

**数据结构大作业报告**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | 项目名称： | 后台商品管理系统 | | 班 级： | 软件2341 | | 学 号： | 2305224134 | | 姓名： | 袁吉栋 |   2023 年 10 月 15 日 |

**目 录**

[1.问题分析和系统整体设计 3](#_Toc2053608839)

[1.1 项目描述 3](#_Toc646582489)

[1.2 项目背景 3](#_Toc844638803)

[1.3项目要求 3](#_Toc977455351)

[2.数据结构和程序结构设计 4](#_Toc1989668354)

[2.1数据结构设计 4](#_Toc1888158241)

[2.1.1 管理员数据结构 4](#_Toc909704768)

[2.1.2 商品数据结构 4](#_Toc1471952783)

[2.1.3 订单数据结构 5](#_Toc98810441)

[2.1.4 销售数据结构 6](#_Toc702222756)

[2.1.5 库存数据结构 7](#_Toc1835219827)

[2.2类的设计 8](#_Toc232010528)

[2.2.1 controller层类的设计 8](#_Toc1718124791)

[2.2.2 service层类的设计 9](#_Toc1458244775)

[2.2.3 mapper层类的设计 9](#_Toc1636553861)

[2.3程序结构设计 9](#_Toc590191051)

[2.3.1 商品信息管理模块 9](#_Toc114028664)

[2.3.2进货管理模块 10](#_Toc924342724)

[2.3.3 销售管理模块 10](#_Toc531459870)

[2.3.4 库存管理模块 10](#_Toc861547217)

[3. 关键问题和算法 11](#_Toc1689328045)

[3.1 新增商品的问题 11](#_Toc655155328)

**[3.1.1 Controller 11](#_Toc1046939527)**

**[3.1.2 Service 12](#_Toc1579110418)**

**[3.1.3 Mapper 14](#_Toc1505885700)**

**[3.1.4 Sql 15](#_Toc1326180005)**

[3.2 上架和下架商品的问题 16](#_Toc374571822)

[3.2.1 Controller 17](#_Toc1154042997)

[3.2.2 Service 18](#_Toc2075834522)

[3.2.3 Mapper 19](#_Toc531482092)

[3.2.4 Sql 19](#_Toc1235032371)

[3.3 查询订单的问题 20](#_Toc1759611142)

[3.3.1 Controller 20](#_Toc787160757)

[3.3.2 Service 21](#_Toc1311577379)

[3.3.3 Mapper 22](#_Toc1908856045)

[3.3.4 Sql 23](#_Toc885345782)

[4. 总结和回顾 23](#_Toc92368011)

[4.1 商品管理 24](#_Toc1945967743)

[4.2 商品上下架 24](#_Toc1851396438)

[4.3 查询和日期验证 24](#_Toc1529372083)

# 1.问题分析和系统整体设计

## 1.1 项目描述

随着电子商务的飞速发展，小型网店在现代商业中扮演着重要的角色。然而，对于小型网店的后台商品管理和库存追踪通常需要一种高效的解决方案。为了满足这一需求，我们提出了一个简单而功能强大的后台商品管理系统。

## 1.2 项目背景

小型网店通常无法承担昂贵的企业级库存管理系统，因此他们需要一个经济实惠、易于使用的解决方案，以管理商品、跟踪库存、进行销售和进货操作。这个项目旨在满足这些小型网店的需求，提供一个易于使用的后台商品管理系统，帮助他们实时管理库存、追踪销售和进货记录。

