**书城**

# 课程目标：

* 巩固前面学习的Java基础和JavaWeb的知识，学以至用
* 熟悉公司项目开发流程
* 掌握前后台分离思想

# 1前奏

## 开发文档介绍

## 系统环境

Eclipse Neno + JDK1.8 + Tomcat8.5 + MySQL5.7

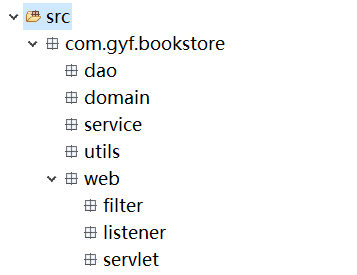
Eclipse Neno + JDK1.7 + Tomcat7.0 + MySQL5.7

## 数据库表创建

|  |
| --- |
| create database bookStore;  use bookStore;  #用户表  CREATE TABLE `user` (  `id` INT(11) AUTO\_INCREMENT,  `username` VARCHAR(20) ,  `PASSWORD` VARCHAR(20) ,  `gender` VARCHAR(10) ,  `email` VARCHAR(50) ,  `telephone` VARCHAR(20) ,  `introduce` VARCHAR(100),  `activeCode` VARCHAR(50) ,  `state` INT(11) ,  `role` VARCHAR(10) DEFAULT '普通用户',  `registTime` TIMESTAMP ,  PRIMARY KEY (`id`)  );  #商品表  CREATE TABLE `products` (  `id` VARCHAR(100) ,  `name` VARCHAR(40) ,  `price` DOUBLE ,  `category` VARCHAR(40) ,  `pnum` INT(11) ,  `imgurl` VARCHAR(100) ,  `description` VARCHAR(255) ,  PRIMARY KEY (`id`)  );  #订单表  CREATE TABLE `orders` (  `id` VARCHAR(100) ,  `money` DOUBLE ,  `receiverAddress` VARCHAR(255) ,  `receiverName` VARCHAR(20) ,  `receiverPhone` VARCHAR(20) ,  `paystate` INT(11) ,  `ordertime` TIMESTAMP ,  `user\_id` INT(11) ,  PRIMARY KEY (`id`),  FOREIGN KEY (`user\_id`) REFERENCES `user` (`id`)  );  #订单详情表  CREATE TABLE `orderitem` (  `order\_id` VARCHAR(100) ,  `product\_id` VARCHAR(100),  `buynum` INT(11) ,  PRIMARY KEY (`order\_id`,`product\_id`),  FOREIGN KEY (`order\_id`) REFERENCES `orders` (`id`),  FOREIGN KEY (`product\_id`) REFERENCES `products` (`id`)  ); |

## 包的划分

* 按照JavaEE 三层结构
* Java EE（Java Platform，Enterprise Edition）是sun公司（2009年4月20日甲骨文将其收购）推出的企业级应用程序版本。这个版本以前称为 J2EE。
* 模型：domain,model,pojo,po,entity



## jar包的导入

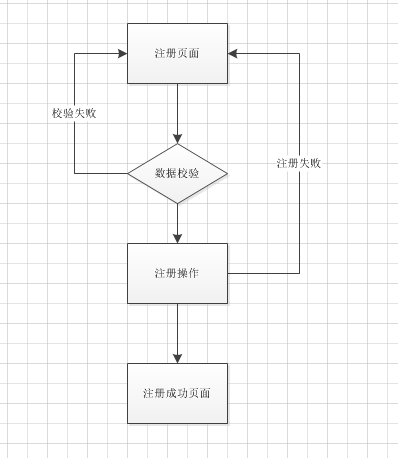
|  |
| --- |
| 导入mysql驱动  导入c3p0  导入dbutils  导入beanutils  导入fileupload  导入javamail mail.jar  导入jstl jstl.jar standard.jar |

