn2,dn3" rule="sharding-by-murmur-order" />
```

我们测试一下，创建品优购的订单表,并插入数据，测试分片效果：

创建表

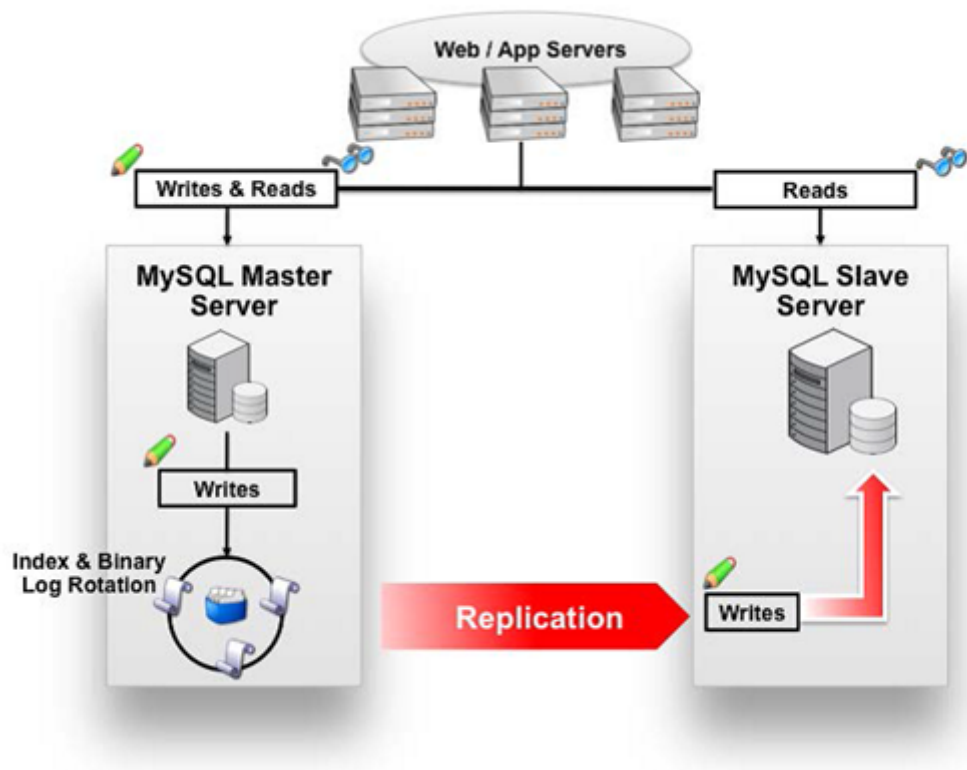
```
create table tb_order(  
    id int(11) primary key,  
    money int(11),  
    content varchar(200)  
)engine=InnoDB ;
```

```
INSERT INTO tb_order (id,money,content) VALUES(1, 100 , UUID());  
INSERT INTO tb_order (id,money,content) VALUES(212, 100 , UUID());  
INSERT INTO tb_order (id,money,content) VALUES(312, 100 , UUID());  
INSERT INTO tb_order (id,money,content) VALUES(412, 100 , UUID());  
INSERT INTO tb_order (id,money,content) VALUES(534, 100 , UUID());  
INSERT INTO tb_order (id,money,content) VALUES(621, 100 , UUID());  
INSERT INTO tb_order (id,money,content) VALUES(754563, 100 , UUID());  
INSERT INTO tb_order (id,money,content) VALUES(8123, 100 , UUID());  
INSERT INTO tb_order (id,money,content) VALUES(91213, 100 , UUID());  
INSERT INTO tb_order (id,money,content) VALUES(23232, 100 , UUID());  
INSERT INTO tb_order (id,money,content) VALUES(112321, 100 , UUID());  
INSERT INTO tb_order (id,money,content) VALUES(21221, 100 , UUID());  
INSERT INTO tb_order (id,money,content) VALUES(112132, 100 , UUID());  
INSERT INTO tb_order (id,money,content) VALUES(12132, 100 , UUID());  
INSERT INTO tb_order (id,money,content) VALUES(124321, 100 , UUID());  
INSERT INTO tb_order (id,money,content) VALUES(212132, 100 , UUID());
```

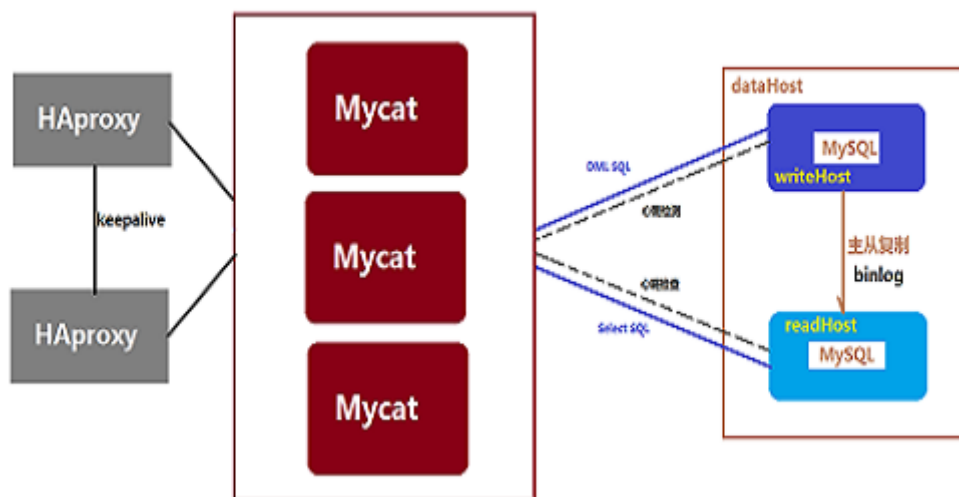
## 4. MyCat读写分离

数据库读写分离对于大型系统或者访问量很高的互联网应用来说，是必不可少的一个重要功能。对于MySQL来说，标准的读写分离是主从模式，一个写节点Master后面跟着多个读节点，读节点的数量取决于系统的压力，通常是1-3个读节点的配置

主从复制原理：



MyCat 实现读写分离:



MyCat读写分离和自动切换机制，需要mysql的主从复制机制配合。

配置如下：

- 1). 检查MySQL的主从复制是否运行正常。
- 2). 修改MyCat 的conf/schema.xml 配置如下:

