# **JavaWeb综合实战—微商城**

## 项目功能分析与设计

此次将使用最原始的开发技术进行WEB基础项目的开发，本次的项目使用的技术：HTML + JavaScript + CSS + JSP + Servlet + DAO + EL + JSTL + MySQL + SmartUpload，本次不使用任何的设计。

在整个项目实战的过程之中，除了要实现功能之外，希望思考一个问题：最原始的实现技术是很纯粹的但是会存在哪些问题？

此次只完成核心的操作功能：商品的管理、浏览、库存、用户及激活、购物车、订单。

### 1.1、需求描述

此次实现的是一个微型的在线商城，那么整个程序功能描述如下：

1. 前台使用的用户：
   1. 用户需要进行注册，注册的时候需要准备出所有的核心信息；

用户ID(email)、密码、用户状态（未激活、锁定、正常）、姓名、电话、联系地址；

* 1. 用户可以进行商品的分类浏览，现在只分一类；

·分类信息：分类编号、名称；

·商品信息：编号、名称、价格、库存量、商品的描述、图片、访问量；

* 1. 用户可以进行订单的创建，一个订单可以购买多个商品信息，同一件商品可以购买多次；
  2. 用户可以查阅自己的订单信息。

1. 后台管理员：
   1. 一定要准备出管理员的信息表，这个信息表要求记录如下内容：

管理员ID、密码、最后一次登录日期时间；

* 1. 可以进行商品的管理，但是所有的商品可以设置状态：

可以设置商品的商家与下架操作

商品信息可以删除，并且要求第一次删除的时候被放到回收站中；

可以清空回收站中的全部商品，或删除一个商品信息。

* 1. 可以针对于分类进行管理；
  2. 可以对用户进行管理，包括用户的锁定问题；
  3. 可以对所有的订单进行管理。

实际中还有可能包含的功能：

·库存量低时的预警操作；

·管理员的登录日志与权限问题；

·用户的登录日志与预警问题；

·一个用户的地址信息管理；

·订单的处理状态；

·退货所造成的库存变更问题；

。。。。。。。

### 1.2、数据库设计

对于数据库的设计肯定要参考设计范式，但是在进行设计的过程之中还有一个核心原则，根据最终的显示结果，可以使用一些冗余字段来消除掉多表查询问题。对于设计范式：

·第一设计范式：单表设计使用核心类型（int、double、varchar、text、datetime）；

·第二设计范式：多对多关系；

·第三设计范式：一对多关系。

1、用户需要进行注册，注册的时候需要准备出所有的核心信息；

·用户ID(email)、密码、用户状态（未激活、锁定、正常）、姓名、电话、联系地址；

这应该使用第一设计范式，没有任何的关系，只是一张数据表



这个时候用户ID一定是由注册用户自己输入的，所以这个时候在进行注册的时候一定要考虑到ID不能重复。

密码最终一定要通过MD5进行数据加密。

用户状态使用了一个整型数据，因为数据库不支持不二型数据，但是对于这俄格整型数据有如下的几个取值；

·用户状态（status = 0）：用户锁定状态，此时的用户无法进行登录；

·用户状态（status = 2）：用户未激活，此时用户无法进行登陆；

·用户状态（status = 1）：用户可以正常使用。

2、用户可以进行商品的分类浏览，现在只分一类；

一个分类下可以有多个商品，一个商品属于一个分类，这是一个典型的一对多关系应用



商品图片如果没有设置使用“nophoto.jpg”描述。

对于访问量一定是每次浏览的时候进行自增操作；

在每次生产订单之后可以修改库存量。

对于商品状态有如下几种：

·上架（status = 1）：表示现在可以正常进行浏览；

·下架（status = 0）：表示商品现在处于下架状态；

·删除（status = 2）：表示此商品已经被删除了，但是还可以在回收站中找到。

·彻底删除（status = ）：表示此商品信息逻辑删除，另外一种做法是彻底删除信息(建议不要彻底删除)；

3、用户可以进行订单的创建，一个订单可以购买多个商品信息，同一件商品可以购买多次；

任何一个订单都需要包含有以下内容：订单编号、联系人、电话、地址、总价钱，但是一个订单会包含有多件商品，而且每件商品可以购买多个。一个用户可以有多个订单，但是每一个订单需要保存多个详情内容。

1. 进行管理员的信息定义；

管理员负责商品的发布。



### 1.3、数据库的创建脚本

1、根据表结构定义脚本；

|  |
| --- |
| -- 删除数据库 **DROP** DATABASE IF **EXISTS** mshop ; -- 创建数据库 **CREATE** DATABASE mshop **CHARACTER SET** UTF8 ; --使用mshop数据库 USE mshop ; |
| -- 删除数据表 **DROP TABLE** IF **EXISTS** details ; **DROP TABLE** IF **EXISTS** orders ; **DROP TABLE** IF **EXISTS** member ; **DROP TABLE** IF **EXISTS** goods ; **DROP TABLE** IF **EXISTS** item ; **DROP TABLE** IF **EXISTS** admin ; |
| -- 创建数据表 -- 1、创建商品类型表 **CREATE TABLE** item (  iid **INT** AUTO\_INCREMENT ,  title **VARCHAR**(200) **NOT NULL** ,  **CONSTRAINT** pk\_iid **PRIMARY KEY**(iid) ) **type**=innodb ;  -- 2、创建管理员信息表 **CREATE TABLE** admin (  aid **VARCHAR**(50) ,  password **VARCHAR**(32) ,  lastdate DATETIME ,  **CONSTRAINT** pk\_aid **PRIMARY KEY**(aid) ) **type**=innodb ;  -- 3、创建用户信息表 **CREATE TABLE** member (  mid **VARCHAR**(20) ,  password **VARCHAR**(32) **NOT NULL** ,  **name VARCHAR**(50) ,  phone **VARCHAR**(50) ,  address **VARCHAR**(100) ,  code **VARCHAR**(100) ,  regdate DATETIME **NOT NULL** ,  photo **VARCHAR**(50) **DEFAULT** 'nophoto.jpg',  **CONSTRAINT** pk\_mid **PRIMARY KEY**(mid) ) **type**=innodb ;  -- 4、创建商品信息表 **CREATE TABLE** goods (  gid **INT** AUTO\_INCREMENT ,  iid **INT** ,  aid **VARCHAR**(50) ,  title **VARCHAR**(50) ,  pubdate DATETIME ,  price **FLOAT** ,  amount **INT** ,  bow **INT** ,  note TEXT ,  photo **VARCHAR**(100) ,  status **INT** ,  **CONSTRAINT** pk\_gid **PRIMARY KEY**(gid) ,  **CONSTRAINT** fk\_iid **FOREIGN KEY**(iid) **REFERENCES** item(iid) **ON DELETE SET NULL** ,  **CONSTRAINT** fi\_aid **FOREIGN KEY**(aid) **REFERENCES** admin(aid) **ON DELETE SET NULL** ) **type**=innodb ;  -- 5、创建订单信息表 **CREATE TABLE** orders (  oid **INT** AUTO\_INCREMENT ,  mid **VARCHAR**(50) ,  **name VARCHAR**(50) ,  phone **VARCHAR**(50) ,  address **VARCHAR**(100) ,  credate DATETIME ,  pay **FLOAT** ,  **CONSTRAINT** pk\_oid **PRIMARY KEY**(oid) ,  **CONSTRAINT** fk\_mid **FOREIGN KEY**(mid) **REFERENCES** member(mid) **ON DELETE CASCADE** -- 如果用户被删除，订单没有必要被保存. ) **type**=innodb ;  -- 6、创建订单详情表 **CREATE TABLE** details (  odid **INT** AUTO\_INCREMENT ,  oid **INT NOT NULL** ,  gid **INT** ,  title **VARCHAR**(50) **NOT NULL** ,  price **FLOAT NOT NULL** ,  amount **INT NOT NULL** ,  **CONSTRAINT** pk\_odid **PRIMARY KEY** (odid) ,  **CONSTRAINT** fk\_oid **FOREIGN KEY**(oid) **REFERENCES** orders(oid) **ON DELETE CASCADE** ,  **CONSTRAINT** fk\_gid **FOREIGN KEY**(gid) **REFERENCES** goods(gid) **ON DELETE SET NULL** ) **type**=innodb ; |

