https://mp.weixin.qq.com/s/p5ZARUF5juxxlnJhDW9n6w

# MySQL读写分离+故障转移

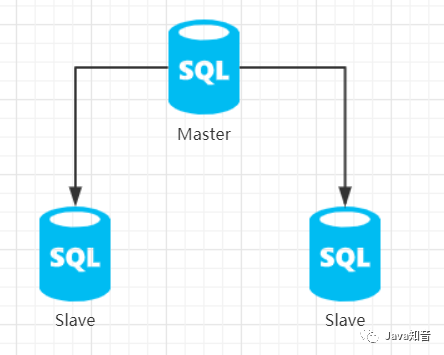
前言

上一篇发了[手动搭建Redis集群和MySQL主从同步(非Docker)](http://mp.weixin.qq.com/s?__biz=MzI4Njc5NjM1NQ==&mid=2247490339&idx=1&sn=355246186ce9cad69a8a4fe578141ffb&chksm=ebd6240fdca1ad19a363b3c0a917352f78ca4cdb20c79f639298ef731295bde3753fd55cf539&scene=21#wechat_redirect)之后，很多同学对文中主从结构提到的读写分离感兴趣，本打算在双十一期间直接把读写分离分享给大家，奈何工作一直没停下，所以这周抽空把这些分享出来。

关于MySQL的读写分离的实现，有两种方式，第一种方式即我们**手动在代码层实现逻辑，来解析读请求或者写请求，分别分发到不同的数据库中，实现读写分离**；第二种方式就是**基于MyCat中间件来实现读写分离的效果**；这两种方式我都会在这篇博客中进行详细地介绍、搭建，并且分析其中的优劣。

原理初探

从MySQL的主从同步开始谈起，最开始我们的数据库架构是这样的。



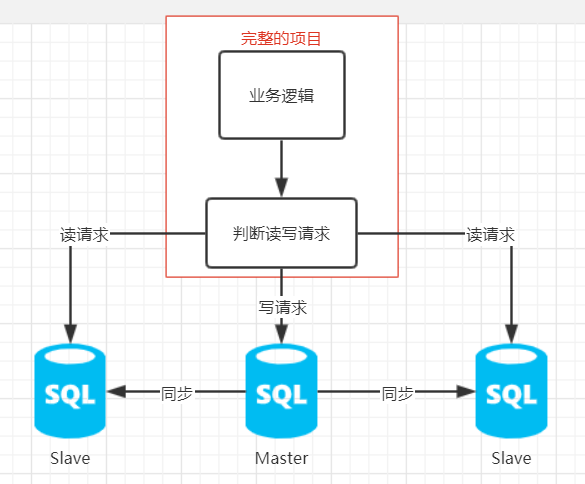
主库负责了所有的读写操作，而从库只对主库进行了备份，就像我在上一篇文章中说的那样，我认为如果只实现了一个备份，不能读写分离和故障转移，不能降低Master节点的IO压力，这样的主从架构看起来性价比似乎不是很高。

我们所希望的主从架构是，当我们在写数据时，请求全部发到Master节点上，当我们需要读数据时，请求全部发到Slave节点上。并且多个Slave节点最好可以存在负载均衡，让集群的效率最大化。

那么这样的架构就不够我们使用了，我们需要找寻某种方式，来实现读写分离。那么实际上有两种方式。

### 方法1：代码层实现读写分离

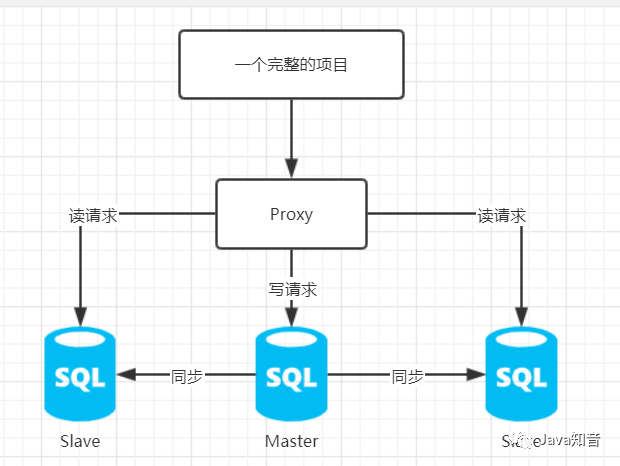
这种方法的优势就是比较灵活，我们可以按照自己的逻辑来决定读写分离的规则。如果使用了这样的方法，我们整个数据库的架构就可以用下面这张图进行概括：



### 方法2：使用中间层（虚拟节点）进行请求的转发

这种方式最主要的特点就是我们在除了数据库以外地方，新构建了一个虚拟节点，而我们所有的请求都发到这个虚拟节点上，由这个虚拟节点来转发读写请求该相应的数据库。

这种方式的特点就是，其构建了一个独立的节点来接收所有的请求，而不用在我们的程序中配置多数据源，我们的项目只需要将url指向这个虚拟节点，然后由这个虚拟节点来处理读写请求。不是有这么一句话吗，**专业的事交给专业的人来做**，大概是这么个意思吧。而现在存在的MyCat等中间件，就是这样的一个”专业的人“。



那么下面我就会动手实现上述两个读写分离的解决方案，代码层实现读写分离和使用中间件实现读写分离。

## 手动实现读写分离

实现读写分离的方法有很多，我这里会说到两种，第一种是使用MyBatis和Spring，手写MyBatis拦截器来判断SQL是读或者写，从而选择数据源，最后交给Spring注入数据源，来实现读写分离；第二种是使用MyCat中间件，配置化地实现读写分离，每种方式都有其可取之处，可以自己视情况选用。

