哈尔滨工业大学

**计算学部**

**2021年春季学期**

**《软件架构与中间件》课程**

**实验报告**

**Lab 3-4：数据层及表示层软件架构实验**

|  |  |  |
| --- | --- | --- |
| **姓名** | **学号** | **联系方式** |
| 梅智敏 | 1183710118 | 1044388658@qq.com |
| 王国庆 | 1183710131 | 1113292969@qq.com |

目 录

[1 实验目的 1](#_Toc75712553)

[2 实验要求 1](#_Toc75712554)

[3 实验内容与过程 1](#_Toc75712555)

[3.1 Mycat数据库分库分表实验 1](#_Toc75712556)

[3.2 Sharding-JDBC数据库分库分表实验 11](#_Toc75712557)

[3.3 Redis数据缓存实验 19](#_Toc75712558)

[3.4 利用React进行表示层的设计及实现 26](#_Toc75712559)

[4 结对开发过程记录 27](#_Toc75712560)

[5 实验总结 29](#_Toc75712561)

[文档全部完成之后，请在上述区域点击右键，选择“更新域”，在打开的对话框中选择“更新整个目录”]

# 实验目的

1）学习使用Mycat和Sharding-JDBC实现数据分库分表

2）学习使用Redis数据库实现数据缓存

3）能够灵活应用Mycat或Sharding-JDBC实现分库分表架构到实际系统

4）能够灵活应用Redis实现数据缓存架构到实际系统

5）学会数据表示层的设计及实现（Lab4）

# 实验要求

1）2人结对成组

2）实验3.1、3.2、3.3均为必做

3）结合《软件过程与工具》课程中进销存系统(或其他实际软件系统)进行数据层架构重构，实现根据业务垂直划分的数据库分库分表；面向海量数据带来的数据检索慢问题，实现数据库水平分片，达到数据检索的性能提升；利用缓存架构实现数据读取的性能提升。

4）应给出关键过程的细节

# 实验内容与过程

根据提供的实验指导书完成实验3.1-3.3，并回答相关问题。

## Mycat数据库分库分表实验

1. 请给出Mycat配置安装过程中遇到的问题和解决方案。

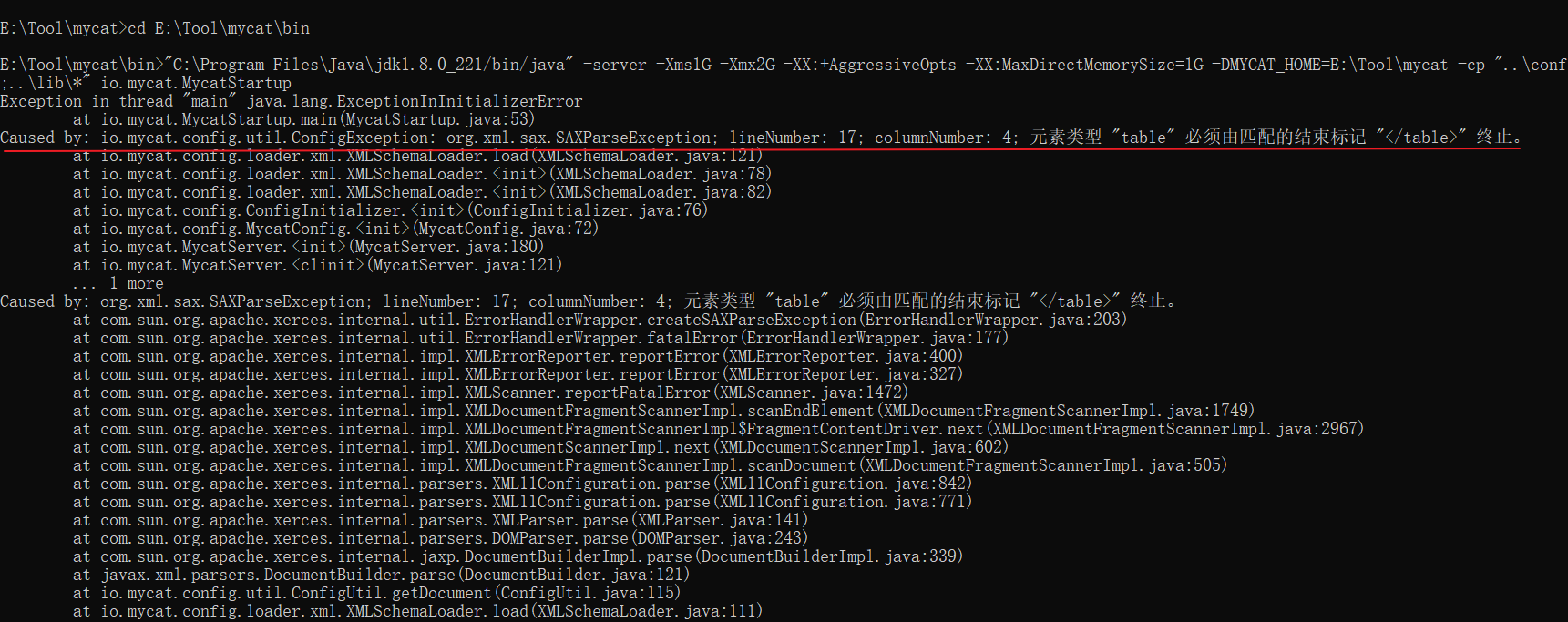
**问题：**



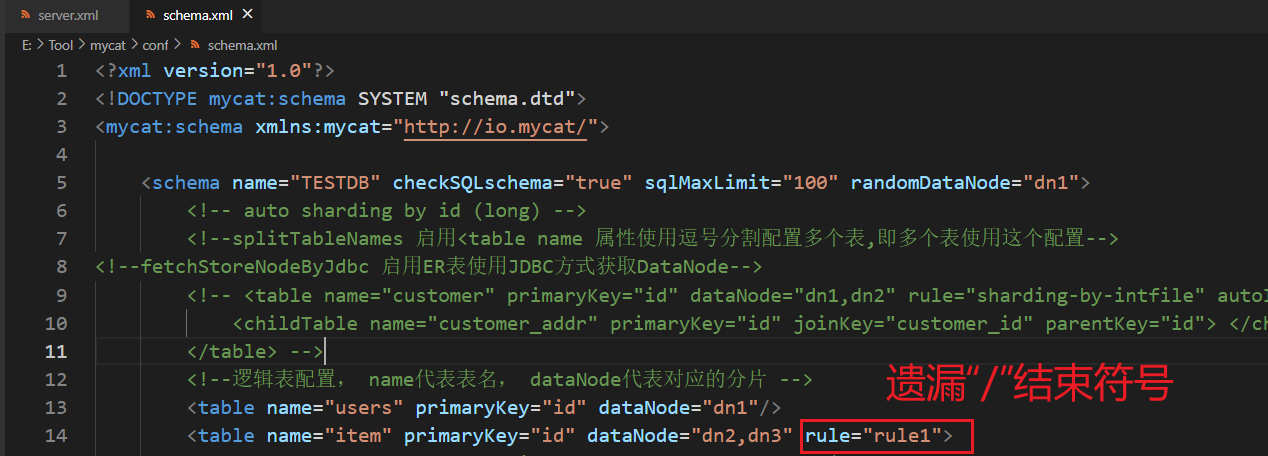
配置完成之后，双击运行上述.bat文件，发现会闪退。

**解决方案：**

使用cmd模式再次运行，即可查看报错信息



根据报错信息找到schema.xml，发现遗漏了符号“/”



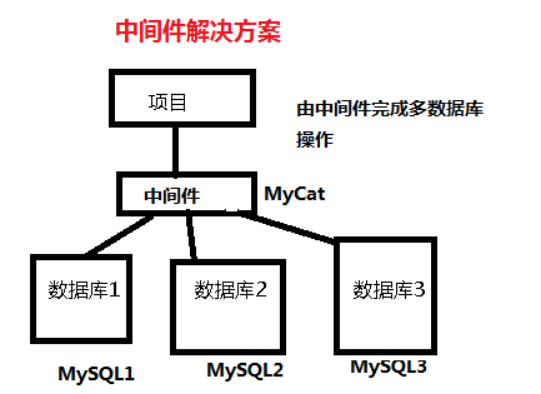
补上之后即可成功启动



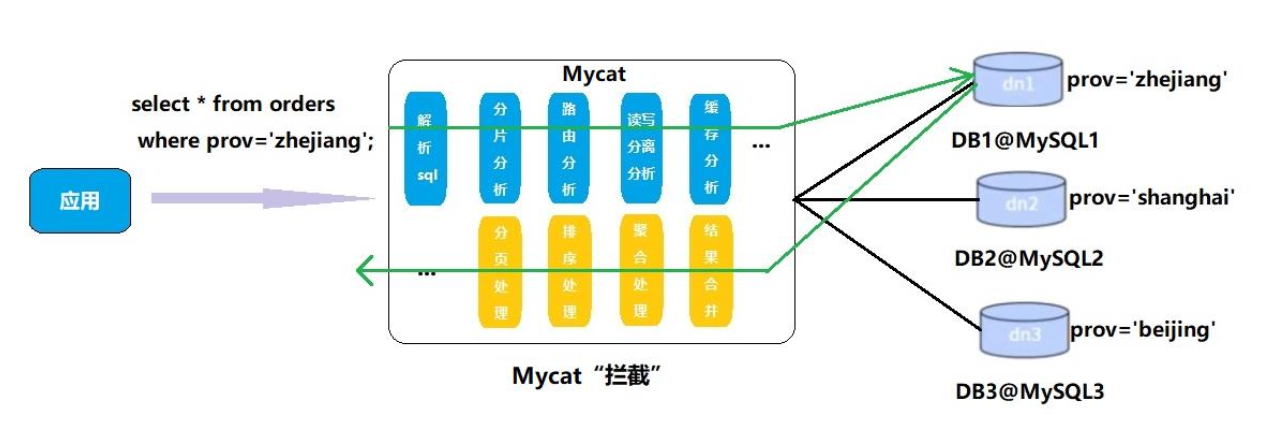
2）请详析Mycat的分库分表原理和操作方法。

**分库分表原理：**

* MyCat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是一个数据库代理，用MySQL客户端工具和命令行访问，而其后端可以用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，即可以将一个较大的数据库垂直分割成多个数据库，也可以将一个大表水平分割为多个小表，存储在后端MySQL服务器里或者其他数据库里。



* Mycat通过定义路由规则（其中定义分片字段和分片算法）来实现分库分表，它会对所有传递的sql语句做路由处理（路由处理的依据就是表是否分片，如果分片，那么需要依据分片字段和对应的分片算法来判断sql应该传递到哪一个、或者哪几个、又或者全部节点去执行）。
* 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。当 Mycat收到一个SQL时，会先解析这个SQL，查找涉及到的表，然后看此表的定义，如果有分片规则， 则获取到SQL里分片字段的值，并匹配分片函数，得到该SQL对应的分片列表，然后将SQL发往这些分片去执行，最后收集和处理所有分片返回的结果数据，并输出到客户端。

