

Java 课程系列之 Mycat

尚硅谷 Java 研究院

版本: V2.0

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。

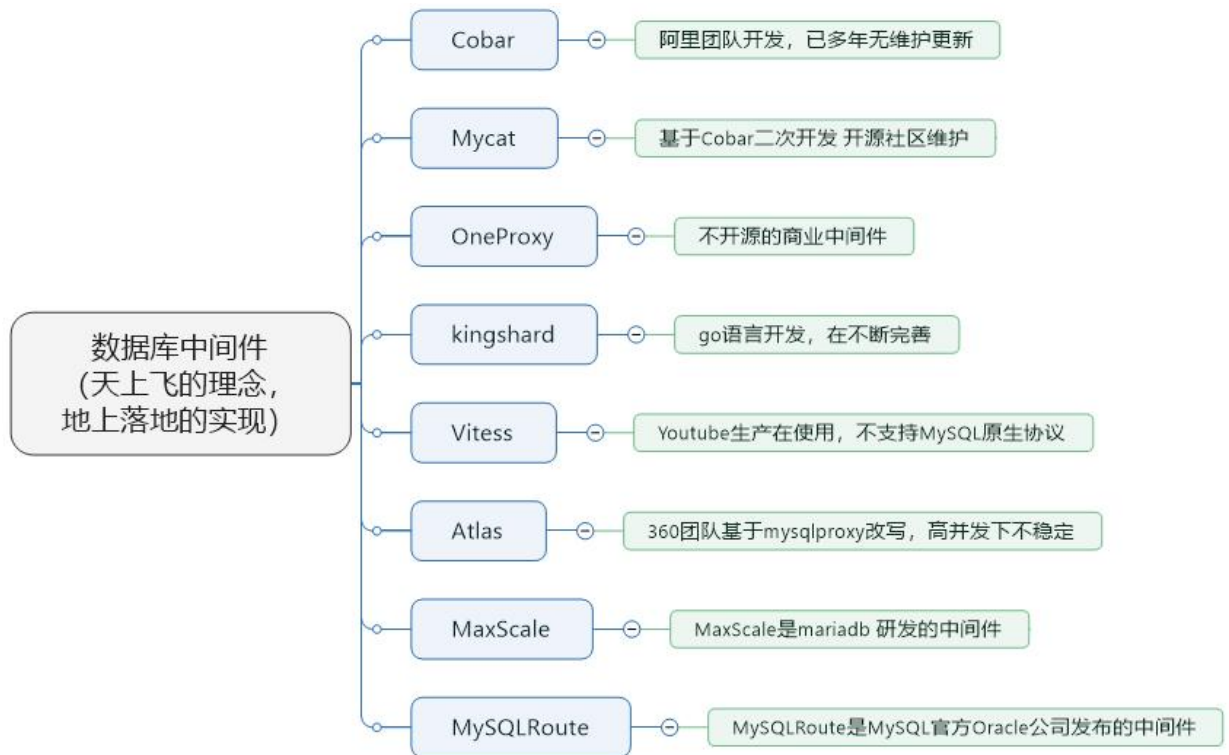
例子：Tomcat，web中间件。

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

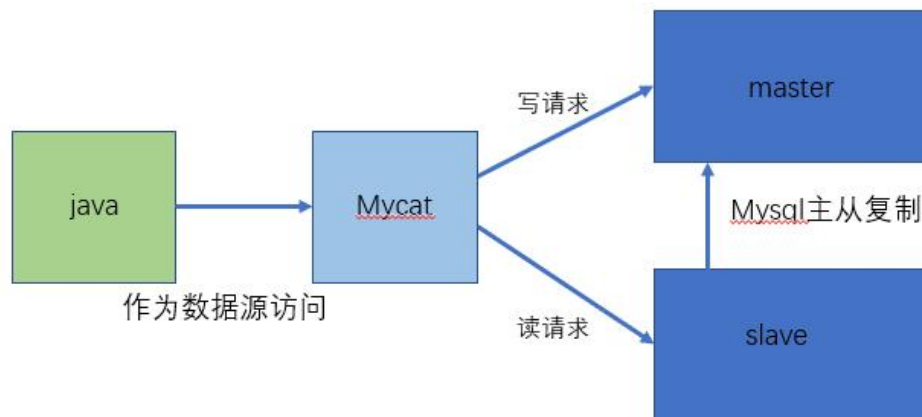
⑧ MySQLRoute是MySQL官方Oracle公司发布的中间件

3、Mycat的官网

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

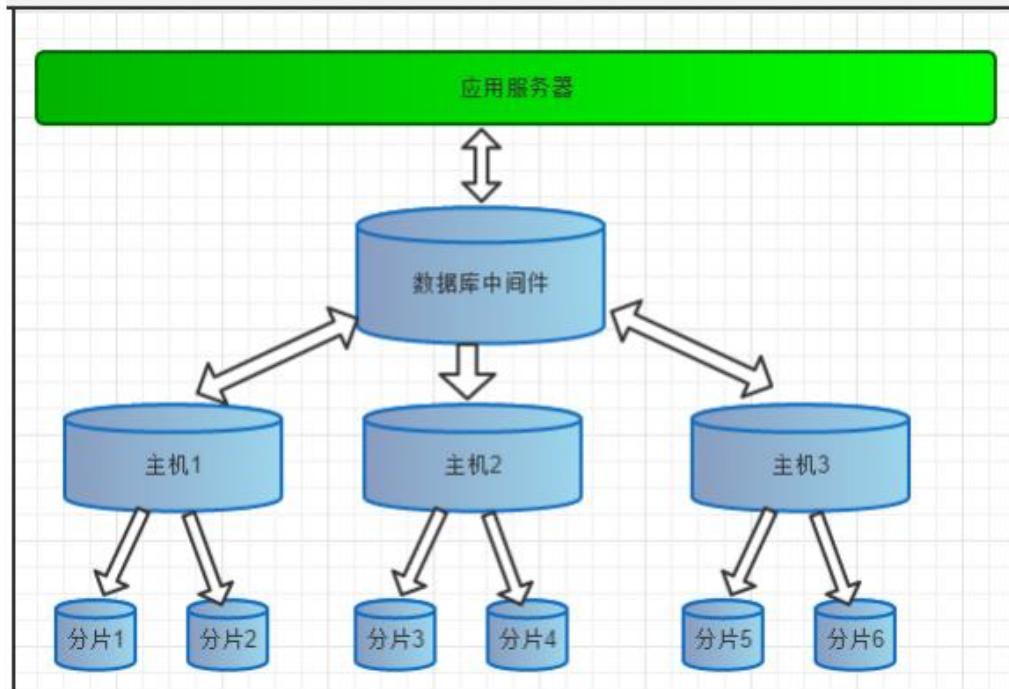
1.2 干什么

1、读写分离

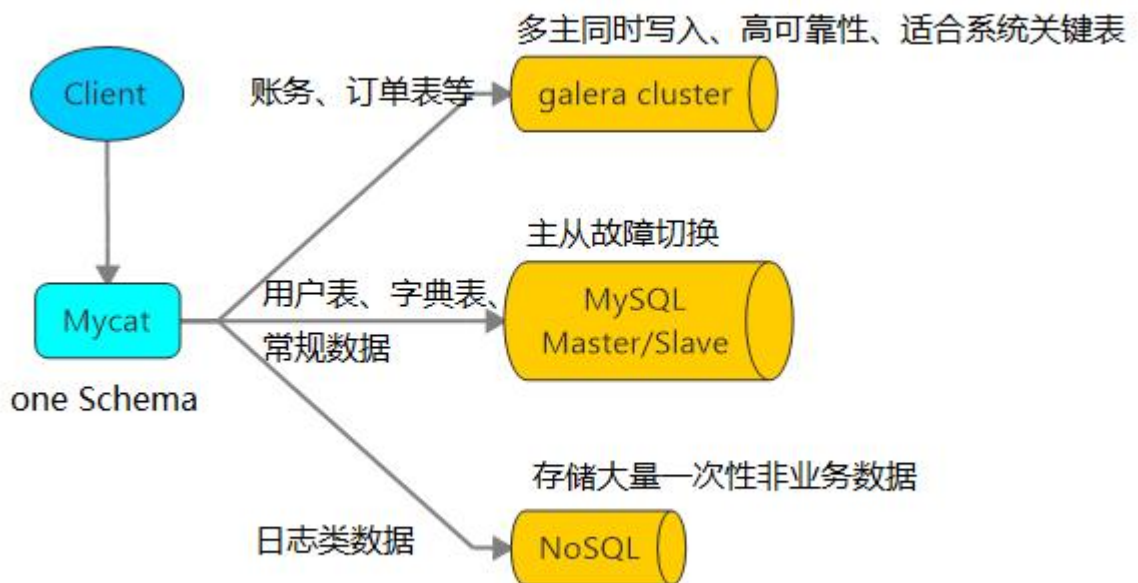


2、数据分片

垂直拆分（分库）、水平拆分（分表）、垂直+水平拆分（分库分表）



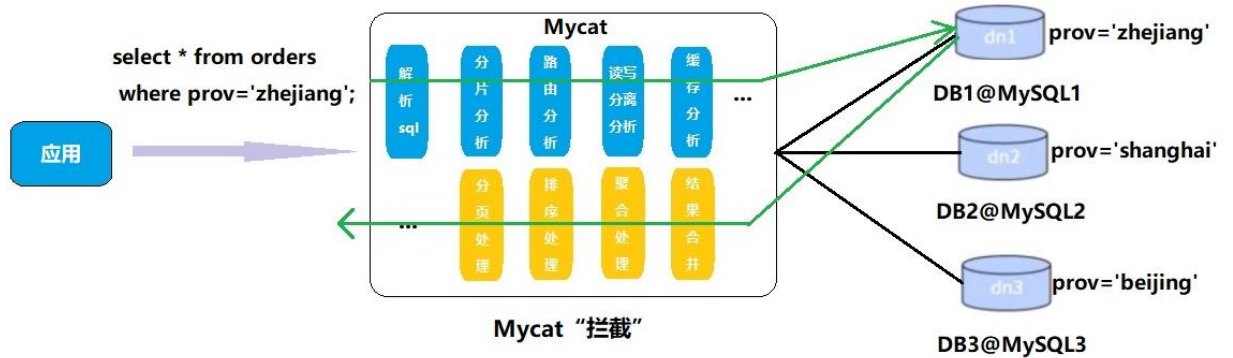
3、多数据源整合



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。

1.4 Mycat1.x 与 Mycat2 功能对比

1、1.x 与 2.0 功能对比图

功能	1.6	2
多语句	不支持	支持
blob 值	支持一部分	支持
全局二级索引	不支持	支持
任意跨库 join (包含复杂查询)	catlet 支持	支持
关联子查询	不支持	支持一部分
分库同时分表	不支持	支持
存储过程	支持固定形式的	支持更多
支持逻辑视图	不支持	支持
支持物理视图	支持	支持

批量插入	不支持	支持
执行计划管理	不支持	支持
路由注释	支持	支持
集群功能	支持	支持更多集群类型
自动 hash 分片算法	不支持	支持
支持第三方监控	支持 mycat-web	支持普罗米斯, kafka 日志等监控
流式合并结果集	支持	支持
范围查询	支持	支持
单表映射物理表	不支持	支持
XA 事务	弱 XA	支持, 事务自动恢复
支持 MySQL8	需要更改 mysql8 的服务器配置支持	支持
虚拟表	不支持	支持
joinClustering	不支持	支持
union all 语法	不支持	支持
BKAJoin	不支持	支持
优化器注释	不支持	支持
ER 表	支持	支持
全局序列号	支持	支持
保存点	不支持	支持

