

《软件工程》实验报告

# 列车票务管理系统开发文档

组长：

组员：

组员：

组员：

指导老师： 陈波

2025年1月7日

目录

[列车票务管理系统开发文档 1](#_Toc1ioc3a)

[目录 2](#_Tocnyiolv)

[前言 5](#_Tocmk7lfk)

[1可行性研究 6](#_Toco3vx52)

[1.1 系统概述 6](#_Tocrull36)

[1.2初步的系统分析设计 8](#_Toc5njpl2)

[1.2.1 业务流程图 9](#_Toc4dxqul)

[1.2.2 数据流图 12](#_Toco8g1ee)

[1.2.3 数据字典 13](#_Toccuvlze)

[1.3技术可行性分析 18](#_Toc0c5gyd)

[1.4 经济/效益可行性分析 19](#_Toc87ieu2)

[1.4.1开发成本 19](#_Tocbwwaey)

[1.4.2 运行费用 19](#_Tocksmjop)

[1.4.3 经济效益 20](#_Toc0w2j11)

[1.5 系统开发计划 21](#_Tocvo0nac)

[2 需求分析 22](#_Toc3rxbvu)

[2.1总体目标 22](#_Toc3slt9j)

[2.2具体目标 22](#_Tocoitqtg)

[2.3系统数据建模 23](#_Tociw7yqk)

[2.4 系统功能建模 29](#_Toczy2fcd)

[2.5 数据字典 30](#_Toc5umrjs)

[3总体设计 36](#_Toc190vsc)

[3.1系统结构图 36](#_Toccrc84j)

[3.2数据库设计 38](#_Toc45lfqa)

[3.2.1逻辑结构设计 38](#_Tocbudngr)

[3.2.2物理结构设计 44](#_Toci538ea)

[3.3 系统模块IPO表 45](#_Toc7g27t5)

[4 详细设计 66](#_Tocbkrzlg)

[4.1用户注册模块 66](#_Tocwvebx2)

[4.2用户个人中心模块 67](#_Tocq07doh)

[4.3用户浏览列车模块 68](#_Toczfm4qh)

[4.4用户查看订单功能模块 69](#_Tocpmr3ls)

[4.5用户查看购物车功能模块 70](#_Tocot2gf6)

[4.6用户付款功能模块 71](#_Tocvkybwb)

[4.7管理员修改个人信息功能模块 71](#_Tochlz9qp)

[4.8管理员注册功能模块 72](#_Tocwg1h96)

[4.9管理员查看列车信息功能模块 74](#_Tochk4ve2)

[4.10管理员查看订单/退单信息功能模块 75](#_Tocoq7dh9)

[4.11用户注销登录模块 76](#_Tocdnj7tt)

[4.12管理员确认专业课程书单信息功能模块 77](#_Toct5a6yu)

[4.13管理员生成报表并查看报表 78](#_Toch4fhf6)

[4.14用户登录模块 79](#_Toc2dv1u5)

[4.15管理员修改个人信息模块 80](#_Toct5sr1t)

[4.16管理员查看订单功能模块 81](#_Toche0rln)

[4.17管理员管理管理员信息功能模块 82](#_Toc7iqh1j)

[5系统实现 84](#_Tocikhpwu)

[5.1 开发平台和开发环境介绍 84](#_Toc7zextv)

[5.2 编码 85](#_Toc1x0mf8)

[5.3 前台界面与后台数据库连接代码 85](#_Toct1g9yp)

[5.4各模块功能的实现。 87](#_Toct62oqe)

[5.4.1 主界面模块实现 87](#_Tocste75d)

[5.4.2注册模块实现 88](#_Toc30johe)

[5.4.3 登录模块实现 88](#_Toc90ygio)

[5.4.4 用户浏览列车模块实现 89](#_Toc57i7rq)

[5.4.5快速检索列车模块实现 90](#_Toc0015q5)

[5.4.6用户注销登录状态模块实现 90](#_Tocrkniyr)

[5.4.7用户查看购物车模块实现 90](#_Toc4ubje0)

[5.4.8用户查询订单和申请退款模块实现 91](#_Tocl02m9g)

[5.4.9用户修改个人信息模块实现 92](#_Tocuuniyi)

[5.4.10 管理员确认订单模块实现 92](#_Toc0jrkcp)

[5.4.11 管理员确认退单模块实现 93](#_Toc5bciw9)

[5.4.12 管理员管理列车信息模块实现 93](#_Tocfnnpdf)

[5.4.13 书店报表生成模块实现 94](#_Tocgb0i3l)

[5.2 测试 95](#_Tocpm8izr)

[5.2.1 软件测试的目标 95](#_Toc22poyw)

[5.2.2 软件测试的步骤 95](#_Tocfslvpg)

[5.2.3 单元/集成/验收测试 95](#_Tocyydd3n)

[5.2.4测试过程 97](#_Toc6gc1c7)

[6 维护 104](#_Toc5oxg5c)

[6.1 系统维护过程 104](#_Tocjprcl0)

[6.2 系统维护策略 104](#_Tocrvrvq0)

[7总结与体会 106](#_Tocafjrr1)

[7.1 总结 106](#_Toc09j6n8)

[7.2 体会 106](#_Tocw3rbcb)

[8 附录 111](#_Tocf2xrc7)

# 前言

本文档是根据软件工程的基本原理、开发方法以及开发过程进行了多次的讨论与实践的成果，本系统按照软件生命周期的各个阶段相应的任务进行开发，主要包括可行性研究、需求分析、总体设计、详细设计、编码、测试等内容，每一章节都图文并茂地阐述了具体的设计过程，使得该文档具有较高的可读性，易于开发人员进行设计和维护。

# 1可行性研究

## 1.1 系统概述

随着社会的发展和人们生活水平的提高，铁路交通作为重要的出行方式，其需求也在不断增长。列车票务管理系统因此成为人们日常生活和出行中不可或缺的一部分，其重要性日益凸显。

对于铁路公司而言，采用列车票务管理系统带来了诸多利益。首先，它极大地节约了人力成本，不再需要大量管理员来处理繁琐的预订工作。通过自动化和在线预订，铁路公司能够更高效地管理乘客信息和列车安排，提高整体运营效率。其次，列车票务管理系统为铁路公司提供了一个与乘客直接互动的平台，给乘客带来了良好的购票体验。

而对于旅客而言，使用列车票务管理系统带来了极大的便利和灵活性。他们可以随时随地通过网络订购列车票，无需亲自前往售票点或排队等候。这不仅简化了购票流程，节省了宝贵的时间和精力，还为旅客提供了更多选择和比较的机会。通过在线系统，旅客可以直观地查看列车时刻表、座位选择、票价比较以及其他额外服务，使得他们能够更好地计划和安排行程。

此外，网上列车票订购系统还大大促进了铁路公司和旅客之间的信息交流和沟通。通过系统提供的反馈渠道，旅客可以轻松地向铁路公司提出疑问、投诉或建议，而铁路公司也能够及时回应和解决问题，提升了服务质量和客户满意度。

综上所述，网上列车票订购系统对铁路公司和旅客都带来了巨大的好处。它不仅提高了铁路公司的运营效率和管理便捷性，还为旅客提供了更便利、灵活和个性化的服务体验。随着技术的不断发展和创新，可以预见，未来的列车票预订系统将会进一步演化，为人们的出行提供更加便捷和智能化的选择。

为了提供更便捷的服务，我们计划开发一个网上列车票订购系统，以满足旅客的需求。通过该系统，旅客可以方便地查询列车信息，如目的地、出发时间和列车班次等。售票员将预定列车票的旅客信息（姓名、性别、身份证号码、旅行时间、旅行目的地等）输入系统后，系统将为旅客安排列车，并生成取票通知和帐单。旅客在收到取票通知和帐单后，可以进行支付，并在列车出发前一天凭取票通知和交款单进行列车票打印。系统将根据旅客提供的信息进行校对，确保列车票准确无误。

旅客可以通过该系统方便地查询列车信息，例如目的地、出发时间和列车班次等。售票员将预定列车票的旅客信息输入系统，系统会为旅客安排列车，并生成取票通知和帐单。旅客在收到取票通知和帐单后，可以进行支付，并在列车出发前一天凭取票通知和交款单进行列车票打印，通过系统校对无误后交付给旅客。

然而，考虑到列车票购买人群的多样性，除了售票员，还有许多散客需要服务。为了满足所有旅客的需求，我们决定将原本的"售票员-旅客"关系改变为"旅客-关联旅客"关系。旅客可以在系统中设定关联旅客的信息，这些信息将存储在数据库中，避免了每次重复输入。这样既满足了售票员一次性为大量旅客订购列车票的需求，也满足了私人出行时为家人或朋友一起订购列车票的需求。

对于取票流程，除了传统的人工服务台取票方式外，我们还决定引入一种简化的自助取票方式。旅客可以在网页登录后，使用银行卡、支付宝、微信等网上支付方式进行付款。系统将已支付信息和电子账单保存在数据库中，旅客可随时在网上查阅。在列车出发前一天，旅客可以在列车站的列车票打印终端登录自己的账号，打印已支付的列车票。旅客也可以选择现场支付后打印车票。这一简化流程取消了实物取票通知、账单和交款单所带来的繁琐和不便。

工作负荷：现有系统需承担计算机学院所有的旅客订/退票业务。

目标：在计算机网络，数据库和先进的开发平台上，利用现有的软件，配置一定的硬件，开发一个具有开放体系结构的、易扩充的、易维护的、具有良好人机交互界面的列车票务管理系统，实现铁路公司的列车票销售的自动化的计算机系统，为企业的决策层提供准确、精细、迅速的列车票销售信息，为旅客提供快捷、方便的服务。综上，这个系统要求能够实现实时，高效，便捷，完善，安全性好等特点，具体如下：

一、性能要求：

订单存储：售票系统的订单必须无差错地存储在列车站的主服务器上。这意味着系统必须能够准确地将所有订单信息保存在服务器上，以便后续的处理和管理。

数据刷新：对服务器的数据必须进行及时的刷新。这意味着系统需要能够及时更新订单状态、可用座位和列车信息等数据，以保证用户获取的信息是最新的和准确的。

二、输出要求：

数据完整：系统输出的数据必须是完整的，不应有任何遗漏或缺失。用户在查询列车信息、选择座位和购买列车票时，系统应该提供所有相关的数据，以便用户做出准确的决策。

数据详实：系统输出的数据应该是详实的，包含列车的具体信息、座位的可用情况、票价和附加服务等。用户应该能够清楚地了解列车的相关信息，以便做出适当的选择和决策。

三、输入要求：

简捷：系统的输入过程应该是简捷的，不应该有过多繁琐的步骤或冗余的信息填写。用户在查询列车、选择座位和购票时，应该能够通过简单直观的界面快速完成操作。

快速：系统应该能够快速响应用户的输入，并及时提供相关的结果和信息。用户不应该遇到长时间的等待或延迟，以确保他们能够顺利地完成购票流程。

实时：系统应该能够提供实时的数据和信息。这意味着系统应该及时更新列车的可用座位、票价和其他相关信息，以便用户获取最新的和准确的数据。

四、安全与保密要求：

服务器管理员权限：服务器的管理员享有对列车时刻表信息库、列车票信息库和订票信息库进行管理和修改的权限。他们可以确保这些数据库的安全性和准确性，包括列车时刻表信息、列车票信息和订票信息的管理。

售票员权限：售票员仅享有对订票信息库进行部分修改的权限，包括写入和读出订票信息。这样的权限限制有助于保护旅客的个人信息和订票数据的安全与保密。

## 1.2初步的系统分析设计

列车票务管理系统包括以下几个系统模块：

用户中心模块：该系统模块描述了用户端关于个人资料及关联用户的功能，具有查看个人资料、修改个人信息、查看关联用户、增添关联用户、删除关联用户功能。用户可以在此模块中管理自己的账户信息，包括但不限于联系方式、密码更改等，同时可以管理与自己账户关联的其他用户信息，方便为家人或朋友预订列车票。

订单模块：该系统模块描述了用户端关于列车票订单操作的功能，具有查看列车票订单、支付列车票款、打印列车票功能。用户可以在此模块中管理自己的订单，包括查看订单详情、进行支付操作以及打印或下载列车票，以便在乘车时使用。

订票模块：该系统模块描述了用户端关于订票的功能，具有检索并查看列车时刻、根据日期为乘车人订购列车票功能。用户可以在此模块中根据出发地、目的地、出发日期等条件搜索列车，选择合适的车次进行预订，并填写乘车人信息完成购票。

管理员模块：该系统模块描述了系统管理员端各项操作功能，具有列车时刻查阅、列车时刻添加、列车时刻管理、列车票阅读、列车票添加、列车票管理、订单查阅、订单管理、用户查阅、用户管理功能。管理员可以在此模块中进行系统维护和管理工作，包括更新列车时刻表、管理列车票信息、处理订单和用户信息等，确保系统的稳定运行和数据的准确性。

### 1.2.1 业务流程图

业务流程图是概括得描绘物理系统的传统工具。其基本思想是利用图形符号以黑匣子的形式描绘组成系统的各个部件。其强调的是数据在系统各部件间的流动情况，选择性地暂时忽略对各数据加工处理的控制过程。以一种物理数据流图的形式展现系统的各个功能运作流程，为人们初步理解项目系统提供帮助。基于上述分析的四个系统模块，本项目系统的业务流程图如下图1.1所示。

(1)用户注册-登录账号业务

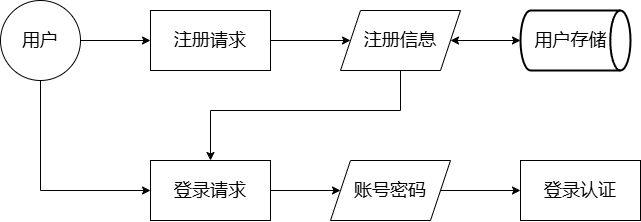


图1.1用户注册-登录账号业务流图

(2)用户修改个人信息业务



图1.2用户修改个人信息业务流图

(3)用户检索-查阅-订购列车票业务

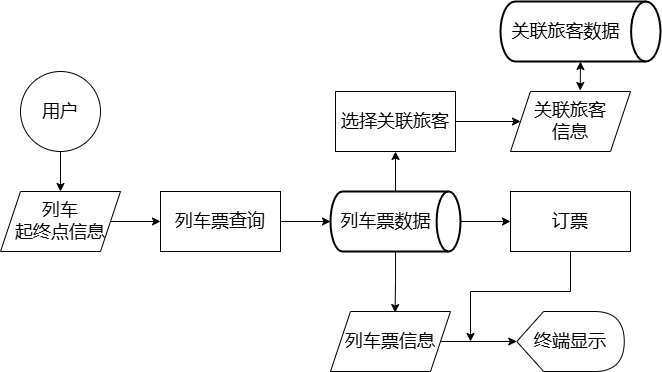


图1.3用户检索-查阅-订购列车票业务流图

(4)用户支付-退订列车票业务

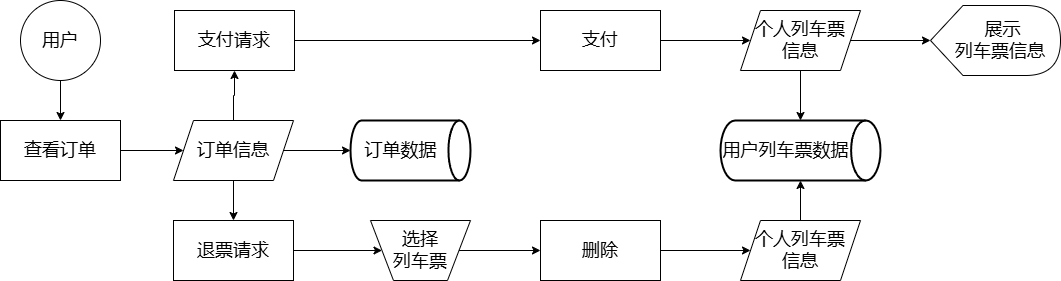


图1.4用户支付-退订列车票业务流图

(5)管理员登录业务



图1.5管理员登录业务流图

(6)管理员查阅-管理列车时刻表业务

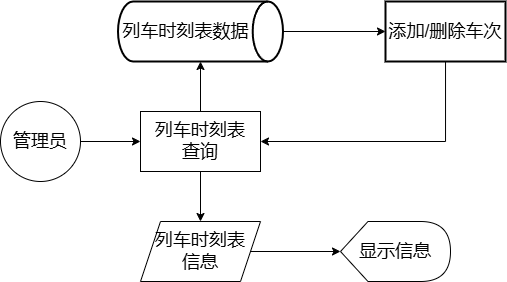


图1.6管理员查阅-管理列车时刻表业务流图

(7)管理员查阅-管理列车票业务

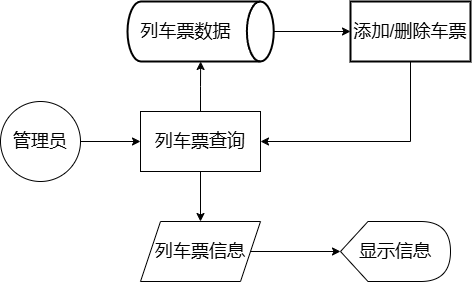


图1.7管理员查阅-管理列车票业务流图

(8)管理员查阅-管理订单业务

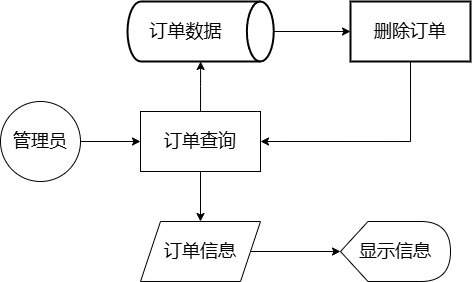


图1.8管理员查阅-管理订单业务流图

(9)管理员查阅-管理用户信息业务



图1.9管理员查阅-管理用户信息业务流图

(10)用户对关联旅客的管理业务

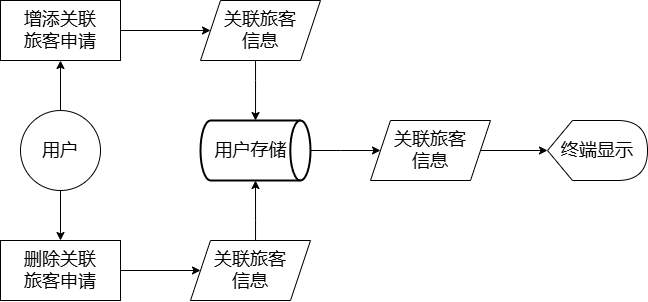
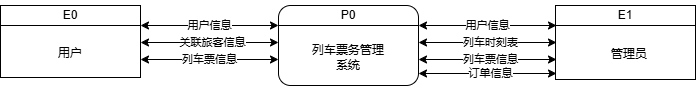


图1.10用户对关联旅客的管理业务流图

### 1.2.2 数据流图

数据流图（DFD）是一种图形化技术，其描绘了信息流和数据从输入移动到输出的过程中所经受的变换。在数据流图中没有任何具体的物理部件，它只是描绘数据在软件中流动和被处理的逻辑过程，构建的是系统的逻辑模型。画数据流图的目的是利用它作为交流信息、分析和设计的工具。首先，基于上述业务流程图，绘制出系统的顶层数据流图，如下图1.11所示。

图1.11顶层数据流图

该顶层数据流图仅用一个加工过程P0来表示整个系统，用E0、E1表示该系统全部的数据输入输出实体。顶层数据流图作用是在最高层次上展示数据流动情况，为后续进一步弄清系统数据流动和处理的过程夯实基础。

其次，在上述顶层数据流图的基础上，项目将对系统加工过程P0进行初步细化，得到0层数据流图。该0层数据流图将系统加工过程按之前的模块分成相应的几个部分，同时为这几个部分链接了对应的数据库，展现出了更为详细且更接近于现实的系统逻辑结构模型，该0层数据流图如下图1.12所示。

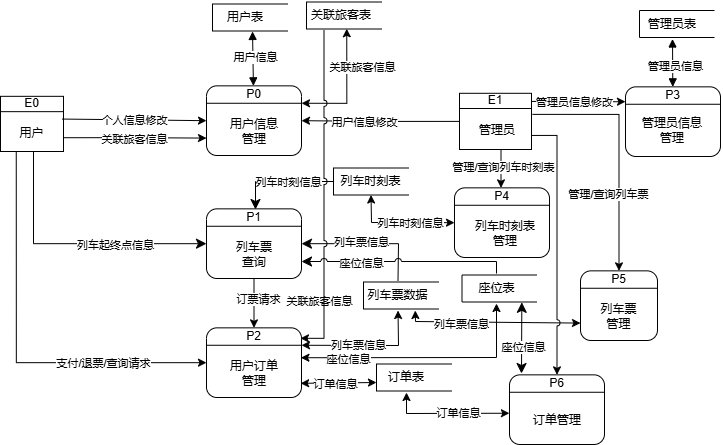


图1.12 0层数据流图

### 1.2.3 数据字典

数据字典是关于数据的信息的集合，也就是对数据流图中包含的所有元素的定义的集合，其主要功能是在软件分析和设计的过程中给人提供关于数据的描述信息。经过分析，该项目需要建立用户表、管理员表、列车票数据、列车时刻表、订单表、关联旅客表、座位表等等，其初步的数据字典如下表1.1所示：

表1.1用户数据字典

|  |
| --- |
| 名称:用户 |
| 描述:预订列车票的用户 |
| 定义:用户=身份证号+姓名+电话+性别+密码+邮箱+账号状态 |
| 字段:身份证号=字符/50位  姓名=字符/30位  电话=字符/30位  性别=字符/2位  密码=字符/50位  邮箱=字符/50位  账号状态=[0|1] |
| 位置:用户表 |

表1.2管理员数据字典

|  |
| --- |
| 名称:管理员 |
| 描述:管理列车票务系统的用户 |
| 定义:管理员=账号+密码 |
| 字段:账号=字符/30位  密码=字符/50位 |
| 位置:管理员表 |

表1.3列车票数据字典

|  |
| --- |
| 名称:列车票 |
| 描述:可供用户购买的商品 |
| 定义:列车票=车票编号+列车号+出发时间+座位号+上车站+下车站+持票人身份证号码+价格+是否付款 |
| 字段:车票编号=字符/50位  列车号=字符/50位  出发日期=日期  出发时间=时间  座位号=字符/10位  上车站=字符/20位  下车站=字符/20位  持票人身份证号=字符/30位  价格=浮点型  是否付款=布尔型 |
| 位置:列车票数据 |

表1.4列车班次数据字典

|  |
| --- |
| 名称:列车班次 |
| 描述:描述列车车次信息 |
| 定义:列车班次=列车号+始发站+终点站+途径站+发车时间+座位数+是否删除 |
| 字段:列车号=字符/50位  始发站=字符/20位  终点站=字符/20位  途径站=字符/10000位  发车日期=日期  发车时间=时间  座位数=整数  是否删除=布尔型 |
| 位置:列车时刻表 |

表1.5订单数据字典

|  |
| --- |
| 名称:订单 |
| 描述:记录订购相关信息的表单 |
| 定义:订单=订单号+用户身份证号+创建时间+所含列车票+金额总数+备注+订单状态 |
| 字段:订单号=字符/20位  用户身份证号=字符/50位  创建日期=日期  创建时间=时间  所含列车票=车票编号  金额总数=浮点型  备注=字符/500位  订单状态=[0|1|2]，分别代表已支付，已修改，已取消 |
| 位置:订单表 |

表1.6关联旅客数据字典

|  |
| --- |
| 名称:关联旅客 |
| 描述:与用户关联的旅客 |
| 定义:关联旅客=用户身份证号+关联身份证号+关联旅客姓名+关联旅客电话+关联旅客性别 |
| 字段:用户身份证号=字符/50位  关联旅客身份证号=字符/50位  关联旅客姓名=字符/30位  关联旅客电话=字符/30位  关联旅客性别=布尔型【男|女】 |
| 位置:关联旅客表 |

表1.7座位数据字典

|  |
| --- |
| 名称:座位 |
| 描述:列车上的座椅 |
| 定义:座位=列车号+座位号+座位类型+列车站点 |
| 字段:列车号=字符/50位  座位号=字符/10位  座位类型=字符/30位  列车站点=链表，节点存储车站和是否有人 |
| 位置:座位表 |

## 1.3技术可行性分析

列车票预订系统是一个基于Java和SQL语言、应用Vue和Spring Boot框架的Web应用系统。MySQL是一个小型关系型数据库管理系统，广泛应用于中小型网站的数据库管理中。由于其体积小、速度快、总体拥有成本低，尤其是开源的特点，因此本项目选择了MySQL作为列车票系统的数据库。

MySQL是一个关系型数据库管理系统，采用关系模型将数据保存在不同的表中，而不是将所有数据放在一个大仓库内。这种方式不仅提高了数据处理的效率，还增强了系统的灵活性。在WEB应用领域，MySQL是最流行的RDBMS之一，尤其在列车票预订系统等应用场景中，MySQL的优势得到了广泛的应用。

MySQL使用标准的SQL语言进行数据操作，这种语言被广泛应用于各种数据库管理系统中。MySQL提供了社区版和商业版，满足不同规模和需求的开发者。由于其小巧、快速和低成本的特点，MySQL通常被选择用于中小型和大型网站的数据库开发。

Navicat是一款强大的数据库管理工具，支持MySQL、Oracle、PostgreSQL、SQLite、SQL Server、MariaDB和MongoDB等多种数据库类型。它可以帮助开发者方便地管理数据库，支持阿里云、腾讯云、AWS、Oracle Cloud等云平台的数据库兼容性。

IDEA（IntelliJ IDEA）是用于Java语言开发的集成开发环境（IDE）。它是业界公认的最佳Java开发工具之一，提供了智能代码助手、代码自动补全、重构、JUnit支持、CVS整合等强大功能。其创新的GUI设计和高度自动化的开发体验，使得开发者能够高效开发列车票预订系统。

Java Web技术是用Java技术解决互联网领域问题的技术栈，包括Web服务端和Web客户端。Java在Web客户端的应用有Java Applet，在服务器端，Java技术通过Servlet、JSP以及各种第三方框架提供强大的支持。Java Web技术为列车票预订系统的后端开发提供了稳固的基础。

Vue.js是一个流行的JavaScript前端框架，常用于构建动态的用户界面。其组件化的开发方式能够帮助开发者构建可复用的UI组件，快速实现交互性强、响应迅速的单页面应用。Vue具有简单易学的语法和灵活性，适合用来开发列车票预订系统的用户界面和交互逻辑。

Spring Boot是一个用于构建Java应用程序的开源框架。基于Spring框架，Spring Boot旨在简化开发和部署流程，提供了自动化配置、内嵌Web服务器、健康检查等功能。Spring Boot为列车票预订系统提供了后端服务的开发支持，处理用户请求、数据持久化和业务逻辑的实现。

为了高质量地完成系统开发工作，还需要遵循以下准则：

1. 做好数据规划，建立稳定的信息模型；
2. 按照功能模块划分系统，确保各子模块按需求组织；
3. 设计友好的人机界面，确保系统的易用性和适应性。

综上所述，列车票预订系统的技术实现是完全可行的，通过合理的技术栈和系统架构设计，可以高效地实现系统功能和优化用户体验。

## 1.4 经济/效益可行性分析

### 1.4.1开发成本

列车票预订系统的开发隶属于一般性软件项目开发工作，其主要涉及的开发成本来源于前期项目准备、软件设计、编码实现以及产品测试人员的人力成本，其所有使用的开发软件、开发框架均属于开源性质。

### 1.4.2 运行费用

该项目的运行费用主要包括后台运营、维护人员的费用和物理服务器的购置与维护费用。

1. 后台运营、维护人员费用：

后台运营、维护人员的主要职责包括网站后端数据库管理、漏洞修复和网络安全维护，以及为用户提供文档、培训等支持。这些是维护一个网站所必需的基本工作。招聘人员需要具备基本的数据库知识、掌握Springboot基础内容和常用JavaWeb框架，同时具备一定的网络安全知识和文字编写能力。考虑到地区差异和任务分工，建议招聘两名人员，每人月薪约为5000元左右，总计约10000元左右。

2. 物理服务器的购置与维护费用：

物理服务器的购置与维护是基本开销。由于该项目用户规模有限，且并发要求不高，因此对服务器性能要求中等即可。项目需要一台主力服务器和一台备用服务器。根据当前市场价分析，每台服务器约为2000元左右，两台服务器的总费用约为4000元左右。

以上费用仅针对运行阶段，不包括项目开发和其他额外费用。具体费用可能根据实际情况和市场变动有所调整。在制定预算时，建议进行详细的成本分析和预算规划，以确保项目运行的可持续性和稳定性。

### 1.4.3 经济效益

该列车票预订系统是一个小型数据库系统，尽管其开发和实施不需要过多的人力和物力投入，但一旦投入使用，将显著提高工作效率，主要体现在以下几个方面：

**自动化劳动替代**：该系统的运行可以替代人工进行许多繁琐的工作，减少了人力资源的消耗。例如，传统的人工售票和票务管理过程中，工作人员需要手动处理大量的车票信息、客户数据和支付流程，而系统能够自动化完成这些任务，从而大大提高工作效率。

**资源节约**：该系统的运行可以节省许多资源，包括人力、财力和物力等方面。传统的纸质售票和手动操作不仅浪费大量的时间，还容易发生错误，而通过该系统，可以减少人为错误并节省办公空间和物资消耗，有效降低了资源浪费。

**时间节约**：该系统能够大大缩短购票时间，使得列车票的购买过程更加高效便捷。用户通过系统可以轻松查询车次、选择座位、支付票款等，避免了排队购票、填写纸质表单等繁琐环节。系统还支持在线选座和票务动态调整，使得用户可以随时随地完成购票操作，极大提升了购票效率。

**个人信息安全**：系统能够提高用户个人信息的安全性。相比传统的人工购票，信息在处理过程中容易出现泄露的风险，而采用该系统后，个人信息通过加密和安全认证进行保护，减少了信息泄露的可能性。这使得用户更加信任并愿意使用该系统，提高了用户满意度。

从铁路公司自身的角度来看，引入这样的系统是非常必要的。在计算机普及之前，信息通常需要通过纸质填写的方式进行，这不仅浪费时间，还容易发生错漏。随着信息技术的快速发展和普及，现代铁路系统需要通过提高效率来应对日益增长的需求，列车票务系统正是提升工作效率、减少人工成本、改善用户体验的关键所在。该系统的开发成本相对较低，所需的其他软件和硬件系统也容易获取。

一旦该系统投入使用，相较于传统的人工售票和纸质车票的方式，系统将具有更高的效率、更低的成本和更优质的服务。通过数字化管理，铁路公司能够节省大量人力、物力和财力资源，同时提高工作效率并降低管理成本。因此，从成本效益分析来看，系统的效益将远大于投入成本。

## 1.5 系统开发计划

在开始我们正式的系统开发前，我们小组进行了大量的讨论与思索，所以在前两周我们完成了系统的可行性分析，需求分析和总体设计部分。把基本结构搭出来后，后面四周我们进行了代码的编写和系统设计的详细话。最后两周我们完成了程序的测试，实现了各模块的功能，完成了我们最终的实验报告。

