**《MyCAT 2 从入门到实战》**



# 【熟悉】MyCAT2简介

## 什么是MyCAT

MyCAT是目前最流行的分布式数据库中间插件，是一个开源的分布式数据库系统，是一个实现了MySQL协议的服务器。前端用户可以把它看作一个数据库代理，用MySQL客户端工具和命令行访问，其后端可以用MySQL原生协议与多个MySQL服务器通信，也可以用JDBC协议与大多数主流数据库服务器通信，其核心功能是**分表分库**，即将一个大表水平分割为N个小表，存储在后端MySQL服务器里或者其他数据库里。

MyCAT发展到目前，已经不是一个单纯的MySQL代理了，它的后端可以支持MySQL、SQL Server、Oracle、DB2、PostgreSQL等主流数据库，也支持MongoDB这种新型NoSQL方式的存储。未来，它还会支持更多类型的存储。

不过，无论是哪种存储方式，在最终用户看，MyCAT里都是一个传统的数据库表，支持标准的SQL语句进行数据的操作，这样一来，对前端业务系统来说，可以大幅降低开发难度，提升开发速度。



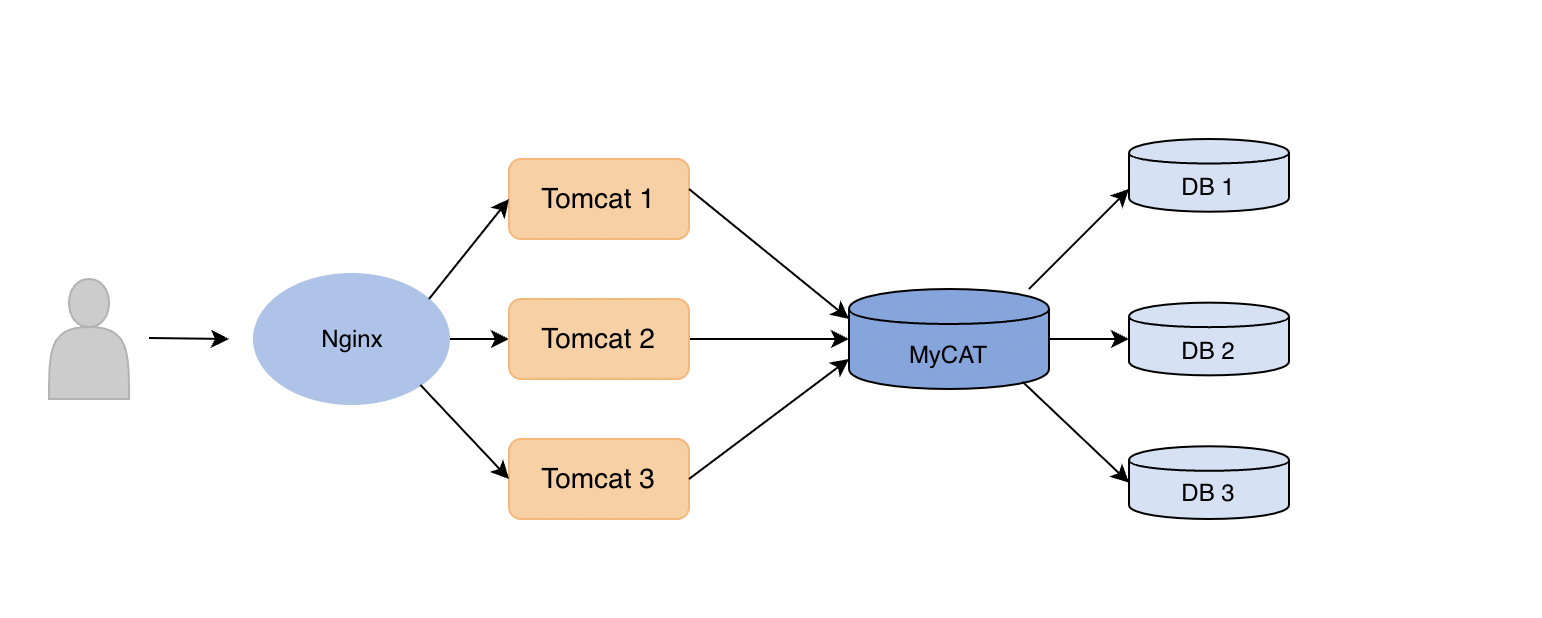
更多解释见官网： <http://www.mycat.org.cn/>

项目地址：源码地址

<https://github.com/MyCATApache/Mycat2>

<https://gitee.com/MycatOne/Mycat2>

## 为什么使用MyCAT



随着互联网的发展，数据的量级也是呈指数的增长，从GB到TB到PB，操作数据也是愈加的困难，传统的关系性数据库已经无法满足快速查询与插入数据的需求。

而NoSQL的出现暂时解决了这一危机。

NoSQL通过降低数据的安全性，减少对事务的支持，减少对复杂查询的支持，来提升性能。 但是，NoSQL存在无法满足某些使用场景的情况，比如，有些使用场景绝对要有事务与安全指标。这时NoSQL肯定无法满足。

所以，我们还是需要使用关系型数据库。

那么，如何使用关系型数据库解决海量存储的问题呢？此时就需要做数据库集群，为了提高查询性能将一个数据库的数据分散到不同的数据库中存储，为应对该问题，MyCAT 就出现了。

**我们来总结一下MyCAT的作用：**

1. 能满足数据库数据大量存储；
2. 提高了查询性能；
3. 实现读写分离，分库分表。

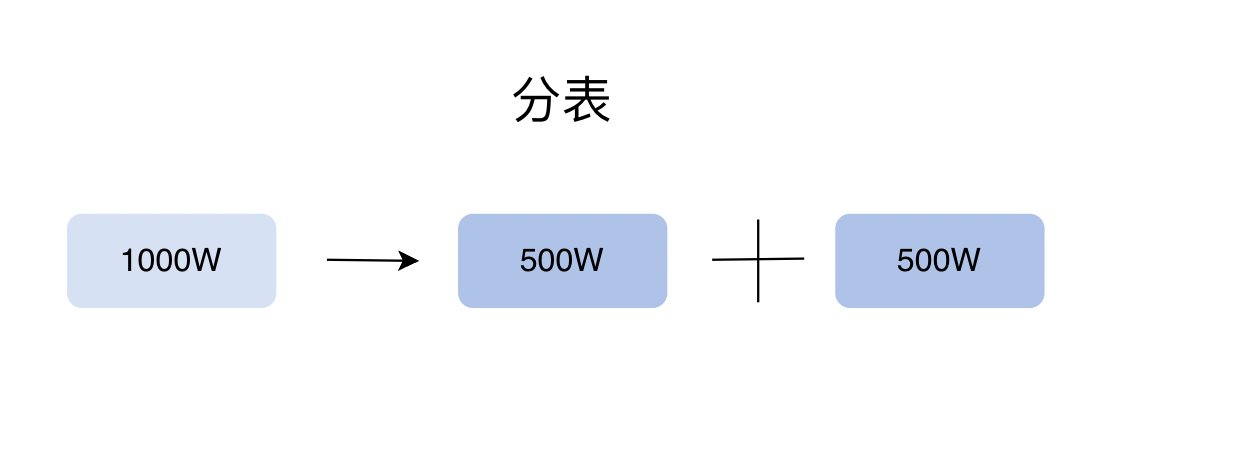
**同时，MyCAT 还可以解决如下问题。**

1. 解决表的容量问题：MySQL的表最大存储多少数据？500W条。
2. 解决表查询的性能问题？我的表里面的数据非常多，导致我查询很慢？
3. 解决表的DML的性能问题？
4. 解决MySQL挂了的问题。

**那么，MyCAT是怎么解决上述问题的呢？我们接着往下看。**

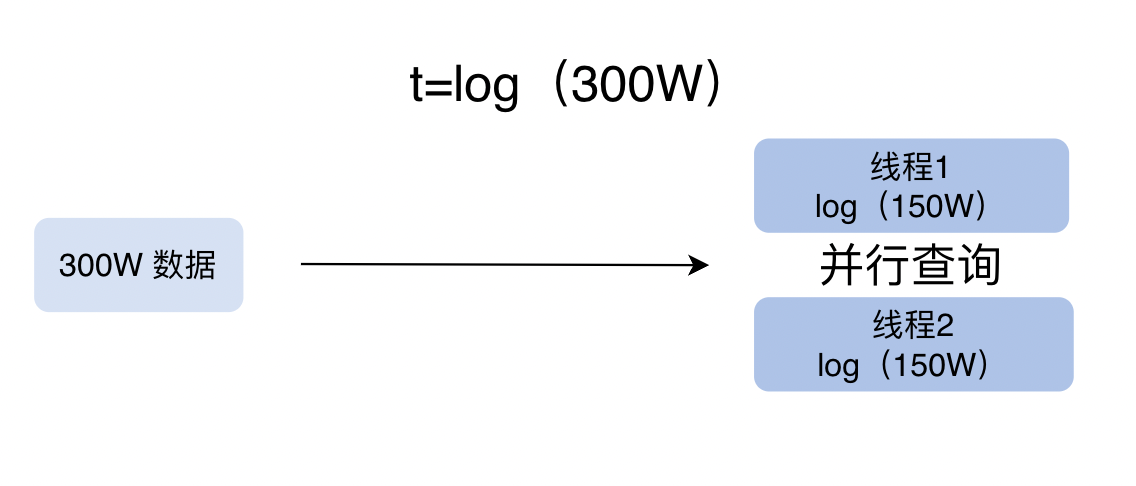
## MyCAT解决问题的思路

### 容量

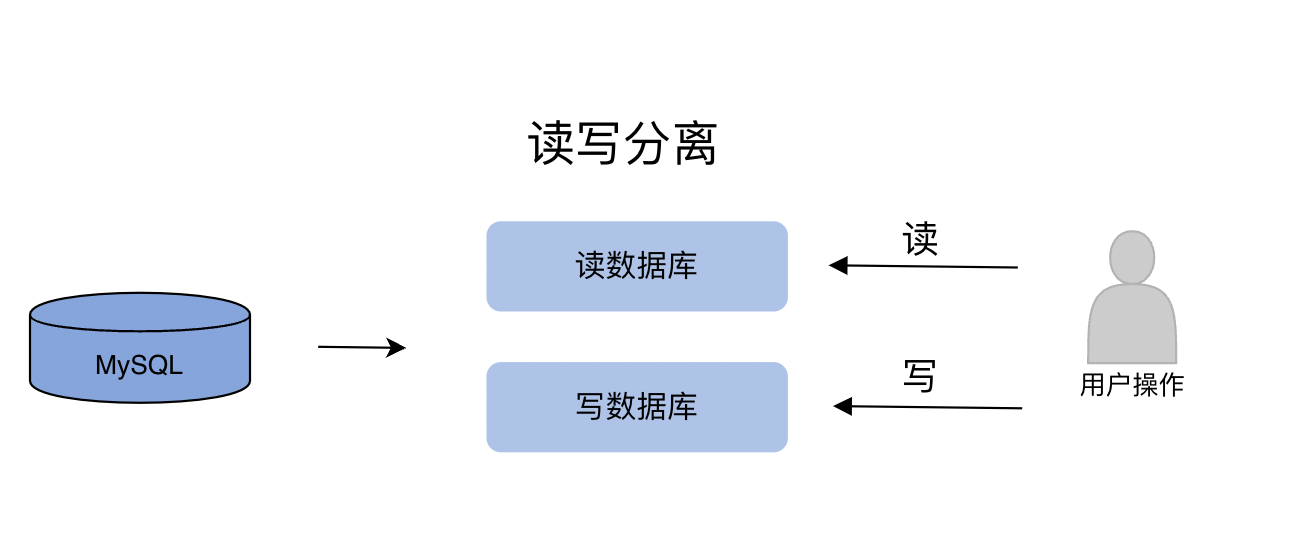


### 查询速度

最快也是Log(n)，你加什么索引都不行。



### 写入问题



## MyCAT和MySQL的区别

我们可以把上层看作是对下层的抽象，例如操作系统是对各类计算机硬件的抽象。那么我们什么时候需要抽象？

假设一个项目只需要一个人完成时，我们不需要Leader，但是当需要几十人完成时，就应该有一个管理者，发挥沟通协调等作用，这个管理者对于他的上层来说就是对项目组的抽象。

同样的，当我们的应用只需要一台数据库服务器时，我们并不需要MyCAT，**如果你需要分库甚至分表，这时候应用要面对很多个数据库时**，这时，就需要对数据库层做一个抽象，来管理这些数据库，最上面的应用只需要面对一个数据库层的抽象或者说数据库中间件就好了，这就是MyCAT的核心作用。

所以你可以这样理解：**数据库是对底层存储文件的抽象，而MyCAT是对数据库的抽象。**

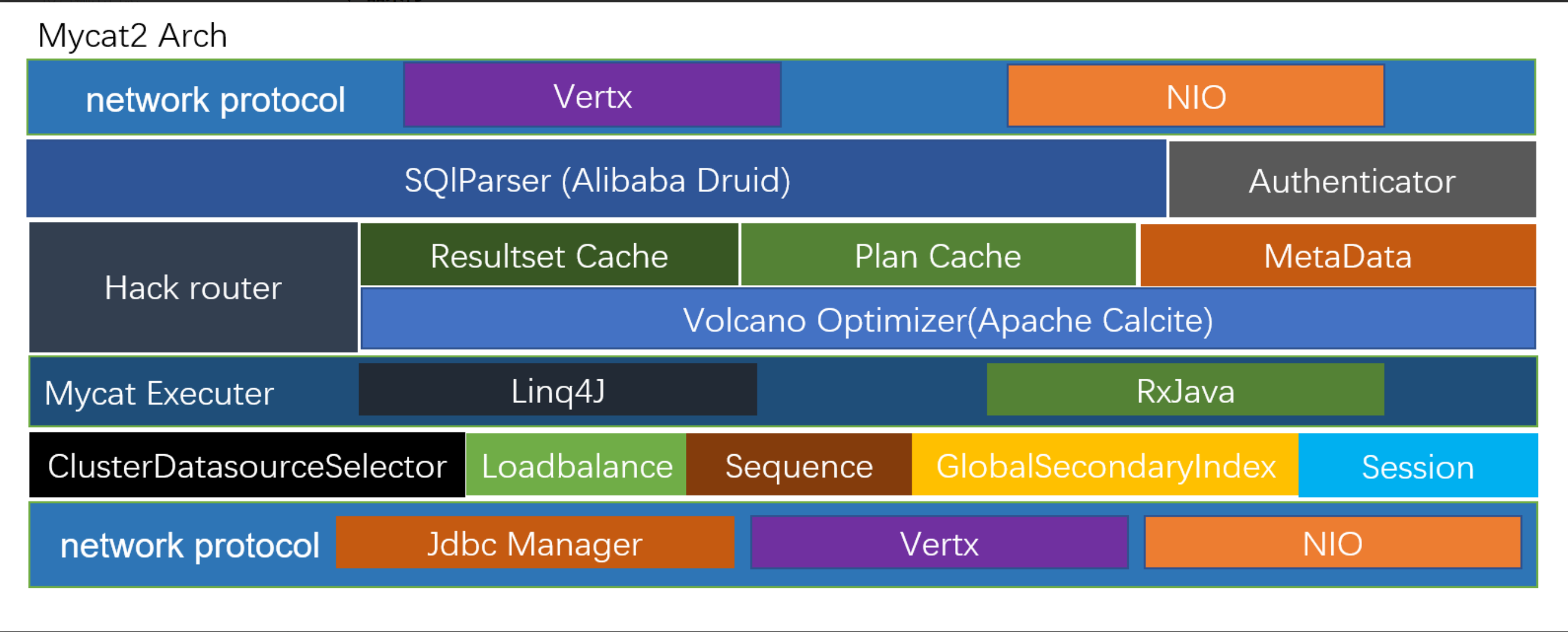
## 支持的数据库

MyCAT 支持的数据库比较多，主要有：MySQL，SQL Server，MongoDB等。



## 架构

前端用户可以把MyCAT看作是一个数据库代理，用MySQL客户端工具（如Navicat）和命令访问，而后端支持MySQL、SQL Server、Oracle等主流数据库，用MySQL Native 协议和多个MySQL服务器通信，也可用JDBC协议与大多数主流数据库服务器通信，其核心功能就是分库分表，即将一个大表水平分割为N个小表，真正的存储在后端MySQL服务器中或其他数据库中（这一点我们在“1.1 什么是 MyCAT”中已经讲过了）。



