

第14章

天下淘网络商城

(Struts 2+Spring+Hibernate+MySQL 实现)

喜欢网上购物的读者一定登录过淘宝，也一定被网页上琳琅满目的商品所吸引，忍不住拍一个自己喜爱的商品，如今也有越来越多的人加入到网购的行列，做网上店铺的老板，做新时代的购物潮人，你是否也想过开发一个自己的网上商城？在接下来的章节中我们将一起进入天下淘网络商城开发的旅程。通过本章学习，你将学到：

- » 了解网上商城的核心业务
- » 网站开发的基本流程
- » SSH2 的整合
- » MVC 的开发模式
- » 支持无限级别树生成的算法
- » 发布配置 Tomcat 服务器

14.1 开发背景

近年来,随着 Internet 的迅速崛起,互联网用户的爆炸式增长以及互联网对传统行业的冲击让其成为了人们快速获取、发布和传递信息的重要渠道,于是电子商务逐渐流行起来,越来越多的商家在网上建起网上商城,向消费者展示出一种全新的购物流念,同时也有越来越多的网友加入到了网上购物的行列,阿里巴巴旗下的淘宝的成功展现了电子商务网站强大的生命力和电子商务网站更加光明的未来。

笔者充分利用 Internet 这个平台,实现一种全新的购物方式——网上购物,其目的是方便广大网友购物,让网友足不出户就可以逛商城买商品,为此构建天下淘商城系统。

14.2 系统分析

14.2.1 需求分析

天下淘商城系统是基于 B/S 模式的电子商务网站,用于满足不同人群的购物需求,笔者通过对现有的商务网站的考察和研究,从经营者和消费者的角度出发,以高效管理、满足消费者需求为原则,要求本系统满足以下要求:

- ☒ 统一友好的操作界面,具有良好的用户体验;
- ☒ 商品分类详尽,可按不同类别查看商品信息;
- ☒ 推荐产品、人气商品以及热销产品的展示;
- ☒ 会员信息的注册及验证;
- ☒ 用户可通过关键字搜索指定的产品信息;
- ☒ 用户可通过购物车一次购买多件商品;
- ☒ 实现收银台的功能,用户选择商品后可以在线提交订单;
- ☒ 提供简单的安全模型,用户必须先登录,才允许购买商品;
- ☒ 用户可查看自己的订单信息;
- ☒ 设计网站后台,管理网站的各项基本数据;
- ☒ 系统运行安全稳定、响应及时。

14.2.2 可行性分析

在正式开发系统之前,首先需要对天下淘商城的技术、操作和经济成本三个方面进行可行性分析。

☒ 技术可行性

本系统采用 MVC 设计模式,使用当前最流行的 Struts 2+Spring2+Hibernate 框架进行开发,在前台用 JSP 进行页面开发和管理用户界面,利用轻巧的 JavaScript 库——jQuery 处理页面的 JavaScript 脚本,



使开发更加高效,提示信息更加完善,界面友好,具有较强的亲和力。后台采用 MySQL 数据库,MySQL 小巧高效的特点主以满足系统的性能要求。本系统使用当前主流的 Java 开源开发工具 Eclipse 和 Tomcat 服务器进行程序开发和发布,它们是完全免费的,可以节约开发成本。本系统采用的技术和开发环境在实际开发中应用非常广泛,充分说明本系统在技术方面可行。

☑ 操作可行性

天下淘商城主要面向的是喜欢网购的网友,只要天下淘商城的用户会一些简单的电脑操作,就可以进行网上购物,不需要用户具有较高的计算机专业知识,而且对于网站基本信息的维护也是十分的简单,管理员可以在任何一台可以上网的机器上对网站进行维护,网站的简单易用性充分说明了天下淘商城的操作可行性。

☑ 经济成本可行性

在实际的销售运营过程中,产品的宣传受到限制,采购商或顾客只能通过上门咨询、电话沟通等方式进行各种产品信息的获取,而且时间与物理的局限性严重影响了产品的销售,并且在无形中提高了产品的销售成本。本系统完全可以改变这种现状,以少量的时间和资金建立企业商务网络,以此来使企业与消费者之间的经济活动变得更加灵活、主动。

系统中应用的开发工具以及技术框架都是免费的,这无疑为网站的成本再一次压缩了空间,从成本可行性分析来看,该系统充分体现了将产品利益最大化的企业原则。

14.3 系统设计

14.3.1 功能结构图

天下淘商城系统分为前台和后台两个部分的操作。前台主要有两大功能,分别是展示产品信息的各种浏览操作和会员用户购买商品的操作,当会员成功登录后,就可以使用购物车进行网上购物。天下淘商城前台功能结构图如图 14.1 所示。

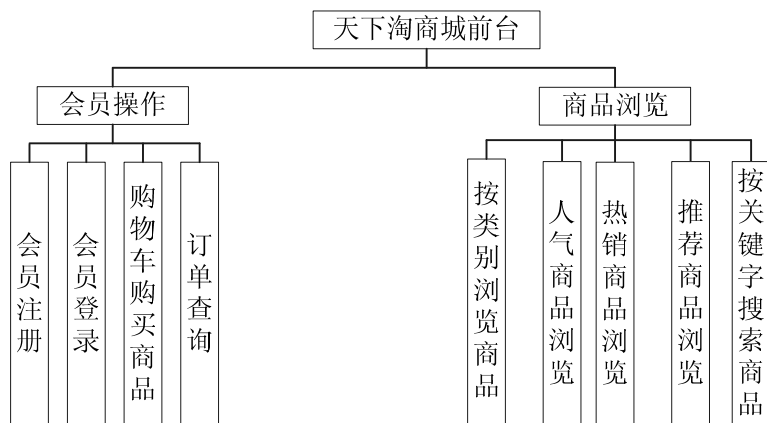


图 14.1 天下淘商城系统前台功能结构图

后台的主要功能是当管理员成功登录后台后，用户可以对网站的基本信息进行维护。例如，管理员可以对商品的类别进行管理，如可以删除和添加产品的类别；如可以对商品信息进行维护，如可以添加、删除、修改和查询产品信息，并上传产品的相关图片；如可以对会员的订单进行集中管理，管理员可以对订单信息进行自定义的条件查询并修改制定的产品信息。天下淘商城后台功能结构图如图 14.2 所示。

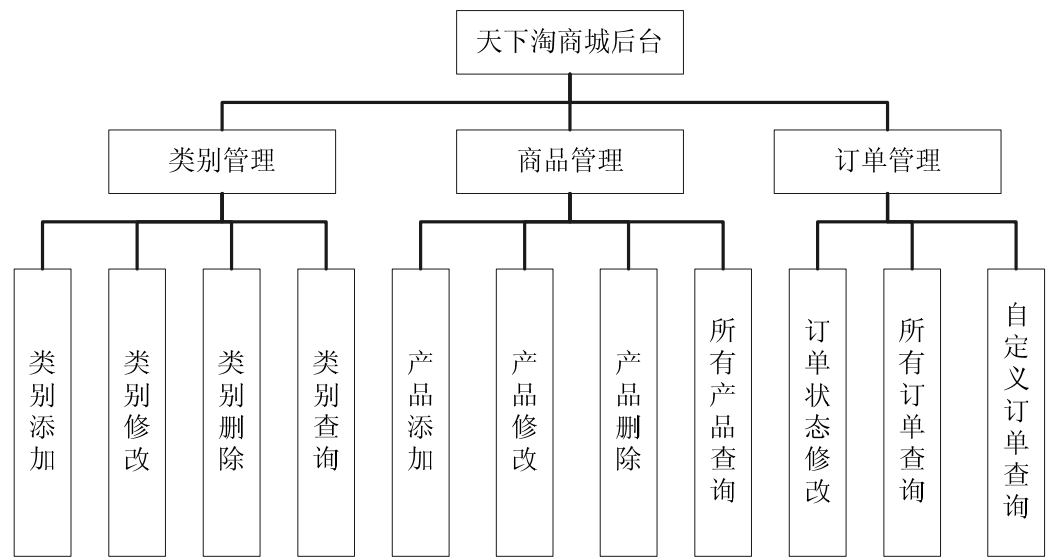


图 14.2 天下淘商城系统后台功能结构图

14.3.2 系统流程图

在天下淘商城中只有会员才允许进行购物操作，所以初次登陆网站的游客如果想进行购物操作必须注册为天下淘商城的会员。成功注册为会员后，会员可以使用购物车选择自己需要的商品，在确认订单付款后，系统将自动生成此次交易的订单基本信息。网站基本信息的维护由网站管理员负责，由管理员负责对商品信息、商品类别信息以及订单信息进行维护，关于订单的维护只能修改订单的状态，并不能修改订单的基本信息，因为订单确认之后就是用户与商家之间交易的凭证，第三方无权修改。

天下淘商城的系统流程图如图 14.3 所示。

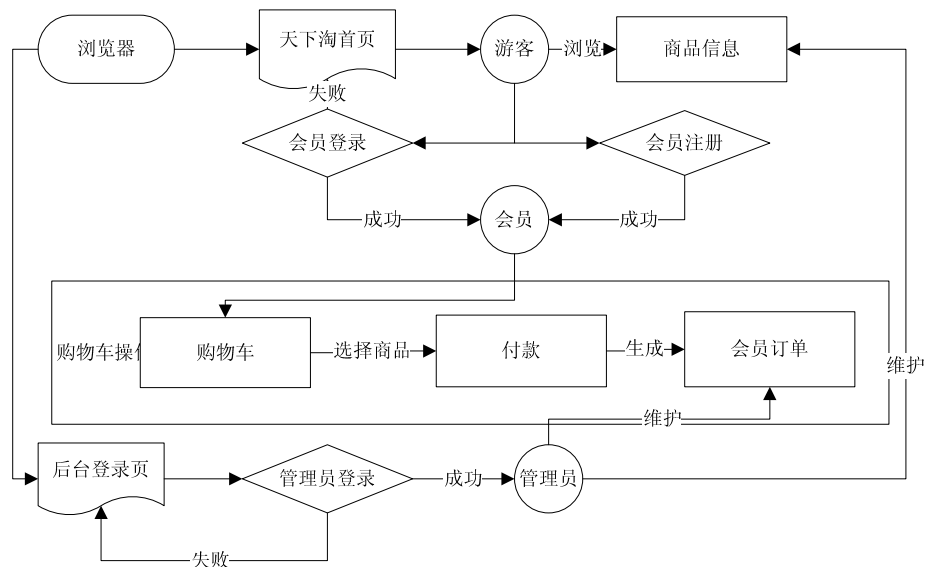


图 14.3 天下淘商城系统流程图

14.3.3 开发环境

在进行天下淘商城网站开发时，需要具备以下开发环境。

服务器端：

- ☑ 操作系统：Windows2003 或者更高版本的服务器操作系统。
- ☑ Web 服务器：Tomcat6.0 或 6.0 以上版本。
- ☑ Java 开发包：JDK1.5 以上。
- ☑ 数据库：MySQL14.1。
- ☑ 浏览器：IE6.0 或者更高版本的浏览器。
- ☑ 分辨率：最低要求为 800*600 像素。

客户端：

- ☑ 浏览器：IE6.0 或者更高版本的浏览器。
- ☑ 分辨率：最低要求为 800*600 像素。

14.3.4 文件夹组织结构

在编写代码之前，可以把系统中可能用到的文件夹先创建出来（例如，创建一个名为 images 的文件夹，用于保存网站中所使用的图片），这样不但可以方便以后的开发工作，也可以规范网站的整体架构。本系统的文件夹组织结构如图 14.4 所示。

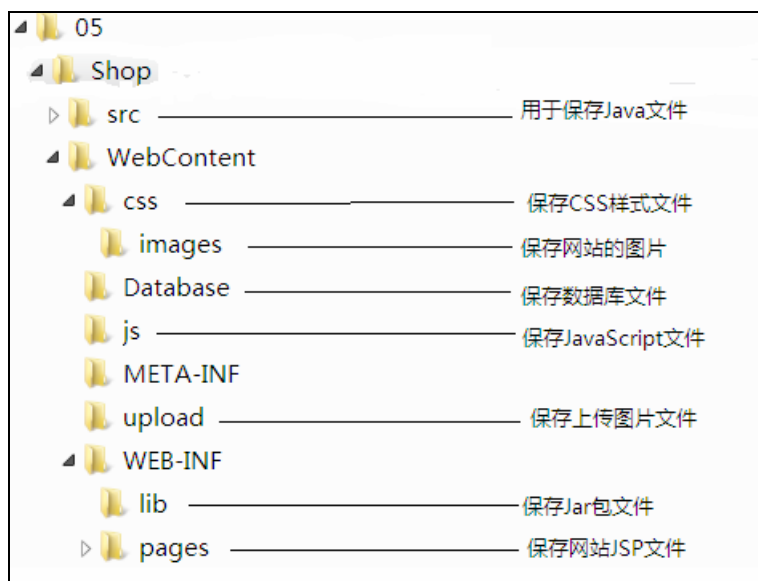


图 14.4 天下淘购物网文件夹组织结构

14.3.5 系统预览

系统预览将以用户交易为例，列出几个关键的页面，商品交易是天下淘商城的核心模块之一，通过该预览的展示，读者可以对天下淘商城有个基本的了解，同时读者也可以在光盘中对本程序的源程序进行查看。

当用户在地址栏中输入天下淘商城的域名，就可以进入到天下淘商城，首页将商品的类别信息分类展现给用户，并在首页展示部分的人气商品、推荐商品、热销商品以及上市新品，首页部分效果如图 14.5 所示。



图 14.5 天下淘商城首页部分页面效果图

如果用户为会员登录后就可以直接进行产品的选购，当用户在商品信息详细页面中单击“直接购买”超链接，就会将该商品放入购物车中，同时用户也可以使用购物车选购多种商品，购物车同时可以保存多件会员采购的商品信息，图 14.6 为用户选购多件产品的效果。



图 14.6 天下淘商城购物车页面效果图

当用户到收银台付款后，系统将自动生成订单，会员可通过单击左侧导航栏中的“我的订单”超

链接查看自己的订单信息，如图 14.7 所示。

我的购物车

我的订单

欢迎 mrsoft

退出

天下淘

搜索

高级搜索

使用帮助

热搜商品：软件

首页

新品上市

热销商品

推荐商品

人气商品

我的帐户

我的购物车

我的订单

> 我的订单

订单号码	订单总金额	收货人	收货地址	支付方式	创建时间	订单状态
201005041012220323561	120.0	mrsoft	吉林省长春市二道区 xxx号xxx小区xxx门	邮局汇款	2010年05月4日 10:12	已发货
201004271843190764180	240.0	mrsoft	吉林省长春市二道区 xxx号xxx小区xxx门	邮局汇款	2010年04月27日 18:43	交易完成
201004260941250469034	120.0	mrsoft	吉林省长春市二道区 xxx号xxx小区xxx门	邮局汇款	2010年04月26日 09:41	已取消

图 14.7 天下淘商城会员订单信息效果图

14.4 数据库设计

整个应用系统的运行离不开数据库的支持，数据库可以说是应用系统的灵魂，没有了数据库的支撑，系统只能说是一个空架子，它将很难完成与用户之间的交互。由此可见，数据库在系统中占有十分重要的地位。本系统采用的是 MySQL 数据库，通过 Hibernate 实现系统的持久化操作。

