南昌大学

NANCHANG UNIVERSITY

Web 开发设计大作业报告

题　　目： 书店信息管理系统

学　　院： 软件学院

专　　业： 信息安全

班　　级： 193 班

完成人数： 1 人

人　　员： 8003119100　丁俊

起讫日期： 2021.5.23-2021.6.18

任课教师： 罗　铭　　**职称：** 教 授

完成时间： 2021.6.18

填表日期： 2021 年 6 月 18 日

目录

一、选题和需求介绍

1.1 选题

该项目是从图书和购买图书信息层面来设计的，用于处理书籍信息和购买图书信息并且可以按要求显示图书的信息。可以方便管理员管理书籍和用户购买指定的图书。

1.2 需求介绍

1.2.1 系统功能需求

该系统是书店信息管理系统，主要模块分为两个方面，分别是管理员和用户顾客两个层面设计。管理员模块其中管理员可以管理图书进行相关操作，如添加、增加和修改，核心内容还包括分页显示。用户模块需要登录和注册账号并进行登录操作后可以购买图书、查看我的书架、多组合条件查询图书、分页显示、修改密码。

1.2.2 系统环境

开发语言：java 语言，面向对象编程

操作系统：Microsoft Windows 10

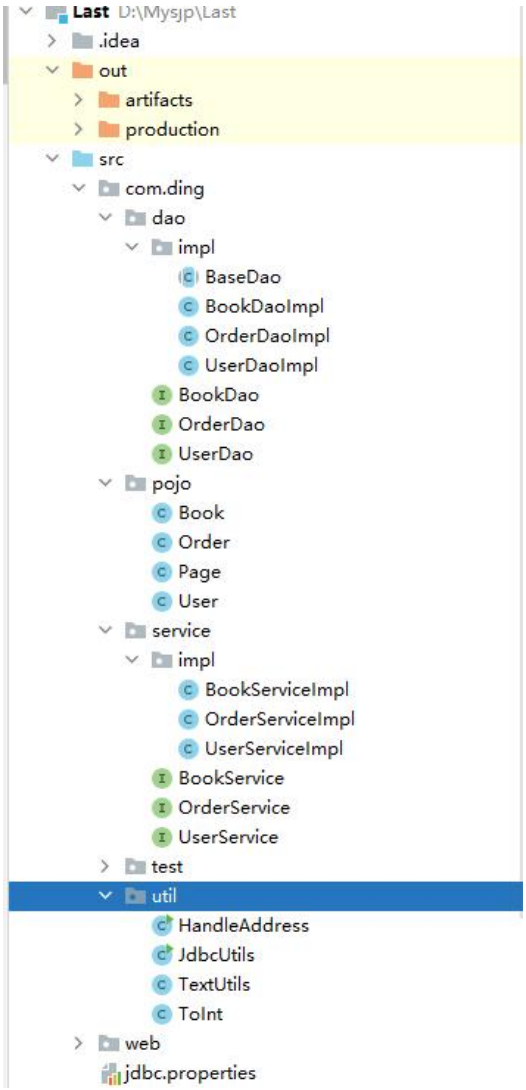
JAVA 开发包：JDK1.8

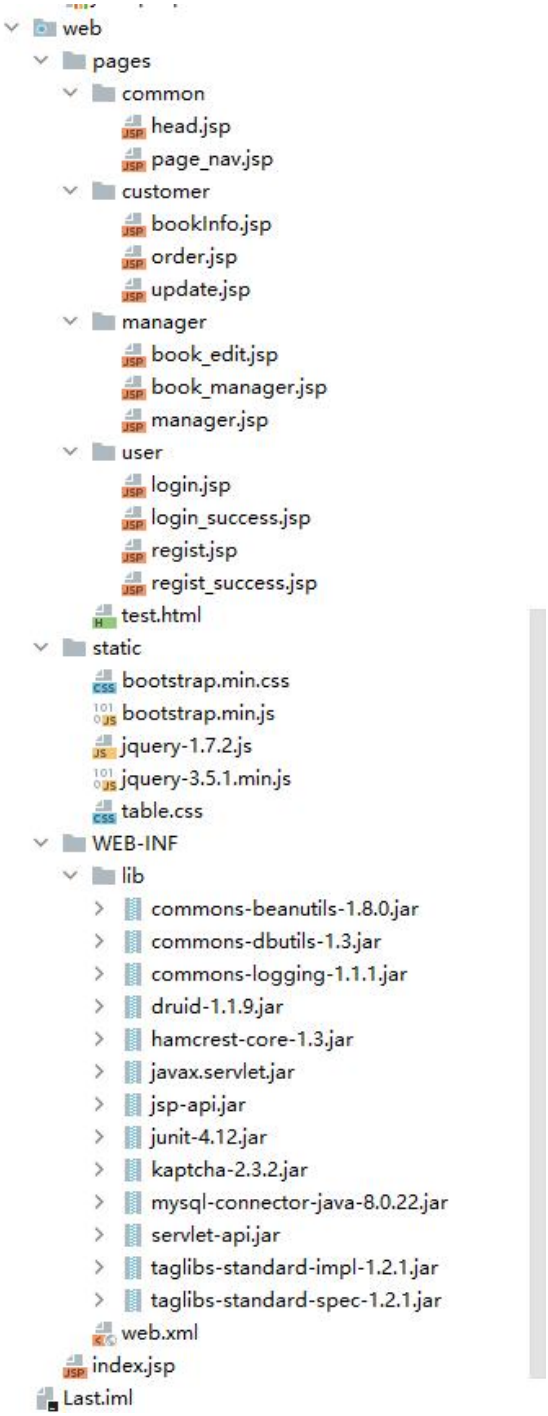
开发平台：IDEA 2020 版本

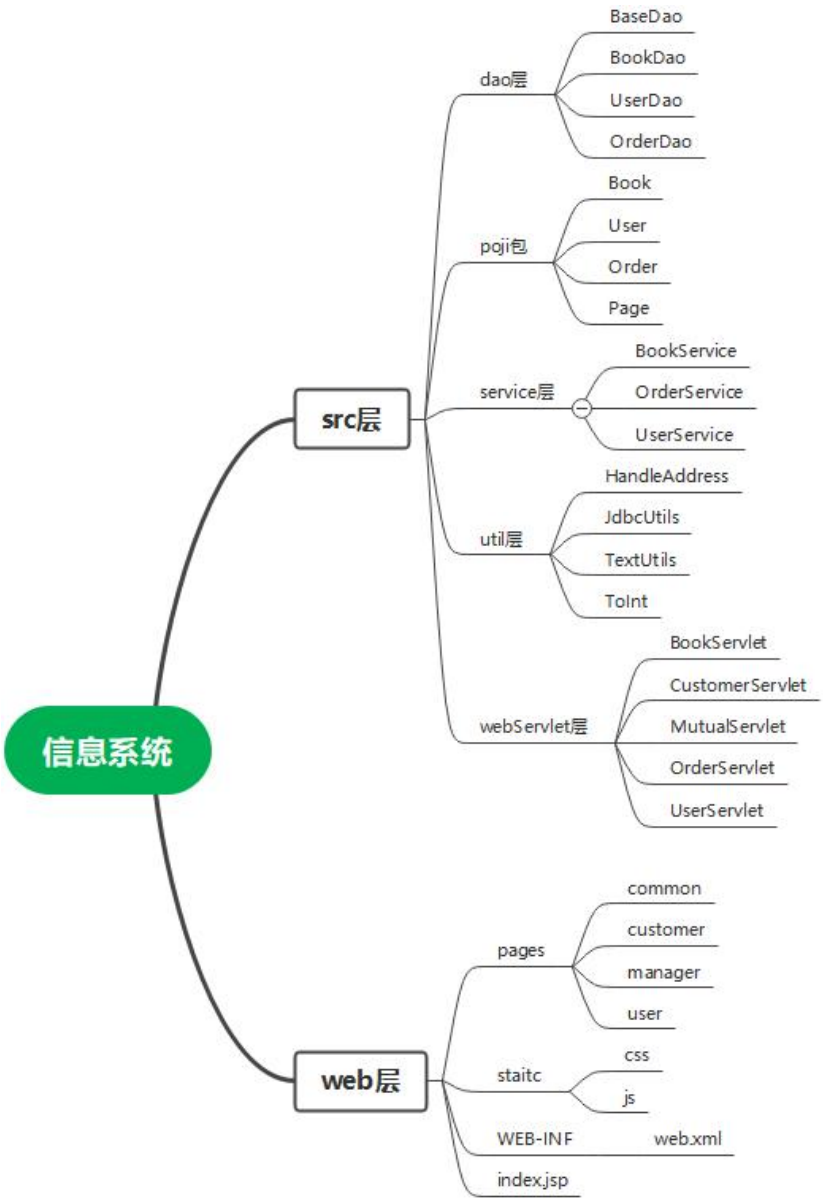
所用 jar 包: commons-beanutils-1.8.0.jar、commons-dbutils-1.3.jar、druid-1.1.9.jar、hamcrest-core-1.3.jar、javax.servlet.jar、jsp-api.jar、mysql-connector-java-8.0.22.jar、servlet-api.jar、taglibs-standard-impl-1.2.1.jar 等

