



综合实训报告

|  |  |
| --- | --- |
| 学院 | 计算机科学与工程学院 |
| 课程 | 面向对象程序设计综合实训 |
| 项目名称 | 图书借阅管理系统的设计与实现 |
| 班级 | 计科22101班 |
| 组长 | 苏一行 2205006224 |
| 成员 | 李明杰 2205006169 |
| 地点 | 新工科 4412 |
| 时间 | 2023-2024学年第二学期 |

目 录

[1. 绪论 1](#_Toc171649733)

[1.1 课题研究的背景及意义 1](#_Toc171649734)

[1.1.1 课题研究的背景 1](#_Toc171649735)

[1.1.2 课题研究的意义 1](#_Toc171649736)

[1.2 课题研究的现状 1](#_Toc171649737)

[1.3 课题研究的内容 1](#_Toc171649738)

[1.4 成员分工 1](#_Toc171649739)

[2. 关键技术 2](#_Toc171649740)

[2.1 Java语言 2](#_Toc171649741)

[2.2 MySQL数据库 2](#_Toc171649742)

[2.3 JavaFX 2](#_Toc171649743)

[3. 需求分析 3](#_Toc171649744)

[3.1 可行性分析 3](#_Toc171649745)

[3.1.1 技术可行性分析 3](#_Toc171649746)

[3.1.2 经济可行性分析 3](#_Toc171649747)

[3.1.3 社会可行性分析 3](#_Toc171649748)

[3.2 需求分析 3](#_Toc171649749)

[3.2.1 系统设计目标 3](#_Toc171649750)

[4. 系统设计 5](#_Toc171649751)

[4.1 总体设计 5](#_Toc171649752)

[4.1.1系统设计目标 5](#_Toc171649753)

[4.1.2 系统功能分析 5](#_Toc171649754)

[4.2详细设计 6](#_Toc171649755)

[4.3数据库设计 8](#_Toc171649756)

[4.3.1概念结构设计 8](#_Toc171649757)

[4.3.2逻辑结构设计 9](#_Toc171649758)

[4.3.3物理结构设计 10](#_Toc171649759)

[5. 系统实现 13](#_Toc171649760)

[5.1 项目结构 13](#_Toc171649761)

[5.2登录的实现 14](#_Toc171649762)

[5.3书籍借阅的实现 20](#_Toc171649763)

[5.4借阅判断的实现 27](#_Toc171649764)

[5.5书籍归还的实现 32](#_Toc171649765)

[5.6 借阅信息可视化的实现 38](#_Toc171649766)

[5.7 AI问询的实现 42](#_Toc171649767)

[5.8个人中心的实现 53](#_Toc171649768)

[5.9管理端的实现 55](#_Toc171649769)

[5.10 Excel数据批量导入的实现 66](#_Toc171649770)

[5.11数据库调用的实现 70](#_Toc171649771)

[5.12 AI调用及Http调用的实现 90](#_Toc171649772)

[6. **系统测试** 108](#_Toc171649773)

[6.1 测试方法 108](#_Toc171649774)

[6.2测试用例设计 108](#_Toc171649775)

[6.3测试结论 110](#_Toc171649776)

[7. 总结与展望 111](#_Toc171649777)

[7.1 总结 111](#_Toc171649778)

[7.2展望 111](#_Toc171649779)

# 1. 绪论

## 1.1 课题研究的背景及意义

1.1.1 课题研究的背景

随着宁夏理工学院新图书馆的建立,现阶段需要一套图书管理体系,同时, 随着图书馆规模的扩大，传统的纸质管理方式已经无法适应现代图书馆的需求,在这种背景下,本套图书管理系统应运而生.

1.1.2 课题研究的意义

图书管理系统的开发还有助于信息资源的共享和利用,可以实现对宁夏理工学院新图书馆资源的高效合理利用。系统中的图书目录和借阅记录等数据可以进行统计和分析，为图书馆和学术机构提供宝贵的参考依据。通过分析读者的阅读喜好和借阅行为，图书馆可以更准确地进行图书采购和管理，满足读者的需求。同时，这些数据也有助于学术研究和图书馆管理者的决策制定。

## 1.2 课题研究的现状

目前，国内外图书管理系统的研究和开发已经比较成熟，有着广泛的应用和发展。

随着信息化建设的不断推进，国内图书管理系统的研究和开发也在不断深入。近年来，国内图书管理系统的开发越来越注重用户体验和功能的完善，例如增加移动端的支持、加强数据分析和图表展示等。同时，国内也有一些新型的图书管理系统正在发展，例如基于云计算和大数据技术的图书管理系统。

总的来说，图书管理系统在国内外的研究和开发已经比较成熟，未来的发展趋势将更加注重用户体验、智能化和拓展应用范围。同时，随着技术的不断升级和发展，图书管理系统也将不断地升级和更新。

## 1.3 课题研究的内容

本研究的核心内容是开发一套包含用户管理、图书分类、图书信息管理、图书借阅管理以及借阅信息数据可视化功能,同时结合现阶段兴起的AIGC技术的智能图书管理系统。具体研究内容包括但不限于：系统架构设计、数据库设计与优化、用户界面开发、图书信息录入与检索机制、数据可视化、人工智能应用。通过这些功能的实现，确保图书管理系统能够满足图书馆日常管理的各项需求，并为读者提供高效便捷的服务体验。

## 1.4 成员分工

**苏一行**：GUI设计，代码开发 **李明杰：**数据库设计、PPT制作

# 2. 关键技术

## 2.1 Java语言

Java是一门面向对象的编程语言，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。

Java具有简单性、面向对象、分布式、健壮性、安全性、平台独立与可移植性、多线程、动态性等特点。 Java可以编写桌面应用程序、Web应用程序、分布式系统和嵌入式系统应用程序等

## 2.2 MySQL数据库

MySQL是一个开源的关系型数据库管理系统，常用于Web应用程序的后端数据存储。它支持多种操作系统和编程语言，具有高性能、可靠性和可扩展性等特点。与其他关系型数据库管理系统相比，MySQL的优点包括高性能，其查询速度非常快，可以处理大量数据。

MySQL提供了多种存储引擎，每种存储引擎都有其特定的用途和性能特点。同时MySQL还支持复制和集群功能，适合不同规模的应用。复制适合中小规模应用，而集群适合大规模应用。索引是MySQL中的一个重要功能，可以加快数据检索操作，但会使数据修改操作变慢。为了在某种程度上弥补这一缺陷，许多SQL命令都有一个DELAY\_KEY\_WRITE选项，可以暂时制止MySQL在每次插入或修改记录后立刻对索引进行刷新。

数据库管理软件采用Data Grip，该软件提供了一个集成式开发、设计和管理环境，提高了开发人员和DBA的工作效率。

## 2.3 JavaFX

比起Swing，JavaFX支持更加现代和富有表现力的界面设计，包括支持CSS样式表和FXML布局文件，使得界面设计更加灵活和美观。同时JavaFX与Java平台更加紧密集成，天然支持跨平台开发，使得应用程序可以无缝运行在不同操作系统上。JavaFX采用了现代化的架构和技术栈，如支持并发性的新型UI渲染引擎，有助于提升性能和响应速度。JavaFX提供了丰富的内置控件库，可以满足大多数应用程序的界面需求，同时也支持自定义控件的开发。

# 3. 需求分析

## 3.1 可行性分析

3.1.1 技术可行性分析

编程语言：Java

主要工具：IntelliJ IEDA 2021、 jdk17、DataGrip 2023

数据库：MySQL

主要技术：JavaFX、JDBC、生成式人工智能（AIGC）、数据可视化操作

在编程语言上，考虑到Java语言具有良好的跨平台型、且对象的编程范式，包括封装、继承和多态，使得代码更易于理解、维护和重用、API简单易学，JavaFX操作方便、功能强大、界面美观。因此整体架构采用了Java语言进行编写。

数据的存储与管理上，考虑到学校图书馆人流量较大、因此需要更高效更便捷的数据存储与管理方式，所以我们采用了数据库进行存储，替代了简单的本地文件存储方式。

在生成式人工智能（AIGC）的实现上，我们调用的是Vivo公司旗下的蓝心大模型的API接口，从而可以实现依据学生需求为学生推荐更为合适的书籍、解答学生的问题。

3.1.2 经济可行性分析

成本预算：

软件成本：IntelliJ IEDA 2021、 DataGrip 2023等软件均采用学生版，因此不收取任何费用。AIGC的实现上，调用Vivo的蓝心大模型API接口，该接口对外免费开源，因此也不需要付费。

硬件成本：该项目的实现的所有功能均为软件项目，不需要额外的硬件支持，因此采用自己的笔记本电脑足以实现项目的需要，成本可忽略不计。

3.1.3 社会可行性分析

该系统率先应用于宁夏理工学院新图书馆,以方便学生的借阅以及图书管理员的管理。同时，在此基础上，我们将不断发现并修改图书管理系统的问题，并加以修改完善，争取向其他学校或图书管理机构进行推广。

## 3.2 需求分析

3.2.1 系统设计目标

经过线下调研，我们发现传统传统图书管理系统的局限于不足，并通过网络问卷与线下走访等形式调查分析了用户对图书管理系统的期望以及管理员对图书管理系统的期望，最终提出如下设计目标。

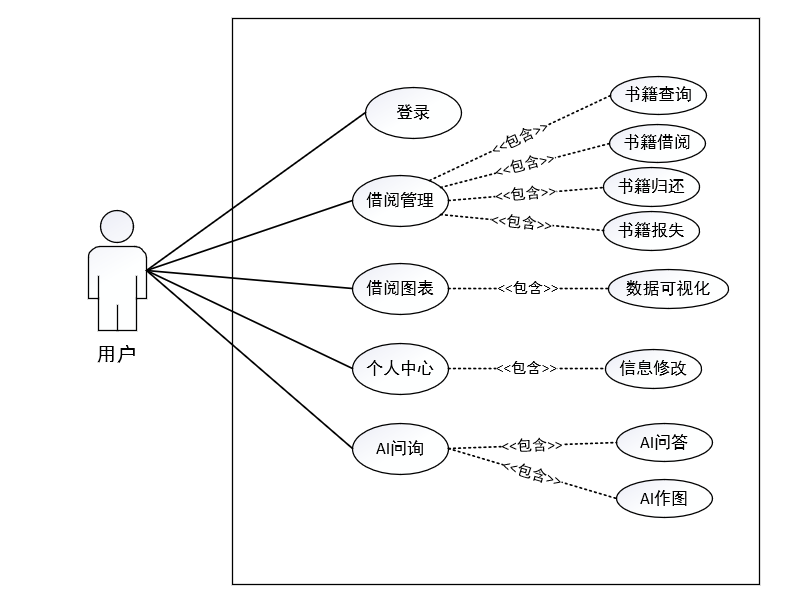


图3.1 用户用例图

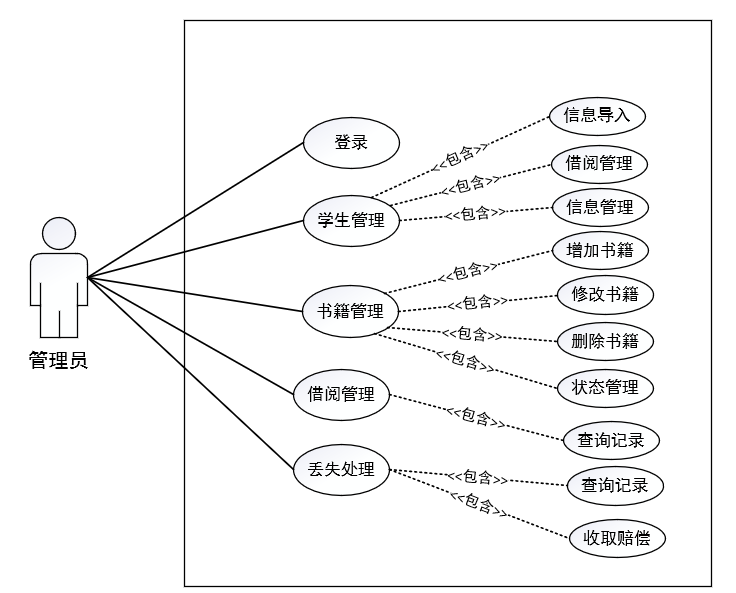


图3.2 管理员用例图

# 4. 系统设计

## 4.1 总体设计

4.1.1系统设计目标

(1) 图书馆管理员能够对图书资源进行管理；

(2) 图书馆管理员能够对读者信息进行管理；

(3) 读者能够根据检索功能，查询并借阅书籍；

(4) 读者可以根据可视化的借阅信息，精确匹配优质书籍；

(5) 读者可以通过AI快速了解书籍概要信息，并获得适合阅读的书籍推荐。

4.1.2 系统功能分析

该系统旨在协助图书馆管理员实现各种日常事务的管理。系统维护数据库，保存图书和读者的资料以及图书流通情况的资料，便于管理员管理图书和读者的有关数据，还可根据需要随时进行数据的查询。利用这些数据，系统可协助管理员进行读者的图书出借、返还、续借等操作，大大减轻了管理员的工作量。

具体功能描述如下：

（1）图书信息维护：主要完成图书馆图书的添加、修改和删除等操作。

（2）读者信息维护：主要完成读者信息的添加、修改和删除等操作。

（3）借书/ 还书处理：记录读者借还书情况并及时反映图书的在库情况及库存量。

（4）丢失损坏管理：收集用户丢失或损坏图书的情况，并获取相应赔偿。

（5）读者借阅记录：让每位读者能及时了解自己的借书情况。

（6）图书检索：读者能够根据不同的信息（如书名、作者、类别等）对图书馆的存书情况进行查找，并进行可视化展示，以便快速的找到自己希望的图书。

（7）书籍推荐：经用户提问与AI智能分析，选择出最适合用户阅读的书籍进行推荐。

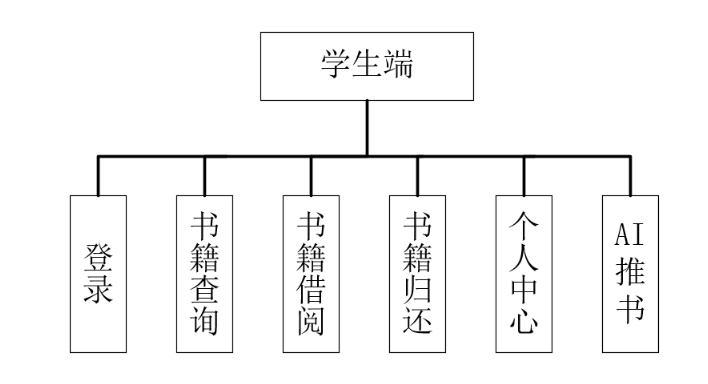


图4.1学生端功能模块图

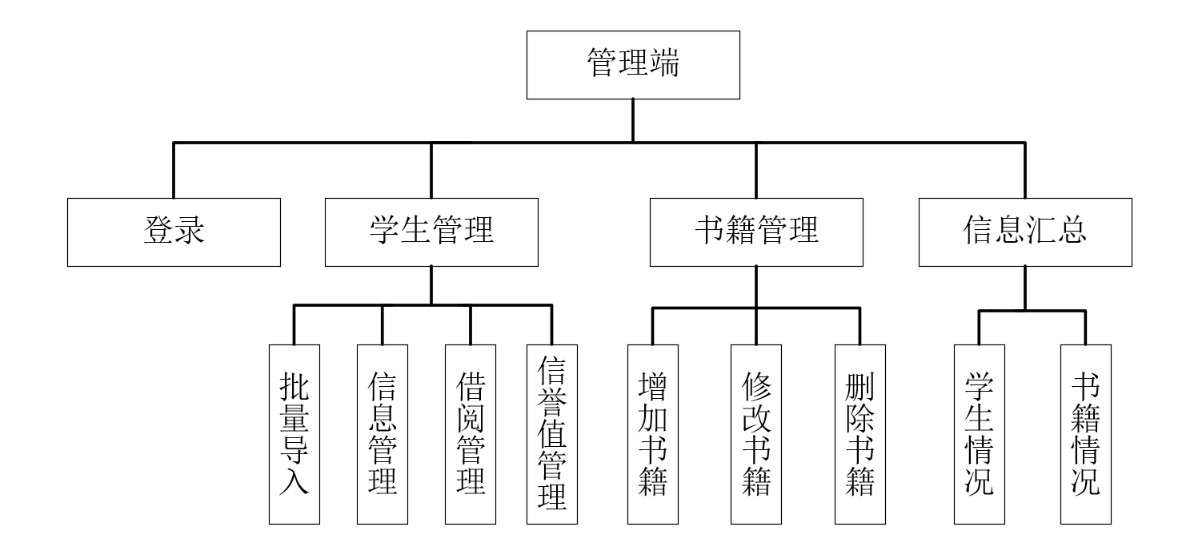


图4.2 管理端功能模块图

## 4.2详细设计

判断用户可以借阅书籍的数量，流程图如下：

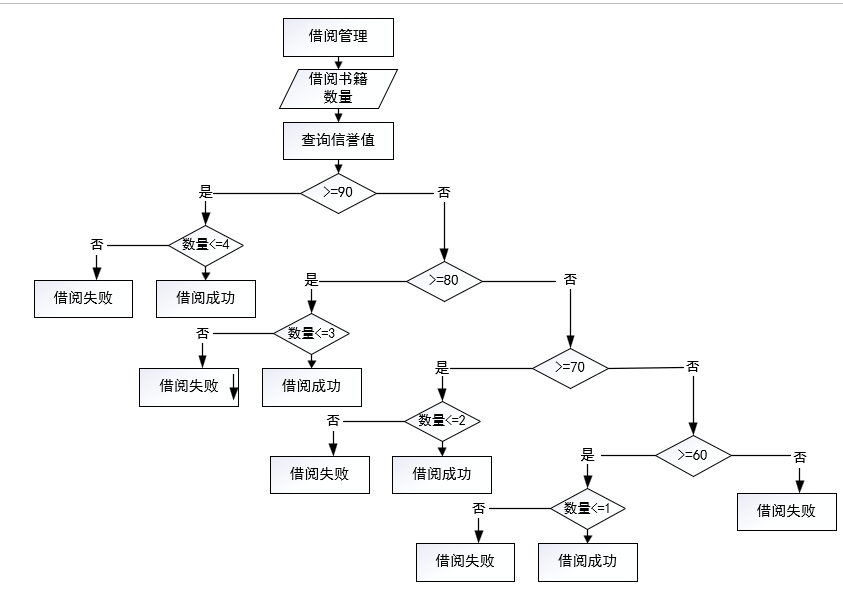


图4.1 书籍借阅流程图

功能函数原型说明示例如下表：

表4.1 主要函数功能说明

| 功能函数 | 说明 |
| --- | --- |
| Arraylist<Student> getAllStudent() | 功能：获取数据库中所有学生的信息。  输入：无  输出：包含所有学生信息（Student）的Arraylist |

表4.1 主要函数功能说明（续）

|  |  |
| --- | --- |
| Arraylist<Book> getAllBook() | 功能：获取数据库中所有的图书信息  输入：无  输出：包含所有书籍信息（book）的Arraylist |
| Arraylist<Book> getBorrowBook(String student\_id) | 功能：获取当前登录学生的借阅信息  输入：当前学生的学号  输出：该学生所借阅的全部书籍 |
| String findStudentPassword(String studentID) | 功能：根据学生ID查找学生密码  输入：学生ID  输出：学生密码 |
| Map<String, Integer> findAllBookNumInBorrow() | 功能:查找每种书籍类别被借阅的次数  输入:无  输出：Map类型变量，其中键为书籍类型，值为被借阅次数 |
| ArrayList<Borrow> findAllBorrowsByStudentID(String studentID) | 功能：通过学生ID查找该学生借阅的所有书籍  输入：学生ID  输出：该学生借阅的所有书籍 |
| long calculateDateDifference(Date date1, Date date2) | 功能：获取两个时间段之间的差值  输入：两个Java.sql.Data类型的时间  输出：两个时间的差值天数 |
| String vivogpt(String input) | 功能：获取AI问答的内容  输入：用户提出的问题  输出：AI回答的结果 |
| String getimageUrlByAI(String input) | 功能：获取所需图片的Url  输入：用户需求描述  输出：AI生成图片的Url地址 |
| Void setImageView(String Url) | 功能：将图片添加到面板中  输入：图片Url  输出：无 |

