**互联网软件开发技术与实践**

**秒杀系统小组项目**

项目名称： 秒杀系统

院 系： 软件与微电子学院

组 员： 潘 江

肖 峰

陈子翔

黄 天

完成日期： 2024年1月16日

目 录

[1 秒杀系统设计思路 2](#_Toc14467)

[1.1 秒杀系统架构设计 2](#_Toc23921)

[1.1.1 案例概述 2](#_Toc30943)

[1.1.2 前端设计（Vue） 2](#_Toc21919)

[1.1.3 后端设计（SpringBoot） 2](#_Toc2357)

[1.1.4 数据库设计（MySQL） 3](#_Toc19490)

[1.1.5 缓存设计（Redis） 3](#_Toc30905)

[1.1.6 消息队列（RabbitMQ） 3](#_Toc10125)

[1.1.7 系统安全和性能 3](#_Toc32569)

[1.2 后端设计思路 3](#_Toc1516)

[1.2.1 系统架构设计 3](#_Toc23073)

[1.2.2 秒杀逻辑实现 4](#_Toc12358)

[1.2.3 性能测试 4](#_Toc21119)

[1.2.4 安全性与日志记录监控 4](#_Toc26378)

[1.3 数据库设计思路 5](#_Toc15415)

[1.3.1 设计原则与性能优化 5](#_Toc14073)

[1.3.2 关键数据表设计 5](#_Toc32712)

[1.4 缓存设计思路 6](#_Toc16974)

[1.5 消息队列设计思路 7](#_Toc2898)

[2 秒杀系统代码实现 7](#_Toc2096)

[2.1 MySQL代码 7](#_Toc11458)

[2.2 后端代码 10](#_Toc17591)

[2.2.1 后端代码结构 10](#_Toc9797)

[2.2.2 实体类定义entity 12](#_Toc5637)

[2.2.3 控制器controller 16](#_Toc6005)

[2.2.4 实现service 20](#_Toc14150)

[2.2.5 Mybatis代码 24](#_Toc26850)

[2.3 前端代码 28](#_Toc26771)

[2.3.1 前端代码结构 28](#_Toc5343)

[2.3.2 关键界面代码 29](#_Toc4878)

[2.4 主要功能实现 35](#_Toc16317)

[2.4.1 Redis缓存 35](#_Toc29356)

[2.4.2 RabbitMQ处理用户的秒杀请求 35](#_Toc25307)

[3 秒杀系统运行效果 38](#_Toc7051)

[3.1 用户秒杀系统 38](#_Toc10547)

[3.2 秒杀后台管理系统 43](#_Toc7882)

[3.3 后台管理系统数据展示 45](#_Toc20808)

[4 秒杀系统测试结果 46](#_Toc29492)

[4.1 秒杀接口的测试 46](#_Toc18346)

[4.2 其他部分的测试 47](#_Toc10311)

# 秒杀系统设计思路

## 秒杀系统架构设计

### 案例概述

1. 目标：

* 实现一个秒杀系统，用户可以在秒杀时间内抢购限量商品。

1. 关键点：

* 前端：用户界面设计，与后端接口交互。
* 后端：处理业务逻辑，与数据库和缓存交互。
* 数据库：存储用户数据和商品信息。
* 缓存（Redis）：减轻数据库压力，提高读取速度。
* 消息队列（RabbitMQ）：异步处理订单，提高系统响应速度。

### 前端设计（Vue）

1. 前端页面

* 商品展示页：显示秒杀商品信息。
* 秒杀详情页：用户参与秒杀的操作界面。

1. 前端技术要点

* 使用Vue框架构建SPA（单页应用程序）。
* 使用axios进行前后端的API交互。
* 使用Vue Router进行页面路由控制。
* 使用ElementUI构建美观的界面。

### 后端设计（SpringBoot）

1. 后端架构

* Controller层：处理HTTP请求。
* Service层：业务逻辑处理。
* DAO层：与数据库交互。

1. 关键技术点

* SpringBoot框架快速搭建应用。
* 使用MyBatis与MySQL数据库交互。
* 使用Redis作为缓存，存储热点数据，如商品的秒杀状态和库存信息。
* 使用RabbitMQ处理秒杀后的订单生成，降低系统压力。

### 数据库设计（MySQL）

1. 数据表设计

* 用户表：存储用户信息。
* 商品表：存储商品信息。
* 订单表：存储用户的订单信息。

1. 数据库优化

* 索引优化，确保查询效率。
* 事务处理，保证数据一致性。

### 缓存设计（Redis）

1. 缓存使用场景

* 缓存商品的库存信息。
* 缓存秒杀开始和结束的时间。

1. 缓存优化

* 设置合理的过期时间。
* 避免缓存雪崩和穿透问题。

### 消息队列（RabbitMQ）

1. 应用场景

* 用户下单后，发送消息到队列，异步生成订单。

1. 队列设计

* 使用不同的队列处理不同的业务逻辑。
* 确保消息的可靠性和顺序性。

### 系统安全和性能

1. 安全措施

* 接口防刷限流。
* 数据加密和用户验证。

1. 性能优化

* 使用多线程和异步处理提高响应速度。
* 负载均衡和数据库读写分离。

## 后端设计思路

### 系统架构设计

1. SpringBoot后端架构：Controller层处理HTTP请求；Service层处理业务逻辑；DAO层与数据库进行交互。
2. MySQL数据库：使用关系型数据库MySQL记录相关信息。
3. 缓存层：引入缓存层，例如Redis，用于存储热门商品信息、用户状态等，以提高读写速度和降低数据库压力。
4. 异步消息队列：使用RabbitMQ消息队列，处理秒杀请求的异步处理。这有助于削峰填谷，提高系统的并发处理能力。

### 秒杀逻辑实现

1. 判断秒杀时间

在秒杀系统中，秒杀活动通常有一个明确的开始时间和结束时间。为了提高性能，可以将秒杀商品的开始时间、结束时间以及秒杀状态存储在 Redis 缓存中。在处理秒杀请求时，首先从 Redis 中获取秒杀状态信息，判断当前是否处于秒杀时间范围内。

1. 判断库存

在执行秒杀前，需要判断商品的秒杀库存是否大于0。可以将库存信息存储进redis数据库中实现快速读取以及减少库存操作。

1. 生成订单

如果秒杀条件满足，生成订单并减少库存。向消息队列中发送一个订单消息。

1. 异步处理

使用 RabbitMQ 处理秒杀请求的异步处理，可以将秒杀请求放入消息队列，由后台异步处理生成订单等操作。这有助于降低系统压力，提高系统的并发处理能力。

### 性能测试

1. 并发量：测试系统能处理的最大并发请求，通过压力测试工具模拟高并发场景。
2. 响应时间：测试系统处理一个请求所需的平均时间，关注系统的响应性能。
3. 系统稳定性：在高并发下测试系统的表现，监测系统的稳定性和可用性。

### 安全性与日志记录监控

1. 防止超卖：采用消息队列、乐观锁、悲观锁等方式，防止用户超过库存购买。
2. 防止恶意请求：使用验证码、限流等手段，防止恶意用户进行刷单等行为。
3. 用户身份验证：确保用户身份的合法性，防止未授权用户进行秒杀操作。
4. 记录日志：记录关键操作的日志，方便排查问题和性能优化。
5. 监控系统：使用监控工具实时监测系统的运行状态，及时发现和处理异常。

## 数据库设计思路

### 设计原则与性能优化

1. 基本设计原则

* 规范化：确保数据的一致性和减少数据冗余。
* 索引优化：通过合理的索引来加快查询速度，特别是在高并发场景下。
* 事务处理：保证数据的完整性和一致性。

1. 数据库性能优化

* 索引：在频繁查询的字段上建立索引，如user\_id, product\_id等。
* 读写分离：在高并发场景下，分离数据库的读写操作，提高性能。

### 关键数据表设计

1. 用户表（t\_user）

|  |  |  |
| --- | --- | --- |
| **用户表（t\_user）** | | |
| **字段名** | **类型** | **描述** |
| id | INT, 主键, 自增 | 用户ID |
| username | VARCHAR(255) | 用户名 |
| password | VARCHAR(255) | 密码 |
| name | VARCHAR(255) | 姓名 |
| registerDate | DATETIME | 注册时间 |
| phone | VARCHAR(100) | 手机号码 |

1. 商品表（t\_goods）

|  |  |  |
| --- | --- | --- |
| **商品表（t\_goods）** | | |
| **字段名** | **类型** | **描述** |
| id | INT, 主键, 自增 | 商品ID |
| name | VARCHAR(255) | 商品名称 |
| price | DECIMAL(10, 2) | 价格 |
| image | VARCHAR(255) | 图片 |
| stock | INT | 库存数量 |
| detail | TEXT | 详情 |

1. 秒杀商品信息表（t\_miaosha\_goods）

|  |  |  |
| --- | --- | --- |
| **秒杀商品信息表（t\_miaosha\_goods）** | | |
| **字段名** | **类型** | **描述** |
| id | INT, 主键, 自增 | 编号 |
| goodsId | INT | 真实商品ID |
| goodsName | VARCHAR(255) | 秒杀商品名称 |
| price | DECIMAL(10, 2) | 秒杀价格 |
| stock | INT | 秒杀库存 |
| startTime | DATETIME | 秒杀开始时间 |
| endTime | DATETIME | 秒杀结束时间 |
| createdTime | DATETIME | 创建时间 |

1. 订单表（t\_order）

|  |  |  |
| --- | --- | --- |
| **订单表（t\_order）** | | |
| **字段名** | **类型** | **描述** |
| id | INT, 主键, 自增 | 订单ID |
| createTime | DATETIME | 创建时间 |
| goodsId | INT | 商品ID |
| miaoShaGoodsId | INT | 秒杀秒杀商品ID |
| payStatus | INT | 支付状态,0代表未支付,1代表已支付 |
| num | INT | 购买数量 |
| userId | INT | 用户ID |
| price | INT | 价格 |

## 缓存设计思路

在秒杀系统中，缓存设计是至关重要的，因为秒杀系统通常面临高并发的读写压力。以下是秒杀系统中的缓存设计思路：

1. 商品信息缓存

* 缓存秒杀商品详情：将秒杀商品的详细信息缓存在Redis中，包括秒杀价格、秒杀库存、秒杀开始时间和结束时间等。使用Hash结构存储，以秒杀商品ID为字段名。
* 缓存秒杀状态：使用缓存记录秒杀活动的状态，包括是否处于秒杀时间范围内。这可以通过在Redis中存储秒杀状态标识来实现。

1. 秒杀活动信息缓存

* 缓存秒杀商品详情：将秒杀商品的详细信息缓存在Redis中，包括秒杀价格、秒杀库存、秒杀开始时间和结束时间等。使用Hash结构存储，以秒杀商品ID为字段名。
* 缓存秒杀状态：使用缓存记录秒杀活动的状态，包括是否处于秒杀时间范围内。这可以通过在Redis中存储秒杀状态标识来实现。

1. 库存信息缓存

* 缓存库存数量：将秒杀商品的库存数量缓存在Redis中，使用List或Bitmap等数据结构表示库存的状态。这可以帮助快速判断商品是否还有库存。

1. 缓存与数据库同步

* 主动刷新：定时刷新缓存，确保缓存中的数据与数据库中的数据保持一致。可以使用定时任务或消息队列触发缓存刷新操作。
* 数据库更新时刷新缓存：在数据库更新操作中，及时刷新对应的缓存。这可以通过在数据库操作事务提交后触发缓存刷新来实现。

1. 缓存失效策略

* 定时失效：设置缓存的过期时间，以保证缓存数据不会太旧。这对于秒杀活动的信息和状态等数据非常有用。

## 消息队列设计思路

在秒杀系统中，消息队列的设计是为了解耦系统组件，提高系统的并发处理能力，以及降低对数据库的直接访问压力。以下是秒杀系统中消息队列的设计思路：

1. 异步处理秒杀请求

* 解耦请求处理：将用户的秒杀请求异步处理，通过消息队列将请求发送到后台处理。这样可以解耦前端请求和后端处理，提高系统的并发处理能力。
* 削峰平谷：通过消息队列将秒杀请求均匀分发到后端，避免瞬时大量请求同时到达后端，减轻系统的压力。

1. 订单生成异步化

* 订单生成：在秒杀成功后，生成订单的操作可以异步化处理。通过消息队列将生成订单的请求发送到后台，降低直接写入数据库的压力。
* 提高系统稳定性：通过异步生成订单，系统能更好地应对瞬时的高并发请求，提高系统的稳定性和可用性。

1. 消息队列与数据库同步

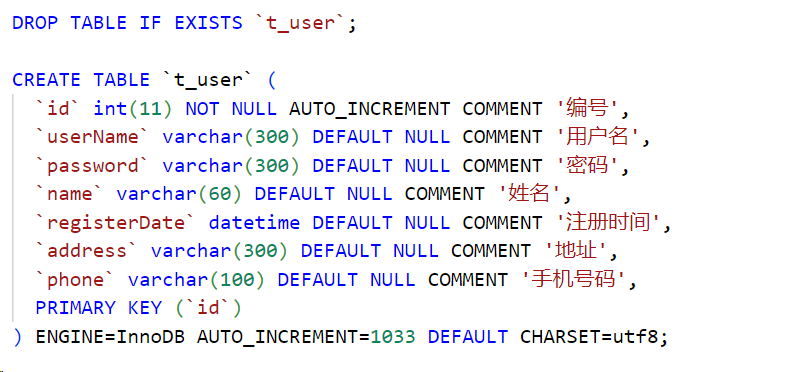
* 缓解数据库压力：通过将秒杀请求和订单生成操作异步化，减轻了直接对数据库的读写压力。消息队列与数据库同步的机制可确保系统数据的一致性。
* 事务性操作：在消息队列与数据库同步时，确保操作的事务性。例如，只有在数据库事务提交成功后，消息才能从队列中移除，以保证消息不会丢失。