### 环境说明

这里用到了我的上篇博客[手动搭建Redis集群和MySQL主从同步(非Docker)](http://mp.weixin.qq.com/s?__biz=MzI4Njc5NjM1NQ==&mid=2247490339&idx=1&sn=355246186ce9cad69a8a4fe578141ffb&chksm=ebd6240fdca1ad19a363b3c0a917352f78ca4cdb20c79f639298ef731295bde3753fd55cf539&scene=21" \l "wechat_redirect" \t "_blank)中所搭建的MySQL主从同步，如果手上没有这套环境的，可以先比着这篇博客进行搭建。但是需要注意的是，要将8.0版本的MySQL改为5.7。

192.168.43.201:3306 Master  
192.168.43.202:3306 Slave

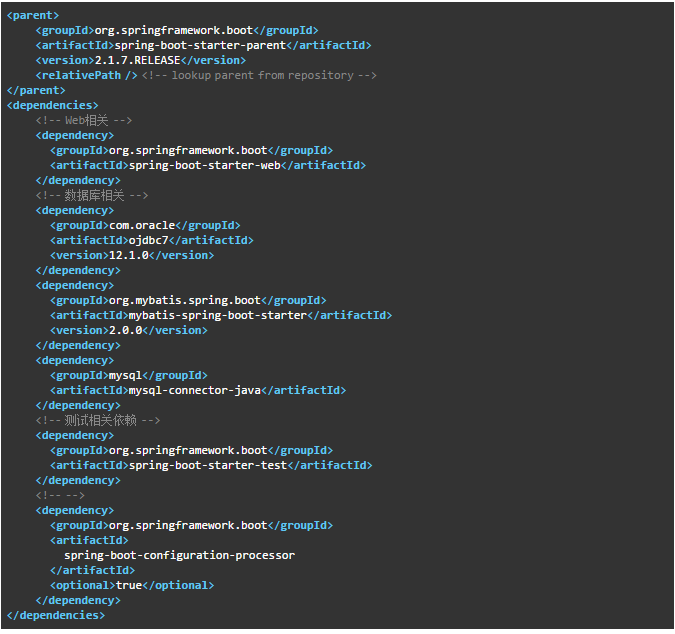
开发环境：

IDE:Eclipse  
Spring boot 2.1.7  
MySQL 5.7  
CentOS 7.3

### 新建Maven项目

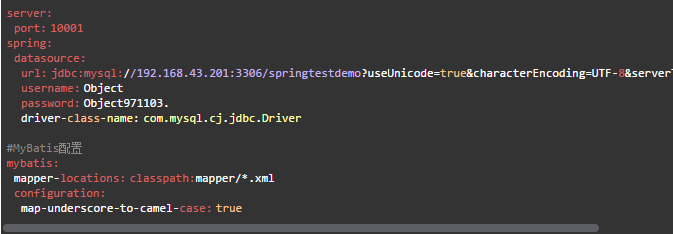
为了演示方便，这里使用SpringBoot作为测试的基础框架，省去了很多Spring需要的xml配置。没有用过SpringBoot的同学也没关系，我会一步一步地进行演示操作。

### 导入依赖

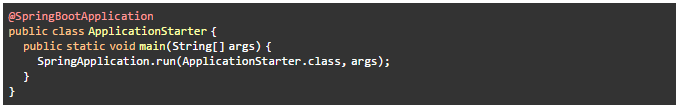


### application.yml

为了测试项目尽量简单，所以我们不用去过多地配置其它东西。只有一些基本配置和数据源配置。



### 编写启动类



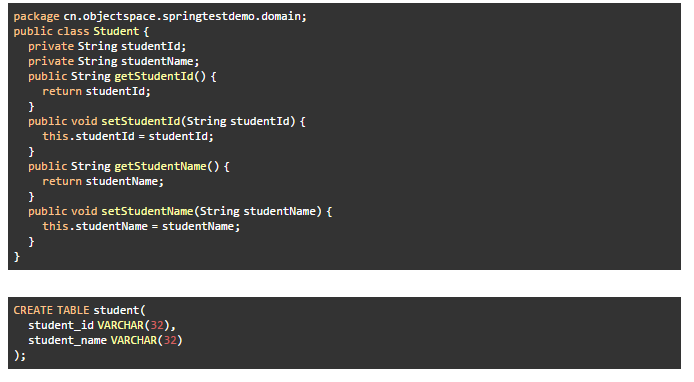
### 启动



出现以上信息代表启动成功。嗯……这应该是一个数据库相关的博客，好像讲了太多的SpringBoot

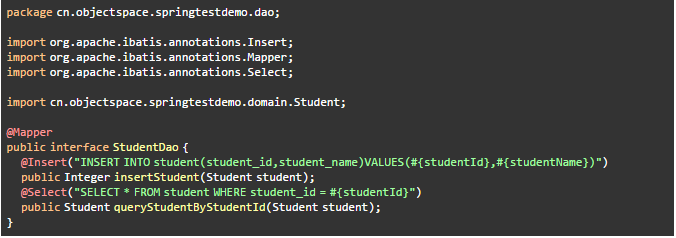
到这里说明我们的SpringBoot项目没有问题，已经搭建成功，如果还不放心，可以自行访问一下http://localhost:10001这个路径，如果出现SpringBoot的404，则代表启动成功。