二、结构设计和技术要求

2.1 程序文件结构



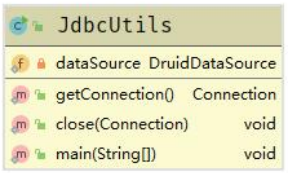


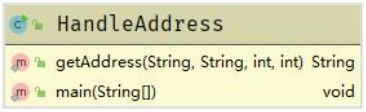


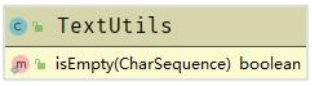
Index.jsp 是主界面

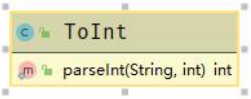
2. 2 整体模块和功能实现

2.2.1Util 模块









先在 mysq 中创建数据库和 t_user、t_book、t_order 表

```
create database book;
use book;
create table t_user(
`id` int primary key auto_increment,
`sname` varchar(42) not null,
`username` varchar(20) not null unique,
`password` varchar(32) not null,
`email` varchar(200)
);
create table t_book(
`id` int primary key auto_increment,
`name` varchar(100),
`author` varchar(100),
`price` decimal(11,2),
`stock` int,
`type` varchar(100)
);
```

```
create table t_order(
`id` int primary key auto_increment,
`username` varchar(100),
`bookId` varchar(100),
`bookName` varchar(100),
`author` varchar(100),
`bookType` varchar(100),
`bookNum` int
);
```

Util 模块提供各种在此项目中频繁要使用到的工具类，分别是 JdbcUtils 数据库连接池，使用配置文件连接数据库连接池，使用到了 dbutil 核心 jar 包。

```
try {
    Properties properties = new Properties();
    // 读取jdbc配置文件
    InputStream inputStream = JdbcUtils.class.getClassLoader().getResourceAsStream( name: "jdbc.properties");
    // 从流中加载数据
    properties.load(inputStream);
    dataSource  = (DruidDataSource) DruidDataSourceFactory.createDataSource(properties);
public static Connection getConnection(){
    Connection conn = null;
    // 获取数据库池中的连接,如返回null连接失败
    try {
        conn = dataSource.getConnection();
    } catch (Exception e) {
        e.printStackTrace();
    }
    return conn;
}
```

HandleArress 类是专门用来处理记录浏览器跳转不同页面所获取的参数。

```
StringBuilder address = new StringBuilder("customer/bookServlet?action=pageByThree");
if(!TextUtils.isEmpty(name)){
    address.append("&name=").append(name);
}
if(!TextUtils.isEmpty(type)){
    address.append("&type=").append(type);
}
if(min != 0){
    address.append("&min=").append(min);
}
if(max != Integer.MAX_VALUE){
    address.append("&max=").append(max);
}
if(TextUtils.isEmpty(name) && TextUtils.isEmpty(type) && min == 0 && max == Integer.MAX_VALUE ){
    address.delete(32,39);
}
// 返回代表地址的字符串
return address.toString();
```

TextUtils 类判断一个字符串是否为空，封装成一个方法。

```
public static boolean isEmpty(CharSequence s){
    return s==null || s.length() == 0;
}
```

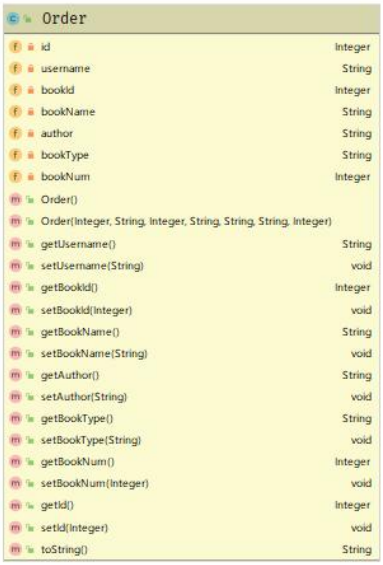
ToInt 类将一个字符串转换成整型，否则转换成自定义的默认值。

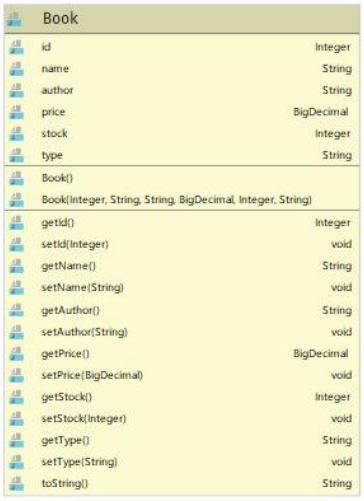
```
public static int parseInt(String strInt,int defaultValue)·
    try{
        return Integer.parseInt(strInt);
    }catch (Exception e){

    }
    return defaultValue;
}
```

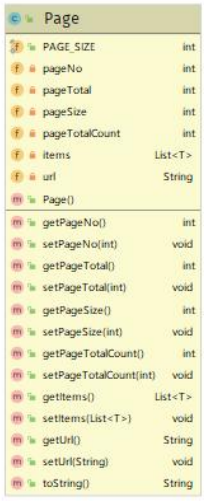
2.2.2poji 模块

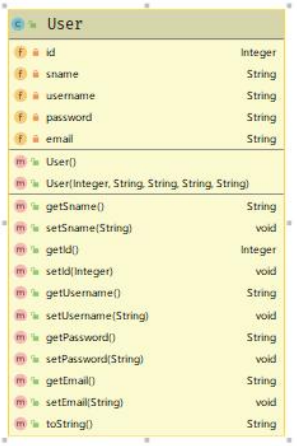
poji 模块中定义了 3 个对象实物类 User、Book、Order 和 1 个分页类 Page。

实体类对象










```
public class Book {

    private Integer id; // 图书编号
    private String name; // 图书名称
    private String author; // 图书作者
    private BigDecimal price; // 图书价格
    private Integer stock; // 图书库存
    private String type; // 类型
```

```
public class User {
    private Integer id;
    private String sname; // 昵称
    private String username; // 账号
    private String password; // 密码
    private String email;
    public String getSname() {
        return sname;
    }
```

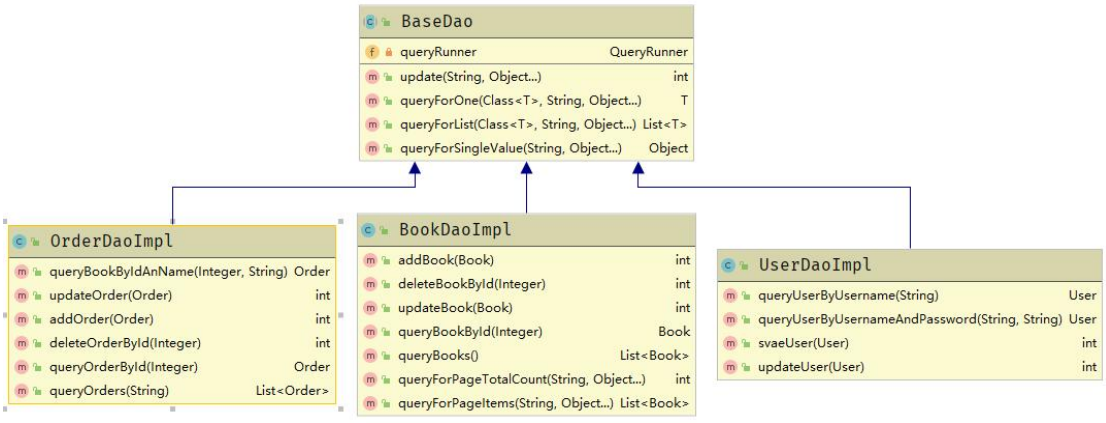
```
public class Order {
    private Integer id; // 记录编号
    private String username; // 用户账号
    private Integer bookId; // 书的id
    private String bookName; // 书名
    private String author; // 书的作者
    private String bookType; // 书的类型
    private Integer bookNum; // 购买的书的数量
```