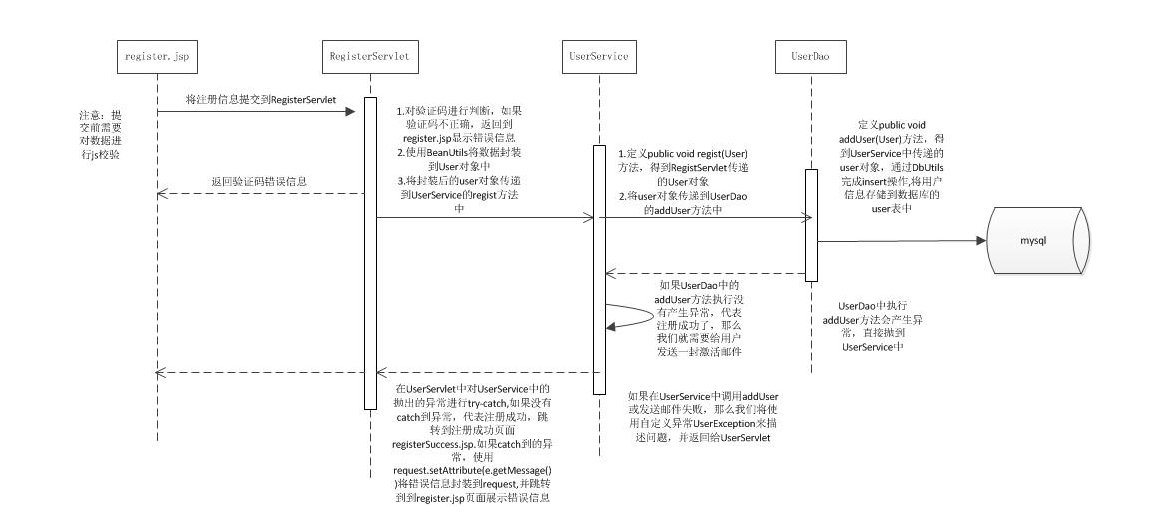
## 页面导入

# 2用户模块

## 2.1用户注册

### 流程图





### 步骤

#### Step1: User模型

|  |
| --- |
| **public** **class** User {  **private** **int** id; // 用户编号  **private** String username; // 用户姓名  **private** String password; // 用户密码  **private** String gender; // 用户性别  **private** String email; // 用户邮箱  **private** String telephone; // 用户联系电话  **private** String introduce; // 用户介绍  **private** String activeCode; // 激活码  **private** String role; // 用户角色  **private** **int** state; // 用户状态  **private** Date registTime;// 注册时间  } |

#### Step2:接收请求参数

|  |
| --- |
|  |

#### Step3:添加请求编码的Filter

|  |
| --- |
| public class MyEncodingFilter implements Filter{  @Override  public void init(FilterConfig filterConfig) throws ServletException {}  @Override  public void destroy() {}    @Override  public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain)  throws IOException, ServletException {  //1.设置POST请求中文乱码的问题  request.setCharacterEncoding("UTF-8");  System.out.println("拦截请求:" + request);    //2.解决get请求的中文乱码问题  //request ： RequestFacade;  HttpServletRequest hsr = (HttpServletRequest)request;  if(hsr.getMethod().equalsIgnoreCase("get")){  MyRequest myRequest = new MyRequest(hsr);  //放行请求  chain.doFilter(myRequest, response);  }else{  chain.doFilter(request, response);  }      }  }  /\*\*  \* Wrapper包装类，装饰设计模式,内部有个真实对象的引用  \* @author gyf  \*  \*/  class MyRequest extends HttpServletRequestWrapper{  private HttpServletRequest request;  private boolean isEncoding = false;//是否已经utf-8编码  public MyRequest(HttpServletRequest request) {  super(request);  this.request = request;  }    @Override  public String getParameter(String name) {  return getParameterMap().get(name)[0];  }    @Override  public Map<String, String[]> getParameterMap() {  Map<String, String[]> map = request.getParameterMap();    if(isEncoding == true){  return map;  }    //遍历vlaue，改成utf-8编码  for(Entry<String, String[]> entry : map.entrySet()){  //取数组值  String[] values = entry.getValue();  for(int i=0;i<values.length;i++){  try {  values[i] = new String(values[i].getBytes("ISO-8859-1"),"UTF-8");  } catch (UnsupportedEncodingException e) {  // TODO Auto-generated catch block  e.printStackTrace();  }  }  }  isEncoding = true;    return map;  }  } |

#### Step4:配置C3P0

|  |
| --- |
| c3p0-config.xml 文件放在src中 |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <c3p0-config>  <default-config>  <property name=*"driverClass"*>com.mysql.jdbc.Driver</property>  <property name=*"jdbcUrl"*>jdbc:mysql://localhost:3306/bookstore  </property>  <property name=*"user"*>root</property>  <property name=*"password"*>123456</property>  <!-- 初始化连接池数量 -->  <property name=*"initialPoolSize"*>10</property>    <!--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"*>60</property>  <!--连接池中保留的最大连接数。Default: 15 -->  <property name=*"maxPoolSize"*>25</property>    <!--连接池中保留的 最小连接数 -->  <property name=*"minPoolSize"*>10</property>  </default-config>  </c3p0-config> |
| C3P0Uitls.java 【copy以前的即可】 |
| **public** **class** C3P0Utils {    **private** **static** DataSource *ds* = **new** ComboPooledDataSource();  **public** **static** DataSource getDataSource(){  **return** *ds*;  }  **public** **static** Connection getConnection(){  **try** {  **return** *ds*.getConnection();  } **catch** (SQLException e) {  // **TODO** Auto-generated catch block  **throw** **new** RuntimeException("服务器错误");  }  }  **public** **static** **void** closeAll(Connection conn,Statement statement,ResultSet resultSet){  **if**(resultSet != **null**){  **try** {  resultSet.close();  } **catch** (SQLException e) {  // **TODO** Auto-generated catch block  e.printStackTrace();  }  resultSet = **null**;  }  **if**(statement != **null**){  **try** {  statement.close();  } **catch** (SQLException e) {  // **TODO** Auto-generated catch block  e.printStackTrace();  }  statement = **null**;  }  **if**(conn != **null**){  **try** {  conn.close();  } **catch** (SQLException e) {  // **TODO** Auto-generated catch block  e.printStackTrace();  conn = **null**;  }  }  }  } |