## 1.3项目要求

（1）存储商品信息：每种商品的信息包括商品名和单价。

（2）进货管理：支持商品的进货操作和进货历史信息查询。

（3）销售管理：支持商品的销售操作和销售历史查询。

（4）库存管理：提供库存商品的查询和管理功能，包括商品统计和即将售罄商品的名录。

# 2.数据结构和程序结构设计

## 2.1数据结构设计

**2.1.1 管理员数据结构**

管理员数据结构用于表示管理员的信息，包括id、用户名和密码

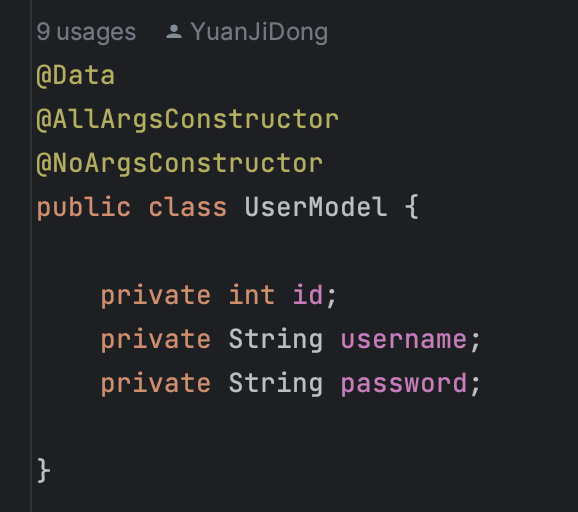


图2.1.1 用户模型类

**2.1.2 商品数据结构**

商品数据结构用于表示每个商品的信息，包括id、商品名、单价、库存id、商品介绍、数量和状态

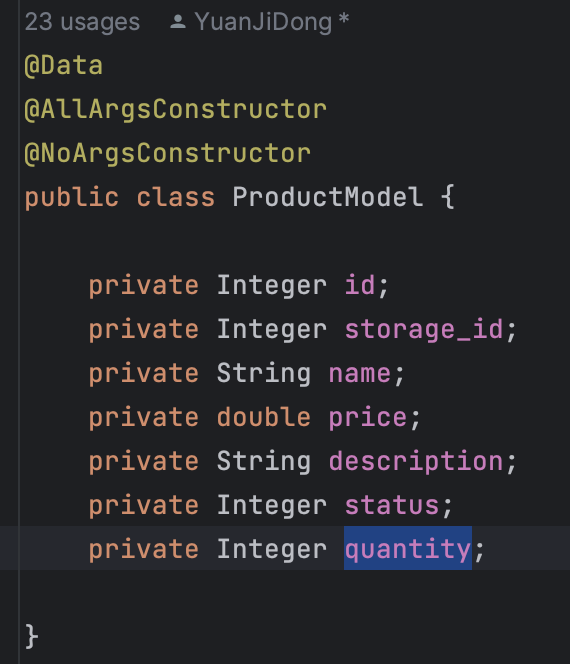


图2.1.2 产品模型类

**2.1.3 订单数据结构**

订单数据结构用于表示每个订单的信息，包括id、订单名称、订单日期、订单数量、订单价格、订单描述和状态



图2.1.1 订单模型类

**2.1.4 销售数据结构**

销售数据结构用于表示每个销售的信息，包括id、销售名称、销售日期、销售数量、销售价格、销售描述和状态

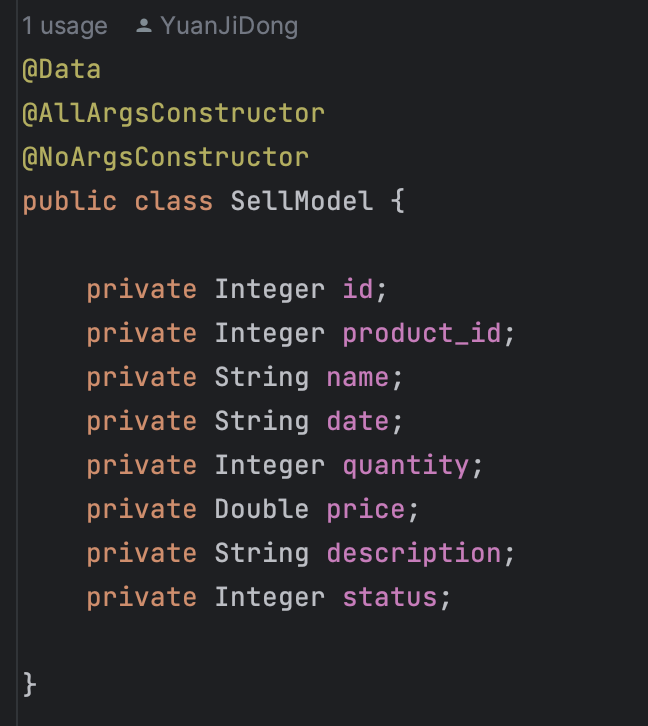


图2.1.14 销售模型类

**2.1.5 库存数据结构**

库存数据结构用于表示每个库存的信息，包括id、订单id、库存名称、入库日期、入库数量、入库价格、入库描述

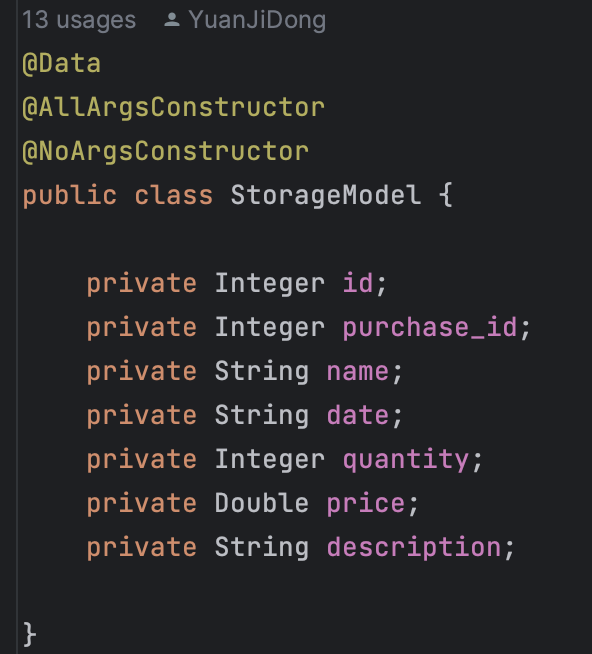


图2.1.5 库存模型类

## 2.2类的设计

将程序分类三层：用于定义接口和返回数据的controller层，用于处理业务逻辑的service层以及用于连接数据库的mapper层

### 2.2.1 controller层类的设计

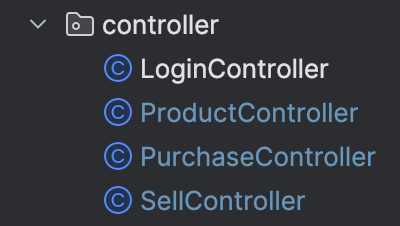
****

图2.2.1 controller层

### 2.2.2 service层类的设计

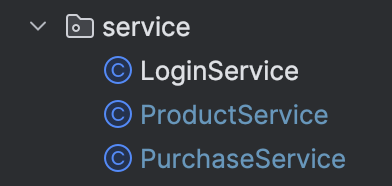
****

图2.1.14 service层

### 2.2.3 mapper层类的设计

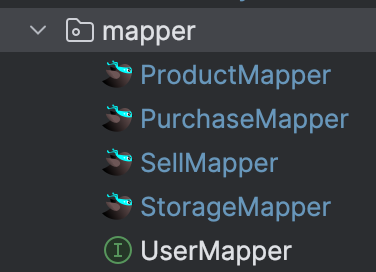
****

图2.1.14 mapper层

## 2.3程序结构设计

### 2.3.1 商品信息管理模块

- 职责：负责存储和管理商品的基本信息，包括商品名和单价。提供了创建、更新和删除商品信息的功能。

- 数据流：可以与其他模块共享商品信息以便进货、销售和库存管理。

- 控制流：管理员可以通过图形用户界面执行商品信息的添加、修改和删除操作。

### 2.3.2进货管理模块

- 职责：处理商品的进货操作，记录进货时间、进货数量和总价值，同时保存进货历史记录。

- 数据流：需要访问商品信息以确定商品的单价，进货记录将更新库存和进货历史记录。