**MyCAT原理：**可以用“拦截”一词形容，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处理，最终返回给用户。

# 【掌握】MyCAT2名词解释

## 分库分表

按照一定规则把数据库中的表拆分为多个带有数据库实例、物理库、物理表访问路径的分表**（重点）**。

## 逻辑库

数据库代理中的数据库，它可以包含多个逻辑表**（重点）**。

## 逻辑表

数据库代理中的表，它可以映射代理连接的数据库中的表(物理表)**（重点）**。

## 物理库

数据库代理连接的数据库中的库，比如MySQL上的information\_schema**（重点）**。

## 物理表

数据库代理连接的数据库中的表，比如MySQL上的information\_schema.TABLES**（重点）。**

## 拆分键

即分片键，描述拆分逻辑表的数据规则的字段**（重点）。**

## 分区键

当使用等值查询的时候，能直接映射一个分区的拆分键**（重点）。**

## 物理分表

指已经进行数据拆分的，在数据库上面的物理表，是分片表的一个分区。

## 物理分库

一般指包含多个物理分表的库。

## 分库

一般指通过多个数据库拆分分片表，每个数据库一个物理分表，物理分库名字相同。

## 分片表、水平分片表

按照一定规则把数据拆分成多个分区的表，在分库分表语境下，它属于逻辑表的一种。

## 全局表(广播表)

每个数据库实例都冗余全量数据的逻辑表**（重点）。**

它通过表数据冗余，使分片表的分区与该表的数据在同一个数据库实例里，达到JOIN运算能够直接在该数据库实例里执行。它的数据一致一般是通过数据库代理分发SQL实现，也有基于集群日志的实现。

## 集群

多个数据节点组成的逻辑节点，在MyCAT2里，它是把对多个数据源地址视为一个数据源地址(名称)，并提供自动故障恢复、转移、即实现高可用、负载均衡的组件。

## 数据源

连接后端数据库的组件，它是数据库代理中连接后端数据库的客户端。

## schema(库)

在MyCAT2中配置表逻辑，视图等的配置。

# 【掌握】MyCAT2的安装

## 前言

使用MyCAT 2要安装JDK，也可以直接使用yum -y install java-1.8.0-openjdk.x86\_64 因为MyCAT 是基于JDK1.8开发的。

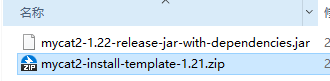
## 下载压缩包和jar包

1）安装程序包：<http://dl.mycat.org.cn/2.0/install-template/mycat2-install-template-1.21.zip>

2）jar包：<http://dl.mycat.org.cn/2.0/1.21-release/mycat2-1.21-release-jar-with-dependencies.jar>

3）wget：<http://dl.mycat.org.cn/2.0/install-template/mycat2-install-template-1.21.zip>

4）wget：<http://dl.mycat.org.cn/2.0/1.21-release/mycat2-1.21-release-jar-with-dependencies.jar>



## 安装MyCAT2

### 创建/data/tools

|  |
| --- |
| mkdir -p /data/tools |

### 进入/data/tools目录

|  |
| --- |
| cd /data/tools |

### 下载

**如果没有wget，就执行yum -y install wget**

wget <http://dl.mycat.org.cn/2.0/install-template/mycat2-install-template-1.21.zip>

wget <http://dl.mycat.org.cn/2.0/1.21-release/mycat2-1.21-release-jar-with-dependencies.jar>

### 下载完成



### 解压并移动到data目录下

|  |
| --- |
| yum -y install unzip  unzip mycat2-install-template-1.21.zip  mv mycat ../ |

### 修改权限

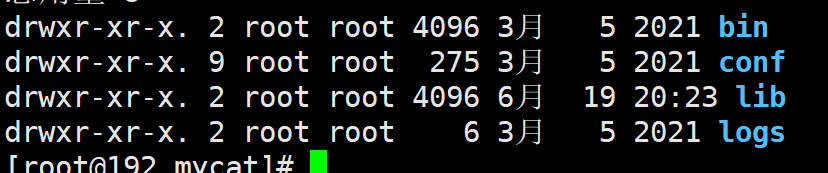
把bin目录的文件加执行权限：

|  |
| --- |
| cd /data/mycat/bin  chmod +x \* |

### 把所需的jar复制到mycat/lib目录

|  |
| --- |
| cd /data/mycat/lib/  cp /data/tools/mycat2-1.21-release-jar-with-dependencies.jar ./ |

## MyCAT的目录结构



其中：

1）bin 执行命令的目录

2）conf 配置文件

3）lib  依赖包

4）logs 日志包

## 启动一个3306的MySQL

回为mycat代理连接启动时需要有一个默认的数据源，所以我们在启动的时候先为其准备一个数据源，接下来我们使用docker启动

|  |
| --- |
| docker run --name mysql -p 3306:3306 -e MYSQL\_ROOT\_PASSWORD=123456 -d mysql:5.7 --lower\_case\_table\_names=1 |

## 配置物理库地址

在启动之前我们要配置物理库的地址，要不然MyCAT启动就会报错。

配置文件位置: mycat/conf/datasources/prototypeDs.datasource.json

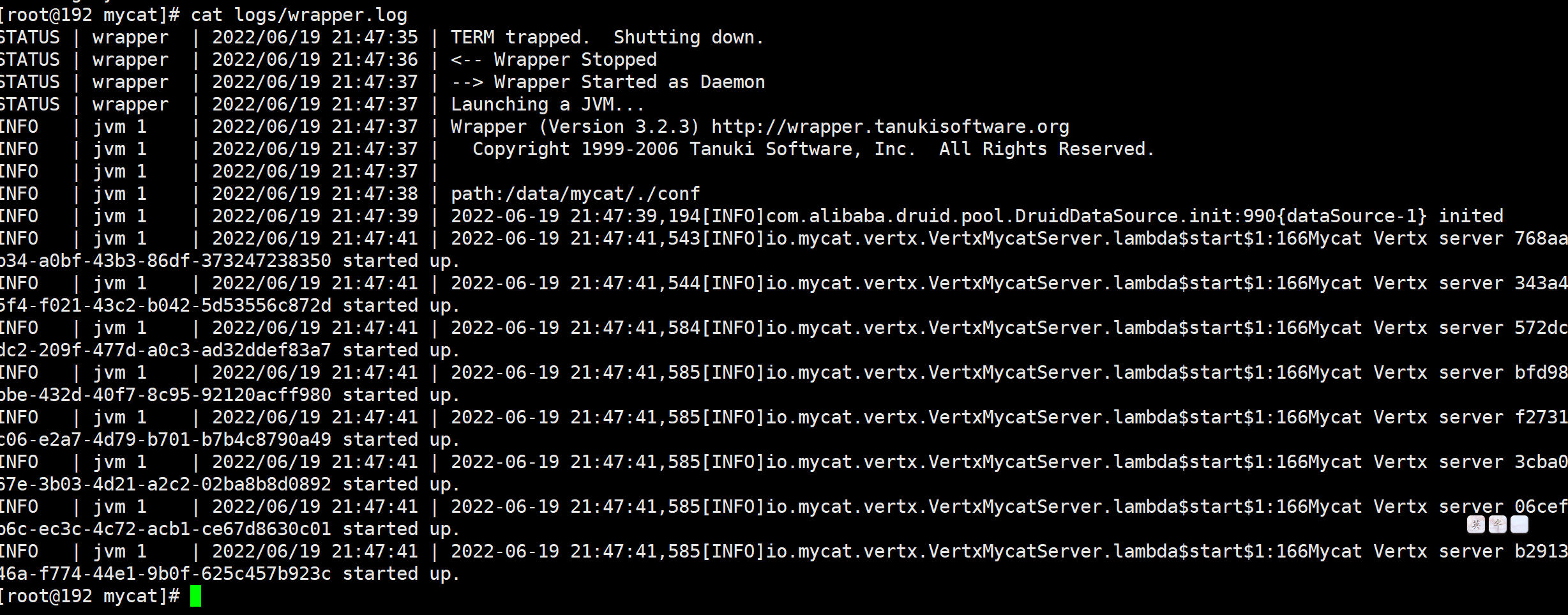
|  |
| --- |
| {  "dbType":"mysql",  "idleTimeout":60000,  "initSqls":[],  "initSqlsGetConnection":true,  "instanceType":"READ\_WRITE",  "maxCon":1000,  "maxConnectTimeout":3000,  "maxRetryCount":5,  "minCon":1,  "name":"prototypeDs",  **"password":"123456",**  "type":"JDBC",  **"url":"jdbc:mysql://localhost:3306?useUnicode=true&serverTimezone=Asia/Shanghai&characterEncoding=UTF-8",**  **"user":"root",**  "weight":0  } |

## 启动MyCAT

|  |
| --- |
| cd /data/mycat/bin  **./mycat start 启动**  **./mycat stop 停止**  ./mycat console 前台运行  ./mycat install 添加到系统自动启动  ./mycat remove 取消随系统自动启动  ./mycat restart 重启  **./mycat pause 暂停**  **./mycat status 查看启动状态** |

### 查看logs/wrapper.log文档看错

cat /data/mycat/logs/wrapper.log 如果出现下面的信息就说明启动成功了。



## 连接MyCAT

我们在外面看MyCAT，认为MyCAT 就是一个MySQL，怎么连接MySQL？

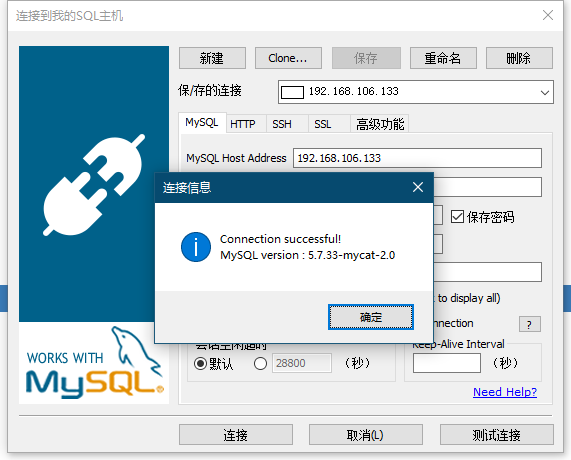
1）Navicat for MySQL 对MyCAT支持查询

2）SQLyog 操作MyCAT更友好



输入IP地址、用户名，和端口号**（注意端口号为8066）。**





**如果出现上面的弹出框，说明MyCAT安装成功。**

# 【掌握】MyCAT2配置文件介绍

## 用户（user）

配置文件位置mycat/conf/user/{用户名}.user.json。这个配置文件主要是用来配置MyCAT的登录用户的，也就是我们连接8066这个端口的用户信息。

|  |
| --- |
| {  "dialect":"mysql",  "ip":null,  "password":"123456",  "transactionType":"proxy",  "username":"root",  "isolation":3  } |
| **配置说明**  **1）dialect：**数据库(方言)类型 2）**ip：**配置白名单使用，一般写null 如果要限制这个用户  **3）password：**配置MyCAT用户的密码（明文）  **4）isolation：**设置初始化的事务隔离级别  READ\_UNCOMMITTED：1  READ\_COMMITTED：2  REPEATED\_READ：3，默认  SERIALIZABLE：4  **5）transactionType**  默认值：proxy（本地事务，在涉及大于1个数据库的事务，commit阶段失败会导致不一致，但是兼容性最好）  可选值：xa（事务，需要确认存储节点集群类型是否支持XA） |

## 数据源（datasource）

**作用：**配置MyCAT连接后端的物理库的数据源。

**配置文位置：**mycat/conf/datasources/{数据源名字}.datasource.json

|  |
| --- |
| {  "dbType": "mysql",  "idleTimeout": 60000,  "initSqls": [],  "initSqlsGetConnection": true,  "instanceType": "READ\_WRITE",  "maxCon": 1000,  "maxConnectTimeout": 3000,  "maxRetryCount": 5,  "minCon": 1,  "name": "prototype",  "password": "123456",  "type": "JDBC",  "url": "jdbc:mysql://127.0.0.1:3306/mysql?useUnicode=true&serverTimezone=UTC",  "user": "root",  "weight": 0,  "queryTimeout":30,//mills  } |
| **配置说明**  **1）dbType：**数据源类型  **2）name：**数据源名字  **3）password：**后端MySQL的密码  **4）url：**后端MySQL的JDBC连接地址  **5）user：**后端MySQL的用户名  **6）weight：**配置数据源负载均衡的使用权重 |

## 逻辑库与逻辑表（logicaltable）

作用：配置MyCAT里面和MySQL对应的逻辑表。

配置位置：mycat/conf/schemas/{库名}.schema.json

|  |
| --- |
| {  "customTables": {},  "globalTables": {},  "normalTables": {},  "schemaName": "test",  "shardingTables": {},  "targetName": "prototype"  } |
| **配置说明**  **1）customTables：**自定义表  **2）globalTables：**全局表  **3）normalTables：**默认表  **4）schemaName：**库名  **5）shardingTables：**分片表  **6）targetName：**数据源名，也可以是集群名 |

## 序列号（sequence）

作用：使用序列号的分片表,对应的自增主键要在建表SQL中体现。

配置位置: mycat/conf/sequences/{数据库名字}\_{表名字}.sequence.json（后面细说）。

## 服务器（server）

这个是针对MyCAT的服务器的配置，一般情况下默认就行了，不用怎么配置。

# 【掌握】MyCAT2注释配置

## 重置配置

|  |
| --- |
| /\*+ mycat:resetConfig{} \*/ |

## 用户相关

### 创建用户

|  |
| --- |
| /\*+ mycat:createUser{  "username":"user",  "password":"",  "ip":"127.0.0.1",  "transactionType":"xa"  } \*/ |

### 删除用户

|  |
| --- |
| /\*+ mycat:dropUser{  "username":"user"} \*/ |

### 显示用户

|  |
| --- |
| /\*+ mycat:showUsers \*/ |

## 数据源相关

### 创建数据源

|  |
| --- |
| /\*+ mycat:createDataSource{  "dbType":"mysql",  "idleTimeout":60000,  "initSqls":[],  "initSqlsGetConnection":true,  "instanceType":"READ\_WRITE",  "maxCon":1000,  "maxConnectTimeout":3000,  "maxRetryCount":5,  "minCon":1,  **"name":"dc1",**  **"password":"123456",**  **"type":"JDBC",**  **"url":"jdbc:mysql://127.0.0.1:3306?useUnicode=true&serverTimezone=UTC&characterEncoding=UTF-8",**  **"user":"root",**  "weight":0  } \*/; |

### 删除数据源

|  |
| --- |
| /\*+ mycat:dropDataSource{  "dbType":"mysql",  "idleTimeout":60000,  "initSqls":[],  "initSqlsGetConnection":true,  "instanceType":"READ\_WRITE",  "maxCon":1000,  "maxConnectTimeout":3000,  "maxRetryCount":5,  "minCon":1,  **"name":"dc1",**  "type":"JDBC",  "weight":0  } \*/; |