# 2 需求分析

## 2.1总体目标

在当今互联网无处不在、不断改变人们生产生活的背景下，基于Web的列车票预订系统借助互联网的普及，旨在为大众提供方便、高效的服务。该系统的具体目标如下：

**1.方便用户购票：**

通过访问前台系统，用户可以轻松浏览和查询列车班次、座位信息，并在家中或任何地方完成购票。用户不再需要前往车站排队购票，系统提供了便捷的在线购票方式，消除了以往需要寻找售票窗口、等待长时间的繁琐过程，从而大大提高了购票的便利性和效率。

**2.实现铁路公司办公自动化：**

后台系统将铁路公司的办公工作流程进行数字化和规范化，大幅提高了工作效率。通过系统化的流程管理，铁路公司可以高效地处理列车信息、订单管理、票务处理和客户服务等任务，减少人工操作和出错率，提升管理员的工作效率。系统还能够自动化生成报表、实时更新列车时刻表和车票库存，确保信息的准确性和及时性。通过基于Web的列车票预订系统，用户能够获得便捷的购票体验，而铁路公司则能够提高工作效率、优化资源分配并提升服务质量，从而更好地满足用户的需求。系统的目标是通过互联网技术的应用，实现用户与铁路公司之间的无缝连接和高效沟通，使购票过程更加顺畅、便捷，提升整体服务水平。

## 2.2具体目标

（1）用户中心模块：

查看个人信息，修改个人资料，查看关联旅客/用户，管理关联旅客/用户。

（2）车票列表模块：

查看已购车票、支付车票、打印车票。

（3）订购车票模块：

检索并查阅票务信息，查询票务时刻/详情，订购票务。

（4）系统管理员模块：

查阅列车班次信息、管理列车班次、查阅车票、管理车票、查阅订单、管理订单、查阅用户及相关信息、管理用户。

## 2.3系统数据建模

系统的数据模型包含三个互相关联的要素：数据对象、数据对象的属性和数据对象之间的关系。为了准确描述用户的数据需求，本项目采用实体-联系图（E-R图）的方式。E-R图是一种面向问题的概念性数据模型，它从用户的角度出发，反映了用户现实环境中的数据关系，而与具体软件系统的实现方法无关。

全局的E-R图设计如下图2.1所示：

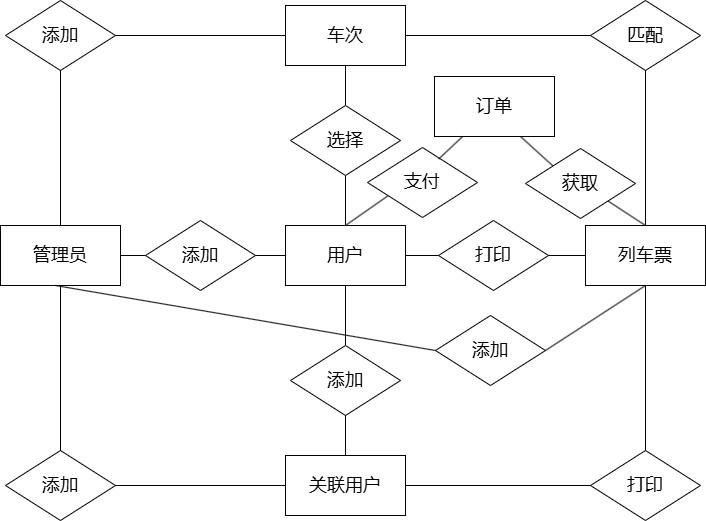


图2.1全局E-R图

对全局E-R图2.1进行分解，得到以下局部E-R图

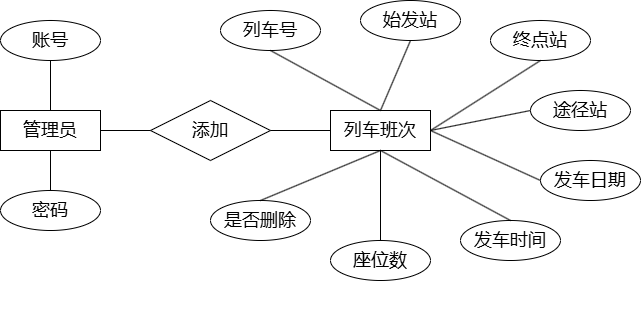


图2.2管理员添加列车班次

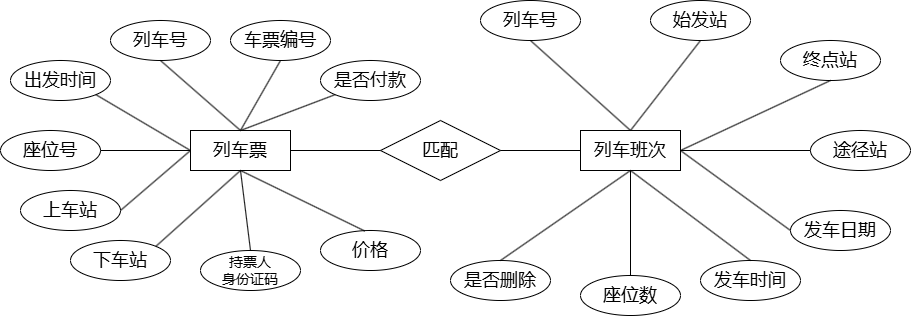


图2.3列车票匹配列车班次

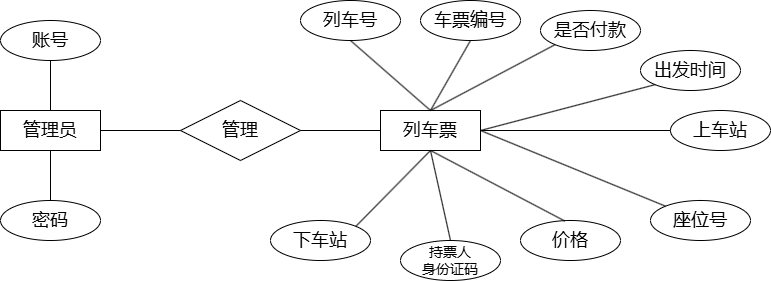


图2.4管理员管理列车票

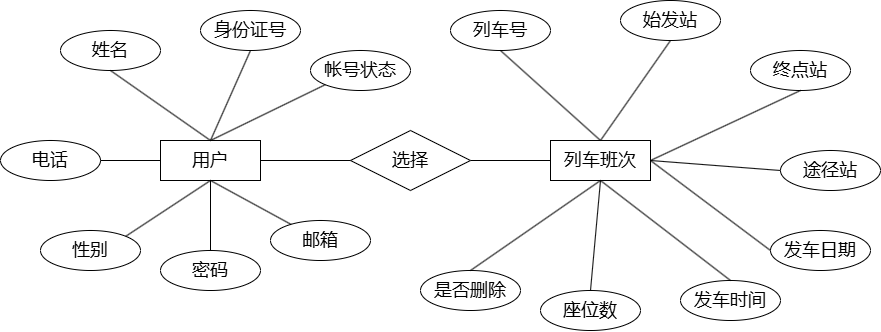


图2.5用户选择列车班次

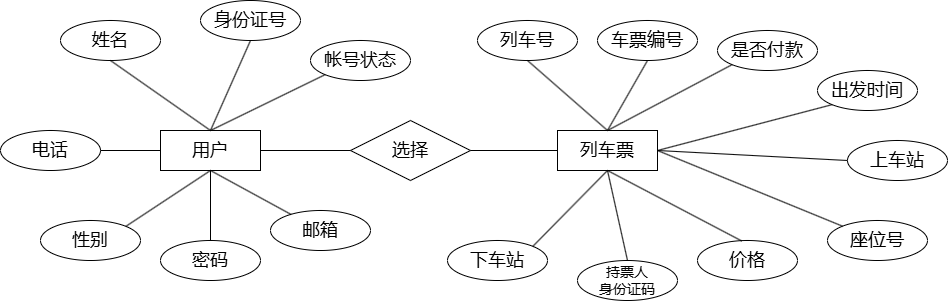


图2.6用户选择列车票

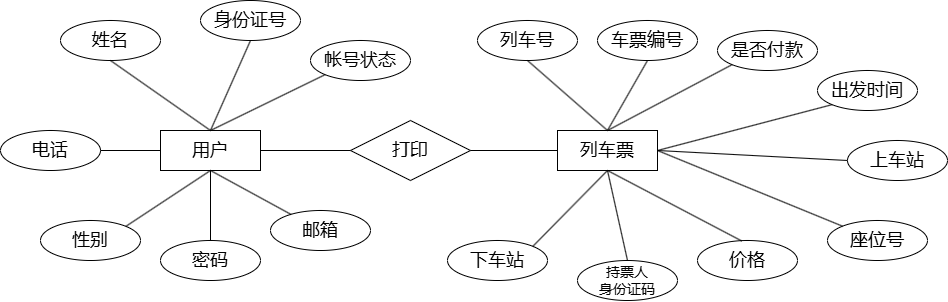


图2.7用户打印列车票

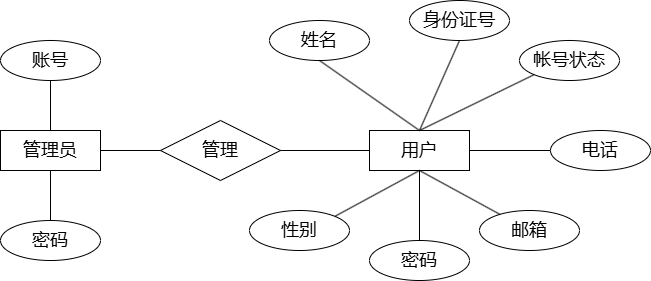


图2.8管理员管理用户

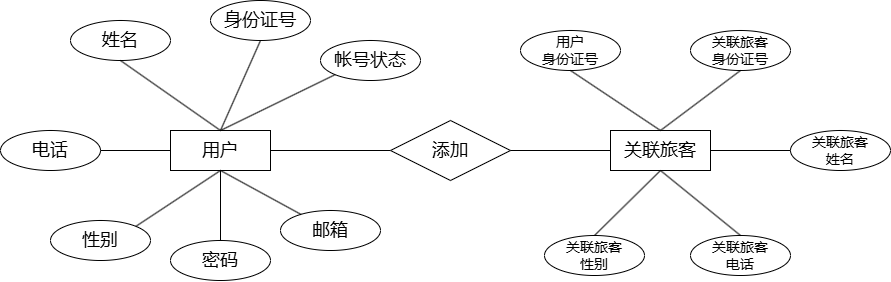


图2.9用户添加关联旅客

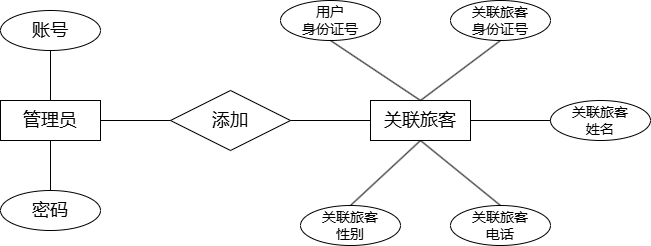


图2.10管理员添加关联旅客

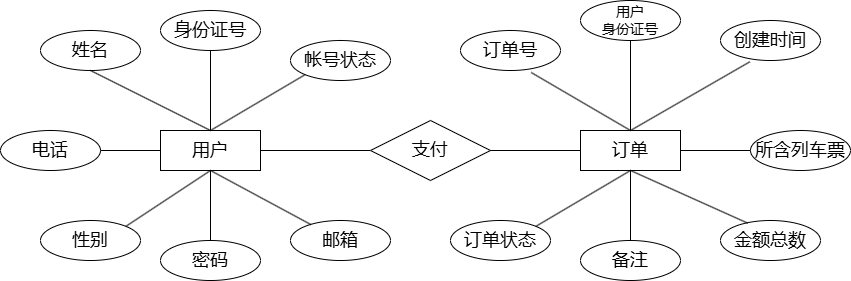


图2.11用户支付订单

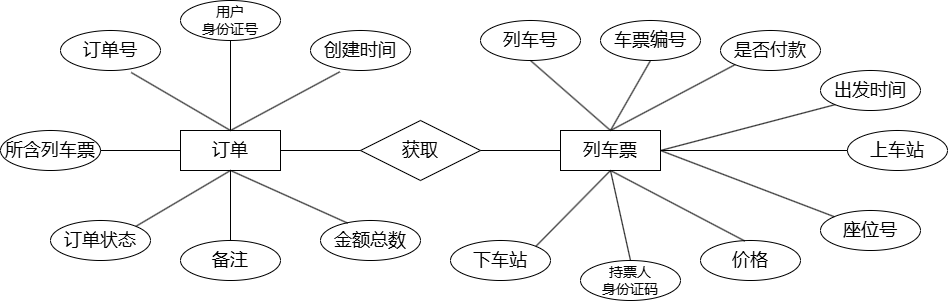


图2.12订单获取列车票

## 2.4 系统功能建模

如下图2.13顶层数据流图所示，该系统整体上可以分为两个实体和一个处理过程。其实体分别为E0用户、E1管理员，处理过程为P0列车票务管理系统。由于系统主要功能在可行性分析阶段已经进行了严谨的分析讨论，因此在该阶段可以直接沿用之前的顶层数据流图。该数据流图以人机交互思想为指导，从整体观念出发，展示了流入和流出系统的所有数据信息。

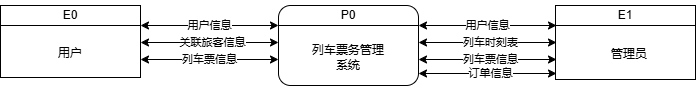
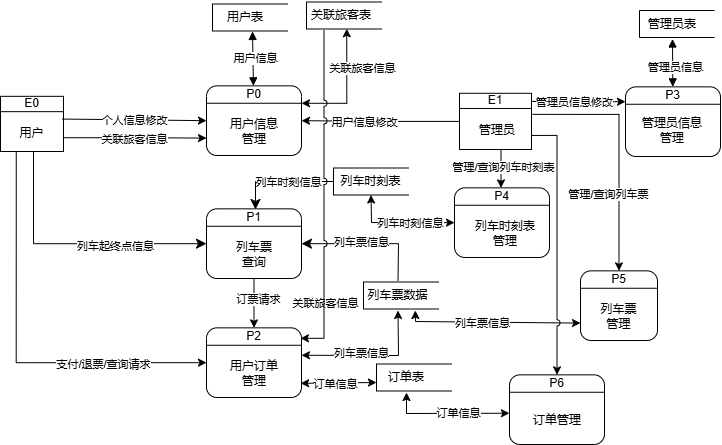


图2.13顶层数据流图

0层数据流图2.14将E0用户、E1管理员之间的信息数据流进行第一步细化。其中，将处理过程P0列车票务管理系统细分为P0用户信息管理、P1列车票查询、P2用户订单管理、P3管理员信息管理、P4列车时刻表管理、P5列车票管理、P6订单管理；并引入了多个数据存储，如用户表、关联旅客表、订单表、管理员表、列车时刻表、列车票数据。由于本系统在可行性研究阶段已经进行了详细的分析论证，因此0层数据流图可在需求分析阶段直接沿用。

图2.14 0层数据流图

## 2.5 数据字典

数据字典最重要的用途是作为分析阶段的工具，典型的情况是，在数据字段中记录数据元素的下列信息：一般信息（名称，描述），定义（数据类型，长度，结构等），使用特点（值的范围，使用频率，使用方式—输入、输出、本地、条件值等），控制信息（来源，用户，使用它的程序，改变权，使用权等）等。根据可行性研究中的初步分析，我们需要建立用户表、管理员表、列车票数据、列车时刻表、订单表、关联旅客表、座位表等。对在可行性分析研究阶段得到的数据字典进行补充完善如下：

表2.1用户数据字典

|  |
| --- |
| 名称:用户 |
| 描述:预订列车票的用户 |
| 定义:用户=身份证号+姓名+电话+性别+密码+邮箱+账号状态 |
| 字段:身份证号=字符/50位  姓名=字符/30位  电话=字符/30位  性别=字符/2位  密码=字符/50位  邮箱=字符/50位  账号状态=[0|1] |
| 位置:用户表 |

表2.2管理员数据字典

|  |
| --- |
| 名称:管理员 |
| 描述:管理列车票务系统的用户 |
| 定义:管理员=账号+密码 |
| 字段:账号=字符/30位  密码=字符/50位 |
| 位置:管理员表 |

表2.3列车票数据字典

|  |
| --- |
| 名称:列车票 |
| 描述:可供用户购买的商品 |
| 定义:列车票=车票编号+列车号+出发日期+出发时间+座位号+上车站+下车站+持票人身份证号码+价格+是否付款 |
| 字段:车票编号=字符/50位  列车号=字符/50位  出发日期=日期  出发时间=时间  座位号=字符/10位  上车站=字符/20位  下车站=字符/20位  持票人身份证号=字符/30位  价格=浮点型  是否付款=布尔型 |
| 位置:列车票数据 |

表2.4列车班次数据字典

|  |
| --- |
| 名称:列车班次 |
| 描述:描述列车车次信息 |
| 定义:列车班次=列车号+始发站+终点站+途径站+发车时间+座位数+是否删除 |
| 字段:列车号=字符/50位  始发站=字符/20位  终点站=字符/20位  途径站=字符/10000位  发车日期=日期  发车时间=时间  座位数=整数  是否删除=布尔型 |
| 位置:列车时刻表 |

表2.5订单数据字典

|  |
| --- |
| 名称:订单 |
| 描述:记录订购相关信息的表单 |
| 定义:订单=订单号+用户身份证号+创建时间+所含列车票+金额总数+备注+订单状态 |
| 字段:订单号=字符/20位  用户身份证号=字符/50位  创建日期=日期  创建时间=时间  所含列车票=车票编号  金额总数=浮点型  备注=字符/500位  订单状态=[0|1|2]，分别代表已支付，已修改，已取消 |
| 位置:订单表 |

表2.6关联旅客数据字典

|  |
| --- |
| 名称:关联旅客 |
| 描述:与用户关联的旅客 |
| 定义:关联旅客=用户身份证号+关联身份证号+关联旅客姓名+关联旅客电话+关联旅客性别 |
| 字段:用户身份证号=字符/50位  关联旅客身份证号=字符/50位  关联旅客姓名=字符/30位  关联旅客电话=字符/30位  关联旅客性别=布尔型【男|女】 |
| 位置:关联旅客表 |

表2.7座位数据字典

|  |
| --- |
| 名称:座位 |
| 描述:列车上的座椅 |
| 定义:座位=列车号+座位号+座位类型+列车站点 |
| 字段:列车号=字符/50位  座位号=字符/10位  座位类型=字符/30位  列车站点=链表，节点存储车站和是否有人 |
| 位置:座位表 |

# 3总体设计

## 3.1系统结构图

因为本系统在前面需求分析阶段已经得到了较为完善可行的顶层数据流图、0层数据流图以及1层数据流图，所以在本总体设计阶段直接沿用上述得到的图即可，具体详见图2.7-图2.11。

层次图可以用来描述软件的层次结构，适用于在自顶向下设计软件的过程中使用，本系统分别从用户端、管理员端、网站管理员端三个方面绘制系统结构图，具体如下所示：

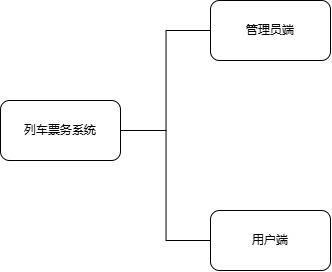


图3.1总体层次结构图

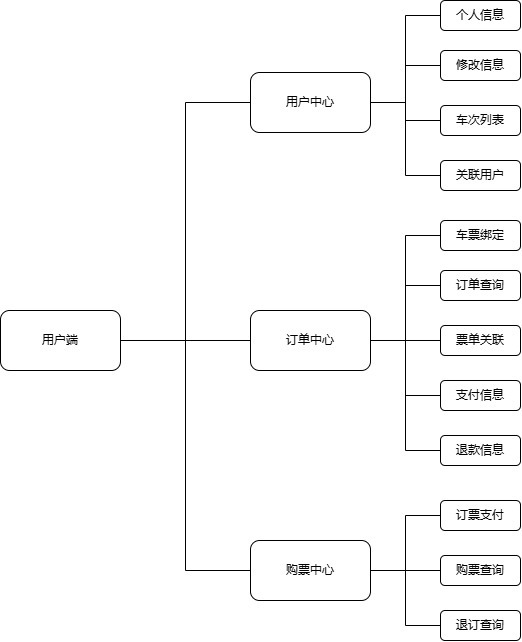


图3.2用户端层次结构图

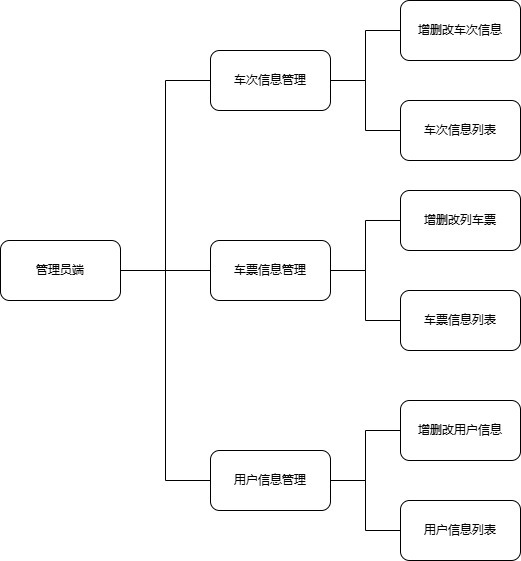


图3.3管理员端层次结构图

## 3.2数据库设计

### 3.2.1逻辑结构设计

#### （1）关系模式设计

系统的关系模式可由系统数据建模中的E-R图分析设计而得。其中E-R图模式中的实体和联系都可以转化成关系型，其属性可以转化为关系型的属性，实体型的主键也可以转化为关系型的主键。而对实体集之间的联系转换受到其自身类型的影响，需要具体问题进行具体分析。针对该项目中的联系转换，下面根据E-R图对本项目的关系模式做简要的介绍：

(1) 1:1型：无

(2) 1:n型： 车次-列车票（一个车次能生成若干张列车票，一张列车票隶属于一个车次），管理员-车次（一个管理员可以添加若干个车次，一个车次只能被添加一次），管理员-列车票（一个管理员能添加若干张列车票，一张列车票只能被添加一次），购票人-乘客（一个购票人可以为多个乘客购票，一个乘客只能的列车票只能被一个购票人购买），购票人-列车票（一个购票人可以购买多张列车票，一张列车票只能被一个购票人购买）。

(3) n:m型： 购票人-车次（一个购票人能购买多个车次的列车票，一个车次的列车票能被多个购票人购买）。

按上述要求建立系统的关系模式，其主要内容如下所示，其中，下划线表示该关系型的主键，波浪线表示该关系型的外键。

①用户表(身份证号,姓名,性别,密码,邮箱，账号状态)

②管理员表(账号，密码)

③列车票表(车票编号，列车号，出发日期，出发时间，座位号，上车站，下车站，持票人身份证号码，价格，是否付款)

④车次表(列车号，始发站，终点站，途径站，座位数，是否删除)

⑤订单(订单号，用户身份证号，创建时间，所含列车票，金额总数，备注，订单状态)

⑥关联旅客(用户身份证号，关联身份证号，关联旅客姓名，关联旅客电话，关联旅客性别)

#### （2）数据类型定义

接下来对关系模式设计中的属性定义类型、长度和约束，得到表3.1-表3.12。

表3.1用户表

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 数据项名 | 英文名 | 数据类型 | 长度 | 实体完整性约束 | 参照完整性约束 | 用户自定义性约束 |
| 身份证号 | U\_identity | VARCHAR | 50 | √ | / | / |
| 姓名 | U\_name | VARCHAR | 30 | / | / | not null |
| 电话 | U\_phone | VARCHAR | 30 | / | / | not null |
| 性别 | U\_sex | VARCHAR | 2 | / | / | 男/女 |
| 密码 | U\_password | VARCHAR | 50 | / | / | 默认为123456 |
| 邮箱 | U\_mail | VARCHAR | 50 | / | / | not null |
| 帐号状态 | U\_state | TINYINT  UNSIGNED | / | / | / | 0/1 |

表3.2 管理员表

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 数据项名 | 英文名 | 数据类型 | 长度 | 实体完整性约束 | 参照完整性约束 | 用户自定义性约束 |
| 账号 | A\_username | VARCHAR | 30 | √ | / | / |
| 密码 | A\_password | VARCHAR | 50 | / | / | 默认为123456 |

表3.3 列车票数据表

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 数据项名 | 英文名 | 数据类型 | 长度 | 实体完整性约束 | 参照完整性约束 | 用户自定义性约束 |
| 车票编号 | Ticket\_number | VARCHAR | 50 | √ | / | / |
| 列车号 | Train\_number | VARCHAR | 50 | / | √ | / |
| 出发日期 | Boarding\_date | DATE | / | / | / | / |
| 出发时间 | Boarding\_time | TIME | / | / | / | / |
| 座位号 | Seat | VARCHAR | 10 | / | / | not null |
| 上车站 | Boarding\_station | VARCHAR | 20 | / | / | not null |
| 下车站 | Arriving\_station | VARCHAR | 20 | / | / | not null |
| 持票人身份证号 | Holder\_identity | VARCHAR | 50 | / | √ | not null |
| 价格 | Price | DOUBLE | / | / | / | not null |
| 是否付款 | Payment | BOOLEAN | / | / | / | not null |

表3.4 列车时刻表

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 数据项名 | 英文名 | 数据类型 | 长度 | 实体完整性约束 | 参照完整性约束 | 用户自定义性约束 |
| 列车号 | Train\_number | VARCHAR | 50 | √ | / | not null |
| 始发站 | Departure\_station | VARCHAR | 20 | / | / | not null |
| 终点站 | Terminal\_station | VARCHAR | 20 | / | / | not null |
| 发车日期 | Departure\_date | DATE | / | / | / | not null |
| 发车时间 | Departure\_time | TIME | / | / | / | not null |
| 座位数 | Seat\_number | TINYINT  UNSIGNED | / | / | / | not null |
| 是否删除 | Delete | BOOLEAN | / | / | / | / |

表3.5 订单表

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 数据项名 | 英文名 | 数据类型 | 长度 | 实体完整性约束 | 参照完整性约束 | 用户自定义性约束 |
| 订单号 | Order\_number | VARCHAR | 20 | √ | / | not null |
| 用户身份证号 | U\_identity | VARCHAR | 50 | / | √ | not null |
| 创建日期 | Create\_date | DATE | / | / | / | not null |
| 创建时间 | Create\_time | TIME | / | / | / | not null |
| 所含列车票 | Ticket\_number | VARCHAR | / | / | / | not null |
| 金额总数 | Amount | DOUBLE | / | / | / | not null |
| 备注 | Remark | VARCHAR | 500 | / | / | / |
| 订单状态 | Order\_state | TINYINT UNSIGNED | / | / | / | not null |

表3.6 关联旅客表

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 数据项名 | 英文名 | 数据类型 | 长度 | 实体完整性约束 | 参照完整性约束 | 用户自定义性约束 |
| 用户身份证号 | U\_identity | VARCHAR | 50 | √ | √ | not null |
| 关联身份证号 | C\_identity | VARCHAR | 50 | √ | / | not null |
| 关联旅客姓名 | C\_name | VARCHAR | 30 | / | / | not null |
| 关联旅客电话 | C\_phone | VARCHAR | 30 | / | / | not null |
| 关联旅客性别 | C\_sex | BOOLEAN | / | / | / | not null |

### 3.2.2物理结构设计

数据库最终被存储在物理设备上，而数据库在物理设备上的存储结构和存取方法被称为数据库的物理结构。数据库的物理结构是依赖于具体的计算机系统的。数据库的物理结构设计是选择最适合应用环境的物理结构和存取方法，以提高数据库的访问速度并有效地利用存储空间。

物理结构设计的目的是通过优化数据的物理存储方式和访问方法，以提高数据库系统的性能和效率。这包括选择适当的存储设备、文件组织方式、索引策略、数据分区和数据压缩等技术手段，以满足应用需求，并确保数据的安全性和可靠性。

通过合理的物理结构设计，可以最大程度地提升数据库系统的性能，并为应用环境提供高效的数据访问和存储管理。

#### （1）聚簇设计

聚簇是一种数据存储结构，它将相关的数据元组集中存放在一个物理块内、若干相邻物理块内或同一柱面内，以提高查询效率。在建立聚簇时，需要考虑以下原则：

1.主要应用属性列：当某个关系的某些属性列是该关系的主要应用字段，而其他属性列的访问较少或是次要应用时，可以考虑在这些主要应用属性列上建立聚簇。

2.值重复率高的属性列：如果一个关系的某些属性列上的值重复率很高，可以考虑在这些属性列上建立聚簇索引。

3.数据变动较少的属性列：如果一个关系在数据装入后，某些属性列的值很少被修改，也很少增加或删除元组，可以考虑在这些属性列上建立聚簇索引。

需要注意的是，在Navicat Premium 15中，每张基本表已默认在主键属性列上建立了聚簇索引。由于每张基本表最多只能建立一个聚簇索引，因此不需要再进行额外的聚簇索引设计。

#### （2）分区设计设计

磁盘分区设计的核心目标是确定数据库数据的存储位置，以提高系统性能，这是数据库物理设计的重要内容之一。

在进行磁盘分区设计时，应遵循以下一般原则：

1.减少访问冲突，提高I/O并发性：当多个事务并发访问同一磁盘时，可能会导致磁盘访问冲突，从而降低系统效率。通过将事务访问的数据分布在不同的磁盘上，可以实现I/O并发执行，提高数据库的访问速度。

2.分散热点数据，均衡I/O负载：数据库中的数据访问频率通常是不均匀的，某些数据被频繁访问，称为热点数据。为了均衡各个磁盘的负载，可以将这些热点数据分散存储在不同的磁盘上，充分利用多磁盘并行操作的优势。

3.保证关键数据的快速访问，缓解系统瓶颈：在数据库中，某些数据如数据字典等具有高访问频率。为了确保对这些关键数据的快速访问，可以将其存储在某个特定的磁盘上，以保证其快速响应。

对于实验中的列车票务管理系统，由于其规模较小，不涉及大规模数据处理，因此不需要进行磁盘分区设计。

## 3.3 系统模块IPO表

### 3.3.1用户注册主页模块

(1)功能描述

对需要注册的用户提供注册服务。

(2)模块IPO表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 系统名称 | 列车票务管理系统 | | | |
| 模块名称 | 用户注册 | | 模块编号 | 01 |
| 模块描述 | 对需要注册的用户提供注册服务。 | | | |
| 调用模块 | 数据库模块 | | | |
| 被调用模块 | 无 | | | |
| 输入 | 项目 | 身份证号、姓名、电话、性别、密码、邮箱 | | |
| 格式 | 身份证号：varchar姓名：varchar电话：varchar性别：boolean密码：varchar确认密码：varchar 邮箱：varchar | | |
| 处理 | 1、用户输入身份证号、姓名、电话、性别、密码、确认密码  、邮箱  2、进行邮箱验证，验证码匹配后进入下一步  3、模块获得身份证号、姓名、电话、性别、密码、邮箱等信息  4、模块将信息并录入数据库 | | | |
| 输出 | 用户注册成功/失败。 | | | |
| 局部数据元素 | 数据库表 | | | |
| 注释 | 需要数据库信息表 | | | |

