[MyBatis中文网](https://mybatis.net.cn/index.html)

查文档！！！

目录

[项目结构 3](#_Toc96006069)

[一、MyBatis部分 4](#_Toc96006070)

[连接数据库配置 4](#_Toc96006071)

[操作数据库 4](#_Toc96006072)

[Mybatis三剑客 6](#_Toc96006073)

[Mybatis-generator的使用： 6](#_Toc96006074)

[Mybatis-plugin： 8](#_Toc96006075)

[Mybatis-PageHelper：实现分页 8](#_Toc96006076)

[二、用户模块 9](#_Toc96006077)

[注册模块 9](#_Toc96006078)

[service层 9](#_Toc96006079)

[controller层 10](#_Toc96006080)

[登录模块 13](#_Toc96006081)

[service层 13](#_Toc96006082)

[controller层 13](#_Toc96006083)

[单元测试 16](#_Toc96006084)

[三、分类模块 17](#_Toc96006085)

[service层 17](#_Toc96006086)

[controller层 19](#_Toc96006087)

[单元测试 19](#_Toc96006088)

[四、商品模块 20](#_Toc96006089)

[商品列表模块 20](#_Toc96006090)

[service层 20](#_Toc96006091)

[controller层 22](#_Toc96006092)

[商品详情模块 22](#_Toc96006093)

[service层 22](#_Toc96006094)

[controller层 23](#_Toc96006095)

[单元测试 23](#_Toc96006096)

[五、购物车模块 24](#_Toc96006097)

[表单统一验证 24](#_Toc96006098)

[增删改查购物车 24](#_Toc96006099)

[service层 24](#_Toc96006100)

[controller层 25](#_Toc96006101)

[单元测试 26](#_Toc96006102)

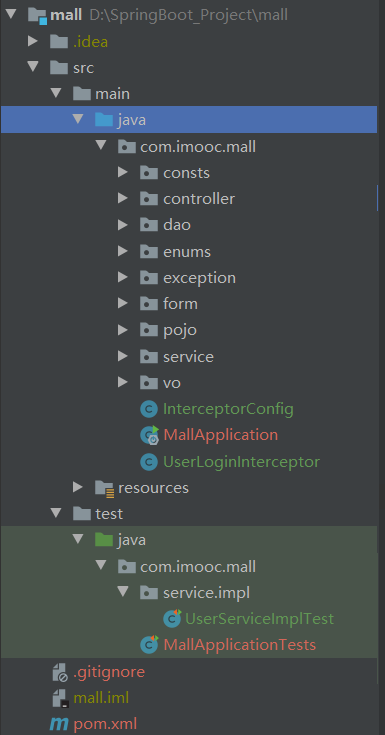
[六、收货地址模块 27](#_Toc96006103)

[service层 27](#_Toc96006104)

[Controller层 29](#_Toc96006105)

[单元测试 29](#_Toc96006106)

# 项目结构



controller层：前端页面接口控制层

service层：业务逻辑层，封装接口（用于规范约束）和实现类

dao层：数据库操作层

pojo层：数据库对应的实体类层

enums层：封装一些枚举对象，便于代码的维护

vo层：view object层，用于后端打包前端所需要的数据

form层：前端向controller层提交的表单

exception层：错误处理层

consts层：储存静态常量

UserLoginInterceptior类：用户登录拦截器

InterceptorConfig类：用户拦截器使用配置

test包：运行service模块单测的地方

# 一、MyBatis部分

### 连接数据库配置

Springboot连接数据库配置：application.yml

spring:  
 datasource:  
 driver-class-name: com.mysql.cj.jdbc.Driver  
 username: root  
 password: root  
 url: jdbc:mysql://127.0.0.1:3306/mall?characterEncoding=utf-8&useSSL=false&serverTimezone=UTC

并增加连接数据库的依赖：（不需要填写版本号）

<dependency>  
 <groupId>mysql</groupId>  
 <artifactId>mysql-connector-java</artifactId>  
</dependency>

### 操作数据库

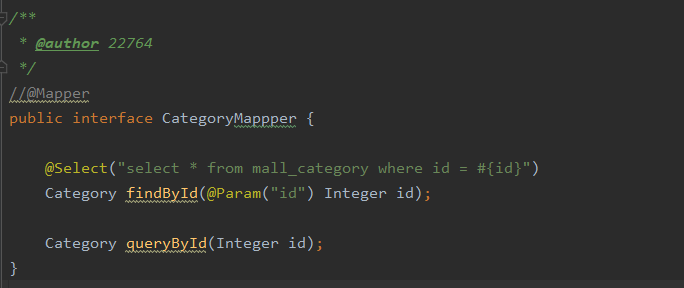
1、pojo包中新建实体类对象Category，并添加@Data注解，变量类型和数据库保持一致，用驼峰命名规则，并设为private

Lombok插件：通过@Data注解可自动获取get、set方法和toString方法，减少代码量

导入依赖：

<dependency>  
 <groupId>org.projectlombok</groupId>  
 <artifactId>lombok</artifactId>  
</dependency>

2、dao包中新建CategoryMapper类用于操作数据库



Mybatis的两种使用方法：注解 和 xml

* 上图中三个注解实现注解方式

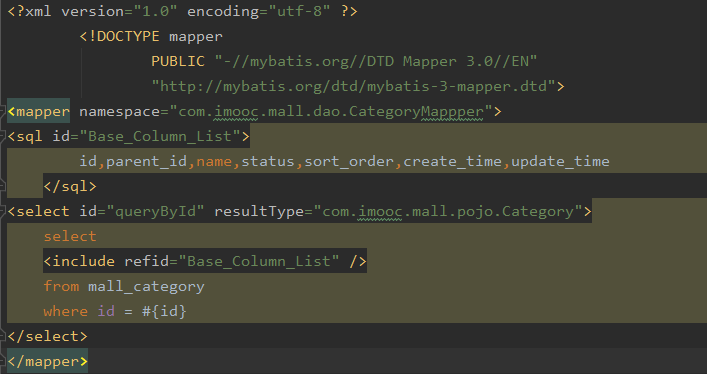
@Mapper：通过注解的方式，由于在每个对数据库的操作类上都添加@Mapper比较麻烦，因此可以在启动类上添加注解@MapperScan(basePackages = "com.imooc.mall.dao")，里面参数为扫描的包