其中“type=innodb”表示可以执行事务，优势第三方插件不支持使用该语句

2、编写测试数据

|  |
| --- |
| -- 增加商品分类信息 **INSERT INTO** item(title) **VALUES**('厨房用品') ; **INSERT INTO** item(title) **VALUES**('儿童玩具') ; **INSERT INTO** item(title) **VALUES**('医疗器械') ; **INSERT INTO** item(title) **VALUES**('运动健身') ; **INSERT INTO** item(title) **VALUES**('办公用品') ; |
| -- 增加管理员信息: admin / admin **INSERT INTO** admin(aid,password) **VALUES** ('admin','21232F297A57A5A743894A0E4A801FC3') ; |
| -- 增加普通用户信息：root / 123456 **INSERT INTO** member(mid,password,regdate) **VALUES** ('root','E10ADC3949BA59ABBE56E057F20F883E','2018-04-05') ; |
| -- 提交事务 **COMMIT** |

其他的信息内容比较多，准备测试数据会比较麻烦。

### 1.4、配置开发包

#### 1.4.1、配置程序中的jar文件

1、整个项目里面需要配置的jar文件：JSTL（两个）、SmartUpload、Servlet-api、MySQL的驱动程序。

2、由于项目中需要使用到JSTL，所以要在web.xml文件里面配置JSTL的标签引用名称。

|  |
| --- |
| <jsp-config>  <taglib>  <taglib-uri>http://www.mshop.cn/c</taglib-uri>  <taglib-location>/WEB-INF/c.tld</taglib-location>  </taglib> </jsp-config> <welcome-file-list>  <welcome-file>index.jsp</welcome-file> </welcome-file-list> |

3\准备好Servlet的开发包

#### 1.4.2、设置Java程序文件

1、在之前建立数据库的时候已经准备好了MD5Code代码；

2、建立一个cn.shop.dao包，将IDAO接口拷贝到此包之中；

3、建立一个cn.shop.dbc包，将DatabaseConnection程序拷贝到此包之中；

数据库的名称为mshop。

#### 1.4.3、设置web目录

对于WEB目录依然需要有一个css、js、images目录，而且必须要有一个index.jsp的1、首页，随后还需要准备出pages目录，但是此目录划分如下：

·pages

|- back：保存所有的后台文件；

|- login.jsp：定义登陆页面 ；

|- admin：保存所有需要被后台管理的页面；

|- index.jsp：定义显示的菜单；

|- member：管理用户；

|- goods：管理商品；

|- item：管理商品类型；

|-orders：管理订单。

|- front：保存所有的前台文件；

|- goods：指的是商品信息的浏览操作页面；

|- member：指的是用户信息的操作；

|- orders：定义所有的订单信息的操作；

2、随后建立一个upload目录，但是现在对于图片的上传要分为两种情况：

·用户的图片，保存在member目录下；

·商品的图片，保存在goods目录下。

3、由于项目里面需要进行分页控制，将分页程序的代码拷贝到pages目录下。

4、定义一个模板页面，名称为others.jsp；

|  |
| --- |
| <%@ **page** contentType="text/html;charset=UTF-8" language="java" %> <%@ **page** pageEncoding="utf-8"%> <%@**taglib** prefix="c" uri="http://www.mshop.cn/c"%> **<%** String path = request.getContextPath() ;  String basePath = request.getScheme() + "://"  + request.getServerName() + ":" + request.getServerPort()  + path + "/" ; **%>** <html> <head>  <base href="**<%**basePath**%>**">  <title>微商城综合实战</title>  <link type="text/css" rel="stylesheet" href="css/hello.css">  <script type="text/javascript" src="js/hello.js"></script> </head> <body> </body> </html> |

5、建立pages/forward.jsp页面，用于进行信息提示。

|  |
| --- |
| <%@ **page** contentType="text/html;charset=UTF-8" language="java" %> <%@ **page** pageEncoding="utf-8"%> <%@**taglib** prefix="c" uri="http://www.mshop.cn/c"%> **<%** String path = request.getContextPath() ;  String basePath = request.getScheme() + "://"  + request.getServerName() + ":" + request.getServerPort()  + path ; **%>** <html> <head>  <base href="**<%**basePath**%>**">  <title>微商城综合实战</title>  <link type="text/css" rel="stylesheet" href="css/hello.css">  <script type="text/javascript" src="js/hello.js"></script> </head> <body> <script type="text/javascript">  window.alert("**${**msg**}**") ;  window.location = "**<%**basePath**%>${**url**}**" ; </script> </body> </html> |

6、建立errors.jsp页面，负责信息显示；

7、所有的代码一定都需要处理乱码，所以建立一个过滤器，专门处理乱码。

直接在cn.util.filter下建立。

|  |
| --- |
| package cn.util.filter;  import javax.servlet.\*; import javax.servlet.annotation.WebFilter; import java.io.IOException; @WebFilter(filterName = "encodingFilter",urlPatterns = "/\*") public class EncodingFilter implements Filter {  @Override  public void init(FilterConfig filterConfig) throws ServletException {  }   @Override  public void doFilter(ServletRequest servletRequest, ServletResponse servletResponse, FilterChain filterChain) throws IOException, ServletException {  servletRequest.setCharacterEncoding("UTF-8");  filterChain.doFilter(servletRequest,servletResponse);  servletResponse.setCharacterEncoding("UTF-8");  }   @Override  public void destroy() {  } } |

这个程序最大的好处在于以后直接复制过来可以使用了。

### 1.5、用户注册与管理模块

本次开发功能肯定是前台和后台一起共同完成，开发流程：根据功能需要编写业务层方法或者是数据层方法。

#### 1.5.1、用户的注册

如果要想是想用户的注册功能，那么需要提供的数据就是用户名、密码。其中对于用户名不能够重复，而密码在注册的时候也不能够重复。

##### 1.5.1.1、开发业务层

###### （1）开发Member.java的VO类

|  |
| --- |
| package cn.shop.vo;  import java.util.Date;  public class Member implements java.io.Serializable {  private String mid ;  private String password ;  private String name ;  private String phone ;  private String address ;  private String code ;  private Integer status;   private Date regdate ;  private String photo ; } |

###### （2）定义IMemberDAO接口；

在用户组测的过程之中，一定需要检查mid是否存在，这个操作可以利用findById()方法实现。

###### （3）对于所有的DAO实现子类操作实际上都可以编写一些公共代码。

·在cn.util.dao下建立一个AbstractDAOImpl子类；

|  |
| --- |
| package cn.util.dao; import java.sql.Connection; import java.sql.PreparedStatement; import java.sql.ResultSet; import java.util.Iterator; import java.util.Set; public abstract class AbstractDAOImpl {  protected Connection conn ;  protected PreparedStatement pstmt ;  public AbstractDAOImpl(Connection conn) {  this.conn = conn ;  }  */\*\*  \* 实现数据的批量删除，这个时候的批量删除属于彻底删除功能  \** ***@param*** *table 表名称  \** ***@param*** *column 删除表的列名称  \** ***@param*** *ids 所有的id数据，使用set集合可以避免重复  \** ***@return*** *如果删除成功返回true，否则返回false  \** ***@throws*** *Exception  \*/* public boolean rezmoveHandle(String table, String column, Set<?> ids) throws Exception {  if (ids.size() == 0){ //表示现在没有任何的数据  return false ;  }  StringBuffer buf = new StringBuffer() ;  buf.append("DELETE FROM ").append(table).append(" WHERE ").append(column).append(" IN (")  Iterator<?> iter = ids.iterator() ;  while (iter.hasNext()) {  buf.append(iter.next()).append(",") ;  }  buf.delete(buf.length() - 1 , buf.length()).append(")") ;  this.pstmt = this.conn.prepareStatement(buf.toString()) ;  return this.pstmt.executeUpdate() == ids.size() ;  }  */\*\*  \* 负责统计出数据量  \** ***@param*** *table 要统计数据的表名称  \** ***@param*** *column 模糊查询数据列  \** ***@param*** *keyWord 模糊查询数据量  \** ***@return*** *返回指定表的数据量，如果没有数据，返回0 ；  \** ***@throws*** *Exception  \*/* public Integer countHandle(String table, String column, String keyWord) throws Exception {  StringBuffer sql = new StringBuffer() ;  sql.append("SELECT COUNT(\*) FROM ").append(table).append(" WHERE ").append(column).append(" LIKE ? ") ;  this.pstmt = this.conn.prepareStatement(sql.toString()) ;  this.pstmt.setString(1,"%" + keyWord + "%") ;  ResultSet rs = this.pstmt.executeQuery() ;  if (rs.next()) {  return rs.getInt(1);  }  return 0 ;  } } |

