目录

1 引言

标题

〔1〕软件系统名称：网上订餐系统〔onlineorder〕

〔2〕模块名称：数据库设计模块、菜单管理模块、用户注册模块、订单管理模块、用户管理模块、留言管理模块、用户个人信息管理模块、用户订餐模块。

模块开发情况表

表 1-1　模块开发情况表

模块名	输入	要求的处理	输出
数据库设计模块	运行单元测试文件	创立数据库表	能成功创立数据库表
菜单管理模块	选择相应菜单功能	执行相应菜单管理方法	执行相应的添加、删除、修改和查询功能
用户注册模块	选择相应注册功能	执行用户注册方法	执行相应的添加、删除、修改和查询功能
订单管理模块	选择相应订单功能	执行相应订单管理方法	执行相应的添加、删除、修改和查询功能
用户管理模块	选择相应用户管理功能	执行相应用户管理方法	执行相应的添加、删除、修改和查询功能
留言管理模块	选择相应留言管理功能	执行相应留言管理方法	执行相应的添加、删除、修改和查询功能
用户个人信息管理模块	选择相应个人信息管理功能	执行相应信息管理方法	执行相应的添加、删除、修改和查询功能
用户订餐模块	选择相应订餐功能	执行相应订餐方法	执行相应的添加、删除、修改和查询功能

2 模块 1〔database〕

说明

数据库底层设计，涉及直接对数据库表中数据操作的公共方法的设计。

设计说明

〔1〕本模块在网上订餐系统中处于底层，包括系统内的各个实体类及属性的设计，通过 Dao 层与其它层进行交互，在 Daoimpl 里面定义对 Dao 层的具体实现。

〔2〕考虑到假设数据库被意外改动不方便重新建立，所以使用 Hibernate 进行数据库的连接和数据表的创立等工作。而且由管理员进行定期的备份工作，方便数据表信息恢复。

〔3〕由于系统中几乎对所有类对象的操作都要涉及到增加、删除、修改和查询，为了增强代码的重用性，所以把对数据库增删改查的操作都提取出来作为公共代码使用。

原代码清单

2 子模块 1〔valueObject〕

〔1〕用户信息表〔t_user〕//用Hibernate的XML方式，在中，管理用户表

```
<hibernate-mapping package="cn.soft.vo">
<class name="Person" table="t_user">
      <id name="user_id" type="integer" ><generator class="native"/></id>
      <property name="user_name" length="30" not-null="true" />
      <property name="user_pass" length="30" not-null="true"/>
      <property name="user_mail" length="30" not-null="true"/>
      <property name="user_realname" length="50"/>
      <property name="user_sex" length="30"/>
      <property name="user_role" length="4" not-null="true"/>
      <one-to-one name="userinfo" property-ref="user" cascade="delete"/>
      <!—配置跟用户信息表的一对一的关系映射-- >
            <key column="user_id" />
        <one-to-many class="Orders"/>
      </set>
      <set name="message"    inverse="true" cascade="delete">
      <!-- 配置跟留言表Message的一对多的关系映射 -->
            <key column="user_id"/>
        <one-to-many class="Message"/>
      </set>
</class>
```

```
</hibernate-mapping>
```

〔2〕菜单信息表〔t_menu〕//用Hibernate的XML方式，在中，管理菜单表

```
<class name="MyMenu" table="t_menu">
        <id name="menu_id" type="integer" ><generator class="native"/></id>
        <property name="menu_name" length="50" not-null="true"/>
        <property name="menu_intro"    length="50" not-null="true"/>
        <property name="menu_price" length="5" not-null="true" />
        <set name="orders" inverse="false" cascade="delete">
            <key column="menu_id" />
            <one-to-many class="Orders"/>
        </set>
    </class>
```

