

图书管理系统

系统需求分析

目的

- 开发一个功能完善、操作简单、界面友好的图书管理系统，实现图书和用户的增删查改、借阅、归还、记录查询、热门图书和活跃用户统计等功能，提高图书馆的管理效率和服务质量。

系统需求

图书管理

- 新建图书**
 - 输入图书信息，包括名称、作者、分类、关键词、简介。
- 修改图书信息**
 - 输入需要修改的图书名称，显示图书信息，输入修改后的信息。
- 删除图书**
 - 输入需要删除的图书名称，确认删除。
- 模糊查询图书**
 - 输入搜索词，显示查询结果。

用户管理

- 新建用户**
 - 输入用户信息，包括用户名。
- 修改用户信息**
 - 输入需要修改的用户名，输入新用户名。
- 删除用户**
 - 输入需要删除的用户名，确认删除。
- 模糊查询用户**
 - 输入搜索词，显示查询结果。

借阅管理

- 图书借阅**
 - 输入图书名称、用户名，确认借阅。

- 图书归还

- 输入图书名称、用户名，确认归还。

- 借阅信息查询

- 输入用户姓名，显示借阅信息。

数据管理

- 存储图书数据

- 每本书用一个 txt 文件存储，包括名称、作者、分类、关键词、简介、借阅状态、借阅次数。

- 存储用户数据

- 每个用户用一个 txt 文件存储，包括用户名、借阅记录（借阅图书、借阅时间、归还时间、是否已归还）。

- 查看十大热门图书

- 统计图书借阅次数，获取借阅次数最多的十本书。

- 查看十大活跃用户

- 统计用户借阅次数，获取借阅次数最多的十个用户。

- 清空数据

图形化界面

- 提供操作提示和帮助信息，支持中文显示。

输入输出

- 输入验证

- 对用户输入进行验证，确保输入合法。包括非空验证、格式验证、范围验证。

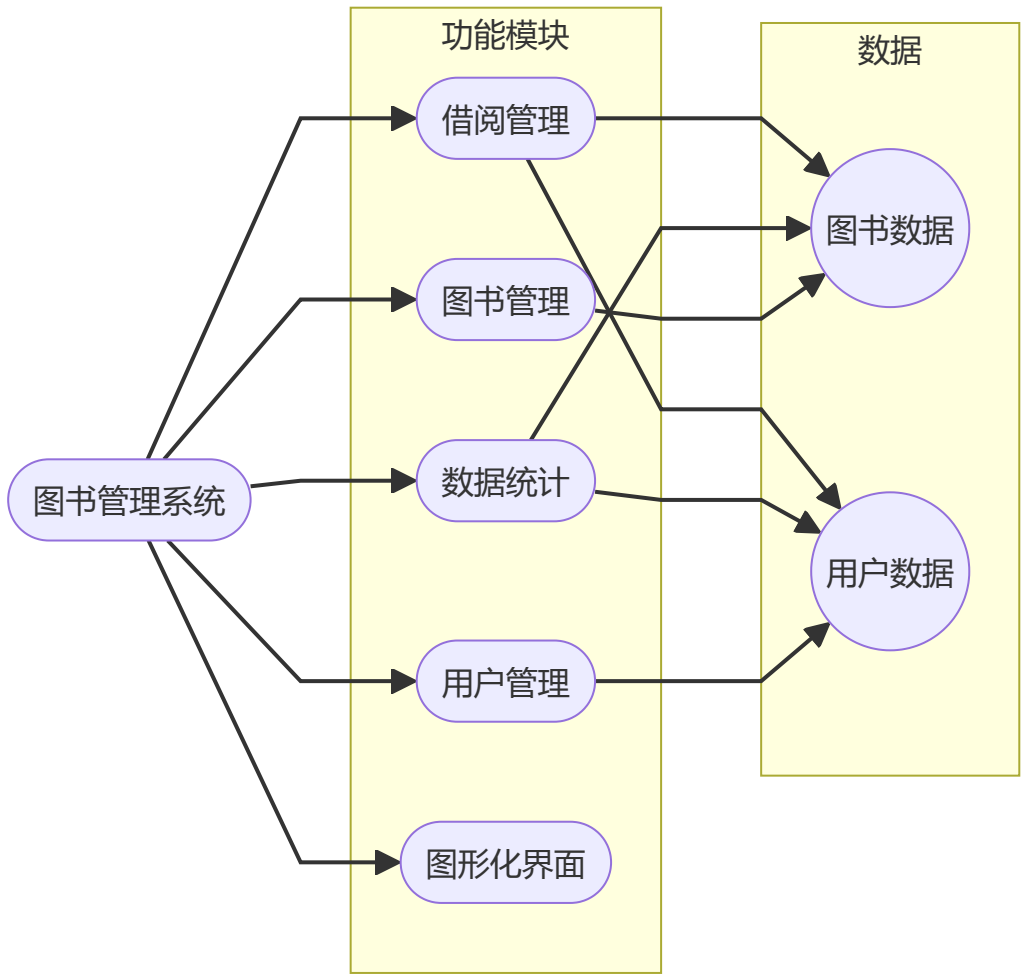
- 输出显示

- 显示操作结果，包括提示、成功、失败、错误信息。

总体设计

图书管理系统包含四个主要功能，分别为图书增删改查、用户增删改查、借阅归还、数据统计。提示用户输入操作编号，根据用户输入调用相应的功能模块，完成相应的操作。

系统功能设计及模块图



详细设计

数据文件

本系统采用文本文件存储图书和用户数据，便于修改和备份。文件名和数据格式遵循一定的规范，确保数据的准确性和易读性。

文件结构

- **图书数据**
 - 存储在 `./data/book/` 目录下，文件名与图书名称相同（自动转换为 `GBK` 编码），扩展名为 `.txt`。
- **用户数据**
 - 存储在 `./data/user/` 目录下，文件名与用户名相同（自动转换为 `GBK` 编码），扩展名为 `.txt`。

数据格式

- **图书数据**
 - 每条数据占用一行。
 - 字段顺序为：标题、作者、分类、关键词、简介、借出状态（1 表示已借出，0 表示未借出）、借出次数。
 - 字段自动转换为 `UTF-8` 编码。

- 用户数据

- 每条数据占用一行。
- 字段顺序为：书名、借书时间、还书时间、是否已归还（1 表示已归还，0 表示未归还）。
- 字段自动转换为 UTF-8 编码。

示例

- 图书数据文件

文件名: `./data/book/三国演义.txt`

```
1 | 三国演义
2 | 罗贯中
3 | 历史小说
4 | 三国
5 | 汉末群雄逐鹿中原
6 | 0
7 | 50
```

- 用户数据文件

文件名: `./data/user/张三.txt`

```
1 | 三国演义
2 | 2023-04-01 15:18:24
3 | 2023-04-10 15:18:24
4 | 1
5 | 西游记
6 | 2023-04-10 15:28:24
7 |
8 | 0
```

类的层次

顶层类 Library

- 这是程序的主类，负责启动程序并调用 GUI 类来展示用户界面和执行操作。

GUI

- 继承自 Manager 类，提供了用户界面和用户交互功能。
- 实现了 Manager 类的所有方法，并提供了一些额外的方法来展示菜单、处理用户输入等。

Manager

- 继承自 BookManager 和 UserManager 类，整合了图书和用户的管理功能。
- 提供了获取当前日期时间、借书、还书等方法。

BookManager

- 负责图书的管理，包括添加、删除、查找、编辑等操作。
- 提供了获取图书、搜索图书、获取热门图书等方法。

UserManager

- 负责用户的管理，包括添加、删除、查找、编辑等操作。
- 提供了获取用户、搜索用户、获取活跃用户等方法。

Base

- 定义了所有图书和用户类共有的接口，包括添加、保存、删除、编辑等操作。

Book

- 继承自 Base 类，定义了图书的信息，包括书名、作者、分类、关键词、简介、借出次数和借出状态。
- 提供了图书的添加、保存、删除、编辑等方法。

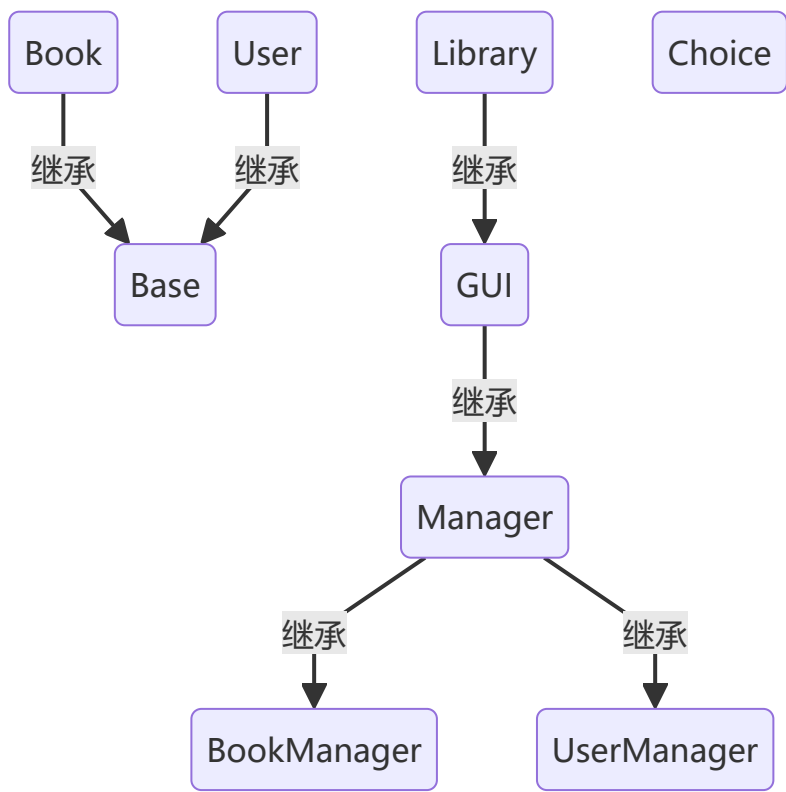
User

- 继承自 Base 类，定义了用户的信息，包括姓名、借阅记录和借阅次数。
- 提供了用户的添加、保存、删除、编辑等方法。

Record

- 定义了借阅记录的信息，包括图书名称、借书时间、还书时间和还书状态。

类的关系图



界面设计和各功能模块实现

界面设计

该图书管理系统采用简单的命令行界面设计，主要特点如下：

- **菜单显示**
 - 标题：界面顶部显示“图书管理系统”标题，清晰明了。
 - 菜单项：每个菜单项以编号和文字描述的方式呈现，方便用户选择。
 - 分隔线：使用分隔线将菜单项与标题和底部提示信息隔开，使界面更加清晰。
- **提示信息**
 - 操作提示：每个功能模块开始时，都会显示相应的提示信息，说明操作流程。
 - 输入提示：每次需要用户输入信息时，都会显示输入提示，例如：请输入书名、请输入用户名等。
 - 结果提示：每次操作完成后，都会显示结果提示，例如：保存成功、删除失败等。
- **响应方式**
 - 用户通过输入数字选择菜单项，系统根据用户输入执行相应的操作。
 - 用户输入完成后，按回车键确认，系统开始执行操作。
- **优点**
 - 简单易用：界面简洁明了，操作流程清晰，方便用户快速上手。
 - 成本低：无需安装额外的软件，即可运行系统。

