

# 《软件开发方法课程设计》子报告

项目名称 基于 Web 的电子商城系统设计与实现

学 号 xxxxxxxxxx

姓 名 xxxxxx

专 业 计算机科学与技术

任课教师 xxxxxx

日 期 2019.09.05

## 一、 写在前面

本次课程设计我选择的子项目是“子项目 6——SSH2 整合实现电子商城前台功能”，在做此子项目之前，我对 Struts 2 框架、Hibernate 5 框架、Spring 4 框架进行了系统的学习，并对其有了初步了解，在模拟例程的时候，对三者在一个 Web 项目扮演的角色也有了更深层次的了解，通过编写该子项目，我对它们组成的 M (model) V (view) C (controller) 模式有了一定的掌握，并且真实体会到它们的含义。

在已经给出的代码的基础上，通过结合自己写的 UML 模型和任务要求，我对功能进行了整合，并且通过 HTML+CSS+JavaScript 结合的方式自行编写了前段 UI 功能界面。

## 二、 系统分析与设计

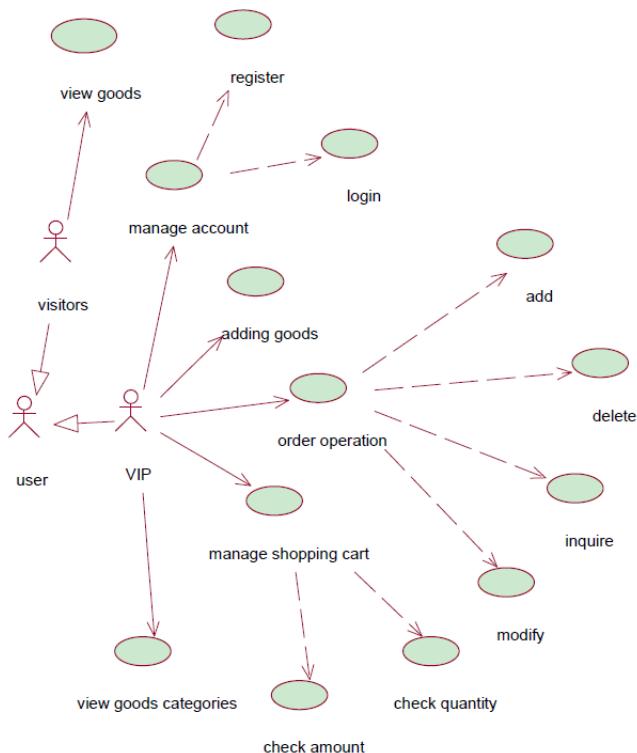
对于该子项目涉及的电子商城系统部分，主要包含了用户端的一些操作，其中包括在前台浏览商品，添加购物车，管理购物车（在购物车中添加商品数量，删除不需要的商品条目），管理订单（从购物车直接添加订单，查看订单，删除订单等），还有用户的登录和注册行为（包含验证）我也整合到其中，利用 SSH2 框架去实现。

### 1. 用例图

当前子项目的主要参与者是用户。

用户分为普通用户（游客）和会员用户，前者指只在本商城商品页面进行浏览，后者指注册了本商城账号的用户，登录账户后即可进行购买商品以及对自己账户的购物车和订单进行管理。

用户参与者用例图如下：



### 【用例说明】(虚线箭头原型是 include)

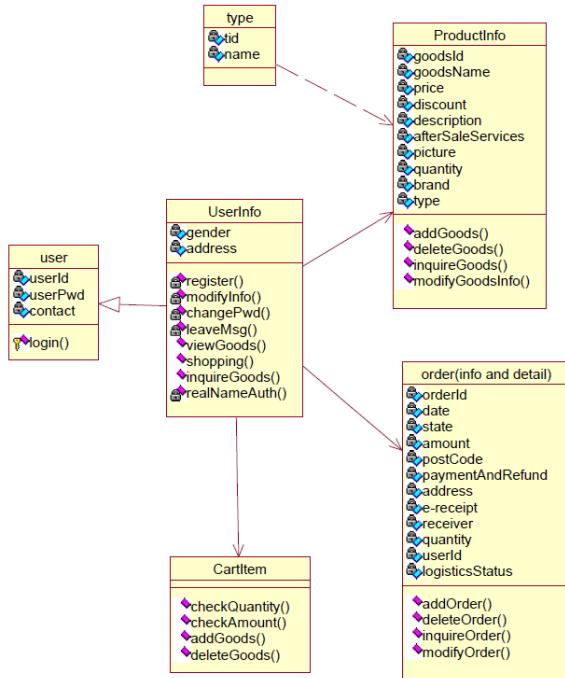
VIP 参与者, visitors 参与者: VIP 会员用户和 visitors 都是从 user 泛化而来。

- (1) view goods 用例: 游客可以浏览电子商城的商品。
- (2) manage account 用例: 用户可以用来管理账户, 其中包括注册、登录和修改个人信息。
- (3) register 用例: 用户注册商城账户, 包括设置账户名、密码、联系方式 (邮箱 or 电话号码)。
- (4) login 用例: 用户输入账号密码登录账户。
- (5) adding goods 用例: 向购物车中添加商品。
- (6) order operation 用例: 订单操作包括添加订单, 删除 (取消) 订单, 查询订单, 修改订单操作。
- (7) add 用例: 添加订单。
- (8) delete 用例: 删除 (取消订单)。
- (9) inquire 用例: 查询订单状态。
- (10) modify 用例: 修改订单信息。包括收货地址, 联系方式等信息。
- (11) manage shopping cart 用例: 管理购物车, 包括修改购买商品数量和查看总购买金额。
- (12) check quantity 用例: 核对商品数量, 可以增加或者减少购买商品数量。
- (13) check amount 用例: 核对金额, 查看购买总金额。
- (14) view goods categories 用例: 用户可以查看商品分类。

## 2. 类图

在该子项目包含的电子商城系统中, 应该有如下类:

user\_info 类 (用户信息类), product\_info 类 (商品信息类), order\_info 类 (订单信息类) order\_detail (订单明细类), CartItem (购物车类), type 类 (商品类型类)。

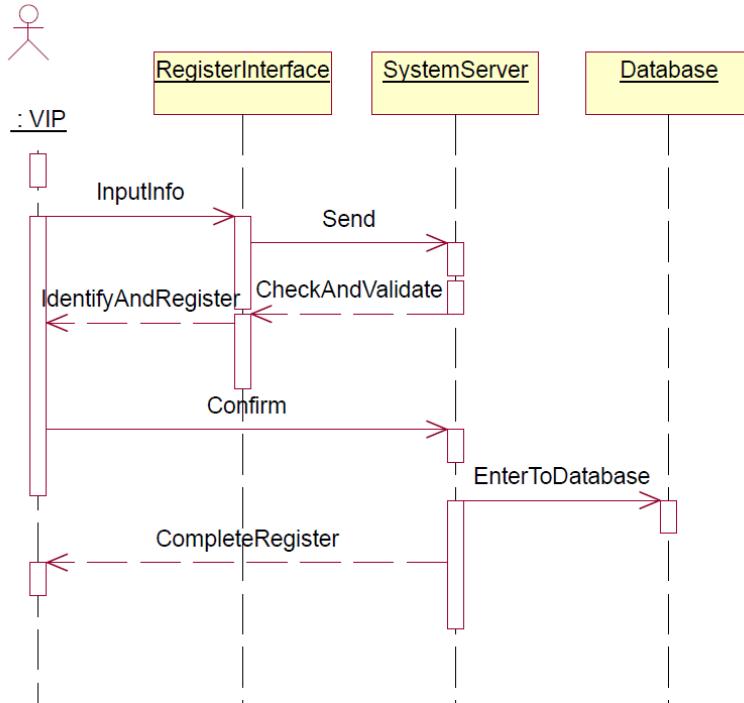


### 【类图说明】

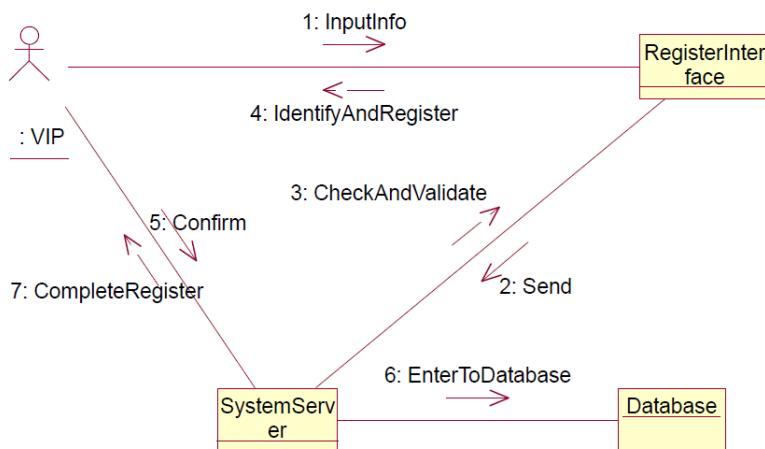
- (1) **User** 类: 所有的类的父类, 包括属性 `userId` (用户登录账户名/账号)、`userPwd` (用户登录密码)、`contact` (用户联系方式), 方法 `login` (用户登录操作)。
- (2) **user\_info (VIP)** 类: 会员类, 除了继承了 **User** 类的属性和方法之外, 还有属性 `gender` (性别)、`address` (收货地址), 方法 `register` (注册账户)、`modifyInfo` (修改资料)、`changePwd` (修改登录密码)、`leaveMsg` (给商家留言)、`viewGoods` (浏览商品)、`inquireGoods` (查询商品)、`realNameAuth` (实名认证)。
- (3) **CartItem** 类: 购物车类, 有方法 `checkQuantity` (核对商品数量)、`checkAmount` (核对金额)、`addGoods` (添加商品), `deleteGoods` (删除商品)。
- (4) **product\_info** 类: 商品类, 有属性 `goodsId` (商品编号)、`goodsName` (商品名称)、`price` (商品价格)、`discount` (商品折扣)、`description` (商品文字描述介绍)、`afterSaleServices` (售后服务)、`picture` (商品图片描述)、`quantity` (商品数量库存)、`brand` (商品品牌)、`type` (商品类型), 方法 `addGoods` (商品管理员添加商品), `deleteGoods` (商品管理员删除商品)、`inquireGoods` (商品管理员查询商品)、`modifyGoodsInfo` (商品管理员修改商品信息)。
- (5) **order\_info** 类: 订单 (订单信息和订单明细) 类, 有属性 `orderId` (订单号)、`date` (下单日期)、`state` (订单处理状态)、`amount` (订单金额)、`quantity` (商品数量)、`userId` (购物会员账号)、`receiver` (收货人)、`postCode` (邮政编码)、`paymentAndRefund` (退付款)、`address` (收货地址)、`e-receipt` (电子发票)、`logisticsStatus` (物流状态)。
- (6) **type** 类: 商品类型类, 有商品类型编号和类型名称。

### 3. 顺序图及协作图

#### 3.1.1 用户注册顺序图



3.1.2 用户注册协作图



#### 【顺序图及协作图说明】

参与者: VIP 会员

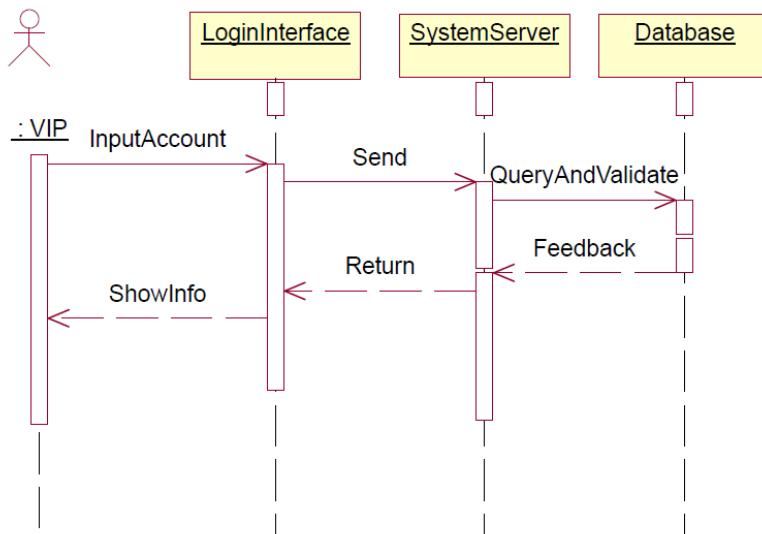
RegisterInterface: 注册界面;

SystemServer: 系统服务器;

