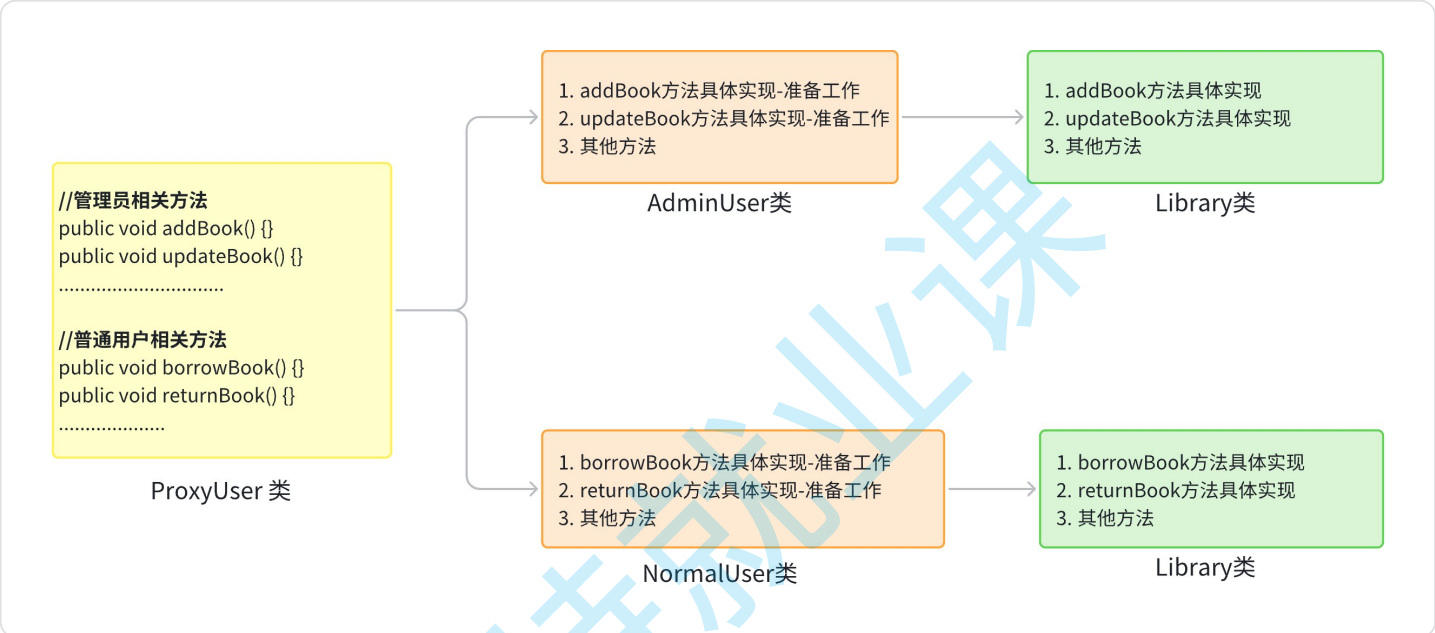
21.图书系统项目（五）

本节主要完成**管理员**业务逻辑具体实现

1. 回顾业务流程

我们前面已经分析过了，整个业务逻辑框架的实现流程如下：



2. 管理员端业务实现

2.1 添加书籍

检查权限是否合法-》调用管理员内部添加方法准备数据-》调用Libary方法进行业务具体实现

1. **ProxyUser类：**

```
1   /**
```

```
     * 检查当前代理用户是否为管理员
     */
    private void checkRealUserWhetherAdminUser(String exceptionMessage) {
        if(!(realUser instanceof AdminUser)){
            //自定义异常类，表示权限异常
             throw new PermissionException(exceptionMessage);
        }
    }

    //添加书籍操作
    public void addBook() {
        //查看代理的对象是是不是管理员
        checkRealUserWhetherAdminUser("普通用户没有权限上架图书");
        ((AdminUser) realUser).addBook();
    }
```

