# **尚硅谷网上书城**

## 第一阶段：登录、注册的验证

1. 使用jQuery技术对登录中的用户名、密码进行非空验证
2. 使用jQuery技术和正则表达式对注册页面中的用户名、密码、确认密码、邮箱等进行格式验证

## 第二阶段：实现登录、注册

### 软件的三层架构

* 生活中的三层架构



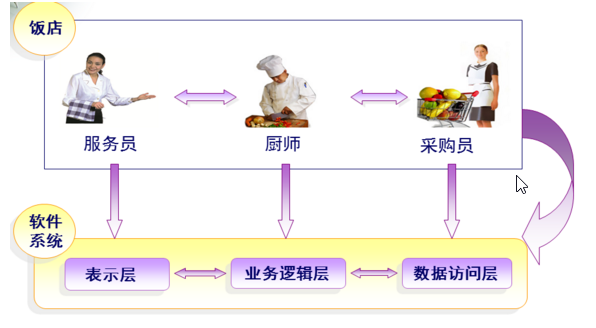
* **服务员：**只管接待客人。
* **厨师：**只管做客人点的菜。
* **采购员：**只管按客人点菜的要求采购食材。

他们各负其职，服务员不用了解厨师如何做菜，不用了解采购员如何采购食材；厨师不用知道服务员接待了哪位客人，不用知道采购员如何采购食材；同样，采购员不用知道服务员接待了哪位客人，不用知道厨师如何做菜。

* 软件中的三层架构

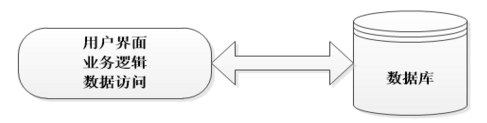


* **UI (User Interface layer表示层):**主要是指与用户交互的界面。用于接收用户输入的数据和显示处理后用户需要的数据。
* **BLL (** **Business Logic Layer业务逻辑层):**UI层和DAL层之间的桥梁。实现业务逻辑。业务逻辑具体包含：验证、计算、业务规划等等。
* **DAL (Data access layer数据访问层):**与数据库打交道。主要实现对数据的增、删、改、查。将存储在数据库中的数据提交给业务层，同时将业务层处理的数据保存到数据库。
* 当然这些操作都是基于UI层的，用户的需求反映给界面UI，UI反映给BLL，BLL反映给DAL，DAL进行数据的操作，操作后再一一返回，直到将用户所需数据反馈给用户。
* 为什么使用三层架构



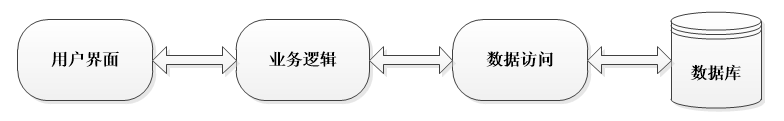
* 使用三层架构的目的：**解耦！！！**
* 同样拿上面饭店的例子来讲：
  + 服务员（UI层）辞职——另找服务员；厨师（BLL层）辞职——招聘另一个厨师；采购员（DAL）辞职——招聘另一个采购员。
  + 店里服务态度不好找服务员；菜有问题找厨师。
* **任何一层发生变化都不会影响到另外一层！！！**
* 三层与一层的对比

**一层：**



* 当任何一个地方发生变化时，都需要重新开发整个系统。“多层”放在一层，分工不明确，**耦合度高**——难以适应需求变化，可维护性低、可扩展性低。

**三层：**



* 哪一层发生变化，只需有针对该层进行修改，不需要更改整个系统。
* 层次清晰，分工明确，每层之间**耦合度低**
* 提高了效率，适应需求变化，可维护性高，可扩展性高。
* 三层架构的优势和劣势
* **优势：**
  1. 结构清晰、耦合度低
  2. 可维护性高，可扩展性高
  3. 利于开发任务同步进行
  4. 容易适应需求变化
* **劣势：**

1. 降低了系统的性能。这是不言而喻的。如果不采用分层式结构，很多业务可以直接造访数据库，以此获取相应的数据，如今却必须通过中间层来完成。
2. 有时会导致级联的修改。这种修改尤其体现在自上而下的方向。如果在表示层中需要增加一个功能，为保证其设计符合分层式结构，可能需要在相应的业务逻辑层和数据访问层中都增加相应的代码。

### 书城的三层架构

* 表示层
* HTML、Servlet
* 接受用户的请求，调用业务逻辑层处理用户请求，显示处理结果
* 业务逻辑层
* Service
* 调用数据访问层处理业务逻辑
* 采用面向接口编程的思想，先定义接口，再创建实现类
* 数据访问层
* Dao
* 用来操作数据库，对数据库进行增删改查
* 采用面向接口编程的思想，先定义接口，再创建实现类

### 书城的三层架构具体实现

## 第三阶段：项目优化

### 将所有HTML页面改为JSP页面

* 复制一份BookStore02，重命名为BookStore03，仅此还不够！还需要在BookStore03工程上右键→Properties→搜索Web→单机Web Project Settings→修改Context root为BookStore03。
* 在每一个HTML页面的第一行添加JSP的page指令：

<%@ page language=*"java"* contentType=*"text/html; charset=UTF-8"* pageEncoding=*"UTF-8"*%>，然后将HTML页面的文件扩展名改为jsp，即将.html改为.jsp。**注意：一定要先在HTML页面中添加jsp指令再修改扩展名**，否则页面中会出现乱码现象。