3 个对象实体类都定义了相关的 get 和 set 方法、toString 方法和初始化构造方法。

分类对象类

```
public class Page<T> {

    public static final int PAGE_SIZE = 4;
    // 当前页码
    private int pageNo;
    // 总页码
    private int pageTotal;
    // 当前页显示数量
    private int pageSize = PAGE_SIZE;
    // 总记录数
    private int pageTotalCount;
    // 当前页面数据
    private List<T> items;
    // 分页条的请求地址(sql语句和不同角色决定)
    private String url;
    public Page() {
```

2.2.3Dao 模块



BaseDao 抽象类：提供数据库查询操作接口，创建使用 QueryRunner 对象方法访问数库，还有 ResultSetHandler 结果处理器接收查询结果。BeanHandler 表示把一行的数据封装到一个实体对象中，BeanListHandler　把多行的数据封装到一个 List 集合中，ScalarHandler 结果返回单个值。

执行一条语句

```
public int update(String sql,Object... args){
    Connection connection = JdbcUtils.getConnection();

    try {
        return queryRunner.update(connection,sql,args);
```

查询返回一条 javabean

```
public <T> T queryForOne(Class<T> type,String sql,Object... args){
    Connection con = JdbcUtils.getConnection();
    try {
        return queryRunner.query(con,sql,new BeanHandler<T>(type),args);
```

查询返回多个 javabean

```
public <T> List<T> queryForList(Class<T> type, String sql, Object... args) {
    Connection con = JdbcUtils.getConnection();
    try {
        return queryRunner.query(con, sql, new BeanListHandler<T>(type), args);
```

查询返回单个值、一行一列

```
public Object queryForSingleValue(String sql,Object... args){
    Connection conn = JdbcUtils.getConnection();
    try {
        return queryRunner.query(conn,sql,new ScalarHandler(),args);
```

BookDao 接口和 BookDaoImpl 实现类：提供对图书操作和管理的接口和实现方法，BookDaoImpl 实现 BookDao 接口并且继承 BaseDao 抽象类。

```
public int addBook(Book book);
public int deleteBookById(Integer id);

public int updateBook(Book book);

public Book queryBookById(Integer id);

public List<Book> queryBooks();

public int queryForPageTotalCount(String sql,Object... args);
public List<Book> queryForPageItems(String sql,Object... args);
```

以上方法都直接使用封装好的 BaseDao 抽象类中的查询方法，把底层对数据库连接池的操作封装成函数直接进行调用进而形成新的封装方法。

UserDao 接口和 UserDaoImpl 实现类：提供对用户信息的管理方法，UserDaoImp 实现 UserDao 接口和继承 BaseDao 抽象类。

```
public User queryUserByUsername(String username);

public User queryUserByUsernameAndPassword(String username,String password);

public int svaeUser(User user);

// 更新用户信息
public int updateUser(User user);
```

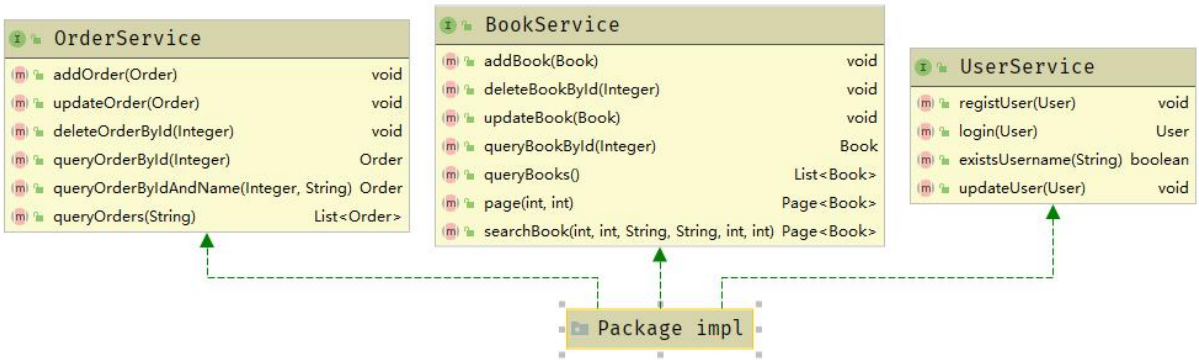
分别是根据账号查询用户、根据账号密码两条件查询、保存插入用户信息到数据库中、更新用户信息，上述方法直接调用 BaseDao 抽象类中的方法，也把函数进行封装提供后层更快捷的接口和函数方法。

OrderDao 接口和 OrderDaoImp 实现类：提供对订单管理的方法，

```
public Order queryBookByIdAnName(Integer bookId, String username);

public int updateOrder(Order order);

public int addOrder(Order order);

public int deleteOrderById(Integer id);

public Order queryOrderById(Integer id);

public List<Order> queryOrders(String username);
```

分别是根据书的 id 和用户账号两个条件查询订单记录、更新修改订单、添加订单、根据订单 id 删除订单、根据 id 查询订单、根据账号查询订单记录

2.2.4Service 模块



Service 模块中定义的方法在 Servlet 方法充当直接被调用的对象方法类，在函数方法体中初始化了一个 BookDao、OrderDao、UserDao 对象，对 Dao 层的对象函数进行再一次封装。

BookService

接口定义：

```
public interface BookService {
    public void addBook(Book book);

    public void deleteBookById(Integer id);

    public void updateBook(Book book);

    public Book queryBookById(Integer id);

    public List<Book> queryBooks();

    /**
     * 查询某一页的数据
     * @param pageNo 第几页
     * @param pageSize 每页的记录数
     * @return 该页数据
     */
    public Page<Book> page(int pageNo, int pageSize);

    public Page<Book> searchBook(int pageNo,int pageSize,String name,String type,int min,int max);
```

添加图书

```
public BookDao bookDao = new BookDaoImpl(); // 数据库操作
@Override
public void addBook(Book book) {
    bookDao.addBook(book);
}
```

修改图书

```
public void updateBook(Book book) {
    bookDao.updateBook(book);
}
```

删除图书

```
public void deleteBookById(Integer id) {
    bookDao.deleteBookById(id);
}
```

根据图书 id 查找单本书

```
public Book queryBookById(Integer id) {
    return bookDao.queryBookById(id);
}
```

图书分页方法(查询所有图书)

```
public Page<Book> page(int pageNo, int pageSize) {
    Page<Book> page = new Page<~>();


    // 每页的记录数
    page.setPageSize(pageSize);
    // 求总记录数
    int pagetotalCount = bookDao.queryForPageTotalCount( sql: "select count(*) from t_book");
    // 设置总记录数
    page.setPageTotalCount(pagetotalCount);
    // 总页码
    int pageTotal = (pagetotalCount % pageSize > 0) ?  pagetotalCount/pageSize+1 : pagetotalCount/pageSize;
    page.setPageTotal(pageTotal);
    if(pageNo < 1){
        pageNo=1;
    }
    if(pageNo > pageTotal){
        pageNo = pageTotal;
    }
    // 设置当前页码
    page.setPageNo(pageNo);
    int begin = (page.getPageNo()-1)*pageSize; // 查询起点
    if(begin < 0){
        begin = 0;
    }
    String sql = "select * from t_book" + " limit " + begin + "," + pageSize;
    // 当前页的数据
    List<Book> items = bookDao.queryForPageItems(sql);
    // 设置当前页数据
    page.setItems(items);
    return page;
```

函数参数是 pageNo 当前页和 pageSize 页面大小，“select count(*) from t_book”查询图书的数目，通过总记录数/每页大小=总页数(除不尽要加 1)，查询起点 begin = (当前页-1)*每页大小，再通过“limit begin,pageSize”有限查询某页的多行记录返回一个列表设置为 page 对象中 list 集合，返回处理好的 page 对象。