任何的项目里面都一定会存在有逻辑删除的操作，以后可以针对于这个工具类进行一些工具的扩充。

·在定义MemberDAOImpl子类的时候多继承一个父类。但是这个时候里面的方法暂时不需要全部都覆写，只需要覆写doCreate()、findById()两个方法就够了

|  |
| --- |
| public boolean doCreate(Member vo) throws Exception {  String sql = "INSERT INTO member(mid,password,code,regdate,status) VALUES(?,?,?,?,?)" ;  super.pstmt = super.conn.prepareStatement(sql) ;  super.pstmt.setString(1,vo.getMid());  super.pstmt.setString(2,vo.getPassword());  super.pstmt.setString(3,vo.getCode());  super.pstmt.setTimestamp(4,new Timestamp(vo.getRegdate().getTime()));  super.pstmt.setInt(5,vo.getStatus());  super.pstmt.setString(6,vo.getPhoto());  return super.pstmt.executeUpdate() > 0; } |
| @Override public Member findById(String id) throws Exception {  Member vo = null ;  String sql = "SELECT mid,password,name,phone,address,code,status,regdate,photo FROM member WHERE mid=?" ;  super.pstmt = super.conn.prepareStatement(sql);  super.pstmt.setString(1,id);  ResultSet rs = super.pstmt.executeQuery() ;  if (rs.next()) {  vo = new Member() ;  vo.setMid(rs.getString(1));  vo.setPassword(rs.getString(2));  vo.setName(rs.getString(3));  vo.setPhone(rs.getString(4));  vo.setAddress(rs.getString(5));  vo.setCode(rs.getString(6));  vo.setStatus(rs.getInt(7));  vo.setRegdate(rs.getTimestamp(8));  }  return vo; } |

（4）随后需要定义DAOFactory.java类

|  |
| --- |
| package cn.shop.factory; import cn.shop.dao.IMemberDAO; import cn.shop.dao.impl.MemberDAOImpl; import java.sql.Connection; public class DAOFactory {  public static IMemberDAO getIMemberDAOInstance(Connection conn) {  return new MemberDAOImpl(conn) ;  } } |

###### （5）定义前台业务层代码。

|  |
| --- |
| package cn.shop.service.front; import cn.shop.vo.Member; public interface IMemberServiceFront {  */\*\*  \* 实现用户的注册操作，注册时要提供有mid，password，regdate，code，status；<br>  \* 本程序将调用如下操作：<br>  \* <li>调用IMemberDAO.findById()方法判断注册mid是否存在。</li>  \* <li>调用IMemberDAO.doCreate()方法保存基本信息</li>  \** ***@param*** *vo 包含有注册信息的VO类对象  \** ***@return*** *注册成功返回true，否则返回false  \** ***@throws*** *Exception  \*/* public boolean regist(Member vo) throws Exception ; } |

###### （6）定义实现子类

|  |
| --- |
| package cn.shop.service.front.Impl; import cn.shop.dbc.DatabaseConnection; import cn.shop.factory.DAOFactory; import cn.shop.service.front.IMemberServiceFront; import cn.shop.vo.Member; public class MemberServiceFrontImpl implements IMemberServiceFront {  private DatabaseConnection dbc = new DatabaseConnection() ;  @Override  public boolean regist(Member vo) throws Exception {  try {  if (DAOFactory.*getIMemberDAOInstance*(this.dbc.getConnection()).findById(vo.getMid()) == null) {  return DAOFactory.*getIMemberDAOInstance*(this.dbc.getConnection()).doCreate(vo) ;  }  return false;  } catch (Exception e) {  throw e ;  } finally {  this.dbc.close();  }  } } |

###### （7）定义前台业务层的工厂类

|  |
| --- |
| package cn.shop.factory; import cn.shop.service.front.IMemberServiceFront; import cn.shop.service.front.Impl.MemberServiceFrontImpl; public class ServiceFrontFactory {  public static IMemberServiceFront getIMemberServiceFrontInstance() {  return new MemberServiceFrontImpl() ;  } } |

程序只提供有注册的功能。

##### 1.5.1.2、开发前台

前台就是JSP + Servlet程序（99%的情况下都是多业务处理）。

###### 开发MemberServletFront程序类；

|  |
| --- |
| package cn.shop.servlet.front; import javax.servlet.ServletException; import javax.servlet.annotation.WebServlet; import javax.servlet.http.HttpServlet; import javax.servlet.http.HttpServletRequest; import javax.servlet.http.HttpServletResponse; import java.io.IOException; @WebServlet(name = "MemberSservletFront",urlPatterns = "/pages/MemberServletFront/\*") public class MemberServletFront extends HttpServlet {  @Override  public void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  String path = "/pages/errors.jsp" ;  String status = request.getRequestURI().substring(request.getRequestURI().lastIndexOf("/") + 1) ;  if (status != null) {  if ("regist".equals(status)) {  path = this.regist(request) ;  }  }  request.getRequestDispatcher(path).forward(request,response);  }  @Override  public void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  this.doGet(request,response);  }  public String regist(HttpServletRequest request) {  return "/pages/forward.jsp" ;  }  } |

###### （2）验证建立工具类

所有的服务器端接收到数据之后严格来讲都必须进行验证，所以现在专门准备一个可以进行验证的工具类，此类的验证主要是判断数据是否为null，以及数据是否是指定的正则格式，如果有验证码，则可以在进行验证码的检查，那么可以在，cn.util.validate包下建立要给叫ValidateUtil.java的工具类，这个类设计的时候重点考虑与WEB的集合。

·分析以下的两种情况：

|  |  |  |
| --- | --- | --- |
| **定义：** | public static boolean validateEmpty(HttpServletRequest request, String paramName) {  String value = request.getParameter(paramName) ; } | public static boolean validateEmpty(String data) { } |
| **调用：** | ValidateUtil.*validateEmpty*(request,"mid") ; | ValidateUtil.*validateEmpty*(request.getParameter("mid")) ; |

严格来讲以上的两种方案作用一样，但是第二种方案会好，因为第二种方案不会与WEB联系紧密。

|  |
| --- |
| package cn.util.validate; import javax.servlet.http.HttpServletRequest; public class ValidateUtil {  */\*\*  \* 验证输入的数据是否为空  \** ***@param*** *data 输入数据  \** ***@return*** *如果不是null返回true，否则返回false  \*/* public static boolean validateEmpty(String data) {  if (data == null || "".equals(data)) {  return false ;  }  return true ;  }  */\*\*  \* 进行数据的正则操作验证  \** ***@param*** *data 要验证的数据  \** ***@param*** *regex 要执行验证的正则表达式  \** ***@return*** *如果验证通过返回true，否则返回false  \*/* public static boolean validateRegex(String data, String regex) {  if (*validateEmpty*(data)) { // 数据不为空  return data.matches(regex) ; // 验证通过  }  return false ;  }  */\*\*  \* 验证两个数据是否相同，不区分大小写  \** ***@param*** *dataa 数据一  \** ***@param*** *datab 数据二  \** ***@return*** *如果相同返回true，否则返回false  \*/* public static boolean validateSame(String dataa, String datab) {  if (*validateEmpty*(dataa) && *validateEmpty*(datab)) {  return dataa.equalsIgnoreCase(datab) ;  }  return false ;  } } |

###### （3）完善regist()方法

|  |
| --- |
| public String regist(HttpServletRequest request) {  String msg = null ;  String url = null ; // 执行forward.jsp的使用  String mid = request.getParameter("mid") ;  String password = request.getParameter("password") ;  // 必须验证通过，才可以进行后续的功能  if (ValidateUtil.*validateEmpty*(mid) && ValidateUtil.*validateEmpty*(password)) {  Member vo = new Member() ; // 要将数据封装在VO类对象里  vo.setMid(mid);  vo.setPassword(new MD5Code().*GetMD5Code*(password)); // md5加密  vo.setRegdate(new Date()); // 今天为注册日期  vo.setPhoto("nophoto.jpg");  vo.setCode(UUID.*randomUUID*().toString()); // 生成一个随机的Code码  vo.setStatus(2); // 现在属于待激活状态  try {  if (ServiceFrontFactory.*getIMemberServiceFrontInstance*().regist(vo)) {  msg = "用户注册成功，请进行账户激活！" ;  url = "index.jsp" ;  System.*out*.println("【发激活邮件】" + BasePath.*getBasePath*(request) + "/pages/member\_active.jsp?mid" + mid + "&code" + vo.getCode());  } else {  msg = "用户注册失败，请填写新的注册信息！" ;  url = "/pages/member\_regist.jsp" ;  }  } catch (Exception e) {  e.printStackTrace();  }  } else {  msg = "输入的用户注册信息不正确，请重新注册！" ;  url = "/pages/member\_regist.jsp" ;  }  request.setAttribute("msg", msg);  request.setAttribute("url", url);  return "/pages/forward.jsp" ; } |

###### （4）用户注册页面

在pages目录下建立member\_regist.jsp页面，而后在根目录下同时建立一个index.jsp页面。但是现在必须要考虑一个情况，所有的程序站点都有头部信息和尾部信息。