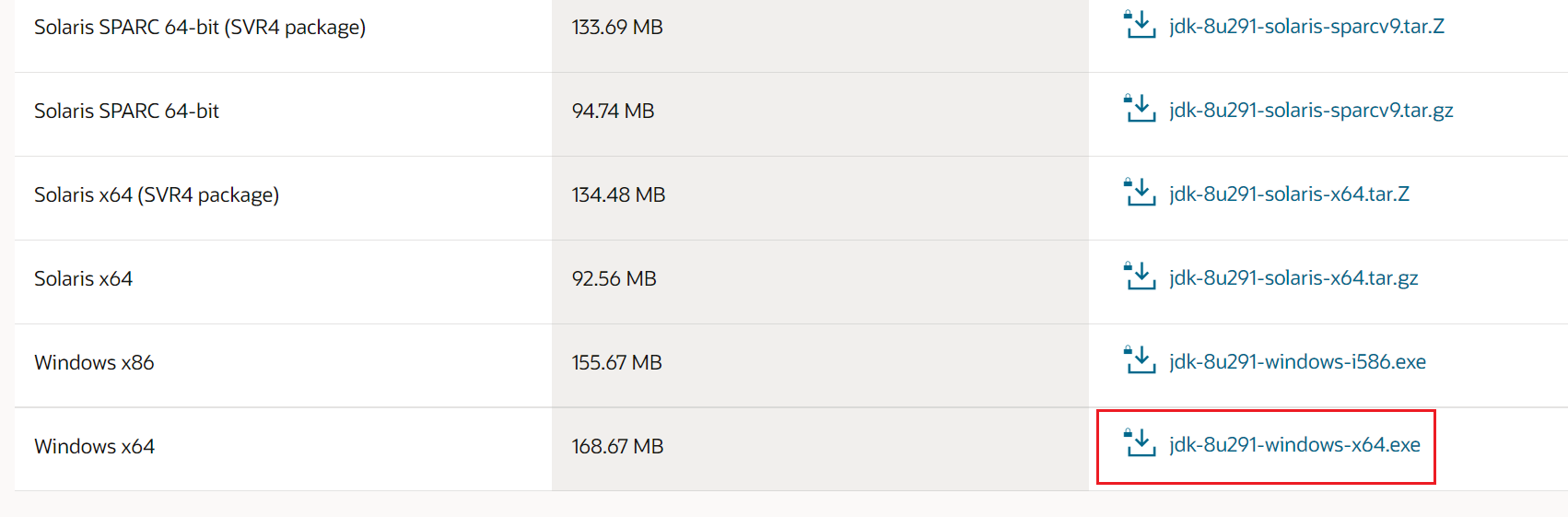


这种方式把数据库的分布式从代码中解耦出来，程序员察觉不出来后台使用 Mycat 还是普通的MySQL。

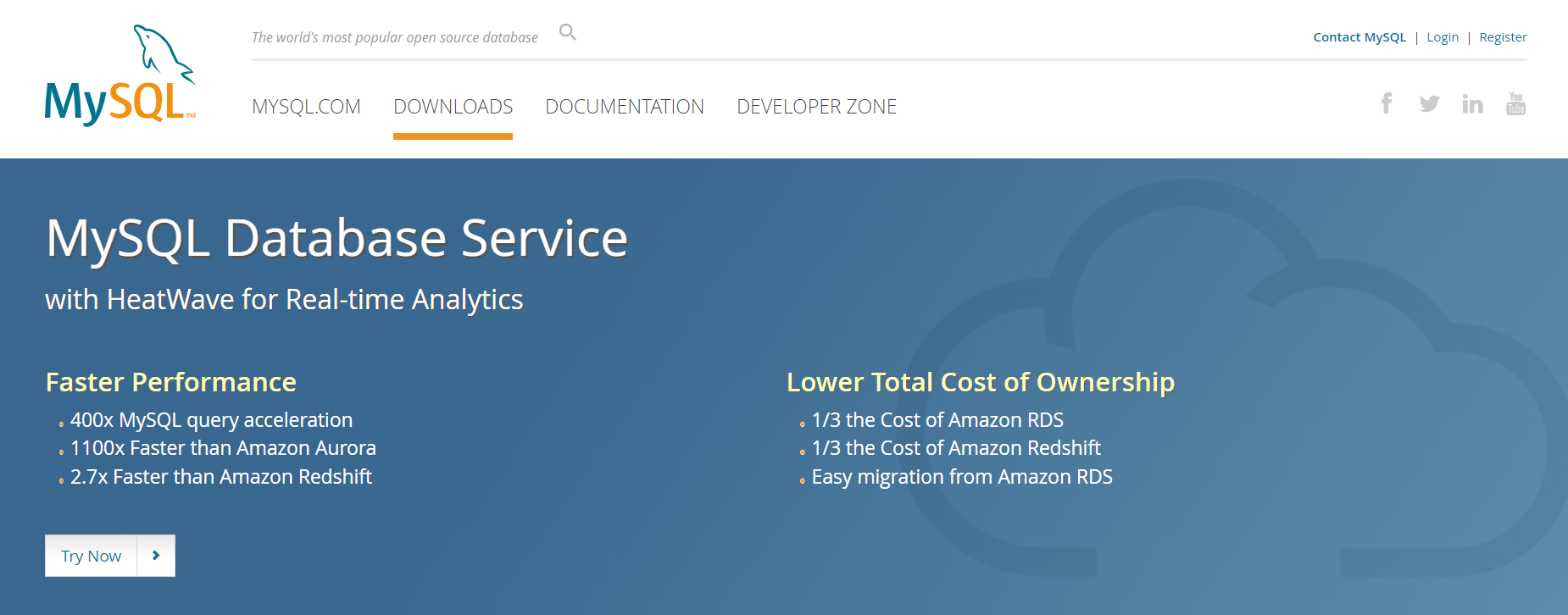
**操作方法：**

* **基本开发环境准备**

安装JDK1.8并完成环境配置



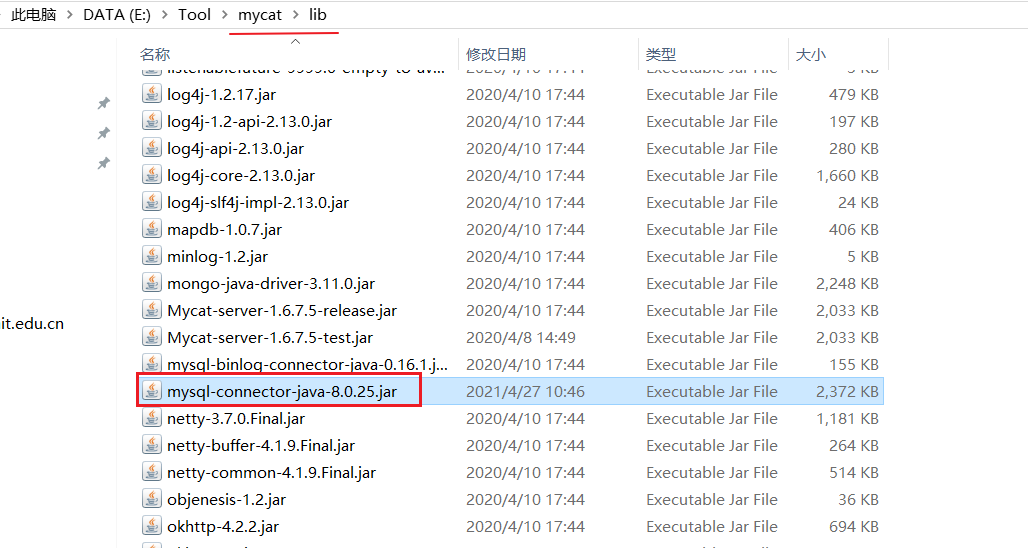
安装MySQL8.0并完成环境配置



安装MyCat-1.6.7.6

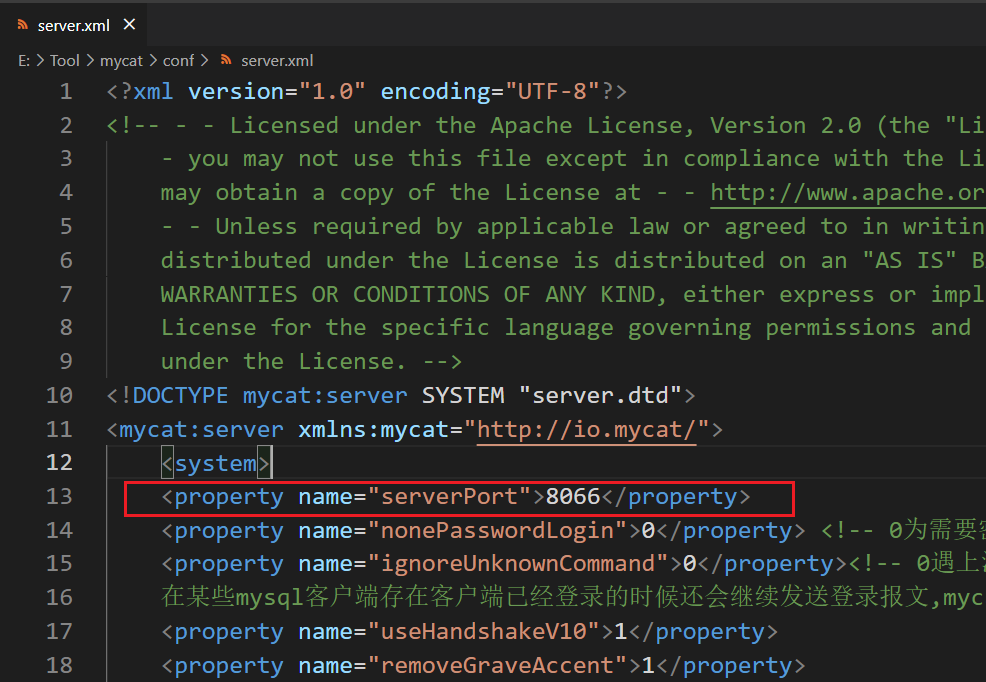


在Mycat的安装目录的lib子目录下，下载MySQL8.0版本的JDBC驱动，并替换默认的MySQL5.0版本的驱动



* **Mycat系统参数配置**

编辑mycat/conf/server.xml，在system标签下添加启动端口配置，端口设置为8066：



设置登陆Mycat的用户名、密码、逻辑库：



* **Mycat逻辑库、表分片配置**

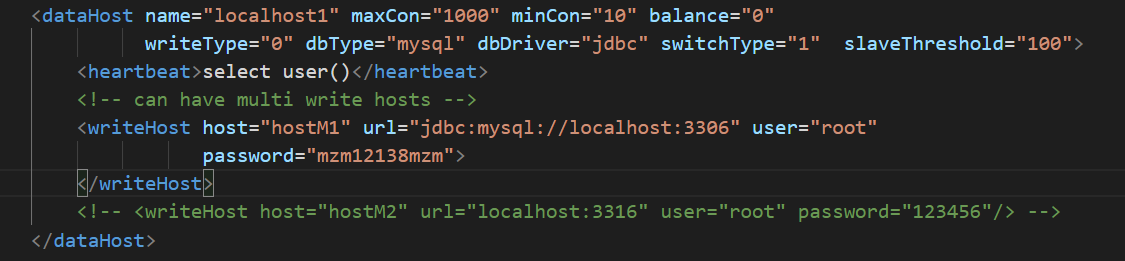
编辑mycat/conf/schema/xml，配置使用逻辑数据库为TESTDB，配置逻辑表为users和item，配置它们的主键和对应的物理数据库位置，以及切分表的规则（如有分片的话）



