# 19.图书系统项目（三）

## 1. book模块

### 1.1 Book类的定义

关于书籍类我们书籍的属性如下：


注意构造方法的实现对于某些字段可以不加入参数：

1. bookId 【这里后续会进行累加/自增操作，此时在构造Book对象的时候，并不知道当前ID】
2. isBorrowed 【任何一种书籍，默认都是未借出状态，此时可以不进行处理】
3. borrowCount 【借阅次数默认为0，此时可以不进行处理】

```
1   public class Book implements Comparable<Book>{
2       private int bookId;           //书id
3       private String title;         // 书名
4       private String author;        // 作者
5       private String category;      // 类别
6       private int publishYear;      // 出版年份
7       private boolean isBorrowed;   // 借阅状态
8       private int borrowCount;      // 借阅次数
9       private LocalDate shelfDate;       // 上架时间
10
```

```java
    // 构造函数，初始化图书对象  书籍ID、借阅状态和 借阅次数不用进行参数传递
    public Book(String title, String author, String category,
                int publishYear, LocalDate shelfDate) {
        this.title = title;
        this.author = author;
        this.category = category;
        this.publishYear = publishYear;
        this.isBorrowed = false;
        this.borrowCount = 0;
        this.shelfDate = shelfDate;
    }

    public int getBookId() {
        return bookId;
    }

    public void setBookId(int bookId) {
        this.bookId = bookId;
    }

    public String getTitle() {
        return title;
    }

    public void setTitle(String title) {
        this.title = title;
    }

    public String getAuthor() {
        return author;
    }

    public void setAuthor(String author) {
        this.author = author;
    }

    public String getCategory() {
        return category;
    }

    public void setCategory(String category) {
        this.category = category;
    }

    public int getPublishYear() {
        return publishYear;
    }
```

```
58
59      public void setPublishYear(int publishYear) {
60          this.publishYear = publishYear;
61      }
62
63      public boolean isBorrowed() {
64          return isBorrowed;
65      }
66
67      public void setBorrowed(boolean borrowed) {
68          isBorrowed = borrowed;
69      }
70
71      public int getBorrowCount() {
72          return borrowCount;
73      }
74
75      public void setBorrowCount(int borrowCount) {
76          this.borrowCount = borrowCount;
77      }
78
79      public void incrementBorrowCount() {
80          this.borrowCount++;
81      }
82
83      public void decreaseBorrowCount() {
84          this.borrowCount--;
85      }
86
87      public LocalDate getShelfDate() {
88          return shelfDate;
89      }
90
91      public void setShelfDate(LocalDate shelfDate) {
92          this.shelfDate = shelfDate;
93      }
94
95      @Override
96      public String toString() {
97          return "Book{" +
98                  "bookId='" + bookId + '\'' +
99                  ",title='" + title + '\'' +
100                 ", author='" + author + '\'' +
101                 ", category='" + category + '\'' +
102                 ", publishYear=" + publishYear +
103                 ", isBorrowed=" + isBorrowed +
104                 ", borrowCount=" + borrowCount +
```

```
                ", shelfDate=" + shelfDate +
                '}';
    }

    @Override
    public int compareTo(Book o) {
        return o.getBorrowCount()-this.getBorrowCount();
    }
}
```

这里我们默认实现了 `Comparable` 接口，对后期对象的比较做一个准备工作。

## 1.2 Library类的定义

### 1.2.1 Library中的设计思路

我们期望书籍等数据最好可以进行持久化，所谓持久化就是将数据存储到MySQL、文件等介质中，不会因为断电等情况影响数据。否则每次运行程序数据都需要重写进行存储等比较繁琐。由于目前知识储备，我们还没有学习到MySQL和文件，我们这里会采用文件进行存储，把文件相关操作封装到 `jar` 包，借助已经写好的工具来进行操作。我们这里先根据《17.图书系统项目（一）》中把给定的jar包进行导入项目。接下来我们来看整体的设计流程。

实例化 `Library` -》加载文件当中的内容-》对文件进行操作-》将操作后的数据写入文件


### 1.2.2 `AnalyzingBook` 类实现文件内容的写入

在util包下创建 `AnalyzingBook` 类：实现的功能如下

```
public class AnalyzingBook {

    public void storeObject(Book[] books, String filename) throws IOException
{

    }

    public Book[] loadObject(String filename) throws IOException {
        //从文件读取数据

    }

```

```
12
13      private Book parseBookJson(String json) {
14
15      }
16  }
```

现有一组书籍进行文件的存储

```
1   Book[] books = new Book[4];
2   books[0] = new Book("java", "gaobo", "编程", 1994, LocalDate.of(2023, 9, 24));
3   books[1] = new Book("mysql", "lisi", "编程", 1999, LocalDate.of(2024, 2, 10));
4   books[2] = new Book("php", "gaobo", "编程", 2020, LocalDate.of(2023, 9, 23));
5   books[3] = new Book("西游记", "吴承恩", "小说", 2024, LocalDate.of(2023, 9,
    23));
```