〔3〕订单信息表〔t_order〕//用Hibernate的XML方式，在中，管理订单表

```
    <class name="Orders" table="orders">
        <id name="order_id" type="integer" ><generator class="native"/></id>
        <property name="order_num" not-null="true"/>
        <property name="order_notice" />
        <property name="other_notice"/>
        <property name="states" length="4" not-null="true"/>
        <property name="order_date"/>
        <many-to-one name="user" column="user_id" not-null="true"/>
        <many-to-one name="menu" column="menu_id"/>
    </class>
```

〔4〕留言信息表〔t_message〕//用Hibernate的XML方式，在中，管理留言表

```
    <class name="Message" table="t_message">
        <id name="id" type="integer" ><generator class="native"/></id>
        <property name="subject" length="45"    not-null="true"/>
        <property name="content" length="200" not-null="true"/>
        <property name="dates" column="mg_date" length="45" not-null="true"/>
        <many-to-one name="user" column="user_id" not-null="true"/>
    </class>
```

〔5〕配送信息表〔userInfo〕//用Hibernate的XML方式，在中，管理留言表

```
    <class name="UserInfo" table="t_userInfo">
        <id name="id" type="integer" ><generator class="native"/></id>
        <property name="address" length="100" not-null="true"/>
        <property name="tel" length="15"/>
        <property name="mobile" not-null="true" length="11"/>
        <property name="notice" length="150"/>
        <property name="sendDate" column="send_date" length="45"/>
        <many-to-one name="user" column="user_id" unique="true"/>
    </class>
```

2.3.2 子模块 2〔connectConfig〕

〔1〕ApplicationContext.xml 中配置和数据库的连接，设置 Hibernate 对数据库的自动管理，由 Spring 将所有类当作 Bean 进行管理。

```
<context:component-scan base-package="cn.soft"/>
<bean id="dataSource" class="com.mchange.v2.c3p0.ComboPooledDataSource"
destroy-method="close"><!—配置数据库连接名、用户名和密码等，配置数据源-->
</bean>
<bean id="sessionFactory"
    class="org.springframework.orm.hibernate3.LocalSessionFactoryBean">
    <property name="dataSource" ref="dataSource"/>
    <property name="mappingResources">
        <list><!--将所有实体类的配置文件列出，在hibernate配置文件中添加映射-->
            <value>cn/soft/vo/Person.hbm.xml</value>
        </list>
    </property>
    <property name="hibernateProperties">
        <props>
            <prop key="hibernate.dialect">org.hibernate.dialect.MySQL5Dialect</prop>
            <prop key="hibernate.hbm2ddl.auto">update</prop>
            <prop key="hibernate.transaction.factory_class">
                org.hibernate.transaction.JDBCTransactionFactory </prop>
        </props>
    </property>
</bean>
<bean id="txManager" class="org.springframework.orm.hibernate3.HibernateTransactionManager">
    <property name="sessionFactory" ref="sessionFactory"/>
</bean>
<tx:annotation-driven  proxy-target-class="true"  transaction-manager="txManager"/>
<!—配置类名和bean的id名的对应，方便在struts配置文件中使用-- >
```

〔2〕hibernate.cfg.xml中添加映射，应用XML方式创立数据库

```
<session-factory>
    <property name="hibernate.connection.driver_class">com.mysql.jdbc.Driver </property>
ost/order</property>
    <property name="hibernate.connection.username">root</property>
    <property name="hibernate.connection.password">root</property>
    <property name="hbm2ddl.auto">update</property><!-- 这里最好用update -->
    <property name="format_sql">true</property>
    <property name="hibernate.dialect">org.hibernate.dialect.MySQL5Dialect</property>
        <mapping resource="cn/soft/vo/Person.hbm.xml"></mapping>
        <mapping class="cn.soft.vo.Person" />
        <mapping resource="ApplicationContext.xml"/> -->
  </session-factory>
```