### 显示数据源

|  |
| --- |
| /\*+ mycat:showDataSources{} \*/ |

## 集群相关

### 创建集群

|  |
| --- |
| /\*! mycat:createCluster{  "clusterType":"MASTER\_SLAVE",  "heartbeat":{  "heartbeatTimeout":1000,  "maxRetry":3,  "minSwitchTimeInterval":300,  "slaveThreshold":0  },  "masters":[  "dc1" //主节点  ],  "maxCon":2000,  "name":"c0",  "readBalanceType":"BALANCE\_ALL",  "replicas":[  "dc2" //从节点  ],  "switchType":"SWITCH"  } \*/; |

**readBalanceType可选值:**

BALANCE\_ALL(默认值)

获取集群中所有数据源

BALANCE\_ALL\_READ

获取集群中允许读的数据源

BALANCE\_READ\_WRITE

获取集群中允许读写的数据源,但允许读的数据源优先

BALANCE\_NONE

获取集群中允许写数据源,即主节点中选择

### 删除集群

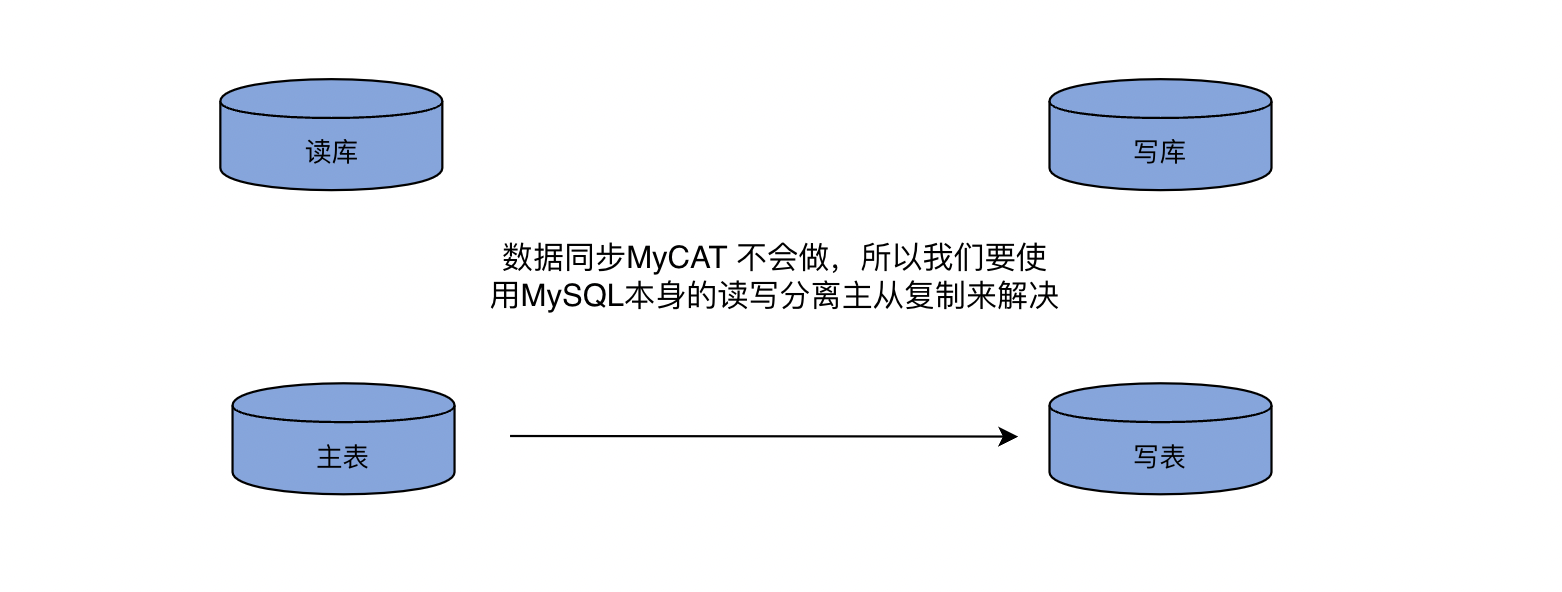
|  |
| --- |
| /\*! mycat:dropCluster{  "name":"c0"  } \*/; |

### 显示集群

|  |
| --- |
| /\*+ mycat:showClusters{} \*/ |

# 【熟悉】MySQL主从搭建

因为MyCAT只能路由、分发，不能把多个数据库里面的数据进行同步，所以要数据同步必须还要使用MySQL的读写分离，主从复制。



## 概述主从介绍

MySQL主从又叫Replication、AB复制。简单讲就是A与B两台机器做主从后，在A上写数据，另外一台B也会跟着写数据，实现数据实时同步。有这样几个关键点：

1）MySQL主从是基于binlog，主上需开启binlog才能进行主从；

2）主从过程大概有3个步骤；

3）主将更改操作记录到binlog里；

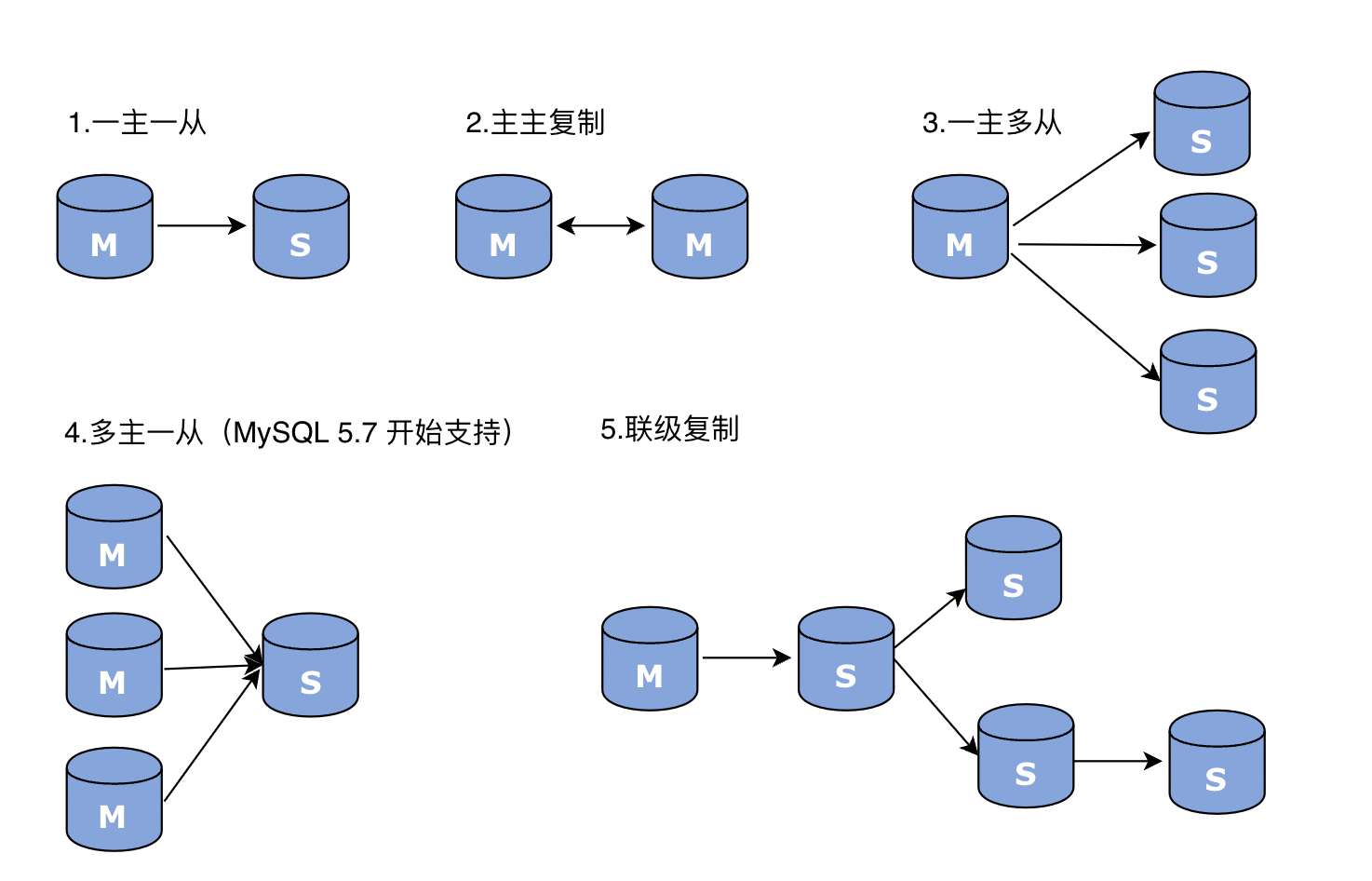
4）从将主的binlog事件（SQL语句） 同步本机上并记录在relaylog里；

5）从根据relaylog里面的SQL语句按顺序执行。

## 主从作用

**主从作用有：**实时灾备，用于故障切换；读写分离，提供查询服务；备份，避免影响业务。

**主从形式有：**



1）一主一从

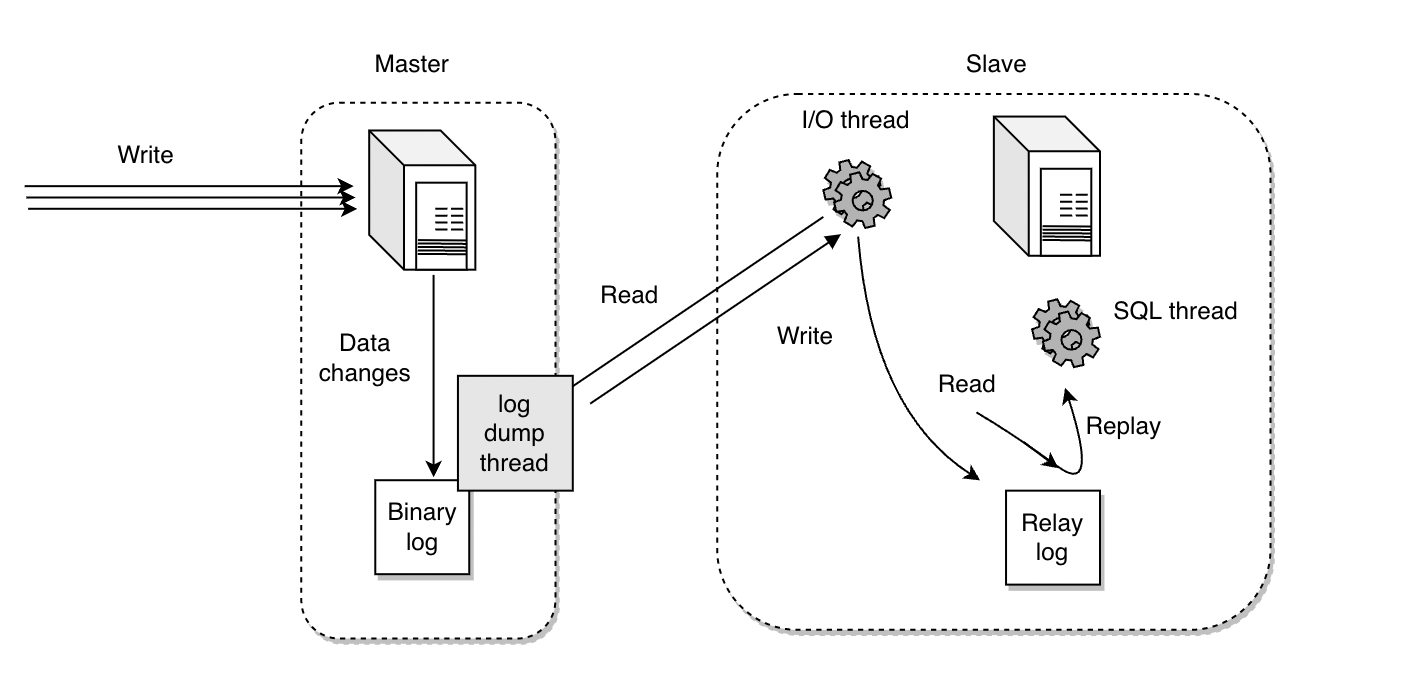
2）主主复制

3）一主多从：扩展系统读取的性能，因为读是在从库读取的。

4）多主一从：5.7版本开始支持

5）联级复制

## 主从复制原理



## 主从复制步骤

主库将所有的写操作记录在binlog日志中，并生成log dump线程，将binlog日志传给从库的I/O线程

从库生成两个线程，一个是I/O线程，另一个是SQL线程I/O线程去请求主库的binlog日志，并将binlog日志中的文件写入relay log（中继日志）中SQL线程会读取relay loy中的内容，并解析成具体的操作，来实现主从的操作一致，达到最终数据一致的目的。

## 主从复制配置步骤

1）确保从数据库与主数据库里的数据一致

2）在主数据库里创建一个同步账户授权给从数据库使用

3）配置主数据库（修改配置文件）

4）配置从数据库（修改配置文件）

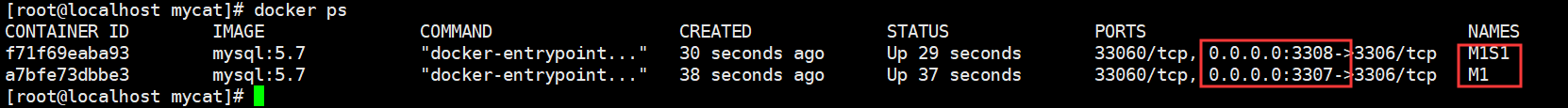
5）需求

6）搭建两台MySQL服务器，一台作为主服务器，一台作为从服务器，主服务器进行写操作，从服务器进行读操作

## 环境说明（使用Docker启动两个）

|  |  |  |
| --- | --- | --- |
| 名称 | Ip | Port |
| M1 | 192.168.106.133 | 3307 |
| M1S1 | 192.168.106.133 | 3308 |

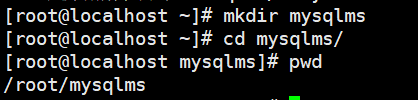




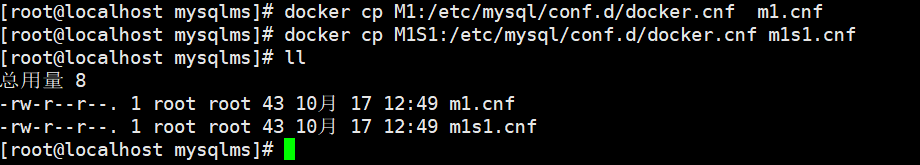
|  |
| --- |
| docker run --name M1 -p 3307:3306 -e MYSQL\_ROOT\_PASSWORD=123456 -d mysql:5.7 --lower\_case\_table\_names=1 |
| docker run --name M1S1 -p 3308:3306 -e MYSQL\_ROOT\_PASSWORD=123456 -d mysql:5.7 --lower\_case\_table\_names=1 |

## 修改配置文件

将容器里面的配置文件复制出来，主要修改服务器的配置；在root目录下创建一个mysqlms的目录存放从Docker容器里面复制过来的配置文件。进入目录：cd /mysqlms

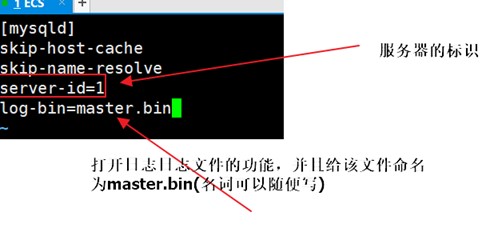


|  |
| --- |
| docker cp M1:/etc/mysql/conf.d/docker.cnf m1.cnf |
| docker cp M1S1:/etc/mysql/conf.d/docker.cnf m1s1.cnf |



