**《网上手机商城》课程设计报告**

**小组组成情况：**

|  |  |  |
| --- | --- | --- |
| **学号** | **姓名** | **贡献率（合计100%）** |
| 20251104157 | 郑圳辉 | 33.3333 |
| 20251104155 | 林晓峰 | 33.3333 |
| 20251104159 | 朱钦瀚 | 33.3333 |

**〇、项目概况**

1、项目名称

网上手机商城

2、项目背景

网上手机商城是一个在线上浏览手机参数信息，进行手机的购买操作的系统，它取代了当前线下门店通过店员面对面介绍的方式了解手机详细参数，并且面对面付款的这样一个过程，可以提高用户对更多品牌商品详细信息的快速了解，让用户不用走访任何一家品牌店就可以获得各种品牌手机的详细参数，价格等等。提高购买效率和优化用户了解的渠道。

3、项目开发目标

网上商城系统是一个基于web的网页商城，在这里用户可以注册一个自己的帐号，接收商城的热门手机商品的推荐、通过搜索框输入品牌等关键字搜索商品，并且有一个商品品牌分类来进行对不同品牌手机的筛选，同时系统管理员实现对用户的帐号以及商品信息进行管理。

1. **需求分析**

1、用户分类：

①用户：可以进行注册和登录商城， 查看自己的购物车页面， 和订单页面， 可以查询浏览不同类别的商品， 进行购买。

2、系统功能划分：浏览功能、登录功能、注册功能、结算功能，加入购物车功能，移除购物车功能。

3、系统功能需求描述：

（1）登录和注册：未注册用户注册，已注册用户登录，使用正确的账号和密码进入个人账号，可以进行购买物品和查询商品。

（2）加入购物车：用户将商品加入购物车。

（3）移出购物车：用户将不要的商品移出购物车清单。

（4）结算：用户可以对商品进行购买。

（5）商品查询：用户可以查询自己想要的相关商品。

。

4、安全需求

在用户注册的时候，必须遵循系统的要求。在用户登录过程中，如果账号密码不匹配，是不被允许登录的。同时，为了保证用户隐私，对未登录状态的用户进行拦截。

1. 系统整体分析及模块分解

在网上商城系统中， 我们前端使用了最近最流行的前端框架之一Vue , 来做页面渲染和页面响应， 后端则采用了阿里巴巴微服务架构， 将项目分成了四个微服务， 服务采用nacos做为服务注册中心， 然后我们用gateway 作为我们的网关， 前端Vue 的请求都会先经过这个网关， 然后这个网关再去访问注册中心的微服务， 值得注意的是， 我们用gateway 去访问注册中心的服务的时候， 会对同一个服务名的微服务进行负载均衡的调用，

然后我们调用一个微服务的时候， 可能这个微服务也会对其他的微服务进行调用， 这里我们采用了openfeign这个技术， 去调用其他服务 ， 同网关一样， 它在调用服务的时候， 可以负载均衡的进行调用， 其内部其实都是采用了loadbalanced技术进行负载均衡， 然后在jdbc方面， 我们用了mybatis-plus ,作为我们操作数据库的桥梁， 数据库方面则用了oracle大型数据库作为我们数据持久化的地方。

又因为， 在界面上， 用户可以注册账户， 登录账户， 然后其他操作可能需要账户的信息， 所以我们第一个微服务就是用户服务， 然后我们是商城，当然少不了商品， 包括首页 的商品展示， 各个类别的商品， 用户将商品加入购物车， 所以第二个必不可少的服务就是商品服务， 当然每个用户都应该有其独立的购物车列表， 所以第三个服务就是购物车服务， 然后当用户结算完了之后， 应该有一个订单， 所以最后一个服务就是订单服务， 我们可以注意到购物车服务是需要调用商品服务的， 然后订单则是商品服务和用户服务都需要调用， 然后整体的模块分解如图下，

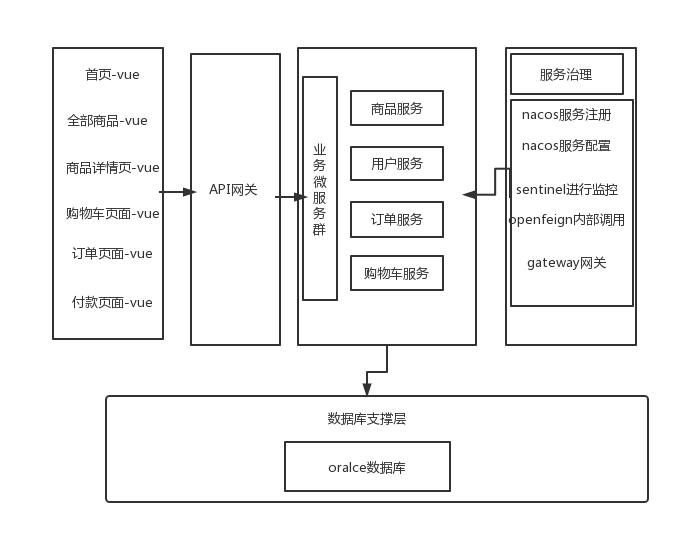


图1 系统功能分解模块图

该系统的部署结构如图 2 所示，其中用户在 PC 机上通过 Web 浏览器访问网服务，网关服务再去调用其他服务， 其他服务再去访问数据库。

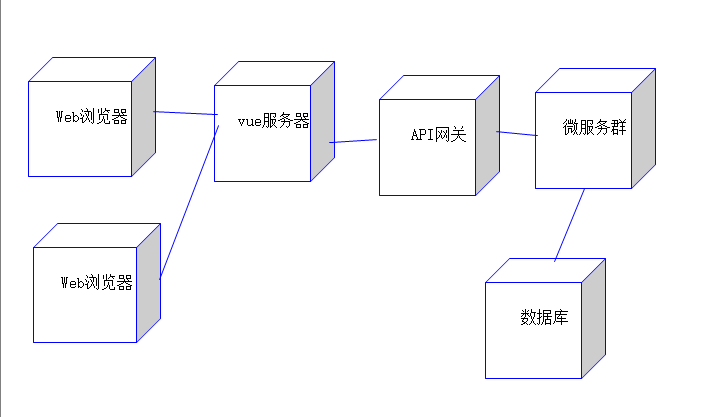


图 2 网上手机商城系统的部署图

1. **概要设计**

1、项目开发工具及环境：

（1）操作系统: Windows 10

（2）开发工具: Intellij IDEA 2021.3.2 (Ultimate Edition)

（3）Java版本: 1.8.0

（4）服务器: Tomcat 8.5

（5）数据库: Oracle11g

（6）前端技术: HTML+CSS+JavaScript+Vue+Axios

（7）后端技术: Java+SpringBoot+MyBatis-plus+Oracle11g+微服务框架+nacos+sentinel+ gateway+openfeign+maven

2、系统架构设计

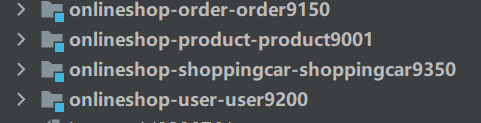
（1）本系统的整体架构主要是微服务架构，我们将由功能将所有服务划分为四个服务。

①商品服务:对商品从数据库进行 CRUD，即增加(Create)、读取查询(Retrieve)、更新(Update)和删除(Delete）。

②用户服务：对用户数据从数据库进行 CRUD，即增加(Create)、读取查询(Retrieve)、更新(Update)和删除(Delete）。

③购物车服务:对购物车数据从数据库进行 CRUD，即增加(Create)、读取查询(Retrieve)、更新(Update)和删除(Delete）。

④订单服务:对订单数据从数据库进行 CRUD，即增加(Create)、读取查询(Retrieve)、更新(Update)和删除(Delete）。



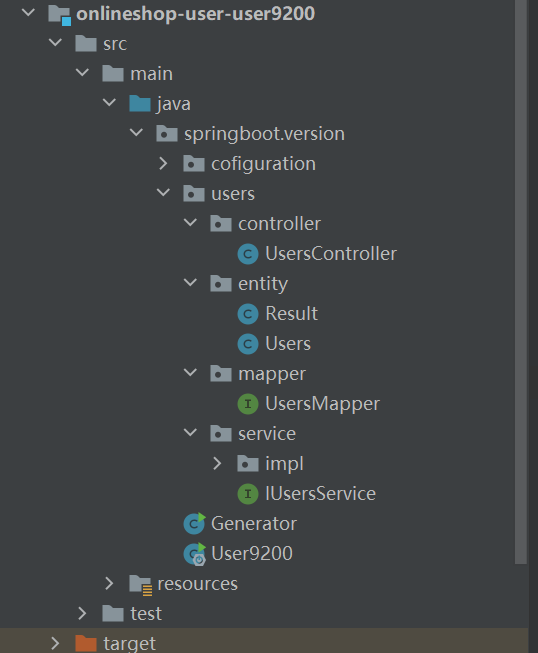
而单一微服务的主要架构是 DAO--->Mapper--->Service-->Controller。