（3）主要算法

【注册】按钮：验证用户注册信息的合法性，合法则弹出注册成功页面，并将用户信息录入数据库。

【返回登录】按钮：弹出用户登录功能窗口。

### 3.3.2用户登录主页模块

(1)功能描述

对登录系统的用户进行身份验证。

(2)模块IPO表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 系统名称 | 列车票务管理系统 | | | |
| 模块名称 | 用户登录 | | 模块编号 | 02 |
| 模块描述 | 对登录系统的用户进行身份验证 | | | |
| 调用模块 | 数据库模块 | | | |
| 被调用模块 | 无 | | | |
| 输入 | 项目 | 身份证号，密码 | | |
| 格式 | 身份证号：varchar密码：varchar | | |
| 处理 | 1. 用户输入身份证号、密码、验证码 2. 检验验证码，验证码正确则进入下面的步骤 3. 模块获得身份证号、密码等信息   4、模块调用数据库表进行比较校验  5、模块返回登录信息 | | | |
| 输出 | 用户中心界面/用户登录失败。 | | | |
| 局部数据元素 | 数据库表 | | | |
| 注释 | 需要数据库信息表 | | | |

（3）主要算法

【登录】按钮：验证用户的合法性。

【注册】按钮：弹出用户注册功能窗口。

### 3.3.3用户查询个人信息功能模块

(1)功能描述

用于用户查看自己的个人信息。

(2)模块IPO表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 系统名称 | 列车票务管理系统 | | | |
| 模块名称 | 用户查询个人信息 | | 模块编号 | 03 |
| 模块描述 | 用于用户对自己信息进行查询 | | | |
| 调用模块 | 数据库模块 | | | |
| 被调用模块 | 无 | | | |
| 输入 | 项目 | 无 | | |
| 格式 | 无 | | |
| 处理 | 1、用户登录系统进入用户界面   1. 用户点击个人资料查看个人资料 2. 根据用户名调用后端数据库 3. 前端界面呈现数据库中用户表对应的的用户信息 | | | |
| 输出 | 用户的身份证号、姓名、电话、性别、密码、邮箱 | | | |
| 局部数据元素 | 数据库表 | | | |
| 注释 | 需要数据库信息表 | | | |

(3)主要算法

【修改】按钮：进入修改个人资料界面。

### 3.3.4用户管理个人信息功能模块

(1)功能描述

用于用户修改自己信息。

(2)模块IPO表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 系统名称 | 列车票务管理系统 | | | |
| 模块名称 | 用户管理个人信息 | | 模块编号 | 04 |
| 模块描述 | 用于用户对自己信息进行修改 | | | |
| 调用模块 | 数据库模块 | | | |
| 被调用模块 | 无 | | | |
| 输入 | 项目 | 用户密码，电话，邮箱 | | |
| 格式 | 用户密码：varchar 电话：varchar 邮箱：varchar | | |
| 处理 | 1、用户点击修改个人资料。   1. 用户可以选择修改电话、密码和邮箱。 2. 用户如果修改邮箱，将进行邮箱验证操作，邮箱验证成功后，进入下一步；如果修改的信息不包含邮箱，跳过此步骤 3. 用户提交更改后的自身信息 4. 系统接收信息写入数据库中 5. 系统弹出更新成功对话框 | | | |
| 输出 | 用户资料修改成功/失败对话框 | | | |
| 局部数据元素 | 数据库表 | | | |
| 注释 | 需要数据库信息表 | | | |

(3)主要算法

【提交】按钮：用户提交更改的信息，将新的个人资料同步更新至数据库。

### 3.3.5用户查阅关联用户模块

(1)功能描述

用于用户查看其关联用户。

(2)模块IPO表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 系统名称 | 列车票务管理系统 | | | |
| 模块名称 | 查看关联用户 | | 模块编号 | 05 |
| 模块描述 | 用于用户查看其关联用户。 | | | |
| 调用模块 | 数据库模块 | | | |
| 被调用模块 | 用户管理关联用户功能模块 | | | |
| 输入 | 项目 | 无 | | |
| 格式 | 无 | | |
| 处理 | 1、用户点击关联旅客模块查看关联旅客资料   1. 模块根据用户信息调用后端数据库 2. 前端界面呈现其关联旅客及信息 | | | |
| 输出 | 关联旅客的姓名，身份证号，电话，性别 | | | |
| 局部数据元素 | 数据库表 | | | |
| 注释 | 需要数据库信息表 | | | |

（3）主要算法

【关联旅客】按钮：查看已关联旅客的资料。

### 3.3.6用户管理关联旅客功能模块

(1)功能描述

用户变更（添加或取消）关联旅客

(2)模块IPO表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 系统名称 | 列车票务管理系统 | | | |
| 模块名称 | 管理关联旅客 | | 模块编号 | 06 |
| 模块描述 | 用户变更（添加或取消）关联用户 | | | |
| 调用模块 | 数据库模块 | | | |
| 被调用模块 | 无 | | | |
| 输入 | 项目 | 关联乘客的身份证号，姓名，电话，性别 | | |
| 格式 | 关联乘客的身份证号：varchar姓名：varchar电话：varchar性别：boolean | | |
| 处理 | 1、用户选择需要取消的关联乘客或者输入需要关联的乘客的身份证号，姓名，电话，性别  2、模块获得关联乘客的身份证号，姓名，电话，性别  3、模块将关联乘客信息删除或录入数据库 | | | |
| 输出 | 用户关联成功/失败。 | | | |
| 约束条件 | 列车号不能重复 | | | |
| 注释 | 需要数据库信息表 | | | |

（3）主要算法

【关联】按钮：将关联信息录入数据库。

【取消关联】按钮：将关联信息从数据库中删除

### 3.3.7用户搜索并查看车次功能模块

(1)功能描述

用于用户搜索并查看车次信息

(2)模块IPO表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 系统名称 | 列车票务管理系统 | | | |
| 模块名称 | 用户搜索并查看车次 | | 模块编号 | 07 |
| 模块描述 | 用于用户搜索并查看车次信息 | | | |
| 调用模块 | 数据库模块 | | | |
| 被调用模块 | 用户下单功能模块，用户改签或退票模块 | | | |
| 输入 | 项目 | 出发日期，上车站，下车站 | | |
| 格式 | 出发日期：date，上车站：varchar，下车站：varchar。 | | |
| 处理 | 1. 进入订票系统 2. 用户输入出发城市和到达城市并点击搜索 3. 模块调用数据库中的列车车次表 4. 前端界面呈现符合检索条件的列车信息 | | | |
| 输出 | 列车号，出发日期，出发时间，剩余座位数，上车站，下车站，价格 | | | |
| 约束条件 | 无 | | | |
| 注释 | 需要数据库信息表 | | | |

(3)主要算法

【订票】按钮：跳转至订票页面。

【搜索】按钮：查看对应城市车次信息。

### 3.3.8用户下单功能模块

(1)功能描述

用于用户提交购票订单信息。

(2)模块IPO表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 系统名称 | 列车票务管理系统 | | | |
| 模块名称 | 用户下单购票 | | 模块编号 | 08 |
| 模块描述 | 用于用户提交购票订单。 | | | |
| 调用模块 | 数据库模块，用户搜索并查看车次功能模块 | | | |
| 被调用模块 | 用户改签或退票模块 | | | |
| 输入 | 项目 | 列车号，出发时间，乘车人 | | |
| 格式 | 列车号：varchar，出发时间：time，乘车人：varchar | | |
| 处理 | 1. 用户在列车车次信息界面选择对应列车和出发时间，并勾选乘车人 2. 用户提交订单 3. 模块将新订单同步更新至后端数据库 | | | |
| 输出 | 提交成功/失败。 | | | |
| 约束条件 | 无 | | | |
| 注释 | 需要数据库信息表 | | | |

(3)主要算法

【提交订单】按钮：将订单及信息同步更新至数据库。

### 3.3.9用户对订单付款或取消功能模块

(1)功能描述

用于用户为已提交订单中的车票付款或取消。

(2)模块IPO表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 系统名称 | 列车票务管理系统 | | | |
| 模块名称 | 用户付款或取消订单 | | 模块编号 | 09 |
| 模块描述 | 用于用户为已提交订单中的车票付款或取消订单 | | | |
| 调用模块 | 数据库模块 | | | |
| 被调用模块 | 无 | | | |
| 输入 | 项目 | 订单号 | | |
| 格式 | 订单号：varchar。 | | |
| 处理 | 1. 用户进入查看个人订单界面 2. 用户选择需要付款或取消的订单号 3. 用户付款或确定取消 4. 模块将订单状态同步更新至后端数据库 | | | |
| 输出 | 支付成功or失败or 取消订单成功 | | | |
| 约束条件 | 无 | | | |
| 注释 | 需要数据库信息表 | | | |

(3)主要算法

【付款】按钮：用户为已提交订单中的车票支付。

【取消】按钮：用户取消已提交的订单。

### 3.3.10用户查看个人车票模块

(1)功能描述

用于用户查看个人购买的车票列表。

(2)模块IPO表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 系统名称 | 列车票务管理系统 | | | |
| 模块名称 | 用户查看个人车票 | | 模块编号 | 10 |
| 模块描述 | 用于用户查看个人购买的车票列表 | | | |
| 调用模块 | 数据库模块 | | | |
| 被调用模块 | 用户改签或退票模块 | | | |
| 输入 | 项目 | 无 | | |
| 格式 | 无 | | |
| 处理 | 1、点击个人车票列表  2、模块根据用户的用户信息调用后端数据库  3. 前端界面呈现个人购票信息 | | | |
| 输出 | 本人已购车票的信息：  车票编号，列车号，出发时间，座位号，上车站，下车站，持票人身份证号码，价格，是否付款 | | | |
| 约束条件 | 无 | | | |
| 注释 | 需要数据库信息表 | | | |

(3)主要算法

【改签】按钮：用户修改车票信息

【退票】按钮：用户对已有车票进行退订

### 3.3.11用户改签或退票模块

(1)功能描述

用于用户改签或退订已有的车票

(2)模块IPO图

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 系统名称 | 列车票务管理系统 | | | |
| 模块名称 | 用户改签或退票模块 | | 模块编号 | 11 |
| 模块描述 | 用于用户改签或退订已有的车票 | | | |
| 调用模块 | 数据库模块、用户下单功能模块 | | | |
| 被调用模块 | 无 | | | |
| 输入 | 项目 | 列车号 | | |
| 格式 | 列车号：varchar | | |
| 处理 | 1、用户选择改签或退订按钮  4、若用户选择改签则需重新进行下单购票，否则跳过  5、用户确认新变更的车票信息或退票  6、系统弹出改签或退订成功的对话框 | | | |
| 输出 | 改签或退票成功的对话框 | | | |
| 约束条件 | 无 | | | |
| 注释 | 需要数据库信息表 | | | |

(3)主要算法

【确认】按键：确认新变更的车票信息并同步到后端数据库

【取消】按键：关闭改签或退票界面

### 3.3.12管理员登录模块

(1)功能描述

对登录系统的管理员进行身份验证

(2)模块IPO表

|  |  |  |  |
| --- | --- | --- | --- |
| 系统名称 | 列车票务管理系统 | | |
| 模块名称 | 管理员登录 | 模块编号 | 12 |
| 模块描述 | 用于系统对管理员的登录进行身份验证 | | |
| 调用模块 | 数据库模块 | | |
| 被调用模块 | 无 | | |
| 输入 | 项目 | 管理员用户名，密码 | |
| 格式 | 管理员用户名：varchar，密码：varchar | |
| 处理 | 1、管理员输入账号、密码  2、模块获得账号、密码等信息  3、模块调用数据库表进行比较校验  4、模块返回登录信息 | | |
| 输出 | 管理员登录成功或登录失败。 | | |
| 局部数据元素 | 数据库表 | | |
| 约束条件 | 验证码 | | |
| 注释 | 需要数据库管理员信息表 | | |

(3)主要算法

【登录】按钮：验证管理员的合法性

### 3.3.13管理员查看列车车次模块

(1)功能描述

管理员查看现有列车车次

(2)模块IPO图

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 系统名称 | 列车票务管理系统 | | | |
| 模块名称 | 管理员查看列车车次信息 | | 模块编号 | 13 |
| 模块描述 | 用于管理员查看现有列车车次 | | | |
| 调用模块 | 数据库模块 | | | |
| 被调用模块 | 管理员管理列车车次信息模块、管理员添加列车信息模块 | | | |
| 输入 | 项目 | 无 | | |
| 格式 | 无 | | |
| 处理 | 1. 管理员登录进入管理员界面 2. 管理员点击查看列车车次信息按钮 3. 模块调用后台数据库中列车车次表的信息 4. 模块将获得的列车车次信息呈现在前端界面 | | | |
| 输出 | 各现有列车车次的列车号，始发站，终点站，途径站，发车时间，座位数，票价等信息 | | | |
| 约束条件 | 无 | | | |
| 注释 | 需要数据库信息表 | | | |

(3)主要算法

【查询】按键：进入查询列车车次信息界面查看列车车次信息

【取消】按键：关闭查询界面

【管理列车车次】按键：跳转管理列车车次信息模块

【添加列车车次】按键：跳转添加列车车次信息模块

### 3.3.14管理员管理列车车次信息模块

(1)功能描述

管理员修改或删除当前已有列车车次的信息

(2)模块IPO图

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 系统名称 | 列车票务管理系统 | | | |
| 模块名称 | 管理员管理列车车次信息 | | 模块编号 | 14 |
| 模块描述 | 用于管理员修改或删除系统中已有列车车次信息 | | | |
| 调用模块 | 数据库模块 | | | |
| 被调用模块 | 无 | | | |
| 输入 | 项目 | 列车号 | | |
| 格式 | 列车号：varchar | | |
| 处理 | 1、管理员登录进入管理员界面  2、管理员在搜索框输入要修改或删除列车的列车号  3、管理员修改或删除列车车次信息，点击修改或删除车次信息按钮  4、系统弹出修改或删除成功的对话框 | | | |
| 输出 | 修改或删除成功的对话框 | | | |
| 约束条件 | 列车号无法修改 | | | |
| 注释 | 需要数据库信息表 | | | |

(3)主要算法

【修改】按键：确认新变更的列车车次信息并同步到后端数据库

【删除】按键：删除选中的列车车次信息并同步到后端数据库

【取消】按键：关闭列车信息管理界面

### 3.3.15管理员添加列车信息模块

(1)功能描述

管理员添加新的列车车次信息

(2)模块IPO图

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 系统名称 | 列车票务管理系统 | | | |
| 模块名称 | 管理员添加列车车次信息 | | 模块编号 | 15 |
| 模块描述 | 用于管理员添加系统中没有的列车车次信息 | | | |
| 调用模块 | 数据库模块 | | | |
| 被调用模块 | 无 | | | |
| 输入 | 项目 | 列车号，始发站，终点站，途径站，发车时间，座位数 | | |
| 格式 | 列车号：varchar，始发站：varchar，终点站：varchar，途径站：varchar，发车时间：time，座位数：tinyint unsigned | | |
| 处理 | 1、管理员登录进入管理员界面  2、管理员选择添加列车信息按钮  3、系统跳转至添加列车信息界面  4、管理员输入要添加列车的列车号等列车基本信息  5、管理员添加列车信息，点击确认  6、系统弹出列车信息添加成功对话框 | | | |
| 输出 | 添加成功对话框 | | | |
| 约束条件 | 列车号不能重复 | | | |
| 注释 | 需要数据库信息表 | | | |

(3)主要算法

【添加】按键：进入添加列车信息界面

【确认】按键：确认添加的列车信息

【取消】按键：关闭列车信息添加界面

### 3.3.16管理员查阅车票功能模块

(1)功能描述

管理员查看用户车票

(2)模块IPO图

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 系统名称 | 列车票务管理系统 | | | |
| 模块名称 | 管理员查阅车票 | | 模块编号 | 16 |
| 模块描述 | 用于管理员查看用户车票 | | | |
| 调用模块 | 数据库模块 | | | |
| 被调用模块 | 管理员管理车票功能模块、管理员添加用户车票模块 | | | |
| 输入 | 项目 | 无 | | |
| 格式 | 无 | | |
| 处理 | 1、管理员登录进入订单界面   1. 点击查询车票按钮 2. 系统跳转查看车票信息界面 3. 模块调用后台数据库中车票表的信息 4. 模块将获得的车票信息呈现在前端界面 | | | |
| 输出 | 各已有车票编号，列车号，出发时间，座位号，上车站，下车站，持票人身份证号码，价格，是否付款 | | | |
| 约束条件 | 无 | | | |
| 注释 | 需要车票表 | | | |

(3)主要算法

【查询】按键：进入查询用户车票信息界面查看用户车票信息

【取消】按键：关闭查询界面

【管理用户车票】按键：跳转管理用户车票信息模块

【添加用户车票】按键：跳转添加用户车票模块

### 3.3.17管理员管理车票功能模块

(1)功能描述

管理员帮助用户改签或退订当前已有车票

(2)模块IPO图

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 系统名称 | 列车票务管理系统 | | | |
| 模块名称 | 管理员管理车票功能模块 | | 模块编号 | 17 |
| 模块描述 | 用于管理员帮助用户改签或退订系统中已有车票  信息 | | | |
| 调用模块 | 数据库模块 | | | |
| 被调用模块 | 无 | | | |
| 输入 | 项目 | 车票编号 | | |
| 格式 | 车票编号：varchar | | |
| 处理 | 1、管理员登录进入订单界面  2、管理员选择改签或退订车票按钮  3、管理员在搜索框输入要改签或退订的车票编号  4、管理员确认修改改签或删除的车票信息  5、系统弹出改签或删除成功的对话框 | | | |
| 输出 | 改签或删除成功的对话框 | | | |
| 约束条件 | 无 | | | |
| 注释 | 需要数据库信息表 | | | |

(3)主要算法

【确认】按键：确认新变更的车票信息并同步到后端数据库

【取消】按键：关闭车票信息管理界面

### 3.3.18管理员添加用户车票模块

(1)功能描述

管理员添加新的用户车票

(2)模块IPO图

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 系统名称 | 列车票务管理系统 | | | |
| 模块名称 | 管理员添加列车车次信息 | | 模块编号 | 18 |
| 模块描述 | 用于管理员帮助用户购票 | | | |
| 调用模块 | 数据库模块 | | | |
| 被调用模块 | 无 | | | |
| 输入 | 项目 | 列车号，出发日期，出发时间，上车站，下车站，持票人身份证号码 | | |
| 格式 | 列车号：varchar，出发日期：date，出发时间：time，上车站：varchar，下车站：varchar，持票人身份证号码：varchar | | |
| 处理 | 1、管理员登录进订单界面  2、管理员选择添加用户车票按钮  3、系统跳转至添加用户车票界面  4、管理员输入要添加车票的基本信息  5、管理员添加车票，点击确认  6、系统弹出车票信息添加成功对话框 | | | |
| 输出 | 添加车票成功对话框 | | | |
| 约束条件 | 无 | | | |
| 注释 | 需要数据库信息表 | | | |

(3)主要算法

【添加】按键：进入添加车票界面

【确认】按键：确认添加的车票

【取消】按键：关闭车票添加界面

### 3.3.19管理员查阅用户信息模块

(1)功能描述

管理员查看现有用户信息

(2)模块IPO表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 系统名称 | 列车票务管理系统 | | | |
| 模块名称 | 管理员查阅用户信息模块 | | 模块编号 | 19 |
| 模块描述 | 管理员查看现有用户信息 | | | |
| 调用模块 | 数据库模块 | | | |
| 被调用模块 | 管理员管理用户信息功能模块 | | | |
| 输入 | 项目 | 无 | | |
| 格式 | 无 | | |
| 处理 | 1、管理员登录进入管理员界面  2、管理员点击查看用户信息按钮  3、系统跳转至查看用户信息界面  4、模块调用后台数据库中用户表的信息  5、模块将获得的用户信息呈现在前端界面 | | | |
| 输出 | 各个用户的身份证号、电话、性别、车票编号等信息。 | | | |
| 局部数据元素 | 数据库表 | | | |
| 注释 | 需要数据库信息表 | | | |

### 3.3.20管理员管理用户信息功能模块

(1)功能描述

管理员修改或删除当前系统中的用户信息

(2)模块IPO图

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 系统名称 | 列车票务管理系统 | | | |
| 模块名称 | 管理员管理用户信息 | | 模块编号 | 20 |
| 模块描述 | 用于管理员修改或删除系统中当前系统中的用户信息 | | | |
| 调用模块 | 数据库模块 | | | |
| 被调用模块 | 无 | | | |
| 输入 | 项目 | 用户身份证号 | | |
| 格式 | 用户身份证号：varchar | | |
| 处理 | 1、管理员登录进入管理员界面  2、管理员选择修改或删除用户信息按钮  3、管理员在搜索框输入要修改或删除用户的身份证号或点击用户显示用户基本信息  4、管理员修改或删除用户信息，点击确认  5、系统弹出修改或删除成功的对话框 | | | |
| 输出 | 修改或删除成功的对话框 | | | |
| 约束条件 | 用户身份证号无法修改 | | | |
| 注释 | 需要数据库信息表 | | | |

(3)主要算法

【确认】按键：确认新变更的列车车次信息并同步到后端数据库

【取消】按键：关闭列车信息管理界面

# 4 详细设计

详细设计阶段的根本目标是确定应该怎样具体地实现所要求的系统，即不仅在逻辑上正确地实现每个模块的功能，更重要的是设计出的处理过程应该尽可能简明易懂。描述程序处理过程的工具称为过程设计的工具，它们可以分为程序流程图、盒图、问题分析图（PAD）图等，由于这些工具各有其优缺点，我们将在下面使用各种不同的过程设计工具对各个模块进行详细设计。