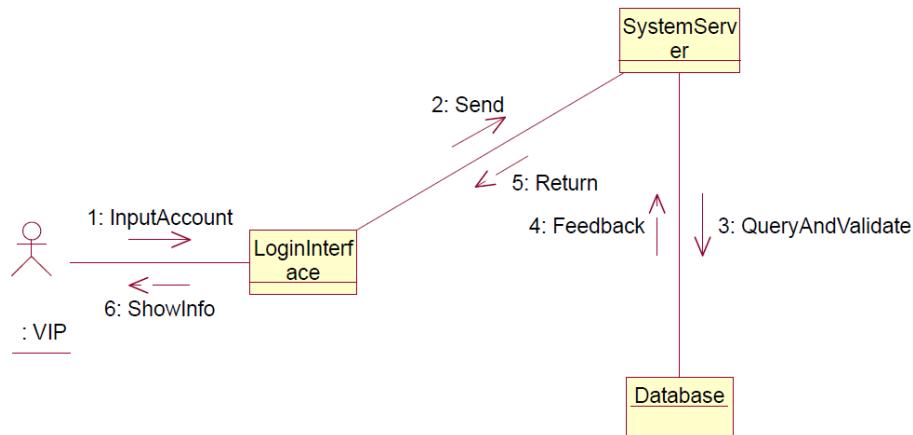
Database: 系统数据库;

- ① InputInfo: 用户在注册界面输入个人信息; 包括账户名, 密码, 绑定手机号, 常用收货地址等重要必备信息。
- ② Send: 将填好的信息发送到服务器;
- ③ CheckAndValidate: 系统服务器进行验证和审核资料;
- ④ IdentityAndRegister: 注册界面提示是否进行确认注册;
- ⑤ Confirm: 用户确认注册账户, 将信息发送给服务器;
- ⑥ EnterToDatabase: 将用户信息录入数据库;
- ⑦ CompleteRegister: 系统提示用户完成注册。

### 3.2.1 用户登录顺序图



### 3.2.2 用户登录协作图



#### 【顺序图及协作图说明】

参与者: VIP 会员;

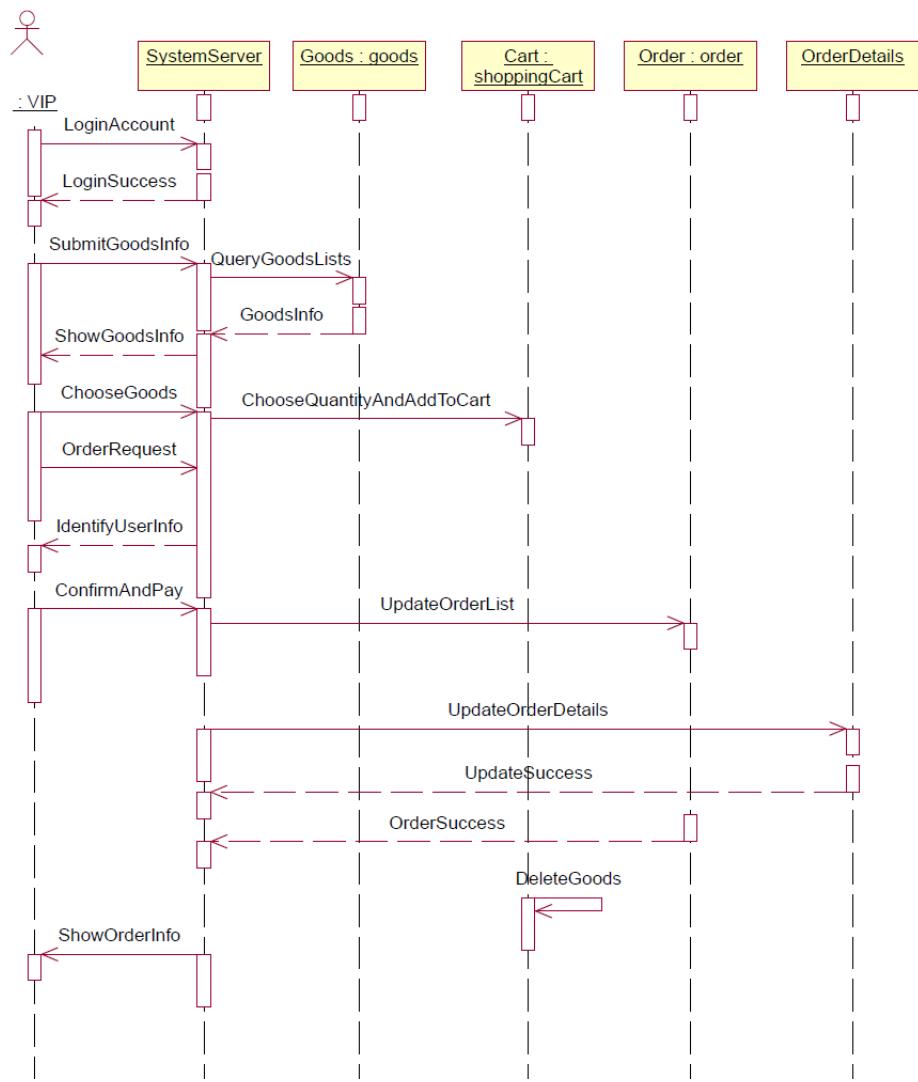
LoginInterface: 登录界面;

SystemServer: 系统服务器;

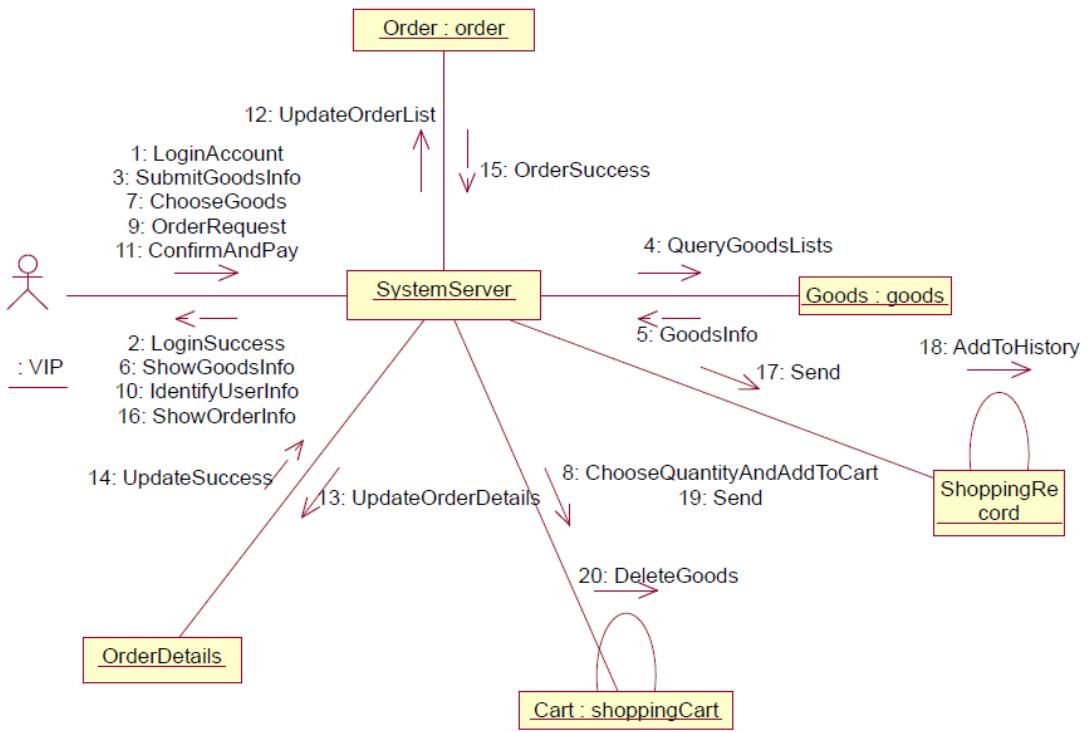
Database: 系统数据库;

- ① InputAccount: 输入账户及密码;
- ② Send: 将输入的信息发送给服务器;
- ③ QueryAndValidate: 在数据库中进行查询及验证账户名是否存在, 账户密码是否正确;
- ④ Feedback: 数据库反馈给服务器信息, 账户密码正确与否;
- ⑤ Return: 服务器返回给登录界面, 登录界面显示登录成功与否信息。
- ⑥ ShowInfo: 登录界面显示给用户信息;

### 3.3.1 会员下订单顺序图



3.3.2 会员下订单协作图



### 【顺序图及协作图说明】

执行者: VIP 会员

SystemServer: 系统服务器;

Product\_info: 商品表;

Order\_info: 订单表;

Order\_detail: 订单细节表;

Cart: 购物车;

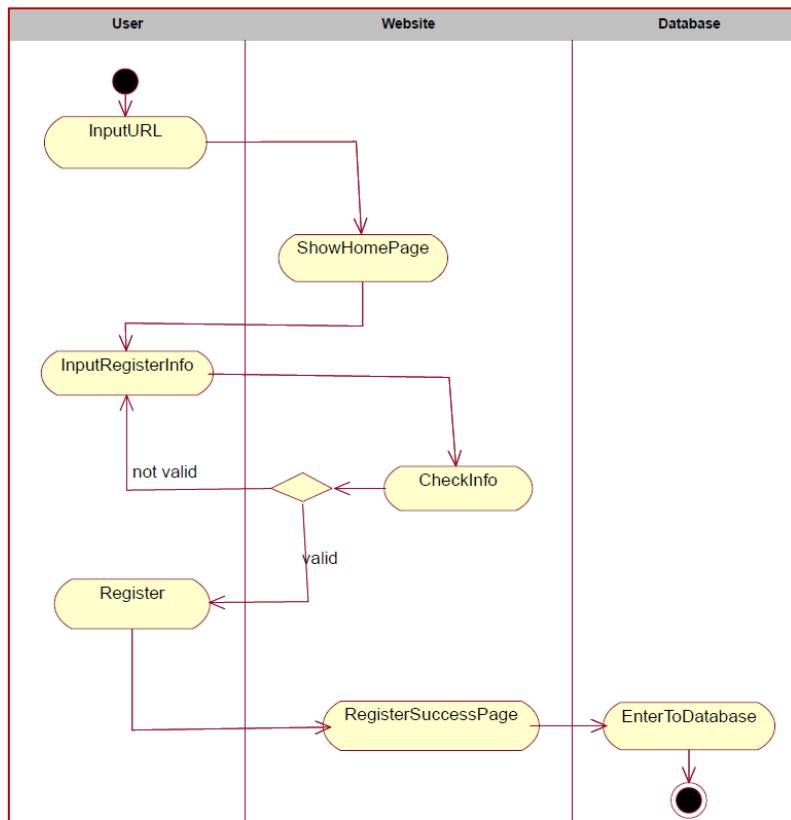
ShoppingRecord: 购物记录;

- ① LoginAccount: 会员登录账户;
- ② LoginSuccess: 系统服务器返回登录成功信息;
- ③ SubmitGoodsInfo: 提交搜索的商品信息;
- ④ QueryGoodsList: 查询商品列表是否有符合该信息的商品;
- ⑤ GoodsInfo: 若有符合条件的商品, 返回商品信息给用户;
- ⑥ ShowGoodsInfo: 显示给用户商品信息;
- ⑦ ChooseGoods: 选择商品;
- ⑧ ChooseQuantityAndAddToCart: 选择商品数量并且添加到购物车;
- ⑨ OrderRequest: 给服务器发送下单请求;
- ⑩ IdentifyUserInfo: 用户审核个人信息;
- ⑪ ConfirmAndPay: 确认下单并且付款;
- ⑫ UpdateOrderList: 服务器根据请求更新订单列表, 如订单编号等;
- ⑬ UpdateOrderDetails: 服务器更新订单细节, 比如用户个人信息等;
- ⑭ UpdateSuccess: 订单细节更新成功;
- ⑮ OrderSuccess: 下单成功;
- ⑯ ShowOrderInfo: 显示给用户订单信息;

- ⑯ Send: 服务器发送命令给“购物记录”;
- ⑰ AddToHistory: 将该商品添加到购物历史中;
- ⑱ Send: 服务器发送命令给购物车;
- ⑲ DeleteGoods: 删 除购物车中相应的商品;