①DAO 层:从数据库进行 CRUD，即增加(Create)、读取查询(Retrieve)、更新(Update)和删除(Delete）。

②Mapper层：sql语句的映射文件。

③Service 层:对 DAO 层获取的数据进行加工处理，实现功能方法。

④Controller 层:显示层，处理 Service 的数据并进行显示。



（2）本系统的采用框架有：

①Vue：Vue (发音为 /vjuː/，类似 view) 是一款用于构建用户界面的 JavaScript 框架。它基于标准 HTML、CSS 和 JavaScript 构建，并提供了一套声明式的、组件化的编程模型，帮助你高效地开发用户界面。无论是简单还是复杂的界面，Vue 都可以胜任。

②Springboot：选择Spring Boot，因为它提供的功能和优点如下 -

* 它提供了一种灵活的方法来配置Java Bean，XML配置和数据库事务。
* 它提供强大的批处理和管理REST端点。
* 在Spring Boot中，一切都是自动配置的; 无需手动配置。
* 它提供基于注释的spring应用程序。
* 简化依赖管理。
* 它包括嵌入式Servlet容器。

//更多请阅读：https://www.yiibai.com/spring-boot/spring\_boot\_introduction.html

3、数据库设计

（1）数据库的设计及数据库信息如下，总共有7张表，分别为：

1.用户表（users）、

2.商品类别表（category）、

3.商品表（product）、

4.购物车表（ shoppingCart）

5.订单表（orders）

6.商品照片表（product\_picture）

7.首页轮番图表（ carousel）

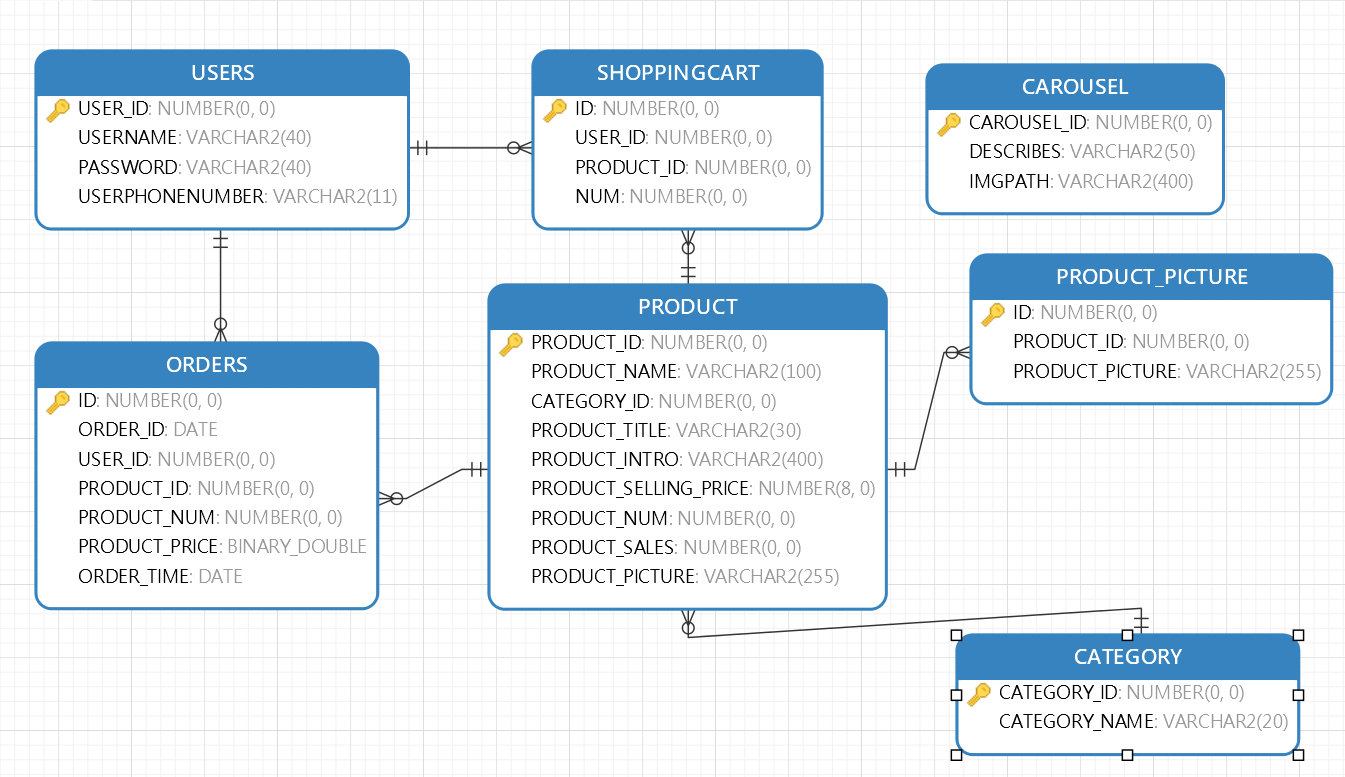


图2 数据库的物理模型

（2）数据库表

1 TAB-001:Users

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 字段名 | 字段含义 | 数据类型 | 是否允许空 | 是否主键 | 是否外键 |
| user\_id | 用户id | number |  |  是 |  |
| userName | 用户名称 | varchar(20) | 否 |  |  |
| password | 用户密码 | varchar(20) | 否 |  |  |
| userPhoneNumber | 用户电话号码 | varchar(11) | 否 |  |  |

2TAB-002:Product

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 字段名 | 字段含义 | 数据类型 | 是否允许空 | 是否主键 | 是否外键 |
| product\_id | 商品id | int |  | 是 |  |
| Product-name | 商品名称 | varchar(20) | 否 |  |  |
| category\_id | 商品类别id | varchar(20) | 否 |  | 是 |
| product\_title | 商品详情页标题 | varchar(200) | 否 | ... | ... |
| product\_intro | 商品简介 | varchar(200) | 否 |  |  |
| product\_picture | 商品主图 | varchar(200) | 否 |  |  |
| product\_selling\_price | 商品价格 | number | 否 |  |  |
| product\_num | 存货 | number | 否 |  |  |
| product\_sales | 销售量 | number | 否 |  |  |

3 TAB003:ShoppingCart

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 字段名 | 字段含义 | 数据类型 | 是否允许空 | 是否主键 | 是否外键 |
| id | 无 | number |  | 是 |  |
| user\_id | 用户 id | varchar(20) | 否 |  | 是 |
| product\_id | 商品id | varchar(20) | 否 |  | 是 |
| num | 商品数量 | number | 否 | ... | 是 |

4 TAB004:Orders

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 字段名 | 字段含义 | 数据类型 | 是否允许空 | 是否主键 | 是否外键 |
| ID | 无 | number |  |  是 |  |
| orderId | 订单编号 | varchar(20) | 否 |  |  |
| userId | 用户id | varchar(20) | 否 |  | 是 |
| productId | 商品id |  | 否 |  | 是 |
| productNum | 商品数量 |  | 否 |  |  |
| productPrice | 总价格 |  | 否 |  |  |
| orderTime | 下订单时间 |  | 否 |  |  |

5 TAB005:category

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 字段名 | 字段含义 | 数据类型 | 是否允许空 | 是否主键 | 是否外键 |
| category\_id | 分类的id | number |  |  是 |  |
| category\_name | 分类的类名 | varchar(20) | 否 |  |  |