* 动态获取base标签

<base href=*"http://*<%=request.getServerName() %>*:*<%=request.getServerPort() %><%=request.getContextPath() %>*/"*>

* 获取IP地址：**request.getServerName()**
* 获取端口号：**request.getServerPort()**
* 获取项目虚拟路径：**request.getContextPath()**
* 提取出页面中重复的部分，在通过include指令将这些内容分别引入网页
* 创建一个base.jsp页面，删除除page指令之外的所有内容；将base标签和link标签复制到base.jsp页面中
* 为了不让用户能够直接访问base.jsp页面，在WEB-INF下创建一个include目录，将base.jsp页面放入该目录中。

### 显示错误消息及表单回显

* 登录、注册失败时我们分别转发到了登录、注册页面，但是并没有给用户一个错误消息，用户体验差，现在我们分别为登录、注册失败时提供一个错误消息并在页面显示：
  + 登录、注册失败时我们将错误消息放到request域中，通过**转发**将错误消息带到登录、注册页面，然后在页面中通过request域对象获取错误消息。
* 登录、注册失败时我们输入的用户名等信息会被清空，为了让用户有更好的体验，让用户看到之前输入的内容，需要对表单进行回显：
  + 给需要回显的表单项设置value属性，value值通过**request.getParameter("属性值")**来获取。

### 局部优化

* 问题1：目前我们的项目中是一个功能对应一个Servlet，登录有LoginServlet、注册有RegistServlet、删除用户有DelUserServlet、修改用户有UpdateUserServlet、添加图书有AddBookServlet。。。这样会导致我们的项目被淹没在Servlet海洋中。
* 优化一（整合Servlet）：目前项目中是一个请求对应一个Servlet，**我们能不能将多个请求对应一个Servlet呢？**比如用户登录、注册的LoginServlet、RegistServlet使用一个UserServlet代替：
* 在用户发送请求的URL地址后面添加一个method请求参数，在UserServlet中添加两个方法login()和regist()。**但是**，两个方法的格式与doGet和doPost要保持一致，可以复制一个doGet或doPost方法，将方法名改为login或regist。
* 登录表单提交到**UserServlet?method=login**，注册表单提交到**UserServlet?method=regist**。在UserServlet的doGet方法中获取请求参数，在doPost方法中调用doGet方法；在doGet方法中使用if else语句判断method的值，如果是login就调用login方法，如果是regist就调用regist方法。
* 在UserServlet中创建UserService对象，将LoginServlet、RegistServlet的doPost方法中的内容分别复制到UserServlet的login方法和regist方法中即可实现之前的登录注册功能。
* 优化二（动态调用方法）：在优化一中实现了多个请求对应一个Servlet的需求，但是我们在UserServlet中每添加一个方法就要在doGet方法中添加一个 else if语句！而且我们发现我们获取的请求参数的值正好是UserServlet中的方法名**，那么我们能不能通过请求参数动态的调用对应的方法呢？**
  + 通过**反射**的形式动态获取方法对象，这样我们再添加新的方法时就不需要再写if else语句了，但是新添加的方法的格式仍需要与doGet或者doPost的格式一致。
* 根据请求方式获取方法名
  + - * String **methodName** = request.getParameter("method");
* 根据方法名获取方法的对象，**getDeclaredMethod()**方法用来获取当前类中的某一个方法；第一个参数是方法名，第二个参数是方法的参数类型
  + - * Method method = this.getClass().getDeclaredMethod(**methodName**, HttpServletRequest.class,HttpServletResponse.class);
* 设置访问权限
* method.setAccessible(true);
* 调用方法。invoke()方法用来调用一个方法；第一个参数是要调用哪个对象的方法，第二个参数是调用方法需要传的参数
  + method.invoke(this, request,response);
* 优化三（创建BaseServlet）：在优化二中我们实现了动态调用对应方法的需求，但是以后每次创建Servlet的时候都需要将以上代码重写一遍，那么我们为何不把以上代码单独放到一个Servlet中，我们给它命名为BaseServlet，以后再创建Servlet时直接让它继承BaseServlet即可。
* 创建一个专门用来被其他Servlet继承的BaseServlet
* **但是不可以重写BaseServlet中的doGet和doPost方法**
* 注意：

1. 当我们使用get请求时，get请求会覆盖action中的请求参数
2. 当使用get请求时，可以在表单中设置一个表单隐藏域，这个域的name属性是method，值是要调用的方法

|  |
| --- |
| <input type=*"hidden"* name=*"method"* value=*"login"* /> |

* **问题2：在大部分的Servlet中都需要先获取用户发送的请求参数，然后将这些请求参数封装为对象，像这些操作都是一些比较简单，但是又比较繁琐（尤其是当我们需要做转型操作）**
* 问题2解决：写一个通用工具类来替咱们做这件事（BeanUtils）。
* JavaBean
* 属性私有化，通过getter和setter方法对属性进行读取和设置
* 有一个无参的构造器
* 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。
* 使用BeanUtils可以直接将一个Map转换一个JavaBean的对象。
* 我们想可不可以获取到请求参数的map，然后将这个map直接转换为我们想要的对象。
* 我们通过BeanUtils的populate方法将请求参数的map直接封装为对象，但是这块有一个问题需要注意表单项中name属性要和类中的属性名一样，否则将不能进行封装。

