**网上书店系统**

**--JavaEE大型实验报告**

**学生姓名：Rush Chuh**

**学号：\*\*\***

**班级：\*\*\***

**指导老师：\*\*\***

****

**完成日期：2015.1.3**

2015

目录

[1、系统概述 2](#_Toc439596950)

[1.1 开发意义 2](#_Toc439596951)

[1.2 系统结构 2](#_Toc439596952)

[1.3 主要功能 3](#_Toc439596953)

[2、系统框架 3](#_Toc439596954)

[2.1 功能架构 3](#_Toc439596955)

[2.2 系统架构 3](#_Toc439596956)

[2.3 模块功能说明 5](#_Toc439596957)

[3、系统详细设计 5](#_Toc439596958)

[3.1 实体类 5](#_Toc439596959)

[3.1.1 用户类 5](#_Toc439596960)

[3.1.2 图书类 6](#_Toc439596961)

[3.1.3 订单类 8](#_Toc439596962)

[3.1.4 订单项类 9](#_Toc439596963)

[3.2 业务逻辑类 10](#_Toc439596964)

[3.2.1 用户逻辑类 10](#_Toc439596965)

[3.2.2 图书逻辑类 11](#_Toc439596966)

[3.2.3 订单逻辑类 12](#_Toc439596967)

[3.3 数据访问类 13](#_Toc439596968)

[3.3.1 用户数据访问类 13](#_Toc439596969)

[3.3.2 图书数据访问类 14](#_Toc439596970)

[3.3.3 订单数据访问类 15](#_Toc439596971)

[3.4 控制器类 16](#_Toc439596972)

[3.4.1 用户控制器类 16](#_Toc439596973)

[3.4.2 图书控制器类 18](#_Toc439596974)

[3.4.3 订单控制器类 20](#_Toc439596975)

[3.5 配置文件 22](#_Toc439596976)

[3.5.1 web.xml 22](#_Toc439596977)

[3.5.2 beans.xml 23](#_Toc439596978)

[3.5.3 struts.xml 26](#_Toc439596979)

[4、数据库设计 27](#_Toc439596980)

[4.1 数据库表 27](#_Toc439596981)

[4.1.1 用户表 27](#_Toc439596982)

[4.1.2 图书表 28](#_Toc439596983)

[4.1.3 订单表 28](#_Toc439596984)

[4.2 概念关系模型 28](#_Toc439596985)

[4.3 类的设计 29](#_Toc439596986)

[4.3.1 实体类 29](#_Toc439596987)

[4.3.2 业务逻辑类 30](#_Toc439596988)

[4.3.3 数据访问类 30](#_Toc439596989)

[4.3.4 控制器类 30](#_Toc439596990)

[4.3.5 工具类 31](#_Toc439596991)

[4.3.6 测试类 31](#_Toc439596992)

[5、系统实现模块 31](#_Toc439596993)

[6、总结 46](#_Toc439596994)

[6.1 存在的问题 46](#_Toc439596995)

[6.2 几点建议 47](#_Toc439596996)

# 1、系统概述

## 1.1 开发意义

随着网络的迅速发展，当今电子商务已经被广大的互联网用户所接受，网上书店已是必不可少的经营策略。由于互联网的普及，网上书店系统可以为读者寻找他们喜爱的书，并且永不下班关门，读者可以随时随地查询和订购图书，无需走出家门，即可亲临书店，并且提供简捷的检索方式，同时还减少了实体商店产生的费用支出。网上书店已经逐渐成为替代传统书店的最有效购书方法。网上书店具有简便的操作方式，使人足不出户，即可阅读各处图书。

另外，图书拥有单价低、标准化的特征，购买风险较小，在线购买的方式被更多网民接受。因此，图书网上交易是图书销售领域未来发展的必然趋势，也是出版发行业摆脱困境，建立新的流通渠道，繁荣出版，繁荣市场的必由之路。

## 1.2 系统结构

说明：采用如图1-1所示的B/S三层结构，保证了其应用的通用性和跨平台性；B/S的应用只需在客户端安装通用的浏览器即可，维护和升级工作都在服务器端进行，不需对客户端进行任何改变，故而大大降低了开发和维护的成本。



图1-1 B/S三层结构图

## 1.3 主要功能

网上书店的主要功能是让未注册用户能浏览书店的图书，已注册用户可以浏览图书并挑选喜欢的图书进行购买，然后可以进行订单查询，取代现实中的书店，为用户购书带来方便快捷的购物体验，同时也可以增加图书的销量，达到商家和顾客共赢的宗旨。

# 2、系统框架

## 2.1 功能架构

说明：如图2-1所示，网上书店分为四个部分，分别是用户操作、图书操作、我的购物车和我的订单，其中用户操作又分为：用户的添加、更新、删除、登录和登出。图书操作分为：图书的添加、更新和删除。我的购物车中具有更新购物车的功能。我的订单中具有订单添加和查询的功能。



图2-1 功能架构图

## 2.2 系统架构

说明：如图2-2所示，为系统架构图，系统分为三层架构，有关三层架构的详细内容请看下面的介绍。



图2-2 系统架构图

三层体系的应用程序将业务规则、数据访问、合法性校验等工作放到了中间层进行处理。通常情况下，客户端不直接与数据库进行交互，而是通过COM/DCOM通讯与中间层建立连接，再经由中间层与数据库进行交互。

客户层：主要表示WEB方式，也可以表示成WINFORM方式，WEB方式也可以表现成：aspx，如果业务层相当强大和完善，无论表现层如何定义和更改，逻辑层都能完善地提供服务。

业务层：主要是针对具体的问题的操作，也可以理解成对数据层的操作，对数据业务逻辑处理，如果说数据层是积木，那逻辑层就是对这些积木的搭建。

数据层：主要是对非原始数据的操作层，而不是指原始数据，也就是说，是对数据的操作，而不是数据库，具体为业务逻辑层或表示层提供数据服务。

优点：1、开发人员可以只关注整个结构中的其中某一层；

2、可以很容易的用新的实现来替换原有层次的实现；

3、可以降低层与层之间的依赖；

4、有利于标准化；

5、利于各层逻辑的复用；

6、结构更加的明确；

7、在后期维护的时候，极大地降低了维护成本和维护时

间。

## 2.3 模块功能说明

用户添加功能：游客可以通过用户添加功能成为网上书店的用户，进行图书的购买操作，管理员也可以直接进行用户的添加。

用户更新功能：收到用户修改密码的请求后，管理员可以对用户的密码进行修改。

用户删除功能：管理员可以对很久未登录用户进行删除操作。

用户登录功能：已注册过的会员可以通过登录功能对图书进行购买。

用户登出功能：想要切换账号的用户可以根据用户登出功能进行用户的切换。

图书添加功能：管理员对图书进行增加操作。

图书修改功能：管理员对图书的各类信息进行更新。

图书删除功能：管理员对售空或不再进行销售的图书进行删除。

更新购物车功能：用户使用更新购物车功能对购物车中图书的数量进行更新操作。

订单添加功能：用户点击结算后，会在订单表中添加一个订单，即购买成功。

订单查询功能：购买成功后，可以查看当前订单的各类信息。

# 3、系统详细设计

## 3.1 实体类

### 3.1.1 用户类

1、简介

映射到数据库中的user表，记录用户的用户名及密码。

2、变量说明

|  |  |  |
| --- | --- | --- |
| 数据类型 | 变量名 | 变量作用 |
| String | username | 定义用户名 |
| String | password | 定义密码 |

3、方法简要说明

|  |  |
| --- | --- |
| 方法名及参数 | 方法作用 |
| public User() | 无参构造方法 |
| public User(……) | 有参构造方法，初始化所有值 |
| public String getUsername() | 获取当前用户姓名 |
| public void setUsername(……) | 向当前用户写入姓名 |
| public String getPassword() | 获取当前用户密码 |
| public void setPassword(……) | 向当前用户写入密码 |

4、方法详细说明

//构造方法

public User(){}

public User(String username, String password){

this.username = username;

this.password = password;

}

public String getUsername() {

return username;

}

public void setUsername(String username) {

this.username = username;

}

public String getPassword() {

return password;

}

public void setPassword(String password) {

this.password = password;

}

### 3.1.2 图书类

1、简介

映射到数据库中的book表，记录图书的书号、书名、价格和图片。

2、变量说明

|  |  |  |
| --- | --- | --- |
| 数据类型 | 变量名 | 变量作用 |
| Integer | bookid | 定义书号 |
| String | bookname | 定义书名 |
| Double | price | 定义价格 |
| String | picture | 定义图书图片 |