# 秒杀系统代码实现

## MySQL代码

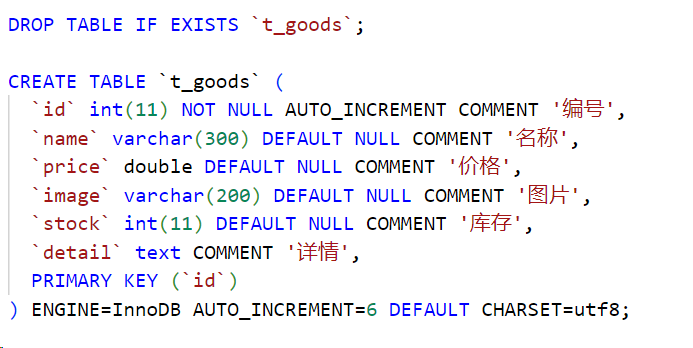
1. t\_user用户信息表

* 存储用户信息，包括用户名、密码、姓名、注册时间、地址、手机号码等。
* 在秒杀场景中，用户信息用于关联订单和记录用户的秒杀行为。



1. t\_goods商品信息表

* 用于存储普通商品的信息，如商品名称、价格、图片、库存等。
* 在秒杀场景中，这个表存储所有可秒杀的商品，每个商品对应一条记录。



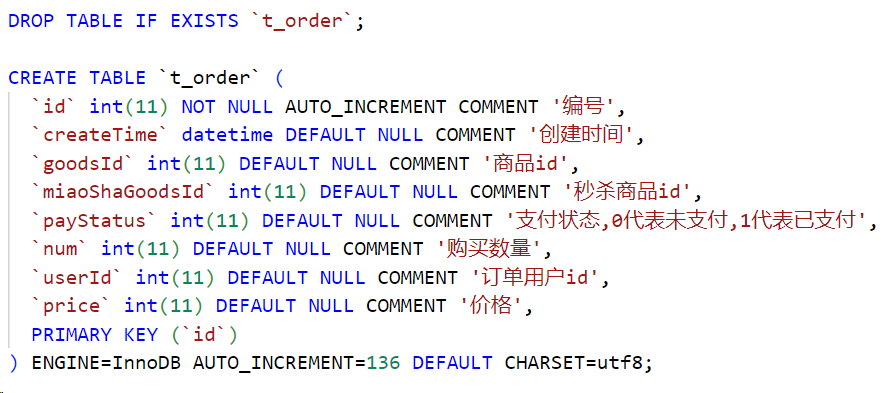
1. t\_miaosha\_goods秒杀商品表

* 存储秒杀商品的信息，包括真实商品的ID、秒杀商品名称、秒杀价格、库存、秒杀开始和结束时间等。
* 用于记录所有参与秒杀的商品信息。



1. t\_order订单表

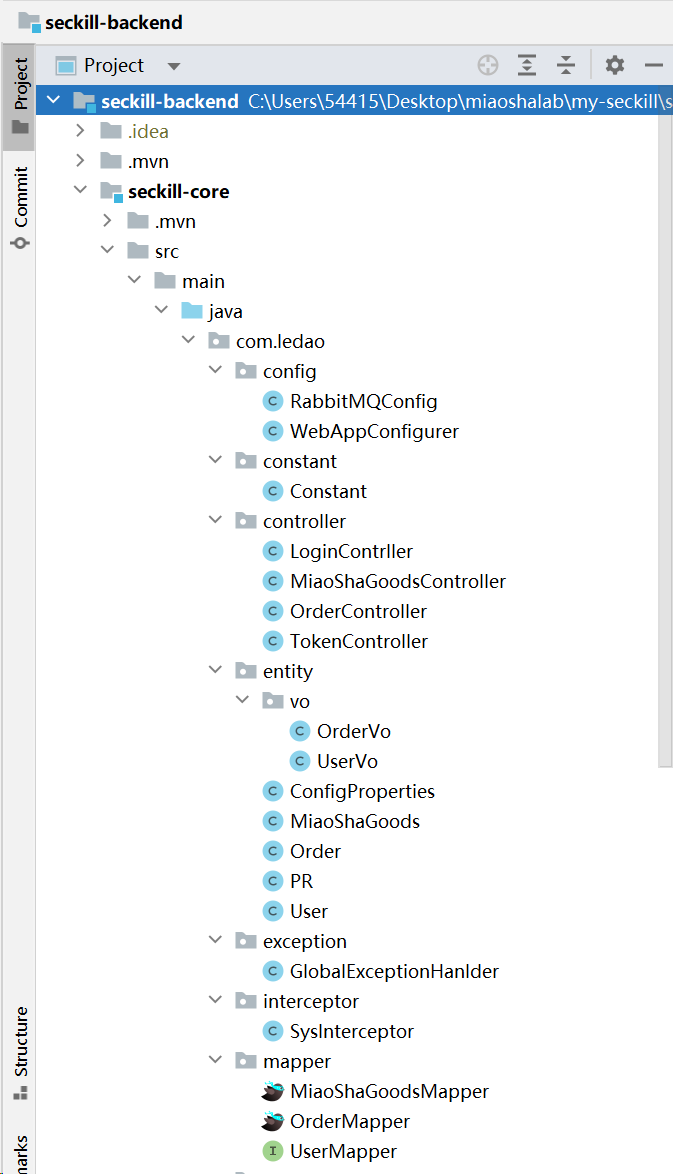
* 用于存储订单信息，包括订单创建时间、商品ID、秒杀商品ID、支付状态、购买数量、用户ID、价格等。
* 在秒杀场景中，每次成功秒杀产生一条订单记录。



上述表结构用于支持秒杀场景，其中t\_user表存储用户信息，t\_goods表存储所有商品信息，t\_miaosha\_goods表存储参与秒杀的商品信息，t\_order表存储订单信息。秒杀商品的库存信息和订单信息通过这些表相互关联，实现秒杀业务的记录和管理。

## 后端代码

### 后端代码结构





1. config：项目配置信息，这里主要是RabbitMQ消息队列的配置信息。
2. controller：处理各个实体类的HTTP请求。
3. mapper：实现各个实体类与数据库交互功能。
4. entity：定义各实体类代码。
5. service：业务逻辑处理代码。
6. util：相关工具类代码。
7. resources/mapper：MyBatis与MySQL数据库交互代码。

### 实体类定义entity

1. User.java用户类

User类用于秒杀系统，表示用户信息，包括编号、用户名、密码等字段。通过Lombok注解简化代码，与秒杀场景结合，可以记录用户在秒杀活动中的注册时间、地址和手机号。



1. MiaoShaGoods.java秒杀商品实体类

MiaoShaGoods类用于秒杀系统，表示秒杀商品信息，包括编号、真实商品ID、关联的商品实体、秒杀价格、秒杀数量、开始时间和结束时间等字段。通过Lombok注解简化代码，与秒杀场景结合，记录了商品的秒杀相关信息，包括价格、数量以及秒杀的时间范围。



1. Order.java订单实体类

Order类用于秒杀系统，表示订单信息，包括编号、创建时间、商品ID、关联的商品实体、秒杀商品ID、支付状态、购买数量、用户ID和价格等字段。通过Lombok注解简化代码，与秒杀场景结合，记录了订单的相关信息，包括商品、秒杀商品、支付状态和购买数量等。



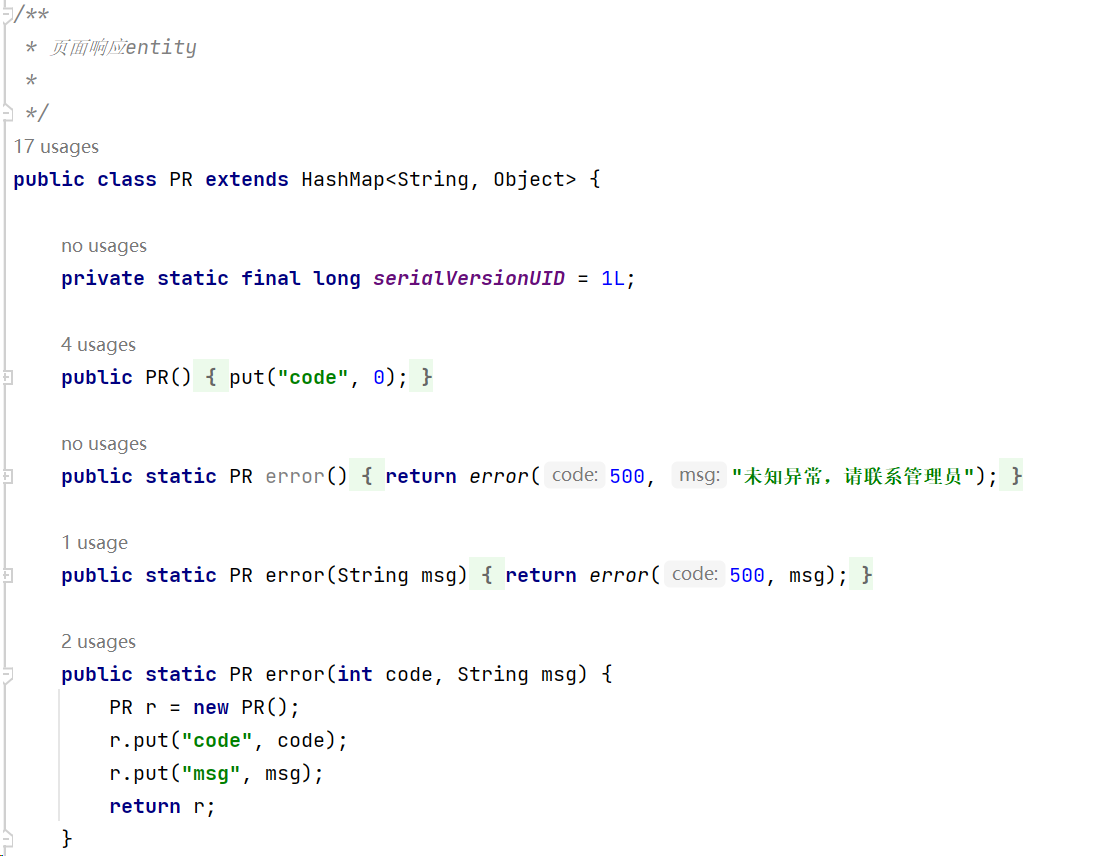
1. ConfigProperties.java配置文件属性实体类

ConfigProperties类是用于处理配置文件属性的实体类，包含了Redis的IP地址、端口和连接密码等属性。通过Lombok注解简化代码，同时使用Spring的@Value注解将配置文件中的属性值注入到对应的字段中。在秒杀场景中，这些属性可能被用于配置与Redis相关的连接信息，用于缓存秒杀活动的数据。



1. PR.java页面响应实体类

PR类是一个用于表示页面响应的实体类，继承自HashMap<String, Object>。通过该类，可以构建包含特定结构的响应信息。其中，通过默认构造函数设置了默认的code为0。提供了静态方法用于构建成功和错误的响应，同时支持自定义code和msg。在秒杀场景中，这个类可以被用于构建接口返回的JSON格式数据，包含成功或错误的状态信息和相关消息。



### 控制器controller

1. LoginContrller.java

LoginController类是处理用户登录和注册的控制器。在秒杀场景中，用户登录和注册是关键的交互环节。

* 登录：
  + 用户提供用户名和密码，系统验证后生成一个唯一的token，并在Redis中存储该token与用户信息的映射。
  + 返回用户信息和生成的token，用于后续操作的身份验证。
* 注册：
  + 检查用户名和手机号是否已存在于数据库，确保唯一性。
  + 注册成功后，将用户信息存入数据库，密码经过MD5加密。
  + 返回注册结果，通常在秒杀场景中注册用户用于参与秒杀活动。
* 注销：
  + 从请求头获取token，通过Redis删除对应的token，实现用户注销操作。

这些操作通过Spring的@RestController和@RequestMapping注解提供API接口，返回数据格式可能采用之前提到的PR类的实例。这样的设计符合秒杀场景下用户身份验证和注册的需求。



1. ProductController.java

MiaoShaGoodsController类是处理秒杀商品相关操作的控制器。在秒杀场景中，该控制器涉及获取当前秒杀商品列表和根据ID查询秒杀商品的功能。

* 获取当前秒杀商品列表：
  + 通过listAllNow方法获取当前可以秒杀的商品列表。
  + 针对已经抢购过的用户，不显示其已经抢购过的商品，避免用户重复秒杀。
  + 通过miaoShaGoodsService和orderService服务层进行数据查询和处理。
* 根据ID查询秒杀商品：
  + 通过findById方法根据商品ID查询对应的秒杀商品信息。
  + 通过miaoShaGoodsService服务层进行数据查询。

这些操作通过Spring的@RestController和@RequestMapping注解提供API接口，返回数据格式可能采用之前提到的PR类的实例。这样的设计符合秒杀场景下获取秒杀商品信息的需求，并确保已经抢购过的商品不会再次显示。



1. OrderController.java

OrderController类是处理订单相关操作的控制器。在秒杀场景中，该控制器涉及获取订单列表、添加或修改订单（秒杀下单）、以及检验秒杀状态的功能。

* 获取订单列表：
  + 通过list方法根据条件获取用户的订单列表。
* 添加或修改订单（秒杀下单）：
  + 通过save方法处理秒杀下单操作。
  + 当订单ID为空时，表示用户正在进行秒杀下单操作。
  + 根据秒杀商品ID查询秒杀商品信息，判断是否在秒杀时间内且库存大于0。
  + 如果满足条件，则将订单信息放入消息队列，由消息队列异步处理生成订单。
* 检验秒杀状态：
  + 通过checkStatus方法检验用户的秒杀状态。
  + 如果订单存在，表示秒杀成功。
  + 如果订单不存在且秒杀商品仍有库存，表示用户在排队中。
  + 如果订单不存在且秒杀商品库存已售罄，表示秒杀已结束。

这些操作通过Spring的@RestController和@RequestMapping注解提供API接口，返回数据格式可能采用之前提到的PR类的实例。这样的设计符合秒杀场景下订单处理和秒杀状态检验的需求，并通过消息队列实现异步处理。



1. TokenController.java

TokenController类是处理Token相关操作的控制器。在秒杀场景中，Token用于保持用户登录状态，通过刷新Token的方式延长用户登录有效期。

* 刷新Token：
  + 通过refreshToken方法实现Token的刷新。
  + 从请求头获取Token。
  + 通过RedisUtil工具类将Token的过期时间重新设置为30分钟。
  + 返回刷新结果，通常用于前端定时刷新Token以保持登录状态。

这个操作通过Spring的@RestController和@RequestMapping注解提供API接口，返回数据格式可能是布尔值或者采用之前提到的PR类的实例。这样的设计符合秒杀场景下保持用户登录状态的需求。