(1) 多语句：解析器会对 SQL 进行拆分依次执行（默认配置）

(2) blob 值：BLOB (binary large object) 二进制大对象，是一个可以存储二进制文件的容器。

- (3) 全局二级索引：使用全局二级索引后,能有效减少全表扫描,对于减少连接使用,减少计算节点与存储节点的数据传输有帮助.
- (4) 关联子查询：支持不能消除关联的关联子查询
- (5) 分库同时分表：把分库分表合一，统一规划
- (6) 存储过程：存储过程支持多结果集返回、支持接收 `affectRow`
- (7) 支持批量插入：支持 `rewriteInsertBatchedStatementBatch` 参数,用于提高批量插入性能（只有把 `rewriteBatchedStatements` 参数置为 `true`，MySQL 驱动才会帮你批量执行 SQL）
- (8) 支持执行计划管理：Mycat2 的执行计划管理主要作用是管理执行计划,加快 SQL 到执行计划的转换,并且提供一个方式可以从持久层读取自定义的执行计划。
- (9) 自动 hash 分片算法：由 1.6 版本的手动配置算法，到 2.0 的自动 hash 分片
- (10) 单表映射物理表：使用自动化建表语句创建测试的物理库物理表,它会自动生成配置文件,然后通过查看本地的配置文件,观察它的属性

2、映射模型区别

Mycat1.6

分库

logical table -> 多个 `dataNode(cluster,schema)`
↓
`datasource`

分表

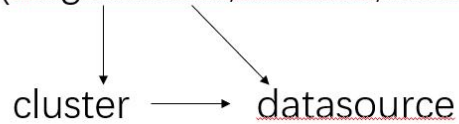
logical table -> 一个 `dataNode(cluster,schema)` + 多个 `tableName`
↓
`datasource`

单表

logical table -> 一个 `dataNode(cluster,schema)`

Mycat2

logical table -> partition(targetName, schema, table)



第二章 安装启动

2.1 安装

1、下载安装包

下载对应的 tar 安装包, 以及对应的 jar 包

tar (zip) 包 :

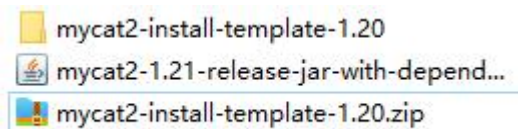
<http://dl.mycat.org.cn/2.0/install-template/mycat2-install-template-1.20.zip>

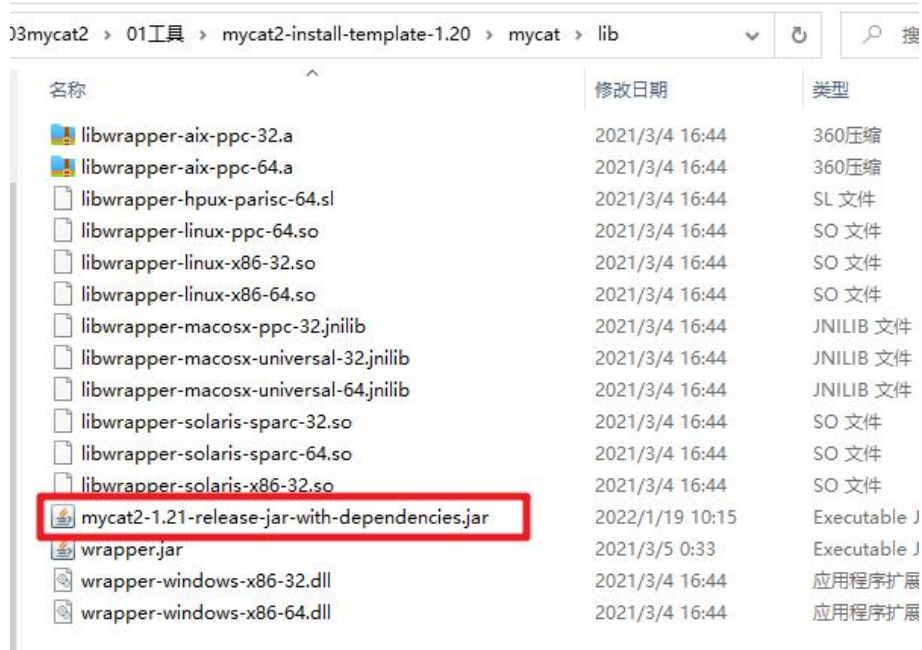
jar 包 :

<http://dl.mycat.org.cn/2.0/1.21-release/> (下载最新的 jar 包)

下载所需的 mycat2 的 fat jar 一般大小为 100mb 的一个 jar 文件

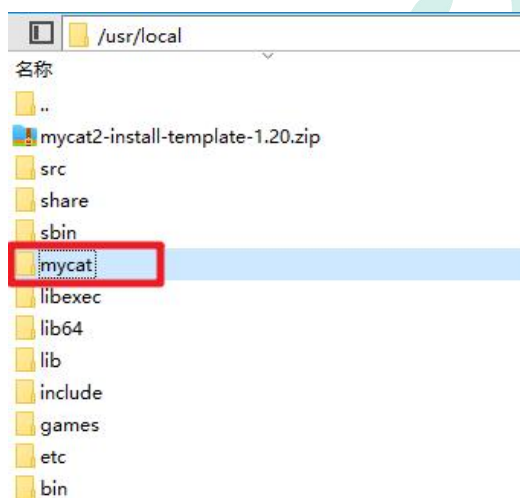
把这个 jar 放进解压的 tar 中的 mycat\lib 文件夹下





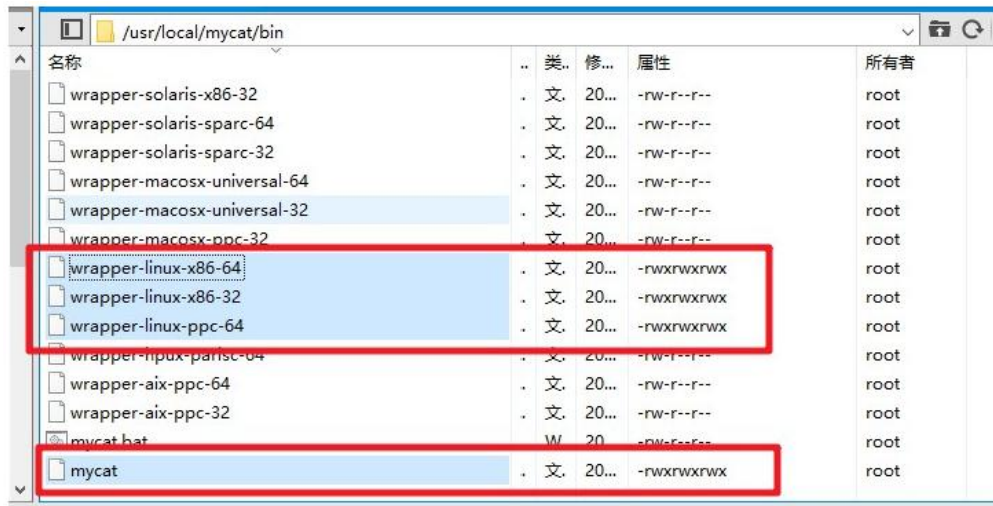
2、解压后即可使用

把整合好的文件夹拷贝到 linux 下 /usr/local/



3、修改文件夹及以下文件的权限

修改成最高权限，否则运行启动命令时，会因权限不足而报错



名称	类	修...	属性	所有者
wrapper-solaris-x86-32	文	20...	-rw-r--r--	root
wrapper-solaris-sparc-64	文	20...	-rw-r--r--	root
wrapper-solaris-sparc-32	文	20...	-rw-r--r--	root
wrapper-macosx-universal-64	文	20...	-rw-r--r--	root
wrapper-macosx-universal-32	文	20...	-rw-r--r--	root
wrapper-macosx-ppc-32	文	20...	-rw-r--r--	root
wrapper-linux-x86-64	文	20...	-rwxrwxrwx	root
wrapper-linux-x86-32	文	20...	-rwxrwxrwx	root
wrapper-linux-ppc-64	文	20...	-rwxrwxrwx	root
wrapper-mpux-parisc-64	文	20...	-rw-r--r--	root
wrapper-aix-ppc-64	文	20...	-rw-r--r--	root
wrapper-aix-ppc-32	文	20...	-rw-r--r--	root
mycat.bat	W	20...	-rw-r--r--	root
mycat	文	20...	-rwxrwxrwx	root

2.2 启动

1、在mycat连接的mysql数据库里添加用户

创建用户，用户名为mycat，密码为123456，赋权限，如下：

```
CREATE USER 'mycat'@'%' IDENTIFIED BY '123456';
```

——必须要赋的权限mysql8才有的

```
GRANT XA_RECOVER_ADMIN ON *.* TO 'root'@'%' ;
```

——视情况赋权限

```
GRANT ALL PRIVILEGES ON *.* TO 'mycat'@'%' ;
```

```
flush privileges;
```

...

2、修改mycat的prototype的配置

启动mycat之前需要确认prototype数据源所对应的mysql数据库配置，修改对应的user(用户)，password(密码)，url中的ip

```
...  
  
vim conf/datasources/prototypeDs.datasource.json  
  
{  
  
    "dbType":"mysql",  
    "idleTimeout":60000,  
    "initSqs":[],  
    "initSqsGetConnection":true,  
    "instanceType":"WRITE",  
    "maxCon":1000,  
    "maxConnectTimeout":3000,  
    "maxRetryCount":5,  
    "minCon":1,  
    "name":"prototypeDs",  
    "password":"123123",  
    "type":"JDBC",  
  
    "url":"jdbc:mysql://localhost:3306/mydb1?useUnicode=true&serverTimezone=Asia/Shanghai&characterEncoding=UTF-8",  
    "user":"root",  
    "weight":0  
}  
  
...
```

3、验证数据库访问情况

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 192.168.140.100 -P 3306
```

```
mysql -uroot -p123123 -h 192.168.140.99 -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mycat

linux启动命令

```
cd mycat/bin
./mycat start
./mycat status
./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（暂未实现）
./mycat remove 取消随系统自动启动（暂未实现）
./mycat restart 重启服务
./mycat pause 暂停
./mycat status 查看启动状态...
```

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
```

#常用命令如下:

show database

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

help;

```
mysql> help;

For information about MySQL products and services, visit:
  http://www.mysql.com/
For developer information, including the MySQL Reference Manual, visit:
  http://dev.mysql.com/
To buy MySQL Enterprise support, training, or other products, visit:
  https://shop.mysql.com/

List of all MySQL commands:
Note that all text commands must be first on line and end with ';':
?          (?) Synonym for 'help'.
clear      (c) Clear the current input statement.
connect    (r) Reconnect to the server. Optional arguments are db and host.
delimiter (d) Set statement delimiter.
edit       (e) Edit command with $EDITOR.
ego        (G) Send command to mysql server, display result vertically.
exit       (q) Exit mysql. Same as quit.
go         (g) Send command to mysql server.
help       (h) Display this help.
nopager    (n) Disable pager, print to stdout.
notee      (t) Don't write into outfile.
pager      (P) Set PAGER [to_pager]. Print the query results via PAGER.
print      (p) Print current command.
prompt     (R) Change your mysql prompt.
quit       (q) Quit mysql.
rehash     (#) Rebuild completion hash.
source     (.) Execute an SQL script file. Takes a file name as an argument.
status     (s) Get status information from the server.
system     (!) Execute a system shell command.
tee        (T) Set outfile [to outfile]. Append everything into given outfile.
use        (u) Use another database. Takes database name as argument.
charset    (C) Switch to another charset. Might be needed for processing binlog with multi-byte charsets.
warnings   (W) Show warnings after every statement.
nowarning  (w) Don't show warnings after every statement.
resetconnection(x) Clean session context.
query_attributes Sets string parameters (name1 value1 name2 value2 ...) for the next query to pick up.

