# 第一阶段 注册页面的验证

# 第二阶段 完成登录和注册的功能

## 软件架构

饭店的架构

服务员 厨师 采购

软件的三层架构

表示层（表现层、web层）

* 用户可以直接访问到的内容
* HTML页面 Servlet

业务逻辑层

* 业务逻辑层负责处理网站的一些业务逻辑（登录、注册、买书、结账）
* Service层

持久化层

* 负责操作数据库做增删改查等基本操作
* DAO层

数据库

持久化层

业务逻辑层

表示层

分层的目的就是解耦！！！

由于业务逻辑层和持久化层在开发过程中非常容易发生改变的，所以这两层我们采用一个面向接口的编程方式。

## 项目结构

## 开发步骤

1. 创建一个新的动态web工程。
2. 创建数据库bookstore\_0816
3. 创建表bs\_user
4. 创建类User
   1. Integer id
   2. String username
   3. String password
   4. String email
5. 导入第三方的jar包
   1. c3p0-0.9.1.2.jar
   2. commons-dbutils-1.3.jar
   3. mysql-connector-java-5.1.7-bin.jar
6. 导入c3p0配置文件
7. 创建JDBCUtils工具类
   1. Connection getConnection()
   2. void releaseConnection(Connection conn)
8. 创建BaseDao<T>
   1. int update()
   2. List<T> getBeanList()
   3. T getbBean()
9. 创建UserDao接口及实现类
   1. User getUserByUsernameAndPassword(User user)
   2. int saveUser(User user)
10. 创建UserService接口及实现类
    1. User login(User user)
    2. boolean regist(User user)
11. 创建LoginServlet和RegistServlet
12. 导入已经写好的页面
    1. 在页面中加入base标签
    2. 修改页面中的路径
    3. 修改表单的method和action属性

# 第三阶段 项目优化（修改html为jsp）

## 将所有的html修改为jsp

* 1. 新建一个jsp文件，并将html页面复制进新页面（主要page指令要保留）
  2. 在html页面中加入一个page指令，然后修改文件扩展名为jsp
  3. 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

## 登录和注册页面的错误消息显示

* 登录或注册失败以后再页面中显示错误消息
* 登录
  + 登录失败时也就是用户名或密码输错了，我们需要在域中设置一个错误消息然后在页面显示出来。

## 表单的回显

* 登录或注册失败以后，在表单中显示用户上次输入的信息

## 项目优化

* 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。
* 问题1思路：一个Serlvet可以处理多个请求，比如有一个UserServlet那么用户相关的请求都可交给该Servlet，比如登录对应一个login方法，注册对应一个regist方法。
  + 步骤：
    - 1.先创建一个UserServlet
    - 2.在Servlet中创建两个方法，一个是login 一个是regist
    - 3.当用户要登录时调用login 注册时调用regist
    - 4.我们希望用户可以告诉Servlet要掉login还是调regist
    - 5.我们通过用户传递method属性来判断要调用的方法。
  + 新问题：
    - 在这个UserServlet中每添加一个方法，就要添加一个与之对应if else语句，这样会导致if else很多。
  + 解决：
    - 如果可以直接根据方法名，获取到方法，我们就可以直接去调用方法，而不需要再手动调用
    - 通过反射动态的获取用户要调用的方法对象，然后在去调用这个方法，这样做的好处是，当在添加新的方法进Servlet时，不需要写if else，用户只需传递相应的方法名即可调用方法。
    - 注意：我们所定义的方法要和doGet和doPost方法结构一样，可以通过重写doGet方法然后改名的形式创建方法。
  + 新问题：
    - 我们将反射这一系列代码编写到了当前Servlet，那么我们每新创建一个Servlet这一堆代码就得在写一遍，我们的代码不能复用。
    - 解决：
      * 创建一个BaseServlet，将这些重复的代码在BaseServlet中编写，其他Servlet只需继承BaseServlet而不用再次编写重复的代码。
  + 注意：
    - 当我们使用get请求时，get请求会覆盖action中的请求参数
    - 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。