本节将根据天下淘商城网站的核心实体类，分别设计对应的 E-R 图和数据表。

14.4.1 数据库概念设计

所谓的数据库概念化设计，就是将现实世界中的对象以 E-R 图的形式展现出来，本节将对程序所应用到的核心实体对象设计对应的 E-R 图。

☑ 会员信息表 tb_customer 的 E-R 图，如图 14.8 所示。

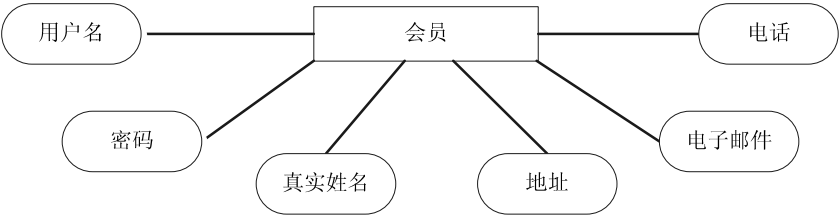


图 14.8 会员信息表 tb_customer 的 E-R 图

☑ 订单信息表 tb_order 的 E-R 图，如图 14.9 所示。

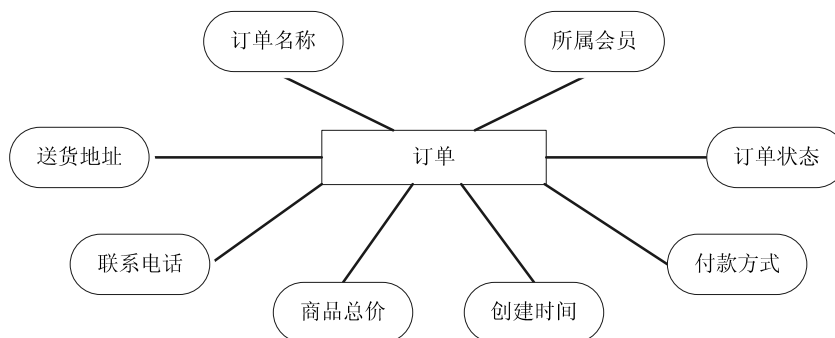


图 14.9 订单信息表 tb_order 的 E-R 图

- ☑ 订单条目信息表 tb_orderitem 的 E-R 图，如图 14.10 所示。

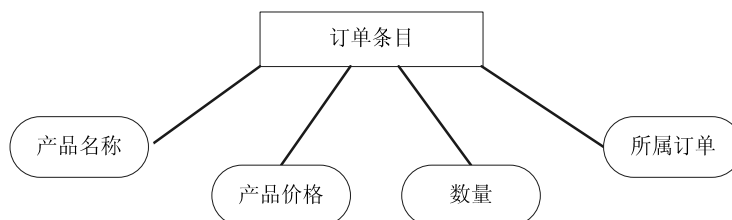


图 14.10 订单条目信息表 tb_orderitem 的 E-R 图

- ☑ 商品信息表 tb_productinfo 的 E-R 图，如图 14.11 所示。

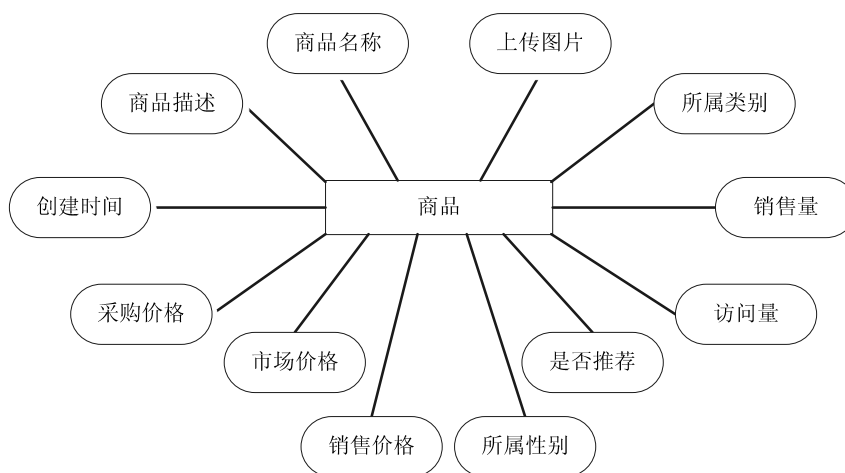


图 14.11 商品信息表 tb_productinfo 的 E-R 图

- ☑ 商品类别信息表 tb_productcategory 的 E-R 图，如图 14.12 所示。

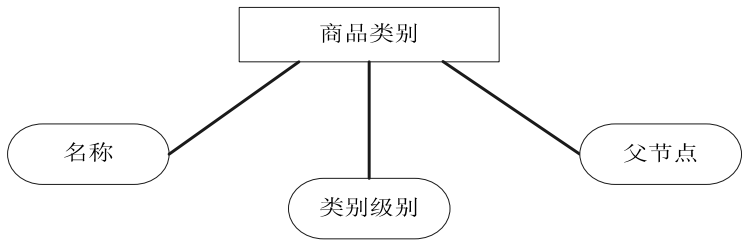


图 14.12 商品类别信息表 tb_productcategory 的 E-R 图

14.4.2 创建数据库及数据表

本系统采用 MySQL 数据库，创建的数据库名称为 db_database19，数据库 db_database19 中包含 7 张数据表。所有数据表的定义如下：

- ☒ tb_customer（会员信息表）
用于存储会员的注册信息，该表的结构如表 14.1 所示。

表 14.1 tb_customer 信息表的表结构

字段名	数据类型	是否为空	是否主键	默认值	说明
id	INT(10)	否	是	NULL	系统自动编号
username	VARCHAR(50)	否	否	NULL	会员名称
password	VARCHAR(50)	否	否	NULL	登录密码
realname	VARCHAR(20)	是	否	NULL	真实姓名
address	VARCHAR(200)	是	否	NULL	地址
email	VARCHAR(50)	是	否	NULL	电子邮件
mobile	VARCHAR(11)	是	否	NULL	电话号码

- ☒ tb_order（订单信息表）
用于存储会员的订单信息，该表的结构如表 14.2 所示。

表 14.2 tb_order 信息表的表结构

字段名	数据类型	是否为空	是否主键	默认值	说明
id	INT(10)	否	是	NULL	系统自动编号
name	VARCHAR(50)	否	否	NULL	订单名称
address	VARCHAR(200)	否	否	NULL	送货地址
mobile	VARCHAR(11)	否	否	NULL	电话
totalPrice	FLOAT	是	否	NULL	采购价格
createTime	DATETIME	是	否	NULL	创建时间
paymentWay	VARCHAR(15)	是	否	NULL	支付方式
orderState	VARCHAR(10)	是	否	NULL	订单状态
customerId	INT(11)	是	否	NULL	会员 ID

- ☒ tb_orderitem（订单条目信息表）

用于存储会员订单的条目信息，该表的结构如表 14.3 所示。

表 14.3 tb_orderitem 信息表的表结构

字段名	数据类型	是否为空	是否主键	默认值	说明
id	INT(10)	否	是	NULL	系统自动编号
productId	INT(11)	否	否	NULL	商品 ID
productName	VARCHAR(200)	否	否	NULL	商品名称
productPrice	FLOAT	否	否	NULL	商品价格
amount	INT(11)	是	否	NULL	商品数量
orderId	VARCHAR(30)	是	否	NULL	订单 ID

☒ tb_productinfo (商品信息表)

用于存储商品信息，该表的结构如表 14.4 所示。

表 14.4 tb_productinfo 信息表的表结构

字段名	数据类型	是否为空	是否主键	默认值	说明
id	INT(10)	否	是	NULL	系统自动编号
name	VARCHAR(100)	否	否	NULL	商品名称
description	TEXT	是	否	NULL	商品描述
createTime	DATETIME	是	否	NULL	创建时间
baseprice	FLOAT	是	否	NULL	采购价格
marketprice	FLOAT	是	否	NULL	市场价格
sellprice	FLOAT	是	否	NULL	销售价格
sexrequest	VARCHAR(5)	是	否	NULL	所属性别
commend	BIT(1)	是	否	NULL	是否推荐
clickcount	INT(11)	是	否	NULL	浏览量
sellCount	INT(11)	是	否	NULL	销售量
categoryId	INT(11)	是	否	NULL	商品类别 ID
uploadFile	INT(11)	是	否	NULL	上传文件 ID

☒ tb_productcategory (商品类别信息表)

用于存储商品的类别信息，该表的结构如表 14.5 所示。

表 14.5 tb_productcategory 信息表的表结构

字段名	数据类型	是否为空	是否主键	默认值	说明
id	INT(10)	否	是	NULL	系统自动编号
name	VARCHAR(50)	否	否	NULL	类别名称
level	INT(11)	是	否	NULL	类别级别
pid	INT(11)	是	否	NULL	父节点类别 ID

☒ tb_user (管理员信息表)

用于存储网站后台管理员信息，该表的结构如表 14.6 所示。



表 14.6 tb_productcategory 信息表的表结构

字段名	数据类型	是否为空	是否主键	默认值	说明
id	INT(10)	否	是	NULL	系统自动编号
username	VARCHAR(50)	否	否	NULL	用户名
password	VARCHAR(50)	否	否	NULL	登录密码

☒ tb_uploadfile (上传文件信息表)

用于存储上传文件的路径信息，该表的结构如表 14.7 所示。

表 14.7 tb_uploadfile 信息表的表结构

字段名	数据类型	是否为空	是否主键	默认值	说明
id	INT(10)	否	是	NULL	系统自动编号
path	VARCHAR(255)	否	是	NULL	文件路径信息

14.5 公共模块的设计

在项目中经常会有一些公共类，例如 Hibernate 的初始化类，一些自定义的字符串处理方法，抽取系统中公共模块更加有利于代码重用，同时也能提高程序的开发效率，在进行正式开发时首先要进行的就是公共模块的编写。下面介绍天下淘商城的公共类。

14.5.1 泛型工具类

Hibernate 提供了高效的对象到关系型数据库的持久化服务，通过面向对象的思想进行数据持久化的操作，Hibernate 的操作对象就是数据表所对应的实体对象，为了将一些公用的持久化方法提取出来，首先需要实现获取实体对象的类型方法，在本应用中通过自定义创建一个泛型工具类 GenericsUtils 来达到此目的，其代码如下：

代码位置：光盘\mr\14\Shop\src\com\lyq\util\HibernateUtils

```
public class GenericsUtils {
    @SuppressWarnings("unchecked")
    public static Class getGenericType(Class clazz){
        Type genType = clazz.getGenericSuperclass();    //得到泛型父类
        Type[] types = ((ParameterizedType) genType).getActualTypeArguments();
        if (!(types[0] instanceof Class)) {
            return Object.class;
        }
        return (Class) types[0];
    }
    // 获取对象的类名称
    @SuppressWarnings("unchecked")
    public static String getGenericName(Class clazz){
```

```

        return clazz.getSimpleName();
    }
}

```

14.5.2 数据持久化类

在本应用中利用 DAO 模式实现数据库基本操作方法的封装,数据库中最基本的操作就是增、删、改、查,据此自定义数据库操作的公共方法。由控制器负责获取请求参数并控制转发,由 DAOSupport 类组织 SQL 语句。

根据自定义的数据库操作的公共方法,创建接口 BaseDao<T>,关键代码如下:

代码位置: 光盘\mr\14\Shop\src\com\lyq\dao\BaseDao

```

public interface BaseDao<T> {
    //基本数据库操作方法
    public void save(Object obj); //保存数据
    public void saveOrUpdate(Object obj); //保存或修改数据
    public void update(Object obj); //修改数据
    public void delete(Serializable ... ids); //删除数据
    public T get(Serializable entityId); //加载实体对象
    public T load(Serializable entityId); //加载实体对象
    public Object uniqueResult(String hql, Object[] queryParams); //使用hql语句操作
}

```

创建类 DaoSupport,该类继承 BaseDao<T>接口,在类中实现接口中自定义的方法,其关键代码如下:

代码位置: 光盘\mr\14\Shop\src\com\lyq\dao\DaoSupport

```

public abstract class DaoSupport<T> implements BaseDao<T>{
    protected Class<T> entityClass = GenericsUtils.getGenericType(this.getClass()); //泛型的类型
    @Autowired
    protected HibernateTemplate template; // Hibernate模板
    public HibernateTemplate getTemplate() {
        return template;
    }
    // 删除指定的对象信息
    public void delete(Serializable ... ids) {
        for (Serializable id : ids) { //遍历标识参数
            T t = (T) getTemplate().load(this.entityClass, id); //加载指定对象
            getTemplate().delete(t); //执行删除操作
        }
    }
    //利用get()方法加载对象,获取对象的详细信息
    @Transactional(propagation=Propagation.NOT_SUPPORTED,readonly=true)
    public T get(Serializable entityId) {
        return (T) getTemplate().get(this.entityClass, entityId); //加载指定对象
    }
    //利用load()方法加载对象,获取对象的详细信息
    @Transactional(propagation=Propagation.NOT_SUPPORTED,readonly=true)

```



```

public T load(Serializable entityId) {
    return (T) getTemplate().load(this.entityClass, entityId);    //加载指定对象
}
//利用save()方法保存对象的详细信息
public void save(Object obj) {
    getTemplate().save(obj); //
}
// 保存或修改信息
public void saveOrUpdate(Object obj) {
    getTemplate().saveOrUpdate(obj);
}
//利用update()方法修改对象的详细信息
public void update(Object obj) {
    getTemplate().update(obj);
}
}

```

14.5.3 分页操作

分页查询是 Java Web 开发中十分常用的技术。在数据库量非常大的情况下，不适合将所有数据显示到一个页面之中，这样既给查看带来不便，又占用程序及数据库的资源，此时就需要对数据进行分页查询。本系统应用 Hibernate 的 find 方法实现数据分页的操作，将分页的方法封装在创建类 DaoSupport 中，下面将介绍 Hibernate 分页实现的方法。

☑ 分页实体对象

首先定义分页的实体对象，封装分页基本属性信息和在分页过程中使用的获取页码的方法。

代码位置：光盘\mr\14\Shop\src\com\lyq\model\PageModel<T>

```

public class PageModel<T> {
    private int totalRecords;    //总记录数
    private List<T> list;        //结果集
    private int pageNo;          //当前页
    private int pageSize;        //每页显示多少条
    // 取得第一页
    public int getTopPageNo() {
        return 1;
    }
    //取得上一页
    public int getPreviousPageNo() {
        if (pageNo <= 1) {
            return 1;
        }
        return pageNo - 1;
    }
    // 取得下一页
    public int getNextPageNo() {
        if (pageNo >= getTotalPages()) {
            //如果当前页大于页码
            return getTotalPages() == 0 ? 1 : getTotalPages();    //返回最后一页
        }
    }
}

```

```

    }
    return pageNo + 1;
    }
    // 取得最后一页
    public int getBottomPageNo() {
        return getTotalPages() == 0 ? 1 : getTotalPages();    //如果总页数为0返回1,反之返回总页数
    }
    // 取得总页数
    public int getTotalPages() {
        return (totalRecords + pageSize - 1) / pageSize;
    }
    .....
    //省略的Setter和Getter方法
}

```

在页面的实体对象中，封装了几个重要的页码获取方法，即获取第一页、上一页、下一页、最后一页以及总页数的方法。

在取得上一页页码的方法 `getPreviousPageNo()` 中，如果当前页的页码数为首页，那么上一页返回的页码数为 1。

在获取最后一页的方法 `getBottomPageNo()` 中，通过三目运算符进行选择判断返回的页码，如果总页数为 0 则返回 1，反之返回总页面数。当数据库中没有任何信息的时候，总页数为 0。

☑ 实现自定义分页方法

在公共接口中定义几种不同的分页方法，这些方法定义使用了相同的分页方法，不同的参数，自定义分页方法关键代码如下：

代码位置：光盘\mr\14\Shop\src\com\lyq\dao\BaseDao

```

public interface BaseDao<T> {
    .....
    //分页操作
    public long getCount();
    public PageModel<T> find(int pageNo, int maxResult);
    //搜索信息分页方法
    public PageModel<T> find(int pageNo, int maxResult, String where, Object[] queryParams);
    //按指定条件排序分页方法
    public PageModel<T> find(int pageNo, int maxResult, Map<String, String> orderby);
    //按指定条件分页和排序的分页方法
    public PageModel<T> find(String where, Object[] queryParams,
        Map<String, String> orderby, int pageNo, int maxResult);
}

```

14.5.4 字符串工具类

`StringUtil` 类中主要实现了字符串与其他数据类型的转换，例如将日期时间型数据转换为指定格式的字符串，处理订单号码的生成以及验证字符串和浮点数的有效性。该类中声明的所有方法都是静态方法，以便在其他类中可以通过 `StringUtil` 类名直接调用。

☑ 日期格式转换方法

在方法中通过 `new Date()` 方法获取当前的系统时间，通过 `SimpleDateFormat` 的 `format()` 方法将日期



格式为指定的日期格式。该方法主要是在操作数据库的时候作为一个有效字段使用，例如订单的创建日期等等，其代码如下：

代码位置：光盘\mr\14\Shop\src\com\lyq\util\StringUtil

```
public static String getStringTime(){
    Date date = new Date();
    SimpleDateFormat sdf = new SimpleDateFormat("yyyyMMddHHmmssSSSS");
    return sdf.format(date);
}
```

//获取当前系统时间
//设置格式化格式
//返回格式化后的时间

14.5.5 实体映射

由于本程序中使用了 Hibernate 框架，所以需要创建实体对象并通过 Hibernate 的映射文件将实体对象与数据库中相应的数据表进行关联。在天下淘商城中有 5 个主要的实体对象，分别是会员实体对象、订单实体对象、订单条目实体对象、商品实体对象以及商品类别实体对象。