#### Step5:添加Service和Dao

|  |
| --- |
| UserDao.java |
|  |
| UserService |
|  |

#### Step6:修改RegisterServlet逻辑

SQL修改username性一:alter table user add constraint username\_uq unique (username);

|  |
| --- |
|  |

#### Step7:验证码

* 把CheckImgServlet添加到项目中，讲解下原理

|  |
| --- |
| 在Register前面添加验证码 |
|  |

## 2.2:激活邮件的发送

### smtp与pop3

* SMTP（Simple Mail Transfer Protocol）即简单邮件传输协议
* SMTP 服务器就是遵循 SMTP 协议的**发送邮件服务器**
* POP3是Post Office Protocol 3的简称，即邮局协议的第3个版本
* POP3服务器则是遵循POP3协议的**接收邮件服务器**

**SMTP是发送邮件协议**

**POP3是接收邮件协议**

### java如何发送邮件

* java中需要使用 这个包来发送邮件
* 帐号准备【注册一个163邮箱，以后公司会有自己的邮件服务器，到时改公司公司的即可】

帐号：15989566325@163.com

密码: gyfitedu

### 发送邮件代码

|  |
| --- |
| **package** com.gyf.bookstore.utils;  **import** java.util.Properties;  **import** javax.mail.Address;  **import** javax.mail.MessagingException;  **import** javax.mail.Session;  **import** javax.mail.Transport;  **import** javax.mail.internet.InternetAddress;  **import** javax.mail.internet.MimeMessage;  **public** **class** SendJMail {  **public** **static** **boolean** sendMail(String email, String emailMsg) {  String from = "15989566325@163.com"; // 邮件发送人的邮件地址  String to = email; // 邮件接收人的邮件地址  **final** String username = "15989566325@163.com"; //发件人的邮件帐户  **final** String password = "gyfitedu"; //发件人的邮件密码  //定义Properties对象,设置环境信息  Properties props = System.*getProperties*();  //设置邮件服务器的地址  props.setProperty("mail.smtp.host", "smtp.163.com"); // 指定的smtp服务器  props.setProperty("mail.smtp.auth", "true");  props.setProperty("mail.transport.protocol", "smtp");//设置发送邮件使用的协议  //创建Session对象,session对象表示整个邮件的环境信息  Session session = Session.*getInstance*(props);  //设置输出调试信息  session.setDebug(**true**);  **try** {  //Message的实例对象表示一封电子邮件  MimeMessage message = **new** MimeMessage(session);  //设置发件人的地址  message.setFrom(**new** InternetAddress(from));  //设置主题  message.setSubject("用户激活");  //设置邮件的文本内容  //message.setText("Welcome to JavaMail World!");  message.setContent((emailMsg),"text/html;charset=utf-8");    //设置附件  //message.setDataHandler(dh);    //从session的环境中获取发送邮件的对象  Transport transport=session.getTransport();  //连接邮件服务器  transport.connect("smtp.163.com",25, username, password);  //设置收件人地址,并发送消息  transport.sendMessage(message,**new** Address[]{**new** InternetAddress(to)});  transport.close();  **return** **true**;  } **catch** (MessagingException e) {  e.printStackTrace();  **return** **false**;  }  }  } |

### 在注册的service中，添加发送邮件功能

|  |
| --- |
|  |

## 2.3 激活用户

|  |
| --- |
| ActiveServlet:激活的Servlet |
|  |
| UserService:添加激活用户方法 |
|  |
| UserDao:添加2个方法 |
|  |

## 2.4 用户登录

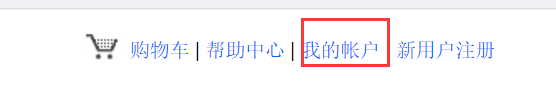
|  |
| --- |
| **UserDao.java** |
|  |
| **UserService.java** |
|  |
| **LoginServlet.java** |
|  |