## 4.3数据库设计

4.3.1概念结构设计

学生表（student）：学号、密码、姓名、性别、所在学院、入学年份、联系电话

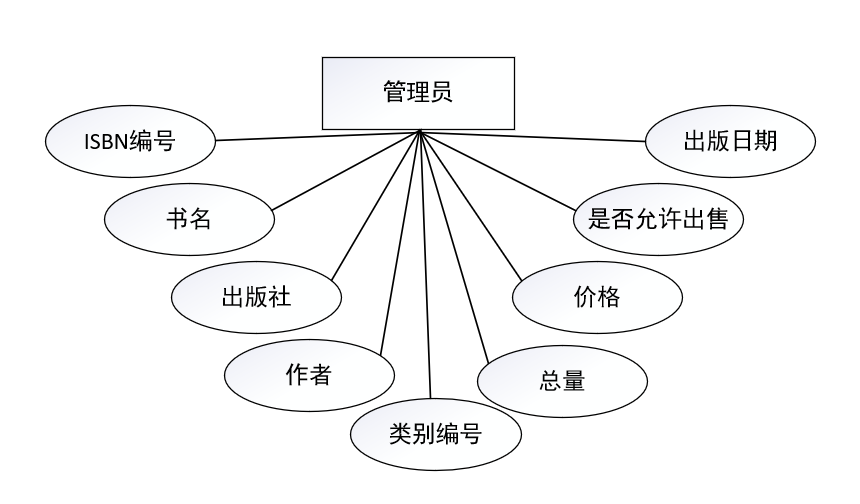


图4.3书籍信息E-R图

书籍（book）：ISBN编号、书名、出版社、出版日期、作者、总量、价格、是否允许出售、类别编号。

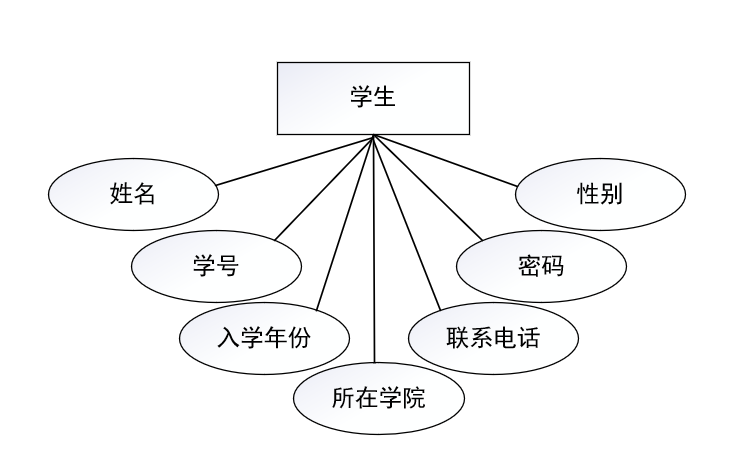


图4.4 学生信息E-R图

管理员（admin）：管理员编号、账号、密码、姓名、联系电话

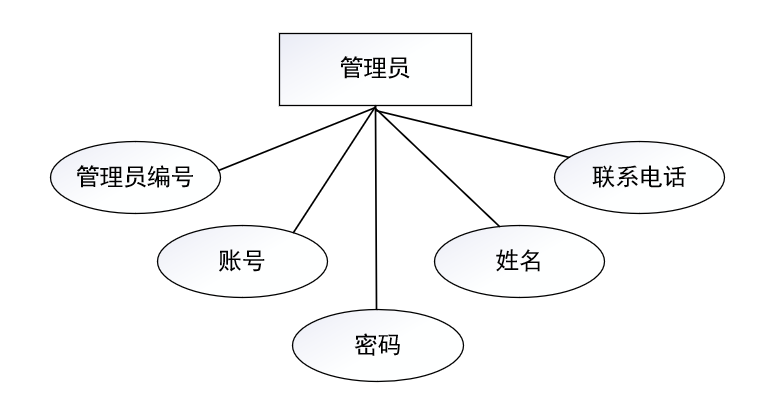


图4.5 管理员信息E-R图

4.3.2逻辑结构设计

学生表（student）：（学号、密码、姓名、性别、所在学院、入学年份、联系电话）

书籍（book）：（ISBN编号、书名、出版社、出版日期、作者、总量、价格、是否允许出售、类别编号）

管理员（admin）：（管理员编号、账号、密码、姓名、联系电话）

类别表：（类别编号、图书类别）

借阅记录：（记录编号、 读者编号、 图书编号（外键）、 借阅时间、应归还时间、实际归还时间、状态）

丢失损坏记录表：（记录编号、学号、ISBN编号、书籍情况、上报时间、赔偿金额）

征信管理：（读者编号、 总借阅次数、延迟归还次数、 实际征信分数、征信等级最大允许借阅数量）



图4.6 书籍借阅系统全局E-R图

4.3.3物理结构设计

表4.2 学生表（student）

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 字段 | 类型 | 是否主键 | 是否为空 | 中文描述 | 备注 |
| student\_id | CHAR(12) | 是 | 否 | 学号 |  |
| password | VARCHAR(12) | 否 | 否 | 密码 |  |
| student\_name | VARCHAR(12) | 否 | 否 | 姓名 |  |
| gender | CHAR(2) | 否 | 否 | 性别 |  |
| faculty | VARCHAR(20) | 否 | 否 | 所在学院 |  |
| year | DATE | 否 | 否 | 入学年份 |  |
| telephone | CHAR(11) | 否 | 否 | 联系电话 | 不重复 |

表4.3 管理员表（admin）：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 字段 | 类型 | 是否主键 | 是否为空 | 中文描述 | 备注 |
| admin\_id | INT | 是 | 否 | 编号 | 自增主键 |
| account | CHAR(12) | 否 | 否 | 账号 | 不重复 |
| password | VARCHAR(12) | 否 | 否 | 密码 |  |
| admin\_name | VARCHAR(12) | 否 | 否 | 姓名 |  |
| telephone | CHAR(11) | 否 | 否 | 联系电话 | 不重复 |

表4.4 书籍表(book)：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 字段 | 类型 | 是否主键 | 是否为可为空 | 中文描述 | 备注 |
| ISBN | CHAR(10) | 是 | 否 | ISBN编号 |  |
| title | VARCHAR(20) | 否 | 否 | 书名 |  |
| author | VARCHAR(12) | 否 | 否 | 作者 |  |
| publisher | VARCHAR(20) | 否 | 否 | 出版社 |  |
| publication\_date | DATE | 否 | 否 | 出版日期 |  |
| total | INT | 否 | 否 | 总量 |  |
| price | DECIMAL(6, 2) | 否 | 否 | 价格 |  |
| type\_id | INT | 否 | 否 | 类型 | 外键 |

表4.5 类别表(type):

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 字段 | 类型 | 是否主键 | 是否为空 | 中文描述 | 备注 |
| type\_id | INT | 是 | 否 | 类别ID | 自增 |
| type | VARCHAR(10) | 否 | 否 | 类别名 |  |

表4.6 丢失/损坏情况记录表（lose）：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 字段 | 类型 | 是否主键 | 是否为空 | 中文描述 | 备注 |
| lose\_id | INT | 是 | 否 | 记录编号 | 自增 |
| student\_id | CHAR(12) | 否 | 否 | 学号 | 外键 |
| ISBN | CHAR(10) | 否 | 否 | ISBN编号 | 外键 |
| status | VARCHAR(5) | 否 | 否 | 状态 | 丢失/损坏 |
| report\_time | DATE | 否 | 否 | 上报时间 | 默认填充当前时间 |
| is\_compensation | BOOLEN | 否 | 否 | 是否赔偿 |  |
| remark | TEXT | 否 | 是 | 备注 |  |

定义触发器事件，在lose表中插入数据时，若时间上报时间为空，则默认填充当前时间。

表4.7 借阅记录表（borrow）：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 字段 | 类型 | 是否主键 | 是否为空 | 中文描述 | 备注 |
| borrow\_id | INT | 是 | 否 | 记录编号 | 自增 |
| student\_id | CHAR(12) | 否 | 否 | 学号 | 外键 |
| ISBN | CHAR(10) | 否 | 否 | ISBN编号 | 外键 |
| borrow\_time | DATE | 否 | 否 | 借阅时间 |  |
| return\_time | DATE | 否 | 否 | 应归还时间 |  |
| actual\_return\_time | DATE | 否 | 是 | 实际归还时间 |  |

定义触发器事件，删除学生或书籍信息时，检查是否存在未归还书籍，当借阅记录表中存在未归还书籍时，该书籍信息及借阅该书籍的学生信息不得删除。

表4.8 学生信誉表（credit）：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 字段 | 类型 | 是否主键 | 是否为空 | 中文描述 | 备注 |
| student\_id | CHAR(12) | 是 | 否 | 学号 |  |
| borrow\_times | INT | 否 | 否 | 总借阅次数 | 默认为0 |
| late\_return\_times | INT | 否 | 否 | 延迟归还次数 | 默认为0 |
| credit\_value | TINYINT | 否 | 否 | 信誉积分 | 默认为100 |
| credit\_rating | CHAR(5) | 否 | 否 | 信誉等级 | 默认为优秀 |
| borrow\_limit | TINYINT | 否 | 否 | 借阅上限 | 默认为4 |

学生信誉表字段更新规则：

该表与学生表、以及借阅记录表同步更新。每一条学生表中的学生数据对应该表中的一条数据。

其中:

borrow\_times = count(student\_id) from borrow where borrow. student\_id = credit. student\_id（借阅记录表中该学生借书总次数）

late\_return\_times = count(return\_time > actual\_return\_time) from borrow where borrow. student\_id = credit. student\_id （借阅记录表中该学生延迟归还次数）

credit\_value = 100 - late\_return\_times \* 5 + borrow\_times （取值范围为0-100，当数值大于100时则取值为100）

credit\_rating = 优秀（credit\_value>=90 ）、良好（90 > credit\_value >= 80）、一般（80 > credit\_value >= 70）、及格（70 > credit\_value >= 60）、不及格（60 > credit\_value）

borrow\_limit （当信誉等级为优秀时，借阅上限为4、当信誉等级为良好时，借阅上限为3、当信誉等级为一般时，借阅上限为2、当信誉等级为及格时，借阅上限为1、当信誉等级不及格时或存在逾期且未归还书籍时（count(return\_time <当前时间 and actual\_return\_time = null) ），借阅上限为0）

# 5. 系统实现

## 5.1 项目结构

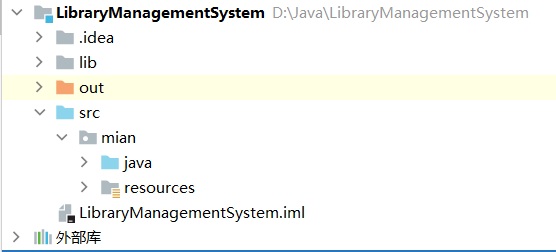


图5.1总体项目结构

项目大文件夹下包括lib与src两个文件，其中，lib存储项目所用到的依赖，src存储项目的实现方法以及资源文件。其中src.mian.java中包含具体的代码，main,src.resources中包含项目所有的资源文件。

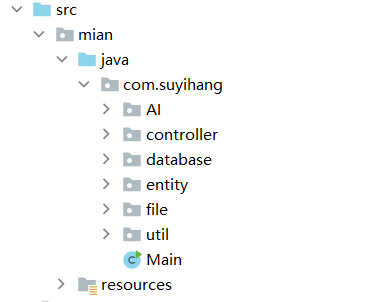


图5.2 java文件夹项目结构

在Java文件夹下，各个文件夹的具体功能分别如下：

AI：连接AI，实现对蓝心大模型接口的调用。

controller：对前端代码进行控住，并将初始化好的数据添加到前端

database： 操纵数据库，实现增删改查

entity： 存储项目所需要的所有实体类

file：通过easy Excel的调用，实现对Excel文件的读写操作

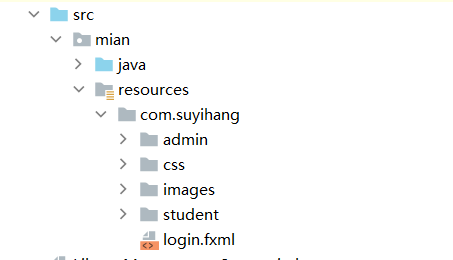
util： 存储该项目所需要的工具类，如字符串处理，时间处理等操作 

图5.3resources文件夹项目结构

在resource文件夹下，各个模块功能如下所示：

admin：用来存储管理端fxml布局文件

css：存储JavaFX所需的css美化效果

images：存储项目所需的所有图片

student：存储用户端的fxml布局文件

## 5.2登录的实现

功能描述：登录界面的实现，可以判断用户名是否存在，密码是否错误，并显示在下方。

界面截图：



图5.4 登录窗口

代码实现：

**package** com.suyihang.controller;  
  
**import** com.suyihang.Main;  
**import** com.suyihang.database.DataSelect;  
**import** com.suyihang.database.DataSelectLogin;  
**import** com.suyihang.util.AdminUtil;  
**import** com.suyihang.util.StringUtil;  
**import** com.suyihang.util.StudentUtil;  
**import** javafx.event.ActionEvent;  
**import** javafx.fxml.FXML;  
**import** javafx.scene.control.\*;  
**import** javafx.scene.layout.StackPane;  
**import** javafx.scene.layout.VBox;  
  
**public class** LoginController {  
  
 @FXML  
 **public** Label **warnLabel**;  
  
 @FXML  
 **private** StackPane **MainPane**;  
  
 @FXML  
 **private** ChoiceBox<String> **choiceBox**;  
  
 @FXML  
 **private** Button **loginButton**;  
  
 @FXML  
 **private** VBox **loginView**;  
  
 @FXML  
 **private** PasswordField **passwordField**;  
  
 @FXML  
 **private** TextField **usernameField**;  
  
 @FXML  
 **void** handleLogin(ActionEvent event) {  
 String value = **choiceBox**.getValue();  
 **warnLabel**.setVisible(**false**);  
 **if** (value == **null**){  
 **warnLabel**.setText(**"请先选择登录信息"**);  
 **warnLabel**.setVisible(**true**);  
 **return**;  
 }  
  
 **if** (value.equals(**"用户登录"**)){  
 skip\_Student();  
 **return**;  
 }  
 **if** (value.equals(**"管理员登录"**)){  
 skip\_Admin();  
 **return**;  
 }  
 System.***out***.println(value);  
 }  
  
 **private void** skip\_Student(){  
 String id = **usernameField**.getText();  
  
 String password = **passwordField**.getText();  
  
 String password2 = DataSelectLogin.*findStudentPassword*(id);  
  
 **if** (StringUtil.*empty*(id)) {  
 **warnLabel**.setVisible(**true**);  
 **return**;  
 }  
  
 **if** (StringUtil.*empty*(password)) {  
 **warnLabel**.setText(**"密码不得为空"**);  
 **warnLabel**.setVisible(**true**);  
 **return**;  
 }  
  
 **if** (StringUtil.*empty*(password2)){  
 System.***out***.println(password2);  
 **warnLabel**.setText(**"用户名不存在"**);  
 **warnLabel**.setVisible(**true**);  
 **return**;  
 }  
  
 **if** (!password.equals(password2)) {  
 **warnLabel**.setText(**"密码错误"**);  
 **warnLabel**.setVisible(**true**);  
 **return**;  
 }  
  
 **try** {  
 StudentUtil.*current\_student* = DataSelect.*findStudentByID*(id);  
 Main.*switchScene*(**"/com/suyihang/student/lendBook.fxml"**);  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 }  
 }  
  
 **private void** skip\_Admin(){  
 String id = **usernameField**.getText();  
  
 String password = **passwordField**.getText();  
  
 String password2 = DataSelectLogin.*findAdminPassword*(id);  
  
 **if** (StringUtil.*empty*(id)) {  
 **warnLabel**.setVisible(**true**);  
 **return**;  
 }  
  
 **if** (StringUtil.*empty*(password)) {  
 **warnLabel**.setText(**"密码不得为空"**);  
 **warnLabel**.setVisible(**true**);  
 **return**;  
 }  
  
 **if** (StringUtil.*empty*(password2)){  
 **warnLabel**.setText(**"用户名不存在"**);  
 **warnLabel**.setVisible(**true**);  
 **return**;  
 }  
  
 **if** (!password.equals(password2)) {  
 **warnLabel**.setText(**"密码错误"**);  
 **warnLabel**.setVisible(**true**);  
 **return**;  
 }  
  
 **try** {  
 AdminUtil.*currentAdmin* = DataSelect.*findAdminByAccount*(id);  
 Main.*switchScene*(**"/com/suyihang/admin/adminMenu.fxml"**);  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 }  
 }  
  
 @FXML  
 **private void** initialize() {  
 *// 设置默认选择* **choiceBox**.setValue(**"用户登录"**);  
 }  
}

## 5.3书籍借阅的实现

功能描述：书籍借阅功能实现，单击表格后，表格数据展示在右侧面板。同时可以实现对书籍的模糊查询。单击菜单组件，可以实现跳转页面

界面截图：



图5.5 书籍借阅



图5.6 书籍筛选

代码实现:

@FXML  
 **void** buyBooks(ActionEvent event) {  
 DialogUtil.*showAlert*(**"提示"**, **"该功能暂未开放。"**);  
 }  
  
 @FXML  
 **void** lendBooks(ActionEvent event) {  
  
 **if** (BookUtil.*currentSelectBook* == **null**) {  
 DialogUtil.*showAlert*(**"提示"**, **"请先选择书籍。"**);  
 } **else** {  
 Main.*addScene*(**"/com/suyihang/student/LendBookItem.fxml"**);  
 Book selectedBook = **booksTableView**.getSelectionModel().getSelectedItem();  
 selectedBook = DataSelectBook.*selectBookByISBN*(selectedBook.getISBN());  
 setRightLabel(selectedBook);  
 findBook();  
 }  
  
 }  
  
 @FXML  
 **void** findBooks(ActionEvent event) {  
 findBook();  
 }  
  
 **private void** findBook() {  
  
 String title = **titleField**.getText();  
 String author = **authorField**.getText();  
 String typeName = **typeBox**.getValue();  
  
 **int** typeID;  
  
 **if** (StringUtil.*empty*(typeName)) {  
 typeID = 0;  
 } **else** {  
 typeID = **typeCom**.get(typeName);  
 }  
  
 *// 按条件查找* ArrayList<Book> bookByCriteria = DataSelectBook.*findBookByCriteria*(title, author, typeID);  
  
 addTableView(bookByCriteria);  
 }  
  
   
 @FXML  
 **private void** initialize() {  
 StudentUtil.*current\_student* = DataSelect.*findStudentByID*(StudentUtil.*current\_student*.getStudentId());  
  
 **typeCom** = DataSelectBook.*findAllType*();  
  
 addTableView(DataSelectBook.*selectAllBooks*());  
  
 addComboBox();  
 }  
  
 **private** ObservableList<Book> convertToObservableList(ArrayList<Book> bookList) {  
 **return** FXCollections.*observableArrayList*(bookList);  
 }  
  
