**华南农业大学珠江学院  
毕业设计**

**咖啡豆购物商城系统设计与实现**

何展豪

指导教师： 赵志俊

学院： 信息工程学院 年级专业： 2018级计算机科学与技术

提交日期： 2022年2月25日 答辩日期： 202 年 月 日

答辩委员会主席（签名）：

评阅人（签名）：

年 月 日

目 录

[1. 引 言 5](#_Toc103005451)

[1.1 课题背景 5](#_Toc103005452)

[1.2 国内研究情况 6](#_Toc103005453)

[1.3 国外研究情况 6](#_Toc103005454)

[1.4 课题研究目标及意义 6](#_Toc103005455)

[2. 相关框架、系统开发环境及技术简介 7](#_Toc103005456)

[2.1 系统开发环境 7](#_Toc103005457)

[2.1.1 Visual Studio Code源代码编辑器 7](#_Toc103005458)

[2.2 数据库简介 7](#_Toc103005459)

[2.3 Express 8](#_Toc103005460)

[2.4 Vue框架 9](#_Toc103005461)

[2.5 NodeJS 10](#_Toc103005462)

[3. 系统开发的思想与技术 10](#_Toc103005463)

[3.1 JS技术与MVVM模式以及系统的基本框架 10](#_Toc103005464)

[4. 系统分析 11](#_Toc103005465)

[4.1 可行性分析 11](#_Toc103005466)

[4.1.1 技术可行性 11](#_Toc103005467)

[4.1.2 操作可行性 12](#_Toc103005468)

[4.1.3 经济可行性 12](#_Toc103005469)

[4.1.4 法律可行性 12](#_Toc103005470)

[4.2 需求分析 13](#_Toc103005471)

[4.2.1 功能性需求 13](#_Toc103005472)

[4.2.2 系统用例图 13](#_Toc103005473)

[4.2.3 非功能性需求 14](#_Toc103005474)

[5. 概要设计 15](#_Toc103005475)

[5.1 系统功能模块设计 15](#_Toc103005476)

[5.2 系统前台设计 16](#_Toc103005477)

[5.2.1 用户注册模块 16](#_Toc103005478)

[5.2.2 用户登录模块 19](#_Toc103005479)

[5.2.3 浏览商品模块 20](#_Toc103005480)

[5.2.4 意见反馈模块 22](#_Toc103005481)

[5.2.5 订单管理模块 23](#_Toc103005482)

[5.2.6 收藏夹模块 25](#_Toc103005483)

[5.2.7 地址管理模块 27](#_Toc103005484)

[5.2.8 购物车模块 29](#_Toc103005485)

[6. 数据库设计 30](#_Toc103005486)

[6.1 数据库结构 30](#_Toc103005487)

[6.2 数据库表的设计及关系 32](#_Toc103005488)

[6.2.1 用户信息表 33](#_Toc103005489)

[6.2.2 商品信息表 33](#_Toc103005490)

[6.2.3 订单信息表 34](#_Toc103005491)

[6.2.4 收藏夹信息表 34](#_Toc103005492)

[6.2.5 购物车信息表 35](#_Toc103005493)

[7. 系统实现 36](#_Toc103005494)

[7.1 后端接口的实现 36](#_Toc103005495)

[7.1.1 封装数据库信息 36](#_Toc103005496)

[7.1.2 封装post请求响应操作 37](#_Toc103005497)

[7.1.3 封装SQL语句 37](#_Toc103005498)

[7.1.4 封装有关用户信息的接口 40](#_Toc103005499)

[7.1.5 封装有关购物车信息的接口 45](#_Toc103005500)

[7.1.6 封装有关商品信息的接口 46](#_Toc103005501)

[7.2 前端页面的实现 46](#_Toc103005502)

[7.2.1 前端页面介绍 46](#_Toc103005503)

[7.2.2 首页页面 47](#_Toc103005504)

[7.2.3 购物车页面 48](#_Toc103005505)

[7.2.4 我的页面 50](#_Toc103005506)

[• 收藏夹页面 55](#_Toc103005507)

[• 意见反馈页面 56](#_Toc103005508)

[• 订单管理页面 57](#_Toc103005509)

[7.2.5 登录注册框 58](#_Toc103005510)

[7.2.6 封装请求拦截器以及API 64](#_Toc103005511)

[8. 测试 66](#_Toc103005512)

[8.1 测试内容 66](#_Toc103005513)

[8.2 测试结果分析 67](#_Toc103005514)

[9. 总结 68](#_Toc103005515)

[参考文献 69](#_Toc103005516)

[致 谢 70](#_Toc103005517)

# 摘 要

本毕业设计的内容是设计并且实现一个基于web技术的咖啡豆购物商城，系统主要以H+C+JS（HTML+CSS+JAVASCRIPT）作为开发基础,主要使用了Express等多种框架的结合使用，使用Visual Studio Code作为开发工具，以MYSQL作为数据库，以Visual Studio Code作为界面美化工具，使用JavaScript这个语言来开发，页面采取Vue动态页面开发技术。本系统界面偏向简洁风格，并且操作方便，容易维护。

关键词：咖啡豆购物商城; Express; HTML; JavaScript; Vue; MYSQL

# Abstract

The content of this graduation design is to design and implement a coffee bean shopping mall based on web technology, the system is mainly based on H+C+JS (HTML+CSS+JAVASCRIPT), mainly using the combination of Express and other frameworks, using Visual Studio Code as a development tool, MYSQL as a database, and Visual Studio Code as an interface beautification tool. Using javascript as a language to develop, the page adopts the Vue dynamic page development technology. The interface of this system is biased towards a simple style, and it is easy to operate and easy to maintain.

Keywords**：**Coffee Bean Shopping Mall; Express; HTML; JavaScript; Vue; MYSQL

设计说明

设计任务的来源。

本课题的设计任务来源于本人的毕业设计项目：“咖啡豆购物商城系统设计”。

在生活水平的不断提高的当下，人们渐渐的开始注重自己的生活质量，在生活上增添许多的调料，有的人去做瑜伽，有的人坐下来喝喝茶或咖啡，虽然国内咖啡普及范围还比较小，但也有不少人喜欢喝，甚至自己买咖啡豆自己冲煮，但也正因为好且便宜的咖啡豆比较难买到，所以本人希望通过这一课题的设计和实现，帮助一些小型咖啡豆的烘焙厂商能够更加容易的去售卖自己的商品，同时也是对本人大学四年来所学的知识和技术进行一次回顾、总结和运用，为以后的工作奠定技术和理论基础。因此本人选择了“咖啡豆购物商城系统”这个课题。

设计的技术路线。

1. 系统需求分析

系统需求分析即在系统进行开发工作之前，经过一系列的调研和分析来确定系统必须要做什么的一个过程。开发人员开发前必须要了解系统的功能性需求、非功能性需求甚至系统可靠性需求等等， 从而为后面进行系统的功能模块设计奠定基础。

1. 系统概要设计

概要设计通过在需求分析阶段得出的结论，即已经搞清楚了“要解决什么问题”，来确定系统要完成的各个功能模块，确定软件系统的总体布局还有各个子模块的功能和模块之间的关系。

1. 系统详细设计

设计数据库的概念模型、逻辑模型、物理模型和具体数据表的数据结构。

1. 系统具体实现

该阶段主要实现软件系统功能模块的代码逻辑和程序的设计与实现。给出系统的配置文件、页面的效果图等。由于本系统采用 MVVM 的开发模式，因此在系统具体实现这块采用分层架构来实现对应的模块逻辑。

1. 系统测试

该阶段主要对开发完成的系统进行各个功能模块的测试，并使用不同的浏览器来完成，以保证系统的可用性和安全性，测试可发现系统可能潜在存在的一些错误或者不足。本系统设计的测试使用黑盒测试的方式来进行。

关键词：设计路线；概要设计；设计模式；MVVM；购物商城

# 引 言

## 课题背景

在国外，咖啡豆的购买要比国内更加的容易购买得到，因为在国外，咖啡已经是外国人们的日常生活中必不可少的一个饮料，上到总统，下到工人，虽然所喝的咖啡品质不一样，但都是需要咖啡豆来萃取出咖啡液来引用，因此，国外的咖啡豆市场也比我们国内的咖啡豆市场更为的成熟，也更为的便捷，他们有更多的购物平台去购买咖啡豆。在国内也有非常大的电商平台，像淘宝、京东，这些大型电商网站，有着大量的用户，但是，想要在这些大型的电商平台去卖咖啡豆，对于一些小批次咖啡豆的烘焙厂商是一笔不小的成本，我现在也有关注到一些小批次咖啡豆的烘焙厂商是在微信群上去售卖他们的产品，他们也是拥有不小的客户，我也有买他们的产品来自己冲煮成咖啡喝，他们的产品质量也是挺不错的，但也因为批次小，利润小，成本固然就不能过高，这种情况下就需要一个比较简单干净的咖啡豆购物商城提供给他们去售卖他们自己的产品，

在这个咖啡豆购物商城的设计上，本咖啡豆购物商城系统采用了MVVM模式，由于只是满足一些小型的企业和商家的需求，所以不太可能把所有的功能都完善，这是不可能实现的对于任何一个网站来说，随着客户需求的增加，后期的升级与改进也是很必要的，当然，如果是大型的网站的话，运用一些架构等就是必须的了，但这样的小型网站对于这种模式来说，就已经足够了[1]。

## 国内研究情况

在国内，咖啡豆一直都不是盛行的产品，知道最近的这几年，人们的生活质量提升了，才开始越来越多的好的咖啡饮品出现在市场上，以往也有，但是品质都不怎么样，现在咖啡店铺也越来越多，比如Manner Coffee、瑞幸咖啡等的咖啡店，他们的出品也是还不错的[2]。咖啡的普及，也意味着咖啡豆需求的增长，现在国内也有许多的个人咖啡冲煮爱好者，那么这些人也是想要买到好的咖啡豆来自己冲煮。所以随之诞生了相应的小程序，比如群接龙、快团团等的小程序都是可以给小型咖啡豆烘焙商去售卖自己的产品，但是也因为这些小程序还没开发多久，所以毛病也是有的[3]。

## 国外研究情况

相对于国内，国外的咖啡盛行程度可就要比国内高得多了，这也让国外更早的开发出售卖咖啡豆的电商平台，例如美国的精品咖啡电商「Trade」，Trade 的出现也解决了咖啡烘培商的痛点，能够降低运营和获客成本。咖啡烘焙商在 Trade 的线上渠道，能够获得更多的流量，被更多的消费者接触到。

Trade 从 2018年创立以来，到了目前为止，已经有超过55家专业的咖啡烘焙商入驻 Trade 平台，累计销售已经超过一百万袋咖啡，包括北美知名的独立咖啡品牌 Intelligentsia、Verve 和 Stumptown等，这些商家都分布在了美国的38个州里面。

## 课题研究目标及意义

一个简单方便的电商购物商城可以让开发费烘焙商更好的去售卖自己的产品。

通过Vue框架，可以高效的开发出符合用户及咖啡烘焙商的用户需求。本课题旨在开发一个适合小型咖啡烘焙商的一个咖啡购物商城系统，帮助小型咖啡烘焙商有更好的销量，同事也对本人大学四年来所学的知识和技术进行一次回顾、总结和运用，为以后的工作奠定技术和理论基础。

# 相关框架、系统开发环境及技术简介

## 系统开发环境

## Visual Studio Code源代码编辑器

Visual Studio Code（简称 VS Code）这是一款由微软自己开发且跨平台的免费源代码编辑器。该软件支持语法高亮、代码自动补全（又称 IntelliSense）、代码重构功能，并且内置了命令行工具和 Git 的版本控制系统。用户可以通过内置的扩展程序商店安装扩展以拓展软件功能，也可以通过更改主题和键盘快捷方式实现个性化设置。

## 数据库简介

数据库使用的是MySQL这个开源数据库。

MySQL的特性：使用了多种编译器进行测试，并可以使用C和C++编写，保证了源代码的可移植性 。支持AIX、FreeBSD、HP-UX、Mac OS、OpenBSD、OS/2 Wrap、Solaris、Windows等的多种操作系统 。API也为多种编程语言提供了。这些编程语言包括了C、C++、Eiffel、Java、Perl、PHP、Python、Ruby和Tcl等。支持多线程充分的利用CPU资源。优化了SQL查询算法，有效地提高查询的速度。既在客户端服务器网络环境当中能够作为一个单独的应用程序应用，也能够作为一个库而嵌入到其他的软件中提供多语言支持, 数据表名称和数据列名称都可以使用常见的编码如中文的GB2312、BIG5，日文的Shift\_JIS等。提供了TCP/IP、ODBC和JDBC等多种数据库连接途径。提供用于管理、检查、优化数据库操作的管理工具。可以处理拥有上千万条记录的大型数据库。

MySQL的应用：与其他的大型数据库例如Oracle、DB2、SQL Server等相比，MySQL自然是有它的不足之处，比如规模小、功能有限（MySQL Cluster的功能和效率都相对比较差）等，但是它受欢迎的程度并没有丝毫减少。对于中小型的企业和一般个人的使用者来说，MySQL提供的功能已经是绰绰有余，而且可以大大降低总体成本，因为MySQL是开放源码软件来的。

## Express

Express 是一个灵活而又简洁的 [node.js](https://baike.baidu.com/item/node.js/7567977) Web应用框架, 帮助你来创建各种Web应用是基于它提供了一系列强大特性。Express 是不会对 node.js 已有的特性进行二次抽象，我们只是在它之上扩展了Web应用所需要的功能。来自Connect框架的中间件随取随用以及丰富的HTTP工具，快速又简单的创建强健、友好的API。

Express 框架核心特性：

1. 响应 HTTP 的请求的话可以设置中间件来响应。
2. 定义了路由表来用于执行不同的 HTTP 的请求动作。
3. HTML 页面的话是可以通过向模板传递参数来动态的渲染。