@Select：在其中写sql语句完成对数据库的操作

@Param：传入参数

* 第二个方法实现xml方式

在项目的resources目录下新建mapper包，包下新建CategoryMapper.xml用于实现xml方式



第一、二个标签内容固定

<mapper>标签的namespace用于表示哪个包中的类用此xml操作数据库

<sql>标签表示表中字段，用id唯一标识，并在后面用<include>使用

<select>标签中写方法对应的sql语句，id表示方法名，result Type表示返回值类型

3、测试数据

测试之前增加配置：（注意xml格式）

mybatis:  
 configuration:  
 map-underscore-to-camel-case: true  
 mapper-locations: classpath:mapper/\*.xml

第一条用于消除数据库字段存在下划线而java命名规则使用驼峰的差异

第二条用于告知使用xml方法时xml文件的位置

走单测放在test中，右键-goto-test-新建

测试类必须拥有这两个注解：

@RunWith(SpringRunner.class)  
@SpringBootTest

## Mybatis三剑客

Mybatis-generator

Mybatis-plugin

Mybatis-PageHelper

### Mybatis-generator的使用：

自动生成xml

开发生成器：连接数据库>获取表结构>生成文件

1、pom.xml中导入插件

<plugin>  
 <groupId>org.mybatis.generator</groupId>  
 <artifactId>mybatis-generator-maven-plugin</artifactId>  
 <version>1.3.7</version>  
  
 <configuration>  
 <overwrite>true</overwrite>  
 </configuration>  
</plugin>

<configuration>标签表示重新自动生成文件时自动覆盖

2、新建配置文件generatorConfig.xml

在resources目录下新建generatorConfig.xml，内容如下：

<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE generatorConfiguration

PUBLIC "-//mybatis.org//DTD MyBatis Generator Configuration 1.0//EN"

"http://mybatis.org/dtd/mybatis-generator-config\_1\_0.dtd">

<generatorConfiguration>

<!-- 配置mysql驱动地址-->

<classPathEntry location="D:\SpringBoot\_Project\mysql-connector-java-5.1.6.jar" />

<context id="DB2Tables" targetRuntime="MyBatis3">

<!-- 不再追加xml内容-->

<plugin type="org.mybatis.generator.plugins.UnmergeableXmlMappersPlugin" />

<commentGenerator>

<property name="suppressAllComments" value="true"/>

</commentGenerator>

<jdbcConnection driverClass="com.mysql.jdbc.Driver"

connectionURL="jdbc:mysql://127.0.0.1:3306/mall?characterEncoding=utf-8&amp;useSSL=false&amp;serverTimezone=UTC"

userId="root"

password="root">

</jdbcConnection>

<javaTypeResolver >

<property name="forceBigDecimals" value="false" />

</javaTypeResolver>

<javaModelGenerator targetPackage="com.imooc.mall.pojo" targetProject="src/main/java">

<property name="enableSubPackages" value="true" />

<!-- <property name="trimStrings" value="true" />-->

</javaModelGenerator>

<sqlMapGenerator targetPackage="mapper" targetProject="src/main/resources">

<property name="enableSubPackages" value="true" />

</sqlMapGenerator>

<javaClientGenerator type="XMLMAPPER" targetPackage="com.imooc.mall.dao" targetProject="src/main/java">

<property name="enableSubPackages" value="true" />

</javaClientGenerator>

<!-- 生成数据库中的表所对应的实体类、dao中的类、mapper文件，并删除一些不必要生成的字段-->

<table tableName="mall\_order" domainObjectName="Order" enableCountByExample="false" enableDeleteByExample="false" enableSelectByExample="false" enableUpdateByExample="false"/>

<table tableName="mall\_order\_item" domainObjectName="OrderItem" enableCountByExample="false" enableDeleteByExample="false" enableSelectByExample="false" enableUpdateByExample="false"/>

</context>

</generatorConfiguration>

具体内容含义可查看文档：<https://www.imooc.com/article/21444>

3、执行：

Terminal执行命令

mvn mybatis-generator:generate

### Mybatis-plugin：

一个插件…

### Mybatis-PageHelper：实现分页

[GitHub - pagehelper/Mybatis-PageHelper: Mybatis通用分页插件](https://github.com/pagehelper/Mybatis-PageHelper)

<!-- https://mvnrepository.com/artifact/com.github.pagehelper/pagehelper-spring-boot-starter -->

<dependency>

<groupId>com.github.pagehelper</groupId>

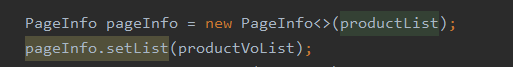
<artifactId>pagehelper-spring-boot-starter</artifactId>

<version>1.2.13</version>

</dependency>

分页信息：

包里有个PageInfo对象，传参为数据库数据列表，还要设置一个参数setList为Vo对象



-------------------------------------------------------------------------------

# 二、用户模块

Content-Type: application/json

开发顺序: Dao(操作数据库) -> Service(业务逻辑) -> Controller(数据组装，返回前端)

单元测试：Service层

Mybatis打印SQL语句（用于debug）

## 注册模块

### service层

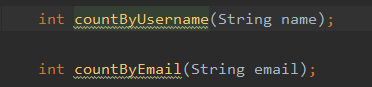
1、编写业务逻辑：

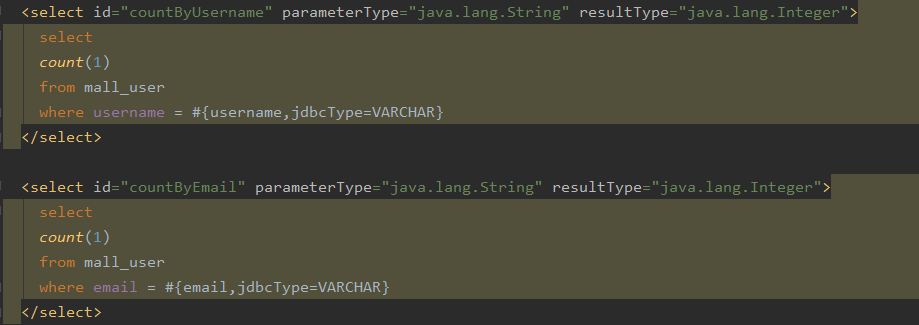
username、email不能重复，用户的role默认设定为1（customer），密码通过Spring自带的MD5摘要算法加密，最后存入数据库；