### 使用EL替换项目中的所有JSP表达式

## 第四阶段：图书的增删改查

### 管理端图书的增删改查

* 创建图书类Book



* private Integer id;
* private String title; //书名
* private String author; //作者
* private Double price; //价格
* private Integer sales; //销量
* private Integer stock; //库存
* private String imgPath; //封面图片的路径
  + 指定一个默认值 static/img/default.jpg
* 创建图书表books
* 创建BookDao接口及实现类
* public void saveBook(Book book); 向数据库中插入一本图书
* public void deleteBook(**String** bookId); 根据图书的ID删除一本图书
* public void updateBook(Book book); 更新一本图书
* public Book getBookById(**String** bookId); 根据图书的ID查询一本图书
* public List<Book> getBookList(); 获取所有的图书
* 创建BookService接口及实现类
* public void saveBook(Book book); 向数据库中插入一本图书
* public void deleteBook(String bookId); 根据图书的ID删除一本图书
* public void updateBook(Book book); 更新一本图书
* public Book getBookById(String bookId); 根据图书的ID查询一本图书
* public List<Book> getBookList(); 获取所有的图书
* 创建BookManagerServlet
* bookList() → 获取图书列表的方法
* addBook() → 添加图书的方法
* deleteBook() → 删除图书的方法
* editBook() → 编辑图书的方法
* saveOrUpdate() → 保存或者更新图书的方法
* 修改图书操作
  + 当我们点击修改图书时，首先我们需要根据当前图书的ID去数据库中将该图书的信息查询出来显示到编辑图书的页面book\_edit.jsp。
    - editBook() → 通过ID从数据库中查询到该图书显示到book\_edit.jsp页面。
* 当修改完图书之后我们需要把修改之后的图书信息提交到一个更新图书的方法中去，不过这个时候我们添加图书和修改图书的页面都是book\_edit.jsp，**那么我们怎么判断我们是在添加图书还是在修改图书呢？**
  + **方式一：**我们可以将添加图书和修改图书分离，再创建一个jsp页面
  + **方式二：**添加图书没有ID，但是修改图书有ID，所有我们可以通过是否有ID来判断当前是添加图书的操作还是修改图书的操作，不过我们需要把图书的ID放到一个隐藏域中。
    - **将之前的addBook()方法改为saveOrUpdate()方法**，在该方法中通过判断获取的图书ID是否为null来决定是调用BookService的saveBook方法还是update()方法。
* 至此，我们管理端图书的增删改查已经完成，但是我们在获取图书列表的方法中是将数据库中的所有图书都查询了出来，如果数据库中的图书过多在页面显示时就会出现滚动条，不但用户体验不好而且一次性从数据库中查询这么多条记录性能也差。所以**我们需要对图书进行分页显示**：
  + 在sql语句中使用limit关键字进行分页

select id,title,author,sales,stock,price,img\_path imgPath from bs\_book limit ?,?

* + 我们知道从数据库中查询出来的图书放到了一个List集合中，但是实现分页功能只有list中的book数据是不够的，我们还需要知道总页数、当前页码、总记录数等信息，但是List中并没有这些信息，所有我们需要**创建一个类来封装分页的信息**。