### 新建Student实体并创建数据库

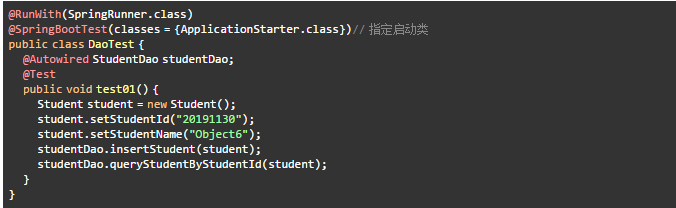


### 编写StudentDao接口，并进行测试

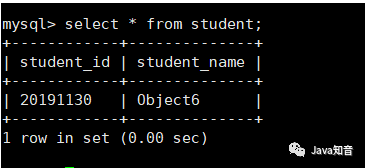
接口：



测试类：



如果可以正确往数据库中插入数据，如下图，则MyBatis搭建成功。

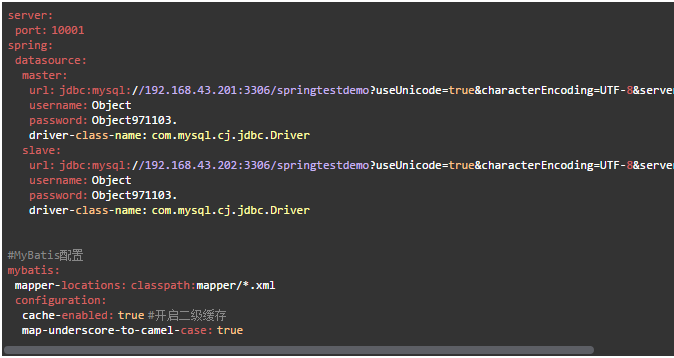


### 正式搭建

通过上面的准备工作，我们已经可以实现对数据库的读写，但是并没有实现读写分离，现在才是开始实现数据库的读写分离。

### 修改application.yml

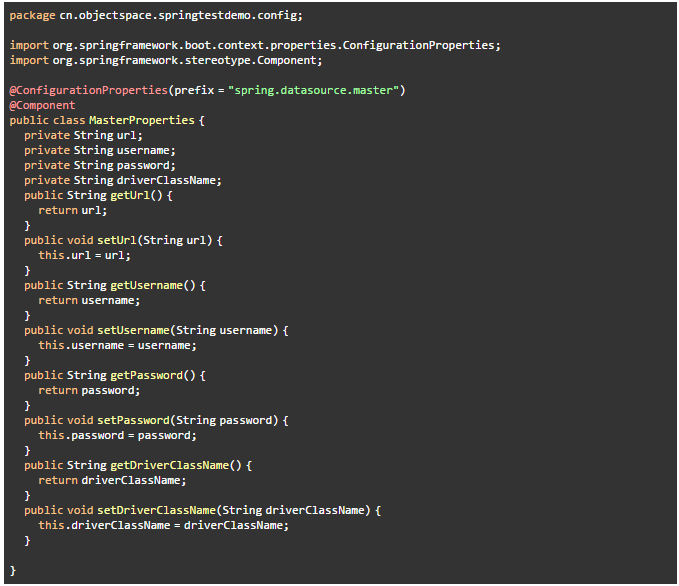
刚才我们的配置文件中只有单数据源，而读写分离肯定不会是单数据源，所以我们首先要在application.yml中配置多数据源。



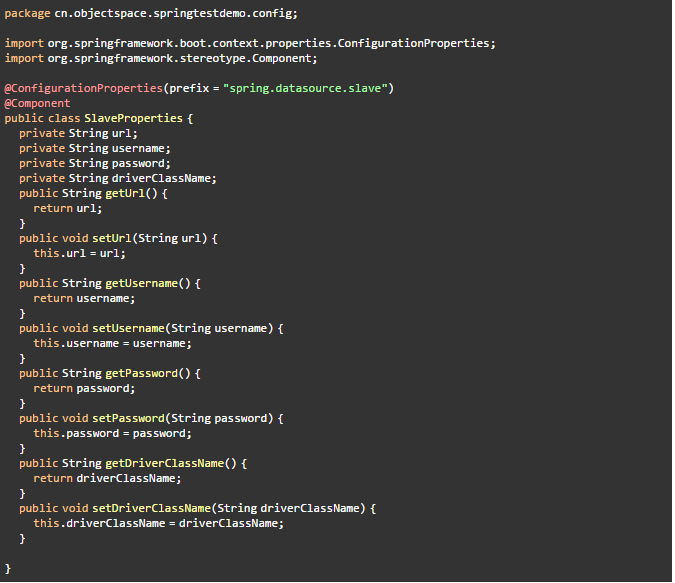
### DataSource的配置

首先要先创建两个ConfigurationProperties类，这一步不是非必须的，直接配置DataSource也是可以的，但是我还是比较习惯去写这个Properties。

