**书城项目说明**

## 第一阶段 完成注册页面的验证功能

通过jQuery对注册页面regist.html中的用户名、密码、确认密码、邮箱等进行验证

## 第二阶段 完成登录与注册功能

### 软件结构

生活案例：

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

第1个问题：采购员或厨师可不可以将做好的“排骨炖蘑菇”端给顾客？

第2个问题：假如厨师或采购员离职了，对其它职位的员工影响它们本职工作吗？

服务员相当于表现层

厨师相当于业务逻辑层

采购员相当于数据持久化层

菜市场相当于数据库

软件的三层架构：

数据库

表现层

持久化层

业务逻辑层

软件详解：

表现层（html servlet)

作用：与用户进行交互

业务逻辑层（Service层）

作用：处理业务，比如登录、注册、购物、结账...

持久化层（Dao层）

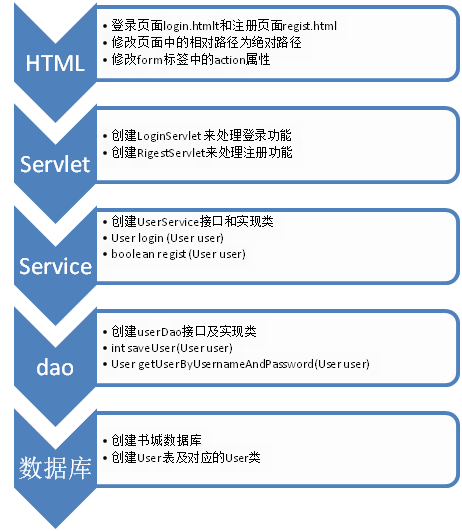
作用：与数据库打交道，主要是操作数据库中的数据

为什么要将软件分出这么多层结构？

第一是解耦，第二是复用

由于业务逻辑层与持久化层非常容易发生变化，所以通常这两层我们都是以面向接口的方式来实现 。这样业务逻辑发生变化时我们可以直接替换，而不会影响其它层次。

### 2.项目结构



### 开发步骤

1. 创建一个动态web工程
2. 创建书城项目的数据库 bookstore\_0310
   1. 创建表 bs\_users

Create table bs\_users(

Id varchar(11) primary key auto\_increment,

Username varchar(50) unique not null,

Password varchar(50) not null,

Email varchar(100)

)

1. 创建User类：Id，用户名username，密码password，邮箱email

Integer id

String username 用户名

String password 密码

String email 邮箱

1. 导入第三方jar包
   1. mysql-connector-java-5.1.7-bin.jar mysql驱动
   2. c3p0-0.9.1.2.jar 使用c3p0来管理数据库连接
   3. commons-dbutils-1.3.jar 使用dbutils来操作数据库
2. 导入c3p0.xml配置文件
3. 创建JDBCUtil工具类
   1. Connection getConnection()
   2. Void releaseConnection(Connection conn)
4. 创建BaseDao<T>来统一处理数据库操作，让其它Dao直接调用BaseDao中的方法
   1. Int update() 修改数据
   2. T getBean() 获取一个对象
   3. List<T> getBeanList() 获取全部对象
5. 创建UserDao接口及实现类
   1. Int saveUser(User user) 用来保存一个用户信息
   2. User getUserByUsernameAndPassword(User user) 根据用户名和密码来查找一个用户信息
6. 对UserDao做单元测试
7. 创建UserService接口和实现类
   1. User login(User user) 根据用户输入的用户名和密码获取用户信息
   2. Boolean regist(User user) 注册用户信息，如果返回0说明此用户已经注册过
8. 对userService做单元测试
9. 创建LoginServlet与RegistServlet来分别接收登录与注册请求
10. 修改login.html 和regist.html 页面
    1. 将相对路径修改成绝对路径
    2. 修改form标签中的action属性值

## 第三阶段 项目优化

### 将所有的html修改为jsp

方案1：创建新的jsp文件，然后将html代码拷贝到jsp文件中

方案2：先在html文档中添加page指令，再修改html文档的扩展名为jsp

### 抽取页面中公共html代码

* 将页面中的相对路径修改为绝对路径
* 抽取页面中的相同html代码
  + 将样式标签、base标签、script标签放到base.jsp中
  + 抽取导航信息，分别放在user.jsp与manager.jsp中