- 跨平台：可在 Windows、Linux 等操作系统上运行。

主要界面如下

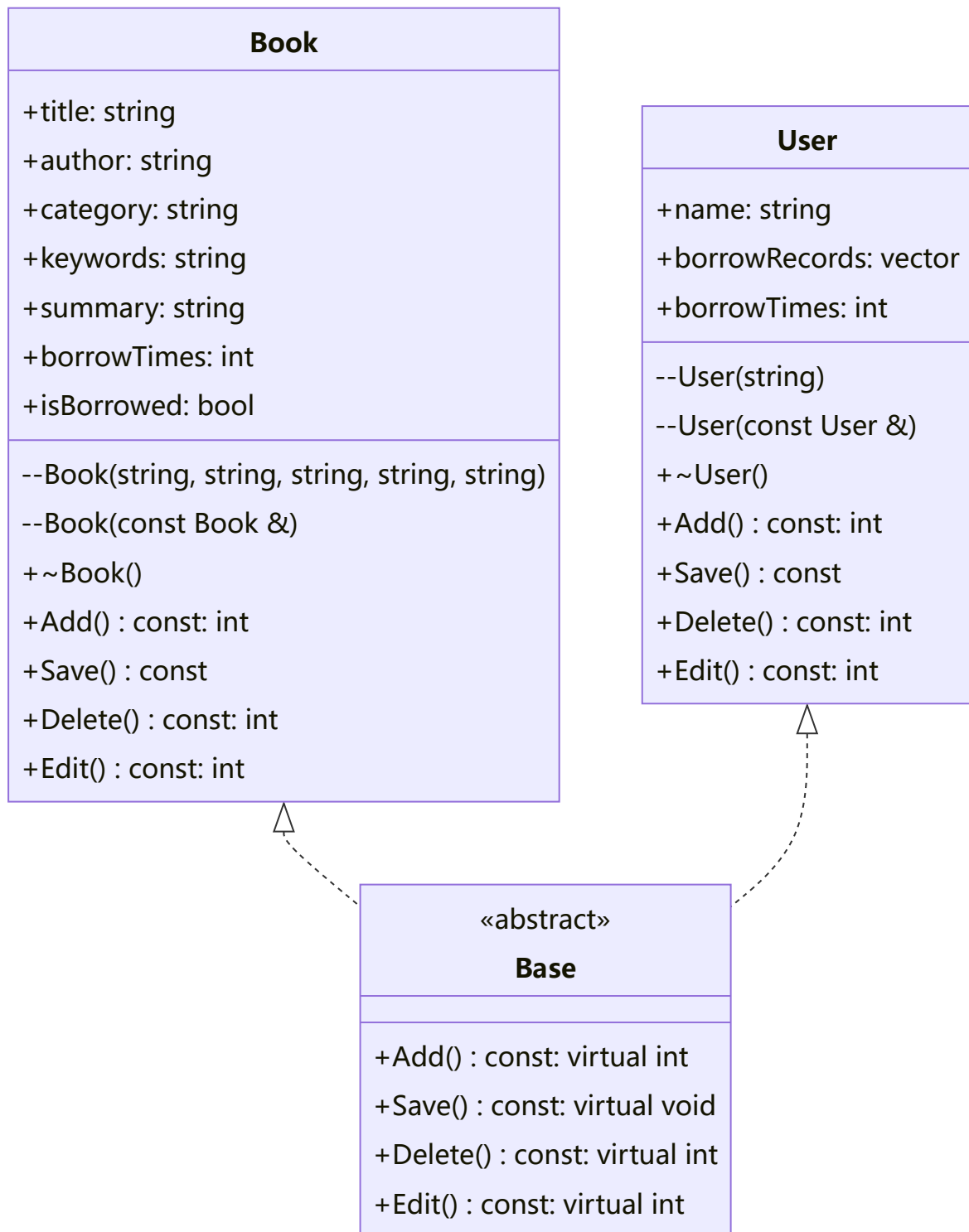
- 主菜单

```
1      图书管理系统
2      -----
3      1. 添加图书          9. 图书借阅
4      2. 删除图书          10. 图书归还
5      3. 查找图书          11. 借阅记录
6      4. 编辑图书          12. 十大热门图书
7      5. 添加用户          13. 十大活跃用户
8      6. 删除用户          14. 删除所有图书
9      7. 查找用户          15. 删除所有用户
10     8. 编辑用户          16. 退出
11     -----
12
13     请选择操作:
```

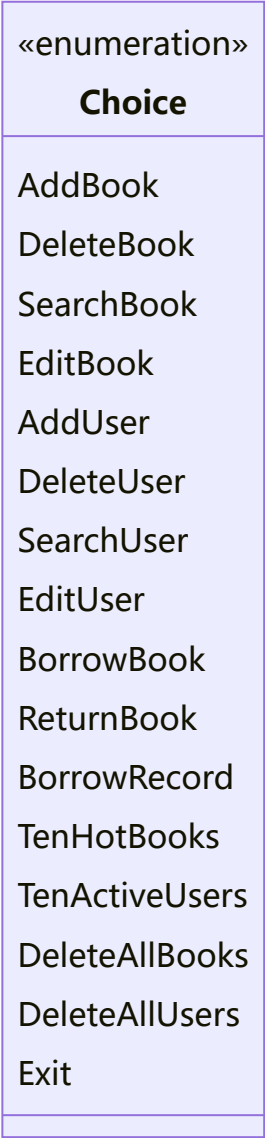
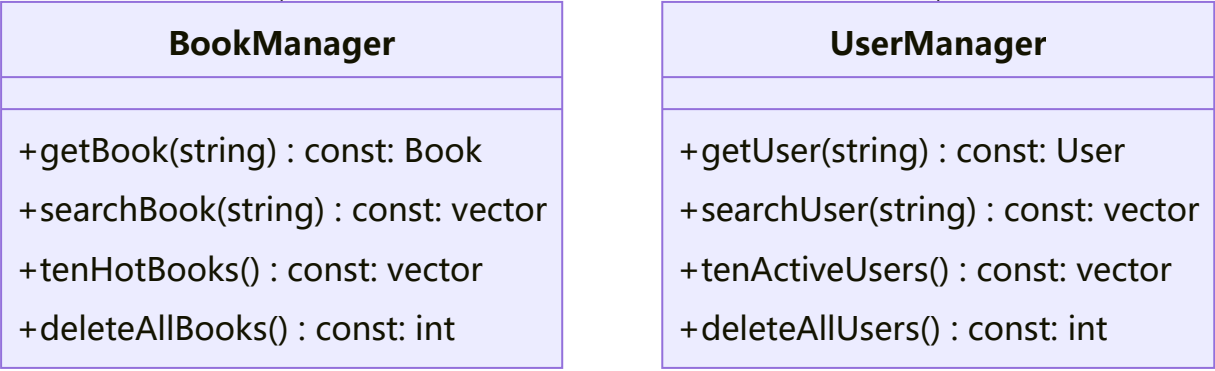
功能模块实现

base 类定义了所有图书和用户类共有的接口，包括添加、保存、删除、编辑等操作。Book 和 User 类继承自 Base 类，分别定义了图书和用户的信息和操作。BookManager 和 UserManager 分别定义了图书和用户的管理操作。Manager 类继承自 BookManager 和 UserManager 类，整合了图书和用户的管理功能。GUI 类继承自 Manager 类，提供了用户界面和用户交互功能。

UML类图







系统调试

字符串编码问题

中文版 Windows 系统默认使用 GBK 编码，因此终端和文件系统默认都是 GBK 编码，而 txt 文件中的默认数据是 UTF-8 编码，因此需要在读写文件时进行编码转换。

1. 不同 C++ 编译器可能使用不同编码，为了统一为 UTF-8 编码，需要在文件头部添加 #pragma execution_character_set("utf-8") ，或者在编译选项中添加 -fexec-charset=UTF-8 。

2. 在 Windows 系统下，终端默认使用 `GBK` 编码，使用 `chcp 65001` 命令切换终端编码为 `UTF-8`，否则终端无法正确显示中文，在 C++ 程序中使用 `system("chcp 65001")` 命令切换终端编码。
3. 使用 `iconv.h` 库进行编码转换，将 `UTF-8` 编码转换为 `GBK` 编码，或者将 `GBK` 编码转换为 `UTF-8` 编码，确保文件路径和文件内容的正确读写。

选项输入验证

用户在输入选项编号时，可能输入非数字字符或者超出范围的数字，因此需要对用户输入进行验证，确保输入合法。

1. `getline()` 函数读取用户输入。
2. 使用 `all_of(input.begin(), input.end(), ::isdigit)` 函数判断用户输入是否全为数字。
3. 使用 `stoi()` 函数将字符串转换为整数，判断是否在合法范围内。

模糊查询

用户在查找图书或用户时，可能输入部分关键词，需要对输入的搜索词进行模糊查询，找到包含搜索词的所有结果。

1. 使用 `find()` 函数查找包含搜索词的所有结果。
2. 将结果保存到一个 `vector` 容器中，便于显示。

日期时间处理

在借阅图书和归还图书时，需要记录借书时间和还书时间，因此需要获取当前日期时间。

1. 使用 `time.h` 库中的 `time()` 函数获取当前时间戳。
2. 使用 `localtime()` 函数将时间戳转换为 `tm` 结构体，获取年月日时分秒。
3. 使用 `strftime()` 函数将 `tm` 结构体转换为字符串，格式化日期时间。