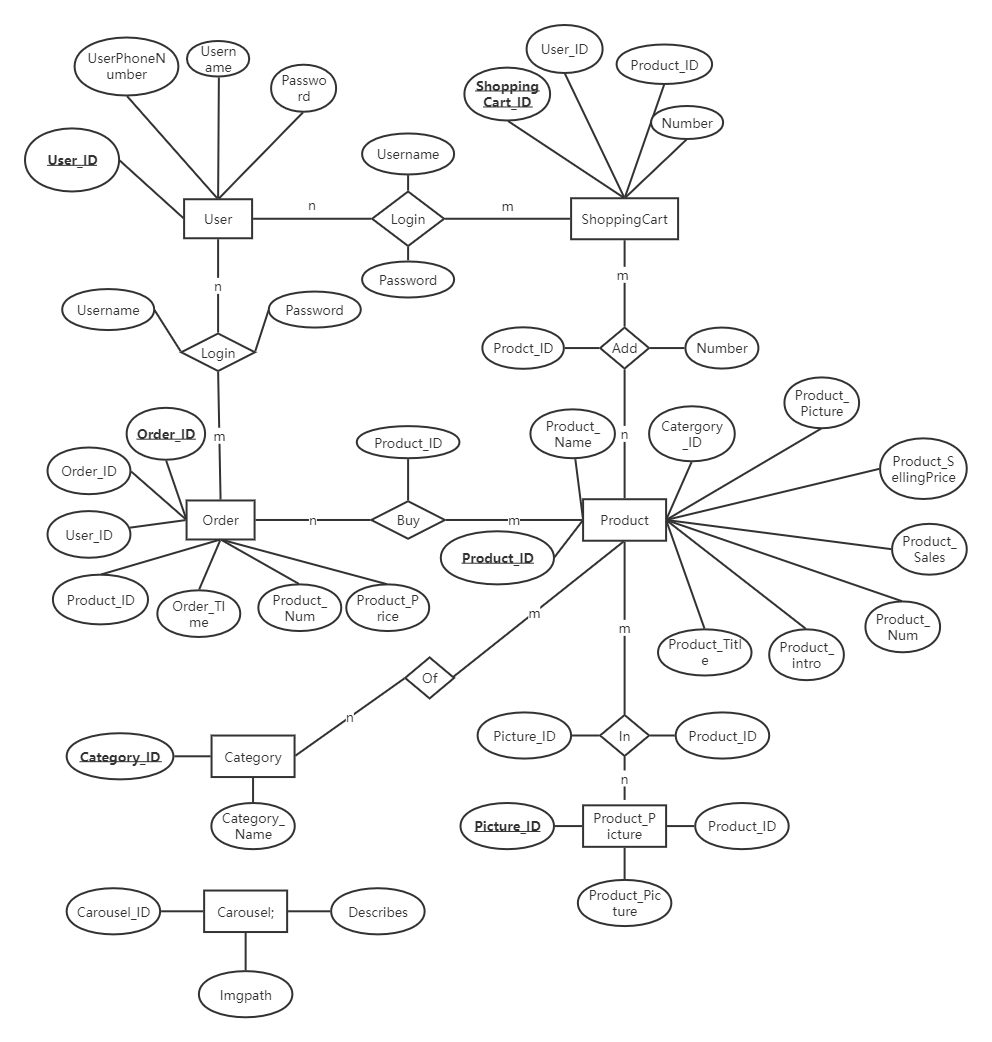
6 TAB006：product\_picture

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 字段名 | 字段含义 | 数据类型 | 是否允许空 | 是否主键 | 是否外键 |
| id | id | number |  |  是 |  |
| product\_id | 商品id | number | 否 |  | 是 |
| product\_picture | 商品图片路径 | varchar(200) |  |  |  |

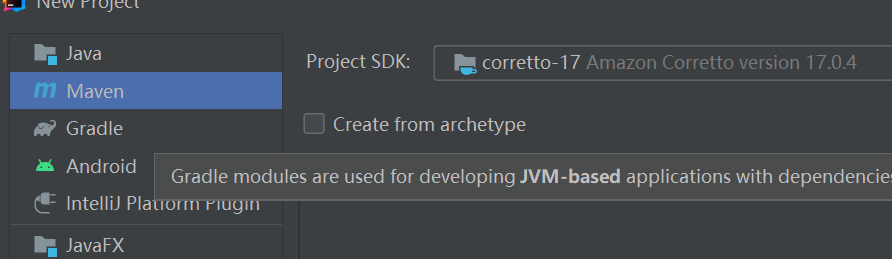
7 TAB007:carousel

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 字段名 | 字段含义 | 数据类型 | 是否允许空 | 是否主键 | 是否外键 |
| carousel\_id | id | number |  |  是 |  |
| imgPath | 图片的路径 | varchar(200) | 否 |  |  |
| describes | 图片描述 | varchar(50) |  |  |  |

(3)E-R图



1. **详细设计**
2. 首先我们将整体的框架计划好后， 第一步就是在springboot中建立一个maven工程，作为我们的父工程， 然后在父工程中配置好相关依赖的版本， 这样我们建立模块，后添加相关依赖就不用添加版本， 这样确保了所有模块的依赖版本是一样的， 即版本管理





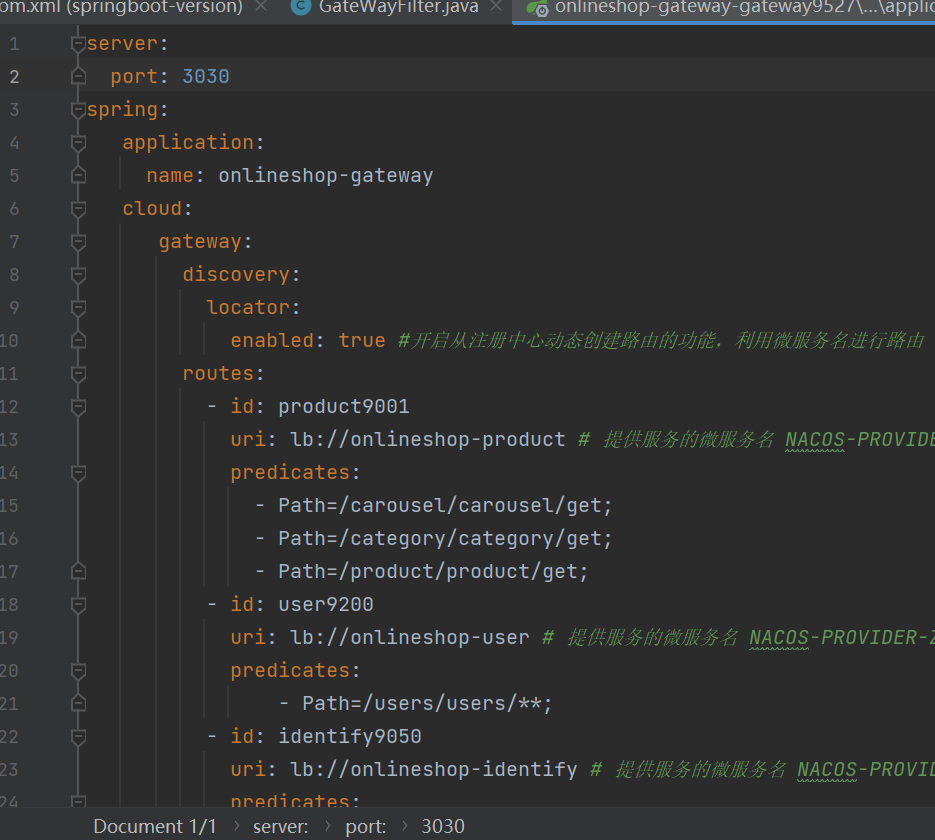
1. 现在我们创建好了一个父工程， 我们就可以创建一个网关， 在项目中添加模块

然后配置gateway作为我们的网关端口号为3030 . 然后服务名为onlineshop-gateway

将这个网关服务注册进nacos，然后同样的用sentinel进行监控， 这样我们就可以对访问进行限流 （记得开启nacos服务和sentinel），然后我们通过添加路由的方式， 就可以使前端通过这个网关访问到后端的所有微服务











