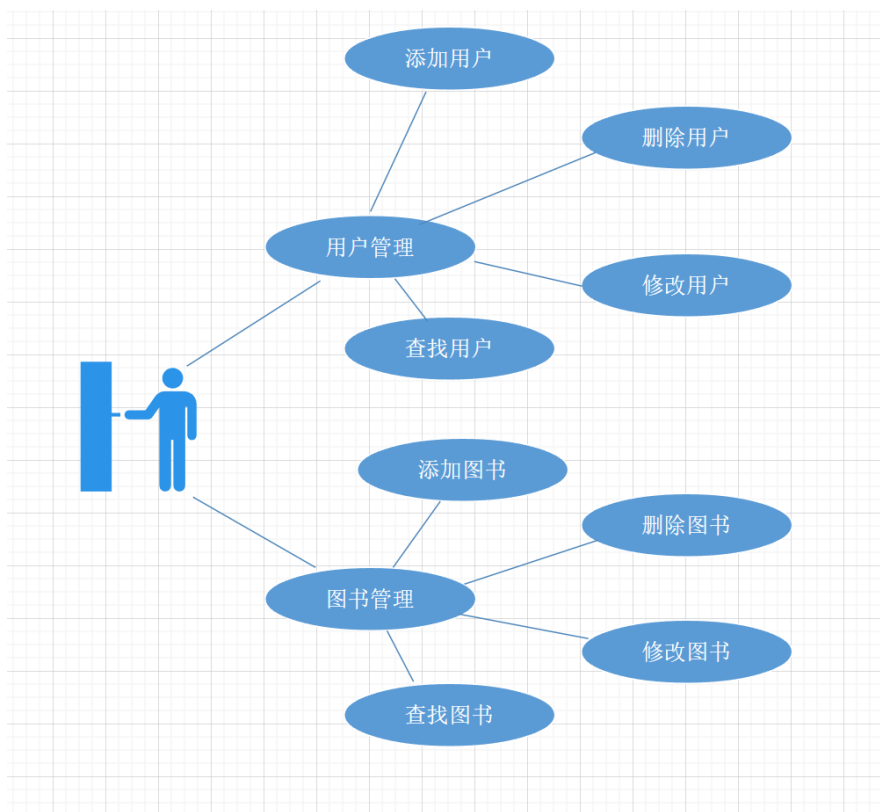


# 图书管理系统开发文档

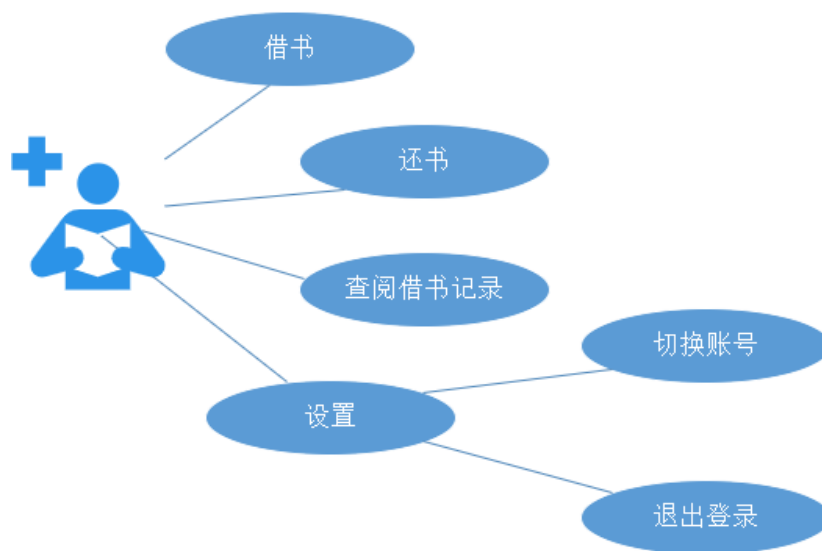
## 一、功能设计：

1. 基本图书信息包括：图书编号，书籍名称，书籍出版社，书籍作者，书籍馆藏书目，书籍出借数目，书籍出版日期。
2. 基本学生信息包括：学生学号，学生姓名，学生专业，学生班级，学生电话，学生状态。
3. 能通过交互界面输入学生信息。
4. 能通过交互界面输入图书信息。
5. 能查询指定的学生信息。
6. 能查询指定的图书信息。
7. 能修改指定的学生信息。
8. 能修改指定的图书信息。
9. 能删除指定的学生信息。
10. 能删除指定的图书信息。
11. 能浏览全部学生的主要信息：学生学号，学生姓名，学生专业，学生班级，学生电话，学生状态。
12. 能浏览全部图书的主要信息：图书编号，书籍名称，书籍出版社，书籍作者，书籍馆藏书目，书籍出借数目，书籍出版日期。
13. 能对学生信息进行文件批量导入操作。
14. 能对图书信息进行文件批量导入操作。
15. 能对学生信息进行批量导入操作。
16. 能将已经发生了变化的学生信息记录保存为一个磁盘文件。
17. 能将已经发生了变化的图书信息记录保存为一个磁盘文件。
18. 能对图书进行归还，借阅操作。
19. 拥有图书推荐功能。
20. 要求所有交互界面安全、友好、美观。

## 二、系统需求分析



图片 1 管理员需求分析



图片 2 学生需求分析

## 1. 管理员登录

**功能描述：**管理员通过交互界面输入用户名和密码登录系统。

**前置条件：**系统已启动，管理员账号存在。

**后置条件：**管理员登录成功，进入功能选择界面。

## 2. 学生登录

**功能描述：**学生通过交互界面输入学号和密码登录系统。

**前置条件：**学生账号已注册，系统正常运行。

**后置条件：**学生登录成功，进入个人信息及书籍管理界面。

## 3. 书籍管理

### 3.1 增加书籍

**功能描述：**管理员通过交互界面录入新书信息，并将其添加到书籍数据库中。

**前置条件：**管理员已登录，书籍信息完整且唯一。

**后置条件：**书籍成功添加到系统。

### 3.2 删除书籍：

**功能描述：**管理员通过交互界面输入书籍编号或其他标识信息删除特定书籍。

**前置条件：**管理员已登录，待删除书籍存在。

**后置条件：**书籍从系统中移除。

### 3.3 查询书籍：

**功能描述：**管理员或学生可通过界面查询特定书籍的详细信息。

**前置条件：**查询书籍信息存在于系统中。

**后置条件：**显示书籍详细信息或提示书籍不存在。

### 3.4 修改书籍：

**功能描述：**管理员可以修改现有书籍的信息。

**前置条件：**管理员已登录，书籍信息存在。

**后置条件：**修改后的书籍信息成功保存。

## 4. 学生管理

### 4.1 增加学生：

**功能描述：**管理员可以添加学生信息到系统中。

**前置条件：**管理员已登录，学生信息完整且唯一。

**后置条件：**学生信息成功录入系统。

### 4.2 毕业学生：

**功能描述：**标记学生为毕业状态，并处理与此相关的系统操作。

**前置条件：**管理员已登录，学生信息存在。

**后置条件：**学生状态成功更新为毕业。

## 4.3 查询学生信息：

**功能描述：**管理员可通过输入学号等方式查询学生的详细信息。

**前置条件：**管理员已登录，查询的学生存在。

**后置条件：**显示学生详细信息或提示学生不存在。

## 4.4 修改学生信息：

**功能描述：**管理员可以修改学生的个人信息。

**前置条件：**管理员已登录，学生信息存在。

**后置条件：**修改后的学生信息成功保存。

# 5. 借还书操作

## 5.1 借书：

**功能描述：**学生可以通过系统借阅书籍。

**前置条件：**学生已登录，所借书籍有库存。

**后置条件：**书籍借阅成功，库存减少。

## 5.2 还书：

**功能描述：**学生可以通过系统归还书籍。

**前置条件：**学生已登录，所还书籍处于借出状态。

**后置条件：**书籍归还成功，库存增加。

## 5.3 书籍推荐

**功能描述：**系统根据所有学生的借阅情况推荐书籍。

**前置条件：**学生已登录，系统有足够的推荐数据。

**后置条件：**向学生显示推荐书籍列表。

## 三、总体设计（类设计和功能设计）

### 1.类的静态设计（类图）

#### 1.1 组织数据和操纵数据的类

描述图书借阅的各项信息是本软件最主要的数据，因此，应该将描述图书的所有信息数据组织在一起，并与对这些数据的操作一起封装在一个类中，该类可命名为 **BookSet**。在该类的各项状态属性中，定义书籍编号、书名、作者、出版社、总库存及借出数量等信息是十分必要的。如何将所有学生和书籍信息数据组织在一起，对于安全、高效、方便地管理和访问是非常重要的，为此，应定义存储和管理学生信息的数据容器类 **StudentSet**。

同时，借阅信息作为连接学生和书籍的桥梁，是管理图书借阅流程中不可或缺的一部分。借阅信息包括书籍编号、借阅日期及书名，这些信息被封装在 **BorrowInfo** 类中，并通过 **BorrowInfoSet** 类进行管理，该类继承自标准容器，提供高效的借阅信息管理功能。

学生节点（**StudentNode**）则是一个特别设计的类，它储存了特定学生的学号及其借书列表（**BorrowInfoSet**），这样的设计使得对学生的借书行为可以进行有效跟踪和管理。此外，**StudentNodeSet** 类作为 **StudentNode** 的集合，支持对整个学生节点集的操作，如添加、查询和更新学生借书信息。

总的来说，这些类的属性和操作以及它们之间的静态关系形成了一个完整的框架，这个框架不仅确保了数据的完整性和一致性，而且提高了系统操作的效率和安全性。这种结构化和模块化的设计是现代软件工程中常见的实践，特别是在需要处理复杂数据和多层次关系的系统中。

### 1.1.1 Student 类

<i>Student</i>
<ul style="list-style-type: none"><li>- studentno: string</li><li>- studentname: string</li><li>- studentmajor: string</li><li>- studentclass: string</li><li>- studentmobile: string</li></ul>
<ul style="list-style-type: none"><li>+ Student()</li><li>+ Student(stu: Student*)</li><li>+ setstuno(no: string): void</li><li>+ setstuname(name: string): void</li><li>+ setstumajor(major: string): void</li><li>+ setstuclass(c: string): void</li><li>+ setstumobi(mobi: string): void</li><li>+ getstuno(): string</li><li>+ getstuname(): string</li><li>+ getstumajor(): string</li><li>+ getstuclass(): string</li><li>+ getstumobi(): string</li><li>+ operator==(s: Student): bool</li><li>+ index_stuno(info: string): Status</li><li>+ index_stuname(info: string): Status</li><li>+ index_stumajor(info: string): Status</li><li>+ index_stuclass(info: string): Status</li><li>+ index_stumobile(info: string): Status</li><li>+ equal_stuno(b: Student): Status</li><li>+ equal_stuma_or_cl(b: Student): Status</li><li>+ toString(): string</li><li>+ operator&gt;&gt;(is: istream&amp;, stu: Student&amp;): istream&amp;</li><li>+ operator&lt;&lt;(os: ostream&amp;, stu: Student&amp;): ostream&amp;</li></ul>

图片 3 Student 类图

属性:

studentno: string - 学号

studentname: string - 姓名

studentmajor: string - 专业

studentclass: string - 班级

studentmobile: string - 手机号

操作:

Student() - 构造函数

Student(stu: Student) - 拷贝构造函数

setstudentno(no: string): void - 设置学号

setstudentname(name: string): void - 设置姓名

setstudentmajor(major: string): void - 设置专业

setstudentclass(cls: string): void - 设置班级

setstudentmobile(mobi: string): void - 设置手机号

getstudentno(): string - 获取学号

getstudentname(): string - 获取姓名

getstudentmajor(): string - 获取专业

getstudentclass(): string - 获取班级

getstudentmobile(): string - 获取手机号

operator==(s: Student): bool - 比较操作符

operator>>(is: istream, st: Student): istream - 流输入操作

operator<<(os: ostream, st: Student): ostream - 流输出操作



### 1.1.2 Book 类

Book
<div>- bookno: string</div> <div>- bookname: string</div> <div>- author: string</div> <div>- publisher: string</div> <div>- totalnum: int</div> <div>- borrownum: int</div> <div>- pubday: Date</div>
<div>+ setbookno(no: string): void</div> <div>+ setbookname(name: string): void</div> <div>+ setauthor(au: string): void</div> <div>+ setpublisher(pub: string): void</div> <div>+ setttotal(i: int): void</div> <div>+ setborrow(i: int): void</div> <div>+ setday(t: Date): void</div> <div>+ getbookno(): string</div> <div>+ getbookname(): string</div> <div>+ getauthor(): string</div> <div>+ getpublisher(): string</div> <div>+ gettotalnum(): int</div> <div>+ getborrownum(): int</div> <div>+ getpubday(): Date&amp;</div> <div>+ operator==(b: Book): bool</div> <div>+ index_bookno(info: string): Status</div> <div>+ index_author(info: string): Status</div> <div>+ index_bookname(info: string): Status</div> <div>+ index_publisher(info: string): Status</div> <div>+ toString(): string</div> <div>+ operator&gt;&gt;(is: istream&amp;, boo: Book&amp;): is</div> <div>+ operator&lt;&lt;(os: ostream&amp;, boo: Book&amp;):</div> <div>+ operator==(book1: Book&amp;, book2: Book</div>

图片 4 Book 类图

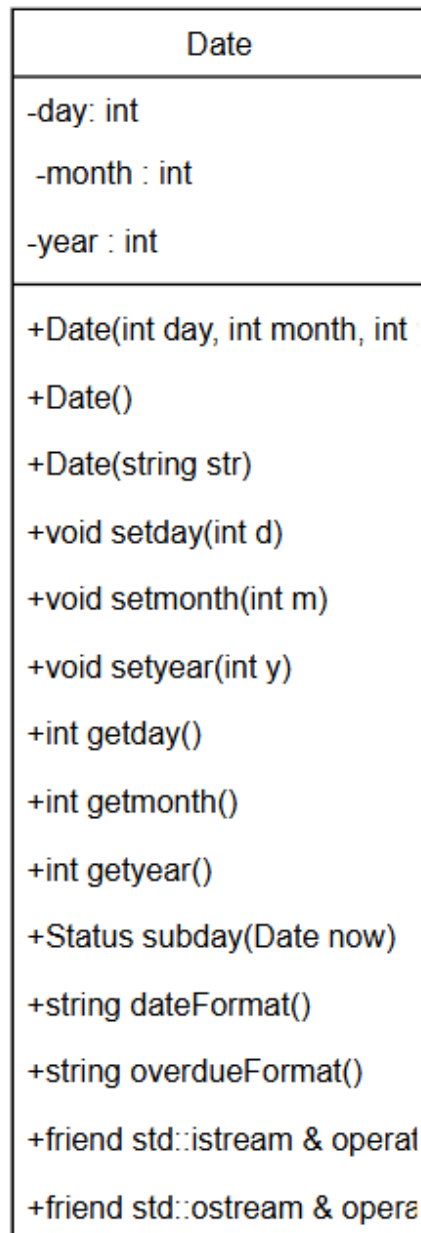
属性:

bookno: string - 书号  
bookname: string - 书名  
author: string - 作者  
publisher: string - 出版社  
totalnum: int - 总库存  
borrownum: int - 借出数量  
pubday: Date - 出版日期

操作:

setbookno(no: string): void - 设置书号  
setbookname(name: string): void - 设置书名  
setauthor(au: string): void - 设置作者  
setpublisher(pub: string): void - 设置出版社  
settotal(ti: int): void - 设置总库存  
setborrow(bi: int): void - 设置借出数量  
setday(t: Date): void - 设置日期  
getbookno(): string - 获取书号  
getbookname(): string - 获取书名  
getauthor(): string - 获取作者  
getpublisher(): string - 获取出版社  
gettotalnum(): int - 获取总库存  
getborrownum(): int - 获取借出数量  
getpubday(): Date - 获取出版日期  
operator==(b: Book): bool - 比较操作符  
operator>>(is: istream, b: Book): istream - 流输入操作  
operator<<(os: ostream, b: Book): ostream - 流输出操作

### 1.1.3 Date 类



图片 5 Date 类图

属性:

year: int - 年

month: int - 月

day: int - 日

操作:

Date() - 默认构造函数

Date(year: int, month: int, day: int) - 参数化构造函数

~Date() - 析构函数

setYear(year: int): void - 设置年

setMonth(month: int): void - 设置月

setDay(day: int): void - 设置日

getYear(): int - 获取年

getMonth(): int - 获取月

getDay(): int - 获取日

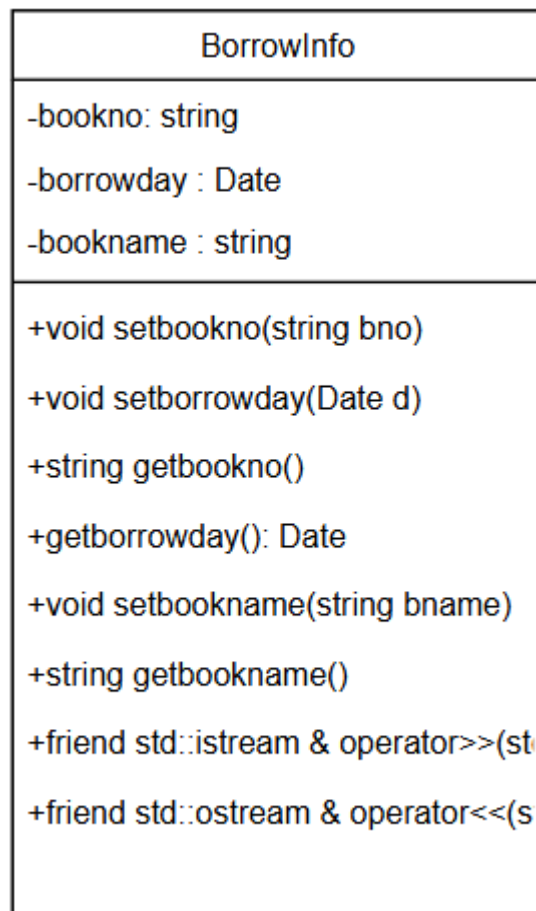
operator()(year: int, month: int, day: int): Date& - 函数调用操作符

Format(): string - 格式化日期

operator>>(is: istream, date: Date): istream - 流输入操作

operator<<(os: ostream, date: Date): ostream - 流输出操作

#### 1.1.4 BorrowInfo 类



图片 6 BookInfo 类图

属性:

bookno: string - 书号

borrowday: Date - 借阅日期

bookname: string - 书名

操作:

setbookno(bno: string): void - 设置书号

setborrowday(d: Date): void - 设置借阅日期

setbookname(bname: string): void - 设置书名

getbookno(): string - 获取书号

getborrowday(): Date - 获取借阅日期

getbookname(): string - 获取书名

operator>>(is: istream, bi: BorrowInfo): istream - 流输入操作

operator<<(os: ostream, bi: BorrowInfo): ostream - 流输出操作

### 1.1.5 StudentSet 类

StudentSet
Inherits from: vector<Student>
<div>+ ListInsert_S(e: Student): Status</div> <div>+ ListLength_S(): int</div> <div>+ ListDelete_S(i: int, e: Student&amp;): Status</div> <div>+ LocateAllElem_S(e: Student, compare: Status)</div> <div>+ GetElem_S(i: int, e: Student&amp;): Status</div> <div>+ SetElem_S(i: int, e: Student): Status</div> <div>+ OpenStuList(): Status</div> <div>+ stu_manage_output(pos: int = 0): Status</div> <div>+ Append_stu(filename: string): void</div> <div>+ stu_manage_append(): void</div> <div>+ stu_manage_delete(): void</div> <div>+ stu_manage_update(): void</div> <div>+ stu_search_char(info: string, fun: Status(Student))</div> <div>+ stu_manage_search(): void</div> <div>+ writeToFile(filename: const std::string&amp;): void</div>

图片 7 StudentSet 类图

继承： 从 vector<Student> 继承

操作：

ListInsert(s: Student): Status - 插入学生到列表

ListDelete(s: Student): Status - 从列表删除学生

ListLength(): int - 返回列表长度

LocateElem(s: Student, compare: Status): Student - 定位元素

GetElem(index: int): Student - 获取指定位置的学生

SetElem(index: int, s: Student): Status - 设置指定位置的学生

OpenStuList(): Status - 打开学生列表

stu\_manage\_output(pos: int = 0): Status - 管理输出

Append\_stu(filename: string): void - 附加学生信息到文件

stu\_manage\_append(): void - 管理附加操作

stu\_manage\_delete(): void - 管理删除操作

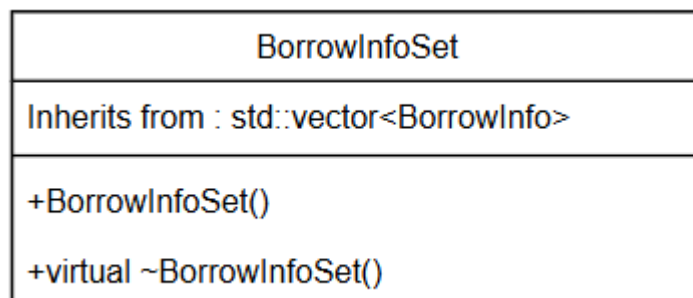
stu\_manage\_update(): void - 管理更新操作

stu\_search(char: string, fun: Status): void - 搜索学生

stu\_manage\_search(): void - 管理搜索操作

writeToFile(filename: string): void - 写入到文件

### 1.1.6 BorrowInfoSet 类



图片 8 BorrowInfoSet 类图

继承： 从 vector<BorrowInfo> 继承

操作：

BorrowInfoSet() - 构造函数

~BorrowInfoSet() - 析构函数

### 1.1.7 BookSet 类

BookSet
Inherits from : std::vector<Book>
<div><div>+BookSet()</div><div>+virtual ~BookSet()</div><div>+void returnBook(string bookNo)</div><div>+void borrowBook(string bookNo)</div><div>+int ListLength_B()</div><div>+Status ListInsert_B(Book e)</div><div>+Status LocateAllElem_B(Book e, Status(*co</div><div>+Status GetElem_B(int i, Book &amp; e)</div><div>+Status SetElem_B(int i, Book e)</div><div>+void OpenBookList()</div><div>+void Output_Book(int pos = 0)</div><div>+void Append_book(string filename)</div><div>+void book_manage_append()</div><div>+void book_search_char(string info, int(Book:</div><div>+void book_search_have()</div><div>+void book_search_date(int year, int month)</div><div>+void book_manage_search()</div><div>+void writeToFile(const std::string &amp; filename)</div></div>