## 2.5 权限管理

* 添加一个权限功能，登录成功后，如果是管理员，进行后台，如果是普通用户进入商品首页

|  |
| --- |
| LoginServlet.java |
|  |

## 2.6 我的帐号



|  |
| --- |
|  |

## 2.7 修改用户信息

### 显示用户信息

|  |
| --- |
|  |

|  |
| --- |
| FindUserByIdServlet |
|  |
| UserService |
|  |
| UserDao |
|  |
| modifyuserinfo.jsp |
| <input type=*"radio"* name=*"gender"* value=*"男"* ${u.gender == "男" ? "checked='checked'" : ''} />男  <input type=*"radio"* name=*"gender"* value=*"女"* ${u.gender == "女" ? "checked='checked'" : ''}/> 女 |

### 修改用户信息

|  |
| --- |
| ModifyUserInfoServlet.java |
|  |
| UserService |
|  |
| UserDao |
|  |

## 2.8用户退出

|  |
| --- |
|  |
|  |

# 商品

## 3.1 商品类别

|  |
| --- |
|  |

### 初始化产品数据

|  |
| --- |
| INSERT INTO products (id,name,price,category,pnum,imgurl,description)  VALUES (1,'Java入门基础-GYF',168.88,'计算机',3,null,'good book');  INSERT INTO products (id,name,price,category,pnum,imgurl,description)  VALUES (2,'HTML入门到精通-GYF',168.88,'计算机',3,null,'good book');  INSERT INTO products (id,name,price,category,pnum,imgurl,description)  VALUES (3,'JS入门到精通-GYF',168.88,'计算机',3,null,'good book');  INSERT INTO products (id,name,price,category,pnum,imgurl,description)  VALUES (4,'PHP入门到精通-GYF',168.88,'计算机',3,null,'good book');  INSERT INTO products (id,name,price,category,pnum,imgurl,description)  VALUES (5,'Android入门到精通-GYF',168.88,'计算机',3,null,'good book');  INSERT INTO products (id,name,price,category,pnum,imgurl,description)  VALUES (6,'iOS入门到精通-GYF',168.88,'计算机',3,null,'good book');  INSERT INTO products (id,name,price,category,pnum,imgurl,description)  VALUES (7,'MySQL入门到精通-GYF',168.88,'计算机',3,null,'good book');  INSERT INTO products (id,name,price,category,pnum,imgurl,description)  VALUES (8,'Oracle入门到精通-GYF',168.88,'计算机',3,null,'good book');  INSERT INTO products (id,name,price,category,pnum,imgurl,description)  VALUES (9,'VMWare入门到精通-GYF',168.88,'计算机',3,null,'good book');  INSERT INTO products (id,name,price,category,pnum,imgurl,description)  VALUES (11,'武炼巅峰',88,'文学',3,null,'good book');  INSERT INTO products (id,name,price,category,pnum,imgurl,description)  VALUES (12,'绝世药神',8.88,'文学',3,null,'good book');  INSERT INTO products (id,name,price,category,pnum,imgurl,description)  VALUES (13,'王牌狙击之霸宠狂妻',8.88,'文学',3,null,'good book');  INSERT INTO products (id,name,price,category,pnum,imgurl,description)  VALUES (14,'非我倾城：王爷要休妃',8.88,'文学',3,null,'good book');  INSERT INTO products (id,name,price,category,pnum,imgurl,description)  VALUES (15,'超品相师',88,'文学',3,null,'good book');  INSERT INTO products (id,name,price,category,pnum,imgurl,description)  VALUES (16,'最强弃少',88,'文学',3,null,'good book');  INSERT INTO products (id,name,price,category,pnum,imgurl,description)  VALUES (17,'折锦春',88,'文学',3,null,'good book');  INSERT INTO products (id,name,price,category,pnum,imgurl,description)  VALUES (18,'金陵春',16,'文学',3,null,'good book');  INSERT INTO products (id,name,price,category,pnum,imgurl,description)  VALUES (19,'总裁的新妻',16,'文学',3,null,'good book');  INSERT INTO products (id,name,price,category,pnum,imgurl,description)  VALUES (21,'武炼巅峰2',88,'文学',3,null,'good book');  INSERT INTO products (id,name,price,category,pnum,imgurl,description)  VALUES (22,'绝世药神2',8.88,'文学',3,null,'good book');  INSERT INTO products (id,name,price,category,pnum,imgurl,description)  VALUES (23,'王牌狙击之霸宠狂妻2',8.88,'文学',3,null,'good book');  INSERT INTO products (id,name,price,category,pnum,imgurl,description)  VALUES (24,'非我倾城：王爷要休妃2',8.88,'文学',3,null,'good book');  INSERT INTO products (id,name,price,category,pnum,imgurl,description)  VALUES (25,'超品相师2',88,'文学',3,null,'good book');  INSERT INTO products (id,name,price,category,pnum,imgurl,description)  VALUES (26,'最强弃少2',88,'文学',3,null,'good book');  INSERT INTO products (id,name,price,category,pnum,imgurl,description)  VALUES (27,'折锦春2',88,'文学',3,null,'good book');  INSERT INTO products (id,name,price,category,pnum,imgurl,description)  VALUES (28,'金陵春2',16,'文学',3,null,'good book');  INSERT INTO products (id,name,price,category,pnum,imgurl,description)  VALUES (29,'总裁的新妻2',16,'文学',3,null,'good book'); |

