哈尔滨工业大学

**计算机科学与技术学院/国家示范性软件学院**

**2020年春季学期**

**《软件架构与中间件》课程**

**实验报告**

**Lab 3：数据层软件架构实验**

|  |  |  |
| --- | --- | --- |
| **姓名** | **学号** | **联系方式** |
| 任永鹏 | 1173710107 | 805402160@qq.com |
| 黄项宇轩 | 1173710106 | 1171845717@qq.com |

目 录

1 实验目的 1

2 实验要求 1

3 实验内容与过程 1

3.1 Mycat数据库分库分表实验 1

3.2 Sharding-JDBC数据库分库分表实验 1

3.3 Redis数据缓存实验 2

4 结对开发过程记录 2

5 实验总结 3

[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”]

1. **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

1. **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

1. **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。

3.1Mycat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。**

1.问题：下载地址<http://dl.mycat.io/>无法打开

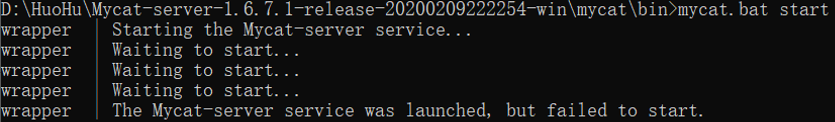
解决：从<http://www.mycat.org.cn/>处下载

2.问题：解压后mycat无法启动

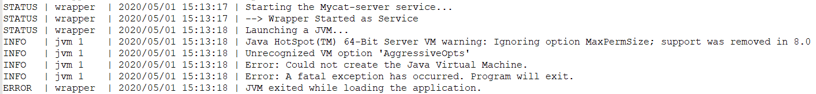


解决：mycat未安装，安装mycat.bat install

3.问题：mycat启动出错



解决：查看日志



根据日志信息作以下四处修改

修改wrapper.conf

添加java路径，原理为java



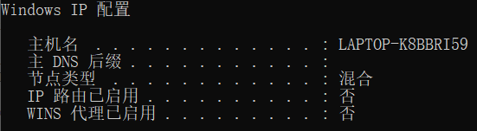
将AggressiveOpts注释



修改JVM内存，原来为4G，1G



修改主机名，原来为localhost



**2）请详析Mycat的分库分表原理和操作方法。**

分库分表原理：

应用Mycat的系统不直接与物理数据库连接，配置连接时系统使用相应的用户登录Mycat端口，在Mycat的配置中为这个用户配置了相应的逻辑数据库，这个数据库不实际存在。逻辑数据库配置了数据库内每张表对应的数据节点或者称数据分片以及分片的策略。数据节点则对应了原本系统使用的物理数据库的连接。可以理解为系统向Mycat中的逻辑数据库写数据，Mycat根据切分的规则将数据分库分表写入不同数据节点的物理数据库中。以上是个人理解，实际上，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。应用程序不再直接访问数据库，而是访问Mycat，由Mycat与数据库交互，数据库数据返回给Mycat，Mycat再返回给应用程序。三个Database才是真正的数据库，又称为三个节点，也称为三个分片。Mycat作为一个中间件，应用程序直接访问它，不用再去管真实的数据库，而由Mycat来与真实的数据库进行交互，真实的数据库可能有多个，这就是分布式架构，即多节点（多分片）。

Database2

Database3

应用程序

Mycat中间件

Database1

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。

当应用只需要一台数据库服务器的时候并不需要Mycat，而如果需要分库分表，应用要面对很多个数据库的时候，就需要对数据库层做一个抽象，来管理这些数据库，而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。

操作方法：

1.基本开发环境准备：

MySQL和jdk已安装，下载Mycat并解压



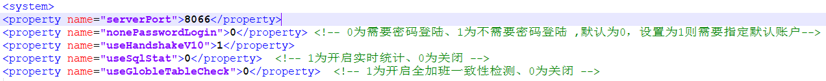
安装



2.Mycat配置

Mycat系统参数配置/conf/server.xml

这个文件可以不做修改

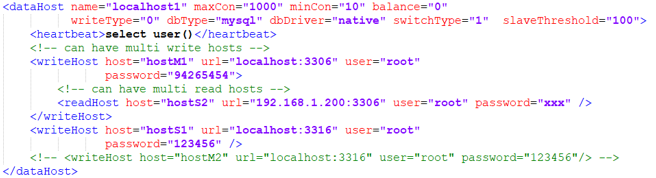


逻辑库、表分片配置/conf/schema.xml

在逻辑数据库添加测试用的两张表users和item

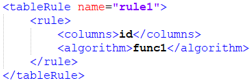


根据本地物理数据库修改



Mycat表切分规则配置/conf/rule.xml

这里一定要修改切分规则



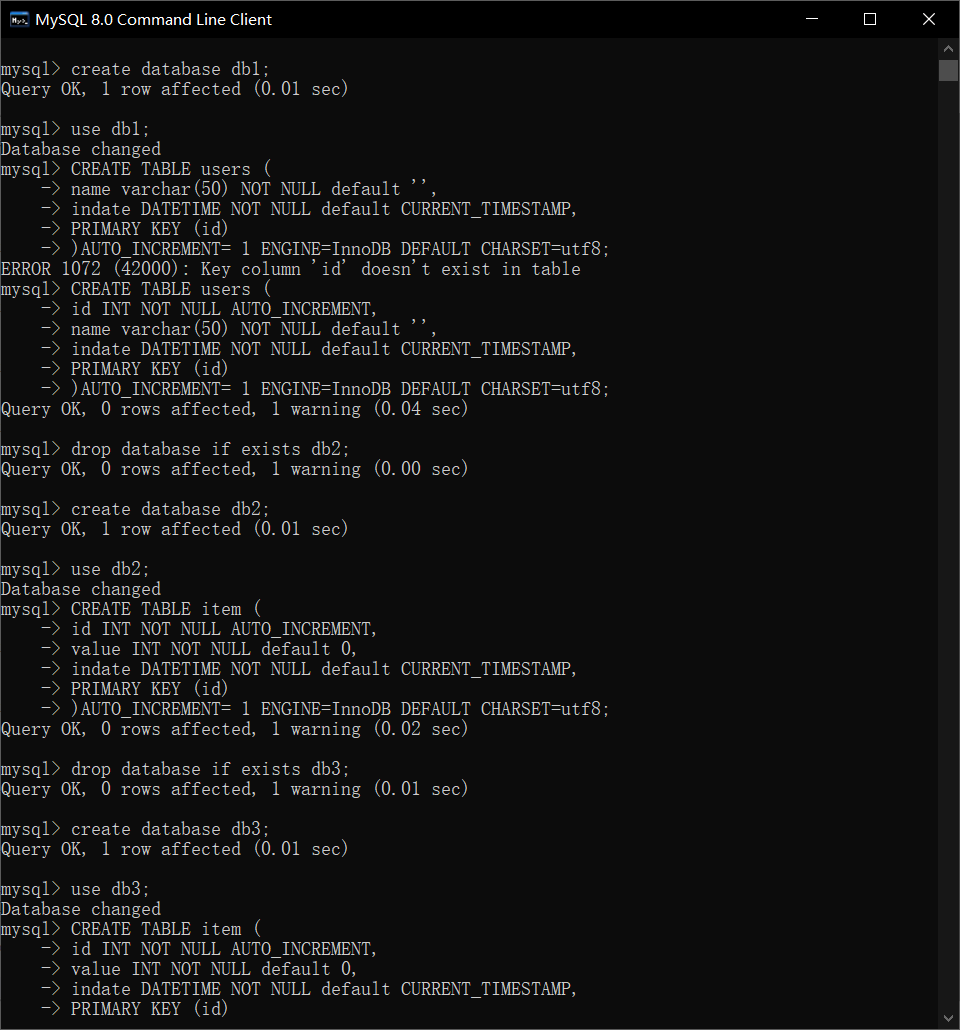


否则会出现如下错误

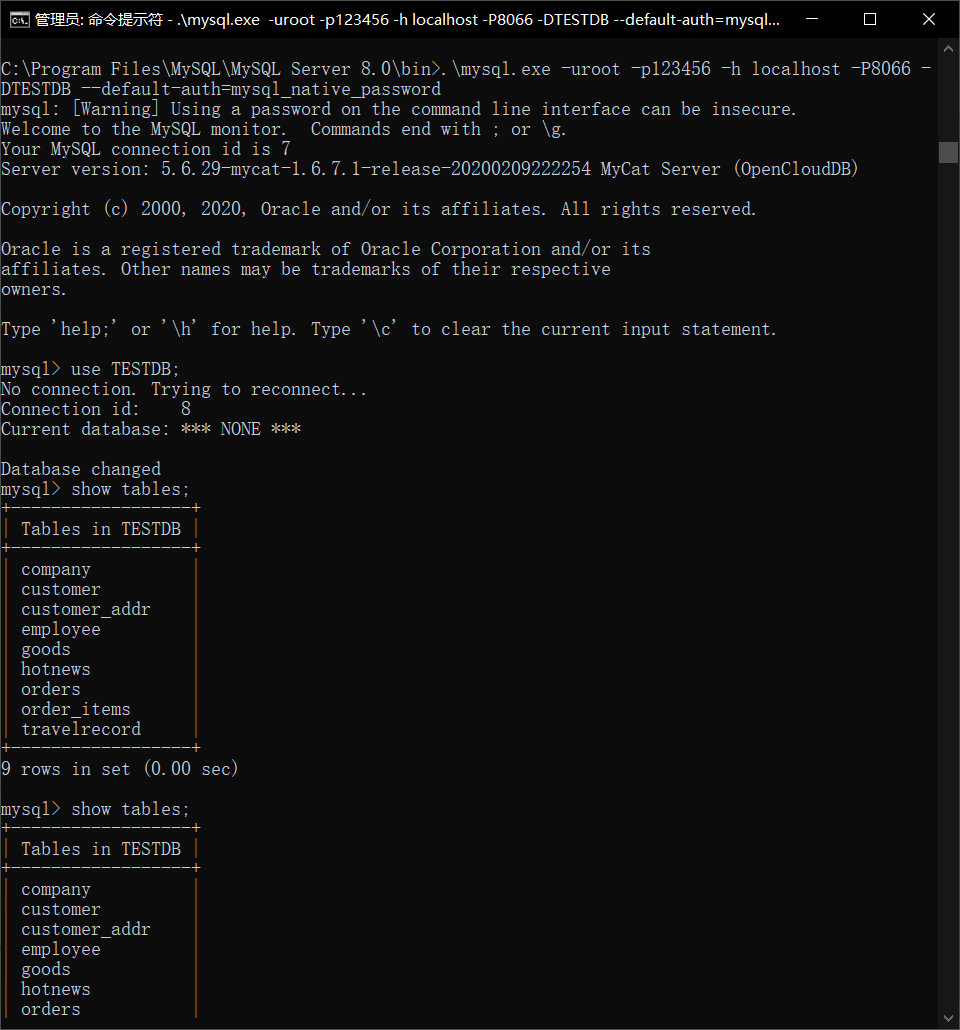


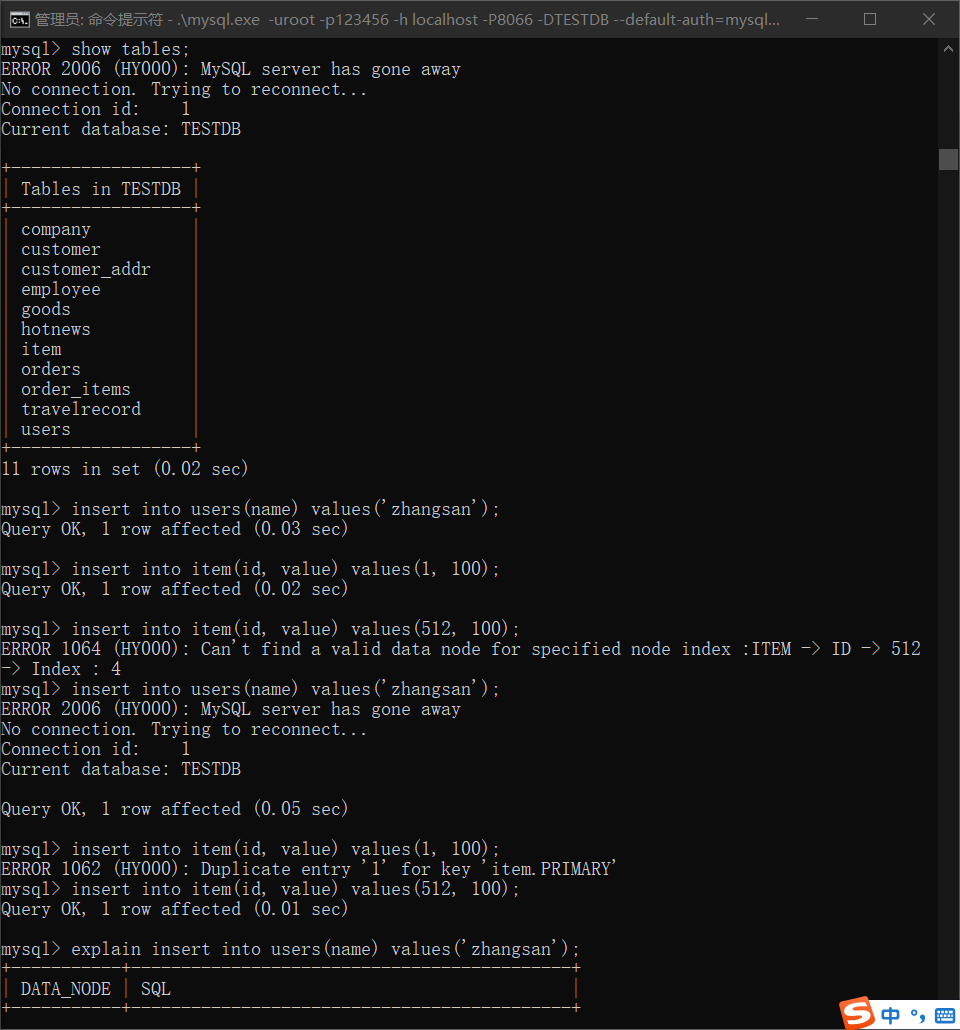
日期范围HASH分片，从sBeginDate时间开始计算，每sPartionDay天的数据为一个分片组，每个分片组可以分布在groupPartionSize个分片上面。上面的例子最多可以有三天进行分片，如果超出则会抛出如上异常。