#### 4. 活动图

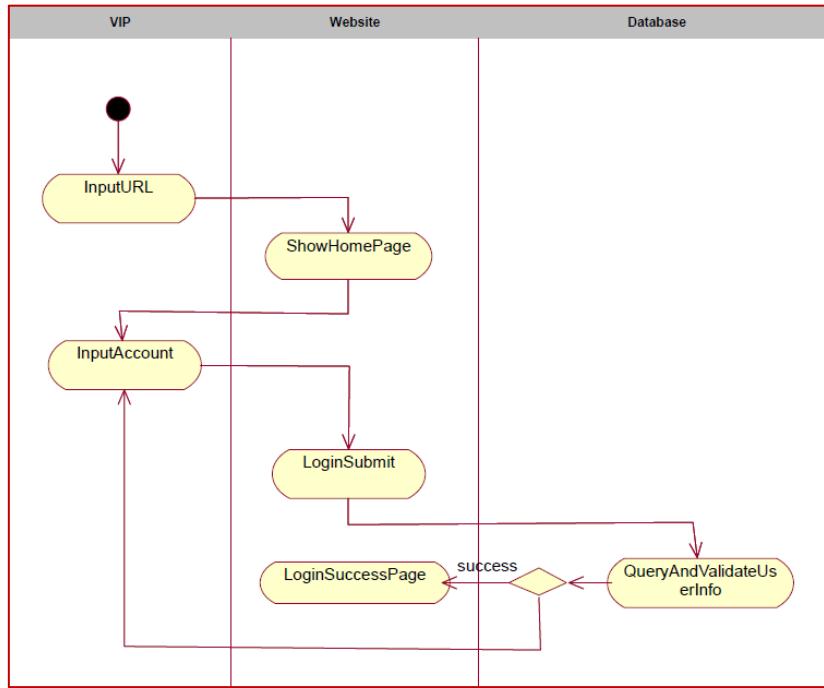
##### 4.1 用户注册活动图



##### 【活动图说明】

- (1) **InputURL**: 用户输入电商网站的链接;
- (2) **ShowHomePage**: 访问 URL 后显示电商网站的主页;
- (3) **InputRegisterInfo**: 用户输入注册信息;
- (4) **CheckInfo**: 网站审核信息的正确有效性;  
若无效，显示错误信息，回到 **InputRegisterInfo** 继续输入;  
若有效则提交 **Register** 表单;
- (5) **RegisterSuccessPage**: 网站显示注册成功页面;
- (6) **EnterToDatabase**: 将用户的注册信息录入数据库;

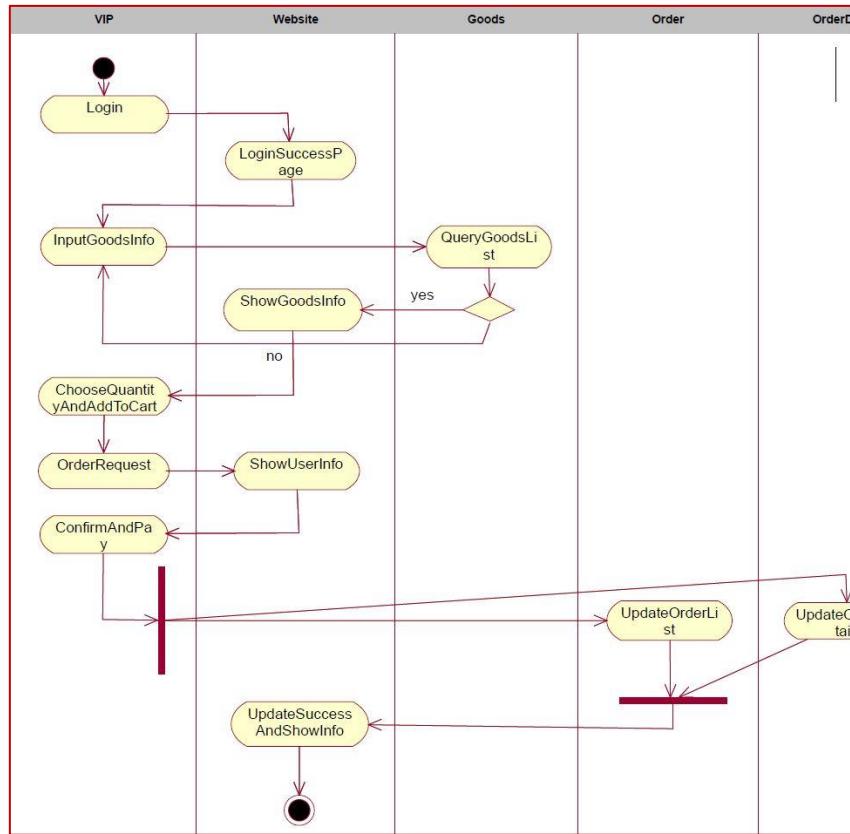
##### 4.2 用户登录活动图



#### 【活动图说明】

- (1) InputURL: 用户输入电商网站的链接;
- (2) ShowHomePage: 访问 URL 后显示电商网站的主页;
- (3) InputAccount: 输入账户名和密码;
- (4) LoginSubmit: 提交登录表单;
- (5) QueryAndValidateUserInfo: 在数据库查询并验证用户输入的账户是否存及账户存在时密码是否正确;
- (6) LoginSuccessPage: 账户名及密码有效且正确, 显示登录成功页面; 否则, 回到 InputAccount 继续输入。

#### 4.3 会员下订单活动图



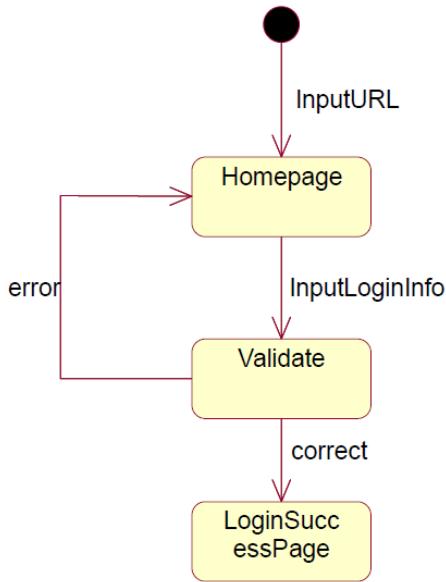
注：图中右边缺少部分为 UpdateOrderDetails 和 OrderDetails。

#### 【活动图说明】

- (1) Login: 登录电商账户;
- (2) LoginSuccessPage: 登录成功页面; (网站主页)
- (3) InputGoodsInfo: 用户输入查询商品信息;
- (4) QueryGoodsList: 查询商品列表若有该商品信息，则在网站页面显示 (ShowGoodsInfo)，若没有，则返回 InputGoodsInfo 继续输入;
- (5) ChooseQuantityAndAddToCart: 选择符合条件的商品，选择数量并且添加到购物车。
- (6) OrderRequest: 下单请求;
- (7) ShowUserInfo: 网站显示下单用户的的具体信息（收货地址，联系方式等）;
- (8) ConfirmAndPay: 用户确认个人信息之后进行付款；之后同时进行更新订单列表的订单编号等管理员掌握的信息以及订单的细节（收货地址，收货人，联系方式等）信息；
- (9) UpdateSuccessAndShowInfo: 更新成功提示及向用户显示下单成功的订单状态。

## 5. 状态图

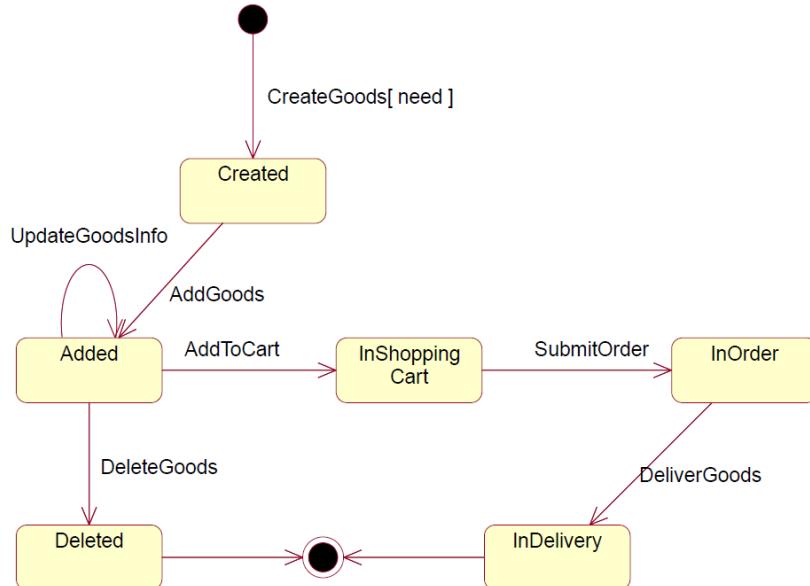
### 5.1 用户登录状态图



#### 【状态图说明】

- (1) Homepage: 网站主页;
- (2) Validate: 验证账户及密码;
- (3) LoginSuccessPage: 用户登录成功页面状态;

#### 5. 2 商品状态图

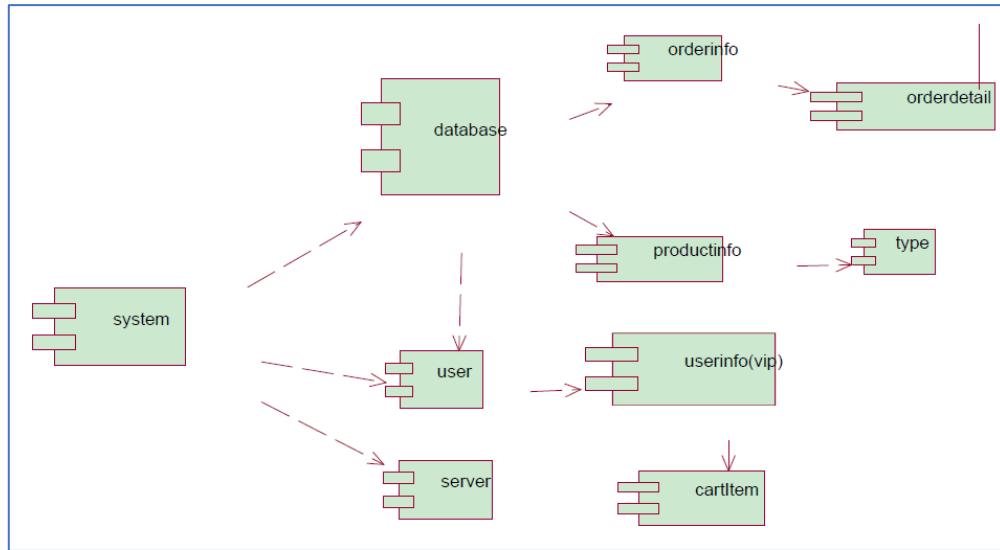


#### 【状态图说明】

- (1) Created: 通过 CreateGoods (创建商品) 事件添加, 处于创建状态;
- (2) Added: 通过 AddGoods (添加商品) 事件添加, 处于加入状态;
- (3) InShoppingCart: 通过 AddToCart (添加到购物车) 事件添加, 处于购物车中状态;
- (4) InOrder: 通过 SubmitOrder (提交订单) 事件添加, 处于下单状态;
- (5) InDelivery: 通过 DeliverGoods (运送商品) 事件添加, 处于运送状态;

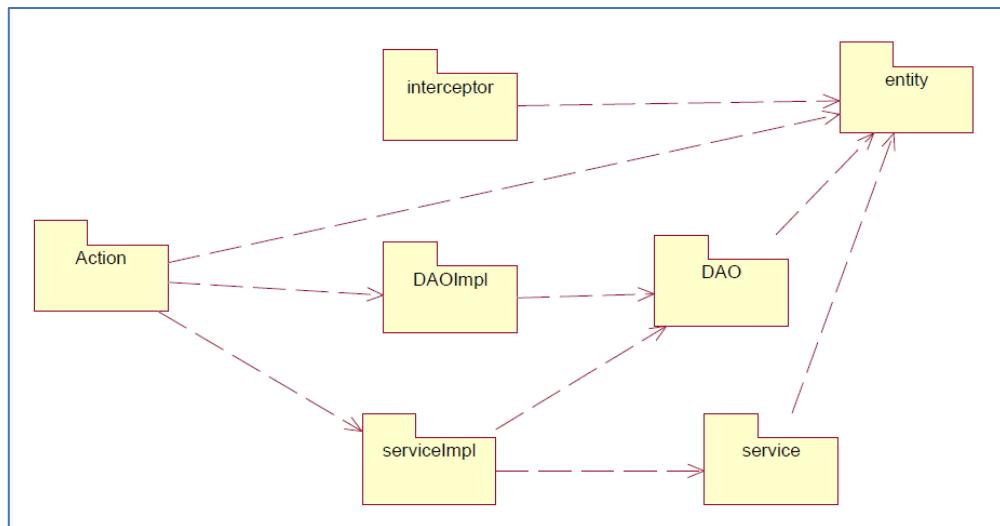
(6) Deleted: 通过 DeleteGoods (删除商品) 事件添加, 处于删除状态;

## 6. 构件图



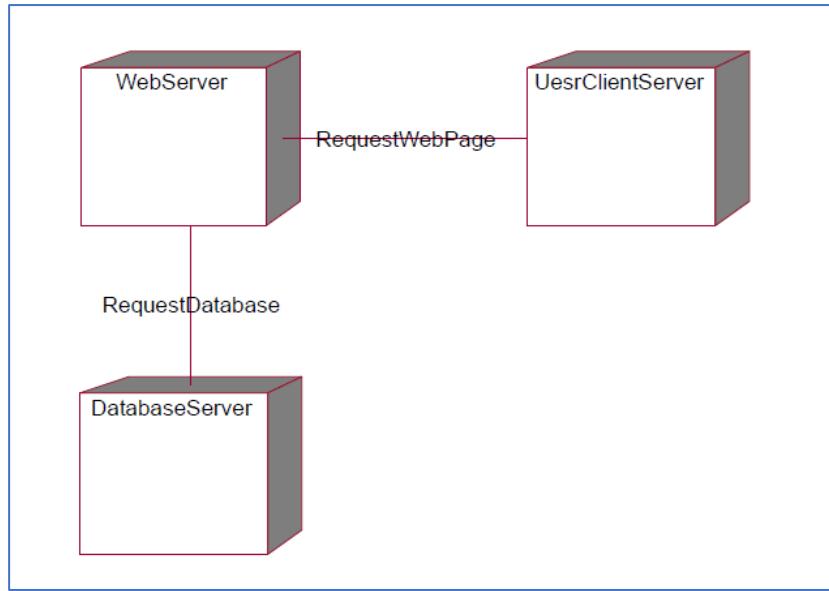
## 7. 包图

包与包之间的依赖关系可以表示为如图, 其中箭头从 A 指向 B 表示 A 调用 B 里面的类。



## 8. 部署图

部署图主要是用来说明如何配置系统的软件和硬件。整个电商网站服务器应该是总服务器, 数据库服务器负责保存整个电商网站的数据记录, 用户客户端服务器为客户提供访问该电商网站提供服务, 应该是一个终端的集合。部署图如下:



### 三、 系统功能与界面实现

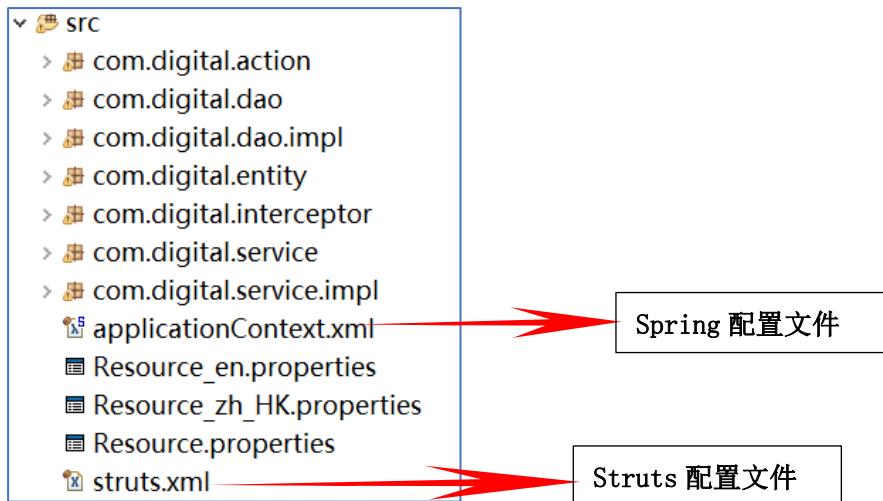
该子项目包含 6 个子任务，从项目环境搭建到商品排行榜浏览逐个实现，其中具体包括了商品列表显示，详细信息，购物车和订单功能等。接下来，我会将关键代码和结果截图一一呈现。

#### 1. 项目环境搭建

该项目用到的 digital 数据库表有 user\_info (用户信息表)、order\_info (订单信息表)、order\_detail (订单明细表)，product\_info (产品信息表)、type (产品信息表)，用到的数据库是 MySQL8.0.16，部署服务器是 Tomcat8.0，在 digital-my 项目中添加 jar 包，其中包括 Struts 2、Hibernate 5 和 Spring 4 需要的关键 jar 包，具体如下：

Web App Libraries	
>	antlr-2.7.7.jar - D:\Program Files (x86)\MyEclips
>	aopalliance-1.0.jar - D:\Program Files (x86)\MyE
>	asm-3.3.jar - D:\Program Files (x86)\MyEclipse2
>	asm-commons-3.3.jar - D:\Program Files (x86)\M
>	asm-tree-3.3.jar - D:\Program Files (x86)\MyEcli
>	aspectjweaver-1.8.9.jar - D:\Program Files (x86)
>	c3p0-0.9.2.1.jar - D:\Program Files (x86)\MyEcli
>	cglib-3.1.jar - D:\Program Files (x86)\MyEclipse2
>	commons-fileupload-1.4.jar - D:\Program Files (
>	commons-io-2.2.jar - D:\Program Files (x86)\My
>	commons-lang3-3.2.jar - D:\Program Files (x86)
>	commons-logging-1.1.3.jar - D:\Program Files (
>	dom4j-1.6.1.jar - D:\Program Files (x86)\MyEclip
>	freemarker-2.3.28.jar - D:\Program Files (x86)\M
>	hibernate-commons-annotations-5.0.1.Final.jar
>	hibernate-core-5.0.6.Final.jar - D:\Program Files
>	hibernate-jpa-2.1-api-1.0.0.Final.jar - D:\Progra
>	jandex-2.0.0.Final.jar - D:\Program Files (x86)\M
>	javassist-3.18.1-GA.jar - D:\Program Files (x86)
>	jboss-logging-3.3.0.Final.jar - D:\Program File
>	jboss-transaction-api_1.1_spec-1.0.1.Final.jar
>	log4j-1.2.17.jar - D:\Program Files (x86)\MyE
>	mchange-commons-java-0.2.3.4.jar - D:\Prog
>	mysql-connector-java-8.0.16.jar - D:\Program
>	ognl-3.0.21.jar - D:\Program Files (x86)\MyE
>	spring-aop-4.2.4.RELEASE.jar - D:\Program Fi
>	spring-aspects-4.2.4.RELEASE.jar - D:\Program
>	spring-beans-4.2.4.RELEASE.jar - D:\Program
>	spring-context-4.2.4.RELEASE.jar - D:\Program
>	spring-core-4.2.4.RELEASE.jar - D:\Program F
>	spring-expression-4.2.4.RELEASE.jar - D:\Prog
>	spring-jdbc-4.2.4.RELEASE.jar - D:\Program F
>	spring-orm-4.2.4.RELEASE.jar - D:\Program Fi
>	spring-tx-4.2.4.RELEASE.jar - D:\Program File
>	spring-web-4.2.4.RELEASE.jar - D:\Program Fi
>	spring-webmvc-4.2.4.RELEASE.jar - D:\Program
>	struts2-convention-plugin-2.3.37.jar - D:\Progr
>	struts2-core-2.3.37.jar - D:\Program Files (x86)
>	struts2-dojo-plugin-2.3.24.1.jar - D:\Program
>	struts2-spring-plugin-2.3.37.jar - D:\Program
>	xwork-core-2.3.37.jar - D:\Program Files (x86)

项目的 src 文件夹下文件目录如下：



WebRoot 文件夹下文件目录如下：



项目的 com.digital.entity 包中的实体类属性对应数据库相应表字段的 get/set 方法及 hibernate 映射到数据库相应表的 xml 文件：

User\_info:

```

public class UserInfo {
    private int id;//用户编号
    private String userName;//用户名
    private String password;//用户密码
    private String realName;//真实姓名
    private String address;//家庭住址
    private String email;//电子邮件
    private String regDate;//注册日期
    private String sex;//性别
}

```

UserInfo.hbm.xml:

```

<hibernate-mapping package="com.digital.entity">
    <class name="UserInfo" table="user_info" catalog="digital">
        <id name="id" type="java.lang.Integer"><!--主键对应的id-->
            <column name="id" />
            <generator class="native"></generator>
        </id>
        <property name="userName" type="java.lang.String">
            <column name="userName" length="16" not-null="true"></column>
        </property>
        <property name="password" type="java.lang.String">
            <column name="password" length="16" not-null="true"></column>
        </property>
        <property name="realName" type="java.lang.String">
            <column name="realName" length="20" not-null="true"></column>
        </property>
        <property name="sex" type="java.lang.String">
            <column name="sex" length="2" /></column>
        </property>
        <property name="address" type="java.lang.String">
            <column name="address" length="255" not-null="true"></column>
        </property>
        <property name="email" type="java.lang.String">
            <column name="email" length="20"></column>
        </property>
        <property name="regDate" type="java.lang.String">
            <column name="regDate"></column>
        </property>
    </class>
</hibernate-mapping>

```

Product\_info:

```

public class ProductInfo {
    private Integer id;//编号
    private String code;//商品编码
    private String name;//商品名
    private Type type;//商品类型
    private String brand;//品牌
    private String picture;//商品大图
    private String img2;//第一张小图
    private String img3;//第二张小图
    private String img4;//第三张小图
    private Integer inventory;//库存
    private Double price;//单价
    private String introduce;//简介
    private Integer status;//处理状态
    private Integer credit;//商品积分
    private String shop;//销售店铺
    private String location;//销售地点
    private Set orderDetail=new HashSet(0); //订单明细
    //一个商品可能是多个订单明细里面的内容（一对多）
    //所以用集合来表示
}

```

ProductInfo.hbm.xml:

```

<hibernate-mapping package="com.digital.entity">
    <class name="ProductInfo" table="product_info" catalog="digital">
        <id name="id" type="java.lang.Integer"><!!--主键对应的id-->
            <column name="id" />
            <generator class="native"></generator>
        </id>
        <!-- ProductInfo映射实体类到type的单向多对一的关联-->
        <many-to-one name="type" class="Type" column="tid" lazy="false" fetch="select">
            <property name="code" type="java.lang.String">
                <column name="code" length="16" not-null="true"></column>
            </property>
            <property name="name" type="java.lang.String">
                <column name="name" length="255" not-null="true"></column>
            </property>
            <property name="brand" type="java.lang.String">
                <column name="brand" length="20" not-null="true"></column>
            </property>
            <property name="picture" type="java.lang.String">
                <column name="picture" length="255" not-null="true"></column>
            </property>
            <property name="img2" type="java.lang.String">
                <column name="img2" length="255" not-null="true"></column>
            </property>
            <property name="img3" type="java.lang.String">
                <column name="img3" length="255" not-null="true"></column>
            </property>
            <property name="img4" type="java.lang.String">
                <column name="img4" length="255" not-null="true"></column>
            </property>
            <property name="inventory" type="java.lang.Integer">
                <column name="inventory" not-null="true"></column>
            </property>
            <property name="price" type="java.lang.Double">
                <column name="price" not-null="true" precision="10" scale="0"></column>
            </property>
            <property name="introduce" type="java.lang.String">
                <column name="introduce" not-null="true"></column>
            </property>
            <property name="status" type="java.lang.Integer">
                <column name="status" not-null="true"></column>
            </property>
            <property name="credit" type="java.lang.Integer">
                <column name="credit"></column>
            </property>
            <property name="shop" type="java.lang.String">
                <column name="shop"></column>
            </property>
            <property name="location" type="java.lang.String">
                <column name="location"></column>
            </property>
        <set name="orderDetail" cascade="delete" inverse="true" lazy="false">
            <key column="pid"/>
            <one-to-many class="OrderDetail"/>
        </set>
    </class>
</hibernate-mapping>

```

### Order\_info:

```

public class OrderInfo {
    private Integer id;//订单编号
    private UserInfo userInfo;//用户信息(匹配用户id)
    private String status;//订单状态
    private String ordertime;//下单时间
    private Double orderprice;//订单总价
    private Set orderDetail=new HashSet(0);//订单明细
    //一个订单信息表可能含有多个订单，每个订单有各自的明细,
    //所以用集合来表示订单明细(一对多)
}

```

### OrderInfo.hbm.xml:

```

<hibernate-mapping package="com.digital.entity">
    <class name="OrderInfo" table="order_info" catalog="digital"><!--主键对应的id -->
        <id name="id" type="java.lang.Integer"><!--主键对应的id -->
            <column name="id" />
            <generator class="native"></generator>
        </id>

        <property name="status" type="java.lang.String">
            <column name="status" length="16" not-null="true"/>
        </property>
        <property name="ordertime" type="java.lang.String">
            <column name="ordertime" length="19" not-null="true"/>
        </property>
        <property name="orderprice" type="java.lang.Double">
            <column name="orderprice" precision="8"/>
        </property>

        <set name="orderDetail" cascade="all" inverse="true" lazy="false">
            <key column="oid"/>
            <one-to-many class="OrderDetail"/>
        </set>
        <!-- userInfo映射实体类到type的单向多对一的关联-->
        <many-to-one name="userInfo" class="UserInfo" column="uid" lazy="false" fetch="select"/>
    </class>
</hibernate-mapping>

```

### Order\_detail:

```

public class OrderDetail {
    private Integer id;//订单明细编号
    private ProductInfo productInfo;//商品信息(匹配商品id)
    private OrderInfo orderInfo;//订单信息 (匹配订单信息id)
    private Integer num;//商品数量
}

```

### OrderDetail.hbm.xml:

```

<hibernate-mapping package="com.digital.entity">
    <class name="OrderDetail" table="order_detail" catalog="digital"><!--主键对应的id -->
        <id name="id" type="java.lang.Integer"><!--主键对应的id -->
            <column name="id" />
            <generator class="native"></generator>
        </id>

        <property name="num" type="java.lang.Integer">
            <column name="num" />
        </property>
        <!-- cascade级联-->多个商品可能是同一订单信息里面的-->
        <many-to-one name="orderInfo" class="OrderInfo" cascade="all" lazy="false" fetch="select">
            <column name="oid"/>
        </many-to-one>
        <!-- ProductInfo实体类映射到type的单向多对一的关联-->多个商品有可能是同一type -->
        <many-to-one name="productInfo" class="ProductInfo" lazy="false" fetch="select">
            <column name="pid"/>
        </many-to-one>
    </class>
</hibernate-mapping>

