

# Hw\_1 ThymeLeaf开发-Fi&Shipping 物流管理系统 系统设计报告

🕒

报告完成时间：2023-6-14 19:31

✍️

撰写人：-20231266 禹浩男

👥

小组成员：-20271006 付柏逢 -20231107 潘明豪

🔗

仓库链接：<https://github.com/laven00/trans>

👤

用户测试账号：pmh 密码: 123

👤

管理员测试账号：yhn 密码: 123

## #0 系统开发简介

homework1系统主要使用Thymeleaf和Springboot框架编写，辅以Mybatis进行数据库查询与操作。本项目在开发时是根据已经完成的作业2中的Vue框架下的物流管理系统重建而成。因时间比较仓促项目较为简单，但是涉及的相关技术栈与考察点齐全，望老师理解。

相较于运行在浏览器中的Vue框架，Thymeleaf的实现原理何其有明显不同。Thymeleaf在前后端的边界概念上和Vue有明显区别。在本项目中，展示在前端的整个页面都由后端生成，浏览器只提供展示的功能而不处理任何用户事件。将作业2中的Vue框架转化为Thymeleaf，需要将各个页面先转化为静态的Thymeleaf模板，使用表达式来绑定数据和模板，它提供了一些预定义的表达式，例如 `${...}`、`*{...}` 等。

