## 第一阶段 注册页面的验证

使用jQuery对注册页面regist.html中的用户名、密码、确认密码、邮箱等信息进行验证

## 第二阶段 完成登录和注册功能

### 1.软件结构：

生活案例：

**顾客 ---> 服务员 ---> 后厨 ---> 采购员 ---> 菜市场**

第1个问题：像后厨、采购员可以直接接待顾客吗？

第2个问题：假若后厨或采购员离职了对其他员工有影响吗？

服务员相当于表现层

后厨相当于业务逻辑层

采购员相当于持久化层

菜市场相当于数据库

软件结构分成三层：

数据库

持久化层

业务逻辑层

表现层

详细描述：

表现层（web层、表示层）

作用：用户可以直接访问到的内容

体现：html页面 Servlet

业务逻辑层（service层）

作用：进行业务的一个处理

体现：注册、登录、添加图书、删除图书、结账等

持久化层（dao层）

作用：负责数据的增删改查等基本的操作，主要与数据库打交道

**为什么要把软件分成多层结构？**

一方面是为了解耦，另一方面是为了复用

通常业务逻辑层和持久化层发生变化的频率比较高，所以我们采用以面向接口的方式来编程。

### 2.书城项目结构：



### 3.开发步骤：

1. 创建动态web工程
2. 创建数据库 bookstore\_web
3. 创建表 bs\_users ( id,username,password,email)
4. User类
5. Integer id,
6. String username,
7. String password,
8. String email
9. 导入第三方jar包
10. mysql-connector-java-5.1.7-bin.jar
11. commons-dbutils-1.3.jar
12. c3p0-0.9.1.2.jar
13. 导入c3p0.xml配置文档
14. JDBCUtil工具类来获取和关闭连接
15. Connection getConnetion();
16. Void releaseConnection(Connection conn);
17. 创建BaseDao<T>
    1. Int update();
    2. T getBean()
    3. List<T> getBeanList()
18. 创建userDao接口及实现类
    1. Int saveUser(User user)
    2. User getUserByusernameAndPassword(User user)
19. 创建userService接口及实现类
    1. Boolean regist(User user)
    2. User login(User user)
20. 创建RegistServlet 和LoginServlet
    1. 获取请求参数
    2. 封装请求参数
    3. 调用service层方法
    4. 根据返回结果返回响应
21. 将项目阶段1中的静态web资源复制过来
    1. 修改页面中的路径为绝对路径，可以使用base标签
    2. 修改form标签的action属性

## 第三阶段 项目优化

### 将html转成jsp

方案一：

首先复制jsp的page指令粘贴到html文件的首先并保存

再将html文件的后缀.html改成.jsp

方案二：

先创建与html文件同名的jsp文件

再将html文件中的内容全部拷贝粘贴覆盖除jsp文件中指令page之外的所有代码

抽取页面中代码重复部分，通过静态include指令将内容引入到当前页面中

### 注册页面与登录页面的错误信息回显

### 表单信息的回显

### 项目优化

* 1. 问题1：目前是一个功能对应着一个servlet。比如说，一个登录功能对应一个LoginServlet，一个注册功能对应一个RegistServlet，那我们后面还有添加图书、修改图书、添加购物车、提交订单等功能，假如也是一个功能对应一个servlet类，那么会创建非常非常多的servlet，这是不是问题？
  2. 问题2：在servlet中的通常都会先获取请求信息，然后再封装成对象，这个过程简单且繁琐，降低工作效率，若遇到需要转int、double问题更麻烦？