## Vue框架

Vue是一套用来构建用户界面的**渐进式框架**。Vue 是被设计为可以自底向上逐层应用，这是它与其它的大型框架不同的地方。Vue 不仅容易于上手，还便于与第三方库或既有的项目整合，它的核心库是只关注视图层的。另一方面， [现代化的工具链](https://cn.vuejs.org/v2/guide/single-file-components.html)以及各种[支持类库](https://github.com/vuejs/awesome-vue#libraries--plugins)与Vue框架结合使用的时候，Vue 也完全能够为复杂的单页应用提供驱动。

Vue.js的优点:

1. 体积小

压缩后仅仅只有33K;

1. 更高的运行效率

基于虚拟DOM, 一种可以预选通过JavaScript来进行各种计算, 把最终的DOM操作计算出来并加以优化的技术，由于这个DOM操作是属于预处理操作， 它并没有真正地操作DOM， 所以叫虚拟DOM。

1. 双向数据绑定

让开发者不用再去操作DOM对象, 可以把更多的精力投入到业务逻辑上。

1. 生态丰富、学习成本低

市场上拥有大量成熟、稳重的基于Vue.js的UI框架和常用的组件！拿来即用实现快速的开发！ 对初学者友好、入门也容易、学习资料多。

## NodeJS

Node.js是基于Chrome JavaScript运行的时候建立的一个平台，它主要是用于创建快速的、可扩展的网络应用[4]，实际上它是对Google Chrome V8引擎进行了封装。

运行于浏览器的JavaScript，浏览器就是JavaScript的代码解析器，而Node.js则是服务器端JavaScript的代码解析器，Node.js是解析和运行存在于服务器端的JavaScript代码的。

JavaScript解析器只是JavaScript代码的一种运行环境，浏览器是JavaScript的一种运行环境，操作DOM对象和window对象等等的接口是浏览器为JavaScript提供的。

# 系统开发的思想与技术

## JS技术与MVVM模式以及系统的基本框架

本系统采用的是JavaScript技术，基于MVVM模式开发，使用Vue框架来增加系统的开发效率。那么所谓的MVVM模式其实是Model-View-ViewModel的简写，它本质上就是MVC 的改进版[5]。MVVM 让我们将视图的UI 和业务逻辑分开，其实就是将其中的View 的状态和行为抽象化。

MVVM模式有几大优点：

1. 低耦合。视图（View）可以独立于Model的变化和修改，一个ViewModel是可以绑定到不同的"View"上，当View变化时Model可以不变，当Model变化时View也可以不变。

2. 可重用性。你可以在一个ViewModel的里面把一些视图逻辑放进去，让很多的view重用这段视图逻辑。

3. 独立的开发。设计人员可以专注于页面设计，开发人员可以专注于业务逻辑和数据的开发（ViewModel），使用Expression Blend可以很容易的设计界面并生成xaml代码。

4. 可测试。界面是比较难测试的，测试是可以针对ViewModel来写。

# 系统分析

## 可行性分析

## 技术可行性

本咖啡豆购物商城系统以JavaScript为基础，结合NodeJS、Express、MySQL进行开发，为了改善用户的体验，前端统一使用Html来作为视图显示，使用Vue框架来优化项目代码，简化开发。后台数据库使用MySQL5.7.26版本、前后端编辑器统一使用Vscode进行开发。

综合上述的几点可以得知，本咖啡豆购物商城系统在技术层面上的实现是完全可行的。

## 操作可行性

只要使用者有一台电脑或者一部智能手机，而且网络可用，就可以通过浏览器访问本咖啡豆商城网站。本咖啡豆商城系统操作简单，不需要具备任何专业的知识即可根据提示完成操作，因此其在操作上完全可行。

## 经济可行性

本咖啡豆购物商城网站在开发的过程中所使用的框架和技术依赖是完全开源和免费的，不需要花钱购买 ，但因本咖啡豆购物商城是全栈项目，如要部署至服务器上需要前后端都部署上去，因为目前的经济状况，不能部署上服务器，但能在本地上浏览该项目，因此从经济上考虑是可行的。

## 法律可行性

本咖啡豆购物商城在开发时所用到的资源和网站的内容是积极、健康和合法的。本系统所有使用的开发技术不涉及任何一个公司的知识产权，以及公司的法律法规，从法律的角度考虑，开发本系统是可行的[6]。本咖啡豆购物商城在开发时所用到的资源和网站的内容是积极、健康和合法的。本系统所有使用的开发技术不涉及任何一个公司的知识产权， 以及公司的法律法规，从法律的角度考虑，开发本系统是可行的。

## 需求分析

## 功能性需求

1. 游客浏览商品、登录、注册。
2. 用户登录、注册、浏览商品。
3. 用户发表意见反馈。已登录的用户可以对网站进行意见反馈。
4. 用户订单管理。已登录的用户可以对订单进行删除。
5. 用户地址管理。已登录的用户可以对地址进行添加和修改。
6. 用户收藏夹。已登录的用户可以对已收藏的商品进行删除和查看。
7. 用户购物车。已登录的用户可以进入购物车提交购买商品的订单。

## 系统用例图

本咖啡豆购物商城的参与者有游客和用户，游客可以不用登录即可浏览商品的信息，用户的权限比游客高，用户在游客权限的基础上，多了意见反馈、订单管理、收藏夹、地址管理、购买商品。

本咖啡豆购物商城所有的参与者有游客和用户，游客和用户只是权限上的不同。主要的商品浏览功能都是可以用的，要购买才需要注册或者登录[7]。

用户的用例图是用来对用户信息进行分析，其中包含有注册的用户和游客两种对象，其中游客的功能有：浏览商品和注册登录，用户的功能有：意见反馈、订单管理、收藏夹、地址管理、购买商品、意见反馈、订单管理、收藏夹、地址管理、购买商品。用户用例图如图 3.1 所示。

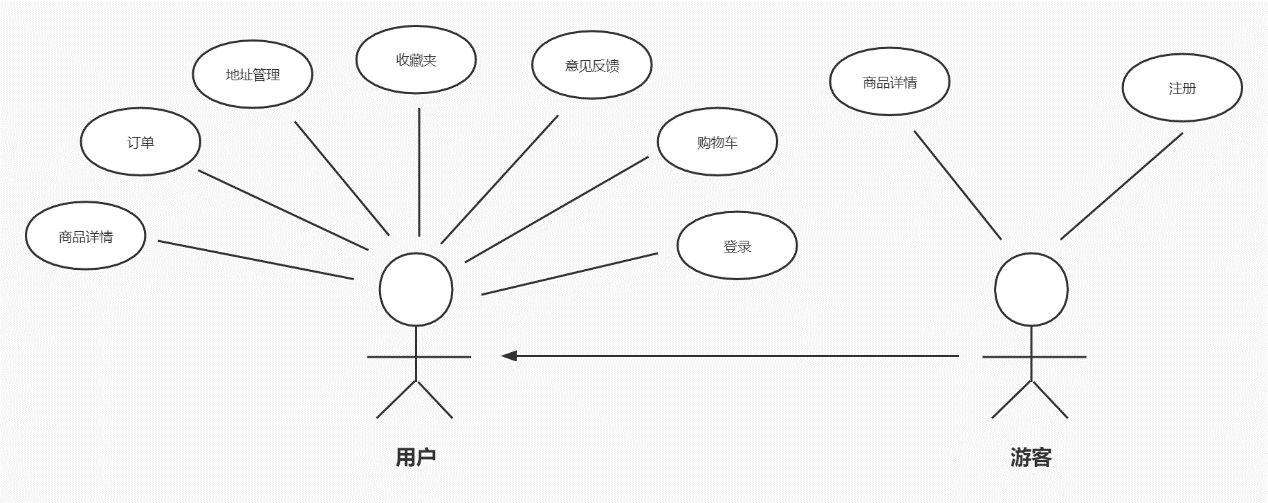


图 4.1

## 非功能性需求

1. 界面需求。因为作为咖啡豆购物商城，在众多的购物商城平台中，要突出自己的个人特色，所以在界面的需求上，要区别于传统的购物，本咖啡豆购物商城系统在界面上保持这简洁、美观、现代化元素的特点，没有大型购物商城网站那么的庞大而杂乱。虽然功能上面是没有大型购物商城网站的功能那么的庞大且多样化，但因为简洁和轻量化，进而显出个人网站的特色，可以让人看起来耳目一新。
2. 安全性需求。因本人往前端的方向发展，所以只能考虑前端用户的安全性，后端用户的安全性，因知识尚浅，未能为用户的密码进行加密，但是能够在为前端提供登录凭证的时候，可以做到不暴露用户的密码，并且在前端发起请求的时候，用户的密码不会暴露在地址栏上，也不会保存在本地，所以也能具有一定的安全性。
3. 系统的响应速度。对于用户来说，系统的响应时间将直接影响到用户的体验。本购物商城系统设计的页面保持简洁，减少代码的冗余以及一些不必要的功能与界面。系统整体采用前后端分离的模式来进行开发，前端与服务器通过 json 数据的形式来交换数据，这种异步请求的方式可以大大提升用户的体验，同时，服务器只负责传输数据，不需要负责页面的渲染，因此响应速度得以提高。
4. 系统的最大并发用户量。系统的最大并发量即系统能同时承受多少个用户的访问。大的购物商城平台每天都有大量的用户访问，这十分考验服务器的性能。本系统作为小型的购物商城系统，没有那么高的并发需求，也因为经济条件原因，无法上线项目，所以本系统只能在本地运行，在本地运行时，参照的性能是看运行的的电脑硬件性能，本人电脑的性还好，本购物商城打开多个网页同时使用问题不大。

# 概要设计

## 系统功能模块设计

本咖啡豆购物商城系统的功能结构有前台模块，前台模块有：用户注册、用户登录、浏览商品、意见反馈、订单管理、收藏夹、地址管理、购物车。其层次结构图如图 4.1 所示。

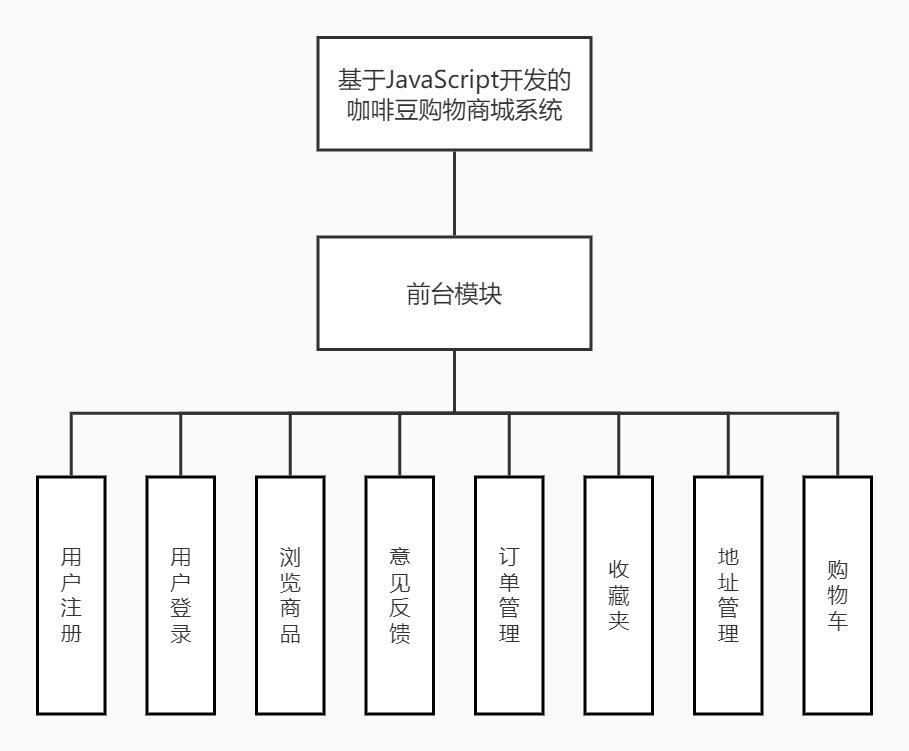


图 5.1

## 系统前台设计

## 用户注册模块

一般的购物商城网站都具备用户注册的这一功能，一方面是为了功能性需要，另一方面也是业务的需要，每个大型网站，只要有了长期的用户，也就能带来稳定的流量。

本购物商城系统在这方面效仿了各大购物商城系统，设计了一个用户注册的功能。[8]在人们第一次通过浏览器访问本购物商城系统的时候，因为没有注册，所以是游客的身份。游客可以浏览商品页面，但是不能购买商品和使用“我的“里面的四个功能模块。

这样设计是为了吸引用户去注册，并且可以划分用户权限，也能过滤掉一些不法用户恶意访问之后，恶意的去大量提交订单。当游客注册成为用户之后，也就有了身份的标识，为每个用户购买的商品不同、每个用户收藏的商品不同等等做出区分。