### 添加产品模型

|  |
| --- |
| **public** **class** Product {  **private** **int** id;  **private** String name;//书名  **private** **double** price;//价格  **private** String category;//分类  **private** **int** pnum;//数量  **private** String imgurl;//图片  **private** String description;//描述  } |

### ProductDao

|  |
| --- |
| **public** **class** ProductDao{  //查询类别的总记录数  **public** **long** count(String category) **throws** SQLException{  QueryRunner qr = **new** QueryRunner(C3P0Utils.*getDataSource*());  String sql = "select count(\*) from products where 1=1";    **long** count = 0;  **if**(category != **null** && !"".equals(category)){  sql+= " and category = ?";  count = (**long**) qr.query(sql, **new** ScalarHandler(1), category);  }**else**{  count = (**long**) qr.query(sql, **new** ScalarHandler(1));  }    **return** count;  }    //分页查找数据  **public** List<Product> findBooks(**int** currentPage,**int** pageCount,String category) **throws** SQLException{  QueryRunner qr = **new** QueryRunner(C3P0Utils.*getDataSource*());  //定义参数  List<Object> prmts = **new** ArrayList<>();    //查询的sql语句  String sql = "select \* from products where 1=1";    //条件  **if**(category!=**null** && !"".equals(category)){  sql += " and category = ?";  prmts.add(category);  }    //分页  sql += " limit ?,?";  **int** start = (currentPage - 1) \* pageCount;  prmts.add(start);  prmts.add(pageCount);    **return** qr.query(sql, **new** BeanListHandler<Product>(Product.**class**), prmts.toArray());  }  } |

### ProductService

|  |
| --- |
| **public** **class** ProductService {  ProductDao productDao = **new** ProductDao();  **public** PageResult<Product> findPageBooks(**int** currentPage,**int** pageCount,String category){  **try** {  //1.创建PageResult  PageResult<Product> pr = **new** PageResult<Product>();    //2.获取总记录数  **long** totalCount = productDao.count(category);    //3.计算总页数  **int** totalPage = (**int**) Math.*ceil*(totalCount \* 1.0 / pageCount) ;    //4.查询数据库  List<Product> list = productDao.findBooks(currentPage, pageCount, category);    //5.设置PageResult  pr.setCurrentPage(currentPage);//当前页  pr.setPageCount(pageCount);//每页显示记录数  pr.setTotalCount(totalCount);//总记录数  pr.setTotalPage(totalPage);//总页数  pr.setList(list);    **return** pr;  } **catch** (SQLException e) {  // **TODO**: handle exception  e.printStackTrace();  }  **return** **null**;  }  } |

### ShowProductByPageServlet

|  |
| --- |
| @WebServlet("/showProductByPage")  **public** **class** ShowProductByPageServlet **extends** HttpServlet{  ProductService productService = **new** ProductService();  @Override  **protected** **void** doGet(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {  // **TODO** Auto-generated method stub  //获取参数  String category = request.getParameter("category");//分类  String page = request.getParameter("page");//显示页数    //2.判断  **int** currentPage = 1;  **int** pageCount = 4;//每页显示5条数据,这个内部定义，不让外面传参数  **if**(page != **null**){  currentPage = Integer.*parseInt*(page);  }      //3.调用service  PageResult<Product> pr = productService.findPageBooks(currentPage, pageCount, category);    //4.跳转  request.setAttribute("pr", pr);  request.setAttribute("category", category);  request.getRequestDispatcher("/product\_list.jsp").forward(request, response);  }  } |

### PageResult

* 提供get/set方法

|  |
| --- |
| **public** **class** PageResult<T> {  **private** List<T> list;//集合  **private** **long** totalCount;//总记录数  **private** **int** totalPage;//总页数  **private** **int** currentPage;//当前页  **private** **int** pageCount = 5;//每页显示的条数 |

### Product\_list.html

|  |
| --- |
| 部分截图 |
|  |
|  |

## 3.2 添加图书图片显示

## 3.3 添加图书信息查看

|  |
| --- |
|  |

|  |
| --- |
| ProductDao |
|  |
| ProductService |
|  |
| ProductInfoservlet |
|  |
| Productinfo.jsp |
|  |