### 登录和注册页面的错误消息显示

### 表单的回显

### 项目优化

* 问题1：目前是一个功能对应着一个servlet，比如一个登录功能对应一个LoginServlet，一个注册功能对应一个registServlet，那么接下来我们还有添加图书、删除图书、修改图书、添加购物车、清空购物车、结账等等功能，是不是这要一个功能对应一个Servlet，这样会有非常多的Servlet，我们会淹没在Servlet的海洋中，这是不是问题？
* 问题2：每一个Servlet一般情况会获取请求参数、组织参数，像这样的重复且降低效率的操作可不可以改进呢？

解决方案:

问题1方案：

一个功能对应一个servlet，会造成出现过多的servlet，我们可以将相关功能组织在一个servlet中，比如像注册、登录等可以放在一个UserServlet中，这样我们更好的管理和维护。

解决办法，可以在servlet中定义不同的方法，根据传入参数的不同调用不同的方法。比如请求参数中传一个method=login，在Servlet中获取method参数，根据获取值的不同调用不同的方法。

* 创建UserServlet，并在其中创建对应的login方法与regist方法，把与用户相关的操作定义成一个方法并组织在一个servlet中。要求方法的声明结构与doPost或doGet相同。
* 要求前端需要传递一个method请求参数，来指明要调用哪一个方法
* 在servlet中获取method请求参数，根据请求参数值通过if...else if...else if来判断调用哪个方法

使用改进方案又带来新问题，过多的servlet问题是解决了，但又出现多过的else if 问题，这怎么办呢？我们可以使用反射根据方法名来调用方法的执行。

问题2方案：servlet中通常都会先获取请求参数，再组织参数这样的常规操作，我们可以使用beanUtil工具来实现。

* 什么是javaBean?JavaBean需要具备以下几点：

1. 所有属性都私有化，使用private 修饰
2. 每一个属性都提供对应的getter方法与sertter方法
3. 提供一个无参构造器

* 使用beanUtil需要导包：

commons-dbutils-1.3.jar

commons-logging-1.1.1.jar

* 通过测试我们知道，BeanUtils使用javaBean的getter方法名与setter方法名去掉get和set后首字母小写来获取/设置属性值的
* 改造servlet获取的请求参数方式。

通过httpServletReqeust提供的getParameterMap()可以获取请求参数的map对象，结合BeanUtils提供的populate()方法，可以很方便达到我们想要的效果

## 第四阶段 使用EL表达式替换jsp表达式

* Base.jsp
* Login.jsp
* regist.jsp

## 第五阶段 图书的增删改查

### 后台图书的增删改



1. 创建Book数据库表bs\_books

Create table bs\_books(

Id int(11) primary key auto\_increment,

Title varchar(50) ,

Author varchar(50) ,

Price double(11,2),

Sale int(10),

Stock int(10),

Img\_path varchar(200)

)

1. 创建Book类

Integer id;

String title;书名

String author;作者

Double price;价格

Int sale;销售

Int stock;库存

String imgPath;封面

1. 创建BookDao及实现类

Int saveBook(Book book); 保存图书信息

Int updateBook(Book book);修改图书信息

Int deleteBook(String bookId); 根据图书id来删除一本图书

Book getBook(String bookId);根据id来查找一本图书

List<Book> getAllBook(); 获取全部图书

1. 创建BookService及实现类

Int saveBook(Book book); 保存图书信息

Int updateBook(Book book);修改图书信息

Int deleteBook(String bookId); 根据图书id来删除一本图书

Book getBook(String bookId);根据id来查找一本图书

List<Book> getAllBook(); 获取全部图书

1. 创建BookManagerServlet
2. addBook() 处理添加图书请求

* 修改book\_edit.jsp页面的form标签的action与method属性
* 修改book\_edit.jsp页面中表单项的name属性值与book的属性值一致