For server side help, type 'help contents'
```

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
```

第三章 Mycat2 相关概念

3.1 概念描述

1、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。

解读: 分库: 一个电商项目, 分为用户库、订单库等等。

分表: 一张订单表数据数百万, 达到 MySQL 单表瓶颈, 分到多个数据库中的多张表

2、逻辑库

数据库代理中的数据库, 它可以包含多个逻辑表。

解读: Mycat 里定义的库, 在逻辑上存在, 物理上在 MySQL 里并不存在。有可能是多个 MySQL 数据库共同组成一个逻辑库。类似多个小孩叠罗汉穿上外套, 扮演一个大人。

3、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)

解读: Mycat 里定义的表, 在逻辑上存在, 可以映射真实的 MySQL 数据库的表。可以一对一, 也可以一对多。

4、物理库

数据库代理连接的数据库中的库

解读: MySQL 真实的数据库。

5、物理表

数据库代理连接的数据库中的表

解读：MySQL 真实的数据库中的真实数据表。

6、拆分键

即分片键, 描述拆分逻辑表的数据规则的字段

解读：比如订单表可以按照归属的用户 id 拆分, 用户 id 就是拆分键

7、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

解读：多个物理分表里的数据汇总就是逻辑表的全部数据

8、物理分库

一般指包含多个物理分表的库

解读：参与数据分片的实际数据库

9、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

解读：分库是个动作, 需要多个数据库参与。就像多个数据库是多个盘子, 分库就是把一串数据葡萄, 分到各个盘子里, 而查询数据时, 所有盘子的葡萄又通过 Mycat2 组成了完整的一串葡萄。

10、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

解读：安按照规则拆分数据, 上个例子中的那串葡萄。

11、单表

没有分片, 没有数据冗余的表,

解读：没有拆分数据，也没有复制数据到别的库的表。

12、全局表, 广播表

每个数据库实例都冗余全量数据的逻辑表。

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

解读：例如系统中翻译字段的字典表，每个分片表都需要完整的字典数据翻译字段。

13、ER 表

狭义指父子表中的子表, 它的分片键指向父表的分片键, 而且两表的分片算法相同

广义指具有相同数据分布的一组表

解读：关联别的表的子表，例如：订单详情表就是订单表的 ER 表

14、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。

解读：集群就是高可用、负载均衡的代名词

15、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

解读：Mycat 通过数据源连接 MySQL 数据库

16、原型库(prototype)

原型库是 Mycat2 后面的数据库, 比如 mysql 库

解读：原型库就是存储数据的真实数据库，配置数据源时必须指定原型库

3.2 配置文件

1、服务 (server)

服务相关配置

(1) 所在目录

mycat/conf

```
/usr/local/mycat/conf
[root@atguigu01 conf]# ll
总用量 32
drwxr-xr-x. 2 root root 116 3月 28 11:17 clusters
drwxr-xr-x. 2 root root 209 3月 28 11:14 datasources
-rw-r--r--. 1 root root 3338 3月 5 2021 dbseq.sql
-rw-r--r--. 1 root root 316 11月 2 14:26 logback.xml
-rw-r--r--. 1 root root 0 3月 5 2021 mycat.lock
drwxr-xr-x. 2 root root 79 3月 28 10:59 schemas
drwxr-xr-x. 2 root root 6 6月 28 2021 sequences
-rw-r--r--. 1 root root 776 12月 28 13:54 server.json
-rw-r--r--. 1 root root 1643 3月 5 2021 simplelogger.properties
drwxr-xr-x. 2 root root 233 6月 28 2021 sql
drwxr-xr-x. 2 root root 6 6月 28 2021 sqlcaches
-rw-r--r--. 1 root root 49 3月 5 2021 state.json
drwxr-xr-x. 2 root root 28 3月 28 16:20 users
-rw-r--r--. 1 root root 211 3月 5 2021 version.txt
-rw-r--r--. 1 root root 4165 1月 13 20:49 wrapper.conf
```

默认配置即可

2、用户 (user)

配置用户相关信息

(1) 所在目录

mycat/conf/users

(2) 命名方式

{用户名}.user.json

(3) 配置内容

```
vim mycat/conf/users/root.user.json

{
    "ip":null,
    "password":"123456",
    "transactionType":"xa",
    "username":"root",
    "isolation":3
}
```

#字段含义

#ip: 客户端访问ip, 建议为空, 填写后会对客户端的ip进行限制

username: 用户名

password: 密码

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

transactionType: 事务类型

可选值:

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

xa 事务, 需要确认存储节点集群类型是否支持 XA

可以通过语句实现切换

```
set transaction_policy = 'xa'
```

```
set transaction_policy = 'proxy'  
可以通过语句查询  
SELECT @@transaction_policy
```

3、数据源 (datasource)

配置Mycat连接的数据源信息

(1) 所在目录

mycat/conf/datasources

(2) 命名方式

{数据源名字}.datasource.json

(3) 配置内容

```
vim mycat/conf/datasources/ prototype. datasources.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  
}
```

#字段含义

dbType: 数据库类型, mysql

name: 用户名

password: 密码

type: 数据源类型, 默认 JDBC

url: 访问数据库地址

idleTimeout: 空闲连接超时时间

initSqls: 初始化sql

initSqlsGetConnection: 对于 jdbc 每次获取连接是否都执行 initSqls

instanceType: 配置实例只读还是读写

可选值:

READ_WRITE, READ, WRITE

#weight : 负载均衡权重

连接相关配置

"maxCon": 1000,

"maxConnectTimeout": 3000,

"maxRetryCount": 5,

"minCon": 1,

4、集群 (cluster)

配置集群信息

(4) 所在目录

mycat/conf/clusters

(5) 命名方式

{集群名字}.cluster.json

(6) 配置内容

```
vim mycat/conf/clusters/prototype.cluster.json

{
    "clusterType": "MASTER_SLAVE",
    "heartbeat": {
        "heartbeatTimeout": 1000,
        "maxRetryCount": 3, //2021-6-4前是maxRetry, 后更正为
maxRetryCount
        "minSwitchTimeInterval": 300,
        "slaveThreshold": 0
    },
    "masters": [ //配置多个主节点, 在主挂的时候会选一个检测存活的数据源作
为主节点
        "prototypeDs"
    ],
}
```

```
"replicas":[//配置多个从节点

    "xxxx"

],

"maxCon":200,

"name":"prototype",

"readBalanceType":"BALANCE_ALL",

"switchType":"SWITCH"

//////////////////////////////////////可选
//////////////////////////////////////

,

"timer":{ //MySQL集群心跳周期, 配置则开启集群心跳, Mycat主动检测主从延迟以及高可用主从切换

    "initialDelay": 30,

    "period":5,

    "timeUnit":"SECONDS"

},

//readBalanceName:"BALANCE_ALL",

//writeBalanceName:"BALANCE_ALL",

}
```

#字段含义

clusterType: 集群类型

可选值:

SINGLE_NODE:单一节点

MASTER_SLAVE:普通主从

GARELA_CLUSTER:garela cluster/PXC 集群

MHA: MHA 集群

MGR: MGR 集群

readBalanceType: 查询负载均衡策略

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

switchType: 切换类型

可选值:

NOT_SWITCH: 不进行主从切换

SWITCH: 进行主从切换

4、逻辑库表 (schema)

配置逻辑库表, 实现分库分表

(7) 所在目录

mycat/conf/schemas

(8) 命名方式

{库名}. schema. json

(9) 配置内容

```
vim mycat/conf/schemas/mydb1.schema.json

#库配置
{
    "schemaName": "mydb",
    "targetName": "prototype"
}

# schemaName: 逻辑库名

# targetName: 目的数据源或集群

targetName自动从prototype目标加载test库下的物理表或者视图作为单表, prototype
必须是mysql服务器

#单表配置
{
    "schemaName": "mysql-test",
    "normalTables": {
        "role_edges": {
            "createTableSQL": null, //可选
            "locality": {
                "schemaName": "mysql", //物理库, 可选
                "tableName": "role_edges", //物理表, 可选
                "targetName": "prototype" //指向集群, 或者数据源
            }
        }
    }
    .....
}

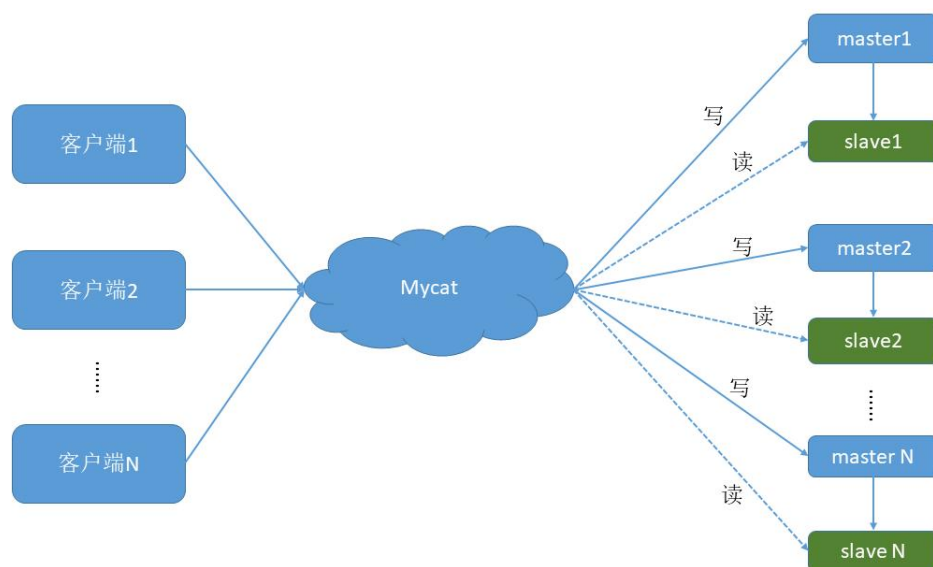
#详细配置见分库分表
```

第四章 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。

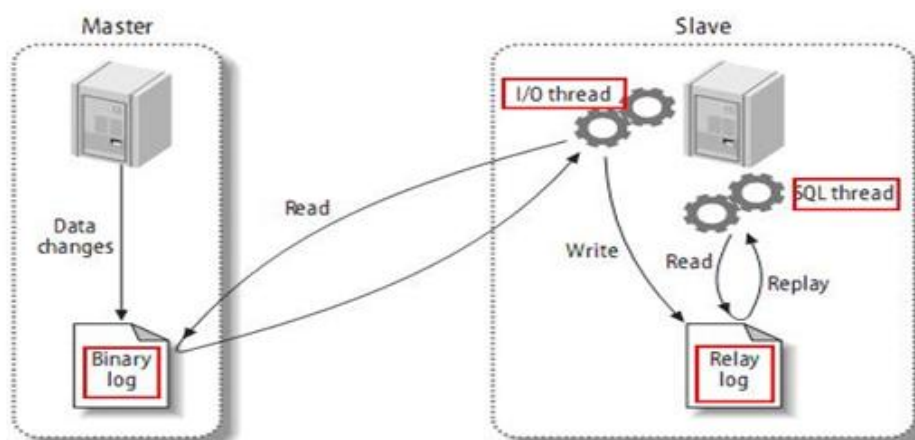
4.1 搭建一主一从

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 (atguigu01)

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

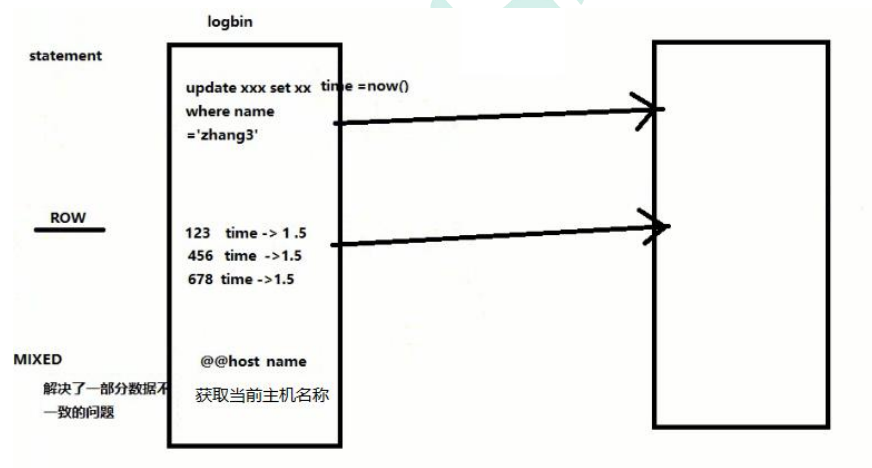
#设置需要复制的数据库

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(atguigu02)