## 3.4 添加购物车

|  |
| --- |
|  |

|  |
| --- |
| AddCartServet.java |
| @WebServlet("/addCart")  **public** **class** AddCartServlet **extends** HttpServlet{  @Override  **protected** **void** doGet(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {  //1.根据id查找图书  String id = request.getParameter("id");  ProductService ps = **new** ProductService();  Product book = ps.findBookById(id);    //2.从session中获取购物车信息  @SuppressWarnings("unchecked")  Map<Product,Integer> cart = (Map<Product, Integer>) request.getSession().getAttribute("cart");    //3.如果是空，创建一个购物边集合  **if**(cart == **null**){  cart = **new** HashMap<Product,Integer>();  }    //4.判断购物车中是否有当前书的记录  **if**(cart.containsKey(book)){  cart.put(book, cart.get(book) + 1);  }**else**{  cart.put(book, 1);  }  System.***out***.println("购物车信息:" + cart);    //5.重新保存到session中  request.getSession().setAttribute("cart", cart);    //6.继续购物或者结算查看购物车  String html = "<a href='"+ request.getContextPath() +"/showProductByPage'>继续购物</a>";  html += "&nbsp;&nbsp;&nbsp;&nbsp;<a href='" + request.getContextPath() + "/cart.jsp'>查看购物车</a>";  response.getWriter().write(html);  }  } |

## 3.5 显示购物车

从Session中获取数据

|  |
| --- |
|  |
| cart.jsp |
| <table width=*"100%"* border=*"0"* cellspacing=*"0"*>  <!-- 先存一个总价格为0的变量 -->  <c:set var=*"totalPrice"* value=*"0"*></c:set>  <!-- 遍历购物车 -->  <c:forEach items="${cart}" var=*"entry"* varStatus=*"vs"*>  <tr>  <td width=*"10%"*>${vs.count}</td>  <td width=*"30%"*>${entry.key.name}</td>  <td width=*"10%"*>${entry.key.price}</td>  <td width=*"20%"*>  <input type=*"button"* value=*'-'*  style="width:*20px*">  <input name=*"text"* type=*"text"* value=*"*${entry.value}*"*  style="width:*40px*;text-align:*center*" /> <input  type=*"button"* value=*'+'* style="width:*20px*">  </td>  <td width=*"10%"*>${entry.key.pnum}</td>  <td width=*"10%"*>${entry.key.price \* entry.value}</td>  <td width=*"10%"*><a href=*"#"*  style="color:*#FF0000*; font-weight:*bold*">X</a></td>  </tr>    <!-- 累加价格 -->  <c:set var=*"totalPrice"* value="${totalPrice + entry.key.price \* entry.value}"></c:set>  </c:forEach>  </table> |

## 3.6 更改购物车信息JS

|  |
| --- |
| Cart.jsp |
|  |
| Js  <script type=*"text/javascript"*>  /\* id : 产品id,书的id  num:更改的数量  totalNumber:产品总数量 \*/  **function** changeNum(id, num, totalNumber) {  console.log('图书ID:' + id + ' 购买数量:' + num + '总量:' + totalNumber);    //当购买的数据大于总商品数量时  **if**(num > totalNumber){  alert('不能购买大于库存的数据');  **return**;  }    //当商品数量等于0时，给个提醒  **if**(num < 1){  **if**(!confirm('是否要删除此商品')){//删除此商品  **return**;  }  }    console.log('执行操作....');  //重新访问后台，更改session的数据  location.href = '${pageContext.request.contextPath}/changeNum?id=' + id + '&num=' + num;    }  </script> |
| ChangeNumServlet.java |
| @WebServlet("/changeNum")  **public** **class** ChangeNumServlet **extends** HttpServlet{  @Override  **protected** **void** doGet(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {  // **TODO** Auto-generated method stub  //1.获取请求参数  String id = request.getParameter("id");  String num = request.getParameter("num");    **if**(id == **null** && num == **null**){  response.getWriter().write("非法访问");**return**;  }    //2.只需要封装个对象，设置个ID即可  ProductService ps = **new** ProductService();  Product book = ps.findBookById(id);  **if**(book == **null**){  response.getWriter().write("非法访问");**return**;  }    @SuppressWarnings("unchecked")  Map<Product,Integer> cart = (Map<Product, Integer>) request.getSession().getAttribute("cart");  **if**(cart == **null**){  response.getWriter().write("非法访问");**return**;  }    **if**(num.equals("0")){  cart.remove(book);//把这个产品从session中移除  }**else**{  **if**(cart.containsKey(book)){//更改数量  cart.put(book, Integer.*parseInt*(num));  }  }    request.setAttribute("cart", cart);  request.getRequestDispatcher("cart.jsp").forward(request, response);  }  } |