1. 现在我们已经有网关了， 我们接下来就需要将四个微服务一个个的完成， 这四个微服务的构建大同小异， 都是五步

（1）建module

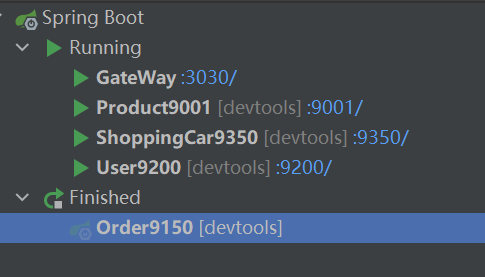
（2）改POM

（3）写yml

（4）主启动

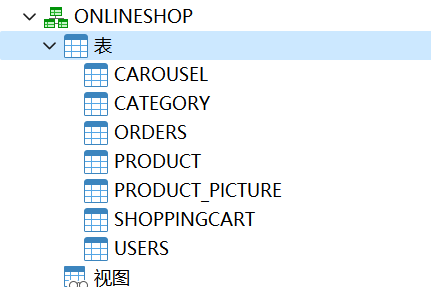
（5）业务类的编写

1. 当我们完成到第四步的时候，我们启动所有微服务的主启动类， 我们可以在nacos中看到这时我们四个微服务已经初步完成





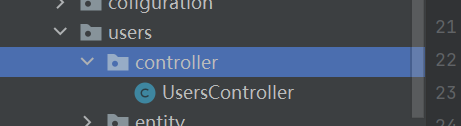
1. 、整合数据库，在Oracle数据库中创建对应的七个表。

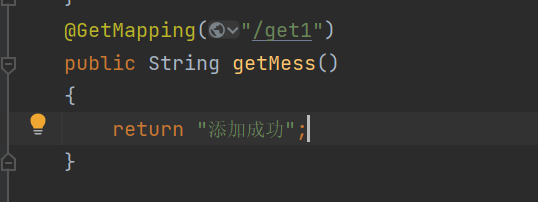


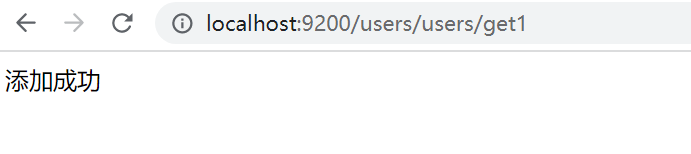
6、修改pom.xml，创建实体类，定义属性、属性名和列名保持一致。创建Dao接口，定义操作数据库的方法，创建sql映射文件实现增删改查，一个表对应一个mapper文件。同时创建Mybatis的主要配置文件，定义创建连接实例的数据源对象以及指定其他mapper文件的位置。

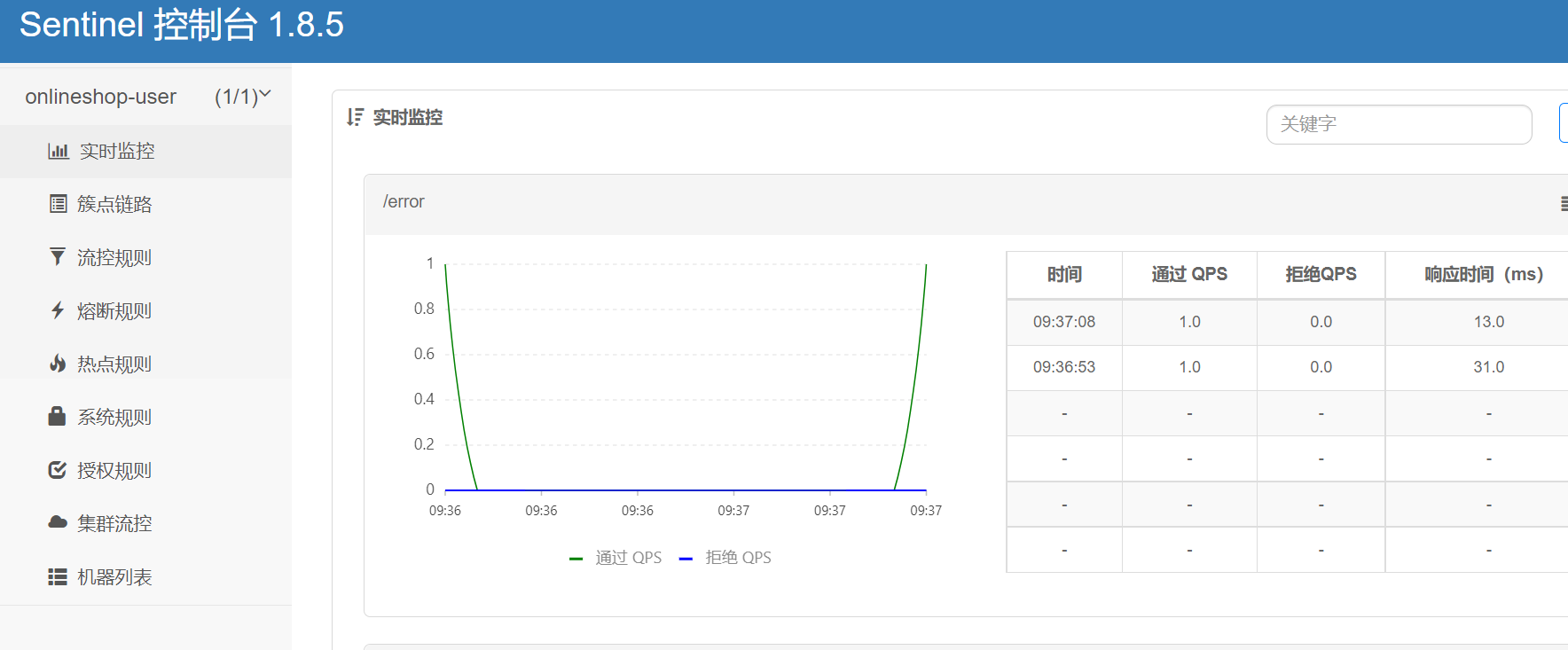
7.然后创建每个服务的controller，接受前端的请求返回数据， 和响应信息

8.进行sentinel监控请求测试



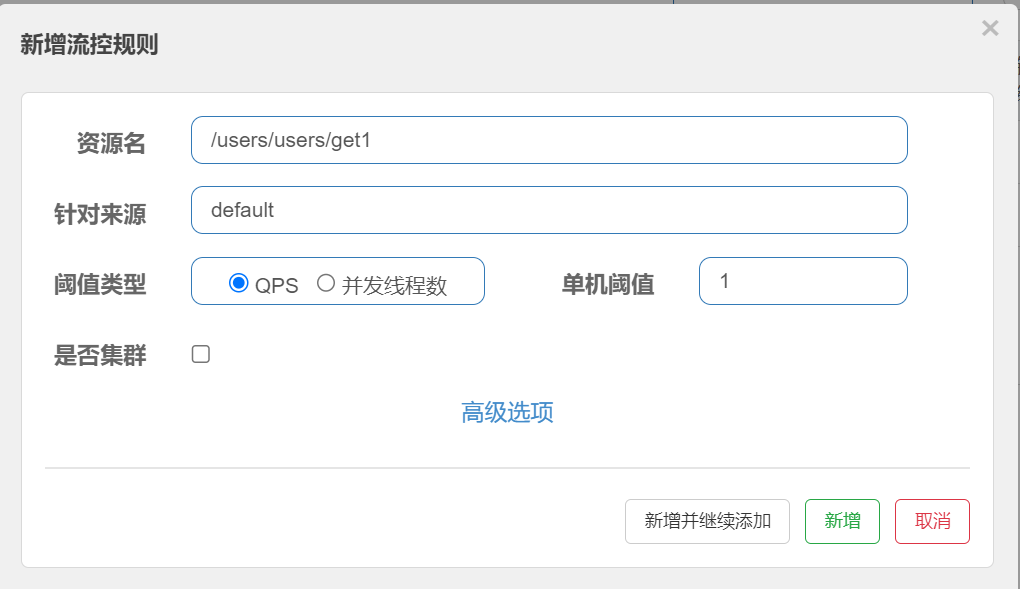






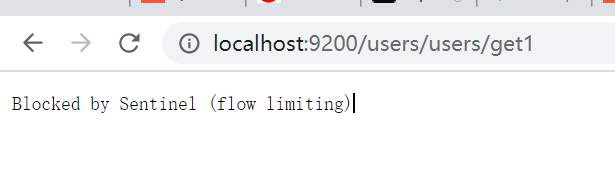


对这个请求进行流控，



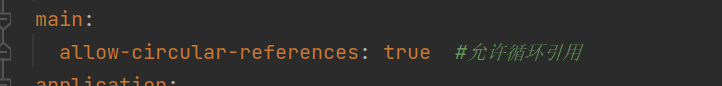
即每秒只允许访问一次



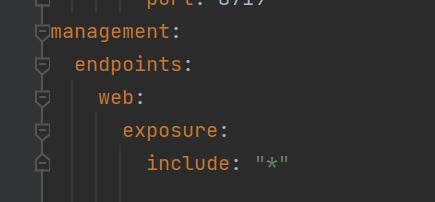


可以看到我们测试成功了， 成功的请求太多次成功的进行了拦截。

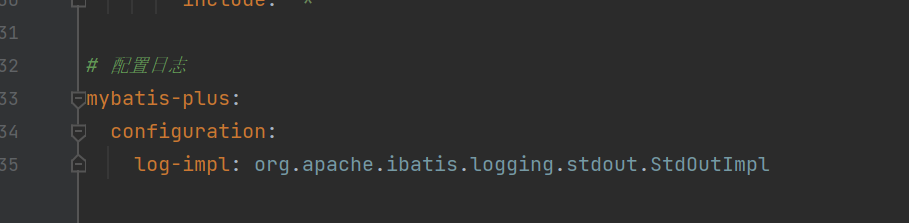
1. 接下来就需要验证是否可以对数据库的表进行操作
2. 最后前端建设就可以了。
3. **技术要点**
4. 版本问题，就是后端使用的技术太多， 有些技术和技术会有重叠的部分造成了循环调用的问题，这个时候你确定没有太大问题的话，就要在yml文件中添加这句