### 实现service

1. UserServiceImpl.java

UserServiceImpl类是UserService接口的实现类，提供了用户相关的服务。在秒杀场景中，用户服务用于处理用户的注册、登录等操作。

* 根据用户名查询用户：
  + 通过findByUserName方法根据用户名查询用户信息。
  + 使用QueryWrapper构建查询条件，实现与数据库的交互。
* 根据手机号查询用户：
  + 通过findByPhone方法根据手机号查询用户信息。
  + 使用QueryWrapper构建查询条件，实现与数据库的交互。
* 添加用户：
  + 通过add方法向数据库添加用户信息。
  + 返回插入结果，通常在用户注册时使用。

这些操作通过Spring的@Service注解提供服务，与MyBatis Plus集成，通过UserMapper与数据库交互。这样的设计符合秒杀场景下用户服务的需求。

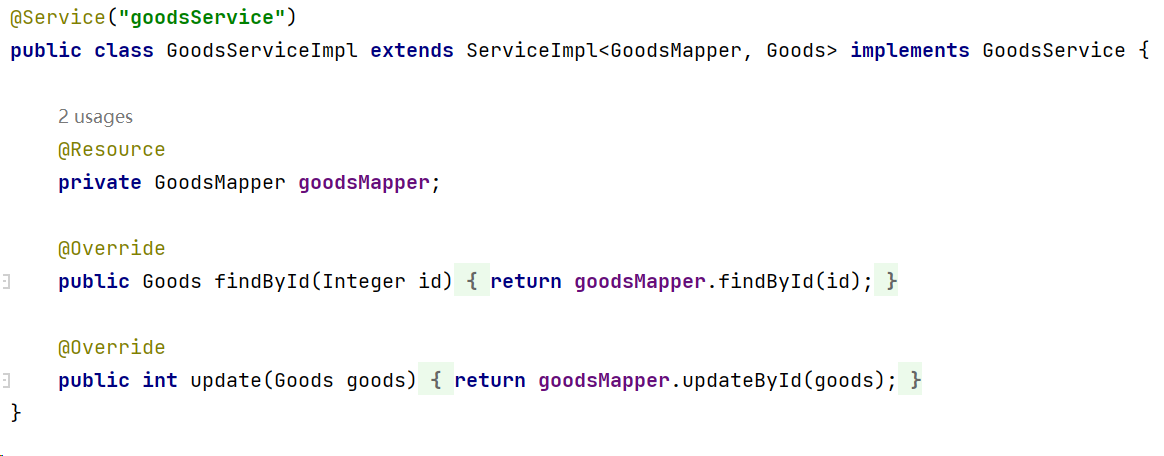


1. GoodsServiceImpl.java

GoodsServiceImpl类是GoodsService接口的实现类，提供了商品相关的服务。在秒杀场景中，商品服务用于查询商品信息和更新商品信息。

* 根据商品ID查询商品：
  + 通过findById方法根据商品ID查询商品信息。
  + 使用自定义的GoodsMapper接口实现查询，与数据库交互。
* 更新商品信息：
  + 通过update方法更新商品信息。
  + 使用MyBatis Plus的updateById方法，与数据库交互。

这些操作通过Spring的@Service注解提供服务，与MyBatis Plus集成，通过GoodsMapper与数据库交互。这样的设计符合秒杀场景下商品服务的需求。

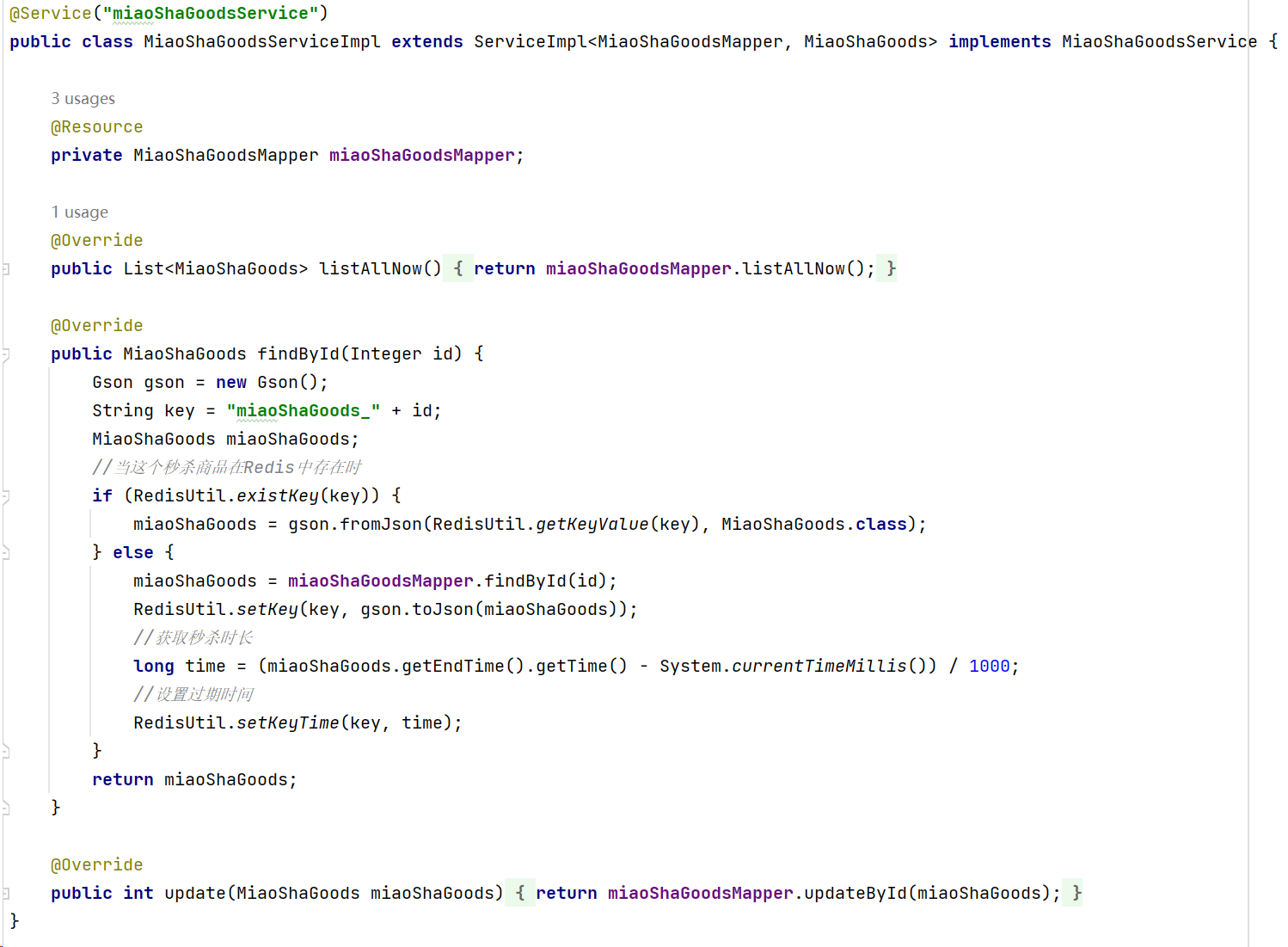


1. MiaoShaGoodsServiceImpl.java

MiaoShaGoodsServiceImpl类是MiaoShaGoodsService接口的实现类，提供了秒杀商品相关的服务。在秒杀场景中，秒杀商品服务用于获取当前可以秒杀的商品列表、查询秒杀商品信息和更新秒杀商品信息。

* 获取当前可以秒杀的商品列表：
  + 通过listAllNow方法获取当前可以秒杀的商品列表。
  + 使用自定义的MiaoShaGoodsMapper接口实现查询，与数据库交互。
* 根据商品ID查询秒杀商品信息：
  + 通过findById方法根据秒杀商品ID查询商品信息。
  + 先检查Redis缓存中是否存在秒杀商品信息，存在则从缓存中获取，否则从数据库中获取，并将结果存入Redis缓存。
* 更新秒杀商品信息：
  + 通过update方法更新秒杀商品信息。
  + 使用MyBatis Plus的updateById方法，与数据库交互。

这些操作通过Spring的@Service注解提供服务，与MyBatis Plus集成，通过MiaoShaGoodsMapper与数据库交互。同时，通过RedisUtil工具类实现秒杀商品信息的缓存，提高查询效率。这样的设计符合秒杀场景下秒杀商品服务的需求。



1. OrderServiceImpl.java

OrderServiceImpl类是OrderService接口的实现类，提供了订单相关的服务。在秒杀场景中，订单服务用于查询订单列表、添加订单、更新订单信息以及根据订单ID查询订单详情。

* 查询订单列表：
  + 通过list方法根据条件查询订单列表。
  + 使用自定义的OrderMapper接口实现查询，与数据库交互。
* 添加订单：
  + 通过add方法添加订单。
  + 使用自定义的OrderMapper接口实现插入订单，同时更新秒杀商品库存、商品库存，并将订单号放入延时消息队列。
* 更新订单信息：
  + 通过update方法更新订单信息。
  + 使用MyBatis Plus的updateById方法，与数据库交互。
* 根据订单ID查询订单详情：
  + 通过findById方法根据订单ID查询订单详情。
  + 使用MyBatis Plus的selectById方法，与数据库交互。
* 根据用户ID和秒杀商品ID查询订单：
  + 通过findByUserIdAndMiaoShaGoodsId方法根据用户ID和秒杀商品ID查询订单。
  + 使用自定义的OrderMapper接口实现查询，与数据库交互。

这些操作通过Spring的@Service注解提供服务，与MyBatis Plus集成，通过OrderMapper与数据库交互。同时，通过GoodsMapper和MiaoShaGoodsMapper更新商品和秒杀商品的库存，并通过RabbitMQProducerService将订单号放入延时消息队列。这样的设计符合秒杀场景下订单服务的需求。



### Mybatis代码

1. GoodsMapper.xml

这是一个MyBatis的XML映射文件，对应于GoodsMapper接口，提供了商品相关的数据库操作。以下是对该XML文件的总结：

* ResultMap定义：
  + 定义了名为"BaseResultMap"的ResultMap，映射了Goods实体类的属性与数据库表字段的关系。
* 查询商品详情：
  + 通过findById查询商品详情，使用id作为参数，返回一个Goods实体对象。
  + SQL查询语句为从t\_goods表中选择所有字段，通过id条件进行筛选。
* 减少商品库存：
  + 通过reduceStock更新商品库存，使用id作为参数。
  + SQL更新语句为将t\_goods表中对应商品的stock字段减去1。

这个XML文件与秒杀场景结合，主要用于秒杀成功后减少商品库存。当有用户成功秒杀商品时，会调用reduceStock更新商品表中对应商品的库存，确保库存正确减少。这样的设计符合秒杀场景中对商品库存的实时控制需求。



1. MiaoShaGoodsMapper.xml

这是一个MyBatis的XML映射文件，对应于MiaoShaGoodsMapper接口，提供了秒杀商品相关的数据库操作。以下是对该XML文件的总结：

* ResultMap定义：
  + 定义了名为"BaseResultMap"的ResultMap，映射了MiaoShaGoods实体类的属性与数据库表字段的关系。同时，通过association关联了Goods实体类，实现了goods属性的级联查询。
* 查询当前可秒杀商品列表：
  + 通过listAllNow查询当前可秒杀商品列表，返回一个MiaoShaGoods实体对象的集合。
  + SQL查询语句为从t\_miaosha\_goods表中选择所有字段，筛选条件为秒杀开始时间早于等于当前时间、结束时间晚于等于当前时间、库存大于0，并按照开始时间降序排列。
* 查询秒杀商品详情：
  + 通过findById查询秒杀商品详情，使用id作为参数，返回一个MiaoShaGoods实体对象。
  + SQL查询语句为从t\_miaosha\_goods表中选择所有字段，通过id条件进行筛选。
* 减少秒杀商品库存：
  + 通过reduceStock更新秒杀商品库存，使用id作为参数。
  + SQL更新语句为将t\_miaosha\_goods表中对应秒杀商品的stock字段减去1，同时确保库存大于0。

这个XML文件与秒杀场景结合，主要用于查询当前可秒杀商品列表、查询秒杀商品详情以及秒杀成功后减少秒杀商品库存。当有用户成功秒杀商品时，会调用reduceStock更新秒杀商品表中对应商品的库存，确保库存正确减少。同时，通过级联查询，获取秒杀商品对应的实际商品信息。这样的设计符合秒杀场景中对商品库存和详情的实时控制需求。