问题1\_\_解决方案：

* 我们希望最好相关的功能放在一个servlet中，比如用户相关的注册和登录功能都放在UserServlet中，这样我们可以精确定位，代码维护起来就方便多了。
* 具体实现是通过定义一个请求参数来标识要处理的操作，比如请求参数中传一个method=regist代表是注册操作，假如传一个method=login则代表是登录操作，然后根据获取到的标识调用不同的方法。但是又出现新问题，一个功能需要写一个else if ，后面有几个功能就需要对应几个else if 。Servlet多的问题虽然解决了，但是else if 又将问题重演了，还是没有彻底解决问题？
* 我们可以根据方法名利用反射技术来实现方法的调用。到这里servlet非常多的问题已经解决。
* UserServlet相关的功能目前基本放在一起，但是后面还会有图书相关的、购物车相关的及订单相关的，同样需要这样实现，为了实现代码复用，我们可以将反射部分提取来写在一个BaseServlet中，让其它servlet继承，以达到复用的效果。

问题2\_\_解决方案：

* 我们希望有那么一种技术可以帮我们实现这些常规繁琐的信息封装。我们可以使用BeanUtil技术写一个工具类
* 什么是JavaBean？
  + 属性私有化，通过getter和setter方法对属性进行读取和设置
  + 有一个无参构造器
  + JavaBean的属性指的是getter方法中属性名，属性名指的getter或setter方法将get或set去掉，然后首字母小写。getName()属性名叫name，如果有一个getAge()属性名叫age
* 可以使用BeanUtil的populate方法帮我们将请求参数封装到对象中
  + 需要导入以下两个jar包：

commons-beanutils-1.8.0.jar

commons-logging-1.1.1.jar

* 下一个问题是需要将请求参数组装成Map? 而request对象就给我们提供了getParameterMap()获取请求参数返回一个Map的方法

## 第四阶段 使用EL表达式代替所有页面中的JSP表达式

* Regist.jsp
* Login.jsp
* Base.jsp

## 第五阶段 图书的增删改查

### 后台图书的增删改查

1. 创建图书数据表bs\_books和Book类



* + 1. Integer id
    2. String title 书名
    3. String author 作者
    4. Double price 价格
    5. Integer sale 销量
    6. Int stock 库存
    7. String img\_path = “static/img/default.jpg” 图书封面

1. 创建图书BookDao接口和实现类
   * 1. Int saveBook(Book book) 保存图书信息
     2. Int deleteBook(String id) 根据id删除图书信息
     3. Int updateBook (Book book) 修改图书信息
     4. Book getBook(String id) 根据id获取一本图书
     5. List<Book> getABook() 获取全部图书
2. 创建图书BookService接口和实现类
   * 1. Int saveBook(Book book) 保存图书信息
     2. Int deleteBook(String id) 根据id删除图书信息
     3. Int updateBook (Book book) 修改图书信息
     4. Book getBook(String id) 根据id获取一本图书
     5. List<Book> getABook() 获取全部图书
3. 后台创建BookManagerServlet
   * 1. bookList() 获取所有图书信息
        1. 需要通过servlet与后台进行交互，来访问数据库获取所有图书信息
        2. 将图书信息放到域中
        3. 通过转发的形式访问页面
        4. 使用jstl来展示数据
     2. addBook() 添加一本图书信息
     3. delBook() 根据图书id删除一本图书信息
     4. updateBook() 修改一本图书信息
4. 问题：

目前我们发现在bookDao层使用了

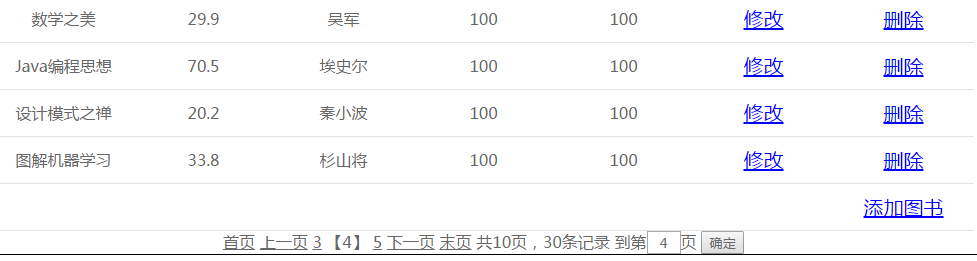
“select id,title,author,price,sales,stock,img\_path from bs\_books”这样一条sql语句，将所有的图书列表都显示出来，假若数据库中的图书有很多很多的话，会造成一页显示不过来，会出现很窄的一条滚动条，造成一是用户体验很差，二是降低服务器性能。所在像这样的语句，工作中坚决不能写。