图片 9 BookSet 类图

继承： 从 vector<Book> 继承

操作：

returnBook(bookNo: string): void - 归还书籍



`borrowBook(bookNo: string): void` - 借出书籍

`ListLength_B(): int` - 获取列表长度

`ListInsert_B(book: Book): Status` - 插入书籍

`LocateElem_B(book: Book, compare: Status): Book` - 定位书籍

`GetElem_B(index: int, book: Book&): Status` - 获取书籍

`SetElem_B(index: int, book: Book): Status` - 设置书籍

`OpenBookList(): void` - 打开书籍列表

`Output_Book(pos: int = 0): void` - 输出书籍信息

`Append_book(filename: string): void` - 附加书籍信息到文件

`book_manage_append(): void` - 管理附加操作

`book_search_char(info: string, into: Book): void` - 搜索书籍

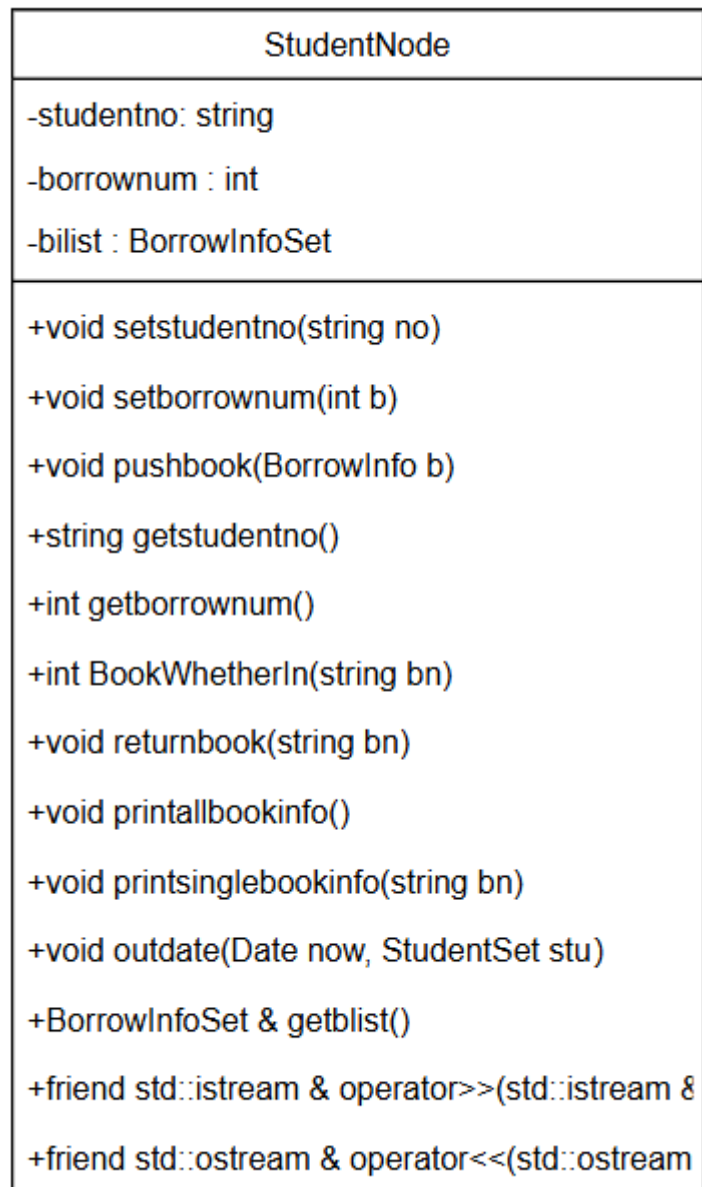
`book_search_have(): void` - 检查书籍存储

`book_search_date(year: int, month: int): void` - 按日期搜索书籍

`book_manage_search(): void` - 管理搜索操作

`writeToFile(filename: string): void` - 写入到文件

### 1.1.8 StudentNode 类



图片 10 StudentNode 类图

属性：

studentno: string - 学号

borrownum: int - 借书数量

blist: BorrowInfoSet - 借书列表

操作：

setstudentno(no: string): void - 设置学号

setborrownum(b: int): void - 设置借书数量

pushbook(b: BorrowInfo): void - 添加借书记录

getstudentno(): string - 获取学号

getborrownum(): int - 获取借书数量

getblist(): BorrowInfoSet& - 获取借书列表

BookWhetherIn(bn: string): int - 检查书籍是否在借书列表中

returnbook(bn: string): void - 归还书籍

printallbookinfo(): void - 打印所有借书信息

printsinglebookinfo(bn: string): void - 打印单个借书信息

outdate(now: Date, stu: StudentSet): void - 检查过期书籍

### 1.1.9 SBList 类

SBList
<div><div>-studentnum: int</div><div>-borrownum : int</div><div>-sblist : StudentNodeSet</div></div>
<div><div>+Status begin_borrow()</div><div>+Status OpenBorrowList()</div><div>+Status WriteToFile()</div><div>+int StuWhetherIn(string no)</div><div>+void pushelem(StudentNode sn)</div><div>+void setstudentnum(int s)</div><div>+void setborrownum(int b)</div><div>+int getstudentnum()</div><div>+int getborrownum()</div><div>+int getpersonalnum(string no)</div><div>+int BookWhetherIn(string sn, string bn)</div><div>+void addanewrelation(string sn, BorrowInfo bi)</div><div>+void returnbook(string sn, string bn)</div><div>+void printbookinfo(string sn)</div><div>+void printsinglebookinfo(string bn)</div><div>+void outdate(Date now, StudentSet stu)</div><div>+StudentNodeSet getSBList()</div><div>+bool bookLimit(string sn)</div></div>

图片 11 SBList 类图

属性:

studentnum: int - 学生数量

borrownum: int - 总借书数量

sblist: StudentNodeSet - 学生节点集合

操作:

begin\_borrow(): Status - 开始借书流程

OpenBorrowList(): Status - 打开借书列表

WriteToFile(): Status - 写入文件

StuWhetherIn(no: string): int - 检查学生是否在列表中

pushElem(sn: StudentNode): void - 添加学生节点

setstudentnum(s: int): void - 设置学生数量

setborrownum(b: int): void - 设置借书数量

getstudentnum(): int - 获取学生数量

getborrownum(): int - 获取借书数量

getpersonalnum(no: string): int - 获取个人借书数量

BookWhetherIn(sn: string, bn: string): int - 检查书籍是否已借出

addanewrelation(sn: string, bi: BorrowInfo): void - 添加新的借书关系

returnbook(sn: string, bn: string): void - 归还书籍

printbookinfo(sn: string): void - 打印书籍信息

printsinglebookinfo(bn: string): void - 打印单本书籍信息

outdate(now: Date, stu: StudentSet): void - 更新逾期书籍信息

getSBList(): StudentNodeSet& - 获取学生借书节点列表

bookLimit(sn: string): bool - 检查借书限制

### 1.1.10 Relationship 类

Relationship
-stuno: string -bookno : string -stumobile : string
+void setstuno(string sno) +void setbookno(string bno) +void setstumobile(string mobi) +string getstuno() +string getbookno() +string getstumobile() +friend std::istream & operator>>(st +friend std::ostream & operator<<(s

图片 12 Relationship 类图

属性:

stuno: string - 学号

bookno: string - 书号

stumobile: string - 学生手机号

操作:

setstuno(stno: string): void - 设置学号

setbookno(bno: string): void - 设置书号

setstumobile(mobi: string): void - 设置手机号

getstuno(): string - 获取学号

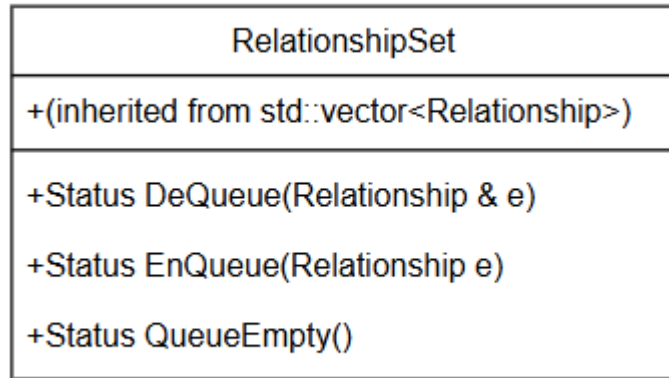
getbookno(): string - 获取书号

getstumobile(): string - 获取手机号

operator>>(is: istream, r: Relationship): istream - 流输入操作

operator<<(os: ostream, r: Relationship): ostream - 流输出操作

#### 1.1.11 RelationshipSet 类



图片 13 RelationshipSet 类图

**继承：** 从 vector<Relationship> 继承

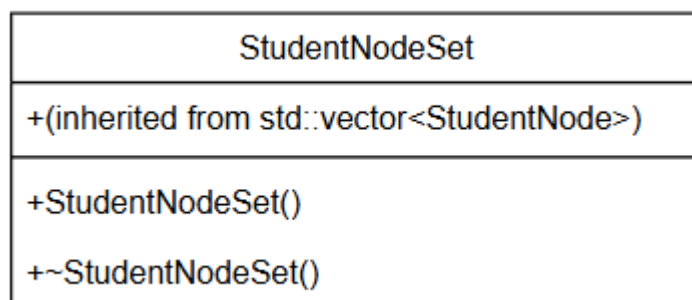
**操作：**

DeQueue(r: Relationship&): Status - 出队关系

EnQueue(r: Relationship): Status - 入队关系

QueueEmpty(): Status - 检查队列是否为空

#### 1.1.12 StudentNodeSet 类



图片 14 StudentNodeSet 类图

**继承：** 从 vector<StudentNode> 继承

**操作：**

StudentNodeSet() - 构造函数

~StudentNodeSet() - 析构函数

AddStudentNode(sn: StudentNode): Status - 添加学生节点

RemoveStudentNode(sn: StudentNode): Status - 删除学生节点

FindStudentNode(no: string): StudentNode& - 查找学生节点

UpdateStudentNode(sn: StudentNode): Status - 更新学生节点

GetSize(): int - 获取集合大小

## 1.2 提供交互界面的类

### 1.2.1 简介

（一）BookMS 界面类是软件的登录页面，作为软件的初始入口，提供用户登录验证，是用户进入图书馆管理系统的第一步。

（二）提供管理和交互图书馆管理系统的管理员界面，使管理员能够有效地管理书籍和学生信息的界面类 **Admin\_InfoDialog**，支持对书籍集合和学生集合的增加、删除、编辑和查询操作，以及对系统设置的管理。

（三）为学生交互提供对话框界面的界面类 **Student\_InfoDialog**，使学生能够查看个人信息、借阅记录、推荐书籍及归还书籍。此界面类方便学生管理与自己相关的图书馆资源和服务。

（四）用于系统中管理借阅操作的界面类小部件 **cell\_borrow**，提供了一个用户友好的界面让用户能够方便地借阅图书馆中的书籍。

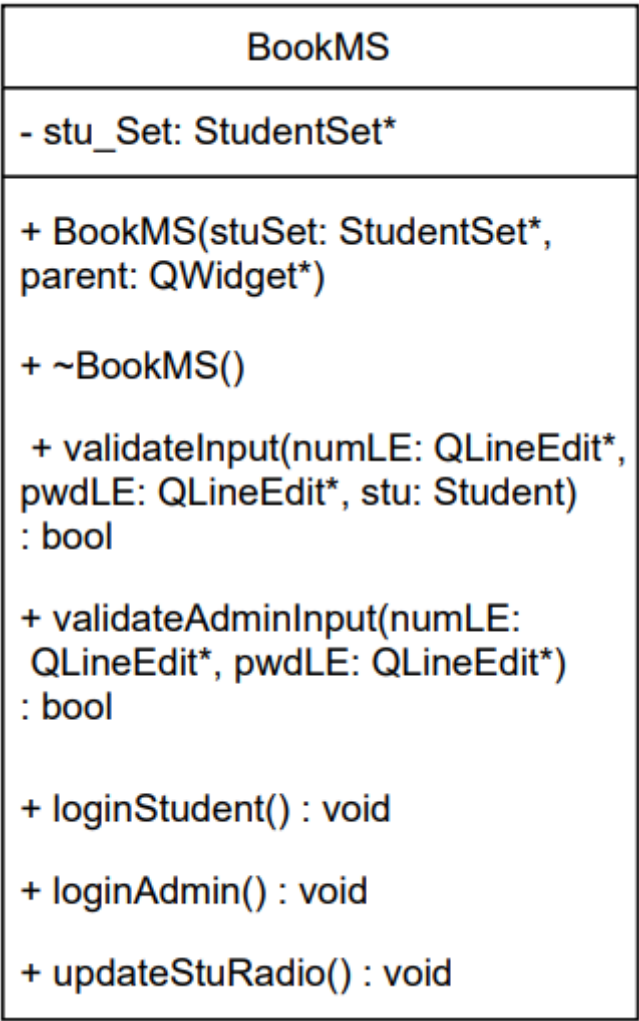
（五）用于显示和管理用户个人信息的界面类小部件 **cell\_personInfo**，提供了一个界面让用户查看和编辑自己的详细资料。

（六）用于为用户推荐书籍的界面类小部件 **cell\_recommend**，根据系统当前的借阅记录设置推荐书籍，增强用户体验。

（七）用于管理图书馆系统中的归还操作的界面类小部件 **cell\_return**，提供用户友好的界面以归还借阅的书籍。



1.2.2 BookMS（登录页面和主控制界面）



图片 15 BookMS 类图

属性：

- stu\_Set: 存储所有学生信息的集合。
- book\_Set: 存储所有书籍信息的集合。
- borrow\_Set: 存储所有借阅信息的集合。

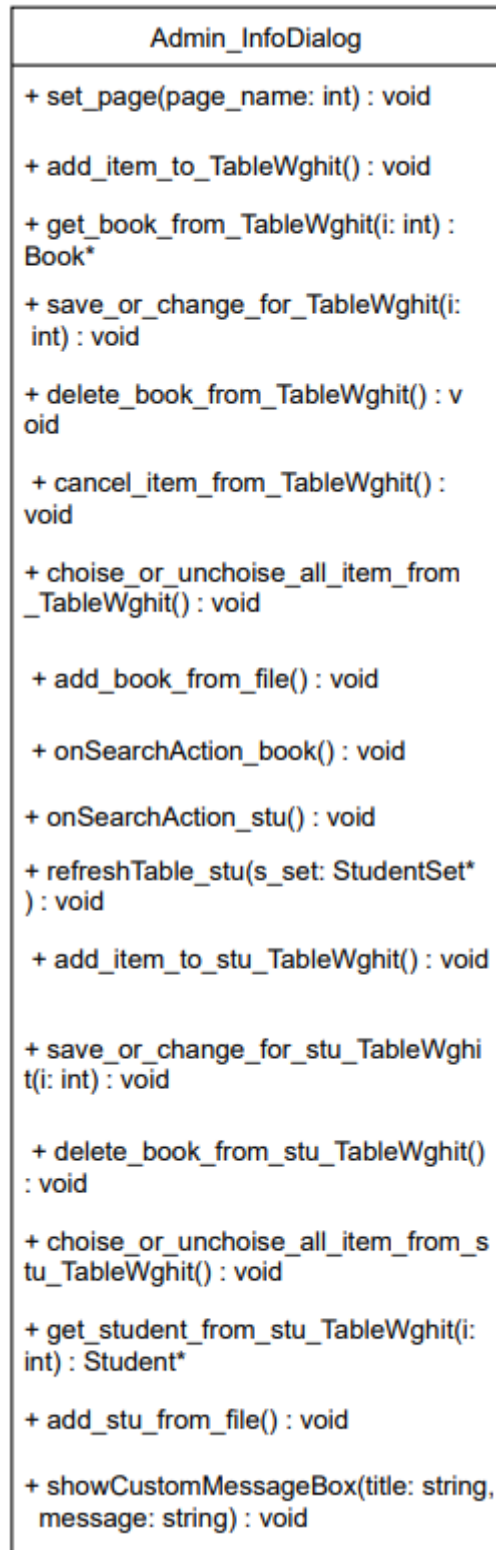
操作：

- loginStudent(): 学生登录验证。
- loginAdmin(): 管理员登录验证。
- validateInput(): 验证输入的用户名和密码。
- updateStuRadio(): 更新登录状态。

### 1.2.3 Admin\_InfoDialog（管理员信息管理界面）

Admin_InfoDialog
<ul style="list-style-type: none"> <li>- ui: Ui::Admin_InfoDialogClass</li> <li>- b_set: BookSet*</li> <li>- s_set: StudentSet*</li> <li>- search_BookSet: BookSet*</li> <li>- search_StuSet: StudentSet*</li> <li>- current_BookSet: BookSet*</li> <li>- current_StuSet: StudentSet*</li> <li>- btn: std::vector&lt;QPushButton*&gt;</li> <li>- mark: std::vector&lt;QCheckBox*&gt;</li> <li>- stu_btn: std::vector&lt;QPushButton*&gt;</li> <li>- stu_mark: std::vector&lt;QCheckBox*&gt;</li> </ul>
<ul style="list-style-type: none"> <li>+ Admin_InfoDialog(parent: QWidget)</li> <li>+ ~Admin_InfoDialog()</li> <li>+ set_TabelWghit() : void</li> <li>+ set_TabelWghit_stu() : void</li> <li>+ refreshTable(set: BookSet*) : void</li> <li>+ exchange_Set(str: string) : void</li> <li>+ find_book_and_return_idx(book: Book*) : int</li> <li>+ find_book_and_return_idx(book: Book*) : int</li> <li>+ find_stu_and_return_idx(stu: Student*) : int</li> <li>+ requireInt(str: QString) : bool</li> <li>+ isValidDateFormat(Qstr: QString) : bool</li> <li>+ getSafeString(item: QTableWidgetItem*) : string</li> <li>+ isNumber(str: string) : bool</li> <li>+ isBooknoUnique(i: int, item: QTableWidgetItem*) : bool</li> <li>+ isStunoUnique(i: int, item: QTableWidgetItem*) : bool</li> </ul>

图片 16 Admin\_InfoDialog 类图 1



图片 17 Admin\_InfoDialog 类图 2

属性:

ui: 界面相关的所有用户界面组件。

b\_set, s\_set: 分别指向当前活动的书籍和学生集合。

操作:

set\_page(): 设置当前显示的页面。

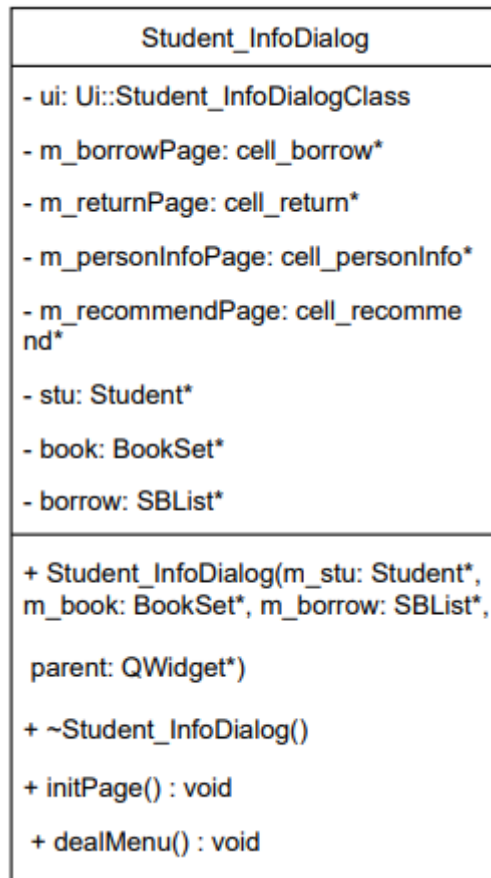
refreshTable(), refreshTable\_stu(): 刷新书籍和学生信息的表格。

add\_item\_to\_TableWghit(): 向表格中添加新项目。

delete\_book\_from\_TableWghit(): 从表格中删除书籍。

save\_or\_change\_for\_TableWghit(): 保存或修改表格中的信息。

#### 1.2.4 Student\_InfoDialog (学生信息对话框)



图片 18 Student\_InfoDialog 类图

属性:

stu: 当前登录的学生对象。

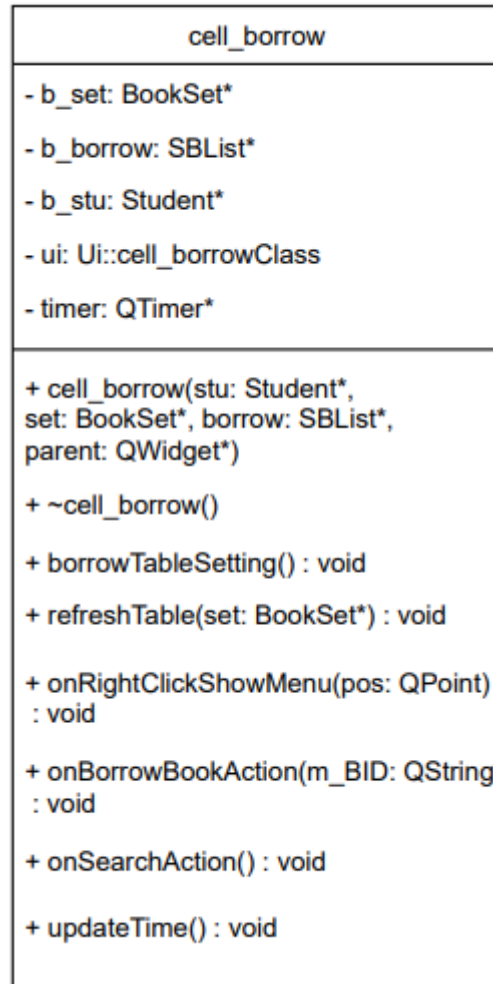
book\_Set, borrow\_Set: 存储书籍和借阅信息的集合。

操作:

initPage(): 初始化页面布局和数据。

dealMenu(): 处理菜单选项操作。

#### 1.2.5 cell\_borrow（借阅操作小部件）



图片 19 cell\_borrow 类图

属性：

`b_set`: 指向书籍集合。

`b_borrow`: 指向借阅信息集合。

`b_stu`: 当前操作的学生对象。

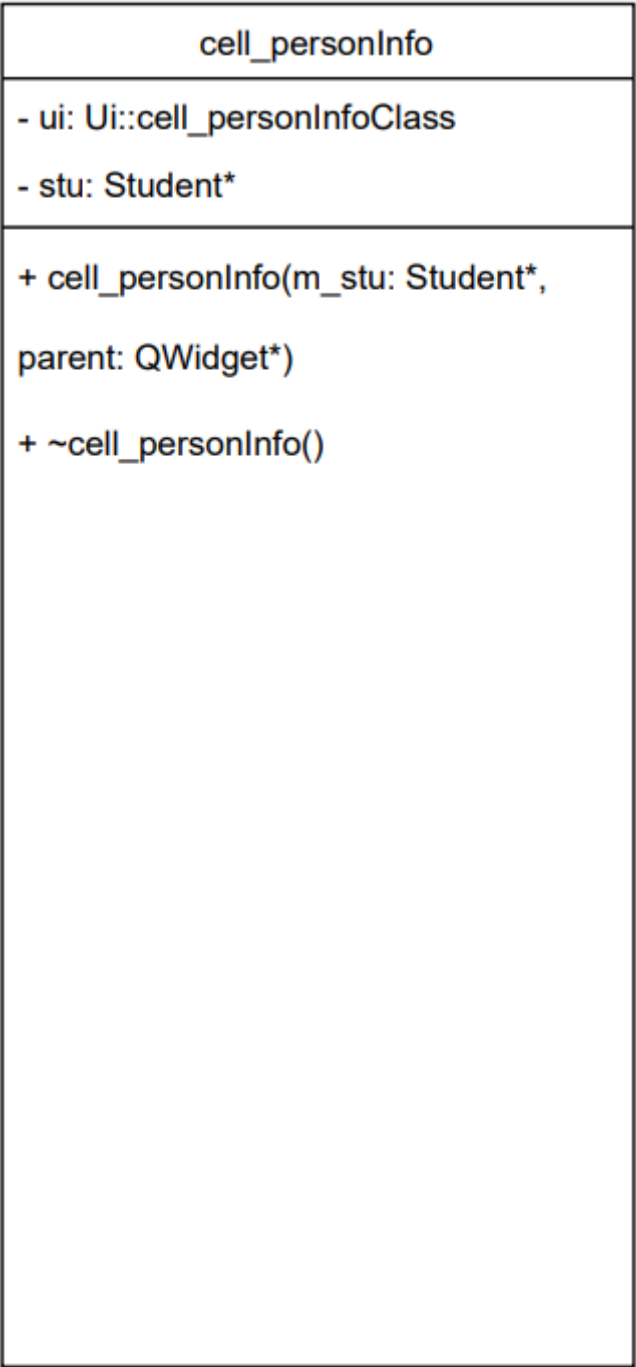
操作：

`borrowTableSetting()`: 设置借阅表格的参数。

`refreshTable()`: 刷新显示的借阅信息。

`onBorrowBookAction()`: 处理借书操作。

1.2.6 cell\_personInfo（个人信息显示小部件）



图片 20 cell\_personInfo 类图

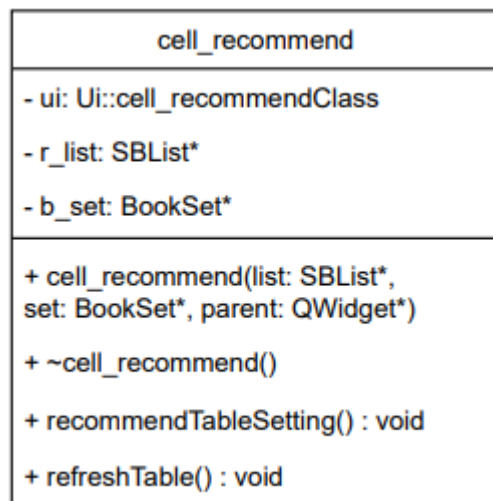
属性：

`stu`: 显示的学生对象。

操作：

无特殊操作，主要用于展示个人信息。

### 1.2.7 cell\_recommend（推荐书籍小部件）



图片 21 cell\_recommend 类图

属性：

r\_list: 推荐书籍的列表。

b\_set: 存储书籍信息的集合。

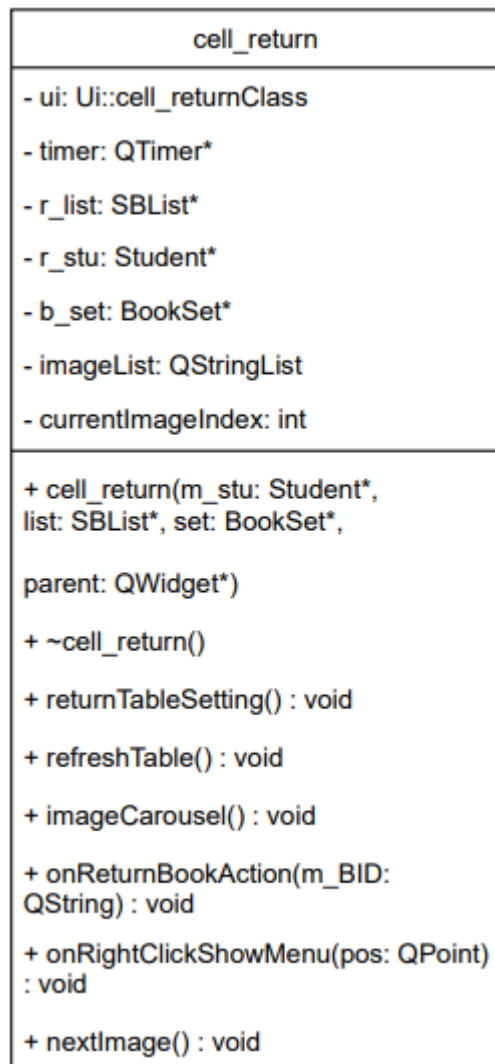
操作：

recommendTableSetting(): 设置推荐表格的显示参数。

refreshTable(): 刷新推荐书籍的显示信息。



### 1.2.8 cell\_return（归还书籍操作小部件）



图片 22 cell\_return 类图

属性：

`r_list`: 归还操作的借阅列表。

`r_stu`: 正在操作归还的学生对象。

`b_set`: 存储书籍信息的集合。

操作：

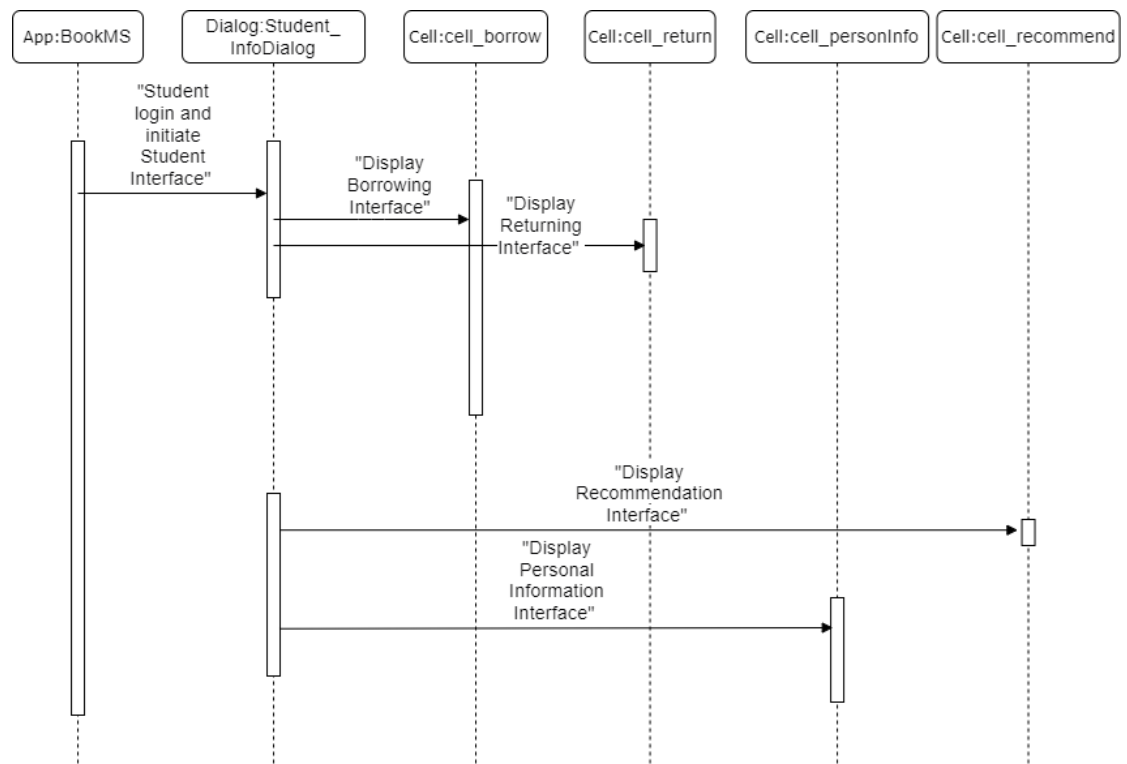
`returnTableSetting()`: 设置归还书籍表格的显示参数。

`refreshTable()`: 刷新归还信息。

`onReturnBookAction()`: 处理归还书籍的操作。

## 2 类的动态设计（顺序图）

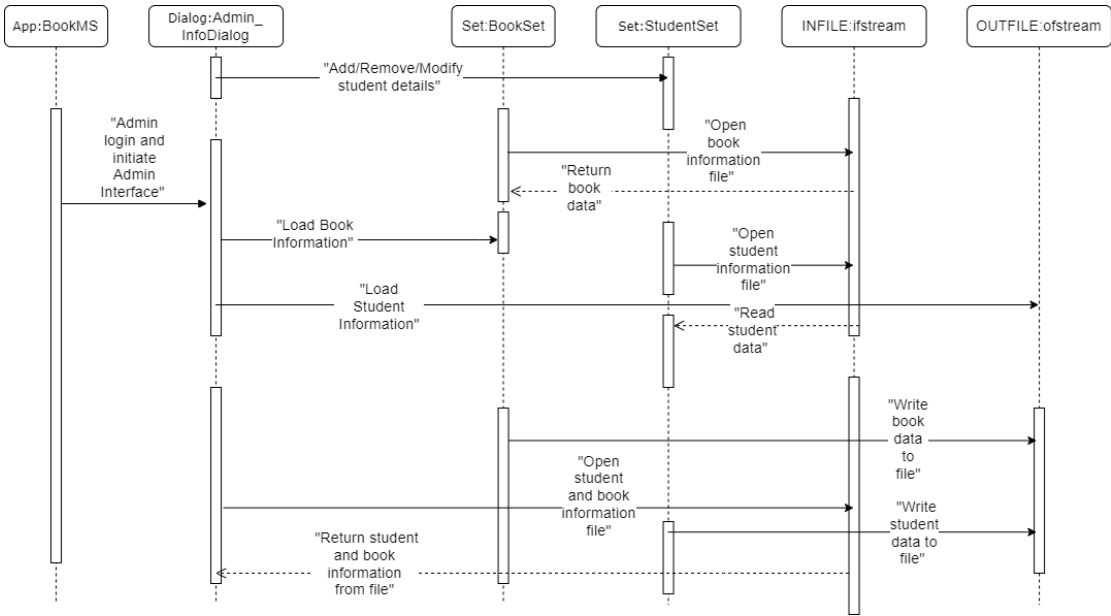
### 2.1 Student 界面交互操作



图片 23 Student 界面顺序图

参与的对象：BookMS、Student\_InfoDialog、cell\_borrow、cell\_return、cell\_personInfo、和 cell\_recommend。

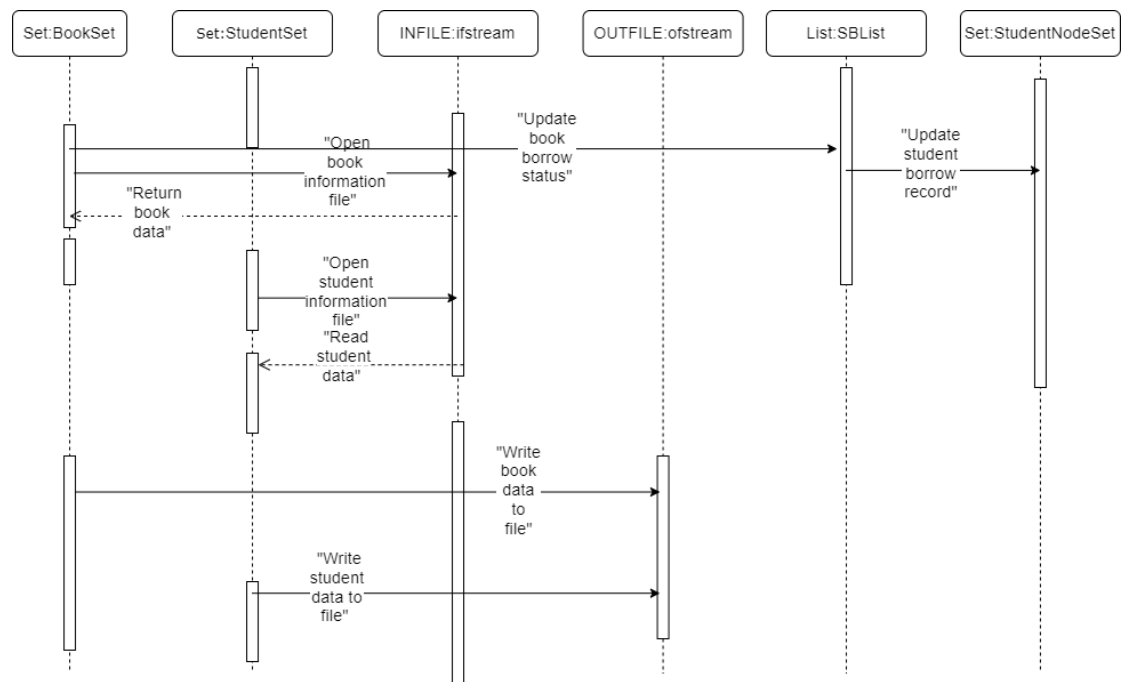
2.2 Admin 界面交互操作（包括对书籍和学生信息的增添，删除，修改，查询）



图片 24 Admin 界面顺序图

参与的对象: BookMS、Admin\_InfoDialog、BookSet、StudentSet、INFILE: ifstream、OUTFILE: ofstream。

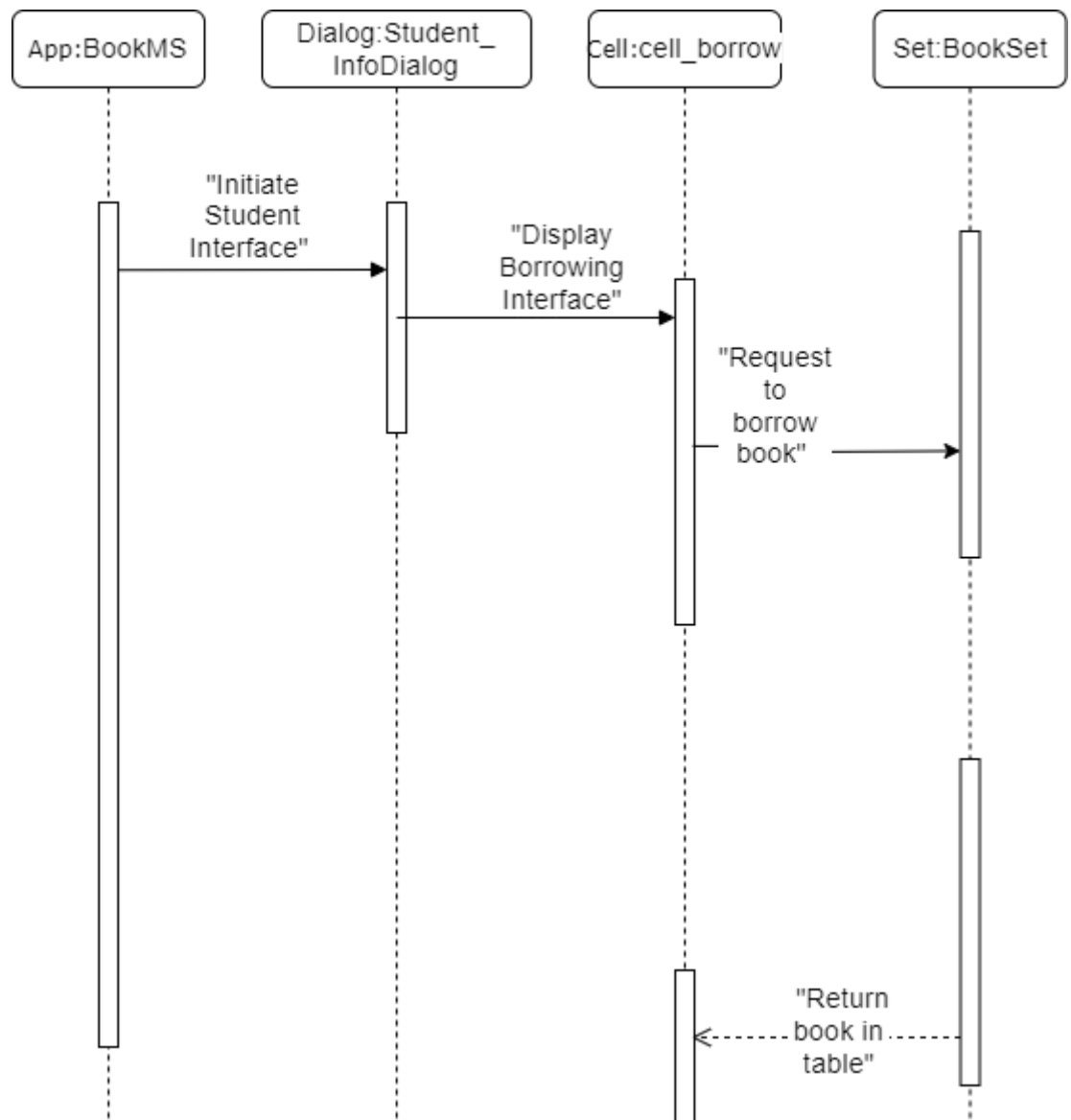
## 2.3 学生对书籍进行借阅



图片 25 学生借书界面顺序图

参与的对象：BookSet、StudentSet、SBList、StudentNodeSet、INFILE: ifstream、OUTFILE: ofstream。