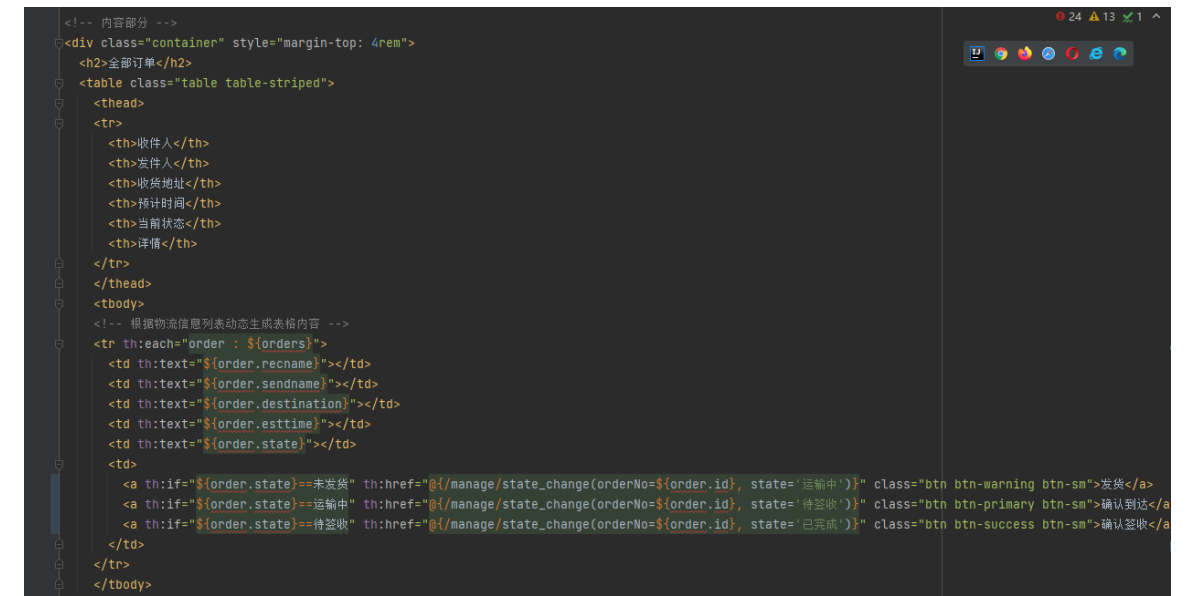


图0.1 部分模板类的编写

Controller层也需要进行相应的改变，首先要修改 Controller 类的注解[RESTful Controller](#) 通常使用 `@RestController` 或 `@Controller` 注解来标识，而 Thymeleaf Controller 则需要使用 `@Controller` 注解。其次需要修改 Controller 方法的返回类型，RESTful Controller 方法通常返回 JSON 数据或其他类型的响应数据，而 Thymeleaf Controller 则需要返回视图名称或模型对象。静态模板和Controller中的数据交互则使用Model类来进行，模板中获取到Model中的数据并“插入”到模板中，最终形成完整的html文件，返回给浏览器进行渲染展示。

```
@RequestMapping("/order/myOrder")
public String myOrder(Model model, HttpSession session){
    User user = (User) session.getAttribute("user");
    if(user!=null){
        String phone =user.getPhone();

        model.addAttribute("identity", user.getIdentity());
        model.addAttribute("receivedOrders",orderService.list_sporders(phone));
        model.addAttribute("sentOrders",orderService.list_rporders(phone));
        model.addAttribute("order",new Order());
        return "myOrder";
    }else{
        model.addAttribute("msg", "请登录");
        return "index";
    }
}
```