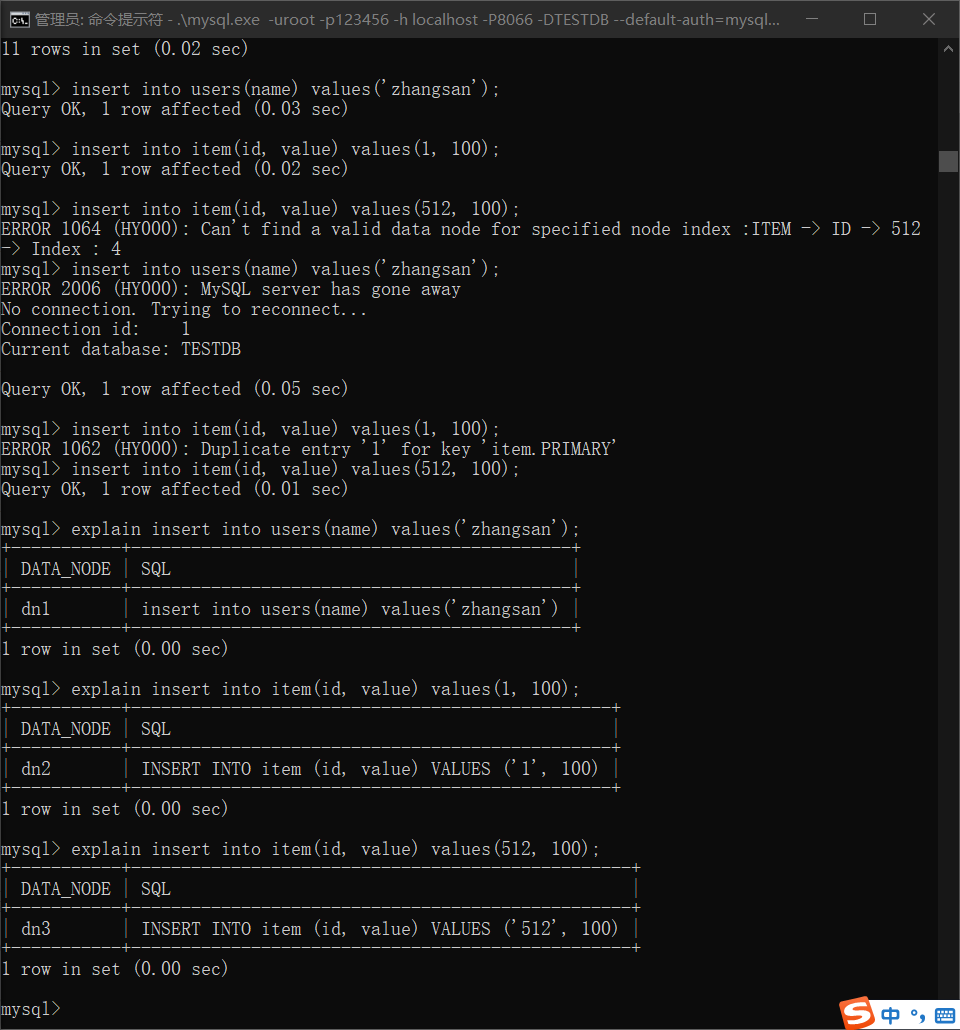
3.物理数据库配置



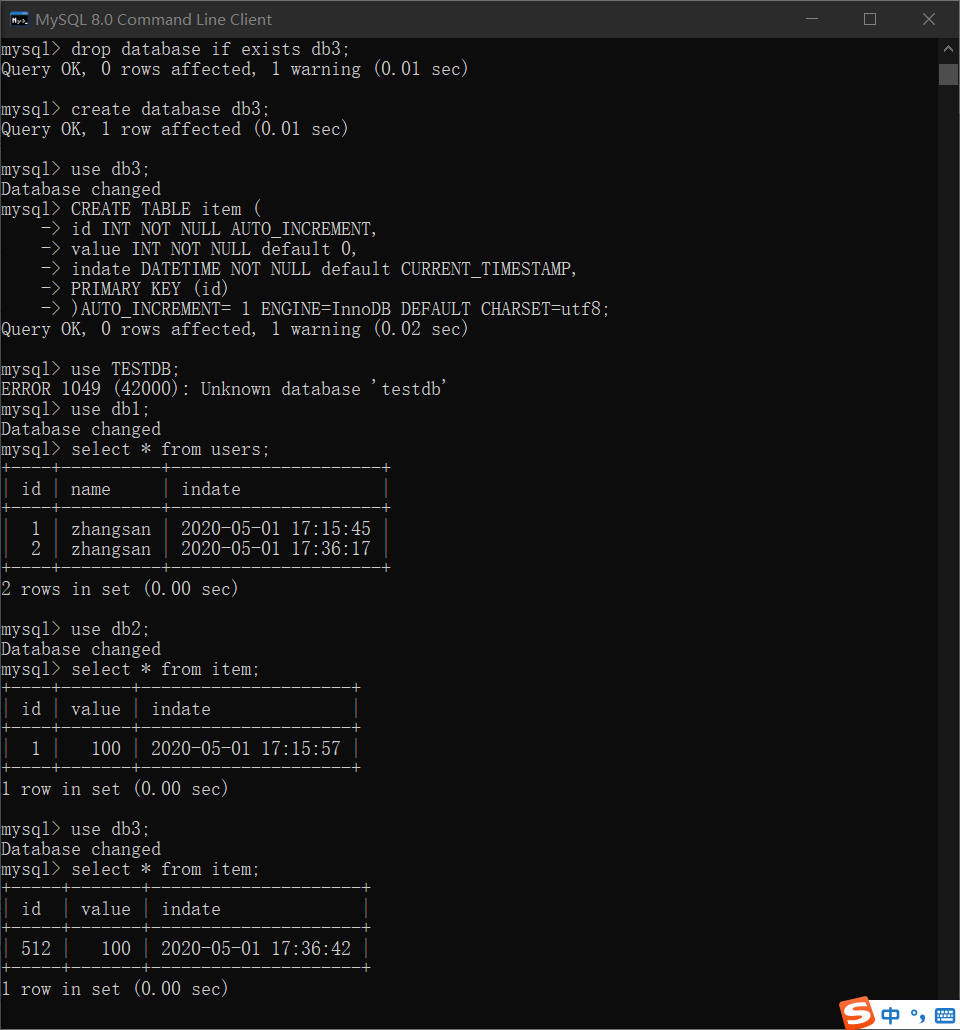
4.启动Mycat服务，测试插入数据





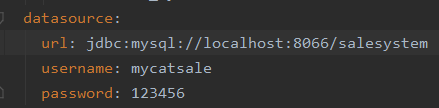


登录物理数据库查看，插入users表中的数据全部在db1中，item表中的数据分布在db2和db3中，根据实际的路由策略进行了分表。

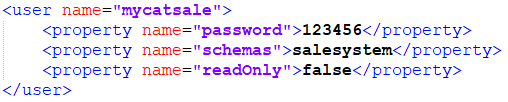


**3）请在进销存系统(或其他实际软件系统)创建具有复杂表结构和含有较大数据量的数据库表， 并基于此库表描述分库分表的结果，且验证分库分表的效果。**

修改原系统中数据库连接配置：



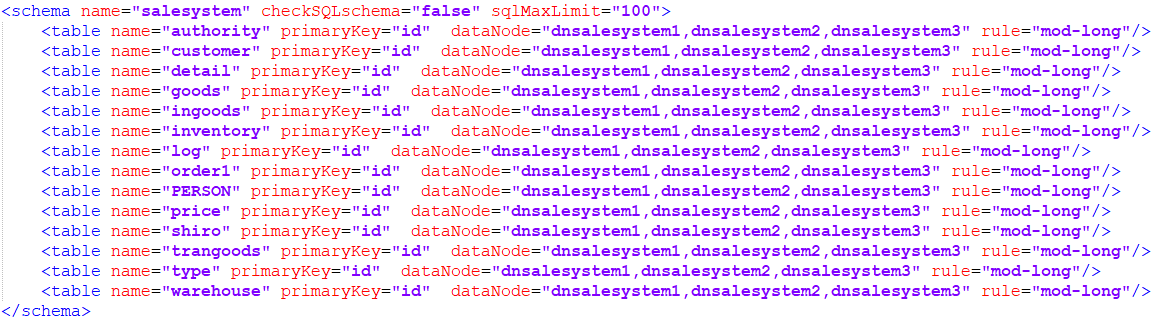
在server.xml中为系统创建用户：



端口号保持8066：



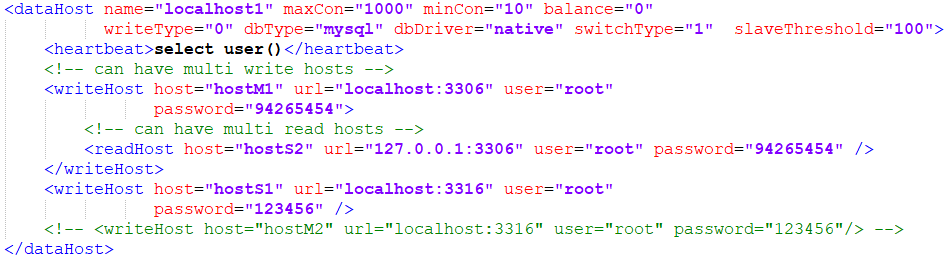