3、方法简要说明

|  |  |
| --- | --- |
| 方法名及参数 | 方法作用 |
| public Book() | 无参构造方法 |
| public Book(……) | 有参构造方法，初始化所有值 |
| public Integer getBookid() | 获取当前图书的书号 |
| public void setBookid(……) | 向当前图书写入书号 |
| public String getBookname() | 获取当前图书的书名 |
| public void setBookname(……) | 向当前图书写入书名 |
| public Double getPrice() | 获取当前图书的价格 |
| public void setPrice(……) | 向当前图书写入价格 |
| public String getPicture() | 获取当前图书的图片地址 |
| public void setPicture(……) | 向当前图书写入图片地址 |

4、方法详细说明

//构造方法

public Book() {}

public Book(String bookname, Double price, String picture) {

this.setBookname(bookname);

this.setPrice(price);

this.setPicture(picture);

}

public Integer getBookid() {

return bookid;

}

public void setBookid(Integer bookid) {

this.bookid = bookid;

}

public String getBookname() {

return bookname;

}

public void setBookname(String bookname) {

this.bookname = bookname;

}

public Double getPrice() {

return price;

}

public void setPrice(Double price) {

this.price = price;

}

public String getPicture() {

return picture;

}

public void setPicture(String picture) {

this.picture = picture;

}

### 3.1.3 订单类

1、简介

映射到数据库的orders表，记录订单的订单号、用户、订单完成时间和订单项。

2、变量说明

|  |  |  |
| --- | --- | --- |
| 数据类型 | 变量名 | 变量作用 |
| Integer | orderid | 定义订单号 |
| User | user | 定义用户 |
| Date | orderdate | 定义下订单时间 |
| Set | orderitems | 定义订单项 |

3、方法简要说明

|  |  |
| --- | --- |
| 方法名及参数 | 方法作用 |
| public Orders() | 无参构造方法 |
| public Orders(……) | 有参构造方法，初始化所有值 |
| public Integer getOrderid() | 获取当前订单的单号 |
| public void setOrderid(……) | 向当前订单写入单号 |
| public User getUser() | 获取当前订单的用户 |
| public void setUser(……) | 向当前订单写入用户 |
| public Date getOrderdate() | 获取当前订单的时间 |
| public void setOrderdate(……) | 向当前订单写入时间 |
| public Set getOrderitems() | 获取当前订单的订单项 |
| public void setOrderitems(……) | 向当前订单写入订单项 |

4、方法详细说明

//构造方法

public Orders(){}

public Orders(Integer orderid,User user,Date orderdate){

this.orderid = orderid;

this.user=user;

this.orderdate = orderdate;

}

public Integer getOrderid() {

return orderid;

}

public void setOrderid(Integer orderid) {

this.orderid = orderid;

}

public User getUser() {

return user;

}

public void setUser(User user) {

this.user = user;

}

public Date getOrderdate() {

return orderdate;

}

public void setOrderdate(Date orderdate) {

this.orderdate = orderdate;

}

public Set getOrderitems() {

return orderitems;

}

public void setOrderitems(Set orderitems) {

this.orderitems = orderitems;

}

### 3.1.4 订单项类

1、简介

记录订单项号、购物车里的每本图书、订单项所属的订单及每本图书的数量。

2、变量说明

|  |  |  |
| --- | --- | --- |
| 数据类型 | 变量名 | 变量作用 |
| Integer | orderitemid | 定义订单项号 |
| Book | book | 定义图书 |
| Orders | orders | 定义订单项所属的订单 |
| Integer | quantity | 定义数量 |

3、方法简要说明

|  |  |
| --- | --- |
| 方法名及参数 | 方法作用 |
| public Integer getOrderitemid() | 获取当前订单项号 |
| public void setOrderitemid(……) | 向当前订单项写入订单项号 |
| public Book getBook() | 获取当前订单项中的图书 |
| public void setBook(……) | 向当前订单项写入图书 |
| public Orders getOrders() | 获取当前订单项所属订单 |
| public void setOrders(……) | 向当前订单项写入订单 |
| public Integer getQuantity() | 获取当前订单项中图书的数量 |
| public void setQuantity() | 向当前订单项写入图书的数量 |

4、方法详细说明

public Integer getOrderitemid() {

return orderitemid;

}

public void setOrderitemid(Integer orderitemid) {

this.orderitemid = orderitemid;

}

public Book getBook() {

return book;

}

public void setBook(Book book) {

this.book = book;

}

public Orders getOrders() {

return orders;

}

public void setOrders(Orders orders) {

this.orders = orders;

}

public Integer getQuantity() {

return quantity;

}

public void setQuantity(Integer quantity) {

this.quantity = quantity;

}

## 3.2 业务逻辑类

### 3.2.1 用户逻辑类

1、简介

传输客户层的用户数据，然后通过调用用户数据访问类的方法进行各项操作：添加、删除、有效性验证、退出、查找和更新。

2、变量说明

|  |  |  |
| --- | --- | --- |
| 数据类型 | 变量名 | 变量作用 |
| IUserDAO | userDAO | 定义用户数据访问类的对象 |

3、方法简要说明

|  |  |
| --- | --- |
| 方法名及参数 | 方法作用 |
| public void setUserDAO(……) | 获取用户数据访问类的对象 |
| public void saveUser(……) | 调用DAO类的saveUser方法 |
| public void deleteUser(……) | 调用DAO类的deleteUser方法 |
| public User validateUser(……) | 调用DAO类的validateUser方法 |
| public boolean exitUser(……) | 调用DAO类的exitUser方法 |
| public User findUser(……) | 调用DAO类的findUser方法 |
| public void updateUser(……) | 调用DAO类的updateUser方法 |

4、方法详细说明

public void setUserDAO(IUserDAO userDAO) {

this.userDAO = userDAO;

}

public void saveUser(User user) {

this.userDAO.saveUser(user);

}

public void deleteUser(String username) {

this.userDAO.deleteUser(username);

}

public User validateUser(String username, String password) {

return userDAO.validateUser(username, password);

}

public boolean exitUser(String username) {

return userDAO.exitUser(username);

}

public User findUser(String username) {

return userDAO.findUser(username);

}

public void updateUser(User user) {

this.userDAO.updateUser(user);

}

### 3.2.2 图书逻辑类

1、简介

传输客户层的图书数据，然后通过调用图书数据访问类的方法进行各项操作：添加、删除、更新、查找、按书名获取图书和按书号获取图书。

2、变量说明

|  |  |  |
| --- | --- | --- |
| 数据类型 | 变量名 | 变量作用 |
| IBookDAO | bookDAO | 定义图书数据访问类的对象 |

3、方法简要说明

|  |  |
| --- | --- |
| 方法名及参数 | 方法作用 |
| public void setBookDAO(……) | 获取图书数据访问类的对象 |
| public void saveBook(……) | 调用DAO类的saveBook方法 |
| public void deleteBook(……) | 调用DAO类的deleteBook方法 |
| public void updateBook(……) | 调用DAO类的updateBook方法 |
| public Book findBook(……) | 调用DAO类的findBook方法 |
| public List<?> getRequiredBookByName(……) | 调用DAO类的getRequiredBookByName方法 |
| public Book getBookById(……) | 调用DAO类的getBookById方法 |

4、方法详细说明

public void setBookDAO(IBookDAO bookDAO) {

this.bookDAO = bookDAO;

}

public void saveBook(Book book) {

this.bookDAO.saveBook(book);

}

public void deleteBook(Integer bookid) {

this.bookDAO.deleteBook(bookid);

}

public void updateBook(Book book) {

this.bookDAO.updateBook(book);

}

public Book findBook(Integer bookid) {

return bookDAO.findBook(bookid);

}

public List<?> getRequiredBookByName(String name) {

return bookDAO.getRequiredBookByName(name);

}

public Book getBookById(Integer bookid) {

return bookDAO.getBookById(bookid);

}

### 3.2.3 订单逻辑类

1、简介

传输客户层的订单数据，然后通过调用订单数据访问类的方法进行添加操作。

2、变量说明

|  |  |  |
| --- | --- | --- |
| 数据类型 | 变量名 | 变量作用 |
| IOrderDAO | orderDAO | 定义订单数据访问类的对象 |

3、方法简要说明

|  |  |
| --- | --- |
| 方法名及参数 | 方法作用 |
| public void setOrderDAO(……) | 获取订单数据访问类的对象 |
| public void saveOrder(……) | 调用DAO类的saveOrder方法 |

4、方法详细说明

public void saveOrder(Orders order) {

this.orderDAO.saveOrder(order);

}

public void setOrderDAO(IOrderDAO orderDAO){

this.orderDAO=orderDAO;

}

## 3.3 数据访问类

### 3.3.1 用户数据访问类

1、简介

直接对业务层传输来的用户数据进行各项操作：添加、删除、有效性验证、退出、查找和更新。

2、变量说明

|  |  |  |
| --- | --- | --- |
| 数据类型 | 变量名 | 变量作用 |
| SessionFactory | factory | 定义会话工厂类的对象 |