若username或email重复，则返回对应的错误信息和错误代码，通过枚举类ResponseEnum表示错误信息，通过ResponseVo返回信息

2、判断重复：

通过给UserMapper添加方法





3、注解

Service接口实现类要加上注解@Service

ResponseEnum类加上注解@Getter表示get方法

ResponseVo类加上注解@Data表示get、set方法，@JsonInclude(value = JsonInclude.Include.NON\_NULL)，表示不返回null类型的字段

4、其他细节

插入数据库时返回一个int类型的值，为0表示插入失败，为1表示插入成功，获取该值以判断是否插入成功

MD5算法：

//MD5摘要算法(Spring自带),Digest(摘要)  
String s = DigestUtils.*md5DigestAsHex*(  
 user.getPassword().getBytes(StandardCharsets.*UTF\_8*));  
user.setPassword(s);

5、service单测

测试时在测试类上添加注解@Transactional，以避免测试数据对数据库造成污染

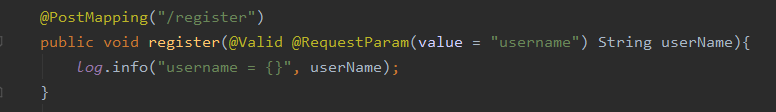
### **controller**层

1、controller基本逻辑

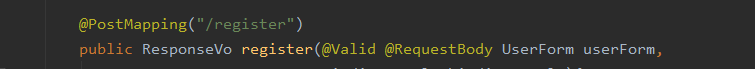
从规定好的接口获取前端提交的数据，对数据进行校验，若校验不通过则返回校验错误信息，否则调用service层方法，将信息写入数据库

2、controller接收参数的两种方法

@RequestParam用来处理x-www-form-urlencoded编码的内容



@RequestBody用来处理Json类型的数据



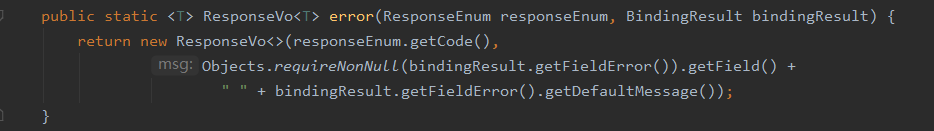
3、验证参数是否为空

1）、定义的接收前端提交表单数据的类UserForm中各个字段添加注解@NotBlank，表示不能为空



2）、添加传入参数BindingResult bindingResult，此参数与注解@NotBlank绑定，若被注解的变量有空，则存在错误，对bindingResult进行错误判断，并返回错误信息数据





4、注解

Controller类添加注解@RestController，作用等同于@Controller + @ResponseBody，表明是一个控制器类

Controller类添加注解@RequestMapping(“/user”)，访问路径

Controller类添加注解@Slf4j，可以打印日志信息

方法上添加注解@PostMapping(“/register”)，前端的访问路径，方法为post

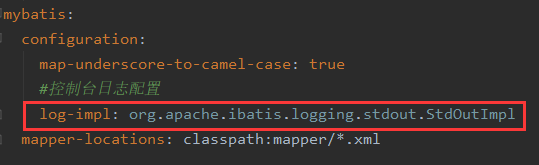
方法参数上添加注解@Valid，实现对表单的校验

@NotBlank 用于String，判断空格(不能为空格)

@NotNull 用于判断是否为Null

@NotEmpty 用于集合（数组、集合等是否为空）

5、控制台输出sql语句，添加配置



6、其他细节

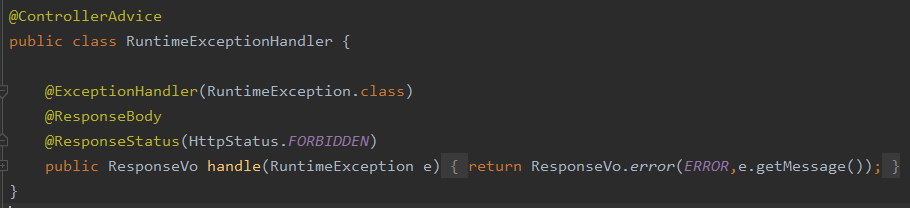
若前端表单提供参数和数据库字段不一致时，定义form层，在form层中定义接收参数的表单实体类UserForm，并利用对象间拷贝的方法

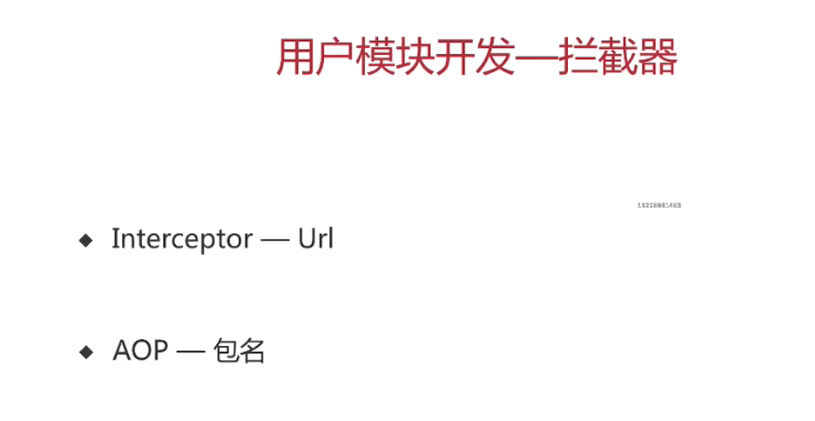
BeanUtils.*copyProperties*(userForm,user);