3模块2〔menuManage〕

3.1 功能说明

本模块主要实现菜单的管理工作，下设菜单添加、菜单删除、菜单修改和菜单查询 4 个子功能模块，可以对菜单中菜品的各项属性进行编辑和修改，可实现添加菜单记录时上传新菜单图片。

3.2 设计说明

〔1〕本模块通过 MenuDao 与其它层进行交互，在 MenuImpl 里面定义 dao 层的具体实现。

〔2〕考虑到假设数据库被意外改动不方便重新建立，所以使用 Hibernate 进行数据库的连接和数据表的创立等工作。而且由管理员进行定期的备份工作，方便数据表信息恢复。

〔3〕在 MenuAction 中定义与菜单表操作有关的方法，并返回字符串的结构通过 struts 中的配置跳转到下一个相应页面或方法。

3.3 原代码清单

3 子模块 1〔menuQuery〕

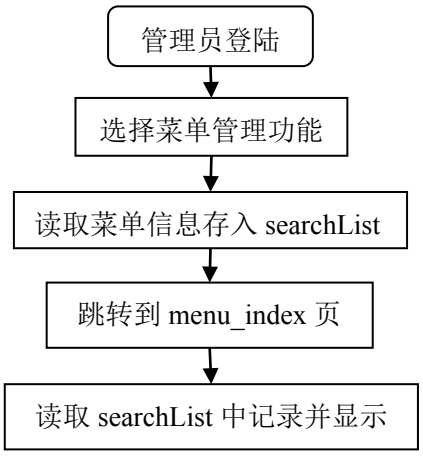


图 3-1　菜单查询流程图

//查找出所有菜单，并分页显示，供管理员管理菜单页面使用，之后返回“menulist”,跳转到manage/menu_manage下的menu_index.jsp页面，通过依次读取searchList中的记录，来逐条显示菜单信息。

```
public String menuList()
{    ActionContext ac = ActionContext.getContext();
     QueryResult<MyMenu> qr = nextPage.viewList(menuDao, page,15, MyMenu.class, "order by
o.menu_id");
     ac.getSession().put("searchList", qr.getResultSet());
     ac.getSession().put("page", page);
```

```
        return "menulist";
    }
```

3.3.2 子模块 2〔menuAdd〕

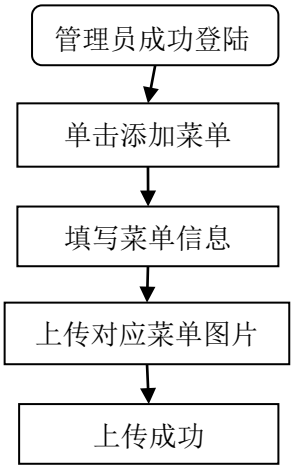


图 3-2　管理员添加菜单流程图

〔1〕添加菜单，自动按升序生成menu_id并将该条记录添加到菜单表里，跳转到上传图片界面。

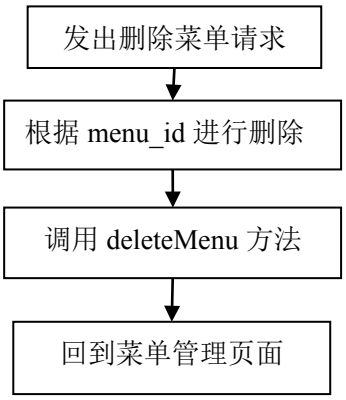
```
public String addMenu()
    {    menuDao.save(menu);
         HttpServletRequest request = ServletActionContext.getRequest();
         int new_id = menu.getMenu_id();
         request.getSession().setAttribute("new_id",new_id);
         return "success";
    }
```