在schema.xml中为用户配置逻辑数据库（经测试表名不区分大小写）：



配置数据节点：

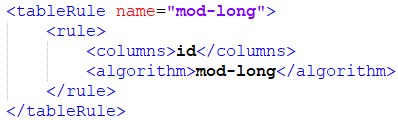


配置实际数据库连接信息：

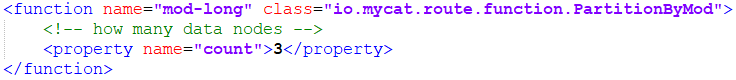


读取不能用localhost，一定要127.0.0.1

在rule.xml中配置分片规则：

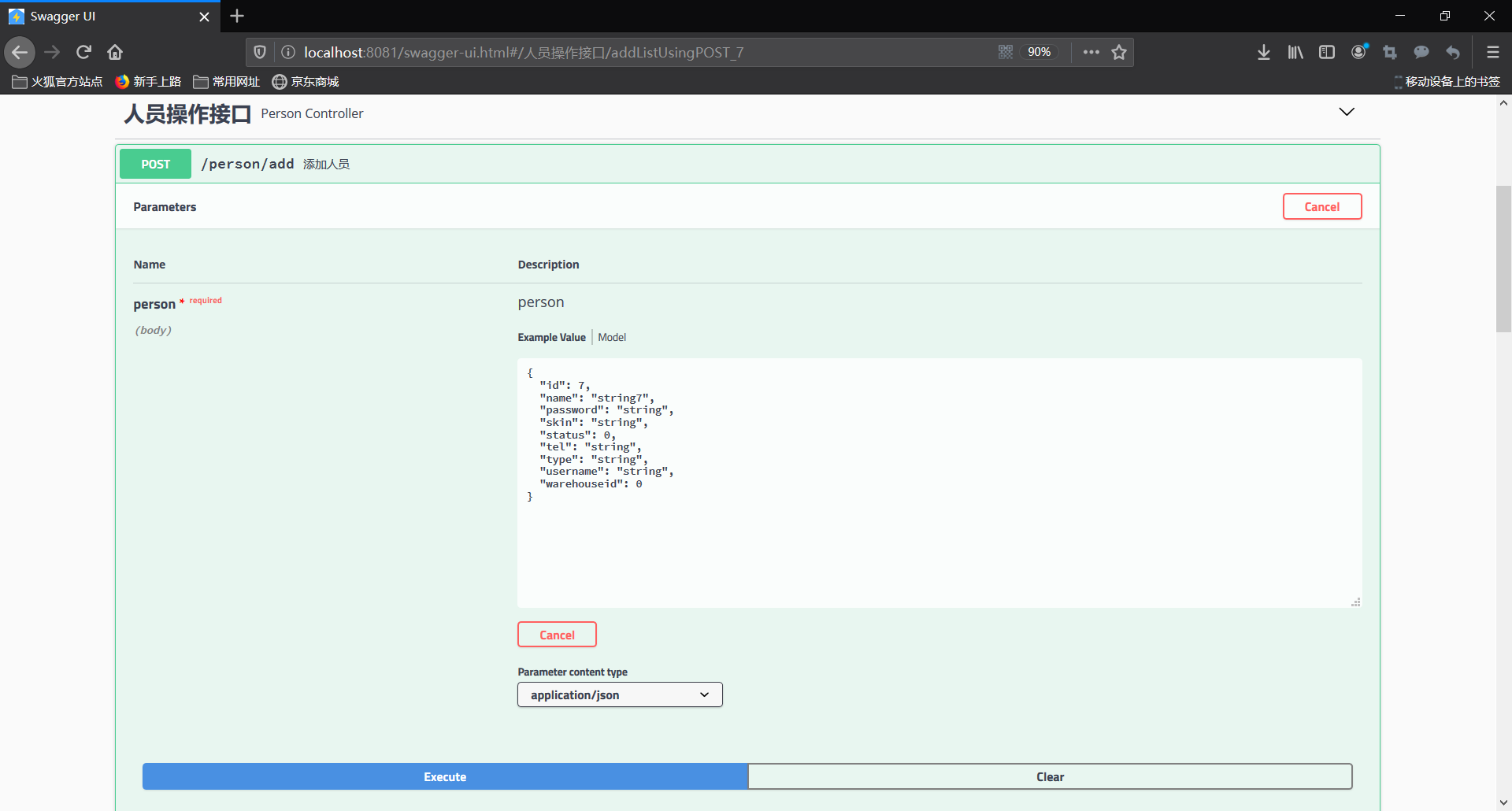


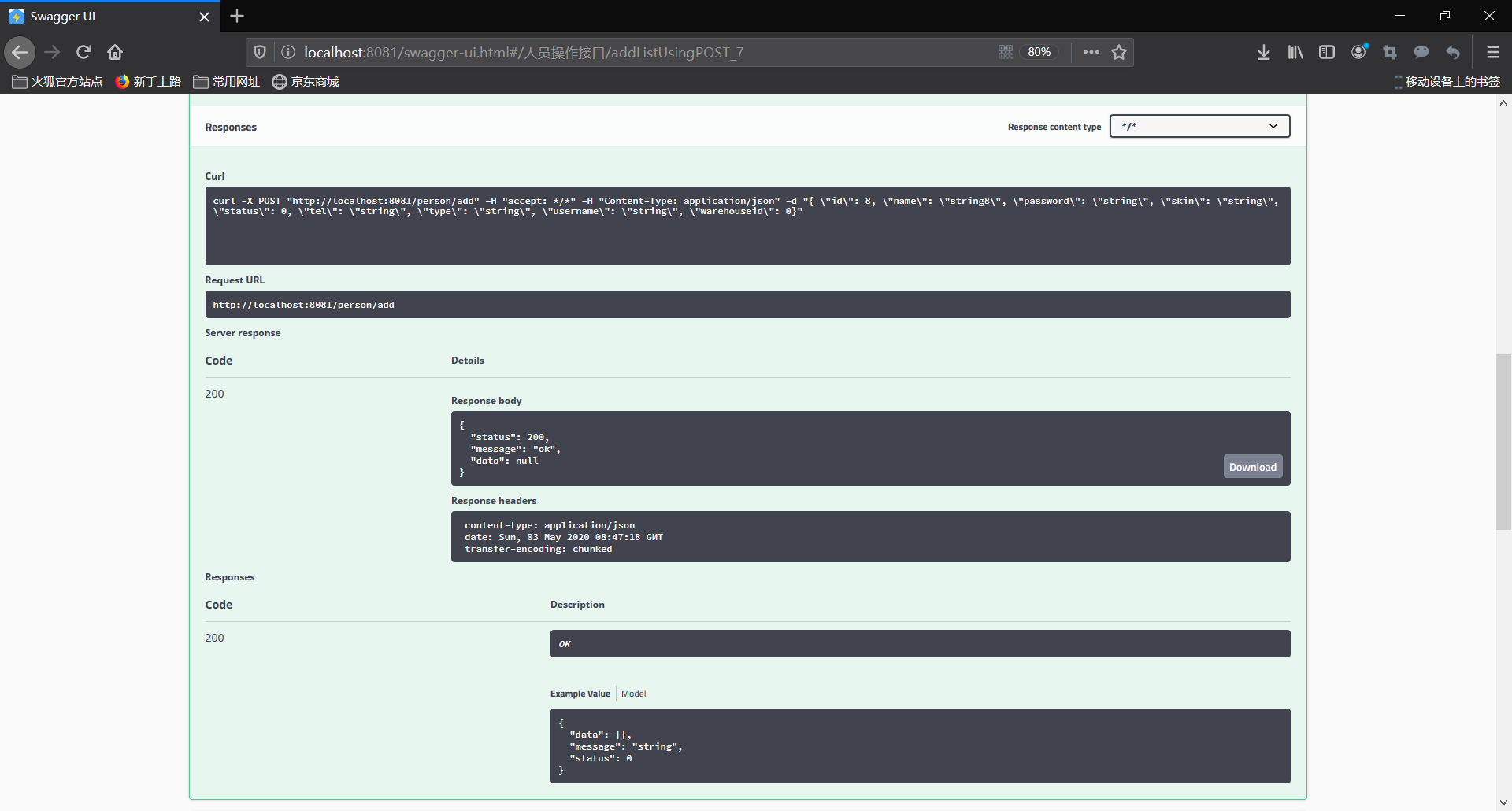
使用mod-long将数据分为三份以缓解单一数据库压力：

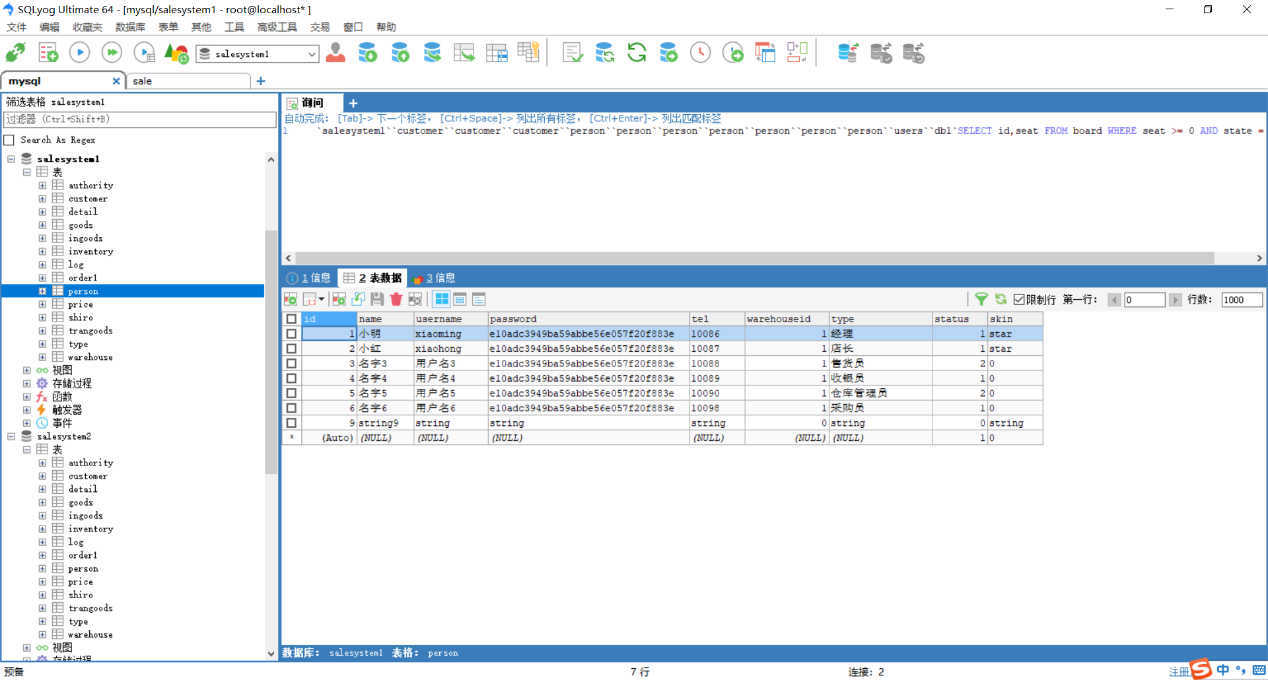


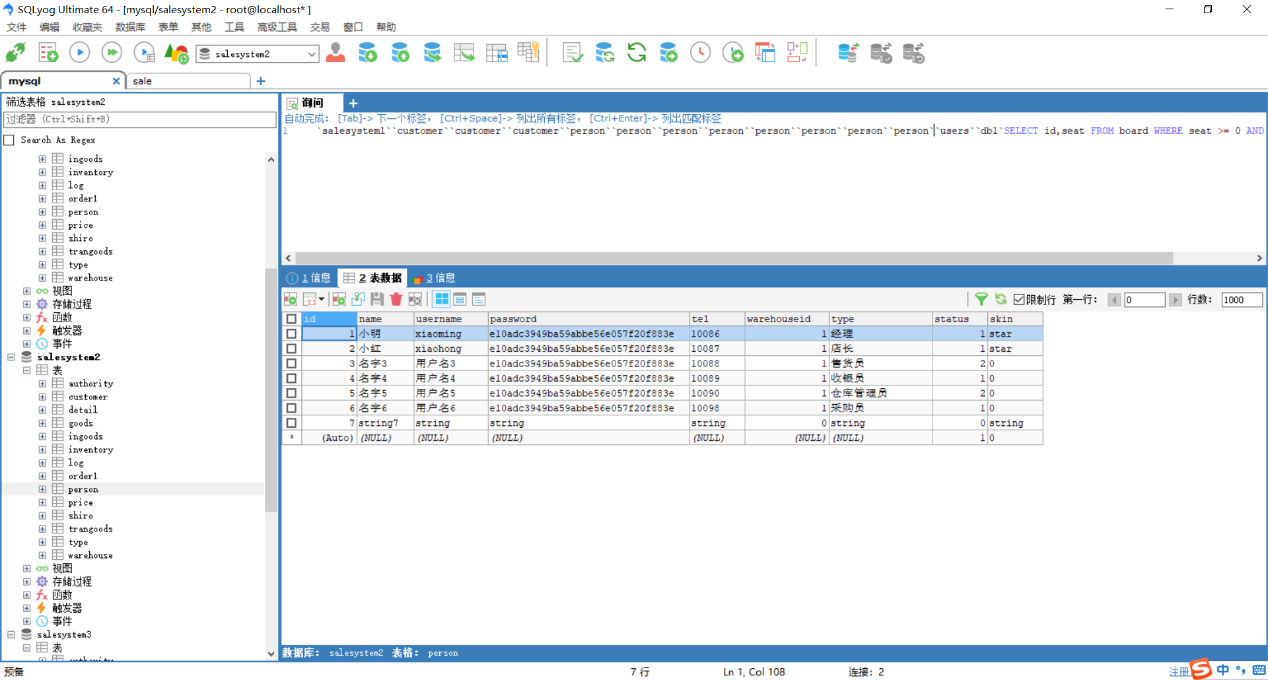
测试：

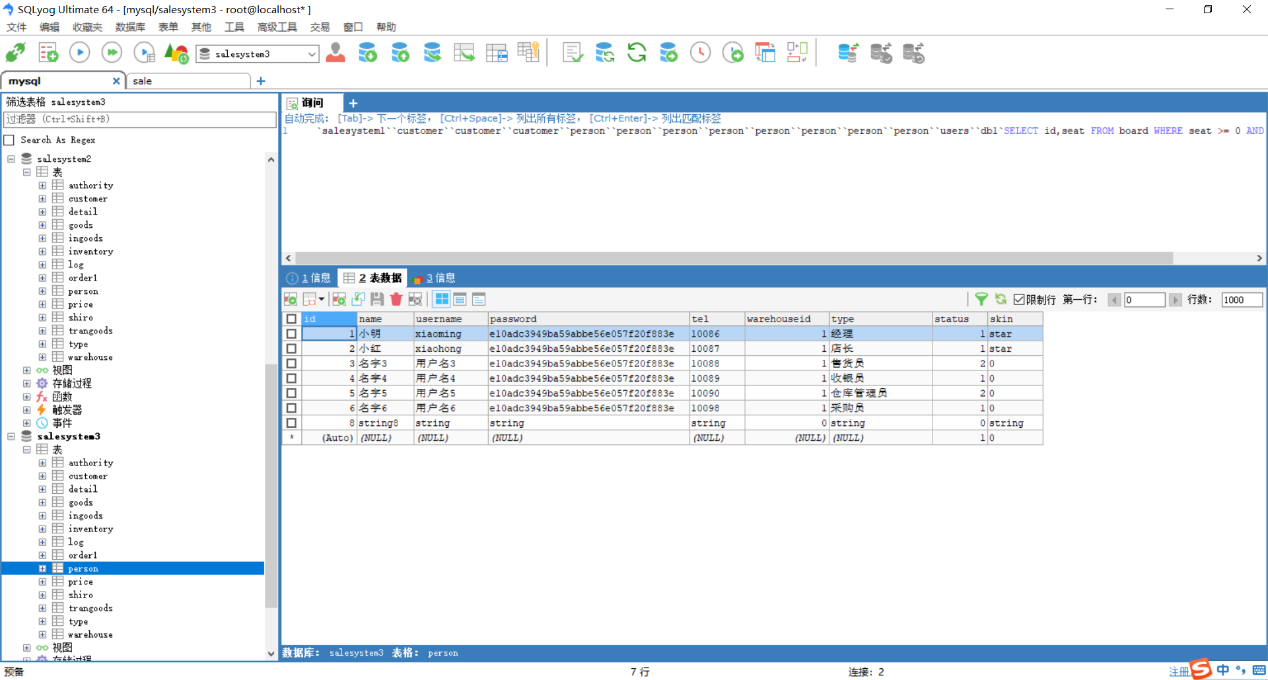
数据被平均写入三个数据库，数据分库成功

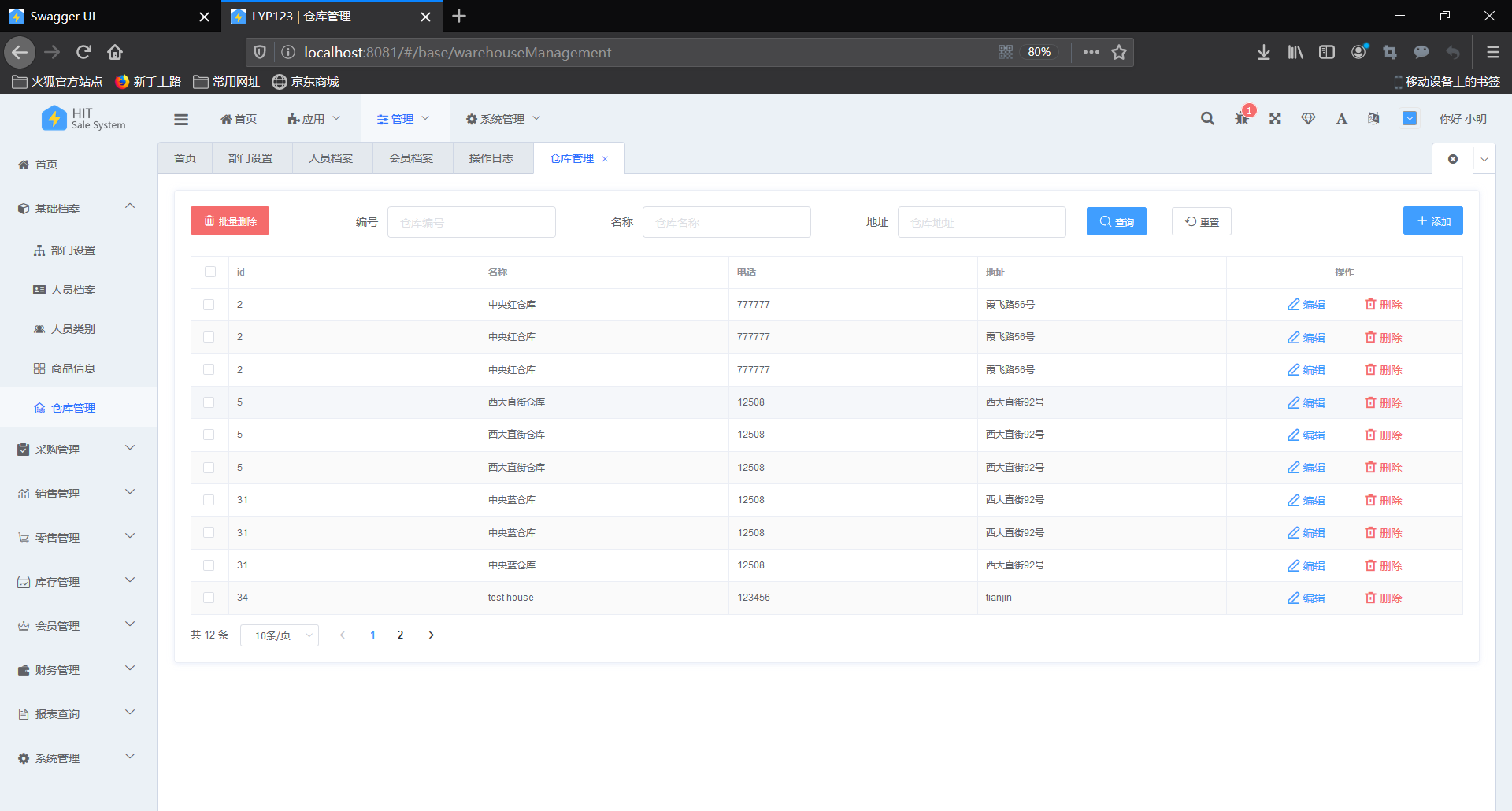