本购物商城系统的用户注册流程如图图 4.2 所示。

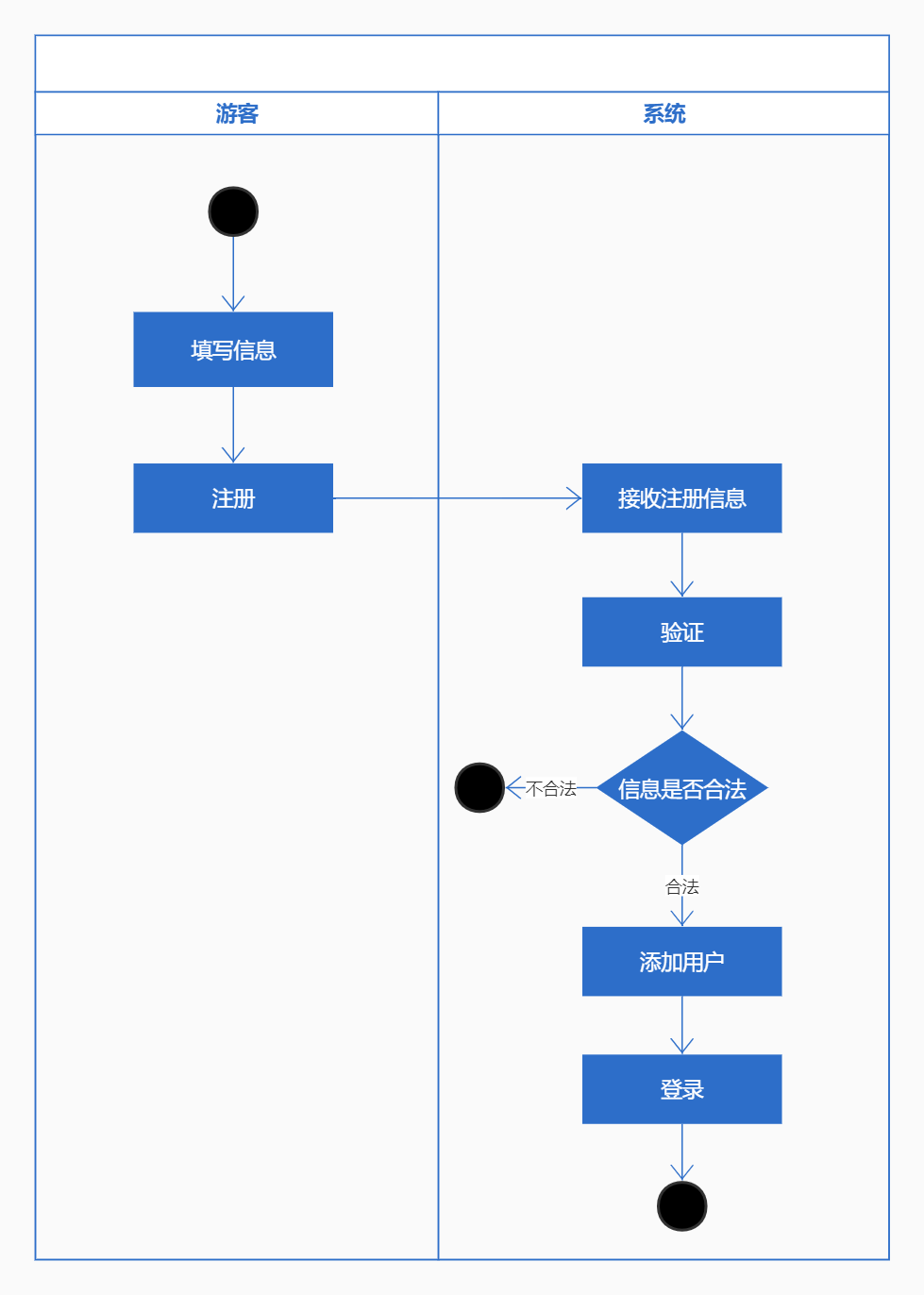


图 5.2

## 用户登录模块

登录功能和注册功能相差不大，主要一个是获取已有的账号信息进行登录，一个是添加一个新的账号之后再登录。

本购物商城系统的登录功能也是效仿各个大型购物商城系统的登录功能。人们在本购物商城中浏览到了喜欢的咖啡商品时，就会有想购买的欲望，但购买的前提是需要用户登录之后才能做的操作。

这样设计的目的可以有效的区分每个用户自己的订单、地址等的信息。

本咖啡豆购物商城系统的用户登录流程如 图 4.3 所示

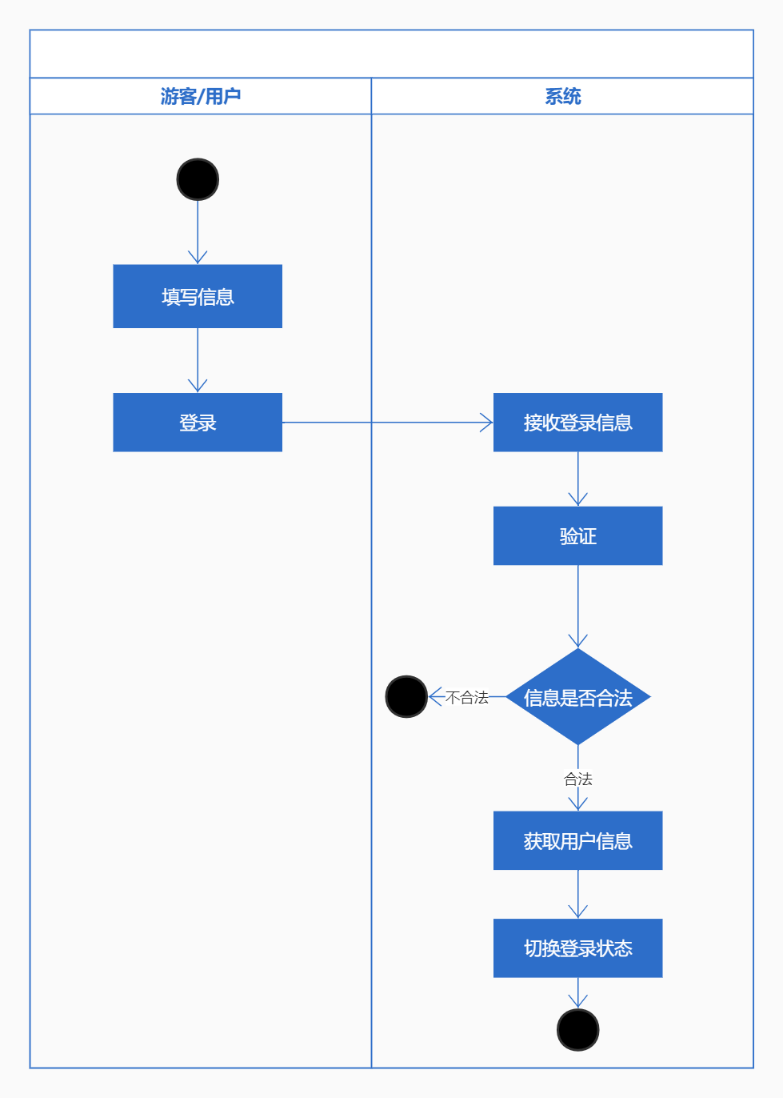


图 5.3

## 浏览商品模块

在一般的购物商城系统当中，都有商品详情的页面，其中的功能就是立即购买、加入购物车和点击收藏，这三个功能可以说在商城系统中最普遍的了，

而且这些功能都必须要用户先登录之后才能去操作，未登录是会提示用户登录。

本咖啡豆购物商城的浏览商品模块如图 4.4 所示

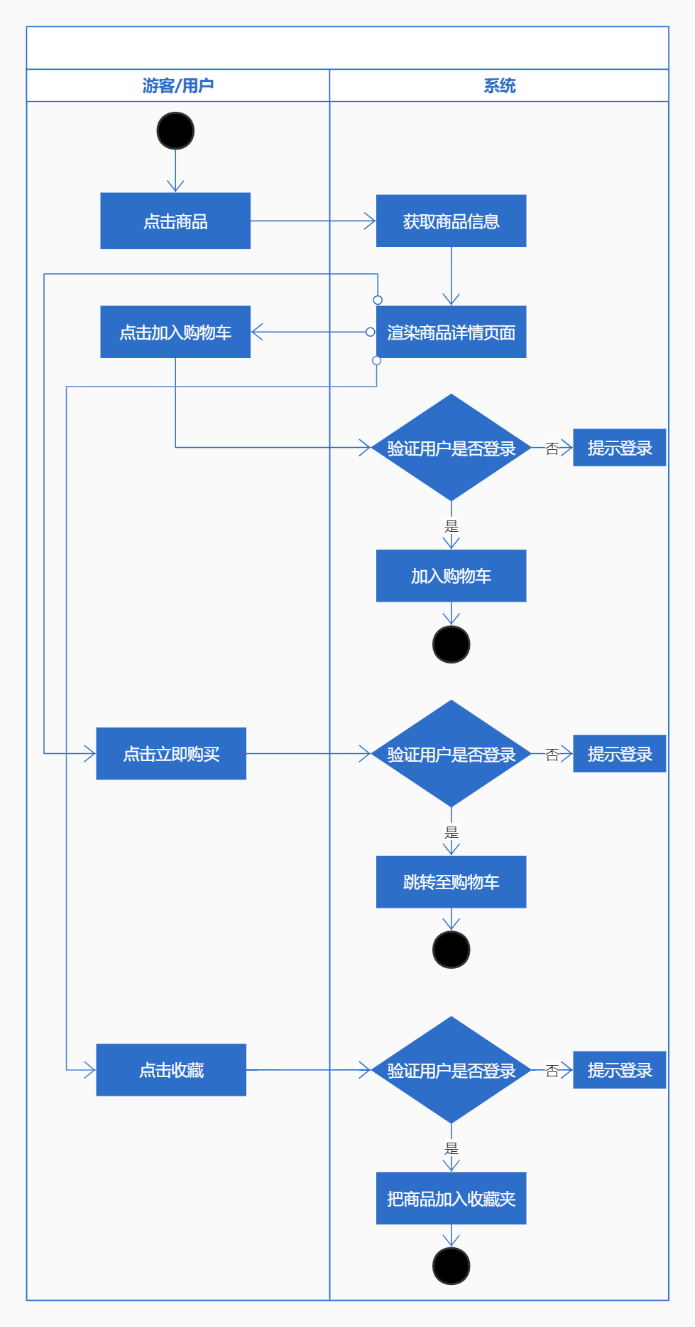


图 5.4

## 意见反馈模块

意见反馈模块可谓是所有的网站和软件都有的一个功能，可以获取到用户的体验感受，当用户有体验差的时候，就可以通过这个功能给开发者提供意见。

设计这个功能模块最主要的是增加用户体验，并让开发者知道哪个地方有bug或者是用户交互做得不好的地方，可以精准的进行修复或者优化，这也大大的方便了开发者了解自己开发的产品的不足。

本咖啡豆购物商城系统的意见反馈模块流程图如图 4.5所示。

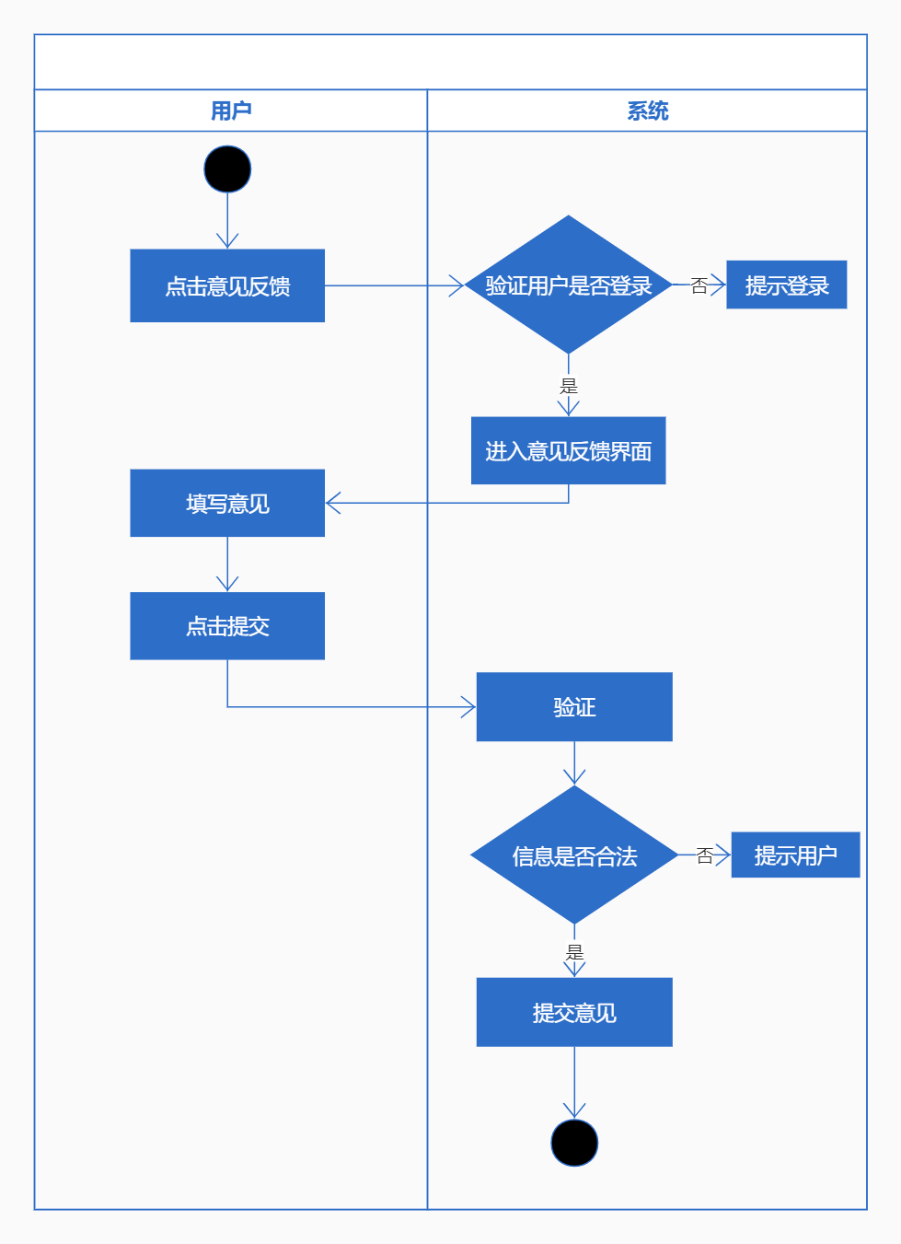


图 5.5

## 订单管理模块

在一般的购物商城网站或软件当中，对于订单的管理都是有一个模块来给到用户自己去管理。在大型的购物商城网站当中的订单管理模块是有非常多的功能，比如说能够修改订单当中的地址，还有申请退款等等的功能，我能也是效仿了其中一个功能，就是可以当下单下错了，可以删除掉该订单。其中删除订单也考虑到了用户可能会误触，所以当用户点击删除按钮的时候，系统会弹出提示框，提示用户是否确认要删除该订单，点击确认之后系统才会真正的去删除订单。