尽量保持controller层代码清晰简洁

返回固定格式的服务器端错误：







## 登录模块

### service层

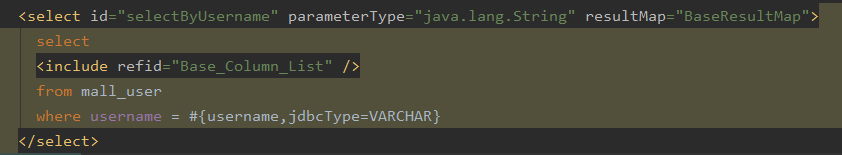
1. 编写业务逻辑

根据用户名从数据库中查找用户，若用户不存在或密码匹配失败，则返回用户名或密码错误，否则返回登录成功信息

1. 判断用户是否存在

给UserMapper添加方法





3、其他细节

返回登录成功信息时要将user变量的password属性设置为空，密码比较时需要忽略大小写，.equalsIgnoreCase，

### controller层

1、基本逻辑

登录：获取前端输入的用户名和密码，判断有无空值，若空则返回错误信息，否则调用userService的login方法登录，并获取ResponseVo对象；登录成功后设置session

获取登录信息：从session中拿到登录信息并返回用户信息

登出：从session中删除登录信息并返回成功

拦截器： 设置拦截器拦截未登录用户的访问权限

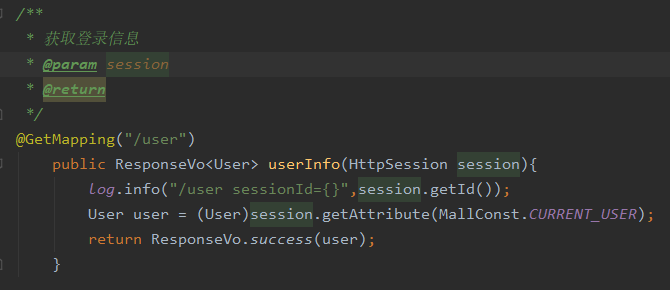
2、session相关操作

1）登陆时设置session：

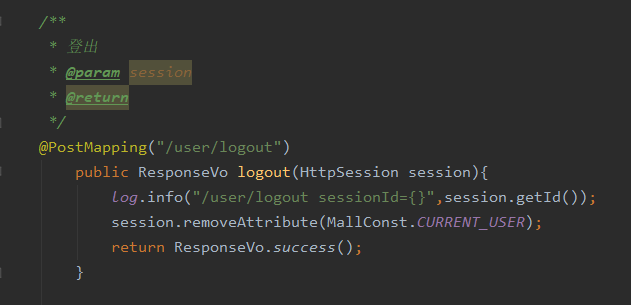


硬编码改为使用常量

2）获取登录信息时获取session



3）登出时删除session

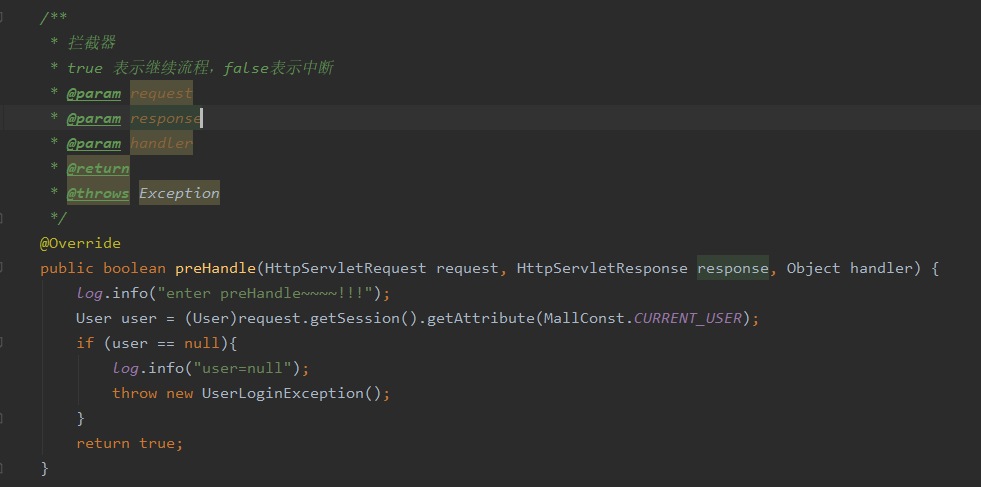


3、拦截器的设置和配置

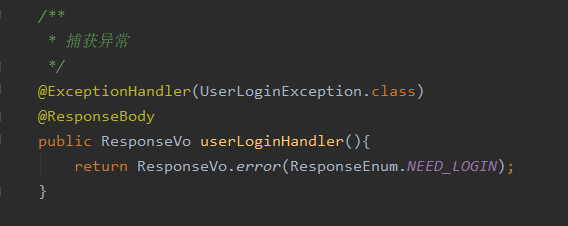
拦截器继承接口HandlerInterceptior，重写preHandle方法。