3、方法简要说明

|  |  |
| --- | --- |
| 方法名及参数 | 方法作用 |
| public void saveUser(……) | 添加新用户 |
| public void deleteUser(……) | 删除用户 |
| public User validateUser(……) | 验证用户的有效性 |
| public Boolean exitUser(……) | 退出用户 |
| public User findUser(……) | 查找用户 |
| public void updateUser(……) | 更新用户 |

4、方法详细说明

public void saveUser(User user) {

this.getHibernateTemplate().save(user);

}

public void deleteUser(String username) {

this.getHibernateTemplate().delete(this.getHibernateTemplate().load(User.class, username));

}

public User validateUser(String username, String password) {

String str[] = { username, password };

List<?> list = this.getHibernateTemplate().find(

"from User u where u.username=? and u.password=?", str);

if (list.size() > 0) {

User user = (User) list.get(0);

return user;

} else

return null;

}

public boolean exitUser(String username) {

List<?> list = this.getHibernateTemplate().find(

"from User where username=?", username);

if (list.size() > 0)

return true;

else

return false;

}

public User findUser(String username) {

return (User)factory.getCurrentSession().get(User.class, username);

}

public void updateUser(User user) {

factory.getCurrentSession().merge(user);

}

### 3.3.2 图书数据访问类

1、简介

直接对业务层传输来的图书数据进行各项操作：添加、删除、更新、查找、根据书名获取图书和根据书号获取图书。

2、变量说明

|  |  |  |
| --- | --- | --- |
| 数据类型 | 变量名 | 变量作用 |
| SessionFactory | factory | 定义会话工厂类的对象 |

3、方法简要说明

|  |  |
| --- | --- |
| 方法名及参数 | 方法作用 |
| public void saveBook(……) | 添加图书 |
| public void deleteBook(……) | 删除图书 |
| public User updateBook(……) | 更新图书 |
| public Book findBook(……) | 查找图书 |
| public List<?> getRequiredBookByName(……) | 根据书名来获取符合条件的图书对象 |
| public Book getBookById(……) | 根据书号来获取图书对象 |

4、方法详细说明

public void saveBook(Book book) {

this.getHibernateTemplate().save(book);

}

public void deleteBook(Integer bookid) {

this.getHibernateTemplate().delete(this.getHibernateTemplate().load(Book.class, bookid));

}

public void updateBook(Book book) {

factory.getCurrentSession().merge(book);//saveOrUpdate

}

public Book findBook(Integer bookid) {

return (Book)factory.getCurrentSession().get(Book.class, bookid);

}

public List<?> getRequiredBookByName(String name) {

try {

List<?> books = this.getHibernateTemplate().find("from Book where bookname like '%" + name + "%'");

return books;

} catch (Exception e) {

e.printStackTrace();

return null;

}

}

public Book getBookById(Integer bookid) {

return (Book)this.getHibernateTemplate().get(Book.class, bookid);

}

### 3.3.3 订单数据访问类

1、简介

直接对业务层传输来的订单数据进行添加操作。

2、变量说明

订单数据访问类中无任何变量。

3、方法简要说明

|  |  |
| --- | --- |
| 方法名及参数 | 方法作用 |
| public void saveOrder(……) | 添加订单 |

4、方法详细说明

public void saveOrder(Orders order) {

this.getHibernateTemplate().save(order);

}

## 3.4 控制器类

### 3.4.1 用户控制器类

1、简介

通过用户控制器类，可以接收浏览器的请求，并返回相应的页面处理：注册成功、删除成功、更新成功、登录成功和登出等。

2、变量说明

|  |  |  |
| --- | --- | --- |
| 数据类型 | 变量名 | 变量作用 |
| User | user | 定义用户类的对象 |
| IUserService | userService | 定义用户逻辑类的对象 |

3、方法简要说明

|  |  |
| --- | --- |
| 方法名及参数 | 方法作用 |
| public User getUser() | 获取当前用户 |
| public void setUser(……) | 写入当前用户 |
| public IUserService getUserService() | 获取当前用户逻辑类对象 |
| public void setUserService(……) | 写入当前用户逻辑类对象 |
| public String register() | 注册用户并返回操作是否成功 |
| public String delete() | 删除用户并返回操作是否成功 |
| public String update() | 更新用户并返回操作是否成功 |
| public String login() | 登录用户并返回操作是否成功 |
| public String logout() | 登出用户并返回操作是否成功 |

4、方法详细说明

public User getUser() {

return this.user;

}

public void setUser(User user) {

this.user = user;

}

public IUserService getUserService() {

return userService;

}

public void setUserService(IUserService userService) {

this.userService = userService;

}

public String register() throws Exception {

System.out.println(user.getUsername());

User user1 = new User();

user1.setUsername(user.getUsername());

user1.setPassword(user.getPassword());

userService.saveUser(user1);

return SUCCESS;

}

public String delete() throws Exception {

userService.deleteUser(user.getUsername());

return SUCCESS;

}

public String update() throws Exception{

String tmpUsername=user.getUsername();

String tmpPassword=user.getPassword();

User user=userService.findUser(tmpUsername);

user.setPassword(tmpPassword);

userService.updateUser(user);

return SUCCESS;

}

public String login() throws Exception{

User u=userService.validateUser(user.getUsername(), user.getPassword());

if(u!=null){

Map session = ActionContext.getContext().getSession();

session.put("user", u);

return SUCCESS;

}

else {

return ERROR;

}

}

public String logout() throws Exception {

Map session = ActionContext.getContext().getSession();

session.remove("user");

session.remove("cart");

return SUCCESS;

}

### 3.4.2 图书控制器类

1、简介

通过图书控制器类，可以接收浏览器的请求，并返回相应的页面处理：添加成功、删除成功、更新成功、图书搜索和浏览图书等。

2、变量说明

|  |  |  |
| --- | --- | --- |
| 数据类型 | 变量名 | 变量作用 |
| Book | book | 定义图书类的对象 |
| IBookService | bookService | 定义图书逻辑类的对象 |
| String | bookname | 定义书名 |

3、方法简要说明

|  |  |
| --- | --- |
| 方法名及参数 | 方法作用 |
| public Book getBook() | 获取当前图书 |
| public void setBook(……) | 写入当前图书 |
| public IBookService getBookService() | 获取当前图书逻辑类对象 |
| public void setBookService(……) | 写入当前图书逻辑类对象 |
| public String getBookname() | 获取当前书名 |
| public void setBookname(……) | 写入当前书名 |
| public String add() | 添加图书并返回操作是否成功 |
| public String delete() | 删除图书并返回操作是否成功 |
| public String update() | 更新图书并返回操作是否成功 |
| public String search() | 搜索图书并返回操作是否成功 |
| public String browseBookPaging() | 列出符合条件的图书 |

4、方法详细说明

public Book getBook() {

return book;

}

public void setBook(Book book) {

this.book = book;

}

public IBookService getBookService() {

return bookService;

}

public void setBookService(IBookService bookService) {

this.bookService = bookService;

}

public String getBookname() {

return bookname;

}

public void setBookname(String bookname) {

this.bookname = bookname;

}

public String add() throws Exception {

System.out.println(book.getBookname());

Book book1 = new Book();

book1.setBookid(book.getBookid());

book1.setBookname(book.getBookname());

book1.setPrice(book.getPrice());

book1.setPicture(book.getPicture());

bookService.saveBook(book1);

return "success";

}

public String delete() throws Exception{

bookService.deleteBook(book.getBookid());

return "success";

}

public String update() throws Exception{

Integer tmpBookid=book.getBookid();

String tmpBookname=book.getBookname();

Double tmpPrice=book.getPrice();

String tmpPicture=book.getPicture();

Book book=bookService.findBook(tmpBookid);

book.setBookname(tmpBookname);

book.setPrice(tmpPrice);

book.setPicture(tmpPicture);

bookService.updateBook(book);

return "success";

}

public String search() throws Exception {

List books = bookService.getRequiredBookByName(this.getBookname());

Map request = (Map) ActionContext.getContext().get("request");

System.out.println(bookname);

request.put("books", books);

return "success";

}

public String browseBookPaging() throws Exception {

List books = bookService.getRequiredBookByName(this.getBookname());

Map request = (Map) ActionContext.getContext().get("request");

request.put("books", books);

return "success";

}

### 3.4.3 订单控制器类

1、简介

通过订单控制器类，可以接收浏览器的请求，并返回相应的页面处理：添加到购物车、更新购物车、结算和订单展示等。

2、变量说明

|  |  |  |
| --- | --- | --- |
| 数据类型 | 变量名 | 变量作用 |
| int | bookid | 定义图书书号 |
| int | quantity | 定义图书数量 |
| Orders | order | 定义订单类的对象 |
| IBookService | bookService | 定义图书逻辑类的对象 |
| IOrderService | orderService | 定义订单逻辑类的对象 |