# 第四阶段 使用EL替换所有jsp表达式

* login.jsp
* regist.jsp
* base.jsp

# 第五阶段 图书的增删改查

## 第一部分：后台图书的增删改查

### 创建图书的类Book和图书的表bs\_book



Interger id

书名：String title

作者：String author

价格：double price

销量：int sales

库存：int stock

封面：String imgPath

### 创建BookDao接口及实现类

int saveBook(Book book) 🡪 向数据库中添加一本图书

int delBook(String bookId) 🡪 根据ID从数据库删除一本图书

int updateBook(Book book) 🡪 修改一本图书

List<Book> getBookList() 🡪 获取所有的图书

Book getBookById(String bookie) 🡪 根据id获取一本图书

### 创建BookService接口及实现类

int saveBook(Book book) 🡪 向数据库中添加一本图书

int delBook(String bookId) 🡪 根据ID从数据库删除一本图书

int updateBook(Book book) 🡪 修改一本图书

List<Book> getBookList() 🡪 获取所有的图书

Book getBookById(String bookie) 🡪 根据id获取一本图书

### 创建BookManagerServlet

addBook() 🡪 添加图书

delBook() 🡪 删除图书

updateBook() 🡪 修改图书

* 修改图书，首先需要将要修改的图书的信息在表单中回显，所以在修改之前需要先做查询操作，查询到要修改的图书，并在页面中进行回显。

bookList() 🡪 获取所有的图书的列表

getBookById() 🡪 获取一本的图书信息 \*\*\*

### 问题

* 目前我们使用SELECT \* FROM bs\_book 一下子将数据库中图书全部查询出来，这样如果数据库的图书较多的话，会使得性能变的非常非常的差。同时也会使得用户体验变得非常差。所以像上边那样的SQL我们在开发能不写就不写。
* 解决方案：
  + 我们可以不一次将所有的数据全都查出来，可以一次查询4条数据。
  + 这里我们就要为图书的显示做一个分页操作。
* 我们需要使用如下SQL语句，进行分页查询
  + SELECT \* FROM bs\_book LIMIT 开始的索引 , 每页显示的条数
  + 由于在页面加入了分页显示，所以之前我们使用List已经不能满足我们的需求了，这里我们就需要来创建一个新的类，来封装分页信息
  + Page



int pageSize

int totalRecord

int totalPage

int pageNumber

List<T> data

int index

* + Page<T>类
    - int pageNumber //当前页码 ， 该数据在Servlet中获取
    - int pageSize //每页显示的条数 ， 在Servlet中指定
    - int index // 分页开始的索引 ， 通过计算获得
    - int totalPage // 总页数 ， 需要通过计算获得
    - int totalRecord // 总记录 ， 通过查询数据库获得
    - List<T> data; // 分页的数据 ， 通过数据库查询
* 分页查询时，DAO、Service的分工
  + Service
    - pageNumber
    - pageSize
  + DAO
    - int totalRecord
    - List<T> data;
* 在BookDao中添加一个新的方法
  + Page<Book> findBook(Page<Book> page)
* 在BookService中添加一个新的方法
  + Page<Book> findBook(String pageNumber , int pageSize)
* 在BookManagerServlet中添加一个新的方法
  + findBook()
* 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

## 第二部分：前台页面图书的显示

* 前台的图书显示不需要在重写写DAO和Service而是直接去调用已有的方法。
* 只需要创建一个新的Servlet。
  + BookClientServlet
    - findBook() 分页查找图书
* 增加根据价格查找图书的功能
  + Sql：SELECT \* FROM bs\_book WHERE price>=10 AND price<=20 LIMIT 0 , 4
  + 步骤：
    - 在BookDao中添加一个新的方法
      * Page<Book> findBookByPrice(Page page , double minPrice , double maxPrice)
    - 在BookService中添加一个新的方法
      * Page<Book> findBookByPrice(String pageNumber , int pageSize , String min , String max)
    - 在BookClientServlet中添加一个新的方法
      * findBookByPrice()