条件查询并分页

```
String cntSql = "select count(*) from t_book where 1=1";
List<String> list = new ArrayList<~>();
// 生成sql语句
if(!TextUtils.isEmpty(name)){
    sql = sql + " and name like ? ";
    cntSql = cntSql + " and name like ? ";
    list.add("%" + name + "%");
}

if(!TextUtils.isEmpty(type)){
    sql = sql + " and type = ? ";
    cntSql = cntSql + " and type = ? ";
    list.add(type);
}
sql = sql + " and price between " + min + " and " + max+ " order by price ";
// 这里order by price 按照价格排序
cntSql = cntSql + " and price between " + min + " and " + max;

// 总记录数
int pagetotalCount = bookDao.queryForPageTotalCount(cntSql,list.toArray());
page.setPageTotalCount(pagetotalCount);

// 总页码
int pageTotal = (pagetotalCount % pageSize > 0) ? pagetotalCount/pageSize+1 : pagetotalCount/pageSize;
page.setPageTotal(pageTotal);
// 总页码
int pageTotal = (pagetotalCount % pageSize > 0) ? pagetotalCount/pageSize+1 : pagetotalCount/pageSize;
page.setPageTotal(pageTotal);

if(pageNo < 1){
    pageNo = 1;
}
if(pageNo > pageTotal){
    pageNo = pageTotal;
}

// 设置当前页码
page.setPageNo(pageNo);
int begin = (page.getPageNo()-1)*pageSize;
if(begin < 0){
    begin = 0;
}
sql = sql + " limit " + begin + "," + pageSize; // 先设置limit值
List<Book> items = bookDao.queryForPageItems(sql,list.toArray());
page.setItems(items);
return page;
```

对传进来的查询参数进行判空，list 集合是用来记录 sql 语句的查询参数的，作为查询参数传到对应的数据库操作函数中，cntsql 是专门用来查询该条件对应所得记录的总数目，sql 是多条件组合查询的语句参数最后加上所求得的 pageNo 和 pageSize 作为 limit 子句中的参数，注意这里的多组合查询语句可能查不到几条数据从而导致 begin<0，所以要进行判断并保证 begin>0。最终把链接的 sql 语句传给 queryForPageItems 函数返回一个集合作为 page 对象的 list 数据并返回 page 对象。

UserService

接口定义：

```
public void registUser(User user);

public User login(User user);

public boolean existsUsername(String username);

public void updateUser(User user);
```

注册：也即保存用户对象信息到数据库中。

```
public void registUser(User user) {
    userDao.svaeUser(user);
}
```

登录：判断输入的账号和密码在数据库中是否能匹配，如果存在就返回这个 User 对象，否则返回 null。

```
public User login(User user) {
    return userDao.queryUserByUsernameAndPassword(user.getUsername(),user.getPassword());
}
```

是否存在该用户名：

```
public boolean existsUsername(String username) {
    if(userDao.queryUserByUsername(username) == null){
        return false;
    }
    return true;
}
```

更新用户信息：

```
public void updateUser(User user) {
    userDao.updateUser(user);
}
```

OrderService

接口定义：

```
public interface OrderService {

    public void addOrder(Order order);

    public void updateOrder(Order order);

    public void deleteOrderById(Integer id);

    public Order queryOrderById(Integer id);

    public Order queryOrderByIdAndName(Integer bookId,String username);

    public List<Order> queryOrders(String username);
```

添加订单：

```
public void addOrder(Order order) {
    orderDao.addOrder(order);
}
```

更新订单：

```
public void updateOrder(Order order) {
    orderDao.updateOrder(order);
}
```

删除订单：

```
public void deleteOrderById(Integer id) {
    orderDao.deleteOrderById(id);
}
```

根据 id 查询订单：

```
public Order queryOrderById(Integer id) {
    return orderDao.queryOrderById(id);
}
```

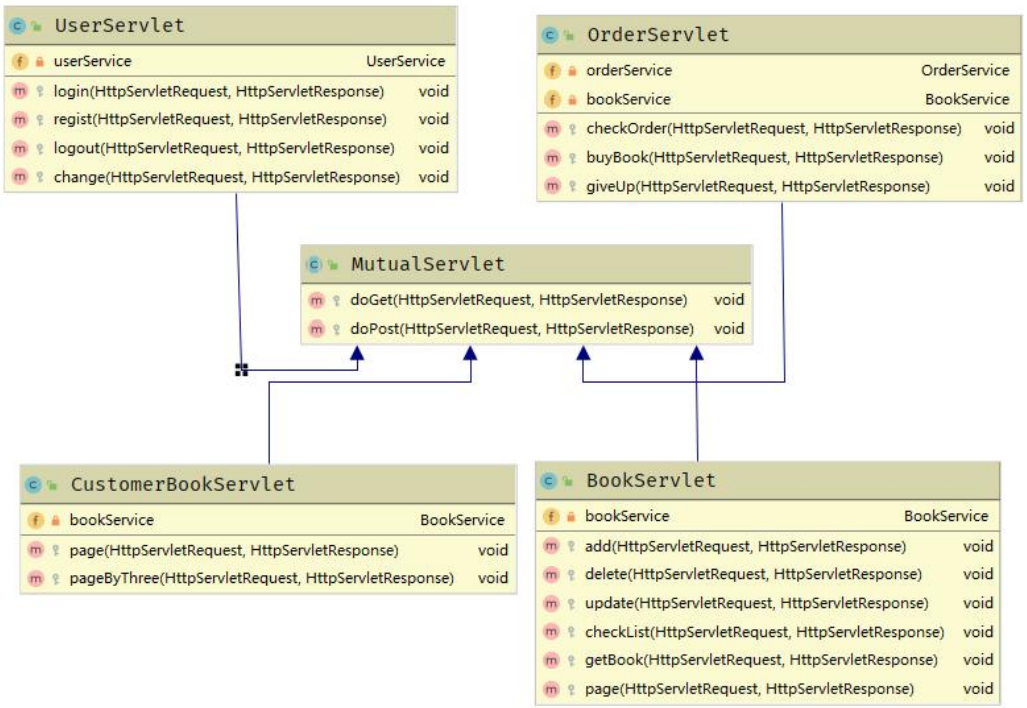
根据图书 id 和登录账号查询：

```
public Order queryOrderByIdAndName(Integer bookId, String username) {
    return orderDao.queryBookByIdAnName(bookId,username);
}
```

查询某个用户有关的图书：

```
public List<Order> queryOrders(String username)
    return orderDao.queryOrders(username);
}
```

2.2.5 web 模块



Web.xml 文件配置

```
<servlet>
    <servlet-name>UserServlet</servlet-name>
    <servlet-class>web.UserServlet</servlet-class>
</servlet>
 <servlet-mapping>
    <servlet-name>UserServlet</servlet-name>
    <url-pattern>/userServlet</url-pattern>
 </servlet-mapping>

 <servlet>
    <servlet-name>BookServlet</servlet-name>
    <servlet-class>web.BookServlet</servlet-class>
 </servlet>
 <servlet-mapping>
    <servlet-name>BookServlet</servlet-name>
    <url-pattern>/manager/bookServlet</url-pattern>
 </servlet-mapping>

 <servlet>
    <servlet-name>CustomerBookServlet</servlet-name>
    <servlet-class>web.CustomerBookServlet</servlet-class>
 </servlet>
 <servlet-mapping>
    <servlet-name>CustomerBookServlet</servlet-name>
    <url-pattern>/customer/bookServlet</url-pattern>
 </servlet-mapping>

 <servlet>
    <servlet-name>OrderServlet</servlet-name>
    <servlet-class>web.OrderServlet</servlet-class>
 </servlet>
 <servlet-mapping>
    <servlet-name>OrderServlet</servlet-name>
    <url-pattern>/orderServlet</url-pattern>
 </servlet-mapping>
```

Web 模块中其实有四个主要的 servlet 类，另外的 MutualServlet 类是公有的被继承的 servlelt 类，处理客户端向服务器发出的 Post 和 Get 请求，其中使用到了带有指定参数的指定对象调用由此 Method 对象表示的底层方法。先获取 action 参数的值，返回一个 Metho 对象，反映此 Class 对象所表示的类或接口的指定已声明方法。例如 action = page 表示调用方法名为“action”的函数方法。

```
protected void doPost(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {
    String act = req.getParameter( s: "action");
    try {
        // 获得action字符串获取相应的业务 方法反射对象
        Method method = this.getClass().getDeclaredMethod(act,HttpServletRequest.class,HttpServletResponse.class);
        // 调用目标业务方法
        method.invoke( obj: this,req,resp);
    }catch (Exception e){
          e.printStackTrace();
    }finally {
    }
}
```