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

```
CREATE USER 'slave2'@'%' IDENTIFIED BY '123123';
```

```
GRANT REPLICATION SLAVE ON *.* TO 'slave2'@'%' ;
```

#此语句必须执行。否则见下面。

```
ALTER USER 'slave2'@'%' IDENTIFIED WITH mysql_native_password BY '123123';
```

```
flush privileges;
```

#查询master的状态

```
show master status;
```

```
mysql> show master status;
+-----+-----+-----+-----+-----+
| File           | Position | Binlog_Do_DB | Binlog_Ignore_DB | Executed_Gtid_ |
+-----+-----+-----+-----+-----+
| 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

#建库语句

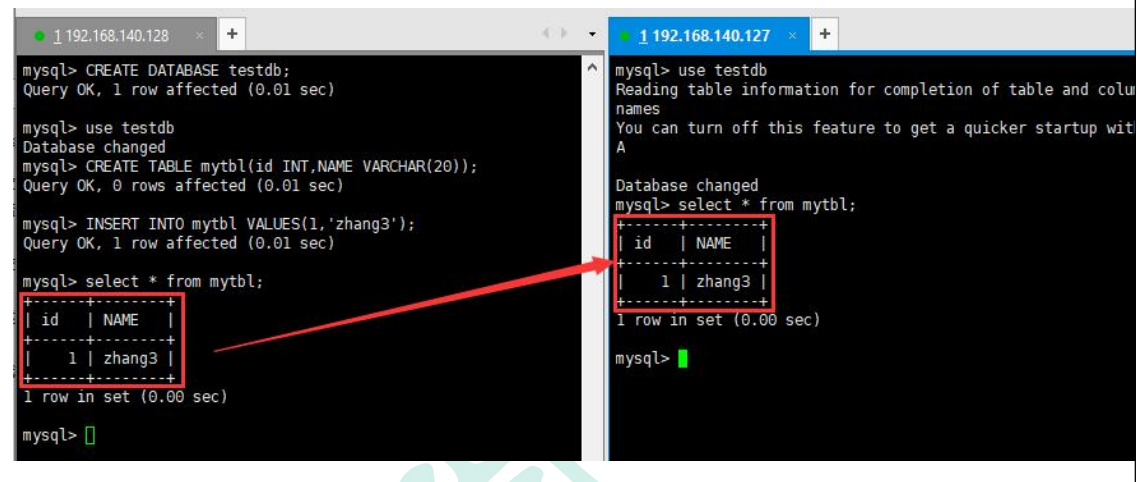
```
CREATE DATABASE mydb1;
```

#建表语句

```
CREATE TABLE mytbl(id INT,NAME VARCHAR(50));
```

#插入数据

```
INSERT INTO mytbl VALUES(1,"zhang3");
```



```
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、配置 Macat 读写分离

登录Mycat，创建逻辑库，配置数据源

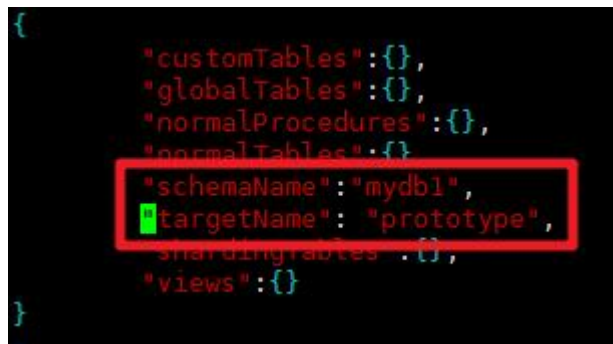
#在Mycat里创建数据库mydb1

#创建db2逻辑库

```
create database mydb1;
```

#修改mydb1.schema.json 指定数据源 "targetName": "prototype", 配置主机数据源

```
vim /usr/local/mycat/conf/schemas/mydb1.schema.json
```



```
{  
  "customTables": {},  
  "globalTables": {},  
  "normalProcedures": {},  
  "normalTables": {},  
  "schemaName": "mydb1",  
  "targetName": "prototype",  
  "standinTables": {},  
  "views": {}  
}
```

使用注解方式添加数据源

#登录Mycat, 注解方式添加数据源, 指向从机

```
/*+ mycat:createDataSource{ "name":"rwSepw",  
"url":"jdbc:mysql://192.168.140.100:3306/mydb1?useSSL=false&characterEncoding=UTF-8&useJDBCCompliantTimezoneShift=true", "user":"root",  
"password":"123123" } */;
```

```
/*+ mycat:createDataSource{ "name":"rwSepr",  
"url":"jdbc:mysql://192.168.140.99:3306/mydb1?useSSL=false&characterEncoding=UTF-8&useJDBCCompliantTimezoneShift=true", "user":"root",  
"password":"123123" } */;
```

#查询配置数据源结果

```
/*+ mycat:showDataSources{} */;
```

更新集群信息, 添加dr0从节点. 实现读写分离

#更新集群信息, 添加dr0从节点.

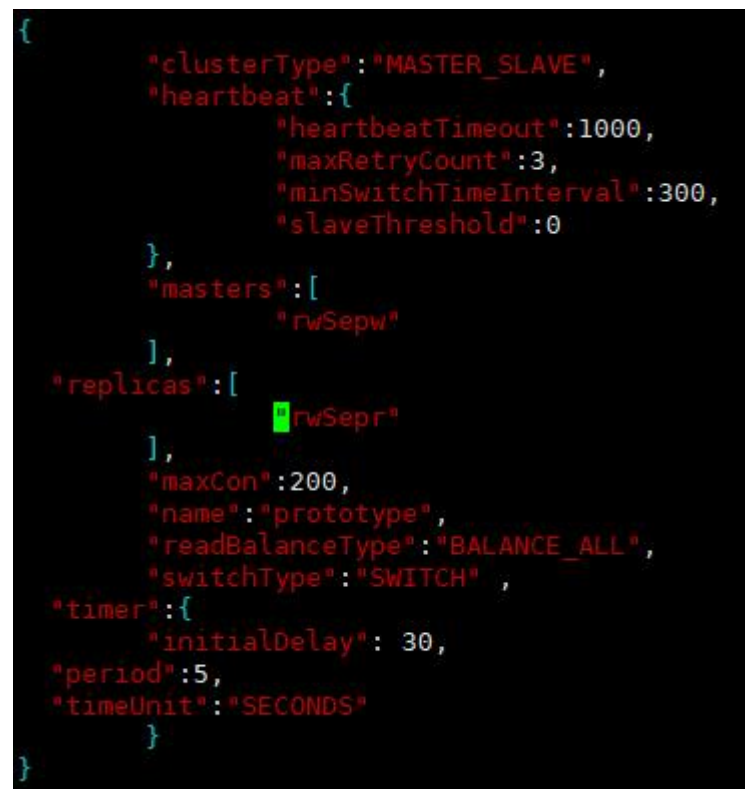
```
/*!  
mycat:createCluster{"name":"prototype","masters":["rwSepw"],"replicas":["rw  
Sepr"]} */;
```

#查看配置集群信息

```
/*+ mycat:showClusters {} */;
```

#查看集群配置文件

```
vim /usr/local/mycat/conf/clusters/prototype.cluster.json
```



```
{  
  "clusterType": "MASTER_SLAVE",  
  "heartbeat": {  
    "heartbeatTimeout": 1000,  
    "maxRetryCount": 3,  
    "minSwitchTimeInterval": 300,  
    "slaveThreshold": 0  
  },  
  "masters": [  
    "rwSepw"  
  ],  
  "replicas": [  
    "rwSepr"  
  ],  
  "maxCon": 200,  
  "name": "prototype",  
  "readBalanceType": "BALANCE_ALL",  
  "switchType": "SWITCH",  
  "timer": {  
    "initialDelay": 30,  
    "period": 5,  
    "timeUnit": "SECONDS"  
  }  
}
```

readBalanceType

查询负载均衡策略

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

switchType

NOT_SWITCH: 不进行主从切换

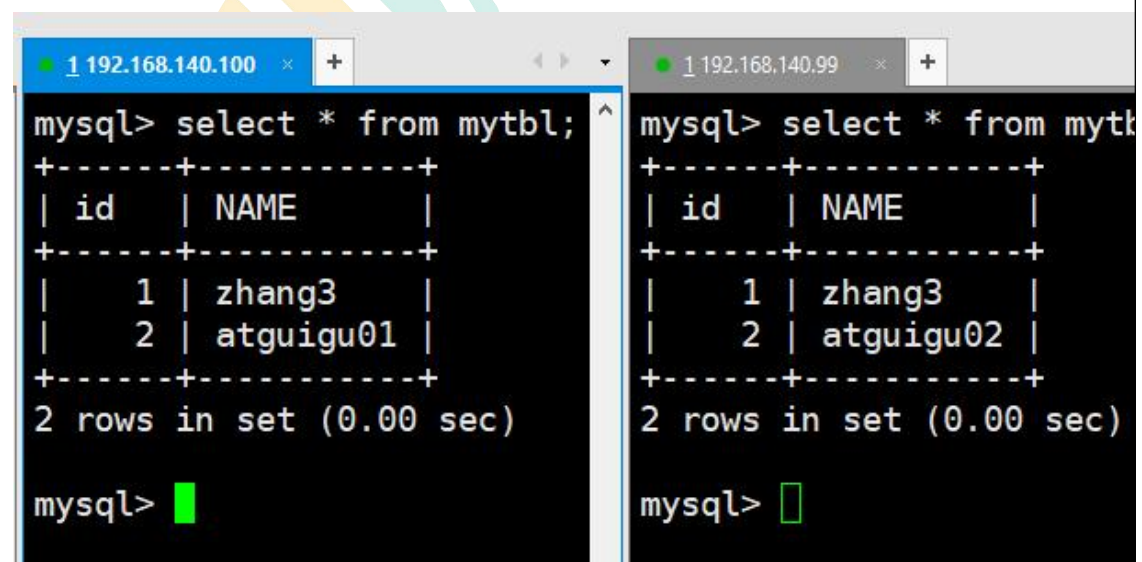
SWITCH: 进行主从切换

3、重新启动 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);
```



```
mysql> select * from mytbl;
+----+-----+
| id  | NAME    |
+----+-----+
| 1   | zhang3  |
| 2   | atguigu01 |
+----+-----+
2 rows in set (0.00 sec)

mysql>

mysql> select * from mytbl;
+----+-----+
| id  | NAME    |
+----+-----+
| 1   | zhang3  |
| 2   | atguigu02 |
+----+-----+
2 rows in set (0.00 sec)

mysql>
```

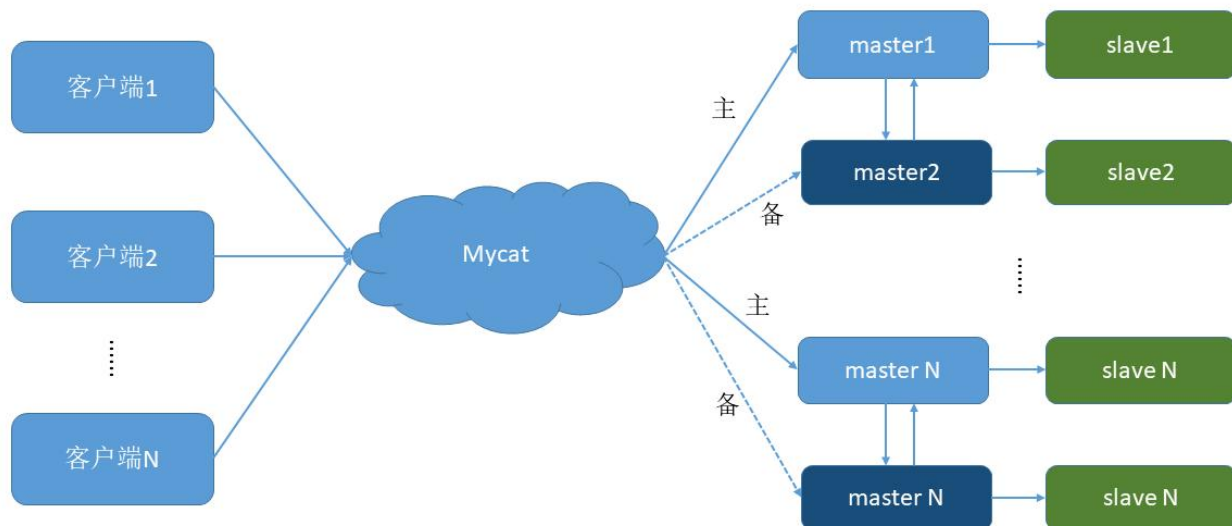
(2) 在Mycat里查询mytbl表, 可以看到查询语句在主从两个主机间切换

id	NAME
1	zhang3
2	atguigu01
*	NULL NULL

id	NAME
1	zhang3
2	atguigu02
*	NULL NULL

4.2 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	192.168.140.100	atguigu01
2	Slave1	192.168.140.99	atguigu02
3	Master2	192.168.140.98	atguigu03

4	Slave2	192.168.140.97	atguigu04
---	--------	----------------	-----------

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