 @FXML  
 **private void** handleRowSelect(MouseEvent event) {  
 **if** (event.getClickCount() == 1) { *// Double-click to select* Book selectedBook = **booksTableView**.getSelectionModel().getSelectedItem();  
 **if** (selectedBook != **null**) {  
 System.***out***.println(**"Selected book: "** + selectedBook);  
 setRightLabel(selectedBook);  
 *// You can now use selectedBook to perform further actions* }  
 }  
 }  
  
 **private void** setRightLabel(Book book) {  
 **titleLabel**.setText(book.getTitle());  
 **authorLabel**.setText(book.getAuthor());  
 **publicLabel**.setText(book.getPublisher());  
 **publicDateLabel**.setText(book.getPublicationDate().toString());  
 **typeLabel**.setText(String.*valueOf*(book.getType()));  
 **priceLabel**.setText(String.*valueOf*(book.getPrice()));  
 **repetoryLabel**.setText(String.*valueOf*(book.getRepertory()));  
  
 BookUtil.*currentSelectBook* = book;  
 }  
  
 **private void** addTableView(ArrayList<Book> bookList) {  
 **booksTableView**.getItems().clear();  
  
 **nameLabel**.setText(StudentUtil.*current\_student*.getStudentName());  
 *// 初始化表格的列* **bookTitleColumn**.setCellValueFactory(**new** PropertyValueFactory<>(**"title"**));  
 **authorColumn**.setCellValueFactory(**new** PropertyValueFactory<>(**"author"**));  
 **publicColumn**.setCellValueFactory(**new** PropertyValueFactory<>(**"type"**));  
 **repetoryColumn**.setCellValueFactory(**new** PropertyValueFactory<>(**"repertory"**));  
  
 *// 将 ArrayList 转为 ObservableList 并添加到表格中* **booksTableView**.setItems(convertToObservableList(bookList));  
 }  
  
 **private void** addComboBox() {  
  
 **typeBox**.getItems().add(**""**);  
  
 **typeBox**.setValue(**""**);  
  
 **for** (Map.Entry<String, Integer> entry : **typeCom**.entrySet()) {  
 **typeBox**.getItems().add(entry.getKey());  
 }  
 }  
  
}

页面跳转代码实现：

@FXML  
 **void** skipAIGC(MouseEvent event) {  
 AIGCUtil.*currentAI* = **new** AIGCrecord();  
 **try** {  
 Main.*switchScene*(**"/com/suyihang/student/AIGC.fxml"**);  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 }  
 }  
  
 @FXML  
 **void** skipBookFind(MouseEvent event) {  
 **try** {  
 Main.*switchScene*(**"/com/suyihang/student/chart.fxml"**);  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 }  
 }  
  
 @FXML  
 **void** skipBookLend(MouseEvent event) {  
 **try** {  
 Main.*switchScene*(**"/com/suyihang/student/lendBook.fxml"**);  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 }  
 }  
  
 @FXML  
 **void** skipBookReturn(MouseEvent event) {  
 **try** {  
 Main.*switchScene*(**"/com/suyihang/student/returnBook.fxml"**);  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 }  
 }  
  
 @FXML  
 **void** skipPersonalCenter(MouseEvent event) {  
 **try** {  
 Main.*switchScene*(**"/com/suyihang/student/personPanel.fxml"**);  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 }  
 }

## 5.4借阅判断的实现

功能描述：借阅状态判断及借阅功能的具体实现

界面截图：



图5.7 判断能否借阅

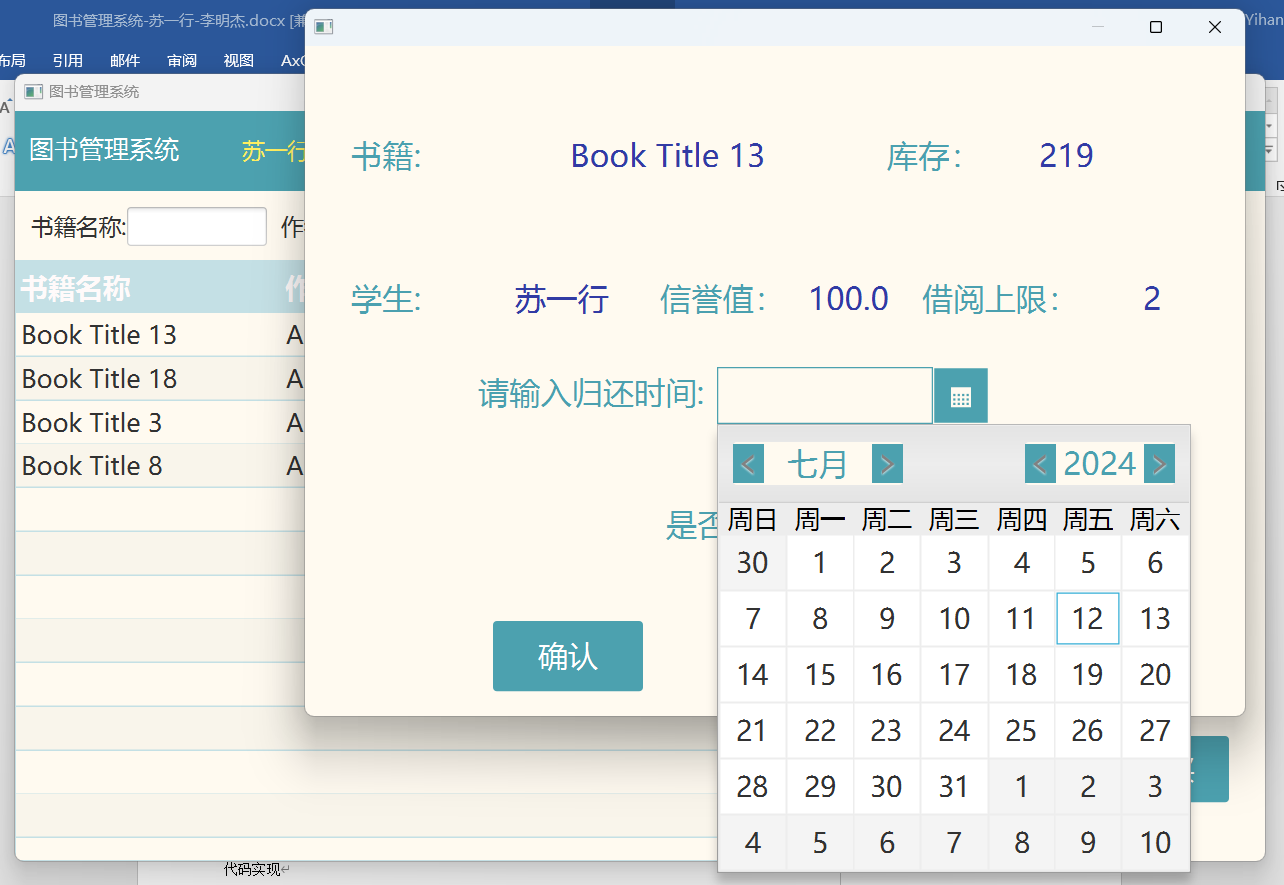


图5.8 选择还书时间

代码实现：

**public class** LendBookItemController {  
  
 @FXML  
 **private** Label **borrowLimitLabel**;  
  
 @FXML  
 **private** Button **noBut**;  
  
 @FXML  
 **private** Label **repertoryLabel**;  
  
 @FXML  
 **private** DatePicker **returnDatePicker**;  
  
 @FXML  
 **private** Label **studentCreditLabel**;  
  
 @FXML  
 **private** Label **studentNameLabel**;  
  
 @FXML  
 **private** Label **titleLabel**;  
  
 @FXML  
 **private** Button **yesBut**;  
  
 @FXML  
 **void** noLent(ActionEvent event) {  
 Stage stage = (Stage) ((Node) event.getSource()).getScene().getWindow();  
 stage.close();  
 }  
  
 @FXML  
 **void** yesLent(ActionEvent event) {  
 **int** borrowLimitNum = **student**.getBorrowLimit() - **student**.getBorrowedBookNum();  
 **if** (borrowLimitNum < 1) {  
 DialogUtil.*showAlert*(**"提示"**, **"借阅失败,您当前状态不可借书,请检查是否存在未及时归还的书籍"**);  
 **return**;  
 }  
  
 **if** (**book**.getRepertory() <= 0) {  
 DialogUtil.*showAlert*(**"提示"**, **"借阅失败,书籍库存不足"**);  
 **return**;  
  
 }  
  
 LocalDate returnDate = **returnDatePicker**.getValue();  
 **if** (returnDate == **null**) {  
 DialogUtil.*showAlert*(**"提示"**, **"请填写归还日期"**);  
 **return**;  
 }  
 Date date = Date.*valueOf*(returnDate);  
 Date currentDate = DateUtil.*getCurrentTime*();  
 System.***out***.println(date);  
 **if** (DateUtil.*calculateDateDifference*(currentDate, date) > 90){  
 DialogUtil.*showAlert*(**"提示"**, **"最长借阅时间不得超过90天"**);  
 **return**;  
 }  
 **if** (DateUtil.*timeCompare*(currentDate, date) == 1) {  
 DialogUtil.*showAlert*(**"提示"**, **"归还时间不得早于当前时间"**);  
 **return**;  
 }  
 DialogUtil.*showAlert*(**"提示"**, **"借阅成功,请在规定时间内归还"**);  
  
 insertData(currentDate, date);  
  
 }  
  
 **private void** insertData(Date currentDate, Date returnDate) {  
 DataInsert.*insertBorrow*(**student**.getStudentId(), **book**.getISBN(), currentDate, returnDate, **null**);  
 StudentUtil.*current\_student* = DataSelect.*findStudentByID*(**student**.getStudentId());  
 BookUtil.*currentSelectBook* = DataSelectBook.*selectBookByISBN*(**book**.getISBN());  
 initLabel();  
 }  
  
  
 @FXML  
 **private void** initialize() {  
 initLabel();  
 }  
  
 Student **student** = StudentUtil.*current\_student*;  
 Book **book** = BookUtil.*currentSelectBook*;  
  
 **private void** initLabel() {  
  
 **student** = StudentUtil.*current\_student*;  
 **book** = BookUtil.*currentSelectBook*;  
  
 **titleLabel**.setText(**book**.getTitle());  
 **repertoryLabel**.setText(String.*valueOf*(**book**.getRepertory()));  
  
 **studentNameLabel**.setText(**student**.getStudentName());  
 **studentCreditLabel**.setText(String.*valueOf*(**student**.getCreditValue()));  
 **int** borrowLimitNum = **student**.getBorrowLimit() - **student**.getBorrowedBookNum();  
 **borrowLimitLabel**.setText(String.*valueOf*(borrowLimitNum));  
 }  
  
}

## 5.5书籍归还的实现

功能描述：书籍归还功能实现，点击表格，书籍借阅信息显示在右侧边框，同时可以选择还书，延长借阅时间，以及申报书籍损坏或丢失。

界面截图：



图5.9 还书界面



图5.10 申报遗失界面

代码实现:

@FXML  
**void** applyLoseButAction(ActionEvent event) {  
 **returnPanel**.setVisible(**false**);  
 **losePanel**.setVisible(**true**);  
}  
  
@FXML  
**void** applyReturnButAction(ActionEvent event) {  
 **returnPanel**.setVisible(**true**);  
 **losePanel**.setVisible(**false**);  
}  
  
@FXML  
**void** delayButAction(ActionEvent event) {  
 **if** (**selectedBorrow** == **null**) {  
 DialogUtil.*showAlert*(**"提示"**, **"请先选择书籍"**);  
 **return**;  
 }  
 Date returnTime = Date.*valueOf*(**bookReturnDate**.getValue());  
 **if** (DataUpdate.*updateExpectedReturnTime*(**selectedBorrow**.getBorrowId(), returnTime)) {  
 **long** day = DateUtil.*calculateDateDifference*(**selectedBorrow**.getReturnTime(), returnTime);  
 DialogUtil.*showAlert*(**"提示"**, **"您已成功申请延时"** + day + **"天"**);  
 } **else** {  
 DialogUtil.*showAlert*(**"提示"**, **"申请失败,该书籍已归还"**);  
 }  
 addTableView(DataSelectBook.*findAllBorrowsByStudentID*(StudentUtil.*current\_student*.getStudentId()));  
}  
  
@FXML  
**void** returnButAction(ActionEvent event) {  
 **if** (**selectedBorrow** == **null**) {  
 DialogUtil.*showAlert*(**"提示"**, **"请先选择书籍"**);  
 **return**;  
 }  
 Date returnTime;  
 **if** (**bookReturnDate**.getValue() == **null**) {  
 returnTime = DateUtil.*getCurrentTime*();  
 } **else** {  
 returnTime = Date.*valueOf*(**bookReturnDate**.getValue());  
 }  
 **if** (DateUtil.*timeCompare*(returnTime, DateUtil.*getCurrentTime*()) != 0) {  
 DialogUtil.*showAlert*(**"提示"**, **"归还时间只能选择当日"**);  
 } **else** {  
 **if** (DataUpdate.*updateBorrow*(**selectedBorrow**.getBorrowId(), returnTime)) {  
 DialogUtil.*showAlert*(**"提示"**, **"归还成功"**);  
 } **else** {  
 DialogUtil.*showAlert*(**"提示"**, **"操作失败,该书籍已归还"**);  
 }  
 }  
 addTableView(DataSelectBook.*findAllBorrowsByStudentID*(StudentUtil.*current\_student*.getStudentId()));  
}  
  
  
  
@FXML  
**void** submitButAction(ActionEvent event) {  
 **if** (**selectedBorrow** == **null**) {  
 DialogUtil.*showAlert*(**"提示"**, **"请先选择书籍"**);  
 **return**;  
 }  
 String student\_id = StudentUtil.*current\_student*.getStudentId();  
 String ISBN = **selectedBorrow**.getISBN();  
 String status = **bookStatus**.getText();  
 Date reportTimes = Date.*valueOf*(**reportTime**.getValue());  
 **int** isCompensation = **yesRadioBut**.isSelected() ? 1:0;  
 String note = **bookNotes**.getText();  
 **if** (DataInsert.*insertLose*(student\_id, ISBN, status, reportTimes, isCompensation, note)) {  
 **if** (DataUpdate.*updateBorrow*(**selectedBorrow**.getBorrowId(), reportTimes)) {  
 DialogUtil.*showAlert*(**"提示"**, **"提交成功"**);  
 } **else** {  
 DialogUtil.*showAlert*(**"提示"**, **"提交失败,书籍已归还"**);  
 }  
 } **else** {  
 DialogUtil.*showAlert*(**"提示"**, **"提交失败"**);  
 }  
 addTableView(DataSelectBook.*findAllBorrowsByStudentID*(StudentUtil.*current\_student*.getStudentId()));  
}  
  
**private void** addTableView(ArrayList<Borrow> borrowArrayList) {  
 **tableView**.getItems().clear();  
  
 **nameLabel**.setText(StudentUtil.*current\_student*.getStudentName());  
 *// 初始化表格的列* **bookName**.setCellValueFactory(**new** PropertyValueFactory<>(**"title"**));  
 **author**.setCellValueFactory(**new** PropertyValueFactory<>(**"author"**));  
 **type**.setCellValueFactory(**new** PropertyValueFactory<>(**"type"**));  
 **lendData**.setCellValueFactory(**new** PropertyValueFactory<>(**"borrowTime"**));  
 **remainData**.setCellValueFactory(**new** PropertyValueFactory<>(**"days"**));  
  
 *// 将 ArrayList 转为 ObservableList 并添加到表格中* **tableView**.setItems(convertToObservableList(borrowArrayList));  
  
 *// 设置默认排序* **tableView**.getSortOrder().add(**remainData**);  
 **remainData**.setSortType(TableColumn.SortType.***DESCENDING***);  
 **tableView**.sort();  
}  
  
**private** ObservableList<Borrow> convertToObservableList(ArrayList<Borrow> borrowArrayList) {  
 **return** FXCollections.*observableArrayList*(borrowArrayList);  
}  
  
@FXML  
**private void** initialize(){  
 addTableView(DataSelectBook.*findAllBorrowsByStudentID*(StudentUtil.*current\_student*.getStudentId()));  
 addAction();  
}  
Borrow **selectedBorrow**;  
**private void** addAction(){  
 *// 添加单击事件处理* **tableView**.setRowFactory(tv -> {  
 TableRow<Borrow> row = **new** TableRow<>();  
 row.setOnMouseClicked(event -> {  
 **if** (event.getButton() == MouseButton.***PRIMARY*** && event.getClickCount() == 1 && !row.isEmpty()) {  
 **selectedBorrow** = **tableView**.getSelectionModel().getSelectedItem();  
 handleRowDoubleClick(**selectedBorrow**);  
 }  
 });  
 **return** row;  
 });  
}  
  
**private void** handleRowDoubleClick(Borrow selectedBorrow) {  
 *// 设置还书* **bookTitleTextField**.setText(selectedBorrow.getTitle());  
 **bookLentDateTextField**.setText(String.*valueOf*(selectedBorrow.getBorrowTime()));  
 **bookRemainDateTextField**.setText(selectedBorrow.getDays());  
  