图0.2 myOrder Controller的改写

## #1 功能模块划分及描述

### 1.1 系统功能模块结构图

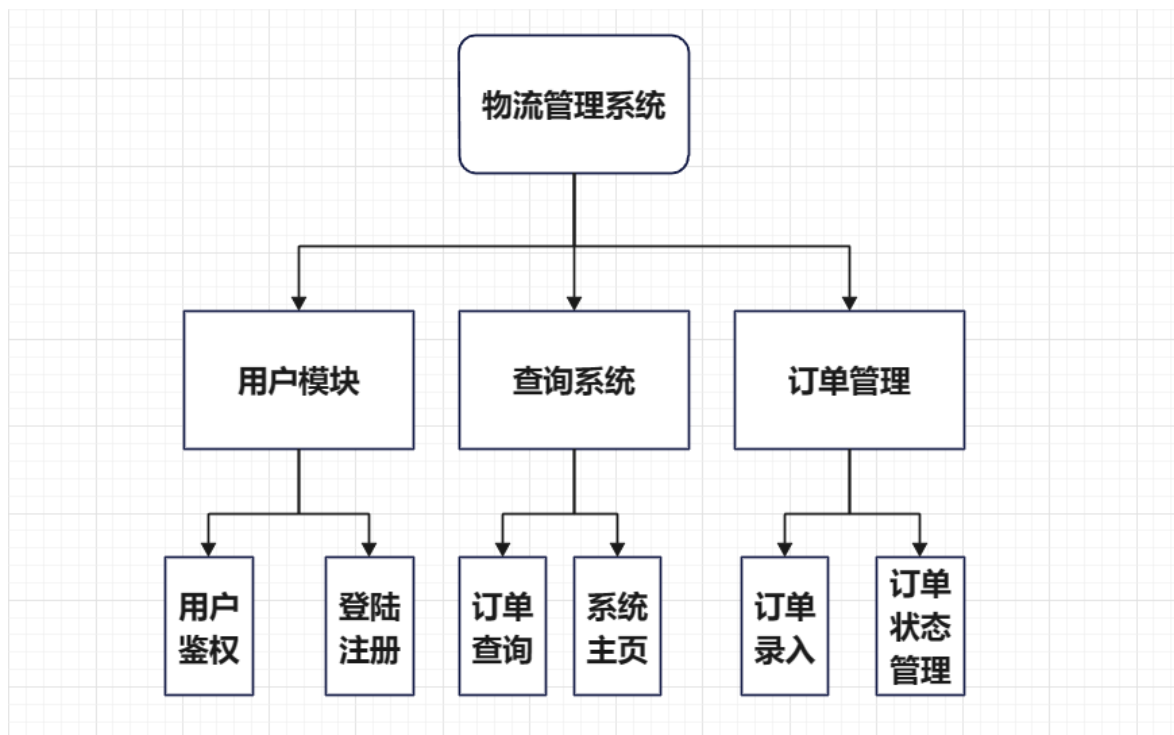


图1 系统功能模块结构图

## 1.2 系统功能模块描述

### 1.订单查询模块

该模块主要负责用户和系统管理员的订单查询模块。该模块根据登陆的用户返回与其相关的所有物流订单给予显示。

### 2.用户系统首页模块

该模块主要负责面向用户的首页显示，主要涉及订单信息、待办事项等信息的展示。

### 3.用户权限模块

该模块主要负责不同用户的鉴权。目前开发进度只设计管理员和普通用户两种级别，后续拟引入运力管理员、运力资源用户等层级。

### 4.订单录入模块

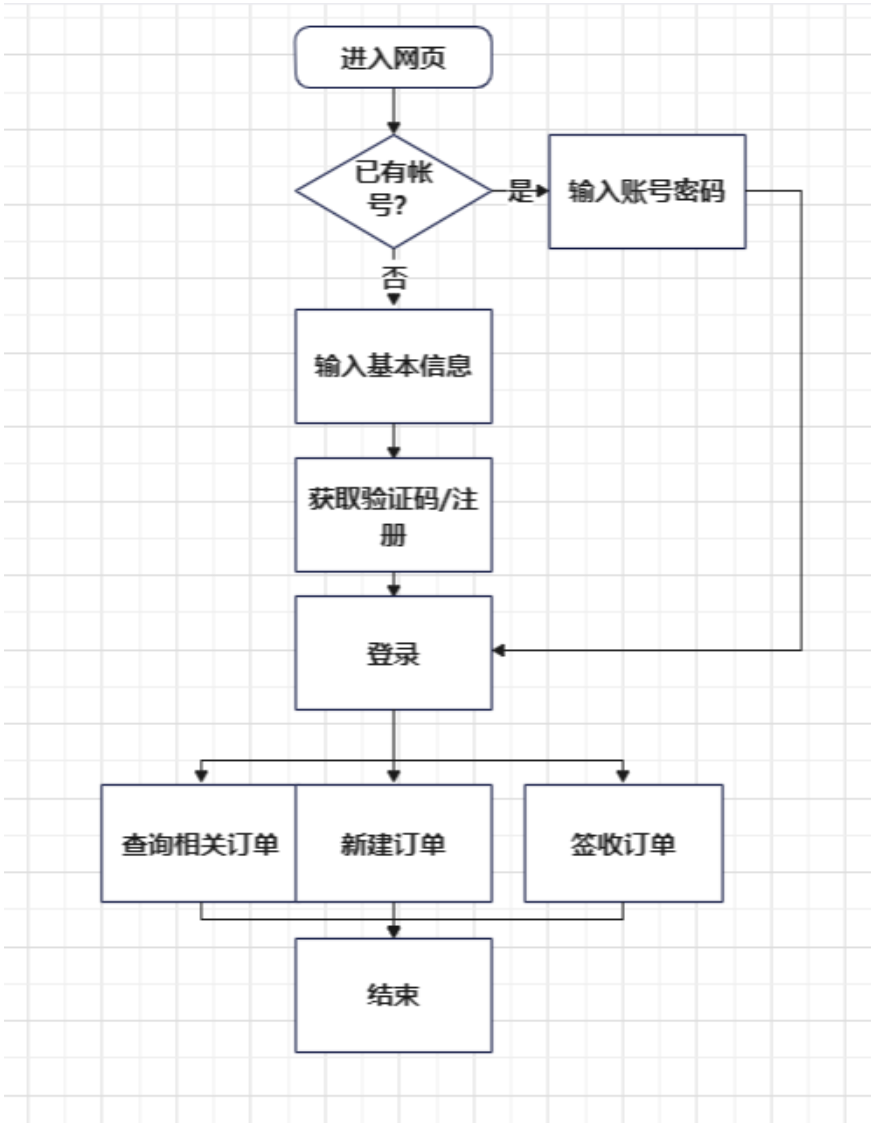
该模块主要负责订单信息的录入工作。