```
CREATE USER 'slave2'@'%' IDENTIFIED BY '123123';
```

```
GRANT REPLICATION SLAVE ON *.* TO 'slave2'@'%' ;
```

#此语句必须执行。否则见下面。

```
ALTER USER 'slave2'@'%' IDENTIFIED WITH mysql_native_password 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_ |
+-----+-----+-----+-----+-----+
| 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_ |
+-----+-----+-----+-----+-----+
| 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;
```

#Slava1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:
```

#Slava2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

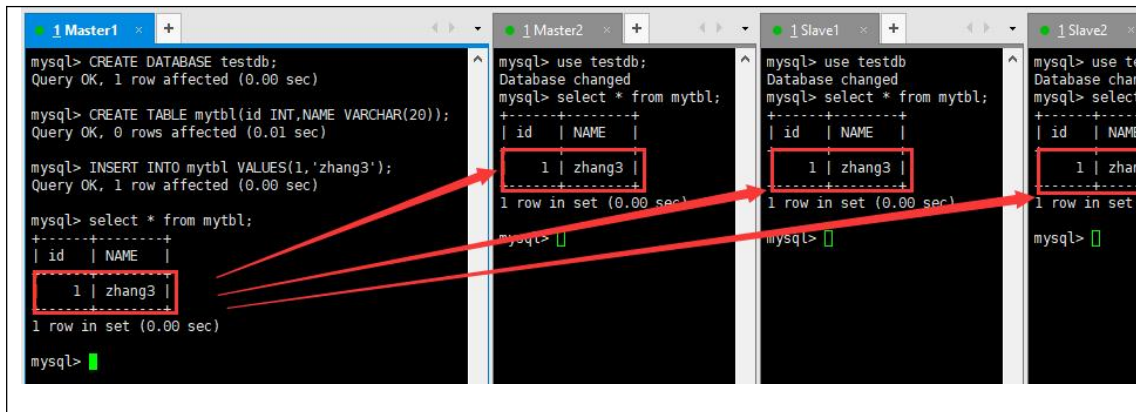
```
mysql> show slave status\G;
***** 1. row *****
      Slave_IO_State: Waiting for master to send event
      Master_Host: 192.168.140.126
      Master_User: slave
      Master_Port: 3306
      Connect_Retry: 60
      Master_Log_File: mysql-bin.000001
      Read_Master_Log_Pos: 154
      Relay_Log_File: 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



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 的集群配置实现多种主从

Mycat2.0的特点把集群概念凸显了出来，和mysql主从复制、集群配合实现多节点读写分离

(1) 双主双从集群角色划分

*m1: 主机

*m2: 备机，也负责读

*s1, s2: 从机

(3) 增加两个数据源

```
/*+ mycat:createDataSource{ "name":"rwSepw2",
"url":"jdbc:mysql://192.168.140.98:3306/mydb1?useSSL=false&characterEncoding=UTF-8&useJDBCCompliantTimezoneShift=true", "user":"root",
"password":"123123" } */;

/*+ mycat:createDataSource{ "name":"rwSepr2",
```



```
"url":"jdbc:mysql://192.168.140.97:3306/mydb1?useSSL=false&characterEncoding=UTF-8&useJDBCCompliantTimezoneShift=true", "user":"root",  
"password":"123123" } */;
```

(2) 修改集群配置文件

```
vim /usr/local/mycat/conf/clusters/prototype.cluster.json
```

```
{  
    "clusterType":"MASTER_SLAVE",  
    "heartbeat":{  
        "heartbeatTimeout":1000,  
        "maxRetryCount":3,  
        "minSwitchTimeInterval":300,  
        "slaveThreshold":0  
    },  
    "masters":[  
        "rwSepw", "rwSepw2"  
    ],  
    "replicas":[  
        "rwSepw2", "rwSepr", "rwSepr2"  
    ],  
    "maxCon":200,  
    "name":"prototype",  
    "readBalanceType":"BALANCE_ALL",  
    "switchType":"SWITCH" ,  
    "timer":{  
        "initialDelay": 30,  
        "period":5,  
    },  
}
```

```
"timeUnit":"SECONDS"

    }

}
```

(3) 重启Mycat生效

3、读写分离配置扩展

通过对集群配置的修改，可以根据需求实现更多种情况的读写分离配置，总结如下

(1) 读写分离(一主一从, 无备) (m是主, s是从)

```
{

    "clusterType":"MASTER_SLAVE",

    "heartbeat":{

        "heartbeatTimeout":1000,

        "maxRetryCount":3,

        "minSwitchTimeInterval":300,

        "slaveThreshold":0

    },

    "masters":[

        "m"

    ],

    "replicas":[

        "s"

    ],

    "maxCon":200,

    "name":"prototype",
```

```
    "readBalanceType": "BALANCE_ALL",  
    "switchType": "SWITCH" ,  
    "timer": {  
        "initialDelay": 30,  
        "period": 5,  
        "timeUnit": "SECONDS"  
    }  
}
```

(2) 读写分离(一主一从,一备)(m是主,s是从备)

```
{  
    "clusterType": "MASTER_SLAVE",  
    "heartbeat": {  
        "heartbeatTimeout": 1000,  
        "maxRetryCount": 3,  
        "minSwitchTimeInterval": 300,  
        "slaveThreshold": 0  
    },  
    "masters": [  
        "m", "s"  
    ],  
    "replicas": [  
        "s"  
    ],  
    "maxCon": 200,  
    "name": "prototype",  
    "readBalanceType": "BALANCE_ALL",  
    "switchType": "SWITCH" ,  
}
```



```
"timer": {  
    "initialDelay": 30,  
    "period": 5,  
    "timeUnit": "SECONDS"  
}  
}
```

(3) 读写分离(一主一从,一备)(m是主,s是从,b是备)

```
{  
    "clusterType": "MASTER_SLAVE",  
    "heartbeat": {  
        "heartbeatTimeout": 1000,  
        "maxRetryCount": 3,  
        "minSwitchTimeInterval": 300,  
        "slaveThreshold": 0  
    },  
    "masters": [  
        "m", "b"  
    ],  
    "replicas": [  
        "s"  
    ],  
    "maxCon": 200,  
    "name": "prototype",  
    "readBalanceType": "BALANCE_ALL",  
    "switchType": "SWITCH",  
    "timer": {  
        "initialDelay": 30,
```

```
"period":5,  
  "timeUnit":"SECONDS"  
}  
}
```

(4) MHA (一主一从, 一备) (m是主, s是从, b是备, READ_ONLY判断主)

```
{  
  "clusterType":"MHA",  
  "heartbeat":{  
    "heartbeatTimeout":1000,  
    "maxRetryCount":3,  
    "minSwitchTimeInterval":300,  
    "slaveThreshold":0  
  },  
  "masters":[  
    "m", "b"  
  ],  
  "replicas":[  
    "s"  
  ],  
  "maxCon":200,  
  "name":"prototype",  
  "readBalanceType":"BALANCE_ALL",  
  "switchType":"SWITCH" ,  
  "timer":{  
    "initialDelay": 30,  
    "period":5,  
    "timeUnit":"SECONDS"  
  }  
}
```

```
}

(5) MGR (一主一从, 一备) (m是主, s是从, b是备, READ_ONLY判断主)

{

    "clusterType": "MGR",

    "heartbeat": {

        "heartbeatTimeout": 1000,

        "maxRetryCount": 3,

        "minSwitchTimeInterval": 300,

        "slaveThreshold": 0

    },

    "masters": [

        "m", "b"

    ],

    "replicas": [

        "s"

    ],

    "maxCon": 200,

    "name": "prototype",

    "readBalanceType": "BALANCE_ALL",

    "switchType": "SWITCH" ,

    "timer": {

        "initialDelay": 30,

        "period": 5,

        "timeUnit": "SECONDS"

    }

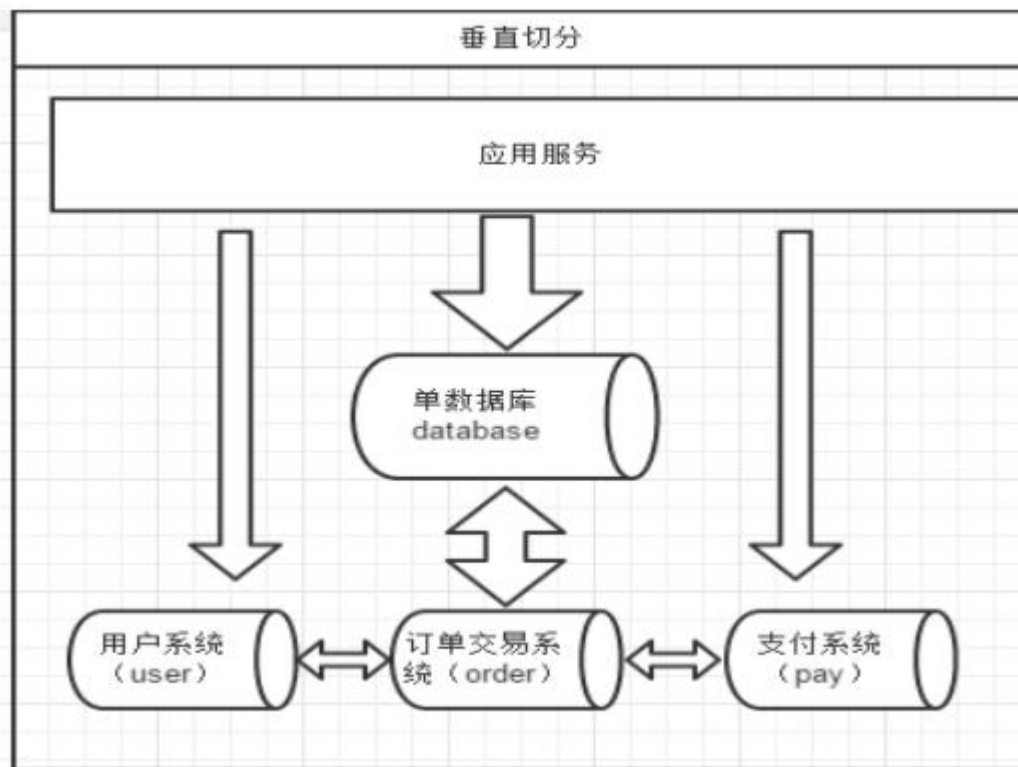
}
```

(6) GARELA_CLUSTER (一主一从, 一备) (m是主, s是从, b多主)