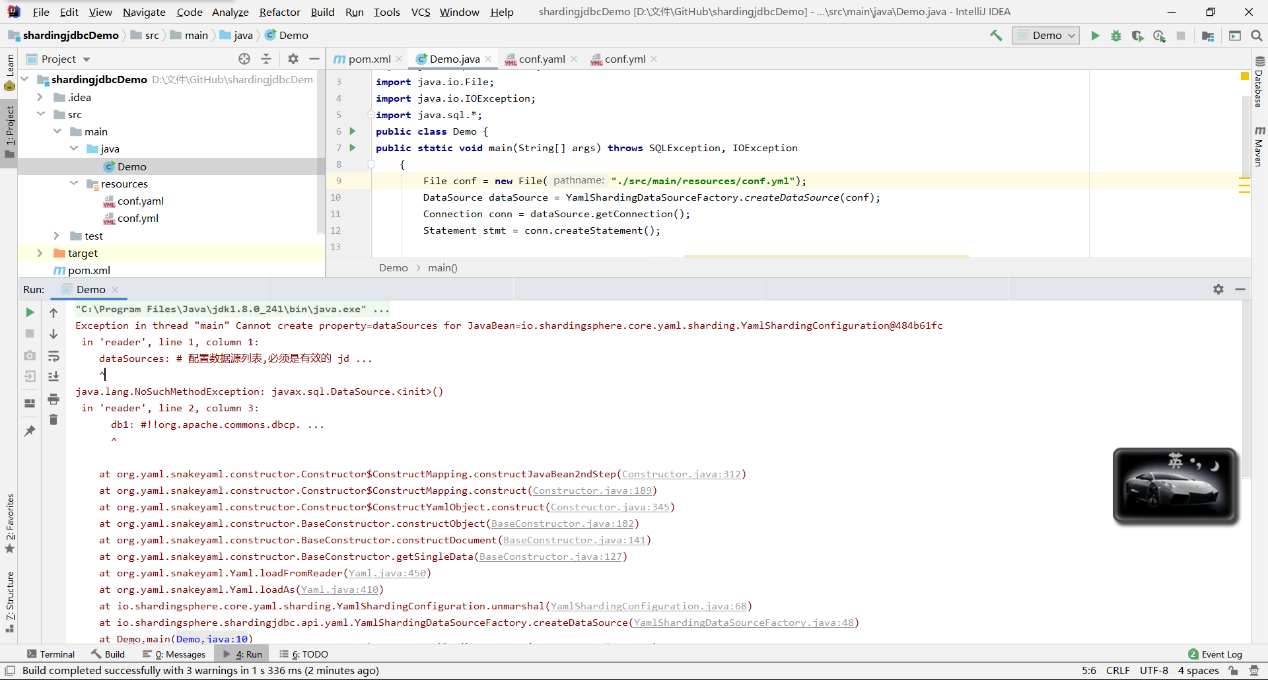




3.2Sharding-JDBC数据库分库分表实验

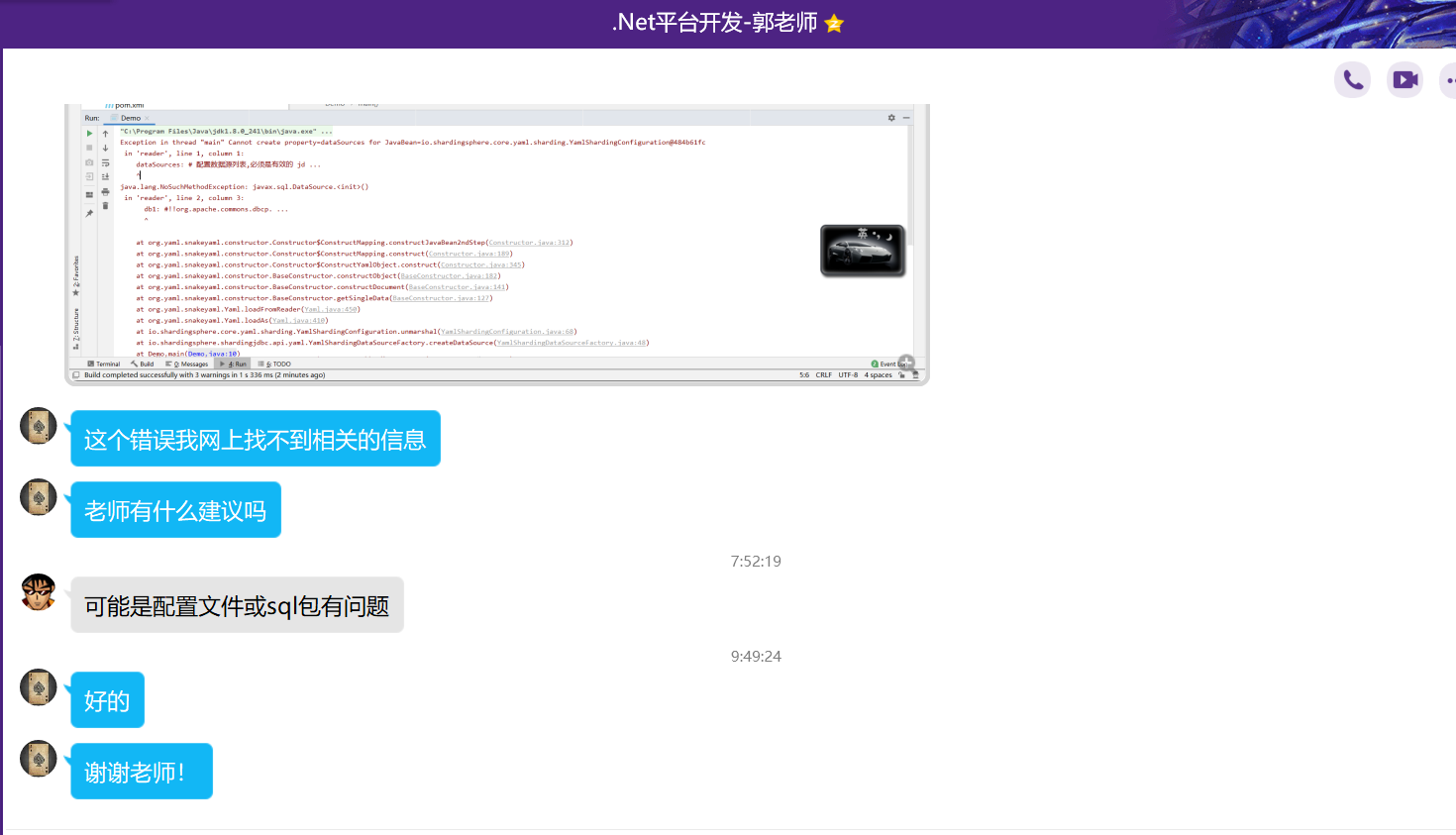
**1）请给出Sharding-JDBC配置安装过程中遇到的问题和解决方案。**

问题一：出现问题nosuchMethodException

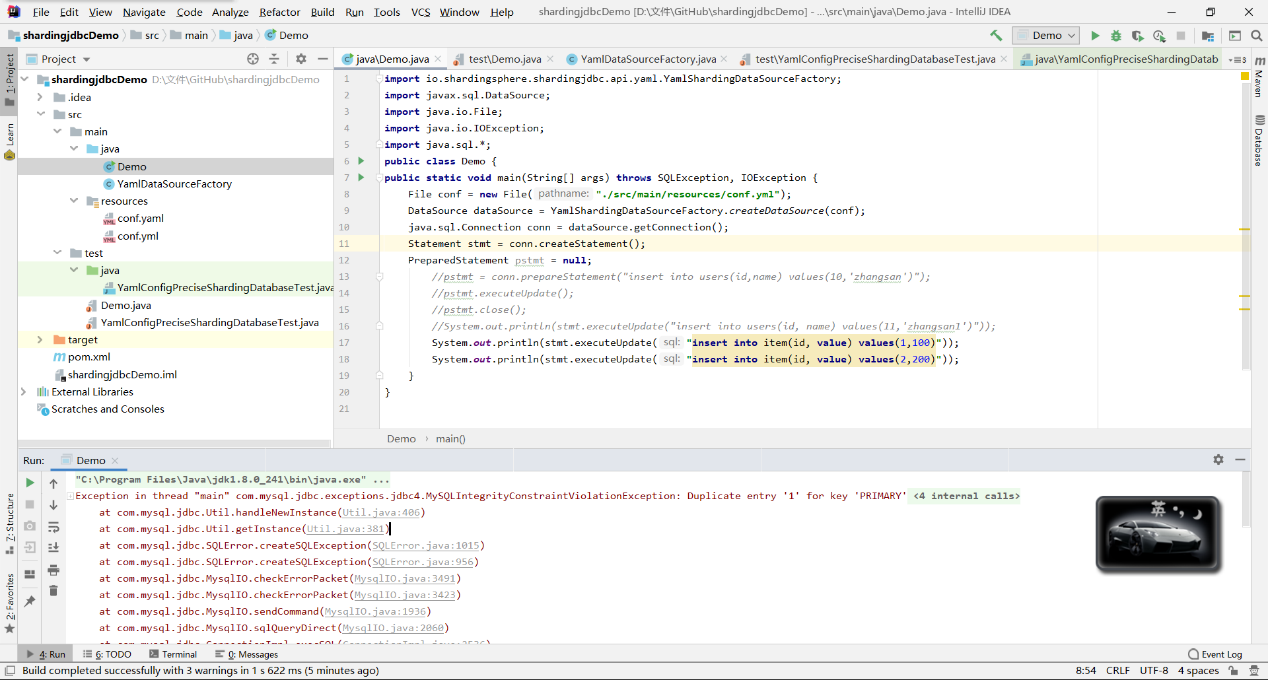


解决办法：

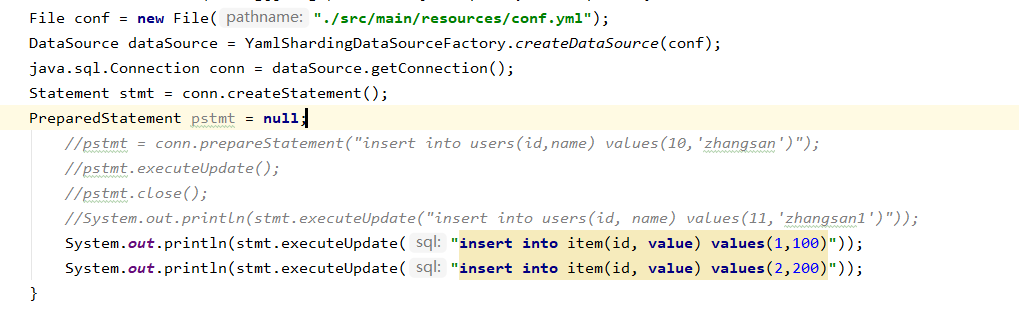
询问老师，修改yml配置文件



问题二：conn.prepareStatement返回空指针 以及 Duplicate entry '1' for key 'PRIMARY



解决办法：在指导书中的demo.java文件，因为数据库表id为自增，sql语句不能插入id，取消id主键，同时添加新的字段做为主键，还有就是取消库表的promary索引，这样就可以通过yml配置文件的id来区分进行分库分表操作