3、方法简要说明

|  |  |
| --- | --- |
| 方法名及参数 | 方法作用 |
| public int getBookid() | 获取当前图书的书号 |
| public void setBookid(……) | 写入当前图书的书号 |
| public int getQuantity() | 获取当前图书的数量 |
| public void setQuantity(……) | 写入当前图书的数量 |
| public Orders getOrder() | 获取当前订单 |
| public void setOrder(……) | 写入当前订单 |
| public IBookService getBookService() | 获取当前图书逻辑类的对象 |
| public void setBookService(……) | 写入当前图书逻辑类的对象 |
| public IOrderService getOrderService() | 获取当前订单逻辑类的对象 |
| public void setOrderService(……) | 写入当前订单逻辑类的对象 |
| public String addToCart() | 将图书对象添加到购物车 |
| public String updateCart() | 更新购物车的内容 |
| public String checkout() | 结算 |
| public String viewOrders() | 将刚刚完成的订单显示出来 |

4、方法详细说明

public int getBookid() {

return bookid;

}

public void setBookid(int bookid) {

this.bookid = bookid;

}

public int getQuantity() {

return quantity;

}

public void setQuantity(int quantity) {

this.quantity = quantity;

}

public Orders getOrder() {

return order;

}

public void setOrder(Orders order) {

this.order = order;

}

public IBookService getBookService() {

return bookService;

}

public void setBookService(IBookService bookService) {

this.bookService = bookService;

}

public IOrderService getOrderService() {

return orderService;

}

public void setOrderService(IOrderService orderService) {

this.orderService = orderService;

}

public String addToCart() throws Exception {

Book book = bookService.getBookById(bookid);

Orderitem orderitem = new Orderitem();

orderitem.setBook(book);

orderitem.setQuantity(quantity);

Map session = ActionContext.getContext().getSession();

Cart cart = (Cart) session.get("cart");

if (cart == null) {

cart = new Cart();

}

cart.addBook(bookid, orderitem);

session.put("cart", cart);

return "success";

}

public String updateCart() throws Exception { //更新购物车

Map session = ActionContext.getContext().getSession();

Cart cart = (Cart) session.get("cart");

//直接调用购物车模型中的方法实现修改图书数量

cart.updateCart(bookid, this.getQuantity());

System.out.println(this.getQuantity());

session.put("cart", cart);

return "success";

}

public String checkout() throws Exception { //结算

Map session = ActionContext.getContext().getSession();

User user = (User) session.get("user");

Cart cart = (Cart) session.get("cart");

if (user == null || cart == null)

return ActionSupport.ERROR; //如果没有登录返回登录界面

Orders order = new Orders();

order.setOrderdate(new Date());

order.setUser(user);

for (Iterator it = cart.getItems().values().iterator(); it.hasNext();) {

Orderitem orderitem = (Orderitem) it.next();

orderitem.setOrders(order);

order.getOrderitems().add(orderitem);

}

orderService.saveOrder(order);

Map Session = ActionContext.getContext().getSession();

Session.put("orders", order);

Map request = (Map) ActionContext.getContext().get("request");

request.put("order", order);

return "success";

}

public String viewOrders()throws Exception { //展示订单

Map Session = ActionContext.getContext().getSession();

Orders order = (Orders) Session.get("orders");

Map request = (Map) ActionContext.getContext().get("request");

request.put("order", order);

System.out.println(order.getOrderdate());

return "success";

}

## 3.5 配置文件

### 3.5.1 web.xml

<?xml version="1.0" encoding="UTF-8"?>

<web-app version="2.4"

xmlns="http://java.sun.com/xml/ns/j2ee"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee

http://java.sun.com/xml/ns/j2ee/web-app\_2\_4.xsd">

<!-- 指定spring的配置文件，默认从web根目录寻找配置文件，我们可以通过spring提供的classpath:前缀指定从类路径下寻找 -->

<context-param>

<param-name>contextConfigLocation</param-name>

<param-value>classpath:beans.xml</param-value>

</context-param>

<!--对Spring容器进行实例化-->

<listener>

<listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>

</listener>

<!--配置struts2-->

<filter>

<filter-name>struts2</filter-name>

<filter-class>org.apache.struts2.dispatcher.FilterDispatcher</filter-class>

</filter>

<filter-mapping>

<filter-name>struts2</filter-name>

<url-pattern>/\*</url-pattern>

</filter-mapping>

<welcome-file-list>

<welcome-file>index.jsp</welcome-file>

</welcome-file-list>

</web-app>

### 3.5.2 beans.xml

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xmlns:context="http://www.springframework.org/schema/context"

xmlns:aop="http://www.springframework.org/schema/aop"

xmlns:tx="http://www.springframework.org/schema/tx"

xsi:schemaLocation="http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans-2.5.xsd

http://www.springframework.org/schema/context

http://www.springframework.org/schema/context/spring-context-2.5.xsd

http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop-2.5.xsd

http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx-2.5.xsd">

<context:component-scan base-package="pers.rush.bookstore"/>

<bean id="dataSource" class="com.mchange.v2.c3p0.ComboPooledDataSource" destroy-method="close">

<property name="driverClass" value="org.gjt.mm.mysql.Driver"/>

<property name="jdbcUrl" value="jdbc:mysql://localhost:3306/bookstore?useUnicode=true&amp;characterEncoding=UTF-8"/>

<property name="user" value="root"/>

<property name="password" value="rush"/>

<!--初始化时获取的连接数，取值应在minPoolSize与maxPoolSize之间。Default: 3 -->

<property name="initialPoolSize" value="1"/>

<!--连接池中保留的最小连接数。-->

<property name="minPoolSize" value="1"/>

<!--连接池中保留的最大连接数。Default: 15 -->

<property name="maxPoolSize" value="300"/>

<!--最大空闲时间,60秒内未使用则连接被丢弃。若为0则永不丢弃。Default: 0 -->

<property name="maxIdleTime" value="60"/>

<!--当连接池中的连接耗尽的时候c3p0一次同时获取的连接数。Default: 3 -->

<property name="acquireIncrement" value="5"/>

<!--每60秒检查所有连接池中的空闲连接。Default: 0 -->

<property name="idleConnectionTestPeriod" value="60"/>

</bean>

<bean id="sessionFactory" class="org.springframework.orm.hibernate3.LocalSessionFactoryBean">

<property name="dataSource" ref="dataSource"/>

<property name="mappingResources">

<list>

<value>pers/rush/bookstore/vo/User.hbm.xml</value>

<value>pers/rush/bookstore/vo/Book.hbm.xml</value>

<value>pers/rush/bookstore/vo/Orders.hbm.xml</value>

<value>pers/rush/bookstore/vo/Orderitem.hbm.xml</value>

</list>

</property>

<property name="hibernateProperties">

<value>

hibernate.dialect=org.hibernate.dialect.MySQL5Dialect

hibernate.hbm2ddl.auto=update

hibernate.show\_sql=false

hibernate.format\_sql=false

</value>

</property>

</bean>

<bean id="userDAO" class="pers.rush.bookstore.dao.impl.UserDAO">

<property name="sessionFactory" ref="sessionFactory"></property>

</bean>

<bean id="userService" class="pers.rush.bookstore.service.impl.UserService">

<property name="userDAO">

<ref bean="userDAO" />

</property>

</bean>

<bean id="userAction" class="pers.rush.bookstore.action.UserAction">

<property name="userService">

<ref bean="userService" />

</property>

</bean>

<bean id="bookDAO" class="pers.rush.bookstore.dao.impl.BookDAO">

<property name="sessionFactory" ref="sessionFactory"></property>

</bean>

<bean id="bookService" class="pers.rush.bookstore.service.impl.BookService">

<property name="bookDAO">

<ref bean="bookDAO" />

</property>

</bean>

<bean id="bookAction" class="pers.rush.bookstore.action.BookAction">

<property name="bookService">

<ref bean="bookService" />

</property>

</bean>

<bean id="orderDAO" class="pers.rush.bookstore.dao.impl.OrderDAO">

<property name="sessionFactory" ref="sessionFactory"></property>

</bean>

<bean id="orderService" class="pers.rush.bookstore.service.impl.OrderService">

<property name="orderDAO">

<ref bean="orderDAO" />

</property>

</bean>

<bean id="orderAction" class="pers.rush.bookstore.action.OrderAction">

<property name="bookService">

<ref bean="bookService" />

</property>

<property name="orderService">

<ref bean="orderService" />

</property>

</bean>

<bean id="txManager" class="org.springframework.orm.hibernate3.HibernateTransactionManager">