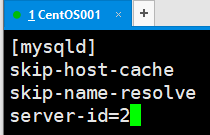
### 主机的配置m1.cnf

主机里面要记录SQL 语句，以后从机会把该SQL 语句偷过去



|  |
| --- |
| server-id=1  log-bin=master.bin |

### 从机的配置m1s1.cnf



|  |
| --- |
| server-id=2 |

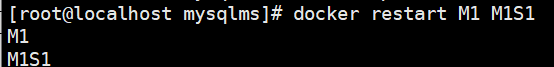
### 配置文件修改后，复制到容器里面

|  |
| --- |
| docker cp m1.cnf M1:/etc/mysql/conf.d/docker.cnf  docker cp m1s1.cnf M1S1:/etc/mysql/conf.d/docker.cnf |

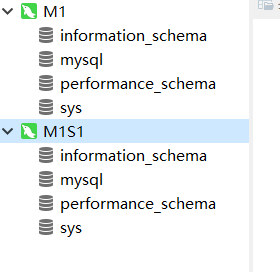


**重启MySQL（m1,m1s1）：**

|  |
| --- |
| docker restart M1 M1S1 |



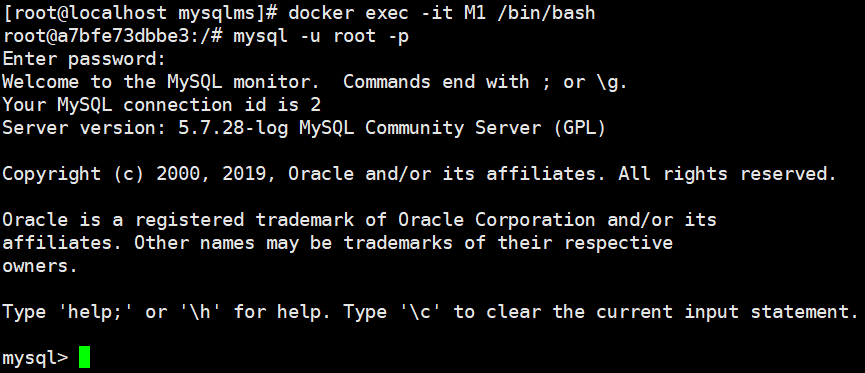
**测试连接：**



## 执行SQL语句

### 进入主机里面执行相关配置

|  |
| --- |
| docker exec -it M1 bash  mysql -uroot -p123456 |



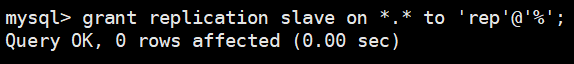
**创建用户：**

|  |
| --- |
| create user 'rep'@'%' identified by '123456'; |



**给该用户授予权限：**

|  |
| --- |
| grant replication slave on \*.\* to 'rep'@'%'; |



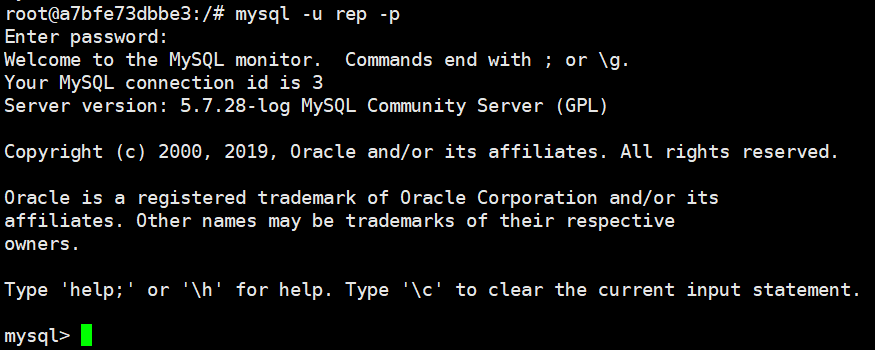
**刷新权限：**

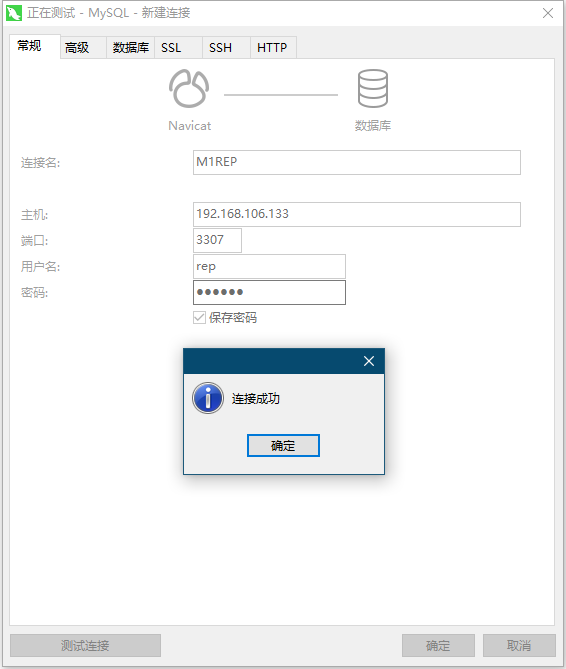
|  |
| --- |
| flush privileges; |



至此，M1 里面已经创建了一个用户：rep 123456 拥有所有库，所有表replication slave。

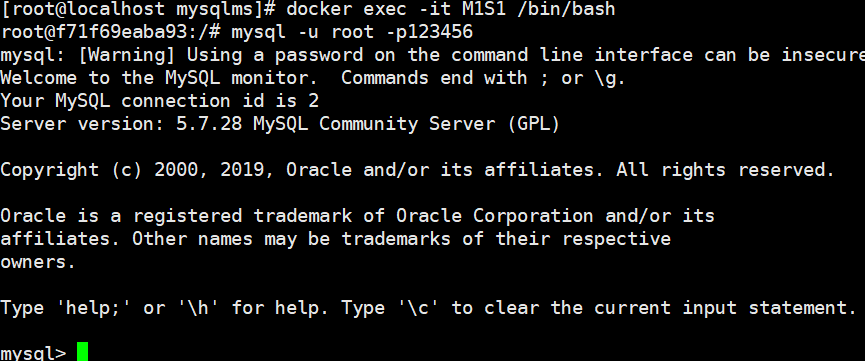
接下来，我们尝试使用M1 里面的rep 用户登录：





### 进入从机里面执行相关配置

|  |
| --- |
| docker exec -it M1S1 bash  mysql -u root -p123456 |

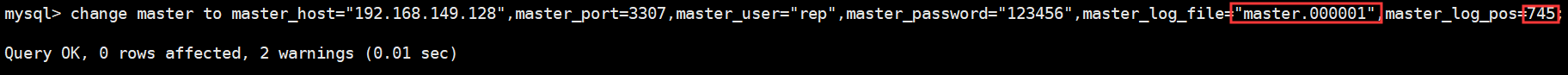


|  |
| --- |
| change master to master\_host="**192.168.106.133**",master\_port=**3307**,master\_user="**rep**",master\_password="**123456**",master\_log\_file="**master.000001**",master\_log\_pos=**745;** |

其中，master\_log\_file：该文件具体叫什么名称，需要从主机里面去看看。进入M1 里面使用root 用户登录M1，执行下面的SQL：**show master status;**



**修改上面的SQL执行：**



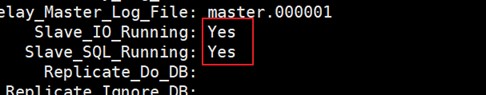
**启动主从：（在M1S1里面执行）：**start slave ;



**查询主从的状态（M1S1）：**show slave status \G;



**成功的标志：**



## 搭建失败的原因

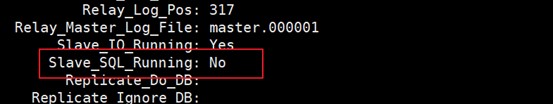
### 第一个不是yes，是connecting

是因为从机使用你配置的主机信息没有登陆到主机里面！修改(从机里面)

|  |
| --- |
| stop slave;  change master to master\_host="192.168.106.133",master\_port=3307,master\_user="rep",master\_password="123456",master\_log\_file="master.000001",master\_log\_pos=745;  start slave; |

### 第二个不是yes，是no

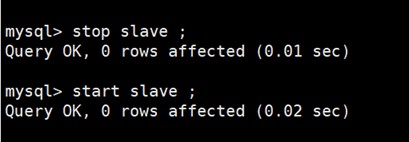
**原因是主机和从机里的数据不一致：**



从机会复制主机里面的SQL语句，来自己执行！实验时先把从机里面的db3 删除—>再把主机里面的db3 删除->从机里面复制该删除的命令->从机执行删除的命令（db3），事务无法提交，将一直阻塞！



现在从机里面要删除db3 ，但是没有db3，导致一直阻塞，以后的主从复制不会进行了。解决：在从机新建一个db3，然后停止主从，启动主从：



### 第一个不是yes，是no

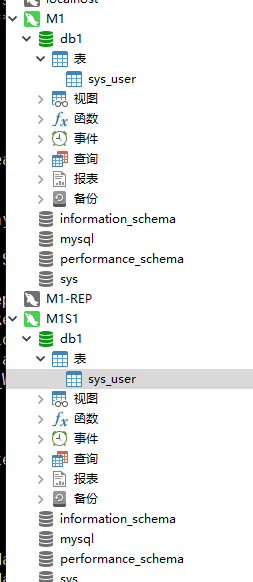
就是你的server-id 没有配置成功的原因，需要重新修改配置文件，复制配置文件到容器里面，然后重启就ok

### MySQL主从的操作规范

1. 只能在主机里面执行DML 语句，不能在从机里面执行DML语句（会破坏主从）
2. 在从机里面可以执行查询语句
3. 主机只有一台，但是从机可以有多台

## 测试

在M1 里面创建数据库，看M1S1 有没有复制过去



# 【掌握】MyCAT2的主从配置

前提，先搭建好MySQL的主从配置，登录MyCAT 2在MyCAT2里面操作，也就是连接8066这个端口。

## 创建数据源

|  |
| --- |
| **添加读写的数据源**  /\*+ mycat:createDataSource{  "dbType":"mysql",  "idleTimeout":60000,  "initSqls":[],  "initSqlsGetConnection":true,  **"instanceType":"READ\_WRITE",**  "maxCon":1000,  "maxConnectTimeout":3000,  "maxRetryCount":5,  "minCon":1,  "name":"m1",  "password":"123456",  "type":"JDBC",  "url":"jdbc:mysql://127.0.0.1:3307/db1?useUnicode=true&serverTimezone=UTC&characterEncoding=UTF-8",  "user":"root",  "weight":0  } \*/; |
| **添加读的数据源**  /\*+ mycat:createDataSource{  "dbType":"mysql",  "idleTimeout":60000,  "initSqls":[],  "initSqlsGetConnection":true,  **"instanceType":"READ",**  "maxCon":1000,  "maxConnectTimeout":3000,  "maxRetryCount":5,  "minCon":1,  "name":"m1s1",  "password":"123456",  "type":"JDBC",  "url":"jdbc:mysql://127.0.0.1:3308/db1?useUnicode=true&serverTimezone=UTC&characterEncoding=UTF-8",  "user":"root",  "weight":0  } \*/; |

## 查询数据源

|  |
| --- |
| /\*+ mycat:showDataSources{} \*/ |

## 创建集群

|  |
| --- |
| /\*! mycat:createCluster{  "clusterType":"MASTER\_SLAVE",  "heartbeat":{  "heartbeatTimeout":1000,  "maxRetry":3,  "minSwitchTimeInterval":300,  "slaveThreshold":0  },  **"masters":[**  **"m1"**  **],**  "maxCon":2000,  "name":"prototype",  "readBalanceType":"BALANCE\_ALL",  **"replicas":[**  **"m1s1"**  **],**  "switchType":"SWITCH"  } \*/; |

## 查询集群

|  |
| --- |
| /\*+ mycat:showClusters{} \*/ |

## 创建逻辑库

|  |
| --- |
| CREATE DATABASE db1 DEFAULT CHARACTER SET utf8mb4 COLLATE utf8mb4\_general\_ci; |

## 修改逻辑库的数据源

|  |
| --- |
| 修改conf/schemas/db1.schema.json  vim /data/mycat/conf/schemas/db1.schema.json  在里面添加  "targetName":"prototype", |

## 查看集群的配置文件

我们查看集群配置的结果，发现文件里面自动帮我们添加了。

## 测试读写分离是否成功(在MyCAT里面测试)

重启MyCAT：



**1）在MyCAT里面创建一个sys\_user表：**

|  |
| --- |
| CREATE TABLE SYS\_USER(  ID BIGINT PRIMARY KEY,  USERNAME VARCHAR(200) NOT NULL,  ADDRESS VARCHAR(500)  ) |

**2）通过注释生成物理库和物理表：**

如果物理表不存在，在 MyCAT2 能正常启动的情况下，根据当前配置自动创建分片表，全局表和物理表：

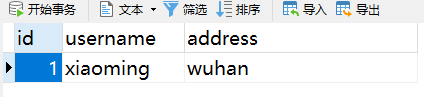
|  |
| --- |
| /\*+ mycat:repairPhysicalTable{} \*/; |

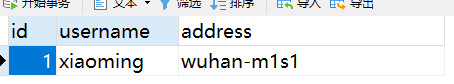
**3）查看后端物理库：**发现物理库和物理表都生成了。

**4）在MyCAT里面向sys\_user表添加一条数据：**

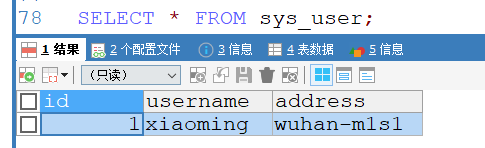
|  |
| --- |
| INSERT INTO SYS\_USER(ID,USERNAME,ADDRESS) VALUES(1,"XIAOMING","WUHAN"); |

**5）修改MySQL里面的让数据不一样：**





**6）在MyCAT里面查询数据，会发现每次查询的结果不一样：**



**到此，我们使用MyCAT2主从搭建就完成了。**

# 【掌握】MySQL的集群搭建（双主双从）

## 集群搭建概述

### 是什么

集群（Cluster）是一种较新的技术，通过集群技术，可以在付出较低成本的情况下，获得在性能、可靠性、灵活性方面的相对较高的收益，其任务调度则是集群系统中的核心技术。