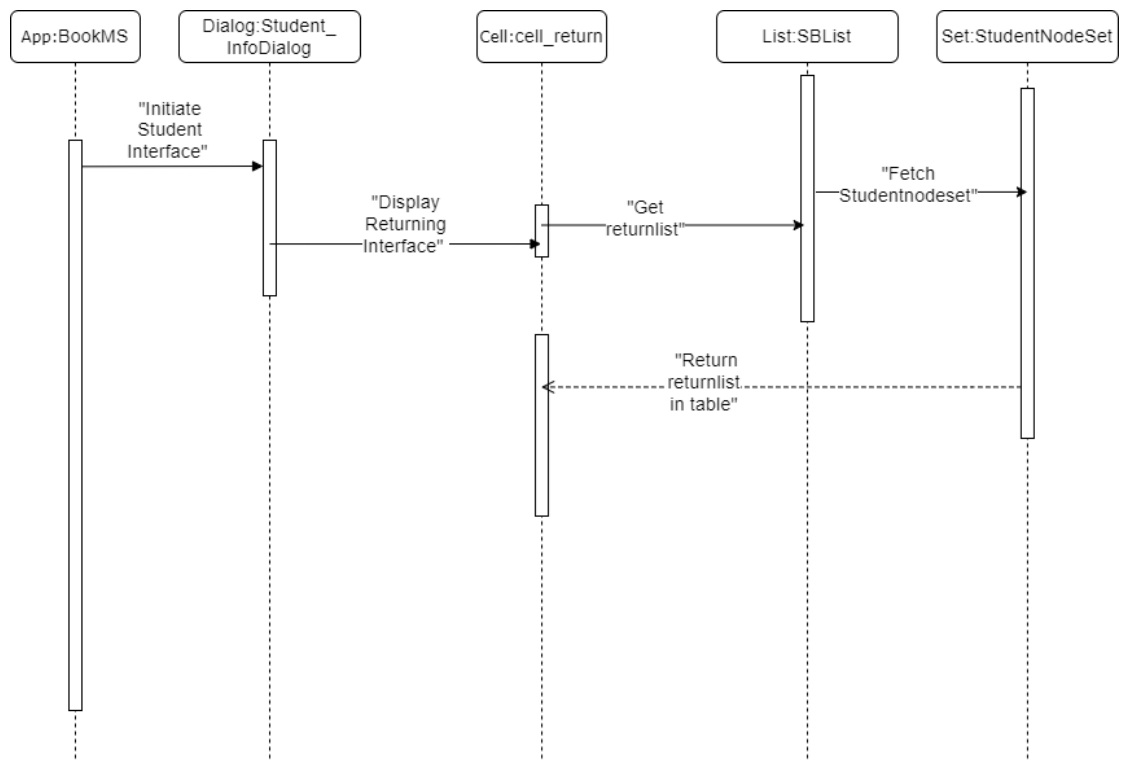
## 2.4 学生借书界面展示



图片 26 学生借书界面顺序图

参与的对象：BookMS、Student\_InfoDialog、cell\_borrow、BookSet。

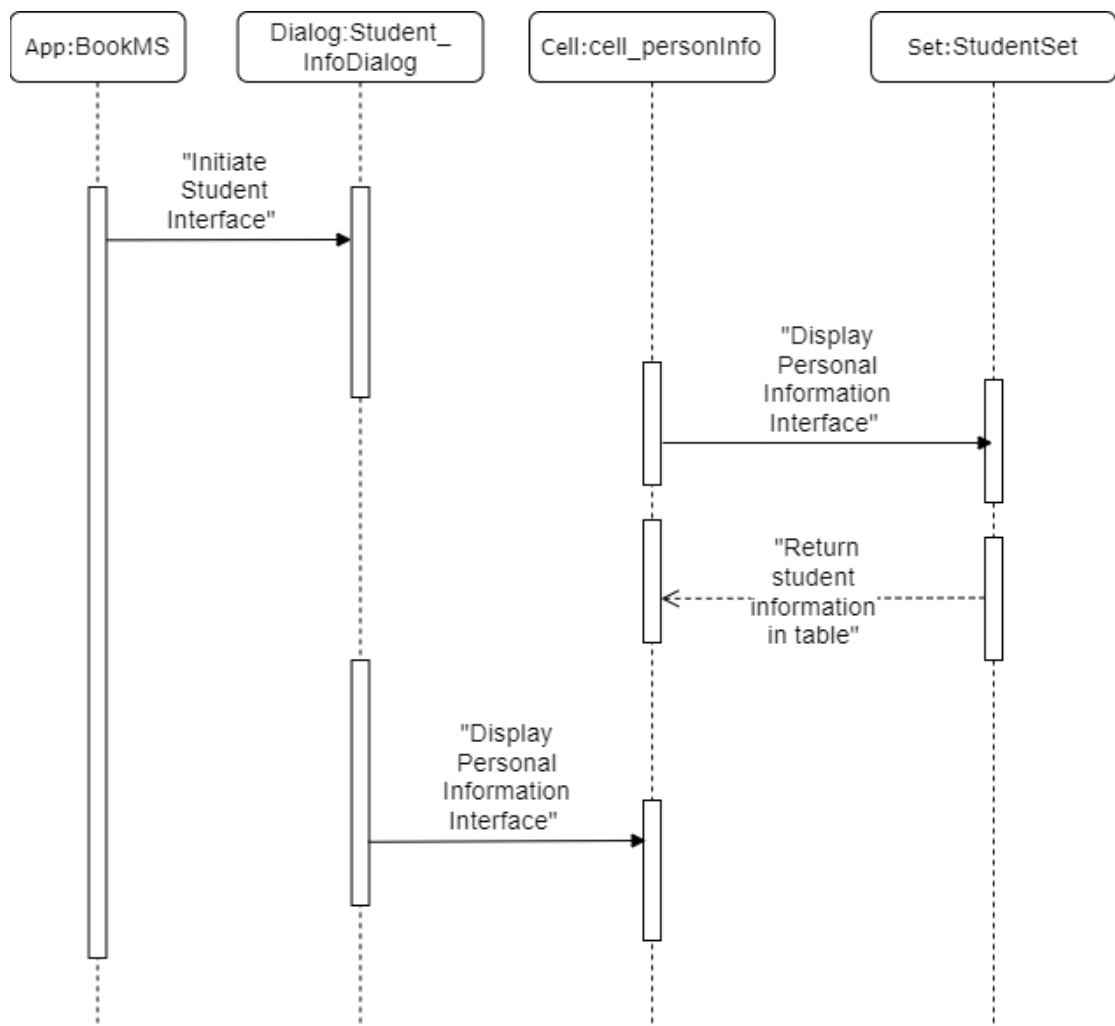
## 2.5 学生还书界面展示



图片 27 学生还书界面顺序图

参与的对象：BookMS、Student\_InfoDialog、cell\_return、SBLList、StudentNodeSet。

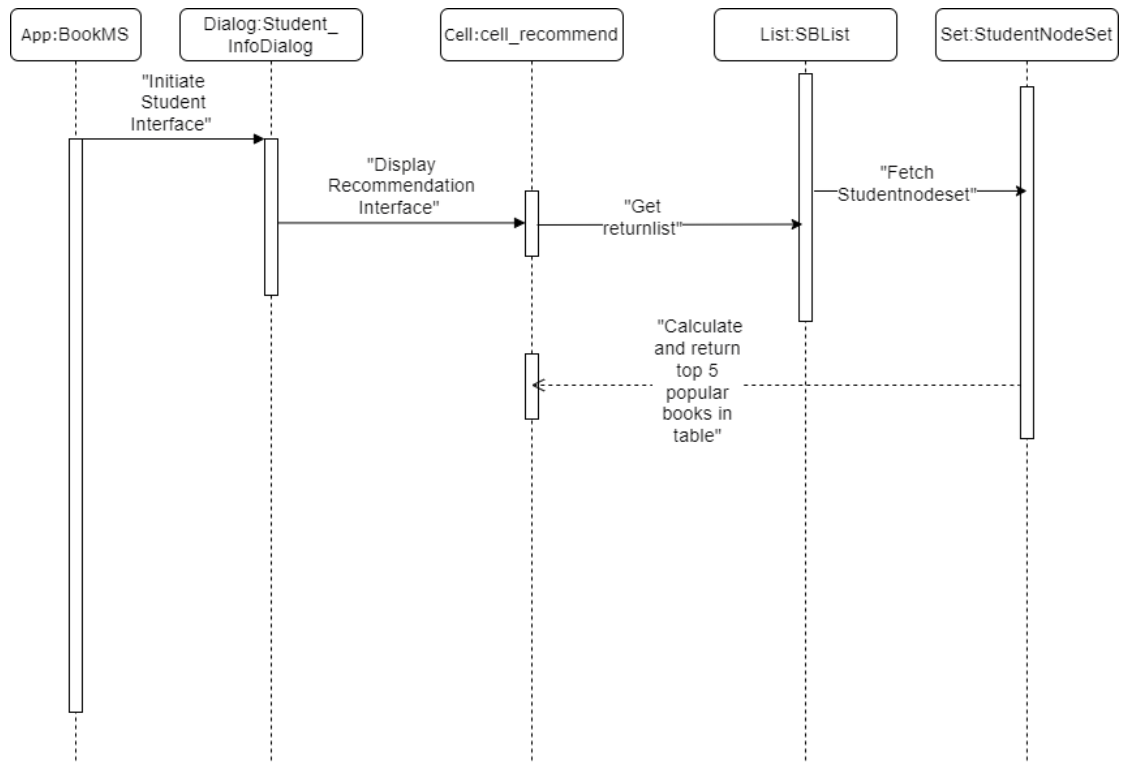
## 2.6 学生个人信息界面展示



图片 28 学生个人信息界面顺序图

参与的对象：BookMS、Student\_InfoDialog、cell\_personInfo、StudentSet。

## 2.7 图书推荐界面展示

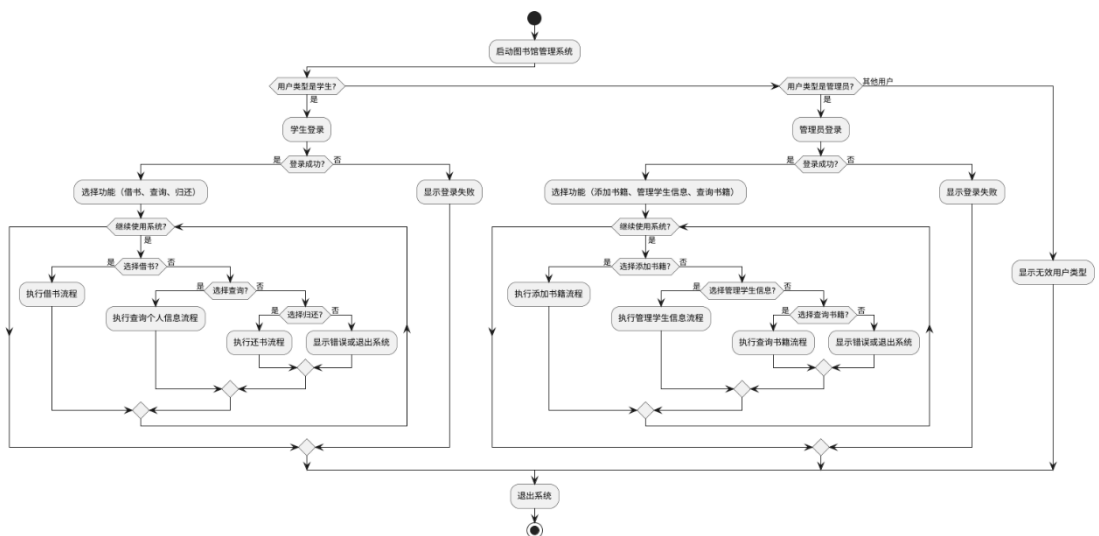


图片 29 图书推荐界面顺序图

参与的对象：BookMS、Student\_InfoDialog、cell\_recommend、SBList、StudentNodeSet。

### 3. 实现过程设计（活动图）

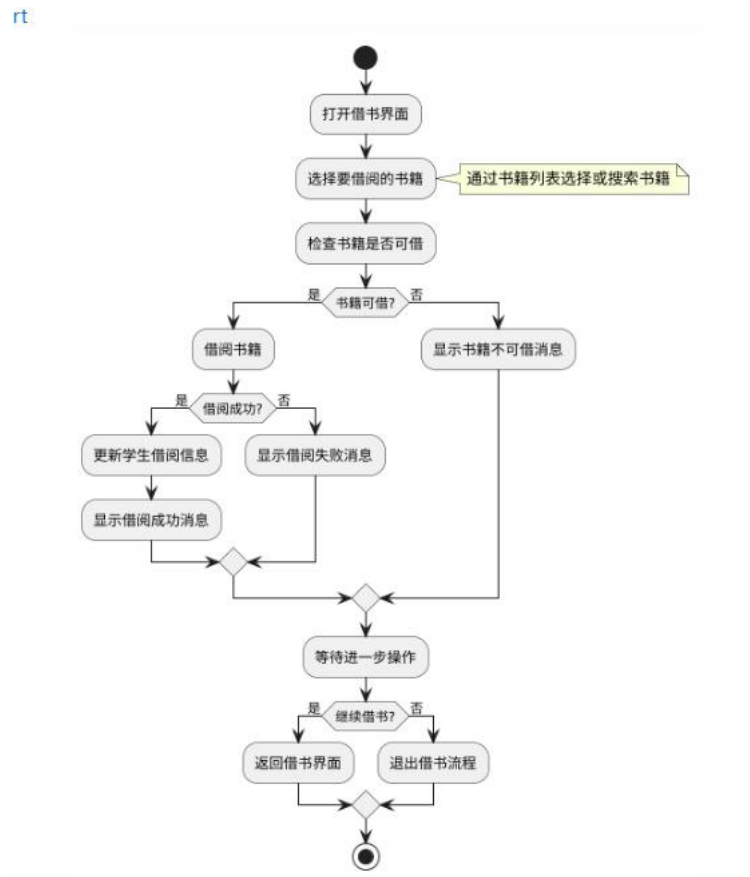
#### 3.1 总活动图



图片 30 总活动图

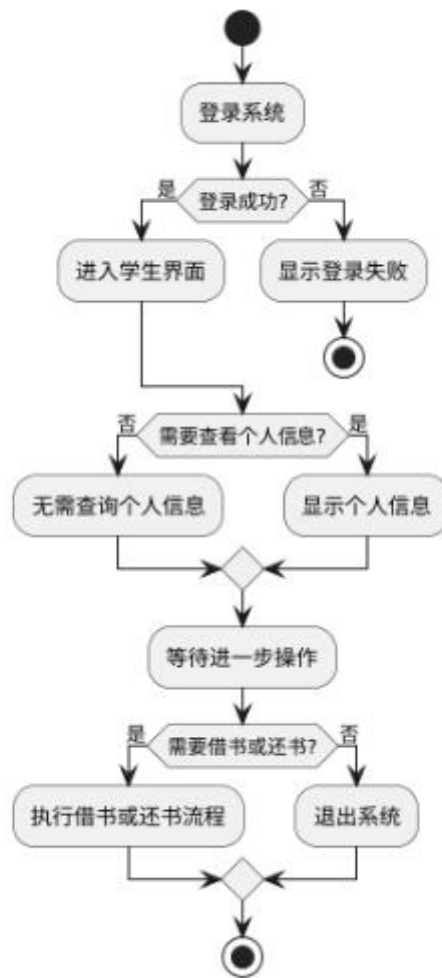


### 3.2 学生借书活动图



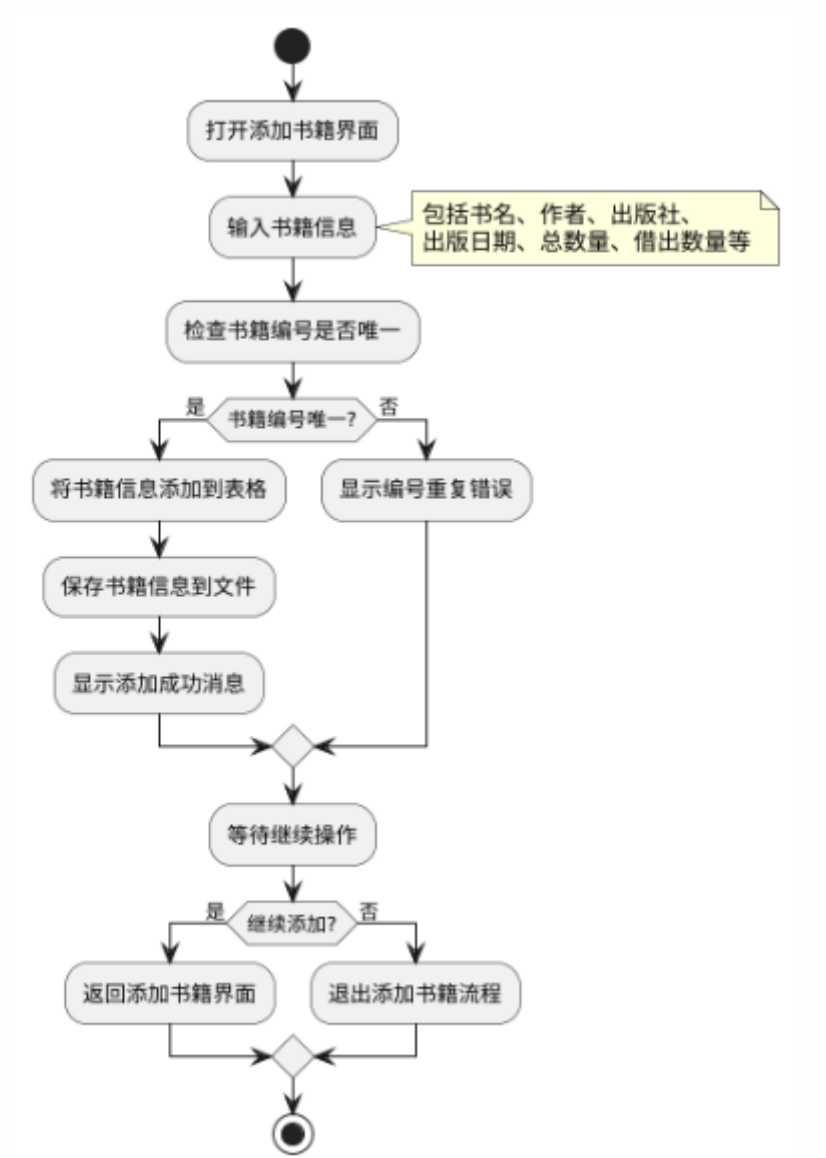
图片 31 学生借书活动图

### 3.3 学生个人信息查询活动图



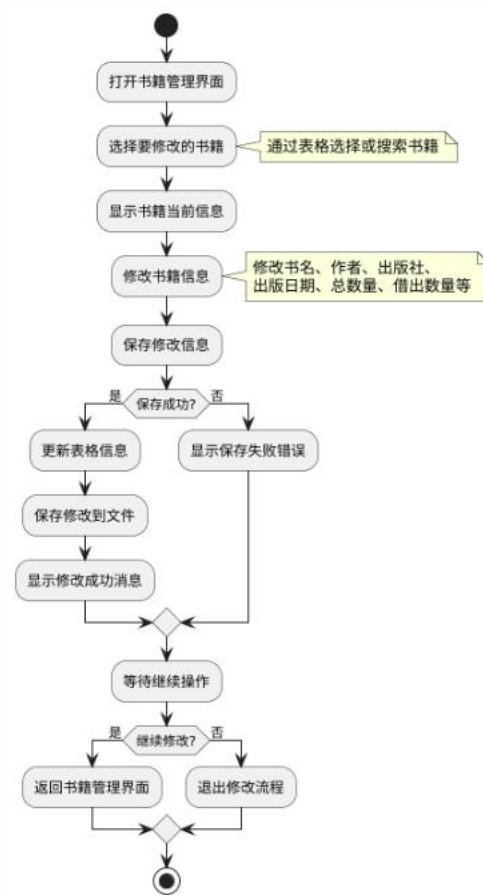
图片 32 学生个人信息查询活动图

### 3.4 管理员增加书籍



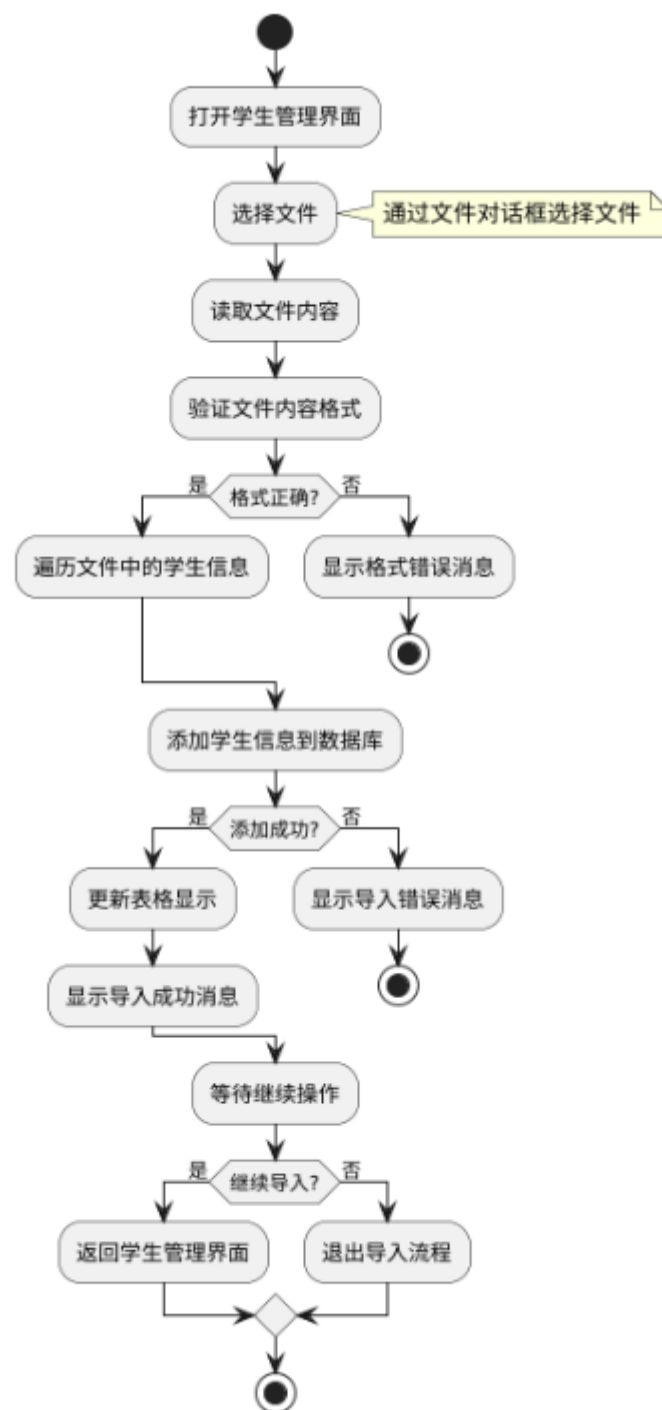
图片 33 管理员添加书籍活动图

### 3.5 管理员修改书籍



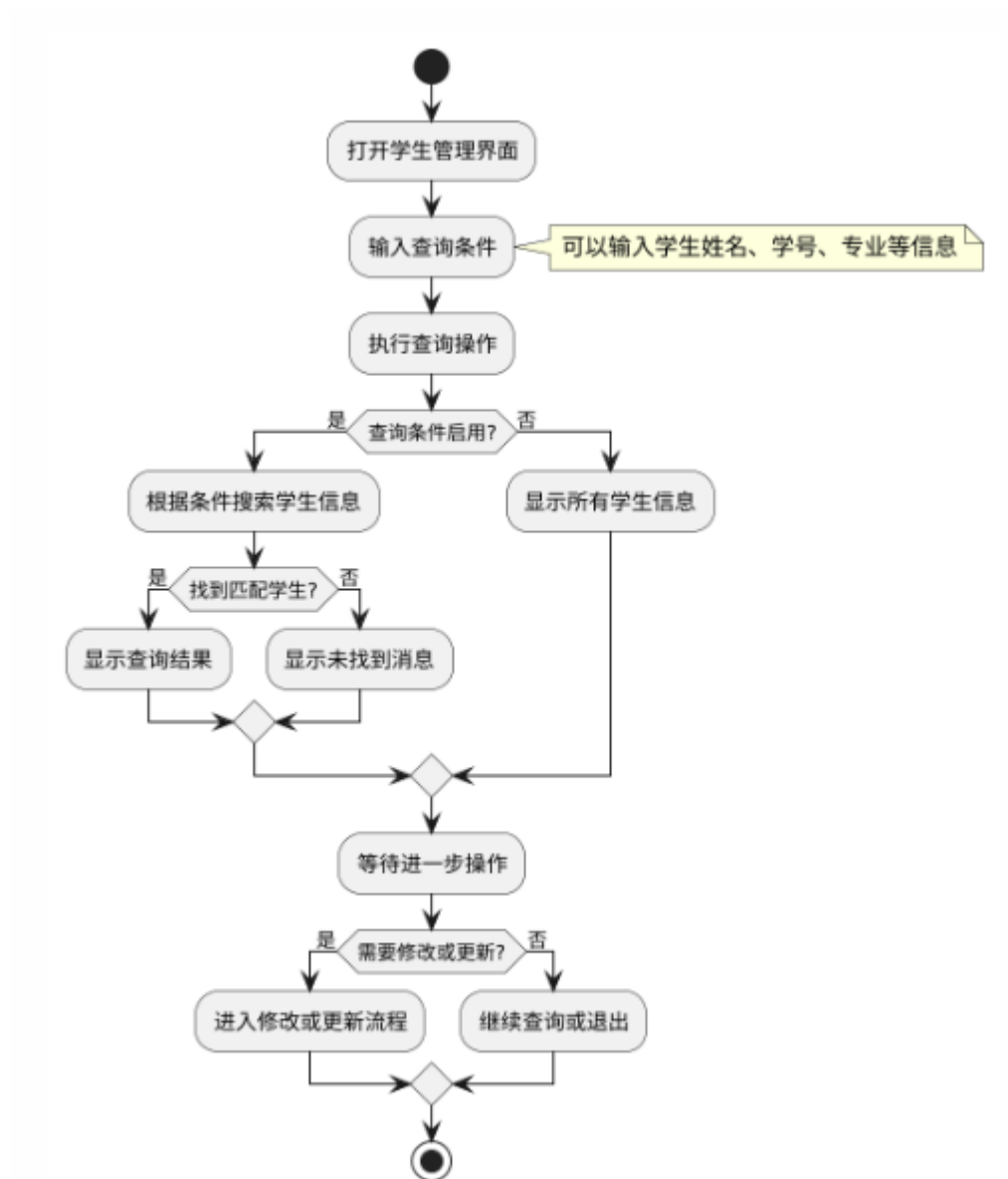
图片 34 管理员修改书籍

### 3.6 管理员批量增加学生



图片 35 管理员批量增加书籍

### 3.7 管理员查询学生信息



图片 36 管理员查询学生信息

### 三、系统详细设计（类的定义和主要操作的实现算法、功能实现算法和程序运行控制的实现算法）

#### 1. 类的详细设计

##### 1.1 Book 类

###### 1.1.1 类定义

```
class Book
{
private:
    string bookno;      //书号
    string bookname;    //姓名
    string author;      //作者
    string publisher;    //出版社
    int totalnum;        //馆藏图书量
    int borrownum;       //借出数量
    Date pubday;         //出版日期

public:
    // 设置书号
    void setbookno(string no);

    // 设置书名
    void setbookname(string name);

    // 设置作者名
    void setauthor(string au);

    // 设置出版社
    void setpublisher(string pub);

    // 设置馆藏总量
    void settotal(int i);
```

```

// 设置借出数量

    void setborrow(int i);

// 设置出版日期

    void setday(Date t);

// 获取书号

    string getbookno();

// 获取书名

    string getbookname();

// 获取作者名

    string getauthor();

// 获取出版社名称

    string getpublisher();

// 获取馆藏总量

    int gettotalnum();

// 获取借出数量

    int getborrownum();

// 获取出版日期

    Date& getpubday();

// 重载相等运算符，比较两本书是否相等

    bool operator == (Book b);

// 查找与指定书号匹配的书籍

    Status index_bookno(string info);

    // 查找与指定作者匹配的书籍

    Status index_author(string info)

        ;

    // 查找与指定书名匹配的书籍

    Status index_bookname(string info);

    // 查找与指定出版社匹配的书籍

    Status index_publisher(string info);

```



```

        friend std::istream& operator>>(std::istream& is, Book& boo);    //重载输入输出运算符

        friend std::ostream& operator<<(std::ostream& os, Book& boo);    // 重载输出运算符，用于向输出流写入书籍信息

        friend bool operator ==(Book& book1, Book& book2); // 重载全局相等运算符，用于比较两本书是否完全相同

        // 将书籍信息转化为字符串形式

        std::string toString();

};

```

## 1.1.2 主要算法

### ①toString

```
std::string Book::toString();
```

参数说明：无

返回说明：string 类对象。

**BEGIN**

获取书号 bookno 并添加到结果字符串 str。

获取书名 bookname 并添加到结果字符串 str。

获取作者 author 并添加到结果字符串 str。

获取出版社 publisher 并添加到结果字符串 str。

获取总库存量 totalnum，转换为字符串后添加到结果字符串 str。

获取借出数量 borrownum，转换为字符串后添加到结果字符串 str。

获取出版日期 pubday（年、月、日），转换为字符串后分别添加到结果字符串 str。

将以上各字段以空格分隔，组成最终的字符串。

```
return str
```

**END**

## 1.2 Student 类

### 1.2.1 类定义

```
class Student
{
private:
    string studentno;    //学号
    string studentname;  //姓名
    string studentmajor; //专业
    string studentclass; //班级
    string studentmobile; //手机

public:
    // 默认构造函数
    Student() {};

    // 拷贝构造函数，用另一个学生对象来初始化此对象
    Student(Student* stu);

    void setstuno(string no); //设置学号
    void setstuname(string name); //设置姓名
    void setstumajor(string major); //设置专业
    void setstuclass(string c); //设置班级
    void setstumobi(string mobi); //设置手机号

    string getstuno(); //获取学号
    string getstuname(); //获取姓名
    string getstumajor(); //获取专业
    string getstuclass(); //获取班级
    string getstumobi(); //获取手机号

    // 重载等于操作符，用于比较两个学生是否相等
```

```

    bool operator == (Student s);

    Status index_stuno(string info);

    Status index_stuname(string info);

    Status index_stumajor(string info);

    Status index_stuclass(string info);

    Status index_stumobile(string info);

    Status equal_stuno(Student b);      //比较学生学号是否相同

    Status equal_stuma_or_cl(Student b);    //比较学生的专业或班级是否相同

    friend std::istream& operator>>(std::istream& is, Student& stu);      // 重载输入运
算符，用于从输入流中读取学生信息

    friend std::ostream& operator<<(std::ostream& os, Student& stu);      // 重载输出
运算符，用于向输出流写入学生信息

    // 将学生对象的信息转换为字符串

    std::string toString();

};