这个功能的设计也是考虑到了用户下单的时候可能会不小心点错了某个选项，但是又不能更改，就可以通过订单管理模块来删除自己不想要的订单了。

本咖啡豆购物商城系统的订单管理模块流程图如图 4.6所示

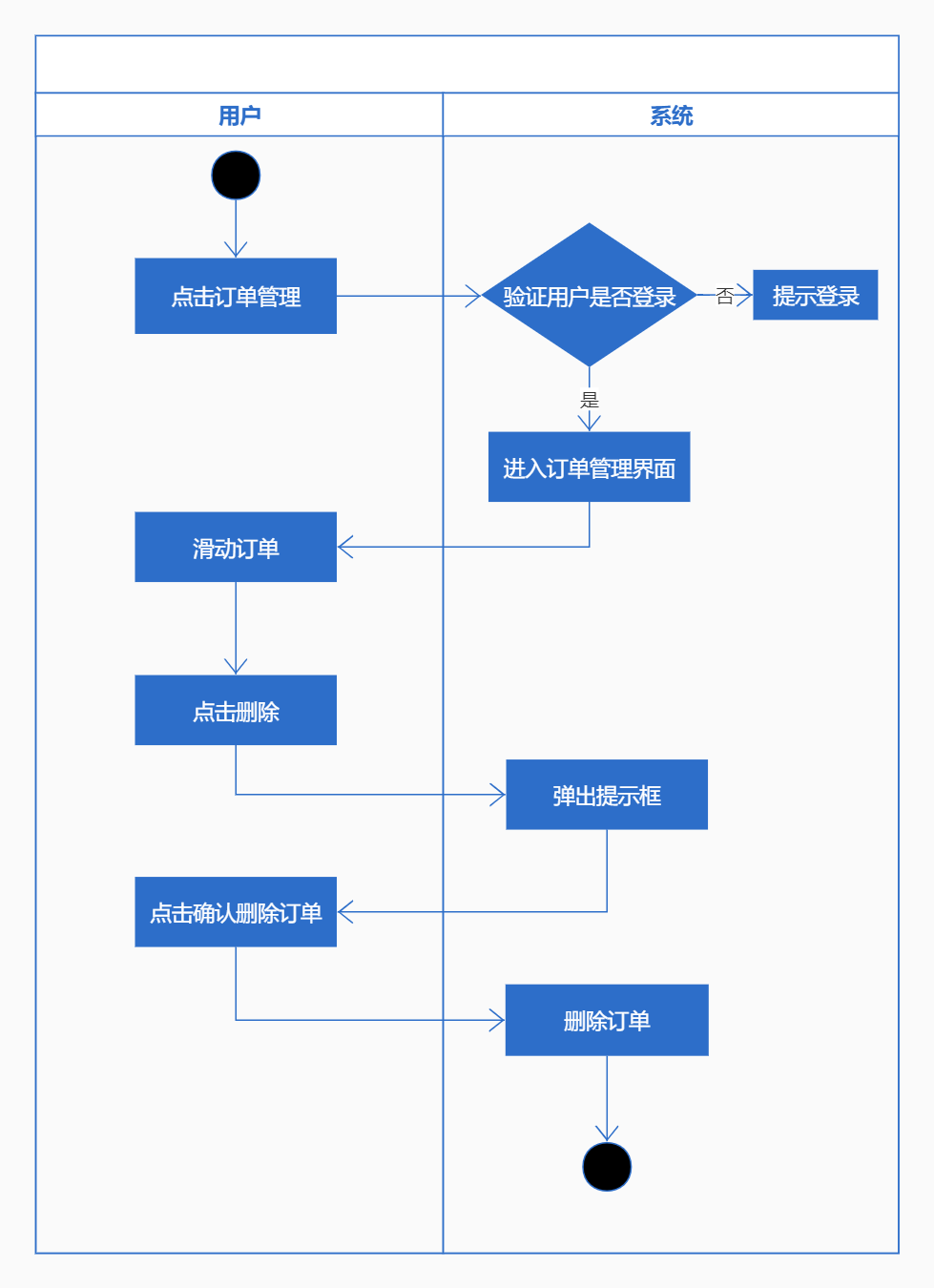


图 5.6

## 收藏夹模块

收藏夹这个模块基本上所有的购物商城系统都有，基本都是为了用户在收藏商品之后，商品失效了，或者是售罄的时候，可以方便用户进到收藏夹模块里去取消收藏是失效的商品或者是售罄的商品亦或者是不想收藏的商品。

这样的设计是为了方便用户去管理自己收藏的商品，而不是说当用户想要取消收藏一个商品的时候，还要进到商品页面里面才能取消收藏，这会对用户的体验造成很大的影响，当用户觉得体验差的时候，就很有可能会再也不用你这个网站去购买商品，一个用户这样之后，就会一传十，十传百，造成大量的用户损失，顺带的造成经济损失，所以当设计一个功能模块的时候，要时时刻刻的考虑到用户的体验感是否感到良好，这是首要的目标。

本咖啡豆购物商城系统的收藏夹模块流程图如图 4.7所示。

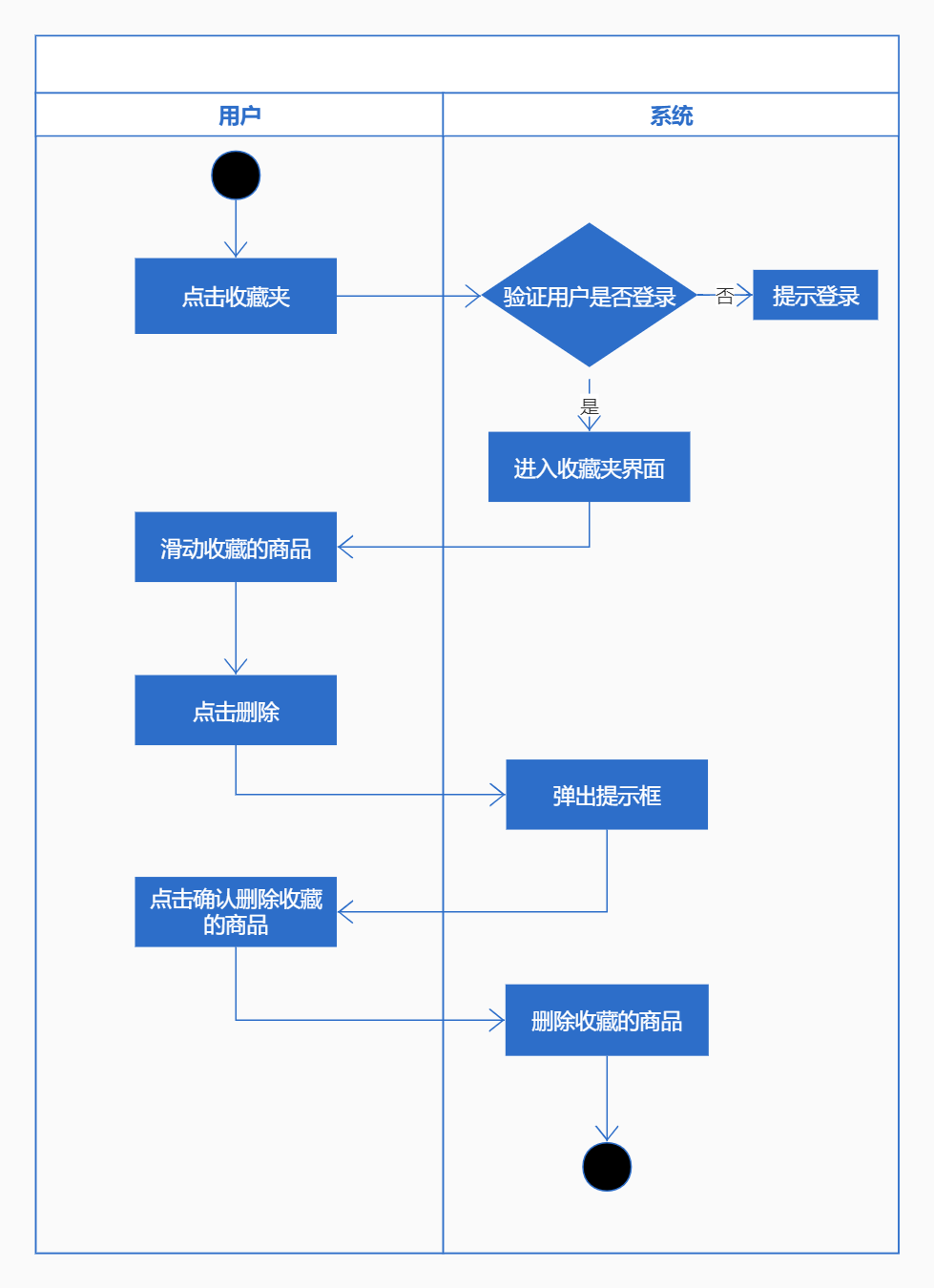


图 5.7

## 地址管理模块

地址是每一个用户都要用到的信息，也是购买商品时必填的信息，如果用户购买商品之后没有填写地址信息的话，那么商家就不知道该把商品寄送到哪里去，所以地址信息是必要的[9]。

那么也要让用户可以去管理自己的地址信息才算是一个合格商城系统，所以设计了这一个模块来让用户去自己管理自己的地址信息，新增地址信息或者是修改地址信息，这样就可以方便用户想要把购买的商品寄去哪里就寄去哪里，防止用户体验差。

本咖啡豆购物商城系统的地址管理模块流程图如图 4.8所示。

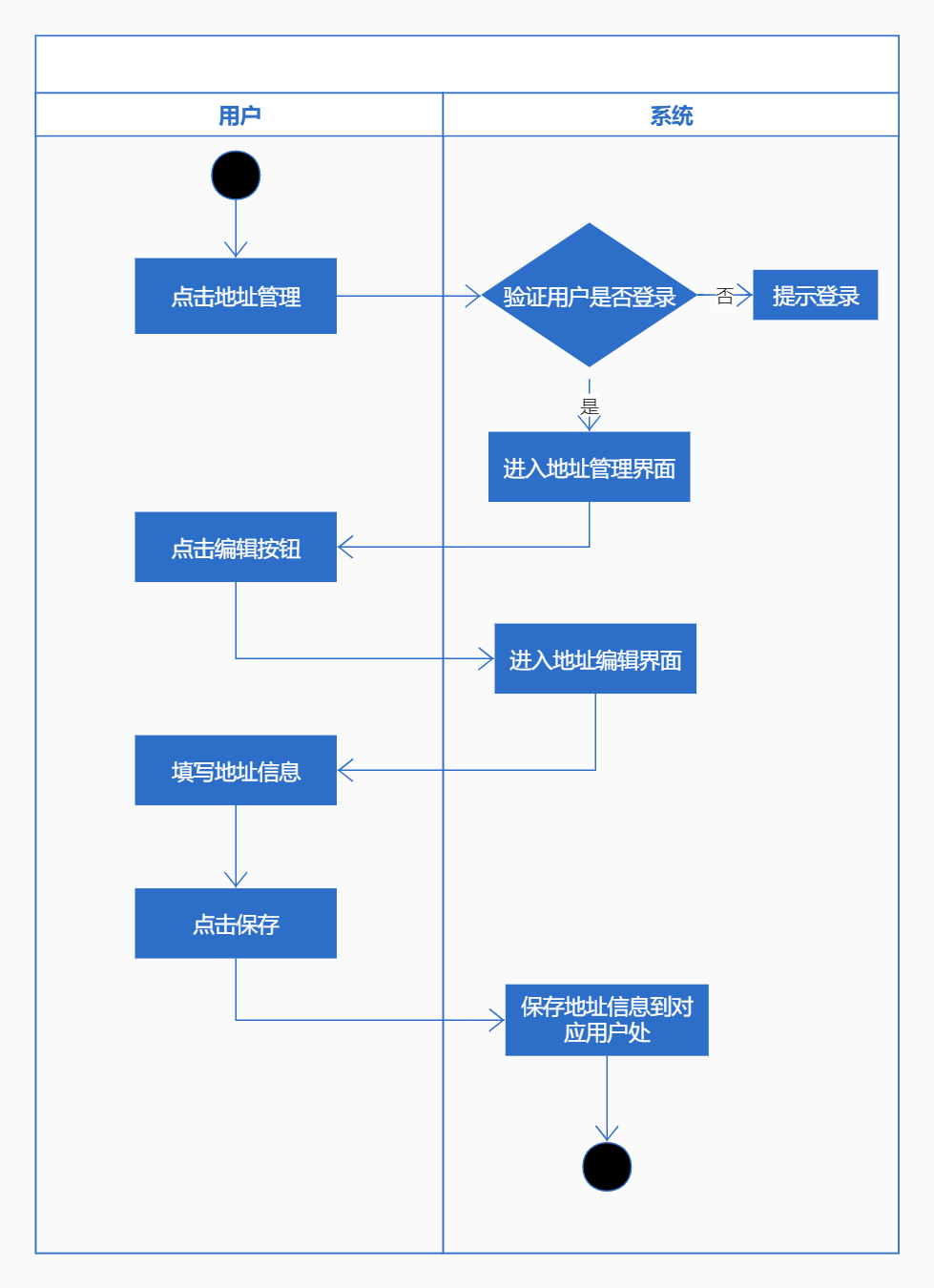
****

图 5.8

## 购物车模块

购物车模块可以说是整个商城系统中的核心之一了，在各个大型的购物商城网站里面必定是有购物车这一个模块的，因为购物车能给用户提升不少的体验感。

这样的设计师为了让用户在购物车中可以暂时保存自己想要购买的商品，当用户把想买的商品都加入到购物车之后，再去购物车提交订单，就可以省去一个一个商品在对应的商品页面去单独下单的麻烦。