拦截器返回true表示继续流程，false表示中断

由于未登录不能直接返回false，但拦截器只能返回true或false，因此当未登录被拦截时可抛出异常，交给统一异常处理类处理

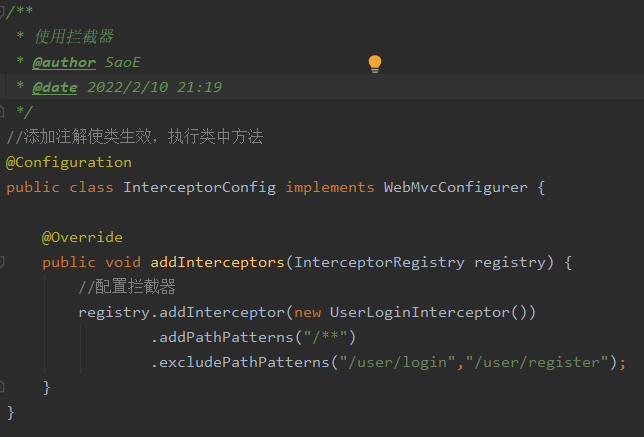


统一异常处理类捕捉到异常过后，返回所需返回的数据



配置拦截器：

继承接口WebMvcConfigurer，重写addInterceptors方法，并添加拦截器，设置拦截的页面url

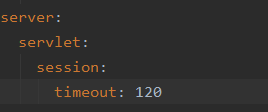


4、注解：

拦截器配置中的@Configuration注解使得配置类生效，执行类中方法

5、其他细节：

配置文件中可配置session的有效时长，默认为30分钟，最低为1分钟



单位为秒

6、cookie和session //TODO

Cookie特性：Cookie跨域

Session和cookie的安全性问题

## 单元测试

mvn clean package

mvn clean package -Dmaven.test.skip=true



# 三、分类模块

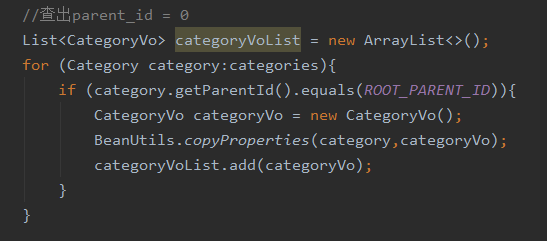
### service层

1、基本逻辑

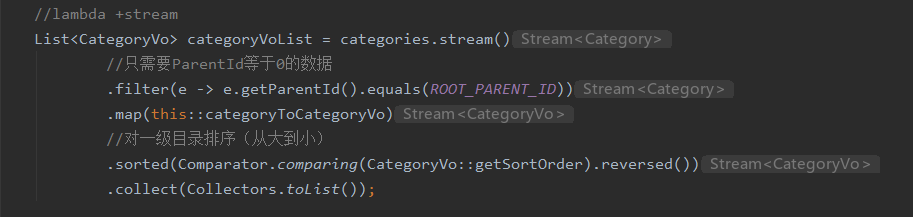
先从数据库表中查出全部商品信息，再检索出根目录同时排序，再依次查询子目录并返回相关内容

2、查询根目录并排序

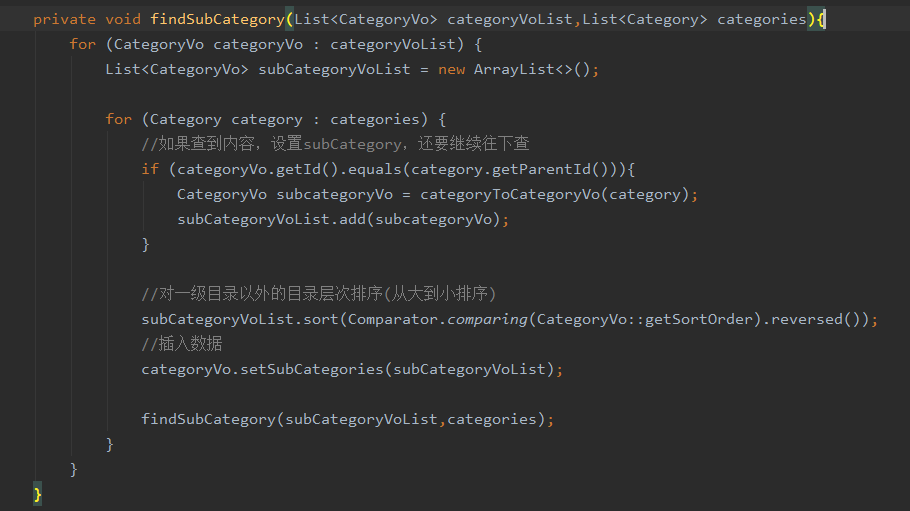
使用for循环遍历：



使用lambda＋stream



3、查询子目录并排序



采用递归的方式完成，基本逻辑为：遍历categoryVoList列表中的每一项，若categories中存在一项的parentId等于categoryVoList列表中的某一项的id，则将其添加到临时子类中，排序后递归调用，查找多级目录的子项，最后添加进根目录的子项中

4、其他细节

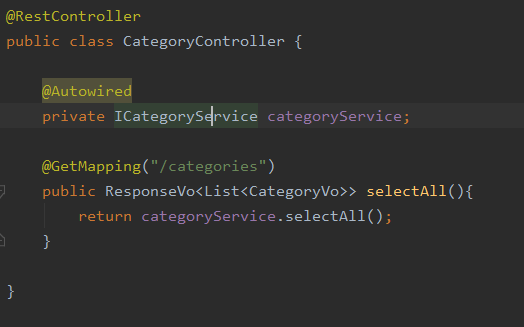
在进行分类模块实现时，要先将数据库商品表中全部数据一次查出，再进行相关操作，而不是一次次到数据库里去查找，这样做可以减少耗时。