# 第六阶段 登录、登出、注册验证码、购物车

## 1．完成登录功能

* 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。

## 2．用户的注销（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。

## 3．注册页面的验证码

* 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。

## 4．购物车

* 购物车是用来暂存咱们想要购买的商品。
* 购物车实现的三种方式：
  + 基于Cookie的，将购物的信息保存为Cookie发送给浏览器。
  + 基于Session的，将购物的信息保存到Session中。一旦关闭浏览器后，购物车里商品就没了。√
  + 基于数据库的，将购物的信息保存到一个数据库的表中。
* 创建一个用于封装购物车信息的类：



CartItem（购物项）

Cart（购物车）

我们的购物车是基于Session的，所以不用再编写DAO和Service。咱们可以将对购物车的操作方法定义在购物车类中。

* Cart
  + Map<String bookId , CartItem cartItem> map
  + int totalCount 🡪 购物车中商品的总数量
  + double totalAmount 🡪 购物车中的商品的总金额
  + 购物车操作的方法：
    - List<CartItem> getCartItems() 🡪 获取所有的购物项的列表
    - void delCartItem(String bookId) 🡪 删除一个购物项
    - void clear() 🡪 清空购物车中的所有商品
    - void addBook2Cart(Book book) 🡪 向购物车中添加一本图书
    - void updateCount(String bookId , String countStr) 🡪 修改某本图书的数量
* CartItem
  + Book book 🡪 图书的信息
  + int count 🡪　单品种图书的数量
  + double amount 🡪 单品种图书的金额
* 创建一个CartServlet
  + 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
  + updateCount() 🡪 修改购物车中的商品的数量

# 第七阶段 结账

* 将购物车中的商品信息存入数据库，并生成一个订单的信息。
* 结账的流程：



OrderDao

OrderItemDao

BookDao

OrderService

createOrder(Cart , User)

1. 读取购物车中的信息，将他们转换为一个订单，并插入进数据库。
2. 修改图书的销量和库存
3. 清空购物车
4. 返回订单号

OrderClientServlet

checkOut()

1. 检查用户是否登录
   1. 未登录，去登录页面登录
   2. 已登录，调用OrderService

## 创建用来封装订单信息的类

Order类

String id 🡪 订单号

Date orderTime 🡪 订单生成的时间

int totalCount 🡪 商品的总数量

double totalAmount 🡪 商品的总金额

int state 🡪 订单的状态 0未发货 1已发货 2交易完成

int user\_id 🡪 订单属于哪个用户

用户和订单的关系：一对多的关系

OrderItem订单项的信息

Integer id 🡪 订单项的编号

int count 🡪 当前图书数量

double amount 🡪 当前图书的金额

String title 🡪 书名

String author 🡪 作者

double price 🡪 书的单价

String imgPath 🡪 书的封面

String order\_id 🡪 订单项是属于哪个订单的

订单和订单项的关系：一对多的关系

订单号：

* + - 订单号确定订单的唯一。
    - 订单号用于订单的售后，要求订单号容易辨认。
    - 订单号要尽量少的透漏订单信息。
    - 订单号的规则：
      * 时间戳+””+用户的id

## 创建OrderDao及实现类

* + - int saveOrder(Order order); 🡪 保存订单信息
    - int updateState(String orderId , int state); 🡪 修改订单状态
    - List<Order> getOrderList() 🡪 查询所有订单，管理员调用
    - List<Order> getOrderListByUserId(String userId) 🡪 根据用户id查找订单，给普通用户调用

## 创建OrderItemDao及实现类

* + - int saveOrderItem(OrderItem orderItem) 🡪　向数据库中添加一个订单项
    - List<OrderItem> getOrderItemByOrderId(String orderId) 🡪 根据订单号查找订单项