1. delBook() 处理删除图书请求

* 在book\_manager.jsp页面添加js确认删除代码

1. updateBook() 修改图书请求
2. listBook() 显示所有图书请求

* 导入jstl包：taglibs-standard-impl-1.2.1.jar和taglibs-standard-spec-1.2.1.jar
* 在book\_manager.jsp中引入c标签库

目前存在的问题：

当数据库中的图书信息过多时，后台图书的显示会带来加载延时降低性能，甚至造成服务器崩溃。造成这个问题最直接的原因是我们在BookDao层使用了如下sql语句：

select id,title,author,price,sale,stock,img\_path imgPath from bs\_books

它会将数据库中的所有图书数据查询出来。但是实际上不需要这么做，一是用户根本看不过来，二是体验非常差，我们需要修改这条sql语句为：

SELECT id,title,author,price,sale,stock,img\_path imgPath FROM bs\_books LIMIT 索引,条数

由于加入了分页功能，仅传递list集合是不能满足需要的，所以我们这里需要设计一个新的类来封装分页中的信息，即Page<T>类:



条数

索引

总条数

总页数

当前页

List<Book>

Page<T>类：

List<T> list;需要展示的数据，从数据库中获取

Int totalCount; 总记录数，从数据库中获取

Int pageNo; 当前页，由用户指定

Int pageSize; 显示条数，由用户指定

Int totalPage; 总页数，根据总记录数与显示条数计算出来的

Int index; 索引，根据当前页与显示条数计算出来

分页操作步骤：

1. 在BookDao中添加分页方法：

Page<Book> findBook(Page<Book> page)

1. 在bookService中添加分页方法：

Page<Book> findBook(String pageSizeStr,String pageNoStr)

1. 在BookManagerServlet中添加分页方法：

findBook()

1. 修改页面中的访问路径

新问题：分页页码过多？如下：



解决方案：可以采用百度分页方案













解析百度分页思路，我们可以在域中设置最小与最大两个变量，通过c:foreach来遍历实现分页：

1. 总页数小于10时，将全部显示出来，最小值设置1，最大值设置总页数
2. 总页数大于10，当前页小于6时，最小值设置1，最大值设置为10;

在当前面大于6时，最小值设置当前页-5，最大值设置为当前页+4

此种情况又出现最大值大于总页面的情况，此时最小值为总页数-10，最大值为总页数

在我们这里每次显示5页，根据不同情况在域里设置变量的初始值

第一种情况：当总页数小于5时，begin设置为1，end设置为总页数

第二种情况：总页面大于5且当前页小于3时，begin设置为1，end设置为5

第三种情况：总页数大于5且当前页大于3时，begin设置为当前页-2，end设置为当前页+2

当end大于总页数时，begin设置为总页数-4 ，end设置为总页数

提取分页功能，应用到其它页面

### 前台图书显示

* 前台图书的显示不需要重新写service与dao层，而是调用已有的
* 我们只需要创建一个BookClientServlet：

1. 创建方法findBook()

用户访问页面需要通过BookClientServlet转发到页面。在index.jsp中转发到BookClientServlet的findBook()方法中，查找需要的数据设置进域中，再到页面中显示出来

1. 显示带有价格查询条件的图书信息：
2. 在BookDao创建Page<Book> findBookByPrice(Page<Book> page,int min,int max)
3. 在BookService创建

Page<Book> findBookByPrice(String pageNoStr,String pageSizeStr,String min,String max)

1. 在BookClientServlet中创建方法：

findBookByPrice()

1. 修改页面home.jsp

* 查询条件回显
* 解决分页查询问题

## 第六阶段 登录登出、验证码、购物车

### 登录登出

判断用户是否登录，可根据域中是否有用户信息来决定，有代表登录，否则没有。

登出可直接将session失效方式来实现

### 验证码

项目中的验证码与我们学的token令牌原理一样

验证码流程：

发送请求到servlet中

用户根据图片中的信息输入

在页面中引入图片

将token字符串转成图片

由服务器生成token字符串,并将token字符串保存到session中

false

true

表单重复

正常处理请求

在Servlet中，获取request中的token字符串和session域中的字符串进行比较，并移除session中的token字符串

可以使用第三方工具：kaptcha-2.3.2.jar

步骤：

1. 导入jar包kaptcha-2.3.2.jar
2. 在web.xml中配置KaptchaServlet
3. 修改regist.jsp页面
4. 修改UserServlet的regist()方法