#### MasterProperpties



#### SlaveProperties



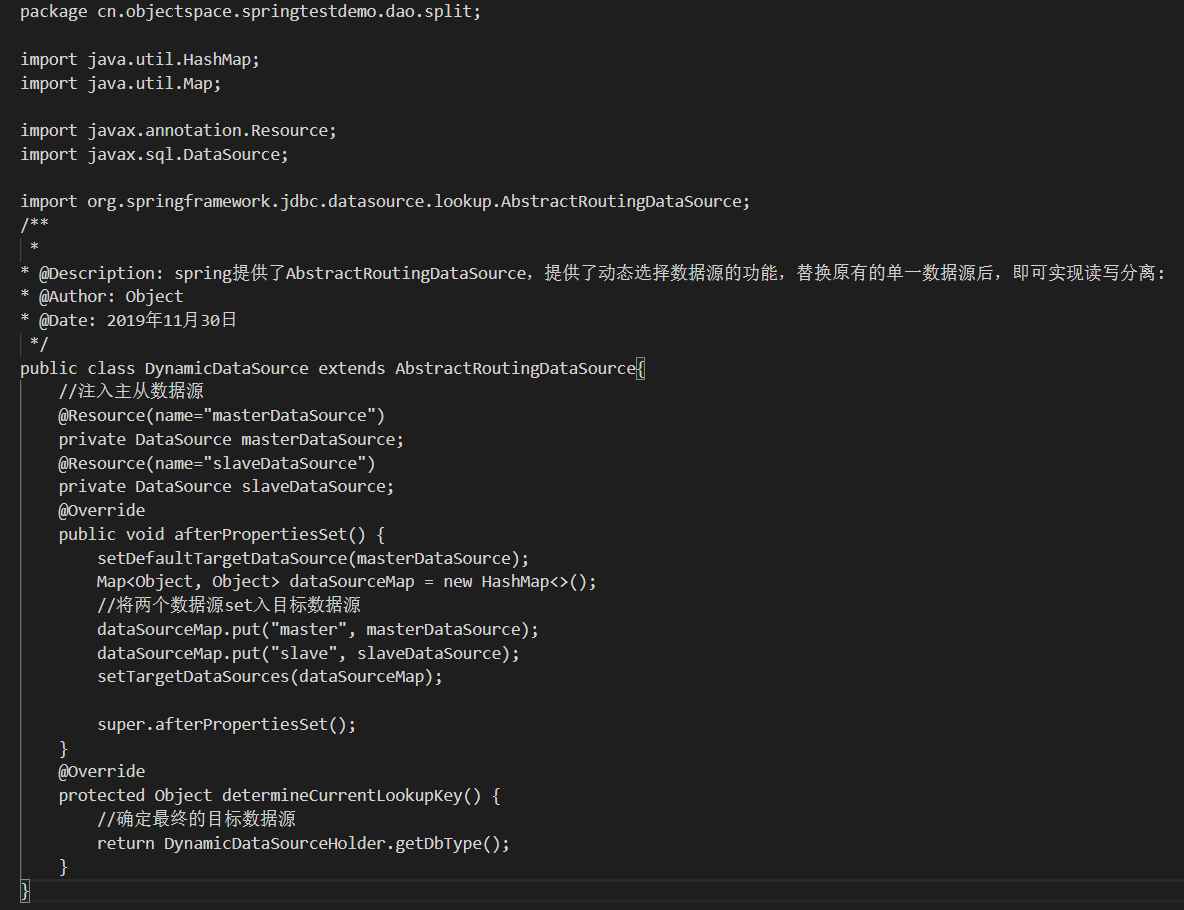
#### DataSourceConfig

这个配置主要是对主从数据源进行配置。



### 动态数据源的切换

这里使用到的主要是Spring提供的AbstractRoutingDataSource，其提供了动态数据源的功能，可以帮助我们实现读写分离。其determineCurrentLookupKey()可以决定最终使用哪个数据源，这里我们自己创建了一个DynamicDataSourceHolder，来给他传一个数据源的类型（主、从）。