本咖啡豆购物商城系统的购物车模块流程图如图 4.9所示。

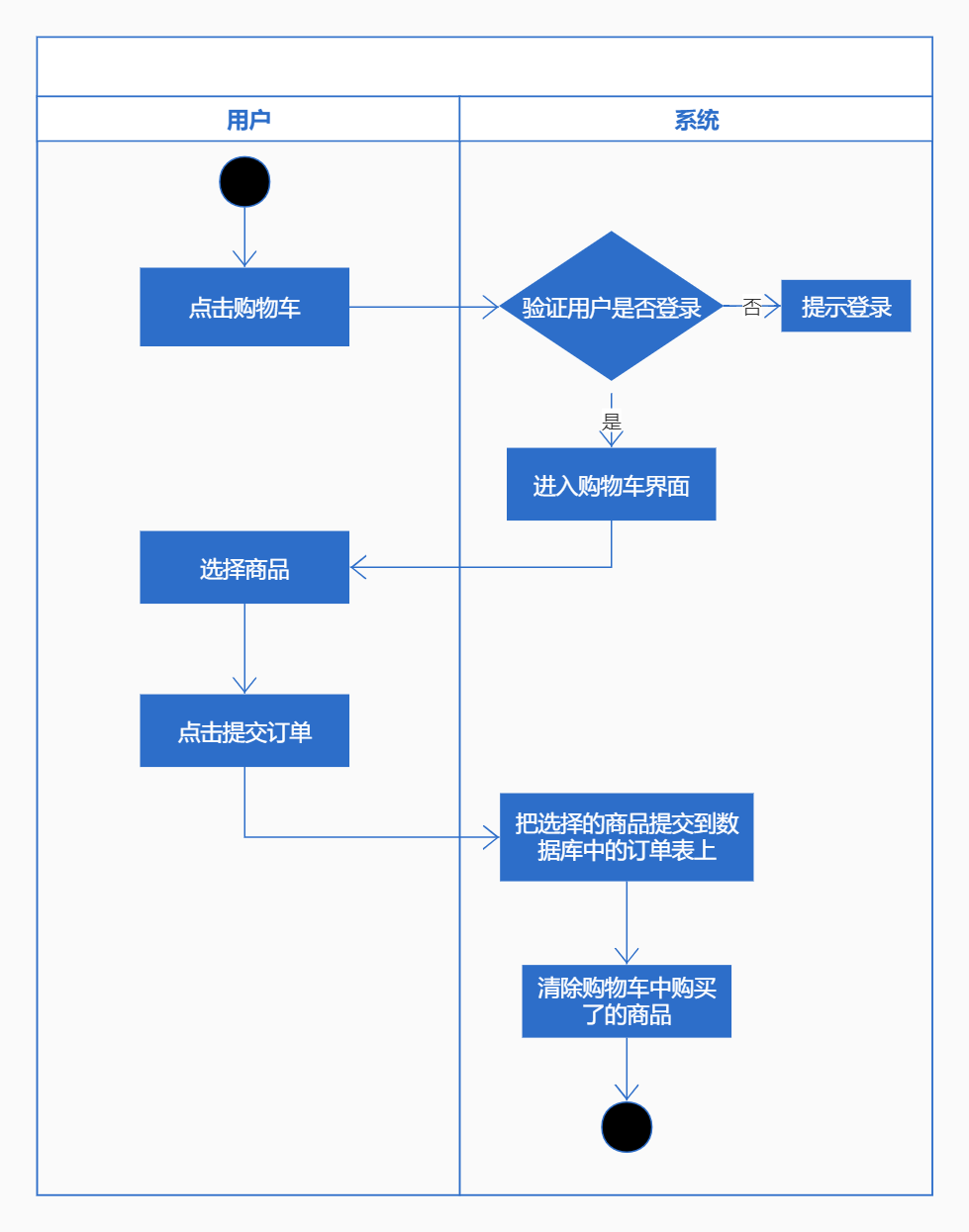


图 5.9

# 数据库设计

## 数据库结构

本咖啡豆购物商城系统后端数据库以MySQL为基础，构建了五个数据库表用来存储本咖啡豆购物商城系统所需要的各种信息，包括用户信息和商品信息。因此使用了五张数据库表来存储购物商城系统的信息，其中两张是主表，包含用户信息和商品信息，其他三张表都是副表，是为了“我的”里面的四个功能模块而建立的表，里面包含订单信息、收藏夹信息和购物车信息的表[10]。

本咖啡豆购物商城系统数据库表逻辑结构分别如图 6.1、图 6.2、图 6.3、图 6.4、图 6.5所示

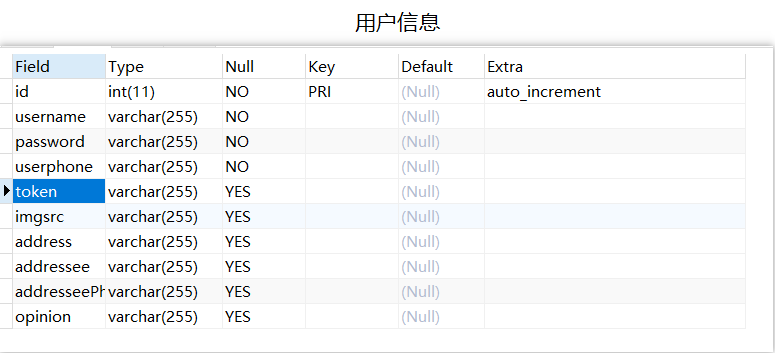


图 6.1

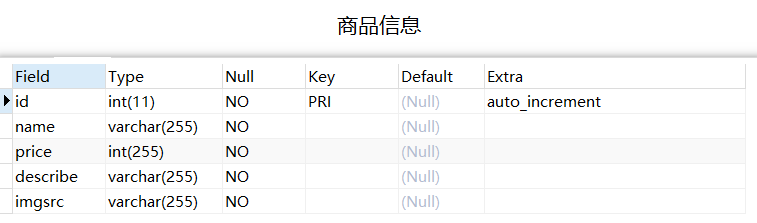


图 6.2

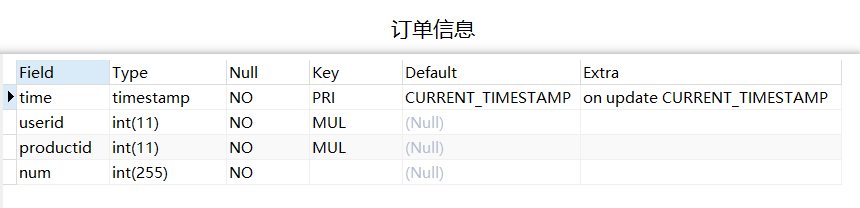


图 6.3

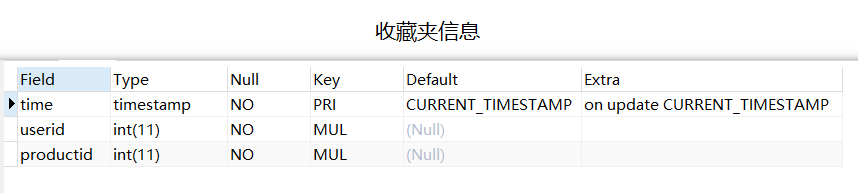


图 6.4

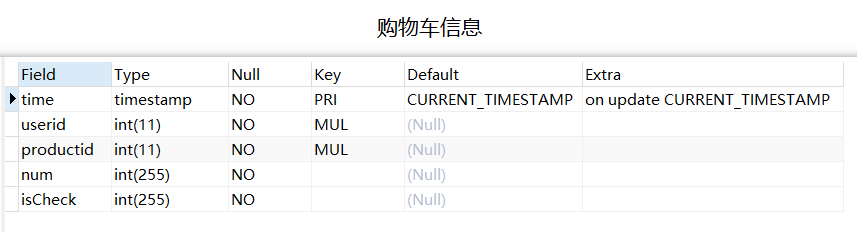


图 6.5

## 数据库表的设计及关系

本咖啡豆购物商城系统的数据库中的表总共有五张，其中一个是用户的个人信息表，另一个是商品的信息表，这两个表是整个数据库的核心，也是五个表当中的主表，另外三个表都是跟住两个主表，依靠外键来获取主表信息的副表。

下面将把每一个表的信息及关系一一介绍。

## 用户信息表

用户信息表是所有表中的主表之一，其中含有用户id、用户名、密码、电话等的用户信息，整个表的主键是用户id，因为节省数据库的空间，所以把用户的地址信息和用户的意见反馈也合并到了用户信息表里面方便管理，也方便查询。其中的用户id设置自动递增，在新增用户的时候，数据库会自动给用户添加一个用户id，无需前端给用户添加用户id，简化了前端的代码。

本咖啡豆购物商城系统的数据库用户信息表设计如图 6.6所示。

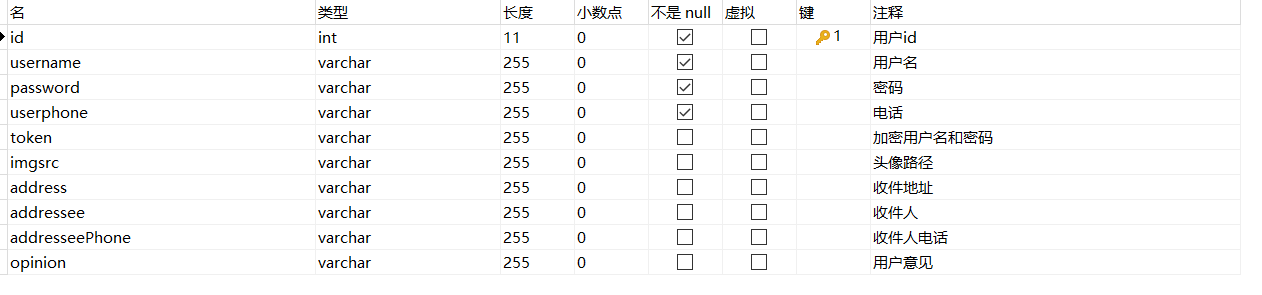


图 6.6

## 商品信息表

商品信息表是所有表中的主表之一，其中含有商品id、商品名称、商品价格等的信息，整个表的主键是商品id，查询表中信息就是靠商品id去查询的。

本咖啡豆购物商城系统的数据库商品信息表设计如图 6.7所示。



图 6.7

## 订单信息表

订单信息表是五张表中的副表之一，里面是通过外键链接商品信息表与用户信息表形成的表，表中主要包括提交订单的时间、用户id、商品id、商品数量。其中用户id是外键，用于与用户信息表相关联；商品id也是外键，用于与商品信息表相关联；加入订单的时间设置根据当前时间戳更新，在提交订单的时候，数据库会自动给订单添加一个提交订单的时间，无需前端给订单添加提交订单的时间，简化了前端的代码。

本咖啡豆购物商城系统的数据库订单信息表设计如图 6.8、图 6.9所示。



图 6.8



图 6.9

## 收藏夹信息表

收藏夹信息表是五张表中的副表之一，里面也是通过外键链接商品信息表与用户信息表形成的表，表中主要包括用户收藏商品的时间、用户id、商品id。其中用户id是外键，用于与用户信息表相关联；商品id也是外键，用于与商品信息表相关联；加入收藏夹的时间设置根据当前时间戳更新，在商品加入收藏夹的时候，数据库会自动给收藏的商品添加一个商品加入收藏夹的时间，无需前端给收藏的商品添加商品加入收藏夹的时间，简化了前端的代码。

本咖啡豆购物商城系统的数据库收藏夹信息表设计如图 6.10、图 6.11所示。



图 6.10



图 6.11

## 购物车信息表

购物车信息表是五张表中的副表之一，里面也是通过外键链接商品信息表与用户信息表形成的表，表中主要包括商品加入购物车的时间、用户id、商品id、商品数量和商品是否选中的值。其中用户id是外键，用于与用户信息表相关联；商品id也是外键，用于与商品信息表相关联；商品是否被选中的值这个字段是为了在写前端页面的时候，可以方便判断购物车中的商品是否被选中；加入购物车的时间设置根据当前时间戳更新，在商品加入购物车的时候，数据库会自动给加入购物车的商品添加一个商品加入购物车的时间，无需前端给加入购物车的商品添加商品加入购物车的时间，简化了前端的代码。