UserServlet

处理与用户相关的业务比如登陆注册、注销、修改密码。

登录函数

```
protected void login(HttpServletRequest req, HttpServletResponse resp) throws ServletEx
    // 获取参数
    String username = req.getParameter( s: "username");
    String password = req.getParameter( s: "password");

    // 调用 userService.login()登录处理业务
    User loginUser = userService.login(new User( id: null, sname: null,username,password, e
    // 如果等于null,说明登录失败
    if(loginUser == null){
        // 登陆失败跳回登录页面
        req.setAttribute( s: "msg", o: "用户名或密码错误!");
        req.setAttribute( s: "username",username);
        req.getRequestDispatcher( s: "/pages/user/login.jsp").forward(req,resp);
    }else{
        // 登录成功页面,跳转到分页显示界面 action = page执行page方法
        req.getSession().setAttribute( s: "user",loginUser);
        req.getRequestDispatcher( s: "/pages/user/login_success.jsp").forward(req,resp);
```

获取输入框的内容并在数据库中查找该用户，如果查不到说明不存在该用户，提示用户名或密码错误；否则利用 session 对象设置全局变量的对象值，然后跳转到登录成功界面，并显示用户数据。

注册函数

```
protected void regist(HttpServletRequest req, HttpServletResponse resp) throws ServletEx
    req.setCharacterEncoding("utf-8");
    String sname = req.getParameter( s: "sname");
    String username = req.getParameter( s: "username");
    String password = req.getParameter( s: "password");
    String email = req.getParameter( s: "email");
    if(userService.existsUsername(username)){
        // 检查用户名是否可用
        // 跳回注册页面
        System.out.println("用户名"+username+"已存在");
        req.setAttribute( s: "msg", o: "用户名已存在");
        req.setAttribute( s: "username",username);
        req.setAttribute( s: "email",email);
        req.getRequestDispatcher( s: "/pages/user/regist.jsp").forward(req,resp);
    }else{
        // 可用调用数据库
        userService.registUser(new User( id: null,sname,username,password,email));
        // 跳转到注册成功页面
        req.getRequestDispatcher( s: "/pages/user/regist_success.jsp").forward(req,resp);
```

判断输入框的账号是否在数据库中存在，如果存在给出提示信息，反之跳转到注册成功界面。

注销函数

```
protected void logout(HttpServletRequest req, HttpServ
        1、销毁 Session 中用户登录的信息（或者销毁 Session）
    req.getSession().invalidate();
2、重定向到首页（或登录页面）。
    resp.sendRedirect(req.getContextPath());
```

销毁 Session 中用户登录的信息，重定向跳转到主工程路径下的 index.jsp 文件。

修改密码

```
String password = user.getPassword();
// 得到原有的用户账号
String username = user.getUsername();
String newpassword = req.getParameter( s: "password");
String email = req.getParameter( s: "email");
User newUser = new User( id: null,sname,username,password,email);
req.getSession().setAttribute( s: "user",newUser);
// 更新用户信息
userService.updateUser(new User( id: null,sname,username,newpassword,email));

if(newpassword.equals(password)){
    req.getRequestDispatcher( s: "/pages/user/login_success.jsp").forward(rec
}else {
    req.setAttribute( s: "msg", o: "成功修改密码，请重新登陆");
    req.getRequestDispatcher( s: "/pages/user/login.jsp").forward(req,resp);
```

先更新用户信息，获取 Session 对象中原有的用户密码和现在输入的密码，如果一致说明没有修改密码，然后跳转到登录成功页面；如果密码修改过则跳转重新登陆界面。

BookServlet

处理与书籍相关的业务，比如添加、修改、删除、分页查询。

添加图书

```
protected void add(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {
    // 获取参数
    req.setCharacterEncoding("utf-8");
    String name = req.getParameter( s: "name");
    String author = req.getParameter( s: "author");
    String type = req.getParameter( s: "type");
    BigDecimal price = new BigDecimal(req.getParameter( s: "price"));
    Integer stock = Integer.valueOf(req.getParameter( s: "stock"));
    int pag = ToInt.parseInt(req.getParameter( s: "pageNo"), defaultValue: 0);
    pag += 1;
    bookService.addBook(new Book( id: null,name,author,price,stock,type));
    // 返回跳转到图书显示界面更新
    resp.sendRedirect( s: req.getContextPath() + "/manager/bookServlet?action=page&pageNo="+pag); //checkLis
```

删除图书

```
protected void delete(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {
    req.setCharacterEncoding("utf-8");
    Integer id = Integer.parseInt(req.getParameter( s: "id"));
      String pageNum = req.getParameter("pageNo");
    bookService.deleteBookById(id);
    resp.sendRedirect( s: req.getContextPath() + "/manager/bookServlet?action=page&pageNo="+req.getParameter( s: "pageNo"));
    // 第二次采用分页所有要跳到分页处理,删除一条记录后在当前页面显示
```

更新图书

```
protected void update(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {
    req.setCharacterEncoding("utf-8");
    // 先获取当前页码
      String pageNum = req.getParameter("pageNo");
    String name = req.getParameter( s: "name");
    String author = req.getParameter( s: "author");
    String type = req.getParameter( s: "type");
    BigDecimal price = new BigDecimal(req.getParameter( s: "price"));
    Integer stock = Integer.valueOf(req.getParameter( s: "stock"));
    Integer id = Integer.valueOf(req.getParameter( s: "id"));
    Book book = new Book(id,name,author,price,stock,type);
    bookService.updateBook(book);
    resp.sendRedirect( s: req.getContextPath() + "/manager/bookServlet?action=page&pageNo="+req.getParameter( s: "pageNo"));
```

获得分页对象

```
protected void page(HttpServletRequest req, HttpServletResponse resp) throws ServletExc
    req.setCharacterEncoding("utf-8");
    // 获取请求参数pageNo 和 pageSize
    int PageNo = ToInt.parseInt(req.getParameter( s: "pageNo"), defaultValue: 1);
    int PageSize = Page.PAGE_SIZE;
    // 调用BookService.page对象,可求出总页数和总记录数
    Page<Book> page = bookService.page(PageNo,PageSize);
    // 后台访问地址
    page.setUrl("manager/bookServlet?action=page");
    // 保存到request域中
    req.setAttribute( s: "page",page);
    // 请求转发到book_manager.jsp页面
    req.getRequestDispatcher( s: "/pages/manager/book_manager.jsp").forward(req,resp);
```

CustomerBookServlet

客户的分页显示和多条件显示功能。

多条件查询分页业务函数

```
protected void pageByThree(HttpServletRequest req, HttpServletResponse resp) throws
    req.setCharacterEncoding("utf-8");
    int pageNo = ToInt.parseInt(req.getParameter( s: "pageNo"), defaultValue: 1);
    int pageSize = Page.PAGE_SIZE;
    int min = ToInt.parseInt(req.getParameter( s: "min"), defaultValue: 0);
    int max = ToInt.parseInt(req.getParameter( s: "max"),Integer.MAX_VALUE);
    String name = req.getParameter( s: "name");
    String type = req.getParameter( s: "type");
    Page<Book> page = bookService.searchBook(pageNo,pageSize,name,type,min,max);
    // 先获取url地址
    String url = HandleAddress.getAddress(name,type,min,max);
    // 没有查询条件都为空时
    if(url.equals("customer/bookServlet?action=page")){
        // 这是无查询条件的默认查询所以书籍
        page = bookService.page( pageNo: 1,pageSize);
    }
    page.setUrl(url);
    req.setAttribute( s: "page",page);
    req.getRequestDispatcher( s: "/pages/customer/bookInfo.jsp").forward(req,resp);
}
```

获取客户端传过来的分页参数值，有书籍名模糊查询、书籍价格范围、书籍类型等。

OrderServlet

处理订单相关的业务，比如查询当前用户所属的订单、买书(加入书架)、删除书架中的书。

查询订单(书架)

```
protected void checkOrder(HttpServletRequest req, HttpServletResponse resp
    // 获取当前登录的用户
    User user = (User)req.getSession().getAttribute( s: "user");
    // 查询该用户的购买订单
    List<Order> orderList = orderService.queryOrders(user.getUsername());
    req.setAttribute( s: "orders",orderList);
```