```

### Type:

```

public class Type {
    private int id;//类型编号
    private String name;//类型名称
    private Set<ProductInfo> pis=new HashSet(0); //商品集合
    //ProductInfoSet
}

```

### Type.hbm.xml:

```

<hibernate-mapping package="com.digital.entity">
    <class name="Type" table="type" catalog="digital"><!--主键对应的id -->
        <id name="id" type="java.lang.Integer"><!--主键对应的id -->
            <column name="id" />
            <generator class="native"></generator>
        </id>

        <property name="name" type="java.lang.String">
            <column name="name" length="15" not-null="true"/>
        </property>

        <!-- 配置一对多关联映射-->一个type有多个product -->
        <set name="pis" inverse="true" lazy="false">
            <key column="tid"/>
            <one-to-many class="ProductInfo"/>
        </set>
    </class>
</hibernate-mapping>

```

### applicationContext.xml 配置 Hibernate 及事务通知和 AOP 配置：

```
<!-- 配置数据源DataSource 在hibernate配置文件中-->
<bean id="dataSource" class="com.mchange.v2.c3p0.ComboPooledDataSource">
    <property name="driverClass" value="com.mysql.cj.jdbc.Driver"/>
    <property name="jdbcUrl" value="jdbc:mysql:///digital?serverTimezone=GMT"/>
    <property name="user" value="root"/>
    <property name="password" value="zdz199804103033"/>
    <property name="minPoolSize" value="5"/>
    <property name="maxPoolSize" value="10"/>
</bean>

<!-- 配置hibernate的sessionfactory -->
<bean id="sessionFactory" class="org.springframework.orm.hibernate5.LocalSessionFactoryBean">
    <!-- 配置数据源属性 -->
    <property name="dataSource">
        <ref bean="dataSource"/>
    </property>
    <!-- 配置hibernate基本属性 -->
    <property name="hibernateProperties">
        <props>
            <prop key="hibernate.dialect">
                org.hibernate.dialect.MySQLDialect
            </prop>
        </props>
    </property>
    <!-- 配置hibernate映射文件的位置及名称 -->
    <property name="mappingResources">
        <list>
            <value>com/digital/entity/UserInfo.hbm.xml</value>
            <value>com/digital/entity/Type.hbm.xml</value>
            <value>com/digital/entity/ProductInfo.hbm.xml</value>
            <value>com/digital/entity/OrderInfo.hbm.xml</value>
            <value>com/digital/entity/OrderDetail.hbm.xml</value>
        </list>
    </property>
</bean>

<!--声明hibernate事务管理器 -->
<bean id="transactionManager" class="org.springframework.orm.hibernate5.HibernateTransactionManager">
    <property name="sessionFactory" ref="sessionFactory"/>
</bean>

<!-- 定义事务通知 需要事务管理器 -->
<tx:advice id="txAdvice" transaction-manager="transactionManager">
    <!-- 指定事务传播规则 -->
    <tx:attributes>
        <!-- 对业务中的所有方法应用REQUIRED事务规则 -->
        <tx:method name="*" propagation="REQUIRED"/>
    </tx:attributes>
</tx:advice>

<!-- 定义切面（AOP配置），并将事务通知和切面组合（定义哪些方法应用哪些规则）-->
<aop:config>
    <!-- 配置切点，对com.digital.service 包下的所有类的所有方法都应用事务规则-->
    <aop:pointcut expression="execution(* com.digital.service.*.*(..))" id="serviceMethods"/>
    <!-- 将事务通知和切点组合 -->
    <aop:advisor advice-ref="txAdvice" pointcut-ref="serviceMethods"/>
</aop:config>
```

## 2. 前台商品列表显示

DAO 层：

```
public interface ProductInfoDAO {
    //得到所有商品列表
    public List<ProductInfo> getAll();
```

DAO 的实现类：(调用 sessionFactory)

```
@Override
public List<ProductInfo> getAll() {
    List<ProductInfo> pList=null;
    //通过sessionfactory获取session
    Session session=sessionFactory.getCurrentSession();
    //创建Criteria对象
    Criteria criteria=session.createCriteria(ProductInfo.class);
    //执行查询，获取对象
    pList=criteria.list();
    return pList;
}
```

Service 层：

```
public interface ProductInfoService {
    //得到所有商品信息
    public List<ProductInfo> getAllProductInfo();
```

Service 实现类：(调用 DAO 接口)

```

ProductInfoDAO productInfoDAO;
public void setProductInfoDAO(ProductInfoDAO productInfoDAO) {
    this.productInfoDAO = productInfoDAO;
}

@Override
public List<ProductInfo> getAllProductInfo() {
    return productInfoDAO.getAll();
}

```

Action 类: (ProductInfoAction 调用 service 接口)

```

public String list() throws Exception{
    List<Type> typeList=typeService.getAllWithDistinctBrand();
    if(typeList.size()>0){
        request.put("typeList", typeList);
    }
    List<ProductInfo> piList=productInfoService.getAllProductInfo();
    if(piList.size()>0){
        request.put("piList", piList);
    }
}

```

applicationContext.xml 实例化 bean, 并为 sessionFactory、DAO、Service 接口属性注入值:

```

<bean id="productInfoDAO" class="com.digital.dao.impl.ProductInfoDAOImpl">
    <property name="sessionFactory" ref="sessionFactory"></property>
</bean>

<bean id="productInfoService" class="com.digital.service.impl.ProductInfoServiceImpl">
    <property name="productInfoDAO" ref="productInfoDAO"></property>
</bean>

<!-- 定义productinfoaction类, 并为其中的productInfoService属性注入值 -->
<bean name="piAction" class="com.digital.action.ProductInfoAction" scope="prototype">
    <property name="productInfoService" ref="productInfoService"></property>
    <property name="typeService" ref="typeService"></property>
</bean>

```

Struts 配置文件:

```

<!-- 商品类型及列表 -->
<action name="list" class="piAction" method="list">
    <result name="index"/>index.jsp</result>
</action>

```

Jsp 用 EL 表达式获取 Action 中得到值:

```

<s:iterator id="piItem" value="#request.piList">
    <div class="product-info">
        <div class="product-picture-box">
            <a href="display?pi.id=${piItem.id}">
                
            </a>
        </div>
        <div class="product-price-box">
            <div class="product-price">
                <strong>¥${piItem.price}</strong>
            </div>
        </div>
        <div class="product-intro-box">
            <a href="display?pi.id=${piItem.id}">
                ${piItem.introduce}
            </a>
        </div>
        <div class="product-shop-location">
            <div class="product-shop">
                <i class="fa fa-bars" style="color: #FB5300; "></i>
                <a href="#">
                    <span>${piItem.shop}</span>
                </a>
            </div>
            <div class="product-location">${piItem.location}</div>
        </div>
        <div class="wangwang">
            <a href="addToShopCart?productId=${piItem.id}"><i clas
            <a href="#">
    </div>
</s:iterator>

```

## 功能界面实现：(在此只显示部分商品)

消息 网站首页 注册 登录
我的唯购 购物车 收藏夹 我的订单 网站导航



**商品服务分类**

- [电脑 >](#)
- [冰箱 >](#)
- [电视机 >](#)
- [羽毛球拍 >](#)
- [数码相机 >](#)
- [运动鞋 >](#)
- [手机 >](#)

宝贝  高级搜索
搜索

[天猫](#) [聚划算](#) [玩转二手](#) [拍卖](#) [3C数码](#) [电器城](#) [服装](#) [旅行](#) [手机电脑](#)



**入门级定制  
聚享羽拍套装**  
**赠199大礼包**

① 李宁原装拍包\*1  
② 耐打羽线\*2  
③ 李宁手胶\*2



**惠普132nw激光一体机**  
**满1479减120  
无线三合一升级款**  
**到手价: 1359**



**家用彩色连供打印机**  
**打印/复印/扫描  
自动双面**  
**到手价: 648**



**Panasonic 双面打印多功能一体机**  
**KX-BP160 简便 打印  
复印 扫描**  
**活动期间: 8.25-8.26**



**6.6包邮!!  
2014淘宝养车节  
车品低至1折/买就送**



**长安带你逛车展  
送1500元新车养护礼包  
长安全系普惠**

所有宝贝 天猫 二手 浏览排行榜



**MacBook Air**  
十四年五皇冠老店 品质之选  
南京深圳多仓发货

**¥9999.0**

下单送多重好礼 (可叠加)  
南京实体店 支持自提



**Lenovo 联想 全球热卖**  
**拯救者Y7000**  
**正装野兽 游戏玩家**  
**小红点电竞屏**  
**联想拯救者Y7000 2019款游戏本笔记本电脑**  
**九代酷睿i5-9300H/16G/256G SSD/MQD32CH/A 超薄笔记本电脑灰银金**

**¥4199.0**

下单立减100元  
收藏加购送豪华礼包  
抽1000元购物津贴



**能玩吃鸡的超薄本**  
7秒极速开机 512G固态  
到手价: 4399

购机送多重豪礼  
3期免息 同城闪送



**联想拯救者Y7000 2019款游戏本笔记本电脑**  
**i5-9300H/16G/256G SSD/MQD32CH/A 超薄笔记本电脑灰银金**

**¥6299.0**

联想拯救者Y7000 2019款游戏本笔记本电脑  
九代酷睿i5-9300H/16G/256G SSD/MQD32CH/A 超薄笔记本电脑灰银金



**Microsoft 微软中国官方旗舰店**  
**Surface Pro 6 i7 8G 256GB 学生豪华版**  
**超2K分辨率触摸屏  
企业级面部识别登录**

**¥10788.0**

微软 Surface Book Pro 6 笔记本电脑 12.3寸  
英特尔第八代酷睿i7/8G/512G/GTX1060-8G  
独显

### 3. 商品详细信息查看

DAO 层：

```
//根据商品ID查询商品信息
public ProductInfo getProductInfoByPiId(int piId);
```

DAO 的实现类：(调用 sessionFactory)

```
@Override
public ProductInfo getProductInfoByPiId(int piId) {
    Session session=sessionFactory.getCurrentSession();
    return (ProductInfo)session.get(ProductInfo.class, piId);
}
```

Service 层：

21

```
//根据商品id得到商品的信息  
public ProductInfo getProductInfoByPiId(int piId);
```

Service 实现类: (调用 DAO 接口)

```
@Override  
public ProductInfo getProductInfoByPiId(int piId) {  
    return productInfoDAO.getProductInfoByPiId(piId);  
}
```

Action: (ProductInfoAction 调用 service 接口)

```
public String display() throws Exception{  
    ProductInfo detailProductInfo=productInfoService.getProductInfoByPiId(pi.getId());  
    request.put("detailProductInfo", detailProductInfo);
```

applicationContext.xml 实例化 bean, 并为 sessionFactory、DAO、Service 接口属性注入值:

此处因为还是调用 ProductInfoService 接口, 故 Spring 配置和子任务 2 中的配置一样。

Struts 配置文件:

```
<!-- 商品详细信息展示 -->  
<action name="display" class="piAction" method="display">  
    <result name="display"/>/display.jsp</result>  
</action>
```

Jsp 页面用 EL 表达式获取 Action 中得到的值:

```

<div class="product-detail-wrapper">
  <div class="product-detail-img">
    <div class="box">
      <div class="productImg-smallBox">
        
        <div class="mask"></div>
      </div>
      <div class="productImg-bigBox"> 
    </div>
    <div class="product-small-img">
      <span>
        
        
        
        
      </span>
    </div>
    <div class="product-collect-share">
      <a href="#"> <i class="fa fa-share-square-o"> 分享</i></a>&ampnbsp
      <a href="#"> <i class="fa fa-star-o"> 收藏</i></a>&ampnbsp
      <a href="#"> <i class="fa fa-bars"> ${requestScope.detailProductInfo.shop} </i></a>&ampnbsp
    </div>
  </div>
  <div class="product-detail-others">
    <div class="product-detail-intro">
      <span>${requestScope.detailProductInfo.introduce}</span>
    </div>
    <div class="product-detail-item">
      选购商品&ampnbsp <span>${requestScope.detailProductInfo.name}</span>
    </div>
    <div class="product-detail-item">
      商品编号&ampnbsp <span>${requestScope.detailProductInfo.code}</span>
    </div>
    <div class="product-detail-price">
      销售价格&ampnbsp <span>¥${requestScope.detailProductInfo.price}</span>
    </div>
    <div class="product-detail-item">
      库存数量&ampnbsp <span>${requestScope.detailProductInfo.inventory} 件 </span>
    </div>
    <div class="product-detail-item">
      选购数量&ampnbsp
      <span>
        <input style="width: 30px;text-align:center;" type="text" readonly="readonly" value="1"/>
      </span>
    </div>
    <div class="product-detail-price">
      小计金额&ampnbsp
      <span>¥${requestScope.detailProductInfo.price}</span>
    </div>
    <div class="product-detail-item product-detail-end">
      <ul>
        <li><a href="#">立即购买</a></li>
        <li>
          <a href="addToShopCart?productId=${requestScope.detailProductInfo.id}">
            <i class="fa fa-shopping-cart"></i>加入购物车</a>
          </li>
        </ul>
    </div>
  </div>
</div>