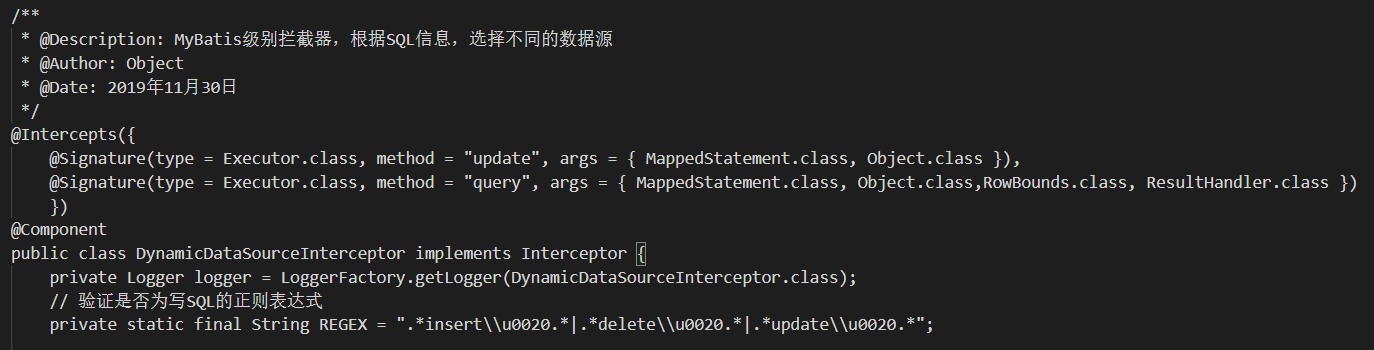
### DynamicDataSourceHolder的实现

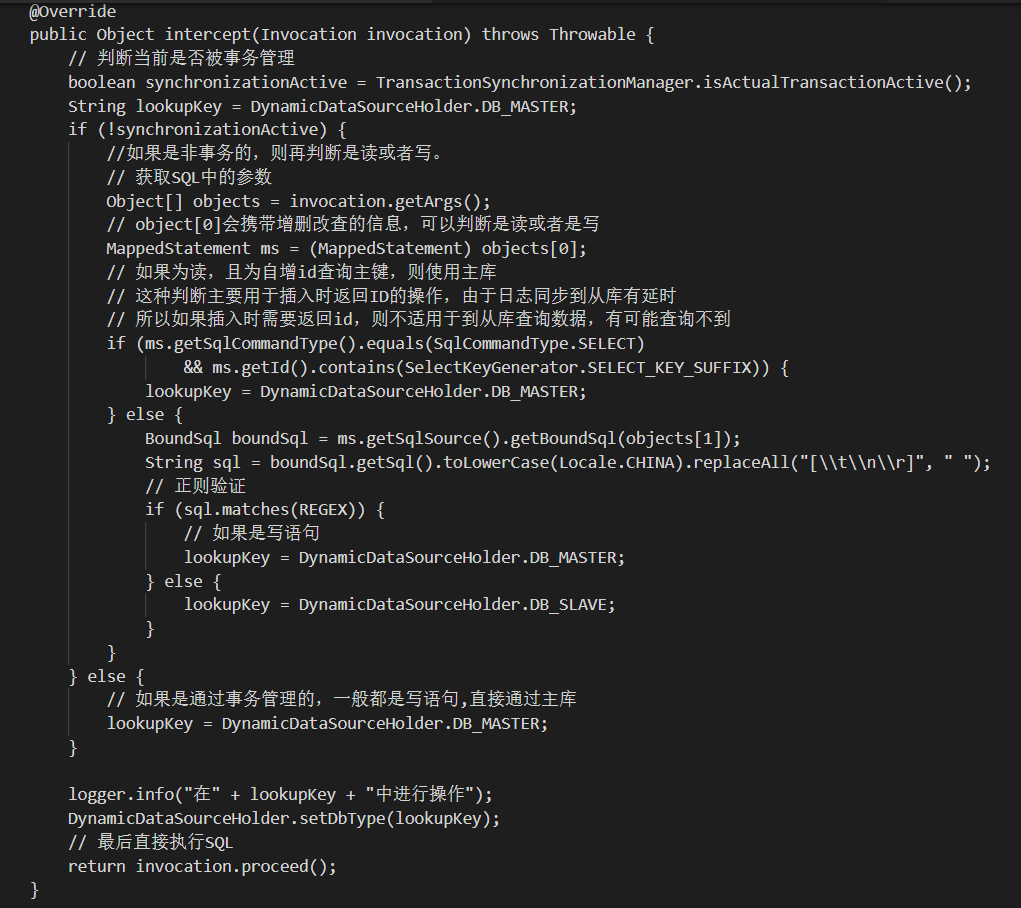
这个类由我们自己实现，主要是提供给Spring我们需要用到的数据源类型。



### MyBatis拦截器的实现

最后就是我们实现读写分离的核心了，这个类可以对SQL进行判断，是读SQL还是写SQL，从而进行数据源的选择，最终调用DynamicDataSourceHolder的setDbType方法，将数据源类型传入。





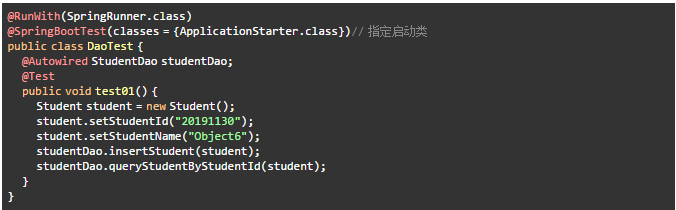
### 代码梳理

通过上文中的程序，我们已经可以实现读写分离了，但是这么看着还是挺乱的。所以在这里重新梳理一遍上文中的代码。

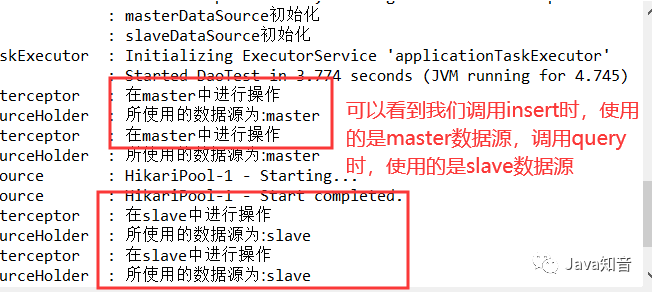
其实逻辑并不难：

* 通过@Configuration实现多数据源的配置。
* 通过MyBatis的拦截器，DynamicDataSourceInterceptor来判断某条SQL语句是读还是写，如果是读，则调用DynamicDataSourceHolder.setDbType("slave")，否则调用DynamicDataSourceHolder.setDbType("master")。
* 通过AbstractRoutingDataSource的determineCurrentLookupKey()方法，返回DynamicDataSourceHolder.getDbType();也就是我们在拦截器中设置的数据源。
* 对注入的数据源执行SQL。

### 测试



测试结果：



至此，代码层读写分离已完整地实现。

## 基于MyCat中间件实现读写分离、故障转移

### 简介

在上文中我们已经实现了使用手写代码的方式对数据库进行读写分离，但是不知道大家发现了没有，我只使用了一主一从。那么**为什么我有一主二从的环境却只实现一主一从的读写分离呢？因为，在代码层实现一主多从的读写分离我也不会写。那么假设数据库集群不止于一主二从，而是一主三从，一主四从，多主多从呢？如果Master节点宕机了，又该怎么处理？**

每次动态增加一个节点，我们就要重新修改我们的代码，这不但会给开发人员造成很大的负担，而且不符合开闭原则。

所以接下来的MyCat应该可以解决这样的问题。并且我会直接使用一主二从的环境演示。

### MyCat介绍

这里直接套官方文档。

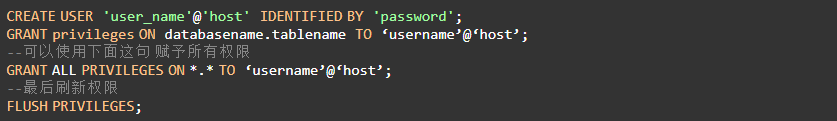
* 一个彻底开源的，面向企业应用开发的大数据库集群
* 支持事务、ACID、可以替代MySQL的加强版数据库
* 一个可以视为MySQL集群的企业级数据库，用来替代昂贵的Oracle集群
* 一个融合内存缓存技术、NoSQL技术、HDFS大数据的新型SQL Server
* 结合传统数据库和新型分布式数据仓库的新一代企业级数据库产品
* 一个新颖的数据库中间件产品

### 环境说明

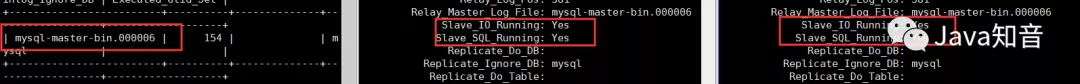
MyCat 192.168.43.90  
MySQL master 192.168.43.201  
MySQL slave1 192.168.43.202  
MySQL slave2 192.168.43.203