 *// 设置丢失* **bookTitleTextField1**.setText(selectedBorrow.getTitle());  
}

## 5.6 借阅信息可视化的实现

功能描述：可视化图表的实现，可以选择柱状图或饼状图同时可以筛选填充数据

界面截图：



图5.11柱状图1

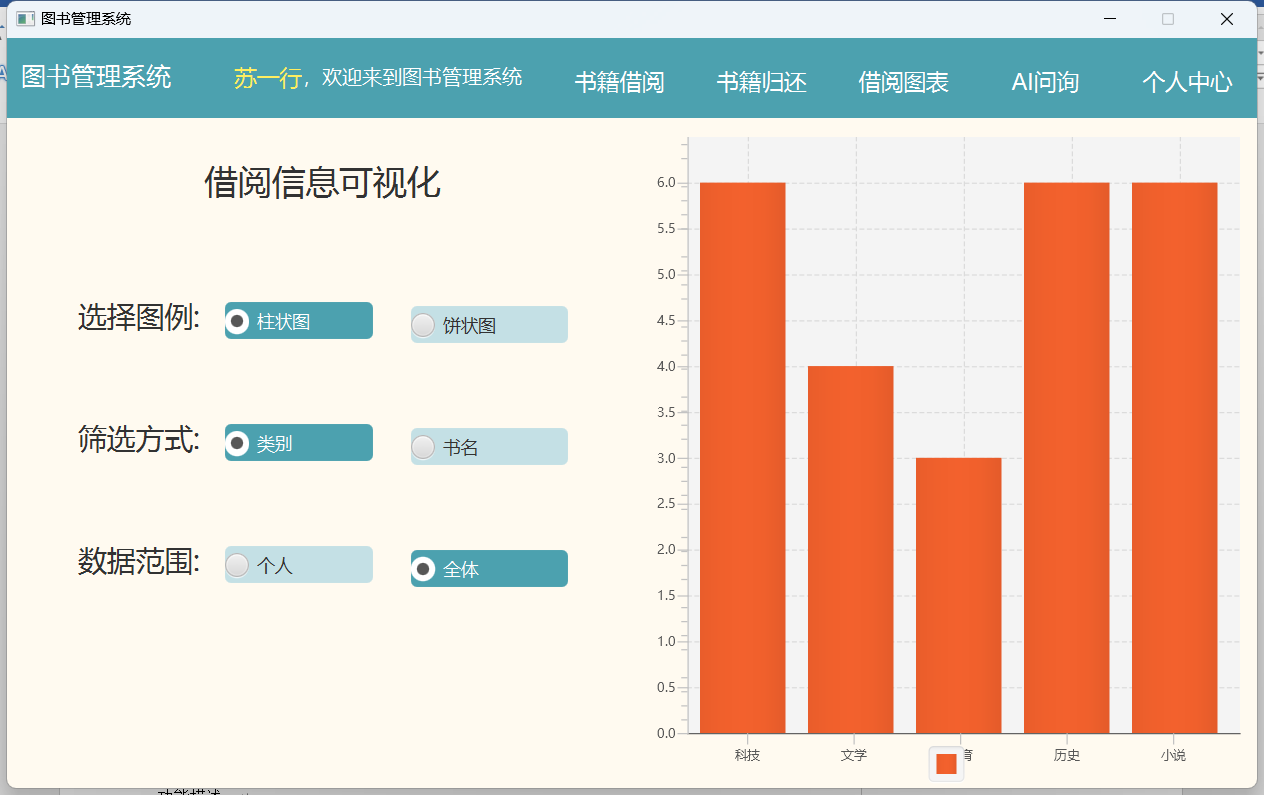


图5.12 柱状图2

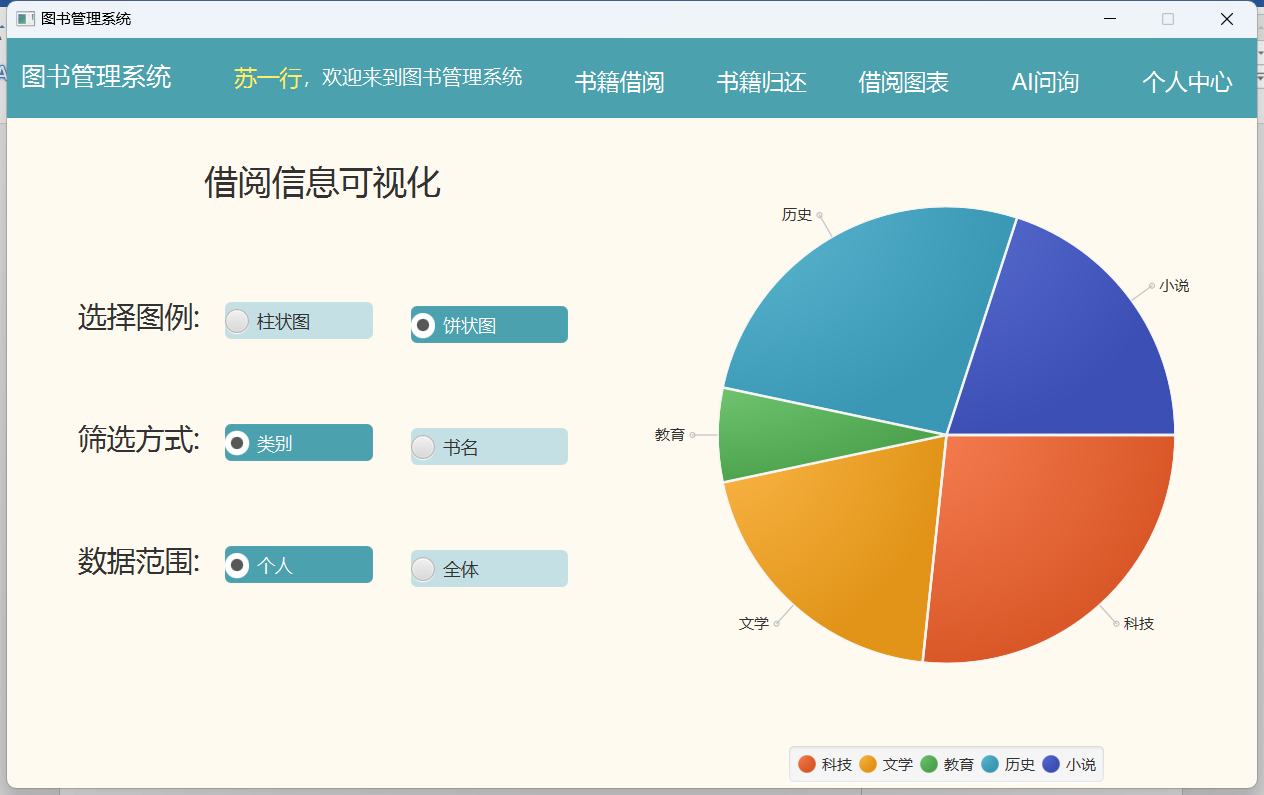


图5.13 饼状图1



图5.14 饼状图2

代码实现：

@FXML  
**void** barChartAction(ActionEvent event) {  
 **barChart**.setVisible(**true**);  
 **pieChart**.setVisible(**false**);  
}  
  
@FXML  
**void** pieChartAction(ActionEvent event) {  
 **barChart**.setVisible(**false**);  
 **pieChart**.setVisible(**true**);  
}  
  
@FXML  
**void** allAction(ActionEvent event) {  
 **if** (**typeRadio**.isSelected()) {  
 setChartData(DataSelectBook.*findAllTypeinBorrowTableNum*());  
 } **else** {  
 setChartData(DataSelectBook.*findAllBookNumInBorrow*());  
 }  
}  
  
@FXML  
**void** personalAction(ActionEvent event) {  
 **if** (**typeRadio**.isSelected()) {  
 setChartData(DataSelectBook.*findAllTypeinBorrowTableNumByStudentID*(StudentUtil.*current\_student*.getStudentId()));  
 } **else** {  
 setChartData(DataSelectBook.*findAllBookNumInBorrowByStudentID*(StudentUtil.*current\_student*.getStudentId()));  
 }  
}  
  
@FXML  
**void** titleAction(ActionEvent event) {  
 **if** (**personalRadio**.isSelected()) {  
 setChartData(DataSelectBook.*findAllBookNumInBorrowByStudentID*(StudentUtil.*current\_student*.getStudentId()));  
 } **else** {  
 setChartData(DataSelectBook.*findAllBookNumInBorrow*());  
  
 }  
}  
  
@FXML  
**void** typeAction(ActionEvent event) {  
 **if** (**personalRadio**.isSelected()) {  
 setChartData(DataSelectBook.*findAllTypeinBorrowTableNumByStudentID*(StudentUtil.*current\_student*.getStudentId()));  
 } **else** {  
 setChartData(DataSelectBook.*findAllTypeinBorrowTableNum*());  
 }  
}  
  
**public void** setChartData(Map<String, Integer> data) {  
 *// 清空现有的数据* **pieChart**.getData().clear();  
 **barChart**.getData().clear();  
  
 *// 添加数据到饼状图* **for** (Map.Entry<String, Integer> entry : data.entrySet()) {  
 PieChart.Data pieData = **new** PieChart.Data(entry.getKey(), entry.getValue());  
 **pieChart**.getData().add(pieData);  
 }  
  
 *// 添加数据到柱状图* XYChart.Series<String, Number> series = **new** XYChart.Series<>();  
 **for** (Map.Entry<String, Integer> entry : data.entrySet()) {  
 XYChart.Data<String, Number> barData = **new** XYChart.Data<>(entry.getKey(), entry.getValue());  
 series.getData().add(barData);  
 }  
 **barChart**.getData().add(series);  
}  
  
@FXML  
**private void** initialize(){  
  
 **nameLabel**.setText(StudentUtil.*current\_student*.getStudentName());  
  
 setChartData(DataSelectBook.*findAllTypeinBorrowTableNumByStudentID*(StudentUtil.*current\_student*.getStudentId()));  
}

## 5.7 AI问询的实现

功能描述：AI问答的实现，用户提出书籍相关的问题，AI可以进行回答，同时可以根据描述绘制相应图片，调用过程采取多线程，不会影响主线程的运行。下载的图片可以保存到本地。

界面截图：



图5.15 AI问答问题一测试



图5.16 问题二测试



图5.17 AI生成图片

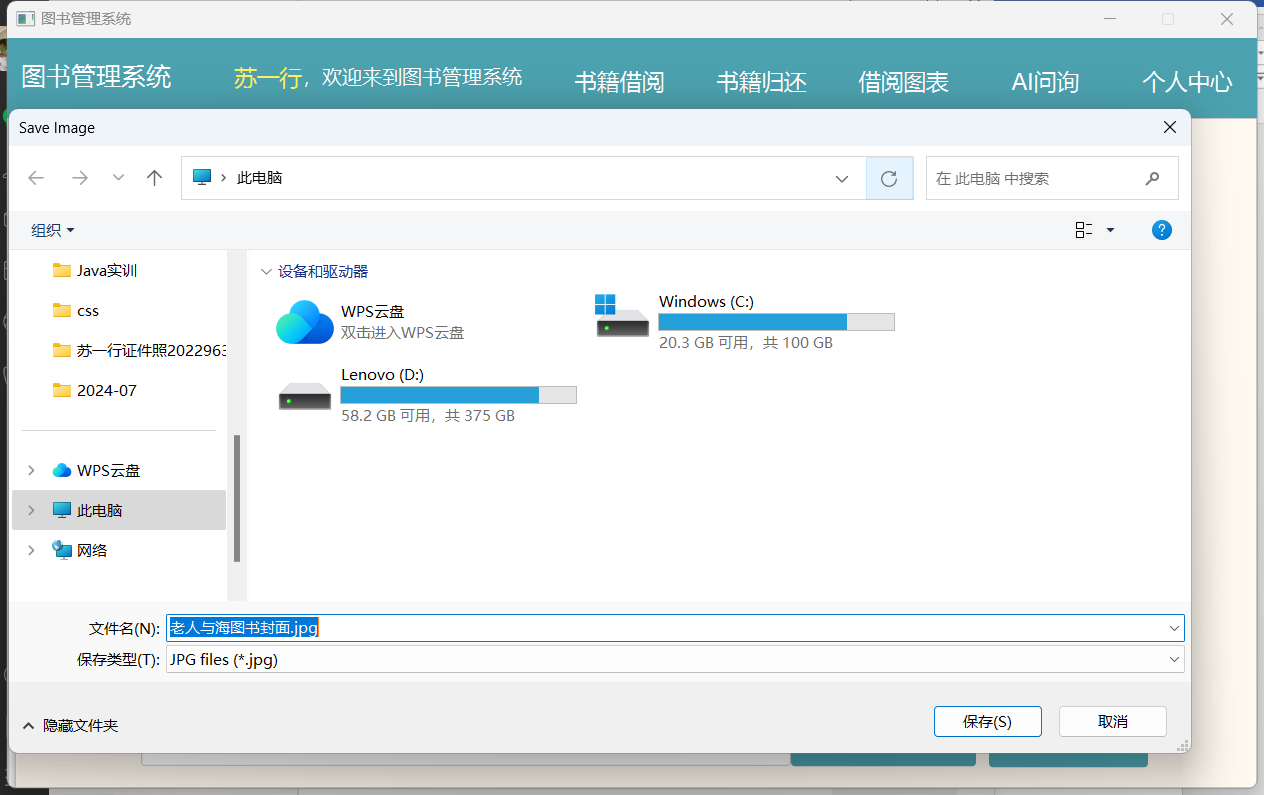


图5.18 保存图片

代码实现：

@FXML  
 **private void** initialize() {  
 addTableView();  
 setTableView();  
 addAction();  
 }  
  
 **private void** addTableView() {  
  
 **tableView**.getItems().clear();  
  
 **nameLabel**.setText(StudentUtil.*current\_student*.getStudentName());  
 *// 初始化表格的列* **tableColumn**.setCellValueFactory(**new** PropertyValueFactory<>(**"text"**));  
  
 *// 将 ArrayList 转为 ObservableList 并添加到表格中* **tableView**.setItems(convertToObservableList(AIGCUtil.*aigCrecords*));  
 }  
  
 **private** ObservableList<AIGCrecord> convertToObservableList(ArrayList<AIGCrecord> bookList) {  
 **return** FXCollections.*observableArrayList*(bookList);  
 }  
  
 @FXML  
 **void** sentMessage(ActionEvent event) {  
 sentMess();  
 insertAIGC(AIGCUtil.*currentAI*);  
 addTableView();  
 }  
  
 *// 将数据发送到TextArea中, 同时传递给大模型* **private void** sentMess() {  
 String message = **textField**.getText().trim();  
 **textField**.setText(**""**);  
  
 **if** (!StringUtil.*empty*(message)) {  
  
 **GPTTextArea**.appendText(**"\nUser : "** + message);  
 *// 创建后台任务* Task<String> task = **new** Task<String>() {  
 @Override  
 **protected** String call() **throws** Exception {  
 *// 调用后台函数并传递输入消息* **return** WAIDialogue.*vivogptLibrary*(message, AIGCUtil.*currentAI*.getSessionId());  
 }  
 };  
  
 *// 任务完成后的操作* task.setOnSucceeded(event -> {  
 String result = task.getValue();  
 **GPTTextArea**.appendText(**"\nReport: "** + result + **"\n"**);  
 insertAIGC(AIGCUtil.*currentAI*);  
 });  
  
 *// 任务失败后的操作* task.setOnFailed(event -> {  
 **GPTTextArea**.appendText(**"\nTask failed: "** + task.getException().getMessage());  
 });  
  
 *// 启动后台任务* Thread thread = **new** Thread(task);  
 thread.setDaemon(**true**); *// 确保后台线程在应用程序退出时也能结束* thread.start();  
 }  
 }  
  
 **private void** insertAIGC(AIGCrecord aigCrecord) {  
  
 **if** (!StringUtil.*empty*(**GPTTextArea**.getText()) && AIGCUtil.*currentAI* != **null**) {  
 AIGCUtil.*currentAI*.setText(**GPTTextArea**.getText());  
 }  
  
 **if** (!exitAIGC(aigCrecord)) {  
 AIGCUtil.*aigCrecords*.add(aigCrecord);  
 }  
 }  
  
 **private boolean** exitAIGC(AIGCrecord aigCrecord) {  
 **for** (AIGCrecord aigCrecord1:AIGCUtil.*aigCrecords*) {  
 **if** (aigCrecord1.getSessionId().equals(aigCrecord.getSessionId())) {  
 **return true**;  
 }  
 }  
 **return false**;  
 }  
  
 **private void** setTableView() {  
 Platform.*runLater*(() -> {  
 TableViewSkin<?> skin = (TableViewSkin<?>) **tableView**.getSkin();  
 **if** (skin != **null**) {  
 **for** (Node node : skin.getChildren()) {  
 **if** (node **instanceof** ScrollBar) {  
 ScrollBar scrollBar = (ScrollBar) node;  
 **if** (scrollBar.getOrientation() == Orientation.***HORIZONTAL***) {  
 scrollBar.setDisable(**true**);  
 scrollBar.setVisible(**false**);  
 System.***out***.println(**"Horizontal scroll bar disabled and hidden."**);  
 }  
 }  
 }  
 } **else** {  
 System.***out***.println(**"TableView skin is not initialized."**);  
 }  
 });  
 }  
  
  
  
 *// 点击表格事件* **private void** addAction(){  
 *// 添加双击事件处理* **tableView**.setRowFactory(tv -> {  
 TableRow<AIGCrecord> row = **new** TableRow<>();  
 row.setOnMouseClicked(event -> {  
 **if** (event.getButton() == MouseButton.***PRIMARY*** && event.getClickCount() == 2 && !row.isEmpty()) {  
 AIGCUtil.*currentAI* = row.getItem();  
 handleRowDoubleClick();  
 }  
 });  
 **return** row;  
 });  
 }  
  
 **private void** handleRowDoubleClick() {  
 *// 处理双击事件* **GPTTextArea**.setText(AIGCUtil.*currentAI*.getText());  
 }  
  
 String **text**;  
 String **imageUrl** = **null**;  
  
 @FXML  
 **public void** imageButAction(ActionEvent actionEvent) {  
 **text** = **imageTF**.getText();  
 **if** (StringUtil.*empty*(**text**)) {  
 DialogUtil.*showAlert*(**"提示"**, **"请先输入内容"**);  
 **return**;  
 }  
  
 **imageTF**.setText(**""**);  
  
 *// 创建一个Task来加载图像* Task<Image> loadImageTask = **new** Task<>() {  
 @Override  
 **protected** Image call() **throws** Exception {  
 **imageUrl** = WAIPainting.*waitForImage*(WAIPainting.*task\_submit*(**text**));  
 **return new** Image(**imageUrl**);  
 }  
  
 @Override  
 **protected void** succeeded() {  
 **super**.succeeded();  
 *// 在UI线程中更新ImageView* **imageView**.setImage(getValue());  
 }  
  
 @Override  
 **protected void** failed() {  
 **super**.failed();  
 *// 处理任务失败的情况* System.***err***.println(**"Image loading failed: "** + getException().getMessage());  
 }  
 };  
  
 *// 创建一个新的线程来执行任务* **new** Thread(loadImageTask).start();  
  
 }  
  
 @FXML  
 **public void** downloadButAction() {  
  
 System.***out***.println(111);  
  
 **if** (StringUtil.*empty*(**imageUrl**)) {  
 DialogUtil.*showAlert*(**"提示"**, **"请先生成图片"**);  
 **return**;  
 }  
 String destinationFile = **text** + **".jpg"**;  
  
 immm();  
  
 }  
  
 **private void** immm() {  
 *// 构建FileChooser* FileChooser fileChooser = **new** FileChooser();  
 fileChooser.setTitle(**"Save Image"**);  
  
 *// 设置默认的文件名和文件类型过滤器* fileChooser.setInitialFileName(**text** + **".jpg"**);  
 FileChooser.ExtensionFilter extFilter = **new** FileChooser.ExtensionFilter(**"JPG files (\*.jpg)"**, **"\*.jpg"**);  
 fileChooser.getExtensionFilters().add(extFilter);  
  
 *// 显示文件保存对话框* File file = fileChooser.showSaveDialog(Main.*getMainStage*());  
  
 *// 如果用户选择了文件，开始下载和保存图片* **if** (file != **null**) {  
*// imageUrl = "https://ai-painting-image.vivo.com.cn/ai-painting/763783fb8ee6844ee6b35b5ba473b6e6006b69b9-0.jpg";* downloadImage(**imageUrl**, file);  
 }  
 }  
  
 **private void** downloadImage(String imageUrl, File file) {  
 **try** {  
 URL url = **new** URL(imageUrl);  
 InputStream inputStream = url.openStream();  
 OutputStream outputStream = **new** FileOutputStream(file);  
  
 **byte**[] buffer = **new byte**[2048];  
 **int** length;  
  
 **while** ((length = inputStream.read(buffer)) != -1) {  
 outputStream.write(buffer, 0, length);  
 }  
  
 inputStream.close();  
 outputStream.close();  
  
 System.***out***.println(**"Image downloaded successfully to: "** + file.getAbsolutePath());  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 }  
 }

## 5.8个人中心的实现

功能描述：个人模板实现，可以修改个人信息

界面截图：



图5.19 个人中心

代码实现：

@FXML  
**void** Modifications(ActionEvent event) {  
 **if** (DataUpdate.*updateStudent*(**studentIdTF**.getId(), **passwordTF**.getText(), **studentNameTF**.getText(),  
 **facultyTF**.getText(), **telephoneTF**.getText())) {  
 DialogUtil.*showAlert*(**"提示"**, **"修改成功"**);  
 } **else** {  
 DialogUtil.*showAlert*(**"提示"**, **"修改失败,输入结果不得为空"**);  
 }  
}

@FXML  
**public void** initialize(){  
  
 **nameLabel**.setText(StudentUtil.*current\_student*.getStudentName());  
  
 Student student = DataSelect.*findStudentByID*(StudentUtil.*current\_student*.getStudentId());  
  
 *// 基本信息* **studentIdTF**.setText(student.getStudentId());  
 **passwordTF**.setText(student.getPassword());  
 **studentNameTF**.setText(student.getStudentName());  
 **genderTF**.setText(student.getGender());  
 **facultyTF**.setText(student.getFaculty());  
 **yearTF**.setText(String.*valueOf*(student.getYear()));  
 **telephoneTF**.setText(student.getTelephone());  
  