〔2〕上传图片类，上传到工程webRoot文件夹下的dishes-img文件夹中，根据刚刚上传的菜单id命名为某菜单id.jpg。

```
    HttpServletResponse response = ServletActionContext.getResponse();
    HttpServletRequest request = ServletActionContext.getRequest();
    HttpSession session=request.getSession();
    public File getUpload() {return upload;}
    public void setUpload(File upload) {this.upload = upload;}
    public String getUploadFileName() {return uploadFileName;}
    public void setUploadFileName(String uploadFileName) {this.uploadFileName = uploadFileName;}
    public String getUploadContentType() {return uploadContentType;}
    public void setUploadContentType(String uploadContentType)
    {this.uploadContentType = uploadContentType;}
    public String upload()
    {
         ActionContext act=ActionContext.getContext();
         PrintWriter out=null;
         try {
              response.setCharacterEncoding("GBK");
              out=response.getWriter();
```

```
        } catch (IOException e1) {e1.printStackTrace();}
        String realpath=ServletActionContext.getServletContext().getRealPath("/dishes-img");
        if(upload!=null)
        {File saveDir=new File(realpath);
         if(!saveDir.exists())
              {saveDir.mkdirs();}
              int new_id1=(Integer)session.getAttribute("new_id");
              String img_id=String.valueOf(new_id1);
              String img_name=img_id.concat(".jpg");
              this.setUploadFileName(img_name);
              File saveFile=new File(saveDir,uploadFileName);
              try {FileUtils.copyFile(upload, saveFile);
              } catch (IOException e) {e.printStackTrace();}
        }
        return "menulist";
    }
    }
```

3.3.3 子模块 3〔menuDelete〕



3-3　删除菜单流程图

```
//删除菜单方法，跳转到菜单管理页面。
    public String deleteMenu()
    {    HttpServletRequest request = ServletActionContext.getRequest();
         int id = Integer.parseInt(request.getParameter("menuID"));
         menuDao.delete(MyMenu.class, id);
         return "success";
    }
```

3.3.4 子模块 4〔menuUpdate〕

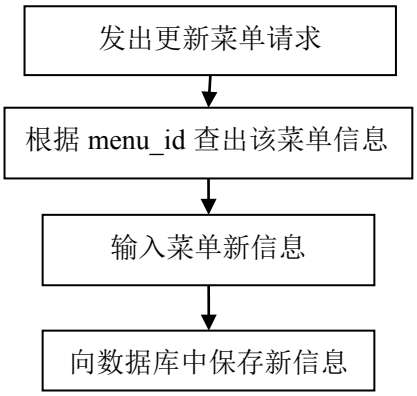


图 3-4　更新菜单流程图

〔1〕根据id先查出对应菜单的信息，然后跳转到更新页面

```
    public String updateUI()
    {
        HttpServletRequest request = ServletActionContext.getRequest();
rameter("menuID"));
        MyMenu m    = menuDao.find(MyMenu.class, id);
        request.getSession().setAttribute("menu", m);
        return "update";
    }
    //更新菜单信息,调用menuDao的update方法进行更新
    public String updateMenu()
    {   MyMenu m    = menuDao.find(MyMenu.class, menu.getMenu_id());
        m.setMenu_name(menu.getMenu_name());
        m.setMenu_price(menu.getMenu_price());
        m.setMenu_intro(menu.getMenu_intro());
        menuDao.update(m);
        return "success";
    }
    public String getName() {return name;}
    public void setName(String name) {this.name = name;}
    public int getPage() {return page;}
    public void setPage(int page) {this.page = page;}
    public MyMenu getMenu() {return menu;}
    public void setMenu(MyMenu menu) {this.menu = menu;}
    }
```

4模块3〔messageManage〕

4.1 功能说明

本模块中下设添加留言、浏览留言和删除留言 2 个子功能模块，用户可以对自己的留言进行管理，管理员可以对所有用户的留言进行管理。

4.2 设计说明

〔1〕本模块通过 MessageDao 与其它层交互，在 MessageImpl 里面定义 dao 层的具体实现。

〔2〕考虑到假设数据库被意外改动不方便重新建立，所以使用 Hibernate 进行数据库的连接和数据表的创立等工作。而且由管理员进行定期的备份工作，方便数据表信息恢复。