MySQL集群技术在分布式系统中为MySQL数据提供了冗余特性，增强了安全性，使得单个MySQL服务器故障不会对系统产生巨大的负面效应，系统的稳定性得到保障。

### 优点

主要有以下两个优点。

**1）高可伸缩性：**服务器集群具有很强的可伸缩性，随着需求和负荷的增长，可以向集群系统添加更多的服务器，在这样的配置中，可以有多台服务器执行相同的应用和数据库操作。

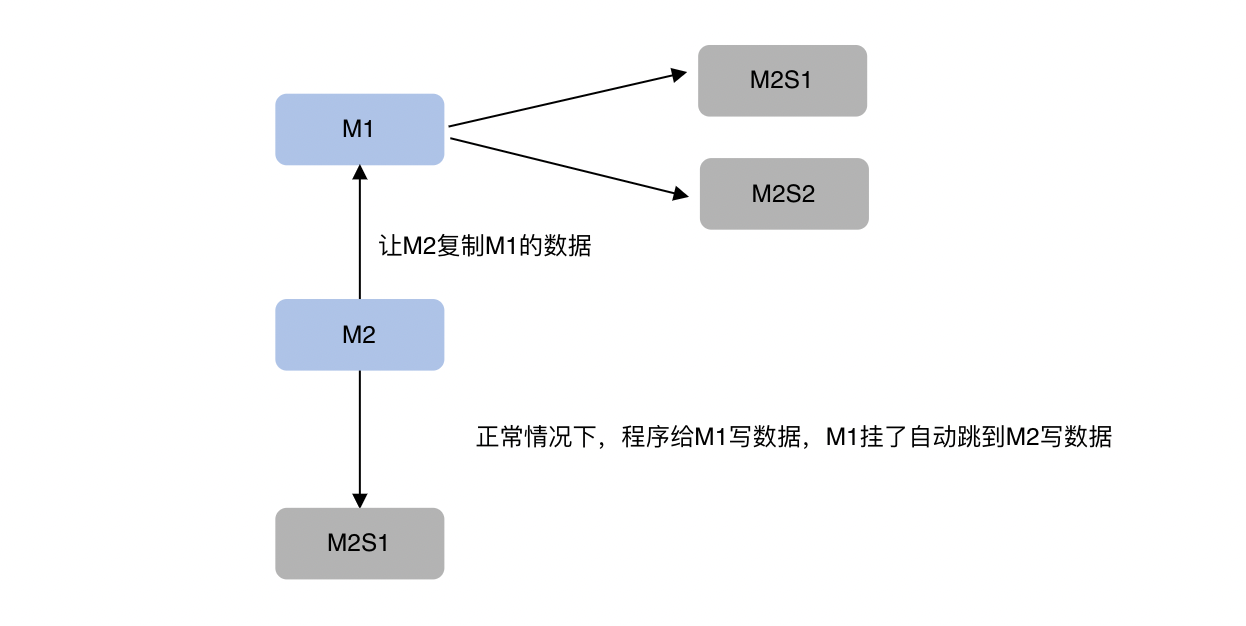
1. **高可用性：**在不需要操作者干预的情况下，防止系统发生故障或从故障中自动恢复的能力。通过把故障服务器上的应用程序转移到备份服务器上运行，集群系统能够把正常运行时间提高到大于99.9%，大大减少服务器和应用程序的停机时间。

### 缺点

我们知道，集群中的应用只在一台服务器上运行，如果这个应用出现故障，其他的某台服务器会重新启动这个应用，接管位于共享磁盘柜上的数据区，进而使应用重新正常运转。

而整个应用的接管过程大体需要三个步骤：侦测并确认故障、后备服务器重新启动该应用、接管共享的数据区。因此在切换的过程中需要花费一定的时间，原则上根据应用的大小不同切换的时间也会不同，越大的应用切换的时间越长。

## 搭建思路图



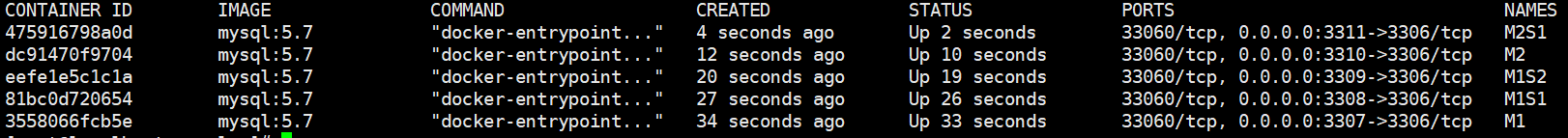
## 搭建准备工作

### 使用Docker启动5台MySQL（至少要5台）

|  |  |  |
| --- | --- | --- |
| 名称 | Ip | Port |
| M1 | 192.168.106.133 | 3307 |
| M1S1 | 192.168.106.133 | 3308 |
| M1S2 | 192.168.106.133 | 3309 |
| M2 | 192.168.106.133 | 3310 |
| M2S1 | 192.168.106.133 | 3311 |

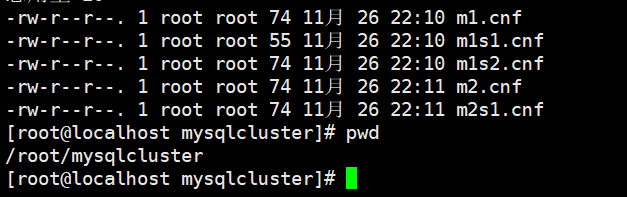
前面主从已启动了两个，所以只用启动三个就够了

|  |
| --- |
| docker run --name M1 -p 3307:3306 -e MYSQL\_ROOT\_PASSWORD=123456 -d mysql:5.7 --lower\_case\_table\_names=1 |
| docker run --name M1S1 -p 3308:3306 -e MYSQL\_ROOT\_PASSWORD=123456 -d mysql:5.7 --lower\_case\_table\_names=1 |
| docker run --name M1S2 -p 3309:3306 -e MYSQL\_ROOT\_PASSWORD=123456 -d mysql:5.7 --lower\_case\_table\_names=1 |
| docker run --name M2 -p 3310:3306 -e MYSQL\_ROOT\_PASSWORD=123456 -d mysql:5.7 --lower\_case\_table\_names=1 |
| docker run --name M2S1 -p 3311:3306 -e MYSQL\_ROOT\_PASSWORD=123456 -d mysql:5.7 --lower\_case\_table\_names=1 |



### 配置文件的修改

**先准备5 个配置文件：**



分别修改。

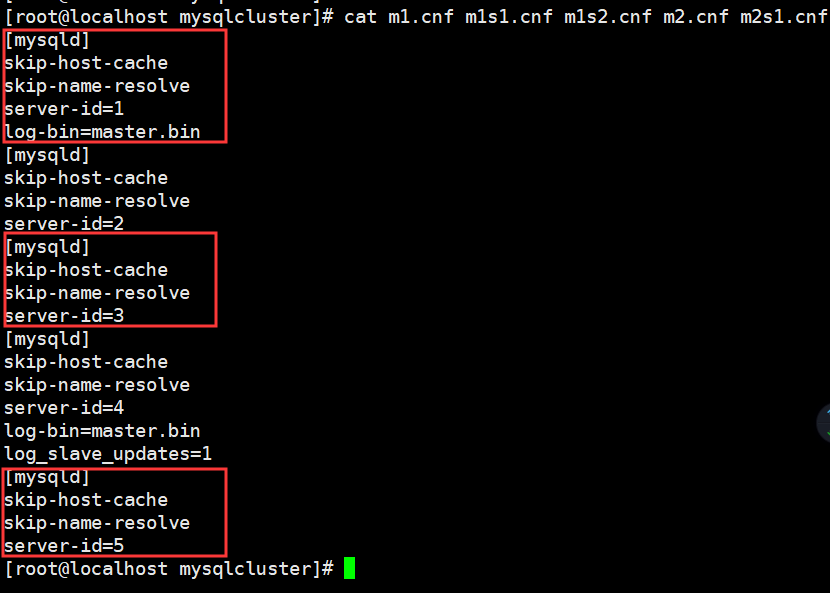
1）因为在上面的主从里面已有两个配置文件，所以只用创建三个就够了：m1s2.cnf、m2.cnf、m2s1.cnf

2）M2 需要添加一个配置项(具体配置看下面截图)

3）M2 会从M1 复制数据，但是M2 从M1复制的数据，不会记录下来，则M2S1 里面没有数据！

4）我们需要打开M2的级联复制功能，让M2 也能记录从M1 里面复制的数据。

**最后结果如下：**

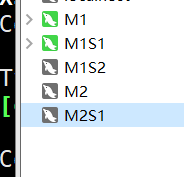


配置文件修改好了后，复制到容器里面，并且重启容器！

|  |
| --- |
| #docker cp m1.cnf M1:/etc/mysql/conf.d/docker.cnf  #docker cp m1s1.cnf M1S1:/etc/mysql/conf.d/docker.cnf  docker cp m1s2.cnf M1S2:/etc/mysql/conf.d/docker.cnf  docker cp m2.cnf M2:/etc/mysql/conf.d/docker.cnf  docker cp m2s1.cnf M2S1:/etc/mysql/conf.d/docker.cnf |

### 重启新建的三个容器

|  |
| --- |
| docker restart M1S2 M2 M2S1 |

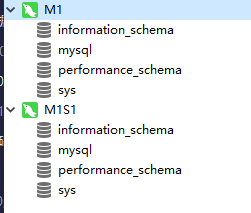


## 进入容器执行SQL

在搭建MySQL的集群时，先清空所有机器里面的数据。

现在有数据的是：M1 M1S1，只需要清空M1的数据，就都没有了。

将之前新建的db1删除就ok。



**4.1，M1修改**

因为前面搭建好了，所有不用修改

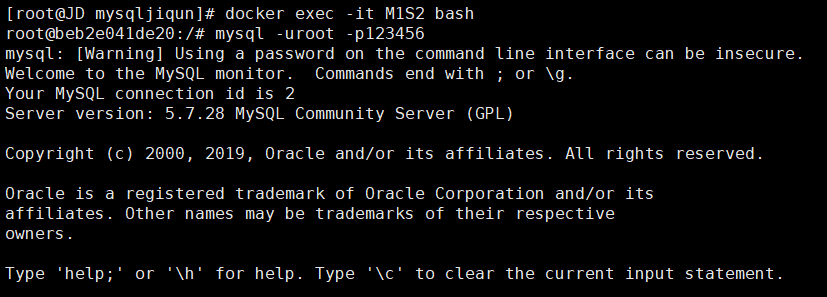
**4.2，M1S1修改**

同理上面的4.1

**4.3，M1S2修改（从机）**

给它设置一个主机就ok

|  |
| --- |
| docker exec -it M1S2 bash  mysql -uroot -p123456 |

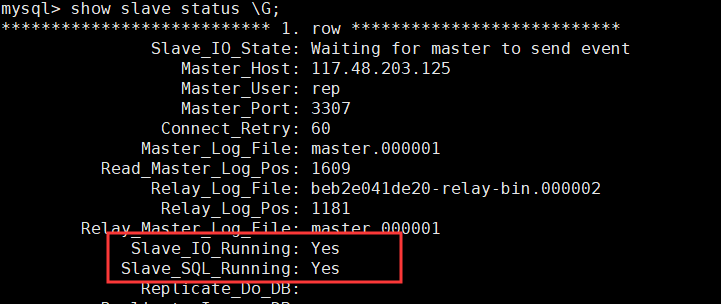
****

**关联主机：**

|  |
| --- |
| change master to master\_host="192.168.106.133",master\_port=3307,master\_user="rep",master\_password="123456",master\_log\_file="master.000001",master\_log\_pos=3574; |

**启动主从：**start slave ;

**查看状态：**show slave status \G;

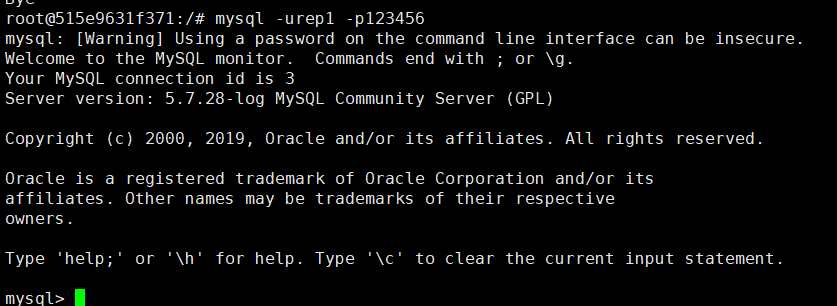


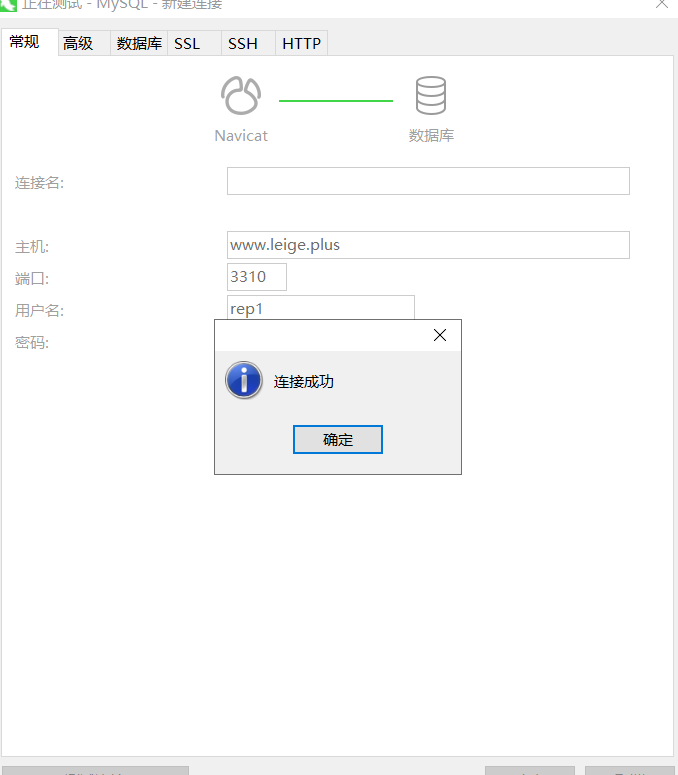
**4.4，M2修改**

进入M2使用Root登录

|  |
| --- |
| docker exec -it M2 bash  mysql -uroot -p123456 |

1. **新建用户：**create user 'rep1'@'%' identified by '123456';
2. **给用户授权：**grant replication slave on \*.\* to 'rep1'@'%';
3. **刷新权限：**flush privileges;
4. **使用新的用户尝试登录：**

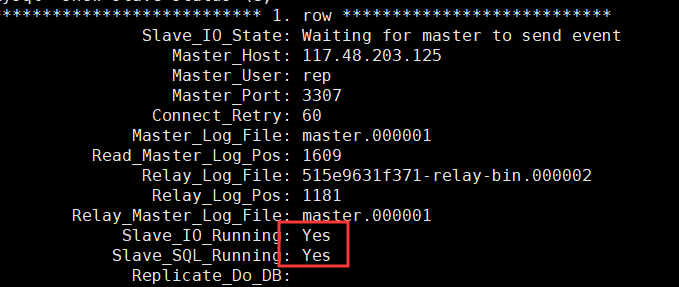




M2 作为M1的从机需要执行的SQL：（登录M2时要使用root用户）

|  |
| --- |
| change master to master\_host="192.168.106.133",master\_port=3307,master\_user="rep",master\_password="123456",master\_log\_file="master.000001",master\_log\_pos=3574; |

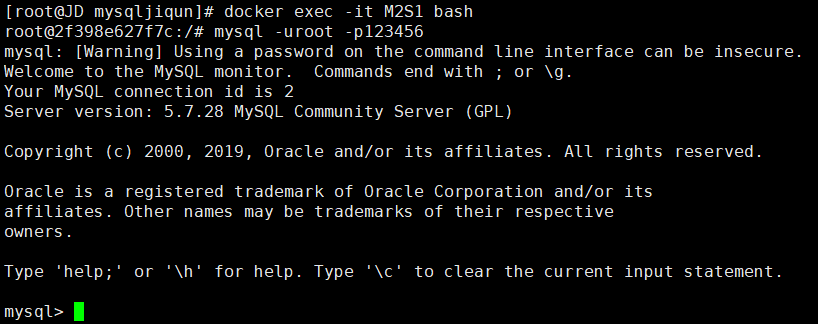
|  |
| --- |
| start slave ;  show slave status \G; |