根据当前登录的 Session 中的 User 对象中的账号查询该用户所对应的订单，即该用户所购买的所以书籍，然后在 request 域中设置 orders 对象的值。

买书(加入书架)

```
protected void buyBook(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {
    // 得到当前登录的用户对象
    User user = (User) req.getSession().getAttribute( s: "user");
    // 获得购买的图书的id
    Integer bookId = Integer.valueOf(req.getParameter( s: "bookId"));
    // 获得该书籍对象
    Book book = bookService.queryBookById(bookId);
    // 没有买过该书，则添加订单记录
    Order order = orderService.queryOrderByIdAndName(bookId,user.getUsername());
    if(order == null){
        orderService.addOrder(new Order( id: null,user.getUsername(),book.getId(),book.getName(),book.getAuthor(),book.getType(),
    }else {
        // 买过该书，则订单记录中书的数量加1
        orderService.updateOrder(new Order( id: null,order.getUsername(),order.getBookId(),order.getBookName(),order.getAuthor(),
                order.getBookType(), bookNum: order.getBookNum()+1));
    }
      resp.sendRedirect(req.getContextPath() + "/" + url + "&pageNo="+pageNo);
    // 从哪个页面来的回到哪个页面去
    resp.sendRedirect(req.getHeader( s: "Referer"));
```

查询该用户所属的订单(书架)中是否存在该书，如果存在则数量更新加 1；不存在添加一条记录且所属的该书数量为 1。

删除书架中的书

```
protected void giveUp(HttpServletRequest req, HttpServletResponse resp) throws Serv
    int orderId = ToInt.parseInt((req.getParameter( s: "id")), defaultValue: 0);
    Order order = orderService.queryOrderById(orderId);
    if(order.getBookNum() > 1){
        orderService.updateOrder(new Order( id: null,order.getUsername(),order.getBoo
                order.getBookType(), bookNum: order.getBookNum()-1));
    }else{
        orderService.deleteOrderById(orderId);
    }
    resp.sendRedirect( s: req.getContextPath() + "/orderServlet?action=checkOrder");
}
```

同理，当不要某本书时，如果该书数量大于 1，更新数量减 1；当等于 1 时，则直接删除有关该书的记录。

2.2.6 功能实现

公共文件 head.jsp

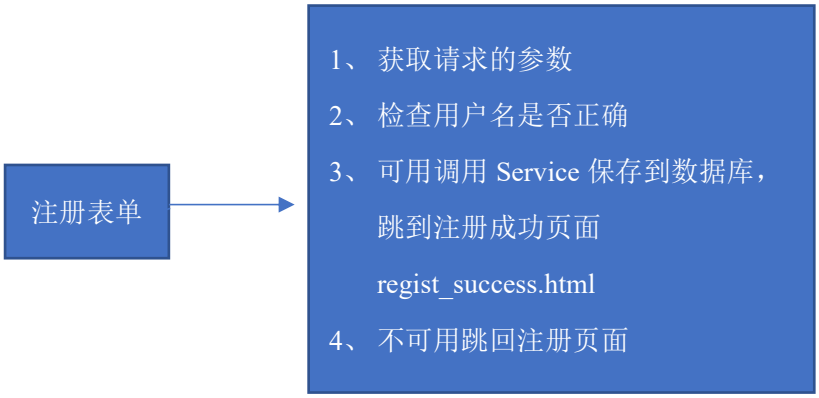
```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<%
    String basePath = request.getScheme()
            + "://"
            + request.getServerName()
            + ":"
            + request.getServerPort()
            + request.getContextPath()
            + "/";

    pageContext.setAttribute("basePath",basePath);
%>
<!--写base标签，永远固定相对路径跳转的结果-->
<base href="<%=basePath%>">
<link rel = "stylesheet" href = "static/bootstrap.min.css">
<link rel="stylesheet" href="static/table.css">
<script type="text/javascript" src="static/jquery-1.7.2.js"></script>
```

在 head.jsp 文件中包含了工程路径，<base>标签将当前文件的相对路径位置定位到 web 目录下，并且把必要的 css 和 js 文件包含，方便需要这些样式的其他 jsp 文件静态包含，利于代码复用功能，避免过多的冗余。

主界面

```
<div class="row">
    <div class="col-md-4"></div>
    <div class="col-md-4">
        <ul class="nav nav-pills nav-stacked">
            <li class="active"><a href="pages/user/login.jsp">登录</a></li>
            <li><a href="pages/user/regist.jsp">注册</a></li>
            <li><a href="pages/manager/manager.jsp">管理员</a></li>
        </ul>
    </div>
```

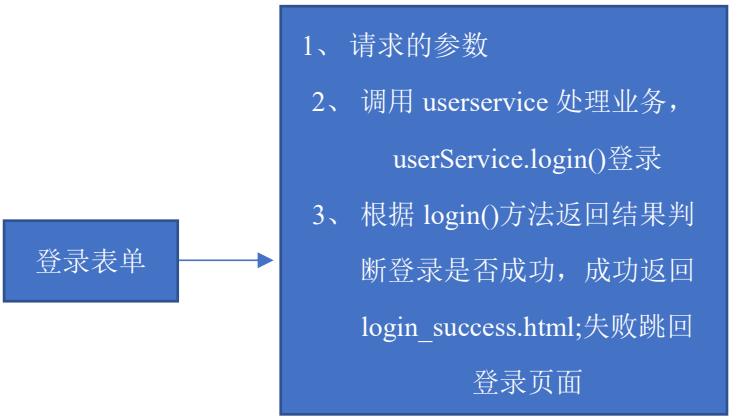
注册功能



```
    <span class="errorMsg">
        ${empty requestScope.msg ? "请输入用户名和密码":requestScope.msg}
    </span>
</div>
    <form style="text-align: center" action="userServlet" method="post">
        <input type = "hidden" name = "action" value = "regist" />
```

表单提交的地址为 userServlet，上部有一个提示框，当注册失败时，可以通过设置 request 中 msg 字符串的值进行错误显示。

登录功能




```
<div class="container" style = "text-align: center">
    <span>
        ${empty requestScope.msg ? "请输入用户名和密码":requestScope.msg}
    </span>
</div>
<form style="..." action="userServlet" method="post">
    <input type = "hidden" name = "action" value = "login" />
    <div class="form-group">
        <label for="user" stype="display:inline;">账号：</label>
        <input type="text" class="form-control" id="user" name="userna
        value="${ requestScope.username}"/>
```

设置 action 参数的值为 regist，顶部的错误提示框是当登录失败时跳转回来的时候读取 request 域中的数据显示的。

分页显示功能

这里有个单独的分页条 page_nav.jsp，可以被静态包含

```
<c:if test="${requestScope.page.pageNo > 1}">
    <a href = "${requestScope.page.url}&pageNo=1">首页</a>
    <a href="${requestScope.page.url}&pageNo=${requestScope.page.pageNo-1}">上一页</a>
</c:if>
<c:if test="${requestScope.page.pageNo <= 1}">
    首页   上一页
</c:if>
第【${requestScope.page.pageNo }】页
<%--    防止末页越界--%>
<c:if test="${requestScope.page.pageNo < requestScope.page.pageTotal}">
    <a href="${requestScope.page.url}&pageNo=${requestScope.page.pageNo+1}">下一页</a>
    <a href="${requestScope.page.url}&pageNo=${requestScope.page.pageTotal}">末页</a>
</c:if>
<c:if test="${requestScope.page.pageNo >= requestScope.page.pageTotal}">
    下一页   末页
</c:if>
<%--            param.pageNo显示当前页码--%>
共[${requestScope.page.pageTotal}]页 [${requestScope.page.pageTotalCount}]条记录 转到第
<input value="${param.pageNo}" name="pn" id="pn_input" style="..."/>页
<input id="Topage" type="button" value="确定">
```

当此时在第一页时，上一页和首页的链接就利用 c:if 判断改成文字而不是链接；同理当此时在最后一页时，下一页和末页的链接也改成纯文字，以防发生越界而显示数据为空的异常。这个分页条是通用的，点击跳页链接可以跳转到分页处理业务然后又跳回来，并且显示了当前页和总页数，这些都已经被封装在 page 对象中，直接读取处理好的 page 对象中的数据即可。

```
$(function () {
    // 跳到指定的页码
    $("#Topage").click(function () {
        var pageNo = $("#pn_input").val();

        var pageTotal = ${requestScope.page.pageTotal}; // 获取总页码
        if(pageNo<1 || pageNo > pageTotal){
            alert("不能跳转,页面超过范围或不存在");
            return;
        }
        window.location.href = "${pageScope.basePath}${requestScope.page.url}&pageNo="+
            pageNo;
```