```

## 1.2.2 主要算法

### ① toString

参数说明： 无

返回说明： 返回一个 `std::string` 类型的对象，该对象包含学生的学号、姓名、专业、班级和手机号码，各字段之间用空格分隔。

**BEGIN**

从学生对象中获取学号 `studentno` 并将其添加到结果字符串。

获取学生的姓名 `studentname` 并添加到结果字符串。

获取学生的专业 `studentmajor` 并添加到结果字符串。

获取学生的班级 `studentclass` 并添加到结果字符串。

获取学生的手机号码 `studentmobile` 并添加到结果字符串。

将以上各字段以空格分隔，组成最终的字符串。

return 最终构成的字符串。

END

## ② equal\_stuno

参数说明： b: 传入的 Student 对象，用于与当前对象的学号进行比较。

返回说明： 返回 Status 类型，如果两个学生的学号相同则返回 TRUE，否则返回 FALSE。

BEGIN

比较当前学生对象的学号 studentno 和传入学生对象 b 的学号 b.getstuno()。

如果两个学号相等，则返回 TRUE。

如果不相等，则返回 FALSE。

END

## ③ equal\_stuma\_or\_cl

参数说明： b: Student 类的对象，将与当前对象的专业或班级进行比较。

返回说明： 返回 Status 类型，如果当前学生的专业或班级与传入的学生 b 的专业或班级至少有一个相同，则返回 TRUE，否则返回 FALSE。

BEGIN

检查当前学生对象的专业 studentmajor 是否与传入学生对象 b 的专业 b.getstumajor() 相同，或者当前学生对象的班级 studentclass 是否与传入学生对象 b 的班级 b.getstuclass() 相同。

如果任一条件为真（即专业或班级至少有一个相同），则函数返回 TRUE。

如果两个条件都不满足（即专业和班级都不相同），则函数返回 FALSE。

END

## ④ index

参数说明：

s1: string 类型，作为搜索的主字符串。

s2: string 类型，是需要在 s1 中查找的子字符串。

`a: int` 类型，该参数在函数定义中未使用，可能是为将来的功能预留或需重新考虑其用途。

返回说明： 返回 `int` 类型，如果 `s2` 子字符串存在于 `s1` 主字符串中，则返回 `OK`（假设 `OK` 已定义为成功的状态码）。如果 `s2` 不包含在 `s1` 中，则返回 `ERROR`（假设 `ERROR` 已定义为错误的状态码）。

函数描述： 此函数使用 C++ 标准库中的 `std::string` 类的 `find` 方法来检查 `s2` 是否为 `s1` 的子字符串。`std::string::find` 方法在找到子字符串时返回其在主字符串中的位置索引，如果未找到，则返回 `std::string::npos`。

BEGIN

- 使用 `s1.find(s2)` 方法尝试在 `s1` 中查找子字符串 `s2`。
- 检查返回值：
  - 如果返回值不等于 `std::string::npos`（意味着找到了 `s2`），则返回 `OK`。
  - 如果返回值等于 `std::string::npos`（意味着没有找到 `s2`），则返回 `ERROR`。

END

## 1.3 BorrowInfo 类

### 1.3.1 类定义

```
class BorrowInfo    //借阅信息
{
private:
    string bookno;    //书号
    Date borrowday;    //借阅日期
    string bookname; //书名
public:
    // 设置书号
    void setbookno(string bno);
    // 设置借阅日期
```

```

void setborrowday(Date d);

// 获取书号

string getbookno();

// 获取借阅日期

Date getborrowday();

friend std::istream& operator>>(std::istream& is, BorrowInfo& bi);    //重载输入输出
运算符

friend std::ostream& operator<<(std::ostream& os, BorrowInfo& bi);

// 设置书名

void setbookname(string bname);

// 获取书名

string getbookname();

};

```

## 1.4 StudentSet 类

### 1.4.1 类定义

```

class StudentSet :public vector < Student >
{
public:
    //StudentSet();

    //virtual ~StudentSet();

    Status ListInsert_S(Student e);    //学生表插入元素

    int ListLength_S();// 获取学生列表的长度

    Status ListDelete_S(int i, Student& e);    //删除学生表元素

    Status LocateAllElem_S(Student e, Status(Student::* compare)(Student), int* find);// 定
位所有满足特定条件的学生元素

    Status GetElem_S(int i, Student& e);    //学生表 La 的第 i 个元素存于 e 中

    Status SetElem_S(int i, Student e);    //学生表 La 的第 i 个元素设为 e

```

```

        Status OpenStuList();                                //载入学生信息

        Status stu_manage_output(int pos = 0);

        void Append_stu(string filename);// 将学生信息追加到指定文件

        void stu_manage_append();// 管理追加操作，可能包括将新的学生信息追加到文件
或列表

        void stu_manage_delete(); //学生信息毕业处理

        void stu_manage_update(); //修改学生信息

        void stu_search_char(string info, Status(Student::* fun)(string)); //查询学生信息

        void stu_manage_search();//学生信息处理系统

        void writeToFile(const std::string& filename);      // 将学生信息写入到指定的文件
};

```

## 1.4.2 主要算法

### ① Append\_stu

参数说明： filename: string 类型，指定从中读取学生数据的文件名。

返回说明： 无。

BEGIN

- 初始化局部变量 k 用于存储当前学生集合的长度, e 用于临时存储从文件中读取的学生对象。

- 尝试打开指定的文件名 filename。

- 如果文件成功打开：

使用 ListLength\_S() 方法获取当前学生集合中的元素数量并存储在变量 k 中。

使用循环读取文件中的每个学生信息：

使用重载的输入运算符 >> 从文件中读取一个学生对象到 e。

检查读取的学生对象的学号是否为空(空学号可能表示文件中的数据结束或格式错误)，若为空则终止读取。

如果学号不为空，调用 ListInsert\_S(e) 方法将读取的学生对象 e 插入到学生集合中。

文件读取完毕后关闭文件。

调用 `stu_manage_output(k + 1)` 以输出新增的学生信息到控制台或 UI，参数 `k + 1` 表示从新增的第一条记录开始输出。

END

## 1.5 SBList 类

### 1.5.1 类定义

```
class SBList
{
private:
    int studentnum;           //借阅书籍的人数
    int borrownum;           //借出去的书总数
    StudentNodeSet sblist;    //学生借阅表
public:
    // 开始一个新的借阅流程
    Status begin_borrow();

    // 打开并加载借阅列表，通常从文件中读取
    Status OpenBorrowList();

    // 将当前的借阅信息写入到文件中
    Status WriteToFile();

    // 检查指定学生是否已在借阅列表中
    int StuWhetherIn(string no);

    // 向借阅列表中添加一个新的学生节点
    void pushelem(StudentNode sn);

    // 设置当前借阅书籍的人数
    void setstudentnum(int s);

    // 设置借出去的书总书数
    void setborrownum(int b);
```



```

// 获取当前借阅书籍的人数
int getstudentnum();

// 获取借出去的书籍总数
int getborrownum();

// 获取指定学生的个人借阅书籍数量
int getpersonalnum(string no);

// 检查指定书籍是否已被指定学生借阅
int BookWhetherIn(string sn, string bn);

// 为指定学生添加一个新的借阅关系
void addanewrelation(string sn, BorrowInfo bi);

// 学生归还一本书
void returnbook(string sn, string bn);

// 打印指定学生的所有借阅信息
void printbookinfo(string sn);

// 打印单一书籍的借阅信息
void printsinglebookinfo(string bn);

// 更新借阅列表中的逾期书籍信息
void outdate(Date now, StudentSet stu);

// 获取学生借阅节点集
StudentNodeSet getSBList();

// 检查指定学生的借书限制是否达到上限
bool bookLimit(string sn);
};

```

## 1.5.2 主要算法

### ① OpenBorrowList

参数说明： 无。

返回说明： 返回 `Status` 类型，如果文件成功读取并处理，则返回 `OK`；如果文件读取

失败，则返回 **ERROR**。

**BEGIN**

- 尝试打开文件 "borrow.txt" 以读取借阅数据。
- 检查文件是否成功打开：

如果文件未成功打开，输出错误信息并返回 **ERROR**。

- 重置 **studentnum**（借阅人数）为 0。
- 重置 **borrownum**（借出去的书总数）为 0。
- 通过循环读取文件中的每个学生节点（**StudentNode**）：

对每个读取的学生节点，使用 **pushelem** 方法将其添加到借阅列表中。

累加每个学生节点的借出书籍数量到 **borrownum**。

- 文件读取完成后，关闭文件。
- 返回 **OK** 表示操作成功。

**END**

## ② WriteToFile

参数说明： 无。

返回说明： 返回 **Status** 类型，如果文件成功打开并数据成功写入，则返回 **OK**；如果文件打开失败，则返回 **ERROR**。

**BEGIN**

- 尝试打开文件 "borrow.txt" 用于写入借阅信息。
- 检查文件是否成功打开：

如果文件未成功打开，返回 **ERROR**。

- 写入学生借阅信息的数量 **studentnum** 到文件中。
- 通过调用 **getSBList** 方法获取当前所有的 **StudentNode**，并遍历每个 **StudentNode**：

使用重载的输出运算符 **<<** 将每个 **StudentNode** 的信息写入到文件中。

- 文件写入完成后，关闭文件。
- 返回 **OK** 表示操作成功。

**END**

### ③ bookLimit

参数说明: `sn: string` 类型, 表示学生的学号。

返回说明: 返回 `bool` 类型:

返回 `true` 表示学生可以继续借书。

返回 `false` 表示学生已达到借书数量上限, 不能继续借书。

**BEGIN**

- 遍历 `SList` 中的 `slist` 学生借阅列表, 搜索匹配学号 `sn` 的学生节点。

如果找到匹配的学生节点, 检查该学生的借书数量。

- 判断条件:

如果遍历结束没有找到匹配的学生 (即迭代器 `iter` 等于 `slist.end()`), 说明学生尚未借书, 返回 `true`。

如果找到学生节点, 检查其 `getborrownum` 返回的借书数量是否少于 5 本:

如果少于 5 本, 返回 `true`。

如果等于或多于 5 本, 返回 `false`。

**END**

### ④ returnbook

参数说明:

- `sn: string` 类型, 表示归还书籍的学生的学号。
- `bn: string` 类型, 表示被归还的书籍的书号。

返回说明: 无返回值 (`void` 类型)。

**BEGIN**

- 遍历 `slist` 学生借阅列表, 寻找学号为 `sn` 的学生节点。

如果找到对应学生, 终止搜索。

- 调用学生节点的 `returnbook` 方法, 传入书号 `bn`, 执行归还书籍操作。
- 将总借出书籍数量 `borrownum` 减一。
- 检查该学生节点归还后的借书数量:

如果 `getborrownum` 方法返回值为 0, 表示学生无借阅书籍:

从 sblist 中移除该学生节点。

将借阅学生人数 studentnum 减一。

END

## 1.6 Admin\_InfoDialog 类

### 1.6.1 类定义

```
class Admin_InfoDialog : public QDialog
{
    Q_OBJECT

public:
    Admin_InfoDialog(QWidget* parent = nullptr);
    ~Admin_InfoDialog();

    void set_TabelWghit();    // 设置图书表格的界面元素
    void set_TabelWghit_stu();    // 设置学生表格的界面元素
    std::vector<QPushButton*> btn; // 表格上的控件
    std::vector<QCheckBox*> mark; // 表格上的选中标记
    std::vector<QPushButton*> stu_btn;    // 存储学生表格中的按钮
    std::vector<QCheckBox*> stu_mark;    // 存储学生表格中的复选框
    void refreshTable(BookSet* set);    // 刷新书籍信息表格，显示给定的书籍集
    void exchange_Set(const string str);

    int find_book_and_return_idx(Book* book);    // 在书籍集中查找指定书籍并返回
其索引
    int find_stu_and_return_idx(Student* stu);    // 在学生集中查找指定学生并返回
其索引

    //安全性验证
    bool requireInt(const QString& str); // 验证字符串是否为整数
    bool isValidDateFormat(const QString& Qstr); // 验证日期格式是否有效
    string getSafeString(const QTableWidgetItem* item); // 从表项中获取安全字符串
```

bool isNumber(string str);// 检查字符串是否全部为数字

bool isBooknoUnique(int i, QTableWidgetItem\* item);// 检查书号是否唯一

bool isStunoUnique(int i, QTableWidgetItem\* item);// 检查学号是否唯一

public slots:

// 设置当前页面

void set\_page(int page\_name);

// 添加项目到书籍表格

void add\_item\_to\_TableWghit();

// 从表格获取书籍对象

Book\* get\_book\_from\_TableWghit(int i);

// 保存或更改表格中的项目

void save\_or\_change\_for\_TableWghit(int i);

// 从书籍表格删除项目

void delete\_book\_from\_TableWghit();

// 取消书籍表格中的选定项目

void cancel\_item\_from\_TableWghit();

// 全选或取消选择所有项目

void choise\_or\_unchoise\_all\_item\_from\_TableWghit();

// 从文件添加书籍

void add\_book\_from\_file();

// 执行书籍搜索操作

void onSearchAction\_book();

// 执行学生搜索操作

void onSearchAction\_stu();

// 刷新学生信息表格

void refreshTable\_stu(StudentSet\* s\_set);

// 添加项目到学生表格

void add\_item\_to\_stu\_TableWghit();

// 保存或更改学生表格中的项目

void save\_or\_change\_for\_stu\_TableWghit(int i);

```

// 从学生表格删除项目

void delete_book_from_stu_TableWghit();

// 全选或取消选择所有学生表格项目

void choise_or_unchoise_all_item_from_stu_TableWghit();

// 从学生表格获取学生对象

Student* get_student_from_stu_TableWghit(int i);

// 从文件添加学生

void add_stu_from_file();

// 显示自定义消息框

void showCustomMessageBox(const QString& title, const QString& message);

private:

    Ui::Admin_InfoDialogClass ui;// 管理员界面对象

    BookSet* b_set;// 主书籍集

    StudentSet* s_set;// 主学生集

    BookSet* search_BookSet; // 搜索结果书籍集

    StudentSet* search_StuSet; // 搜索结果学生集

    BookSet* current_BookSet; // 当前活动的书籍集

    StudentSet* current_StuSet; // 当前活动的学生集

};

```

## 1.6.2 主要算法

### ① Admin\_InfoDialog 构造函数

参数说明：

- parent: QWidget\* 类型，指向父窗口的指针。默认为 nullptr，表示该对话框可能是顶级窗口。

返回说明： 无返回值（构造函数）。

BEGIN

构造过程：

调用基类 `QDialog` 的构造函数，传入 `parent` 参数。

初始化数据成员：

创建 `BookSet` 和 `StudentSet` 的实例用于存储书籍和学生数据。

调用 `OpenBookList()` 和 `OpenStuList()` 方法分别加载书籍和学生数据。

设置当前活动的数据集为新创建的书籍和学生集合。

调用 `setupUi` 方法设置界面布局。

调用 `set_TabelWghit()` 和 `set_TabelWghit_stu()` 方法初始化书籍和学生信息表格界面。

使用 `Qt` 的信号与槽机制连接界面上的按钮（如页面切换按钮），以支持页面内容的切换。

信号连接：

连接 `ui.pb_page1` 和 `ui.pb_page2` 的 `clicked` 信号到 `set_page` 槽函数，以实现页面切换功能。

END

## ② onSearchAction\_book

参数说明： 无参数。

返回说明： 无返回值（`void` 类型）。

BEGIN

操作步骤：

检查界面上的搜索开关 `ui.search_on` 是否被勾选：

如果勾选，进行搜索操作：

清空之前的搜索结果集 `search_BookSet`。

获取搜索框 `ui.searchLE` 中的文本，并转换为标准字符串 `searchText`。

遍历当前的书籍集 `b_set`，对每本书籍使用多个字段进行搜索匹配：

作者（`getauthor`）

书名（`getbookname`）

书号（`getbookno`）

借出数量（`getborrownum`），转换为字符串后匹配

出版日期（`getpubday().dateFormat()`），假设有一个返回格式化日期的函数 `dateFormat`

出版社（getpublisher）

总库存量（gettotalnum），转换为字符串后匹配

如果找到匹配项，将该书籍加入到 search\_BookSet。

调用 exchange\_Set 将当前书籍集设置为搜索结果集。

刷新表格显示 refreshTable，展示搜索结果。

如果未勾选，重置显示为原始书籍集：

调用 exchange\_Set 将当前书籍集重置为原始集 b\_set。

刷新表格显示 refreshTable，展示原始书籍数据。

END

### ③ add\_book\_from\_file

参数说明：

- parent: QWidget\* 类型，指向父窗口的指针。默认为 nullptr，表示该对话框可能是顶级窗口。

返回说明：无返回值（构造函数）。

BEGIN

构造过程：

调用基类 QDialog 的构造函数，传入 parent 参数。

初始化数据成员：

创建 BookSet 和 StudentSet 的实例用于存储书籍和学生数据。

调用 OpenBookList() 和 OpenStuList() 方法分别加载书籍和学生数据。

设置当前活动的数据集为新创建的书籍和学生集合。

调用 setupUi 方法设置界面布局。

调用 set\_TabelWghit() 和 set\_TabelWghit\_stu() 方法初始化书籍和学生信息表格界面。

使用 Qt 的信号与槽机制连接界面上的按钮（如页面切换按钮），以支持页面内容的切换。

信号连接：

连接 ui.pb\_page1 和 ui.pb\_page2 的 clicked 信号到 set\_page 槽函数，以实现页面切换功能。



END

## ② onSearchAction\_book

参数说明： 无参数。

返回说明： 无返回值（void 类型）。

BEGIN

打开文件对话框：

使用 `QFileDialog::getOpenFileName` 方法打开一个文件选择对话框，允许用户选择文件。

参数包括父窗口指针（`this`），对话框标题（`tr("select file")`），默认路径（空字符串，表示当前目录），文件过滤器（支持所有文件和文本文件）。

处理文件选择：

如果用户选择了文件（`filePath` 不为空）：

显示一个消息框，通知用户选择的文件路径。

调用 `BookSet` 的 `Append_book` 方法，将选择的文件中的书籍数据加载到当前书籍集 `b_set` 中。

如果用户取消了文件选择（`filePath` 为空）：

显示一个消息框，通知用户他们没有选择任何文件。

更新界面和保存数据：

调用 `refreshTable` 方法，更新表格视图以显示最新的书籍数据。

调用 `BookSet` 的 `writeToFile` 方法，将当前书籍集的数据保存到 `"book.txt"` 文件中。

END

## ③ delete\_book\_from\_stu\_TableWghit

参数说明： 无参数。

返回说明： 无返回值（void 类型）。

BEGIN

遍历所有标记复选框：

使用 `for` 循环遍历 `stu_mark` 数组，该数组存储表格中每行对应的复选框。

对于每个复选框，检查是否被勾选 (`isChecked() == true`)。

删除选中的学生信息：

如果复选框被勾选：

销毁对应的复选框和按钮资源，使用 `delete` 操作符。

从表格 `ui.tableWidget_stu` 中移除对应的行。

从 `stu_mark` 和 `stu_btn` 数组中移除对应的元素。

使用 `find_stu_and_return_idx` 函数获取该学生在当前学生集 `current_StuSet` 中的索引。

从学生集 `s_set` 中移除对应的学生条目。

调整循环变量 `i` 以正确处理下一个元素（因为数组长度已经减少）。

更新文件：

调用 `StudentSet` 的 `writeToFile` 方法，将更新后的学生信息写入到 "student1.txt" 文件中。

END

#### ④ `requireInt`

参数说明： `str`: `QString` 类型，需要进行验证的字符串。

返回说明： 返回 `bool` 类型：

返回 `true`，如果字符串 `str` 中的所有字符都是数字。

返回 `false`，如果字符串 `str` 中包含任何非数字字符。

BEGIN

使用 `std::all_of` 函数遍历 `str` 的每个字符：

传入的范围是从 `str.begin()` 到 `str.end()`。

使用 `lambda` 表达式 `[](QChar ch) { return ch.isDigit(); }` 作为条件，检查每个字符是否为数字。

如果所有字符都满足 `isDigit` 条件，即它们都是数字，则 `std::all_of` 返回 `true`。

否则，如果任何字符不是数字，`std::all_of` 返回 `false`。

END

## ⑤ isBooknoUnique

参数说明：

**i:** int 类型，当前正在检查或编辑的书籍的索引。它帮助识别是否对现有书籍进行修改。

**item:** QTableWidgetItem\*, 包含需要检查唯一性的字符串的表格项。

返回说明： 返回 bool 类型：

如果书号唯一或条目是新添加的，则返回 true。

如果在数据集中发现重复的书号（排除当前正在检查的条目），则返回 false。

BEGIN

使用 `getSafeString` 函数从 `QTableWidgetItem` 中提取字符串，确保安全地提取字符串进行比较。

初始化 `Book` 类型的指针 (`ptr`) 为 `NULL`。如果索引 `i` 在 `b_set` 的范围内，将 `ptr` 指向当前书籍对象，以便在重复检查时排除它。

遍历整个 `b_set` 集合，检查是否有任何书籍与输入的字符串具有相同的书号：

如果 `ptr` 不为 `NULL` 且指向迭代中的当前书籍，则跳过比较。

将集合中每本书的书号与输入字符串进行比较。

如果发现重复 (`it->getbookno() == str`)，使用 `showCustomMessageBox` 弹出自定义消息框，警告用户书号重复。

如果检测到重复，返回 `false`。

如果在集合中检查所有书籍后未发现重复，则返回 `true`。

END

## 1.7 Student\_InfoDialog 类

### 1.7.1 类定义

```
class Student_InfoDialog : public QDialog
{
```

```

        Q_OBJECT

public:
    Student_InfoDialog(Student* m_stu, BookSet* m_book,SBList* m_borrow, QWidget
*parent = nullptr);

    ~Student_InfoDialog();

    // 初始化页面，设置各个子页面的初始状态

    void initPage();

public slots:

    // 处理菜单动作，根据用户的选择调整界面显示

    void dealMenu();

private:

    Ui::Student_InfoDialogClass ui;// 用户界面的设置，由 Qt 的 UIC 工具自动生成和
管理

    // 学生借阅页面

    cell_borrow* m_borrowPage;

    // 学生归还页面

    cell_return* m_returnPage;

    // 学生个人信息页面

    cell_personInfo* m_personInfoPage;

    // 推荐书籍页面

    cell_recommend* m_recommendPage;

    // 指向当前操作的学生对象

    Student* stu;

    // 指向当前操作的书籍集合

    BookSet* book;

    // 指向当前借阅信息的链表集合

    SBList* borrow;

};