当存入文件之后，文件一行代表一本书籍的书籍对象。写入书籍思路：

1. 实现 `AnalyzingBook` 类
2. 先将书籍数组对象中的每个书籍对象进行字符串的序列化。
3. 每本书籍与书籍之间使用 `\n` 进行换行分割

> books 代表书籍数组对象
>
> filename 代表文件名称【如果不存在该文件，会自动创建该文件】

```
1   public void storeObject(Book[] books, String filename) throws IOException {
2       //先遍历books数组当中不为空的数据多少个？
3       int booksUseLen = 0;
4       for (int i = 0; i < books.length; i++) {
5           if(books[i] != null) {
6               booksUseLen++;
7           }
8       }
9       StringBuilder jsonArray = new StringBuilder();
10      for (int i = 0; i < booksUseLen; i++) {
11          if(books[i] != null) {
12              jsonArray.append(books[i].toJSON());
13              if (i != booksUseLen-1) {
14                  //一本书籍完成后以\n进行分割
15                  jsonArray.append("\n");
16              }
17          }
18      }
```

```
        //数据写入文件
        FileUtils.writeFile(jsonArray.toString(),filename);
    }
```

注意：

- 上述代码当中toJSON需要我们自己实现
- FileUtils中的writeFile方法已经实现好，可以直接使用

### 1.2.3 BOOk类中toJson的实现

```
1,java,gaobo,编程,1994,true,1,2023-09-24
2,mysql,lisi,编程,1999,false,0,2024-02-10
3,PHP,gaobo,编程,2020,true,1,2023-09-23
4,西游记,吴承恩,小说,2024,false,0,2023-09-23
5,图书测试,author,测试类别,2024,false,0,2024-10-10
```

在Book类当中实现 `toJson` 方法，实现逻辑是将每个属性使用逗号进行拼接，存储到文件即可

```
public String toJSON() {
    StringBuilder json = new StringBuilder();
    json.append(bookId).append(",");
    json.append(title).append(",");
    json.append(author).append(",");
    json.append(category).append(",");
    json.append(publishYear).append(",");
    json.append(isBorrowed).append(",");
    json.append(borrowCount).append(",");
    json.append(shelfDate != null ?
shelfDate.format(DateTimeFormatter.ISO_LOCAL_DATE) : "null");
    return json.toString();
}
```

### 1.2.4 `AnalyzingBook` 类实现文件内容的读取

读取文件的每一行字符串，转化为书籍对象。

1. 先读取文件当中的所有内容
2. 使用 `\n` 作为分隔符进行字符串分割，方便得到每一个书籍对象对应的字符串
3. 把对应字符串“组装”成书籍对象

```java
public Book[] loadObject(String filename) throws IOException {
    //1. 从文件读取数据
    String content = FileUtils.readFile(filename);
    if (content == null || content.isEmpty()) {
        System.out.println("File is empty or does not exist: " + filename);
        return null;
    }
    //2. 使用\n作为分隔符进行字符串分割
    String[] bookJsonStrings = content.split("\n");

    //3. 把对应字符串“组装”成书籍对象
    Book[] bookList = new Book[bookJsonStrings.length];
    for (int i = 0;i < bookJsonStrings.length;i++) {
        Book book = parseBookJson(bookJsonStrings[i]);
        bookList[i] = book;
    }

    return bookList;
}
```

**`parseBookJson` 方法实现**

1. 检查json是否是空的
2. 使用逗号分割出每个属性的值
3. 每个属性进行转化赋值
4. 构造书籍对象

```java
private Book parseBookJson(String json) {
    //1. 检查json是否是空的
    if(json.isEmpty()) {
        return null;
    }
    //2. 使用逗号分割出每个属性的值
    String[] pairs = json.split(",");
    //3. 每个属性进行转化赋值
    int bookId = Integer.parseInt(pairs[0]);
    String title = pairs[1];
    String author = pairs[2];
    String category = pairs[3];
    int publishYear = Integer.parseInt(pairs[4]);
    boolean isBorrowed = Boolean.parseBoolean(pairs[5]);
    int borrowCount = Integer.parseInt(pairs[6]);
    LocalDate shelfDate = LocalDate.parse(pairs[7]);
    //4. 构造书籍对象
```

```
        if (title != null && author != null && category != null && shelfDate != null) {
            Book book = new Book(title, author, category, publishYear, shelfDate);
            book.setBorrowed(isBorrowed);
            book.setBorrowCount(borrowCount);
            book.setBookId(bookId);
            return book;
        }
        return null;
    }
```