·建立一个header.jsp页面，保存头部信息；

|  |
| --- |
| <div id="headerDiv">  <img src="kuangsan.jpg">  <a href="">商城首页</a>  <a href="">我的购物车</a>  <a href="">个人中心</a>  <a href="">登陆</a>  <a href="">注册</a> </div> |

·建立一个footer.jsp页面，保存尾部信息；

|  |
| --- |
| <div id="footerDiv">  Created by IntelliJ IDEA.  User: MM  Date: 2018/4/6  Time: 9:05 </div> |

·建立member\_regist.jsp页面

|  |
| --- |
| String registUrl = basePath + "/pages/MemberServletFront/regist" ; |
| <script type="text/javascript" src="js/member.js"></script> |

随后注册成功后会给出激活信息。

###### （5）用户激活

用户激活的时候只需要判断给出的Code是否相同，如果相同，直接修改status的内容即可。

1. 在IMemberDAO接口里面增加两个操作方法：一个用于判断mid与code值是否相同，另外一个是更新status的状态内容。

|  |
| --- |
| package cn.shop.dao; import cn.shop.vo.Member; public interface IMemberDAO extends IDAO<String,Member> {  */\*\*  \* 判断给定的mid与给定的code是否相同  \** ***@param*** *mid 要激活的用户id  \** ***@param*** *code 设置好的激活码  \** ***@return*** *如果用户id与激活码相匹配则返回true，否则返回false  \*/* public boolean findByCode(String mid, String code) throws Exception ;  */\*\*  \* 更新指定用户的状态操作  \** ***@param*** *mid 用户id  \** ***@param*** *status 用户状态（0表示锁定、1表示激活、2表示待激活）  \** ***@return*** *\** ***@throws*** *Exception  \*/* public boolean doUpdateStatus(String mid, Integer status) throws Exception ; } |

1. 实现操作子类

|  |
| --- |
| public boolean findByCode(String mid, String code) throws Exception {  String sql = "SELECT *COUNT*(\*) FROM member WHERE mid=? AND code=? AND status=2" ;  super.pstmt = super.conn.prepareStatement(sql);  super.pstmt.setString(1,mid);  super.pstmt.setString(2,code);  ResultSet rs = super.pstmt.executeQuery() ;  if (rs.next()) {  if (rs.getInt(1) > 0) {  return true  }  }  return false; } |
| @Override public boolean doUpdateStatus(String mid, Integer status) throws Exception {  String sql = "UPDATE member SET status=? WHERE mid=?" ;  super.pstmt = super.conn.prepareStatement(sql) ;  super.pstmt.setInt(1,status);  super.pstmt.setString(2,mid);  return super.pstmt.executeUpdate() > 0; } |

3、定义业务层中的激活操作

|  |
| --- |
| */\*\*  \* 实现注册用户的激活操作，只有激活后的用户才可以登录，本操作要调用如下功能：<br>  \* <li>调用IMemberDAO.findByCode()方法,判断用户的激活码是否正确！</li>  \* <li>调用IMemberDAP.doUpdateStatus()方法修改激活码</li>  \** ***@param*** *vo 包含有mid与code的数据  \** ***@return*** *激活成功返回true，否则返回false  \** ***@throws*** *Exception  \*/* public boolean active(Member vo) throws Exception ; |
| @Override public boolean active(Member vo) throws Exception {  try{  if (DAOFactory.*getIMemberDAOInstance*(this.dbc.getConnection()).findByCode(vo.getMid(),vo.getCode())) {  return DAOFactory.*getIMemberDAOInstance*(this.dbc.getConnection()).doUpdateStatus(vo.getMid(),1) ;  }  return false ;  }catch (Exception e){  throw e ;  } finally {  this.dbc.close();  } } |

4、在MemberServletFront类里面扩充active()方法

|  |
| --- |
| public String active(HttpServletRequest request) {  String msg = null ;  String url = null ;  String mid = request.getParameter("mid") ;  String code = request.getParameter("code") ;  if (ValidateUtil.*validateEmpty*(mid) && ValidateUtil.*validateEmpty*(code)) {  Member vo = new Member() ;  vo.setMid(mid) ;  vo.setCode(code);  try {  if (ServiceFrontFactory.*getIMemberServiceFrontInstance*().active(vo)) {  msg = "用户激活成功，请登录！" ;  url = "/pages/member\_login.jsp" ;  } else {  msg = "用户激活失败，请与管理员联系！" ;  url = "/index.jsp" ;  }  } catch (Exception e) {  e.printStackTrace();  }  } else {  msg = "错误的激活操作，请与管理员联系！" ;  url = "/index.jsp" ;  }   return "/pages/forwoard.jsp" ;  } |

5、为了保证激活成功之后页面不会显示404，所以建立好member\_login.jsp页面；

#### 1.5.2、用户的登录

如果要想登录，肯定要验证用户名和密码的信息，同时还应该显示出用户的照片信息，所以用户登录后，要求找出用户的照片信息；

##### 1.5.2.1、在IMemberDAO接口里面定义一个登录的操作代码

|  |
| --- |
| */\*\*  \* 用户的登录检查操作，登录合法之后可以查询出用户的照片信息，  \* 由于参数接收的是Member对象，所以可以直接将照片信息保存在Member对象  \** ***@param*** *vo 包含了mid与password的VO类对象  \** ***@return*** *登录成功返回true，否则返回false  \** ***@throws*** *Exception  \*/* public boolean findLogin(Member vo) throws Exception ; |
| public boolean findLogin(Member vo) throws Exception {  boolean flag = false ;  // 必须保证用户的登陆状态是1（正常状态）  String sql = "SELECT photo FROM member WHERE mid=? AND password=? AND status=1" ;  super.pstmt = super.conn.prepareStatement(sql) ;  super.pstmt.setString(1,vo.getMid()) ;  super.pstmt.setString(2,vo.getPassword()) ;  ResultSet rs = super.pstmt.executeQuery() ;  if (rs.next()) {  flag = true ;  vo.setPhoto(rs.getString(1));  }  return flag; |

##### 1.5.2.2、业务层开发

业务层就定义一个login()方法，但这个方法返回boolean；

|  |
| --- |
| */\*\*  \* 用户登陆操作，调用IMemberDAO.findLogin()方法，只能登录激活的状态  \** ***@param*** *vo 包含有mid与password的VO类对象  \** ***@return*** *登陆成功返回true，否则返回false  \** ***@throws*** *Exception  \*/* public boolean login(Member vo) throws Exception ; |
| @Override public boolean login(Member vo) throws Exception {  try {  return DAOFactory.*getIMemberDAOInstance*(this.dbc.getConnection()).findLogin(vo) ;  } catch (Exception e) {  throw e ;  } finally {  this.dbc.close() ;  } } |

##### 1.5.2.3、在Servlet中增加登陆的操作功能

在MemberServletFront.java类里面增加一个登录的操作方法；

|  |
| --- |
| public String login(HttpServletRequest request) {  String msg = null ;  String url = null ;  String mid = request.getParameter("mid") ;  String password = request.getParameter("password") ;  String code = request.getParameter("code") ;  String rand = (String) request.getSession().getAttribute("rand") ;  if (ValidateUtil.*validateEmpty*(mid) && ValidateUtil.*validateEmpty*(password) && ValidateUtil.*validateEmpty*(code) && ValidateUtil.*validateEmpty*(rand)) {  if (ValidateUtil.*validateSame*(code,rand)) {  Member vo = new Member() ;  vo.setMid(mid);  vo.setPassword(new MD5Code().*GetMD5Code*(password));  try {  if (ServiceFrontFactory.*getIMemberServiceFrontInstance*().login(vo)) {  request.getSession().setAttribute("mid",mid);  request.getSession().setAttribute("photo",vo.getPhoto());  msg = "登录成功，欢迎光临！" ;  url = "/index.jsp" ;  } else {  msg = "登陆失败，错误的用户名或密码！" ;  url = "/pages/member\_login.jsp" ;  }  } catch (Exception e) {  e.printStackTrace();  }  } else {  msg = "验证码输入错误，请重新输入！" ;  url = "/pages/member\_login.jsp" ;  }  } else {  msg = "信息输入错误，请重新输入！" ;  url = "/pages/member\_login.jsp" ;  }  request.setAttribute("msg",msg);  request.setAttribute("url",url);  return "/pages/forward.jsp" ; } |