* Page<T>类
* public int pageSize; //每页显示的条数 在Servlet中指定
* private int pageNo; //当前页码 ， 该数据在Servlet中获取
* Private int index // 分页开始的索引 ， 通过计算获得
* private int totalPageNo; //总页数 ， 需要通过计算获得
* private int totalRecord; //总记录 ， 通过查询数据库获得
* private List<T> list; //分页的数据 ， 通过数据库查询
* 修改BookDao：添加一个分页的方法
  + Page<Book> findBook(Page<Book> page) → 分页查找图书的方法
    - 该方法由BookService调用，传过来一个带有pageNo的Page<Book> 类型的page参数，通过该方法从数据库中先获取总的记录数，然后通过page的(pageNo-1)\*PAGE\_SIZE和PAGE\_SIZE获取一个带分页的List集合。给page设置了总的记录数和List后再将page返回。
* 修改BookService：同样添加一个分页的方法
  + Page<Book> findBook(String pageNo,int pageSize) → 分页查找图书的方法
    - 该方法由Servlet调用，接受的请求参数都是String类型的，所以传入的页码pageNo设置为了字符串类型。实现该方法是首先要创建一个Page<Book>对象，然后设置pageNo属性，不过这里需要将获取的字符串类型的pageNo转换为int类型。**注意：有可能会出现转换异常（例如用户输入的是字符串abc）**
* 在BookManagerServlet中添加findBook()方法
* **这时我们发现我们原来的bookList()已经没用了……**
  + 就会导致原来添加图书、修改图书、删除图书出现bug。
  + 改完bug之后考虑一个问题：目前我们执行添加图书、删除图书、修改图书之后都回到了图书列表的首页，**我们能不能让它回到执行之前的那个页面呢？（提示：我们可以通过请求头中的Referer获取之前的地址）**
* 关于分页页码的问题
  + 当页数过多时，页码的导航栏会显得非常非常的长，看着非常非常的不爽
  + [首页](http://localhost:8080/BookStore_EG05/manager/BookManagerServlet?method=findBook&pageNumber=1) [上一页](http://localhost:8080/BookStore_EG05/manager/BookManagerServlet?method=findBook&pageNumber=18) [1](http://localhost:8080/BookStore_EG05/manager/BookManagerServlet?method=findBook&pageNumber=1) [2](http://localhost:8080/BookStore_EG05/manager/BookManagerServlet?method=findBook&pageNumber=2) [3](http://localhost:8080/BookStore_EG05/manager/BookManagerServlet?method=findBook&pageNumber=3) [4](http://localhost:8080/BookStore_EG05/manager/BookManagerServlet?method=findBook&pageNumber=4) [5](http://localhost:8080/BookStore_EG05/manager/BookManagerServlet?method=findBook&pageNumber=5) [6](http://localhost:8080/BookStore_EG05/manager/BookManagerServlet?method=findBook&pageNumber=6) [7](http://localhost:8080/BookStore_EG05/manager/BookManagerServlet?method=findBook&pageNumber=7) [8](http://localhost:8080/BookStore_EG05/manager/BookManagerServlet?method=findBook&pageNumber=8) [9](http://localhost:8080/BookStore_EG05/manager/BookManagerServlet?method=findBook&pageNumber=9) [10](http://localhost:8080/BookStore_EG05/manager/BookManagerServlet?method=findBook&pageNumber=10) [11](http://localhost:8080/BookStore_EG05/manager/BookManagerServlet?method=findBook&pageNumber=11) [12](http://localhost:8080/BookStore_EG05/manager/BookManagerServlet?method=findBook&pageNumber=12) [13](http://localhost:8080/BookStore_EG05/manager/BookManagerServlet?method=findBook&pageNumber=13) [14](http://localhost:8080/BookStore_EG05/manager/BookManagerServlet?method=findBook&pageNumber=14) [15](http://localhost:8080/BookStore_EG05/manager/BookManagerServlet?method=findBook&pageNumber=15) [16](http://localhost:8080/BookStore_EG05/manager/BookManagerServlet?method=findBook&pageNumber=16) [17](http://localhost:8080/BookStore_EG05/manager/BookManagerServlet?method=findBook&pageNumber=17) [18](http://localhost:8080/BookStore_EG05/manager/BookManagerServlet?method=findBook&pageNumber=18) [19] [20](http://localhost:8080/BookStore_EG05/manager/BookManagerServlet?method=findBook&pageNumber=20) [21](http://localhost:8080/BookStore_EG05/manager/BookManagerServlet?method=findBook&pageNumber=21) [22](http://localhost:8080/BookStore_EG05/manager/BookManagerServlet?method=findBook&pageNumber=22) [23](http://localhost:8080/BookStore_EG05/manager/BookManagerServlet?method=findBook&pageNumber=23) [24](http://localhost:8080/BookStore_EG05/manager/BookManagerServlet?method=findBook&pageNumber=24) [25](http://localhost:8080/BookStore_EG05/manager/BookManagerServlet?method=findBook&pageNumber=25) [26](http://localhost:8080/BookStore_EG05/manager/BookManagerServlet?method=findBook&pageNumber=26) [27](http://localhost:8080/BookStore_EG05/manager/BookManagerServlet?method=findBook&pageNumber=27) [28](http://localhost:8080/BookStore_EG05/manager/BookManagerServlet?method=findBook&pageNumber=28) [29](http://localhost:8080/BookStore_EG05/manager/BookManagerServlet?method=findBook&pageNumber=29) [30](http://localhost:8080/BookStore_EG05/manager/BookManagerServlet?method=findBook&pageNumber=30) [31](http://localhost:8080/BookStore_EG05/manager/BookManagerServlet?method=findBook&pageNumber=31) [32](http://localhost:8080/BookStore_EG05/manager/BookManagerServlet?method=findBook&pageNumber=32) [33](http://localhost:8080/BookStore_EG05/manager/BookManagerServlet?method=findBook&pageNumber=33) [34](http://localhost:8080/BookStore_EG05/manager/BookManagerServlet?method=findBook&pageNumber=34) [35](http://localhost:8080/BookStore_EG05/manager/BookManagerServlet?method=findBook&pageNumber=35) [36](http://localhost:8080/BookStore_EG05/manager/BookManagerServlet?method=findBook&pageNumber=36) [37](http://localhost:8080/BookStore_EG05/manager/BookManagerServlet?method=findBook&pageNumber=37) [下一页](http://localhost:8080/BookStore_EG05/manager/BookManagerServlet?method=findBook&pageNumber=20) [末页](http://localhost:8080/BookStore_EG05/manager/BookManagerServlet?method=findBook&pageNumber=37) 共37页