- 控制流：管理员通过输入进货信息执行进货操作。

### 2.3.3 销售管理模块

- 职责：处理商品的销售操作，记录销售时间、销售数量和总价值，同时保存销售历史记录。

- 数据流：需要访问商品信息以确定商品的单价，销售记录将更新库存和销售历史记录。

- 控制流：管理员通过输入销售信息执行销售操作。

### 2.3.4 库存管理模块

- 职责：查询和管理库存商品，包括商品统计和即将售罄商品的警告。

- 数据流：需要访问商品信息以获取商品的单价，库存记录将更新库存信息。

- 控制流：管理员可以查询库存信息，获取商品统计和即将售罄商品的警告。

# 关键问题和算法

* 1. 新增商品的问题

关键问题：新增的产品只能从库存中选，不能自己定义，且必须为未上架状态，商品名称不可为空，商品价格不可小于0，商品描述不可为空，商品数量不可大于库存数量

算法：

**3.1.1 Controller**

@PostMapping("api/add-product")  
public Result addProduct (@RequestBody AdditionProductModel additionProductModel) {  
 int storageId = additionProductModel.getStorageId();  
 String storageName = additionProductModel.getStorageName();  
 ProductModel productModel = new ProductModel();  
 productModel.setName(additionProductModel.getName());  
 productModel.setPrice(additionProductModel.getPrice());  
 productModel.setDescription(additionProductModel.getDescription());  
 productModel.setQuantity(additionProductModel.getQuantity());  
  
 int status = productService.checkStorageProduct(storageId, storageName);  
 // 验证仓库的商品是否有效  
 if (status == 0) {  
 int state = productService.checkAdditionProduct(productModel, storageId);  
 // 验证提交的新增商品信息是否有效  
 if (state == 2) return Result.*error*("商品名称不可为空");  
 if (state == 3) return Result.*error*("商品价格不可小于0");  
 if (state == 4) return Result.*error*("商品描述不可为空");  