1. 监控问题， 当我们想要监控一个服务的时候， 要放开这个服务的监控权限， 使得外部的能够访问到服务的状态



1. 用了mybatis-plus调用其功能的时候不知道它具体是怎么访问数据库的时候， 我们就可以添加这个配置

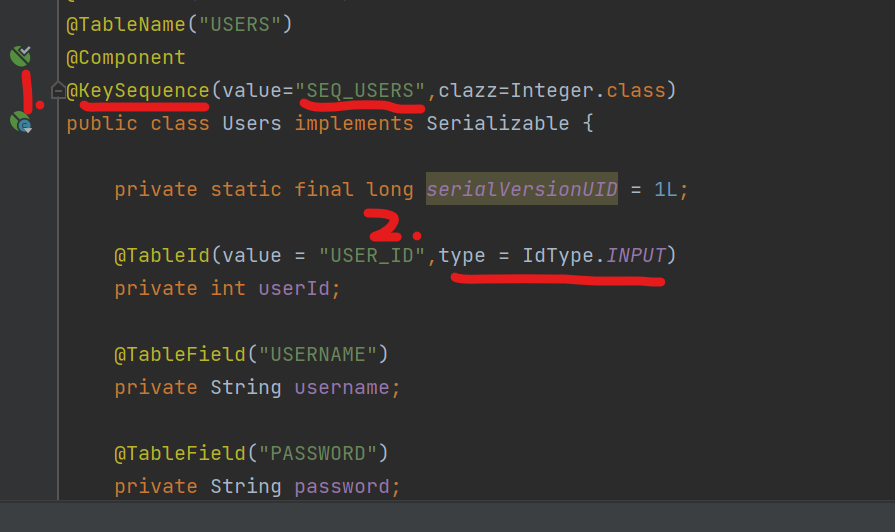


4.Oracle数据库中，无论表名还是字段名，都不区分大小写，默认都是大写，保持小写要加上双引号。所以创建表时，注意要严格区分大小写。

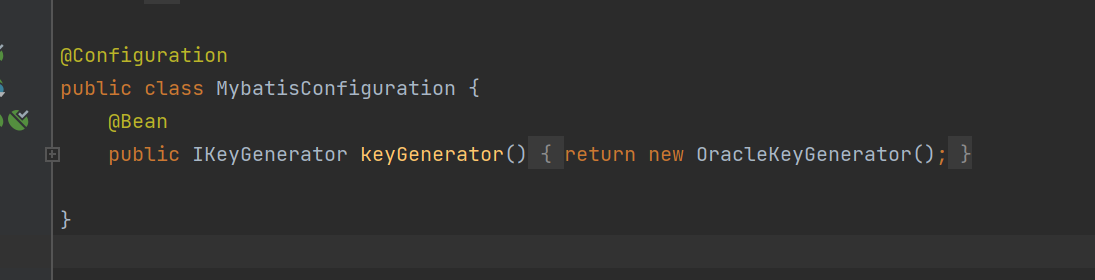
5.Oracle没有主键自增的功能，需要使用序列和触发器实现主键自增功能，部分代码如下所示。

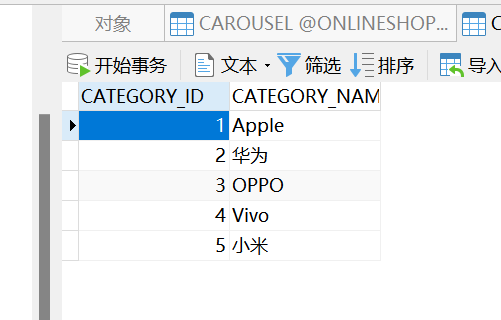


同时你在springboot这边的实体类要对应一个序列的名字，这样才能实现自增

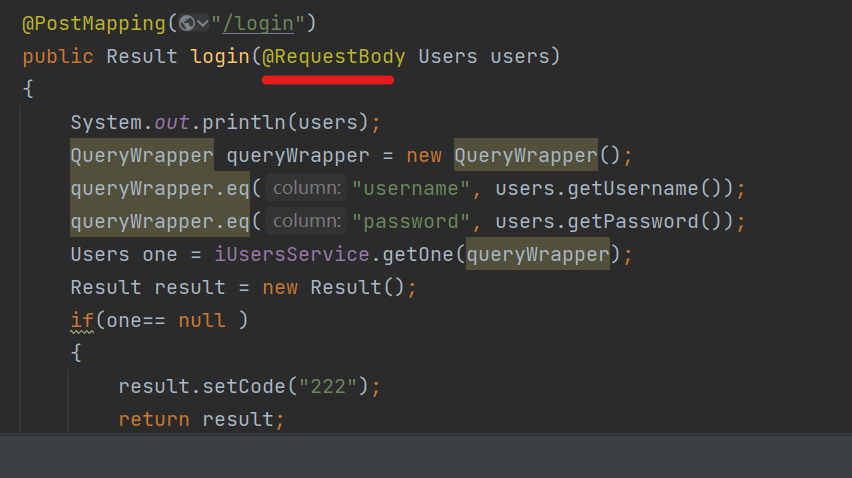


还要添加这个组件





6.springboot和vue进行交互的时候接受参数要加一个注解才能正确的接受数据



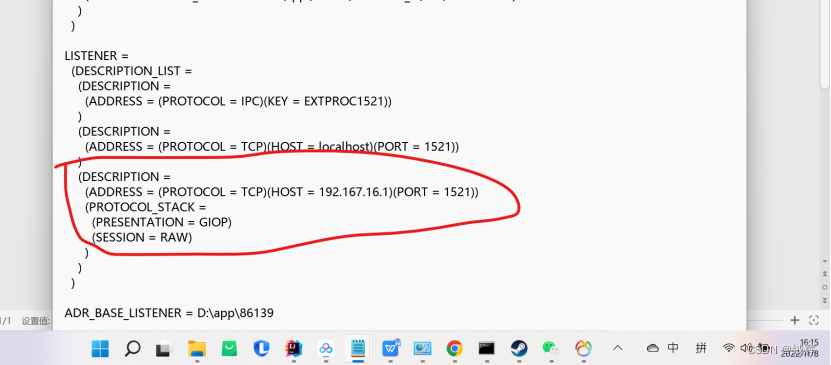
1. 在进行数据库远程连接的时候， 就比如同宿舍的下载了Oracle数据库我想用navicat 远程访问并直接进行使用的时候， 遇到了问题，解决方法如下

首先服务器端（有oracle数据库的一方）

（1）关闭防火墙（或者开放端口1521）

\*\*在防火墙--高级设置--入站规则--新建规则，可以新建规则将特定端口号打开\*\*

1. 在listener.ora文件中添加自己的ip地址（descributioN）



(3)确认自己的服务名， ip地址， 用户名， 密码

客户端（navicat）（自行下载-一个oracal的可视化操作平台）

https://www.oracle.com/database/technologies/instant-client/winx64-64-downloads.html