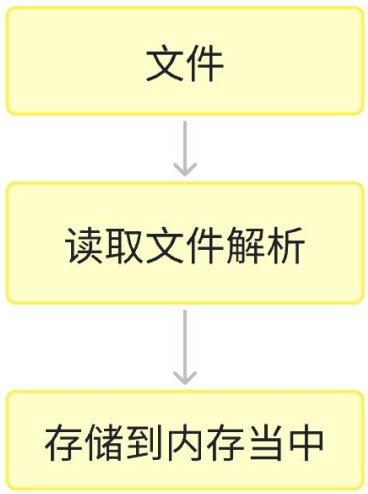
**测试Test**

```
public static void main(String[] args) throws IOException {
        AnalyzingBook myProperties = new AnalyzingBook();

        // 存储数据
        Book[] books = new Book[4];
        books[0] = new Book("java", "gaobo", "编程", 1994, LocalDate.of(2023, 9, 24));
        books[1] = new Book("mysql", "lisi", "编程", 1999, LocalDate.of(2024, 2, 10));
        books[2] = new Book("php", "gaobo", "编程", 2020, LocalDate.of(2023, 9, 23));
        books[3] = new Book("西游记", "吴承恩", "小说", 2024, LocalDate.of(2023, 9, 23));
        myProperties.storeObject(books, Constant.ALL_BOOK_FILE_NAME);

        // 读取数据
        Book[] loadedBooks = myProperties.loadObject(Constant.ALL_BOOK_FILE_NAME);
        if (loadedBooks != null) {
            System.out.println("Loaded books:");
            for (Book book : loadedBooks) {
                System.out.println(book);
            }
        }
    }
```



## 1.2.5 Library文件&读取存储具体细节设计

### 1.2.5.1 文件内容读取到内存中进行存储




既然我们要把解析的数据存储到内存当中，那么我们需要在内存当中定义一个数组来存储读取到的书籍对象。

**定义Library类**

```
public class Library {
    private Book[] books;//当前图书数组
    private int bookCount;//实际存储的图书数量

    public Library() {
        //当调用该构造方法的时候，要加载文件当中的数据进行到books数组当中
    }
}
```