文件夹不存在问题

在读写文件时，可能会遇到文件夹不存在的问题，因此需要在程序中检查文件夹是否存在，如果不存在则创建文件夹。

1. 使用 `filesystem::exists()` 函数检查文件夹是否存在。
2. 使用 `filesystem::create_directory()` 函数创建文件夹。

结果分析

数据生成

由于本系统是一个图书管理系统，需要大量的图书和用户数据，因此需要生成一定数量的数据，以便测试系统的性能和稳定性。由于使用txt文件存储数据，因此可以手动编写数据文件。测试文件下载地址：[图书管理系统测试数据](#)。

功能测试

1. 添加图书

1	添加图书
2	
3	请输入书名：梦回大明
4	请输入作者：二月河
5	请输入分类：历史小说
6	请输入关键词：明朝、宫廷、权谋
7	请输入简介：本书以明朝为背景，讲述了一段波澜壮阔的宫廷斗争故事。
8	
9	保存成功
10	
11	按任意键返回

1	添加图书
2	
3	
4	请输入书名：1984
5	图书已存在
6	
7	按任意键返回

2. 删除图书（用户）

1	删除图书
2	
3	请输入书名：梦回大明
4	
5	确认删除？（y/n）y
6	
7	删除成功
8	
9	按任意键返回

3. 查找图书

```
1  查找图书
2
3
4  请输入搜索词（回车显示所有图书）： 世界
5
6  查询结果
7
8  图书1
9  书名：平凡的世界
10 作者：路遥
11 分类：现代文学
12 关键词：奋斗、生活、命运
13 简介：讲述了主人公孙少平在平凡的世界里，努力拼搏、追求幸福的故事。
14 借出状态：未借出
15 借出次数： 9
16
17 图书2
18 书名：苏菲的世界
19 作者：乔斯坦·贾德
20 分类：哲学读物
21 关键词：哲学、思考、人生
22 简介：以小说的形式，向读者介绍了西方哲学史上的主要流派和思想。
23 借出状态：未借出
24 借出次数： 44
25
26
27 按任意键返回
```

4. 编辑图书

```
1  编辑图书
2
3
4  请输入书名：1984
5
6  书名：1984
7  作者：George Orwell
8  分类：Dystopian Fiction
9  关键词：Totalitarianism, Surveillance, Reality Control
10 简介：A dystopian novel depicting a society under constant surveillance and ruled by a
    totalitarian regime.
11 借出状态：未借出
12 借出次数： 94
13
14 请输入新书名：1948
15 请输入新作者：Orwell George
16 请输入新分类：Fiction Dystopian
```

17	请输入新关键词: Reality Control
18	请输入新简介: A dystopian novel
19	
20	保存成功
21	
22	按任意键返回

5. 添加用户

1	添加用户
2	
3	
4	请输入用户名: 张三
5	
6	保存成功
7	
8	按任意键返回

6. 查找用户

1	查找用户
2	
3	请输入用户名 (回车显示所有用户): 张
4	
5	查询结果
6	
7	用户1: 张三
8	
9	用户2: 张飞
10	
11	按任意键返回

7. 编辑用户

1	编辑用户
2	
3	请输入用户名: 张三
4	
5	请输入新用户名: 李四
6	
7	保存成功
8	
9	按任意键返回

8. 图书借阅 (归还)

1

图书借阅

2

3

请输入书名：平凡的世界

4

5

请输入用户名：李四

6

7

图书借阅成功

8

9

按任意键返回

9. 借阅记录

1

借阅记录

2

3

请输入用户名：李四

4

5

借阅次数：6

6

7

借阅记录：

8

9

书名：平凡的世界

10

借书时间：2024-07-13 23:57:37

11

还书时间：2024-07-13 23:58:15

12

13

书名：百年孤独

14

借书时间：2024-07-13 23:59:40

15

还书时间：2024-07-14 00:00:35

16

17

书名：哈利·波特

18

借书时间：2024-07-13 23:59:56

19

还书时间：未还

20

21

书名：红楼梦

22

借书时间：2024-07-14 00:00:02

23

还书时间：未还

24

25

书名：活着

26

借书时间：2024-07-14 00:00:09

27

还书时间：未还

28

29

书名：三国演义

30

借书时间：2024-07-14 00:00:16

31

还书时间：2024-07-14 00:01:03

32

33

按任意键返回

10. 十大热门图书

1	十大热门图书
2	
3	图书1
4	书名: 三体
5	作者: 刘慈欣
6	分类: 科幻小说
7	关键词: 外星人、文明、战争
8	简介: 地球文明与三体文明展开了一场跨越光年的宇宙战争，人类命运岌岌可危。
9	借出状态: 未借出
10	借出次数: 121
11	
12	图书2
13	书名: 1984
14	作者: George Orwell
15	分类: Dystopian Fiction
16	关键词: Totalitarianism, Surveillance, Reality Control
17	简介: A dystopian novel depicting a society under constant surveillance and ruled by a totalitarian regime.
18	借出状态: 未借出
19	借出次数: 94
20	
21	图书3
22	书名: Pride and Prejudice
23	作者: Jane Austen
24	分类: Romantic Literature
25	关键词: Love, Social Class, Manners
26	简介: A witty and humorous tale of manners, marriage, and the pursuit of love in early 19th-century England.
27	借出状态: 未借出
28	借出次数: 65
29	
30	图书4
31	书名: 小王子
32	作者: 安托万·德·圣埃克苏佩里
33	分类: 童话故事
34	关键词: 成长、哲理、星球
35	简介: 一部充满哲理的童话故事，讲述了小王子在不同星球上的冒险经历。
36	借出状态: 未借出
37	借出次数: 62
38	
39	图书5
40	书名: 哈利·波特
41	作者: J.K.罗琳 (J.K.Rowling)
42	分类: 奇幻文学
43	关键词: 魔法, 友谊, 勇气

44 简介：这一系列小说讲述了一个名叫哈利·波特的孤儿在11岁生日时发现自己是一个巫师，随后被邀请进入霍格沃茨魔法与巫术学校学习 的故事。

45 借出状态：已借出

46 借出次数：61

47

48 图书6

49 书名：时间简史

50 作者：史蒂芬·霍金

51 分类：科普读物

52 关键词：宇宙、物理、黑洞

53 简介：本书以浅显易懂的语言，介绍了宇宙的起源、发展和结构，让读者领略物理学的魅力。

54 借出状态：未借出

55 借出次数：56

56

57 图书7

58 书名：苏菲的世界

59 作者：乔斯坦·贾德

60 分类：哲学读物

61 关键词：哲学、思考、人生

62 简介：以小说的形式，向读者介绍了西方哲学史上的主要流派和思想。

63 借出状态：未借出

64 借出次数：44

65

66 图书8

67 书名：围城

68 作者：钱钟书

69 分类：现代文学

70 关键词：婚姻、爱情、人生

71 简介：以幽默诙谐的笔触，揭示了都市男女在爱情和婚姻中的困惑与挣扎。

72 借出状态：未借出

73 借出次数：34

74

75 图书9

76 书名：解忧杂货店

77 作者：东野圭吾

78 分类：推理小说

79 关键词：温情、治愈、推理

80 简介：一家神奇的杂货店，为人们解决烦恼，串联起一个个温馨的故事。

81 借出状态：未借出

82 借出次数：25

83

84 图书10

85 书名：活着

86 作者：余华

87 分类：现代文学

88 关键词：生存，苦难，人性

89 简介：小说以20世纪40年代至70年代中国农村为背景，通过讲述主人公福贵的一生，展现了农民在社会变革中所经历的磨难和折磨。

```
90 | 借出状态: 已借出
91 | 借出次数: 14
92 |
93 | 按任意键返回
```

11. 十大活跃用户（示例）

```
1 | 十大活跃用户
2 |
3 | 用户1: 李四
4 | 借阅次数: 18
5 |
6 | 用户2: 关羽
7 | 借阅次数: 8
8 |
9 | 用户3: 刘备
10 | 借阅次数: 2
11 |
12 | 按任意键返回
```

12. 删除所有图书（用户）

```
1 | 确认删除所有图书? (y/n)n
2 |
3 | 取消删除
4 |
5 | 按任意键返回
```

13. 非法输入

总结

这次大作业完成的较为满意，完成了所有功能开发、测试，考虑了系统兼容性、乱码等问题，数据便于直接修改。不过也有一些不足，例如 Windows 系统路径对大小写不敏感，这可能导致文件路径的错误。另外，对于一些边界情况和异常情况的处理还可以进一步完善。在未来的版本中，可以考虑添加更多的功能和优化用户体验，比如增加图书推荐功能、用户评分功能等。总体来说，这次大作业是一个很好的学习和实践的机会，让我对图书管理系统的开发有了更深入的了解。我会继续努力学习，提升自己的编程能力。