```
{  
  "clusterType": "GARELA_CLUSTER",  
  "heartbeat": {  
    "heartbeatTimeout": 1000,  
    "maxRetryCount": 3,  
    "minSwitchTimeInterval": 300,  
    "slaveThreshold": 0  
  },  
  "masters": [  
    "m", "b"  
  ],  
  "replicas": [  
    "s"  
  ],  
  "maxCon": 200,  
  "name": "prototype",  
  "readBalanceType": "BALANCE_ALL",  
  "switchType": "SWITCH",  
  "timer": {  
    "initialDelay": 30,  
    "period": 5,  
    "timeUnit": "SECONDS"  
  }  
}
```

第五章 分库分表

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。

5.1 如何分库

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。

5.2 如何分表

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

5.3 实现分库分表

Mycat2 一大优势就是可以在终端直接创建数据源、集群、库表，并在创建时指定分库、分表。与 1.6 版本比大大简化了分库分表的操作

1、添加数据库、存储数据源

```
/*+ mycat:createDataSource{  
"name": "dw0",
```

```
"url":"jdbc:mysql://192.168.140.100:3306",
"user":"root",
"password":"123123"
} */;

/*+ mycat:createDataSource{
"name":"dr0",
"url":"jdbc:mysql://192.168.140.100:3306",
"user":"root",
"password":"123123"
} */;

/*+ mycat:createDataSource{
"name":"dw1",
"url":"jdbc:mysql://192.168.140.99:3306",
"user":"root",
"password":"123123"
} */;

/*+ mycat:createDataSource{
"name":"dr1",
"url":"jdbc:mysql://192.168.140.99:3306",
"user":"root",
"password":"123123"
} */;

#通过注释命名添加数据源后，在对应目录会生成相关配置文件

cd /usr/local/mycat/conf/datasources

#如下图
```



```
[root@atguigu01 datasources]# ll
总用量 20
-rw-r--r--. 1 root root 430 2月 15 10:49 dr0.datasource.json
-rw-r--r--. 1 root root 429 2月 15 10:49 dr1.datasource.json
-rw-r--r--. 1 root root 430 2月 15 10:49 dw0.datasource.json
-rw-r--r--. 1 root root 429 2月 15 10:49 dw1.datasource.json
-rw-r--r--. 1 root root 412 2月 14 17:07 prototype.datasource.json
```

2、添加集群配置

把新添加的数据源配置成集群

```
#//在 mycat 终端输入

/*!
mycat:createCluster{"name":"c0","masters":["dw0"],"replicas":["dr0"]}
*/;

/*!
mycat:createCluster{"name":"c1","masters":["dw1"],"replicas":["dr1"]}
*/;

#可以查看集群配置信息

cd /usr/local/mycat/conf/clusters

#如下图
```

```
[root@atguigu01 clusters]# ll
总用量 16
-rw-r--r--. 1 root root 312 2月 15 10:59 c0.cluster.json
-rw-r--r--. 1 root root 312 2月 15 10:59 c1.cluster.json
```

3、创建全局表

#添加数据库db1

```
CREATE DATABASE db1;
```

#在建表语句中加上关键字 BROADCAST（广播，即为全局表）

```
CREATE TABLE db1.`travelrecord` (
```

```

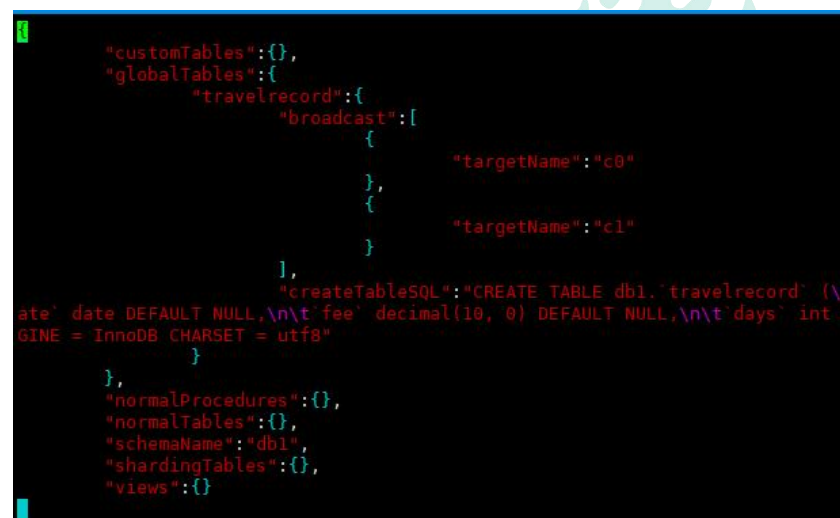
`id` bigint NOT NULL AUTO_INCREMENT,
`user_id` varchar(100) DEFAULT NULL,
`traveldate` date DEFAULT NULL,
`fee` decimal(10,0) DEFAULT NULL,
`days` int DEFAULT NULL,
`blob` longblob,
PRIMARY KEY (`id`),
KEY `id` (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 BROADCAST;

```

#进入相关目录查看 schema 配置

```
vim /usr/local/mycat/conf/schemas/db1.schema.json
```

#可以看到自动生成的全局表配置信息



```

{
  "customTables": {},
  "globalTables": {
    "travelrecord": {
      "broadcast": [
        {
          "targetName": "c0"
        },
        {
          "targetName": "c1"
        }
      ],
      "createTableSQL": "CREATE TABLE db1.`travelrecord` (\n\t`ate` date DEFAULT NULL,\n\t`fee` decimal(10, 0) DEFAULT NULL,\n\t`days` int\n\tENGINE = InnoDB CHARSET = utf8"
    },
    "normalProcedures": {},
    "normalTables": {},
    "schemaName": "db1",
    "shardingTables": {},
    "views": {}
  }
}

```

4、创建分片表(分库分表)

#在 Mycat 终端直接运行建表语句进行数据分片

```

CREATE TABLE db1.orders(
    id BIGINT NOT NULL AUTO_INCREMENT,
    order_type INT,

```

```
customer_id INT,  
amount DECIMAL(10,2),  
PRIMARY KEY(id),  
KEY `id` (`id`)  
)ENGINE=INNODB DEFAULT CHARSET=utf8  
  
dbpartition BY mod_hash(customer_id) tbpartition BY mod_hash(customer_id)  
tbpertitions 1 dbpartitions 2;
```

#数据库分片规则，表分片规则，以及各分多少片

```
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);  
  
SELECT * FROM orders;
```

#同样可以查看生成的配置信息

#进入相关目录查看 schema 配置

```
vim /usr/local/mycat/conf/schemas/db1.schema.json
```

```

    "shardingTables":{
      "orders":{
        "createTableSQL":"CREATE TABLE db1.orders (\n\tid BIGINT NOT NULL AUTO_INCREMENT\n\t2)\n\tPRIMARY KEY (id),\n\tKEY `id` (`id`)\n\tENGINE = INNODB CHARSET = utf8\n\tPARTITION BY mod_hash(id)\n\tfunction":{
          "properties":{
            "dbNum":"1",
            "mappingFormat":"c${targetIndex}/db1_${dbIndex}/orders_${index}"
            "tableNum":"2",
            "tableMethod":"mod_hash(id)",
            "storeNum":2,
            "dbMethod":"mod_hash(id)"
          }
        }
      },
      "shardingIndexTables":{}
    },
    "views":{}
  },
  "views":{}
}

```

查看数据库可见，分片数据

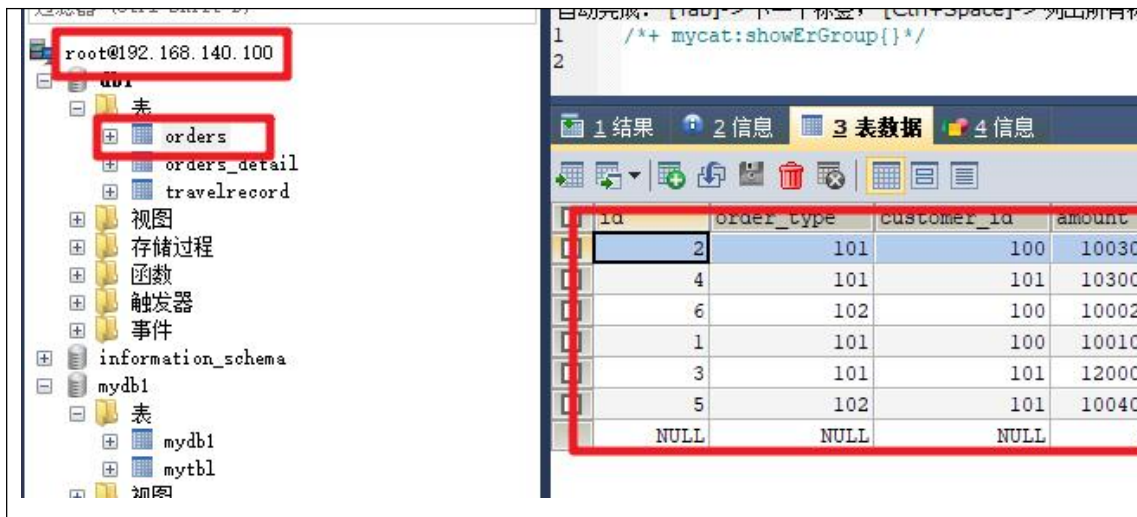


The screenshot displays two MySQL database instances connected to a Mycat proxy. The top instance (root@192.168.140.100) shows the 'db1_0' database with a table 'orders_0'. The bottom instance (root@192.168.140.99) shows the 'db1_1' database with a table 'orders_1'. Both tables contain data from the 'orders' table, demonstrating data sharding.

id	order_type	customer_id	amount
2	101	100	100300
4	101	101	103000
6	102	100	100020

id	order_type	customer_id	amount
1	101	100	100100.00
3	101	101	120000.00
5	102	101	100400.00

在 Mycat 终端查询依然可以看到全部数据



5、创建 ER 表

与分片表关联的表如何分表，也就是 ER 表如何分表，如下

#在 Mycat 终端直接运行建表语句进行数据分片

```
CREATE TABLE orders_detail(
```

```
  `id` BIGINT NOT NULL AUTO_INCREMENT,
```

```
  detail VARCHAR(2000),
```

```
  order_id INT,
```

```
  PRIMARY KEY(id)
```

```
)ENGINE=INNODB DEFAULT CHARSET=utf8
```

```
dbpartition BY mod_hash(order_id) tpartition BY mod_hash(order_id)
tpartitions 1 dbpartitions 2;
```

```
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);
```

*对比数据节点 1

过滤器 (Ctrl+Shift+B)

root@192.168.140.100

db1

db1_0

表

orders_0

orders_detail_0

视图

存储过程

函数

触发器

事件

information_schema

mycat

自动完成: [Tab]-> 下一个标签, [Ctrl+Space]-> 列出所有标签, [C

1

1 信息 2 表数据 3 信息

id	order_type	customer_id	amount
1	101	100	100100.00
2	101	100	100300.00
6	102	100	100020.00
(Auto)	(NULL)	(NULL)	(NULL)

1

1 信息 2 表数据 3 信息

id	detail	order_id
2	detail1	2
4	detail1	4
6	detail1	6
(Auto)	(NULL)	(NULL)

*对比数据节点 2

root@192.168.140.99

db1

db1_1

表

orders_1

orders_detail_1

视图

存储过程

函数

触发器

事件

information_schema

1

1 信息 2 表数据 3 信息

id	order_type	customer_id	amount
3	101	101	120000.00
4	101	101	103000.00
5	102	101	100400.00
(Auto)	(NULL)	(NULL)	(NULL)

1

1 信息 2 表数据 3 信息

id	detail	order_id
1	detail1	1
3	detail1	3
5	detail1	5
(Auto)	(NULL)	(NULL)

#上述两表具有相同的分片算法, 但是分片字段不相同

#Mycat2 在涉及这两个表的 join 分片字段等价关系的时候可以完成 join 的下推

#Mycat2 无需指定 ER 表, 是自动识别的, 具体看分片算法的接口

#查看配置的表是否具有 ER 关系, 使用

```
/*+ mycat:showErGroup {} */
```

groupId	schemaName	tableName
0	db1	orders
0	db1	orders_detail

#group_id 表示相同的组, 该组中的表具有相同的存储分布

运行关联查询语句