### 购物车

购物车用来暂时存放购买的图书

设计购物车的方案：

1. 基于cookie，将购买的图书信息保存在客户端
2. 基于session，将购买的图书信息保存在服务器中的session里
3. 基于数据库的，将购买的图书信息保存在数据库中



CartItem

总金额

总量

List<CartItem>

Cart类：

Map<String,CartItem> map; 用来保存图书信息

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：

List<CartItem> getList(); 获取所有的cartItem项

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

Void deleteCartItem(String bookId); 根据图书id删除指定购物项

Void changeCount(String bookid,int count); 修改指定id图书的数量

Void clear(); 清空购物车

CartItem类：

Book book; 图书信息

Int count; 购买单品种图书数量;

Double amount; 购买单品种图书的金额

开发步骤：

1. 创建CartItem类与Cart类
2. 创建CartServlet

* addBook2Cart() 添加图书到购物车
* deleteCartItem() 删除某图书
* changeCount() 改变图书数量
* Clear() 清空购物车

## 第七阶段 结账与订单详情

结账就是将购物车中的购物项保存到数据库中，并生成订单号返回



OrderItemDao

BookDao

调用OrderService中createOrder()将购物车中的购物项保存到数据库中，并返回订单号

1. 生成订单号,保存订单
2. 保存购物项
3. 修改图书库存和销量
4. 清空购物车

发送请求到OrderClientServlet的checkout方法():

1. 获取用户信息
2. 查询用户是否登录，如果没有去登录页面，有则继续
3. 获取购物车，调用OrderService层的createOrder()
4. 返回结账页面checkout.jsp

OrderDao

### 创建Order与OrderItem类

分别用来封装订单信息与订单项信息，并创建对应的数据库表

1. Order类：

String id; 订单号（根据一定规则生成）

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 ; 订单总金额

Int totalCount ; 订单总数量

Int userId; 用户id

（用户与订单 ：1对n 关系）

订单号特点：

* 具有唯一性
* 尽量少的透露商业信息
* 用于售后，易识别

订单号规则：时间戳+””+用户id

1. OrderItem类：

Integer id ;

String title; 书名

String author; 作者

Double price ; 价格

Int count; 数量

Double amount ; 金额

String path; 封面

String orderId; 订单号

(订单与订单项的关系：1对n关系)

### 创建OrderDao接口及实现类

Int saveOrder(Order order) 保存订单信息

int updateState(String orderid, int state) 修改订单状态

List<Order> getOrderListByUserId(String userId) 提供给用户使用，查看当前用户的所有订单

List<Order> getOrderList() 提供给管理员使用

### 创建OrderItemDao接口及实现类

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

List<OrderItem> getOrderItemListByOrderId(String OrderId) 根据订单号查询所有订单项

### 生成订单功能

* 在BookDao中创建
  + updateStockAndSale(int sale,int stock,int bookId)
* 创建OrderService接口及实现类
  + createOrder(User user,Cart cart) 生成订单
* 创建OrderClientServlet类
  + Checkout() 处理结账请求

检查用户是否登录

检查购物车是否不为空

* 修改cart.jsp页面链接及checkout.jsp页面订单号

### 订单查询

* 在OrderService中
  + List<Order> getOrderListByUserId(String userId) 供用户查询订单
  + List<Order> getOrderList() 供管理员查询订单
* 在OrderClientServlet中
  + orderList() 用户查看所有订单
* 创建OrderManagerServlet
  + orderList() 管理员查看所有订单

### 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接口中添加方法
  + sendBook(String orderId) 发货，管理员调用，状态修改为1
  + takeBook(String orderId) 确认收货，用户调用，状态修改为2
* 在OrderManagerServlet中
  + sendBook() 处理管理员发货请求
* 在OrderClientServlet中
  + takeBook() 处理用户收货请求