问题三：Unknown column 'string' in 'field list'

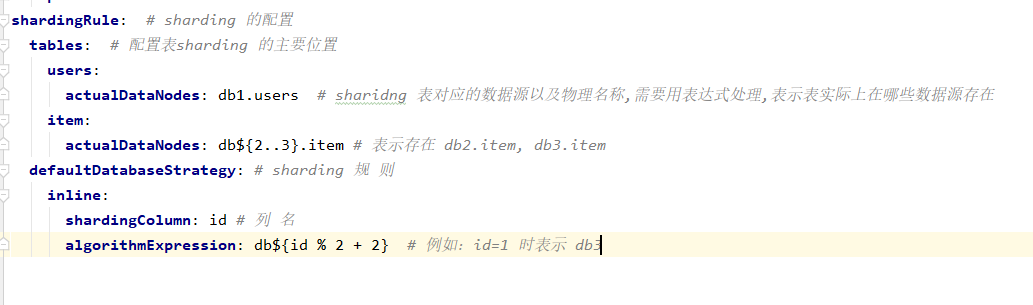


解决办法：string字段需要使用双引号里面嵌套单引号，从而保证string类型

**2）请详析Sharding-JDBC的分库分表原理和操作方法。**

分库分表原理：

如下图，在tables下声明所有的库表，以及声明对应的库，在defaultdatabasestrategy里面进行定义分库规则。



操作方法：

Pom.xml添加依赖

<**dependency**>  
 <**groupId**>io.shardingsphere</**groupId**>  
 <**artifactId**>sharding-jdbc-core</**artifactId**>  
 <**version**>3.1.0</**version**>  
</**dependency**>  
<**dependency**>  
 <**groupId**>org.slf4j</**groupId**>  
 <**artifactId**>slf4j-nop</**artifactId**>  
 <**version**>1.7.2</**version**>  
</**dependency**>