```

## 1.7.2 主要算法

### ① initPage

参数说明： 无参数。

返回说明： 无返回值（void 类型）。

BEGIN

- 创建各功能页面的实例：

m\_borrowPage: 创建借阅页面的实例，传递学生信息、书籍集合、借阅列表和父窗口指针。

m\_returnPage: 创建归还页面的实例，同样传递相关参数。

m\_personInfoPage: 创建个人信息页面的实例，传递学生信息和父窗口指针。

m\_recommendPage: 创建推荐书籍页面的实例，传递借阅列表、书籍集合和父窗口指针。

- 添加页面到堆栈小部件：

使用 ui.stackedWidget->addWidget 方法将上述创建的页面依次添加到界面的堆栈小部件中。

- 设置初始显示的页面：

使用 ui.stackedWidget->setCurrentIndex(0) 方法设置堆栈小部件初始显示的页面为第一个添加的页面（借阅页面）。

- 连接工具条按钮的信号：

遍历 ui.tool 容器的子对象，查找所有具有 "Btn" 在其对象名中的按钮。

对于找到的每个按钮，使用 connect 函数连接其 clicked 信号到 dealMenu 槽函数，以便根据按钮点击切换显示的页面。

END

### ② dealMenu

参数说明： 无参数。

返回说明： 无返回值（void 类型）。

BEGIN

- 获取触发者信息：

使用 `sender()` 方法获取当前信号的发送者（即被点击的按钮），并调用 `objectName()` 获取其对象名称，存储于局部变量 `str`。

- 判断触发者并执行对应操作：

使用 `do-while(false)` 结构允许在条件满足时跳出执行，类似于 `switch-case` 的行为。

对于每个可能的按钮名称，进行比较并执行对应的页面刷新或切换操作：

**bookTBtn:** 如果是借阅书籍按钮，调用 `m_borrowPage->refreshTable(book)` 刷新借阅页面，并将堆栈小部件的当前页面设置为借阅页面 (`setCurrentIndex(0)`)。

**borrowTBtn:** 如果是归还书籍按钮，调用 `m_returnPage->refreshTable()` 刷新归还页面，并切换到归还页面 (`setCurrentIndex(1)`)。

**personTBtn:** 如果是个人信息按钮，直接切换到个人信息页面 (`setCurrentIndex(2)`)。

**recommendTBtn:** 如果是推荐书籍按钮，调用 `m_recommendPage->refreshTable()` 刷新推荐书籍页面，并切换到推荐页面 (`setCurrentIndex(3)`)。

END

## 1.8 cell\_borrow 类

### 1.8.1 类定义

```
class cell_borrow : public QWidget
{
    Q_OBJECT

public:
    cell_borrow(Student* stu, BookSet* set, SBList* borrow, QWidget *parent = nullptr);
    ~cell_borrow();

    // 初始化借阅表格的设置，如列头、布局等
    void borrowTableSetting();

    // 刷新借阅表格，显示给定的书籍集
    void refreshTable(BookSet* set);

public slots:
```

```

// 右键点击表格时显示菜单

void onRightClickShowMenu(QPoint pos);

// 处理借书动作，根据书籍 ID 处理借书逻辑

void onBorrowBookAction(const QString& m_BID);

// 执行搜索动作，根据用户输入更新表格显示

void onSearchAction();

// 更新时间，可能用于刷新界面或处理超时

void updateTime();

private:

    BookSet* b_set; // 存储与当前操作相关的书籍集合

    SBList* b_borrow; // 存储与当前操作相关的借阅信息集合

    Student* b_stu; // 指向当前操作的学生对象

    Ui::cell_borrowClass ui; // 用户界面的设置，由 Qt 的 UIC 工具自动生成和管理

    QTimer* timer; // 计时器，用于周期性执行 updateTime

};

```

## 1.8.2 主要算法

### ① onSearchAction

参数说明： 无参数。

返回说明： 无返回值（void 类型）。

BEGIN

- 初始化搜索结果集：

创建一个 BookSet 类型的局部变量 searchedSet 来存储搜索结果。

获取搜索文本：

从界面的搜索框 (ui.searchLE) 中获取文本，转换为标准字符串 searchText。

判断搜索文本是否为空：

如果 searchText 为空，将所有书籍从 b\_set 添加到 searchedSet。

如果 searchText 非空，执行详细搜索。

执行搜索：

遍历书籍集合 `b_set`，对每本书籍执行以下搜索条件：

使用 `std::string::find` 方法检查书籍的作者 (`getauthor()`)、书名 (`getbookname()`)、书号 (`getbookno()`)、出版日期 (`getpubday().dateFormat()`)、出版社 (`getpublisher()`) 和当前可借数量 (`gettotalnum() - getborrownum()`，转换为字符串后比较) 是否包含 `searchText`。

捕捉并忽略所有可能的异常，保证搜索过程的稳定性。

刷新显示搜索结果：

调用 `refreshTable(&searchedSet)` 方法，传入搜索结果集 `searchedSet`，以更新界面显示。

END

## ② onBorrowBookAction

参数说明：`m_BID`: `QString` 类型，用户试图借阅的书籍的 ID。

返回说明：无返回值 (`void` 类型)。

BEGIN

初始化书籍对象并设置书号：

创建 `Book` 类型的对象 `book`，并使用 `m_BID.toString()` 设置书号。

查找书籍并处理借阅：

如果书籍集合 `b_set` 不为空，使用 `std::find` 查找匹配的书籍。

如果找到书籍，并且该书的借出数量小于总数，执行借阅操作：

更新书籍的借出数量。

创建 `BorrowInfo` 对象并设置相应的借阅信息。

检查学生是否在借阅列表中，以及是否还可以借阅更多书籍。

更新文件和显示结果：

如果学生可以继续借阅，添加新的借阅关系，更新相关文件，并通过消息框显示借阅成功的信息。

如果达到借阅上限，显示借阅失败的消息。

如果书籍已全部借出，显示相应的消息。

刷新表格显示：



调用 `refreshTable` 方法，更新界面上的书籍显示列表。

错误处理：

如果书籍不存在或不可借，显示适当的错误消息。

消息框用法：

使用 `QMessageBox` 显示借阅成功或失败的详细信息，包括设置消息框的图标、标题和文本内容。

END

## 1.9 cell\_return 类

### 1.9.1 类定义

```
class cell_return : public QWidget
{
    Q_OBJECT

public:
    cell_return(Student* m_stu, SBList* list, BookSet* set, QWidget *parent = nullptr);
    ~cell_return();

    // 初始化归还表格的设置，如列头、布局等
    void returnTableSetting();

    // 刷新归还表格，显示当前的借阅信息
    void refreshTable();

    // 设置和管理图片轮播
    void imageCarousel();

public slots:
    // 处理归还书籍动作，根据书籍 ID 处理归还逻辑
    void onReturnBookAction(const QString& m_BID);

    // 右键点击表格时显示菜单
```

```

void onRightClickShowMenu(QPoint pos);

// 显示下一张图片的轮播功能

void nextImage();

private:

    Ui::cell_returnClass ui; // 用户界面的设置，由 Qt 的 UIC 工具自动生成和管理

    QTimer* timer; // 计时器，用于周期性执行 nextImage

    SBLIST* r_list; // 存储与当前操作相关的借阅信息集合

    Student* r_stu; // 指向当前操作的学生对象

    BookSet* b_set; // 存储与当前操作相关的书籍集合

    QStringList imageList; // 存储图片路径的列表

    int currentIndex; // 当前显示的图片索引

};

```

## 1.9.2 主要算法

### ① imageCarousel

参数说明： 无参数。

返回说明： 无返回值（void 类型）。

**BEGIN**

创建并配置计时器：

创建一个新的 QTimer 实例，与当前的 cell\_return 对象关联。

设置计时器的超时信号 timeout 连接到 nextImage 槽函数，用于在计时器触发时调用 nextImage 方法更换图片。

设置图片列表和初始图片：

初始化 imageList，一个 QStringList，包含所有图片的资源路径。

设置初始图片索引 currentIndex 为 0。

使用 ui.imageLabel->setPixmap 方法，将标签的图片设置为 imageList 中的第一张图片（通过索引 currentIndex 访问）。

启动计时器：

调用 `timer->start(3000)`，设置计时器每 3 秒触发一次，以实现图片自动轮播。

END

## ② onReturnBookAction

参数说明： `m_BID: QString` 类型，用户试图归还的书籍的 ID。

返回说明： 无返回值（`void` 类型）。

BEGIN

初始化书籍对象并设置书号：

创建 `Book` 类型的对象 `book`，并使用 `m_BID.toString()` 设置书号。

检查书籍集合是否为空：

如果 `b_set` 不为空，继续执行；否则，显示归还失败的消息。

查找书籍并处理归还：

使用 `std::find` 查找匹配的书籍。

如果找到书籍，执行以下操作：

减少书籍的借出数量。

创建 `BorrowInfo` 对象并设置归还日期。

检查学生是否在借阅列表中，更新学生的借阅记录。

调用 `SList` 的 `returnbook` 方法更新借阅信息。

更新书籍和借阅信息文件。

显示归还成功的消息。

显示结果：

使用 `QMessageBox` 显示归还成功或失败的信息。

刷新表格显示：

调用 `refreshTable` 方法，更新界面上的书籍显示列表。

错误处理与用户反馈：

如果书籍不在集合中，显示归还失败的消息。

使用 `QMessageBox` 显示操作结果，增加用户体验。

END

## 2. 主要功能函数的算法

### 2.1 主函数

```
int main(int argc, char *argv[])

BEGIN

    /* 创建一个 QApplication 对象。这个对象处理事件循环和其他应用程序级别的任
    务。对于任何 Qt 应用程序来说，

        QApplication 对象是必需的。所有的 Qt GUI 应用程序必须有且只有一个
    QApplication 对象。 */

    QApplication a(argc, argv);

    /* 创建一个 PersonSet 对象，用于存储人员信息。 */

    PersonSet personSet;

    /* 调用 LoadLibToSet 函数，将数据从库加载到 personSet 对象中。 */

    LoadLibToSet(personSet);

    /* 创建 PersonInfListDlg 对象，传入 personSet 对象的地址。这个对话框用于显示
    和操作人员信息。 */

    PersonInfListDlg w(&personSet);

    /* 显示 w 窗体 */

    w.show();

    /* 开启 Qt 的事件循环，函数阻塞运行直到程序结束 */

    a.exec();

    /* 调用 SaveSetToLib 函数，将 personSet 对象中的数据保存回库。 */

    SaveSetToLib(personSet);

    /* 返回 1 表示程序正常结束 */

    return 1;

END
```

## 2.2 图书信息读取函数

```
void BookSet::OpenBookList() //载入 TXT 的图书信息

BEGIN

    /* 打开名为 "book.txt" 的文件，用于存储图书信息。*/

    ifstream INFILE("book.txt");

    /* 检查文件是否成功打开。如果文件打开失败，则退出函数。*/

    if (INFILE)

    {

        /* 初始化计数器 k 用于跟踪读取的图书数量。*/

        int k = 0;

        /* 循环读取文件直到文件末尾。*/

        while (!INFILE.eof())

        {

            /* 声明一个 Book 对象 e 用于存储读取的图书信息。*/

            Book e;

            /* 从文件中读取图书信息并存储到 Book 对象 e 中。*/

            INFILE >> e;

            /* 检查读取的图书信息是否为空，如果是，则退出循环。*/

            if (e.getbookno() == "")

                break;

            /* 将读取的图书信息插入到 BookSet 集合中。*/

            this->ListInsert_B(e);

            /* 增加读取的图书数量计数器。*/

            k++;

        }

        /* 关闭文件。*/

        INFILE.close();

        /* 输出图书信息载入成功的消息。*/

        cout << "图书信息载入成功\n";
```

```

    }
else
{
    /* 如果文件打开失败，输出错误消息。*/
    cout << "无法打开文件，图书信息载入失败。\\n";
}
END

```

## 2.3 学生信息读取函数

```

Status StudentSet::OpenStuList()                //载入学生信息
BEGIN
    /* 打开名为 "student1.txt" 的文件，用于存储学生信息。*/
    ifstream INFILE("student1.txt");

    /* 检查文件是否成功打开。如果文件打开失败，则返回错误状态。*/
    if (INFILE)
    {
        /* 初始化计数器 k 用于跟踪读取的学生数量。*/
        int k = 0;

        /* 循环读取文件直到文件末尾。*/
        while (!INFILE.eof())
        {
            /* 声明一个 Student 对象 e 用于存储读取的学生信息。*/
            Student e;

            /* 从文件中读取学生信息并存储到 Student 对象 e 中。*/
            INFILE >> e;

            /* 检查读取的学生信息是否为空，如果是空，则退出循环。*/
            if (e.getstuno() == "")
                break;
        }
    }
}

```

```

        /* 将读取的学生信息插入到 StudentSet 集合中。*/
        this->ListInsert_S(e);

        /* 增加读取的学生数量计数器。*/

        k++;

    }

    /* 关闭文件。*/

    INFILE.close();

    /* 输出学生信息载入成功的消息。*/

    cout << "student1.txt 载入已成功!\n";

    /* 返回操作成功状态。*/

    return OK;

}

else

{

    /* 如果文件打开失败，输出错误消息。*/

    cout << "信息载入失败！请检查\n";

    /* 返回操作失败状态。*/

    return ERROR;

}

END

```

## 2.4 学生信息插入算法：

```

Status StudentSet::ListInsert_S(Student e)    //学生表插入元素

BEGIN

    /* 获取学生集合的开始和结束迭代器。*/

    vector<Student>::iterator begin = this->begin(), end = this->end();

    /* 检查集合中是否已存在相同的学生信息。如果存在，则不插入。*/

    if (find(begin, end, e) == this->end())

```

```

{
    /* 如果没有找到相同的学生信息，则在集合末尾插入新的学生信息。*/
    vector<Student>::push_back(e);

    /* 返回操作成功状态。*/
    return OK;
}

/* 如果找到相同的学生信息，则返回错误状态。*/
return ERROR;
END

```

## 2.5 学生信息加载算法

### 1. 函数

```

Status StudentSet::OpenStuList()                //载入学生信息
BEGIN
    /* 打开存放学生信息的文件 "student1.txt"。*/
    ifstream INFILE("student1.txt");

    /* 检查文件是否成功打开。如果文件打开失败，则返回错误状态。*/
    if (INFILE)
    {
        /* 初始化计数器 k 用于跟踪读取的学生数量。*/
        int k = 0;

        /* 循环读取文件直到文件末尾。*/
        while (!INFILE.eof())
        {
            /* 声明一个 Student 对象 e 用于存储读取的学生信息。*/
            Student e;

```



```

        /* 从文件中读取学生信息并存储到 Student 对象 e 中。*/
        INFILE >> e;

        /* 检查读取的学生信息是否为空，如果是空，则退出循环。*/
        if (e.getstuno() == "")
            break;

        /* 将读取的学生信息插入到 StudentSet 集合中。*/
        this->ListInsert_S(e);

        /* 增加读取的学生数量计数器。*/
        k++;
    }

    /* 关闭文件。*/
    INFILE.close();

    /* 输出学生信息载入成功的消息。*/
    cout << "student1.txt 载入已成功!\n";

    /* 返回操作成功状态。*/
    return OK;
}

else
{
    /* 如果文件打开失败，输出错误消息。*/
    cout << "信息载入失败！请检查\n";

    /* 返回操作失败状态。*/
    return ERROR;
}

```

END

## 2.6 管理员书籍信息加载算法

### 1. 函数

```
void Admin_InfoDialog::refreshTable(BookSet* set)
```

```
// 刷新表格显示
```

```
BEGIN
```

```
    /* 清空表格行数。*/
```

```
    ui.tableWidget->setRowCount(0);
```

```
    /* 获取当前集合的大小。*/
```

```
    int rowCount = set->size();
```

```
    /* 清空按钮和复选框列表。*/
```

```
    btn.clear();
```

```
    mark.clear();
```

```
    /* 遍历集合，为表格设置数据。*/
```

```
    BookSet::iterator it;
```

```
    for (it = set->begin(), i = 0; it != set->end(); it++, i++) {
```

```
        /* 设置列宽。*/
```

```
        int col = 1;
```

```
        ui.tableWidget->setItem(i, col++, new
```

```
QTableWidgetItem(QString::fromStdString(it->getbookno())));
```

```
        ui.tableWidget->setItem(i, col++, new
```

```
QTableWidgetItem(QString::fromLocal8Bit(it->getbookname())));
```

```
        ui.tableWidget->setItem(i, col+, new
```

```
QTableWidgetItem(QString::fromLocal8Bit(it->getpublisher())));
```

```
        ui.tableWidget->setItem(i, col+, new
```

```

QTableWidgetItem(QString::fromLocal8Bit(it->getauthor())));

                ui.tableWidget->setItem(i,                col+,                new
QTableWidgetItem(QString::number(it->gettotalnum())));

                ui.tableWidget->setItem(i,                col+,                new
QTableWidgetItem(QString::number(it->getborrownum())));

                ui.tableWidget->setItem(i,                col+,                new
QTableWidgetItem(QString::fromLocal8Bit(it->getpubday().dateFormat())));

                /* 为新增行添加更新按钮和复选框。 */

                btn.push_back(new QPushButton(QString::fromLocal8Bit("更新")));

                mark.push_back(new QCheckBox());

                connect(btn[i],                &QPushButton::clicked,                [this,                i]()
{Admin_InfoDialog::save_or_change_for_TableWghit(i); });

                ui.tableWidget->setCellWidget(i, 8, btn[i]);

                ui.tableWidget->setCellWidget(i, 0, mark[i]);

        }

END

```

## 2.7 管理员学生信息添加算法

### 1. 函数

```

void Admin_InfoDialog::add_item_to_stu_TableWghit()
{
    /* 获取当前表格行数。 */

    int currentRowCount = this->ui.tableWidget_stu->rowCount();

    /* 增加一行。 */

    this->ui.tableWidget_stu->setRowCount(currentRowCount + 1);

    /* 为新增行添加更新按钮和复选框。 */

```

```

        stu_btn.push_back(new QPushButton(QString::fromLocal8Bit("确认")));

        stu_mark.push_back(new QCheckBox());

        stu_mark[currentRowCount]->setStyleSheet("QRadioButton { max-width:20px; }");

        this->ui.tableWidget_stu->setCellWidget(currentRowCount,                                6,
stu_btn[currentRowCount]);

        this->ui.tableWidget_stu->setCellWidget(currentRowCount,                                0,
stu_mark[currentRowCount]);

        /* 连接按钮点击事件处理函数。*/

        connect(stu_btn[currentRowCount], &QPushButton::clicked, [this, currentRowCount]()
{Admin_InfoDialog::save_or_change_for_stu_TableWghit(currentRowCount); });

        /* 连接复选框点击事件处理函数。*/

        //connect(stu_mark[currentRowCount], &QCheckBox::clicked, [this, currentRowCount]()
{Admin_InfoDialog::chosed_mark_for_TableWghit(currentRowCount); });

    }

END

```

## 2.8 学生书籍信息加载算法

### 1. 函数

```
void Admin_InfoDialog::refreshTable(BookSet* set)
```

```
// 刷新表格显示
```

```
BEGIN
```

```
    /* 清空表格行数。*/
```

```
    ui.tableWidget->setRowCount(0);
```

```
    ui.tableWidget->setRowCount(set->size());
```

```
    /* 清空按钮和复选框列表。*/
```

```
    btn.clear();
```

```
    mark.clear();
```

```

/* 遍历集合，为表格设置数据。*/

BookSet::iterator it;

for (it = set->begin(), i = 0; it != set->end(); it++, i++) {

    /* 设置列宽。*/

    int col = 1;

    ui.tableWidget->setItem(i, col++, new
QTableWidgetItem(QString::fromStdString(it->getbookno())));

    ui.tableWidget->setItem(i, col++, new
QTableWidgetItem(QString::fromLocal8Bit(it->getbookname())));

    ui.tableWidget->setItem(i, col+, new
QTableWidgetItem(QString::fromLocal8Bit(it->getpublisher())));

    ui.tableWidget->setItem(i, col+, new
QTableWidgetItem(QString::fromLocal8Bit(it->getauthor())));

    ui.tableWidget->setItem(i, col+, new
QTableWidgetItem(QString::number(it->gettotalnum())));

    ui.tableWidget->setItem(i, col+,
QTableWidgetItem(QString::number(it->getborrownum())));

    ui.tableWidget->setItem(i, col+,
QTableWidgetItem(QString::fromLocal8Bit(it->getpubday().dateFormat())));

    /* 为新增行添加更新按钮和复选框。*/

    btn.push_back(new QPushButton(QString::fromLocal8Bit("更新")));
    mark.push_back(new QCheckBox());

    connect(btn[i], &QPushButton::clicked, [this, i]()
{Admin_InfoDialog::save_or_change_for_TableWghit(i); });

    ui.tableWidget->setCellWidget(i, 8, btn[i]);

    ui.tableWidget->setCellWidget(i, 0, mark[i]);

}

END

```

## 2.9 学生信息添加算法

### 1. 函数

```
void Admin_InfoDialog::add_item_to_stu_TableWghit()
{
    /* 获取当前表格行数。*/
    int currentRowCount = this->ui.tableWidget_stu->rowCount();

    /* 增加一行。*/
    this->ui.tableWidget_stu->setRowCount(currentRowCount + 1);

    /* 为新行添加更新按钮和复选框。*/
    stu_btn.push_back(new QPushButton(QString::fromLocal8Bit("确认")));
    stu_mark.push_back(new QCheckBox());
    stu_mark[currentRowCount]->setStyleSheet("QRadioButton { max-width:20px; }");
    this->ui.tableWidget_stu->setCellWidget(currentRowCount, 6,
    stu_btn[currentRowCount]);

    this->ui.tableWidget->setCellWidget(currentRowCount, 0, stu_mark[currentRowCount]);

    /* 连接按钮点击事件处理函数。*/
    connect(stu_btn[currentRowCount], &QPushButton::clicked, [this, currentRowCount]()
    {Admin_InfoDialog::save_or_change_for_stu_TableWghit(currentRowCount); });

    /* 连接复选框点击事件处理函数。*/
    //connect(stu_mark[currentRowCount], &QCheckBox::clicked, [this, currentRowCount]()
    {Admin_InfoDialog::chosed_mark_for_TableWghit(currentRowCount); });
}

END
```

## 2.10 推荐书籍加载算法：

### 1. 函数

```
void cell_recommend::refreshTable()

// 刷新推荐书籍表格显示

BEGIN

    /* 遍历推荐列表，统计每本书的借阅次数。*/

    BorrowInfoSet r_borrowSet;

    std::map<std::string, int> booknoCount; // 存储每本书的借阅次数


    for (auto i : r_list->getSBList()) {

        r_borrowSet = i.getblist();


        int row;


        BorrowInfoSet::iterator it;


        // 统计每本书的借阅次数。*/

        for (it = r_borrowSet.begin(), row = 0; it != r_borrowSet.end(); it++, row++) {

            // 更新书号对应的借阅次数。*/

            booknoCount[it->getbookno()]++;

        }

    }


    // 将书号和借阅次数转换为 vector 存储，以便排序。*/

    std::vector<std::pair<std::string, int>> booknoCountVec(booknoCount.begin(),
booknoCount.end());


    // 对借阅次数进行降序排序。*/

    std::sort(booknoCountVec.begin(), booknoCountVec.end(), [](const std::pair<std::string,
```

```

int>& a, const std::pair<std::string, int>& b) {

    return a.second > b.second;

});

// 提取前 5 本推荐书籍的信息。*/
std::vector<std::pair<std::string, int>> top5Booknos;

if (booknoCountVec.size() > 5) {

    top5Booknos.assign(booknoCountVec.begin(), booknoCountVec.begin() + 5);

}

else {

    top5Booknos = booknoCountVec;

}

/* 创建新的推荐书籍集合并添加前 5 本书。*/
BookSet* recommendSet = new BookSet;

for (const auto& pair : top5Booknos) {

    // 检查书籍是否已在集合中，若不在则添加。*/
    for (auto it = b_set->begin(); it != b_set->end(); it++) {

        if (it->getbookno() == pair.first) {

            recommendSet->push_back(*it);

            break;

        }

    }

}

/* 遍历推荐书籍集合，为表格设置数据。*/
BookSet::iterator it;

int row;

ui.recommendTable->setRowCount(0);

ui.recommendTable->setRowCount(recommendSet->size());

```



```

int expectedRows = recommendSet->size();

for (it = recommendSet->begin(), row = 0; it != recommendSet->end(); it++, row++) {

    int col = 0;

    // 统计并显示书籍的推荐指数（星数）。*/

    int count = booknoCount[it->getbookno()];

    QString stars;

    if (count == 5) {

        stars = QString(QChar(0x2605)).repeated(5);

    }

    else if (count == 4) {

        stars = QString(QChar(0x2605)).repeated(4);

    }

    else if (count == 3) {

        stars = QString(QChar(0x2605)).repeated(3);

    }

    else if (count == 2) {

        stars = QString(QChar(0x2605)).repeated(2);

    }

    else if (count == 1) {

        stars = QString(QChar(0x2605));

    }

    else {

        stars = QString(QChar(0x2605)).repeated(5); // 如果超出 5 本书范围 默认
为 5 星

    }

    ui.recommendTable->setItem(row, col++, new
QTableWidgetItem(QString::fromLocal8Bit(it->getbookno())));

    ui.recommendTable->setItem(row, col++, new