 if (state == 5) return Result.*error*("商品数量不可大于库存数量");  
 if (state == 1) {  
 productModel.setStatus(0);  
 productModel.setStorage\_id(storageId);  
 productService.addProduct(productModel);  
 return Result.*success*("添加成功");  
 }  
 }  
  
 return Result.*error*("该库存商品不存在");  
}

**3.1.2 Service**

// 验证仓库的商品是否有效  
public int checkStorageProduct (int storageId, String storageName) {  
 // 验证上架的商品id是否包含在仓库内  
 if (productMapper.checkExist(storageId, storageName) > 0) {  
 return 0;  
 }  
 return -1;  
}  
  
// 验证提交的新增商品信息是否有效  
public int checkAdditionProduct (ProductModel productModel, int storageId) {  
 // 商品名称不可为空  
 if (productModel.getName() == null || productModel.getName().isEmpty()) {  
 return 2;  
 }  
  
 // 商品价格不可小于0  
 if (productModel.getPrice() < 0) {  
 return 3;  
 }  
  
 // 商品描述不可为空  
 if (productModel.getDescription() == null || productModel.getDescription().isEmpty()) {  
 return 4;  
 }  
  
 // 商品数量不可大于库存数量  
 if (productMapper.getStorageQuantity(storageId) < productModel.getQuantity()) {  
 return 5;  
 }  
  
 return 1;  
}

**3.1.3 Mapper**

// 检查商品是否存在  
int checkExist (int id, String name);  
  
// 获取商品状态  
int getProductStatus (String name);  
  
// 获取库存数量  
int getStorageQuantity (int id);  
  
// 添加商品  
void addProduct (ProductModel productModel);