##### 1.5.2.4、编写member\_login.jsp页面；

|  |
| --- |
| String loginUrl = basePath + "pages/MemberServletFront/login" ; |

##### 1.5.2.5、头部信息修改

既然都已经登陆成功了那么头部信息的位置就应该发生一些变化，至少登陆与注册链接不应该出现。

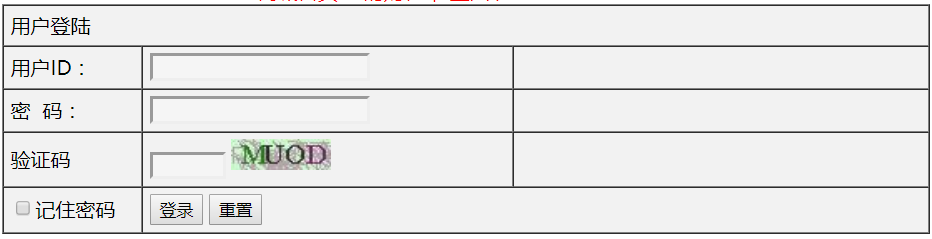
|  |
| --- |
| <**c:if** test="**${**mid == **null}**">  <a href="pages/member\_login.jsp">登陆</a>  <a href="pages/member\_regist.jsp">注册</a> </**c:if**> <**c:if** test="**${**mid != **null}**">  <a href="">个人中心</a>  <<img src="upload/member/**${**photo**}**" style="width: 50px;height: 50px;"> </**c:if**> |

如果用户没有登陆过，才应该出现相应的链接提示，如果登陆过了，那么应该出现其他的界面

#### 1.5.3、记住密码

如果每一次用户的时候都要进行一个登录操作，那么实在是太不方便了，所以最好的做法是利用Cookie，将用户名和密码保存在客户端（加密后的内容），而后当用户每一次访问站点时，如果发现Session中的内容为null，则可以通过Cookie找到用户名和密码，而后自动进行登陆检查。

##### 1.5.3.1、修改member\_login.jsp页面，这个页面增肌啊一个选项；



##### 1.5.3.2、Cookie的操作是利用response对象完成的，而取得Cookie通过request对象完成。

如果要是登陆成功了，则应该将用户名和密码保存在Cookie之中，可以专门使用一个类来完成这个功能。

|  |
| --- |
| package cn.util; import javax.servlet.http.Cookie; import javax.servlet.http.HttpServletRequest; import javax.servlet.http.HttpServletResponse; public class CookieUtil {  */\*\*  \* 定义一个专门负责保存Cookie的方法，可以设置Cookie的名字与保存时间  \** ***@param*** *response 主要可以使用addCookie()方法保存Cookie对象  \** ***@param*** *request 主要设置Cookie的保存路径，如果不设置保存不上  \** ***@param*** *name 要保存的Cookie名字  \** ***@param*** *value 要保存的Cookie的内容  \** ***@param*** *expiry Cookie的失效时间  \*/* public static void save(HttpServletResponse response, HttpServletRequest request, String name, String value, int expiry) {  Cookie c = new Cookie(name, value) ;  c.setMaxAge(expiry);  **c.setPath(request.getContextPath());** // 设置保存路径  response.addCookie(c);  } } |
| if (request.getParameter("reme") != null) { // 表示选择了复选框  int expiry = Integer.*parseInt*(request.getParameter("reme")) ;  CookieUtil.*save*(response, request, "mid", mid, expiry);  CookieUtil.*save*(response, request, "password", vo.getPassword(), expiry); } |

##### 1.5.3.3、取出Cookie

用户可能访问本站点的任意页面，那么在任意的页面访问之前都应该先去取出Cookie所以最好的做法是利用过滤器完成Cookie的取得，同时自动实现验证。

·定义一个取出全部Cookie的操作方法。

|  |
| --- |
| public static Map<String, String> load(HttpServletRequest request) {  Map<String, String> map = new HashMap<String, String>() ;  Cookie c [] = request.getCookies() ;  for (int x = 0 ; x < c.length ; x ++) {  map.put(c[x].getName(), c[x].getValue()) ;  }  return map ; } |

使用Map是为了方便查询

·定义一个过滤器，实现自动登录的检查。

|  |
| --- |
| package cn.util; import javax.servlet.http.Cookie; import javax.servlet.http.HttpServletRequest; import javax.servlet.http.HttpServletResponse; import java.util.HashMap; import java.util.Map; public class CookieUtil {  */\*\*  \* 定义一个专门负责保存Cookie的方法，可以设置Cookie的名字与保存时间  \** ***@param*** *response 主要可以使用addCookie()方法保存Cookie对象  \** ***@param*** *request 主要设置Cookie的保存路径，如果不设置保存不上  \** ***@param*** *name 要保存的Cookie名字  \** ***@param*** *value 要保存的Cookie的内容  \** ***@param*** *expiry Cookie的失效时间  \*/* public static void save(HttpServletResponse response, HttpServletRequest request, String name, String value, int expiry) {  Cookie c = new Cookie(name, value) ;  c.setMaxAge(expiry);  c.setPath("/"); // 设置保存路径  response.addCookie(c);  }  public static Map<String, String> load(HttpServletRequest request) {  Map<String, String> map = new HashMap<String,String>() ;  Cookie [] c = request.getCookies();  if (c != null) {  for (int x = 0; x < c.length; x++) {  map.put(c[x].getName(), c[x].getValue());  }  }  return map ;  } } |

##### 1.5.3.4、提供一个登录注销的功能。

这个功能要清除所有的session数据。

|  |
| --- |
| <a href="/pages/MemberServletFront/logout">安全退出</a>> |

·既然又是与Cookie有关的操作，那么直接交给CookieUtil类完成即可

|  |
| --- |
| public static void clear(HttpServletRequest request, HttpServletResponse response) {  Map<String,String> map = *load*(request) ;  Iterator<Map.Entry<String,String>> iter = map.entrySet().iterator() ;  while (iter.hasNext()) {  Map.Entry<String,String> me = iter.next() ;  Cookie c = new Cookie((me.getKey()),"") ;   c.setPath("/");  c.setMaxAge(0);  response.addCookie(c);  } } |

·随后就可以调用session中的invalidate()方法取消掉当前的session信息。

|  |
| --- |
| public String logout(HttpServletRequest request , HttpServletResponse response) {  CookieUtil.*clear*(request,response) ;  request.getSession().invalidate();  request.setAttribute("msg","您已安全退出！");  request.setAttribute("url","/index.jsp");  return "/pages/forward.jsp" ; } |

关于记住密码的操作一定要记住，基本上都要结合过滤器一起完成。

### 1.6、管理员登陆

在进行管理员登陆的时候需要保存一个最后一次登录日期，而这一操作就属于更新admin表的操作。

#### 1.6.1、定义管理员操作的DAO程序

·定义Admin.java的VO类；

|  |
| --- |
| package cn.shop.vo; import java.io.Serializable; import java.util.Date; public class Admin implements Serializable {  private String aid ;  private String password ;  private Date lastdate ; } |

·在定义IAdminDAO接口的时候，这个接口需要扩充一个登陆操作方法以及一个更新最后一次登录日期的方法；

|  |
| --- |
| package cn.shop.dao; import cn.shop.vo.Admin; public interface IAdminDAO extends IDAO<String,Admin> {  */\*\*  \* 本操作实现管理员的登录功能，在登录完成之后需要将上一次的登陆时间取出  \* 传递的是一个VO类对象，所以直接将登陆日期设置到次对象中即可返回  \** ***@param*** *vo 包含有aid与password数据  \** ***@return*** *登录成功返回true，否则返回false  \** ***@throws*** *Exception  \*/* public boolean findLogin(Admin vo) throws Exception ;   */\*\*  \* 本操作是更新最后一次的登录日期，只需要传入更新的管理员编号即可  \** ***@param*** *aid 管理员编号  \** ***@return*** *更新成功返回true，否则返回false  \** ***@throws*** *Exception  \*/* public boolean doUpdateLastDate(String aid) throws Exception ; } |
| @Override public boolean findLogin(Admin vo) throws Exception {  String sql = "SELECT lastdate FROM admin WHERE aid=? AND password=?" ;  super.pstmt = super.conn.prepareStatement(sql) ;  super.pstmt.setString(1,vo.getAid());  super.pstmt.setString(2,vo.getPassword());\  ResultSet rs = super.pstmt.executeQuery() ;  if (rs.next()) {  vo.setLastdate(rs.getTimestamp(1));  return true ;  }  return false; } |
| @Override public boolean doUpdateLastDate(String aid) throws Exception {  String sql = "UPDATE admin SET lastdate=? WHERE aid= ?" ;  super.pstmt = super.conn.prepareStatement(sql) ;  super.pstmt.setTimestamp(1,new Timestamp(new java.util.Date().getTime())) ;  super.pstmt.setString(2,aid);  return super.pstmt.executeUpdate() > 0; } |