接上篇博客的MySQL数据库一主二从，不过**MySQL版本需要从8.0改为5.7**，否则会出现密码问题无法连接。

**另外，我们需要在每个数据库中都为MyCat创建一个账号并赋上权限：**



在开始之前，先保证主从库的搭建是成功的：



如何安装MyCat在这里我就不说了，百度上有很多帖子有，按照上面的教程一步一步来其实没有多大问题。我们着重说说和我们MyCat配置相关的两个配置文件——schema.xml和server.xml，当然还有一个rules.xml，但是这里暂时不介绍分库分表，所以这个暂且不提。

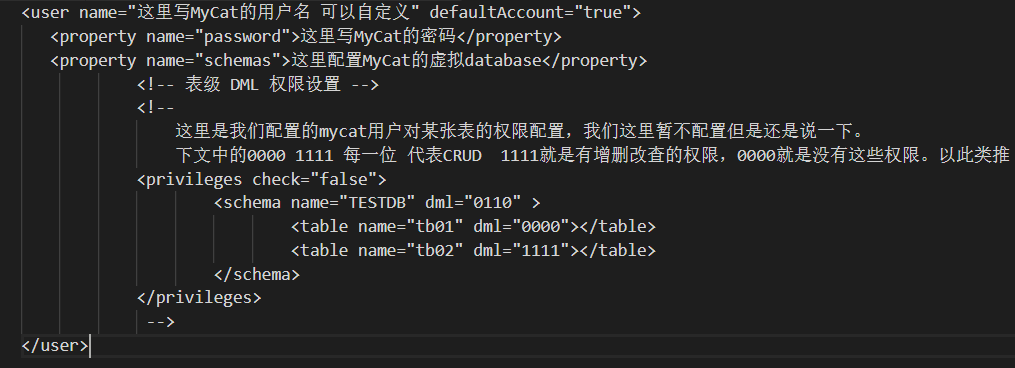
### 配置文件说明

#### server.xml

打开mycat安装目录下的/conf/server.xml文件，这个配置文件比较长，看着比较费脑，但其实对于初学者来说，我们需要配置的地方并不多，所以不用太害怕这种长篇幅的配置文件。（其实在上一篇文章的结尾，我也说过，面对一个新技术的时候首先不能懵逼，一步一步地去分析并接受他，扯远了）配置文件简化之后大概是这样的一个结构。



这样看起来是不是简单多了，其实对于Server.xml，我们主要配置的就是下面的user模块，我们把它展开，着重讲讲这部分的配置。



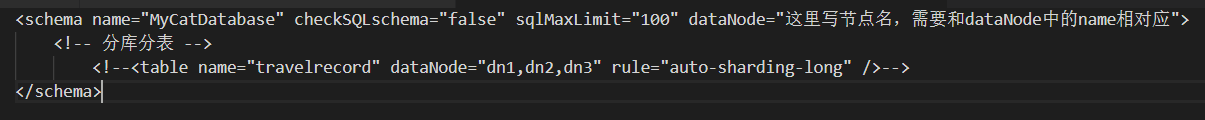
user代表MyCat的用户，我们在使用MySQL的时候都会有一个用户，**MyCat作为一个虚拟节点，我们可以把它想象成它就是一个MySQL**，所以自然而然它也需要有一个用户。但是他的用户并不是我们用命令创建的，而是直接在配置文件中配置好的，我们之后登录MyCat，就是用这里的用户名和密码进行登录。至于如何配置，我在上面的配置中都写好啦。跟着做就没有问题。

#### schema.xml

打开MyCat安装目录的conf/**schema.xml，这个配置文件是我们需要关注的一个配置文件**，因为我们的读写分离、分库分表、故障转移、都配置在这个配置文件中。但是这个配置文件并不长，我们可以一点一点慢慢分析。

首先是<schema></schema>标签中的内容。这个标签主要是为MyCat虚拟出一个数据库，我们连接到MyCat上能看到的数据库就是这里配置的,而**分库分表也主要在这个标签中进行配置**。

这个标签中的name属性，就是为虚拟数据库指定一个名字，也是我们连接MyCat看到的数据库的库名，dataNode是和下文的dataNode标签中的name相对应的，代表这个虚拟的数据库和下面的dataNode进行绑定。