```

功能界面实现：

#### 4. 用户购物车管理

该子任务中包含的主要购物车操作是，“结算”（即生成订单），“在购物车页面修改购买的商品数量”，“在网站首页或商品详情页将商品添加到购物车”，“在购物车页面点击商品图片或者文字简介显示商品详情”，“清空购物车”，“删除购物车中的某一商品”。

DAO 层：

```
//根据商品ID查询商品信息
public ProductInfo getProductInfoByPiId(int piId);
```

DAO 的实现类：（调用 sessionFactory）

```
@Override
public ProductInfo getProductInfoByPiId(int piId) {
    Session session=sessionFactory.getCurrentSession();
    return (ProductInfo)session.get(ProductInfo.class, piId);
}
```

Service 层：

```
//根据商品id得到商品的信息
public ProductInfo getProductInfoByPiId(int piId);
```

Service 实现类：（调用 DAO 接口）

```
@Override
public ProductInfo getProductInfoByPiId(int piId) {
    return productInfoDAO.getProductInfoByPiId(piId);
}
```

Action 类：（CartAction 调用 service 接口）

```

public String addToShopCart() {
    //从session中取出购物车，放入map对象的cart中
    Map cart=(Map) session.get("cart");
    //获取当前要添加至购物车的商品
    ProductInfo productInfo=(ProductInfo) productService.getProductInfoById(productId);
    //如果购物车不存在，则创建购物，并存入到session中
    if(cart==null){
        cart=new HashMap();
        session.put("cart", cart);
    }
    //如果存在购物车则判断要添加的商品是否已在购物车
    CartItemBean cartItem=(CartItemBean)cart.get(productInfo.getId());
    if(cartItem!=null){
        //如果商品在购物车中，则更新其数量即可
        cartItem.setQuantity(cartItem.getQuantity()+1);
    }else{
        //否则创建一个商品条目至购物车中
        cart.put(productId, new CartItemBean(productInfo, 1));
    }
    //页面跳转到cart.jsp
    return "shopcart";
}

```

```

//更新修改后的商品数量
public String updateProductQuantity() {
    Map cart=(Map) session.get("cart");
    CartItemBean cartItem=(CartItemBean)cart.get(productId);
    cartItem.setQuantity(quantity);
    return "shopcart";
}
//删除选中的购物车商品项
public String deleteSelectedItem() {
    Map cart=(Map) session.get("cart");
    cart.remove(productId);
    return "shopcart";
}
//清空购物车
public String clearCart() {
    Map cart=(Map) session.get("cart");
    cart.clear();
    return "shopcart";
}

Map<String, Object>request;
@Override
public void setRequest(Map<String, Object> request) {
    this.request=request;
}

//在购物车中点击商品的图片或简介查看商品详情
public String showDetailInCart(){
    ProductInfo detailProductInfo=productService.getProductInfoById(productId);
    request.put("detailProductInfo", detailProductInfo);
    return "display";
}

```

applicationContext.xml 实例化 bean，并为 sessionFactory、DAO、Service 接口属性注入值：

在 CartAction 中调用的业务层还是 ProductInfoService，具体 Spring 配置同子任务 2。

Struts 配置文件：

```

<!-- 加入购物车-->
<action name="addToShopCart" class="cartAction" method="addToShopCart">
    <result name="shopcart">/cart.jsp</result>
    <interceptor-ref name="loginCheck"/>
    <interceptor-ref name="defaultStack"/>
</action>

```

```

<!-- 更新购物车中选择的商品数量 可输入数量后回车 -->
<action name="updateSelectedQuantity" class="cartAction" method="updateSelectedQuantity">
    <result name="shopcart">/cart.jsp</result>
</action>

```

```

<!-- 删除选中的购物车商品项 -->
<action name="deleteSelectedItem" class="cartAction" method="deleteSelectedItem">
    <result name="shopcart">/cart.jsp</result>
</action>

```

```
<!-- 清空购物车 -->
<action name="clearCart" class="cartAction" method="clearCart">
    <result name="shopcart"/>/cart.jsp</result>
</action>
```

```
<!-- 展示购物车订单页面的商品详细信息 -->
<action name="showDetailInCart" class="cartAction" method="showDetailInCart">
    <result name="display"/>/display.jsp</result>
</action>
```

### Jsp 页面 EL 表达式获取 Action 中获取的值:

```
<s:set var="sumPrice" value="0"/>
<s:set var="sumNum" value="0"/>
<s:iterator id="cartItem" value="#session.cart">
<tr>
    <th colspan="8" class="shopInfo">店铺:</th>
        <a href="#"><s:property value="value.pi.shop"/></a>
        
</tr>
<tr id="product1">
    <td class="cart_td_1">
        <input name="cartCheckBox" type="checkbox" value="product1" onclick="selectSingle()" />
    </td>
    <td class="cart_td_2">
        
    </td>
    <td class="cart_td_3">
        <a href="showDetailInCart?productId=${value.pi.id}"><s:property value="value.pi.introduce"/></a>
    </td>
    <td class="cart_td_4"><s:property value="value.pi.credit"/></td>
    <td class="cart_td_5">¥<s:property value="value.pi.price"/></td>
    <td class="cart_td_6">
        <input id="num" type="text" value="${value.quantity}" class="num_input"
            onchange="window.location='updateSelectedQuantity?productId=${value.pi.id}&quantity='+this.value;" />
    </td>
    <td class="cart_td_7"><s:property value="value.quantity*value.pi.price"/></td>!— 小计 —>
    <td class="cart_td_8"><a href="deleteSelectedItem?productId=${value.pi.id}">删除</a></td>
</tr>
<s:set var="sumPrice" value="#sumPrice+value.quantity*value.pi.price"/></s:set>
<s:set var="sumNum" value="#sumNum+value.quantity"/></s:set>
</s:iterator>
```

```
<tr>
    <td class="shopend" colspan="2">
        <span><a href="clearCart">清空购物车</a></span>
        <td class="shopend" colspan="2">
            <span>已选商品</span>
            <span style="font-size: 20px; color:#FB5B0B;"> <s:property value="#sumNum"/> </span>件
    </td>
    <td class="shopend" colspan="2">
        <span>合计:(不含运费)</span>
        <span style="color:#FB5B0B;">¥<s:property value="#sumPrice"/>
        <s:set var="sumPrice" value="#sumPrice" scope="session"></s:set>
    </span>
    </td>
    <td class="settle" colspan="2">
        <a href="addOrderInfo">结 &ampnbsp算</a>
    </td>
</tr>
```

### 功能界面实现:

The screenshot shows a shopping cart page with the following details:

我的购物车						
您的位置: 首页 > 我的喵购 > 我的购物车		店铺: 华硕官方旗舰店		操作		
<input type="checkbox"/> 全选	店铺宝贝	积分	单价(元)	数量	小计(元)	
<input type="checkbox"/>	华硕灵耀S2代S4300FN 8代i5超薄笔记本电脑 英特尔酷睿八代i5-8265U MX150-2G独显	400	¥6299.0	<input type="button" value="1"/>	6299.0	<a href="#">删除</a>
<input type="checkbox"/>	捷高子净味 滚筒变频风冷无霜	224	¥2690.0	<input type="button" value="1"/>	2690.0	<a href="#">删除</a>
<a href="#">清空购物车</a>			已选商品 2 件		合计:(不含运费) ￥8989.0	
<a href="#">结算</a>						

消息 网站首页 注册 登录 我的购物车 购物车 收藏夹 我的订单 网站导航

## 我的购物车

您的位置：首页 > 我的购物车 > 我的购物车

全选	店铺宝贝	积分	单价（元）	数量	小计（元）	操作
<input type="checkbox"/>	 <p>松下 NR-EC25WS1-N 特价三开门智能家用三门式电冰箱节能风冷无霜</p>	224	¥2690.0	<input type="text" value="1"/>	2690.0	<a href="#">删除</a>

[清空购物车](#) 已选商品 1 件 合计:(不含运费) ￥2690.0 [结算](#)

苏宁易购 | Panasonic 7天包退 15天包换 松下电器

255L 银离子净味 风冷无霜 节能 8月23日-8月26日

到手价：**¥ 2690** 7天包退 15天包换 立即抢购> 赠压缩机5年保修



松下 NR-EC25WS1-N 特价三开门智能家用三门式电冰箱节能风冷无霜

选购商品 松下 冰箱

商品编号 888888

销售价格 **¥ 2690.0**

库存数量 4455件

选购数量

小计金额 **¥ 2690.0**

浏览排行榜 联想拯救者Y7000

消息 网站首页 注册 登录 我的购物车 购物车 收藏夹 我的订单 网站导航

## 我的购物车

您的位置：首页 > 我的购物车 > 我的购物车

全选	店铺宝贝	积分	单价（元）	数量	小计（元）	操作
<input type="checkbox"/>	<a href="#">清空购物车</a>	已选商品 0 件	合计:(不含运费) ￥0	<a href="#">结算</a>		

## 5. 订单功能

该子任务主要包括“删除选定的订单”，“查看订单明细”。

DAO 层：

```
public interface OrderInfoDAO {
    //获取指定用户的订单
    public List<OrderInfo> getOrderInfoByUserInfoId(int userInfoId);
    //根据订单编号加载订单
    public OrderInfo getInfoById(int id);
    //删除订单
    public void deleteOrderInfo(OrderInfo orderInfo);
}
```

```
public interface OrderDetailDAO {
    //由订单信息表的id得到订单的详细信息
    public List<OrderDetail> getOrderDetailById(int id);
    //查看订单明细
    public void lookOrderDetail(OrderDetail orderDetail);
}
```

### DAO 的实现类: (调用 sessionFactory)

```
public class OrderInfoDAOImpl implements OrderInfoDAO {  
    SessionFactory sessionFactory;  
  
    public void setSessionFactory(SessionFactory sessionFactory) {  
        this.sessionFactory = sessionFactory;  
    }  
    //根据用户id得到其订单列表  
    @Override  
    public List<OrderInfo> getOrderInfoByUserInfoId(int userInfoId) {  
        Session session=sessionFactory.getCurrentSession();  
        Criteria criteria=session.createCriteria(OrderInfo.class);  
        criteria.add(Restrictions.eq("userInfo.id", userInfoId));  
        return criteria.list();  
    }  
    //根据订单编号得到订单信息  
    @Override  
    public OrderInfo getInfoById(int id) {  
        Session session=sessionFactory.getCurrentSession();  
        OrderInfo orderInfo=(OrderInfo)session.get(OrderInfo.class, id);  
        return orderInfo;  
    }  
    //删除订单信息  
    @Override  
    public void deleteOrderInfo(OrderInfo orderInfo) {  
        Session session=sessionFactory.getCurrentSession();  
        session.delete(orderInfo);  
    }  
}
```

```
public class OrderDetailDAOImpl implements OrderDetailDAO {  
    SessionFactory sessionFactory;  
  
    public void setSessionFactory(SessionFactory sessionFactory) {  
        this.sessionFactory = sessionFactory;  
    }  
    //查看订单明细  
    @Override  
    public void lookOrderDetail(OrderDetail orderDetail) {  
        Session session=sessionFactory.getCurrentSession();  
        session.saveOrUpdate(orderDetail);  
    }  
    //根据订单的id得到订单明细  
    @Override  
    public List<OrderDetail> getOrderDetailById(int id) {  
        Session session=sessionFactory.getCurrentSession();  
        Criteria criteria=session.createCriteria(OrderDetail.class);  
        criteria.add(Restrictions.eq("orderInfo.id", id));  
        return criteria.list();  
    }  
}
```

### Service 层:

```
public interface OrderInfoService {  
    //根据用户id获得用户的订单信息表  
    public List<OrderInfo> getOrderInfoByUserInfoId(int userInfoId);  
    //删除指定订单编号的订单  
    public void deleteOrderInfoById(int id);  
}
```

```
public interface OrderDetailService {  
    //查看订单明细  
    public void lookOrderDetail(OrderDetail orderDetail);  
    //根据订单信息表的编号得到订单的详细信息  
    public List<OrderDetail> getOrderDetailById(int id);  
}
```

### Service 的实现类: (调用 DAO 接口)

```

public class OrderInfoServiceImpl implements OrderInfoService {
    OrderInfoDAO orderInfoDAO;
    public void setOrderInfoDAO(OrderInfoDAO orderInfoDAO) {
        this.orderInfoDAO = orderInfoDAO;
    }
    @Override
    public List<OrderInfo> getOrderInfoByUserInfoId(int userInfoId) {
        return orderInfoDAO.getOrderInfoByUserInfoId(userInfoId);
    }
    @Override
    public void deleteOrderInfoById(int id) {
        OrderInfo orderInfo=orderInfoDAO.getInfoById(id);
        orderInfoDAO.deleteOrderInfo(orderInfo);
    }
}