```
SELECT * FROM orders o INNER JOIN orders_detail od ON od.order_id=o.id;
```

3
4
5

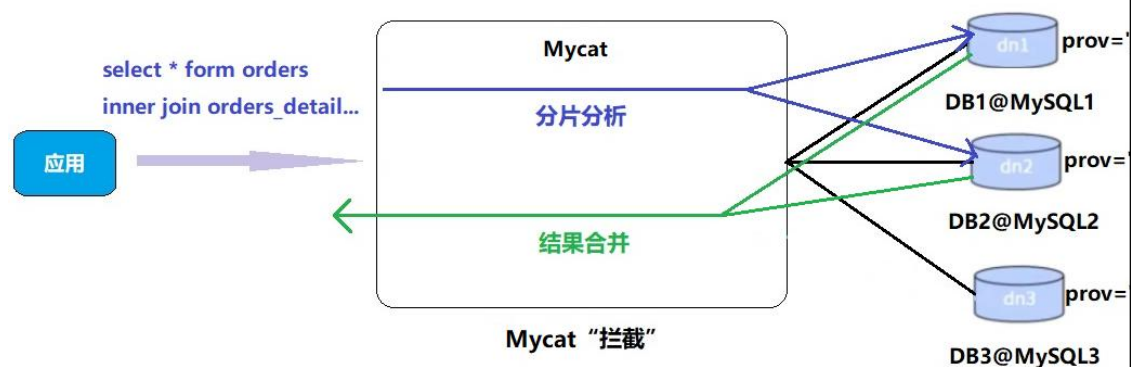
```
SELECT * FROM orders o INNER JOIN orders_detail od ON od.order_id=o.id;
```

1 结果 2 个配置文件 3 信息 4 表数据 5 信息

(只读)

id	order_type	customer_id	amount	id0	detail	order_id
2	101	100	100300.0	2	detail1	2
4	101	101	103000.0	4	detail1	4
6	102	100	100020.0	6	detail1	6
1	101	100	100100.0	1	detail1	1
3	101	101	120000.0	3	detail1	3
5	102	101	100400.0	5	detail1	5

#原理如下



5.4 常用分片规则

1、分片算法简介

Mycat2 支持常用的(自动)HASH 型分片算法也兼容 1.6 的内置的(cobar)分片算法.

HASH 型分片算法默认要求集群名字以 c 为前缀, 数字为后缀, c0 就是分片表第一个节点, c1 就是第二个节点. 该命名规则允许用户手动改变

2、Mycat2 与 1.x 版本区别

Mycat2 Hash 型分片算法多数基于 MOD_HASH (MOD 对应 JAVA 的%运算), 实际上是取余运算。

Mycat2 Hash 型分片算法对于值的处理, 总是把分片值转换到列属性的数据类型再运算。

而 1.x 系列的分片算法统一转换到字符串类型再运算且只能根据一个分片字段计算出存储节点下标。

Mycat2 Hash 型分片算法适用于等价条件查询。

而 1.x 系列由于含有用户经验的路由规则。1.x 系列的分片规则总是先转换成字符串再运算。

3、分片规则与适用性

分片算法	描述	分库	分表	数值类型
MOD_HASH	取模哈希	是	是	数值, 字符串
UNI_HASH	取模哈希	是	是	数值, 字符串
RIGHT_SHIFT	右移哈希	是	是	数值
RANGE_HASH	两字段其一取模	是	是	数值, 字符串
YYYYMM	按年月哈希	是	是	DATE, DATETIME
YYYYDD	按年日哈希	是	是	DATE, DATETIME
YYYYWEEK	按年周哈希	是	是	DATE, DATETIME

HASH	取模哈希	是	是	数值, 字符串, 如果不是, 则转换成字符串
MM	按月哈希	否	是	DATE, DATETIME
DD	按日期哈希	否	是	DATE, DATETIME
MMDD	按月日哈希	是	是	DATE, DATETIME
WEEK	按周哈希	否	是	DATE, DATETIME
STR_HASH	字符串哈希	是	是	字符串

4、常用分片规则简介

(1) MOD_HASH

[数据分片]HASH 型分片算法-MOD_HASH

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

分库键和分表键是同键:

分表下标=分片值%(分库数量*分表数量)

分库下标=分表下标/分表数量

分库键和分表键是不同键:

分表下标= 分片值%分表数量

分库下标= 分片值%分库数量

(2) RIGHT_SHIFT

[数据分片]HASH 型分片算法-RIGHT_SHIFT

RIGHT_SHIFT(字段名, 位移数)

仅支持数值类型

分片值右移二进制位数, 然后按分片数量取余

(3) YYYYMM

[数据分片]HASH 型分片算法-YYYYMM

仅用于分库

$(YYYY * 12 + MM) \% \text{分库数}$. MM 是 1-12

(4) MMDD

仅用于分表

仅 DATE/DATETIME

一年之中第几天%分表数

tbpartitions 不超过 366

5.5 全局序列

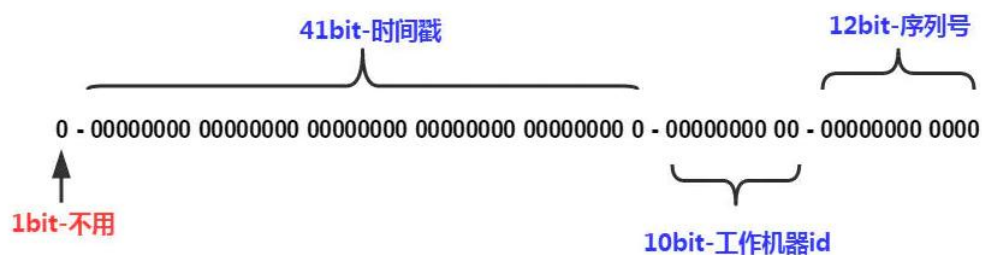
Mycat2 在 1.x 版本上简化全局序列, 自动默认使用雪花算法生成全局序列号, 如不需要 Mycat 默认的全局序列, 可以通过配置关闭自动全局序列

1、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 的全局序列号. 这样, 对应的插入 sql 在 mysql 处理, 由 mysql 的自增主键功能补全自增值.

雪花算法: 引入了时间戳和 ID 保持自增的分布式 ID 生成算法

snowflake-64bit



建表 sql 可以自动在原型库对应的逻辑表的物理表获取, 如果逻辑表的建表 SQL 与物理表的建表 SQL 不对应, 则需要在配置文件中配置建表 SQL.

例如:

#带 AUTO_INCREMENT 关键字使用默认全局序列

```
CREATE TABLE db1.`travelrecord` (  
  `id` bigint NOT NULL AUTO_INCREMENT,  
  `user_id` varchar(100) DEFAULT NULL,  
  `traveldate` date DEFAULT NULL,  
  `fee` decimal(10,0) DEFAULT NULL,  
  `days` int DEFAULT NULL,  
  `blob` longblob,  
  PRIMARY KEY (`id`),  
  KEY `id` (`id`)  
) ENGINE=InnoDB DEFAULT CHARSET=utf8 BROADCAST;
```

#去掉关键字, 不使用

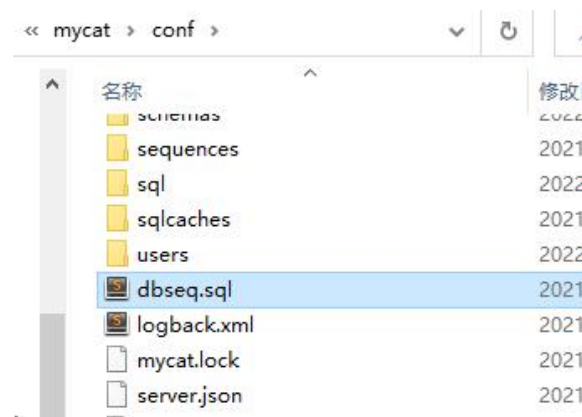
```
CREATE TABLE db1.`travelrecord` (  
  `id` bigint NOT NULL,  
  `user_id` varchar(100) DEFAULT NULL,  
  `traveldate` date DEFAULT NULL,  
  `fee` decimal(10,0) DEFAULT NULL,  
  `days` int DEFAULT NULL,  
  `blob` longblob,  
  PRIMARY KEY (`id`),  
  KEY `id` (`id`)  
) ENGINE=InnoDB DEFAULT CHARSET=utf8 BROADCAST;
```

2、设置 Mycat 数据库方式获取全局序列

#1、在prototype服务器的db1库导入dbseq.sql文件

Mycat2已经为用户提供了相关sql脚本，需要在对应数据库下运行脚本，不能通过Mycat客户端执行。

脚本所在目录mycat/conf



脚本内容：

```
DROP TABLE IF EXISTS MYCAT_SEQUENCE;

CREATE TABLE MYCAT_SEQUENCE ( name VARCHAR(64) NOT NULL, current_value
BIGINT(20) NOT NULL, increment INT NOT NULL DEFAULT 1, PRIMARY KEY
(name) ) ENGINE=InnoDB;

-----

-- Function structure for `mycat_seq_currval`
-----

DROP FUNCTION IF EXISTS `mycat_seq_currval`;

DELIMITER ;;

CREATE FUNCTION `mycat_seq_currval`(seq_name VARCHAR(64)) RETURNS
varchar(64) CHARSET latin1

    DETERMINISTIC

BEGIN
```

```
DECLARE retval VARCHAR(64);

SET retval="-1,0";

SELECT concat(CAST(current_value AS CHAR),",",CAST(increment AS CHAR) )
INTO retval FROM MYCAT_SEQUENCE WHERE name = seq_name;

RETURN retval ;

END

;;

DELIMITER ;

-- -----
-- Function structure for `mycat_seq_nextval`
-- -----

DROP FUNCTION IF EXISTS `mycat_seq_nextval`;

DELIMITER ;;

CREATE FUNCTION `mycat_seq_nextval`(seq_name VARCHAR(64)) RETURNS
varchar(64) CHARSET latin1

    DETERMINISTIC
BEGIN

    DECLARE retval VARCHAR(64);

    DECLARE val BIGINT;

    DECLARE inc INT;

    DECLARE seq_lock INT;

    set val = -1;

    set inc = 0;

    SET seq_lock = -1;

    SELECT GET_LOCK(seq_name, 15) into seq_lock;

    if seq_lock = 1 then

        SELECT current_value + increment, increment INTO val, inc FROM
        MYCAT_SEQUENCE WHERE name = seq_name for update;
```

```
        if val != -1 then

            UPDATE MYCAT_SEQUENCE SET current_value = val WHERE name =
seq_name;

            end if;

            SELECT RELEASE_LOCK(seq_name) into seq_lock;

            end if;

            SELECT concat(CAST((val - inc + 1) as CHAR),",",CAST(inc as CHAR)) INTO
retval;

            RETURN retval;
END
;;
DELIMITER ;

-----

-- Function structure for `mycat_seq_setvals`
-----

DROP FUNCTION IF EXISTS `mycat_seq_nextvals`;
DELIMITER ;;

CREATE FUNCTION `mycat_seq_nextvals`(seq_name VARCHAR(64), count INT)
RETURNS VARCHAR(64) CHARSET latin1

    DETERMINISTIC
BEGIN

    DECLARE retval VARCHAR(64);

    DECLARE val BIGINT;

    DECLARE seq_lock INT;

    SET val = -1;

    SET seq_lock = -1;

    SELECT GET_LOCK(seq_name, 15) into seq_lock;

    if seq_lock = 1 then

        SELECT current_value + count INTO val FROM MYCAT_SEQUENCE WHERE
```

```
name = seq_name for update;

    IF val != -1 THEN

        UPDATE MYCAT_SEQUENCE SET current_value = val WHERE name =
seq_name;

    END IF;

    SELECT RELEASE_LOCK(seq_name) into seq_lock;

end if;

    SELECT CONCAT(CAST((val - count + 1) as CHAR), ",", CAST(val as CHAR))
INTO retval;

    RETURN retval;
END
;;
DELIMITER ;

-----

-- Function structure for `mycat_seq_setval`
-----

DROP FUNCTION IF EXISTS `mycat_seq_setval`;

DELIMITER ;;