* 解决：
  + 百度的页码是如何显示
  + [首页](http://localhost:8080/BookStore_EG05/manager/BookManagerServlet?method=findBook&pageNumber=1) [上一页](http://localhost:8080/BookStore_EG05/manager/BookManagerServlet?method=findBook&pageNumber=18) [[1](http://localhost:8080/BookStore_EG05/manager/BookManagerServlet?method=findBook&pageNumber=1)] [2](http://localhost:8080/BookStore_EG05/manager/BookManagerServlet?method=findBook&pageNumber=2) [3](http://localhost:8080/BookStore_EG05/manager/BookManagerServlet?method=findBook&pageNumber=3) [4](http://localhost:8080/BookStore_EG05/manager/BookManagerServlet?method=findBook&pageNumber=4) [5](http://localhost:8080/BookStore_EG05/manager/BookManagerServlet?method=findBook&pageNumber=5) [6](http://localhost:8080/BookStore_EG05/manager/BookManagerServlet?method=findBook&pageNumber=6) [7](http://localhost:8080/BookStore_EG05/manager/BookManagerServlet?method=findBook&pageNumber=7) [8](http://localhost:8080/BookStore_EG05/manager/BookManagerServlet?method=findBook&pageNumber=8) [9](http://localhost:8080/BookStore_EG05/manager/BookManagerServlet?method=findBook&pageNumber=9) [10](http://localhost:8080/BookStore_EG05/manager/BookManagerServlet?method=findBook&pageNumber=10) [末页](http://localhost:8080/BookStore_EG05/manager/BookManagerServlet?method=findBook&pageNumber=37) 共37页
  + [首页](http://localhost:8080/BookStore_EG05/manager/BookManagerServlet?method=findBook&pageNumber=1) [上一页](http://localhost:8080/BookStore_EG05/manager/BookManagerServlet?method=findBook&pageNumber=18) [1](http://localhost:8080/BookStore_EG05/manager/BookManagerServlet?method=findBook&pageNumber=1) [[2](http://localhost:8080/BookStore_EG05/manager/BookManagerServlet?method=findBook&pageNumber=2)] [3](http://localhost:8080/BookStore_EG05/manager/BookManagerServlet?method=findBook&pageNumber=3) [4](http://localhost:8080/BookStore_EG05/manager/BookManagerServlet?method=findBook&pageNumber=4) [5](http://localhost:8080/BookStore_EG05/manager/BookManagerServlet?method=findBook&pageNumber=5) [6](http://localhost:8080/BookStore_EG05/manager/BookManagerServlet?method=findBook&pageNumber=6) [7](http://localhost:8080/BookStore_EG05/manager/BookManagerServlet?method=findBook&pageNumber=7) [8](http://localhost:8080/BookStore_EG05/manager/BookManagerServlet?method=findBook&pageNumber=8) [9](http://localhost:8080/BookStore_EG05/manager/BookManagerServlet?method=findBook&pageNumber=9) [10](http://localhost:8080/BookStore_EG05/manager/BookManagerServlet?method=findBook&pageNumber=10) [末页](http://localhost:8080/BookStore_EG05/manager/BookManagerServlet?method=findBook&pageNumber=37) 共37页
  + [首页](http://localhost:8080/BookStore_EG05/manager/BookManagerServlet?method=findBook&pageNumber=1) [上一页](http://localhost:8080/BookStore_EG05/manager/BookManagerServlet?method=findBook&pageNumber=18) [1](http://localhost:8080/BookStore_EG05/manager/BookManagerServlet?method=findBook&pageNumber=1) [2](http://localhost:8080/BookStore_EG05/manager/BookManagerServlet?method=findBook&pageNumber=2) [[3](http://localhost:8080/BookStore_EG05/manager/BookManagerServlet?method=findBook&pageNumber=3)] [4](http://localhost:8080/BookStore_EG05/manager/BookManagerServlet?method=findBook&pageNumber=4) [5](http://localhost:8080/BookStore_EG05/manager/BookManagerServlet?method=findBook&pageNumber=5) [6](http://localhost:8080/BookStore_EG05/manager/BookManagerServlet?method=findBook&pageNumber=6) [7](http://localhost:8080/BookStore_EG05/manager/BookManagerServlet?method=findBook&pageNumber=7) [8](http://localhost:8080/BookStore_EG05/manager/BookManagerServlet?method=findBook&pageNumber=8) [9](http://localhost:8080/BookStore_EG05/manager/BookManagerServlet?method=findBook&pageNumber=9) [10](http://localhost:8080/BookStore_EG05/manager/BookManagerServlet?method=findBook&pageNumber=10) [末页](http://localhost:8080/BookStore_EG05/manager/BookManagerServlet?method=findBook&pageNumber=37) 共37页