设置dataNode对应的数据库，以及Mycat连接的地址dataHost；表切分后需要配置映射到哪几个数据库中，Mycat的分片实际上就是数据库的别名；例如配置了三个分片dn1,dn2,dn3分别对应到物理机映射dataHost localhost1的三个库上



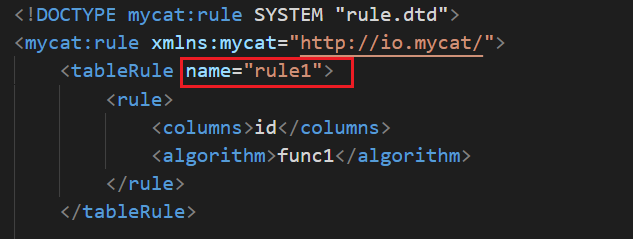
接下来配置物理库分片映射，Mycat作为数据库代理需要逻辑库、逻辑用户、表切分后需要配置分片，分片也就需要映射到真实的物理主机上。



* **Mycat表切分规则配置**

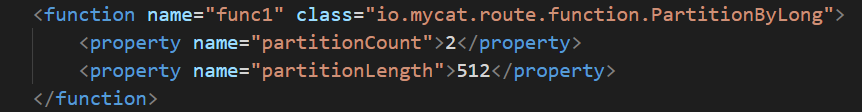
编辑mycat/conf/rule.xml，此配置文件用于编写表切分规则，它决定了数据切分后性能的好坏，因此是最重要的配置。

name为schema.xml 中 table 标签中对应的 rule="rule1" ,也就是配置表的分片规则；columns 是表的切分字段；algorithm 是规则对应的切分规则：映射到 function 的 name



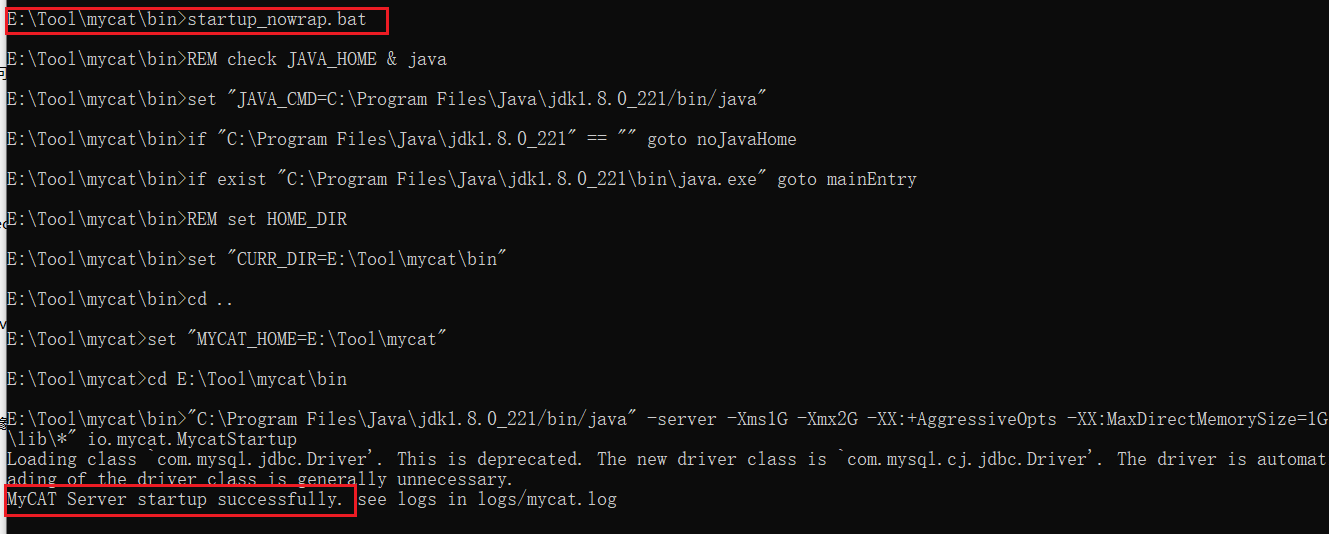
前面的algorithm所指定的切分规则会映射到function，function配置了分片规则。

name 为切分规则的名称，名字任意取，但是需要与tableRule 中匹配；class 是切分规则对应的切分类，写死，需要哪种规则则配置哪种；property 标签是切分规则对应的不同属性，不同的切分规则配置不同



* **启动Mycat**

使用cmd运行bin目录下的startup\_nowrap.bat即可

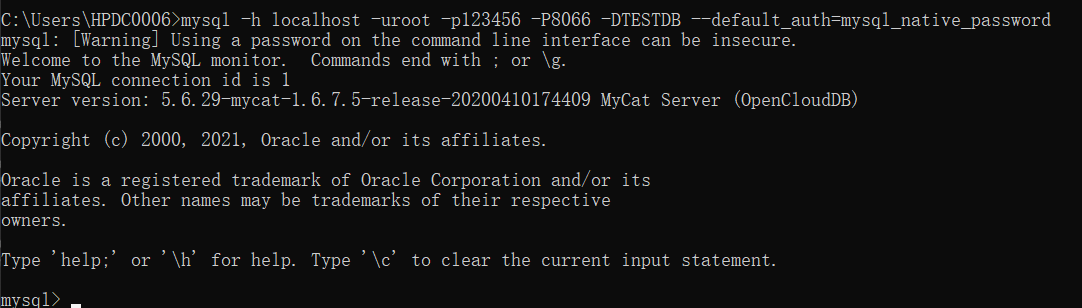


* **在物理数据库上操作**

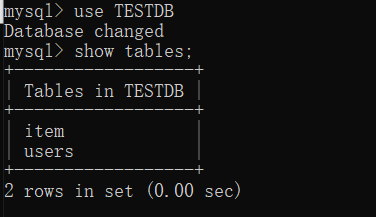
首先进入本地的物理数据库，按照指导书要求创建3个数据库db1、db2、db3，并分别在各个数据库中创建一些Table