☑ 实体对象总体设计

实体对象是 Hibernate 中非常重要的一个环节，因为 Hibernate 只有通过映射文件建立实体对象与数据库数据表之间的关系，才能进行系统的持久化操作。在天下淘商城网站中主要实体对象及其关系如图 14.13 所示。

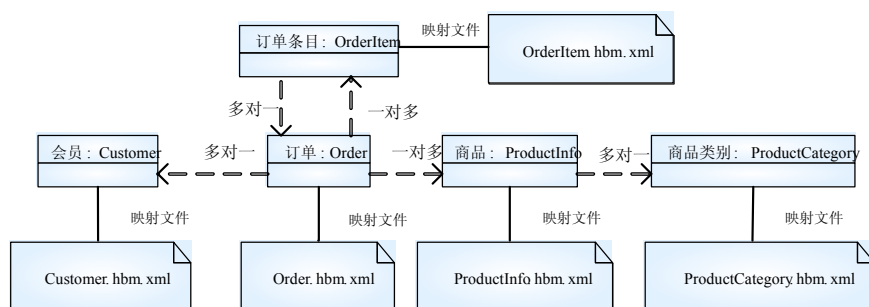


图 14.13 天下淘商城主要实体对象及其关系

从图 14.13 中可以看到，该项目主要有五个实体对象，分别是会员实体对象 Customer 类、订单实体对象 Order 类、订单条目实体对象 OrderItem 类、商品实体对象 ProductInfo 类和商品类别实体对象 ProductCategory 类。

从中可以看到会员与订单是一对多的关系，一个会员可以对应多张订单，但是每张订单只能对应一个会员；订单条目与订单为多对一的关系，一张订单中可以包含多个订单条目，但是每个订单条目只能对应一张订单；订单与产品是一对多关系，一张订单可以对应多个商品；商品与商品类别是多对一关系，多件商品可以对应一个商品类别。

其中的“*.hbm.xml”文件为实体对象的 Hibernate 映射文件。

☑ 会员信息

Customer 类为会员信息实体类，用于封装会员的注册信息，其关键代码如下：

代码位置：光盘\mr\14\Shop\src\com\lyq\user\Customer


```

public class Customer implements Serializable{
    private Integer id;           // 用户编号
    private String username;      // 用户名
    private String password;      // 密码
    private String realname;      // 真实姓名
    private String email;         // 邮箱
    private String address;       // 住址
    private String mobile;        // 手机
    .....                        //省略的Setter和Getter方法
}

```

创建会员信息实体类的映射文件 Customer.hbm.xml，在映射文件中配置会员实体类属性与数据表 tb_customer 响应字段的关联，并声明用户编号的主键生成策略为自动增长，配置文件中的关键代码如下：

代码位置：光盘\mr\14\Shop\src\com\lyq\user\Customer.hbm.xml

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC
    "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
    "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd" >
<hibernate-mapping package="com.lyq.model.user">
    <class name="Customer" table="tb_customer">
        <id name="id">
            <generator class="native"/>
        </id>
        <property name="username" not-null="true" length="50"/>
        <property name="password" not-null="true" length="50"/>
        <property name="realname" length="20"/>
        <property name="address" length="200"/>
        <property name="email" length="50"/>
        <property name="mobile" length="11"/>
    </class>
</hibernate-mapping>

```

☒ 订单信息

Order 类为订单信息实体类，用户封装订单的基本信息，但是不包括详细的订购信息，其关键代码如下：

代码位置：光盘\mr\14\Shop\src\com\lyq\model\order\Order

```

public class Order implements Serializable {
    private String orderId;           // 订单编号(手动分配)
    private Customer customer;        // 所属用户
    private String name;              // 收货人姓名
    private String address;           // 收货人住址
    private String mobile;            // 收货人手机
    private Set<OrderItem> orderItems; // 所买商品
    private Float totalPrice;         // 总额
    private PaymentWay paymentWay;    // 支付方式
    private OrderState orderState;    // 订单状态
    private Date createTime = new Date(); // 创建时间
}

```



```
..... // 省略的Setter和Getter方法
}
```

创建订单信息实体类的映射文件 `Order.hbm.xml`，在映射文件中配置订单实体类属性与数据表 `tb_order` 字段的关联，声明主键 `orderId` 的主键生成策略为手动分配，并配置订单与会员的多对一关系，订单与订单项的一对多关系，其关键代码如下：

代码位置：光盘\mr\14\Shop\src\com\lyq\model\order\Order

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC
    "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
    "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd" >
<hibernate-mapping package="com.lyq.model.order">
    <class name="Order" table="tb_order">
        <id name="orderId" type="string" length="30">
            <generator class="assigned"/>
        </id>
        <property name="name" not-null="true" length="50"/>
        <property name="address" not-null="true" length="200"/>
        <property name="mobile" not-null="true" length="11"/>
        <property name="totalPrice"/>
        <property name="createTime" />
        <property name="paymentWay" type="com.lyq.util.hibernate.PaymentWayType" length="15"/>
        <property name="orderState" type="com.lyq.util.hibernate.OrderStateType" length="10"/>
        <!-- 多对一映射用户 -->
        <many-to-one name="customer" column="customerId"/>
        <!-- 映射订单项 -->
        <set name="orderItems" inverse="true" lazy="extra" cascade="all">
            <key column="orderId"/>
            <one-to-many class="OrderItem"/>
        </set>
    </class>
</hibernate-mapping>
```

☒ 订单条目信息

`OrderItem` 类为订单条目的实体对象，用于封装一个订单中的一条详细商品采购信息，其关键代码如下：

代码位置：光盘\mr\14\Shop\src\com\lyq\model\order\OrderItem

```
public class OrderItem implements Serializable{
    private Integer id;                // 商品条目编号
    private Integer productId;         // 商品id
    private String productName;        // 商品名称
    private Float productMarketprice; // 市场价格
    private Float productPrice;        // 商品销售价格
    private Integer amount=1;          // 购买数量
    private Order order;               // 所属订单
    .....                             // 省略的Setter和Getter方法
}
```

创建订单条目信息实体类的映射文件 `OrderItem.hbm.xml`，在映射文件中配置订单条目实体类属性

与数据表 `tb_orderitem` 字段的关联，声明主键 `id` 的主键生成策略为自动增长，并配置订单条目与订单的多对一关系，其关键代码如下：

代码位置：光盘\mr\14\Shop\src\com\lyq\model\order\OrderItem.hbm.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC
    "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
    "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd" >
<hibernate-mapping package="com.lyq.model.order">
    <class name="OrderItem" table="tb_orderItem">
        <id name="id">
            <generator class="native"/>
        </id>
        <property name="productId" not-null="true"/>
        <property name="productName" not-null="true" length="200"/>
        <property name="productPrice" not-null="true"/>
        <property name="amount"/>
        <!-- 多对一映射订单 -->
        <many-to-one name="order" column="orderId"/>
    </class>
</hibernate-mapping>
```

☒ 商品信息

`ProductInfo` 类为商品信息实体类，主要用户封装商品相关的基本信息，它是整个系统中最为重要的一个实体对象，也是应用最多的一个实体对象，整个网站的业务流程都以商品为核心进行展开，其关键代码如下：

代码位置：光盘\mr\14\Shop\src\com\lyq\model\product\ProductInfo

```
public class ProductInfo implements Serializable {
    private Integer id;                // 商品编号
    private String name;               // 商品名称
    private String description;        // 商品说明
    private Date createTime = new Date(); // 上架时间
    private Float baseprice;           // 商品采购价格
    private Float marketprice;        // 现在市场价格
    private Float sellprice;           // 商城销售价格
    private Sex sexrequest;            // 所属性别
    private Boolean commend = false;    // 是否是推荐商品（默认值为false）
    private Integer clickcount = 1;     // 访问量（统计受欢迎的程度）
    private Integer sellCount = 0;      // 销售数量（统计热销商品）
    private ProductCategory category;  // 所属类别
    private UploadFile uploadFile;     // 上传文件
    .....                             // 省略的Setter和Getter方法
}
```

创建商品信息实体类的映射文件 `ProductInfo.hbm.xml`，在映射文件中配置商品实体类属性与数据表 `tb_productinfo` 字段的关联，并声明其主键 `id` 的生成策略为自动增长，并配置商品与商品类别多对一关联关系、商品与商品上传文件的多对一关联关系，其关键代码如下：

代码位置：光盘\mr\14\Shop\src\com\lyq\model\product\ProductInfo.hbm.xml



```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC
    "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
    "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd" >
<hibernate-mapping package="com.lyq.model.product">
    <class name="ProductInfo" table="tb_productInfo">
        <id name="id">
            <generator class="native"/>
        </id>
        <property name="name" not-null="true" length="100"/>
        <property name="description" type="text"/>
        <property name="createTime"/>
        <property name="baseprice"/>
        <property name="marketprice"/>
        <property name="sellprice"/>
        <property name="sexrequest" type="com.lyq.util.hibernate.SexType" length="5"/>
        <property name="commend"/>
        <property name="clickcount"/>
        <property name="sellCount"/>
        <!-- 多对一映射类别 -->
        <many-to-one name="category" column="categoryId"/>
        <!-- 多对一映射上传文件 -->
        <many-to-one name="uploadFile" unique="true" cascade="all" lazy="false"/>
    </class>
</hibernate-mapping>

```

☒ 商品类别信息

ProductCategory 类为商品类别的实体对象，主要用户封装商品类别的基本信息，其关键代码如下：

代码位置：光盘\mr\14\Shop\src\com\lyq\model\product\ProductCategory

```

public class ProductCategory implements Serializable {
    private Integer id;                // 类别编号
    private String name;               // 类别名称
    private int level = 1;             // 层次
    private Set<ProductCategory> children; // 子产品类别
    private ProductCategory parent;     // 父类别
    private Set<ProductInfo> products = new TreeSet<ProductInfo>(); // 包含商品
    .....                             // 省略的Setter和Getter方法
}

```

创建商品类别信息实体类的映射文件 ProductCategory.hbm.xml，在映射文件中配置商品类别实体类属性与数据表 tb_productcategory 字段的关联，并配置商品类别与商品的一对多的关联关系，商品类别与其父节点多对一的关联关系，商品类别与其子节点一对多的关联关系，其关键代码如下：

代码位置：光盘\mr\14\Shop\src\com\lyq\model\product\ProductCategory.hbm.xml

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC
    "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
    "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd" >
<hibernate-mapping package="com.lyq.model.product">

```

```

<class name="ProductCategory" table="tb_productCategory">
  <id name="id">
    <generator class="native"/>
  </id>
  <property name="name" not-null="true" length="50"/>
  <property name="level"/>
  <!-- 映射包含的商品集合 -->
  <set name="products" inverse="true" lazy="extra">
    <key column="categoryId"/>
    <one-to-many class="ProductInfo" />
  </set>
  <!-- 映射父节点 -->
  <many-to-one name="parent" column="pid"/>
  <!-- 映射子节点 -->
  <set name="children" inverse="true" lazy="extra" cascade="all" order-by="id">
    <key column="pid"/>
    <one-to-many class="ProductCategory"/>
  </set>
</class>
</hibernate-mapping>

```

14.6 项目环境搭建

在项目正式开发的第一步就是搭建项目的环境以及项目集成的框架等，都说万丈高楼平地起，从此开始将踏上万里征程的第一步，在此之前需要将 Spring、Struts 2、Hibernate 以及系统应用的其他 jar 包导入到项目的 lib 文件下。

14.6.1 配置 Struts 2

struts.xml 文件是 Struts 2 重要的配置文件，通过对该文件的配置实现程序的 Action 与用户请求之间的映射、视图映射等重要的配置信息。在项目的 ClassPath 下创建 struts.xml 文件，其配置代码如下：

代码位置：光盘\mr\14\Shop\src\struts.xml

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE struts PUBLIC
  "-//Apache Software Foundation//DTD Struts Configuration 2.1//EN"
  "http://struts.apache.org/dtds/struts-2.1.dtd" >
<struts>
  <!-- 前后台公共视图的映射 -->
  <include file="com/lyq/action/struts-default.xml" />
  <!-- 后台管理的Struts 2配置文件 -->
  <include file="com/lyq/action/struts-admin.xml" />
  <!-- 前台台管理的Struts 2配置文件 -->
  <include file="com/lyq/action/struts-front.xml" />
</struts>

```



为了便于程序的维护和管理，将前后台的 Struts 2 配置文件进行分开处理，然后通过 include 标签加载在系统默认加载的 Struts 2 配置文件中。在此将 Struts 2 配置文件分为三个部分，struts-default.xml 文件为前后台公共的视图映射配置文件，其代码如下：

代码位置：光盘\mr\ 14\Shop\src\com\lyq\action\struts-default.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE struts PUBLIC
    "-//Apache Software Foundation//DTD Struts Configuration 2.1//EN"
    "http://struts.apache.org/dtds/struts-2.1.dtd">
<struts>
    <!-- OGNL可以使用静态方法 -->
    <constant name="struts.ognl.allowStaticMethodAccess" value="true"/>
    <package name="shop-default" abstract="true" extends="struts-default">
    <global-results>
        .....<!--省略的配置信息 -->
    </global-results>
    <global-exception-mappings>
    <exception-mapping result="error"
        exception="com.lyq.util.AppException"></exception-mapping>
    </global-exception-mappings>
    </package>
</struts>
```

后台管理的 Struts 2 配置文件 struts-admin.xml 主要负责后台用户请求的 Action 和视图映射，其代码如下：

代码位置：光盘\mr\ 14\Shop\src\com\lyq\action\struts-admin.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE struts PUBLIC
    "-//Apache Software Foundation//DTD Struts Configuration 2.1//EN"
    "http://struts.apache.org/dtds/struts-2.1.dtd" >
<struts>
    <!-- 后台管理 -->
    <package name="shop.admin" namespace="/admin" extends="shop-default">
    <!-- 配置拦截器 -->
    <interceptors>
        <!-- 验证用户登录的拦截器 -->
        <interceptor name="loginInterceptor"
            class="com.lyq.action.interceptor.UserLoginInterceptor"/>
        <interceptor-stack name="adminDefaultStack">
            <interceptor-ref name="loginInterceptor"/>
            <interceptor-ref name="defaultStack"/>
        </interceptor-stack>
    </interceptors>
    <action name="admin_*" class="indexAction" method="{1}">
        <result name="top">/WEB-INF/pages/admin/top.jsp</result>
        .....<!--省略的Action配置 -->
        <interceptor-ref name="adminDefaultStack"/>
    </action>
```

```

</package>
<package name="shop.admin.user" namespace="/admin/user" extends="shop-default">
    <action name="user_*" method="{1}" class="userAction"></action>
</package>
<!-- 栏目管理 -->
<package name="shop.admin.category" namespace="/admin/product" extends="shop.admin">
    <action name="category_*" method="{1}" class="productCategoryAction">
        .....<!--省略的Action配置 -->
        <interceptor-ref name="adminDefaultStack"/>
    </action>
</package>
<!-- 商品管理 -->
<package name="shop.admin.product" namespace="/admin/product" extends="shop.admin">
    <action name="product_*" method="{1}" class="productAction">
        .....<!--省略的Action配置 -->
        <interceptor-ref name="adminDefaultStack"/>
    </action>
</package>
<!-- 订单管理 -->
<package name="shop.admin.order" namespace="/admin/product" extends="shop.admin">
    <action name="order_*" method="{1}" class="orderAction">
        .....<!--省略的Action配置 -->
        <interceptor-ref name="adminDefaultStack"/>
    </action>
</package>
</struts>

```

前台管理的 Struts 2 配置文件 struts-front.xml 主要负责后台用户请求的 Action 和视图映射，其代码如下：

代码位置：光盘\mr\ 14\Shop\src\com\lyq\action\struts-front.xml

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE struts PUBLIC
    "-//Apache Software Foundation//DTD Struts Configuration 2.1//EN"
    "http://struts.apache.org/dtds/struts-2.1.dtd" >
<struts>
    <!-- 程序前台 -->
    <package name="shop.front" extends="shop-default">
        <!-- 配置拦截器 -->
        <interceptors>
            <!-- 验证用户登录的拦截器 -->
            <interceptor name="loginInterceptor"
                class="com.lyq.action.interceptor.CustomerLoginInteceptor"/>
            <interceptor-stack name="customerDefaultStack">
                <interceptor-ref name="loginInterceptor"/>
                <interceptor-ref name="defaultStack"/>
            </interceptor-stack>
        </interceptors>
        <action name="index" class="indexAction">
            <result>/WEB-INF/pages/index.jsp</result>

