使用 RPC 技术的一个基于 C/S 架构的书籍信息管理系统

一、 实验要求：

客户端实现用户交互，服务器端实现书籍信息存储和管理。客户端与服务器端利用 RPC 机制进行协作。中间件任选。

服务器端至少暴露如下 RPC 接口：

bool add(Book b)　添加一个书籍对象。

Book queryByID(int bookID) 查询指定 ID 号的书籍对象。

BookList queryByName(String name) 按书名查询符合条件的书籍对象列表，支持模糊查询。

bool delete((int bookID) 删除指定 ID 号的书籍对象。

二、 实验代码及代码分析：

Book 类的代码如下：

```
import java.io.Serializable;

public class Book implements Serializable {
    private int bookID;
    private String name;

    public Book(int bookID, String name) {
        this.bookID = bookID;
        this.name = name;
    }

    public String getName() {
        return name;
    }

    public int getID() {
        return bookID;
    }
}
```

在 Book 类中定义了 Book 对象所具有的的属性，如 ID 和 Name。

BookList 类的定义如下：

```
import java.io.Serializable;
import java.util.Vector;

public class BookList implements Serializable {
    Vector<Book> books;

    public BookList() {
        books = new Vector<Book>();
    }

    public BookList(Vector<Book> bs) {
        books = new Vector<Book>(bs);
    }

    public boolean add(Book book) {// 往存书处存书
        for (int i = 0; i < books.size(); i++) {
            if (book.getID() == books.get(i).getID()) {
                return false;
            }
        }
        return books.add(book);
    }

    public boolean delete(int BookID) {
        int flag = -1;
        for (int i = 0; i < books.size(); i++) {
            if (BookID == books.get(i).getID()) {
                flag = i;
                break;
            }
        }
        if (flag == -1) {
            return false;
        } else {
            books.remove(flag);
            return true;
```

```
        }

    }

    public Book queryByID(int BookID) {
        int flag = -1;
        for (int i = 0; i < books.size(); i++) {
            if (BookID == books.get(i).getID()) {
                flag = i;
                break;
            }
        }
        if (flag == -1) {
            return null;
        } else {
            return books.get(flag);
        }
    }

    public Vector<Book> queryByName(String name) {
        Vector<Book> result = new Vector<Book>();
        for (int i = 0; i < books.size(); i++) {
            if (books.get(i).getName().matches("(.*)" + name + "(.*)"))
{
                result.add(books.get(i));
            }
        }
        return result;
    }

    public int size() {
        return books.size();
    }

    public void dispaly() {
        if (books.size() == 0) {
            System.out.println("书库为空！");
        } else {
            for (int i = 0; i < books.size(); i++) {
                System.out.println(books.get(i).getID() + ":" +
books.get(i).getName());
            }
        }
    }
```

```
}
```

在该类中，我使用了 Vector 变长数组来作为数据结构来存储书籍（Book 对象）。另外，具体地实现了添加、删除、按 ID 查找以及按书名查找的功能。其中按照书名查找实现了模糊查找。

InterfaceImpl 类的定义如下：

```
import java.rmi.RemoteException;

public class InterfaceImpl implements myInterface{
    private BookList library;
    public InterfaceImpl(BookList Library) {
        library = Library;
    }

    @Override
    public boolean add(Book book) throws RemoteException {
        return library.add(book);
    }

    @Override
    public Book queryByID(int bookID) throws RemoteException {
        return library.queryByID(bookID);
    }

    @Override
    public BookList queryByName(String name) throws RemoteException {
        BookList bookList = new BookList(library.queryByName(name));
        return bookList;
    }

    @Override
    public boolean delete(int bookID) throws RemoteException {
        return library.delete(bookID);
    }
}
```

将 BookList 中的方法重载并封装，对外只展示接口而不展示实现细

节，将代码规范化、抽象化，便于代码的维护和修改以及用户的使用。

myInterface 类的定义如下：

```
import java.rmi.Remote;
import java.rmi.RemoteException;

public interface myInterface extends Remote {
    boolean add(Book book) throws RemoteException;// 添加一个书籍对象

    Book queryByID(int bookID) throws RemoteException;// 查询指定 ID 号的书籍对象

    BookList queryByName(String name) throws RemoteException;// 按书名查询书籍对象列表。

    boolean delete(int bookID) throws RemoteException;// 删除指定 ID 号的书籍对象。
}
```