2. **AdminUser类：**

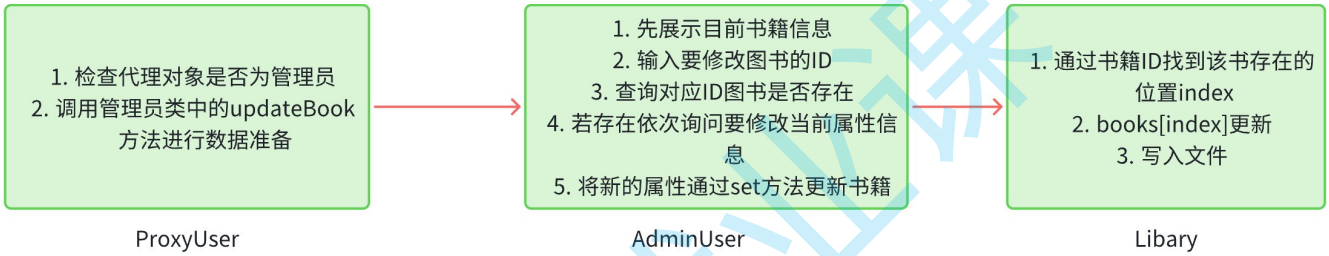
```
    //上架图书
    public void addBook() {
        scanner.nextLine();
        System.out.println("请输入书名：");
        String title = scanner.nextLine();  // 输入书名
        System.out.println("请输入作者：");
        String author = scanner.nextLine();  // 输入作者
        System.out.println("请输入类别：");
        String category = scanner.nextLine();  // 输入图书类别
        System.out.println("请输入出版年份：");
        int year = scanner.nextInt();  // 输入出版年份
        scanner.nextLine();  // 吞掉换行符

        LocalDate shelfDate = LocalDate.now();  // 当前时间作为上架时间
        Book newBook = new Book(title, author, category, year, shelfDate);  // 创建新书对象

        //调用图书类 添加图书
        library.addBook(newBook);
    }
```

3. **Library类**

```
    //添加图书
    public void addBook(Book book) {
```

```
        if(bookCount >= books.length) {
            System.out.println("图书馆已满，无法上架更多图书！");
            return;
        }
        //修改bookId为自增
        book.setBookId(bookCount+1);
        books[bookCount++] = book;
        //此时存储数据的时候 会把书籍对象全部存储，虽然部分属性没有输入赋值
        storeBook();
        System.out.println("图书 "+book.getTitle()+"上架成功！");
    }
```

2.2 更新书籍



1. ProxyUser类：

```
    //更新书籍操作
    public void updateBook() {
        checkRealUserWhetherAdminUser("普通用户没有权限更新图书");
        ((AdminUser) realUser).updateBook();
    }
```

2. AdminUser类：

```
    //图书修改 支持修改书名 作者 类别
    public void updateBook() {
        //1. 先展示一下目前的所有书籍
        library.displayBooks();
        System.out.println("请输入要修改的图书id: ");
        int bookId = scanner.nextInt();
        // 吞掉换行符
        scanner.nextLine();
        // 获取对应的图书
        Book book = library.searchById(bookId);
```