```



```

        </action>
    </package>
    <!-- 消费者 Action -->
    <package name="shop.customer" extends="shop-default" namespace="/customer">
        <action name="customer_*" method="{1}" class="customerAction"></action>
    </package>
    <!-- 商品 Action -->
    <package name="shop.product" extends="shop-default" namespace="/product">
        <action name="product_*" class="productAction" method="{1}">
            .....<!--省略的Action配置 -->
        </action>
    </package>
    <!-- 购物车 Action -->
    <package name="shop.cart" extends="shop.front" namespace="/product">
        <action name="cart_*" class="cartAction" method="{1}">
            .....<!--省略的Action配置 -->
            <interceptor-ref name="customerDefaultStack"/>
        </action>
    </package>
    <!-- 订单 Action -->
    <package name="shop.order" extends="shop.front" namespace="/product">
        <action name="order_*" class="orderAction" method="{1}">
            .....<!--省略的Action配置 -->
            <interceptor-ref name="customerDefaultStack"/>
        </action>
    </package>
</struts>

```

□ Struts 2 与 Struts 1 的几点不同

Struts 2 与 Struts 1 是完全两个不同的开发框架，除其核心控制器不同以外，还有其他的几个不同点。

(1) 命名空间的不同

在 Struts 2 中的 Filter 默认的扩展名为 “.action”，这是在 default.properties 文件的 “struts.action.extension” 属性中定义的；而 Struts 1 中则通过 <init-pattern> 属性来配置。Struts 1 与 Struts 2 两个框架的命名空间不一样，因此 Struts 1 和 Struts 2 可以在同一个 Web 应用系统中无障碍的共存。

(2) 设置系统属性的不同

Struts 2 的配置信息中不需要通过 <init-param> 来设置系统的属性，它并不是取消了这些属性，而是使用 default.properties 文件作为默认的配置选项文件，并可以通过 “struts.properties” 的文件设置不同的属性值来覆盖默认文件的值实现自己的配置。

(3) 映射文件名的配置参数的不同

在 Struts 2 中没有提供映射文件名的配置参数，取而代之的是默认配置文件 “struts.xml”。

14.6.2 配置 Hibernate

Hibernate 配置文件主要用于配置数据库连接和 Hibernate 运行时所需的各种属性，这个配置文件位

于应用程序或 Web 程序的类文件夹 classes 中。Hibernate 配置文件支持两种形式，一种是 Xml 格式的配置文件，另一种是 Java 属性文件格式的配置文件，采用“键=值”的形式。建议采用 Xml 格式的配置文件。

在 Hibernate 的配置文件中配置连接的数据库的连接信息，数据库方言以及打印 SQL 语句等属性，其关键代码如下：

代码位置：光盘\mr\14\Shop\src\hibernate.cfg.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC
    "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
    "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd" >
<hibernate-configuration>
    <session-factory>
        <!-- 数据库方言 -->
        <property name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>
        <!-- 数据库驱动 -->
        <property name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</property>
        <!-- 数据库连接信息 -->
        <property name="hibernate.connection.url">
            jdbc:mysql://localhost:3306/db_database19</property>
        <property name="hibernate.connection.username">root</property>
        <property name="hibernate.connection.password">111</property>
        <!-- 打印SQL语句 -->
        <property name="hibernate.show_sql">true</property>
        <!-- 不格式化SQL语句 -->
        <property name="hibernate.format_sql">false</property>
        <!-- 为Session指定一个自定义策略 -->
        <property name="hibernate.current_session_context_class">thread</property>
        <!-- C3P0 JDBC连接池 -->
        <property name="hibernate.c3p0.max_size">20</property>
        <property name="hibernate.c3p0.min_size">5</property>
        <property name="hibernate.c3p0.timeout">120</property>
        <property name="hibernate.c3p0.max_statements">100</property>
        <property name="hibernate.c3p0.idle_test_period">120</property>
        <property name="hibernate.c3p0.acquire_increment">2</property>
        <property name="hibernate.c3p0.validate">true</property>
        <!-- 映射文件 -->
        <mapping resource="com/lyq/model/user/User.hbm.xml"/>
        .....<!--省略的映射文件 -->
    </session-factory>
</hibernate-configuration>
```

说明：C3P0 是一个随 Hibernate 一同分发的开发的 JDBC 连接池，它位于 Hibernate 源文件的 lib 目录下。如果在配置文件中设置了 hibernate.c3p0.* 的相关属性，Hibernate 将会使用 C3P0ConnectionProvider 来缓存 JDBC 连接。



14.6.3 配置 Spring

利用 Spring 加载 Hibernate 的配置文件以及 Session 管理类，所以在配置 Spring 的时候，只需要配置 Spring 的核心配置文件 applicationContext-common.xml，其代码如下：

代码位置：光盘\mr\14\Shop\src\applicationContext-common.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:context="http://www.springframework.org/schema/context"
    xmlns:aop="http://www.springframework.org/schema/aop"
    xmlns:tx="http://www.springframework.org/schema/tx"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans-2.14.xsd
        http://www.springframework.org/schema/context
        http://www.springframework.org/schema/context/spring-context-2.14.xsd
        http://www.springframework.org/schema/aop
        http://www.springframework.org/schema/aop/spring-aop-2.14.xsd
        http://www.springframework.org/schema/tx
        http://www.springframework.org/schema/tx/spring-tx-2.14.xsd">
    <context:annotation-config/>
    <context:component-scan base-package="com.lyq"/>
    <!-- 配置sessionFactory -->
    <bean id="sessionFactory"
        class="org.springframework.orm.hibernate3.LocalSessionFactoryBean">
        <property name="configLocation">
            <value>classpath:hibernate.cfg.xml</value>
        </property>
    </bean>
    <!-- 配置事务管理器 -->
    <bean id="transactionManager"
        class="org.springframework.orm.hibernate3.HibernateTransactionManager">
        <property name="sessionFactory">
            <ref bean="sessionFactory" />
        </property>
    </bean>
    <tx:annotation-driven transaction-manager="transactionManager" />
    <!-- 定义Hibernate模板对象 -->
    <bean id="hibernateTemplate" class="org.springframework.orm.hibernate3.HibernateTemplate">
        <property name="sessionFactory" ref="sessionFactory"/>
    </bean>
</beans>
```

14.6.4 配置 web.xml

任何 MVC 框架都需要与 Servlet 应用整合，而 Servlet 则必须在 web.xml 文件中进行配置。web.xml 的配置文件是项目的基本配置文件，通过该文件设置实例化 Spring 容器、过滤器、配置 Struts 2 以及设



置程序默认执行的操作，其关键代码如下：

代码位置：光盘\mr\14\Shop\WebContent\WEB-INF\web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_14.xsd"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_14.xsd"
  id="WebApp_ID" version="2.5">
  <display-name>Shop</display-name>
  <!-- 对Spring容器进行实例化 -->
  <listener>
  <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
  </listener>
  <context-param>
  <param-name>contextConfigLocation</param-name>
  <param-value>classpath:applicationContext-*.xml</param-value>
  </context-param>
  <!-- OpenSessionInViewFilter过滤器 -->
  <filter>
  <filter-name>openSessionInViewFilter</filter-name>
  <filter-class>org.springframework.orm.hibernate3.support.OpenSessionInViewFilter</filter-class>
  </filter>
  <filter-mapping>
  <filter-name>openSessionInViewFilter</filter-name>
  <url-pattern>/*</url-pattern>
  </filter-mapping>
  <!-- Struts 2配置 -->
  <filter>
    <filter-name>Struts 2</filter-name>
    <filter-class>org.apache.Struts 2.dispatcher.ng.filter.StrutsPrepareAndExecuteFilter</filter-class>
  </filter>
  <filter-mapping>
    <filter-name>Struts 2</filter-name>
    <url-pattern>/*</url-pattern>
  </filter-mapping>
  <!-- 设置程序的默认欢迎页面-->
  <welcome-file-list>
  <welcome-file>index.jsp</welcome-file>
  </welcome-file-list>
</web-app>
```

14.7 登录注册模块设计

如果要提高网站的安全性，防止非法用户进入网站，可以让用户进入网站前先进行注册，注册成功的用户才可以通过购物车购买商品。用户注册在大多数网站中都是不可缺少的功能，也是用户参与

网站活动最为直接的桥梁。通过用户注册，可以有效地对用户信息进行采集，并将合法的用户信息保存到指定的数据表中，通常情况下，当用户注册操作完毕，将直接登录该网站。

14.7.1 模块概述

由于天下淘商城分为前台和后台两个部分，所以登录也分为前台登录和后台登录两个部分功能，前台的登录针对的是在天下淘商城注册的会员，后台登录主要针对的是网站管理员，而注册模块主要针对的就是前台想进行购物的游客，如图 14.14 所示。

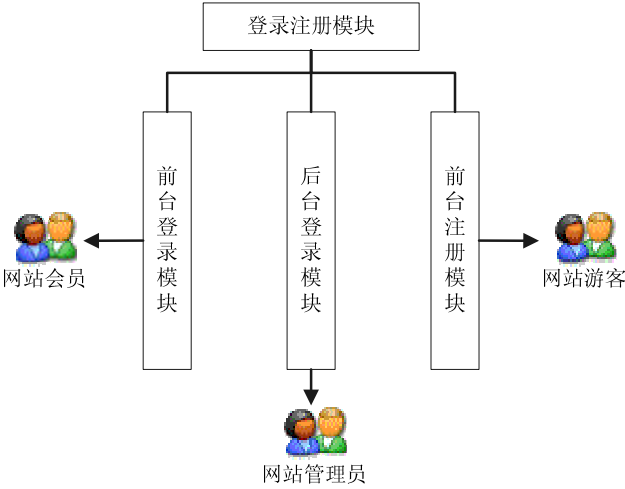



图 14.14 登录注册模块框架图

14.7.2 注册模块的实现

 本模块使用的数据表: tb_customer 表

在安全注册与登录操作过程中，主要是将表单内容进行严格的校验，这样可以提高网站的安全性，防止非法用户进入网站。

在本模块中，用户注册页面为 customer_reg.jsp，如图 14.15 所示，用户注册主要包括用户名、密码、确认密码、邮箱地址、住址以及手机号码 6 个表单控件。其中，用户名要求 5~32 个字符，密码表单与确认密码表单必须一致；邮箱地址表单必须是正确的地址，这里通过 Struts 2 的校验器进行校验；住址与手机号码两个表单主要是在用户购买商品生成订单时直接获取相关的送货信息，方便用户的操作。

图 14.15 会员注册页面

当用户成功注册信息后，就可以进行登录操作，并在天下淘商城中购买商品。

☑ 表单验证

在 Web 开发之中，为了确保数据的安全性，通常情况下都会对页面提交的数据进行统一验证，从而保障数据的合法性。

在 Struts 2 中有两种表单的验证方式，在本模块中将使用 XML 文件对表单中的信息进行合法性验证，利用 `requiredstring` 校验器对 `CustomerAction` 类中的字段进行非空验证；利用 `stringlength` 校验器对 `CustomerAction` 类中的字段长度进行验证，利用 `email` 校验器对邮箱地址的格式进行验证，在 `CustomerAction` 类的包下新建 XML 文件

☑ 保存注册信息

当用户单击会员注册页面中的“注册”超链接时，系统将会发送一个 `customer_reg.html` 的 URL 请求，该请求将会执行 `CustomerAction` 类中的 `save()` 方法，在该方法中首先判断用户名是否可用，如果可用就将注册信息保存在数据库中，否则返回错误信息，其关键代码如下：

代码位置：光盘\mr\14\Shop\src\com\lyq\action\user\CustomerAction

```
public String save() throws Exception{
    ❶ boolean unique = customerDao.isUnique(customer.getUsername());    //判断用户名是否可用
    if(unique){                                                         //如果用户名可用
        customerDao.save(customer);                                     //保存注册信息
        return CUSTOMER_LOGIN;                                          //返回会员登录页面
    }else{
        throw new AppException("此用户名不可用");                    //否则返回页面错误信息
    }
}
```

🔊 关键代码解析

❶ 验证用户名是否唯一： `isUnique()` 方法将以注册的用户名为查询条件，如果返回的结果集中的 List 的长度为大于 0 则返回 `false`，否则返回 `true`。

14.8 前台商品信息查询模块设计

商品是天下淘商城的灵魂，只有好的商品展示以及丰富的商品信息才能吸引顾客的眼球，提高网站的关注度，这也是为企业创造效益的决定性因素，所以天下淘商城的前台商品展示在整个系统中占有非常重要的地位。

14.8.1 模块概述

根据前台的页面设计将前台商品信息查询模块划分为 5 个模块，主要包括商品分类查询、人气商品查询、热销商品查询、推荐商品查询以及商品的模糊查询，如图 14.16 所示。

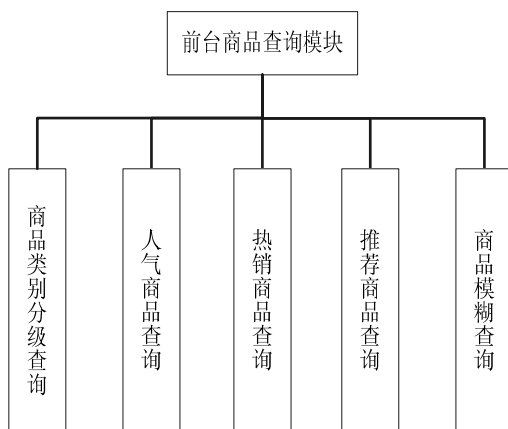


图 14.16 前台商品信息查询模块框架图

14.8.2 前台商品信息查询模块技术分析

☑ 1. 实现分级查询

📖 本模块使用的数据表：tb_productcategory 表。

在前台的首页商品展示中，首先展现给用户的就是商品类别的分级显示，方便用户按类别对商品进行查询，同时也能体现出天下淘商城产品种类的丰富多样。

实现商品类别的分级查询，首先需要查询所有的一级节点，通过公共模块持久化类中封装的 find() 方法实现该功能，在首页的 Action 请求 IndexAction 的 execute() 方法中，调用封装的 find() 方法，其关键代码如下：


代码位置：光盘\mr\14\Shop\src\com\lyq\action\IndexAction

```
public String execute() throws Exception {  
    // 查询所有类别  
    String where = "where parent is null";  
    categories = categoryDao.find(-1, -1, where, null).getList();  
    .....  
    //省略的Setter和Getter方法  
}
```

在 find()方法中含有四个参数,其中“-1”参数分别代表是当前页数和每页显示的记录数,根据这两个参数,“where”参数代表的是查询条件,“null”参数代表是数据排序的条件参数。find()方法会根据提供的两个“-1”参数执行以下代码:

```
// 如果maxResult<0, 则查询所有
if(maxResult < 0 && pageNo < 0){
    list = query.list();           //将查询结果转化为List对象
}
```

14.8.3 商品搜索的实现

 本模块使用的数据表: tb_productinfo 表

在天下淘商城中主要实现普通搜索,在对数据表的简单搜索中,当搜索表单中没有输入任何数据时,单击“搜索”按钮后,可以对数据表中的所有内容进行查询;当在关键字文本框中输入要搜索的内容,单击“搜索”按钮后,可以按关键字内容查询数据表中所有的内容。该功能方便了用户对商品信息的查找,用户可以在首页的文本输入框中输入关键字搜索指定的商品信息,如图 14.17 所示。



图 14.17 商品搜索的效果

商品搜索的方法封装在 ProductAction 类中,通过 HQL 的 like 条件语句实现商品的模糊查询的功能,其关键代码如下:

代码位置: 光盘\mr\14\Shop\src\com\lyq\action\product\ProductAction

```
public String findByName() throws Exception {
    if(product.getName() != null){
        String where = "where name like ?";           //查询的条件语句
    }
```

```

Object[] queryParams = {"%" + product.getName() + "%"};           //为参数赋值
pageModel = productDao.find(pageNo, pageSize, where, queryParams ); //执行查询方法
}
return LIST;                                                       //返回列表首页
}

```

在商品的列表页面中, 通过 Struts 2 的<s:iterator>标签遍历返回的商品 List 集合, 其关键代码如下:

代码位置: 光盘\mr\14\Shop\WebContent\WEB-INF\pages\product\product_list.jsp

