# 1. 绪论

在信息技术高速迭代的时代背景下，人类的生活模式及消费行为经历了显著转变。在图书零售行业中，传统书店正遭遇租金不断攀升、经营成本提高、市场竞争激增等多重困境，与此相对照的，网络图书销售平台却展现出一种生机勃勃的发展趋势。相关数据显示，近年来全球在线图书销售额持续增长，越来越多的读者倾向于通过网络渠道购买图书。在线图书销售平台不仅为读者提供了更为丰富的图书选择，还能依据读者的阅读习惯和偏好，提供个性化的推荐服务，极大提升购书效率和体验。对于图书销售企业而言，在线销售平台有助于降低运营成本、扩大市场覆盖范围，进而提高销售业绩和竞争力。

## 1.1 研究背景介绍

在国民整体素养持续攀升的情境中，图书消费需求展现出多样化的趋势。众多国内外网络平台，诸如亚马逊、京东以及当当网，纷纷在在线图书零售领域取得了显著成就。中小型图书销售企业在构建及营运高效稳定的网络销售平台方面，遭遇着显著的挑战。本研究致力于构建一套集成度高、操作简易且便于管理的网络图书销售平台，意在迎合市场发展需求，从而增加了企业市场竞争能力。

1.2 研究现状

近几年，由于全球经济下行，购买力下降，网购以价格低廉便捷的优势成为大家的首选。调查发现，尽管网上存在诸多图书交易平台，且部分已取得显著成就，但仍存在缺陷和漏洞。有些平台用户界面与大众所需不吻合，操作严重影响用户的购物体验，传统库存管理系统难以应对大规模的图书种类和频繁的库存变动，容易导致库存积压或缺货现象。

## 1.3 研究目的与意义

网络购物已经进入了日常生活，成为不可替代的存在。在此背景下，设计并实现一个功能完备、高效稳定的在线图书销售系统具有重要意义。该系统旨在为用户提供友好的界面和便捷的操作流程，优化购物体验，为商家提供高效的库存管理和订单处理机制，减少库存积压和缺货情况。此外，在线图书销售系统还能推动传统图书销售行业的数字化转型，打破地域限制，促进文化传播，提升整个行业的运营效率和服务质量。

## 1.4 本章小结

本章节综合论述了课题的研究背景、现状及关键内容，通过充分的调研工作，为在线图书销售管理平台的设计奠定了坚实基础，确保课题能够顺利推进。

# 2.相关技术简介

## 2.1 Nodejs介绍

Nodejs代表着一种高效的Web服务器解决方案，其设计旨在实现卓越的服务性能。通过Nodejs，用户能够在各种网络浏览器环境中，运用JavaScript语言进行编程与交互。Nodejs的核心应用领域涵盖后台管理平台、即时通信系统、单页应用软件，以及处理高并发请求的Web应用程序等多个范畴。该系统具备直接对数据库执行操作的能力，同时，利用应用程序编程接口，实现数据至前端及移动端的有效传输。将前端浏览器中普遍运用的事件驱动机制借鉴至后端开发环境中，Nodejs结合异步输入/输出操作，实现了将事件传递至业务逻辑层进行高效处理的架构模式。针对构建网站服务及应用程序的快速开发需求，Nodejs凭借其简洁性及可扩展性，成为处理高并发请求、I/O密集型任务以及业务逻辑简化情形下的理想选择。

## 2.2 Vue介绍

Vue是一套用于搭建界面的渐进式框架，可部分使用。其核心库专注于视图层，是一个独立的功能模块，既可以单独使用声明渲染，也能通过自有的组件系统方便地与第三方库进行整合，在驱动复杂的单页应用时表现出色。Vue的设计十分灵活，无论是PC端还是移动端都能直接使用，项目开发者通过其便捷的集成路径，逐步集成到项目中。Vue具有轻量特性，能够自动追踪依赖的模板表达式和计算属性，拥有灵活的API，便于开发人员理解和上手。

## 2.3 MySQL介绍

MySQL在数据库管理系统的范畴内，MySQL以低成本、高速运行及较小体积的显著特征而著称。此外，MySQL作为一款开源软件，其在降低整体应用成本方面发挥了显著作用。不同数据集在MySQL中得以分配至各个独立的表格，此举不仅显著提升了数据处理效率，亦赋予数据管理以更高的弹性。此外，该数据库系统允许在单一查询语句中集成来源于不同数据库的表格数据。通过对SQL查询算法的深度优化，MySQL数据库管理系统在数据检索速度上得到显著提升，同时亦促进了存储效率的优化进程。在本论章内，对构建在线图书销售体系所涉及的技术理论进行了细致的解析，旨在为具体开发过程奠定坚实的理论基石，从而保障在线图书管理系统的开发流程得以顺畅实施。

## 2.4 本章小结

本章仔细阐述了在线图书销售系统用到的技术理论，主要目的是为实际开发提供坚实的理论基础，为在线图书销售系统的开发保驾护航。

# 系统需求分析

## 3.1 需求调研

为了准确理解用户需求，我们进行了实际购买图书体验加上广泛的市场调研。调研对象包括普通用户、图书管理员和企业管理人员。通过调研发现，在线图书销售平台的主要需求集中在用户体验、库存管理、个性化推荐等方面，用户希望界面友好、可操作性能好、可快速下单实现购买且能根据其搜索记录和购买行为分析提供个性化的图书推荐。图书管理员需要实现高效便捷的库存管理和订单处理机制，帮助减轻库存负担。

## 3.2 需求分析

根据需求调研结果，该在线图书销售管理平台，旨在为用户与管理员提供全方位的信息服务。我们将系统的实际使用角色分为管理员和用户。在线图书销售系统的功能结构图如图3-1。

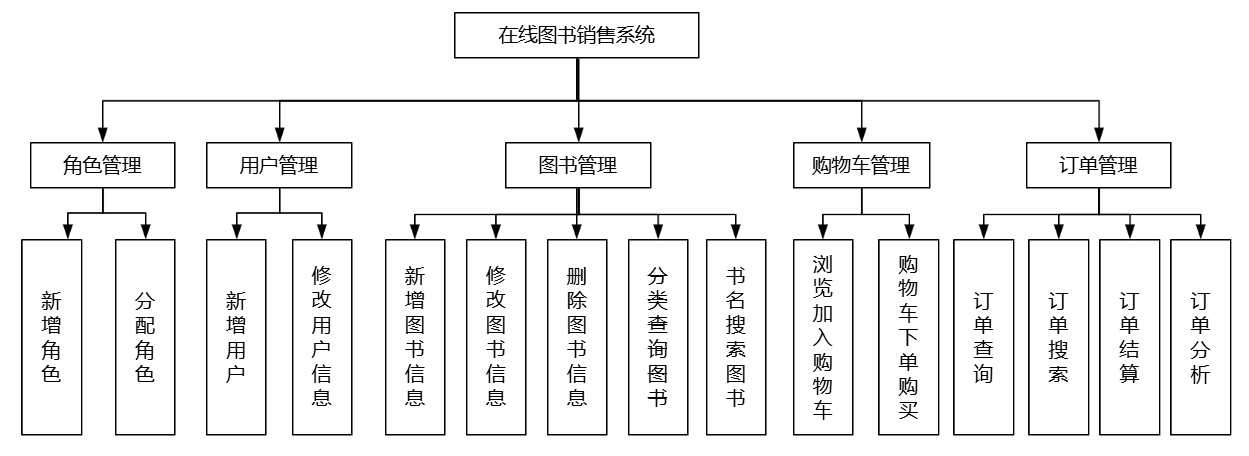


图3-1 在线图书销售系统功能结构图

### 3.2.1 普通用户需求

依据系统结构图分析可知，普通用户具备登录注册的基础需求，进而实现线上图书浏览、搜索、分类查询、展示以及提交并生成订单以完成交易等一系列操作。

(1)在线浏览图书，进入系统后，提供按书名搜索，支持切换类别浏览，可切换首页的Tab（小说、童书、计算机网络、文化等）进行查看。展示图书列表，包含图书封面、书名、作者、价格等基本信息。

(2)图书详情查看，包括书名、作者、出版社、出版时间、简介等内容展示。在图书详情页显示库存数量，可以在详情页直接“加入购物车”或者“立即购买”。

(3)购物车，用户可将图书添加到购物车，在购物车可修改图书购买数量，删除购物车商品，能实时计算购物车商品总价，显示商品数量和总金额。可勾选购物车的部分商品下单结算。

(4)订单查看，用户下单时生成的订单可以在订单列表查看。下单成功后生成订单编号，显示订单详情（包括订单编号、下单时间、商品信息、下单数量、支付金额等）。用户可查询订单状态，对于未支付的订单可以点击“去付款”进行支付。

### 3.2.2 管理员需求

根据系统结构图分析，管理员需要实现高效管理图书及用户信息，确保图书管理工作的有序开展，实现图书流通环节的高效运作与精准把控。

(1)角色信息管理。新增修改分配系统的角色信息。

(2)用户信息管理。查看修改用户角色、邮箱、联系方式，联系地址等详细信息。

(3)图书管理。记录图书入库信息，包括入库时间、入库数量等，新增图书信息，包括书名、作者、出版社、出版时间、内容简介、图片、图书类别、价格、库存数量等详细信息。编辑图书信息，可对已上架图书的各项信息进行修改。删除图书信息，可将下架或不再销售的图书从系统中删除，删除操作需谨慎确认。库存盘点，定期对库存进行盘点，确保实际与在售一致。