```

```

QTableWidgetItem(QString::fromLocal8Bit(it->getbookname())));

                ui.recommendTable->setItem(row,                col+,                new
QTableWidgetItem(QString::fromLocal8Bit(it->getauthor())));

                ui.recommendTable->setItem(row,                col+,                new
QTableWidgetItem(QString::fromLocal8Bit(it->getpublisher())));

                ui.recommendTable->setItem(row, col+, new QTableWidgetItem(stars));

                ui.recommendTable->item(row, 0)->setTextAlignment(Qt::AlignCenter);
                ui.recommendTable->item(row, 1)->setTextAlignment(Qt::AlignCenter);
                ui.recommendTable->item(row, 2)->setTextAlignment(Qt::AlignCenter);
                ui.recommendTable->item(row, 3)->setTextAlignment(Qt::AlignCenter);
                ui.recommendTable->item(row, 4)->setTextAlignment(Qt::AlignCenter);

                for (int col = 0; col < 5; ++col) {

                    QTableWidgetItem* pItem = ui.recommendTable->item(row, col);

                    pItem->setFlags(pItem->flags() & (~Qt::ItemIsEditable));

                }

            }

END

```

## 四、程序的简单测试

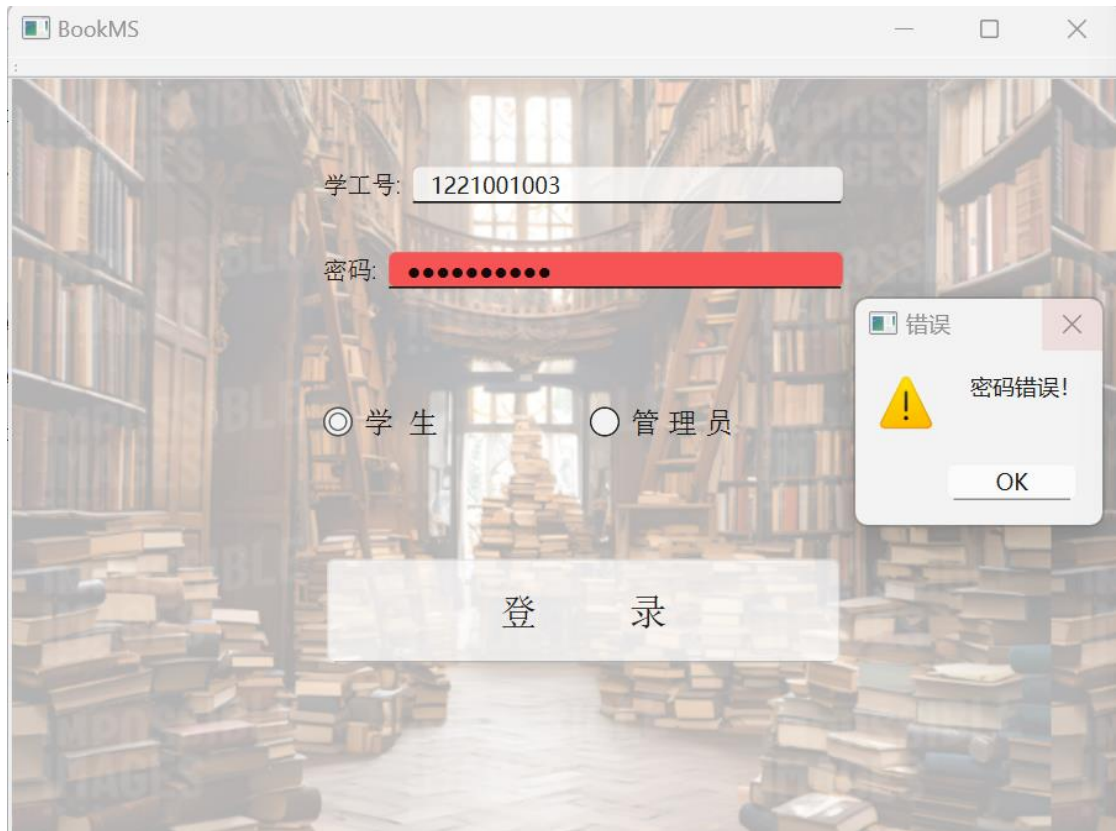
### 1. 操作界面测试

#### 1.1 登录界面测试

该界面是一个拥有学工号和密码文本编辑框控件的对话框，只有在选中登录类型后才会进入密码校验。

对该对话框在相应功能实现过程中操作正确性测试包括对学生和管理员学工号和密码校验的部分。

① 密码输入错误将有消息盒子提示，并将密码栏标红。



图片 37 密码错误提示

## 1.2 图书借阅界面测试

该界面是一个拥有可借阅图书信息表单控件的对话框，只有在图书有剩余存量，读者借阅图书数量未达到上限时才会进入完成图书借阅操作。

对该对话框在相应功能实现过程中操作正确性测试包括对读者上限和图书存量下限检验两部分。

- ①读者借阅数量达到上限将有消息盒子提示。



图片 38 图书借阅达到上限

②图书存量达到下限将有消息盒子提示。



图片 39 图书已全部借出

### 1.3 图书归还界面测试

该界面是一个拥有已借阅图书信息表单控件的对话框，图书均可正常归还，不可擅自修改图书表单。



图片 40 图书归还界面

## 1.4 读者个人信息界面测试

该界面是一个显示读者个人详细信息可编辑文本框（禁用）控件的对话框，读者不可擅自更改个人信息。



图片 41 读者个人信息界面

## 1.5 书籍推荐界面测试

该界面是一个显示书籍推荐信息表单控件的对话框，读者不可擅自更改推荐书籍信息。



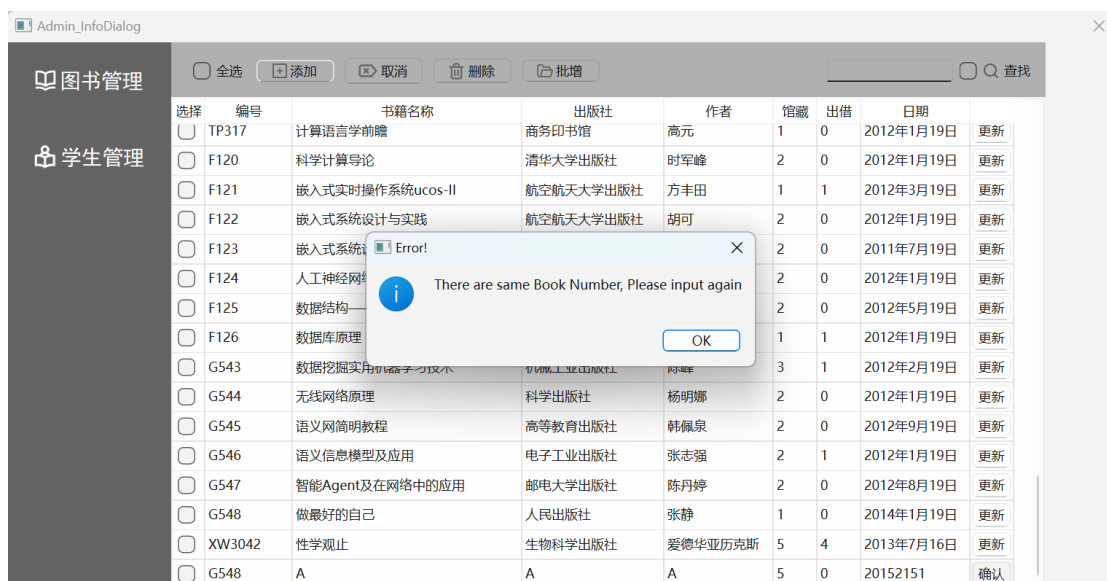
图片 42 书籍推荐信息

## 1.6 图书管理界面测试

该界面是一个拥有图书信息表单控件的对话框，在添加或修改图书信息时，会对多个模块进行安全性校验。

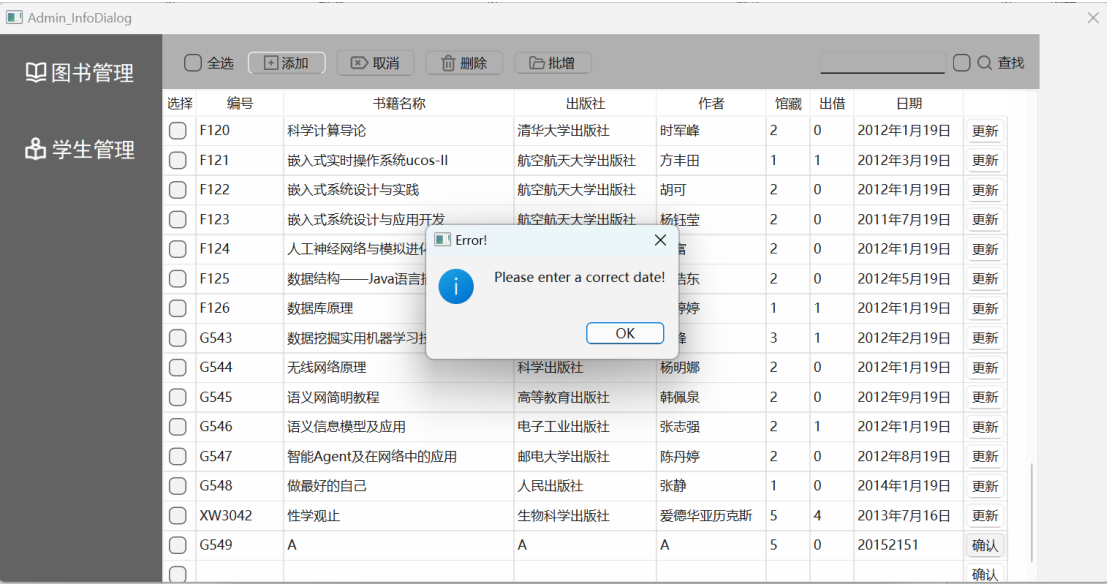
对该对话框在相应功能实现过程中操作正确性测试包括对重复书号和非法日期信息检验等部分。

①出现重复书号有消息盒子提示。



图片 43 出现重复信息

②出现非法日期信息有消息盒子提示。



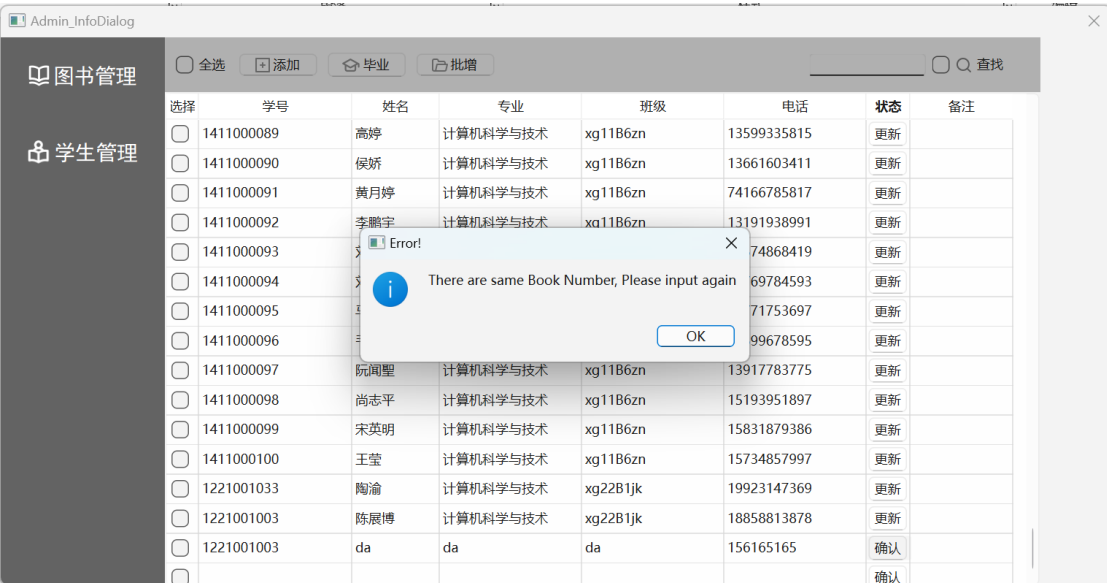
图片 44 非法日期信息出现

### 1.7 学生管理界面测试

该界面是一个拥有学生信息表单控件的对话框，在添加或修改图书信息时，会对多个模块进行安全性校验。

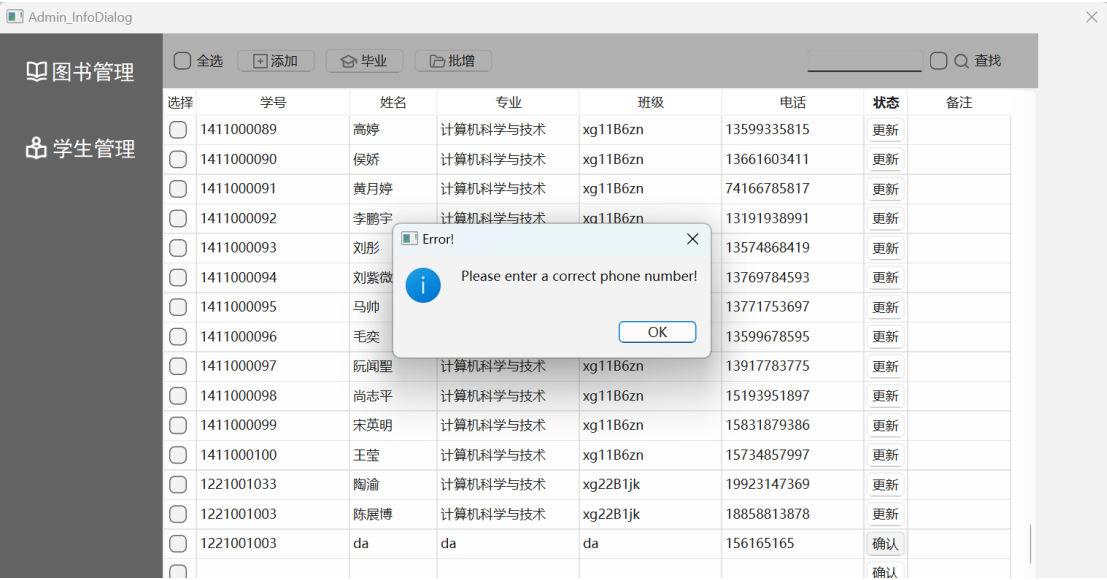
对该对话框在相应功能实现过程中操作正确性测试包括对重复学号和非法电话信息检验等部分。

①出现重复学号有消息盒子提示。



图片 45 重复学号出现

②出现非法电话信息有消息盒子提示。



图片 46 非 11 位非法电话出现

## 2.功能测试

### 2.1 信息文件装载

① 在信息文件存在，且文件中的人员信息记录不为 0 的情况下进行测试，则将该文件中全部信息顺序读入并添加到人员信息集合中，同时显示在表格部件。

② 在信息文件存在，但文件中记录为 0 的情况下进行测试。

上述操作在系统开始运行时自动完成，符合功能的设计要求。



## 2.2 信息添加

读者信息中学号是唯一区别不同读者的标志。通过添加具有唯一学号的读者信息和具有相同学号的读者信息进行测试，则前者被成功添加，而后者被拒绝添加。

图书信息中书号是唯一区别不同图书的标志。通过添加具有唯一书号的图书信息和具有相同书号的图书信息进行测试，则前者被成功添加，而后者被拒绝添加。

符合功能的设计要求。

## 2.3 信息文件保存

由于添加、删除、清空、修改、排序操作，信息集合中信息已经被改变，在系统退出时，在保存信息之前将原有的信息文件保存，以便下次运行程序正确显示信息。

# 3.内存泄露测试

在 Debug 状态下，借助插件 Visual LeakDetector，运行《图书管理系统程序》，在执行各功能操作后，正常退出系统运行。查看系统运行的内存泄漏状况信息（部分）如下：

Call Stack (TID 15012):

ucrtbased.dll!malloc()

D:\a\_work\1\s\src\vctools\crt\vcstartup\src\heap\new\_scalar.cpp (36):

QtWidgetsApplication1.exe!operator new() + 0xA bytes

F:\Junior1\QT\BookMS(v1.1.0)\QtWidgetsApplication1\QtWidgetsApplication1\Admin\_InfoDialog.cpp (27): QtWidgetsApplication1.exe!Admin\_InfoDialog::Admin\_InfoDialog() + 0xA bytes

F:\Junior1\QT\BookMS(v1.1.0)\QtWidgetsApplication1\QtWidgetsApplication1\BookMS.cpp (103): QtWidgetsApplication1.exe!BookMS::loginAdmin() + 0x3F bytes

F:\Qt\6.7.2\msvc2019\_64\include\QtCore\qobjectdefs\_impl.h (145):

QtWidgetsApplication1.exe!QtPrivate::FunctorCall<QtPrivate::IndexesList<>,QtPrivate::List<>,void,void (\_\_cdecl BookMS::\*)(void)>::call() + 0x32 bytes

```

F:\Qt\6.7.2\msvc2019_64\include\QtCore\qobjectdefs_impl.h (183):
QtWidgetsApplication1.exe!QtPrivate::FunctionPointer<void (__cdecl
BookMS::*)(void)>::call<QtPrivate::List<>,void>()

F:\Qt\6.7.2\msvc2019_64\include\QtCore\qobjectdefs_impl.h (556):
QtWidgetsApplication1.exe!QtPrivate::QCallableObject<void (__cdecl
BookMS::*)(void),QtPrivate::List<>,void>::impl()

Qt6Cored.dll!QFileInfo::filePath() + 0x1D7D27 bytes
Qt6Cored.dll!QFileInfo::filePath() + 0x29F7C4 bytes
Qt6Cored.dll!QFileInfo::filePath() + 0x285D4C bytes
Qt6Widgetsd.dll!QLineEdit::`default constructor closure'() + 0x339264 bytes
Qt6Widgetsd.dll!QLineEdit::`default constructor closure'() + 0x33B982 bytes
Qt6Widgetsd.dll!QLineEdit::`default constructor closure'() + 0x33A799 bytes
Qt6Widgetsd.dll!QLineEdit::`default constructor closure'() + 0x339CF1 bytes
Qt6Widgetsd.dll!QLineEdit::`default constructor closure'() + 0x11CA4D bytes
Qt6Widgetsd.dll!QLineEdit::`default constructor closure'() + 0x339509 bytes
Qt6Widgetsd.dll!QLineEdit::`default constructor closure'() + 0x50BFC4 bytes
Qt6Widgetsd.dll!QLineEdit::`default constructor closure'() + 0x36D86 bytes
Qt6Widgetsd.dll!QLineEdit::`default constructor closure'() + 0x31B24 bytes
Qt6Cored.dll!QFileInfo::filePath() + 0x1C1A1B bytes
Qt6Cored.dll!QFileInfo::filePath() + 0x1C18AF bytes
Qt6Widgetsd.dll!QLineEdit::`default constructor closure'() + 0x3A2F8 bytes
Qt6Widgetsd.dll!QLineEdit::`default constructor closure'() + 0x17870B bytes
Qt6Widgetsd.dll!QLineEdit::`default constructor closure'() + 0x17650E bytes
Qt6Widgetsd.dll!QLineEdit::`default constructor closure'() + 0x36D86 bytes
Qt6Widgetsd.dll!QLineEdit::`default constructor closure'() + 0x34196 bytes
Qt6Cored.dll!QFileInfo::filePath() + 0x1C1A1B bytes
Qt6Cored.dll!QFileInfo::filePath() + 0x1C18AF bytes
Qt6Guid.dll!QTextCharFormat::setFontLetterSpacingType() + 0x194165 bytes
Qt6Guid.dll!QTextCharFormat::setFontLetterSpacingType() + 0x197567 bytes
Qt6Guid.dll!QTextCharFormat::setFontLetterSpacingType() + 0x2BFD44 bytes

```

Qt6Guid.dll!QTextCharFormat::setFontLetterSpacingType() + 0x9657A2 bytes

Qt6Cored.dll!QFileInfo::filePath() + 0x623763 bytes

Qt6Guid.dll!QTextCharFormat::setFontLetterSpacingType() + 0x96574B bytes

Qt6Cored.dll!QFileInfo::filePath() + 0x1E87F0 bytes

Qt6Cored.dll!QFileInfo::filePath() + 0x1E8AF8 bytes

Qt6Cored.dll!QFileInfo::filePath() + 0x1BEB6E bytes

Qt6Guid.dll!QTextCharFormat::setFontLetterSpacingType() + 0x18ED3A bytes

Qt6Widgets.dll!QLineEdit::`default constructor closure'() + 0x30DE7 bytes

F:\Junior1\QT\BookMS(v1.1.0)\QtWidgetsApplication1\QtWidgetsApplication1\main.cpp (25):

QtWidgetsApplication1.exe!main() + 0x6 bytes

C:\Users\qt\work\qt\qtbase\src\entrypoint\qtenrypoint\_win.cpp (45):

QtWidgetsApplication1.exe!qtEntryPoint() + 0xE bytes

C:\Users\qt\work\qt\qtbase\src\entrypoint\qtenrypoint\_win.cpp (64):

QtWidgetsApplication1.exe!WinMain()

D:\a\_work\1\s\src\vctools\crt\vcstartup\src\startup\exe\_common.inl (107):

QtWidgetsApplication1.exe!invoke\_main()

D:\a\_work\1\s\src\vctools\crt\vcstartup\src\startup\exe\_common.inl (288):

QtWidgetsApplication1.exe!\_\_srt\_common\_main\_seh() + 0x5 bytes

D:\a\_work\1\s\src\vctools\crt\vcstartup\src\startup\exe\_common.inl (331):

QtWidgetsApplication1.exe!\_\_srt\_common\_main()

D:\a\_work\1\s\src\vctools\crt\vcstartup\src\startup\exe\_winmain.cpp (17):

QtWidgetsApplication1.exe!WinMainCRTStartup()

KERNEL32.DLL!BaseThreadInitThunk() + 0x1D bytes

ntdll.dll!RtlUserThreadStart() + 0x28 bytes

----- Block 460582 at 0x00000000EE20CBF0: 16 bytes -----

Leak Hash: 0x8D321E30, Count: 1, Total 16 bytes

Call Stack (TID 15012):

Data:

D0 47 20 EE    49 02 00 00    00 00 00 00    00 00 00 00    .G..l... .....

----- Block 462928 at 0x00000000EE20CD30: 16 bytes -----

Leak Hash: 0x1FFCB9F6, Count: 1, Total 16 bytes

Call Stack (TID 15012):

Data:

B0 49 20 EE	49 02 00 00	00 00 00 00	00 00 00 00	.I..I... .....
-------------	-------------	-------------	-------------	----------------

----- Block 457123 at 0x00000000EE20CD80: 16 bytes -----

Leak Hash: 0x1FFCB9F6, Count: 1, Total 16 bytes

Call Stack (TID 15012):

Data:

C8 45 20 EE	49 02 00 00	00 00 00 00	00 00 00 00	.E..I... .....
-------------	-------------	-------------	-------------	----------------

----- Block 458270 at 0x00000000EE20CDD0: 16 bytes -----

Leak Hash: 0x1FFCB9F6, Count: 1, Total 16 bytes

Call Stack (TID 15012):

Data:

90 46 20 EE	49 02 00 00	00 00 00 00	00 00 00 00	.F..I... .....
-------------	-------------	-------------	-------------	----------------

----- Block 460585 at 0x00000000EE20CE20: 16 bytes -----

Leak Hash: 0x1FFCB9F6, Count: 1, Total 16 bytes

Call Stack (TID 15012):

Data:

20 48 20 EE	49 02 00 00	00 00 00 00	00 00 00 00	.H..I... .....
-------------	-------------	-------------	-------------	----------------

----- Block 459423 at 0x00000000EE20CE70: 16 bytes -----

Leak Hash: 0x4311F827, Count: 1, Total 16 bytes

Call Stack (TID 15012):

Data:

30 47 20 EE    49 02 00 00    00 00 00 00    00 00 00 00    0G..l... .....