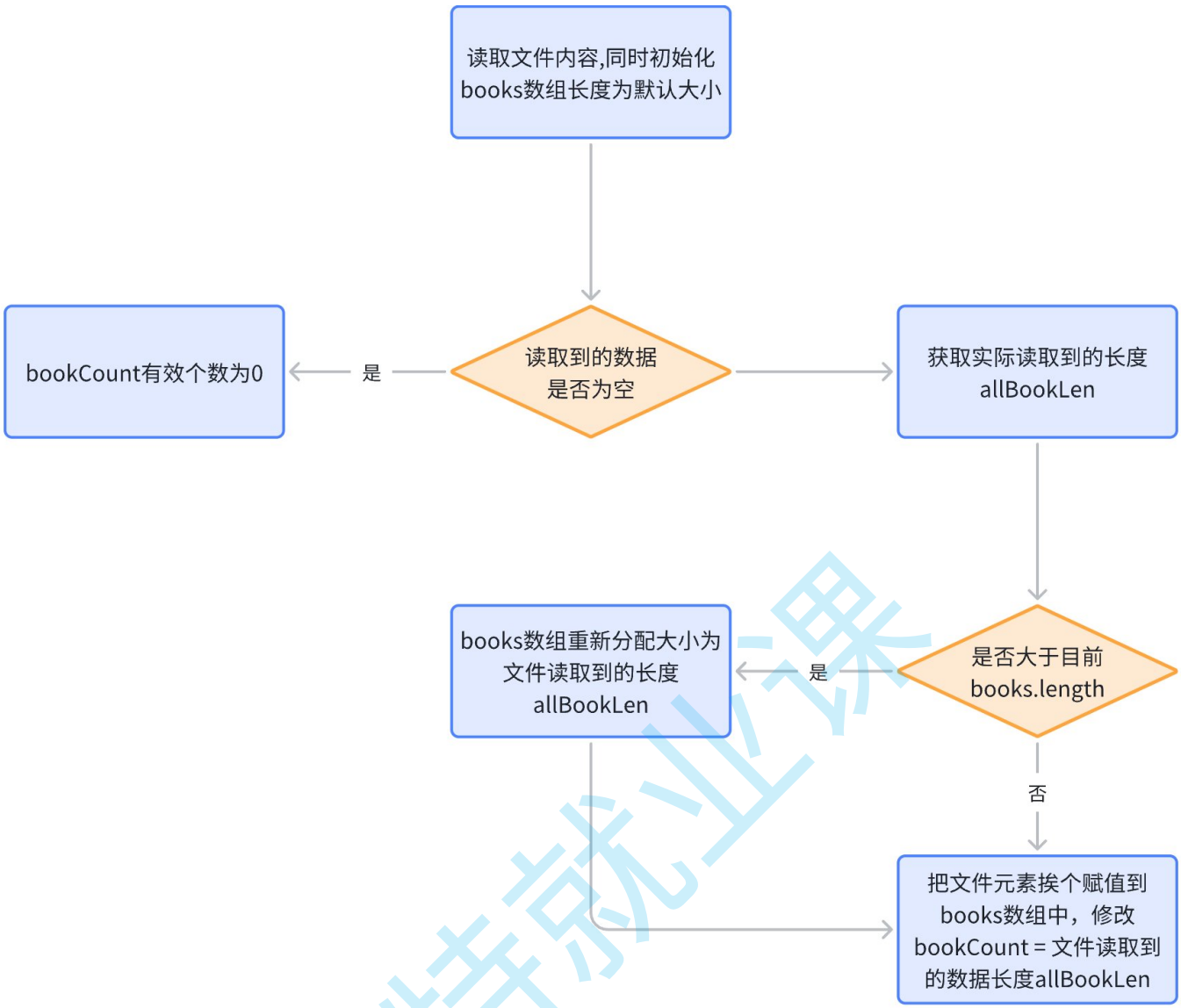
那么问题来了，books数组初始化多大合适呢？我们的思路是：

1. 默认数组容量给一个默认值，比如 **5**
2. 然后根据实际读取到的文件内容，来确定大小

这个默认值是一个常量，我们在 `constant` 包下定义一个常量类 `Constant` 来存储这些常量

```
public class Constant {
    //内存中的书籍数组初识容量
    public static final int CAPACITY = 5;

}
```

**实现文件读取内容的存储**




代码实现如下：

```
1   private void loadAllBook() {
2       try {
3           //1. 读取文件内容
4           Book[] allBook =
    analyzingBook.loadObject(Constant.ALL_BOOK_FILE_NAME);
5           //默认大小为5
6           books = new Book[Constant.CAPACITY];
7           //2. 是否有数据 没有数据 有效书籍个数为 0
8           if(allBook == null) {
9               bookCount = 0;
10          }else {
11              //3. 查看实际书籍长度是多少 是否大于默认的长度5
12              int allBookLen = allBook.length;
13              //3.1 大于默认长度 books数组 分配实际的大小
14              if (allBookLen > books.length) {
```

```
                //按照实际情况进行分配数组内存
                books = new Book[allBookLen];
            }
            //3.2 把读到的元素进行赋值
            for (int i = 0; i < allBookLen; i++) {
                books[i] = allBook[i];
            }
            //4.修改实际有效书籍个数
            bookCount = allBookLen;
        }
    } catch (IOException e) {
        throw new RuntimeException(e);
    }
}
```

> analyzingBook：为成员变量AnalyzingBook的引用

#### 1.2.5.2 内存中的书籍数组进行存储到文件

```
//存储图书到文件中
private void storeBook() {
    try {
        analyzingBook.storeObject(books,Constant.ALL_BOOK_FILE_NAME);
    } catch (IOException e) {
        throw new RuntimeException(e);
    }
}
```

> analyzingBook：为成员变量AnalyzingBook的引用

**常量类添加常量：Constant.ALL_BOOK_FILE_NAME**

```
public class Constant {

    //内存中的书籍数组初识容量
    public static final int CAPACITY = 5;

    //存储所有图书的文件
    public static final String ALL_BOOK_FILE_NAME = "allbook.txt";
}
```

什么情况下，我们要定义常量?