 *// 借阅信息* **borrowTimesTF**.setText(String.*valueOf*(student.getBorrowTimes()));  
 **lateReturnTimesTF**.setText(String.*valueOf*(student.getLateReturnTimes()));  
 **creditValueTF**.setText(String.*valueOf*(student.getCreditValue()));  
 **creditRatingTF**.setText(student.getCreditRating());  
 **borrowLimit**.setText(String.*valueOf*(student.getBorrowLimit()));  
 **borrowedBookNumTF**.setText(String.*valueOf*(student.getBorrowedBookNum()));

## 5.9管理端的实现

功能描述：管理端增删查改的实现

界面截图：



图5.20 书籍管理：



图5.21 用户管理：



图5.22 借阅管理



图5.23 损坏管理

代码实现：

@FXML  
 **void** bookManageAction(MouseEvent event) {  
 **bookManagePanel**.setVisible(**true**);  
 **userManagePanel**.setVisible(**false**);  
 **borrowManagePanel**.setVisible(**false**);  
 **loseManagePanel**.setVisible(**false**);  
 }  
  
 @FXML  
 **void** userManageAction(MouseEvent event) {  
 **bookManagePanel**.setVisible(**false**);  
 **userManagePanel**.setVisible(**true**);  
 **borrowManagePanel**.setVisible(**false**);  
 **loseManagePanel**.setVisible(**false**);  
 }  
  
 @FXML  
 **void** borrowManageAction(MouseEvent event) {  
 **bookManagePanel**.setVisible(**false**);  
 **userManagePanel**.setVisible(**false**);  
 **borrowManagePanel**.setVisible(**true**);  
 **loseManagePanel**.setVisible(**false**);  
 }  
  
 @FXML  
 **void** loseManageAction(MouseEvent event) {  
 **bookManagePanel**.setVisible(**false**);  
 **userManagePanel**.setVisible(**false**);  
 **borrowManagePanel**.setVisible(**false**);  
 **loseManagePanel**.setVisible(**true**);  
 }  
  
 @FXML  
 **void** logoutAction(MouseEvent event) {  
 **try** {  
 Main.*switchScene*(**"/com/suyihang/login.fxml"**);  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 }  
 }  
  
 @FXML  
 **private void** initialize() {  
 **nameLabel**.setText(AdminUtil.*currentAdmin*.getAdminName());  
   
 Thread thread1 = **new** Thread(() -> setBookTableView(DataSelectBook.*selectAllBooks*()));  
 Thread thread2 = **new** Thread(**this**::addBookTableViweAction);  
 Thread thread3 = **new** Thread(() -> setUserTableView(DataSelect.*findAllStudents*()));  
 Thread thread4 = **new** Thread(() -> addUserTableViweAction());  
 Thread thread5 = **new** Thread(() -> setBorrowTableView(DataSelectBook.*findAllBorrows*()));  
 Thread thread6 = **new** Thread(() -> setLoseTableView(DataSelect.*findAllLose*()));  
  
 thread1.start();  
 thread2.start();  
 thread3.start();  
 thread4.start();  
 thread5.start();  
 thread6.start();  
  
*// 等待所有线程执行完成* **try** {  
 thread1.join();  
 thread2.join();  
 thread3.join();  
 thread4.join();  
 thread5.join();  
 thread6.join();  
 } **catch** (InterruptedException e) {  
 e.printStackTrace();  
 }  
  
  
 addButtonAction();  
  
 addImportTFAction();  
 }  
  
 **private void** setBookTableView(ArrayList<Book> bookList) {  
 **bookTableView**.getItems().clear();  
  
 *// 初始化表格的列* **bookName**.setCellValueFactory(**new** PropertyValueFactory<>(**"title"**));  
 **bookAuthor**.setCellValueFactory(**new** PropertyValueFactory<>(**"author"**));  
 **bookISBN**.setCellValueFactory(**new** PropertyValueFactory<>(**"ISBN"**));  
 **bookNum**.setCellValueFactory(**new** PropertyValueFactory<>(**"repertory"**));  
  
 *// 将 ArrayList 转为 ObservableList 并添加到表格中* **bookTableView**.setItems(FXCollections.*observableArrayList*(bookList));  
 }  
  
 **private void** setUserTableView(ArrayList<Student> studentList) {  
 **userTableView**.getItems().clear();  
  
 *// 初始化表格的列* **userID**.setCellValueFactory(**new** PropertyValueFactory<>(**"studentId"**));  
 **userPassword**.setCellValueFactory(**new** PropertyValueFactory<>(**"password"**));  
 **userName**.setCellValueFactory(**new** PropertyValueFactory<>(**"studentName"**));  
 **userTelephone**.setCellValueFactory(**new** PropertyValueFactory<>(**"telephone"**));  
  
 *// 将 ArrayList 转为 ObservableList 并添加到表格中* **userTableView**.setItems(FXCollections.*observableArrayList*(studentList));  
 }  
  
 **private void** setBorrowTableView(ArrayList<Borrow> borrowList) {  
 **borrowTableView**.getItems().clear();  
  
 *// 初始化表格的列* **borrowName**.setCellValueFactory(**new** PropertyValueFactory<>(**"name"**));  
 **borrowBookTitle**.setCellValueFactory(**new** PropertyValueFactory<>(**"title"**));  
 **borrowDate**.setCellValueFactory(**new** PropertyValueFactory<>(**"borrowTime"**));  
 **borrowRemainDate**.setCellValueFactory(**new** PropertyValueFactory<>(**"days"**));  
  
 *// 将 ArrayList 转为 ObservableList 并添加到表格中* **borrowTableView**.setItems(FXCollections.*observableArrayList*(borrowList));  
 }  
  
 **private void** setLoseTableView(ArrayList<Lose> loseList) {  
 **loseTableView**.getItems().clear();  
  
 *// 初始化表格的列* **loseName**.setCellValueFactory(**new** PropertyValueFactory<>(**"Name"**));  
 **loseBookTitle**.setCellValueFactory(**new** PropertyValueFactory<>(**"title"**));  
 **loseStatus**.setCellValueFactory(**new** PropertyValueFactory<>(**"status"**));  
 **loseNotes**.setCellValueFactory(**new** PropertyValueFactory<>(**"remark"**));  
  
 *// 将 ArrayList 转为 ObservableList 并添加到表格中* **loseTableView**.setItems(FXCollections.*observableArrayList*(loseList));  
 }  
  
 **private void** addBookTableViweAction(){  
 *// 添加双击事件处理* **bookTableView**.setRowFactory(tv -> {  
 TableRow<Book> row = **new** TableRow<>();  
 row.setOnMouseClicked(event -> {  
 **if** (event.getButton() == MouseButton.***PRIMARY*** && event.getClickCount() == 1 && !row.isEmpty()) {  
 Book selectedBook = **bookTableView**.getSelectionModel().getSelectedItem();  
 **bookISBNTF**.setText(selectedBook.getISBN());  
 **bookNameTF**.setText(selectedBook.getTitle());  
 **bookAuthorTF**.setText(selectedBook.getAuthor());  
 **bookPublicTF**.setText(selectedBook.getPublisher());  
 **bookPublicDateTF**.setText(String.*valueOf*(selectedBook.getPublicationDate()));  
 **bookTotalNumTF**.setText(String.*valueOf*(selectedBook.getTotal()));  
 **bookPriceTF**.setText(String.*valueOf*(selectedBook.getPrice()));  
 **bookTypeTF**.setText(selectedBook.getType());  
 }  
 });  
 **return** row;  
 });  
 }  
  
 **private void** addUserTableViweAction(){  
 *// 添加双击事件处理* **userTableView**.setRowFactory(tv -> {  
 TableRow<Student> row = **new** TableRow<>();  
 row.setOnMouseClicked(event -> {  
 **if** (event.getButton() == MouseButton.***PRIMARY*** && event.getClickCount() == 1 && !row.isEmpty()) {  
 Student selectedBorrow = **userTableView**.getSelectionModel().getSelectedItem();  
 **userAccountTF**.setText(selectedBorrow.getStudentId());  
 **userPasswordTF**.setText(selectedBorrow.getPassword());  
 **userNameTF**.setText(selectedBorrow.getStudentName());  
 **userGenderTF**.setText(selectedBorrow.getGender());  
 **userCreditValueTF**.setText(String.*valueOf*(selectedBorrow.getCreditValue()));  
 **userCollageTF**.setText(selectedBorrow.getFaculty());  
 **userYearTF**.setText(String.*valueOf*(selectedBorrow.getYear()));  
 **userTelephoneTF**.setText(selectedBorrow.getTelephone());  
 }  
 });  
 **return** row;  
 });  
 }  
  
 **private void** addButtonAction() {  
 **boolDelete**.setOnAction(e -> {  
 **boolean** b = DialogUtil.*showConfirmationDialog*(**"提示"**, **"是否确认删除"**);  
 System.***out***.println(b);  
  
 **if** (b) {  
 DialogUtil.*showAlert*(**"提示"**, **"删除失败,该书籍未全部收回"**);  
 }  
  
 });  
  
 **bookAdd**.setOnAction(e -> {  
 DialogUtil.*showAlert*(**"提示"**, **"书籍编号重复"**);  
 });  
  
 **bookUpdate**.setOnAction(e -> {  
 DialogUtil.*showAlert*(**"提示"**, **"修改成功"**);  
 });  
  
 **userDelete**.setOnAction(e -> {  
 **boolean** b = DialogUtil.*showConfirmationDialog*(**"提示"**, **"是否确认删除"**);  
 System.***out***.println(b);  
  
 **if** (b) {  
 DialogUtil.*showAlert*(**"提示"**, **"删除失败,该用户有书籍未归还"**);  
 }  
  
 });  
  
 **userAdd**.setOnAction(e -> {  
 DialogUtil.*showAlert*(**"提示"**, **"学号重复"**);  
 });  
  
 **userUpdate**.setOnAction(e -> {  
 DialogUtil.*showAlert*(**"提示"**, **"修改成功"**);  
 });  
 }  
  
 String **fileName** = **null**;  
  
 *// 设置文件拖动到文本框的事件* **private void** addImportTFAction(){  
 *// 设置拖动进入的事件* **userImportTF**.setOnDragOver(event -> {  
 **if** (event.getGestureSource() != **userImportTF** && event.getDragboard().hasFiles()) {  
 event.acceptTransferModes(TransferMode.***COPY***);  
 }  
 event.consume();  
 });  
  
 *// 设置拖动释放的事件* **userImportTF**.setOnDragDropped(event -> {  
 Dragboard db = event.getDragboard();  
 **boolean** success = **false**;  
 **if** (db.hasFiles()) {  
 success = **true**;  
 File file = db.getFiles().get(0); *// 获取第一个拖动的文件* **userImportTF**.setText(file.getAbsolutePath()); *// 显示文件的绝对路径* **fileName** = file.getAbsolutePath();  
 }  
 event.setDropCompleted(success);  
 event.consume();  
 });  
 }  
  
 @FXML  
 **public void** userImportButAction(ActionEvent actionEvent) {  
  
 **if** (StringUtil.*empty*(**userImportTF**.getText().trim())) {  
 FileChooser fileChooser = **new** FileChooser();  
 File file = fileChooser.showOpenDialog(Main.*getMainStage*());  
 **if** (file != **null**) {  
 **userImportTF**.setText(file.getAbsolutePath());  
 **fileName** = file.getAbsolutePath();  
 } **else** {  
 **return**;  
 }  
 } **else** {  
 **fileName** = **userImportTF**.getText();  
 }  
  
 **if** (**fileName**.endsWith(**".xlsx"**)) {  
 ExcelToDatabase.*importStudentToDatabase*(**fileName**);  
 DialogUtil.*showAlert*(**"提示"**, **"成功导入"** + StudentImportDataListener.*successCount* + **"条学生数据"**);  
 setUserTableView(DataSelect.*findAllStudents*());  
 } **else** {  
 DialogUtil.*showAlert*(**"提示"**, **"文件格式错误,仅支持xlsx文件"**);  
 }  
 }

## 5.10 Excel数据批量导入的实现

功能描述：Excel数据批量导入的实现

界面截图；



图5.24 数据导入：

代码实现：

**package** com.suyihang.file;  
  
**import** com.alibaba.excel.EasyExcel;  
  
**public class** ExcelToDatabase {  
  
 **public static void** importStudentToDatabase(String excelFilePath) {  
 EasyExcel.*read*(excelFilePath, StudentImport.**class**, **new** StudentImportDataListener()).sheet().doRead();  
 }  
  
}

**package** com.suyihang.file;  
  
**import** com.alibaba.excel.EasyExcel;  
**import** com.alibaba.excel.context.AnalysisContext;  
**import** com.alibaba.excel.event.AnalysisEventListener;  
**import** com.suyihang.database.DataControl;  
  
**import** java.sql.Connection;  
**import** java.sql.PreparedStatement;  
**import** java.sql.SQLException;  
**import** java.util.ArrayList;  
**import** java.util.List;  
  
**public class** StudentImportDataListener **extends** AnalysisEventListener<StudentImport> {  
 **private** List<StudentImport> **studentList** = **new** ArrayList<>();  
 **public static int** *successCount* = 0; *// 新增计数器用于记录成功插入的行数* @Override  
 **public void** invoke(StudentImport student, AnalysisContext context) {  
 **studentList**.add(student);  
 }  
  
 @Override  
 **public void** doAfterAllAnalysed(AnalysisContext context) {  
 saveData(**studentList**);  
 System.***out***.println(**"成功导入的数据条数："** + *successCount*); *// 打印成功导入的数据条数* }  
  
 **private void** saveData(List<StudentImport> students) {  
 **for** (StudentImport student : students) {  
 **if** (insertStudent(student)) {  
 *successCount*++; *// 每次成功插入增加计数* }  
 }  
 }  
  
 **private boolean** insertStudent(StudentImport student) {  
 String sql = **"INSERT INTO student (student\_id, password, student\_name, gender, faculty, year, telephone) VALUES (?, ?, ?, ?, ?, ?, ?)"**;  
 **try** (Connection conn = DataControl.*openData*();  
 PreparedStatement stmt = conn.prepareStatement(sql)) {  
  
 stmt.setString(1, student.getStudentId());  
 stmt.setString(2, student.getPassword());  
 stmt.setString(3, student.getStudentName());  
 stmt.setString(4, student.getGender());  
 stmt.setString(5, student.getFaculty());  
 stmt.setString(6, student.getYear());  
 stmt.setString(7, student.getTelephone());  
  
 **int** rowsInserted = stmt.executeUpdate();  
 **return** rowsInserted > 0; *// 返回插入成功与否的布尔值* } **catch** (SQLException e) {  
 e.printStackTrace();  
 **return false**;  
 }  
 }  
  
 **public static void** importExcelToDatabase(String excelFilePath) {  
 EasyExcel.*read*(excelFilePath, StudentImport.**class**, **new** StudentImportDataListener()).sheet().doRead();  
 }  
}

## 5.11数据库调用的实现

功能描述:数据库调用的实现，增删改查

代码实现：

**package** com.suyihang.database;  
  
**import** java.sql.\*;  
  
**public class** DataControl {  
  
 *// 打开数据库* **public static** Connection openData() **throws** SQLException {  
 Connection con = **null**;  
  
 String url = **"jdbc:mysql://127.0.0.1:3306/librarymanagementsystem?serverTimezone=UTC"**;  
 String username = **"root"**;  
 String password = **"123456"**;  
  
 con = DriverManager.*getConnection*(url, username, password);  
  
 **return** con;  
 }  
  
 *// 关闭数据库* **public static boolean** closeData(Connection con) {  
 **if** (con != **null**) {  
 **try** {  
 con.close();  
*// System.out.println("Database connection closed successfully");* **return** con.isClosed();  
 } **catch** (SQLException e) {  
 System.***err***.println(**"Failed to close database connection: "** + e.getMessage());  
 }  
 }  
 **return false**; *// Connection was null or closing failed* }  
  
}

**package** com.suyihang.database;  
  
**import** java.sql.Connection;  
**import** java.sql.Date;  
**import** java.sql.PreparedStatement;  
**import** java.sql.SQLException;  
  
**public class** DataInsert {  
  
 */\*\*  
 \* 插入借阅记录到借阅表  
 \*  
 \** ***@param studentID*** *学生ID  
 \** ***@param ISBN*** *图书ISBN  
 \** ***@param borrowTime*** *借阅时间  
 \** ***@param returnTime*** *应归还时间  
 \** ***@param actualReturnTime*** *实际归还时间（可为空）  
 \** ***@return*** *插入记录的结果，成功返回1，失败返回0  
 \*/* **public static boolean** insertBorrow(String studentID, String ISBN, Date borrowTime, Date returnTime, Date actualReturnTime) {  
 String sql = **"INSERT INTO borrow (student\_id, ISBN, borrow\_time, return\_time, actual\_return\_time) "** +  
 **"VALUES (?, ?, ?, ?, ?)"**;  
  
 **try** (Connection conn = DataControl.*openData*();  
 PreparedStatement stmt = conn.prepareStatement(sql)) {  
  
 stmt.setString(1, studentID);  
 stmt.setString(2, ISBN);  
 stmt.setDate(3, borrowTime);  
 stmt.setDate(4, returnTime);  
 **if** (actualReturnTime != **null**) {  
 stmt.setDate(5, actualReturnTime);  
 } **else** {  
 stmt.setNull(5, java.sql.Types.***DATE***);  
 }  
  
 **return** stmt.executeUpdate() != 0;  
  
  
 } **catch** (SQLException e) {  
 e.printStackTrace();  
 **return false**;  
 }  
 }  
  
 *// 插入丢失记录表,若成功则返回true* **public static boolean** insertLose(String studentId, String isbn, String status, Date reportTime, **int** isCompensation, String remark) {  
 String sql = **"INSERT INTO lose (student\_id, ISBN, status, report\_time, is\_compensation, remark) VALUES (?, ?, ?, ?, ?, ?)"**;  
 **boolean** success = **false**;  
  
 **try** (Connection conn = DataControl.*openData*();  
 PreparedStatement stmt = conn.prepareStatement(sql)) {  
  
 stmt.setString(1, studentId);  
 stmt.setString(2, isbn);  
 stmt.setString(3, status);  
 stmt.setDate(4, **new** java.sql.Date(reportTime.getTime()));  
 stmt.setInt(5, isCompensation);  
 stmt.setString(6, remark);  
  
 **int** rowsInserted = stmt.executeUpdate();  
 **if** (rowsInserted > 0) {  
 success = **true**;  
 }  
  
 } **catch** (SQLException e) {  
 e.printStackTrace();  
 }  
  
 **return** success;  
 }  
  
  
}

**package** com.suyihang.database;  
  
**import** com.suyihang.entity.\*;  
  
**import** java.sql.\*;  
**import** java.util.ArrayList;  
  
**public class** DataSelect {  
  
 *// 根据学生ID查找学生信誉表* **public static** Credit seleceCreditbyStudentID(String ID) {  
 Connection con = **null**;  
 PreparedStatement pstmt = **null**;  
 ResultSet rs = **null**;  
 Credit credit = **null**;  
 **try** {  
 *// 打开数据库连接* con = DataControl.*openData*();  
  
 *// 准备查询SQL语句* String query = **"SELECT borrow\_times, late\_return\_times, credit\_value, credit\_rating, borrow\_limit FROM credit WHERE student\_id = ?"**;  
 pstmt = con.prepareStatement(query);  
 pstmt.setString(1, ID);  
  
 *// 执行查询* rs = pstmt.executeQuery();  
  