<property name="sessionFactory" ref="sessionFactory"/>

</bean>

<!--使用基于注解方式配置事务 -->

<tx:annotation-driven transaction-manager="txManager"/>

</beans>

### 3.5.3 struts.xml

<!DOCTYPE struts PUBLIC

"-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"

"http://struts.apache.org/dtds/struts-2.0.dtd">

<struts>

<constant name="struts.i18n.encoding" value="utf-8" />

<include file="struts-default.xml" />

<package name="default" extends="struts-default">

<!-- action -->

<!-- 用户注册 -->

<action name="user\_register" class="userAction" method="register">

<result name="success">/user\_register\_success.jsp</result>

</action>

<!-- 用户删除 -->

<action name="user\_delete" class="userAction" method="delete">

<result name="success">/user\_delete\_success.jsp</result>

</action>

<!-- 用户更新 -->

<action name="user\_update" class="userAction" method="update">

<result name="success">/user\_update\_success.jsp</result>

</action>

<!-- 用户登录 -->

<action name="user\_login" class="userAction" method="login">

<result name="success">/index.jsp</result>

<result name="error">/user\_login.jsp</result>

</action>

<!-- 用户注销 -->

<action name="logout" class="userAction" method="logout">

<result name="success">/index.jsp</result>

</action>

<!-- 添加图书 -->

<action name="book\_add" class="bookAction" method="add">

<result name="success">/book\_add\_success.jsp</result>

</action>

<!-- 删除图书 -->

<action name="book\_delete" class="bookAction" method="delete">

<result name="success">/book\_delete\_success.jsp</result>

</action>

<!-- 更新图书 -->

<action name="book\_update" class="bookAction" method="update">

<result name="success">/book\_update\_success.jsp</result>

</action>

<!-- 图书搜索 -->

<action name="book\_search" class="bookAction" method="search">

<result name="success">/book\_search.jsp</result>

</action>

<!-- 浏览图书页面 -->

<action name="browseBookPaging" class="bookAction" method="browseBookPaging">

<result name="success">/book\_search.jsp</result>

</action>

<!-- 添加图书到购物车 -->

<action name="addToCart" class="orderAction" method="addToCart">

<result name="success">/cart\_add\_success.jsp</result>

</action>

<!-- 更新购物车 -->

<action name="updateCart" class="orderAction" method="updateCart">

<result name="success">/cart\_show.jsp</result>

</action>

<!-- 结算 -->

<action name="checkout" class="orderAction" method="checkout">

<result name="success">/order\_success.jsp</result>

<result name="error">/user\_login.jsp</result>

</action>

<!-- 订单展示 -->

<action name="viewOrders" class="orderAction" method="viewOrders">

<result name="success">/order\_show.jsp</result>

<result name="error">/user\_login.jsp</result>

</action>

</package>

</struts>

# 4、数据库设计

## 4.1 数据库表

### 4.1.1 用户表

说明：用户表中有两个字段，一个是用户名，一个是密码，其中用户名为主键，密码是非空的，无外键。如图4-1所示。

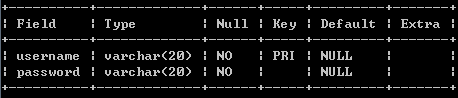


图4-1 用户表结构

### 4.1.2 图书表

说明：图书表中有四个字段，分别是书号、书名、价格和图片的路径，其中书号是主键，而且是自动增长的，无外键。如图4-2所示。

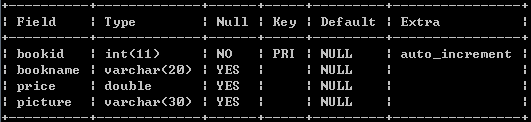


图4-2 图书表结构

### 4.1.3 订单表

说明：订单表中有三个字段，分别是订单号、用户和订单完成时间，其中订单号是主键，而且是自动增长的，用户是外键。如图4-3所示。

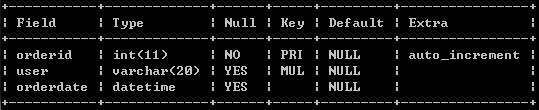


图4-3 订单表结构

## 4.2 概念关系模型

说明：未注册用户可以**注册**多个账号，但是一个账号只能供一个用户同时使用，所以是一对多关系；用户可以**浏览**多本图书，每本图书也可以被多个用户**浏览**，所以是多对多关系；已注册用户可以**购买**多本图书，同样的，每本图书也可以被多个用户**购买**，所以是多对多关系；因为网上书店管理员只设置了一个，管理员可以**管理**所有用户和图书的信息，所以是一对多关系。如图4-4所示。



图4-4 网上书店E-R图

## 4.3 类的设计

### 4.3.1 实体类

说明：实体类分为四个部分，它们分别是：用户类、图书类、订单类和订单项类。每个部分又分为：类和它们的映射文件。如图4-5所示。

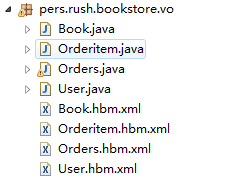


图4-5 实体类图

### 4.3.2 业务逻辑类

说明： 业务逻辑类分为三个部分，它们分别是：用户逻辑类、图书逻辑类和订单逻辑类。每个部分又分为：接口类和具体实现类。如图4-6所示。

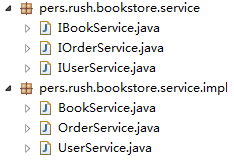


图4-6 业务逻辑类图

### 4.3.3 数据访问类

说明：数据访问分为三个部分，它们分别是：用户数据访问类、图书用户数据访问类和订单用户数据访问类。每个部分又分为：接口类和具体实现类。如图4-7所示。

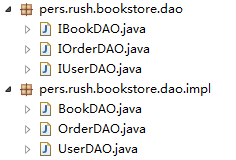


图4-7 数据访问类图

### 4.3.4 控制器类

说明：控制器类也分为三个部分，它们分别是：用户控制器类、图书控制器类和订单控制器类。如图4-8所示。

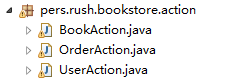


图4-8 控制器类图

### 4.3.5 工具类

说明：工具类包含一个叫购物车的类，用于购物车的各种操作。如图4-9所示。



图4-9 工具类图

### 4.3.6 测试类

说明：测试类同样也分为三个部分，它们分别是：用户测试类、图书测试类和订单测试类，每次创建完一个实体类后都要对其的增删改查进行测试，这就用到了测试类。如图4-10所示。

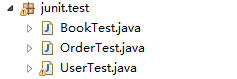


图4-10 测试类图

# 5、系统实现模块

1、欢迎界面

说明：用户登录主界面可以直接进行搜书，费了很长时间才调成跟百度一个样子，网站的主题是绿色，也就是搜索框的颜色，在后面的页面还有许多应用，这样应该比较好看一点吧，右上角显示有登录、免费注册、购物车以及管理员操作的菜单，屏幕下方有关于我们的介绍以及友情链接，这里是链接到了当当网（book.dangdang.com）。如图5-1所示。