本咖啡豆购物商城系统的数据库购物车信息表设计如图 6.12、图 6.13所示。



图 6.12



图 6.13

# 系统实现

## 后端接口的实现

## 封装数据库信息

为了方便复用，使用模块化管理，把数据库的服务器ip、用户名密码等的信息封装成一个db.js文件，方便后面编写接口的时候复用。

db.js代码如下：

**module.exports = {**

**mysql: {**

**host: "127.0.0.1", *// 服务器ip***

**user: "root", *// 用户名***

**password: "root", *// 密码***

**port: "3306", *//连接数据库的端口***

**database: "graduation-project", *// 数据库名称***

**},**

**};**

## 封装post请求响应操作

在后端接口中将会有比较多的post请求为了方便复用post请求响应之后的操作，把这个操作封装成了jsonWrite.js文件

jsonWrite.js代码如下：

**module.exports = function (res, ret) {**

**if (typeof ret === "undefined") {**

**res.json({**

**code: 1,**

**msg: "操作失败",**

**});**

**} else {**

**res.json({**

**code: 200,**

**msg: "操作成功",**

**});**

**}**

**};**

## 封装SQL语句

在编写后端接口中，不管是post请求还是get请求都是需要用到SQL语句去操作或者查询数据库内的数据，所以为了方便使用，我把所有的SQL语句都封装成了sqlmap.js文件，当我要使用的时候只需要引入文件，然后点出来就可以使用了，不需要每次都在接口处编写SQL语句，导致接口文件的代码冗余、难看，这也是一种优化[11]。

sqlmap.js代码如下：

***// sqlmap.js***

**var sqlMap = {**

***// 用户***

**user: {**

***// search:"select \* from userinfo",***

**search: "select \* from userinfo where username=?", *//查询表中是否有***

**search\_id: "select \* from userinfo where id=?", *//查询表中是否有***

**add: "insert into userinfo(username,password,userphone,token) values (?,?,?,?)", *//向数据库的某个表插入数据***

**upheadPortrait: "UPDATE userinfo SET imgsrc = ? WHERE id = ?",**

**},**

***// 地址***

**address: {**

**upaddress: "UPDATE userinfo SET addresseePhone = ?,address = ?,addressee = ? WHERE id = ?", *//修改userinfo中的地址***

**},**

***// 意见***

**feedback: {**

**upfeedback: "UPDATE userinfo SET opinion = ? WHERE id = ?", *//修改userinfo中的意见***

**},**

***// 订单***

**orderform: {**

**search: "SELECT \* FROM productinfo p RIGHT JOIN userproducts u on p.id=u.productid WHERE userid = ? ORDER BY time", *// 查询用户的订单***

**add: "insert into userproducts(userid,productid,num) values (?,?,?)",**

**},**

***// 收藏夹***

**favorites: {**

**search: "SELECT \* FROM productinfo p RIGHT JOIN favorites f on p.id=f.productid WHERE userid = ? ORDER BY time", *// 查询用户的收藏夹***

**add: "insert into favorites(userid,productid) values (?,?)",**

**delete: "DELETE FROM favorites WHERE userid=? AND productid=?",**

**},**

***// 商品***

**products: {**

**search: "select \* from productinfo",**

**add: "insert into productinfo(name,price,describe,imgsrc) values (?,?,?,?)",**

**delete: "DELETE FROM productinfo WHERE id=?",**

**},**

***// 购物车***

**shoppingCart: {**

**search: "SELECT \* FROM productinfo p RIGHT JOIN `shopping-cart` s on p.id=s.productid WHERE userid = ? ORDER BY time", *// 查询购物车信息***

**p\_search: "SELECT \* FROM productinfo p RIGHT JOIN `shopping-cart` s on p.id=s.productid WHERE userid = ? and productid = ?", *// 查询购物车信息***

**add: "insert into `shopping-cart`(userid,productid,num) values (?,?,?)", *//购物车添加商品***

**upshopcart: "UPDATE `shopping-cart` SET num = ? WHERE productid = ? and userid = ?", *//修改userinfo中的地址***

**delete: "DELETE FROM `shopping-cart` WHERE userid=?,productid=?", *//删除购物车中的商品***

**},**

**};**

**module.exports = sqlMap;**

## 封装有关用户信息的接口

把有关用户信息的接口都封装成user.js文件，在主文件index.js中引入使用即可，这样封装可以使代码看起来更加的整洁，也会加快整个接口的运行速度。

user.js核心代码如下：

**var conn;**

**function handleError() {**

***//创建一个mysql连接对象***

**conn = mysql.createConnection(models.mysql);**

***//连接错误，2秒重试***

**conn.connect(function(err) {**

**if (err) {**

**console.log("error when connecting to db:", err);**

**setTimeout(handleError, 2000);**

**}**

**});**

***//监听错误***

**conn.on("error", function(err) {**

**console.log("db error", err);**

***// 如果是连接断开，自动重新连接***

**if (err.code === "PROTOCOL\_CONNECTION\_LOST") {**

**handleError();**

**} else {**

**throw err;**

**}**

**});**

**console.log("连接成功");**

**}**

**handleError();**

**const SECRET\_KEY = "JahoCaffee";**

***// 登录/查询用户的信息***

**router.get("/userinfo", function(req, res) {**

**let sql = $sql.user.search;**

**let params = req.query;**

**conn.query(sql, [params.username], function(err, result) {**

**if (err) {**

**res.json({ msg: "查询失败", code: 0 });**

**}**

**if (result) {**

**res.json({ msg: "查询成功", code: 200, data: result });**

**}**

**});**

**});**

***// 登录/查询用户的信息***

**router.get("/userinfo/id", function(req, res) {**

**let sql = $sql.user.search\_id;**

**let params = req.query;**

**conn.query(sql, [params.id], function(err, result) {**

**if (err) {**

**res.json({ msg: "查询失败", code: 0 });**

**}**

**if (result) {**

**res.json({ msg: "查询成功", code: 200, data: result });**

**}**

**});**

**});**

***// 用户的注册***

**router.post("/adduser", (req, res) => {**

**var sql = $sql.user.add;**

**var parms = req.body;**

**let token = jsonWebToken.sign(**

**{**

**username: parms.username,**

**userphone: parms.userphone,**

**},**

**SECRET\_KEY,**

**{**

***// expiresIn: "24h", //token有效期***

**expiresIn: 60 \* 60 \* 24 \* 365, *// 两种写法,***

***// algorithm:"HS256"  默认使用 "HS256" 算法***

**}**

**);**

**conn.query(sql, [parms.username, parms.password, parms.userphone, token], function(err, result) {**

**if (err) {**

**res.json({ msg: "注册失败", code: 0 });**

**}**

**if (result) {**

**jsonWrite(res, result);**

**}**

**});**

**});**

***// 头像更改***

**router.post("/user/upheadPortrait", (req, res) => {**

**var sql = $sql.user.upheadPortrait;**

**var parms = req.body;**

**if (req.method == "POST") {**

**var data = [];**

**req.on("data", chunk => {**

**data.push(chunk);**

**});**

**req.on("end", () => {**

**var buffer = Buffer.concat(data);**

**fs.writeFile("./a", buffer, err => {**

**if (!err) {**

**res.end("ok");**

**}**

**});**

**});**

**}**

**conn.query(sql, [imgsrc, parms.id], function(err, result) {**

**if (err) {**

**res.json({ msg: "头像更改失败", code: 0 });**

**}**

**if (result) {**

**jsonWrite(res, result);**

**}**

**});**

**});**

***// 用户地址管理***

**router.post("/addAddress", (req, res) => {**

**var sql = $sql.address.upaddress;**

**var parms = req.body;**

**conn.query(sql, [parms.addresseePhone, parms.address, parms.addressee, parms.id], function(err, result) {**

**if (err) {**

**res.json({ msg: "地址添加失败", code: 0 });**

**}**

**if (result) {**

**jsonWrite(res, result);**

**}**

**});**

**});**

***// 用户的留言***

**router.post("/feedback", (req, res) => {**

**var sql = $sql.feedback.upfeedback;**

**var parms = req.body;**

***// let opinion = parms.opinion.toString();***

***// console.log(opinion);***

**conn.query(sql, [parms.opinion, parms.id], function(err, result) {**

**if (err) {**

**res.json({ msg: "留言失败", code: 0 });**

**}**

**if (result) {**

**jsonWrite(res, result);**

**}**

**});**

**});**

***// 订单查询***

**router.get("/orderform/search", function(req, res) {**

**let sql = $sql.orderform.search;**

**let params = req.query;**

**conn.query(sql, [params.userid], function(err, result) {**

**if (err) {**

**res.json({ msg: "查询失败", code: 0 });**

**}**

**if (result) {**

**res.json({ msg: "查询成功", code: 200, data: result });**

**}**

**});**

**});**

***// 提交订单***

**router.post("/orderform/add", (req, res) => {**

**var sql = $sql.orderform.add;**

**var parms = req.body;**

**conn.query(sql, [parms.userid, parms.productid, parms.num], function(err, result) {**

**if (err) {**

**res.json({ msg: "提交订单失败", code: 0 });**

**}**

**if (result) {**

**jsonWrite(res, result);**

**}**

**});**

**});**

***// 查询收藏夹信息***

**router.get("/favorites/search", function(req, res) {**

**let sql = $sql.favorites.search;**

**let params = req.query;**

**conn.query(sql, [params.userid], function(err, result) {**

**if (err) {**

**res.json({ msg: "查询失败", code: 0 });**

**}**

**if (result) {**

**res.json({ msg: "查询成功", code: 200, data: result });**

**}**

**});**

**});**

***// 收藏夹添加***

**router.post("/favorites/add", (req, res) => {**

**var sql = $sql.favorites.add;**

**var parms = req.body;**

**conn.query(sql, [parms.userid, parms.productid], function(err, result) {**

**if (err) {**

**res.json({ msg: "收藏失败", code: 0 });**

**}**

**if (result) {**

**jsonWrite(res, result);**

**}**

**});**

**});**

***// 收藏夹删除***

**router.post("/favorites/del", (req, res) => {**

**var sql = $sql.favorites.delete;**

**var parms = req.body;**

**conn.query(sql, [parms.userid, parms.productid], function(err, result) {**

**console.log(parms.productid);**

**if (err) {**

**res.json({ msg: "取消收藏失败", code: 0 });**

**console.log(err);**

**}**

**if (result) {**

**jsonWrite(res, result);**

**}**

**});**

**});**

## 封装有关购物车信息的接口

把有购物车信息的接口都封装成shoppingCart.js文件，在主文件index.js中引入使用即可，这样封装可以使代码看起来更加的整洁，也会加快整个接口的运行速度。

shoppingCart.js核心代码如下：

***// 修改商品数量***

**router.post("/update", (req, res) => {**

**var sql = $sql.shoppingCart.upshopcart;**

**var parms = req.body;**

**conn.query(sql, [parms.num, parms.productid, parms.userid], function (err, result) {**

**if (err) {**

**res.json({ msg: "商品修改失败", code: 0 });**

**}**

**if (result) {**

**jsonWrite(res, result);**

**}**

**});**

**});**

***// 删除购物车中的商品***

**router.post("/del", (req, res) => {**

**var sql = $sql.shoppingCart.delete;**

**var parms = req.body;**

**conn.query(sql, [parms.userid, parms.productid], function (err, result) {**

**if (err) {**

**res.json({ msg: "商品删除失败", code: 0 });**

**}**

**if (result) {**

**jsonWrite(res, result);**

**}**

**});**

**});**

## 封装有关商品信息的接口

把有商品信息的接口都封装成product.js文件，在主文件index.js中引入使用即可，这样封装可以使代码看起来更加的整洁，也会加快整个接口的运行速度。

product.js核心代码如下：

***// 查询商品信息***

**router.get("/search", function (req, res) {**

**let sql = $sql.products.search;**

**conn.query(sql, function (err, result) {**

**if (err) {**

**res.json({ msg: "查询失败", code: 0 });**

**}**

**if (result) {**

**res.json({ msg: "查询成功", code: 200, data: result });**

**}**

**});**

**});**

## 前端页面的实现

## 前端页面介绍

整个前端页面分为三大模块：首页、购物车和我的，主题颜色以偏天蓝色为主，整体以清新简洁的风格设计。“我的”里面还有五个小模块：订单管理、地址管理、收藏夹、意见反馈和登录注册。