该类中声明了四种功能方法，该类的作用是说明用户可以调用哪些方法。

RMIServer 类的定义如下：

```
import java.rmi.RemoteException;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
import java.rmi.server.UnicastRemoteObject;

public class RMIServer {

    public static void main(String[] args) throws RemoteException {

        try {
            String name = "Books";
            BookList Library = new BookList();// 存书的变长数组
            myInterface myInterface = new InterfaceImpl(Library);// 接口
```

```
            myInterface skeleton = (myInterface)
UnicastRemoteObject.exportObject(myInterface, 0);// 注入接口，生成
skeleton 对象
            Registry registry = LocateRegistry.createRegistry(7777);
            registry = LocateRegistry.getRegistry("127.0.0.1", 7777);//
获取注册中心的引用
            System.out.println("regitry Books object");
            registry.rebind(name, skeleton);
        } catch (Exception e) {
            System.err.println("Exception:" + e);
            e.printStackTrace();
        }

    }

}
```

该类实例化了一个 BookList 对象 Library，并用它来存储书籍，实例化了一个 myInterface 对象 skeleton，与客户端进行联系。声明了对端口 7777 的引用。

RMIClient 类的定义如下：

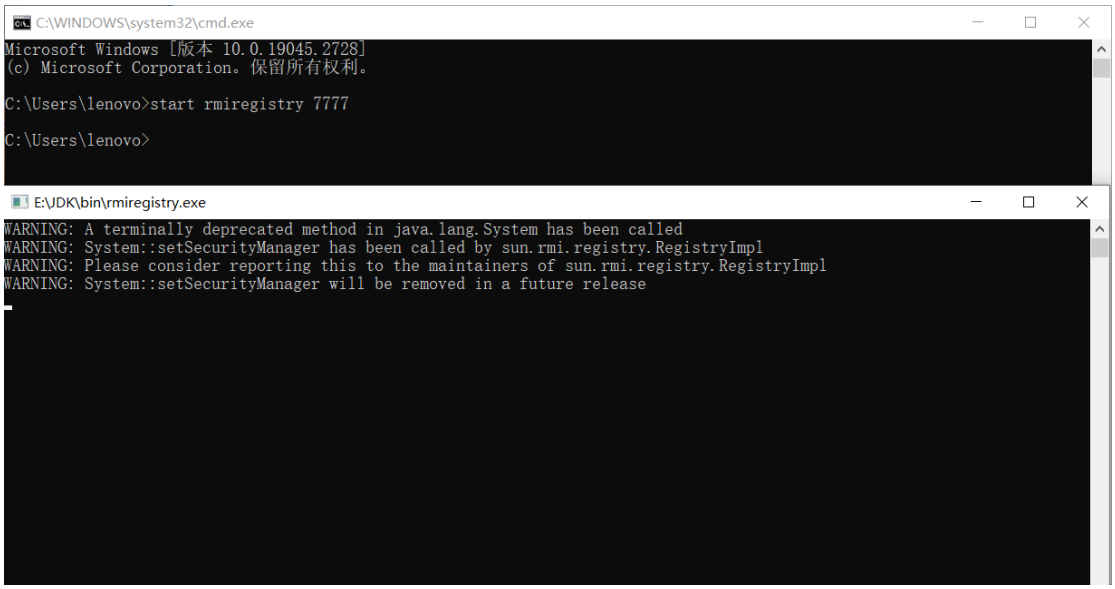
```
import java.rmi.NotBoundException;
//客户端交互
import java.rmi.RemoteException;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
import java.util.Scanner;

import javax.swing.text.PlainDocument;

public class RMIClient {

    public static void main(String[] args) throws RemoteException,
NotBoundException {
        // TODO Auto-generated method stub
        String name = "Books";
        String ServerIP = "127.0.0.1";
```