**3.1.4 Sql**

<select id="checkExist" resultType="int">  
 SELECT  
 COUNT(\*)  
 FROM  
 storage  
 WHERE  
 name=#{name} AND id=#{id}  
</select>  
  
<select id="getProductStatus" resultType="int">  
 SELECT  
 status  
 FROM  
 purchase  
 WHERE  
 id=#{id}  
</select>  
  
<select id="getStorageQuantity" resultType="int">  
 SELECT  
 quantity  
 FROM  
 storage  
 WHERE  
 id=#{id}  
</select>  
  
<insert id="addProduct">  
 INSERT INTO  
 product (storage\_id, name, price, description, quantity, status)  
 VALUES  
 (#{storage\_id}, #{name}, #{price}, #{description}, #{quantity}, #{status})  
</insert>

## 上架和下架商品的问题

关键问题：上架前商品状态只能为未上架或已下架，下架前商品状态只能为已上架或已售罄

算法：

### 3.2.1 Controller

@GetMapping("api/modify-product-status")  
public Result putProduct (int id, int status) {  
 String message = "请重新尝试";  
 // 上架商品  
 if (status == 1) {  
 return productService.checkPutProduct(id, 1) == 1 ?  
 Result.*success*("上架成功") :  
 Result.*error*(message);  
 }  
 // 下架商品  
 if (status == 2) {  
 return productService.checkDeleteProduct(id, 3) == 1 ?  
 Result.*success*("下架成功") :  
 Result.*error*(message);  
 }  
 return Result.*success*(message);  
}

### 3.2.2 Service

// 验证上架是否合法：上架前商品状态只能为未上架或已下架(从已下架变为上架时商品数量不可为0)  
public int checkPutProduct (int id, int status) {  
 // 从已下架变为已上架，要检查库存的数量是否大于0  
 if (productMapper.checkPutProduct(id) == 0 || (  
 productMapper.checkPutProduct(id) == 3 &&  
 productMapper.getStorageQuantityByProductId(id) > 0)) {  
 productMapper.setProductStatus(id, status);  
 return 1;  
 }  
 return 0;  
}  
  
// 验证下架是否合法：下架前商品状态只能为已上架或已售磬  
public int checkDeleteProduct (int id, int status) {  
 if (productMapper.checkPutProduct(id) == 1 || productMapper.checkPutProduct(id) == 2) {  
 productMapper.setProductStatus(id, status);  
 return 1;  
 }  
 return 0;  
}

### 3.2.3 Mapper

// 检查上架/下架商品  
int checkPutProduct (int id);  
  
// 上架/下架  
void setProductStatus (int id, int status);

### 3.2.4 Sql

<select id="checkPutProduct" resultType="int">  
 SELECT  
 status  
 FROM  
 product  
 WHERE  
 id=#{id}  
</select>  
  
<update id="setProductStatus">  
 UPDATE  
 product  
 SET  
 status=#{status}  
 WHERE  
 id=#{id}  
</update>

## 查询订单的问题

关键问题：管理员最少给定一个开始日期、结束日期或商品名，验证开始日期和结束日期的格式是否为yyyy-mm-dd、开始日期小于等于结束日期、如果开始日期有值结束日期没值，结束日期为当前日期、如果开始日期没值结束日期有值，开始日期为2000-01-01，给定商品名时模糊查询，未给定则选出日期对应的数据

算法：

### 3.3.1 Controller

// 查询订单  
@PostMapping("api/search-purchase")  
public Result searchPurchase (@RequestBody SearchModel searchModel) {  
 return Result.*success*(purchaseService.findPurchaseByName(searchModel.getName(), searchModel.getStart(), searchModel.getEnd()));  
}