1. 当有一个值，它在程序运行期间不会改变，并且需要在多个地方使用。
2. 当需要表示某些固定的值，比如常数，错误代码、颜色代码、消息类型等。

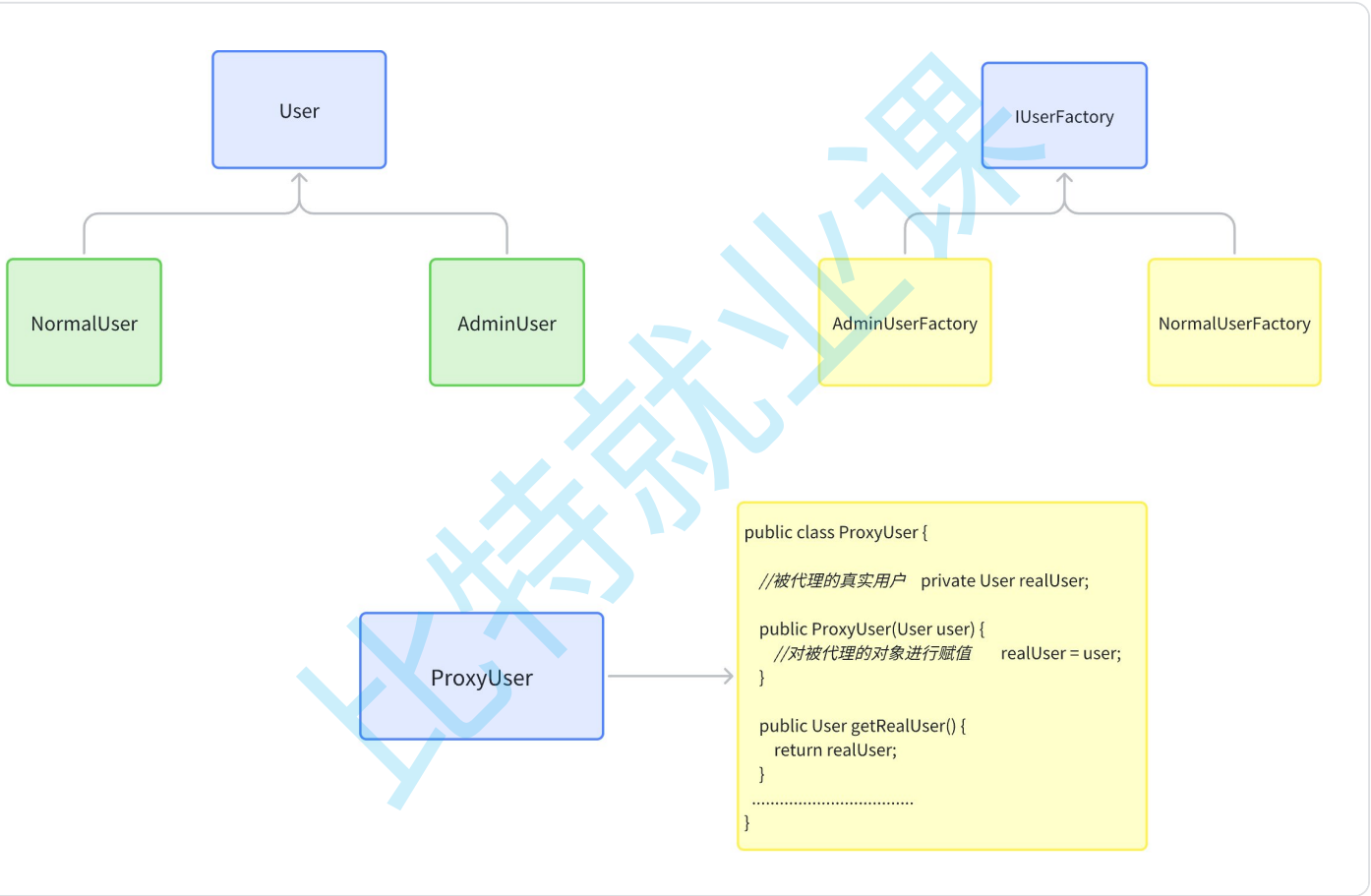
#### 1.2.5.3 Library整合文件读取&存储

```
public class Library {
    private Book[] books;//当前图书数组
    private int bookCount;//实际图书数量

    //书籍解析相关类
    private final AnalyzingBook analyzingBook = new AnalyzingBook();

    public Library() {
        loadAllBook();
    }

    //读取文件
    private void loadAllBook() {
        try {
            //1. 读取文件内容
            Book[] allBook =
analyzingBook.loadObject(Constant.ALL_BOOK_FILE_NAME);
            books = new Book[Constant.CAPACITY];
            //2. 是否有数据 没有数据 有效书籍个数为 0
            if(allBook == null) {
                bookCount = 0;
            }else {
                //3. 查看实际书籍长度是多少 是否大于默认的长度5
                int allBookLen = allBook.length;
                //3.1 大于默认长度 books数组 分配实际的大小
                if (allBookLen > books.length) {
                    //按照实际情况进行分配数组内存
                    books = new Book[allBookLen];
                }
                //3.2 把读到的元素进行赋值
                for (int i = 0; i < allBookLen; i++) {
                    books[i] = allBook[i];
                }
                //4.修改实际有效书籍个数
                bookCount = allBookLen;
            }
        } catch (IOException e) {
            throw new RuntimeException(e);
        }
    }
    //存储图书到文件中
```