  + [首页](http://localhost:8080/BookStore_EG05/manager/BookManagerServlet?method=findBook&pageNumber=1) [上一页](http://localhost:8080/BookStore_EG05/manager/BookManagerServlet?method=findBook&pageNumber=18) [1](http://localhost:8080/BookStore_EG05/manager/BookManagerServlet?method=findBook&pageNumber=1) [2](http://localhost:8080/BookStore_EG05/manager/BookManagerServlet?method=findBook&pageNumber=2) [3](http://localhost:8080/BookStore_EG05/manager/BookManagerServlet?method=findBook&pageNumber=3) [[4](http://localhost:8080/BookStore_EG05/manager/BookManagerServlet?method=findBook&pageNumber=4)] [5](http://localhost:8080/BookStore_EG05/manager/BookManagerServlet?method=findBook&pageNumber=5) [6](http://localhost:8080/BookStore_EG05/manager/BookManagerServlet?method=findBook&pageNumber=6) [7](http://localhost:8080/BookStore_EG05/manager/BookManagerServlet?method=findBook&pageNumber=7) [8](http://localhost:8080/BookStore_EG05/manager/BookManagerServlet?method=findBook&pageNumber=8) [9](http://localhost:8080/BookStore_EG05/manager/BookManagerServlet?method=findBook&pageNumber=9) [10](http://localhost:8080/BookStore_EG05/manager/BookManagerServlet?method=findBook&pageNumber=10) [末页](http://localhost:8080/BookStore_EG05/manager/BookManagerServlet?method=findBook&pageNumber=37) 共37页
  + [首页](http://localhost:8080/BookStore_EG05/manager/BookManagerServlet?method=findBook&pageNumber=1) [上一页](http://localhost:8080/BookStore_EG05/manager/BookManagerServlet?method=findBook&pageNumber=18) [1](http://localhost:8080/BookStore_EG05/manager/BookManagerServlet?method=findBook&pageNumber=1) [2](http://localhost:8080/BookStore_EG05/manager/BookManagerServlet?method=findBook&pageNumber=2) [3](http://localhost:8080/BookStore_EG05/manager/BookManagerServlet?method=findBook&pageNumber=3) [4](http://localhost:8080/BookStore_EG05/manager/BookManagerServlet?method=findBook&pageNumber=4) [[5](http://localhost:8080/BookStore_EG05/manager/BookManagerServlet?method=findBook&pageNumber=5)] [6](http://localhost:8080/BookStore_EG05/manager/BookManagerServlet?method=findBook&pageNumber=6) [7](http://localhost:8080/BookStore_EG05/manager/BookManagerServlet?method=findBook&pageNumber=7) [8](http://localhost:8080/BookStore_EG05/manager/BookManagerServlet?method=findBook&pageNumber=8) [9](http://localhost:8080/BookStore_EG05/manager/BookManagerServlet?method=findBook&pageNumber=9) [10](http://localhost:8080/BookStore_EG05/manager/BookManagerServlet?method=findBook&pageNumber=10) [末页](http://localhost:8080/BookStore_EG05/manager/BookManagerServlet?method=findBook&pageNumber=37) 共37页
  + [首页](http://localhost:8080/BookStore_EG05/manager/BookManagerServlet?method=findBook&pageNumber=1) [上一页](http://localhost:8080/BookStore_EG05/manager/BookManagerServlet?method=findBook&pageNumber=18) [2](http://localhost:8080/BookStore_EG05/manager/BookManagerServlet?method=findBook&pageNumber=2) [3](http://localhost:8080/BookStore_EG05/manager/BookManagerServlet?method=findBook&pageNumber=3) [4](http://localhost:8080/BookStore_EG05/manager/BookManagerServlet?method=findBook&pageNumber=4) [5](http://localhost:8080/BookStore_EG05/manager/BookManagerServlet?method=findBook&pageNumber=5) [[6](http://localhost:8080/BookStore_EG05/manager/BookManagerServlet?method=findBook&pageNumber=6)] [7](http://localhost:8080/BookStore_EG05/manager/BookManagerServlet?method=findBook&pageNumber=7) [8](http://localhost:8080/BookStore_EG05/manager/BookManagerServlet?method=findBook&pageNumber=8) [9](http://localhost:8080/BookStore_EG05/manager/BookManagerServlet?method=findBook&pageNumber=9) [10](http://localhost:8080/BookStore_EG05/manager/BookManagerServlet?method=findBook&pageNumber=10) [11](http://localhost:8080/BookStore_EG05/manager/BookManagerServlet?method=findBook&pageNumber=11) [下一页](http://localhost:8080/BookStore_EG05/manager/BookManagerServlet?method=findBook&pageNumber=20) [末页](http://localhost:8080/BookStore_EG05/manager/BookManagerServlet?method=findBook&pageNumber=37) 共37页
  + 百度是页里最多只显示10个页码
  + 咱们设计为页面中最多显示5页码
    - [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
  + 这个问题就转变成了需要动态的设置forEach中begin和end的值
  + 根据不同的情况去设置begin和end的值
    - 第一种情况，总页数小于等于5
      * begin=1 end=总页数
    - 第二种情况，当前页 小于等于 3
      * begin=1 end=5
    - 第三种情况，当前页 大于3
      * begin=当前页-2 end=当前页+2