## 4.1用户注册模块

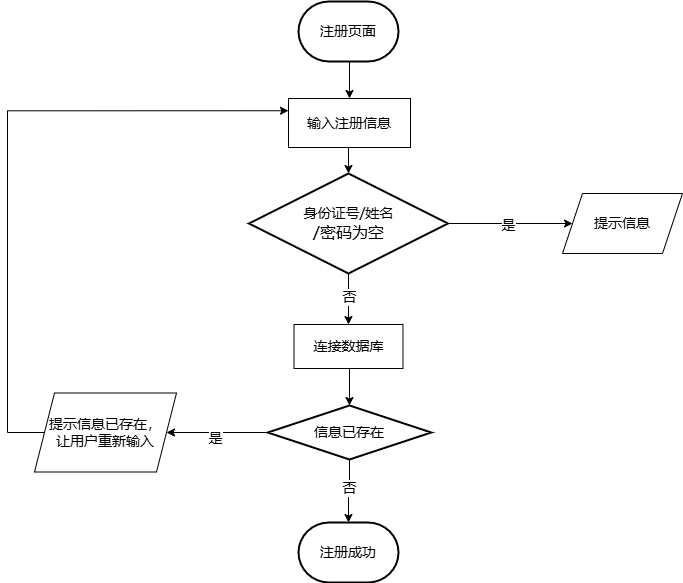


图4.1用户注册流程图

**伪代码设计**

Begin

输入身份证号+姓名+电话+性别+密码+邮箱；

If(身份证号/姓名/密码==””)则给出相应提示信息；

Else连接数据库

If(信息已存在)则提示信息已存在，让用户重新输入

Else提示注册成功，进入主界面

End

## 4.2用户登录模块

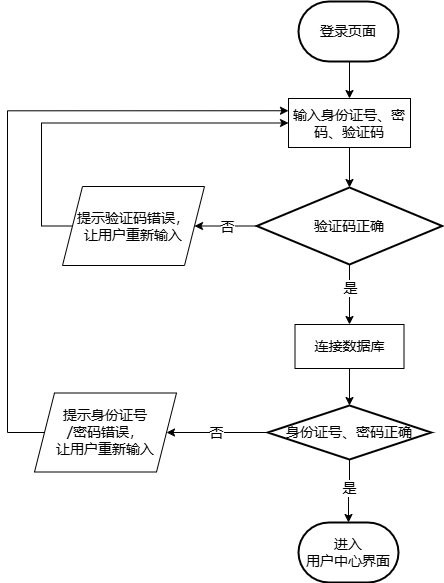


图4.2用户登录流程图

**伪代码设计**

Begin

输入身份证号、密码、验证码；

If(验证码错误)则给出相应提示信息，让用户重新输入；

Else连接数据库

If(身份证号/密码错误)则给出相应提示信息，让用户重新输入；

Else提示登录成功，进入用户中心界面

End

## 4.3用户查询个人信息模块

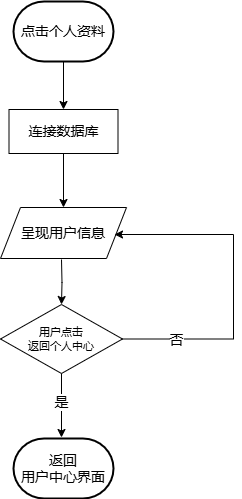


图4.3用户查询个人信息功能流程图

**伪代码设计**

Begin

用户点击个人资料；

连接至数据库；

显示用户信息；

If（用户点击返回）

Return 用户中心界面；

End

## 4.4用户管理个人信息模块

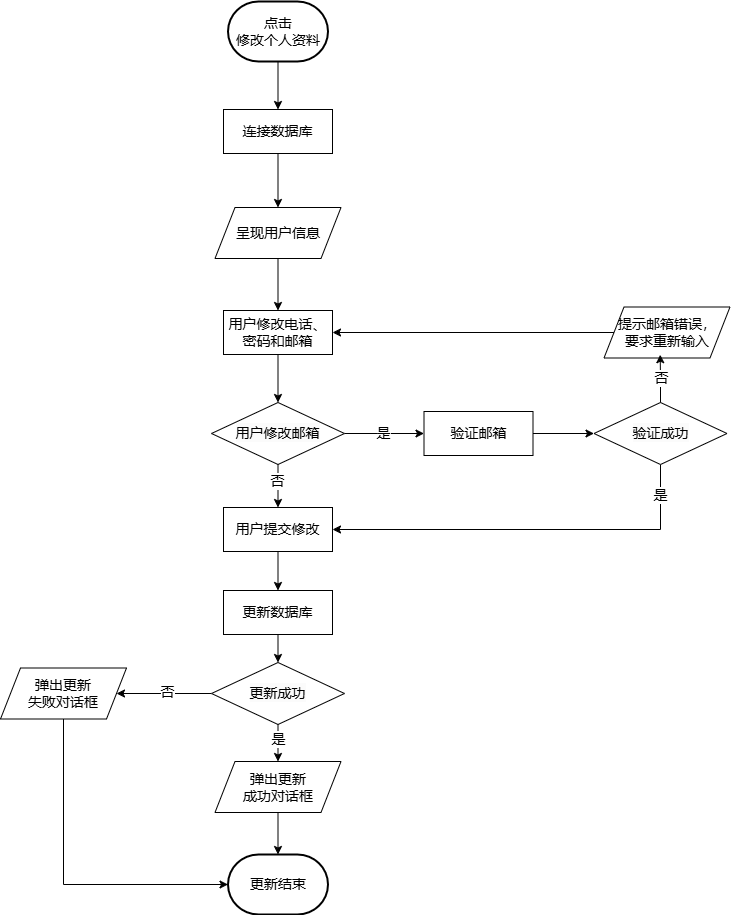


图4.4用户管理个人信息流程图

**伪代码设计**

Begin

用户点击修改个人资料；

连接至数据库；

显示用户信息；

用户修改电话、密码和邮箱；

If（用户修改邮箱）

验证邮箱；

if（邮箱验证失败） 提示用户重新输入；

用户提交更改；

更新数据库；

if（更新成功）

弹出更新成功对话框；

else

弹出更新失败对话框；

用户返回修改电话、密码和邮箱；

End

## 4.5用户查看关联旅客模块

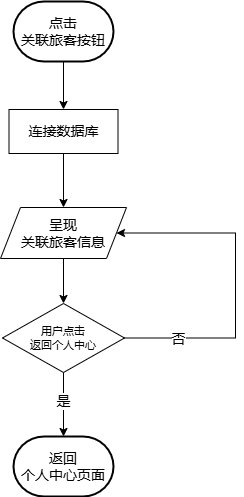


图4.5 用户查看关联用户流程图

**伪代码设计**

Begin

用户点击关联旅客；

连接至数据库；

显示关联旅客信息；

If（用户点击返回）

Return 用户中心界面；

End

## 4.6用户管理关联旅客模块

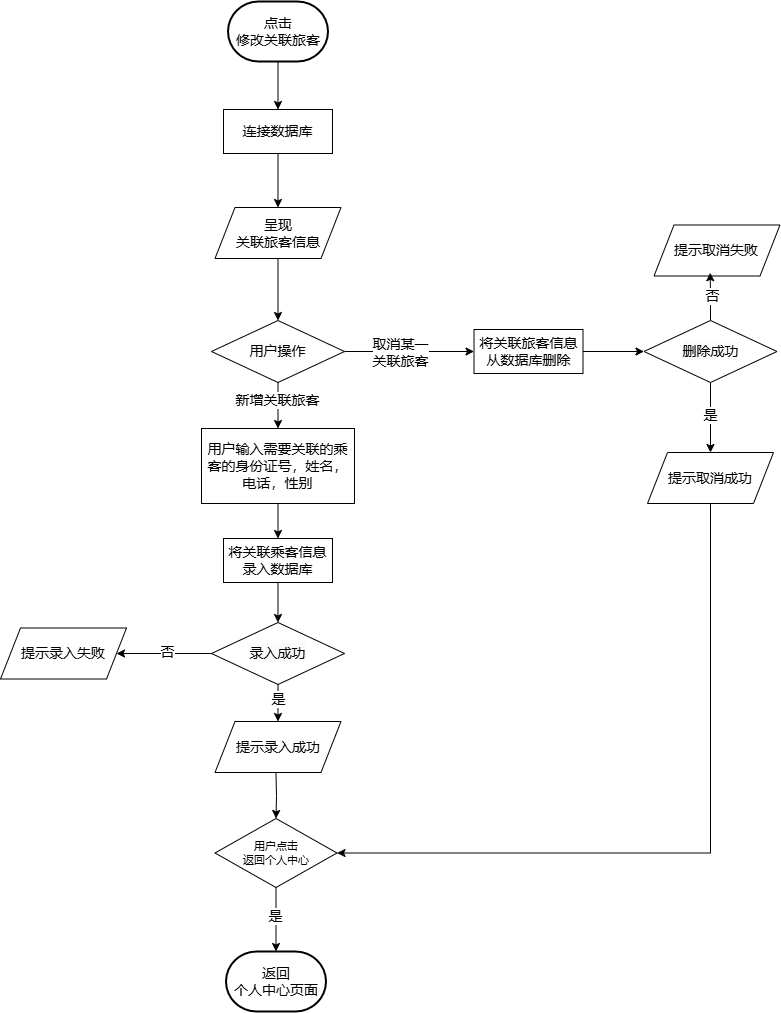


图4.6 用户管理关联用户流程图

**伪代码设计**

Begin

用户点击修改关联旅客；

连接至数据库；

显示关联旅客信息；

If（用户点击关联）

获取用户输入需要关联的乘客的身份证号，姓名，电话，性别；

将关联乘客信息录入数据库；

if（录入成功）

弹出录入成功对话框；

else

弹出录入失败对话框；

If（用户点击取消关联）

获取用户要取消的关联旅客信息；

将该关联从数据库中删除；

if（删除成功）

弹出取消成功对话框；

else

弹出取消失败对话框；

If（用户点击返回）

Return 用户中心界面；

End

## 4.7用户搜索并查看车次模块

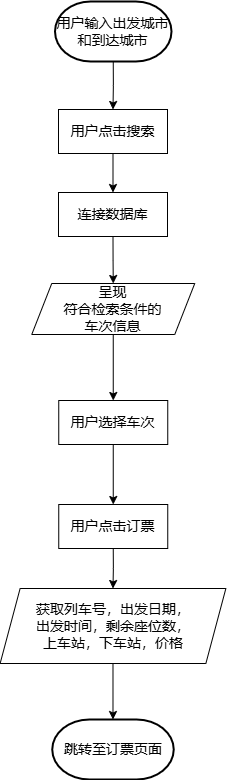


图4.7用户搜索并查看车次流程图

**伪代码设计**

Begin

用户进入订票系统；

If（用户点击搜索）

获取用户输入出发城市和到达城市；

连接数据库；

呈现符合检索条件的车次信息和剩余座位数；

If（用户点击订票）

获取用户选择的列车号，出发日期，出发时间，座位类型，上车站，下车 站，价格；

跳转订票界面；

End

## 4.8用户下单购票模块

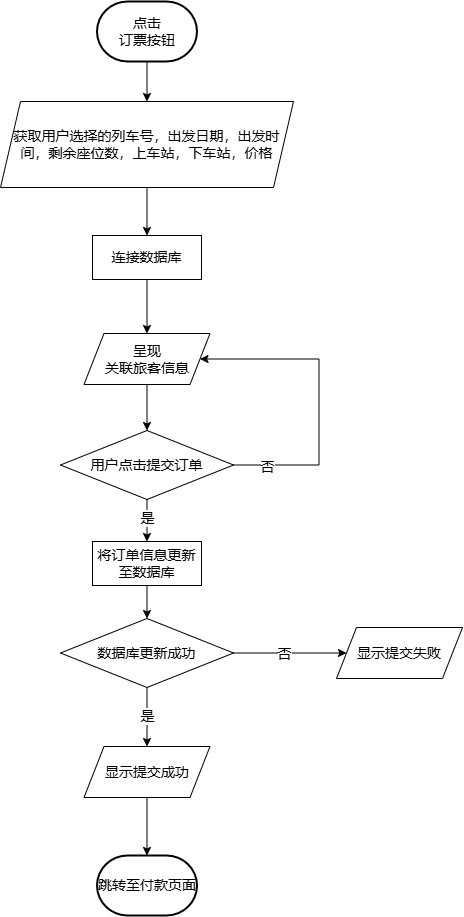


图4.8用户下单购票流程图

**伪代码设计**

Begin

获取用户选择的列车号，出发日期，出发时间，座位类型，上车站，下车站，价格；

连接数据库；

显示关联旅客；

If（用户点击提交订单）

将订单信息更新至数据库；

显示提交成功/失败；

If（提交成功）

跳转至付款页面；

End

## 4.9用户付款或取消订单模块

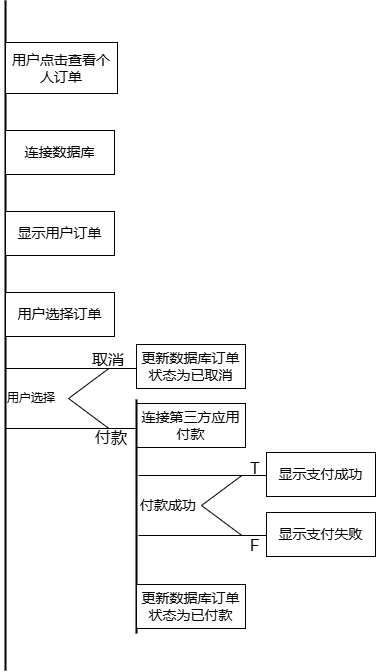


图4.9用户付款或取消订单PAD图

**伪代码设计**

Begin

用户点击查看个人订单；

连接数据库；

显示用户未付款订单；

用户选择订单；

If（用户点击付款）

连接第三方应用付款；

If（付款成功）

显示支付成功；

else 显示支付失败；

更新数据库订单状态；

If（用户点击取消）

设置数据库订单状态为已取消；

End

## 4.10用户查看个人车票模块

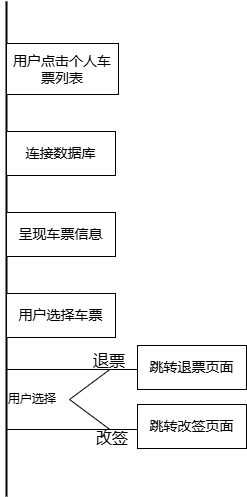


图4.10 用户查看个人车票PAD图

**伪代码设计**

Begin

用户点击个人车票列表；

连接数据库；

呈现车票信息；

用户选择车票；

If（用户点击改签按钮）

跳转改签页面；

If（用户点击退票按钮）

跳转退票页面；

End

## 4.11用户改签或退票模块

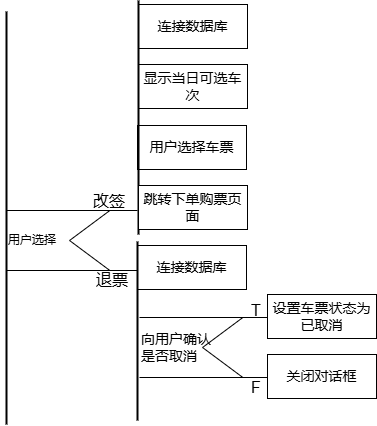


图4.11用户改签或退票模块PAD图

**伪代码设计**

Begin

If（用户点击改签按钮）

连接数据库；

显示当日可选车次；

If（用户点击确认）

设置数据库中用户拥有的该车票状态为已取消；

调用第三方应用退还金额；

跳转用户下单购票页面；

If（用户点击退票按钮）

连接数据库；

发送对话框向用户确认是否取消；

If（用户点击确认）

设置数据库中用户拥有的该车票状态为已取消；

调用第三方应用退还金额；

End

## 4.12管理员登录模块

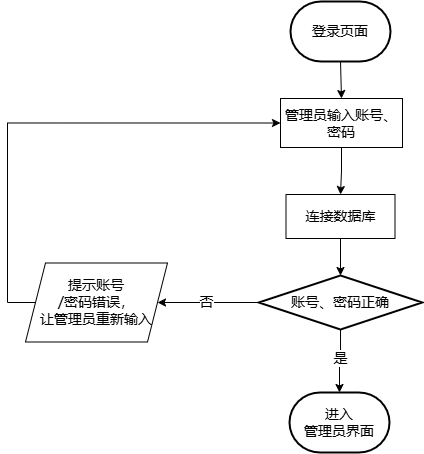


图4.12管理员登录流程图

**伪代码设计**

Begin

管理员输入账号、密码；

连接数据库；

If(账号/密码错误)则给出相应提示信息，让管理员重新输入；

Else提示登录成功，进入管理员界面；

End

## 4.13管理员查看列车车次信息模块

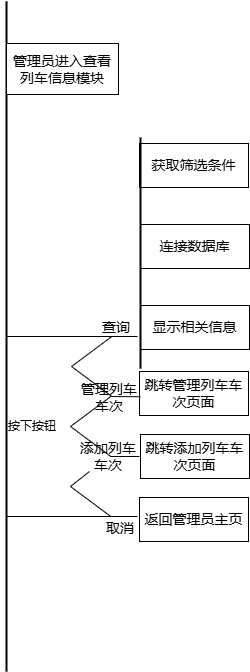


图4.13管理员查看列车车次信息PAD图

**伪代码设计**

Begin

管理员进入查看列车信息模块；

While（按下按钮）{

Switch（按钮）{

case 查询：

获取筛选条件；

连接数据库，筛选符合条件车次；

显示相关信息；

break；

case 管理列车车次：

跳转管理列车车次页面；

break；

case 添加列车车次：

跳转添加列车车次页面；

break；

case 取消：

返回管理员主页；

}

}

End

## 4.14管理员管理列车车次信息模块

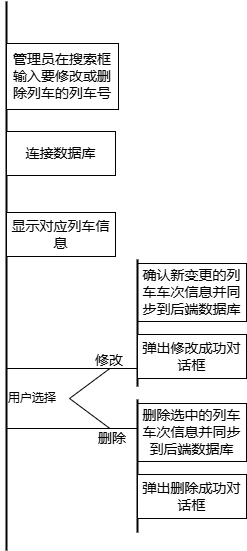


图4.14管理员管理列车车次信息PAD图

**伪代码设计**

Begin

管理员进入管理列车车次界面；

管理员在搜索框输入要修改或删除列车的列车号；

连接数据库；

显示对应列车信息；

if(点击修改)

确认新变更的列车车次信息并同步到后端数据库；

弹出修改成功对话框；

Else if（点击删除）

删除选中的列车车次信息并同步到后端数据库；

弹出删除成功对话框；

End

## 4.15管理员添加列车信息模块

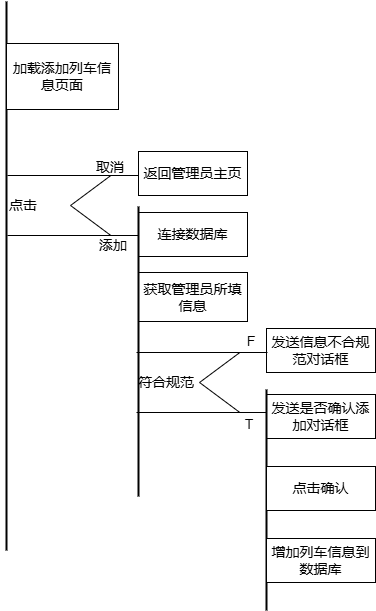


图4.15管理员添加列车信息PAD图

**伪代码设计**

Begin

进入添加列车信息页面；

If(点击添加)

{

连接数据库；

获取管理员所填信息；

校对格式以及信息是否符合规范；

If（符合规范）

发送是否确认添加对话框；

If（点击确认）

增加列车信息到数据库；

else 发送信息不合规范对话框；

}

If（点击取消）

返回管理员主页；

End

## 4.16管理员查阅车票模块

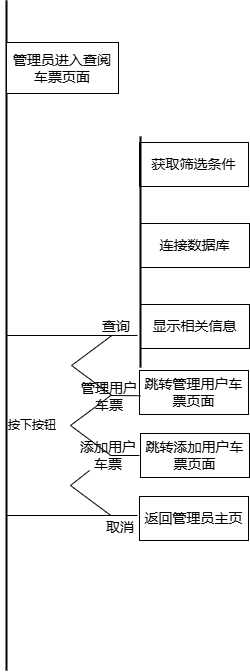


图4.16管理员查阅车票PAD图

**伪代码设计**

Begin

进入查阅车票页面；

While（按下按钮）{

Switch（按钮）{

case 查询：

获取筛选条件；

连接数据库，筛选符合条件车票；

显示相关信息；

break；

case 管理用户车票：

跳转管理用户车票页面；

break；

case 添加用户车票：

跳转添加用户车票页面；

break；

case 取消：

返回管理员主页；

}

}

End

## 4.17管理员管理车票模块

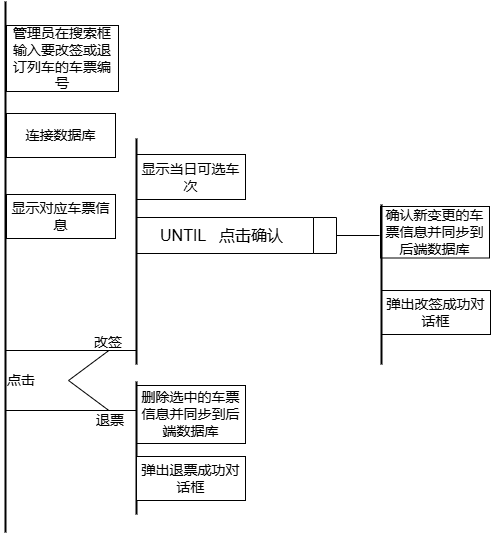


图4.17管理员管理车票PAD图

**伪代码设计**

Begin

管理员进入管理车票界面；

管理员在搜索框输入要改签或退订列车的车票编号；

连接数据库；

显示对应车票信息；

if(点击改签)

显示当日可选车次；

If（点击确认）

确认新变更的车票信息并同步到后端数据库；

弹出改签成功对话框；

Else if（点击退票）

删除选中的车票信息并同步到后端数据库；

弹出退票成功对话框；

End

## 4.18管理员添加用户车票模块

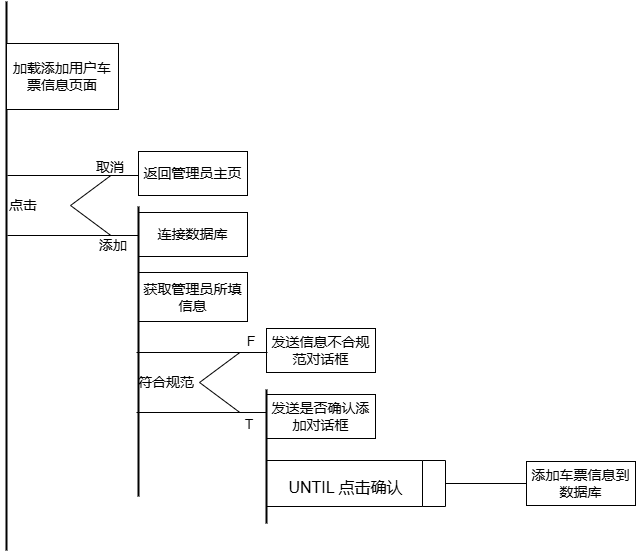


图4.18管理员添加用户车票PAD图

**伪代码设计**

Begin

进入添加用户车票信息页面；

If(点击添加)

{

连接数据库；

获取管理员所填信息；

校对格式以及信息是否符合规范；

If（符合规范）

发送是否确认添加对话框；

If（点击确认）

添加车票信息到数据库；

else 发送信息不合规范对话框；

}

If（点击取消）

返回管理员主页；

End

## 4.19管理员查阅用户信息模块

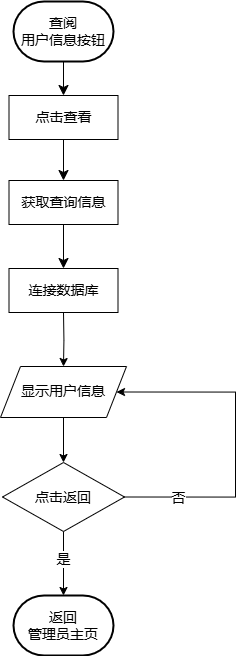


图4.19管理员查阅用户信息流程图

**伪代码设计**

Begin

进入管理员界面；

点击查阅用户信息按钮；

If（点击查看）

获取查询信息；

连接数据库；

显示用户信息；

End

## 4.20管理员管理用户信息模块

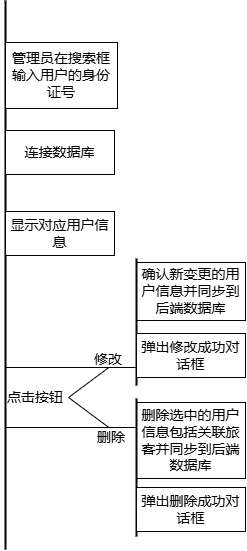


图4.20管理员管理用户信息模块PAD图

**伪代码设计**

Begin

管理员进入管理用户信息界面；

管理员在搜索框输入用户的身份证号；

连接数据库；

显示对应用户信息；

if(点击修改)

确认新变更的用户信息并同步到后端数据库；

弹出修改成功对话框；

Else if（点击删除）

删除选中的用户信息包括关联旅客并同步到后端数据库；

弹出删除成功对话框；

End

# 5 测试用例说明

在列车电子票务系统的测试用例说明中，我们可以使用红灯区和绿灯区的概念来帮助明确系统的功能范围和测试重点。具体来说：

1. **绿灯区 (Green Zone)**：

• 绿灯区代表系统中“已完成”且“已通过测试”的功能模块。在这些区域内的功能已经按照需求实现，并且经过了充分的验证，确保它们符合设计预期。

• 对于这些模块，我们可以认为它们是“安全的”，无需再进行重大修改或重新设计。测试用例的重点是在这些模块的日常操作和功能维护上，确保它们持续表现良好。

绿灯区的相关测试通过用例图将在下面每个小节进行具体展示，其中我们通过交了大部分比较基础的测试点。

2. **红灯区 (Red Zone)**：

• 红灯区则表示系统中“未完成”或“存在已知问题”的区域。在这些区域内的功能模块可能尚未完全实现，或者存在设计缺陷、性能瓶颈、错误等问题。

• 这些模块需要优先关注，测试用例的目标是尽可能覆盖这些区域的异常情况和边界条件，确保问题能够被及时发现和修复。

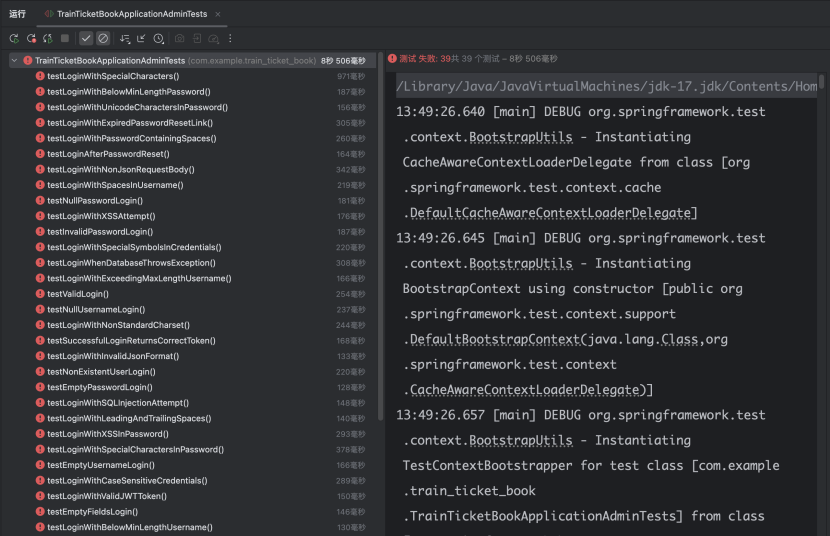


图5.1 TDD测试中的红灯区示例图

### 5.1 用户注册/登陆测试

对于登录进行测试，具体的测试用例必须覆盖以下组合：

① userId == null，password == null

② userId != null，password != null

③ userMapper.selectUserByUserId(userId) == null

④ userMapper.selectUserByUserId(userId) != null

⑤ authenticateUser(userId, password) == true

⑥ authenticateUser(userId, password) == false

具体的测试用例如下所示：

① userId == null，password == null 覆盖1

② userId == "12345"，password == "password123"，userMapper.selectUserByUserId("12345") != null，authenticateUser("12345", "password123") == true 覆盖2,4,5

③ userId == "12345"，password == "wrongPassword"，userMapper.selectUserByUserId("12345") != null，authenticateUser("12345", "wrongPassword") == false 覆盖2,4,6

④ userId == "99999"，password == "password123"，userMapper.selectUserByUserId("99999") == null 覆盖2,3

⑤ userId == "admin"，password == "adminPass"，userMapper.selectUserByUserId("admin") != null，authenticateUser("admin", "adminPass") == true，且账户已锁定，覆盖2,4,5

⑥ userId == "lockedUser"，password == "password123"，userMapper.selectUserByUserId("lockedUser") != null，authenticateUser("lockedUser", "password123") == false（由于账户被锁定），覆盖2,4,6

部分测试代码如下所示：

1. @Test
2. public void testUserIdNullLogin() {
3. User user = new User();
4. user.setpId(null);
5. user.setpPwd("password123");
6. boolean isValid = authenticateUser(user.getpId(), user.getpPwd());
7. assertFalse(isValid, "Login should fail if user ID is null");
8. }
9. @Test
10. public void testLockedAccountLogin() {
11. User user = new User();
12. user.setpId("12345");
13. user.setpPwd("password123");
14. boolean isAccountLocked = isAccountLocked(user.getpId());
15. assertTrue(isAccountLocked, "Login should fail if account is locked");
16. }
17. @Test
18. public void testSuccessfulLoginReturnsUserInfo() {
19. User user = new User();
20. user.setpId("12345");
21. user.setpPwd("password123");
22. boolean isValid = authenticateUser(user.getpId(), user.getpPwd());
23. if (isValid) {
24. User returnedUser = getUserInfo(user.getpId());
25. assertEquals(user.getpId(), returnedUser.getpId(), "User ID should match");
26. assertEquals(user.getpPwd(), returnedUser.getpPwd(), "Password should match");
27. } else {
28. fail("Login should be successful with valid credentials");
29. }
30. }
31. @Test
32. public void testIncorrectEmailFormatLogin() {
33. User user = new User();
34. user.setpId("12345");
35. user.setpPwd("password123");
36. user.setpMail("invalid-email-format");
37. boolean isValidEmail = isValidEmailFormat(user.getpMail());
38. assertFalse(isValidEmail, "Login should fail if the email format is invalid");
39. }
40. @Test
41. public void testUserAlreadyLoggedIn() {
42. User user = new User();
43. user.setpId("12345");
44. user.setpPwd("password123");
45. boolean isLoggedIn = isUserAlreadyLoggedIn(user.getpId());
46. assertTrue(isLoggedIn, "Login should fail if the user is already logged in");
47. }
48. @Test
49. public void testExpiredPasswordResetLink() {
50. User user = new User();
51. user.setpId("12345");
52. user.setpPwd("password123");
53. boolean isExpiredLink = isPasswordResetLinkExpired("expiredLink12345");
54. assertTrue(isExpiredLink, "Password reset should fail if the link is expired");
55. }



图5.2 TDD测试中的用户注册/登陆绿灯区图

### 5.2 列车信息查询测试

对于查询列车信息进行测试，具体的测试用例必须覆盖以下组合：

① trainSourcePlace == null，trainArrivalPlace == null，trainSourceDay == null

② trainSourcePlace != null，trainArrivalPlace != null，trainSourceDay != null

③ trainMapper.selectList(new LambdaQueryWrapper<Train>().eq(Train::getTrainSourcePlace, trainSourcePlace).eq(Train::getTrainArrivalPlace, trainArrivalPlace).eq(Train::getTrainSourceDay, trainSourceDay)) != null

④ trainMapper.selectList(new LambdaQueryWrapper<Train>().eq(Train::getTrainSourcePlace, trainSourcePlace).eq(Train::getTrainArrivalPlace, trainArrivalPlace).eq(Train::getTrainSourceDay, trainSourceDay)) == null

⑤ trainMapper.selectPage(new Page<>(pageNum, 10), new LambdaQueryWrapper<Train>().orderByAsc(Train::getTrainNumber)).getRecords().size() > 0

⑥ trainMapper.selectPage(new Page<>(pageNum, 10), new LambdaQueryWrapper<Train>().orderByAsc(Train::getTrainNumber)).getRecords().size() <= 0

⑦ trainMapper.insert(new Train(trainNumber, trainSourcePlace, trainArrivalPlace, trainSourceDay, trainSourceTime, trainSeats, trainDelete)) == 1

⑧ trainMapper.insert(new Train(trainNumber, trainSourcePlace, trainArrivalPlace, trainSourceDay, trainSourceTime, trainSeats, trainDelete)) == 0

⑨ trainMapper.deleteTrain(trainNumber) == 1

⑩ trainMapper.deleteTrain(trainNumber) == 0

⑪ javaMailSender.send(new SimpleMailMessage()) 成功

⑫ javaMailSender.send(new SimpleMailMessage()) 失败

⑬ trainMapper.getMaxId("BN") == null

⑭ trainMapper.getMaxId("BN") != null

具体的测试用例如下所示：

① trainSourcePlace == null，trainArrivalPlace == null，trainSourceDay == null 覆盖1

② trainSourcePlace == "北京"，trainArrivalPlace == "上海"，trainSourceDay == "2024-12-31"，trainMapper.selectList(new LambdaQueryWrapper<Train>().eq(Train::getTrainSourcePlace, "北京").eq(Train::getTrainArrivalPlace, "上海").eq(Train::getTrainSourceDay, "2024-12-31")) 返回 [train1, train2] 覆盖2, 3

③ trainSourcePlace == "广州"，trainArrivalPlace == "深圳"，trainSourceDay == "2024-12-31"，trainMapper.selectList(new LambdaQueryWrapper<Train>().eq(Train::getTrainSourcePlace, "广州").eq(Train::getTrainArrivalPlace, "深圳").eq(Train::getTrainSourceDay, "2024-12-31")) 返回空列表 覆盖2, 4

④ trainController.findAllTrainPage(1)，trainMapper.selectPage(new Page<>(1, 10), new LambdaQueryWrapper<Train>().orderByAsc(Train::getTrainNumber)).getRecords() 返回 [train1, train2] 覆盖5

⑤ trainController.findAllTrainPage(2)，trainMapper.selectPage(new Page<>(2, 10), new LambdaQueryWrapper<Train>().orderByAsc(Train::getTrainNumber)).getRecords() 返回空分页 覆盖6

⑥ trainController.addTrain(new Train("BN0001", "北京", "南京", "2024-12-31", "09:00:00", "10", "0")) 返回成功，trainMapper.insert(...) == 1 覆盖7

⑦ trainController.addTrain(new Train("BN0002", "北京", "南京", "2024-12-31", "10:00:00", "0", "1")) 返回失败，trainMapper.insert(...) == 0 覆盖8

⑧ trainController.deleteTrain("TR123") 返回成功，trainMapper.deleteTrain("TR123") == 1 覆盖9

⑨ trainController.deleteTrain("TR999") 返回失败，trainMapper.deleteTrain("TR999") == 0 覆盖10

⑩ trainController.sendEmail("user@example.com", "测试邮件", "这是邮件内容") 成功，javaMailSender.send(...) 成功 覆盖11

⑪ trainController.sendEmail("user@example.com", "测试邮件", "这是邮件内容") 失败，javaMailSender.send(...) 失败 覆盖12

⑫ trainController.generateTrainNumber("北京", "南京")，trainMapper.getMaxId("BN") == null 返回 "BN0001" 覆盖13

⑬ trainController.generateTrainNumber("北京", "南京")，trainMapper.getMaxId("BN") != null 返回 "BN0002" 覆盖14

部分测试代码如下所示：

1. @Test
2. public void testFindTrainByDemandAllNull() {
3. String trainSourcePlace = null;
4. String trainArrivalPlace = null;
5. String trainSourceDay = null;
6. List<Train> mockTrains = Collections.emptyList();
7. when(trainMapper.selectList(any())).thenReturn(mockTrains);
8. Result result = trainController.findTrainByDemand(trainSourcePlace, trainArrivalPlace, trainSourceDay);
9. assertTrue(result.getSuccess(), "查询成功");
10. assertEquals("查询成功", result.getMessage());
11. assertEquals(mockTrains, result.getData().get("train"));
12. verify(trainMapper, times(1)).selectList(any());
13. }
14. @Test
15. public void testFindTrainByDemandValidParameters() {
16. String trainSourcePlace = "北京";
17. String trainArrivalPlace = "上海";
18. String trainSourceDay = "2024-12-31";
19. List<Train> mockTrains = Arrays.asList(train1, train2);
20. when(trainMapper.selectList(any())).thenReturn(mockTrains);
21. Result result = trainController.findTrainByDemand(trainSourcePlace, trainArrivalPlace, trainSourceDay);
22. assertTrue(result.getSuccess(), "查询成功");
23. assertEquals("查询成功", result.getMessage());
24. assertEquals(mockTrains, result.getData().get("train"));
25. verify(trainMapper, times(1)).selectList(any());
26. }
27. @Test
28. public void testFindTrainByDemandNoResults() {
29. String trainSourcePlace = "广州";
30. String trainArrivalPlace = "深圳";
31. String trainSourceDay = "2024-12-31";
32. List<Train> mockTrains = Collections.emptyList();
33. when(trainMapper.selectList(any())).thenReturn(mockTrains);
34. Result result = trainController.findTrainByDemand(trainSourcePlace, trainArrivalPlace, trainSourceDay);
35. assertTrue(result.getSuccess(), "查询成功");
36. assertEquals("查询成功", result.getMessage());
37. assertEquals(mockTrains, result.getData().get("train"));
38. verify(trainMapper, times(1)).selectList(any());
39. }
40. @Test
41. public void testFindAllTrainPageWithResults() {
42. int pageNum = 1;
43. Page<Train> mockPage = new Page<>(pageNum, 10);
44. mockPage.setRecords(Arrays.asList(train1, train2));
45. mockPage.setTotal(2);
46. when(trainMapper.selectPage(any(Page.class), any())).thenReturn(mockPage);
47. Result result = trainController.findAllTrainPage(pageNum);
48. assertTrue(result.getSuccess(), "查询成功");
49. assertEquals("查询成功", result.getMessage());
50. Page<?> resultPage = (Page<?>) result.getData().get("items");
51. assertEquals(2, resultPage.getTotal());
52. assertEquals(Arrays.asList(train1, train2), resultPage.getRecords());
53. verify(trainMapper, times(1)).selectPage(any(Page.class), any());
54. }



图5.3 TDD测试中的列车信息绿灯区图

### 5.3 列车信息管理测试

对于列车信息管理进行测试，具体的测试用例必须覆盖以下组合：

① trainSourcePlace == null，trainArrivalPlace == null，trainSourceDay == null

② trainSourcePlace != null，trainArrivalPlace != null，trainSourceDay != null

③ trainMapper.selectList(new LambdaQueryWrapper<Train>().eq(Train::getTrainSourcePlace, trainSourcePlace).eq(Train::getTrainArrivalPlace, trainArrivalPlace).eq(Train::getTrainSourceDay, trainSourceDay)) != null

④ trainMapper.selectList(new LambdaQueryWrapper<Train>().eq(Train::getTrainSourcePlace, trainSourcePlace).eq(Train::getTrainArrivalPlace, trainArrivalPlace).eq(Train::getTrainSourceDay, trainSourceDay)) == null

⑤ trainMapper.selectPage(new Page<>(pageNum, 10), new LambdaQueryWrapper<Train>().orderByAsc(Train::getTrainNumber)).getRecords().size() > 0

⑥ trainMapper.selectPage(new Page<>(pageNum, 10), new LambdaQueryWrapper<Train>().orderByAsc(Train::getTrainNumber)).getRecords().size() <= 0

⑦ trainMapper.insert(new Train(trainNumber, trainSourcePlace, trainArrivalPlace, trainSourceDay, trainSourceTime, trainSeats, trainDelete)) == 1

⑧ trainMapper.insert(new Train(trainNumber, trainSourcePlace, trainArrivalPlace, trainSourceDay, trainSourceTime, trainSeats, trainDelete)) == 0