### 5.订单状态管理模块

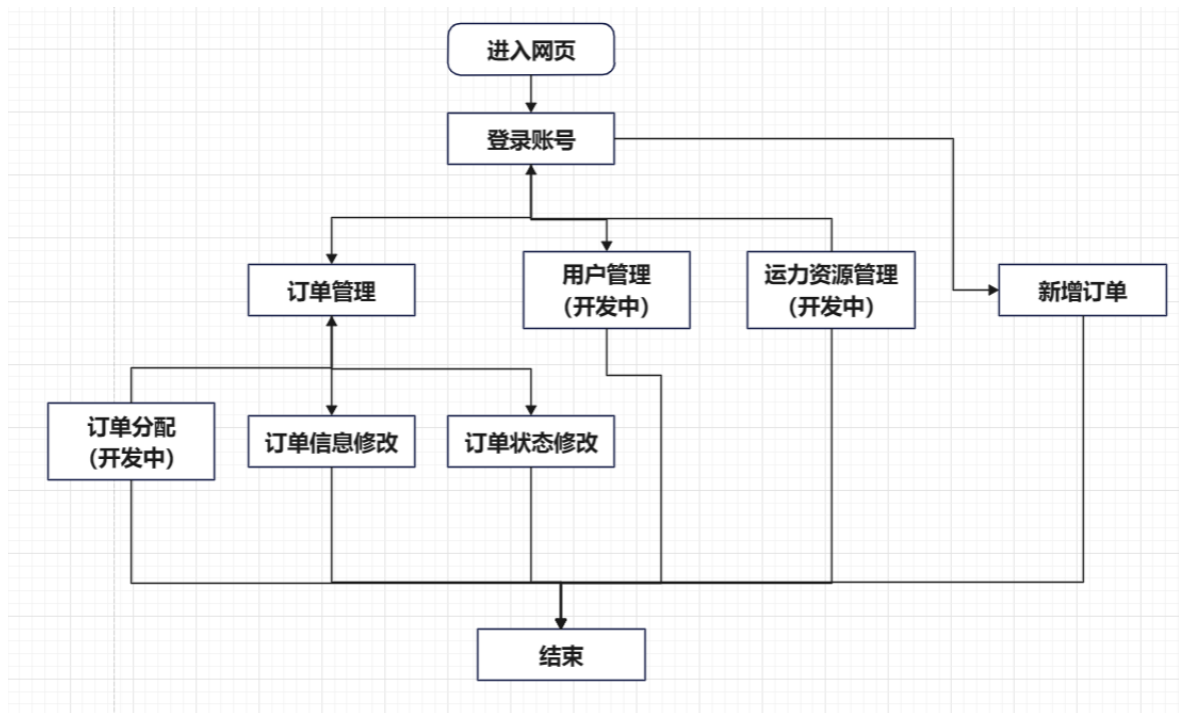
该模块使用者为系统管理员，主要用于处理客户下的订单，变更其当前状态、修改订单信息等功能。

## #2 系统流程图

对于目前的开发进度，使用对象分别为普通用户和管理员。已实现的功能的流程图分别为下图。



图二 用户系统流程图



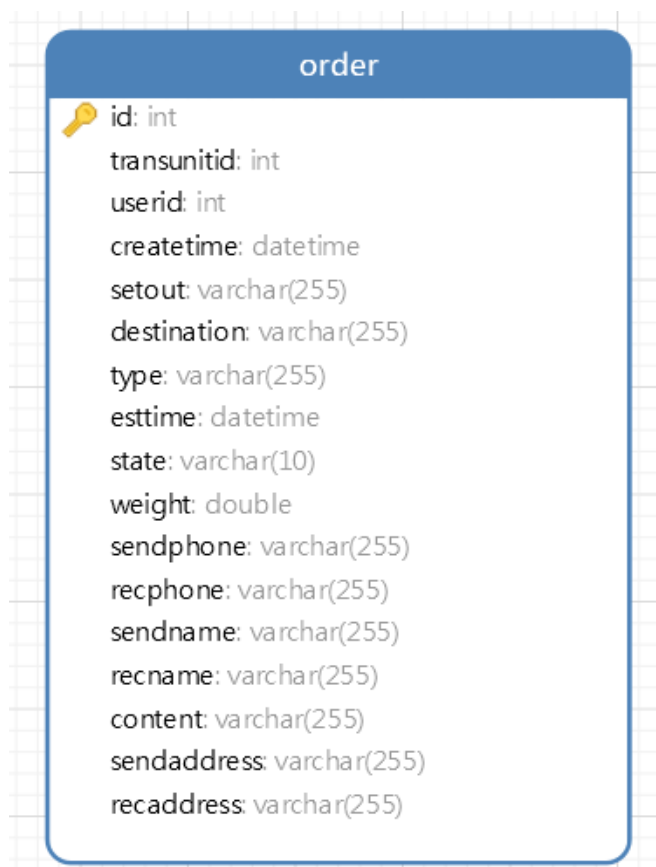
图三 管理员系统流程图

## #3 数据库设计

### 3.1 订单信息表

该表主要用于记录存入的订单信息。除和订单相关的基础信息外，较重要字段为

1. id: 自增的订单id，为订单的唯一标识
2. transunitid: 运力资源预留字段，后期和**表3 运力资源表**查询订单承运单位等信息。
3. type: 运输方式
4. content: 备注