解决方案：

可以采取分批分次获取数据展示数据，也就是给图书列表做一个分页操作。比如说每次我只显示4条图书信息，这样我们需要修改sql语句为：

SELECT \* FROM bs\_books LIMIT 开始的索引，每而显示的条数;

由于我们为图书列表添加了分页功能，所以现在使用list集合已经不能满足我们的需求，我们需要重新设计一个类来封装分页信息。这个新类，我们设计成Page<T>类：



当前页

List<Book>

显示条数

开始索引

共几页

共几条

* Page<T>:

List<T> list; 第页显示的数据，需要通过数据库查询获取

Int totalRecord; 需要查询数据库

Int totalPage; 需要计算得出

Int index; 需要计算得出

Int pageSize; 需要用户指定

Int pageNo; 需要用户指定

* 在BookDao中添加一个分页方法

Page<Book> findBook(Page page)

* 在BookService中添加分页方法

Page<Book> findBook(String pageSize,String pageNo);

* 在BookManagerServlet中添加分页方法

findBook();

* 新问题1：页面在分页显示时，如果总共有9页，通过点击下一页会访问到第10页，显然这里没有10页以后的了，所以要对当前页进行处理。如果大于总页数的请求，那么就返回最大页数；如果小于第1页的请求，那么都返回第1页。
* 新问题2：分页页码过多，如下：



解决方案：采用百度方式分页显示

【1】 2 3 4 5

1 【2】 3 4 5

1 2【3】 4 5

2 3【4】 5 6

3 4【5】 6 7

4 5【6】 7 8

可以根据不同情况，在域里设置变量的初始值

第1种情况：当总页数小于5时：begin=1 ，end=总页数

第2种情况：当前页小于3时：begin=1,end=5

第3种情况：当前页大于3时：begin=当前页-2，end=当前页+2

当end变量值大于总页数时，begin=总页数-4，end=总页数

* 问题3：修改之前的图书的删除与修改功能，哪来回哪去

解决方案：

删除功能，在servlet中通过重定向到请求头referer可以解决

修改功能，需要先在edit\_book.jsp中设置请求头referer参数，再重定向到referer

添加功能，需要修改一下Dao层的sql语句，添加一个降序排列

### 前台图书的页面显示

* 前台图书的显示功能不需要重写Dao层与service层，直接调用已有的方法就可以
* 我们只需要重定义一个BookClientServlet

1. findBook() 分页查询图书方法
2. findBookByPrice() 根据价格查询图书

步骤：

1. 在BookDao定义方法及实现：

Page<Book> findBookByPrice(Page<Book> page,String min,String max)

1. 在BookService定义方法及实现：

Page<Book> findBookByPrice(String pageNo,String pageSize,String min,String max)

1. 在BookClientServlet中定义方法：

findBookByPrice() 根据价格查询图书

## 第六阶段 登录、登出、验证码、购物车

### 登录

可以将用户信息放到session中，通过检查session中是否有用户信息来判断用户是否登录

### 登出

可以直接使用session失效的方式实现登出功能。

步骤：

1）页面中修改“注销”连接地址为：

<a href=*"Client/UserServlet?method=logout"*>注销</a>

2）需要在userServlet中定义一个logout方法，直接调用session.invalidate()方法，然后重定向到index.jsp

### 验证码

* 验证码和我们学和token的原理是一样的，只不过我们的token是不需要手动输入验证信息的，而验证码需要用户输入信息
* 验证码的主要作用是为了避免恶意注册。
* 在注册页面输入验证码，在servlet中检查验证码是否正确，如果正确则处理请求，否则不处理请求
* 验证码的使用流程：