(4)订单管理。具体包括订单编号、下单时间、用户信息、商品信息、订单状态等，可根据订单状态进行操作。支持订单编号、订单状态等条件搜索查询。订单统计分析，统计不同时间段的订单数量、销售额、客单价等数据，生成订单统计报表。

3.3 本章小结

本章通过实际购买流程体验和市场调研的方式，分析了不同角色的需求。对于普通用户，我们明确了从登录注册到在线浏览、搜索图书，再到分类检索，最终提交订单完成交易的需求。对于管理员而言，需求集中在高效管理图书信息和用户数据上，包括但不限于维护角色用户信息、新增编辑图书、库存管理和订单处理等方面。基于以上的调研结果，再次细化各个功能模块的设计。通过这一系列步骤，我们确保所构建的在线图书销售管理系统完全满足用户的需求。

# 系统设计

## 4.1 设计原则

从系统整体流程和规划来看，需要构建具有高安全性和高实用性的目标系统。整体设计遵循以下原则：注重实用性能高，从实际出发，以用户需求为导向设计系统流程和功能，确保操作流畅，提升用户体验。强调扩展性强，鉴于技术的不断更新迭代，遵循业务差异预留标准化API接口，为系统升级和与其他系统集成提供空间。利用现有的技术栈，开发现有资源，将先进技术应用于系统，缩减开发成本。

## 4.2 数据库设计

数据库设计挑起了整个项目开发的脊梁。数据库设计包括前期的需求分析、中期的概念模型设计、后期的数据库建表三大过程。需求分析就是将用户需求转换系统需求分析的过程，概念模型设计包括概念模型和逻辑结构设计相结合，最后建立合适的数据库表导入。

### 4.2.1 数据库概念设计

通过前期需求分析后，进入系统数据库的概念模型设计阶段。概念模型设计主要是用户表达需求观点的数据全局的逻辑结构的模型，设计概念模型的主要目的是为了让用户能够表达他们的需求和观点，我们从用户的角度理解，建模项目所需数据。

在线图书销售系统中可能包含用户（普通用户和管理员）、购物车、订单、书籍、角色等实体，User包括普通用户和管理员用户,Cart购物车信息，Order订单信息，Book图书信息，Role用户角色信息。User和Role是一对多关系，单个用户仅有一个角色，但单个角色可分配多个用户。User和Cart是一对多关系，单用户可以有多个购物车项，但每项只属于一个用户。Order和User是一对多关系，单用户可以下多订单，但每个订单仅由一位用户下单。Order和Book是多对多关系，单笔订单可下单多本书籍，多个订单可以购买同一种图书。在线图书系统的全局E-R模型如图4-1。

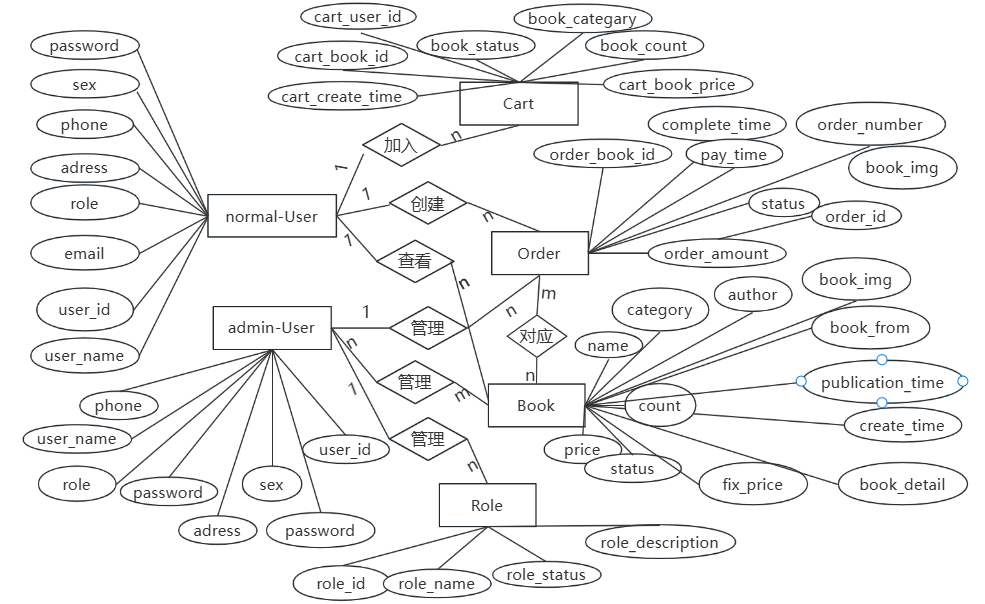


图4-1 数据库E-R模型图

### 4.2.2 数据库表详细设计

由于数据库中的数据庞大又复杂，因此设计数据库时应特别注意，表的数量能少则少，表中的属性尽量一目了然，数量也不宜过多，各个表之间的相互联系尽量少但数据关系必须要紧凑，可高效保证数据的准确性。因此本系统应该由多个表组成：用户(User)信息表，角色(Role)信息表，订单(Order)信息表，图书(Book)信息表，购物车(Cart)表。

表4-1 角色(Role)信息表

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **字段名** | **数据类型** | **长度** | **主键** | **能否为空** | **说明** |
| id | bigint | 20 | 是 | 否 | id |
| role\_id | varchar | 64 | 是 | 否 | 角色编号 |
| role\_name | varchar | 64 | 否 | 否 | 角色名称 |
| role\_status | varchar | 64 | 否 | 否 | 角色状态 |
| role\_description | varchar | 64 | 否 | 否 | 角色描述 |
| create\_time | datetime | 0 | 否 | 否 | 创建时间 |
| modify\_time | timestamp | 0 | 否 | 否 | 更新时间 |

表4-2 用户(User)信息表

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **字段名** | **数据类型** | **长度** | **主键** | **能否为空** | **说明** |
| id | bigint | 20 | 是 | 否 | id |
| user\_id | varchar | 20 | 是 | 否 | 用户id |
| user\_name | varchar | 64 | 否 | 否 | 用户姓名 |
| password | varchar | 64 | 否 | 否 | 密码 |
| sex | varchar | 64 | 否 | 否 | 性别 |
| role | varchar | 64 | 否 | 否 | 角色 |
| email | varchar | 64 | 否 | 否 | 邮箱 |
| phone | varchar | 64 | 否 | 否 | 手机号码 |
| address | varchar | 128 | 否 | 否 | 地址 |
| create\_time | datetime | 0 | 否 | 否 | 创建时间 |
| modify\_time | timestamp | 0 | 否 | 否 | 更新时间 |

表4-3 图书(Book)信息表

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **字段名** | **数据类型** | **长度** | **主键** | **能否为空** | **说明** |
| id | int | 11 | 是 | 否 | id |
| book\_name | varchar | 64 | 否 | 否 | 图书名称 |
| category | varchar | 64 | 否 | 否 | 图书类别 |
| book\_detail | varchar | 256 | 否 | 否 | 图书详细介绍 |
| book\_from | varchar | 64 | 否 | 否 | 出版社 |
| author | varchar | 64 | 否 | 否 | 作者 |
| book\_img | varchar | 512 | 否 | 否 | 图书图片 |
| book\_count | int | 64 | 否 | 否 | 库存数量 |
| book\_id | varchar | 64 | 是 | 否 | 图书序列号 |
| price | varchar | 64 | 否 | 否 | 在售价格 |
| fix\_price | varchar | 64 | 否 | 否 | 指导价格 |
| status | varchar | 64 | 否 | 否 | 图书状态 |
| publication\_time | varchar | 0 | 否 | 否 | 出版时间 |
| create\_time | datetime | 0 | 否 | 否 | 创建时间 |
| modify\_time | timestamp | 0 | 否 | 否 | 更新时间 |

表4-4 订单(Order)信息表

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **字段名** | **数据类型** | **长度** | **主键** | **能否为空** | **说明** |
| id | int | 20 | 是 | 否 | id |
| order\_id | int | 20 | 是 | 否 | 订单编号 |
| order\_amount | varchar | 64 | 否 | 否 | 付款金额 |
| order\_book\_count | Int | 20 | 否 | 否 | 下单数量 |
| order\_status | varchar | 64 | 否 | 否 | 订单状态 |
| order\_book\_id | int | 20 | 否 | 否 | 订单图书id |
| order\_pay\_time | datetime | 0 | 否 | 否 | 订单付款时间 |
| order\_complete\_time | datetime | 0 | 否 | 否 | 订单完成时间 |
| create\_time | datetime | 0 | 否 | 否 | 订单创建时间 |
| modify\_time | timestamp | 0 | 否 | 否 | 更新时间 |