图四 订单信息表字段

### 3.2 用户信息表

该表主要用于记录注册的用户信息。

除用户基本字段外，**identity**字段用于记录用户等级，用于登录时的鉴权操作。

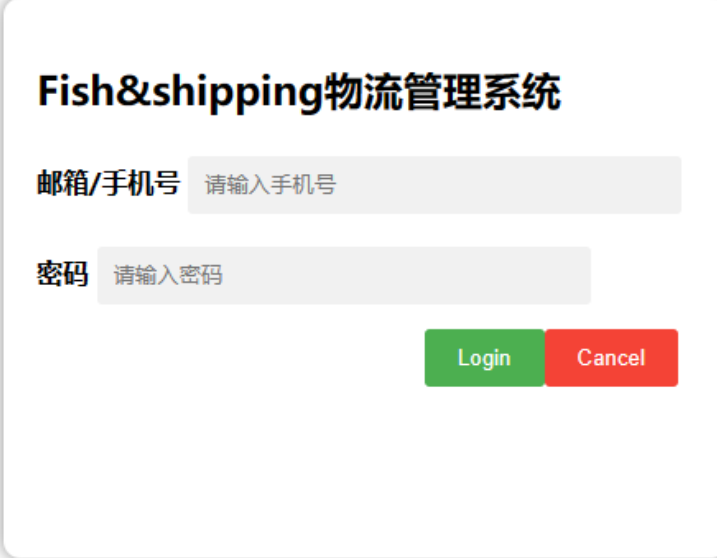


图五 用户表字段

## #4 系统模块设计

### 4.1 登陆注册页面

本系统作为面向物流管理业务的系统，登陆后使用本系统是硬性要求。下图为本系统的登录注册页面，该模块主要完成两点功能，一是普通用户的注册，二是各等级用户的鉴权。



The image shows a login and registration form for a system titled "Fish&shipping物流管理系统". The form is contained within a white rounded rectangle with a subtle drop shadow. It features two input fields: the first is labeled "邮箱/手机号" (Email/Phone Number) with a placeholder "请输入手机号" (Please enter phone number), and the second is labeled "密码" (Password) with a placeholder "请输入密码" (Please enter password). Below these fields are two buttons: a green "Login" button and a red "Cancel" button.

图六 登录页面

### 4.2 新增订单页面

本页面使用者为普通用户和管理员，用于订单的录入工作。在该页面输入所有新增订单需要的信息后即可创建订单，页面设计如下。订单新增后会存入云端数据库，默认状态为待发货，等待管理员分配运力资源及进一步修改其后续状态。

Fi&shpping 物流管理系统

注册

主页

新增订单

我的订单

管理订单

发件人信息

收件人信息

寄件人姓名

请输入寄件人姓名

寄件人手机号

请输入寄件人手机号

详细始发地

请输入详细始发地

重量 (kg)

请输入重量 (kg)

收件人姓名

请输入收件人姓名

收件人手机号

请输入收件人手机号

收货详细地址

请输入收货详细地址

备注

请输入备注

提交

图七 新增订单界面

### 4.3 用户端订单查询

本页面使用者为普通用户，用于订单信息的查询。该页面进入时请求数据库，根据其手机号信息查询与其相关的订单并渲染至前端。因面向用户，结果分为寄出和收到的两部分。

Fi&shpping 物流管理系统

注册

主页

新增订单

我的订单

管理订单

我收到的

发件人	收货地址	预计时间	当前状态	详情
11	1	Thu Jun 15 00:00:00 CST 2023	未发货	
禹浩男	四川省成都市锦江区	Thu Jun 15 00:00:00 CST 2023	已完成	
程美玲	辽宁省沈阳市沈河区	Wed Jun 28 16:45:00 CST 2023	待签收	确认签收
张海燕	江苏省南京市秦淮区	Sat Jul 01 09:30:00 CST 2023	已完成	
李明	湖南省长沙市岳麓区	Wed Jun 28 10:00:00 CST 2023	运输中	
谢阳娜	贵州省贵阳市云岩区	Fri Jun 30 14:20:00 CST 2023	未发货	
徐丽	安徽省合肥市蜀山区	Sat Jul 01 07:40:00 CST 2023	待签收	确认签收
刘建	山东省青岛市市南区	Sun Jul 02 13:00:00 CST 2023	运输中	