〔3〕在 MessageAction 中定义与菜单表操作有关的方法，并返回字符串的结构通过 struts 中的配置跳转到下一个相应页面或方法。

4.3 原代码清单

4 子模块 1〔messageQuery〕

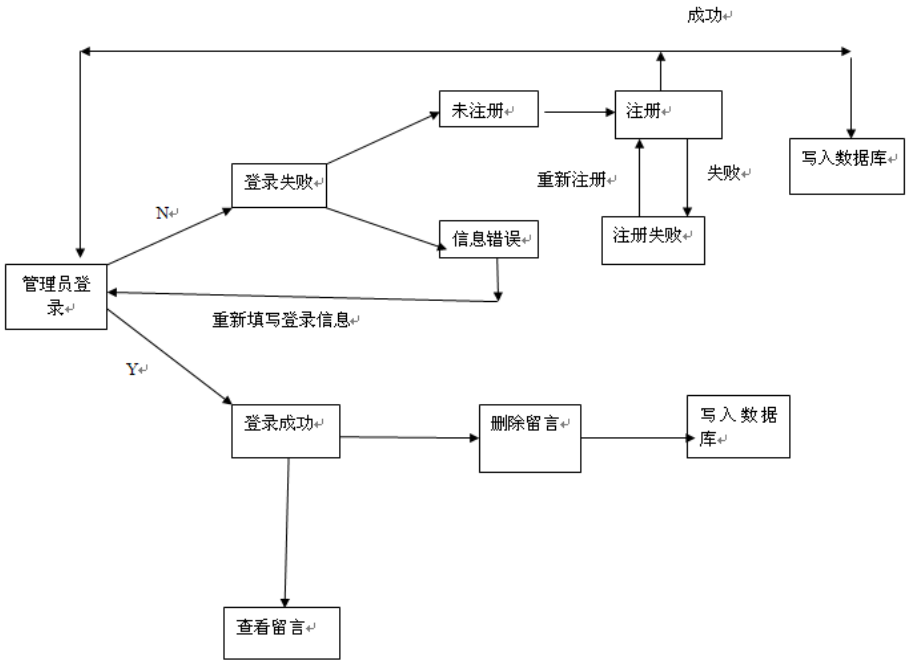


图 4-1　管理员查看留言流程图

〔1〕/*如果用户已处于登录状态，就查询出所有留言记录,返回到一个显示留言列表的视图,如果没有登录那么返回的到登录界面视图让用户先登录*/

public String messageUI()

```
{
ActionContext ac = ActionContext.getContext();
ssion().get("user_name")!=null)
{   String orderby = "order by o.id desc";//编写order by子句来决定输出的顺序
    ac.getSession().put("page", page); //将当前页号放入session范围，供分页链接使用
    ac.getSession().put("messageList", pageUtil(orderby));//将查到的相关的留言记录放入session范围，供页面显示使用
    if(ac.getSession().get("user_role").equals("0"))   //是管理员那么跳转到管理留言视图
        return "managelist";
    else   return "mglist";//如果不是管理员，那么跳转到用户添加留言视图
    }
else   return "fail"; //如果用户没有登录，那么跳转到此视图提醒用户登录
}
```

〔2〕管理员查看所有用户留言

```
ath }/test/mg_deleteMessage.action" name="myform" method="post">
    <table width="550px" align="center">
        <tr class="tr_sty"><th width="80px"></th><th>编号</th><th width="150px">主题</th><th>作者</th><th>时间</th></tr>
        <c:forEach items="${messageList }" var="mg" begin="0" step="1">
        <tr class="<% if((j++)%2==0) out.print("tr_sty0"); else out.print("tr_sty1");%>" height="25px">
        <td align="center"><input type="checkbox" name="messages" value="${mg.messageID }" ></td>
        <td align="center"><%=i++ %></td><td align="center"><a href="/onlineorder/test/mg_showOneMessage.action?messageID=${mg.messageID }">${mg.subject }</a></td>
        <td align="center">${mg.username }</td> <td align="center">${mg.date }</td>
        </tr>
        </c:forEach>
    </table>
    <input name="selectButton" type="button" value="全部选择" onclick="selectAll()">  
    <input type="submit" value="删除已选项">
</form>
```