由系统生成随机字符串，然后将字符串保存到session中

验证不正确

验证正确

将图片放入注册页面

用户根据图片中的信息输入验证码

将字符串转成图片

请求提交到UserServlet中的regist方法

返回注册页面

直接处理请求

在regist中，检查验证码是否正确，并移除session中的字符串

可以使用第三方工具帮我们完成这些功能：

使用步骤：

1. 需要引入kaptcha-2.3.2.jar
2. 需要在web.xml中配置com.google.code.kaptcha.servlet.KaptchaServlet类

<servlet>

<servlet-name>KaptchaServlet</servlet-name>

<servlet-class>com.google.code.kaptcha.servlet.KaptchaServlet</servlet-class>

<init-param>

<param-name>kaptcha.session.key</param-name>

<param-value>code</param-value>

</init-param>

</servlet>

<servlet-mapping>

<servlet-name>KaptchaServlet</servlet-name>

<url-pattern>/code.jpg</url-pattern>

</servlet-mapping>

1. 可以在servlet标签中更改初始化参数的值 。

* 如：kaptcha.session.key 默认 KAPTCHA\_SESSION\_KEY

### 购物车

* 购物车的作用就是用来暂时存放图书的
* 实现购物车有三种方式：

1. 基于cookie，将图书信息保存在cookie中发送给浏览器
2. 基于session，将图书信息保存在session中，一但关闭浏览器信息将丢失
3. 保存在数据库中，可以实现不同浏览器及设备的信息共享



集合

CartItem

总金额

总数量

* CartItem类：

Book book; 图书信息

Int totalCount; 单品种图书的总数量

Double totalAmount; 单品种图书的总金额

* 由于我们的Cart类是基于session的，所以不需要定义dao层与service层。既然这样，我们可以直接将方法定义在cart类中
* Cart类：
* 属性：

Map<String,CartItem> map = new HashMap<>();用来保存cartItem

Int totalCount; 购物车中图书的总数量

Double totalAmount ; 购物车中图书的总金额

* 方法：

Void addBook2Cart(Book book); 添加图书到购物车

Void delCartItem(String bookId); 删除单品种购物项

Void updateCount(String bookId,int number); 修改购物车中某本图书的数量

List<CartItem> getCartItemList(); 返回所有的购物项

Void clear() 清除购物车功能

* 创建一个cartServlet
* addBook2Cart 添加图书到购物车
* Clear() 清空购物车
* delCartItem() 删除单品种购物项
* UpdateCount() 修改单品种购物项的数量