```

<s:iterator value="pageModel.list">
    <ul>
        <li>
            <table border="0" width="100%" cellpadding="0" cellspacing="0">
                <tr>
                    <td rowspan="5" width="160">
                        <s:a action="product_select" namespace="/product">
                            <s:param name="id" value="id"></s:param>
                            /upload
                                </s:property value="uploadFile.path"/>">
                        </s:a></td>
                    </tr>
                    <tr bgcolor="#f2eec9">
                        <td align="right" width="90">商品名称: </td>
                        <td><s:a action="product_select" namespace="/product">
                            <s:param name="id" value="id"></s:param>
                            <s:property value="name" />
                        </s:a></td>
                    </tr>
                    <tr>
                        <td align="right" width="90">市场价格: </td>
                        <td><font style="text-decoration: line-through;">
                            <s:property value="marketprice" /> </font></td>
                    </tr>
                    <tr bgcolor="#f2eec9">
                        <td align="right" width="90">天下淘价格: </td>
                        <td><s:property value="sellprice" />
                            <s:if test="sellprice <= marketprice">
                                <font color="red">节省
                                <s:property value="marketprice-sellprice" /></font>
                            </s:if></td>
                    </tr>
                    <tr>
                        <td colspan="2" align="right">
                            <s:a action="product_select" namespace="/product">
                                <s:param name="id" value="id"></s:param>
                                
                            </s:a></td>
                    </tr>
            </table>
        </li>
    </ul>

```




```

        </table>
    </li>
</ul>
</s:iterator>

```

14.8.4 前台商品其他查询的实现

 本模块使用的数据表: tb_productinfo 表

人气商品推荐模块(如图 14.18 所示)、推荐商品模块(如图 14.19 所示)、热销商品模块(如图 14.20 所示)三个查询方式的实现方式基本相同,都是通过条件语句进行排序查询,只不过查询的条件不同。



图 14.18 人气商品模块



图 14.19 推荐商品模块



图 14.20 热销商品模块

☑ 人气商品模块的实现

人气商品的定义是按照商品的浏览量最多进行定义的,系统将筛选出商品中浏览量最多的几件商品进行展示,商品结果集中的信息将按照商品浏览量进行倒序排列,其实现的方法封装在 `ProductAction` 类中,其关键代码如下:

代码位置: 光盘\mr\14\Shop\src\com\lyq\action\product\ProductAction

```

public String findByClick() throws Exception {
    Map<String, String> orderby = new HashMap<String, String>();           //定义Map集合
    orderby.put("clickcount", "desc");                                     //为Map集合赋值
    pageModel = productDao.find(1, 8, orderby);                             //执行查找方法
    return "clickList";                                                    //返回product_click_list.jsp页面
}

```

在调用的 `find()` 方法中使用了三个参数,前两个参数设置显示的起始位置和显示记录数,最后一个参数是商品信息排序的条件,程序最终返回到 `product_click_list.jsp` 页面。代码如下:

代码位置: 光盘\mr\14\Shop\WebContent\WEB-INF\pages\product\product_list.jsp

```

<:set var="context_path" value="#request.get('javax.servlet.forward.context_path')"></s:set>
<table width="193" height="23" border="0" cellpadding="0" cellspacing="0">
    <s:iterator value="pageModel.list">
    <tr>

```

```
  |
```

☑ 推荐商品和热销商品模块的实现

推荐商品和热销商品模块的实现与商品搜索模块的实现比较类似，都是通过 hql 的排序语句实现的，推荐商品为商品推荐字段 commend 为 true 的商品，并且按商品销量的倒序排列，其实现的关键的代码如下：

代码位置：光盘\mr\14\Shop\src\com\lyq\action\product\ProductAction

```

public String findByCommend() throws Exception{
    Map<String, String> orderby = new HashMap<String, String>();           //定义Map集合
    orderby.put("sellCount", "desc");                                     //为Map集合赋值
    String where = "where commend = ?";                                   //设置条件语句
    Object[] queryParams = {true};                                       //设置参数值
    pageModel = productDao.find(where, queryParams, orderby, pageNo, pageSize); //执行查询方法
    return "findList";                                                    //返回推荐商品页面
}

```

热销商品为销售量较多的商品，只需按照商品销量的倒序进行排列即可，并以分页的方式取出前 6 条信息，其实现的关键代码如下：

代码位置：光盘\mr\14\Shop\src\com\lyq\action\product\ProductAction

```

public String findBySellCount() throws Exception{
    Map<String, String> orderby = new HashMap<String, String>();           //定义Map集合
    orderby.put("sellCount", "desc");                                     //为Map集合赋值
    pageModel = productDao.find(1, 6, orderby);                           //执行查询方法
    return "findList";                                                    //返回热销商品页面
}

```

两个模块在页面中展示的信息方式相同，仅以推荐产品为例，其代码如下：

代码位置：光盘\mr\14\Shop\WebContent\WEB-INF\pages\product\product_list.jsp

```

<s:set var="context_path"
value="#request.get('javax.servlet.forward.context_path')"></s:set>
<div style="width: 195px;">
<s:iterator value="pageModel.list">
<div style="float: left; width:45%; text-align: center;">
<s:a action="product_select" namespace="/product">
<s:param name="id" value="id"></s:param>

/upload/<s:property value="uploadFile.path"/>">
<p style="width: 80px;"><s:property value="name"/></p>

```

```

</s:a>
</div>
</s:iterator>
</div>

```

在 Struts 2 的前台 Action 配置文件 struts-front.xml 中，配置前台商品管理模块的 Action 以及视图映射关系，关键代码如下：

代码位置：光盘\mr\14\Shop\src\com\lyq\action\struts-front.xml

```

<!-- 商品Action -->
<package name="shop.product" extends="shop-default" namespace="/product">
  <action name="product_*" class="productAction" method="{1}">
    <result name="list">/WEB-INF/pages/product/product_list.jsp</result>
    <result name="select">/WEB-INF/pages/product/product_select.jsp</result>
    <result name="clickList">/WEB-INF/pages/product/product_click_list.jsp</result>
    <result name="findList">/WEB-INF/pages/product/product_find_list.jsp</result>
  </action>
</package>

```

14.8.5 单元测试

为了确保前台商品信息查询模块程序能够正确运行，需要对该模块进行测试，并对模块中容易出现的错误进行分析。

所有的商品查询模块都是基于公共持久化类 BaseDao 中的 find() 方法，重点对该方法进行测试。可以使用 JUnit 对 find() 方法进行单元测试。

测试步骤如下：

(1) 鼠标右击项目，在弹出的快捷菜单中选择“构建路径”，在弹出的对话框中单击“添加库”按钮并选择 JUnit，如图 14.21 所示。

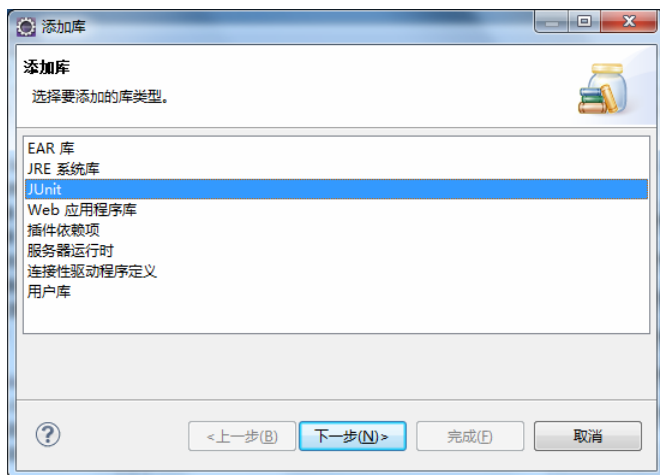


图 14.21 添加“JUnit”库对话框

(2) 单击“下一步”按钮，选择 JUnit 的版本，本项目中选择的 JUnit 版本为“JUnit4”，单击“完

成”按钮，完成 JUnit 测试环境的搭建。

(3) 创建 find()方法所在的 DaoSupport 类的测试类。右击 DaoSupport.java 类文件，在弹出的快捷菜单中选择“新建/JUnit 测试用例”，如图 14.22 所示，在弹出的“JUnit 测试用例”对话框中对生成的测试类进行配置，选择要创建类的名称、测试方法以及生成类所在的包文件，如图 14.23 所示。

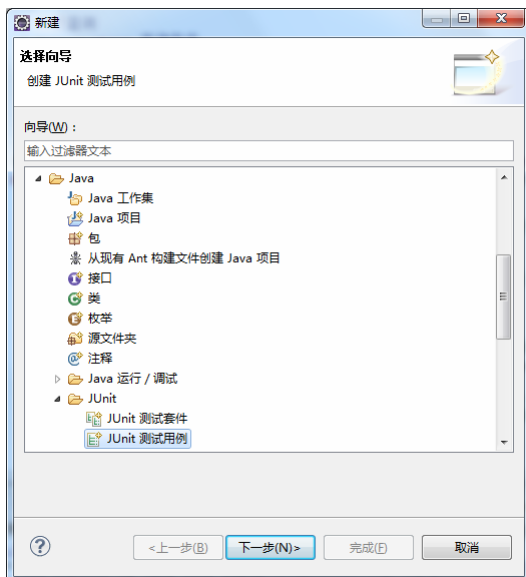


图 14.22 “新建”的提示对话框

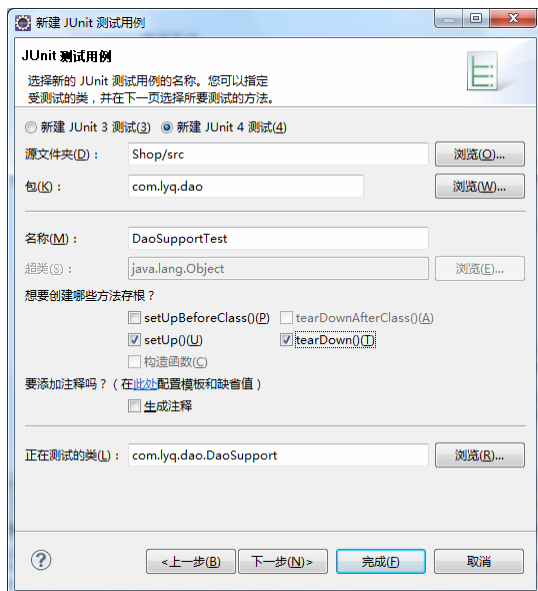


图 14.23 “新建测试用例”的提示对话框

(4) 单击“新建测试用例”对话框中的“下一步”按钮，在弹出的新对话框中选择需要测试的方法，这里仅以 find()方法中参数最多的方法为例，如图 14.19 所示。

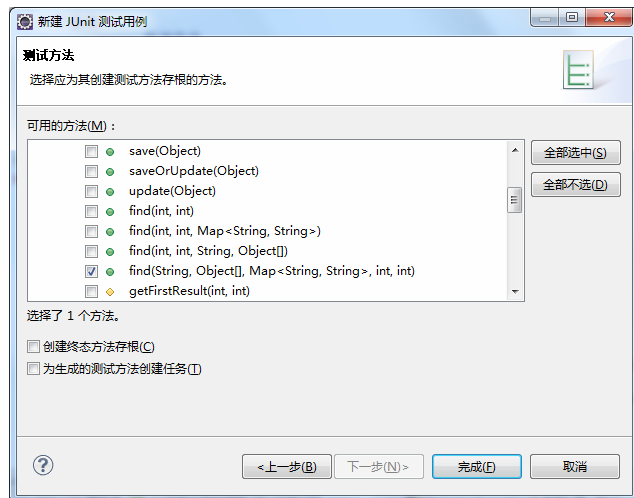


图 14.19 选择类中需要测试的方法

(5) 单击对话框中的“完成”按钮，完成 DaoSupport 类的测试类 DaoSupportTest 的创建。并对 DaoSupportTest 进行编码操作，详细代码如下：

代码位置：光盘\mr\14\Shop\src\com\lyq\dao\DaoSupportTest

```
public class DaoSupportTest {
    private DaoSupport<Order> daoSupport;           //公共持久化类
    @Before
    public void setUp() throws Exception {           //初始化方法，最先执行的方法
        daoSupport = new DaoSupport<Order>();
    }
    @After
    public void tearDown() throws Exception {        //清理方法，最后执行的方法
        daoSupport=null;                             //销毁对象
    }
    @Test
    public void testFindIntInt() {
        Map<String, String> orderby = new HashMap<String, String>(1); //定义Map集合
        orderby.put("createTime", "desc");                //设置按创建时间倒序排列
        PageModel<Order> pageModel = daoSupport.find(1, 3); //执行查询方法
        assertNotNull("无法获取数据！", pageModel);        //判断find()方法是否成功查询到值
    }
}
```

(6) 在程序文件中右击鼠标，在弹出的快捷菜单中选择“运行方式/JUnit 测试”，如果程序出现问题，说明程序存在 bug 或者异常信息，运行后的效果如图 14.25 所示，如果程序没有问题，运行后的效果如图 14.26 所示。

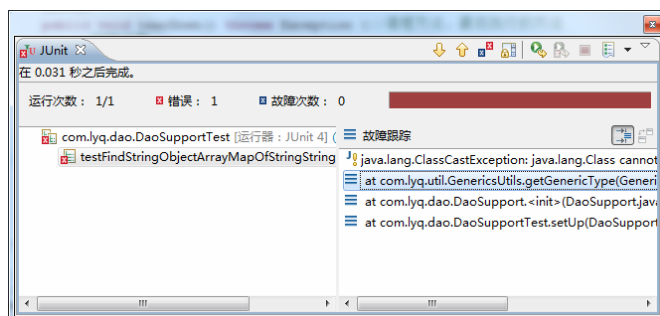


图 14.25 测试失败效果图

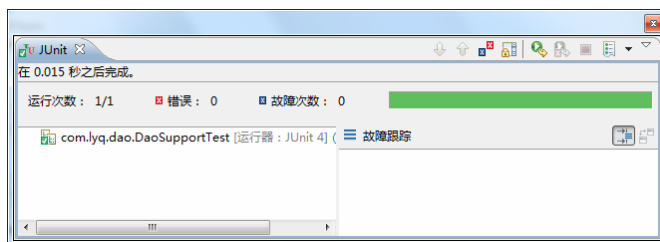


图 14.26 测试成功效果图

14.9 购物车模块设计

购物车是商务网站中必不可少的功能，购物车的设计很大程度上会决定网站是否受到用户的关注。商务网站中的购物车会将用户选购的未结算的商品保存一段时间，防止错误或意外发生时购物车中的商品丢失，方便了用户的使用。所以在天下淘商城中购物车也是必不可少的一个模块。

14.9.1 模块概述

天下淘商城购物车实现的主要功能包括添加选购的新商品、自动更新选购的商品数量、清空购物车、自动调整商品总价格以及生成订单信息等。本模块实现的购物车的功能结构如图 14.27 所示。

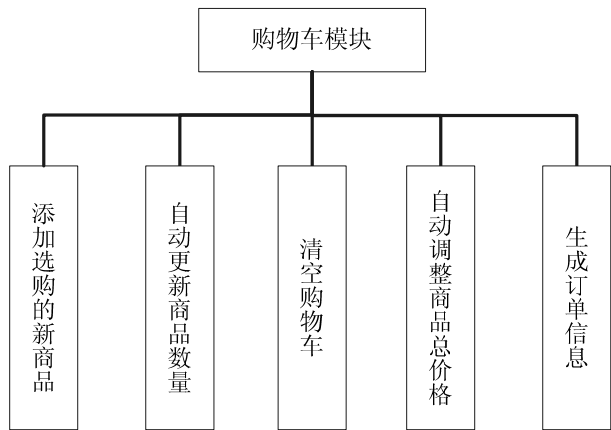


图 14.27 购物车模块的功能结构图

如果用户需要选购商品，必须登录，否则用户无法使用购物车功能。当用户进入购物车后，可以进行结算、清空购物车以及继续选购等操作。当用户进入结算操作后，需要填写订单信息，并选择支付方式，当用户确认支付时系统会生成相应的订单信息。其功能流程图如图 14.28 所示。

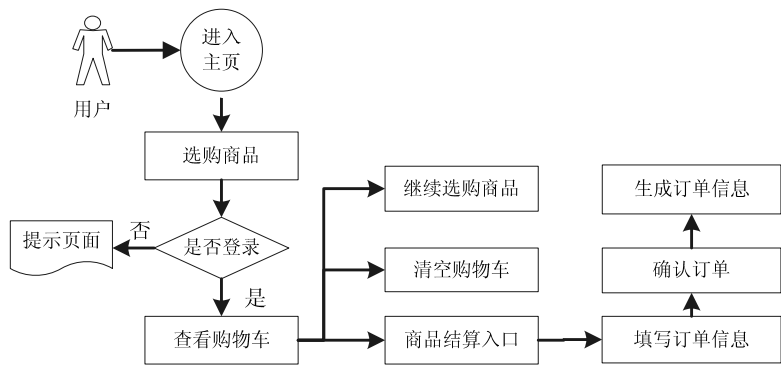


图 14.28 购物车流程图

14.9.2 购物车模块技术分析

消除无用订单信息

在开发时一定要注意有时加入购物车中没有任何的商品采购信息，当用户确认订单的时候，系统同样会生成一个消费金额为 0.0 元且无任何订单条目的订单信息，在系统中的该信息是没有任何意义的，而且有可能导致系统不可预知的错误，为了避免这种情况的发生，需要修改前台订单的保存方法，即 OrderAction 类中的 save() 方法，判断购物车对象是否为空，如果为空返回错误信息的提示页面，不进行任何的后续操作。，在 save() 方法中添加如下代码：

代码位置：光盘\mr\14\Shop\src\com\lyq\action\order\OrderAction