### 查看订单详情

查看订单详情实际就是查看当前订单号下的所有订单项信息

* 在OrderService中

List<OrderItem> getOrderInfoByOrderId(String orderId) 根据订单号获取所有订单项信息

* 在OrderClientServlet中添加方法

orderInfo() 处理用户查询订单详情请求

* 在OrderManagerServlet中

orderInfo() 处理用户查询订单详情请求

### 生成订单出现的问题

目前我们在orderService类的createOrder方法中对每一个订单项orderItem获取出来就直接保存到数据库中，同时每一个订单项都要修改一次book表的库存与销量，而这两种底层操作的都是BaseDao的update()方法，在这里对数据库进行打开连接，关闭连接的操作，也就是说，在createOrder()方法中每遍历一个订单项，数据库连接操作就需要获取两次，关闭两次，假如有10个订单项，就需要连接20次，关闭20次，这样会造成资源的浪费，降低服务的性能。解决方案，我们可以采用批量操作。

* BaseDao中需要添加批量方法
  + Batch(String sql ,Object ... params)
* BookDao添加批量修改方法
  + BatchUpdateSaleAndStock(Object[][] params);
* OrderItemDao添加批量保存方法
  + BatchSave(Object[][] params);
* 对orderService中createOrder()进行优化

## 第八阶段 登录验证与结账事务

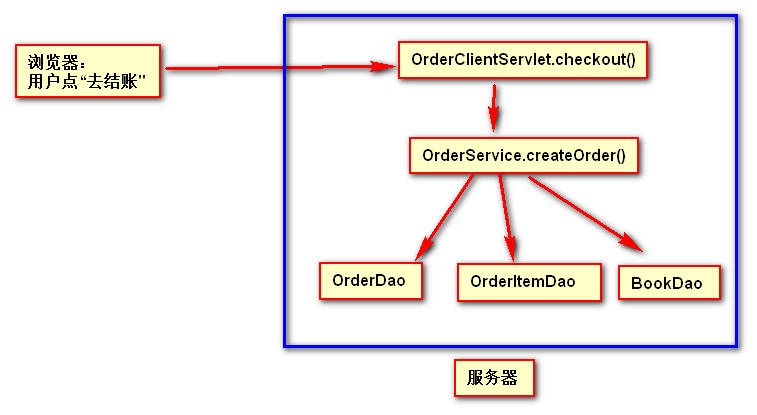
### 1.登录验证

由于OrderClientServlet下的所有方法都需要用户登录后才可进行访问，而如果对每一个方法分别进行获取用户然后进行判断是否登录，这样易出现代码冗余问题，我们希望有哪么一个地方可以统一对用户是否登录进行验证判断。

解决方案：创建一个LoginFilter过滤器用来判断用户是否登录，用来对/client/orderclientServlet进行过滤。如果登录，则进行处理请求，也就是放行;如果没有则重定向到登录页面进行登录操作

### 结账事务处理

* 当我们将OrderDao层的保存订单saveOrder方法中的sql语句写错时，是不能正常保存订单数据的，同时由于订单项需要使用订单的id做为外键找不到原因而导致也不能正常保存，但是图书的库存和销量却被修改。
* 由于保存订单、保存订单项及修改图书库存和销量，它们是属于同一个业务，即然是一个业务，我们希望要么都成功，要么都失败。



1. 这里涉及到事务管理
2. 常规事务用法

Connection conn = JDBCUtil.*getConnection*();

**try** {

//取消自动提交

conn.setAutoCommit(**false**);

//调用Dao处理数据

//提交事务

conn.commit();

} **catch** (SQLException e) {

e.printStackTrace();

**try** {

//不成功，回滚

conn.rollback();

} **catch** (SQLException e1) {

// **TODO** Auto-generated catch block

e1.printStackTrace();

}

}**finally**{

JDBCUtil.*releaseConnection*(conn);

}

1. 在事务中我们使用connection对象必须是同一个，而在Dao层中的每一个方法都是分别获取connection对象，这种方式目前是行不通需要改进。
2. 思路1：可以将connection对象作为参数传入来保证是同一个对象。但是这种方式改动太大，不可行
3. 思路2:
4. 只要保证connection是同一个对象就行，既然如此，我们可将其修改成一个单例模式
5. 添加事务，需要创建一个TransactionFilter过滤器来统一处理事务
6. 出现问题：
7. java.sql.SQLException: You can't operate on a closed Connection!!!