## 生成订单的功能

* + - 创建一个OrderService接口及实现类
      * String createOrder(Cart cart , User user); 🡪 生成订单并返回订单号
    - 创建OrderClientServlet类
      * checkOut()

## 订单查询

* + - 在OrderService添加一个方法
      * List<Order> getOrderListByUserId(String userId);
      * List<Order> getOrderList();
    - 在OrderClientServlet中添加一个方法
      * orderList()
    - 创建一个OrderManagerServlet
      * orderList()

## 发货和确认收货

* + - 发货：将订单的状态修改为1
    - 确认收货: 将订单的状态修改为2
    - 在OrderService添加方法
      * void sendBook(String orderId) 🡪 修改订单状态为1已发货。管理员调
      * void takeBook(String orderId) 🡪 修改订单状态为2 交易完成。客户掉
    - 在OrderManagerServlet
      * sendBook() 🡪 发货
    - 在OrderClientServlet
      * takeBook() 🡪 收货

## 订单的详情的显示

* + - 查看订单详情，实际上就是来查看当前订单下的所有的订单项。
    - 在OrderService添加方法
      * List<OrderItem> getOrderInfo(String orderId);
    - 在OrderCLientServlet添加一个方法
      * orderInfo()

## 问题，修改项目为批量操作

* + 目前我们的createOrder方法，在for循环中去分别保存OrderItem和修改Book的效率和库存，也就是如果我们有10个订单项，那么整个过程我们要操作20次数据库，我们希望使用一个批量操作，一次性插入或修改全部数据。
  + 在BaseDao中添加支持批量操作的方法
    - batchUpdate()
  + 修改BookDao，添加一个方法
    - void batchUpdateSalesAndStock(Object[][] params) 批量修改销量和库存
  + 修改OrderItemDao，添加一个方法
    - void batchSave(Object[][] params); 批量插入订单项的信息
  + 改造createOrder方法

# 第八阶段 在项目中引入Filter

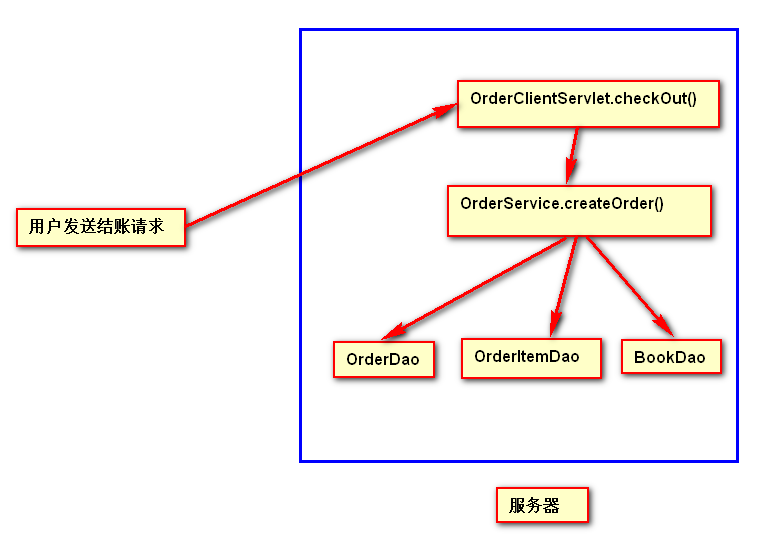
* 目前我们的登录验证时放在，OrderClientServlet的每一个方法中，我们业务要求只要访问OrderClientServlet就必须要登录，所有我们不得不在OrderClientServlet的每一个方法中都加上一个登录验证，这么做有点麻烦，所以我们想在项目中添加一个统一的登录验证。

## 在项目中加入统一的登录验证

* + LoginFilter
  + 过滤哪些请求？
    - 所有向OrderClientServlet发送的请求。/client/OrderClientServlet