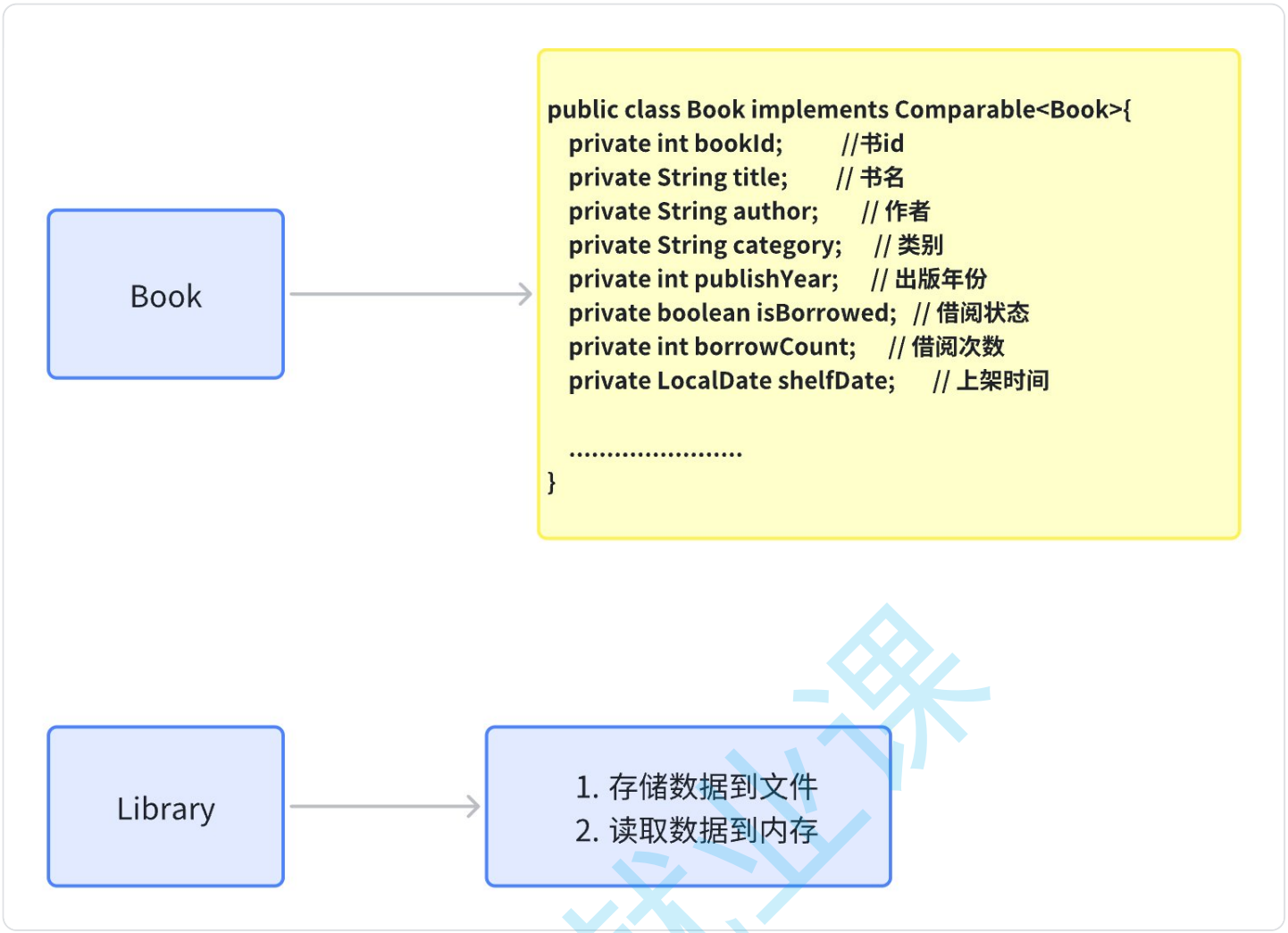
```
41      private void storeBook() {
42          try {
43              analyzingBook.storeObject(books,Constant.ALL_BOOK_FILE_NAME);
44          } catch (IOException e) {
45              throw new RuntimeException(e);
46          }
47      }
48  }
```