## 首页页面

首页当中有一个轮播图，下面还有全部商品的展示。

首页页面如图 7.1所示。

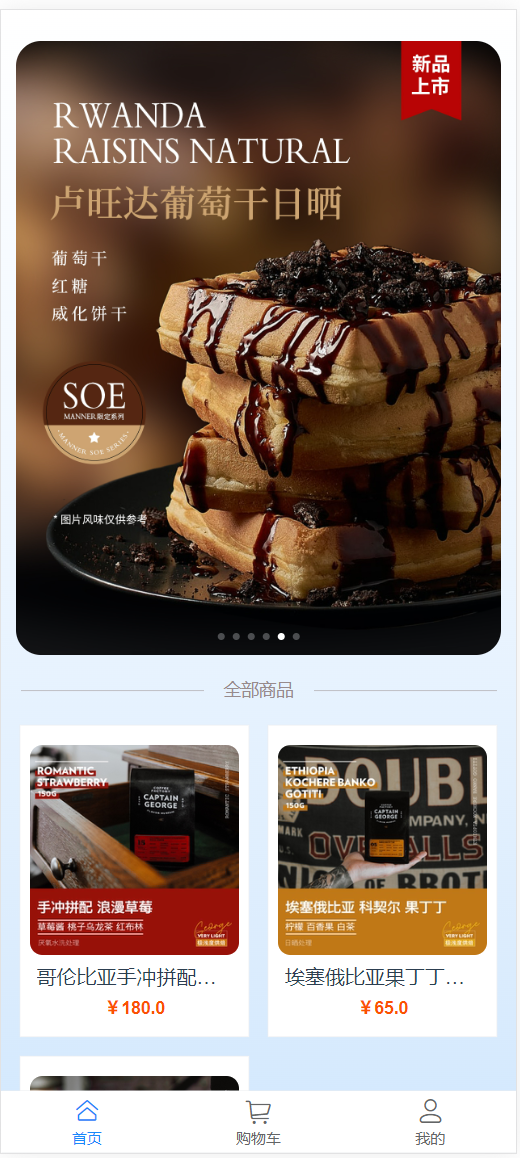


图 7.1

Html结构代码：

**<template>**

**<div class="home">**

***<!-- <form action="/">***

***<van-search v-model="value" shape="round" placeholder="请输入搜索关键词" @search="onSearch" />***

***</form> -->***

**<HomeSwiper name="swiper"></HomeSwiper>**

**<HomeProducts></HomeProducts>**

**<div class="box"></div>**

**</div>**

**</template>**

## 购物车页面

购物车里面放着的是用户加入购物车的商品，并且可以提交订单和全选，顶上有一个优惠通知栏。

购物车页面如图 7.2所示



图 7.2

Html结构代码：

**<template>**

**<div class="shop">**

**<div class="title">购物车</div>**

***<!-- 滚动通知 -->***

**<van-notice-bar left-icon="volume-o" text="乔治队长 哥伦比亚手冲拼配咖啡豆草莓蜜桃双重厌氧冷泡150g包邮 限时8折优惠" class="inform" />**

***<!-- 商品 -->***

**<ShopProducts></ShopProducts>**

***<!-- 提交订单 -->***

**<van-submit-bar :price="0" button-text="提交订单" @submit="onSubmit" class="submit-your-order">**

**<van-checkbox v-model="checkAll">全选</van-checkbox>**

**</van-submit-bar>**

**</div>**

**</template>**

## 我的页面

我的页面里面最主要的就是登录注册模块，其余还有订单管理、地址管理、收藏夹、意见反馈四个模块。

我的页面如图 7.3所示。

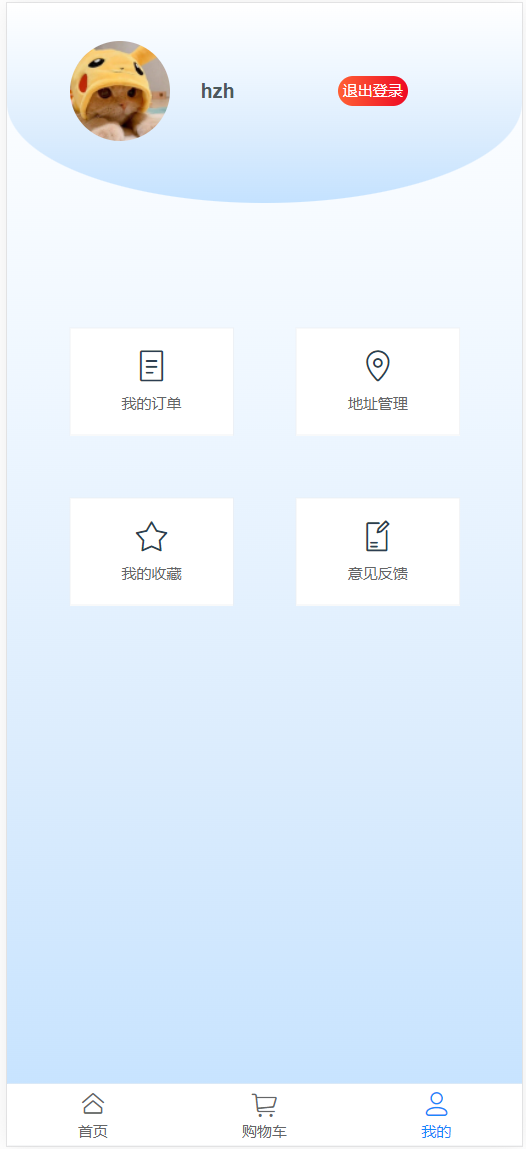


图 7.3

Html结构代码：

**<template>**

**<div class="mine" :key="refresh">**

**<router-view></router-view>**

**<div class="unlogin-box" v-show="!isShowlogin">**

**<div class="head-portrait" @click="shadeOpenClick">**

**<img**

**src="https://s2.loli.net/2022/03/23/L7AefwBqNhGSj3T.png"**

**alt=""**

**/>**

**</div>**

**<div class="userinfo" @click="shadeOpenClick">**

**<div class="username">未登录</div>**

**<div class="gt">&gt;</div>**

**</div>**

**</div>**

**<div class="login-box" v-show="isShowlogin">**

**<div class="head-portrait">**

***<!-- <van-uploader :after-read="afterRead" /> -->***

**<img**

**src="https://s2.loli.net/2022/03/18/W9TdYPSBqgyfOi8.jpg"**

**alt=""**

**/>**

**</div>**

**<div class="userinfo">**

**<div class="l\_username">{{ userInfo.username }}</div>**

**<van-button**

**class="relogin"**

**type="danger"**

**@click="relogin"**

**size="mini"**

**color="linear-gradient(to right, #ff6034, #ee0a24)"**

**:round="true"**

**>退出登录</van-button**

**>**

**</div>**

**</div>**

***<!-- 四宫格 -->***

**<van-grid :gutter="50" column-num="2" class="grid">**

**<van-grid-item**

**icon="orders-o"**

**text="我的订单"**

**class="grid"**

**@click="fm('order')"**

**/>**

**<van-grid-item**

**icon="location-o"**

**text="地址管理"**

**class="grid"**

**@click="fm('address')"**

**/>**

**<van-grid-item**

**icon="star-o"**

**text="我的收藏"**

**class="grid"**

**@click="fm('collect')"**

**/>**

**<van-grid-item**

**icon="records"**

**text="意见反馈"**

**class="grid"**

**@click="fm('feedback')"**

**/>**

**</van-grid>**

***<!-- 遮罩 -->***

**<van-overlay :show="show" @click="(show = false), (isShowModal = true)">**

***<!-- 登录模态框 -->***

**<div class="login-modal" @click.stop v-show="isShowModal">**

**<div class="back" @click="show = false">×</div>**

**<input**

**type="text"**

**name=""**

**id="username"**

**placeholder="请输入用户名"**

**v-model="username"**

**/>**

**<input**

**type="password"**

**name=""**

**id="psd"**

**placeholder="请输入密码"**

**v-model="password"**

**ref="l\_psd\_inp"**

**/>**

**<drag-verify**

**ref="dragVerify7"**

**:height="34"**

**:width="201"**

**:isPassing.sync="isPassing7"**

**completedBg="#93a6ff"**

**text="请按住滑块拖动"**

**successText="验证通过"**

**handlerIcon="el-icon-d-arrow-right"**

**successIcon="el-icon-circle-check"**

**handlerBg="#fff"**

**@passcallback="passcallback3"**

**>**

**</drag-verify>**

**<div class="login" @click="loginClick">登录</div>**

**<div class="signin" @click="isShowModal = false">注册</div>**

**</div>**

***<!-- 注册模态框 -->***

**<div class="sigin-modal" @click.stop v-show="!isShowModal">**

**<div class="back" @click="(show = false), (isShowModal = true)">**

**×**

**</div>**

**<input**

**type="text"**

**name=""**

**id="s\_username"**

**placeholder="请输入用户名"**

**v-model="s\_username"**

**/>**

**<input**

**type="password"**

**name=""**

**id="s\_psd"**

**placeholder="请输入密码"**

**v-model="s\_password"**

**/>**

**<input**

**type="text"**

**name=""**

**id="s\_phone"**

**placeholder="请输入手机号"**

**v-model="s\_userphone"**

**/>**

**<drag-verify**

**ref="dragVerify7"**

**:height="34"**

**:width="201"**

**:isPassing.sync="isPassing7"**

**completedBg="#93a6ff"**

**text="请按住滑块拖动"**

**successText="验证通过"**

**handlerIcon="el-icon-d-arrow-right"**

**successIcon="el-icon-circle-check"**

**handlerBg="#fff"**

**@passcallback="passcallback3"**

**>**

**</drag-verify>**

**<div class="login" @click="sigin">注册</div>**

**<div class="signin" @click="isShowModal = true">登录</div>**

**</div>**

**</van-overlay>**

**</div>**

**</template>**

## 收藏夹页面

Html结构代码：

**<div class="collect">**

***<!-- 导航栏 -->***

**<van-nav-bar title="收藏夹" left-text="返回" left-arrow @click-left="onClickLeft" />**

**<van-swipe-cell v-for="item in favorites" :key="item.id"**

**><van-card :price="item.price + '.00'" :desc="item.describe" :title="item.name" :thumb="item.imgsrc" />**

**<template #right>**

**<van-button square text="取消收藏" type="warning" class="delete-button" @click="unSell(item.productid, item.name)" />**

**</template>**

**</van-swipe-cell>**

**</div>**

核心代码：

**unSell(productid, message) {**

**Dialog.confirm({**

**title: "确定要取消收藏",**

**message,**

**})**

**.then(async () => {**

**let userid = localStorage.getItem("id");**

**let res = await unSellAPI({ userid, productid });**

**if (res.code == 200) {**

**Notify({ type: "success", message: "取消收藏成功" });**

**await this.getFavorites();**

***// this.$forceUpdate();***

**} else {**

**Notify({ type: "warning", message: "取消收藏失败" });**

**}**

**})**

**.catch(() => {**

***// on cancel***

**});**

**},**

**async getFavorites() {**

**let userid = localStorage.getItem("id");**

**let res = await getFavoritesAPI({ userid });**

**if (res.code == 200) {**

**this.favorites = res.data;**

**} else {**

**Notify({ type: "warning", message: res.msg });**

**}**

**},**

**},**

## 意见反馈页面

Html结构代码：

**<div class="feedback">**

***<!-- 导航栏 -->***

**<van-nav-bar title="意见反馈" left-text="返回" left-arrow @click-left="onClickLeft" />**

***<!-- 留言 -->***

**<van-field v-model="message" rows="6" autosize label="留言" type="textarea" maxlength="500" placeholder="请输入留言" show-word-limit />**

**<div class="btn-container" @click="submitMessage">**

**<a href="#" class="btn-3d blue">提交</a>**

**</div>**

**</div>**

核心代码：

**async submitMessage() {**

**let id = localStorage.getItem("id");**

**let opinion = this.message;**

**if (opinion == "") {**

**Notify({ type: "warning", message: "请输入留言!" });**

**return;**

**}**

**let res = await submitMessageAPI({**

**id,**

**opinion,**

**});**

**if (res.code == 200) {**

**Toast.success("留言成功");**

**this.message = "";**

**} else {**

**Toast.fail("留言失败");**

**}**

**},**

## 订单管理页面

Html结构代码：

**<div class="order">**

***<!-- 导航栏 -->***

**<van-nav-bar title="我的订单" left-text="返回" left-arrow @click-left="onClickLeft" />**

**<van-card**

**:num="item.num"**

**:price="item.price + '.00'"**

**:desc="item.describe"**

**:title="item.name"**

**:thumb="item.imgsrc"**

**v-for="item in orderForm"**

**:key="item.id"**

**/>**

**</div>**

核心代码：

**async created() {**

**let userid = localStorage.getItem("id");**

**let res = await getOrderFormAPI({ userid });**

***// console.log(res);***

**if (res.code == 200) {**

**this.orderForm = res.data;**

**} else {**

**Notify({ type: "warning", message: res.msg });**

**}**

**},**

**methods: {**

**onClickLeft() {**

**this.$router.push("/mine");**

**},**

**},**

0

## 登录注册框

登录的时候，登录框会显示账号输入框、密码输入框、滑块验证码、登录按钮以及注册按钮，注册的时候会比登录的时候多一个手机号码输入框。当点击登录的时候会校验两个输入框的内容是否合法，合法才能发送请求，如果未注册会提示先注册，已经注册过的会直接登录。注册和登录的逻辑差不多，就是发送的请求变成了向数据库增加用户的请求，当注册成功之后会自动登录。