```

```

public class OrderDetailServiceImpl implements OrderDetailService {
    OrderDetailDAO orderDetailDAO;
    public void setOrderDetailDAO(OrderDetailDAO orderDetailDAO) {
        this.orderDetailDAO = orderDetailDAO;
    }
    //查看订单明细
    @Override
    public void lookOrderDetail(OrderDetail orderDetail) {
        orderDetailDAO.lookOrderDetail(orderDetail);
    }
    //根据订单id(oid)得到订单明细
    @Override
    public List<OrderDetail> getOrderDetailById(int id) {
        return orderDetailDAO.getOrderDetailById(id);
    }
}

```

Action 类: (OrderAction 调用 service 接口)

```

//添加订单
public String addOrderInfo() throws Exception{
    OrderInfo orderInfo=new OrderInfo();
    //封装orderInfo的实体对象
    orderInfo.setStatus("未处理");
    //取得当前系统时间
    SimpleDateFormat simpleDateFormat=new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
    orderInfo.setOrderTime(simpleDateFormat.format(new Date()));
    //取得当前登录用户
    UserInfo userInfo=(UserInfo)session.get("CURRENT_USER");
    orderInfo.setUserInfo(userInfo);
    //取得存放在session中的sumPrice总金额
    orderInfo.setOrderPrice((Double)session.get("sumPrice"));
    //取得购物车对象
    Map cart=(HashMap)session.get("cart");
    Iterator it=cart.keySet().iterator();
    try{
        while(it.hasNext()){
            Object key=it.next();
            CartItemBean cartItem=(CartItemBean)cart.get(key);
            //设置订单明细
            OrderDetail orderDetail=new OrderDetail();
            orderDetail.setProductInfo(cartItem.getPi());
            orderDetail.setNum(cartItem.getQuantity());
            orderDetail.setOrderInfo(orderInfo);
            //查看订单明细
            orderDetailService.lookOrderDetail(orderDetail);
        }
    }catch (Exception e) {
        e.printStackTrace();
    }
    //已经生成订单，所以删除购物车
    session.remove("cart");
    return "index";
}

```

```

OrderInfoService orderInfoService;

public void setOrderInfoService(OrderInfoService orderInfoService) {
    this.orderInfoService = orderInfoService; //set方法注入值
}

//获取指定用户的订单列表
public String toMyOrderInfo() {
    //获取指定用户
    UserInfo userInfo = (UserInfo) session.get("CURRENT_USER");
    //获取指定用户的订单列表
    List myOrderInfoList = orderInfoService.getOrderByUserInfo(userInfo.getId());
    request.put("myOrderInfoList", myOrderInfoList);
    return "myOrderInfo";
}

int id;

public int getId() {
    return id;
}

public void setId(int id) {
    this.id = id;
}

//根据订单信息表的编号得到订单的详细信息
public String toOrderDetail() {
    List orderDetailList = orderDetailService.getOrderDetailById(id);
    request.put("orderDetailList", orderDetailList);
    return "toOrderDetail";
}

//删除指定编号的订单
public String deleteSelectedOrderInfo() {
    orderInfoService.deleteOrderInfoById(id);
    return "myOrderInfo";
}

```

applicationContext.xml 实例化 bean，并为 sessionFactory、DAO、Service 接口属性注入值：

```

<bean id="orderDetailDAO" class="com.digital.dao.impl.OrderDetailDAOImpl">
    <property name="sessionFactory" ref="sessionFactory"></property>
</bean>

<bean id="orderInfoDAO" class="com.digital.dao.impl.OrderInfoDAOImpl">
    <property name="sessionFactory" ref="sessionFactory"></property>
</bean>

<bean id="orderDetailService" class="com.digital.service.impl.OrderDetailServiceImpl">
    <property name="orderDetailDAO" ref="orderDetailDAO"></property>
</bean>

<bean id="orderInfoService" class="com.digital.service.impl.OrderInfoServiceImpl">
    <property name="orderInfoDAO" ref="orderInfoDAO"></property>
</bean>

<bean name="orderAction" class="com.digital.action.OrderAction" scope="prototype">
    <property name="orderDetailsService" ref="orderDetailService"></property>
    <property name="orderInfoService" ref="orderInfoService"></property>
</bean>

```

Struts 配置文件：

```

<!-- 添加订单信息 之后返回首页 -->
<action name="addOrderInfo" class="orderAction" method="addOrderInfo">
    <result name="index" type="redirectAction">list</result>
</action>
<!-- 得到我的订单信息 -->
<action name="toMyOrderInfo" class="orderAction" method="toMyOrderInfo">
    <result name="myOrderInfo">/order.jsp</result>
</action>
<!-- 得到详细订单列表 -->
<action name="toOrderDetail" class="orderAction" method="toOrderDetail">
    <result name="toOrderDetail">/orderdetail.jsp</result>
</action>
<!-- 删除指定编号的订单 -->
<action name="deleteSelectedOrderInfo" class="orderAction" method="deleteSelectedOrderInfo">
    <result name="myOrderInfo" type="redirectAction">toMyOrderInfo</result>
</action>

```

Jsp 页面 EL 表达式/struts 标签获取 Action 中得到的值：

```

<s:set var="total" value="0"/>
<s:iterator id="myOrder" value="#request.myOrderInfoList">
<tr id="product1">
    <td class="cart_td_1">
        <input name="cartCheckBox" type="checkbox" value="product1" onclick="selectSingle()" />
    </td>
    <td class="cart_td_2"></td>
    <td class="cart_td_3" style="text-align: center;"><s:property value="ordertime"/></td>
    <td class="cart_td_4">5</td>
    <td class="cart_td_5" style="font-size: 16px; color: orange; font-weight: bold;">¥<s:property value="orderprice"/>
    <td class="cart_td_6"><s:property value="status"/></td>
    <td class="cart_td_7"><a href="#" toOrderDetail?id=${id}>查看</a></td>
    <td class="cart_td_8">
        <s:if test="#myOrder.status=='未处理'">
            <a href="#" deleteSelectedOrderInfo?id=${id}>删除</a>
        </s:if>
    </td>
</tr>
<tr><td colspan="8" class="line"></td></tr>
<s:set var="total" value="#total+orderprice"/>
</s:iterator>

<tr><td colspan="8" class="line"></td></tr>
<tr><td class="blank"></td></tr>
<tr>
    <td class="shopend" colspan="6">
        <span>合计:(不含运费)</span>
        <span style="color: darkorange; font-size: 20px;">¥<s:property value="#total"/></span>
    </td>
    <td class="settle" colspan="2">
        <a href="#">提交订单</a>
    </td>
</tr>

```

```

<s:set var="total" value="0"/>
<s:iterator id="orderDetailList" value="#request.orderDetailList">
<tr>
    <th colspan="8" class="shopInfo">店铺:</th>
        <a href="#"><s:property value="productInfo.shop"/></a>
        
</tr>
<tr id="product1">
    <td class="cart_td_1">
        <span><s:property value="id"/></span><i class="fa fa-star fa-orange"></i></td>
    </td>
    <td class="cart_td_2">
        
    </td>
    <td class="cart_td_3">
        <a href="#"><s:property value="productInfo.introduce"/></a>
    </td>
    <td class="cart_td_4"><s:property value="productInfo.credit*num"/></td>
    <td class="cart_td_5"><s:property value="productInfo.price"/></td>
    <td class="cart_td_6">
        <span><s:property value="num"/></span>
    </td>
    <td class="cart_td_7" colspan="2" style="text-align: right; padding-right: 10px;">¥<s:property value="productInfo.price*num"/></td>
</tr>
<tr><td class="blank"></td></tr>
<s:set var="total" value="#total+productInfo.price*num"/>
</s:iterator>
<tr><td colspan="8" class="line"></td></tr>
<tr><td class="blank"></td></tr>
<tr>
    <td class="shopend" colspan="8" style="text-align: right; padding-right: 10px;">
        <span>合计:(不含运费)</span>
        <span style="color: #fe6400; font-size: 19px;">¥<s:property value="#total"/></span>
    </td>
</tr>

```

## 功能界面实现：

我的订单						
您的位置：首页 > 我的喵购 > 我的订单						
<input type="checkbox"/> 全选	下单时间	总积分	小计 (元)	订单状态	订单明细	
<input type="checkbox"/>	2019-09-04 22:45:07	5	¥30782.0	未处理	<a href="#">查看</a>	<a href="#">删除</a>
<input type="checkbox"/>	2019-09-04 22:45:40	5	¥13135.0	未处理	<a href="#">查看</a>	<a href="#">删除</a>

合计:(不含运费) ¥43917.0 [提交订单](#)

消息 网站首页 注册 登录 我的淘宝 购物车 收藏夹 我的订单 网站导航

## 订单明细

您的位置：首页 > 我的喵购 > 我的订单 > 订单明细

编号	店铺宝贝	积分	单价(元)	数量	小计(元)
83 ★	联想拯救者Y7000 2019款游戏本笔记本电脑九代酷睿i5吃鸡15.6英寸 i5-9300H/16G/256G固态/GTX1650-4G独显	1800	4199.0	4	¥16796.0
84 ★	华硕灵耀S2代S4300FN 8代i5超薄笔记本电脑 英特尔酷睿八代i5-8265U MX150-2G独显	400	6299.0	1	¥6299.0
85 ★	小米mi9 6+128G 骁龙855，索尼4800W三摄 绝对的性价比，绝对的运行流畅度	230	2599.0	1	¥2599.0
86 ★	苹果iPhone XR 128G A12仿生芯片超流畅iOS系统 后置1200W双摄 retina视网膜屏 提升你的视听体验	300	5088.0	1	¥5088.0
合计:(不含运费) <b>¥30782.0</b>					

消息 网站首页 注册 登录 我的喵购 购物车 收藏夹 我的订单 网站导航

## 我的订单

您的位置：首页 > 我的喵购 > 我的订单

<input type="checkbox"/> 全选	下单时间	总积分	小计(元)	订单状态	订单明细	操作
<input type="checkbox"/>	2019-09-04 22:45:40	5	¥13135.0	未处理	<a href="#">查看</a>	<a href="#">删除</a>
合计:(不含运费) <b>¥13135.0</b>						<a href="#">提交订单</a>

## 6. 前台页面浏览排行榜

DAO 层：

```
//根据商品id查询商品列表
public List<ProductInfo> getByPids(String pids);
```

DAO 的实现类：(调用 sessionFactory)

```
@Override
public List<ProductInfo> getByPids(String pids) {
    Session session=sessionFactory.getCurrentSession();
    Query query=session.createQuery("from ProductInfo pi where pi.id in "+pids);
    return query.list();
}
```

Service 类：

```
//根据商品ids的字符串查询已经浏览过的商品列表
public List<ProductInfo> getBrowsingProductInfo(String ids);
```

Service 的实现类：(调用 DAO 接口)

```

@Override
public List<ProductInfo> getBrowsingProductInfo(String ids) {
    return productInfoDAO.getByPids(ids);
}

```

### Action 类: (调用 service 接口)

```

public String list() throws Exception{
    List<Type> typeList=typeService.getAllWithDistinctBrand();
    if(typeList.size()>0){
        request.put("typeList", typeList);
    }
    List<ProductInfo> piList=productInfoService.getAllProductInfo();
    if(piList.size()>0){
        request.put("piList", piList);
    }

    //浏览商品的历史信息
    Cookie []cookies=req.getCookies();
    Cookie cookie=null;
    String ids="";
    if(cookies!=null){
        boolean flag=true;
        for(Cookie c:cookies){
            if("BrowsingSample".equals(c.getName())){
                ids=(c.getValue().substring(0,c.getValue().toString().length()-1)+");
                break;
            }
        }
    }
    if(!"".equals(ids)){
        List<ProductInfo> browsePiList=productInfoService.getBrowsingProductInfo(ids);
        if(browsePiList.size()>0){
            //用session存储浏览过的商品列表，生命周期长
            session.put("browsePiList",browsePiList);
        }
    }
    return "index";
}

```

```

public String display() throws Exception{
    ProductInfo detailProductInfo=productInfoService.getProductInfoByPiId(pi.getId());
    request.put("detailProductInfo", detailProductInfo);

    //得到cookies, 浏览过的商品的信息
    Cookie []cookies=req.getCookies();
    Cookie cookie=null;
    String ids="";
    if(cookies!=null){
        boolean flag=true;
        for(Cookie c:cookies){
            if("BrowsingSample".equals(c.getName())){
                flag=false;
                cookie=c;
                break;
            }
        }
        if(flag){
            //如果cookies不存在browsingSample，则创建
            cookie=new Cookie("BrowsingSample", "");
            //设置生命周期
            cookie.setMaxAge(24*60*60);
        }
    }
    if (" ".equals(cookie.getValue()) || String.valueOf(pi.getId()).indexOf(cookie.getValue())<0){
        ids += cookie.getValue()+pi.getId()+" ";
        cookie.setValue(ids);
    }
    resp.addCookie(cookie);
    return "display";
}

```

applicationContext.xml 实例化 bean，并为 sessionFactory、DAO、Service 接口属性注入值：

在 ProductAction 中调用的业务层还是 ProductInfoService，具体 Spring 配置同子任务 2.