**4.5，M2S1修改**

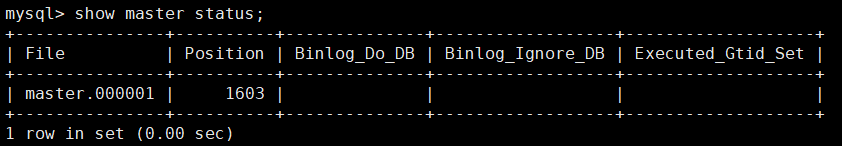
进入容器

|  |
| --- |
| docker exec -it M2S1 bash  mysql -uroot -p123456 |



M2S1 是M2的从机，执行的SQL

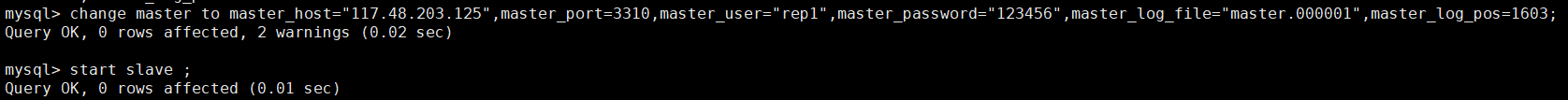
**进入M2查看pos：**show master status;



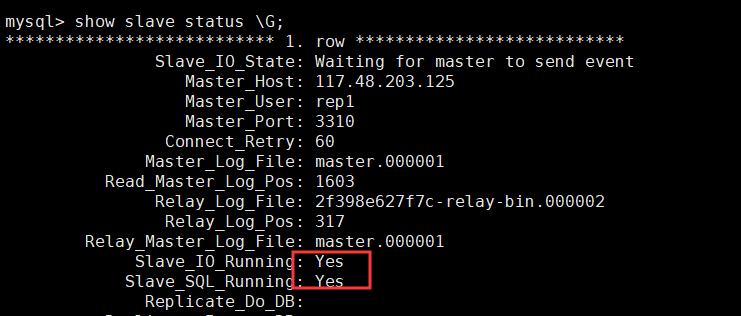
**在M2S1里面执行下面的命令：**

|  |
| --- |
| change master to master\_host="192.168.106.133",master\_port=3310,master\_user="rep1",master\_password="123456",master\_log\_file="master.000001",master\_log\_pos=747; |

**启动主从：**start slave ;

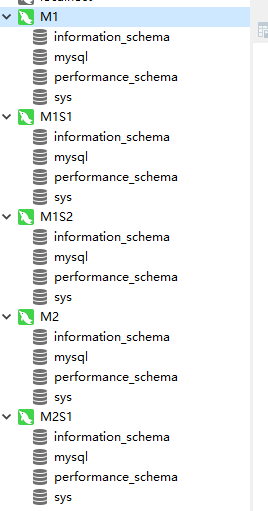


show slave status \G;



## 证集群是否成功

使用工具连接上所有的MySQL

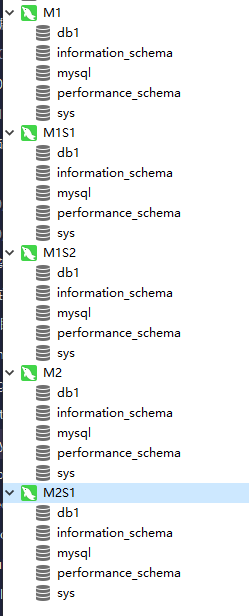


1）往M1 写数据，看 M2S1

2）在M1 新建数据库

3）观察M2S1

4）往M1 写数据，看M1S1



# 【掌握】MyCAT2操作双主双从

**目地：在MyCAT中集群配置实现双主双从。**

## 角色说明

|  |
| --- |
| M1 主机  M1S1 是M1的从机  M1S2 是M1的从机  M2 主机  M2S1 是M2的从机 |

## 添加数据源

|  |
| --- |
| **添加M1S2读的数据源**  /\*+ mycat:createDataSource{  "dbType":"mysql",  "idleTimeout":60000,  "initSqls":[],  "initSqlsGetConnection":true,  **"instanceType":"READ",**  "maxCon":1000,  "maxConnectTimeout":3000,  "maxRetryCount":5,  "minCon":1,  "name":"m1s2",  "password":"123456",  "type":"JDBC",  "url":"jdbc:mysql://127.0.0.1:3309?useUnicode=true&serverTimezone=UTC&characterEncoding=UTF-8",  "user":"root",  "weight":0  } \*/; |
| **添加M2的数据源**  /\*+ mycat:createDataSource{  "dbType":"mysql",  "idleTimeout":60000,  "initSqls":[],  "initSqlsGetConnection":true,  "instanceType":"READ\_WRITE",  "maxCon":1000,  "maxConnectTimeout":3000,  "maxRetryCount":5,  "minCon":1,  "name":"m2",  "password":"123456",  "type":"JDBC",  "url":"jdbc:mysql://127.0.0.1:3310?useUnicode=true&serverTimezone=UTC&characterEncoding=UTF-8",  "user":"root",  "weight":0  } \*/; |
| **添加M2S1的数据源**  /\*+ mycat:createDataSource{  "dbType":"mysql",  "idleTimeout":60000,  "initSqls":[],  "initSqlsGetConnection":true,  **"instanceType":"READ",**  "maxCon":1000,  "maxConnectTimeout":3000,  "maxRetryCount":5,  "minCon":1,  "name":"m2s1",  "password":"123456",  "type":"JDBC",  "url":"jdbc:mysql://127.0.0.1:3311?useUnicode=true&serverTimezone=UTC&characterEncoding=UTF-8",  "user":"root",  "weight":0  } \*/; |

## 修改配置集群

因为我们现在后端的MySQL集群结构发生了变化，现在修改集群的配置**【只是修改】**

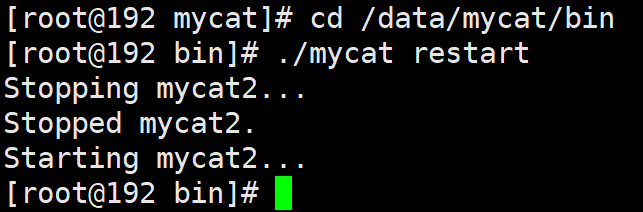
|  |
| --- |
| {  "clusterType":"MASTER\_SLAVE",  "heartbeat":{  "heartbeatTimeout":1000,  "maxRetryCount":0,  "minSwitchTimeInterval":300,  "showLog":true,  "slaveThreshold":0.0  },  "masters":[  "m1"**,"m2"**  ],  "maxCon":2000,  "name":"prototype",  "readBalanceType":"BALANCE\_ALL",  "replicas":[  "m1s1"**,"m1s2","m2s1"**  ],  "switchType":"SWITCH"  } |

## 重启MyCAT测试

### 重启MyCAT

|  |
| --- |
| cd /data/mycat/bin  ./mycat restart |

**重启成功如下：**



### 开始测试

**1）在MyCAT创建db1库：**

|  |
| --- |
| CREATE DATABASE db1 DEFAULT CHARACTER SET utf8mb4 COLLATE utf8mb4\_general\_ci; |

**2）在MyCAT创建表：**

|  |
| --- |
| create table sys\_user(  id bigint primary key,  username varchar(200) not null,  address varchar(500)  ) |

**3）刷新逻辑表到物理库**

|  |
| --- |
| /\*+ mycat:repairPhysicalTable{} \*/; |

刷新完成之后我们可以在后台的MySQL里面看到数据表已创建完成。

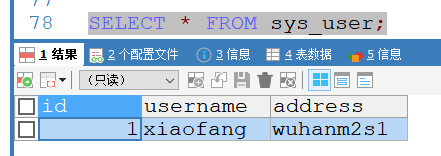
**4）在MyCAT里面添加数据：**

|  |
| --- |
| insert INTO sys\_user(id,username,address) values(1,"xiaofang","wuhan"); |

完成之后刷新后端MySQL物理库，我们发现有数据了。

**5）在MyCAT里查询测试：**

修改上面5个MySQL数据库中db1库中sys\_user表里面的数据，让它不一样，再在MyCAT里面执行查询 结果如下:



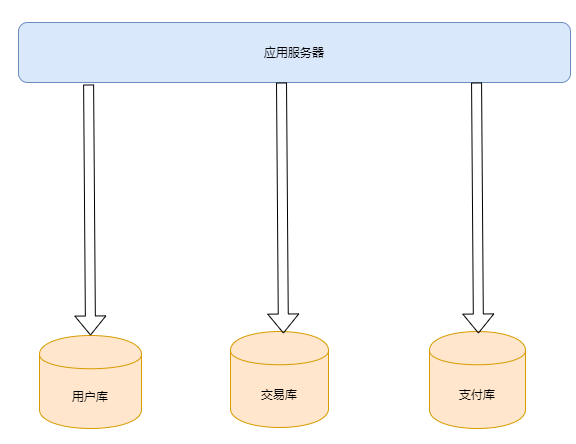
# 【掌握】MyCAT2分库分表

## 分库分表的原理

一个数据库由很多表构造成，每个表对应不同的业务，垂直切分是指按业务将表进行分类，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。

### 垂直切分

基于表或字段划分，表结构不同。我们有单库的分表，也有多库的分表，如下图所示



**注意：**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面

**换句话说：**就是我们很多库合在一起组成一个完整的数据库系统（这个不同到集群哦）。

### 水平切分

基于数据划分，**表结构相同**，**数据不同**，也有同库的水平切分和多库的切分。

当我们的客户表数量已经到达数千万甚至上亿的时候，单表的存储容量和查询效率都会出现问题，我们需要进一步对单张表的数据进行水平切分。水平切分的每个数据库的表结构都是一样的，只是存储的数据不一样，比如每个库存储 1000 万的数据。

|  |
| --- |
| **客户表 20W数据**  CREATE TABLE customer(  ID INT AUTO\_INCREMENT,  NAME VARCHAR(30),  PRIMARY KEY(ID)  )  **订单表 600W数据**  CREATE TABLE orders(  ID BIGINT AUTO\_INCREMENT,  ORDER\_TYPE INT,  CUSTOMER\_ID INT,  AMOUNT DECIMAL(10,2),  PRIMARY KEY(ID)  )  **订单详情表 800W数据**  CREATE TABLE orders\_detail(  ID BIGINT AUTO\_INCREMENT,  detail VARCHAR(2000),  order\_id BIGINT,  PRIMARY KEY(ID)  )  **字典表 100条数据**  CREATE TABLE dict\_order\_type(  ID INT AUTO\_INCREMENT,  order\_type VARCHAR(200),  PRIMARY KEY(ID )  ) |

## 如何分表

### 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1000W条数据就达到了瓶颈，会严重影响查询效率，所以我们需要进行水平拆分进行优化。例如：我们订单表和详情表里面的数据达到600W行数据，需要进行优化，那么我们就要把订单表的数据进行拆分存储了。

### 选择分表的定段

|  |  |  |
| --- | --- | --- |
| 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，一个客户所有的订单都在同一个节点，这样查询就快 |

## 分库分表的环境准备

我们可以按第6点的方式，搭建MySQL两组主从复制模型。

### 准备四个数据库

|  |  |  |
| --- | --- | --- |
| 名称 | ip | port |
| dw0 | 192.168.106.133 | 3312 |
| dr0 | 192.168.106.133 | 3313 |
| dw1 | 192.168.106.133 | 3314 |
| dr1 | 192.168.106.133 | 3315 |

### 分别使用Docker启动

|  |
| --- |
| docker run --name dw0 -p 3312:3306 -e MYSQL\_ROOT\_PASSWORD=123456 -d mysql:5.7 --lower\_case\_table\_names=1 |
| docker run --name dr0 -p 3313:3306 -e MYSQL\_ROOT\_PASSWORD=123456 -d mysql:5.7 --lower\_case\_table\_names=1 |
| docker run --name dw1 -p 3314:3306 -e MYSQL\_ROOT\_PASSWORD=123456 -d mysql:5.7 --lower\_case\_table\_names=1 |
| docker run --name dr1 -p 3315:3306 -e MYSQL\_ROOT\_PASSWORD=123456 -d mysql:5.7 --lower\_case\_table\_names=1 |
| **启动完成 docker ps 查看：** |

### 进入dw0配置

将容器里面的配置文件复制出来，主要修改服务器的配置。在root目录下创建一个mysqlms的目录存放从docker容器里面复制过来的配置文件。

**1）进入目录：**

|  |
| --- |
| cd /mysqlms  docker cp dw0:/etc/mysql/conf.d/docker.cnf dw0.cnf  docker cp dr0:/etc/mysql/conf.d/docker.cnf dr0.cnf |

**2）修改dw0.cnf：**

|  |
| --- |
| server-id=1  log-bin=master.bin |

**3）修改dr0.cnf：**

|  |
| --- |
| server-id=2 |

**4）配置文件修改后，复制到容器里面：**

|  |
| --- |
| docker cp dw0.cnf dw0:/etc/mysql/conf.d/docker.cnf  docker cp dr0.cnf dr0:/etc/mysql/conf.d/docker.cnf |

**5）重启MySQL（dw0,dr0）：**

|  |
| --- |
| docker restart dw0 dr0 |

**6）进入dw0主机里面执行相关配置：**

|  |
| --- |
| docker exec -it **dw0** bash  mysql -uroot -p123456 |

**7）创建用户**

|  |
| --- |
| create user 'rep'@'%' identified by '123456'; |

**8）给该用户授予权限：**

|  |
| --- |
| grant replication slave on \*.\* to 'rep'@'%'; |

**9）刷新权限：**

|  |
| --- |
| flush privileges; |

至此，dw0里面已经创建了一个用户：rep 123456 拥有所有库，所有表replication slave

**10）查看状态后面备用：**

|  |
| --- |
| show master status; |



### 进入dr0配置

**1）进入容器并登录：**

|  |
| --- |
| docker exec -it dr0 bash  mysql -u root -p123456 |

**2）执行主从配置：**

|  |
| --- |
| change master to master\_host="**192.168.106.133**",master\_port=**3312**,master\_user="**rep**",master\_password="**123456**",master\_log\_file="**master.000001**",master\_log\_pos=**745;** |

master\_log\_file：该文件具体叫什么名称，需要从主机里面去看看上面【**查看状态**】

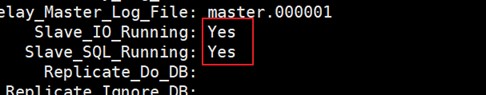
**3）启动主从（在dr0里面执行）：**

|  |
| --- |
| start slave ; |

**4）查询主从的状态（dr0）：**

|  |
| --- |
| show slave status \G; |

**5）成功的标志：**



### 进入dw1配置

将容器里面的配置文件复制出来，主要修改服务器的配置。在root目录下创建一个mysqlms的目录存放从docker容器里面复制过来的配置文件。

**1）进入目录：**

|  |
| --- |
| cd /mysqlms  docker cp dw1:/etc/mysql/conf.d/docker.cnf dw1.cnf  docker cp dr1:/etc/mysql/conf.d/docker.cnf dr1.cnf |

**2）修改dw1.cnf**

|  |
| --- |
| server-id=1  log-bin=master.bin |

**3）修改dr1.cnf**

|  |
| --- |
| server-id=2 |

**4）配置文件修改后，复制到容器里面：**

|  |
| --- |
| docker cp dw1.cnf dw1:/etc/mysql/conf.d/docker.cnf  docker cp dr1.cnf dr1:/etc/mysql/conf.d/docker.cnf |