## 事务的处理问题

* + 当我们将saveOrder的sql语句修改为一个错误的sql语句时，订单将不能正常生成。这时由于订单没有生成，创建OrderItem是由于找不到外键也会导致操作失败。但是，图书的销量和库存却被修改了。
  + 我们说，订单的生成，和订单项生成以及图书的销量及库存的修改，他们是同一个业务，我们在处理这个业务的时候，我们希望要么三个操作都成功，要么三个操作都失败。



* + 这里就涉及到事务的管理。
    - 常规的用法

|  |
| --- |
| //获取数据库连接  Connection conn = JDBCUtils.*getConnection*();    //设置事务手动提交  **try** {  conn.setAutoCommit(**false**);    //调用DAO去操作数据库    //如果没有异常，则提交事务  conn.commit();    } **catch** (SQLException e) {  e.printStackTrace();  //出现异常，回滚是否  **try** {  conn.rollback();  } **catch** (SQLException e1) {  // **TODO** Auto-generated catch block  e1.printStackTrace();  }    } **finally**{  JDBCUtils.*releaseConnection*(conn);  } |

* + - 这种方式要求，我们在service中对事务进行控制，还要求我们DAO的方法不能自己获取Connection对象，而是通过外部传进来。
    - 事务的关键就是一次业务中要求，所有的数据库操作都是使用的同一个数据库连接对象（Connection）。
    - 第一种思路：修改BaseDao，还要修改所有的其他的Dao以及所有的Service。
    - 第二种思路：能不能每一次获取的Connection对象都是相同的呢？可以为JDBCUtils修改为单例模式，也就是说我只有一个Connection对象。
      * 修改Connection对象为单例以后，要求Connection对象要统一获取，统一释放。
      * 创建一个TracsactionFilter来统一处理事务，用户每一次发送请求，首先会到达Filter，我们在Filter先获取数据库连接，并且将事务设置手动提交，然后放行请求，这样以后的业务中所使用到Connection都是同一个Connection我们只需要在Filter中对Connection进行统一处理，而不需要修改其他层了。
      * 我们已经在Filter中对Connection进行统一处理了，所以在BaseDao中就不需要再去关闭数据库连接。
      * 第一次尝试时，异常在BaseDao中被处理了，并没有向上抛出，所以在BaseDao将异常转换为了一个运行时异常并向上抛出。
      * 当将异常转换为一个运行时异常向上抛出时，Filter依然没有收到异常，而是控制台抛出了java.lang.reflect.InvocationTargetException，当我们通过反射去调用一个对象方法时，如果该方法抛出异常，会导致出现如下异常java.lang.reflect.InvocationTargetException。
      * 我们又在BaseServlet中对异常进行了捕获，所以异常还不能到的Filter，我们于是在BaseServlet中将异常转换为运行时异常继续向上抛。
      * 当将异常继续上抛时，Filter并没有处理异常，而是将异常又抛给了服务器，服务器给咱们来了一个500页面。产生这个问题的原因是，在Filter只捕获SQLException，而没有捕获其他的异常，所以RuntimeException最终会抛给服务器，这里我们只需要修改Filter的异常的捕获范围。
      * 采用这种方式来处理事务，当遇到并发的情况时，有可能在一个线程中Connection已经被关闭，而另一个线程还要使用，这种情况会导致抛出java.sql.SQLException: You can't operate on a closed Connection!!!。
      * 产生问题的原因，在于我傻傻的将Connection整成了单例了，也就是同一时刻只有一个Connection对象，这样会导致不同的业务之间会互相影响，所以单例不行！！！
    - 第三种思路：
      * 现在多个线程对应一个数据库连接不靠谱，那么就干脆一个线程对应一个数据库连接对象。问题：我们要将数据库连接保存到哪呢？
      * 我们发现线程和数据库连接是一个键值对结构，所以我们不妨将数据库连接保存到一个Map中，map的key是当前线程，值为数据库连接（Connection）。
      * 我们使用HashMap来保存Connection，但是HashMap不是线程安全的，虽然他比单例强，但是它同样也会出现线程安全问题，有么有一个map是线程安全的呢？