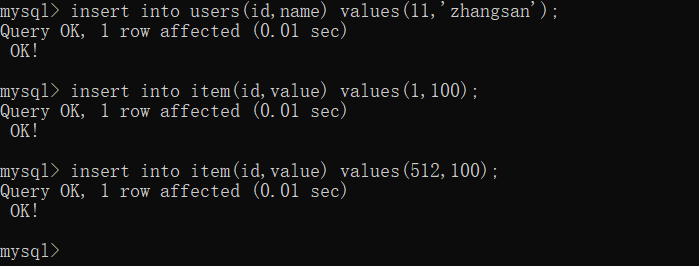


连接mycat

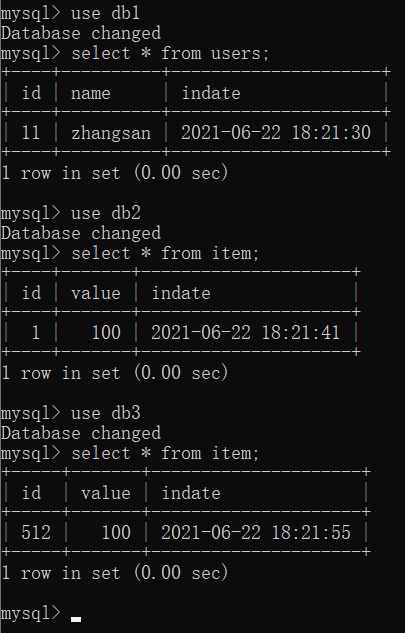


插入三条数据：





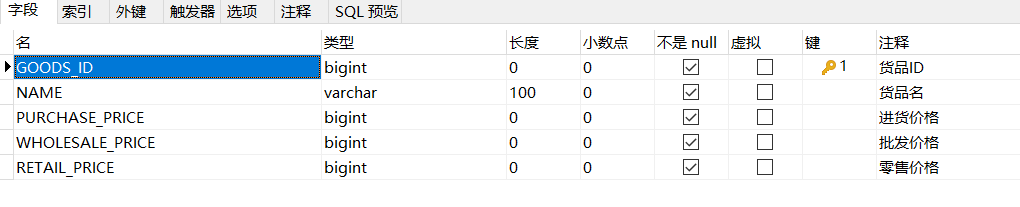
登录物理数据库，检验是否成功加入



可以发现：插入的users表中的数据全部在db1中，而item表中的数据分布在db2和db3中，这样就根据实际的路由策略进行了分表。

3）请在进销存系统(或其他实际软件系统)创建具有复杂表结构和含有较大数据量的数据库表， 并基于此库表描述分库分表的结果，且验证分库分表的效果。

进销存系统中存在Table：t\_goods，存储全部的商品价格信息，各个字段如下：

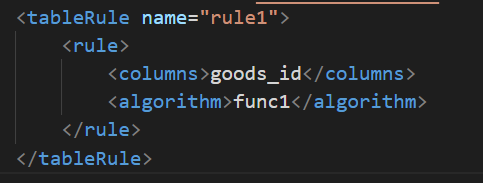


由于进销存系统中涉及到许多种货品，故该表内容极为庞大，所以我们需要对该表进行分片以提高后续在该表上的sql语句执行效率。

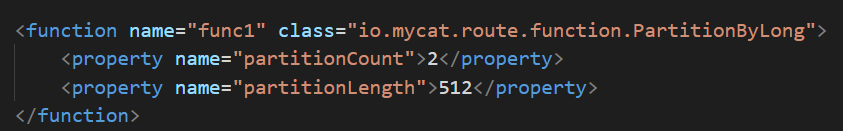
将t\_goods进行分片：dn1和dn2，分片规则为rule1



rule1按照goods\_id属性来进行分片，规则为func1



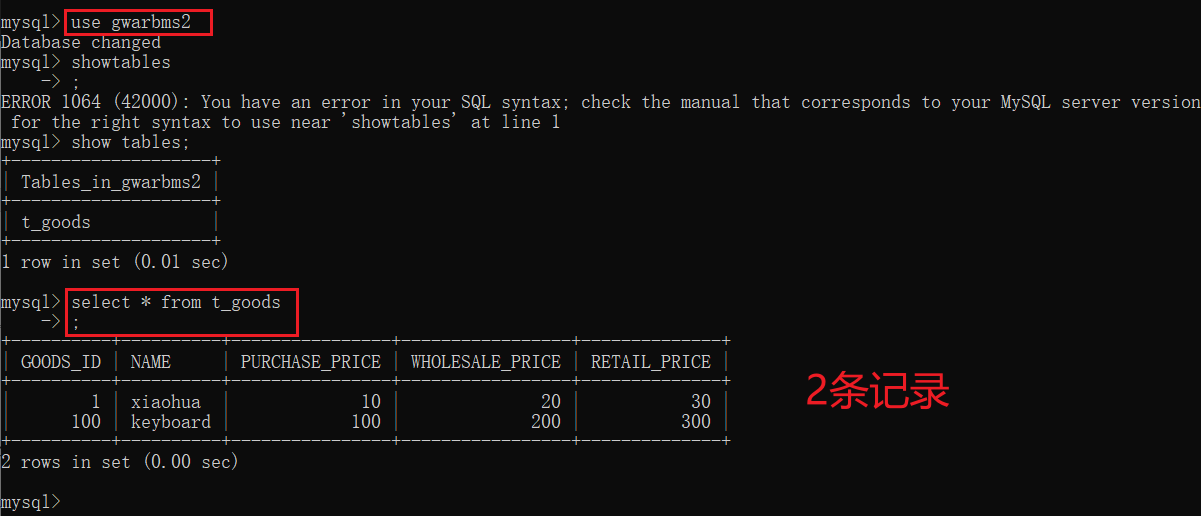
Func1规定单表最大长度512

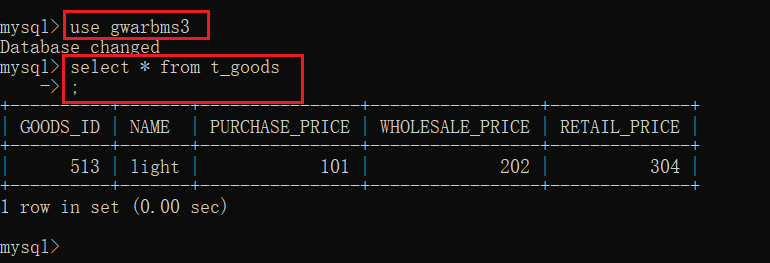


下面进行插入数据验证，首先连接Mycat插入3条数据



进入真实物理数据库查看结果：





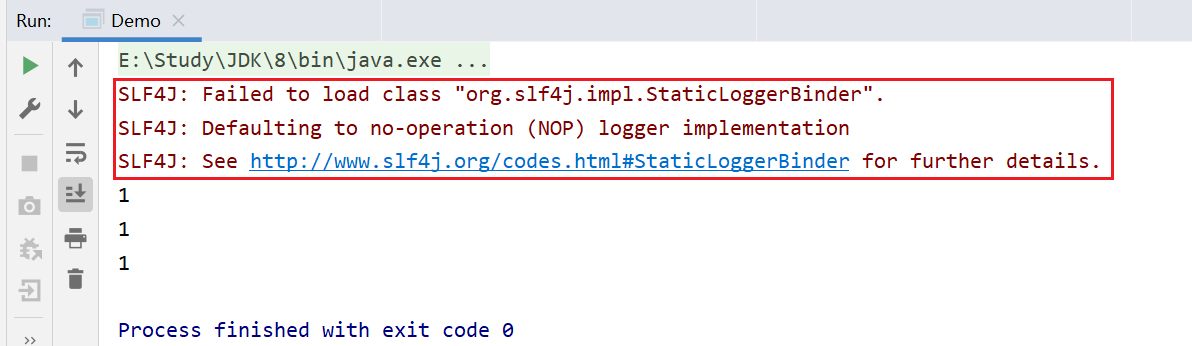
可以发现的确按照我们预先设置的分片规则func1来实现了数据库表的分片（goods\_id属性值低于512的被分到gwarbms2中，否则被分到gwarbms3中）。

## Sharding-JDBC数据库分库分表实验

1. 请给出Sharding-JDBC配置安装过程中遇到的问题和解决方案。

**问题：**

运行实验指导书中的Demo.main()之后有报错：



**解决方案：**

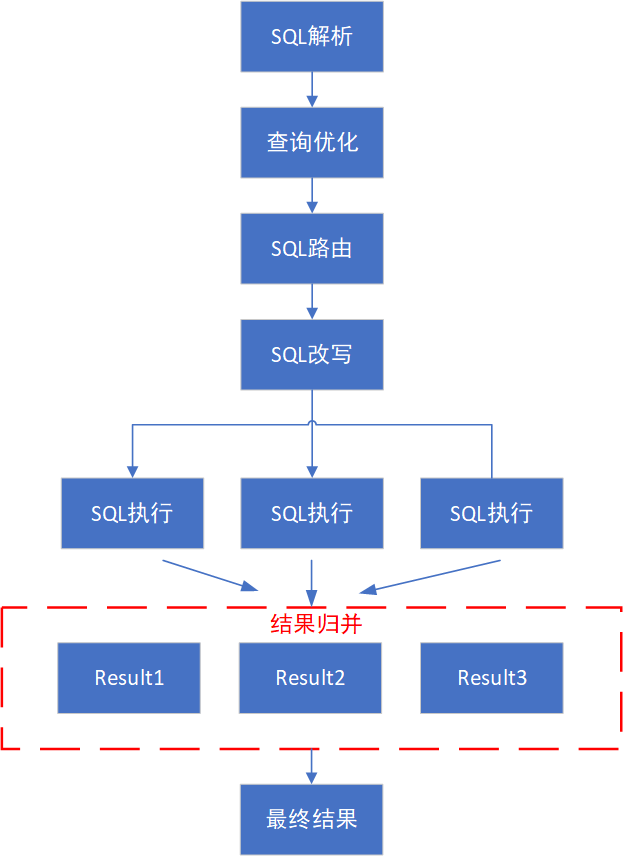
在maven配置文件pom.xml中引入缺少的日志包org.slf4j即可解决



2）请详析Sharding-JDBC的分库分表原理和操作方法。

**原理：**

一张表经过分库分表后被拆分成多个子表，并分散到不同的数据库中，在不修改原业务 SQL 的前提下，Sharding-JDBC 会对 SQL进行一些改造才能正常执行。大致的执行流程：SQL 解析 -> 执⾏器优化 -> SQL 路由 -> SQL 改写 -> SQL 执⾏ -> 结果归并

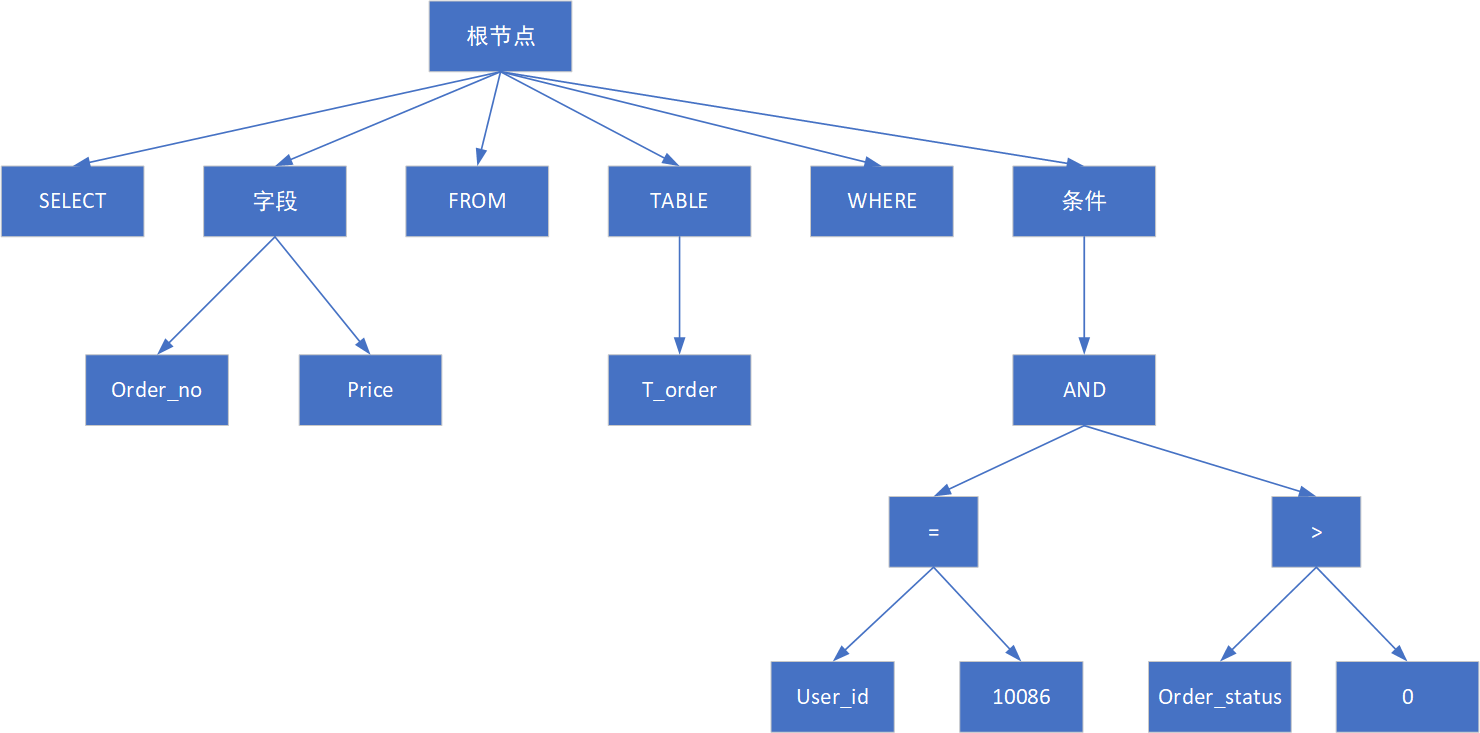


* **SQL解析**

SQL解析过程分为词法解析和语法解析两步，比如下边这条查询用户订单的SQL，先用词法解析将SQL拆解成不可再分的原子单元。在根据不同数据库方言所提供的字典，将这些单元归类为关键字，表达式，变量或者操作符等类型。



接着语法解析会将拆分后的SQL转换为抽象语法树，通过对抽象语法树遍历，提炼出分片所需的上下文，上下文包含查询字段信息（Field）、表信息（Table）、查询条件（Condition）、排序信息（Order By）、分组信息（Group By）以及分页信息（Limit）等，并标记出 SQL中有可能需要改写的位置。



* **查询优化**

执⾏器优化对SQL分片条件进行优化，处理像关键字OR这种影响性能的部分。

* **SQL路由**

SQL 路由通过解析分片上下文，匹配到用户配置的分片策略，并生成路由路径。简单点理解就是可以根据我们配置的分片策略计算出 SQL该在哪个库的哪个表中执行，而SQL路由又根据有无分片健区分出“分片路由”和“广播路由”。

有分⽚键的路由叫分片路由，细分为直接路由、标准路由和笛卡尔积路由这3种类型。

* **SQL改写**

分为两部分，一部分是将分表的逻辑表名称替换为真实表名称。另一部分是根据SQL解析结果替换一些在分片环境中不正确的功能。

* **SQL执⾏**

将路由和改写后的真实 SQL 安全且高效发送到底层数据源执行。但这个过程并不是简单的将 SQL 通过JDBC 直接发送至数据源执行，而是平衡数据源连接创建以及内存占用所产生的消耗，它会自动化的平衡资源控制与执行效率。