表4-5 购物车(Cart)表

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **字段名** | **数据类型** | **长度** | **主键** | **能否为空** | **说明** |
| id | int | 20 | 是 | 否 | Id |
| book\_categary | int | 20 | 是 | 否 | 编号 |
| cart\_user\_id | varchar | 64 | 否 | 否 | 用户Id |
| cart\_book\_id | varchar | 64 | 否 | 否 | 图书id |
| cart\_book\_count | varchar | 64 | 否 | 否 | 加购数量 |
| cart\_book\_price | varchar | 64 | 否 | 否 | 在售价格 |
| cart\_book\_status | varchar | 64 | 否 | 否 | 是否结算 |
| cart\_create\_time | datetime | 0 | 否 | 否 | 加购时间 |
| modify\_time | timestamp | 0 | 否 | 否 | 更新时间 |

## 4.3 系统流程设计

流程设计是将用户需求转化为系统需求的过程，通过开发语言和设计代码实现用户的想法。本系统进行了完整的流程管理设计，包括用户登录注册、角色管理、用户管理、订单管理、图书管理、以及用户进行在线图书浏览、分类或书名查找图书、下单购买等功能。

### 4.3.1 登录注册流程设计

系统都需要用户进行登录，首次登录用户需要进行注册，注册完并登录成功后系统会自动判断是管理员还是用户，不同的角色看到不同的页面。如图4-2。

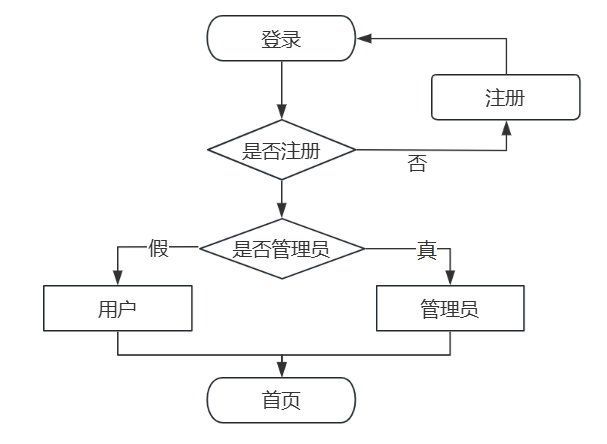


图4-2 登录注册流程图

### 4.3.2 角色信息管理流程设计

管理员进入系统后可以看到角色管理页面，也可以对角色进行维护。具体如图4-3。

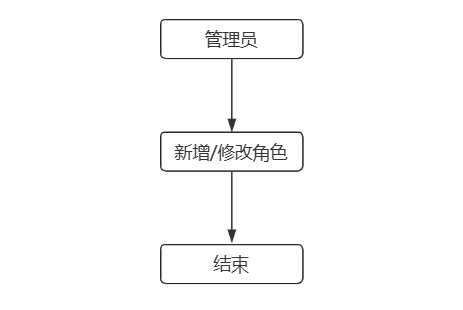


图4-3 角色信息管理流程图

### 4.4.3 用户信息管理流程设计

管理员进入首页可以看到用户管理页面，也可以对用户的信息进行修改，如图4-4。

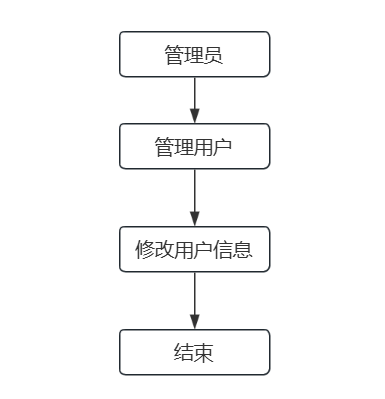


图4-4用户信息管理流程图

### 4.3.4 在线图书管理流程设计

用户和管理员都可以看到图书的信息，在首页可以看到推荐的图书信息，进入图书信息管理列表页管理员可以搜索新增修改删除图书，用户只能搜索查看图书的基本信息，如图4-5。

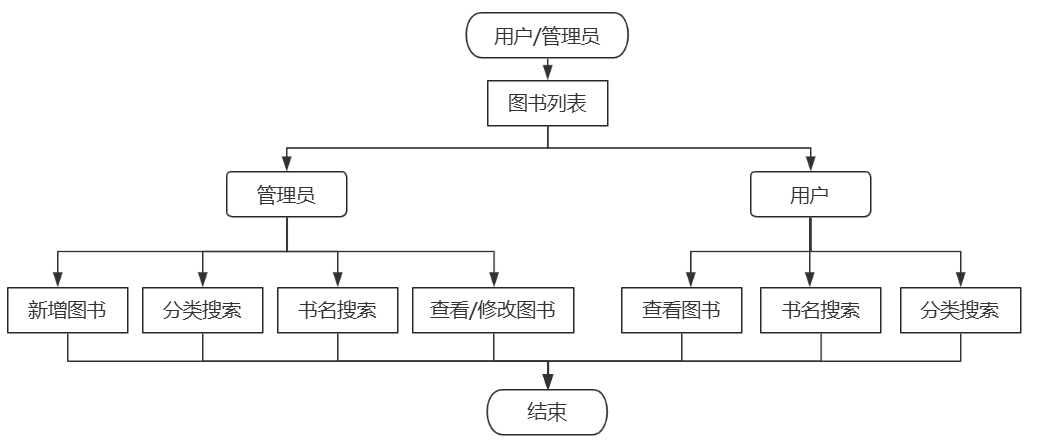


图4-5 图书管理流程图

### 4.3.5 在线下单购买流程设计

用户在系统中浏览图书后可以直接下单，也可以通过购物车下单两种方式购买图书，购买成功后可以在订单列表查看订单信息。如图4-6。

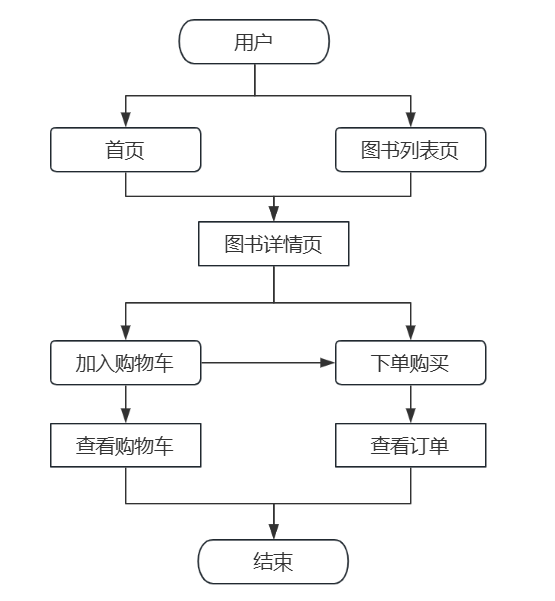


图4-6下单购买流程图

### 4.3.6 订单信息管理流程设计

管理员可以看到所有的订单信息，用户只能看到自己的下单信息（根据当前用户id过滤订单数据），未付款的订单可以点击去付款进行结算。如图4-7。

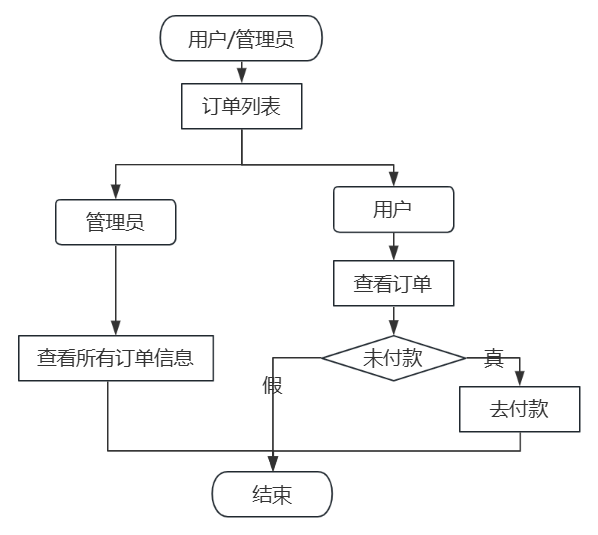


图4-7 订单信息管理流程图

4.4 本章小结

本章主要介绍了系统的设计原则，并对在线图书销售系统的流程设计和数据库设计进行了详细的分析和表述。首先，我们详细阐述了设计系统时所依据的基本原则，确保系统不仅能够满足当前的需求，还能适应未来的发展和变化。接着，我们深入探讨了数据库的设计，包括概念设计和表详细设计，为系统的数据存储和管理提供了坚实的架构基础，最后通过流程设计工作，我们不仅细化了各个功能模块的具体实现方式，也为后续的开发实施制定了清晰的蓝图。

# 5. 系统实现

通过数据库设计和业务流程设计，确定了在线图书销售系统的主要功能，根据以上的流程设计图我们实现了系统的主要功能。

## 5.1 注册登陆

在系统中购买下单的用户需进行注册登录，登录后方可正常下单购买。注册登录功能应确保用户能够顺利注册和登录，注册登录后的个人信息可进行修改，管理员也可对用户信息进行管理。用户登录注册时，系统会调用vue的表单验证功能进行提交前的数据验证，用户必须填写用户名和密码才能重新登录。新用户注册成功后，登录提交验证通过，后台数据库会根据用户id判断用户角色，然后通过vue的路由管理，使不同角色的用户看到不同的页面详细信息。登录注册页面效果如图5-1。