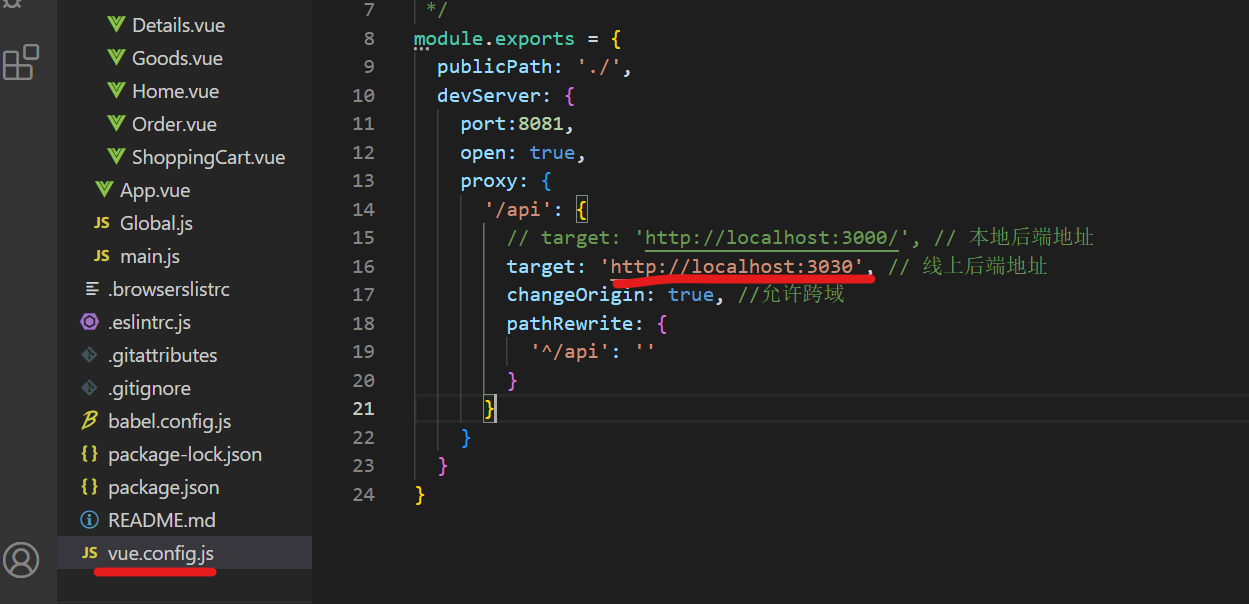
下载完navicat后将根据自己电脑的版本下载相应的客户端

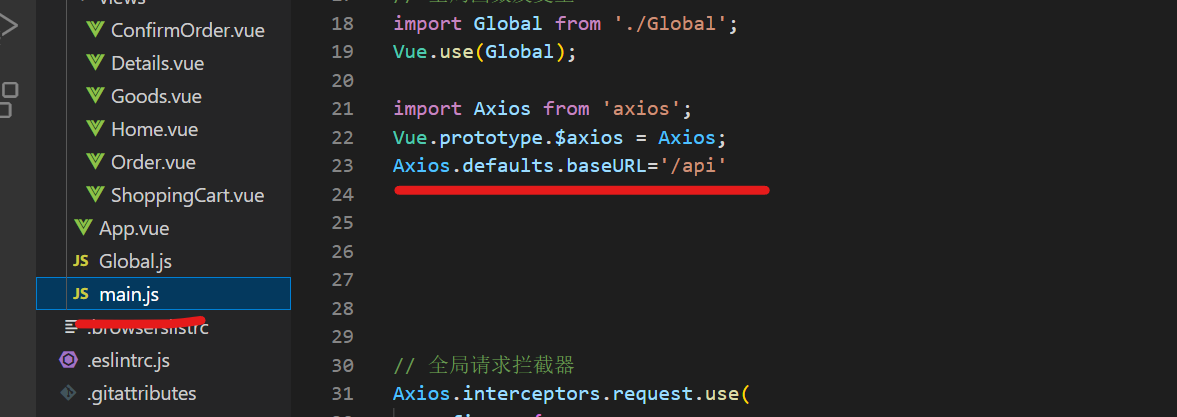
然后把（工具-选项-环境）

就好了

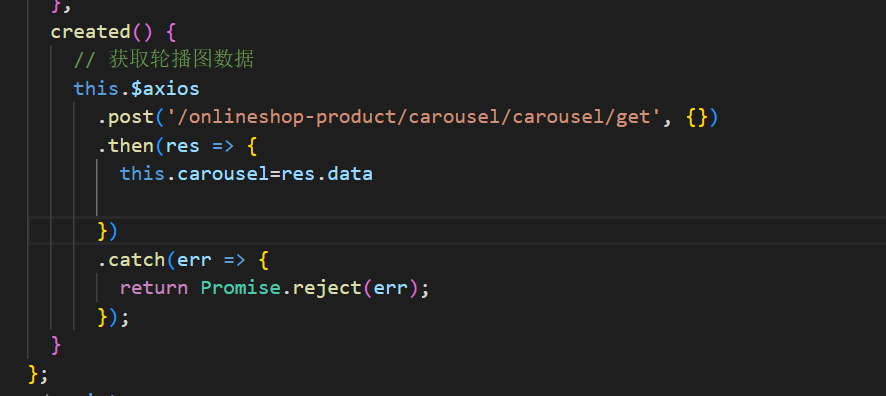
7在微服务架构整体技术选择上我们也遇到了一些选择上的问题

8.Vue进行页面设计的时候我们也遇到了怎么设置一个全局访问后端的前缀的问题时， 解决方式为





1. 在如何Vue如何访问后端上我们用的axios技术， 也是一大难关



1. **主要实现代码和运行界面**

用户的注册功能：我们使用的是mybatis-plus组件，以及oracle数据库，再完成注册前我们会查询数据库中是否存在有该用户，即用户名，运行成功的话，即用户注册完成。

核心代码：

public ResponseResult Register(String userName, String password) {

Users users=new Users();

users.setUsername(userName);

users.setPassword(password);

save(users);

return new ResponseResult("001","注册成功");

}

@Override

public ResponseResult FindUserName(String userName) {

QueryWrapper<Users> queryWrapper=new QueryWrapper<>();

queryWrapper.eq("userName",userName);

if (list(queryWrapper).get(0)!=null)

{

return new ResponseResult("004","用户名已经存在，不能注册");

}

return new ResponseResult("001","用户名不存在，可以注册");

}

用户登录功能：用户登录时候检查用户名和密码是否正确，正确即登录，错误即返回。

核心代码：

@Override

public Users Login(String userName, String password) {

QueryWrapper<Users>queryWrapper=new QueryWrapper<>();

queryWrapper.eq("userName",userName);

queryWrapper.eq("password",password);

return list(queryWrapper).get(0);

}

获取购物车列表功能：从数据库中查询购物车里面的信息，并且返回一个列表。

核心代码：

public List GetShoppingCart(Integer user\_id) {

QueryWrapper<ShoppingCart> queryWrapper=new QueryWrapper<>();

queryWrapper.eq("user\_id",user\_id);

return list(queryWrapper);

}

获取购物车某个详细商品：从数据库中查询购物车里面的某个商品。

核心代码：

@Override

public ShoppingCart FindShoppingCart(Integer user\_id, Integer product\_id) {

QueryWrapper<ShoppingCart>queryWrapper=new QueryWrapper<>();

queryWrapper.eq("user\_id",user\_id);

queryWrapper.eq("product\_id",product\_id);

return list(queryWrapper).get(0);

}

将商品加入购物车：接受前端传来的数据，将某个商品加入购物车。执行成功之后更新数据库信息。

核心代码：

@Override

public ShoppingCart AddShoppingCart(Integer user\_id, Integer product\_id) {

ShoppingCart shoppingcart=new ShoppingCart(user\_id,product\_id,1);

save(shoppingcart);

return shoppingcart;

}

更新购物车信息： 修改购物车某个商品的数量，执行成功之后更新数据库信息。

核心代码：

@Override

public ResponseResult UpdateShoppingCart(Integer NewNum, Integer user\_id, Integer product\_id) {

UpdateWrapper<ShoppingCart> updateWrapper=new UpdateWrapper<>();

updateWrapper.eq("user\_id",user\_id);

updateWrapper.eq("product\_id",product\_id);

updateWrapper.set("num",NewNum);

baseMapper.update(null,updateWrapper);

return new ResponseResult("001","修改购物车数量成功");

}

删除购物车信息：删除购物车之中的某个商品，成功之后删除数据库中的购物车信息。

核心代码：

@Override

public ResponseResult DeleteShoppingCart(Integer user\_id, Integer product\_id) {

QueryWrapper<ShoppingCart>queryWrapper=new QueryWrapper<>();

queryWrapper.eq("user\_id",user\_id);

queryWrapper.eq("product\_id",product\_id);

baseMapper.delete(queryWrapper);

return new ResponseResult("001","删除购物车成功");

}

获取所有商品：获取所有商品信息，执行成功之后返回数据给前端。

核心代码：

public List GetAllProduct(Integer currentPage, Integer pageSize) {

QueryWrapper<Product>queryWrapper=new QueryWrapper<>();

Page<Product> page=new Page<>();

page.setSize(pageSize);

page.setCurrent(currentPage);

page(page,queryWrapper);

return page.getRecords();

}

获取某个商品详细信息：执行成功之后返回商品详细数据给前端。

核心代码：

@Override

public List GetDetails(Integer productID) {

QueryWrapper<Product>queryWrapper=new QueryWrapper<>();

queryWrapper.eq(" product\_id",productID);

return list(queryWrapper);

}

模糊查询商品信息：执行成功之后返回模糊查询的列表。

核心代码：

@Override

public List GetProductBySearch(Integer currentPage, Integer pageSize, String search) {

QueryWrapper<Product>queryWrapper=new QueryWrapper<>();

Page<Product>page=new Page<>();

queryWrapper.like("product\_name",search).or().like("product\_intro",search);

page.setCurrent(currentPage);

page.setSize(pageSize);

page(page,queryWrapper);

return page.getRecords();