附录：源程序清单

base.hpp

```
1  /**
2   * @file base.hpp
3   * @brief 包含字符串编码转换、字符串检查和基类定义的头文件。
4   * 此文件定义了两个主要的字符串编码转换函数(utf8_to_gbk, gbk_to_utf8)，一个字符串检查函数
   (IsPureNumber)以及一个用于演示虚函数概念的基类(Base)。
5   */
6
7  #pragma once
8  #include <iconv.h>
9  #include <filesystem>
10 #include <algorithm>
11 #include <vector>
12 #include <fstream>
13 #include <iostream>
14 #include <conio.h>
15
16 using namespace std;
17
18 /**
19  * 将UTF-8编码的字符串转换为GBK编码。
20  * @param utf8_str UTF-8编码的字符串。
21  * @return 转换后的GBK编码字符串。如果转换失败，返回空字符串。
22  */
23 string utf8_to_gbk(const string &utf8_str)
24 {
25     iconv_t cd = iconv_open("GBK", "UTF-8");
26     if (cd == (iconv_t)-1)
27         return "";
28     size_t in_bytes_left = utf8_str.size();
29     size_t out_bytes_left = in_bytes_left * 2;
30     char *in_buf = const_cast<char *>(utf8_str.c_str());
31     char out_buf[out_bytes_left];
32     char *out_buf_start = out_buf;
33     size_t ret = iconv(cd, &in_buf, &in_bytes_left, &out_buf_start, &out_bytes_left);
34     if (ret == (size_t)-1)
35     {
36         iconv_close(cd);
37         return "";
38     }
39     *out_buf_start = '\0';
40     iconv_close(cd);
41     return string(out_buf);
42 }
43
44 /**
```

```

45     * 将GBK编码的字符串转换为UTF-8编码。
46     * @param gbk_str GBK编码的字符串。
47     * @return 转换后的UTF-8编码字符串。如果转换失败，返回空字符串。
48     */
49     string gbk_to_utf8(const string &gbk_str)
50     {
51         iconv_t cd = iconv_open("UTF-8", "GBK");
52         if (cd == (iconv_t)-1)
53             return "";
54         size_t in_bytes_left = gbk_str.size();
55         size_t out_bytes_left = in_bytes_left * 2;
56         char *in_buf = const_cast<char *>(gbk_str.c_str());
57         char out_buf[out_bytes_left];
58         char *out_buf_start = out_buf;
59         size_t ret = iconv(cd, &in_buf, &in_bytes_left, &out_buf_start, &out_bytes_left);
60         if (ret == (size_t)-1)
61         {
62             iconv_close(cd);
63             return "";
64         }
65         *out_buf_start = '\0';
66         iconv_close(cd);
67         return string(out_buf);
68     }
69
70     /**
71     * 检查字符串是否全部由数字组成。
72     *
73     * @param input 待检查的字符串。
74     * @return 如果字符串全部由数字组成，则返回true；否则返回false。
75     */
76     bool IsPureNumber(const string &input)
77     {
78         return all_of(input.begin(), input.end(), ::isdigit);
79     }
80
81     /**
82     * 基类，定义了一组接口，用于演示虚函数的概念。
83     */
84     class Base
85     {
86     public:
87         /**
88         * 纯虚函数，要求派生类实现加法操作。
89         *
90         * @return 加法操作的结果。
91         */
92         virtual int Add() const = 0;

```

```

93
94     /**
95      * 纯虚函数，要求派生类实现保存操作。
96      */
97     virtual void Save() const = 0;
98
99     /**
100    * 纯虚函数，要求派生类实现删除操作。
101    *
102    * @return 删除操作的结果。
103    */
104    virtual int Delete() const = 0;
105
106    /**
107    * 纯虚函数，要求派生类实现编辑操作。
108    *
109    * @return 编辑操作的结果。
110    */
111    virtual int Edit() const = 0;
112 };

```

book.hpp

```

1  /**
2   * @file book.hpp
3   * 定义了Book类，用于表示图书信息，并继承自Base类。
4   */
5
6  #define FILESYSTEM_BOOK "./data/book/"
7  #include "base.hpp"
8
9  /**
10   * Book类用于存储和操作图书信息。
11   * 包括图书的标题、作者、分类、关键词、简介、借阅次数和借阅状态。
12   * 提供了图书信息的增加、保存、删除和编辑功能。
13   */
14  class Book : public Base
15  {
16  public:
17      // 图书属性
18      string title;           ///< 图书标题
19      string author;          ///< 作者
20      string category;         ///< 分类
21      string keywords;         ///< 关键词
22      string summary;          ///< 简介
23      int borrowTimes = 0;     ///< 借阅次数

```

```
24     bool isBorrowed = false; ///< 借阅状态
25
26     /**
27      * 构造函数，用于创建一个新的Book对象。
28      * @param Title 图书标题，默认为空字符串。
29      * @param Author 作者，默认为空字符串。
30      * @param Category 分类，默认为空字符串。
31      * @param Keywords 关键词，默认为空字符串。
32      * @param Summary 简介，默认为空字符串。
33      */
34     Book(string Title = "", string Author = "", string Category = "", string Keywords
= "", string Summary = "") : title(Title), author(Author), category(Category),
keywords(Keywords), summary(Summary) {}
35
36     /**
37      * 拷贝构造函数，用于创建一个新的Book对象，复制已有的Book对象。
38      * @param book 已有的Book对象。
39      */
40     Book(const Book &book) : title(book.title), author(book.author),
category(book.category), keywords(book.keywords), summary(book.summary),
borrowTimes(book.borrowTimes), isBorrowed(book.isBorrowed) {}
41
42     /**
43      * 析构函数。
44      */
45     ~Book() {}
46
47     /**
48      * 添加图书信息到文件系统。
49      * @return 成功返回1，文件已存在返回0，失败返回-1。
50      */
51     int Add() const override
52     {
53         string filePath = FILESYSTEM_BOOK + utf8_to_gbk(this->title) + ".txt";
54         if (ifstream(filePath))
55             return 0;
56         else
57         {
58             ofstream file(filePath);
59             if (!file)
60                 return -1;
61             else
62             {
63                 file << this->title << endl;
64                 file << this->author << endl;
65                 file << this->category << endl;
66                 file << this->keywords << endl;
67                 file << this->summary << endl;
```

```

68         file << this->isBorrowed << endl;
69         file << this->borrowTimes << endl;
70         file.close();
71         return 1;
72     }
73 }
74 }
75
76 /**
77  * 保存图书信息到文件系统。
78  */
79 void Save() const override
80 {
81     string filePath = FILESYSTEM_BOOK + utf8_to_gbk(this->title) + ".txt";
82     ofstream file(filePath);
83     file << this->title << endl;
84     file << this->author << endl;
85     file << this->category << endl;
86     file << this->keywords << endl;
87     file << this->summary << endl;
88     file << this->isBorrowed << endl;
89     file << this->borrowTimes << endl;
90     file.close();
91 }
92
93 /**
94  * 从文件系统中删除图书信息。
95  * @return 成功返回1, 失败返回-1。
96  */
97 int Delete() const override
98 {
99     string filePath = FILESYSTEM_BOOK + utf8_to_gbk(this->title) + ".txt";
100     if (remove(filePath.c_str()) == 0)
101         return 1;
102     else
103         return -1;
104 }
105
106 /**
107  * 编辑已有的图书信息。
108  * @return 成功返回1, 失败返回-1。
109  */
110 int Edit() const override
111 {
112     ofstream file(FILESYSTEM_BOOK + utf8_to_gbk(this->title) + ".txt");
113     if (!file)
114         return -1;
115     else

```

```

116         {
117             this->Save();
118             file.close();
119             return 1;
120         }
121     }
122
123     friend ostream &operator<<(ostream &, const Book &);
124 };
125
126 /**
127  * 重载输出操作符，用于打印图书信息。
128  * @param os 输出流对象。
129  * @param book 要输出的Book对象。
130  * @return 输出流对象。
131  */
132 ostream &operator<<(ostream &os, const Book &book)
133 {
134     os << "书名: " << book.title << endl;
135     os << "作者: " << book.author << endl;
136     os << "分类: " << book.category << endl;
137     os << "关键词: " << book.keywords << endl;
138     os << "简介: " << book.summary << endl;
139     if (book.isBorrowed)
140         os << "借出状态: 已借出" << endl;
141     else
142         os << "借出状态: 未借出" << endl;
143     os << "借出次数: " << book.borrowTimes << endl;
144     return os;
145 }

```

user.hpp

```

1  /**
2   * @file user.hpp
3   * @brief 定义User类和Record结构体，用于管理用户信息和借阅记录。
4   */
5
6  #define FILESYSTEM_USER "./data/user/"
7  #include "base.hpp"
8
9  /**
10   * @struct Record
11   * @brief 存储单个借阅记录的详细信息。
12   */
13  struct Record

```



```
14 {
15     string bookName;    ///< 书名
16     string borrowTime;  ///< 借书时间
17     string returnTime;  ///< 还书时间
18     bool isReturned;    ///< 是否已还书
19
20     /**
21      * @brief 默认构造函数，初始化借阅记录。
22      */
23     Record() : bookName(""), borrowTime(""), returnTime(""), isReturned(false) {}
24 };
25
26 /**
27  * @class User
28  * @brief 表示一个用户及其借阅记录。
29  * 继承自Base类，用于表示一个用户及其借阅记录。
30  */
31 class User : public Base
32 {
33 public:
34     string name;                ///< 用户名
35     vector<Record> borrowRecords; ///< 用户的借阅记录列表
36     int borrowTimes = 0;        ///< 用户的借阅次数
37
38     /**
39      * @brief 构造函数，创建一个新的用户对象。
40      * @param Name 用户名
41      */
42     User(string Name = "") : name(Name) {}
43
44     /**
45      * @brief 拷贝构造函数，用另一个User对象初始化此对象。
46      * @param user 另一个User对象
47      */
48     User(const User &user) : name(user.name), borrowRecords(user.borrowRecords),
49 borrowTimes(user.borrowTimes) {}
50
51     /**
52      * @brief 析构函数。
53      */
54     ~User() {}
55
56     /**
57      * @brief 添加用户信息到文件系统。
58      * @return 成功返回1，如果文件已存在返回0，失败返回-1。
59      */
60     int Add() const override
61     {
```

```

61         string filePath = FILESYSTEM_USER + utf8_to_gbk(this->name) + ".txt";
62         if (ifstream(filePath))
63             return 0;
64         else
65         {
66             ofstream file(filePath);
67             if (!file)
68                 return -1;
69             else
70             {
71                 file.close();
72                 return 1;
73             }
74         }
75     }
76
77     /**
78     * @brief 保存用户的借阅记录到文件系统。
79     */
80     void Save() const override
81     {
82         string filePath = FILESYSTEM_USER + utf8_to_gbk(this->name) + ".txt";
83         ofstream file(filePath);
84         for (auto record : this->borrowRecords)
85         {
86             file << record.bookName << endl;
87             file << record.borrowTime << endl;
88             file << record.returnTime << endl;
89             file << record.isReturned << endl;
90         }
91         file.close();
92     }
93
94     /**
95     * @brief 从文件系统中删除用户信息。
96     * @return 成功返回1, 失败返回-1。
97     */
98     int Delete() const override
99     {
100         string filePath = FILESYSTEM_USER + utf8_to_gbk(this->name) + ".txt";
101         if (remove(filePath.c_str()) == 0)
102             return 1;
103         else
104             return -1;
105     }
106
107     /**
108     * @brief 编辑用户信息。