图5-1 登录注册页面效果图

登录注册核心代码如下：

|  |
| --- |
| <template>   <template>    <div class="login-container">      <el-form ref="loginForm" :model="loginForm" :rules="loginRules" class="login-form" autocomplete="on" label-position="left">        <div class="title-container">          <h3 class="title">图书销售系统</h3>        </div>        <el-form-item prop="username">          <span class="svg-container">            <svg-icon icon-class="user" />          </span>          <el-input            ref="username"            v-model="loginForm.username"            placeholder="用户名"            name="username"            type="text"            tabindex="1"            autocomplete="on"          />        </el-form-item>        <el-tooltip v-model="capsTooltip" content="Caps lock is On" placement="right" manual>          <el-form-item prop="password">            <span class="svg-container">              <svg-icon icon-class="password" />            </span>            <el-input              :key="passwordType"              ref="password"              v-model="loginForm.password"              :type="passwordType"              placeholder="密码"              name="password"              tabindex="2"              autocomplete="on"              @keyup.native="checkCapslock"              @blur="capsTooltip = false"              @keyup.enter.native="handleLogin"            />            <span class="show-pwd" @click="showPwd">              <svg-icon :icon-class="passwordType === 'password' ? 'eye' : 'eye-open'" />            </span>          </el-form-item>        </el-tooltip>        <el-button :loading="loading" type="primary" @click.native.prevent="handleLogin">          登录        </el-button>      </el-form>    </div>  </template>  // 后端代码  async login() {      const { ctx, service } = this      const { STATUS } = ctx      const postParam = ctx.params.permit('username', 'password')      const { user, msg, logined } = await service.user.login({        username,        password,      })      const rolesInfo = await service.user.findRolesByUserpkey(pkey)      if (isSave !== 'OK') {        ctx.status = STATUS.FORBIDDEN        ctx.body = {          code: ctx.ERROR,          message: '登录失败',        }        return      }      ctx.body = {        code: ctx.SUCCESS\_CODE,        message: '登录成功',        data: { token },      }    } |

## 5.2 角色管理

仅管理员可查看角色管理。需要使用vue的router控制，在路由配置里面设置role为管理员的角色。管理员需要对系统中所有的用户分配角色，也可以对角色进行增删等操作。角色列表页效果如图5-2。



图5-2 角色列表页效果图

角色列表页核心代码如下：

|  |
| --- |
| <template>    <div class="app-container">      <el-button type="primary" @click="handleAddCar">新增</el-button>      <el-table :data="roleList" style="width: 100%;margin-top:30px;" border>        <el-table-column align="center" label="角色编号">          <template slot-scope="scope">            {{ scope.row.role\_id }}          </template>        </el-table-column>  ...        <el-table-column align="center" label="操作" width="300">          <template slot-scope="scope">            <el-button type="primary" size="small" @click="handleEdit(scope)">编辑</el-button>            <el-button type="danger" v-if="scope.row.role\_status === '停用'" size="small" @click="handleDelete(scope)">删除</el-button>          </template>        </el-table-column>      </el-table>      <el-dialog        :title="dialogType==='edit'?'修改角色信息':'新增角色信息'"        :visible="dialogFormVisible">        <el-form :model="formUser">          <el-form-item label="角色编号">            <el-input v-model="formUser.role\_id" />          </el-form-item>  ...        </el-form>        <div slot="footer" class="dialog-footer">          <el-button @click="dialogFormVisibleCancelClick(formUser)">取 消</el-button>          <el-button type="primary" @click="dialogFormVisibleSureClick(formUser)">确 定</el-button>        </div>      </el-dialog>    </div>  </template>  // 后端代码  class RoleController extends Controller {      async index() {          const { model, service, helper } = this.ctx          const result = await service.role.search({ ...postParam })      async save() {          this.ctx.logger.info('Start create, update and delete roles')          const postParam = ctx.params.permit('insertRecords', 'removeRecords', 'updateRecords')          let resultCreated = {}          if (postParam.insertRecords && postParam.insertRecords.length > 0) {              resultCreated = await service.role.createBulk({ ...postParam }.insertRecords)          }          else {              resultCreated = { code, data, message }          }          let result = await service.role.saveBulk({ ...postParam });          this.ctx.body = result;          this.ctx.logger.info(`End of create, update and delete roles${postParam}`)      }  } |

## 5.3 用户信息管理

用户信息管理需要在路由配置里面设置role为管理员的角色。管理员可以在用户管理页面进行用户信息的修改，查看用户的id，会员名称，用户电话，性别，角色，手机号码，注册时间，地址，也可以进行删除当前用户的操作。用户列表页实现效果如图5-3。



图5-3 用户列表页效果图

用户信息管理能进行用户信息修改的操作，例如修改用户的邮箱信息。修改用户邮箱效果如图5-4。

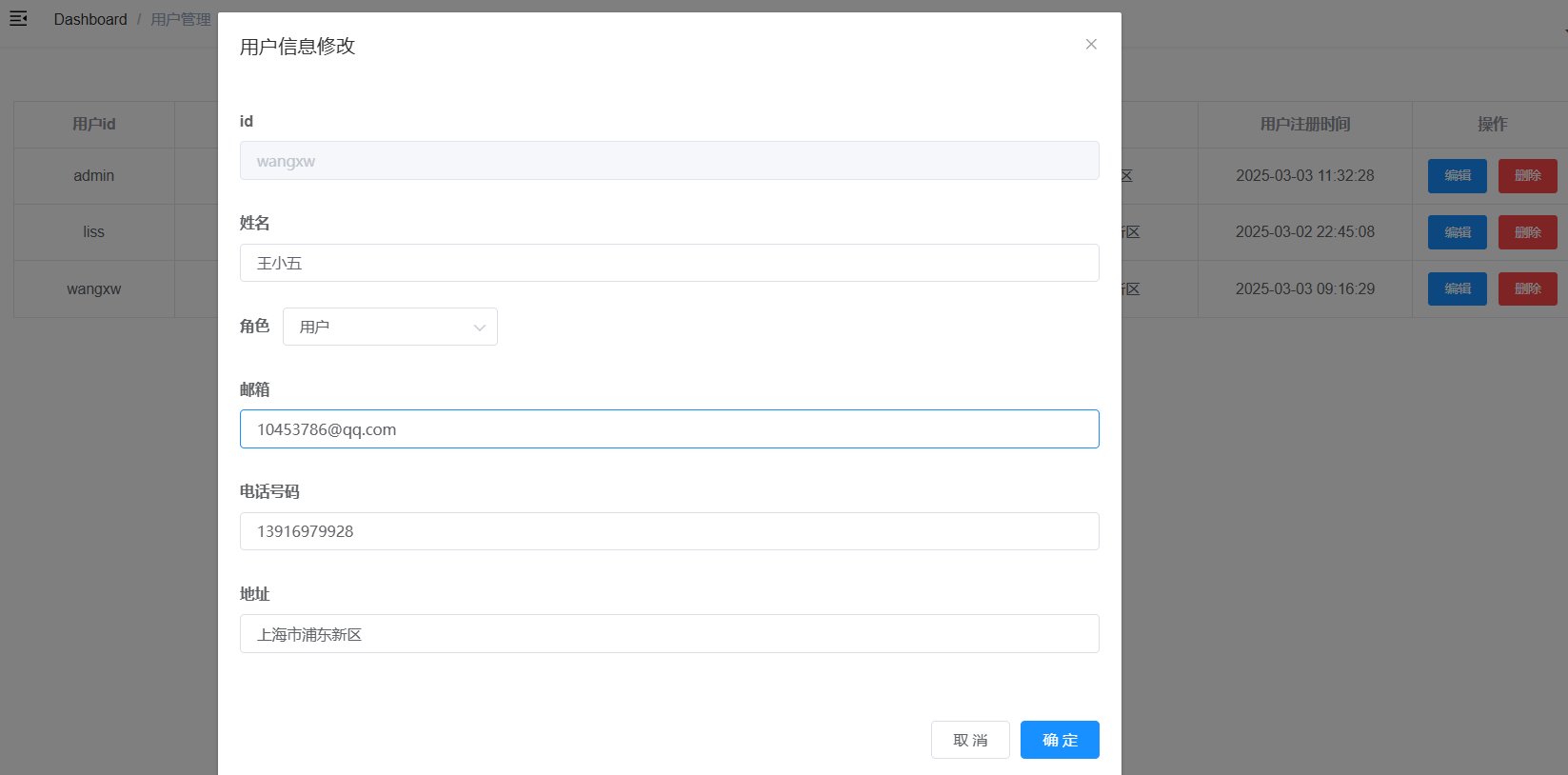


图5-4 修改用户邮箱信息界面图

用户列表页核心代码如下：

|  |
| --- |
| <template>    <div class="app-container">      <el-table :data="userList" style="width: 100%;margin-top:30px;" border>        <el-table-column align="center" label="用户id" width="150">          <template slot-scope="scope">            {{ scope.row.id }}          </template>        </el-table-column>        <el-table-column align="center" label="用户名称" width="150">          <template slot-scope="scope">            {{ scope.row.name }}          </template>        </el-table-column>  ...      </el-table>      <el-dialog title="用户信息修改" :visible="dialogFormVisible">        <el-form :model="formUser">          <el-form-item label="id">            <el-input v-model="formUser.id" disabled />          </el-form-item>          ...        </el-form>        <div slot="footer" class="dialog-footer">          <el-button @click="dialogFormVisibleCancelClick(formUser)">取 消</el-button>          <el-button type="primary" @click="dialogFormVisibleSureClick(formUser)">确 定</el-button>        </div>      </el-dialog>    </div>  </template>  // 后端代码  class UserController extends Controller {      async index() {          const { ctx, service } = this          ctx.logger.info('Start to search user );          errors && ctx.logger.info('errors from validation from user index:', errors)          const result = await service.user.search({ ...params })          ctx.logger.info(`End of search user${params}`)      }  } |