}

通过主键查询某个商品：执行成功后返回商品信息。

核心代码：

@Override

public Product GetProductById(Integer id) {

return baseMapper.selectById(id);

}

分类查询商品：返回某个商品的分列表给前端。

核心代码：

@Override

public List GetProductByCategory(List categoryId, Integer currentPage, Integer pageSize) {

QueryWrapper<Product>queryWrapper=new QueryWrapper<>();

for(Object i:categoryId)

{

queryWrapper.eq("category\_id",(Integer)i);

}

Page<Product>page=new Page<>();

page.setSize(pageSize);

page.setCurrent(currentPage);

page(page,queryWrapper);

return page.getRecords();

}

用户下单：增加订单信息，执行成功后更新数据库信息。

核心代码：

public ResponseResult AddOrder(Integer user\_id, ShoppingCartData shoppingCartData) {

Orders orders=new Orders();

Date currenDate=new Date();

orders.setOrderId(user\_id.toString()+currenDate.toString());

orders.setUserId(user\_id);

orders.setProductId(shoppingCartData.getProductID());

orders.setProductPrice(shoppingCartData.getPrice());

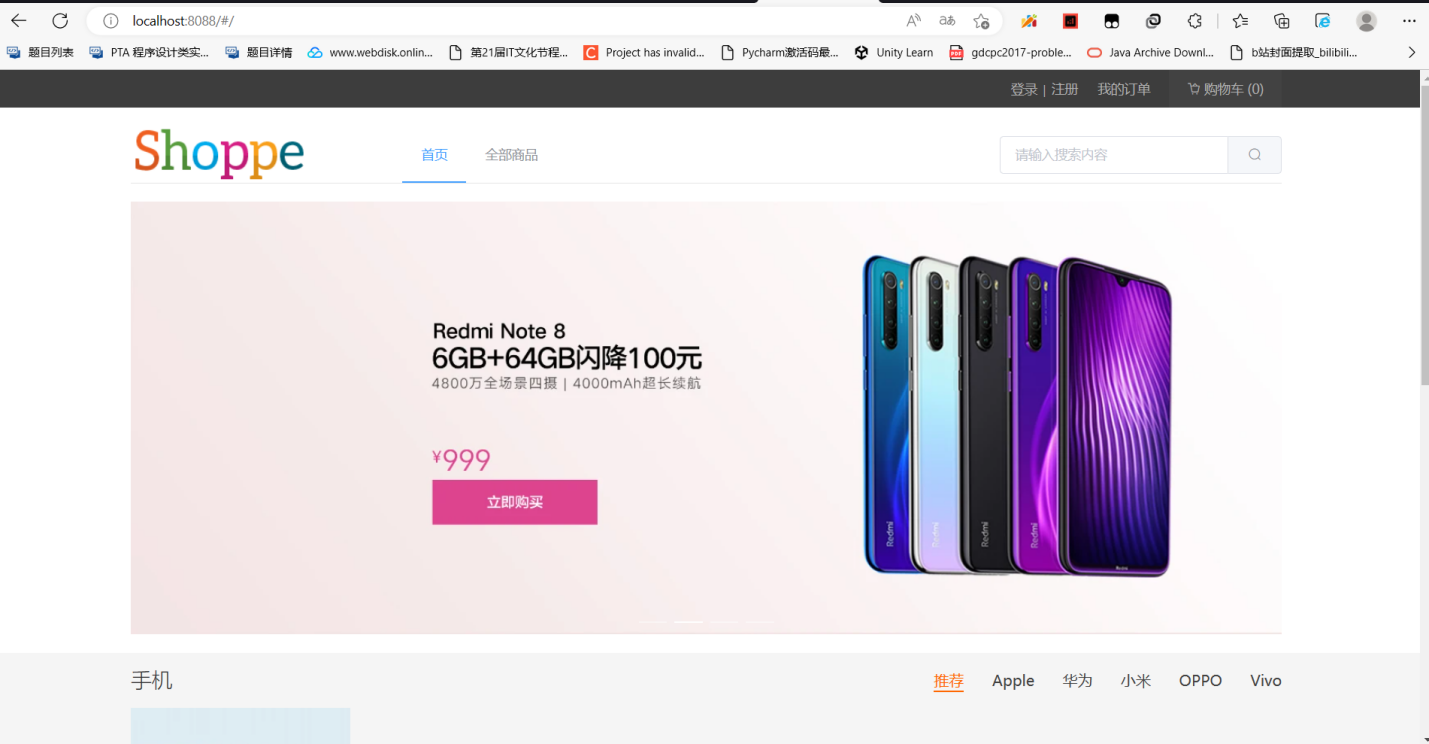
orders.setProductNum(shoppingCartData.getNum());

orders.setOrderTime(new Date());

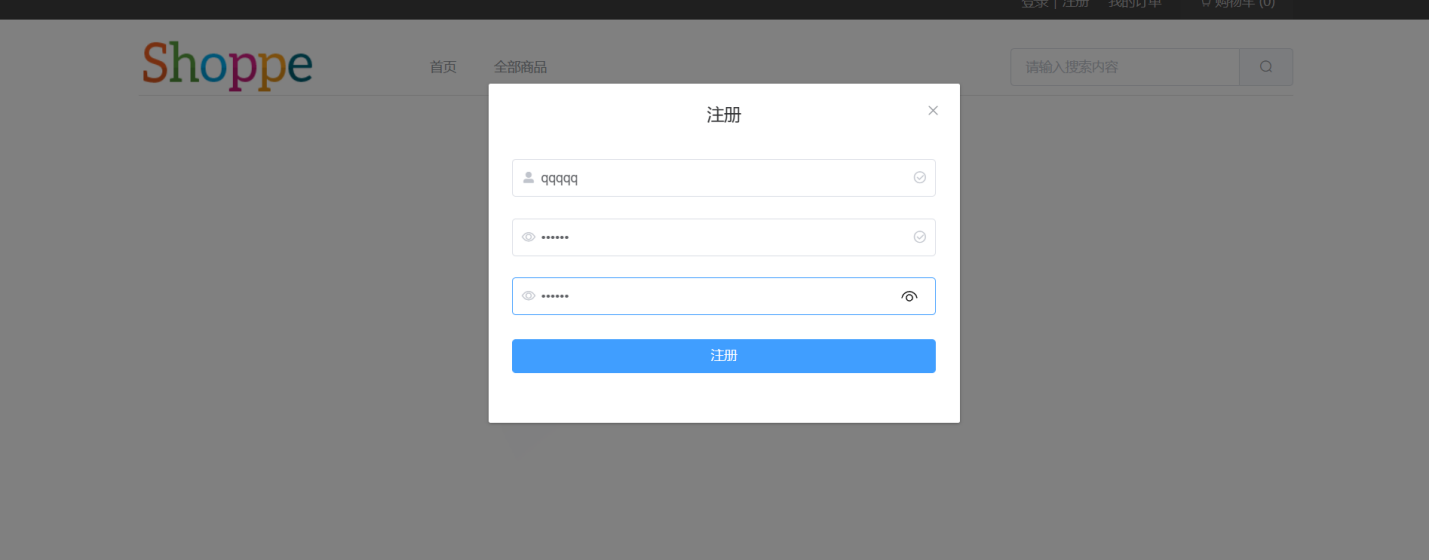
save(orders);

return new ResponseResult("001","购买成功");