 *// 处理查询结果* **if** (rs.next()) {  
 **int** borrowTimes = rs.getInt(**"borrow\_times"**);  
 **int** lateReturnTimes = rs.getInt(**"late\_return\_times"**);  
 **int** creditValue = rs.getInt(**"credit\_value"**);  
 String creditRating = rs.getString(**"credit\_rating"**);  
 **int** borrowLimit = rs.getInt(**"borrow\_limit"**);  
  
 credit = **new** Credit(borrowTimes, lateReturnTimes, creditValue, creditRating, borrowLimit);  
 }  
 } **catch** (SQLException e) {  
 System.***err***.println(**"SQL exception occurred: "** + e.getMessage());  
 } **finally** {  
 *// 关闭结果集、声明和连接* **try** {  
 **if** (rs != **null**) rs.close();  
 **if** (pstmt != **null**) pstmt.close();  
 **if** (con != **null**) DataControl.*closeData*(con);  
 } **catch** (SQLException e) {  
 System.***err***.println(**"Failed to close resources: "** + e.getMessage());  
 }  
 }  
 **return** credit;  
 }  
  
 *// 通过学生ID查找该学生借阅的所有书籍记录* **public static** ArrayList<Borrow> selectBorrowByStudentID(String id) {  
 Connection con = **null**;  
 Statement stmt = **null**;  
 ResultSet rs = **null**;  
 ArrayList<Borrow> borrowList = **new** ArrayList<>();  
 **try** {  
 con = DataControl.*openData*();  
 stmt = con.createStatement();  
 String query = **"SELECT \* FROM borrow WHERE student\_id = '"** + id + **"'"**;  
 rs = stmt.executeQuery(query);  
  
 **while** (rs.next()) {  
 Borrow borrow = **new** Borrow();  
 borrow.setBorrowId(rs.getInt(**"borrow\_id"**));  
 borrow.setStudentId(rs.getString(**"student\_id"**));  
 borrow.setISBN(rs.getString(**"ISBN"**));  
 borrow.setBorrowTime(rs.getDate(**"borrow\_time"**));  
 borrow.setReturnTime(rs.getDate(**"return\_time"**));  
 borrow.setActualReturnTime(rs.getDate(**"actual\_return\_time"**));  
 borrowList.add(borrow);  
 }  
 } **catch** (SQLException e) {  
 System.***err***.println(**"SQL exception occurred: "** + e.getMessage());  
 } **finally** {  
 **try** {  
 **if** (rs != **null**) rs.close();  
 **if** (stmt != **null**) stmt.close();  
 **if** (con != **null**) DataControl.*closeData*(con);  
 } **catch** (SQLException e) {  
 System.***err***.println(**"Failed to close resources: "** + e.getMessage());  
 }  
 }  
 **return** borrowList;  
 }  
  
 *// 查找该学生借阅总次数，即该学生ID在借阅表中出现的次数(true)  
 // 查找书籍被借阅次数， 即书籍编号在借阅表中出现的次数(false)* **public static int** getBorrowTimes(String Id, **boolean** isStudent) {  
 Connection conn = **null**;  
 PreparedStatement stmt = **null**;  
 ResultSet rs = **null**;  
 **int** borrowTimes = 0;  
 String sql;  
  
 **try** {  
 conn = DataControl.*openData*();  
  
 **if** (isStudent){  
 sql = **"SELECT COUNT(\*) AS borrow\_times FROM borrow WHERE student\_id = ?"**;  
 } **else** {  
 sql = **"SELECT COUNT(\*) AS borrow\_times FROM borrow WHERE ISBN = ?"**;  
 }  
  
*// sql = "SELECT COUNT(\*) AS borrow\_times FROM borrow WHERE student\_id = ?";* stmt = conn.prepareStatement(sql);  
 stmt.setString(1, Id);  
 rs = stmt.executeQuery();  
  
 **if** (rs.next()) {  
 borrowTimes = rs.getInt(**"borrow\_times"**);  
 }  
 } **catch** (SQLException e) {  
 e.printStackTrace();  
 *// 可以根据具体情况处理异常* } **finally** {  
 **try** {  
 **if** (rs != **null**) rs.close();  
 **if** (stmt != **null**) stmt.close();  
 **if** (conn != **null**) conn.close();  
 } **catch** (SQLException e) {  
 e.printStackTrace();  
 }  
 }  
  
 **return** borrowTimes;  
 }  
  
 *// 查找该学生延迟归还次数* **public static int** getLateReturnTimes(String StrdentID){  
 Connection conn = **null**;  
 PreparedStatement stmt = **null**;  
 ResultSet rs = **null**;  
 **int** borrowTimes = 0;  
  
 **try** {  
 conn = DataControl.*openData*();  
  
 String sql = **"SELECT COUNT(\*) AS late\_return\_times FROM borrow WHERE actual\_return\_time > return\_time AND student\_id = ?"**;  
  
*// sql = "SELECT COUNT(\*) AS borrow\_times FROM borrow WHERE student\_id = ?";* stmt = conn.prepareStatement(sql);  
 stmt.setString(1, StrdentID);  
 rs = stmt.executeQuery();  
  
 **if** (rs.next()) {  
 borrowTimes = rs.getInt(**"late\_return\_times"**);  
 }  
 } **catch** (SQLException e) {  
 e.printStackTrace();  
 *// 可以根据具体情况处理异常* } **finally** {  
 **try** {  
 **if** (rs != **null**) rs.close();  
 **if** (stmt != **null**) stmt.close();  
 **if** (conn != **null**) conn.close();  
 } **catch** (SQLException e) {  
 e.printStackTrace();  
 }  
 }  
  
 **return** borrowTimes;  
 }  
  
 *//通过学生ID查找学生* **public static** Student findStudentByID(String id) {  
 Connection con = **null**;  
 PreparedStatement pstmt = **null**;  
 ResultSet rs = **null**;  
 Student student = **null**;  
  
 **try** {  
 con = DataControl.*openData*();  
 String query = **"SELECT \* FROM student WHERE student\_id = ?"**;  
 pstmt = con.prepareStatement(query);  
 pstmt.setString(1, id);  
 rs = pstmt.executeQuery();  
  
 **if** (rs.next()) {  
 student = **new** Student(  
 rs.getString(**"student\_id"**),  
 rs.getString(**"password"**),  
 rs.getString(**"student\_name"**),  
 rs.getString(**"gender"**),  
 rs.getString(**"faculty"**),  
 rs.getDate(**"year"**),  
 rs.getString(**"telephone"**)  
 );  
 }  
 } **catch** (SQLException e) {  
 System.***err***.println(**"SQL 异常发生: "** + e.getMessage());  
 } **finally** {  
 **try** {  
 **if** (rs != **null**) rs.close();  
 **if** (pstmt != **null**) pstmt.close();  
 **if** (con != **null**) DataControl.*closeData*(con);  
 } **catch** (SQLException e) {  
 System.***err***.println(**"关闭资源失败: "** + e.getMessage());  
 }  
 }  
  
 **return** student;  
 }  
  
 *// 获取当前学生借出书籍的数量* **public static int** getNumberStudentLent(String StudentID) {  
 **int** numberOfBooksLent = 0;  
 String sql = **"SELECT COUNT(\*) AS times\_borrowed "** +  
 **"FROM borrow "** +  
 **"WHERE student\_id = ? "** +  
 **"AND actual\_return\_time IS NULL"**;  
  
 **try** (Connection conn = DataControl.*openData*();  
 PreparedStatement stmt = conn.prepareStatement(sql)) {  
  
 stmt.setString(1, StudentID);  
  
 ResultSet rs = stmt.executeQuery();  
 **if** (rs.next()) {  
 numberOfBooksLent = rs.getInt(**"times\_borrowed"**);  
 }  
 } **catch** (SQLException e) {  
 e.printStackTrace();  
 }  
  
 **return** numberOfBooksLent;  
 }  
  
  
 *// 通过账号查找管理员信息* **public static** Admin findAdminByAccount(String account) {  
 Admin admin = **null**;  
 String sql = **"SELECT admin\_id, account, password, admin\_name, telephone FROM admin WHERE account = ?"**;  
  
 **try** (Connection conn = DataControl.*openData*();  
 PreparedStatement stmt = conn.prepareStatement(sql)) {  
  
 stmt.setString(1, account);  
  
 **try** (ResultSet rs = stmt.executeQuery()) {  
 **if** (rs.next()) {  
 **int** adminId = rs.getInt(**"admin\_id"**);  
 String adminAccount = rs.getString(**"account"**);  
 String password = rs.getString(**"password"**);  
 String adminName = rs.getString(**"admin\_name"**);  
 String telephone = rs.getString(**"telephone"**);  
  
 admin = **new** Admin(adminId, adminAccount, password, adminName, telephone);  
 }  
 }  
 } **catch** (SQLException e) {  
 e.printStackTrace();  
 }  
  
 **return** admin;  
 }  
  
 *// 查找所有学生* **public static** ArrayList<Student> findAllStudents() {  
 ArrayList<Student> students = **new** ArrayList<>();  
 String sql = **"SELECT student\_id, password, student\_name, gender, faculty, year, telephone FROM student"**;  
  
 **try** (Connection conn = DataControl.*openData*();  
 PreparedStatement stmt = conn.prepareStatement(sql);  
 ResultSet rs = stmt.executeQuery()) {  
  
 **while** (rs.next()) {  
 String studentId = rs.getString(**"student\_id"**);  
 String password = rs.getString(**"password"**);  
 String studentName = rs.getString(**"student\_name"**);  
 String gender = rs.getString(**"gender"**);  
 String faculty = rs.getString(**"faculty"**);  
 Date year = rs.getDate(**"year"**);  
 String telephone = rs.getString(**"telephone"**);  
  
 Student student = **new** Student(studentId, password, studentName, gender, faculty, year, telephone);  
 students.add(student);  
 }  
 } **catch** (SQLException e) {  
 e.printStackTrace();  
 }  
  
 **return** students;  
 }  
  
 *// 查找全部丢失损坏信息* **public static** ArrayList<Lose> findAllLose() {  
 ArrayList<Lose> loses = **new** ArrayList<>();  
 String sql = **"SELECT student\_id, ISBN, status, report\_time, is\_compensation, remark FROM lose"**;  
  
 **try** (Connection conn = DataControl.*openData*();  
 PreparedStatement stmt = conn.prepareStatement(sql);  
 ResultSet rs = stmt.executeQuery()) {  
  
 **while** (rs.next()) {  
 String studentId = rs.getString(**"student\_id"**);  
 String ISBN = rs.getString(**"ISBN"**);  
 String status = rs.getString(**"status"**);  
 Date reportTime = rs.getDate(**"report\_time"**);  
 **boolean** isCompensation = rs.getBoolean(**"is\_compensation"**);  
 String remark = rs.getString(**"remark"**);  
  
 Lose lose = **new** Lose(studentId, ISBN, status, reportTime, isCompensation, remark);  
 loses.add(lose);  
 }  
 } **catch** (SQLException e) {  
 e.printStackTrace();  
 }  
  
 **return** loses;  
 }  
  
 **public static void** main(String[] args) {  
*// Credit credit = seleceCreditbyStudentID("S10001");  
  
 // 查找学生借书次数* System.***out***.println(*getBorrowTimes*(**"S10001"**, **true**));  
  
 *// 查找书籍被借阅次数* System.***out***.println(*getBorrowTimes*(**"1234567890"**, **false**));  
  
 System.***out***.println(*getLateReturnTimes*(**"S10001"**));  
  
 System.***out***.println(*findStudentByID*(**"S10001"**).toString());  
  
 String account = **"suyihang"**; *// 示例账号* Admin admin = *findAdminByAccount*(account);  
 **if** (admin != **null**) {  
 System.***out***.println(admin);  
 } **else** {  
 System.***out***.println(**"Admin not found"**);  
 }  
 }  
}

**package** com.suyihang.database;  
  
**import** com.suyihang.entity.Book;  
**import** com.suyihang.entity.Borrow;  
**import** com.suyihang.entity.Type;  
  
**import** java.math.BigDecimal;  
**import** java.sql.\*;  
**import** java.util.ArrayList;  
**import** java.util.HashMap;  
**import** java.util.Map;  
  
**public class** DataSelectBook {  
 *// 查找书籍,精确查找，模糊查找  
  
 // 查找全部书籍信息* **public static** ArrayList<Book> selectAllBooks() {  
 ArrayList<Book> books = **new** ArrayList<>();  
 Connection con = **null**;  
 Statement stmt = **null**;  
 ResultSet rs = **null**;  
  
 **try** {  
 *// 打开数据库连接* con = DataControl.*openData*();  
  
 *// 创建Statement对象* stmt = con.createStatement();  
  
 *// 执行查询SQL语句* String query = **"SELECT ISBN, title, author, publisher, publication\_date, total, price, type\_id FROM book"**;  
 rs = stmt.executeQuery(query);  
  
 *// 处理查询结果* **while** (rs.next()) {  
 String isbn = rs.getString(**"ISBN"**);  
 String title = rs.getString(**"title"**);  
 String author = rs.getString(**"author"**);  
 String publisher = rs.getString(**"publisher"**);  
 Date publicationDate = rs.getDate(**"publication\_date"**);  
 **int** total = rs.getInt(**"total"**);  
 **double** price = rs.getDouble(**"price"**);  
 **int** typeId = rs.getInt(**"type\_id"**);  
  
 Book book = **new** Book(isbn, title, author, publisher, publicationDate, total, price, typeId);  
 books.add(book);  
 }  
 } **catch** (SQLException e) {  
 System.***err***.println(**"SQL exception occurred: "** + e.getMessage());  
 } **finally** {  
 *// 关闭结果集、声明和连接* **try** {  
 **if** (rs != **null**) rs.close();  
 **if** (stmt != **null**) stmt.close();  
 **if** (con != **null**) DataControl.*closeData*(con);  
 } **catch** (SQLException e) {  
 System.***err***.println(**"Failed to close resources: "** + e.getMessage());  
 }  
 }  
 **return** books;  
 }  
  
 */\*\*  
 \* 根据书籍ISBN编号查找书籍  
 \*  
 \** ***@param isbn*** *书籍的ISBN编号  
 \** ***@return*** *找到的书籍对象，如果未找到则返回null  
 \*/* **public static** Book selectBookByISBN(String isbn) {  
 Book book = **null**;  
 Connection con = **null**;  
 PreparedStatement pstmt = **null**;  
 ResultSet rs = **null**;  
  
 **try** {  
 *// 打开数据库连接* con = DataControl.*openData*();  
  
 *// 创建PreparedStatement对象* String query = **"SELECT ISBN, title, author, publisher, publication\_date, total, price, type\_id FROM book WHERE ISBN = ?"**;  
 pstmt = con.prepareStatement(query);  
 pstmt.setString(1, isbn);  
  
 *// 执行查询SQL语句* rs = pstmt.executeQuery();  
  
 *// 处理查询结果* **if** (rs.next()) {  
 String title = rs.getString(**"title"**);  
 String author = rs.getString(**"author"**);  
 String publisher = rs.getString(**"publisher"**);  
 Date publicationDate = rs.getDate(**"publication\_date"**);  
 **int** total = rs.getInt(**"total"**);  
 **double** price = rs.getDouble(**"price"**);  
 **int** typeId = rs.getInt(**"type\_id"**);  
  
 book = **new** Book(isbn, title, author, publisher, publicationDate, total, price, typeId);  
 }  
 } **catch** (SQLException e) {  
 System.***err***.println(**"SQL exception occurred: "** + e.getMessage());  
 } **finally** {  
 *// 关闭结果集、声明和连接* **try** {  
 **if** (rs != **null**) rs.close();  
 **if** (pstmt != **null**) pstmt.close();  
 **if** (con != **null**) DataControl.*closeData*(con);  
 } **catch** (SQLException e) {  
 System.***err***.println(**"Failed to close resources: "** + e.getMessage());  
 }  
 }  
 **return** book;  
 }  
  
 *// 查找书籍类别，返回值Map<int, String>  
  
 // 查找书籍全部类别返回值ArrayList<Type>* **public static** ArrayList<Type> getAllType() {  
 ArrayList<Type> types = **new** ArrayList<>();  
 Connection con = **null**;  
 Statement stmt = **null**;  
 ResultSet rs = **null**;  
  
 **try** {  
 *// 打开数据库连接* con = DataControl.*openData*();  
  
 *// 创建Statement对象* stmt = con.createStatement();  
  
 *// 执行查询SQL语句* String query = **"SELECT type\_id, type FROM type"**;  
 rs = stmt.executeQuery(query);  
  
 *// 处理查询结果* **while** (rs.next()) {  
 **int** typeId = rs.getInt(**"type\_id"**);  
 String typeName = rs.getString(**"type"**);  
  
 *// 创建Type对象并添加到列表中* Type type = **new** Type(typeId, typeName);  
 types.add(type);  
 }  
 } **catch** (SQLException e) {  
 System.***err***.println(**"SQL exception occurred: "** + e.getMessage());  
 } **finally** {  
 *// 关闭结果集、声明和连接* **try** {  
 **if** (rs != **null**) rs.close();  
 **if** (stmt != **null**) stmt.close();  
 **if** (con != **null**) DataControl.*closeData*(con);  
 } **catch** (SQLException e) {  
 System.***err***.println(**"Failed to close resources: "** + e.getMessage());  
 }  
 }  
 **return** types;  
 }  
  
 *// 查找类别id在书籍表中出现的次数* **public static int** findTypeIDinBookTable(**int** id) {  
 **int** count = 0;  
 Connection con = **null**;  
 PreparedStatement stmt = **null**;  
 ResultSet rs = **null**;  
  
 **try** {  
 *// 打开数据库连接* con = DataControl.*openData*();  
  
 *// 创建PreparedStatement对象，使用参数化查询防止SQL注入攻击* String query = **"SELECT COUNT(\*) AS count FROM book WHERE type\_id = ?"**;  
 stmt = con.prepareStatement(query);  
 stmt.setInt(1, id);  
  
 *// 执行查询* rs = stmt.executeQuery();  
  
 *// 处理查询结果* **if** (rs.next()) {  
 count = rs.getInt(**"count"**);  
 }  
 } **catch** (SQLException e) {  
 System.***err***.println(**"SQL exception occurred: "** + e.getMessage());  
 } **finally** {  
 *// 关闭结果集、声明和连接* **try** {  
 **if** (rs != **null**) rs.close();  
 **if** (stmt != **null**) stmt.close();  
 **if** (con != **null**) DataControl.*closeData*(con);  
 } **catch** (SQLException e) {  
 System.***err***.println(**"Failed to close resources: "** + e.getMessage());  
 }  
 }  
 **return** count;  
 }  
  
 *// 查找类别id在借阅表中出现的次数* **public static int** findTypeIDinBorrowTable(**int** id) {  
 **int** count = 0;  
 Connection con = **null**;  
 PreparedStatement stmt = **null**;  
 ResultSet rs = **null**;  
  
 **try** {  
 *// 打开数据库连接* con = DataControl.*openData*();  
  
 *// 创建PreparedStatement对象，使用参数化查询防止SQL注入攻击* String query = **"SELECT COUNT(\*) AS count FROM borrow b INNER JOIN book bk ON b.ISBN = bk.ISBN WHERE bk.type\_id = ?"**;  
 stmt = con.prepareStatement(query);  
 stmt.setInt(1, id);  
  
 *// 执行查询* rs = stmt.executeQuery();  
  
 *// 处理查询结果* **if** (rs.next()) {  
 count = rs.getInt(**"count"**);  
 }  
 } **catch** (SQLException e) {  
 System.***err***.println(**"SQL exception occurred: "** + e.getMessage());  
 } **finally** {  
 *// 关闭结果集、声明和连接* **try** {  
 **if** (rs != **null**) rs.close();  
 **if** (stmt != **null**) stmt.close();  
 **if** (con != **null**) DataControl.*closeData*(con);  
 } **catch** (SQLException e) {  
 System.***err***.println(**"Failed to close resources: "** + e.getMessage());  
 }  
 }  
 **return** count;  
 }  
  