* **结果归并**

将从各个数据节点获取的多数据结果集，合并成一个大的结果集并正确的返回至请求客户端，称为结果归并。而我们SQL中的排序、分组、分页和聚合等语法，均是在归并后的结果集上进行操作的。

**操作方法：**

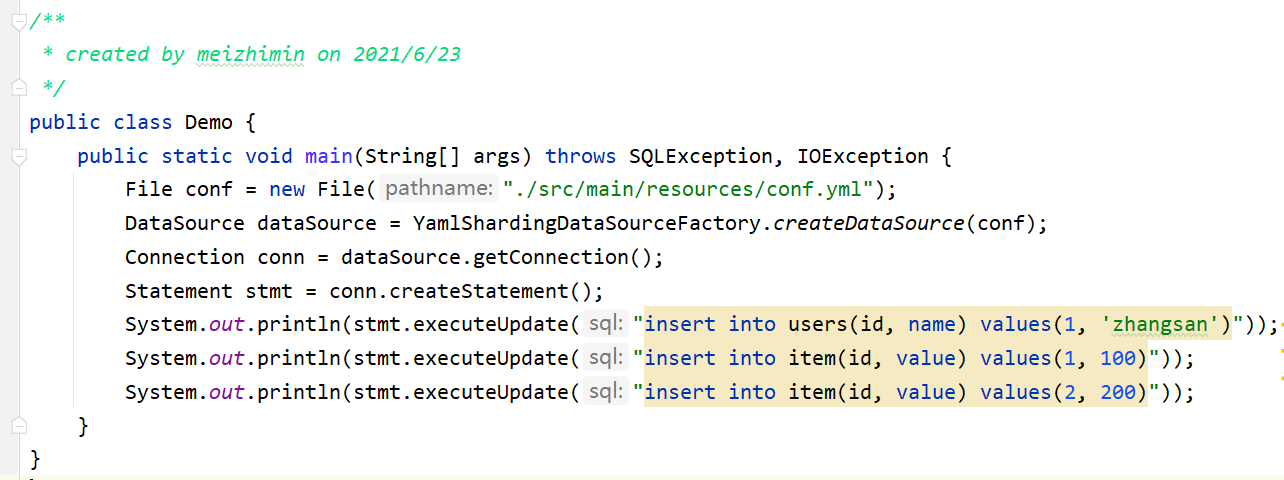
* **登录物理数据库并创建库表**



* **创建maven项目，并配置pom文件**



* **创建/src/main/java/Demo.java并编写**

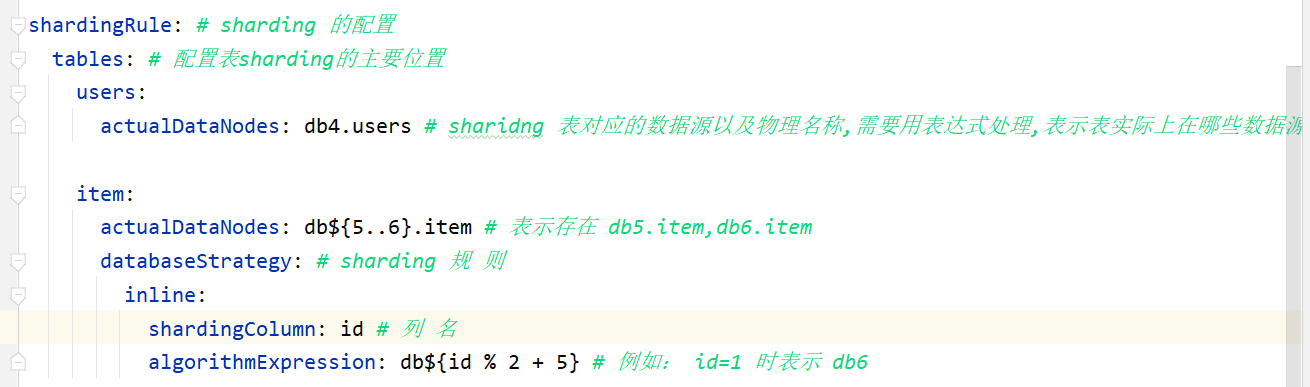


* **编写文件/src/main/resources/conf.yml**

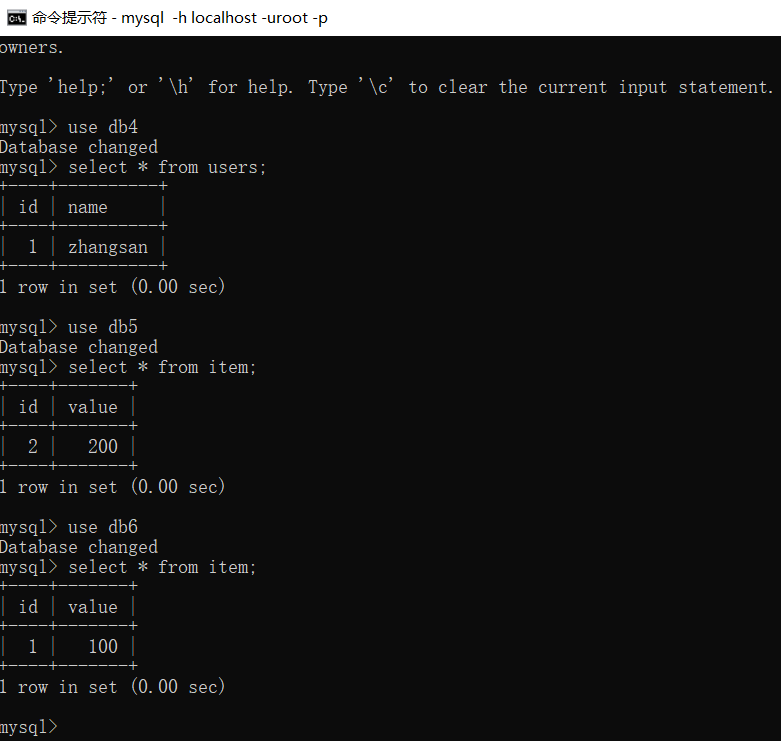
配置好数据源列表如下：



也配置好分库分表规则：



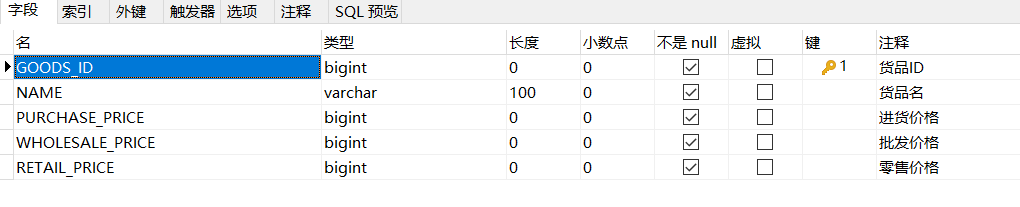
* **运行Demo.mian()后再登入物理数据库检验是否插入成功**



结果证明确实按照我们预先设定的分片规则将不同的数据分别存储到了不同的物理数据节点中。

3）请在进销存系统(或其他实际软件系统)创建具有复杂表结构和含有较大数据量的数据库表， 并基于此库表描述分库分表的结果，且验证分库分表的效果。

进销存系统中存在Table：t\_goods，存储全部的商品价格信息，各个字段如下：

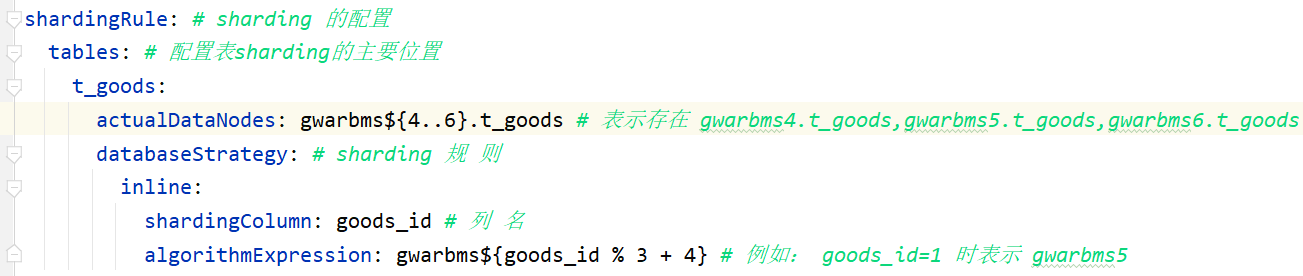


由于进销存系统中涉及到许多种货品，故该表内容极为庞大，所以我们需要对该表进行分片以提高后续在该表上的sql语句执行效率。

首先配置数据源列表：共使用三个数据源



然后配置分库分表规则，按照goods\_id%3的结果将各个数据分别存储到不同的库表中。



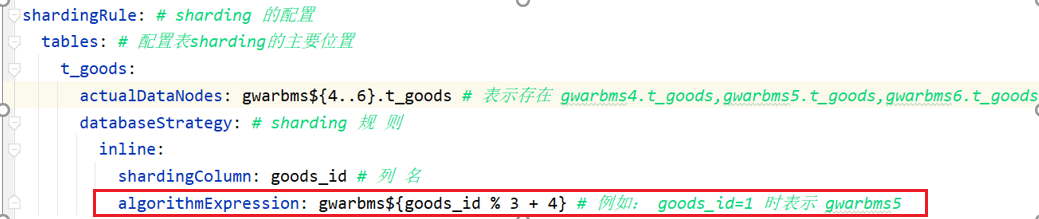
编写测试主程序，往t\_goods中插入6条数据记录，goods\_id从1到6



执行完main之后，登入物理数据库检验是否成功插入



可以发现，插入到逻辑表t\_goods中的数据按照我们预先设定的分库分表规则，被插入到三个不同的数据源中：数据被插入到gwarbms${goods\_id%3+4}中。