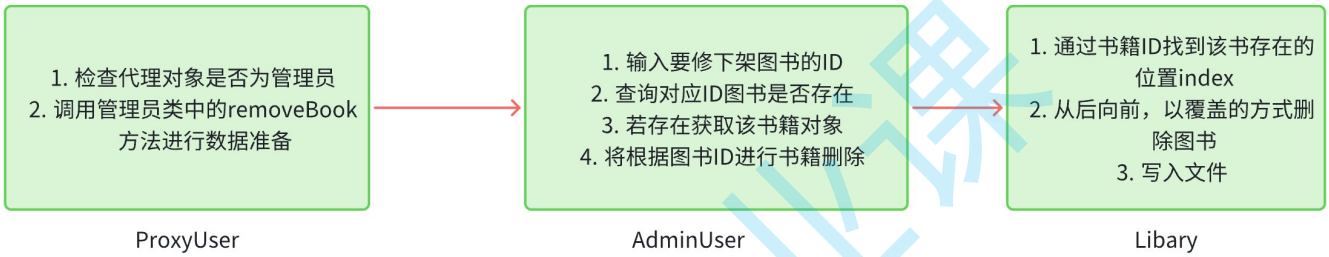
```
        if(book == null) {
            System.out.println("没有ID为："+bookId+" 的书籍！");
            return;
        }
        System.out.println("当前书名：" + book.getTitle());
        System.out.println("请输入新的书名：");
        String newTitle = scanner.nextLine();  // 输入新的书名
        System.out.println("当前作者：" + book.getAuthor());
        System.out.println("请输入新的作者：");
        String newAuthor = scanner.nextLine();  // 输入新的作者
        System.out.println("当前类别：" + book.getCategory());
        System.out.println("请输入新的类别：");
        String newCategory = scanner.nextLine();  // 输入新的类别

        //更新对应书籍的信息
        book.setTitle(newTitle);
        book.setAuthor(newAuthor);
        book.setCategory(newCategory);

        library.updateBook(book);
    }
```

3. Library类

```
    //更新图书
    public void updateBook(Book book) {
        //先找到该书是哪个下标
        int index = searchByIdReturnIndex(book.getBookId());
        books[index] = book;
        //一定要进行存储
        storeBook();
    }
    //根据bookId 返回 书籍索引位置
    private int searchByIdReturnIndex(int bookId) {
        loadAllBook();
        for (int i = 0; i < getBookCount(); i++) {
            Book book = books[i];
            if(book.getBookId()==bookId) {
                return i;
            }
        }
        return -1;
    }

    //根据bookId 查找对应书籍
```

```java
public Book searchById(int bookId) {
    loadAllBook();
    for (int i = 0; i < getBookCount(); i++) {
        Book book = books[i];
        if(book.getBookId()==bookId) {
            return book;
        }
    }
    return null;
}
```

2.3 移除(下架)书籍



1. ProxyUser类：

```java
//移除图书
public void removeBook() {
    checkRealUserWhetherAdminUser("普通用户没有权限删除图书");
    ((AdminUser) realUser).removeBook();
}
```

2. AdminUser类：

```java
//删除书籍
public void removeBook() {
    //1.展示一下所有的图书
    library.displayBooks();
    System.out.println("请输入要删除的图书的ID: ");
    int bookId = scanner.nextInt();
    scanner.nextLine();  // 吞掉换行符

```

```
        //记录一下删除的图书对象
        Book removeBook =library.searchById(bookId);
        //开始删除
        library.removeBook(bookId);

        System.out.println("图书: "+removeBook.getTitle()+" 已经被删除! ");
    }
```

3. Library类

```
    //删除图书
    public void removeBook(int bookId) {

        int index = searchByIdReturnIndex(bookId);
        //开始删除 从当前位置的后边往前移动
        for (int i = index; i < bookCount-1; i++) {
            books[i] = books[i+1];
        }
        books[bookCount-1] = null;

        storeBook();

        bookCount--;
    }
```

2.4 查看图书借阅次数

1. ProxyUser类:

```
    //查看图书的借阅次数
    public void borrowCount( ) {
        checkRealUserWhetherAdminUser("普通用户没有权限查看图书的借阅次数");
        ((AdminUser) realUser).borrowCount();
```

```
5   }
```

2. **AdminUser类：**

```
1   //统计每本书的借阅次数
2   public void borrowCount() {
3       library.borrowCount();
4   }
```

3. **Library类**

```
1   //每本书的借阅次数
2   public void borrowCount() {
3       loadAllBook();
4       for (int i = 0; i < bookCount; i++) {
5           Book book = books[i];
6           System.out.println("书名:"+book.getTitle()+
7                   " 借阅次数: "+book.getBorrowCount());
8       }
9   }
```