此时如果要调用业务层的话一定是以上两个方法同时进行。

#### 1.6.2、定义业务层操作

此时的业务层不再属于前台，而属于后台业务，所以应该保存service.back包下；

|  |
| --- |
| package cn.shop.service.back; import cn.shop.vo.Admin; public interface IAdminServiceBack {  */\*\*  \* 实现管理员的登录操作，本操作要执行如下代码：<br>  \* <li>调用IAdminDAO.findLoginF()方法确定用户名和密码是否正确 ，如果正确返回最后一次登录日期；</li>  \* <li>调用IAdminDAO.doUpdateLastdate()方法，更新最后一次的登录日期</li>  \** ***@param*** *vo 只包含有aid与password  \** ***@return*** *登录成功反回true，否则返回false  \** ***@throws*** *Exception  \*/* public boolean login(Admin vo ) throws Exception ; } |
| package cn.shop.service.back.impl; import cn.shop.dbc.DatabaseConnection; import cn.shop.factory.DAOFactory; import cn.shop.service.back.IAdminServiceBack; import cn.shop.vo.Admin; public class AdminServiceBackImpl implements IAdminServiceBack {  private DatabaseConnection dbc = new DatabaseConnection() ;  @Override  public boolean login(Admin vo) throws Exception {  try {  if (DAOFactory.*getIAdminDAOInstance*(this.dbc.getConnection()).findLogin(vo)) {  return DAOFactory.*getIAdminDAOInstance*(this.dbc.getConnection()).doUpdateLastDate(vo.getAid()) ;  }  return false ;  }catch (Exception e) {  throw e ;  } finally {  this.dbc.close();  }  } } |

#### 1.6.3、编写AdminLoginServletBack

### 1.7用户的批量操作

用户批量操作主要是牵扯到如下几个功能：

·将多个用户的状态由待激活变为激活；

·将多个用户的状态由激活变为锁定；

·将多个用户的状态由锁定变为激活。

#### 1.7.1、需要批量的更新状态的操作功能；

直接在IMemberDAO里面增加批量的更新操作；

|  |
| --- |
| */\*\*  \* 是进行数据的批量更新，状态由外部设置  \** ***@param*** *ids  \** ***@param*** *status  \** ***@return*** *\** ***@throws*** *Exception  \*/* public boolean doUpdateStatus(Set<String> ids, Integer status) throws Exception; |

**注意：在业务层里面不能够设置status参数，因为如果要修改状态的话，数值应该由业务层定义完整。**

|  |
| --- |
| @Override public boolean updateActive(Set<String> ids) throws Exception {  try {  return DAOFactory.*getIMemberDAOInstance*(this.dbc.getConnection()).doUpdateStatus(ids,1) ;  } catch (Exception e) {  throw e ;  } finally {  this.dbc.close();  } }  @Override public boolean updateLock(Set<String> ids) throws Exception {  try {  return DAOFactory.*getIMemberDAOInstance*(this.dbc.getConnection()).doUpdateStatus(ids,0) ;  } catch (Exception e) {  throw e ;  } finally {  this.dbc.close();  } } |

### 1.8、用户信息查看

可以打开一个窗口查看用户的完整信息，同时也可以实现用户状态的更新。

1. 如果要查看用户的状态，则应该覆写好findVyId()方法。
2. 在业务层中增加一个show()的方法。
3. 在Servlet里面增加一个show()方法的定义。
4. 建立member\_show.jsp页面进行显示

### 1.9、商品信息管理

#### 1.9.1、商品类型管理

商品的类型使用的是item表，这张表主要是进行基础的CRUD操作并且本表里面不需要使用分页处理。

1. 建立数据层标准；

2、定义业务层标准

3、定义控制层

4、定义item\_insert.jsp页面实现分类信息的添加

5、定义item\_list.jsp页面。

在之前所有的操作代码之中如果要进行数据的修改都要求提供有一个updatePre()的操作，查询出所有的内容，但是现在不需要此方法，因为item表的字段很少。以下的实现操作，适合于表数据少的情况。



在整个页面过程之中最重要的问题是该如何取得文本框的数据 。

最终数据还是依靠表单进行提交，所以可以定义一个表单，只不过这个表单里面只有隐藏域数据；

|  |
| --- |
| <form id="itemForm" method="post">  <input type="hidden" name="iid" id="iid">  <input type="hidden" name="title" id="title"> </form> |

在最终编写函数的时候明确的将取得的数据设置到隐藏域的内容下，而后手工提交表单。

|  |
| --- |
| **function** goUpdate(iid) {  **if** (validateEmpty("title-" + iid)) { // 验证可以通过  title = document.getElementById("title-" + iid).value ;  document.getElementById("iid").value = iid ;  document.getElementById("title").value = title ;  document.getElementById("itemForm").submit() ;  }  } |

这种形式只适合于表字段少的情况，一般表字段只有1~3个的时候常用，为了节约页面。

#### 1.9.2、商品管理

商品管理牵扯到文件的上传操作，所以还是按照习惯性的顺序式思路，不集中进行数据层与业务层设计。

##### 1.9.2.1、商品添加

1、准备好程序的VO类，但是在定义VO类的时候一定要记住要配置好与Item、Admin的关系；

2、建立IGoodsDAO接口

|  |
| --- |
| package cn.shop.dao; import cn.shop.vo.Item; public interface IItemDAO extends IDAO<Integer,Item> { } |

如果只考虑添加操作，只需要覆写doCreate()方法即可

3、定义IGoodsServiceBack接口，因为要想实现增加操作需要两个方法支持:

* 1. insertPre()，找到所有的栏目信息；
  2. insert()，执行增加操作；

4、定义GoodsServletBack程序类，实现增加操作；

5、定义goods\_insert.jsp页面；

##### 1.9.2.2、商品基础列表

由于商品存在有一个状态问题，所以必须可以根据状态列出所有的商品信息。

1、完善数据的分页显示操作；

2、在IGoodsServiceBack接口里面定义列表显示的方法；

3、随后在Servlet中要处理分页；

4、编写goods\_list.jsp页面进行列表显示



##### 1.9.2.3、商品分类列表

开始进行分类列表，对于商品列表操作应该分为：上架商品列表、下架商品列表、删除商品列表（可以彻底删除）；

1、在IGoodsDAO接口里面增加新的操作方法；

2、在业务层里面定义三个方法；

·查询所有上架；

·查询所有下架；

·查询所有下架；

3、在GoodsServletBack程序类里面还是根据状态进行列表，而后判断操作类型；

4、功能加强：根据状态进行列表的功能并不难实现。

·如果已经是下架商品列表了，那么不应该再出现下架操作；

·如果已经是上架商品列表了，那么不应该在出现上架操作；

·如果已经是已删除商品了，那么不应该出现移到回收站操作，应该出现彻底删除操作；

##### 1.9.2.4、修改商品状态

商品的状态就是上架、下架、删除几个操作，并且在前台页面都已经有了相应支持。

1、在IGoodsDAO接口里面定义修改状态的方法；

2、而在IGoodsServiceBack接口里面需要扩充有三个方法可以对外部隐藏status的设置内容；

3、关键问题是在GoodsServletBack程序类里面需要定义更新状态的方法，并且，更具操作形式调用不同的业务层功能。

##### 1.9.2.5、商品修改

商品信息修改操作，值得是要针对于所有的数据状态进行变更，而且在修改之前需要进行数据的回显问题。

1. 在IGoodsDAO,接口子类中将findById()方法、doUpdate()方法覆写；
2. 在IGoodsServiceBack接口里面要扩充修改操作：updatePre()、update()方法；
3. 在GoodsServletBack类里面定义相应的操作方法，同时将数据传递；
4. 可以拷贝出goods\_insert.jsp页面为goods\_update.jsp页面；
5. 修改goods\_list.jsp的链接地址；

##### 1.9.2.6、商品信息删除

商品的删除操作发生在回收站里面。那么针对于删除商品信息的时候请一定要删除其对应的图片数据。

1. 在IGoodsDAO接口里面增加一个查询出所有图片信息的方法；

考虑到很多的功能里面都可能存在有取得全部图片的功能，为了方便可以在AbstractDAOImpl类里面扩充功能。

|  |
| --- |
| public Set<String> photoHandle(String table,String photoColumn,String column,Set<?> ids) throws Exception {  Set<String> all = new HashSet<String>() ;   StringBuffer sql = new StringBuffer() ;  sql.append(" SELECT ").append(photoColumn).append(" FROM ").append(table).append(" WHERE ").append(column).append(" INT( ") ;  Iterator<?> iter = ids.iterator() ;  while (iter.hasNext()) {  sql.append(iter.next()).append(",") ;  }  sql.delete(sql.length() - 1,sql.length()).append(" ) ") ;  sql.append(" AND ").append(photoColumn).append("<>'nophoto.jpg'") ;  this.pstmt = this.conn.prepareStatement(sql.toString() ) ;  ResultSet rs = this.pstmt.executeQuery() ;  while (rs.next()) {  all.add(rs.getString(1)) ;  }  return all ; } |