## 3.7 结算功能

|  |
| --- |
|  |
| 结帐Servlet |
|  |
| Order.jsp |
|  |

## 3.8 下单

**注意点：**

**把之前写的ManagerThreadLocal拿过来用**

**下单成功后，把session的购物车清空**

### Order.jsp

|  |
| --- |
| **Order.jsp的表单通过js提交** |
|  |
|  |

### CreateOrderServlet

|  |
| --- |
| Order order = **new** Order();  //获取当前登录用户的信息  User user = (User) request.getSession().getAttribute("user");    **try** {  //1.把请求通过封闭到Order中  BeanUtils.*populate*(order, request.getParameterMap());    //2.设置用户  order.setUser(user);    //3.设置ID  order.setId(UUID.*randomUUID*().toString());    //4.设置定单详情  @SuppressWarnings("unchecked")  Map<Product,Integer> cart = (Map<Product, Integer>) request.getSession().getAttribute("cart");    List<OrderItem> orderItems = **new** ArrayList<OrderItem>();  **for**(Entry<Product,Integer> entry : cart.entrySet()){  OrderItem oi = **new** OrderItem();  oi.setBuynum(entry.getValue());  oi.setP(entry.getKey());  oi.setOrder(order);  orderItems.add(oi);  }  order.setOrderItems(orderItems);    //5.设置定单时间  order.setOrdertime(**new** Date());    //6.订单保存  OrderService os = **new** OrderService();  os.addOrder(order);  //7.进入支付页面  request.getRequestDispatcher("/pay.jsp").forward(request, response);  } **catch** (Exception e) {  e.printStackTrace();  } |

### OrderService

|  |
| --- |
| **public** **class** OrderService {  OrderDao orderDao = **new** OrderDao();  OrderItemDao orderItemDao = **new** OrderItemDao();  ProductDao productDao = **new** ProductDao();  /\*\*  \* 保存定单  \* **@param** order  \*/  **public** **void** addOrder(Order order){  //1.开启事务  ManagerThreadLocal.*beginTransaction*();  **try** {  //2.保存定单表  orderDao.saveOrder(order);  //3.保存订单详情表  **for**(OrderItem oi : order.getOrderItems()){  orderItemDao.saveOrderItem(oi);  //更改产品可卖数量  productDao.updateProductNum(oi.getP().getId(),oi.getBuynum());  }  //4.结束事务  ManagerThreadLocal.*commitTransaction*();  } **catch** (Exception e) {  e.printStackTrace();  //5.回滚  ManagerThreadLocal.*rollback*();  }  } } |

### OrderDao

|  |
| --- |
|  |

### OrderItemDao

|  |
| --- |
|  |

### ProductDao

|  |
| --- |
|  |

**定单详情的另一种批处理**

|  |
| --- |
| /\*\*  \* 批量保存订单详情  \* **@param** items  \* **@throws** SQLException  \*/  **public** **void** saveOrderItems(List<OrderItem> items) **throws** SQLException{  //1.封装参数  Object[][] params = **new** Object[items.size()][];  **for**(**int** i=0;i<items.size();i++){  OrderItem oi = items.get(i);  params[i] = **new** Object[]{oi.getOrder().getId(),oi.getP().getId(),oi.getBuynum()};  }    //2.批处理  String sql = "insert into orderitem values (?,?,?)";  QueryRunner qr = **new** QueryRunner();  qr.batch(ManagerThreadLocal.*getConnection*(), sql,params );  } |

## 3.9查看定单信息

### OrderDao

|  |
| --- |
|  |

### OrderService

|  |
| --- |
|  |

### FindOrderByIdServlet

|  |
| --- |
|  |

### Orderlist.jsp

|  |
| --- |
|  |

## 3.10 查看定单详情

### OrderDao

|  |
| --- |
| /\*\*  \* 根据定单ID查询定单和详情信息  \*/  **public** Order findOrderByOrderId(String orderId)**throws** SQLException {  // **TODO** Auto-generated method stub  QueryRunner qr = **new** QueryRunner(C3P0Utils.*getDataSource*());    //1.查询定单表  String sql = "select \* from orders where id = ?";  Order order = qr.query(sql, **new** BeanHandler<Order>(Order.**class**),orderId);    //2.查询定单下面的详情  sql = "SELECT o.\*,p.name,p.price FROM orderitem o, products p WHERE o.product\_id = p.id AND o.order\_id = ?";  List<OrderItem> orderItems = qr.query(sql, **new** ResultSetHandler<List<OrderItem>>(){    @Override  **public** List<OrderItem> handle(ResultSet rs) **throws** SQLException {  // **TODO** Auto-generated method stub  List<OrderItem> items = **new** ArrayList<OrderItem>();  **while**(rs.next()){  //封装定单详情  OrderItem item = **new** OrderItem();  item.setBuynum(rs.getInt("buynum"));    //封装产品【只获取商品名称和价格】  Product p = **new** Product();  p.setName(rs.getString("name"));  p.setPrice((rs.getDouble("price")));    //把产品放入详情  item.setP(p);    //把详情添加到集合  items.add(item);  }  **return** items;  }    },orderId);    order.setOrderItems(orderItems);    **return** order;  } |