### 3.3.2 Service

// 查找订单  
public ArrayList<PurchaseModel> findPurchaseByName (String name, String start, String end) {  
 // 验证日期格式  
 SimpleDateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd");  
 Date startDate = null;  
 Date endDate = null;  
  
 try {  
 if (start != null && !start.isEmpty()) {  
 startDate = dateFormat.parse(start);  
 }  
 if (end != null && !end.isEmpty()) {  
 endDate = dateFormat.parse(end);  
 }  
 } catch (ParseException e) {  
 throw new IllegalArgumentException("日期格式必须为yyyy-MM-dd");  
 }  
  
 if (startDate != null && endDate != null && startDate.after(endDate)) {  
 throw new IllegalArgumentException("开始日期不能晚于结束日期");  
 }  
  
 // 处理日期范围  
 if (start != null && !start.isEmpty() && (end == null || end.isEmpty())) {  
 // 如果start有值而end没有值，将end设置为当前日期  
 endDate = new Date();  
 }  
 if (end != null && !end.isEmpty() && (start == null || start.isEmpty())) {  
 // 如果end有值而start没有值，将start设置为2000-01-01  
 try {  
 startDate = dateFormat.parse("2000-01-01");  
 } catch (ParseException e) {  
 // 处理解析异常  
 }  
 }  
 return purchaseMapper.findPurchaseByName("%" + name + "%", startDate, endDate);  
}

### 3.3.3 Mapper

// 根据名字查找订单  
ArrayList<PurchaseModel> findPurchaseByName (String name, Date start, Date end);

### 3.3.4 Sql

<select id="findPurchaseByName" resultType="com.yuanjidong.shop.model.PurchaseModel">  
 SELECT  
 \*  
 FROM  
 purchase  
 <where>  
 <if test="name != null">  
 name LIKE #{name}  
 </if>  
 <if test="start != null and end != null">  
 AND date BETWEEN #{start} AND #{end}  
 </if>  
 </where>  
</select>

# 

# 总结和回顾

在项目的开发过程中，解决了几个关键问题，以下是对项目的总结和回顾：

## 4.1 商品管理

一项重要的任务是有效管理商品的创建和库存控制。确保了以下方面的功能：

- 新增商品只能从库存中选取，防止了不合法的商品添加。

- 商品必须为未上架状态，保证了只有在特定条件下才能上架商品。

- 商品名称不能为空，商品价格不可小于0，商品描述不可为空，商品数量不可大于库存数量，这些限制确保了商品信息的完整性和准确性。

## 4.2 商品上下架

商品的上下架状态管理是网店运营的关键。规定了以下规则：

- 商品状态只能从未上架转变为已上架，或从已上架转变为已售罄，确保了商品状态的合理流转。

- 商品下架前只能为已上架或已售罄状态，保证了下架操作的合法性。

## 4.3 查询和日期验证

有效的查询和日期验证是数据分析和报告生成的关键。实现了以下功能：

- 用户必须至少给定一个开始日期、结束日期或商品名，确保了查询的有效性。

- 验证了日期的格式为 yyyy-mm-dd，开始日期小于等于结束日期，以及日期的默认值设定。

- 模糊查询的商品名，以满足用户的查询需求。

总的来说，项目的实现有效地解决了关键问题，确保了系统的稳定性和用户友好性。我们成功开发了一个后台商品管理系统，使小型网店能够轻松地管理商品、库存、进货和销售操作。这个项目的优点包括：

- 数据结构设计合理，模块化的程序结构使系统易于扩展和维护。

- 合理的数据验证和限制，确保了数据的完整性和安全性。

- 响应用户需求的功能，包括商品管理、销售和进货历史查询，以及报表生成。

- 满足了小型网店的后台商品管理需求，提高了库存管理的效率和精度。

然而，还有一些改进的空间，例如增加更多的附加功能、提供更多的查询选项以满足不同用户需求，以及改进用户界面以提高用户体验。项目的未来工作可能包括扩展功能和优化性能。

总的来说，项目的完成对小型网店管理提供了有力的工具，帮助他们提高效率、减少错误，并更好地满足客户需求。这个项目为后台商品管理系统的设计和实现提供了一个有用的示例。