耗时：http（请求微信Api）> 磁盘 > 内存，MySQL查找通过内网＋磁盘

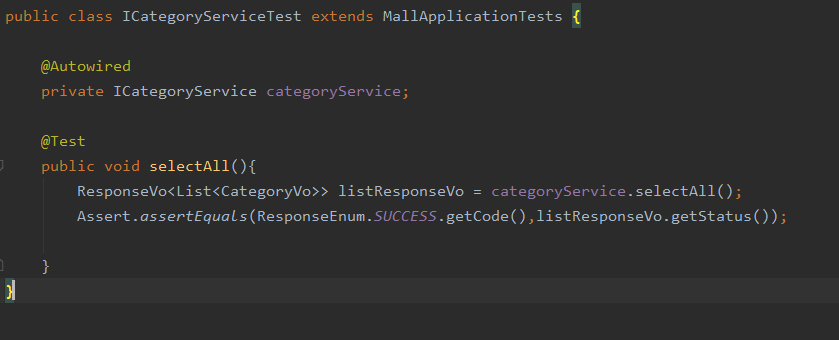
5、lambda+stream

//TODO

### controller层



## 单元测试



：

# 四、商品模块

开发顺序：先生成数据库对象-》新建Vo -》写service逻辑 -》单测 （-》单侧通过写返回值）-》写controller

## 商品列表模块

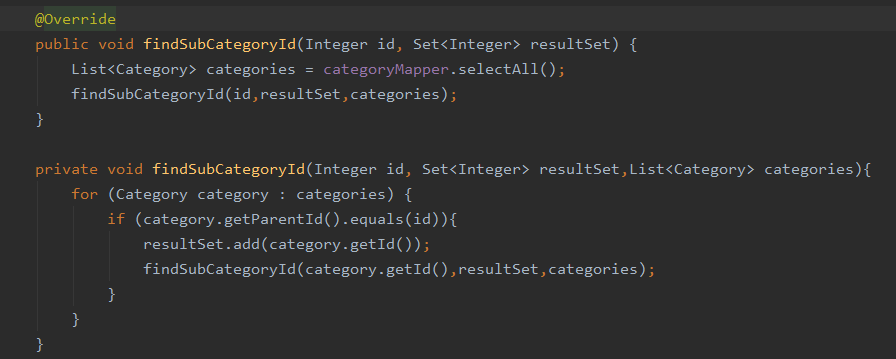
### service层

1、基本逻辑

首先获取根目录商品、子目录商品的id集合，在通过id集合查找商品列表，返回对应数据，显示时实现分页

2、开发工作

1）开发CategoryService接口的findSubCategoryId方法



封装一个同名方法，防止每次遍历都查一遍数据库

2）生成数据库对象

<table tableName="mall\_product" domainObjectName="Product" enableCountByExample="false" enableDeleteByExample="false" enableSelectByExample="false" enableUpdateByExample="false">  
 <columnOverride column="detail" jdbcType="VARCHAR" />  
 <columnOverride column="sub\_images" jdbcType="VARCHAR" />  
</table>

将detail和sub\_images字段类型设置为VARCHAR

3）新建操作数据库的selectByCategoryIdSet方法

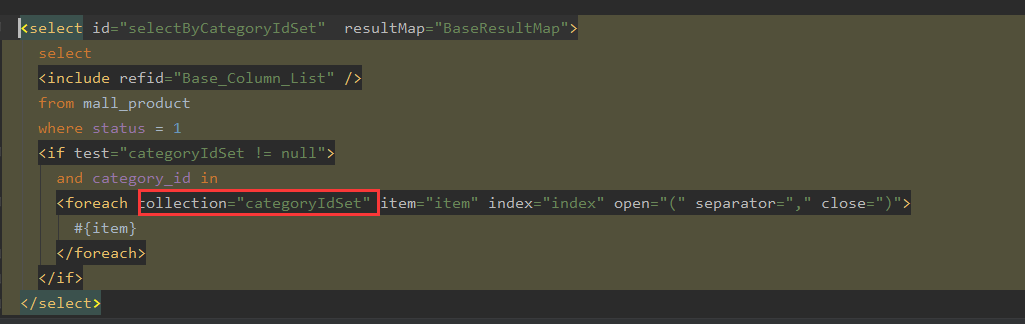




Debug:







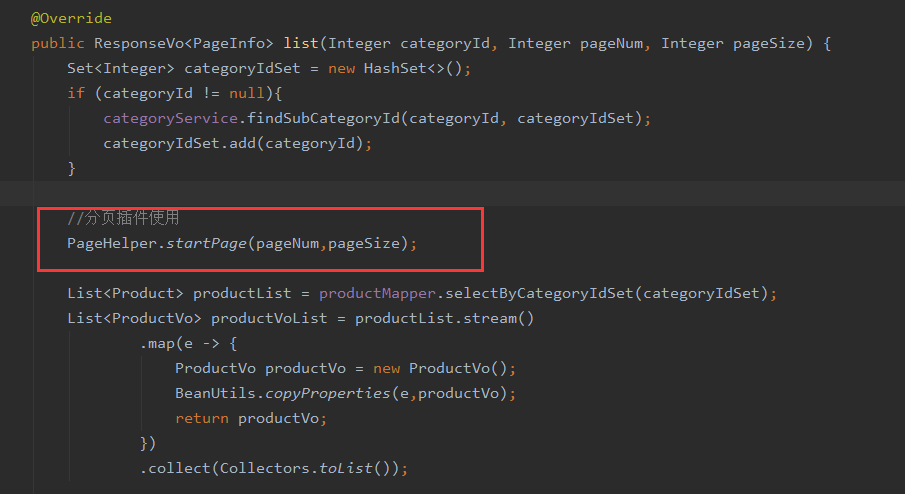
一旦方法传入的参数不是基本类型，那么需要在参数前加注解@Param(“”)其中参数为参数名，sql语句中的collection也要改为参数名

要主义test中语句size() > 0，而不能用!=null，

4）显示分页

添加分页插件依赖

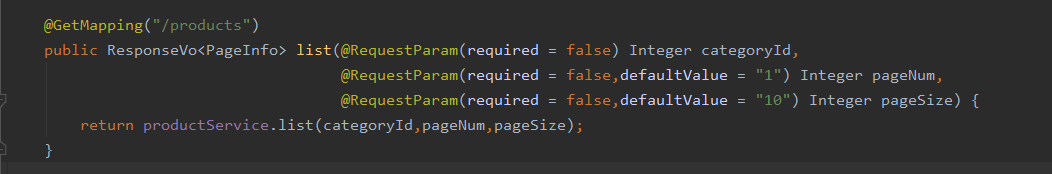
<!-- https://mvnrepository.com/artifact/com.github.pagehelper/pagehelper-spring-boot-starter -->  
<dependency>  
 <groupId>com.github.pagehelper</groupId>  
 <artifactId>pagehelper-spring-boot-starter</artifactId>  
 <version>1.2.13</version>  
</dependency>