2.5 查看最受欢迎的前K本书

1. **ProxyUser类：**

```
1   //查看最受欢迎的前K本书
2   public void generateBook() {
3       checkRealUserWhetherAdminUser("普通用户没有权限查看最受欢迎的前k本书");
4       ((AdminUser) realUser).generateBook();
5   }
```

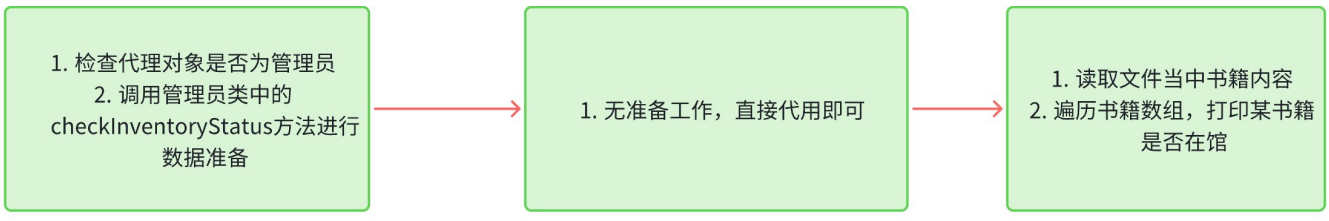
2. AdminUser类：

```
//查询最受欢迎的前n本书
public void generateBook() {
    System.out.println("请输入你要查看的最受欢迎的前K本书,注意k值不能超过: "+library.getBookCount());
    int k = scanner.nextInt();
    if(k <= 0 || k > library.getBookCount()) {
        System.out.println("没有最受欢迎的前"+k+"本书! ");
        return;
    }
    library.generateBook(k);
}
```

3. Library类

```
//查询最受欢迎的前n本书
public void generateBook(int k) {
    //1. 加载已有的全部的书籍
    loadAllBook();
    //2.把所有书籍放在 临时数据 进行排序
    Book[] tmp = new Book[getBookCount()];
    for (int i = 0; i < getBookCount(); i++) {
        tmp[i] = books[i];
    }
    //2.1 开始排序
    Arrays.sort(tmp);
    //3. 把前k本书拷贝到新数组    可以不定义临时数组，直接输出前K个就行
    Book[] generateBooks = new Book[k];

    for (int i = 0; i < k; i++) {
        generateBooks[i] = tmp[i];
    }
    //4.打印新数组
    System.out.println("最受欢迎书籍如下: ");
    for (int i = 0; i < generateBooks.length; i++) {
        Book book = generateBooks[i];
        System.out.println("索引: "+i+" 书名: "+ book.getTitle()+" 作者: "+
                book.getTitle()+" 借阅次数: "+book.getBorrowCount());
    }
}
```

2.6 查看库存状态



1. **ProxyUser类：**

```java
//查看库存状态
public void checkInventoryStatus() {
    checkRealUserWhetherAdminUser("普通用户没有权限查看库存状态");
    ((AdminUser) realUser).checkInventoryStatus();
}
```

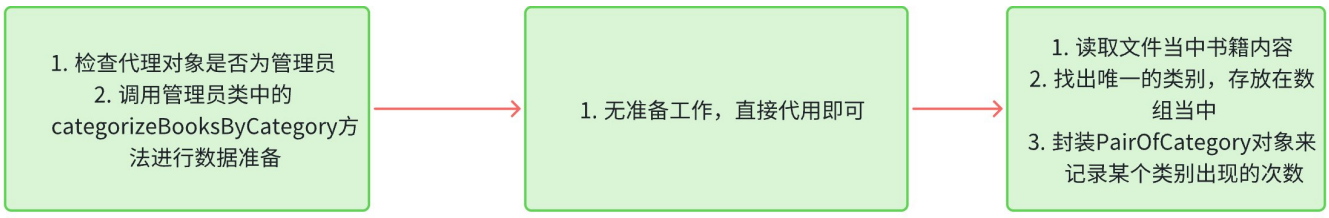
2. **AdminUser类：**

```java
//查看库存状态
public void checkInventoryStatus() {
    library.checkInventoryStatus();
}
```