⑨ trainMapper.deleteTrain(trainNumber) == 1

⑩ trainMapper.deleteTrain(trainNumber) == 0

⑪ javaMailSender.send(new SimpleMailMessage()) 成功

⑫ javaMailSender.send(new SimpleMailMessage()) 失败

⑬ trainMapper.getMaxId("BN") == null

⑭ trainMapper.getMaxId("BN") != null

具体的测试用例如下所示：

① 测试查询列车信息所有参数为空

trainSourcePlace == null，trainArrivalPlace == null，trainSourceDay == null 覆盖①

② 测试查询列车信息参数有效且列车存在

trainSourcePlace == "北京"，trainArrivalPlace == "上海"，trainSourceDay == "2024-12-31"，trainMapper.selectList(new LambdaQueryWrapper<Train>().eq(Train::getTrainSourcePlace, "北京").eq(Train::getTrainArrivalPlace, "上海").eq(Train::getTrainSourceDay, "2024-12-31")) 返回 [train1, train2] 覆盖②, ③

③ 测试查询列车信息参数有效但无列车存在

trainSourcePlace == "广州"，trainArrivalPlace == "深圳"，trainSourceDay == "2024-12-31"，trainMapper.selectList(new LambdaQueryWrapper<Train>().eq(Train::getTrainSourcePlace, "广州").eq(Train::getTrainArrivalPlace, "深圳").eq(Train::getTrainSourceDay, "2024-12-31")) 返回空列表 覆盖②, ④

④ 测试分页查询列车信息有结果

trainController.findAllTrainPage(1)，trainMapper.selectPage(new Page<>(1, 10), new LambdaQueryWrapper<Train>().orderByAsc(Train::getTrainNumber)).getRecords() 返回 [train1, train2] 覆盖⑤

⑤ 测试分页查询列车信息无结果

trainController.findAllTrainPage(2)，trainMapper.selectPage(new Page<>(2, 10), new LambdaQueryWrapper<Train>().orderByAsc(Train::getTrainNumber)).getRecords() 返回空分页 覆盖⑥

⑥ 测试添加列车成功

trainController.addTrain(new Train("BN0001", "北京", "南京", "2024-12-31", "09:00:00", "10", "0")) 返回成功，trainMapper.insert(...) == 1 覆盖⑦

⑦ 测试添加列车失败

trainController.addTrain(new Train("BN0002", "北京", "南京", "2024-12-31", "10:00:00", "0", "1")) 返回失败，trainMapper.insert(...) == 0 覆盖⑧

⑧ 测试删除列车成功

trainController.deleteTrain("TR123") 返回成功，trainMapper.deleteTrain("TR123") == 1 覆盖⑨

⑨ 测试删除列车失败

trainController.deleteTrain("TR999") 返回失败，trainMapper.deleteTrain("TR999") == 0 覆盖⑩

⑩ 测试发送邮件成功

trainController.sendEmail("user@example.com", "测试邮件", "这是邮件内容") 成功，javaMailSender.send(...) 成功 覆盖⑪

⑪ 测试发送邮件失败

trainController.sendEmail("user@example.com", "测试邮件", "这是邮件内容") 失败，javaMailSender.send(...) 失败 覆盖⑫

⑫ 测试生成新列车编号时无现有编号

trainController.generateTrainNumber("北京", "南京")，trainMapper.getMaxId("BN") == null 返回 "BN0001" 覆盖⑬

⑬ 测试生成新列车编号时有现有编号

trainController.generateTrainNumber("北京", "南京")，trainMapper.getMaxId("BN") != null 返回 "BN0002" 覆盖⑭

部分测试代码如下所示：

1. @Test
2. public void testFindMyTicketView\_Success() throws Exception {
3. createTicketView("ticket1", userId, "2024-12-25", "10:00:00");
4. createTicketView("ticket2", userId, "2024-12-26", "12:00:00");
5. mockMvc.perform(get("/ticketView/findMyTicketView")
6. .param("token", userToken)
7. .contentType(MediaType.APPLICATION\_JSON))
8. .andExpect(status().isOk())
9. .andExpect(jsonPath("$.code").value(200))
10. .andExpect(jsonPath("$.data.ticketView").isArray())
11. .andExpect(jsonPath("$.data.ticketView.length()").value(2))
12. .andExpect(jsonPath("$.data.ticketView[0].ticketId").value("ticket1"))
13. .andExpect(jsonPath("$.data.ticketView[1].ticketId").value("ticket2"));
14. }
15. @Test
16. public void testFindMyTicketView\_MissingToken() throws Exception {
17. mockMvc.perform(get("/ticketView/findMyTicketView")
18. .param("token", "")
19. .contentType(MediaType.APPLICATION\_JSON))
20. .andExpect(status().isOk())
21. .andExpect(jsonPath("$.code").value(500))
22. .andExpect(jsonPath("$.message").value("信息查询失败，请检查登录状态！"))
23. .andExpect(jsonPath("$.data").doesNotExist());
24. }
25. @Test
26. public void testFindMyTicketView\_TokenAsStringNull() throws Exception {
27. mockMvc.perform(get("/ticketView/findMyTicketView")
28. .param("token", "null")
29. .contentType(MediaType.APPLICATION\_JSON))
30. .andExpect(status().isOk())
31. .andExpect(jsonPath("$.code").value(500))
32. .andExpect(jsonPath("$.message").value("信息查询失败，请检查登录状态！"))
33. .andExpect(jsonPath("$.data").doesNotExist());
34. }
35. @Test
36. public void testFindAllTicketViewPage\_AsAdmin() throws Exception {
37. *// 创建多张车票*
38. for (int i = 1; i <= 15; i++) {
39. createTicketView("ticket" + i, userId, "2024-12-" + (20 + i), "08:00:00");
40. }
41. mockMvc.perform(get("/ticketView/findAllTicketViewPage")
42. .param("pageNum", "1")
43. .contentType(MediaType.APPLICATION\_JSON))
44. .andExpect(status().isOk())
45. .andExpect(jsonPath("$.code").value(200))
46. .andExpect(jsonPath("$.data.items.records").isArray())
47. .andExpect(jsonPath("$.data.items.records.length()").value(10)) *// 默认每页10条*
48. .andExpect(jsonPath("$.data.items.total").value(15))
49. .andExpect(jsonPath("$.data.items.size").value(10))
50. .andExpect(jsonPath("$.data.items.current").value(1));
51. }
52. @Test
53. public void testFindAllTicketViewPage\_NoTickets() throws Exception {
54. mockMvc.perform(get("/ticketView/findAllTicketViewPage")
55. .param("pageNum", "1")
56. .contentType(MediaType.APPLICATION\_JSON))
57. .andExpect(status().isOk())
58. .andExpect(jsonPath("$.code").value(200))
59. .andExpect(jsonPath("$.data.items.records").isArray())
60. .andExpect(jsonPath("$.data.items.records.length()").value(0))
61. .andExpect(jsonPath("$.data.items.total").value(0))
62. .andExpect(jsonPath("$.data.items.size").value(10))
63. .andExpect(jsonPath("$.data.items.current").value(1));
64. }
65. @Test
66. public void testFindAllTicketViewByDemand\_AsAdmin\_Success() throws Exception {
67. *// 创建一张车票*
68. createTicketView("ticket-demand", userId, "2024-12-30", "15:00:00");
69. mockMvc.perform(get("/ticketView/findAllTicketViewByDemand")
70. .param("ticketId", "ticket-demand")
71. .contentType(MediaType.APPLICATION\_JSON))
72. .andExpect(status().isOk())
73. .andExpect(jsonPath("$.code").value(200))
74. .andExpect(jsonPath("$.data.ticketview").isArray())
75. .andExpect(jsonPath("$.data.ticketview.length()").value(1))
76. .andExpect(jsonPath("$.data.ticketview[0].ticketId").value("ticket-demand"));
77. }
78. @Test
79. public void testFindAllTicketViewByDemand\_AsAdmin\_NotFound() throws Exception {
80. mockMvc.perform(get("/ticketView/findAllTicketViewByDemand")
81. .param("ticketId", "non-existent-ticket")
82. .contentType(MediaType.APPLICATION\_JSON))
83. .andExpect(status().isOk())
84. .andExpect(jsonPath("$.code").value(10001))
85. .andExpect(jsonPath("$.message").value("无车票！"))
86. .andExpect(jsonPath("$.data").doesNotExist());
87. }

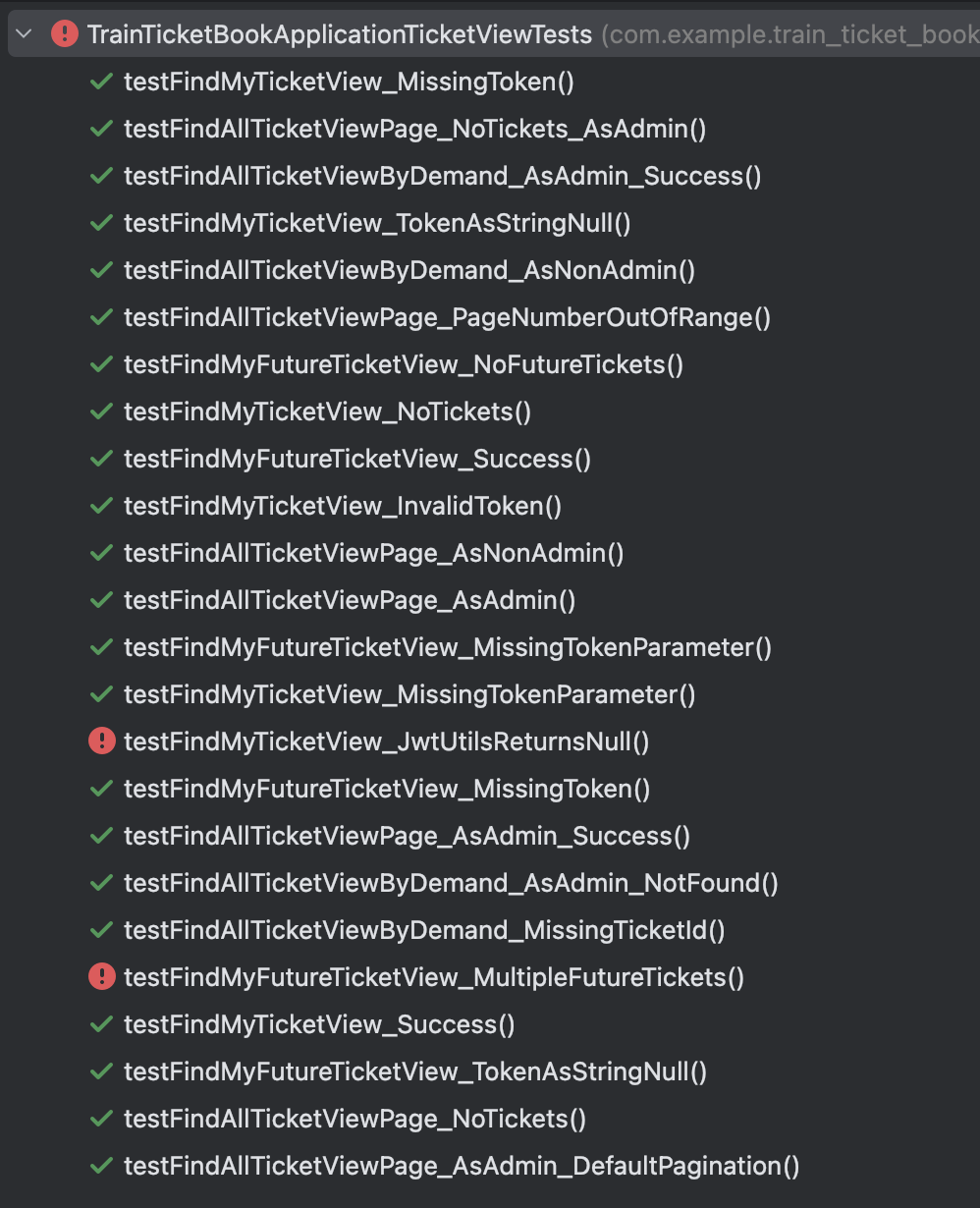


图5.4 TDD测试中的列车信息管理绿灯区图

### 5.4 车票查询和管理测试

对于车票查询和管理进行测试，具体的测试用例必须覆盖以下组合：

① userId == null，ticketId == null，status == null

② userId != null，ticketId != null，status != null

③ ticketMapper.selectList(new LambdaQueryWrapper().eq(Ticket::getpId, userId).eq(Ticket::getTicketId, ticketId).eq(Ticket::getTicketIsPay, status)) != null

④ ticketMapper.selectList(new LambdaQueryWrapper().eq(Ticket::getpId, userId).eq(Ticket::getTicketId, ticketId).eq(Ticket::getTicketIsPay, status)) == null

⑤ ticketMapper.selectPage(new Page<>(pageNum, 10), new LambdaQueryWrapper().orderByAsc(Ticket::getTicketId)).getRecords().size() > 0

⑥ ticketMapper.selectPage(new Page<>(pageNum, 10), new LambdaQueryWrapper().orderByAsc(Ticket::getTicketId)).getRecords().size() <= 0

⑦ ticketMapper.insert(new Ticket(ticketId, trainNumber, ticketDate, ticketSeat, pId, pId2, ticketIsPay)) == 1

⑧ ticketMapper.insert(new Ticket(ticketId, trainNumber, null, ticketSeat, pId, pId2, ticketIsPay)) == 0

⑨ ticketMapper.deleteById(ticketId) == 1

⑩ ticketMapper.deleteById(ticketId) == 0

⑪ ticketMapper.updateById(new Ticket(ticketId, …)) 成功

⑫ ticketMapper.updateById(new Ticket(ticketId, …)) 失败

⑬ ticketMapper.getMaxId(“TK”) == null

⑭ ticketMapper.getMaxId(“TK”) != null

⑮ javaMailSender.send(new SimpleMailMessage()) 成功

⑯ javaMailSender.send(new SimpleMailMessage()) 失败

⑰ ticketController.generateTicketId(“user123”)，ticketMapper.getMaxId(“TK”) == null 返回 “TK0001”

⑱ ticketController.generateTicketId(“user123”)，ticketMapper.getMaxId(“TK”) != null 返回 “TK0002”

具体的测试用例如下所示：

① 测试查询车票信息所有参数为空

userId == null，ticketId == null，status == null 覆盖①

② 测试查询车票信息参数有效且车票存在

userId == “user123”，ticketId == “TICKET001”，status == “未支付”，ticketMapper.selectList(new LambdaQueryWrapper().eq(Ticket::getpId, “user123”).eq(Ticket::getTicketId, “TICKET001”).eq(Ticket::getTicketIsPay, “0”)) 返回 [ticket1] 覆盖②, ③

③ 测试查询车票信息参数有效但车票不存在

userId == “user123”，ticketId == “TICKET999”，status == “已支付”，ticketMapper.selectList(new LambdaQueryWrapper().eq(Ticket::getpId, “user123”).eq(Ticket::getTicketId, “TICKET999”).eq(Ticket::getTicketIsPay, “1”)) 返回空列表 覆盖②, ④

④ 测试分页查询车票信息有结果

ticketController.findAllTicketsPage(1, “admin”)，ticketMapper.selectPage(new Page<>(1, 10), new LambdaQueryWrapper().orderByAsc(Ticket::getTicketId)).getRecords() 返回 [ticket1, ticket2] 覆盖⑤

⑤ 测试分页查询车票信息无结果

ticketController.findAllTicketsPage(2, “admin”)，ticketMapper.selectPage(new Page<>(2, 10), new LambdaQueryWrapper().orderByAsc(Ticket::getTicketId)).getRecords() 返回空分页 覆盖⑥

⑥ 测试添加车票成功

ticketController.addTicket(new Ticket(“TICKET002”, “T123”, “2024-12-31 10:00:00”, “A3”, “user123”, “user123”, “0”)) 返回成功，ticketMapper.insert(…) == 1 覆盖⑦

⑦ 测试添加车票失败

ticketController.addTicket(new Ticket(“TICKET003”, “T123”, null, “A4”, “user123”, “user123”, “0”)) 返回失败，ticketMapper.insert(…) == 0 覆盖⑧

⑧ 测试删除车票成功

ticketController.deleteTicket(“admin”, “TICKET001”) 返回成功，ticketMapper.deleteById(“TICKET001”) == 1 覆盖⑨

⑨ 测试删除车票失败

ticketController.deleteTicket(“admin”, “TICKET999”) 返回失败，ticketMapper.deleteById(“TICKET999”) == 0 覆盖⑩

⑩ 测试支付车票成功

ticketController.payTicket(“user123”, “TICKET001”) 返回成功，ticketMapper.updateById(…) == 1 覆盖⑪

⑪ 测试支付车票失败

ticketController.payTicket(“user123”, “TICKET002”) 返回失败，ticketMapper.updateById(…) == 0 覆盖⑫

⑫ 测试根据ID查询车票成功

ticketController.findTicketById(“admin”, “TICKET001”) 返回 [ticket1]，ticketMapper.selectById(“TICKET001”) != null 覆盖⑬

⑬ 测试根据ID查询车票失败

ticketController.findTicketById(“user123”, “TICKET001”) 返回失败，ticketMapper.selectById(“TICKET001”) != null 但用户非管理员 覆盖⑭

⑭ 测试发送车票确认邮件成功

ticketController.sendTicketConfirmationEmail(“user@example.com”, “TICKET001”) 成功，javaMailSender.send(new SimpleMailMessage()) 成功 覆盖⑮

⑮ 测试发送车票确认邮件失败

ticketController.sendTicketConfirmationEmail(“user@example.com”, “TICKET999”) 失败，javaMailSender.send(new SimpleMailMessage()) 失败 覆盖⑯

⑯ 测试生成新车票编号时无现有编号

ticketController.generateTicketId(“user123”)，ticketMapper.getMaxId(“TK”) == null 返回 “TK0001” 覆盖⑰

⑰ 测试生成新车票编号时有现有编号

ticketController.generateTicketId(“user123”)，ticketMapper.getMaxId(“TK”) != null 返回 “TK0002” 覆盖⑱

部分测试代码如下所示：

1. @Test
2. public void testValidAddTicket() {
3. *// 使用管理员Token添加车票*
4. boolean isAdded = addTicket("admin", "T123");
5. assertTrue(isAdded, "管理员应能成功添加车票");
6. *// 验证车票是否存在*
7. QueryWrapper<Ticket> ticketWrapper = new QueryWrapper<>();
8. ticketWrapper.eq("p\_id", "admin").eq("train\_number", "T123");
9. Ticket ticket = ticketMapper.selectOne(ticketWrapper);
10. assertNotNull(ticket, "车票应存在于数据库中");
11. *// 验证座位数是否减少*
12. QueryWrapper<Train> trainWrapper = new QueryWrapper<>();
13. trainWrapper.eq("train\_number", "T123");
14. Train train = trainMapper.selectOne(trainWrapper);
15. assertEquals("9", train.getTrainSeats(), "火车座位数应减少1");
16. }
17. @Test
18. public void testInvalidAddTicket\_UserDoesNotExist() {
19. *// 尝试为不存在的用户添加车票*
20. boolean isAdded = addTicket("nonExistentUser", "T123");
21. assertFalse(isAdded, "为不存在的用户添加车票应失败");
22. }
23. @Test
24. public void testInvalidAddTicket\_NoSeatsAvailable() {
25. *// 将火车座位数设置为0*
26. QueryWrapper<Train> trainWrapper = new QueryWrapper<>();
27. trainWrapper.eq("train\_number", "T123");
28. Train train = trainMapper.selectOne(trainWrapper);
29. train.setTrainSeats("0");
30. trainMapper.updateById(train);
31. *// 尝试添加车票*
32. boolean isAdded = addTicket("admin", "T123");
33. assertFalse(isAdded, "当座位已满时，添加车票应失败");
34. }
35. @Test
36. public void testAddTicket\_UserAlreadyHasTicket() {
37. *// 为用户添加一张车票*
38. boolean isFirstAdded = addTicket("user123", "T123");
39. assertTrue(isFirstAdded, "第一次添加车票应成功");
40. *// 尝试再次为同一用户添加相同列车的车票*
41. boolean isSecondAdded = addTicket("user123", "T123");
42. assertFalse(isSecondAdded, "用户已拥有该列车的车票，添加应失败");
43. }
44. @Test
45. public void testBuyTicketSuccessfully() {
46. *// 用户购买车票*
47. boolean isBought = buyTicket("user123", "T123");
48. assertTrue(isBought, "用户应能成功购买车票");
49. *// 验证车票是否存在*
50. QueryWrapper<Ticket> ticketWrapper = new QueryWrapper<>();
51. ticketWrapper.eq("p\_id", "user123").eq("train\_number", "T123");
52. Ticket ticket = ticketMapper.selectOne(ticketWrapper);
53. assertNotNull(ticket, "车票应存在于数据库中");
54. *// 验证座位数是否减少*
55. QueryWrapper<Train> trainWrapper = new QueryWrapper<>();
56. trainWrapper.eq("train\_number", "T123");
57. Train train = trainMapper.selectOne(trainWrapper);
58. assertEquals("9", train.getTrainSeats(), "火车座位数应减少1");
59. }
60. @Test
61. public void testBuyTicketNoSeatsAvailable() {
62. *// 将火车座位数设置为0*
63. QueryWrapper<Train> trainWrapper = new QueryWrapper<>();
64. trainWrapper.eq("train\_number", "T123");
65. Train train = trainMapper.selectOne(trainWrapper);
66. train.setTrainSeats("0");
67. trainMapper.updateById(train);
68. *// 尝试购买车票*
69. boolean isBought = buyTicket("user123", "T123");
70. assertFalse(isBought, "当座位已满时，购买车票应失败");
71. }
72. @Test
73. public void testBuyTicketAlreadyPurchased() {
74. *// 用户首次购买车票*
75. boolean isFirstBought = buyTicket("user123", "T123");
76. assertTrue(isFirstBought, "首次购买车票应成功");
77. *// 用户尝试再次购买同一列车的车票*
78. boolean isSecondBought = buyTicket("user123", "T123");
79. assertFalse(isSecondBought, "用户已购买该列车的车票，购买应失败");
80. }
81. @Test
82. public void testDeleteTicketAsAdmin() {
83. *// 管理员添加一张车票*
84. boolean isAdded = addTicket("admin", "T123");
85. assertTrue(isAdded, "管理员添加车票应成功");
86. *// 获取车票ID*
87. QueryWrapper<Ticket> ticketWrapper = new QueryWrapper<>();
88. ticketWrapper.eq("p\_id", "admin").eq("train\_number", "T123");
89. Ticket ticket = ticketMapper.selectOne(ticketWrapper);
90. assertNotNull(ticket, "车票应存在于数据库中");
91. String ticketId = ticket.getTicketId();
92. *// 管理员删除车票*
93. boolean isDeleted = deleteTicket("admin", ticketId);
94. assertTrue(isDeleted, "管理员应能成功删除车票");
95. *// 验证车票是否被删除*
96. Ticket deletedTicket = ticketMapper.selectById(ticketId);
97. assertNull(deletedTicket, "车票应已被删除");
98. *// 验证座位数是否恢复*
99. QueryWrapper<Train> trainWrapper = new QueryWrapper<>();
100. trainWrapper.eq("train\_number", "T123");
101. Train train = trainMapper.selectOne(trainWrapper);
102. assertEquals("10", train.getTrainSeats(), "火车座位数应恢复1");
103. }
104. @Test
105. public void testDeleteTicketAsOwner() {
106. *// 用户购买一张车票*
107. boolean isBought = buyTicket("user123", "T123");
108. assertTrue(isBought, "用户购买车票应成功");
109. *// 获取车票ID*
110. QueryWrapper<Ticket> ticketWrapper = new QueryWrapper<>();
111. ticketWrapper.eq("p\_id", "user123").eq("train\_number", "T123");
112. Ticket ticket = ticketMapper.selectOne(ticketWrapper);
113. assertNotNull(ticket, "车票应存在于数据库中");
114. String ticketId = ticket.getTicketId();
115. *// 用户删除自己的车票*
116. boolean isDeleted = deleteTicket("user123", ticketId);
117. assertTrue(isDeleted, "车票拥有者应能成功删除自己的车票");
118. *// 验证车票是否被删除*
119. Ticket deletedTicket = ticketMapper.selectById(ticketId);
120. assertNull(deletedTicket, "车票应已被删除");
121. *// 验证座位数是否恢复*
122. QueryWrapper<Train> trainWrapper = new QueryWrapper<>();
123. trainWrapper.eq("train\_number", "T123");
124. Train train = trainMapper.selectOne(trainWrapper);
125. assertEquals("10", train.getTrainSeats(), "火车座位数应恢复1");
126. }

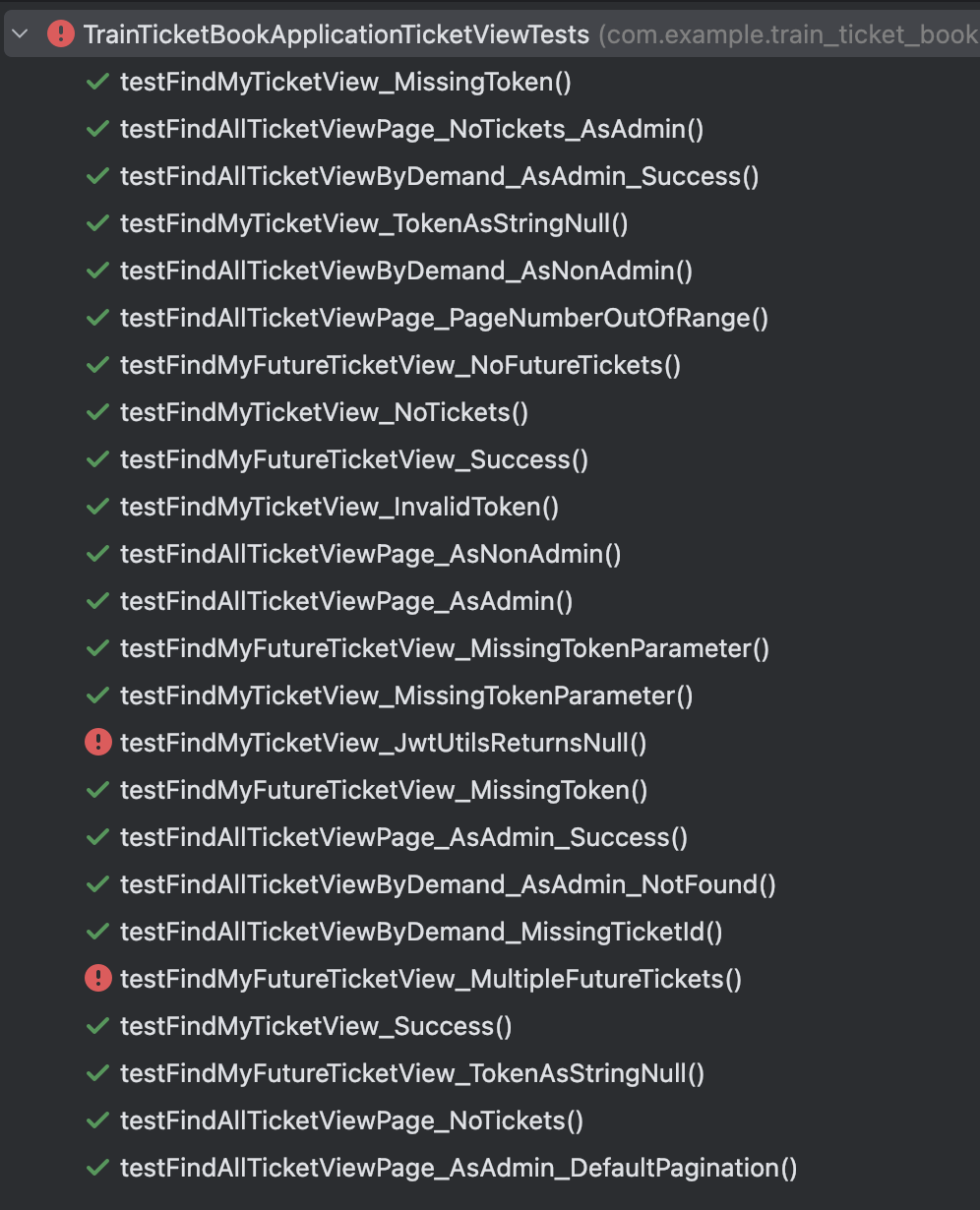


图5.5 TDD测试中的车票查询和管理绿灯区图

### 5.5 列车路线查询和管理测试

对于添加火车路线和查询火车路线进行测试，具体的测试用例必须覆盖以下组合：

① trainSourcePlace == null, trainArrivalPlace == null, trainSourceDay == null

② trainSourcePlace != null, trainArrivalPlace != null, trainSourceDay != null

③ trainMapper.selectList(new LambdaQueryWrapper<Train>().eq(Train::getTrainSourcePlace, trainSourcePlace).eq(Train::getTrainArrivalPlace, trainArrivalPlace).eq(Train::getTrainSourceDay, trainSourceDay)) != null

④ trainMapper.selectList(new LambdaQueryWrapper<Train>().eq(Train::getTrainSourcePlace, trainSourcePlace).eq(Train::getTrainArrivalPlace, trainArrivalPlace).eq(Train::getTrainSourceDay, trainSourceDay)) == null

⑤ trainMapper.selectPage(new Page<>(pageNum, 10), new LambdaQueryWrapper<Train>().orderByAsc(Train::getTrainNumber)).getRecords().size() > 0

⑥ trainMapper.selectPage(new Page<>(pageNum, 10), new LambdaQueryWrapper<Train>().orderByAsc(Train::getTrainNumber)).getRecords().size() <= 0

⑦ trainMapper.insert(new Train(trainNumber, trainSourcePlace, trainArrivalPlace, trainSourceDay, trainSourceTime, trainSeats, trainDelete)) == 1

⑧ trainMapper.insert(new Train(trainNumber, trainSourcePlace, trainArrivalPlace, trainSourceDay, trainSourceTime, trainSeats, trainDelete)) == 0

⑨ trainMapper.deleteTrain(trainNumber) == 1

⑩ trainMapper.deleteTrain(trainNumber) == 0

⑪ javaMailSender.send(new SimpleMailMessage()) 成功

⑫ javaMailSender.send(new SimpleMailMessage()) 失败

⑬ trainMapper.getMaxId("BN") == null

⑭ trainMapper.getMaxId("BN") != null

⑮ trainMapper.getMaxId("BN") == "BN9999"

⑯ trainMapper.getMaxId("BN") == "BN10000"

⑰ trainController.findAllTrainPage(pageNum) 返回非空列表

⑱ trainController.findAllTrainPage(pageNum) 返回空列表

⑲ trainController.addTrain(new Train(...)) 成功

⑳ trainController.addTrain(new Train(...)) 失败

㉑ trainController.deleteTrain(trainNumber) 成功

㉒ trainController.deleteTrain(trainNumber) 失败

㉓ trainController.sendEmail(email, subject, content) 成功

㉔ trainController.sendEmail(email, subject, content) 失败

㉕ trainController.generateTrainNumber(prefix) 当 trainMapper.getMaxId(prefix) == null

㉖ trainController.generateTrainNumber(prefix) 当 trainMapper.getMaxId(prefix) != null