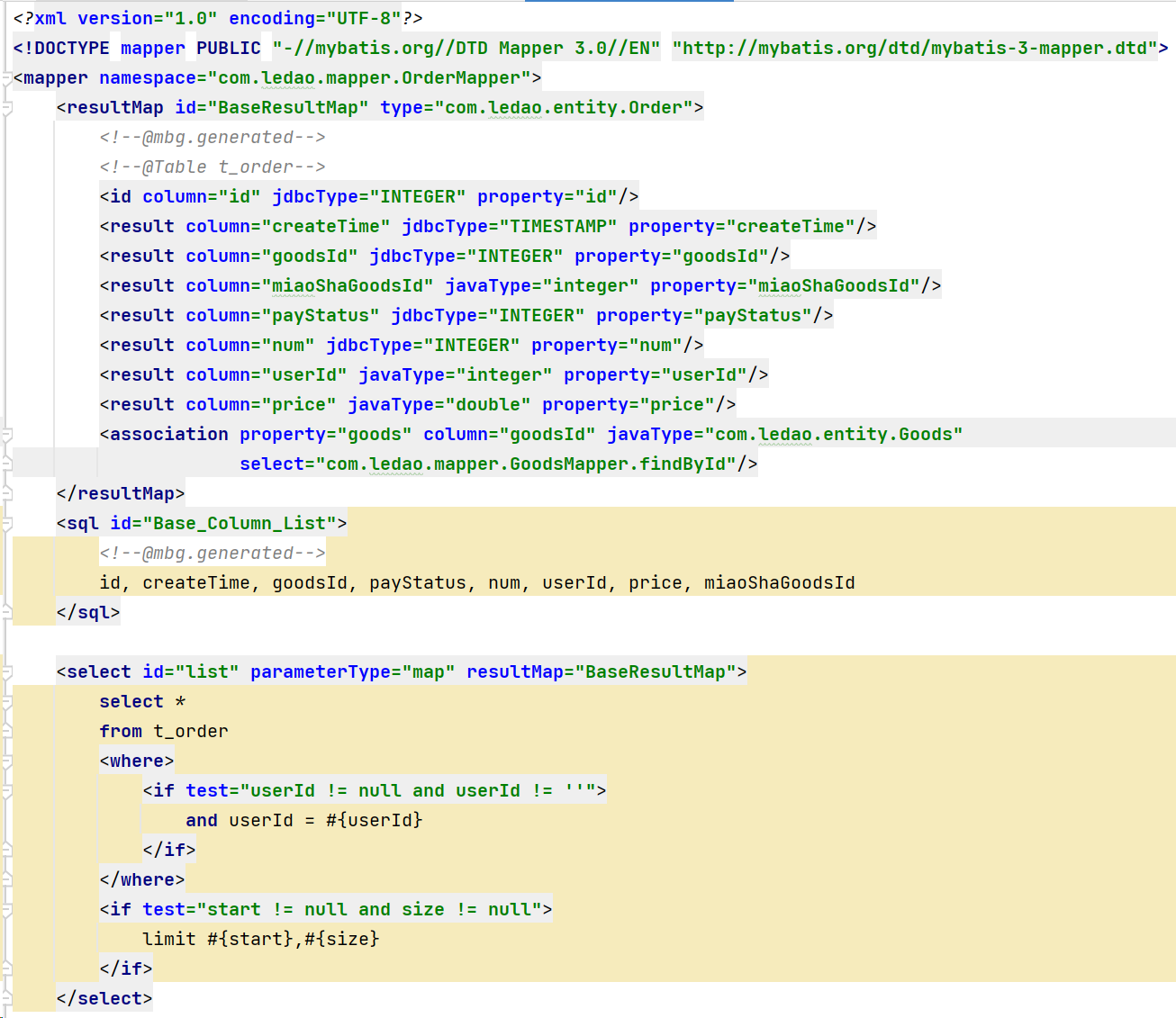


1. OrderMapper.xml

这是一个MyBatis的XML映射文件，对应于OrderMapper接口，提供了订单相关的数据库操作。以下是对该XML文件的总结：

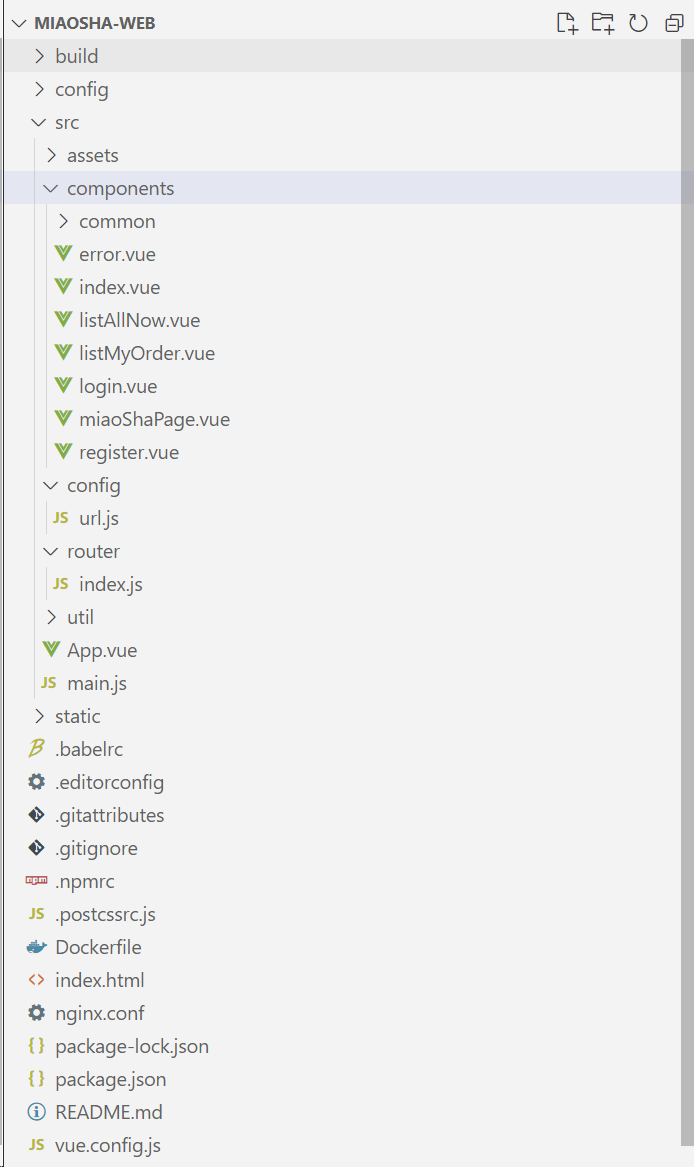
* ResultMap定义：
  + 定义了名为"BaseResultMap"的ResultMap，映射了Order实体类的属性与数据库表字段的关系。同时，通过association关联了Goods实体类，实现了goods属性的级联查询。
* 查询订单列表：
  + 通过list查询订单列表，使用map作为参数，返回一个Order实体对象的集合。
  + SQL查询语句为从t\_order表中选择所有字段，通过userId条件进行筛选，同时可以根据start和size进行分页查询。
* 根据用户ID和秒杀商品ID查询订单：
  + 通过findByUserIdAndMiaoShaGoodsId查询订单，使用map作为参数，返回一个Order实体对象。
  + SQL查询语句为从t\_order表中选择所有字段，通过userId和miaoShaGoodsId条件进行筛选。
* 添加订单：
  + 通过add添加订单，使用Order实体类作为参数，同时设置了主键自动生成和返回。
  + SQL插入语句为向t\_order表中插入订单信息，包括创建时间、商品ID、支付状态、购买数量、用户ID、价格和秒杀商品ID。

这个XML文件与秒杀场景结合，主要用于处理订单的查询和添加操作。通过查询订单列表和根据用户ID和秒杀商品ID查询订单，实现了秒杀场景下对订单的管理。在添加订单时，会记录订单的各项信息，包括购买的商品ID、支付状态、购买数量、用户ID、价格以及秒杀商品ID。这样的设计满足了秒杀场景下对订单数据的实时更新和查询需求。



## 前端代码

### 前端代码结构



### 关键界面代码

1. 登陆界面login.vue

登录页面提供了用户登录的功能，用户可以通过输入正确的用户名和密码完成登录流程，同时确保输入信息的合法性，提高系统的安全性和用户体验。



1. 注册界面register.vue

通过这个页面，用户可以完成注册流程，确保注册信息的合法性和唯一性，提高系统的安全性和用户体验。



1. 主界面index.vue

这个页面的设计使得用户能够方便地浏览正在秒杀的商品、查看自己的订单，并提供了友好的注销功能，保障了用户在秒杀系统中的基本操作需求。



1. 秒杀界面listAllNow.vue

这个页面是秒杀商品的列表页面，提供了以下主要功能：

* 秒杀商品列表：
  + 显示了秒杀商品的图片、名称、原价、秒杀价、开始时间等信息。
  + 为每个商品提供了“去秒杀”按钮，用户点击可查看该商品的秒杀详情。
* 秒杀详情弹窗：
  + 弹窗包含了秒杀商品的详细信息，如商品图片、名称、原价、秒杀价、秒杀时间等。
  + 在秒杀进行中，显示剩余时间、抢购状态和操作按钮。
  + 用户可以点击“抢购”按钮进行商品抢购操作。
* 秒杀倒计时：
  + 在秒杀详情弹窗中显示了距离秒杀结束的倒计时。
  + 当倒计时为0时，秒杀状态改变为已结束，用户无法再进行抢购。
* 商品抢购操作：
  + 用户点击“抢购”按钮后，系统进行抢购操作。
  + 抢购成功后，提示用户排队中，等待支付。
  + 抢购失败或商品已售罄时，给予相应提示。
* 订单状态检查：
  + 系统定时检查用户抢购的订单状态。
  + 若抢购成功，提示用户支付；若抢购失败，提示用户未成功抢购。

通过这些功能，用户可以浏览当前秒杀商品列表，选择感兴趣的商品查看详情，进行商品的抢购，并实时了解秒杀状态和剩余时间。



1. 订单界面listMyOrder.vue

这个页面是用户个人订单列表页面，主要功能如下：

* 订单列表：
  + 显示了用户个人的订单信息，包括商品图片、商品名称、价格、订单创建时间和支付状态。
  + 提供了“支付”操作按钮，允许用户对未支付的订单进行支付。
* 支付确认弹窗：
  + 用户点击“支付”按钮后，弹出支付确认弹窗，询问用户是否要进行支付。
  + 用户可以选择“是”进行支付，或者“否”取消支付操作。
* 支付操作：
  + 用户确认支付后，系统进行支付操作，并更新订单支付状态。
  + 若支付成功，页面提示支付成功，并刷新订单列表。
* 订单状态标签：
  + 在订单列表中显示订单的支付状态标签，包括“已支付”和“30分钟未支付自动取消”等状态。

通过这些功能，用户可以方便地查看个人订单情况，及时进行支付操作，以完成购物流程。



1. 错误处理界面error.vue

这个界面用于处理请求链接不存在的情况，自动返回上一个页面。



## 主要功能实现

### Redis缓存



### RabbitMQ处理用户的秒杀请求

1. RabbitMQ生产者实现机制

这是一个基于RabbitMQ的消息生产者服务实现，提供了两个主要功能，分别是发送即时信息和发送延时消息。以下是对该实现类的详细介绍，同时结合秒杀场景：

* 发送即时信息 (sendInformation 方法)：
  + 通过注入的AmqpTemplate对象，使用convertAndSend方法将消息发送到指定的Direct Exchange（直连交换机）和Routing Key（路由键）。
  + 在秒杀场景中，这个方法可能用于发送即时的秒杀信息，通知相关系统组件进行相应的处理。例如，在用户成功秒杀商品后，可以通过此方法发布消息，告知订单系统生成相应订单。
* 发送延时消息 (sendMessageDelayed 方法)：
  + 通过注入的AmqpTemplate对象，同样使用convertAndSend方法，但在这里设置了消息的延时属性。
  + 在秒杀场景中，这个方法可以用于处理一些延时任务，例如订单支付超时未支付的处理。通过将订单信息发送到带有延时特性的交换机，系统可以在一定时间后再进行处理，例如取消订单。
* 结合秒杀场景：
  + 延时消息的使用在秒杀场景中尤为重要。例如，用户成功秒杀后，可以通过延时消息来触发订单的支付检查，以确保订单在一定时间内完成支付。
  + 另外，延时消息也可用于处理库存的回滚操作。如果用户在一定时间内未支付，可以通过延时消息通知系统回滚相应商品的库存。

总体而言，这个RabbitMQ生产者服务实现提供了在秒杀场景中处理即时和延时消息的功能，以支持相关业务逻辑的实时处理和调度。



1. RabbitMQ消费者实现机制

这是一个基于RabbitMQ的消息消费者服务实现，用于处理秒杀场景中的相关请求和延时消息。以下是对该实现类的详细介绍，同时结合秒杀场景：

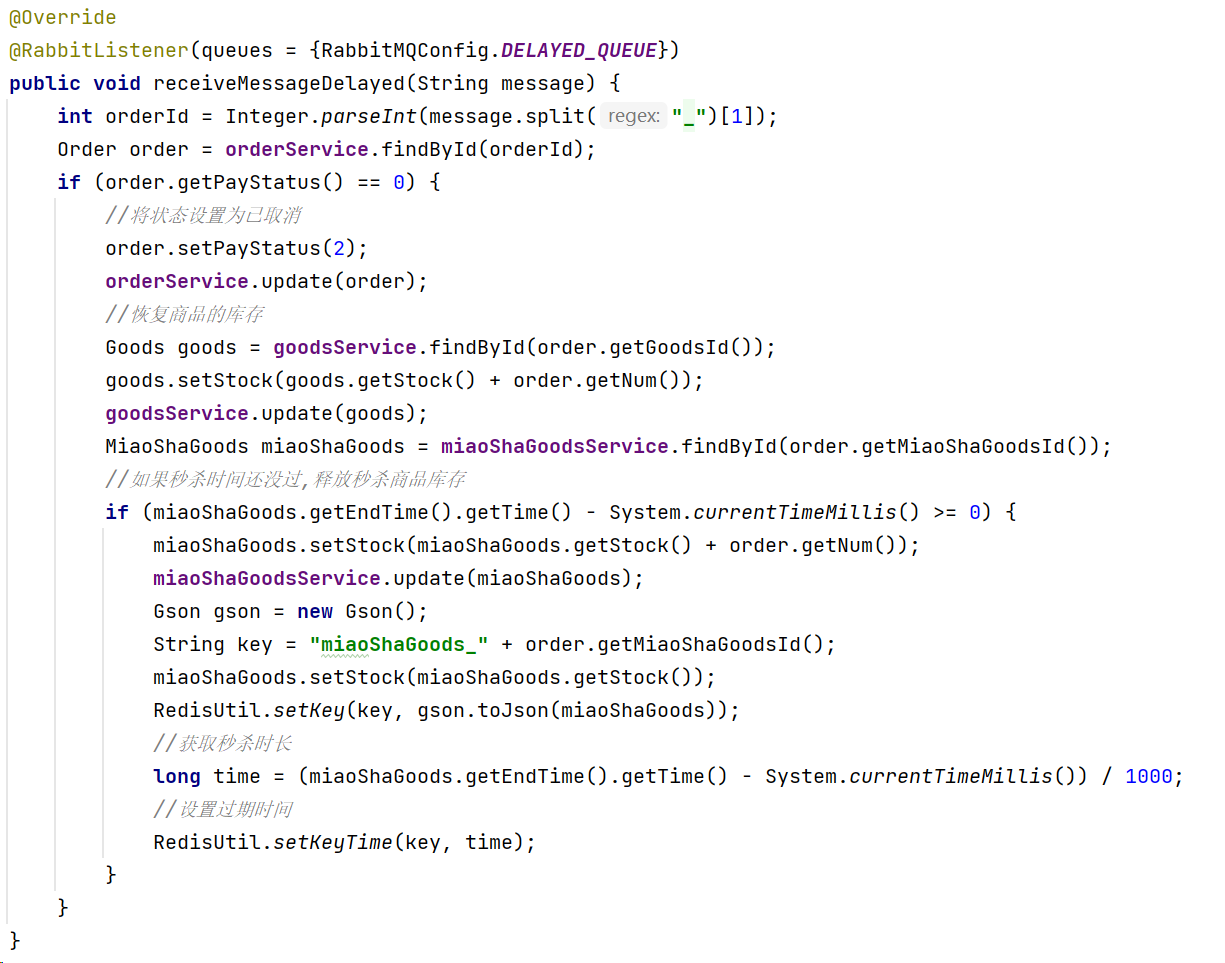
handleMiaoShaRequst方法详细介绍：

* @RabbitListener 注解：
  + 通过该注解，将方法标记为RabbitMQ的消息监听器。指定监听的队列为RabbitMQConfig.DIRECT\_QUEUE，即处理秒杀请求的队列。
* 消息处理逻辑：
  + 打印接收到的消息，用于调试和记录日志。
  + 使用Gson将接收到的消息字符串反序列化为OrderVo对象，其中包含了秒杀订单的相关信息。
  + 创建Map对象，存储用户ID和秒杀商品ID，用于查询用户是否已经秒杀过该商品。
  + 查询数据库获取秒杀商品信息，判断秒杀商品库存是否大于0。
  + 如果条件满足，判断用户是否已经秒杀过该商品，若没有则创建新订单。
* 业务逻辑说明：
  + 该方法主要处理即时的秒杀请求，确保在秒杀商品有库存的情况下，用户可以成功下单，并防止用户多次秒杀同一商品。



handleMiaoShaRequst方法详细介绍：

* @RabbitListener注解：
  + 同样通过该注解，将方法标记为RabbitMQ的消息监听器。指定监听的队列为RabbitMQConfig.DELAYED\_QUEUE，即处理延时消息的队列。
* 消息处理逻辑：
  + 解析消息中的订单ID，获取相应订单的信息。
  + 判断订单支付状态是否为未支付（0），如果是，则执行取消订单的逻辑。
  + 将订单状态设置为已取消（2），并回滚相关商品和秒杀商品的库存。
  + 如果秒杀商品的秒杀时间仍未过，释放相应秒杀商品的库存。
* 业务逻辑说明：该方法处理了延时消息，用于取消