### 使用yml配置文件，配置配置DataSource：

### 

### 使用insert语句进行分库分表插入操作

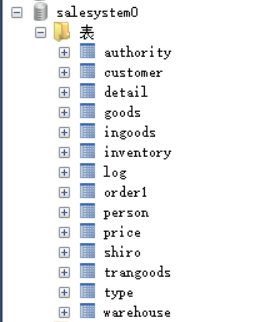
### 

### 结果：如下图可以看到，插入语句实现分库操作

### 

**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。**

在进销存系统中，有十四张表，如下：



原本的数据库是在服务器上，为了进行修改测试，我把原本服务器上的数据库导入到本地，并复制一份进行分库分表操作。

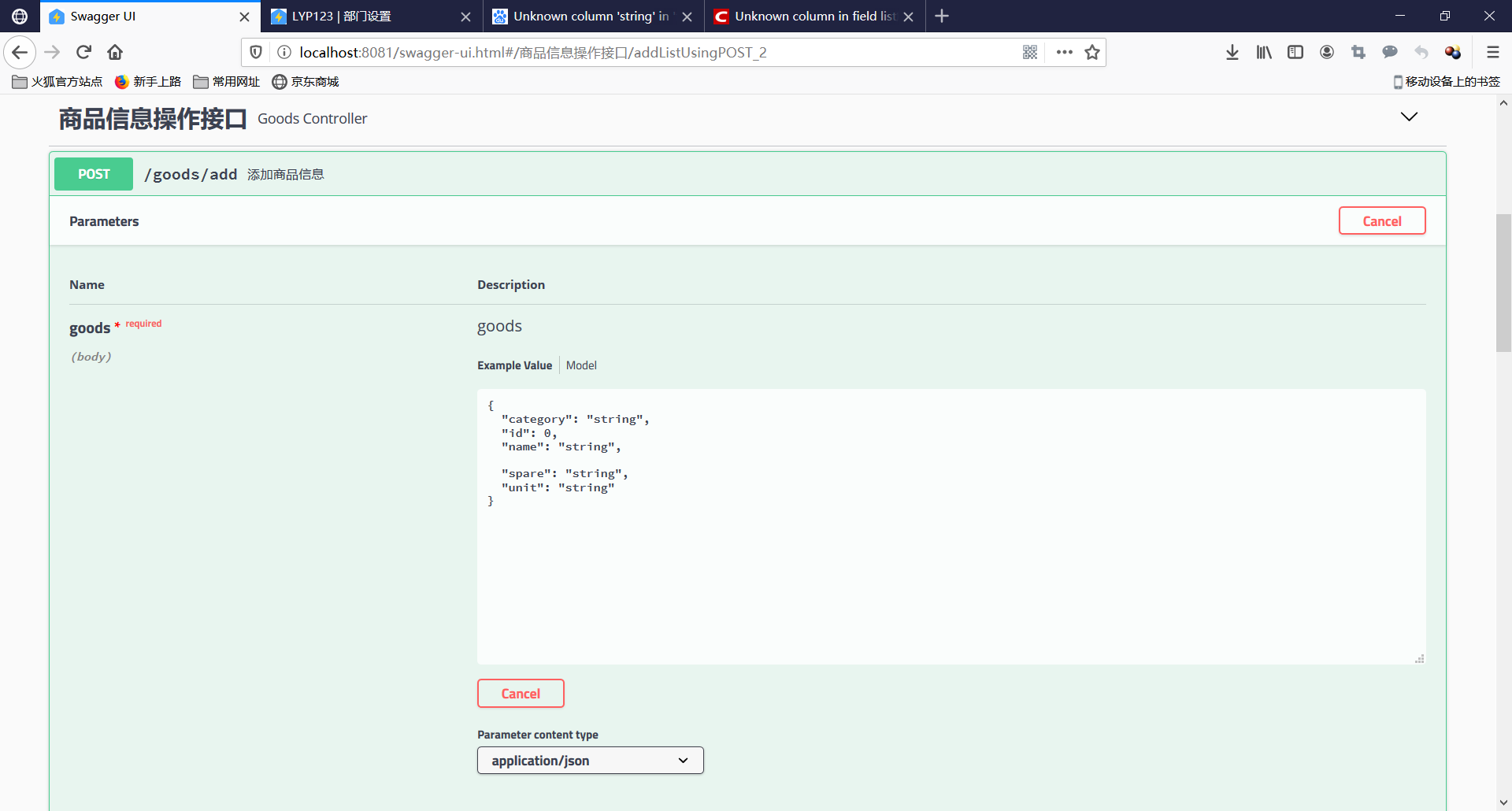
下面是yml配置文件



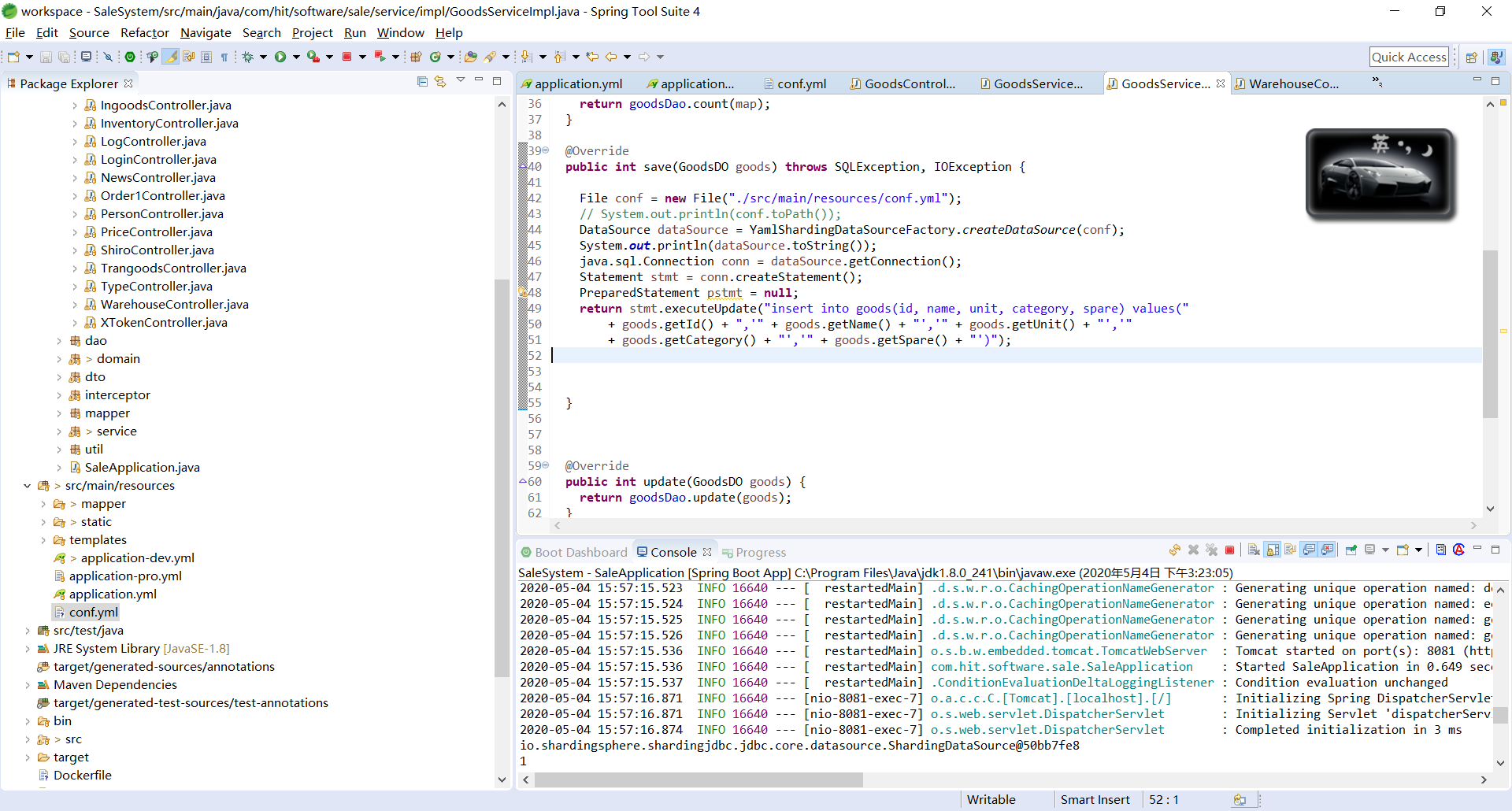
可以看到，数据库连接已经修改到本地，这里使用goods表进行测试。

修改表结构，使goods表具有分类的属性。

这里为了让结果查看更加清楚，使用swagger进行测试

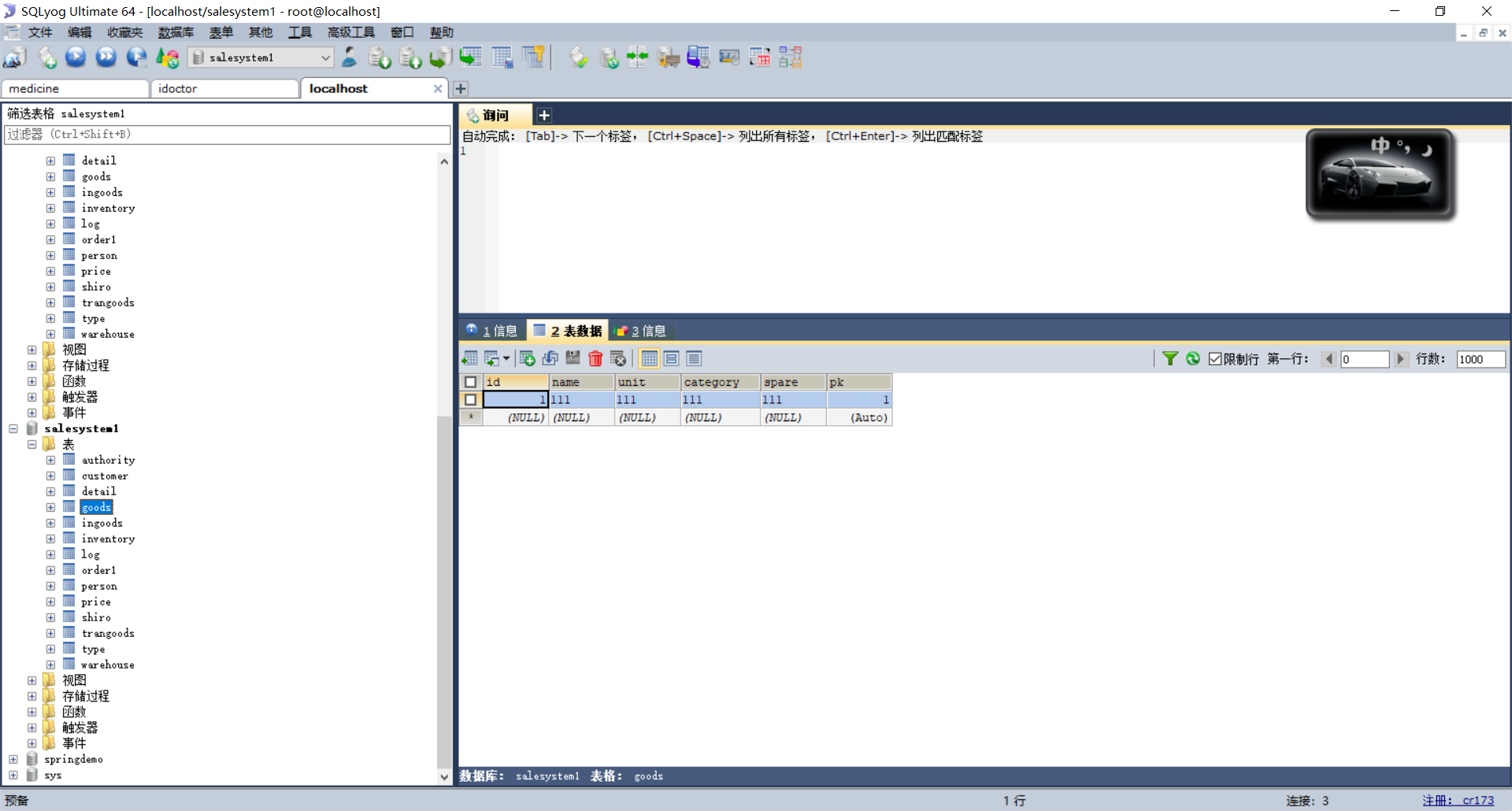
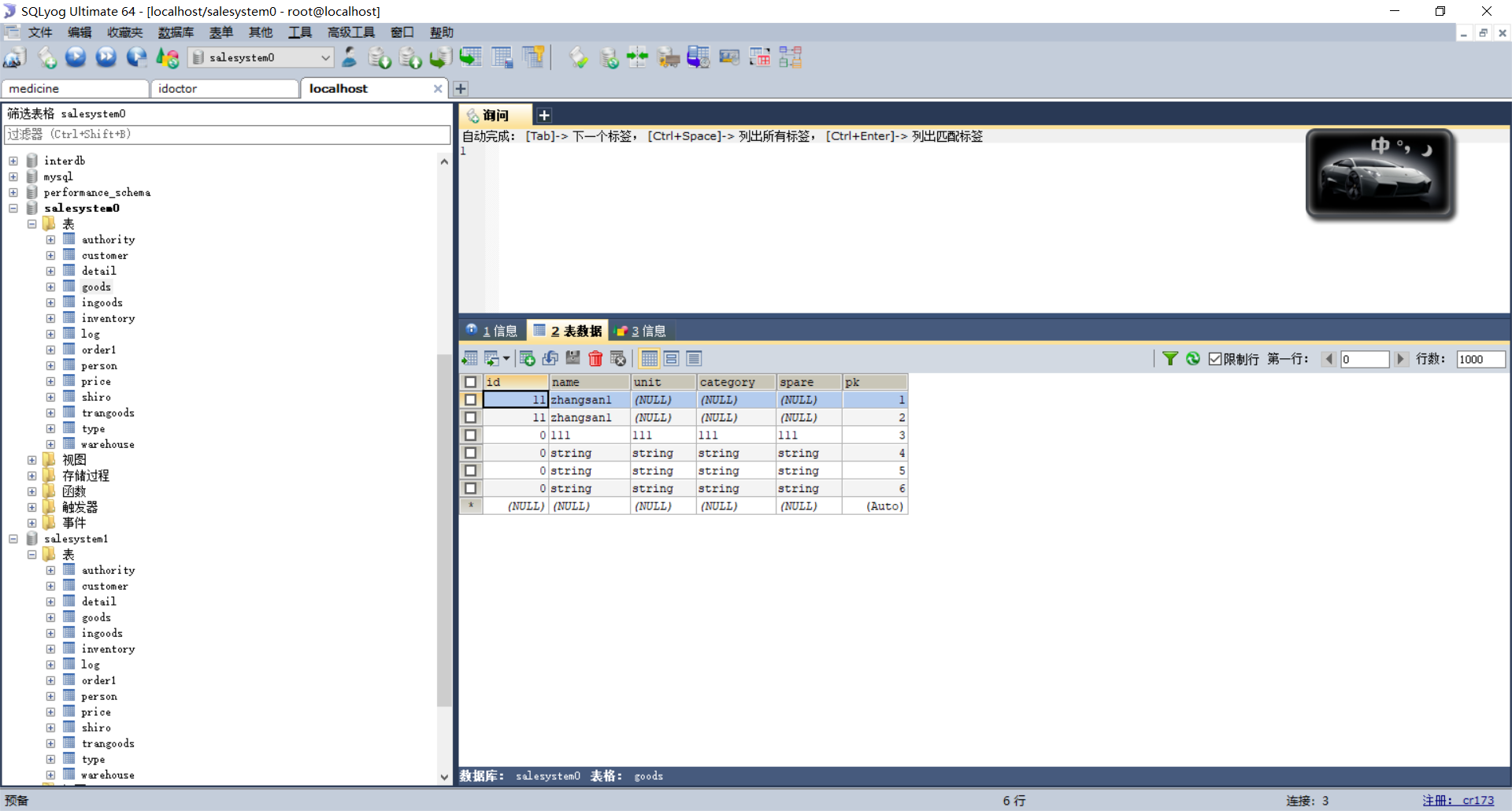