## 5.4 管理员在线图书管理

图书信息管理根据管理员和用户拆分不同的需求。管理员需要对图书进行库存管理，每当有图书需要入库或者出库的时候，可以及时进行图书库存信息的修改，月底或者年底需要进行盘库，系统需要实现线上库存与实际库存一致。主要功能需求有图书的基本信息的录入，修改，对已经入库图书信息进行分类修改，数量整理等。用户要对图书进行分类查询，按书名查询，查看所有的图书列表，点击查看图书详细信息。

因此，图书管理信息是一个列表信息页面，包含新增按钮，按分类查询图书，按书名查询图书的功能，列表主要信息包含图书姓名，价格，图片，图书类型，作者，出版社，图书状态，图书库存，入库时间，编辑图书信息，删除等操作。每次有图书上新或者下架的时候都可以在此进行图书修改的操作。点击图书名称可以跳转图书详情页面，每本图书都可以进行编辑操作，方便管理员对图书信息进行修改，删除实际库存中已经没有的图书。图书列表页实现效果如图5-5。



图5-5图书信息列表页效果图

当图书需要入库时，点击“新增”按钮，可以实现新增图书信息的操作。新增图书效果图如5-6。

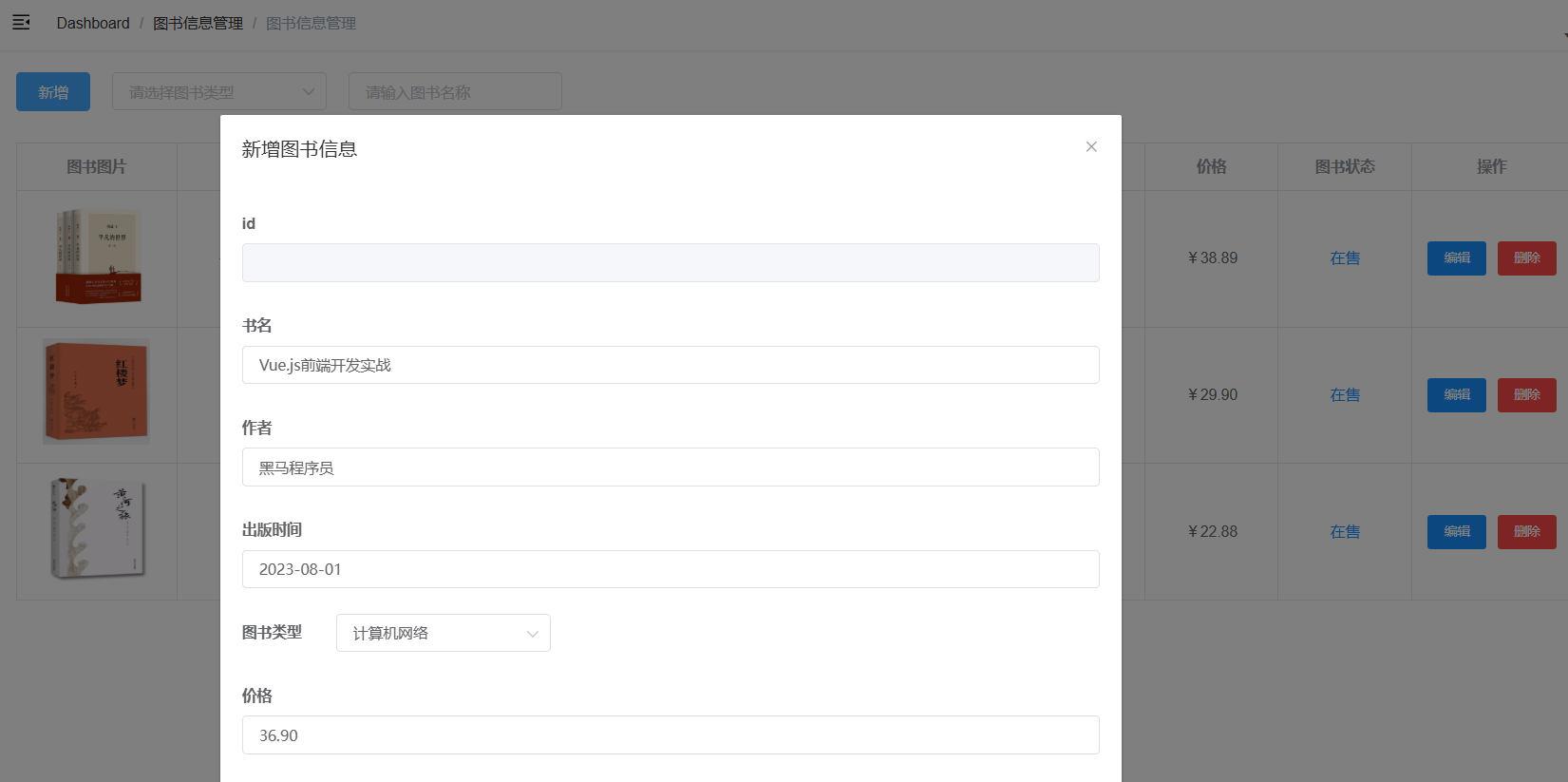


图5-6 新增图书界面图

管理员需要对新入库的图书信息进行修改，点击“编辑”按钮进入图书修改页面。修改图书信息图如5-7。



图5-7 修改图书信息界面图

盘库后，管理员需要对实际不存在的图书信息删除，点击“删除”按钮删除实际已不存在的图书。删除图书信息图如5-8。

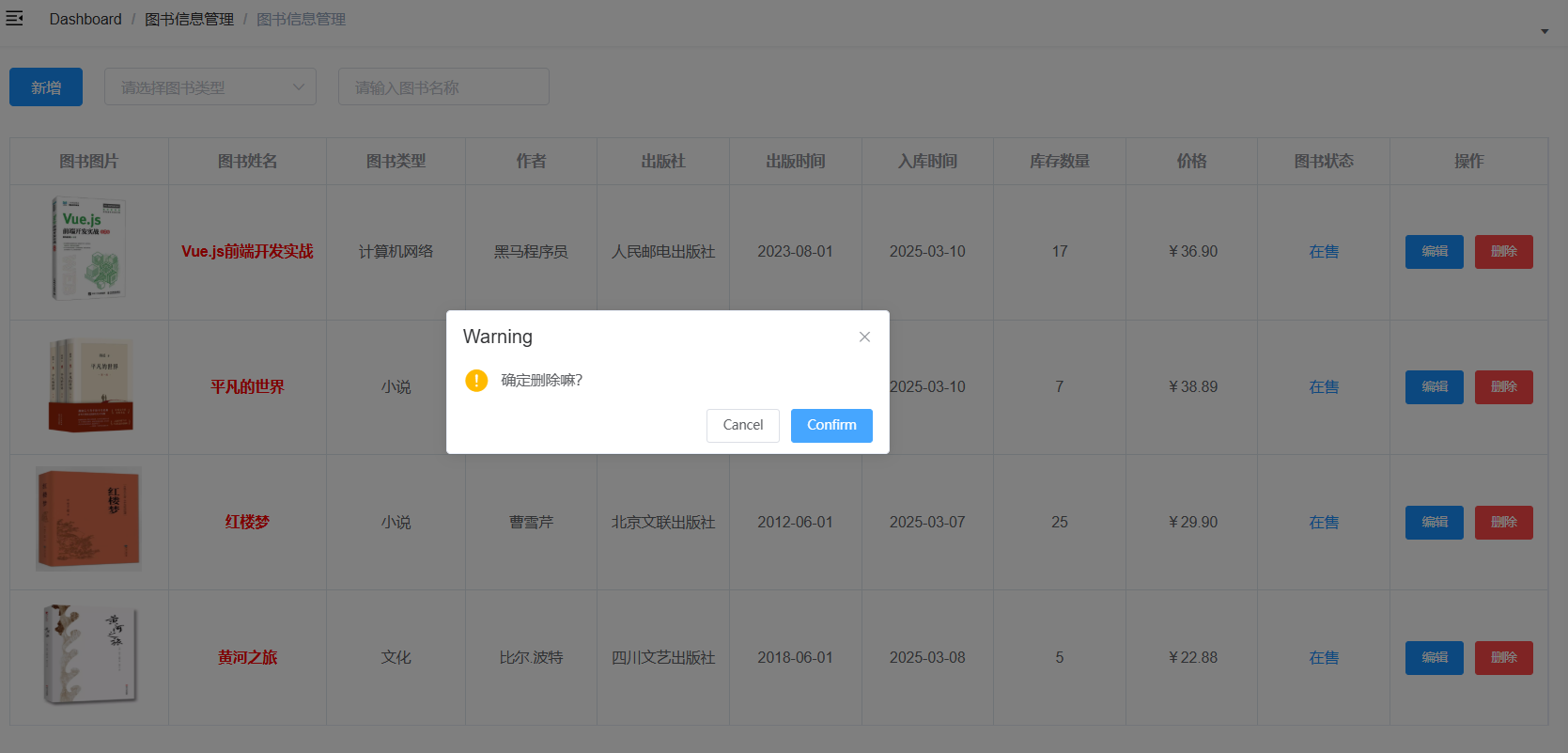


图5-8 修改图书信息界面图

由于图书列表显示图书信息较多，查询时可以按照分类查询图书信息。按分类查询图书效果图如5-9。



图5-9 按分类查询图书界面图

由于图书列表显示图书信息较多，可以按照图书名称查找图书信息。按名称查询图书效果图如5-10。

图5-10 按名称查询图书界面图

图书信息列表核心代码如下：

|  |
| --- |
| <template>    <div class="app-container">      <el-button type="primary" @click="handleAddBook">新增</el-button>  <el-select        v-model="categary"        placeholder="请选择图书类型"        class="left-input"        clearable        @change="selectChange"  >        <el-option          v-for="item in options"          :key="item.value"          :label="item.label"          :value="item.value"        />      </el-select>      <el-input v-model="bookName" placeholder="请输入图书名称" class="left-input" style="width: 200px;" @keyup.enter.native="handleFilter" />      <el-table :data="bookList" style="width: 100%;margin-top:30px;" border>        <el-table-column align="center" label="图书图片" width="150">          <template slot-scope="scope">            <img style="width:100px; height: 100px;" :src="scope.row.imgs">          </template>        </el-table-column>        ...        <el-table-column align="center" label="操作" width="150">          <template slot-scope="scope">            <el-button type="primary" size="small" @click="handleEdit(scope)">编辑</el-button>            <el-button type="danger" size="small" @click="handleDelete(scope)">删除</el-button>          </template>        </el-table-column>      </el-table>      <el-dialog        :visible.sync="dialogFormVisible"        :title="dialogType==='edit'?'修改图书信息':'新增图书信息'"      >        <el-form :model="formBook">          <el-form-item label="id">            <el-input v-model="formBook.id" disabled />          </el-form-item>          <el-form-item label="书名">            <el-input v-model="formBook.name" />          </el-form-item>...        </el-form>      </el-dialog>    </div>  </template>  // 后端代码  class bookController extends Controller {      async index() {          ctx.logger.info('Start to search book')          const params = ctx.params.permit('bookName', 'categary')          const result = await service.book.search({ ...params })          ctx.body = {              code: ctx.SUCCESS\_CODE,              message: '查询成功',              data: result          };          ctx.logger.info(`End of search book${params}`)      }      async save() {          const { ctx, service } = this;          ctx.logger.info('Start to create, update and delete books')          const errors = ctx.validate(bookRule.saveRule, params)          errors && ctx.logger.info('errors from validation from book save:', errors)          const resultVal = await this.ctx.service.common.checkUniqueByKey('book', 'bookId', params)          if (!(resultVal?.isUnique === true)) {              this.ctx.throw(403, `ID不能重复: ${resultVal?.dupKeys}`)          }          let result = await service.book.saveBulk({ ...params })          ctx.logger.info(`End of create, update and delete books${params}`)      }  } |

## 5.5 用户在线浏览图书下单购买

在线浏览并下单购买图书可以通过两个不同的入口进行。用户可以是从网站首页在线浏览后选择感兴趣的图书进行下单购买。也可以通过图书列表页面进入，筛选条件查找图书后进入详情页下单。

### 5.5.1 首页浏览下单结算

网站主页进行各种各样的图书展示，默认显示当前用户的购买历史推荐，或当下点击率最高的图书。点击切换Tab可查看不同类型的图书。具体显示图书的展示图片，图书名称，图书价格，点击图书的名称或图片，实现效果如图5-11。

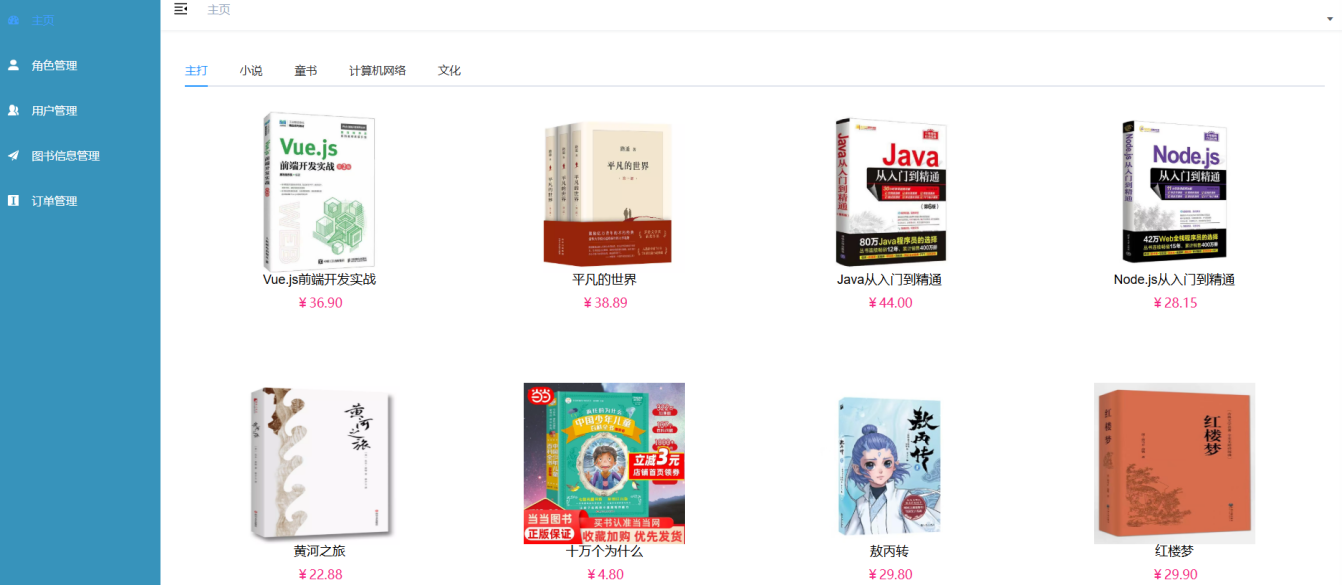


图5-11在线浏览图书页效果图

点击图片可以实现跳转到图书的详细页面，可以查看图书的详细内容，也可以直接在详情页面下单购买图书。实现效果如图5-12。



图5-12首页点击图书跳转详情页直接下单界面图

网站首页(在线浏览首页)核心代码如下：

|  |
| --- |
| <template>    <div class="dashboard-container">      <div class="dashboard-top">        <el-tabs          v-model="activeName"          class="dashboard-tabs"          @tab-click="handleClick"        >          <el-tab-pane label="主打" name="first" />          <el-tab-pane label="小说" name="second" />          <el-tab-pane label="童书" name="third" />          <el-tab-pane label="计算机网络" name="fourth" />          <el-tab-pane label="文化" name="five" />        </el-tabs>      </div>      <div class="dashboard-body">        <el-row :gutter="20">          <el-col v-for="(item, id) of filteredReports" :key="id" :span="6">            <div v-if="item.isShow" class="dashboard-list-tab" @click="imgClick(item.route)">              <img class="dashboard-list-img" :src="item.icon">              <div>{{ item.title }}</div>              <div class="dashboard-list-text">{{ item.price }}</div>            </div>          </el-col>        </el-row>      </div>    </div>  </template> |

### 5.5.2 列表页浏览下单结算

在图书列表页点击书名后金图书详情页面，主要有书名，价格，分类，出版社，库存数量，详细介绍等。可以修改数量后加入购物车或直接下单结算，购物车数量限制修改不能小于1，当修改后的数量超过库存数量系统会自动提示加购数量不能超过实际库存。图书详情页实现效果图如图5-13。