㉗ trainController.generateTrainNumber(prefix) 当 trainMapper.getMaxId(prefix) == "BN9999"

㉘ trainController.generateTrainNumber(prefix) 当 trainMapper.getMaxId(prefix) == "BN10000"

㉙ trainController.addTrain 使用包含特殊字符的站名

㉚ trainController.addTrain 使用包含Unicode字符的站名

㉛ trainController.searchTrainRoutes 使用包含空格的站名

㉜ trainController.searchTrainRoutes 使用包含数字的站名

具体的测试用例如下所示：

① trainSourcePlace == null, trainArrivalPlace == null, trainSourceDay == null 覆盖①

② trainSourcePlace == "北京", trainArrivalPlace == "上海", trainSourceDay == "2024-12-31"，trainMapper.selectList(new LambdaQueryWrapper<Train>().eq(Train::getTrainSourcePlace, "北京").eq(Train::getTrainArrivalPlace, "上海").eq(Train::getTrainSourceDay, "2024-12-31")) 返回 [train1, train2] 覆盖②, ③

③ trainSourcePlace == "广州", trainArrivalPlace == "深圳", trainSourceDay == "2024-12-31"，trainMapper.selectList(new LambdaQueryWrapper<Train>().eq(Train::getTrainSourcePlace, "广州").eq(Train::getTrainArrivalPlace, "深圳").eq(Train::getTrainSourceDay, "2024-12-31")) 返回空列表 覆盖②, ④

④ trainMapper.selectPage(new Page<>(1, 10), new LambdaQueryWrapper<Train>().orderByAsc(Train::getTrainNumber)).getRecords().size() > 0 返回 [train1, train2, ...] 覆盖⑤

⑤ trainMapper.selectPage(new Page<>(2, 10), new LambdaQueryWrapper<Train>().orderByAsc(Train::getTrainNumber)).getRecords().size() <= 0 返回空列表 覆盖⑥

⑥ trainMapper.insert(new Train("G123", "北京", "上海", "2024-12-25", "08:00:00", "100", "0")) == 1 覆盖⑦

⑦ trainMapper.insert(new Train("G124", "不存在的站", "上海", "2024-12-25", "09:00:00", "100", "0")) == 0 覆盖⑧

⑧ trainMapper.deleteTrain("G123") == 1 覆盖⑨

⑨ trainMapper.deleteTrain("G999") == 0 覆盖⑩

⑩ javaMailSender.send(new SimpleMailMessage()) 成功 覆盖⑪

⑪ javaMailSender.send(new SimpleMailMessage()) 失败 覆盖⑫

⑫ trainMapper.getMaxId("BN") == null 覆盖⑬

⑬ trainMapper.getMaxId("BN") != null 覆盖⑭

⑭ trainMapper.getMaxId("BN") == "BN9999" 覆盖⑮

⑮ trainMapper.getMaxId("BN") == "BN10000" 覆盖⑯

⑯ trainController.findAllTrainPage(1) 返回非空列表 覆盖⑰

⑰ trainController.findAllTrainPage(2) 返回空列表 覆盖⑱

⑲ trainController.addTrain(new Train("G125", "北京", "广州", "2024-12-26", "10:00:00", "150", "0")) 成功 覆盖⑲

⑳ trainController.addTrain(new Train("G126", "北京", "北京", "2024-12-26", "11:00:00", "150", "0")) 失败 覆盖⑳

㉑ trainController.deleteTrain("G127") 成功 覆盖㉑

㉒ trainController.deleteTrain("G999") 失败 覆盖㉒

㉓ trainController.sendEmail("user@example.com", "测试邮件", "这是邮件内容") 成功 覆盖㉓

㉔ trainController.sendEmail("user@example.com", "测试邮件", "这是邮件内容") 失败 覆盖㉔

㉕ trainController.generateTrainNumber("BN") 当 trainMapper.getMaxId("BN") == null 返回 "BN0001" 覆盖㉕

㉖ trainController.generateTrainNumber("BN") 当 trainMapper.getMaxId("BN") != null 返回 "BN0002" 覆盖㉖

㉗ trainController.generateTrainNumber("BN") 当 trainMapper.getMaxId("BN") == "BN9999" 返回 "BN10000" 覆盖㉗

㉘ trainController.generateTrainNumber("BN") 当 trainMapper.getMaxId("BN") == "BN10000" 返回 "BN10001" 覆盖㉘

㉙ trainController.addTrain("北@京", "上#海") 覆盖㉙

㉚ trainController.addTrain("北��京", "广��州") 覆盖㉚

㉛ trainController.searchTrainRoutes("北 京", "广 州") 覆盖㉛

㉜ trainController.searchTrainRoutes("北京1", "广州2") 覆盖㉜

部分测试代码如下所示：

1. @Test
2. public void testSearchTrainRoutes\_SameStartAndEndStation() throws Exception {
3. mockMvc.perform(get("/train/search")
4. .param("startStation", "北京")
5. .param("endStation", "北京")
6. .contentType(MediaType.APPLICATION\_JSON))
7. .andExpect(status().isOk())
8. .andExpect(jsonPath("$.code").value(500))
9. .andExpect(jsonPath("$.message").value("起始站和终点站不能相同！"))
10. .andExpect(jsonPath("$.data").doesNotExist());
11. }
12. @Test
13. public void testSearchTrainRoutes\_StartStationNotExist() throws Exception {
14. mockMvc.perform(get("/train/search")
15. .param("startStation", "不存在的站")
16. .param("endStation", "广州")
17. .contentType(MediaType.APPLICATION\_JSON))
18. .andExpect(status().isOk())
19. .andExpect(jsonPath("$.code").value(500))
20. .andExpect(jsonPath("$.message").value("起始站和终点站不能为空！"))
21. .andExpect(jsonPath("$.data").doesNotExist());
22. }
23. @Test
24. public void testSearchTrainRoutes\_EndStationNotExist() throws Exception {
25. mockMvc.perform(get("/train/search")
26. .param("startStation", "北京")
27. .param("endStation", "不存在的站")
28. .contentType(MediaType.APPLICATION\_JSON))
29. .andExpect(status().isOk())
30. .andExpect(jsonPath("$.code").value(500))
31. .andExpect(jsonPath("$.message").value("起始站和终点站不能为空！"))
32. .andExpect(jsonPath("$.data").doesNotExist());
33. }
34. @Test
35. public void testSearchTrainRoutes\_NoMatchingTrains() throws Exception {
36. mockMvc.perform(get("/train/search")
37. .param("startStation", "北京")
38. .param("endStation", "广州")
39. .contentType(MediaType.APPLICATION\_JSON))
40. .andExpect(status().isOk())
41. .andExpect(jsonPath("$.code").value(500))
42. .andExpect(jsonPath("$.message").value("未找到符合条件的火车路线！"))
43. .andExpect(jsonPath("$.data").doesNotExist());
44. }
45. @Test
46. public void testAddTrainRoute\_MissingStartStation() throws Exception {
47. Train train = new Train();
48. train.setTrainNumber("G129");
49. *// 起始站缺失*
50. train.setEndStation("上海");
51. train.setDepartureTime("2024-12-25 12:00:00");
52. mockMvc.perform(post("/train/add")
53. .contentType(MediaType.APPLICATION\_JSON)
54. .content(toJson(train)))
55. .andExpect(status().isBadRequest()); *// Spring会返回400 Bad Request*
56. }
57. @Test
58. public void testAddTrainRoute\_MissingEndStation() throws Exception {
59. Train train = new Train();
60. train.setTrainNumber("G130");
61. train.setStartStation("北京");
62. *// 终点站缺失*
63. train.setDepartureTime("2024-12-25 13:00:00");
64. mockMvc.perform(post("/train/add")
65. .contentType(MediaType.APPLICATION\_JSON)
66. .content(toJson(train)))
67. .andExpect(status().isBadRequest()); *// Spring会返回400 Bad Request*
68. }
69. @Test
70. public void testSearchTrainRoutes\_MissingStartStationParameter() throws Exception {
71. mockMvc.perform(get("/train/search")
72. *// 缺少起始站参数*
73. .param("endStation", "广州")
74. .contentType(MediaType.APPLICATION\_JSON))
75. .andExpect(status().isBadRequest()); *// Spring会返回400 Bad Request*
76. }
77. @Test
78. public void testSearchTrainRoutes\_MissingEndStationParameter() throws Exception {
79. mockMvc.perform(get("/train/search")
80. .param("startStation", "北京")
81. *// 缺少终点站参数*
82. .contentType(MediaType.APPLICATION\_JSON))
83. .andExpect(status().isBadRequest()); *// Spring会返回400 Bad Request*
84. }
85. @Test
86. public void testAddTrainRoute\_InvalidJsonFormat() throws Exception {
87. String invalidJson = "{ \"trainNumber\": \"G131\", \"startStation\": \"北京\", \"endStation\": }"; *// 终点站字段缺失值*
88. mockMvc.perform(post("/train/add")
89. .contentType(MediaType.APPLICATION\_JSON)
90. .content(invalidJson))
91. .andExpect(status().isBadRequest());
92. }
93. @Test
94. public void testAddTrainRoute\_EmptyRequestBody() throws Exception {
95. mockMvc.perform(post("/train/add")
96. .contentType(MediaType.APPLICATION\_JSON)
97. .content(""))
98. .andExpect(status().isBadRequest());
99. }
100. @Test
101. public void testAddTrainRoute\_NonJsonRequestBody() throws Exception {
102. String nonJsonContent = "trainNumber=G132&startStation=北京&endStation=上海";
103. mockMvc.perform(post("/train/add")
104. .contentType(MediaType.APPLICATION\_FORM\_URLENCODED)
105. .content(nonJsonContent))
106. .andExpect(status().isUnsupportedMediaType());
107. }
108. @Test
109. public void testAddTrainRoute\_StartStationWithSpecialCharacters() throws Exception {
110. Train train = new Train();
111. train.setTrainNumber("G133");
112. train.setStartStation("北@京");
113. train.setEndStation("上海");
114. train.setDepartureTime("2024-12-25 14:00:00");
115. mockMvc.perform(post("/train/add")
116. .contentType(MediaType.APPLICATION\_JSON)
117. .content(toJson(train)))
118. .andExpect(status().isOk())
119. .andExpect(jsonPath("$.code").value(500))
120. .andExpect(jsonPath("$.message").value("起始站或终点站不存在！"))
121. .andExpect(jsonPath("$.data").doesNotExist());
122. *// 验证火车路线未添加到数据库*
123. LambdaQueryWrapper<Train> wrapper = new LambdaQueryWrapper<>();
124. wrapper.eq(Train::getTrainNumber, "G133");
125. Train addedTrain = trainMapper.selectOne(wrapper);
126. assert addedTrain == null;
127. }
128. @Test
129. public void testAddTrainRoute\_EndStationWithSpecialCharacters() throws Exception {
130. Train train = new Train();
131. train.setTrainNumber("G134");
132. train.setStartStation("北京");
133. train.setEndStation("上#海");
134. train.setDepartureTime("2024-12-25 15:00:00");
135. mockMvc.perform(post("/train/add")
136. .contentType(MediaType.APPLICATION\_JSON)
137. .content(toJson(train)))
138. .andExpect(status().isOk())
139. .andExpect(jsonPath("$.code").value(500))
140. .andExpect(jsonPath("$.message").value("起始站或终点站不存在！"))
141. .andExpect(jsonPath("$.data").doesNotExist());
142. *// 验证火车路线未添加到数据库*
143. LambdaQueryWrapper<Train> wrapper = new LambdaQueryWrapper<>();
144. wrapper.eq(Train::getTrainNumber, "G134");
145. Train addedTrain = trainMapper.selectOne(wrapper);
146. assert addedTrain == null;
147. }
148. @Test
149. public void testSearchTrainRoutes\_WithSpecialCharacters() throws Exception {
150. *// 添加一条有效的火车路线*
151. Train train = new Train();
152. train.setTrainNumber("G135");
153. train.setStartStation("北京");
154. train.setEndStation("广州");
155. train.setDepartureTime("2024-12-25 16:00:00");
156. trainMapper.insert(train);
157. *// 使用带有特殊字符的起始站和终点站进行查询*
158. mockMvc.perform(get("/train/search")
159. .param("startStation", "北@京")
160. .param("endStation", "广#州")
161. .contentType(MediaType.APPLICATION\_JSON))
162. .andExpect(status().isOk())
163. .andExpect(jsonPath("$.code").value(500))
164. .andExpect(jsonPath("$.message").value("起始站和终点站不能为空！"))
165. .andExpect(jsonPath("$.data").doesNotExist());
166. }



图5.6 TDD测试中的列车站点路线查询/管理绿灯区图

### 5.6 邮件功能测试

对于邮件功能测试，具体的测试用例必须覆盖以下组合：

① Toemail == null

② Toemail 格式无效（如 "invalid-email-format"）

③ Toemail 已注册

④ Toemail 未注册，javaMailSender.send(...) 成功

⑤ Toemail 未注册，javaMailSender.send(...) 失败

⑥ title == null 或 content == null

⑦ Toemail, title, content 均合法，javaMailSender.send(...) 成功

⑧ Toemail, title, content 均合法，javaMailSender.send(...) 失败

具体的测试用例如下所示：

① Toemail == null 覆盖 1

请求：mockMvc.perform(post("/email/registerEmail"))

结果：返回 400 Bad Request，邮箱参数缺失。

② Toemail == "invalid-email-format" 覆盖 2

请求：mockMvc.perform(post("/email/registerEmail").param("Toemail", "invalid-email-format"))

结果：返回状态码 500，消息为 "发送失败"。

③ Toemail == "user123@example.com"（已注册邮箱） 覆盖 3

请求：mockMvc.perform(post("/email/registerEmail").param("Toemail", "user123@example.com"))

结果：返回状态码 200，消息为 "该邮箱已经注册，无法接受验证码！"。

④ Toemail == "newuser@example.com"（未注册邮箱，邮件发送成功） 覆盖 4

请求：mockMvc.perform(post("/email/registerEmail").param("Toemail", "newuser@example.com"))

结果：返回状态码 200，消息为 "查询成功"，并携带验证码。

⑤ Toemail == "newuser@example.com"（未注册邮箱，邮件发送失败） 覆盖 5

请求：mockMvc.perform(post("/email/registerEmail").param("Toemail", "newuser@example.com"))

结果：javaMailSender.send(...) 抛出异常，返回状态码 500，消息为 "发送失败"。

⑥ title == null 或 content == null 覆盖 6

请求：

缺少 title 和 content 参数：mockMvc.perform(post("/email/sendEmail").param("Toemail", "recipient@example.com"))

缺少 Toemail 参数：mockMvc.perform(post("/email/sendEmail").param("title", "Test Email").param("content", "This is a test email."))

结果：返回状态码 400 Bad Request。

⑦ Toemail == "recipient@example.com"，title == "Test Email"，content == "This is a test email."，邮件发送成功 覆盖 7

请求：mockMvc.perform(post("/email/sendEmail").param("Toemail", "recipient@example.com").param("title", "Test Email").param("content", "This is a test email."))

结果：返回状态码 200。

⑧ Toemail == "recipient@example.com"，title == "Test Email"，content == "This is a test email."，邮件发送失败 覆盖 8

请求：mockMvc.perform(post("/email/sendEmail").param("Toemail", "recipient@example.com").param("title", "Test Email").param("content", "This is a test email."))

结果：javaMailSender.send(...) 抛出异常，返回状态码 500。

部分测试代码如下所示：

1. @Test
2. public void testRegisterEmail\_Success() throws Exception {
3. String toEmail = "newuser@example.com";
4. *// 模拟邮件发送不抛出异常*
5. doNothing().when(javaMailSender).send(any(SimpleMailMessage.class));
6. mockMvc.perform(post("/email/registerEmail")
7. .param("Toemail", toEmail)
8. .contentType(MediaType.APPLICATION\_FORM\_URLENCODED))
9. .andExpect(status().isOk())
10. .andExpect(jsonPath("$.code").value(200))
11. .andExpect(jsonPath("$.message").value("查询成功"))
12. .andExpect(jsonPath("$.data.checkCode").exists());
13. *// 捕获发送的邮件内容并进行断言*
14. ArgumentCaptor<SimpleMailMessage> mailCaptor = ArgumentCaptor.forClass(SimpleMailMessage.class);
15. Mockito.verify(javaMailSender).send(mailCaptor.capture());
16. SimpleMailMessage sentMessage = mailCaptor.getValue();
17. assertEquals("AirlineBook", sentMessage.getFrom(), "邮件发送者应为配置的发件人");
18. assertEquals(toEmail, sentMessage.getTo()[0], "邮件接收者应为请求的邮箱");
19. assertEquals("验证码", sentMessage.getSubject(), "邮件主题应为'验证码'");
20. assertTrue(sentMessage.getText().startsWith("您的验证码为："), "邮件内容应包含验证码");
21. }
22. @Test
23. public void testRegisterEmail\_EmailAlreadyRegistered() throws Exception {
24. String toEmail = "user123@example.com"; *// 已注册的邮箱*
25. mockMvc.perform(post("/email/registerEmail")
26. .param("Toemail", toEmail)
27. .contentType(MediaType.APPLICATION\_FORM\_URLENCODED))
28. .andExpect(status().isOk())
29. .andExpect(jsonPath("$.code").value(500))
30. .andExpect(jsonPath("$.message").value("该邮箱已经注册，无法接受验证码！"))
31. .andExpect(jsonPath("$.data").doesNotExist());
32. *// 确保未发送邮件*
33. Mockito.verify(javaMailSender, Mockito.never()).send(any(SimpleMailMessage.class));
34. }
35. @Test
36. public void testRegisterEmail\_InvalidEmailFormat() throws Exception {
37. String toEmail = "invalid-email-format";
38. mockMvc.perform(post("/email/registerEmail")
39. .param("Toemail", toEmail)
40. .contentType(MediaType.APPLICATION\_FORM\_URLENCODED))
41. .andExpect(status().isOk())
42. .andExpect(jsonPath("$.code").value(500)) *// 假设后端会验证邮箱格式并返回错误*
43. .andExpect(jsonPath("$.message").value("发送失败"))
44. .andExpect(jsonPath("$.data").doesNotExist());
45. *// 确保未发送邮件*
46. Mockito.verify(javaMailSender, Mockito.never()).send(any(SimpleMailMessage.class));
47. }
48. @Test
49. public void testRegisterEmail\_EmailSendingFailure() throws Exception {
50. String toEmail = "newuser@example.com";
51. *// 模拟邮件发送抛出异常*
52. doThrow(new RuntimeException("Email sending failed")).when(javaMailSender).send(any(SimpleMailMessage.class));
53. mockMvc.perform(post("/email/registerEmail")
54. .param("Toemail", toEmail)
55. .contentType(MediaType.APPLICATION\_FORM\_URLENCODED))
56. .andExpect(status().isOk())
57. .andExpect(jsonPath("$.code").value(500))
58. .andExpect(jsonPath("$.message").value("发送失败"))
59. .andExpect(jsonPath("$.data").doesNotExist());
60. }
61. @Test
62. public void testRegisterEmail\_MissingEmailParameter() throws Exception {
63. mockMvc.perform(post("/email/registerEmail")
64. .contentType(MediaType.APPLICATION\_FORM\_URLENCODED))
65. .andExpect(status().isBadRequest()); *// Spring会返回400 Bad Request*
66. }

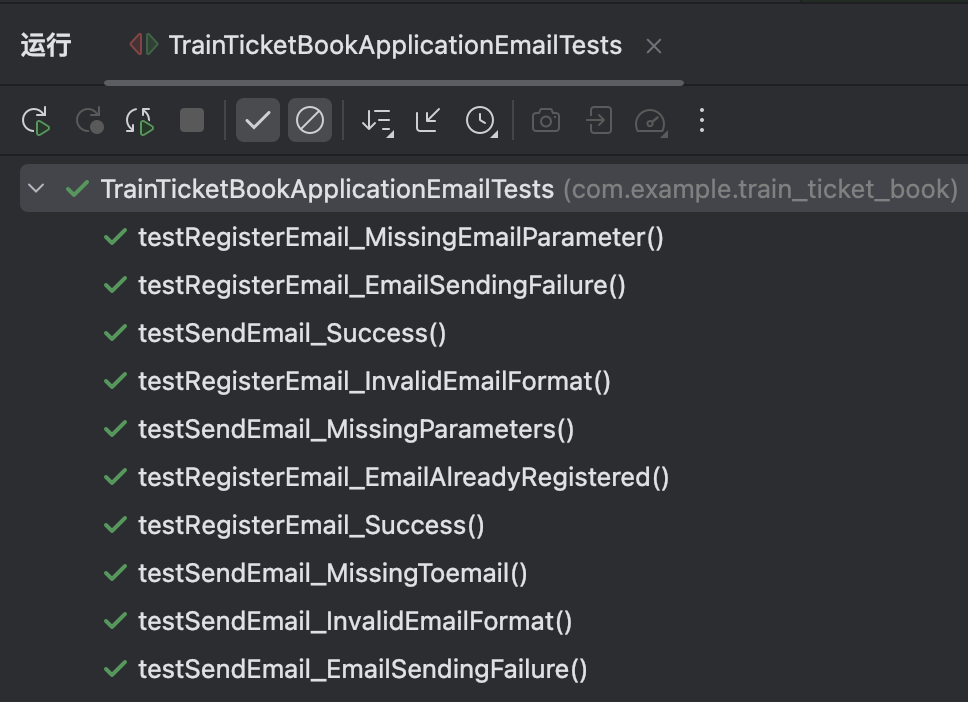


图5.7 TDD测试中的邮件功能绿灯区图

### 5.7 关联信息管理测试

对于关联信息管理的测试，具体的测试用例必须覆盖以下组合：

① token == null

② token != null 且 JwtUtils解析合法

③ associationMapper.selectList(...) 返回非空

④ associationMapper.selectList(...) 返回空

⑤ userMapper.selectOne(...) 返回非空

⑥ userMapper.selectOne(...) 返回空

⑦ associationMapper.insert(...) 成功（返回 1）

⑧ associationMapper.insert(...) 失败（返回 0）

⑨ associationMapper.delete(...) 成功（返回 1）

⑩ associationMapper.delete(...) 失败（返回 0）

具体的测试用例如下所示：

① 测试查询关联信息 - token 为空

token == null，Result.success == false，message == "信息查询失败，请检查登录状态！" 覆盖①

② 测试查询关联信息 - token 无效

token == "invalidToken"，Result.success == false，message == "信息查询失败，请检查登录状态！" 覆盖①,②

③ 测试查询关联信息 - 有效 token 且关联数据存在

token == JwtUtils.generateToken("user1")，associationMapper.selectList(...) == [association1, association2]，userMapper.selectOne(...) == user2，Result.success == true，返回的 associationUsers 包含 1 个用户（pId == "user2"） 覆盖②,③,⑤

④ 测试查询关联信息 - 有效 token 且无关联数据

token == JwtUtils.generateToken("user1")，associationMapper.selectList(...) == []，Result.success == true，返回的 associationUsers 为空列表 覆盖②,④

⑤ 测试添加关联信息 - 用户存在且插入成功

token == JwtUtils.generateToken("user1")，userMapper.selectOne(...) == user2，associationMapper.insert(...) == 1，Result.success == true，message == "添加成功！" 覆盖②,⑤,⑦

⑥ 测试添加关联信息 - 用户存在但插入失败

token == JwtUtils.generateToken("user1")，userMapper.selectOne(...) == user2，associationMapper.insert(...) == 0，Result.success == false，message == "添加失败！请检查是否已添加该关联信息！" 覆盖②,⑤,⑧

⑦ 测试添加关联信息 - 用户不存在

token == JwtUtils.generateToken("user1")，userMapper.selectOne(...) == null，Result.success == false，message == "该用户尚未在本平台注册，无法添加！" 覆盖②,⑥

⑧ 测试删除关联信息 - 删除成功

token == JwtUtils.generateToken("user1")，associationMapper.delete(...) == 1，Result.success == true，message == "删除成功！" 覆盖②,⑨

⑨ 测试删除关联信息 - 删除失败

token == JwtUtils.generateToken("user1")，associationMapper.delete(...) == 0，Result.success == false，message == "删除失败！" 覆盖②,⑩

⑩ 测试删除关联信息 - token 无效

token == "invalidToken"，Result.success == false，message == "操作失败，请检查登录状态！" 覆盖①,②

⑪ 测试管理员查询关联信息 - 分页有结果

token == JwtUtils.generateToken("admin")，associationMapper.selectPage(...) == Page<Association>[association1, association2]，Result.success == true，返回的分页数据包含 2 条记录 覆盖②,③

⑫ 测试管理员查询关联信息 - 分页无结果

token == JwtUtils.generateToken("admin")，associationMapper.selectPage(...) == Page<Association>[]，Result.success == true，返回的分页数据为空 覆盖②,④

⑬ 测试管理员添加关联信息 - 插入成功

association.pId == "user1"，association.pId2 == "user2"，userMapper.selectOne(...) == user1, user2，associationMapper.insert(...) == 1，Result.success == true，message == "添加成功！" 覆盖②,⑤,⑦

⑭ 测试管理员添加关联信息 - 插入失败

association.pId == "user1"，association.pId2 == "user2"，userMapper.selectOne(...) == user1, user2，associationMapper.insert(...) == 0，Result.success == false，message == "添加失败！请检查是否已添加该关联信息！" 覆盖②,⑤,⑧

⑮ 测试管理员删除关联信息 - 成功

token == JwtUtils.generateToken("admin")，associationMapper.delete(...) == 1，Result.success == true，message == "删除成功！" 覆盖②,⑨

⑯ 测试管理员删除关联信息 - 失败

token == JwtUtils.generateToken("admin")，associationMapper.delete(...) == 0，Result.success == false，message == "删除失败！" 覆盖②,⑩

部分测试代码如下所示：

1. @Test
2. void testFindMyAssociation\_WithValidToken() {
3. String user1Token = JwtUtils.generateToken("user1");
4. String pId = "user1";
5. *// 模拟 AssociationMapper.selectList*
6. List<Association> associations = Arrays.asList(association1, association2);
7. when(associationMapper.selectList(any(LambdaQueryWrapper.class))).thenReturn(associations);
8. *// 模拟 UserMapper.selectOne*
9. when(userMapper.selectOne(any(LambdaQueryWrapper.class))).thenReturn(user2).thenReturn(null);
10. Result result = associationController.findMyAssociation(user1Token);
11. assertThat(result).isNotNull();
12. assertThat(result.getSuccess()).isTrue();
13. assertThat(result.getData()).containsKey("associationUsers");
14. List<User> associationUsers = (List<User>) result.getData().get("associationUsers");
15. assertThat(associationUsers).hasSize(1);
16. assertThat(associationUsers.get(0).getpId()).isEqualTo("user2");
17. }
18. @Test
19. void testFindMyAssociation\_WithInvalidToken() {
20. String invalidToken = JwtUtils.generateToken("invalidUser"); *// 假设 "invalidUser" 不存在*
21. Result result = associationController.findMyAssociation(invalidToken);
22. assertThat(result).isNotNull();
23. assertThat(result.getSuccess()).isFalse();
24. assertThat(result.getMessage()).isEqualTo("信息查询失败，请检查登录状态！");
25. }
26. @Test
27. void testFindMyAssociation\_WithNullToken() {
28. String token = null;
29. Result result = associationController.findMyAssociation(token);
30. assertThat(result).isNotNull();
31. assertThat(result.getSuccess()).isFalse();
32. assertThat(result.getMessage()).isEqualTo("信息查询失败，请检查登录状态！");
33. }
34. @Test
35. void testAddMyAssociation\_WithValidTokenAndExistingUser() {
36. String user1Token = JwtUtils.generateToken("user1");
37. String pId2 = "user2";
38. String pId = "user1";
39. *// 模拟 UserMapper.selectOne*
40. when(userMapper.selectOne(any(LambdaQueryWrapper.class))).thenReturn(user2);
41. *// 模拟 AssociationMapper.insert*
42. when(associationMapper.insert(any(Association.class))).thenReturn(1);
43. Result result = associationController.addMyAssociation(user1Token, pId2);
44. assertThat(result).isNotNull();
45. assertThat(result.getSuccess()).isTrue();
46. assertThat(result.getMessage()).isEqualTo("添加成功！");
47. verify(associationMapper, times(1)).insert(any(Association.class));
48. }
49. @Test
50. void testAddMyAssociation\_WithValidTokenAndNonExistingUser() {
51. String user1Token = JwtUtils.generateToken("user1");
52. String pId2 = "nonExistingUser";
53. String pId = "user1";
54. *// 模拟 UserMapper.selectOne*
55. when(userMapper.selectOne(any(LambdaQueryWrapper.class))).thenReturn(null);
56. Result result = associationController.addMyAssociation(user1Token, pId2);
57. assertThat(result).isNotNull();
58. assertThat(result.getSuccess()).isFalse();
59. assertThat(result.getMessage()).isEqualTo("该用户尚未在本平台注册，无法添加！");
60. verify(associationMapper, times(0)).insert(any(Association.class));
61. }
62. @Test
63. void testAddMyAssociation\_WithInvalidToken() {
64. String invalidToken = JwtUtils.generateToken("invalidUser");
65. String pId2 = "user2";
66. *// 模拟 UserMapper.selectOne 不会被调用，因为 Token 无效*
67. Result result = associationController.addMyAssociation(invalidToken, pId2);
68. assertThat(result).isNotNull();
69. assertThat(result.getSuccess()).isFalse();
70. assertThat(result.getMessage()).isEqualTo("关联信息添加失败，请检查登陆状态！");
71. verify(associationMapper, times(0)).insert(any(Association.class));
72. }

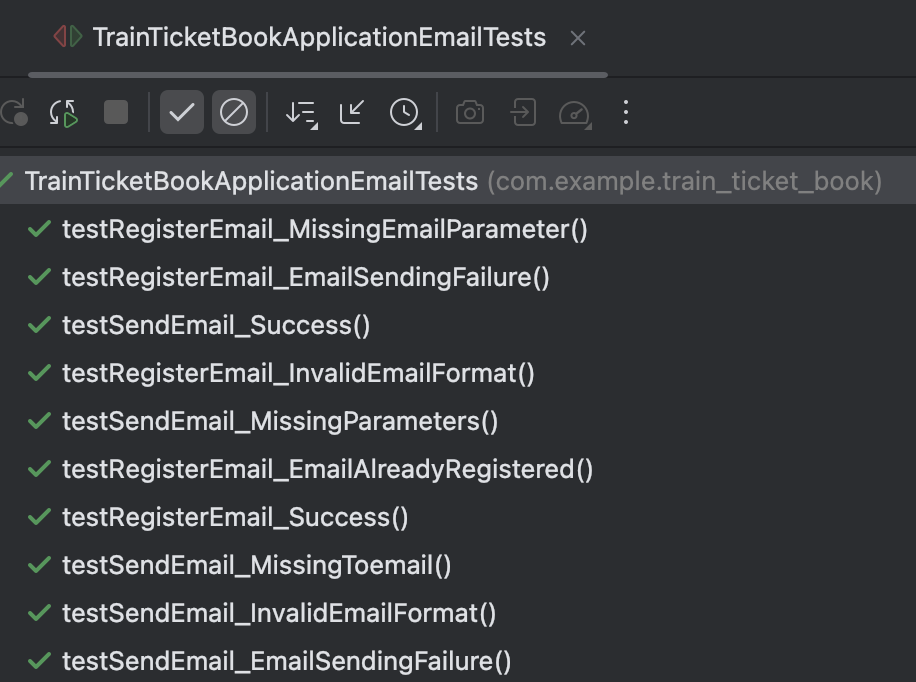


图5.8 TDD测试中的用户关联信息管理绿灯区图

### 5.8 管理员测试

对于管理员登录进行测试，具体的测试用例必须覆盖以下组合：

① username == null，password == null

② username == ""，password == ""