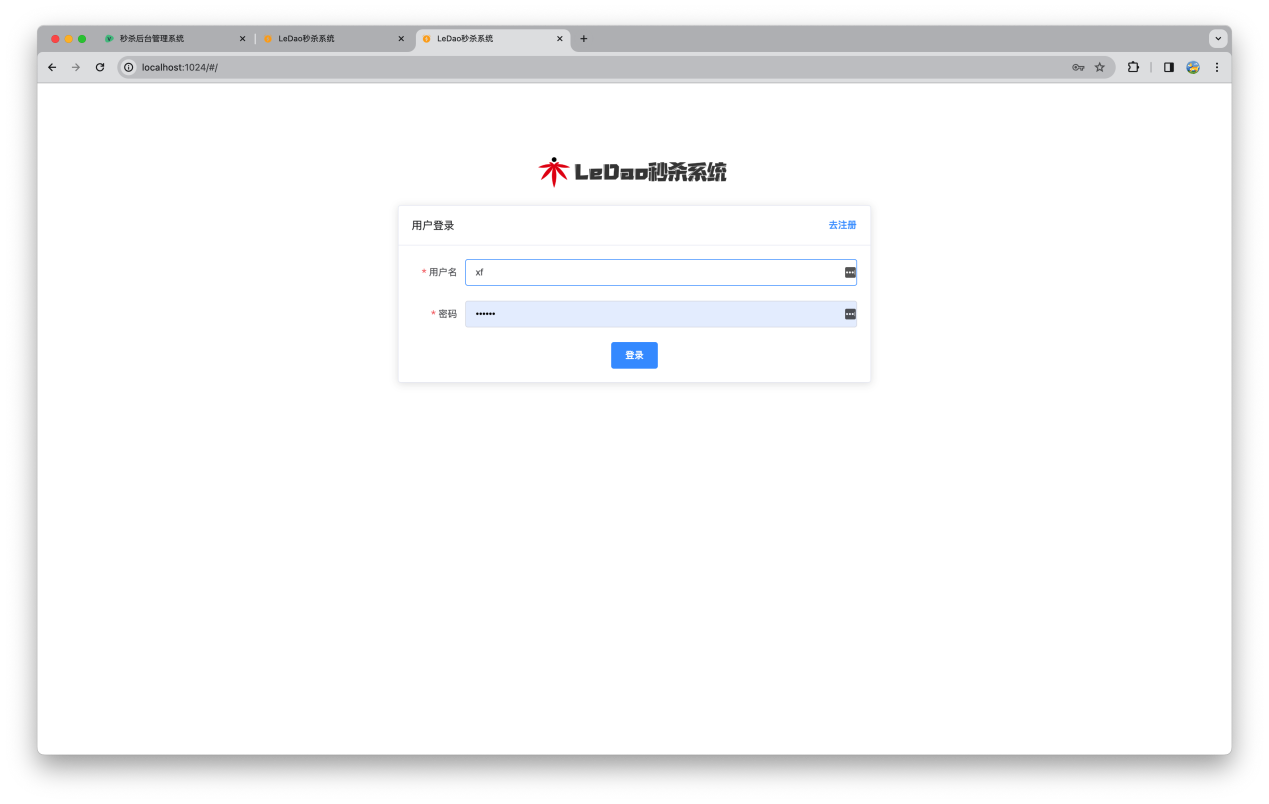


这个RabbitMQ消费者服务实现提供了在秒杀场景中处理即时和延时消息的功能，以支持相关业务逻辑的实时处理和调度。

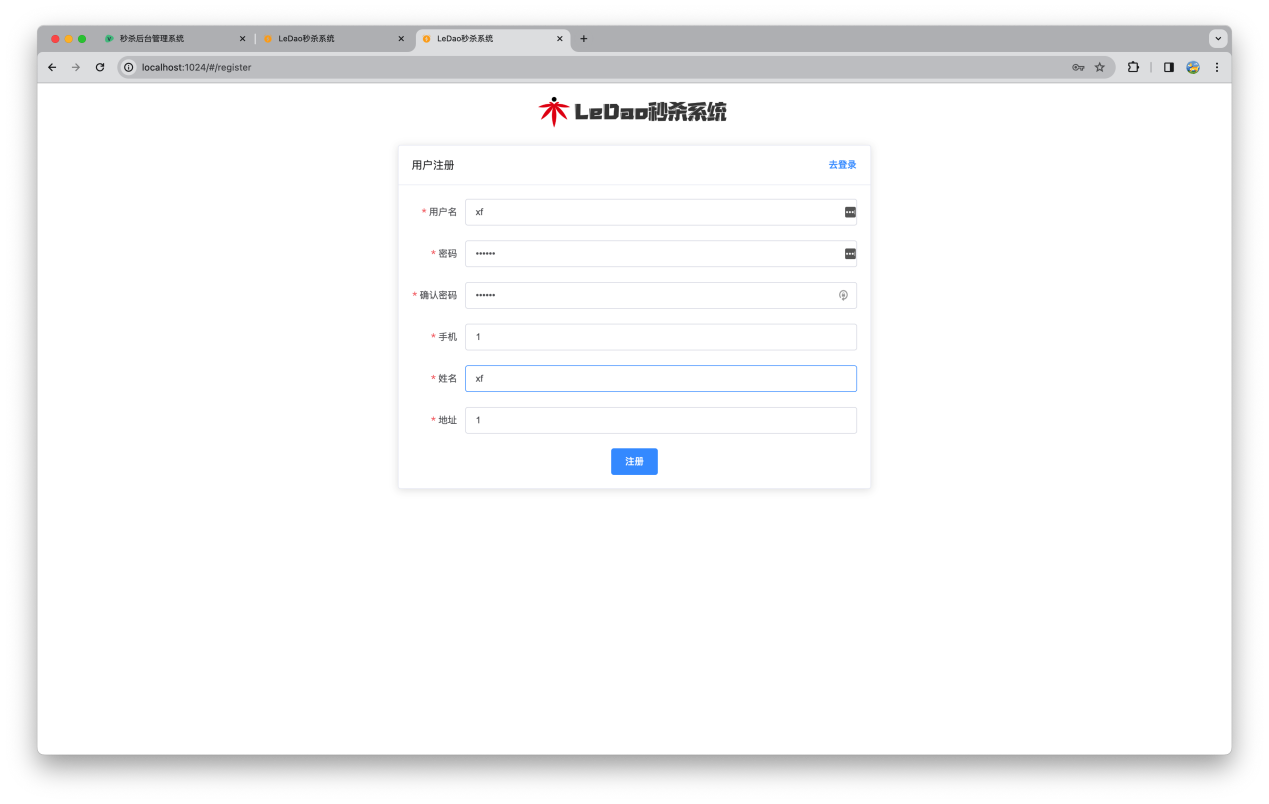
# 秒杀系统运行效果

## 用户秒杀系统

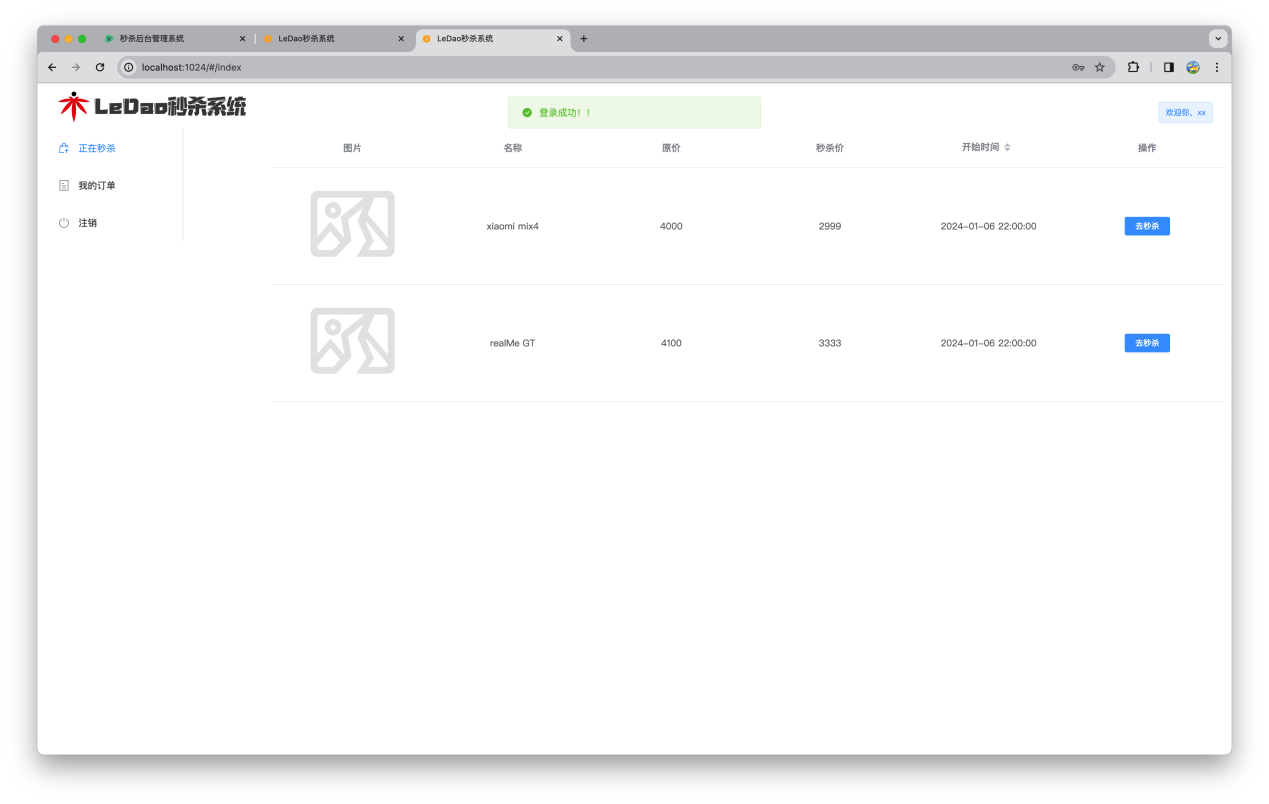
1. 用户登录



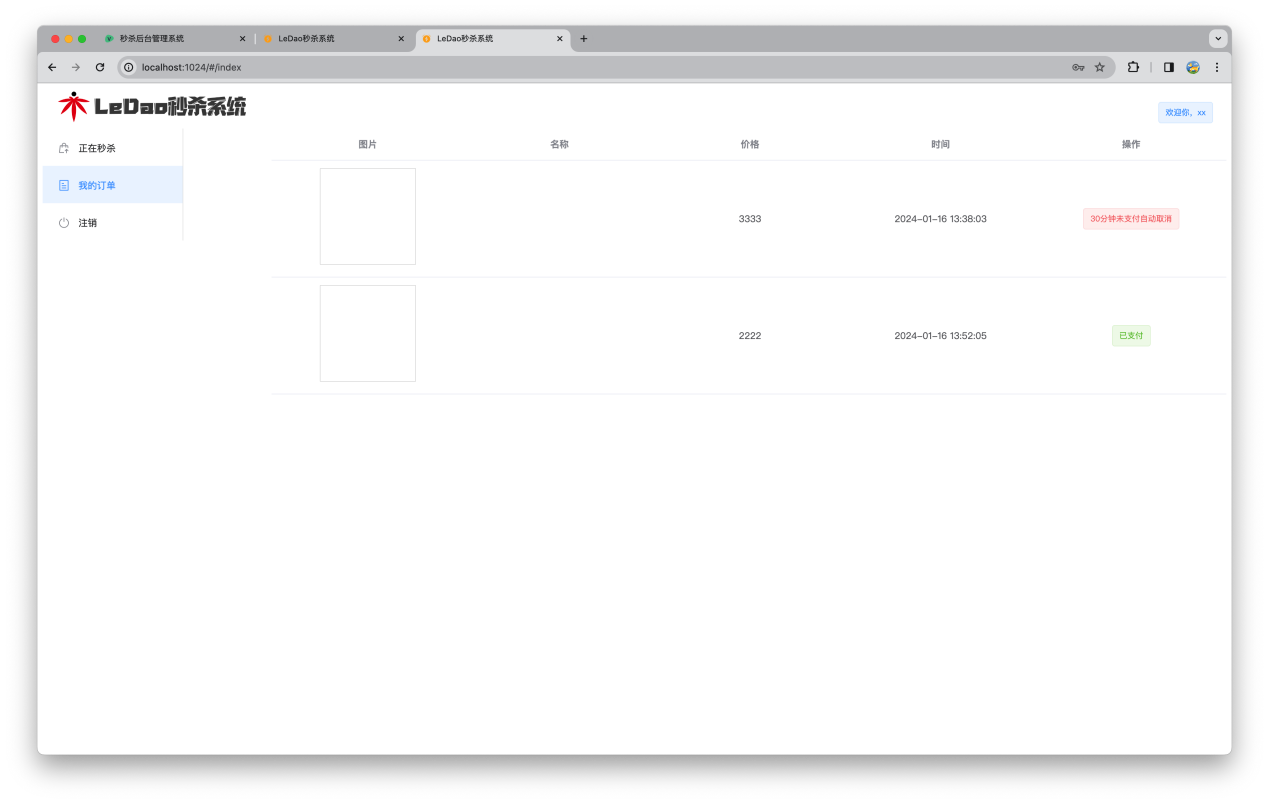
1. 用户注册



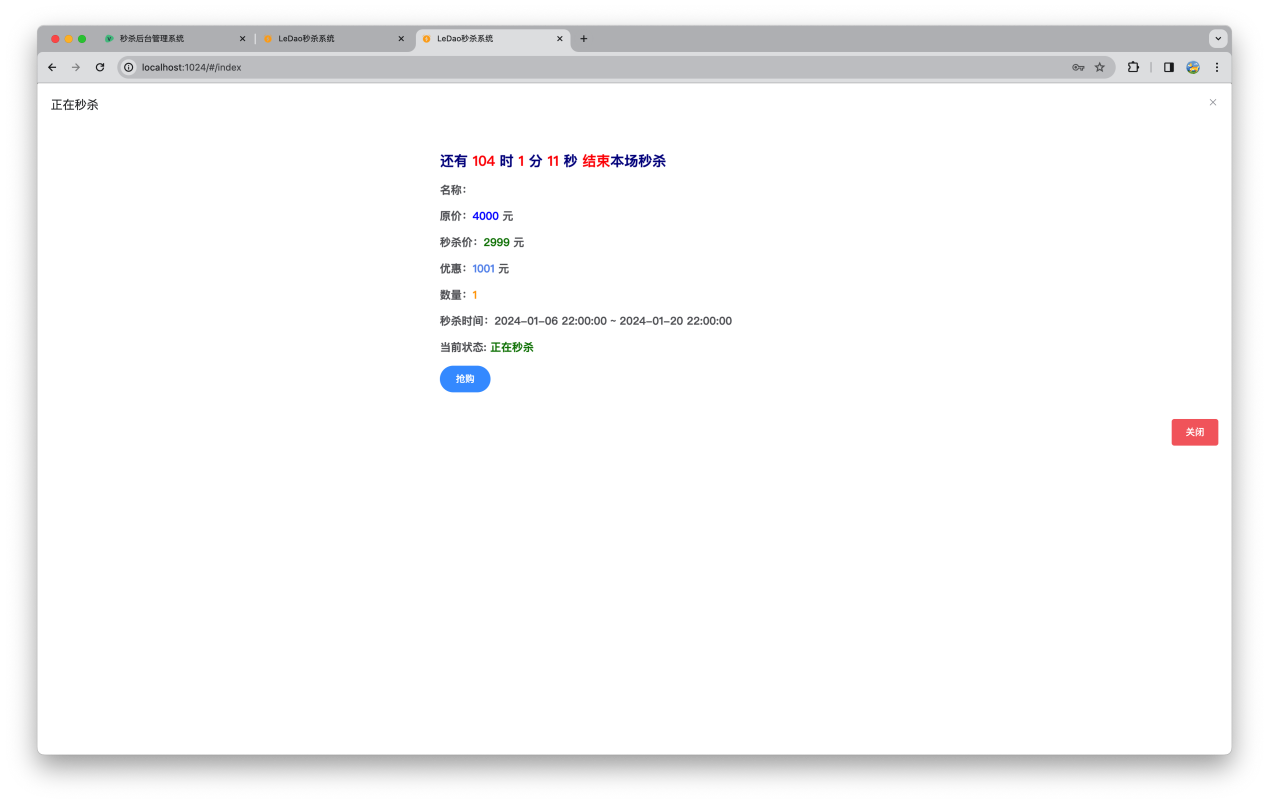
1. 登陆成功主页面



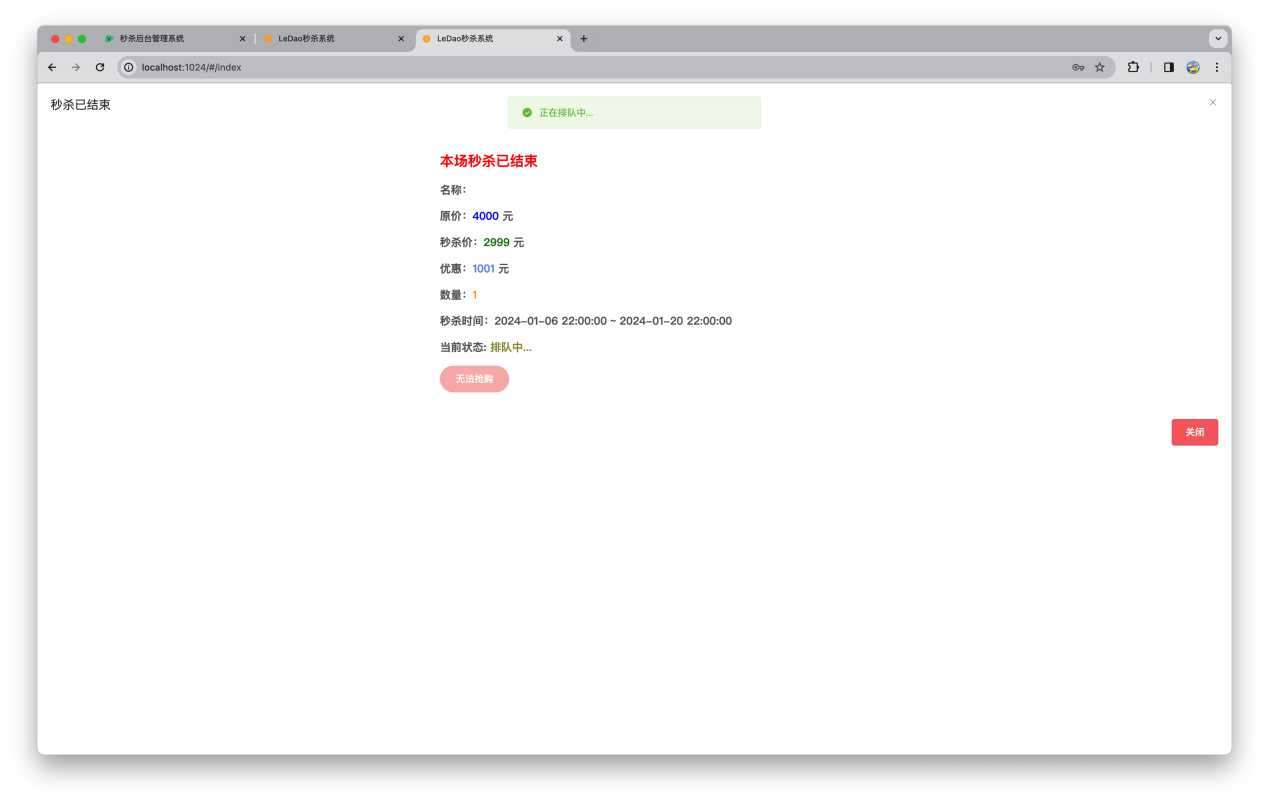
1. 我的订单界面



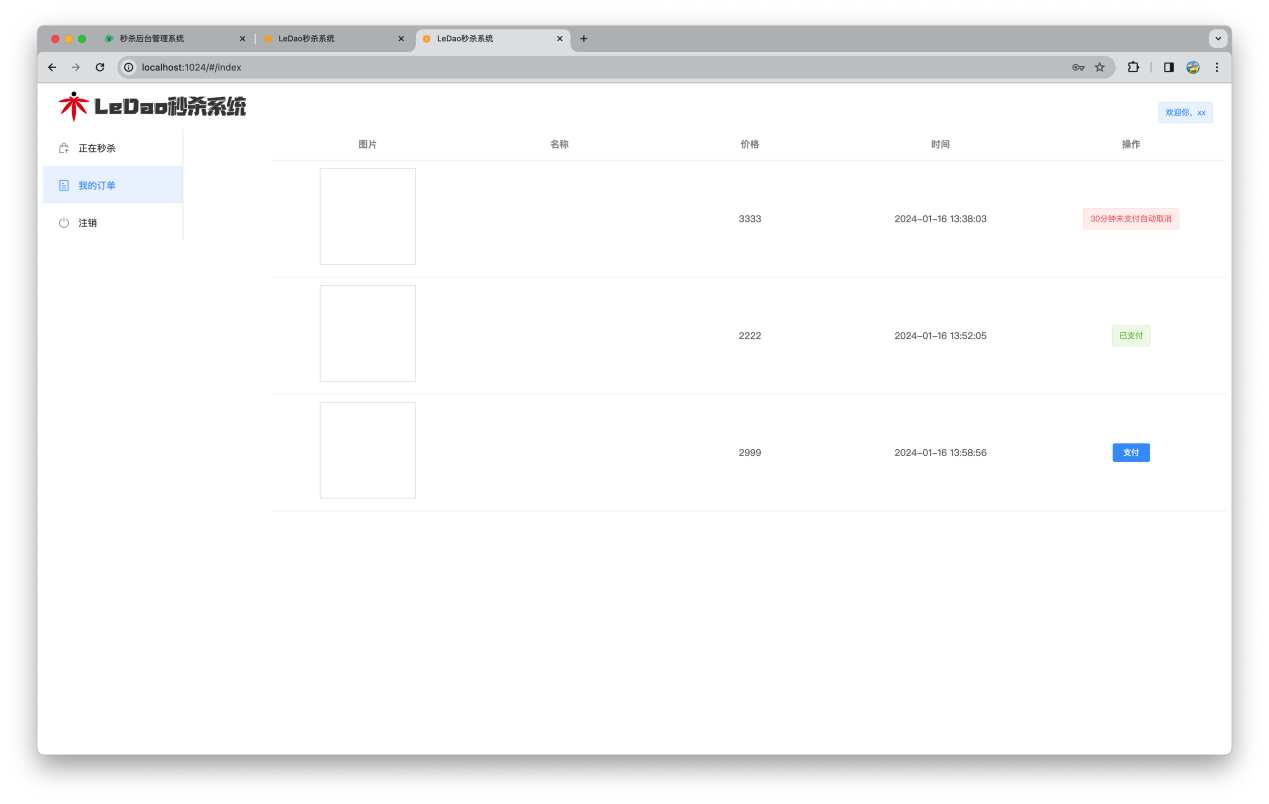
1. 正在秒杀

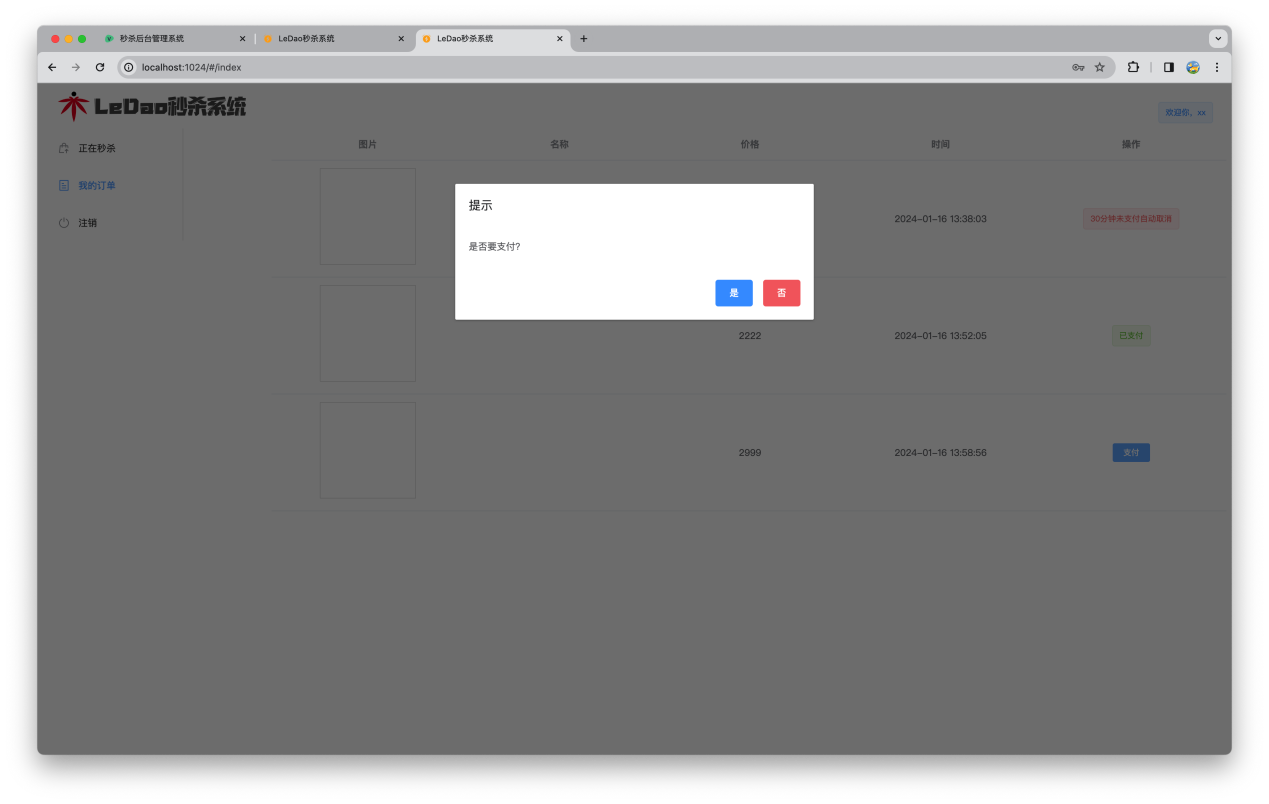


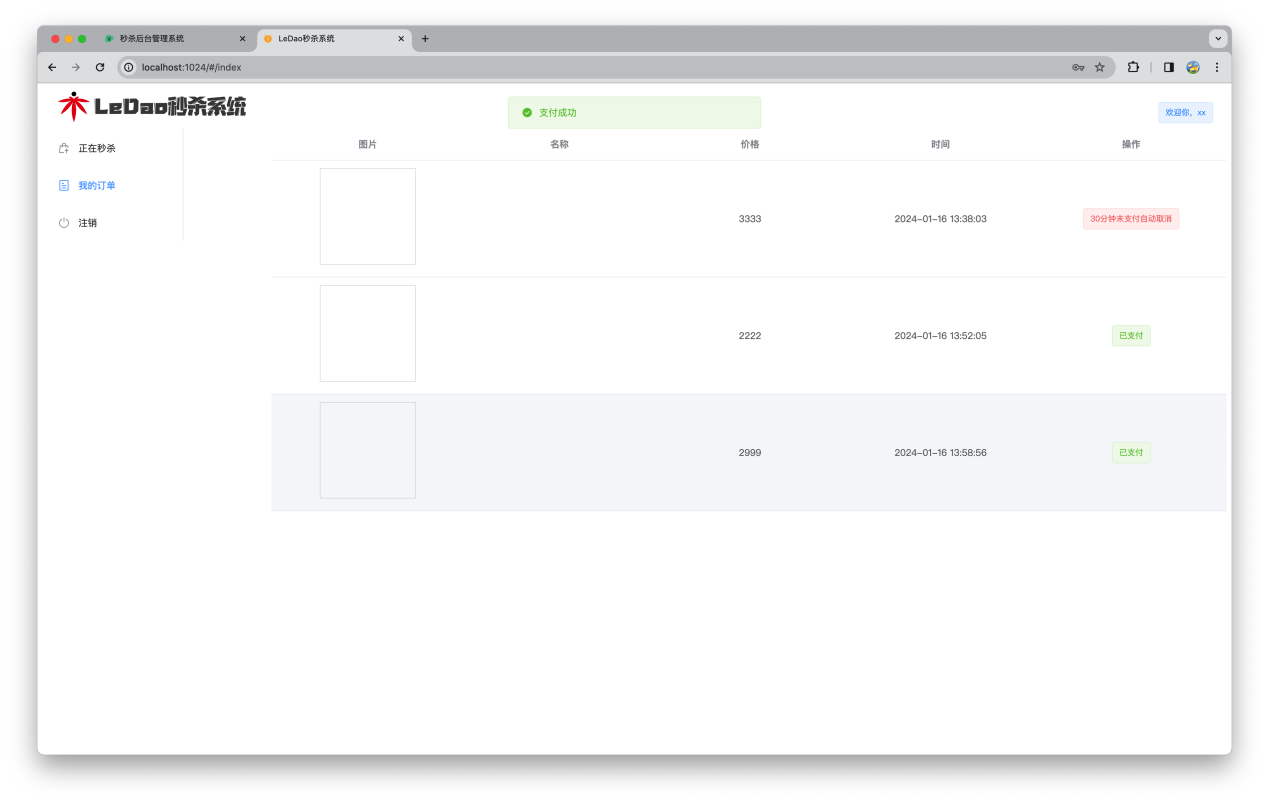
1. 秒杀失败（秒杀已结束）



1. 支付订单，秒杀成功