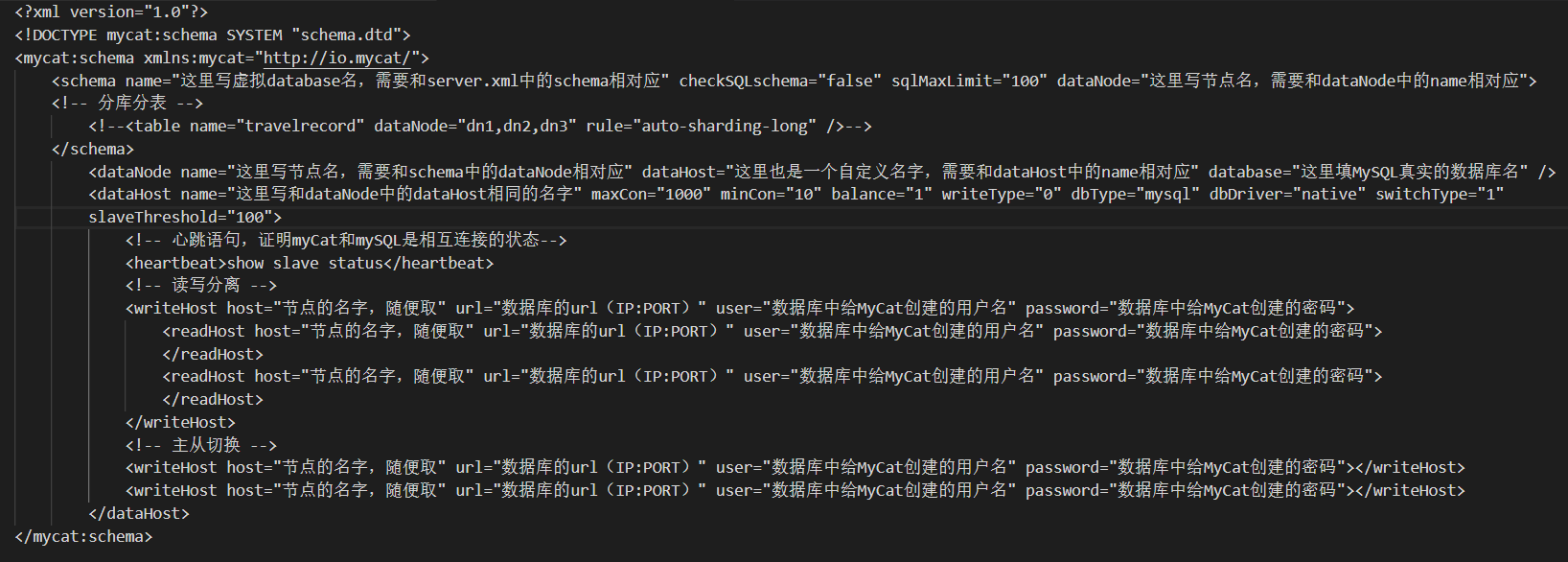


第二个标签是<dataNode/>标签，这个标签是和我们真实数据库中的database联系起来的，name属性是我们对这个dataNode自定义的一个名字，要注意的是，这个名字需要和schema标签中的dataNode内容一致，database属性写的是我们真实数据库中的真实database的名字。而dataHost的内容需要和之后<dataHost></dataHost>标签中的name属性的值相对应。



第三个标签要说的是<dataHost></dataHost>标签，这个标签是和我们真实数据库的主从、读写分离联系起来的标签，什么意思呢。这个标签中有这么两个子标签<writeHost>和<readHost>分别代表我们的写库和读库，<writeHost>中配置的库可以用于读或者写，而<readHost>中配置的库只能用于读。

可以看到schema.xml的配置是一环扣一环的，每个标签之间都有相互进行联系的属性。我们最后配置完的schema.xml应该长下面这个样子：



### MyCat配置读写分离

上文中我们对MyCat的两个配置文件进行了基本的解读，那么现在就开始搭建一个基于MyCat的读写分离。我这里有三个数据库，一主二从，再说一遍环境吧：

192.168.43.201 master库  
192.168.43.202 slave库  
192.168.43.203 slave库

那么对于server.xml和schema.xml的配置如下：

#### server.xml:



#### schema.xml:



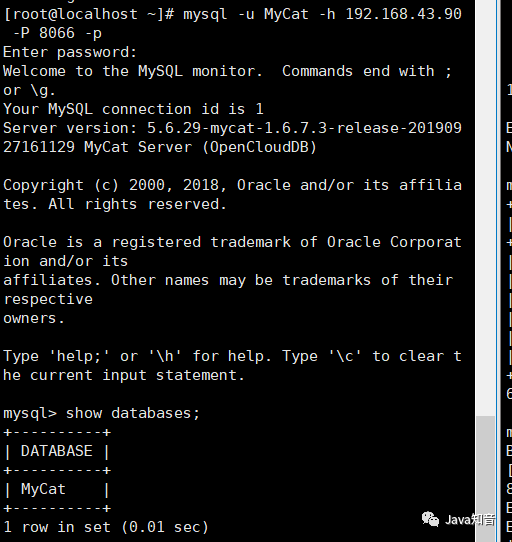
**启动MyCat并测试:**

启动MyCat:

./mycat start

连接MyCat:

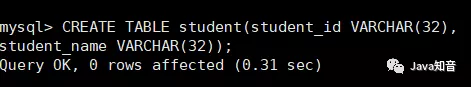
mysql -u MyCat -h 192.168.43.90 -P 8066 -p

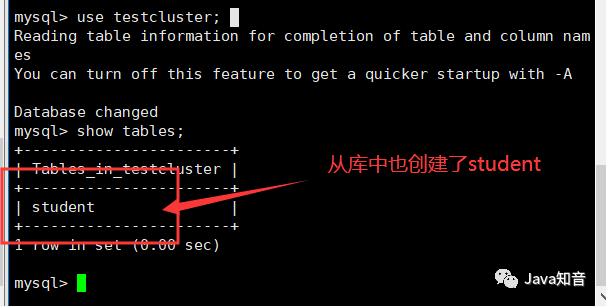


可以正确看到MyCat中存在一个数据库，名字叫MyCat，而这个数据库是我们虚拟出来的，并不真实存在，实际上就是我们配置的<schema name="MyCat"></schema>起了作用。

**在MyCat库中创建一张表**

CREATE TABLE student(student\_id VARCHAR(32),student\_name VARCHAR(32));