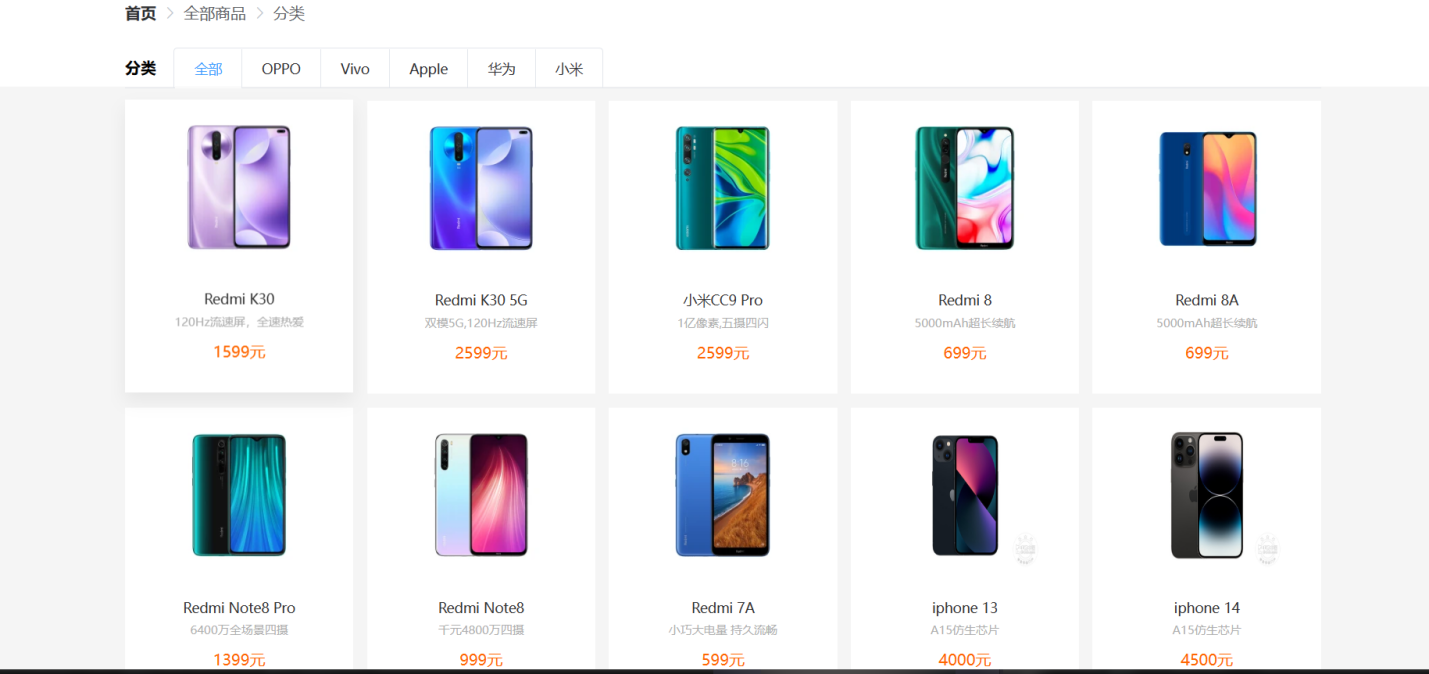
}



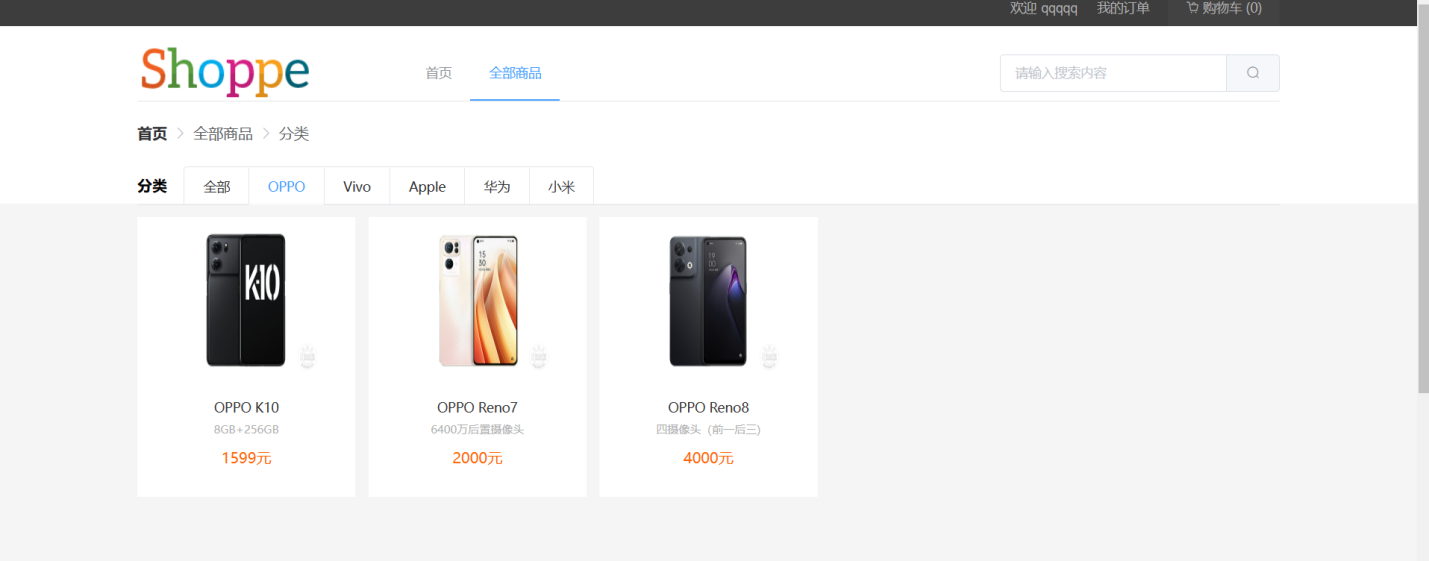
注册用户：



*登录之后页面显示：*



商品分类（oppo）：



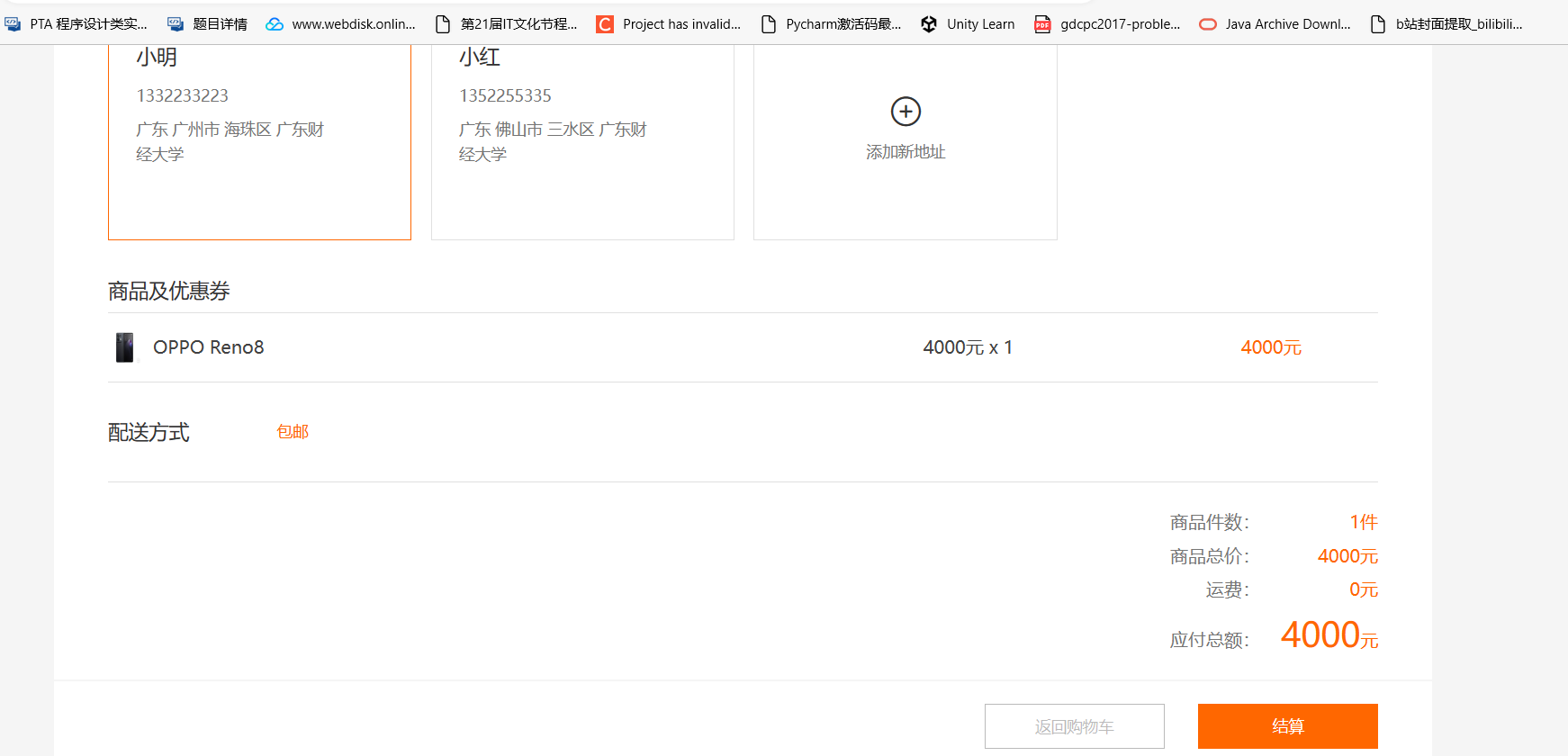
商品详细：

**

*加入购物车：*

**

*下单：*

**

**