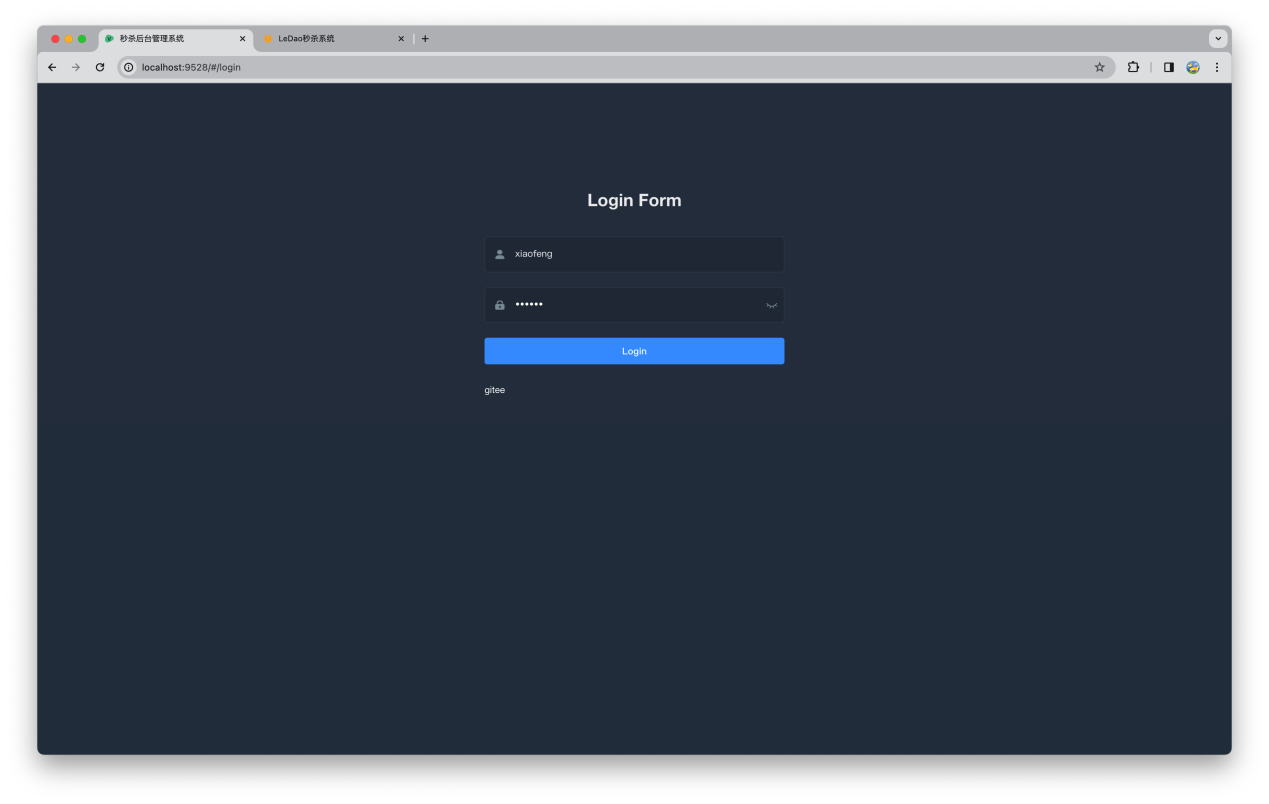




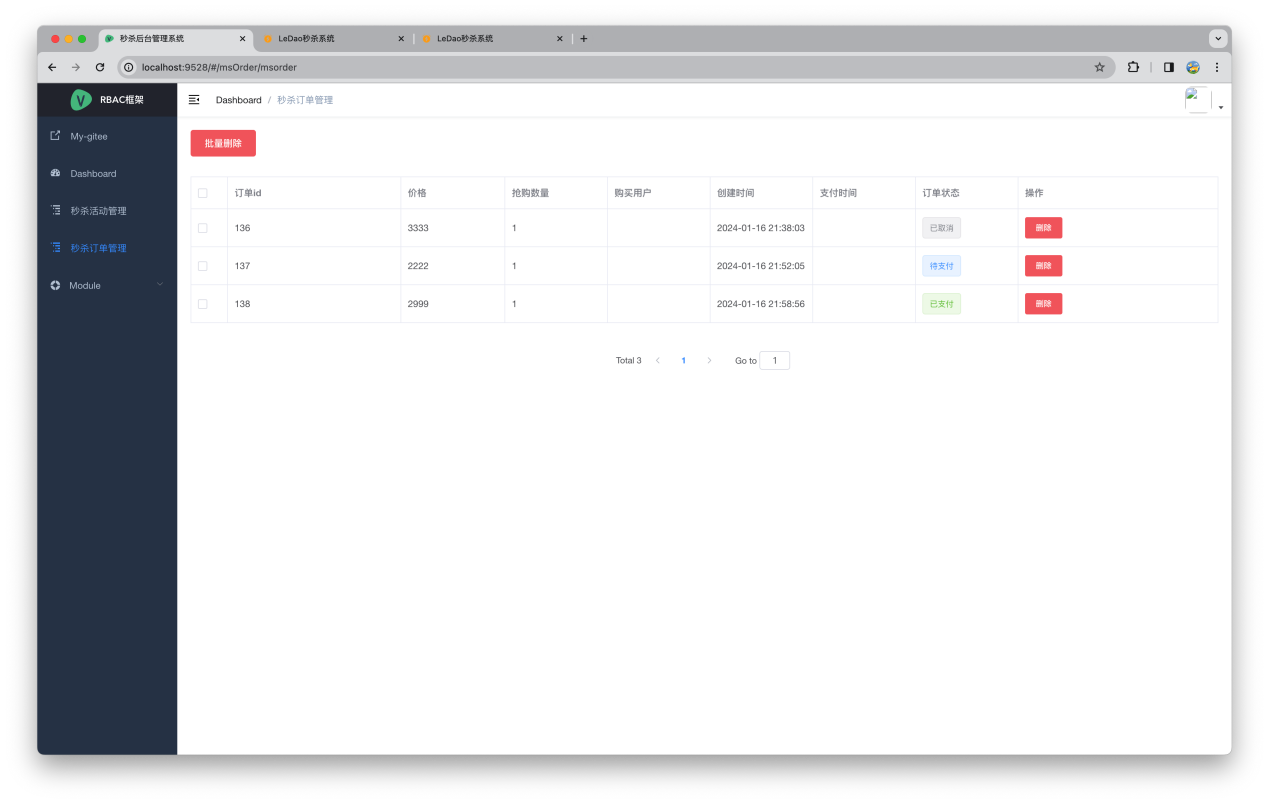


## 秒杀后台管理系统

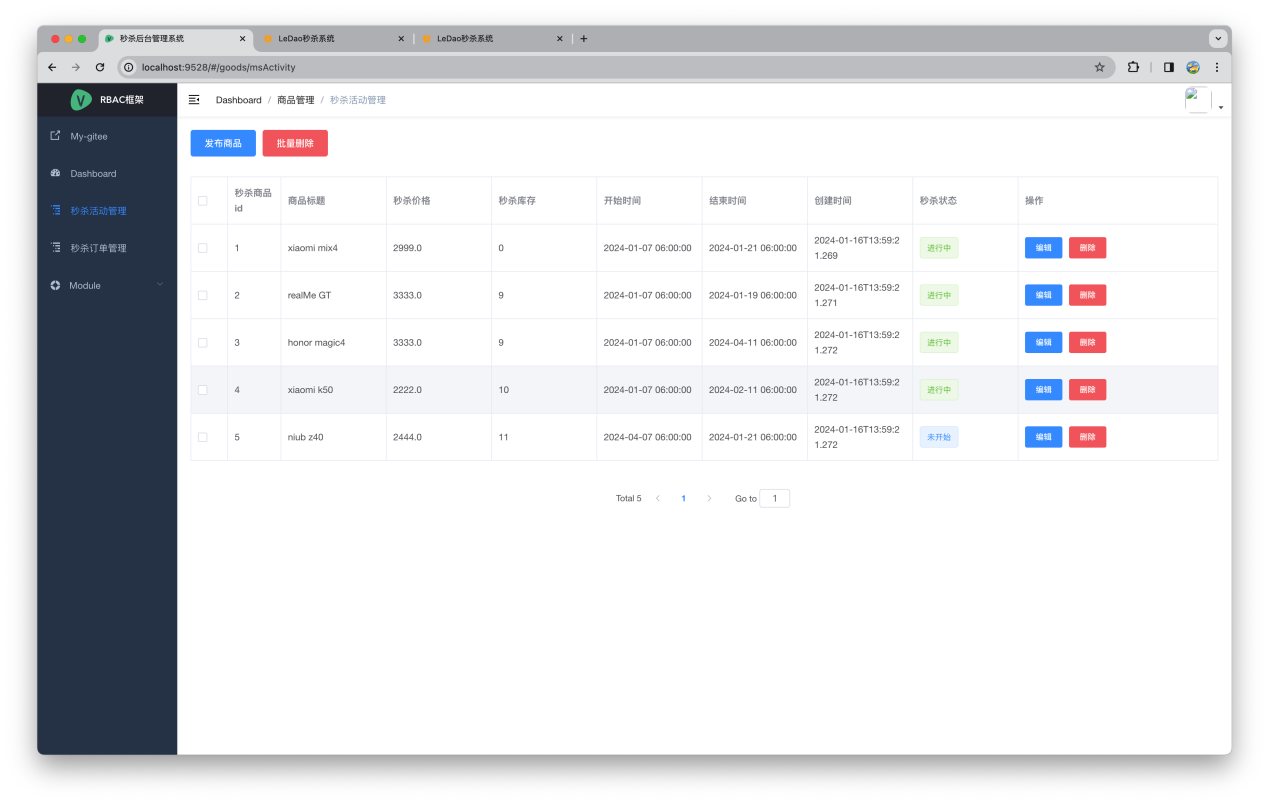
1. 登录后台管理系统



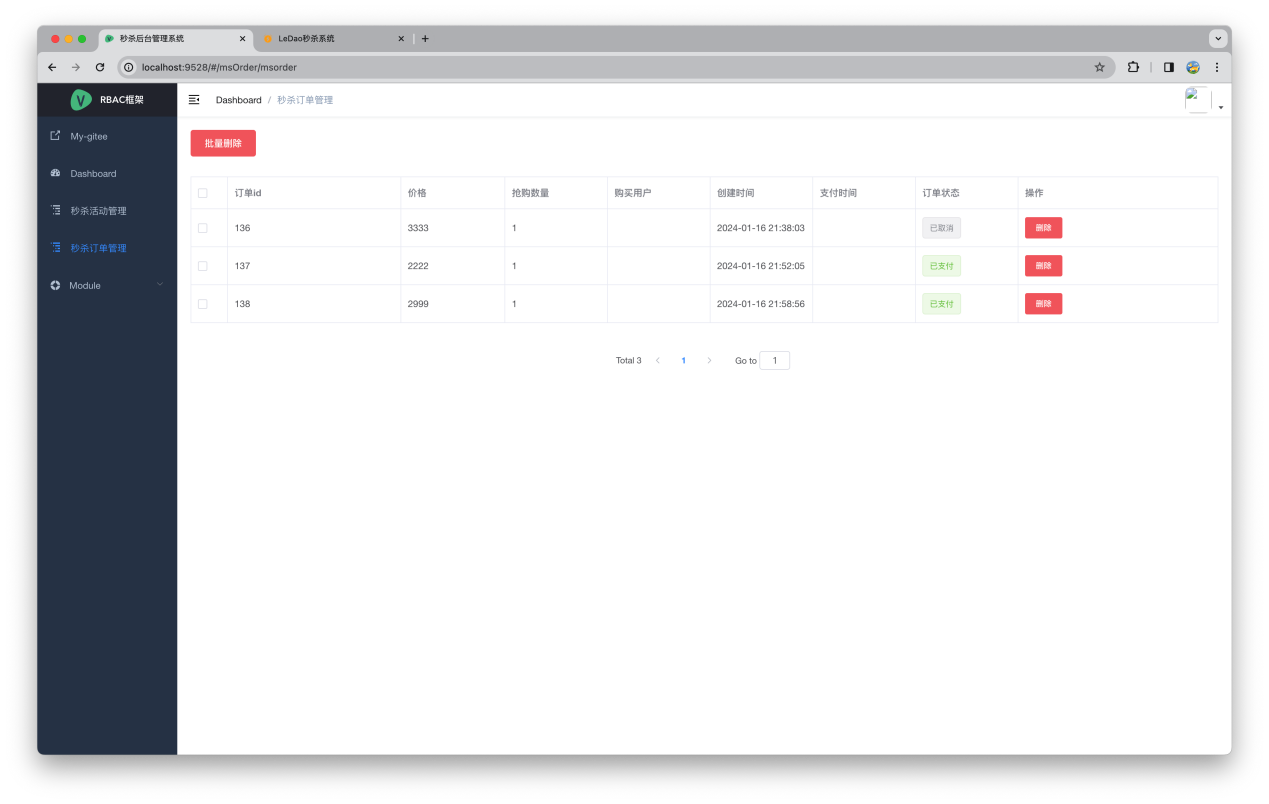
1. 秒杀订单管理



1. 秒杀活动管理

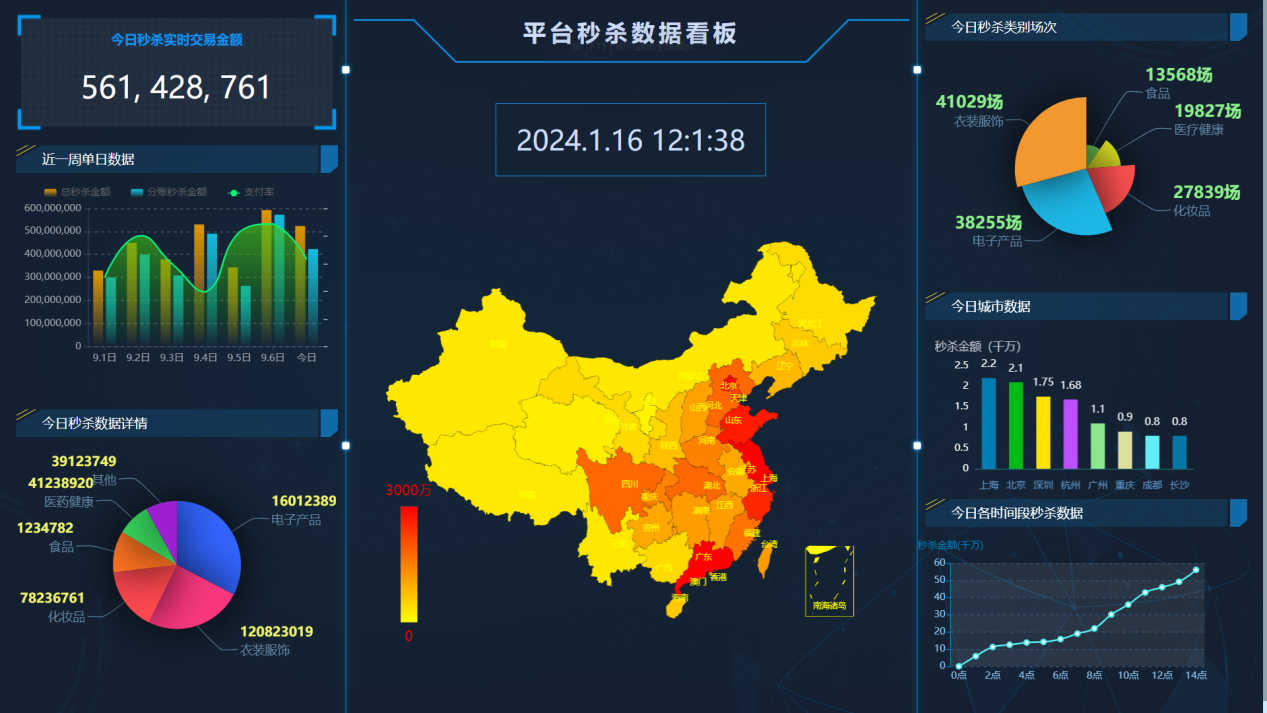


1. 已支付订单



## 后台管理系统数据展示

下面是后台管理的数据大屏看板，为了更好的展示看板的效果，我们在数据库中自己填充了相关信息。在最中间是全国各地的秒杀金额热力图，从图中可以看到今日的秒杀北京地区较为火爆，总订单额达到了2200万。同时在最左边有实时的订单交易金额，还有近一周的单日数据，以及支付率，还有今日秒杀数据的商品类别占比，以及城市数据统计，按照小时订单数据统计。

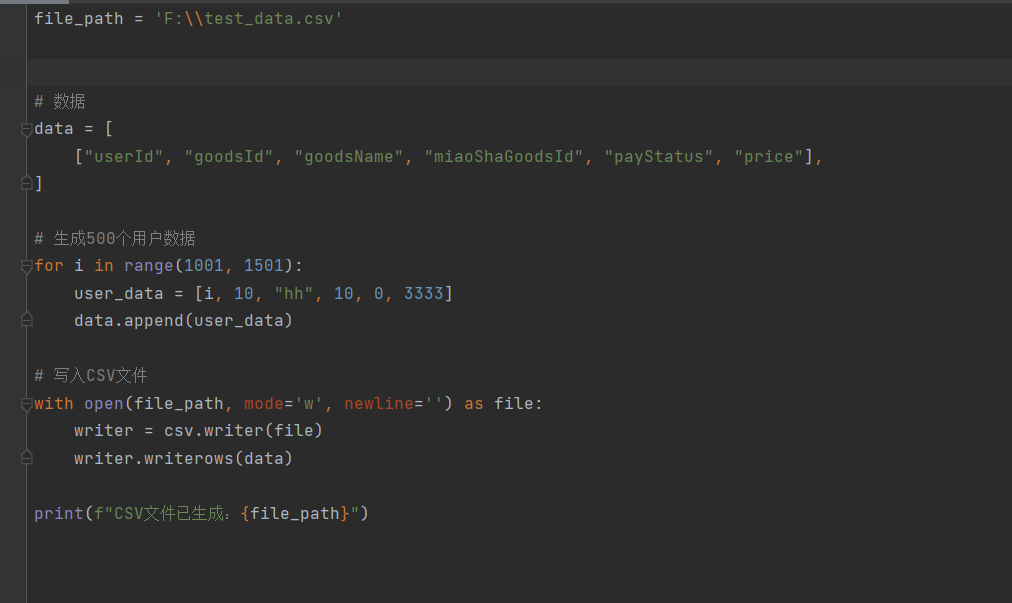


# 秒杀系统测试结果

## 秒杀接口的测试

1. 生成测试文件：