查询所有图片工具类

2、在IGoodsServiceBack接口里面扩充新的方法；

3、控制层中定义的方法一定要删除对应的图片信息；

### 2.0、购物车功能的实现

三种购物扯得实现原理。

在一些购物商城的站点上，用户可以把自己喜欢的商品临时性的增加到一个位置上，而且通过购物车可以直接实现订单的创建。在购物车里面除了要保存商品之外，还需要保存商品的数量。

那么在整个购物车里面需要保存是购买商品的编号和所购买商品的数量。

#### 2.0.1、商品的前台列表

如果要想进行前台的商品列表，那么一定是可以根据商品分类进行不同的商品列表。流程：首先进入到商品列表之中会显示所有类型的商品，而后可以再查看某一个类型的商品。

1. 用户可以查看上架商品，所以必须根据商品状态查询出全部商品；

还需要定义一个可以根据商品类型查看，所以要在IGoodsDAO接口里面重新定义一个方法；

findAllByStatus()方法与findAllByItem()这两个方法里面最本质的区别在于，一个有分类ID，另一个没有分类ID。

1. 商品的信息列表属于前台页面功能，那么应该定义一个前台的服务接口 —— IGoodsServiceFront.java；

|  |
| --- |
| package cn.shop.service.front; import java.util.Map; public interface IGoodsServiceFront {  */\*\*  \* 进行全部商品列表的时候使用本操作要执行如下的调用：<br>  \* <li>要调用IItemDAO.findAll()方法查询出全部的商品分类</li>  \* <li>要调用IGoodsDAO.findAllByStatus()方法查询出全部数据</li>  \* <li>要调用IGoodsDAO.getAllCountByStatus()方法查询出全部数据量</li>  \* <li>调用的时候status设置的内容为1，表示上架商品</li>  \** ***@param*** *currentPage  \** ***@param*** *lineSize  \** ***@param*** *column  \** ***@param*** *keyWord  \** ***@return*** *返回的数据一共包含有三类内容：<br>  \* <li>key = allItems、value = IItemDAO.findAll()，保存的是List<Item></li>  \* <li>key = allGoods、value = IGoodsDAO.findAllByStatus()，保存的是List<Goods></li>  \* <li>key = goodsCount、value = IGoodsDAO.getAllCountByStatus(),保存的是Integer</li>  \** ***@throws*** *Exception  \*/* public Map<String,Object> list(int currentPage,int lineSize,String column,String keyWord) throws Exception ;  */\*\*  \* 进行全部商品列表的时候使用本操作要执行如下的调用：<br>  \* <li>要调用IItemDAO.findAll()方法查询出全部的商品分类</li>  \* <li>要调用IGoodsDAO.findAllByItem()方法查询出全部数据</li>  \* <li>要调用IGoodsDAO.getAllCountByItem()方法查询出全部数据量</li>  \* <li>调用的时候status设置的内容为1，表示上架商品</li>  \** ***@param*** *iid 商品类型的编号  \** ***@param*** *currentPage  \** ***@param*** *lineSize  \** ***@param*** *column  \** ***@param*** *keyWord  \** ***@return*** *返回的数据一共包含有三类内容：<br>  \* <li>key = allItems、value = IItemDAO.findAll()，保存的是List<Item></li>  \* <li>key = allGoods、value = IGoodsDAO.findAllByItem()，保存的是List<Goods></li>  \* <li>key = goodsCount、value = IGoodsDAO.getAllCountByItem(),保存的是Integer</li>  \** ***@throws*** *Exception  \*/* public Map<String,Object> listByItem(int iid,int currentPage,int lineSize,String column,String keyWord) throws Exception ; } |

3、编写GoodsServletFront程序类实现商品列表的实现功能；

在进行前台列表的时候一定要记住考虑两种情况：

·情况一：所有的商品都显示，这个实在信息加载的时候准备；

·情况二：可以根据分类显示，那么假设要想显示分类的参数名称为iid。

4、随后就需要在前台写出链接地址；

5、建立goods\_list.jsp页面进行信息显示



如果要想实现分类查看，那么只需要设置一个下拉列表就行了

#### 2.0.2、查看商品的信息

在查看商品信息的时候一定需要更新商品访问次数的。此功能可以直接在业务层完成。

1. 在IGoodsDAO接口里面增加新的更新操作方法，更新访问次数；

|  |
| --- |
| */\*\*  \* 更新访问次数，每次调用访问次数加 1  \** ***@param*** *id  \** ***@return*** *\** ***@throws*** *Exception  \*/* public boolean doUpdateBow(Integer id) throws Exception ; |

本次需要doUpdateBow()与findById()两方法一起完成操作。

1. 在前台业务层中增加一个查看商品信息的方法；
2. 在GoodsFrontServlet程序里面定义show()方法进行商品信息显示；
3. 定义goods\_show.jsp页面进行商品的完整显示；

#### 2.0.3、购物车实现

对于购物车实现可以规划为两种方式：

·第一种方式：临时信息保存；

|- 由于每一个人所购买的商品信息是不一样的，程序中可以利用session区分不同的用户，那么将用户购买的信息保存在session之中，但是这样的做法有一个问题，当浏览器关闭之后信息就不存在了，所以可以进一步的延伸，利用Cookie进行信息的保存，即使浏览器关闭了，那么所购买的商品信息依然存在；

·第二种方式：持久化信息保存，利用数据库进行购买商品的信息保存。

所有购物车的核心的几个功能：增加购物车、修改购物车中的商品数量（列表）。

##### 2.0.3.1、利用session实现购物车

现在在购物车里面一定需要保存有购买商品的编号以及购买商品的数量，所以可以在session中保存一个Map集合，这样就可以同时保存一对数据。

1. 实现购物车数据的添加；

对于session中要保存Map集合而言，每一个用户在创建session的时候都不存在Map集合。可以在用户操作session属性之前做一个判断。

|  |
| --- |
| public String insert(HttpServletRequest request) {  // 保存商品编号（Integer、key）购买数量（Integer、value）  Map<Integer,Integer> map = null ;  // 表示已经存储过来Map集合  if (request.getSession().getAttribute("allCars") != null) {  // 找出原始的保存信息  map = (Map<Integer, Integer>) request.getSession().getAttribute("allCars") ;  } else {  // 如果没有session属性，可能是第一次操作  map = new HashMap<Integer, Integer>() ;  }  int gid = Integer.*parseInt*(request.getParameter("gid")) ;  // 商品数量  int count = 1 ;   // 判断指定的key是否存在，如果存在证明此商品已经保存过了  if (map.containsKey(gid)) {  // 在原始数量上加1  count = map.get(gid) + 1 ;  }  // map集合的特点是：如果key重复则会发生替换  map.put(gid,count) ;  request.getSession().setAttribute("allCars",map);  String msg = "成功加入购物车！" ;  String url = request.getHeader("referer") ;  request.setAttribute("msg",msg);  request.setAttribute("url",url);  return "/pages/forward.jsp" ; } |

在页面上增加操作链接；

1. 需要查看购物车中的全部数据信息，查看的时候肯定不能够只是单纯的列出商品id，需要展示的是所有商品的完整内容，例如：图片、名称、价格、购买数量。

·需要增加一个根据指定id查询出所有满足此ID商品的操做方法；

|  |
| --- |
| */\*\*  \* 查询指定编号的所有商品信息  \** ***@param*** *ids  \** ***@return*** *\** ***@throws*** *Exception  \*/* public List<Goods> findAllByGid(Set<Integer> ids) throws Exception ; |

·定义一个IShopcarServiceFront接口需要在业务层之中编写一个listCar()方法，可以列出所有的商品信息；

1. 在Servlet之中取出Map中全部的key（keySet()）查询出所有的商品信息。
2. 修改购买数量，利用的是参数的传递；
3. 计算每种商品的购买总价，以及总的花费金额，这一操作利用JavaScript完成。

##### 2.0.3.2、基于Cookie的实现

以上通过session的操作已经实现了购物车的基本功能，但是现在有一点比较麻烦，就是说，所购买的商品浏览器一关闭数据就没了，那么为了让浏览器即使被关闭，也可以找到所有的商品信息，可以采用Cookie的方式保存。

但是这里面有一点问题。很多情况下都可能像用户的Cookie存放一堆内容，最典型的： JSESSIONID，所以为了可以明确的区分出存放的是购物车的内容，那么在设置Cookie内容的时候：