给可自定义跳转按钮添加响应，判断输入的页面是否超过总页码，如果没有超过则提示不能跳转，反之跳转到指定页面。

添加和修改图书功能

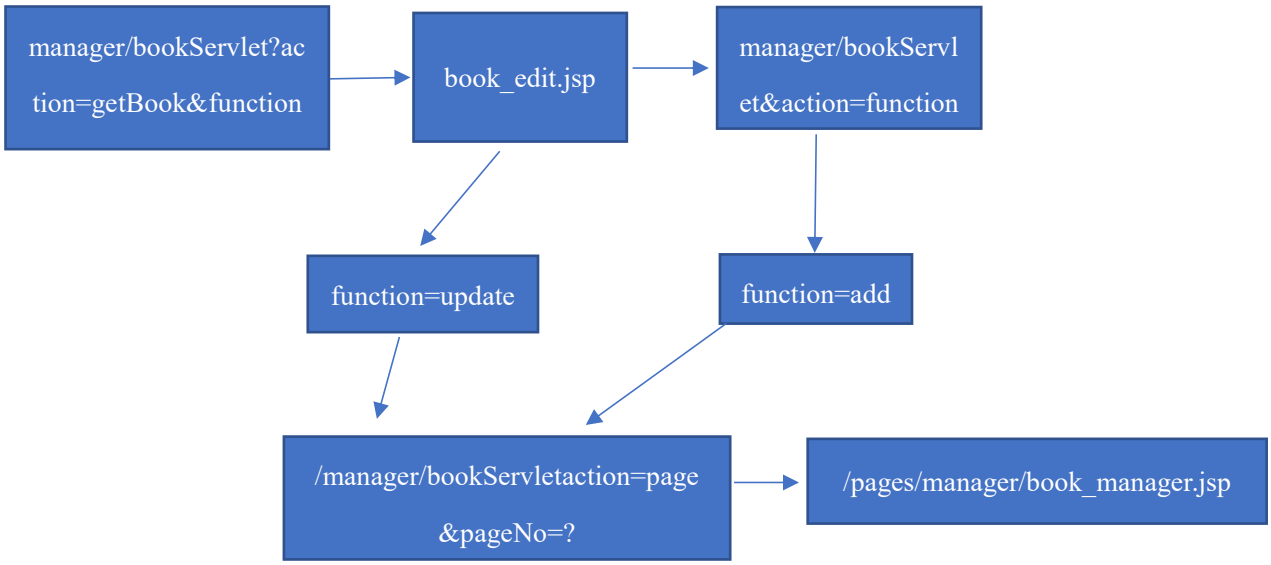
```
<c:forEach items="${requestScope.page.items}" var="book">
    <tr>
        <td>${book.name}</td>
        <td>${book.author}</td>
        <td>${book.price}</td>
        <td>${book.stock}</td>
        <td>${book.type}</td>
        <td><a href="manager/bookServlet?action=getBook&id=${book.id}&function=update&pageNo=${requestScope.page.pageNo}">修改</a></td>
            修改图书把当前页码pageNo传过去--%>
        <td><a class="delete" href="manager/bookServlet?action=delete&id=${book.id}&pageNo=${requestScope.page.pageNo}">删除</a></td></tr>
```

```
<td><a href="pages/manager/book_edit.jsp?function=add&pageNo=${requestScope.page.pageTotal}">添加图书</a></td>
```

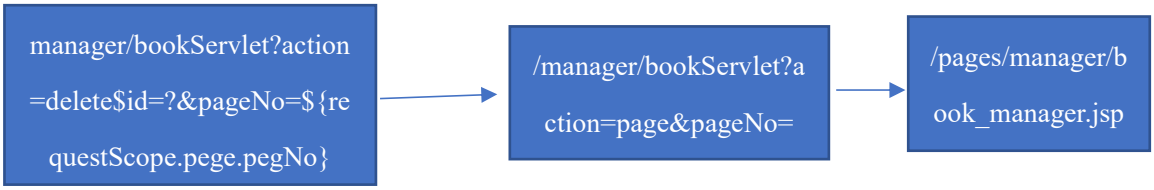
```
protected void getBook(HttpServletRequest req, HttpServletResponse resp) throws Servl
    Integer id = Integer.valueOf(req.getParameter( s: "id")); // 获取传过来的id
    Book book = bookService.queryBookById(id); // 查询该图书
    req.setAttribute( s: "book",book); // 保存图书到request域中
    req.getRequestDispatcher( s: "/pages/manager/book_edit.jsp").forward(req,resp); //
```

```
<form action="manager/bookServlet" method="get">
    <input type = "hidden" name = "pageNo" value="${param.pageNo}"/>
    <input type="hidden" name="action" value="${param.function}" />
    <input type = "hidden" name="id" value="${requestScope.book.id}">
        把id传到bookServlet对指定的book进行修改--%>
    <table class="table table-striped table-condensed" style ="...">
```

value=“${param.function}”获取当前操作是添加还是修改，把值传给 action。利用 c:forEach 语句遍历 page 分页对象，增加图书记录数据，修改和删除三个链接。



删除图书功能

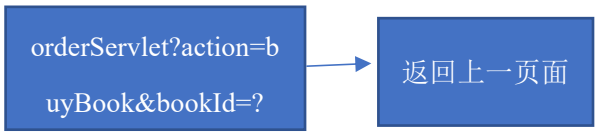


并且添加和更新完成后可以通过获取总页码在总页数的最后一页进行显示，删除可以在当前页面(获取当前页码)进行显示。

用户买书和删除书

```
<c:forEach items="${requestScope.page.items}" var="book">
    <tr>
        <td>${book.name}</td>
        <td>${book.author}</td>
        <td>${book.price}</td>
        <td>${book.stock}</td>
        <td>${book.type}</td>
        <td>
            <a href="orderServlet?action=buyBook&bookId=${book.id}">购买</a>
        </td>
    </tr>
</c:forEach>
```

```
resp.sendRedirect(req.getContextPath() +  /  + url
// 从哪个页面来的回到哪个页面去
resp.sendRedirect(req.getHeader( s: "Referer"));
```



用户点击购买图书之后，跳转到 orderServlet 中的 buyBook 方法，处理完成后直接跳转回原来的页面，相当于一直在当前页面没动。

```
<c:forEach items="${requestScope.orders}" var="order">
    <tr>
        <td>${order.bookName}</td>
        <td>${order.author}</td>
        <td>${order.bookNum}</td>
        <td>${order.bookType}</td>
        <td><a href="orderServlet?action=giveUp&id=${order.id}">不想要了</a></td>
    </tr>
</c:forEach>
```

当用户点击查询订单后，即看我的书架界面，跳转到 orderServlet?action=giveUp，删除书架中的书籍后，跳回我的订单显示并且需要更新，即地址 /orderServlet?action=checkOrder。

多条件查询

```
<div class="container">
    <form action="customer/bookServlet" method="get">
        <input type="hidden" name="action" value="pageByThree">
        书名:<input id = "book_name" type="text" name="name" value="${param.name}">
        价格: <input id="min" type="text" name="min" value="${param.min}">
        <input id = "max" type="text" name="max" value="${param.max}">元
        种类:<input id="type" type="text" name="type" value="${param.type}">
        <input type="submit" value="查询" />
    </form>
```

```
Page<Book> page = bookService.searchBook(pageNo,pageSize,name,type,min,max);
// 先获取url地址
String url = HandleAddress.getAddress(name,type,min,max);
// 没有查询条件都为空时
if(url.equals("customer/bookServlet?action=page")){
    // 这是无查询条件的默认查询所以书籍
    page = bookService.page( pageNo: 1,pageSize);
}
page.setUrl(url);
req.setAttribute( s: "page",page);
req.getRequestDispatcher( s: "/pages/customer/bookInfo.jsp").forward(req,resp);
```

查询表单，因为要把查询条件记录下来，所以要在输入框中获取 request 域中设置查询前输入的数据并连续记录，这样就能实现多条件查询并分页显示，不记录的话查询条件就会无效。然后设置 url 的地址，记录在 page 对象中再次挑战到 bookInfo.jsp 文件。