登录注册框如所示。

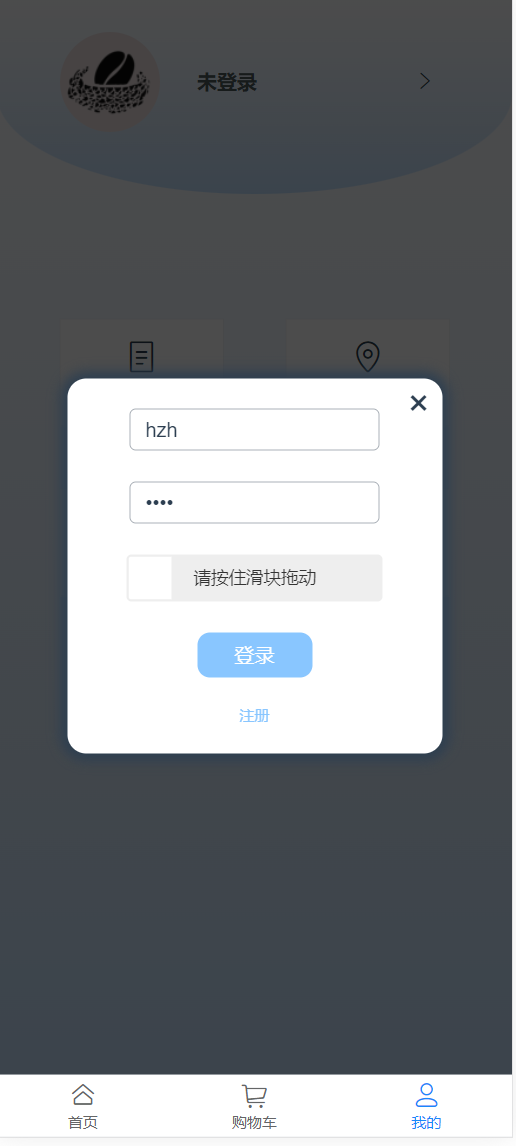


图 7.4



图 7.5

实现代码如下：

***// 登录模态框的显示***

**shadeOpenClick() {**

**this.show = true;**

**},**

***// 登录按钮***

**async loginClick() {**

***// 验证用户名是否为空***

**if (this.username == "") {**

**Notify({ type: "warning", message: "账号不能为空!" });**

**return;**

**}**

***// 请求用户数据,并用isGet判断是否已经请求过一次有内容的数据***

**if (this.isGet == false) {**

***// setTimeout(() => {***

**let res = await getUserInfoAPI({**

**username: this.username,**

**});**

**this.userInfo = res.data[0];**

**this.isGet = true;**

***// }, 100);***

**}**

***// 验证账号是否注册***

**if (this.userInfo == undefined || this.userInfo.length == 0) {**

**Notify({**

**type: "warning",**

**message: "您输入的账号未注册,请先注册账号!",**

**});**

***// this.isShowModal = false;***

**this.show = true;**

**this.isGet = false;**

***// console.log(this.userInfo);***

***// console.log(1);***

**return;**

**}**

***// 验证账号是否有误***

**if (this.username != this.userInfo.username) {**

**Notify({**

**type: "warning",**

**message: "您输入的账号有误或未注册!",**

**});**

**this.show = true;**

**this.isGet = false;**

**return;**

**}**

***// 验证密码是否为空***

**if (this.password == "") {**

**Notify({ type: "warning", message: "密码不能为空!" });**

**return;**

**}**

***// 密码正确性校验***

**if (this.password != this.userInfo.password) {**

**Notify({ type: "warning", message: "密码错，请重新输入！" });**

**this.password = "";**

**this.$nextTick(() => {**

***//正确写法***

**this.$refs.l\_psd\_inp.focus();**

**});**

***// this.reset\_d();***

**return;**

**}**

***// 滑块校验***

**if (this.isPass == false) {**

**Notify({ type: "warning", message: "请把滑块滑至正确的位置!" });**

**return;**

**}**

**let token = this.userInfo.token;**

**let id = this.userInfo.id;**

**localStorage.setItem("token", token);**

**localStorage.setItem("id", id);**

**this.isShowlogin = true;**

**this.show = false;**

**Notify({ type: "success", message: "登录成功!" });**

***// this.reset\_d();***

***// }***

**},**

***// 滑块验证码成功的回调函数***

**passcallback3() {**

**this.isPass = true;**

**},**

***// 退出登录***

**relogin() {**

**localStorage.removeItem("token");**

**localStorage.removeItem("id");**

***// 切换回未登录的状态***

**this.isShowlogin = false;**

***// 恢复验证码为未认证的状态***

**this.isPassing7 = false;**

**this.isPass = false;**

***// 更改key值实现页面重载以重置验证码未认证的状态***

**this.refresh++;**

**},**

**async sigin() {**

***// 验证用户名是否为空***

**if (this.s\_username == "") {**

**Notify({ type: "warning", message: "账号不能为空!" });**

**return;**

**}**

***// 验证用户名是否按要求填写***

**if (!validateUserName(this.s\_username)) {**

**Notify({**

**type: "warning",**

**message:**

**"账号必须字母开头，长度在 5-16 之间，可以使用字母数字下划线!",**

**});**

**return;**

**}**

***// 验证密码是否为空***

**if (this.s\_password == "") {**

**Notify({ type: "warning", message: "密码不能为空!" });**

**return;**

**}**

***// 验证密码是否按要求填写***

**if (!validatePassword(this.s\_password)) {**

**Notify({**

**type: "warning",**

**message:**

**"密码必须包含大小写字母和数字的组合，可以使用特殊字符，长度必须在5-15之间哟!",**

**});**

**return;**

**}**

***// 验证密码是否为空***

**if (this.s\_userphone == "") {**

**Notify({ type: "warning", message: "手机不能为空!" });**

**return;**

**}**

***// 验证手机号是否正确***

**if (!validatePhoneNumber(this.s\_userphone)) {**

**Notify({ type: "warning", message: "请输入正确的手机号!" });**

**return;**

**}**

***// 滑块校验***

**if (this.isPass == false) {**

**Notify({ type: "warning", message: "请把滑块滑至正确的位置!" });**

**return;**

**}**

***// 发送注册请求***

**let res = await userRegistrationAPI({**

**username: this.s\_username,**

**password: this.s\_password,**

**userphone: this.s\_userphone,**

**});**

**console.log(res.code);**

**if (res.code == 200) {**

***// 发送用户信息请求***

**let res1 = await getUserInfoAPI({**

**username: this.s\_username,**

**});**

**this.userInfo = res1.data[0];**

***// 设置token和id***

**let token = this.userInfo.token;**

**let id = this.userInfo.id;**

**localStorage.setItem("token", token);**

**localStorage.setItem("id", id);**

***// 切换为登录状态***

**this.isShowlogin = true;**

***// 关闭注册界面***

**this.show = false;**

**Notify({**

**type: "success",**

**message: "注册成功,已为您自动登录!",**

**});**

**} else {**

**Notify({ type: "warning", message: "注册失败!" });**

**}**

**},**

## 封装请求拦截器以及API

通常在一个项目里面，常常会有许多的请求需要发送，就像上面的登录注册就要发送两个请求，当发送的请求多了之后，如果把每一个请求都写一遍在发送请求的位置，那么代码就会极其的臃肿、不好看并且还麻烦，每次都要重新写一遍一模一样的代码，而且还会影响整个系统的运行效率，因此，为了提升系统的运行效率，也为了代码的好看，我选择把请求封装起来，并把所有的请求都拦截下来，在发送请求时拦截、响应时拦截，并统一作出处理。我还把API封装了起来，每一个API都经过拦截器再回到前端页面。这样编写代码有利于系统运行速度的提升。

request.js核心代码如下：

***// 创建axios实例(instance)***

**const instance = axios.create({**

**baseURL: "/api",**

**timeout: 5000,**

**});**

***// 请求拦截器***

**instance.interceptors.request.use(**

**config => {**

***// 在请求之前做的操作***

**console.log(config);**

**return config;**

**},**

**err => {**

**return Promise.reject(err);**

**}**

**);**

***// 响应拦截器***

**instance.interceptors.response.use(**

**res => {**

***// 响应之后,做一些数据的统一处理***

**let res\_data = res.data;**

**let arr = [200];**

**if (!arr.includes(res\_data.code)) {**

**return false;**

**}**

**return res\_data;**

**},**

**err => {**

**Notify({ type: "danger", message: "响应错误" });**

**return Promise.reject(err);**

**}**

**);**

api.js核心代码如下：

***// 发起获取用户信息请求(使用用户名查询)***

**export const getUserInfoAPI = params => instance.get("/Mine/userinfo", { params });**

***// 发起获取用户信息请求(使用id查询)***

**export const idGetUserInfoAPI = params => instance.get("/Mine/userinfo/id", { params });**

***// 上传头像***

**export const uploadHeadImgAPI = params => instance.post("/user/upheadPortrait", qs.stringify(params));**

***// 用户注册***

**export const userRegistrationAPI = params => instance.post("/Mine/adduser", qs.stringify(params));**

***// 获取用户购物车的数据***

**export const getShopCartAPI = params => instance.get("/ShoppingCart/search", { params });**

***// 获取商品列表***

**export const getProductInfoAPI = () => instance.get("/Product/search");**

***// 发送留言请求***

**export const submitMessageAPI = params => instance.post("/Mine/feedback", qs.stringify(params));**

***// 获取用户订单列表***

**export const getOrderFormAPI = params => instance.get("/Mine/orderform/search", { params });**

***// 获取用户收藏夹列表***

**export const getFavoritesAPI = params => instance.get("/Mine/favorites/search", { params });**

***// 发起取消收藏请求***

**export const unSellAPI = params => instance.post("/Mine/favorites/del", qs.stringify(params));**

***// 修改用户地址***

**export const updateAddressAPI = params => instance.post("/Mine/addAddress", qs.stringify(params));**

# 测试

对系统进行测试是为了发现本系统中可能潜在但是并未发现的错误和缺陷，在系统上线之前进行测试可以保证系统的可用性和安全性[12]。

本咖啡豆购物商城系统为了保证每个功能模块的可用性和准确性，也进行了系统的测试从各方面测试本系统的功能是否可用，以及界面的显示是否正常。本系统的测试使用黑盒测试的方式来进行。

## 测试内容

本系统开发完成后，部署在 window10 操作系统上进行测试。数据库使用 MySQL5.6 版本。测试浏览器分别使用谷歌浏览器、火狐浏览器和 IE 浏览器。由于篇幅关系，这里只列出了部分功能模块的测试内容。

前端页面用户注册模块测试如图 8.1 所示。



图 8.1

## 测试结果分析

按照功能模块的测试来看，几乎所有的测试用例都能通过，这也正面验证了系统的功能模块的可用性和完整性。在用户的输入这方面，本系统默认用户的所有输入都是不可信任的[13]，在此基础上本系统的前台和后台都进行了输入数据的合法性校验，防止用户进行非法操作。

本次的测试以黑盒测试的形式进行，并在不同浏览器上进行相同的测试。其中谷歌和火狐浏览器完美运行，而 IE 浏览器上几乎所有测试均失败，页面功能无法正常使用。经调试发现是由于系统在前端编码时采用了更高级的 ES6 （即 ECMAScript 6.0 ， ECMAScript 是 JavaScript 的正式名称）的方式，而目前低版本的浏览器上的 JavaScript 都是 ES5 的版本，所以无法提供支持，至此本系统的测试和结果分析结束[14]。

# 总结

本次毕业设计开发大约用时两个月，期间的实习中不断的学习新的知识，并在毕设上加以应用，整个系统开发完成之后，对于我自己是受益良多的，起码让我刚学习的Vue框架又再一次的去运用起来，使得我对Vue的熟悉程度更深，也对之后的工作上的项目有所帮助。当然，开发期间也是有一些问题的出现，比如说有一些样式改不了、后端服务器长时间不用自动关闭等，不过到了后面上网去搜索各种方案去解决这些小bug，这些问题也让我开发这个毕设的时间变长了，其实就是实践的经验太少了，bug遇到得太少了。现在遇到的bug修复了，对于我以后是好事，因为见过这个bug，知道怎么修复，那么开发的时间就会大大的缩短。

想要在互联网这个领域里面立足，就要不断的学习，现在学的可能也就只能维持个四五年的工作，如果后面不继续学，跟不上各种框架系统的更新换代，就只能被淘汰，学无止境，只有不断的学习，才能真正的立足于这个社会。

对于未来，希望自己不断的学习，巩固以前的知识，为之后在社会的立足中打下坚实的基础。

### 参考文献

1. 明日科技.JavaScript从入门到精通[M].北京:清华大学出版社,2020:50-55.
2. 张阳灿,林怡呈. 世界咖啡豆烘焙履历图鉴[M].福建: 福建科技出版社,2021.01:70-82.
3. 詹姆斯·霍夫曼.世界咖啡地图（新版）[M].北京:中信出版社.2020.06:23-45.
4. [郑冬松](http://search.dangdang.com/?key2=%D6%A3%B6%AC%CB%C9&medium=01&category_path=01.00.00.00.00.00),[王贤明](http://search.dangdang.com/?key2=%CD%F5%CF%CD%C3%F7&medium=01&category_path=01.00.00.00.00.00),[邓文华](http://search.dangdang.com/?key2=%B5%CB%CE%C4%BB%AA&medium=01&category_path=01.00.00.00.00.00),[吴宗大](http://search.dangdang.com/?key2=%CE%E2%D7%DA%B4%F3&medium=01&category_path=01.00.00.00.00.00).数据库应用技术教程（SQL Server 2017）-微课视频版[M].北京:[清华大学出版社](http://search.dangdang.com/?key3=%C7%E5%BB%AA%B4%F3%D1%A7%B3%F6%B0%E6%C9%E7&medium=01&category_path=01.00.00.00.00.00),2021.03:15-45.
5. 李代平,杨成义.软件工程(第四版)[M].北京:清华大学出版社,2017.10:17-133.
6. [明日科技](http://search.dangdang.com/?key2=%C3%F7%C8%D5%BF%C6%BC%BC&medium=01&category_path=01.00.00.00.00.00),HTML5从入门到精通[M].北京:[清华大学出版社](http://search.dangdang.com/?key3=%C7%E5%BB%AA%B4%F3%D1%A7%B3%F6%B0%E6%C9%E7&medium=01&category_path=01.00.00.00.00.00),2019.10:50-63.
7. [石毅](http://search.dangdang.com/?key2=%CA%AF%D2%E3&medium=01&category_path=01.00.00.00.00.00).动态网页设计与开发——JavaScript + jQuery[M].北京:[电子工业出版社](http://search.dangdang.com/?key3=%B5%E7%D7%D3%B9%A4%D2%B5%B3%F6%B0%E6%C9%E7&medium=01&category_path=01.00.00.00.00.00), 2020.09:23-68.
8. 陈丹丹,高飞.JSP项目开发全程实录[M].北京:清华大学出版社,2013.12:20-145.
9. [唐连生](http://search.dangdang.com/?key2=%CC%C6%C1%AC%C9%FA&medium=01&category_path=01.00.00.00.00.00).电子商务运营管理实务[M].北京:[中国财富出版社](http://search.dangdang.com/?key3=%D6%D0%B9%FA%B2%C6%B8%BB%B3%F6%B0%E6%C9%E7&medium=01&category_path=01.00.00.00.00.00),2020.06:36-58.
10. 传智播客高教产品研发部.HTML5+CSS3网站设计基础教程[M].北京:人民邮电出版社,2019.7:90-306.
11. 马克・艾伦・维斯.数据结构与算法分析 Java语言描述[M].北京:机械工业出版社,2016.03:15-68.
12. 谢雨飞,田启川.计算机网络与通信基础[M].北京:清华大学出版社,2019.08:62-74.
13. Joshua Bloch. Effective Java[M]. Piscataway,N.J:IEEE Press,2009:13-23.
14. Yang Ming-Hsien, Weng Sung-Shun, Hsiao Pei-I.Measuring blog service innovation in social media services[J].Internet Research,2014(5):100-138.

### 致 谢

时间飞逝，大学四年的时候就这样结束了，在此之后，我们也将各奔东西，踏上了自己的人生道路。在此，我感谢华南农业大学珠江学院对喔的栽培、感谢老师们不留余力的付出和教导、感谢同学和朋友们多年来的陪伴。往后的人生，祝你们前程似锦，事事顺利，马到成功！

在本次系统分析和开发的过程中，我遇到了很多疑惑，最后能圆满地解决，少不了赵志俊老师耐心的指导和同学们的鼎力相助。在此我很感谢他们，能在我遇到问题的时候给我拨开迷雾，指引方向。今后，我会更加地努力，坚守自己的初心，奋发前进！