就此一行代码

### controller层

1. 基本逻辑：传参调用
2. 注解



Get请求的参数形式：localhost:8080/products?pageNum=1&pageSize=2

接收这样形式的参数时，使用注解@RequestParam

参数required = false表示不是必传的

defaultValue =““表示默认值，此处只能用字符形式

## 商品详情模块

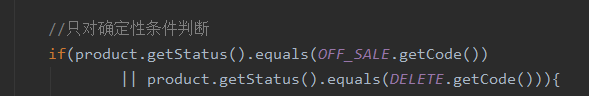
### service层

基本逻辑

根据传入productId，调用方法查找商品，如果商品删除或下架，则返回错误信息，否则返回所需的vo对象数据

其他细节

1）判断商品是否查到时，只对确定性条件判断



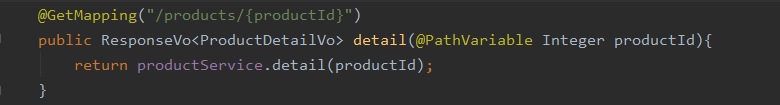
2）返回数据时要对敏感字段进行处理，例如库存



### controller层

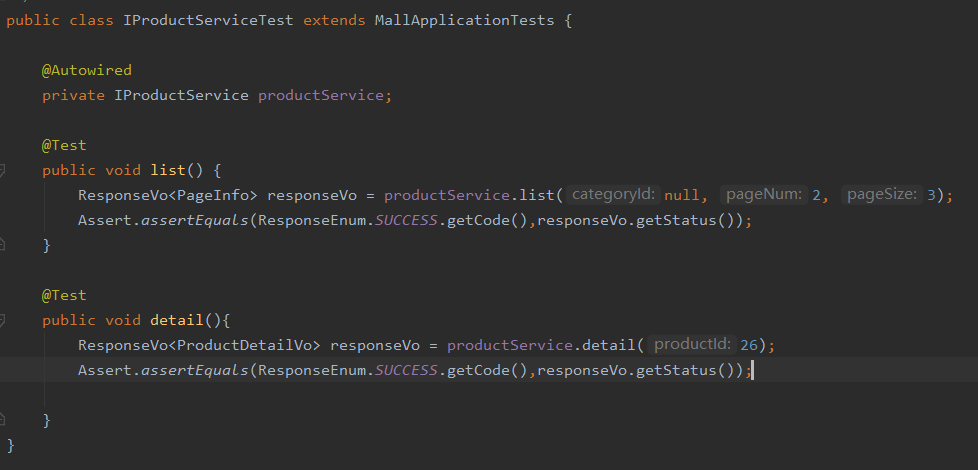
1、基本逻辑：传参调用

2、注解



参数包含在在url里时，用@PathVariable拿到参数

## 单元测试



# 五、购物车模块

## 表单统一验证

对于接收前端数据时的参数为空问题，根据他会报出异常

MethodArgumentNotValidException

因此我们可以在统一异常处理类中统一捕获该异常，并返回所对应的异常信息这样就避免了每写一个方法都要验证一次的重复工作

@ExceptionHandler(MethodArgumentNotValidException.class)  
@ResponseBody  
public ResponseVo notValidExceptionHandle(MethodArgumentNotValidException e){  
 BindingResult bindingResult = e.getBindingResult();  
 Objects.*requireNonNull*(bindingResult.getFieldError());  
 return ResponseVo.*error*(ResponseEnum.*PARAM\_ERROR*,  
 bindingResult.getFieldError().getField() +" " + bindingResult.getFieldError().getDefaultMessage());  
}

## 增删改查购物车

### service层

1）Service层基本逻辑：

添加商品到购物车：

根据传入的增添商品的表单获取商品id，查找数据库中是否有此id的商品，若不存在，则返回商品不存在异常，若存在，则判断商品是否正常在售且库存是否充足，若有异常则返回对应异常信息，若均验证通过，则写入redis购物车。写入时若没有该商品，则新增，若有该商品则数量加1。

获取购物车商品列表：

通过redisKey拿到购物车里全部的信息，遍历信息，拿到每一条信息的productId，在数据库中进行查询，最后对数据进行封装并返回

更新购物车商品：

可更新商品的数量和选中状态。通过传入更新信息的表单和productId，通过productId查询购物车中是否有该数据，若没有则返回错误，若存在则根据表单内容进行修改，修改完成后重新put进redis，最后返回所需数据

删除购物车商品：

通过传入productId，查询购物车中是否存在该商品，若没有该商品则报错，若存在，则删除，最后返回结果

全选商品和全部选商品：

先根据uid查询购物车中全部商品，遍历所有商品，设置每一项的选中状态为true或false，完成后写回数据库，最后返回结果

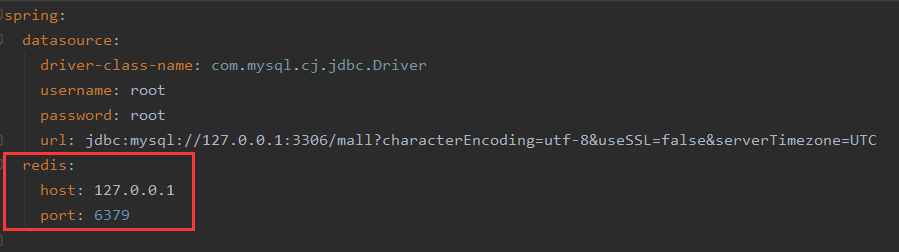
购物车商品总数：

查询购物车给中全部商品，将商品的数量累计求和并返回结果

2）辅助工具

Redis

<dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-data-redis</artifactId>  
</dependency>



Gson

<!--也可以用fastjson-->  
<dependency>  
 <groupId>com.google.code.gson</groupId>  
 <artifactId>gson</artifactId>  
</dependency>

### controller层

各接口传参调方法即可，须交验的表单信息在参数前添加注解@Valid，在类中变量前添加注解@Notxxx，并注意配置拦截器

## 单元测试

利用Assert语句判断

@Slf4j

public class ICartServiceTest extends MallApplicationTests {

@Autowired

private ICartService cartService;

private Gson gson = new GsonBuilder().setPrettyPrinting().create();

private Integer productId = 26;

private Integer uid = 1;

@Before

public void add() {

log.info("【新增购物车。。。】");

CartAddForm cartAddForm = new CartAddForm();

cartAddForm.setProductId(productId);

cartAddForm.setSelected(true);