图5-13 图书信息详情页效果图

图书信息详情页面核心代码如下：

|  |
| --- |
| <template>    <div class="dashboard-container">      <div class="dashboard-top">        <div class="dashboard-top-left-top">          图书详情        </div>      </div>      <div class="dashboard-body">        <img          class="dashboard-list-img"          :src="reports.img"        >        <div class="book-right">          <div class="dashboard-list-text" style="font-weight: bold; font-size: 24px;">{{ reports.title }}</div>          <div class="dashboard-list-text">            <div>单价:              <span style="color: red; font-weight: bold; font-size: 20px;">                ￥ {{ reports.price }}              </span>              <span class="price-css">定价:                <span style="text-decoration: line-through;">                  ￥ {{ reports.fix\_price }}                </span>              </span>            </div>          </div>          <div class="dashboard-list-text">分类：{{ reports.category }}</div>          <div class="dashboard-list-text">出版社：{{ reports.book\_from }}</div>          <div class="dashboard-list-text">出版时间：{{ reports.publication\_time }}</div>          <div class="dashboard-list-text">            <el-input              v-model="num"              oninput="value=value.replace(/[^\d]/g,'')"              autocomplete="off"              style="width:140px"              size="mini"              @change="handleInput(num)"            >              <el-button slot="prepend" size="mini" @click="del(num)">                <i class="el-icon-minus" />              </el-button>              <el-button slot="append" size="mini" @click="add(num)">                <i class="el-icon-plus" />              </el-button>            </el-input>          </div>          <div class="dashboard-list-text">            实际库存数量：            <span style="font-weight: bold;">              {{ reports.count}}            </span>          </div>          <div>            <el-button type="primary" size="small" @click="shop()">加入购物车</el-button>            <el-button type="warning" size="small" @click="settlement()">立即下单</el-button>          </div>        </div>      </div>      <div class="dashboard-title">        <h3>书籍主要内容介绍：</h3>        <div>{{ reports.body }}</div>      </div>    </div>  </template> |

进入详情页后能直接“加入购物车”，用户选择将有需求的图书加入购物车，购物车页面就会增加一条数据，购物车页面会调用vue的Ajax方法调用后台数据并通过CSS静态分页方式显示用户已经加入购物车的所有图书。如果用户选择在购物车页面下单购买，系统会根据购物车里面的图书信息，计算勾选的购买下单的图书信息，创建订单信息，清除已下单的购物车图书，同时调用后台接口减少库存信息，订单结算完成后系统会提示用户下单成功。购物车实现效果如图5-14。



图5-14 购物车效果图

点击购物车“去结算”按钮，可以生成订单和结算信息，购物车结算实现效果图如图5-15。



图5-15 购物车结算页效果图

购物车页面核心代码如下：

|  |
| --- |
| <template>    <div class="content shopCart">      <div class="breadcrumb">        <el-breadcrumb>          <el-breadcrumb-item :to="{ path: '/dashboard' }">产品</el-breadcrumb-item>          <el-breadcrumb-item>购物车</el-breadcrumb-item>        </el-breadcrumb>      </div>      <div class="table">        <el-table          ref="multipleTable"          style="width: 100%"          element-loading-text="正在为您拼命加载中..."          :data="tableData"          :header-cell-style="{background:'#f8f8f8',color:'#999'}"          :close-on-click-modal="false"          :close-on-press-escape="false"          @selection-change="handleSelectionChange"        >          <el-table-column type="selection" width="75" align="center" />          <el-table-column prop="shopImg" align="center" width="150" label="商品">            <template slot-scope="scope">              <img :src="scope.row.shopImg" class="shopImg" alt="">            </template>          </el-table-column>          <el-table-column prop="shop" align="center">            <template slot-scope="scope">              <span class="shop">{{scope.row.shop}}</span>            </template>          </el-table-column>          <el-table-column prop="price" label="单价" align="center">            <template slot-scope="scope">              <span class="price">¥{{scope.row.price}}</span>            </template>          </el-table-column>          <el-table-column prop="number" label="数量" align="center">            <template slot-scope="scope">              <el-input                v-model.number="scope.row.number"                oninput="value=value.replace(/[^\d]/g,'')"                autocomplete="off"                style="width:140px"                size="mini"                @change="handleInput(scope.row)"              >                <el-button slot="prepend" size="mini" @click="del(scope.row)">                  <i class="el-icon-minus" />                </el-button>                <el-button slot="append" size="mini" @click="add(scope.row)">                  <i class="el-icon-plus" />                </el-button>              </el-input>            </template>          </el-table-column>          <el-table-column prop="count" label="小计" align="center">            <template slot-scope="scope">              <span class="count">¥{{scope.row.goodTotal}}</span>            </template>          </el-table-column>          <el-table-column label="操作" align="center">            <template slot-scope="scope">              <i                class="el-icon-delete"                style="font-size:18pxcursor: pointer"                @click="handleDelete(scope.$index, scope.row)"              />            </template>          </el-table-column>        </el-table>        <div class="submit">          <p class="submitBar">            <span>继续购物</span>            <span>              共 <b> {{ tableData.length }} </b> 件商品,              已选择 <b> {{ multipleSelection.length }} </b> 件            </span>          </p>          <div class="submit-price submitBar">            <span style="background: #fff;">产品总额：{{ totalPrice }}</span>            <el-button              size="small"              :disabled="multipleSelection.length<=0"              button-color="#bfa548"              @click="settlement"            >              去结算            </el-button>          </div>        </div>      </div>    </div>  </template> |

## 5.6 订单管理

订单列表页主要进行查看订单的功能，在用户下单后，可以看到用户的购买支付等相关信息，如有异常，需要及时跟进处理。主要的功能需求有查看订单信息，在用户下单后，管理员需要核对下库存信息，查看库存是否发生变化。订单信息管理列表页，该页面分条显示已经购买的图书，显示图书的封面图片，图书姓名（点击书名可以跳转图书详情页），下单时间，下单数量，交易是否成功，实际付款金额等。管理员可以删除交易失败的订单。订单列表页实现效果图如图5-16。

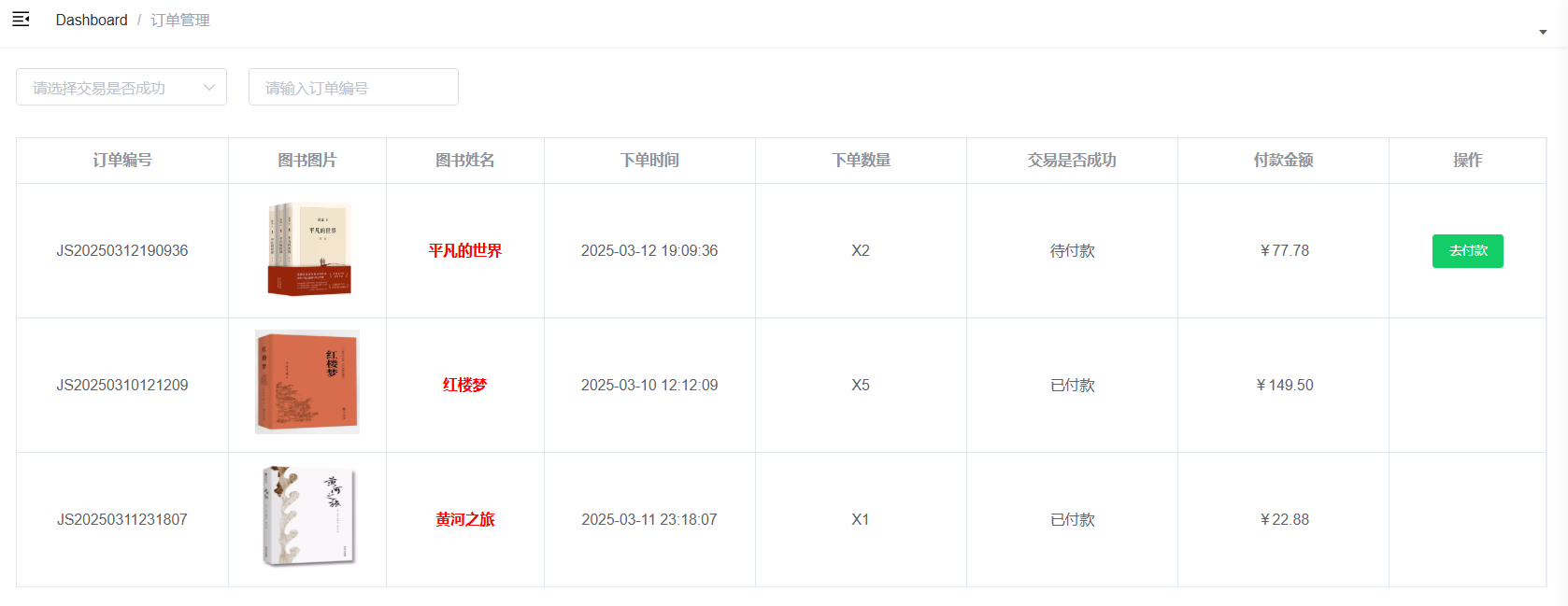


图5-16 订单信息列表页效果图

在订单管理列表页面，可以看到用户的下单购买信息，如果用户购买交易未成功，点击去付款可以直接去付款。结算成功会显示已付款，图书的库存信息随之减少。未付款订单直接去付款实现效果如图5-17。



图5-17 未付款订单直接去付款效果图

订单信息列表页核心代码如下：

|  |
| --- |
| <template>    <div class="app-container">      <el-table :data="orderList" style="width: 100%;margin-top:30px;" border>        <el-table-column align="center" label="订单编号">          <template slot-scope="scope">            {{ scope.row.order\_id }}          </template>        </el-table-column>        <el-table-column align="center" label="图书图片" width="150">          <template slot-scope="scope">            <img style="width:100px; height: 100px;" :src="scope.row.imgs">          </template>        </el-table-column>  ...      </el-table>    </div>  </template>  // 后端代码  class orderController extends Controller {      async index() {          const { ctx, service } = this          ctx.logger.info('Start to search order')          errors && ctx.logger.info('errors from validation from order index:', errors)          const result = await service.order.search({ ...params })          ctx.logger.info(`End of search order${params}`)      }  } |