三、界面设计

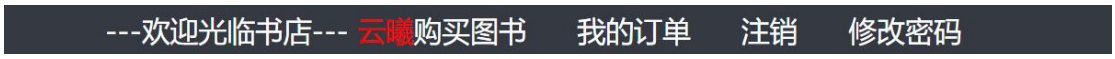
主界面

管理员界面

登陆界面

注册界面

登录成功



购买图书和多条件查询界面

光明书店

书名	作者	价格	库存	类型	操作
java入门	丁俊	100.00	5	编程	购买
大话西游	小锅	40.00	20	历史	购买
c++程序设计	钱能	92.00	20	编程	购买
悲惨世界	雨果	100.00	20	文学	购买

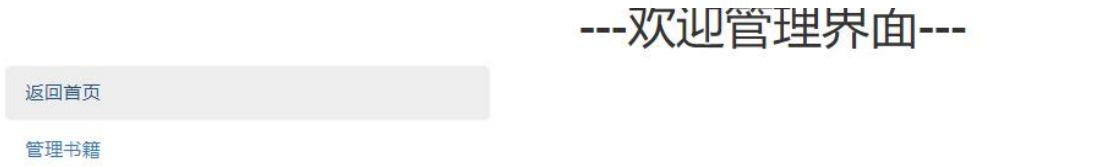
首页 上一页 第【1】页 下一页 末页 共[9]页 [36]条记录 转到第 页 确定

我的订单

我的书架

书名	作者	数量	类型	
悲惨世界	雨果	1	文学	不想要了
大话西游	小锅	2	历史	不想要了
java入门	丁俊	1	编程	不想要了
国史大纲	钱穆	1	历史	不想要了

管理界面



四、系统测试

用户

登录

输入错误的账号或密码会有提示信息

登录成功后上部有多个选项，点击购买图书即开始分页显示内容，输入要查询的组合条件点击查询。

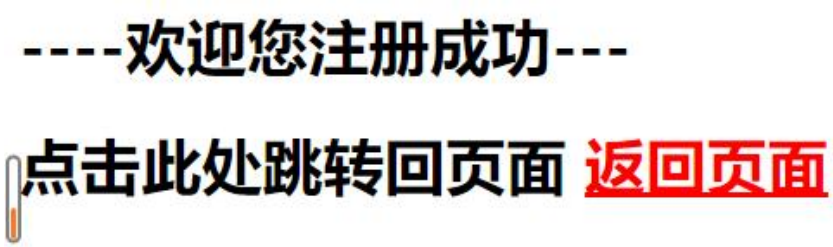
注册

当用户名已存在会提示这个用户名账号以前别的用户注册过。

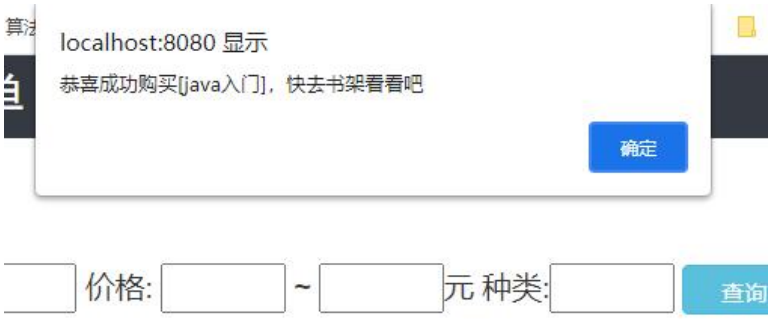
密码不一致

邮箱格式不合法

注册成功



购买图书



我的书架

书名	作者	数量	类型	
野果游乐园	商务印书馆	1	自然	不想要了
大话西游	小锅	1	历史	不想要了
java入门	丁俊	2	编程	不想要了

购买后新增图书或数量

书名	作者	数量	类型	
野果游乐园	商务印书馆	1	自然	不想要了
大话西游	小锅	3	历史	不想要了
java入门	丁俊	3	编程	不想要了
c++程序设计	钱能	2	编程	不想要了
悲惨世界	雨果	1	文学	不想要了

删除书架图书

·欢迎光临书店--- 云曦购买图书　我的订单　注销　修改密码

我的书架

书名	作者	数量	类型	
c++程序设计	钱能	2	编程	不想要了
悲惨世界	雨果	1	文学	不想要了

多组合条件查询

查询包含“c”字符的书籍

请输入查询关键字: 书名:c 作者: 价格: ~ 元 种类: 查询

光明书店

书名	作者	价格	库存	类型	操作
C语言	谭浩强	30.00	10	编程	购买
C Primer Plus	史蒂芬普拉达	80.00	100	编程	购买
c++程序设计	钱能	92.00	20	编程	购买

首页　上一页 第【1】页 下一页　末页 共[1]页 [3]条记录 转到第 页 确定

查询价格区间为 30~120，种类为编程的书籍

请输入查询关键字: 书名: 作者: 价格: 30 ~ 120 元 种类: 编程 查询

光明书店

书名	作者	价格	库存	类型	操作
C语言	谭浩强	30.00	10	编程	购买
数据结构	王涛	60.00	10	编程	购买
C Primer Plus	史蒂芬普拉达	80.00	100	编程	购买
萨乌丁发	丁俊	80.00	100	编程	购买

首页　上一页 第【1】页 下一页 末页 共[3]页 [9]条记录 转到第 1 页 确定

查询编程类书籍

请输入查询关键字: 书名: 作者: 价格: ~ 元 种类: 编程 查询

光明书店

书名	作者	价格	库存	类型	操作
C语言	谭浩强	30.00	10	编程	购买
数据结构	王涛	60.00	10	编程	购买
C Primer Plus	史蒂芬普拉达	80.00	100	编程	购买
萨乌丁发	丁俊	80.00	100	编程	购买

首页　上一页 第【1】页 下一页 末页 共[3]页 [9]条记录 转到第 1 页 确定

管理员

添加图书

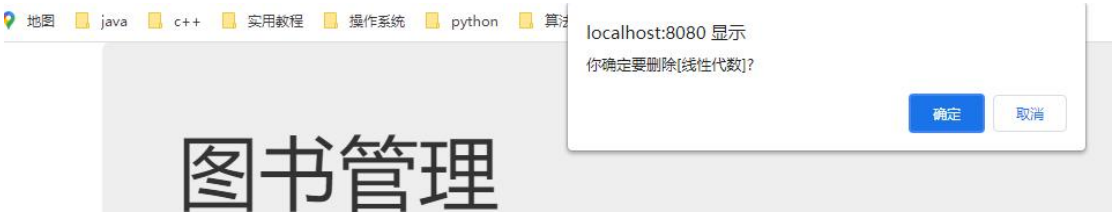
书名	作者	价格	库存	类型	操作	
如何获取富婆芳心	小case	75.00	20	生活	修改	删除
中华上下五千年	炎帝	115.00	30	历史	修改	删除
深入理解计算机系统	大卫	85.00	132	编程	修改	删除

添加图书

首页 上一页 第【6】页 下一页 末页 共[6]页 [23]条记录 转到第 6 页 确定

现在处于最后一页，当添加图书后书籍会自动在最后一页显示。

删除图书



书名	作者	价格	库存	类型	操作	
如何获取富婆芳心	小case	75.00	20	生活	修改	删除
中华上下五千年	炎帝	115.00	30	历史	修改	删除
深入理解计算机系统	大卫	85.00	132	编程	修改	删除

添加图书

修改图书

书名	作者	价格	库存	类型	操作	
如何获取富婆芳心	小case	75.00	20	生活	修改	删除
中华上下五千年	炎帝	115.00	30	历史	修改	删除
深入理解计算机系统	兰德尔	95.00	132	编程	修改	删除

添加图书

首页 上一页 第【6】页 下一页 末页 共[6]页 [23]条记录 转到第 6 页 确定

五、思考与总结

通过这次 web 大作业我深刻理解了网站前台和后台的服务和分工合作的关系，后台对于用户服务的需求实现至关重要，在实现页面跳转以及数据修改和保存方面我也遇到了很多问题，一直遇到的 404、500 问题也提升了我对代码的 debug 能力。让我的编程思维又进一步得到了提升。同时也收获到了有关于网页和网页响应方面的知识。