这里使用id进行区分，id=0表示数据插入到salesystem0数据库中，id=1表示数据插入到salesystem1数据库当中。



Service层输出1表示数据插入成功。

查看数据库信息：



可以看到，当id为0时，添加数据到salesystem0数据库当中。

当id=1时，添加数据到salesystem1数据库当中。

3.3Redis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。**

1.问题：Redis官方不支持Windows

解决：从Github下载MicroSoft提供的旧版本

配置安装过程：

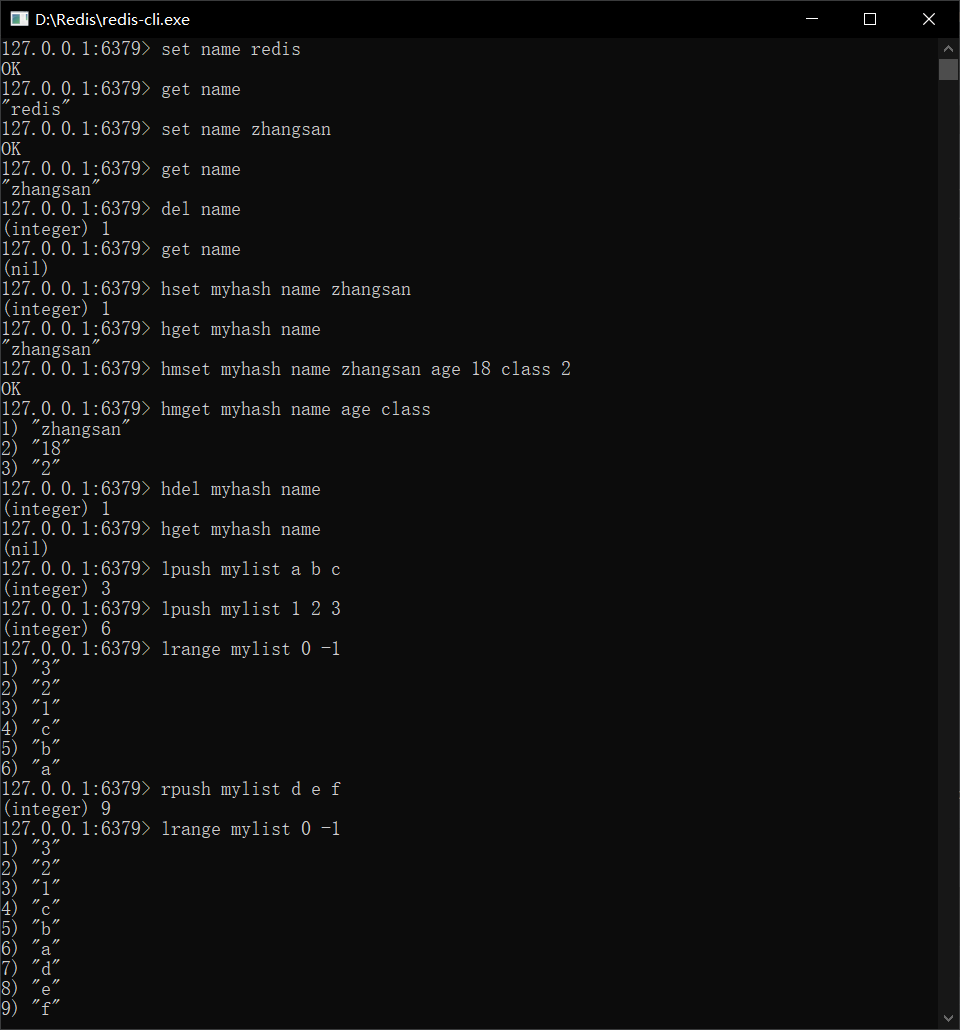
1.基本开发环境准备：

安装redis：



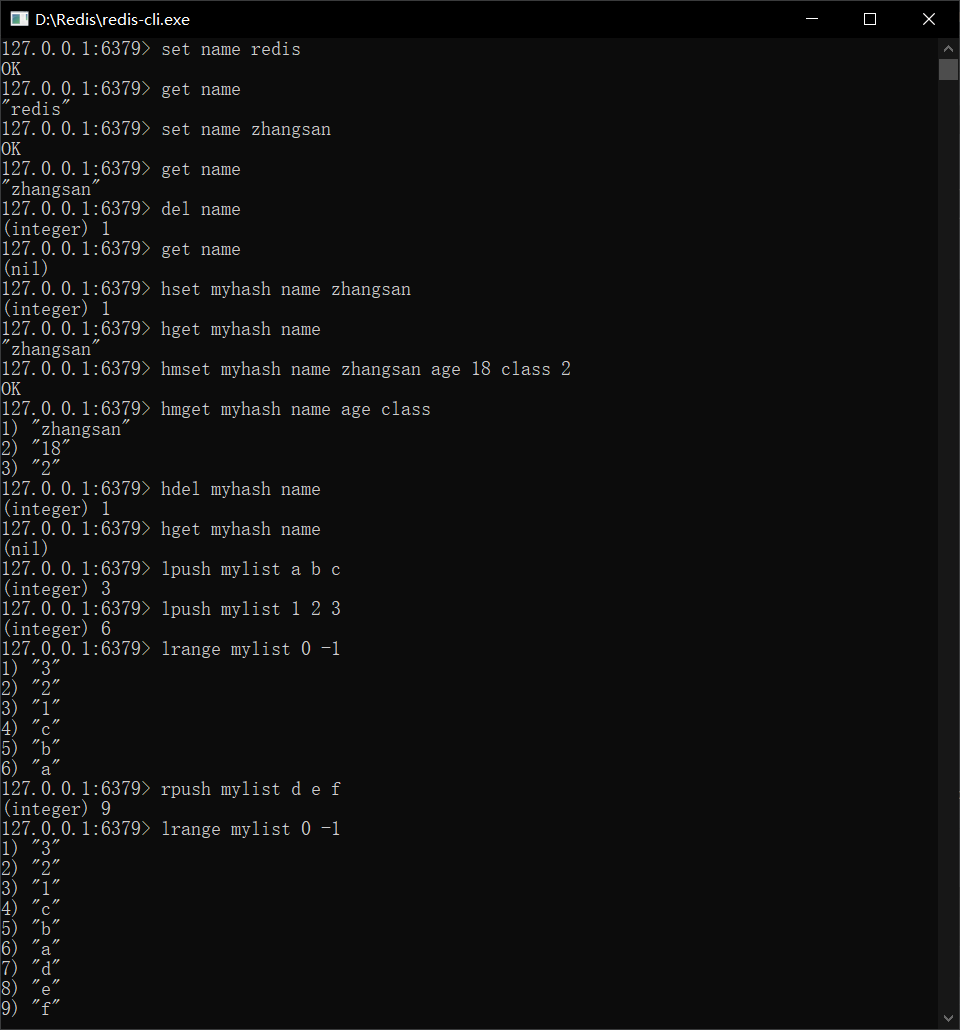
2.redis-cli的使用：

数据类型：string



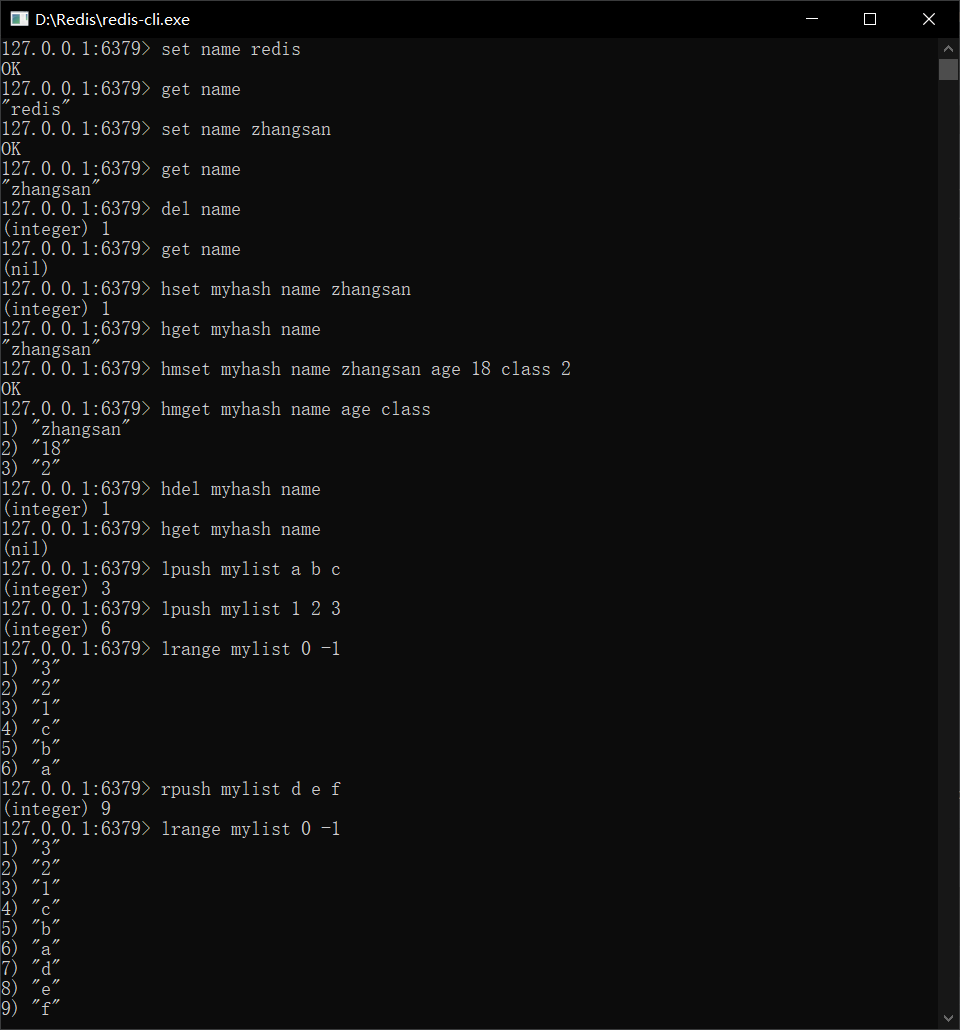
数据类型：hash

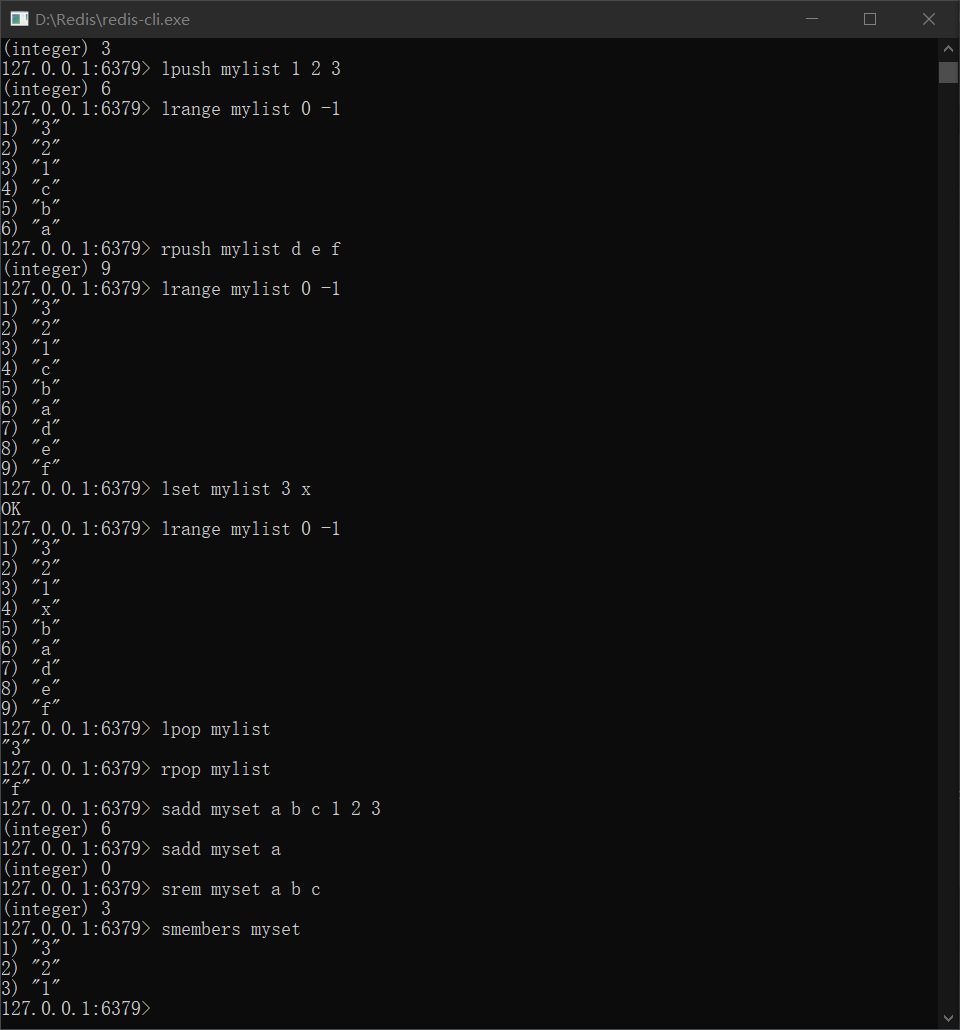
hash可以存储多个键值对之间的映射



数据类型：list

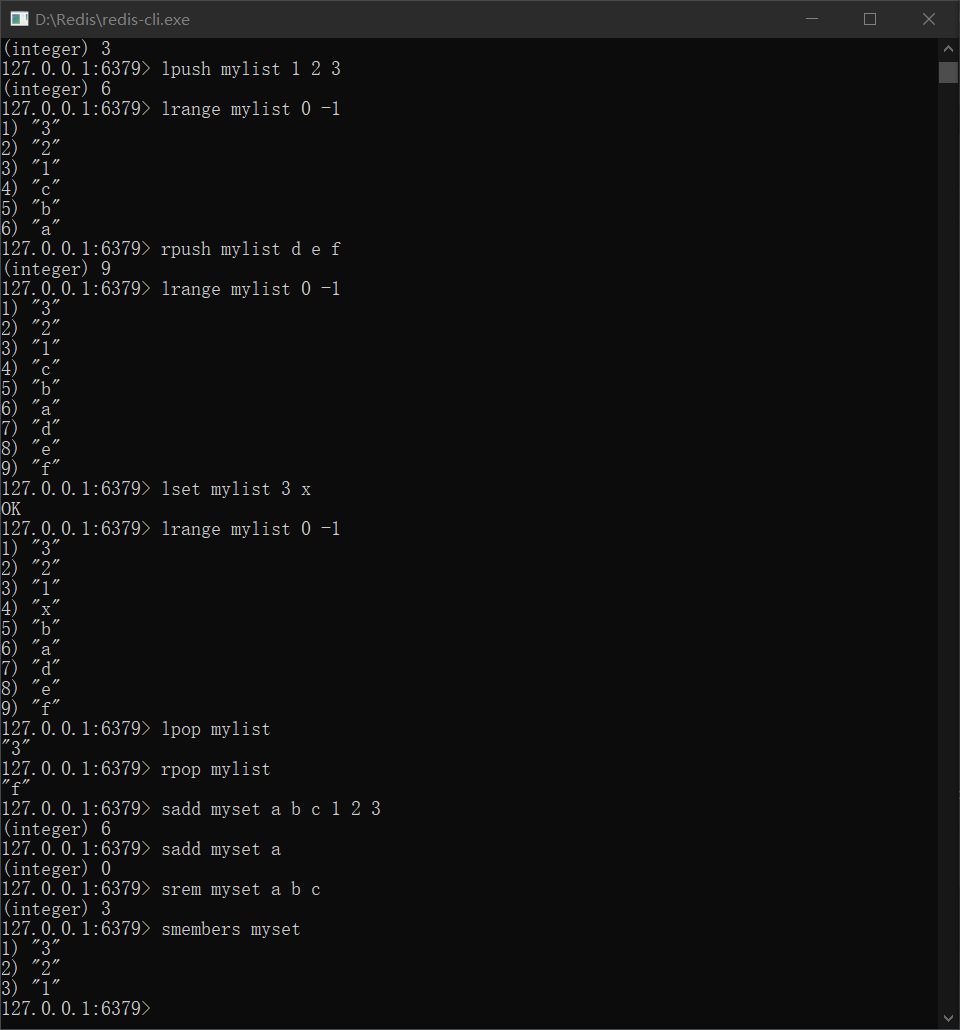
list的顺序是按照插入的顺序，可以在头部跟尾部插入数据





数据类型：set

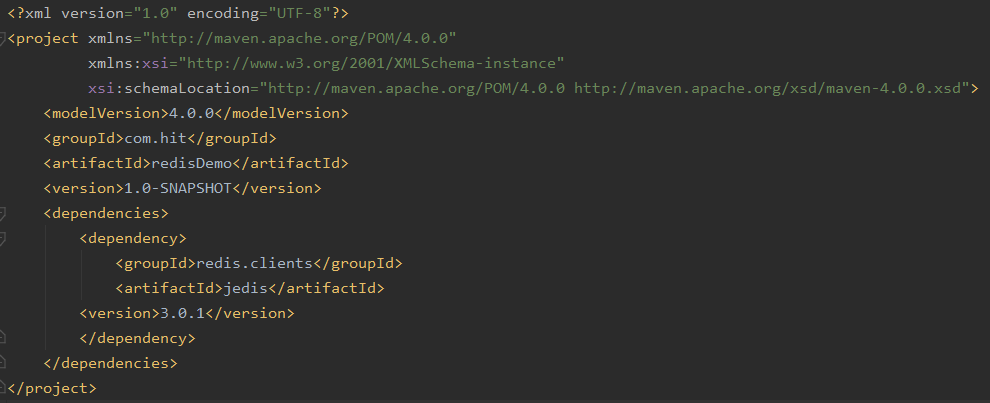
set 中不允许出现重复的元素，没有顺序



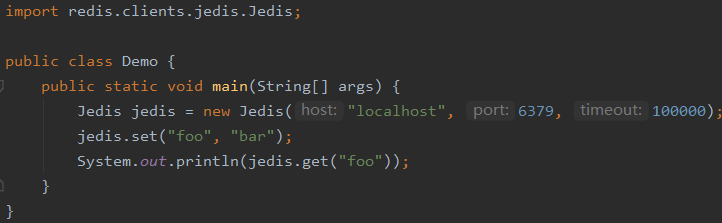
3.redis java client: Jedis的使用

使用idea创建maven项目

修改pom.xml添加jedis依赖，修改后的文件内容如下：

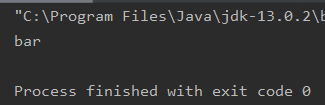


新建文件/src/main/java/Demo.java，文件内容如下：



运Demo.main()：

输出：bar



**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。**

缓存清洗策略：

Redis缓存淘汰策略指在Redis内存使用超过一定值的时候使用的淘汰策略；Redis键的过期策略是指通过定期删除和惰性删除两者结合的方式进行内存淘汰。

过期策略：定期删除：redis会把设置了过期时间的key放在单独的字典中，每隔一段时间执行一次删除过期key的操作；惰性删除：key过期的时候不删除，每次从数据库获取key的时候去检查是否过期，若过期，则删除，返回null；定时删除：在设置key的过期时间的同时，为该key创建一个定时器，让定时器在key的过期时间来临时，对key进行删除。

淘汰策略：常见的淘汰算法：FIFO：First In First Out，先进先出。判断被存储的时间，离目前最远的数据优先被淘汰。LRU：Least Recently Used，最近最少使用。判断最近被使用的时间，目前最远的数据优先被淘汰。LFU：Least Frequently Used，最不经常使用。在一段时间内，数据被使用次数最少的，优先被淘汰。Redis提供的淘汰策略：noeviction：当内存使用超过配置的时候会返回错误，不会驱逐任何键。allkeys-lru：加入键的时候，如果过限，首先通过LRU算法驱逐最久没有使用的键。volatile-lru：加入键的时候如果过限，首先从设置了过期时间的键集合中驱逐最久没有使用的键。allkeys-random：加入键的时候如果过限，从所有key随机删除。volatile-random：加入键的时候如果过限，从过期键的集合中随机驱逐。volatile-ttl：从配置了过期时间的键中驱逐马上就要过期的键。volatile-lfu：从所有配置了过期时间的键中驱逐使用频率最少的键。allkeys-lfu：从所有键中驱逐使用频率最少的键。

数据迁移：

离线迁移：停服务器，搬迁数据，可以通过RDB或者AOF；

主从同步转移：

1.首先在新服务器上直接进入redis-cli，执行从库配置slaveof 192.168.1.100 6379，这里假设要将192.168.1.100的6379端口的redis服务转移过来。这样就已经开始同步了。通过info可以查看当前服务器是slave。

2.然后通过info命令查看master\_link\_status，如果为up，表示同步完成。在同步过程中，执行查询的时候还是会提示"Redis is loading the dataset in memory"，这属于正常情况.把数据从磁盘文件加载到内存中可能会消耗很长的一段时间。

3.最后断开主从关系，在redis-cli命令行下执行slaveof no one提示OK，再通过info查看，该新服务器已经自己变成master了。

扩容策略：

Redis的内部扩容机制；

Redis是一个键值对数据库服务器，Redis服务器中的所有数据库保存在db数组中，其中redisDb结构的dict字典保存了数据库中的所有键值对，Redis的扩容机制，指的就是字典中哈希表的rehash重新散列操作。为了让哈希表的负载因子维持在一个合理的范围内，会使用rehash操作对哈希表进行相应的扩展或收缩。

Redis可扩展集群搭建：

可以使用多台服务器来存储也可以在一台服务器上部署多个Redis实例。Redis的作者提出了一种叫做presharding的方案来解决动态扩容和数据分区的问题，实际就是在同一台机器上部署多个Redis实例的方式，当容量不够时将多个实例拆分到不同的机器上，这样实际就达到了扩容的效果。拆分过程如下：