```
public String save() throws Exception {  
    ..... //省略的代码  
    Set<OrderItem> cart = getCart(); //获取购物车  
    if(cart.isEmpty()){ //判断条目信息是否为空  
        return ERROR; //返回订单信息错误提示页面  
    }  
    ..... //省略的代码  
}
```

并创建前台订单错误的提示页面 order_error.jsp，当用户误操作导致的系统生成的错误订单信息将不会保存到数据库中，而是跳转到错误提示页面。

14.9.3 购物车基本功能的实现

本模块使用的数据表：tb_productinfo 表和 tb_orderitem 表

购物车的基本功能包括向购物车中添加商品、清空购物车以及删除购物车中指定的商品订单条目信息三项功能，购物车的功能是基于 Session 变量实现的，Session 充当了一个临时信息存储平台，当 Session 失效后，其保存的购物车信息也将全部丢失。其效果图如下图 14.29 所示：

> 我的购物车				
				总价: ¥0元
商品名称	市场价	价格	数量	删除
Java 编程词典	¥150.0元	¥120.0元 为您节省: ¥30.0元	1	删除
清空购物车 继续挑选其它商品 收银台结账				

图 14.29 购物车内的商品信息

☑ 向购物车添加商品

购物车的主要工作就是保存用户的商品购买信息，当登录会员浏览商品详细信息，并单击页面上的“立即购买”超链接时，系统就会将该商品放入购物车内，如图 14.29 所示。

在本系统中，将购物车的信息保存在 Session 变量中，其保存的是商品的购买信息，也就是订单的

条目信息。所以在向购物车添加商品时，首先要获取商品 ID 进行判断，如果购物车中存在相同的 ID 值，就修改该商品的数量，自动加 1；如果购物车中无相同 ID，则向购物车中添加新的商品购买信息，向购物车添加商品信息的方法封装在 CartAction 类中，其关键代码如下：

代码位置：光盘\mr\14\Shop\src\com\lyq\action\order\CartAction

```
public String add() throws Exception {
    if(productId != null && productId > 0){
        Set<OrderItem> cart = getCart();           //获取购物车
        // 标记添加的商品是否是同一件商品
        boolean same = false;                       //定义same布尔变量
        for (OrderItem item : cart) {               //遍历购物车中的信息
            if(item.getProductId() == productId){
                // 购买相同的商品，更新数量
                item.setAmount(item.getAmount() + 1);
                same = true;                         //设置same变量为“true”
            }
        }
        // 不是同一件商品
        if(!same){
            OrderItem item = new OrderItem();        //实例化订单条目信息实体对象
            ProductInfo pro = productDao.load(productId); //加载商品对象
            item.setProductId(pro.getId());           //设置id
            item.setProductName(pro.getName());       //设置商品名称
            item.setProductPrice(pro.getSellprice()); //设置商品销售价格
            item.setProductMarketprice(pro.getMarketprice()); //设置商品市场价格
            cart.add(item);                          //将信息添加到购物车中
        }
        session.put("cart", cart);                  //将购物车保存在Session对象中
    }
    return LIST;
}
```

程序运行结束够将返回订单条目信息的列表页面，即 cart_list.jsp，代码如下：

代码位置：光盘\mr\14\Shop\WebContent\WEB-INF\pages\cart\cart_list.jsp

```
//遍历Session对象：通过Struts 2的<s:iterator>标签遍历Session对象中存放的订单条目信息
<s:iterator value="#session.cart">
    <s:set value="%{#sumall +productPrice*amount}" var="sumall" />
    .....<!-- 省略的布局代码 -->
    <td width="213" height="30" align="center">
        <s:property value="productName" /></td>
        <td width="130" align="center">
            <span style="text-decoration: line-through;"> ¥
            <s:property value="productMarketprice" />元</span></td>
        <td width="130" align="center">¥
            <s:property value="productPrice" />元<br>为您节省： ¥
        //计算“为您节省”金额：其金额的计算公式为（市场价格-销售价格）
        <s:property value="productMarketprice*amount - productPrice*amount" />元</td>
        <td width="104" align="center" class="red">
            <s:property value="amount" /></td>
```



```

<td width="111" align="center"><s:a action="cart_delete" namespace="/product">
  <s:param name="productId" value="productId"></s:param>
  
</s:a></td>
.....<!-- 省略的布局代码 -->
</s:iterator>

```

☑ 删除购物车中指定商品订单条目信息

当用户想删除购物车中的某个商品的订单条目信息时，可以单击信息后的“删除”超链接，就会自动清除该商品的订单条目信息。实现该方法的关键代码如下：

代码位置：光盘\mr\14\Shop\src\com\lyq\action\order\CartAction

```

public String delete() throws Exception {
    Set<OrderItem> cart = getCart();           // 获取购物车
    // 此处使用Iterator，否则出现java.util.ConcurrentModificationException
    Iterator<OrderItem> it = cart.iterator();
    while(it.hasNext()){                       //使用迭代器遍历商品订单条目信息
        OrderItem item = it.next();
        if(item.getProductId() == productId){
            it.remove();                       //移除商品订单条目信息
        }
    }
    session.put("cart", cart);                //将清空后的信息重新放入Session中
    return LIST;                              //返回购物车页面
}

```

☑ 清空购物车

清空购物车的实现较为简单，由于信息是暂时存放于 Session 对象中，所以用户在执行情况购物车操作时，直接清空 Session 对象即可。当用户单击购物车页面中的“清空购物车”超链接时，系统会向服务器发送一个 cart_clear.html 的 URL 请求，该请求执行的是 CartAction 类中的 clear()方法。

代码位置：光盘\mr\14\Shop\src\com\lyq\action\order\CartAction

```

public String clear() throws Exception {
    session.remove("cart");                   //移除信息
    return LIST;                             //返回订单列表页面
}

```

☑ 查找购物信息

当用户登录后，可以单击首页顶部的“购物车”链接，查看自己的购物车的相关信息。

当用户单击“购物车”超链接后，系统会发送一个 cart_list.html 的 URL 请求，该请求执行的是 CartAction 中的 list()方法，实现该方法的关键代码如下：

代码位置：光盘\mr\14\Shop\src\com\lyq\action\order\CartAction

```

public String list() throws Exception {
    return LIST;                             //返回购物车页面
}

```

在购物车页面中是通过 Struts 2 的<s:iterator>标签遍历 Session 对象中购物车的相关信息的，在程序模块中并不需要执行任何的操作，只需要返回购物车页面即可。

在 Struts 2 的前台 Action 配置文件 struts-front.xml 中，配置购物车管理模块的 Action 以及视图映




射关系，关键代码如下：

代码位置：光盘\mr\14\Shop\src\com\lyq\action\struts-front.xml

```
<!-- 购物车Action -->
<package name="shop.cart" extends="shop.front" namespace="/product">
    <action name="cart_*" class="cartAction" method="{1}">
        <result name="list">/WEB-INF/pages/cart/cart_list.jsp</result>
        <interceptor-ref name="customerDefaultStack"/>
    </action>
</package>
```

14.9.4 订单相关功能的实现

 本模块使用的数据表：tb_order 表

要为选购的商品进行结算，就需要先生成一个订单，订单信息中包括收货人信息、送货方式、支付方式、购买的商品以及订单总价格，当用户在购物车中单击“收银台结账”超链接后，将进入到订单填写的页面，其中包含了订单的基本信息，例如收货人姓名、收货人地址、收货人电话以及支付方式，该页面为 order_add.jsp，如图 14.30 所示。下面介绍实现过程。

> 我的订单

收货人姓名：

收货人地址：

收货人电话：

支付方式：

☐ 网上银行

☐ 支付宝

☐ 货到付款

☒ 邮局汇款

收款人地址：吉林省长春市xxx大厦xxx室 收款人姓名：xxx 收款人邮编：xxxx

付款

图 14.30 天下淘商城订单信息添加页面

☒ 下订单操作

当用户单击购物车“收银台结账”超链接时，系统将发送一个 order_add.html 的 URL 请求，该请求执行的是 OrderAction 类中的 add()方法，通过该方法将用户的基本信息从 Session 对象中取出，添加到订单表单中指定的位置，并跳转到我的订单页面，其关键代码如下：

代码位置：光盘\mr\14\Shop\src\com\lyq\action\order\OrderAction

```
public String add() throws Exception {
```

```

order.setName(getLoginCustomer().getUsername()); //设置收货人姓名
order.setAddress(getLoginCustomer().getAddress()); //设置收货人地址
order.setMobile(getLoginCustomer().getMobile()); //设置收货人电话
return ADD; //返回我的订单页面
}

```

☑ 订单确认

在我的订单页面单击“付款”超链接，如图 14.30 所示，将进入订单确认的页面，如图 14.31 所示，在该页面将显示订单的条目信息，也就是用户购买商品的信息清单，以便用户进行确认。

图 14.31 订单确认页面

当用户单击我的订单页面中的“付款”超链接时，系统将发送一个 order_confirm.html 的 URL 请求，该请求执行的是 OrderAction 类中的 confirm() 方法，该方法中只是实现的页面的跳转操作，其关键代码如下：

代码位置：光盘\mr\14\Shop\src\com\lyq\action\order\OrderAction

```

public String confirm() throws Exception {
    return "confirm"; //返回订单确认页面
}

```

该方法将返回 order_confirm.jsp，该页面即为订单确认页面，其订单条目信息的显示与购物车页面中订单条目信息显示的方法相同。

☑ 订单保存

在订单确认页面单击“付款”超链接时，系统将正式生成用户的购物订单，标志着正式的交易开始进行，该链接将会触发 OrderAction 类中的 save() 方法，save() 方法将把订单信息保存到数据库，其关键代码如下：

代码位置：光盘\mr\14\Shop\src\com\lyq\action\order\OrderAction

```

public String save() throws Exception {
    if(getLoginCustomer() != null){ //如果用户已登录
        order.setOrderId(StringUtil.createOrderId()); // 设置订单号
        order.setCustomer(getLoginCustomer()); // 设置所属用户
        Set<OrderItem> cart = getCart(); // 获取购物车
        // 依次将更新订单项中的商品的销售数量
    }
}

```

```

        for(OrderItem item : cart){
            Integer productId = item.getProductId();
            ProductInfo product = productDao.load(productId);
            product.setSellCount(product.getSellCount() + item.getAmount());
            productDao.update(product);
        }
        order.setOrderItems(cart);
        order.setOrderState(OrderState.DELIVERED);
        float totalPrice = 0f;
        for (OrderItem orderItem : cart) {
            totalPrice += orderItem.getProductPrice() * orderItem.getAmount();
        }
        order.setTotalPrice(totalPrice);
        orderDao.save(order);
        session.remove("cart");
    }

    return findByCustomer();
}

```

//遍历购物车中的订单条目信息
 //获取商品ID
 //装载商品对象
 //更新商品销售数量
 //修改商品信息
 // 设置订单项
 // 设置订单状态
 // 计算总额的变量
 //遍历购物车中的订单条目信息
 //商品单价*商品数量
 //设置订单的总价格
 //保存订单信息
 //清空购物车
 //返回消费者订单查询的方法

执行 save()方法后将返回订单查询的方法 findByCustomer(), 在该方法中将以登录用户的 ID 为查询条件, 查询该用户的所有订单信息, 其关键代码如下:

代码位置: 光盘\mr\14\Shop\src\com\lyq\action\order\OrderAction

```

public String findByCustomer() throws Exception {
    if(getLoginCustomer() != null){
        String where = "where customer.id = ?";
        Object[] queryParams = {getLoginCustomer().getId()};
        Map<String, String> orderby = new HashMap<String, String>(1);
        orderby.put("createTime", "desc");
        pageModel = orderDao.find(where, queryParams, orderby, pageNo, pageSize);
    }
    return LIST;
}

```

//如果用户已登录
 //将用户id设置为查询条件
 //创建对象数组
 //创建Map集合
 //设置排序条件及方式
 //执行查询方法
 //返回订单列表页面

该方法将返回用户的订单列表页面 order_list.jsp。

在 Struts 2 的前台 Action 配置文件 struts-front.xml 中, 配置前台订单管理模块的 Action 以及视图映射关系, 关键代码如下:

代码位置: 光盘\mr\14\Shop\src\com\lyq\action\struts-front.xml

```

<!-- 订单Action -->
<package name="shop.order" extends="shop.front" namespace="/product">
    <action name="order_*" class="orderAction" method="{1}">
        <result name="add">/WEB-INF/pages/order/order_add.jsp</result>
        <result name="confirm">/WEB-INF/pages/order/order_confirm.jsp</result>
        <result name="list">/WEB-INF/pages/order/order_list.jsp</result>
        <result name="error">/WEB-INF/pages/order/order_error.jsp</result>
        <interceptor-ref name="customerDefaultStack"/>
    </action>
</package>

```

14.10 后台商品管理模块设计

商品是天下淘商城的灵魂，如何管理好琳琅满目的商品信息也是天下淘商城后台管理的一个难题，良好后台商品管理机制是一个商务网站的基石，如果没有商品信息维护，商务网站将没有意义。

14.10.1 模块概述

根据商务网站的基本要求，天下淘商城网站的商品管理模块主要实现商品信息查询、修改商品信息、删除商品信息以及添加商品信息等功能。后台商品管理模块的框架图如图 14.32 所示。

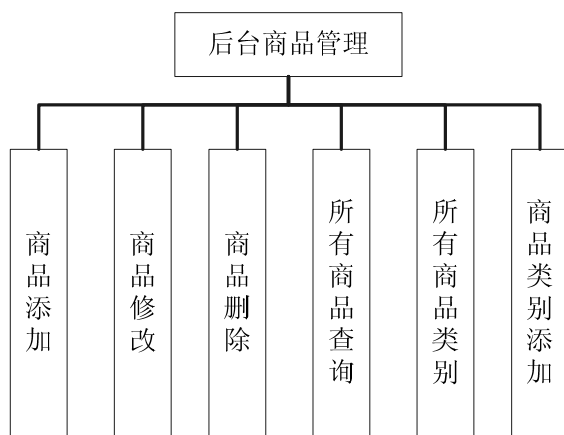


图 14.32 后台商品管理模块框架图

14.10.2 后台商品管理模块技术分析


解决 Struts 2 中文乱码问题

解决 Struts 2 的乱码问题可以在 `struts.properties` 文件进行如下配置。

```
struts.i18n.encoding=UTF-8
```

“`struts.i18n.encoding`”用来设置 Web 的默认编码方式，天下淘商城使用了 UTF-8 作为默认的编码方式，虽然该方法可以有效解决表单的中文乱码问题，但是该模式要求表单的 `method` 属性必须为 `post`，由于 Struts 2 中的 `form` 表单标签默认的 `method` 属性就为 `post`，所以不必再进行额外的设置，如果页面中的表单没有使用 Struts 2 的表单标签，需要在表单中指定 `method` 的属性值。

14.10.3 商品管理功能的实现

 本模块使用的数据表：`tb_productinfo` 表

在商品管理的基本模块中，包括商品的查询、修改、删除以及添加等功能。

☒ 后台商品查询

在天下淘商城的后台管理页面中，单击左侧导航栏中的“查看所有商品”超链接，显示所有商品查询页面的运行效果如图 14.33 所示。

ID	商品名称	所属类别	采购价格	销售价格	是否推荐	适应性别	编辑	删除
1	Java 编程词典	软件	98.0	120.0	true	男		
2	C# 编程词典	软件	98.0	120.0	true	男		
3	.NET 编程词典	软件	98.0	120.0	true	男		

[首页](#)
[上一页](#)
[\[1\]](#)
[下一页](#)
[尾页](#)

图 14.33 后台商品信息列表页面

后台商品列表页面实现的关键代码如下：

代码位置：光盘\mr\14\Shop\WebContent\WEB-INF\pages\admin\product\product_list.jsp