```

```
109     * @return 成功返回1, 失败返回-1。
110     */
111     int Edit() const override
112     {
113         ofstream file(FILESYSTEM_USER + utf8_to_gbk(this->name) + ".txt");
114         if (!file)
115             return -1;
116         else
117         {
118             this->Save();
119             file.close();
120             return 1;
121         }
122     }
123
124     friend ostream &operator<<(ostream &, const User &);
125 };
126
127 /**
128  * @brief 重载输出操作符, 用于打印用户信息和借阅记录。
129  * @param os 输出流对象
130  * @param user 用户对象
131  * @return 输出流对象
132  */
133 ostream &operator<<(ostream &os, const User &user)
134 {
135     os << "借阅次数: " << user.borrowTimes << endl;
136     os << endl;
137     os << "借阅记录: " << endl;
138     os << endl;
139     for (auto record : user.borrowRecords)
140     {
141         os << "书名: " << record.bookName << endl;
142         os << "借书时间: " << record.borrowTime << endl;
143         if (record.isReturned)
144             os << "还书时间: " << record.returnTime << endl;
145         else
146             os << "还书时间: 未还" << endl;
147         os << endl;
148     }
149     return os;
150 }
```

bookmanager.hpp

```
1  /**
2   * @file bookmanager.hpp
3   * @brief 管理书籍信息的类，包括获取书籍、搜索书籍、获取热门书籍和删除所有书籍的功能。
4   */
5
6  #include "book.hpp"
7
8  /**
9   * @class BookManager
10   * @brief 用于管理书籍信息。
11   * 提供了获取单本书籍信息、根据关键字搜索书籍、获取借阅次数最多的十本书籍以及删除所有书籍信息的功能。
12   */
13  class BookManager
14  {
15  public:
16      /**
17       * @brief 根据书名获取书籍信息。
18       * @param title 书籍的标题。
19       * @return 如果找到书籍，则返回书籍对象；否则，返回一个空的书籍对象。
20       */
21      Book getBook(const string &title) const
22      {
23          string filePath = FILESYSTEM_BOOK + utf8_to_gbk(title) + ".txt";
24          if (!ifstream(filePath))
25          {
26              return Book();
27          }
28          else
29          {
30              ifstream file(filePath);
31              if (!file)
32              {
33                  return Book();
34              }
35              else
36              {
37                  Book book;
38                  getline(file, book.title);
39                  getline(file, book.author);
40                  getline(file, book.category);
41                  getline(file, book.keywords);
42                  getline(file, book.summary);
43                  string line;
44                  getline(file, line);
45                  book.isBorrowed = (line == "1");
```

```

46         getline(file, line);
47         book.borrowTimes = stoi(line);
48         file.close();
49         return book;
50     }
51 }
52 }
53
54 /**
55  * @brief 根据关键字搜索书籍。
56  * @param keyword 搜索书籍时使用的关键字。
57  * @return 包含所有匹配关键字的书籍对象的向量。
58  */
59 vector<Book> searchBook(const string &keyword) const
60 {
61     vector<Book> results;
62     for (const auto &entry : filesystem::directory_iterator(FILESYSTEM_BOOK))
63     {
64         string filePath = entry.path().string();
65         filePath = utf8_to_gbk(filePath);
66         ifstream file(filePath);
67         if (file)
68         {
69             Book book;
70             getline(file, book.title);
71             getline(file, book.author);
72             getline(file, book.category);
73             getline(file, book.keywords);
74             getline(file, book.summary);
75             string line;
76             getline(file, line);
77             book.isBorrowed = (line == "1");
78             getline(file, line);
79             book.borrowTimes = stoi(line);
80             file.close();
81             if (book.title.find(keyword) != string::npos ||
82                 book.author.find(keyword) != string::npos ||
83                 book.category.find(keyword) != string::npos ||
84                 book.keywords.find(keyword) != string::npos ||
85                 book.summary.find(keyword) != string::npos)
86             {
87                 results.push_back(book);
88             }
89         }
90     }
91     return results;
92 }
93

```

```

94     /**
95      * @brief 获取借阅次数最多的十本书籍。
96      * @return 包含借阅次数最多的十本书籍的向量。
97      */
98     vector<Book> tenHotBooks() const
99     {
100         vector<Book> results;
101         for (const auto &entry : filesystem::directory_iterator(FILESYSTEM_BOOK))
102         {
103             string filePath = entry.path().string();
104             filePath = utf8_to_gbk(filePath);
105             ifstream file(filePath);
106             if (file)
107             {
108                 Book book;
109                 getline(file, book.title);
110                 getline(file, book.author);
111                 getline(file, book.category);
112                 getline(file, book.keywords);
113                 getline(file, book.summary);
114                 string line;
115                 getline(file, line);
116                 book.isBorrowed = (line == "1");
117                 getline(file, line);
118                 book.borrowTimes = stoi(line);
119                 file.close();
120                 if (book.borrowTimes > 0)
121                 {
122                     results.push_back(book);
123                 }
124             }
125         }
126         sort(results.begin(), results.end(), [](Book a, Book b)
127             { return a.borrowTimes > b.borrowTimes; });
128         if (results.size() > 10)
129         {
130             results.resize(10);
131         }
132         return results;
133     }
134
135     /**
136      * @brief 删除所有书籍信息。
137      * @return 总是返回1, 表示操作完成。
138      */
139     int deleteAllBooks() const
140     {
141         for (const auto &entry : filesystem::directory_iterator(FILESYSTEM_BOOK))

```

```

142         {
143             string filePath = entry.path().string();
144             filePath = utf8_to_gbk(filePath);
145             remove(filePath.c_str());
146         }
147         return 1;
148     }
149 };

```

usermanager.hpp

```

1  /**
2   * @file usermanager.hpp
3   * @brief 用户管理类定义文件
4   * 提供了对用户信息进行管理的类，包括获取单个用户信息、搜索用户、获取活跃用户列表和删除所有用户等功能。
5   */
6
7  #include "user.hpp"
8
9  /**
10   * @class UserManager
11   * @brief 用户管理类
12   * 用于管理用户信息，包括获取、搜索、列出活跃用户和删除用户等操作。
13   */
14  class UserManager
15  {
16  public:
17      /**
18       * 根据用户名从文件系统中读取用户信息，包括借阅记录等，并返回一个用户对象。
19       * @brief 获取单个用户的信息
20       * @param name 用户名
21       * @return User 用户对象，如果用户不存在则返回空的用户对象
22       */
23      User getUser(const string &name) const
24      {
25          string filePath = FILESYSTEM_USER + name + ".txt";
26          filePath = utf8_to_gbk(filePath);
27          if (!ifstream(filePath))
28          {
29              return User();
30          }
31          else
32          {
33              ifstream file(filePath);
34              if (!file)
35              {

```

```

36         return User();
37     }
38     else
39     {
40         User user(name);
41         string line;
42         while (getline(file, line))
43         {
44             Record record;
45             record.bookName = line;
46             getline(file, line);
47             record.borrowTime = line;
48             getline(file, line);
49             record.returnTime = line;
50             getline(file, line);
51             record.isReturned = (line == "1");
52             user.borrowRecords.push_back(record);
53         }
54         user.borrowTimes = user.borrowRecords.size();
55         file.close();
56         return user;
57     }
58 }
59 }
60
61 /**
62  * 在所有用户中搜索包含指定关键字的用户名，并返回一个包含这些用户的列表。
63  * @brief 搜索包含关键字的用户列表
64  * @param keyword 搜索关键字
65  * @return vector<User> 包含关键字的用户列表
66  */
67 vector<User> searchUser(const string &keyword) const
68 {
69     vector<User> results;
70     for (const auto &entry : filesystem::directory_iterator(FILESYSTEM_USER))
71     {
72         string filePath = entry.path().string();
73         filePath = utf8_to_gbk(filePath);
74         ifstream file(filePath);
75         if (file)
76         {
77             User user;
78             user.name = gbk_to_utf8(filePath.substr(12, filePath.size() - 16));
79             string line;
80             while (getline(file, line))
81             {
82                 Record record;
83                 record.bookName = line;

```



```

84         getline(file, line);
85         record.borrowTime = line;
86         getline(file, line);
87         record.returnTime = line;
88         getline(file, line);
89         record.isReturned = (line == "1");
90         user.borrowRecords.push_back(record);
91     }
92     user.borrowTimes = user.borrowRecords.size();
93     file.close();
94     if (user.name.find(keyword) != string::npos)
95     {
96         results.push_back(user);
97     }
98 }
99 }
100 return results;
101 }
102
103 /**
104  * 根据用户的借阅次数，获取最活跃的用户列表，列表最多包含10个用户。
105  * @brief 获取最活跃的用户列表
106  * @return vector<User> 最活跃的用户列表，最多10个
107  */
108 vector<User> tenActiveUsers() const
109 {
110     vector<User> results;
111     for (const auto &entry : filesystem::directory_iterator(FILESYSTEM_USER))
112     {
113         string filePath = entry.path().string();
114         filePath = utf8_to_gbk(filePath);
115         ifstream file(filePath);
116         if (file)
117         {
118             User user;
119             user.name = gbk_to_utf8(filePath.substr(12, filePath.size() - 16));
120             string line;
121             while (getline(file, line))
122             {
123                 Record record;
124                 record.bookName = line;
125                 getline(file, line);
126                 record.borrowTime = line;
127                 getline(file, line);
128                 record.returnTime = line;
129                 getline(file, line);
130                 record.isReturned = (line == "1");
131                 user.borrowRecords.push_back(record);

```

```

132         }
133         user.borrowTimes = user.borrowRecords.size();
134         file.close();
135         if (user.borrowTimes > 0)
136         {
137             results.push_back(user);
138         }
139     }
140 }
141 sort(results.begin(), results.end(), [](User a, User b)
142     { return a.borrowTimes > b.borrowTimes; });
143 if (results.size() > 10)
144 {
145     results.resize(10);
146 }
147 return results;
148 }
149
150 /**
151  * 删除文件系统中所有用户的信息。
152  * @brief 删除所有用户
153  * @return int 操作结果，成功返回1，失败返回0
154  */
155 int deleteAllUsers() const
156 {
157     for (const auto &entry : filesystem::directory_iterator(FILESYSTEM_USER))
158     {
159         string filePath = entry.path().string();
160         filePath = utf8_to_gbk(filePath);
161         remove(filePath.c_str());
162     }
163     return 1;
164 }
165 };

```

manager.hpp

```

1  /**
2   * @file manager.hpp
3   * @brief 管理图书和用户的主要功能类
4   * Manager 类继承自 BookManager 和 UserManager，提供了管理图书和用户的高级功能，包括借书、还书以及
   获取当前日期时间。
5   */
6
7  #include "bookmanager.hpp"
8  #include "usermanager.hpp"