3. **Library类**

```java
//查看库存状态
public void checkInventoryStatus() {
    loadAllBook();
    for (int i = 0; i < bookCount; i++) {
        Book book = books[i];
        String status = book.isBorrowed() ? "已借出" : "在馆";
        System.out.println("书名: "+book.getTitle()+" 目前状态: "+status);
    }
}
```

2.7 按照类别统计图书



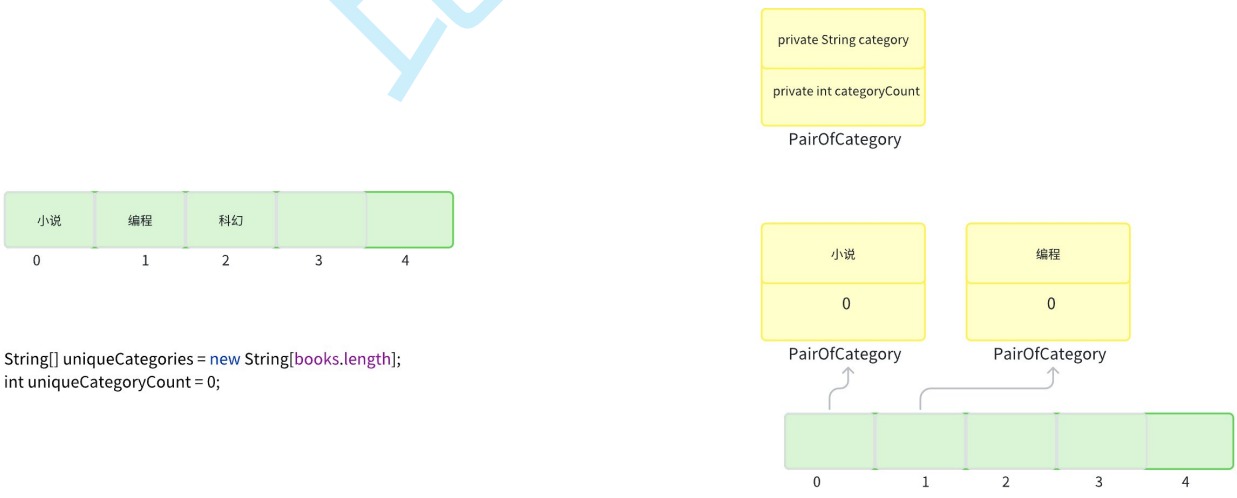
1. ProxyUser类：

```
1   //按照类别  统计图书
2   public void categorizeBooksByCategory() {
3       checkRealUserWhetherAdminUser("普通用户没有权限查看按照类别 统计图书");
4       ((AdminUser) realUser).categorizeBooksByCategory();
5   }
```

2. AdminUser类：

```
1   //按照类别 统计图书
2   public void categorizeBooksByCategory() {
3       library.categorizeBooksByCategory();
4   }
```

3. Library类



- 定义 `String[] uniqueCategories = new String[books.length];` 数组存储每种类别，这里存储结果是唯一的。
- 在book包中定义 `PairOfCategory` 类对象，定义2个属性，分别代表类别和数量