```
<table width="693" height="29" border="0" class="word01">
    <tr>
        <td width="37" height="27" align="center">ID</td>
        <td width="120" align="center">商品名称</td>
        <td width="78" align="center">所属类别</td>
        <td width="79" align="center">采购价格</td>
        <td width="79" align="center">销售价格</td>
        <td width="79" align="center">是否推荐</td>
        <td width="79" align="center">适应性别</td>
        <td width="52" align="center">编辑</td>
        <td width="52" align="center">删除</td>
    </tr>
</table>
</div>
<div id="right_mid">
<div id="tiao">
<table width="693" height="29" border="0">
    <s:iterator value="pageModel.list">
    <tr>
        <td width="37" height="27" align="center"><s:property value="id" /></td>
        <td width="120" align="center"><s:a action="product_edit" namespace="/admin/product">
            <s:param name="id" value="id"></s:param><s:property value="name" /></s:a></td>
        <td width="78" align="center"><s:property value="category.name" /></td>
        <td width="79" align="center"><s:property value="baseprice" /></td>
        <td width="79" align="center"><s:property value="sellprice" /></td>
        <td width="79" align="center"><s:property value="commend" /></td>
        <td width="79" align="center"><s:property value="sexrequest.name" /></td>
        <td width="52" align="center"><s:a action="product_edit" namespace="/admin/product">
            <s:param name="id" value="id"></s:param>
            </s:a></td>
        <td width="52" align="center"><s:a action="product_del" namespace="/admin/product">
            <s:param name="id" value="id"></s:param>
            </s:a></td>
    </tr>
    </s:iterator>
</table>
</div>
</div>
```

```

        </tr>
    </s:iterator>
</table>

```


当用户单击该链接时系统将会发送一个 `product_list.html` 的 URL 请求, 该请求执行的是 `ProductAction` 类中的 `list()` 方法, `ProductAction` 类继承了 `BaseAction` 类和 `ModelDriven` 接口, 其关键代码如下:

代码位置: 光盘\mr\14\Shop\src\com\lyq\action\order\ProductAction

```

public String list() throws Exception {
    pageModel = productDao.find(pageNo, pageSize);           //调用公共的查询方法
    return LIST;                                             //返回后台商品列表页面
}

```

当用户单击列表中的商品名称超链接或是列表中的  图标, 将进入商品信息的编辑页面, 如图 14.34 所示, 在该页面可以对商品的信息进行修改, 该操作触发的是商品详细信息的查找方法, `ProductAction` 类中的 `edit()` 方法, 该方法将以商品的 ID 值作为查询条件, 其关键代码如下:

代码位置: 光盘\mr\14\Shop\src\com\lyq\action\order\ProductAction

```

public String edit() throws Exception {
    this.product = productDao.get(product.getId());           //执行封装的查询方法
    createCategoryTree();                                     //生成商品的类别树
    return EDIT;                                              //返回商品信息编辑页面
}

```



图 14.34 商品信息编辑页面

商品编辑页面与商品添加页面的实现代码是基本相同的, 区别是在编辑页面中需要显示查询到的商品信息, 商品编辑页面的关键代码如下:

代码位置: 光盘\mr\14\Shop\WebContent\WEB-INF\pages\admin\product\product_edit.jsp

```

<table width="685" height="19" border="0">
    <tr>
        <td width="119" height="22" bgcolor="#c6e8ff" align="right">商品名称: </td>

```

```

<td width="119" height="22" bgcolor="#c6e8ff" align="right">
<s:textfield name="name"></s:textfield></td>
<td rowspan="7">

/upload/<s:property value="uploadFile.path"/>"></td>
</tr>
<tr>
<td width="119" height="22" bgcolor="#c6e8ff" align="right">选择类别: </td>
<td colspan="2">
<s:select name="category.id" list="map" value="category.id"></s:select></td>
</tr>
<tr>
<td width="119" height="22" bgcolor="#c6e8ff" align="right">采购价格: </td>
<td colspan="2"><s:textfield name="baseprice"></s:textfield></td>
</tr>
<tr>
<td width="119" height="22" bgcolor="#c6e8ff" align="right">市场价格: </td>
<td colspan="2"><s:textfield name="marketprice"></s:textfield></td>
</tr>
<tr>
<td width="119" height="22" bgcolor="#c6e8ff" align="right">销售价格: </td>
<td colspan="2"><s:textfield name="sellprice"></s:textfield></td>
</tr>
<tr>
<td width="119" height="22" bgcolor="#c6e8ff" align="right">是否为推荐: </td>
<td colspan="2">
<s:radio name="commend" list="#{'true': '是', 'false': '否'}" value="commend"></s:radio></td>
</tr>
<tr>
<td width="119" height="22" bgcolor="#c6e8ff" align="right">所属性别: </td>
<td colspan="2">
<s:select name="sexrequest" list="@com.lyq.model.Sex@getValues()"
value="sexrequest.getName()"></s:select></td>
</tr>
<tr>
<td width="119" height="22" bgcolor="#c6e8ff" align="right">上传图片: </td>
<td colspan="2"><s:file id="file" name="file"></s:file></td>
</tr>
<tr>
<td width="119" height="22" bgcolor="#c6e8ff" align="right">商品说明: </td>
<td colspan="2"><s:textarea name="description" cols="50" rows="6"></s:textarea></td>
</tr>
</table>

```

☒ 商品修改

当用户编辑完商品信息，单击页面的“提交”超链接，系统将会把用户修改后的信息保存到数据库中，该操作会发送一个 `product_save.html` 的 URL 请求，它会调用 `ProductAction` 类中的 `save()` 方法，在 `save()` 方法包括图片的上传和向数据表中添加数据的操作，其具体的实现代码如下：

代码位置：光盘\mr\14\Shop\src\com\lyq\action\order\ProductAction


```
public String save() throws Exception{
    if(file != null ){
        //获取服务器的绝对路径
        String path = ServletActionContext.getServletContext().getRealPath("/upload");
        File dir = new File(path);
        if(!dir.exists()){
            //如果文件夹不存在
            dir.mkdir();
            //创建文件夹
        }
        String fileName = StringUtl.getStringTime() + ".jpg"; //自定义图片名称
        FileInputStream fis = null; //输入流
        FileOutputStream fos = null; //输出流
        try {
            fis = new FileInputStream(file); //根据上传文件创建InputStream实例
            fos = new FileOutputStream(new File(dir,fileName)); //创建写入服务器地址的输出流对象
            byte[] bs = new byte[1019 * 4]; //创建字节数组实例
            int len = -1;
            while((len = fis.read(bs)) != -1){
                //循环读取文件
                fos.write(bs, 0, len); //向指定的文件夹中写数据
            }
            UploadFile uploadFile = new UploadFile(); //实例化对象
            uploadFile.setPath(fileName); //设置文件名称
            product.setUploadFile(uploadFile); //设置上传路径
        } catch (Exception e) {
            e.printStackTrace();
        } finally{
            fos.flush();
            fos.close();
            fis.close();
        }
        //如果商品类别和商品类别ID不为空，则保存商品类别信息
        if(product.getCategory() != null && product.getCategory().getId() != null){
            product.setCategory(categoryDao.load(product.getCategory().getId()));
        }
        //如果上传文件和上传文件ID不为空，则保存文件的上传路径信息
        if(product.getUploadFile() != null && product.getUploadFile().getId() != null){
            product.setUploadFile(uploadFileDao.load(product.getUploadFile().getId()));
        }
        productDao.saveOrUpdate(product); //保存商品信息
        return list(); //返回商品的查询方法
    }
}
```

文件的上传是网络中应用最为广泛的一种技术，在 Web 应用中实现文件上传需要通过 Form 表单实现，此时表单必须以 POST 方式提交（**Struts 2 标签的 form 表单默认提交方式为 POST**），并且必须设置 **enctype="multipart/form-data"** 属性，在表单中需要实现一个或多个文件选择框供用户选择文件。当提交表单后，选择的文件内容会通过流的方式进行传递，在接收表单的 Servlet 或 JSP 页面中获取该流并将流中的数据读到一个字节数组中，此时字节数组中存储了表单请求中的内容，其中包括了所有



上传文件的内容，因此还需要从中分离出每个文件自己的内容，最后将分离出的这些文件写到磁盘中，完成上传操作。需要注意的是在进行分离的过程中，操作的内容是以字节形式存在的。

☑ 商品删除

当用户单击列表中的  图标，将执行商品信息的删除操作，该操作将会向系统发送一个 product_del.html 的 URL 请求，它将触发 ProductAction 类中的 del() 方法，该方法将以商品的 ID 为参数，执行持久化类中封装的 delete() 方法，delete() 方法中调用的是 Hibernate 的 Session 对象中的 delete() 方法，其关键代码如下：

代码位置：光盘\mr\14\Shop\src\com\lyq\action\order\ProductAction

```
public String del() throws Exception {
    productDao.delete(product.getId());           //执行删除操作
    return list();                                //返回商品列表查找方法
}
```

☑ 商品添加

当用户单击后台管理页面左侧导航栏中的“商品添加”超链接时，将会进入商品添加的页面，如图 14.35 所示。

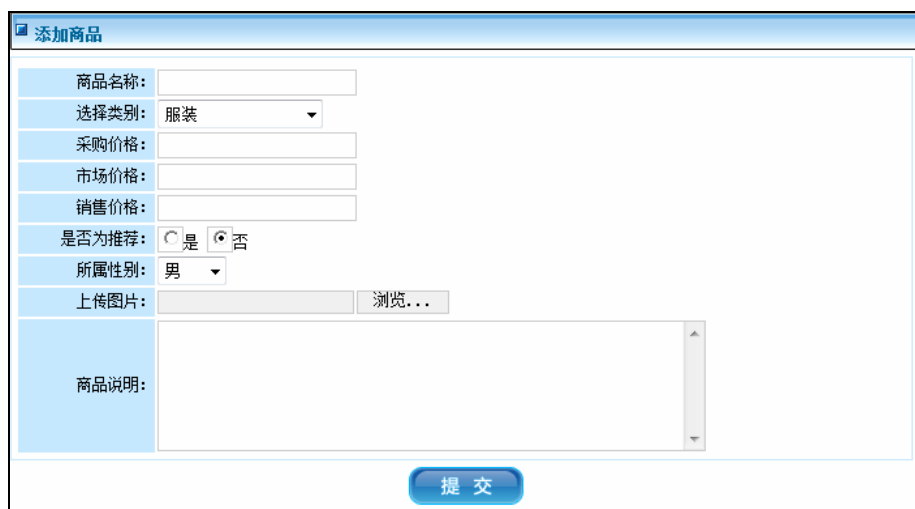


图 14.35 商品的添加页面

用户编辑完商品信息，单击页面中的“提交”超链接，该操作将会向系统发送一个 product_save.html 的 URL 请求，它与商品修改触发的是一个方法，都是 ProductAction 类中的 save() 方法。


在 Struts 2 的后台 Action 配置文件 struts-admin.xml 中，配置商品管理模块的 Action 以及视图映射关系，关键代码如下：

代码位置：光盘\mr\14\Shop\src\com\lyq\action\struts-admin.xml

```
<!-- 商品管理 -->
<package name="shop.admin.product" namespace="/admin/product" extends="shop.admin">
    <action name="product_*" method="{1}" class="productAction">
        <result name="list">/WEB-INF/pages/admin/product/product_list.jsp</result>
        <result name="input">/WEB-INF/pages/admin/product/product_add.jsp</result>
        <result name="edit">/WEB-INF/pages/admin/product/product_edit.jsp</result>
    </action>
</package>
```

```
<interceptor-ref name="adminDefaultStack"/>
</action>
</package>
```

14.10.4 商品类别管理功能的实现

 本模块使用的数据表: tb_productinfo 表

商品类别的维护中主要包括商品类别的查询、修改、删除以及添加。

☒ 商品类别查询

商品类别在后台中分为两种, 分别是商品类别树形下拉框的查询以及商品类别列表信息的查询, 商品类别树形下拉框的查询的实现较为复杂一些, 通过迭代的方式遍历所有的节点。

在后台的商品类别查询中, 通过树形下拉框的形式展现给用户, 如图 14.36 所示。

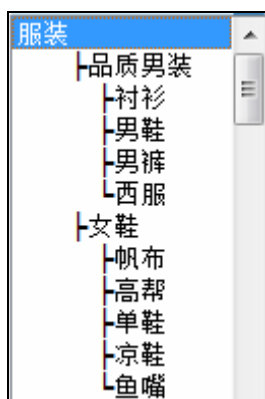


图 14.36 商品添加页面中的商品类别树形下拉框

在进入商品页面的 edit()方法中, 调用了 createCategoryTree()方法用来创建商品类别树, 其关键代码如下:

代码位置: 光盘\mr\14\Shop\src\com\lyq\action\product\ProductAction

```
private void createCategoryTree(){
String where = "where level=1"; //查询一级节点
PageModel<ProductCategory> pageModel = categoryDao.find(-1, -1,where ,null);//执行查询方法
List<ProductCategory> allCategorys = pageModel.getList();
map = new LinkedHashMap<Integer, String>(); //创建新的集合
for(ProductCategory category : allCategorys){ //遍历所有的一级节点
setNodeMap(map,category,false); //将其子节点添加到集合中
}
}
```

在 setNodeMap()方法中, 首先判断节点是否为空, 如果节点为空则停止遍历, 程序中根据获取的节点级别为类别名称添加字符串和空格, 用以生成渐进的树形结构, 将拼接后的节点放入 Map 集合中, 并获取其子节点重新调用 setNodeMap()方法, 直到遍历的节点为空为止, 其关键代码如下:

代码位置: 光盘\mr\14\Shop\src\com\lyq\action\product\ProductAction

```
private void setNodeMap(Map<Integer, String> map,ProductCategory node,boolean flag){
```

```

        if (node == null) { //如果节点为空
            return; //返回空，结束程序运行
        }
        int level = node.getLevel(); //获取节点级别
        StringBuffer sb = new StringBuffer(); //定义字符串对象
        if (level > 1) { //如果不是根节点
            for (int i = 0; i < level; i++) {
                sb.append(" "); //添加空格
            }
            sb.append(flag ? "└" : "┌"); //如果为末节点添加"┌"，反之添加"└"
        }
        map.put(node.getId(), sb.append(node.getName()).toString()); //将节点添加的集合中
        Set<ProductCategory> children = node.getChildren(); //获取其子节点
        // 包含子类别
        if (children != null && children.size() > 0) { //如果节点不为空
            int i = 0;
            // 遍历子类别
            for (ProductCategory child : children) {
                boolean b = true;
                if (i == children.size() - 1) { //如果子节点长度减1为i,说明为末节点
                    b = false; //设置布尔常量为false
                }
                setNodeMap(map, child, b); //重新调用该方法
            }
        }
    }
}

```

在商品添加页面中，通过<s:select>标签将商品类别树显示在下拉框中，其关键代码如下：

代码位置：光盘\mr\14\Shop\WEB-INF\pages\admin\product\product_add.jsp

```

<tr>
    <td width="119" height="22" bgcolor="#c6e8ff" align="right">选择类别: </td>
    <td><s:select list="map" name="category.id"></s:select></td>
</tr>

```

当用户单击后台管理页面左侧导航栏中的“查询所有类别”超链接时，会向系统发送一个 category_list.html 的 URL 请求，它将会触发 ProductCategoryAction 类中的 list() 方法，其关键代码如下：

代码位置：光盘\mr\14\Shop\src\com\lyq\action\product\ProductCategoryAction

```

public String list() throws Exception {
    Object[] params = null; //对象数组为空
    String where; //查询条件变量
    if (pid != null && pid > 0) { //如果有父节点
        where = "where parent.id=?"; //执行查询条件
        params = new Integer[]{pid}; //设置参数值
    } else {
        where = "where parent is null"; //查询根节点
    }
    pageModel = categoryDao.find(pageNo, pageSize, where, params); //执行封装的查询方法
    return LIST; //返回后台类别列表页面
}

```

该方法将返回后台的商品类别列表页面，如图 14.37 所示。

ID	类别名称	子类别	添加子类别	所属父类	编辑	删除
1	服装	有9个子类别	添加	无		
51	配饰	有6个子类别	添加	无		
83	家居	有8个子类别	添加	无		

首页 上一页 [1] 下一页 尾页

图 14.37 后台商品类别信息列表

☑ 商品类别添加

单击导航栏中“添加商品类别”或是商品类别列表页面中的“添加”超链接时，会进入到商品类别的添加页面，如图 14.38 所示。

添加商品类别

类别名称:

提交

图 14.38 商品类别添加页面

在类别名称中输入类别名称后，单击“提交”超链接，将会触发 `ProductCategoryAction` 类中的 `save()` 方法，在 `save()` 方法首先判断该节点的父节点参数是否存在，如果存在则先设置其父节点属性，然后再保存商品类别信息，其关键代码如下：

代码位置：光盘\mr\14\Shop\src\com\lyq\action\product\ProductCategoryAction