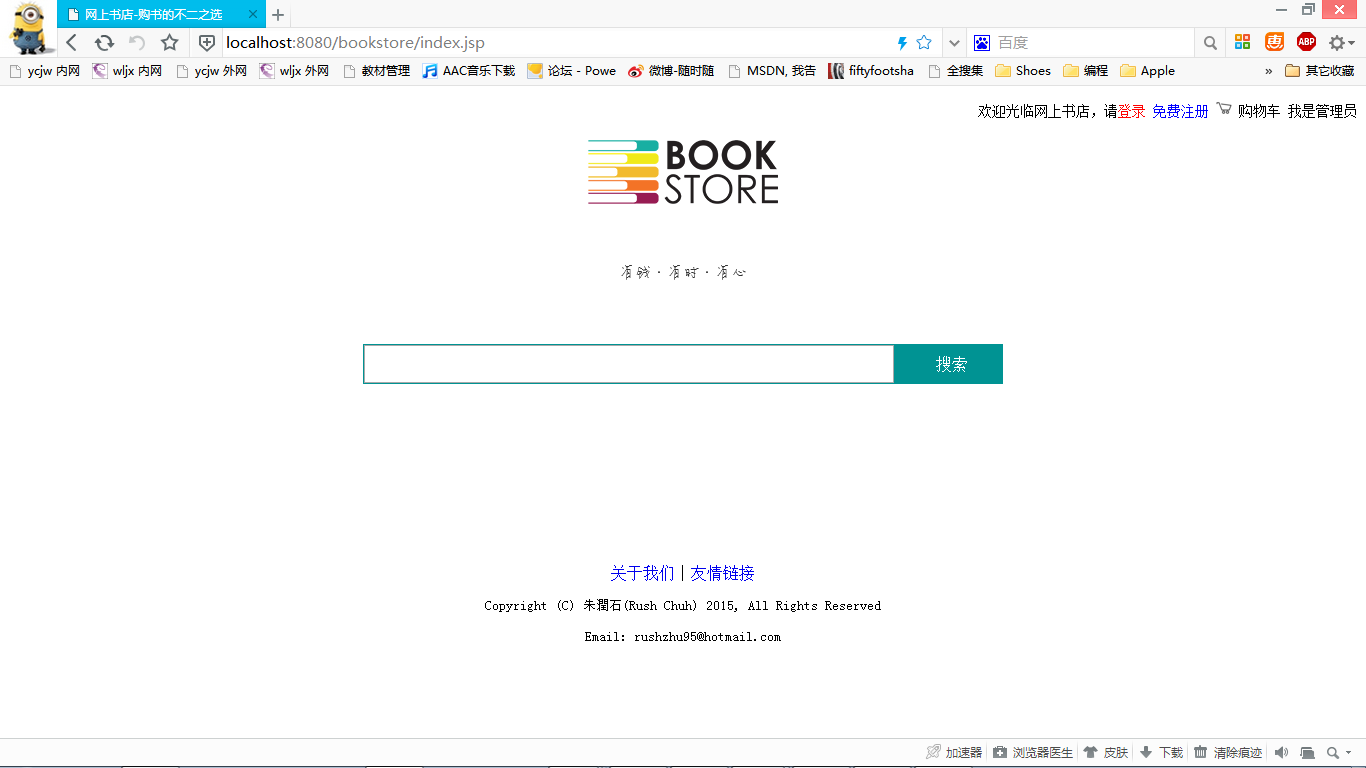


图5-1 欢迎界面

2、管理员操作

说明：在主页面点击右上角的“我是管理员”即可跳转到此页面，管理员操作分为用户管理和图书管理两部分，每个部分又包括添加、更新和删除功能，下面来分别展示下面的各个功能。如图5-2所示。