## 第七阶段 结账

结账就是将购物车中的信息保存到数据库中，并生成一张订单信息

结账的流程：



在OrderService层需要创建一个createOrder()方法将购物车中的商品信息保存到数据库中，并返回订单号

1. 将商品信息保存到数据库
2. 修改图书的销量与库存
3. 清空购物车
4. 返回订单号

点击结账请求会发送到一个OrderClientServlet中checkout()方法

1、检查用户是否登录

2、如果登录直接调用OrderService层处理订单

3、如果没有去登录页面

OrderItemDao

OrderDao

BookDao

### 创建用来封装订单信息的类和表

Order类：

String id; 订单号

Date createDate; 生成日期

Int State; 订单的状态 （0代表未发货 1代表已发货 2代表交易完成）

Double totalAmount; 订单金额

Int totalCount; 订单数量

Int userId; 用户Id

用户与订单的关系:一对多的关系

OrderItem类：

Integer id 订单项编号

String title 书名

String author 作者

Double price 价格

String img\_path = “static/img/default.jpg” 图书封面

Int totalCount; 购买数量

Double totalAmount 金额

String orderId 隶属哪个订单

订单与订单项的关系：一对多的关系

订单号：

1. 具有唯一性
2. 尽量少的透漏商业机密
3. 订单号用于售后，要求易识别

订单号规则：时间戳+””+用户ID

### 创建OrderDao及实现类

Int saveOrder(Order order) 保存订单信息

Int updateState(String orderId,int state) 修改指定订单的状态

List<Order> getOrderListByUserId(String userid) 根据用户id获取订单信息，提供给用户使用

List<Order> getAllOrderList() 获取全部订单，提供给管理员使用

### 创建OrderItemDao及实现类

Int saveOrderItem(OrderItem orderItem）保存订单项信息

List<OrderItem> getOrderItemListByOrderId(String orderId) 根据订单号获取全部订单项信息

### 生成订单功能

* 创建OrderService接口及实现类
  + String createOrder(Cart cart ,User user) 生成订单
* 创建OrderClientServlet
  + Checkout() 处理结账请求
* 修改cart.jsp页面链接及checkout.jsp页面

### 订单查询

* 在OrderService层
  + getOrderListByUserId(int id) 根据用户id获取订单信息，提供给用户
  + getAllOrderList() 获取所有的订单信息，提供给管理员调用
* 创建OrderManagerServlet
  + orderList() 查看所有订单信息
* 在OrderClientServlet中添加一个方法
  + orderList() 根据用户id查看订单

### 发货和确认收货

* 发货就是将订单的状态修改为1
* 确认收货就是将订单的状态修改为2
* 在OrderService接口中添加两个方法
  + sendBook(String orderId); 将订单状态修改为1，管理员调用
  + takeBook(String orderId); 将订单状态修改为2，用户调用
* 在OrderManagerServlet
  + sendBook()管理员发货
* 在OrderClientServlet
  + takeBook() 用户确认收货

### 查看订单详情

查看订单详情实际就是查看当前订单里面的所有订单项。

* 在OrderService中添加方法
  + List<OrderItem> getOrderInfoByOrderId(String OrderId);根据订单号获取订单项
* 在OrderClientServlet中添加方法
  + orderInfo()

### 问题

在orderService的createOrder()方法中我们使用for循环遍历的方式去保存orderItem和修改book的库存和销量，也就是说如果我们有10个orderItem的话，那么整个过程需要对数据库操作20次，存在着资源浪费的问题。我们希望分批进行操作，来提高效率。

* BaseDao中添加一个批量方法
  + Batch(String sql ,Object[][] params)
* orderItemDao中添加一个批量方法
  + BatchSave(Object[][] params)
* BookDao中添加方法
  + batchUpdateSaleAndStock(Object[][] params)
* 对orderService中的createOrder进行优化

## 第八阶段 用户统一登录验证与结账事务处理

### 用户统一登录验证

我们希望访问OrderClientServlet的所有请求都需要用户登录之后才可以操作，而目前用户只要记住路径，在没有登录情况下也是可以访问页面的。所以我们需要给OrderClientServlet处理的每一个请求添加用户登录验证功能。在OrderClientServlet中有多个方法，第一种验证方案就是给每一个方法都加入一段代码来验证用户是否登录，但显然这种方式有代码冗余问题，比较繁琐，通常不采用。第二种方案：我们可以利用Filter来实现，需要定义一个LoginFilter类来统一管理用户是否登录。使用路径匹配：/Client/OrderClientServlet

### 2.结账事务处理

* 数据库事务：是指数据库管理系统执行过程中的一个逻辑单元，由一个数据库操作序列构成。
* 我们这里的事务是针对结账进行的订单插入、订单项插入、修改图书销量和图书库存的一系列操作。我们希望要么都成功，要么都失败。
* 问题：在OrderDaoImpl的saveOrder方法中的sql语句写错了，会导致订单无法正常插入到数据库中，但我们发现图书的库存与销量的值却进行了修改。执行顺序：

浏览器

OrderClientServlet.checkout()

OrderService.createOrder()

BookDao

OrderItemDao

OrderDao

回顾一下常规事务：

//获取数据库连接

Connection conn = **null**;

**try** {

//设置事务手动提交

conn.setAutoCommit(**false**);

//调用Dao操作数据库

//...