```

```

9
10 class Manager : public BookManager, public UserManager
11 {
12 public:
13     /**
14      * @brief 获取当前日期和时间的字符串表示形式
15      * @return 当前日期和时间的字符串, 格式为 YYYY-MM-DD HH:MM:SS
16      */
17     string getCurrentDateTime() const
18     {
19         time_t now = time(0);
20         tm *ltm = localtime(&now);
21         char buffer[80];
22         strftime(buffer, sizeof(buffer), "%Y-%m-%d %H:%M:%S", ltm);
23         return buffer;
24     }
25
26     /**
27      * 根据用户名和书名进行借书操作。如果书已被借出或用户/书不存在, 则返回错误代码。
28      * @brief 借书操作
29      * @param userName 借书用户的用户名
30      * @param bookName 被借的书名
31      * @return 操作结果代码: 1 成功, 0 书不存在, -1 书已被借出, -2 用户不存在
32      */
33     int borrowBook(const string &userName, const string &bookName) const
34     {
35         Book book = getBook(bookName);
36         if (book.title == "")
37             return 0;
38         User user = getUser(userName);
39         if (user.name == "")
40             return -2;
41         if (book.isBorrowed)
42             return -1;
43         Record record;
44         record.bookName = bookName;
45         record.borrowTime = getCurrentDateTime();
46         record.isReturned = false;
47         user.borrowRecords.push_back(record);
48         user.borrowTimes = user.borrowRecords.size();
49         user.Save();
50         book.isBorrowed = true;
51         book.borrowTimes++;
52         book.Save();
53         return 1;
54     }
55
56     /**

```

```

57     * 根据用户名和书名进行还书操作。如果书未被当前用户借出或用户/书不存在，则返回错误代码。
58     * @brief 还书操作
59     * @param userName 还书用户的用户名
60     * @param bookName 被还的书名
61     * @return 操作结果代码：1 成功，0 书不存在，-1 书未被当前用户借出，-2 用户不存在
62     */
63     int returnBook(const string &userName, const string &bookName) const
64     {
65         Book book = getBook(bookName);
66         if (book.title == "")
67             return 0;
68         User user = getUser(userName);
69         if (user.name == "")
70             return -2;
71         bool found = false;
72         for (auto &record : user.borrowRecords)
73         {
74             if (record.bookName == bookName && !record.isReturned)
75             {
76                 record.returnTime = getCurrentDateTime();
77                 record.isReturned = true;
78                 user.Save();
79                 book.isBorrowed = false;
80                 book.Save();
81                 return 1;
82             }
83         }
84         return -1;
85     }
86 };

```

gui.hpp

```

1  /**
2   * @file GUI.hpp
3   * @brief 图书管理系统的图形用户界面(GUI)类定义文件。
4   * 该文件包含GUI类的定义。GUI类继承自Manager类，提供了图书管理系统的用户界面功能，包括显示菜单、添
5   */
6
7  #include "manager.hpp"
8
9  /**
10   * @class GUI GUI.hpp "GUI.hpp"
11   * @brief 图书管理系统的图形用户界面(GUI)类。

```

* GUI类提供了图书管理系统的所有用户界面功能。它允许用户通过文本菜单进行操作，如添加、删除、查找和编辑图书和用户信息。该类继承自Manager类，使用继承而来的方法来执行用户请求的操作。

*/

```
class GUI : public Manager
```

```
{
```

```
public:
```

```
    /**
```

```
    * @brief 显示主菜单。
```

```
    * 该方法清屏并显示图书管理系统的主菜单，包括所有可用的操作选项。
```

```
    */
```

```
void ShowMenu() const
```

```
{
```

```
    system("cls");
```

```
    cout << endl;
```

```
    cout << endl;
```

```
    cout << "                图书管理系统" << endl;
```

```
    cout << "-----" << endl;
```

```
    cout << "1. 添加图书          9. 图书借阅" << endl;
```

```
    cout << "2. 删除图书          10. 图书归还" << endl;
```

```
    cout << "3. 查找图书          11. 借阅记录" << endl;
```

```
    cout << "4. 编辑图书          12. 十大热门图书" << endl;
```

```
    cout << "5. 添加用户          13. 十大活跃用户" << endl;
```

```
    cout << "6. 删除用户          14. 删除所有图书" << endl;
```

```
    cout << "7. 查找用户          15. 删除所有用户" << endl;
```

```
    cout << "8. 编辑用户          16. 退出" << endl;
```

```
    cout << "-----" << endl;
```

```
    cout << endl;
```

```
    cout << "请选择操作: ";
```

```
}
```

```
    /**
```

```
    * @brief 移除字符串前后的空白字符。
```

```
    * @param str 输入的字符串。
```

```
    * @return 移除前后空白字符后的字符串。
```

```
    */
```

```
string RemoveBlank(const string &str) const
```

```
{
```

```
    auto start = str.find_first_not_of(" \t\n\r\f\v");
```

```
    if (start == string::npos)
```

```
        return "";
```

```
    auto end = str.find_last_not_of(" \t\n\r\f\v");
```

```
    return str.substr(start, end - start + 1);
```

```
}
```

```
    /**
```

```
    * @brief 显示单本图书的详细信息。
```

```
    * @param book 要显示的图书对象。
```

```
    */
```

```
59 void DisplayBook(const Book &book) const
60 {
61     cout << book;
62 }
63
64 /**
65  * @brief 添加新图书的界面和逻辑。
66  * 该方法引导用户通过一系列提示输入新图书的详细信息，然后尝试添加图书到系统中。
67  */
68 void AddBook() const
69 {
70     system("cls");
71     Book book;
72     cout << endl;
73     cout << endl;
74     cout << "添加图书" << endl;
75     cout << endl;
76     cout << endl;
77     cout << "请输入书名: ";
78     getline(cin, book.title);
79     book.title = RemoveBlank(book.title);
80     if (book.title.empty())
81     {
82         cout << endl;
83         cout << "书名不能为空" << endl;
84         cout << endl;
85         cout << "按任意键返回" << endl;
86         getch();
87         return;
88     }
89     else if (getBook(book.title).title == book.title)
90     {
91         cout << "图书已存在" << endl;
92         cout << endl;
93         cout << "按任意键返回" << endl;
94         getch();
95         return;
96     }
97     cout << "请输入作者: ";
98     getline(cin, book.author);
99     book.author = RemoveBlank(book.author);
100    cout << "请输入分类: ";
101    getline(cin, book.category);
102    book.category = RemoveBlank(book.category);
103    cout << "请输入关键词: ";
104    getline(cin, book.keywords);
105    book.keywords = RemoveBlank(book.keywords);
106    cout << "请输入简介: ";
```

```

107         getline(cin, book.summary);
108         book.summary = RemoveBlank(book.summary);
109         cout << endl;
110         int result = book.Add();
111         switch (result)
112         {
113             case 0:
114                 cout << "图书已存在" << endl;
115                 break;
116             case -1:
117                 cout << "保存失败" << endl;
118                 break;
119             case 1:
120                 cout << "保存成功" << endl;
121                 break;
122         }
123         cout << endl;
124         cout << "按任意键返回" << endl;
125         getch();
126     }
127
128     /**
129     * @brief 删除图书的界面和逻辑。
130     * 该方法提示用户输入要删除的图书名称，然后尝试从系统中删除指定的图书。
131     */
132     void DeleteBook() const
133     {
134         system("cls");
135         cout << endl;
136         cout << endl;
137         cout << "删除图书" << endl;
138         cout << endl;
139         cout << endl;
140         cout << "请输入书名: ";
141         string title;
142         getline(cin, title);
143         title = RemoveBlank(title);
144         cout << endl;
145         if (title.empty())
146         {
147             cout << "书名不能为空" << endl;
148             cout << endl;
149             cout << "按任意键返回" << endl;
150             getch();
151             return;
152         }
153         if (getBook(title).title.empty())
154         {

```

```

155         cout << "图书不存在" << endl;
156         cout << endl;
157         cout << "按任意键返回" << endl;
158         getch();
159         return;
160     }
161     Book book(title);
162     cout << "确认删除? (y/n)";
163     string c;
164     getline(cin, c);
165     cout << endl;
166     if (c != "y")
167     {
168         cout << "取消删除" << endl;
169         cout << endl;
170         cout << "按任意键返回" << endl;
171         getch();
172         return;
173     }
174     int result = book.Delete();
175     switch (result)
176     {
177     case -1:
178         cout << "删除失败" << endl;
179         break;
180     case 1:
181         cout << "删除成功" << endl;
182         break;
183     }
184     cout << endl;
185     cout << "按任意键返回" << endl;
186     getch();
187 }
188
189 /**
190  * @brief 查找图书的界面和逻辑。
191  * 该方法提示用户输入搜索词（书名），然后显示所有匹配的图书信息。
192  */
193 void SearchBook() const
194 {
195     system("cls");
196     cout << endl;
197     cout << endl;
198     cout << "查找图书" << endl;
199     cout << endl;
200     cout << endl;
201     cout << "请输入搜索词（回车显示所有图书）： ";
202     string title;

```



```

203     getline(cin, title);
204     title = RemoveBlank(title);
205     cout << endl;
206     cout << "查询结果" << endl;
207     cout << endl;
208     vector<Book> books = searchBook(title);
209     int result = books.size() == 0 ? 0 : 1;
210     switch (result)
211     {
212     case 0:
213         cout << "图书不存在" << endl;
214         break;
215     case 1:
216         for (int i = 0; i < books.size(); i++)
217         {
218             cout << "图书" << i + 1 << endl;
219             DisplayBook(books[i]);
220             cout << endl;
221         }
222     }
223     cout << endl;
224     cout << "按任意键返回" << endl;
225     getch();
226 }
227
228 /**
229  * @brief 编辑图书信息的界面和逻辑。
230  * 该方法首先提示用户输入要编辑的图书名称，然后允许用户修改图书的详细信息。
231  */
232 void EditBook() const
233 {
234     system("cls");
235     cout << endl;
236     cout << endl;
237     cout << "编辑图书" << endl;
238     cout << endl;
239     cout << endl;
240     cout << "请输入书名: ";
241     string title;
242     getline(cin, title);
243     title = RemoveBlank(title);
244     cout << endl;
245     if (title.empty())
246     {
247         cout << "书名不能为空" << endl;
248         cout << endl;
249         cout << "按任意键返回" << endl;
250         getch();