<c:choose>

<c:when test=*"*${page.totalPageNo < 5 }*"*>

<c:set var=*"begin"* value=*"1"*></c:set>

<c:set var=*"end"* value=*"*${page.totalPageNo }*"*></c:set>

</c:when>

<c:when test=*"*${page.pageNo <=3 }*"*>

<c:set var=*"begin"* value=*"1"*></c:set>

<c:set var=*"end"* value=*"5"*></c:set>

</c:when>

<c:otherwise>

<c:set var=*"begin"* value=*"*${page.pageNo - 2 }*"*></c:set>

<c:set var=*"end"* value=*"*${page.pageNo + 2 }*"*></c:set>

<c:if test=*"*${end > page.totalPageNo }*"*>

<c:set var=*"begin"* value=*"*${page.totalPageNo - 4 }*"*></c:set>

<c:set var=*"end"* value=*"*${page.totalPageNo}*"*></c:set>

</c:if>

</c:otherwise>

</c:choose>

* **分页完成之后如何复用分页**
  + 使页码超链接的请求地址动态显示
    - 获取请求地址
    - 获取查询字符串
    - 将请求地址与查询字符串使用?拼接
    - 给Page类添加一个path属性，将拼接好的地址设置到page对象的path属性中

### 客户端图书的显示

* 创建BookClientServlet
  + 添加FindBook()方法获取分页图书信息。
* 根据用户输入的价格查询图书
  + 在BookDao中添加一个根据价格查询图书记录的方法
    - Page<Book> findBookByPrice(Page<Book> page,double min,double max);
* 在BookService中添加一个根据价格查询图书记录方法
  + Page<Book> findBookByPrice(String pageNo,String min,String max);
* 在BookClientServlet中添加findBookByPrice()方法
* 完成了查询按钮之后我们发现一个问题：
  + **点击下一页时查询条件丢失，如何实现带查询条件的分页呢：**
    - 由于我们查询操作的表单使用的是Post请求，请求参数在请求体中，所以获取请求参数时获取不到最低价格和最高价格。
    - 如果我们将post请求改为**get请求**，那么我们提交表单时action属性中的method请求参数将被覆盖，即method=findBookByPrice将丢失，所有我们决定将**findBookByPrice请求参数放到一个隐藏域中**
* **修改index.jsp页面**
* 将首页里的内容复制到另一个页面中
  + 删除首页的内容,之后使用<jsp:forward>标签转发到获取图书列表的servlet

## 第五阶段：登录、登出 、验证码、购物车

### 登录、登出

* 登录
* 目前项目登录和没登录没有任何却别，那我们为了区分没登录和登录的状态，我们需要在用户登录成功以后将用户信息放入到Session域中，这样我们就可以根据域中是否包含有User对象来判断用户是否登录。
* 登出
* 用户做登出操作的时候，需要访问一个Servlet，所以我们在UserServlet中添加一个新的方法，叫logout，在这个方法中，直接将session强制失效。

### 验证码

* 验证码的原理实际上和token一样，只不过我们之前所使用的token不用用户手动输入，而验证码需要用户手动输入。
* 验证码的作用，防止恶意注册。
* 在注册页面需要输入验证码，Servlet在处理注册功能时，需要先检查验证码是否正确，如果正确才允许注册，否则直接返回注册页面重新输入。
* 验证码使用流程：

在系统随机生成一个字符串，将字符串保存到session中

将请求提交给UserServlet的regist方法

用户对照图片输入验证码

将图片放入到页面中

将字符串转换为一个图片

返回注册页面重新输入

验证码正确

直接处理请求

在方法中检查验证码是否正确，将字符串从session中移除。

错误

使用一个第三方工具，替咱们干这些事。kaptcha-2.3.2

Kaptcha这个工具中有一个Servlet，com.google.code.kaptcha.servlet.KaptchaServlet，我们需要在项目中手动对该Servlet进行映射，当我们通过浏览器去访问这个Servlet，在该Servlet中，首先他会随机生成一个字符串，然后将字符串放入进session域中，最后返回一个由该字符串转换成的图片。

可以在通过Servlet的初始化参数来对Kaptcha进行一个个性化的设置。

* 使用Google提供的第三方jar包
  + 导入kaptcha-2.3.2.jar包
  + 在web.xml文件中注册Servlet
  + 在Session对象中获取验证码，**获取之后不要忘记移除**

### 购物车

* 购物车实现的三种方式：
  + 基于Cookie的，将购物信息都放入到Cookie中，由浏览器保存。
  + 基于Session的，将购物信息都放入到Session中，由服务器保存。√
  + 基于数据库表的，将购物信息都放入数据库中，由数据库保存。



购物项

购物车

* 创建购物项类CartItem
  + private Book book; //图书信息
  + private int count; //图书的数量
  + private double amount; // 购物项中图书的金额，通过计算得到
* 创建购物车类Cart
  + private Map<String, CartItem> map = new LinkedHashMap<String, CartItem>();
    - 保存购物项的Map，key是bookId，value是CartItem
    - 使用LinkedHashMap是为了保证购物车中添加图书的顺序
* private int totalCount;
  + 购物车中商品的总数量，通过计算得到
* private double totalAmount;
  + 购物车中商品的总金额，通过计算得到
* **另外还需要在购物车类中添加一些方法**：
  + public List<CartItem> getCartItems();
    - 获取购物车中所有的购物项，以便以后遍历
* public void addBook2Cart(Book book);
  + 向购物车中添加图书
    - **注意：需要判断购物车中是否已经存在该图书：如果存在，让数量加一；如果不存在，首先需要创建一个购物项，然后在购物项中setBook，设置数量为1，最后将该购物项添加到购物车的map中。**
* public void delCartItem(String bookId);
  + 删除一个购物项
* public void updateCount(String bookId, String countStr);
  + 更新购物项
    - 根据图书ID和用户输入的数量更新购物项中图书的数量
* public void Clear();
  + 清空购物车
* 处理购物车请求的CartServlet

里面需要四个方法：添加图书、更新购物项、删除购物项、清空购物车

* + add2Cart() 🡪 添加一本图书到购物车
  + clear() 🡪 清空购物车
  + delCartItem() 🡪 删除指定的购物项

## 第六阶段：结账、添加事务、使用Ajax

### 结账

* 结账就是把我们购物车中的东西转换成订单，我们之前将购物车中的东西都放在了Session对象中，但是订单中的东西我们就需要**保存在数据库中**了，所以我们就需要创建类和表。
* 订单