·name： 保存的商品编号sc-gid

·value：保存购买数量。

在得到全部的Cookie之后可以判断以“sc-”开头的为购物车数据。

1. 需要定义一个可以操作Cookie的类；
2. 而后再定义一个购物车的Cookie操作类，这个类可以根据全部取出的Cookie，再筛选出所有的与购物车有关的内容。

|  |
| --- |
| package cn.shop.util; import cn.util.CookieUtil; import javax.servlet.http.HttpServletRequest; import javax.servlet.http.HttpServletResponse; import java.util.HashMap; import java.util.Iterator; import java.util.Map; import java.util.Set; public class ShopcarCookieUtil {  public static void addCar( HttpServletResponse response,HttpServletRequest request,int gid,int count) {  CookieUtil.*save*(response,request,"sc-" + String.*valueOf*(gid),String.*valueOf*(count),50000);  }    public static Map<Integer,Integer> loadCar(HttpServletRequest request) {  Map<Integer,Integer> result = new HashMap<Integer, Integer>() ;  Map<String,String> map = CookieUtil.*load*(request) ;  Iterator<Map.Entry<String,String>> iter = map.entrySet().iterator() ;  while (iter.hasNext()) {  Map.Entry<String,String> me = iter.next() ;  if (me.getKey().startsWith("sc-")) { // 为购物车数据  result.put(Integer.*parseInt*(me.getKey().split("-")[1]),Integer.*parseInt*(me.getValue())) ;  }  }  return result ;  }    public static void removeCar(HttpServletRequest request, HttpServletResponse response,Set<Integer> ids) {  Iterator<Integer> iter = ids.iterator() ;  while (iter.hasNext()) {  CookieUtil.*save*(response,request,"sc-" + String.*valueOf*(iter.next()),"0",0);  }  }  public static void clearCar(HttpServletRequest request, HttpServletResponse response) {  CookieUtil.*clear*(request,response);  }  } |

与之前的CookieUtil类相比，此时的类具备了数字的操作，但是核心的功能依然是Cookie的操作。

3、随后再进行购物车处理的时候直接操作Cookie数据即可；

##### 2.0.3.1、基于数据库的实现

使用Cookie的操作一定要比Session更加方便，但是也会带来一个问题，例如，用户在A电脑上选择了商品，而后到B电脑，那么商品就不在了，所以这一点在移动时代根本就不可能去接受，移动的时候所有的数据都必须保存在服务器（数据库），但如果要想基于数据库的实现，必须首先创建一张可以保存信息的数据表。



1、准备出数据库创建脚本。

2、定义DAO操作接口；

在整个购物车应用之中，具备如下的功能：

·增加购物车信息，但是这个信息牵扯到一个数据增加的问题；

·修改购物车中的购买信息，需要先删除掉所有的用户的购买记录，而后再重新保存新的记录；

·购物车数据列表，应该得到所有购物车的购买商品记录；

3、定义业务层操作接口；

4、修改ShopcarServletFront类；

此时由于要与用户名结合在一起使用，所以程序必须登陆后才可以操作 ；

5、增加过滤器配置，只有登陆用户可以进行购物车操作。

### 2.1、订单生成与管理

#### 2.1.1、完善个人信息

个人信息是否完善直接影响到最后的订单操作，所以应该首先是实现完善个人信息的功能。

只有登陆过的用户才可以进行个人信息的修改所以对于修改的表单里面不再需要编写隐藏域传递用户ID。

1、完善IMemberDAO接口方法；

2、在进行信息更新前需要查询出数据，而后再进行更新操作，在IMemberServiceFront接口里面扩充方法。

3、而后在MemberInfoServletFront程序类里面定义同样方法名称；

千万要记住，此时所有的mid通过session取得

#### 2.1.2、创建订单

如果现在要进行订单创建，则必须考虑到如下的几种情况：

·所有的订单一定是要通过选择购物车中的内容进行创建，所以必须保证购物车不能为空，也就是说购物车里面必须有数据，如果没有数据应该进行一个错误提示；

·创建订单时必须保证用户的个人信息已经完善，如果没有完善则应该提示用户先进行个人信息完善的处理。

·在创建订单的时候还需要在订单详情里面进行保存，还需要计算出总的金额；

·创建订单之后还必须减少相应商品的库存量，订单详情一定需要有订单编号，但是订单编号是自动增长的，所以现在必须清楚自动增长后的订单ID；

·在每一条订单详情里面还需要知道商品名称和价格。

如果要定义业务层应该只提供一个insert()方法，最终之后返回成功或失败的标记。

|  |
| --- |
| public Boolean insert(String mid); |

那么所有的这些错误信息**采用自定义异常类的方式**处理。

1. 创建对应的订单和订单详情的VO类，同时配置好关系；
   1. 一个用户有多个订单；
   2. 一个订单包含有多个详情，每个详情要与商品信息对应；
2. 定义订单的数据层操作接口；

|  |
| --- |
| */\*\*  \* 调用last\_insert\_id()函数取得当前增长后的订单编号，为订单详情准备  \** ***@return*** *返回最后的自动增长编号  \** ***@throws*** *Exception  \*/* public Integer findLastInsertId() throws SQLException;  public boolean doCreatOrders(Orders vo) throws SQLException ; } |

3、定义订单详情的数据接口；

|  |
| --- |
| */\*\*  \* 批量创建订单详情  \** ***@param*** *vos  \** ***@return*** *\** ***@throws*** *Exception  \*/* public boolean doCreateBath(List<Details> vos) throws SQLException ; } |

4、还需要在商品信息里面增加修改商品库存量的方法。

|  |
| --- |
| */\*\*  \* 要进行商品库存量的变更  \** ***@param*** *gid  \** ***@param*** *num  \** ***@return*** *\** ***@throws*** *Exception  \*/* public boolean doUpdateByAmount(Integer gid,Integer num) throws SQLException ; |

5、定义订单的业务层接口；

有可能出现的错误：

·用户个人信息不完善，你可以抛出UnCompleteMemberInfomrationException ;

·商品库存信息不足，可以抛出一个UnEnoughAmountException;

·购物车里面如果没有数据，不能够创建：EmptyShopcarException;

|  |
| --- |
| package cn.shop.service.front; import cn.shop.exception.EmptyShopcarException; import cn.shop.exception.UnCompleteMemberInfomrationException; import cn.shop.exception.UnEnoughAmountException; import java.sql.SQLException; public interface IOrdersServiceFront {  */\*\*  \* 创建订单操作  \** ***@param*** *mid 要创建订单的用户ID  \** ***@return*** *如果订单创建成功则返回true，否则返回false  \** ***@throws*** *UnCompleteMemberInfomrationException 个人信息不完整时抛出异常  \** ***@throws*** *UnEnoughAmountException 没有足够的库存量时抛出异常  \** ***@throws*** *EmptyShopcarException 购物车没有添加任何商品时抛出异常  \** ***@throws*** *SQLException JDBC错误所造成的异常  \*/* public boolean insert(String mid) throws UnCompleteMemberInfomrationException,UnEnoughAmountException,EmptyShopcarException,SQLException ; } |

1. 定义servlet程序（相对较容易）。

#### 2.1.3、个人订单信息

每一个用户都应该可以查看自己的订单信息，所以需要准备一个订单信息列表的操作。

1、在IOdersDAO接口里面提供有一个根据用户ID查看所有订单的功能；

|  |
| --- |
| */\*\*  \* 根据用户的编号列出所有的订单信息  \** ***@param*** *mid  \** ***@return*** *\** ***@throws*** *Exception  \*/* public List<Orders> findAllByMember(String mid,Integer curentPage,Integer lineSize) throws Exception ;  public Integer getAllCountByMember(String mid) throws Exception ; |
| */\*\*  \* 查询一个用户的一个订单信息  \** ***@param*** *mid  \** ***@param*** *oid  \** ***@return*** *\** ***@throws*** *Exception  \*/* public Orders findByIdAndMember(String mid ,Integer oid) throws Exception ; |

在查询一个用户的一个订单信息时，需要增加一个用户id，避免修改订单号而查询到他人订单信息。

2、查询出订单对应的订单详情信息，这个信息不需要分页

3、定义IOdersServiceFront接口的新方法，一个是根据用户查看所有的订单，另外一个是查看一个完整订单信息。

4、定义控制层，订单列表需要分页显示，而详情不需要分页；

5、定义前台页面：

·orders\_list.jsp页面，负责显示出所有的用户订单；

·orders\_show.jsp页面，负责查看一个订单信息。

#### 2.1.4、管理员查看订单

所有的订单最后一定要给管理员查看。

1、直接覆写OrdersDAOImpl子类中的数据查询方法；

2、建立一个IOrdersServiceBack子类，主要是查看信息。

3、控制层的基本定义形式与前台区别不大。

对于JSP页面直接复制已有形式即可。

## 总结

1. 业务层的设计已经彻底打破了原有的思路；
2. 传统的基础MVC虽然可以开发项目，可是代码的重复性以及可维护性都很差。