```

```
251         return;
252     }
253     Book oldBook = getBook(title);
254     int result = oldBook.title.empty() ? 0 : 1;
255     switch (result)
256     {
257     case 0:
258         cout << "图书不存在" << endl;
259         break;
260     case 1:
261         DisplayBook(oldBook);
262         cout << endl;
263         cout << "请输入新书名: ";
264         Book book;
265         getline(cin, book.title);
266         book.title = RemoveBlank(book.title);
267         cout << endl;
268         Book newBook = getBook(book.title);
269         int result_ = newBook.title.empty() ? 0 : 1;
270         if (book.title.empty())
271         {
272             cout << "书名不能为空" << endl;
273             cout << endl;
274             cout << "按任意键返回" << endl;
275             getch();
276             return;
277         }
278         else if (result_)
279         {
280             cout << "图书已存在" << endl;
281             cout << endl;
282             cout << "按任意键返回" << endl;
283             getch();
284             return;
285         }
286         cout << "请输入新作者: ";
287         getline(cin, book.author);
288         book.author = RemoveBlank(book.author);
289         cout << "请输入新分类: ";
290         getline(cin, book.category);
291         book.category = RemoveBlank(book.category);
292         cout << "请输入新关键词: ";
293         getline(cin, book.keywords);
294         book.keywords = RemoveBlank(book.keywords);
295         cout << "请输入新简介: ";
296         getline(cin, book.summary);
297         book.summary = RemoveBlank(book.summary);
298         cout << endl;
```

```

299         book.isBorrowed = oldBook.isBorrowed;
300         book.borrowTimes = oldBook.borrowTimes;
301         oldBook.Delete();
302         int result = book.Edit();
303         switch (result)
304         {
305             case -1:
306                 cout << "保存失败" << endl;
307                 break;
308             case 1:
309                 cout << "保存成功" << endl;
310                 break;
311         }
312     }
313     cout << endl;
314     cout << "按任意键返回" << endl;
315     getch();
316 }
317
318 /**
319  * @brief 显示单个用户的详细信息。
320  * @param user 要显示的用户对象。
321  */
322 void DisplayUser(const User &user) const
323 {
324     cout << user;
325 }
326
327 /**
328  * @brief 添加新用户的界面和逻辑。
329  * 该方法引导用户通过一系列提示输入新用户的详细信息，然后尝试添加用户到系统中。
330  */
331 void AddUser() const
332 {
333     system("cls");
334     User user;
335     cout << endl;
336     cout << endl;
337     cout << "添加用户" << endl;
338     cout << endl;
339     cout << endl;
340     cout << "请输入用户名: ";
341     getline(cin, user.name);
342     user.name = RemoveBlank(user.name);
343     cout << endl;
344     if (user.name.empty())
345     {
346         cout << "用户名不能为空" << endl;

```

```
347         cout << endl;
348         cout << "按任意键返回" << endl;
349         getch();
350         return;
351     }
352     int result = user.Add();
353     switch (result)
354     {
355     case 0:
356         cout << "用户已存在" << endl;
357         break;
358     case -1:
359         cout << "保存失败" << endl;
360         break;
361     case 1:
362         cout << "保存成功" << endl;
363         break;
364     }
365     cout << endl;
366     cout << "按任意键返回" << endl;
367     getch();
368 }
369
370 /**
371  * @brief 删除用户的界面和逻辑。
372  * 该方法提示用户输入要删除的用户名，然后尝试从系统中删除指定的用户。
373  */
374 void DeleteUser() const
375 {
376     system("cls");
377     cout << endl;
378     cout << endl;
379     cout << "删除用户" << endl;
380     cout << endl;
381     cout << endl;
382     cout << "请输入用户名: ";
383     string name;
384     getline(cin, name);
385     name = RemoveBlank(name);
386     cout << endl;
387     if (name.empty())
388     {
389         cout << "用户名不能为空" << endl;
390         cout << endl;
391         cout << "按任意键返回" << endl;
392         getch();
393         return;
394     }
```

```

395         if (getUser(name).name.empty())
396         {
397             cout << "用户不存在" << endl;
398             cout << endl;
399             cout << "按任意键返回" << endl;
400             getch();
401             return;
402         }
403         User user(name);
404         cout << "确认删除? (y/n)";
405         string c;
406         getline(cin, c);
407         cout << endl;
408         if (c != "y")
409         {
410             cout << "取消删除" << endl;
411             cout << endl;
412             cout << "按任意键返回" << endl;
413             getch();
414             return;
415         }
416         int result = user.Delete();
417         switch (result)
418         {
419             case 1:
420                 cout << "删除成功" << endl;
421                 break;
422             case -1:
423                 cout << "删除失败" << endl;
424                 break;
425         }
426         cout << endl;
427         cout << "按任意键返回" << endl;
428         getch();
429     }
430
431     /**
432     * @brief 搜索用户。
433     * 清屏并提示用户输入用户名，根据输入搜索用户。如果未输入用户名，则显示所有用户。根据搜索结果，显
434     示用户信息或提示用户不存在。
435     */
436     void SearchUser() const
437     {
438         system("cls");
439         cout << endl;
440         cout << endl;
441         cout << "查找用户" << endl;
442         cout << endl;

```

```

442         cout << endl;
443         cout << "请输入用户名 (回车显示所有用户) : ";
444         string name;
445         getline(cin, name);
446         cout << endl;
447         cout << "查询结果" << endl;
448         cout << endl;
449         vector<User> users = searchUser(name);
450         int result = users.size() == 0 ? 0 : 1;
451         switch (result)
452         {
453             case 0:
454                 cout << "用户不存在" << endl;
455                 break;
456             case 1:
457                 for (int i = 0; i < users.size(); i++)
458                 {
459                     cout << "用户" << i + 1 << ": " << users[i].name << endl;
460                     cout << endl;
461                 }
462             }
463             cout << endl;
464             cout << "按任意键返回" << endl;
465             getch();
466         }
467
468     /**
469     * @brief 编辑用户。
470     * 清屏并提示用户输入用户名，然后输入新的用户名进行更新。如果用户存在，则更新用户信息；否则，提示
    用户不存在。
471     */
472     void EditUser() const
473     {
474         system("cls");
475         cout << endl;
476         cout << endl;
477         cout << "编辑用户" << endl;
478         cout << endl;
479         cout << endl;
480         cout << "请输入用户名: ";
481         string oldname;
482         getline(cin, oldname);
483         oldname = RemoveBlank(oldname);
484         cout << endl;
485         if (oldname.empty())
486         {
487             cout << "用户名不能为空" << endl;
488             cout << endl;

```

```
489         cout << "按任意键返回" << endl;
490         getch();
491         return;
492     }
493     User oldUser = getUser(oldname);
494     int result = oldUser.name.empty() ? 0 : 1;
495     switch (result)
496     {
497     case 0:
498         cout << "用户不存在" << endl;
499         break;
500     case 1:
501         cout << "请输入新用户名: ";
502         User user;
503         getline(cin, user.name);
504         user.name = RemoveBlank(user.name);
505         cout << endl;
506         User newUser = getUser(user.name);
507         int result_ = oldUser.name.empty() ? 0 : 1;
508         if (user.name.empty())
509         {
510             cout << "用户名不能为空" << endl;
511             cout << endl;
512             cout << "按任意键返回" << endl;
513             getch();
514             return;
515         }
516         else if (result_)
517         {
518             cout << "用户已存在" << endl;
519             cout << endl;
520             cout << "按任意键返回" << endl;
521             getch();
522             return;
523         }
524         oldUser.Delete();
525         int result = user.Edit();
526         switch (result)
527         {
528         case -1:
529             cout << "保存失败" << endl;
530             break;
531         case 1:
532             cout << "保存成功" << endl;
533             break;
534         }
535     }
536     cout << endl;
```

```
537         cout << "按任意键返回" << endl;
538         getch();
539     }
540
541     /**
542     * @brief 借阅图书。
543     * 清屏并提示用户输入书名和用户名，然后尝试借阅图书。根据操作结果，显示相应的提示信息。
544     */
545     void BorrowBook() const
546     {
547         system("cls");
548         cout << endl;
549         cout << endl;
550         cout << "图书借阅" << endl;
551         cout << endl;
552         cout << endl;
553         cout << "请输入书名: ";
554         string title;
555         getline(cin, title);
556         title = RemoveBlank(title);
557         cout << endl;
558         if (title.empty())
559         {
560             cout << "书名不能为空" << endl;
561             cout << endl;
562             cout << "按任意键返回" << endl;
563             getch();
564             return;
565         }
566         cout << "请输入用户名: ";
567         string name;
568         getline(cin, name);
569         name = RemoveBlank(name);
570         cout << endl;
571         if (name.empty())
572         {
573             cout << "用户名不能为空" << endl;
574             cout << endl;
575             cout << "按任意键返回" << endl;
576             getch();
577             return;
578         }
579         int result = borrowBook(name, title);
580         switch (result)
581         {
582         case 0:
583             cout << "图书不存在" << endl;
584             break;
```



```
585         case -1:
586             cout << "图书已借出" << endl;
587             break;
588         case -2:
589             cout << "用户不存在" << endl;
590             break;
591         case 1:
592             cout << "图书借阅成功" << endl;
593             break;
594     }
595     cout << endl;
596     cout << "按任意键返回" << endl;
597     getch();
598 }
599
600 /**
601  * @brief 归还图书。
602  * 清屏并提示用户输入书名和用户名，然后尝试归还图书。根据操作结果，显示相应的提示信息。
603  */
604 void ReturnBook() const
605 {
606     system("cls");
607     cout << endl;
608     cout << endl;
609     cout << "图书归还" << endl;
610     cout << endl;
611     cout << endl;
612     cout << "请输入书名: ";
613     string title;
614     getline(cin, title);
615     title = RemoveBlank(title);
616     cout << endl;
617     if (title.empty())
618     {
619         cout << "书名不能为空" << endl;
620         cout << endl;
621         cout << "按任意键返回" << endl;
622         getch();
623         return;
624     }
625     cout << "请输入用户名: ";
626     string name;
627     getline(cin, name);
628     name = RemoveBlank(name);
629     cout << endl;
630     if (name.empty())
631     {
632         cout << "用户名不能为空" << endl;
```

```

633         cout << endl;
634         cout << "按任意键返回" << endl;
635         getch();
636         return;
637     }
638     int result = returnBook(name, title);
639     switch (result)
640     {
641     case 0:
642         cout << "图书不存在" << endl;
643         break;
644     case -1:
645         cout << "未借此图书" << endl;
646         break;
647     case -2:
648         cout << "用户不存在" << endl;
649         break;
650     case 1:
651         cout << "图书归还成功" << endl;
652         break;
653     }
654     cout << endl;
655     cout << "按任意键返回" << endl;
656     getch();
657 }
658
659 /**
660  * @brief 查看借阅记录。
661  * 清屏并提示用户输入用户名，然后显示该用户的借阅记录。如果用户不存在，显示相应的提示信息。
662  */
663 void BorrowRecord() const
664 {
665     system("cls");
666     cout << endl;
667     cout << endl;
668     cout << "借阅记录" << endl;
669     cout << endl;
670     cout << endl;
671     cout << "请输入用户名: ";
672     string name;
673     getline(cin, name);
674     name = RemoveBlank(name);
675     cout << endl;
676     if (name.empty())
677     {
678         cout << "用户名不能为空" << endl;
679         cout << endl;
680         cout << "按任意键返回" << endl;