//没有异常，进行事务提交

conn.commit();

} **catch** (SQLException e) {

//出现异常，回滚事务

**try** {

conn.rollback();

} **catch** (SQLException e1) {

e1.printStackTrace();

}

e.printStackTrace();

}**finally**{

//关系数据库连接

**try** {

conn.close();

} **catch** (SQLException e) {

e.printStackTrace();

}

}

思路一：我们需要将事务加在哪里？加在Dao层不合适，因为有可能service层会调用多个Dao类，没办法控制;放在service层与servlet层是同一个道理。那怎么办呢？我们可以添加一个Filter层，定义一个TransactionFilter类来管理事务

思路二：事务操作的关键是同一个业务中使用的connection，必须是同一个。也就是TransactionFilter中获取的数据库连接与BaseDao中获取的数据库连接必须是同一个。如果想达到这个目的，我们可以将数据库连接从Filter层传到servlet，再传到service,再传给Dao,通过外部传的方式来获取数据库连接，但这种方式想想可行吗？不采用。那怎么办？我们可以将数据库连接改成单例模式

* 出现问题：java.sql.SQLException: You can't operate on a closed Connection!
* 原因我们在BaseDao中关闭了数据库连接

思路三：修改后发现事务没有启作用，原因是我们在BaseDao中捕获到异常并自行处理了。但是这个问题不应该在这里处理，我们希望它能抛向Filter层进行统一处理。当我们再次运行时发现问题没有处理完全，在BaseServlet的dopost方法又进行了捕获处理，所以控制台报出java.lang.reflect.InvocationTargetException异常。

* 思路四：当我们将TransactionFilter中的关闭连接打开后，我们发现控制台又出现java.sql.SQLException: You can't operate on a closed Connection!异常。原因是浏览器向服务器访问web资源，而web资源中有可能还包含了其它web资源，比如请求一个index.jsp，会发送多个请求，这样出现上面异常的概率就非常大。怎么办呢？我们可以采用一线程对应一个数据库连接，也就是定义一个Map<Thread,Connection>,这样就可以改善线程间互相影响的情况.除可以使用concurrentHashMap外，java还提供另一个类ThreadLocal也可以实现一个线程对应一个连接

OrderClientServlet.checkout()

TransactionFilter

浏览器

BaseDao

OrderItemDao

BookDao

OrderDao

OrderService.createOrder()

## 第九阶段 将ajax应用到项目中

### 首页的加入购物车功能可以引入AJAX

步骤：

* 给button按钮加入一个class=”add2cart”和name=book.id
* 给button按钮绑定单击事件，通过jQuery提供的ajax发送请求
* 修改首页的信息显示
* 请求发送到CartServlet的addBook2Cart方法中，并对该方法进行修改，返回json字符串

### 购物车页面可以使用AJAX

使用ajax修改购物车中图书的数量，步骤：

* 在cart.jsp页面中给图书数量文本框绑定change事件，发送ajax请求到cartServlet的updateCount方法中
* 对updateCount方法进行修改，并响应json字符串
* 在car.jsp中修改数量文本框的兄弟td,及div标签中的两个span标签

### 注册页面可以使用AJAX来检查用户名是否存在

实现步骤：

* 在userDao中创建一个新方法：User getUserByUsername(String username);
* userService中定义一个新方法: boolean checkUsername(String username);
* 在UserServlet中定义一个新方法：checkUsername()
* 在regist.jsp页面中对用户名文本框绑定change事件，发送ajax请求

## 第十阶段 修改图书封面

说明：图片是保存到服务器上的，并没有保存到数据库中，而将图片在服务器上路径保存到数据库中

* 目前需要在book\_edit.jsp页面中添加上传图书封面图片的文件表单项
* 请求会发送到BookManagerServlet?method=editBook方法中，我们需要修改此方法
  + 导入commons-fileupload-1.3.1.jar 和 commons-io-2.4.jar