- 定义 `PairOfCategory` 数组对象，存储类别和次数的情况

`PairOfCategory` 类的定义：

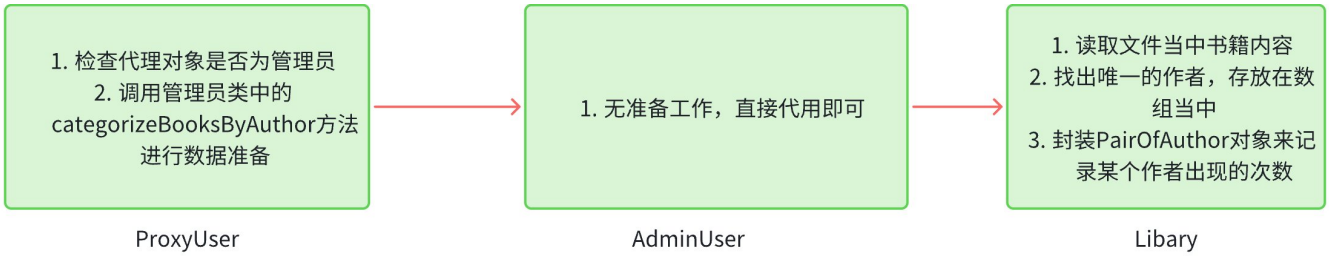
```
public class PairOfCategory {
    //类别
    private String category;
    //该类别的数量
    private int categoryCount;
    //构造方法
    public PairOfCategory(String category, int categoryCount) {
        this.category = category;
        this.categoryCount = categoryCount;
    }

    public int getCategoryCount() {
        return categoryCount;
    }

    public void setCategoryCount(int categoryCount) {
        this.categoryCount = categoryCount;
    }

    public String getCategory() {
        return category;
    }

    public void setCategory(String category) {
        this.category = category;
    }
    //数量自增方法
    public void incrementCount() {
        this.categoryCount++;
    }
}
```

方法实现：

```
//按照类别 统计图书
public  void categorizeBooksByCategory() {
    loadAllBook();
    // 第一步：找出所有唯一的类别
    String[] uniqueCategories = new String[books.length];
    //不重复的类别的个数 相当于uniqueCategories 数组的下标
    int uniqueCategoryCount = 0;
```

```
8
9       for (int j = 0; j < getBookCount();j++) {
10          Book book = books[j];
11          String category = book.getCategory();
12          boolean found = false;
13          //遍历 uniqueCategories 数组 是否存在该类别 如果存在 判断下一步书
14          for (int i = 0; i < uniqueCategoryCount; i++) {
15              if (uniqueCategories[i].equals(category)) {
16                  found = true;
17                  break;
18              }
19          }
20          //如果这里是false说明没有进去循环，可以直接存放
21          if (!found) {
22              uniqueCategories[uniqueCategoryCount] = category;
23              //下标需要加加
24              uniqueCategoryCount++;
25          }
26      }
27      //这里完成后【编程，学习，小说】
28      // 第二步：创建categoryCount数组并计数
29      PairOfCategory[] categoryCounts = new PairOfCategory[uniqueCategoryCount];
30      //默认给每一种类型 设置为 0 比如：编程:0  小说:0
31      for (int i = 0; i < uniqueCategoryCount; i++) {
32          categoryCounts[i] = new PairOfCategory(uniqueCategories[i], 0);
33      }
34      //开始统计种类出现的次数
35      for (int j = 0; j < getBookCount();j++) {
36          //获取每一本书籍对象
37          Book book = books[j];
38          //获取该书籍对象的种类
39          String category = book.getCategory();
40          //假设category为小说，则遍历categoryCounts数组，找到小说，则自增
41          for (PairOfCategory categoryCount : categoryCounts) {
42              if (categoryCount.getCategory().equals(category)) {
43                  categoryCount.incrementCount();
44                  break;
45              }
46          }
47      }
48
49      // 打印结果
50      for (PairOfCategory pairOfCategory : categoryCounts) {
51          System.out.println(pairOfCategory.getCategory() + ": " +
    pairOfCategory.getCategoryCount());
52      }
53  }
```