```

public String save() throws Exception{
    if(pid != null && pid > 0 ){           //如果有父节点
        category.setParent(categoryDao.load(pid)); //设置其父节点
    }
    categoryDao.saveOrUpdate(category);      //添加类别信息
    return list();                          //返回类别列表的查找方法
}

```

☑ 商品类别修改

当网站管理员单击商品类别列表中超链接时，将进入商品类别修改的页面，如图 14.39 所示。

编辑商品类别


类别名称:

提交

图 14.39 商品类别修改页面

修改商品类别信息完毕后，单击页面中的“提交”超链接，其触发的也是商品类别添加中 `ProductCategoryAction` 类的 `save()` 方法。

☑ 商品类别删除

当用户单击商品类别列表中的  图标，将执行商品类别信息的删除操作，该操作将会向系统发送一个 `category_del.html` 的 URL 请求，它将触发 `ProductCategoryAction` 类中的 `del()` 方法，该方法将以商品类别的 ID 为参数，执行持久化类中封装的 `delete()` 方法，删除指定的信息，其关键代码如下：

代码位置：光盘\mr\14\Shop\src\com\lyq\action\product\ProductCategoryAction

```
public String del() throws Exception {
    if(category.getId() != null && category.getId() > 0){           //判断是否获得ID参数
        categoryDao.delete(category.getId());                     //执行删除操作
    }
    return list();                                                 //返回商品类别列表的查找方法
}
```

在商品类别管理中添加、修改以及删除的操作实现都较为简单，商品类别信息的查询方法支持无限级的树形分级查询。

在 Struts 2 的后台 Action 配置文件 `struts-admin.xml` 中，配置商品类别管理模块的 Action 以及视图映射关系，关键代码如下：

代码位置：光盘\mr\14\Shop\src\com\lyq\action\struts-admin.xml

```
<!-- 类别管理 -->
<package name="shop.admin.category" namespace="/admin/product" extends="shop.admin">
    <action name="category_*" method="{1}" class="productCategoryAction">
        <result name="list">/WEB-INF/pages/admin/product/category_list.jsp</result>
        <result name="input">/WEB-INF/pages/admin/product/category_add.jsp</result>
        <result name="edit">/WEB-INF/pages/admin/product/category_edit.jsp</result>
        <interceptor-ref name="adminDefaultStack"/>
    </action>
</package>
```

14.11 后台订单管理模块的设计

网站管理员可以对会员的订单进行维护，但这种维护只能修改订单的状态，并不能修改订单中的任何信息，因为当用户确认订单时该订单就已经生效，它相当于用户与商家交易的一个契约，是用户与商家之间的一个交易凭证，所以不能进行任何的修改。

14.11.1 模块概述

在后台的订单管理模块中，主要分为两个基本模块，分别是 **订单的查询** 和 **订单状态的修改**，其中订单的查询又可分为订单的全部查询和用户自定义的条件查询，框架模块图如图 14.40 所示。

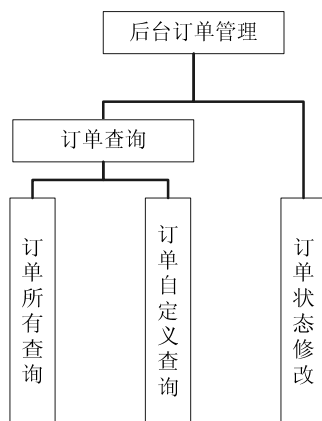


图 14.40 订单管理模块框架图

14.11.2 后台订单管理模块技术分析

按钮的触发事件和窗口的自动刷新

当用户在订单列表页面中，单击“更新订单状态”按钮时，将会弹出提示对话框，让用户选择修改的状态信息，在订单列表页面中通过模态窗体的形式弹出该对话框，为更新按钮绑定触发事件的关键代码如下：

代码位置：光盘\mr\14\Shop\WebContent\pages\admin\order\order_list.jsp

```
<td width="150" align="center">
<:url action="order_select" namespace="/admin/product" var="order_select">
    <s:param name="orderId" value="orderId"></s:param></s:url>
<input type="button" value="更新订单状态" onclick="openWindow('${order_select}',350,150);">
</td>
```

在弹出的子窗体中，如果是模态的，子窗体不关闭将无法进行主窗体的任何操作；如果是非模态的，子窗体不关闭同样可以进行主窗体的操作。

根据页面中的代码可知，Action 请求 order_select 跳转的页面为 order_select.jsp，也就是弹出的模态窗体

当用户在弹出的模块窗体中单击该页面，将会向系统发送一个 order_update.html 的 URL 请求，该请求触发的是 OrderAction 类中的 update() 方法，其关键代码如下：

代码位置：光盘\mr\14\Shop\src\com\lyq\action\order\OrderAction

```
public String update() throws Exception {
    OrderState orderState = order.getOrderState();           //获取设置的订单状态
    order = orderDao.load(order.getId());                     //装载订单对象
    order.setOrderState(orderState);                          //设置的订单状态
    orderDao.update(order);                                   //修改订单状态
    return "update";                                          //放回订单状态修改成功页面
}
```

当订单状态修改成功后，会弹出订单信息修改成功的提示对话框，通过 JavaScript 对该窗体进行设置，该窗体 3 秒后自动关闭，并刷新主页面。




在订单更新成功的页面中，设置窗体自动关闭的 JavaScript 关键代码如下：

代码位置：光盘\mr\14\Shop\WebContent\pages\admin\order\ order_update_success.jsp

```
<script type="text/javascript">
    function closewindow(){
        if(window.opener){
            window.opener.location.reload(true);    //刷新父窗体
            window.close();                        //关闭提示窗体
        }
    }
    function clock(){
        i = i -1;
        if(i > 0){                                //如果i大于0
            setTimeout("clock();",1000);          //1秒后重新调用clock()方法
        }else{
            closewindow();                        //调用关闭窗体方法
        }
    }
    var i = 3;                                    //设置i值
    clock();                                      //页面加载后自动调用clock()
</script>
```

在上述程序中通过量 i 来设置窗体自动的关闭时间，在 clock()方法中，当 i 的值为零时调用关闭窗体的方法，并且通过 setTimeout()方法设置方法调用时间，参数“1000”的单位为毫秒。

14.11.3 后台订单查询的实现

 本模块使用的数据表：tb_order 表

在后台订单的查询中，分为订单所有查询和订单的自定义条件查询，在订单的自定义查询中用户可以根据设定不同的查询条件查询订单的指定信息。在本应用程序中两者调用的都是同一个查询方法，在管理页面左侧导航栏中单击“查看订单”超链接时，将进入订单状态管理页面，如图 14.41 所示。

订单号	总金额	消费者	支付方式	创建时间	订单状态	修改
201005041012220323561	120.0	mrsoft	邮局汇款	2010年05月4日 10:12	已发货	<input type="button" value="更新订单状态"/>
201004271843190764180	240.0	mrsoft	邮局汇款	2010年04月27日 18:43	已发货	<input type="button" value="更新订单状态"/>
201004260941250469034	120.0	mrsoft	邮局汇款	2010年04月26日 09:41	已发货	<input type="button" value="更新订单状态"/>
首页 上一页 1 下一页 尾页						

图 14.41 订单状态管理页面

当用户单击左侧导航栏中的“订单查询”超链接时，将会进入订单查询条件的自定义页面，如图 14.42 所示。

图 14.42 订单查询条件的自定义页面

当用户单击导航栏中的“查看订单”超链接或单击订单查询条件的自定义页面中的“提交”超链接时，都会向系统发送一个 `order_list.html` 的 URL 请求，该请求触发的是 `OrderAction` 中的 `list()` 方法，其关键代码如下：

代码位置：光盘\mr\14\Shop\src\com\lyq\action\order\OrderAction

```
public String list() throws Exception {
    Map<String, String> orderby = new HashMap<String, String>(1);    //定义Map集合
    orderby.put("createTime", "desc");    //设置按创建时间倒序排列
    StringBuffer whereBuffer = new StringBuffer("");    //创建字符串对象
    List<Object> params = new ArrayList<Object>();
    if(order.getOrderId() != null && order.getOrderId().length() > 0){    //如果订单号不为空
        whereBuffer.append("orderId = ?");    //以订单号为查询条件
        params.add(order.getOrderId());    //设置参数
    }
    if(order.getOrderState() != null){    //如果订单状态不为空
        if(params.size() > 0) whereBuffer.append(" and ");    //增加查询条件
        whereBuffer.append("orderState = ?");    //设置订单状态为查询条件
        params.add(order.getOrderState());    //设置参数
    }
    if(order.getCustomer() != null && order.getCustomer().getUsername() != null
        && order.getCustomer().getUsername().length() > 0){    //如果会员名不为空
        if(params.size() > 0) whereBuffer.append(" and ");    //增加查询条件
        whereBuffer.append("customer.username = ?");    //设置会员名为查询条件
        params.add(order.getCustomer().getUsername());    //设置参数
    }
    if(order.getName() != null && order.getName().length() > 0){    //如果收款人姓名不为空
        if(params.size() > 0) whereBuffer.append(" and ");    //增加查询条件
        whereBuffer.append("name = ?");    //设置收款人姓名为查询条件
        params.add(order.getName());    //设置参数
    }
    //如果whereBuffer为空则查询条件为空，否则以whereBuffer为查询条件
    String where = whereBuffer.length() > 0 ? "where " + whereBuffer.toString() : "";
    pageModel = orderDao.find(where, params.toArray(), orderby, pageNo, pageSize);    //执行查询方法
    return LIST;    //返回后台订单列表
}
```

“查看订单”超链接并没有为 `list()` 方法传递任何的参数，所以最后传给 `find()` 方法的 `where` 查询条件字符串为空，`find()` 方法将会从数据库中查询所有的订单信息并**按创建时间的倒序输出**。

`list()` 方法将会返回后台的订单信息列表页面，在该页面利用 Struts 2 的 `<s:iterator>` 方法遍历输出返

回结果集中的信息即可。后台订单信息列表页面的关键代码如下：

代码位置：光盘\mr\14\Shop\WebContent\pages\admin\order\order_list.jsp

```
<table width="693" height="29" border="0" class="word01">
  <tr>
    <td width="140" align="center">订单号</td>
    <td width="60" align="center">总金额</td>
    <td width="63" align="center">消费者</td>
    <td width="70" align="center">支付方式</td>
    <td width="140" align="center">创建时间</td>
    <td width="70" align="center">订单状态</td>
    <td width="150" align="center">修改</td>
  </tr>
</table>
<table width="693" height="29" border="0">
  <s:iterator value="pageModel.list">
    <tr>
      <td width="140" align="center"><s:property value="orderId" /></td>
      <td width="60" align="center"><s:property value="totalPrice" /></td>
      <td width="63" align="center"><s:property value="customer.username" /></td>
      <td width="70" align="center"><s:property value="paymentWay.getName()" /></td>
      <td width="140" align="center">
        //遍历操作
        <s:date name="createTime" format="yyyy年MM月d日 HH:mm" /></td>
        <td width="70" align="center"><s:property value="orderState.getName()" /></td>
        <td width="150" align="center">
          <s:url action="order_select" namespace="/admin/product" ar="order_select">
            <s:param name="orderId" value="orderId"></s:param>
          </s:url> <input type="button" value="更新订单状态"
onclick="openWindow('${order_select}',350,150);"></td>
      </tr>
    </s:iterator>
  </table>
```

14.11.4 单元测试

在代码初期对后台订单自定义查询进行测试的时候发现，如果订单号为空，填写后面的查询条件进行查询的时候就会报错，如图 14.43 所示。

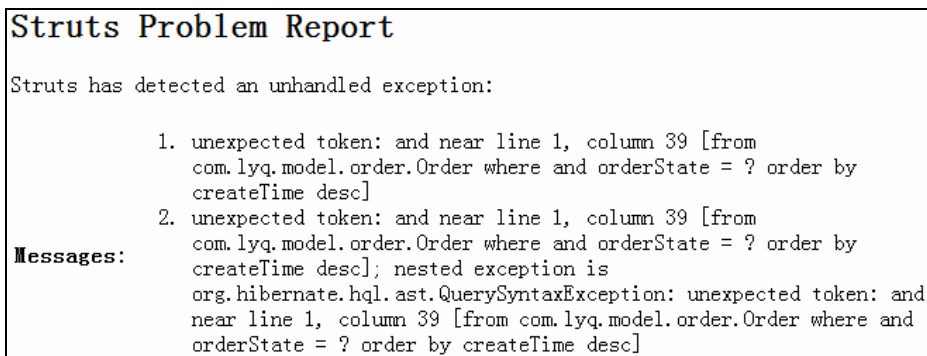


图 14.43 订单号为空查询时系统的报错信息

从错误信息中可以发现在 `where` 条件查询语句中关键字“`where`”直接连接的是关键字“`and`”，也就是说在订单状态前置查询条件订单号码为空的情况下同样将关键字“`and`”拼接到了查询条件中，解决该问题只需对查询条件的 `Object` 数组的长度进行判断，如果数组的长度不为 0，就将关键字“`and`”添加到查询条件中，以订单状态的查询条件为例，其关键代码如下：

代码位置：光盘\mr\14\Shop\src\com\lyq\action\order\OrderAction

```

if(order.getOrderState() != null){           //如果订单状态不为空
    if(params.size() > 0){
        whereBuffer.append(" and ");         //增加查询条件
    }
    .....                                   //省略的代码
}

```

14.12 开发技巧与难点分析

从网站的安全性考虑，用户在没有登录或是 `Session` 失效的时候是不允许进行购物和后台维护的，一般客户端的 `Session` 是有时间限制的（根据服务器中的配置决定，一般为 20 分钟），如果超出时间限制，系统就会报出现空指针的异常信息，出现这种情况的原因是系统从 `Session` 中取得的信息为空，即获取的用户登录信息为空，空值造成了这种情况的发生。

这个问题可以通过 `Struts 2` 的拦截器来解决，根据拦截器判断 `Session` 是否为空，并根据判断结果执行不同的操作。

拦截器（`Interceptor`）是 `Struts 2` 框架中一个非常重要的核心对象，它可以动态增强 `Action` 对象的功能，通过对登录拦截器的配置，如果会员的 `Session` 失效后，用户将无法使用购物车功能，除非重新登录；如果管理员 `Session` 失效后，将无法进入后台进行操作，没有直接登录的用户在地址栏中直接输入 `URL` 地址也将被拦截器拦截，并返回系统的登录页面，这样很大程度地提升了系统的安全性。

拦截器动态的作用于 `Action` 与 `Result` 之间，它可以动态的 `Action` 以及 `Result` 进行增强（在 `Action` 与 `Result` 加入新功能）。当客户端发送请求时，会被 `Struts 2` 的过滤器所拦截，此时 `Struts 2` 对请求持有控制权，`Struts 2` 会创建 `Action` 的代理对象，并通过一系列的拦截器对请求进行处理，最后再交给指定的 `Action` 进行处理。拦截器实现的核心思想是 `AOP`（`Aspect Oriented Programming`）面向切面编程。

首先创建会员登录拦截器 CustomerLoginInteceptor，其关键代码如下：

```
public String intercept(ActionInvocation invocation) throws Exception {
    ActionContext context = invocation.getInvocationContext();           // 获取ActionContext
    Map<String, Object> session = context.getSession();                 // 获取Map类型的session
    if(session.get("customer") != null){                                // 判断用户是否登录
        return invocation.invoke();                                       // 调用执行方法
    }
    return BaseAction.CUSTOMER_LOGIN;                                     // 返回登录
}
```

在前台的 Struts 2 的配置文件中配置该拦截器，其关键代码如下：

```
<package name="shop.front" extends="shop-default">
<!-- 配置拦截器 -->
<interceptors>
    <!-- 验证用户登录的拦截器 -->
    <interceptor name="loginInterceptor" class="com.lyq.action.interceptor.CustomerLoginInteceptor"/>
    <interceptor-stack name="customerDefaultStack">
        <interceptor-ref name="loginInterceptor"/>
        <interceptor-ref name="defaultStack"/>
    </interceptor-stack>
</interceptors>
<action name="index" class="indexAction">
    <result>/WEB-INF/pages/index.jsp</result>
</action>
</package>
```

通常情况下，拦截器对象实现的功能比较单一，它类似于 Action 对象的一个插件，为 Action 对象动态的织入新的功能。

系统后台拦截器的配置与前台类似，这里就不再进行详细的说明。

14.13 本章小结

本系统只是实现了电子商务网站一些基本的功能，真正的商务网站的开发难度和工作量要比本系统复杂和烦琐的多，但是希望通过本系统的开发，可以让读者了解网站的开发简单流程、SSH2 框架的整合以及 MVC 的设计模式，相信读者可以融会贯通。