```
<mycat:schema xmlns:mycat="http://io.mycat/">

  <schema name="ITCAST" checkSQLschema="true" sqlMaxLimit="100">
    <table name="tb_test" dataNode="dn1,dn2,dn3" rule="auto-sharding-long"
  />
    <table name="tb_order" dataNode="dn1,dn2,dn3" rule="sharding-by-murmur"
  />

    <table name="user" dataNode="dn4" primaryKey="id"/>
  </schema>
```

```

<dataNode name="dn1" dataHost="localhost1" database="db1" />
<dataNode name="dn2" dataHost="localhost1" database="db2" />
<dataNode name="dn3" dataHost="localhost1" database="db3" />

<dataNode name="dn4" dataHost="localhost2" database="test01" />

<dataHost name="localhost1" maxCon="1000" minCon="10" balance="0"
writeType="0" dbType="mysql" dbDriver="native" switchType="1"
slaveThreshold="100">
    <heartbeat>select user()</heartbeat>

    <writeHost host="hostM1" url="localhost:3306" user="root"
password="itcast"></writeHost>

</dataHost>

<dataHost name="localhost2" maxCon="1000" minCon="10" balance="1"
writeType="0" dbType="mysql"
    dbDriver="native" switchType="1" slaveThreshold="100">
    <heartbeat>select user()</heartbeat>
    <writeHost host="hostM1" url="192.168.192.138:3306" user="root"
password="itcast">
        <readHost host="hosts1" url="192.168.192.141:3306" user="root"
password="itcast" />
    </writeHost>
</dataHost>

</mycat:schema>

```

3). 配置完毕之后, 重启MyCat服务;

在原有配置基础上, 做如下修改:

A. checkSQLSchema="true"

B. 增加table

```
<table name="user" dataNode="dn4" primaryKey="id"/>
```

C. 增加dataNode

```
<dataNode name="dn4" dataHost="localhost2" database="test01" />
```

D. 增加dataHost

```
<dataHost name="localhost2" maxCon="1000" minCon="10" balance="1" writeType="0"
dbType="mysql"
  dbDriver="native" switchType="1" slaveThreshold="100">
  <heartbeat>select user()</heartbeat>
  <writeHost host="hostM1" url="192.168.192.138:3306" user="root"
password="itcast">
    <readHost host="hostS1" url="192.168.192.141:3306" user="root"
password="itcast" />
  </writeHost>
</dataHost>
```

含义说明:

checkSQLSchema

当该值设置为true时，如果我们执行语句"select \* from test01.user ;" 语句时，MyCat则会把schema字符去掉，可以避免后端数据库执行时报错；

balance

负载均衡类型，目前取值有4种：

**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。

**balance="1"**：全部的readHost与stand by writeHost都参与select语句的负载均衡，简而言之，就是采用双主双从模式(M1 --> S1，M2 --> S2)；

**balance="2"**：所有的读写操作都随机在writeHost，readHost上分发

**balance="3"**：所有的读请求随机分发到writeHost对应的readHost上执行，writeHost不负担读压力；balance=3只在MyCat1.4之后生效。