CREATE FUNCTION `mycat_seq_setval`(seq_name VARCHAR(64), value BIGINT)
RETURNS varchar(64) CHARSET latin1

    DETERMINISTIC

BEGIN

    DECLARE retval VARCHAR(64);

    DECLARE inc INT;

    SET inc = 0;

    SELECT increment INTO inc FROM MYCAT_SEQUENCE WHERE name = seq_name;

    UPDATE MYCAT_SEQUENCE SET current_value = value WHERE name = seq_name;

    SELECT concat(CAST(value as CHAR), ",", CAST(inc as CHAR)) INTO retval;
```

```
RETURN retval;

END

;;

DELIMITER ;

INSERT INTO MYCAT_SEQUENCE VALUES ('GLOBAL', 1, 1);
```

#2、添加全局序列配置文件

进入/mycat/conf/sequences目录，添加配置文件

{数据库名字}_{表名字}.sequence.json

配置内容：

```
{
    "clazz": "io.mycat.plugin.sequence.SequenceMySQLGenerator",
    "name": "db1_travelrecord",
    "targetName": "prototype",
    "schemaName": "db1" //指定物理库名
}
```

可选参数targetName 更改序列号服务器

"targetName": "prototype" 是执行自增序列的节点, 也是dbseq.sql导入的节点

dbseq.sql导入的当前库的库名与逻辑表的逻辑库名一致

导入后检查库下有没有mycat_sequence表。

NAME	current_value	increment
testSchema_sharding	3	1
(NULL)	(NULL)	1

其中increment是序列号自增的步伐, 为1的时候严格按1递增, 当1000的时候, mycat会每次批量递增1000取序列号. 此时在多个mycat访问此序列号表的情况下, 不能严格自增

NAME列中的值是对应的 库名_表名 该值需要用户设置, 即插入一条逻辑表相关的记录, 用于记录序列号

#3、切换为数据库方式全局序列号

使用注释前要导入dbseq.sql以及设置mycat_sequence表内的逻辑表记录

通过注释设置为数据库方式全局序列号

```
/*+ mycat:setSequence{  
  "name":"db1_travelrecord",  
  "clazz":"io.mycat.plugin.sequence.SequenceMySQLGenerator",  
  "name":"db1_travelrecord",  
    "targetName": "prototype",  
    "schemaName":"db2"  
} */;
```

#4、切换为雪花算法方式全局序列号

```
/*+ mycat:setSequence{"name":"db1_travelrecord","time":true} */;
```

第六章 Mycat 安全设置

6.1 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 mycat/conf/users 目录下的 {用户名}.user.json 进行配置。

```
#root.user.json

{
    "dialect": "mysql",
    "ip": null,
    "password": "123456",
    "transactionType": "xa",
    "username": "root"
}
```

#如下图



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
ip	建议为空, 填写后会对客户端的 ip 进行限制
dialect	使用语言, 默认 mysql
transactionType	事务类型, 默认 proxy proxy: 本地事务, 在涉及大于 1 个数据库的事务, commit 阶段失败

	<p>会导致不一致,但是兼容性最好</p> <p>xa: 分布式事务, 需要确认存储节点集群类型是否支持 XA</p> <p>更改命令: <code>set transaction_policy = 'xa'</code></p> <p><code>set transaction_policy = 'proxy'</code></p> <p>查看命令: <code>SELECT @@transaction_policy</code></p>
--	--

2、权限说明

Mycat2 权限分为两块: 登录权限、sql 权限

(1) 登录权限:

Mycat2 在 MySQL 网络协议的时候检查客户端的 IP, 用户名, 密码

其中 IP 使用正则表达式匹配, 一旦匹配成功, 就放行

(2) sql 权限

使用自定义拦截器实现

第七章 Mycat 相关工具

7.1 Mycat2 UI

Mycat2 UI 是官方推出的 Mycat2 监控工具。

1、下载

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

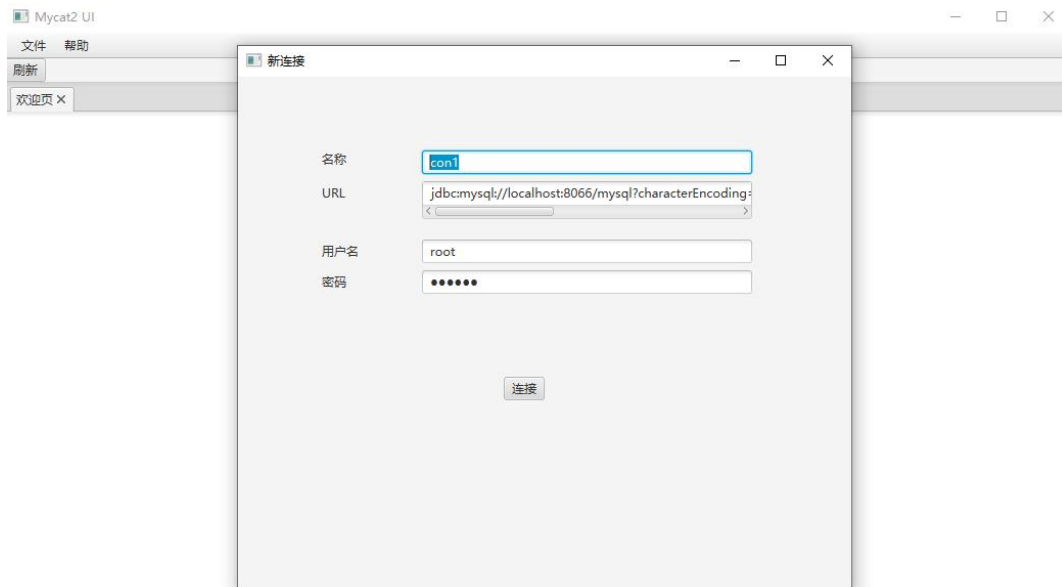
2、运行环境

在安装 JDK8 的环境, 双击 jar 包就可以打开

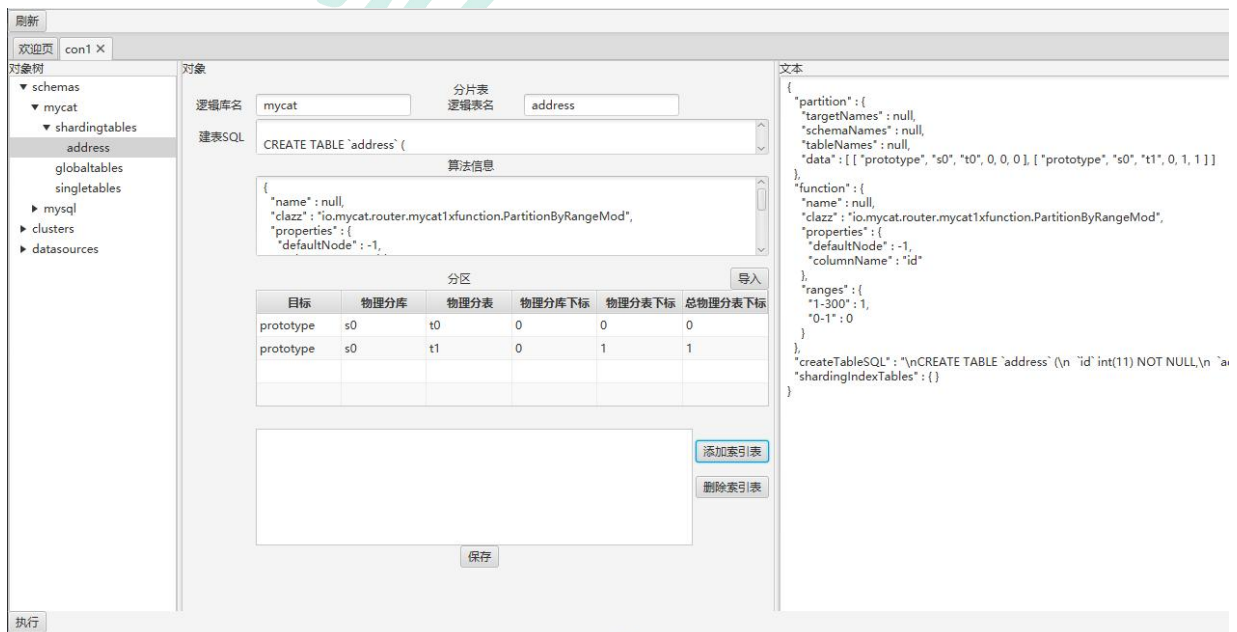
3、使用

(1) 连接

连接 Mycat2 (需要 Mycat2 服务器正常启动)



(2) 编辑分片表



导入文件是 csv 格式, 无表头

(6 项)

prototype, s0, t0, 0, 0, 0

prototype, s0, t1, 0, 1, 1

(3 项)

prototype, s0, t0

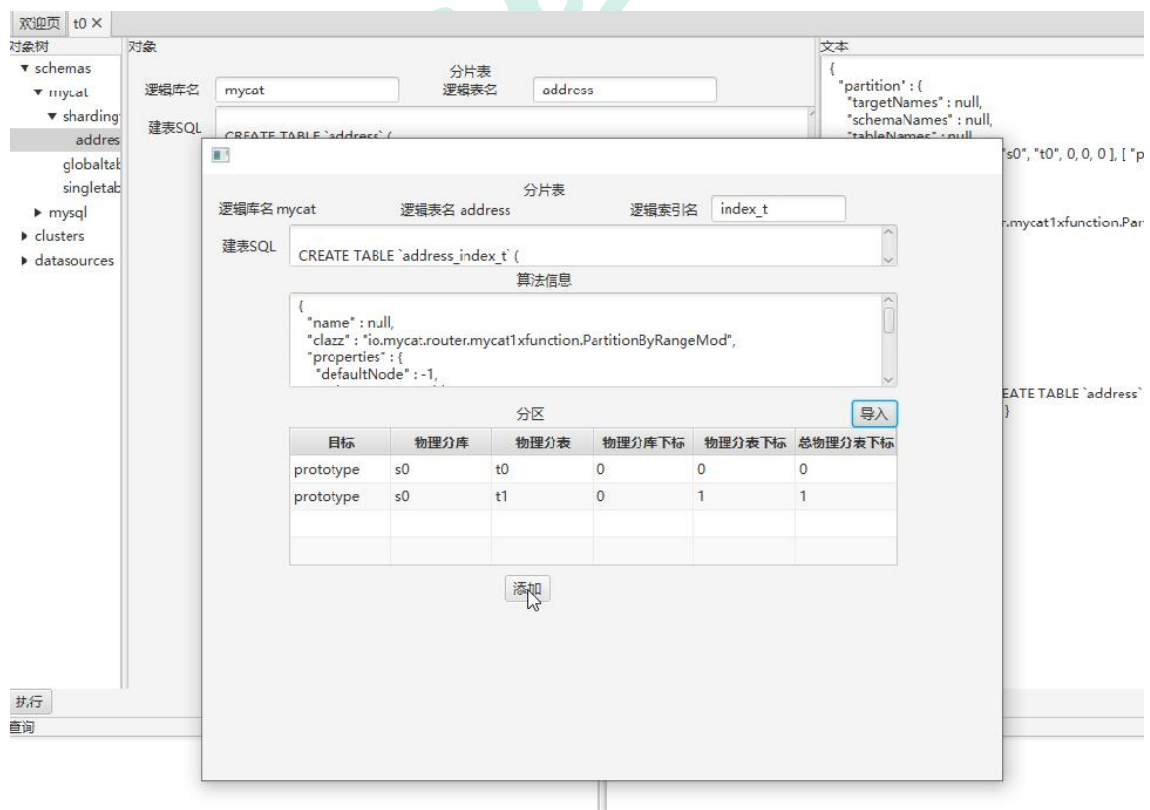
prototype, s0, t1

暂时不支持自动 HASH 型算法的分区导入

物理分库下标, 物理分表下标是根据分片算法要求填入, 没有明确要求不需要填写

对于 1.6 的分片算法, 物理分库下标, 物理分表下标是没有意义的, 只有总物理分表下标有意义 (总分表的下标)

(3) 编辑索引表



(4) 编辑全局表