通过上述实验证明我们确实利用shardingJDBC完成了数据库的分库分表！

## Redis数据缓存实验

1）请给出Redis配置安装过程中遇到的问题和解决方案。

问题1：解压redis包后执行make命令时无法识别。

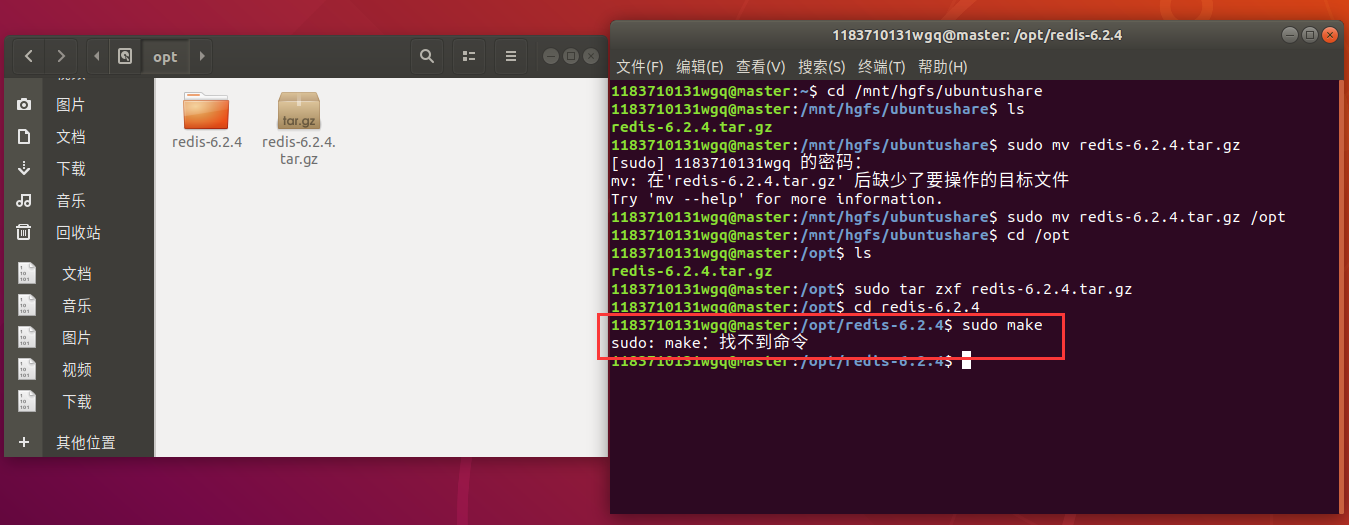


图3.3.1 无法识别make指令

解决方案：缺少make指令工具包，sudo apt update更新软件包后，再apt install make工具包即可，如下图。

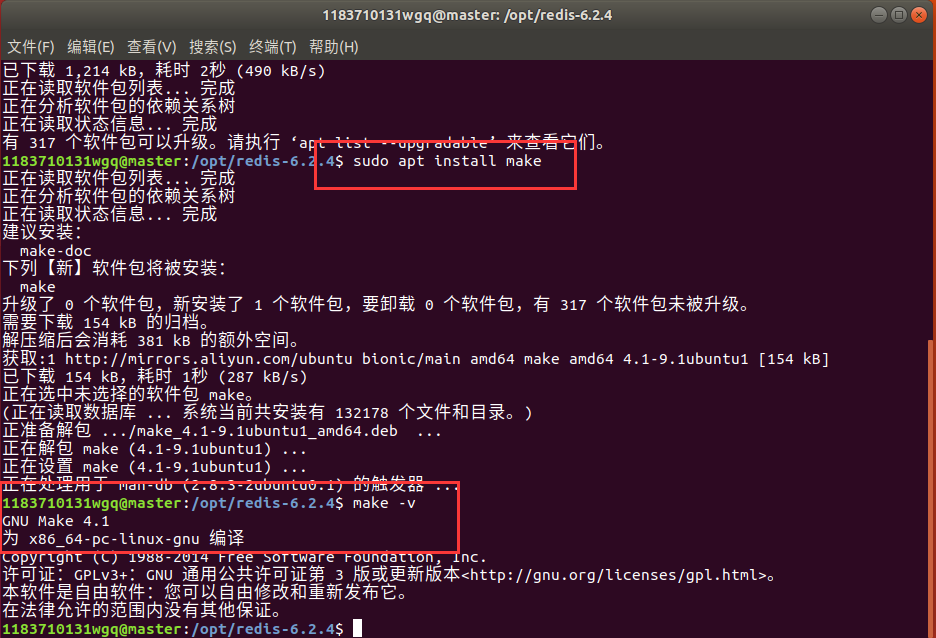


图3.3.2 问题1解决方案

问题2：在redis目录下执行make报错ERROR 2和ERROR 127，并且继续执行redis启动命令失败。

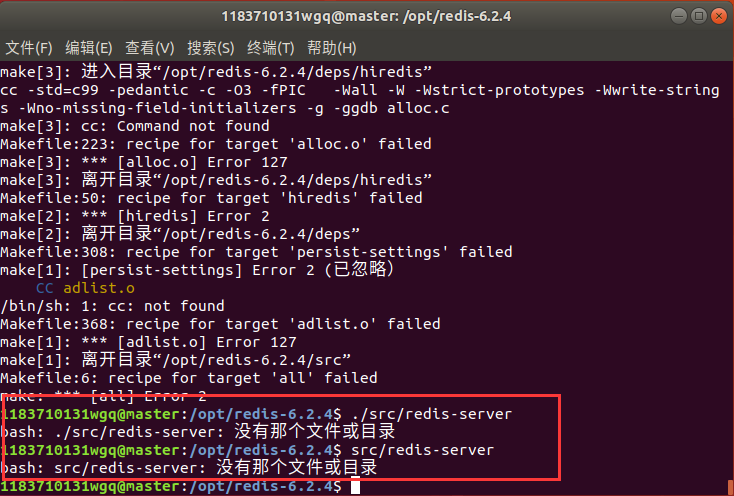


图3.3.3 无法识别redis相关命令

解决方案：因为缺少gcc工具包，所以make指令缺少相关依赖环境，apt install gcc安装即可，如下图。

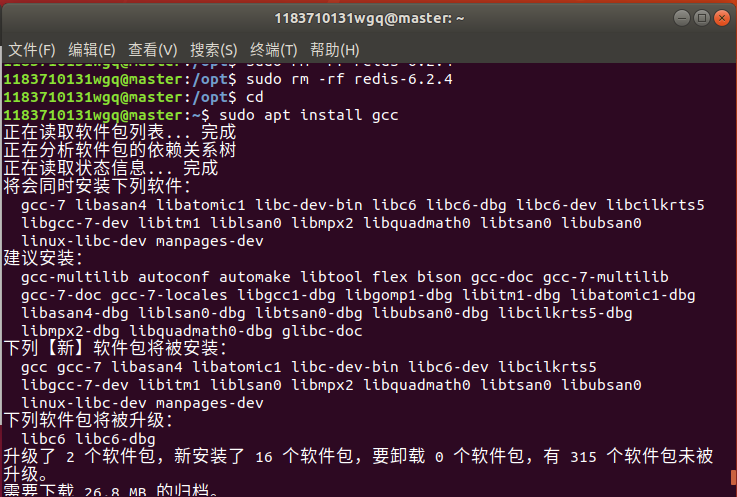


图3.3.4 安装gcc解决make执行报错问题

重新安装make后成功启动redis服务，如下图。

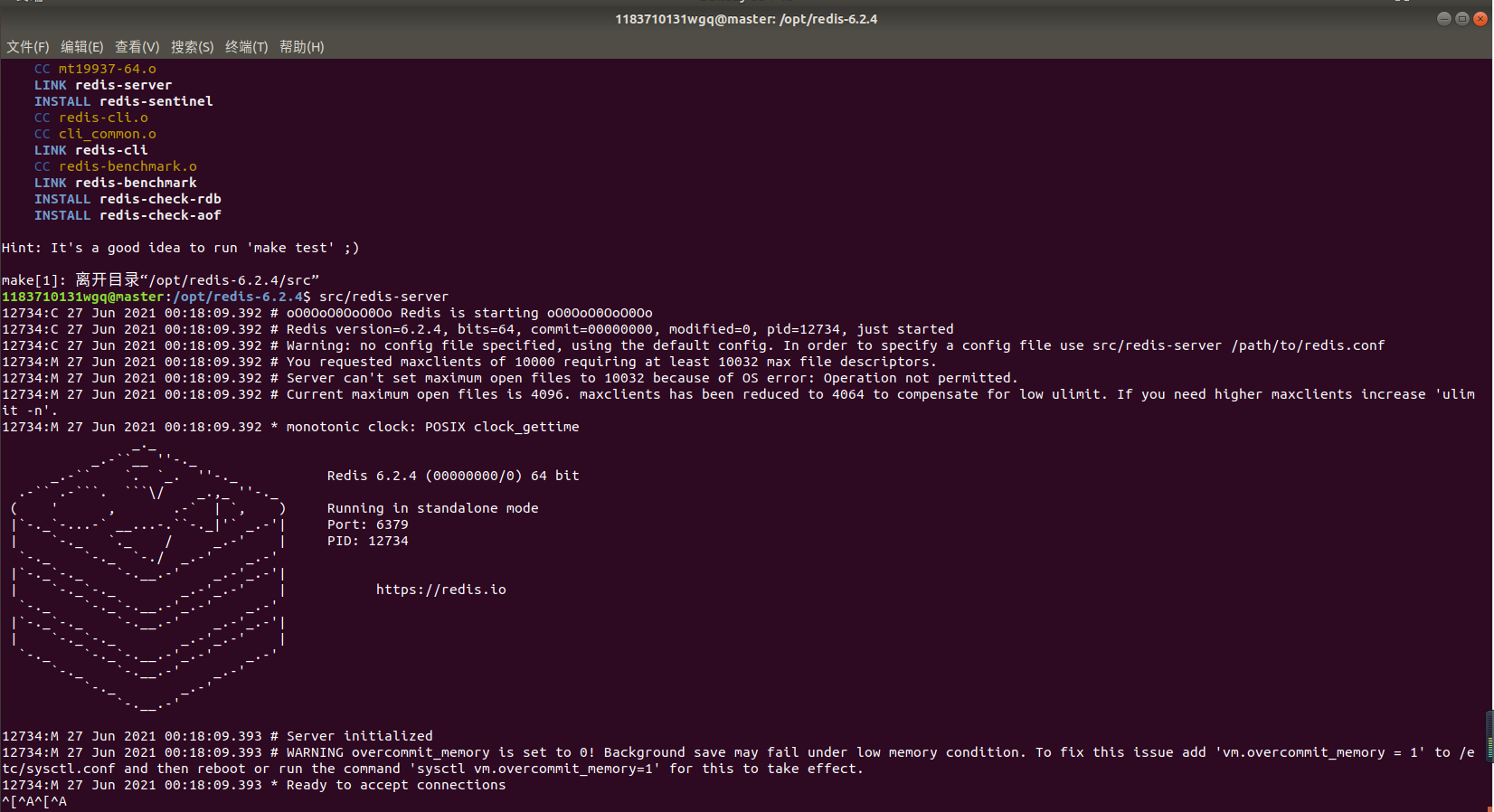
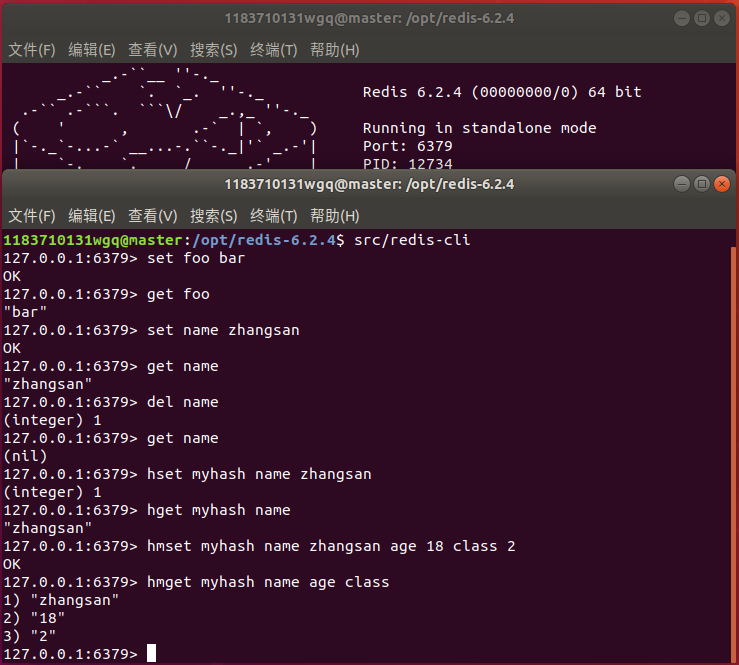