**5）重启MySQL（dw1,dr1）：**

|  |
| --- |
| docker restart dw1 dr1 |

**6）进入dw1主机里面执行相关配置**

|  |
| --- |
| docker exec -it **dw1** bash  mysql -uroot -p123456 |

**7）创建用户：**

|  |
| --- |
| create user 'rep'@'%' identified by '123456'; |

**8）给该用户授予权限：**

|  |
| --- |
| grant replication slave on \*.\* to 'rep'@'%'; |

**9）刷新权限：**

|  |
| --- |
| flush privileges; |

至此：dw1里面已经创建了一个用户：rep 123456 拥有所有库，所有表replication slave

**10）查看状态后面备用：**

|  |
| --- |
| show master status; |



### 进入dr1配置

**1）进入容器并登录：**

|  |
| --- |
| docker exec -it dr1 bash  mysql -u root -p123456 |

**2）执行主从配置：**

|  |
| --- |
| change master to master\_host="**192.168.106.133**",master\_port=**3314**,master\_user="**rep**",master\_password="**123456**",master\_log\_file="**master.000001**",master\_log\_pos=**745;** |

master\_log\_file：该文件具体叫什么名称，需要从主机里面去看看上面【**查看状态**】

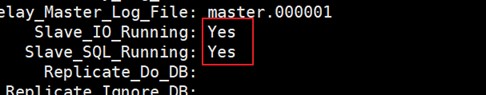
**3）启动主从：（在dr1里面执行）**

|  |
| --- |
| start slave ; |

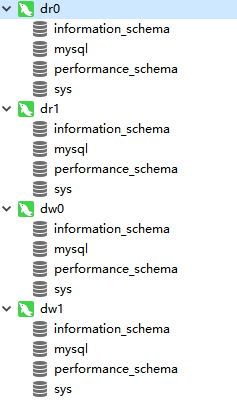
**4）查询主从的状态（dr1）**

|  |
| --- |
| show slave status \G; |

**5）成功的标志：**



### 使用Navicat连接测试



### 配置MyCAT数据源

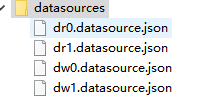
通过上面的配置，我们准备了两组主从，分别为：

1. dw0 dr0
2. dw1 dr1

接下来，我们要在MyCAT里面配置这四个数据源。

|  |
| --- |
| **添加dw0数据源**  /\*+ mycat:createDataSource{  "name":"dw0",  "password":"123456", "url":"jdbc:mysql://127.0.0.1:3312?useUnicode=true&serverTimezone=UTC&characterEncoding=UTF-8",  "user":"root",} \*/; |
| **添加dr0数据源**  /\*+ mycat:createDataSource{  "name":"dr0",  "password":"123456", "url":"jdbc:mysql://127.0.0.1:3313?useUnicode=true&serverTimezone=UTC&characterEncoding=UTF-8",  "user":"root",} \*/; |
| **添加dw1数据源**  /\*+ mycat:createDataSource{  "name":"dw1",  "password":"123456", "url":"jdbc:mysql://127.0.0.1:3314?useUnicode=true&serverTimezone=UTC&characterEncoding=UTF-8",  "user":"root",} \*/; |
| **添加dr1数据源**  /\*+ mycat:createDataSource{  "name":"dr1",  "password":"123456", "url":"jdbc:mysql://127.0.0.1:3315?useUnicode=true&serverTimezone=UTC&characterEncoding=UTF-8",  "user":"root",} \*/; |

**执行之后我们在MyCAT里面看到如下数据源的配置文件。**



### 配置MyCAT集群配置

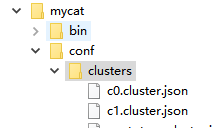
注意自动分片**默认要求集群名字以c为前缀，**数字为后缀：

1. c0就是分片表第一个节点；
2. c1就是第二个节点。

一般情况下我们使用默认的就可以了。

|  |
| --- |
| /\*! mycat:createCluster{  **"name":"c0",**  "masters":[  "dw0"  ],  "replicas":[  "dr0"  ]  } \*/; |
| /\*! mycat:createCluster{  **"name":"c1",**  "masters":[  "dw1"  ],  "replicas":[  "dr1"  ]  } \*/; |

创建完成之后查看MyCAT配置文件里面内容如下：



|  |
| --- |
| {  "clusterType":"MASTER\_SLAVE",  "heartbeat":{  "heartbeatTimeout":1000,  "maxRetryCount":3,  "minSwitchTimeInterval":300,  "showLog":false,  "slaveThreshold":0.0  },  "masters":[  **"dw0"**  ],  "maxCon":2000,  "name":"c0",  "readBalanceType":"BALANCE\_ALL",  "replicas":[  **"dr0"**  ],  "switchType":"SWITCH"  } |

## 全局表(广播表)配置

全局表：所有分片库中都有全量数据的表。分库分表的环境准备好之后，接下来我们在MyCAT里面执行相关的命令，就可以帮我们创建全局表。

### 创建数据库

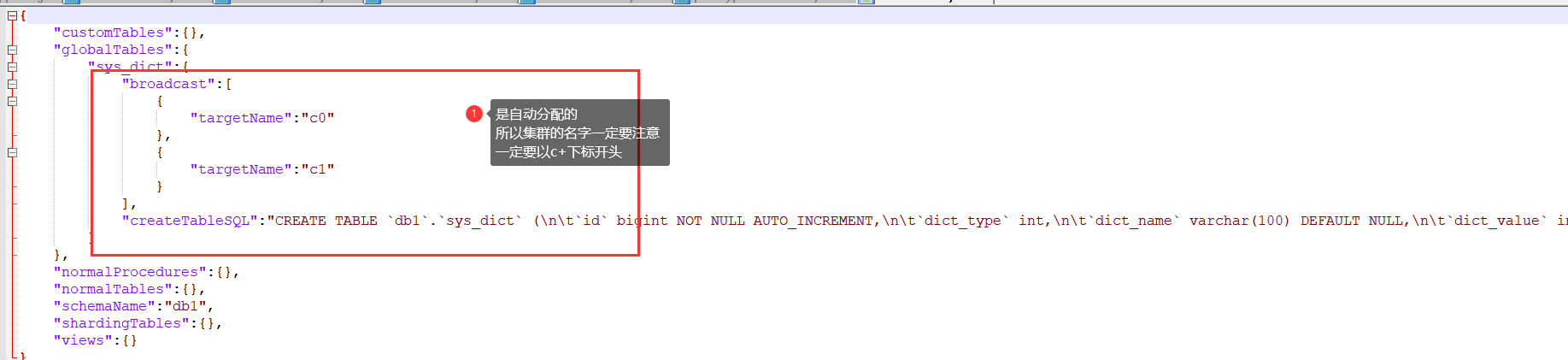
|  |
| --- |
| CREATE DATABASE db1 DEFAULT CHARACTER SET utf8mb4 COLLATE utf8mb4\_general\_ci; |

### 创建表

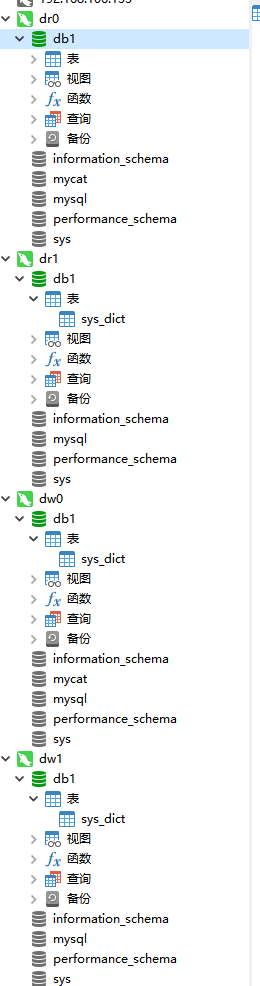
|  |
| --- |
| use db1;  CREATE TABLE `sys\_dict` (  `id` bigint NOT NULL AUTO\_INCREMENT,  `dict\_type` int ,  `dict\_name` varchar(100) DEFAULT NULL,  `dict\_value` int ,  PRIMARY KEY (`id`)  ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 **BROADCAST**; |
| 上面的SQL中有一个**BROADCAST** 这个就是全局表的标识。 |

### 查看结果

**MyCAT的配置文件中：**



**后端数据库中发现所有的表全部出现了。**

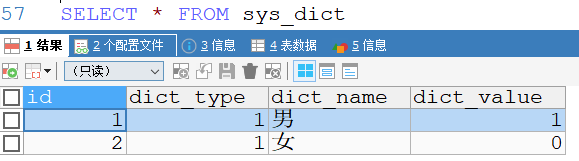


### 添加数据查看结果

|  |
| --- |
| INSERT INTO sys\_dict(dict\_type,dict\_name,dict\_value) VALUES(1,"男",1);  INSERT INTO sys\_dict(dict\_type,dict\_name,dict\_value) VALUES(1,"女",0); |
| 所有的库中都有的数据 |

### 查询数据查看结果

|  |
| --- |
| select \* from sys\_dict |



发现数据并没有重复。

## 分片表配置【重点】

关键字：dbpartition、tbpartitition、tbpartitions、dbpartitions。以上的运行成功必须是c0、c1的数据源配置没有问题才行。

### 创建表

|  |
| --- |
| CREATE TABLE orders(  ID BIGINT NOT NULL AUTO\_INCREMENT,  ORDER\_TYPE INT,  CUSTOMER\_ID INT,  AMOUNT DECIMAL(10,2),  PRIMARY KEY(ID)  ) ENGINE=INNODB DEFAULT CHARSET=utf8mb4  **dbpartition BY mod\_hash(CUSTOMER\_ID) tbpartition By mod\_hash(CUSTOMER\_ID)**  **tbpartitions 1 dbpartitions 2** |
| **dbpartition BY mod\_hash(CUSTOMER\_ID)：** 指定数据库的分片算法及使用哪一条数据进行分片HASH  **tbpartition BY mod\_hash(CUSTOMER\_ID) ：**指定表的分片算法及使用哪一条数据进行分片HASH  **tbpartitions 1** 表的分片数量  **dbpartitions 2** 数据库的分片数量 |

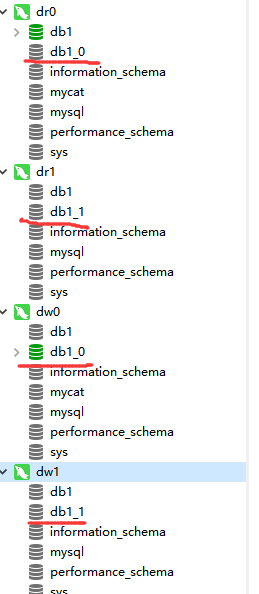
### 查看MyCAT生成的配置



### 添加数据

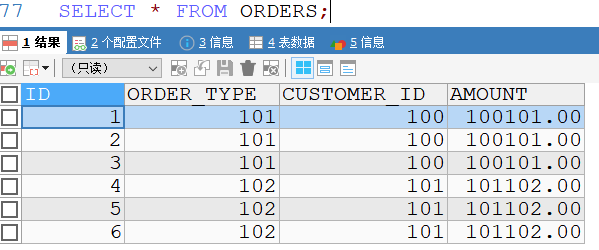
|  |
| --- |
| INSERT INTO ORDERS(ID,ORDER\_TYPE,CUSTOMER\_ID,AMOUNT) VALUES(1,101,100,100101);  INSERT INTO ORDERS(ID,ORDER\_TYPE,CUSTOMER\_ID,AMOUNT) VALUES(2,101,100,100101);  INSERT INTO ORDERS(ID,ORDER\_TYPE,CUSTOMER\_ID,AMOUNT) VALUES(3,101,100,100101);  INSERT INTO ORDERS(ID,ORDER\_TYPE,CUSTOMER\_ID,AMOUNT) VALUES(4,102,101,101102);  INSERT INTO ORDERS(ID,ORDER\_TYPE,CUSTOMER\_ID,AMOUNT) VALUES(5,102,101,101102);  INSERT INTO ORDERS(ID,ORDER\_TYPE,CUSTOMER\_ID,AMOUNT) VALUES(6,102,101,101102); |

### 查询后台物理库



从图中我们可以看到数据库生成了。打开数据库后，里面的表也生成了，里面的数据也分开了，并不在一个表里。

### MyCAT中查询



从上图中，我们发现查询的结果也帮我们合并了。

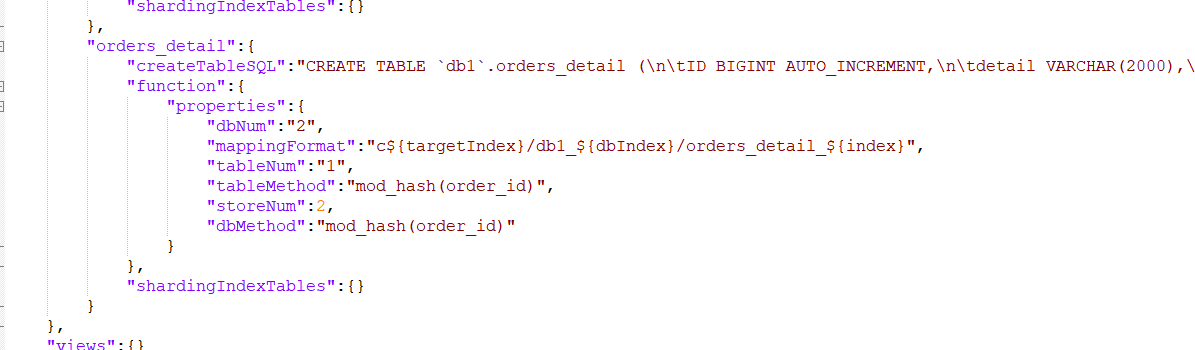
## ER表配置

说明：在1.6的版本中，我们ER表的配置有关系的数据必须存放在相同的库中，但是在2.0中不用了，MyCAT2自动帮我们优化了。上面我们创建了一张订单表，接下来我们创建一张订单详情表。

### 创建表

|  |
| --- |
| CREATE TABLE orders\_detail(  ID BIGINT AUTO\_INCREMENT,  detail VARCHAR(2000),  order\_id BIGINT,  PRIMARY KEY(ID)  ) ENGINE=INNODB DEFAULT CHARSET=utf8mb4  **dbpartition BY mod\_hash(order\_id) tbpartition By mod\_hash(order\_id)**  **tbpartitions 1 dbpartitions 2** |
| **dbpartition BY mod\_hash(order\_id) ：**指定数据库的分片算法及使用哪一条数据进行分片HASH  **tbpartition BY mod\_hash(order\_id) ：**指定表的分片算法及使用哪一条数据进行分片HASH  **tbpartitions 1 表的分片数量**  **dbpartitions 2 数据库的分片数量** |

### 查看MyCAT生成的配置

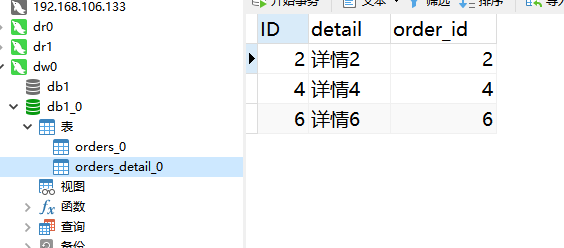


从上图中我们可以出已经放到分片表里面了。

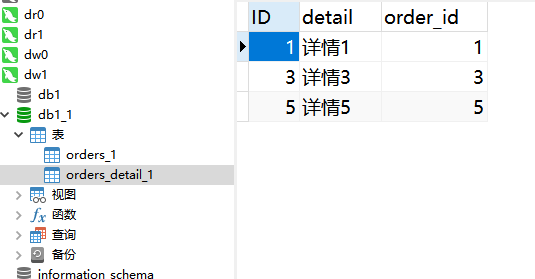
### 添加数据

|  |
| --- |
| INSERT INTO orders\_detail VALUES(1,"详情1",1);  INSERT INTO orders\_detail VALUES(2,"详情2",2);  INSERT INTO orders\_detail VALUES(3,"详情3",3);  INSERT INTO orders\_detail VALUES(4,"详情4",4);  INSERT INTO orders\_detail VALUES(5,"详情5",5);  INSERT INTO orders\_detail VALUES(6,"详情6",6); |