### OrderService

|  |
| --- |
| /\*\*  \* 通过定单号返回定单及详细信息  \*/  **public** Order findOrderByOrderId(String orderId){  **try** {  **return** orderDao.findOrderByOrderId(orderId);  } **catch** (SQLException e) {  // **TODO** Auto-generated catch block  e.printStackTrace();  }  **return** **null**;  } |

### FindOrderItemsByOrderIdServlet

|  |
| --- |
| @WebServlet("/findOrderItemsByOrderId")  **public** **class** FindOrderItemsByOrderIdServlet **extends** HttpServlet{  @Override  **protected** **void** doGet(HttpServletRequest request, HttpServletResponse response)  **throws** ServletException, IOException {  //获取请求参数  String orderid = request.getParameter("orderid");    //查询定单  OrderService os = **new** OrderService();  Order order = os.findOrderByOrderId(orderid);    //存数据到request  request.setAttribute("order", order);  request.getRequestDispatcher("/orderInfo.jsp").forward(request, response);  //System.out.println(orderid);  }  } |

### orderInfo.jsp

|  |
| --- |
|  |

# 用户模块优化

## 优化一

将相同的模块的功能抽取到一个Servlet中【画图讲解】

|  |
| --- |
| @WebServlet("/user/\*")  **public** **class** UserServlet **extends** HttpServlet {  **private** **static** **final** **long** ***serialVersionUID*** = 1L;  @Override  **protected** **void** doPost(HttpServletRequest request, HttpServletResponse response)  **throws** ServletException, IOException {  // **TODO** Auto-generated method stub  // 1.获取action参数  String action = request.getParameter("action");  System.***out***.println("action:" + action);  **if** (action.equals("login")) {  login(request, response);  } **else** **if** (action.equals("register")) {  register(request, response);  } **else** **if**(action.equals("logout")){  logout(request, response);  }**else**{  response.getWriter().write("不合法访问");  }  }  @Override  **protected** **void** doGet(HttpServletRequest request, HttpServletResponse response)  **throws** ServletException, IOException {  // **TODO** Auto-generated method stub  doPost(request, response);  }  } |
| 访问时就以下面的方式访问  http://locahost:8080/bookstore/user?action=login  http://locahost:8080/bookstore/user?action=register  http://locahost:8080/bookstore/user?action=logout |

## 优化二

抽取BaseServlet

|  |
| --- |
| **public** **class** BaseServlet **extends** HttpServlet{  @Override  **protected** **void** service(HttpServletRequest request, HttpServletResponse response)  **throws** ServletException, IOException {    //1.方法名  String action = request.getParameter("action");    **try** {  //2.通过反制获取方法名  Method method = **this**.getClass().getDeclaredMethod(action, HttpServletRequest.**class**,HttpServletResponse.**class**);    //3.调用方法  method.invoke(**this**,request,response);  } **catch** (NoSuchMethodException e) {  e.printStackTrace();  }**catch** (SecurityException e) {  e.printStackTrace();  }**catch** (IllegalAccessException e) {  e.printStackTrace();  } **catch** (IllegalArgumentException e) {  e.printStackTrace();  } **catch** (InvocationTargetException e) {  e.printStackTrace();  }  }  } |
|  |

UserServlet只需要继承baseservlet

这样做，不需要像第一种那样过多的方法名判断

|  |
| --- |
|  |

# 管理员与普通用户的权限

|  |
| --- |
| //只拦截admin路径下的请求  @WebFilter("/admin/\*")  **public** **class** RoleFilter **implements** Filter{  @Override  **public** **void** init(FilterConfig filterConfig) **throws** ServletException {}  @Override  **public** **void** doFilter(ServletRequest req, ServletResponse resp, FilterChain chain)  **throws** IOException, ServletException {  // **TODO** Auto-generated method stub  HttpServletRequest request = (HttpServletRequest) req;  HttpServletResponse response = (HttpServletResponse) resp;  User user = (User) request.getSession().getAttribute("user");    **if**(user != **null**){  **if**(!"管理员".equals(user.getRole())){  response.getWriter().write("权限不足，没有管理员权限,使用管理员帐号登录");  response.setHeader("Refresh", "3;url="+ request.getContextPath()+"/index.jsp");  **return**;  }**else**{  //放行  chain.doFilter(request, response);  **return**;  }    }    //进入登录界面  response.sendRedirect(request.getContextPath()+"/login.jsp");  }  @Override  **public** **void** destroy() {}  } |