〔3〕管理员查看单个留言

```
//此方法是用来获取一条留言记录，并将其封装到ShowMessage类中供页面显示，然后返回到oneMessage视图，此视图对应一个显示一条留言的页面
public String showOneMessage()
{   HttpServletRequest request = ServletActionContext.getRequest();
    int id = Integer.parseInt(request.getParameter("messageID"));
    Message mg = messageDao.findById(id);
    ShowMessage sm = new ShowMessage();
    sm.setUsername(mg.getUser().getUser_name());
    sm.setContent(mg.getContent());
    sm.setDate(mg.getDates());
    sm.setSubject(mg.getSubject());
```

```
        request.getSession().setAttribute("oneMg", sm);
        return "oneMessage";
    }
```

〔4〕每个用户的留言记录全部查出来，如果用户有留言那么返回到show视图，如果没有留言记录那么返回到message视图用来显示提示信息

```
    public String showOneself()
    {   ActionContext ac = ActionContext.getContext();
        int id = (Integer)ac.getSession().get("user_id");
        Set<Message> set = personDao.findMessagByID(id);
        if(set.size()>0)
        {   ac.getSession().put("oneselfList", set);
            return "show";
        }else
        {   String message = "现在还没有你的留言记录,返回";
            ac.getSession().put("message", message);
            return "message";
        }
    }
```

4.3.2 子模块 2〔messageAdd〕

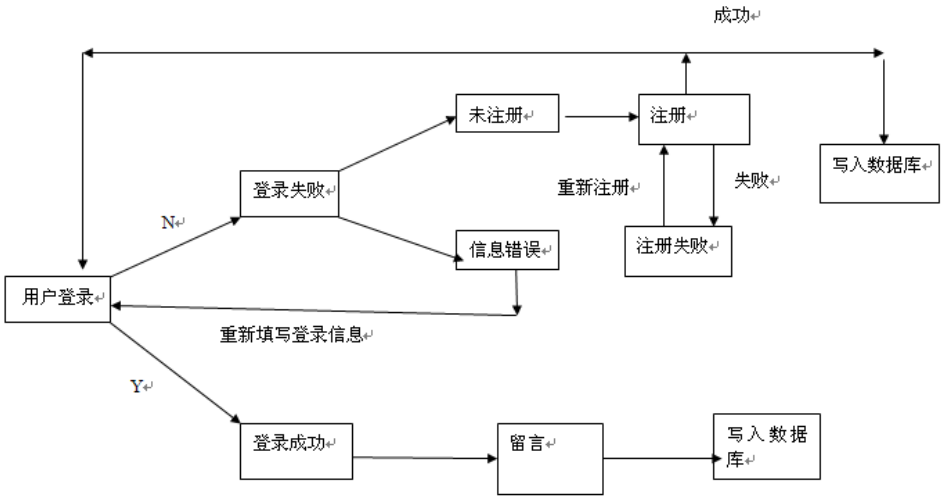


图 4-2　用户添加留言流程图

用户添加留言，成功后跳转到显示所有留言页面

```
    public String addMessage()
    {   ActionContext ac = ActionContext.getContext();
        int id = (Integer)ac.getSession().get("user_id");
        Person p = personDao.find(Person.class, id);
        message.setUser(p);
        Calendar nowTime = Calendar.getInstance();
        SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd HH：mm：ss");
        String time = sdf.format(nowTime.getTime());
        message.setDates(time);              //设置留言的时间
```