图3.3.5 redis-server成功启动

由于server启动后不能结束，只能挂到后台再进入redis-cli，故开启两个终端执行相关操作，如下图。



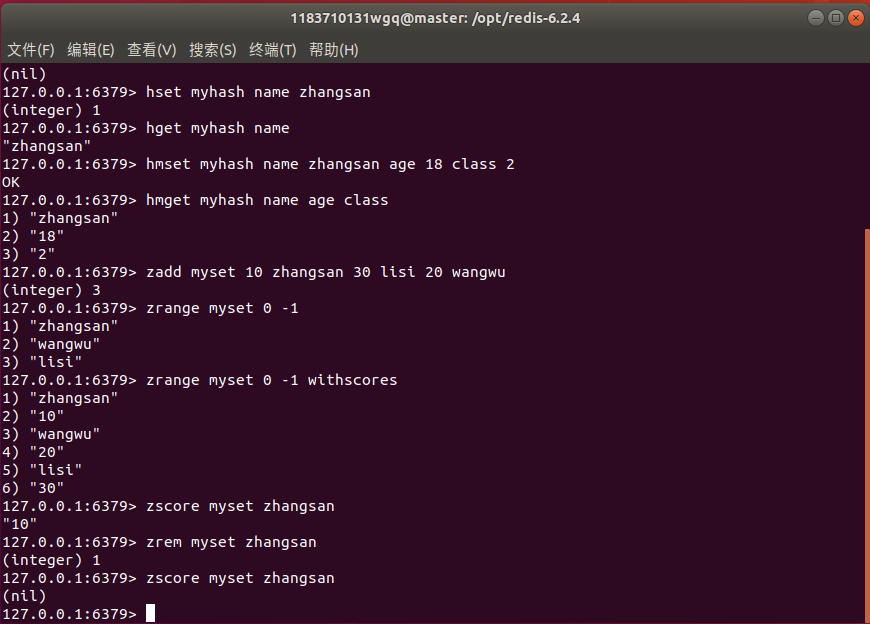


图3.3.6 redis相关指令

创建maven项目编写执行相关Java程序结果如下。

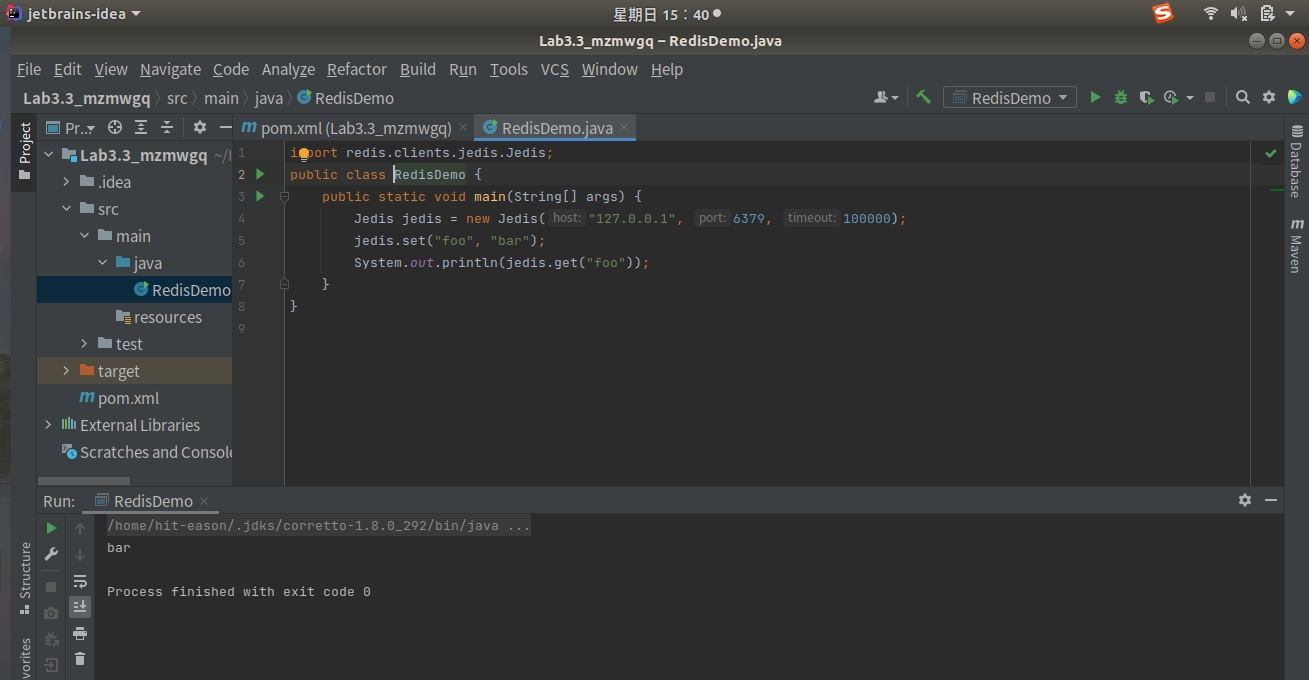


图3.3.7 Java项目引入redis执行结果

2）请详析Redis的缓存清洗策略，数据迁移及扩容策略，面向缓存雪崩、穿透等问题的策略。

**①缓存清洗策略**

关于缓存清洗，一般情况分为主动删除和被动删除，主动删除常见的策略是定时删除和定期删除，被动删除常见的策略是惰性删除。

**A.定时删除**：在设置数据过期时间时，会创建一个定时器，让定时器在数据过期时间来临时，立即执行对键的删除。

定时删除的优势是对内存能更快释放存储空间，但是对于CPU不够友好，尤其是当过期数据较多了，更占用大量CPU资源去进行清理工作。

**B.定期删除**：每隔一段时间，程序就会对数据库进行一次检查，删除里边的过期数据，对于删除多少数据，以及删除几个库的数据都由算法决定；

定期删除策略是每隔一段时间来执行删除动作，是通过限制执行的时长和频率来减少删除操作对CPU时间的影响。合理定期删除能有效降低过期数据对于存储空间的浪费。但是关于难点就是选定执行时间和周期。

**C.惰性删除**：暂时不管数据的过期时间，当获取数据时，再进行检查数据是否过期，如果已经过期就删除，未过期，返回数据。

由于惰性删除只会在获取数据时进行检查数据过期，进而判断是否需求删除，所以惰性删除相对于前两种删除策略是最对CPU最友好的。不过惰性删除的劣势也很明显，当过期数据不被获取，进不会进行删除，就会一直存在数据库中，占用内存，浪费空间。

**②数据迁移及扩容策略**

**A.数据迁移：**

1.迁移的目标服务器设置为被迁移服务器的slave服务器。

2.待slave追上master的进度后，停掉写redis业务（无法停机的话，需要考虑迁移期间的流水重做，或者在业务低峰期执行迁移，接受短暂的数据丢失）。

3.主从服务器完全同步后，修改写redis业务配置，指向slave服务器，并断开主从关系。

4.恢复写redis业务

**B.扩容策略：**

Redis扩容策略为字典扩容：

Redis的字典采用的是一种‘单线程渐进式rehash’，这里的单线程是指只有一个线程在扩容，而在扩容的同时其他的线程可以并发的进行读写。Redis系统后台会定时给予扩容的那个线程足够的运行时间，这样不会导致它饿死。

大致过程：

ht[0]，是存放数据的table，作为非扩容时容器。

ht[1]，只有正在进行扩容时才会使用，它也是存放数据的table，长度为ht[0]的两倍。

扩容时，单线程A负责把数据从ht[0] copy到ht[1] 中。如果这时有其他线程

进行读操作：会先去ht[0]中找，找不到再去ht[1]中找。

进行写操作：直接写在ht[1]中。

进行删除操作：与读类似。

**③缓存血崩、缓存穿透等问题策略**

**A.缓存雪崩**

缓存雪崩指的是大批量缓存在同一时间失效或者是缓存层支撑不住宕机，导致流量直接涌入数据库中，会造成数据库压力过大甚至挂掉。

(例如：我们设置缓存时采用了相同的过期时间，在同一时刻出现大面积的缓存过期)，所有原本应该访问缓存的请求都去查询数据库了，而对数据库CPU和内存造成巨大压力，严重的会造成数据库宕机。从而形成一系列连锁反应，造成整个系统崩溃。

解决办法：

大多数系统设计者考虑用加锁( 最多的解决方案)或者队列的方式保证来保证不会有大量的线程对数据库一次性进行读写，从而避免失效时大量的并发请求落到底层存储系统上。还有一个简单方案就时讲缓存失效时间分散开。

**B.缓存穿透**

缓存穿透是指查询一个根本不存在的数据， 缓存层和存储层都不会命中， 通常出于容错的考虑，如果从存储层查不到数据则不写入缓存层。缓存穿透将导致不存在的数据每次请求都要到存储层去查询， 失去了缓存保护后端存储的意义。

解决办法：

最常见的则是采用布隆过滤器，将所有可能存在的数据哈希到一个足够大的Bitmap中，一个一定不存在的数据会被这个Bitmap拦截掉，从而避免了对底层存储系统的查询压力。

另外，还有一个更为简单的方法，如果一个查询返回的数据为空(不管是数据不存在，还是系统故障)，我们仍然把这个空结果进行缓存，但它的过期时间会很短，最长不超过五分钟。通过这个直接设置的默认值存放到缓存，这样第二次到缓冲中获取就有值了，而不会继续访问数据库，这种办法最简单。

Bitmap和布隆过滤器(Bloom Filter)对于空间的利用到达了一种极致，下面进行适当解释。

**Bitmap**：

效率高,不需要进行比较和移位，并且占用内存少，典型的有哈希表。但是缺点是Bitmap对于每个元素只能记录1bit信息，如果还想完成额外的功能，恐怕只能靠牺牲更多的空间、时间来完成了。

**布隆过滤器**:

Bloom-Filter一般用于在大数据量的集合中判定某元素是否存在。

它相当于引入了k(k>1)k(k>1)个相互独立的哈希函数，保证在给定的空间、误判率下，完成元素判重的过程。

它的优点是空间效率和查询时间都远远超过一般的算法，缺点是有一定的误识别率和删除困难。Bloom-Filter算法的核心思想就是利用多个不同的Hash函数来解决“冲突”。Hash存在一个冲突(碰撞)的问题，用同一个Hash得到的两个URL的值有可能相同。为了减少冲突，我们可以多引入几个Hash，如果通过其中的一个Hash值我们得出某元素不在集合中，那么该元素肯定不在集合中。只有在所有的Hash函数告诉我们该元素在集合中时，才能确定该元素存在于集合中。这便是Bloom-Filter的基本思想。