③ username != null，password != null

④ username != ""，password != ""

⑤ adminMapper.selectOne(new LambdaQueryWrapper<Admin>().eq(Admin::getUser, username)) == null

⑥ adminMapper.selectOne(new LambdaQueryWrapper<Admin>().eq(Admin::getUser, username)) != null

⑦ password 与数据库中的密码匹配

⑧ password 与数据库中的密码不匹配

⑨ JwtUtils.generateToken(username) 成功生成 token

⑩ JwtUtils.generateToken(username) 抛出异常

具体的测试用例如下所示：

① 测试管理员登录 - 用户名和密码均为空

username == ""，password == ""，Result.code == 500，message == "用户不存在，请检查用户名是否正确！" 覆盖①,②,⑤

② 测试管理员登录 - 用户名为空

username == ""，password == "adminPass123"，Result.code == 500，message == "用户不存在，请检查用户名是否正确！" 覆盖①,⑤

③ 测试管理员登录 - 密码为空

username == "adminUser"，password == ""，Result.code == 500，message == "密码错误" 覆盖③,⑧

④ 测试管理员登录 - 用户名和密码均为 null

username == null，password == null，Result.code == 500，message == "用户不存在，请检查用户名是否正确！" 覆盖①,⑤

⑤ 测试管理员登录 - 用户名为空格

username == " ", password == "adminPass123"，Result.code == 500，message == "用户不存在，请检查用户名是否正确！" 覆盖①,⑤

⑥ 测试管理员登录 - 密码为空格

username == "adminUser"，password == " ", Result.code == 500，message == "密码错误" 覆盖③,⑧

⑦ 测试管理员登录 - 用户名正确，密码正确

username == "adminUser"，password == "adminPass123"，Result.code == 200，返回的 token == "admin-token" 覆盖③,④,⑥,⑦,⑨

⑧ 测试管理员登录 - 用户名正确，密码错误

username == "adminUser"，password == "wrongPassword"，Result.code == 500，message == "密码错误" 覆盖③,④,⑥,⑧

⑨ 测试管理员登录 - 用户名不存在

username == "nonExistentUser"，password == "anyPassword"，Result.code == 500，message == "用户不存在，请检查用户名是否正确！" 覆盖③,⑤

⑩ 测试管理员登录 - 用户名包含特殊字符

username == "admin!@#", password == "adminPass123"，Result.code == 200，返回的 token == "special-admin-token" 覆盖③,④,⑥,⑦,⑨

部分测试代码如下所示：

1. @Test
2. public void testInvalidPasswordLogin() throws Exception {
3. Login login = new Login();
4. login.setUsername(adminUsername);
5. login.setPassword("wrongPassword");
6. mockMvc.perform(post("/admin/login")
7. .contentType(MediaType.APPLICATION\_JSON)
8. .content(toJson(login)))
9. .andExpect(status().isOk())
10. .andExpect(jsonPath("$.code").value(500))
11. .andExpect(jsonPath("$.message").value("密码错误"))
12. .andExpect(jsonPath("$.data").doesNotExist());
13. }
14. @Test
15. public void testNonExistentUserLogin() throws Exception {
16. Login login = new Login();
17. login.setUsername("nonExistentUser");
18. login.setPassword("anyPassword");
19. mockMvc.perform(post("/admin/login")
20. .contentType(MediaType.APPLICATION\_JSON)
21. .content(toJson(login)))
22. .andExpect(status().isOk())
23. .andExpect(jsonPath("$.code").value(500))
24. .andExpect(jsonPath("$.message").value("用户不存在，请检查用户名是否正确！"))
25. .andExpect(jsonPath("$.data").doesNotExist());
26. }
27. @Test
28. public void testEmptyUsernameLogin() throws Exception {
29. Login login = new Login();
30. login.setUsername("");
31. login.setPassword(adminPassword);
32. mockMvc.perform(post("/admin/login")
33. .contentType(MediaType.APPLICATION\_JSON)
34. .content(toJson(login)))
35. .andExpect(status().isOk())
36. .andExpect(jsonPath("$.code").value(500))
37. .andExpect(jsonPath("$.message").value("用户不存在，请检查用户名是否正确！"))
38. .andExpect(jsonPath("$.data").doesNotExist());
39. }
40. @Test
41. public void testEmptyPasswordLogin() throws Exception {
42. Login login = new Login();
43. login.setUsername(adminUsername);
44. login.setPassword("");
45. mockMvc.perform(post("/admin/login")
46. .contentType(MediaType.APPLICATION\_JSON)
47. .content(toJson(login)))
48. .andExpect(status().isOk())
49. .andExpect(jsonPath("$.code").value(500))
50. .andExpect(jsonPath("$.message").value("密码错误"))
51. .andExpect(jsonPath("$.data").doesNotExist());
52. }
53. @Test
54. public void testEmptyFieldsLogin() throws Exception {
55. Login login = new Login();
56. login.setUsername("");
57. login.setPassword("");
58. mockMvc.perform(post("/admin/login")
59. .contentType(MediaType.APPLICATION\_JSON)
60. .content(toJson(login)))
61. .andExpect(status().isOk())
62. .andExpect(jsonPath("$.code").value(500))
63. .andExpect(jsonPath("$.message").value("用户不存在，请检查用户名是否正确！"))
64. .andExpect(jsonPath("$.data").doesNotExist());
65. }
66. @Test
67. public void testNullUsernameLogin() throws Exception {
68. Login login = new Login();
69. login.setUsername(null);
70. login.setPassword(adminPassword);
71. mockMvc.perform(post("/admin/login")
72. .contentType(MediaType.APPLICATION\_JSON)
73. .content(toJson(login)))
74. .andExpect(status().isOk())
75. .andExpect(jsonPath("$.code").value(500))
76. .andExpect(jsonPath("$.message").value("用户不存在，请检查用户名是否正确！"))
77. .andExpect(jsonPath("$.data").doesNotExist());
78. }

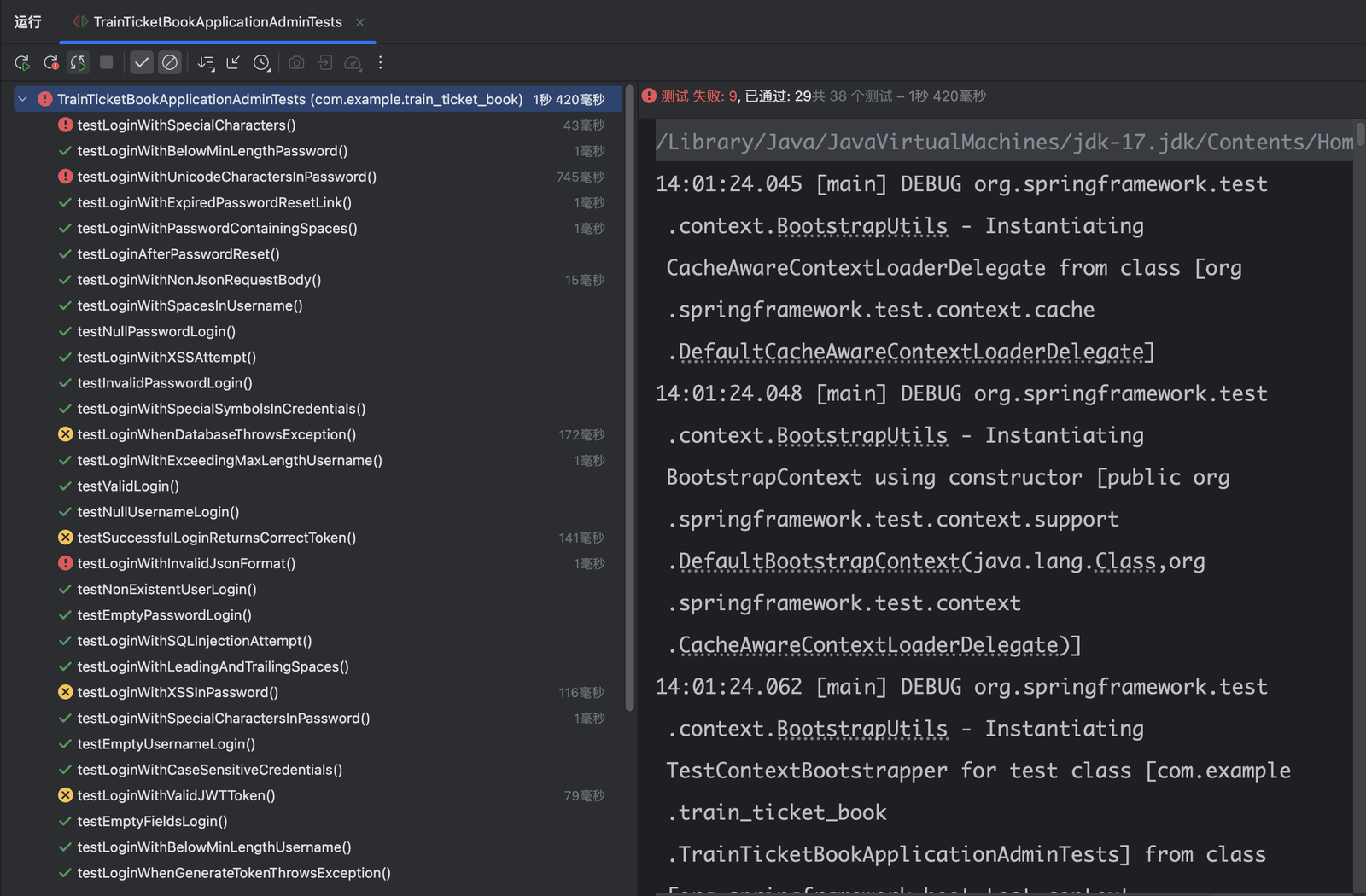


图5.9 TDD测试中的管理员功能绿灯区图

# 6系统实现

## 6.1 开发平台和开发环境介绍

开发平台：IDEA 2024.3.1 和WebStorm 2024.3.1

开发环境：JDK1.8 +MySQL

(1) IntelliJ IDEA 是由 JetBrains 开发的一款强大的、基于 Java 的集成开发环境（Integrated Development Environment，IDE）。作为一款商业化产品，它以智能、高效、便捷著称，专为提升开发效率而设计。IntelliJ IDEA 不仅支持 Java，还支持多种编程语言，例如 Kotlin、Scala、Python、PHP 等，适用于多种应用程序开发。它是目前最受欢迎的跨平台应用程序开发环境之一。

IntelliJ IDEA 提供了广泛的功能和工具，包括智能代码补全、强大的代码分析和自动化重构，使开发者能够以更高效、更直观的方式编写代码。此外，IntelliJ IDEA 内置了丰富的调试工具、版本控制集成（如 Git）、测试支持以及终端，能够为开发者提供一站式的开发体验。

本实验中开发的列车票务预订系统就是在 IntelliJ IDEA 2023.2 平台上采用 Java 语言开发的。

Java是一门面向对象编程语言，不仅吸收了C++语言的各种优点，还摒弃了C++里难以理解的多继承、指针等概念，因此Java语言具有功能强大和简单易用两个特征。Java语言作为静态面向对象编程语言的代表，极好地实现了面向对象理论，允许程序员以优雅的思维方式进行复杂的编程。

(2)MySQL是一种关系数据库管理系统，由瑞典MySQL AB公司开发，目前属于Oracle旗下产品，是最流行的关系型数据库管理系统之一。MySQL 软件采用了双授权政策，分为社区版和商业版，由于其体积小、速度快、总体拥有成本低，尤其是开放源码这一特点，一般中小型网站的开发都选择 MySQL 作为网站数据库。

运行环境：Apache Tomcat/9.0.68

列车票务预订系统属于Web应用系统，运行时需要投放在相应服务器上，方可在浏览器中使用。对于正式投放市场的情况，平台需额外租用服务器投放系统程序来满足用户访问的条件。而对于当前的演示部分，用户仅需使用Tomcat应用软件(一个免费的开放源代码的Web 应用服务器)即可在本地网页访问该平台。

## 6.2 编码

在软件工程中，编码是将软件设计结果转化为使用特定程序设计语言编写的程序的过程。通常将编码和测试统称为实现阶段。

编码阶段将软件设计的抽象概念转换为具体的可执行代码。这一阶段的目标是将设计文档中的算法、数据结构和逻辑转化为计算机可以理解和执行的指令。编码过程中，开发人员需要遵循编码规范和最佳实践，确保代码的可读性、可维护性和可重用性。

编码的质量主要依赖于前期软件设计的质量。良好的软件设计能够提供清晰的结构和逻辑，使得编码过程更加顺利和有效。因此，在编码之前，开发团队应该进行充分的软件设计和规划，确保设计的准确性和一致性。

编码阶段也包括代码的版本控制和文档化工作，以便于团队成员之间的协作和代码的维护。编码完成后，代码需要进行测试和调试，以验证其功能的正确性和性能的可接受性。编码和测试作为实现的重要组成部分，共同确保软件系统的质量和可靠性。

## 6.3 前台界面与后台数据库连接代码

在基于springboot框架的列车票务预订系统中，数据库连接部分主要通过Mybatis-Plus来实现。下面是通过读取"application.properties"文件中的实体配置信息并创建相应的数据库连接的主要代码示例：

spring.datasource.type=com.alibaba.druid.pool.DruidDataSource

spring.datasource.driver-class-name=com.mysql.jdbc.Driver

spring.datasource.url=jdbc:mysql://localhost:3306/ticket?useUnicode=true&characterEncoding=utf-8&useSSL=false

spring.datasource.username=root

spring.datasource.password=123456

mybatis-plus.configuration.log-impl=org.apache.ibatis.logging.stdout.StdOutImpl

在上述代码中，我们使用了MySQL数据库作为示例，并提供了数据库的URL、用户名和密码。Mybatis-Plus可以通过这些信息，自动连接到数据库。在实际应用中，可以根据具体的数据库和配置信息进行相应的修改。

## 6.4各模块功能的实现

### 6.4.1用户注册模块实现

该模块用于实现用户注册的功能，用户需输入个人信息，申请提交，满足数据库中设置的各项约束性条件才能注册成功。

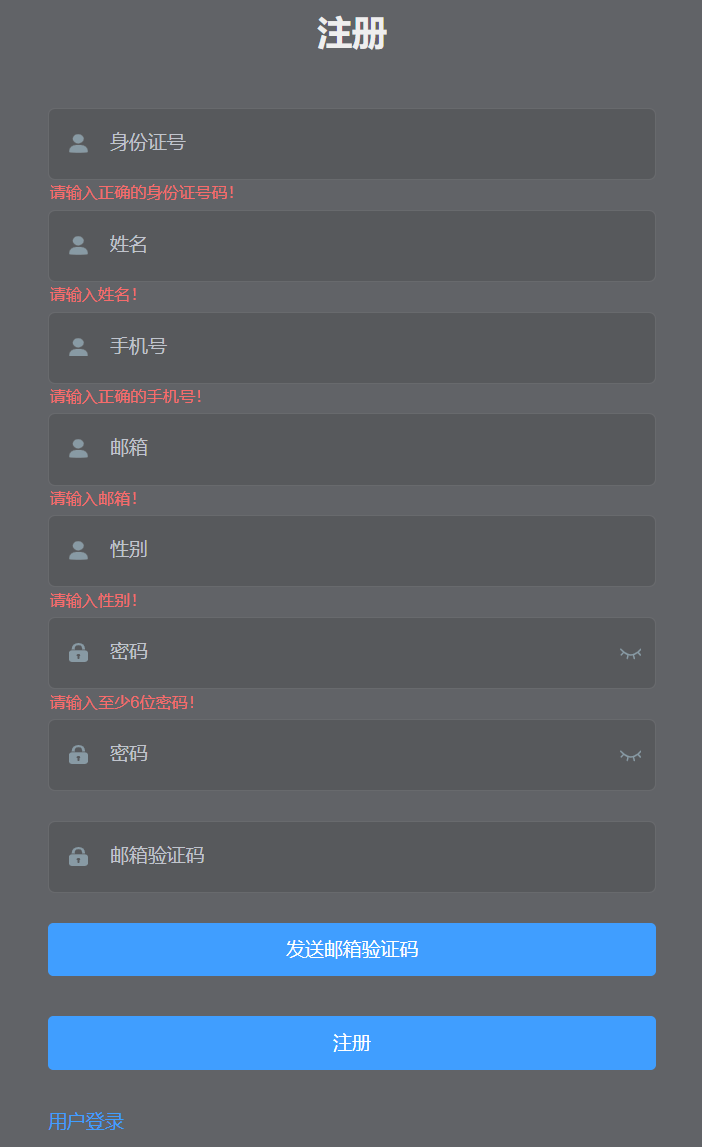
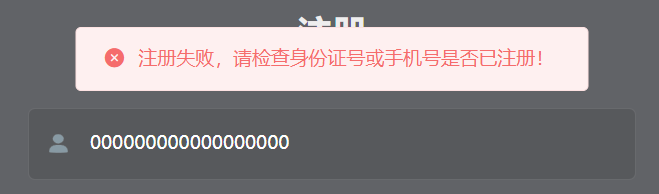
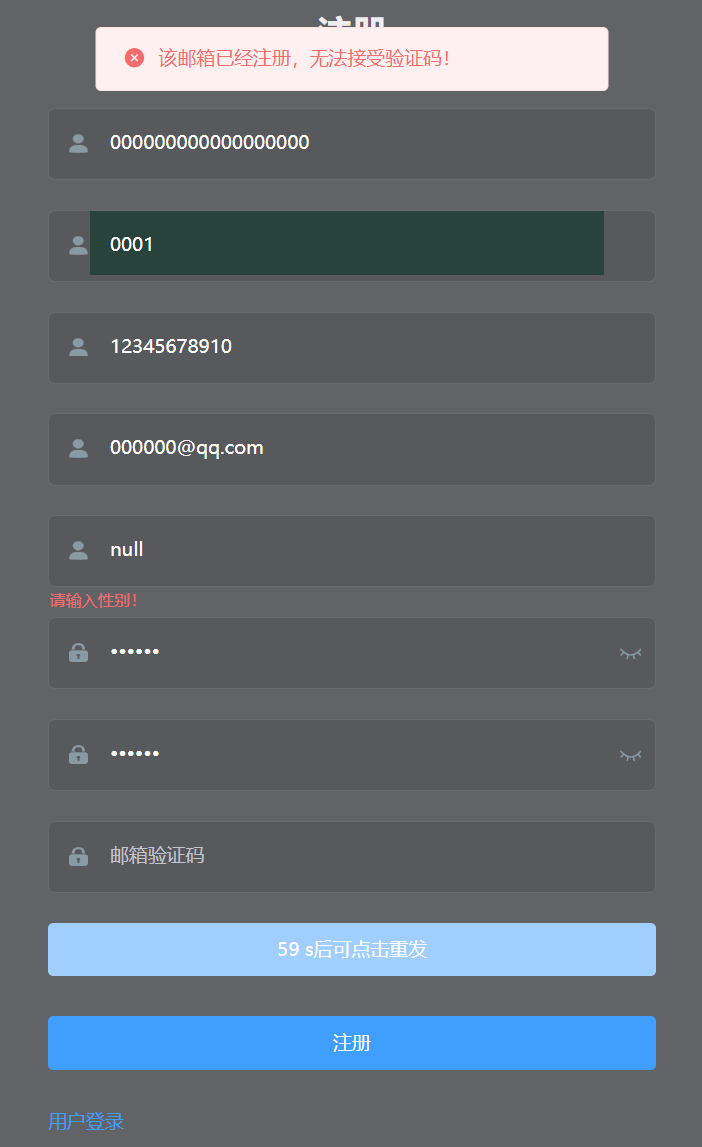
 



图6.1 用户注册模块实现效果图（数据非空约束、唯一性约束）

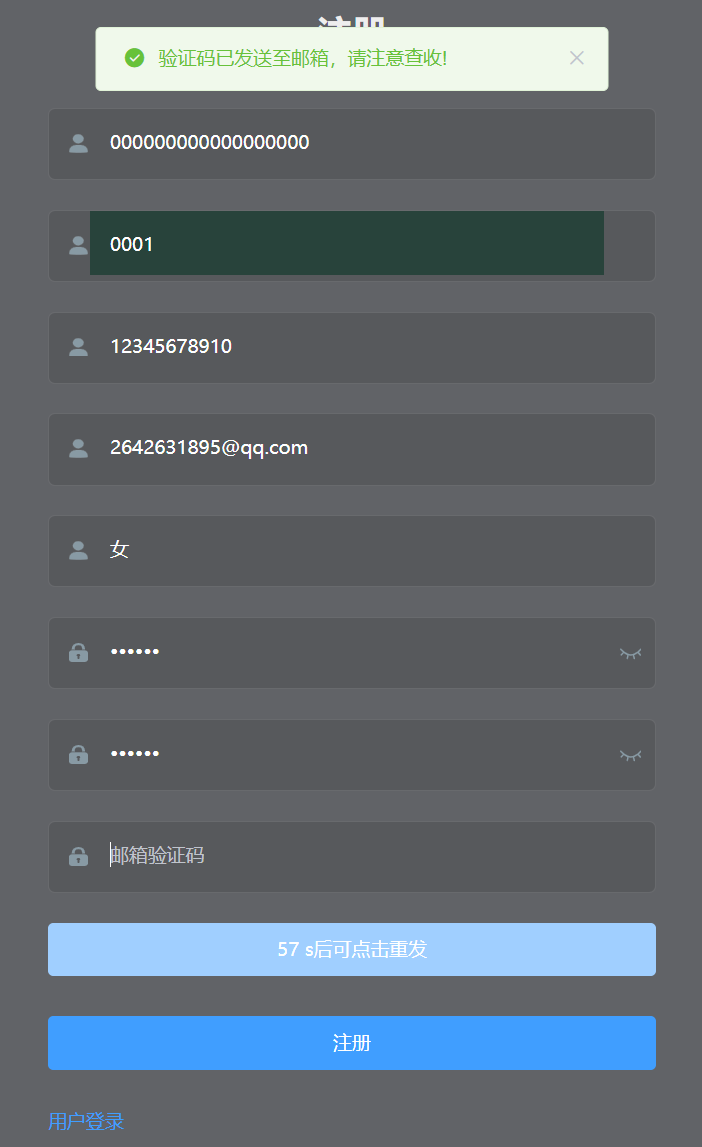




图6.2 用户注册模块实现效果图（信息提交、接受邮件验证码图）



图6.3 用户注册模块实现效果图（注册成功图）

### 6.4.2用户登录模块实现

该模块用于实现用户登录的功能，输入账号、密码和验证码，通过信息验证后方可登录成功

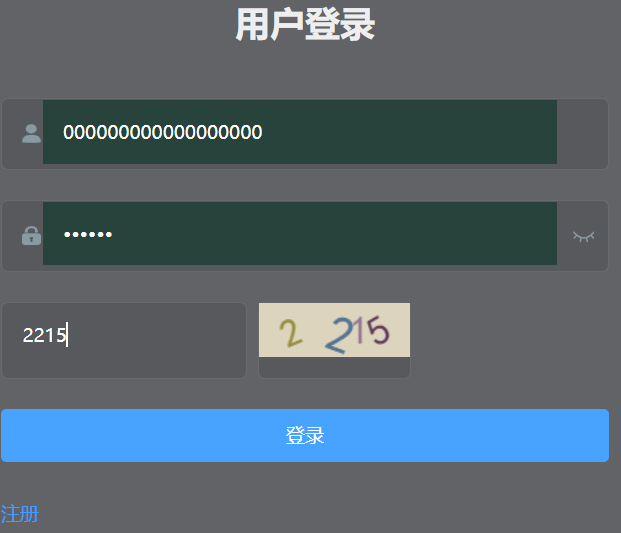


图6.4用户登录模块实现效果图

### 6.4.3用户查询个人信息功能模块

该模块实现用户查阅个人信息的功能，系统会用户本人的各项信息（身份证号、姓名、性别、手机号、邮箱、密码）呈现在前端界面，用户也可在此模块进行对个人资料的编辑更改操作。