### 查询后台物理库



**从上图可知，dw0里面只存了三条。**



**发现dw1里面也存放了三条。**

### MyCAT中关联查询

|  |
| --- |
| select \* from orders o inner join orders\_detail od on(o.id=od.order\_id) |
|  |

### 疑问

从物理库中我们看到一个库里面的详情数据和定义数据不配套，那是为什么呢？在1.6的版本里面是不允许的。

接下来我们说明下原因：

1）MyCAT2在涉及两个表的JOIN分片字段等价关系的时候可以完成JOIN的下推。

2）MyCAT2无需要指定ER表，是自动识别的。

3）查询配置的表是否具有ER关系 使用如下注释：

|  |
| --- |
| /\*+ mycat:showErGroup{}\*/ |

**结果如下：**



上面的group\_id 表示相同的组，该组中的表具有相同的存储分布(在MyCAT2中它是这么规定的)。

## 分片算法简介

### 取模哈希分片 MOD\_HASH

1）如果分片值是字符串则先对字符串进行Hash转换为数值类型

2）分库键和分表键是同键

3）分表下标=分片值%(分库数量\*分表数量)

4）分库下标=分表下标/分表数量

5）分库键和分表键是不同键

6）分表下标= 分片值%分表数量

7）分库下标= 分片值%分库数量

|  |
| --- |
| create table travelrecord (  ....  ) ENGINE=InnoDB DEFAULT CHARSET=utf8  dbpartition by MOD\_HASH (id) dbpartitions 6  tbpartition by MOD\_HASH (id) tbpartitions 6; |

### 范围哈希分片 RANGE\_HASH

1）RANGE\_HASH(字段1, 字段2, 截取开始下标)

2）仅支持数值类型，字符串类型

3）当时字符串类型时候，第三个参数生效

4）计算时候优先选择第一个字段,找不到选择第二个字段

5）如果是字符串则根据下标截取其后部分字符串，然后该字符串hash成数值

6）根据数值按分片数取余

7）要求截取下标不能少于实际值的长度

8）两个字段的数值类型要求一致

|  |
| --- |
| create table travelrecord(  ...  )ENGINE=InnoDB DEFAULT CHARSET=utf8  dbpartition by RANGE\_HASH(id,user\_id,3) dbpartitions 3  tbpartition by RANGE\_HASH(id,user\_id,3) tbpartitions 3; |

### 字符串哈希分片 UNI\_HASH

1）如果分片值是字符串则先对字符串进行hash转换为数值类型

2）分库键和分表键是同键

3）分库下标=分片值%分库数量

4）分表下标=(分片值%分库数量)\*分表数量+(分片值/分库数量)%分表数量

5）分库键和分表键是不同键

6）分表下标= 分片值%分表数量

7）分库下标=分片值%分库数量

|  |
| --- |
| create table travelrecord (  ....  ) ENGINE=InnoDB DEFAULT CHARSET=utf8  dbpartition by UNI\_HASH (id) dbpartitions 6  tbpartition by UNI\_HASH (id) tbpartitions 6; |

### 日期哈希分片 YYYYDD

1）仅用于分库

2）DD是一年之中的天数

3）(YYYY\*366+DD)%分库数

|  |
| --- |
| create table travelrecord (  ....  ) ENGINE=InnoDB DEFAULT CHARSET=utf8  dbpartition by YYYYDD(xxx) dbpartitions 8  tbpartition by xxx(xxx) tbpartitions 12; |

# 【掌握】MyCAT2全局ID的生成方式

## 为什么要全局ID

在复杂的分布式系统中，需要对大量的数据和消息进行唯一标识。如在阿里，淘宝，支付，等系统中，数据日渐增长，对数据分库分表后需要有一个唯一ID来标识一条数据或消息；还有如美团和饿了吗的骑手ID 商家ID  优惠券ID等，从以上可以得出，一个能够生成全局唯一ID的系统是非常必要的。

在MyCAT2中，自动默认使用雪花片法生成全局序列号。

如果不需要MyCAT默认全局序列，可以通过配置关闭自动加全局序列；建表语句方式关闭全局序列。

如果不需要使用MyCAT的自增序列，而使用MySQL本身的自增主键的功能。需要在配置中更改对应的建表SQL。不设置auto\_increment关键字，这样 MyCAT就不认为这个表有自增主键的功能。就不会使用MyCAT全局序列号，这样，对应的插入SQL在MySQL中去处理，由MySQL的自增主键功能补全。

雪花算法：引入了时间戳的ID保持自增的分布式ID生成算法。

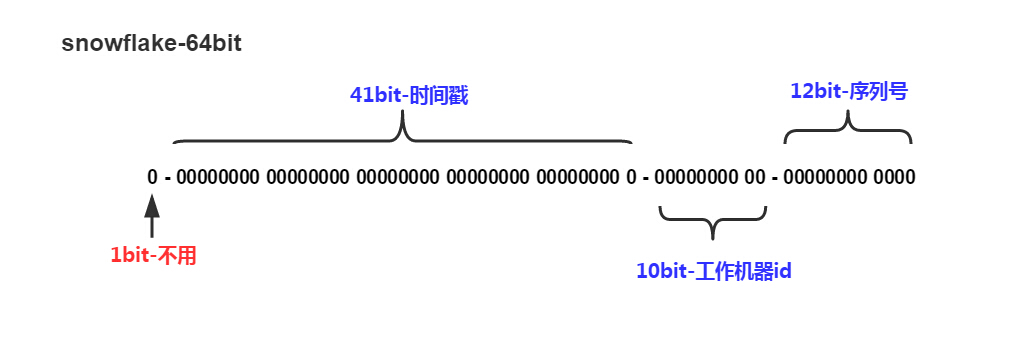
## 雪花算法原理说明

Twitter开源分布式生成ID算法。

1）优点：基本解决了所有问题

2）缺点：每个节点时间可能不同，生成ID是整体趋势递增的

Snowflake的结构如下(每部分用-分开):



图中的信息如下：

1）第一位未使用，因为二进制中最高位是符号位同，1表示负数，0表示正数。ID不可能为负数。

2）时间戳：41位，最后为2的41次方，大概69.73年，10==2+8

3）然后是5位datacenterId和5位workerId(10位的长度最多支持部署1024个节点） **也可以是3位datacenterId和7位workerId**

4）最后12位是毫秒内的计数（12位的计数顺序号支持每个节点每毫秒产生4096个ID序号）每秒生成409.6万个【可怕】

一共加起来刚好64位，为一个Long型。(转换成字符串长度为18)

我们可以使用<https://tool.oschina.net/hexconvert/>  进制转化工具测试。

**优点**：

1）毫秒数在高位，自增序列在低位，整个ID都是趋势递增的。

2）不依赖数据库等第三方系统，以服务的方式部署，稳定性更高，生成ID的性能也是非常高的。

3）可以根据自身业务特性分配bit位，非常灵活。

**缺点**：强依赖机器时钟，如果机器上时钟回拨，会导致发号重复或者服务会处于不可用状态。

**结论**：适用于大规模分布式架构。

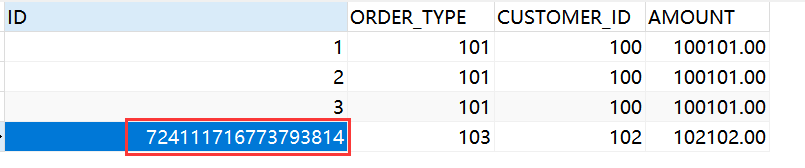
## 使用默认的雪花算法验证

如果不需要使用MyCAT的自增序列，而使用MySQL本身的自增主键的功能，需要在配置中更改对应的建表SQL,不设置AUTO\_INCREMENT关键字，这样MyCAT就不认为这个表有自增主键的功能，就不会使用MyCAT的全局序列号。对应的插入SQL在MySQL处理，由MySQL的自增主键功能补全自增值。

### 向订单表里面添加数据

|  |
| --- |
| INSERT INTO ORDERS(ORDER\_TYPE,CUSTOMER\_ID,AMOUNT) VALUES(103,102,102102); |

上面的SQL里面没有使用添加ID，那么后台是怎么处理的呢？查看后台数据发现使用的雪花算法

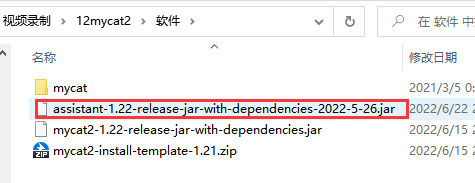
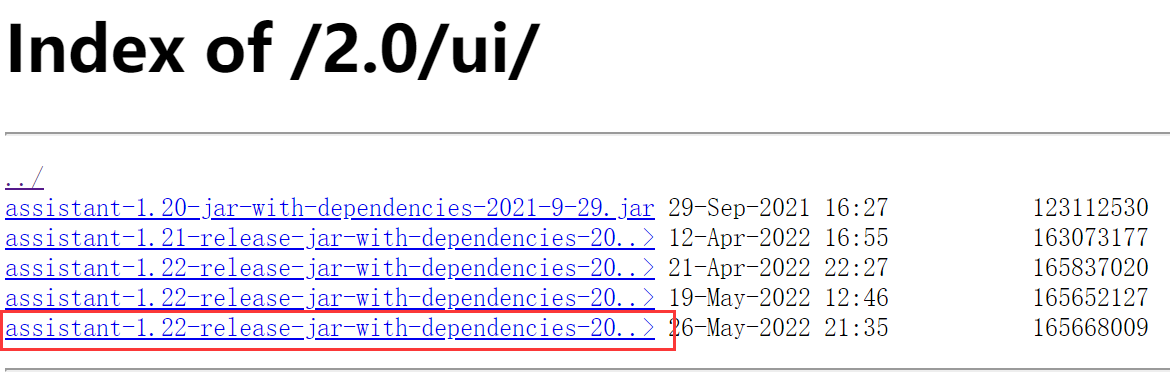


# 【了解】MyCAT2工具的使用

就是一个用来连接MyCAT修改MyCAT配置的工具。

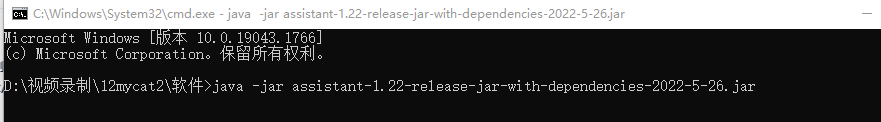
## 下载

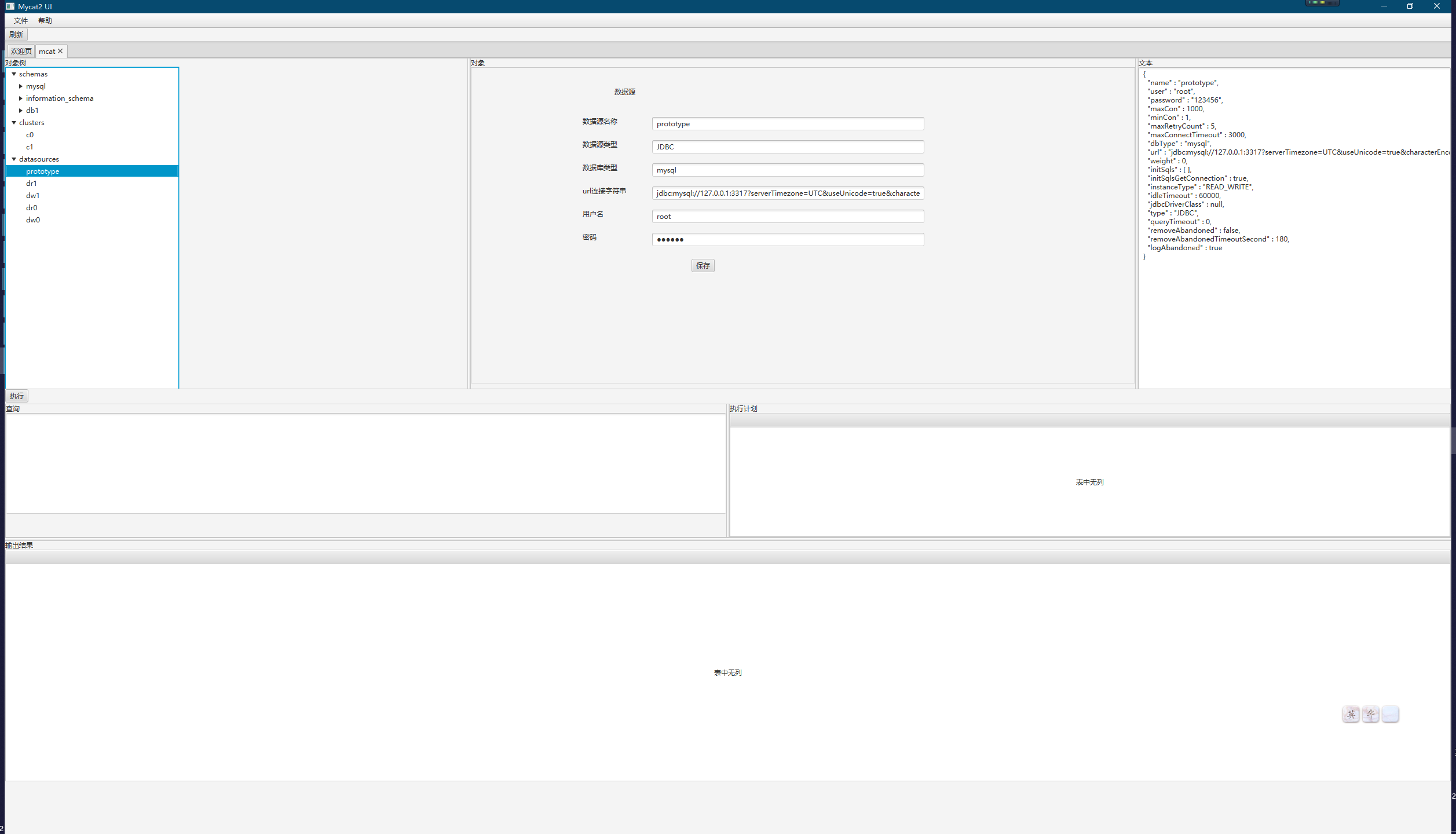
<http://dl.mycat.org.cn/2.0/ui/>



## 运行

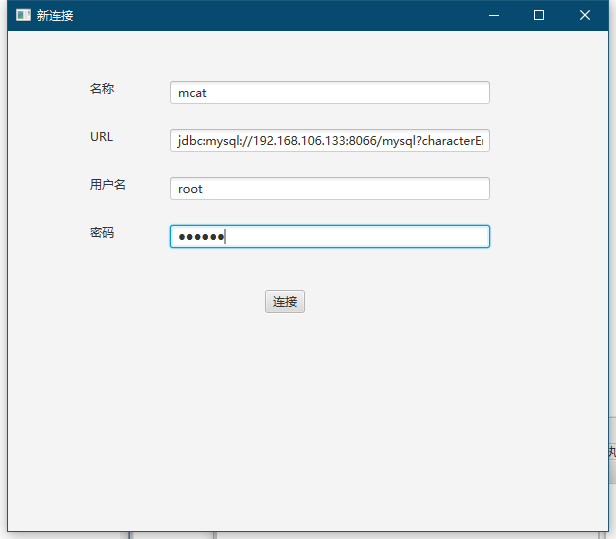
前提，安装的电脑必须安装JDK1.8及以上的版本





## 创建连接并操作

**文件-新连接：**



连接成功之后我们可以看到我们之前配置的所有MyCAT的配置：