 *// 查找类别表中的全部信息并以键值对形式存储  
 /\*\*  
 \* 查询所有书籍类型及其在书籍表中出现的次数  
 \** ***@return*** *包含类型名称及出现次数的映射  
 \*/* **public static** Map<String, Integer> findAllTypeinBookTableNum() {  
 Map<String, Integer> typeCounts = **new** HashMap<>();  
 Connection con = **null**;  
 Statement stmt = **null**;  
 ResultSet rs = **null**;  
  
 **try** {  
 *// 打开数据库连接* con = DataControl.*openData*();  
  
 *// 创建Statement对象* stmt = con.createStatement();  
  
 *// 执行查询SQL语句，获取所有类型* String query = **"SELECT type\_id, type FROM type"**;  
 rs = stmt.executeQuery(query);  
  
 *// 遍历每种类型* **while** (rs.next()) {  
 **int** typeId = rs.getInt(**"type\_id"**);  
 String typeName = rs.getString(**"type"**);  
  
 *// 查询该类型在书籍表中出现的次数* **int** count = *findTypeIDinBookTable*(typeId);  
  
 *// 将类型名称及其出现次数存入映射中* typeCounts.put(typeName, count);  
 }  
 } **catch** (SQLException e) {  
 System.***err***.println(**"SQL exception occurred: "** + e.getMessage());  
 } **finally** {  
 *// 关闭结果集、声明和连接* **try** {  
 **if** (rs != **null**) rs.close();  
 **if** (stmt != **null**) stmt.close();  
 **if** (con != **null**) DataControl.*closeData*(con);  
 } **catch** (SQLException e) {  
 System.***err***.println(**"Failed to close resources: "** + e.getMessage());  
 }  
 }  
 **return** typeCounts;  
 }  
  
 *// 查找所有类型书籍被借阅次数* **public static** Map<String, Integer> findAllTypeinBorrowTableNum() {  
 Map<String, Integer> typeCounts = **new** HashMap<>();  
 Connection con = **null**;  
 Statement stmt = **null**;  
 ResultSet rs = **null**;  
  
 **try** {  
 *// 打开数据库连接* con = DataControl.*openData*();  
  
 *// 创建Statement对象* stmt = con.createStatement();  
  
 *// 执行查询SQL语句，获取所有类型* String query = **"SELECT type\_id, type FROM type"**;  
 rs = stmt.executeQuery(query);  
  
 *// 遍历每种类型* **while** (rs.next()) {  
 **int** typeId = rs.getInt(**"type\_id"**);  
 String typeName = rs.getString(**"type"**);  
  
 *// 查询该类型在借阅表中出现的次数* **int** count = *findTypeIDinBorrowTable*(typeId);  
  
 *// 将类型名称及其出现次数存入映射中* typeCounts.put(typeName, count);  
 }  
 } **catch** (SQLException e) {  
 System.***err***.println(**"SQL exception occurred: "** + e.getMessage());  
 } **finally** {  
 *// 关闭结果集、声明和连接* **try** {  
 **if** (rs != **null**) rs.close();  
 **if** (stmt != **null**) stmt.close();  
 **if** (con != **null**) DataControl.*closeData*(con);  
 } **catch** (SQLException e) {  
 System.***err***.println(**"Failed to close resources: "** + e.getMessage());  
 }  
 }  
 **return** typeCounts;  
 }  
  
 *// 查找所有类型书籍被某学生借阅次数* **public static** Map<String, Integer> findAllTypeinBorrowTableNumByStudentID(String studentID) {  
 Map<String, Integer> typeCounts = **new** HashMap<>();  
 Connection con = **null**;  
 Statement stmt = **null**;  
 ResultSet rs = **null**;  
  
 **try** {  
 *// 打开数据库连接* con = DataControl.*openData*();  
  
 *// 创建Statement对象* stmt = con.createStatement();  
  
 *// 定义SQL查询语句* String query = **"SELECT t.type AS category, COUNT(\*) AS borrow\_count "** +  
 **"FROM borrow b "** +  
 **"JOIN book bk ON b.ISBN = bk.ISBN "** +  
 **"JOIN type t ON bk.type\_id = t.type\_id "** +  
 **"WHERE b.student\_id = '"** + studentID + **"' "** +  
 **"GROUP BY t.type\_id, t.type"**;  
  
 *// 执行查询* rs = stmt.executeQuery(query);  
  
 *// 处理结果集* **while** (rs.next()) {  
 String category = rs.getString(**"category"**);  
 **int** borrowCount = rs.getInt(**"borrow\_count"**);  
 typeCounts.put(category, borrowCount);  
 }  
  
 } **catch** (SQLException e) {  
 System.***err***.println(**"SQL exception occurred: "** + e.getMessage());  
 } **finally** {  
 *// 关闭结果集、声明和连接* **try** {  
 **if** (rs != **null**) rs.close();  
 **if** (stmt != **null**) stmt.close();  
 **if** (con != **null**) DataControl.*closeData*(con);  
 } **catch** (SQLException e) {  
 System.***err***.println(**"Failed to close resources: "** + e.getMessage());  
 }  
 }  
 **return** typeCounts;  
 }  
  
 **public static** Map<String, Integer> findAllType() {  
 Map<String, Integer> typeCounts = **new** HashMap<>();  
 Connection con = **null**;  
 Statement stmt = **null**;  
 ResultSet rs = **null**;  
  
 **try** {  
 *// 打开数据库连接* con = DataControl.*openData*();  
  
 *// 创建Statement对象* stmt = con.createStatement();  
  
 *// 执行查询SQL语句，获取所有类型* String query = **"SELECT type\_id, type FROM type"**;  
 rs = stmt.executeQuery(query);  
  
 *// 遍历每种类型* **while** (rs.next()) {  
 **int** typeId = rs.getInt(**"type\_id"**);  
 String typeName = rs.getString(**"type"**);  
  
 *// 将类型名称及其出现次数存入映射中* typeCounts.put(typeName, typeId);  
 }  
 } **catch** (SQLException e) {  
 System.***err***.println(**"SQL exception occurred: "** + e.getMessage());  
 } **finally** {  
 *// 关闭结果集、声明和连接* **try** {  
 **if** (rs != **null**) rs.close();  
 **if** (stmt != **null**) stmt.close();  
 **if** (con != **null**) DataControl.*closeData*(con);  
 } **catch** (SQLException e) {  
 System.***err***.println(**"Failed to close resources: "** + e.getMessage());  
 }  
 }  
 **return** typeCounts;  
 }  
  
 *// 按条件查找书籍* **public static** ArrayList<Book> findBookByCriteria(String title, String author, **int** typeID) {  
 ArrayList<Book> books = **new** ArrayList<>();  
 Connection con = **null**;  
 PreparedStatement stmt = **null**;  
 ResultSet rs = **null**;  
  
 **try** {  
 *// 打开数据库连接* con = DataControl.*openData*();  
  
 *// 构建查询语句* StringBuilder queryBuilder = **new** StringBuilder(**"SELECT \* FROM book WHERE 1=1"**);  
 **if** (title != **null** && !title.isEmpty()) {  
 queryBuilder.append(**" AND title LIKE ?"**);  
 }  
 **if** (author != **null** && !author.isEmpty()) {  
 queryBuilder.append(**" AND author LIKE ?"**);  
 }  
 **if** (typeID != 0) {  
 queryBuilder.append(**" AND type\_id = ?"**);  
 }  
  
 *// 创建PreparedStatement对象，使用参数化查询* stmt = con.prepareStatement(queryBuilder.toString());  
  
 *// 设置参数* **int** paramIndex = 1;  
 **if** (title != **null** && !title.isEmpty()) {  
 stmt.setString(paramIndex++, **"%"** + title + **"%"**);  
 }  
 **if** (author != **null** && !author.isEmpty()) {  
 stmt.setString(paramIndex++, **"%"** + author + **"%"**);  
 }  
 **if** (typeID != 0) {  
 stmt.setInt(paramIndex, typeID);  
 }  
  
 *// 执行查询* rs = stmt.executeQuery();  
  
 *// 处理查询结果* **while** (rs.next()) {  
 String ISBN = rs.getString(**"ISBN"**);  
 String bookTitle = rs.getString(**"title"**);  
 String bookAuthor = rs.getString(**"author"**);  
 String publisher = rs.getString(**"publisher"**);  
 Date publicationDate = rs.getDate(**"publication\_date"**);  
 **int** total = rs.getInt(**"total"**);  
 Double price = rs.getDouble(**"price"**);  
 **int** typeId = rs.getInt(**"type\_id"**);  
  
 *// 创建Book对象并添加到列表中* Book book = **new** Book(ISBN, bookTitle, bookAuthor, publisher, publicationDate, total, price, typeId);  
 books.add(book);  
 }  
 } **catch** (SQLException e) {  
 System.***err***.println(**"SQL exception occurred: "** + e.getMessage());  
 } **finally** {  
 *// 关闭结果集、声明和连接* **try** {  
 **if** (rs != **null**) rs.close();  
 **if** (stmt != **null**) stmt.close();  
 **if** (con != **null**) DataControl.*closeData*(con);  
 } **catch** (SQLException e) {  
 System.***err***.println(**"Failed to close resources: "** + e.getMessage());  
 }  
 }  
 **return** books;  
 }  
  
 **public static** String findBookTypeByID(**int** id) {  
 String typeName = **null**;  
 Connection con = **null**;  
 Statement stmt = **null**;  
 ResultSet rs = **null**;  
  
 **try** {  
 *// 打开数据库连接* con = DataControl.*openData*();  
  
 *// 创建Statement对象* stmt = con.createStatement();  
  
 *// 执行查询SQL语句，获取所有类型* String query = **"SELECT type FROM type WHERE type\_id = "** + id;  
 rs = stmt.executeQuery(query);  
  
 *// 遍历每种类型* **if** (rs.next()) {  
 typeName = rs.getString(**"type"**);  
 }  
 } **catch** (SQLException e) {  
 System.***err***.println(**"SQL exception occurred: "** + e.getMessage());  
 } **finally** {  
 *// 关闭结果集、声明和连接* **try** {  
 **if** (rs != **null**) rs.close();  
 **if** (stmt != **null**) stmt.close();  
 **if** (con != **null**) DataControl.*closeData*(con);  
 } **catch** (SQLException e) {  
 System.***err***.println(**"Failed to close resources: "** + e.getMessage());  
 }  
 }  
 **return** typeName;  
 }  
  
 *// 获取当前书籍被借出的数量  
 // 获取当前书籍被借出的数量* **public static int** getNumberofBooksLent(String ISBN) {  
 **int** numberOfBooksLent = 0;  
 String sql = **"SELECT COUNT(\*) AS times\_borrowed "** +  
 **"FROM borrow "** +  
 **"WHERE ISBN = ? "** +  
 **"AND actual\_return\_time IS NULL"**;  
  
 **try** (Connection conn = DataControl.*openData*();  
 PreparedStatement stmt = conn.prepareStatement(sql)) {  
  
 stmt.setString(1, ISBN);  
  
 ResultSet rs = stmt.executeQuery();  
 **if** (rs.next()) {  
 numberOfBooksLent = rs.getInt(**"times\_borrowed"**);  
 }  
 } **catch** (SQLException e) {  
 e.printStackTrace();  
 }  
  
 **return** numberOfBooksLent;  
 }  
  
 **public static** ArrayList<Borrow> findAllBorrows() {  
 ArrayList<Borrow> borrows = **new** ArrayList<>();  
 String sql = **"SELECT borrow\_id, student\_id, ISBN, borrow\_time, return\_time, actual\_return\_time FROM borrow"**;  
  
 **try** (Connection conn = DataControl.*openData*();  
 PreparedStatement stmt = conn.prepareStatement(sql);  
 ResultSet rs = stmt.executeQuery()) {  
  
 **while** (rs.next()) {  
 **int** borrowId = rs.getInt(**"borrow\_id"**);  
 String studentId = rs.getString(**"student\_id"**);  
 String isbn = rs.getString(**"ISBN"**);  
 Date borrowTime = rs.getDate(**"borrow\_time"**);  
 Date returnTime = rs.getDate(**"return\_time"**);  
 Date actualReturnTime = rs.getDate(**"actual\_return\_time"**);  
  
 Borrow borrow = **new** Borrow(borrowId, studentId, isbn, borrowTime, returnTime, actualReturnTime);  
 borrows.add(borrow);  
 }  
 } **catch** (SQLException e) {  
 e.printStackTrace();  
 }  
  
 **return** borrows;  
 }  
  
 *// 根据学生ID查找* **public static** ArrayList<Borrow> findAllBorrowsByStudentID(String studentID) {  
 ArrayList<Borrow> borrows = **new** ArrayList<>();  
 String sql = **"SELECT borrow\_id, student\_id, ISBN, borrow\_time, return\_time, actual\_return\_time "** +  
 **"FROM borrow WHERE student\_id = ?"**;  
  
 **try** (Connection conn = DataControl.*openData*();  
 PreparedStatement stmt = conn.prepareStatement(sql)) {  
  
 stmt.setString(1, studentID);  
  
 **try** (ResultSet rs = stmt.executeQuery()) {  
 **while** (rs.next()) {  
 **int** borrowId = rs.getInt(**"borrow\_id"**);  
 String studentId = rs.getString(**"student\_id"**);  
 String isbn = rs.getString(**"ISBN"**);  
 Date borrowTime = rs.getDate(**"borrow\_time"**);  
 Date returnTime = rs.getDate(**"return\_time"**);  
 Date actualReturnTime = rs.getDate(**"actual\_return\_time"**);  
  
 Borrow borrow = **new** Borrow(borrowId, studentId, isbn, borrowTime, returnTime, actualReturnTime);  
 borrows.add(borrow);  
 }  
 }  
 } **catch** (SQLException e) {  
 e.printStackTrace();  
 }  
  
 **return** borrows;  
 }  
  
 **public static** Date findActualReturnTimeByBorrowID(**int** borrowID) {  
 Date actualReturnTime = **null**;  
 String sql = **"SELECT actual\_return\_time FROM borrow WHERE borrow\_id = ?"**;  
  
 **try** (Connection conn = DataControl.*openData*();  
 PreparedStatement stmt = conn.prepareStatement(sql)) {  
  
 stmt.setInt(1, borrowID);  
  
 **try** (ResultSet rs = stmt.executeQuery()) {  
 **if** (rs.next()) {  
 actualReturnTime = rs.getDate(**"actual\_return\_time"**);  
 }  
 }  
  
 } **catch** (SQLException e) {  
 e.printStackTrace();  
 }  
  
 **return** actualReturnTime;  
 }  
  
 *// 在借阅表与书籍表中进行查找,返回为Map<String, Integer> 类型,其中String,存储书籍名称, Integer存储该睡觉在表中出现的次数* **public static** Map<String, Integer> findAllBookNumInBorrow() {  
 Map<String, Integer> bookCounts = **new** HashMap<>();  
 String sql = **"SELECT bk.title, COUNT(b.ISBN) AS borrow\_count "** +  
 **"FROM borrow b "** +  
 **"JOIN book bk ON b.ISBN = bk.ISBN "** +  
 **"GROUP BY bk.title"**;  
  
 **try** (Connection conn = DataControl.*openData*();  
 PreparedStatement stmt = conn.prepareStatement(sql);  
 ResultSet rs = stmt.executeQuery()) {  
  
 **while** (rs.next()) {  
 String title = rs.getString(**"title"**);  
 **int** count = rs.getInt(**"borrow\_count"**);  
 bookCounts.put(title, count);  
 }  
 } **catch** (SQLException e) {  
 e.printStackTrace();  
 }  
  
 **return** bookCounts;  
 }  
  
 *// 在借阅表与书籍表中通过学生ID进行查找,返回为Map<String, Integer> 类型,其中String,存储书籍名称, Integer存储该睡觉在表中出现的次数* **public static** Map<String, Integer> findAllBookNumInBorrowByStudentID(String studentID) {  
 Map<String, Integer> bookCounts = **new** HashMap<>();  
 String sql = **"SELECT bk.title, COUNT(b.ISBN) AS borrow\_count "** +  
 **"FROM borrow b "** +  
 **"JOIN book bk ON b.ISBN = bk.ISBN "** +  
 **"WHERE b.student\_id = ? "** +  
 **"GROUP BY bk.title"**;  
  
 **try** (Connection conn = DataControl.*openData*();  
 PreparedStatement stmt = conn.prepareStatement(sql)) {  
  
 stmt.setString(1, studentID);  
 **try** (ResultSet rs = stmt.executeQuery()) {  
 **while** (rs.next()) {  
 String title = rs.getString(**"title"**);  
 **int** count = rs.getInt(**"borrow\_count"**);  
 bookCounts.put(title, count);  
 }  
 }  
 } **catch** (SQLException e) {  
 e.printStackTrace();  
 }  
  
 **return** bookCounts;  
 }  
  
 **public static void** main(String[] args) {  
 String studentID = **"suyihang"**; *// 示例学生ID* Map<String, Integer> bookNumInBorrowByStudent = *findAllBookNumInBorrowByStudentID*(studentID);  
 bookNumInBorrowByStudent.forEach((title, count) -> {  
 System.***out***.println(**"Book Title: "** + title + **", Borrow Count: "** + count);  
 });  
 }  
  
}

## 5.12 AI调用及Http调用的实现

实验代码：

**package** com.suyihang.AI;  
  
**import** org.slf4j.Logger;  
**import** org.slf4j.LoggerFactory;  
**import** org.springframework.http.HttpHeaders;  
  
**import** java.io.UnsupportedEncodingException;  
**import** java.net.URLEncoder;  
**import** java.nio.charset.Charset;  
**import** java.util.\*;  
**import** javax.crypto.Mac;  
**import** javax.crypto.spec.SecretKeySpec;  
**import** java.nio.charset.StandardCharsets;  
  
  
**public class** VivoAuth {  
  
 **private static final** Logger ***logger*** = LoggerFactory.*getLogger*(VivoAuth.**class**);  
 **private static final** Charset ***UTF8*** = StandardCharsets.***UTF\_8***;  
  
 **private static** String generateRandomString(**int** len) {  
 String chars = **"0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz"**;  
 Random rnd = **new** Random();  
 StringBuilder sb = **new** StringBuilder(len);  
 **for** (**int** i = 0; i < len; i++)  
 sb.append(chars.charAt(rnd.nextInt(chars.length())));  
 **return** sb.toString();  
 }  
  
  
 **private static** String generateCanonicalQueryString(String queryParams) **throws** UnsupportedEncodingException {  
 **if** (queryParams == **null** || queryParams.length() <= 0) {  
 **return ""**;  
 }  
  
 HashMap<String, String> params = **new** HashMap<>();  
 String[] param = queryParams.split(**"&"**);  
 **for** (String item : param) {  
 String[] pair = item.split(**"="**);  
 **if** (pair.**length** == 2) {  
 params.put(pair[0], pair[1]);  
 } **else** {  
 params.put(pair[0], **""**);  
 }  
 }  
 SortedSet<String> keys = **new** TreeSet<>(params.keySet());  
 StringBuilder sb = **new** StringBuilder();  
 **boolean** first = **true**;  
 **for** (String key : keys) {  
 **if** (!first) {  
 sb.append(**"&"**);  
 }  
 String item = URLEncoder.*encode*(key, ***UTF8***) + **"="** + URLEncoder.*encode*(params.get(key), ***UTF8***);  
 sb.append(item);  
 first = **false**;  
 }  
  