## 1.3 整体回顾当前项目框架

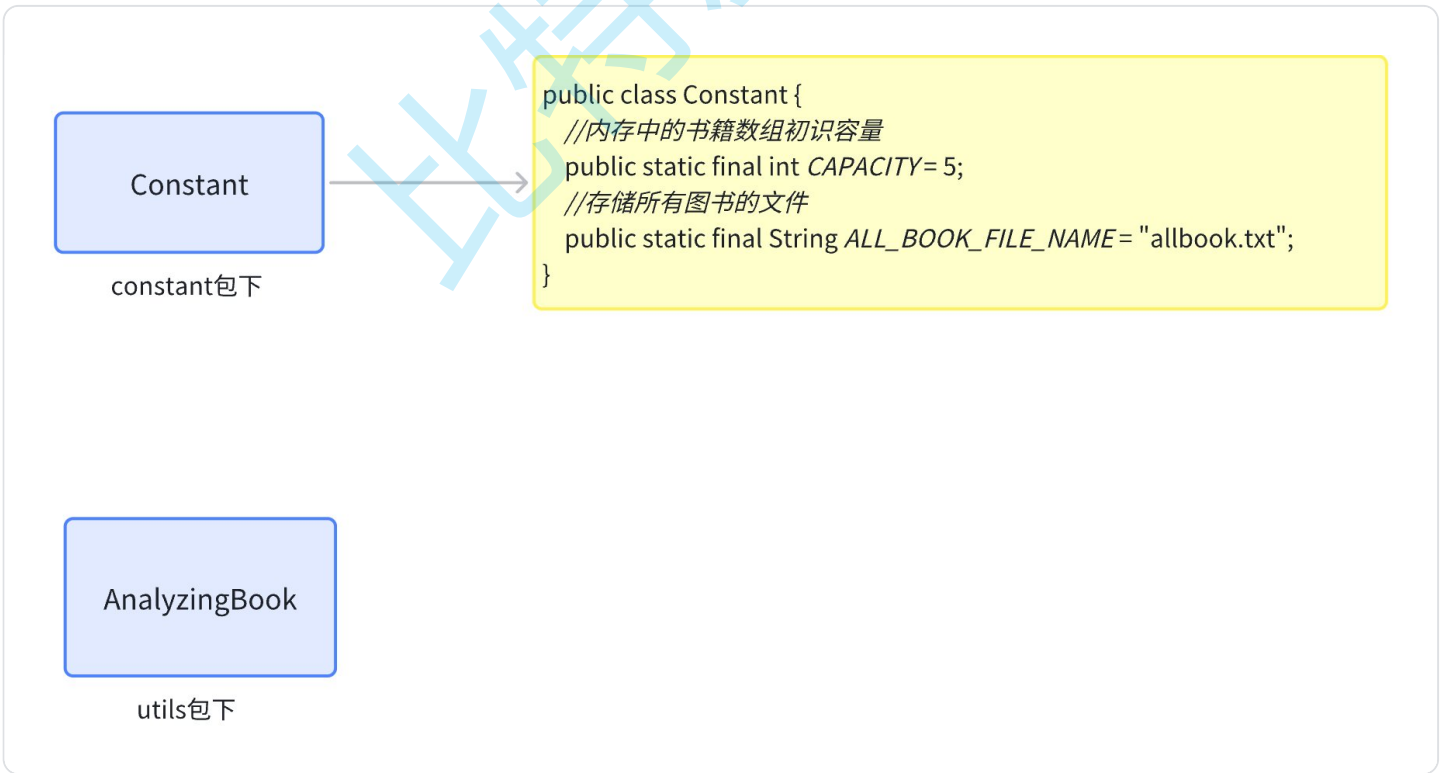
### 1.3.1 用户相关




### 1.3.2 书籍相关




### 1.3.3 其他模块




## 1.4 业务操作逻辑框架分解

接下来我们需要完成业务逻辑的实现，首先我们需要对框架进行梳理，在正式梳理之前，我们需要完善遗留工作：根据用户不同选择进入不同等级的用户角色下！

#### 1.4.1.1 完善 `LibrarySystem` 类

**在上一节中，我们已经完成了用户逻辑的整理：**

`LibrarySystem` 类中进行整合

```
public class LibrarySystem{
    public static void main(String[] args) {

        IUserFactory adminUserFactory = new AdminUserFactory();
        User adminUser = adminUserFactory.createUser("刘备",1);

        IUserFactory normalUserFactory = new NormalUserFactory();
        User normalUser1 = normalUserFactory.createUser("关羽",2);
        User normalUser2 = normalUserFactory.createUser("张飞",3);
        /**
        * 1.4 使用代理模式来管理权限
        * 使用代理模式来控制 对象的访问
        */

        ProxyUser proxyUserAdmin = new ProxyUser(adminUser);
        ProxyUser proxyUserNormalG = new ProxyUser(normalUser1);
        ProxyUser proxyUserNormalZ = new ProxyUser(normalUser2);

    }
}
```

此时的 `proxyUserAdmin` 、 `proxyUserNormalG` 、 `proxyUserNormalZ` ，是我们的代理对象，我们需要根据用户的选择，来确定最终的角色。

在 `LibrarySystem` 类中实现 `selectProxyRole` 方法：

```
//选择对应角色进行登录
public  ProxyUser selectProxyRole(ProxyUser proxyUserAdmin,
                                  ProxyUser proxyUserNormalW,ProxyUser
proxyUserNormalL) {
    System.out.println("选择角色进行登录：");
    System.out.println("1.管理员\n2.普通用户(关羽)\n3.普通用户(张飞)\n4.退出系统");
    ProxyUser currentUser = null;

    Scanner scanner = new Scanner(System.in);

```

```
        int choice = scanner.nextInt();
        switch (choice) {
            case 1:
                currentUser = proxyUserAdmin;
                break;
            case 2:
                currentUser = proxyUserNormalW;
                break;
            case 3:
                currentUser =  proxyUserNormalL;
                break;
            case 4:
                System.exit(0);
                System.out.println("系统已退出..");
                break;
            default:
                break;
        }
        return currentUser;
    }
```

完善 `LibrarySystem` 类：

```
public static void main(String[] args) {

    IUserFactory adminUserFactory = new AdminUserFactory();
    User adminUser = adminUserFactory.createUser("刘备",1);

    IUserFactory normalUserFactory = new NormalUserFactory();
    User normalUser1 = normalUserFactory.createUser("关羽",2);
    User normalUser2 = normalUserFactory.createUser("张飞",3);

    /**
    * 1.4 使用代理模式来管理权限
    * 使用代理模式来控制 对象的访问
    */

    ProxyUser proxyUserAdmin = new ProxyUser(adminUser);
    ProxyUser proxyUserNormalG = new ProxyUser(normalUser1);
    ProxyUser proxyUserNormalZ = new ProxyUser(normalUser2);


    LibrarySystem librarySystem = new LibrarySystem();

    //选择对应角色进行登录
```

```
        ProxyUser currentUser =
librarySystem.selectProxyRole(proxyUserAdmin,proxyUserNormalG,proxyUserNormalZ
);


        while (true) {

            int choice = currentUser.getRealUser().display();

            //此时无需关系是 管理员还是普通用户，代理类会做权限判断
            currentUser.handleOperation(choice);

        }

    }
```