**C.缓存击穿**

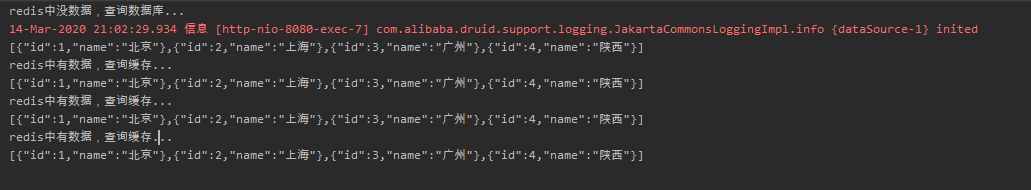
缓存击穿：缓存击穿是指一个Key非常热点，在不停的扛着大并发，大并发集中对这一个点进行访问，当这个Key在失效的瞬间，持续的大并发就穿破缓存，直接请求数据库，就像在一个完好无损的桶上凿开了一个洞。

解决方案：在访问key之前，采用SETNX(set if not exists)来设置另一个短期key来锁住当前key的访问，访问结束再删除该短期key。

结合一个处理的案例：背景双机拿token，token在存一份到redis，保证系统在token过期时都只有一个线程去获取token;线上环境有两台机器，故使用分布式锁实现。

1. 请在进销存系统(或其他实际软件系统)设计一个简单场景，实现缓存读写操作，缓存更新操作，给出缓存的效果，分析2问题中相关策略的效果。

redis缓存供货商执行结果：



具体核心代码：



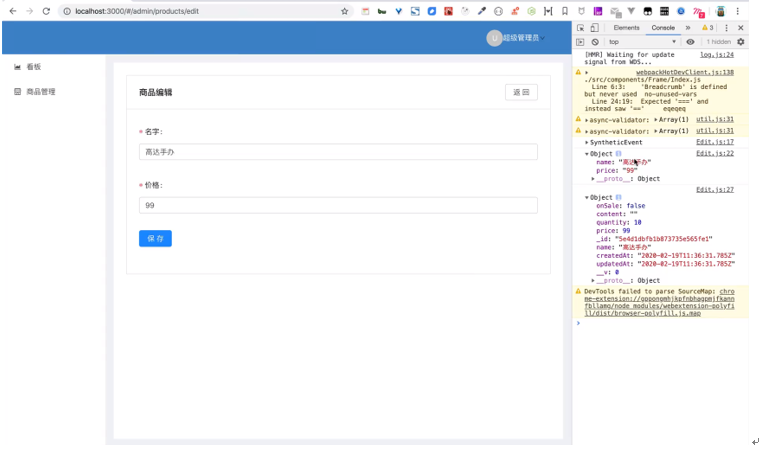
## 利用React进行表示层的设计及实现

1. 利用React开发2-3个进销存系统(或其他实际软件系统)的交互界面。

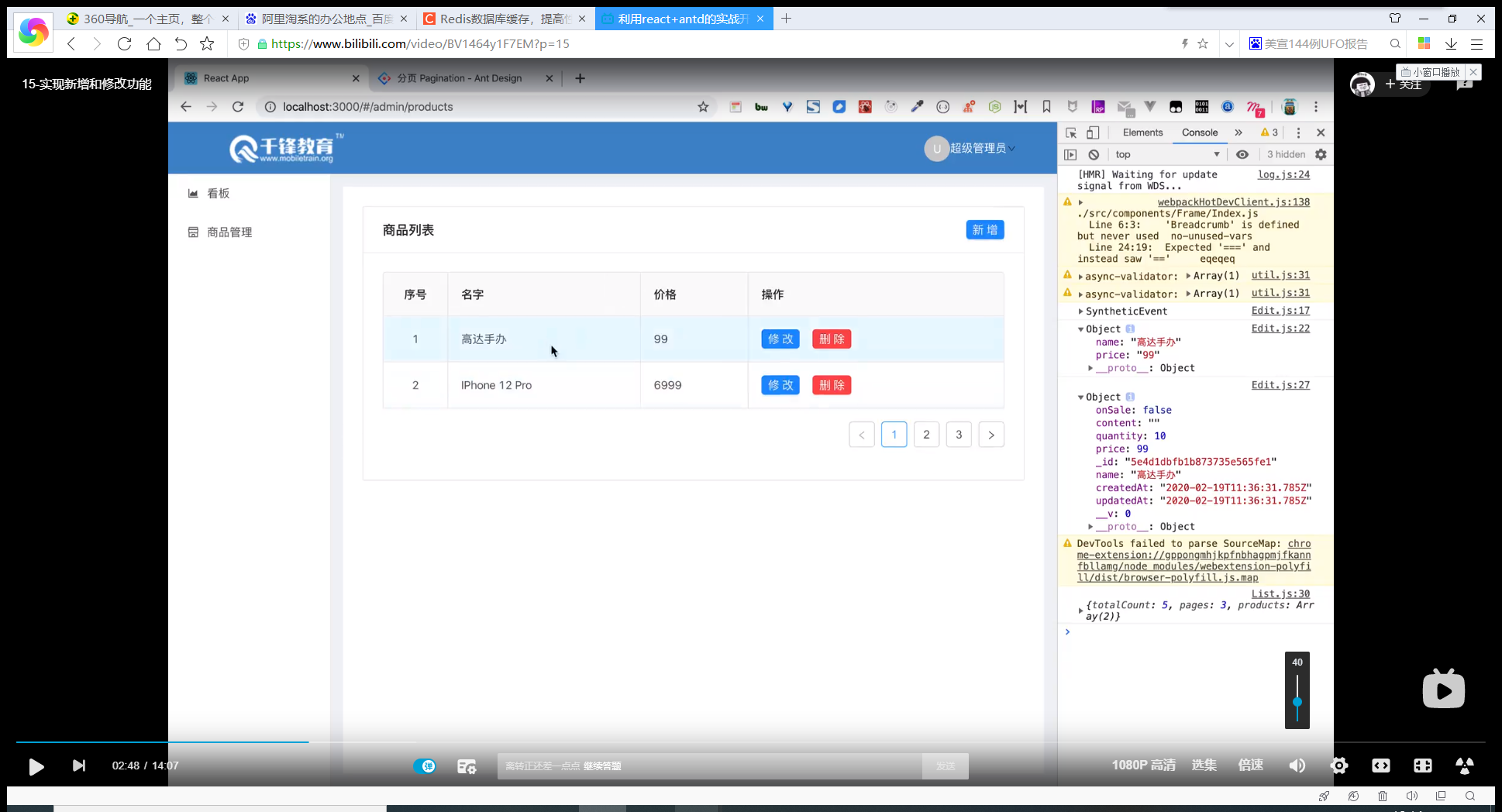
系统通知界面



商品编辑交互页面如下：



查看商品界面



# 结对开发过程记录

**（1）角色切换与任务分工**

表1-1结对开发角色与任务分工

| 日期 | 时间(HH:MM - HH:MM) | 驾驶员角色 | 领航员角色 | 本段时间的任务 |
| --- | --- | --- | --- | --- |
| 5.22 | 14:00 – 18:00 | 梅智敏 | 王国庆 | 开始实验3.1 |
| 5.23 | 14:00 – 19:30 | 王国庆 | 梅智敏 | 完成实验3.1的剩余部分 |
| 5.26 | 20:00 – 23:00 | 梅智敏 | 王国庆 | 开始实验3.2 |
| 5.27 | 10:00 – 12:30 | 王国庆 | 梅智敏 | 完成实验3.2剩余部分 |
| 6.5 | 15:00 – 22:00 | 王国庆 | 梅智敏 | 完成实验3.3 |
| 6.9 | 19:00 – 22:30 | 梅智敏 | 王国庆 | 开始实验4 |
| 6.15 | 10:00 – 12:30 | 梅智敏 | 王国庆 | 完成实验4 |
| 6.20 | 19:00 – 22:30 | 王国庆 | 梅智敏 | 完成实验报告，并截图 |

**（2）工作日志**

由领航员负责记录，记录结对开发期间的遇到的问题、两人如何通过交流合作解决每个问题的。

表1-2 结对开发工作日志

| 日期/时间 | 问题描述 | 最终解决方法 | 交流过程 |
| --- | --- | --- | --- |
| 5.22/14:00 – 18:00 | 在执行mysql -uroot -p123456 -h 127.0.0.1 -P8066 -DTESTDB启动之后，执行数据库插入出现报错 | 注释掉默写代码且完成相关文件的配置 | 两人上网查询原因，再进行修改 |
| 5.26/20:00 – 23:00 | 按指导书配置依赖后执行程序报错 | 引入依赖后，启动还是报错，原因是缺少<scope>compile</scope>，再添加后才能正确添加编译环境下的依赖。 | 根据报错信息上网查询，再进行修改 |
| 6.5/15:00 – 22:00 | 执行make 命令 需要安装相关的工具包 | 进行相应工具包的安装 | 根据相关提示进行安装 |
| 6.5/15:00 – 22:00 | 无法在同一个终端下启动redis服务后继续执行Redis命令行 | 开启server后，开辟另一个终端redis-cli即可 | 上网查询相关案例，进行改正 |

【注意】该表格可自行增加更多的行

**（3）结对开发工作现场照片或线上讨论截图**

至少2张。

结对开发现场照片1 结对开发现场照片2

# 实验总结

通过本次实验我们学会了使用Mycat和Sharding-JDBC实现数据分库分表，在其期间遇到了一些问题，在经过讨论和搜索相关资料后解决，在这个过程中我们对分库分表有了进一步的认识，对其性能上的提升有了更深的了解。 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。当 Mycat收到一个SQL时，会先解析这个SQL，查找涉及到的表，然后看此表的定义，如果有分片规则， 则获取到SQL里分片字段的值，并匹配分片函数，得到该SQL对应的分片列表，然后将SQL发往这些分片去执行，最后收集和处理所有分片返回的结果数据，并输出到客户端。一张表经过分库分表后被拆分成多个子表，并分散到不同的数据库中，在不修改原业务 SQL 的前提下，Sharding-JDBC 会对 SQL进行一些改造才能正常执行。大致的执行流程：SQL 解析 -> 执⾏器优化 -> SQL 路由 -> SQL 改写 -> SQL 执⾏ -> 结果归并

我们学习使用Redis数据库实现数据缓存，了解了Redis的缓存清洗策略，数据迁移及扩容策略，缓存雪崩、穿透等问题的策略的相关知识，对数据库中间件有了进一步的认识。

最后针对表示层，我们也学习掌握了react的初步知识，并与我们在软件过程与工具中的进销货系统进行结合，创建了一些新的前端界面，通过本次学习对数据表示层的相关框架知识有了更深的理解。