* 查看详情



* 和购物车类似，我们需要创建订单类和订单项类
* 订单类Order
  + private String id; // 订单号
  + private Date orderTime; // 生成订单的时间
  + private int totalCount; // 商品总数量
  + private double totalAmount; // 商品总金额
  + private int state; // 订单状态 0：未发货 1：已发货 2：交易完成
  + private int userId; // 订单所属的用户
  + 对于订单号的要求：
    - 不能重复，唯一
    - 尽量少体现出订单信息
  + 订单和用户的关系
    - 多对一的关系：一个用户可以有多个订单，一个订单只能属于一个用户
* 订单项类OrderItem
  + private Integer id; // 订单项id
  + private int count; // 买了多少本图书
  + private double amount; // 买了count本图书花费的钱数
  + private String title; // 所买图书的书名
  + private String author; // 图书的作者
  + private double price; // 图书的价格
  + private String imgPath; // 图书封面
  + private String orderId; // 订单项所属的订单
  + 订单项和订单的关系
    - 多对一的关系：一个订单可以包含多个订单项，一个订单项属于一个订单
* 操作订单的接口OrderDao
  + public void saveOrder(Order order); // 向数据库中添加订单
  + public List<Order> getOrders(); //获取所有订单
  + public List<Order> getOrdersByUserId(int userId); //根据用户id查询用户的所有订单
  + public void updateOrderState(String orderId, int state); //根据用户id和状态码更新订单状态
* 操作订单项的接口OrderItemDao
  + public void saveOrderItem(OrderItem orderItem); //向数据库中添加订单项
  + public List<OrderItem> getOrderItemsByOrderId(String orderId); //根据订单号查询所有订单项
* 处理订单相关业务的接口OrderService
  + public String createOrder(Cart cart, User user); // 生成订单，返回订单号
    - **注意：这个方法里面的业务逻辑比较多**
      1. **根据当前时间和用户Id拼接订单号**
      2. **创建订单对象，将购物车中的信息封装到订单对象中并将订单保存到数据库中**
      3. **创建订单项对象，将购物车中购物项的信息封装到订单项中，其中不要忘记更新图书的销量和库存，然后将订单项保存到数据库中**
      4. **最后还需要清空购物车**
* public List<Order> getOrders(); //获取所有的订单
* public List<Order> getOrdersByUserId(int userId); //根据用户ID获取该用户的订单
* public List<OrderItem> getOrderItemsByOrderId(String orderId); //根据订单号获取所有订单项
* public void sendOrder(String orderId); //管理员发货
* public void takeOrder(String orderId); //用户确认收货
* **不管是发货还是收货，本质就是更新订单的状态**
* 客户端订单管理
  + 结账、我的订单、订单详情、确认收货
    - 以上操作都需要判断用户是否登录
* 管理端订单管理
  + 订单管理、发货

### 添加事务

* 如果我们将saveOrder的SQL语句故意写错，模拟一个数据库出现异常的情况，订单将不能正常插入到数据库中，同时，由于订单号不存在，订单项也将不能插入进数据库。**但是**，图书的**库存**和**销量**却发生了改变。
* 所以我们要在项目中添加事务控制
* 我们在结账的操作中分别调用了三个DAO，OrderDao、OrderItemDao、BookDao。对数据库做操作，**这三次操作就是一个事务，三个操作要么都成，要么都失败。**
* **事务控制的关键：在同一个事务中，不同的操作要求使用同一个的数据库连接**
* **使用ThreadLocal对象保证一个线程对应一个数据库连接**
  + 在Java中有一个类叫ThreadLocal，在ThreadLocal的内部实际维护着一个Map，这个Map的Key就是当前线程对象，值就是你想存的的对象，**所以我们一般使用ThreadLocal来在同一个线程中共享数据**，他的作用有点像域对象。
  + **ThreadLocal<T>中常用的方法**
    - void set(T t) 向ThreadLocal保存一个对象
    - T get() 获取当前线程中保存的对象
    - void remove() 移除当前线程中保存的对象
* **使用过滤器Filter控制事务**
  + 我们在项目中添加一个TransactionFilter，用于控制事务，我们在Filter中先获取数据库连接，然后开启事务，接着放行请求，当响应回来的时候，还会回到Filter，这时我们的Filter可以对异常进行捕获，如果出现异常则回滚事务，如果没出现则提交事务。
  + 如果添加TransactionFilter，**那么所有的数据库连接的关闭操作都应该在Filter中统一处理**，而不应该再在DAO中操作，所以BaseDao中所有释放数据库连接的代码，全都应该注释掉。
  + 我们在Filter统一处理事务，那么就希望异常可以一直抛到Filter中，然后Filter一旦捕获到异常就可以自动的回滚。但是我们发现异常在BaseDao中都被try…catch捕获了，如果直接throws向上抛异常那么它的父类也需要throws，比较麻烦！
  + 所以我们就需要在BaseDao中将所有捕获到SQLException转换为RuntimeException向上抛。
  + 当我们在BaseDao中将异常向上抛以后，又出现了这么一个异常：
    - java.lang.reflect.InvocationTargetException
    - 这个异常一般在通过反射调用一个方法时，当那个方法有异常时，会导致抛出该异常。
  + 我们在BaseServlet中通过反射调用Servlet中的方法，当方法有异常以后，又被BaseServlet捕获了，所以我们需要在BaseServlet中继续将异常向上抛。
  + 当我们在BaseServlet中将异常向上抛以后，的确出现变化了，但是页面并没有转到错误页面，也是说异常并没有被Filter捕获到，而是出现500，证明异常被服务器捕获了。
  + **原因**：我们在Filter中捕获的是SQLException，但是异常过来的时候是RuntimeExeption，所以我们Filter没有捕获该异常，**这里我们需要修改Filter捕获的异常类型为Exception。**

### 使用Ajax

* 注册页面使用Ajax验证用户名是否可用
* 修改购物车中图书数量使用Ajax