2.8 按照作者统计图书



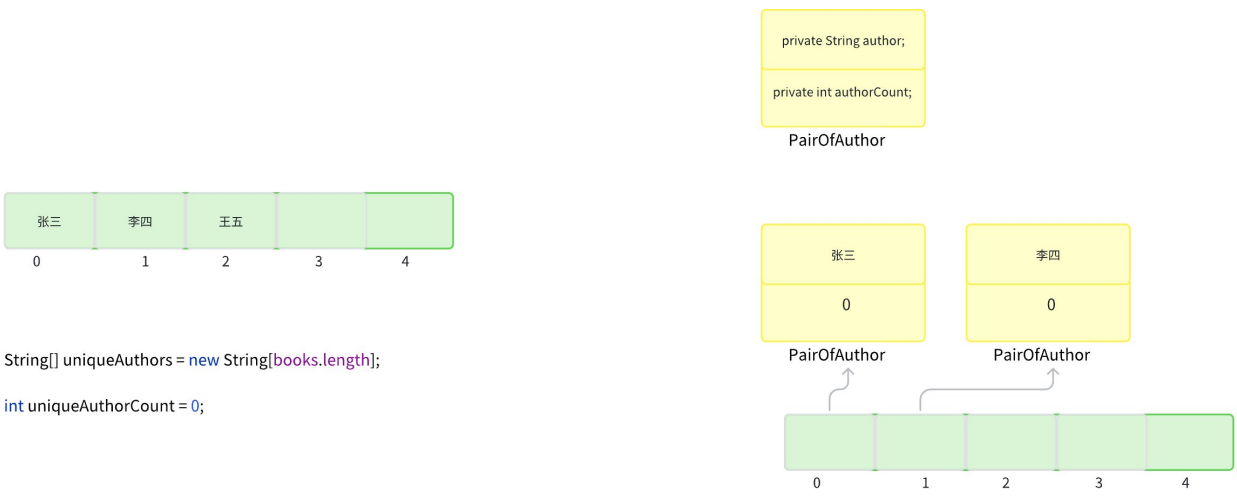
1. ProxyUser类：

```java
//按照作者 统计图书
public void categorizeBooksByAuthor() {
    checkRealUserWhetherAdminUser("普通用户没有权限查看按照作者统计图书");
    ((AdminUser) realUser).categorizeBooksByAuthor();
}
```

2. AdminUser类：

```java
//按照作者统计图书
public void categorizeBooksByAuthor() {
    library.categorizeBooksByAuthor();
}
```

3. Library类



- 定义 `String[] uniqueAuthors = new String[books.length];` 数组存储每种类别，这里存储结果是唯一的。
- 定义 `PairOfAuthor` 类对象，定义2个属性，分别代表类别和数量
- 定义 `PairOfAuthor` 数组对象，存储类别和次数的情况

`PairOfAuthor` 类的定义：

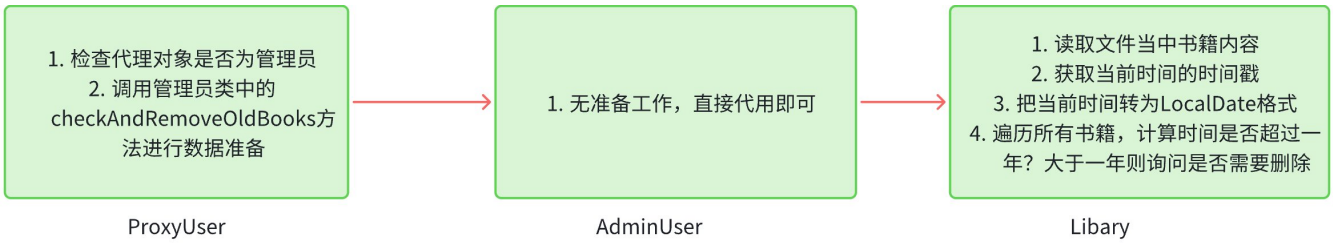
```
public class PairOfAuthor {
    //作者姓名
    private String author;
    //作者出现的次数
    private int authorCount;

    public PairOfAuthor(String author, int authorCount) {
        this.author = author;
        this.authorCount = authorCount;
    }

    public String getAuthor() {
        return author;
    }

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

    public int getAuthorCount() {
        return authorCount;
    }

    public void setAuthorCount(int authorCount) {
        this.authorCount = authorCount;
    }

    public void incrementCount() {
        this.authorCount++;
    }
}
```

```
//按照作者 统计图书
public void categorizeBooksByAuthor() {
```

```
        loadAllBook();
        // 第一步：找出所有唯一的类别
        String[] uniqueAuthors = new String[books.length];
        int uniqueAuthorCount = 0;

        //这里取唯一的作者，只要之前出现就不加了
        for (int j = 0; j < getBookCount();j++) {
            Book book = books[j];
            String author = book.getAuthor();
            boolean found = false;
            for (int i = 0; i < uniqueAuthorCount; i++) {
                if (uniqueAuthors[i].equals(author)) {
                    found = true;
                    break;
                }
            }
            //如果这里是false说明没有进去循环，可以直接存放
            if (!found) {
                uniqueAuthors[uniqueAuthorCount] = author;
                //下标自增
                uniqueAuthorCount++;
            }
        }

        // 第二步：创建authorCounts数组并计数  创建数组默认出现次数为0
        //[曹操：0    曹雪芹：0]
        PairOfAuthor[] authorCounts = new PairOfAuthor[uniqueAuthorCount];
        for (int i = 0; i < uniqueAuthorCount; i++) {
            authorCounts[i] = new PairOfAuthor(uniqueAuthors[i], 0);
        }

        //遍历书籍数组
        for (int j = 0; j < getBookCount();j++) {
            Book book = books[j];
            String author = book.getAuthor();
            //查看当前书籍作者在authorCounts数组中是否存在，存在则++
            for (PairOfAuthor pairOfAuthor : authorCounts) {
                if (pairOfAuthor.getAuthor().equals(author)) {
                    pairOfAuthor.incrementCount();
                    break;
                }
            }
        }

        // 打印结果
        for (PairOfAuthor pairOfAuthor : authorCounts) {
```

```
49          System.out.println(pairOfAuthor.getAuthor() + ": " +
    pairOfAuthor.getAuthorCount());
50      }
51  }
```