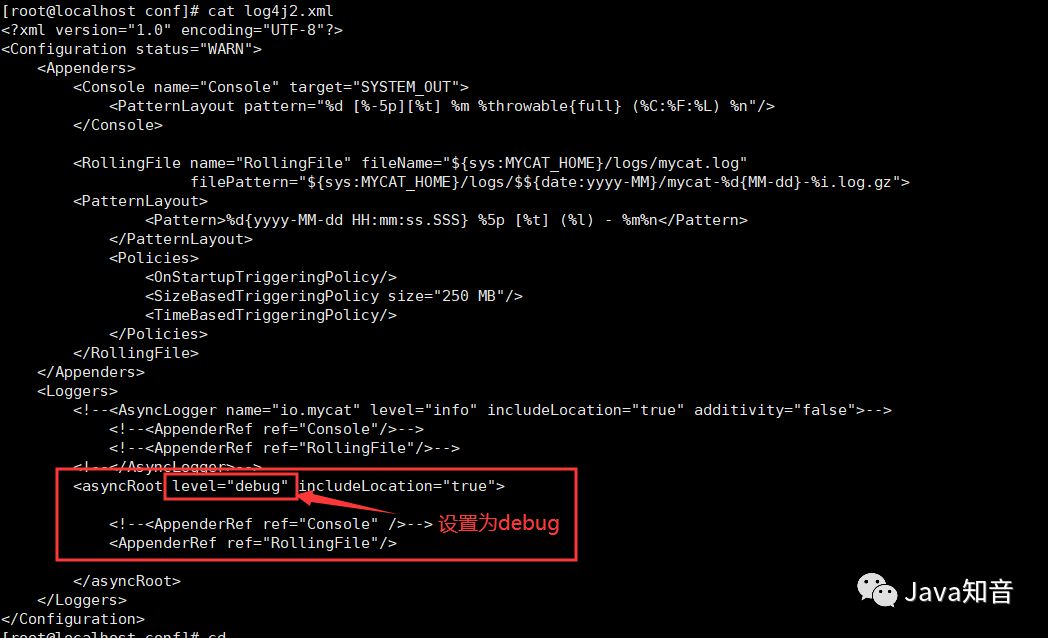


这样就可以证明我们的mycat连接真实数据库成功。那么我们下面就要开始证明读写分离，何谓读写分离呢？就是读数据操作从从库中读取，而主库只负责写操作，下面我们开始进行验证。

在验证之前，我们需要将MyCat的日志设置为debug模式，因为在info模式下，是不能在日志中显示SQL语句转发到哪一个数据库中进行查询的。

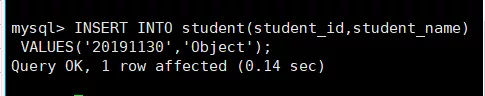
#### 如何设置：

打开conf/log4j2.xml



执行一条插入语句：

INSERT INTO student(student\_id,student\_name) VALUES('20191130','Object');



查看日志：



可以看到，SELECT 语句是在202中执行的，202是Slave库，也就是读库。

再执行一次：



这个时候读语句在203中执行，还是读库,**这两个读库是基于负载均衡规则来进行读取的。**

这样就完成了读写分离的配置，当我们需要进行INSERT/UPDATE/DELETE时，会直接到Master中进行写入，然后同步到Slave库，而要进行SELECT操作时，就改为去Slave中读，不影响Master的写入，**这种读写分离，拓展了MySQL主从同步的功能，可以在容灾备份的同时，提升数据库的性能。**

### MyCat配置故障转移

我们在上文中已经完成了MyCat关于读写分离的配置，那么我们大胆假设，假如我们的Master数据库突然宕机了，那么是否整个集群就丧失了写功能呢？

在没有故障转移之前，这个答案是肯定的，当主库宕机时，从库作为读库，是不会有写的功能的，整个集群也就丧失了写的功能，这是我们不希望看到的。

我们希望看到的场景是：当**主库宕机，某一个从库自动变为主库，承担写的功能，保证整个集群的可用性。**

那么我们开始进行配置，其实思路很简单，MyCat的标签中有一个switchType属性，其决定了切换的条件。

switchType指的是切换的模式，目前的取值也有4种：

* switchType='-1' 表示不自动切换
* switchType='1' 默认值，表示自动切换
* switchType='2' 基于MySQL主从同步的状态决定是否切换,心跳语句为 show slave status
* switchType='3'基于MySQL galary cluster的切换机制（适合集群）（1.4.1），心跳语句为 show status like 'wsrep%'。

我们直接将switchType修改为2，然后将两个读库配置为第一个写库同级的写库。

配置文件如下:



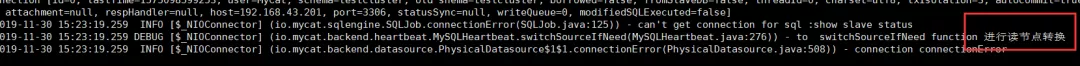
#### 重启MyCat

现在我们来停掉Master库，然后执行写操作，看看是什么结果。

service mysqld stop

MyCat日志：





执行

INSERT INTO student(student\_id,student\_name)VALUES('test','testdown');

MyCat日志



可以看到，现在当我们执行完这个语句时，他自动切换到202数据库进行写入，而202是Slave而非master，这就说明MyCat对写库进行了自动切换，我们的MySQL集群依旧可以提供写的功能。

**当然，此时我们MySQL的主从架构已经被破坏，如果需要恢复主从结构，就需要手动地重新去恢复我们的主从架构。我们需要将201和203作为Slave，202作为Master，因为Master拥有最完整的数据。**

## 优劣分析

关于这两种方式的优劣，相信如果仔细看完这篇文章的同学都会有一个深刻的体会。

**代码层实现读写分离**，主要的优点就是**灵活，可以自己根据不同的需求对读写分离的规则进行定制化开发**，但其缺点也十分明显，就是当我们动态增减主从库数量的时候，都需要对代码进行一个或多或少的修改。并且当主库宕机了，如果我们没有实现相应的容灾逻辑，那么整个数据库集群将丧失对外的写功能。

**使用MyCat中间件实现读写分离**，优点十分明显，我们只需要进行配置就可以享受读写分离带来的效率的提升，不用写一行代码，并且当主库宕机时，我们还可以通过配置的方式进行主从库的自动切换，这样即使主库宕机我们的整个集群也不会丧失写的功能。其缺点可能就是我们得多付出一台服务器作为虚拟节点了吧，毕竟服务器也是需要成本的。

**两种方式如何抉择：**如果你目前的项目比较小，或者干脆是一个毕业设计、课程设计之类的，不会有动态增减数据库的需求，那么自己动手实现一个数据库的读写分离会比较适合你，毕竟答辩的时候，可以一行一行代码跟你的导师和同学解（zhuang）释（bi）。如果项目比较大了，数据库节点有可能进行增减，并且需要主从切换之类的功能，那么就使用第二种方式吧。这种配置化的实现可以降低第二天洗头时候下水管堵塞的几率。