```

```
681         getch();
682         return;
683     }
684     User user = getUser(name);
685     int result = user.name.empty() ? 0 : 1;
686     switch (result)
687     {
688     case 0:
689         cout << "用户不存在" << endl;
690         break;
691     case 1:
692         DisplayUser(user);
693     }
694     cout << endl;
695     cout << "按任意键返回" << endl;
696     getch();
697 }
698
699 /**
700  * @brief 查看十大热门图书。
701  * 清屏并显示当前图书馆系统中借阅次数最多的十本图书。
702  */
703 void TenHotBooks() const
704 {
705     system("cls");
706     cout << endl;
707     cout << endl;
708     cout << "十大热门图书" << endl;
709     cout << endl;
710     cout << endl;
711     vector<Book> books = tenHotBooks();
712     if (books.size() == 0)
713     {
714         cout << "无记录" << endl;
715         cout << endl;
716         cout << "按任意键返回" << endl;
717         getch();
718         return;
719     }
720     for (int i = 0; i < books.size(); i++)
721     {
722         cout << "图书" << i + 1 << endl;
723         DisplayBook(books[i]);
724         cout << endl;
725     }
726     cout << endl;
727     cout << "按任意键返回" << endl;
728     getch();
```

```
729     }
730
731     /**
732     * @brief 查看十大活跃用户。
733     * 清屏并显示当前图书馆系统中借阅图书次数最多的十名用户。
734     */
735     void TenActiveUsers() const
736     {
737         system("cls");
738         cout << endl;
739         cout << endl;
740         cout << "十大活跃用户" << endl;
741         cout << endl;
742         cout << endl;
743         vector<User> users = tenActiveUsers();
744         if (users.size() == 0)
745         {
746             cout << "无记录" << endl;
747             cout << endl;
748             cout << "按任意键返回" << endl;
749             getch();
750             return;
751         }
752         for (int i = 0; i < users.size(); i++)
753         {
754             cout << "用户" << i + 1 << ": " << users[i].name << endl;
755             cout << "借阅次数: " << users[i].borrowTimes << endl;
756             cout << endl;
757         }
758         cout << endl;
759         cout << "按任意键返回" << endl;
760         getch();
761     }
762
763     /**
764     * @brief 删除所有图书。
765     * 清屏并提示用户确认是否删除所有图书。根据用户的选择，执行删除操作或取消。
766     */
767     void DeleteAllBooks() const
768     {
769         system("cls");
770         cout << endl;
771         cout << endl;
772         cout << "确认删除所有图书? (y/n)";
773         string c;
774         getline(cin, c);
775         cout << endl;
776         if (c == "y")
```

```

777     {
778         int result = deleteAllBooks();
779         switch (result)
780         {
781             case 1:
782                 cout << "删除成功" << endl;
783                 break;
784             default:
785                 cout << "删除失败" << endl;
786                 break;
787         }
788     }
789     else
790     {
791         cout << "取消删除" << endl;
792     }
793     cout << endl;
794     cout << "按任意键返回" << endl;
795     getch();
796 }
797
798 /**
799  * @brief 删除所有用户。
800  * 清屏并提示用户确认是否删除所有用户。根据用户的选择，执行删除操作或取消。
801  */
802 void DeleteAllUsers() const
803 {
804     system("cls");
805     cout << endl;
806     cout << endl;
807     cout << "确认删除所有用户? (y/n)";
808     string c;
809     getline(cin, c);
810     cout << endl;
811     if (c == "y")
812     {
813         int result = deleteAllUsers();
814         switch (result)
815         {
816             case 1:
817                 cout << "删除成功" << endl;
818                 break;
819             default:
820                 cout << "删除失败" << endl;
821                 break;
822         }
823     }
824     else

```

```

825     {
826         cout << "取消删除" << endl;
827     }
828     cout << endl;
829     cout << "按任意键返回" << endl;
830     getch();
831 }
832
833 /**
834  * @brief 退出程序。
835  * 安全退出图书馆系统。
836  */
837 void Exit() const
838 {
839     exit(0);
840 }
841
842 /**
843  * @brief 显示错误信息。
844  * 当用户输入无效时，显示错误信息并提示重新输入。
845  */
846 void Error() const
847 {
848     cout << endl;
849     cout << "无效输入，请重新输入" << endl;
850 }
851 };

```

library.hpp

```

1  /**
2   * @file library.hpp
3   * @brief 提供Library类的定义，该类继承自GUI类，用于实现图书馆系统的核心功能。
4   * 该文件包含Library类的定义，该类扩展了GUI类，添加了检查和创建必要文件系统目录的功能，并且定义了一个
   枚举类型Choice，用于表示用户在图书馆系统中可以进行的操作。
5   */
6
7  #include "gui.hpp"
8
9  /**
10   * @enum Choice
11   * @brief 定义用户在图书馆系统中可以选择进行的操作。
12   * 该枚举包含了用户可以在图书馆系统中执行的所有操作，如添加、删除、搜索和编辑图书和用户，借阅和归还图
   书，查看借阅记录，查看热门图书和活跃用户，删除所有图书和用户，以及退出系统。
13   */
14  enum Choice

```

```

15 {
16     AddBook = 1,
17     DeleteBook,
18     SearchBook,
19     EditBook,
20     AddUser,
21     DeleteUser,
22     SearchUser,
23     EditUser,
24     BorrowBook,
25     ReturnBook,
26     BorrowRecord,
27     TenHotBooks,
28     TenActiveUsers,
29     DeleteAllBooks,
30     DeleteAllUsers,
31     Exit
32 };
33
34 /**
35  * @class Library
36  * @brief 图书馆系统的核心类，继承自GUI类。
37  * Library类继承自GUI类，提供了检查和创建图书馆系统所需的文件系统目录的功能。该类是图书馆系统的核心，
    通过继承GUI类，它也间接提供了用户界面和与用户交互的功能。
38  */
39 class Library : public GUI
40 {
41 public:
42     /**
43      * @brief 检查并创建图书和用户信息存储所需的目录。
44      * 该方法检查图书和用户信息存储所需的目录是否存在，如果不存在，则创建这些目录。这是图书馆系统启动时
    进行的初始化步骤之一。
45      */
46     void CheckDirectory() const
47     {
48         if (!filesystem::exists(FILESYSTEM_BOOK))
49             filesystem::create_directories(FILESYSTEM_BOOK);
50         if (!filesystem::exists(FILESYSTEM_USER))
51             filesystem::create_directories(FILESYSTEM_USER);
52     }
53 };

```

main.cpp

```

1  /**
2  * @file main.cpp

```

```

3  * @brief 图书馆系统的入口点。
4  * 该文件包含main函数，是图书馆系统的入口点。它初始化Library类的实例，并进入一个循环，不断显示菜单、获取用户输入，并根据输入执行相应的操作。如果用户输入无效，将显示错误信息并重新显示菜单。
5  */
6
7  #include "library.hpp"
8
9  /**
10   * @brief 程序的主入口点。
11   * 主函数初始化图书馆系统，设置字符编码为UTF-8，检查必要的目录结构，并进入主循环，等待用户输入。根据用户的选择，执行相应的操作，直到用户选择退出程序。
12   * @return 程序退出状态。正常退出时返回0。
13   */
14  int main()
15  {
16      Library library;          // 图书馆系统的实例
17      bool error = false;       // 错误标志，用于指示是否需要显示错误信息
18      system("chcp 65001");     // 设置控制台字符编码为UTF-8
19
20      while (true) // 主循环
21      {
22          library.CheckDirectory(); // 检查并创建必要的目录结构
23          library.ShowMenu();       // 显示主菜单
24          if (error)                // 如果之前的输入无效，显示错误信息
25              library.Error();
26
27          string input;             // 用户输入
28          int choice;               // 用户选择的操作
29          getline(cin, input);     // 获取用户输入
30          // 验证输入是否为有效数字且在操作范围内
31          if (!IsPureNumber(input) ||
32              input.empty() ||
33              (choice = stoi(input)) > Exit || choice < AddBook)
34          {
35              error = true; // 设置错误标志
36              continue;     // 重新进入循环，显示菜单
37          }
38
39          // 根据用户选择执行相应操作
40          switch (choice)
41          {
42          case AddBook:
43              library.AddBook();
44              break;
45          case DeleteBook:
46              library.DeleteBook();
47              break;
48          case SearchBook:

```



```
49         library.SearchBook();
50         break;
51     case EditBook:
52         library.EditBook();
53         break;
54     case AddUser:
55         library.AddUser();
56         break;
57     case DeleteUser:
58         library.DeleteUser();
59         break;
60     case SearchUser:
61         library.SearchUser();
62         break;
63     case EditUser:
64         library.EditUser();
65         break;
66     case BorrowBook:
67         library.BorrowBook();
68         break;
69     case ReturnBook:
70         library.ReturnBook();
71         break;
72     case BorrowRecord:
73         library.BorrowRecord();
74         break;
75     case TenHotBooks:
76         library.TenHotBooks();
77         break;
78     case TenActiveUsers:
79         library.TenActiveUsers();
80         break;
81     case DeleteAllBooks:
82         library.DeleteAllBooks();
83         break;
84     case DeleteAllUsers:
85         library.DeleteAllUsers();
86         break;
87     case Exit:
88         library.Exit(); // 退出程序
89         return 0;       // 正常退出
90     }
91     error = false; // 重置错误标志
92 }
93 }
```