ResponseVo<CartVo> responseVo = cartService.add(uid, cartAddForm);

log.info("list={}", gson.toJson(responseVo));

Assert.assertEquals(ResponseEnum.SUCCESS.getCode(),responseVo.getStatus());

}

@Test

public void list(){

ResponseVo<CartVo> responseVo = cartService.list(uid);

log.info("list={}", gson.toJson(responseVo));

Assert.assertEquals(ResponseEnum.SUCCESS.getCode(),responseVo.getStatus());

}

@Test

public void update(){

CartUpdateForm form = new CartUpdateForm();

form.setQuantity(15);

form.setSelected(true);

ResponseVo<CartVo> responseVo = cartService.update(uid, productId, form);

log.info("result={}",gson.toJson(responseVo));

Assert.assertEquals(ResponseEnum.SUCCESS.getCode(),responseVo.getStatus());

}

@After

public void delete(){

log.info("【删除购物车。。。】");

ResponseVo<CartVo> responseVo = cartService.delete(uid, productId);

log.info("result={}",gson.toJson(responseVo));

Assert.assertEquals(ResponseEnum.SUCCESS.getCode(),responseVo.getStatus());

}

@Test

public void selectAll(){

ResponseVo<CartVo> responseVo = cartService.selectAll(uid);

log.info("result={}",gson.toJson(responseVo));

Assert.assertEquals(ResponseEnum.SUCCESS.getCode(),responseVo.getStatus());

}

@Test

public void unSelectAll(){

ResponseVo<CartVo> responseVo = cartService.unSelectAll(uid);

log.info("result={}",gson.toJson(responseVo));

Assert.assertEquals(ResponseEnum.SUCCESS.getCode(),responseVo.getStatus());

}

@Test

public void sum(){

ResponseVo<Integer> responseVo = cartService.sum(1);

log.info("result={}",gson.toJson(responseVo));

Assert.assertEquals(ResponseEnum.SUCCESS.getCode(),responseVo.getStatus());

}

}

# 六、收货地址模块

## service层

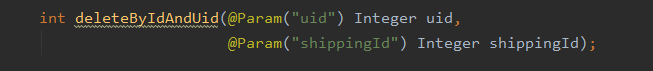
1）service层基本逻辑

新增收货地址：

将传入的表单参数form传给数据库对应类，调用insertSelective方法插入数据库，并返回影响行数，对行数进行判断，若为0，则返回异常，最后返回成功添加信息

删除收货地址：

创建操作数据库的方法deleteByIdAndUid，用于删除同时匹配uid和shippingId的地址，返回影响行数，对行数进行判断，若为0则返回错误信息，否则返回成功



//TODO @Param

更新收获地址：

将更新表单复制对象给数据库实体类对象，并设置缺少的字段，完成后写入，最后对影响行数进行判断并返回相应数据

获取收货地址列表并分页显示：

利用分页插件，根据uid查询收货地址列表，返回相应信息

2）其他细节

Mapper的insert和insertSelective方法

1、selective的意思是：选择性  
2、insertSelective--选择性保存数据；  
比如User里面有三个字段:id，name，age，password  
但是我只设置了一个字段；  
User u=new user();  
u.setName（"张三"）；  
insertSelective（u）；  
3、insertSelective执行对应的sql语句的时候，只插入对应的name字段；（主键是自动添加的，默认插入为空）  
insert into tb\_user （id，name） value （null，"张三"）；  
4、而insert则是不论你设置多少个字段，统一都要添加一遍，不论你设置几个字段，即使是一个。  
User u=new user();  
u.setName（"张三"）；  
insertSelective（u）；

insert into tb\_user （id，name，age，password） value （null，"张三"，null，null）；

5、关于insertSelective()返回值，如果插入成功则返回1，失败返回0；对于updateByPrimaryKeySelective()的返回值，成功更新几条返回则返回相应的条数，比如更新了3条，返回值为3，如果更新失败返回0。

## Controller层

各接口传参调方法即可，须交验的表单信息在参数前添加注解@Valid，在类中变量前添加注解@Notxxx，并注意配置拦截器，uid在session中拿到

## 单元测试

/\*\*

\* @author SaoE

\* @date 2022/2/16 17:59

\*/

@Slf4j

public class IShippingServiceTest extends MallApplicationTests {

@Autowired

private IShippingService shippingService;

private Integer uid = 1;

private ShippingForm shippingForm;

private Integer shippingId = 10;

@Before

public void before(){

shippingForm = new ShippingForm();

shippingForm.setReceiverName("肖逸");

shippingForm.setReceiverAddress("新姜窑花园");

shippingForm.setReceiverCity("苏州");

shippingForm.setReceiverMobile("0512-66264276");

shippingForm.setReceiverPhone("18120063446");

shippingForm.setReceiverProvince("江苏");

shippingForm.setReceiverDistrict("吴中区");

shippingForm.setReceiverZip("215000");

add();

}

public void add() {

ResponseVo<Map<String, Integer>> responseVo = shippingService.add(uid, shippingForm);

log.info("result={}",responseVo);

this.shippingId = responseVo.getData().get("shippingId");

Assert.assertEquals(ResponseEnum.SUCCESS.getCode(),responseVo.getStatus());

}

@After

public void delete() {

ResponseVo responseVo = shippingService.delete(uid, shippingId);

log.info("result={}",responseVo);

Assert.assertEquals(ResponseEnum.SUCCESS.getCode(),responseVo.getStatus());

}

@Test

public void update() {

shippingForm.setReceiverCity("南京");

ResponseVo responseVo = shippingService.update(uid,shippingId, shippingForm);

log.info("result={}",responseVo);

Assert.assertEquals(ResponseEnum.SUCCESS.getCode(),responseVo.getStatus());

}

@Test

public void lsit() {

ResponseVo responseVo = shippingService.list(uid,1,10);

log.info("result={}",responseVo);

Assert.assertEquals(ResponseEnum.SUCCESS.getCode(),responseVo.getStatus());

}

}

订单：

不可变：商品图片、收货地址

可变：状态