原因：我们在BaseDao层的方法中关闭了connection对象

解决方案：我们不能在BaseDao层方法中关闭connection对象

1. java.sql.SQLException: You have an error in your SQL syntax;

原因：我们在BaseDao层方法中已经处理异常并没有向上抛出导致

解决方案：希望这个异常向上抛出，到TransactionFilter中进行处理

1. java.lang.reflect.InvocationTargetException

原因：是在servlet层通过反射调用目标方法时抛出的异常

解决方案：仍然是向上抛出异常

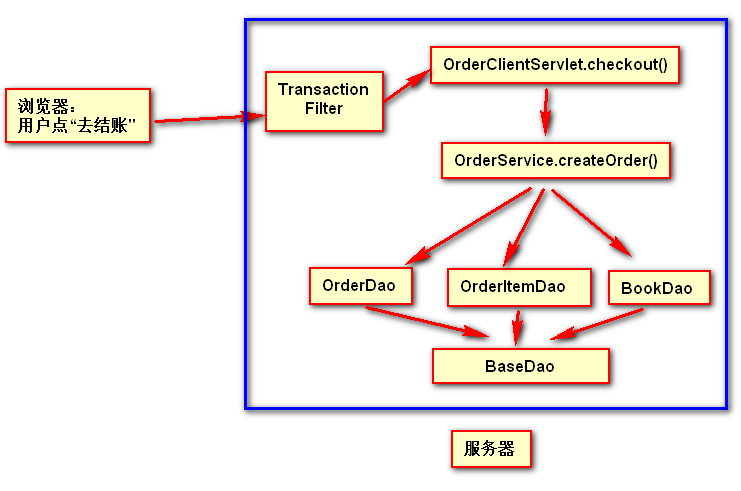
（到此事务问题看似已经解决了，但是在TransactionFilter中还没有真正关闭connection对象)

1. 当在TransactionFilter过滤器中关闭connection对象时，又导致 java.sql.SQLException: You can't operate on a closed Connection!!!异常出现。

原因：由于关闭connection导致，而根本原因是connection对象是一个单例的

解决方案：我们希望一个请求对应一个connection对象，是一个键值对形式，那我们可以对JDBCUtil进行改造，定义一个map结构来保存connection对象。而一个请求实际上就是一个线程，也就是说key就是当前线程，而value就是connection对象。对获取和释放connection对象方法进行修改

1. 到此事务算时解决了。但是并没有彻底解决，因为haspMap不是线程安全的，同学之间可来做这个测试，就是启动项目后，让其他同学在同一时刻向应用发送请求，仍会出现异常。解决方案：可以使用ThreadLocal类



## 第九阶段 使用AJAX完成与后台交互

1. 首页的加入购物车功能可以引入AJAX
   1. 修改home.jsp页面发送ajax到服务器
   2. 修改CartServlet中的addBook2Cart方法，返回json字符串
2. 购物车页面可以使用AJAX
   1. 修改cart.jsp页面发ajax到服务器
   2. 修改CartServlet中的changeCount方法，返回json字符串
3. 注册页面可以使用AJAX来检查用户名是否存在
   1. 在UserDao中添加方法

User getUserByUsername(String username);

* 1. 在userService中添加方法

Boolean checkUsername(String username);

* 1. 在UserServlet中添加方法

checkUsername()

（注意需要导入gson-2.2.4.jar包）

## 第十阶段 修改图书封面

修改图书封面就是将本地图片上传到服务器，将图片保存到static/img目录下，然后将在static/img目录下的图片路径保存到数据库中

步骤：

1. 修改页面book\_edit.jsp：

* 新增上传图片表单项一栏
* 修改form表单的enctype属性值为multipart/form-data

1. 导入jar包

* commons-fileupload-1.3.1.jar
* commons-io-2.4.jar

1. 修改BookManagerServlet中的updateBook方法