图5-2 管理员操作

3、添加用户

说明：通过点击“添加用户”按钮即可跳转到此页面，输入想要添加的用户名和密码，点击“立即注册”。



图5-3 添加用户

说明：添加用户成功会有提示信息，如图5-4所示。



图5-4 添加用户成功

说明：查看后台数据库中的用户表，验证是否成功添加用户，如图5-5所示，添加用户成功。

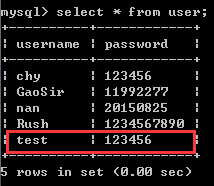


图5-5 后台数据库验证添加成功

4、更新用户

说明：通过点击“更新用户”按钮即可跳转到此页面，输入想要更新的用户名和密码，点击“更新”。



图5-6 更新用户

说明：更新用户成功会有提示信息，如图5-7所示。



图5-7 更新用户成功

说明：查看后台数据库中的用户表，验证是否成功更新用户，如图5-8所示，更新用户成功。

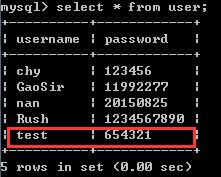


图5-8 后台数据库验证更新成功

5、删除用户

说明：通过点击“删除用户”按钮即可跳转到此页面，输入想要删除的用户名，点击“删除”。



图5-9 删除用户

说明：删除用户成功会有提示信息，如图5-10所示。



图5-10 删除用户成功

说明：查看后台数据库中的用户表，验证是否成功删除用户，如图5-11所示，删除用户成功。

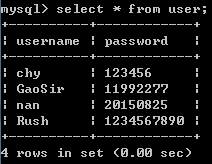


图5-11 后台数据库验证删除成功

6、添加图书

说明：通过点击“添加图书”按钮即可跳转到此页面，输入想要添加的书名、价格和图片，点击“添加”。如图5-12所示。



图5-12 添加图书

说明：添加图书成功会有提示信息，如图5-13所示。



图5-13 添加图书成功

说明：查看后台数据库中的图书表，验证是否成功添加图书，如图5-14所示，添加图书成功。



图5-14后台数据库验证添加成功

7、更新图书

说明：通过点击“更新图书”按钮即可跳转到此页面，输入想要更新的书号、书名、价格和图片，点击“更新”。如图5-15所示。



图5-15 更新图书

说明：更新图书成功会有提示信息，如图5-16所示。



图5-16 更新图书

说明：查看后台数据库中的图书表，验证是否成功更新图书，如图5-17所示，更新图书成功。

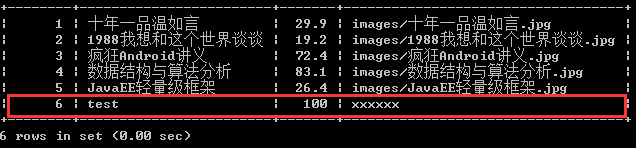


图5-17 后台数据库验证更新成功

8、删除图书

说明：通过点击“删除图书”按钮即可跳转到此页面，输入想要删除的书号，点击“删除”。如图5-18所示。



图5-18 删除图书

说明：删除图书成功会有提示信息，如图5-19所示。



图5-19 删除图书

说明：查看后台数据库中的图书表，验证是否成功删除图书，如图5-17所示，删除图书成功。



图5-20 后台数据库验证删除成功

9、用户注册

说明：因为与管理员操作中的“添加用户”页面一致，所以在此不进行赘述了。如图5-3、5-4、5-5所示。

10、用户登录

说明：用户通过在欢迎界面点击右上角的“登录”按钮，跳转到此页面进行登录。如图5-21所示。



图5-21 用户登录

说明：用户登录成功后自动跳转到欢迎页面，然后对右上角文字进行更新，页面显示如图5-22所示。

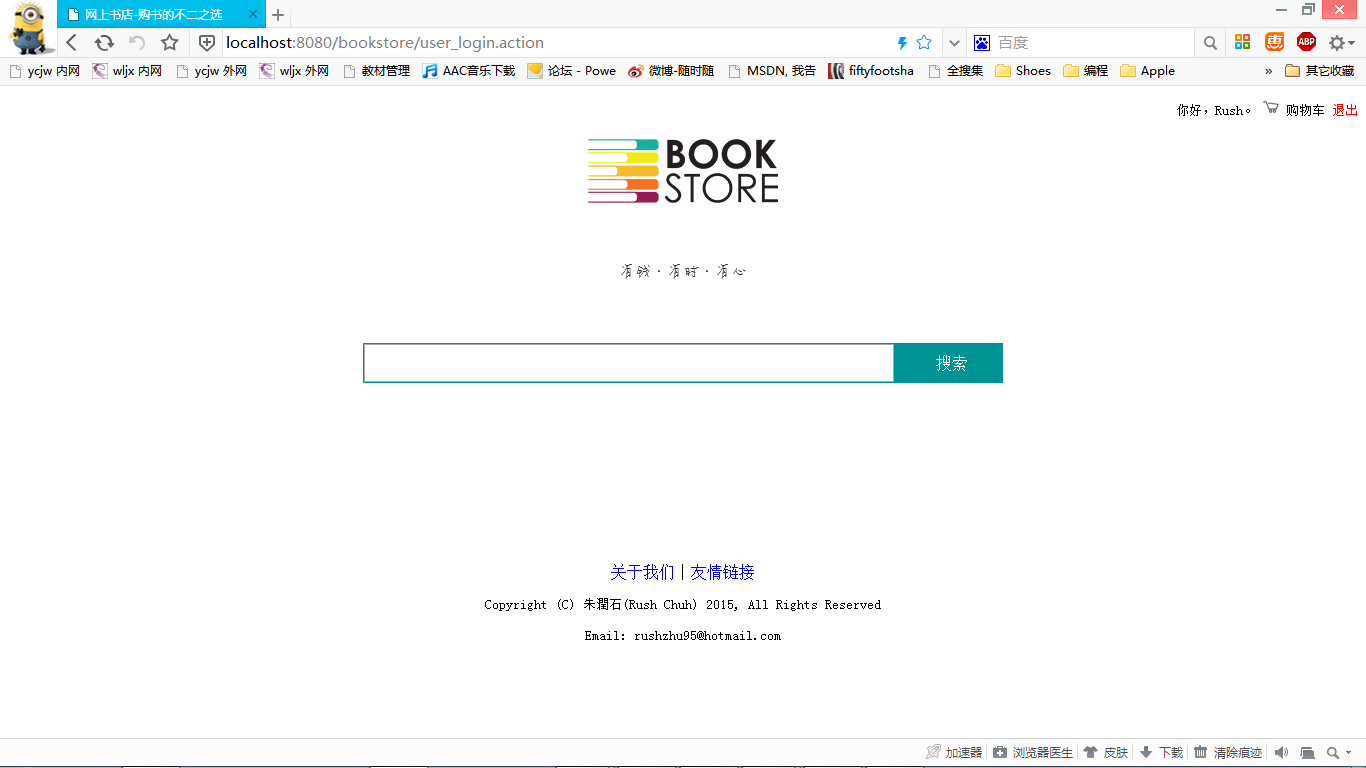


图5-22 用户登录成功

11、用户登出