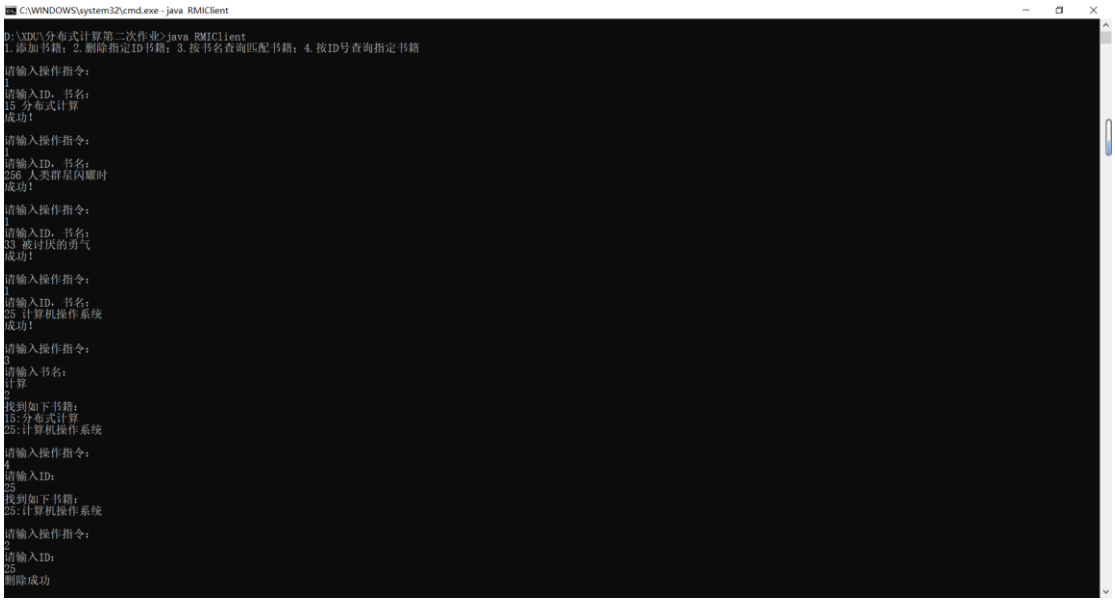
```
        int Serverport = 7777;
        Registry registry = LocateRegistry.getRegistry(ServerIP,
Serverport);// 获取注册中心的引用
        myInterface proxy = (myInterface) registry.lookup(name);
        int option;
        int BookID;
        String Bookname = null;
        BookList bookList;
        Book book;
        Scanner scanner = new Scanner(System.in);
        System.out.println("1.添加书籍；2.删除指定 ID 书籍；3.按书名查询匹配
书籍；4.按 ID 号查询指定书籍");
        while (true) {
            System.out.println("\n 请输入操作指令：");
            option = scanner.nextInt();//输入缓冲区问题未解决

            switch (option) {
            case 1:
                System.out.println("请输入 ID，书名：");
                BookID = scanner.nextInt();
                Bookname = scanner.next();
                if(proxy.add(new Book(BookID, Bookname))) {
                    System.out.println("成功！");
                }else {
                    System.out.println("ID 重复！");
                }
                break;
            case 2:
                System.out.println("请输入 ID：");
                BookID = scanner.nextInt();
                if (proxy.delete(BookID)) {
                    System.out.println("删除成功");
                } else {
                    System.out.println("查无此书！");
                }
                break;
            case 3:
                System.out.println("请输入书名：");
                Bookname = scanner.next();
                bookList = proxy.queryByName(Bookname);
                System.out.println(bookList.size());
                System.out.println("找到如下书籍：");
                bookList.dispaly();
                break;
```

```
            case 4:
                System.out.println("请输入 ID: ");
                BookID = scanner.nextInt();
                book = proxy.queryByID(BookID);
                System.out.println("找到如下书籍: ");
                if(book == null) {
                    System.out.println("没有相关书籍");
                }else {
                    System.out.println(book.getID()+":"+book.getName());
                }
                break;
            default:
                System.out.println("输入无效! ");
                break;
            }
        }
    }

}
```

该类主要规定了客户端对接口的使用规则。

三、 实验结果展示





四、 实验总结与反思

通过本次实验，初步理解了 RPC 的原理和使用方法。本次实验中的图书管理系统仍然有很大的改进空间，例如添加书籍可以使用二叉堆，这样的话查找书籍可以使用二分查找，这样执行效率会大大提高，不过鉴于本次实验的数据规模很小，因此没有实现这样的做法。除此之外，还可以与 SQL 数据库进行连接，使用数据库可以对数据进行更加安全和规范的使用以及储存，

这也是可以在已有基础上改进的地方。