----- Block 459420 at 0x00000000EE20CEC0: 16 bytes -----

Leak Hash: 0x8ED4738A, Count: 1, Total 16 bytes

Call Stack (TID 15012):

Data:

E0 46 20 EE    49 02 00 00    00 00 00 00    00 00 00 00    .F..l... .....

----- Block 461748 at 0x00000000EE20CF10: 16 bytes -----

Leak Hash: 0xECA7040D, Count: 1, Total 16 bytes

Call Stack (TID 15012):

Data:

48 48 20 EE    49 02 00 00    00 00 00 00    00 00 00 00    HH..l... .....

----- Block 461750 at 0x00000000EE20CF60: 16 bytes -----

Leak Hash: 0x8D321E30, Count: 1, Total 16 bytes

Call Stack (TID 15012):

Data:

98 48 20 EE    49 02 00 00    00 00 00 00    00 00 00 00    .H..l... .....

----- Block 461753 at 0x00000000EE20CFB0: 16 bytes -----

Leak Hash: 0x1FFCB9F6, Count: 1, Total 16 bytes

Call Stack (TID 15012):

Data:

E8 48 20 EE      49 02 00 00      00 00 00 00      00 00 00 00      .H..l... .....

----- Block 462924 at 0x00000000EE20D000: 16 bytes -----

Leak Hash: 0x8ED4738A, Count: 1, Total 16 bytes

Call Stack (TID 15012):

Data:

38 49 20 EE      49 02 00 00      00 00 00 00      00 00 00 00      8l..l... .....

----- Block 470541 at 0x00000000F7BF2F00: 16 bytes -----

Leak Hash: 0x63BDDCC0, Count: 1, Total 16 bytes

Call Stack (TID 15012):

Data:

B8 5B 74 ED      49 02 00 00      00 00 00 00      00 00 00 00      .[t..l... .....

----- Block 470542 at 0x00000000F7BF2FF0: 16 bytes -----

Leak Hash: 0xCD64ABD9, Count: 1, Total 16 bytes

Call Stack (TID 15012):

Data:

E0 5B 74 ED      49 02 00 00      00 00 00 00      00 00 00 00      .[t..l... .....

----- Block 470543 at 0x00000000F7BF3810: 16 bytes -----

Leak Hash: 0x35959408, Count: 1, Total 16 bytes

Call Stack (TID 15012):

Data:

08 5C 74 ED      49 02 00 00      00 00 00 00      00 00 00 00      .\t.l... .....

----- Block 470540 at 0x00000000F7BF3EF0: 16 bytes -----

Leak Hash: 0x4BA743AA, Count: 1, Total 16 bytes

Call Stack (TID 15012):

Data:

90 5B 74 ED      49 02 00 00      00 00 00 00      00 00 00 00      .[t.l... .....

----- Block 468274 at 0x00000000F7BF58E0: 16 bytes -----

Leak Hash: 0xE9AC0E3F, Count: 1, Total 16 bytes

Call Stack (TID 15012):

Data:

E0 59 74 ED      49 02 00 00      00 00 00 00      00 00 00 00      .Yt.l... .....

----- Block 468272 at 0x00000000F7BF5B10: 16 bytes -----

Leak Hash: 0xC33F77DC, Count: 1, Total 16 bytes

Call Stack (TID 15012):

Data:

90 59 74 ED      49 02 00 00      00 00 00 00      00 00 00 00      .Yt.l... .....

----- Block 468273 at 0x00000000F7BF60B0: 16 bytes -----

Leak Hash: 0x3BCE480D, Count: 1, Total 16 bytes

Call Stack (TID 15012):

Data:

B8 59 74 ED      49 02 00 00      00 00 00 00      00 00 00 00      .Yt.l... .....

----- Block 468275 at 0x00000000F7BF6470: 16 bytes -----

Leak Hash: 0x115D31EE, Count: 1, Total 16 bytes

Call Stack (TID 15012):

Data:

08 5A 74 ED    49 02 00 00    00 00 00 00    00 00 00 00    .Zt.l... .....

----- Block 472673 at 0x00000000F7E8A990: 32 bytes -----

Leak Hash: 0x687787B6, Count: 1, Total 32 bytes

Call Stack (TID 15012):

Data:

BC C6 CB E3    BB FA BF C6    D1 A7 D3 EB    BC BC CA F5    .....  
00 CD CD CD    CD CD CD CD    CD CD CD CD    CD CD CD CD    .....

Visual Leak Detector detected 1304 memory leaks (151086 bytes).

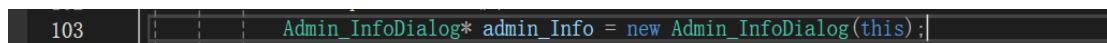
Largest number used: 687617 bytes.

Total allocations: 34221675 bytes.

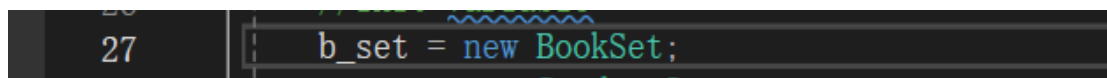
Visual Leak Detector is now exiting.

程序 “[22540] QtWidgetsApplication1.exe” 已退出，返回值为 0 (0x0)。

检测到 Admin\_InfoDialog.cpp (27)和 BookMS.cpp (103)存在内存泄漏情况。



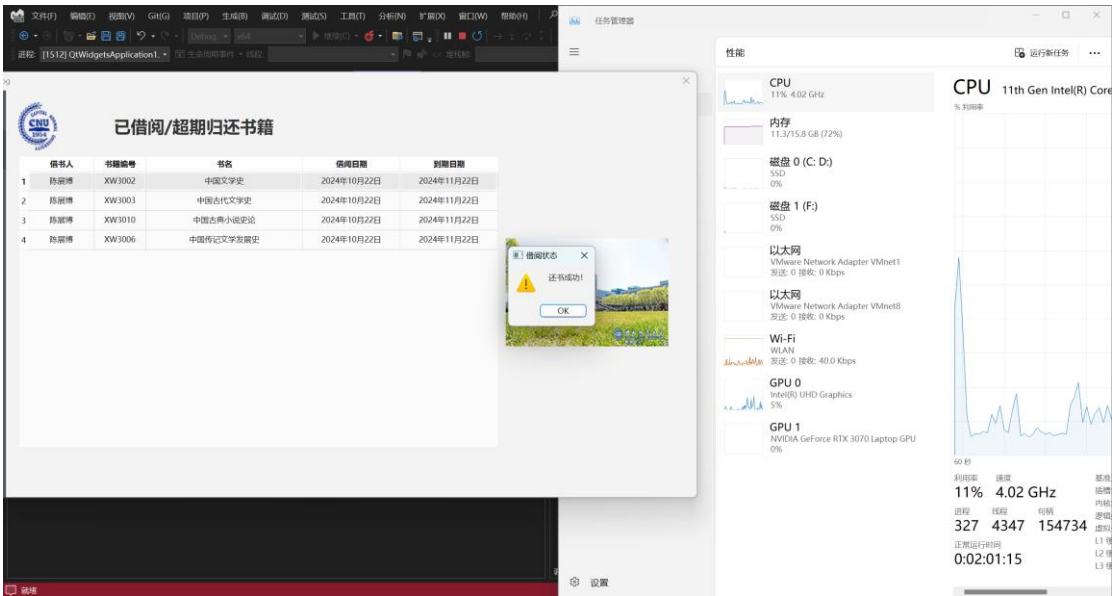
图片 47 内存泄漏 BookMS.cpp (103)



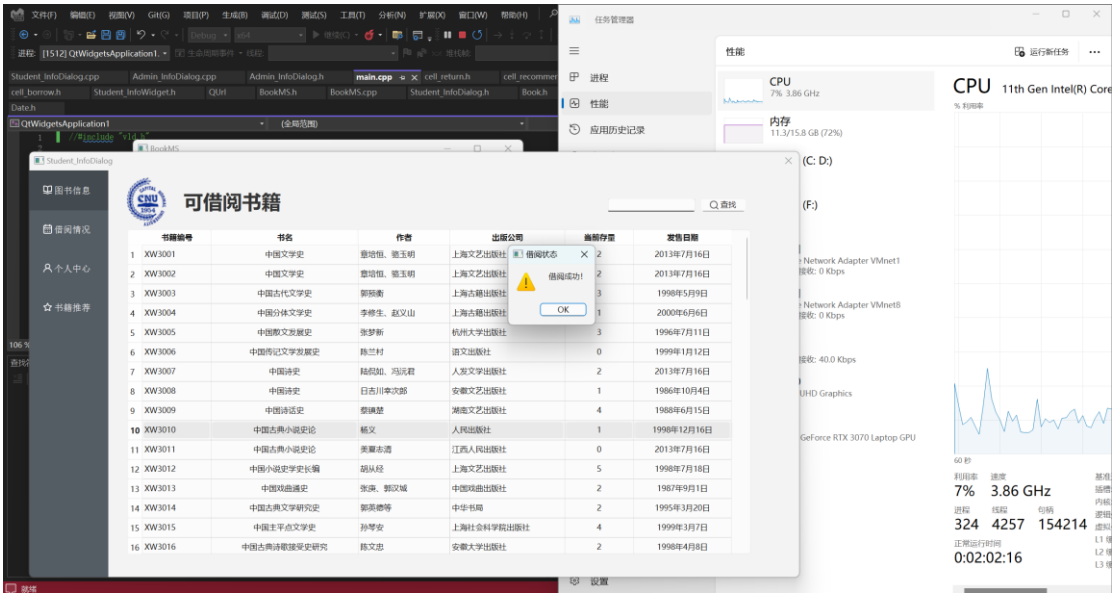
图片 48 内存泄漏 Admin\_InfoDialog.cpp (27)



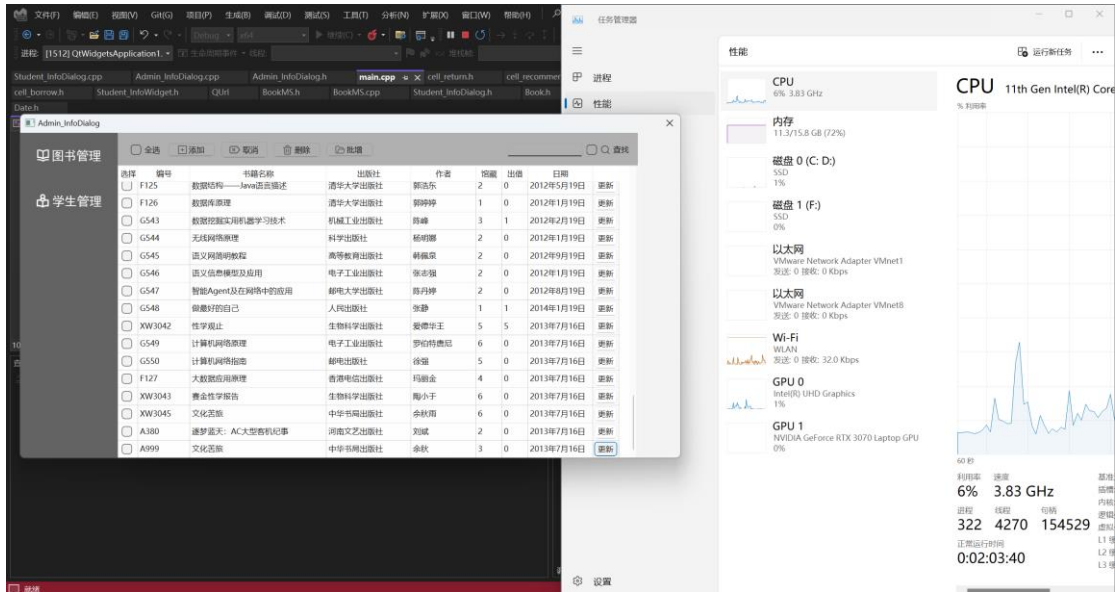
4.cpu 占用测试



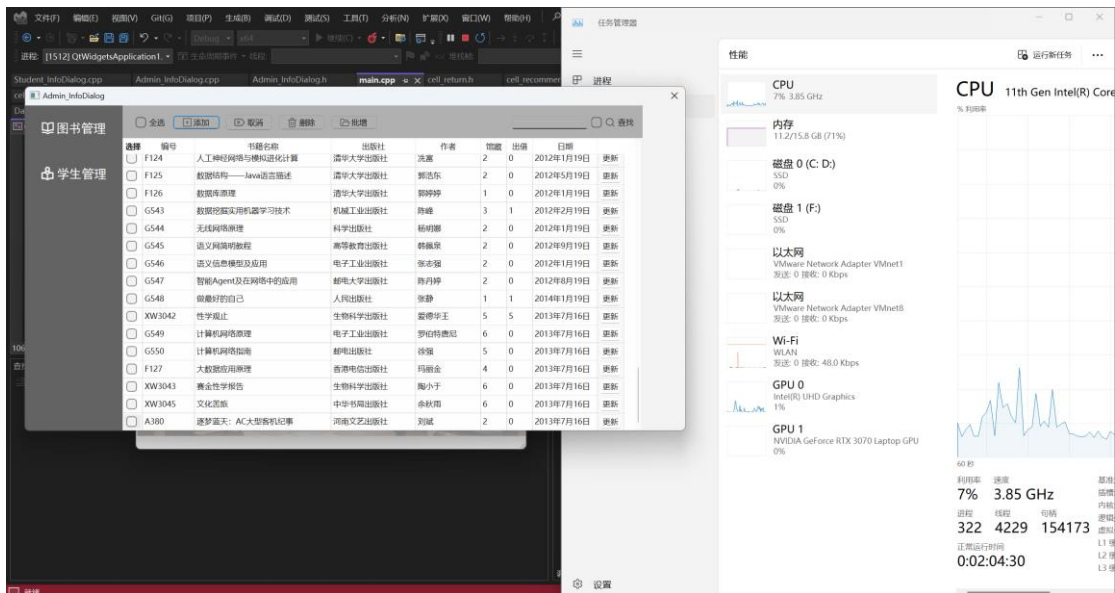
图片 49 还书 CPU 占用测试



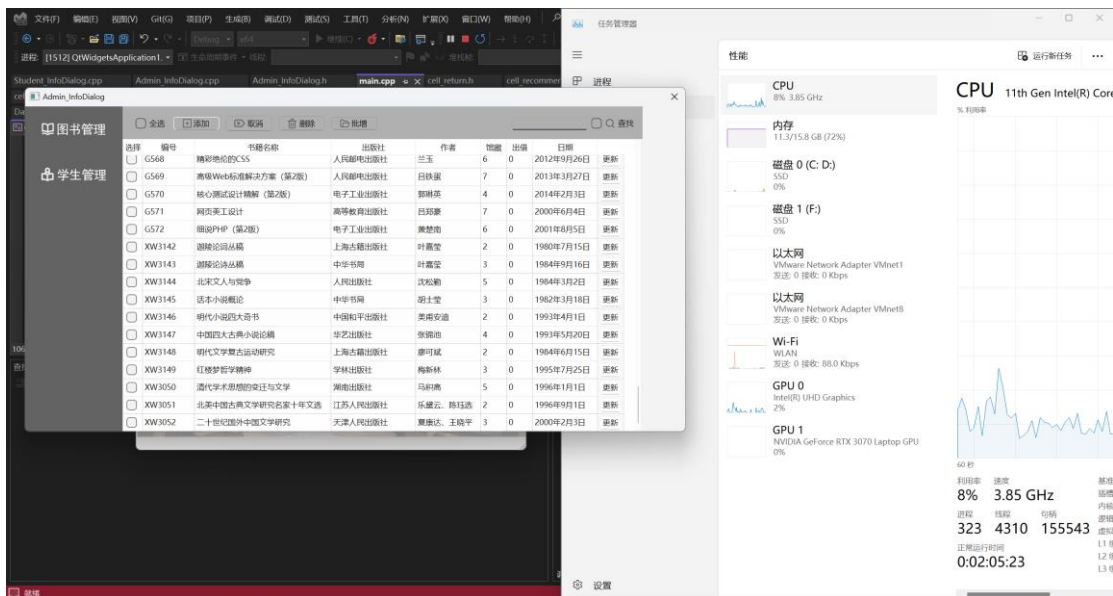
图片 50 借书 CPU 占用测试



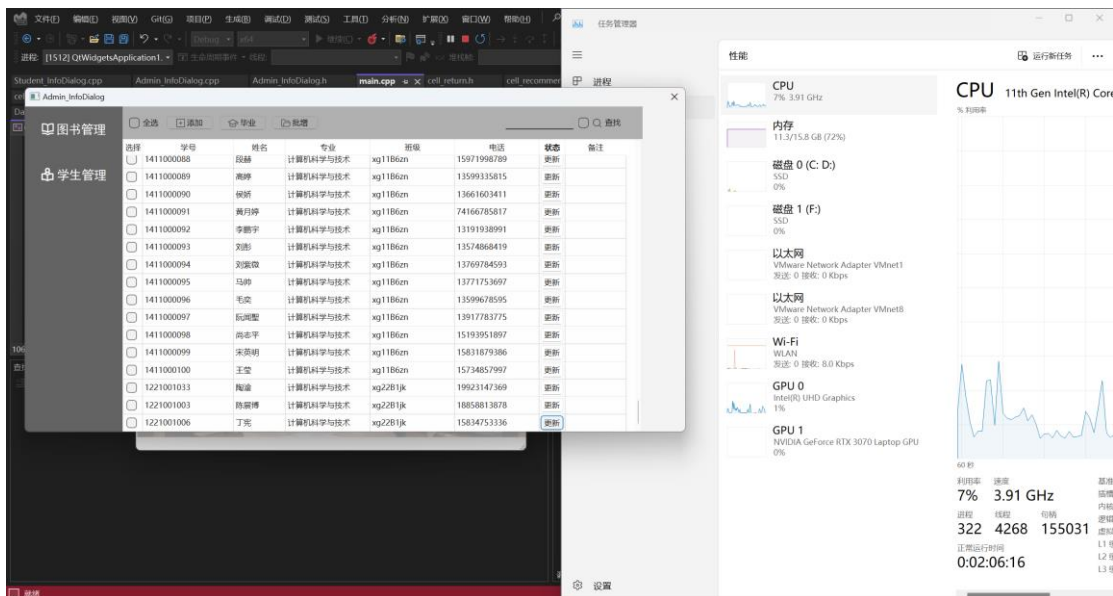
图片 51 增添书籍 CPU 占用测试



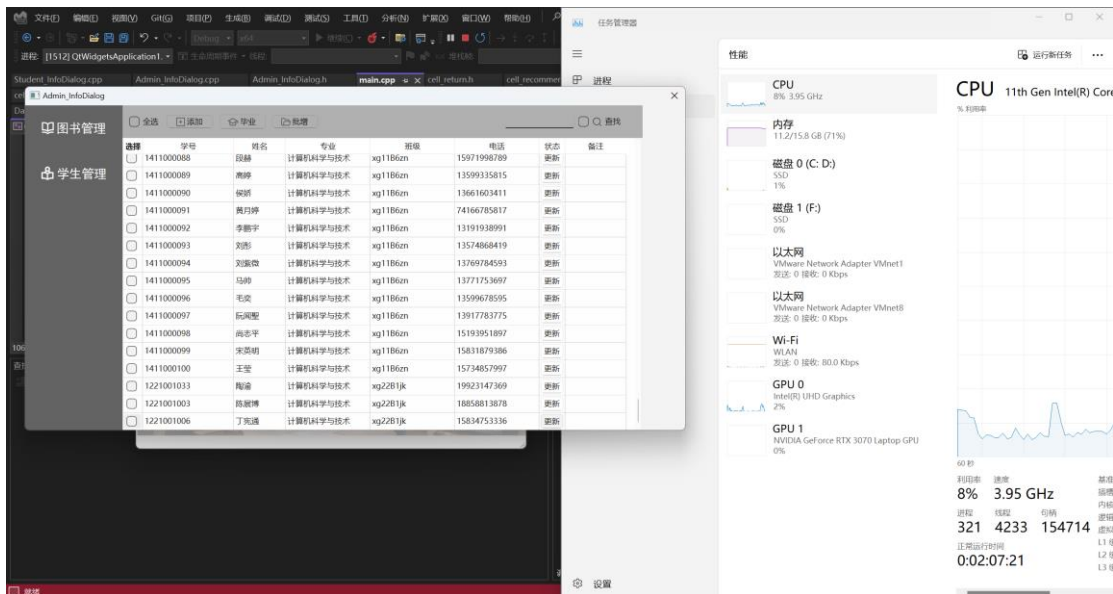
图片 52 删除书籍 CPU 占用测试



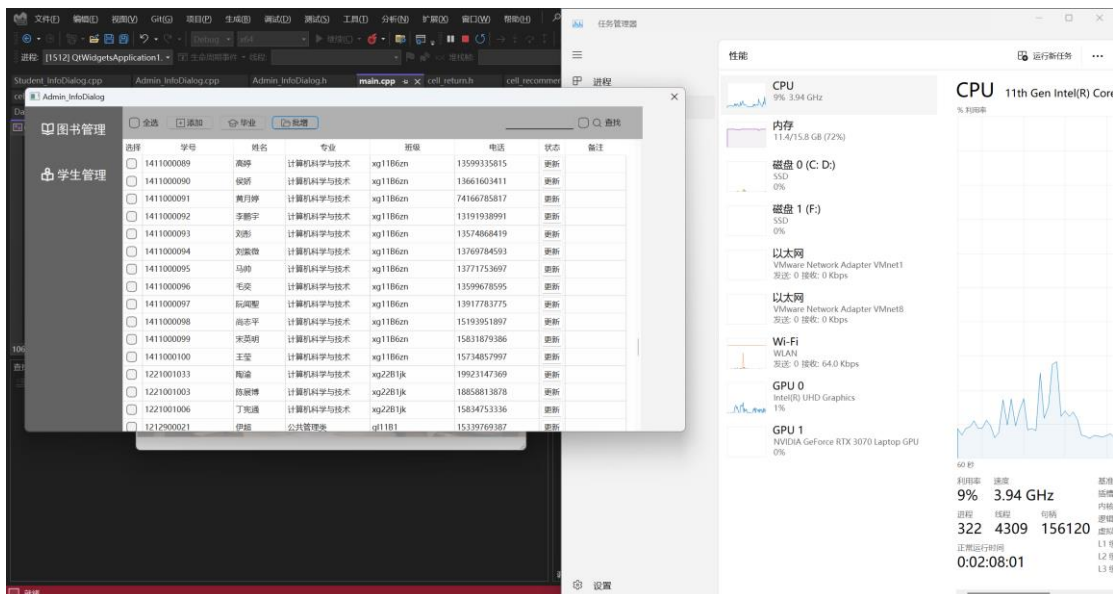
图片 53 批量增添书籍 CPU 占用测试



图片 54 修改读者信息 CPU 占用测试



图片 55 毕业读者操作 CPU 占用测试



图片 56 批量增添读者 CPU 占用测试

经测试，程序在运行时不存在 CPU 占用异常的情况。