### Struts 配置文件:

因为此处的浏览排行榜是在 index 页面和 display 页面实现，所以将它们写在 list()

和 `display()` 方法里，在前面子任务中已经实现，故再在此处不用配置。

### 功能界面实现：

所有宝贝 天猫 二手
浏览排行榜

**MacBook Air**  
十四年五皇冠老店 品质之选  
南京深圳多仓发货

下单多重好礼(可定购)  
**支持自提**

**¥ 9999.0**

苹果Apple MacBook Air 2019新款Apple/苹果 MacBook Air 八代i5 8G/256G SSD/MQD32CH/A 超薄笔记本电脑灰银金

苹果官方旗舰店 广州

**Lenovo 联想 全球来电**

**正装野兽 游戏玩家**  
i5-8300H处理器 8G DDR4内存

领券立减**100元** 收藏加购送豪华礼包 抽1000元购物津贴

**¥ 4199.0**

联想拯救者Y7000 2019款游戏本笔记本电脑九代酷睿i5吃鸡15.6英寸 i5-9300H/16G/256G固态/GTX1650-4G独显

联想官方旗舰店 上海

**能玩吃鸡的超薄本**  
7秒极速开机 512G固态

立省**200元** 购机送多重豪礼 3期免息 同城闪送

**¥ 6299.0**

华硕灵耀S2代S4300FN 8代i5超薄笔记本电脑 英特尔八代酷睿i5-8265U MX150-2G独显

华硕官方旗舰店 南京

**Microsoft**  
微软中国官方旗舰店

**Surface Pro 6 i7 8G 256GB**  
12期 2年 免息 保固 学生专享优惠

超2K分辨率触摸屏 企业级面部识别登录

**¥ 10788.0**

微软 Surface Book Pro6 笔记本电脑 12.3寸 英特尔八代酷睿i7/8G/512G/GTX1060-8G独显

微软官方旗舰店 上海

### 商品详细信息

所有宝贝 天猫 二手
搜索

**Lenovo 联想 全球来电**

**正装野兽 游戏玩家**  
i5-8300H处理器 8G DDR4内存

领券立减**100元** 收藏加购送豪华礼包 抽1000元购物津贴

**¥ 4199.0**

联想拯救者Y7000 2019款游戏本笔记本电脑九代酷睿i5吃鸡15.6英寸 i5-9300H/16G/256G固态/GTX1650-4G独显

选购商品 联想拯救者Y7000

商品编号 123313

销售价格 **¥ 4199.0**

库存数量 97件

选购数量

小计金额 **¥ 4199.0**

**Microsoft**  
微软中国官方旗舰店

**Surface Pro 6 i7 8G 256GB**  
12期 2年 免息 保固 学生专享优惠

超2K分辨率触摸屏 企业级面部识别登录

**¥ 10788.0**

微软 Surface Book Pro6 笔记本电脑 12.3寸 英特尔八代酷睿i7/8G/512G/GTX1060-8G独显

所有宝贝 天猫 二手
浏览排行榜

**MacBook Air**  
十四年五皇冠老店 品质之选  
南京深圳多仓发货

下单多重好礼(可定购)  
**支持自提**

**¥ 9999.0**

苹果Apple MacBook Air 2019新款Apple/苹果 MacBook Air 八代i5 8G/256G SSD/MQD32CH/A 超薄笔记本电脑灰银金

苹果官方旗舰店 广州

**Lenovo 联想 全球来电**

**正装野兽 游戏玩家**  
i5-8300H处理器 8G DDR4内存

领券立减**100元** 收藏加购送豪华礼包 抽1000元购物津贴

**¥ 4199.0**

联想拯救者Y7000 2019款游戏本笔记本电脑九代酷睿i5吃鸡15.6英寸 i5-9300H/16G/256G固态/GTX1650-4G独显

联想官方旗舰店 上海

**能玩吃鸡的超薄本**  
7秒极速开机 512G固态

立省**200元** 购机送多重豪礼 3期免息 同城闪送

**¥ 6299.0**

华硕灵耀S2代S4300FN 8代i5超薄笔记本电脑 英特尔八代酷睿i5-8265U MX150-2G独显

华硕官方旗舰店 南京

**Microsoft**  
微软中国官方旗舰店

**Surface Pro 6 i7 8G 256GB**  
12期 2年 免息 保固 学生专享优惠

超2K分辨率触摸屏 企业级面部识别登录

**¥ 10788.0**

微软 Surface Book Pro6 笔记本电脑 12.3寸 英特尔八代酷睿i7/8G/512G/GTX1060-8G独显

微软官方旗舰店 上海

**小米mi9 6+128G**

华硕灵耀S2代 S4300FN

**华为P30 Pro 6+128G**

**iPhone XR 128G**

**荣耀honor20 Pro 8+128G**

### 7. 其他功能实现

在此子项目中，为了使其能够构成一个完整的运行网站，我还用 SSH2 框架实现了登录、注册、struts2 验证框架进行登录和注册的验证、struts2 国际化、商品类别

34

及品牌的显示、退出网站登录。功能界面实现同“总报告中的基本任务模块”。

在此模块，我只列出登录、注册的核心代码。

(1) 登录：

DAO 层：

```
//寻找登录时用户是否存在  
public List<UserInfo> search(UserInfo user);
```

DAO 的实现类：(调用 sessionfactory)

```
@Override  
public List<UserInfo> search(UserInfo user) {  
    List<UserInfo> uiList=null;  
    Session session=sessionFactory.getCurrentSession();  
    Criteria criteria=session.createCriteria(UserInfo.class);  
    Example example=Example.create(user);  
    criteria.add(example);  
    uiList=criteria.list();  
    return uiList;  
}
```

Service 层：

```
//登录  
public List<UserInfo> login(UserInfo user);
```

Service 的实现类：(调用 DAO 接口)

```
@Override  
public List<UserInfo> login(UserInfo user) {  
    return userInfoDAO.search(user);  
}
```

Action 类：(调用 service 接口)

```
public String login() throws Exception {  
    List<UserInfo> uiList = userInfoService.login(ui);  
    if (uiList.size() > 0) {  
        session.put("CURRENT_USER", uiList.get(0));  
        long currentTime=System.currentTimeMillis();  
        Long startTime=(Long)session.get("startTime");  
        if(startTime==null){  
            startTime=currentTime;  
            session.put("stratTime", startTime);  
        }  
        long usedTime=(currentTime-startTime)/1000/60;  
        if(usedTime>=60){  
            setMessage("您已经访问喵购网"+usedTime+"分钟，起来活动一下呦！");  
        }else if(usedTime==0){  
            setMessage("您已经开始访问喵购网，祝您愉快！");  
        }  
        else{  
            setMessage("您已经访问喵购网"+usedTime+"分钟！");  
        }  
        return "index";  
    } else {  
        // 登陆失败，重定向到login.jsp  
        setMessage("登录失败，请检查用户名和密码是否正确！");  
        return "login";  
    }  
  
    public String logOut(){  
        UserInfo userInfo=(UserInfo)session.get("CURRENT_USER");  
        if(userInfo!=null){  
            session.remove("CURRENT_USER");  
        }  
        return "index";  
    }
```

applicationContext.xml 实例化 bean，并为 sessionFactory、DAO、Service 接口属性注入值：

```
<bean id="userInfoDAO" class="com.digital.dao.impl.UserInfoDAOImpl">
    <property name="sessionFactory" ref="sessionFactory"></property>
</bean>

<bean id="userInfoService" class="com.digital.service.impl.UserInfoServiceImpl">
    <property name="userInfoDAO" ref="userInfoDAO"></property>
</bean>

<bean name="uiAction" class="com.digital.action.UserInfoAction" scope="prototype">
    <property name="userInfoService" ref="userInfoService"></property>
</bean>
```

Struts 配置文件：

```
<!-- 登录 -->
<action name="login" class="uiAction" method="login"><!-- name随便起就可 -->
    <!-- <result name="index" />/index.jsp</result> -->
    <result name="index" type="redirectAction">list</result>
    <result name="login" type="redirect">login.jsp</result>
</action>

<!-- 退出账户登录 -->
<action name="logOut" class="uiAction" method="logOut">
    <result name="index" type="redirectAction">list</result>
</action>
```

(2) 注册：

DAO 层：

```
//注册用户时向数据库添加用户
public int addUser(UserInfo user);
```

DAO 的实现类：(调用 sessionFactory)

```
@Override
public int addUser(UserInfo user) {
    Session session=null;
    try{
        session = sessionFactory.openSession();
        Transaction transaction=session.beginTransaction();
        session.save(user);
        transaction.commit();
    }catch(Exception e){
        e.printStackTrace();
        System.out.println("提交数据出错了");
        return 0;
    }finally {
        session.close();
    }
    return 1;
}
```

Service 层：

```
//注册
public int register(UserInfo user);
```

Service 的实现类：(调用 DAO 接口)

```
@Override
public int register(UserInfo user) {
    return userInfoDAO.addUser(user);
}
```

Action 类：(调用 service 接口)

```

UserInfoService userInfoService;

public void setUserInfoService(UserInfoService userInfoService) {
    this.userInfoService = userInfoService;
}
public String register() throws Exception{
    SimpleDateFormat sdf=new SimpleDateFormat("yyyy-MM-dd");
    user.setRegDate(sdf.format(new Date()));
    UserInfo userInfo=new UserInfo(
        user.getUserName(),
        user.getPassword(),
        user.getRealName(),
        user.getAddress(),
        user.getEmail(),
        user.getRegDate());
    int result=userInfoService.register(userInfo);
    if(result==0){
        return "input";
    }else{
        return "success";
    }
}

```

applicationContext.xml 实例化 bean，并为 sessionFactory、DAO、Service 接口属性注入值：

```

<bean id="userInfoDAO" class="com.digital.dao.impl.UserInfoDAOImpl">
    <property name="sessionFactory" ref="sessionFactory"></property>
</bean>

```

```

<bean id="userInfoService" class="com.digital.service.impl.UserInfoServiceImpl">
    <property name="userInfoDAO" ref="userInfoDAO"></property>
</bean>

```

```

<bean name="registerAction" class="com.digital.action.RegisterAction" scope="prototype">
    <property name="userInfoService" ref="userInfoService"></property>
</bean>

```

Struts 配置文件：

```

<!--注册action-->
<action name="register" class="registerAction" method="register">
    <result name="success">/success.jsp</result>
    <result name="input">/register.jsp</result>
</action>

```

## 四、设计总结

本次“基于 Web 的电子商城系统的设计与实现”项目让我真正认识了 SSH2 框架的内涵和核心，对三个框架从最初认识到逐渐理解并且学会运用，在这过程中虽然有一些自己难以理解的问题，但还是借助网上的一些资料，对自己比较迷惑的地方进行逐个击破。

首先学习并运用的是 struts2 框架，它是很流行的 MVC 框架，它的功能就是将显示给浏览器的页面和网站后台的逻辑功能进行分离，它有两个核心的部分就是 Action 和 interceptor。它的工作原理就是浏览器发送请求→核心控制器根据请求找到对应的 action→action 执行相应的 method，将结果返回客户端。

接下来学习的是 Hibernate5 框架，个人感觉还是比较好理解的，它就是简单地将数据库的表格视图等映射成为 Java 的类对象，这些类对象的操作，都是通过 hibernate 映射到对数据库的操作，同时 hibernate 封装了对数据库的操作，同 jdbc 相比，它省去

了许多操作代码，虽然在逻辑上两者是相同的，但是 hibernate 更容易理解和操作，还解决了 java 同不同的数据库连接时 SQL 语句问题，使用 hibernate，必须的有 JavaBean 实体类，Hibernate 的配置文件，还有数据库表和 JavaBean 实体类的映射文件。

最后接触到的是 spring4 框架，spring4 主要就是进行逻辑层开发，从某个程度上来说，他将 struts2 和 hibernate5 框架进行了粘合，使得整个 Web 系统能够整合起来。它提供的容器功能，可以管理对象的生命周期及他们之间的依赖关系，在容器启动之后，对象就被实例化好了，依赖关系也建立好了，就直接可以进行使用。还有就是它的注入功能，在本项目中我用的是 set 方法注入，其中它的 bean 就是要注入的对象，我们只需要在 java 后台设置属性的 set 方法即可实现注入值。

项目中的 applicationContext.xml 配置文件就是指定不同的 bean 对象之间的控制和依赖关系的。

对于该项目的前端设计，我使用的是 HTML+CSS+JavaScript 结合的方式。在设计的过程中，我掌握了网页布局的设计方式以及 javascript+CSS 对网页效果优化方式。同时在移植到 jsp 页面时，又掌握了 EL 表达式获取和 struts2 标签显示 java 后台数据的方式。总之，本次 Web 项目的设计，我的能力有了明显的提升，并感受到这其中的趣味和意义。