使用python脚本生成test\_data.csv文件，其中包含了用户id，商品id等参数。为了模拟真实的秒杀场景，使用500个不同用户（用户id为1001~1501）对库存为100的商品id为10的商品进行秒杀。



1. 创建测试计划：

打开JMeter，创建一个新的测试计划。在测试计划上右键单击，选择"Add" -> "Threads (Users)" -> "Thread Group"，配置线程组的属性，包括用户数和循环次数。

1. 添加Sampler：

在线程组上右键单击，选择"Add" -> "Sampler" -> 选择适当的 Sampler，比如HTTP请求。

1. 添加Listener：

在线程组上右键单击，选择"Add" -> "Listener" -> 选择适当的Listener，比如Summary Report和View Results Tree。

1. 配置CSV数据文件：

在JMeter中，右键单击测试计划，选择"Add" -> "Config Element" -> "CSV Data Set Config"。配置CSV Data Set Config元件，指定刚才创建的CSV文件的路径。

1. 使用变量引用：

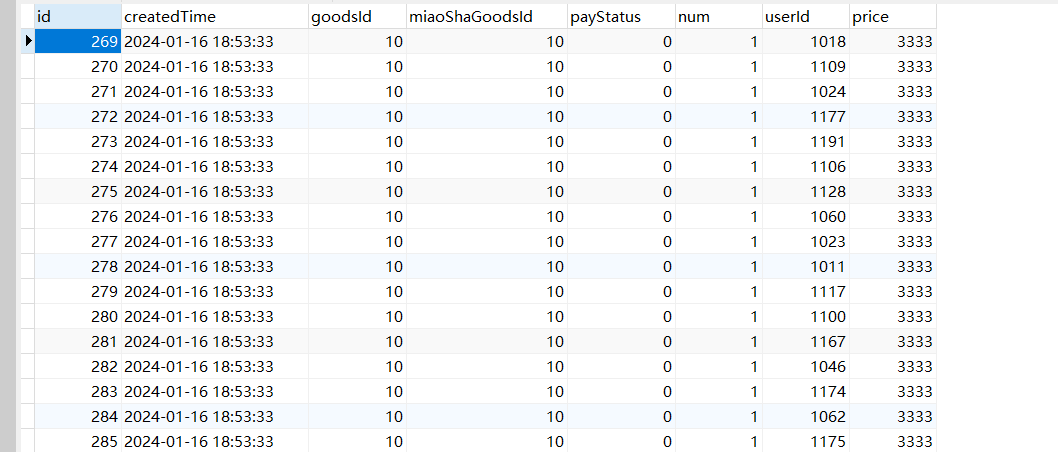
在HTTP请求中，使用${variable\_name}的形式引用CSV文件中的变量，例如${userId}。

1. 测试结果：

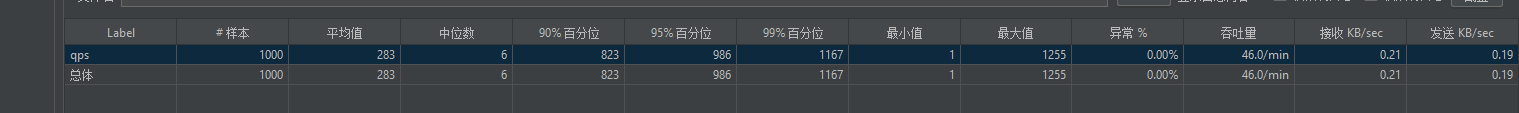
* 没有出现超卖现象：库存正常减为0，没有出现负数的情况



* 订单生成成功，一共100条订单数据，由100个不同用户抢到商品



* QPS的测试



## 其他部分的测试

1. 对Controller层进行单元测试：

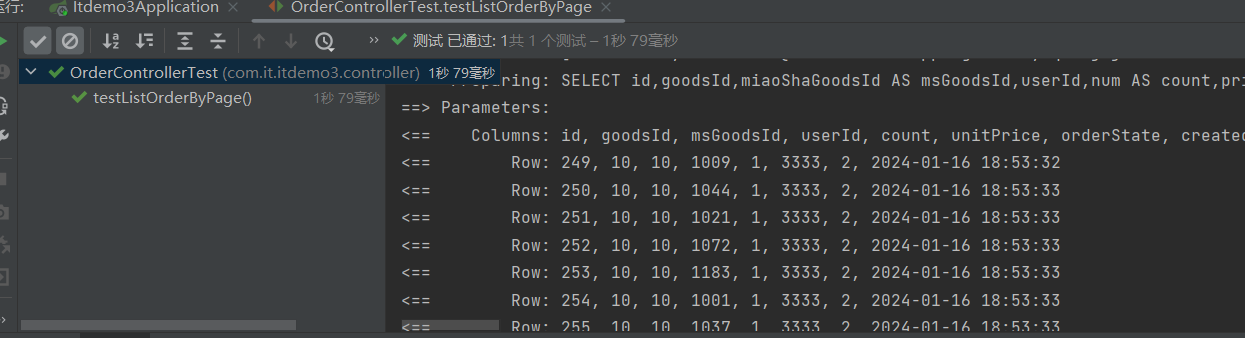
接口代码如下



在test中建立springBoot的test文件，对该接口进行测试

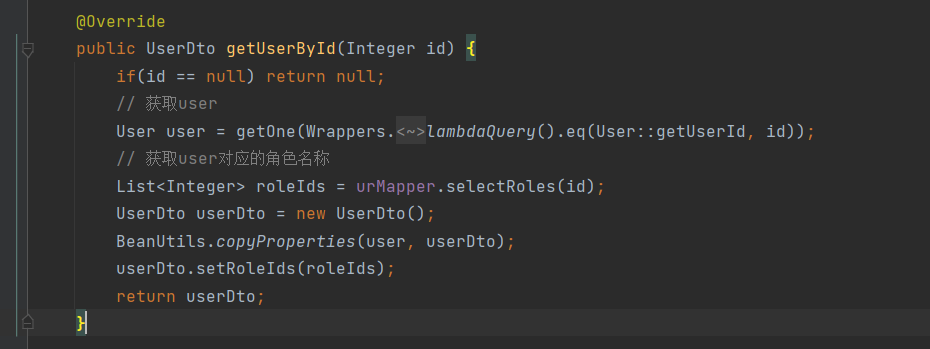


测试结果：该接口成功返回了分页查询的订单列表

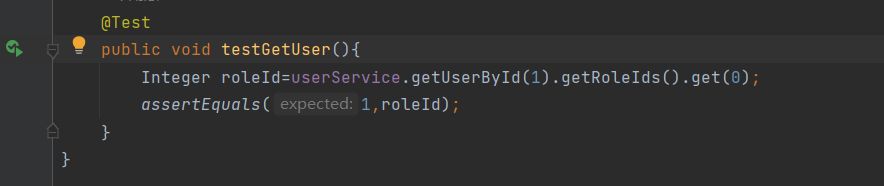


1. 对Service层进行单元测试：

service层的接口如下



在test中建立springBoot的test文件，对该接口进行测试



测试结果：成功的返回了用户对应的角色id