我发出的

收件人	收货地址	预计时间	当前状态	详情
陈志强	江苏省无锡市滨湖区	Tue Jun 27 12:00:00 CST 2023	待签收	
王丹	重庆市渝北区	Thu Jun 29 15:00:00 CST 2023	已完成	
李红	江苏省苏州市姑苏区	Sat Jul 01 08:00:00 CST 2023	已完成	

图八 用户端订单查询界面

### 4.4 管理员端订单管理

本页面使用者为管理员，用于订单的各项管理工作。此页面仅当登录时被鉴权为管理员时可被显示。此页面显示相关的所有订单，并可以根据订单状态筛选。目前开发进度为管理员可以通过点击修改状态按钮来修改订单的目前状态。对于订单各字段的修改在完成报告时正处于开发中。





图九 管理员端订单管理界面

## #5 考察功能点

### 5.1 用户认证和鉴权

用户登录后会根据用户的角色提供不同权限。如对于管理员，有订单管理模块，而对于普通用户只有查询模块。



## 5.2 单元测试

对各个Controller进行单元测试，这里使用到JUnit对代码进行测试，使用

`MockMvcRequestBuilders` 类构建模拟的HTTP请求，例如 `get()`、`post()`、`put()` 等方法，可以设置请求的URL、参数、请求体等。然后使用 `MockMvc` 对象的 `perform()` 方法执行模拟的HTTP请求，可以传入一个 `RequestBuilder` 对象作为参数。最后对执行结果进行断言：使用 `ResultMatcher` 对象进行断言，例如 `status()`、`content()`、`header()` 等方法，可以验证HTTP响应的状态码、响应内容、头部信息等。

以最核心的OrderController为例，编写的代码部分如下。

```

1 package com.yufop.tran.controller;
2
3 import ...
4
19 @RunWith(SpringRunner.class)
20 @SpringBootTest
21 @AutoConfigureMockMvc
22 class OrderControllerTest {
23
24     // 创建 MockMvc 对象
25     @Autowired
26     WebApplicationContext webApplicationContext;
27     MockMvc mockMvc;
28
29     // Before 就是每次运行测试类之前就会执行下边的函数
30     @BeforeEach
31     void setUp(){
32         // 自动去查找Controller
33         mockMvc = MockMvcBuilders.webAppContextSetup(webApplicationContext).build();
34     }
35
36     @Test
37     void listordersp() {
38         // mockMvc.perform执行一个请求
39         try {
40             mockMvc.perform(MockMvcRequestBuilders
41                 // MockMvcRequestBuilders.get("XXX")代表get请求指定路径xxx
42                 .post( uriTemplate: "/order/selectBysp")
43                 // 设置返回类型
44                 .accept(MediaType.APPLICATION_JSON_VALUE)
45                 // 添加请求参数
46                 .param( name: "phone", ...values: "18224426057")
47                 //ResultActions.andExpect添加执行完成后的断言

```

对于最初版的Orderlist中的5个方法，测试均通过。

测试结果	耗时
OrderControllerTest	1秒 718毫秒
listorder()	1秒 7毫秒
newOrder()	6毫秒
myOrder()	16毫秒
listorderrp()	643毫秒
listordersp()	46毫秒

```

C:\Users\Jarvis2K\.jdk\openjdk-18.0.1.1\bin\jav
---- IntelliJ IDEA coverage runner ----
sampling ...
include patterns:
com\yufop\tran\controller\...*
exclude patterns:
17:15:36.474 [main] DEBUG org.springframework.te
17:15:36.490 [main] DEBUG org.springframework.te
17:15:36.576 [main] DEBUG org.springframework.te
17:15:36.591 [main] INFO org.springframework.boc
17:15:36.597 [main] DEBUG org.springframework.te
17:15:36.598 [main] DEBUG org.springframework.te
17:15:36.598 [main] INFO org.springframework.tes

```