 **return** sb.toString();  
 }  
  
 **private static** String generateSignature(String appKey, String signingString) {  
 **try** {  
 Mac mac = Mac.*getInstance*(**"HmacSHA256"**);  
 SecretKeySpec secret = **new** SecretKeySpec(appKey.getBytes(***UTF8***), mac.getAlgorithm());  
 mac.init(secret);  
 **return** Base64.*getEncoder*().encodeToString(mac.doFinal(signingString.getBytes()));  
 } **catch** (Exception err) {  
 ***logger***.error(**"create sign exception"**, err);  
 **return ""**;  
 }  
 }  
  
  
 **public static** HttpHeaders generateAuthHeaders(String appId, String appKey, String method, String uri, String queryParams)  
 **throws** UnsupportedEncodingException {  
 String nonce = *generateRandomString*(8);  
 String timestamp = String.*valueOf*(System.*currentTimeMillis*() / 1000);  
 String canonical\_query\_string = *generateCanonicalQueryString*(queryParams);  
 String signed\_headers\_string = String.*format*(**"x-ai-gateway-app-id:%s\n"** +  
 **"x-ai-gateway-timestamp:%s\nx-ai-gateway-nonce:%s"**, appId, timestamp, nonce);  
*// System.out.println(signed\_headers\_string);* String[] fields = {  
 method,  
 uri,  
 canonical\_query\_string,  
 appId,  
 timestamp,  
 signed\_headers\_string  
 };  
 **final** StringBuilder buf = **new** StringBuilder(fields.**length** \* 16);  
 **for** (**int** i = 0; i < fields.**length**; i++) {  
 **if** (i > 0) {  
 buf.append(**"\n"**);  
 }  
 **if** (fields[i] != **null**) {  
 buf.append(fields[i]);  
 }  
 }  
*// System.out.println(buf.toString());* HttpHeaders headers = **new** HttpHeaders();  
 headers.add(**"X-AI-GATEWAY-APP-ID"**, appId.toString());  
 headers.add(**"X-AI-GATEWAY-TIMESTAMP"**, timestamp.toString());  
 headers.add(**"X-AI-GATEWAY-NONCE"**, nonce.toString());  
 headers.add(**"X-AI-GATEWAY-SIGNED-HEADERS"**, **"x-ai-gateway-app-id;x-ai-gateway-timestamp;x-ai-gateway-nonce"**);  
 headers.add(**"X-AI-GATEWAY-SIGNATURE"**, *generateSignature*(appKey, buf.toString()));  
 **return** headers;  
 }  
  
 **public static void** printHeaders(HttpHeaders headers) {  
 **for** (Map.Entry<String, List<String>> entry : headers.entrySet()) {  
 System.***out***.println(entry.getKey() + **":"** + entry.getValue());  
 }  
 }  
}

**package** com.suyihang.AI;  
  
  
  
**import** com.fasterxml.jackson.core.type.TypeReference;  
**import** com.fasterxml.jackson.databind.ObjectMapper;  
**import** org.springframework.http.\*;  
**import** org.springframework.web.client.RestTemplate;  
  
**import** java.util.HashMap;  
**import** java.util.Map;  
**import** java.util.UUID;  
  
**public class** WAIDialogue {  
  
 **public static** String vivogpt(String input) **throws** Exception {  
  
 String appId = **"3034198228"**;  
 String appKey = **"nxJqCucwgccxQwGj"**;  
 String URI = **"/vivogpt/completions"**;  
 String DOMAIN = **"api-ai.vivo.com.cn"**;  
 String METHOD = **"POST"**;  
 UUID requestId = UUID.*randomUUID*();  
*// System.out.println("requestId: " + requestId);* Map<String, Object> map = **new** HashMap<>();  
 map.put(**"requestId"**, requestId.toString());  
 String queryStr = *mapToQueryString*(map);  
  
 *//构建请求体* Map<String, Object> data = **new** HashMap<>();  
  
 data.put(**"prompt"**, input);  
 data.put(**"model"**, **"vivo-BlueLM-TB"**);  
 UUID sessionId = UUID.*randomUUID*();  
 data.put(**"sessionId"**, sessionId.toString());  
  
 *// 设置角色情景* data.put(**"systemPrompt"**, **"你是负责管理图书的小助手,当用户输入书籍标题时,"** +  
 **"你需要对该书籍写出简要的介绍.同时,你也应该回答用户关于书籍的其他问题"**);  
  
 Map<String, Object> extra = **new** HashMap<>();  
 extra.put(**"temperature"**, 0.2);  
*// extra.put("max\_new\_tokens", 3); 设置最大生成字符数* data.put(**"extra"**, extra);  
  
 *printMap*(data);  
  
 HttpHeaders headers = VivoAuth.*generateAuthHeaders*(appId, appKey, METHOD, URI, queryStr);  
 headers.add(**"Content-Type"**, **"application/json"**);  
 *//System.out.println(headers);* String url = String.*format*(**"http://%s%s?%s"**, DOMAIN, URI, queryStr);  
 String requsetBodyString = **new** ObjectMapper().writeValueAsString(data);  
 RestTemplate restTemplate = **new** RestTemplate();  
 HttpHeaders httpHeaders = **new** HttpHeaders();  
 httpHeaders.setContentType(MediaType.*valueOf*(MediaType.***APPLICATION\_JSON\_VALUE***));  
 httpHeaders.addAll(headers);  
 HttpEntity<String> requestEntity = **new** HttpEntity<>(requsetBodyString, httpHeaders);  
  
 ResponseEntity<String> response = restTemplate.exchange(url, HttpMethod.***POST***, requestEntity, String.**class**);  
 **if** (response.getStatusCode() == HttpStatus.***OK***) {  
 ObjectMapper objectMapper = **new** ObjectMapper();  
 Map<String, Object> responseData = objectMapper.readValue(response.getBody(), **new** TypeReference<Map<String, Object>>() {});  
 Map<String, Object> text = (Map<String, Object>) responseData.get(**"data"**);  
 String content = (String) text.get(**"content"**);  
 *//System.out.println("Response body: " + response.getBody());  
 //System.out.println("Content: " + content);* **return** content;  
 } **else** {  
 System.***out***.println(**"Error response: "** + response.getStatusCode());  
 **return "Error response:"** + response.getStatusCode();  
 }  
  
 }  
  
  
 **public static** String vivogptLibrary(String input , String sessionId) **throws** Exception {  
  
 String appId = **"3034198228"**;  
 String appKey = **"nxJqCucwgccxQwGj"**;  
 String URI = **"/vivogpt/completions"**;  
 String DOMAIN = **"api-ai.vivo.com.cn"**;  
 String METHOD = **"POST"**;  
 UUID requestId = UUID.*randomUUID*();  
*// System.out.println("requestId: " + requestId);* Map<String, Object> map = **new** HashMap<>();  
 map.put(**"requestId"**, requestId.toString());  
 String queryStr = *mapToQueryString*(map);  
  
 *//构建请求体* Map<String, Object> data = **new** HashMap<>();  
  
 data.put(**"prompt"**, input);  
 data.put(**"model"**, **"vivo-BlueLM-TB"**);  
 data.put(**"sessionId"**, sessionId);  
  
 *// 设置角色情景* data.put(**"systemPrompt"**, **"你是负责管理图书的小助手,当用户输入书籍标题时,"** +  
 **"你需要对该书籍写出简要的介绍.同时,你也应该回答用户关于书籍的其他问题"**);  
  
  
 Map<String, Object> extra = **new** HashMap<>();  
 extra.put(**"temperature"**, 0.5);  
*// extra.put("max\_new\_tokens", 3); 设置最大生成字符数* data.put(**"extra"**, extra);  
  
*// printMap(data);* HttpHeaders headers = VivoAuth.*generateAuthHeaders*(appId, appKey, METHOD, URI, queryStr);  
 headers.add(**"Content-Type"**, **"application/json"**);  
 *//System.out.println(headers);* String url = String.*format*(**"http://%s%s?%s"**, DOMAIN, URI, queryStr);  
 String requsetBodyString = **new** ObjectMapper().writeValueAsString(data);  
 RestTemplate restTemplate = **new** RestTemplate();  
 HttpHeaders httpHeaders = **new** HttpHeaders();  
 httpHeaders.setContentType(MediaType.*valueOf*(MediaType.***APPLICATION\_JSON\_VALUE***));  
 httpHeaders.addAll(headers);  
 HttpEntity<String> requestEntity = **new** HttpEntity<>(requsetBodyString, httpHeaders);  
  
 ResponseEntity<String> response = restTemplate.exchange(url, HttpMethod.***POST***, requestEntity, String.**class**);  
 **if** (response.getStatusCode() == HttpStatus.***OK***) {  
 ObjectMapper objectMapper = **new** ObjectMapper();  
 Map<String, Object> responseData = objectMapper.readValue(response.getBody(), **new** TypeReference<Map<String, Object>>() {});  
 Map<String, Object> text = (Map<String, Object>) responseData.get(**"data"**);  
 String content = (String) text.get(**"content"**);  
 *//System.out.println("Response body: " + response.getBody());  
 //System.out.println("Content: " + content);* **return** content;  
 } **else** {  
 System.***out***.println(**"Error response: "** + response.getStatusCode());  
 **return "Error response:"** + response.getStatusCode();  
 }  
  
 }  
  
 **public static** String mapToQueryString(Map<String, Object> map) {  
 **if** (map.isEmpty()) {  
 **return ""**;  
 }  
 StringBuilder queryStringBuilder = **new** StringBuilder();  
 **for** (Map.Entry<String, Object> entry : map.entrySet()) {  
 **if** (queryStringBuilder.length() > 0) {  
 queryStringBuilder.append(**"&"**);  
 }  
 queryStringBuilder.append(entry.getKey());  
 queryStringBuilder.append(**"="**);  
 queryStringBuilder.append(entry.getValue());  
 }  
 **return** queryStringBuilder.toString();  
 }  
  
 *// 递归打印 Map 内容* **public static void** printMap(Map<String, Object> map) {  
 **for** (Map.Entry<String, Object> entry : map.entrySet()) {  
 System.***out***.print(entry.getKey() + **": "**);  
 **if** (entry.getValue() **instanceof** Map) {  
 System.***out***.println();  
 *printMap*((Map<String, Object>) entry.getValue());  
 } **else** {  
 System.***out***.println(entry.getValue());  
 }  
 }  
 }  
  
  
}

package com.suyihang.AI;

import com.alibaba.fastjson.JSON;

import com.alibaba.fastjson.JSONArray;

import com.alibaba.fastjson.JSONObject;

import org.springframework.http.HttpEntity;

import org.springframework.http.HttpHeaders;

import org.springframework.http.HttpMethod;

import org.springframework.http.HttpStatus;

import org.springframework.http.ResponseEntity;

import org.springframework.web.client.RestTemplate;

import java.io.UnsupportedEncodingException;

import java.util.HashMap;

import java.util.Map;

public class WAIPainting {

final static String APP\_ID = "3034198228";

final static String APP\_KEY = "nxJqCucwgccxQwGj";

public static String task\_submit(String input) throws UnsupportedEncodingException {

final String METHOD = "POST";

final String URI = "/api/v1/task\_submit";

final String DOMAIN = "api-ai.vivo.com.cn";

Map<String, Object> data = new HashMap<>();

data.put("height", 768);

data.put("width", 576);

data.put("prompt", input);

data.put("styleConfig", "55c682d5eeca50d4806fd1cba3628781");

String requestBody = JSON.toJSONString(data);

Map<String, Object> query = new HashMap<>();

String queryString = mapToQueryString(query);

HttpHeaders headers = VivoAuth.generateAuthHeaders(APP\_ID, APP\_KEY, METHOD, URI, queryString);

headers.add("Content-Type", "application/json");

String url = String.format("http://%s%s", DOMAIN, URI);

RestTemplate restTemplate = new RestTemplate();

HttpEntity<String> requestEntity = new HttpEntity<>(requestBody, headers);

ResponseEntity<String> response = restTemplate.exchange(url, HttpMethod.POST, requestEntity, String.class);

if (response.getStatusCode() == HttpStatus.OK) {

JSONObject jsonResponse = JSON.parseObject(response.getBody());

JSONObject result = jsonResponse.getJSONObject("result");

return result.getString("task\_id");

} else {

return null;

}

}

public static String taskProgress(String taskId) throws UnsupportedEncodingException {

final String METHOD = "GET";

final String URI = "/api/v1/task\_progress";

final String DOMAIN = "api-ai.vivo.com.cn";

Map<String, Object> data = new HashMap<>();

data.put("task\_id", taskId);

String queryString = mapToQueryString(data);

HttpHeaders headers = VivoAuth.generateAuthHeaders(APP\_ID, APP\_KEY, METHOD, URI, queryString);

String url = String.format("http://%s%s?%s", DOMAIN, URI, queryString);

headers.add("Content-Type", "application/json");

RestTemplate restTemplate = new RestTemplate();

HttpEntity requestEntity = new HttpEntity<>(headers);

ResponseEntity<String> response = restTemplate.exchange(url, HttpMethod.GET, requestEntity, String.class);

if (response.getStatusCode() == HttpStatus.OK) {

return response.getBody();

} else {

return null;

}

}

public static String waitForImage(String taskId) throws UnsupportedEncodingException, InterruptedException {

while (true) {

Thread.sleep(5000);

String progressResponse = taskProgress(taskId);

JSONObject jsonResponse = JSON.parseObject(progressResponse);

JSONObject result = jsonResponse.getJSONObject("result");

boolean finished = result.getBoolean("finished");

if (finished) {

JSONArray imagesUrlArray = result.getJSONArray("images\_url");

if (imagesUrlArray.size() > 0) {

return imagesUrlArray.getString(0);

}

}

}

}

public static String mapToQueryString(Map<String, Object> map) {

if (map.isEmpty()) {

return "";

}

StringBuilder queryStringBuilder = new StringBuilder();

for (Map.Entry<String, Object> entry : map.entrySet()) {

if (queryStringBuilder.length() > 0) {

queryStringBuilder.append("&");

}

queryStringBuilder.append(entry.getKey());

queryStringBuilder.append("=");

queryStringBuilder.append(entry.getValue());

}

return queryStringBuilder.toString();

}

}

6. **系统测试**

6.1 测试方法

数据测试：使用大量实际数据进行测试，确保各种边界条件和极端值都能得到正确处理，操作测试：检查系统的各种显示和输出是否与设计要求一致。

6.2测试用例设计

表6.1 用户登录功能测试表

|  |  |  |  |
| --- | --- | --- | --- |
| 编号 | 测试内容 | 预期结果 | 实际结果 |
| 1 | 输入正确用户名，正确密码，点击登录 | 用户登录成功，系统跳转至主页面 | 与预期相符 |
| 2 | 输入正确用户名，错误的密码，点击登录 | 用户登录失败，保留在登录页面 | 与预期相符 |
| 3 | 输入错误用户名，正确密码，点击登录 | 用户登录失败，保留在登录页面 | 与预期相符 |
| 4 | 输入错误用户名，错误密码，点击登录 | 用户登录失败，保留在登录页面 | 与预期相符 |

表6.2 用户借阅功能测试表

|  |  |  |  |
| --- | --- | --- | --- |
| 编号 | 测试内容 | 预期结果 | 实际结果 |
| 1 | 输入数据库中书籍编号点击查询 | 跳转至搜索结果页面 | 与预期相符 |
| 2 | 输入未在数据库中书籍编号点击查询 | 提示查询失败 | 与预期相符 |
| 3 | 在书籍介绍页面单击借阅 | 提示借阅成功 | 与预期相符 |
| 4 | 填写书籍信息后确认归还 | 提示归还成功 | 与预期相符 |
| 5 | 填写书籍信息后申请延时 | 提示延时成功 | 与预期相符 |
| 6 | 填写书籍信息后申请遗失 | 提示提交成功 | 与预期相符 |

表6.3 借阅图表功能测试表

|  |  |  |  |
| --- | --- | --- | --- |
| 编号 | 测试内容 | 预期结果 | 实际结果 |
| 1 | 多次借阅小说书籍并查看全体柱状图饼状图 | 图中显示小说占比份额较高 | 与预期相符 |
| 2 | 多次借阅科技和历史并查看全体柱状图饼状图 | 图中显示科技和历史书籍占比份额较高 | 与预期相符 |
| 3 | 个人初次借阅文学和历史查看个人柱状图饼状图 | 图中显示文学和历史 | 与预期相符 |
| 4 | 个人初次借阅西游记和红楼梦查看书名柱状图饼状图 | 图中显示西游记和红楼梦 | 与预期相符 |

表6.4AI问询功能测试表

|  |  |  |  |
| --- | --- | --- | --- |
| 编号 | 测试内容 | 预期结果 | 实际结果 |
| 1 | 输入西游记 | 显示一段西游记的简介 | 与预期相符 |
| 2 | 推荐书籍 | 推荐热门书籍 | 与预期相符 |

表6.5书籍功能测试表

|  |  |  |  |
| --- | --- | --- | --- |
| 编号 | 测试内容 | 预期结果 | 实际结果 |
| 1 | 填写书籍信息单击添加 | 添加成功 | 与预期相符 |
| 2 | 填写书籍信息单击修改 | 修改成功 | 与预期相符 |
| 3 | 填写书籍信息单击删除 | 删除成功 | 与预期相符 |
| 4 | 修改书籍状态为已归还 | 修改成功 | 与预期相符 |
| 5 | 修改书籍状态为已丢失 | 修改成功 | 与预期相符 |
| 6  7 | 修改书籍为页面破损  选择文件批量导入 | 修改成功  导入成功 | 与预期相符  与预期相符 |

6.3测试结论

功能测试结果分析：各项功能均可正常运行。在测试过程中，采用数据测试：使用大量实际数据进行测试，确保各种边界条件和极端值都能得到正确处理，操作测试：检查系统的各种显示和输出是否与设计要求一致。经过测试，用户端借阅管理，借阅图表，个人中心个模块功能及管理员端用户管理和书籍管理及信息汇总功能均能按设计逻辑正常运行。

测试结论：基于测试结果，得出书籍管理系统满足预期功能的结论。

# 7. 总结与展望

7.1 总结

该系统具有以下优点：

功能完善：现代图书管理系统包括书籍借阅管理、学生信息管理、借阅图表、AI问询、个人信息管理，批量导入等功能，能够有效提升图书馆管理效率。首先界面设计简洁直观，操作便捷，提升了用户体验，包括读者和管理员两个用户角色。其次数据管理能够准确记录图书的借还情况、读者的借阅历史和统计分析功能，为图书馆决策提供数据支持。最后能够根据图书馆的规模和需求进行灵活配置和定制，满足不同图书馆的管理需求。

同时该系统有些不足数据准确性和一致性：在复杂的图书馆环境中，有时候可能会出现数据不一致或者错误的情况，如图书状态更新延迟、借还记录错误等，这需要系统在数据管理方面更加严谨和精确。同时图书管理系统需要更多关注数据安全和用户隐私保护，避免遭受数据泄露或者恶意攻击的风险。

7.2展望

随着时代的进步与发展,未来图书管理系统可能加入人工智能技术，如智能推荐系统，根据读者的阅历史和兴趣推荐图书，提升借阅率和读者满意度。图书管理系统可以开发移动应用程序，读者可以通过手机或平板电脑进行借阅、续借、查询等操作，增强便利性和灵活性。图书管理系统可部署在云端，实现跨地域、跨机构的数据共享和管理，提升资源利用效率。结合电子书和数字化资源管理，使图书馆不仅管理纸质图书，还能管理电子书籍和数字资源，提升服务范围和效率。图书管理系统引入社交功能，如读者社区或讨论平台，促进读者之间的交流和分享，扩展图书馆的社会功能和影响力。