2.9 移除上架超过1年的书籍



1. ProxyUser类：

```
1   //移除上架超过1年的书籍
2   public void checkAndRemoveOldBooks() {
3       checkRealUserWhetherAdminUser("普通用户没有权限移除上架超过一年的图书");
4       ((AdminUser) realUser).checkAndRemoveOldBooks();
5   }
```

2. AdminUser类：

```
1   //并移除上架超过一年的图书
2   public void checkAndRemoveOldBooks() {
3       library.checkAndRemoveOldBooks(scanner);
4   }
```

3. Library类

```
1   //移除上架超过一年的图书
2   public void checkAndRemoveOldBooks(Scanner scanner) {
3       loadAllBook();
4       // 获取当前时间戳
5       long currentTimestamp = System.currentTimeMillis();
6
7       // 将当前时间戳转换为 LocalDate
```

```
8       LocalDate currentDate = Instant.ofEpochMilli(currentTimestamp)
9               .atZone(ZoneId.systemDefault())
10              .toLocalDate();
11
12
13      boolean flg = false;
14      for (int i = 0; i < getBookCount(); i++) {
15          Book book = books[i];
16          //获取当前书籍的上架时间
17          LocalDate specifiedDate = book.getShelfDate();
18
19          // 计算两个日期之间的差值（以年为单位）
20          long yearsBetween = ChronoUnit.YEARS.between(specifiedDate,
    currentDate);
21
22          if(yearsBetween >= 1) {
23              System.out.print("图书 " + book.getTitle() + " 已经上架超过一年，是否
    移除？ (y/n): ");
24              scanner.nextLine();
25              String response = scanner.nextLine();
26              if (response.equalsIgnoreCase("y")) {
27                  //确认删除调用remove方法进行删除
28                  removeBook(i);
29                  i--;  // 因为后面的书已经向前移动，所以要重新检查当前索引位置
30              }
31              flg = true;
32          }
33      }
34      if(!flg) {
35          System.out.println("没有上架超过一年的图书！");
36      }
37  }
```

1. `Instant.ofEpochMilli(currentTimestamp)` 方法的作用是将毫秒级时间戳转换为 Instant 对象(表示时间线上的一个瞬时点)
2. 将 Instant 对象（UTC时间）根据系统默认时区转换为带时区的 ZonedDateTime 对象。
3. toLocalDate() 将ZonedDateTime转换为LocalDate对象