`handleOperation` 方法会在 `ProxyUser` 类中进行实现

#### 1.4.1.2 业务框架搭建

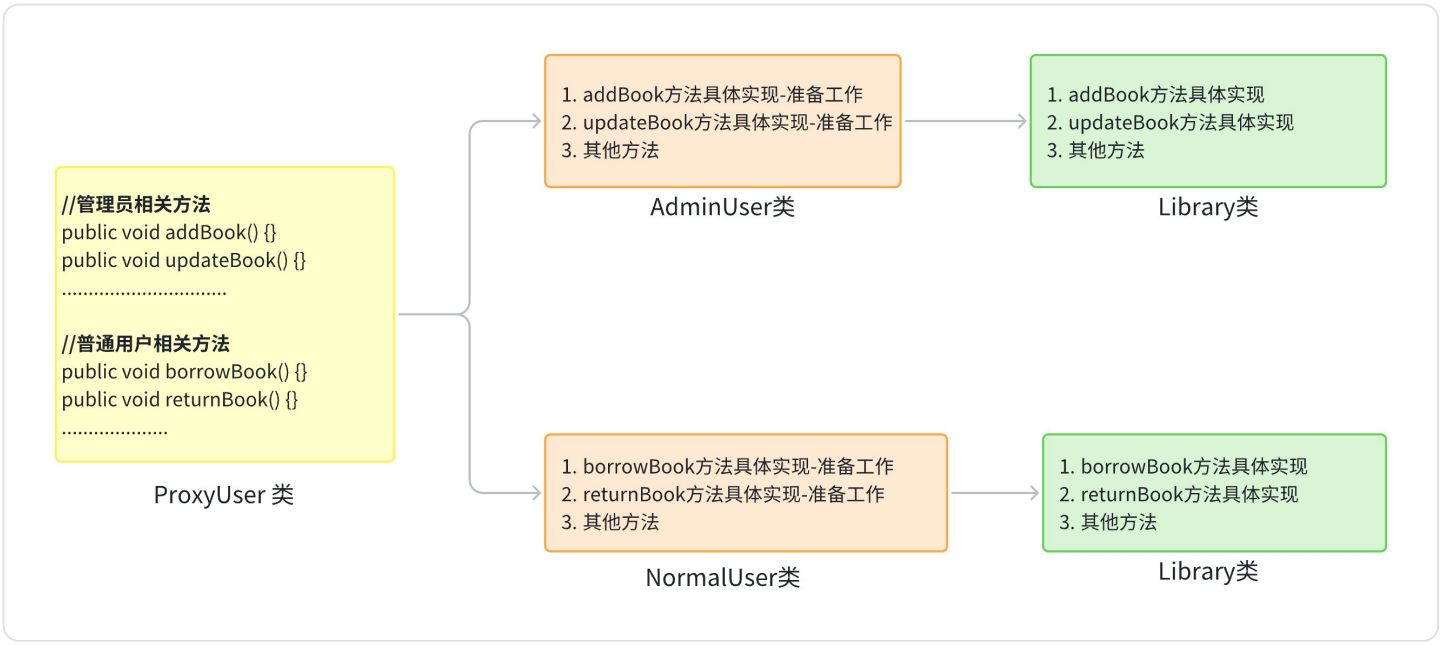
**管理员的业务逻辑有**

- 查找图书
- 显示图书
- 退出系统
- 上架图书
- 更新图书
- 下架图书
- 查看图书的借阅次数
- 查看最受欢迎的前K本书
- 查看库存状态
- 按照类别 统计图书
- 按照作者 统计图书
- 移除上架超过1年的书籍

**普通用户的逻辑有**

- 查找图书
- 显示图书
- 退出系统
- 借阅图书
- 归还图书
- 查看个人借阅情况

整个业务逻辑从 `ProxyUser` 类进行管理触发，整个框架为：


//管理员相关方法
public void addBook() {}
public void updateBook() {}
.....
//普通用户相关方法
public void borrowBook() {}
public void returnBook() {}
.....
ProxyUser 类
1. addBook方法具体实现-准备工作
2. updateBook方法具体实现-准备工作
3. 其他方法
AdminUser类
1. addBook方法具体实现
2. updateBook方法具体实现
3. 其他方法
Library类
1. borrowBook方法具体实现-准备工作
2. returnBook方法具体实现-准备工作
3. 其他方法
NormalUser类
1. borrowBook方法具体实现
2. returnBook方法具体实现
3. 其他方法
Library类

---

完