## 5.7 本章小结

本章主要介绍了系统主要功能的实现，系统选择前后端分离的架构模式，前端基于Vue框架构建用户界面，确保了良好的用户体验,后端则使用nodejs提供API接口，保证了服务端的高效稳定性。详细介绍了在线图书销售系统的核心模块的实现细节，通过对各个关键模块的详细介绍和实现说明，证明实现的功能是可以满足设定的功能需求，且还能支持后续系统的优化和扩展。此外，注重用户体验和数据安全的设计理念贯穿于整个实现过程，确保了系统能够在实际应用中稳定运行，并为用户和管理员提供卓越的服务体验。

# 6. 系统测试

系统测试是确保软件能够正常运行的关键环节。随着软件开发技术复杂度的提升以及功能的日益丰富，软件存在漏洞的可能性也相应增加，这使得软件测试的重要性愈发凸显。

## 6.1 测试环境

搭建测试环境，首先需安装MySQL 5.7数据库完成安装后，将代码进行打包处理，在Windows11系统上配置环境变量，这一步骤是为了让系统能够正确识别和运行相关程序。之后，修改数据库的连接配置，确保系统与数据库之间能够建立稳定、准确的连接。最后，导入测试数据，这些数据用于模拟真实场景下系统的运行情况。准备工作完成后，发布测试地址，并在Chrome浏览器中打开该地址进行测试。在测试过程中，密切关注系统运行状态，及时发现并指出程序存在的 bug，随后进行修复，以确保系统能够稳定运行。

## 6.2 测试计划

依据需求分析的内容编写对应的测试用例，针对不同角色设置不同的测试流程。用户角色完整测试流程包括：检查用户是否可以正常创建账号，登录后是否能看到网站主页信息，是否可以将图书加入购物车、完成下单购买并实现交易功能。管理员角色的完整测试流程包括：确认管理员能否查看用户信息、图书信息和订单信息，每个页面的数据信息显示是否完整，能否进行用户信息修改、图书信息修改，以及订单信息是否完整显示等，如表6-1。

表6-1 功能测试计划表

|  |  |  |  |
| --- | --- | --- | --- |
| **测试功能** | **测试步骤** | **测试结果** | **备注** |
| 登录注册 | 点击登录-输入用户名和密码点击注册 | 正常登录到首页，看到首页推荐的内容 | 可以正常登录到首页，符合预期 |
| 点击注册-填写用户信息 | 注册成功，跳转首页正常登录 |
| 首页 | 进入首页后点击切换Tab（主打/小说/童书等） | 切换Tab显示为对应分类的图书 | 数据符合预期 |
| 用户信息 | 管理员登录-查看用户信息-修改用户信息 | 用户信息修改成功，刷新页面后显示的是修改后的信息 | 能正常修改数据符合预期 |
| 角色信息 | 管理员登录-查看角色信息-修改角色信息 | 提示角色信息修改成功，刷新页面后显示修改后的信息 | 数据符合预期 |
| 图书信息 | 管理员登录-图书信息列表页-新增图书信息 | 可以在列表看到新增加的图书信息 | 能正常修改数据符合预期 |
| 管理员登录-编辑图书信息 | 修改后图书信息显示为修改后的图书信息 | 正常修改数据符合预期 |
| 管理员登录-删除图书信息 | 删除后无法查到当前的图书信息 | 正常修改数据符合预期 |
| 管理员/登录-图书信息列表页-按书名查询图书 | 输入书名后仅显示当前书名相关的图书 | 正常查询数据符合预期 |
| 管理员/用户登录-图书信息列表页-按分类查询图书 | 选择分类后仅显示当前分类的图书 | 正常查询数据符合预期 |
| 下单购买 | 在线选购图书-加入购物车-提交订单-生成订单并付款 | 生成订单并在订单管理里面看到生成的订单信息 | 可以正常下单并看到生成订单信息并在结算完成后将成功结算状态写入订单测试结果符合预期 |
| 在线选购图书-立即购买-提交订单-生成订单并付款 | 生成订单并在订单管理里面看到生成的订单信息对未付款订单信息可以在订单信息列表页点击去付款进行结算 |
| 订单信息 | 管理员登录-查看订单信息 | 可以查询到所有的订单信息 | 数据符合预期 |
| 用户登录-查看订单信息-未付款订单去付款 | 仅查询自己所有的订单信息 | 数据符合预期 |

API测试。web应用程序不是独立存在的，它需要与服务器进行通信，具体包括数据的请求、响应，表单的提交、数据的验证等。在用户进行登录以后主页面的渲染通过接口将数据调出传给网站展示，例如图书页面展示10条数据，还需要检查数据库的数据是否跟展示数据相匹配。通过接口实现的一些增删改查，也是需要将接口数据和实际的数据进行对比，确保业务数据和实际数据库数据的一致。

验收测试。由用户使用实际的业务数据进行测试。在测试过程中，需要考虑软件的功能和性能是否与需求一致，兼容性是否在用户的接受范围内，可维护性成本是否过高，是否有一定的错误的恢复功能等。

## 6.3 测试结果

系统经过以上测试计划均可正常运行，所有的模块正常工作。测试过程中发现的一些小问题也都及时进行了改正，功能测试与实际业务需求相符合，各模块之间衔接流畅，可以实现数据交互。检测结果达到了预期。

## 6.4 本章小结

在本章中，我们详细介绍了在线图书销售管理系统的测试过程，包括测试环境的搭建、测试计划的设计与执行，以及最终测试结果的分析。通过系统化的测试流程，我们确保了该系统能够在实际应用中稳定、高效地运行，并满足用户和管理员的各项需求。本章的工作是项目成功不可或缺的一部分，它直接关系到系统能否顺利部署和广泛应用。

# 7. 总结

论文关注线上销售图书，于是设计实现了网上售书系统，功能齐全，操作方便。该系统不仅提升了用户的购书体验，还为企业提供了高效的管理工具，降低了运营成本，提高了市场竞争力。但其他方面仍存在不足之处，系统对于超大规模的用户访问性能上需要提高，复杂的业务逻辑处理不够完善；图书的分类不够丰富，需要继续扩充；还可以继续扩展数据库，引入更多是图书种类，完善用户的选择。后期也可以继续增加更多智能化功能，如语音搜索、虚拟现实（VR）书店等，以更好地服务用户需求。

# 致 谢

在这段时间里，我全力以赴，将平日里所学的理论知识与工作实践深度融合。本次论文的撰写，正是基于我实际工作经验展开的。长期的工作积累，让我对业务有了深刻的理解，且巩固了理论知识，使我得到二次成长。回顾实际项目的开展过程，从现场用户需求调研，对需求进行专业的剖析设计，转交技术人员进行系统开发，到最后的系统测试。这每一个环节都凝聚着心血与汗水。

如今，论文即将完成，我心中满是感激。在此，我要特别感谢我的论文指导老师。从开题报告的构思到正文的撰写，老师始终为我指明方向。正是因为有了老师的帮助，我才能在论文创作的道路上少走弯路，顺利完成这篇论文。未来的道路上，我将带着老师的教诲和期望，继续努力前行，不断提升自己的能力和水平。再次衷心感谢老师的辛勤付出。

**参考文献：**

[1]张海藩，软件工程导论（第6版），北京:清华大学出版社,2013.8.

[2]张贵强,王美玲.基于NodeJS的企业网站的设计与实现[J].信息技术与信息化,2019(12):58-60.

[3]何玉洁，数据库原理与应用教程，北京：机械工业出版社, 2010.9

[4][师晓利](https://www.kongfz.com/writer/2609467/" \t "http://item.kongfz.com/book/_blank),[刘志远](https://www.kongfz.com/writer/27607/" \t "http://item.kongfz.com/book/_blank).Vue.js前端开发实战.[人民邮电出版社](https://www.kongfz.com/publisher/3060/" \t "http://item.kongfz.com/book/_blank)，2020

[5]史济民.软件工程——原理、方法与应用(第三版)，.北京:高等教育出版社,2009

[6]张浩洋,顾丹鹏,陈肖勇.基于Vue的数据管理平台实践与应用[J].计算机时代,2022(07):66-67+72.DOI:10.1s6644/j.cnki.cn33-1094/tp.2022.07.017.

[7]何玉洁，数据库基础与实践技术，北京:机械工业出版社,2020.

[8]赵韶平，PowerDesigner系统分析与建模（第2版）北京:清华大学出版社,2010.1.

[9]许倩文,李治堂.流量经济下图书营销低价策略问题及治理研究[J].北京印刷学院学报,2025,33(01):41-47.DOI:10.19461/j.cnki.1004-8626.2025.01.007.

[10]何振兴.计算机软件开发的数据库测试技术探析[J].中国战略新兴产业,2025,(06):17-19.