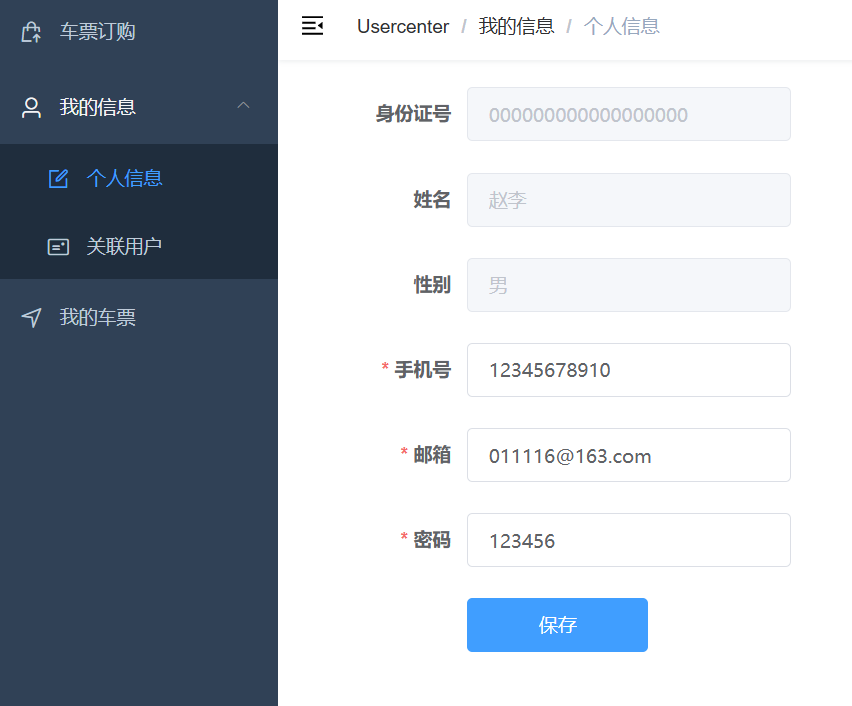


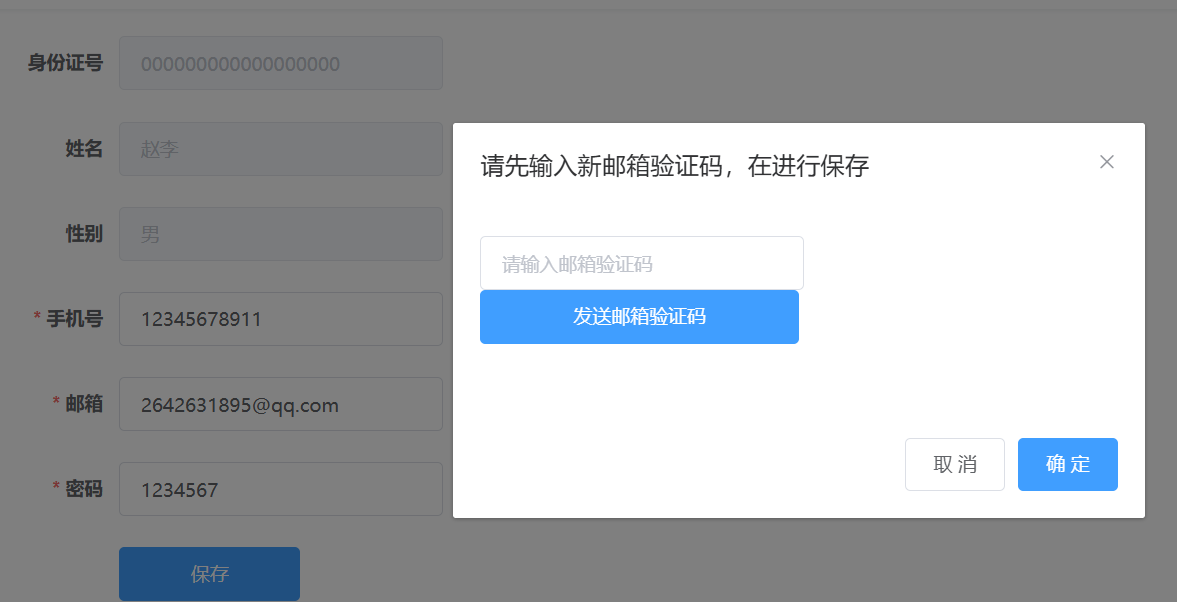
图6.5 用户查阅个人资料模块实现效果图

### 6.4.4用户管理个人信息功能模块

本模块实现了用户修改个人资料的功能，按需更改（身份证号码、手机号码、性别不可自行修改），点击保存后方可添加成功



图6.6用户修改个人资料功能模块实现效果图（修改信息合法性判定图）



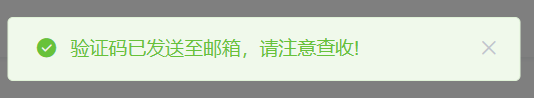


图6.7用户修改个人资料功能模块实现效果图（更改邮箱并重新接受验证码图）

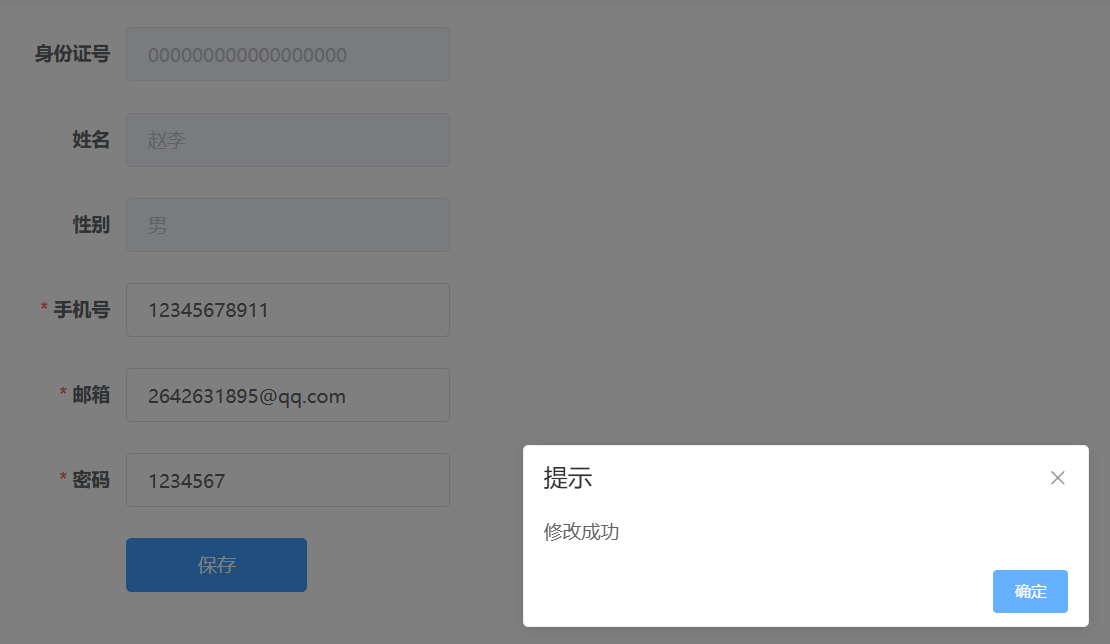


图6.8 用户修改个人资料功能模块实现效果图（个人资料修改成功图）

### 6.4.5用户查阅关联用户模块

该模块实现用户查阅关联乘客的功能，系统会将全部关联用户及其信息（电话号码、姓名、身份证号）呈现在前端界面，用户也可在此模块进行对关联用户的增删操作。



图5.8用户查阅关联乘客模块实现效果图

### 6.4.6用户管理关联用户模块

本模块实现了用户管理关联用户的功能，可以实现对关联用户的增删操作：

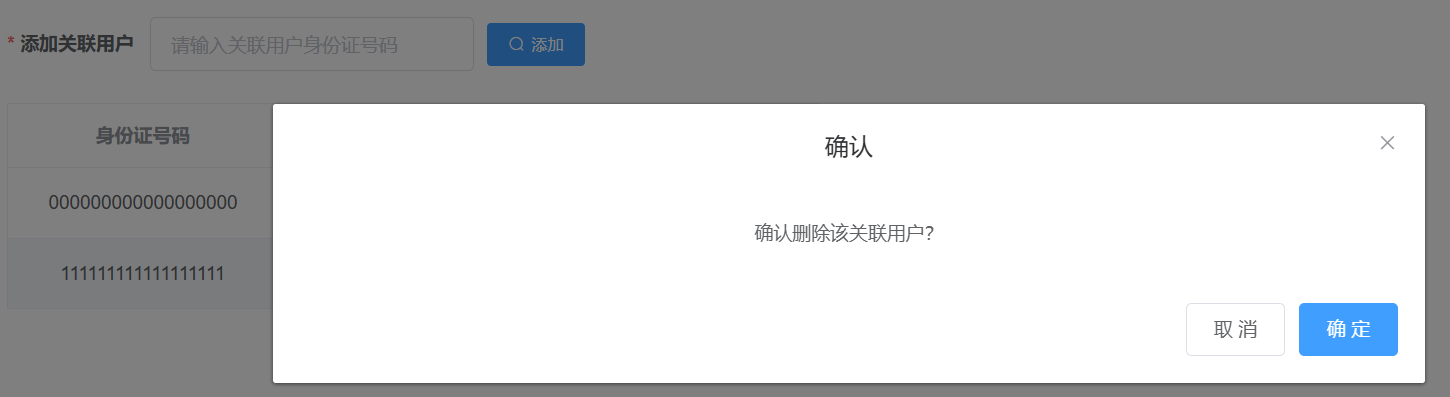






图6.9用户管理关联乘客的功能模块实现效果图（关联用户的删除）

通过搜索关联用户的身份证号码来添加关联用户。当数据库中不存在某用户或已经存在该关联用户时会提示添加失败。

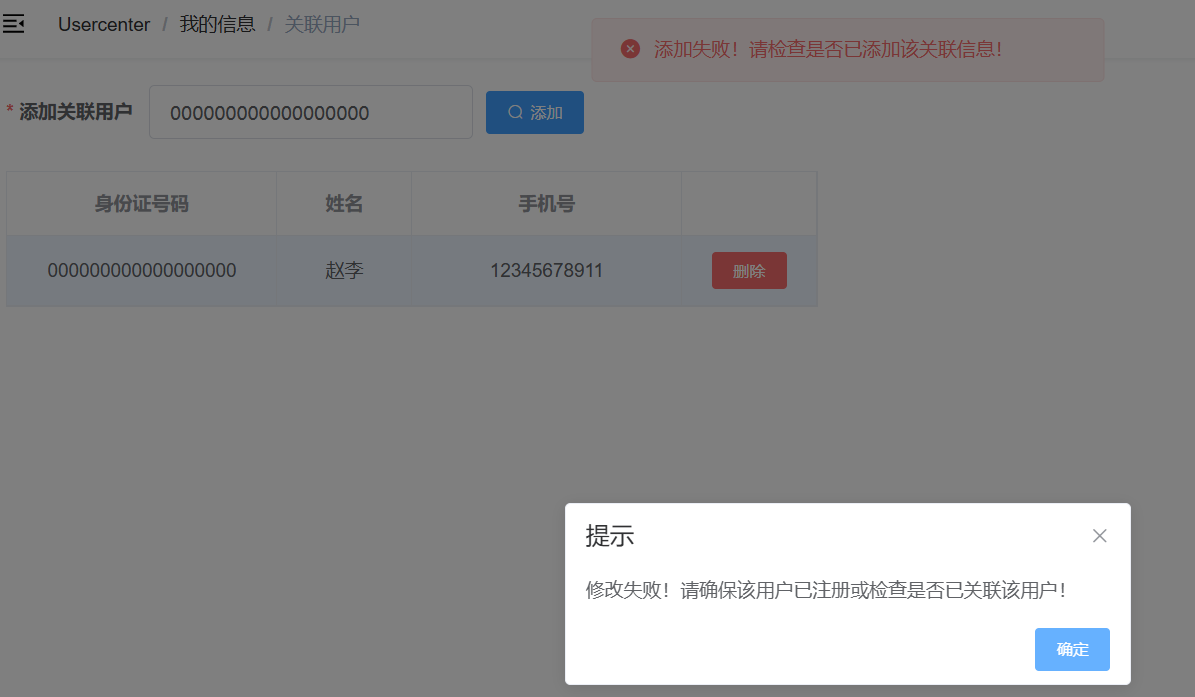


图6.10用户管理关联乘客的功能模块实现效果图（关联用户的添加失败图）

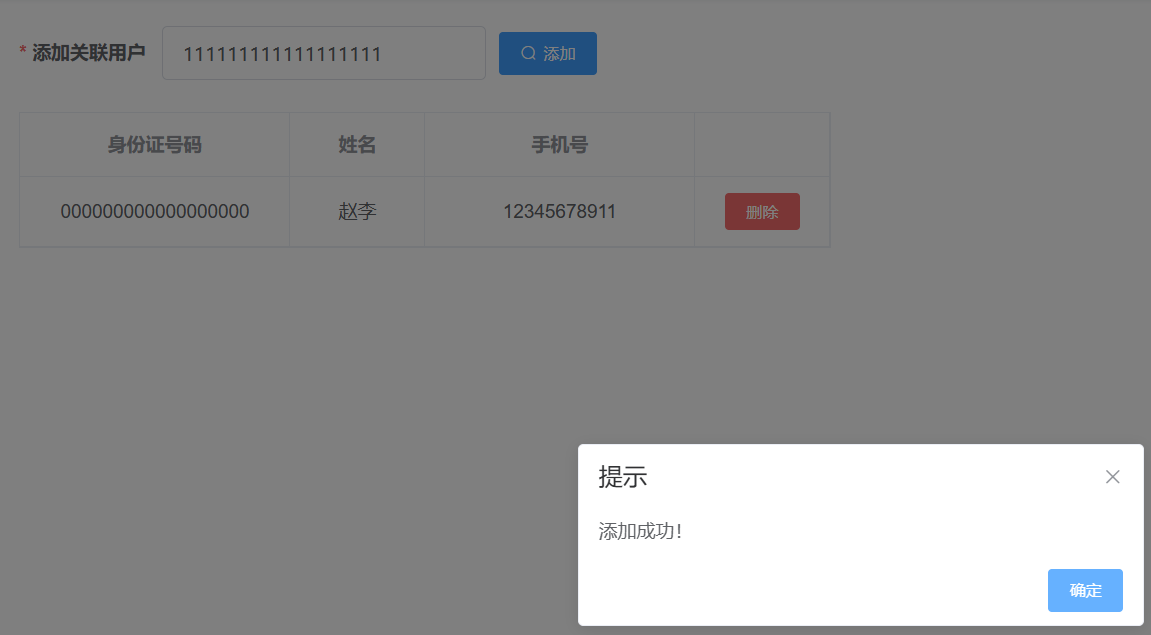




图6.11用户管理关联乘客的功能模块实现效果图（关联用户的添加成功图）

### 6.4.7用户搜索并查看车次功能模块

本模块实现了用户搜索并查看车次的功能，需输入上车站和下车站，点击搜索后方可查阅符合检索条件的列车并进行购买。



图6.12用户搜索并查看车次功能模块实现效果图

### 6.4.8用户下单购票模块

本模块实现了用户订购车票的功能，需选择乘车区间并勾选乘车人，点击“确定”后即可下单生成车票。



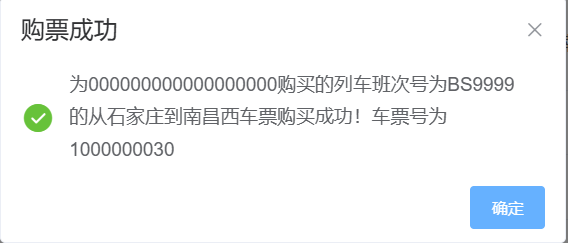


图6.13用户下单购票功能模块实现效果图（选择乘车区间并勾选乘车人并点击确定图）



图6.14用户下单购票功能模块实现效果图（成功下单后生成车票图）

### 6.4.9用户对订单付款或取消功能模块



图6.15用户订单取消（退票）模块实现效果图

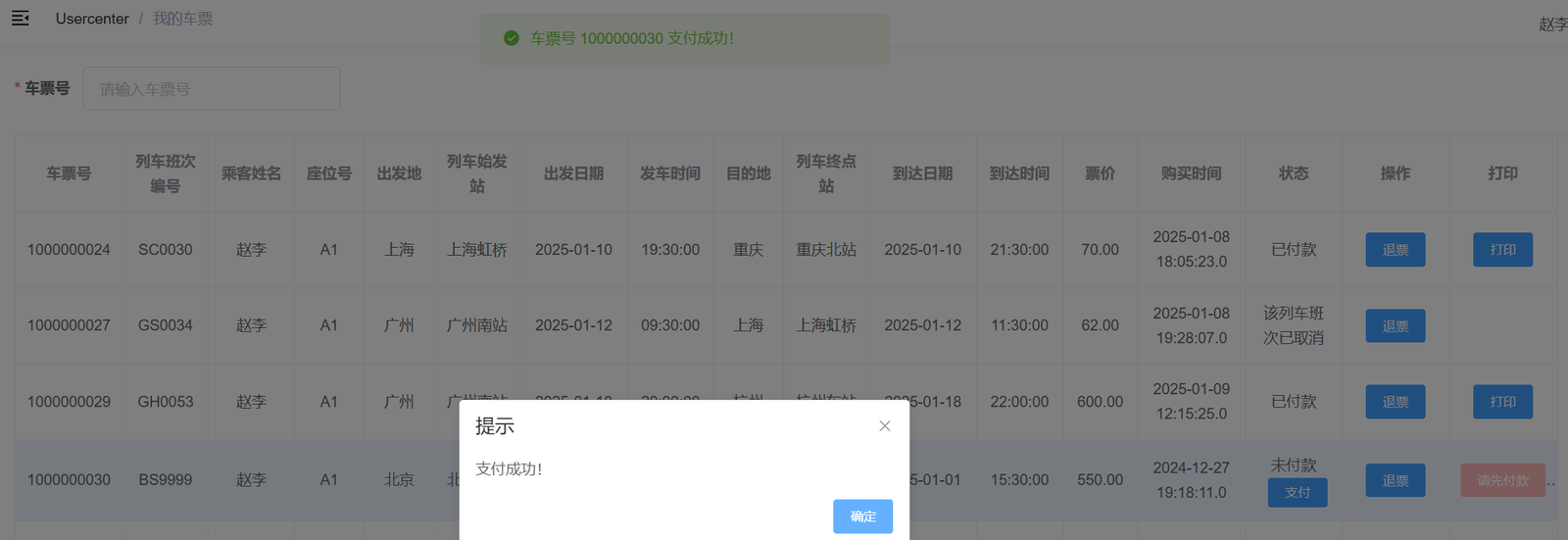


图6.16用户订单付款模块实现效果图

### 6.4.10用户查看个人车票模块

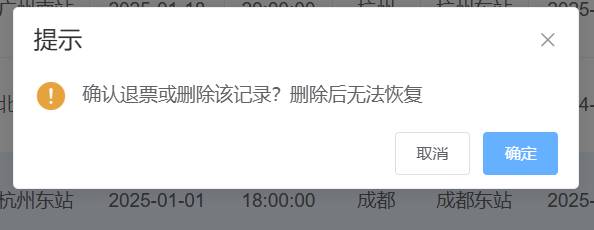
该模块实现用户查阅个人车票列表的功能，系统会将全部已购车票及其信息（车票号、列车班次编号、乘客姓名、座位号、出发地、出发日期和时间、目的地、到达日期和时间、票价、购买时间、状态等）呈现在前端界面。其中，如果车票出发日期和时间早于系统当前时间会显示车票已过期，这时用户可以对该车票进行删除操作。



图6.17用户查看个人车票列表模块实现效果图

### 6.4.11用户删除车票或退票模块

该模块实现用户对于已经过期的车票进行删除功能和对已经购买的车票进行退票功能。用户在查阅个人车票的表单中可以选择对应的车票进行删除或者退票：



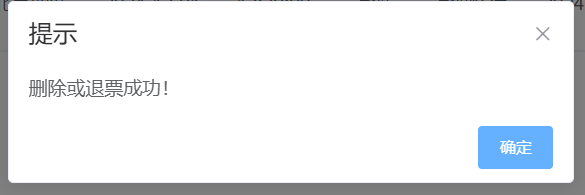


图6.18用户删除车票或退票模块实现效果图

### 6.4.12管理员登录模块

该模块用于实现管理员登录系统的功能，管理员通过输入用户名、密码、验证码，模块会进行数据库校对，当输入的验证码和密码符合其用户名且输入正确时，可以成功进入本系统。





图6.19管理员登录模块实现效果图

### 6.4.13管理员查看列车车次模块

该模块实现管理员查阅现有车次的功能，系统会将所有的列车班次及其信息呈现在前端界面，管理员也可在此模块进行对车次的删除操作。

图6.20管理员查看列车车次模块实现效果图

### 6.4.14管理员管理列车车次信息模块

该模块实现管理员删除现有列车车次的功能，其中以删除BG0037列车为例：

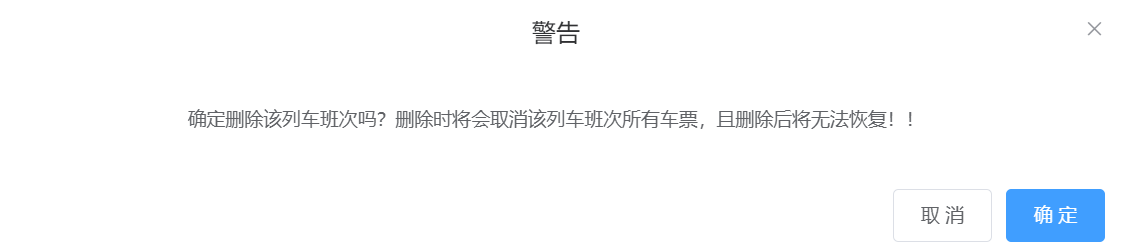




图6.21管理员管理列车车次模块实现效果图（删除列车）

### 6.4.15管理员添加列车信息模块

该模块实现管理员添加航班的功能，需分别输入起点、终点、始发站、终点站、发车时间、到达时间，点击提交后方可添加成功





图6.22管理员添加列车车次模块实现效果图（添加列车）

### 6.4.16管理员查阅车票功能模块

该模块实现管理员查阅现有车票的功能，系统会将所有的车票及其信息呈现在前端界面，管理员也可在此模块进行对车票的删除操作。



图6.23管理员查看车票模块实现效果图

### 6.4.17管理员管理车票功能模块

该模块实现管理员删除现有车票的功能，其中以删除1000000029车票为例：

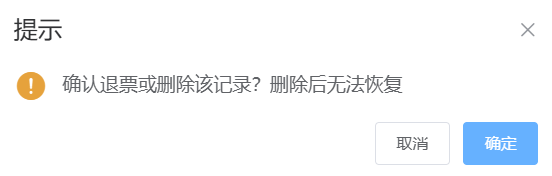




图6.24管理员管理车票模块实现效果图

### 6.4.18管理员添加用户车票模块

本模块实现了管理员添加车票的功能，需分别输入乘客身份证号、列车班次号、日期，点击提交后方可添加成功。



当乘客身份证号或列车班次号不存在时，系统将提示添加失败。



图6.25管理员添加车票模块实现效果图（添加失败图）

输入信息与数据库中的信息相对应后，系统将提示添加成功。



图6.26管理员添加车票模块实现效果图（添加成功图）

### 6.4.19管理员查阅用户信息模块

该模块实现管理员查阅现有用户的信息的功能，系统会将所有的用户及其信息（身份证号、姓名、手机号等）呈现在前端界面，管理员也可在此模块进行对用户的删除操作。



图6.27管理员查阅用户信息模块实现效果图

### 6.4.20管理员添加用户模块

本模块实现了管理员添加用户的功能，需分别输入乘客身份证号、手机号等身份信息，点击提交后方可添加成功。



图6.28管理员添加用户模块实现效果图（界面图）

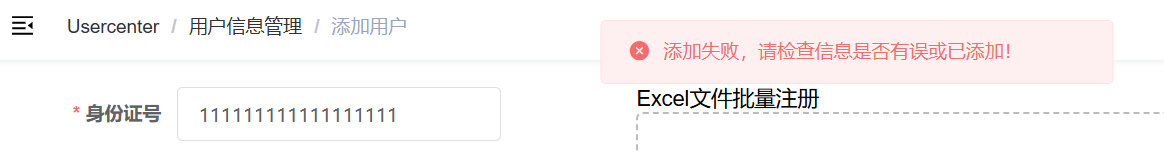


图6.29管理员添加用户模块实现效果图（添加失败图）

当乘客身份证号或手机号等信息已存在时，系统将提示添加失败。



图6.30管理员添加用户模块实现效果图（添加成功图）

# 7 维护

维护是软件生命周期中不可或缺的重要环节，随着系统运行时间的增加以及用户需求的变化，系统需要不断进行维护以确保其正常运行、可靠性和安全性。

在软件维护阶段，维护人员需要进行以下工作：

1. 故障修复：及时处理和修复软件系统在运行过程中发现的错误和故障，确保系统的稳定性和可靠性。
2. 改进和优化：根据用户反馈和需求变化，对软件系统进行改进和优化，提升系统的性能和用户体验。
3. 更新和升级：根据用户环境的变化和新的需求，对软件进行局部更新和升级，以适应新的功能和技术要求。
4. 配置管理：管理软件系统的配置信息，确保系统的正确配置和版本控制。
5. 文档维护：更新和维护软件系统的文档，包括用户手册、操作指南等，以保持文档与实际系统的一致性。

通过软件维护工作，可以确保软件系统在使用过程中持续运行，并满足用户的需求和期望。软件维护是软件生命周期中不可或缺的一部分，对于保持软件系统的价值和可持续发展具有重要意义。

## 7.1 系统维护过程

我们的系统维护的过程包括以下几个阶段：

1. 问题检测与报告：通过系统日志、用户反馈或自动化监控工具发现问题。输入包括系统日志报告、用户反馈和监控工具生成的警告，输出为问题报告（包含问题描述、影响范围和优先级）。
2. 问题分析与分类：对问题进行详细分析，确定其原因和影响范围，并根据优先级划分修复顺序，同时判断问题来源（如代码错误、硬件故障或配置错误）。输出为问题分类报告和修复需求分析文档。
3. 制定维护计划：根据问题优先级制定维护方案和时间表，包括解决方案分析、计划时间安排（避免业务高峰期进行高风险操作）。输出为维护计划文档（包括目标、时间、负责人和风险预估）。
4. 问题解决与测试：按照维护计划进行问题修复，并对修复内容进行功能测试、性能测试及回归测试，确保问题被完全解决且无新的问题引入。输出为修复后的系统版本和测试报告。
5. 发布与验收：将修复内容部署到生产环境，并进行验收，确保修复的正确性和稳定性。发布后进行用户确认，输出发布记录和用户验收文档。
6. 维护记录与总结：对维护过程进行总结并记录经验教训，归档维护过程中发现的问题及解决方法，同时为后续维护提供参考。输出维护记录和问题预防建议。

## 7.2 系统维护策略

列车票务管理系统依赖于数据库数据的实时更新。如果数据库的数据未能及时更新到相应的表格，或者触发器失效，应立即采取紧急措施。具体措施如下： 1.建立备用数据库

在主数据库失效时，备用数据库可用于确保系统的正常运行。备用数据库应与主数据库保持实时同步，通过使用双向复制实现高可用性。当主数据库发生故障时，系统能够无缝切换到备用数据库以提供连续服务，并在主数据库恢复后进行数据补全。

2.使用云备份进行数据库备份

将数据库的全量和增量备份存储在云端，以确保数据的安全性和可靠性。云备份应支持自动化的备份任务，并具备高可用性和灾难恢复能力。在发生数据丢失时，系统可以快速从云端恢复到最新的可用版本。

3.自动调用和保存数据

在系统运行过程中，列车票务系统应每天自动调用关键数据（列车信息、乘客订单、票务状态等），并保存到临时存储（本地缓存文件或Redis内存数据库）中。数据更新频率可根据业务需求设置为每30分钟或每小时更新一次，以确保数据的实时性和准确性。同时，针对核心票务数据，可在临时文档中建立快照机制，以便在突发情况下快速恢复。

4.定期将云备份数据转存到固定设备

每周将云备份的数据转存到磁盘或其他固定设备上，形成多层次的数据备份策略。通过定期的离线备份测试，确保离线数据的可用性与完整性，从而提升系统的容灾能力和数据恢复速度。

5.建立日志记录与异常监控

系统应建立详细的日志记录机制，覆盖以下内容：

数据库操作日志：记录增删改查操作，包括操作用户、操作时间及相关SQL语句。

系统异常日志：包括触发器执行失败、事务回滚等异常信息。

审计日志：记录票务系统的敏感数据访问记录。

日志应通过实时监控系统进行分析，及时发现异常事件，并设置告警机制，帮助快速定位问题并进行修复。

6.增强触发器与数据校验机制

确保数据库触发器能够按预期工作，并定期检查触发器的有效性。通过编写冗余校验机制防止因触发器失效而导致的数据异常。同时，建立定期触发器测试计划，模拟各种场景以验证其可靠性。

7.启用灾难恢复演练计划

每季度组织一次全系统的灾难恢复演练，模拟主数据库失效、数据丢失、备份恢复等场景。通过演练测试数据恢复流程的可操作性和恢复速度，并优化应急预案，确保系统能够在突发情况下快速恢复。

8.优化数据库架构与性能监控

在高并发场景下，列车票务系统的数据库可能因锁等待、连接超时等问题影响实时更新。为此需要优化数据库索引和查询语句，减少数据写入和读取延迟；启用数据库性能监控工具监测慢查询、死锁等问题，及时优化数据库操作；为高并发操作提供限流与降级措施，保障数据库稳定运行。

# 8总结与体会

## 8.1 总结

在本次软件工程大作业练习中，我们团队四位成员通过紧密合作，深入体验了软件工程的各个环节，包括需求分析、系统设计、模块实现、测试与维护等。整个项目过程中，我们深刻认识到软件工程方法论在指导项目开发中的重要性，尤其是在前期规划和设计阶段，充分的准备工作为后续的开发奠定了坚实的基础。

在项目推进过程中，我们遇到了一些设计上的挑战，尤其是在系统后期发现前期设计缺陷时，虽然由于时间和系统复杂性的限制，无法完全修复这些问题，但这也成为了宝贵的学习经验。通过这些经历，我们更加理解了持续改进和灵活应对的重要性。尽管存在一些遗憾，我们最终还是实现了系统的大部分功能，构建了相对完善的框架，并确保了购票操作流程的便利性和人性化。

技术方面，我们后端采用了基于Java的Spring Boot框架，负责处理用户的业务操作和数据库交互；前端则使用了Vue.js框架，为用户提供了视觉友好的服务界面。同时，我们在系统中集成了必要的错误提示功能，帮助用户更好地掌握平台的使用方法。这次项目不仅提升了我们的编程能力和问题解决能力，也增强了我们的团队协作和项目管理技能，为未来的软件开发项目积累了宝贵的经验。

## 8.2 体会

**xxx心得体会：**

在本次软件工程课程设计中，我参与了测试模块的实现以及系统核心功能的开发任务。这次课程设计让我全面了解了软件工程的开发流程，包括需求分析、系统设计、模块实现、测试与迭代等环节。在任务中，我特别关注了测试驱动开发（TDD）这一技术的学习与实践。通过编写测试用例并引导功能开发，我们有效地提高了代码质量，减少了后期维护的难度，并确保了系统在开发过程中始终满足设计要求。

此外，我在列车电子票务系统的搭建过程中，不仅负责从核心功能模块设计到测试模块集成的工作，还主动参与了代码优化、异常处理机制设计以及接口测试等环节。通过这一过程，我深刻体会到了团队协作的重要性。团队中采用了敏捷开发方法，我们通过每日站会明确任务分工，定期评审项目进展，并根据实际需求动态调整开发计划，最终高效地完成了各阶段任务。

在课程设计中，我还学会了使用版本控制工具（如Git）进行协作开发，提高了团队合作的效率。此外，我对面向对象设计原则（如SOLID原则）的理解得到了进一步深化，并能够将其灵活运用到代码设计中。与此同时，面对复杂需求和实际问题，我也培养了快速查阅资料和解决技术难题的能力。

通过这次课设，我不仅巩固了软件工程的理论基础，还提升了自己的编程能力、问题解决能力和协作能力。更重要的是，我认识到开发过程中持续改进和迭代优化的重要性，这次经历让我对软件开发的全流程有了更加全面和深刻的理解，为未来在实际项目中应用所学知识积累了宝贵的经验。总之，这次课程设计不仅是对理论学习的一次实践验证，更是一场关于个人成长和能力提升的全面锻炼。

**xxx心得体会：**

在本次软件工程课设中，全身心投入到总体设计的IPO表设计与系统实现环节，负责将理论构想转化为实际代码，这一过程让我收获颇丰。

起初，面对IPO表的设计，需深入剖析每个模块的功能、输入、处理和输出，这使我对系统架构有了更清晰的认识。设计过程中，我不断权衡各种方案，力求让每个模块的逻辑严谨且高效，如同精心雕琢一件艺术品，这个过程锻炼了我的逻辑思维能力，让我学会从整体到细节地规划系统。

进入系统实现阶段，编码过程充满挑战。我依据IPO表的设计蓝图，用代码将各个模块的功能逐一实现，这个过程不仅需要熟练掌握编程语言，还需注重代码的规范性和可读性，以方便后续的维护与扩展。每当成功实现一个功能模块，那种成就感便油然而生，激励我继续前行。

通过此次课设，我的编程能力得到了显著提升，对软件开发流程有了更深刻的理解。同时，也明白了团队协作的重要性，每个环节都紧密相连，需要与团队成员密切配合。在未来的学习与工作中，我将继续努力，不断提升自己在软件工程领域的专业素养。

**xxx心得体会：**

通过这次小组合作实验，我不仅对软件工程有了更深层次的理解，而且还得到了技术能力的提升。

在整个项目中，我们严格遵循了软件工程的流程，这一过程让我认识到，良好的前期规划对于项目的成功至关重要。通过前期的一步步讨论和改进，我们不仅能够清晰地表达系统的整体架构，还能有效地与团队成员沟通设计理念，确保每个人都对系统有共同的理解。

在项目中我主要负责了数据流图、数据字典和详细设计部分。制作数据流图和数据字典是一个细致入微的过程，它要求我们对每个数据项、文件以及处理过程进行详尽描述。这个任务为后续的数据库设计提供了明确的指导，同时也减少了因数据理解不同而产生的错误。在绘制模块流程图和PAD图时，我学会了如何将复杂的逻辑以图形化的方式展现出来，也体会到了不同图在表达时的优劣势。这不仅有助于我自己理清思路，也为团队其他成员提供了一个直观的理解途径。此外，编写伪代码进一步加深了我对算法逻辑的理解，也使得团队之间工作的一致性得到保证。

项目进展也并非一帆风顺，遇到问题时，团队成员之间的协作显得尤为重要。我们一起讨论解决方案，分享各自的见解，最终找到了最优的实施路径。这段经历让我深刻体会到，一个优秀的软件设计团队不仅要能提高工作效率，更要能激发每个人的潜力，创造更大的价值。

这次列车票务系统的设计是对所学软件工程方法论的一次综合应用，让我更深刻地理解到如何在一个完整的项目周期内，把每一个环节都做到精益求精。它不仅增强了我的专业技能，也培养了我的耐心和毅力，希望在未来能够继续运用这些宝贵的经验，完成更多优秀有意义的作品。

**xxx心得体会：**

在本次软件工程课程设计中，我深入学习并实践了测试驱动开发（TDD）以及项目管理的相关知识。具体参与了业务流程设计、系统数据建模等分析工作，并参与测试用例的设计。这些经验不仅丰富了我的技术能力，也提升了我的组织和领导能力。

在数据库设计阶段，我运用了实体关系建模的方法，对系统中的各个实体及其关系进行了详细分析和设计。通过与团队成员的讨论，我们确定了系统所需的主要数据实体，如用户、车票、列车等，并定义了它们之间的关联关系。为了确保数据库的规范化，我进行了多次范式检查，避免了数据冗余和更新异常，优化了数据库结构。这一过程极大地提升了我的数据建模能力和逻辑思维能力。

通过在功能开发之前编写测试用例，我能够明确每个模块的预期行为，从而指导代码的实现。这种方法有效地提升了代码的可靠性和可维护性，减少了潜在的错误和漏洞。在实际操作中，我学会了如何编写高质量的测试用例，确保每个功能点都经过充分验证。这促使我更加注重代码的设计，使其更加模块化和松耦合，便于后续的扩展和优化。

作为项目经理，我在团队协作和项目管理方面也得到了显著的成长。负责协调团队成员的任务分配和进度跟踪，我学会了如何有效地沟通和解决团队内部的问题。通过尝试制定详细的项目计划和时间表，我尽可能确保了各项任务能够按时完成，并及时调整计划以应对突发状况。此外，我还组织了定期的项目评审会议，评估项目进展并分享各自的工作成果，这不仅提高了团队的整体效率，也增强了成员之间的合作意识。

通过此次课程设计，我不仅巩固了软件工程的理论知识，还在TDD和项目管理方面积累了宝贵的实践经验。这些技能使我在面对复杂项目时，能够更加自信和从容地应对各种挑战。这些经验不仅为我今后的学习和工作打下了坚实的基础，也增强了我在软件工程领域学习发展的信心。

# 9 附录

过程文件如下所示：

1.业务流程图

2.系统数据建模

3.层次结构图

4.顶层和0层数据流图

5.详细设计流程图&PAD图

6.Ticket-Vue

7.Ticket-Springboot

8.Ticket.sql