1.在新机器上启动好对应端口的Redis实例。

2.配置新端口为待迁移端口的从库。

3.待复制完成，与主库完成同步后，切换所有客户端配置到新的从库的端口。

4.配置从库为新的主库。

5.移除老的端口实例。

6.重复上述过程迁移好所有的端口到指定服务器上。

缓存雪崩、穿透：

缓存穿透：key对应的数据在数据源并不存在，每次针对此key的请求从缓存获取不到，请求都会到数据源，从而可能压垮数据源。一个一定不存在缓存及查询不到的数据，由于缓存是不命中时被动写的，并且出于容错考虑，如果从存储层查不到数据则不写入缓存，这将导致这个不存在的数据每次请求都要到存储层去查询，失去了缓存的意义。有效地解决缓存穿透问题最常见的是采用布隆过滤器，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。另一个更为方法，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。

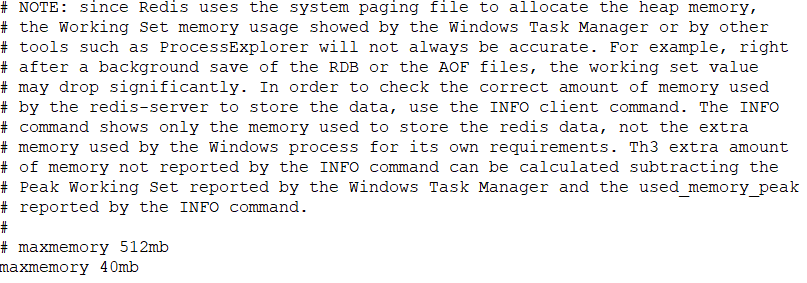
缓存击穿：key对应的数据存在，但在redis中过期，此时若有大量并发请求过来，这些请求发现缓存过期一般都会从后端DB加载数据并回设到缓存，这个时候大并发的请求可能会瞬间把后端DB压垮。key可能会在某些时间点被超高并发地访问，是一种非常“热点”的数据。这个时候，需要考虑一个问题：缓存被“击穿”的问题。使用互斥锁，在缓存失效的时候，判断拿出来的值为空，不是立即去load db，而是先使用缓存工具的某些带成功操作返回值的操作去set一个mutex key，当操作返回成功时，再进行load db的操作并回设缓存；否则，就重试整个get缓存的方法。

缓存雪崩：当缓存服务器重启或者大量缓存集中在某一个时间段失效，这样在失效的时候，也会给后端系统带来很大压力。与缓存击穿的区别在于这里针对很多key缓存，前者则是某一个key，缓存正常从Redis中获取。用加锁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，但是加锁排队只是为了减轻数据库的压力，并没有提高系统吞吐量。假设在高并发下，缓存重建期间key是锁着的，这时过来1000个请求999个都在阻塞的。同样会导致用户等待超时。另一个方案是将缓存失效时间分散开，在原有的失效时间基础上增加一个随机值，这样每一个缓存的过期时间的重复率就会降低，就很难引发集体失效的事件。

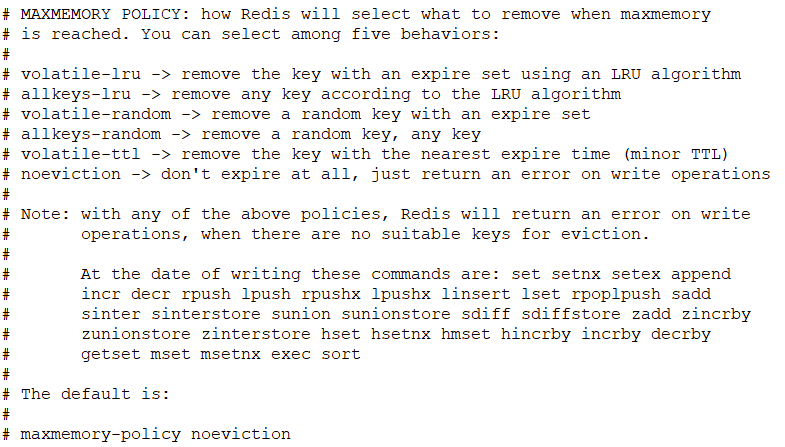
**3）请在进销存系统(或其他实际软件系统)设计一个简单场景，实现缓存读写操作，缓存更新操作，给出缓存的效果，分析2问题中相关策略的效果。**

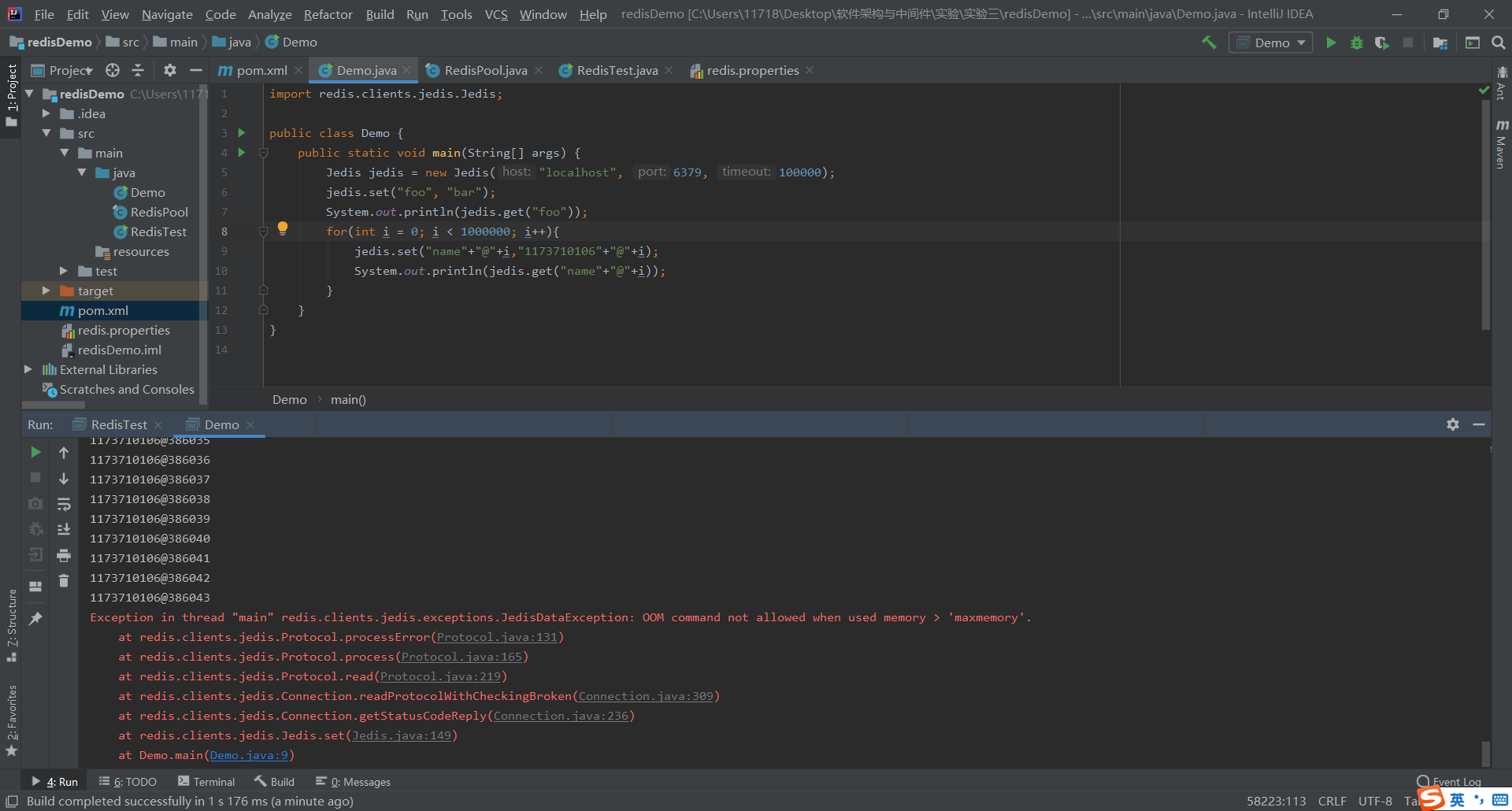
修改redis.windows-service.conf的配置

修改最大内存，方便对比效果：

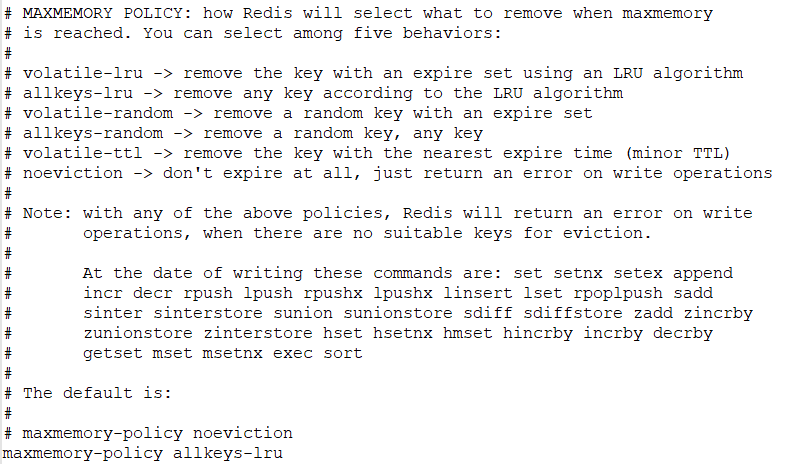


内存淘汰策略，默认超出最大内存时直接报错





更换缓存淘汰策略为在主键空间中，优先移除最近未使用的key：





1. **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 |
| 5.2 | 9:00-12:00  15:00-18:00 | 黄项宇轩 | 任永鹏 | 熟悉实验内容，完成3.1 |
| 5.3 | 10:00-12:00  15:00-19:00 | 任永鹏 | 黄项宇轩 | 完成3.2 |
| 5.4 | 10:00-13:00  15:30-18:00 | 黄项宇轩 | 任永鹏 | 完成3.3 |
| 5.4 | 18:00-20:00 | 任永鹏 | 黄项宇轩 | 完成报告 |

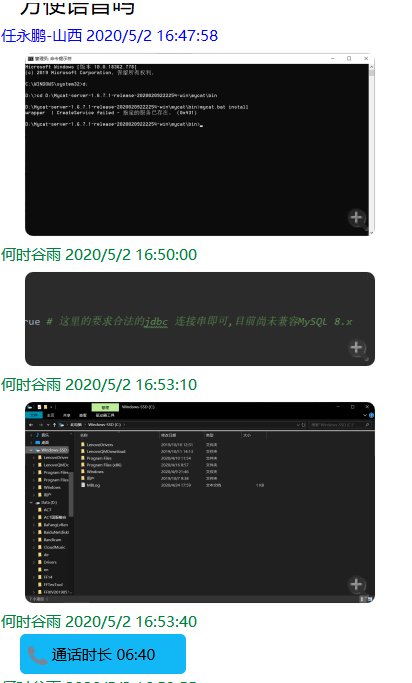
**（2）工作日志**

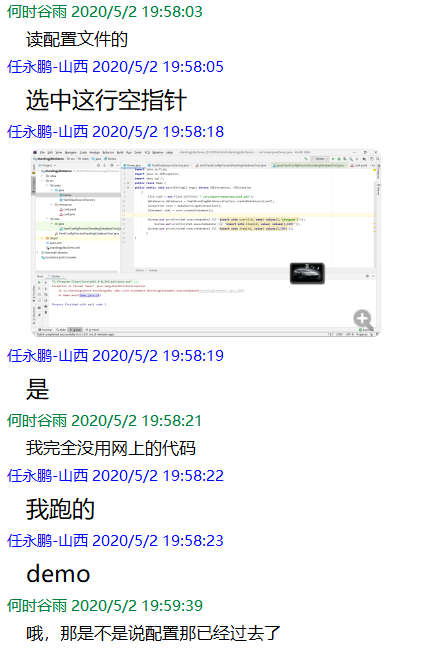
由领航员负责记录，记录结对开发期间的遇到的问题、两人如何通过交流合作解决每个问题的。

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 |
| 5.2  9:30 | Mycat启动报错 | 查看logs逐条修改出错配置 | 搜索引擎查询分享 |
| 5.2  15:00 | Mycat读取报错 | 读连接不能用localhost | 搜索引擎查询分享 |
| 5.3  11:00 | yml配置文件错误 | 以排列组合的方式尝试标签和缩进 | 组内交流 |
|  |  |  |  |
| 5.3  16:00 | Jdbc插入错误 | 分片策略与原系统冲突 | 组内交流 |
| 5.3  17:00 | Unknown column  &apos;string&apos;  in &apos;field list&apos; | string字段需要使用双引号里面嵌套单引号，从而保证string类型 | 组内交流 |
| 5.4  12:00 | 依赖包无法自动导入 | 更换工具STS | 组内交流 |
| 5.4  17:00 | Redis直接超出内存 | 推测存在内存下限或者有基础内存需求 | 组内交流 |

**（3）结对开发工作现场照片或线上讨论截图**







1. **实验总结**

经过本次的实验，我们组成员学会使用Mycat和Sharding-JDBC实现数据分库分表

以及使用Redis数据库实现数据缓存，并且能够灵活应用Mycat，Sharding-JDBC，Redis到实际的系统当中，学会了利用缓存架构实现数据读取的性能提升。经过3个小实验的学习，我们组成员在解决尚未熟悉领域的错误的能力具有显著提升，这种对于新领域知识的学习有很大帮助。