说明：已登录的用户可以在任何一个页面的右上角点击红色的“退出”按钮，进行用户登出操作。如图5-23所示。

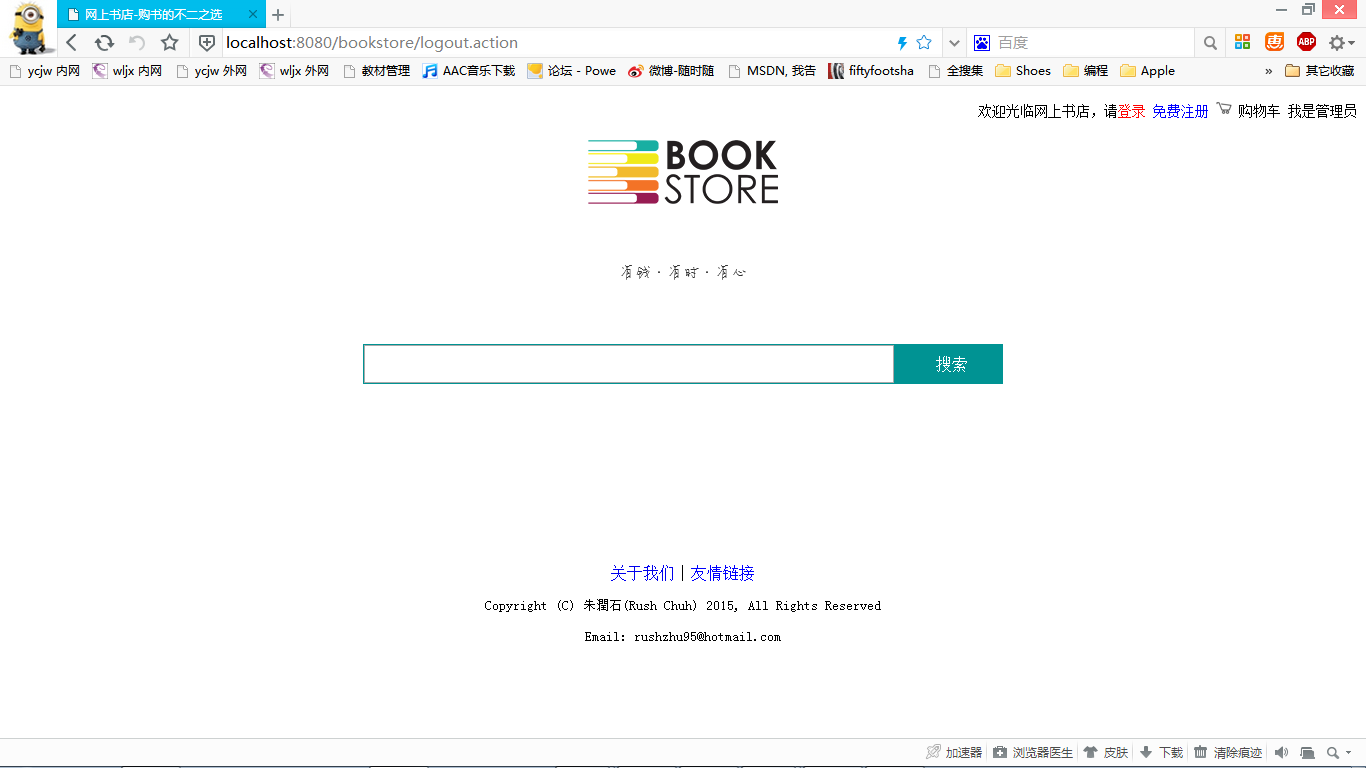


图5-23 用户登出成功

12、图书搜索

说明：在欢迎页面直接点击搜索会将所有图书都罗列出来。如图5-24所示。

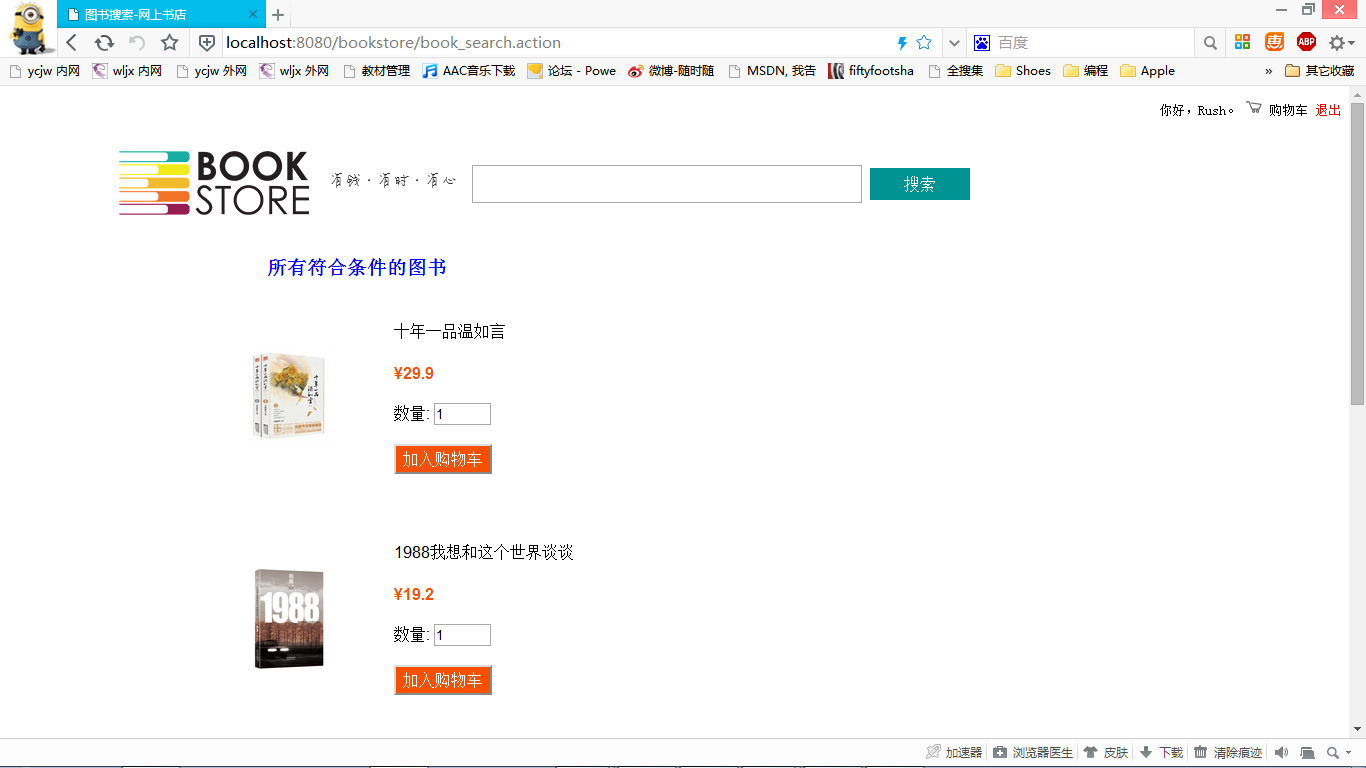


图5-24 图书搜索

说明：在搜索框输入关键字进行图书模糊搜索，这里测试的是输入关键字“a”。搜索结果如图5-25所示。

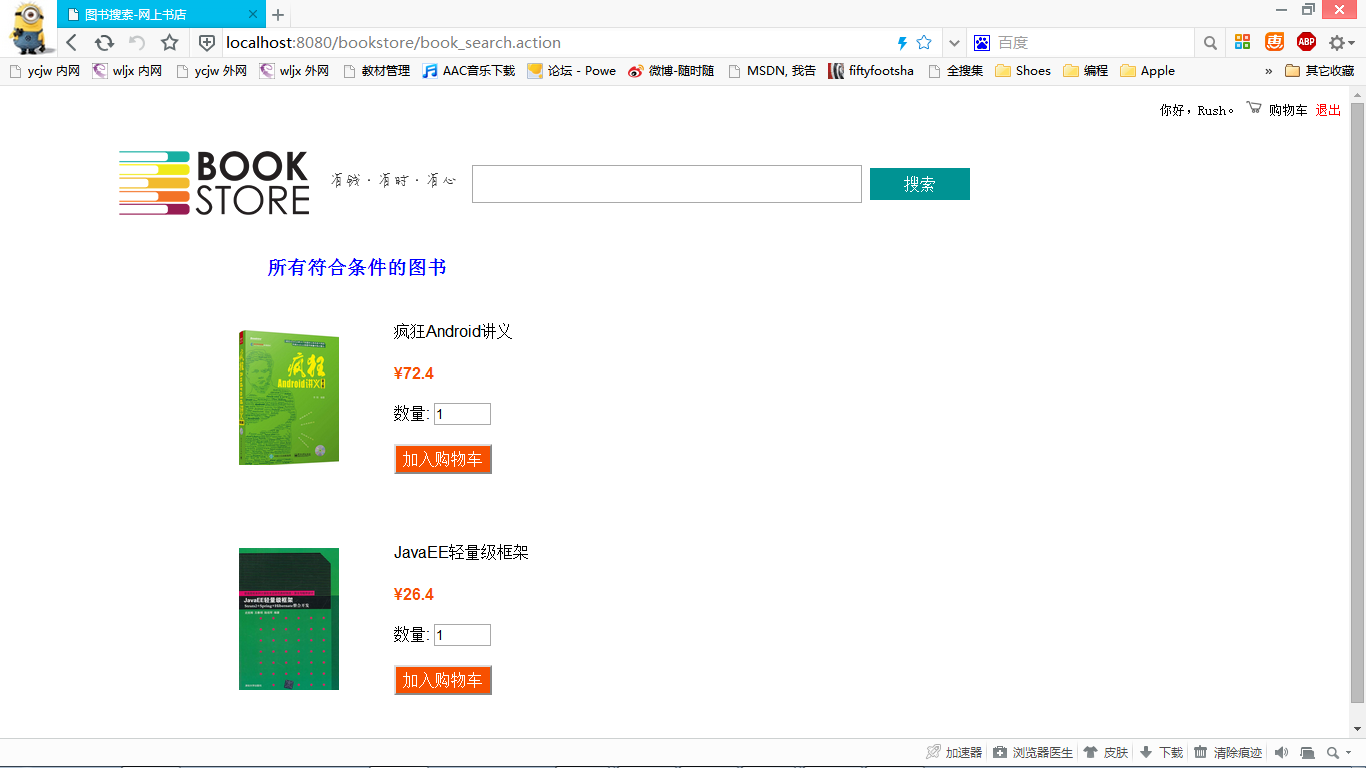


图5-25 图书模糊搜索

13、购买图书

说明：选好想买的图书后，点击加入购物车，即可将此图书加入至购物车。如图5-26所示。

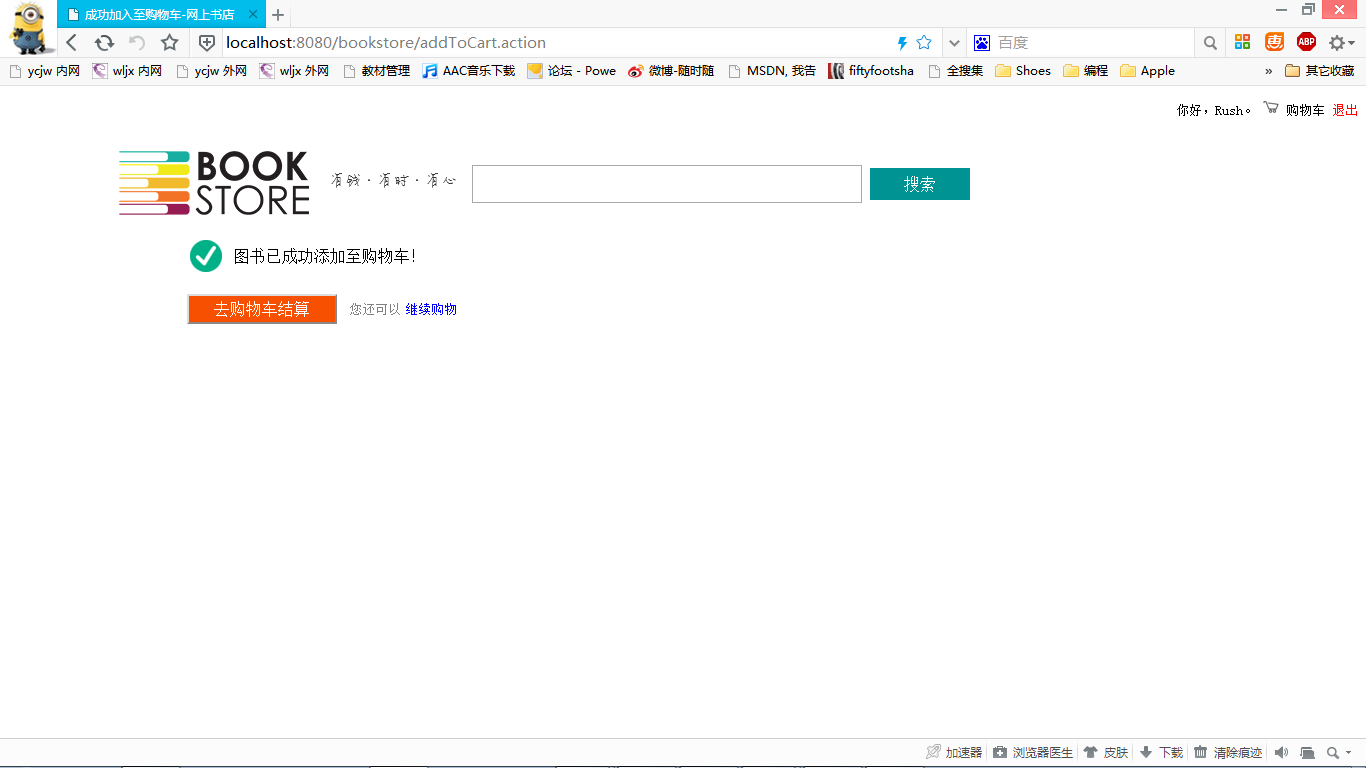


图5-26 加入购物车

说明：此时可以进行结算或者继续购物，如果点击“继续购物”的话将会跳转至图书搜索界面，如图5-24所示，如果点击“去购物车结算”的话将会跳转至订单结算页面，如图5-27所示。



图5-27 订单结算

说明：此时点击“确认结算”按钮后就会进行结算成功，如图5-28所示。



图5-28 结算成功

14、查看订单

说明：此时点击“查看订单”按钮，可以对刚刚购买成功的订单信息进行查看，如图5-29所示。



图5-29 查看订单

# 6、总结

## 6.1 存在的问题

目前系统存在的问题我认为主要有五点：

一是如果用户没有登录的情况下，点击结算，会跳转到用户登录界面，这里是的逻辑是没有问题的，但是在用户进行登录成功后，没有想到办法跳转到刚刚结算的页面，默认是跳转到欢迎页面的，要点击购物车后结算才可以继续结算。这算是一个小缺陷吧，我想过，加判断的话首先要获取到是从什么页面跳转到的登录界面，如果从欢迎界面跳转到的登录界面，登录成功后要返回到欢迎页面；如果是从购买页面跳转到登录页面时，那么逻辑上登陆成功后也要返回到购买页面，其他的页面也是如此，但目前没有找到方法能实现这个操作；

二是没有编写个人中心这个页面，查询订单只能让刚刚购买的用户查询后保存下订单的信息才可以，不太人性化，可以进行改进；

三是用户没有权限修改自己的密码，只能向管理员发出请求，然后管理员直接进行密码修改；

四是修改购物车中的图书数量只能通过点击在购物车页面的“更新图书数量”按钮实现，不够人性化，但是又没想到解决办法；

五是数据库没有进行加密，直接在控制台搜索就可以知道用户的账号和密码，非常不安全，即使作为管理员，用户的密码对管理员来说也应该是不可见的，所以这点有待改进。

目前就想到这些，如果老师发现，可以进行补充下，谢谢。

## 6.2 几点建议

我暂时想到有如下几点建议，它们是：

1、管理员的账号有点少，感觉不应该只有一个管理员，因为如果网上书店的图书数量增加后，有各种各样的需求，所以应该增加许多管理员，另外应该设置不同权限的管理员，有专门管理图书的管理员，有专门管理用户的管理员，还有专门管理订单的管理员，这样将职能划分成好几个部分，便于书店对信息的管理；

2、应该在购物车里面图书的数量框旁边添加“+”和“-”按钮，直接对数量进行操作，比较便捷，也更加人性化。

3、购买过图书可以给那本图书添加评价和打星，写一写书评，以供之后浏览此图书的用户进行参考，来决定到底是否要购买此书。

4、可以根据快递公司有一些业务合作，这样可以在个人中心对快递的位置进行查询。

5、用户界面还是可以继续改进，变得更加活泼，讨好用户的眼睛。